THE CLASSIC WORK
NEWLY UPDATED AND REVISED

The Art of Computer Programming

VOLUME 1

Fundamental Algorithms
Third Edition

DONALD E. KNUTH

目　录

第1章　基本概念

1.1　算法 …… 1

1.2　数学准备 …… 7

1.2.1　数学归纳法 …… 8

1.2.2　数、幂和对数 …… 17

1.2.3　和与积 …… 31

1.2.4　整数函数和初等数论 …… 31

1.2.5　排列和阶乘 …… 37

1.2.6　二项式系数 …… 43

1.2.7　调和数 …… 61

1.2.8　斐波那契数 …… 65

1.2.9　生成函数 …… 71

1.2.10　一个算法的分析 …… 78

*1.2.11　渐近表示 …… 86

*1.2.11.1　O记号 …… 86

*1.2.11.2　欧拉求和公式 …… 89

*1.2.11.3　一些渐近计算 …… 92

1.3　MIX …… 99

1.3.1　MIX的描述 …… 99

1.3.2　MIX汇编语言 …… 114

1.3.3　对排列的应用 …… 133

1.4　某些基本的程序设计技术 …… 151

1.4.1　子程序 …… 151

1.4.2　共行程序 …… 157

1.4.3　解释性程序 …… 163

1.4.3.1　MIX模拟程序 …… 165

*1.4.3.2　跟踪程序 …… 174

1.4.4　输入和输出 …… 176

1.4.5　历史和参考文献 …… 188

第2章　信息结构

2.1　引论 …… 192

2.2　线性表 …… 196

2.2.1　堆栈，排队和双排队 …… 196

2.2.2　顺序分配 …… 201

2.2.3　链接分配 …… 210

2.2.4　循环表 …… 227

2.2.5　双重链接表 …… 233

2.2.6　数组和正交表 …… 251

2.3　树 …… 259

2.3.1　遍历二叉树 …… 267

2.3.2　树的二叉树表示 …… 280

2.3.3　树的其它表示 …… 293

2.3.4　树的基本数学性质 …… 305

2.3.4.1　自由树 …… 305

*2.3.4.2　有向树 …… 313

*2.3.4.3　"无限性引理" …… 321

*2.3.4.4　树的枚举 …… 325

2.3.4.5　通路长度 …… 336

*2.3.4.6　历史和参考文献 …… 341

2.3.5　列表和废料收集 …… 343

2.4　多重链接结构 …… 356

2.5　动态存储分配 …… 365

2.6　历史和参考文献 …… 383

习题答案 …… 390

附录A　记号索引 …… 533

附录B　数值数量表格 …… 537

1. 基本常数(10进的)

2. 基本常数(8进的)

3. 调和数，贝努里数，斐波那契数

名词和姓名中英对照表 …… 541

第1章 基本概念

1.1 算 法

对于所有的计算机程序设计说来，算法的概念总是基本的，所以我们应当首先细心地分析这一概念。

“算法”（Algorithm）一词本身就是十分有趣的。初看起来，这词好像是某人打算要写的“对数”（Logarithm）一词但却把头四个字母写得前后颠倒了。这个词直到1957年之前在《韦氏新世界词典》（Webster's New World Dictionary）中还未出现，我们只能找到带有它的古代含意的较老形式的“算术”（Algorism），指的是用阿拉伯数字进行算术运算的过程。在中世纪时，珠算家用算盘进行计算，而算术家用算术进行计算。中世纪之后，对这个词的起源已经拿不准了，早期的语言学家试图推断它的来历，认为它是从把algiros〔费力的〕+arithmos〔数字〕组合起来派生而成的，但另一些人则不同意这种说法，认为这个词是从“喀斯迪耳国王Algor”派生而来的。最后，数学史学者发现了算术“algorism”一词的真实起源：它来源于有名的《波斯教科书》（Persian textbook）的作者的名字阿布·贾法·穆哈默德·伊本·穆萨·阿科瓦里茨米（Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî）（约公元825年）——从字面上看，是“Ja,far的父亲，Mohammed,Moses的儿子，Khowârizm的土人”。Khowârizm是今天苏联基发（ХИВА）市的小城镇。Al-Khowârizm写了著名的书《复原和化简的规则》（Kitab al jabr w'al-muqabala）；另一个词，“algebra”（代数），是从他的书的标题引出来的，尽管这本书实际上根本不是讲代数的。

逐渐地，“algorism”的形式和意义就变得面目全非了。如牛津英语词典所说明的，这个词是由于同arithmetic(算术)相混淆而形成的错误的造作。由algorism又变成algorithm。只要了解人们已经忘记了这个词原来派生的实况，就不难理解这种变化。一本早期的德文数学词典《大全数学辞典》（Vollständiges Mathematisches Lexicon莱比锡，1747），给出了Algorithmus（算法）一词的如下的定义：“在这个名称之下，组合了四种类型的算术计算的概念，即是，加法，减法，乘法和除法”。拉丁短语algorithmus infinitesimalis（无限小算法），在当时就用来表示“莱布尼兹（Leibnitz）所发明的以无限小量进行计算的方法”。

1950年左右，algorithm一词经常地同欧几里得算法“Euclid's algorithm”联系在一起。这算法就是在欧几里得的《几何原本》（Euclid's Elements，第VII卷，命题i和ii）中所阐述的求两个数的最大公因子的过程。在这里给出欧几里得算法来，也将是有益的：

算法E（欧几里得算法）。给定两个正整数m和n，求它们的最大公因子，即能够同时整除m和n的最大的正整数。

E1. 〔求余数〕以n除m并令r为所得余数（我们将有$0 \leqslant r < n$）。

E2. 〔余数为0？〕若$r = 0$，算法结束；n即为答案。

E 3.〔互换〕 置$m \leftarrow n$， $n \leftarrow r$，并返回步骤 E1. ▌

当然，欧几里得并非就是以这样的方式来给出他的算法的。上面的格式充分演示了在给出本书中所有的算法时贯穿全书的风格。

我们对所讨论的每一个算法，都给出了一个标识的字母（例如上边算法中的E），并且用这个字母后面接上数字（例如E 1，E 2 等等）来标识这个算法的步骤。书中各章分为编了号的节，在一节之内的算法仅用字母来标记；但当在其它节中引用这些算法时，则附以相应的节号。例如，我们现在是在第 1.1 节；在这一节中，欧几里得算法叫做算法 E，而在后边的节引用它时，就记作算法 1.1 E。

一个算法的每一步（例如上边的步骤 E 1）以一个方括号中的一个短句开始，它尽可能简短地概括这一步的主要内容。这一短句通常也出现在一个与这个算法相对应的框图（例如图 1）中。借助于框图，读者将有可能更直观地看出这个算法的流程。

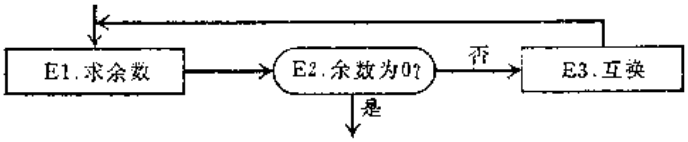


图 1　算法 E 的框图

在概括性的短句之后，是有待执行的某个动作或有待作出的某个判断的文字符号的叙述。偶而也有带圆括号的注释（例如，步骤 E 1 的第二句），它是关于这一步的说明性信息，通常指出变量的某些特征或这一步的当前目标，等等；带圆括号的注释并不影响属于这一算法的动作，而只是为了便于帮助读者理解。

步骤 E 3 中的箭头←是最重要的替代运算（有时叫做赋值或代换）。“$m \leftarrow n$”意思是变量 m 的值代之以变量 n 的当前值。当算法 E 开始时，m 和 n 的值是原来给出的数；但当它结束时，一般说来，这些变量将有不同的值。箭号用来把替代运算同相等关系加以区别：我们将不说“置 $m = n$”，但也许我们将要问“是否 $m = n$？”“=”号标记一个可被测试的条件，“←”号标记一个可被执行的动作。n 增 1 的运算通过“$n \leftarrow n + 1$”来标记（读作以 $n + 1$ 代替 n）。一般说来，“变量←式子”意味着用出现于式子内的所有变量的当前值来计算式子，并将结果代替箭头左边的变量的原先值。不熟悉计算机工作的人们有时有这样的倾向，即用“$n \rightarrow n + 1$”来标记 n 增 1 的运算，说是“n 变成 $n + 1$”，这只能导致混乱，因为它同标准的约定正相冲突，因此我们应加以避免。

注意，步骤 E 3 中的动作的次序是重要的。“置 $m \leftarrow n$， $n \leftarrow r$”完全不同于“置 $n \leftarrow r$， $m \leftarrow n$”。因为后者意味着，在用 n 的值来置 m 之前，n 的原先值已经失去了。因此，后一次序等价于“置 $n \leftarrow r$， $m \leftarrow r$”。当若干个变量全都置成等于相同的量时，我们使用多重的箭头；于是，“$n \leftarrow r$， $m \leftarrow r$”可以写成“$n \leftarrow m \leftarrow r$”。两个变量的值相互交换，我们可以写成“交换 $m \longleftrightarrow n$”；这一动作也可通过使用一个新变量 t，并写“置 $t \leftarrow m$， $m \leftarrow n$， $n \leftarrow t$”来确定。

除非另有说明，一个算法总是在编号最小的步骤处，通常是在步骤1处开始，并且顺序逐步执行之。在步骤E3中，强制的“返回步骤E1”以一种明显的方式说明了进行计算的次序。在步骤E2中，以条件“若$r=0$”作为动作进行的前提条件；所以，如果$r\neq 0$，就不应用该句子的剩下的话。虽然这里并未说明动作，但我们总是理解成：可以加上不言而喻的句子“如果$r\neq 0$，进行步骤E3”。

粗竖线▌，出现于步骤E3的末尾，用来表示算法已结束而正文将继续。

至此，我们实际上讨论了用于本书的算法中的所有记号的约定，剩下只需要来约定一个用来表示一个有序数组的元素的“带下标的”或“带足标的”项的记号了。假设我们有n个量$v_1, v_2, \cdots, v_n$；对于第j个元素v_j，我们经常使用记号$v[j]$来代替。类似地，对于双下标的记号如a_{ij}，有时更愿意使用$a[i, j]$。有时，变量用多个字母的名称，并把这些字母都写成大写的。例如，TEMP可以是一个用于暂时地保存一个计算的值的变量的名称，PRIME[K]可以表示第K个质数，等等。

关于算法的形式就谈这么多；现在让我们来实现一个算法看一看。应当立即指出，读者不要期望象读一篇小说那样来读一个算法。如果是抱着这样的企图的话，那他就会很难理解文中正在叙述的内容。一个算法应该是可以信赖的，而且学习一个算法直到彻底掌握的最好方法是反复进行试验。当读者遇到一个算法时，应立即拿起笔和纸，找出这个算法的一个例子；通常要给出这个例子的要点，要不然就是读者能容易地把它想象出来。为了能够理解一个给定的算法，这是一个简单且省力的方法，经验证明，其它的途径都是不成功的。

因此，让我们找出算法E的一个例子。假设我们给出$m=119$和$n=544$。让我们准备好由步骤E1开始（读者现在就要象我们在下边一步一步叙述的那样来遵循这个算法）。在这种情况下，以n除m是十分简单的。简直是太简单了，因为商是0，而余数是119。于是$r\leftarrow 119$。我们进行到步骤E2，而且由于$r\neq 0$，步骤E2不发生作用。在步骤E3，我们置$m\leftarrow 544$，$n\leftarrow 119$。显然，如果原来$m<n$，步骤E1中的商数总为0，而算法总是首先以这样比较麻烦的方式来相互交换m和n。如果愿意，我们可以增加一个新的步骤：

“E0[保证$m\geqslant n$]如果$m<n$，交换$m\longleftrightarrow n$。”这样一来，在算法中并无实质性的改动，只是增加了算法的长度，但却大约有一半的情况减少了用来实现算法所需的时间。

返回到步骤E1，我们发现$\frac{544}{119}=4\ \frac{68}{119}$。所以$r\leftarrow 68$。步骤E2再一次不能用。而且在步骤E3，我们置$m\leftarrow 119$，$n\leftarrow 68$。下一轮置$r\leftarrow 51$，而且最终$m\leftarrow 68$，$n\leftarrow 51$。然后，$r\leftarrow 17$，且$m\leftarrow 51$，$n\leftarrow 17$。最后，当以17除51时，$r\leftarrow 0$，于是算法在步骤E2处结束。119和544的最大公因子是17。

所以一个算法就是这样。算法的现代的意义十分类似于处方、过程、方法、技术、规程、程序，只不过是算法一词的含意有时稍微有一点特殊。一个算法，就是一个有穷规则的集合，其中之规则规定了一个解决某一特定类型的问题的运算序列；此外，一个算法还有五个重要的特性：

1）有穷性 一个算法必须总是在执行有穷步之后结束。算法E是满足这个条件的。因为在步骤E1之后，r的值小于n，所以如果$r\neq 0$，下一次进行步骤E1时，n的值

已经减小。正整数的递降序列必然最后要终止。所以，对于任何给定的 n 的原始值，步骤E1仅仅执行有穷次。然而，注意，步骤的数目可以变成任意大；选择巨大的 m 和 n，就可能使步骤E1执行百万次以上。

（具有算法的其它所有特征，但却可能不具有有穷性的步骤，可以叫做“计算方法”。除了给出求两个整数的最大公因子的算法外，欧几里得还给出了实际上等价于算法E的一个几何构造，不过它是求两个线段的长度的“最大公度”的步骤。这是一个计算方法，因为如果所给的两个长度是“无公度的”，那它就不终止）。

2）确定性 算法的每一个步骤，必须是确切地定义的。对于每种情况，有待执行的动作必须严格地和不含混地规定之。本书中的算法都应当符合这一准则。但是由于它们是以自然语言描述的，读者有可能未能精确地理解作者的意思是什么。为了避免这一困难，就专为描述算法设计了形式地定义的程序设计语言或计算机语言，其中每一个语句有着非常确切的意义。本书中的许多算法，将同时以自然语言和以一种计算机语言给出。当用一个计算机语言来描述一个计算方法时，其表达形式就叫做一个程序。

在算法E中，确定性准则意味着：例如应用于步骤E1，则假定读者确切地知道 n 除 m 是什么意思而余数又是什么。事实上，如果 m 和 n 不是正整数，那末这究竟是什么意思，已经没有一般的约定。$-\pi$ 除 -8 的余数是什么？0除59/13的余数是什么？因此，确定性准则意味着我们必须保证，当执行步骤E1时，m 和 n 的值总是正整数。按假设，这在开始时是正确的。而在步骤E1之后，r 是一个非负整数。如果要进行步骤E3，r 还必须是非0的，所以 m 和 n 事实上总象所要求的那样，都是正整数。

3）输入 一个算法有0个或多个的输入。它即是，在算法开始之前，对算法最初给出的量。这些输入取自于特定的对象集合。例如，在算法E中，有两个输入，即 m 和 n，它们都取自于正整数集合。

4）输出 一个算法有一个或多个的输出。它即是，同输入有某个特定关系的量。算法E有一个输出，即是步骤E2中的 n，它是两个输入的最大公因子。

（此数确实是最大公因子，我们可以很容易证明如下。在步骤E1后，我们有

$$m = qn + r$$

其中 q 是某个整数。若 $r = 0$，则 m 是 n 的一个倍数，且显然在这种情况下，n 就是 m 和 n 的最大公因子。若 $r \neq 0$，注意同时除尽 m 和 n 两数的任何数必定也除尽 $m - qn = r$，而且同时除尽 n 和 r 两数的任何数必定也除尽 $qn + r = m$；所以 m 和 n 的公因子的集合必定与 n 和 r 的公因子的集合相同，而且，特别地，m 和 n 的最大公因子与 n 和 r 的最大公因子相同。因此步骤E3并不改变原来问题的答案）。

5）能行性 一般说来，还期望一个算法是能行的。这意味着算法中所有有待实现的运算必须都是相当基本的，即是说，它们原则上都是能够精确地进行的，而且人们用笔和纸做有穷次即可完成。算法E仅仅使用一个正整数除以另一个正整数的除法，测试一个整数是否为0，并置一个变量的值等于另一个变量的值这样一些运算。这些运算都是能行的，因为整数可以用有穷的方式在纸上表示之，而且至少有一个方法（“除法算法”）来进行一个整数除以另一个整数的运算。但若两数之值是由无穷的十进展开所确定的任何实数，则同一运算就不是能行的。又若两数之值是不能精确地加以确定的物理线段的长度，则同一

运算也就不是能行的。非能行的步骤的另一个例子是，“如果 2 是使方程 $x^n+y^n=z^n$ 有正整数解 x，y，z 的 n 当中最大的整数，则进行步骤 E4”。这样一个语句不是一个能行的运算，除非有人能够说明有一个算法来确定 2 是否具有所说的性质的最大整数。

现在让我们把算法的概念同烹调方法的概念作一类比：烹调方法大概总有有穷性要求（虽然说是“心急水不沸”），输入（鸡蛋，面粉，等等）和输出（盒装便餐，等等）。但如所周知，却缺乏确定性。经常有缺乏确定性的情况。例如，“加少许盐”。“少许”定义作“少于 1/8 茶匙”；“盐”也许已经足够明确地定义了；但盐该加到哪儿（在顶上，边上，等等）？也许象“轻轻地颠簸直到混合物发脆为止”，“在小长柄锅里热白兰地酒”等等指示，作为对素有训练的厨师说来，说明是十分足够了，但是一个算法必须确定到这样一个程度，即使得一台计算机也能遵循这个指示。此外，通过研究一本好的菜谱，一个计算机程序员也能学到许多东西（事实上，作者好容易才没有把本书的书名叫做“程序员的烹调技巧”。也许某一天，作者将试图写一本叫做“厨房的算法”的书）。

我们应当说明，有穷性的限制对于实际的使用说来，实在是不够强的。一个有用的算法应该不仅要求有穷的步骤，而且应该是很有限的步骤，一个合理的数目。例如，有一个算法，它确定棋赛是否总由白子强行获胜（见习题2.2.3-28）。这个算法所解决的问题，是一个为成千上万的人们强烈感兴趣的问题。然而这仍然是一个安全的比赛。因为，由于执行这个算法需要非常大量的——尽管是“有穷的”——时间，以至我们在自己的一生中决不能知道这个问题的答案。某些有穷数，它们是如此之大，实际上已是不可理解的了。关于这个问题的讨论，请参看第 8 章。

实际上，我们不仅需要算法，而且还要求有好的算法。但对于好——美好，没有严格的定义。所谓好的一个准则是执行算法所需要的时间的长短；这可以借助于执行每一步的执行次数来表达。另一些准则是算法对于计算机的适应性，它的简单性以及精致性，等等。

有时候，我们对于同一个问题可以有若干个算法，因而我们必须判定哪一个最好。这导致我们进到了一个极端有趣的和最为重要的算法分析领域：给定一个算法，问题是确定它的性能特征。

例如，我们可以从这一观点来考察欧几里得算法。假设我们问这样一个问题：“假定 n 的值已知，但 m 允许取所有正整数的值，试问执行算法 E 的步骤 E1 的平均次数 T_n 是多少？”首先，我们需要校验一下，看这个问题是否能有一个有意义的回答（因为我们正在试图对无限多个 m 的选择来取平均数）。但显然，在第一次执行了步骤 E1 后，有关的只不过是 m 除以 n 后所得的余数。所以，问题归结成为，我们所要求的平均数 T_n 就是要对 $m=1$，$m=2$，…，$m=n$ 来试验算法，计算出执行步骤 E1 的总次数，然后除以 n。

现在，重要的问题是确定 T_n 的性质。它是否近似于 $\frac{1}{3}n$，$\sqrt{n}$ 等等？事实上，对于这个问题的回答是极为困难的和令人困惑的数学问题，还没有完全解决。在 4.5.3 节中我们将更细地加以讨论。对于 n 的充分大的值，有可能来证明，T_n 近似于 $(12\ln 2/\pi^2)\ln n$，即是说，同 n 的自然对数成比例，并且带有一个不可能轻易推测到的比例常数。关于欧几里得算法，以及其它计算最大公因子的办法的更详细的讨论，请参看 4.5 节。

作者希望用“算法分析”这个名称来描述诸如此类的研究。一般的思想是取出一个具体的算法，并且来确定它的通常的行为；偶尔地，我们也研究一个算法是否在某种意义下

为“最优的”。至于“算法论”，则完全是另一个课题，主要是讨论计算具体数量的能行算法之存在性或不存在性；这样一套理论在这部书中没有很深入地进行研究，只是在第11章中简单地作了讨论。

到目前为止，我们对于算法的讨论是较为不精确的；而且一个面向数学的读者有理由认为，以前的解释所打下的基础对于建立关于算法的任何理论来说，是非常不牢靠的。因此，在结束这一节的讨论之前，我们要简单地提示一个方法，通过它，可以借助于数学的集合论，坚实地建立算法的概念。让我们形式地定义，一个计算方法是一个四字式（Q，I，Ω，f），其中Q是一个包含子集I和Ω的集合，f是由Q到它自身的一个函数。其次，f应当保留Ω的每个元素不动，即是说，对于Ω的所有元素q，$f(q)$应当等于q。这四个量Q，I，Ω，f分别用来表示计算的状态，输入，输出和计算的规则。集合I中的每个输入x定义一个计算序列x_0，x_1，x_2，…如下：

$$x_0=x \quad 且 \quad x_{k+1}=f(x_k) \quad 对于 \quad k\geqslant 0 \tag{1}$$

计算序列说是在第k步终止，如果k是使x_k在Ω中的最小整数，而且在这种情况下说是由x产生输出x_k（注意，如果x_k在Ω中，则x_{k+1}也是，因为在这样的情况下，$x_{k+1}=x_k$）。某些计算序列可能永不终止。一个算法是对于I中的所有x，都在有穷多步内终止的一个计算方法。

例如，算法E可以以这些术语形式化如下：设Q是所有单个的数（n），所有有序偶（m、n）和所有有序的四字式（m、n，r，1），（m，n，r，2）以及（m，n，p，3）的集合，其中m，n和p是正整数，而r是一个非负整数。设I是所有有序偶（m，n）做成的子集，并设Ω是所有单个的数（n）做成的子集。设f定义如下：

$$\begin{aligned}
&f(m,\ n)=(m,n,0,1);\quad f(n)=(n);\\
&f(m,n,r,1)=(m,n,n\text{除}m\text{的余数},2);\\
&f(m,n,r,2)=(n)\text{ 如果 }r=0,(m,n,r,3)\text{否则};\\
&f(m,n,p,3)=(n,p,p,1)。
\end{aligned} \tag{2}$$

这个记号和算法E之间的对应关系是显然的。

“算法”这一概念的上述形式化并不包括早先提出的“能行性”的限制。例如，Q可以表示不能用笔和纸的方法来计算的无穷序列，或者f可以包括人类通常不能施行的运算。如果我们希望限制算法的概念，使其仅仅包括初等的运算，那末就可以对Q，I，Ω和f加以种种限制。例如说：设A是字母的有穷集合，并设A^*是A上所有的串的集合（即是，所有的有序序列$x_1x_2\cdots x_n$的集合，其中$n\geqslant 0$，且对于$1\leqslant j\leqslant n$，x_j在A中）。这一思想是把计算的状态编码以便通过A^*中的串来表示它们。现在假设N是一个非负整数，且设Q是所有（σ，j）的集合，其中σ在A^*中而j是一个整数，$0\leqslant j\leqslant N$；设I是带有$j=0$的Q的子集，且设Ω是带有$j=N$的子集。如果θ和σ是A^*中的串，我们说θ在σ中出现，如果σ有$\alpha\theta\omega$的形式，其中α和ω是串。为了完成我们的定义，再设f是由串θ_j，ϕ_j和整数a_j，b_j所定义的下列类型的一个函数（其中$0\leqslant j<N$）：

$$\begin{aligned}
&f(\sigma,j)=(\sigma,a_j)\qquad \text{如果}\theta_j\text{不在}\sigma\text{中出现};\\
&f(\sigma,j)=(\alpha\phi_j\omega,b_j)\quad \text{如果}\alpha\text{是使}\sigma=\alpha\theta_j\omega\text{的最短可能的串};\\
&f(\sigma,N)=(\sigma,N)。
\end{aligned} \tag{3}$$

这样一个计算方法显然是“能行的”．而且经验证明，对于做我们用手所能做的事情来说，它也足够强有力了。此外，还有许多其它的实质上都是等价的方法，来阐述“能行的”计算方法这一概念（例如，用图林机器）。上边的阐述实际上和安·安·马尔科夫在他所写的书《算法论》（Теория Алгорифмов, А. А. Марков ИАН СССР, 1954；有中译本，胡世华，唐稚松，何成武译，科学出版社，1959）中给出的是一样的。

习题

1．〔10〕 正文中说明了怎样用替代的记号来互换变量 m 和 n 的值，即：通过置 $t \leftarrow m$，$m \leftarrow n$，$n \leftarrow t$。试说明四个变量的值（a，b，c，d）怎样通过一串替代重排成（b，c，d，a）。换句话说，a 的新值等于 b 的原先值，余此类推。试用最少的替代来实现之。

2．〔15〕 证明在步骤 E1 开始时，m 总大于 n，除了头一次这一步可能出现相反的情况外。

3．〔20〕 改变算法 E（为了提高性能），使得在步骤 E3 中，我们不去互换数值而立即用 r 除 n，且命 m 是余数。加上新的适当步骤以便避免所有平凡的替代运算。以算法 E 的格式来写这个算法，并称之为算法 F。

4．〔16〕 2166 和 6099 的最大公因子是什么？

▶5．〔12〕 试说明在本书前言部分中给出的“阅读本书的步骤”不是真正的算法，它对于我们所提出的算法五项准则，竟有三项不符合！此外，并请指出它与算法 E 的格式之间的某些差别。

6．〔20〕 按照本节接近于结束处所使用的记号，T_5 是什么？

▶7．〔*M*21〕 假设 m 是已知的，而 n 允许取任意正整数。设 U_m 是在算法 E 中执行步骤 E1 的平均次数。试证 U_m 是有确定定义的，并问 U_m 是否以某种方式与 T_m 有关？

8．〔*M*25〕 通过确定类似于方程（3）中的 0_j，ϕ_j，a_j，b_j，试给出计算正整数 m 和 n 的最大公因子的一个“能行”的形式算法。设输入表示成串 $a^m b^n$，即是，m 个 a 后面跟着 n 个 b。请尽可能简单地给出你的解决办法来〔提示：用算法 E，但不用步骤 E1 中的除法，而置 $r \leftarrow |m-n|$，$n \leftarrow \min(m, n)$〕。

▶9．〔*M*30〕 假设 $C_1=(Q_1, I_1, \Omega_1, f_1)$ 和 $C_2=(Q_2, I_2, \Omega_2, f_2)$ 是计算方法。例如，C_1 可以代表类似于方程（2）那样的算法 E，除了 m 和 n 在数量上应有所限制外；而 C_2 可以代表算法 E 的一个计算机程序的实现（Q_2 可以是机器的所有状态的集合，即是，它的存储器和寄存器的所有可能的状态的集合；f_2 可以是机器的各个动作的规定；而 I_2 可以包括：用来确定最大公因子的程序的初始状态，以及 m 和 n 的值）。

请对于“C_2 是 C_1 的一个表示”的概念，给出一个集合论的形式定义：这直观上意味着 C_1 的任何计算序列均由 C_2 所“模写”。但是 C_2 可以采取更多的步骤以进行计算，而且它可以在它的状态中保留更多的信息（因此得到语句“程序 X 是算法 Y 的一个实现”的一个严格的解释）。

1.2 数学准备

在这一节中，我们将讨论贯穿在本章的剩下部分所使用的数学记号，而且将推导在这部书中一再使用的若干基本公式。对于不想涉及更复杂的数学推导的读者，至少应当熟悉

各种公式的意义，以便能够使用这些推导的结果。

在这部书中，使用数学记号有两个主要的目的：(1)用来描述一个算法的诸部分；(2)用来分析一个算法的性能特征。用于描述算法的记号，就象在前一节所说明的那样，是十分简单的。但当分析算法的性能时，将使用其它更专门的记号。

这部书中的大多数算法，都伴随有确定该算法的预期之运行速度的数学计算。这些计算几乎引用到每一个数学分支，而且需要写出一部专门的书来研讨在这里或那里所使用的全部数学概念。然而，大多数计算用高等代数的知识就能进行，而且具有初等微积分知识的读者就能理解几乎所有的遇到的数学。在个别地方，需要使用复变函数论、群论、数论、概率论等等较深入的结果，尔后或者以初等的方式来阐明这一题目，或者给出其它参考文献以供参考。

算法分析中涉及的数学技术通常都有一种特殊的风味：将经常地发现，我们自己总是在对有理数进行有限求和，或者对递推关系进行求解。这样一些题目在数学课程中传统地只是很“轻快地”处理一下就过去了，因此以下的小节用来“深入地”说明用于这类问题的计算和技术的类型，同时给出对所定义之记号的用法的全面练习。

重要的说明。尽管以下几个小节提供了为研究计算机算法所需要的数学技巧方面的较为广泛的训练，但大多数读者一开始不会看出这些材料同计算机程序设计之间有任何紧密的联系（除了在 1.2.1 小节外）。读者可以一面确信作者的断言，即是这里所讨论的题目确实都是非常有关系的，一面细心地阅读以下的几小节；或者他可以首先轻快地掠过这一节，而后（在阅读了这里所述的技术在后面几章的许多应用之后）再回过头来阅读这一节以便进行更透彻的研究。第二种办法可能更好些，因为这样做读者会感到学习起来目的性明确，仿佛有一股强烈的吸引力吸引他继续阅读；而如果在第一次阅读本书时花太多的时间来阅读这些材料，人们可能会觉得自己根本不是在钻研计算机程序设计的题目！然而，每个读者至少应当使自己熟悉这些小节的一般内容，并且应当动手至少做一点练习，即使是头一次阅读也应如此。1.2.10小节应予以特别的注意，因为它是大多数后边展开的理论材料的起点。1.3 节突然离开了“纯粹数学”的领域，而进入到“纯粹的计算机程序设计”中去。

1.2.1 数学归纳法

设$P(n)$是关于整数n的某个命题。例如，$P(n)$可以是“n乘$(n+3)$是一个偶数”，或者“如果$n \geqslant 10$，则$2^n > n^3$”。假设我们要来证明$P(n)$对所有的正整数n成立。证明此式成立的一个重要的方法是：

a）给出一个关于$P(1)$为真的证明；

b）给出一个关于“如果$P(1)$，$P(2)$，…，$P(n)$都为真，则$P(n+1)$也为真”的证明，这个证明必须对任何正整数n都成立。

作为一个例子，我们考虑以下的一系列等式——自古以来，已经有许多人独立地发现了它们：

$$1=1^2,\ 1+3=2^2,\ 1+3+5=3^2,\ 1+3+5+7=4^2,\ 1+3+5+7+9=5^2 \quad (1)$$

我们可以把这个一般的性质写成如下的公式：

$$1+3+\cdots+(2n-1)=n^2 \qquad (2)$$

暂且，让我们把这个等式叫做$P(n)$。我们希望来证明$P(n)$对所有正数n为真。遵照上述步骤，我们有

a）“$P(1)$为真，因为$1=1^2$。”

b）“如果$P(1)$，$P(2)$，…$P(n)$都为真，则$P(n)$为真，所以等式（2）成立。等式两边加上$2n+1$，我们得到

$$1+3+\cdots+(2n-1)+(2n+1)=n^2+2n+1=(n+1)^2$$

由此证明了$P(n+1)$亦为真。”

我们可以把这个方法看成是一个“算法证明的步骤”。事实上，假定上面的步骤（a）和（b）已经完成，以下的算法就产生出$P(n)$对于任何正整数n的证明：

算法I（构造一个证明）。给定一正整数n，这个算法将输出$P(n)$为真的一个证明。

I1.〔证明$P(1)$〕置$k\leftarrow 1$，而且根据（a），输出一个$P(1)$的证明。

I2.〔$k=n$?〕如果$k=n$，终止此算法。所需的证明已经输出。

I3.〔证明$P(k+1)$〕根据（b），输出一个关于“如果$P(1)$，…，$P(k)$都为真，则$P(k+1)$为真”的证明。而且输出“我们已经证明了$P(1)$，…，$P(k)$；因此$P(k+1)$为真”。

I4.〔k增1〕k加1并转到步骤I2。■

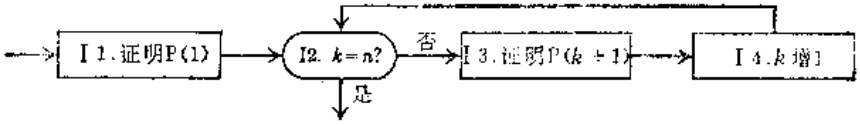


图2　算法I：数学归纳法

由于这一算法显然地给出了$P(n)$对于任何给定的n的一个证明，因此我们知道，上述的证明技术（a），（b）在逻辑上是正确的。这个证明方法叫做用数学归纳法证明。

应当把“数学归纳法”的概念同科学上通常所说的“归纳推理”区别开来。一位科学家进行特定的观察并且通过“归纳”，从而建立起说明这些事实的一般理论或假设；例如，他可以观察上述（1）中的五个关系式并且给出公式（2）。在这个意义下，“归纳”充其量只不过是某人关于这个情况的猜测。在数学中，我们将把这说成是一个经验结果或一个猜测。

看看另一个例子也是有益处的。设$p(n)$表示“n的分划”的数目，即是说，不管其次序，把n写成若干个正整数的和的不同方法的数目。因为

$$5=1+1+1+1+1=2+1+1+1=2+2+1$$
$$=3+1+1=3+2=4+1=5$$

所以我们有$p(5)=7$。事实上，前头几个值是容易确定的，

$$p(1)=1,\quad p(2)=2,\quad p(3)=3,\quad p(4)=5,\quad p(5)=7$$

至此，可以通过“归纳”，试行提出这样的假设，即序列$p(n)$跑遍所有质数。为检验这个假设，我们来计算$p(6)$而且刚巧$p(6)=11$，和我们的推测相符。

〔不幸，$p(7)$结果却是15，坏了事了，因此必须再试其它的假设。这个问题已被公

认是十分困难的，尽管斯·拉马纽燕（S. Ramanujan）关于 $p(n)$ 这数成功地猜测到并证明了许多值得注意的事情。欲了解进一步的信息，请参看戈·哈·哈迪（G. H. Hardy）所著的《拉马纽燕》(Ramanujan)一书（伦敦，剑桥大学印刷厂，1940 年）第 6 和第 8 章。）

另一方面，“数学归纳法”却完全不同于平常的“归纳”方法。它完全不是猜测的工作，而是一个命题的结论性的证明。事实上，在这里它是对于每个 n 都有一个的，因而是无限多个的命题的一个证明。它之所以叫做“归纳法”，仅仅是因为，人们在应用数学归纳法的技术之前，首先必须稍微判定一下，他要来证明的是什么。今后，在本书中，将把归纳法一词仅使用于这种情况，即当希望用数学归纳法来进行证明的情况——归纳法就指数学归纳法而言。

为证明等式(2)，有一个几何的方法。图 3 表明，对于 $n=6$，n^2 个小格分组成 $1+3+\cdots+(2n-1)$ 个小格。然而，在最终的分析中，只有当我们证明了对所有的 n，这种作图都可以实现时，才能认为这种图形是一个“证明”。而这实际上和用归纳法证明是相同的。

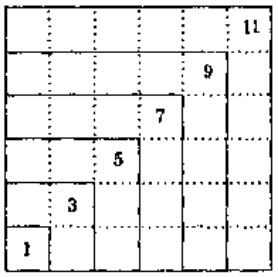


图 3　奇数之和是一个平方数

上述等式(2)的证明仅仅使用了(b)的特殊情况。我们证明了仅只 $P(n)$ 的正确性就意味着 $P(n+1)$ 的正确性。这是一个经常发生的重要的简单情况。让我们用下边的例子稍微进一步地说明这个方法的效力。用这样的规则，来定义斐波那契序列 $F_0, F_1, F_2, \cdots$，即 $F_0=0$，$F_1=1$ 而且以后每个项，都是它前边两个项的和。于是，这个序列开始是 0，1，1，2，3，5，8，13，…；这一序列将在 1.2.8 节中详细研究。我们现在将证明，如果 ϕ 是数 $(1+\sqrt{5})/2$，则对所有正整数 n，我们有

$$F_n \leqslant \phi^{n-1} \tag{3}$$

如果 $n=1$，则 $F_1=1=\phi^0=\phi^{n-1}$。所以步骤(a)已经完成。现在必须来做步骤(b)。$P(2)$ 也成立，因为 $F_2=1<1.6<\phi^1=\phi^{2-1}$。现在，如果 $P(1)$，$P(2)$，…，$P(n)$ 都为真而且 $n>1$，特别地，我们有 $P(n-1)$ 和 $P(n)$ 也为真；所以 $F_{n-1}\leqslant\phi^{n-2}$ 和 $F_n\leqslant\phi^{n-1}$。把这两个不等式加起来，得到

$$F_{n+1}=F_{n-1}+F_n\leqslant\phi^{n-2}+\phi^{n-1}=\phi^{n-2}(1+\phi) \tag{4}$$

数 ϕ 的重要性质在于：

$$\phi^2=\phi+1 \tag{5}$$

事实上，这正是为什么对于这个问题，一开始就选择这个数的原因。把(5)代入(4)中，就给出 $F_{n+1}\leqslant\phi^n$。这就是 $P(n+1)$。所以步骤(b)已经完成。于是(3)已经用数学归纳法证明了。注意在这里我们以两种不同的方法来达到步骤(b)：当 $n=1$ 时，直接证明了 $P(n+1)$；而当 $n>1$ 时我们用了归纳法。这是必要的，因为当 $n=1$ 时，我们对于 $P(n-1)=P(0)$ 的引用已经不合法了。

现在来看看，如何用数学归纳法来证明有关算法的事情。考虑欧几里得算法的以下的推广：

算法 E（扩充的欧几里得算法）。给了两个正整数 m 和 n，我们要计算它们的最大公因子 d 和两个整数 a 和 b，使得 $am+bn=d$。

E1.〔初始化〕 置 $a' \leftarrow b \leftarrow 1$，$a \leftarrow b' \leftarrow 0$，$c \leftarrow m$，$d \leftarrow n$。

E2.〔除〕 命 q，r 分别是 c 除以 d 的商和余数（有 $c = qd + r$，$0 \leqslant r < d$）。

E3.〔余数为0？〕 如果 $r = 0$，这个算法结束；在这种情况下，我们得到所希望的：$am + bn = d$。

E4.〔重新循环〕 置 $c \leftarrow d$，$d \leftarrow r$，$t \leftarrow a'$，$a' \leftarrow a$，$a \leftarrow t - qa$，$t \leftarrow b'$，$b' \leftarrow b$，$b \leftarrow t - qb$，并转回E2。∎

如果从这个算法中删掉变量 a，b，a' 和 b'，而且用 m，n 作为辅助变量 c，d，我们就得到原先的算法1.1E。除了确定系数 a，b 之外，新的算法并没有多做什么事情。假设 $m = 1769$ 和 $n = 551$。逐次地有（在步骤E2之后）。

a'	a	b'	b	c	d	q	r
1	0	0	1	1769	551	3	116
0	1	1	−3	551	116	4	87
1	−4	−3	13	116	87	1	29
−4	5	13	−16	87	29	3	0

答案是正确的：$5 \times 1769 - 16 \times 551 = 8845 - 8816 = 29$，即是 1769 和 551 的最大公因子。

问题是要证明这个算法对于所有的 m 和 n 都是行之有效的。我们可以通过设 $P(n)$ 是命题“算法E对于 n 和所有整数 m 有效”，试图把这搞成适宜用数学归纳法的样子。然而，这并不是那样容易地就证明了的，而且我们还需要证明某些额外的事实。在稍稍进行研究之后，我们发现，关于 a，b，a' 和 b' 必须证明某些事情，亦即每当执行步骤E2时，相应地下式：

$$a'm + b'n = c \quad am - bn = d \tag{6}$$

总是成立的。可以直接证明等式(6)如下；注意当我们头一次达到E2时，这肯定是正确的。而步骤E4并不改变(6)的正确性（参看习题1.2.1-6）。

现在，通过对 n 用归纳法，已经容易证明，算法E是正确的；如果 m 是 n 的一个倍数，显然算法有效，因为在头一次的E3时就立即完成了。当 $n = 1$ 时总是出现这个情况。剩下的仅有的情况是当 $n > 1$ 且 m 不是 n 的倍数时。在这种情况下，在头一次执行之后，算法进行到置 $c \leftarrow n$，$d \leftarrow r$，而且由于 $r < n$，由归纳法，我们可以假定最后的 d 值是 n 和 r 的最大公因子。根据在1.1节给出的论证，数偶 m，n 和 n，r 有相同的公因子，它们有相同的最大公因子。因此 d 是 m 和 n 的最大公因子，而且由等式(6)，$am - bn = d$。

上述证明中的黑体字的短句，是在归纳法证明中经常使用的习惯语言：当在做归纳法结构的部分(b)时，不是说“我们现在将假定 $P(1)$，$P(2)$，…，$P(n)$，并在这个假定下，我们来证明 $P(n+1)$”，而通常是简单地说成“我们现在将证明 $P(n)$；由归纳法，我们可以假定当 $1 \leqslant k < n$ 时，$P(k)$ 为真”。

如果非常严密地检查上述的论证，并稍微改变我们的观点，就能看到一个可以应用于证明任何算法的正确性的一般方法。这个思想是对某个算法给出一个框图，且当计算通过框图中的每个箭头时，为其标以关于当前事态的论断。请看图4，其中每个断言已经标为 $A1$，$A2$，…，$A6$（所有这些断言都有附加的约定，即变量均为整数；为节省篇幅起见一概略去不写了）。$A1$ 给出了进入算法时的初始假定，$A4$ 说明了关于输出值 a，b 和 d，

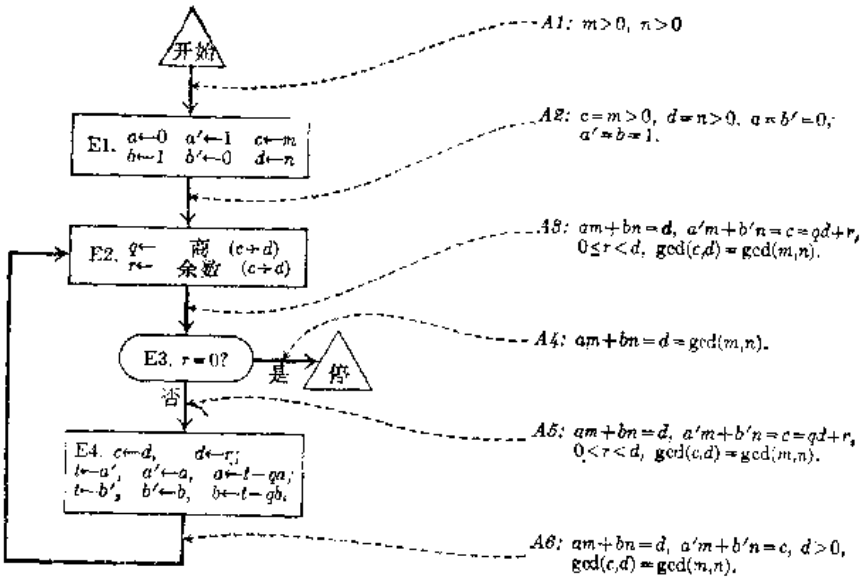


图 4　算法 E 的框图，并标以证明算法之正确性的论断

我们希望证明的是什么？

一般的方法是这样组成的，即对于框图中的每一框，来证明：如果在执行本框的运算之前，在指向本框的诸箭头中有任何一个，其上的断言为真，则在执行了本框的运算之后，所有由本框引出的诸箭头上的断言也都为真。例如，我们必须证明，在 E2 前面的或者 $A2$ 或者 $A6$ 都能推出在 E2 后面的 $A3$（在这种情况下，$A2$ 是比 $A6$ 更强的命题，即是说，$A2$ 蕴涵 $A6$，所以仅仅需要证明：在 E2 前面的 $A6$ 推出后面的 $A3$。注意 $A6$ 中的条件 $d>0$ 是必要的，因为它正好保证了 E2 的运算能有意义）；还必须证明 $A3$ 和 $r=0$ 推出 $A4$；$A3$ 和 $r\neq 0$ 推出 $A5$ 等等。每个所需的证明都是非常直接了当的。

一旦上边的黑体字的命题已经对每一框都证明了，则可结论，在算法的任何一次执行时，所有的断言都为真。因为，现在可以对计算的步数——指的是在框图中通过的箭头的个数——使用归纳法。在通过头一个箭头，即由“开始”引出的箭头时，断言 $A1$ 为真，因为总是假定我们的输入值满足此断言的规定。所以在通过头一个箭头时的断言是正确的。如果标于第 n 个箭头的断言为真，则根据黑体字的命题，标于第 $n+1$ 个箭头的断言也为真。

使用这一般的方法，很显然地，就把证明一个给定的算法的正确性的问题，基本上归结为这样一个问题，即提出正确的断言并置入框图中去。一旦完成了这种“归纳跳跃”，关于“从进入一个框的每一断言都能推出从该框引出的每一断言”的证明，主要是事务性的工作。事实上，一旦人们找到了很少几个断言之后，提出其它断言的工作就是非常事务性的了。这样说来，在这里的例子中，一旦给出了 $A1$ 和 $A6$，要想写出 $A2$、$A3$、$A4$ 和 $A5$ 实质上必须是什么，就显得非常简单了。在我们的例子中，只有断言 $A6$ 才是证明的“创造性”部分，而所有其余的部分，原则上说都可以机械地补充之。因此，在这以后，本书将不打算对于每一个算法都给出详细的形式证明——只须指出关键的归纳法断言就足够了，而其余的断言或者出现在算法后边的讨论中，或者在算法本身的叙述中作为带圆括号的解释来给出。

对于算法的上述证明原则，也许还有甚至更为重要的另一个方面：它反映了我们“理解”一个算法的途径。回想一下，在1.1节里，我曾经告诫读者不要期望象读一篇小说那样来读一个算法；建议读者要以某些数据为例来顺着算法跑一两趟。之所以要特别如此，是因为算法的实施实例会帮助人们在脑子里形成各种断言。作者认为，只有当我们已经达到了这样的程度，即在脑子里完全地映入了所有的断言时，如同在图4中所做的那样，我们才能真正理解一个算法为什么是正确的。为了能把算法由一个人向另一个人（或者当一个人在数月之后来考察他自己的算法时，可能就成为一个人向他自己）进行适当的介绍，这个观点导出重要的“心理学上的”推论：它意味着，那些不能由自动机器简易地推导出来的关键性断言，当正在向另一个人说明一个算法时，总是应该明确地加以指出。当提供算法E时，断言$A6$尤其应当指出。

然而，细心的读者将会发现在关于算法E的最后证明中的一个漏洞。根本没有证明算法终止；所证明的只是，如果它终止，那么它给出的回答正确！

（例如，注意一下，如果允许变量m，n，c和r取形如$u+v\sqrt{2}$的值，其中u和v都是整数，则算法E仍然有意义。变量q，a，b，a'，b'都仍然取整数的值。比如说，我们以$m=12-6\sqrt{2}$和$n=20-10\sqrt{2}$来开始这个算法，将算出一个“最大公因子”$d=4-2\sqrt{2}$，而$a=+2$，$b=-1$。甚至在这个扩充的假定之下，断言$A1$到$A6$的证明仍然保持有效。因此，贯穿算法的任何一个执行，所有断言都为真。但是，如果我们以$m=1$和$n=\sqrt{2}$来开始这一步骤，则计算就永不得终止（见习题12）。因此，断言$A1$到$A6$的证明逻辑上并不保证算法是有穷的）。

因此，关于终止的证明通常是独立地处理的。在许多重要的情况下，也有可能来扩充上述的方法，使得关于终止的证明，作为一个付产品被包括在内，如习题13中所示。

现在已经两次证明了算法E的正确性。为了逻辑上的严格性，也应当试图来证明本节中的头一个算法，即算法I是正确的。事实上，我们已经用算法I来确立了用归纳法证明的任何证明的正确性。然而，如果又试图来证明算法I本身行之有效，我们就陷入了前后矛盾的困境——如果不再次地使用归纳法，我们就不能真正地证明它！这个论证将是循环的。

在最后的分析中，各整数的每一个性质也必须用归纳法，大体沿着这条路线来进行证明。因为如果我们着力于研究基本概念，则整数实质上要用归纳法来定义。因此，我们可以把这样一个思想取作公理，即任何正整数n或者等于1，或者可以从1开始，重复地“加1”来达到它；这就足以证明算法I是正确的了。（关于整数的基础概念的严格的研究，请参看利昂·亨金（Leon Henkin）的论文《论数学归纳法》(On Mathematical Induction)，AMM 67 (1960)，323～338）。

蕴涵在数学归纳法中的思想，同数的概念有着极其密切的关系。首次应用数学归纳法来进行严格的证明的欧洲人，是意大利科学家弗朗西斯科·莫洛里科（Francesco Maurolico），其时是1575年。皮尔·德·费尔玛（Pierre de Fermat）于17世纪初作了进一步的改进；他把它叫做“无限后继的方法”。这个思想在布莱斯·帕斯卡尔(Blaise Pascal)稍后的著作中（1653年）也可清楚地看到。“数学归纳法”一词是由阿·德·摩根（A. de Morgan）在19世纪初创造的（见AMM 24(1917)，199～207；25(1918)，197～201；Arch.

Hist. Exact. Sci. 9 (1972), 1～21〕。关于数学归纳法的进一步的讨论，请参看乔治·波利亚（G. Po′lya)所著的《数学中的归纳与类比》(Induction and Analogy in Mathematics)（新泽西州普林斯顿：普林斯顿大学印刷厂，1954 年）一书中的第七章。

如上所给出的，算法证明借助于断言和归纳法的系统阐述，实质上应归功于罗·W·弗洛伊德（R. W. Floyd)。他指出在一个程序设计语言中，每个运算的语义定义，最为严格地以一个逻辑规则给出，这个规则确切地指出，在运算之后什么断言可被证明，预先是从什么断言开始的〔请参看《对程序赋以意义》(Assigning Meanings to Programs), Proc. Symp. Appl. Math. Amer. Math. Soc., 19 (1967), 19～32〕。类似的思想也由彼得·瑙尔（Peter Naur）独立地提出了〔见 BIT 6 (1966), 310～316〕，他把断言叫做"一般的快照"。查·安·理·霍尔（C. A. R. Hoare）引进了一个重要的改进，即"不变量"的思想。例如，参看 CACM 14(1971), 39～45。归纳法断言的思想实际上是以萌芽的形式出现于1946年，在赫·海·戈德斯坦(H. H. Goldstine)和约·冯·诺依曼（J. Von Neumann)引进框图的概念的同时，在那些原始的框图中就包括了"断言框"，它们非常类似于图 4 中的断言〔见 John von Neumann, Collected Works 5 (纽约：麦克米伦，1963)，91～99〕。

习题

1.〔05〕 如果我们要对于所有的非负整数，即对于 $n=0,1,2,\cdots$，而不是对于 $n=1,2,3,\cdots$ 来证明某个命题 $P(n)$，试说明在这种情况下，怎样来修改用数学归纳法证明的思想？

▶2.〔15〕 以下的证明中必然有某些错误，试指出错误之所在？"定理。设 a 为任意正数。则对所有的正整数 n，我们有 $a^{n-1}=1$。证明：如果 $n=1$，$a^{n-1}=a^{1-1}=a^0=1$。而且用归纳法，假定定理对于 $1,2,\cdots,n$ 都为真，我们有

$$a^{(n+1)-1}=a^n=\frac{a^{n-1}\times a^{n-1}}{a^{n-2}}=\frac{1\times 1}{1}=1$$

所以定理对于 $n+1$ 也为真。"

3.〔18〕 以下的归纳法证明似乎是正确的，但由于某种原因，对 $n=6$，等式的左边给出 $\frac{1}{2}+\frac{1}{6}+\frac{1}{12}+\frac{1}{20}+\frac{1}{30}=\frac{5}{6}$，而右边却给出 $\frac{3}{2}-\frac{1}{6}=\frac{4}{3}$。你能找出一个错误来吗？"定理：

$$\frac{1}{1\times 2}+\frac{1}{2\times 3}+\cdots+\frac{1}{(n-1)\times n}=\frac{3}{2}-\frac{1}{n}$$

证明 我们对 n 使用归纳法。对于 $n=1$，$\frac{3}{2}-\frac{1}{n}=1/(1\times 2)$；而且，假定定理对于 n 为真，则

$$\frac{1}{1\times 2}+\cdots+\frac{1}{(n-1)\times n}+\frac{1}{n\times(n+1)}=\frac{3}{2}-\frac{1}{n}$$

$$+\frac{1}{n(n+1)}=\frac{3}{2}-\frac{1}{n}+\left(\frac{1}{n}-\frac{1}{n+1}\right)=\frac{3}{2}-\frac{1}{n+1}"$$

4.〔20〕 证明，除等式（3）外，还有 $F_n\geqslant\phi^{n-2}$。

5.〔21〕 一个质数是一个大于1的，而且除了1和其本身之外没有其它真因子的整数。利用这个定义和数学归纳法，证明每个大于1的正整数，均可以写作质数的乘积。

6.〔20〕 证明如果等式（6）在执行步骤E4之前成立，则在执行之后也成立。

7.〔23〕 对于1^2，2^2-1^2，$3^2-2^2+1^2$，$4^2-3^2+2^2-1^2$，$5^2-4^2+3^2-2^2+1^2$等等和数，用公式表示之，并用数学归纳法予以证明。

▶8.〔25〕（a）用归纳法证明以下的尼科梅彻斯（Nicomachus）定理（约公元100年）：$1^3=1$，$2^3=3+5$，$3^3=7+9+11$，$4^3=13+15+17+19$，等等。（b）利用这一结果来证明下面著名的公式

$$1^3+2^3+\cdots+n^3=(1+2+\cdots+n)^2$$

〔注记：罗·W·弗洛伊德向作者提示了这个公式的有趣的几何解释，如图5所示。这个思想同尼科梅彻斯定理和图3有关〕。

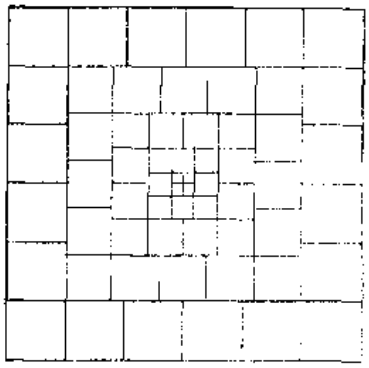


图5 对于$n=5$，习题8的几何形式

$$\begin{aligned}
\text{边}&=5+5+5+5+5+5=5\cdot(5+1)\\
\text{边}&=5+4+3+2+1+1+2+3+4+5\\
&=2(1+2+\cdots+5)\\
\text{面积}&=4\cdot1^2+4\cdot2\cdot2^2+4\cdot3\cdot3^2\\
&\quad+4\cdot4\cdot4^2+4\cdot5\cdot5^2\\
&=4(1^3+2^3+\cdots+5^3)
\end{aligned}$$

9.〔20〕 用归纳法证明，如果$0<a<1$，则$(1-a)^n\geqslant1-na$。

10.〔*M*22〕 用归纳法证明，如果$n\geqslant10$，则$2^n>n^3$。

11.〔*M*30〕 求出和

$$\frac{1^3}{1^4+4}-\frac{3^3}{3^4+4}+\frac{5^3}{5^4+4}-\cdots+\frac{(-1)^n(2n+1)^3}{(2n+1)^4+4}$$

的简单公式并证明之。

12.〔*M*25〕 试说明算法E可以如何加以推广，使得如正文中所述的那样，它将接受形如$u+v\sqrt{2}$的输入值，其中u和v都是整数，而且计算仍然能以初等的方式进行（即，无须使用$\sqrt{2}$的无限的十进展开）。然而，试证明，如果$m=1$和$n=\sqrt{2}$，则计算将不终止。

▶13.〔*M*23〕 通过加上一个新变量T和在每步的开始处加上运算“$T\leftarrow T+1$”，来扩充算法E（这样，T就好象是个时钟一样，计算着执行的步数）。假设T开始为0，故图4中的断言*A*1变成为“$m>0$，$n>0$，$T=0$”。附加的条件“$T=1$”将类似地添入*A*2。试说明如何把附加的条件补充到诸断言中，使得*A*1，*A*2，…，*A*6中的任何一个都推出$T\leqslant n$；而且使得归纳法证明仍然得以进行（因此，计算必须在至多$3n$步内终止）。

14.〔50〕（罗·W·弗洛伊德）试编制一个计算机程序，它接受某种程序设计语言的程序连同任选的断言一起作为输入，而且它还试图来添进剩下的，为证明此计算机程序的正确性所必要的断言（例如，努力来编出一个能够证明算法E的正确性的程序，假定仅仅给定*A*1和*A*6。关于这个问题的进一步的讨论，请参看罗·W·弗洛伊德和詹·科·金（J. C. King）的论文，发表在1971年的IFIP Congress Proceedings）。

▶15.〔*HM*28〕（广义的归纳法）正文中说明了怎样来证明依赖于单个整数 n 的命题 $P(n)$，但没有叙述怎样来证明依赖于两个整数的命题 $P(m, n)$。在这种情况下，通常是通过某种类型的“双重归纳法”来给出一个证明的，它经常显然含混不清。实际上，有一个比简单的归纳法更为一般的重要原理，它不仅可应用于这种情况，而且也可应用于对于不可数的集合来证明命题的情况。例如说，对于所有的实数 x 的 $P(x)$。这个一般原理叫做良序原则。

设“$\prec$”是集合 S 上的一个关系，它满足以下的性质：

i）给定 S 中的 x，y，z，如果 $x \prec y$ 和 $y \prec z$，则 $x \prec z$。

ii）给定 S 中的 x，y，以下三种可能性中恰有一种为真：$x \prec y$，$x = y$，$y \prec x$。

iii）如果 A 是 S 的任何非空子集，则 A 中有一个元素 x，使得对于 A 中所有的 y，有 $x \preceq y$。

则称这个关系为 S 的一个良序关系。例如，对于通常的“小于”关系 $<$，正整数显然是良序的。

a）试证所有整数的集合对于 $<$ 不是良序的。

b）对于所有整数的集合，试定义一个良序关系。

c）所有非负实数的集合对于 $<$ 是否良序？

d）（辞典编辑次序。）设 S 对于 $\prec$ 良序，而且对于 $n > 0$，设 T_n 是所有的 n 元的 $(x_1, x_2, \cdots, x_n)$ 集合，其中元素 x_i 在 S 中。定义 $(x_1, x_2, \cdots, x_n) \prec (y_1, y_2, \cdots, y_n)$，如果有某一个 k，$1 \leqslant k \leqslant n$，使得对于 $1 \leqslant j < k$，$x_j = y_j$，但在 S 中有 $x_k \prec y_k$。问 $\prec$ 是 T_n 的一个良序吗？

e）作为小习题（d）的一部分，设 $T = U_{n \geqslant 1} T_n$。定义 $(x_1, x_2, \cdots, x_n) \prec (y_1, y_2, \cdots, y_m)$，如果对于某个 $k \leqslant m$，n，有：$x_j = y_j$ 对于 $1 \leqslant j < k$，而且 $x_k \prec y_k$；或者如果有：$x_j = y_j$ 对于 $1 \leqslant j \leqslant n$，而且 $n < m$。问 $\prec$ 是 T 的一个良序吗？

f）试证：$\prec$ 是 S 的一个良序，当且仅当它满足上述的（i）和（ii），而且对于所有的 $j \geqslant 1$，不存在具有 $x_{j+1} \prec x_j$ 的无限序列 x_1，x_2，x_3，$\cdots$。

g）设 S 对于 $\prec$ 为良序集，且设 $P(x)$ 是关于 S 之元素 x 的一个命题。试证：如果 $P(x)$ 能在 $P(y)$ 对于所有的 $y \prec x$ 为真的假定下得证，则 $P(x)$ 对 S 中所有的 x 为真。

〔注意：小题（g），是原来的简单归纳法的推广。在 S 等于正整数的情况下，它恰巧是在正文中所讨论的数学归纳法的简单情况。注意，要求我们证明，如果对于所有的正整数 $y < 1$，$P(y)$ 为真，则 $P(1)$ 为真；这就等于说，我们应当证明 $P(1)$。因为对于所有这样的 y，$P(y)$ 肯定地（空虚地）为真。结果，人们发现，在许多情况下，使用一个特殊的论证，$P(1)$ 就不必证明了。

小题（d），同小题（g）相联系，特别给我们提供了用来证明关于 n 个正整数 m_1，m_2，$\cdots$，m_n 的命题 $P(m_1, m_2, \cdots, m_n)$ 的 n 元归纳法——更为强有力的方法。

小题（f）对于计算机算法有着进一步的应用：如果我们能够把一个计算的诸状态映射到一个良序集 S，使得计算的每一步骤总是从一个状态 x 进入到一个状态 y，且有 $f(y) \prec f(x)$，则算法必须终止。这一原理推广了在证明算法 1.1E 终止中所使用的，关于 n 的

值严格递降的论证)。

1.2.2 数、幂和对数

现在让我们好好来考察考察这个一直都在谈论着的“数”，以便开始我们对于数值数学的研究。整数是全体数

$$\cdots, -3, \ -2, \ -1, \ 0, \ 1, \ 2, \ 3, \cdots$$

(正的，负的或0)。有理数是两个整数的比(商) p/q，其中 q 为正。实数是具有一个“十进展开”：

$$x = n + 0.d_1d_2d_3\cdots \tag{1}$$

的量，其中 n 是一个整数，而每一个 d_i 是0与9之间的一个数字，并且不出现9的无限序列。表示式(1)意味着，对于所有的正整数 k，

$$n + \frac{d_1}{10} + \frac{d_2}{100} + \cdots + \frac{d_k}{10^k} \leqslant x < n + \frac{d_1}{10} + \frac{d_2}{100} + \cdots + \frac{d_k}{10^k} + \frac{1}{10^k} \tag{2}$$

实数而不是有理数的两个例子是：

$\pi = 3.14159265358979\cdots$，一个圆的圆周对于它的直径的比值；

$\phi = 1.61803398874989\cdots$，“黄金比率”$(1+\sqrt{5})/2$(见1.2.8小节)

附录B中给出了有四十位小数的精确度的重要常数表。我们将不讨论加法、减法、乘法、除法，以及实数之比较等许多熟知的性质。

在这一小节中，我们始终以字母 b 表示一个正实数。如果 n 为一整数，则 b^n 以熟知的规则定义之：

$$b^0 = 1, \quad b^n = b^{n-1}b \quad \text{如果} \quad n > 0, \quad b^n = b^{n+1}/b \quad \text{如果} \quad n < 0 \tag{3}$$

用归纳法容易证明，当 x 和 y 为整数时，指数律成立：

$$b^{x+y} = b^x b^y \qquad (b^x)^y = b^{xy} \tag{4}$$

如果 u 是一个正实数，且 m 是一个正整数，则总有唯一的正实数 v，是 u 的“m次根”，即 $v^m = u$。我们把它写成 $v = \sqrt[m]{u}$。

我们现在对于有理数 r，定义 b^r 如下：

$$b^{p/q} = \sqrt[q]{b^p} \tag{5}$$

这个定义是由奥雷斯姆 (Oresme) (约1360年) 给出的。这是一个很好的定义，因为 $b^{np/nq} = b^{p/q}$，而且还因为，甚至当 x 和 y 是任意有理数时，指数律仍然成立(见习题9)。

最后，我们对所有的实数值的 x 来定义 b^x。首先假定 $b > 1$。如果 x 是由等式(1)给出的，则我们要求

$$b^{n+d_1/10+\cdots+d_k/10^k} \leqslant b^x < b^{n+d_1/10+\cdots+d_k/10^k+1/10^k} \tag{6}$$

这就把 b^x 定义成为一个唯一的正实数了。因为方程(6)左端和右端之间的差是 $b^{n+d_1/10+\cdots+d_k/10^k}(b^{1/10^k}-1)$，由以下的习题13，这个差小于 $b^{n+1}(b-1)/10^k$，因此如果取充分大的 k 值，则对于 b^x 就可以得到任何所希望的精度。

例如，我们求得

$$10^{0.30102999} = 1.9999999737\cdots, \quad 10^{0.30103000} = 2.0000000198\cdots \tag{7}$$

因此，如果 $b = 10$，$x = 0.30102999\cdots$，我们就知道 10^x 的具有精度优于千万分之一的值

(尽管我们仍然不知道 10^x 的十进展开究竟是 1.999…还是 2.000…!)

当 $b<1$ 时，我们定义 $b^x=(1/b)^{-x}$。而当 $b=1$ 时，定义 $1^x=1$。通过这个定义，可以证明对任何的实数值 x 和 y，指数律（等式(4)）成立。定义 b^x 的这个思想是由约翰·沃利斯（John Wallis，1655）和伊萨·牛顿（Issac Newton，1669）首先系统阐述的。

现在我们转到一个重要的问题。假设给定了一个正实数 y，我们能否找到一个实数 x，使得 $y=b^x$？回答是“能”（假定 $b\neq 1$）。因为当给定了 $b^x=y$ 时，我们可以简单地使用方程（6）来进行求逆，以确定 n 和 d_1，d_2，…。如此所得的数 x 叫做以 b 为底的 y 的对数，而且我们把这写作 $x=\log_b y$。通过这个定义，我们有

$$x=b^{\log_b x}=\log_b(b^x) \tag{8}$$

作为一个例子，等式（7）说明

$$\log_{10} 2=0.30102999\cdots \tag{9}$$

从指数律，得出

$$\log_b(xy)=\log_b x+\log_b y \quad \text{如果 } x>0,\ y>0 \tag{10}$$

且

$$\log_b(c^y)=y\log_b c \quad \text{如果 } c>0 \tag{11}$$

等式（9）是所谓的“常用对数”，即以 10 为底的对数。人们可能会想到，在计算机中，二进制对数（即以 2 为底）可能是更为有用的，因为计算机中通常使用二进的算术。实际上，我们将看到，二进制对数确是非常有用的，但并非仅仅由于这个原因；主要原因在于，一个计算机算法通常都分叉成两路分支。

既然二进对数如此经常地出现，因而就希望对它使用一个更简短的记号。因此，我们将写

$$\lg x\equiv\log_2 x$$

现在出现了这样一个问题，就是在 $\lg x$ 和 $\log_{10} x$ 之间有没有任何关系？好得很，确有一个。因为根据等式（8）和（11），

$$\log_{10} x=\log_{10}(2^{\log_2 x})=(\log_2 x)(\log_{10} 2)$$

因此，$\lg x=\log_{10} x/\log_{10} 2$，而且，一般说来，我们求得

$$\log_c x=\log_b x/\log_b c \tag{12}$$

等式（10），（11）和（12）是处理对数的基本规则。

然而，在大多数情况下，真正最方便地切合实际应用的，既不是以 10 为底，也不是以 2 为底。有一个实数，记之为 e，$e=2.718281828459045\cdots$，以它为底的对数有着更为简单的性质。习惯上，我们把以 e 为底的对数称作“自然对数”，而且我们写

$$\ln x\equiv\log_e x \tag{13}$$

这种显得颇为随意的定义（事实上，我们并没有真正地定义过 e），大概不能使读者信服是一个非常“自然”的对数。但我们将会发现，越是用它来计算，$\ln x$ 就越是显得自然。约翰·内皮尔（John Napier）实际上在公元 1590 年之前就发现了自然对数（稍微有些变态，而且没有把它同乘幂联系起来），这比知道其它类型的对数要早许多年。不加证明，我们立即就能给出两个简短的例子，来说明为什么这种对数才显得最“自然”：（a）在图 6 中，阴影部分的面积是 $\ln x$。（b）如果一个银行以利率 r 来支付复利，一年两次地计

算复利，则每一元钱一年的本利为 $(1+r/2)^2$ 元。如果以季来计算复利，则将得到 $(1+r/4)^4$ 元。而如果以天来计算，则大概将得出 $(1+r/365)^{365}$ 元。现在，如果连续地来计算复利，则每一元将精确地得出 e^r 元（忽略舍入误差）！在现今这个计算机的时代里，某些银行家现在实际上已经达到了这个极限的公式。

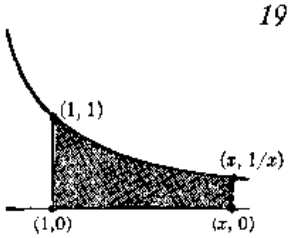


图 6　自然对数

关于对数和指数概念的有趣的历史，请参看费　卡乔里（F. Cajori）的一系列论文：*AMM* 20(1913)，5～14，35～47，75～84，107～117，148～151，173～182，205～210。

通过考虑怎样来计算对数，我们来把这一小节总结一下。一个方法是直接由方程（6）引出的：如果我们设 $b^x=y$，而且将方程（6）的所有部分都作 10^k 次方幂，我们将得出

$$b^m \leqslant y^{10^k} < b^{m+1} \tag{14}$$

其中 m 为某一整数。为了求得 y 的对数，我们所要做的，就是把 y 自乘到这么大次幂，然后找出这个结果是在 b 的怎样的次幂（m，$m+1$）之间，最后 $m/10^k$ 才是精确到 k 位小数的答案。

把这个明显地不实际的方法稍作修改，就导出一个简单而又合理的步骤。我们现在就来说明，应当怎样来计算 $\log_{10} x$ 并把它表示成二进制的形式：

$$\log_{10} x = n + b_1/2 + b_2/4 + b_3/8 + \cdots$$

首先，我们把 x 的小数点进行左移或右移，使得 $1 \leqslant x/10^n < 10$，这就为我们确定了 n。现在，我们置 $x_0 = x/10^n$，而且对于 $k \geqslant 1$，置

$$\begin{aligned} &b_k = 0 \qquad x_k = x_{k-1}^2 \qquad \text{如果} \quad x_{k-1}^2 < 10 \\ &b_k = 1 \qquad x_k = x_{k-1}^2/10 \qquad \text{如果} \quad x_{k-1}^2 \geqslant 10 \end{aligned} \tag{15}$$

从而得到了 b_1，b_2，b_3，…。这个步骤的正确性，是由如下事实推出的：即对于 $k=0$，1，2，…，

$$1 \leqslant x_k = x^{2^k}/10^{2^k(n+b_1/2+\cdots+b_k/2^k)} < 10 \tag{16}$$

而本事实是容易用归纳法证明的。

当然，事实上我们必然只能以有限的精度来进行计算，所以我们不能精确地置 $x_k = x_{k-1}^2$，我们只是置 $x_k = x_{k-1}^2$ 的舍入或截取到某个小数位的数。例如，这里是 $\log_{10} 2$ 舍入到四位有效数字的计算：

$x_0 = 2.000$			
$x_1 = 4.000$	$b_1 = 0$	$x_6 = 1.845$	$b_6 = 1$
$x_2 = 1.600$	$b_2 = 1$	$x_7 = 3.404$	$b_7 = 0$
$x_3 = 2.560$	$b_3 = 0$	$x_8 = 1.159$	$b_8 = 1$
$x_4 = 6.554$	$b_4 = 0$	$x_9 = 1.343$	$b_9 = 0$
$x_5 = 4.295$	$b_5 = 1$	$x_{10} = 1.804$	$b_{10} = 0$

等等。计算的误差已经引起了误差的传播。x_{10} 的真实的值是 1.7977。这将最终地导致不能正确地计算 b_{10}；而且我们得到二进制值 0.0100110100010000011，它相当于十进制的 0.301031，而不是等式（9）中所给出的真实的值。

对于类似于此的任何一个方法，有必要来检查由于强加的限制所引起的计算误差。这

一小节的习题 27 推出了这个误差的上限。如果计算到如上所述的四位数字，则我们推出，对数值中的误差将小于 0.00014。而我们上边的答案，比起这个来，是更加精确的。这主要是因为 x_0，x_1，x_2 和 x_3 是精确地得出的。

这个方法虽简单，但十分有趣。然而它大概不是在计算机上计算对数的最好的途径。习题 25 中给出了另一种方法。

习题

1.〔00〕 什么是最小的正有理数？

2.〔00〕 $1+0.239999999\cdots$ 是一个十进展开吗？

3.〔02〕 $(-3)^{-3}$ 等于什么？

▶4.〔05〕 $(0.125)^{-2/3}$ 等于什么？

5.〔05〕 我们已经借助于一个十进展开定义了实数。换一下，试讨论我们怎样借助于一个二进展开来定义，并给出替换方程（2）的一个定义来。

6.〔10〕 设 $x=m+0.d_1d_2\cdots$ 和 $y=n+0.e_1e_2\cdots$ 都是实数。请给出一个以十进表示为基础的，用来确定究竟是 $x=y$，$x<y$ 或 $x>y$ 的规则。

7.〔*M*23〕 假定 x 和 y 都是整数。试由等式（3）给出的定义开始，证明指数律。

8.〔25〕 设 m 是一个正整数。证明每一个正实数 u 有一个唯一的正的 m 次方根——通过给出一个方法，逐次地构造出这个根的十进展开的 n，d_1，d_2，等等来证明。

9.〔*M*23〕 假定在 x 和 y 为整数时指数律成立。证明在 x 和 y 都是有理数时指数律也成立。

10.〔18〕 证明 $\log_{10}2$ 不是一个有理数。

▶11.〔10〕 如果 $b=10$ 且 $x=\log_{10}2$，则为了确定 b^x 的十进展开的头三位小数，我们需要知道 x 值的精度为多少位小数？（注意：在讨论中，你可以利用习题 10 的结果）。

12.〔02〕 说明怎样从等式（7）得出等式（9）？

▶13.〔*M*23〕 （a）假定 x 是一个正实数且 n 是一个正整数，证明 $\sqrt[n]{1+x}-1\leqslant x/n$。（b）用这一事实来论证方程（6）之后的陈述。

14.〔15〕 证明等式（11）。

15.〔10〕 证明或者反驳：

$$\log_b x/y=\log_b x-\log_b y,\ \text{如果 } x,y>0。$$

16.〔00〕 借助于 $\ln x$ 和 $\ln 10$ 怎样来表达 $\log_{10}x$？

▶17.〔05〕 $\lg 32$ 等于什么？$\log_\pi \pi$ 呢？$\ln e$ 呢？$\log_b 1$ 呢？$\log_b(-1)$ 呢？

18.〔10〕 证明或者反驳：$\log_8 x=-\frac{1}{2}\lg x$。

▶19.〔20〕 如果 n 是一个 14 位整数，则 n 的值能适合于一个有 47 个二进位加上一位符号位的计算机的字长吗？

20.〔10〕 在 $\log_{10}2$ 和 $\log_2 10$ 之间有无任何简单的关系？

21.〔15〕 借助于 $\ln(\ln x)$，$\ln(\ln b)$ 和 $\ln b$ 来表达 $\log_b(\log_b x)$。

▶22.〔20〕 证明

$$\lg x \approx \ln x + \log_{10} x$$

且误差小于 1 %！（于是也可用自然对数表和常用对数表来查到二进对数的近似值）。

23.〔*M*25〕 以图 6 为基础，给出 $\ln xy = \ln x + \ln y$ 的一个几何证明。

24.〔15〕 说明在这小节的末尾处用来计算以 10 为底的对数的方法，如何加以修改，以产生以 2 为底的对数。

25.〔20〕 假设我们有一个二进制的计算机和一个数 x，$1 \leqslant x < 2$。证明以下的算法，它仅使用移位，加法，减法（同所需要的精度的小数位数相适应的）运算，即可以用来计算 $y = \log_b x$ 的一个近似值：

L1.〔初始化.〕 置 $y \leftarrow 0$，$z \leftarrow x$ 右移一位，$k \leftarrow 1$。

L2.〔检查是否结束.〕 如果 $x = 1$，即停止。

L3.〔移位.〕 如果 $x - z < 1$，则置 $z \leftarrow z$ 右移一位，$k \leftarrow k + 1$。并重复这一步骤。

L4.〔缩小数值.〕 置 $x \leftarrow x - z$，$z \leftarrow x$ 右移 k 位，$y \leftarrow y + \log_b(2^k/(2^k - 1))$，并转到 L2。▌

〔注意：这一方法非常类似于在计算机中进行除法的方法。这个思想实际上可追溯到亨利·布里格斯（Henry Briggs），他用它（以十进形式而不是二进形式）来计算对数。发表于 1624 年。我们需要有一个和计算机的精度一样多位的 $\log_b 2$，$\log_b(4/3)$，$\log_b(8/7)$ 等的一个辅助的表。这个算法包含有人为的计算误差，因为数被右移了，所以 x 的值终将缩小到 1，而且算法将终止。这一习题用来说明为什么上面的算法将终止和为什么它算出 $\log_b x$ 的一个近似值〕。

26.〔*M*27〕 以算术运算中的精度为基础，来确定前面习题中算法的精度的上限。

▶27.〔*M*25〕 考虑在正文中讨论的计算 $\log_{10} x$ 的方法。命 x'_k 表示所计算的 x_k 的近似值，如下地确定之：$x(1-\eta) \leqslant 10^n x'_0 \leqslant x(1+\epsilon)$；而且在由等式（15）确定 x'_k 时，用数量 y_k 代替 $(x'_{k-1})^2$，其中 $(x'_{k-1})^2(1-\eta) \leqslant y_k \leqslant (x'_{k-1})^2(1+\epsilon)$ 且 $1 \leqslant y_k < 100$。这里 η 和 ϵ 都是小常数，用来反映由于舍入或截取而产生的最高和最低误差。如果以 $\log' x$ 表示计算的结果，试证明在 k 步之后，我们有

$$\log_{10} x + 2\log_{10}(1-\eta) - 1/2^k < \log' x \leqslant \log_{10} x + 2\log_{10}(1+\epsilon)$$

28.〔*M*30〕 理·范曼（R. Feynman）提出一个仅仅使用移位，加法和减法（类似于习题 25 中的算法）的方法，来计算 b^x（当 $0 \leqslant x < 1$ 时），并分析其精度。

29.〔*HM*20〕 设 x 是大于 1 的实数。（a）对于什么实数 $b > 1$，$b \log_b x$ 为极小值？（b）对于什么整数 $b > 1$，它的值为极小？（c）对于什么整数 $b > 1$，$(b+1)\log_b x$ 为极小值？

1.2.3 和与积

设 a_1，a_2，…是数的任意序列。我们经常要研究诸如 $a_1 + a_2 + \cdots + a_n$ 这样的和，而且用下边的记号把这个和更紧凑地写成：

$$\sum_{1 \leqslant j \leqslant n} a_j \qquad (1)$$

如果 n 为 0 或负数，则这个和数的值定义作 0。一般地说，如果 $R(j)$ 是关于 j 的任意关

系，符号

$$\sum_{R(j)} a_j \tag{2}$$

表示对于所有那样的 a_j 求和，其中 j 是满足条件 $R(j)$ 的整数。如果不存在这样的整数，记号（2）就表示 0。（1）和（2）中的字母 j 是一个“哑下标”或“下标变量”，引进它仅仅是为了用于上述记号。用作下标变量的符号通常是字母 i，j k，m，n，r，s，t（偶而还带有下标或撇号）。利用 Σ 和下标变量来表示求和是由约·拉格朗日（J. Lagrange）于 1772 年引进的。

记号 $\Sigma_{R(j)}a_j$ 在本书中用作（2）的一个压缩的形式。

严格说来，记号（1）是含混的。因为求和究竟是对于 j 来取还是对于 n 来取，是不完全清楚的。在这个具体情况下，把（1）解释成对于 $n \geqslant j$ 的值求和，那是未免太糊涂了。但是，却十分可能造成有意义的例子，其中之下标变量并不是明确地确定的。例如 $\Sigma_{j<k}b^j$。在这样的情况下，在行文中必须说清楚哪个变量是一个哑变量？又哪一个变量还要在这个记号之外出现，还有另外的意义？上面那句中的例子大概仅当或者 j 或者 k（而非两者都是）有外部意义时才使用。

在大多数情况下，记号（2）仅当求和是有限时，即仅当有限个 j 值满足 $R(j)$ 时，如象（1）中那样，才使用。当使用一个无限的求和时，例如

$$\sum_{j \geqslant 1} a_j = a_1 + a_2 + a_3 + \cdots$$

则必须使用微积分的技术。因此（2）的精确的意义是

$$\sum_{R(j)} a_j = \left(\lim_{n \to \infty} \sum_{R(j),\ 0 \leqslant j \leqslant n} a_j \right) + \left(\lim_{n \to \infty} \sum_{R(j),\ -n \leqslant j < 0} a_j \right) \tag{3}$$

假定两个极限都存在时。如果有一个或者两个极限不存在，即无限和是“发散”的，则（2）不存在。

如果在 Σ 符号下边放上两个或多个的条件，象在（3）中那样，则我们指的是所有的条件都必须成立。

关于求和，有四个简单的代数运算是非常重要的，而且只有熟悉了这些变换才有可能来求解许多问题。我们现在就讨论这四个运算。

a）对于和数之积的分配律：

$$\left(\sum_{R(i)} a_i \right) \left(\sum_{S(j)} b_j \right) = \sum_{R(i)} \left(\sum_{S(j)} a_i b_j \right) \tag{4}$$

例如，考虑特殊情况

$$\left(\sum_{1 \leqslant i \leqslant 2} a_i \right) \left(\sum_{1 \leqslant j \leqslant 3} b_j \right) = (a_1 + a_2)(b_1 + b_2 + b_3)$$

$$= (a_1 b_1 + a_1 b_2 + a_1 b_3) + (a_2 b_1 + a_2 b_2 + a_2 b_3)$$

$$= \sum_{1 \leqslant i \leqslant 2} \left(\sum_{1 \leqslant j \leqslant 3} a_i b_j \right)$$

习惯上常把(4)右边的圆括号去掉，"多重求和" $\Sigma_{R(i)}(\Sigma_{S(i)}a_{ij})$ 就简单地写成 $\Sigma_{R(i)}\Sigma_{S(i)}a_{ij}$。

b）改变变量：

$$\sum_{R(i)} a_i = \sum_{R(j)} a_j = \sum_{R(p(j))} a_{p(j)} \tag{5}$$

这个等式表示了两种类型的变换。在头一种情况下，我们只不过是改变下标变量的名称。第二种情况有点意思。这里 $p(j)$ 是 j 的一个函数，它表示该变程的一个排列，即是说，对于满足关系 $R(j)$ 的每一个整数 j，必然恰有一个满足关系 $R(p(j))$ 的整数 j，且反之亦然。在 $p(j)=c+j$ 或 $p(j)=c-j$ 的重要情况下，这个条件总是满足的，其中 c 是一个不依赖于 j 的整数，这两者都是在应用中最常使用的情况。例如，

$$\sum_{1\leqslant j\leqslant n} a_j = \sum_{1\leqslant j-1\leqslant n} a_{j-1} = \sum_{2\leqslant j\leqslant n+1} a_{j-1} \tag{6}$$

读者应当仔细地研究这些例子。

对于所有的无限求和，不得以 $p(j)$ 来替代 j。如果和前面一样，$p(j)=c\pm j$，则运算还总是正确的。但是在其它某些情况下，就必须小心在意〔例如，请参看汤·迈·阿波斯托尔（T. M. Apostol）著《数学分析》(Mathematical Analysis)，马萨诸塞州雷丁：爱迪生-韦斯利，1957 年)，第 12 章。为保证（5）对于整数的任何排列 $p(j)$ 的正确性，一个充分条件是 $\Sigma_{R(j)}|a_j|$ 存在〕。

c）交换求和的次序：

$$\sum_{R(i)}\sum_{S(j)} a_{ij} = \sum_{S(j)}\sum_{R(i)} a_{ij} \tag{7}$$

让我们考虑这个等式的一个非常简单的情况：

$$\sum_{R(i)}\sum_{1\leqslant j\leqslant 2} a_{ij} = \sum_{R(i)} (a_{i1}+a_{i2})$$

$$\sum_{1\leqslant j\leqslant 2}\sum_{R(i)} a_{ij} = \sum_{R(i)} a_{i1} + \sum_{R(i)} a_{i2}$$

由等式（7），这两者是相等的；这无非是说

$$\sum_{R(i)} (b_i+c_i) = \sum_{R(i)} b_i + \sum_{R(i)} c_i \tag{8}$$

这里我们设

$$b_i = a_{i1} \quad 和 \quad c_i = a_{i2}$$

交换求和次序的运算是极其有用的。因为经常会出现这样一种情况，就是我们只知道 $\Sigma_{R(i)}a_{ij}$ 的一个简单形式，但却不知道 $\Sigma_{S(j)}a_{ij}$ 的。在一种更为一般的情况下，即关系 $S(j)$ 既依赖于 i 也依赖于 j 时，我们也经常需要来交换求和的次序。在这种情况下，我们可用 "$S(i,j)$" 来表示这个关系。求和的交换，至少在理论说来，总可以如下来进行：

$$\sum_{R(i)}\sum_{S(i,j)} a_{ij} = \sum_{S'(j)}\sum_{R'(i,j)} a_{ij} \tag{9}$$

其中$S'(j)$是关系“有一个整数i，使得$R(i)$和$S(i,j)$两者都为真”，而$R'(i,j)$是关系“$R(i)$和$S(i,j)$两者都为真”。例如，如果求和是$\sum_{1\leqslant i\leqslant n}\sum_{1\leqslant j\leqslant i}a_{ij}$，则$S'(j)$是关系“有一个整数$i$，使得$1\leqslant i\leqslant n$且$1\leqslant j\leqslant i$”，即是$1\leqslant j\leqslant n$；而$R'(i,j)$是关系“$1\leqslant i\leqslant n$且$1\leqslant j\leqslant i$”，即是$j\leqslant i\leqslant n$。于是，

$$\sum_{1\leqslant i\leqslant n}\sum_{1\leqslant j\leqslant i}a_{ij}=\sum_{1\leqslant j\leqslant n}\sum_{j\leqslant i\leqslant n}a_{ij} \tag{10}$$

〔注意：也象在情况（b）中一样，交换求和次序的运算对于无穷级数并不总是正确的。如果这个级数是“绝对收敛”的，即如果$\sum_{R(i)}\sum_{S(j)}|a_{ij}|$存在，则能够证明等式（7）和等式（9）都正确。而且，如果$R(i)$或$S(j)$之一确定等式（7）中的一个有限和，而且如果出现的每一个无穷和收敛，则交换是正当的；特别地，对于收敛的无穷和，等式（8）总是正确的〕。

d）处理作用域。如果$R(j)$和$S(j)$是两个关系，我们有

$$\sum_{R(j)}a_j+\sum_{S(j)}a_j=\sum_{R(j)\text{或}S(j)}a_j+\sum_{R(j)\text{且}S(j)}a_j \tag{11}$$

例如，假定$m\leqslant n$，则

$$\sum_{1\leqslant j\leqslant m}a_j+\sum_{m\leqslant j\leqslant n}a_j=\left(\sum_{1\leqslant j\leqslant n}a_j\right)+a_m \tag{12}$$

在这种情况下，“$R(j)$且$S(j)$”简单地变成“$j=m$”，所以我们就把第二个求和归结成简单的“a_m”。在等式（11）的大多数应用中，或者是$R(j)$和$S(j)$仅对一个或两个j值同时地被满足，或者否则，不可能有同一个j使$R(j)$和$S(j)$两者都成立。在后一种情况下，等式（11）右边的第二个求和就干脆不出现了。

现在，我们已经给出了四个处理求和的基本规则。下面，让我们更进一步来说明怎样应用这些技术。

例 1.

$$\sum_{0\leqslant j\leqslant n}a_j=\sum_{\substack{0\leqslant j\leqslant n\\ j\text{偶}}}a_j+\sum_{\substack{0\leqslant j\leqslant n\\ j\text{奇}}}a_j \qquad \text{由规则（d）}$$

$$=\sum_{\substack{0\leqslant 2j\leqslant n\\ 2j\text{偶}}}a_{2j}+\sum_{\substack{0\leqslant 2j+1\leqslant n\\ 2j+1\text{奇}}}a_{2j+1} \qquad \text{由规则（b）}$$

$$=\sum_{0\leqslant j\leqslant n/2}a_{2j}+\sum_{0\leqslant j<n/2}a_{2j+1}$$

这最后一步只不过就是简化了Σ之下的关系。

例 2. 设

$$S_1=\sum_{0\leqslant i\leqslant n}\sum_{0\leqslant j\leqslant i}a_ia_j=\sum_{0\leqslant j\leqslant n}\sum_{j\leqslant i\leqslant n}a_ia_j \qquad \text{由规则（c）〔参照等式(10)〕}$$

$$= \sum_{0\leqslant i\leqslant n} \sum_{i\leqslant j\leqslant n} a_i a_j \qquad \text{由规则（b）}$$

只要交换 i 和 j 的名称，并认清 $a_j a_i = a_i a_j$。如果我们以 S_2 来表示后一个求和，就有

$$2S_1 = S_1 + S_2 = \sum_{0\leqslant i\leqslant n} \left(\sum_{0\leqslant j\leqslant i} a_i a_j + \sum_{i\leqslant j\leqslant n} a_i a_j \right) \qquad \text{由等式（8）}$$

$$= \sum_{0\leqslant i\leqslant n} \left(\left(\sum_{0\leqslant j\leqslant n} a_i a_j \right) + a_i a_i \right) \qquad \text{由规则（d）〔参照等式(12)〕}$$

$$= \sum_{0\leqslant i\leqslant n} \sum_{0\leqslant j\leqslant n} a_i a_j + \sum_{0\leqslant i\leqslant n} a_i a_i \qquad \text{由等式（8）}$$

$$= \left(\sum_{0\leqslant i\leqslant n} a_i \right) \left(\sum_{0\leqslant j\leqslant n} a_j \right) + \left(\sum_{0\leqslant i\leqslant n} a_i^2 \right) \qquad \text{由规则（a）}$$

$$= \left(\sum_{0\leqslant i\leqslant n} a_i \right)^2 + \left(\sum_{0\leqslant i\leqslant n} a_i^2 \right) \qquad \text{由规则（b）}$$

这样，我们就推出了重要的恒等式

$$\sum_{0\leqslant i\leqslant n} \sum_{0\leqslant j\leqslant i} a_i a_j = \frac{1}{2} \left(\left(\sum_{0\leqslant i\leqslant n} a_i \right)^2 + \left(\sum_{0\leqslant i\leqslant n} a_i^2 \right) \right) \qquad (13)$$

例 3. 几何级数的和。假定 $x \neq 1$，$n \geqslant 0$。则

$$a + ax + \cdots + ax^n = \sum_{0\leqslant j\leqslant n} ax^j \qquad \text{由定义（3）}$$

$$= a + \sum_{1\leqslant j\leqslant n} ax^j \qquad \text{由规则（d）}$$

$$= a + x \sum_{1\leqslant j\leqslant n} ax^{j-1} \qquad \text{由（a）的一个非常特殊的情况}$$

$$= a + x \sum_{0\leqslant j\leqslant n-1} ax^j \qquad \text{由规则（b）〔参照等式（6）〕}$$

$$= a + x \sum_{0\leqslant j\leqslant n} ax^j - ax^{n+1} \qquad \text{由规则（d）}$$

比较一下第一和第五个关系式，我们就有

$$(1 - x) \sum_{0\leqslant j\leqslant n} ax^j = a - ax^{n+1}$$

这样，我们就得到了基本的公式

$$\sum_{0\leqslant j\leqslant n} ax^j = a \left(\frac{1 - x^{n+1}}{1 - x} \right) \qquad (14)$$

例 4. 算术级数的和。假定 $n \geqslant 0$。则

$$
\begin{aligned}
& a+(a+b)+\cdots+(a+nb) \\
&= \sum_{0\leqslant j\leqslant n}(a+bj) && \text{由定义(2)} \\
&= \sum_{0\leqslant n-j\leqslant n}(a+b(n-j)) && \text{由规则(b)} \\
&= \sum_{0\leqslant j\leqslant n}(a+bn-bj) && \text{通过化简} \\
&= \sum_{0\leqslant j\leqslant n}(2a+bn)-\sum_{0\leqslant j\leqslant n}(a+bj) && \text{由等式(8)} \\
&= (n+1)(2a+bn)-\sum_{0\leqslant j\leqslant n}(a+bj)
\end{aligned}
$$

因为第一个和数不过就是 $(n+1)$ 个不依赖于 j 的项之和数。现在通过第一个和第五个表达式之等式，除之以 2，我们就得到

$$\sum_{0\leqslant j\leqslant n}(a+bj)=a(n+1)+\frac{1}{2}bn(n+1) \tag{15}$$

注意我们已经仅仅通过使用和数的简单处理，而得到了重要的等式 (13)，(14) 和 (15)。大多数的教科书可能都会简单地叙述这些公式，并用归纳法来证明它们。当然，这是完全正确的步骤，只是并没有给出其来龙去脉，人们起初究竟是怎样臆想出这些公式来的，或者只是某人侥幸地猜中的。在分析算法时，我们面临着数以百计的和数，而且它们并不与任何明显的型式相吻合，通过如上的对这些和数的处理，我们无须进行机敏的猜测，通常就能得到答案。

类似于我们对于和数的记号，有一个对于乘积的记号：

$$\prod_{R(j)} a_j \tag{16}$$

表示所有那样的 a_j 的乘积，对于这些 a_j，整数 j 满足 $R(j)$。如果不存在这样的整数 j，就把乘积定义成取值 1（不是 0）。无穷乘积的问题在习题 21 中考虑。

运算 (b)，(c) 和 (d) 对于 Π 记号，也如同对于 Σ 记号一样，都是成立的，只不过有些适当的简单修改。这一小节末尾的习题给出了使用乘积记号的一些例子。

在结束这一小节之前，我们还要提出对于多重求和的另一个记号，这一记号常常是很方便的：单个 Σ 记号可以用于若干个下标变量的一个或多个关系，来表示这个和是对满足这些条件的所有组合来取的。例如，

$$\sum_{0\leqslant i\leqslant n}\sum_{0\leqslant j\leqslant n}a_{ij}=\sum_{0\leqslant i,\ j\leqslant n}a_{ij};\qquad \sum_{0\leqslant i\leqslant n}\sum_{0\leqslant j\leqslant i}a_{ij}=\sum_{0\leqslant j\leqslant i\leqslant n}a_{ij}$$

为了说明这一记号的有用性，我们再举个进一步的例子。这就是

$$\sum_{\substack{j_1+\cdots+j_n=n\\ j_1\geqslant\cdots\geqslant j_n\geqslant 0}} a_{j_1\cdots j_n}$$

其中 a 是一个 n 元下标变量。例如，如果 $n=5$，则这一记号表示

$$a_{11111}+a_{21110}+a_{22100}+a_{31100}+a_{32000}+a_{41000}+a_{50000}$$

（请参看 1.2.1 小节中关于一个数的分划的讨论。）

习题——第一组

1.〔01〕若 $n=3.14$，则记号（1）的意思是什么？

2.〔10〕不使用 Σ 记号，分别写出与

$$\sum_{0\leqslant n\leqslant 5}\frac{1}{2n+1}$$

和

$$\sum_{0\leqslant n^2\leqslant 5}\frac{1}{2n^2+1}$$

相等的式子。

▶3.〔13〕尽管有规则（b），说明上题的两个结果却是不同的。

4.〔10〕不用 Σ 记号，对于 $n=3$ 的情况把等式（10）两边的等价式写成一些和数之和。

▶5.〔$HM20$〕证明对于任何无穷级数，规则（a）成立。

6.〔$HM20$〕证明对于任何无穷级数，规则（d）成立。

7.〔$HM23$〕假定 c 为一整数，证明 $\Sigma_{R(j)}a_j=\Sigma_{R(c-j)}a_{c-j}$，甚至当两边都是无穷级数时也成立。

8.〔$HM25$〕举出一个使等式（7）为假的无穷级数的例子。

▶9.〔05〕如果 $n=-1$，等式（14）的推导是否成立？

10.〔05〕如果 $n=-2$，等式（14）的推导是否成立？

11.〔08〕如果 $x=1$，则等式（14）右边应当是什么？

12.〔10〕$1+\frac{1}{7}+\frac{1}{49}+\frac{1}{343}+\cdots+\left(\frac{1}{7}\right)^n$ 等于什么？

13.〔10〕利用等式（15）并假定 $m\leqslant n$，计算 $\Sigma_{m\leqslant j\leqslant n}j$。

14.〔15〕用上题的结果，计算 $\Sigma_{m\leqslant j\leqslant n}\Sigma_{r\leqslant k\leqslant s}jk$。

▶15.〔$M22$〕对于小的 n 值计算和数 $1\times 2+2\times 2^2+3\times 2^3+\cdots+n2^n$ 你是否已看出这些数所展示的型式？如未看出，通过类似于导出等式（14）的处理，来发现它。

16.〔$M22$〕证明：如果 $x\neq 1$，则

$$\sum_{0\leqslant j\leqslant n}jx^j=\frac{nx^{n+2}-(n+1)x^{n+1}+x}{(x-1)^2}$$

请不要使用数学归纳法来证。

▶17.〔*M*00〕设S是整数的一个集合，问Σ_j在S中1等于什么？

18.〔*M*20〕假定$R(i)$是关系“n是i的倍数”而$S(i,j)$是关系“$1\leqslant j<i$”，说明如何象等式（9）一样交换求和的次序。

19.〔20〕$\Sigma_{m\leqslant j\leqslant n}(a_j-a_{j-1})$等于什么？

▶20.〔25〕欧·乔·马特里克（I. J. Matrix）博士看出了一个值得注意的公式序列：

$9\times 1+2=11$，$9\times 12+3=111$，$9\times 123+4=1111$，$9\times 1234+5=11111$

a）借助于Σ记号写出他的这个很有趣的发现。

b）你对于题（a）的答案毫无疑问地要以数10作为十进制系统的基底。把这个公式加以推广，以便得到一个也许将适用于任何基底的公式。

c）使用正文中推出的公式或上边的习题16来证明习题（b）中的公式。

21.〔*M*25〕给出一个既同等式（3），也同高等微积分中的标准数学约定相容的无穷乘积的定义。

▶22.〔20〕关于和数的等式（5），（7），（8）和（11），试指出对应的关于乘积的类似的等式。

23.〔10〕说明，当没有整数满足$R(j)$时，为什么把$\Sigma_{R(j)}a_j$和$\Pi_{R(j)}a_j$分别定义成0和1，才是一个好办法？

24.〔20〕假设$R(j)$仅对有限多个j为真，试对于满足$R(j)$的整数个数用归纳法，证明$\log_b\Pi_{R(j)}a_j=\Sigma_{R(j)}(\log_b a_j)$，假定所有$a_j>0$。

▶25.〔15〕考虑下面的推导，有无什么差错？

$$\left(\sum_{1\leqslant i\leqslant n}a_i\right)\left(\sum_{1\leqslant j\leqslant n}\frac{1}{a_j}\right)=\sum_{1\leqslant i\leqslant n}\sum_{1\leqslant j\leqslant n}\frac{a_i}{a_j}=\sum_{1\leqslant i\leqslant n}\sum_{1\leqslant i\leqslant n}\frac{a_i}{a_i}=\sum_{1\leqslant i\leqslant n}1=n$$

26.〔25〕通过习题22所述的对Π记号的处理，证明$\Pi_{0\leqslant i\leqslant n}\Pi_{0\leqslant j\leqslant i}a_ia_j$可以借助于$\Pi_{0\leqslant i\leqslant n}a_i$来表达。

27.〔*M*20〕通过证明：假定$0<a_j<1$，则

$$\prod_{1\leqslant j\leqslant n}(1-a_j)\geqslant 1-\sum_{1\leqslant j\leqslant n}a_j$$

来推广1.2.1小节的习题9的结果。

28.〔*M*22〕求出一个$\Pi_{2\leqslant j\leqslant n}(1-1/j^2)$的简单公式。

▶29.〔*M*30〕（a）借助于本小节末尾所说明的多重求和记号，来表达$\Sigma_{0\leqslant i\leqslant n}\Sigma_{0\leqslant j\leqslant i}\Sigma_{0\leqslant k\leqslant j}a_ia_ja_k$。（b）借助于$\Sigma_{0\leqslant i\leqslant n}a_i$，$\Sigma_{0\leqslant i\leqslant n}a_i^2$和$\Sigma_{0\leqslant i\leqslant n}a_i^3$来表达同一个和〔参照等式(13)〕。

30.〔*M*23〕不用归纳法，来证明“拉格朗日恒等式”

$$\left(\sum_{1\leqslant j\leqslant n}a_jb_j\right)^2=\left(\sum_{1\leqslant j\leqslant n}a_j^2\right)\left(\sum_{1\leqslant j\leqslant n}b_j^2\right)-\sum_{1\leqslant k<j\leqslant n}(a_kb_j-a_jb_k)^2$$

▶31.〔*M*23〕证明$\Sigma_{1\leqslant j<k\leqslant n}(a_j-a_k)(b_j-b_k)$可以借助于$\Sigma_{1\leqslant j\leqslant n}a_jb_j$，$\Sigma_{1\leqslant j\leqslant n}a_j$和$\Sigma_{1\leqslant i\leqslant n}b_j$来表达。不要用归纳法。

32.〔*M*20〕证明

$$\prod_{1\leqslant j\leqslant n}\sum_{1\leqslant i\leqslant m}a_{ij}=\sum_{1\leqslant i_1,\cdots,i_n\leqslant m}a_{i_1 1}\cdots a_{i_n n}$$

▶33.〔$M30$〕一个傍晚，马特里克博士发现了一些公式，这些公式甚至可以归入比习题20中的那些更为值得注意的类型中去：

$$\frac{1}{(a-b)(a-c)}+\frac{1}{(b-a)(b-c)}+\frac{1}{(c-a)(c-b)}=0$$

$$\frac{a}{(a-b)(a-c)}+\frac{b}{(b-a)(b-c)}+\frac{c}{(c-a)(c-b)}=0$$

$$\frac{a^2}{(a-b)(a-c)}+\frac{b^2}{(b-a)(b-c)}+\frac{c^2}{(c-a)(c-b)}=1$$

$$\frac{a^3}{(a-b)(a-c)}+\frac{b^3}{(b-a)(b-c)}+\frac{c^3}{(c-a)(c-b)}=a+b+c$$

证明这些公式是一般定律的一个特殊情况；设 x_1，x_2，…，x_n 是不同的数，证明

$$\sum_{1\leqslant j\leqslant n}\left(x_j^r/\prod_{\substack{1\leqslant k\leqslant n,\\ k\neq j}}(x_j-x_k)\right)=\begin{cases}0, & \text{如果 } 0\leqslant r<n-1\\ 1, & \text{如果 } r=n-1\\ \sum_{1\leqslant j\leqslant n}x_j, & \text{如果 } r=n\end{cases}$$

34.〔$M25$〕证明

$$\sum_{1\leqslant k\leqslant n}\frac{\prod_{1\leqslant r\leqslant n,\ r\neq m}(x-k-r)}{\prod_{1\leqslant r\leqslant m,\ r\neq k}(k-r)}=1$$

假定 $1\leqslant m\leqslant n$ 且 x 是任意的。例如，如果 $n=4$ 和 $m=2$，则

$$\frac{x(x-2)(x-3)}{(-1)(-2)(-3)}+\frac{(x+1)(x-1)(x-2)}{(1)(-1)(-2)}+\frac{(x-2)x(x-1)}{(2)(1)(-1)}+\frac{(x+3)(x+1)x}{(3)(2)(1)}=1$$

35.〔$HM20$〕记号 $\sup_{R(j)}a_j$ 以完全类似于 Σ 和 Π 记号的方式，用来表示元素 a_j 的最小上界（当仅有有限多个的 j 满足 $R(j)$ 时，通常才用记号 $\max_{R(j)}a_j$ 来表示该同一数量）。试说明规则（a），（b），（c）和（d）怎样才可以适合于对这个记号的处理。特别是，讨论以下的类似于规则（a）的规则：

$$\left(\sup_{R(i)}a_i\right)+\left(\sup_{S(j)}b_j\right)=\sup_{R(i)}\left(\sup_{S(j)}(a_i+b_j)\right)$$

而且当没有 j 满足 $R(j)$ 时，给出这个记号的适当定义。

习题——第二组

行列式和矩阵。下列有趣的问题是提供给那些至少对于行列式和初等矩阵理论已经入门并有些经验的读者的。一个行列式可以通过灵活地组合这几种运算来计算：（a）从一行或一列分解出一个公因子量；（b）把一行（或一列）的倍数加到另一行（或一列）上；（c）按余因子展开。运算（c）的最简单最常用的形式是：假定左上角的元素是 1，而整个头一行或整个头一列的其余元素皆为 0，则简单地删去整个头一行和头一列，然后计

算所得的较小的行列式。一般说来，$n \times n$ 行列式中一个元素 a_{ij} 的余因子是 $(-1)^{i+j}$ 乘以由删去 a_{ij} 所在的那一行和那一列而得到的 $(n-1)\times(n-1)$ 行列式。一个行列式的值等于 $\Sigma a_{ij}\cdot \mathrm{cofactor}(a_{ij})$●，其中求和是这样进行的：或者 i 或者 j 中的一个保持不变，而另一个下标则从 1 变到 n。

如果 (b_{ij}) 是矩阵 (a_{ij}) 的逆，则 b_{ij} 等于 a_{ji} 的余因子（注意，不是 a_{ij}），除以整个原矩阵的行列式。记号 δ_{ij} 表示值 1，当 $i=j$ 时，否则为 0。

下列类型的矩阵有特殊的重要性：

范特蒙德矩阵

$$a_{ij}=x_j^i$$

$$\begin{vmatrix} x_1 & x_2 & \cdots & x_n \\ x_1^2 & x_2^2 & \cdots & x_n^2 \\ \vdots & & & \vdots \\ x_1^n & x_2^n & \cdots & x_n^n \end{vmatrix}$$

组合矩阵

$$a_{ij}=y+\delta_{ij}x$$

$$\begin{vmatrix} x+y & y & \cdots & y \\ y & x+y & \cdots & y \\ \vdots & & & \vdots \\ y & y & \cdots & x+y \end{vmatrix}$$

柯西矩阵

$$a_{ij}=1/(x_i+y_j)$$

$$\begin{vmatrix} 1/(x_1+y_1) & 1/(x_1+y_2) & \cdots & 1/(x_1+y_n) \\ 1/(x_2+y_1) & 1/(x_2+y_2) & \cdots & 1/(x_2+y_n) \\ \vdots & & & \vdots \\ 1/(x_n+y_1) & 1/(x_n+y_2) & \cdots & 1/(x_n+y_n) \end{vmatrix}$$

36.〔*M*23〕证明组合矩阵的行列式是 $x^{n-1}(x+ny)$。

▶37.〔*M*24〕证明范特蒙德矩阵的行列式是

$$\prod_{1\leqslant j\leqslant n} x_j \prod_{1\leqslant i<j\leqslant n}(x_j-x_i)$$

▶38.〔*M*25〕证明柯西矩阵的行列式是

$$\prod_{1\leqslant i<j\leqslant n}(x_j-x_i)(y_j-y_i)/\prod_{1\leqslant i,\ j\leqslant n}(x_i+y_j)$$

39.〔*M*23〕证明组合矩阵的逆由 $b_{ij}=(-y+\delta_{ij}(x+ny))/x(x+ny)$ 给出。

40.〔*M*24〕证明范特蒙德矩阵的逆由

$$b_{ij}=(-1)^{j+1}\sum_{\substack{1\leqslant k_1<\cdots<k_{n-j}\leqslant n \\ k_1,\ \cdots,\ k_{n-j}\neq i}}(x_{k_1}x_{k_2}\cdots x_{k_{n-j}})/x_i\prod_{\substack{1\leqslant k\leqslant n \\ k\neq i}}(x_k-x_i)$$

给出。不要为分子中复杂的求和所吓倒——它只不过就是多项式 $(x_1-x)\cdots(x_n-x)/(x_i-x)$

● $\mathrm{cofactor}(a_{ij})$ 表示 a_{ij} 的余因子。——译注

中 x^{j-1} 的系数。

41.〔$M26$〕证明柯西矩阵的逆由

$$b_{ij}=\left(\prod_{1\leq k\leq n}(x_j+y_k)(x_k+y_i)\right)\Big/(x_j+y_i)\left(\prod_{\substack{1\leq k\leq n\\ k\neq j}}(x_j-x_k)\right)\left(\prod_{\substack{1\leq k\leq n\\ k\neq i}}(y_i-y_k)\right)$$

给出。

42.〔$M18$〕组合矩阵的逆中全部 n^2 个元素之和是什么？

43.〔$M24$〕范特蒙德矩阵的逆中全部 n^2 个元素之和是什么？〔提示：用习题 33。〕

▶44.〔$M26$〕柯西矩阵的逆中全部 n^2 个元素之和是什么？

▶45.〔$M25$〕希尔伯特矩阵，有时叫做“(无穷的)希尔伯特矩阵的 $n\times n$ 段”，是一个 $a_{ij}=1/(i+j-1)$ 的矩阵。证明这是柯西矩阵的一个特殊情况，试求其逆，证明其逆的每个元素都是一个整数，并证明逆的全部元素之和是 n^2（注意：希尔伯特矩阵经常用来检验各种矩阵处理的算法，因为它们在数值上是不稳定的，而且它们有已知的逆。然而，想把在本习题中的已知的逆同一个希尔伯特矩阵的计算出来的逆进行比较是错误的，因为有待求逆的矩阵必须事先表示成舍入的数；一个近似的希尔伯特矩阵的逆，由于出现了不稳定性，将稍微不同于精确的希尔伯特矩阵的逆。由于逆的元素都是整数，还由于逆矩阵恰和原来的矩阵一样不稳定，这个逆可以精确地加以描述，而且人们可以试图对这个逆来求逆；然而，出现在逆当中的整数是十分巨大的）。对于这个问题的解，需要有关于阶乘和二项式系数方面的初步知识。在 1.2.5 和 1.2.6 小节中要讨论这些知识。

▶46.〔$M30$〕设 A 是一个 $m\times n$ 矩阵，设 B 是一个 $n\times m$ 矩阵。假定 $1\leqslant j_1,j_2,\cdots,j_m\leqslant n$，命 $A_{j_1j_2\cdots j_m}$ 表示由 A 的 $j_1,\cdots,j_m$ 列组成的 $m\times m$ 矩阵，而 $B_{j_1j_2\cdots j_m}$ 表示由 B 的 $j_1,\cdots,j_m$ 行组成的 $m\times m$ 矩阵。证明

$$\det(AB)=\sum_{1\leqslant j_1<j_2<\cdots<j_m\leqslant n}\det(A_{j_1j_2\cdots j_m})\det(B_{j_1j_2\cdots j_m})$$

（注意特殊情况：(i) $m=n$，(ii) $m=1$，(iii) $B=A^T$）[1]。

1.2.4 整数函数和初等数论

如果 x 是任意实数，则我们写

$\lfloor x\rfloor$ =小于或等于 x 的最大整数（x 的低限或“地板”）；

$\lceil x\rceil$ =大于或等于 x 的最小整数（x 的高限或“天棚”）。

别的书上常用〔x〕表示这里的前一个或后一个函数，而通常是表示前一个。这里的记号是由肯·尤·艾弗森(K. E. Iverson)给出的是更有用的，因为这两个函数实际上经常几乎平等地出现。函数 $\lfloor x\rfloor$ 有时叫做 *entier* 函数，此词取自于法文的“整数”一词。

容易验证下列的公式和例子：

$$\lfloor\sqrt{2}\rfloor=1\qquad\lceil\sqrt{2}\rceil=2$$

[1] $\det(M)$ 表示方阵 M 的行列式。——译注

$$\left\lfloor +\frac{1}{2}\right\rfloor = 0 \qquad \left\lceil -\frac{1}{2}\right\rceil = 0 \qquad \left\lfloor -\frac{1}{2}\right\rfloor = -1 \text{（不是 0！）}$$

$\lceil x\rceil = \lfloor x\rfloor$，当且仅当，$x$ 为整数时，

$\lceil x\rceil = \lfloor x\rfloor + 1$，当且仅当，$x$ 不是整数时；

$\lfloor -x\rfloor = -\lceil x\rceil$；$x-1 < \lfloor x\rfloor \leqslant x \leqslant \lceil x\rceil < x+1$。

本小节结尾的习题列出了关于低限和高限运算的其它重要的公式。

如果 x 和 y 是任意实数．则我们定义以下的二元运算：

$$x \bmod y = x - y\lfloor x/y\rfloor \quad \text{若 } y \neq 0 \qquad x \bmod 0 = x \tag{1}$$

从这个定义，我们可以看出：当 $y \neq 0$ 时，

$$0 \leqslant \frac{x}{y} - \left\lfloor \frac{x}{y}\right\rfloor = \frac{x \bmod y}{y} < 1 \tag{2}$$

因此，

a）若 $y > 0$，则 $0 \leqslant x \bmod y < y$；

b）若 $y < 0$，则 $0 \geqslant x \bmod y > y$；

c）数量 $x - (x \bmod y)$ 是 y 的一个整倍数；而且因此我们可以把 $x \bmod y$ 想象作 x 除以 y 时的余数。

因此，当 x 和 y 是整数时，“mod”乃是一个熟知的运算：

$$\begin{aligned} 5 \bmod 3 &= 2 \\ 18 \bmod 3 &= 0 \\ -2 \bmod 3 &= 1 \end{aligned} \tag{3}$$

我们有：$x \bmod y = 0$ 当且仅当 x 是 y 的倍数，即是，当且仅当 x 可为 y 所整除．

当 x 和 y 取任意实数值时，“mod”运算也是有用的。例如，对于三角函数，我们可以写

$$\tan x = \tan(x \bmod \pi)$$

数量 $x \bmod 1$ 是 x 的“小数部分”；由等式（1）我们有

$$x = \lfloor x\rfloor + (x \bmod 1) \tag{4}$$

在数论中，“mod”这一缩写用于不同的但又都相互关联的意义。我们将用下列形式来表达数论的同余概念：

$$x \equiv y \pmod{z} \tag{5}$$

意味着 $x \bmod z = y \bmod z$，即是，差 $x - y$ 是 z 的整倍数。表达式（5）读作“x 与 y 模 z 同余”。

现在让我们来指出同余的基本的初等性质，这些性质将在本书的数论论述中用到。下列的公式中所有的变量均取整数值。说两个整数是互质的，如果它们无公因子，即是说，如果它们的最大公因子为 1。互质整数的概念也是一个熟知的概念，因为当分子与分母互质时，习惯上就说一个分数是最简的。

定律A．如果 $a \equiv b$ 和 $x \equiv y$，则 $a \pm x \equiv b \pm y$ 且 $ax \equiv by \pmod{m}$。

定律B．如果 $ax \equiv by$ 和 $a \equiv b$，且如果 a 与 m 互质，则 $x \equiv y \pmod{m}$。

定律C．$a \equiv b \pmod{m}$ 当且仅当 $an \equiv bn \pmod{mn}$。

定律D. 如果 r 与 s 互质，则 $a \equiv b \pmod{rs}$，当且仅当 $a \equiv b \pmod{r}$ 且 $a \equiv b \pmod{s}$。

定律A指出，我们可以进行 modulo m 的加法、减法和乘法（而且从而我们可以对 $n \geqslant 0$ 取幂 x^n），恰如我们进行通常的加法，减法，乘法和乘幂一样。定律B考虑了除法的运算，而且说明，在某些情况下（就是说，除数与模数互质），我们也能除去公因子。定律C和定律D考虑了模数改变时的关系。

作为这些关系的一个例子，我们来证明一个重要的定理。

定理F（费尔玛定理，1640）。如果 p 是一个质数，则 $a^p \equiv a \pmod{p}$。

证明. 如果 a 是 p 的倍数，则显然 $a^p \equiv 0 \equiv a \pmod{p}$。所以我们仅仅需要考虑 $a \bmod p \neq 0$ 的情况。因为 p 是质数，这意味着 a 与 p 互质。考虑数

$$0 \bmod p,\quad a \bmod p,\quad 2a \bmod p,\quad \cdots,\quad (p-1)a \bmod p \tag{6}$$

这 p 个数都是不相同的。因为倘若 $ax \bmod p = ay \bmod p$，则由定义（5），$ax \equiv ay \pmod{p}$，于是由定律B，将有 $x \equiv y \pmod{p}$。

既然（6）给出了 p 个不同的数，所有这些数还都是非负的且小于 p，故我们看出，头一个数为0而其余的数按某种次序为整数 $1, 2, \cdots, p-1$。因此由定律A，

$$(a)(2a)\cdots((p-1)a) \equiv 1\cdot 2\cdots(p-1) \pmod{p} \tag{7}$$

以 a 乘这同余式的两边，我们就得到

$$a^p(1\cdot 2\cdots(p-1)) \equiv a(1\cdot 2\cdots(p-1)) \pmod{p} \tag{8}$$

这就证明了定理，因为因子 $1, 2, \cdots, (p-1)$ 的每一个都与 p 互质，因而就可根据定律B将它们约去。∎

以下的习题17到21，提供了奠定初等数论之基础的基本定律。

习题

1. 〔00〕$\lfloor 1.1 \rfloor$，$\lfloor -1.1 \rfloor$，$\lceil -1.1 \rceil$，$\lfloor 0.99999 \rfloor$ 和 $\lfloor \lg 35 \rfloor$ 等于什么？

▶2. $\lceil \lfloor x \rfloor \rceil$ 等于什么？

3. 〔$M10$〕设 n 是整数，且设 x 是一个实数。证明

a）$\lfloor x \rfloor < n$ 当且仅当 $x < n$；　b）$n \leqslant \lfloor x \rfloor$ 当且仅当 $n \leqslant x$；

c）$\lceil x \rceil \leqslant n$ 当且仅当 $x \leqslant n$；　d）$n < \lceil x \rceil$ 当且仅当 $n < x$；

e）$\lfloor x \rfloor = n$ 当且仅当 $x - 1 < n \leqslant x$，又当且仅当 $n \leqslant x < n + 1$；

f）$\lceil x \rceil = n$ 当且仅当 $x \leqslant n < x + 1$，又当且仅当 $n - 1 < x \leqslant n$。

〔这些公式是为证明关于 $\lfloor x \rfloor$ 和 $\lceil x \rceil$ 的命题的最重要的工具。〕

▶4. 〔$M10$〕利用上题的结果，证明 $\lfloor -x \rfloor = -\lceil x \rceil$。

5. 〔16〕假定 x 是正实数，试指出一个表达“x 舍入成最近整数”的简单公式。所希望的舍入规则是当 $x \bmod 1 < \frac{1}{2}$ 时产生 $\lfloor x \rfloor$，而当 $x \bmod 1 \geqslant \frac{1}{2}$ 时产生 $\lceil x \rceil$。你的答案应当是概括这两种情况的一个统一的公式。试讨论当 x 是负数时由你的公式所得到的舍入。

▶6. 〔20〕对于所有的正实数 x，下列等式那一个为真：

(a) $\lfloor\sqrt{\lfloor x\rfloor}\rfloor=\lfloor\sqrt{x}\rfloor$；(b) $\lceil\sqrt{\lceil x\rceil}\rceil=\lceil\sqrt{x}\rceil$；(c) $\lfloor\sqrt{\lceil x\rceil}\rfloor=\lceil\sqrt{x}\rceil$?

7. 〔*M*15〕证明 $\lfloor x\rfloor+\lfloor y\rfloor\leqslant\lfloor x+y\rfloor$，而且其中之等式当且仅当 $x \bmod 1+y \bmod 1<1$ 时成立。对于高限，类似的公式是否成立?

8. 〔00〕$100 \bmod 3$，$100 \bmod 7$，$-100 \bmod 7$，$-100 \bmod 0$ 等于什么?

9. 〔05〕$5 \bmod -3$，$18 \bmod -3$，$-2 \bmod -3$ 等于什么?

▶10. 〔10〕$1.1 \bmod 1$，$0.11 \bmod .1$，$0.11 \bmod -.1$ 等于什么?

11. 〔00〕对于我们的约定说来，"$x\equiv y\ (\text{modulo } 0)$"意味着什么?

12. 〔00〕什么整数与1互质?

13. 〔*M*00〕根据习惯，我们约定0和 n 的最大公因子为 $|n|$。问什么整数与0互质?

▶14. 〔12〕如果 $x \bmod 3=2$ 和 $x \bmod 5=3$，则 $x \bmod 15$ 等于什么?

15. 〔10〕证明 $z(x \bmod y)=(zx) \bmod (zy)$。(注意定律C是这个分配律的直接推论。)

16. 〔*M*10〕假定 $y>0$。证明：如果 $(x-z)/y$ 是一个整数且如果 $0\leqslant z<y$，则 $z=x \bmod y$。

17. 〔*M*15〕直接从同余的定义来证明定律A，并来证明定律D的一半：如果 $a\equiv b\ (\text{modulo } rs)$，则 $a\equiv b\ (\text{modulo } r)$ 和 $a\equiv b\ (\text{modulo } s)$。(这里 r，s 是任意整数。)

18. 〔*M*15〕用定律B，证明定律D的另一半：假定 r 与 s 互质；如果 $a\equiv b\ (\bmod r)$ 和 $a\equiv b\ (\bmod s)$，则 $a\equiv b\ (\bmod rs)$。

▶19. 〔*M*10〕(关于逆的定律)。如果 n 与 m 互质，则有一个整数 n' 使得 $nn' \bmod m=1$。请用推广了的欧几里得算法（算法1.2.1E）来证明这一点。

20. 〔*M*15〕用关于逆的定律和定律A来证明定律B。

21. 〔*M*22〕用定律B和习题1.2.1-5来证明：每个整数 $n>1$ 可以唯一地表示成质数的乘积（除了其因子的次序可有颠倒之外），即是，恰有一个方式把 n 写成为 $n=p_1p_2\cdots p_k$，其中每一个 p_j 是质数且 $p_1\leqslant p_2\leqslant\cdots\leqslant p_k$。

▶22. 〔*M*10〕举出一个例子来说明当 a 与 m 不互质时，定律B不总成立。

23. 〔*M*10〕举出一个例子来说明如果 r 与 s 不互质，则定律D不总成立。

▶24. 〔*M*20〕定律A，B，C和D能被推广到什么程度，以便可以应用到任何实数，而不仅是整数?

25. 〔*M*00〕根据定理F，证明：当 p 是一个质数时，如果 a 不是 p 的倍数则 $a^{p-1} \bmod p=1$，如果 a 是 p 的倍数则 $a^{p-1} \bmod p=0$。

26. 〔*M*15〕设 p 是一个奇质数，设 a 是任意整数，且设 $b=a^{(p-1)/2}$。证明 $b \bmod p$ 或等于0，或等于1，或等于 $p-1$〔提示：考虑 $(b+1)(b-1)$〕。

27. 〔*M*15〕假定 n 是一个正整数，命 $\varphi(n)$ 是 $0,1,\cdots,n-1$ 当中与 n 互质的数的个数。于是，$\varphi(1)=1$，$\varphi(2)=1$，$\varphi(3)=2$，$\varphi(4)=2$，等等。试证：如果 p 是一个质数，则 $\varphi(p)=p-1$；并请计算 $\varphi(p^e)$ 的值，其中 e 为一正整数。

▶28. 〔*M*25〕说明用以证明定理F的方法，也可以用来证明以下的（定理F的）推广，这个推广叫做欧拉定理：对于任意正整数 m，当 a 与 m 互质时，有 $a^{\varphi(m)} \bmod m=1$。(特别是，习题19中的数 n' 可以取作 $n^{\varphi(m)-1} \bmod m$.)。

29.〔*M*20〕正整数 n 的一个函数叫做乘性的，如果当 r 与 s 互质时 $f(rs)=f(r)f(s)$。证明以下的函数是乘性的：(a) $f(n)=n^k$；(b) $f(n)=0$ 当 n 可被某个整数 $k>1$ 的平方 k^2 所整除时，否则 $f(n)=1$；(c) $f(n)=c^k$，其中 k 是整除 n 的不同质数的个数；(d) 任意两个乘性函数的乘积。

30.〔*M*30〕证明 27 题的函数 $\varphi(n)$ 是乘性的。用这一事实，计算 $\varphi(1000000)$ 并且对于 n 已经分解成质数的情形，给出一个以简单的方式来计算 $\varphi(n)$ 的方法。

31.〔*M*22〕证明如果 $f(n)$ 是乘性的，则 $g(n)=\sum_{d\backslash n} f(d)$ 也是乘性的。$d\backslash n$ 这一记号的含意是“d 整除 n”，即，d 是一个正整数而且 $n \bmod d = 0$。

32.〔*M*18〕联系上题中的记号，证明对于任何函数 $f(x,y)$，有

$$\sum_{d\backslash n}\sum_{c\backslash d} f(c,d)=\sum_{c\backslash n}\sum_{d\backslash(n/c)} f(c,cd)$$

33.〔*M*18〕如果 n，m 是整数，计算

(a) $\left\lfloor \frac{n+m}{2} \right\rfloor + \left\lfloor \frac{n-m+1}{2} \right\rfloor$　　(b) $\left\lceil \frac{n+m}{2} \right\rceil + \left\lceil \frac{n-m+1}{2} \right\rceil$

(特殊情况 $m=0$ 是值得注意的)。

▶34.〔*M*21〕为保证对于所有实数 $x\geqslant 1$，都有 $\lfloor \log_b x \rfloor = \lfloor \log_b \lfloor x \rfloor \rfloor$，对于实数 $b>1$ 应加上什么条件才是必要和充分的？

▶35.〔*M*20〕假定 m，n 是整数且 $n>0$。证明对于所有实数 x，$\lfloor (x+m)/n \rfloor = \lfloor (\lfloor x \rfloor + m)/n \rfloor$（当 $m=0$ 时，我们有一个重要的特殊情况）。对于高限函数，类似的结果是否成立？

36.〔*M*23〕证明 $\sum_{1\leqslant k\leqslant n}\lfloor k/2 \rfloor = \lfloor n^2/4 \rfloor$ 并计算 $\sum_{1\leqslant k<n}\lceil k/2 \rceil$。

▶37.〔*M*30〕设 m，n 是整数，$n>0$。证明

$$\sum_{0\leqslant k<n}\left\lfloor \frac{mk+x}{n} \right\rfloor = \frac{(m-1)(n-1)}{2} + \frac{d-1}{2} + d\lfloor x/d \rfloor$$

其中 d 是 m 和 n 的最大公因子，而 x 是任意实数。

38.〔*M*22〕证明对于所有正整数 n 和任意实数 x，

$$\lfloor x \rfloor + \left\lfloor x+\frac{1}{n} \right\rfloor + \cdots + \left\lfloor x+\frac{n-1}{n} \right\rfloor = \lfloor nx \rfloor$$

在你的证明中不要使用习题 37 的结果。

39.〔*HM*35〕一个函数 f，如果对于它，当 n 是正整数时有

$$f(x)+f\left(x+\frac{1}{n}\right)+\cdots+f\left(x+\frac{n-1}{n}\right)=f(nx)$$

则叫做重迭函数。上题表明 $\lfloor x \rfloor$ 是重迭函数。证明下列函数也是重迭的：

a) $f(x)=x-\frac{1}{2}$；

b）$f(x)=1$ 当 x 为一整数时，$=0$ 否则；

c）$f(x)=1$ 当 x 为一正整数时，$=0$ 否则；

d）$f(x)=1$ 如果存在一个有理数 r 和一个整数 m 使得 $x=r\pi+m$，$=0$ 否则；

e）和（d）中的函数类似的三个其它的函数，其中 r 和/或 m 限制为正值；

f）$f(x)=\log|2\sin\pi x|$，如果允许 $f(x)=-\infty$；

g）任意两个重迭函数之和；

h）一个重迭函数乘以一个常数；

i）函数 $g(x)=f(x-\lfloor x\rfloor)$，其中 $f(x)$ 是重迭函数。

40.〔$HM46$〕研究重迭函数的类；确定特殊类型的所有重迭函数（例如，习题39（a）中的函数是不是仅有的连续的重迭函数？）。再来研究一下更一般的函数类，对于它，有

$$f(x)+\cdots+f\left(x+\frac{n-1}{n}\right)=a_n f(nx)+b_n$$

这里 a_n，b_n 是依赖于 n 但不依赖于 x 的数。这些函数的微商和（如果 $b_n=0$）积分都属同一类型。如果我们要求 $b_n=0$，则我们有，例如，贝努里多项式，三角函数 $\cot\pi x$ 和 $\csc^2\pi x$，以及胡尔维兹的广义从塔函数 $\zeta(s,x)=\sum_{k\geqslant 1}1/(k+x)^s$ 对于固定的 s。对于 $b_n\neq 0$，我们还有其它熟知的函数，例如普塞函数，关于这些函数进一步的性质，请参看路·乔·莫德尔著《算术特征的积分公式》（Integral Formulae of Arithmetical Character），载于《伦敦数学协会杂志》33(1958)，371～375。

41.〔$M23$〕设 $a_1,a_2,a_3,\cdots$ 是序列 1，2，2，3，3，3，4，4，4，4，…；借助于 n，求出对于 a_n 的表达式（使用低限和/或高限运算）。

42.〔$M24$〕（a）证明

$$\sum_{1\leqslant k\leqslant n}a_k=na_n-\sum_{1\leqslant k<n}k(a_{k+1}-a_k)\quad\text{当 } n>0 \text{ 时；}$$

（b）上边的公式，对于计算涉及低限函数的某些和数是有用的。证明，如果 b 是一个 $\geqslant 2$ 的整数，则

$$\sum_{1\leqslant k\leqslant n}\lfloor\log_b k\rfloor=(n+1)\lfloor\log_b n\rfloor-(b^{\lfloor\log_b n\rfloor+1}-b)/(b-1)$$

43.〔$M23$〕求 $\sum_{1\leqslant k\leqslant n}\lfloor\sqrt{k}\rfloor$。

44.〔$M24$〕证明当 b 和 n 是整数，$n\geqslant 0$ 和 $b\geqslant 2$ 时，有 $\sum_{k\geqslant 0}\sum_{1\leqslant j<b}\lfloor(n+jb^k)/b^{k+1}\rfloor=n$。当 $n<0$ 时，这个和数的值是什么？

▶45.〔$M28$〕习题 37 的结果是有些令人惊异的，因为它意味着

$$\sum_{0\leqslant k<n}\left\lfloor\frac{mk+x}{n}\right\rfloor=\sum_{0\leqslant k<m}\left\lfloor\frac{nk+x}{m}\right\rfloor$$

许多类似的公式都有这种“互反关系”（参照 3.3.3 小节）。证明对于任意函数 f，

$$\sum_{0\leqslant j<n}f\left(\left\lfloor\frac{mj}{n}\right\rfloor\right)=\sum_{0\leqslant r<m}\left\lfloor\frac{rn}{m}\right\rfloor(f(r-1)-f(r))+nf(m-1)$$

特别地，证明

$$\sum_{0\leqslant j<n}\binom{\lfloor mj/n\rfloor+1}{k}+\sum_{0\leqslant j<m}\left\lceil\frac{jn}{m}\right\rceil\binom{j}{k-1}=n\binom{m}{k}$$

〔提示：考虑变量的变化，$r=\lfloor mj/n\rfloor$。二项式系数$\binom{m}{k}$在 1.2.6 小节中讨论。〕

46.〔*M*29〕（一般的互反定律）推广习题 45 的公式以得到关于$\sum_{0\leqslant j<\alpha n}f(\lfloor mj/n\rfloor)$的公式，其中$\alpha$是任意正实数。

47.〔*M*31〕 当p是奇质数时，定义勒让德符号$\left(\frac{q}{p}\right)$如下（参照习题 26）：

$$\left(\frac{q}{p}\right)=\begin{cases}+1 & \text{当 } q^{(p-1)/2}\bmod p=1\text{ 时}\\ 0 & =0\text{ 时}\\ -1 & =p-1\text{ 时}\end{cases}$$

a）假定q不是p的倍数，证明诸数

$$(-1)^{\lfloor 2kq/p\rfloor}(2kq \bmod p),\quad 0<k<p/2$$

以某种次序与诸数 2，4，…，$p-1$同余$(\bmod p)$。因此$\left(\frac{q}{p}\right)=(-1)^{\sigma}$，其中$\sigma=\sum_{0\leqslant k<p/2}\lfloor 2kq/p\rfloor$。

b）用（a）的结果来计算$\left(\frac{2}{p}\right)$。

c）假定q是奇数，证明$\sum_{0\leqslant k<p/2}\lfloor 2kq/p\rfloor\equiv\sum_{0\leqslant k<p/2}\lfloor kq/p\rfloor\ (\text{modulo } 2)$〔提示：考虑$\lfloor(p-1-2k)q/p\rfloor$〕。

d）用习题 46 中的一般的互反定律来得到二次互反定律：假定p和q是不同的奇质数，则$\left(\frac{q}{p}\right)\left(\frac{p}{q}\right)=(-1)^{(p-1)(q-1)/4}$。

48.〔*M*26〕 证明或反驳下列的恒等式：对于给定的整数m和n，有

（a） $\left\lfloor\frac{m+n-1}{n}\right\rfloor=\left\lceil\frac{m}{n}\right\rceil$；（b）$\left\lfloor\frac{n+2-\lfloor n/25\rfloor}{3}\right\rfloor=\left\lfloor\frac{8n+24}{25}\right\rfloor$。

1.2.5 排列和阶乘

n个对象的排列，就是把n个不同的对象放在一行上的一种安排。对于三个对象a、b、c，有六个排列：

$$a\ b\ c,\quad a\ c\ b,\quad b\ a\ c,\quad b\ c\ a,\quad c\ a\ b,\quad c\ b\ a。\qquad(1)$$

在算法分析中，排列的性质有着巨大的重要性，而且在本书的稍后部分，将推演出关于它们的许多有趣的问题。这里，我们将简单地计算它们，即将确定n个对象的排列有多少种可能：有n种方式来选择最左边的对象，而一旦完成了这一选择，就可以有$(n-1)$种方式把一个不同的对象选放在下一位置上；这样，对于头两个位置，就给我们提供了$n(n-1)$种选择。类似地，我们发现对于不同于头两个对象的第三个对象，有$(n-2)$种选择；而且，因此为了选择头三个对象，总共就有$n(n-1)(n-2)$种可能的方式。

一般说来，如果p_{nk}表示从n个当中选择k个对象并把它们排成一行的方式的种数，则我们看到

$$p_{nk}=n(n-1)\cdots(n-k+1)\qquad(2)$$

排列的总数是 $p_{nn}=n(n-1)\cdots(1)$

以一种归纳法的方式：假定 $n-1$ 个对象的所有排列已经构造好了，而来构造 n 个对象的所有排列的过程，在我们的应用当中是很重要的。让我们用数字 1，2，3 来代替字母 a，b，c 而把(1)重写一下，"阶"3 的排列是

$$1\ 2\ 3,\ 1\ 3\ 2,\ 2\ 1\ 3.\ 2\ 3\ 1,\ 3\ 1\ 2,\ 3\ 2\ 1 \qquad (3)$$

现在，考虑怎样从这一数列得到 4 个对象的排列。为了从 $n-1$ 个对象过渡到 n 个对象，有两种主要的方法：

方法 1 对于 $n-1$ 个元素的每一个排列 $a_1a_2\cdots a_{n-1}$，通过在所有可能的位置上插入数 n，以形成 n 个所求的排列，便得到了

$$n\,a_1a_2\cdots a_{n-1},\ a_1\,n\,a_2\cdots a_{n-1},\ \cdots$$
$$a_1a_2\cdots n\,a_{n-1},\ a_1a_2\cdots a_{n-1}n$$

例如，从(3)中的排列 2 3 1，我们得到 4 2 3 1，2 4 3 1，2 3 4 1，2 3 1 4。显然，以这种方式就得到了 n 个对象的所有排列，而且不会产生同一个排列出现多于一次的情形。

方法 2 对于元素 $\{1,\ 2,\ \cdots,\ n-1\}$ 的每一个排列 $a_1a_2\cdots a_{n-1}$，形成 n 个其它的排列如下：首先，构造数列

$$a_1a_2\cdots a_{n-1}\frac{1}{2},\ a_1a_2\cdots a_{n-1}\frac{3}{2},\ \cdots,\ a_1a_2\cdots a_{n-1}\left(n-\frac{1}{2}\right)$$

然后，用数 1，2，…，n 保留次序地重新命名每个排列的诸元素。例如，从(3)中的排列 2 3 1，我们得到

$$2\ 3\ 1\ \frac{1}{2},\ 2\ 3\ 1\ \frac{3}{2},\ 2\ 3\ 1\ \frac{5}{2},\ 2\ 3\ 1\ \frac{7}{2}$$

然后，重新命名，我们得到

$$3\ 4\ 2\ 1,\ 3\ 4\ 1\ 2,\ 2\ 4\ 1\ 3,\ 2\ 3\ 1\ 4$$

为描述同一过程，另一方法是取一个排列 $a_1a_2\cdots a_{n-1}$ 和一个数 k，$1\leqslant k\leqslant n$；把每一个 $\geqslant k$ 的 a_i 加 1，于是就得到元素 $\{1,\ \cdots,\ k-1,\ k+1,\ \cdots,\ n\}$ 上的一个排列 $b_1b_2\cdots b_{n-1}$。现在，$b_1b_2\cdots b_{n-1}\,k$ 就是 $\{1,\ \cdots,\ n\}$ 上的一个排列。

显然，我们通过这一构造，还是得到了 n 个元素的每个排列恰巧一次。当然也可以使用类似的方法（即把 k 放在左边而不是右边，或者是把 k 放在任何其它固定的位置上）。

如果 p_n 是 n 个对象的排列数，则上边两种方法都说明 $p_n=n\,p_{n-1}$，而且这就为我们提供了关于 $p_n=n(n-1)\cdots(1)$ 的两个进一步的证明，如同我们已在等式(2)中建立起来的那样。

这个重要的数量 p_n 叫做 n 的阶乘，而且把它写成为

$$n!=1\cdot 2\cdot\cdots\cdot n=\prod_{1\leqslant k\leqslant n}k \qquad (4)$$

我们约定对于空的乘积（参照 1.2.3 小节），其值为

$$0!=1 \qquad (5)$$

而且，通过这一约定，对于所有正整数 n，有基本恒等式

$$n!=(n-1)!\ n \qquad (6)$$

成立。

在计算机的工作中，阶乘经常出现，因此建议读者要记住头几个阶乘的值：

$$0!=1,\quad 1!=1,\quad 2!=2,\quad 3!=6,\quad 4!=24,\quad 5!=120$$

阶乘之增长非常迅速，数 1000! 是一个位数超过 2500 的整数。

记住

$$10!=3,628,800$$

是有帮助的。人们必须记住，10! 大约是 $3\frac{1}{2}$ 百万。在某种意义下，10! 标志着实际是能计算还是不能计算这两者之间的一个近似的分界线。如果一个算法需要校验多于10! 种情况，则在实际的计算机上进行过长时间的运行可能是冒险的。反过来，如果要检验 10! 种情况，而每种情况需要比如说一毫秒的计算机时间，则整个运行将需要约一个小时。当然，这些解释是非常含糊的。但对于给出一个什么是计算上可行的直观的概念来说，是有帮助的。

很自然，人们极想知道，这 $n!$ 与数学上其它的量有什么关系。如不吃力地执行在等式(4)中所示的乘法，还能不能有什么办法来说出 1000! 究竟是多大呢？詹姆斯·斯特林(James Stirling)在他著名的著作《微分方法》(Methodus Differentialis(1730))的第 137 页中，给出了答案。我们有

$$n!\approx\sqrt{2\pi n}\left(\frac{n}{e}\right)^n \tag{7}$$

符号"$\approx$"表示"近似地等于"，而"e"是在 1.2.2 小节中所介绍的自然对数的底。我们将在 1.2.11.2 中证明斯特林的近似公式(7)。

作为使用这一公式的一个例子，我们可以计算

$$40320=8!\approx 4\sqrt{\pi}\left(\frac{8}{e}\right)^8=2^{25}\sqrt{\pi}e^{-8}\approx(67108864)(1.77245)(0.00033546)\approx 39902$$

在这种情况下，误差大约是 1%。我们稍后将看到，相对误差近似于 $1/(12n)$。

除了由等式(7)给出的近似值外，我们也可以较为容易地得到 $n!$ 分解成质因子的精确值。事实上，质数 p 是 $n!$ 的重数为

$$\mu=\left\lfloor\frac{n}{p}\right\rfloor+\left\lfloor\frac{n}{p^2}\right\rfloor+\left\lfloor\frac{n}{p^3}\right\rfloor+\cdots=\sum_{k>0}\left\lfloor\frac{n}{p^k}\right\rfloor \tag{8}$$

的一个因子。例如，如果 $n=1000$ 和 $p=3$，则我们有

$$\mu=\left\lfloor\frac{1000}{3}\right\rfloor+\left\lfloor\frac{1000}{9}\right\rfloor+\left\lfloor\frac{1000}{27}\right\rfloor+\left\lfloor\frac{1000}{81}\right\rfloor+\left\lfloor\frac{1000}{243}\right\rfloor+\left\lfloor\frac{1000}{729}\right\rfloor$$
$$=333+111+37+12+4+1=498$$

所以 1000! 能为 3^{498}所整除，但不能被 3^{499}所整除。尽管公式(8)写成一个无穷求和的形式，但对于任何具体的 n 和 p 值，它实际上是有限的，因为后边的所有项总归都是 0。从习题 1.2.4-35 推知，$\lfloor n/p^{k+1}\rfloor=\lfloor\lfloor n/p^k\rfloor/p\rfloor$；这一事实简化了等式(8)中的计算，因为我们仅仅需要用 p 来除前项的值并抛弃余数。

为证明等式(8)的正确性，我们可以来考察$\lfloor n/p^k\rfloor$，它是诸整数$\{1,\ 2,\ \cdots,\ n\}$当中为 p^k 之倍数者的个数。因此，如果我们来研究乘积(4)中的整数，则任何能被 p^j 但不能被 p^{j+1}整除的整数，都被精确地算了 j 次：在$\lfloor n/p\rfloor$中算一次，在$\lfloor n/p^2\rfloor$中算一次，

…，在$\lfloor n/p^j \rfloor$中算一次。这就是$n!$中p作为因子出现的总次数。

产生了另外一个自然的问题：现在，我们已经对非负整数定义了$n!$，也许阶乘函数对于有理数值n，甚而对于实数值也有意义？例如，$\left(\frac{1}{2}\right)!$等于什么？这里，让我们通过引进"项"函数

$$n? = 1 + 2 + \cdots + n = \sum_{1 \leqslant k \leqslant n} k \tag{9}$$

来说明这一点。这个函数类似于阶乘函数，只是我们用加来代替乘。我们已经知道这个算术级数的和是（参照等式 1.2.3-15）：

$$n? = \frac{1}{2} n(n+1) \tag{10}$$

通过使用等式(10)来代替等式(9)，就提供了一个把"项"函数推广到任意n的好办法。我们有$\left(\frac{1}{2}\right)? = \frac{3}{8}$。

斯特林本人曾经几次试图把$n!$推广到非整数n的情形。他把近似公式(7)展成一个无穷和，但不幸这个和不是对任意的n值都收敛。这个近似的方法给出了极好的近似值，但是它不能被推广来计算精确的值〔对于这一比较异常情况的讨论，请看康·诺普(K. Knopp)的《无穷级数的理论和应用》(Theory and Application of Infinite Series)第二版（格拉斯哥：布莱其，1951），第518～520，527，534页〕。

注意到（我们将在下一小节来证明这个公式）

$$\begin{aligned} n! = 1 &+ \left(1 - \frac{1}{1!}\right) n + \left(1 - \frac{1}{1!} + \frac{1}{2!}\right) n(n-1) \\ &+ \left(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!}\right) n(n-1)(n-2) + \cdots \end{aligned} \tag{11}$$

斯特林进行了再次的尝试。等式(11)中表面上无穷的求和实际上对于任意非负整数n是有限的；然而，它未提供所希望的$n!$的推广，因为除了当n是一个非负整数时外，此无穷求和不存在（参照习题 16）。

斯特林仍然不退缩，发现了一个序列$a_1, a_2, \cdots$使得

$$\ln n! = a_1 n + a_2 n(n-1) + \cdots = \sum_{k \geqslant 0} a_{k+1} \prod_{0 \leqslant j \leqslant k} (n - j) \tag{12}$$

但他不能证明，这个和对于所有为小数的n值，都定义了$n!$，尽管他有能力求出$\left(\frac{1}{2}\right)! = \sqrt{\pi}/2$。

大约在同一时间，伦哈特·欧拉(Leonhard Euler)研究了同一问题，而且他首先发现了适当的推广：

$$n! = \lim_{m \to \infty} \frac{m^n m!}{(n+1)(n+2)\cdots(n+m)} \tag{13}$$

欧拉在1729年10月13日写了一封信把这个思想告诉给克里斯琴·戈德巴赫(Christian Goldbach)。他的公式对于除了负整数外（当等式(13)的分母变成0时）的任意n值定义了$n!$，而且在n是负整数的情况下变成无穷。

大约两个世纪以后，到1900年，查·赫密特（C. Hermite）证明了，斯特林的思想（等式(12)）实际上已经对于非整数 n 定义了 $n!$，而且事实上欧拉和斯特林的推广是一致的。等式(13)并不是非常神秘的，稍稍给点启发（见习题22），读者自己就能发现它。

历史上，对于阶乘使用了许多记号。欧拉实际上写成 (n)，高斯写成 $\pi(n)$，而在英国使用了 $\lfloor n$ 和 $n\rfloor$ 等符号。今天普遍采用的记号 $n!$（当 n 为整数时），是由相对地不大出名的数学家克里斯琴·克兰普（Christian Kramp），在1808年写的一本代数课本中引进的。

然而，当 n 不是整数时，$n!$ 的记号很少使用。代替的是我们习惯上使用的，由艾·玛·勒让德（A. M. Legendre）给出的一个记号：

$$n! = \Gamma(n+1) = n\,\Gamma(n) \tag{14}$$

函数 $\Gamma(x)$ 叫做伽玛函数，而且由等式(13)，我们有定义

$$\Gamma(x) = \lim_{m\to\infty} \frac{m^x m!}{x(x+1)(x+2)\cdots(x+m)} \tag{15}$$

图7中给出了这个函数的图象。

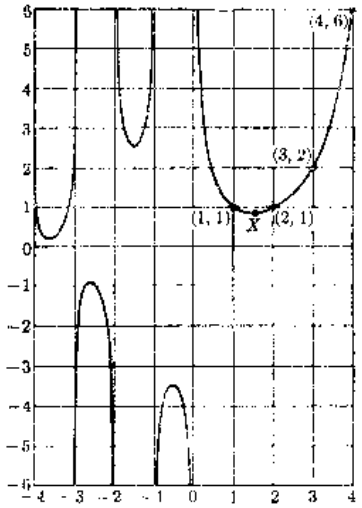


图7　函数 $\Gamma(x) = (x-1)!$ 点X的坐标为 (1.4616321450, 0.8856031944)

从斯特林那时开始到今天，关于阶乘的有趣历史，P. J. 戴维斯(P. J. Davis)在他的论文《欧拉积分：伽玛函数的历史侧面》(Leonhard Euler's Integral: A Historical Profile of the Gamma Function, AMM66, 1959, 849～869)中，作了记述。

习题

1.〔00〕 有多少种办法来洗52张纸牌？

2.〔10〕 在等式(2)的记号中，证明 $p_{n(n-1)} = p_{nn}$，并说明为什么出现这一情况。

3.〔10〕 从排列 3 1 2 4，分别地使用方法1和2，将构造出关于1，2，3，4，5的什么排列？

▶4.〔13〕 根据 $\log_{10}1000! = 2567.60164\cdots$，精确地确定在数 $1000!$ 中有多少位数？最高有效位数字是什么？最低有效位数字是什么？

5.〔15〕 用以下更精确形式的斯特林渐近公式来近似计算 $8!$：

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\frac{1}{12n}\right)。$$

▶6.〔17〕 用等式(8)，把 $20!$ 写成质因子的乘积。

7.〔*M*10〕 证明等式(10)中的推广的"项"函数，对所有实数 x 满足恒等式 $x^? = x + (x-1)^?$。

8.〔*HM*15〕 证明当 n 是非负整数时，等式(13)中的极限等于 $n!$。

9.〔*M*10〕 给定 $\left(\frac{1}{2}\right)! = \sqrt{\pi}/2$，试确定 $\Gamma\left(\frac{1}{2}\right)$ 和 $\Gamma\left(-\frac{1}{2}\right)$ 的值。

▶10.〔*HM*20〕 是否对所有实数 x，等式 $\Gamma(x+1) = x\,\Gamma(x)$ 都成立？（参照习题7）

11.〔*M*15〕 设在二进制系统中 n 的表示式是 $n = 2^{e_1} + 2^{e_2} + \cdots + 2^{e_r}$，其中 $e_1 > e_2 > \cdots$

$>e_r \geqslant 0$。证明 $n!$ 能被 2^{n-r} 但不能被 2^{n-r+1} 整除。

▶12.〔$M22$〕（艾·勒让德，1808）推广上题的结果，设 p 是一质数，且设在 p 进制系统中 n 的表示式是 $n=a_k p^k+a_{k-1}p^{k-1}+\cdots+a_1 p+a_0$。试以含有 n，p 和诸 a 的简单公式表示等式(8)中的数 μ。

13.〔$M23$〕（威尔逊(Wilson)定理，实际上是莱布尼兹给出的，1682年）。如果 p 是质数，则 $(p-1)!\mod p = p-1$。通过把 $1,2,\cdots,p-1$ 当中两者乘积 $\mod p$ 等于1的这样的数两两配对的办法，来证明这一定理。

▶14.〔$M28$〕（路·斯蒂克勒贝格(L. Stickelberger)1890）用习题12的记号，我们可以借助对于任意整数 n 的 p 进表示来确定 $n!\mod p$，从而推广威尔逊定理。事实上，可证 $n!/p^{\mu}\equiv(-1)^{\mu}a_0!\ a_1!\ \cdots a_k!\ (\text{modulo}\ p)$。

15.〔$HM15$〕对于正方矩阵，我们已经定义了行列式。现在，如果让行列式的展开式中的每项都带上正号而不出现负号，则称为“永久式”。于是，

$$\begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix}$$

的永久式是 $aei+bfg+cdh+gec+hfa+idb$。今问

$$\begin{pmatrix} 1\times 1 & 1\times 2 & \cdots & 1\times n \\ 2\times 1 & 2\times 2 & \cdots & 2\times n \\ \vdots & & & \vdots \\ n\times 1 & n\times 2 & \cdots & n\times n \end{pmatrix}$$

的永久式是什么？

16.〔$HM15$〕证明等式(11)中的无穷求和不收敛，除非 n 是非负整数。

17.〔$HM20$〕证明：如果 $\alpha_1+\cdots+\alpha_k=\beta_1+\cdots+\beta_k$，而且如果 β 都不是负整数，则无穷乘积

$$\prod_{n\geqslant 1}\frac{(n+\alpha_1)\cdots(n+\alpha_k)}{(n+\beta_1)\cdots(n+\beta_k)}$$

有值 $\Gamma(1+\beta_1)\cdots\Gamma(1+\beta_k)/\Gamma(1+\alpha_1)\cdots\Gamma(1+\alpha_k)$。

18.〔$M20$〕我们有 $\pi/2=\frac{2}{1}\cdot\frac{2}{3}\cdot\frac{4}{3}\cdot\frac{4}{5}\cdot\frac{6}{5}\cdot\frac{6}{7}\cdot\cdots$。（这是瓦里斯乘积，是由约·沃里斯（J. Wallis）于1656年得到的，我们将在习题1.2.6-43中证明之）。试用上题的结果，证明 $\left(\frac{1}{2}\right)!=\left(\frac{1}{2}\right)\sqrt{\pi}$。

19.〔$HM22$〕用 $\Gamma_m(x)$ 来表示出现在等式(15)中“$\lim_{m\to\infty}$”之后的量，证明：如果 $x>0$，则

$$\Gamma_m(x)=\int_0^m\left(1-\frac{t}{m}\right)^m t^{x-1}dt=m^x\int_0^1(1-t)^m t^{x-1}dt$$

20.〔$HM21$〕用这样一个事实：若 $0\leqslant t\leqslant m$，则 $0\leqslant e^{-t}-(1-t/m)^m\leqslant t^2e^{-t}/m$，和上一个习题，证明：当 $x>0$ 时，$\Gamma(x)=\int_0^{\infty}e^{-t}t^{x-1}dt$。

21. 〔*HM*25〕（伐敌奔(Faa di Bruno)公式）设 $D_x^k u$ 表示 u 关于 x 的第 k 次导数。"链锁规则"指出 $D_x^1 w = D_u^1 w D_x^1 u$。如果我们把这公式应用于二次导数，则我们求得 $D_x^2 w = D_u^2 w (D_x^1 u)^2 + D_u^1 w D_x^2 u$。试证明一般的公式是

$$D_x^n w = \sum_{0 \leqslant j \leqslant n} \sum_{\substack{k_1 + k_2 + \cdots + k_n = j \\ k_1 + 2k_2 + \cdots + nk_n = n \\ k_1,\ k_2,\ \cdots,\ k_n \geqslant 0}} D_u^j w \frac{n!}{k_1!\ (1!)^{k_1} \cdots k_n!\ (n!)^{k_n}} (D_x^1 u)^{k_1} \cdots (D_x^n u)^{k_n}$$

▶22. 〔*HM*20〕你自己也设身处地为欧拉想一想，找出一个把 $n!$ 推广到非整数的 n 值的办法。由于 $\left(n+\frac{1}{2}\right)! \,/\, n!$ 乘 $\left(\left(n+\frac{1}{2}\right)+\frac{1}{2}\right)! \,/\left(n+\frac{1}{2}\right)!$ 等于 $(n+1)!/n! = n+1$，看来 $\left(n+\frac{1}{2}\right)! \,/\, n!$ 似乎应近似于 $\sqrt{n}$。类似地，$\left(n+\frac{1}{3}\right)! \,/\, n!$ 应近似于 $\sqrt[3]{n}$。当 n 趋于无穷时，试想出关于比值 $(n+x)! \,/\, n!$ 的一个假设。当 x 为整数时，你的假设是否正确？当 x 不是整数时，关于 $x!$ 的值，这假设能否合理地说明什么？

1.2.6 二项式系数

n 个对象每次取 k 个的组合，是从 n 个对象的集合中，取 k 个不同元素的可能的选择。五个对象 $\{a, b, c, d, e\}$，每次取三个的组合是

$$abc,\ abd,\ abe,\ acd,\ ace,\ ade,\ bcd,\ bce,\ bde,\ cde \qquad (1)$$

为了计算 n 个对象取 k 个的组合总数，一个简单的办法是：上一小节的等式(2)告诉我们，选择头 k 个对象以形成一排列的方法有 $n(n-1)\cdots(n-k+1)$ 种；而在这些排列中，每一 k 个元素的组合恰巧出现 $k!$ 次，因为每一组合都出现在它的所有排列中。因此组合的总数，我们记之以 $\binom{n}{k}$，是

$$\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots(1)} \qquad (2)$$

例如，

$$\binom{5}{3} = \frac{5 \cdot 4 \cdot 3}{3 \cdot 2 \cdot 1} = 10$$

它就是我们在(1)中找出的组合的数目。

量 $\binom{n}{k}$ 称为二项式系数。这些数的应用格外之多，它们大概是研究算法分析的最重要的量。因此，我们强调读者应当熟悉它们。

甚至当 n 不是一个整数时，等式(2)也可用来定义 $\binom{n}{k}$。我们现在将对所有实数 r 和所有整数 k 来定义符号 $\binom{r}{k}$：

$$\left.\begin{aligned} \binom{r}{k} &= \frac{r(r-1)\cdots(r-k+1)}{k(k-1)\cdots(1)} = \prod_{1 \leqslant j \leqslant k} \left(\frac{r+1-j}{j}\right) && \text{整数 } k \geqslant 0 \\ \binom{r}{k} &= 0 && \text{整数 } k < 0 \end{aligned}\right\} \qquad (3)$$

对于特殊情况，我们有

$$\binom{r}{0}=1,\quad \binom{r}{1}=r,\quad \binom{r}{2}=\frac{r(r-1)}{2} \tag{4}$$

表 1 给出了对于小的整数 r 和 k 的二项式系数之值。对于 $0<r\leqslant 4$ 的那些都应当记住。

表 1　二项式系数表(帕斯卡尔(Pascal)三角)

r	$\binom{r}{0}$	$\binom{r}{1}$	$\binom{r}{2}$	$\binom{r}{3}$	$\binom{r}{4}$	$\binom{r}{5}$	$\binom{r}{6}$	$\binom{r}{7}$	$\binom{r}{8}$
0	1	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0
2	1	2	1	0	0	0	0	0	0
3	1	3	3	1	0	0	0	0	0
4	1	4	6	4	1	0	0	0	0
5	1	5	10	10	5	1	0	0	0
6	1	6	15	20	15	6	1	0	0
7	1	7	21	35	35	21	7	1	0
8	1	8	28	56	70	56	28	8	1

二项系数有一段漫长而有趣的历史。表 1 之所以叫做“帕斯卡尔三角”，是因为它出现于 1653 年的布莱斯·帕斯卡尔（Blaise Pascal）的著作《算术三角专著》（Traité du triangle arithmetique）中。这本专著是有历史意义的，因为它是关于概率论最早的论著之一。但帕斯卡尔不是二项式系数的发明者（那时二项式系数在欧洲已为人所熟知）。表1也出现在 1303 年中国数学家朱世杰的专著《四元玉鉴》中，书中说二项式系数是一项老发明。已经知道的二项式系数，其出现可追溯到十世纪的一部由哈拉尤达(Halāyudha)所著的，关于一位古代印度作家钱达-萨特拉(Chandah Sûtra) 的评论。大约 1150 年，印度数学家巴斯加拉·阿凯里雅（Bhāscara Achārya）在他所著的书《Lilávati》第六部分第四章中，给出了关于二项式系数的非常清楚的说明。对于小的 k 值，知道得更早；它们以一个几何解释出现在古希腊和罗马的一些著作中（参照图 8）。记号 $\binom{r}{k}$ 是由安德烈亚斯·冯·埃廷肖欣(Andreas von Ettingshausen)在他所写的书《组合分析》（Die Combinatorische Analysis)(维也纳，1826) 中引进的。

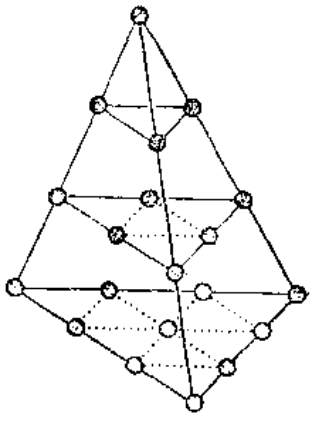


图 8 $\binom{n+2}{3}$ 的几何解释，$n=4$

读者大概已经看出出现在表 1 中的若干有趣的样式。关于二项式系数完全可以推出数以千计的恒等式来。而且若干世纪以来，它们的令人惊异的性质已经陆续地被发现。事实上，现有的关系式已经是如此之多，以致当某人发现一个新的恒等式时，除了发现者本人之外，再没有多少人还为之感到激动！为了处理出现在算法分析中的公式，必须简化对于二项式系数的处理。因此在这一小节里试图以简明的方式来说明怎样对这些数进行演算。马克·吐温（Mark Twain）曾经尝试把所有的笑话归结成一打左右的非常原始的类型，而我们则将尝试把成千个恒等式压缩成一小类基本的运算，通过这些基本运算，几乎就能够解决我们所面对的涉及这些数的每一个问题。

在大多数应用中，出现在$\binom{r}{k}$中的数 r 和 k 都将是整数。我们将叙述的某些技术仅当 r 和 k 都是整数时才可应用。所以，我们必须注意在每个编了号的等式右边列出，对于其中出现的变量的限制。例如，在等式(3)中，我们提出了 k 是一个整数的要求，而对 r 没有限制。

现在让我们来研究对于二项式系数进行运算的基本技术。

A．用阶乘来表示 从等式(3)，我们立即有

$$\binom{n}{k}=\frac{n!}{k!\,(n-k)!} \qquad \text{整数 } n\geqslant\text{整数 } k\geqslant 0 \tag{5}$$

这样，就能把阶乘的组合表示成二项式系数，或者反过来，把二项式系数表示成阶乘的组合。

B．对称条件 由等式(3)和(5)，我们有

$$\binom{n}{k}=\binom{n}{n-k} \qquad \text{整数 } n\geqslant 0,\ k \text{ 为整数} \tag{6}$$

这个公式对所有的整数 k 都成立。当 k 是负数或大于 n 时，二项式系数为 0（假定 n 是正整数）。

C．移进和移出括弧 由定义(3)，我们有

$$\binom{r}{k}=\frac{r}{k}\binom{r-1}{k-1} \qquad \text{整数 } k\neq 0 \tag{7}$$

这个公式，对于把一个二项式系数同一个表达式的其它部分组合在一起，是非常有用的。通过初等的变换，我们有规则

$$k\binom{r}{k}=r\binom{r-1}{k-1},\quad \frac{1}{r}\binom{r}{k}=\frac{1}{k}\binom{r-1}{k-1}$$

上边两个规则中的头一个对所有整数 k 都成立。而第二个只要不出现以 0 作除数的情况，就都是成立的。我们还有一个类似的关系：

$$\binom{r}{k}=\frac{r}{r-k}\binom{r-1}{k} \qquad \text{整数 } k\neq r \tag{8}$$

现在让我们通过交替地使用等式(6)和(7)来证明等式(8)，也从中来说明上列的变换技术：

$$\binom{r}{k}=\binom{r}{r-k}=\frac{r}{r-k}\binom{r-1}{r-1-k}=\frac{r}{r-k}\binom{r-1}{k}$$

〔注意 这个推导仅当 r 是一个 $\neq k$ 的正整数时才正确，因为这是包含在等式(6)和(8)中的限制。然而等式(8)的断言对任意 $r\neq k$ 仍成立。这可以以一个简单但是重要的方法进行证明：我们已经验证了

$$r\binom{r-1}{k}=(r-k)\binom{r}{k}$$

对无穷多个的 r 值成立。这个等式的两边是关于 r 的多项式。一个 n 次的非 0 多项式至多能有 n 个不同的零点，所以（通过减法），如果两个次数$\leqslant n$的多项式在 $n+1$ 或更多个不同的点上一致，则这两个多项式恒等。这一原理可以用来把对于整数成立的许多恒等式推广成对于所有实数的情形〕。

D. 加法公式 从表1中明显地看出一个基本关系

$$\binom{r}{k}=\binom{r-1}{k}+\binom{r-1}{k-1} \quad \text{整数 } k \tag{9}$$

(每一个值等于其上一个与这上一个的左边那一个之和)。而且，我们从等式(3)即可容易地进行一般的验证。或者，通过等式(7)和(8)，我们有

$$r\binom{r-1}{k}+r\binom{r-1}{k-1}=(r-k)\binom{r}{k}+k\binom{r}{k}=r\binom{r}{k}$$

当 r 是整数时，如果要对 r 用归纳法进行证明，则等式(9)通常是有用的。

E. 求和公式 反复地应用等式(9)，我们得到两个重要的求和公式：

$$\sum_{0\leqslant k\leqslant n}\binom{r+k}{k}=\binom{r}{0}+\binom{r+1}{1}+\cdots+\binom{r+n}{n}=\binom{r+n+1}{n}, \quad \text{整数 } n\geqslant 0 \tag{10}$$

$$\sum_{0\leqslant k\leqslant n}\binom{k}{m}=\binom{0}{m}+\binom{1}{m}+\cdots+\binom{n}{m}=\binom{n+1}{m+1}, \quad \text{整数 } m\geqslant 0,\ \text{整数 } n\geqslant 0 \tag{11}$$

通过对 n 用归纳法可以容易地证明等式(11)。但是看一看，它又怎样可以从等式(10)和两次应用等式(6)而推导出来，是有趣的：

$$\begin{aligned}\sum_{0\leqslant k\leqslant n}\binom{k}{m}&=\sum_{-m\leqslant k\leqslant n-m}\binom{m+k}{m}=\sum_{-m\leqslant k<0}\binom{m+k}{m}+\sum_{0\leqslant k\leqslant n-m}\binom{m+k}{k}\\&=0+\binom{m+(n-m)+1}{n-m}=\binom{n+1}{m+1}\end{aligned}$$

假定 $n\geqslant m$。而且，如果 $n<m$，则等式(11)是显然的。

在应用中，等式(11)出现得非常频繁。事实上，我们已经在上一小节里导出了它的特殊情况。例如，当 $m=1$ 时，我们有

$$\binom{0}{1}+\binom{1}{1}+\cdots+\binom{n}{1}=0+1+\cdots+n=\binom{n+1}{2}=\frac{(n+1)n}{2}$$

它是我们的老朋友了，它就是一个算术级数的和。

假设我们要求 $1^2+2^2+\cdots+n^2$ 的和。只要注意到 $k^2=2\binom{k}{2}+\binom{k}{1}$，这就可以解决了：

$$\sum_{0\leqslant k\leqslant n}k^2=\sum_{0\leqslant k\leqslant n}\left(2\binom{k}{2}+\binom{k}{1}\right)=2\binom{n+1}{3}+\binom{n+1}{2}$$

如果需要，所得到的表示为二项式系数的这一答案，也可以变回来用多项式的记号来写：

$$1^2+2^2+\cdots+n^2=2\frac{(n+1)n(n-1)}{6}+\frac{(n+1)n}{2}=\frac{1}{3}n\left(n+\frac{1}{2}\right)(n+1) \tag{12}$$

以类似的方式还可以得到 $1^3+2^3+\cdots+n^3$ 的和。只要适当地选择系数 b_0，…，b_m，则任何多项式 $a_0+a_1k+a_2k^2+\cdots+a_mk^m$ 都可表达成 $b_0\binom{k}{0}+b_1\binom{k}{1}+\cdots+b_m\binom{k}{m}$。后边，我们还要返回到这个论题上来。

F. 二项式定理 当然，二项式定理是我们的主要工具之一：

$$(x+y)^r=\sum_k\binom{r}{k}x^ky^{r-k} \quad \text{整数 } r\geqslant 0 \tag{13}$$

(终于，我们说明了为什么把这些数叫做“二项式系数”——这就是本名称的来由)。

说明一下。在等式(13)中，我们写的是“Σ_k”，而不是象以前常写的那样的“$\Sigma_{0\leqslant k\leqslant r}$”。注意到这一点是重要的。如果对 k 不加限制，则我们就对所有的整数值，$-\infty<k<+\infty$，来求和。但在当前情况下，这两个记号完全等价，因为当 $k<0$ 或 $k>r$ 时，等式(13)中的那些项都是 0。较简单的形式“Σ_k”要更好些，因为如果求和的条件愈简单，则对于求和的全部操作也就愈简单。如果我们不需要记住求和的上限或下限，则我们就省去了很多麻烦事。所以，只要有可能，就应当把限制放宽成无穷。我们的记号还有另一个优点：如果 r 不是一个非负整数，则等式(13)变成一个无穷和；而微积分学的二项式定理指出：如果 $|x/y|<1$，则等式(13)对所有实数 r 仍然成立。

应当指出，公式(13)给出了

$$0^0=1 \tag{14}$$

而且我们应始终沿用这一约定。

等式(13)中，$y=1$ 的特殊情况尤其重要，所以我们特别地指出来：

$$\sum_k \binom{r}{k} x^k=(1+x)^r \qquad \text{整数 } r\geqslant 0 \text{ 或} |x|<1 \tag{15}$$

二项式定理的发现是由伊萨·牛顿在1676年6月13日给奥尔登柏(Oldenburg)的信中宣告的。他显然没有给出这一公式的实际证明(而且，在当时，还没有充分认识到严格证明的必要性)。第一个尝试的证明是由伦·欧拉于1774年给出的，然而该证明也缺乏严格性。最后，卡·弗·高斯于1812年给出了头一个真正的证明。事实上，高斯的工作第一次给出了关于无穷求和之性质方面的完满的证明。

十九世纪初，尼·阿贝尔(N.Abel)发现了关于二项式公式(13)的一个惊人的推广：

$$(x+y)^r=\sum_k \binom{r}{k} x(x-kz)^{k-1}(y+kz)^{r-k}, \text{ 整数 } r\geqslant 0,\ x\neq 0。 \tag{16}$$

这是三个变量 x、y 和 z 的一个恒等式(参照习题50到52)。阿贝尔在德国的《纯粹与应用数学杂志》(Journal für die reine und angewandte Mathematik)(1826)第1卷(pp. 159～160)上，发表和证明了这个公式。有趣的是，阿贝尔在本卷中，还同时发表了好多篇论文，其中包括有关于五次以上代数方程的不可解性，以及关于二项式定理等等著名的论文。对于等式(16)的一些参考文献，请参看 AMM69(1962)，572。

G．把上标取负 由定义(等式(3))，把分子的每一项取负，直接得出基本恒等式

$$\binom{-r}{k}=(-1)^k\binom{r+k-1}{k} \quad \text{整数 } k \tag{17}$$

这常常是一个有用的关于上标的变换。

这里我们将给出一个使用等式(17)的例子，即用它来证明求和公式

$$\sum_{k\leqslant n}\binom{r}{k}(-1)^k=\binom{r}{0}-\binom{r}{1}+\cdots+(-1)^n\binom{r}{n}=(-1)^n\binom{r-1}{n} \quad \text{整数 } n\geqslant 0 \tag{18}$$

这个恒等式可以通过等式(9)用归纳法来证明。但我们也能用等式(17)和(10)容易地证明：

$$\sum_{k\leqslant n}\binom{r}{k}(-1)^k=\sum_{k\leqslant n}\binom{-r+k-1}{k}=\binom{-r+n}{n}=(-1)^n\binom{r-1}{n}$$

当 r 是一个整数时，可以给出等式(17)的一个重要的应用：

$$\binom{n}{m}=(-1)^{n-m}\binom{-(m+1)}{n-m}\quad \text{整数 } n\geqslant 0\quad \text{整数 } m \tag{19}$$

〔在等式(17)中，取 $n=-r$，$k=n-m$。〕这样就把 n 从上边的位置移到下边来了。

H. 简化乘积 当出现二项式系数的乘积时，通常有若干不同的方法重新来表达这个乘积：先把它展开成阶乘，然后再次使用等式（5），重又归拢成二项式系数的形式。例如，

$$\binom{r}{m}\binom{m}{k}=\binom{r}{k}\binom{r-k}{m-k}\quad \text{整数 } m\quad \text{整数 } k \tag{20}$$

为证明等式(20)，只须证明：当 r 是一个 $\geqslant m$ 的整数时〔参照等式(8)之后的论述〕，以及当 $0\leqslant k\leqslant m$ 时，有等式(20)成立。这样，

$$\binom{r}{m}\binom{m}{k}=\frac{r!m!}{m!(r-m)!k!(m-k)!}$$

$$=\frac{r!(r-k)!}{k!(r-k)!(m-k)!(r-m)!}=\binom{r}{k}\binom{r-k}{m-k}$$

当一个下标（即是 m）在上边和下边的位置同时出现时，等式(20)是非常有用的。而且我们所希望的是，要它出现在一个而不是两个位置上。注意，等式(7)是等式(20)当 $k=1$ 时的特殊情况。

I. 乘积的和 为了完备我们对二项式系数的整套处理方法，我们提出下列非常一般的恒等式，它们的证明留在本小节末尾的习题里。这些公式说明怎样对于两个二项式系数的一些乘积来进行求和，其中考虑到游动变量 k 可以出现在各种不同的位置上：

$$\sum_k\binom{r}{k}\binom{s}{n-k}=\binom{r+s}{n}\quad \text{整数 } n \tag{21}$$

$$\sum_k\binom{r}{k}\binom{s}{n+k}=\binom{r+s}{r+n}\quad \text{整数 } n\quad \text{整数 } r\geqslant 0 \tag{22}$$

$$\sum_k\binom{r}{k}\binom{s+k}{n}(-1)^k=(-1)^r\binom{s}{n-r}\quad \text{整数 } n\quad \text{整数 } r\geqslant 0 \tag{23}$$

$$\sum_{0\leqslant k\leqslant r}\binom{r-k}{m}\binom{s}{k-t}(-1)^k=(-1)^t\binom{r-t-s}{r-t-m}$$

$$\text{整数 } t\geqslant 0\quad \text{整数 } r\geqslant 0\quad \text{整数 } m\geqslant 0 \tag{24}$$

$$\sum_{0\leqslant k<r}\binom{r-k}{m}\binom{s+k}{n}=\binom{r+s+1}{m+n+1}$$

$$\text{整数 } n\geqslant \text{整数 } s\geqslant 0\quad \text{整数 } m\geqslant 0\quad \text{整数 } r\geqslant 0 \tag{25}$$

$$\sum_{k\geqslant 0}\binom{r-tk}{k}\binom{s-t(n-k)}{n-k}\frac{r}{r-tk}=\binom{r+s-tn}{n}\quad \text{整数 } n \tag{26}$$

在这些等式中，等式(21)是至今最重要的一个，而且应当记住它。记住它的一个方法是把

右边解释成从 r 个男人和 s 个女人当中选择 n 个人的方法的种数，左边的每项是选择 k 个男人和 $n-k$ 个女人的方法的种数。等式(21)通常叫做范特蒙德结合式，因为阿·范特蒙德于1772年在 MEM.Acad.Roy.Sci.Paris(489～498)发表了这个等式。

如果等式(26)中的 $r=tk$，则我们可通过删去分子中的一个因子以避免零分母。这样一来，等式(26)就是关于变量 r、s、t 的一个多项式的恒等式。显然，等式(21)是等式(26)当 $t=0$ 时的特殊情况。这些公式，是我们在解决困难的求和时的主要工具。

我们将指出等式(23)和(25)的一个不明显的用法。以左边更复杂的表达式来代替右边简单的二项式系数，交换求和的次序，并进行简化，这些，通常都是有帮助的。我们可以把左边看成是

$$\binom{s}{n-a}\text{的表示为}\binom{s+k}{n}$$

的展开式。公式(23)是对于负 a 的情形，公式(25)是对于正 a 的情形。

这就完成了我们对于“二项式系数学”的研究。建议读者特别地来学等式(5)，(6)，(7)，(9)，(13)，(17)，(20)和(21)——把它们用黑框标出来！

通过上面所安排的所有这些方法，我们应该有能力来解决“几乎任何”遇到的问题——至少有三种不同的办法。下列的例子说明了这些技术。

问题 1 当 r 是正整数时，

$$\sum_k \binom{r}{k}\binom{s}{k}k$$

的值等于什么？

解 为解决外边的 k，公式(7)是有用的：

$$\sum_k \binom{r}{k}\binom{s}{k}k=\sum_k \binom{r}{k}\binom{s-1}{k-1}s=s\sum_k \binom{r}{k}\binom{s-1}{k-1}$$

而现在可对于 $n=-1$，来应用公式(22)。因此答案是

$$\sum_k \binom{r}{k}\binom{s}{k}k=\binom{r+s-1}{r-1}s \qquad \text{整数 } r\geqslant 0$$

问题 2 当 $n\geqslant 0$ 时，

$$\sum_{k\geqslant 0}\binom{n+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1}$$

的值等于什么？

解 现在这个问题是比较棘手的，因为求和的下标 k 竟在六个位置上出现！首先，我们应用等式(20)，因而得到

$$\sum_{k\geqslant 0}\binom{n+k}{k}\binom{n}{k}\frac{(-1)^k}{k+1}$$

我们现在能松口气了，因为原来公式的若干吓人的特征现在已经消失，下一步应当是显然的了。我们以类似于在问题1中所使用的技术，来应用等式(7)：

$$\sum_{k\geqslant 0}\binom{n+k}{k}\binom{n+1}{k+1}\frac{(-1)^k}{n+1} \tag{27}$$

这样另一个 k 又消失了。现在，有两个同样有希望的着手的路子。我们可以用

$$\binom{n+k}{n}\text{来代替}\binom{n+k}{k}$$

并使用等式(23)：

$$\sum_{k\geqslant 0}\binom{n+k}{n}\binom{n+1}{k+1}\frac{(-1)^k}{n+1}=-\frac{1}{n+1}\sum_{k\geqslant 1}\binom{n-1+k}{n}\binom{n+1}{k}(-1)^k$$

$$=-\frac{1}{n+1}\sum_{k\geqslant 0}\binom{n-1+k}{n}\binom{n+1}{k}(-1)^k+\frac{1}{n+1}\binom{n-1}{n}$$

$$=-\frac{1}{n+1}(-1)^{n+1}\binom{n-1}{-1}+\frac{1}{n+1}\binom{n-1}{n}=\frac{1}{n+1}\binom{n-1}{n}$$

现在，

$$\binom{n-1}{n}$$

的值等于 0，除非 $n=0$ 时，那时它等于 1。

为便于表示我们的答案，可使用克罗内克尔的德尔塔记号：

$$\delta_{ij}=\begin{cases}0 & \text{若 } i\neq j\\ 1 & \text{若 } i=j\end{cases} \tag{28}$$

用 δ 记号，我们已经求得答案是 δ_{n0}。

从等式(27)着手处理的另一条路子是使用等式(17)，得到

$$\sum_{k}\binom{-(n+1)}{k}\binom{n+1}{k+1}\frac{1}{n+1}$$

在这里，不可应用等式(22)(因为它要求 $r\geqslant 0$)，但我们可以应用等式(6)，使得进而可以应用等式(21)：

$$\sum_{k}\binom{-(n+1)}{k}\binom{n+1}{n-k}\frac{1}{n+1}=\binom{0}{n}\frac{1}{n+1}$$

因而我们再次地推导出了答案是：

$$\sum_{k\geqslant 0}\binom{n+k}{2k}\binom{2k}{k}\frac{(-1)^k}{k+1}=\delta_{n0}\qquad \text{整数 } n\geqslant 0 \tag{29}$$

问题 3 对于正整数 m，n，

$$\sum_{k\geqslant 0}\binom{n+k}{m+2k}\binom{2k}{k}\frac{(-1)^k}{k+1}$$

的值等于什么？

解 如果 m 为 0，则我们就得到和问题 2 一样的式子。然而，现在 m 的出现意味着，我们甚至一开始就不可能来使用前解中的方法。因为上题中的头一步是使用等式(20)，而现在已不能再用它了。在这种情况下，我们宁肯用形如

$$\binom{x+k}{2k}$$

之项的和来代替

$$\binom{n+k}{m+2k}$$

虽然这反而使式子更加复杂化了，但由此我们的问题却变成了一个我们已经知道怎样来解决的求和问题了！因此，我们以

$$r=n+k-1,\quad m=2k,\quad s=0,\quad n=m-1$$

来使用等式(25)，而且我们有

$$\sum_{k\geqslant 0}\sum_{0\leqslant j\leqslant n+k-1}\binom{n+k-1-j}{2k}\binom{2k}{k}\binom{j}{m-1}\frac{(-1)^k}{k+1} \tag{30}$$

我们希望首先对 k 进行求和。为交换求和的次序，要求我们对$\geqslant 0$ 和$\geqslant j-n+1$ 的 k 值进行求和。后一条件出了问题，因为当 $j\geqslant n$ 时，我们就不知道所希望的和了。然而，如果我们观察到，当 $n\leqslant j\leqslant n+k-1$ 时，式(30)中的项都为 0，即可挽回局面。这个条件意味着 $k\geqslant 1$；于是 $0\leqslant n+k-1-j\leqslant k-1<2k$，这样(30)中的头一个二项式系数就消失了。因此，我们可以用“$0\leqslant j<n$”来代替第二个求和的条件，而且交换求和就容易进行了。将等式(29)对 k 进行求和，现在就给出

$$\sum_{0\leqslant j<n}\binom{j}{m-1}\delta_{(n-1-j)0}$$

而且除了 $j=n-1$ 外所有的项都为 0。因而，我们最后的答案是

$$\binom{n-1}{m-1}$$

这个问题的解答相当复杂，但也并非神秘莫测。对于每一步骤都有一个正当的理由。应该严密地研究这些推导，因为它体现了在各个等式的不同条件下，种种巧妙的演习！然而，实际上还有一个更好的方式解决这个问题，这要留给读者自己来进行：找出一个办法来，把给定的问题加以变换，以便应用等式(26)(见习题30)。

问题 4　证明

$$\sum_k A_k(r,t)A_{n-k}(s,t)=A_n(r+s,t)\quad \text{整数 } n\geqslant 0 \tag{31}$$

其中 $A_n(x,t)$是 x 的 n 次多项式，它满足

$$A_n(x,t)=\binom{x-nt}{n}\frac{x}{x-nt}\quad \text{对于 } x\neq nt$$

解　我们不妨设 $r\neq kt\neq s$ 对于 $0\leqslant k\leqslant n$，因为(31)是 r，s，t 的一个多项式。我们的问题是要计算

$$\sum_k\binom{r-kt}{k}\binom{s-(n-k)t}{n-k}\frac{r}{r-kt}\,\frac{s}{s-(n-k)t}.$$

看起来它比我们以前的可怕的问题还坏得多！然而，请注意它与等式（26）有强烈的相似性，也注意 $t=0$ 的情况。

我们试探着把

$$\binom{r-kt}{k}\frac{r}{r-kt}\text{ 改变成 }\binom{r-kt-1}{k-1}\frac{r}{k}$$

只是，后者显得反而失去了与等式(26)的相似性，而且当 $k=0$ 时还行不通。进行处理的最好办法是使用“部分分式”的技术，即是一个复杂的分母通常可以为若干较简单的分母之和所代替。事实上，我们有

$$\frac{1}{r-kt}\,\frac{1}{s-(n-k)t}=\frac{1}{r+s-nt}\left(\frac{1}{r-kt}+\frac{1}{s-(n-k)t}\right)$$

将其代入我们的求和式中，就得到

$$\frac{s}{r+s-nt}\sum_k\binom{r-kt}{k}\binom{s-(n-k)t}{n-k}\frac{r}{r-kt}$$

$$+\frac{r}{r+s-nt}\sum_k\binom{r-kt}{k}\binom{s-(n-k)t}{n-k}\frac{s}{s-(n-k)t}$$

现在用等式(26)即可计算出这两个式子，只要我们在第二个式子中把 k 变成 $(n-k)$。这样，就立即得出所希望的结果。恒等式(26)和(31)是由海·奥·罗特（H.A.Rothe）在《级数反演的公式》(Formulae de serierum reversione. 莱比锡，1793)中给出的。这些公式的特殊情况仍然屡屡被“发现”。关于这些恒等式的有趣历史及某些推广，请参看亨·沃·古尔德(H.W.Gould)和约·考克基(J.Kaucky)：《组合理论杂志》(Journal of Combinatorial Theory) 1 (1966)，233～248。

问题 5 对所有非负整数 n，确定 $a_0, a_1, a_2, \cdots$ 的值，使得

$$n!=a_0+a_1n+a_2n(n-1)+a_3n(n-1)(n-2)+\cdots \tag{32}$$

解 这个问题在前一小节（参照等式1.2.5-11)中就提出来了，而且我们已经给出了答案，只是未予证明。现在让我们假托我们并不知道答案。显然，这个问题有解，因为我们可以置 $n=0$ 并确定 a_0，然后置 $n=1$ 并确定 a_1，等等。

首先，我们会愿意借助二项式系数来写等式(32)：

$$n!=\sum_k\binom{n}{k}k!a_k \tag{33}$$

象这样的对 a_k 解隐式方程的问题叫做反演问题，而且所使用的技术同样也可应用于类似的问题。

这个思想是以下列的等式(23)的特殊情况（$s=0$）为基础的：

$$\sum_k\binom{r}{k}\binom{k}{n}(-1)^k=(-1)^r\binom{0}{n-r}=(-1)^r\delta_{nr}\quad\text{整数 }n\quad\text{整数 }r\geqslant 0 \tag{34}$$

这个公式的重要性在于：它说明当 $n\neq r$ 时，和数即为零。这样，就象在问题 3 中那样，大量的项都消失掉了，从而使我们得以解决所求的问题：

$$\sum_n n!\binom{m}{n}(-1)^n=\sum_n\sum_k\binom{n}{k}k!a_k\binom{m}{n}(-1)^n=\sum_k k!a_k\sum_n\binom{n}{k}\binom{m}{n}(-1)^n$$

$$=\sum_k k!a_k(-1)^m\delta_{km}=(-1)^m m!a_m$$

值得注意的是，我们怎样才能得到一个其中仅仅有一个 a_m 值出现的等式——把等式(33)的适当倍数对于 $n=0$，1，2，…加在一起。现在，我们有

$$a_m=\sum_{n\geqslant 0}(-1)^{m+n}\frac{n!}{m!}\binom{m}{n}=\sum_{0\leqslant n\leqslant m}\frac{(-1)^{m+n}}{(m-n)!}=\sum_{0\leqslant n\leqslant m}\frac{(-1)^n}{n!}$$

这就完成了问题5的求解。现在让我们更进一步来考察等式(34)给予我们的启示：我们有

$$\sum_k\binom{r}{k}(-1)^k\left(c_0\binom{k}{0}+c_1\binom{k}{1}+\cdots+c_r\binom{k}{r}\right)=(-1)^r c_r$$

因为在求和之后，前头诸项全都消失了。通过适当地选择系数 c_r，我们就能把关于 k 的任何多项式表示成为若干上标为 k 的二项式系数之和。因此，我们可得

$$\sum_k\binom{r}{k}(-1)^k(b_0+b_1k+\cdots+b_rk^r)=(-1)^r r!b_r,\quad \text{整数 } r\geqslant 0 \tag{35}$$

其中 $b_0+\cdots+b_rk^r$ 表示次数为 r 或是更低次数的任意多项式(这个公式对于数值分析的学生将不会太惊奇，因为 $\sum_k\binom{r}{k}(-1)^{r+k}f(x+k)$ 是函数 $f(x)$ 的"r 次差分")。

用等式(35)，我们立即能得到许多其它的关系式。这些关系式乍看起来显得很复杂，而且通常都需要很长的证明。例如，

$$\sum_k\binom{r}{k}\binom{s-kt}{r}(-1)^k=t^r\quad \text{整数 } r\geqslant 0 \tag{36}$$

通常在各种教科书里，就象本书一样，都给出了大量的灵巧变幻的例子，从而给人留下了深刻的印象。但是，却从不谈及那些看起来简单，然而上述这些技术却失灵了的问题。上列诸例子可能已经给人们留下了"二项式系数是万能的"这样一种深刻的印象。然而，必须指出，虽然有等式(10)，(11)和(18)，可是对于类似的求和：

$$\sum_{0\leqslant k\leqslant n}\binom{m}{k}=\binom{m}{0}+\binom{m}{1}+\cdots+\binom{m}{n},\quad \text{当 } n<m \text{ 时}$$

却没有简单的公式(对于 $n=m$，答案是简单的。它等于什么？见习题36)。

二项式系数的概念有许多推广，我们来简单地讨论一下。首先，我们可以考虑 $\binom{r}{k}$ 中的下标 k 的任意实数值，见习题40到45。我们还有推广

$$\binom{r}{k}_q=\frac{(1-q^r)(1-q^{r-1})\cdots(1-q^{r-k+1})}{(1-q^k)(1-q^{k-1})\cdots(1-q^1)} \tag{37}$$

当 q 趋于极限值1时，它就变成普通的二项式系数 $\binom{r}{k}_1=\binom{r}{k}$(通过以 $(1-q)$ 来除分子和分母中的每一项，即可看出)。在习题58中，讨论了这样的"q 项式系数"的基本性质。

然而，对于我们的目的说来，最重要的推广是多项式系数

$$\binom{k_1+k_2+\cdots+k_m}{k_1,k_2,\cdots,k_m}=\frac{(k_1+k_2+\cdots+k_m)!}{k_1!k_2!\cdots k_m!}\quad \text{整数 } k_i\geqslant 0 \tag{38}$$

多项式系数的主要性质是等式(13)的推广：

$$(x_1+x_2+\cdots+x_m)^n=\sum_{k_1+k_2+\cdots+k_m=n}\binom{n}{k_1,k_2,\cdots,k_m}x_1^{k_1}x_2^{k_2}\cdots x_m^{k_m} \tag{39}$$

重要的是要注意到：可以借助二项式系数来表达任何多项式系数：

$$\binom{k_1+k_2+\cdots+k_m}{k_1,k_2,\cdots,k_m}=\binom{k_1+k_2}{k_1}\binom{k_1+k_2+k_3}{k_2+k_3}\cdots\binom{k_2+k_3+\cdots+k_m}{k_1+\cdots+k_{m-1}}$$

所以可以应用我们所已知道的处理二项式系数的技术。注意(20)是一个三项式系数。

在结束这一小节之前，要简单地分析一下，如何把一个表达成 k 之幂的多项式变换为一个表达成二项式系数的多项式。在这个变换中所包含的系数叫做斯特林数，而且这些数在本书的后面章节中将出现多次。

斯特林数的出现有两种形式：我们以$\begin{bmatrix}n\\k\end{bmatrix}$来表示头一种类型的斯特林数，而以$\begin{Bmatrix}n\\k\end{Bmatrix}$来表示第二种类型的斯特林数。表 2 列出了"斯特林三角"，在某些方面，它类似于帕斯卡尔三角。

今天，对于斯特林数的记号没有绝对一致的意见。某些作者所定义的斯特林数，有一半恰与这里所给出的值符号相反。然而，这里所使用的记号，其中所有的斯特林数都是非负的，这种记号使我们更易于记住它与二项式系数的相似性。

头一类的斯特林数用于把二项式系数转换成幂的形式：

$$\begin{aligned}n!\binom{x}{n}&=x(x-1)\cdots(x-n+1)\\&=\begin{bmatrix}n\\n\end{bmatrix}x^n-\begin{bmatrix}n\\n-1\end{bmatrix}x^{n-1}+\cdots+(-1)^n\begin{bmatrix}n\\0\end{bmatrix}\\&=\sum_k(-1)^{n-k}\begin{bmatrix}n\\k\end{bmatrix}x^k\end{aligned} \tag{40}$$

例如，由表 2，

$$\binom{x}{5}=\frac{1}{120}(x^5-10x^4+35x^3-50x^2+24x)$$

第二类的斯特林数用于把幂转换成二项式系数

$$x^n=\begin{Bmatrix}n\\n\end{Bmatrix}\binom{x}{n}n!+\cdots+\begin{Bmatrix}n\\1\end{Bmatrix}\binom{x}{1}1!+\begin{Bmatrix}n\\0\end{Bmatrix}\binom{x}{0}0!=\sum_k\begin{Bmatrix}n\\k\end{Bmatrix}\binom{x}{k}k! \tag{41}$$

例如，由表 2，

$$\begin{aligned}x^5&=\binom{x}{5}5!+10\binom{x}{4}4!+25\binom{x}{3}3!+15\binom{x}{2}2!+\binom{x}{1}1!\\&=120\binom{x}{5}+240\binom{x}{4}+150\binom{x}{3}+30\binom{x}{2}+\binom{x}{1}\end{aligned}$$

现在让我们列出关于斯特林数的最重要的恒等式（在这些等式中，变量 m 和 n 总表示非负整数）。

加法公式：

$$\left.\begin{aligned}\begin{bmatrix}n\\m\end{bmatrix}&=(n-1)\begin{bmatrix}n-1\\m\end{bmatrix}+\begin{bmatrix}n-1\\m-1\end{bmatrix}\\\begin{Bmatrix}n\\m\end{Bmatrix}&=m\begin{Bmatrix}n-1\\m\end{Bmatrix}+\begin{Bmatrix}n-1\\m-1\end{Bmatrix}\end{aligned}\right\}\text{当 } n>0 \text{ 时} \tag{42}$$

表 2　第一类和第二类 Stirling 数❶

$\begin{bmatrix} n \\ 0 \end{bmatrix}$	$\begin{bmatrix} n \\ 1 \end{bmatrix}$	$\begin{bmatrix} n \\ 2 \end{bmatrix}$	$\begin{bmatrix} n \\ 3 \end{bmatrix}$	$\begin{bmatrix} n \\ 4 \end{bmatrix}$	$\begin{bmatrix} n \\ 5 \end{bmatrix}$	$\begin{bmatrix} n \\ 6 \end{bmatrix}$	$\begin{bmatrix} n \\ 7 \end{bmatrix}$	$\begin{bmatrix} n \\ 8 \end{bmatrix}$	n	$\begin{Bmatrix} n \\ 0 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 1 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 2 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 3 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 4 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 5 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 6 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 7 \end{Bmatrix}$	$\begin{Bmatrix} n \\ 8 \end{Bmatrix}$
1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	2	0	1	1	0	0	0	0	0	0
0	2	3	1	0	0	0	0	0	3	0	1	3	1	0	0	0	0	0
0	6	11	6	1	0	0	0	0	4	0	1	7	6	1	0	0	0	0
0	24	50	35	10	1	0	0	0	5	0	1	15	25	10	1	0	0	0
0	120	274	225	85	15	1	0	0	6	0	1	31	90	65	15	1	0	0
0	720	1764	1624	735	175	21	1	0	7	0	1	63	301	350	140	21	1	0
0	5040	13068	13132	6769	1960	322	28	1	8	0	1	127	966	1701	1050	266	28	1

❶ 对于进一步的值，请参看由米·阿布拉莫维茨(M. Abramowitz)和艾·安·斯特冈(I. A. Stegun)合编的《数学函数手册》(Handbook of Mathematical Functions)(美国政府出版社，1964)手册中的表24.3和24.4，其中$\begin{bmatrix} n \\ m \end{bmatrix}$以$(-1)^{n+m}S_n^{(m)}$表示，$\begin{Bmatrix} n \\ m \end{Bmatrix}$以$\$_n^m$表示。对于充分大的$m$和$n$时的近似值，请参看弗·奈·戴维(F. N. David)和戴·埃·巴顿(D. E. Barton)的《组合机会》(Combinatorial Chance)(纽约：哈法尼尔，1962)，第16章。

反演公式(对照等式34)：

$$\sum_k \begin{bmatrix} n \\ k \end{bmatrix} \begin{Bmatrix} k \\ m \end{Bmatrix} (-1)^k = (-1)^n \delta_{mn}, \quad \sum_k \begin{Bmatrix} n \\ k \end{Bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^k = (-1)^n \delta_{mn} \tag{43}$$

特殊的值：

$$\binom{0}{n} = \begin{bmatrix} 0 \\ n \end{bmatrix} = \begin{Bmatrix} 0 \\ n \end{Bmatrix}, \quad \binom{n}{n} = \begin{bmatrix} n \\ n \end{bmatrix} = \begin{Bmatrix} n \\ n \end{Bmatrix} = 1 \tag{44}$$

$$\begin{bmatrix} n \\ n-1 \end{bmatrix} = \begin{Bmatrix} n \\ n-1 \end{Bmatrix} = \binom{n}{2} \tag{45}$$

$$\begin{bmatrix} n \\ 0 \end{bmatrix} = \begin{Bmatrix} n \\ 0 \end{Bmatrix} = 0, \quad \begin{bmatrix} n \\ 1 \end{bmatrix} = (n-1)!, \quad \begin{Bmatrix} n \\ 1 \end{Bmatrix} = 1, \quad \begin{Bmatrix} n \\ 2 \end{Bmatrix} = 2^{n-1} - 1 \quad \text{若 } n > 0 \tag{46}$$

展开公式：

$$\sum_k \begin{bmatrix} n \\ k \end{bmatrix} \binom{k}{m} = \begin{bmatrix} n+1 \\ m+1 \end{bmatrix}, \quad \sum_k \begin{bmatrix} n+1 \\ k+1 \end{bmatrix} \binom{k}{m} (-1)^k = \begin{Bmatrix} n \\ m \end{Bmatrix} (-1)^m \tag{47}$$

$$\sum_k \begin{Bmatrix} k \\ m \end{Bmatrix} \binom{n}{k} = \begin{Bmatrix} n+1 \\ m+1 \end{Bmatrix}, \quad \sum_k \begin{Bmatrix} k+1 \\ m+1 \end{Bmatrix} \binom{n}{k} (-1)^k = \begin{Bmatrix} n \\ m \end{Bmatrix} (-1)^n \tag{48}$$

$$\sum_k \binom{n}{k} k^m (-1)^k = (-1)^n n! \begin{Bmatrix} m \\ n \end{Bmatrix} \tag{49}$$

$$\left.\begin{aligned} \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{Bmatrix} m+k \\ k \end{Bmatrix} &= \begin{bmatrix} n \\ n-m \end{bmatrix} \\ \sum_k \binom{m-n}{m+k} \binom{m+n}{n+k} \begin{bmatrix} m+k \\ k \end{bmatrix} &= \begin{Bmatrix} n \\ n-m \end{Bmatrix} \end{aligned}\right\} \text{若 } n \geqslant m \tag{50}$$

$$\sum_k \begin{Bmatrix} n+1 \\ k+1 \end{Bmatrix} \begin{bmatrix} k \\ m \end{bmatrix} (-1)^k = (-1)^m \binom{n}{m} \tag{51}$$

$$\sum_{k \leqslant n} \begin{bmatrix} k \\ m \end{bmatrix} \frac{n!}{k!} = \begin{bmatrix} n+1 \\ m+1 \end{bmatrix}, \quad \sum_{k \leqslant n} \begin{Bmatrix} k \\ m \end{Bmatrix} (m+1)^{n-k} = \begin{Bmatrix} n+1 \\ m+1 \end{Bmatrix} \tag{52}$$

其它一些基本的斯特林数的恒等式出现在习题1.2.6-61，1.2.7-6以及1.2.9小节的等式(23)，(26)，(27)和(28)中。关于斯特林数的进一步的介绍，请参看卡罗里(查尔斯)约当(Karoly(Charles)Jordan)著的《有限差分计算》(Calculas of Finite Differences)(纽约：切尔西，1947)，第4章。

习题

1. 〔00〕 从n件事物中每次取$n-1$件，总共有多少可能的组合？

2. 〔00〕 $\binom{0}{0}$等于什么？

3. 〔00〕 可能有多少种桥牌手牌（即从52张牌中取出13张来）？

4. 〔10〕 把问题3的答案以质数的乘积形式给出来。

▶5. 〔05〕 借助于帕斯卡尔三角来说明$11^4=14641$。

▶6. 〔10〕 通过使用加法公式和等式(9)，可以沿所有的方向来扩充帕斯卡尔三角(表1)。找出表1顶部往上去的头三行（即是，$r=-1$，-2，-3的情形）。

7. 〔12〕 若n是一个固定的正整数，则为使$\binom{n}{k}$之值为极大，k应取何值？

8. 〔00〕 在“对称条件”等式(6)中，反映了帕斯卡尔三角的什么性质？

9. 〔01〕 $\binom{n}{n}$的值等于什么？（考虑所有的整数n。）

▶10. 〔*M*25〕 若p是质数，证明：

a) $\binom{n}{p}\equiv\left[\begin{matrix}n\\p\end{matrix}\right]$(modulo p)。

b) $\binom{p}{k}\equiv 0$(modulo p)，对于$1\leqslant k\leqslant p-1$。

c) $\binom{p-1}{k}\equiv(-1)^k$(modulo p)，对于$0\leqslant k\leqslant p-1$。

d) $\binom{p+1}{k}\equiv 0$(modulo p)，对于$2\leqslant k\leqslant p-1$。

e) (埃·路卡斯(E.Lucas)，1877)

$$\binom{n}{k}\equiv\binom{\lfloor n/p\rfloor}{\lfloor k/p\rfloor}\binom{n \bmod p}{k \bmod p}\quad(\text{modulo } p)。$$

f) 若n，k在p进数系统中的表示为

$$n=a_r p^r+\cdots+a_1 p+a_0,\quad k=b_r p^r+\cdots+b_1 p+b_0,\quad \text{则}\binom{n}{k}\equiv\binom{a_r}{b_r}\cdots\binom{a_1}{b_1}\binom{a_0}{b_0}(\text{modulo } p)。$$

▶11. 〔*M*20〕 (厄·库莫尔(E.Kummer)，1852)设p为质数。证明若p^n能整除

$$\binom{a+b}{a}$$

但p^{n+1}不能整除，则n就等于在p进数系统中，a加b时所出现的进位个数（参照习题1.2.5-12)。

12. 〔*M*22〕 存在这样的正整数n否，它使得帕斯卡三角的第n行中的所有非零项都是奇数？若有，试把所有这样的n找出来。

13. 〔$M13$〕 证明求和公式，等式(10)。

14. 〔$M21$〕 计算 $\Sigma_{0\leqslant k\leqslant n} k^4$。

15. 〔$M15$〕 证明二项式公式，等式(13)。

16. 〔$M15$〕 给定 n，k 为正整数，证明

$$(-1)^n\binom{-n}{k-1}=(-1)^k\binom{-k}{n-1}。$$

▶17. 〔$M18$〕 利用 $(1+x)^{r+s}=(1+x)^r(1+x)^s$ 的思想，由等式(13)来证明基本恒等式(21)。

18. 〔$M15$〕 利用等式(21)和(6)，证明等式(22)。

19. 〔$M18$〕 用归纳法证明等式(23)。

20. 〔$M20$〕 用等式(21)和(19)，来证明等式(24)，然后说明等式(19)的另一个应用：得出等式(25)。

▶21. 〔$M05$〕 等式(25)的两边都是 s 的多项式；为什么该等式不是关于 s 的恒等式？

22. 〔$M20$〕 对于 $s=n-1-r+mt$ 的特殊情况，来证明等式(26)。

23. 〔$M13$〕 假定对于 $(r,\ s,\ t,\ n)$ 和 $(r,\ s-t,\ t,\ n-1)$ 等式(26)成立，证明对于 $(r,\ s+1,\ t,\ n)$ 它也成立。

24. 〔$M15$〕 说明为什么把上两题的结果结合起来，就给出了等式(26)的一个证明？

25. 〔$HM30$〕 设多项式 $A_n(x,\ t)$ 之定义如等式(31)所示。设 $z=x^{t+1}-x^t$。证明 $\Sigma_k A_k(r,\ t)z^k=x^r$，假定 z 足够小〔注意：当 $t=0$ 时，这个结果实质上就是二项式定理，而且这个等式是二项式定理的一个重要的推广。二项式定理（等式(15)）在证明中可以采用〕。提示：由恒等式

$$\sum_j(-1)^j\binom{k}{j}\binom{r-jt}{k}\frac{r}{r-jt}=\delta_{k0}$$

开始。

26. 〔$HM25$〕 利用上题的假定，证明

$$\sum_k\binom{r-tk}{k}z^k=\frac{x^{r-1}}{(t+1)x-t}$$

27. 〔$HM20$〕 利用习题25的结果来解正文中的问题4；并由上两题来证明等式(26)。

28. 〔$M25$〕 证明当 n 是非负整数时，有

$$\sum_k\binom{r+tk}{k}\binom{s-tk}{n-k}=\sum_{k\geqslant 0}\binom{r+s-k}{n-k}t^k$$

29. 〔$M20$〕 说明等式(35)仅仅是习题1.2.3-33中所证明的一般恒等式的一个非常特殊的情况。

▶30. 〔$M24$〕 通过把求和处理成可以应用等式(26)那样，证明有一个比书上所用的办法还好的解决问题3的办法。

▶31. 〔$M20$〕 借助于 r，s，m 和 n，计算

$$\sum_k\binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r+k}{m+n}$$

假定m和n皆为非负整数。通过以

$$\sum_j \binom{r}{m+n-j}\binom{k}{j} \quad \text{来代替} \binom{r+k}{m+n}$$

开始。

32. 〔$M20$〕 用记号$x^{\overline{n}}$表示$x(x+1)\cdots(x+n-1)$。证明$\Sigma_k \left[\begin{matrix} n \\ k \end{matrix}\right] x^k = x^{\overline{n}}$。

33. 〔$M20$〕 用上题的记号，证明当用修正的“幂”来代替普通的幂时，二项式公式也是成立的，即证明：

$$(x+y)^{\overline{n}} = \sum_k \binom{n}{k} x^{\overline{k}} y^{\overline{n-k}}$$

34. 〔$M23$〕 (拖勒利(Torelli)和数)按照上题的方法，证明对于修正的“幂”，关于二项式公式的阿贝尔推广：

$$(x+y)^{\overline{n}} = \sum_k \binom{n}{k} x\,(x-kz+1)^{\overline{k-1}}(y+kz)^{\overline{n-k}} \quad \text{(参照等式16)也成立。}$$

35. 〔$M23$〕 直接由定义，即由等式(40)和(41)，来证明对于斯特林数的加法公式(42)。

36. 〔$M10$〕 帕斯卡尔三角的每行中诸数之和$\Sigma_k \binom{n}{k}$等于什么？带有交替的正负号的这些数之和$\Sigma_k \binom{n}{k}(-1)^k$等于什么？

37. 〔$M10$〕 从上题的答案，推导出在一行中每隔一项之和的数值：$\binom{n}{0}+\binom{n}{2}+\binom{n}{4}+\cdots$

38. 〔$HM30$〕 (克·拉马斯(C. Ramus)，1834)推广上题的结果，证明：对于给定的$0 \leqslant k < m$，我们有以下的公式：

$$\binom{n}{k}+\binom{n}{m+k}+\binom{n}{2m+k}+\cdots = \frac{1}{m}\sum_{0\leqslant j<m}\left(2\cos\frac{j\pi}{m}\right)^n \cos\frac{j(n-2k)\pi}{m}$$

例如，

$$\binom{n}{1}+\binom{n}{4}+\binom{n}{7}+\cdots = \frac{1}{3}\left(2^n+2\cos\frac{(n-2)\pi}{3}\right)$$

〔提示：求出右边的以m次单位根乘这些系数的组合〕。当$m \geqslant n$时，这个恒等式是特别值得注意的。

39. 〔$M10$〕 第一种斯特林三角每行中诸数之和$\Sigma_k \left[\begin{matrix} n \\ k \end{matrix}\right]$等于什么？带有交替的正负号的这些数之和等于什么？(参照习题36)

40. 〔$HM17$〕 对于正实数x，y，通过公式$B(x, y) = \int_0^1 t^{x-1}(1-t)^{y-1}dt$定义贝塔函数$B(x, y)$。

a) 证明 $B(x, 1) = B(1, x) = 1/x$。

b) 证明 $B(x+1, y) + B(x, y+1) = B(x, y)$。

c) 证明 $B(x, y) = ((x+y)/y)B(x, y+1)$。

41. 〔$HM22$〕 当m为一正整数时，通过指出$\Gamma_m(x) = m^x B(x, m+1)$，我们就得到了习题1.2.5-19中的伽玛函数与贝塔函数之间的一个关系式。

a）证明 $B(x,y)=\dfrac{\Gamma_m(y)m^x}{\Gamma_m(x+y)}B(x,\ y+m+1)$

b）证明 $B(x,y)=\dfrac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$

42.〔$HM10$〕借助于如上定义的贝塔函数来表达二项式系数$\binom{r}{k}$（这就给了我们把二项式系数的定义推广到所有实数值的 k 的办法）。

43.〔$HM20$〕证明$B\left(\frac{1}{2},\ \frac{1}{2}\right)=\pi$（由习题41，我们现在可以结论$\Gamma\left(\frac{1}{2}\right)=\sqrt{\pi}$）。

44.〔$HM20$〕利用习题42中所提出的推广的二项式系数，证明

$$\binom{r}{1/2}=2^{2r+1}\Big/\binom{2r}{r}\pi$$

45.〔$HM21$〕利用习题42中所提出的推广的二项式系数，求$\lim_{r\to\infty}\binom{r}{k}\Big/r^k$。

▶46.〔$M21$〕利用斯特林近似公式（等式1.2.5-7），试求

$$\binom{x+y}{y}$$

的近似值，假定 x 和 y 都充分大。特别地，试求当 n 充分大时，$\binom{2n}{n}$的近似数值。

47.〔M21〕给定 k 为一整数，证明

$$\binom{r}{k}\binom{r-1/2}{k}=\binom{2r}{k}\binom{2r-k}{k}\Big/4^k$$

对于 $r=-1/2$ 的特殊情况给出一个更简单的公式。

▶48.〔$M25$〕只要分母不为0，便有

$$\sum_{k\geqslant 0}\binom{n}{k}\frac{(-1)^k}{k+x}=\frac{n!}{x(x+1)\cdots(x+n)}=\frac{1}{x\binom{n+x}{n}}$$

〔注意这个公式给予我们的是二项式系数的倒数，以及$1/x(x+1)\cdots(x+n)$的部分分式展开〕。

49.〔$M20$〕证明：由恒等式$(1+x)^r=(1-x^2)^r(1-x)^{-r}$可推出关于二项式系数的一个关系式。

50.〔$M20$〕证明在 $x+y=0$ 的特殊情况下的阿贝尔公式(16)。

51.〔$M21$〕通过写$y=(x+y)-x$，把右边展开成$(x+y)$的幂，以及应用上题的结果，来证明阿贝尔公式(16)。

52.〔$HM11$〕通过计算当 $r=x=-1$，$y=z=+1$ 时的右边的值，证明当 r 不是非负整数时，阿贝尔二项式公式(16)并非总成立。

53.〔M25〕（a）对 m 用归纳法，证明恒等式

$$\sum_{0\leqslant k\leqslant m}\binom{r}{k}\binom{s}{n-k}(nr-(r+s)k)=(m+1)(n-m)\binom{r}{m+1}\binom{s}{n-m}\text{整数}m,\ n。$$

（b）利用重要的关系式

$$\binom{-1/2}{n}=\frac{(-1)^n}{2^{2n}}\binom{2n}{n},\quad \binom{1/2}{n}=\frac{(-1)^{n-1}}{2^{2n}(2n-1)}\binom{2n}{n}=\frac{(-1)^{n-1}}{2^{2n-1}(2n-1)}\binom{2n-1}{n}$$

来证明下列的公式（可以作为小题（a）中的恒等式的特殊情况而得到）：

$$\sum_{0\leqslant k\leqslant m}\binom{2k-1}{k}\binom{2n-2k}{n-k}\frac{-1}{2k-1}=\frac{n-m}{2n}\binom{2m}{m}\binom{2n-2m}{n-m}+\frac{1}{2}\binom{2n}{n}$$

(本结果大大推广了在 $r=-1$，$s=0$，$t=-2$ 情况下的等式(26))。

54.〔$HM21$〕把帕斯卡尔三角（如表 1 所示）当作一个矩阵来考虑，试问此矩阵的逆是什么？

55.〔$M21$〕把斯特林三角（表 2）的每个当作矩阵来考虑，试确定它们的逆。

▶56.〔20〕（"二项式数系"）对于每一个整数 $n=0$，1，2，…，20，求三个整数 a，b，c 使得 $n=\binom{a}{1}+\binom{b}{2}+\binom{c}{3}$ 且 $0\leqslant a<b<c$。你能否看出，对于更大的 n 值如何继续下去？

▶57.〔$M22$〕说明斯特林在他推广阶乘函数（等式1.2.5-12)的尝试中的系数 a_m 是

$$\frac{(-1)^m}{m!}\sum_{k\geqslant 1}(-1)^k\binom{m-1}{k-1}\ln k$$

58.〔$M21$〕在等式(37)的记号下，证明"q 项式定理"：

$$(1+x)(1+qx)\cdots(1+q^{n-1}x)=\sum_k\binom{n}{k}_q q^{k(k-1)/2}x^k$$

59.〔$M25$〕一数列 A_{nk}，$n\geqslant 0$，$k\geqslant 0$，满足关系式 $A_{n0}=1$，$A_{0k}=\delta_{0k}$，$A_{nk}=A_{(n-1)k}+A_{n(k-1)}+\binom{n}{k}$。试求此 A_{nk}。

▶60.〔24〕我们已经看到，$\binom{n}{k}$ 是从 n 个事物中每次取 k 的组合的数目，即是从 n 个事物的集合中选出 k 个不同事物的方法的种数。有重复组合类似于通常的组合，所不同的只是我们可以任意多次地选择每一事物。于是，如果我们考虑有重复组合的话，则式(1)将被扩充成也包括 aaa，aab，aac，aad，aae，abb 等等组合。如果允许重复，则从 n 个事物取 k 个的组合，共有多少种？

61.〔$M25$〕计算和数

$$\sum_k\begin{bmatrix}n+1\\k+1\end{bmatrix}\begin{Bmatrix}k\\m\end{Bmatrix}(-1)^k$$

由此得到一个与等式(51)配对的公式。

▶62.〔$M23$〕正文中已经给出了一系列求两个二项式系数的诸乘积之和的公式。在关于三个二项式系数的诸乘积之和中，下列恒等式和习题31的恒等式似乎是最有用的：

$$\sum_k(-1)^k\binom{l+m}{l+k}\binom{m+n}{m+k}\binom{n+l}{n+k}=\frac{(l+m+n)!}{l!m!n!}\quad\text{整数 } l,\ m,\ n\geqslant 0$$

(注意这个和式包括了正的和负的 k 值)。试证明这个恒等式〔提示：有一个非常简短的证明，它通过应用习题31的结果开始〕。

63.〔46〕编出计算机程序，来简化关于二项式系数的和式。

▶64.〔$M22$〕证明 $\begin{Bmatrix}n\\m\end{Bmatrix}$ 是把 n 个元素的集合划分成 m 个不相交非空子集的方法的种

数。例如，集合$\{1, 2, 3, 4\}$可以有$\left\{ {4 \atop 2} \right\}=7$种方法划分成两个子集：$\{1, 2, 3\}\{4\}$；$\{1, 2, 4\}\{3\}$；$\{1, 3, 4\}\{2\}$；$\{2, 3, 4\}\{1\}$；$\{1, 2\}\{3, 4\}$；$\{1, 3\}\{2, 4\}$；$\{1, 4\}\{2, 3\}$。提示：用事实

$$\left\{ {n \atop m} \right\}=m\left\{ {n-1 \atop m} \right\}+\left\{ {n-1 \atop m-1} \right\}$$

来证明。注意本题的结果给我们提供了一种记忆的手段，以便使我们记住关于斯特林数的两种记号“$\{\ \}$”与“$(\ )$”之间的差别，因为“$\{\ \}$”一般地也用于表示集合。而另一种斯特林数$\left[{n \atop k} \right]$也有一个组合的解释：$\left[{n \atop k} \right]$是有$k$个“循环”的$n$个字母的排列数；请参看1.3.3小节。

1.2.7 调和数

在我们后面的工作中，下列和数具有巨大的重要性：

$$H_n=1+\frac{1}{2}+\frac{1}{3}+\cdots+\frac{1}{n}=\sum_{1\leqslant k\leqslant n}\frac{1}{k}\qquad n\geqslant 0 \tag{1}$$

在经典数学中，这一和数并不经常出现，对它也没有一个标准的记号；但在算法分析中，它几乎每时每刻都与我们的探讨有关，而且我们将自始至终地使用符号H_n来表示这一数量（除了H_n外，在数学著作中偶尔也使用h_n和S_n的记号。字母H代表“调和”，而且我们把H_n叫做调和数，因为（1）习惯上叫做调和级数）。

乍看，当n取充分大的值时，H_n未必会得到太大的值，因为我们总是加上越来越小的数。但实际上，不难看出，只要n取得充分大，则H_n就能取到我们所需要的不论多大的值，因为我们有以下的规则：

$$H_{2^m}\geqslant 1+\frac{m}{2} \tag{2}$$

为证明这一规则，试观察下列的事实：对于$m>0$，

$$\begin{aligned}H_{2^{m+1}}&=H_{2^m}+\frac{1}{2^m+1}+\frac{1}{2^m+2}+\cdots+\frac{1}{2^{m+1}}\\&>H_{2^m}+\frac{1}{2^{m+1}}+\frac{1}{2^{m+1}}+\cdots+\frac{1}{2^{m+1}}\\&=H_{2^m}+\frac{1}{2}\end{aligned}$$

所以随着m增加1，则等式（2）左边至少增加$\frac{1}{2}$。

重要的是对于H_n的值，要给出比等式（2）所给出的更为详细的说明。H_n的近似数量是一个熟知的数量（至少在数学界里是这样），它可以表达如下：

$$H_n=\ln n+\gamma+\frac{1}{2n}-\frac{1}{12n^2}+\frac{1}{120n^4}-\epsilon\qquad 0<\epsilon<\frac{1}{252n^6} \tag{3}$$

这里，$\gamma=0.57721\ 56649\cdots$是欧拉常数。在附录B的表中，给出了对于小的$n$的$H_n$值，以及$\gamma$的一个40位的值。在1.2.11.2小节，我们将证明等式（3）。

因此，H_n相当接近n的自然对数。习题7说明了H_n略具某些对数的行迹。

在某种意义下，当 n 取充分大值时，H_n “仅仅勉强地”趋向无穷，因为可以证明和数

$$1+\frac{1}{2^r}+\frac{1}{3^r}+\cdots+\frac{1}{n^r} \tag{4}$$

对于所有的 n，当 r 是大于 1 的任何实数值指数时，都保持有界(见习题 3)。我们用 $H_n^{(r)}$ 来表示等式（4）中的和数。

当等式（4）的 r 为 2 或更大时，$H_n^{(r)}$ 的值是相当接近于它的极大值 $H_\infty^{(r)}$ 的，除非 n 的值很小。数量 $H_\infty^{(r)}$ 在数学上作为黎曼的仄塔函数而非常闻名：

$$H_\infty^{(r)}=\zeta(r) \tag{5}$$

当 r 是一个偶整数时，已知 $\zeta(r)$ 的值等于

$$H_\infty^{(r)}=\frac{1}{2}|B_r|\frac{(2\pi)^r}{r!} \tag{6}$$

其中 B_r 是一个贝努利数，见（1.2.11.2 小节和附录 B）。特别是，

$$H_\infty^{(2)}=\frac{\pi^2}{6},\quad H_\infty^{(4)}=\frac{\pi^4}{90},\quad H_\infty^{(6)}=\frac{\pi^6}{945},\quad H_\infty^{(8)}=\frac{\pi^8}{9450} \tag{7}$$

有关的讨论和证明，请参看康·诺普著，罗·塞·希·扬（R. C. H. Young）的英译本《无穷级数的理论和应用》(格拉斯哥：布莱其，1951) 第 32.4 节。

现在让我们来考虑关于求和的一些重要的性质。首先，

$$\sum_{1\leqslant k\leqslant n}H_k=(n+1)H_n-n \tag{8}$$

这是从和数的简单变换得来的：

$$\sum_{1\leqslant k\leqslant n}\sum_{1\leqslant j\leqslant k}\frac{1}{j}=\sum_{1\leqslant j\leqslant n}\sum_{j\leqslant k\leqslant n}\frac{1}{j}=\sum_{1\leqslant j\leqslant n}\frac{n+1-j}{j}$$

公式（8）是和数 $\sum_{1\leqslant k\leqslant n}\binom{k}{m}H_k$ 的一个特殊情况，我们现在来确定这一和数。这里所用的“窍门”叫做部分求和。当数量 $\sum a_k$ 和 $(b_{k+1}-b_k)$ 有简单的形式时，它是一个确定 $\sum a_kb_k$ 的有用的技术（见习题 10)。在现在情况下，我们注意

$$\binom{k}{m}=\binom{k+1}{m+1}-\binom{k}{m+1}$$

且因此

$$\binom{k}{m}H_k=\binom{k+1}{m+1}\left(H_{k+1}-\frac{1}{k+1}\right)-\binom{k}{m+1}H_k$$

从而

$$\begin{aligned}
&\sum_{1\leqslant k\leqslant n}\binom{k}{m}H_k\\
&\quad=\left(\binom{2}{m+1}H_2-\binom{1}{m+1}H_1\right)+\cdots\\
&\qquad+\left(\binom{n+1}{m+1}H_{n+1}-\binom{n}{m+1}H_n\right)-\sum_{1\leqslant k\leqslant n}\binom{k+1}{m+1}\frac{1}{k+1}\\
&\quad=\binom{n+1}{m+1}H_{n+1}-\binom{1}{m+1}H_1-\frac{1}{m+1}\sum_{0\leqslant k\leqslant n}\binom{k}{m}+\frac{1}{m+1}\binom{0}{m}
\end{aligned}$$

应用等式 1.2.6-11，就得到所希望的公式：

$$\sum_{1\leqslant k\leqslant n}\binom{k}{m}H_k=\binom{n+1}{m+1}\left(H_{n+1}-\frac{1}{m+1}\right) \tag{9}$$

（上边的推导和最后的结果同定积分计算中确定 $\int_1^n x^m\ln x\,dx$ 有些类似）。

让我们通过考虑一个不同类型的和数 $\sum_k\binom{n}{k}x^kH_k$ 来结束这一小节。为了简洁起见，我们暂时把这和数表示为 S_n。我们发现

$$S_{n+1}=\sum_k\left(\binom{n}{k}+\binom{n}{k-1}\right)x^kH_k=S_n+x\sum_k\binom{n}{k-1}x^{k-1}\left(H_{k-1}+\frac{1}{k}\right)$$

$$=S_n+xS_n+\frac{1}{n+1}\sum_{k\geqslant 1}\binom{n+1}{k}x^k$$

因此 $S_{n+1}=(x+1)S_n+((x+1)^{n+1}-1)/(n+1)$，而且我们有

$$\frac{S_{n+1}}{(x+1)^{n+1}}=\frac{S_n}{(x+1)^n}+\frac{1}{n+1}-\frac{1}{(n+1)(x+1)^{n+1}}$$

这一等式，连同 $S_1=x$ 的事实一起，向我们说明了

$$\frac{S_n}{(x+1)^n}=H_n-\sum_{1\leqslant k\leqslant n}\frac{1}{k(x+1)^k} \tag{10}$$

剩下的和是 $\ln(1/(1-1/(x+1)))=\ln(1+1/x)$ 的无穷级数的一部分；而且当 $x>0$ 时，这个级数收敛，差数为

$$\sum_{k>n}\frac{1}{k(x+1)^k}<\frac{1}{(n+1)(x+1)^{n+1}}\sum_{k\geqslant 0}\frac{1}{(x+1)^k}=\frac{1}{(n+1)(x+1)^nx}$$

这就证明了下列定理：

定理 A．若 $x>0$，则

$$\sum_{1\leqslant k\leqslant n}\binom{n}{k}x^kH_k=(x+1)^n\left(H_n-\ln\left(1+\frac{1}{x}\right)\right)+\epsilon$$

其中 $0<\epsilon<1/x(n+1)$。 ∎

习题

1．〔01〕 H_0，H_1 和 H_2 等于什么？

2．〔13〕 说明正文中用来证明 $H_{2^m}\geqslant 1+m/2$ 的简单论证，只要稍加修改，即可证明 $H_{2^m}\leqslant 1+m$。

3．〔*M*21〕 推广上题所使用的论证，来证明：假定 $r>1$，则 $H_n^{(r)}$ 对所有的 n 保持有界；并请求出一个上界来。

▶4．〔10〕 对于所有的正整数 n，下列命题中哪一个为真？

（a）$H_n<\ln n$；（b）$H_n>\ln n$；（c）$H_n>\ln n+\gamma$。

5．〔15〕 利用附录 B 中的表，给出 H_{10000} 的值到 15 位小数。

6．〔*M*15〕 证明调和数直接同上节所介绍的斯特林数有关，事实上

$$H_n=\left[\begin{matrix}n+1\\2\end{matrix}\right]\quad n!$$

7. 〔*M*21〕 设$T(m,n)=H_m+H_n-H_{mn}$。(a)说明当m或n增加时，$T(m,n)$就减小（假定m和n皆为正）。(b)计算$m,\ n>0$时$T(m,\ n)$的极小值和极大值。

8. 〔*M*18〕 将等式（8）与$\Sigma_{1\leqslant k\leqslant n}\ln k$进行比较，估计它们的差，将其表示成为$n$的一个函数。

▶9. 〔*M*18〕 仅当$x>0$时，定理A才可应用。当$x=-1$时，所考虑的和数之值等于什么？

10. 〔*M*20〕 （部分求和）在习题1.2.4-42和等式(9)的推导中，我们已经使用了一般部分求和方法的特殊情况。试证一般公式

$$\sum_{1\leqslant k\leqslant n}(a_{k+1}-a_k)b_k=a_nb_n-a_1b_1-\sum_{1\leqslant k<n}a_{k+1}(b_{k+1}-b_k)$$

▶11. 〔*M*21〕 利用部分求和，计算

$$\sum_{1<k\leqslant n}\frac{1}{k(k-1)}H_k$$

▶12. 〔*M*10〕 计算$H_{\infty}^{(100)}$精确到至少100位小数。

13. 〔*M*22〕 证明恒等式

$$\sum_{1\leqslant k\leqslant n}\frac{x^k}{k}=H_n+\sum_{1\leqslant k\leqslant n}\binom{n}{k}\frac{(x-1)^k}{k}$$

（特别注意$x=0$的特殊情况，它向我们提供了一个与习题1.2.6-48有关的恒等式）。

14. 〔*M*22〕证明

$$\sum_{1\leqslant k\leqslant n}\frac{H_k}{k}=\frac{1}{2}(H_n^2+H_n^{(2)})$$

并计算$\Sigma_{1\leqslant k\leqslant n}H_k/(k+1)$。

▶15. 〔*M*23〕 借助于n和H_n来表达$\Sigma_{1\leqslant k\leqslant n}H_k^2$。

16. 〔18〕 借助于调和数来表达和式$1+\frac{1}{3}+\cdots+1/(2n+1)$。

17. 〔*M*24〕 （爱·华林(E. Waring)，1782）设p为一奇质数，证明H_{p-1}的分子能为p所整除。

18. 〔*M*33〕 （约·赛尔夫利齐）能整除$1+\frac{1}{3}+\cdots+1/(2n-1)$的分子的2的最高次幂是多少？

▶19. 〔*M*30〕 列出使H_n为一整数的所有非负整数n〔提示：如果$H_n=\text{奇}/\text{偶}$，则它不可能是一个整数〕。

20. 〔*HM*22〕 有一个解决求和问题的，如同导出这一节中的定理A那样的分析方法：若$f(x)=\Sigma_{k\geqslant 0}a_kx^k$，而且这一级数对于$x=x_0$收敛，则可证明

$$\sum_{k\geqslant 0}a_kx_0^kH_k=\int_0^1\frac{f(x_0)-f(x_0y)}{1-y}dy$$

21. 〔$M24$〕 计算$\sum_{1\leqslant k\leqslant n}H_k/(n+1-k)$。

22. 〔$M28$〕 计算$\sum_{1\leqslant k\leqslant n}H_kH_{n+1-k}$。

▶23. 〔$HM20$〕 通过考虑函数$\Gamma'(x)/\Gamma(x)$，来说明我们怎样才能把H_n自然地推广到非整数的n值。你可以利用$\Gamma'(1)=-\gamma$的事实和预先利用下一习题。

24. 〔$HM21$〕 证明

$$xe^{\gamma x}\prod_{k\geqslant 1}\left(\left(1+\frac{x}{k}\right)e^{-x/k}\right)=\frac{1}{\Gamma(x)}$$

(考虑这无穷乘积的部分乘积)。

1.2.8 斐波那契数

序列

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\cdots \tag{1}$$

中，每个数都是它前两者之和。这个序列在我们后面将要研究的许多表面上似乎无关的算法中，起着重要的作用。兹以F_n来表示这个序列的数，并将其形式地定义为

$$F_0=0,\ F_1=1,\ F_{n+2}=F_{n+1}+F_n,\quad n\geqslant 0 \tag{2}$$

这个著名的序列起源于1202年，是由比萨的伦纳德首次提出的。此人有时叫做伦纳德·斐波那契（菲利亚斯·波那契，波那契的儿子），他所著的《珠算的书》(Liber Abbaci)包含下列的习题："在一年的时间里，一对兔子可以生出多少对兔子来？"为解决这个问题，我们被告知：假定每对兔子每个月生出新的一对兔子来，新的每一对兔子在一个月里就变成为能生育的；其次，这些兔子将都不死去。这样，在一个月之后，就将有2对兔子；两个月后，将有3对；随后的一个月原来的一对和在头一个月生出的一对都将生出新的一对，因此共有5对兔子；如此类推。

斐波那契是中世纪以来最伟大的欧洲数学家。他研究了阿尔科瓦利兹米(在此人之后，就有了"algorithm"(即算法)一词，见1.1小节)的工作，而且他对于算术和几何都作出了许多开创性的贡献。斐波那契的著作1857年曾再版过〔鲍·邦康帕尼(B. Boncompagni)：《比萨的伦纳德的文集》(Scritti di Leonardo Pisano)，罗马，1857～1862，2卷；F_n出现在第1卷第283～285页上〕。当然，兔子问题并不是作为对生物学和人口增长问题的实际应用，它只是加法方面的一个习题。事实上，它还是一个关于加法的比较好的计算机习题（参照习题3）。斐波那契写道："这种加法可以以如此顺序无穷多个月地进行下去。"

同样的序列也出现在1611年开普勒的著作中，其出现与所谓"叶序"问题，即关于植物生长中的树叶和花朵之排列的研究有关。开普勒大概不知道斐波那契关于这一序列的简单的陈述。大概是由于类似于兔子问题的原始假定的原因，斐波那契数的性质常常引起人们的关注。

1844年，当加·拉姆（G. Lamé）用斐波那契序列研究欧几里得算法的效率时，头一次指出了F_n与算法之间的密切联系。他证明了，如果算法1.1E中的数m，n不大于F_k，则步骤E2将至多执行$k+1$次。这是斐波那契序列的头一次的实际应用。在以后的五十年期间，数学家埃·路卡斯得到了关于斐波那契数的非常深刻的结果。特别是，他利用它们证明了39位的数$2^{127}-1$是质数。路卡斯对序列F_n赋予了"斐波那契数"的名称，这

个名称一直沿用至今。

在 1.2.1 小节（等式（3）和习题 4），我们简略地考察了斐波那契序列。在那里，我们发现当 n 是一正整数且当

$$\phi=\frac{1}{2}(1+\sqrt{5}) \tag{3}$$

时，$\phi^{n-2}\leqslant F_n\leqslant\phi^{n-1}$。我们立刻就会看到，这个量 ϕ 如何密切地同斐波那契数联系在一起！

数 ϕ 本身有着一段非常有趣的历史。欧几里得把它叫做“中末比”。如果 A 对 B 的比是 ϕ 的话，则 A 对 B 的比等于（$A+B$）对 A 的比。文艺复兴时期的作者把它叫做“神圣比例”。而在上一世纪，人们普遍地把它叫做“黄金分割”。在艺术界，ϕ 对 1 的比例被说成是艺术上最惹人喜爱的比例，而且这一意见从计算机程序设计艺术的观点来看也是符合的。关于 ϕ 的历史，请参看哈·斯·麦·考克斯特（H. S. M. Coxeter）的精采论文《黄金分割，叶序与维佐夫博奕》(The Golden Section, Phyllotaxis and Wythoff's Game, Scripta Math. 19(1953), 135～143；也可见马丁·加德纳（Martin Gardner)编著的《科学的美国人的第二本数学游戏和娱乐集》（The 2nd Scientific American Book of Mathematical Puzzles and Diversions）第 8 章（纽约：西蒙和舒斯特，1961)。

我们在这一小节里使用的记号是有点不庄重的。在大多数专门的数学著作里，F_n 改成叫做 u_n，而 ϕ 改叫做 τ。我们的记号几乎普遍地为趣味性的数学著作（以及某些奇特的著作！）所通用，而且迅速地得到越来越广泛的使用。记号 ϕ 来自于古希腊艺术家菲迪亚斯(Phidias)的名字。他曾说过，在他的雕塑中经常要使用“黄金分割”。记号 F_n 是根据斐波那契季刊（Fibonacci Quarterly）杂志（1963 年开始出版）中的用法而使用的。在该杂志中，读者可以找到大量的有关斐波那契序列的问题。关于斐波那契序列的经典著作方面，有很完全的参考文献，见伦·尤·迪克森（L. E. Dickson）著的《数论历史》(History of the Theory of Numbers)(纽约：切尔西，1952）第 1 卷第 17 章。

斐波那契数满足许多有趣的恒等式，其中有些出现在本小节末尾的习题中。最普遍引用的关系之一，就是由琼·多·卡西尼（J. D. Cassini）〔Histoire Acad. Roy. Paris 1 (1680), 201〕给出的关系式：

$$F_{n+1}F_{n-1}-F_n^2=(-1)^n \tag{4}$$

这个公式是容易用归纳法证明的。但更为巧妙的证明方法，可由矩阵恒等式

$$\begin{pmatrix}1&1\\1&0\end{pmatrix}^n=\begin{pmatrix}F_{n+1}&F_n\\F_n&F_{n-1}\end{pmatrix} \tag{5}$$

的简单归纳证明开始。然后我们取这个恒等式两边的行列式即得所证。

关系（4）说明 F_n 与 F_{n+1} 是互质的，因为其任何公因子都将是 $(-1)^n$ 的一个因子。

由定义（2），我们立即推出

$$F_{n+3}=F_{n+2}+F_{n+1}=2F_{n+1}+F_n;\quad F_{n+4}=3F_{n+1}+2F_n$$

而且，一般说来，通过归纳法可以得到：对于任意正整数 m，有

$$F_{n+m}=F_mF_{n+1}+F_{m-1}F_n \tag{6}$$

如果我们在等式（6）中取m是n的一个倍数，则我们归纳地推出

$$F_{nk}\text{是}F_k\text{的一个倍数}$$

由此可见，每第三个数是偶数，每第四个数是3的倍数，每第五个数是5的倍数，等等。

事实上，还有比这多得多的事实成立。如果我们用$\gcd(m, n)$来表示m和n的最大公因子，则我们就有下列相当令人惊奇的定理：

定理A （埃·路卡斯 1876）一个数整除F_m和F_n，当且仅当它是F_d的一个因子，其中$d=\gcd(m, n)$。特别地，

$$\gcd(F_m, F_n)=F_{\gcd(m,n)} \tag{7}$$

证明 通过使用欧几里得算法来证明这一结果。我们注意到，由于等式（6），F_m和F_n的任何公因子也是F_{n+m}的一个因子；而且，反过来，F_{n+m}和F_n的任何公因子是F_mF_{n+1}的一个因子。由于F_{n+1}与F_n互质，因而F_{n+m}和F_n的公因子也整除F_m。这样，我们已经证明了：对任意数d，

$$d\text{整除}F_m\text{和}F_n\text{，当且仅当，}d\text{整除}F_{n+m}\text{和}F_n \tag{8}$$

我们现在将证明，任何使得命题（8）成立而且$F_0=0$的序列F_n必满足定理A。

首先，显然，通过对k用归纳法，可以把命题（8）推广成规则

d整除F_m和F_n，当且仅当，d整除F_{m+kn}和F_n，其中k是任意非负整数。这个结果可以更简洁地叙述成为：

$$d\text{整除}F_{m \bmod n}\text{和}F_n\text{，当且仅当，}d\text{整除}F_m\text{和}F_n \tag{9}$$

现在如果r是m除以n之后的余数，即是，如果$r=m \bmod n$，则F_m和F_n的公因子是F_n和F_r的公因子。这就得出：贯穿算法1.1E的处理，F_m，F_n的公因子的集合，当m，n改变时，仍保持不变。最后，当$r=0$时，公因子只不过就是$F_0=0$和$F_{\gcd(m,n)}$的因子。

关于斐波那契数的大多数重要的结果，都可以借助于Φ来表示F_n而推导出来，我们现在就来进行推导。在下面的推导中，我们所使用的方法是极为重要的，专长数学的读者应当精心地研究之。在下一小节，我们还将详细地研究这同一方法。

我们通过建立无穷级数

$$\begin{aligned} G(z)&=F_0+F_1z+F_2z^2+F_3z^3+F_4z^4+\cdots \\ &=z+z^2+2z^3+3z^4+\cdots \end{aligned} \tag{10}$$

开始。我们没有理由事先预料这个无穷和是存在的或者$G(z)$是断然值得研究的——让我们对它抱着乐观的态度来看看，如果函数$G(z)$存在的话，则我们能对它作出什么结论。这样一种处理的优点在于，$G(z)$是一个同时表示整个斐波那契序列的单个的量；而且，如果我们发现$G(z)$是一个“已知的”函数，则其系数就可以确定下来。$G(z)$叫做对于序列$\langle F_n\rangle$的生成函数。

现在，让我们如下地继续来观察$G(z)$：

$$\begin{aligned} zG(z)&=F_0z+F_1z^2+F_2z^3+F_3z^4+\cdots \\ z^2G(z)&=F_0z^2+F_1z^3+F_2z^4+\cdots \end{aligned}$$

通过相减，得

$$\begin{aligned} (1-z-z^2)G(z)=&F_0+(F_1-F_0)z+(F_2-F_1-F_0)z^2 \\ &+(F_3-F_2-F_1)z^3+(F_4-F_3-F_2)z^4+\cdots=z \end{aligned}$$

因为由 F_n 的定义，所有后边的项都为 0 。因此，我们看到，如果$G(z)$存在，则

$$G(z)=z/(1-z-z^2) \tag{11}$$

事实上，这个函数可以展开成 z 的无穷级数（台劳级数)。回溯原设，我们推知，等式(11)的幂级数展开式的系数必定就是斐波那契数。

我们现在已经能处理$G(z)$并找出关于斐波那契序列的更多结果了。分母 $1-z-z^2$ 是一个二次方程，有两个根$\frac{1}{2}(-1\pm\sqrt{5})$。经过少量的计算后，发现$G(z)$可以通过部分分式的方法展开成如下形式：

$$G(z)=\frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z}-\frac{1}{1-\hat{\phi}z}\right) \tag{12}$$

其中

$$\hat{\phi}=1-\phi=\frac{1}{2}(1-\sqrt{5}) \tag{13}$$

数量 $1/(1-\phi z)$是无穷几何级数 $1+\phi z+\phi^2z^2+\cdots$的和，所以有

$$G(z)=\frac{1}{\sqrt{5}}(1+\phi z+\phi^2z^2+\cdots-1-\hat{\phi}z-\hat{\phi}^2z^2-\cdots)$$

现在来观察 z^n 的系数，它必然等于 F_n，因而我们求得

$$F_n=\frac{1}{\sqrt{5}}(\phi^n-\hat{\phi}^n) \tag{14}$$

这是对于斐波那契数的一个重要的“封闭形式”表示，它最先是在十八世纪初由阿·德·莫尤勒（A. de. Moivre）发现的(见德·莫尤勒的《分析集锦》(Miscellanea Analytica)，(伦敦：1730)，26～42，其中对于一般线性递推的解实际上是从我们推导（14）的办法得到)。

我们也可以直接叙述等式（14)，并且用归纳法来证明它。但是，上面冗长的推导的目的，是要利用生成函数的重要方法——一个对于解决许多问题都有价值的技巧——来说明起初怎样才有可能发现这个等式。

由等式（14）可以证明许多事情。首先，我们注意到，$\hat{\phi}$ 是一个负数(−0.61803…)，其绝对值小于 1 。所以当 n 取充分大时，$\hat{\phi}^n$ 就变得很小。事实上，$\hat{\phi}^n/\sqrt{5}$ 总是足够地小，所以我们有

$$F_n=\phi^n/\sqrt{5}\text{ 舍入到最接近的整数} \tag{15}$$

另一些结果可以从$G(z)$直接得到。例如，

$$G(z)^2=\frac{1}{5}\left(\frac{1}{(1-\phi z)^2}+\frac{1}{(1-\hat{\phi}z)^2}-\frac{2}{1-z-z^2}\right) \tag{16}$$

而且$G(z)^2$中 z^n 的系数是 $\sum_{0\leqslant k\leqslant n}F_kF_{n-k}$。因此我们就推出

$$\sum_{0\leqslant k\leqslant n}F_kF_{n-k}=\frac{1}{5}((n+1)(\phi^n+\hat{\phi}^n)-2F_{n+1})$$

$$=\frac{1}{5}((n+1)(F_n+2F_{n-1})-2F_{n+1})=\frac{n-1}{5}F_n+\frac{2}{5}nF_{n-1} \tag{17}$$

(本推演中的第二步由习题 11 的结果得出)。

习题

1.〔10〕 在伦纳德·斐波那契的问题当中，在第 k 个月之后，有多少对兔子？问题是一年之后该有多少对兔子，答案是什么？

▶2.〔20〕 由等式（15）看来，F_{1000} 的近似值等于多少？（利用附录B的表中给出的对数）。

3.〔34〕 写出某个计算机上用来计算和打印 F_1 到 F_{1000} 的程序（关于所需处理的数的大小，请参照上题）。

▶4.〔14〕 求出使 $F_n=n$ 的所有的 n。

5.〔20〕 求出使 $F_n=n^2$ 的所有的 n。

6.〔$HM10$〕 证明等式（5）。

▶7.〔15〕 若 n 不是一个质数，则 F_n 也不是一个质数（仅有一个例外）。请证明这一命题并找出这一个例外。

8.〔15〕 在许多情况下，对于负数的 n 也定义 F_n 是方便的。其方法是对于所有的整数 n，假定 $F_{n+2}=F_{n+1}+F_n$。试探索这种可能性。F_{-1} 等于什么？F_{-2} 等于什么？能否以简单的形式，借助于 F_n 来表达 F_{-n}？

9.〔$M20$〕 利用上题的约定，判定当允许下标取任意整数时，等式（4），（6），（14）和（15）是否仍然成立？

10.〔15〕 $\phi^n/\sqrt{5}$ 是大于 F_n 还是小于 F_n？

11.〔$M20$〕 证明对于所有整数 n，$\phi^n=F_n\phi+F_{n-1}$，$\hat{\phi}^n=F_n\hat{\phi}+F_{n-1}$。

▶12.〔$M26$〕 通过规则

$$\mathfrak{F}_0=0,\quad \mathfrak{F}_1=1,\quad \mathfrak{F}_{n+2}=\mathfrak{F}_{n+1}+\mathfrak{F}_n+F_n$$

来定义“二阶”斐波那契序列，借助于 F_n 来表达 $\mathfrak{F}_n$〔提示：利用生成函数〕。

▶13.〔$M22$〕 借助于斐波那契数来表示以下的序列：

a）$a_0=r$，$a_1=s$，$a_{n+2}=a_{n+1}+a_n$，$n\geqslant 0$。

b）$b_0=0$，$b_1=1$，$b_{n+2}=b_{n+1}+b_n+c$，$n\geqslant 0$。

14.〔$M28$〕 设 m 是一个固定的正整数。给定

$$a_0=0,\quad a_1=1,\quad a_{n-2}=a_{n-1}+a_n+\binom{n}{m}$$

求 a_n。

15.〔$M22$〕 设 $f(n)$，$g(n)$ 是任意函数。设

$$a_0=0,\quad a_1=1,\quad a_{n+2}=a_{n+1}+a_n+f(n)$$

$$b_0=0,\quad b_1=1,\quad b_{n+2}=b_{n+1}+b_n+g(n)$$

$$c_0=0,\quad c_1=1,\quad c_{n+2}=c_{n+1}+c_n+xf(n)+yg(n)$$

试用 x，y，a_n，b_n 和 F_n 来表示 c_n。

▶16.〔$M20$〕 如果从次对角线来观察，就可以发现斐波那契数隐涵地出现在帕斯卡尔三角中。即是，请证明下列的二项式系数之和是一个斐波那契数：

$$\sum_{0\leqslant k\leqslant n}\binom{n-k}{k}$$

17.〔$M24$〕 利用习题 8 的约定，证明等式（4）的下列推广：

$$F_{n+k}F_{m-k}-F_nF_m=(-1)^nF_{m-n-k}F_k$$

18.〔20〕 $F_n^2+F_{n+1}^2$ 是否总是一个斐波那契数？

19.〔$M27$〕 $\cos 36^\circ$ 等于什么？

20.〔$M16$〕 用斐波那契数来表示 $\Sigma_{0\leqslant k\leqslant n}F_k$。

21.〔$M25$〕 $\Sigma_{0\leqslant k\leqslant n}F_kx^k$ 等于什么？

▶22.〔$M20$〕 证明 $\Sigma_k\binom{n}{k}F_{m+k}$ 是一个斐波那契数。

23.〔$M23$〕 推广上一习题，证明 $\Sigma_k\binom{n}{k}F_t^kF_{t-1}^{n-k}F_{m+k}$ 总是一个斐波那契数。

24.〔$HM20$〕 计算 $n\times n$ 行列式

$$\begin{pmatrix} 1 & -1 & 0 & 0 & \cdots & 0 & 0 & 0 \\ 1 & 1 & -1 & 0 & \cdots & 0 & 0 & 0 \\ 0 & 1 & 1 & -1 & \cdots & 0 & 0 & 0 \\ \vdots & & & & & & & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 & 1 & -1 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 1 \end{pmatrix}$$

25.〔$M21$〕 证明

$$2^nF_n=2\sum_{k\text{ 奇}}\binom{n}{k}5^{(k-1)/2}$$

▶26.〔$M20$〕 利用上题证明：当 p 是奇质数时，有 $F_p\equiv 5^{(p-1)/2}(\text{modulo } p)$。

27.〔$M20$〕 利用上题证明：当 p 是不等于 5 的质数时，则 F_{p-1} 或 F_{p+1} 之一是（不都是）p 的倍数。

28.〔$M21$〕 $F_{n+1}-\phi F_n$ 等于什么？

▶29.〔$M23$〕 （“斐波那契系数”）以类似于二项式系数的方式定义

$$\left(\!\!\binom{n}{k}\!\!\right)=\frac{F_nF_{n-1}\cdots F_{n-k+1}}{F_kF_{k-1}\cdots F_1}=\prod_{1\leqslant j\leqslant k}\binom{F_{n-k+j}}{F_j}$$

（a）造出一个对于 $0\leqslant n\leqslant 6$ 的 $\left(\!\!\binom{n}{k}\!\!\right)$ 的表来。（b）证明

$$\left(\!\!\binom{n}{k}\!\!\right)=F_{k-1}\left(\!\!\binom{n-1}{k}\!\!\right)+F_{n-k+1}\left(\!\!\binom{n-1}{k-1}\!\!\right)$$

▶30.〔$M38$〕 （达·贾登(D. Jarden)）斐波那契数的 m 次幂的序列满足一个递推关系。在这个关系中，每一项依赖于它前边的 $m+1$ 项。即可证明：如果 $m>0$，则

$$\sum_k\left(\!\!\binom{m}{k}\!\!\right)(-1)^{\lceil(m-k)/2\rceil}F_{n+k}^{m-1}=0$$

例如，当 $m=3$ 时，我们得到恒等式

$$F_n^2-2F_{n+1}^2-2F_{n+2}^2+F_{n+3}^2=0$$

31.〔$M20$〕 设 $\psi=\phi-1=1/\phi$。证明 $(F_{2n}\psi)\bmod 1=1-\psi^{2n}$ 且 $(F_{2n+1}\psi)\bmod 1=\psi^{2n+1}$。

32.〔$M24$〕 一个斐波那契数除以另一个斐波那契数的余数，是±斐波那契数：证明

$$F_{mn+r}\equiv F_r,\ (-1)^{r+1}F_{n-r},\ (-1)^nF_r,\ \text{或}(-1)^{r+1+n}F_{n-r}(\text{modulo } F_n)$$

按照$m \mod 4$分别地$=0$，1，2，或3。

33.〔$HM24$〕给定$z=\pi/2+i\ln\phi$，证明$\sin(nz)/\sin z=i^{1-n}F_n$。

▶34.〔$M24$〕（斐波那契数系）设记号$k\gg m$表示$k\geqslant m+2$。证明：每一个正整数n有唯一的表示$n=F_{k_1}+F_{k_2}+\cdots+F_{k_r}$，其中$k_1\gg k_2\gg\cdots\gg k_r\gg 0$。

35.〔$M24$〕（ϕ数系）考虑以ϕ为底用数字0和1写成的实数(于是$100.1=\phi^2+\phi^{-1}$)，证明：有无穷多的方式来表示数1（例如，$1=.11=.011111\cdots$）；但是如果我们要求不得出现相邻的两个1，以及不得出现无穷序列“01010101…”，则每个数就有唯一的表示。

▶36.〔$M32$〕（斐波那契串）设$S_1=$“a”，$S_2=$“b”，且$S_{n+2}=S_{n+1}S_n$，$n>0$。换言之，S_{n+2}是通过把S_n放在S_{n+1}的右边形成的。我们有$S_3=$“ba”，$S_4=$“bab”，$S_5=$“$babba$”，等等。显然，S_n有F_n个字母。试探索S_n的性质（何处有重复字母出现？你能预测S_n的第k个字母是什么吗？b的密度是多少？等等）。

▶37.〔$M35$〕（罗·尤·加斯克尔(R. E. Gaskell)，迈·詹·惠尼罕（M. J. Whinihan））在下列的游戏中，有两个选手在比赛：有一迭纸牌，共n张。第一个选手可取任意张，但不得取全部。从此开始，两个选手交替地取。每个人均可以取一张或一张以上，但不得取多于对方刚才取的数目的一倍。谁取最后的片子即谁胜(例如，假设$n=11$。选手A取3张；选手B可以顶多取6张，而他取1张。剩下7张；选手A可以取1张或2张，而他取2张；选手B可以取到4张，而他取1张。现在还有4张；选手A取1张，选手B必须至少取1张。因此选手A在下一次取时就得胜了)。

如果开始时有1000张，对于头一个选手，最好的取法为何？

38.〔35〕写出一个计算机程序，来做上题中所述的游戏，而且要使游戏最优地进行。

39.〔$M24$〕给定$a_0=0$，$a_1=1$，$a_{n+2}=a_{n+1}+6a_n$，试找出a_n的封闭形式的表达式。

1.2.9 生成函数

每当我们想要得到关于数列$\langle a_n\rangle=a_0$，a_1，a_2，…的任何信息时，我们可以借助一个参数z来建立无穷和：

$$G(z)=a_0+a_1z+a_2z^2+\cdots=\sum_{n\geqslant 0}a_nz^n \tag{1}$$

然后，我们就可以试验来得到关于函数G的信息。函数G是一个表示整个序列$\langle a_n\rangle$的单一的量。如果序列$\langle a_n\rangle$已经归纳地定义（即是说，如果a_n已经通过a_0，a_1，…，a_{n-1}加以定义），那末，这就是一个很重要的优点。其次，假定等式(1)中的无穷和对于某些z值存在，则通过使用微分技术，我们就可以从函数$G(z)$回收a_0，a_1，…的值。

$G(z)$叫做对于序列a_0，a_1，a_2，…的生成函数。生成函数的利用，为我们引进了全新的技术，而且它广泛地增强了我们解决问题的能力。如同在上一节所述及的，亚·德·莫尤勒引进生成函数为的是解决一般的线性递推问题。詹姆斯·斯特林把它稍微推广到比较更复杂的递推问题上，他说明了怎样来应用微分和积分以及算术运算[《微分方法》(Methodus Differentialis)，(伦敦1730)，命题15]。稍后几年，伦·欧拉开始以若干新的途径来使用生成函数（例如，见他关于分划的论文，Commentarii acad. sci. Pet. 13(1741)，64～93；Novi comment. acad. sci. Pet. 3(1750)，125～169)。皮埃尔·西·拉普拉斯(Pi-

erre S. Laplace)在他著名的论文《概率的解析理论》(Théorie Analytique des Probabilités)(巴黎，1812) 中又进一步发展了这一技术。

无穷和等式（1）的收敛性问题，有着一定的重要性。任何一本关于无穷级数理论的教本都会证明：

a）如果对于一个特殊的 z 值 $z=z_0$，等式（1）存在（"收敛"），则它对于所有满足 $|z|<z_0$ 的 z 值都收敛。

b）这个级数对于某个 $z\neq 0$ 收敛，当且仅当，序列 $\sqrt[n]{|a_n|}$ 有界（如果这个条件不满足，则也有可能对于一个有关的序列，例如，对于序列 $\langle a_n/n!\rangle$，得到一个收敛的级数）。

另一方面，当使用生成函数求解时，通常不必为级数的收敛性而担心，因为我们仅仅是探索对于某个问题的可能的解决途径。当通过无论什么途径发现解时，不论这个解是多么草率，都有可能独立地来论证这个解的正确性。例如，在上一小节中，使用生成函数导出等式 (14)，而一旦找到了这个等式，再用归纳法来证明它，那就是一件简单的事情了，而且再也不必提及我们曾经利用生成函数来发现这个关系式。其次，能够证明不顾级数的收敛性，而严格地论证：用生成函数所作的大多数（如果不是全部）运算是正确的。例如，见埃·坦·贝尔（E. T. Bell），Trans. Amer. Math. Soc. 25(1923), 135～154，以及伊凡·尼文（Ivan Niven），AMM 76(1969), 871～889。

现在来研究使用生成函数的主要技术。

A．加法 如果 $G_1(z)$ 是对于 a_0，a_1，…的生成函数，而 $G_2(z)$ 是对于 b_0，b_1，…的生成函数，则 $\alpha G_1(z)+\beta G_2(z)$ 是对于 $\alpha a_0+\beta b_0$，$\alpha a_1+\beta b_1$，…的生成函数：

$$\alpha\sum_{k\geqslant 0}a_kz^k+\beta\sum_{k\geqslant 0}b_kz^k=\sum_{k\geqslant 0}(\alpha a_k+\beta b_k)z^k \tag{2}$$

B．移位 如果 $G(z)$ 是对于 a_0，a_1，…的生成函数，则 $z^nG(z)$ 是对于 0，…，0，a_0，a_1，…的生成函数：

$$z^n\sum_{k\geqslant 0}a_kz^k=\sum_{k\geqslant n}a_{k-n}z^k \tag{3}$$

只要认为对于任意负的 k 值 $a_k=0$，就可以把右边的求和扩展到全部的 $k\geqslant 0$。

类似地，$(G(z)-a_0-a_1z-\cdots-a_{n-1}z^{n-1})/z^n$ 是对于 a_n，a_{n+1}，…的生成函数：

$$z^{-n}\sum_{k\geqslant n}a_kz^k=\sum_{k\geqslant 0}a_{k+n}z^k \tag{4}$$

我们把运算A和B结合在一起，来解决前一小节中的斐波那契问题。$G(z)$ 是对于 $\langle F_n\rangle$ 的生成函数，$zG(z)$ 是对于 $\langle F_{n-1}\rangle$ 的，$z^2G(z)$ 是对于 $\langle F_{n-2}\rangle$ 的，而 $(1-z-z^2)G(z)$ 是对于 $\langle F_n-F_{n-1}-F_{n-2}\rangle$。当 $n\geqslant 2$ 时，后一个序列是 0。所以 $(1-z-z^2)G(z)$ 是一个多项式。类似地，给定任何"线性递推"序列 $\langle a_n\rangle$，其中 $a_n=c_1a_{n-1}+\cdots+c_ma_{n-m}$，其生成函数是一个多项式除以 $(1-c_1z-\cdots-c_mz^m)$。

现在让我们考虑所有生成函数当中最简单的情形：如果 $G(z)$ 是对于常数序列 1，1，1，…的生成函数，则 $zG(z)$ 生成 0，1，1，…，所以 $(1-z)G(z)=1$。这就给

出了非常重要的公式

$$\frac{1}{1-z}=1+z+z^2+\cdots \tag{5}$$

C．乘法 如果 $G_1(z)$ 是对于 a_0，a_1，…的生成函数，而 $G_2(z)$ 是对于 b_0，b_1，…的生成函数，则

$$\begin{aligned}G_1(z)G_2(z)&=(a_0+a_1z+a_2z^2+\cdots)(b_0+b_1z+b_2z^2+\cdots)\\&=(a_0b_0)+(a_0b_1+a_1b_0)z+(a_0b_2+a_1b_1+a_2b_0)z^2+\cdots\end{aligned}$$

于是 $G_1(z)G_2(z)$ 是对于 s_0，s_1，…的生成函数，其中

$$s_n=\sum_{0\leqslant k\leqslant n}a_kb_{n-k} \tag{6}$$

等式（3）是这个等式的非常特殊的情况。当每个 b_n 都为 1 时，又出现了另外一个重要的特殊情况：

$$\frac{1}{1-z}G(z)=a_0+(a_0+a_1)z+(a_0+a_1+a_2)z^2+\cdots \tag{7}$$

这里，我们已经获得了对于原来的序列的和数的生成函数。

由等式（6）又可得出三个函数的乘积的规则。$G_1(z)G_2(z)G_3(z)$ 生成 $s_0,s_1,\cdots$，其中

$$s_n=\sum_{\substack{i,j,k\geqslant 0\\ i+j+k=n}}a_ib_jc_k \tag{8}$$

对于任意多个函数的乘积（当其有意义时）的一般规则是

$$\prod_{j\geqslant 0}\Big(\sum_{k\geqslant 0}a_{jk}z^k\Big)=\sum_{n\geqslant 0}z^n\Big(\sum_{\substack{k_0,k_1,\cdots\geqslant 0\\ k_0+k_1+\cdots=n}}a_{0k_0}a_{1k_1}\cdots\Big) \tag{9}$$

当关于某个序列的递推关系包含有二项式系数时，常常需要得到一个对于序列 c_0，c_1，…的生成函数，其中该序列定义为

$$c_n=\sum_k\binom{n}{k}a_kb_{n-k} \tag{10}$$

在这种情况下，利用对于序列$\langle a_n/n!\rangle$，$\langle b_n/n!\rangle$，$\langle c_n/n!\rangle$的生成函数通常是较好的。因为我们有

$$\Big(\frac{a_0}{0!}+\frac{a_1}{1!}z+\frac{a_2}{2!}z^2+\cdots\Big)\Big(\frac{b_0}{0!}+\frac{b_1}{1!}z+\frac{b_2}{2!}z^2+\cdots\Big)=\Big(\frac{c_0}{0!}+\frac{c_1}{1!}z+\frac{c_2}{2!}z^2+\cdots\Big) \tag{11}$$

其中 c_n 是由等式（10）给出的。

D．z 的变化 显然 $G(cz)$ 是对于序列 a_0，ca_1，c^2a_2，…的生成函数。具体地说，对于序列 1，c，c^2，c^3，…，其生成函数是 $1/(1-cz)$。

要抽出一个级数的交替的项，有一个熟知的技巧：

$$\left.\begin{aligned}\frac{1}{2}(G(z)+G(-z))&=a_0+a_2z^2+a_4z^4+\cdots\\ \frac{1}{2}(G(z)-G(-z))&=a_1z+a_3z^3+a_5z^5+\cdots\end{aligned}\right\} \tag{12}$$

利用复数的单位根，我们可以把这个思想加以推广，即来抽取每一个m次项：设$\omega=e^{2\pi i/m}$，则我们有

$$\sum_{k \bmod m=r} a_k z^k=\frac{1}{m}\sum_{1\leqslant j\leqslant m}\omega^{-jr}G(\omega^j z),\quad 0\leqslant r<m \tag{13}$$

例如，如果$m=3$和$r=1$，则我们有$\omega=\cos 120^\circ+i\sin 120^\circ$（一个复数的三次单位根），从而得出

$$a_1 z+a_4 z^4+a_7 z^7+\cdots=\frac{1}{3}(G(z)+\omega^{-1}G(\omega z)+\omega^{-2}G(\omega^2 z))$$

证明留给读者（习题14）。

E．微分与积分 微积分的技术给了我们进一步的运算。如果$G(z)$由等式（1）给出，则其导数为

$$G'(z)=a_1+2a_2 z+3a_3 z^2+\cdots=\sum_{k\geqslant 0}(k+1)a_{k+1}z^k \tag{14}$$

对于序列$\langle na_n\rangle$，其生成函数是$zG'(z)$。因此我们可以通过处理生成函数而把一个序列的第n项同n的一个多项式结合起来。

逆转这个过程，积分给了我们另一个有用的运算：

$$\int_0^z G(t)dt=a_0 z+\frac{1}{2}a_1 z^2+\frac{1}{3}a_2 z^3+\cdots=\sum_{k\geqslant 1}\frac{1}{k}a_{k-1}z^k \tag{15}$$

作为特殊情况，我们有等式（5）的微分和积分：

$$\frac{1}{(1-z)^2}=1+2z+3z^2+\cdots=\sum_{k\geqslant 0}(k+1)z^k \tag{16}$$

$$\ln\frac{1}{1-z}=z+\frac{1}{2}z^2+\frac{1}{3}z^3+\cdots=\sum_{k\geqslant 1}\frac{1}{k}z^k \tag{17}$$

我们把第二个公式同等式（7）结合起来，就得到对于调和数的生成函数：

$$\frac{1}{1-z}\ln\frac{1}{1-z}=z+\frac{3}{2}z^2+\frac{11}{6}z^3+\cdots=\sum_{k\geqslant 0}H_k z^k \tag{18}$$

F．已知的生成函数 只要有可能来确定一个函数的幂级数展开，就意味着我们已经得到了一个对于特殊序列的生成函数。这些特殊的函数，一经与上述的运算联系起来，就可能是十分有用的。

以下列出最重要的幂级数展开式。

i）二项式定理

$$(1+z)^r=1+rz+\frac{r(r-1)}{2}z^2+\cdots=\sum_{k\geqslant 0}\binom{r}{k}z^k \tag{19}$$

当r是一个负整数时，我们得到了一个已经在等式（5）和(16)中反映出来的特殊情况：

$$\frac{1}{(1-z)^{n+1}}=\sum_{k\geqslant 0}\binom{-n-1}{k}(-z)^k=\sum_{k\geqslant 0}\binom{n+k}{n}z^k \tag{20}$$

还有一个推广，其证明在习题1.2.6-25中给出：如果x是解方程$x^{t-1}=x^t+z$的z的

连续函数，其中 $x=1$ 当 $z=0$ 时，则

$$x^r=1+rz+\frac{r(r-2t-1)}{2}z^2+\cdots=\sum_{k\geq 0}\binom{r-kt}{k}\frac{r}{r-kt}z^k \tag{21}$$

ii) 指数级数

$$e^z=1+z+\frac{1}{2!}z^2+\cdots=\sum_{k\geq 0}\frac{1}{k!}z^k \tag{22}$$

一般地说，我们有下列的有关斯特林数的公式：

$$(e^z-1)^n=z^n+\frac{1}{n+1}\binom{n+1}{n}z^{n+1}+\cdots=n!\sum_k\binom{k}{n}z^k/k! \tag{23}$$

iii) 对数级数（参看 (17)）

$$\ln(1+z)=z-\frac{1}{2}z^2+\frac{1}{3}z^3-\cdots=\sum_{k\geq 1}\frac{(-1)^{k+1}}{k}z^k \tag{24}$$

$$\left(\frac{1}{(1-z)^{m+1}}\right)\ln\left(\frac{1}{1-z}\right)=\sum_{k>0}\left(\prod_{m+k}-\prod_m\right)\binom{m+k}{m}z^k,\quad m>0 \tag{25}$$

利用斯特林数（参照等式 (23)），我们就有更一般的等式：

$$\left(\ln\left(\frac{1}{1-z}\right)\right)^n=z^n+\frac{1}{n+1}\begin{bmatrix}n+1\\n\end{bmatrix}z^{n+1}+\cdots=n!\sum_k\begin{bmatrix}k\\n\end{bmatrix}z^k/k! \tag{26}$$

关于进一步的推广，见 D. A. 佐夫 (D. A. Zove)，Inf. Proc. Letters 5(1976)。

iv) 其它的

$$z(z+1)\cdots(z+n-1)=\sum_k\begin{bmatrix}n\\k\end{bmatrix}z^k \tag{27}$$

$$\frac{z^n}{(1-z)(1-2z)\cdots(1-nz)}=\sum_k\begin{Bmatrix}k\\n\end{Bmatrix}z^k \tag{28}$$

$$\frac{z}{e^z-1}=1-\frac{1}{2}z+\frac{1}{12}z^2+\cdots=\sum_{k\geq 0}\frac{B_kz^k}{k!} \tag{29}$$

出现在上边最后一个公式的系数 B_k 是贝努利数。在 1.2.11.2 小节里，以及在附录 B 的贝努利数表中，还将对它们作进一步的考察。

类似于等式 (21)，有另一个恒等式如下（见习题 2.3.4.4-29）：

$$x^r=1+rz+\frac{r(r+2t)}{2}z^2+\cdots=\sum_{k\geq 0}\frac{r(r+kt)^{k-1}}{k!}z^k \tag{30}$$

如果 x 是解方程 $x=e^{tx^t}$ 的 z 的连续函数时，其中 $x=1$ 当 $z=0$ 时。

在结束这一小节之前，我们还要回过头来解决一个问题，这个问题我们在 1.2.3 小节中只是部分地解决了。我们已经看到（等式 1.2.3-13 以及习题 1.2.3-29）

$$\sum_{1\leq i\leq j\leq n}x_ix_j=\frac{1}{2}\left(\left(\sum_{1\leq k\leq n}x_k\right)^2+\left(\sum_{1\leq k\leq n}x_k^2\right)\right)$$

$$\sum_{1\leqslant i\leqslant j\leqslant k\leqslant n} x_i x_k x_j = \frac{1}{6}\left(\left(\sum_{1\leqslant k\leqslant n} x_k\right)^3 + 3\left(\sum_{1\leqslant k\leqslant n} x_k\right)\left(\sum_{1\leqslant k\leqslant n} x_k^2\right) + 2\left(\sum_{1\leqslant k\leqslant n} x_k^3\right)\right)$$

一般地说，假设我们有 n 个数 x_1，x_2，…，x_n，而我们欲求和数

$$h_m = \sum_{1\leqslant j_1\leqslant\cdots\leqslant j_m\leqslant n} x_{j_1}\cdots x_{j_m}$$

如果可能的话，要求这个和数借助 S_1，S_2，…，S_m 来表达，其中

$$S_j = \sum_{1\leqslant k\leqslant n} x_k^j \tag{31}$$

是 j 次幂的和数。利用这个更为紧致的记号，上边的公式即变成 $h_2 = \frac{1}{2}S_1^2 + \frac{1}{2}S_2$；$h_3 = \frac{1}{6}S_1^3 + \frac{1}{2}S_1S_2 + \frac{1}{3}S_3$。

通过建立生成函数

$$G(z) = 1 + h_1 z + h_2 z^2 + \cdots = \sum_{k\geqslant 0} h_k z^k \tag{32}$$

我们即可着手处理这个问题。按照级数相乘的规则，我们求得

$$G(z) = (1 + x_1 z + x_1^2 z^2 + \cdots)\cdots(1 + x_n z + x_n^2 z^2 + \cdots) = \frac{1}{(1-x_1 z)\cdots(1-x_n z)} \tag{33}$$

所以 $G(z)$ 是一个多项式的倒数。通常，对一个乘积取对数是有利的。这样，我们求得

$$\ln G(z) = \ln\left(\frac{1}{1-x_1 z}\right) + \cdots + \ln\left(\frac{1}{1-x_n z}\right) = \left(\sum_{k\geqslant 1}\frac{x_1^k z^k}{k}\right) + \cdots + \left(\sum_{k\geqslant 1}\frac{x_n^k z^k}{k}\right) = \sum_{k\geqslant 1}\frac{S_k z^k}{k}$$

现在我们已经用 S 表达了 $\ln G(z)$。所以，为了得到我们问题的答案，当前必须要做的，就是再次来计算 $G(z)$ 的幂级数展开：

$$\begin{aligned} G(z) = e^{\ln G(z)} &= \exp\left(\sum_{k\geqslant 1}\frac{S_k z^k}{k}\right) = \prod_{k\geqslant 1} e^{S_k z^k/k} \\ &= \left(1 + S_1 z + \frac{S_1^2 z^2}{2!} + \cdots\right)\left(1 + \frac{S_2 z^2}{2} + \frac{S_2^2 z^2}{2^2\cdot 2!} + \cdots\right)\cdots \\ &= \sum_{m\geqslant 0}\left(\sum_{\substack{k_1,k_2,\cdots,k_m\geqslant 0 \\ k_1+2k_2+\cdots+mk_m=m}} \frac{S_1^{k_1}}{1^{k_1}k_1!}\,\frac{S_2^{k_2}}{2^{k_2}k_2!}\cdots\frac{S_m^{k_m}}{m^{k_m}k_m!}\right) z^m \end{aligned} \tag{34}$$

圆括号中的数量就是 h_m。仔细地来考察一下，这个似乎令人兴叹的和数，实际上并不复杂。对于一个具体的 m 值，其项数为 $p(m)$，即 m 之分划的数目（参看 1.2.1 小节）。例如，12 的一个分划是

$$12 = 5+2+2+2+1$$

这对应于方程 $k_1 + 2k_2 + \cdots + 12k_{12} = 12$ 的一个解，其中 k_j 是 j 在分划中的个数。在我们的例子中，$k_1 = 1$，$k_2 = 3$，$k_5 = 1$，而其余的 k 都是 0。这样我们就得到了项

$$\frac{S_1}{1^1 1!}\,\frac{S_2^3}{2^3 3!}\,\frac{S_5}{5^1 1!} = \frac{1}{240}S_1 S_2^3 S_5$$

这是 h_{12} 的表达式的一部分。

等式（34）中的系数的表，以及伐敌奔公式（习题 1.2.5-21）类似的系数的表，见米·阿布拉莫维兹和艾·安·斯特冈编的《数学函数手册》（美国政府出版社，1964），表24.2。

乔·波利亚已经给出了关于生成函数之应用的有趣介绍《关于用图画记载的方法》（On picture writing），AMM 63(1956)，689～697。

习题

1.〔*M*12〕 对于序列 2，5，13，35，…=〈2^n+3^n〉的生成函数是什么？

▶2.〔*M*13〕 证明等式（11）。

3.〔*HM*21〕 试求对于〈H_n〉的生成函数（等式 18）的微商，并把它同对于〈$\Sigma_{1\leq k\leq n}H_k$〉的生成函数加以比较。你能推导出什么关系来？

4.〔*M*01〕 说明等式（19）为什么是等式（21）的一个特殊情况？

5.〔*M*20〕 对 n 用归纳法，证明等式（23）。

▶6.〔*HM*15〕 求出对于

$$\sum_{1\leq k<n} k\left(\begin{matrix}1\\ n-k\end{matrix}\right)$$

的生成函数，对它取微商并借助调和数来表达其系数。

7.〔*M*20〕 验证导出等式（34）的所有步骤。

8.〔*M*23〕 求出对于 n 的分划数 $p(n)$ 的生成函数。

9.〔*M*11〕 用等式(31)和（32）的记号，如何借助于 S_1，S_2，S_3 和 S_4 来表示 h_4？

▶10.〔*M*25〕 初等对称函数由公式

$$a_m=\sum_{1\leq j_1<\cdots<j_m\leq n} x_{j_1}\cdots x_{j_m}$$

定义（这和等式（32）中的 h_m 是一样的，但这不允许取相同的下标）。求出对于 a_m 的生成函数，并借助于等式（31）中的 S_j 来表达 a_m。写出对于 a_1，a_2，a_3 和 a_4 的公式。

11.〔*HM*30〕 建立对于序列〈$n!$〉的生成函数，并研究这个函数的性质。

▶12.〔*M*20〕 假设我们有对于 m，$n=0$，1，…的双重下标的序列 a_{mn}，说明怎样通过一个单一的两个变数的生成函数，来表示这个双重序列，并确定对于序列 $a_{mn}=\binom{n}{m}$ 的生成函数。

13.〔*HM*22〕 一个函数 $f(x)$ 的“拉普拉斯变换”是函数 $\mathrm{L}f(s)=\int_0^\infty e^{-st}f(t)\,dt$。给定无穷序列 a_0，a_1，a_2，…，它具有收敛的生成函数。设 $f(x)$ 是阶梯函数 $\Sigma_{0\leq k\leq x}a_k$。试用对于这个序列的生成函数 G 来表达 $f(x)$ 的拉普拉斯变换。

14.〔*HM*21〕 证明等式（13）。

15.〔*M*28〕 通过考虑 $H(w)=\Sigma_{n\geq 0}G_n(z)w^n$，找出生成函数

$$G_n(z)=\sum_{0\leq k\leq n}\binom{n-k}{k}z^k$$

的“封闭形式”。

16. 〔$M22$〕 给出对于生成函数$G_{nr}(z)=\sum_k a_{nkr}z^k$的一个简单公式，其中$a_{nkr}$是在$n$个对象中选择出$k$个的方法的种数，但附加上每一对象至多可取$r$次的条件（如果$r=1$，则我们有$\binom{n}{k}$种方法；而如果$r\geqslant k$，则我们就得到有重复组合的数目（参看习题1.2.6-60））。

17. 〔$M25$〕 如果把函数$1/(1-z)^w$展开成关于z和w的双重幂级数，则其系数是什么？

▶18. 〔$M25$〕 给定正整数n和r，求出对于以下和数之值的简单公式：

$$\text{(a)}\ \sum_{1\leqslant k_1<k_2<\cdots<k_r\leqslant n} k_1k_2\cdots k_r \qquad \text{(b)}\ \sum_{1\leqslant k_1\leqslant k_2\leqslant\cdots\leqslant k_r\leqslant n} k_1k_2\cdots k_r$$

（例如，当$n=3$，$r=2$时，和数分别为$1\cdot2+1\cdot3+2\cdot3$和$1\cdot1+1\cdot2+1\cdot3+2\cdot2+2\cdot3+3\cdot3$）。

19. 〔$HM32$〕 （卡·弗·高斯）如所熟知，下列无穷级数的和数是

$$1-\frac{1}{2}+\frac{1}{3}-\frac{1}{4}+\cdots=\ln 2 \qquad 1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+\cdots=\frac{\pi}{4}$$

$$1-\frac{1}{4}+\frac{1}{7}-\frac{1}{10}+\cdots=\frac{\pi}{3\sqrt{3}}+\frac{1}{3}\ln 2$$

这些级数可以分别地写作

$$\frac{1}{2}\sum_{n\geqslant0}\left(\frac{1}{n+\frac{1}{2}}-\frac{1}{n+1}\right);\quad \frac{1}{4}\sum_{n\geqslant0}\left(\frac{1}{n+\frac{1}{4}}-\frac{1}{n+1}\right)-\frac{1}{4}\sum_{n\geqslant0}\left(\frac{1}{n+\frac{3}{4}}-\frac{1}{n+1}\right)$$

和

$$\frac{1}{6}\sum_{n\geqslant0}\left(\frac{1}{n+\frac{1}{6}}-\frac{1}{n+1}\right)-\frac{1}{6}\sum_{n\geqslant0}\left(\frac{1}{n+\frac{2}{3}}-\frac{1}{n+1}\right)$$

试证明：一般说来，当p和q是整数且$0<p<q$时，级数

$$\sum_{n\geqslant0}\left(\frac{1}{n+p/q}-\frac{1}{n+1}\right)$$

有值

$$\frac{\pi}{2}\cot\frac{p}{q}\pi+\ln 2q-2\sum_{0<k<q/2}\cos\frac{2pk}{q}\pi\cdot\ln\sin\frac{k}{q}\pi$$

〔提示：根据阿贝尔极限定理，这个和数是

$$\lim_{x\to1-}\sum_{n\geqslant0}\left(\frac{1}{n+p/q}-\frac{1}{n+1}\right)x^{p+nq}$$

利用等式（13）把这个幂级数表达成这样的形式，即使得这个极限值能够容易地计算的形式〕。

1.2.10 一个算法的分析

现在让我们应用前几小节的某些技术，来研究一个典型的算法。

算法M（求极大值） 给定n个元素$X[1]$，$X[2]$，…，$X[n]$，我们要求出m和j，使得$m=X[j]=\max_{1\leqslant k\leqslant n}X[k]$，而且使其中之$j$尽可能地大。

M1. 〔初始化〕置 $j \leftarrow n$，$k \leftarrow n-1$，$m \leftarrow X[n]$。

M2. 〔全都试过了?〕若 $k=0$，则算法终止。

M3. 〔比较〕若 $X[k] \leqslant m$，则转M5。

M4. 〔改变m〕置 $j \leftarrow k$，$m \leftarrow X[k]$（现在m是当前的极大值）。

M5. 〔k减1〕k减1，返回到M2。▌

这个颇为明显的算法可能显得如此平凡，以至不屑详细地分析它，但它实际上却很好地揭示了我们将要研究的更复杂的算法的风格。在计算机程序设计中，算法分析是十分重要的。因为，通常对于一个具体的应用有若干个算法可资利用，我们很需要知道哪一个算法是最好的。

算法M所要求的存储数量是固定的，所以仅仅分析执行它所需要的时间。为此，计算执行每一步的次数（参照图9）：

步骤号	次数
M1	1
M2	n
M3	$n-1$
M4	A
M5	$n-1$

知道了每步执行的次数，就给了我们为确定在一个具体的计算机上的运行时间所必要的信息。

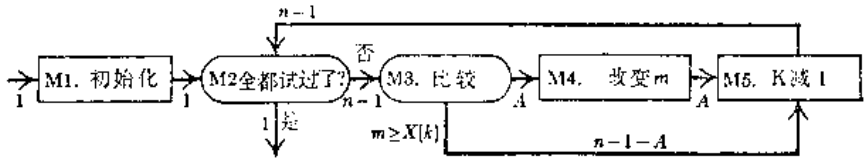


图9 算法M。箭头上所标的数量指出走过该条通路的次数。注意：必须满足"克希霍夫第一定律"，即是进入每一节点的流量必须等于从中出来的流量。

在上列的表中，除了数量A外，每一个数量我们都是知道的，而数量A乃是必须改变当前的极大值之次数。为了完成整个分析，必须研究这个有趣的数量A。

这种分析通常包括以下几个组成部分：求A的极小值（对于乐观的人们），A的极大值（对于悲观的人们）和A的平均值（对于抱希望的人们）以及A的标准离差（这是一个定量指标，以指出预期的值可能接近平均值的程度）。

A的极小值为0；如果$X[n]=\max_{1\leqslant k\leqslant n} X[k]$时，即出现这种情况。极大值是$n-1$；在$X[1]>X[2]>\cdots>X[n]$的情况下，即出现这种情况。

因此，平均值是在0与$n-1$之间。它等于$\frac{1}{2}n$吗？等于$\frac{1}{3}n$？为了回答这个问题，就需要定义所谓的"平均"意味着什么；而且为了适当地定义"平均"，还必须对输入数据$X[1]$，$X[2]$，…，$X[n]$的预期的特征作某些假定。假定$X[k]$的值是互不相同的，而且这些值的$n!$种排列的每一种都是同等可能的（这在大多数情况下是一个合理的假定，但是这个分析也可以在其它假定之下进行，如这一小节末尾的习题所示那样）。

算法M的性能并不依赖于$X[k]$的精确的值的大小，而仅仅涉及其相对的次序。例如，

假设 $n=3$。假定以下六种情况的每一种都是同等可能的：

情况	A的值	情况	A的值
$X[1]<X[2]<X[3]$	0	$X[2]<X[3]<X[1]$	1
$X[1]<X[3]<X[2]$	1	$X[3]<X[1]<X[2]$	1
$X[2]<X[1]<X[3]$	0	$X[3]<X[2]<X[1]$	2

当 $n=3$ 时，A的平均值为 $(0+1+0+1+1+2)/6=5/6$。

显然，可以取$X[1]$，$X[2]$，…，$X[n]$为在某种次序下的 1，2，…，n；在我们的假定下，$n!$种排列的每一种都是同等可能的。A有值 k 的概率将是

$$p_{nk}=(n\text{ 个对象的排列中满足 }A=k\text{ 者之排列数})/n! \qquad (1)$$

例如，由上列表，$p_{30}=\frac{1}{3}$，$p_{31}=\frac{1}{2}$，$p_{32}=\frac{1}{6}$。

象通常那样，平均值（"中值"）定义为

$$A_n=\sum_k k p_{nk} \qquad (2)$$

方差 V_n 定义为 $(A-A_n)^2$ 的平均值；因此有

$$V_n=\sum_k (k-A_n)^2 p_{nk}=\sum_k k^2 p_{nk}-2A_n\sum_k k p_{nk}+A_n^2\sum_k p_{nk}$$

$$=\sum_k k^2 p_{nk}-2A_nA_n+A_n^2=\sum_k k^2 p_{nk}-A_n^2 \qquad (3)$$

最后，标准离差 σ_n 定义为$\sqrt{V_n}$。

也许，能够最好地了解 σ_n 之意义的办法，是通过说明：对于所有的 $r\geqslant 1$，A不落入它的平均值的 $r\sigma_n$ 之内的概率小于 $1/r^2$。例如，$|A-A_n|>2\sigma_n$ 之概率$<\frac{1}{4}$〔证明：以 p 来表示所述的概率，则 $(A-A_n)^2$的平均值大于 $p\cdot(r\sigma_n)^2+(1-p)\cdot 0$，即是 $V_n>pr^2V_n$，除非 $p=0$〕。这叫做契贝雪夫（Чебышев）不等式。

通过确定概率 p_{nk}，即能确定A的特性。但这不难归纳地求得：按照等式(1)，需要来计算关于 n 个元素的使有 $A=k$ 的排列的数目。

考虑关于{1，2，…，n}的排列 $x_1x_2\cdots x_n$（参看 1.2.5 小节）。如果 $x_1=n$，则A的值等于它对 $x_2\cdots x_n$ 所得到的值加 1；如果 $x_1\neq n$，则A的值恰巧等于它对 $x_2\cdots x_n$所得到的值。因此，得出

$$p_{nk}=\frac{1}{n}p_{(n-1)(k-1)}+\frac{n-1}{n}p_{(n-1)k} \qquad (4)$$

如果提供初始条件

$$p_{1k}=\delta_{0k} \qquad \text{以及 } p_{nk}=0 \qquad \text{当 } k<0 \text{ 时} \qquad (5)$$

则这个等式将确定 p_{nk}。

现在，可以通过生成函数得到关于数量 p_{nk} 的信息。设

$$G_n(z)=p_{n0}+p_{n1}z+\cdots=\sum_k p_{nk}z^k \qquad (6)$$

我们知道 $A\leqslant n-1$，所以对于充分大的 k 值，$p_{nk}=0$。因此 $G_n(z)$ 实际上是一个多项

式——把它写成是一个无穷和，只是为图方便起见。

由等式（5）有 $G_1(z)=1$，而且由等式（4）有

$$G_n(z)=\frac{z}{n}G_{n-1}(z)+\frac{n-1}{n}G_{n-1}(z)=\frac{z+n-1}{n}G_{n-1}(z) \tag{7}$$

（读者应当仔细地研究等式（4）和等式（7）之间的关系）。现在可看到

$$G_n(z)=\frac{z+n-1}{n}\cdot\frac{z+n-2}{n-1}G_{n-2}(z)=\cdots$$

$$=\frac{1}{n!}(z+n-1)(z+n-2)\cdots(z+1)=\frac{1}{n!}\left(\begin{matrix}z+n\\ n\end{matrix}\right) \tag{8}$$

所以 $G_n(z)$ 实质上是一个二项式系数！

在前一小节中已经遇到过这个函数（等式 1.2.9-27），在那里有

$$G_n(z)=\frac{1}{n!}\sum_k\left[\begin{matrix}n\\ k\end{matrix}\right]z^{k-1}$$

因此，借助斯特林数可表达 p_{nk}：

$$p_{nk}=\left[\begin{matrix}n\\ k+1\end{matrix}\right]/n! \tag{9}$$

图 10 示明了当 $n=12$ 时 p_{nk} 的近似数值。

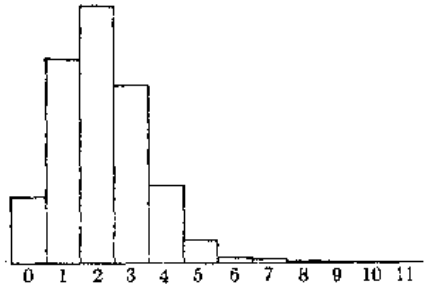


图10 当 $n=12$ 时，对于步骤 M4 的概率分布。平均值是 58301/22720≈2.11

现在所要做的一切，就是把 p_{nk} 的这个值插入到等式（2）和（3）中去，从而得到所求的平均值。但是，这件事说起来容易，做起来就要费事些。事实上，通常是不大可能明显地确定 p_{nk} 的。在大多数问题中，将求得生成函数 $G_n(z)$，而对于实际的概率却没有任何特殊的了解。重要的事实是：由生成函数本身，就能容易地确定平均值和方差。

为了了解这一点，现在假设有一个生成函数，其系数表示概率：

$$G(z)=p_0+p_1z+p_2z^2+\cdots$$

这里 p_k 是某事件具有值 k 的概率。再来计算等式

$$\text{mean}(G)=\sum_k kp_k,\quad \text{var}(G)=\sum_k k^2p_k-(\text{mean}(G))^2 \tag{10}$$

利用微分，不难发现怎样来做此事。注意

$$G(1)=1 \tag{11}$$

因为，$G(1)=p_0+p_1+p_2+\cdots$ 是所有可能的概率之和。类似地，由于 $G'(z)=\sum_k kp_kz^{k-1}$，有

$$\text{mean}(G)=\sum_k kp_k=G'(1) \tag{12}$$

最后，再次应用微分而且得到（见习题 2）

$$\text{var}(G)=G''(1)+G'(1)-G'(1)^2 \tag{13}$$

等式（12）和（13）借助于生成函数给出了所求的平均值和方差的表达式。

在当前的情况下，我们希望来计算 $G_n'(1)=A_n$。由等式（7），有

$$G_n'(z)=\frac{1}{n}G_{n-1}(z)+\frac{z+n-1}{n}G_{n-1}'(z)$$

$$G_n'(1)=\frac{1}{n}+G_{n-1}'(1)$$

由初始条件 $G_1'(1)=0$，因此得到

$$A_n=G_n'(1)=H_n-1 \tag{14}$$

这就是所求的执行步骤M4的平均次数，当 n 很大时，它近似于 $\ln n$〔注意：第 r 次矩量是 $(1-z)^{-1}\Sigma_k \left\{ {r \atop k} \right\} \ln(1/(1-z)^k)$ 中 z^n 的系数，且它的近似值是 $(\ln n)^r$；见CACM9(1966), 342。头一个研究 A 的分布的是弗·戈·福斯特（F. G. Foster）和艾·斯图尔特(A. Stuart), J. Roy. Stat. Soc. B-16(1954), 1～22〕。

现在可以类似地进行方差 V_n 的计算。在这之前，让我们叙述一个重要的简化：

定理 A 设 G, H 是满足 $G(1)=H(1)=1$ 的两个生成函数。如果以等式 (12), (13) 定义数量 $\mathrm{mean}(G)$，$\mathrm{var}(G)$，则有

$$\mathrm{mean}(GH)=\mathrm{mean}(G)+\mathrm{mean}(H) \qquad \mathrm{var}(GH)=\mathrm{var}(G)+\mathrm{var}(H) \tag{15}$$

我们将在后边证明这个定理。这定理告诉我们：生成函数的一个乘积的平均值和方差可以简化为一个和数。

设 $Q_n(z)=(z+n-1)/n$，有 $Q_n'(1)=1/n$，$Q_n''(1)=0$；因此

$$\mathrm{mean}(Q_n)=\frac{1}{n} \qquad \mathrm{var}(Q_n)=\frac{1}{n}-\frac{1}{n^2}$$

最后，因为 $G_n(z)=\Pi_{2\leqslant k\leqslant n}Q_k(z)$，由此得出

$$\mathrm{mean}(G_n)=\sum_{2\leqslant k\leqslant n}\mathrm{mean}(Q_k)=\sum_{2\leqslant k\leqslant n}\frac{1}{k}=H_n-1$$

$$\mathrm{var}(G_n)=\sum_{2\leqslant k\leqslant n}\mathrm{var}(Q_k)=\sum_{1\leqslant k\leqslant n}\left(\frac{1}{k}-\frac{1}{k^2}\right)=H_n-H_n^{(2)}$$

综合起来，已经得出所求的关于数量 A 的统计：

$$A=(\min 0,\ \mathrm{ave}\, H_n-1,\ \max n-1,\ \mathrm{dev}\sqrt{H_n-H_n^{(2)}}) \tag{16}$$

〔=(极小值，平均值，极大值，标准离差)。〕在本书中，将始终使用等式 (16) 中所用的记号，来描述其它概率数量的这些统计特征。

我们已经完成了算法M的分析。在这个分析中出现的新特征，就是引进了概率论。对于本书中大多数应用来说，并不需要多少概率论——上面所给出的简单的计数技术和平均值、方差以及标准离差的定义就已经足够了。

让我们考虑某些简单的概率问题，以便获得一点使用这些方法的实践。在所有的概率问题中，人们首先想到的一个问题就是硬币投掷问题。假设我们用手指弹掷一个硬币 n 次，而且在每次投掷中“图面”朝上的概率为 p。今问“图面”将出现的平均数是多少？标准离差等于什么？

我们认为硬币是有偏向的，即是，不假定 $p=\frac{1}{2}$。这就使问题更有趣些，而且，其实是，每个实际的硬币都是有偏向的（否则就无从区别这一面还是那一面了）！

象前边一样来进行，假设 p_{nk} 是 k 次出现图面的概率，并设 $G_n(z)$ 是对应的生成函

数。显然有

$$p_{nk}=p\cdot p_{(n-1)(k-1)}+q\cdot p_{(n-1)k} \tag{17}$$

这里，$q=1-p$是在每次投掷时“字面”朝上的概率。和以前一样，由等式（17）论证$G_n(z)=(q+pz)G_{n-1}(z)$，而且，由显然的初始条件$G_1(z)=q-pz$，有

$$G_n(z)=(q+pz)^n \tag{18}$$

因此，

$$\mathrm{mean}(G_n)=n\,\mathrm{mean}(G_1)=pn$$
$$\mathrm{var}(G_n)=n\,\mathrm{var}(G_1)=(p-p^2)\,n=pqn$$

对于“图面”的数目，因此有

$$(\min 0,\ \mathrm{ave}\ pn,\ \max n,\ \mathrm{dev}\sqrt{pqn}) \tag{19}$$

图11说明了当$p=\frac{3}{5}$，$n=12$时p_{nk}的值。当标准离差与$\sqrt{n}$成正比时而且当极大值与极小值之差与n成正比时，可以认为这是关于平均值的“稳定”状态。

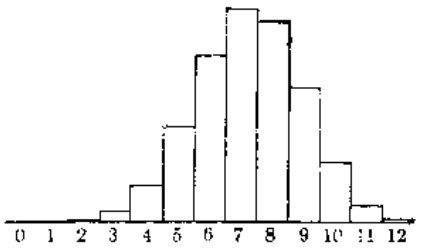


图11 硬币投掷的概率分布。每次投掷时成功机会等于$\frac{3}{5}$的12次投掷

这里让我们来考虑一个更简单的问题。假设在某个进程中，有相等的概率来得到值$1,2,\cdots,n$。对于这种情况的生成函数是

$$G(z)=\frac{1}{n}z+\frac{1}{n}z^2+\cdots+\frac{1}{n}z^n=\frac{1}{n}\,\frac{z^{n+1}-z}{z-1} \tag{20}$$

在经过稍微烦琐的计算之后，求得

$$G'(z)=\frac{nz^{n+1}-(n+1)z^n+1}{n(z-1)^2}$$
$$G''(z)=\frac{n(n-1)z^{n+1}-2(n+1)(n-1)z^n+n(n+1)z^{n-1}-2}{n(z-1)^3} \tag{21}$$

现在，为了计算平均值和方差，需要知道$G'(1)$和$G''(1)$。但是当我们替入$z=1$时，就已经把这些等式表达成$0/0$的形式。这就使得有必要来求出，当z趋近于1时的极限，而这是一个不平凡的任务（参照习题6）。这里所遇到的是这样一种情况，即直接地从概率来进行计算，而不是从生成函数来推导平均值和方差，反而要容易得多。在这种情况下的统计是

$$\left(\min 1,\ \mathrm{ave}\ \frac{n+1}{2},\ \max n,\ \mathrm{dev}\sqrt{\frac{(n+1)(n-1)}{12}}\right) \tag{22}$$

在这种情况下，近似地为$0.289n$的离差给我们提供了一个可感到的不稳定状态的例子。

在结束这一小节之前，证明定理A并把我们的概念同经典的概率论联系起来。当$G(z)=p_0+p_1z+p_2z^2+\cdots$表示对于某个数量$X$的一个概率分布时——即是，如果$p_k$是$X=k$的概率，而且$X$仅取非负的整数值——有$p_k\geqslant 0$和$G(1)=1$。数量$G(e^{it})=p_0+p_1e^{it}+p_2e^{2it}+\cdots$习惯上叫做这个分布的特征函数。由两个这样的生成函数的乘积给出的分布称为这两个分布的*摺积*，而且它表示属于这些分布的两个独立随机变量的和。

平均值和方差恰恰就是蒂利（Thiele）于1903年引进的两个所谓*半不变量*或*累积量*。半不变量k_1，k_2，k_3，…是由规则

$$\frac{k_1 t}{1!}+\frac{k_2 t^2}{2!}+\frac{k_3 t^3}{3!}+\cdots=\ln G(e^t) \tag{23}$$

定义的。我们有

$$k_n=\frac{d^n}{dt^n}\ln G(e^t)\Big|_{t=0}$$

特别地，

$$k_1=\frac{e^t G'(e^t)}{G(e^t)}\Big|_{t=0}=G'(1)$$

而且

$$k_2=\frac{e^{2t}G''(e^t)}{G(e^t)}+\frac{e^t G'(e^t)}{G(e^t)}-\frac{e^{2t}G'(e^t)^2}{G(e^t)^2}\Big|_{t=0}=G''(1)+G'(1)-G'(1)^2$$

由于半不变量是借助于生成函数的对数而定义的，因而定理A是显然的，而且，事实上，它可以进一步推广以应用到所有的半不变量。

正态分布是这样一个分布，除了平均值和方差之外，所有的半不变量都为0。在一个正态分布中，随机值与其平均值之差以占总数的

$$\frac{1}{\sqrt{2\pi}}\int_{-1}^{+1}e^{-t^2/2}dt=68.268949213709\%$$

小于标准离差。这个差以占总数的95.449973610364%小于标准离差的两倍，又以占总数的99.730020393674%小于标准离差的三倍。由等式（8）和等式（18）所确定的两个分布，当n充分大时，都近似于正态分布（见习题13和14）。

习题

1.〔10〕 由等式（4）和（5），并考虑算法M，来确定p_{n0}的值，并解释这一结果。

2.〔*HM*16〕 由等式（10）推导等式（13）。

3.〔*M*15〕 如果我们利用算法M来求1000个随机地编序的、不同的数之极大值，则执行步骤M4的次数之极小值，极大值，平均值以及标准离差等于什么？（答案用十进制的近似值给出。）

4.〔*M*10〕 对于硬币投掷实验，即等式（17）中的p_{nk}的值，给出一个明显的封闭的公式。

5.〔*M*13〕 图11所示之分布的平均值和标准离差是什么？

6.〔*HM*23〕 由等式（21），利用洛必达（L'Hospital）法则求出$G'(1)$和$G''(1)$。

▶7.〔*M*27〕 在我们对算法M的分析中，我们假定$X[k]$是互不相同的。换一下，如我们仅仅作一个较弱的假定，即$X[1]$，$X[2]$，…，$X[n]$确切地含有m个互不相同的值。除了附加这个条件之外，这些值在其它方面是随机的。在这种情况下A的概率分布是什么？

▶8.〔*M*20〕 假设每个$X[k]$是从一个具有M个不同元素的集合中随机地取出的，所以对于$X[1]$，$X[2]$，…，$X[n]$的M^n种可能的选择的每一种，都看成是同等可能的。试问所有$X[k]$都不同的概率是什么？

9.〔*M*25〕 推广上题的结果，求出在那些X当中恰恰出现m个不同的值的概率的公式。借助于斯特林数来表达你的公式。

10. 〔$M20$〕 把上边三题的结果结合起来，在每个X都是由M个对象的集合中随机地选取的假定下，给出对于$A=k$的概率的公式。

11. 〔$HM20$〕 假定$G(z)=p_0+p_1z+p_2z^2+\cdots$表示一个概率分布，设$M_n=\sum_k k^n p_k$($M_n$叫做"第$n$次矩量")。证明$G(e^t)=1+M_1t+M_2t^2/2!+\cdots$。然后利用伐故奔公式（习题1.2.5-21），证明

$$k_n=\sum_{\substack{k_1,\cdots,\ k_n\geq 0\\ k_1+2k_2+\cdots=n}}\frac{n!(k_1+k_2+\cdots+k_n-1)!(-1)^{k_1+\cdots+k_n-1}}{k_1!(1!)^{k_1}\cdots k_n!(n!)^{k_n}}M_1^{k_1}\cdots M_n^{k_n}$$

〔具体地说，$k_1=M_1$，$k_2=-M_1^2+M_2$（如同我们已经知道的），$k_3=2M_1^3-3M_1M_2+M_3$，$k_4=-6M_1^4-3M_2^2+12M_1^2M_2-4M_1M_3+M_4$〕。

▶12. 〔$M15$〕 如果我们把$G(z)$变成$G_1(z)=z^nG(z)$，则一个分布的半不变量将发生什么情况？

13. 〔$HM38$〕 一个具有平均值μ_n和离差σ_n的特征函数的序列$G_n(z)$说是趋近于一个正态分布，如果对于t的所有虚数的值，即是，每当$t=ui$时，其中u为实数，都有

$$\lim_{n\to\infty}e^{-t\mu_n/\sigma_n}G_n(e^{t/\sigma_n})=e^{t^2/2}$$

试利用由等式（8）给出的那样的$G_n(z)$，证明$G_n(z)$趋近于一个正态分布〔这是一个贡查罗夫（Гончаров）定理。苏联科学院消息，数学部分（Изв. Akad. Hayk СССР сер Мате) 8 (1944)〕。

注意 可以证明，这里所定义的"趋近于正态分布"等价于

$$\lim_{n\to\infty}\text{概率}\left(\frac{X_n-\mu_n}{\sigma_n}\leqslant x\right)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{x}e^{-t^2/2}dt$$

其中X_n为一个随机量，其概率由$G_n(z)$确定。这是重要的保·利维（P. Levy）"连续性定理"的一个特殊情况，而连续性定理乃是数学概率论中的一个基本结果。这个结果的证明尽管不是极其困难的，但它将使我们颇感茫然〔例如，参看鲍·弗·格涅坚科（Б.В. Гнеденко）和安·尼·柯尔莫哥罗夫（А.Н. Колмогоров）著的《关于独立随机变量之和的极限分布》(Limit Distributions for Sums of Independent Random Variable)，钟开来译成英文（马萨诸塞州雷丁：爱迪生-韦斯利，1954），王寿仁译成中文，科学出版社，1955〕。

14. 〔$HM30$〕 （亚·德·莫尤勒）利用上题的约定，证明由等式(18)给出的二项式分布$G_n(z)$趋近于正态分布。

▶15. 〔$M21$〕 设z是一个正数。如果A是算法M的分析中出现的量，则当取遍阶n的所有排列时，数量z^A的平均值等于什么？

16. 〔$HM23$〕 当某个数量有值k的概率是$e^{-\mu}(\mu^k/k!)$时，就说它有"平均值为μ的泊松分布"。

a）对于这类概率的生成函数是什么？

b）半不变量的值是什么？

c）证明当$n\to\infty$，有平均值np的泊松分布趋近于在习题13意义下的正态分布。

▶17. 〔$M27$〕 设$f(z)$和$g(z)$是表示概率分布的生成函数。

a）证明 $h(z)=g(f(z))$也是一个表示概率分布的生成函数。

b）借助于 $f(z)$和 $g(z)$来解释 $h(z)$的意义（由 $h(z)$的系数所表示的概率之意义是什么？）

c）借助于 f，g 的平均值和方差，给出 h 的平均值和方差的公式。

18.〔M28〕假设在算法M中，由 $X[1]$，$X[2]$，…，$X[n]$所取的不同的值，恰好包括以随机次序排列的 k_1 个 1，k_2 个 2，…，k_n 个 n（这里

$$k_1+k_2+\cdots+k_n=n$$

注意，正文中假定的是 $k_1=k_2=\cdots=k_n=1$）。证明在这种推广的情形下，生成函数等式(8)变成为

$$\left(\frac{k_{n-1}z+k_n}{k_{n-1}+k_n}\right)\left(\frac{k_{n-2}z+k_{n-1}+k_n}{k_{n-2}+k_{n-1}+k_n}\right)\cdots\left(\frac{k_1z+k_2+\cdots+k_n}{k_1+k_2+\cdots+k_n}\right)$$

利用约定 $0/0=1$。

*1.2.11 渐近表示

为把一个数同另一个数加以比较，我们常常需要求一个数量的近似值，而不是精确值。例如，关于 $n!$ 的斯特林近似表达式就是这种类型的一个非常有用的表示；还有，我们也要利用事实 $H_n\approx\ln n+\gamma$。

这样的“渐近公式”的推导一般都涉及到更高等的数学。而在下面的一些小节里，为了求得所需要的结果，我们只不过是利用了初等微积分罢了。

***1.2.11.1 O记号** 为处理近似值，可采用一个非常方便的记号，这就是由保·巴克曼（P.Bachmann）于 1892 年在《解析数论》（Analytische Zahlentheorie）一书中引进的“大O”记号。这个记号，允许我们用“$=$”号来代替“$\approx$”号。例如，

$$H_n=\ln n+\gamma+O\left(\frac{1}{n}\right) \tag{1}$$

（读作，H下标 n 等于 n 的自然对数加上欧拉常数加上 n 分之一的大O）。

一般说来，当 $f(n)$是正整数 n 的一个函数时，就可以使用记号 $O(f(n))$；它表示一个这样的数量，即我们并不明显地知道它，而只知道它的绝对值不太大。记号 $O(f(n))$的每一个出现确切地意味着：有一个正的常数 M 使得由 $O(f(n))$表示的数 x_n，对于所有的 $n\geqslant n_0$ 都满足 $|x_n|\leqslant M|f(n)|$。我们不必说出常数 M 和 n_0 是什么，而且事实上这些常数对于 O 的每个出现往往是不同的。

例如，等式（1）意味着 $|H_n-\ln n-\gamma|\leqslant M/n$。常数 M 并没有进一步说明，但是即使我们不知道它的值，然而我们却知道数量 $O\left(\frac{1}{n}\right)$ 当 n 足够大时将是任意小的。

让我们再多考察一些例子。我们知道

$$1^2+2^2+\cdots+n^2=\frac{1}{3}n\left(n+\frac{1}{2}\right)(n+1)=\frac{1}{3}n^3+\frac{1}{2}n^2+\frac{1}{6}n$$

由此得出

$$1^2+2^2+\cdots+n^2=O(n^3) \tag{2}$$

$$1^2+2^2+\cdots+n^2=\frac{1}{3}n^3-O(n^2) \tag{3}$$

等式（3）是比等式（2）更强的命题。为了验证这些等式，我们将证明，如果$P(n)=a_0+a_1n+\cdots+a_mn^m$是一个次数为$m$或更小的任意多项式，则$P(n)=O(n^m)$。事实上，因为当$n\geqslant 1$时，

$$\begin{aligned}|P(n)|&\leqslant|a_0|+|a_1|n+\cdots+|a_m|n^m=(|a_0|/n^m+|a_1|/n^{m-1}+\cdots+|a_m|)n^m\\&\leqslant(|a_0|+|a_1|+\cdots+|a_m|)n^m\end{aligned}$$

所以我们可以取$M=|a_0|+\cdots+|a_m|$和$n_0=1$。

在近似计算中，O记号有很大的用处。因为它简明地描述了一个经常出现的概念，它又隐藏着通常未曾顾及的详细的信息。还有，假定小心一点的话，还可以以熟悉的方式来进行代数上的处理。

许多重要的代数规则都可以同O记号一起使用，但是，还必须指出某些重要的差别。最重要的考虑是单向相等性的思想：写$\frac{1}{2}n^2+n=O(n^2)$，决不写$O(n^2)=\frac{1}{2}n^2+n$（否则，由于$\frac{1}{4}n^2=O(n^2)$，就可能引出荒谬的关系$\frac{1}{4}n^2=\frac{1}{2}n^2+n$来）。我们总是使用这样的约定，即一个等式的右边不能给出比左边更多的信息，右边乃是左边的“粗略化”。

关于“=”号使用的这一约定，还可以更精确地叙述如下：“关于$O(f(n))$记号的公式可以把其看成是n之函数的集合。符号$O(f(n))$表示所有这样的函数g的集合，即存在着一个常数M，使得对于所有大的n，满足$|g(n)|\leqslant M|f(n)|$。如果S和T是函数的集合，则$S+T$就表示集合$\{g+h\mid g\subset S\text{ 且 }h\in T\}$。以类似的方式，定义$S+c$，$S-T$，$S\cdot T$，$\log S$等等。如果$\alpha(n)$和$\beta(n)$是涉及$O(f(n))$记号的公式，则记号$\alpha(n)=\beta(n)$意味着由$\alpha(n)$表示的函数集合包含于由$\beta(n)$表示的函数的集合中。”因此，对于“=”号就可以实行大多数我们已经习惯于做的运算：如果$\alpha(n)=\beta(n)$，而且$\beta(n)=\gamma(n)$，则$\alpha(n)=\gamma(n)$。还有，如果$\alpha(n)=\beta(n)$，又公式$\gamma(n)$中某处出现$\alpha(n)$，而且如果$\delta(n)$是以$\beta(n)$替换$\alpha(n)$所得到的公式，则$\gamma(n)=\delta(n)$。由这两个命题可推知，例如，如果$g(x_1,x_2,\cdots,x_m)$是无论什么样的任意实数函数，而且如果对于$1\leqslant k\leqslant m$有$\alpha_k(n)=\beta_k(n)$，则$g(\alpha_1(n),\alpha_2(n),\cdots,\alpha_m(n))=g(\beta_1(n),\beta_2(n),\cdots,\beta_m(n))$。

下边是我们可以对O记号进行的某些简单的运算：

$$f(n)=O(f(n)) \tag{4}$$

$$c\cdot O(f(n))=O(f(n))\quad\text{如果 }c\text{ 是一常数} \tag{5}$$

$$O(f(n))+O(f(n))=O(f(n)) \tag{6}$$

$$O(O(f(n)))=O(f(n)) \tag{7}$$

$$O(f(n))O(g(n))=O(f(n)g(n)) \tag{8}$$

$$O(f(n)g(n))=f(n)O(g(n)) \tag{9}$$

这种O记号还经常地用于一个实变量x的函数。这时，必须说明x取值的一个具体的范围，例如，$a\leqslant x\leqslant b$。而且写$O(f(x))$来表示任何这样的数量$g(x)$，它使得$|g(x)|\leqslant M|f(x)|$每当$a\leqslant x\leqslant b$时（和前面一样，M是一个未加确定的常数）。上面所讨论的记号$O(f(n))$是把变量x限制为正整数值的特殊情况。在这种特殊情况下，通常叫变量为n而不叫做x。

假设$g(x)$是一个由无穷级数给出的函数，

$$g(x)=\sum_{k\geqslant 0}a_kx^k,\quad |x|\leqslant r$$

其中绝对值的和$\sum_{k\geqslant 0}|a_kx^k|$也存在，则我们总能写

$$g(x)=a_0+a_1x+\cdots+a_mx^m+O(x^{m+1})\quad |x|\leqslant r \tag{10}$$

因为，$g(x)=a_0+a_1x+\cdots+a_mx^m+x^{m+1}(a_{m+1}+a_{m+2}x+\cdots)$；这样我们仅仅必须证明：当$|x|\leqslant r$时带圆括号的量是有界的。其实容易看出，$|a_{m+1}|+|a_{m+2}|r+|a_{m+3}|r^2+\cdots$就是一个上界。

例如，考虑在1.2.9小节中给出的生成函数。我们有重要的关系式

$$e^x=1+x+\frac{1}{2!}x^2+\cdots+\frac{1}{m!}x^m+O(x^{m+1})$$

$$|x|\leqslant r\quad \text{任意固定的 } r \tag{11}$$

$$\ln(1+x)=x-\frac{1}{2}x^2+\cdots+\frac{(-1)^{m+1}}{m}x^m+O(x^{m+1})$$

$$|x|\leqslant r\quad \text{任意固定的 } r<1 \tag{12}$$

$$(1+x)^\alpha=1+\alpha x+\binom{\alpha}{2}x^2+\cdots+\binom{\alpha}{m}x^m+O(x^{m+1})$$

$$|x|\leqslant r\quad \text{任意固定的 } r<1 \tag{13}$$

r是“固定的”这一要求意味着，当使用O记号时r必须有一个确定的值。当$|x|\leqslant r$时，我们显然有$e^x=O(1)$，因为$|e^x|\leqslant e^r$。但由O记号所隐含的常数M是依赖于r的。事实上，容易看出，如果允许x取遍$-\infty<x<\infty$的所有的值，则对于任意的m，$e^x\neq O(x^m)$。

现在让我们给出一个我们至此所介绍的这些概念的简单例子。考虑数量$\sqrt[n]{n}$。当n很大时，开n次方的运算趋向于使这个值减小。但$\sqrt[n]{n}$究竟是减小还是增加，并不是立即就能看出的。结果是$\sqrt[n]{n}$减小而成1。现在让我们考虑稍微更复杂些的数量$n(\sqrt[n]{n}-1)$。现在$(\sqrt[n]{n}-1)$当n越增大时就越减小，而$n(\sqrt[n]{n}-1)$将发生什么样的情况呢？

通过应用上边的公式，这个问题就比较容易解决了。我们有

$$\sqrt[n]{n}=e^{\ln n/n}=1+(\ln n/n)+O((\ln n/n)^2) \tag{14}$$

这个等式证明了我们前边的论点，即$\sqrt[n]{n}\to 1$。进一步，它告诉我们

$$n(\sqrt[n]{n}-1)=n(\ln n/n+O((\ln n/n)^2))=\ln n+O((\ln n)^2/n)$$

所以，我们发现$n(\sqrt[n]{n}-1)$近似地等于$\ln n$；其差是$O((\ln n)^2/n)$，当n趋于无穷时该差趋于0（见习题8）。

习题

1.〔$HM01$〕 $\lim_{n\to\infty}O(n^{-1/3})$等于什么？

▶2.〔$M10$〕 通过使用公式$O(f(n))-O(f(n))=0$，布·西·杜尔(B.C.Dull)先生得出了令人惊讶的结果。他的错误是什么？他的公式的右边应当是什么？

3.〔$M15$〕 以$(n+O(\sqrt{n}))$来乘$(\ln n+\gamma+O(1/n))$，并以O记号来表达你的答案。

▶4.〔$M15$〕 如果$a>0$，试给出$n(\sqrt[n]{a}-1)$的渐近展开式到项$O(1/n^3)$。

5.〔$M20$〕 (a) 给定$r>0$和$P(x)=c_0+c_1x+\cdots+c_mx^m$，证明当$x\geqslant r$时$P(x)=O(x^m)$。(b) 证明或者反驳：当$x>0$时$P(x)=O(x^m)$。

▶6.〔$M20$〕 以下的论证中错误何在：“因为$n=O(n)$，$2n=O(n)\cdots$，故我们有

$$\sum_{1\leqslant k\leqslant n} kn = \sum_{1\leqslant k\leqslant n} O(n) = O(n^2)。"$$

7.〔$HM15$〕 证明：如果允许 x 的值任意大，则对于任何次幂 m，$e^x \neq O(x^m)$。

8.〔$HM20$〕 证明当 $n\to\infty$ 时 $(\ln n)^m/n \to 0$。

9.〔$HM20$〕 证明 $e^{O(x^m)} = 1 + O(x^m)$，$|x| \leqslant r$。

10.〔$HM22$〕 对于 $\ln(1+O(x^m))$ 作出类似于上题的一个命题。

11.〔$M11$〕 说明等式（14）为什么是正确的。

＊1.2.11.2 欧拉求和公式 为了得到好的近似表达式，也许最有用的方法要算 1732 年欧拉给出的一个方法了。他的方法是以一个积分来近似一个有限和，而且给了我们一个在许多情况下获得好上加好的近似表达式的手段〔Commentarii Acadamiæ Petropolitanæ 6 (1732)，68～97〕。

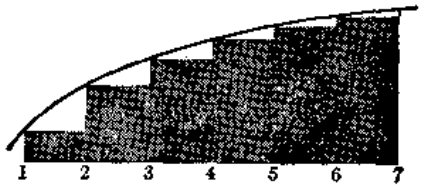


图12 一个和数与一个积分之比较

图 12 示明了积分 $\int_0^n f(x)dx$ 与 $\sum_{1\leqslant k<n} f(k)$，$n=7$ 之比较。假定 $f(x)$ 是一个可微函数，则欧拉方法给出了一个关于这两个量之差的有用的公式。

为方便起见，我们使用记号

$$\{x\} = x \bmod 1 = x - \lfloor x \rfloor \tag{1}$$

我们的推导从下列的恒等式开始：

$$\int_k^{k+1}\left(\{x\}-\frac{1}{2}\right)f'(x)dx = \left(x-k-\frac{1}{2}\right)f(x)\Big|_k^{k+1} - \int_k^{k+1} f(x)dx$$

$$= \frac{1}{2}(f(k+1)+f(k)) - \int_k^{k+1} f(x)dx \tag{2}$$

（这是由分部积分得到的）。对于 $1\leqslant k<n$ 来加这个等式的两边，我们求得

$$\int_1^n\left(\{x\}-\frac{1}{2}\right)f'(x)dx = \sum_{1\leqslant k<n} f(k) + \frac{1}{2}(f(n)-f(1)) - \int_1^n f(x)dx$$

即是，

$$\sum_{1\leqslant k<n} f(k) = \int_1^n f(x)dx - \frac{1}{2}(f(n)-f(1)) + \int_1^n B_1(\{x\})f'(x)dx \tag{3}$$

其中 $B_1(x)$ 是多项式 $x-\frac{1}{2}$。这就是所求的和数与积分之间的关系式。

如果我们继续进行分部积分，就可以求得更进一步的近似值。但在这样做以前，我们需要先来讨论一下贝努利数，这就是下列无穷级数的系数：

$$\frac{x}{e^x-1} = B_0 + B_1x + \frac{B_2x^2}{2!} + \cdots = \sum_{k\geqslant 0}\frac{B_kx^k}{k!} \tag{4}$$

这个级数的系数，出现在各种各样的问题中，它是由詹姆斯·贝努利于1713年引进的（某些书中对于贝努利数使用了不同的记号。但在最近的文献中，则都使用上面的记号）。我们有

$$B_0=1,\quad B_1=-\frac{1}{2},\quad B_2=\frac{1}{6},\quad B_3=0,\quad B_4=-\frac{1}{30} \tag{5}$$

在附录B中还给出了进一步的值。由于

$$\frac{x}{e^x-1}+\frac{x}{2}=\frac{x}{2}\,\frac{e^x+1}{e^x-1}=-\frac{x}{2}\,\frac{e^{-x}+1}{e^{-x}-1}$$

是一个偶函数，我们看出

$$B_3=B_5=B_7=B_9=\cdots=0 \tag{6}$$

如果我们以 e^x-1 来乘定义（贝努利数的）等式（4）的两边，而且写下两边 x 的高次幂的系数之等式，则我们就得到公式

$$\sum_k\binom{n}{k}B_k=B_n+\delta_{n1} \tag{7}$$

（参照等式1.2.9-11）。我们现在定义“贝努利多项式”

$$B_m(x)=\sum_k\binom{m}{k}B_kx^{m-k} \tag{8}$$

如果 $m=1$，则 $B_1(x)=B_0x+B_1=x-\frac{1}{2}$，正好是上边的等式（3）中所用的多项式。如果 $m>1$，则由公式（7），我们有 $B_m(1)=B_m=B_m(0)$；换句话说，$B_m(\{x\})$ 在整数点 x 处没有间断。

贝努利多项式和贝努利数与我们的问题之关联，马上就清楚了。我们从等式（8）求出

$$B_m'(x)=\sum_k\binom{m}{k}(m-k)B_kx^{m-k-1}=m\sum_k\binom{m-1}{k}B_kx^{m-1-k}=mB_{m-1}(x) \tag{9}$$

而且，由此，当 $m\geqslant 1$ 时，我们可如下地进行分部积分

$$\frac{1}{m!}\int_1^n B_m(\{x\})f^{(m)}(x)dx=\frac{1}{(m+1)!}(B_{m+1}(1)f^{(m)}(n)-B_{m+1}(0)f^{(m)}(1))$$
$$-\frac{1}{(m+1)!}\int_1^n B_{m+1}(\{x\})f^{(m+1)}(x)dx$$

从这个结果出发，我们即可继续来改进近似表达式（3），而且得到欧拉的一般公式：

$$\sum_{1\leqslant k<n}f(k)=\int_1^n f(x)dx-\frac{1}{2}(f(n)-f(1))+\frac{B_2}{2!}(f'(n)-f'(1))+\cdots$$
$$+\frac{(-1)^mB_m}{m!}(f^{(m-1)}(n)-f^{(m-1)}(1))+R_m$$
$$=\int_1^n f(x)dx+\sum_{1\leqslant k\leqslant m}\frac{B_k}{k!}(f^{(k-1)}(n)-f^{(k-1)}(1))+R_m \tag{10}$$

由于（6），其中

$$R_m=\frac{(-1)^{m+1}}{m!}\int_1^n B_m(\{x\})f^{(m)}(x)dx \tag{11}$$

当$B_m(\{x\})f^{(m)}(x)/m!$非常小时，余式R_m也将很小。而且事实上，已知当m是偶数时，$|B_m(\{x\})|\leqslant|B_m|$，并且

$$\left|\frac{B_m(\{x\})}{m!}\right|<\frac{4}{(2\pi)^m} \tag{12}$$

〔见康·诺普的《无穷级数的理论和应用》（格拉斯哥：布莱其，1951），第14章〕。另一方面，通常得出的结果是，当m增大时，$f^{(m)}(x)$也随之变大，所以m有一个使R_m取最小值的"最好"的值。

假设当x从1增加到n时，$f^{(2k+1)}(x)$和$f^{(2k+3)}(x)$单调地趋于0，则已经知道

$$R_{2k}=\theta\frac{B_{2k+2}}{(2k+2)!}(f^{(2k+1)}(n)-f^{(2k+1)}(1))\quad 0<\theta<1 \tag{13}$$

（所以在这种情况下，余式与第一个删去的项同号，而且小于该删去项）。习题3是这个结果的较简单的形式。

现在让我们来举一些应用欧拉公式的重要的例子。首先，我们置$f(x)=1/x$。导数是$f^{(m)}(x)=(-1)^m m!/x^{m+1}$，所以由等式(10)，我们有

$$H_{n-1}=\ln n+\sum_{1\leqslant k\leqslant m}\frac{B_k}{k}(-1)^{k-1}\left(\frac{1}{n^k}-1\right)+R_{mn} \tag{14}$$

现在我们求出

$$\gamma=\lim_{n\to\infty}(H_{n-1}-\ln n)=\sum_{1\leqslant k\leqslant m}\frac{B_k(-1)^k}{k}+\lim_{n\to\infty}R_{mn} \tag{15}$$

而$\lim_{n\to\infty}R_{mn}=-\int_1^\infty(B_m(\{x\})/x^{m+1})dx$存在之事实，说明了常数$\gamma$事实上是存在的。现在把等式(14)和(15)合在一起，我们就导出了一个对于调和数的一般的近似表达式：

$$\begin{aligned}H_{n-1}&=\ln n+\gamma+\sum_{1\leqslant k\leqslant m}\frac{(-1)^{k-1}B_k}{kn^k}+\int_n^\infty\frac{B_m(\{x\})dx}{x^{m+1}}\\&=\ln n+\gamma+\sum_{1\leqslant k\leqslant m}\frac{(-1)^{k-1}B_k}{kn^k}+O\left(\frac{1}{n^m}\right)\end{aligned} \tag{16}$$

进一步，根据等式(13)，我们看出，误差小于删去的头一项。作为一个特殊情况（两边加以$1/n$），我们有

$$H_n=\ln n+\gamma+\frac{1}{2n}-\frac{1}{12n^2}+\frac{1}{120n^4}-\epsilon,\quad 0<\epsilon<\frac{B_6}{6n^6}=\frac{1}{252n^6}$$

这就是等式1.2.7-3。对于大的k值，贝努利数B_k也非常之大〔当k是偶数时，近似于$2(k!/(2\pi)^k)$〕。所以对于任何固定的n值，等式(16)不能扩充成一个收敛的无穷级数。

同样的技术也可以用来导出斯特林近似公式。这次我们置$f(x)=\ln x$，并应用等式(10)，我们得到

$$\ln(n-1)!=n\ln n-n+1-\frac{1}{2}\ln n+\sum_{1<k\leqslant m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{n^{k-1}}-1\right)+R_{mn} \tag{17}$$

如上边那样进行，我们推知

$$\lim_{n\to\infty}\left(\ln n!-n\ln n+n-\frac{1}{2}\ln n\right)=1+\sum_{1<k\leqslant m}\frac{B_k(-1)^{k+1}}{k(k-1)}+\lim_{n\to\infty}R_{mn}$$

存在；暂且把它叫做 σ（“斯特林常数”）。我们得到了斯特林的结果

$$\ln n!=\left(n+\frac{1}{2}\right)\ln n-n+\sigma+\sum_{1<k\leqslant m}\frac{B_k(-1)^k}{k(k-1)n^{k-1}}+O\left(\frac{1}{n^m}\right) \tag{18}$$

特别地，令 $m=5$，我们有

$$\ln n!=\left(n+\frac{1}{2}\right)\ln n-n+\sigma+\frac{1}{12n}-\frac{1}{360n^3}+O\left(\frac{1}{n^5}\right)$$

取指数，我们有

$$n!=e^{\sigma}\sqrt{n}\left(\frac{n}{e}\right)^n\exp\left(\frac{1}{12n}-\frac{1}{360n^3}+O\left(\frac{1}{n^5}\right)\right)$$

利用 $e^{\sigma}=\sqrt{2\pi}$ 之事实（见习题 5），并把这个指数函数展开，我们就得到了最后的结果

$$n!=\sqrt{2\pi n}\left(\frac{n}{e}\right)^n\left(1+\frac{1}{12n}+\frac{1}{288n^2}-\frac{139}{51840n^3}-\frac{571}{2488320n^4}+O\left(\frac{1}{n^5}\right)\right) \tag{19}$$

习题

1.〔$M18$〕 证明等式（7）。

2.〔$HM20$〕 注意对于任意序列 B_n，而不仅仅是对于由等式（4）定义的序列，由等式（8）都可得出等式（9）。为使等式（10）成立，为什么却又必须是由等式（4）定义的序列？试说明之。

3.〔$HM20$〕 如果 $f^{(2k)}(x)$ 对于 $1\leqslant x\leqslant n$ 有一个固定的正负号，试证

$$|R_{2k}|\leqslant\left|\frac{B_{2k}}{(2k)!}(f^{(2k-1)}(n)-f^{(2k-1)}(1))\right|$$

所以余式的绝对值小于最近新计算出来的那项的绝对值。

▶4.〔$HM20$〕 当 $f(x)=x^m$ 时，f 的高阶微商全都为 0。所以借助于贝努利数，欧拉求和公式给出了一个对于 $\Sigma_{0\leqslant k\leqslant n}k^m$ 的精确的值。试通过贝努利多项式来表达这个值，并对于 $m=0$，1，2来校验你的答案（注意所需的求和是从 0 到 n 进行而不是从 1 到 n；欧拉求和公式可以全部地应用到以 0 代替 1 的情形）。

5.〔$HM30$〕 给定

$$n!=k\sqrt{n}\left(\frac{n}{e}\right)^n\left(1+O\left(\frac{1}{n}\right)\right)$$

试利用沃利斯乘积（习题 1.2.5-18）证明 $k=\sqrt{2\pi}$ 〔提示：对充分大的 n 值考虑 $\binom{2n}{n}$〕。

▶6.〔$HM30$〕 证明：对于非整的 n 值，斯特林近似公式也成立，即证明：

$$\Gamma(x+1)=\sqrt{2\pi x}\left(\frac{x}{e}\right)^x\left(1+O\left(\frac{1}{x}\right)\right)\qquad x\geqslant a>0$$

〔提示：在欧拉求和公式中，设 $f(x)=\ln(x+c)$，并应用在 1.2.5 小节给出的 $\Gamma(x)$ 的定义〕。

▶7.〔$HM32$〕 $1^1\cdot2^2\cdot3^3\cdot\cdots\cdot n^n$ 的近似值是什么？

***1.2.11.3 一些渐近计算** 在这一小节里我们将研究以下三个令人感兴趣的和式，目的是要导出它们的近似值：

$$P(n)=1+\frac{n-1}{n}+\frac{n-2}{n}\cdot\frac{n-2}{n-1}+\cdots=\sum_{0\leqslant k\leqslant n}\frac{(n-k)^k(n-k)!}{n!} \tag{1}$$

$$Q(n)=1+\frac{n-1}{n}+\frac{n-1}{n}\cdot\frac{n-2}{n}+\cdots=\sum_{1\leqslant k\leqslant n}\frac{n!}{(n-k)!n^k} \tag{2}$$

$$R(n)=1+\frac{n}{n+1}+\frac{n}{n+1}\cdot\frac{n}{n+2}+\cdots=\sum_{0\leqslant k}\frac{n!n^k}{(n+k)!} \tag{3}$$

这些函数，在表面上很类似，但是实质上却很不同。在我们后边即将分析的若干算法中，将要遇到这些函数。$P(n)$和$Q(n)$都是有限和，而$R(n)$是一个无限和。看上去，当n很大时，好象是所有这三个和都将近乎相等，尽管它们每个的近似值是什么并不明显。关于求这些函数的近似值问题之研究还将为我们导出一些非常有启示的附带的结果来（希望读者能够暂时停止往下阅读，先自己动手尝试来研究这些函数，然后再继续往下阅读，看看这里是如何着手解决这些函数问题的）。

首先，我们要注意到$Q(n)$与$R(n)$之间的一个重要的联系：

$$Q(n)+R(n)=\frac{n!}{n^n}\left(\left(1+n+\cdots+\frac{n^{n-1}}{(n-1)!}\right)+\left(\frac{n^n}{n!}+\frac{n^{n+1}}{(n+1)!}+\cdots\right)\right)=\frac{n!e^n}{n^n} \tag{4}$$

下一步，我们必须来考虑对于e^n的级数的部分求和。通过使用带余项的公式

$$f(x)=f(0)+f'(0)x+\cdots+\frac{f^{(n)}(0)x^n}{n!}+\int_0^x\frac{t^n}{n!}f^{(n+1)}(x-t)dt \tag{5}$$

我们立即导出一个重要的函数，通常称为不完全的伽玛（Γ）函数：

$$\gamma(a,x)=\int_0^x e^{-t}t^{a-1}dt \tag{6}$$

我们将假定$a>0$。由习题1.2.5-20，我们有$\gamma(a,\infty)=\Gamma(a)$。这就是之所以取名“不完全的伽玛函数”的理由。它有两个有用的关于x的幂级数展开（见习题2和3）：

$$\gamma(a,x)=\frac{x^a}{a}-\frac{x^{a+1}}{a+1}+\frac{x^{a+2}}{2!(a+2)}-\cdots=\sum_{k\geqslant 0}\frac{(-1)^kx^{k+a}}{k!(k+a)} \tag{7}$$

$$e^x\gamma(a,x)=\frac{x^a}{a}+\frac{x^{a+1}}{a(a+1)}+\frac{x^{a+2}}{a(a+1)(a+2)}+\cdots=\sum_{k\geqslant 0}\frac{x^{k+a}}{a(a+1)\cdots(a+k)} \tag{8}$$

从这第二个公式，我们看出了与$R(n)$的联系：

$$R(n)=\frac{n!e^n}{n^n}\left(\frac{\gamma(n,n)}{(n-1)!}\right) \tag{9}$$

这个等式被有意地写成了一种不应有的复杂的形式，因为$\gamma(n,n)$是$\gamma(n,\infty)=\Gamma(n)=(n-1)!$的一部分。于是$R(n)$位于0和$n!e^n/n^n$之间的某处；由斯特林公式，$n!e^n/n^n$近似于$\sqrt{2\pi n}$。

问题就归结成为：求得对于$\gamma(n,n)/(n-1)!$的好的估计。我们现在将确定$\gamma(x+1,x+y)/\Gamma(x+1)$当$y$固定而$x$很大时的近似值。这里所用的方法比起所得

的结果来更为重要，所以读者应当仔细地研究以下的推导。

按定义，我们有

$$\frac{\Upsilon(x+1,x+y)}{\Gamma(x+1)}=\frac{1}{\Gamma(x+1)}\int_0^{x+y}e^{-t}t^x dt$$

$$=1-\frac{1}{\Gamma(x+1)}\int_x^{\infty}e^{-t}t^x dt+\frac{1}{\Gamma(x+1)}\int_x^{x+y}e^{-t}t^x dt$$

$$=1-I_1+I_2 \tag{10}$$

现在我们分别地考虑每一个积分。

I_1 的估计：在积分 I_1 中，我们通过替换 $t=x(1+u)$ 把积分转换成从 0 到无穷：

$$I_1=\frac{e^{-x}x^x}{\Gamma(x+1)}\int_0^{\infty}xe^{-xu}(1+u)^x du=\frac{e^{-x}x^x}{\Gamma(x+1)}\int_0^{\infty}xe^{-xv}\left(1+\frac{1}{u}\right)dv \tag{11}$$

如果 $$v=u-\ln(1+u);\quad dv=\left(1-\frac{1}{1+u}\right)du$$

把变量从 u 变换成 v 是正确的，因为 v 是 u 的一个单调函数。

在最后的积分中，我们将以 v 的一个幂级数来替代 $1+1/u$。我们有

$$v=\frac{1}{2}u^2-\frac{1}{3}u^3+\frac{1}{4}u^4-\frac{1}{5}u^5+\cdots=(u^2/2)\left(1-\frac{2}{3}u+\frac{1}{2}u^2-\frac{2}{5}u^3+\cdots\right)$$

如果 $w=\sqrt{2v}$，则我们有

$$w=u\left(1-\frac{2}{3}u+\frac{1}{2}u^2-\frac{2}{5}u^3+\cdots\right)^{1/2}=u-\frac{1}{3}u^2+\frac{7}{36}u^3-\frac{73}{540}u^4+\frac{1331}{12960}u^5+O(u^6)$$

（由二项式定理即可得到这个展开式。4.7节中讨论了为进行这一变换，以及以下所做的其它幂级数处理的有效方法）。我们现在可把 u 解作 w 的幂级数：

$$u=w+\frac{1}{3}w^2+\frac{1}{36}w^3-\frac{1}{270}w^4+\frac{1}{4320}w^5+O(w^6)$$

$$1+\frac{1}{u}=1+\frac{1}{w}-\frac{1}{3}+\frac{1}{12}w-\frac{2}{135}w^2+\frac{1}{864}w^3+O(w^4)$$

$$=\frac{1}{\sqrt{2}}v^{-1/2}+\frac{2}{3}+\frac{\sqrt{2}}{12}v^{1/2}-\frac{4}{135}v+\frac{\sqrt{2}}{432}v^{3/2}+O(v^2) \tag{12}$$

在所有这些公式中，O 记号都是用于自变量的小的值，即是，$|u|\leqslant r$，$|v|\leqslant r$，$|w|\leqslant r$，对于充分小的正 r。这已足够充分了吗？在等式（11）中，v 替换 $1+1/u$ 是假定的对于 $0\leqslant v<\infty$ 成立，而不仅仅对于 $|v|\leqslant r$ 啊！幸好，原来从 0 到 ∞ 的积分值几乎完全依赖于被积函数在 0 附近的值。事实上，对于任何固定的 $r>0$ 和对于很大的 x，我们有（见习题 4）

$$\int_r^{\infty}xe^{-xv}\left(1+\frac{1}{u}\right)dv=O(e^{-rx}) \tag{13}$$

我们感兴趣的只是直到 $O(x^{-m})$ 的近似值，而且因为对于任何正的 r，m，$O((1/e^r)^x)$ 比起 $O(x^{-m})$ 来要小得多，所以我们仅需要对于任何固定的正 r 的从 0 到 r 的积分。因此，我们可取 r 足够小，使得上边所作的所有的幂级数的处理都是完全正确的（参照等式 1.2.11.1-10和12）。

现在

$$\int_0^\infty x e^{-xv} v^n dv = \frac{1}{x^{n+1}} \int_0^\infty e^{-q} q^n dq = \frac{1}{x^{n+1}} \Gamma(n+1), \quad 若 n > -1 \tag{14}$$

所以，通过把等式（12）的级数代入到等式（11）的积分中，最后我们就得到

$$I_1 = \frac{e^{-x}x^x}{\Gamma(x+1)} \left(\sqrt{\frac{\pi}{2}} x^{1/2} + \frac{2}{3} + \frac{\sqrt{2\pi}}{24} x^{-1/2} - \frac{4}{135} x^{-1} + \frac{\sqrt{2\pi}}{576} x^{-3/2} + O(x^{-2}) \right) \tag{15}$$

I_2 的估计：在积分 I_2 中，我们替入 $t = u + x$ 并得到

$$I_2 = \frac{e^{-x}x^x}{\Gamma(x+1)} \int_0^y e^{-u} \left(1 + \frac{u}{x}\right)^x du \tag{16}$$

现在对于 $0 \leqslant u \leqslant y$ 和很大的 x，

$$e^{-u}\left(1 + \frac{u}{x}\right)^x = \exp\left(-u + x \ln\left(1 + \frac{u}{x}\right)\right) = \exp\left(\frac{-u^2}{2x} + \frac{u^3}{3x^2} + O(x^{-3})\right)$$

$$= 1 - \frac{u^2}{2x} + \frac{u^4}{8x^2} + \frac{u^3}{3x^2} + O(x^{-3})$$

因此我们求出

$$I_2 = \frac{e^{-x}x^x}{\Gamma(x+1)} \left(y - \frac{y^3}{6} x^{-1} + \left(\frac{y^4}{12} + \frac{y^5}{40}\right) x^{-2} + O(x^{-3}) \right) \tag{17}$$

最后，我们来分析出现在等式（15）和（17）中的系数 $e^{-x}x^x/\Gamma(x+1)$。由斯特林近似公式（由习题1.2.11.2-6，我们知道此公式对伽玛函数 Γ 是有效的），我们得到

$$\frac{e^{-x}x^x}{\Gamma(x+1)} = \frac{e^{-1+1/12x+O(x^{-3})}}{\sqrt{2\pi x}}$$

$$= \frac{1}{\sqrt{2\pi}} x^{-1/2} - \frac{1}{12\sqrt{2\pi}} x^{-3/2} + \frac{1}{288\sqrt{2\pi}} x^{-5/2} + O(x^{-7/2}) \tag{18}$$

现在来个大总结——把等式（10），（15），（17）和（18）综合在一起．我们就有

定理 A 对于很大的 x 值和固定的 y，

$$\frac{\Gamma(x+1, x+y)}{\Gamma(x+1)} = \frac{1}{2} + \left(\frac{y - 2/3}{\sqrt{2\pi}}\right) x^{-1/2} + \frac{1}{\sqrt{2\pi}} \left(\frac{23}{270} - \frac{y}{12} - \frac{y^3}{6}\right) x^{-3/2} + O(x^{-5/2}) \tag{19}$$

我们所使用的方法还说明了怎样把这个近似公式推广到我们所要求的 x 的任何更高的次幂。

·通过利用等式（4）和（9），这个定理即可用来获得 $R(n)$ 和 $Q(n)$ 的近似值．但是我们将把这个计算推迟到后边去做。现在让我们回过头来看 $P(n)$，对于它，似乎需要某些不同的方法。

$$P(n) = \sum_{0 \leqslant k \leqslant n} \frac{k^{n-k} k!}{n!} = \frac{\sqrt{2\pi}}{n!} \sum_{0 \leqslant k \leqslant n} k^{n+1/2} e^{-k} \left(1 + \frac{1}{12k} + O(k^{-2})\right) \tag{20}$$

因此为得到 $P(n)$ 的值，我们就必须研究形如

$$\sum_{0 \leqslant k \leqslant n} k^{n+1/2} e^{-k}$$

的和式。

命 $f(x)=x^{n+1/2}e^{-x}$ 并应用欧拉求和公式：

$$\sum_{0\leqslant k\leqslant n} k^{n+1/2}e^{-k}=\int_0^n x^{n+1/2}e^{-x}dx+\frac{1}{2}n^{n+1/2}e^{-n}+\frac{1}{24}n^{n-1/2}e^{-n}-R \tag{21}$$

对余式的分析(参照习题 5)表明 $R=O(n^{n-1/2}e^{-n})$；而且因为这个积分是一个不完全的伽玛函数，我们有

$$\sum_{0\leqslant k\leqslant n} k^{n+1/2}e^{-k}=\gamma\left(n+\frac{3}{2},\ n\right)+\frac{1}{2}n^{n+1/2}e^{-n}+O(n^{n-1/2}e^{-n}) \tag{22}$$

我们的公式，等式（20），还要求对于和式

$$\sum_{0\leqslant k\leqslant n} k^{n-1/2}e^{-k}=\sum_{0\leqslant k\leqslant n-1} k^{(n-1)+1/2}e^{-k}+n^{n-1/2}e^{-n}$$

的估计，而且这也可以由等式（22）得到。

至此，我们为确定 $P(n)$，$Q(n)$ 和 $R(n)$ 之近似值而进行的处理，已经为我们提供了足够的公式了！剩下的只不过是进行替换和相乘等等问题。在这个过程中，我们还需要用到展开式

$$(n+\alpha)^{n+\beta}=n^{n+\beta}e^{\alpha}\left(1+\alpha\left(\beta-\frac{\alpha}{2}\right)\frac{1}{n}+O(n^{-2})\right) \tag{23}$$

习题 6 将对此式进行证明。(21)的方法仅仅得出对于 $P(n)$ 的渐近级数的头三项；通过使用习题 14 中所描述的具有启发性的技术，可以得到进一步的项。

所有这些计算的结果给了我们所需要的渐近公式：

$$P(n)=\sqrt{\frac{\pi n}{2}}-\frac{2}{3}+\frac{11}{24}\sqrt{\frac{\pi}{2n}}+\frac{4}{135n}-\frac{71}{1152}\sqrt{\frac{\pi}{2n^3}}+O(n^{-2}) \tag{24}$$

$$Q(n)=\sqrt{\frac{\pi n}{2}}-\frac{1}{3}+\frac{1}{12}\sqrt{\frac{\pi}{2n}}-\frac{4}{135n}+\frac{1}{288}\sqrt{\frac{\pi}{2n^3}}+O(n^{-2}) \tag{25}$$

$$R(n)=\sqrt{\frac{\pi n}{2}}+\frac{1}{3}+\frac{1}{12}\sqrt{\frac{\pi}{2n}}+\frac{4}{135n}+\frac{1}{288}\sqrt{\frac{\pi}{2n^3}}+O(n^{-2}) \tag{26}$$

这里所研究的函数，在公开发表的著作中只是略加叙述过。$P(n)$ 展开式中的头一项 $\sqrt{\pi n/2}$ 是由霍·B·德穆思（H.B.Demuth）给出的〔斯坦福大学博士论文，1956 年 10月，67～68〕。利用这一结果，和一张对于 $n\leqslant 2000$ 的 $P(n)$ 的数值表，以及一支好的计算尺，作者导出了一个经验估计 $P(n)\approx\sqrt{\pi n/2}-0.6667+0.575/\sqrt{n}$。自然地会猜测 0.6667实际上是 $\frac{2}{3}$ 的近似值，而 0.575 也许来自于 $\gamma=0.57721\cdots$ 的近似值。（但为什么又不是很适切的？）后来，才象这一小节所写出的这样，得出上列的 $P(n)$ 的展开式，而且验证了对于 $\frac{2}{3}$ 的猜测；对于0.575，我们不是从 γ 得来，而是从 $\frac{11}{24}\sqrt{\pi/2}\approx 0.574$。这就使理论和经验估计很好地相符了。

等价于$Q(n)$和$R(n)$之渐近值的公式首先是由卓越的印度的自学数学家斯·拉马纳燕确定的。他在《印度数学学会杂志》(*J. Indian Math. Soc.*) 3 (1911), 128; 4 (1912), 151～152 上提出了估计 $n!e^n/2n^n - Q(n)$的问题。在对于这个问题的回答中，他给出了渐近级数$\frac{1}{3}+\frac{4}{135}n^{-1}-\frac{8}{2835}n^{-2}-\frac{1}{8505}n^{-3}+\cdots$，它相当大地超过了等式 (25)。他的推导比起上边描述的方法显得要更漂亮些；为了估计I_1，他替入$t = x + u\sqrt{2x}$、而且把被积函数表达成一形如$c_{jk}\int_0^\infty \exp(-u^2)u^j n^{-k/2}du$之项的和。积分$I_2$可以完全避免掉，因为$a\gamma(a, x)=x^a e^{-x}+\gamma(a+1, x)$，当$a>0$时（见(8)）。我们所用的推导，尽管不必要地复杂，但是却富有启发性，它是由罗·弗兹 (R. Furch) 提出的〔《物理学杂志》(*Zeitschrift für Physik*) 112 (1939), 92～95〕，此人主要对使得$\gamma(x+1, x+y)=\frac{1}{2}\cdot\Gamma(x+1)$的$y$值感兴趣。关于$Q(n)$的其它研究的文献目录，请看亨沃古尔德，*AMM* 75 (1968), 1019～1021。不完全的伽玛函数的渐近性质后来被弗·贾·特里戈米 (F. G. Tricomi) 推广到复变量的情形〔《不完全的Γ函数的渐近性质》(Asymptotische Eigenschaften der unvollständigen Gamma-funktion), *Math. Zeitschrift* 53 (1950), 136～148〕。

关于函数$P(n)$, $Q(n)$和$R(n)$的进一步的研究是很有趣的。上边给出的推导仅仅使用了初等微积分的简单技术。但请注意，我们对每个函数所使用的方法是不同的！实际上我们使用习题 14 的技术就可以全部解决这三个问题。在 5.1.4 和 5.2.2 小节中我们将进一步说明这一点，那是更精采的但却稍逊启发性了。

关于进一步的介绍，有兴趣的读者可查阅尼·戈·德·布鲁因 (N. G. de Bruijn)所写的精采的书《分析中的渐近方法》(*Asymptotic Methods in Analysis*)(阿姆斯特丹：北荷兰出版社，1961)。

习题

1. 〔*HM*20〕 对n用归纳法证明等式(5)。
2. 〔*HM*20〕 由等式(6)推出等式(7)。
3. 〔*M*20〕 由等式(7)推出等式(8)。

▶4. 〔*HM*10〕 证明等式(13)。

5. 〔*HM*24〕 证明等式(21)中的R是$O(n^{n-1/2}e^{-n})$。

▶6. 〔*HM*20〕 证明等式(23)。

▶7. 〔*HM*30〕 在计算I_2时，我们已经考虑了

$$\int_c^y e^{-u}\left(1+\frac{u}{x}\right)^x du。$$

给出对于

$$\int_0^{yx^{1/4}} e^{-u}\left(1+\frac{u}{x}\right)^x du$$

的渐近表示直到$O(x^{-2})$的项，当y固定和x很大时。

8. 〔*HM*30〕 假定$0\leqslant r\leqslant\frac{1}{2}$。假设$f(x)=O(x^r)$。证明：如果$m=\lceil(s+2r)/(1-r)\rceil$，则

$$\int_0^{f(x)} e^{-u}\left(1+\frac{u}{x}\right)^x du=\int_0^{f(x)} \exp\left(\frac{-u^2}{2x}+\frac{u^3}{3x^2}-\cdots+\frac{(-1)^{m-1}u^m}{mx^{m-1}}\right)du+O(x^{-s})$$

〔这特别地证明了一个由特里戈米给出的结果：如果 $f(x)=O(\sqrt{x})$，则

$$\int_0^{f(x)} e^{-u}\left(1+\frac{u}{x}\right)^x du=\sqrt{2x}\int_0^{f(x)/\sqrt{2x}} e^{-t^2}dt+O(1)〕$$

▶9.〔$HM36$〕 对于很大的 x，$\Gamma(x+1,\ px)/\Gamma(x+1)$的状态什么样？（这里 p 是一个实常数；而且，如果 $p<0$，我们假定 x 是一个整数，所以 t^x 对于负的 t 也有定义）在求助于 O 项之前，至少要求出渐近展开式的两项来。

10.〔$HM34$〕 在上题的假定下，对 $p\neq 1$，对于固定的 y，求出 $\Gamma(x+1,px+py/(p-1))-\Gamma(x+1,px)$ 的渐近展开式，直到和上题中所得到的同样阶次的项。

▶11.〔$HM35$〕 通过引进参数 x，让我们推广函数 $Q(n)$，$R(n)$：

$$Q_x(n)=1+\left(\frac{n-1}{n}\right)x+\left(\frac{n-1}{n}\right)\left(\frac{n-2}{n}\right)x^2+\cdots$$

$$R_x(n)=1+\left(\frac{n}{n+1}\right)x+\left(\frac{n}{n+1}\right)\left(\frac{n}{n-2}\right)x^2+\cdots$$

剖析其中状况并求出当 $x\neq 1$ 时的渐近公式。

12.〔$HM20$〕 同正态分布相关而出现的函数 $\int_0^x e^{-t^2/2}dt$（见 1.2.10 小节），可以表达成不完全的伽玛函数的一个特殊情况。试求出 a，b，y 的值，使得 $b\gamma(a,y)$ 等于上述函数。

13.〔$HM46$〕 （斯·拉马纳燕）证明 $R(n)-Q(n)=\frac{2}{3}+8/(135(n-0(n)))$。其中 $\frac{2}{21}\leqslant\theta(n)\leqslant\frac{8}{45}$〔这隐涵了更弱得多的结果 $R(n+1)-Q(n+1)<R(n)-Q(n)$〕。

▶14.〔$HM39$〕 （尼·戈·德·布鲁因）本题的目的是来求出对于固定的 α，当 $n\to\infty$ 时，$\sum_{0\leqslant k\leqslant n}k^{n+\alpha}e^{-k}$ 的渐近展开。（a）以 $n-k$ 替换 k，证明给定的和等于 $n^{n+\alpha}e^{-n}\sum_{0\leqslant k<n}e^{-k^2/2n}\times f(k,n)$，其中 $f(k,n)=(1-k/n)^\alpha\exp(-k^3/3n^2-k^4/4n^3-\cdots)$。（b）证明对于所有的 $m\geqslant 0$ 和 $\epsilon>0$，$f(k,n)$ 可写成形式 $\sum_{0\leqslant i\leqslant j\leqslant m}c_{ij}k^{2i+j}n^{-i-j}+O(n^{(m+1)(-1/2+\epsilon)})$，当 $0\leqslant k\leqslant n^{1/2+\epsilon}$ 时。（c）作为（b）的一个推论，证明：对于所有的 $\delta>0$，$\sum_{0\leqslant k<n^{1/2+\delta}}k^{2i+j}e^{-k^2/2n}\times f(k,n)=\sum_{0\leqslant i\leqslant j\leqslant m}c_{ij}n^{-i-j}\sum_{k\geqslant 0}k^{2i+j}e^{-k^2/2n}+O(n^{-m/2+\delta})$。〔提示：在变程 $n^{1/2+\epsilon}<k<\infty$ 上，对于所有的 r，此和为 $O(n^{-r})$〕。（d）证明通过欧拉求和公式，可以得到对于固定的 $t\geqslant 0$ 的 $\sum_{k\geqslant 0}k^te^{-k^2/2n}$ 的渐近展开式。（e）因此，最后

$$\sum_{0\leqslant k\leqslant n}k^{n+\alpha}e^{-k}=n^{n+\alpha}e^{-n}\left(\sqrt{\frac{\pi n}{2}}-\frac{1}{6}-\alpha+\left(\frac{1}{12}+\frac{1}{2}\alpha+\frac{1}{2}\alpha^2\right)\sqrt{\frac{\pi}{2n}}+O(n^{-1})\right)$$

对于任何所希望的 r 值。这个计算原则上可推广到 $O(n^{-r})$。

15.〔$HM20$〕 说明下列积分是与 $Q(n)$ 有关的：

$$\int_0^\infty\left(1+\frac{z}{n}\right)^n e^{-z}dz$$

16. [*M*24] 证明恒等式

$$\sum_k (-1)^k \binom{n}{k} k^{n-1} Q(k) = (-1)^n (n-1)!.\ \text{当}\ n > 0\ \text{时}$$

17. [*HM*29] (肯·威·米勒)对称性促使我们也考虑第四个级数，它是与$P(n)$对偶的，就如$R(n)$与$Q(n)$对称一样：

$$S(n) = 1 + \frac{n}{n+1} + \frac{n}{n+2}\,\frac{n+1}{n+2} + \cdots = \sum_{0 \leqslant k} \frac{(n-k-1)!}{(n-1)!(n+k)^k}$$

这个函数的渐近特性如何？

1.3 MIX

本书自始至终有许多地方，都需要引用一种计算机的"机器语言"。我们所用的计算机是一台想象中的称为"MIX"的计算机。MIX 除了也许是更精巧外，几乎非常类似于现有的每一台计算机。我们把 MIX 的语言设计成足够地强有力，以便对大多数的算法，都可以写出简单的程序来。而且该语言也还足够地简单，使得它的操作是易于学习的。

希望读者仔细地来学习这一小节，因为 MIX 语言在本书的许多地方都要遇到。对于学习一种新的机器语言，应当是毫不犹豫的；其实，作者觉得，在一周时间内，以六种不同的机器语言来写程序，是不足为奇的！凡对计算机不是仅仅一时感兴趣的人，在他的一生中，大概总要接触到很多不同的机器语言。MIX 经过精心地设计，与大多数现有的机器语言极其类似，所以 MIX 的特征是易于理解的。

1.3.1 MIX 的描述

MIX 是世界上第一流的多种未饱和状态的计算机。类似于大多数的机器，它有一个标识代号——1009。这个代号是这样得出的：即取 16 台实在的计算机，它们非常类似于 MIX，而且在它们上边可以很容易地模拟MIX，然后以同等的权来求它们的代号的平均值：

$$\lfloor (360 + 650 + 709 + 7070 + \text{U3} + \text{SS80} + 1107 + 1604 + \text{G20} + \text{B220} + \text{S2000} + 920 + 601 + \text{H800} + \text{PDP4} + \text{II})/16 \rfloor = 1009 \qquad (1)$$

通过把 MIX 看作罗马数字，亦可更简单地来得到这同一个代号（在罗马数字中，M为1000 IX 为 9 ——译者注）。

MIX 有一个独特的性质，即它同时既是二进制的又是十进制的。程序员不必实际地知道他是正在以二进制还是正在以十进制来对一台机器进行程序设计。这样就使得以 MIX 语言写成的算法，稍经变化就可以为每一种类型的机器所使用，而且也使得在每一种类型的机器上可以容易地来模拟 MIX。习惯于二进制机器的那些程序员可以把 MIX 想象成是二进制的；而习惯于十进制的程序员则又可以认为 MIX 是十进制的；来自另一行业的那些程序员则也可能选择把 MIX 当作是三进制的机器。

字 信息的基本单位是字节。每个字节所包含的信息总量●，我们不予规定。但是，它必须能够保存至少 64 个不同的值。即是说，我们知道，0 与 63 之间的任何数（包括 0 和

● 即字节大小(byte size)。——译者注

63 在内）都可包含在一个字节内。另一方面，每个字节至多含有 100 个不同的值。因此，在一个二进计算机上，一个字节必须由 6 位组成；在一个十进计算机上，每一个字节有两位数字。

以 MIX 语言表达的程序应该写成，不大于 64 的值永远占用一个字节。如果我们希望来处理数 80，则我们总是保留两个相邻的字节来表达它，尽管在一个十进计算机上只一个字节就足够了。一个 MIX 的算法，应该不管字节有多大都能适当地工作。尽管完全有可能写出依赖于字节大小的程序，但那是不合法的，是不容许的。唯一合法的程序是对于所有字节大小都能给出正确结果的程序。这一根本规则通常并不难遵守，而且由此我们将发现，对一个十进计算机进行程序设计与对一个二进计算机进行程序设计，最终并没有多大差别。

两个相邻的字节可表达数 0 到 4095。

三个相邻的字节可表达数 0 到 262143。

四个相邻的字节可表达数 0 到 16777215。

五个相邻的字节可表达数 0 到 1073741823。

一个计算机字是五个字节再加一个符号位。符号位仅有两个可能的值，＋和－。

寄存器 MIX中有 9 个寄存器（见图 13）：

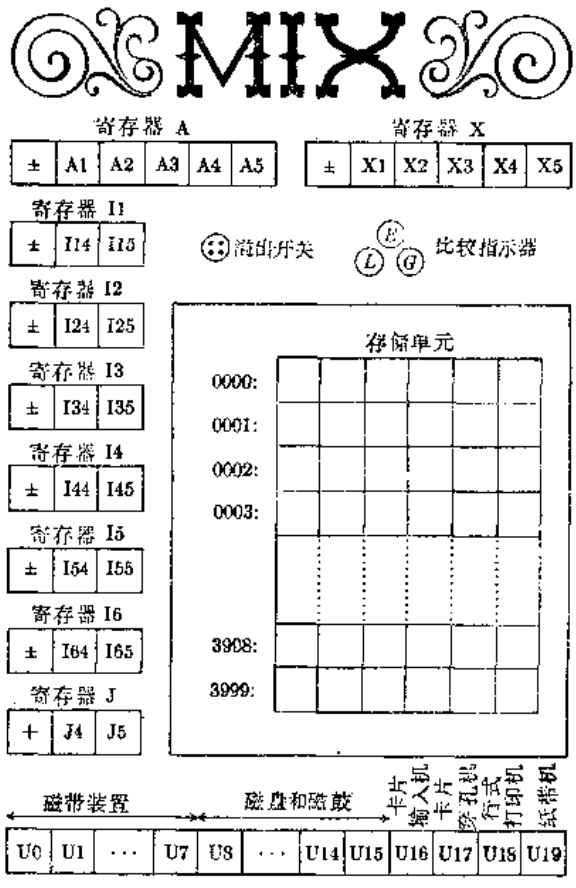


图13 MIX计算机

A寄存器（累加器）是五个字节再加一符号位。

X寄存器（扩充）也是五个字节再加一符号位。

I 寄存器（变址寄存器）I1, I2, I3, I4, I5 和 I6 每个拥有两个字节再加一符号位。

J 寄存器（转移地址）拥有两个字节，而且它的符号总是＋。

我们将使用一个冠在名称之前的小写字母“r”，来标识 MIX 的寄存器。于是，“rA”意味着“寄存器A”。

A寄存器有许多用途，特别是用作对数据进行算术运算和操作。X 寄存器是 rA 右边的一个扩充，可用它与 rA 相连接以保存一个乘积或一个被除数的十个字节，也可用它来保存右移到 rA 之外的信息。变址寄存器 rI1，rI2，rI3，rI4，rI5 和 rI6 主要用来计数和访问可变的存储器地址。J 寄存器总是保存紧跟着“JUMP(转移)”指令之后的那条指令的地址，它主要用来同子程序相连接。

除了这些寄存器外，MIX 含有

一个溢出开关（一位，或者为“开”，或者为“关”）。

一个比较指示器（它有三个值：L(小于)，E(等于)，或G(大于)。

存储器（4000 个存储字，每个字有五个字节再加一符号位)。

还有输入输出装置（卡片，带，等等)。

字的部分场 一个计算机字的五个字节和一位符号可以编号如下：

0	1	2	3	4	5
±	字节	字节	字节	字节	字节

(2)

大多数指令都允许程序员在他需要时，仅选用一个字的一部分。在这种情况下，要给出一个“场说明”。允许的场是在一个计算机字中相邻的那些字节与符号位，以（L：R）来表示，其中L是场的左边部分的号数，R是右边部分的号数。场说明的例子是：

(0：0)，仅是符号位。

(0：2)，符号位和头两个字节。

(0：5)，整个字。这是最普通的场说明。

(1：5)，除了符号位之外的整个字。

(4：4)，仅是第四个字节。

(4：5)，两个最低有效字节。

这些场说明的用法随着指令的不同而略有不同，而且在用到处，对每条指令都将详细地予以阐述。

尽管一般地对于程序员说来并不重要，场(L：R)在机器中是通过一个单一的数8L＋R来表示的，而且这个数将填入到一个字节中。

指令格式 用于指令的计算机字有如下的形式：

0	1	2	3	4	5
±	A	A	I	F	C

(3)

最右边的字节C，是说明执行的是什么操作的操作码。例如，C＝8是操作 LDA，“装入A寄存器”。

F字节保存对操作码的补充说明。F通常是一个场说明（L：R)＝8L＋R。例如，如果C＝8和F＝11，则操作就是“以(1：3）场装入A寄存器”。有时F也用于其它目的。例如，对于输入输出指令，F是代表进行输入或输出的设备的号码。

指令的左边部分±AA是“地址”(注意，符号是地址的一部分)。I场，它紧接在地址之后，是“变址说明”。这可以用来修改一条指令的地址。如果I＝0，则就不加改变地来使用±AA；否则I就应当包含1与6之间的一个数i，而且变址寄存器I的内容即代数地与±AA相加，其结果才用作指令的地址。这种变址过程每一条指令都有。我们将用字母M来表示进行了任何所规定的变址之后的地址。(如果把变址寄存器与地址±AA相加，得出了两个字节容纳不下的结果，则M的值就是无定义的。)

在大多数指令中，M将涉及到一个存储单元。我们把4000个存储单元，编号成0到3999。因此每个存储单元都可以以两个字节来编址。对于每一条指令，如果其中之M是涉及到一个存储单元的，则我们必须有$0 \leqslant M \leqslant 3999$，而且在这种情况下我们将写

CONTENTS(M) 来表示存于存储单元M中的值（M之内容）。

在某些指令中，“地址”M有另一种意义，而且它甚至可以是负的。于是，一条指令可以把M加到一个变址寄存器上，而且这个操作考虑到了M的符号。

记法 为了以易于阅读的方式来讨论指令，我们将使用记号

操作符 地址，I(F) (4)

来表示象(3)那样的指令。这里，操作符是对该指令的操作码（C部分）给出的一个符号名称；地址是±AA部分，而I和F则分别表示I和F场。

如果I为0，则省略“，I”。如果F是对于这个具体的操作符的“正常的”F说明，则也就不需要写“(F)”。对于几乎所有的操作符，正常的F说明是（0：5），表示一个全字。如果所说明的F是非正常的，则当我们讨论一个具体的操作符时，将明确地把F写出来。

例如，把一个数装入累加器的指令叫做LDA，它的操作码是8。我们有

约定的表示	实际的数码指令			
LDA 2000,2(0:3)	+	2000	2	3	8
LDA 2000,2(1:3)	+	2000	2	11	8
LDA 2000(1:3)	+	2000	0	11	8
LDA 2000	+	2000	0	5	8
LDA -2000,4	-	2000	4	5	8

(5)

如果要读这些指令，则指令“LDA 2000，2(0：3)”可以读作“地址2000，以rI2变址，所得单元的内容的0至3场装入A”。

为了表示一个MIX字的数值内容，我们将总是使用一个象上边那样的方框记号。注意在字

+	2000	2	3	8

中，数+2000填在两个相邻的字节和符号位上；而字节（1：1）和字节（2：2）的实际内容将随着MIX计算机的不同而不同，因为字节的大小是可变的。对于MIX字的这种记法的进一步的例子，有：方框

-	10000	3000

表示一个具有两个场的字，一个场是包含−10000的三字节再加符号位的场，另一个是包含3000的两个字节的场。当一个字被分成多于一个场时，就说它是“被集装”的。

每条指令的规则 上述的(3)后面的论述，已经对于用作一条指令的每个字，定义了数量M，F和C。我们现在将定义每条指令所对应的动作。

装入(Load)操作符

·LDA(装入A)．C＝8，F＝场。

CONTENTS(M)的所说明的场代替寄存器A的内容。

对于所有的那些操作，即其中一个部分场是用作输入的，如果符号是该部分场的一部分，则就使用符号，否则就把符号理解为+。随着它之被装入，这个场就被移位到寄存器的右边部分。

例　如果F是正常的场说明（0：5），则就装入单元M的整个内容。如果F是(1:5)，则装入CONTENTS(M)的绝对值连同一个正号。如果M包含一个指令字以及如果F是(0：2)，则“±AA”场被装入而成为

±	0	0	0	A	A

假设单元2000包含字

-	30	3	5	4

（6）

则我们通过装入各种部分场而得到以下的结果：

指　令	经操作后rA的内容
LDA　2000	- 30 3 5 4
LDA　2000　(1:5)	+ 30 3 5 4
LDA　2000　(3:5)	+ 0 0 3 5 4
LDA　2000　(0:3)	- 0 0 30 3
LDA　2000　(4:4)	+ 0 0 0 0 5
LDA　2000　(0:0)	- 0 0 0 0 0
LDA　2000　(1:1)	+ 0 0 0 0 ?

（最后一个例子有一个部分未知的结果，因为字节大小是可变的）。

·LDX(装入X)．C＝15；F＝场。

除了装入的是rX而不是rA外，本指令和LDA相同。

·LD*i*(装入*i*)．C＝8＋*i*；F＝场。

除了装入的是r*i*而不是rA外，这指令和LDA相同。一个变址寄存器仅含有两个(而不是五个)字节加上符号位；字节1，2，3总认为是0。如果LD*i*将导致置字节1，2，3为任何非0者，则认为此LD*i*指令无定义。

在所有指令的描述中，“*i*”总表示一个整数，$1 \leqslant i \leqslant 6$。于是，LD*i*表示六个不同的指令：LD1，LD2，…，LD6．

·LDAN(把负的装入A)．C＝16；F＝场。

·LDXN(把负的装入X)．C＝23；F＝场。

·LD*i*N（把负的装入*i*）．C＝16＋*i*；F＝场。

除了把相反的符号装入外，这八条指令分别地与LDA，LDX和LD*i*相同。

存储（Store）操作符

·STA（存储A）．C＝24；F＝场。

rA的内容代替CONTENTS(M)的由F所说明的场。CONTENTS(M)的其它部分不变。

场F对于存储操作的意义，正好与对于装入操作的意义相反。场中字节的数取自寄存器的右边，需要时就进行左移，而且插入到CONTENTS(M)的适当的场中去。除非符号是场的一部分，否则符号不改变。寄存器的内容也不受影响。

例　假设单元2000含有

-	1	2	3	4	5

且寄存器A含有

+	6	7	8	9	0

则

指 令	经操作后的单元2000的内容
STA 2000	+ 6 7 8 9 0
STA 2000(1:5)	- 6 7 8 9 0
STA 2000(5:5)	- 1 2 3 4 0
STA 2000(2:2)	- 1 0 3 4 5
STA 2000(2:3)	- 1 9 0 4 5
STA 2000(0:1)	+ 0 2 3 4 5

·STX（存储X）. C＝31； F＝场。

除了存储 rX 而不是 rA 外，和 STA 相同。

·ST*i*（存储 *i*）. C＝24＋ *i*； F＝场。

除了存储 r*i* 而不是 rA 外，和 STA 相同。变址寄存器的字节 1，2，3 都是 0，因此如果 r*i* 包含

±	*m*	*n*

则可以想象作是

±	0	0	0	*m*	*n*

·STJ（存储 J）. C＝32； F＝场。

除了存储 rJ 外，和 ST*i* 相同，而且它的符号总是＋。

对于 STJ，F 的正常场说明是（0：2），而不是（0：5）。这是自然的，因为 STJ 几乎总是对一条指令的地址场来做。

·STZ（存储 0）. C＝33； F＝场。

除了存储正号和 0 外，和 STA 相同。换句话说，就是把 CONTENTS(M) 所说明的场清成 0。

算术操作符 对于加法、减法、乘法和除法操作，允许一个场说明。场说明“(0:6)”可以用来表示“浮点”操作(见 4.2 小节)。但在我们将要为 MIX 写的程序中，只有很少几个程序使用这个特性。浮点指令将主要地用于由第 12 章中所讨论的编译程序写成的程序中。

象通常那样，正常的场说明是(0：5)。其它场的处理和 LDA 一样。我们将使用字母 V 来表示 CONTENTS(M) 所说明的场。于是，如果操作符是 LDA，则 V 就是应被装入寄存器 A 的值。

·ADD. C＝1； F＝场。

V 加到 rA 上。如果结果的数量对寄存器 A 太大，则溢出开关接通，在 rA 中出现的是加法的余数，就想象成“1”已经进位到 A 左边的另一个寄存器去了（否则溢出开关的置位不变）。如果结果为 0，则 rA 的符号位不变。

例 下列的指令序列给出了寄存器 A 的五个字节之和：

```
STA   2000
LDA   2000 (5:5)
ADD   2000 (4:4)
ADD   2000 (3:3)
ADD   2000 (2:2)
ADD   2000 (1:1)
```

这有时叫做“横加”。

·SUB（减法）. C = 2； F = 场。

从 rA 减去 V。也象在 ADD 中一样，可能出现上溢。

注意，因为按定义，字节大小是可变的，所以在某一台 MIX 计算机上出现上溢时，在另一台 MIX 计算机上却可不出现上溢。因此，我们不能说：“如果值大于 1073741823 时，则上溢将肯定地出现”，而是说：“按照字节的大小，当结果的数量大于五个字节的内容时，就出现上溢”。人们仍然能够不管字节的大小，而编制出能够适当地工作，并且给出相同的最终答案的程序来。

·MUL（乘法）. C = 3； F = 场。

V 乘（rA）的 10 个字节的乘积代替寄存器 A 和 X，rA 和 rX 的符号全都被置成结果的代数符号（即是，如果 V 和 rA 符号相同则置为 +，如果符号不同则置为 −）。

·DIV（除法）. C = 4； F = 场。

rA 和 rX 的值，处理成一个 10 字节的数，并带有 rA 的符号，除以 V 的值。如果 V = 0 或者如果商的数量大于五个字节（这等价于条件 $|rA| \geqslant |V|$），则寄存器 A 和 X 都填之以无定义的信息，且溢出开关接通。否则商存于 rA 中，而余数存于 rX 中。执行操作后 rA 的符号是商的代数符号，执行操作后 rX 的符号是 rA 以前的符号。

算术指令的例子。在大多数情况下，算术运算仅对诸 MIX 字来进行，这些字都是完整的五字节数，而不被集装成若干个场。如果遵守某些告诫，则也有可能对装的 MIX 字进行算术操作。对下面的例子应当仔细地加以研究（“?”符号表示一个未知的值）。（这些例子是本着这样一种指导思想给出的，即与其给一个不完备的直接

```
ADD  1000
```

+	1234	1	150	执行前的rA
+	100	5	50	1000单元
+	1334	6	200	执行后的rA

```
SUB  1000
```

−	1234	0	0	9	执行前的rA
−	2000	150	0		1000单元
+	766	149	?		执行后的rA

```
MUL  1000(1:1)
```

−					112	执行前的rA
?	2	?	?	?	?	1000单元
−					0	执行后的rA
−					224	执行后的rX

```
MUL  1000
```

−	50	0	112	4	执行前的rA
−	2	0	0	0	1000单元
+	100	0	224		执行后的rA
+	8	0	0	0	执行后的rX

```
DIV  1000
```

+					0	执行前的rA
+					17	执行前的rX
+					3	1000单元
+					5	执行后的rA
+					2	执行后的rX

```
DIV  1000
```

−					0	执行前的rA
+	1235	0	3	1	执行前的rX
−	0	0	0	2	0	1000单元
+	0	617	?	?		执行后的rA
−	0	0	0	?	1	执行后的rX

了当的叙述，还是就给出一个完全的复杂的叙述更好些）。

地址传送操作符 在下边的操作中，“地址”M(可能变址）当作一个带符号的数来使用，而不是作为存储器中单元的地址。

·ENTA（进入A） C＝48； F＝2。

把数量M装入rA。这个动作等价于把一个含有该带符号的M值的存储字进行“LDA”的动作。如果M＝0，则装入本指令的符号。

例 “ENTA 0”把rA置成0。“ENTA 0，1”置rA为变址寄存器i的当前的内容，但－0改变成＋0。

·ENTX（进入X） C＝55； F＝2。

·ENT i（进入 i） C＝48＋ i； F＝2。

类似于ENTA，装入相应的寄存器。

·ENNA（负的进入A） C＝48； F＝3。

·ENNX（负的进入X） C＝55； F＝3。

·ENN i （负的进入 i） C＝48＋ i； F＝3。

除了装入的是相反的符号外，与ENTA、ENTX以及ENT i 相同。

例 “ENN3 0，3”是以r13的负值代替r13。

·INCA（A增值） C＝48； F＝0。

数量M加到rA上。这个动作等价于对含有值M的一个存储字进行“ADD”的动作。有可能产生溢出，而且也就象在ADD中那样处理。

例 “INCA 1”把rA的值增加1。

·INCX（X增值） C＝55； F＝0。

数量M加到rX上。如果出现溢出，则其动作就等价于ADD的相应动作，所不同的只是以rX代替rA。这一指令决不影响寄存器A。

·INC i （ i 增值） C＝48＋ i； F＝0。

把M加到r i。不允许出现溢出。如果得到的结果大于两个字节，则这个指令的结果无定义。

·DECA（A减值） C＝48； F＝1。

·DECX（X减值） C＝55； F＝1。

·DEC i （ i 减值） C＝48＋ i； F＝1。

这八条指令分别地和INCA，INCX及INC i 相同，只是从各寄存器减去数量M，而不是加上。

注意，对于ENTA，ENNA，INCA以及DECA其操作码C皆相同。在这种情况下，就用F场来区分各种操作。

比较操作符 所有的比较操作符都是将寄存器中的值同存储器中的一个值进行比较。然后，根据寄存器的值是小于，等于或大于存储单元的值，把比较指示器置为LESS，EQUAL或GREATER。负的0和正的0相等。

·CMPA（比较A） C＝56； F＝场。

所指定的A的场与CONTENTS(M)的同一个场进行比较。如果场F不包括符号位，

则两个场就都认为是正的；否则在比较时要把符号考虑在内。(如果F是(0:0)，则所进行的比较总是相等的，因为负0等于正0。)

·CMPX（比较X）C=63；F=场。

这类似于CMPA。

·CMPi（比较i）C=56+i；F=场。

类似于CMPA。在比较时，变址寄存器的字节1，2和3当作0来处理。

转移操作符 通常，指令是顺序地执行的，即是说，在执行了单元P中的一条指令后，接着应该执行的是在单元P+1中的指令。而若干"转移"指令允许来改变这一顺序。当发生这样的转移时，正规地，把J寄存器置成为下一条指令的地址（即是，如果没有转移，下一条将要执行的指令的地址）。因此，如果需要的话，程序员可以用"存储J"指令STJ来设置另一条指令的地址场，以便后面用来返回该程序中原先的位置。每当在一个程序中实际地出现了一个转移时（JSJ除外），J寄存器就改变；而当不出现转移时，它就决不改变。

·JMP（转移）C=39；F=0。

无条件转移：下一条指令从单元M中取出。

·JSJ（转移，保留J）C=39；F=1。

除了rJ内容不变外，和JMP相同。

·JOV（溢出时转移）C=39；F=2。

如果溢出开关接通，则把它断开并出现一个转移（JMP)；否则什么事情也不干。

·JNOV（不溢出时转移）C=39；F=3。

如果溢出开关断开，则出现一个转移（JMP)；否则将它断开。

·JL，JE，JG，JGE，JNE，JLE（小于，等于，大于，大于或等于，不等于，小于或等于时转移)。C=39；分别地F=4，5，6，7，8，9。

如果比较指示器已置成所述的条件，则进行转移。例如，如果比较指示器是LESS(小于）或GREATER(大于)，则JNE将进行转移。这些指令不改变比较指示器。

·JAN，JAZ，JAP，JANN，JANZ，JANP（A为负，0，正，非负，非0，非正时转移)。C=40；分别地F=0，1，2，3，4，5。

如果rA的内容满足所述的条件，则出现转移（JMP)；否则什么事情也不干。"正的"意味着大于0(非0)；"非正"意味着相反，即0或负的。

·JXN，JXZ，JXP，JXNN，JXNZ，JXNP（X为负，0，正，非负，非0，非正时转移)。C=47；分别地，F=0，1，2，3，4，5。

·JiN，JiZ，JiP，JiNN，JiNZ，JiNP（i为负，0，正，非负，非0，非正时转移)。C=40+i；分别地，F=0，1，2，3，4，5。

这些指令类似于对rA的相应操作。

其他操作符

·MOVE C=7；F=个数。

把由单元M开始的，按F所指定的个数的那些字，移动到由变址寄存器1的内容所指定的单元起的一片单元。一次传送一个字，而且在这个操作结束时，r11增加F的值。如

果F＝0，则不产生传送。

当所涉及的一组单元有重迭时，必须要小心。例如，假设 F＝3 和 M＝1000。如果(rI1)＝999，则我们把（1000）传送到（999），（1001）传送到（1000），而（1002）传送到（1001）。这里不出现什么不正常的情况。但如果(rI1)换成是 1001，则我们将把(1000)移到（1001），然后（1001）移到（1002），再后（1002）移到（1003），所以我们已经把同一个字（1000）移到三个位置。

·SLA，SRA，SLAX，SRAX，SLC，SRC（左移 A，右移 A，左移 AX，右移 AX，循环左移 AX，循环右移 AX）。C＝6，分别地，F＝0，1，2，3，4，5。

这些都是移位命令。寄存器A，X的符号在任何方式下都不受影响。M说明被左移或右移的字节的个数，M必须非负。SLA 和SRA 不影响 rX；而其余的移位都影响到两个寄存器，因为已把它们当作是一个统一的 10 字节的寄存器了。通过 SLA，SRA，SLAX 和 SRAX，0 被移入到寄存器的一边，而字节则消失于另一边。指令SLC和SRC称作是“循环”移位，在这些指令中，离开一端的字节又进入到另一端中去。rA 和 rX 全都参与到循环移位当中。

例

	寄存器 A					寄存器 X					
初始的内容	-	1	2	3	4	5	-	6	7	8	9	10
SRAX 1	+	0	1	2	3	4	-	5	6	7	8	9
SLA 2	+	2	3	4	0	0	-	5	6	7	8	9
SRC 4	+	6	7	8	9	2	-	3	4	0	0	5
SRA 2	+	0	0	6	7	8	-	3	4	0	0	5
SLC 501	+	0	6	7	8	3	-	4	0	0	5	0

·NOP（不操作）C＝0。

不出现操作，本指令即被通过。F和M皆予忽略。

·HLT（停机）C＝5；F＝2。

机器停机。当计算机操作员重新启动它时，其实际效果等价于 NOP。

输入输出操作符 MIX 有相当数量的输入输出设备（这些设备是以额外的价格任选的）。对每个设备赋予一个号码如下：

设备号	外围设备	区段的大小
t	磁带设备号为（$0 \leqslant t \leqslant 7$）	100字
d	设备号为 d 的($8 \leqslant d \leqslant 15$)磁盘或磁鼓	100字
16	卡片输入机	16字
17	卡片穿孔机	16字
18	打印机	24字
19	打字机和纸带机	14字

并非每个 MIX 装置都要有所有这些设备；我们将不时地作出关于存在某些设备的适当假定。某些设备不可以既用作输入又用作输出。上表中列出的字数是关于每一个设备的一个固定区段的大小。

磁带，磁盘或磁鼓设备之输入或输出是读或写整个的字(五个字节加符号)。然而，具有设备号 16 到 19 的设备之输入或输出则总以字符代码来进行工作，其中每个字节表示一个字符。于是，每个 MIX 字传输五个字符。表 1 的上部给出了字符代码。在本节的结束处和本书的末尾都有这个表。码 00 对应于“--”，它表示一个空格。码 01-29 对应字母 A 到 Z，其中还插进几个希腊字母；码 30-39 表示数字 0，1，…，9；而再往后的码 40，41，…表示标点符号和其它的特殊字符。不得输入或输出一个字节所可能有的全部的值，因为有某些组合是没有定义的。也未必所有的输入输出设备都有能力处理字符集合中的所有字符；例如，出现在字母当中的符号Φ和Π也许将不为卡片输入机所接受。当作字符代码的输入时，所有字的符号都置成“+”，而当输出时，则略去符号。

磁盘和磁鼓设备是各含 b^2 个 100 字区段的大型外部存储设备，其中 b 为字节大小。对如下所定义的每个 IN，OUT 或 IOC 指令，为指令所涉及的具体的 100 字区段，由 rX 的两个最低有效字节的当前内容来指定。

·IN（输入）C＝36；F＝设备。

这一指令启动所指定的输入设备向由M开始的连续的单元传送信息。传送的单元数等于这个设备的区段大小(见上表)。如果对于同一个设备，前边的操作还未完成，则机器将在这点上停候等待。由这条指令开始的信息传送，将一直延迟到稍后的某个时刻才完成。延迟的时间决定于输入设备的速度。所以直到这输入完成之前，一个程序必须不涉及存储器中的信息。还有，倘若企图从磁带上来读这样的任何区段，即这个区段是跟在此带上所写的最后的区段之后的，则这样的企图也是不适当的。

·OUT（输出）C＝37；F＝设备。

这条指令启动从M开始的存储单元到所指定的输出设备的信息传送（机器一直等到此设备已准备就绪，如果它不是一开始就已准备就绪的话)。这个传送也将一直延迟到稍后的某个时刻才完成。延迟的时间决定于输出设备的速度。所以直到这输出完成之前，一个程序必须不改变存储器中的信息。

·IOC（输入输出控制）C＝35；F＝设备。

如需要，机器一直等待，直到所指定的设备不占线为止。然后，依赖于正在使用的具体设备，而实施一个控制操作。下面的例子在本书的许多部分中都将用到：

磁带：如果M＝0，则磁带就反绕。如果M＜0，则磁带就向后跳过－M个记录，或者跳到这个带的开始处，就看那个情况首先出现。如果M＞0，则这个带就向前跳。向前跳到任何这样的一些区段（即该区段是跟在此带上最后所写的区段之后的区段）都是不适当的。

例如，序列“OUT 1000（3）；IOC －1（3）；IN 2000（3）”往带 3 上写（输出）一百个字，然后又再回过头来读（输入）这些字。除了对带的可靠性有疑问外不要这样用。这个序列的最后两条指令是把字 1000-1099 移动到 2000-2099 单元的一种慢速方式。但序列“OUT 1000（3）；IOC＋1（3）”则是不适当的。

磁盘或磁鼓　M 应当为 0。功能是按照 rX 来对该设备进行定位(到所指定的区段上)，以便在该设备上进行下条指令 IN 或 OUT 的操作就可少花些时间，如果这些操作使用这同一个 rX 设置时。

打印机　M应当为0。“IOC　0(18)”使打印机跳到下一页的顶头。

纸带输入机　反绕纸带（M应当为0）。

·JRED（就绪转移）C=38；F=设备。

如果所指定的设备是就绪的，即是说，前边的由IN，OUT或IOC启动的操作已经完成了，则出现转移。

·JBUS（占线转移）C=34；F=设备。

除了是在相反的情况下出现转移外，与JRED相同，即是说，当所指定的设备未就绪时转移。

例　在单元1000中的指令“JBUS 1000(16)”将反复地被执行，直到设备16就绪为止。

上述的简单的操作就是MIX的全部的输入输出指令。这里没有“带校验”指示器等等，来报告外部设备的异常情况。任何这样的情况（诸如，带塞住了，设备断开了，带过头了等等）都使得设备保持占线，响铃，并且，熟练的计算机操作员应按照通常的维护规程人工地进行处置。某些更复杂的外围设备，它比起这里所述的较为旧式的带，鼓和盘要更昂贵并更能代表当代的设备，我们将在5.4.6和5.4.9小节讨论之。

转换操作符

·NUM（转换成数值的）C=5；F=0。

这一操作用来把字符代码转换成数值码。M予以忽略。假定寄存器A，X含有一个10字节的字符代码的码数，则NUM指令置rA的数值等于这个码数的数值（处理成一个十进制数）。rX的值和rA的符号都不变。字节00，10，20，30，40，…转换成数字0；字节01，11，21，…转换成数字1，等等。可能有溢出，而且在这种情况下保留的是以字的大小为模所得的余数。

·CHAR（转换成字符）C=5；F=1。

这个操作用来把数值码转换成适合于在卡片上或打印机上进行输出的字符代码。rA中的值转换成一个10字节的十进制数，它以字符代码的形式放到寄存器A和X中。rA和rX的符号不变。M予以忽略。

例

	寄存器A					寄存器X					
初始内容	−	00	00	31	32	39	+	37	57	47	30	30
NUM 0	−					12977700	+	37	57	47	30	30
INCA 1	−					12977699	+	37	57	47	30	30
CHAR 0	−	30	30	31	32	39	+	37	37	36	39	39

计时　为了给出关于MIX程序是何等“好”的定量的信息，兹赋予每一个MIX的操作以一个对于今天的计算机说来是典型的执行时间。

ADD，SUB，所有的LOAD操作，所有的STORE（包括STZ）操作，所有的移位指令，以及所有的比较操作都花费两个单位时间。MOVE需要一个单位加上每移动一个字再花费两个单位的时间。MUL需要10个单位时间，DIV需要12个单位时间。对于浮

点操作的时间不予规定。所有其余的操作都花费一个单位时间，加上对于 IN，OUT，IOC 或 HLT 指令，计算机可能空闲的时间。

注意，特别地，ENTA 花费一个单位的时间，而 LDA 则花费两个单位的时间。计时的规则是容易记住的，因为有这样一个事实：时间的单位数就等于访问内存的次数（包括取指令本身的访问），除了对于移位，MUL 和 DIV 外。

时间的“单位”是一个相对的量度。我们将简单地用 u 来表示它。可以把它认为，比如说是 10 微秒（对于不太贵的机器）或者是 1 微秒（对于相对高价格的机器）。

例　序列 LDA 1000；INCA 1；STA 1000 恰巧花费 5 u 的时间。

总结　除了“GO 按钮”外，至此我们已经讨论了 MIX 的全部特性。“GO 按钮”在习题 26 中讨论。尽管 MIX 有近于 150 种不同的操作，但由于它们已分成了少数几种简单的类型，因此就便于记忆了。表 1 综述了对于每个C建立的操作。每个操作符的名称后边跟着的，是括在圆括号中的该操作符的正常的 F 场。

下列的习题用来快速地复习本小节的内容。大多数是很简单的，因此读者应该尝试把它们全都做一做。

习题

1.〔00〕 如果 MIX 是一个三进制(基数为 3)的计算机，则每个字节需要有多少“三进数”位？

2.〔02〕 如果在 MIX 内，待表示的一个值可以达到 99999999 这么大，则为包含这一数量应使用多少相邻的字节？

3.〔02〕 对于(a)地址场，(b)变址场，(c)场的场，以及(d)一条 MIX 指令的操作码的场，给出部分场的说明 (L:R)。

4.〔00〕 (5)中最后的例子是“LDA-2000, 4”——从存储器的地址不应是负的这一事实来看，这怎样才能合法呢？

5.〔10〕 对应于字(6)的[象在(4)中那样的]符号记号是什么？

▶6.〔10〕 假定 3000 单元包含

+	5	1	200	15

则下列指令的结果是什么（指出这些当中有没有哪些个是无定义的或仅仅是部分地定义的）：(a)LDAN 3000；(b)LD2N 3000(3:4)；(c)LDX 3000(1:3)；(d) LD6 3000；(e)LDXN 3000(0:0)？

7.〔15〕 利用代数运算 $X \bmod Y$ 和 $\lfloor X \rfloor$，对于不出现溢出的所有情况，给出 DIV 指令之结果的精确定义。

8.〔15〕 对于 105 页上所写的其“执行前的 rX”等于

+	1235	0	3	1

的关于 DIV 指令的最后的例子，如果“执行前的 rX”改成为

-	1234	0	3	1

而其它部分不变，则在执行了 DIV 指令之后，寄存器 A、X 将包含着什么？

▶9.〔15〕 列出可能影响溢出开关之状态的所有的 MIX 操作符（不包括浮点操作符）。

10.〔15〕 列出可能影响比较指示器之状态的所有的 MIX 操作符。

▶11.〔15〕 列出可能影响 rI1 之置位的所有的 MIX 操作符。

12.〔10〕 找出一条以 2 来乘 rI3 的当前内容并将此结果仍保留在 rI3 中的指令。

▶13.〔10〕 假设 1000 单元含有指令“JOV 1001”。如果溢出开关接通，则这条指令就使溢出开关断开（而且在任何情况下，下一条被执行的指令将是在单元 1001 中的）。如果这条指令改成为“JNOV 1001”，则将有什么不同？如果把它改成为“JOV 1000”或“JNOV 1000”，则又有什么变化？

14.〔20〕 对于每一个 MIX 操作符，考虑是否有一个方法来置指令的±AA 部分，I 部分，和 F 部分，使得这一指令的执行结果，除了执行时间可能更长些之外，完全等价于 NOP。假定对于任何寄存器或任何存储单元的内容什么也不知道，则每当可能产生一个 NOP 时，请指出它可以怎样做？例：如果地址和变址部分都为 0，则 INCA 就是一个 NOP。而 JMP 决不可能是 NOP，因为它要影响 rJ。

15.〔10〕 在一个打字机区段中有多少字符？在卡片输入机或穿孔机区段中呢？在打印机区段中呢？

16.〔20〕 写出把存储单元 0000-0099 全都清为 0 的程序，而且这一程序要：(a)尽可能短；(b)尽可能快〔提示：考虑使用 MOVE 指令〕。

17.〔26〕 和上题一样，所不同的是，要把 0000 到 N（包括 0000 和 N 在内）的单元全清为 0，其中 N 是 rI2 的当前内容。这个程序应当对于 0≤N≤2999 的任意值进行工作，它应当从单元 3000 开始。

▶18.〔22〕 当下列的“1 号”程序执行过后，寄存器、溢出开关以及存储器发生了什么变化？（例如，rI1 的最后的状态是怎样的？rX 呢？溢出开关和比较指示器呢？）

```
STZ     1
ENNX    1
STX     1(0:1)
SLAX    1
ENNA    1
INCX    1
ENT1    1
SRC     1
ADD     1
DEC1    -1
STZ     1
CMPA    1
MOVE    -1, 1(1)
NUM     1
CHAR    1
HLT     1
```

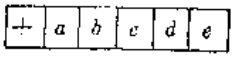
▶19.〔14〕 上题的程序的执行时间等于多少？不计 HLT 指令在内。

20.〔20〕 写出一个置所有的4000个存储单元都成为"HLT"指令而后停机的程序。

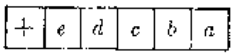
▶21.〔24〕 (a)J寄存器还可能为0吗?(b)写出一个程序，让它在r14中给出一个数N，置寄存器J等于N，假定$0<N\leqslant 3000$。你的程序应当从3000单元开始。当你的程序执行结束后，所有存储单元的内容必须不变。

▶22.〔28〕 设2000单元包含一整数X。试写出两个计算X^{18}并将结果存入寄存器A而后停机的程序。一个程序应当使用最少数量的MIX的存储单元；另一个程序则应当需要尽可能少的执行时间。假定X^{18}能放入单个字中。

23.〔27〕 0200单元含有一个字

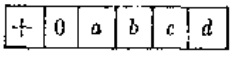


试写出计算"反射"字

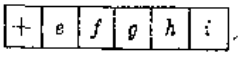


并把结果存入寄存器A中而后停机的两个程序。一个程序应当不使用MIX的"部分场"的特色来做。在所述的条件下，两个程序都应当用尽可能少的存储单元（包括程序所使用的以及为临时存储中间结果所使用的那些单元在内）。

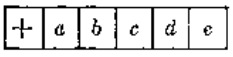
24.〔21〕 假定寄存器A和X分别含有



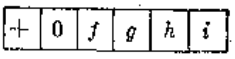
和



试写出把这两个寄存器的内容分别改变成为



和



的两个程序，使用(a)最少的存储单元和(b)最少的执行时间。

▶25.〔30〕 假设MIX的制造者希望出产一台更强有力的计算机("卡MIX"?)，而且他要说服现在拥有MIX计算机的人尽可能多地投资于这更昂贵的机器。他想把这新的硬件设计成MIX的在这种意义下的扩充，即是，凡MIX的程序，不必改变就都可以正确地在新的机器上工作。请提出想要的能添入这个扩充的东西（例如，你能更好地利用指令的I场吗?）

▶26.〔32〕 这个问题是编写一个装入卡片的程序。为了获得初始的进入计算机的信息和正确地启动等等，每台计算机都有各自的特殊问题。对于MIX说来，一张卡片的内容仅能以字符代码的形式读入，而且这包括含有装入程序本身的卡片在内。并不是所有可能的字节的值都能从卡片读入的，而且从卡片读入的每个字都是正的。

MIX还有一个我们在正文中未及说明的特性：这就是有一个"GO按钮"，从计算机存储单元含有任意信息的"杂乱"状态下，用它来启动计算机。当这按钮被计算机操作员揿下来的时候，即发生下列的动作：

a）一整张卡片读入单元0000–0015中；这实际上等价于指令"IN 0(16)"。

b）当此卡片已经完全读完，并且卡片输入机不再占线时，出现一个到0000单元的JMP。J寄存器也被置成0。

c）机器现在开始执行已经从此卡片读进来的程序。

（注意：那些没有卡片输入机的 MIX 计算机，都有它们隶属于纸带输入机设备 19 的 GO 按钮。但在本问题里，我们将假定存在着卡片输入机设备 16）。

有待编写的装入程序必须满足下列条件：

a）输入卡片叠由装入程序开始，接着是包含着待装入的数码的信息卡片，然后是一张"转移卡片"，它停止装入程序并转移到程序的开始。装入程序必须填到两张卡片上。你来设计出一张合适的转移卡片。

b）信息卡片有如下的格式：

1-5 列，装入程序忽略它。

6 列，在这张卡片上待装入的连续的字的个数（1 至 7 个）。

7-10 列，字 1 的单元，它总大于 100（所以它不复盖装入程序）。

11-20 列，字 1。

21-30 列，字 2（如果 6 列上的个数≥2）。

…

71-80 列，字 7（如果 6 列上的个数＝7）。

字 1，字 2，…的信息，都以十进数穿孔成数值的形式。如果这个字应该是负的，则负号（"11 穿孔"）就重迭穿孔到最低有效数字上，例如，在 20 列上。假定这使得字符代码的输入成为 10，11，12，…，19，而不是 30，31，32，…，39。例如，一张在 1-10 列穿有

ABCDE3100001234567890000000001000000010J

的卡片，就使得下列的信息被装入：

1000：+0123456789；1001：+0000000001；1002：−0000000100

c）装入程序应该对所有的字节大小都能进行工作，而无须对穿着装入程序的卡片作任何改变。卡片不应包含有对应于字节 20，21，49，50，…的任何字符（即是字符Φ，Π，$，<，…），因为这些字符不能为所有的卡片输入机所读入。特别是，不能使用指令 ENT1 和 INC1（C＝49），因为它们不能穿到卡片上。

1.3.2 MIX 汇编语言

利用符号语言，可以相当可观地简化 MIX 程序的阅读和编写工作，从而减轻程序员对繁琐的书写细节的烦恼。这些繁琐的细节，常常造成不应有的错误。这个语言 MIXAL（"MIX 汇编语言"（MIX Assembly Language）读作米克萨尔），是在前一小节中对指令所使用的记号的扩充。这个扩充的主要特点是：随意地使用字母名称来代表数，并随意地使用把名称同存储单元联结在一起的单元场。

如果我们先来考虑一个简单的例子，则 MIXAL 就很容易理解了。下列的代码是一个更大的程序的一小部分。它是一个子程序，是按照算法 1.2.10 M来求 n 个元素 $X[1]$，…，$X[n]$之极大值的。

程序 M（求极大值） 寄存器分配：rA≡m，rI1≡n，rI2≡j，rI3≡k；$X[i]$≡单元(X＋i)。

表 1

字符代码：

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
Δ	A	B	C	D	E	F	G	H	I	@	J	K	L	M	N	O	P	Q	R	Φ	Π	S	T	U	V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	·	,	(	)	+	−	*	/	$	=	空	;	'

00 1	01 2	02 2	03 10	04 12	05 1	06 2	07 1+2F
不操作 NOP(0)	rA←rA+V ADD(0:5) FADD(6)	rA←rA−V SUB(0:5) FSUB(6)	rAX←rAX*V MUL(0:5) FMUL(6)	rA←rAX/V rX←余数 DIV(0:5) FDIV(6)	特殊的 NUM(0) CHAR(1) HLT(2)	移位M个字节 SLA(0) SRA(1) SLAX(2) SRAX(3) SLC(4) SRC(5)	移动F个字 从M到rI MOVE(1)
08 2	09 2	10 2	11 2	12 2	13 2	14 2	15 2
rA←V LDA(0:5)	rI1←V LD1(0:5)	rI2←V LD2(0:5)	rI3←V LD3(0:5)	rI4←V LD4(0:5)	rI5←V LD5(0:5)	rI6←V LD6(0:5)	rX←V LDX(0:5)
16 2	17 2	18 2	19 2	20 2	21 2	22 2	23 2
rA←−V LDAN(0:5)	rI1←−V LD1N(0:5)	rI2←−V LD2N(0:5)	rI3←−V LD3N(0:5)	rI4←−V LD4N(0:5)	rI5←−V LD5N(0:5)	rI6←−V LD6N(0:5)	rX←−V LDXN(0:5)
24 2	25 2	26 2	27 2	28 2	29 2	30 2	31 2
F(M)←rA STA(0:5)	F(M)←rI1 ST1(0:5)	F(M)←rI2 ST2(0:5)	F(M)←rI3 ST3(0:5)	F(M)←rI4 ST4(0:5)	F(M)←rI5 ST5(0:5)	F(M)←rI6 ST6(0:5)	F(M)←rX STX(0:5)
32 2	33 2	34 1	35 1+T	36 1+T	37 1+T	38 1	39 1
F(M)←rJ STJ(0:2)	(FM)←0 STZ(0:5)	设备F占线? JBUS(0)	控制，设备F IOC(0)	输入，设备F IN(0)	输出，设备F OUT(0)	设备F就绪? JRED(0)	转移 JMP(0)JSJ(1) JOV(2)JNOV(3) 还有以下的[*]
40 1	41 1	42 1	43 1	44 1	45 1	46 1	47 1
rA:0，转移 JA[+]	rI1:0，转移 J1[+]	rI2:0，转移 J2[+]	rI3:0，转移 J3[+]	rI4:0，转移 J4[+]	rI5:0，转移 J5[+]	rI6:0，转移 J6[+]	rX:0，转移 JX[+]
48 1	49 1	50 1	51 1	52 1	53 1	54 1	55 1
rA←[rA]? ±M INCA(0)DECA(1) ENTA(2)ENNA(3)	rI1←[rI1]? ±M INC1(0)DEC1(1) ENT1(2)ENN1(3)	rI2←[rI2]? ±M INC2(0)DEC2(1) ENT2(2)ENN2(3)	rI3←[rI3]? ±M INC3(0)DEC3(1) ENT3(2)ENN3(3)	rI4←[rI4]? ±M INC4(0)DEC4(1) ENT4(2)ENN4(3)	rI5←[rI5]? ±M INC5(0)DEC5(1) ENT5(2)ENN5(3)	rI6←[rI6]? ±M INC6(0)DEC6(1) ENT6(2)ENN6(3)	rX←[rX]? ±M INCX(0)DECX(1) ENTX(2)ENNX(3)
56 2	57 2	58 2	59 2	60 2	61 2	62 2	63 2
rA(F):V→CI CMPA(0:5) FCMP(6)	rI1(F):V→CI CMP1(0:5)	rI2(F):V→CI CMP2(0:5)	rI3(F):V→CI CMP3(0:5)	rI4(F):V→CI CMP4(0:5)	rI5(F):V→CI CMP5(0:5)	rI6(F):V→CI CMP6(0:5)	rX(F):V→CI CMPX(0:5)

一般形式



C＝操作码，指令的(5:5)场
F＝OP的补充说明，指令的(4:4)场
M＝变址后的指令的地址部分
V＝F(M)＝单元M的F域的内容
(OP)＝操作的符号名称
(F)＝标准值F的设置
t＝执行时间；T＝互锁时间

rA＝寄存器A
rX＝寄存器X
rAX＝寄存器AX当作一个
rIi＝变址寄存器i，1≤i≤6
rJ＝寄存器J
CI＝比较指示器

[*]：
JL(4) ＜ N(0)
JE(5) ＝ Z(1)
JG(6) ＞ P(2)
JGE(7) ≥ NN(3)
JNE(8) ≠ NZ(4)
JLE(9) ≤ NP(5)

[+]：

	汇编的指令					行号	单元	操作符	地址	次数	注释
						01	X	EQU	1000		
						02		ORIG	3000		
3000:	+	3009	0	2	32	03	MAXIMUM	STJ	EXIT	1	子程序链接
3001:	+	0	1	2	51	04	INIT	ENT3	0,1	1	$M1$. 初始化. $k \leftarrow n$.
3002:	+	3005	0	0	39	05		JMP	CHANGEM	1	$j \leftarrow n$, $m \leftarrow X[n]$, $k \leftarrow n-1$.
3003:	+	1000	3	5	56	06	LOOP	CMPA	X,3	$n-1$	$M3$. 比较.
3004:	+	3007	0	7	39	07		JGE	*+3	$n-1$	
3005:	+	0	3	2	50	08	CHANGEM	ENT2	0,3	$A+1$	$M4$. 改变 m. $j \leftarrow k$.
3006:	+	1000	3	5	08	09		LDA	X,3	$A+1$	$m \leftarrow X[k]$
3007:	+	1	0	1	51	10		DEC3	1	n	$M5$. k 减 1.
3008:	-	3003	0	2	43	11		J3P	LOOP	n	$M2$. 全都试过了?
3009:	+	3009	0	0	39	12	EXIT	JMP	*	1	返回主程序. ■

这个程序是一个同时说明好几件事情的例子：

a）以“单元　操作符　地址”打头的列是最主要的。这三列包含了一个以 MIXAL 符号机器语言写成的程序，后边我们将要说明这个程序的细节。

b）以“汇编的指令”打头的列说明了对应于 MIXAL 程序的实际的数值机器语言。我们已经把 MIXAL 设计成使得任何 MIXAL 程序，都可以相当简单地翻译成数值的机器语言。这个过程可以由另一个称为汇编程序的计算机程序来实现。因此，程序员就可以专心于以 MIXAL 来进行他的“机器语言”的程序设计，而完全不必为确定等价的数值的机器语言本身所烦恼。实际上，本书中的所有 MIX 的程序都是以 MIXAL 语言写就的。第九章对一个汇编程序进行了完整的描述。这个汇编程序，把 MIXAL 程序翻译成为一种易于装入到 MIX 的内存的形式的机器语言。

c）以“行号”打头的列并非 MIXAL 程序的实质部分；这一列仅仅是为了编排本书的 MIXAL 程序，以便在正文中能引用本程序的各个部分。

d）以“注释”打头的列，对程序给出了说明，而且它同算法 1.2.10 M 的各个步骤互相参照。读者应该参照这个算法。注意在把该算法转写成一个 MIX 程序时，已经使用了一点“程序员的灵活性”，例如，步骤 $M2$ 已经放到后边去了。并请注意在程序 M 开始处所述的“寄存器分配”，它说明 MIX 的哪些成分与算法中的变量正相对应。

e）以“次数”打头的列，在本书的许多 MIX 程序中都将给出来。它表示在执行此程序的过程中，执行这一行上的指令的次数。于是，行 6 将被执行 $n-1$ 次，等等。由这一信息，我们就能确定为执行这个子程序所需要的时间长度；它等于 $(5+5n+3A)u$，其中 A 就是我们在 1.2.10 小节中仔细地进行过分析的量。

现在，我们就来讨论程序 M 的 MIXAL 部分。行 1，“X　EQU　1000”指出符号 X 等价于数 1000。在行 6 上可以看出它的作用；在那里，与指令“CMPA　X，3”等价的数值是

+	1000	3	5	56

即是，“CMPA 1000，3”。

行 2 指出，往后各行的存放单元应该顺序地选定，由 3000 单元开始。因此出现于行 3 的单元场上的符号 MAXIMUM 就等价于数 3000，而 INIT 等价于 3001，LOOP 等价于 3003，等等。

行 3 到行 12 上的操作符场包含了 MIX 指令的符号名称 STJ，ENT3，等等。另一方

而，行1和行2上的“操作符”场，却包含了不是MIX操作符的“EQU”和“ORIG”。它们被称做伪操作符，因为它们仅仅出现在MIXAL的符号程序中。伪操作符是用来说明一个符号程序的形式的，它们并不是这个程序的指令本身。因此“X EQU 1000”这一行只是谈及到程序，它并不表明当运行程序M时，有哪一个变量被置成等于1000。注意对于行1和行2，并没有汇编出指令来。

行3是一条“存储J”指令，它把寄存器J的内容存储到单元“EXIT”的（0：2）场，即是，存储到行12上的指令的地址部分。

如同早先所已指出的那样，预期程序M是一个更大的程序的一小部分。在别处，例如，序列

```
EXIT  100
JMP   MAXIMUM
STA   MAX
```

将转移到程序M且置 n 为100。然后程序M将找出元素 $X[1], \cdots, X[100]$ 之最大者，并在rA中存放此最大值和在r12中存放它的位置 j，而返回到指令“STA MAX”（参照习题3）。

行5把控制转移到行8。行4，5，6不必进一步说明了。行7引进了一个新的记号：一个星号（读作“自身”），指的是本行的单元。因此，“*+3”（“自身加3”）指的是当前这一行再往下数三个单元。由于行7是对应于单元3004的一条指令，所以“*+3”就对应于单元3007。

剩下的符号代码是自明的。注意在行12上又再次出现了星号（参照习题2）。

我们的下一个例子，能说明汇编语言的更稍多些的特性。这个例子的任务是打印出一张头500个质数的表，打印成10列，每列50个数。这个表应呈如下形式：

```
FIRST FIVE HUNDRED PRIMES
          0002 0233 0547 0877 1229 1597 1993 2371 2749 3187
          0003 0239 0557 0881 1231 1601 1997 2377 2753 3191
          0005 0241 0563 0883 1237 1607 1999 2381 2767 3203
          ⋮                                              ⋮
          0229 0541 0863 1223 1583 1987 2357 2741 3181 3571
```

我们将使用下边的方法。

算法P（打印500个质数的表） 本算法分为两个不同部分：步骤P1–P8准备500个质数的内部表，而步骤P9–P11是以上述形式来印出答案。程序的后边部分用了两个“缓冲”区，即是，在其中形成一个行的映象的两个存储段；在印出一个缓冲区的同时，就往另一个缓冲区送数。

P1.〔开始造表〕 置PRIME[1]←2，N←3，J←1。（N将跑遍所有奇数，奇数是质数的候选者；J用来记存至今已找到多少个质数）。

P2.〔N是质数〕 置J←J+1，PRIME[J]←N。

P3.〔找到500个了？〕 若J=500，则转P9。

P4.〔增加N〕 置N←N+2。

P5. 〔K←2〕置 K←2。（PRIME〔K〕将跑遍 N 的所有可能的质因子）。

P6. 〔PRIME〔K〕\N？〕以 PRIME〔K〕除 N；设Q是商，R是余数。若R = 0（因此N不是质数），则转P4。

P7. 〔PRIME〔K〕足够大了？〕若 Q≤PRIME〔K〕，则转P2（在这种情况下，N必然为质数；这一事实的证明是有趣的．而且有一点不寻常——见习题6）。

P8. 〔增加K〕K加1，并转P6。

P9. 〔打印标题〕现在我们准备来打印这个表。把打印机推进到下一页。置BUFFER〔0〕成为标题的行，并印刷此行。置B←1，M←1。

P10. 〔设置行〕以适当的格式将 PRIME〔M〕, PRIME〔50+M〕, …, PRIME〔150+M〕放置到缓冲区 BUFFER〔B〕。

P11. 〔打印行〕打印 BUFFER〔B〕；置B←1−B(借以转换到另一个缓冲区)；且M加1。若M≤50，则转P10；否则算法终止。■

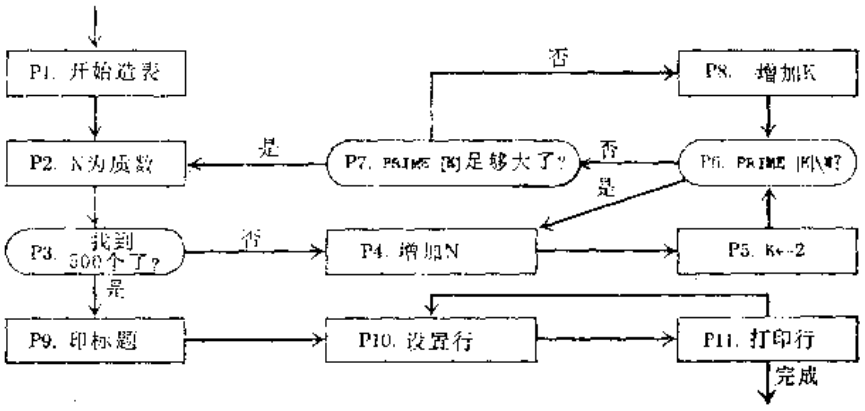


图14 算法P

程序 P（印 500 个质数的表） 这个程序是经过推敲的。我们有意识地写得稍微拙劣些，为的是想用一个完整的程序来说明 MIXAL 的大多数特性。rI1≡J−500；rI2≡N；rI3≡K；rI4 指示B；rI5 是M加 50 的倍数。

```
01    *EXAMPLE PROGRAM…TABLE OF PRIMES
02    *
03    L         EQU   500          要找的质数个数
04    PRINTER   EQU   18           打印机的设备号
05    PRIME     EQU   -1           质数表的存储区域
06    BUF0      EQU   2000         BUFFER〔0〕的存储区域
07    BUF1      EQU   BUF0+25       BUFFER〔1〕的存储区域
08              ORIG  3000
09    START     IOC   0(PRINTER)    跳到新的页
10              LD1   =1-L=        P1. 开始造表。J←1.
11              LD2   =3=          N←3
12    2H        INC1  1            P2. N是质数。J←J+1
13              ST2   PRIME+L,1    PRIME〔J〕←N
```

```
14                  J1Z     2F                  P3. 找到500个了?
15      4H          INC2    2                   P4. 增加N。
16                  ENT3    2                   P5. K←2。
17      6H          ENTA    0                   P6. PRIME[K]\N?
18                  ENTX    0
19                  DIV     PRIME, 3
20      JXZ         4B                          R = 0?
21                  CMPA    PRIME, 3            P7. PRIME[K]足够大了?
22                  INC3    1                   P8. 增加K。
23                  JG      6B                  若Q>PRIME[K]则转。
24                  JMP     2B                  否则N是质数。
25      2H          OUT     TITLE(PRINTER)      P9. 打印标题。
26                  ENT     BUF1+10             置B←1。
27                  ENT5    -50                 置M←0。
28      2H          INC5    L+1                 增加M。
29      4H          LDA     PRIME, 5            P10. 设置行。(自右到左)
30                  CHAR
31                  STX     0, 4(1:4)
32                  DEC4    1
33                  DEC5    50                  (rI5减少50，直到非正为止)
34                  J5P     4B
35                  OUT     0, 4(PRINTER)       P11. 打印行。
36                  LD4     24, 4               转换缓冲区。
37                  J5N     2B                  若rI5=0，则我们已完成。
38                  HLT
39  * INITIAL CONTENTS OF TABLES AND BUFFERS
40                  ORIG    PRIME+1
41                  CON     2                   头一个质数是2。
42                  ORIG    BUF0-5
43  TITLE           ALF     FIRST               标题行的字符信息
44                  ALF     FIVE
45                  ALF     HUND
46                  ALF     RED P
47                  ALF     RIMES
48                  ORIG    BUF0+24
49                  CON     BUF1+10             每一缓冲区参照另一个。
50                  ORIG    BUF1+24
51                  CON     BUF0+10
52                  END     START               程序结束。
```

下面是关于这一程序需要说明的值得注意的几点：

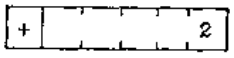
1. 行01，02和39以一个星号开始；这表示这些行都是“注释”行，仅起说明的作用，对汇编的程序没有实际的影响。

2. 如同在程序M中那样，行03中的“EQU”设置一个符号的等价值；在现在情况下，

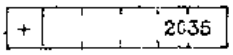
与L等价的值是500(在程序的行10-24中，L表示待计算的质数的个数)。注意在行05中，符号PRIME获得一个负的等价值。一个符号的等价值可以是任意五个字节再加一个正负号的数。在行07中，BUF1的等价值，按BUF0+25计算，就是2025。MIXAL对于数提供了有限数量的算术运算。另外的例子，可参看行13，其中PRIME+L的值（在现在情况下，它的值是499)是通过汇编程序计算的。

3．注意在行25和35的F部分，已经使用了符号PRINTER。总括在圆括号里边的F部分，完全和地址场的其它部分一样，可以是数值的或符号的。注意部分场说明的符号是以一个冒号来分隔的，如同在行31中的"1:4"那样。

4．MIXAL还有多种用来说明非指令字的方式。行41利用操作码CON，指出一个普通的常数"2"；行41的结果是汇编成字



行49说明一个稍微更复杂些的常数，"BUF1+10"，它汇编成字



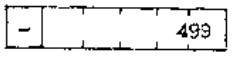
一个常数可以括在等号当中，而后它就变成一个文字常数（见行10和11)。汇编程序自动地为文字常数建立内部名称并插入"CON"行。例如，程序P的行10和行11实际上将被变成为

```
10      LD1   con1
11      LD2   con2
```

而后，在程序结尾处的行51和52之间，实际上被插入行

```
51a     con1   CON   1-L
51b     con2   CON   3
```

作为对文字常数进行汇编的步骤的一部分。行51a将汇编成字



使用文字常数无疑是很方便的，因为它意味着程序员不需要来对常数想一个名字，而且他不需要在程序结尾处插入该常数。他可以把自己的注意力集中在中心的问题上，而不必在编写程序的同时为这样的例行事务所烦恼。当然，在程序P中我们并没有特别好地利用文字常数，因为行10和行11要是写成"ENT1　1－L；ENT2　3"就更适当些！

5．一个好的汇编语言将模仿程序员考虑机器程序的途径，所以他应能流利地表达他自己。象我们刚刚提及的，这一指导思想的一个例子是文字常数的利用。另一个例子是"*"的利用，在程序M中我们已经对它作了说明。第三个例子是局部符号的思想，例如出现在行12，25和28的单元场中的符号2H。

局部符号是特殊的符号，它的等价值可以根据需要，重新定义任意多次。象PRIME这样的符号，在整个程序中自始至终仅有一种意义；而且如果它在多于一行的单元场中出现，则汇编程序就将指出一个错误来。但局部符号却有不同的性质。例如，我们在单元场写2H("这里2")，并在一MIXAL行的地址场写2F("前方2")或2B("后方2")；

2 B意味着最接近的以前的单元 2 H

2 F意味着最接近的随后的● 单元 2 H

例如，行 14 中的"2 F"是指的行 25；行 24 中的"2 B"是指后方的行 12；而行 37 中的"2 B"是指的行 28。2 F 或 2 B 的地址决不指同一行：例如，三行

```
2H   EQU    10
2H   MOVE   2F(2B)
2H   EQU    2B-3
```

实际上就等价于单一的行

```
MOVE *-3(10)
```

符号 2F，2B 决不能用于单元场中，且 2H 决不能用于地址场。总共有十个局部符号，这些符号是通过以由 0 到 9 的任何数字来代替上例中的"2"而得到的。

局部符号的思想是由梅·爱·康伟（M. E. Conway）于 1958 年，在关于尤尼瓦克 I（UNIVAC 1）的一个汇编程序中引进的。局部符号使得程序员在他要访问前后几行内的一条指令时，他不必考虑对这个地址取一个符号名称。当对于程序中附近的单元进行访问时，往往没有合适的具有多大意义的名称，所以程序员就倾向于使用象 X1，X2，X3 等等这样的符号，这就可能造成两次使用同一符号的危险。这就是为什么读者不久就会发现，当你写 MIXAL 程序时，局部符号的使用是自然地引出来的，如果你原来未曾熟悉这一思想的话。

6．在行 30 和 38 中，地址部分是空白。这意味着地址就是 0。同样地，我们也可以让行 17 中的地址也变成空白，但这样将使本程序变得不太易于阅读了。

7．行 43-47 使用了"ALF"操作，它以 MIX 的文字字符代码形式建立起一个五字节的常数。例如，行 45 将汇编成字

+	00	08	21	15	04

这些行用来存储标题行的头 25 个字符。所有的在 MIXAL 程序中未说明其内容的单元，一般都置成 0（但为装入程序所使用的单元除外，这些单元通常是 3700-3999）。因此没有必要把标题行的其它字都置成空白。

8．注意，在 ORIG 行上可以使用算术运算，例如在行 40，42 和 48 上那样。

9．一完整的 MIXAL 程序的最后一行的操作符总是 END，该行的地址是程序装入存储器时的开始地址。

10．作为对程序 P 的最后一点说明，读者可以来考察一下，我们是怎样来进行编码，以使得变址寄存器的计数最后变成 0 的；而且只要可能，就判断其是否为 0。例如，数量 J-500，而不是 J，存入 rI1 中。行 26-34 是特别值得注意的，虽然它也许显得微妙一些。

对程序 P 的实际运行来进行一下统计，可能是有兴趣的。行 19 中的除法指令执行了 9538 次，执行行 10-24 的时间是 182144 u。

● 当自前至后访问各单元时，按行进方向而言，以前的就是后方的，随后的就是前方的。——译者注

可以象图 15 中所示的那样，把 MIXAL 程序穿在卡片上。使用如下的格式：

列 1—10 单元场，

列 12—15 操作符场，

列 17—80 地址场和随意的注解，

列 11，16 空白。

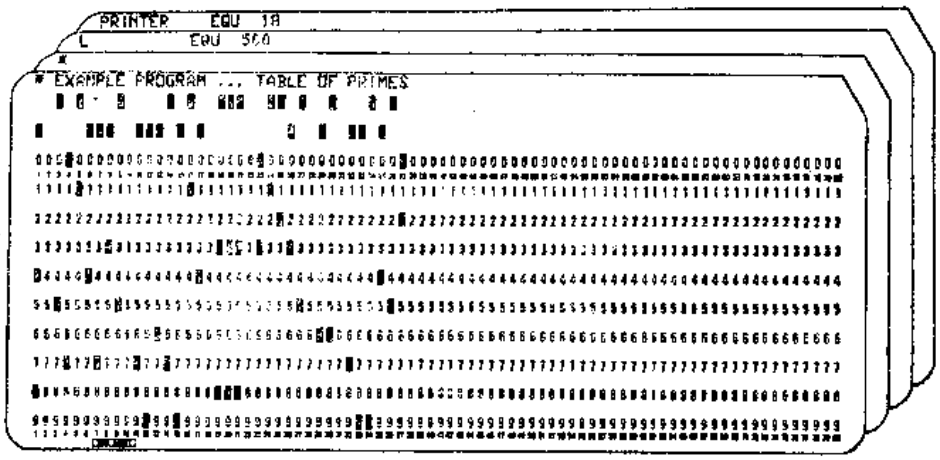


图15 穿在卡片上的程序P的头四行

然而，如果列 1 含有一个星号，则整张卡片就当作一个注释来处理。地址场以列 16 之后的头一个空白列结尾。任何说明信息可以穿在这头一个空白列的右边，它对汇编的程序无影响（例外情形：当操作符场是“ALF”时，则注释总是在列 22 处开始）。

MIX 汇编程序（见 9.3 节）接收以这种方式准备好的卡片叠，并且把它们转换成为可装入形式的机器语言程序。在顺利的情况下，读者将要接触 MIX 汇编程序和 MIX 模拟程序，并通过这些程序就能解决本书中给出的各种习题。

至此，我们已经看到了 MIXAL 能做些什么。在结束这一小节之前，我们还要更仔细地来说明一些规则，特别是我们将要说明：在 MIXAL 中哪些是不允许的。以下用相当少的几条规则，来定义这个语言。

1．一个符号是一个包含一到十个字母或数字的，而且至少必须含有一个字母的字符串。例：PRIME TEMP 20BY20。特殊符号 dH，dF，dB，其中 d 是一个数字，对于这一定义来说，将根据以前所述的“局部符号”的约定，而换成另外一些唯一的符号。

2．一个数是一个一到十位数字的串。例：00052。

3．一个符号在一个 MIXAL 程序中的每一次出现，或者称为是一个“定义了的符号”，或者称为是一个“未来的访问”。一个定义了的符号是出现在这个 MIXAL 程序的前边行的单元场中的符号。一个未来的访问是一个未以这样方式定义的符号。

4．一个原子表达式是

a）一个数，或者是

b）一个定义了的符号（表示与这符号等价的数值，见规则 13），或者是

c）一个星号（表示㊀的值；见规则 10 和 11）。

5．一个表达式是

a）一个原子表达式，或者是

b）一个正号或负号，后边接上一个原子表达式，或者是

c）一个表达式，后边跟随一个二元运算，后边再跟随一个原子表达式。

六个允许的二元操作是+，－，＊，／，//，：，它们对数值的 MIX 字定义如下：

```
C = A + B   LDA AA; ADD BB; STA CC
C = A - B   LDA AA; SUB BB; STA CC
C = A * B   LDA AA; MUL BB; STX CC
C = A / B   LDA AA; SRAX 5; DIV BB; STA CC
C = A // B  LDA AA; ENTX 0; DIV BB; STA CC
C = A : B   LDA AA; MUL =8=; SLAX 5; ADD BB; STA CC。
```

这里 AA，BB，CC 分别表示含有符号 A，B，C所代表的值的单元。一个表达式内的运算是自左至右地进行的。例：

－1＋5	等于 4
－1＋5＊20/6	等于 4＊20/6 等于 80/6 等于 13(从左到右地进行)
1//3	等于一个 MIX 字，其值近似于$(b^5/3)$，其中 b 是字节大小；即是，一个表示其小数点在左边的小数 $\frac{1}{3}$ 的字
1:3	等于 11(通常用于部分场的说明)
＊－3	等于㊀减 3
＊＊＊	等于㊀乘㊀

6．一个*A*部分（它用来描述一条 MIX 指令的地址场）是

a）空白（表示值 0），或者是

b）一个表达式，或者是

c）一个未来的访问（表示与这符号等价的最后的数值，见规则 13）。

7．一个变址部分（它用来描述一条 MIX 指令的变址场）是

a）空白（表示值 0），或者是

b）一个逗号，后边接上一个表达式（表示该表达式的值）。

8．一个*F*部分（它用来描述一条 MIX 指令的 F 场）是

a）空白（按照上下文，表示正常的 F 的设置），或者是

b）一个左圆括号，后边接上一个表达式，后边再接上一个右圆括号（表示这表达式的值）。

9．一个*W*值（它用来描述 MIX 的一个全字长的常数）是

a）一个表达式后边接上一个 F 部分（在这种情况下，一个空白的 F 部分表示(0:5)），或者是

b）一个*W*值，后边接上一个逗号，后边再接上一个形如(*a*)的*W*值。

一个W值表示如下确定的一个数值MIX字的值：设W值具有形式“$E_1(F_1)$，$E_2(F_2)$，…，$E_n(F_n)$”，其中 $n \geqslant 1$，诸E是表达式，诸F是场；则所求的结果是：执行了这样一个假设的程序“STZ CON；LDA C_1；STA CON(F_1)；…；LDA C_n；STA CON(F_n)。”之后，将出现于存储单元 CON 中的最后的值。这里 C_1，…，C_n 表示含有表达式 E_1，…，E_n 值的单元。每一个 F_i 必须有 $8L_i+R_i$ 的形式，其中 $0 \leqslant L_i \leqslant R_i \leqslant 5$。例：

1	是字
1,-1000(0:2)	是字
-1000(0:2),1	是字

+				1
-	1000			1
+				1

10. 汇编的过程利用由初始值为 0 的㊀（叫做单元计数器）表示的值。㊀的值将总是一个可填入两个字节的非负整数。当一行的单元场不是空白时，它就必须含有一个以前未曾定义的符号。然后，该符号的等价值将定义成㊀的当前值。

11. 当单元场经过象规则 10 中所述那样的处理之后，汇编的进程就取决于操作符 OP 场的值。对于 OP，共有六种可能性：

a）OP 是 MIX 的一个符号操作符（见七一节末尾的表 1）。这个图表定义了对于这个操作符的正常的C和F的值。在这种情况下，地址将是一个A部分（规则 6），后边接上一个变址部分（规则 7），后边再接上一个F部分（规则 8）。我们由此得到四个值：C，F，A和I；其作用是来把由序列 "LDA C; STA WORD; LDA F; STA WORD(4:4); LDA I; STA WORD(3:3); LDA A; STA WORD(0:2)" 确定的字汇编到由㊁确定的单元，并把㊀加 1。

b）OP 是 "EQU"。地址将是一个W值（见规则 9）；如果单元场不是空白，则出现在这里的符号的等价值就置成等于在地址中所说明的值。这个规则应比规则 10 优先，㊀的值不变（作为一个非平凡的例子，考虑行

```
BYTESIZE  EQU  1(4:4)
```

这个行允许程序员使用一个其值依赖于字节大小的符号。只要得到的程序对于每个可能的字节大小都有意义，则这就是一个可接受的情况）。

c）OP 是 "ORIG"。地址应是一个W值（见规则 9）；单元计数器㊀就置成这个值。注意：由于规则 10，出现在一张 ORIG 卡片❶ 的单元场上的一个符号，其等价值取为㊀的改变之前的值。例：

```
TABLE  ORIG  * + 100
```

是把 TABLE 的等价值置成 100 个单元的第一个。

d）OP 是 "CON"。地址将是一个W值；作用是把有这个值的一个字，汇编成由㊁指定的单元，并且对㊀加 1。

e）OP 是 "ALF"。作用是汇编由这张卡片的 17-21 列形成的字符代码的字，其它方面就象 CON 那样做。

f）OP 是 "END"。地址应是一个W值，这W值以其(4:5)场说明程序开始的指令单元。END 卡片通告一个 MIXAL 程序的结束。对应于所有未定义的符号和文字常数（见规则 12 和 13），汇编程序恰恰在 END 卡片的前面，实际上以任意次序插入相应的附加的行。于是，END 卡片的单元场上的一个符号，将表示紧接着插入的字之后的头一个单元。

12. 文字常数：9 个或更少个数的字符，这样的W值可以括在"="号之间并当作一个未来的访问。作用是想象成产生一个新的符号，并且恰恰插入到 END 卡片之前（见程序 P 后面的论述 4）。

---

❶ 一张卡片即一行。——译者注

13. 每个符号有一个且仅有一个等价的值；这是一个全字长的 MIX 的数，它或者根据规则 10 和规则 11(b)，依照符号在单元场中的出现而确定；否则，就有这样一行实际上插入到 END 卡片之前，即这一行的单元场上有着这个符号的名称，且 OP＝"CON" 而地址场＝"0"。

注意：以上规则的最重要的推论，是关于未来的访问之限制。一个未曾在以前的卡片的单元场中定义的符号，除非作为一条指令的 A 部分，否则不能使用。特别是，它不能(a)同算术运算一起使用；或者(b)在 EQU，ORIG 或 CON 的地址场中使用。例如，

LDA 2F + 1　　和　　CON 3F

都是不合法的。附加上这个限制，为的是使得对程序能进行更高效的汇编；而且，在这一套书的编写过程中所获得的经验已经证明，这是一个影响很小的温和的限制。

实际上，MIX 有两个汇编语言：MIXAL，面向机器的语言，我们设计它是为了便于通过一个相对地比较短的汇编程序来进行一遍扫描的翻译；以及 PL/MIX，它更适当地反映了数据和控制的结构，而且看起来很象是 MIXAL 程序的注释场。PL/MIX 将在第 9 章描述之。

习题——第一组

1. 〔00〕 正文中论述了，"X EQU 1000"并不表示是任何置一个变量之值的指令。假设你正在写一个 MIX 程序，其中你希望把含于某个存储单元（单元的符号名称是 X）的值置成等于 1000。你怎样以 MIXAL 写出这个来呢？

▶2. 〔10〕 程序 M 的行 12 写着"JMP *"。因为 * 表示这行的单元，那为什么程序不进入一个无穷的循环，即不终止地重复执行这一指令呢？

▶3. 〔23〕 如果以下程序同程序 M 一起使用，则执行以下程序的结果是什么？

```
START   IN      X + 1(0)
        JBUS    *(0)
        ENT1    100
1H      JMP     MAXIMUM
        LDX     X, 1
        STA     X, 1
        STX     X, 2
        DEC1    1
        J1P     1B
        OUT     X + 1(1)
        HLT
        END     START
```

▶4. 〔25〕 用手工来汇编程序 P，即是，对应于该符号程序，存储器的实际的数值内容是什么？

5. 〔11〕 为什么程序 P 不需要用一条 JBUS 指令来确定打印机已否准备就绪？

6. 〔*M*20〕 (a)证明：如果 n 不是质数，则 n 有一个满足 $1 < d \leqslant \sqrt{n}$ 的因子 d。

(b) 利用这一事实来证明：在算法 P 的步骤 P7 中的判断判明了 N 是质数。

7.〔10〕 在程序P的行34中"4B"的意义是什么？如果行15的单元改成"2H"，且行20的地址改成"2B"，则将产生什么影响——如果有影响的话？

▶8.〔24〕 下面的程序做什么？（不要在一台计算机上运行它，而是用手工来弄清楚！）

```
*MYSTERY   PROGRAM
PRINTER    EQU    18
BUF        ORIG   *+3000
1H         ENT1   1
           ENT2   0
           LDX    4F
2H         ENT3   0,1
3H         STZ    BUF,2
           INC2   1
           DEC3   1
           J3P    3B
           STX    BUF,2
           INC2   1
           INC1   1
           CMP1   =75=
           JL     2B
           ENN2   2400
           OUT    BUF+2400,2(PRINTER)
           INC2   24
           J2N    *-2
           HLT
4H         ALF    AAAAA
           END    1B
```

习题——第二组

这些习题是简短的程序设计问题，代表了典型的计算机应用并包括了广泛的技术。建议每个读者都能选择其中的少量程序，以便得到使用MIX的某些经验以及对基本的程序设计技巧进行完善的复习。如果需要的话，这些习题可以随着阅读第一章的剩下部分同时来做。

下面的表列出了所出现的程序设计方法的类型：

利用开关表（多路判断）：习题9和23。

利用变址寄存器：二维数组：习题10，21，22和23。

"散开的"字符：习题13和23。

整数的和"取整的"十进算术运算：习题14,16和18。

实时控制：习题20。

图形显示：习题23。

输入缓冲：习题13。

输出缓冲：习题 21 和 23。

使用子程序：习题 14 和 20。

每当本书中一个习题说："写出一个 MIX 程序"或者"写出一个 MIX 子程序"时，只要对所要求的写出符号代码就行了。这个程序将仅仅是一个更大的程序的一个小片断；也许在这个片断中没有进行输入或输出，等等。人们只需要来写出 MIXAL 行的单元场，操作符场和地址场（也可能还有注释，特别是如果有别的某人要对答案进行评阅的话！），而不必要写出数值的机器语言，行号或"次数"列，除非要求这样做。

如果习题说："写出一个完整的 MIX 程序"，则它意味着，要写出 MIXAL 的可执行的程序（特别是要包括最后的 END 卡片）；有希望的是，大多数读者都可采用汇编程序和 MIX 模拟程序，借以考验完整的程序。

▶9.〔25〕 单元 INST 含有一个意欲作为一条指令的 MIX 的字。试编制一段这样的程序：如果这个字有正确的 C 场，正确的 ±AA 场，正确的 I 场，以及正确的 F 场，则它就转移到单元 GOOD 去；否则就转移到单元 BAD 去。记住，对于 F 场的正确性测试，同 C 场有关。例如，如果 C = 7 (MOVE)，则任何 F 场都是可接受的；但如果 C = 8 (LDA)，则 F 场就必须有 8L + R 的形式，其中 $0 \leqslant L \leqslant R \leqslant 5$。除非 C 说明的是一条需要一个存储器地址的指令，"±AA" 场才算是正确的；否则 I = 0 而且 ±AA 不算是一个正确的存储器地址。

注意：没有经验的程序员往往想通过写出长长的一系列对 C 的测试来着手解决这一类的问题。例如，LDA C；JAZ 1F；DECA 5；JAN 2F；JAZ 3F；DECA 2；JAN 4F；等等。这不是一个好的做法！每当进行这样的多路判断时，最好是准备一张辅助的表，在这张表中填上为便于进行所需要的判断的信息。比如说，如果有 64 格的一张表，则我们可以写 "LD1 C；LD1 TABLE，1；JMP 0，1"——借此非常快地转移到所希望的程序。在这样一张表中还可以记上其它的信息。在这种情况下，造表的方法不过使程序变得稍微长些（包括这张表在内），但却大大地提高了速度。

▶10.〔31〕 假设我们有一个 9×8 的矩阵

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{18} \\ a_{21} & a_{22} & a_{23} & & a_{28} \\ \vdots & & & & \vdots \\ a_{91} & a_{92} & a_{93} & \cdots & a_{98} \end{pmatrix}$$

a_{ij} 存放到存储器的 $1000+8i+j$ 单元中。因此，在存储器中矩阵以如下形式出现：

$$\begin{pmatrix} (1009) & (1010) & (1011) & \cdots & (1016) \\ (1017) & (1018) & (1019) & & (1024) \\ \vdots & & & & \vdots \\ (1073) & (1074) & (1075) & & (1080) \end{pmatrix}$$

如果有某个位置是它所在的行当中的最小值和它所在的列当中的最大值，则说这个矩阵有一个"马鞍点"。用符号来表示，即是：如果

$$a_{ij} = \min_{1 \leqslant k \leqslant 3} a_{ik} = \max_{1 \leqslant k \leqslant 9} a_{kj}$$

则说 a_{ij} 是一个马鞍点。试编写一个 MIX 程序，这个程序计算出

a）一个马鞍点的单元，如果至少有一个马鞍点；

b）值 0，如果没有马鞍点；

并且在 rI 1 中存这个值而后停机。

11.〔*M*29〕 假设上题中的 72 个元素都是不同的，而且假设全部72! 个排列都是机会均等的，则矩阵中有一个马鞍点的概率是什么？如果我们换成假定矩阵中的元素都是 0 和 1，而且全部 2^{72} 个这样的矩阵都是机会均等的，则概率又是什么？

12.〔*M*47〕 "习题答案"对习题 10 给出了两个答案，并且还提示了第三个答案，但并不清楚所给出的答案中哪一个更好些。试用习题 11 的每一个假定，来分析这些算法，并判定那一个是更好的方法。

13.〔28〕 一位密码分析家需要对字母在某种密码中的出现频率进行计算。这个密码已经穿成纸带，而且以一个星号来表示其结束。试写出一个完整的程序，它把这条纸带输入进来，计算每个字符直到第一个星号为止的出现频率，然后以如下形式把结果打印出来：

```
A   0010257
B   0000179
D   0794301
```

等等，即每一行一个字符。既不要计算空格的个数，也不要打印计数为 0 的字符（例如上边的 C）。为了提高效率，实行输入缓冲，即是，当正在输入一个区段到一个存储区域的同时，你就可以从另一个存储区域来对字符进行计数。你可以假定在输入纸带上还有一个额外的区段（跟在包含着结束星号的区段之后）。

▶14.〔31〕 下边的算法，是意大利天文学家阿洛伊修斯·莉里留斯(Aloysius Lilius)和德国耶稣会数学家克里斯托弗·克拉维尤斯（Christopher Clavius）在 16 世纪末给出的。这算法被大多数西方教会用来计算确定 1582 年之后每一年耶稣复活节的日期。〔对于以前的一些年份，请参看 *CACM* 5 (1962)，209–210。为计算复活节的头一个系统的算法，是由法国阿基坦尼亚的维克托里尤斯（Victorius）所著《标准复活节》(*Canon paschalis*)给出的（公元 457 年）在中世纪期间，有许多迹象表明，在欧洲，算术的唯一重要的应用就在于计算复活节的日期；因此这样的算法在历史上是很有意义的。关于进一步的说明，请参看托·海·奥贝恩(T. H. O'Beirne)所著的《难题与悖论》(*Puzzles and Paradoxes*)(伦敦：牛津大学印刷厂，1965）的第十章〕。

算法 E（复活节的日期） 设 Y 是欲求复活节日期的年份。

E1.〔黄金数〕 置 $G \leftarrow (Y \bmod 19)+1$（$G$ 称作这一年份在 19 年"梅顿"周期下的"黄金数"）。

E2.〔世纪〕 置 $C \leftarrow \lfloor Y/100 \rfloor + 1$（当 Y 不为 100 的倍数时，C 就是世纪数；即是，1970 是在二十世纪中）。

E3.〔校正〕 置 $X \leftarrow \lfloor 3C/4 \rfloor - 12$，$Z \leftarrow \lfloor (8C+5)/25 \rfloor - 5$（$X$ 是这样的年数，例如 1900 年，即其中跳过了闰年，为的是同太阳的运行一致起来。Z 是用来使复活节同月亮的轨道同步的一个特殊的校正值）。

E4.〔求星期日〕 置 $D \leftarrow \lfloor 5Y/4 \rfloor - X - 10$〔三月$((-D) \bmod 7)$实际上将是一个星

期日]。

E5.〔岁首(阳历一年间先于阴历的日子)〕 置$E \leftarrow (11G+20+Z-X) \bmod 30$。如果$E=25$且黄金数$G$大于11，或者如果$E=24$，则$E$增1($E$是所谓的"岁首"，它说明何时出现满月)。

E6.〔求满月〕 置$N \leftarrow 44-E$。如果$N<21$，则置$N \leftarrow N+30$。(复活节就强行规定为"从三月二十一日起出现的头一个满月之后的第一个星期日。"实际上，月亮轨道的摄动使得这点并不严格地正确。但我们在这里关心的是"日历的月亮"而不是真正的月亮。三月N日是日历的一个满月)。

E7.〔进到星期日〕 置$N \leftarrow N+7-((D+N) \bmod 7)$。

E8.〔得出月份〕 若$N>31$，则日期是四月$(N-31)$日；否则日期是三月N日。■

假定年份小于100000。试写出计算和印出给定的年份的复活节日期的一个子程序(输出应有"*dd* MONTH, *yyyyy*"的形式，其中*dd*是日期，*yyyyy*是年份)。再写出一个完整的MIX程序，它利用这个子程序来编制从1950年到2000年的复活节日期表。

15.〔*M*30〕 在编写上题的程序的过程中，一个相当普遍的错误是没有认识到，步骤E5中的数量$(11G+20+Z-X)$可能是负的，而且因此有时算出来的不是正的余数 mod 30。〔见*CACM* 5 (1962), 556。〕例如，在14250年，我们将求得$G=1$，$X=95$，$Z=40$。所以如果我们有$E=-24$而不是$E=+6$，则我们就将得到滑稽的答案"四月37日"。试写出一个完整的程序，这个程序将找出这样的最早的年份，即在这个年份，这个错误造成了算出不正确的复活节的日期。

16.〔31〕 在1.2.7小节里我们证明了和数$1+\frac{1}{2}+\frac{1}{3}+\cdots$成为无穷大。但如果通过一台计算机以有限的精度来计算这个和数，则它在某种意义下实际上是存在的。这是因为后边的项终于变得如此之小，以至如果把它们逐一地加到和数上去，则它们对和数无所贡献。例如，假设我们以舍入到头一位小数来计算和数，则我们有$1+0.5+0.3+0.3+0.2+0.2+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1+0.1=3.9$。

更精确地说，设$r_n(x)$是数x舍入到第n位小数的值；我们定义$r_n(x)=\left\lfloor 10^n x+\frac{1}{2} \right\rfloor /10^n$。然后我们希望来求

$$S_n=r_n(1)+r_n\left(\frac{1}{2}\right)+r_n\left(\frac{1}{3}\right)+\cdots$$

我们已知$S_1=3.9$，问题是要写出一个完整的对于$n=2$，3，4和5来计算并印出S_n的程序。

注：实现这个求和，比之于一次加一个数$r_n\left(\frac{1}{m}\right)$，直到$r_n\left(\frac{1}{m}\right)$变成0这个简单的步骤来，有一个快得多的方法(例如，对于从66667到200000的所有的m值，我们都有$r_5\left(\frac{1}{m}\right)=0.00001$。这就是总共节省了133，334次来计算$1/m$的好想法！)倒是应当使用沿着如下的路线的算法：

A. 以$m_h=1$，$S=1$开始。

B．置 $m_e=m_h+1$ 并计算 $r_n(1/m_e)=r$。

C．求 m_h，即使得 $r_n(1/m)=r$ 的最大的 m。

D．把 $(m_h-m_e+1)r$ 加到 S 上并返回步骤B。

17．〔$M38$〕 利用上题的记号，证明或反驳

$$\lim_{n\to\infty}(S_{n+1}-S_n)=\ln 10$$

18．〔25〕 分母$\leqslant n$的0与1之间的所有既约分数的递升序列称作"n阶法雷(Faray)级数"。例如，7阶法雷级数是

$$\frac{0}{1},\ \frac{1}{7},\ \frac{1}{6},\ \frac{1}{5},\ \frac{1}{4},\ \frac{2}{7},\ \frac{1}{3},\ \frac{2}{5},\ \frac{3}{7},\ \frac{1}{2},\ \frac{4}{7},\ \frac{3}{5},$$
$$\frac{2}{3},\ \frac{5}{7},\ \frac{3}{4},\ \frac{4}{5},\ \frac{5}{6},\ \frac{6}{7},\ \frac{1}{1}$$

如果我们以 x_0/y_0，x_1/y_1，x_2/y_2，…来表示这个级数，则可证明

$$x_0=0\quad y_0=1\qquad x_1=1\quad y_1=n$$
$$x_{k+2}=\lfloor(y_k+n)/y_{k+1}\rfloor x_{k+1}-x_k \qquad (*)$$
$$y_{k+2}=\lfloor(y_k+n)/y_{k+1}\rfloor y_{k+1}-y_k$$

试通过把 x_k 和 y_k 的值分别地存入 $X+k$ 和 $Y+k$ 单元，写出一个计算 n 阶法雷级数的MIX子程序（这个级数的总项数近似地是 $3n^2/\pi^2$，所以你可以假设 n 是比较小的）。

19．〔$M30$〕（a）证明上题中由（*）定义的数 x_k，y_k 满足关系式 $x_{k+1}y_k-x_ky_{k+1}=1$。（b）利用（a）中证明的事实，来证由（*）给出的数 x_k/y_k 的确是 n 阶法雷级数。

▶20．〔33〕 假设X寄存器和MIX的溢出开关已经如下地连接到德尔马大宁和伯克利大路的交叉处的交通信号上：

rX(2:2)＝德尔马 交通灯
rX(3:3)＝伯克里 交通灯
} 0关闭，1绿色，2黄色，3红色

rX(4:4)＝德尔马 人行灯
rX(5:5)＝伯克里 人行灯
} 0关闭，1"行走"，2"不许行走"

希望穿过大宁沿着伯克利大路行进的汽车或行人必须触动一个开关，以使MIX的溢出开关接通。如果这个条件总不出现，则德尔马的灯应该始终保持绿色。

循环时间如下：

德尔马交通灯是绿色≥30秒，黄色8秒；

伯克利交通灯是绿色20秒，黄色5秒。

当对于一个方向交通灯是绿色或黄色时，另一个方向就开红灯。当交通灯是绿色时，对应的"行走"灯就接通；但在绿灯转为黄色之前，"不许行走"灯闪亮12秒钟如下：

"不许行走"灯 $\frac{1}{2}$秒
关闭 $\frac{1}{2}$秒
} 重复8次

"不许行走"灯 4秒（并且保持在黄灯和红灯的整个循环期间）。

如果正当伯克利灯是绿色时溢出开关被接通，则汽车或行人将在这期间里通过。但是

如果在黄色或红色灯亮时它被接通，则必须等到在德尔马的交通往来已经通过之后的下一次循环。

假设一个 MIX 时间单位等于 10 微秒。按照由溢出开关给出的输入，写出一个完整的通过操纵 rX 来控制这套交通灯的 MIX 程序。所述的时间应该精确地加以遵守，除非已不可能这样做。注意：rX 的状态精确地在一条 LDX 或 INCX 指令完成时改变。还要注意，不必为这个问题在经济上的不可行性而担忧。

21.〔28〕 一个 n 阶纵横图就是在一个正方的阵列中对于数 1 到 n^2 的一种安排，使得每行的和，每列的和，以及两个主对角线的和都相等。图 16 示出了一个 7 阶的纵横图。生成纵横图的规则是容易看出来的：从顶上一行中间的 1 开始，然后走到顶头并向左转沿对角线上行（当走到边缘上时，就把整个平面想象成一个卷筒，而把两边缘上的方格邻接起来）直至遇到已填的方格为止；然后由最近新填的方格退下一个方格，继续进行。每当 n 是奇数时，就可以用这个方法。

28	19	10	01	48	39	30
29	27	18	09	07	47	38
37	35	26	17	08	06	46
45	36	34	25	16	14	05
04	44	42	33	24	15	13
12	03	43	41	32	23	21
20	11	02	49	40	31	22

图16 纵横图

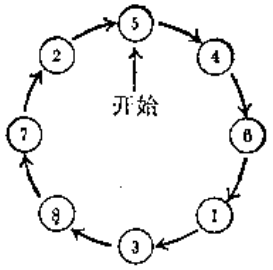


图17 约瑟夫斯问题，$n=8$，$m=4$

利用以类似于习题 10 的方式分配的存储，写出一个完整的通过上述方法来生成 23×23纵横图的 MIX 程序；然后印出这个纵横图。上述算法是由西·德·拉·劳伯利(S. de La Loubère) 于 1687 年从泰国传到法国的。对于其它各种有趣的纵横图的构造，有许多是程序设计的好习题，请参看沃·威·劳斯·鲍尔（W.W.Rouse Ball）著的《数学游戏和小品》；经哈·斯·麦·考克斯特校订（纽约：麦克米伦，1962），第 7 章〕。

22.〔31〕（约瑟夫斯（*Josephus*）问题）有 n 个人排列在一个圆周上。从一个特殊位置开始，我们沿着这个圆计数并且无情地处决每第 m 个人（即每隔 $m-1$ 个人处决一个，这个圆则随着这些人之被斩首而逐渐收缩）。例如，当 $n=8$，$m=4$ 时，执行的次序是 54613872，如图 17 所示：头一个人第 5 号被处决，第二人是第 4 号被处决，等等。试写出一个完整的 MIX 程序，它印出 $n=24$，$m=11$ 时的执行次序。试设计一个当 n 和 m 很大时以高速进行工作的灵巧的算法（它可能拯救你的生命）。参考文献：威·阿伦斯（W. Ahrens）《数学娱乐和游戏》（*Mathematische Unterhaltungen und Spiele*）2（莱比锡：托伊布纳，1918），第 15 章。

23.〔37〕 这道习题旨在提供计算机许多这样的应用的一些经验，即对于这些应用，其输出是以图形来加以显示的，而不是以通常的表格形式。在现在情况下，目标是来“绘画”一个纵横字谜的图形。

给你一个 0 和 1 的矩阵作为输入。0 的位置表示一个白的方格，1 表示一个黑的方

格。输出应该是一个纵横字谜的图形，在这图形的方格上，带有对于字进行“纵横”和“上下”的适当的编号。

例如，给定矩阵

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

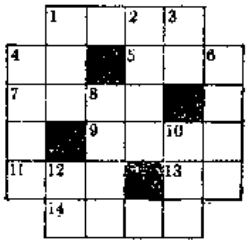


图18　对应于习题23中的矩阵的图形

则对应的字谜图形将如图 18 所示。如果一个方格是白的以及或者（a）它下边的方格是白的且在紧挨它的上边没有白方格，或者（b）在紧挨它的左边没有白方格而且它右边的方格是白的，则对它进行编号。如果在边上有黑的方格，则应该从图形中去掉它们。图18中说明了这一点，其中已去掉角上的黑方格。实现这一点的简单方法是：在给定的输入矩阵的顶部、底部以及两边上人为地插入－1的行和列，然后把每一个与“－1”相邻的“+1”改变成为“－1”，直到在任何“－1”的紧邻处没有“+1”为止。

以下的方法将用来打印最后的图形：字谜的每格应当对应于输出页上的5列和3行。这15个位置应如下填写：

未编号的白方格：

```
bbbbb+
bbbbb+
+++++
```

编号 n n 的白方格：

```
nnbbb+
bbbbb+
+++++
```

黑方格：

```
+++++
+++++
+++++
```

“－1”方格，取决于是否有－1在右边或下边而为：

```
bbbb+   bbbb+   bbbbb   bbbbb   bbbbb
bbbb+   bbbb+   bbbbb   bbbbb   bbbbb
++++    bbbb+   +++++   bbbb+   bbbbb
```

于是，图18中所示之图形将打印成如图19所示。

打印机的行宽——120个字符——足以印出23列的纵横字谜。提供作输入的数据将是由0和1组成的23×23矩阵，每行穿孔到一张输入卡片的1～23列中。在上述例子中，头一张卡片将穿孔成“10000 11111 11111 11111 111”。图形不一定是对称的，而且它可能以离奇的方式把黑方格连接到外边去而形成长长的通路。

1.3.3 对排列的应用

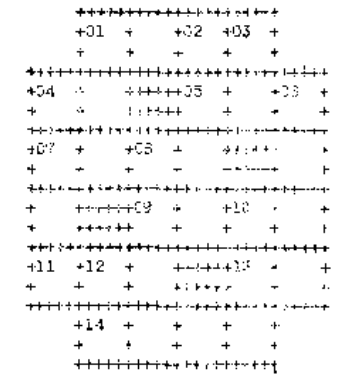


图19 打印机输出的前18行表示

在这一小节里，我们将给出若干更复杂的 MIX 程序，同时将介绍排列的某些重要的性质。这些研究也将阐明一般计算机程序设计的某些有趣的方面。

早在 1.2.5 小节，我们就已经讨论过排列。我们把排列 $cdfbea$ 处理作六个对象 a，b，c，d，e，f 在一直线上的一种安排。还可以有另外一种观点：我们可以把一个排列想象作一个重新安排或对象的更名●。对于这个解释，习惯上使用两行的记号，例如

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix} \tag{1}$$

表示“a 变成 c，b 变成 d，c 变成 f，d 变成 b，e 变成 e，f 变成 a”。考虑作一个重新排列，这意味着对象 c 移到原来由 a 占据的位置；考虑作一个更名，这意味着对象 a 更名为 c。两行的表示不受列的次序改变的影响，即是，排列（1）也可以写成

$$\begin{pmatrix} c & d & f & b & a & e \\ f & b & a & d & c & e \end{pmatrix}$$

以及其它的 718 种方式。

同这种解释相联系，通常使用一个循环记号。排列（1）可以写成

$$(a\ c\ f)(b\ d) \tag{2}$$

这还是意味着“a 变成 c，c 变成 f，f 变成 a，b 变成 d，d 变成 b”。循环 $(x_1x_2\cdots x_n)$ 意味着“x_1 变成 x_2，x_2 变成 x_3，…，x_{n-1} 变成 x_n，x_n 变成 x_1”。由于在这个排列之下，e 是“固定的”，所以在循环记号中它不出现。即是，象“(e)”这样的单一元素的循环，习惯上不写出来。如果一个排列固定所有的元素，以致仅有单一元素的循环出现，则它就叫做恒等排列，而且根据实际上并非太充分的理由，习惯上就以“(1)”来表示它。

循环的记号并不是唯一的。例如，

$$(b\ d)(a\ c\ f),\quad (c\ f\ a)(b\ d),\quad (d\ b)(f\ a\ c) \tag{3}$$

等等，都等价于（2）。然而，“$(a\ f\ c)(b\ d)$”则不是同一个循环，因为它说的是 a 变成 f。

容易看出，为什么循环记法总是可能的。由任何元素 x_1 开始，比如说，这个排列把 x_1 变成 x_2，x_2 变成 x_3，等等，直到最后（因为仅仅有有限多个元素），我们就得到某个已经在 $x_1,\cdots,x_n$ 当中出现的元素 x_{n+1}。现在，x_{n+1} 必须等于 x_1。因为，如果它等于，比方说 x_3，而我们已经知道 x_2 变成 x_3，而且由假定 $x_n\neq x_2$ 也变成 x_{n+1}。总之我们就有循环 $(x_1x_2\cdots x_n)$，$n\geqslant 1$，作为我们的排列的一部分。如果这还没有把整个排列包括在内，则我们就找另一个元素 y_1，而且以同样的方法求得另一个循环 $(y_1,\ y_2,\cdots,y_m)$。这些 y 中没有一个能等于任何一个 x。因为，$x_i=y_j$ 就意味着 $x_{i+1}=y_{j+1}$，等等，而且我们最终地将对

● 即置换。——译者注

某个 k，求出 $x_k=y_1$，这就和 y_1 的选择相矛盾。以这种方法，终于将求出所有的循环。

每当要重新排列有 n 个对象的某个集合时，就会提出把这些概念应用到程序设计中去的问题。为了不用辅助存储来重新安排这些对象，实际上我们就必须采用循环的结构。例如，为了进行重新安排（1），即是说，为了置

$$(a,b,c,d,e,f)\leftarrow(c,d,f,b,e,a)$$

实际上我们将采用循环的结构（2），并且逐次地置

$$t\leftarrow a,\ a\leftarrow c,\ c\leftarrow f,\ f\leftarrow t;\quad t\leftarrow b,\ b\leftarrow d,\ d\leftarrow t$$

任何这样的变换都可以通过若干类似的不相交的循环来进行，认识到这一点往往是很有用的。

排列的乘积● 我们可以把两个排列“乘”在一起。这里，乘法是这样来理解的，即在执行一个排列之后再执行另一个排列。例如，如果紧接着排列（1）之后，有排列

$$\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix}$$

则我们有 a 变成 c，然后 c 还变成 c；b 变成 d，然后 d 变成 a，等等：

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}\times\begin{pmatrix} a & b & c & d & e & f \\ b & d & c & a & f & e \end{pmatrix}=\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}\times\begin{pmatrix} c & d & f & b & e & a \\ c & a & e & d & f & b \end{pmatrix}$$

$$=\begin{pmatrix} a & b & c & d & e & f \\ c & a & e & d & f & b \end{pmatrix}\qquad(4)$$

很显然，排列的乘法是不可交换的，即是说，当 π_1 和 π_2 是排列时，则 $\pi_1\times\pi_2$ 未必等于 $\pi_2\times\pi_1$。读者可以验证，当两个因子交换时，则（4）中的乘积就给出不同的结果（见习题 3）。

有些人自右到左地来进行排列的乘法，而不是如（4）中所示的自左到右的比较更自然的顺序。事实上，在这点上，数学家分成了两派。应用变换 T_1，然后应用 T_2 的结果，应该用 T_1T_2 表示，还是应该用 T_2T_1？这里我们用 T_1T_2。

利用循环记号，则等式（4）应写成如下：

$$(a\ c\ f)(b\ d)(a\ b\ d)(e\ f)=(a\ c\ e\ f\ b)\qquad(5)$$

注意乘法的符号“×”习惯上常予省略。这同循环的记法并不矛盾，因为容易看出，排列 $(a\ c\ f)(b\ d)$ 实际上是排列 $(a\ c\ f)$ 和 $(b\ d)$ 的乘积。

排列的乘法可以直接借助于循环记号来进行。例如，为计算若干排列的乘积

$$(a\ c\ f\ g)(b\ c\ d)(a\ e\ d)(f\ a\ d\ e)(b\ g\ f\ a\ e)\qquad(6)$$

我们发现（自左到右地进行）“a 变成 c，然后 c 变成 d，而后 d 变成 a，其后 a 变成 d，后来 d 不变”，所以在（6）之下的实际结果是 a 变成 d。因此我们写下“$(a\ d$”作为部分的答案。现在我们考虑对 d 的作用：“d 变成 b 变成 g”，因而我们有了部分的结果“$(a\ d\ g$”。现在考虑 g，我们发现“g 变成 a，变成 e，变成 f，变成 a”，因此头一个循环就结束了，“$(a\ d\ g)$”。现在挑一个至今未出现的新元素，比如说 c，我们发现 c 变成 e，而且读者

● 即置换的乘积。——译者注

可以验证，对于（6）最终地得到答案“$(a\ d\ g)(c\ e\ b)$”。

现在让我们试验用计算机来进行这一过程。下边的算法把在上一小段所描述的方法，以机器计算所能遵循的方式加以形式化。

算法 A（循环形式的排列的乘法） 这一算法接受如象（6）式这样的循环的乘积，并以不相交循环的乘积之形式计算出结果的排列。为简便起见，单个元素的循环之删去在这里不予描述；这不过是本算法的一个相当简单的扩充。随着这一算法的实施，我们逐次地“标记”输入公式的元素，即是，对输入公式的已经处理的那些符号给以某种标记。

A1.〔第一遍〕 对所有左圆括弧加标记，并且以加标记复写该元素替换所有的右圆括弧紧跟着使之与左圆括弧相对。（见表1中的例子）。

A2.〔开括弧〕 从左到右进行检索，找头一个未曾标记的输入元素（如果所有元素都已加了标记，则这个算法就结束）。置 START 等于它；输出一个左圆括弧；输出此元素；并标记此元素。

表1 算法A应用于(6)

“X”表示一个加了标记的元素

经本步骤行	START	CURRENT	(a (c f g a) (b c d b) (a e d a) (f a d e f) (h g f a e h)	输出
A1			X X X

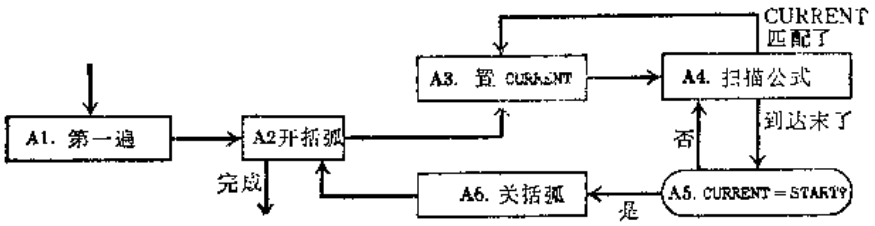


图20 进行排列乘法的算法A

MIX 程序 为了用 MIX 来实现这个算法，可以通过使用一个字的符号位来进行标记。假设我们的输入卡片是以如下的格式穿孔：80列的卡片分成为 16 个场，每个场 5 个字符，每个场或者是(a)"␣␣␣(␣"，表示开始一个循环的左括弧；或者是（b)")␣␣␣␣"，表示结束一个循环的右括弧；或者是（c)"␣␣␣␣␣"，全空白，它可以插在任何地方来填空；或者是（d）任何别的字符，表示被排列的一个元素。通过 76～80 列等于"␣(␣␣)␣="来识别输入的最后一张卡片。例如，（b）可以穿孔到两张卡片上如下：

```
( A C F G )    ( B C D )    ( A E D )
( F A D E )    ( B G F A E )                =
```

我们程序的输出将这样来组成，即：一个输入的复本，后边紧接着的是同样格式的答案。

程序 A（循环形式的排列的乘法） 这一程序实现算法 A，而且它还包括输入、输出部分，以及删去单个元素的循环的部分。

```
01    CARDS    EQU    16                    卡片输入机设备号
02    PRINTER  EQU    18                    打印机设备号
03    ANS      ORIG   *+1000                答案的位置
04    OUTBUF   ORIG   *+24                  用于复写输入
05    PERM     ORIG   *+1000                输入排列
06    BEGIN    IN     PERM(CARDS)           读头一张卡片
07             ENT2   0
08             LDA    EQUALS
09    1H       JBUS   *(CARDS)              等待循环完成
10             CMPA   PERM+15,2
11             JE     *+2                   是最后一张卡片?
12             IN     PERM+16,2(CARDS)      否，读另一张。
13             ENT1   OUTBUF
14             JBUS   *(PRINTER)            打印输入卡
15             MOVE   PREM,2(16)
16             OUT    OUTBUF(PRINTER)
17             INC2   16
18             JNE    1B                    重复直到输入完成
19    *                                     在这里，(rI2)个输入字是
20             DEC2   1              1              在 PERM，PERM+1，…
21             ST2    SIZE           1
22             ENT3   0              1      A1.第一遍。
```

```
23    2H      LDAN   PERM,3          A   取输入的下一个元素
24            CMPA   LPREN(1:5)      A   是“(”?
25            JNE    1F              A
26            STA    PERM,3          B   给它加标记
27            INC3   1               B   置下一个非空白的元素
28            LDXN   PERM,3          B   于rX中
29            JXZ    *-2             B
30    1H      CMPA   RPREN(1:5)      C
31            JNE    *+2             C
32            STX    PERM,3          D   以已加标记的rX代替“)”
33            INC3   1               C
34            CMP3   SIZE            C   所有元素都处理过了?
35            JL     2B              C
36    *
37            LDA    LPREN           1   为主程序作准备
38            ENT1   ANS             1   rI1=存后面答案的位置
39    OPEN    ENT3   0               E   A2.开括弧
40    1H      LDXN   PERM,3          F   寻找未加标记的元素
41            JXN    GO              F
42            INC3   1               G
43            CMP3   SIZE            G
44            JL     1B              G
45    *                                  全都加标记了，现在该输出了
46    DONE    CMP1   =ANS=               答案是恒等排列?
47            JNE    *+2                 若是，变成“(1)”
48            MOVE   LPREN(3)
49            MOVE   =0=                 在答案之后放置23个空白字
50            MOVE   -1,1(22)
51            ENT3   0
52            OUT    ANS,3(PRINTER)
53            INC3   24
54            LDX    ANS,3               按需要来打印诸行
55            JXNZ   *-3
56            HLT
57    LPREN   ALF    (                   程序中使用的常数
58            ALF    I
59    RPREN   ALF    )
60    EQUALS  ALF    =
61    *
62    GO      MOVE   LPREN           H   为输出中的一个循环开括弧
63            MOVE   PERM,3          H
64            STX    START           H
65    SUCC    STX    PERM,3          J   标记一个元素
66            INC3   1               J   向右移一步
67            LDXN   PERM,3(1:5)     J   A3.置CURRENT(即rX)
68            JXN    1F              J   跳过空格
69            JMP    *-3             0
70    4H      CMPX   PERM,3(1:5)     K   A4.扫描公式
71            JE     SUCC            K   元素=CURRENT?
72    1H      INC3   1               L   向右移
73            CMP3   SIZE            L   公式末了?
74            JL     4B              L
```

```
75              CMPX START(1:5)          P   A5. CURRENT = START?
76              JE    CLOSE              P
77              STX   0,1                Q   否，输出 CURRENT
78              INC1  1                  Q
79              ENT3  0                  Q   再次扫描公式
80              JMP   4B                 Q   返回 A4
81    CLOSE     MOVE  RPREN              R   A6. 关括弧
82              CMPA  -3,1               R   注意：rA = "("
83              JNE   OPEN               R
84              INC1  -3                 S   删去单个元素的循环
85              JMP   OPEN               S
86              END   BEGIN
```

这一约有 70 条指令的程序比之于前一小节的程序要长些，而且事实上它比我们在本书中将要遇到的大多数程序都长。然而，它的长度并不可怕，因为它分成相当独立的若干个小的部分。06～18行读进输入卡片并印出每张卡片的复本；行 20～35 实现算法的步骤 A1——输入的预处理；行 37～44 和行 62～85 完成算法 A 的主要工作；而行 46～55 输出答案。读者将会感到，尽其所能地研究本书中尽量多的 MIX 程序，将是有教益的——阅读其他人的计算机程序，这对于获得技巧有着重要的作用，而且这样一种训练在许许多多计算机教程中竟被忽视了，它已使计算机的使用效率低的可怜。

计时 程序A中不涉及输入输出的部分，已经给出了“计时”指示（参照程序1.2.10M）。例如，假设行 27 执行B次。为方便起见，已经假定，除了在最右端外，在输入中不出现空白字；这样，行 69 根本不执行，而且行 29 中的转移也根本不出现。

通过简单的加法，得出程序执行的总时间是

$$(7+5A+6B+7C+2D+E+3F+4G+8H+6J+3K+4L+3P+5Q+6R+2S)u \qquad (7)$$

加上输入输出的时间。为了理解公式（7）的意义，我们需要考虑 15 个未知数 $A, B, C, D, E, F, G, H, J, K, L, P, Q, R, S$，而且我们必须把它们与关于输入的有关特征联系起来。现在我们来说明为处理这一类型问题的一般原则。

首先，我们应用电路理论的克希霍夫（Kirchhoff）第一定律：一个指令被执行的次数必须等于我们转移到该指令的次数。这个看来显然的规则，往往以一种不明显的方式，与若干数量有关。分析程序A的流程，我们得到以下的方程：

从行	我们导出
23, 35	$A=1+(C-1)$
30, 25	$C=B-(A-B)$
39, 83, 85	$E=1+R$
40, 44	$F=E\div(G-1)$
62, 41	$H=F-G$
65, 68, 71	$J=H+(K-(L-J))$
70, 74, 80	$K=(L-P)\div Q$
81, 76	$R=P-Q$

通常，由克希霍夫定律给出的方程并非全都是独立的。在上述情况下，第一个和第二

个方程明显地是等价的。其次，最后两个方程也是等价的，因为由第三，第四和第五方程推出，$H=R$；因此，第六个方程说明$K=L-R$。总而言之，我们已经消去了15个未知数中的6个：

$$\begin{array}{lll} A=C & E=R+1 & F=R+G \\ H=R & K=L-R & Q=P-R \end{array} \tag{8}$$

克希霍夫第一定律是一个有效的工具，在2.3.4.1小节里我们将更严密地来分析它。

下一步是试图以数据的重要特征来匹配这些变量。我们由行21，22，27和33发现

$$B+C=\text{输入字的个数}=16X-1 \tag{9}$$

其中X是输入的卡片数。由行25，

$$B=\text{输入中“(”的个数}=\text{输入中循环的个数。} \tag{10}$$

类似地，由行31，

$$D=\text{输入中“)”的个数}=\text{输入中循环的个数。} \tag{11}$$

现在，(10)和(11)给出了未能由克希霍夫定律导出的事实：

$$B=D \tag{12}$$

由行62，

$$H=\text{输出中循环（包括单个元素的循环）的个数} \tag{13}$$

行81说明R等于这同一个量。$H=R$这一事实，在现在情况下是可从克希霍夫定律导出的，因为在（8）中它已经出现了。

每一个非空白的字最终都加了标记。利用这一事实，以及行26，32和65，我们得到

$$J=Y-2B \tag{14}$$

其中Y是在输入的排列中出现的非空白字的个数。出现在输入的排列中的每个不同的元素，恰巧写到输出一次，或者是在行63，或者是在行77。从这一事实，我们得出（见等式(8)）

$$P=H+Q=\text{输入中不同元素的个数} \tag{15}$$

联想到行75这也是显然的。最后，我们从行84看出

$$S=\text{输出中单个元素的循环的个数} \tag{16}$$

显然，我们现在已予解释的量B，C，H，J，P和S，实际上都是些独立的参数，这些参数是预期要参与程序A之计时的。

我们至今所已得到的结果，使我们仅剩下未知的G和L尚待分析了。为此，我们必须使用一点技巧。从行39和行79开始的对输入的扫描，总是或者在行45（最后的一遍）或者在行75结束。在这些$P+1$回循环的每一回中，执行“1NC3 1”指令$B+C$次，这只是在行42，66和72处进行。所以我们就得到了连接我们所未知的G和L的非平凡的关系

$$G+J+L=(B+C)(P+1) \tag{17}$$

正巧，计时公式是$G+L$的函数(它包括$\cdots+3F+4G+\cdots+3K+4L+\cdots=\cdots+7G+\cdots+7L+\cdots$)，因此我们已无需试图再对单个的数量$G$和$L$进行任何进一步的分析。

综合以上所有的结果，我们推知，除了输入输出之外，总共的时间计为

$$(112NX+304X+N-2M-Y+10U+2V-11)u \tag{18}$$

在这个公式中，关于数据之特征方面所使用的新名称含义如下：

$X=$输入卡片的张数

$$
\begin{aligned}
&Y = \text{输入中非空白的场的个数（不包括最后的“=”）}\\
&M = \text{输入中循环的个数}\\
&N = \text{输入中不同元素名称的个数}\\
&U = \text{输出中循环的个数（包括单个元素的循环）}\\
&V = \text{输出中单个元素的循环的个数}
\end{aligned}
\tag{19}
$$

我们已经看到，用这种方法来分析象程序P这样的程序，在许多方面就和解一个有趣的难题差不多。

稍后我们将说明，如果假定输出的排列是随机的，则数量U和V，就平均值说来，将分别是H_N和1。

另一个解法 算法A就象通常人们要做这同一作业那样，把排列乘到一起。我们常常发现，有待由计算机解决的问题，非常类似于许多年来用人工奋战的问题。就这样，凡是一种由来已久的，已经为象我们这样的世人推广使用的求解方法，对于计算机算法也照样是一种适当的步骤。

然而，就如通常那样，我们发现，十分不适宜于人类使用的某种方法，实际上竟是对计算机最优越的方法。核心的原因是计算机的“思维”是不同的；它事实上有着不同类型的记忆。从我们的排列相乘的问题中，即可看出这种差别的情况——使用以下的算法，计算机就能够实现乘法，它扫视公式一遍，一齐记住正在进行乘法的排列的当前状态。如果说算法A为输出每一个元素，都要扫描整个公式一次；那么，新的算法却全都在一次扫描中来进行。这是凡人所不能可靠地做到的一招绝技。

现在我们就来观察这个为进行排列相乘的面向计算机的方法。从右至左地进行更为方便。考虑下列的表：

```
      ( a c f g ) ( b c d ) ( a e d ) ( f a d e ) ( b g f a e )
a --> d d a a a a a a a a a a a d d d d d d e e e e e e e e a a
b --> c c c c c c c c g g g g g g g g g g g g g g g g b b b b b
c --> e e e d d d d d d c c c c c c c c c c c c c c c c c c c c
d --> g g g g g g g ) ) ) d d ) ) ) b b b b b d d d d d d d d d
e --> b b b b b b b b b b b b b b a a a ) ) ) ) b b ) ) ) ) ) e
f --> f f f f e e e e e e e e e e e e e e e a a a a a a a a f f f
g --> a ) ) ) ) f f f f f f f f f f f f f f f f f f f f g g g g
```

在循环形式的每个字符之下的列，说明了直到右边的部分循环表示什么排列。例如，片断的公式“$\cdots d\ e)(b\ g\ f\ a\ e)$”表示排列

$$
\begin{pmatrix} a & b & c & d & e & f & g \\ e & g & c & b & ? & a & f \end{pmatrix}
$$

它出现在这个表的最右边的d之下。

从右到左地对这个表进行观察，就看出了它是怎样被构造出来的。字母x之下的那列，与其右边一列之不同处，仅仅是在行x处。在该行和该列处的新值，是一个在前一变化中已被改变了的值。更精确地说，我们有以下的算法：

算法B（循环形式的排列的乘法） 这个算法实际上与算法A有同样的结果。假定排列的元素名称是$x_1, x_2, \cdots, x_n$。我们使用一个辅助的表$T[1], T[2], \cdots, T[n]$；到这个算法结束的时候，当且仅当$T[i]=j$时，把输入排列的x_i变成x_j。

B1.〔起始〕 对 $1 \leqslant k \leqslant n$，置 $T[k] \leftarrow k$。而且，准备从右到左地扫描输入。

B2.〔下个元素〕（从右到左地）检查输入的下一个元素。如果输入已经穷尽，则算法结束。如果元素是“)”，则置 $Z \leftarrow 0$ 并重复步骤 B2；如果它是“(”，则转 B4；否则这个元素是对于某个 i 的 x_i，转到 B3。

B3.〔改变 $T[i]$〕 交换 $Z \longleftrightarrow T[i]$。如果这使得 $T[i]=0$，则置 $j \leftarrow i$。返回步骤 B2。

B4.〔改变 $T[j]$〕 置 $T[j] \leftarrow Z$。〔这时，j 是表示一个“)”所在的行，与 B2 转来时之左圆括弧“(”相匹配的右圆括弧“)”所在的行●。〕返回步骤 B2。

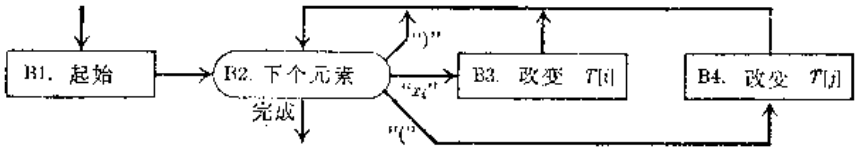
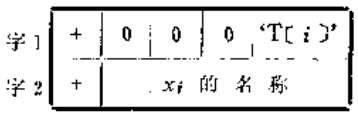


图 21 进行排列相乘的算法 B

当然，在执行了这个算法之后，我们仍然必须以循环形式输出表 T 的内容。通过“加标记”的方法，这很容易做到，如我们将在下面看到的那样。

现在让我们来写一个基于这个新算法的 MIX 程序。我们希望使用和在程序 A 中同样的基本规则，即是，输入和输出的形式实际上应当是相同的。这本身产生了一点小问题，就是，在预先还不知道元素 $x_1, x_2, \cdots, x_n$ 是什么的情况下，我们怎么能实现算法 B？我们预先不知道 n，而且我们预先不知道名为 b 的元素将是 x_1，还是 x_2，等等。为了解决这个问题，一个简单的方法，就是登记一张至今所遇到元素之名称的表，而且每次都来检索一下当前的名称（见以下程序中的 31～36 行）。

程序 B（与程序 A 的效果相同）。rX=Z；rI4=i；rI1=j；rI3=$2\times$(名字表大小)+1。名字表由两个字的项目组成：



（为 x_i 的项目的地址，如果 x_i 变成 x_j，）

（字符代码）

```
01  NAMES   ORIG   *+1000          名字表
02  CARDS   EQU    16
...                                与程序 A 的 01～19 行相同
20  *                                  这里，输入的 (rI2) 个字是在
21          DEC2   1          1   PERM, PERM+1, …中而且
22          ENT3   1          1   NAMES 表是空的
23  RIGHT   ENTX   0          A   置 Z ← 0。
24  SCAN    DEC2   1          B   B2. 下个元素
25          LDA    PERM,2     B
26          JAZ    CYCLE      B   跳过空格
27          CMPA   RPREN      C
```

● 参照前例中之表。而“)”今表示为 0。——译者注

```
28                  JE      RIGHT       C     下个元素是")"?
29                  CMPA    LPREN       D
30                  JE      LEFT        D     它是"("?
31                  ENT4    0,3         E     准备进行检索
32                  STA     NAMES       E     存储在表的开始处
33          2H      DEC4    2           F     检索名字表
34                  CMPA    NAMES+1,4   F
35                  JNE     2B          F     重复到匹配建立
36                  J4P     FOUND       G     这名称以前出现过?
37                  STA     NAMES+1,3   H     把新项目添入表中
38                  ST3     NAMES,3     H     置T[i]←i。
39                  ENT4    0,3         H
40                  INC3    2           H     增加表的大小
41  FOUND           LDA     NAMES,4     J     B3.改变T[i]
42                  STX     NAMES,4     J     存储Z
43                  SRC     5           J     置Z
44                  JANZ    SCAN        J
45                  ENT1    0,4         K     如果Z为0，则置j←i。
46                  JMP     SCAN        K
47  LEFT            STX     NAMES,1     L     B4.改变T[j]
48  CYCLE           J2P     SCAN        P     返回B2,除非完成了。
49  *
50  OUTPUT          ENT1    ANS         1     已经扫描全部输入
51                  DEC3    2           1     名字表含有答案
52          1H      LDAN    NAMES+1,3   Q     现在我们构造循环记号
53                  JAP     SKIP        Q     名字已加标记?
54                  CMP3    NAMES,3     R     有单个元素的循环?
55                  JE      SKIP        R
56                  MOVE    LPREN       S     开一个循环
57          2H      MOVE    NAMES+1,3   T
58                  STA     NAMES+1,3   T     对名字加标记
59                  LD3     NAMES,3     T     寻找元素的后继者
60                  LDAN    NAMES+1,3   T
61                  JAN     2B          T     它是否已经标记?
62                  MOVE    RPREN       W     是，关循环
63  SKIP            DEC3    2           Z     移到下一个名称
64                  J3P     1B          Z
65  *
66  DONE            CMP1    =ANS=       }
...                                     } 与程序A的46-60行相同
80  EQUALS          ALF     =           }
81                  END     BEGIN
```

行50-64，它是由T表（即是名字表）来构造循环记号的，它形成了一个值得进行某些研

究的堪称漂亮的小算法。参与这个程序之计时的量A、B，…，R，S，T，W，Z．当然不同于在程序A的分析中具有相同名称的那些量。读者将会发现，关于这些次数的分析，乃是一个饶有趣味的习题（见习题10）。

经验证明，程序B的执行时间，其主要部分将花在检索NAMES表上——这是计时中的数量F。实际上，为检索和构造这样一个NAMES表，可以利用更好得多的算法；这些算法称作符号表算法，而且它们在计算机的应用中有着巨大的重要性。第6章包含了关于高效的符号表算法之全面的讨论。

求逆 一个排列π的逆π^{-1}，就是一个消除π的效果的重新安排；如果在π之下，i变成j，则在π^{-1}之下，j变成i。因此乘积$\pi\pi^{-1}$等于恒等排列。

每一个排列有一个逆。例如，

$$\begin{pmatrix} a & b & c & d & e & f \\ c & d & f & b & e & a \end{pmatrix}\text{的逆是}\begin{pmatrix} c & d & f & b & e & a \\ a & b & c & d & e & f \end{pmatrix}=\begin{pmatrix} a & b & c & d & e & f \\ f & d & a & b & e & c \end{pmatrix}$$

我们现在将考虑为计算一个排列之逆的某些简单的算法。

对于这一小节的剩下部分，让我们假定我们正在处理数{1，2，…，n}的排列。如果$X[1]X[2]\cdots X[n]$是一个这样的排列，则有一个计算它的逆的简单方法：对于$1\leqslant k\leqslant n$，置$Y[X[k]]\leftarrow k$，则$Y[1]Y[2]\cdots Y[n]$就是所求的逆。这一方法使用$2n$个存储单元，给X n个，给Y n个。

就说是为了好玩吧，假设n是非常大的，也假设，我们希望不必使用很多的额外的存储单元，要来计算$X[1]X[2]\cdots X[n]$的逆。我们要“就地”计算逆，使得在我们的算法结束后，$X[1]X[2]\cdots X[n]$就是原先的排列之逆。单纯对$1\leqslant k\leqslant n$置$X[X[k]]\leftarrow k$肯定是要失败的。但是，通过考虑循环结构，我们却能导出以下简单的算法：

算法I（就地求逆） 以其逆来代替一个关于{1，2，…，n}之排列$X[1]X[2]\cdots X[n]$。参照：*CACM* 8 (1965)，670。

I1．〔起始〕置$m\leftarrow n$。

I2．〔下个元素〕置$i\leftarrow X[m]$。如果$i<0$，则置$X[m]\leftarrow -i$并转到I6（这个元素已经处理过）。如果$i=m$，则转到I6（这个元素对排列是固定的）。

I3．〔开循环〕置$k\leftarrow m$。

I4．〔对一元素求逆〕（在原先的排列中，$X[k]=i$。）置$j\leftarrow X[i]$，$X[i]\leftarrow -k$。

I5．〔结束循环〕如果$j=m$，则置$X[m]\leftarrow i$；否则置$k\leftarrow i$，$i\leftarrow j$，并且返回I4。

I6．〔还有元素？〕m减1；如果$m>0$，则转I2。否则算法结束。 ∎

表2 通过算法I计算621543的逆

（从左到右地来读各列）在*处，循环（163）已经求出了逆。

经本步骤后：	I1	I3	I4	I4	I6*	I3	I4	I6	I6	I6	I6	I6
$X[1]$	6	6	6	−3	−3	−3	−3	−3	−3	−3	−3	3
$X[2]$	2	2	2	2	2	2	2	2	2	2	2	2
$X[3]$	1	1	−6	−6	−6	−6	−6	−6	−6	6	6	6
$X[4]$	5	5	5	5	5	5	−5	−5	5	5	5	5
$X[5]$	4	4	4	4	4	4	4	4	4	4	4	4
$X[6]$	3	3	3	3	1	1	1	1	1	1	1	1
m	6	6	6	6	5	5	5	4	3	2	1	0
i		3	3	1	1	4	4	4	−5	−6	2	−3
j			1	6	6	6	5	5	5	5	5	5
k		6	6	3	3	5	5	5	5	5	5	5

关于这个算法的一个例子，见表 2。这个方法是以排列的逐个循环之求逆为基础的。

算法 I 类似于算法 A 的某些部分，而且它非常类似于程序 B 中的寻找循环的算法（50–64 行）。总之，它在许多涉及重新安排的算法中是典型的。实现它的 MIX 程序是十分简单的；见以下的程序 I。

程序 I（就地求逆） rI1≡m；rI2≡i；rI3≡$(-k)$；rI4≡j；$n\equiv N$，这是当本程序被汇编成一个更大的程序的一部分时，一个待定义的符号。

```
01    INVERT  ENT1  N          1          I1. 起始。m ← n
02    2H      LD2   X,1        N          I2. 下个元素。i ← X[m]
03            J2NN  *+3        N
04            STZ   X,1(0:0)   N-C        置X[m]为正
05            JMP   6F         N-C
06            CMP1  X,1        C          i = m?
07            JE    6F         C
08            ENN3  0,1        C-S        I3. 开循环。k ← m
09            JMP   1F         C-S
10    3H      INC4  0,1        N-2C+S
11            ENN3  0,2        N-2C+S     k ← i
12            ENT2  0,4        N-2C+S     i ← j
13    4H      LD4   X,2        N-C        I4. 对一元素求逆。j ← X[i]
14            ST3   X,2        N-C        X[i] ← -k
15            DEC4  0,1        N-C        I5. 结束循环
16            J4NZ  3B         N-C        j = m?
17            ST2   X,1        C-S        是，置X[m] ← i
18    6H      DEC1  1          N          I6. 还有元素?
19            J1P   2B         N          若m > 0 则转I2。 ∎
```

对这个程序的计时，很容易以如前边所示的方式来进行：结果是 $(17N-8C-S+1)u$，其中 N 是排列的阶，C 是循环的总数，而 S 是固定元素（单个元素的循环）的个数。后面，我们还要来分析在一个随机的排列中的量 C，S。

为完成任何给定的任务，几乎总是不只有一种算法。所以我们理应予期可能还有另外的求一排列之逆的方法。下列巧妙的算法是由约·博特罗伊德（J.Boothroyd）给出的：

算法 J.（就地求逆） 本算法与算法 I 的效果一样，但所用的方法不同。

J1.〔全部取负〕置 $X[k]\leftarrow -X[k]$，对于 $1\leqslant k\leqslant n$。且置 $m\leftarrow n$。

J2.〔给 j 初值〕置 $j\leftarrow m$。

J3.〔找负的项目〕置 $i\leftarrow X[j]$。若 $i>0$，则置 $j\leftarrow i$ 并重复这一步骤。

J4.〔求逆〕置 $X[j]\leftarrow X[-i]$，$X[-i]\leftarrow m$。

J5.〔对 m 进行循环〕m 减 1；如果 $m>0$，则返回 J2。否则算法结束。∎

关于这个算法的一个例子，请见表 3。这个算法实际上还是以循环结构为基础的。但这次，算法真正是如何地工作，已不太明显。留给读者自行验证（见习题 13）去吧！

表 3　通过算法 J 计算621543的逆

经本步骤后，	J 2	J 3	J 5	J 3	J 5	J 3	J 5	J 3	J 5	J 3	J 5	J 3	J 5
X[1]	−6	−6	−6	−6	−6	−6	−6	−6	3	3	3	3	3
X[2]	−2	−2	−2	−2	−2	−2	−2	−2	−2	−2	2	2	2
X[3]	−1	−1	6	6	6	6	6	6	6	6	6	6	6
X[4]	−5	−5	−5	−5	5	5	5	5	5	5	5	5	5
X[5]	−4	−1	−4	−4	−5	−5	4	4	4	4	4	4	4
X[6]	−3	−3	−1	−1	−1	−1	−1	−1	−6	−6	−6	−6	1
m	6	6	5	5	4	4	3	3	2	2	1	1	0
i		−3	−3	−4	−4	−5	−5	−1	−1	−2	−2	−6	−6
j	6	6	6	5	5	5	5	6	6	2	2	6	6

程序 J（类似于程序 I）　$r[1]\equiv m$；$r[2]\equiv j$；$r[3]\equiv(-i)$。

```
01    INVERT  ENN1  N              1     J1. 全部取负
02            ST1   X+N+1,1(0:0)   N     置负符号
03            INC1  1              N
04            J1N   *-2            N     还有元素
05            ENT1  N              1     m←n
06    2H      ENN3  0,1            N     J2. 给J初值
07            ENN2  0,3            A
08            LD3N  X,2            A     J3. 寻找负的项目
09            J3N   *-2            A     i>0?
10            LDA   X,3            N     J4. 求逆
11            STA   X,2            N     X[j]←X[-i]
12            ST1   X,3            N     X[-i]←m
13            DEC1  1              N     J5. 对m进行循环
14            J1P   2B             N     若m>0则转J2.
```

这个程序较前一个要略短些。为了判断它运行的时间究竟有多快，我们需要知道数量 A；这个量是很有趣的，并且很有启发性，因此把它留作一道习题（见习题 14）。

不论算法 J 如何精巧，但我们仍须不得不报告，对这两个算法进行分析的结果表明，算法 I 肯定地更为优越。事实上，结果是，算法 I 的运行时间实质上与 n 成比例，而算法 J 的运行时间则实质上与 $n\ln n$ 成比例。当 n 趋于无穷大时（这些算法主要是针对充分大的 n 来考虑的），两者执行时间的比率竟趋于 0！也许是不应有的——在这种情况下，更精巧的算法反倒失利了。但是对算法所作的分析却明确地告诉我们这一真实的情况。不过，它们完全可能相反于个人的审美观点而运行。可能有一天，有人将发现使用算法 J（或者是它的某一有关的修正）是非常绝妙的一位，以致不能将其全然忘却！

不寻常的对应　我们已经看到，对于一个排列，其循环记法并不是唯一的。对于 6 个元素的排列 (163)(45)，也可以写成 (54)(316)，等等。因而我们宜于来考虑循环记法的典型形式。典型形式就是唯一的了。为得到典型形式，可如下进行：

a）明显地写出所有的单个元素的循环。

b）在每个循环内，把最小的数写成头一个。

c）以诸循环中的头一个数的递降顺序而将循环排序。

例如，从 (316)(54) 开始，我们将得到

(a)：(316)(54)(2)；　(b)：(163)(45)(2)；　(c)：(45)(2)(163)　(20)

这种典型形式的重要性质，是圆括弧还可去掉而且仍可唯一地再重新将其构造出来。例如，在“452163”中仅有一种方式来插入圆括弧，以获得典型形式。人们必须把左圆括弧恰恰插入到每一个自左到右的极小值之前（即是，恰恰插入到每一个其前边再没有更小的元素的那样的元素之前）。

圆括弧的这种插入和撤消，给了我们一个不寻常的一一对应关系，这就是以循环形式表示的所有排列的集合，与以线性形式表示的所有排列的集合之间的一一对应关系〔例如，排列 621543 在循环形式下是 (45)(2)(163)；撤消圆括弧得到 452163，其循环形式则为 (2563)(14)；撤消圆括弧又得到 256314，其循环形式又为(364)(125)，等等〕。

这个对应对于研究各种类型的排列有许多应用。例如，让我们来问：“n 个元素的一个排列，就平均值说来，有多少个循环?”为了回答这个问题，我们考虑以典型形式表示的所有的 $n!$ 个排列的集合，并去掉圆括弧；我们得到在某种次序下的所有的 $n!$ 个排列的集合。于是，我们原来的问题就等价于：“n 个元素的排列，平均说来，有多少个从左到右的极小值?”在 1.2.10 小节我们已经回答了这个问题（实际上，我们讨论的是从右到左的极大值的平均数；由对称性，它实质上是相同的）；这就是在算法 1.2.10M 的分析中的数量 $(A+1)$，对于它我们已求得统计

$$\min 1,\ \text{ave}\, H_n,\ \max n,\ \text{dev}\sqrt{(H_n-H_n^{(2)})} \tag{21}$$

而且，我们已求得，n 个对象的一个排列有 k 个循环（即是，有 k 个自左到右的极小值）的概率是 $\left[{n \atop k}\right]/n!$。

我们还可以问及自左到右的极小值之间的平均距离，它就等价于一个循环的平均长度。由(21)，在所有的 $n!$ 个排列当中，循环的总个数是 $n!\,H_n$（因为它是 $n!$ 乘以循环的平均个数）。如果我们随机地选择一个循环，则它的平均长度是什么呢?

考虑以循环记法写出来的 $\{1,\ 2,\ \cdots,\ n\}$ 的所有的 $n!$ 个排列。我们问有多少个三（个元素的）循环出现？为了回答这个问题，让我们考虑一个特殊的三循环 $(x\ y\ z)$ 出现多少次：显然，循环 $(x\ y\ z)$ 精确地出现在 $(n-3)!$ 个排列中。因为，这是剩下的 $n-3$ 个元素可以被排列的总的数目。现在，可能的不同的三循环 $(x\ y\ z)$ 的个数是 $n(n-1)(n-2)/3$。因为，对 x 有 n 种选择，对 y 有 $n-1$ 种选择，对 z 有 $n-2$ 种选择，而且在这些 $n(n-1)(n-2)$ 种选择当中，每个不同的三循环已经以三种形式 $(x\ y\ z)$，$(y\ z\ x)$，$(z\ x\ y)$ 出现。因此，在所有的 $n!$ 个排列当中，三循环的总数是 $n(n-1)(n-2)/3$ 乘 $(n-3)!$，即是 $n!/3$。类似地，m 循环的总数是 $n!/m$，$1\leqslant m\leqslant n$（这就提供了循环的总数是 $n!H_n$ 这一事实的另一个简单的证明。因此，如同我们已经知道的：在一个排列中，循环的平均个数是 H_n）。如果我们考虑 $n!H_n$ 个循环是同等可能的，则一个随机地选定的循环的平均长度就是 n/H_n；如果在一个随机的排列中随机地选定一个元素，则含有该元素的循环的平均长度，就比这个数要大些（见习题 17）。

为了完成我们对算法 A，B 和 I 的分析，我们将希望知道，在一个随机的排列中的单（个元素的）循环的平均个数。这是一个有趣的问题。假设我们如此地写下 $n!$ 个排列：首先列出那些不带有单循环的，然后是仅带有一个单循环的，等等；例如，若 $n=4$，则列表为

无固定元素的： 2143 2341 2413 3142 3412 3421 4123 4312 4321

一个固定元素的：$\bar{1}342$ $\bar{1}423$ $3\bar{2}41$ $42\bar{1}3$ $243\bar{1}$ $4\bar{1}32$ $23\bar{1}\underline{4}$ $31\underline{2}\bar{4}$

两个固定元素的：$\underline{12}43$ $\underline{14}\bar{3}\underline{2}$ $\bar{1}\underline{32}\bar{4}$ $\underline{4}2\underline{3}1$ $\underline{3}\bar{2}\underline{1}\bar{4}$ $\underline{21}\bar{3}\underline{4}$

三个固定元素的：

四个固定元素的：$\overline{1234}$

（单循环，即固定的元素，在这个表中已经特殊地标出）。无固定元素的排列称作更列；更列的个数就是，要把 n 封信装到 n 个信封里边，让它们全都装错的所有可能的方式的数目。

设 P_{nk} 是 n 个对象的恰有 k 个固定元素的排列的个数，例如

$$P_{40}=9,\quad P_{41}=8,\quad P_{42}=6,\quad P_{43}=0,\quad P_{44}=1$$

研究上列的表，我们即可看出这些个数之间的主要关系。我们可以通过这样的方法来获得具有 k 个固定元素的所有排列：首先选择待固定的 k 个元素（这可以有 $\binom{n}{k}$ 种选法），而后在再没有任何固定元素的情况下，以所有的 $P_{(n-k)0}$ 种方法来排列剩下的 $n-k$ 个元素。因此，

$$P_{nk}=\binom{n}{k}P_{(n-k)0} \tag{22}$$

我们还有“全体等于其部分之和”这样一条规则：

$$n!=P_{nn}+P_{n(n-1)}+P_{n(n-2)}+P_{n(n-3)}+\cdots \tag{23}$$

把等式（22）和（23）结合起来，并稍微重写一下结果，我们求得

$$n!=P_{n0}+\frac{1}{1!}nP_{n0}+\frac{1}{2!}n(n-1)P_{n0}+\frac{1}{3!}n(n-1)(n-2)P_{n0}+\cdots \tag{24}$$

这是一个对所有正整数 n 都必须成立的等式。这个等式在以前我们就曾经碰见过——它曾经在 1.2.5 小节中，在关于斯特林试图推广阶乘函数的论述处出现过——并在 1.2.6 小节中给出过关于这些系数的简单推导（等式 32 及其后）。我们结论

$$\frac{1}{m!}P_{m0}=1-\frac{1}{1!}+\frac{1}{2!}-\cdots+(-1)^m\frac{1}{m!} \tag{25}$$

现在命 p_{nk} 为在 n 个对象的一个排列中，恰有 k 个单循环的概率。因为 $p_{nk}=P_{nk}/n!$，我们从等式(22)和(25)得到

$$p_{nk}=\frac{1}{k!}\left(1-\frac{1}{1!}+\frac{1}{2!}-\cdots+(-1)^{n-k}\frac{1}{(n-k)!}\right) \tag{26}$$

生成函数 $G_n(z)=p_{n0}+p_{n1}z+p_{n2}z^2+\cdots$ 因此是

$$G_n(z)=1+\frac{1}{1!}(z-1)+\cdots+\frac{1}{n!}(z-1)^n=\sum_{0\leqslant j\leqslant n}\frac{1}{j!}(z-1)^j \tag{27}$$

从这个公式，我们得出 $G_n'(z)=G_{n-1}(z)$；而且用 1.2.10 小节的方法，我们得到关于单循环个数的以下统计：

$$(\min 0,\ \text{ave}\ 1,\ \max n,\ \text{dev}\ 1),\quad \text{当}\ n\geqslant 2\ \text{时} \tag{28}$$

为了计算没有单循环的排列的个数，有一种稍微更直接的方法，这就是根据“包括和排除的原理”来进行。对于许多枚举问题，这原理都是一个重要的方法。一般的包括和排除原理，可以系统阐述如下：给定 N 个元素和这些元素的 M 个子集 S_1，S_2，…，S_M；我们的目标是要来计算不在这些子集中的元素有多少个。以 $|S|$ 表示在一个集合 S 中的元素

个数，则所求的不在任一集合 S_j 中的对象的个数是

$$N-\sum_{1\leqslant j\leqslant M}|S_j|+\sum_{1\leqslant j<k\leqslant M}|S_j\cap S_k|-\sum_{1\leqslant i<j<k\leqslant M}|S_i\cap S_j\cap S_k|+\cdots$$
$$+(-1)^M|S_1\cap\cdots\cap S_M| \tag{29}$$

（这样，我们首先从总数 N 中，减去在 S_1，…，S_M 中的元素个数，但这就把所求的总数低估了；所以我们还须把同属于一对集合，即是对于每对 S_j 和 S_k，属于 $S_j\cap S_k$ 的元素个数加回去；然后又减去同属于三个集合的元素个数，等等）。要证明这个公式，可以有很多种办法，鼓励读者自己去找出一种办法来。

为了计算 n 个元素的没有单循环的排列的个数，我们来考虑 $N=n!$ 个排列，并命 S_j 为在其中元素 j 形成一单循环的排列之集合。如果 $1\leqslant j_1<j_2<\cdots<j_k\leqslant n$，则在 $S_{j_1}\cap S_{j_2}\cap\cdots\cap S_{j_k}$ 中之元素的个数，即是在其中 j_1，j_2，…，j_k 皆为单循环的排列的个数，而这个数显然是 $(n-k)!$。于是公式（29）变成为

$$n!-\binom{n}{1}(n-1)!+\binom{n}{2}(n-2)!-\binom{n}{3}(n-3)!+\cdots+(-1)^n\binom{n}{n}0!$$

这正好与（25）是一致的。

包括和排除的原理是由阿·德·莫尤勒给出的（见他所著的《机会原理》(Doctrine of chances)（伦敦，1718年），61-63；第三版（1756年，由切尔西重印，1957年），110-112）。但它的意义未被普遍地理解，直到威·艾·惠特沃思(W. A. Whitworth) 在其著名的书《选择与机会》(Choice and Chance)（1867年，剑桥）中进行了宣传并进一步发展之后，才引起普遍的重视。

在5.1节中，将对排列的组合性质作进一步的阐述。

习题

1.〔10〕 证明通过规则 x 变成 $(2x)\bmod 7$ 来定义的，数{0，1，2，3，4，5，6}的变换是一个排列，并以循环形式把它写出来。

2.〔10〕 正文当中说明了，通过使用一系列的替代运算和一个辅助变量 t，我们就能实现置 $(a,\ b,\ c,\ d,\ e,\ f)\leftarrow(c,\ d,\ f,\ b,\ e,\ a)$。试说明通过使用一系列的交换运算（即是，$x\leftrightarrow y$）而且不用辅助变量，又怎样来实现之？

3.〔16〕 计算乘积

$$\begin{pmatrix}a&b&c&d&e&f\\b&d&c&a&f&e\end{pmatrix}\times\begin{pmatrix}a&b&c&d&e&f\\c&d&f&b&e&a\end{pmatrix}$$

并以两行的记法（参照等式4）把答案表达出来。

4.〔10〕 借助于不相交的循环表达 $(a\ b\ d)(e\ f)(a\ c\ f)(b\ d)$。

▶5.〔*M*10〕 等式(3)说明了以循环形式表达同一排列的若干等价的方式。如果所有的单循环都去掉的话，则写出这一排列有多少种可能的不同方式？

6.〔*M*23〕 如果我们去掉关于所有空白字都出现在最右端的假定，则程序A的计时将会发生什么变化？

7.〔10〕 如果以输入（6）来写出程序A，则（19）式中的量 X，Y，M，N，U 和 V 是什么？除了输入输出之外，程序A所需要的时间是多少？

▶8.〔23〕 能否把算法B修改成从左到右地而不是从右到左地扫描输入?

▶9.〔10〕 程序A和B都是接受相同的输入，而且以实质上相同的形式给出答案。在这两个程序之下，输出真的是完完全全相同吗?

▶10.〔*M*28〕 检查程序B的计时特征，即是那里已说明过的量A，B，…，Z；借助于X、Y，M，N，U，V〔参照(19)〕和量F。来表达总共的时间。对于输入(6)，在现在的情况下，利用$F=74$这一事实，比较程序B与程序A的总时间(参照习题7)。

11.〔15〕 如果排列π是以循环形式给出的，试找出一个以循环形式写出π^{-1}的简单规则。

12.〔*M*27〕 (转置一长方阵)。假设一个$m\times n$矩阵(a_{ij})，$m\neq n$，以类似于习题1.3.2-10的方式存于存储器中，使得a_{ij}的值出现在单元$L+n(i-1)+(j-1)$中，其中L是a_{11}的单元。问题是要找出一个方法，来把这个矩阵转置，以得到一个$n\times m$矩阵(b_{ij})，其中$b_{ij}=a_{ji}$，而且b_{ij}就存于单元$L+m(i-1)+(j-1)$中。如此，矩阵就"通过自身"进行转置。(a)试说明这个转置的变换，把出现于单元$L+x$的值，移到单元$L+(mx)\bmod N$中，其中$0\leqslant x<N=mn-1$。(b)试讨论通过计算机来进行这一转置的方法。

▶13.〔*M*24〕 证明算法J是正确的。

▶14.〔*M*34〕 试求在算法J的计时中数量A的平均值。

15.〔*M*12〕 是否存在一个排列，在不带圆括弧的典型循环形式下和在线性形式之下，它恰好都表示相同的变换?

16.〔*M*15〕 由线性记法的排列1324开始；把它转换成典型的循环形式，然后去掉圆括弧；重复这个过程，直至达到原先的排列为止。在这个过程中，都出现了些什么排列?

17.〔*M*24〕 (a)正文中证明了在n个元素的所有排列当中，有$n!H_n$个循环。如果这些循环(包括单循环)都逐个地写在$n!H_n$张纸上，而且如果随机地选择这些张纸中的一张来，则如此选出之循环的平均长度是什么?(b)如果我们在$n!$张纸上写出$n!$个排列，而且如果我们随机地选择一个数k，也选择这些张纸中的一张来，试问含有这元素k的循环为一m循环的概率是什么?含有k的循环的平均长度又是什么?

▶18.〔*M*27〕 n个对象的一个排列，其恰有k个m循环的概率p_{nkm}是什么?相应的生成函数$G_{nm}(z)$是什么?m循环的平均个数是什么?标准偏差又为何?(正文中仅仅考虑了$m=1$的情况)。

19.〔*HM*21〕 在等式(25)的记号下，说明对于所有的$n\geqslant 1$，更列的个数P_{n0}恰好等于$(n!/e)$舍入到最接近的整数。

20.〔*M*20〕 假定所有的单循环都要明显地写出来。问要写出一个有α_1个一循环，α_2个二循环，…的排列的循环记法，共有多少种不同的方式?(参照习题5.)

21.〔*M*22〕 n个对象的一个排列，其恰有α_1个一循环，α_2个二循环，等等的概率$P(n;\ \alpha_1,\ \alpha_2,\ \cdots)$是什么?

▶22.〔*HM*34〕 (由劳·希普(L.Shepp)和斯·菲·劳埃德(S.P.Lloyd)提出的下列方法，给出了一个为解决关于随机排列的循环结构问题之方便而又有效的方法。)不再认为对象的个数n是固定的，而排列是可变的，现在，让我们假设，按照某种概率分布，而独立地

选择出现于习题 20 和 21 中的量 α_1，α_2，α_3，…。命 w 为 0 与 1 之间的任何实数。

（a）假设我们对于某个函数 $f(w, m, k)$，按照“$\alpha_m = k$ 之概率为 $f(w, m, k)$”的规则，来选择随机变量 α_1，α_2，α_3，…。试确定 $f(w, m, k)$ 之值，使得下列两个条件成立：

i）$\sum_{k\geq 0} f(w, m, k) = 1$，对于 $0 < w < 1$ 和 $m \geqslant 1$。

ii）$\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots = n$ 和 $\alpha_1 = k_1$，$\alpha_2 = k_2$，$\alpha_3 = k_3$，…的概率是 $(1-w)w^n P(n; k_1, k_2, k_3, \cdots)$，其中 $P(n; k_1, k_2, k_3, \cdots)$ 如习题 21 所示。

（b）其循环结构为 α_1，α_2，α_3，…的一个排列，显然地恰好排列 $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots$ 个对象。试证明：如果诸 α 是按照小题（a）中的概率分布而随机地选定的，则 $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots = n$ 的概率是 $(1-w)w^n$；而 $\alpha_1 + 2\alpha_2 + 3\alpha_3 + \cdots$ 等于无穷的概率是 0。

（c）设 $\phi(\alpha_1, \alpha_2, \cdots)$ 是无穷多个数 α_1，α_2，…的任意函数。试证：如果诸 α 是按照（a）中的概率分布选定的，则 ϕ 的平均值是 $(1-w)\sum_{n\geq 0} w^n \phi_n$；这里 ϕ_n 表示 ϕ 遍取 n 个对象之所有排列的平均值，其中 α_1，α_2，…表示排列的循环个数〔例如，如果 $\phi(\alpha_1, \alpha_2, \cdots) = \alpha_1$，则正文中已说明 $\phi_n = 1$，此即不管 n 为何值的单循环的平均个数〕。

（d）利用这个方法，试求在 n 个对象的一个随机排列中，其长度为偶数的循环的平均个数。

（e）利用这个方法，来解习题 17（参照习题 1.2.10-15）。

23.〔*HM44*〕（戈龙卜（Golomb），希普，劳埃德。如果 l_n 表示在 n 个对象的一个排列中，最长循环的平均长度，试证 $l_n \approx \lambda n + \frac{1}{2}\lambda$，其中 $\lambda \approx 0.62433$ 是一个常数。事实上也就是证明 $\lim_{n\to\infty}\left(l_n - \lambda n - \frac{1}{2}\lambda\right) = 0$。

24.〔*M41*〕 试求参与算法 J 的计时中之量 A 的方差（参照习题 14）。

25.〔*M22*〕 证明等式（29）。

▶26.〔*M24*〕 推广包括和排除原理，以得到一个关于恰好是在子集 S_1，S_2，…，S_M 之中的 r 个子集中的元素的个数公式（正文中仅考虑 $r = 0$ 的情况）。

27.〔*M20*〕 利用包括和排除原理，来计算在 $0 \leqslant n < am_1m_2\cdots m_t$ 的范围内，不为任何 m_1，m_2，…，m_t 所整除的整数 n 的个数。这里，a，m_1，m_2，…，m_t 都是正整数，而且当 $j \neq k$ 时，$\gcd(m_j, m_k) = 1$。

28.〔*M21*〕（I. 卡普兰斯基（I. Kaplansky））如果在习题 1.3.2-22 中所定义的“约瑟夫斯排列”以循环形式表示，则当 $n = 8$ 和 $m = 4$ 时，我们就得到（1536824）（7）。试证在一般情况下，这个排列是乘积 $(n, n-1, \cdots, 2, 1)^{m-1}(n, n-1, \cdots, 2)^{m-1}\cdots(n, n-1)^{m-1}$。

29.〔*M25*〕 证明当 $m = 2$ 时，约瑟夫斯排列的循环形式可以通过下述方法来得到，即：首先表示 $\{1, 2, \cdots, 2n\}$ 的“双倍”排列，它是循环形式的，把 j 变成 $(2j) \bmod (2n+1)$；然后把左边与右边颠倒过来并删去所有大于 n 的数。例如，当 $n = 11$ 时，双倍排列是 (1, 2, 4, 8, 16, 9, 18, 13, 3, 6, 12)(5, 10, 20, 17, 11, 22, 21, 19, 15, 7, 14)，而约瑟夫斯排列是 (7, 11, 10, 5)(6, 3, 9, 8, 4, 2, 1)。

30.〔*M24*〕 利用习题 29 来证明，当 $m = 2$ 时约瑟夫斯排列的固定元素就精确地等于

诸数使得 $(2^{d-1}-1)(2n+1)/(2^d-1)$ 为整数的所有正整数 d，$(2^{d-1}-1)(2n+1)/(2^d-1)$ 的诸值。

31. [*HM33*] 推广习题 29 和 30，证明对于一般的 m 和 n，待处决的第 k 个人，是位于位置 x 处的人，这个位置 x 可以如下来计算：置 $x \leftarrow km$，然后重复地置 $x \leftarrow \lfloor (m(x-n)-1)/(m-1) \rfloor$ 直到 $x \leqslant n$。因而，对于 $1 \leqslant n \leqslant N$ 以及固定的 m，当 $N \to \infty$ 时固定元素的平均个数，趋向于 $\sum_{k\geqslant 1}(m-1)^k/(m^{k+1}-(m-1))$。（由于这个值位于 $(m-1)/m$ 与 1 之间，约瑟夫斯排列比起随机的排列来，有稍微少些的固定元素）。

1.4 某些基本的程序设计技术

1.4.1 子程序

当在一个程序中的若干不同的地方，都要执行同一个确定的任务时，通常不希望在每个地方都重复地来对这一任务进行编码。为避免这种重复情况，这个代码（称为"子程序"）可以只放在一个地方，而且在子程序完成之后，可以附加少量额外的指令，以便适当地重新开始外部的程序。子程序与主程序之间控制的转移，称为"子程序的链接"。

为了实现有效的子程序链接，每部机器都有它自己特殊的方式，通常需要有特殊的指令。在 MIX 中，J 寄存器就用于这一目的；我们的讨论就将以 MIX 机器为基础，但是类似的说明也适用于其它计算机的子程序链接。

子程序用来节省一个程序的空间；但它并不节省任何时间。除了通过使用较少的空间而含蓄地节省了时间之外（例如，为装入程序而使用较少的时间，或者在程序中只须较少遍数的扫描，或者在配备有若干级存储器的机器上更好地使用高速存储），为进入和离开一个子程序所花费的额外时间，通常是微不足道的。

子程序还有许多其它的优点。它使得比较易于来想象一个庞大而又复杂的程序的结构；诸子程序把整个问题分割成若干逻辑段，而且这通常会使程序的查错更容易些。由于许多子程序除了供它们的设计人员使用之外，还能为其他人员所使用，这就使它们有了额外的价值。

大多数计算机装置，都配备了有用的子程序之大型的"程序库"，而且这样一个程序库，大大地方便了所出现的标准的计算机应用之程序设计。然而，一个程序员不应该把这就认为是子程序唯一的用途；子程序不应该只认为仅仅是为公众使用的"通用"程序。甚至使用非常专用的子程序——这种专用子程序是预期仅仅出现在一个程序中的，也都是一项重要的技术。

最简单的子程序是仅仅有一个入口和一个出口的那样的子程序，例如我们所已讨论过的 MAXIMUM 子程序（见 1.3.2 小节的程序 M）。为便于参考，我们在这里重新转抄这个程序，但是略作更改，使其检索固定数量为 100 个单元，以寻找极大值：

```
MAX100  STJ   EXIT      子程序链接
        ENT3  100       M1. 起始
        JMP   2F
1H      CMPA  X,3       M3. 比较
        JGE   *+3                                  (1)
```

```
2H      ENT2    0,3     M4. 改变m.
        LDA     X,3     (找到新的极大值)
        DEC3    1       M5. k减1.
        J3P     1B      M2. 全都试过了?
EXIT    JMP     *       返回主程序。
```

在一个更大的以此代码作为一个子程序的程序中，一条指令“JMP MAX100”，就把寄存器A置成单元X＋1到X＋100的当前极大值，而且极大值的地址将出现于rI2中。在这种情况下，子程序链接是通过“MAX100 STJ EXIT”以及后面的“EXIT JMP＊”这样两条指令来实现的。按照J寄存器的这种操作方式，出口指令“EXIT JMP＊”就将跳到原来对MAX100进行访问的位置之后的那个单元。

当使用子程序时，不难得到关于因此而节省的空间数量和因此而损失的时间数量之定量性的论断。假设一块代码要占用k个单元，并且在程序中它出现于m处。如果把它改写成子程序，则我们需要一条额外的指令STJ和对于子程序的一条出口行，加上在调用子程序的m处的每一处需要一条JMP指令。因此，总共为$m+k+2$个单元，而不是mk。所以节省的数量是

$$(m-1)(k-1)-3 \tag{2}$$

如果k为1或m为1，则我们不可能通过使用子程序来节省任何空间；当然，这是明显的。如果k为2，为要有所获益，则m必须大于4，等等。

所损失的时间的总数，就是由于额外的JMP，STJ和JMP指令所花去的时间，因为如果不使用子程序，则就不需要这些指令。因此，如果在运行一个程序时，使用这个子程序4次，则需要$4t$个额外的时间周期。

这些估计务必不要全信，因为它们是对于理想化了的情况来给出的。许多子程序不能简单地用一条JMP指令来调用。而且，如果不使用子程序的方法，而让这个代码在一个程序的许多部分重重复复，则对于每一部分的代码，还可能利用它所在程序的具体部分的特殊特征之有利条件。另一方面，为要使用一个子程序，代码就必须针对最一般的情况，而不是针对一个特定的情况来编写，当然这通常就要增加一些额外的指令。

当把一个子程序写成来处理一般的情况时，通常都要借助于参数，用参数的值来支配子程序的动作；当由子程序的一次调用转到另一次调用时，参数的值将随之改变。

外部程序中，把控制转移到子程序，并适当地启动子程序的代码，称作“调用序列”。在调用子程序时所提供参数的具体值，则称作自变量。对于我们的MAX100子程序，调用序列很简单，就是“JMP MAX100”；但当必须提供自变量时，一般就需要较长的调用序列了。例如，程序1.3.2M是MAX100的一个推广，它找的是这个表中头n个元素的极大值。参数n出现于变址寄存器1中，而且我们可以认为调用序列是

```
LD1   = n =
JMP   MAXIMUM
```

如果调用序列占C个存储单元，则关于所节省的存储空间的数量的公式（2）就变成

$$(m-1)(k-c)-\text{常数} \tag{3}$$

而由于子程序链接所损失的时间也就略有增加。

由于某些寄存器可能需要予以保留和恢复，上边的公式可能还需要进一步加以修正。

例如，在 MAX100 子程序中，程序员必须记住，通过写出“JMP MAX100”，他不仅仅在寄存器 A 中得到了极大值，而且在寄存器 I2 中得到了它的地址；他还把寄存器 I3 置成了 0。一个子程序可能破坏寄存器的内容，而这是务必细心牢记的。为了防止 MAX100 改变 rI3 的内容，又需要附加额外的指令。对于 MIX 说来，实现这一点的最短和最快的方法，是在 MAX100 之后插入指令“ST3 3F(0:2)”，在 EXIT 之前插入“3H ENT3 *”。这样，所花的纯代价就是额外的两行代码，加上每次调用子程序的三个机器周期。

子程序可以看作是对计算机机器语言的扩充。一旦在内存中存储了 MAX100 子程序，则我们就有了一条寻求极大值的指令（即是“JMP MAX100”）。重要的是，要细心地定义每个子程序的作用，一如对于机器语言操作符本身所已定义了的那样；而且，程序员必须保证写出每个子程序的特征，即使他本人乃是唯一的使用它的人也得这样。在 MAXIMUM 的情况下，如在 1.3.2 小节中所给出的，其特征如下：

$$\begin{aligned}&\text{调用序列：}\quad \text{JMP}\quad \text{MAXIMUM}\\&\text{进入条件：}\ rI1 = n\text{；假定 } n \geqslant 1\text{。}\\&\text{出口条件：}\ rA = \max_{1\leqslant k\leqslant n}(X-k)\text{ 的内容}\\&\qquad\qquad = (X + (rI2))\text{ 的内容；}\\&\qquad rI3 = 0\text{；}\ rJ\text{ 和 CI 也受影响。}\end{aligned}\tag{4}$$

（我们习惯上将不谈论寄存器 J 和比较指示器受一个子程序影响的事实；这里提及它仅仅是为了完整性。）注意 rX 和 rI1 不为子程序的动作所影响，否则这些寄存器都将在出口条件中提出。也还需要说明子程序之外还有那些存储单元受到影响；在这种情况下，我们可以断定，没有存储什么东西，因为在（4）中并没有提到关于存储器的改变。

现在让我们考虑子程序的多个入口。假设我们有一个需要一般的子程序 MAXIMUM 的程序，但其中最经常需要用到的是对于 $n=100$ 的特殊情况 MAX100。那么，就可以把这两者结合起来如下：

```
MAX100    ENT3    100      头一个入口
MAXN      STJ     EXIT     第二个入口
          JMP     2F       如（1）那样继续下去          (5)
...
EXIT      JMP     *
```

子程序（5）实质上和（1）是一样的，只是头两条指令变换了；我们利用了“ENT3”不改变 J 寄存器内容这一事实。如果我们要对这个子程序增加第三个入口 MAX50，则我们将在开始处插入代码

```
MAX50     ENT3    50
          JSJ     MAXN                                  (6)
```

（回忆一下，“JSJ”指的是不改变寄存器 J 的转移）。

当参数较少时，为了把参数传送给子程序，通常都希望或者把它们存放在方便的寄存器内（就象我们用 rI3 来保存 MAXN 中的参数 n 那样，也象我们用 rI1 来保存 MAXIMUM 中的参数 n 那样），或者把它们放在固定的存储单元中。还有另一种提供自变量的方式，通常说来也很方便，就是简单地把它们列在 JMP 指令之后。由于子程序知道 J 寄存器的

内容，所以它就可以访问它的参数。

例如，如果我们想把对 MAXN 的调用序列写成

```
        JMP     MAXN
        CON     n                                          (7)
```

则子程序即可写成

```
MAXN    STJ     *+1
        ENT1    *           rI1←rJ
        LD3     0,1         rI3←n
        JMP     2F          如（1）那样继续下去            (8)
...
        J3P     1B
        JMP     1,1         返回
```

在象 360 系统那样的机器上对于这些机器，通常是通过把出口单元放进变址寄存器来实现链接的。上述的步骤来得特别方便。当有相当多的自变量待传送到一个子程序时，也可以使用上述的步骤。还有，在同由编译程序写成的程序相联系时，同样也可以使用上述的步骤（见 12 章）。但是，我们上边使用的多个入口的技巧，在这种情况下常常失效；然而，我们可以通过写成

```
MAX100  STJ     1F
        JMP     MAXN
        CON     100
1H      JMP     *
```

来“冒充”它，但这已不象（5）那样有吸引力了。

类似于在转移指令之后列出自变量这样的一种技巧，通常都是对于带有多个出口的子程序来使用的。多个出口意味着，依据子程序所侦知的情况，我们要求子程序返回到若干不同的单元之一。在最严格的意义下，子程序的出口的单元是一个参数；所以如果依据情况，子程序有若干个出口的位置，则这些位置都应该作为自变量来提供。我们最后的“求极大值”子程序的例子，将有两个入口和两个出口。调用序列是：

```
        对一般的 n                                  对 n = 100
ENT3    n
JMP     MAXN                                        JMP     MAX100
出口在此若 max≤0 或max≥rX                          出口在此若 max≤0 或 max≥rX
出口在此若 0<max<rX                                 出口在此若 0<max<rX
```

（换句话说，当极大值为正而且小于寄存器X的内容时，则出口在转移指令之下两个单元处）。对于这些条件的子程序，是容易写出的：

```
MAX100  ENT3    100         对于 n = 100的入口
MAXN    STJ     EXIT        对于一般的 n 的入口
        JMP     2F          如（1）中那样继续下去
...
        J3P     1B
        JANP    EXIT        极大值为正?                (9)
        STX     TEMP
```

```
          CMPA    TEMP
          JGE     EXIT          它小于rX?
          INC3    1             置r13←-1
EXIT      JMP     *,3           返回适当位置
```

总而言之，为节省一个程序的空间和减少它的复杂性，经常地须要子程序。始终是，我们并不白白地得到什么，在运行时间上要有所损失。如果为调用一个子程序所花的额外时间，与这个子程序的总的执行时间相比起来很小，并且如果子程序相当长而又十分经常地被访问，则把它写成一个子程序，比起在程序中一次又一次地重写这个代码，就更好些。关于子程序在一个庞大的程序中的典型的使用，请参见1.4.3.1小节的一些例子和第9章中整个的汇编程序。

子程序又可以调用其它子程序。在复杂的程序中，子程序的调用嵌套到五层以上，并不是不经常的情况。然而，当使用这里所叙述的链接时，则必须遵守这样的限制，就是子程序不得调用任何其它的（直接或间接地）调用它的子程序。例如，

```
〔主程序〕          〔子程序 A〕          〔子程序 B〕          〔子程序 C〕
                A   STJ  EXITA       B   STJ  EXITB       C   STJ  EXITC
    ⋮               ⋮                    ⋮                    ⋮
JMP  A          JMP  B               JMP  C               JMP  A          (10)
    ⋮               ⋮                    ⋮                    ⋮
                EXITA JMP  *         EXITB JMP  *         EXITC JMP  *
```

如果主程序调用到A，A调用B，而B又调用C，而后C又调用A，则EXITA中的访问主程序的地址就被破坏了，因而已无法再返回到主程序去。类似的说明也适用于为每个子程序所使用的全部临时存储单元和寄存器。要使子程序链接也能妥善地处理上述的"递归"状态，那也是可能的，这将在第8章中讨论。

在结束这一小节之前，我们来简单地讨论一下，怎样写出一个复杂的和长长的程序。我们怎样来判断，我们将需要什么类型的子程序，并且应当使用什么调用序列呢？为确定这些问题，有一个成功的方法，这就是使用一套迭代的步骤●：

步骤0（开始的想法）。首先，我们笼统地决定着手这个程序的总计划。

步骤1（程序的一个粗略草案）。开始，现在我们以任何方便的语言，写出程序的"外层"。埃·怀·迪伊克斯特拉(E.W.Dijkstra)的《结构程序设计》(Structured Programming)，科学出版社（Academic Press, 1972）第1章，以及尼·沃思(N.Wirth)，CACM 14 (1971)，221–227，非常精辟地叙述了一个关于进行这项工作的颇为系统的方法。我们可以通过把整个程序分成一些小块来开始，可以把这些小块暂时想象作一些子程序，尽管它们仅仅被调用一次。这些块又逐次地纯化，精炼，加细，成为越来越小的部分，这些小的部分相应地有更简单的工作任务。每当出现某件事情，它似乎在别处也要类似地出现，或者它在别处已经出现过，则我们就可以定义一个（实际的一个）完成这项工作的子程序。这时我们还不写出这个子程序来，我们假定这个子程序已经执行了它的任务，而继续来写主程序。最后，在我们已经把主程序构划出来之后，我们才回过头来着手写子程序。首先，要试图拿下最复杂的子程序以及它们的子子程序，等等。用这样的方法，我们就得到了一张子程序清单。每个子程序的实际的功能可能已经改动了若干次，以致我们前一部分的草案，如今已

● 作者下文讲述的，实际上是程序设计方法论，即由顶而下和自底而上相结合，反复迭代，逐步求精的方法。极其重要，读者切勿轻易掠过。——译者注

经不适用了。但这并不成问题，因为它只不过是一个草案。对于每个子程序，我们现在对于怎样调用它以及它的通用程度如何，已经有了一个相当好的想法了。通常，宜于让每个子程序的通用性都广一点。

步骤2(头一个工作程序)。这一步是以相反于步骤1的方向来进行的。我们现在用计算机语言，比如说用MIXAL或PL/MIX来写；这一次，我们由最低层的子程序开始，最后才来写主程序。尽可能地，在一个子程序本身已经编成之前，我们决不试图去写任何调用这个子程序的指令。(在步骤1时，我们是以相反的方向来进行的，即是，在写出一个子程序的所有调用之前，决不试图去考虑这个子程序。)

随着在这过程中写出了越来越多的子程序，我们的自信就越来越增长，因为我们正在连续地扩充我们正在为之进行程序设计的机器的功能。在编出一个独立的子程序之后，我们就要立即准备一个完整的描述：它干什么工作，以及它的调用序列是什么，等等，就如(4)中那样。不要让临时存储单元互相重迭起来，这也是要紧的。如果每个子程序都访问单元TEMP，那可能就是极大的灾难，尽管在步骤1准备草案的时候，为图方便而未为这个问题担心。为了避免重迭的麻烦，一个显然的办法，就是每个子程序都只用它自己的临时存储单元；但如果这太浪费空间，则另一个相当好的方案，就是来命名单元TEMP1，TEMP2，等等。在一个子程序内，编号从TEMP j 开始，其中 j 是一个比这个子程序的任何子子程序所用的最大的数还要大的数。

步骤3(重新检查)。步骤2的结果应该是非常接近于一个工作程序了，但是也许还有可能进行改进。为此，一个好办法就是再次地扭转方向，对于每个子程序，研究所有对它的调用。可能这样会更好些，就是把子程序所作的事情加以扩充，使它能做某些更为普遍的事情——外部程序在其调用此子程序之前或之后所经常要做的那些事情。也许某些子程序应该合并成一个；或者也许一个子程序仅被调用过一次(如果我们很幸运，则也许竟一次也没有调用)，因此根本就不应把它作为一个子程序。

至此，废弃每件事情并重新从步骤1开始，通常是一个好的想法！这并不是开玩笑；到这一步为止所花费的时间并不是被浪费了，因为我们已经对于解决我们的问题学到了许多东西。我们可能会知道对于程序的组织所能作的若干改进；没有理由害怕要返回步骤1——在一个程序写完之后，再次来通过以前步骤，肯定是容易得多了。而且，由于把时间花在重写程序上，我们将节省许多以后查错的时间。某些已经写成的最好的计算机程序，都可以把它们的许多成功，归结成这样一条经验，即是，在这个阶段前后的所有工作全都推倒重来，因而编写要从头开始。

另一方面，大概决不可能有那样的时候，就是一个复杂的计算机程序就不能再作稍许改进了。因此，步骤1和2不应被无限地重复下去；关于进一步的讨论请参看第9章。当明显地能够作出重大的改进时，就很值得要求再花额外的时间重新开始，但是，最终是会达到不再反复的境界的。

步骤4(调试)。在程序经过最后的推敲之后，也许还包括存储分配和其它微小的细节，现在，就是对它进行考察的时候了！而这仍然是以相反于步骤1、2和3这三者所沿的方向来进行的——我们是以计算机执行程序的顺序来考察程序的。当然，这可以通过手工，或者通过机器来做。作者感到，在这时，利用系统例行程序，让这例行程序跟踪每条指令的

头两次执行，是十分有帮助的；重要的是，要重新考虑那些奠定程序之基础的想法并验证每一件真正地发生的事情，是否恰如所预期的那样。

调试是一门技巧，是需要进行许多进一步的研究的。而且，实现调试的途径，高度地依赖于每一计算机装置所可利用的设施。为进行有效的调试，其良好的开端，如同在第9章所讨论的那样，通常就是准备恰当的测试数据。最有效的调试技术，似乎就是被设计和被构造在程序本身之中的那些——今天许多最好的程序员，都常贡献出他们近一半的程序，来方便另一半程序的调试过程；这前一半，它通常是由相当直接了当的例行程序组成的，这些例行程序要以便于阅读的格式显示有关的信息；但最终，这前一半将被甩掉，而唯一剩下的结果，就是在生产率方面的令人震惊的收获。

另一个好的调试作法，是把所出现的每一个错误的记录保存下来。诚然，这大概将是十分麻烦的，但是，这样的信息对于进行关于调试问题研究的任何人，都是无法估价的，而且它还将帮助你学习怎样来减少未来的错误。

习题

1.〔10〕 说明子程序（5）的特征，就象（4）给出子程序1.3.2M的特征那样。

2.〔10〕 试不用JSJ指令，而提出代替（6）的代码。

3.〔M15〕 试通过确切地指出：由于子程序执行的结果，寄存器J和比较指示器发生了什么情况，来完备（4）中的信息；并指出：如果寄存器I1的内容不是正的，则将发生什么情况。

▶4.〔21〕 将MAXN加以扩充，写出一个求$X[1], X[1+r], X[1+2r], \cdots, X[n]$之极大值的子程序来，其中$r$和$n$是参数。对于$r=1$的情况给出一个特殊的入口。

5.〔21〕 假设MIX没有J寄存器了。试提出一种不利用寄存器J而进行子程序链接的手段；并通过写出一个在效果上等价于（1）的MAX100子程序，来给出一个关于你所发明的例子。以类似于（4）的方式说明这个子程序的特征。

▶6.〔26〕 假设MIX没有了MOVE操作符。试写出一个标题为MOVE的子程序来，使得调用序列

```
JMP     MOVE
NOP     A,1(F)
```

有着与“MOVE A,1(F)”——当允许使用它时——完全相同的效力。唯一的区别就是：关于它对寄存器J的作用，还有，为执行这个子程序的时间要稍微长些。

1.4.2 共行程序

子程序是更一般的，称为“共行程序”的程序组成部分的特殊情况。如果说，主程序与子程序之间的关系是非对称的，那么，相反地，在共行程序之间却有着完全对称的关系，它们是彼此调用的

为了理解共行程序的概念，让我们来考察有关子程序的另一种思路。在上一小节中所持的观点是，一个子程序仅仅是计算机硬件的一个扩充，是为了节省代码行而引进的。这可以是正确的，但是另一种观点也是可能的：我们可以把主程序和子程序看成是一个程序

的班子，这个班子中的每一个成员都担负着某项工作任务。主程序在完成其任务的过程中，将激活子程序；子程序实现它自己的功能而后激活主程序。我们可以把我们的想象加以延伸，进而确认，从子程序的观点来看，当它出口时，它是正在调用主程序；主程序又继续来执行其任务，然后“出口”到子程序。子程序行动，然后再次调用主程序。

这个有些牵强附会的观点，在实际上不可能区别哪一个是另一个的子程序的时候，就引出了共行程序的概念。假设我们有共行程序A和B；当我们正在对A进行程序设计时，我们可以把B当作我们的子程序，而当正在对B进行程序设计时，我们又可以把A当作我们的子程序。即是，在共行程序A中，利用指令“JMP B”来激活共行程序B。在共行程序B中，又再利用指令“JMP A”来激活共行程序A。每当一个共行程序被激活时，它就在其上次被停止处恢复执行它的程序。

例如，共行程序A和B是两个下棋的程序。我们可以把它们组合在一起，使得它们来彼此对弈。

对于MIX，共行程序之间的这样的链接，可以通过在程序中设置如下的四条指令来进行：

A	STJ	BX	B	STJ	AX	
AX	JMP	A1	BX	JMP	B1	(1)

这就需要有四个机器周期，控制每一条路线的转移。开始，AX和BX被置成跳到每个共行程序的开始位置A1和B1。假设我们于单元A1处首先启动共行程序A。当共行程序A，比方说，执行单元A2中的“JMP B”时，地址B中的指令就把rJ存入AX中，于是AX被置成“JMP A2+1”。BX中的指令把我们带到地址B1，在共行程序B开始它的执行后，它终将遇到，比方说是于地址B2中的，一条指令“JMP A”。我们把rJ存入BX中并跳到单元A2+1，继续执行共行程序A，直到它再次跳到B，B把J存入AX并跳到B2+1，等等。

程序-子程序的链接与共行程序-共行程序的链接之间，其实质性的区别，就如同通过研究上述例子所见到的那样，乃在于：子程序总是在它的开始处起始，即是说，在一个固定的位置处起始，而主程序或共行程序却总是在它上次终止处之后的位置处起始。

实际上，共行程序最自然地出现在它与输入和输出的算法相联系的时候。例如，假设共行程序A的任务是读卡片并对输入作某些变换，将其归约成一个条款的序列。另一个共行程序，我们称之为B，则实现对这些条款的进一步的处理，并印出答案；B将周期性地调用由A建立的逐个的输入条款。于是，每当需要下一个输入条款时，共行程序B就跳到A去；而每当一个输入条款已经建立，共行程序A就跳到B去。读者可以说：“很好，B是主程序，而A仅仅是为进行输入的一个子程序。”然而，当进程A非常复杂时，这就显得不大对头了；事实上，我们也可以把A想象成主程序，而把B想象成为进行输出的子程序，则上述描述仍然适用。当A和B两者都很复杂，而且每一个都在许多位置上调用另一个时，在这两个极端的中间，共行程序的思想就显得有用了。要找出简短的共行程序的例子，来说明这一思想的重要性，是稍有困难的；最有用的共行程序的应用，一般地，都十分长。

为了研究共行程序的“动作”，让我们来考虑一个“凑成的”例子。假设我们要来写一个把一种代码翻译成另一种代码的程序。需要翻译的输入代码是一个以句点结尾的字母数

字字符序列，例如，

A 2 B 5 E 3426FGOZYW321OPQ89 R　　(2)

已经在卡片上穿了孔；对于这些卡片上的空白列，予以忽略。这个输入要如下地自左到右来理解：如果下一字符是一个数字（即是，0，1，…，9），比方说是 n，则它表示后一个字符的（$n+1$）个重复，不论这后一个字符是一个数字与否。非数字简单地就表示它本身。我们程序的输出是由以这种方式规定的序列组成的，并且分成为每三个字符一组（其中最后一组可能少于三个字符）。例如，通过我们的程序，(2) 应当翻译成

ABB　BEE　FEE　E44　446　66F　GZY　W22　220　OPQ　999　999　999　R　　(3)

注意 3426F 并非指的是字母 F 的 3427 个重复；它指的是 4 个 4 和 3 个 6，后边接上 F。我们的程序是要把输出穿孔到卡片上，每张卡片穿 16 组，每组由三个字符组成。

为实现这一翻译，我们将写出两个共行程序和一个子程序。设计称为 NEXTCHAR 的子程序，来逐个地找输入的非空白字符，并把下一个字符放进寄存器 A 中：

```
01    *SUBROUTINE   FOR    CHARACTER          INPUT
02    READER        EQU    16                 卡片输入机设备号
03    INPUT         ORIG   *+16               输入卡片的位置
04    NEXTCHAR      STJ    9F                 子程序入口
05                  JXNZ   3F                 开始 rX = 0
06    1H            J6N    2F                 开始 rI6 = 0
07                  IN     INPUT(READER)      读下张卡片
08                  JBUS   *(READER)          等待完成
09                  ENN6   16                 让 rI6 指向头一个字
10    2H            LDX    INPUT-16,6         取下一个输入字
11                  INC6   1                  推进指示器
12    3H            ENTA   0
13                  SLAX   1                  下一个字符→rA
14    9H            JANZ   *                  跳过空白
15                  JMP    NEXTCHAR+1         ■
```

本子程序有如下的特征：

调用序列：JMP　NEXTCHAR

进入条件：rI6　指向下一个字，或者 rI6＝0 表示必须读一张新卡片；rX＝有待使用的字符。

出口条件：rA＝输入的下一个非空白字符；rX，rI6 针对 NEXTCHAR 的下一次进入而设置。

我们的头一个共行程序，叫做 IN，要找出输入代码的字符并进行相应的处置（即适当地复写）：

```
16    *FIRST    COROUTINE
17    2H        INCA     30          找到非数字
18              JMP      OUT         把它送到 OUT 共行程序
19    IN1       JMP      NEXTCHAR    取字符
20              DECA     30
21              JAN      2B          是一字母?
```

```
22              CMPA    =10=
23              JGE     2B              是一特殊字符?
24              STA     *+1(0:2)        找到数字 n
25              ENT5    *               rI5←n
26              JMP     NEXTCHAR        取下一个字符
27              JMP     OUT             把它送到 OUT 共行程序
28              DEC5    1               n 减 1
29              J5NN    *-2             必要时重复
30              JMP     IN1             重新开始新的循环
```

(回忆一下，在 MIX 的字符代码中，数字 0-9 有代码 30-39)。这个共行程序有以下特征：

调用序列：JMP IN

出口条件(当跳到 OUT 时)：rA＝经相应处置的输入的下一字符； rI4 保留入口时的值不变。

入口条件(返回时)：rA，rX，rI5，rI6 应保留它们在上次出口时的值不变。

另一个共行程序，叫做 OUT，它把这代码弄成三个一组，并穿孔到卡片上：

```
31    *SECOND   COROUTINE
32              ALF                             留空用的常数
33    OUTPUT    ORIG    *+16                    答复所用的缓冲区
34    PUNCH     EQU     17                      卡片穿孔机设备号
35    OUT1      ENT4    -16                     开始新的输出卡片
36              ENT1    OUTPUT
37              MOVE    -3,1(16)                把输出区弄成空白
38    1H        JMP     IN                      取下一个翻译了的字符
39              STA     OUTPUT+16,4(1:1)        存入输出
40              CMPA    PERIOD                  是句号"."?
41              JE      9F
42              JMP     IN                      若否，则取另一个字符
43              STA     OUTPUT+16,4(2:2)        存入它
44              CMPA    PERIOD                  是句点"."?
45              JE      9F
46              JMP     IN                      若否，则取另一个字符
47              STA     OUTPUT+16,4(3:3)        存入它
48              CMPA    PERIOD                  是句点"."?
49              JE      9F
50              INC4    1                       送到输出的下一个字
51              J4N     1B                      卡片结束
52    9H        OUT     OUTPUT(PUNCH)           若是，则穿孔
53              JBUS    *(PUNCH)                等待完成
54              JNE     OUT1                    为剩下的而返回，除非已经
                                                传送"."了
55              HLT
56    PERIOD    ALF
```

这个共行程序，有如下的特征：

调用序列：JMP OUT

出口条件（当跳到 IN 时）：rA，rX，rI5，rI6 保留它们在进入时的值不变；rI1 可能要受影响；以前的字符记录在输出中。

进入条件（返回时）：rA＝经相应处置的下一个输入字符，rI4 保留它在上次出口时的值不变。

为了完成整个程序，我们需要来编写共行程序的链接（参照（1））并提供适当的初始化。共行程序的初始化需要一点技巧，虽说并不十分困难。

```
57    *INITIALIZATION   AND    LINKAGE
58    START             ENT6   0          为 NEXTCHAR 给 rI6 初值
59                      ENTX   0          为 NEXTCHAR 给 rX 初值
60                      JMP    OUT1       由 OUT 启动（参照习题 2）
61    OUT               STJ    INX        共行程序链接
62    OUTX              JMP    OUT1
63    IN                STJ    OUTX
64    INX               JMP    IN1
65                      END    START
```

程序至此就完成了。读者应该仔细地研究它，特别是注意每个共行程序，怎样能在把另一个共行程序想象成它的子程序的情况下，独立地编写出来。

IN 和 OUT 共行程序的入口和出口条件，在上述程序中，是完全啮合的。一般地说，我们并不都是这样幸运，而且共行程序的链接也将包括装入和存储适当的寄存器。例如，如果 OUT 将破坏寄存器 A 的内容，则共行程序的链接将变成

```
OUT     STJ    INX
        STA    HOLDA      当离开 IN 时存储 A
OUTX    JMP    OUT1
IN      STJ    OUTX                                  (4)
        LDA    HOLDA      当离开 OUT 时恢复 A
INX     JMP    IN1
```

在共行程序和多趟扫描的算法之间，有着重要的关系。例如，我们刚才描述的翻译过程可以在不同的两趟中来完成：我们首先要进行的只是 IN 共行程序，我们把它应用于整个的输入并且以适当数量的复写而把每个字符写到磁带上。在完成之后，我们将重绕磁带，然后只是进行 OUT 共行程序，三个一组地从磁带上取字符。这就叫做“两趟”的进程（直观地说，“趟”乃是指的对输入的一次完整的扫描。这个定义是不精确的，而且在许多算法中，其所取的趟数并不十分明显；然而，虽说趟的直观概念有其含糊性，但它还是有用的）。

图 22（a）说明了一个四趟的进程。我们将十分经常地发现，这同一进程只要在一趟中就能完成，如图中的（b）部分所示，只要我们以共行程序 A，B，C，D 来代替分别的趟 A，B，C，D。当趟 A 已经往带 1 上写完了一个输出条款时，共行程序 A 将跳到 B；当趟 B 已经从带 1 读完了一个输入条款时，共行程序 B 将跳到 A，而且当趟 B 已经往带 2 上写完了一个输出条款款时，共行程序 B 将跳到 C；等等。

反之，一个由 n 个共行程序完成的进程，通常都能被转换成一个 n 趟的进程。由于有

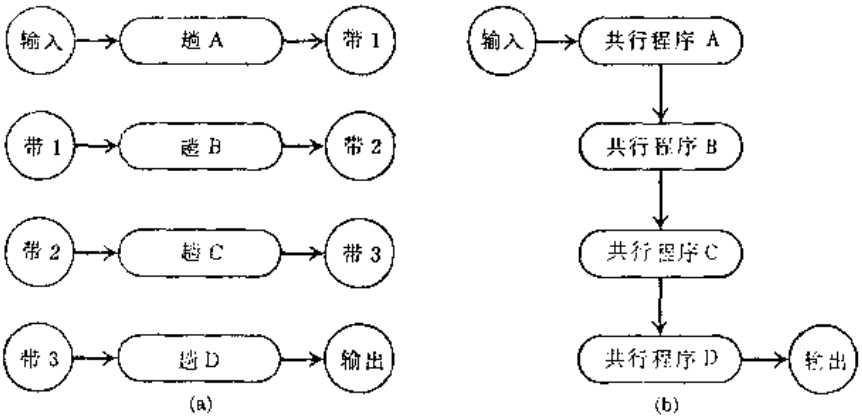


图22 趟数：（a）一个四趟的算法；（b）一个一趟的算法。

这一对应关系，因此就值得对多趟算法与一趟算法作一比较。

a）心理上的区别 对同一问题，多趟算法一般地比单趟算法更易于建立和理解。把一个进程分割成为一系列相继发生的小的步骤，是比考虑把所有这些事情同时地一齐包罗进来的一个进程要更易于理解的。

而且，如果正在从事的是非常大的问题，还有，如果是许多人协作来生产这个计算机程序，则多趟算法就提供了一种把工作进行分工的自然的方式。

多趟算法的这些优点，在共行程序中也仍然存在，因为共行程序实质上可以相互独立地来写出，而链接却使得一个表面上多趟的算法合成一个单趟的算法。

b）时间上的区别 在诸趟之间为组装、写、读和卸装中间数据（例如，图 22 中诸带上的信息）所需要的时间，在单趟的算法中就可以加以避免。因此之故，单趟算法将是更快的。

c）空间上的区别 单趟算法需要有为同时地把全部程序都存入内存中的空间，而多趟算法则仅需要为进行一次扫描所需要的空间。这就可能影响速度，甚至于比在陈述（b）中所指出的时间还要大些。例如，许多计算机都有一个有限数量的“快速存储”和一个更大数量的慢速存储；如果每趟都能存入快速存储之中，则结果就将比我们在单趟中使用共行程序要快得多（因为，使用共行程序，大概将迫使大多数程序都出现于慢速存储之中）。

偶而地，也需要一次就对多种计算机配置设计出算法来，而这些计算机配置中的某些比其它的有更大的内存容量。在这种情况下，将可能这样来写这个程序，就是，借助于共行程序，而且让存储的容量支配趟数：尽可能多地一起装入许多个共行程序，而对于那些失却的链接则提供输入或输出子程序。

虽然共行程序与趟数之间的这种关系是重要的，但我们应该牢记，并非所有的共行程序的应用都能被分割成多趟的算法。例如，如果共行程序 B 从 A 获得输入，而且向 A 发回决定性的信息，则共行程序 B 就不能转换成趟 B 跟在趟 A 之后。

反之，显然，并非所有的多趟算法都能转换成共行程序。某些算法本性是多趟的；例如，第二趟可能需要从第一趟获得累积的信息（如输入中某个字出现的总数）。关于这一点，有一桩老笑话值得提及：

一位老太太，乘坐公共汽车。她问：“小孩，到帕萨迪纳大街时，提醒我一下好吗？”

小孩：“只要你看着我，并且在我下车前两站下车即可”。

（这个笑话就是，这个小孩给出了一个两趟的算法）。

关于多趟算法就谈这些吧。我们将在本书中的许多地方见到关于共行程序的进一步的例子，例如，象 1.4.4 小节中的缓冲方案部分。在离散系统的模拟中，共行程序也起着重要的作用；见 2.2.5 小节。在第 8 章中讨论了复写共行程序的重要概念，在第 10 章中还可以找到这个概念的某些有趣的应用。

习题

1.〔10〕 简略说明为什么教科书的作者很难举出共行程序的简单例子。

▶2.〔20〕 正文中的程序首先启动 OUT 共行程序。如果说 IN 要首先执行，即是，如果行 60 由 "JMP OUT1" 改成为 "JMP IN1"，则将发生什么情况？

3.〔20〕 是真还是假：OUT 中的三条 "CMPA PERIOD" 指令都可以省略，而程序仍能工作（要小心考察）。

4.〔20〕 试对于你所熟悉的若干实际的计算机，说明可以怎样给出类似于（1）的共行程序链接。

5.〔15〕 假设共行程序 IN 和 OUT 都要求寄存器 A 的内容在出口和入口之间不改动；于是，假设无论指令 "JMP IN" 出现于 OUT 之内的什么地方，当控制返回到下一行时，寄存器 A 的内容都不改变，而且对于 IN 之内的 "JMP OUT" 也作同样的假定。试问需要什么样的共行程序链接？〔参照(4)〕

▶6.〔22〕 对于三个共行程序 A，B，C 的情形，给出类似于（1）的共行程序链接，这三个共行程序的每一个都要能跳到其它两个中任一个。（每当一个共行程序被激活时，它就在它上次停止处开始。）

▶7.〔30〕 写出一个程序，它把正文中的程序所进行的翻译逆转过来，即是说，它将把类似（3）的穿孔卡片转换成为类似（2）的穿孔卡片。输出应该是一个尽可能地短的字符串，使得（2）中 Z 之前的 O 实际上不由（3）产生。

1.4.3 解释性程序

在这一小节里，我们将研究一种通常类型的计算机程序，即解释性程序（简称解释程序）。解释性程序是这样一种计算机程序，它实现另一个程序的指令，而这另一个程序是以某种类似机器语言写成的。所谓类似机器语言，我们指的是，比如说有操作码、地址、等等表示指令的某种方式。（这个定义，象大多数当今的计算机术语的定义一样，是不精确的，也不是它所应有的；根本不可能精确地说出哪一个程序正好是解释程序，而哪一个程序则不是，其分界线究竟在何处）。

历史上，头一批解释程序是围绕着这样的要求而构造出来的，即是，为了简单地进行程序设计，而专门设计的类似机器语言；这种类似机器语言，将是比机器语言更便于使用的一种语言。程序设计语言的出现，已经使得解释程序的这一功能逐渐地过时，但这并不意味着解释程序灭迹了。相反地，它们的使用已经发展了，以至达到了这样的程度，即是，可以认为解释程序的有效利用，乃是现代程序设计的实质性特征之一。解释程序的新的应用，主要是由下列原因引起的：

a）为了以精致的有效的方式来表示一系列相当复杂的判断和动作；

b）为了在多趟程序的趟与趟之间进行通讯。

在这些情况下，为用于一个具体的程序，就创立了专用的类似机器语言，而且这一机器经常是一直以这种语言写“程序”的唯一个体（当今熟练的程序员，也都是优秀的机器设计师，因为他们不仅建立了解释程序，而且也定义了其语言将被解释的虚拟机器）。

解释性技术还有进一步的优点，这就是相对地独立于机器——当机器改变时，仅须重写解释程序。而且，容易把便于调试的手段构造到一个解释系统中去。

类型（a）的解释程序的一些例子，出现在本书后边的若干处。例如，第8章中的递归解释程序，第10章的“分析机器”解释程序，以及第12章的XMIX解释程序。

典型的例子是一个这样的程序，其中出现有许许多多的特殊情况，其间全都类似，但却没有果真单纯的式样。例如，考虑写一个编译程序（参见第12章），在这个编译程序中，我们希望去生成为把两个量加到一起的高效的目标程序。可能有10种类型的量（例如，常数，简单变量，临时存储，下标变量，在一个累加器或变址寄存器中的量，定点数和浮点数，等等），而且因此总共可得到100种不同的配对组合。为了去做每一情况下的不同的事情，就需要一个长长的程序。对于这个问题的解释性答案，是组成一种语言，其“指令”填于一个字节中。然后，保存100个这种语言的“程序”的一张表，其中每个程序由一到五条指令组成，因而填于一个字中。其想法不过就是挑出表中适当的条目并执行在该处找到的程序。这一技巧是简单而有效的。

类型（b）的解释程序的一个例子，出现于唐·欧·克努特所写的论文《计算机画框图》(Computer Drawn Flowcharts)，CACM 6(1963)，555-563中。在多趟程序中，较早的趟必须向较后的趟传送信息。这个信息，通常以一种稍微类似于机器语言的语言，作为对于较后的趟的一组指令，而最有效地传输。因此较后的趟只不过是一个专用的解释程序，而较早的趟是一个专用的“编译程序”。这个多趟操作的原理，其特征乃在于：每当可能时，告诉较后的趟要做什么，而不是简单地向它介绍大量的事实并要求它去想出要做什么。

类型（b）的解释程序的另一个例子，其出现与专用语言的编译程序有关。如果一种语言包括许多特性，这些特性在机器上是不易实现的，除非通过子程序来做，则得到的目标程序将是非常长的子程序调用序列。例如，如果这种语言主要是涉及多精度算术运算，便将出现上述情况。在这样的情况下，如果目标程序以一种解释性语言来表达时，则它将大大缩短。这种方法的一个说明可在第12章中找到，在那里讨论了TROL语言和它的解释程序。也见于布·兰德尔（B. Randell）和劳·约·拉塞尔（L. J. Russell）所著的书《ALGOL 60的实现》(ALGOL 60 Implementation)（纽约：科学出版社，1964），书中描述了一个把ALGOL 60翻译成一个解释性语言的编译程序，而且也描述了对于这个语言的解释程序。还可以看小阿瑟·埃文斯(Arthur Evans Jr.)所作《ALGOL 60的一个编译程序》(An ALGOL 60 Compiler)，Ann. Rev. Auto Programming 4 (1964)，87-124，作为在一个编译程序内部用的解释程序的例子。微程序设计的机器的出现，已经使得这个解释的方法甚至更有价值。

观察一个以解释性语言写成的程序，还有另一种观点——它可以被认为是一个接着一个的一系列子程序调用。这样一个程序事实上可以展开成一个长长的对于子程序的调用序

列，而且反过来，这样一个序列通常都能被集装成一种易于解释的代码形式。解释性技术的优点，是表示的紧凑性，与机器的无关性，以及诊断能力的增强。通常，一个解释程序能够写成：使得花在解释代码本身并转移到适当程序的时间可忽略不计。

1.4.3.1 MIX 模拟程序 一个解释程序通常称作一个模拟程序，如果提供给这个解释程序的语言是另一台机器的机器语言。

依作者的意见，程序员花费在写这样的模拟程序上的时间以及计算机花费在使用它们上的时间，实在是太多了。搞模拟程序的动机是简单的●：一部计算机装置添置了一台新的机器，而且仍然想运行为旧的机器编制的程序（而不想去重写这些程序）。但是，这比起临时求用程序员专业队伍来重新进行程序设计来，通常都是，其所花的代价甚大，而其所得的结果甚差。例如，作者曾经一度参加过这样的一个重新进行程序设计的计划，而且还在原来的程序中，发现了一个严重的错误，虽说是原来的程序已经使用若干年了。新的程序除换成了正确的答案之外，还以五倍于旧的程序的速度来进行工作！（并非所有的模拟程序都是要不得的。例如，在造出一台新的机器之前，计算机厂家来模拟它通常是有利的，以便这台新机器的软件可以尽快地研制出来。但这只是非常特殊的应用。）计算机模拟程序低效率的使用，有一个极端的例子，就是实际上有这样的故事：机器 A 模拟机器 B，而机器 B 则运行一个模拟机器 C 的程序！这种搞法导致了：一台庞大的昂贵的计算机，却给出比它便宜得多的同类计算机要差的结果。

考虑到所有这些，那为什么还让这样一个模拟程序，在本书中崭露头角呢？有两个原因：

a）我们下边将要描述的模拟程序，是典型的解释程序的一个好例子。这里说明了应用于解释程序中的基本技巧。它也说明了子程序在一个适度长的程序中的用法。

b）我们将描述一个（特别地）以 MIX 语言写成的 MIX 计算机的模拟程序。这将便于对大多数类似的计算机来编写 MIX 模拟程序。我们程序的编码有意地避免大量使用面向 MIX 的特色。MIX 的模拟程序，作为同本书以及还可能同其它的书相配合的教学辅助工具，将是有益的。

如本小节所述的这样的计算机模拟程序，应当与离散系统的模拟程序区别开来。后者是将在 2.2.5 小节研究的重要的程序。

现在让我们转到编写 MIX 模拟程序的任务上来。MIX 的指令 LDA，LD1，…，LDX 的编号，以及一些其它类似的编号，提示我们如下地以连续的单元来保存这些寄存器的模拟内容：AREG，I1REG，I2REG，I3REG，I4REG，I5REG，I6REG，XREG，JREG，ZERO。这里 ZERO 是一个始终填以 0 的“寄存器”。JREG 和 ZERO 的位置是受了指令 STJ 和 STZ 的操作码号的提示。

记住我们写模拟程序的原则，是把它想作不是用 MIX 来处置的。因此，我们将把符号位处理作一个字的独立部分。例如，许多计算机不能表示数“负 0”，而 MIX 则肯定能表示。所以，我们在这个程序中将总是特殊地处理符号。单元 AREG，I1REG，…，ZERO 将总是包含相应寄存器内容的绝对值；在我们程序里的另一组单元，叫做 SIGNA，SIGN1，

● 现代的计算机已在某种程度上较好地解决了这种程序“兼容性”问题，主要通过利用微程序技术来仿真，但仍然是以解释性技术为基础。——译者注

…，SIGNZ，将视相应的寄存器的符号为正还是为负，而包含＋1或－1。

一个解释程序一般都有一个中心的控制部分，来调度在所释的指令之间的动作。在这种情况下，当每条所模拟的指令结束时，程序即转移到单元CYCLE。

控制程序完成对所有的指令都统一的工作，把指令分划成它的各个部分，并把这些部分放到方便的位置以便以后使用。以下的程序置

rI6＝下一条指令的单元；

rI5＝M（现行指令的地址，加变址）；

rJ4＝现行指令的操作码●；

rI3＝现行指令的F场；

INST＝现行指令。

程序 M

```
01    *MIX      SIMULATOR
02              ORIG    3500          (模拟的存储从地址 0000 开始)
03    BEGIN     STZ     TIME(0:2)
04              STZ     OVTOG         OVTOG 是模拟的溢出开关
05              STZ     COMPI         COMPI，±1 或 0，是比较指示器
06              ENT6    0             从单元 0 取头一条指令
07    CYCLE     LDA     CLOCK         控制程序的头
08    TIME      INCA    0             这个地址置成前一条指令的执行时间，见33行
09              STA     CLOCK
10              LDA     0，6          模拟指令→rA
11              STA     INST
12              INC6    1             推进单元计数器
13              LDX     INST(1:2)     取地址的绝对值
14              SLAX    5             把符号附加到地址上
15              STA     M
16              LD2     INST(3:3)     检查变址场
17              J2Z     1F            它为 0？
18              DEC2    6
19              J2P     INDEXERROR    说明的是非法的变址？
20              LDA     SIGN6，2      取变址寄存器的符号
21              LDX     I6REG，2      取变址寄存器的数值
22              SLAX    5             附加符号
23              ADD     M             为变址作带符号加法
24              CMPA    ZERO(1:3)     结果太大？
25              JNE     ADDRERROR
26              STA     M             地址已经找到
27    1H        LD3     INST(4:4)     F 场→rI3
28              LD5     M             M→rI5
29              LD4     INST(5:5)     C 场→rI4
30              DEC4    63
```

● 实为 C-63。——译者注

```
31          J4P     OPERROR          操作码≥64?
32          LDA     OPTABLE,4(4:4)   从表中取执行时间
33          STA     TIME(0:2)
34          LD2     OPTABLE,4(0:2)   取真正程序的地址
35          JNOV    0, 2             跳到操作符
36          JMP     0, 2             (防止溢出)
```

请读者特别注意行 34-36; 64 个操作符的“开关表”是模拟程序的一部分;它考虑到当前的指令而迅速地跳到正确的程序。这是一个重要的节省时间的技巧(参看习题1.3.2-9)。

称为 OPTABLE 的 64 字的开关表,也给出了对于各种操作符的执行时间;以下诸行指出了该表的内容。

```
37          NOP     CYCLE(1)         操作符表:
38          ADD     ADD(2)           典型的表目是
39          SUB     SUB(2)           “操作码  地址  (时间)”
40          MUL     MUL(10)
41          DIV     DIV(12)
42          HLT     SPEC(1)
43          SLA     SHIFT(2)
44          MOVE    MOVE(1)
45          LDA     LOAD(2)
46          LD1     LOAD, 1(2)
            ...
51          LD6     LOAD, 1(2)
52          LDX     LOAD(2)
53          LDAN    LOADN(2)
54          LD1N    LOADN, 1(2)
            ...
60          LDXN    LOADN(2)
61          STA     STORE(2)
            ...
69          STJ     STORE(2)
70          STZ     STORE(2)
71          JBUS    JBUS(1)
72          IOC     IOC(1)
73          IN      IN(1)
74          OUT     OUT(1)
75          JRED    JRED(1)
76          JMP     JUMP(1)
77          JAP     REGJUMP(1)
            ...
84          JXP     REGJUMP(1)
85          INCA    ADDROP(1)
86          INC1    ADDROP, 1(1)
            ...
```

```
 92              INCX   ADDROP(1)
 93              CMPA   COMPARE(2)
                 ...
100    OPTABLE  CMPX   COMPARE(2)
```

(对于操作符 LDi，LDiN，以及 INCi 的表目，有一个附加的“,1”，以置 (3:3) 场为非 0。这在下面的行 289-290 中，用来指出这样一个事实，即在相应的变址寄存器之中的数量的大小，在模拟这些操作之后，必须加以校验)。

我们的模拟程序的下一部分，就是列出那些用来包含所模拟的寄存器的内容的单元：

```
101    AREG     CON    0          A 寄存器的数值
102    I1REG    CON    0          变址寄存器的数值
103    I2REG    CON    0
104    I3REG    CON    0
105    I4REG    CON    0
106    I5REG    CON    0
107    I6REG    CON    0
108    XREG     CON    0          X 寄存器的数值
109    JREG     CON    0          J 寄存器的数值
110    ZERO     CON    0          常数 0，为了“STZ”
111    SIGNA    CON    1          A 寄存器的符号
112    SIGN1    CON    1          变址寄存器的符号
113    SIGN2    CON    1
114    SIGN3    CON    1
115    SIGN4    CON    1
116    SIGN5    CON    1
117    SIGN6    CON    1
118    SIGNX    CON    1          X 寄存器的符号
119    SIGNJ    CON    1          J 寄存器的符号
120    SIGNZ    CON    1          为“STZ”存储的符号
121    INST     CON    0          正在模拟的指令
122    COMPI    CON    0          比较指示器
123    OVTOG    CON    0          溢出开关
124    CLOCK    CON    0          模拟的执行时间
```

现在我们将考虑为模拟程序所使用的各种子程序。首先是 MEMORY 子程序：

调用序列：JMP MEMORY

进入条件：rI5＝有效的存储地址（否则子程序将跳到 MEMERROR)。

出口条件：rX＝存储单元（rI5）中字的符号；rA＝存储单元（rI5）中字的数值。

```
125    *SUBROUTINES
126    MEMORY   STJ    9F         从存储器取数的子程序
127             J5N    MEMERROR
128             CMP5   =BEGIN=    所模拟的存储是在单元 0000 到 BEGIN
129             JGE    MEMERROR
130             LDX    0,5
```

```
131             ENTA    1
132             SRAX    5               字的符号→rX
133             LDA     0,5(1:5)        字的数值→rA
134     9H      JMP     *               出口 ■
```

FCHECK 子程序处理部分场说明，以确保它具有 8L＋R 的形式，其中 L≤R≤5。

调用序列：JMP FCHECK

进入条件：rI3 是有效的场说明（否则子程序将跳到 FERROR）。

出口条件：rA＝rI1＝L，rX＝R。

```
135     FCHECK  STJ     9F              场校验子程序
136             ENTA    0
137             ENTX    0,3             rAX←场说明
138             DIV     =8=             分成 L 和 R
139             CMPX    =5=             R＞5？
140             JG      FERROR
141             STX     R
142             STA     L
143             LDI     L               rI1←L
144             CMPA    R
145     9H      JLE     *               出口，除非 L＞R
146             JMP     FERROR          ■
```

最后一个子程序 GETV，求用于各种 MIX 运算符的量 V，如同第 1.3.1 小节所定义的（即是，单元M的适当的场）。

调用序列：JMP GETV

进入条件：rI5＝有效的存储器地址；

rI3＝有效的场。

出口条件：rA＝V 的数值；rX＝V 的符号；rI1＝L；rI2＝－R。

第二个入口：JMP GETAV，仅用于比较操作符，以从一寄存器抽取一个场。

```
147     GETAV   STJ     9F              特殊入口，见行 300
148             JMP     1F
149     GETV    STJ     9F              求 V 的子程序
150             JMP     FCHECK          处理场；L→rI1
151             JMP     MEMORY          rA←存储器的数值，
                                        rX←符号
152     1H      J1Z     2F              符号是场的部分？
153             ENTX    1               若否，则置符号为正
154             SLA     -1,1            摘去场左边的字节
155             SRA     -1,1
156     2H      LD2N    R               右移到适当的位置
157             SRA     5,2
158     9H      JMP     *               出口 ■
```

现在我们进入针对各个操作符的程序。在这里给出这些程序是为了完整性，而读者应

当仅仅研究其中的少量几个，除非你特别有信心；建议把对于SUB和JUMP的那些程序，作为研究的典型例子。注意对于类似的操作的程序，是如何巧妙地组合起来的，并注意JUMP程序如何使用另一个开关表，以支配各类转移。

```
159     *INDIVIDUAL     OPERATORS
160     ADD     JMP     GETV        取V的值到rA，rX中
161             ENT1    0           命rI1指A寄存器
162             JMP     INC         转到"增量"程序
163     SUB     JMP     GETV        取V的值到rA，rX中
164             ENT1    0           命rI1指A寄存器
165             JMP     DEC         转到"减量"程序
166     *
167     MUL     JMP     GETV        取V的值到rA，rX中
168             CMPX    SIGNA       符号相同?
169             ENTX    1
170             JE      *+2         置rX为结果符号
171             ENNX    1
172             STX     SIGNA       把它放进两个模拟的寄存器
173             STX     SIGNX
174             MUL     AREG        乘操作数
175             JMP     STOREAX     存数值
176     *
177     DIV     LDA     SIGNA       置余数符号
178             STA     SIGNX
179             JMP     GETV        取V的值到rA，rX中
180             CMPX    SIGNA       符号相同?
181             ENTX    1
182             JE      *+2         置rX为结果符号
183             ENNX    1
184             STX     SIGNA       把它放入模拟的rA中
185             STA     TEMP
186             LDA     AREG        除操作数
187             LDX     XREG
188             DIV     TEMP
189     STOREAX STA     AREG        存数值
190             STX     XREG
191     OVCHECK JNOV    CYCLE       刚才出现过溢出?
192             ENTX    1           若是，则接通模拟的溢出开关
193             STX     OVTOG
194             JMP     CYCLE       返回控制程序
195     *
196     LOADN   JMP     GETV        取V的值到rA，rX中
197             ENT1    47, 4       rI1←C-16；指寄存器
198     LOADN1  STX     TEMP        符号取反
199             LDXN    TEMP
200             JMP     LOAD1       把LOADN变为LOAD
201     LOAD    JMP     GETV        取V的值到rA，rX中
202             ENT1    55, 4       rI1←C-8，指寄存器
```

```
203     LOAD1   STA     AREG, 1         存数值
204             STX     SIGNA, 1        存符号
205             JMP     SIZECHECK       校验数值太大否
206     *
207     STORE   JMP     FCHECK          rI1←L
208             JMP     MEMORY          取存储单元的内容
209             J1P     1F              符号是场的部分?
210             ENT1    1               如是，则把L变成1
211             LDX     SIGNA+39, 4         并“存”寄存器的符号
212     1H      LD2N    R               rI2←－R
213             SRAX    5, 2            保存场右边的区域
214             LDA     AREG+39, 4      把寄存器插入场中
215             SLAX    5, 2
216             ENN2    0, 1            rI2←－L
217             SRAX    6, 2
218             LDA     0, 5            恢复场左边的区域
219             SRA     6, 2
220             SRAX    -1, 1           附加符号
221             STX     0, 5            存入存储器
222             JMP     CYCLE           返回控制程序
223     *
224     JUMP    DEC3    9               转移操作符
225             J3P     FERROR          F太大?
226             LDA     COMPI           比较指示器→rA
227             JMP     JTABLE, 3       转到适当的程序
228     JMP     ST6     JREG            置模拟的J寄存器
229             JMP     JSJ
230             JMP     JOV
231             JMP     JNOV
232             JMP     LS
233             JMP     EQ
234             JMP     GR
235             JMP     GE
236             JMP     NE
237     JTABLE  JMP     LE              转移表
238     JOV     LDX     OVTOG           检验是否溢出转移
239             JMP     *+3
240     JNOV    LDX     OVTOG
241             DECX    1               取溢出开关之补
242             STZ     OVTOG           断开溢出开关
243             JXNZ    JMP             转移
244             JMP     CYCLE           不转移
245     LE      JAZ     JMP             若rA为0或负则转移
246     LS      JAN     JMP             若rA为负则转移
247             JMP     CYCLE           不转移
248     NE      JAN     JMP             若rA为负或正则转移
249     GR      JAP     JMP             若rA为正则转移
250             JMP     CYCLE           不转移
```

```
251    GE          JAP     JMP              若 rA 为正或 0 则转移
252    EQ          JAZ     JMP              若 rA 为 0 则转移
253                JMP     CYCLE            不转移
254    JSJ         JMP     MEMORY           校验有效的存储地址
255                ENT6    0, 5             模拟一个转移
256                JMP     CYCLE            返回主控制程序
257    *
258    REGJUMP     LDA     AREG+23,4        寄存器转移
259                JAZ     *+2              寄存器为 0?
260                LDA     SIGNA+23,4       若否，把符号放入 rA
261                DEC3    5
262                J3NP    JTABLE,3         变成有条件的 JMP,除非 F 说明太大
263                JMP     FERROR
264    *
265    ADDOP       DEC3    3                地址转移操作符
266                J3P     FERROR           F 太大?
267                ENTX    0,5
268                JXNZ    *+2              求M的符号
269                LDX     INST
270                ENTA    1
271                SRAX    5                rX = M 的符号
272                LDA     M(1:5)           rA = M 的数值
273                ENT1    15,4             rI1 指寄存器
274                JMP     1F,3             四路的转移
275                JMP     INC              增量
276                JMP     DEC              减量
277                JMP     LOAD1            入口
278    1H          JMP     LOADN1           负的入口
279    DEC         STX     TEMP             符号取反
280                LDXN    TEMP             变为"增量"
281    INC         CMPX    SIGNA,1          加法程序
282                JE      1F               符号相同?
283                SUB     AREG,1           否: 数值相减
284                JANP    2F               寄存器中符号变化?
285                STX     SIGNA,1          改变寄存器符号
286                JMP     2F
287    1H          ADD     AREG,1           数值相加
288    2H          STA     AREG,1(1:5)      存结果的数值
289    SIZECHECK   LD1     OPTABLE,4(3:3)   我们刚装入一变址寄存器?
290                J1Z     OVCHECK
291                CMPA    ZERO(1:3)        若是，则确保结果填入两个字节
292                JE      CYCLE
293                JMP     SIZEERROR
294    *
295    COMPARE     JMP     GETV             取 V 的值到 rA,rX 中
296                SRAX    5                附加符号
297                STX     V
298                LDA     XREG,4           取适当寄存器的场
```

```
299              LDX     SIGNX,4
300              JMP     GETAV
301              SRAX    5               附加符号
302              CMPX    V               比较(注意 -0 = +0)
303              STZ     COMPI           把比较指示器置成或0，或正1，或负1
304              JE      CYCLE
305              ENTA    1
306              JG      *+2
307              ENNA    1
308              STA     COMPI
309              JMP     CYCLE           返回控制程序
310    *
311              END     BEGIN
```

上面的代码坚持了在第1.3.1小节已经陈述过的颇为微妙的一个规则：指令“ENTA −0”把负0装入寄存器A中，就象“ENTA −5，1”当变址寄存器1包含+5时所作的那样。一般说来，当M为0时，ENTA就装入本指令的符号，而ENNA则装入相反的符号。在作者准备第1.3.1小节的初稿时，竟未能注意到说明这一条件的必要；这样的问题，仅当计算机程序是遵循这些规则来编制时，才显示出来。

尽管上述程序很长，但它在若干方面还是不完全的：

a）它不能识别浮点运算。

b）对于操作码5，6和7的编码，已留作习题。

c）对于输入输出操作符的编码，已留作习题。

d）没有提供被模拟程序的装入（见习题4）。

e）未包括出错程序

INDEXERROR, ADDRERROR, OPERROR, MEMERROR, FERROR, SIZEERROR,

这些都是有待处理的在模拟程序中发现的出错条件。

f）没有提供诊断设施（例如，随着程序之执行，印出寄存器的内容）。

习题

1.〔14〕 研究FCHECK子程序在模拟程序中的所有用法。你能提出组织这个程序中的子程序的更好方式吗？（参照1.4.1小节末尾的讨论中的步骤3。）

2.〔20〕 写出SHIFT程序，在正文里它从程序中略去了（操作码6）。

▶3.〔22〕 写出MOVE程序，在正文里它从程序中略去了（操作码7）。

4.〔14〕 改变正文里的这个程序，使得当MIX的“GO按钮”被按下时，它开始运行（参照习题1.3.1-26）。

▶5.〔24〕 确定为模拟LDA和ENTA操作符所需要的时间，并与MIX直接地执行这些操作的实际时间进行比较。

6.〔28〕 写出对于输入输出操作符JBUS，IOC，IN，OUT和JRED的程序，在正

文里它们从程序中略去了；但仅允许设备 16 和 18。假定输入一张卡片或跳到新的一页需要 10000 u，而打印则需 7500 u（注意：经验证明，应该通过把“JBUS *”处理成一种特殊情况来模拟 JBUS 指令；否则模拟程序似乎要停止）。

▶7.〔32〕 以这样一种办法来修改上题的解，即 IN 或 OUT 的执行不立即引起输入/输出的传输；传输应该在模拟的设备所需要的时间已经过去近一半之后才发生（这将防止学生不适当地使用 IN 和 OUT 操作码而经常出现错误）。

8.〔20〕 是真还是假：每当执行模拟程序的行 10 时，我们总有 $0 \leqslant$ rI6 $<$ BEGIN。

***1.4.3.2 跟踪程序** 当在一台机器上模拟它自身时（就象上一小节中，在 MIX 上模拟 MIX 那样），这是模拟程序的特殊情况，我们称之为跟踪或监督程序。有时使用这种程序来帮助调试，因为它印出被模拟的程序如何逐步地动作的清单。

上一小节的程序，虽是作为另一台计算机模拟 MIX 而写成的。但对于跟踪程序使用的却是截然不同的方法：我们一般地都让寄存器就表示它们本身，并且让操作符就实现它们本身。

在一个跟踪程序中，我们通常都要设法让机器执行大多数指令；例外的是转移指令或条件转移指令，它们不得被不加修改地执行（否则跟踪程序将失去控制）。每台机器还有其各自的特性，这些特性使跟踪程序面临更多的挑战；在 MIX 的情况下 这就是 J 寄存器。

以下给出的跟踪程序，当主程序转移到单元 ENTER，且寄存器 J 置成开始跟踪的地址，寄存器 X 置成停止跟踪的地址时，即启动运行。这个程序是有趣的，也是值得细心研究的。

```
01    *TRACE ROUTINE
02    ENTER   STX     TEST(0:2)       设置出口单元
03            STX     LEAVEX(0:2)
04            STA     AREG            保存 rA 的内容
05            STJ     JREG            保存 rJ 的内容
06            LDA     JREG(0:2)       取开始跟踪单元
07    CYCLE   STA     PREG(0:2)       存储下条指令的单元
08    TEST    DECA    *               它是出口单元吗?
09            JAZ     LEAVE
10    PREG    LDA     *               取下条指令
11            STA     INST            复写它
12            SRA     2
13            STA     INST1(0:3)      存地址和变址部分
14            LDA     INST(5:5)       取操作码 C
15            DECA    38
16            JANN    1F              C≥38(JRED)?
17            INCA    6
18            JANZ    2F              C≠32?
19            LDA     INST(0:4)       C = 32(STJ)
20            STA     *+2(0:4)        改变成 STA
21    JREG    LDA     *               外部的 rJ 的内容→rA
22            STA     *
```

```
23              JMP     INCP
24      2H      DECA    2
25              JANZ    2F              C≠34?
26              JMP     3F              C = 34(JBUS)
27      1H      DECA    9               测试转移指令
28              JAP     2F              C≥48?
29      3H      LDA     8F(0:3)         检测出转移指令
30              STA     INST(0:3)       它的地址变成"JUMP"
31      2H      LDA     AREG            恢复A寄存器
32      *                               除J之外，现在所有寄存器
33      *                               都有了关于外部程序是适当的值
34      INST    NOP     *               执行此指令
35              STA     AREG            再次存寄存器A
36      INCP    LDA     PREG(0:2)       移到下一条指令
37              INCA    1
38              JMP     CYCLE
39      8H      JSJ     JUMP            用于行29，40的常数
40      JUMP    LDA     8B(4:5)         已经出现一转移
41              SUB     INST(4:5)       它原是JSJ?
42              JAZ     *+4
43              LDA     PREG(0:2)       若否，则更新模拟的寄存器J
44              INCA    1
45              STA     JREG(0:2)
46      INST1   ENTA    *
47              JMP     CYCLE           移到这一指令
48      LEAVE   LDA     AREG            恢复A寄存器
49      LEAVEX  JMP     *               停止跟踪
50      AREG    CON     0               外部的rA的内容
```

关于一般的跟踪程序，特别是这个程序，有下列事情应该说明：

1）我们仅仅介绍了一个跟踪程序的最有趣的部分，即在执行另一个程序的同时保留控制的部分。要想使一个跟踪程序真正可用，还必须有一个用于写出寄存器的内容的程序，而这尚未被包含进来——因为尽管这样一个程序是重要的，但是它会扰乱一个跟踪程序最微妙的特性。必要的修改留作习题（见习题2）。

2）一般说来，空间是比时间更为重要的。就是说，程序应当写得尽可能地短。这样就使得跟踪程序能同大程序共存，而且运行的时间总是消耗于输出。

3）必须细心地避免破坏大多数寄存器的内容；事实上，这个程序仅使用了MIX的A寄存器。这跟踪程序既不影响比较指示器，也不影响溢出开关（我们所做的事情越少，则我们需要恢复的事情也就越少）。

4）当出现到单元JUMP的转移时，由于rA不可能已经变化，所以不需要来"STA AREG"。

5）在离开跟踪程序后，J寄存器并未真正地还原。习题1说明了怎样来补救这一点。

6）被跟踪的程序仅仅受到三个限制：（a）它必须不存任何东西到跟踪程序所使用的单元中去。（b）它必须不使用在其上记录跟踪信息的输出设备（例如，JBUS将给出一个出错的指示）。（c）当跟踪时，这个程序是以不同的速率来执行的。

习题

1.〔22〕 修改正文中的跟踪程序，使得当离开时，它恢复寄存器J（你可以假定寄存器J不为0）。

2.〔26〕 修改正文中的跟踪程序，使得在执行程序的每一步之前，它都在磁带设备0上写出如下的信息：

字1，(0:2) 场：单元。
字2：指令
字3：寄存器A（执行之前）
字4-9：寄存器I1-I6（执行之前）
字10：寄存器X（执行之前）
字1，(4:5) 场：寄存器J（执行之前）
字1，(3:3) 场：若比较是大于则为2，若比较是等于则为1，若是小于则为0；若溢出开关未接通则加8。
字11-100：9个在同样格式下的进一步的10字组。

3.〔10〕 上一道题提示我们：跟踪程序要把它的输出写到带上。试讨论为什么这比直接印出还好些？

▶4.〔25〕 如果跟踪程序跟踪它本身，则将发生什么情况？特别地，如果两条指令ENTX LEAVAX；JMP *+1就放在ENTER的紧前边，试考虑其行为。

5.〔28〕 以类似于解决上道题的方式，考虑这样一种情况，就是把跟踪程序的两个副本放在存储器的不同位置，而且每一个被用来跟踪另一个。试问这将发生什么情况？

▶6.〔40〕 试设计一个跟踪程序，它能在习题4的意义下，来跟踪它本身，即是，它将以较缓慢的速度印出它自己的程序的步骤，而且还以较缓慢的速度，无休止地跟踪它本身，直到超过了存储容量为止。

▶7.〔25〕 讨论怎样来编写一个高效的转移跟踪程序，它发出比一个通常的跟踪要少得多的输出。代替显示寄存器的内容，转移跟踪只简单地记录所出现的转移。它输出一系列的数偶 (x_1, y_1)，(x_2, y_2)，…，来表示这个程序从单元 x_1 转移到 y_1，然后（在执行了单元 y_1, y_1+1，…，x_2 中的指令之后）它从 x_2 转移到 y_2，等等〔利用这一信息，就有可能通过一个随后的程序来重新画出这个程序的流程，并且来推导每条指令被执行的频率情况〕。

1.4.4 输入和输出

也许，在一台计算机和另一台计算机之间，其最突出的差别在于：为进行输入和输出所可利用的设备，以及支配这些外部设备的计算机指令。我们不可能企望，在单单一部书中，就要讨论在这一领域中出现的所有问题和技术。所以，我们将把我们自己限制在，只

来研究应用于大多数计算机的典型的输入输出方法。MIX 的输入输出操作符，代表了实际的计算机中所应用的千差万别的设备之间的一种折衷。为了给出如何来考虑关于输入输出问题的一个例子，让我们在这一小节里，来讨论一下关于获得最好的 MIX 输入输出的问题。

许多计算机用户，都觉得输入输出实际上并不是“真正的程序设计”的一个组成部分；它们仅仅是为了得到机器内外的信息，而不得不（不幸地）加以完成的事项。因此在对所有其它的特性进行考察之前，通常总不会来学习一台计算机的输入和输出设备的；而且经常出现这样的情况：关于输入和输出的细节，仅仅是一台具体的计算机的一小部分程序员知道得较多些。这种局面是比较自然的，因为机器的输入输出设备总不是特别精巧的。然而，直到引起更多的人们对于这一课题进行严重的思虑之前，是不能期望来改进这一状况的。在这一小节以及别处（例如，第 5.4.6 小节），我们将会看到，某些非常有趣的事情是与输入输出相联系而出现的，而且其中还存在着某些有意思的算法哩！

在这里，也许宜于先给出一个关于术语的简单的约定。尽管现行的词典似乎都认为，词“输入”和“输出”仅仅是名词（例如，人们常说：“我们采用的是什么类型的输入？”），但现在，在文法习惯上，要把它们用作形容词（例如，我们常说：“不要抹去输入带。”），以至用作及物动词（例如，我们可以说：“为什么这一程序输出这个‘废料’？”）。组合的术语“输入-输出”最经常地通过简写“I/O（输入/输出）”来引用。输入通常叫做读，而类似地，输出叫做写。作为输入或输出的资料，一般地称作“数据（data）”——严格地说，这个词是数据（datum）的复数形式；但是它是集体地来使用的，就好象它是单数的一样（例如，我们常说：“这数据（data）还未读。”）。这段“语法课”就讲完了。

假设现在我们希望从磁带来读。MIX 的 IN 操作符，如 1.3.1 小节所定义的，仅仅起始输入的过程；而且计算机在进行输入的同时，继续执行进一步的指令。于是，指令“IN 1000（5）”将开始从设备号为 5 的磁带上读入 100 个字到存储单元 1000–1099，但是随后的程序直到这以后一段时间，不得去访问这些存储单元。只有在：（a）另一个访问设备5的输入/输出操作（IN, OUT 或 IOC）已经起始之后，或（b）条件转移指令 JBUS（5）或 JRED（5）指示设备 5 已不再“占线”之后，输入才算完成了。

因此，要把一个带区读进单元 1000–1099，并且要提供信息，其最简单的方式，是两个指令的序列

```
IN      1000(5)
JBUS    *(5)                                    (1)
```

我们在 1.4.2 小节的程序中，已经使用了这一初步的方法（见行 07–08 和行 52–53）。然而这个方法一般是浪费计算机时间的，因为大量潜在地有用的计算机时间，比如说 1000u 乃至 10000u，都被“JBUS”指令的重复执行所消耗了。如果这额外的时间被利用于计算的话，则程序的运行速度可能会增加好多好多倍（见习题 4 和 5）。

为了避免浪费这个计算时间，一个办法是要用两个存储区域以进行输入；我们可以读到一个区域中去，而且在进行这个输入的同时，程序可以由另一个区域的数据来进行计算。例如，假设程序由指令

```
IN      2000(5)          开始读头一个带区          (2)
```

开始。随后，每当需要一个带区时，我们现在就可以给出如下的五条指令：

```
ENT1     1000          为MOVE操作符作准备
JBUS     *(5)          等待直到设备5已准备就绪
MOVE     2000(50)      (2000-2049)→(1000-1049)          (3)
MOVE     2050(50)      (2050-2099)→(1050-1099)
IN       2000(5)       开始读下一个带区
```

这些，在总的效果上，与（1）是相同的。

在前一个带区检查完之前，这个程序就把一个带区读进单元2000-2099。这叫做“提前读”或抢先输入——这是在这样的基础上进行的，即坚信这个带区最终将需要。然而，事实上，我们则可能获悉（通过检查传送到1000-1099的带区的内容），并不真正需要那么多的输入。例如，考虑1.4.2小节的共行程序中的类似情况，其中输入来自穿孔卡片，而不是来自带；出现于卡片中任何地方的“·”都意味着，该卡片乃是这迭卡片的最后一张。这样一种情况将使得抢先输入已成为不可能，除非我们假定：或者是（a）一张空白卡片，或某种其它类型的特殊的结尾卡片，必须跟在这迭输入卡片之后；或者是（b）一个标识号（例如“·”）必须出现在这迭卡片最后一张上的第80列处。每当输入抢先进行时，总是必须在程序结尾处提供妥善地终止输入的某些手段。

使计算时间与输入/输出时间相重迭的技术，称作缓冲。初步的方法（1）称作“无缓冲的”输入。用来保存（3）中的抢先输入的存储区2000-2099，以及传送输入于其中的区域1000-1099，称作“缓冲区”。《韦氏新世界辞典》定义“缓冲”为“用来减少冲击的任何人或物”，这是很适当的，因为缓冲技术可以使输入/输出设备趋于平稳地运行。（计算机工程师经常在另外的意义下来使用“缓冲区”这个词，即表示输入/输出设备的一部分，这一部分是在传送时存储信息的；但在本书中，“缓冲区”将表示程序员用来保存输入/输出数据的存储器区域。）

序列（3）并不总比（1）优越，尽管例外情形是很少的。让我们来比较一下执行时间。假设T是为输入100个字所需要的时间，并假设C是介于输入请求之间的计算时间。方法（1）对每个带区实际上需要$T+C$的时间，而方法（3）则实际上需要$\max(C,T)+202u$（$202u$这个量是两条MOVE指令所需要的时间）。要考察这个运行时间，有一个办法，就是来考虑所谓的“临界通路时间”，在当前情况下，这就是输入/输出设备在使用之间的空闲时间数量。方法（1）使设备保持C个单位空闲时间，而方法（3）则使它保持202个单位的空闲时间（假设$C<T$）。

（3）中相当缓慢的MOVE指令是不受欢迎的，特别是因为，它们花费了磁带设备必须处于不活动时的临界通路时间。有一个几乎是显然的改进方法，允许我们来避免这些MOVE指令：外边的程序可以修改成，使得它交替地访问单元1000-1099和2000-2099。在读入一个缓冲区域的同时，我们可以以另一个缓冲区域中的信息来进行计算。这就是重要的称为缓冲区交换的技术。所关注的当前缓冲区的单元，将保存在一个变址寄存器中（或者，如果没有可利用的变址寄存器，则就保存在一个存储单元中）。我们已经看到过一个缓冲区交换的例子，那是应用于算法1.3.2P（见步骤P9-P11）和相随程序之输出的。

作为用于输入的缓冲区交换的一个例子，假设我们有一计算机应用，其中每个带区由

100个分别的一字条款所组成。下列的程序，是一个读取下一个输入字的子程序，而且如果当前的这个带区已经穷尽，则它就读入一个新的带区：

```
01    WORDIN   STJ    1F          存出口单元
02             ENC6   1           进到下一个字
03    2H       LDA    0,6         它是缓冲区的
04             CMPA   =SENTINEL=  末尾?                    (4)
05             JNE    *           若否，则出口
06             IN     -100,6(U)   重填这个缓冲区
07             LD6    1,6         取另一个缓冲区的
08             JMP    2B          地址并返回
```

在这个程序中，变址寄存器6用作输入的最后一个字的地址；我们假设外边的程序不影响这个寄存器。符号U指的是一个带设备；而符号SENTINEL● 指的是一个这样的值，它告知（按程序的特征）已经不在任何带区内了。与这个子程序相随的，是缓冲区的如下配置：

```
09    INBUF1   ORIG   *+100       头一个缓冲区
10             CON    SENTINEL    在缓冲区末尾的'SENTINEL'
11             CON    *+1         另一个缓冲区的地址
12    INBUF2   ORIG   *+100       第二个缓冲区
13             CON    SENTINEL    在缓冲区末尾的'SENTINEL'
14             CON    INBUF1      另一个缓冲区的地址
```

关于这个程序，有若干事项需要说明：

1）常数'SENTINEL'作为每个缓冲区的第101个字出现，而且它形成了对缓冲区末尾的方便的测试。然而，在许多应用中，这一技术将不是可靠的，因为任何字都可能出现在带上。如果我们是在进行卡片输入，则总可以使用类似的技术（使用置缓冲区的第17个字等于SENTINEL）；在这种情况下，任何负的字都可以作为一个SENTINEL，因为从卡片输入的总是非负的字。

2）每个缓冲区都含有另一个缓冲区的地址（见行07，11和14）。这种“共同链接”方便了交换过程。

3）不需要“JBUS”指令，因为在前一带区的任何字被存取之前，就开始下一个输入了。如果量C和T，如前所述，指的是计算时间和带的时间，则每个带区的执行时间现在是$\max(C, T)$；因此如果$C \leq T$，则就有可能使带以全速运行。（注意：然而MIX在这方面是一台理想化了的计算机，因为本程序不必去处理输入/输出错误。对于大多数计算机，在这里的“IN”指令紧前边，都有必要有某些检测前一操作是否已成功地完成的指令。）

4）为使这个子程序正确地工作，当程序开始时，就必须占有俾能顺当地着手的一切。其细节留给读者处理（见习题6）。

5）WORDIN子程序使得所涉及的带设备，仿佛是一个区段长度为1，而不是象程序的其余部分那样长度为100的设备。把若干个面向程序的记录填入到一单个实际的带区中

● SENTINEL（哨兵），意为一带区越界之检测者。——译者注

的思想，叫做“记录的分区”。

我们对于输入来说明的技术，只要稍加改变，也同样地可应用于输出（见习题2和3）。

多缓冲区 缓冲区交换只是使用N个缓冲区之一般方法于$N = 2$的特殊情形。在某些应用中，希望要有2个以上的缓冲区；例如，考虑下述类型的算法：

步骤1. 快速地接连读5个带区。

步骤2. 基于这个数据，进行相当长的计算。

步骤3. 返回步骤1。

这里将希望有5个或6个缓冲区，以便在进行步骤2时，把下一轮的五个区段读入。由于输入/输出活动的这种“被集拢的”趋势，就促使缓冲区交换进而发展成为多缓冲区。对于那样一些设备（例如某些尤尼瓦克卡片输入机），也需要多缓冲区的技术，即其中一IN指令起始输入的过程中，若干另外的（起始后来的区段输入过程的）IN，在头一个区段可以利用之前就可给出。这样，为了能在一个合理的速度下有任何机会来运行输入设备，多缓冲区就成为必要的了。

假设我们有某个输入或输出过程，它有N个缓冲区，但只使用一个输入/输出设备。我们就把这N个缓冲区想象成是排在一个圆圈上，如图23所示。就这一输入/输出设备而论，我们将假定缓冲过程外部的程序，有如下形式：

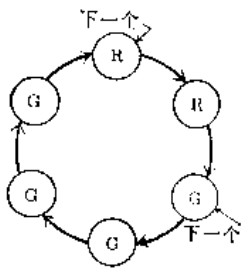


图23 缓冲区的一个圆圈
($N = 6$)

```
⋮
指定
⋮
释放
⋮
指定
⋮
释放
⋮
```

即是，用其它的计算，把“指定”和“释放”的交替的动作分隔开来，而这其它的计算是不影响对于这个设备的缓冲区之处理的。

指定 意味着程序获得下一个缓冲区域的地址，即是，这个地址被指定为某个程序变量的值。

释放 意味着程序已不再与当前的缓冲区域有关系了。

在指定与释放之间，程序正通过诸缓冲区中，称作当前缓冲区域者，来进行通讯；在释放与指定之间，程序不访向任何的缓冲区域。

可以想象，指定将立即跟随释放，而且对于缓冲技术的讨论，通常都在这个假定的基础上来进行。然而，如果尽可能快地完成释放，则缓冲的过程就有更大的自由度，并且将是更有效的；通过把两个实际上不同的功能（指定和释放）分隔开来，我们就会觉得缓冲技术更易于理解了；而且我们的讨论甚至对于$N = 1$仍将是有意义的。

为了更明了些，让我们分别考虑输入和输出的情况。对于输入，假设我们正在处理一台卡片输入机。指定的动作，意味着程序需要从一张新的卡片获得信息；我们将乐于把一个变址寄存器置成存储器地址，在这个地址处，放置了下一张卡片的映象。释放的动作出现于：当不再需要当前卡片映象中的信息时——它已经在某处为程序所消化，也许已被复

写到存储器的另一部分，等等。从而，当前的缓冲区域可以填以进一步的抢先输入。

对于输出，考虑一打印机的情形。当需要一个空的缓冲区域，以便在其中放置一个欲打印之行的映象时，就出现指定动作。我们希望把一个变址寄存器置成等于这样一个区域的存储器地址。当这个行的映象已经以准备好打印的形式，在缓冲区域中完全建立时，就出现释放动作。

例：为打印单元 0800–0823 的内容，我们可以写

```
JMP     ASSIGNP          (设r5为缓冲区单元)
ENT1    0,5
MOVE    800(24)          把24个字传送到缓冲区
JMP     RELEASEP
```
(5)

其中 ASSIGNP（指定P）和 RELEASEP（释放P）表示为打印机（P）完成这两个缓冲功能的子程序。

在最优的情况下（从计算机的观点来看），指定操作实际上将不需要执行时间。在输入时，这将意味着，每个卡片映象已经抢先输入，因此当程序一经准备就绪要使用数据时，数据即可利用；而在输出时，它将意味着，在存储器中总是有一空的位置来记录行的映象。将不用花费等候输入/输出设备的时间。

为了帮助描述缓冲技术的算法，使它更加精采，我们将说缓冲区域或是“绿色的”，或是“黄色的”，或是“红色的”（图24 中示为G，Y 和R）。

绿色G意味着这个区域已准备好来加以指定的；这意味着它已经以抢先的信息填入（在一输入的情况下），或者意味着它是一个空的区域（在一输出的情况下）。

黄色Y意味着这个区域已被指定，而且尚未释放；这意味着它是当前的缓冲区，而且程序正在与之进行通讯。

红色R意味着这个区域已经释放；因此它是一个空的区域（在一输入的情况下），或者它已经以信息填入（在一输出的情况下）。

图 23 说明了两个与缓冲区圆圈相关联的“指针”。概念上，这些都是程序中的变址寄存器。NEXTG 和 NEXTR 分别指向“下一个绿色G”和“下一个红色R”缓冲区。当其出现时，第三个指针 CURRENT（图 24 中所示）指黄色Y缓冲区。

尽管算法应用于输出也同样好，但为了确定起见，我们将首先考虑从一个卡片输入机进行输入的情况。假设一个程序已经达到如图 23 中所示的状态。这意味着，四个卡片映象已经通过缓冲过程抢先输入，而且它们驻留在绿色G缓冲区中。这时，就同时发生两件事情：（a）程序紧跟着一个释放操作而进行计算；（b）一张卡片正在读入由 NEXTR 所指的缓冲区。这种事态将一直继续到该输入周期结束时为止（设备然后将从“占线”变成“就绪”），或者一直到程序执行一个指定操作为止。假设后一种情况出现，则由 NEXTG所指的缓冲区变为黄色Y（它被指定为当前的缓冲区），NEXTG 沿顺时针方向移动，而且我们到达图 24（a）中所示的状态。如果现在输入已完成，则出现另一个抢先的区段，因此缓冲区由红色R变成绿色G，而且 NEXTR 前移，如图 24（b）中所示。如果再后的是释放操作，则我们得到图 24（c）。

作为关于输出的例子，见习题 9 中的图 27。那里的说明，把缓冲区域的“颜色”看成

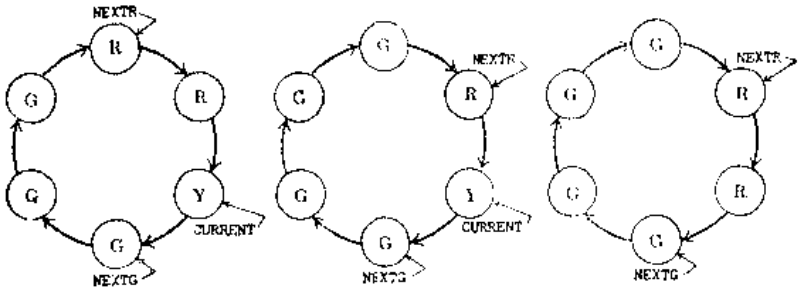


图24 缓冲区的变迁

(a) 在指定之后；(b) 在输入/输出完成之后；(c) 在释放之后。

是在一个程序中时间的函数，这个程序从四个快速的输出开始，然后以缓慢的速度产生四个输出，而且最后以快速连续地发生两个输出作为程序的结束。在这个例子中出现了三个缓冲区。

指针 NEXTR 和NEXTG 兴高采烈地围绕圆圈前进，每一个以独立的速度顺时针方向移动。这是一场在程序（它把缓冲区从绿色G转变成红色R）与输入/输出缓冲过程(它把缓冲区从红色R转变成绿色G）之间进行的速率竞赛●。可能出现两种冲突的情况：

a）如果 NEXTG 试图越过 NEXTR，则程序已经跑在输入/输出设备前边，而且程序必须等待直到设备准备就绪为止。

b）如果 NEXTR 试图越过 NEXTG，则输入/输出设备已经跑在程序前边，而且我们必须把设备停止下来，直到程序给出下一个释放为止。这两种情况，都已在图 27 中作了描述（见习题 9）。

幸而，尽管刚才对于缓冲区圆圈之后的思想，所给出的说明显得略微长些，但为处理这种情况的算法，却倒是非常简单的。在下列的描述中，

$$N=\text{缓冲区的总个数} \qquad (6)$$
$$n=\text{红色的缓冲区的当前个数}$$

在以下的算法中，变量n用来避免 NEXTG 与 NEXTR 之间的相互干扰。

算法 A（指定动作） 如上所述，本算法列出了在一个计算的程序之内，指定意味着哪些步骤。

A1.〔等待$n<N$〕 如果$n=N$，则暂停程序直到$n<N$为止(如果$n=N$，则没有准备就绪而可供指定的缓冲区；但下面的平行于本算法而运行的算法 B，将终于成功地产生一个绿色G缓冲区)。

A2.〔CURRENT←NEXTG〕置 CURRENT←NEXTG（从而指定当前的缓冲区)。

A3.〔推进 NEXTG〕把 NEXTG 推进到顺时针方向的下一个缓冲区。 ∎

算法 R（释放动作） 如上所述，本算法列出了在一个计算的程序之内，释放意味着哪些步骤。

R1.〔增大n〕 n加 1。 ∎

算法B〔缓冲区控制〕 这一算法实现机器中的输入/输出操作的实际起始。在下述的

● 其意为：程序之运行推动 NEXTG 前进，输入/输出缓冲过程促使 NEXTR 前移，随着程序和缓冲过程“兴高采烈”地运行，就形成了 NEXTG 和 NEXTR 这两个指针竞走的局面。——译者注

意义下，它是与主程序“同时地”执行的。

B1.〔计算〕 让主程序在一个短时间周期里进行计算；在延迟某一时间之后，即延迟到输入/输出设备已经为另一个操作准备就绪时，来执行步骤 B2。

B2.〔$n=0$？〕 若 $n=0$，则转 B1（于是，如果没有红色 R 缓冲区，则就没有可执行的输入/输出动作）。

B3.〔起始输入/输出〕 起始由NEXTR 所指示的缓冲区域与输入/输出设备之间的传送。

B4.〔计算〕 让主程序运行一个时间周期；然后当输入/输出操作已经完成时，则转到步骤 B5。

B5.〔推进 NEXTR〕 把 NEXTR 推进到顺时针方向的下一个缓冲区。

B6.〔减小 n〕 n 减 1，并转到 B2。 ∎

在这些算法中，我们有两个“同时地”进行的独立进程：即缓冲控制程序和计算程序。事实上，这些是共行程序，我们将称之为 CONTROL 和 COMPUTE。共行程序 CONTROL 在步骤 B1 和 B4 转移到 COMPUTE；共行程序 COMPUTE 则通过把“准备好转移”指令，散布在其程序中散开的区间处，而转移到 CONTROL。

这个算法，要对 MIX 进行编码是极为简单的。为方便起见，假设诸缓冲区是这样夹链接的，即是，在每个缓冲区之前的那个字是下一个缓冲区的地址；亦即，如果 $N=3$，则

内容（BUF1-1）=BUF2，

内容（BUF2-1）=BUF3，以及

内容（BUF3-1）=BUF1。

其中 BUF1，BUF2，BUF3 分别表示缓冲区 1，缓冲区 2，缓冲区 3（的起始地址）。

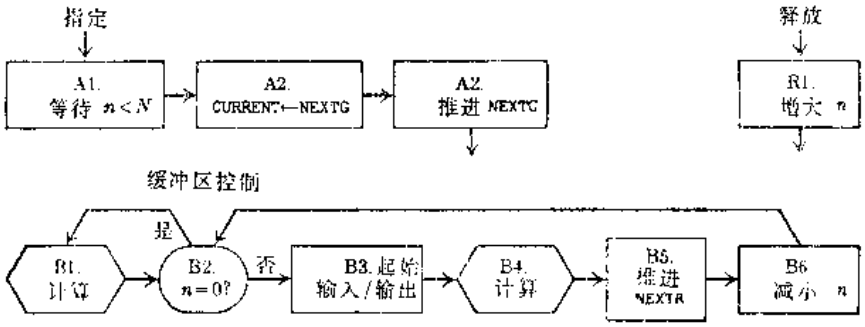


图25 对于多缓冲区的算法

程序 A（指定，在COMPUTE 共行程序之内的一个子程序）。rI4≡CURRENT；rI6≡n；调用序列是 JMP ASSIGN；出口时，rX 包含 NEXTG。

```
ASSIGN  STJ     9F              子程序链接
1H      JRED    CONTROL(U)      A1. 等待 n<N
        CMP6    =N=
        JE      1B
        LD1     NEXTG           A2. CURRENT←NEXTG
        LDX     -1,4            A3. 推进NEXTG
        STX     NEXTG
9H      JMP     *               出口  ∎
```

荣

程序 R（释放，用于 COMPUTE 共行程序之内的代码）。rI6≡ n。在每一个希望释放的地方，即插进这一简短的代码。

```
        INC6    1               R1. 增大 n
        JRED    CONTROL(U)      可能的到 CONTROL 共行程序的转移
```

程序 B（CONTROL 共行程序）。rI6≡ n，rI5≡NEXTR。

```
CONT1   JMP     COMPUTE         B1. 计算
1H      I62     *-1             B2. n = 0?
        IN      0,5(U)          B3. 起始输入/输出
        JMP     COMPUTE         B4. 计算
        LD5     -1,5            B5. 推进 NEXTR
        DEC6    1               B6. 减小 n
        JMP     1B
```

除上述代码之外，我们还有通常的共行程序链接

```
CONTROL   STS   COMPUTEX        COMPUTE   STS   CONTROLX
CONTROLX  JMP   CONT1           COMPUTEX  JMP   COMP1
```

而且，在 COMPUTE 之内，大约在每 50 条指令中放一条指令"JRED CONTROL(U)"。

这样，对于多缓冲技术的程序，实质上对于 CONTROL 程序总计仅七条指令，对于 ASSIGN 八条，对于 RELEASE 两条。

也许值得注意的是，对于输入和输出这两者，恰恰是同一个算法在进行工作。区别究竟是什么——控制程序怎么能知道是抢先（对于输入说来）还是落后（对输出说来)?答案就在于初始条件：对于输入，我们以 $n = N$ 来启动（即所有缓冲区都为红色 R)；而对于输出，我们则以 $n = 0$（所有缓冲区皆为绿色 G)来启动。一旦进程已经适当地启动，它就分别地，或者作为输入或者作为输出而继续动作。另一个起始条件是 NEXTR=NEXTG，两者都指向同一个缓冲区。

当程序完结时，输入/输出进程必须停止（如果它是输入）或者一直等到它完成(对于输出)；细节留给读者自己考虑（见习题 12 和 13)。

重要的是问，用什么 N 值最好。肯定地，当 N 取增大时，程序的速度将不会减小，但也并不是任何一方都无限地增加，因而我们会达到一个递减报酬的境界。让我们再次来参照量 C 和 T，这是表示输入/输出操作之间的计算时间和输入/输出时间本身的两个量。更确切地说，命 C 是在相继的指定之间的时间，并命 T 是为传送一个区段所需要的时间。如果 C 总是大于 T，则 $N = 2$ 是可取的，因为不难看出，通过两个缓冲区，我们就使得计算机总保持占线。如果 C 总是小于 T，则 $N = 2$ 还是可取的，因为我们总是使输入/输出设备保持占线。因此，仅当 C 在很小的值与很大的值之间变动时，较大的 N 值才是有用的。如果 C 的值远远地大于 T，则对于 N，取值为 1 再加上 C 相继出现的很小值平均个数，可能就更合适些（然而，如果所有的输入都出现在程序的开始处，以及所有的输出都出现在程序的末尾处，则缓冲技术的优点，实际上也就化为乌有了)。如果指定与释放之间的时间总是十分小，则在整个上边的讨论中，N 的值都可以减 1，这对运行时间影响甚微。

上边关于缓冲技术的方法，可以从许多方面来采用，而且我们将简略地提及少许。到现在为止，我们都是假定，正在使用的只是一个输入/输出设备；实际上，当然要同时使用

好些个。

对于解决多个设备的课题，有许许多多途径。在最简单的情况下，对于每一设备，我们都可以有各自的缓冲区圆圈。对于每一设备，都将有各自的 n，N，NEXTR，NEXTG 和 CURRENT 的值，以及不同的 CONTROL 共行程序。这将对每个输入/输出设备，同时地给出有效的缓冲动作。

也有可能，把相同大小的缓冲区“联营”成一个缓冲“池”，即是，有两个或更多的设备按一张公共的表来共享诸缓冲区。这将通过使用第 2 章的链接存储器的技术来处理：所有的红色输入缓冲区和绿色输出缓冲区将全被链接在一起。在这种情况下，就有必要来区别输入和输出，并且重写算法而不使用 n 和 N 了。如果池里的缓冲区中，全都填以抢先的输入，则这算法就可能不可改变地固执在一个方面。因此，应该进行一个这样的检验：即任何时刻至少要有一个缓冲区（对于每一个设备有一个就更好）不是绿色输入的；仅当对某个输入设备，在步骤 A1 处，COMPUTE 程序被暂停时，我们才允许从这个设备，对池里的最后一个缓冲区进行输入。

某些机器对于输入输出设备的使用还有另外的限制，使得不可能在同一时间从某些成对的设备来传送数据（例如，借助于单个“通道”，使若干设备都可附加到计算机上）。这个限制也影响我们的缓冲程序；当我们必须选择首先起动某一个输入/输出设备时，怎样来作这种选择呢？这就是所谓的“预报”。一般情况下最好的预报规则，似乎是选取其缓冲区圆圈有最大 n/N 值的设备，假定圆圈中的缓冲区个数已明智地选定时。

为了结束这个讨论，我们再来谈谈，在某些条件下，按同一个缓冲区圆圈进行输入和输出的一个有用的方法。在图 26 中，我们增加了另一个颜色的（紫色 P）缓冲区。在这种情况下，绿色 G 缓冲区表示抢先的输入；程序 ASSIGN（指定）一个绿色 G 缓冲区并使它变成黄色 Y，然后至 RELEASE（释放）时它转为红色 R 而表示一个有待输出的区段。这输入和输出的进程，就和前述的一样，独立地围绕着圆圈进行，所不同的是，现在我们在输出完成之后，是把红色 R 缓冲区转化成紫色 P，而在输入时则把紫色 P 转换成绿色 Y。必须保证指针 NEXTG，NEXTR，NEXTP 没有一个越过另一个。在图 26 中所示的情况下，程序正在 ASSIGN 和 RELEASE 之间进行计算，用黄色 Y 缓冲区；同时，输入正进入由 NEXTP 指出的缓冲区；而输出正来自由 NEXTR 指出的缓冲区。

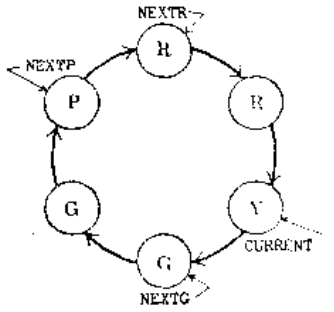


图26 按同一个圆圈进行输入和输出

习题

1.〔05〕 如果把 MOVE 指令放在 JBUS 指令之前而不是放在之后，则序列（3）是否仍将是正确的？如果把 MOVE 指令放在 IN 指令之后，则又将怎样呢？

2.〔10〕 指令

```
OUT     1000(6)
JBUS    *(6)
```

可以用来以无缓冲的方式输出一个带区，恰如指令（1）对于输入所做的那样。通过使用

MOVE 指令和在单元 2000-2099 中的一个辅助缓冲区，给出类似于（2）和（3）的，对这个输出进行缓冲的一个方法。

▶3.〔22〕 写出类似于（4）的一个缓冲区交换的输出子程序。这个子程序，称为 WORDOUT，将把字存到 rA 中来作为输出的下一个字；而如果一个缓冲区已满，则它就应当把 100 个字写到带设备 V 上。变址寄存器 5 应当用来指示当前的缓冲区位置。在把任何字存入一个缓冲区之前，应将其清为 0。说明缓冲区域的设置，并说明在程序的开始和结尾处，需要有什么指令（如果有的话），以保证头一个和最后的区段能被妥善地写出。

4.〔*M*20〕 证明：如果一个程序访问单个的输入/输出设备，则通过实行输入/输出缓冲，在顺利的情况下，有可能加倍运行的速度；但是不可能把运行速度提高到大于无缓冲输入/输出所花时间总量的 1/2 以上。

▶5.〔*M*21〕 把上题的情况推广到程序访问 n 个输入/输出设备，而不是仅仅访问一个的情形。

6.〔12〕 在一个程序的开始处应该放置什么指令，使得 WORDIN 子程序（4）能顺当地开始起动？（例如，变址寄存器 6 必须置成某种内容）。

7.〔22〕 写出一个实质上类似于（4）的称作 WORDIN 的子程序，但它不利用 "SENTINEL（哨兵）"。

8.〔11〕 正文中描述了一个假设的输入情况，这一情况是从图 23 直到图 24 的部分（*a*），（*b*）和（*c*）导出的。假定对打印机的输出正在进行，而不是从卡片进行输入，试解释这同一情况。（例如，在图 23 中所示的时间里，正发生什么事情？）

▶9.〔21〕 导致图 27 中所示的缓冲区内容的一个程序，可用以下的时间表来表征：

A, 1000, *R*, 1000, *A*, 1000, *R*, 1000, *A*, 1000, *R*, 1000, *A*, 1000, *R*, 1000

A, 7000, *R*, 5000, *A*, 7000, *R*, 5000, *A*, 7000, *R*, 5000, *A*, 7000, *R*, 5000

A, 1000, *R*, 1000, *A*, 2000, *R*, 1000

这张表意味着"指定，计算 1000 *u*，释放，计算 1000 *u*，指定，…，计算 2000 *u*，释放，计算 1000 *u*。"给定的计算时间不包括计算机可能要等候输出设备赶上来的任何时间间隔

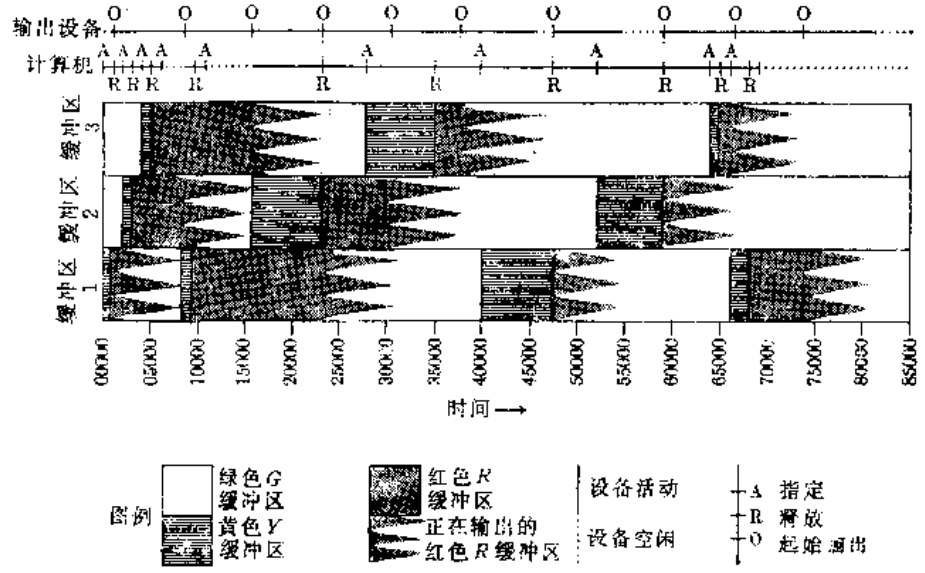


图27 具有三个缓冲区的输出（参照习题 9）

(如象图 27 中的第四个“指定”处)。输出设备以每个区段 7500 u 的速度进行操作。

下列图表说明了随着时间的推移，图 27 中所示的动作：

时间	动作	时间	动作	时间	动作
0	指定(缓冲区 1)	16000	缓冲区 2 被指定，输出缓冲区 3	54500	输出停止
1000	释放，输出缓冲区 1	23000	释放	59000	释放，输出缓冲区 2
2000	指定(缓冲区 2)	23500	输出缓冲区 1	61000	指定(缓冲区 3)
3000	释放	28000	指定(缓冲区 3)	65000	释放
4000	指定(缓冲区 3)	31000	输出缓冲区 2	66000	指定(缓冲区 1)
5000	释放	35000	释放	66500	输出缓冲区 3
6000	指定(等候)	38500	输出缓冲区 3	68000	释放
8500	缓冲区 1 被指定，输出缓冲区 2	40000	指定(缓冲区 1)	69000	计算停止
9500	释放	46000	输出停止	74000	输出缓冲区 1
10500	指定(等候)	47000	释放，输出缓冲区 1	81500	输出停止
		52000	指定(缓冲区 2)		

因此总共需要的时间是 81500 u；在 6000-8500，10500-16000，以及 69000-81500，或者说总共 20500 u，计算机处于空闲状态；输出设备在 0-1000，46000-47000，以及 54500-59000 处于空闲状态，或者说总共空闲了 6500 u。

若假定仅有两个缓冲区。试对于这同一程序，造出类似上边的“时间-动作”图表。

10.〔21〕 对于四个缓冲区的情形，重复习题 9。

11.〔21〕 对于仅一个缓冲区的情形，重复习题 9。

12.〔24〕 假设正文中的多缓冲区算法正用于卡片输入，而且假设，只要已经读到一张在 80 列有“·”的卡片，则输入就结束。试说明 CONTROL 共行程序（即是算法 B 和程序 B）应作如何改变，以使得输入能以这种方式停止下来。

13.〔20〕 如果缓冲算法正应用于输出，则为确保全部信息从缓冲区输出去，在正文中的COMPUTE 共行程序结尾处，应该包括些什么指令？

▶14.〔20〕 如果计算程序不在交替的指定和释放之间进行，而代之以在一系列的动作…指定…指定…释放…释放…之间进行，则情况将如何呢？这对于正文中所描述的算法，有什么影响？它还可能有用吗？

▶15.〔22〕 只利用三个缓冲区，试写出把 100 个带区从带设备 0 复写到带设备 1 的一个程序。这程序应是尽可能快的。

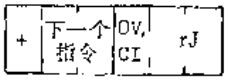
16.〔29〕 利用三个共行程序(一个用来控制输入设备，一个控制输出设备，以及一个计算的共行程序)，照正文中给出的多缓冲算法的方式，系统阐述图26中提出的“绿 G-黄 Y-红 R-紫 P”算法。

17.〔40〕 改编多缓冲算法，使其适应把缓冲区联营成“池”的要求；要在其中建立起使进程避免由于过多的抢先输入而减缓的方法。应力图使这算法尽可能地精巧。把你的方法与应用于现实生活问题的不联营的方法作一比较。

▶18.〔30〕 设计一个对 MIX 的修改方案，对其引进“中断能力”。这将如下所述地来进行。本题就是要来修改正文中的算法及程序 A，R，和 B，使得它们使用这些中断设施而

不是使用"JRED"指令。

新 MIX 的特色包括一片附加的 3999 个存储单元，单元－3999 到－0001。机器有两个内部状态——正常状态和控制状态。在正常状态下，单元－3999 直到－0001 是不允许的存储单元，而且 MIX 计算机就如通常那样动作。当由于后边所阐明的条件而出现一"中断"时，则单元－0009 直到－0001 即被置成等于 MIX 的寄存器的内容：rA 在－0009，rI1 到 rI6 在－0008 到－0003；rX 在－0002；而 rJ、溢出开关 OV、比较指示器 CI 以及下一指令的地址全都存入－0001 中，如下所示



进入控制状态，而且机器转移到一个依中断类型而定的单元。

单元－0010 起着一个"时钟"的作用：每 1000 u 的时间，出现于这个单元的数就减 1，而且如果结果为 0，则出现一个到单元－0011 的中断。

新的 MIX 指令"INT"（C＝5，F＝7）如下地进行工作：（a）在正常状态下，出现一个到单元－0012 的中断（这样，一个程序员就可以强行实施一个中断，以与一控制程序进行通讯；INT 的地址不起作用；尽管控制程序可以用它来作为区别中断类型的信息）。（b）在控制状态下，全部 MIX 寄存器复以－0009 到－0001 装入，计算机进入正常状态，并恢复执行。在每种情况下，对于 INT 的执行时间是 2 u。

在控制状态下给出的一条指令 IN，OUT，或 IOC 指令，一旦完成了输入/输出操作，就将引起一个中断出现。此中断转到单元－(0020－设备号)。

在控制状态下，禁止中断出现；任何中断条件都被一直"保留"到下一个 INT 操作之后，而且中断将出现于一条正常状态的指令已被执行之后。

▶19.〔37〕 某些计算机没有能力与计算同时地进行输入输出；输入/输出操作特使计算机一直等待到传输完成为止。这些计算机没有与 MIX 的 JBUS 或 JRED 操作符相等价的指令，因为设备总是"准备就绪"的；而且也没有如上题所给出的中断能力。然而，有时通过给出一条引起两个操作出现的"IN-IN"或"IN-OUT"或"OUT-OUT"指令（而且计算机要一直等候到两个操作都完成时），就有能力一次在两个不同的设备上来做输入/输出操作。

为了尽可能频繁地重叠输入/输出操作，在这种情况下仍可以利用多缓冲技术。例如，如果一个输出有待完成，则抢先输入即可同时地读入一个缓冲区中。

试建立对于这种情况的算法，假定程序使用两条输入带和一条输出带。应当有三个缓冲区圆圈，在头一个圆圈中有 N_1 个缓冲区，第二个中有 N_2 个，第三个中有 N_3 个。对于每个设备给出指定和释放的算法，使其尽可能地类似于本小节的算法 A，R 和 B。利用计算机模拟来测试你的算法。

1.4.5 历史和参考文献

1.4 节中所述的多数基本技术，是由许多不同的人独立地发展起来的；而这些思想的确切历史，可能根本就不知道。我们在这里，就是企图来记录历史上最重要的贡献，并给出正确的评价。

子程序是由一些程序员发明的，是减轻劳动的首要手段。在19世纪，查尔斯·巴贝格（Charles Babbage）对他的“分析机器”想象了一个程序库（参看《查尔斯·巴贝格和他的计算机器》（Charles Babbage and his Calculating Engines），菲利普（Philip）和埃米莉·莫里森（Emily Morrison）编（多佛，1961），56）；而且，我们可以说，当格雷斯·默·霍珀（Grace M. Hopper）于1944年在哈佛的马克I（Mark I）计算器上，写出了一个用于计算 $\sin x$ 的子程序时，他的梦想即变成了现实。然而，这实质上是一个“开子程序”，即是，在一个程序中每一需要处即嵌入之，而不是动态地链接的；这是因为，巴贝格所设想的机器（如同在雅克夸德（Jacquard）织布机上那样，由一系列穿孔卡片来控制）和马克I（由纸带来控制），与今天的存储程序式的计算机，是十分不同的。

适合于存储程序式的机器，并且带有以参数提供返回地址的子程序链接，是由赫尔曼·海·戈德斯坦和冯·诺依曼在他们关于程序设计的专著中提出的。这一专著写于1946年，发行广泛；见冯·诺依曼的《选集》（Collected Works）第5卷（纽约：麦克米伦，1963），215-235。他们程序中的主程序，是负责把一些参数存入子程序体，而不是把必要的信息传送到寄存器中。在英国，艾·马·图林（A. M. Turing）设计了计算机硬件，以便进行子程序链接；参看《第二次大型数字计算机会议文集》（Proceedings of a Second Symposium on Large-Scale Digital Calculating Machinery）（马萨诸塞，剑桥，哈佛大学，1949），89-90。头一本计算机程序设计的教程是《为电子数字计算机准备程序》（The Preparation of Programs for an Electronic Digital Computer）● 第一版，作者莫·文·威尔克斯（M. V. Wilkes），戴·约·惠勒（D. J. Wheeler）和斯·吉尔（S. Gill），（马萨诸塞，雷丁：爱迪生-韦斯利，1951）；其主要课题，就是极富才艺的子程序库的使用和构造。

“共行程序”一词是由梅·爱·康伟于1958年创造的；在他提出这一概念之后，他还首先将它应用于构造一个汇编程序。大约在同一时间，乔·厄尔德文（J. Erdwinn）和杰·默纳（J. Merner）也独立地研究了共行程序。他们写了一篇题为《双边链接》（Bilateral linkage）的论文，而他们却自认为不值得发表；遗憾的是，今天似乎已不存在这篇论文的复制品了。关于共行程序概念头一个公开的阐述，很晚才出现于康伟的文章《一个可独立的转换框图编译程序的设计》（Design of a Separable Transition-Diagram Compiler）CACM 6（1963），396-408中（实际上，共行程序链接的一个原始形式，在一份早期的尤尼瓦克出版物（The Programmer 1，2（1954年2月），4）中，作为一个“程序设计妙法”而被简要说明了）。在类似ALGOL的语言中，一个对于共行程序的适当记号，是在达尔（Dahl）和尼加德（Nygaard）的SIMULA I〔CACM 9（1966），671-678〕中引进的；而在奥—约·达尔，埃·怀·迪伊克斯特拉和查·安·里·霍尔合著的书《结构程序设计》（Structured Programming）第3章中，出现了许多精采的共行程序（包括琴键的共行程序）的例子。

可以说，头一个解释程序就是通用“图林机”，即一个有能力来模拟任何别的图林机的图林机（见第11章）。这些都不是实际的机器，它们是用来证明某些问题“不可解”的工具。通常意义下的解释程序，是由约翰·马赫里（John Mauchly）在他于1946年对莫尔（Moore）学校的讲演中提出的。早期最著名的解释程序，主要是想提供一个方便的进行浮

● 本书第二版（1957年）有中译本：《电子数字计算机的程序设计》安其春译，科学出版社，1962年。——译者注

点算术的工具，这就是用于沃尔温德 I（Whirlwind I）（由查·威·亚当斯（C. W. Adams）编制）以及用于伊利亚克（由戴·约·惠勒等人编制）的若干程序。图林也参与了这个研究；飞行者 ACE（Pilot ACE）计算机的解释系统是在他的指导下写出的。有关 50 年代初解释程序的情况，请看约·梅·贝内特（J. M. Bennett），D. G. 普林兹（D. G. Prinz）和 M. L. 伍斯（M. L. Woods）的论文《解释性子程序》（Interpretative Sub-routines），Proc. ACM Nat. Conf.（1952），81-87；也见于美国首都华盛顿海洋研究所出版的《数字计算机的自动程序设计会议文集》（Proceeding of the Symposium on Automatic Programming for Digital Computers）（1954）中的各篇文章。

最广泛应用的早期解释系统，大概是约翰巴克斯（John Backus）的《IBM 701 快速编码系统》（IBM 701 Speedcoding system）〔见 JACM 1（1954），4-6〕。这一系统，由贝尔电话实验室的维·迈·沃伦提斯（V. M. Wolontis）等人稍加修改。而非常巧妙地编写到 IBM 650 上；他们的程序，称作“贝尔解释系统”，极为流行。由艾·纽厄尔（A. Newell），约·克·肖（J. C. Shaw）及赫·阿·西蒙（H. A. Simon）于 1956 年设计的应用于十分不同的问题的 IPL 解释系统（见 2.6 节），也广泛地用作一个程序设计工具。解释程序的现代用法，如象在 1.4.3 小节的引论中所述，经常在计算机著作中顺便提到；有关稍微更详细地讨论解释程序的论文，请看这小节所列的参考文献。

头一个跟踪程序是由斯坦利·吉尔（Stanley Gill）于 1950 年编制的；请参看他在伦敦皇家学会会报（Proceedings of the Royal Society of London），Series A，206（1951 年 5 月），538-554 的有趣文章。上边提到的威尔克斯·惠勒和吉尔的教本，也列出了若干跟踪程序清单。其中最有趣的也许是戴·约·惠勒编的子程序 C-10，它包括一个设施用来在一个库子程序入口处压住跟踪、全速执行这库子程序、然后再继续跟踪。关于跟踪程序的公开的情报，在一般的计算机著作中是十分鲜见的；主要是因为，这个方法固有地是面向特殊机器的。就作者所知道的，仅有的其他早期文献是 H. V. 米克（H. V. Meek），《IBM 705 的实验监控程序》（An Experimental Monitoring Routine for the IBM 705），《西部计算机会议会报》（Proc. Western Joint Computer Conf.）（1956），68-70，它讨论了对于机器的一个跟踪程序，这机器上的这个问题是特别困难的。现今对于跟踪程序的重点，已经移到有选择的符号输出和程序性能的测定；见埃·萨特斯韦特（E. Satterthwaite），《软件的实践与经验》（Software-Practice and Experience）2（1972），197-217。

缓冲技术原来是由计算机硬件，以类似于代码 1.4.4-3 的方式进行的；其中程序员不能访问的内部“缓冲区域”，起着 2000-2099 单元的作用；而且当给出一条输入命令时，其中即执行了序列 1.4.4-3。自 40 年代后期以来，对于分类特别有用的特殊软件缓冲技术，已由早期的尤尼瓦克程序员加以发展了（见 5.5 节）。对于输入/输出的一般原理，在 1952 年有一篇很好的概述，见于这一年的《东部联合计算机会议会报》（Proceeding of the Eastern Joint Computer Conference）。

DYSEAC 计算机〔艾伦·L·莱因尼尔（Alan L. Leiner），JACM 1（1954），57-81〕引进了，在一个程序正在运行的同时，输入输出设备直接地同内存进行通讯。然后在完成时对程序进行中断的思想。这样一个系统，意味着已经发展了缓冲算法，但其细节未予公开。在我们所描述的意义下的缓冲技术的头一篇公开的文献，给出了一个高度巧妙的方法，

见于欧·莫克（O. Mock）和查·詹·斯威夫特（C. J. Swift)的《输入输出缓冲的程序设计》(Programmed Input-Output Buffering)，Proc. ACM. Nat. Conf. (1958) 19 页及 JACM 6 (1959)，145-151。(告诫读者，这些论文含有大量地方色采的专门术语，可能要费些时间才能了解；但 JACM 6 中邻近的论文将是有帮助的。)使输入输出的缓冲技术得以实现的中断系统，是由荷兰的埃·怀·迪伊克斯特拉于 1957 和 1959 年，就 B. J. 卢珀斯特拉（B. J. Loopstra）的X-1 计算机，独立地建立起来的〔参照 Comp. J. 2 (1959), 39-43〕。迪伊克斯特拉的博士论文《同一台计算机进行通讯》(Communication with an Automatic Computer)(1958，现已绝版)，提出了缓冲技术；在这种情况下，由于这些程序主要涉及纸带和打字机的输入/输出，因此它包含了很长的缓冲区圆圈；每个缓冲区包含或者一单个的字符或者一单个的数。他后来把这个思想发展成重要的信号灯的一般概念。这个概念不仅对于输入输出，而且对于所有种类的共行进程的控制，都是基本的〔见弗·吉奴伊斯（F. Genuys 编《程序设计语言》(Programming Languages)(美国：科学出版社，1968) 43-112；BIT (1968)，174-186；Acta Informatica 1 (1971)，115-138〕。戴维·埃·弗格森（David E. Ferguson）的论文《输入输出缓冲和 FORTRAN》(Input-Output Buffering and FORTRAN) JACM，(1960)，1-9，描述了缓冲区圆圈，并给出了一次有许多设备的简单缓冲技术的详细描述。

第2章 信息结构

2.1 引 论

计算机的程序，通常都是对一些信息表进行操作。在大多数情况下，这些表并不是无组织的数值集团；它们含有数据元素间的重要结构关系。

就其最简单的形式而言，一个表可以是一个线性的元素表，这时其有关的结构性质可以包括对诸如下列问题的回答：哪一个元素是表中的头一个？哪个元素是最后的？哪些元素在一个给定的元素之前和之后？表中有多少个元素？甚至在这种似乎很简单的情况下，关于结构也有不少的话要说（见2.2节）。

在更复杂的情况下，这种表可以是一个二维的数组（即是，一个矩阵或格栅，有一个行和一个列的结构），或者是一个对于较大的 n 值的 n 维数组；它可以是一个树形结构，表示着等级的或分枝的关系；或者它可以是一个拥有大量的交互联系的复杂的多链接的结构，就如同人的大脑那样。

为要能适当地使用一台计算机，必须着重了解：数据内部的结构关系，以及在计算机内表示和处理这样的结构的技术。

本章综述了关于信息结构的最重要的事实：不同类型的结构的静态和动态性质；进行存储分配的手段以及结构数据的表示；为建立，改变，存取以及破坏结构数据的有效算法。在研究过程中，我们也将给出若干重要的例子，来说明这些方法对于各种各样问题的应用。这些例子包括：拓扑的分类，多项式算术，离散系统的模拟，稀疏矩阵的运算，代数公式的操作，以及对于编写编译程序和操作系统的应用。我们所关心的，几乎完全是在一台计算机内部所表示的结构。从外部到内部表示的转换，是第9章和第10章的课题。

我们将要讨论的许多材料，通常都称为“列表处理”。因为我们已经设计出一些程序设计系统（例如，IPL-V，LISP，和SLIP），以便于对若干称作列表的一般类型的结构进行工作（在这一章中，当出现列表这个词时，是在一种技术的意义下来使用的，它表示一种在2.3.5小节中要详加研究的特殊类型的结构）。尽管列表处理系统在许多情况下都有用，但它对程序员强加了常常不必要的限制；通常更宜于把这一章的一些方法直接地用于人们自己的程序——选择合适的数据格式并视特殊的应用来处理算法。遗憾的是，还有许多人仍然觉得列表处理技术十分复杂（因此必须使用某些由他人精心编写的解释系统或子程序组），而且列表处理必须是仅以某一固定的方式来完成的。我们将看到，关于处理复杂结构的方法，并没有什么不可思议，神秘莫测或困难重重；这些技术应是每一位程序员所需知识中的一个重要组成部分，而且，无论用汇编语言，还是用象FORTRAN或ALGOL这样的编译程序语言来写程序，他都能容易地使用它们。

我们将借助于MIX计算机来说明处理信息结构的方法。不能细心地研读详细的MIX程序的读者，至少要看看信息结构在MIX的内存中的表示方式。

在这里，重要的是定义若干术语和记号；从现在起，我们将经常使用它们。一个表中

的信息是由一组节点（某些作者称之为“记录”，“实体”或“珠子”）组成的；有时我们也说成“条款”或“元素”，而不说是“节点”。每一个节点由计算机内存中的一个或多个连续的字组成，它又分成有名称的叫做场的诸部分。在最简单的情况下，一个节点恰好是内存中的一个字，而且它仅有一个包括整个字的场。作为一个较有趣的例子，假设我们打算用表的元素来表示扑克牌，则我们可以有分成五个场的两个字的节点，这五个场就是 TAG，SUIT，RANK，NEXT 和 TITLE：

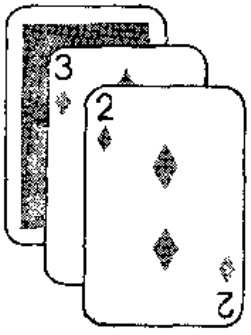
+	TAG	SUIT	RANK	NEXT
+	TITLE			

（1）

（这个格式反映了两个 MIX 字的内容。回想一下，一个 MIX 字由五个字节再加一个符号位组成；见 1.3.1 小节。在这个例子中，我们假定在每个字中的符号皆为+）。一个节点的地址，也称作链接，指针，或者是指向该节点的访问，是该节点的头一个字的内存地址。地址通常是相对于某个“基准”单元来取的；但在这一章中，为简便起见，我们将把地址取作绝对的内存地址。

一个节点内的任何场，其内容可以表示数，字母字符，链接，或者是程序员所希望的任何别的东西。就上述例子而言，让我们假定，我们希望来表示可能出现于一场单人纸牌游戏中的一叠扑克牌；TAG＝1 意味着牌是背面的，TAG＝0 意味着它是正面的；SUIT（花色）＝1，2，3 或 4 分别表示梅花，方块，红心或黑桃；RANK（点数）＝1，2，…，13 表示 A，2，…，老 K。NEXT（下一张）是在这叠牌中，本张牌中的指向下张牌之链接；而 TITLE 是这张牌的名称（五个字符），用于打印输出。典型的一叠牌可以象下面这样：

实际的纸牌　　计算机表示

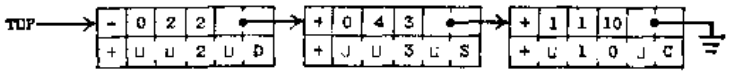


100:	+	1	1	10	A
101:	+	U	1	0	U C
386:	+	0	4	3	100
387:	+	U	U	3	U S
242:	+	0	2	2	386
243:	+	U	U	2	U D

（2）

在这里的计算机表示中，诸存储单元是 100，386，242；就这个例子来说，它们也可以是任何别的数目——只要每张牌链接到下一张就行。注意在节点 100 中的特殊的链接“Λ”；我们使用希腊字母 Λ 来表示空链接，即是不指向任何节点的链接。空链接 Λ 出现于节点 100 中，因为梅花 10 是这叠牌中最底下的那张牌。在机器内，就用某个不可能是节点地址，又易于识别的值，来表示 Λ。一般，我们将假定在单元 0 处不出现节点，而且，因此，在 MIX 程序中，Λ 几乎将总被表示为链接值 0。

引进指向其它数据元素的链接，是计算机程序设计中的一个极端重要的思想。这是表示复杂结构的关键。在图示节点的计算机表示时，用箭头来表示链接，通常是很方便的。于是，我们上述的例子将呈现为：



（3）

实际的地址242，386，和100（它归根到底是不相干的）已不再出现于表示（3）中。空链接可以画成电子线路记号中的“接地”。在（3）中我们加进了“TOP(顶)”，代表一个链接变量，通常称作一个指针变量，即是，在计算机程序之内的一个变量，其值是一个链接。在一个程序中，对节点的所有访问，都通过链接变量（或链接常数）直接地进行，或者通过别的节点的链接场间接地进行。

现在，我们给出记号的最重要部分，这是用于访问节点之内诸场的手段。办法很简单，就是先写出场的名称，并把指向所希望的节点的链接用圆括弧括起来紧接于其后。例如，在（1）、（2）、（3）中，我们有

$$\text{TITLE(TOP)}=\text{“}\square\square 2\square\text{D”};\qquad \text{SUIT(TOP)}=2;$$
$$\text{RANK(100)}=10;\qquad \text{RANK(NEXT(TOP))}=3 \tag{4}$$

读者应该仔细研究这些例子，因为在这一章和以下几章的许多算法中要使用这些记号。为使这些思想更清楚，我们现在来叙述把一张新的正面的牌放到这叠牌顶上的一个简单算法。假定NEWCARD是一个链接变量，其值为指向新牌的链接：

A1. 置NEXT(NEWCARD)←TOP（这是在新的纸牌节点中设置相应的链接）。

A2. 置TOP←NEWCARD（使TOP继续指向这叠牌的顶上）。

A3. 置TAG(TOP)←0（标志这张牌是“正面的”）。▌

另一个例子是计算当前在这叠牌中有几张牌的算法：

B1. 置N←0，X←TOP（这里N是一个整数变量，X是一个链接变量）。

B2. 若X＝A，则停止；N是这叠牌中的纸牌张数。

B3. 置N←N＋1，X←NEXT(X)，并转回到步骤B2。▌

注意在这些算法中，我们使用符号名称来表示两个截然不同的东西：对于变量的名称（TOP，NEWCARD，N，X）和对于场的名称（TAG，NEXT）。这些量不得混淆。如果F是场的名称，并且L≠A是一个链接，则F（L）是一个变量；但是，F本身不是一个变量——除非它已通过一个非空的链接来“定量”，否则它就不具有值。

还要使用两个记号，以便在地址与存于该处的值之间进行转换：

a) CONTENTS，总是表示一个字的节点的全字长的场；因此，CONTENTS（1000）就表示存储单元1000中的值，即是，它是一个取这个值的变量。如果V是一个链接变量，则CONTENTS（V）就表示由V指出的值（而不是V本身的值）。

b) 如果V是保存在一个存储单元中的某个值的名称，则LOC（V）就表示这个单元的地址。因此，如果V是一个变量，其值占有内存的一个全字，则我们有CONTENTS（LOC（V））＝V。

虽说MIXAL的记号并不算先进，但也很容易来把上述的记号变换成MIXAL汇编语言代码。链接变量的值置于变址寄存器中，而且MIX的部分场的能力就可以利用来指示所希望的场。例如，上边的算法A可以写成这样：

```
NEXT    EQU     4:5             对于汇编程序，定义NEXT
TAG     EQU     1:1                 和 TAG 场
        LD1     NEWCARD         A1. rI1←NEWCARD
        LDA     TOP                 rA←TOP                  (5)
        STA     0,1 (NEXT)          NEXT (rI1)←rA
        ST1     TOP             A2. TOP←rI1
        STZ     0,1 (TAG)       A3. TAG (rI1)←0
```

计算机能方便而高效地执行这些操作，是“链接存储器”概念所以重要的主要原因。

有时，我们还用一个单变量，来表示整个节点（即是，一组场而不只是一个场）。这样，我们可以写

$$\text{CARD}\leftarrow\text{NODE (TOP)} \tag{6}$$

其中 NODE 是一个象 CONTENTS 一样的场说明，只是它指示一整个节点，而且这里 CARD 是一个似（1）那样取值的变量。如果在一个节点中有 c 个字，则记号（6）就是 c 个赋值

$$\text{CONTENTS (LOC (CARD)}+j)\leftarrow\text{CONTENTS (TOP}+j),\quad 0\leqslant j<c \tag{7}$$

的一种缩写。

在汇编语言与用于算法中的记号之间，有一个重要的差别。因为汇编语言非常“低”级（接近于机器），MIXAL 程序中用的符号代表地址而不是值。因此在（5）中，符号 TOP 实际上表示地址，在内存中指向顶上纸牌的指针即出现于此；然而在（6）和（7）中它却表示 TOP 的值，即是顶上纸牌节点的地址。汇编语言与编译程序语言之间的这个差异，是新程序员经常造成混乱的来源，所以希望读者要做习题 7，其它习题，对于这一节中所引进的记号约定，也提供了有用的训练。

习题

1.〔04〕 在（3）中，

a) SUIT(NEXT(TOP))；b) NEXT(NEXT(NEXT(TOP))) 的值是什么？

2.〔10〕 正文中指出，在许多情况下 CONTENTS(LOC(V))＝V。然而在什么条件下，我们有 LOC(CONTENTS(V))＝V呢？

3.〔11〕 给出一个算法，这个算法实际上与算法 A 的作用相反，即是，它取走这叠纸牌的顶上纸牌（如果这叠纸牌不是空的话），并且把 NEWCARD 置为这张纸牌的地址。

4.〔18〕 给出类似于算法 A 的一个算法，但它是在这叠牌的底下放置新的背面的纸牌（这叠纸牌可能是空的）。

▶5.〔21〕 给出一个算法，这个算法实际上与习题 4 的作用相反，即是，假定这叠牌不是空的，而且叠底下的纸牌是背面的，该算法要取走这底下的纸牌并使得 NEWCARD 链接到这张纸牌（这个算法，在单人纸牌游戏中，有时叫做“诈取”）。

6.〔06〕 在扑克牌的例子中，假设 CARD 是一个变量的名称，其值是一整节点。操作 CARD←NODE（TOP）把 CARD 的各个场分别置成等于这叠纸牌中的顶上纸牌的诸场。在这个操作之后，下列记号中哪一个代表顶部纸牌的花色：（a）SUIT（CARD）；（b）

SUIT (LOC (CARD)); (c) SUIT (CONTENTS (CARD)); (d) SUIT (TOP)?

▶7. [04] 在正文里举为例子的 MIX 程序(5)中，链接变量TOP 被存入 MIX 计算机的一个字中，这个字的汇编 语言名称是TOP。若采用场结构 (1)，则以下哪个代码序列把量 NEXT (TOP) 带入寄存器A中?说明为什么另一个序列是不正确的?

a) LDA TOP (NEXT)

b) LD1 TOP
LDA 0,1(NEXT)

▶8. [18] 写出一个对应于算法 B 的 MIX 程序。

9. [23] 写出一个MIX 程序，它从顶部那张牌开始，印出这叠纸牌当前内 容的字母名称，每行印一张，而且把背面的纸牌带上圆括弧。

2.2 线 性 表

2.2.1 堆栈，排队和双排队

出现在数据中的结构信息，比起我们真正直接地要在一台计算机中表示的，通常要多得多。例如，在上一节的每个“扑克牌”节点里，我们用一个 NEXT 场来说明，在这叠牌中，什么牌是在它之下；但是，却没有直接的方法来求什么牌是在给定的牌之上，如果有的话；或者来求，一张给定的牌是在哪一叠里。当然，任何实际的一付扑克所具有的信息是很多的，它们都已经为计算机的表示所掩盖了，例如：关于纸牌背面的设计细节；在玩牌的房间里，纸牌与其它东西的关系；组成这些纸牌的分子，等等。可以想象，这样的一些结构信息将与特定的计算机应用有关，但是，显然我们决不想来保存出现于每种情况下的所有结构。事实上，对于大多数纸牌游戏，我们保存在我们早先的例子中的所有 事 实。就说 TAG 场，它告诉我们纸牌是正面的还是背面的，但通常这是用不着的。

因此，显然我们在每种情况下，都必须决定，在我们的表中究竟要表示多少结构，并且怎样才能使每项信息都可以存取。为了作出这样的决定，我们就需要知道，对于数据将要执行些什么操作。因此，在这一章中，我们不仅考虑数据的结构，而且还考虑将施加于数据之上的操作的种类；关于计算机表示的设计，既依赖于数据所应有的功能，也依赖于数据的内在性质。一般地说，既强调“功能”又强调“形式”，这对于设计问题是基本的。

为了进一步说明这一点，让我们来考虑计算机硬件设计中的一个简单例子。计算机的存储器：或者是“随机存取存储器”，即是，MIX 的主存储器；或者是“只读存储器”，即是用来保存实质上不变的信息的存储器；或者是“大容量辅助 存储 器”，象 MIX 的盘设备，它不能以高速度进行存取，然而它却能存储大量的信息；或者是“联想存储器”，更 适 当地应叫做“按内容寻址的存储器”，即是，对于这种存储器，信息是按存储器所存的值，而不是按它在什么单元来寻址的；等等。注意，对于每类存储器，我们打算要的功能是如此重要，以致已经反映到具体的存储器类型的名称上了；所有这些设备都是“存储”设备，但设置它们的目的，却深深地影响着它们的设计和它们的价格。

一个**线性表**是 $n \geqslant 0$ 个节点X[1]，X[2]，…，X[n]的集合，它的结构性质实质上仅涉及这些节点的线性（一维）的相对位置，这就是：如果 $n > 0$，则X[1]是头一个节点；当 $1 < k < n$ 时，则第 k 个节点X[k] 是在X[$k-1$] 之后，且在X[$k+1$]之

前；而且X〔n〕是最后的节点。

对于线性表，我们所要实施的操作，举例说来，包括如下一些：

（i）存取表的第k个节点，检查或改变其诸场的内容。

ii）在第k个节点的紧前边，插入一个新的节点。

iii）删去第k个节点。

iv）把两个或两个以上的线性表合成一个表。

v）把一个线性表拆成两个或两个以上的表。

vi）复制一个线性表。

vii）确定表中节点的个数。

viii）把表的节点加以重排，使其对它们的节点的某个场而言是递增的次序。

ix）检索这表，以寻找在其某个场中具有一特定值的节点。

在操作（i），（ii）和（iii）中，特殊情况$k=1$和$k=n$有其根本的重要性，因为一个线性表的头一项和最后一项比起一般的项来更易于找到。在这一章里，我们将不讨论操作（viii）和（ix），因为这些问题分别是第5章和第6章的课题。

完全一般地要求上述的所有九个操作，这在一个计算机的应用中是很少有的。所以我们发觉，按照最经常进行的操作的种类，有许多方式来表示线性表。对于线性表，要设计一个统一的对所有这些操作都高效的表示方法，是有困难的；例如，如果我们正在插入和删去一个长长的表中间的条款，与此同时，又要对随机的k获得存取这个表的第k个节点之能力，那是比较难于做到的。因此，我们依据所欲实施的主要操作，来区别线性表的类型——就如象我们所已说明的，计算机的存储器是按照所预期的应用来加以区别的。

我们经常遇到这样一些线性表，对于它们，插入、删去和存取一些值，几乎总是出现于头一个或最后一个节点。我们给这些表以特殊的名称：

堆栈是这样一种线性表，对于它，所有的插入和删去（以及，通常所有的存取）都是在该表的一端进行的。

排队是这样一种线性表，对于它，所有的插入都在表的一端进行，又所有的删去（以及，通常所有的存取）都在另一端进行。

双排队（“双端点的”排队）是这样一种线性表，对于它，所有的插入和删去（以及，通常所有的存取）都在表的两端进行。

因此，双排队比起堆栈或排队来，更为一般些；它与一叠纸牌有着某些共同的性质，而且将以相同的方式处理之。我们还要区别输出受限或输入受限的双排队，即其中仅仅允许分别地在一端进行删去或插入。

在某些学科中，“排队”一词已被使用于一种更广泛的意义，即用以描述任何类型的要进行插入和删去的表；于是，上边所标出的诸特殊情况，就被说成是各种“排队纪律”。然而在本书中，只打算采用“排队”这一术语的加了限制的用法，它类似于人们为了等候服务而有规则地站排那样。

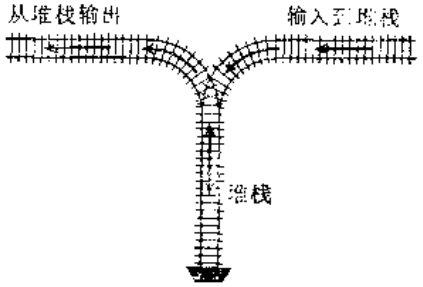


图1　一个表示成铁路转轨网络的堆栈

如同埃·怀·迪伊克斯特拉所建议的那样，通过与铁路车辆的转轨作一类比（见图1），

来理解一个堆栈的机构，有时是有帮助的。关于双排队的相应的图示请见图 2 。

对于堆栈，我们总是取走当前在表中“最年青”的条款，即是，比起任何其它条款来，它是最近新被插入的。反过来，那就恰好是排队了：对于排队，则总是取走“最老”的条款；节点离开表的顺序，与其进入表的顺序相同。

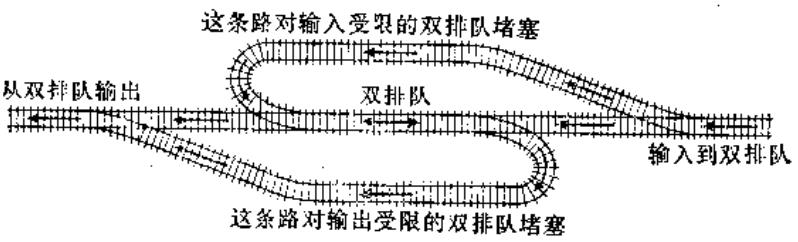


图 2　一个表示成铁路转轨网络的双排队

许多认识到堆栈和排队之重要性的人，曾独立地对于这些结构给出了其它一些名称：堆栈曾被称作下推表，反转存储，地窖，嵌套存储，叠堆，后进先出表（“LIFO”表），甚至叫做哨哨表！排队有时叫做循环存储或者先进先出（“FIFO”）表。LIFO 和 FIFO 这两个术语作为估价存货方法的名称，已经为会计人员使用许多年了。还有另一些术语，“架子”已经应用于输出受限的双排队，而输入受限的双排队曾叫做“卷轴”或“案卷”。这些其它名称的多重性，本身就是有趣的，因为它证明了这些概念的重要性。堆栈和排队这两个词正逐渐地变成为标准的术语。而在上边所列的所有其它的词当中，仅有“下推表”仍然用得相当普遍，特别是联系到自动机理论时。

实践中经常出现堆栈。作为一个简单的例子，我们来看这种情形：我们在审核一批数据，并记录一张关于异常条件或稍后要做的事情的清单。当处理完原先的这批数据后，我们就回到这张清单上，来做随后的处理，撤消清单中的项目直到它变为一张空单为止（例如，见习题 1.3.2-10 的“鞍点”问题）。为做此事，要用堆栈或用排队，而且一般说来用堆栈更为方便。当我们解决各种问题时，我们心目中始终都用着“堆栈”：一个问题导致另一个问题，而这另一问题又导致其它问题；我们把问题及（其所导致的）“子”问题堆积起来，并随着它们之被解决而撤消。类似地。在执行一个计算机程序时，进入和离开子程序的过程，有一种类似堆栈的行为。堆栈对于处理带有嵌套结构的语言，诸如：程序设计语言，算术表达式，德文的“套句”，是特别有用的。一般说来，在与明显的或隐涵的递归算法相联系时，最经常地出现堆栈，后边我们将在整个第 8 章中来讨论这个联系。

在涉及这些结构的算法中，一般都使用特殊的术语：我们把一个条款放到一个堆栈的顶上，或者取走顶上的条款（见图 3 a）。堆栈底下是最低的可存取的条款，而且直到所有其它的条款都被删去时，它才被取走。（人们常说他们把一个条款压入一个堆栈，而当删去顶上的条款时则说是弹出堆栈。这个术语是由于经常在自动食堂里看到盘碟的叠积，就其间的相似性而引进的；或者是就它与若干穿孔卡片设备上卡片的叠积之间的相似性而引进的。“压入”和“弹出”的简洁性有它的优点，但这些术语错误地意味着在计算机存储器之内来运动整个表。实际上并没有“被压下”，条款是被加到顶上，就象堆于草或摞箱子那样）。对于排队，我们着眼于这个排队的前头和后尾。事物进入后尾，并且，当其最终地达到前头的位置时，即被取走（见图 3 b）。当考虑双排队时，我们着眼于左端和右端

(图 3 c)。顶上，底下，前头和后尾的概念，有时候也应用于正作为堆栈或排队来使用的双排队，至于顶上，前头和后尾是出现在左边还是右边，并没有标准的约定。

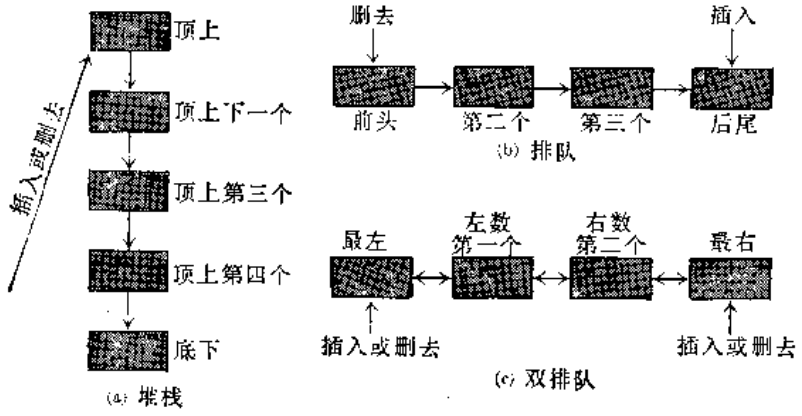


图 3　线性表的三个重要种类

这样一来，就使我们能够顺利地把语文中丰富的描述性词汇，用到我们的算法中：对于堆栈是“上下”的术语，双排队是“左右”的术语，排队是“站队等候”的术语。

为处理堆栈和排队，已经证明，再用一些附加的记号是方便的：我们写

$$A \Leftarrow x \qquad (1)$$

（当A是堆栈时）就意味着值x被插入到堆栈A的顶上，或者（当A是一个排队时）意味着x被插入到排队的后尾。类似地，记号

$$x \Leftarrow A \qquad (2)$$

用来表示，把变量x置成等于堆栈A之顶上或排队A之前头的值，而且从A中删去这个值。当A为空时，即是当A不包含值时，记号（2）没有意义。

当A是非空的堆栈时，我们可以写

$$\mathrm{top}(A) \qquad (3)$$

来表示它顶上的元素。

习题

1.〔06〕 一个输入受限的双排队是这样一个线性表，即条款只可在其一端插入，但却可从任一端取走；显然，一个输入受限的双排队可以或者作为堆栈或者作为排队来操作，如果我们一贯地从两个端点之一来取走所有条款的话。一个输出受限的双排队能否也作为堆栈或排队来操作？

▶2.〔15〕 想象四列列车位于图 1 中堆栈的输入一边，分别地编号成 1，2，3 和 4，假设我们实施以下的一系列操作（要求操作的方向与图中箭头的方向相一致，而且列车不得“跳越”另一列车）：（a）把列车 1 调入堆栈；（b）把列车 2 调入堆栈；（c）把列车 2 调去输出；（d）把列车 3 调入堆栈；（e）把列车 4 调入堆栈；（f）把列车 4 调去输出；（g）把列车 3 调去输出；（h）把列车 1 调去输出。

由于这些操作的结果，原来列车的顺序 1234 已经变成为 2431。这一道题和下几道题的目的，是要来检验用这样一种方式，从堆栈，排队或双排队可以得到什么样的排列。

如果有六列编号为 123456 的列车，那么，它们能否排列成顺序 325641？能否排列成顺序 154623？（在可能的情况下，说明如何来做它。）

3.〔25〕 上题中的操作（a）到（h），可以更简明得多地通过代码 SSXSSXXX 来描述，其中 S 代表"从输入往堆栈调动一列车"，而 X 代表"从堆栈往输出调动一列车"。某些 S 和 X 的序列确定的是无意义的操作，因为对于指定的轨道可能没有列车可供调用；例如，序列 SXXSSXXS 就不可能实现。

现在让我们称 S 和 X 的一个序列为允许的，如果它含有 n 个 S 和 n 个 X，而且如果它不指定不可能实现的操作。试描述一个规则，使得通过这个规则能容易地来区别允许的与不允许的序列；并且证明，没有两个不同的允许序列给出相同的输出排列。

4.〔$M34$〕 求出一个对于 a_n 的简单的公式，其中 a_n 是，使用象在习题 2 中那样的一个堆栈，对于 n 个元素可能得到的排列数。

▶5.〔$M28$〕 证明有可能利用一个堆栈从 $12\cdots n$ 得到排列 $p_1p_2\cdots p_n$，当且仅当，不存在下标 $i<j<k$ 使得 $p_j<p_k<p_i$。

6.〔00〕 以一个排队代替堆栈，来考虑习题 2 的问题。通过使用排队，可以得到 $12\cdots n$ 的什么排列？

▶7.〔25〕 以双排队代替堆栈，来考虑习题 2 的问题。（a）试求 1234 的一个排列，它能够通过一个输入受限的双排队来得到，但不能够通过输出受限的双排队来得到。（b）试求 1234 的一个排列，它能够通过一输出受限的双排队来得到，但不能够通过一输入受限的双排队来得到（作为（a）和（b）的推论：在输入受限和输出受限的双排队之间，有着一个确定的差别）。（c）试求 1234 的一个排列，它或者通过一输入受限的双排队，或者通过一输出受限的双排队，都不能得到它。

8.〔22〕 有没有任何这样的排列，它不可能通过利用一个双排队来得到，即使这双排队是既非输入受限亦非输出受限的？

9.〔$M20$〕 设 b_n 是通过利用一输入受限的双排队可以得到的，对于 n 个元素的排列数（注意，$b_4=22$，如习题 7 所说明的那样）。试说明，b_n 也是通过利用输出受限的双排队可以得到的，对于 n 个元素的排列数。

10.〔$M25$〕（见习题 3）设 S，Q 和 X 分别表示在一输出受限的双排队的左边插入一个元素，在其右边插入一个元素和从其右边略去一个元素的操作。例如，序列 QQXSXSXX 将把输入序列 1234 变换成 1342。序列 SXQSXSXX 也给出相同的变换。

想办法定义符号 S，Q 和 X 的一个序列是允许的概念，使得：（a）每一个允许的序列，实施一个有意义的操作序列，这个有意义的序列定义 n 个元素的一个排列；以及（b）每一通过输出受限的双排队可得到的 n 个元素的排列，恰对应于一个允许的序列。

▶11.〔$M40$〕 作为习题 9 和 10 的推论，数 b_n 是长度为 $2n$ 的可允许序列的个数。试求生成函数 $\sum_{n\geqslant 0}b_nz^n$ 的"封闭形式"。

12.〔$HM34$〕 计算习题 4 和 11 中的量 a_n 和 b_n 之近似值。

13.〔$M49$〕 通过使用一般的双排队，可以得到多少 n 个元素的排列？

2.2.2 顺序分配

在计算机内，保存线性表的最简单和最自然的方式，是把表的条款一个接着一个地放进顺序的单元。于是，我们就有

$$LOC(X[j+1])=LOC(X[j])+c$$

其中 c 是每个节点的字数（通常 $c=1$。当 $c>1$ 时，把一个统一的表分成 c 个"平行的"表有时更为方便，即使得节点 $X[j]$ 的第 k 个字，按照与 $X[j]$ 的头一个字所在单元保持着一个固定的距离来存储。但是，我们还假定，这些相邻的由 c 个字组成的字组，都是一个个单个的节点）。一般地，

$$LOC(X[j])=L_0+cj \tag{1}$$

其中 L_0 是一个叫做基地址的常数，它是人为地采取的节点 $X[0]$ 的地址。

这一表示线性表的技术是如此显然，以致似乎没有必要再费笔墨来加以详述。但我们将会看到这一章的后边所用的许多"更巧妙的"表示方法；首先考察简单的情况，以便说明我们如何处理，也是一个好想法。重要的是要理解顺序分配的局限性和使用的效力。

对于处理堆栈来说，顺序分配是十分方便的。我们只须有一个称作堆栈指针的变量 T。当堆栈为空时，令 $T=0$。为了在堆栈顶上放进一个新的元素 Y，我们置

$$T\leftarrow T+1;\quad X[T]\leftarrow Y \tag{2}$$

而且，当堆栈非空时，我们能把顶上节点置于 Y 中，并通过（2）的逆动作来删去这个节点：

$$Y\leftarrow X[T];\quad T\leftarrow T-1 \tag{3}$$

（在计算机内，由于（1），通常最有效的是维持值 cT，而不是 T。进行这样的修改是容易的，所以我们仍然按 $c=1$ 来继续我们的讨论。）

排队或更一般的双排队的表示是有点技巧的。明显的解决办法是使用两个指针，比方说是 F 和 R（用于排队的前头和后尾），而且当排队是空队时，$F=R=0$。在排队的后尾插入一个元素，将是

$$R\leftarrow R+1;\quad X[R]\leftarrow Y \tag{4}$$

而取走前头的节点（F 恰好指向前头的下一个）将是

$$F\leftarrow F+1;\quad Y\leftarrow X[F];\ 若\ F=R,\ 则置\ F\leftarrow R\leftarrow 0 \tag{5}$$

但是要注意会出现什么情况：如果 R 总是位于 F 的前头（这时至少有一个节点在排队中），则表格所用的条款将是 X[1]，X[2]，…，X[1000]，…直到无穷，这样浪费存储空间是惊人的。因此简单的方法（4），（5），仅当已知 F 经常赶上 R 的情况下，才被采用（例如，如果所有的删去把排队删空时）。

为了解决排队漫越存储器的问题，我们可以让 X[1] 跟在 X[M] 之后，暗中把 M 个节点 X[1]，…，X[M] 联成一个圆圈。于是，上边的过程（4），（5）变成

$$如果\ R=M\quad 则\ R\leftarrow 1，否则\ R\leftarrow R+1;\quad X[R]\leftarrow Y \tag{6}$$

$$如果\ F=M\quad 则\ F\leftarrow 1，否则\ F\leftarrow F+1;\quad Y\leftarrow X[F] \tag{7}$$

这个循环排队的动作，很象我们在输入输出缓冲的讨论（见 1.4.4 小节）中已经见过的那样。

我们上边的讨论已经非常不现实了，因为我们暗中假定了不会出现什么毛病。当我们从一个堆栈或排队删去一个节点时，已经假定了至少要有一个节点存在。当我们把一个节点插入到一个堆栈或排队时，已经假定了存储器中有着供存放它的空间。但显然，方法（6），（7）只允许在整个排队中至多有M个节点，而且方法（2），（3），（4），（5）也只允许T和R达到在任一给定的计算机程序之内的某个极大值。下面的说明指出，对于没有假定这些限制已自动地满足的一般情况，怎样来改写上面的动作：

X⇐Y（插入堆栈）：T←T＋1；若T＞M，则OVERFLOW；

$$X[T]\leftarrow Y \tag{2a}$$

Y⇐X（从堆栈删去）：若T＝0，则UNDERFLOW；Y←X〔T〕；

$$T\leftarrow T-1 \tag{3a}$$

$$X\Leftarrow Y\ \text{（插入排队）：}\begin{cases}\text{若 } R=M,\ \text{则 } R\leftarrow 1,\ \text{否则 } R\leftarrow R+1;\\ \text{若 } R=F,\ \text{则OVERFLOW;}\\ X[R]\leftarrow Y。\end{cases} \tag{6a}$$

$$Y\Leftarrow X\ \text{（从排队删去）：}\begin{cases}\text{若 } R=F,\ \text{则 UNDERFLOW;}\\ \text{若 } F=M,\ \text{则 } F\leftarrow 1,\ \text{否则 } F\leftarrow F+1;\\ Y\leftarrow X[F]。\end{cases} \tag{7a}$$

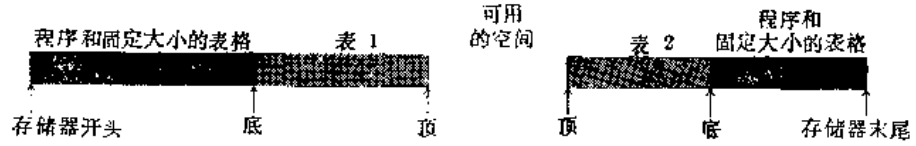
这里我们假定X〔1〕，…，X〔M〕是为本表所允许的空间的总额；OVERFLOW（上溢）和UNDERFLOW（下溢）意味着条款之溢超或欠缺。注意当我们使用（6a）和（7a）时，对于排队指针的初始设置F＝R＝0已不再是正确的了；比如说，我们将以F＝R＝1开始。

建议读者来做习题1，这道题讨论了这个简单的排队机构的一个并非浅显的方面。

下一个问题是，“当出现UNDERFLOW或OVERFLOW时，我们怎么办？”在UNDERFLOW的情况下，我们是试图取走一个不存在的条款；这通常是一个有意义的条件——而不是一个出错的情况——它可以用来支配一个程序的流向，例如，我们可能想要重复地删去条款，直到出现UNDERFLOW时为止。但是，OVERFLOW的情形，通常是一种出错；它意味着这个表格业已填满，然而却还有更多的信息有待放入。在出现OVERFLOW的情况下，通常的策略是必须报告：由于超过了存储容量，程序已不能进行，因而程序终止。

当然，如果我们就终止于一种这样的OVERFLOW的情况，即仅当一个表已经变得太大，而同一程序的其它表又有非常大的剩余空间时，那是颇为遗憾的。在上述讨论中，我们主要是把程序想象为只有一张表。然而，我们经常会遇到涉及若干个堆栈的程序。每一个堆栈的大小都是动态变化的。在这种情况下，我们将不对每个堆栈的大小设定一个极大值。因为堆栈的大小通常是不可预测的；而且即使对于每个堆栈，已经确定了一个极大值，也很少见到全部堆栈都同时被填满的情况。

当恰有两个大小可变的表时，如果我们让这两个表彼此迎面地增长，则它们就能很好地共存在一起：



这里表 1 向右延伸，而表 2（以相反的顺序存储）向左延伸。OVERFLOW 将不出现，除非两个表的总容量耗尽了全部的存储空间。这两个表可以独立地延伸或收缩，使得每一个的有效极大容量显著地大于可利用空间的一半。这种存储空间的安排，经常地被采用。

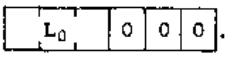
然而，读者会很容易相信，在存储器中没有办法来存三个以上的可变大小的顺序表，使得（a）仅当所有表的总容量超过总的空间时，才出现 OVERFLOW，并且（b）每一表的"底"元素都放在一个固定的单元中。如果有，比如说，十个以上可变大小的表——而这并不是不常见的——则存储分配的问题就是非常有意义的了。如果我们希望满足条件（a），则必须放弃条件（b）；即是说，我们必须允许这些表的"底"元素的位置可变。这就意味着等式（1）的单元 L_0 不再是不变的了；对于这表格已不可能来访问一个绝对的存储地址了，所有的访问都必须相对于基地址 L_0 来进行。在 MIX 的情况下，把一个占一个字的节点送入寄存器 A 的编码，将从

```
LDI   I
LDA   L0,1
```

改变成，例如是

```
LDI   I
LDA   BASE(0:2)
STA   *+1(0:2)                    (8)
LDA   *,1
```

其中 BASE（基）包含



这个相对寻址，同固定的基地址相比较，显然要慢些；尽管我们发现，如果 MIX 有一个"间接寻址"的特性（见习题 3）时，则它将仅仅是稍微地慢一点点而已。

当每个可变大小的表都是一个堆栈时，一个重要的特殊情况出现了。这时，由于在任何时候仅仅关系到每个堆栈的顶上元素，所以，我们就能几乎象以前一样来进行有效的处理。假设我们有 n 个堆栈；上述的插入和删去的算法就变成如下：如果 BASE(i) 和 TOP(i) 是第 i 个堆栈的链接变量，而且如果每一节点都占一个字，则

插入：TOP(i)←TOP(i)+1；如果 TOP(i)≥BASE($i+1$)，则
OVERFLOW；否则置 CONTENTS (TOP(i))← Y。 (9)

删去：如果 TOP(i)=BASE(i)，则 UNDERFLOW；否则置
Y←CONTENTS(TOP(i))，TOP(i)←TOP(i)−1。 (10)

这里 BASE($i+1$)是第 $i+1$ 个堆栈的基单元。条件 TOP(i)=BASE(i)意味着第 i 个堆栈是空的。

在上述情况下，OVERFLOW 已不再象以前那样是一个危急关头了；我们可以来"改组存储器"，即从那些尚未填满的表格中腾出地方，来补给已经溢出的表格。实现这一改组的种种可能的方式，翩翩浮现在我们的眼前；而且，由于这些改组的算法对于线性表的

顺序分配是非常重要的，所以，我们现在就来详细地考虑这个问题。我们将从其中最简单的一个开始，然后再考虑另外的一些。

假设有 n 个堆栈，而且对值 BASE〔i〕和 TOP〔i〕，就如象在（9）.(10）中那样来进行操作。这些堆栈全都共享由满足 $L_0<L\leqslant L_\infty$ 的所有单元 L 组成的公共存储区域（这里 L_0 和 L_∞ 是确定可供利用的总字数的常数)。我们可以从所有的堆栈都为空，而且对于所有的 i 都是 BASE〔i〕=TOP〔i〕=L_0 开始。我们再置 BASE〔$n+1$〕=L_∞ 以便(9）对于 $i=n$ 能适当地工作。现在，除了堆栈 n 之外，每当一个具体的堆栈、获得比它以前已有的还多的条款时，便将出现 OVERFLOW。

当堆栈 i 溢出时，有三种可能性：

a）我们求出满足 $i<k\leqslant n$ 且 TOP〔k〕<BASE〔$k+1$〕的最小的 k，如果存在这种 k 的话。现在把事物上移一格：

置 CONTENTS(L－1)←CONTENTS(L)，对于TOP〔k〕$\geqslant$L>BASE〔$i+1$〕。(注意，这应该对于L值的降序，而不是增序来进行，以免丢失信息。有可能出现 TOP〔k〕=BASE〔$i+1$〕，在这种情况下，什么也不需要移动)。

置 BASE〔j〕←BASE〔j〕+1，TOP〔j〕←TOP〔j〕+1，对于 $i<j\leqslant k$。

b）找不到（a）中那样的 k，但我们找到了满足 $1\leqslant k<i$ 且 TOP〔k〕<BASE〔$k+1$〕的最大的 k。现在下移一格：

置 CONTENTS(L－1)← CONTENTS(L)，对于BASE〔$k+1$〕<L<TOP〔i〕。(注意这应该对L值的增序来进行)。

置 BASE〔j〕←BASE〔j〕－1，TOP〔j〕←TOP〔j〕－1，对于 $k<j\leqslant i$。

c）对于所有的 $k\neq i$，我们都有 TOP〔k〕=BASE〔$k+1$〕。于是显然地，我们已经不可能有余地为新的堆栈条款找到空隙，因此我们只好中止。

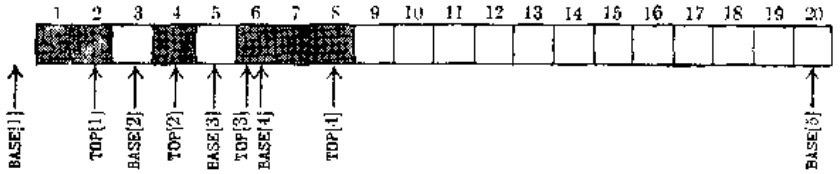


图 4　经若干个插入和删去之后，存储器的组态之示例

图 4 说明了对于 $n=4$，$L_0=0$，$L_\infty=20$，经逐次的动作

$$I_1^*\ I_1^*\ I_4\ I_2^*\ D_1\ I_3^*\ I_1\ I_1^*\ I_2^*\ I_4\ D_2\ D_3$$

之后，存储器的组态（这里 I_j 和 D_j 指的是在堆栈 j 中的插入和删去，而带 * 者指的是出现一个 OVERFLOW，假定开始时对堆栈 1，2 和 3 都没有分配空间)。

显然，如果我们合理地选择我们的初始条件，而不是象上边所提议的那样，一开始时就把全部的空间都分配给第 n 个堆栈，那么，就可以消除在使用这个方法时出现的许多溢出。例如，如果我们预期每一个堆栈的大小都相同，则我们就可以从

$$\mathrm{BASE}〔j〕=\mathrm{TOP}〔j〕=\left\lfloor\left(\frac{j-1}{n}\right)(L_\infty-L_0)\right\rfloor+L_0 \tag{11}$$

开始。根据一个具体的程序的操作经验，还可以给出较好的起始值；然而，不管初始分配

搞得如何之好，至多也只能减少固定数量的溢出，而效果亦仅在一个程序运行的早期阶段，才是显著的。

上述方法的另一个可能的改进，就是在每次改组存储器时，对一个以上的新条款腾出位置。在存储器中移动表格是一项颇费时间的操作，我们通过一次移出2或3个来代替多次移出1个，就可以赢得时间。

这个思想已为简·加威克（J.Garwick）所阐明。他提议，当出现溢出时，要对存储器进行完整的改组，而这改组的依据是：每一堆栈大小在上次改组后之变化。这个算法用了另外一个叫做OLDTOP[i]，$1\leqslant i\leqslant n$的数组，它保存：恰在前次分配了存储器之后，TOP[i]得到的值。开始时，就如同以前一样来设置表格，而且OLDTOP[i]=TOP[i]。新的算法进行如下：

算法G(对顺序表格进行再分配) 假设按照(9)，在堆栈i中已经出现OVERFLOW。在实施了算法G之后，我们或者发现已经超过了存储器的容量，或者已经把存储器重新安排，使得可以来进行NODE(TOP[i])←Y的动作（注意，在算法G进行之前，TOP[i]已经在（9）中增值了）。

G1.〔初始化〕置SUM←$L_{m}-L_{0}$，INC←0。然后对于$1\leqslant j\leqslant n$进行步骤G2(结果将是使得SUM等于剩下的存储空间的总量，而INC等于经上次分配之后表大小的增长总量)。在做完这些之后，即继续做步骤G3。

G2.〔收集统计〕置SUM←SUM−(TOP[j]−BASE[j])。如果TOP[j]>OLDTOP[j]，则置D[j]←TOP[j]−OLDTOP[j]和INC←INC+D[j]；否则置D[j]←0。

G3.〔存储器满？〕若SUM<0，则不能再进行。

G4.〔计算分配因子〕置$\alpha\leftarrow 0.1\times$SUM/$n$，$\beta\leftarrow 0.9\times$SUM/INC（这里$\alpha$和$\beta$都是分数，不是整数，它们都有待计算到合理的精度。下面的步骤是把可利用的空间分给每个表：当前可利用的存储的百分之十左右将为n个表平均地分享，剩下的百分之九十将根据在前次分配后表之大小的增长数量按比例分配)。

G5.〔计算新的基地址〕置NEWBASE[1]←BASE[1]和$\sigma\leftarrow 0$；然后对于$j=2$，3，…，n置$\tau\leftarrow\sigma+\alpha+$D[$j-1$]$\beta$，NEWBASE[$j$]←NEWBASE[$j-1$]+TOP[$j-1$]−BASE[$j-1$]+$\lfloor\tau\rfloor-\lfloor\sigma\rfloor$，以及$\sigma\leftarrow\tau$。

G6.〔改组〕置TOP[i]←TOP[i]−1(这反映了第i个表的真实大小，使得不致企图从表的边界之外来移动信息)。实施以下的算法R，然后恢复TOP[i]←TOP[i]+1最后对于$1\leqslant j\leqslant n$置OLDTOP[j]←TOP[j]。▌

也许，这整个算法的最有趣部分，是一般的改组过程。我们现在就来描述它。改组是不平凡的，因为存储器中某些部分要上移，而另外一些部分要下移，显然，重要的是，移动时不得冲掉任何有用的信息。

算法R(对顺序表格重新定址) 对于$1\leqslant j\leqslant n$按照上面所述的约定，把由BASE[j]和TOP[j]确定的信息，移动到由NEWBASE[j]确定的新位置，而且适当地调整BASE[j]和TOP[j]的值。

R1.〔初始化〕置$j\leftarrow 1$。

R2.〔找移动的起点〕(现在,全部从1到j的表已经移到所需要的位置)。以步长1不断增长j值,直至找到或者

a) NEWBASE〔j〕<BASE〔j〕:则转到R3;或者

b) NEWBASE〔j〕>BASE〔j〕:则转到R4;或者

c) $j>n$:则算法结束。

R3.〔下移〕置$\delta\leftarrow$BASE〔j〕$-$NEWBASE〔j〕。置CONTENTS(L$-\delta$)$\leftarrow$CONTENTS(L),对于L=BASE〔j〕+1,BASE〔j〕+2,…,TOP〔j〕(注意BASE〔j〕有可能等于TOP〔j〕,在这种情况下不要求任何动作)。置BASE〔j〕$\leftarrow$NEWBASE〔j〕,TOP〔j〕$\leftarrow$TOP〔j〕$-\delta$。转到R2。

R4.〔找移动的顶点〕找最小的$k\geqslant j$使得NEWBASE〔$k-1$〕$\leqslant$BASE〔$k+1$〕(注意:NEWBASE〔$n+1$〕将等于BASE〔$n+1$〕,因此这样一个k总存在)。然后对于$i=k,k-1,\cdots,j$进行步骤R5;最后置$j\leftarrow k$并转到R2。

R5.〔上移〕置$\delta\leftarrow$NEWBASE〔i〕$-$BASE〔i〕。置CONTENTS(L$+\delta$)$\leftarrow$CONTENTS(L),对于L=TOP〔i〕,TOP〔i〕-1,…,BASE〔i〕+1(注意,如同在步骤R3中那样,这里也可能不需要任何动作)。置BASE〔i〕$\leftarrow$NEWBASE〔i〕,TOP〔i〕$\leftarrow$TOP〔i〕$+\delta$。■

注意堆栈1根本不需要移动。所以,为了高效起见,如果程序员知道那一个堆栈将是最大的,则他就应该首先设置这个最大的堆栈。

在算法G和R中,我们已经有目的地造成:对于$1\leqslant j\leqslant n+1$,有可能有

$$\text{OLDTOP}〔j〕\equiv \text{D}〔j-1〕\equiv \text{NEWBASE}〔j〕$$

即是,由于这三个值决不会在冲突的时间里需要,因此这三个表格可以共享公共的存储单元。当使用这种重迭技术时,步骤G2的实施必须对于降序的j值来进行,而且在步骤G5中必须置NEWBASE〔$n+1$〕$\leftarrow$BASE〔$n+1$〕。

我们已经对于堆栈描述了这些改组的算法。但是显然,它们也可应用于其当前信息包含在BASE〔j〕和TOP〔j〕之间的那样的任何相对寻址的表。对于这些表,也可以附加另外一些指针(例如,FRONT〔j〕,指前头;REAR〔j〕,指后尾),使得它们成为一个排队或双排队。见习题8,它详细地考虑了一个排队的情况。

类似于上边那些的动态存储分配的算法,其数学分析是极为困难的。在后面的一些习题中,出现了某些有趣的结果,尽管就所涉及的一般行为而言,它们只不过开始抓住了点皮毛。

作为可以导出的理论的一个例子,假设我们来考虑当表格仅仅通过插入而增长的情况;凡是作用相抵消的删去和插入,概予忽略。让我们进一步假定,预期每一个表格都以相同的速率来填满。通过想象一个由m个插入操作$a_1,a_2,\cdots,a_m$组成的序列,可以模拟这种情况;这里每个a_i表示1与n之间的一个整数(表示对于堆栈a_i顶上的一个插入)。例如,序列1,1,2,2,1意味着对于堆栈1的两个插入,接着是对于堆栈2的两个插入,再接之以堆栈1的另一个插入。我们可以认为,n^m种可能的指定$a_1,a_2,\cdots,a_m$的每一种,都是同等地可能的。而后,我们可以问及,随着整个表格之被构成,在改组操作期间,需要把一个字从一个单元移动到另一个单元的平均次数。对于头一个算法,以全部可利用的空间都给

予第 n 个堆栈而起始者，这个问题在习题 9 中分析。我们得到，所求的平均移动操作次数是

$$\frac{1}{2}\left(1-\frac{1}{n}\right)\binom{m}{2} \tag{12}$$

于是，如同我们可以预期的那样，移动的次数实质上与表格中的条款数之平方成正比。如果个别的堆栈非同等可能时，则这同一结果还是正确的（见习题 10）。

以往的教训似乎是，如果相当大数量的条款被放进表格中，则将导致很大数量的移动。这是我们为把大量的顺序的表格紧密地组装在一起，所必须支付的代价。分析算法 G 之特征的理论，尚未建立起来；而且无论如何也不大可能有任何简单的模型，来描述在这样的环境之下实际表格的特征。

经验证明，当存储器仅有一半被装入时（即是，当可利用的空间等于总的空间的一半时），我们很少需要以算法 G 来重新安排诸表格。重要的事情也许是，在“半满”的情况下，本算法工作得很好；而且在几乎全满的情况下，它至少能正确地工作。

但是让我们更仔细地来考虑关于几乎全满的情况。当诸表格几乎填满存储器时，算法 R 就要花费更长些的时间来实现它的任务；而且在存储器行将用完之前，就会引起更为经常地出现坏的情况 OVERFLOW。只有非常少数的程序，竟会近乎把存储器填满，而不在其后不久即完全溢出；而造成存储器溢出的那些情况，在存储器越限之前，将可能在算法 G 和 R 中花费大量的时间。遗憾的是，未调试好的程序却经常地溢出存储器容量。为避免浪费所有这些时间，一个可行的方案是：如果 SUM 小于 $S_{\min}$，其中 $S_{\min}$ 是一个由程序员选择来防止过度的改组的值，则就在步骤 G3 处停止算法 G。当有许多可变大小的顺序表格时，我们不应当期望在存储器超限之前，百分之百地利用存储空间。

习题

▶1. [15] 在由（6a），（7a）给出的排队操作中，一次能有多少条款在排队中，而不致出现 OVERFLOW？

▶2. [22] 把（6a），（7a）的方法推广到应用于任何具有少于 M 个元素的双排队中。换句话说，给出对于另外两个操作“从后尾删去”和“在前头插入”的说明。

3. [21] 假设 MIX 扩充如下：每条指令的 I 场有 $8I_1+I_2$ 的形式，其中 $0\leqslant I_1<8$，$0\leqslant I_2<8$。在汇编语言中，人们写“OP ADDRESS, I_1:I_2”或者（象现在这样）“OP ADDRESS, I_2”如果 $I_1=0$。意思是：首先实施对 ADDRESS 的“地址修改” I_1，然后对所得的地址实施“地址修改” I_2，最后再来对于新地址实施操作 OP。地址之修改定义如下：

0： M＝A

1： M＝A＋(rI1)

2： M＝A＋(rI2)

…

6： M＝A＋(rI6)

7： M＝由在单元 A 中找到的“ADDRESS, I_1:I_2”场所定义的结果地址。在单元 A 中不允许 $I_1=I_2=7$ 的情况（习题 5 中讨论了作此限制的原因）。

这里A表示操作前的地址，M表示经地址修改后的结果地址。在所有情况下，如果M的值不适合于两个字节加一符号位，则这结果就是没有定义的。为实施每个“间接寻址”操作（修改7），执行时间都增加一个单位时间。

作为一个非平凡的例子，假设单元1000包含“NOP 1000,1:7”；单元1001包含“NOP 1000,2”；而且变址寄存器1和2分别包含1和2。于是，指令“LDA 1000，7:2”就等价于“LDA 1004”，因为

1000，7:2＝(1000，1:7)，2＝(1001，7)，2＝(1000，2)，2＝1002，2＝1004。

a）利用这个间接寻址的特性（如果必要的话），说明怎样来简化（8）右边的编码，使得每访问一次表格，都能节省两条指令。你的代码比（8）要快多少？

b）假设有若干个表格，它们的基地址存于单元BASE＋1，BASE＋2，BASE＋3，…；假定rI1中是I，rI2中是J；试问怎样才能利用间接寻址的特性，用一条指令来把第J个表格的第I个元素送到寄存器A中？

c）假定单元X中的（3:3）场为0，试问指令“ENT4 X，7”的效果是什么？

4.〔25〕 假定MIX已如同习题3那样扩充了，试说明对于以下的每一个动作，怎样给出一单条的指令（加上辅助常数）：

i）由于间接寻址永不终止，而无限地循环。

ii）把值LINK(LINK(x))传送到寄存器A中，其中链接变量x的值存于其符号地址为X之单元的（0:2）场中，LINK(x)的值存于单元x的（0:2）场中，等等，假定在这些单元中的（3:3）场皆为0。

iii）在类似于（ii)中所作的假定下，把值LINK(LINK(LINK(x)))传到寄存器A中。

iv）把单元（rI1)＋(rI2)＋…＋(rI6）的内容传到寄存器A中。

v）将rI6的当前值四倍之。

▶5.〔35〕 习题3中所提议的MIX的扩充，有一个不幸的限制，即是在一个间接地寻址的单元中，不允许“7:7”。

a）给出一个例子来指出，如果没有这个限制，则大概就必须要求，MIX的硬件要维持一个长长的三位条款的内部堆栈。（这将是过于昂贵的硬件，甚至对于象MIX这样假想的机器也是这样）。

b）试说明，在现在的限制之下，这样一个堆栈为什么不必要了；换言之，试设计一个算法，通过这个算法，一个计算机的硬件，无须许多附加的寄存器能力，就能实现所希望的地址修改。

c）给出一个比习题3关于7:7的用法要适度些的限制，它减少了习题4(iii）的困难，而且在计算机硬件中，它还能便宜地实现。

6.〔10〕 从图4中所示的存储器组态开始，确定下边的操作序列哪一个引起溢出或下溢：（a）I_1；（b）I_2；（c）I_3；（d）$I_4I_4I_4I_4I_4$；（e）$D_2D_2I_2I_2I_2$。

7.〔12〕 算法G的步骤G4指出一个除以数量INC的除法。当算法进行到这一点时，INC竟能为0吗？

▶8.〔25〕 对于这样的情况，即一个或多个表是如象在（6a），（7a）那样处理作循环的排队的情况，阐明怎样来修改（9），（10）和重组的算法。

▶9.〔*M*27〕 利用在这一小节邻近结尾处描述的数学模型，证明等式（12）确是所求的移动数（注意序列1，1，4，2，3，1，2，4，2，1确定0＋0＋0＋1＋1＋3＋2＋0＋3＋6＝16次移动。）

10.〔*M*28〕 修改习题9的数学模型，俾能预期某些表格比其它的更大些：命 p_k 是对于 $1 \leqslant j \leqslant m$，$1 \leqslant k \leqslant n$，$a_j = k$ 的概率。于是 $p_1 + p_2 + \cdots + p_n = 1$；上题考虑了对于所有的 k，$p_k = 1/n$ 的特殊情况。试对于这个更一般的情况，如象在等式（12）中那样，确定预期的移动次数。有可能重新安排 n 个表的相对次序，使得预期要长些的表都放到预期要短些的表之右边（或左边）；为使以 p_1，p_2，…，p_n 为基础的预期移动次数为极小，什么是 n 个表的最好的相对次序？

11.〔*M*30〕 推广习题9的论述，使得在任何堆栈的头 t 个插入不引起移动，而随后的插入不受影响。于是，如果 $t = 2$，则习题9中的序列确定0＋0＋0＋0＋0＋3＋0＋0＋3＋6＝12次移动。在这一假定下，平均移动总数是什么？（当每个堆栈以 t 个可利用的空间开始时，这就是本算法的近似行为。）

12.〔*M*28〕 在内存中同时共存两个表格，它们彼此相向地扩展，而不是使它们保持在分开的独立的有界区域中，这样做的优点，可以定量地估计（到某一程度）如下：就 $n = 2$ 利用习题9的模型；对于 2^m 个同等可能的序列 a_1，a_2，…，a_m 的每一个，设有 k_1 个1和 k_2 个2（这里 k_1 和 k_2 是在内存填满之后两个表格的分别的大小。当这些表格邻接时，我们能够以 $m = k_1 + k_2$ 个单元来运行算法，而不是以分开的表格，用 $2\max(k_1, k_2)$ 个单元来获得同样的效果）。

$\max(k_1, k_2)$ 的平均值是什么？

13.〔*M*47〕 如果允许随机删去以及随机插入，在表格中引进更大的波动，则在习题12中考察的值 $\max(k_1, k_2)$ 将甚至更大。假设我们改变这一模型，使得序列值 a_j 解释为删去，而不是插入的概率为 p；这过程一直继续到 $k_1 + k_2$（使用中的表格单元的总数）等于 m 为止。从一空表中进行删去将不起作用。

例如，如果 $m = 4$，则可以说明，当上述进程停止时，我们得到概率分布：

(k_1, k_2) 之值（4，0）或（0，4），（3，1）或（1，3），　（2，2），

以概率：$\dfrac{1}{16-12p+4p^2}$，　$\dfrac{1}{4}$，　$\dfrac{6-6p+2p^2}{16-12p+4p^2}$

出现。

因此，随着 p 增加，k_1 与 k_2 之差亦趋向于增加。不难证明，在 p 趋于1的极限下，k_1 的分布实质上变成均匀的，而且 $\max(k_1, k_2)$ 的极限值恰为 $\frac{3}{4}m$，当 m 为一偶数时。这一特性和上题（当 $p = 0$ 时）十分不同；然而，它可能不是特别有意义的。因为当 p 趋于1时，为结束这一进程所需的时间总数剧速地趋近无穷。这一道题提出的问题，是来检验 $\max(k_1, k_2)$ 对于 p 和 m 的相关性；并来确定当 m 趋于无穷时，对于固定的 $p\left(\text{象 } p = \frac{1}{3}\right)$ 的渐近公式。

14.〔*HM*43〕 证明：当 n 固定，m 趋于无穷时，数量

$$\frac{m!}{n^m}\sum_{\substack{k_1+\cdots+k_n=m\\ k_1,\cdots,k_n\geqslant 0}}\frac{\max(k_1,\ \cdots,\ k_n)}{k_1!\cdots k_n!}$$

有渐近形式 $(m/n)+c_n\sqrt{m}+O(1)$，从而把习题 12 的结果推广到任意的 $n\geqslant 2$。并确定常数 c_2，c_3，c_4，和 c_5。

15.〔40〕 利用蒙特卡罗方法，来模拟在改变插入和删去的分布的情况下，算法 G 的行为。你的试验关于算法 G 的效能如何？请把它的性能与更早给出的每次上移或下移一个节点的算法加以比较。

16.〔20〕 正文中说明了可以如何设置两个堆栈，使得它们彼此相向地扩展，从而得以高效地利用一个公共的存储区域。试问：两个排队，或者，一个堆栈与一个排队，能否以同样的有效程度来利用一个公共的存储区域？

2.2.3 链接分配

代替在顺序的存储单元中保存一个线性表，我们可以利用一个更灵活的方案。在这一方案中，每个节点都含有一个指向该表下个节点的链接。

顺序分配：

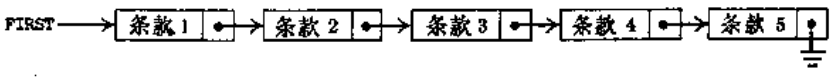
地址	内容
L_0+c：	条款 1
L_0+2c：	条款 2
L_0+3c：	条款 3
L_0+4c：	条款 4
L_0+5c：	条款 5

链接分配：

地址	内容	
A：	条款 1	B
B：	条款 2	C
C：	条款 3	D
D：	条款 4	E
E：	条款 5	A

这里 A，B，C，D 和 E 是内存中的任意单元，而 A 是空链（见 2.1 节）。在顺序分配的情况下，使用本表格的程序，应有一个附加的变量或常数，其值指出该表格的长度是五个条款。不然，这个信息可以在条款 5 中或在随后的单元中，以一个 "sentinel（哨兵）" 代码来说明。使用链接分配的程序，应有一个链接变量或常数指向 A，而从 A 即能找到这表的所有其它条款。

回顾 2.1 节，通常简单地以箭头来表示链接，因为其所占据的实际的存储单元通常是无所谓的。因此上边的链接表格可以图示如下：



（1）

这里 FIRST 是指向本表的头一个节点的链接变量。

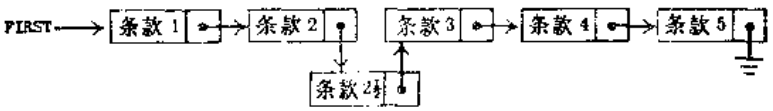
在这两种基本的存储形式之间，我们可以进行若干明显的比较：

1）链接分配为了链接而花费了额外的内存空间。在某些情况下，这可能是决定的因素。然而，我们也经常发现，一个节点中的信息无论如何也花费不了一整个的字，因此本来就提供有一个链接场的空间。而且，在许多应用中，有可能要把若干个条款组合到一个

节点中，使得对于若干条款的信息仅有一个链接（见习题 2.5-2）。但是尤其重要的是，通过链接存储的方法，经常在存储中获得隐涵的收益，因为，其诸表格可覆盖，即共享公共部分；而且，在许多情况下，顺序分配没有链接分配那样有效，除非是把颇为大量的额外的存储单元干脆空起来。例如，在上一节结尾处的讨论说明了，当稠密地装入内存时，那里描述的系统为什么必然是低效能的。

2）从一个链接的表内删去一个条款是很容易的。例如，为了删去条款 3，我们只需要改变与条款 2 相关联的链接。但对于顺序分配，则这样的一个删去，一般地意味着把表的大部分，移动到不同的位置。

3）当使用链接方案时，很容易往一个表的中间插入一个条款。例如，为了把条款 $2\frac{1}{2}$ 插入到(1)中，我们仅需要改变两个链接：



(2)

比较起来，这个操作，在一个长长的顺序表格中是极为耗费时间的。

4）在顺序的情况下，对表的随机部分之访问要快得多。当 k 是一个变量时，为了得以对表中第 k 个条款进行访问，在顺序的情况下，要花费一个固定的时间，但在链接的情况下，则要花费 k 次迭代以顺链行进到正确的位置。因此，采用链接的存储就预示着，在大多数的应用中，我们将需要顺序地，而不是随机地通过表；如果需要的是在表中间的或在底下的一些条款，则我们就应试图来保存一个附加的变量或一些链接变量的表以指出适当的位置。

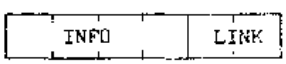
5）链接的方案使得易于把两个表合并在一起或者把一个表拆开来。

6）链接的方案本身就直接地添加了比简单的线性表更错综的结构。我们可以有可变个数的可变大小的表，这表的任何节点又可以是另一个表的一个起点；一些节点可以同时地以许多的与不同的表相应的次序链接在一起；等等。

7）在许多计算机上，象顺序地通过一个表这样的一些简单操作，对于顺序的表是要稍微快些。对于 MIX，这就是“INC1 c”与“LD1 0,1(LINK)”之间的比较，其间只不过相差一个周期；但是，许多机器不具备从一个变址单元装入一个变址寄存器的性能。

因此，我们看到，链接技术，它使我们从计算机内存连续的本性所造成的任何束缚中解放出来，它在某些操作中给了我们大量的好效能，虽则在另些情况下我们又失去了某些性能。在一特定的情况下，哪一项分配技术最适当，通常是清楚的，而且两种方法常常被用于同一个程序的不同的表中。

在以下的几个例子中，为了方便起见，我们将假定，一个节点占一个字，而且它分成两个场 INFO 和 LINK：



(3)

使用链接分配一般地都意味着，存在有某个机构，用来为一新节点寻找可利用的空着的空间，当我们希望把某个新近建立的信息插入一个表时。通常这是通过一个叫做可利用

空间表的特殊的表进行的。我们将把它叫做 AVAIL 表（或者叫做 AVAIL 堆栈，因为通常都以后进先出的方式来处理它）。所有当前未被使用的节点的集合，被一起链接于一个表中，恰象任何其它的表那样；链接变量 AVAIL 指向这个表的顶上元素。这样，如果我们要把链接变量 X 置成一个新节点的地址，并且要保留该节点供将来使用，则我们可如下进行：

$$X \leftarrow AVAIL, \quad AVAIL \leftarrow LINK(AVAIL) \tag{4}$$

这有效地取走了 AVAIL 堆栈的顶上元素，而且使 X 指向这刚被取走的节点。操作(4)是如此经常地出现，我们要对它有一个特殊的记号："$X \Leftarrow AVAIL$"，这将意味着 X 指向一个新的节点。

当删去一个节点而且不再需要它时，可以逆转进程(4)：

$$LINK(X) \leftarrow AVAIL, \quad AVAIL \leftarrow X \tag{5}$$

这一操作把由 X 编址的节点送回到"原料"表；我们以"$AVAIL \Leftarrow X$"表示(5)。

在上述关于 AVAIL 堆栈的讨论中，已省略了若干重要的事情。我们尚未说明，怎样在一程序的开头来设置它；显然这能通过下述步骤来做到：(a)把有待用作链接存储的所有节点链在一起，(b)把 AVAIL 置成这些节点的头一个的地址，以及(c)将最后的节点链到 Λ。可被分配的所有节点的集合叫做存储（联营）池。

在我们的讨论中还忽略了一个更重要的事实，这就是对溢出的检验：在(4)中我们忽略了关于所有可利用的存储空间是否已经用完了的校验。操作 $X \Leftarrow AVAIL$ 实际上应该定义如下：

如果 $AVAIL = \Lambda$，则 OVERFLOW；否则 $X \leftarrow AVAIL$，$AVAIL \leftarrow LINK(AVAIL)$ (6)

必须经常考虑溢出的可能性。这里 OVERFLOW 一般地意味着，我们要遗憾地终止程序；或者进入一个"废料收集"程序，它试图来寻找还可利用的空间。废料收集在 2.3.5 小节中进行讨论。

对于处理 AVAIL 堆栈，还有另一个重要的技术：通常我们预先并不知道有多少存储空间有待用于存储池。可能还有可变大小的顺序表格，与这链接的表格共存于内存中；在这样的情况下，我们不想让这链接的存储区域，花费比实际需要的还要多的存储空间。所以，假设我们希望把这链接存储区域设置到从 L_0 开始的递升的单元中，并假设这个区域不超过变量 SEQMIN（它表示其它表格的当前的下界）的值，则我们即可利用一个新变量 POOLMAX 而如下进行：

a) 开始置 $AVAIL \leftarrow \Lambda$ 和 $POOLMAX \leftarrow L_0$。

b) 操作 $X \Leftarrow AVAIL$ 变成如下：

"如果 $AVAIL \neq \Lambda$，则 $X \leftarrow AVAIL$，$AVAIL \leftarrow LINK(AVAIL)$。

否则置 $POOLMAX \leftarrow POOLMAX + c$，其中 c 是节点大小； (7)

现在如果 $POOLMAX > SEQMIN$，则 OVERFLOW；否则置 $X \leftarrow POOLMAX - c$。"

c) 当程序的其它部分试图减小 SEQMIN 的值时，如果 $SEQMIN < POOLMAX$，则它们应当发出 OVERFLOW 警报。

d) 操作 $AVAIL \Leftarrow X$ 同(5)一样不变。

这个思想实际上与以前的方法差别甚微，只是以一个特殊的矫正过程代替（6）中OVERFLOW的情况而已。其纯效果就是要保持尽可能小的存储池。许多人都喜欢使用这一思想，甚至当所有的表都占据存储池区域时（因此SEQMIN是常数），因为它避免了开始时把所有可利用的单元链接在一起的相当花费时间的操作，而且它有时又便利了调试。

我们现在看到，以这样的一种方式，使之能有效地找到自由节点并在随后返回于池中，来维持一个可利用节点"池"，是十分容易的。这些方法给了我们用作链接表格的原料的源泉。我们的讨论是以所有的节点都有一个固定的大小c这个含蓄的假定为前提的。当出现有不同的节点大小的那些情况，是非常重要的，但我们将把这项讨论一直推迟到2.5节。现在，我们来考虑在涉及堆栈和排队的特殊情况下，一点最普通的表操作。

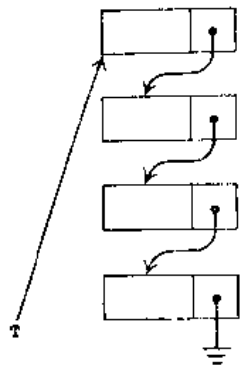


图5 链接的堆栈

堆栈是最简单的链接表。图5图示了一个典型的堆栈，带有一个指向堆栈顶上的指针T。当堆栈是空的时，这个指针将有值Λ。

用一个辅助的指针变量P，则怎样把信息Y插入（"下推"）到堆栈的顶上，就显然了：

$$P \Leftarrow AVAIL,\ INFO(P) \leftarrow Y,\ LINK(P) \leftarrow T,\ T \leftarrow P \qquad (8)$$

反过来，为把Y置成等于堆栈顶上的信息，并"弹出"堆栈：

如果$T = \Lambda$，则UNDERFLOW；

否则置$P \leftarrow T$，$T \leftarrow LINK(P)$，$Y \leftarrow INFO(P)$，$AVAIL \leftarrow P$ （9）

应当把这些操作与顺序分配的堆栈的类似机构，即2.2.2小节中的(2α)和(3α)，作一比较。读者应当仔细地研究(8)和(9)，因为它们是极为重要的操作。

在考察排队的情况之前，让我们来看看，怎样以MIX的程序方便地表达这些操作。用于插入的程序，以P＝rI1，可写成如下：

```
INFO  EQU  0:3          (定义INFO场)
LINK  EQU  4:5          (定义LINK场)
      LD1  AVAIL        P←AVAIL
      JIZ  OVERFLOW     AVAIL=Λ?
      LDA  0,1(LINK)                    } P←AVAIL
      STA  AVAIL        AVAIL←LINK(P)
      LDA  Y
      STA  0,1(INFO)    INFO(P)←Y
      LDA  T
      STA  0,1(LINK)    LINK(P)←T
      ST1  T            T←P
```

(10)

这花费17个周期，对照之下，对于顺序的表格，相应的操作则是12个周期（尽管在顺序的情况下，OVERFLOW在许多情况下将拖得相当长）。在这个程序中，如同这一章随后的其它程序中那样，OVERFLOW表示或是一个结束程序，或是一个再寻找空间并返回到单元(rJ)－2的子程序。

为进行删去的程序是同样简单的：

```
LDA   T              P←T
JIZ   UNDERFLOW      T=Λ?
LDA   0,1(LINK)
STA   T              T←LINK(P)
LDA   0,1(INFO)                                        (11)
STA   Y              Y←INFO(P)
LDA   AVAIL                                  }
STA   0,1(LINK)      LINK(P)←AVAIL           } AVAIL←P
STJ   AVAIL          AVAIL←P                 
```

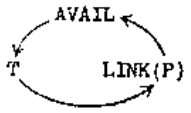
观察一下，这些操作的每一个都包含着三个链的一个循环排列。这是很有趣的。例如，在插入操作中，设P是插入之前AVAIL的值；如果P≠Λ，则我们发现在这个操作之后：

AVAIL的值已经变成LINK(P)的以前的值，

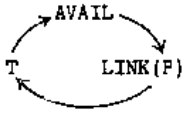
LINK(P)的值已经变成T以前的值；而且

T的值已经变成AVAIL以前的值。

所以插入过程(除了设置INFO(P)←Y之外)是循环排列



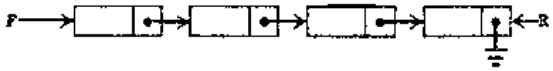
类似地，在删去的情况下，其中P有操作前T的值，而且我们假定P≠Λ，则我们有Y←INFO(P)和



在这些图式中，排列是循环的这一提法，实际上并不是一个中肯的论点。因为，三个元素的任何排列，凡是移动每个元素的，必是循环的。重要之点勿宁是，恰恰有三个链排列在这些操作中。

以上的插入和删去算法，已被描述成用于堆栈。但是，它们可更一般得多地应用于任何线性表的插入和删去。例如，恰恰在由链接变量T指出的节点之前实施插入。上述的条款2——的插入(见(2))将通过使用带有T=LINK(LINK(FIRST))的操作(8)来进行。

链接分配应用于排队，有一种特别方便的方式。在这种情况下，容易看出，这些链接应当从排队的前头向着后尾进行，以便当从前头取走一个节点时，直接地就可描述新的前头节点。对于前头和后尾，我们将使用指针F和R：



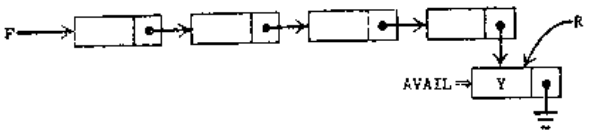
(12)

除了R之外，抽象地看来，这一图式就等同于本章上的图5。

每当设计了一个表的型式之后，重要的是仔细地说明所有的条件，特别是对于表为空时的情况。对于表为空时的情况不能适当地来进行工作，乃是有关链接分配方面所遇到的最普遍的程序设计错误之一。另一个普遍的错误是当正在加工此结构时，忘记了设置所有

的链接。为了避免前一种类型的错误，总要细心地检查"边界条件"。为了避免造成第二种类型的错误，画出"之前和之后"的图式并将它们进行比较，是有帮助的，为的是看看必须改变哪些链接。

让我们通过把它们应用于排队的情况，来说明上一段的论述。首先考虑插入操作：如果(12)是插入之前的状态，则在排队的后尾插入之后，其图式应该是

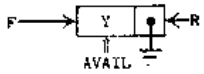


(13)

(这里所使用的记号意味着一个新节点是从AVAIL表得到的)。比较(12)和(13)，我们就看出了，当把信息Y插入到排队的后尾时，我们应该怎样来进行：

$$P \leftarrow AVAIL,\ INFO(P) \leftarrow Y,\ LINK(P) \leftarrow A,\ LINK(R) \leftarrow P,\ R \leftarrow P \qquad (14)$$

现在让我们来考虑当排队为空队时的"边界"状态：在这种情况下，插入之前的状态尚有待确定，而"之后"的状态是



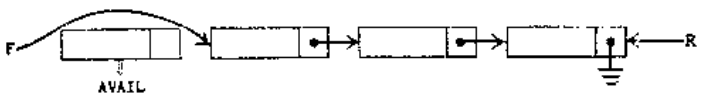
(15)

也希望操作(14)能应用于这种情况，即使得插入到一个空的排队意味着：我们必须既改变F也改变R，而不仅仅是R。我们发现，假定F=LINK(LOC(F))，当排队为空时，如果R=LOC(F)，则(14)将正确地进行工作；如果按这个想法来作，则变量F的值必须存于其单元的LINK场中。为了对于空排队进行尽可能有效的检验，在这种情况下，我们将命F=A。因此我们的政策是

一个空排队以F=A和R=LOC(F)表示

在这些情况之下，如果应用操作(14)，则我们得到(15)。

以类似的方式，可以推导出对于排队的删去操作。如果(12)是删去之前的状态，则删去之后的状态是



(16)

对于边界条件，我们必须确保，当排队在操作之前或之后为空时，删去操作皆得以进行，这些考虑导致我们以如下的方式，一般地来进行一个删去：

如果F=A，则UNDERFLOW；
否则置P←F，F←LINK(P)，Y←INFO(P)，AVAIL←P，
而且如果F=A，则置R←LOC(F)。 (17)

注意当排队变成空队时，R必须加以改变。这恰恰就是我们总须注意观察的"边界条件"的类型。

上述的提议，并不是以线性链接的方式来表示排队的唯一的方式；在这一章的稍后部分我们将给出其它的方法。其实，上边的操作，并没有被规定为做某件事的唯一的方式；而只是打算把它们作为一些用链接表进行操作的基本手段的例子。读者只要对于这样的技

术已有一点预先的经验，就能发现，在往下阅读之前，重新读读这一小节直到这里为止，将是有益的。

在这一章到此为止，我们已经讨论了一些对于表格实施某些操作的方法。但是，我们的讨论在这样的意义下还总是"抽象的"，即我们始终未展示出实际的程序来，说明在其中这些具体的技术是有用的。人们在还没有看够问题的特殊情况以唤起他的兴趣之前，是不能被诱导来研究一个问题的抽象性方面的。至今所讨论的操作（通过插入和删去对可变大小的信息表的处理，以及诸如堆栈或排队之表格的使用）有着如此广泛的应用，以致可望读者将在他自己的程序中足够经常地遇见它们，使得他已经欣然地承认了它们的重要性。但是，现在将离开抽象的领域，而开始来研究这一章之技术的一系列有意义的实例。

我们的头一个例子是所谓拓扑分类问题，它是在网络问题方面需要的一个重要过程，在所谓的 PERT 图方面，甚至在语言学方面也需要这个过程。事实上，每当我们有着一个涉及偏序的问题时，它就可能有用。一集合 S 的一个"偏序"是 S 的对象之间的一个关系，我们可以用符号"$\prec$"来表示这个关系，对于 S 中的任何对象 x，y，和 z（不必是不同的），它满足以下的性质：

i）如果 $x \preceq y$，且 $y \preceq z$，则 $x \prec z$。（传递性）

ii）如果 $x \preceq y$，且 $y \preceq x$，则 $x = y$。（反对称性）

iii）$x \preceq x$。（自反性）

记号 $x \preceq y$ 可以读作"x 前于或等于 y"。如果 $x \preceq y$ 且 $x \neq y$，则我们写 $x \prec y$，且说"x 前于 y"。由(i)，(ii)和(iii)容易看出，我们总有

i′)若 $x \prec y$，且 $y \prec z$，则 $x \prec z$。（传递性）

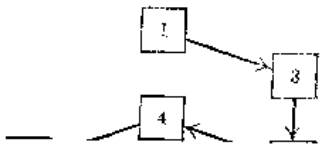
ii′)若 $x \prec y$，则 $y \not\prec x$。（非对称性）

iii′)$x \not\prec x$。（非自反性）

以 $y \not\prec x$ 表示的关系意味着"y 不前于 x"。如果我们以满足性质(i′)、(ii′)和(iii′)的关系 $\prec$ 开始，我们可以把上述过程倒过来，并且定义若 $x \prec y$ 或 $x = y$ 则 $x \preccurlyeq y$；那么，性质(i)，(ii)和(iii)皆为真。因此，我们可以或者把性质(i)，(ii)、(iii)或者把性质(i′)、(ii′)、(iii′)作为偏序的定义（注意性质(ii′)实际上是(i′)和(iii′)的一个推论）。

偏序经常地出现于日常生活以及数学中。作为来自数学的一些例子，我们可以举出实数 x 和 y 之间的关系 $x \leqslant y$；对象集合之间的关系 $x \subseteq y$；正整数之间的关系 $x \setminus y$（x 整除 y）。在 PERT 网络的情况下，S 是必须完成的一批作业，而且关系"$x \prec y$"意味着"x 必须在 y 之前完成"。

我们自然地将假定 S 是一个有穷集合，因为我们要用它在一台计算机之内来进行工作。对于一个有穷集合上的一个偏序关系，总能画出一个如图 6 这样的图式来加以说明，其中小方




```
LDI   T              P ← T
JIZ   UNDERFLOW      T = Λ?
LDV   0,1(LINK)
STA   T              T ← LINK(P)
```

拓扑分类的问题，就是要"把偏序嵌入到一个线性次序中去"，即是，把对象排列成线性序列 a_1，a_2，…，a_n，使得每当 $a_j \prec a_k$ 时，我们都有 $j < k$。从图形上看，这意味着这些方框重新排列成一行，使得所有的箭头都指向右方（见图 7）。这样一种重新排列在每一情况下都是可能的，这一点并不是显然的；然而如果存在任何的回路，则这样一种重新排列肯定地不可能进行。因此，我们将要给出的算法是有趣的，这不仅仅因为它完成一个有用的操作，而且还因为它证明了，对于每个偏序，这个操作都是可能的。

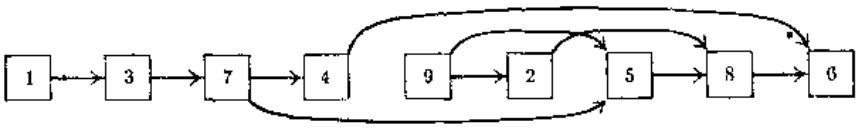


图 7　经拓扑分类之后图 6 的次序关系

作为拓扑分类的一个例子，考虑一个庞大的收录技术术语定义的词汇表。如果词 w_1 的定义直接或间接地依赖于词 w_2，则我们可以写 $w_2 \prec w_1$。假定没有"循环的"定义，则这个关系就是一个偏序。在这种情况下拓扑分类的问题，就是来寻找一种排列词汇表中的诸术语的方式，使得一个术语在业已定义之前，不被使用。在编写程序来处理某汇编语言或编译程序语言中的说明时，也有类似的问题；在编写一本描述一个计算机语言的用户手册或编写关于信息结构的教科书时，也出现这些问题。

有一个非常简单的进行拓扑分类的方法：我们开始取一个对象，在本次序下它处于任何其它对象之前。先输出这个对象。现在我们从集合 S 取走这个对象；而所得的集合仍然是有偏序的，而且这一过程可以一直重复到整个集合都分类完了为止。例如，在图 6 中，我们可以从取走 1 或 9 开始；在取走了 1 之后，可以取 3、等等。这个算法可能失败的唯一情况是，如果有一个非空的偏序集合，在这个集合中，每个元素都有另一个元居其前；因为，在这样的情况下，这个算法将发现无从下手。但如若每个元素果然都有另一个元素居其前，则我们将构造出一个任意长的序列 b_1，b_2，b_3，…，其中 $b_{j+1} \prec b_j$；因为 S 是有穷的，故对于某个 $j < k$，我们必然有 $b_j = b_k$，而这就意味着 $b_k \preceq b_{j+1}$，与(ii)矛盾。

为了有效地用计算机来实现这一过程，我们需要准备来实施以上所描述的那些动作，即是，来定出不被任何其它对象居前的那些对象，并且从该集合中取走它们。我们的实现也受到所希望的输入和输出特征的影响。最一般的程序，将接受对象的字符名称，并将允许对大量的对象来进行分类——比一次可能填入计算机内存的还多。然而，这些复杂性会掩盖我们在这里试图要做的主要之点：字符数据的处理，通过使用第 6 章中的方法，即可有效地进行；而大型网络的处理，可留作读者的有趣的设计。

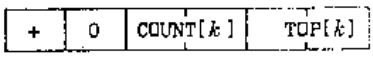
因此，我们将假定有待分类的对象，已任意地编号成从 1 到 n。对于程序的输入将在带设备 1 上；每个带记录包含 50 个数对，其中数对 (j, k) 意味着对象 j 前于对象 k。然而，头一个数对是 $(0, n)$，其中 n 是对象的个数。数对 $(0, 0)$ 结束本输入。我们将假定，n 加上关系的数对个数将称心地适合于内存；而且，我们将假定不需要校验输入的正确性。输出将是：在带设备 2 上，在分类次序下对象的号数后边按以数 0。

作为输入的一个例子，我们可以有数对：

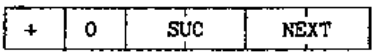
$$\begin{array}{cccc} 9\prec 2 & 3\prec 7 & 7\prec 5 & 5\prec 8 \\ 8\prec 6 & 4\prec 6 & 1\prec 3 & \\ 7\prec 4 & 9\prec 5 & 2\prec 8 & \end{array} \tag{18}$$

没有必要给出比所需要的还要多的数对来表征所希望的部分次序。于是，象 $9\prec 8$ 这样额外的关系（它能从 $9\prec 5$ 和 $5\prec 8$ 推导出来），可以无妨地从输入中省略掉或者加到输入中去。一般地说，仅仅需要给出对应于象图 6 那样图式上箭头的数对。

下面的算法使用一个顺序的表格 X[1], X[2], …, X[n]，而且每个节点 X[k] 都有形式



这里 COUNT[k]是直接居于对象 k 之前者的个数（即是，在输入中出现的 $j\prec k$ 的数对的个数），而 TOP[k]是指向对象 k 的直接后继者的表的开始处的链接，这后一表包含有格式为



的条款，其中 SUC 是 k 的一个直接后继者，而 NEXT 是这表的下一条款。作为这些约定的一个例子，图 8 示出了对应于输入(18)的内存示意的内容。

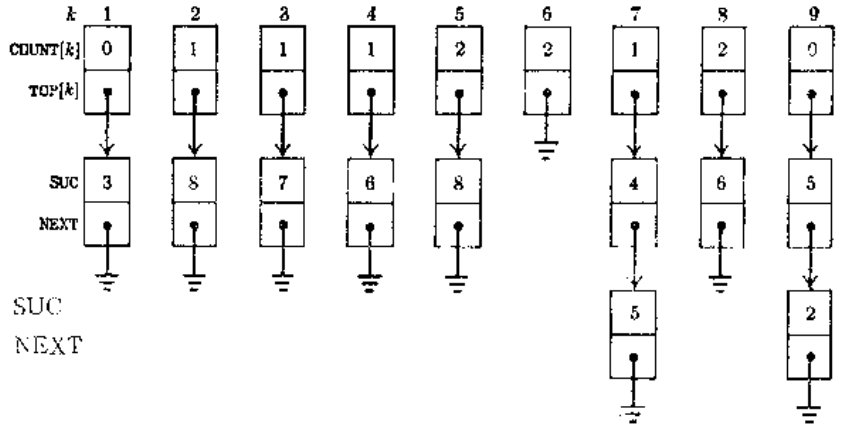


图 8　对应于关系 (18) 的图 6 的计算机表示

利用这一内存安排，不难来作出算法。问题就是来输出其 COUNT 场为 0 的那些节点，然后将这些节点的所有后继者的 COUNT 场减 1。这个技巧是要避免进行对 COUNT 场为 0 的那些节点的"检索"，而且这可以通过保持一个包含其 COUNT 场已经减小到 0 但尚未被输出的那些节点的排队来进行。对于这个排队的链接，就保存在 COUNT 场内；但 COUNT 场直到现在为止已经用作原先的目的，因此，为明瞭起见，在以下的算法中，当 COUNT[k]场已不再被用来保存一个计数时，我们就使用记号 QLINK[k]来表示 COUNT[k]。

算法 T(拓扑分类)　这个算法输入关系的数对"$j\prec k$"，$1\leqslant j$，$k\leqslant n$，它表示在某个偏序下，对象 j 前于对象 k。输出是嵌入线性次序的对象集合。所用的内部表格有：QLINK[0], COUNT[1]=QLINK[1], COUNT[2]=QLINK[2], …, COUNT[n]

=QLINK[n]；TOP[1]，TOP[2]，…，TOP[n]；一个对于每一个输入关系都有一个节点，并且都有SUC和NEXT场的存储池如上所示；P，一个用来访问存储池中的节点的链接变量；F和R，用来访问一个其链接是在QLINK表格内的排队的前头和后尾的整数值变量；另有N，用来计算还有多少个对象有待输出的变量。

T1. 〔初始化〕 输入n的值。置COUNT[k]←0和TOP[k]←Λ，对于$1\leqslant k\leqslant n$。置N←n。

T2. 〔下一个关系〕 从输入中取下一个关系"$j\prec k$"；但是，如果输入已经穷尽，则转到T4。

T3. 〔记录此关系〕 COUNT[k]增加1。置P←AVAIL，SUC(P)←k，NEXT(P)←TOP[j]，TOP[j]←P。〔这就是操作(8)。〕转到T2。

T4. 〔扫描0〕 （至此，我们已经完成了输入阶段；输入(18)现在已经转换成如图8所示的计算机表示。现在我们来初始化输出的排队，该排队是通过QLINK场链接起来的）。置R←0和QLINK[0]←0。对于$1\leqslant k\leqslant n$，检查COUNT[k]，而且如果它为0，则置QLINK[R]←k和R←k。在对于所有的k完成这件事后，置F←QLINK[0]（它将包含头一个遇到的使COUNT[k]为0的k值）。

T5. 〔输出排队的前头〕 输出F的值。如果F=0，则转到T8；否则，置N←N−1，并置P←TOP[F]（因为QLINK和COUNT的表格重迭，所以我们有QLINK[R]=0；因此当排队为空时，即出现条件F=0）。

T6. 〔抹去关系〕 如果P=Λ，则转到T7。否则COUNT[SUC(P)]减小1，而且如果因此它已经减成0，则置QLINK[R]←SUC(P)和R←SUC(P)。置P←NEXT(P)并重复这一步骤（对于某个k，我们正在从系统取走所有形如"F$\prec k$"的关系，而且当节点的所有后继者业已输出时，就把这新的节点放进排队中）。

T7. 〔从排队取走〕 置F←QLINK[F]并转回到T5。

T8. 〔过程结束〕 本算法终止。如果N=0，则我们已经以所希望的"拓扑次序"输出所有的对象号数，后边接以一个0。否则，尚未输出的N个对象之号数包含一个回路，与部分次序的假设相违（对于打印出一个这样的回路的内容的算法，见习题23）。∎

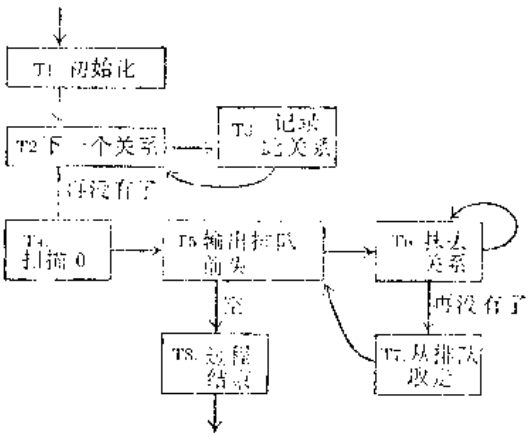


图9 拓扑分类

读者会发觉，亲手对输入(18)来试验这个算法，是有益的。算法T说明了一个在顺序存储与链接存储技术之间的有趣的相互作用。对于"主"表格X[1]，…，X[n]，使用的是顺序存储，这些节点包含了COUNT[k]和TOP[k]的项目；因为，我们要在步骤T3中进行对这个表格的"随机"部分的访问（然而，如果输入的是字符，则为了更快的检索，将使用另一种类型的表格，如象在第6章那样）。对于"直接后继者"表，使用的是链接存储；因为，这些表的条款在输入中是以随机的顺序进来的。等待输出之节点的排队，

通过把这些节点以输出的顺序链接在一起，而驻留在顺序的主表格当中。这个链接是通过表格索引而不是通过地址来进行的。即是说，当排队的前头是X[k]时，我们有F = k 而不是F = LOC(X[k])。在步骤T4，T6和T7中所使用的排队操作与在(14)及(17)中的那些，并不是等同的，因为我们利用了这个系统中的排队的特殊性质；在算法的这个部分，没有节点需要建立或者恢复到可利用的空间中。

把算法T编成MIX汇编语言代码，有几个有趣之点。由于在这个算法中不需要从表格中删去(因为不必释放存储供以后使用)，P←AVAIL操作可以以极其简单的方式来进行，如象在以下的行19和32中所示的那样；我们不必保存任何链接的存储池，我们可以逐次地选择新的节点。这个程序包括了：按照上面提出的约定，通过磁带进行的完整的输入和输出；但为了简便起见，未示出缓冲技术。读者不应该为领会这个程序中的编码的细节而感到非常困难，因为它直接地对应着算法T。这里说明了变址寄存器的有效用法，这是链接存储处理的一个重要方面。

程序T(拓扑分类)。在这个程序中，应该注意以下的等价式：r16≡N，r15≡缓冲区指针，r14≡k，r13≡j及R，r12≡AVAIL及P，r11≡F，TOP[j]≡X + j (1:5)，COUNT[k]≡QLINK[k]≡X + k(2:3)。

```
01    *BUFFER AREA AND FIELD DEFINITIONS
02    COUNT   EQU   2:3                          场的符号名称的定义
03    QLINK   EQU   2:3
04    TOP     EQU   4:5
05    SUC     EQU   2:3
06    NEXT    EQU   4:5
07    TAPEIN  EQU   1                            输入是在带设备1上
08    TAPEOUT EQU   2                            输出是在带设备2上
09    BUFFER  ORIG  *+100                        带缓冲区区域
10            CON   -1                           缓冲区结束的哨兵
11    *INPUT PHASE
12    TOPSORT IN    BUFFER(TAPEIN)   1           T1. 初始化。读入头一个
13            JBUS  *(TAPEIN)                    带区，等候完成
14    1H      LD6   BUFFER+1         1           N←n
15            ENT4  0,6              1
16            STZ   X,4              n+1         置COUNT[k]←0，TOP[k]←k
17            DEC4  1                n+1         对于0≤k≤n
18            J4NN  *-2              n+1         (在步骤T4中予置QLINK[0]←0)
19            ENT2  X,6              1           可利用的存储在X[n]之后开始
20            ENT5  BUFFER-2         1           准备读头一数对(j, k)
21    2H      LD3   0,5              m+b         T2. 下一个关系
22            J3P   3F               m+b         j>0?
23            J3Z   4F               b           输入已穷尽?
24            IN    BUFFER(TAPEIN)   b-1         发现哨兵，读另一带区，等候完成
25            JBUS  *(TAPEIN)
26            ENT5  BUFFER           b-1         恢复缓冲区指针
```

```
27                  JMP    2B                 b - 1
28      3H          LD1    1,5                m          T3. 记录此关系
29                  LDA    X,4(COUNT)         m          COUNT(k)
30                  INCA   1                  m          + 1
31                  STA    X,4(COUNT)         m          →COUNT(k)
32                  INC2   1                  m          AVAIL←AVAIL + 1
33                  LDA    X,3(TOP)           m          TOP(j)
34                  STA    0,2(NEXT)          m          →NEXT(P)
35                  ST1    0,2(SUC)           m          k →SUC(P)
36                  ST2    X,3(TOP)           m          P →TOP(j)
37                  INC5   2                  m          缓冲区指针增值
38                  JMP    2B                 m
39      4H          IOC    0(TAPEIN)          1          重绕磁带
40                  ENT4   0,6                1          T4. 扫描 0
41                  ENT5   -100               1          为输出恢复缓冲区指针
42                  ENT3   0                  1          R ← 0
43      4H          LDA    X,1(COUNT)         n          检查 COUNT(k)
44                  JAP    *-3                n          它非 0?
45                  ST4    X,3(QLINK)         a          QLINK(R)← k
46                  ENT3   0,4                a          R ← k
47                  DEC4   1                  n
48                  J4P    4B                 n          n - k ≥ 1
49      *SORTING PHASE
50                  LD1    X(QLINK)           1          F←QLINK(0)
51      5H          JBUS   *(TAPEOUT)                    T5. 输出排队前头
52                  ST1    BUFFER,5(100:100)  n + 1      把 F 存入缓冲区区域
53                  J1Z    8F                 n + 1      F 为 0?
54                  INC5   1                  n          推进缓冲区指针
55                  J5N    *+3                n          检验缓冲区已满否。
56                  OUT    BUFFER(TAPEOUT)    c ← 1      若然，则输出一个带区
57                  ENT5   -100               c - 1      恢复缓冲区指针
58                  DEC6   1                  n          N←N - 1
59                  LD2    X,1(TOP)           n          P←TOP(F)
60                  J2Z    7F                 n          T6. 拆去关系
61      6H          LD4    0,2(SUC)           m          rI4← SUC(P)
62                  LDA    X,4(COUNT)         m          COUNT(rI4)
63                  DECA   1                  m          - 1
64                  STA    X,4(COUNT)         m          →COUNT(rI4)
65                  JAP    *+3                m          已经达到 0?
66                  ST4    X,3(QLINK)         n - a      若然，则置 QLINK(R)←rI4
67                  ENT3   0,4                n - a      R ← rI4
68                  LD2    0,2(NEXT)          m          P ← NEXT(P)
69                  J2P    6B                 m          若 P≠A，则重复
```

```
79    710    LDI   X,1(QLINK)      n     T7. 从排队取走
71           JMP   5B               n     F←QLINK(F)，转到T5。
72    8H     OUT   BUFFER(TAPEOUT)  1     T8. 过程结束
73           IOC   0(TAPEOUT)       1     输出最后带区并重绕磁带
74           HLT   0,6              1     停机，在控制台上显示N
75    X      END   TOPSORT                 表格区域开头                    ■
```

借助于克希霍夫定律来分析算法T，是十分简单的：执行时间有近似形式 c_1m+c_2n，其中 m 是输入关系的个数，n 是对象的个数，且 c_1 和 c_2 是常数。对于这个问题，难以想象一个更快的算法了！通过上边的程序T，给出了本分析中的精确的数量，其中 a = 无后前者的对象的个数，b = 在输入中带记录的个数 = $\lceil(m+2)/50\rceil$，以及 c = 在输出中带记录的个数 = $\lceil(n+1)/100\rceil$。除开输入/输出操作，在这种情况下总共的运行时间仅为 $(32m+24n+7b+2c+16)u$。

阿·B·卡恩(A. B. Kahn)首先发表了类似于算法T（但没有排队链接的重要特性）的一个拓扑分类技术，见 *CACM* 5 (1962)，558～562。偏序之拓扑分类总是可能的这一事实是由爱·斯皮尔拉因(E. Szpilrajn)在《数学基础》(Fundamenta Mathematica)16(1930)，386～389页中首先证明发表的；他提到，这个结果已为他的若干同事所知道。

不论算法T怎样有效，在7.4节中我们将看到关于拓扑分类的一个更好的算法。

习题

▶1.〔10〕 对由一堆栈弹出的操作(9)，提到了UNDERFLOW的可能性；为什么压入一堆栈的操作(8)不提OVERFLOW的可能性？

2.〔22〕 写出一个为进行插入操作(10)的“通用的”MIX子程序。这个子程序应该有以下的说明（参照1.4.1小节）：

调用序列：JMP INSERT　跳到子程序
　　NOP T　　指针变量的单元

进入条件：rA＝有待放进一个新节点之INFO场的信息。

出口条件：其指针为链接变量T的堆栈，在顶上有此新节点；rI1＝T；rI2，rI3被改变。

3.〔22〕 写出一个进行删去操作（11）的“通用的”MIX子程序，这个子程序应该有如下的说明：

调用序列：JMP　DELETE　　转到子程序
　　NOP　T　　指针变量的单元
　　JMP　UNDERFLOW　头一个出口，如果发现UNDERFLOW。

进入条件：无

出口条件：如果以链接变量T为指针的堆栈是空的，则采用头一个出口；否则删去此堆栈的顶上节点，而且返回到“JMP DELETE”之后的第三个单元处。在后一情况下，rI1＝T，而rA是该被删去节点的INFO场的内容。无论在那一种情况下，rI2和rI3都为这个子程序所使用。

4. [22] 在(10)中的程序，如同在(6)中给出的那样，是以操作P⇐AVAIL为基础的。说明如何来写一个OVERFLOW子程序，使得无须对编码(10)作任何改变，操作P⇐AVAIL就能利用SEQMIN，如象由(7)给出的那样。为了通用，你的子程序不应改变任何寄存器的内容，可能比较指示器要除外；而且它应该从单元(rJ−2)出去，而不是通常的(rJ)。

▶5. [24] 操作(14)和(17)给出了一个排队的效果；说明怎样来定义进一步的操作"在前头插入"，以便得到一个输入受限的双排队的所有动作。怎样能来定义"从后尾删去"的操作(使得我们有一个一般的双排队)?

6. [21] 在操作(14)中，我们置LINK(P)⇐A，然而就是在排队后尾的下一个插入即将改变这同一个链接场的值。试说明，如果我们改动一下在(17)中关于"P=A"的检验，则怎样能去掉(14)中对LINK(P)的设置。

▶7. [23] 设计一个算法，来"颠倒"一个如象(1)那样的链接线性表，即是，来改变它的链接，使得条款以相反的次序出现。(于是，如果表(1)已被颠倒，则我们将有FIRST链接到含有条款5的节点；该节点将链接到含有条款4的节点；等等。)假定这些节点有(3)的形式。

8. [24] 写出一个关于习题7的问题的MIX程序，并力求使你的程序运行得尽可能地快。

9. [20] 下列关系中哪一个是在指定的集合S上的偏序关系：[注意：如果以下定义的是关系"$x \prec y$"，则打算要定义关系"$x \preceq y \equiv (x \prec y$或$x = y)$"，而后判定$\prec$是否为偏序]。(a) S=所有有理数，$x \prec y$意味着$x > y$？(b) S=所有的人，$x \prec y$意味着x是y的祖先？(c) S=所有整数，$x \preceq y$意味着x是y的一个倍数(即是$x \bmod y = 0$)？(d) S=本书中所证明的所有数学结果，$x \prec y$意味着y的证明依赖于x的正确性？(e) S=所有的正整数，$x \preceq y$意味着$x + y$是偶数？(f) S=一些子程序的集合，$x \prec y$意味着"x调用y"，即是，当x在操作时y可能也在操作，假定不允许递归。

10. [M21] 给定"$\subset$"为一个满足偏序的性质(i)和(ii)的关系；证明：由规则"$x \preceq y$当且仅当$x = y$或$x \subset y$"定义的关系"$x \preceq y$"，满足偏序的全部三个性质。

▶11. [24] 拓扑分类的结果并不总是完全确定的，因为可能有若干个方式来排列这些节点并满足拓扑次序的条件。试找出把图6中的节点排列成拓扑次序的所有可能的方式。

12. [M20] n个元素的集合有2^n个子集，而且这些子集对于集合包含关系来说是偏序的。试给出把这些子集排列成为拓扑次序的两种有趣的方式。

13. [M48] 有多少可能的方式，来把习题12中所述的2^n个子集排列成拓扑次序？(以n的一个函数的形式来给出答案)。

14. [M24] 一个集合的一个线性次序，是一个满足附加条件

(iv) 对于S中的任何两个对象x和y，或$x \preceq y$，或$y \preceq x$，二者必居其一

的偏序。试从这个所给的定义，直接地证明：一个拓扑分类可以得到唯一可能的输出，当且仅当，关系$\preceq$是一个线性次序。(你可以假定集合S是有限的。)

15. [M25] 试证：对于一个有限集合S上的任何偏序，都有唯一的"非冗余的"关系的对偶之集合[如象对应于图6的(18)]——这个集合表征了这个次序。当S是一个无穷

集合时，这同一事实也是真的吗？

16.〔*M*22〕 给定一个集合 $S=\{x_1, \cdots, x_n\}$ 上的任何偏序，我们能构造它的“关联矩阵”(a_{ij})，其中 $a_{ij}=1$ 如果 $x_i \leq x_j$，否则 $a_{ij}=0$。试证，有一个方式来排列这个矩阵的行和列，使得对角线以下的所有元素都为 0。

▶17.〔21〕 如果算法 T 以输入（18）来体现，则它将产生什么输出？

18.〔20〕 当算法 T 终止时，QLINK[0]，QLINK[1]，…，QLINK[n]之值的意义是什么，如果有的话？

19.〔18〕 在算法 T 中，我们来检查步骤 T5 中排队的前头位置，但直到步骤 T7 之前并不从排队中取走这个元素。如果我们在步骤 T5 的结尾处而不是在 T7 中，置 F←QLINK[F]，则将发生什么情况？

▶20.〔24〕 算法 T 使用 F，R 和 QLINK 表格来得到一个排队的效果，这个排队包含其 COUNT 场已经变成 0，但是其后继者的关系还未被取走的那些节点。一个堆栈是否可用于此目的以代替一个排队？如果可以，试把得到的算法与算法 T 作一比较。

21.〔21〕 如果关系“$j \prec k$”之一在输入中被重复若干次，则算法 T 是否仍将实现一个正确的拓扑分类？如果输入包含有形如“$j \prec j$”的一个关系又怎样？

22.〔23〕 程序 T 假定了它的输入带含有正确的信息，但是一个打算通用的程序，对于它的输入总要进行仔细的检验，使得能够发现笔误，而且程序不致“自损”。例如，如果对于 k，输入关系之一为负，则程序 T 当存入 X[k]时可能错误地改变它自己的指令之一。试提出一些修改程序 T 的方式，使得它适合于普遍使用。

▶23.〔27〕 当拓扑分类算法由于在输入中发现有一条回路（见步骤 T8）而不能进行时，停机并说“有一条回路”，通常是无济于事的。打印出这些回路之一，借以说明有错误的输入部分，这是有帮助的。试扩充算法 T，使当必要时它将对这条回路进行这种打印〔提示：正文中给出了在步骤 T8 处，当 N > 0 时，存在一条回路之证明；该证明提示了一个算法〕。

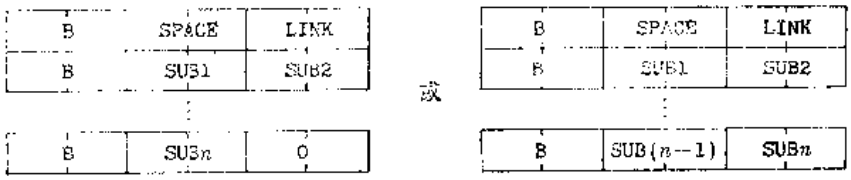
24.〔21〕 把在习题 23 中所作的对于算法 T 的扩充加入到程序 T 中。

25.〔47〕 设计一个尽可能高效的算法，来对于非常大的集合 S 进行拓扑分类，这个集合含有比计算机内存所能容纳的数量还要大得多的节点。假定输入，输出和临时工作空间都用磁带〔可能的提示：对输入作习常的分类，我们即不妨假定，对于一个给定的节点，其所有关系都在一起出现。但之后，可以做什么？（特别是，我们必须考虑最坏的情况；在这种情况下，给定的次序已经是广泛地排列的线性次序；如果可能，则我们要避免进行对于整条数据带的 $O(n)$ 次迭代）〕。

26.〔29〕 （子程序分配）假设我们有一条含有一台计算机装置的主要“子程序库”的带，这些子程序以浮动形式出现。装入程序要来确定对于每个子程序所用的代真的数量，以便它能通过扫描一趟此带，就装入所需要的程序。问题是某些子程序要求其它一些子程序也在内存之中。不常使用的子程序（它靠近带的末端出现）可能调用经常使用的子程序（它靠近带的开头出现），而且在扫过这条带以前，我们要知道所有需要的子程序。

着手处理这个问题的一个方式，是要有填于内存的一个“带目录”。装入程序要访问两个表格：

a）带目录。这个表格是由如下形式的可变长度的节点组成的：



其中 SPACE 是该子程序所需要的内存字数；LINK 是一个指向紧跟在这个子程序之后出现于带上的子程序的目录条款的链接；SUB1，SUB2，…，SUBn（$n \geqslant 0$）是指向为这一个子程序所需要的任何其它子程序的目录条款的链接；除了最后一个字外，对于所有的字，$B=0$，在一个节点的最后一个字上 $B=-1$。对于库带上的头一个子程序的目录条款的地址，由链接变量 FIRST 确定。

b）有待装入的为程序直接访问的子程序表。这被存储在连续的单元 X〔1〕，X〔2〕，…，X〔N〕中，其中 N≥0 是一个为装入程序所已知的变量。这个表中的每一条款，是一个指向所希望的子程序的目录条款的链接。

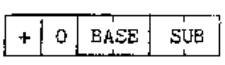
装入程序也知道 MLOC，这就是有待头一个被装入的子程序使用的代真的数量。

作为一个小例子，考虑以下的配置：

	带目录			所需要的子程序表
	B	SPACE	LINK	X〔1〕=1003
1000：	0	20	1005	X〔2〕=1010
1001：	−1	1002	0	
1002：	−1	30	1010	
1003：	0	200	1007	
1004：	−1	1000	1006	
1005：	−1	100	1003	
1006：	−1	60	1000	N=2
1007：	0	200	0	FIRST=1002
1008：	0	1005	1002	MLOC=2400
1009：	−1	1006	0	
1010：	−1	20	1006	

在这种情况下的带目录说明了，带上的子程序依次为 1002，1010，1006，1000，1005，1003，和 1007。子程序 1007 需用 200 个单元而且要求使用子程序 1005，1002，和 1006，等等。有待装入的程序需要子程序 1003 和 1010，它们有待放置到≥2400 的单元中去。这些子程序又依次要求 1000，1006，和 1002 也必须被装入。

子程序分配程序是来改变X表格，使得每个条款X〔1〕，X〔2〕，…有形式



（最后的条款除外，容后再来说明之），其中 SUB 是有待装入的一个子程序，而 BASE 是代真的数量。这些条款是以子程序在带上出现的次序为次序的。在上边的例子中，一个可

能的答案将是

	BASE	SUB
X[1]:	2400	1002
X[2]:	2430	1010
X[3]:	2450	1006
X[4]:	2510	1000
X[5]:	2530	1003
X[6]:	2730	0

注意最后的条款包含了头一个未用的存储地址。

（显然，这并非是处理一个子程序库的唯一的方式。设计一个库的适当的方式密切地依赖于所使用的计算机和有待处理的应用。大型的现代计算机对于子程序库要求完全不同的方法。但这无论如何是一个很好的练习，因为它涉及到对于顺序的和链接的数据这两者的有趣的处理。）

这道习题中的问题，是来设计一个为完成这个子程序分配任务的算法。随着子程序分配程序准备其答案时，它可以以任何方式改造带目录，因为，带目录对于其下一次的分配，能由子程序分配程序来重新读入之，而且带目录不为装入程序的其它部分所需要。

27. [25] 写出一个为实现习题 26 的子程序分配算法的 MIX 程序。

28. [40] 以下的构造说明怎样来“解决”相当一般类型的二人游戏，包括国际象棋，以及许多更简单的游戏：考虑节点的一个有限集合，这些节点的每一个表示游戏中一个可能的“局面”。对于每个局面，有 0 个或多个“着法”。把该局面变换到某个其它的局面。如果有从 x 到 y 的一着，我们说局面 x 是 y 的一个前驱（而且 y 是 x 的一个后继）。无后继的某些局面，被分类成“胜利”或“失败”的局面。走到局面 x 的游戏者是走到 x 之后继的对手。

给出局面的这样一个配置：我们能计算出“胜局”的完备集合（这就是游戏者有可能赢得胜利的那些局面），以及“败局”的完备集合（这就是游戏者面对一个熟练的对手时，必然败北的那些局面）。其办法是重复地进行以下的操作，直到不再产生变化为止：如果一个局面的所有后继皆标以“胜利”，则对这个局面标以“失败”；如果它的后继至少有一个标以“失败”，则对这个局面标以“胜利”。

当这个操作已经尽可能多次地重复之后，可能有某些局面根本未被标记；在这样一个局面上，游戏者既不能获胜，也不可能被迫使败北。

这一用于得到“胜局”和“败局”的完备集合之步骤，可以为计算机接受作一个非常类似于算法 T 的有效的算法。对于每一个局面，我们可以保存对其未标以“胜利”的后继之个数的计数，以及它的所有前驱的一张表。

这道习题中的问题，是给出刚才已经如此笼统地描述的算法之细节，而且把它应用于某些不涉及太多可能局面的有趣的游戏[象“军事游戏”：*Sci. Am.* (October, 1963), 124或埃·路卡斯，《数学娱乐》(Récréations Mathématiques), 3（巴黎，1893）105～116]。

▶29. [21] （a）给出一个“抹去”象（1）那样的整个表的算法，即是，把它的所有节点都放到 AVAIL 堆栈，仅给出 FIRST 的值。这个算法应该尽可能快地进行操作。（b）

对于象（12）那样的表，重复部分（a），给定F和R的值。

2.2.4 循环表

链接方式的轻微变化，给我们提供了对前一小节之方法的重要的改变。

循环链接的表（简单地说：循环表）有这样的性质，它的最后的节点又回过头来链接到头一个节点上，而不是链接到Λ。因而，从任何位置开始，都能来访问这表的全部；我们也达到了额外程度的对称性，而且如果我们进行选择，则我们不必去想象这个表有一个“最后”或“最先”的节点。

以下的情况是典型的：



（1）

假定这些节点，象在上一小节一样，有两个场，INFO和LINK。有一个指向表的最右边节点的链接变量PTR，且LINK（PTR）是最左边节点的地址。以下的原始操作是最重要的：

a）在左边插入Y：P⇐AVAIL，INFO(P)←Y，LINK(P)←LINK(PTR)，LINK(PTR)←P。

b）在右边插入Y：在左边插入Y，然后PTR←P。

c）置Y为左边的节点并删去：P←LINK(PTR)，Y←INFO(P)，LINK(PTR)←LINK(P)，AVAIL⇐P。

操作（b）乍一看来有点奇怪；操作PTR←LINK(PTR)有效地把图式（1）中最左节点移动到右边去；而且如果把这个表看作是一个圆圈，而不是一条具有连接之端点的直线，则这就十分容易理解了。

机灵的读者将注意到，我们在上述操作（a），（b），（c）中已经造成了一个严重的错误。怎么回事？答曰：我们忘记了一个空表的可能性。如果对于表1，应用操作（c）六次，则我们将有指向AVAIL表中的一个节点的PTR，而这可能导致严重的困难；例如，想象对（1）应用操作（c）六次！如果我们主张在一个空表的情况下，PTR将等于Λ，那么，通过插入附加的指令“如果PTR＝Λ，则PTR←LINK(P)←P；否则…”于（a）和（b）的“INFO(P)←Y”之后；通过在（c）之前检验“如果PTR＝Λ，则UNDERFLOW”；而在紧跟着（c）之后，插入“如果PTR＝P，则PTR←Λ”，我们就可以补救上述的操作。

注意操作（a），（b）和（c）给了我们在2.2.1小节的意义下的输出受限的双排队的一个动作。因此，我们特别地发现，一个循环表可被用作一个堆栈或一个排队。操作（a）和操作（c）结合给了我们一个堆栈；操作（b）和（c）给了我们一个排队。这些操作比起上一小节中它们的对应部分来，只是不大直接而已，在上一小节我们看到，操作（a），（b）和（c）可通过使用两个指针F和R来实施于线性表。

对于循环表，另外一些重要的操作就成为有效的了。例如，“抹去”一个表，即是，一下子把整个循环表放到AVAIL堆栈，是非常方便的：

如果 PTR≠Λ，则 AVAIL↔LINK(PTR)　　(2)

(回忆一下，↔操作表示相互交换，即是P←AVAIL; AVAIL←LINK(PTR); LINK(PTR)←P)。如果 PTR 指向循环表的任何地方，则操作(2)显然是正确的。

使用类似的技术，如果 PTR_1 和 PTR_2 分别指向不相交的循环表 L_1 和 L_2，则我们可以把整个表 L_2 插到 L_1 的右边：

如果 $PTR_1 \neq \Lambda$，则

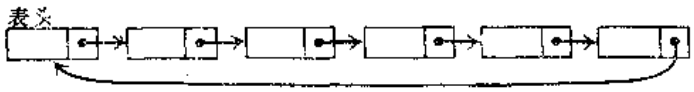
(如果 $PTR_2 \neq \Lambda$，则 $LINK(PTR_1) \leftrightarrow LINK(PTR_2)$;

置 $PTR_2 \leftarrow PTR_1$, $PTR_1 \leftarrow \Lambda$)　　(3)

以各种方式，来把一个循环表分成为两个，乃是可以进行的另一个简单的操作。这些操作对应于串的连接和分开。

于是，我们看到，一个循环表不仅可用来表示固有地循环的结构，而且可用来表示线性的结构：一个具有指向后尾节点之指针的循环表，实质上等价于一个具有指向前头和后尾的两个指针的线性表。关于这些论述，自然要问的问题是："在循环之对称性的观点下，我们如何来发现这个表的结尾？"因为没有Λ链接来标志结尾了。回答是，如果我们正在实施某个操作，而同时从一个节点移到另一个节点地通过该表，则当我们返回到我们开始的位置时(当然假定，我们的开始位置仍将出现于表中)，我们就应该停止下来。

对于刚才提出的问题，另一个解决办法，是设置一个特殊的，可认记的节点于每个循环表中，作为一个方便的停止位置。这个特殊的节点称作表头，而且在一些应用中，我们发现，坚持主张每一个循环表恰好有一个作为它的表头的节点，是十分方便的。一个好处是，这样循环表就将永不是空的。现在，图式(1)变成为



(4)

对于象(4)这样的表的访问，通常不是通过一个指向表右端的指针，而是通过表头进行的，表头通常都在一个固定的存储单元中。在这种情况下，我们抛弃了上边所述的操作(b)。

图式(4)可以与在上一小节开始处的2.2.3节(1)作一比较。在那里，与"条款5"相关联的链接现在指向 LOC(FIRST) 而不是Λ，而 FIRST 现在想象作在节点 NODE(LOC(FIRST)) 之内的一个链接。在(4)与2.2.3节(1)之间，其主要差别在于，通过(4)有可能(尽管未必高效)从任何点来获得表的任何其它点。

作为循环表用法的一个例子，我们来讨论具有整系数的对于变量 x，y 和 z 的多项式之算术运算。有许多问题，其中科学家都要来处理多项式，而不仅仅是处理数。我们正在考虑象

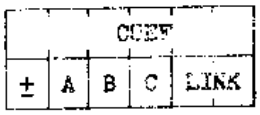
$$(x^4+2x^3y+3x^2y^2+4xy^3+5y^4) \text{ 乘以 } (x^2-2xy+y^2)$$

得到

$$(x^6-6xy^5+5y^6)$$

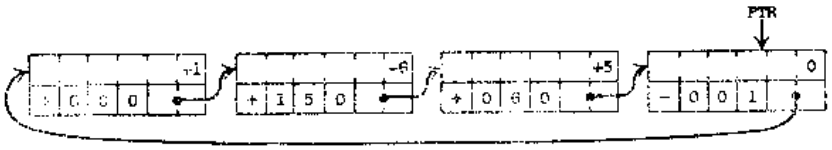
这样的一些运算。为此目的，链接分配是一个自然的工具，因为多项式可能扩展到不可予测的长度，而且我们可能要同时把许多多项式表示在内存中。

这里我们将考虑加法和乘法这两个操作。让我们假设，把一个多项式表示成一个表，在这个表中每个节点代表一个非0项，并有双字的形式：



(5)

这里COEF是$x^Ay^Bz^C$项中的系数。我们将假定系数和指数总是保持在这个格式所允许的范围之内，而且在我们进行计算期间不需要校验这个条件。记号ABC将作为一个整体，用来代表节点(5)的+ABC场。ABC的符号，即是(5)中的第二个字的符号，将总为正，除非是在每个多项式的结尾处，才有一个特殊的节点，对于它有ABC=−1和COEF=0。与我们上边关于一个表头的讨论相类似，这个特殊的节点有很大的方便性，因为它提供了方便的"哨兵"，而且它避免了一个空表（对应于多项式"0"）的问题。如果我们沿着链接的方向的话，表的节点总是以ABC场递减的次序出现。但特殊的节点（它有ABC=−1）则链接到ABC的最大值。例如，多项式$x^6-6xy^6+5y^6$将表示成：



算法A（多项式的加法） 这个算法把多项式（P）加到多项式（Q）上，假定P和Q是指到有上述形式的多项式的指针变量。表P将不被改变，表Q将保存和式。在这个算法结束处，指针变量P和Q都恢复到它们开始的位置；还使用了辅助的指针变量Q1和Q2。

A1. [初始化] 置P←LINK(P)，Q1←Q，Q←LINK(Q)(现在P和Q都指向多项式的首项。贯穿这个算法的大部分，在Q=LINK(Q1)的意义下，变量Q1将是Q"之后一步"。)

A2. [ABC(P):ABC(Q)] 如果ABC(P)<ABC(Q)，则置Q1←Q和Q←LINK(Q)并重复这一步骤。如果ABC(P)=ABC(Q)，则转到步骤A3。如果ABC(P)>ABC(Q)，则转到步骤A5。

A3. [系数相加] (我们发现了有相等指数的项)。如果ABC(P)<0，则算法终止；否则置COEF(Q)←COEF(Q)+COEF(P)。现在，如果COEF(Q)=0，则转到A4；否则，置Q1←Q，P←LINK(P)，Q←LINK(Q)，并转到A2(奇怪啊，这后边的操作竟与步骤A1一样)。

A4. [删去0项] 置Q2←Q，LINK(Q1)←Q←LINK(Q)，和AVAIL←Q2(在步骤A3中建立起来的0项已从多项式(Q)中撤消)。置P←LINK(P)并转到步骤A2。

A5. [插入新项] (多项式(P)含有在多项式(Q)中不出现的一项，所以我们把它插入多项式(Q)中)。置Q2⇐AVAIL，COEF(Q2)←COEF(P)，ABC(Q2)←ABC(P)，LINK(Q2)←Q，LINK(Q1)←Q2，Q1←Q2，P←LINK(P)，并返回步骤A2。∎

算法A最显著的特性之一，乃是指针变量Q1围绕着表紧跟指针Q之方式。这是非常典型的表处理算法，而且我们将看到许多具有这同一特征的算法。读者能看出为什么在算法A中使用这一思想吗？

只要对于链接表有点预先的经验的读者，就会发现，仔细地研究算法A是颇有教益的；作为检验的实例，试把 $x+y+z$ 加到 x^2-2y-z。

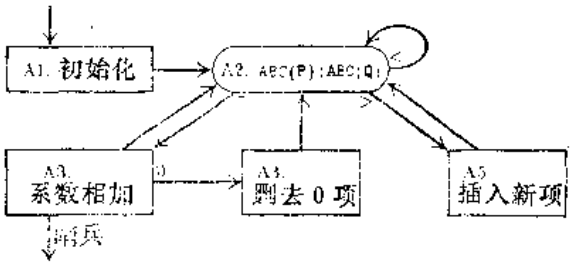


图10 多项式的加法

给出了算法A后，乘法的操作是意外地容易的：

算法M（多项式的乘法） 类似于算法A，这个算法以多项式（Q）+多项式(M)×多项式（P）来代替多项式（Q）。

M1.〔下一乘了〕 置M←LINK(M)。如果ABC(M)<0，则算法终止。

M2.〔乘法循环〕 实施算法A，只是每当记号"ABC(P)"出现于该算法中时，以"如果ABC(P)<0则−1，否则，ABC(P)+ABC(M)"来代替它；每当"COEF(P)"出现于该算法中时，以"COEF(P)×COEF(M)"代替它。然后转回到步骤M1。■

用MIX语言来编算法A的程序，又再一次说明了在一台计算机中处理链接表的容易性。在下列的代码中，我们假定OVERFLOW是一个这样的子程序，它或者使程序终止（由于缺少存储空间），或者找到可利用的空间并以（rJ)−2作为出口。

程序A（多项式的加法） 这是一个子程序，已被写成使它能与一个乘法子程序一起使用（见习题15）。

调用序列：JMP ADD

进入条件：r11=P，r12=Q。

出口条件：多项式（Q）已经为多项式（Q）+多项式（P）所代替；r11和r12皆不变；所有其它的寄存器都有不确定的内容。

在下列的编码中，在算法A的记号下，P=r11，Q=r12，Q1=r13，以及Q2=r16。

```
01    LINK    EQU   4:5                        定义LINK场
02    ABC     EQU   0:3                        定义ABC场
03    ADD     STJ   3F              1          子程序入口
04    A1      ENT3  0,2             1+m'       A1.初始化，置Q1←Q
05            LD2   1,3(LINK)       1+m        Q←LINK(Q1)
06    A2      LD1   1,1(LINK)       1+p        P←LINK(P)
07    SW1     LDA   1,1             1+p        rA(0:3)←ABC(P)
08    2H      CMPA  1,2(ABC)        x          A2.ABC(P):ABC(Q)
09            JE    3F              x          若相等，则转到A3
10            JG    5F              p'+q'      若大于，则转到A5
11            ENT3  0,2             q'         若小于，则置Q1←Q
12            LD2   1:3(LINK)       q'         Q←LINK(Q')
13            JMP   2B              q'         重复
```

14	3H	JAN	*	$m+1$	A3. 系数相加
15	SW2	LDA	0,1	m	COEF(P)
16		ADD	0,2	m	+COEF(Q)
17		STA	0,2	m	→COEF(Q)
18		JANZ	1B	m	结果为 0？
19		ENT6	0,2	m'	A4. 删去 0 项. Q2←Q
20		LD2	1,2(LINK)	m'	Q←LINK(Q)
21		LDX	AVAIL	m'	\
22		STX	1,6(LINK)	m'	\ AVAIL⇐Q2
23		ST6	AVAIL	m'	/
24		ST2	1,3(LINK)	m'	LINK(Q1)←Q
25		JMP	0B	m'	转到推进 P
26	5H	LD6	AVAIL	p'	A5. 插入新项
27		J6Z	OVERFLOW	p'	Q2←AVAIL
28		LDX	1,6(LINK)	p'	
29		STX	AVAIL	p'	
30		STA	1,6	p'	ABC(Q2)←ABC(P)
31	SW3	LDA	0,1	p'	rA←COEF(P)
32		STA	0,6	p'	COEF(Q2)←rA
33		ST2	1,6(LINK)	p'	LINK(Q2)←Q
34		ST6	1,3(LINK)	p'	LINK(Q1)←Q2
35		ENT3	0,6	p'	Q1←Q2
36		JMP	0B	p'	转到推进 P。■

注意算法 A 仅一次就遍历两个表的每一个；它不需要循环若干次。利用克希霍夫定律，我们发现，对于执行的分析不显得困难；执行时间依赖于数量

m' = 彼此相消的匹配的项数；

m'' = 不能消去的匹配的项数；

p' = 多项式（P）中不匹配的项数；

q' = 多项式（Q）中不匹配的项数。

对于程序 A 给出的分析，使用了缩写

$$m = m' + m'', \qquad p = m + p',$$

$$q = m + q', \qquad x = 1 + m + p' + q';$$

对于 MIX 的运行时间是 $(27m' + 18m'' + 27p' + 8q' + 13)u$。在执行此算法时，所需要的存储池中的节点总数至少是 $2 + p + q$，至多为 $2 + p + q + p'$。

习题

1 〔21〕 这一小节开始时，正文中就提出，一个空的循环表可以通过 PTR = A 来表示。以 PTR = LOC(PTR) 来表示一个空的表，可能与循环表的原理更为一致。这个约定是否方便了这一小节开始时所描述的操作（a），（b）或（c）？

2. 〔20〕 假定 PTR_1 和 PTR_2 都 = A，试画出说明连接操作（3）的效果的“之前和之后”图式。

▶3. [20] 如果 PTR1 和 PTR2 两者都指向同一个循环表中的节点，则操作（3）将做什么？

4. [21] 给出对应于表示（4）的插入和删去操作——表示（4）是说明一个堆栈的作用的。

▶5. [21] 试设计一个算法，它采用如同（1）那样的一个循环表，并把所有箭头的方向全都颠倒过来。

6. [18] 给出对于多项式（a）"xz-3"；（b）"0"的表表示的图式。

7. [10] 为什么假定一个多项式表的 ABC 场以递减的次序出现是有用的？

8. [10] 为什么在算法 A 中要 Q1 跟在 Q 之后一步是有用的？

▶9. [28] 如果 P = Q（即是，两个指针变量都指到同一个多项式），则算法 A 能否正常工作？如果 P = M，如果 P = Q，或者如果 M = Q，则算法 M 能否正常工作？

▶10. [20] 这一小节的算法假定，我们在多项式中正在使用着三个变量 x，y 和 z，而且它们的指数一个个都决不超过 $b-1$（其中 b 是在 MIX 情况下的字节大小）。假设，我们代之以要进行仅有一个变量 x 的多项式之加法和乘法，并且假设它的指数取直到 b^3-1 的值。那么，对于算法 A 和 M 应该作什么改变？

11. [21] （本题以及此后许多题的目的，是要建立同程序 A 一起，对于多项式的算术运算有用的"子程序包"。）由于算法 A 和 M 改变了多项式（Q）的值，而有时又希望要有做出一个给定的多项式的付本的子程序。请按如下规定，写出一个 MIX 子程序：

调用序列：JMP COPY

进入条件：rI1 = P

出口条件：rI2 指向新近建立起来等于多项式（P）的一个多项式；rI1 不变；其它寄存器不确定。

12. [21] 试将习题 11 中程序的运行时间与多项式(Q)="0"时算法 A 的程序的运行时间作一比较。

13. [20] 写出一个具有如下说明的 MIX 子程序：

调用序列：JMP ERASE

进入条件：rI1 = P

出口条件：多项式（P）已经被加到 AVAIL 表；所有寄存器的内容都不确定。

[注意：这个子程序可以以序列"LD1 Q；JMP ERASE；LD1 P；JMP COPY；ST2 Q"来与习题 11 的子程序一起使用，以达到"多项式（Q）←多项式（P）"之效果。]

14. [22] 写出一个具有下列说明的 MIX 子程序：

调用序列：JMP ZERO

进入条件：无

出口条件：rI2 指向最新建立起来的等于"0"的一个多项式；其它寄存器的内容都不确定。

15. [24] 写出一个有着如下说明，实施算法 M 的 MIX 子程序：

调用序列：JMP MULT

进入条件：rI1 = P，rI2 = Q，rI4 = M。

出口条件：多项式（Q）←多项式（Q）+多项式（M）×多项式（P）；r11，r12，r14不变；其它寄存器都不确定。

（注意：利用程序A作为一个子程序，同时改变SW1，SW2和SW3的设置。）

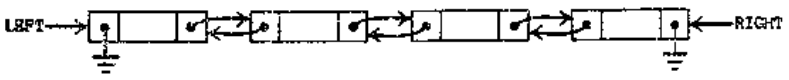
16.〔M22〕 借助于某些有关的参数，来估计一下习题15中的子程序的运行时间。

▶17.〔22〕 如象在这一小节这样，以一个循环表来表示多项式，而不是如象在上一小节那样，以一个终止于Λ的直线形线性表来表示，有什么优点？

18.〔27〕 试设计一个在计算机内部表示循环表的方法，使得这个表能以两个方向有效地被遍历，而且对于每个节点仅仅使用一个链接场（提示：如果我们给出了两个指针，以指向两个相继的节点 x_{k-1} 和 x_k，则应当有可能来确定 x_{k+1} 和 x_{k-2} 的位置。）

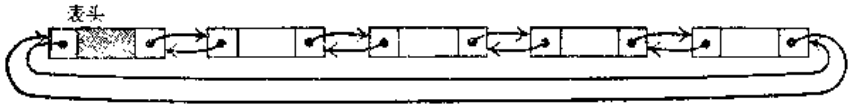
2.2.5 双重链接表

为了使对线性表的操作有更大灵活性，我们可以在每个节点中包括两个链接，以指向该节点之两边的条款：



（1）

这里LEFT和RIGHT是指向这表的左边和右边的指针变量。表的每个节点，包括两个链接，比如说把它们叫做LLINK和RLINK。通过上述的表示，一般的双排队的操作就可以很容易地实施了；见习题1。然而，如果一个表头节点是每个表的一部分，象在上一小节中所述的那样，则双重链接表的操作，几乎总是变成容易得多。当出现一个表头时，我们有下列典型的双重链接表图式：



（2）

表头的RLINK和LLINK场取代了（1）中的LEFT和RIGHT的位置。在左边与右边之间有着完全的对称性；表头也完全同样可以图示在（2）的右边。如果这个表是空的，则表头的两个链接场就指向表头本身。

表表示（2）显然满足条件

$$\mathrm{RLINK}(\mathrm{LLINK}(X)) = \mathrm{LLINK}(\mathrm{RLINK}(X)) = X \tag{3}$$

如果X是表中任何节点（包括表头）的单元。这一事实，是表示（2）比（1）要好的主要原因。

双重链接表通常比单重链接的表要多花费存储空间（尽管有时在未填满整个一个计算机字的节点中，已经有了供另一个链接存储用的地方）。但现在所能高效地实施的另外一些操作，比起这额外的空间来，通常是更丰厚的报偿。除了在检查一个双重链接表时有能力随意地向后或向前这一明显的优点外，主要的新能力之一，是仅仅给定X的值，我们就能从NODE(X)所在的表中删去该节点。这个删去操作容易从一个“之前和之后”的图

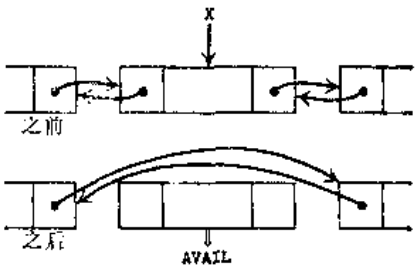


图11 从一个双重链接表删去

式（图 11）导出，而且它是非常简单的：

$$\text{RLINK}(\text{LLINK}(X))\leftarrow\text{RLINK}(X),\qquad \text{LLINK}(\text{RLINK}(X))\leftarrow\text{LLINK}(X),$$
$$\text{AVAIL}\leftarrow X \tag{4}$$

在仅有单向链接的表中，如果我们不知道在这链接的链条中，哪一个节点是在节点X之前，则我们就不能删去NODE(X)，因为，当NODE(X)被删去时，这前边的节点的链接也需要相应地改变。在2.2.3和2.2.4小节所考虑的所有算法中，每当要删去一个节点时，就提出了这个附加的知识；特别地，见算法2.2.4A，其中恰恰是为了这一目的，我们才有紧跟指针Q的指针Q1。但是我们将遇到许多需要从一个表中间取走随机节点的算法，而且双重链接表就恰恰由于这一原因而经常地被采用（我们应该指出，在一个循环表中，给定了X，如果我们跑遍整个循环来寻找X的前驱，则也可能删去NODE(X)。但当表很长时，这种操作显然是低效率的，因此，不用双重链接表，而采用循环表，是难以接受的。也请看习题2.2.4-8）。

类似地，双重链接表允许容易地插入一个或在左边或在右边与NODE(X)相邻的节点。步骤

$$\begin{aligned}&P\leftarrow\text{AVAIL}\qquad \text{LLINK}(P)\leftarrow X\qquad \text{RLINK}(P)\leftarrow\text{RLINK}(X)\\&\text{LLINK}(\text{RLINK}(X))\leftarrow P\qquad \text{RLINK}(X)\leftarrow P\end{aligned} \tag{5}$$

在NODE(X)右边进行这样的插入；而且通过交换左边与右边，我们可以得到为插入到左边的相应的算法。操作（5）改变了五个链接的设置，所以它比在单向的表中的插入操作是要稍微慢些的，在那种情况下，仅仅需要改变三个链接。

作为使用双重链接表的一个例子，我们现在来考虑编写一个离散的模拟程序。“离散的模拟”指的是模拟一个这样的系统，其中系统状态的所有变化可以假定是在某些离散的时间瞬间中发生的。正被模拟的“系统”通常是一个个别的活动体的集合，这些活动体是基本上独立的，虽然它们彼此交互作用；其例子有：商店里的顾客，港口中的船只，一个团体中的人们。在一个离散的模拟中，我们在某个模拟瞬间进行着有待完成的无论什么事情，然后把模拟时钟推进到某个动作预定要出现的下一个时刻。

与此相反，“连续的模拟”将模拟在连续变化之下的动作。例如在公路上的交通运输，宇宙飞船飞向另一个行星，等等。连续的模拟，通常可以通过对于各步之间采取非常小的时间区间的离散的模拟，来进行满意的逼近；然而，在这样一种情况下，我们通常都需要“同步的”离散模拟，在每一个离散的时间区间里，这系统的许多部分已被稍加改变，而且，这样一种应用一般地都要求与这里所考虑的类型略微有些不同的程序组织类型。

下边所编制的程序是模拟加里佛尼亚理工学院的数学楼里的电梯系统。这样一个模拟的结果，也许仅对于相当经常地访问加省理工学院的人们是有用的；甚至对于那些可能只不过是来试验使用电梯若干次，而不是来写一个计算机程序的人也是有用的。但是，由于模拟研究很通常，我们将用来实现模拟的方法，比起由程序所给出的答案来，要远为有趣得多。下面所讨论的方法，说明了对于离散模拟程序所使用的典型的实现技术。

数学楼有五层：下层地下室，地下室，第一层，第二层和第三层。有一个电梯，这电梯有自动控制，并能在每一层处停下来。为方便起见，我们将把各层重新编号成0，1，2，3，和4。

在每一层上有两个呼叫按钮，一个是要求UP(上)的，另一个是要求DOWN(下)的(实际上0层仅有UP，而4层仅有DOWN，但我们可以忽略这种异常，因为多余的按钮将决不被使用)。对应于这些按钮，有十个变量CALLUP(i)和CALLDOWN(i)，$0 \leqslant i \leqslant 4$。还有变量CALLCAR($i$)，$0 \leqslant i \leqslant 4$，表示在电梯车内的一些按钮，它指挥电梯驶往目标层。当一个人按下一个电钮时，他就把相应的变量置成1；而当要求一经满足后，电梯就把这变量清为0。

以上从人的观点描述了电梯；如果从电梯来观察，则情况就更为有趣。电梯处于三种状态之一：GOINGUP（上行），GOINGDOWN（下行），或NEUTRAL（停立）(通过电梯内带灯光的箭头来向乘客指出当前的状态)。如果电梯处于NEUTRAL状态，而且又不在2层，则机器将关闭它的门并且（如果在它的门被关闭之前没有命令给出），它将改变成GOINGUP或GOINGDOWN，而向2层前进（这是"本垒"层，因为大多数乘客都去这里)。在2层时的NEUTRAL状态，门将终于关闭，而且机器将静候另一个命令。第一个收到的往另一层的命令，把机器相应地置成GOINGUP或GOINGDOWN；它一直停留在这个状态，直到在这同一方向没有等候的命令为止，而后它视在CALL变量中的是什么命令，或者改变方向，或者就在打开门之前，改变成NEUTRAL状态。电梯花费一定数量的时间，来打开和关闭它的门，来加速或减速，以及从一层达到另一层。下列算法中指出了所有这些数量，该算法比这个非正式的说明要精确得多。我们现在要研究的这个算法，可能不反映电梯操作的真实原理，但可以确信，它所提供的，乃是作者在编写这一小节时，由经过几个小时实践观察到的所有现象抽象出来的最简单的一组规则。

电梯系统通过使用两个共行程序来模拟。这两个共行程序，一个是对于乘客的，一个是对于电梯的；这些子程序确定了有待实施的所有动作，以及在模拟时有待使用的各种时间延迟。在下列描述中，变量TIME表示模拟的时钟的当前值。所有的时间单位均以十分之一秒给出。还有若干其它的变量：

FLOOR，电梯的当前位置；

D1，此变量之值为0，除非当人们正在进入或离开电梯时；

D2，此变量之值为0，如果电梯已经停在一层楼上而不移动达30秒或更长的时间；

D3，此变量之值为0，除非当电梯门打开着，但没有任何人进入或离开电梯时；

STATE，电梯的当前状态（GOINGUP，GOINGDOWN，或NEUTRAL)。起始，FLOOR＝2，D1＝D2＝D3＝0，而且STATE＝NEUTRAL。

共行程序M（人） 当每个人进入此系统时，他从步骤M1起始，开始来实施以下所

描述的动作。

M1.〔进入，为后来人作准备〕 以某种方式来确定下列的数量，至于什么方式，在这里暂不说明：

IN，新的人进入本系统的那个层号；

OUT，他所要去的那一层号（OUT≠IN）；

INTERTIME，在下一个人将进入此系统之前的时间数量；

GIVEUPTIME，在这个人放弃并决定步行之前，这个人等候电梯的时间数量。

在计算了这些数量之后，模拟程序设置一些事项，使得下一个人在TIME+INTERTIME时进入系统。

M2.〔发信号并等候〕（这一步骤的目的是来呼叫电梯；如果电梯已经在正确的层次上，则就出现某些特殊的情况）。如果FLOOR＝IN而且如果电梯的下一个动作是下面的步骤E6（即是，如果电梯的门现在正在关闭），则立即让电梯转移到它的步骤E3中止撤消它的动作E6（这意味着在电梯移动之前，门将再次打开）。如果FLOOR＝IN而且如果D3≠0，则置D3←0，置D1≠0，且再次启动电梯动作E4（这意味电梯就在这一层上，门是打开的，但有别的人在进出；电梯的步骤E4是这样一个定序步骤，即它容许人们按照通常的礼仪进到电梯中，所以重新启动步骤E4就给了这一个人以在门关闭之前进到电梯的机会）。在所有其它情况下，这人根据OUT>IN或OUT<IN，置CALLUP〔IN〕←1或CALLDOWN〔IN〕←1；而且如果D2＝0，或者如果电梯处于它的"静止"位置E1，则实施以下所描述的DECISION（判断）子程序（DECISION子程序用来使电梯在某些临界的时间处脱离NEUTRAL状态）。

M3.〔进入排队〕 在QUEUE〔IN〕的后尾插入这个人，这个排队是表示在这层楼上等候的人的线性表。现在这个人停止活动；在GIVEUPTIME个时间单位之后，他将实施动作M4，除非以下的电梯程序的步骤E4更早地把他送到M5。

M4.〔放弃〕 如果FLOOR≠IN或D1＝0，则从QUEUE〔IN〕和从模拟系统中除去这个人（他已判定电梯太慢，或者说他乐于活动一下）。如果FLOOR＝IN且D1≠0，则他就停留并等候（因为他知道不久他就可能进入）。

M5.〔进入〕 从QUEUE〔IN〕除去这个人，并把他插入到ELEVATOR(电梯)，这是一个类似堆栈的表，表示现在电梯中的人们。置CALLCAR〔OUT〕←1。

现在如果STATE＝NEUTRAL，则相应地置STATE←GOINGUP或GOINGDOWN，并让电梯活动E5在25个时间单位后被执行（这是该电梯的一个特殊之处，即是，当一个人进入电梯而且电梯处于NEUTRAL状态时，门关上扣闩。25个时间单位使步骤E4有机会来确保，在步骤E5即关门动作出现的时间之前，D1能被适当地置位）。

现在这个人就等候着，直到通过下面的步骤E4，当电梯已经达到他要到达的层次时，他就被送到步骤M6为止。

M6.〔离去〕 从ELEVATOR和从这模拟系统删去这个人。

共行程序E（电梯） 这个共行程序，表示电梯的动作，以及在步骤E4处当人们进入和离去时的控制。

E1.〔等候呼叫〕（这时，电梯正停在第2层，同时关着门以等候发生某件事情）。如果

某人按下一个按钮，则DECISION子程序将使我们进到步骤E3或E6。其间，进行等候。

E2. 〔要改变状态？〕 如果STATE=GOINGUP并且对于所有的j>FLOOR，CALLUP[j]=CALLDOWN[j]=CALLCAR[j]=0，则视对于所有的j<FLOOR都有CALLCAR[j]=0或否，而置STATE←NEUTRAL或STATE←GOINGDOWN，并把当前的层次的所有CALL变量都置为0。如果STATE=GOINGDOWN，则就以相反的方向来做类似的动作。

E3. 〔开门〕 置D1和D2为任何非0的值；让电梯活动E9在300个时间单位之后独立地启动（这个活动在其出现之前，可能在以下的步骤E6中被消除）。并让电梯活动E5在76个时间单位之后独立地启动。然后等候20个时间单位（以模拟开门）并转到E4。

E4. 〔让人出入〕 如果在ELEVATOR表中的任何人，有OUT=FLOOR，则让最近进入的这种类型的人立即转移到他的程序的步骤M6中，等候25个时间单位，并重复步骤E4。如果没有这样的人存在，但是QUEUE[FLOOR]不空，则让这个排队中前头的人立即转移到他的程序中的步骤M5，而不是步骤M4，等候25个时间单位，并重复步骤E4。但如果QUEUE[FLOOR]是空的，则置D1←0，D3≠0，而且等候某个其它的活动以开始进一步的动作（步骤E5将我们送到E6，或者步骤M2将重新开始E4）。

E5. 〔关门〕 如果D1≠0，则等候40个时间单位并重复这一步骤（门稍有振动，但再次弹开来，因为某人仍在出入），否则置D3←0并让电梯在20个时间单位之后在步骤E6启动（这模拟当人们已经结束出入之后的关门；但如果一位新的人进到这一层而这时正在关门，则如同在步骤M2所述，这门将再次打开）。

E6. 〔准备移动〕 置CALLCAR[FLOOR]为0；而且如果STATE≠GOINGDOWN，则置CALLUP[FLOOR]为0，又如果STATE≠GOINGUP，则亦置CALLDOWN[FLOOR]为0。（注意：如果STATE=GOINGUP，则电梯不清除CALLDOWN，因为它假定正在要往下去的人将不进入；但请看习题6）。现在实施DECISION子程序。

如果STATE=NEUTRAL，则即使在已经执行了DECISION子程序之后，亦转到E1。否则，如果D2≠0，则取消电梯活动E9。最后，如果STATE=GOINGUP，则等候15个时间单位（为电梯加速用）并转到E7；如果STATE=GOINGDOWN，则等候15个时间单位并转到E8。

E7. 〔上升一层〕 置FLOOR←FLOOR+1并等候51个时间单位。如果现在CALLCAR[FLOOR]=1或CALLUP[FLOOR]=1，或者如果((FLOOR=2或CALLDOWN[FLOOR]=1)且CALLUP[j]=CALLDOWN[j]=CALLCAR[j]=0对于所有的j>FLOOR)，则等候14个时间单位（为了减速）并转到E2。否则，重复这一步骤。

E8. 〔下降一层〕 这一步骤除了方向相反之外，与E7类似，还有那儿的51和14个时间单位这儿分别改成61和23（电梯下降所花的时间比上升所花的还要多些）。

E9. 〔置不活动指示器〕 置D2←0并实施子程序DECISION（这个独立的动作是在步骤E3开始的，但它几乎总是在步骤E6中被消去。见习题4）。■

子程序D（判断子程序DECISION） 当对电梯的下一个方向须作出判定时，如同在上面的共行程序中所说明的那样，便在若干临界的时刻实施这个子程序。

D1.〔需要判断?〕 如果 STATE≠NEUTRAL，则离开这个子程序。

D2.〔应该开门?〕 如果电梯处于E1的位置而且如果CALLUP[2], CALLCAR[2], 或CALLDOWN[2]非0，则在20个时间单位之后使电梯启动它的活动E8，并离开这个子程序（如果 DECISION 子程序当前正在为独立的活动E9所调用，则电梯其行程序也有可能处于位置E1)。

D3.〔有呼叫?〕 找最小的 j≠FLOOR，使得CALLUP[j], CALLCAR[j], 或CALLDOWN[j]为非0，并转到步骤D4。但如果不存在这样的 j，那么，如果DECISION 子程序当前正为步骤E6所调用，则使置 $j \leftarrow 2$；否则离开这个子程序。

D4.〔置STATE〕 如果FLOOR>j,则置STATE←GOINGDOWN；如果FLOOR<j,则置 STATE←GOINGUP。

D5.〔电梯静止?〕 如果电梯处在步骤E1的位置，而且如果 j≠2，则在20个时间单位之后让电梯来实施步骤E6。离开这个子程序。∎

与我们在本书中所已见到的其它算法比较，上述电梯系统是十分复杂的。但是，选择一个现实生活的系统，比起任何杜撰的"书本上的例子"来，毕竟是更为典型的一个模拟问题。

为了帮助理解这个系统，现在来考虑表1，它给出了一个模拟的部分历史。也许最好是通过检查由时间 4257 开始的简单情况下手：这时电梯正停在2层闲着无事，而且它的门关着，当时有一个人到达（时间为 4384)。两秒钟之后，门打开，而且再过数秒后，他进入；通过按下按钮"3"他启动电梯上升；最终他到达3层下去了并且电梯折回2层。表1中的开头一些条目示出了更多得多的戏剧性的情节：一个人要求电梯到0层，但他颇为急性子，在15.2秒之后放弃。电梯停到0层但发现没有任何人在那儿；然后它向4层前进，因为有好几个呼叫等候要下楼来；等等。

对于一台计算机（在我们的情形，是MIX)，来进行这个系统的程序设计，是值得小心地研究的。在模拟期间的任何给定的时刻，我们可以有许多模拟的人在这系统中（在各个排队中并准备在各个时间里放弃)，而且当电梯正试图关门时，如果有许多人正试图出

表1 电梯系统的某些动作

时间	状态	层次	D1	D2	D3	步骤	动作
0000	*N*	2	0	0	0	*M*1	第1人到达0层，目标是2层
0035	*D*	2	0	0	0	*E*8	电梯下降
0038	*D*	1	0	0	0	*M*1	第2人到达4层，目标是1层
0096	*D*	1	0	0	0	*E*8	电梯下降
0136	*D*	0	0	0	0	*M*1	第3人到达2层，目标是1层
0141	*D*	0	0	0	0	*M*1	第4人到达2层，目标是1层
0152	*D*	0	0	0	0	*M*4	第1人决定放弃并且他离去
0180	*D*	0	0	0	0	*E*2	电梯停车
0180	*N*	0	*X*	*X*	0	*E*3	电梯门开始打开
0256	*N*	0	0	*X*	*X*	*E*5	电梯门开始关闭
0291	*U*	0	0	*X*	0	*M*1	第5人到达3层，目标是1层
0291	*U*	0	0	*X*	0	*E*7	电梯上升
0342	*U*	1	0	*X*	0	*E*7	电梯上升
0361	*U*	2	0	*X*	0	*M*1	第6人到达2层，目标是1层
0393	*U*	2	0	*X*	0	*E*7	电梯上升

（续）

时间	状态	层次	D1	D2	D3	步骤	动作
0434	U	3	0	X	0	E7	电梯上升
0509	U	4	0	X	0	E2	电梯停车
0509	N	4	X	X	0	E3	电梯门开始打开
0529	N	4	X	X	X	M5	第2人进入
0540	D	4	X	X	0	M4	第6人决定放弃并且他离去
0554	D	4	0	X	X	E5	电梯门开始关闭
0589	D	4	0	X	0	E8	电梯下降
0602	D	3	0	X	0	M1	第7人到达1层，目标是2层
0673	D	3	0	X	0	E2	电梯停车
0673	D	3	X	X	0	E3	电梯门开始打开
0693	D	3	X	X	0	M1	第5人进入
0749	D	3	0	X	X	E5	电梯门开始关闭
0784	D	3	0	X	0	E8	电梯下降
0827	D	2	0	X	0	M1	第8人到达1层，目标是0层
0868	D	2	0	X	0	E2	电梯停车
0868	D	2	X	X	0	E3	电梯门开始打开
0876	D	2	X	X	0	M1	第9人到达1层，目标是3层
0888	D	2	X	X	0	M5	第3人进入
0913	D	2	X	X	0	M5	第1人进入
0944	D	2	0	X	X	E5	电梯门开始关闭
0979	D	2	0	X	0	E8	电梯下降
1048	D	1	0	X	0	M1	第10人到达0层，目标是1层
1063	D	1	0	X	0	E2	电梯停车
1065	D	1	X	X	0	E3	电梯门开始打开
1085	D	1	X	X	0	M6	第1人出去，离开系统
1108	D	1	X	X	0	M6	第3人出去，离开系统
1135	D	1	X	X	0	M6	第5人出去，离开系统
1155	D	1	X	X	0	E5	门振动
1158	D	1	X	X	0	M6	第2人出去，离开系统
1179	D	1	X	X	0	E5	门振动
1183	D	1	X	X	0	M5	第7人进入
1208	D	1	X	X	0	M5	第8人进入
1219	D	1	X	X	0	E5	门振动
1235	D	1	X	X	0	M5	第9人进入
1249	D	1	0	X	X	E5	电梯门开始关闭
1284	D	1	0	X	0	E8	电梯下降
1378	D	0	0	X	0	E2	电梯停车
1378	U	0	X	X	0	E3	电梯门开始打开
1398	U	0	X	X	0	M6	第8人出去，离开系统
1423	U	0	X	X	0	M5	第10人进入
1454	U	0	0	X	X	E5	电梯门开始关闭
1529	U	0	0	X	0	E7	电梯上升
1554	U	1	0	X	0	E2	电梯停车
1554	U	1	X	X	0	E3	电梯门开始打开
1630	U	1	0	X	X	E5	电梯门开始关闭
1655	U	1	0	X	0	E7	电梯上升
…							
4257	N	2	0	X	0	E1	电梯静止
4384	N	2	0	X	0	M1	第17人到达2层，目标是3层
4404	N	2	X	X	0	E3	电梯门开始打开

（续）

时间	状态	层次	D1	D2	D3	步骤	动作
1424	V	2	X	X	0	M5	等候人进入
1429	U	2	0	X	X	E5	电梯门开始关闭
1484	U	2	0	X	0	E7	电梯下降
1549	U	3	0	X	0	E2	电梯静止
1549	V	3	X	X	0	E3	电梯门开始打开
1569	V	3	X	X	0	M3	等候人出去，离开系统
1625	V	3	0	X	X	E5	电梯门开始关闭
1690	U	3	0	X	0	E8	电梯下降
1714	U	2	0	X	0	E2	电梯静止
1714	V	2	X	X	0	E3	电梯门开始打开
1830	V	2	0	X	0	E5	电梯门开始关闭
1830	V	2	0	X	0	E1	电梯静止
...							

去，则就有可能实际上同时执行步骤E4，E5和E9。所模拟时间的经过以及“同时性”的处理，可以通过每个实体来进行程序设计。每个实体用一个节点来表示，这个节点包括一个NEXTTIME场（表示对于这一实体下一动作何时发生的时间）以及NEXTINST场（表示这一实体启动执行指令的存储器地址，类似于通常的共行程序链接）。每一个等待时间进行的实体，被放在一个称作WAIT（等待）表的双重链接表中；这个“设事日程”按其节点的NEXTTIME场排序，使得这些动作可以以正确的模拟时间顺序处理。这个程序对于ELEVATOR和对于QUEUE表，也都使用双重链接表。

表示每一个活动（不论是人的还是电梯的活动）的节点，有形式

+	IN	LLINK1	RLINK1
+	NEXTTIME		
+	NEXTINST	0	0	39
+	OUT	LLINK2	RLINK2

（6）

这里LLINK1和RLINK1是对于WAIT表的链接；LLINK2和RLINK2被用作在QUEUE表或ELEVATOR表中的链接。这后两个场和IN以及OUT场，当节点（6）表示一个人的动作时才是相干的；但对于表示电梯动作的节点，它们都是不相干的。节点的第三个字，实际上是MIX的一条“JMP（转移）”指令。

图12示出了WAIT表，ELEVATOR表，以及QUEUE表之一的典型内容；QUEUE表中的每个节点，同时地是在WAIT表中，其有NEXTINST＝M4者；但这在图中并未能指出，因为链接的复杂性，将会掩盖基本的思想。

现在让我们来考虑程序本身。程序很长，尽管（如同所有的长程序一样）它被分成一些小的部分，而每个小的部分本身是很简单的。首先出现一些代码行，它仅用来定义这些表格的初始内容。这里有若干有趣之点：我们有对于WAIT表（行10-11），QUEUE表（行26-31），以及ELEVATOR表（行32-33）的表头。这些表头的每一个都是形如（6）的一个节点，但删去了不重要的字；WAIT的表头仅含一个节点的头两个字，而QUEUE和ELEVATOR的表头仅需要一个节点的最后一个字。我们还有在系统中总要出现的四

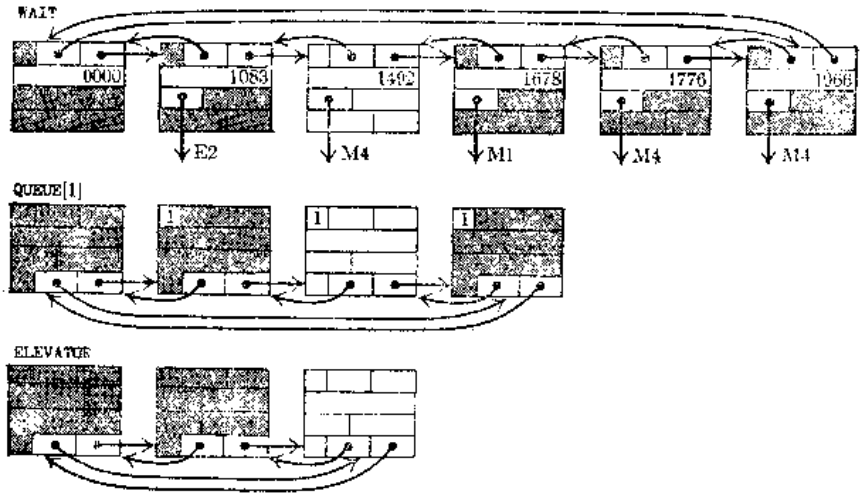


图12　用于电梯模拟程序中的某些表（表头出现在左边）

个节点（行 12-23）：MAN1，一个总是位于步骤M1，准备为一个新的人进入到本系统中的节点；ELEV1，一个在步骤E1，E2，E3，E4，E6，E7 和E8 支配电梯的主要动作的节点；以及 ELEV2 和ELEV3，用于电梯动作E5 和E9 的节点。这些动作的发生是相对于模拟时间的，与其它的电梯动作无关。这四个节点的每一个仅含三个字，因为它们决不出现在QUEUE 或 ELEVATOR 表中。表示系统中每一个实际的人的节点，将出现在紧跟主程序之后的一个存储池中。

```
01    * THE ELEVATOR SIMULATION                          (电梯的模拟)
02    IN          EQU   1:1                              定义在节点内的场
03    LLINK1      EQU   2:3
04    RLINK1      EQU   4:5
05    NEXTINST    EQU   0:2
06    OUT         EQU   1:1
07    LLINK2      EQU   2:3
08    RLINK2      EQU   4:5
09    * FIXED-SIZE TABLES AND LIST HEADS
10    WAIT        CON   *+2(LLINK1),*+2(RLINK1)          WAIT 表的表头
11                CON   0                                   总是NEXTTIME = 0
12    MAN1        CON   *-2(LLINK1),*-2(RLINK1)          这个节点表示动作A1
13                CON   0                                   而且开始时它是 WAIT
14                JWP   M1                                  表中的仅有条款
15    ELEV1       CON   0                                这个节点表示电梯的
16                CON   0                                   动作，但E5 和
17                JMP   E1                                  E9 除外
18    ELEV2       CON   0                                这个节点表示在E5
19                CON   0                                   处的独立的
20                JMP   E5                                  电梯动作
21    ELEV3       CON   0                                这个节点表示在E9
```

```
22                  CON     0                                   处的独立的
23                  JMP     E9                                  电梯动作
24      AVAIL       CON     0                                   链接到可利用的节点
25      TIME        CON     0                                   当前模拟的时间
26      QUEUE       EQU     * - 3
27                  CON     * - 3 (LLINK2), * - 3 (RLINK2)      QUEUE(0)的表头
28                  CON     * - 3 (LLINK2), * - 3 (RLINK2)      QUEUE(1)的表头
29                  CON     * - 3 (LLINK2), * - 3 (RLINK2)      所有排队开始时皆
30                  CON     * - 3 (LLINK2), * - 3 (RLINK2)        为空
31                  CON     * - 3 (LLINK2), * - 3 (RLINK2)      QUEUE(4)的表头
32      ELEVATOR    EQU     * - 3
33                  CON     * - 3 (LLINK2), * - 3 (RLINK2)      ELEVATOR的表头
34                  CON     0
35                  CON     0                                   对CALL表格的"补空"
36                  CON     0                                     (见行183-186)
37                  CON     0
38      CALL        CON     0                                   CALLUP(0), CALLCAR(0),
                                                                  CALLDOWN(0)
39                  CON     0                                   CALLUP(1), CALLCAR(1),
                                                                  CALLDOWN(1)
40                  CON     0                                   CALLUP(2), CALLCAR(2),
                                                                  CALLDOWN(2)
41                  CON     0                                   CALLUP(3), CALLCAR(3),
                                                                  CALLDOWN(3)
42                  CON     0                                   CALLUP(4), CALLCAR(4),
                                                                  CALLDOWN(4)
43                  CON     0
44                  CON     0                                   对CALL表格的"补空"
45                  CON     0                                     (见行178-181)
46                  CON     0
47      D1          CON     0                                   表示门打开，活动
48      D2          CON     0                                   表示延长时停止
49      D3          CON     0                                   表示门打开，不活动
```

程序编码的下一部分，包含基本的子程序和对于模拟过程的主控制程序。子程序INSERT（插入）和DELETE（删去）实现对于双重链接表的典型操作；它们把当前的节点放进一个QUEUE或ELEVATOR表中，或者从那儿取出来（在这个程序中，"当前节点"总是由变址寄存器6表示）。对于WAIT表也有些子程序：子程序SORTIN把当前的节点加到WAIT表，根据它的NEXTTIME场把它排到正确的位置。子程序IMMED把当前的节点插在WAIT表的前头。子程序HOLD把当前的节点放到WAIT表中，同时置NEXTTIME等于当前时间加上寄存器A中的数量。子程序DELETEW从WAIT表删去当前的节点。

程序CYCLE是模拟控制的核心：它判定下一步应该激发哪一个活动（即是，WAIT表的头一个元素，如果我们已知该表是非空的），而且转移到它。对于CYCLE有两个特殊的入口：CYCLE1首先置当前的节点中的NEXTINST、而HOLDC则等同于对HOLD子程序的一个额外调用。于是，设寄存器A中是t，则指令“JMP HOLDC”之效果，是将活动推迟t个模拟时间单位，然后返回到下面的位置。

```
50      *SUBROUTINES  AND  CONTROL  ROUTINE
51  INSERT      STJ     9F                  把节点NODE(C)插到
                                                NODE(rI1)左边；
52              LD2     3,1(LLINK2)         rI2←LLINK(rI1)
53              ST2     3,6(LLINK2)         LLINK(C)←rI2
54              ST6     3,1(LLINK2)         LLINK(rI1)←C
55              ST6     3,2(RLINK2)         RLINK(rI2)←C
56              ST1     3,6(RLINK2)         RLINK(C)←rI1
57  9H          JMP     *                   子程序出口
58  DELETE      STJ     9F                  由它的表中删去NODE(C)；
59              LD1     3,6(LLINK2)         P←LLINK2(C)
60              LD2     3,6(RLINK2)         Q←RLINK2(C)
61              ST1     3,2(LLINK2)         LLINK2(Q)←P
62              ST2     3,1(RLINK2)         RLINK2(P)←Q
63  9H          JMP     *                   子程序出口
64  IMMED       STJ     9F                  把NODE(C)首先插入
                                                WAIT表
65              LDA     TIME
66              STA     1,6                 置NEXTTIME(C)←TIME
67              ENT1    WAIT                P←LOC(WAIT)
68              JMP     2F                  把NODE(C)插到
                                                NODE(P)右边
69  HOLD        ADD     TIME                rA←TIME+rA
70  SORTIN      STJ     9F                  把NODE(C)分类到WAIT
                                                表中；
71              STA     1,6                 置NEXTTIME(C)←rA
72              ENT1    WAIT                P←LOC(WAIT)
73              LD1     0,1(LLINK1)         P←LLINK1(P)
74              CMPA    1,1                 比较NEXTTIME场，从右
                                                到左
75              JL      *-2                 重复直到NEXTTIME(C)≥
                                                NEXTIME(P)
76  2H          LD2     0,1(RLINK1)         Q←RLINK1(P)
77              ST2     0,6(RLINK1)         RLINK1(C)←Q
78              ST1     0,6(LLINK1)         LLINK1(C)←P
79              ST6     0,1(RLINK1)         RLINK1(P)←C
```

```
80                  ST6     0,2(LLINK1)          LLINK1(Q)←C
81          9H      JMP     *                    子程序出口
82          DELETEW STJ     9F                   从 WAIT 表删去NODE(C);
83                  LD1     0,6(LLINK1)          (这与行 58-63 相同，但使
84                  LD2     0,6(RLINK1)            用了 LLINK1,RLINK1 来
85                  ST1     0,2(LLINK1)            代替 LLINK2,RLINK2)
86                  ST2     0,1(RLINK1)
87          9H      JMP     *
88          CYCLE1  STJ     2,6(NEXTINST)        置 NEXTINST(C)←rJ
89                  JMP     CYCLE
90          HOLDC   STJ     2,6(NEXTINST)        置 NEXTINST(C)←rJ
91                  JMP     HOLD                 把 NODE 插入 WAIT,
                                                   延迟(rA)
92          CYCLE   LD6     WAIT(RLINK1)         置当前节点 C←RLINK1
                                                   (LOC(WAIT))
93                  LDA     1,6                  NEXTTIME(C)变成
94                  STA     TIME                   模拟的 TIME 的新值
95                  JMP     DELETEW              从 WAIT 表取走 NODE(C)
96                  JMP     2,6                  跳到 NEXTINST(C)
```

现在来到共行程序M的程序。在步骤M1的开始处，当前的节点C是MAN1（见上面的行12-14），而且本程序M的行099-100使MAN1被重新插入到WAIT表中，使当INTERTIME个模拟时间单位之后，将产生下一个人。下面的101-114行负责为最新产生的人建立一个节点；他的IN和OUT层被记录在这个节点的位置上。AVAIL堆栈单重地链接于每个节点的RLINK1场内。注意行101-108利用POOLMAX技术2.2.3-(7)实现动作"C⇐AVAIL"；在这里，没有必要来检验OVERFLOW，因为存储池的总大小（在任何一个时间里本系统中的人数）很少超过10个节点(40个字)。一个节点到AVAIL堆栈之回返，则出现于行156-158。

整个这个程序，始终是，变址寄存器4就等于变量FLOOR，而且取决于STATE=(GOINGUP, GOINGDOWN, 或NEUTRAL，变址寄存器5分别地为正，负，或0。变量CALLUP[j]，CALLCAR[j]，和CALLDOWN[j]占有单元CALL+j的分别的场(1:1)，(3:3)，和(5:5)。

```
097     * COROUTINE          M                    M1.进入，为后来人作准备
098     M1          JMP     VALUES               计算 IN, OUT, INTERTI-
                                                   ME, GIVEUPTIME
099                 LDA     INTERTIME            INTERTIME 由 VALUES
                                                   子程序计算
100                 JMP     HOLD                 置 NODE(C) 于 WAIT 中，
                                                   延迟 INTERTIME
101                 LD6     AVAIL                C←AVAIL
102                 J6P     1F                   如果 AVAIL≠Λ，则跳
```

```
103             LD6     POOLMAX
104             INC6    4               C←POOLMAX + 1
105             ST6     POOLMAX         POOLMAX←C
106             JMP     * + 3
107     1H      LDA     0,6(RLINK1)
108             STA     AVAIL           AVAIL←RLINK1 (AVAIL)
109             LD1     INFLOOR         r11←—INFLOOR (由上边的
                                          VALUES计算)
110             ST1     0,6(IN)         IN(C)←r11
111             LD2     OUTFLOOR        r12←OUTFLOOR(由
                                          VALUES 计算)
112             ST2     3,6(OUT)        OUT(C)←r12
113             ENTA    39              置常数 39(JMP 的操作码)
                                          于节点
                                          格式(6)的第一字
114             STA     2,6
115     M2      ENTA    0,4             M2. 发信号并等候。置 rA←
                                          FLOOR
116             DECA    0,1             FLOOR-IN
117             ST6     TEMP            保存C的值
118             JANZ    2F              如果 FLOOR≠IN，则跳
119             ENT6    ELEV1           置 C←LOC(ELEV1)
120             LDA     2,6(NEXTINST)   电梯处于E6的位置?
121             DECA    E6
122             JANZ    3F
123             ENTA    E3              若然，则把它重新放到E3处
124             STA     2,6(NEXTINST)
125             JMP     DELETEW         从 WAIT 表取走它
126             JMP     4F                  并把它重新插到 WAIT 前头
127     3H      LDA     D3
128             JAZ     2F              如果 D3 = 0，则跳
129             ST6     D1              否则置 D1≠0
130             STZ     D3              置 D3←0
131     4H      JMP     IMMED           把 ELEV1插入到WAIT 前头
132             JMP     M3              (r11，r12 已经改变)
133     2H      DEC2    0,1             r12←OUT-IN
134             ENTA    1
135             J2P     * + 3           如果正在上升，则跳
136             STA     CALL,1(5:5)     置 CALLDOWN[IN]←1
137             JMP     * + 2
138             STA     CALL,1(1:1)     置 CALLUP[IN]←1
139             LDA     D2
140             JAZ     DECISION        如果 D2 = 0，则调用
                                          DECISION 子程序
```

```
141                 LDA   ELEV1+2(NEXTINST)
142                 DECA  E1                    如果电梯处于E1,则调用
143                 JAZ   DECISION                 DECISION 子程序
144    M3           LD6   TEMP                  M3.进入排队
145                 LD1   0,6(IN)
146                 ENT1  QUEUE,1               r11←LOC(QUEUE[IN])
147                 JMP   INSERT                把 NODE(C)插入 QUEUE
                                                  [IN]的右端
148    M4A          LDA   GIVEUPTIME
149                 JMP   HOLDC                 等候 GIVEUPTIME 个单位
150    M4           LDA   0,6(IN)               M4.放弃
151                 DECA  0,4                   IN(C) - FLOOR
152                 JANZ  *+3
153                 LDA   D1                    FLOOR = IN(C)
154                 JANZ  M4A                   见习题 7
155    M6           JMP   DELETE                M6.离去.从 QUEUE 或
156                 LDA   AVAIL                   ELEVATOR 删去 NODE(C)
157                 STA   0,6(RLINK1)           AVAIL←C
158                 ST6   AVAIL
159                 JMP   CYCLE                 继续模拟
160    M5           JMP   DELETE                M5.进入.从 QUEUE 删去
161                 ENT1  ELEVATOR                NODE(C)
162                 JMP   INSERT                把它插入 ELEVATOR的右边
163                 ENTA  1
164                 LD2   3,6(OUT)
165           STA   CALL,2(3:3)                 置 CALL[CAROUT(C)]←1
166           J5NZ  CYCLE                       如果 STATE≠NEUTRAL, 则跳
167           DEC2  0,4
168           ENT5  0,2                         置 STATE 成正确的方向
169           ENT6  ELEV2                       置 C←LOC(ELEV2)
170           JMP   DELETEW                     从 WAIT 表撤消 E5 动作
171           ENTA  25
172           JMP   E5A                         从现在开始重新启动E5动作 25 个单位
```

对于共行程序E的程序，是以前给出的半形式的描述的颇为直接了当的翻译。也许最有趣的部分，是在步骤 E3 中对于电梯的独立动作的准备，以及在步骤 E4 中检索 ELEVATOR 和 QUEUE 表。

```
173  *COROUTINE    E
174  E1A     JMP   CYCLE1                      置 NEXTINST←E1, 转到 CYCLE
175  E1      EQU   *                           E1.等候呼叫. (无动作)
176  E2A     JMP   HOLDC
177  E2      J5N   1F                          E2.要改变状态?
178          LDA   CALL+1,4                    状态是 GOINGUP
```

```
179             ADD     CALL+2, 4
180             ADD     CALL+3, 4
181             ADD     CALL+4, 4
182             JAP     E3              有更高层的呼叫?
183             LDA     CALL-1,4(3:3)   若无,则电梯中有要求到更低
184             ADD     CALL-2,4(3:3)       层的乘客?
185             ADD     CALL-3,4(3:3)
186             ADD     CALL-4,4(3:3)
187             JMP     2F
188     1H      LDA     CALL-1,4        状态是 GOINGDOWN
189             ADD     CALL-2,4        动作类似行178-186
...
196             ADD     CALL+1,4(3:3)
197     2H      ENN5    0,5             颠倒 STATE 的方向
198             STZ     CALL,4          置 CALL 变量成 0
199             JANZ    E3              如果相反方向呼叫,则跳,
200             ENT5    0                 否则,置 STATE←NEUTRAL
201     E3      ENT6    ELEV3           E3.开门
202             LDA     0, 6            如果活动 E9 已调度,则从 WAIT
203             JANZ    DELETEW           表撤消它
204             ENTA    300
205             JMP     HOLD            在300个单位之后,调度活动 E9
206             ENT6    ELEV2
207             ENTA    76
208             JMP     HOLD            在 76 个单位之后,调度活动 E5
209             ST6     D2              置 D2≠0
210             ST6     D1              置 D1≠0
211             ENTA    20
212     E4A     ENT6    ELEV1
213             JMP     HOLDC
214     E1      ENTA    0,4             E4.让人出入
215             SLA     4               置 rA 的 OUT 场为 FLOOR
216             ENT6    ELEVATOR        C←LOC (ELEVATOR)
217     1H      LD6     3,6(LLINK2      C←LLINK2 (C)
218             CMP6    =ELEVATOR=      从右到左检索 ELEVATOR 表
219             JE      1F              如果 C=LOC (ELEVATOR),则检索完成
220             CMPA    3, 6 (OUT)      比较 OUT(C)与 FLOOR
221             JNE     1B              如果不等,则继续检索,
222             ENTA    M6                否则,准备把人送到 M6
223             JMP     2F
224     1H      LD6     QUEUE+3,4(RLINK2)   置 C←RLINK2 (LOC(QUEUE[FLOOR]))
225             CMP6    3,6 (RLINK2)    C≠RLINK2(C)?
```

```
226             JE      1F              若是，则排队为空
227             JMP     DELETEAW        若否，则对这个人消去动作M1
228             ENTA    M5              准备把人送到M5
229     2H      STA     2,6 (NEXTINST)  置 NEXTINST (C)
230             JMP     IMMED           把它放到 WAIT 表前头
231             ENTA    25
232             JMP     E4A             等候 25 个单位并重复 E4
233     1H      STZ     D1              置 D1 ← 0
234             ST6     D3              置 D3≠ 0
235             JMP     CYCLE           返回去模拟其它事件
236     E5A     JMP     HOLDC
237     E5      LDA     D1              E5. 关门
238             JAZ     *+3             D1 = 0?
239             ENTA    40              若否，则人仍然正在进入或出去
240             JMP     E5A             等候 40 个单位，重复 E5
241             STZ     D3              如果 D1 = 0，则置 D3← 0
242             ENTA    20
243             ENT6    ELEV1
244     E6A     JMP     HOLDC           等候 20个单位，然后转到E6
245     E6      J5N     *+2             E6. 准备移动
246             STZ     CALL,4(1:3)     如果 STATE≠GOINGDOWN，则这一
                                            层上的 CALLUP 和 CALLCAR 被复位
247             J5P     *+2
248             STZ     CALL,4(3:5)     如果≠GOINGUP，则复位 CALLCAR
                                            和 CALLDOWN
249             J5Z     DECISION        实施 DECISION 子程序
250     E6B     J5Z     E1A             如果 STATE = NEUTRAL，则转到E1 并等候
251             LDA     D2
252             JAZ     *+4
253             ENT6    ELEV2           否则，如果 D2≠0，
254             JMP     DELETEW             则消去活动 E9
255             STZ     ELEV3               (见行 202)
256             ENTA    15
257             ENT6    ELEV1           等候15个时间单位
258             J5N     E8A             若 STATE = GOINGDOWN，则转 E8
259     E7A     JMP     HOLDC
260     E7      INC4    1               E7. 上升一层
261             ENTA    51
262             JMP     HOLDC           等候 51 个单位
263             LDA     CALL,4(1:3)     CALLCAR(FLOOR)或
                                            CALLUP(FLOOR)≠0?
264             JAP     1F
265             ENT1    -2,4            若否，则
266             JIZ     2F                  FLOOR = 2?
```

```
267             LDA     CALL, 4(5:5)        若否，CALLDOWN[FLOOR]≠0?
268             JAZ     E7                  若否，则重复步骤E7
269     211     LDA     CALL+1, 4
270             ADD     CALL+2, 4
271             ADD     CALL+3, 4
272             ADD     CALL+1, 4
273             JANZ    E7                  有更高层的呼叫?
274     111     ENTA    14                  它是停止电梯的时间
275             JMP     E2A                 等候14个单位并转到E2
276     E8A     JMP     HOLDC
...                                         (见习题8)
293     E9      STZ     0, 6                E9.设置不活动指示器
294             STZ     D2                  D2←0
295             JMP     DECISION            实施DECISION子程序
296             JMP     CYCLE               返回去模拟其它事件
```

这里，我们将不考虑 DECISION 子程序（见习题 9），也不考虑用来确定关于电梯的要求的 VALUES 子程序。在这个程序的最后有代码

```
BEGIN       ENT4    2       起动于 FLOOR = 2 和
            ENT5    0           STATE = NEUTRAL
            JMP     CYCLE   开始模拟
POOLMAX     END     BEGIN   存储池紧跟文字、临时存贮
```

上述程序，随着它通过各程序步，便实现了精美地模拟电梯系统的工作。但是，运行这个程序是没有用的，因为没有输出！实际上，作者曾附了一个 PRINT 子程序，在上述程序的大多数临界步骤处调用之。而且，这个子程序被用来准备表 1；其细节已被省略，因为它们是非常直接了当的，只不过是填填代码而已。

一直在设计了若干程序设计语言，使其易于描述离散模拟中的动作，并十分容易地使用编译程序来把这些描述翻译成机器语言。当然，在这一节里我们使用的是汇编语言，因为我们在这里所关心的，是操作链接表的技术，以及怎样用一台计算机来真正实现离散模拟的细节（虽然说这已经有些专一的意向了）。我们将在第 8 章中，来考虑用于描述这些系统的高级记号的问题。使用一个 WAIT 表或“议事日程”来控制共行程序的顺序，如同我们已经在这一节中所作的那样，这样的技术称作拟并行处理。

对于这样一个长长的程序，要精确地分析其运行时间，是十分困难的；因为这涉及到错综复杂的交互动作。可是，对程序的各个较小部分（象 INSERT 子程序）进行计时，则是很容易的，而且这给出了它的效率的指标。使用一个特殊的跟踪程序，它执行这个程序，而且记录每条指令被执行次数之多寡，通常是有用的；这样就能看出程序中的“瓶子口”，即应予特殊地注意的地方。作者曾以上边的程序进行过这样一个实验：这个程序运行了 10000 个模拟时间单位，并有 26 人进入此模拟系统。在 SORTIN 循环中的那些指令，行 73–75，是执行得最经常的，达 1432 次，而 SORTIN 子程序本身被调用 437 次。CYCLE 程序被执行 407 次，而且这提示了，DELETEW 子程序不应该在行 95 处来调用，该子程序的此 4 行应该全部写出（以便每次使用 CYCLE 时节省 4 u）。特殊的跟踪程序也说明了，DECISION 子程序仅被调用 32 次，而 E4 中的循环（行216–218）仅被执行 142 次。

希望读者们，将象作者为准备这例子时学习有关电梯的知识那样，从上边的例子中学到尽量多的有关模拟的知识。

习题

1.〔21〕 对于如同(1)中所表示的一个双重链接表，试给出在其左端进行信息的插入和删去的说明（连同在右端的一对操作，它们可以通过对称性得到，我们从而就有了对于一般的双排队的所有操作)。

▶2.〔22〕 说明为什么一个单重链接的表，不能进行如同一般的双排队那样有效的操作；条款的删去仅能在单重链接表的一端有效地进行。

▶3.〔22〕 正文中所述的电梯系统，对于每一层，都使用了三个呼叫变量 CALLUP, CALLCAR, CALLDOWN，表示系统中的人们已经按下了什么按钮。可以想象，在内部，对于每层上的呼叫按钮，电梯仅需要一个或两个继电器(即是，二进变量)，而不是三个。试说明：对于这个电梯系统，人们可以以某个顺序来按下这些按钮，以证明对于每一层确有三个分开的继电器(最顶层和最底层除外)。

4.〔24〕 电梯共行程序中的活动 E9，常常为步骤 E6 所消除，而且即使当它未被消去时，它也不可能执行很多次。试说明在什么环境之下，如果活动 E9 从这个系统中删去的话，则电梯将不同地进行动作（即是说，电梯将以不同的速度进行操作，或者以不同的次序来访问各层)。

5.〔20〕 在表 1 中，第 10 人在时间 1048 时到达 0 层。假设他到达 2 层而不是到达 0 层；试说明在这些条件下，电梯在接收它在 1 层上的乘客之后将要上升，而不是下降，且不顾第 8 人要下到 0 层这一事实。

6.〔23〕 注意在表 1 中，时间 1183-1233，第 7，8 和 9 人都在 1 层进入了电梯；然后电梯下降到 0 层，而且仅仅第 8 人出去。现在电梯停在 1 层，总得运送已经在车上的第 7 人和第 9 人，而且实际上并没有人在 1 层等候进入(这种情况在加州理工学院不是少见的；如果你上了正往错误的方向行进的电梯，你就必须等候一次额外的停止，就如同你从原始的层再次出发一样)。在许多电梯系统中，第 7 和第 9 人将不在时间 1183 时进入电梯，因为电梯外边的灯将示出它正在下降，而不是上升；这些人将须等候，直到电梯回过头来往上，而且为他们而停止下来为止。在所述的系统中，如果没有这样的灯，就不可能来告诉电梯正在沿着哪个方向行进，直到你进入它时为止；因此表 1 反映了这实际的情况。

如果我们来模拟同一个电梯系统，但已带有指示灯，使当它的状态与人们所希望的方向相反时，人们就不进到电梯中去，那么，试问对于共行程序 M 和 E 应作什么改动？

7.〔25〕 尽管程序中的一些“故障”经常使程序员为难，但如果我们从我们的错误中来学习，则我们就应该记录它们并把它们告知其他人，而不是忘掉它们。以下的错误在其它当中是作者在头一次写这一小节中的程序时造成的：行 154 说成是“JANZ CYCLE”而不是“JANZ M4A”。理由是，如果电梯确已到达这个人所在的层，而且他不久就有可能进入，则就没有任何必要对他来实施他的“放弃”活动 M4，所以我们可以简单地转到 CYCLE，并且继续模拟其它的活动。错误是什么？

8.〔22〕 对于步骤 E8，写出行277-292的代码，这段程序在正文中的程序里已被省略。

9.〔23〕 写出在正文里的程序中已被省略了的 DECISION 子程序的代码。

10.〔40〕 指出这样一点也许是有意义的，就是尽管作者已经使用电梯系统好些年了，而且我觉得已经很好地了解了它，但是直到我试图来写这一小节时为止，我才认识到，关于选定方向的电梯系统，还有一些细小的事实，是我还不知道的。我又回过头去对电梯进行了六次独立的实验，而且每一次我还都以为已经最终地对它的操作方法有了完全的了解（现在我们担心它的操作还会出现某些新的与给定的算法相矛盾的方面，以致我对于乘坐它已经很有顾虑了）。直到我们试图对一件事情在一台计算机上进行模拟时为止，我们通常并未能那样细致地知道这件事。

试描述你所熟悉的某些电梯的动作。通过对电梯本身的实验，来校验这个算法（不要去看它的线路）；然后对这系统来设计一个离散的模拟程序并在一台计算机上运行之。

▶11.〔25〕（"存储器内容更换"）在同步的模拟中经常出现以下的问题：系统有 n 个变量 V〔1〕，…，V〔n〕，而且在每一个模拟的步骤里，某些变量的新值是通过这些老值计算的。假定这些计算在这样的意义下是"同时"进行的，就是直到做完所有的赋值之后，这些变量才改变成它们的新值。于是，出现在同一个模拟时间里的两个语句

$$V〔1〕\leftarrow V〔2〕 \text{ 和 } V〔2〕\leftarrow V〔1〕$$

将交换 V〔1〕与 V〔2〕的值，这同顺序计算中所发生的情况是截然不同的。

通过一个附加的表格 NEWV〔1〕，…，NEWV〔n〕，当然可以模拟所希望的动作。在每个模拟步骤之前，对于 $1\leqslant k\leqslant n$ 我们可以置 NEWV〔k〕$\leftarrow$V〔k〕，然后把 V〔k〕的所有变化记录到 NEWV〔k〕中，而且最后，在这一步骤之后，我们可以置 V〔k〕$\leftarrow$NEWV〔k〕，$1\leqslant k\leqslant n$。但是，这种"蛮干"的方法通常不十分令人满意，盖有下列一些原因：（1）通常 n 非常大，但每一步改变的变量数又是比较小的。（2）变量通常不是排列成一个整齐的表格 V〔1〕，…，V〔n〕，而是以一种颇为混乱的方式散列到整个内存中。（3）这个方法不去检测在同一个模拟步骤中什么时候一个变量被给予两个值的情况（通常是模型中的一个错误）。

假设每步中变化的变量个数是比较小的，试设计一个用两个辅助的表格 NEWV〔k〕和 LINK〔k〕，$1\leqslant k\leqslant n$，来模拟所希望的动作的有效算法。倘有可能，如果在同一个步骤上，同一个变量被给以两个不同的值，则你的算法应该给出一个出错停机。

▶12.〔22〕 在这一节的模拟程序中，使用双重链接表而不使用单重链接或顺序的表，为什么说是一个好主意？

2.2.6 数组和正交表

线性表的最简单的推广之一，是二维或高维的信息数组。例如，考虑 $m\times n$ 矩阵

$$\begin{pmatrix} A〔1,1〕 & A〔1,2〕 & \cdots & A〔1,n〕 \\ A〔2,1〕 & A〔2,2〕 & \cdots & A〔2,n〕 \\ \vdots & & & \vdots \\ A〔m,1〕 & A〔m,2〕 & \cdots & A〔m,n〕 \end{pmatrix} \tag{1}$$

在这个二维数组中，每个节点 A〔j，k〕都属于两个线性表："行 j"的表 A〔j，1〕，A〔j，2〕，…，A〔j，n〕和"列 k"的表 A〔1，k〕，A〔2，k〕，…，A〔m，k〕。这些正交的行表和列表，实际上说明了一个矩阵的二维结构。类似的陈述亦可应用于更高维的信

总数组。

顺序分配 当一个数组存于顺序的内存单元时，存储通常被分配成使得

$$\mathrm{LOC}(A[J,K])=a_0+a_1J+a_2K \qquad (2)$$

其中 a_0，a_1 和 a_2 都是常数。让我们考虑更一般的情况：假设我们有对于 $0\leqslant I\leqslant 2$，$0\leqslant J\leqslant 4$，$0\leqslant K\leqslant 10$，$0\leqslant L\leqslant 2$ 的占一个字的元素 $Q[I,J,K,L]$ 的一个四维数组。我们将乐于把存储分配成使得

$$\mathrm{LOC}(Q[I,J,K,L])=a_0+a_1I+a_2J+a_3K+a_4L \qquad (3)$$

这意味着，I，J，K或L的一个变化即导致 $Q[I,J,K,L]$ 之单元的一个易于计算的变化。分配存储的最自然的（以及最普遍使用的）方式，是让数组以它的下标的"字典顺序"出现于内存中，这种顺序有时叫做"行优先顺序"：

$$\begin{aligned}
&Q[0,0,0,0],\ Q[0,0,0,1],\ Q[0,0,0,2],\\
&Q[0,0,1,0],\ Q[0,0,1,1],\ Q[0,0,1,2],\\
&\quad\cdots\qquad\quad\cdots\qquad\quad\cdots\\
&\quad\cdots\qquad\quad\cdots\qquad\quad,\ Q[0,0,10,2],\\
&Q[0,1,0,0],\qquad\cdots\qquad\quad\cdots\\
&\quad\cdots\qquad\quad\cdots\qquad\quad,\ Q[0,4,10,2],\\
&Q[1,0,0,0],\qquad\cdots\qquad\quad\cdots\\
&\quad\cdots\qquad\quad\cdots\qquad\quad,\ Q[2\ \ 4,10,2]。
\end{aligned}$$

容易看出，这种顺序满足(3)的要求，而且我们有

$$\mathrm{LOC}(Q[I,J,K,L])=\mathrm{LOC}(Q[0,0,0,0])+165I+33J+3K+L \qquad (4)$$

一般说来，给了一个对于 $0\leqslant I_1\leqslant d_1$，$0\leqslant I_2\leqslant d_2$，…，$0\leqslant I_k\leqslant d_k$ 的占 c 个字的元素 $A[I_1,I_2,\cdots,I_k]$ 的 k 维数组，我们能把它存于内存中使得

$$\begin{aligned}
\mathrm{LOC}(A[I_1,I_2,\cdots,I_k])&=\mathrm{LOC}(A[0,0,\cdots,0])\\
&\quad+c(d_2+1)\cdots(d_k+1)I_1+\cdots\\
&\quad+c(d_k+1)I_{k-1}+cI_k\\
&=\mathrm{LOC}(A[0,0,\cdots,0])+\sum_{1\leqslant r\leqslant k}a_rI_k
\end{aligned}$$

其中

$$a_r=c\prod_{r<s\leqslant k}(d_s+1) \qquad (5)$$

为了看出这个公式为什么行得通，我们指出，如果 I_1，…，I_r 是常数而且 J_{r+1}，…，J_k 跑遍 $0\leqslant J_{r+1}\leqslant d_{r+1}$，…，$0\leqslant J_k\leqslant d_k$ 的所有值，则 a_r 就是为存储子数组 $A[I_1,\cdots,I_r,J_{r+1},\cdots,J_k]$ 所需的内存数量；因此，按照字典顺序的本性，$A[I_1,\cdots,I_k]$ 的地址，当 I_r 变化了 1 时，应当精确地以这个数量进行改变。

注意公式(5)与在混合进制下的数 $I_1I_2\cdots I_k$ 的值之间的相似性。例如，如果我们有对于 $0\leqslant W<4$，$0\leqslant D<7$，$0\leqslant H<24$，$0\leqslant M<60$ 以及 $0\leqslant S<60$ 的数组 TIME[W, D, H, M, S]，则 TIME[W, D, H, M, S]的单元将是 TIME[0, 0, 0, 0, 0]的单元加上转换成秒的数量"W周＋D天＋H小时＋M分＋S秒"。当然，要有一台好的大型计算

机以及一个好的如意的应用，来利用一个有 2,419,200 个元素的数组。

用于存储数组的上述方法，一般地适合于当数组有一个完整的矩形结构时，就是说，当对于在独立的范围 $l_1 \leqslant 1_1 \leqslant u_1$，$l_2 \leqslant l_2 \leqslant u_2$，…，$l_k \leqslant l_k \leqslant u_k$ 内的下标，所有元素 A〔l_1，l_2，…，l_k〕都出现时。有许多不是这种情况的状态；就中最普通的是三角形矩阵，对于这种情形，我们仅需要存储对于那些比如说是 $0 \leqslant k \leqslant j \leqslant n$ 的项目 A〔j，k〕：

$$\begin{pmatrix} A[0,0] & & & \\ A[1,0] & A[1,1] & & \\ \vdots & & & \\ A[n,0] & A[n,1] & \cdots & A[n,n] \end{pmatrix} \tag{6}$$

我们可能已知所有其余的项目都为 0，或者已知 A〔j，k〕= A〔k，j〕，所以仅有一半的值需要存储。如果我们要把下三角形矩阵（6）存储于 $\frac{1}{2}(n+1)(n-2)$ 个连续的内存位置，则我们就将被迫放弃如象等式（2）中所示的线性分配的可能性；不过，我们现在可以要求替换成形如

$$\mathrm{LOC}(\mathrm{A}[\mathrm{J},\mathrm{K}]) = a_0 + f_1(\mathrm{J}) + f_2(\mathrm{K}) \tag{7}$$

的分配安排，其中 f_1 和 f_2 都是一个变量的函数（如果愿意，常数 a_0 可吸收到或 f_1 或 f_2 中）。当编址有形式（7）时，如果我们保持两个（相当短的）辅助的 f_1 和 f_2 的值表，使得这些函数仅需要被计算一次，则一个随机元素 A〔j，k〕即可迅速地被存取。

结果是，对于数组（6）的下标的字典顺序满足条件（7），而且，假定是一个字的项目，则我们事实上有着颇为简单的公式

$$\mathrm{LOC}(\mathrm{A}[\mathrm{J},\mathrm{K}]) = \mathrm{LOC}(\mathrm{A}[0,0]) + \frac{\mathrm{J}(\mathrm{J}+1)}{2} + \mathrm{K} \tag{8}$$

如果我们足够幸运，竟有相同大小的两个三角形矩阵，则我们就有尤其美好的方式来存储这两三角形矩阵。如果 A〔j，k〕和 B〔j，k〕两者都是对于 $0 \leqslant k \leqslant j \leqslant n$ 有待存储的，则利用约定

$$\mathrm{A}[j,k] = \mathrm{C}[j,k],\quad \mathrm{B}[j,k] = \mathrm{C}[k,j+1] \tag{9}$$

我们就能把它们两个填入一个对于 $0 \leqslant j \leqslant n$，$0 \leqslant k \leqslant n+1$ 的单个矩阵 C〔j，k〕。于是

$$\begin{pmatrix} C[0,0] & C[0,1] & C[0,2] & \cdots & C[0,n+1] \\ C[1,0] & C[1,1] & C[1,2] & \cdots & C[1,n+1] \\ \vdots & & & & \vdots \\ C[n,0] & C[n,1] & C[n,2] & \cdots & C[n,n+1] \end{pmatrix}$$

$$\begin{pmatrix} A[0,0] & B[0,0] & B[1,0] & \cdots & B[n,0] \\ A[1,0] & A[1,1] & B[1,1] & \cdots & B[n,1] \\ \vdots & & & & \vdots \\ A[n,0] & A[n,1] & A[n,2] & \cdots & B[n,n] \end{pmatrix}$$

两个三角形矩阵被一起密集地填塞到 $(n+1)(n-2)$ 个单元的空间内，而且我们有如（2）所示的线性编址。

三角形矩阵到更高的维数的推广，就称作四面体数组。这个有趣的课题是习题 6 到 8 的主题。

作为使用顺序地存储的数组的一个典型的程序设计技巧的例子，见习题 1.3.2-10 和对于此题给出的两个答案。为有效地遍历行和列，以及使用顺序的堆栈的基本技巧，在那些程序内，是特别有趣的。

链接分配 链接的内存分配也以一种自然的方式应用到高维的信息数组。一般说来，我们的节点能包括 k 个链接场，对于此节点所在的每一个表各一个。链接内存，一般地是对于数组本质上并不是严格地呈矩形的那些情况来使用的。

作为一个例子，假设我们有一个表，在此表中每个节点是表示一个人（PERSON），并且假设有四个链接场，即SEX（性别），AGE（年龄），EYES（眼睛）和HAIR（头发）。在EYES场中，我们把有着相同眼色的所有节点链接在一起，等等(见图13)。不难想到为把新的人插进这个表的算法（如若没有双重链接，则删去将是较困难的）。我们也能想到这样的算法，即去做诸如"找21到23岁的所有兰眼睛金发的女人"这样一些事情的各种不同有效程度的算法；见习题9和10。一个表的每个节点同时又归属于若干个其它类型的表当中，这样的问题是较为经常地出现的。其实，在上节中所述的电梯系统的模拟中，就有同时在QUEUE和WAIT两个表中的节点。

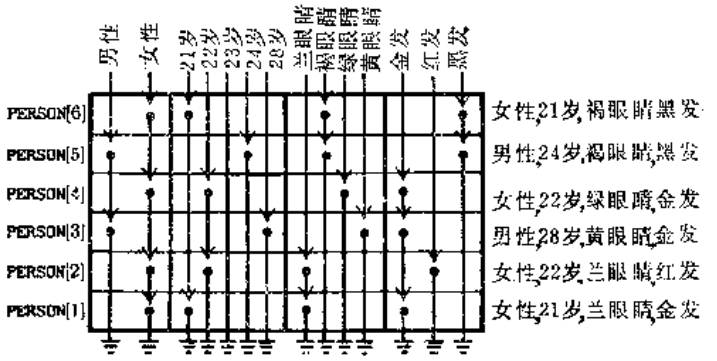
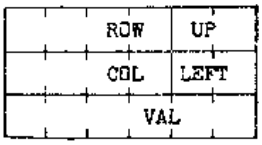


图13 在四个不同的表下的每个节点

作为一个详细的对正交表使用链接分配的例子，我们来考虑稀疏矩阵（即是，其中大多数元素皆为0的大阶矩阵）的情况。目的是，想象整个矩阵都出现，要对这种矩阵进行操作，但是0元素却又不需要表示，以便节省大量的存储空间。为做到这一点，有一个办法，这就是要随机地来访问矩阵的元素，而这将使用第6章的存储并检索的方法，来从链"i，k"找出A[i，k]；然而，为处理稀疏矩阵，还有另一个办法，而且由于它更适当地反映了矩阵的结构，通常它是更受欢迎的，我们在这里就来讨论这个方法。

我们将讨论的表示，由对于每行和每列的循环链接表组成。矩阵的每一节点包含三个字和五个场：



(10)

这里ROW和COL是节点的行和列的下标；VAL是存储在矩阵该处的值；LEFT和UP分别为指向这行左边或这列上边的下一个非0条款的链接。对每行和每列，都有特殊的表头节点BASEROW[i]和BASECOL[j]。这些节点，通过

$$\mathrm{COL}(\mathrm{LOC}(\mathrm{BASEROW}[i]))<0 \text{ 和 } \mathrm{ROW}(\mathrm{LOC}(\mathrm{BASECOL}[j]))<0$$

来标识。象通常在一个循环表中那样，BASEROW[i]的LEFT链接是该行的最右边值的单元，而BASECOL[j]中的UP指向该列中最底下的值。例如，矩阵

$$\begin{pmatrix} 50 & 0 & 0 & 0 \\ 10 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 \\ -30 & 0 & -60 & 5 \end{pmatrix} \tag{11}$$

将被表示成如图 14 中所示。

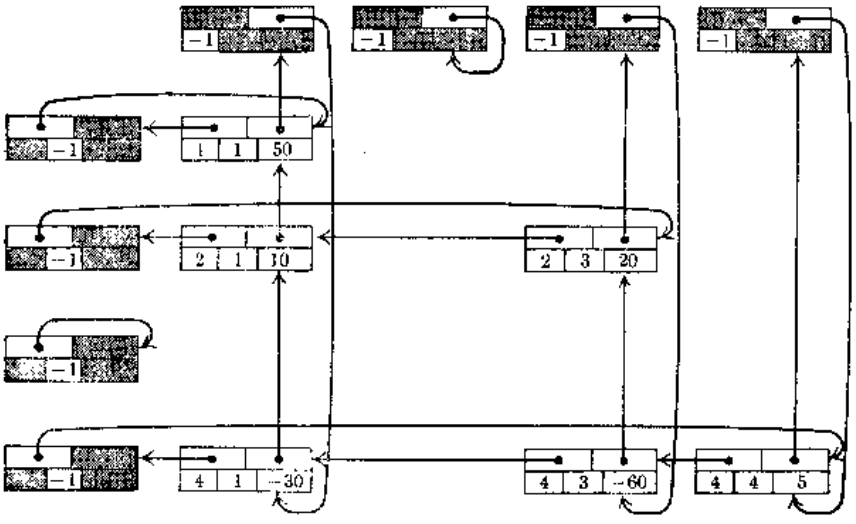
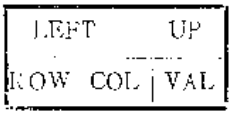


图14 矩阵(11)的表示；这些节点以格式



来解释，表头出现于左边和顶上

如果使用顺序的存储分配，则一个 200×200 的矩阵将花 40000 个字，而这比许多计算机所拥有的内存还要多；但是一个相当稀疏的 200×200 矩阵，甚至在 MIX 的 4000 个字的内存中也能如上地表示之（见习题 11）。为存取一个随机的元素 $A[i,k]$ 所花费的时间也是十分合理的，如果在每行或列中没有几个元素的话；不过，因为大多数矩阵的算法，都是通过顺序地走过一个矩阵来进行的，而不是随机地存取元素。因此，这个链接的表示，只蒙受运行速度的很小损失。

作为一个以这种记法来处理稀疏矩阵的一个非平凡的算法的典型例子，我们来考虑主要步骤的操作，它是为解线性方程组，为求矩阵之逆，和为通过单纯形法解线性规划问题等等算法的一个重要部分。一个主要步骤就是如下的矩阵变换：

$$\begin{array}{c|ccc|ccc}
 & \text{在主要步骤之前} & & & \text{经主要步骤之后} & & \\
 & \text{主要列} & & \text{任何其它列} & \text{主要列} & & \text{任何其它列} \\
\text{主要行} & a & \cdots & b & 1/a & \cdots & b/a \\
\text{任何其它行} & c & \cdots & d & -c/a & \cdots & d-\dfrac{bc}{a}
\end{array} \tag{12}$$

假定“主元素”a 非 0。例如，对矩阵(11)应用一个主要步骤，以行 2 列 1 中的元素 10 作为主元素，就导致

$$\begin{pmatrix}
-5 & 0 & -100 & 0 \\
0.1 & 0 & 2 & 0 \\
0 & 0 & 0 & 0 \\
3 & 0 & 0 & 5
\end{pmatrix} \tag{13}$$

我们的目的是来设计一个算法，这个算法要对被表示成如图 14 的稀疏矩阵实施这个主要步骤操作。显然，变换(12)仅仅影响一个矩阵的那些在主要列中有一个非 0 元素的行，以及仅仅影响那些在主要行中有一个非 0 元素的列。因此，当考虑一个很大的稀疏矩阵时，通过对于非 0 元素的链接表示，我们不仅实现了空间上的缩减，而且也许还实现了进行主要步骤的速度上的增进。

主要步骤的算法，在许多方面，都是我们所已讨论过的链接技术的一个直接了当的应用；特别是，它强烈地类似于对于多项式加法的算法 2.2.4 A。然而，有两件事情，它使得这问题有点技巧：如果在(12)中我们有 $b \neq 0$ 和 $c \neq 0$，但 $d = 0$，则这稀疏矩阵的表示对于 d 没有条款，而现在我们却必须插入一个新的元素；而如果 $b \neq 0$，$c \neq 0$，$d \neq 0$，但 $d - bc/a = 0$，则我们必须从原来有条款的此处把此条款删去。这些在二维数组中的插入和删去操作，比起一维数组的情况，更为有趣；为了进行这些操作，我们必须知道什么链接受到影响。我们的算法，由下到上逐步地处理矩阵的诸行。为能有效地插入和删去，这里包括引进一组指针变量 PTR[j]，对于所考虑的每一列各一个；这些变量向上遍历这些列，同时给我们提供了在两个维数中来更新适当的链接之能力。

算法 S（稀疏矩阵中的主要步骤） 给出如图 14 所表示的那样一个矩阵，我们来实施主要操作(12)。假定 PIVOT(主要)是一个指向主元素的链接变量。这个算法利用了链接变量 PTR[j]的一个辅助的表格，对矩阵的每个列各一个。

S1.〔初始化〕 置 I0←ROW(PIVOT)，J0←COL(PIVOT)，ALPHA←1.0/VAL(PIVOT)，VAL(PIVOT)←1.0，P0←LOC(BASEROW[I0])，Q0←LOC(BASECOL[J0])（注意：变量 ALPHA 和每个节点的 VAL 场假定都是浮点的或有理的量，同时，在这个算法中其它一切又都有整数的值）。

S2.〔处理主要行〕 置 P0←LEFT(P0)，J←COL(P0)。如果 J<0，则转到步骤S3（已经遍历了这主要行）。否则置 PTR[J]←LOC(BASECOL[J]) 及 VAL(P0)←ALPHA×VAL(P0)，并重复步骤 S 2。

S3.〔找新的行〕 置Q0←UP(Q0)(算法的剩下部分从底到顶地逐行处理这样的每行，即对于这些行，它都有一个条款在主要列中)。置 I←ROW(Q0)。如果 I<0，则本算法终止。如果 I=J0，则重复步骤 S 3（我们已经做完了这主要行）。否则，置 P←LOC(BASEROW[I])，P1←LEFT(P)（指针 P 和 P1 现在将从右至左地横行通过行 I，随着 P0 同步地横行通过行 I0；算法 2.2.4A 是相似的。这时，

$$P0 = LOC(BASEROW[I0]).)$$

S4.〔找新的列〕 置P0←LEFT(P0)，J←COL(P0)。如果 J<0，则置 VAL(Q0)←−ALPHA×VAL(P0) 并返回 S 3。如果 J=J0，则重复步骤 S 4（这样，我们是在已经处理了行 I 中的所有其它列的条款之后，才来处理行 I 中的主要列条款；原因是在步骤 S7 中需要 VAL(Q0)）。

S5.〔找 I，J 元素〕 如果 COL(P1)>J，则置 P←P1，P1←LEFT(P)，并重复步骤 S 5。如果 COL(P1)=J，则转到步骤 S 7。否则转到步骤 S 6（我们需要在行 I 的第 J 列处插入一个新元素）。

S6.〔插入 I，J 元素〕 如果 ROW(UP(PTR[J]))>I，则置PTR[J]←UP(PTR[J])，

并重复步骤S6（否则，我们将有ROW(UP(PTR[J]))<I；就在垂直的维数中的NODE(PTR[J])的紧上边，同时就在水平维数中的NODE(P)的紧左边，有待插入新的元素）。否则置X←AVAIL，VAL(X)←0，ROW(X)←I，COL(X)←J，LEFT(X)←P1，UP(X)←UP(PTR[J])，LEFT(P)←X，UP(PTR[J])←X，P1←X。

S7. 〔主要步骤〕 置VAL(P1)←VAL(P1)−VAL(Q0)×VAL(P0)。如果现在VAL(P1)=0，则转到S8（注意，当正在使用的是浮点算术时，这个测试“VAL(P1)=0”应当代之以“|VAL(P1)|<EPSILON”，或者更好地，代之以条件“VAL(P1)的大多数有效数字已在减法中消失”）。否则，置PTR[J]←P1，P←P1，P1←LEFT(P)，并转回到S4。

S8. 〔删去I，J元素〕 如果UP(PTR[J])≠P1（或者，实际上同样的，如果ROW(UP(PTR[J]))>I），则置PTR[J]←UP(PTR[J])并重复步骤S8；否则，置UP(PTR[J])←UP(P1)，LEFT(P)←LEFT(P1)，AVAIL←P1，P1←LEFT(P)。转回到S4。 ∎

对这个算法的程序设计，留给读者作为一个非常有益的习题（见习题15）。这里值得指出，由于每个节点BASEROW[i]，BASECOL[j]的多数场都是不相干的，因此对每一个节点，仅须分配一个字的内存（见图14中带阴影的区域，并见2.2.5小节的程序）。甚至，为了再节约存储空间，值−PTR[j]即可作为ROW(LOC(BASECOL[j]))来存储。算法S的运行时间，非常粗略地，同受到主要操作影响的矩阵元素的个数成正比。

习题

1. 〔17〕 如果A是(1)的矩阵，而且如果这个数组的每个节点都占两个字，试给出对于LOC(A[J,K])的一个公式，假定这些节点是以下标的字典顺序连续地予以存储的。

▶2. 〔21〕 公式(5)已从对于$1\leqslant r\leqslant k$的假定$0\leqslant l_r\leqslant d_r$之下推导出来；试给出一个应用到$l_r\leqslant i_r\leqslant u_r$的一般公式，这里$l_r$和$u_r$是对于维数的任何下界和上界。

3. 〔21〕 正文中考虑了对于$0\leqslant k\leqslant j\leqslant n$的下三角形矩阵A[$j$,$k$]。对于下标从1开始而不是从0开始的情况，即是对于$1\leqslant k\leqslant j\leqslant n$的情况，这样的矩阵的讨论如何能容易地加以修改？

4. 〔22〕 说明：如果我们以下标的字典顺序来存储对于$0\leqslant j\leqslant k\leqslant n$的上三角形矩阵A[$j$,$k$]，则分配满足等式(7)的条件。试给出在这种情况下对于LOC(A[I,K])的公式。

5. 〔20〕 利用习题2.2.2-3的“间接寻址”特性，说明有可能以一条MIX指令来把A[J,K]的值送入寄存器A，甚至当A是如(8)中那样的三角形矩阵时亦然（假定J和K的值都在变址寄存器中）。

▶6. 〔*M*24〕 考虑“四面体数组”A[i,j,k]，B[i,j,k]，这里在A中$0\leqslant k\leqslant j\leqslant i\leqslant n$，而在B中$0\leqslant i\leqslant j\leqslant k\leqslant n$。假设这两个数组都以下标的字典顺序存入连续的内存单元；试说明，对于某些函数f_1，f_2，f_3，有LOC(A[I,J,K])$=a_0+f_1$(I)$+f_2$(J)$+f_3$(K)。能否以类似的方式来表达LOC(B[I,J,K])？

7. 〔*M*23〕 给出一个用于对k维的四面体数组A[i_1, i_2, …, i_k]分配存储的一般公式，

其中 $0 \leqslant i_k \leqslant \cdots \leqslant i_2 \leqslant i_1 \leqslant n$。

8.〔33〕（彼·韦格纳(P. Wegner)）假设A[I, J, K], B[I, J, K], C[I, J, K], D[I, J, K], E[I, J, K], 和F[I, J, K], 对于 $0 \leqslant K \leqslant J \leqslant I \leqslant n$，是待存入内存的六个四面体数组。是否有一个简洁的方式，类似于二维情况下的方式(9)，来实现这种存储？

9.〔22〕假设已经建立了一个表格，它与图13中所表示的相类似，但要大得多，它使得所有的链接都以如那里所示的同样的方向进行（即是，对于所有的节点和链接，LINK(X)<X)。试设计用于寻找出所有年龄为21岁到23岁的蓝眼睛金发女孩的地址的一个算法——要以这样一种方式来贯穿各种链接场，即当这个算法完成时，至多对女性，21岁，22岁，23岁，兰眼睛，和金发这些表的每一个扫描一次。

10.〔26〕你能想出编制人口表的更好的方式吗？要使得如象上题所述的检索将更为有效（对此题的答案不仅仅是“能”或“不能”）。

11.〔11〕假设我们有一个 200×200 的矩阵，其中每行至多有四个非0的元素。如果除了对表头将使用一个字外，每个节点我们都使用三个字，则为了如象图14所示那样来表示这个矩阵，需要有多少存储？

▶12.〔20〕借助于（12）中所用的记号 a，b，c，d，来表达在步骤S7开始处的VAL(Q0)，VAL(P0)，和VAL(P1)。

▶13.〔22〕在图14中，为什么用的是循环表，而不是直接的线性表？算法A能否重新改写，使得它不利用循环链接？

14.〔22〕算法S实际上节省了在一个稀疏矩阵中进行主要操作的时间，因为它避免了去考虑其中主要行有一个0元素的那些列。试说明在一个顺序地存储的大型稀疏矩阵中，借助于一个辅助的表格 LINK[j]，$1 \leqslant j \leqslant n$，也能够如此地节省运行的时间。

▶15.〔29〕写出算法S的一个MIXAL程序。假定VAL场是一个浮点数，并假定MIX的浮点算术操作符FADD，FSUB，FMUL，及FDIV可用于对这个场的运算。为简便起见，假定当所加得的或所减得的操作数的大多数有效位消失时，则FADD和FSUB即给出答案0，从而在步骤S7中可安全地使用对于“VAL(P1)= 0”的测试。浮点操作仅使用寄存器rA，而不使用rX。

16.〔25〕设计一个复写一稀疏矩阵的算法（换言之，这个算法是在内存中产生一个矩阵的两个不同的表示，它们都有图14的形式，并始时仅只给出一个这样的表示）。

17.〔26〕试设计一个对两个稀疏矩阵进行乘法的算法；给定矩阵 A 和 B，形成一个新的矩阵 C，其中 $C[i, j] = \Sigma_k A[i, k] B[k, j]$。两个输入矩阵和这个输出矩阵，都应该表示成如图14所示的那样。

18.〔22〕下列算法把一个矩阵替换成此矩阵之逆。为此，假定矩阵的元素是 $A[i, j]$，对于 $1 \leqslant i, j \leqslant n$，并使用“高斯—约当归约”：

a）对于 $k = 1, 2, \cdots, n$ 来做以下的工作：检索行 k 中所有的还未被用作过主要列的列，以求出具有最大绝对值的元素；置 $C[k]$ 等于其中找到这个元素的那个列的列数，并以这个元素作为主元素来实施一主要步骤（如果所有这样的元素全为0，则矩阵是奇异的，而且没有逆）。

b）置换诸行和诸列，使得原来是行 k 的元素变成行 $C[k]$，原来是列 $C[k]$ 的变成列 k。

本题的问题是，请使用上述算法，用手算来求矩阵

$$\begin{pmatrix} 1 & 2 & 3 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

的逆。

19.〔31〕 修改习题 18 中所述的“高斯-约当归约”算法，使得它得到：以图 14 形式表示的一个稀疏矩阵的逆。要特别地注意有效地进行步骤（b）的行和列的置换操作。

20.〔20〕 一个三对角型矩阵，其元素 a_{ij} 除了对于 $1 \leqslant i, j \leqslant n$，当 $|i-j| \leqslant 1$ 的以外，全都为 0。试证明，有一个形如

$$\mathrm{LOC}(A[I, J]) = a_0 + a_1 I + a_2 J, \qquad |I - J| \leqslant 1$$

的分配函数，它表示了在 $(3n-2)$ 个连续的单元中，一个三对角形矩阵的所有相关的元素。

21.〔20〕 提出对于 $n \times n$ 矩阵的一个存储分配函数，其中 n 是可变的。对于 $1 \leqslant I, J \leqslant n$ 的元素 $A[I, J]$，应该占有 n^2 个连续的单元，而不论 n 是怎样的值。

2.3 树

我们现在转过来研究树，它是在计算机算法中的最重要的非线性结构。一般地说，树结构指的是节点之间的“分枝”关系，很象是在自然界的树那样。

让我们正式地定义一个树：一个或多个节点的有限集合 T，使得

a）有一个特别地标出的称作该树之根 root（T）的节点；以及

b）剩下的节点（除根外）被分成 $m \geqslant 0$ 个不相交的集合 $T_1, \cdots, T_m$，而且这些集合的每一个又都是树。树 $T_1, \cdots, T_m$ 被称作这个根的子树。

刚才给出的定义是递归的，即是说，我们用树来定义树。当然，这里所含的循环是没有问题的，因为具有一个节点的树必然仅由根组成，而具有 $n > 1$ 个节点的树则借助于少于 n 个节点的树来定义。因此，具有两个节点，三个节点，以至最终，具有任意个节点的树之概念，是由这个已给的定义确定的。也有若干定义树的非递归方式（例如，见习题 10，12 和 14，以及 2.3.4 小节），但是递归的定义似乎最适当，因为递归是树结构的一个固有的特征。树的递归特性也呈现于自然界中，因为在一些幼树上的树芽终于要成长为子树，并有它们自己的树芽，等等。习题 3 说明了，基于如上所给出的这样的递归定义，如何通过对一个树的节点数使用归纳法，来对树的一些重要事实给出严格的证明。

从我们的定义直接得出，树的每一个节点，都是某个包含在这整个树中的子树的根。一个节点的子树的个数，称为该节点的次数。次数为 0 的节点，称为终端节点，或者有时称为“叶”。非终端节点通常称作分枝节点。一个节点相对于 T 的层数定义如下：根的层数为 0，而其它节点的层数，比起其相对于这个根的包含这些节点的子树 T_j 的层数，要大 1。

图 15 中说明了这些概念，这个图示出了一个具有七个节点的树。根是 A，而且它有

两个子树{B}和{C, D, E, F, G}。树{C, D, E, F, G}以节点C作为它的根。节点C相对于这整个的树是在层1上，而且它有三个子树形{D}，{E}，和{F, G}；因此C有次数3。图15中的终端节点是B，D，E，和G；F是唯一的次数为1的节点；G是唯一的层数为3的节点。

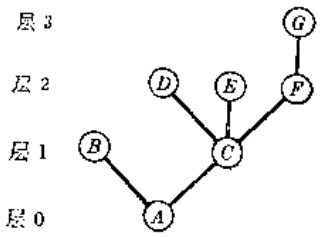


图15 一棵树

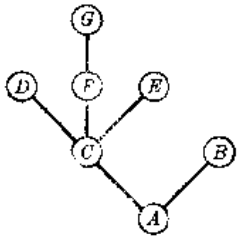


图16 另一棵树

如果在定义的（b）中，子树$T_1, \cdots, T_m$的相对次序是重要的，则我们就说这树是有序树。在一有序树中，当$m \geqslant 2$时，把T_2称作这个根的第二个子树，等等，就是有意义的了。有序树，有些作者也称之为“平面树形”，因为这关系到把这个树嵌入平面的方式。如果，当两个树的差别仅仅在于节点之子树的相对次序时，则我们便不认为它们二者是不同的，并且说这树是有向的，因为其时考虑的仅仅是这些节点的相对方向，而不是它们的次序。正是计算机表示的本性，定义了对于任何树的一种隐含的次序，所以在大多数情况下，有序树是我们最感兴趣的。因此我们将暗中假定，我们讨论的所有树都是有序的，除非另有明确的相反的说明。因此，图15与图16的树一般地将被认为是不同的，虽然作为有向树来说它们将是相同的。

一个森林是0个或多个不相交的树的集合（通常是一个有序的集合）。树的定义的部分（b），换一种方式来陈述，就可以说成：除了根之外，树的节点形成一个森林（某些作者使用“n重有根的树”来表示一个有n棵树的森林）。

在抽象的森林和树之间的差别很小；如果我们删去树的根，则我们就有一个森林，而且，反过来，如果我们仅仅加一个节点到任何森林，则我们就得到一棵树。因此，在非正式地讨论树结构时，树和森林这两个词几乎可互换地使用。

画树的图式可以有许多方式。除了图15的图式之外，取决于根放在什么地方，图17中示出了三种主要的选择。关于如何在图式中画出一个树的考究，并不是多余的，闹着玩

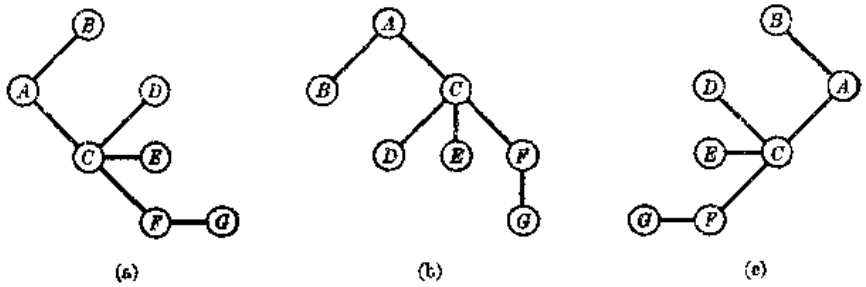


图17 我们将如何来画一棵树？

的，因为有许多时机，其时我们将希望来说一个节点是在另一个节点“之上”或“高于”另一个节点，或者来谈论最右的元素，等等。为处理树形结构的某些算法，已变成为叫做

“自顶向下”的方法，以相对于“由底向上”方法而言。除非我们坚持一个一致的画树的约定，否则这种术语将引起混乱。

图 15 的形式可能显得简单并备受欢迎，因为这反映了自然界中的树如何生长；也并没有任何强迫的原因，要接受任何其它三种形式，我们同样也可以沿袭自然的由来已久的传统。正是牢记着这一点，作者在准备这套书时，一贯地遵守了根在底下的约定。但经试用了两年之后，发现这乃是一种错误：对计算机文献，以及计算机科学家关于各种各样算法的许多非正式讨论的考察，都表明，在所调查的情况中，有 80% 以上的树是以根在顶上画出的。有一种压倒的趋势，使得手画的图是朝下发展而不是朝上的（而且，考虑到我们书写的方式，这也是易于理解的）；甚至于“子树”一词（“子”字暗含“下方的”意义，而不说成是“上方的”树）也趋向于暗示着一个向下的关系。鉴于这些考虑，我们就结论，图 15 是上下颠倒的。今后我们几乎将总是如图 17（b）所示的那样来画树，使得根在顶上而叶在底下。对应于这种定向，我们也许将把根节点叫做树的顶点。

为了讨论树，就必须有完善的描述性的术语。代替有些含混的“上面”和“下面”的提法，我们发现：取自家族树的家谱的词汇，非常适宜于这一目标。图 19 示出了两种普通类型的家族树。这两种类型显然是十分不同的：“家系”说明了一个指定的个人的祖宗，而“直系图”则说明了后裔。

如果出现了“交叉杂交”，则家族树就不是一棵真正的树，因为一棵树（如同我们所已定义的那样）的不同分枝决不能结合在一起。为了补偿这种脱节，注意 Victoria 和 Albert 在图 19（a）的第六代中出现了两次，而 Christian IX 实际上既出现在第五代又出现在第六代。如果家族树的每个节点表示的都不是一个人，而是“一个人在他作为某某人的双亲所扮角色”，那么家族树就是一棵真正的树。

对于树结构的标准术语，取自于家族树的第二种形式，即是直系图：每一个根是它的子树的那些根的父亲，而后面这些根互称为兄弟，并且它们都是它们父亲的儿子。整个树的根没有父亲。例如，在图 18 中，C 有三个儿子，即 D，E，和 F；E 是 G 的父亲；B 与 C 是兄弟。推广这个术语显然是可能的（例如，B 是 F 的一个伯父；A 是 G 的曾祖父；H 和 F 是第一代堂兄弟）。某些作者使用了女性的称呼“母亲，女儿，姐妹”来替代“父亲，儿子，兄弟”；但由于某种原因，男性的词似乎显得更专业化些。另外一些作者，希望促进男女平等，使用了中性的词“父母，子女，兄弟姐妹”来代替。在任何一种情况下，我们使用名词祖宗和后裔，来表示一个可以跨过这树形的若干层的关系：在图 18 中，C 的后裔是 D，E，F，和 G；G 的祖宗是 A，C，和 E。

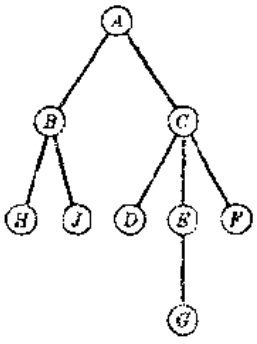


图18 一个树的树形式

图 19（a）是一棵二叉树的例子，它是树形结构的一个重要类型。毫无疑问地，读者已经见到过二叉树的其它例子，例如联系到网球比赛，等等。在一棵二叉树中，每一节点至多有两棵子树，而且在只有一棵子树出现时，我们须要明辨是左还是右子树。更正式地说，让我们定义二叉树为节点的有限集合，这有限集合或者是空集，或者由一个根及两个不相交的称作这个根的左和右子树的二叉树组成。

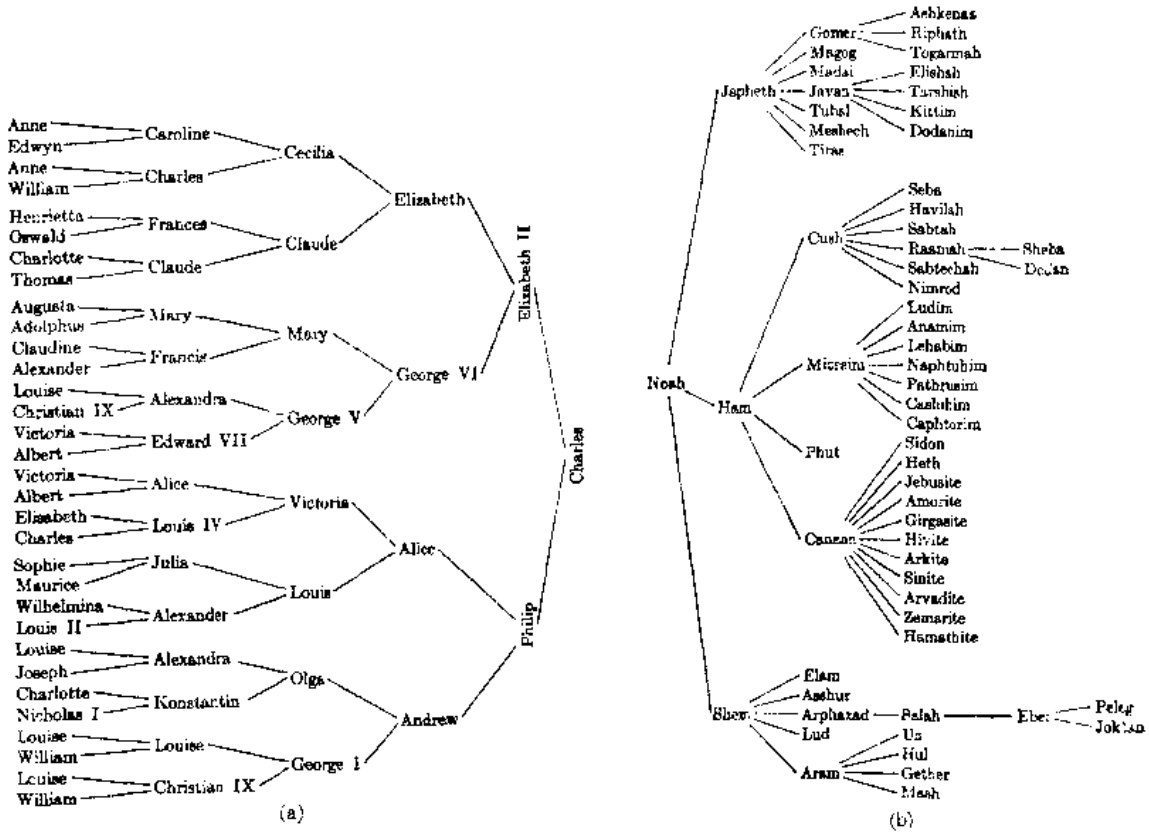
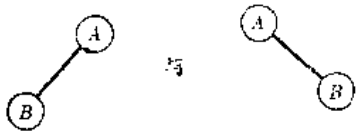


图19 家族树：（a）家系；（b）百系图。（参看：Burkés Peerage (1959), Almanach de Gotha(1871), Genealogisches Handbuch des Adels, Fürstliche Häüser, 1；Genesis 10：1-25.）

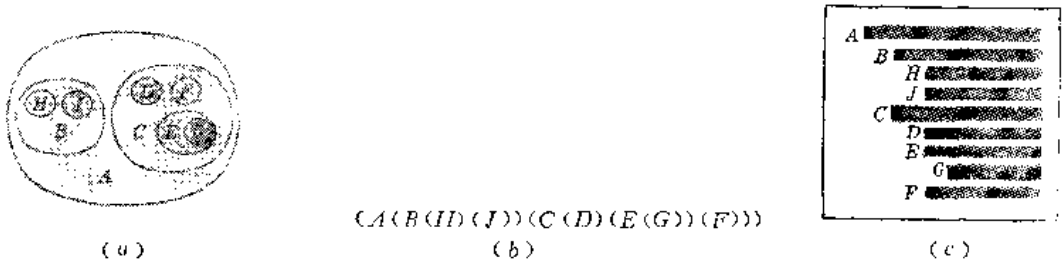
二叉树的这一递归定义，应当仔细地加以研究。注意二叉树并不是树的特殊情形；它完全是另一个概念（尽管我们将看到这两个概念之间的许多关系）。例如，二叉树



（1）

是不同的（在一种情况下，根有空的左子树；而在另一种情况下，根有非空的左子树），尽管作为树它们将是等同的。因此我们将总是小心地来使用“二叉”这一词来区别二叉树与通常的树。某些作者以稍微不同的方式来定义二叉树（见习题 20）。

还可以用若干与实际的树很不相似的方式图式地表示树结构。图 20 示出了三种图式，它们都反映了图 18 的结构；图 20（a）实质上把图 18 表示成一个有向树；这个图式乃是嵌套集合一般概念的一个特殊情况——嵌套集合即是一些集合的集体，其中任何一对集合或者不相交，或者一个包含另一个（见习题 10），该图的部分（b）在一行里示出了嵌套



（a） （b） （c）

图20 显示树结构的其它方式：（a）嵌套集合；（b）嵌套的圆括弧；（c）凹入。

的集合，就好象部分（a）是在一个平面上示出它们那样；在部分（b）中，还指出了树的次序。部分（b）也可以认为是一个含有“嵌套的圆括弧”的代数公式之表达。部分（c）显示的，仍然是表示树结构的又一个普通的方式，并且用的是凹入。不同表示方法的种数，本身就是充足的证据，说明树结构在日常生活中以及在计算机程序设计中的重要性。任何分等级的分类方案，都导致一个树结构。

象图20（c）这样一个凹缺得犬牙交错的表，与本书内容的一览表或目录，其间的相似性是值得注意的。本书本身就有一个树形结构：图21中示出了第2章的树形结构。这里我们注意到一个有意义的思想：在本书中用来对章节进行编号的方法，乃是确定树形结构的另一个方法。这样一个方法，通常叫做对于树的“杜威（Dewey）记数法”，这类似于在图书馆里所用的这一名称的类似的分类方案。对于图18的树的杜威记数法是

1A；1.1B；1.1.1H；1.1.2J；1.2C；1.2.1D；1.2.2E；1.2.2.1G；1.2.3F。

杜威记数法可应用到任意的森林：森林中的第k个树的根给以号码k；而如果α是次数为m的任何节点的号码，则它的儿子就编号成$\alpha.1$，$\alpha.2$，…，$\alpha.m$。杜威记数法满足许多简单的数学性质，而且在树的分析中，它是一个有用的工具。对一个任意树的节点给出自然的顺序的编号，类似于本书内的章节的编序，就是这样的一个例子。

杜威记数法，与我们已经广泛使用的下标变量的记号之间，有着密切的关系。如果F是一些树的森林，则我们可以命$F[1]$表示第一棵树，$F[1][2]=F[1,2]$表示这第一棵树之根的第2棵子树，$F[1,2,1]$又表示后者的第一棵子树，等等。在杜威记数法中，节点$a.b.c.d$就是root($F[a,b,c,d]$)。这个记法，乃是下标记号的一个推广，即是，每个下标所允许的范围，

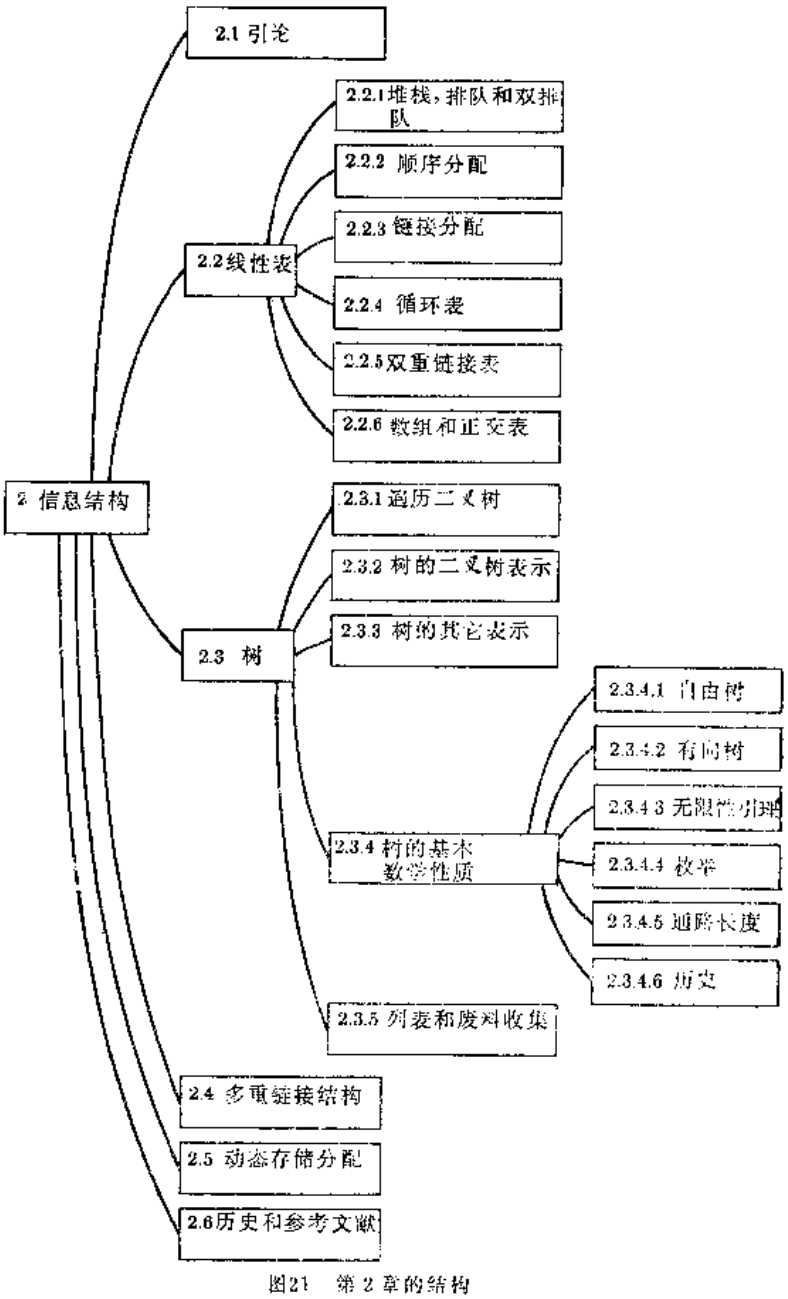
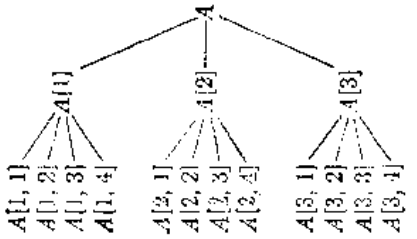


图21 第2章的结构

现在依赖于在前一个下标位置上的值。

特别地，我们看到，任何矩形数组可被想象作树结构的一个特殊情况。例如，这里是一个 3 × 4 矩阵的两种表示：

$$\begin{pmatrix} A[1,1] & A[1,2] & A[1,3] & A[1,4] \\ A[2,1] & A[2,2] & A[2,3] & A[2,4] \\ A[3,1] & A[3,2] & A[3,3] & A[3,4] \end{pmatrix}$$



然而，重要的是要注意，这个树结构并不精确地反映所有的矩阵结构；在树中行的关系明显地显示出来了，但是列的关系却不然。

代数公式向我们提供了另一个树结构的例子。图 22 示出了一个对应于算术表达式

$$a - b\,(c/d + e/f) \qquad (2)$$

的树。如同我们以后将要看到的（特别是在第 10 章和第 12 章），公式和树之间的联系，在应用上是非常重要的。

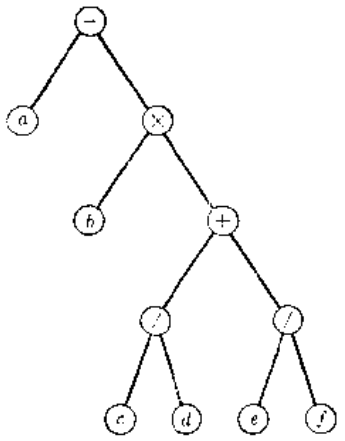


图22　表示公式（2）的树

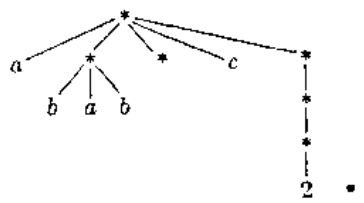
除了树，森林和二叉树之外，还有第四种类型的结构，通常叫做“列表”结构，它与前两种结构都紧密相关。我们现在所用的“列表”一词是在一个非常技术性的意义下的，为此才在“表”这一词前面再冠上一个“列”字，以资区别。我们把列表（递归地）定义作：0 个或多个原子或列表的一个有限的序列。这里“原子”是一个不定义的概念，指的是所希望的任何对象的整体中的元素，只要能从一个列表区别出是一个原子就行。借助于关于逗号和圆括号的自明的记号约定，我们就能区别原子与列表，而且我们还能方便地显示出一个列表内的次序。例如，考虑

$$L = (a, (b, a, b), (\ ), c, (((2)))) \qquad (3)$$

它是具有五个元素的列表：头一个是原子 a，然后是列表 (b, a, b)，然后是空列表“( )”，然后是原子 c，最后是列表$(((2)))$。最后这个列表由 $((2))$ 组成，而这个列表又由列表 (2) 组成，它又是由原子 2 组成。

以下的树形结构对应于 L：

$L =$



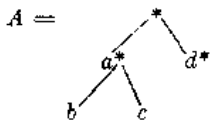
(4)

在这个图式中，星号表示一个列表的定义和出现，以相对于一个原子的出现。下标记号也可应用于列表，就象它可应用于树那样。例如，$L[2] = (b, a, b)$，$L[2, 2] = a$。

在对于（4）中的那些列表的节点中，除了它们都是列表这一事实外，并没有引进什么数据。也可以以信息来标示这些列表的节点，如同我们对于树和其它结构所已做过的那样；例如，

$$A=(\ a:(\ b,\ c\ ),\ \ d:(\ \ ))$$

将对应于一棵树，我们可以把它画出如下：



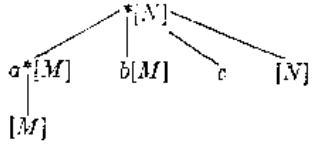
列表与树之间的重大差别在于，列表可以重叠（即是，子列表不必是不相交的）.而且它们甚至可以是递归的（可以包含它们自身）。列表

$$M=(M) \tag{5}$$

既不对应于树形结构，列表

$$N=(\ a:M,\ \ b:M,\ c,\ N) \tag{6}$$

亦不对应于树形结构（在这些例子中，大写字母指的是列表，小写字母指的是标号和原子）。用星号来表示定义了一个列表的位置，我们就可以把（5）和（6）画出图示如下：



(7)

实际上，列表并不象上边的例子所指出的那么复杂；实质上，它们是如象我们在2.2节中所已讨论过的线性表的一种相当简单的推广，即是附加上条件：线性表的元素又可以是指向其它线性表的链接变量（而且可能指向本身）。

小结：密切相关的四种类型的信息结构（树，森林，二叉树，列表），是从许多来源产生的，而且因此它们在计算机算法中是重要的。我们已经看到了画出这些结构之图式的各种方法，而且我们已经考虑了在谈论它们时有用的某些记号和术语。以下数节，将更为详尽地来展开这些思想。

习题

1.〔18〕 对于三个节点A，B，和C，有多少不同的树？

2.〔20〕 对于三个节点A，B，和C，有多少不同的有向树？

▶3.〔$M20$〕 从定义出发，严格地证明，对于一个树中的每个节点X，都有唯一的“通向根的通路”，即是，唯一的$k\geqslant 1$个节点的序列X_1，X_2，…，X_k，使得X_1是这树的根，$X_k=X$，而且X_j是X_{j+1}的父亲，对于$1\leqslant j<k$（这个证明，对于树形结构的几乎所有基本事实之证明而言，都将是典型的）。〔提示：对树中的节点数使用归纳法〕。

4.〔01〕 是真还是假：在通常的树图式中（根在顶上），如果节点X比节点Y有更高的层数，则在图式中节点X比节点Y出现得更低。

5.〔02〕 如果节点A有三个兄弟，而且B是A的父亲，试问B的次数是什么？

▶6.〔21〕 通过模仿家族树，如果$m>0$和$n\geqslant 0$，试把语句“X是Y的相差了n代的第m代堂兄弟”定义成一种在一棵树的节点X与Y之间的有意义的关系（关于家族树中这些术语的意义，请查词典）。

7.〔23〕 把上题给出的定义推广到所有$m\geqslant -1$和所有整数$n\geqslant -(m+1)$，使得对于一棵树的任何两个节点X和Y，都存在唯一的m和n，使得X是Y的相差了n代的第m

代的堂兄弟。

▶8.〔03〕 什么样的二叉树不是树?

9.〔00〕 在(1)中的两棵二叉树中,哪一个节点是根(*B*还是*A*)?

10.〔*M*20〕 非空集合的一个集体说是嵌套的,如果给出任何一对集合*X*和*Y*,则或者$X\subseteq Y$,或者$X\supseteq Y$,或者*X*与*Y*不相交(换言之,$X\cap Y$或者是*X*,或者是*Y*,或者是ϕ)。图20(*a*)表示,任何树都对应于一个嵌套集合的集体;反过来,每一个这样的嵌套集合的集体,是否都对应于一个树呢?

▶11.〔*HM*32〕 通过如题10中那样来考虑嵌套集合的集体,把树的定义推广到无限的树。对于无限树的每个节点,能否定义层数,次数,父亲和儿子这些概念?试给出对应于一个树的实数的嵌套集合的一些例子,使其中

a)每个节点有不可数的次数并且存在有无穷多个层数;

b)存在着具有不可数层数的节点;

c)每个节点有至少为2的次数并且存在有不可数多个层数。

12.〔*M*23〕 在什么条件下,一个部分序集(参照2.2.3小节)对应于一个非有序的树或森林?

13.〔10〕 假设节点*X*在杜威记数法中的编号是$a_1\cdot a_2\cdot\cdots\cdot a_k$,试问在从*X*到根的通路中,节点的个数是多少(见习题3)?

14.〔*M*22〕 设*S*是有形式"$1\cdot a_1\cdot\cdots\cdot a_k$"的元素的任何非空集合,其中$k\geqslant 0$,而且$a_1,\cdots,a_k$是正整数。试证明,当*S*是有限的而且满足下列条件:"如果$\alpha\cdot m$在集合中,那么,假如$m>1$,则$\alpha\cdot(m-1)$也在集合中,或者假如$m=1$,则α亦然"时,则*S*就确定一棵树(这个条件,显然在对于一个树形的杜威记数法中是满足的,所以,作为这个习题的一个推论,它是表征树结构的另一个方式)。

▶15.〔20〕 试设计出对应于树的杜威记数法的二叉树的记法。

16.〔20〕 画出类似于图22的对应于下列算术表达式的树:

a)"$2(a-b/c)$";

b)"$a+b+5c$"。

17.〔01〕 如果*T*是图18的树,则那个节点是root(T〔2,2〕)?

18.〔08〕 在列表(3)中,L〔5,1,1〕是什么?L〔3,1〕是什么?

19.〔15〕 对于列表$L=(a,(L))$,试画出一个类似于(7)的列表图式。在这个列表中L〔2〕是什么?L〔2,1,1〕是什么?

▶20.〔*M*21〕 定义一"*b*树"为其中每个节点恰有0个或两个儿子的树(形式地,一"*b*树"由一个称作它的根的节点,加上0个或两个不相交的*b*树组成)。试证明每棵*b*树有奇数个节点;并给出一个具有*n*个节点的二叉树与具有$2n+1$个节点的(有序)*b*树之间的一一对应关系。

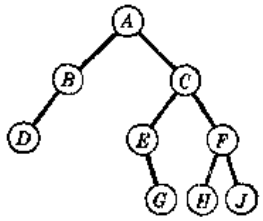
21.〔*M*22〕 如果一棵树有n_1个次数1的节点,n_2个次数2的节点,…,n_m个次数*m*的节点,则它有多少个终端节点?

22.〔45〕 研制一个在阴极射线管上用图形来显示树结构的计算机系统,连同用于对这些结构进行联机处理的设备。

2.3.1 遍历二叉树

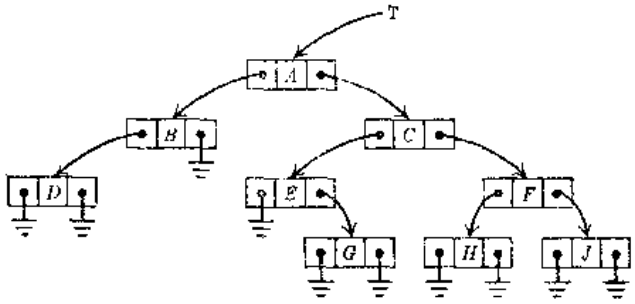
在对树作进一步的研究之前，对二叉树的性质获得很好了解，乃是重要的，因为一般的树，在一台计算机内通常都借助于某个等价的二叉树来表示。

我们已经把二叉树定义作这样一个有限的节点集合，它或者是空集，或者由一个根与两棵二叉树合起来组成。这个定义提示了一个自然的在一台计算机内表示二叉树的方式：在每一个节点之内有两个链接，即 LLINK 和 RLINK，以及作为"指向树的指针"的一个链接变量 T。如果树是空的，则 $T = \Lambda$；否则 T 是树的根节点的地址，而且 LLINK(T) 和 RLINK(T) 分别是指向这根之左边和右边子树的指针。这些规则递归地定义了任何二叉树形的内存表示；例如，



(1)

由



(2)

来表示。

这个简单而自然的内存表示，乃是二叉树结构具有特殊重要性的原因。除了一般树都可方便地表示作二叉树这一事实之外，应用中出现的许多树本身都固有地是二叉的，所以二叉树理应是令人感兴趣的。

有着许多用以处理树结构的算法，而且在这些算法中，反复出现的一个思想是遍历或"走遍"一个树。这是一个系统地检查树的节点，使得每个节点恰好被访问一次的方法。对树之完整的遍历，给了我们以节点的一种线性排列，而且如果我们能谈及：在这样一个序列中，给定的一个节点之前边或后边的下一个节点，那么，就使许多算法变得好办了。

为遍历一棵二叉树，可使用三个主要的方式：我们可以以先根次序，中根次序以及后根次序，来访问这些节点。这三个方法，是递归地来加以定义的。当二叉树为空时，什么都不做就"遍历"了，否则遍历分三步进行：

先根次序的遍历

访问根

遍历左子树

遍历右子树

中根次序的遍历

遍历左子树

访问根

遍历右子树

后根次序的遍历

遍历左子树

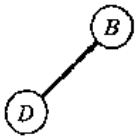
遍历右子树

访问根

如果应用这些定义到二叉树（2），则我们发现，在先根次序下的节点是

$$A\ B\ D\ C\ E\ G\ F\ H\ J \qquad (3)$$

在先根次序下，头一个来到的是根A，然后按先根次序来到左子树，



（最后按先根次序来到右子树）。对于中根次序，我们是在对每棵子树的节点进行访问之间，来访问根的；实际上就象把这些节点“投影”到一条水平线上，因而这给出了序列

$$D\ B\ A\ E\ G\ C\ H\ F\ J \qquad (4)$$

类似地，对于这棵二叉树的节点的后根次序，是

$$D\ B\ G\ E\ H\ J\ F\ C\ A$$

我们将看到，把一个二叉树的节点排列成一个序列的这三种方式，是极为重要的，因为它们最终地联系到处理树的大多数计算机方法。当然，先根次序，中根次序，以及后根次序的名称，是从根相对于它的子树的位置引出来的。在二叉树的许多应用中，在左子树形与右子树形的意义之间，有着更强的对称性，而且在这样的情况下，对称性次序就用作中根次序的同义语。显然，中根次序，它把根置于中间，实际上在左与右之间是对称的；如果关于一个垂直轴来反映树，则对称性次序即简单地被反向。

递归地叙述的定义，如象刚才对于三个基本次序所给出的那样，必须重新加工，为的是使它直接可应用于计算机的实现。为进行这项工作的一般方法，留待在第8章中讨论；象在下列算法中那样，我们通常都利用一个辅助的堆栈：

算法 T（在中根次序下遍历二叉树） 设T是一个指向二叉树的指针，而这二叉树有着如在（2）中那样的表示。这个算法，利用一个辅助堆栈A，在中根次序下访问二叉树的所有节点。

T1.〔初始化〕 置堆栈A为空，并置链接变量P←T。

T2.〔P＝A？〕 如果P＝A，则转到步骤T4。

T3.〔堆栈⇐P〕（现在P指向一个有待遍历的非空的二叉树）。置A⇐P，即是，把P的值压入堆栈A（见2.2.1小节）。然后置P←LLINK(P)并返回到步骤T2。

T4.〔P⇐堆栈〕 如果堆栈A为空，则本算法终止；否则置P⇐A。

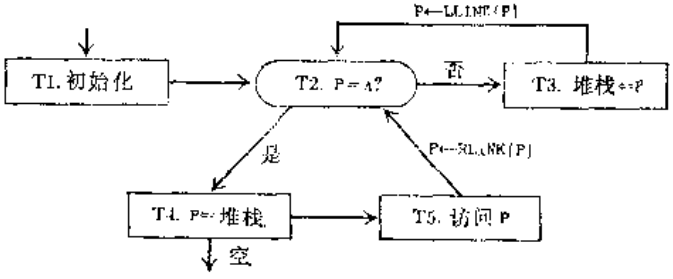


图23 算法T的框图

T5.〔访问P〕 “访问”NODE(P)。然后置P←RLINK(P)并返回到步骤T2。■

在这个算法的最后步骤里，“访问”一词指的是，当遍历树时，我们打算做的任何活动。算法T就象是关于这另一个活动的共行程序那样来运行：主程序激活这个共行程序，每当它需要把P从一个节点移到其在中根次序下的后继时。当然，由于这个共行程序仅仅在一处调用这主程序，所以它与一个子程序没有多大区别（见1.4.2小节）。算法T假定，外部

的活动既不从树删去 NODE(P)，也不删去它的任何祖宗。

如果读者现在利用树（2）作为一种检验的情况，而试图来运用算法T，则他将容易地悟出在这个过程背后的一些理由：当我们到达步骤T3时，我们要来遍历由指针P指出其根的二叉树。这里的思想是把P存到一个堆栈，而后遍历这左子树；当这个已经做完时，我们将到达步骤T4，而且我们将于该堆栈再次找到P的旧值，经在步骤T5中访问了这个根 NODE(P) 之后，余下的任务就是去遍历右子树。

鉴于算法T在我们以后将要看到的许多个其它算法当中，是典型的，因而，来考察在上一段中所作陈述的一个形式证明，乃是有教益的。现在让我们通过对 n 用归纳法，来证明算法T在中根次序下遍历 n 个节点的二叉树。如果我们能证明下述的稍微更一般的结果，则我们的目的即容易实现：

"以P作为一个指向 n 个节点的二叉树的指针，以及以堆栈A含有A[1]…A[m]，对某个 $m\geqslant 0$，从步骤T2起动，则步骤T2—T5将在中根次序下遍历所述的那个二叉树，而且然后将达到步骤T4，同时堆栈A恢复到它原有的值A[1]…A[m]。"

当 $n=0$ 时，由于步骤T2，这个命题显然是正确的。如果 $n>0$，则命 P_0 是进入步骤T2时P的值。由于 $P_0\neq\Lambda$，我们将实施步骤T3，而这意味着，堆栈A被变成为A[1]…A[m]P_0，且P被置为LLINK(P_0)。现在，左子树有少于 n 个节点，所以按归纳法我们将在中根次序下遍历这左子树，并且最终地将以A[1]…A[m]P_0 于堆栈中而到达步骤T4。步骤T4把堆栈恢复成为A[1]…A[m]，并置$P\leftarrow P_0$。步骤T5现在访问NODE(P_0)并置$P\leftarrow$RLINK(P_0)。现在右子树少于 n 个节点了，所以按归纳法我们将象所要求的那样，在中根次序下遍历这右子树，而且到达T4。根据中根次序的定义，树已在中根次序下被遍历。这就完成了证明。

不难构造一个在先根次序下遍历二叉树的算法，它几乎是相同的（见习题12）。实现在后根次序下的遍历，要稍微更困难些（见习题13），而且由于这个原因，就使得后根次序对于二叉树，不如其它两个那么重要。

对于节点在各种次序下的后继和前驱，定义一个新的记号，是方便的。如果P指向二叉树的一个节点，则命

$$\begin{aligned}
&P* = \text{在先根次序下 NODE(P) 的后继的地址}\\
&P\$ = \text{在中根次序下 NODE(P) 的后继的地址}\\
&P\# = \text{在后根次序下 NODE(P) 的后继的地址}\\
&*P = \text{在先根次序下 NODE(P) 的前驱的地址}\\
&\$P = \text{在中根次序下 NODE(P) 的前驱的地址}\\
&\#P = \text{在后根次序下 NODE(P) 的前驱的地址}
\end{aligned}\qquad (5)$$

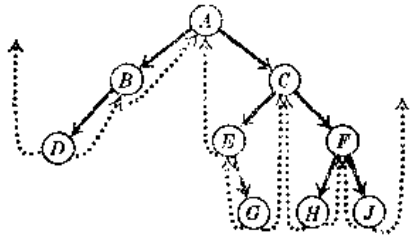
如果没有 NODE(P) 的这样的后继或前驱，则一般地就使用值 LOC(T)，其中T是一个指向所述的该树的指针。我们有 $*(P*)=(*P)*=P$，$\$(P\$)=(\$P)\$=P$，等等。作为这个记号的一个例子，命 INFO(P) 为树形（2）中 NODE(P) 中所示的字母；那么，如果P指向根，则我们有 INFO(P)=A，INFO(P*)=B，INFO(P$)=E，INFO($P)=B，INFO(#P)=C，以及 $P\#=*P=$LOC(T)。

此刻，读者也许会体验到关于P*，P$，等等之直观意义的一种不安全感。随着我们

进一步进行时，这些思想将逐渐变得明朗起来；这一小节末尾的习题 16 可能也有帮助。

对于（2）中给出的二叉树之内存表示，有一个重要的改变，它有些类似于在循环表与直线的单向表之间的差别。注意在树（2）中，竟有比指针还多的空链接，而且事实上，对于任何通过这个方法表示的二叉树，这都是如此（见习题14）。看来这是浪费内存空间的，而且有着更有效地利用内存的种种途径；例如，我们可以对于每个节点存入两个"标签"指示位，它们只以两位存储来告知：LLINK 或 RLINK，或两者，是否为空；那么，用于终端的链接之存储空间，又可用于其它目的。

艾·J·珀利斯（A. J. Perlis）和查·桑顿（C. Thornton）曾经提出一种巧妙地使用这额外的内存空间的方法，设计了所谓穿线的树表示。在这个方法中，终端的链接由连到树的其它部分的"穿线"所代替，以作为遍历树之辅助。等价于（2）的穿线树是



（6）

这里虚线表示"穿线"，它总是通向树的更高的节点。现在每个节点都有两个链接；某些节点，比如C，有两个连到左子树和右子树的通常的链接；其它节点，比如H，有两个穿线链接；而某些节点则有每种类型的一个链接。从D和J引出的特殊穿线将在后面说明；它们出现于"最左"和"最右"的节点。

在一个穿线的二叉树的内存表示中，有必要来明辨虚线的与实线的链接；这与上边提出的一样，是通过在每个节点的两个额外的一位场 LTAG 和 RTAG 来实现的。穿线表示，可以精确地定义如下：

非穿线的表示	穿线的表示
LLINK(P)=A	LTAG(P)="−", LLINK(P)=$P
LLINK(P)=Q≠A	LTAG(P)="+", LLINK(P)=Q
RLINK(P)=A	RTAG(P)="−", RLINK(P)=P$
RLINK(P)=Q	RTAG(P)="+", RLINK(P)=Q

按照这个定义，每个新的穿线的链接直接指到中根次序（对称次序）下该所述节点之前驱或后继。图 24 说明了，在任何二叉树中，穿线链接的一般定向。

在某些算法中，可以保证，任何子树的根，将总是出现于比子树的其它节点要低的内存单元。于是 LTAG(P) 将为"−"，当且仅当，LLINK(P)<P；从而 LTAG（且类似地 RTAG）就是多余的了。

穿线树的最大优点是，遍历算法变得更为简单。例如，下列算法，对于给定的 P，计算 P$：

算法 S（在一个穿线的二叉树中对称的（中根次序的）后继）。如果 P 指向一个穿线的二进树的一个节点，则这个算法置Q←P$。

S1. [RLINK(P)是一个穿线链接？] 置Q←RLINK(P)。如果 RTAG(P)="−"，则终止算法。

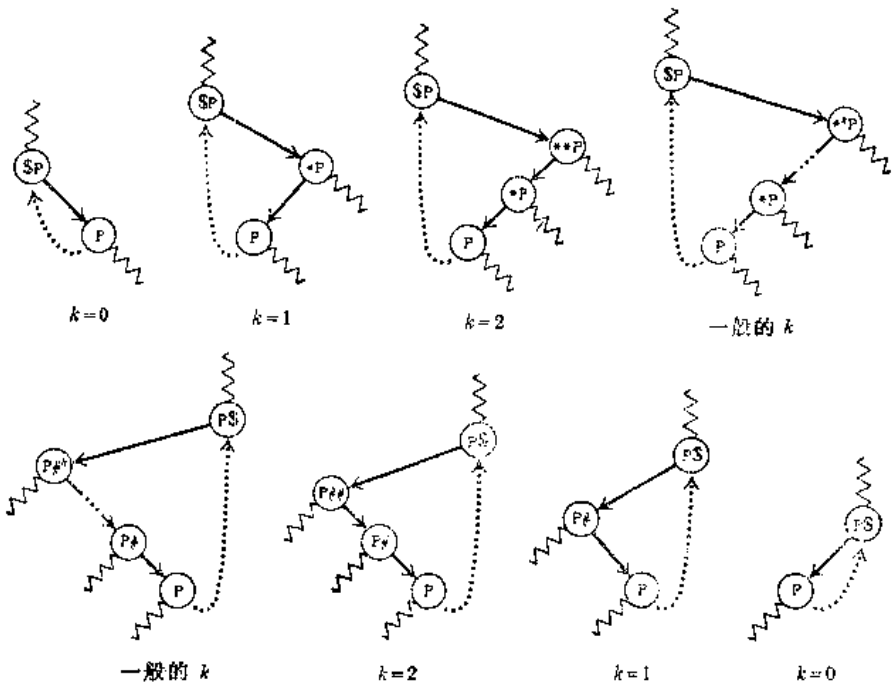


图24　在一棵二叉树中左和右穿线链接的一般定向。"∧∨"折线表示指向该树之其它部分的链接或穿线

S2.〔往左检索〕如果 LTAG(Q)="+"，则置Q←LLINK(Q)并重复这个步骤。否则算法终止。■

注意这里并不需要堆栈，就实现了在算法T中用堆栈所做的事情。事实上，如果仅仅给出了树中一个随机点P的地址，则原先的表示（2），并不可能用来有效地找出P$；因为在非穿线的表示中没有指向上方的链接，因而也就没有线索来提示：什么节点是在一个给定的节点之上方，除非我们保留我们如何达到这点的历史（而这实际上正是算法T中的堆栈）。

我们声称，算法S是"有效的"，尽管这个性质并不是立即明了的，因为步骤S2能被执行任意多次。鉴于步骤S2中的循环，是否也许是，如同算法T那样使用一个堆栈，终归还更快些？为了研究这个问题，我们将考虑当P是树中的一个"随机"点时，步骤S2必须被实施的平均次数；或者同样地，当重复地使用算法S来遍历整个树时，我们将确定实施步骤S2的总次数。

在进行这个分析的同时，来研究对于算法S和T两者的MIX程序是有启发的。以下的程序假定节点有两个字的形式。

LTAG	LLINK	INFO1
RTAG	RLINK	INFO2

在一个非穿线的树形中，LTAG和RTAG将总是"+"，而且终端的链接将以0表示之。缩写LLINKT和RLINKT将被用来分别代表组合的LTAG-LLINK和RTAG-RLINK场。

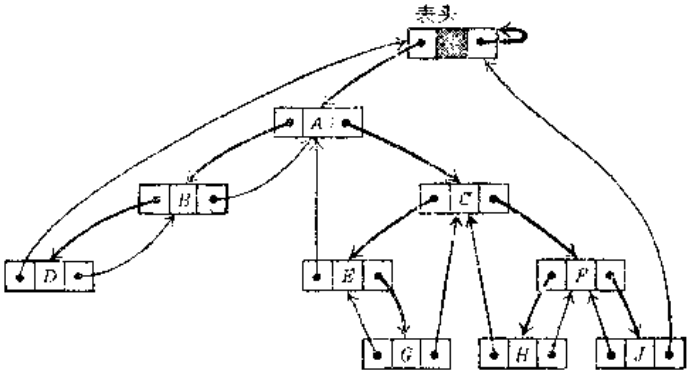
和通常一样，我们应当小心地来建立我们所有的算法，使得它们对于空树也能适当地进行工作；而且如果T是指向树的指针，则我们将希望LOC(T)*和LOC(T)$分别是在先根次序下和对称次序下的第一个节点。对于穿线的树，结果是，只要NODE(LOC(T))做成对于树的一个"表头"，且

LLINK(HEAD)=T，RLINK(HEAD)=HEAD，RTAG(HEAD)="+" (7)

则一切就都将如愿以偿了（HEAD 表示 LOC(T)，表头地址）。一个空的穿线树将满足条件

LLINK(HEAD)=HEAD，LTAG(HEAD)="−" (8)

这棵树通过把节点插入到表头的左边而成长起来（这些初始条件主要地取决于计算 P* 的算法，这个算法出现于习题 17 中）。按照这些约定，作为一个穿线的树，对于二叉树(1)的计算机表示是



(9)

有了这些准备工作，现在我们言归正传，准备来考虑对于算法 S 和 T 的 MIX 程序。下面两个程序，通过变址寄存器 5 指向当前所关注的节点，周期性地跳到单元 VISIT，在对称次序（即，中根次序）下遍历一棵二叉树。

程序 T 在算法 T 的这个实现中，堆栈是单元 A＋1，A＋2，…，A＋MAX；而且如果堆栈增长得太大，则出现 OVERFLOW。r6 是堆栈指针而 r5≡P。这个程序已经由算法 T 稍加重新安排（步骤 T2 出现了三次），使当直接地从 T3 进行到 T2 到 T4 时，不需要进行空堆栈的测试。

```
01    LLINK   EQU   1:2
02    RLINK   EQU   1:2
03    T1      LD5   HEAD(LLINK)      1   T1. 初始化。置P← T
04    T2B     J5Z   DONE             1   如果P = Λ，则停止
05            ENT6  0                1
06    T3      DEC6  MAX              n   T3. 堆栈← P
07            J6NN  OVERFLOW         n   堆栈已达到容量?
08            INC6  MAX + 1          n   若否，堆栈指针增量
09            ST5   A, 6             n   P 存入堆栈
10            LD5   0,5(LLINK)       n   P←LLINK(P)
11    T2A     J5NZ  T3               n   如果 P ≠Λ，则到 T3
12    T4      LD5   A,6              n   T4. P ⇐堆栈
13            DEC6  1                n   堆栈指针减量
14    T5      JMP   VISIT            n   T5. 访问 P
15            LD5   1,5(RLINK)       n   P←RLINK(P)
16    T2      J5NZ  T3               n   T2. P = Λ?
17            J6NZ  T4               a   测试堆栈空否
18    DONE    ...
```

程序 S 增加了算法 S 的初始化条件和结束条件，让这个程序可与程序 T 相比较。

```
01    LLINKT  EQU   0:2
02    RLINKT  EQU   0:2
03    S0      ENT6  HEAD          1      S0. 初始化。置 Q←HEAD
04            JMP   S2            1
05    S3      JMP   VISIT         n      S3. 访问 P
06    S1      LD5N  1,5(RLINKT)   n      S1. RLINK(P)是一个穿线链接?
07            J5NN  1F            n      如果 RTAG(P)="-", 则跳
08            ENN6  0,5           n-a    否则置 Q←RLINK(P)
09    S2      ENT5  0,6           n+1    置 P←Q
10            LD6   0,5(LLINKT)   n+1    Q←LLINKT(P)
11            J6P   *-2           n+1    如果 LTAG(P)="+", 则重复
12    1H      ENT6  -HEAD,5       n+1
13            J6NZ  S3            n+1    访问, 除非 P=HEAD。
```

在上述的代码中，同时示出了对运行时间的分析。用克希霍夫定律和下列事实，即容易确定这些数量：

i）在程序 T 中，插入堆栈的个数必须等于删去的个数；

ii）在程序 S 中，每个节点的 LLINK 和 RLINK 恰恰被检查一次；

iii）“访问”的数目，是树中节点的个数。

这个分析告诉我们，程序 T 花了 $15n+a+4$ 个时间单位，而程序 S 花了 $11n-a-8$ 个时间单位，其中 n 是树中节点的个数，而 a 是终端的右链接（无右子树的节点）的个数。假定 $n\neq 0$，则数量 a 可以小于 1，而且它可能大到 n；而且如果左边与右边是对称的，那么，作为习题 14 中证明的事实的推论，a 的平均值是 $(n+1)/2$。

在这个分析的基础上，我们所得到的主要结论是：

i）平均每执行一次算法 S，其中仅仅实施一次这个算法 S 的步骤 S2，如果 P 是树的一个随机节点的话。

ii）对于穿线的树，遍历是较为快的，因为它不需要堆栈操作。

iii）算法 T 需要比算法 S 要多的内存空间，因为需要辅助的堆栈。在程序 T 中，我们是在连续的内存单元来存放堆栈，因而，对于它的大小需要加上任意的界限。如果超过了这个界限，那将是非常麻烦的，所以必须把它设置得相当之大（见习题10）；于是程序 T 需要的内存比程序 S 要多得多。一个复杂的计算机应用，将经常地独立地一次遍历若干树，而且在程序 T 的情况下，对于每个树都将需要一个独立的堆栈。这提示我们，程序 T 可以对它的堆栈使用链接的分配（见习题20）；这样，它的执行时间就变成为 $30n+a-4$ 个单位，大约比以前时间慢两倍；然而，当加进其它共行程序的执行时间时，这可能不是非常重要的。还有另一个选择，就是如象习题 21 中所讨论的那样，仍然是以非常巧妙的方式在树自身之内来存放堆栈的链接。

iv）当然，算法 S 比算法 T 更为一般，因为当我们不需要遍历整个的二叉树时，它允许我们从 P 进行到 P$。

所以，一个穿线的二叉树，就遍历说来，断然地优于非穿线的二叉树。这些优点，在

某些应用中，由于在一个穿线的树中为插入和删去节点，需要略微增加时间，就被抵销了。有时，使用非穿线的表示，也有可能通过“共享”共同的子树来节省内存空间，然而穿线的树则要求坚持一个无重叠的子树的严格的树结构。

穿线的链接也能用来计算P*，$P和#P，并与算法S的效率相当。函数*P和P#稍微难以计算，就象它们是作为非穿线的树表示一样。忠告读者来做习题17。

如果首先就难以建立穿线的链接，则穿线的树之大多数用途，都将付诸东流。使这个思想真正付诸实施的事实是，穿线树几乎与通常的树同样容易地长成。我们有下列的算法：

算法I（插入到一个穿线的二叉树中） 这个算法加入一个节点NODE(Q)。如果NODE(P)的右子树是空的（即，如果RTAG="−"），则作为NODE(P)的右子树，否则它把NODE(Q)插入到NODE(P)与NODE(RLINK(P))之间。假定在其中进行插入的二叉树，是有待穿线成像（9）那样的；作为一个修正，见习题23。

I1.〔调整标签和链接〕 置RLINK(Q)←RLINK(P)，RTAG(Q)←RTAG(P)，RLINK(P)←Q，RTAG(P)←"+"，LLINK(Q)←P，LTAG(Q)←"−"。

I2.〔RLINK(P)曾是一穿线链接?〕 如果RTAG(Q)="+"，则置LLINK(Q$)←Q（这里Q$由算法S确定，即便LLINK(Q$)现在是指向NODE(P)而不是NODE(Q)，算法S仍可适当地工作。这个步骤，仅当插入到一个穿线树的中间而不是仅只插入一个“新的叶”时，才是必要的）。■

通过倒换左边与右边的作用（特别是，通过在步骤I2中以$Q来代替Q$），我们便得到一个以类似的方式插入到左边的算法。

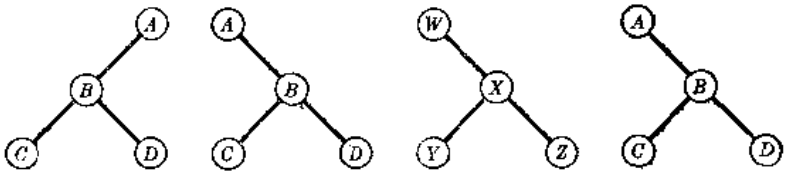
迄今为止，我们关于穿线二叉树的讨论，都是把穿线链接运用到左边和右边两个方面。在完全非穿线与完全穿线的表示方法之间，有着一个重要的中间地带：一个右穿线的二叉树，可通过穿线的RLINK，但利用LLINK = Λ表示空的左子树形，把这两个方法结合在一起（类似地，一个左穿线的二叉树仅仅穿那些空的LLINK）。算法S并不真正地利用穿线的LLINK；如果我们把步骤S2中的测试“LTAG=+”变成“LLINK≠Λ”，就可得到一个以对称的次序遍历右穿线二叉树的算法。程序S对右穿线的情况无须进行改变就可工作。二叉树结构的大量应用，都只是仅仅利用函数P$或P*，自左至右地遍历树，而且对于这些应用，没有必要穿线LLINK。我们已经描述了既在左边又在右边两个方面的穿线，为的是表明对称性以及产生这种情况的可能性，但是实际上，单边的穿线是更为普通的。

现在让我们来考虑关于二叉树的一个重要的概念，以及它与遍历的联系。说两个二叉树T与T'是相似的，是指如果它们具有相同的结构的话；形式地说，这就是：（a）它们两个都是空的，或者（b）它们两个都不空，而且它们的左、右子树都分别相似。非形式地说，相似性意味着T与T'的图式有相同的“形状”。分析相似性的另一个方法是说明，在结构T与T'的节点之间存在一一对应的关系：如果T中的节点u_1和u_2分别对应于T'中的u_1'和u_2'，那么，要说u_1是在u_2的左子树中，当且仅当，u_1'是在u_2'的左子树中，而且对于右子树，也有同样的事实成立。

二叉树T与T'是等价的，是说如果它们是相似的，而且，对应的节点包含相同的信息。形式地说，命info(u)表示包含在某个节点u中的信息；于是，说树是等价的，乃是

当且仅当（a）它们都是空的，或者（b）它们都是非空的，而且info (root(T))=info (root(T'))，以及它们的左子树和右子树分别是等价的。

作为这些定义的例子，试考察下面四个二叉树



其中头两个是不相似的；第二，第三与第四个则是相似的，事实上，第二个与第四个还是等价的。

某些涉及到树形结构的计算机的应用，要求有一个能判定两个二叉树是否相似或等价的算法。在这方面，下列定理是有用处的：

定理 A 设二叉树 T 与 T' 的节点，按照先根次序分别是

$$u_1,\ u_2,\ \cdots,\ u_n \quad 与 \quad u'_1,\ u'_2,\ \cdots,\ u'_{n'}$$

对于任一节点 u，规定

$$\begin{aligned}&\text{如果 } u \text{ 有一个非空的左子树，则 } l(u)=1，\text{否则 } l(u)=0\\&\text{如果 } u \text{ 有一个非空的右子树，则 } r(u)=1，\text{否则 } r(u)=0\end{aligned}\tag{10}$$

于是，说 T 与 T' 是相似的，当且仅当 $n=n'$ 且

$$l(u_j)=l(u'_j),\quad r(u_j)=r(u'_j)\quad \text{对于 } 1\leqslant j\leqslant n\tag{11}$$

说 T 与 T' 是等价的，除上述条件外还需有

$$\text{info}(u_j)=\text{info}(u'_j)\quad \text{对于 } 1\leqslant j\leqslant n\tag{12}$$

注意 l 和 r 实质上就是在一个穿线的树中 LTAG 和 RTAG 的内容，以 1 和 0 代替“+”和“−”。这个定理是借助于两个 0 和 1 的序列，来表征任何二叉树的结构。

证明 显然，如果我们证明了关于相似性的条件，将会立即推出关于二叉树形等价性的条件；其次 $n=n'$ 和 (11) 肯定地是必要的，因为相似树形的对应的节点，在先根次序下必须有相同的位置。因此，只要证明条件 (11) 和 $n=n'$ 能保证 T 与 T' 的相似性，就足够了。证明可通过对 n 采用归纳法来进行，要用到以下的辅助结果：

引理 P 设一个非空的二叉树的节点，在先根次序下是 $u_1,\ u_2,\ \cdots,\ u_n$，又设 $f(u)=l(u)+r(u)-1$。于是

$$\begin{aligned}&f(u_1)+f(u_2)+\cdots+f(u_n)=-1 \qquad \text{且}\\&f(u_1)+\cdots+f(u_k)\geqslant 0 \quad \text{对于 } 1\leqslant k\leqslant n\end{aligned}\tag{13}$$

证明 对于 $n=1$，结果是显然的。如果 $n>1$，则二叉树是由它的根 u_1 和更进一步的节点所组成。如果 $f(u_1)=0$，则或者左子树为空或者右子树为空，所以按照归纳法，这个条件显然是成立的。如果 $f(u_1)=1$，则设左子树有 n_l 个节点，按归纳法我们便有

$$\begin{aligned}&f(u_1)+\cdots+f(u_k)>0 \quad \text{对于 } 1\leqslant k\leqslant n_l\\&f(u_1)+\cdots+f(u_{n_l+1})=0\end{aligned}\tag{14}$$

而且条件 (13) 再次又是显然的。∎

（对于类似于引理 P 的定理，见第 11 章关于“波兰记号”的讨论）。

为了完成定理 A 的证明，注意当 $n=0$ 时，这显然是正确的。如果 $n>0$，则依先根

次序的定义意味着 u_1 与 u'_1 分别是它们的树的根，而且存在有整数 n_l，n'_l（左子树的大小），使得

$$u_2, \cdots, u_{n_l+1} \text{ 与 } u'_2, \cdots, u'_{n'_l+1} \text{ 是 } T \text{ 与 } T' \text{ 的左子树}$$

$$u_{n_l+2}, \cdots, u_n \text{ 与 } u'_{n'_l+2}, \cdots, u'_n \text{ 是 } T \text{ 与 } T' \text{ 的右子树}$$

如果我们能够证明 $n_l = n'_l$，按照归纳法这个证明就算完成了。存在有三种情况：

如果 $l(u_1) = 0$，则 $n_l = 0 = n'_l$；

如果 $l(u_1) = 1$，$r(u_1) = 0$，则 $n_l = n - 1 - n'_l$；

如果 $l(u_1) = r(u_1) = 1$，则按引理 P 我们能够找到一个最小的 $k > 0$ 使得 $f(u_1) + \cdots + f(u_k) = 0$；而且 $n_l = k - 1 = n'_l$（参照等式 14）。∎

作为定理 A 的一个推论，我们还可以通过前单地在先根次序下遍历它们并校验 INFO 和 TAG 场，来测试两个穿线的二叉树的等价性或相似性。

安·杰·布利格尔（A. J. Blikle）已经得到了定理 A 的某些有趣的推广，见《波兰科学院报告数学、天文学、物理学部分》(Bull. de l'Acad Polonaise des Sciences, Série. des sciences math., astr., Phys.) 14 (1966)203～208；他考虑了无穷类可能的遍历次序，仅其中的六个（包括先根次序），由于它们的简单性质，而被称为“无地址”类。

在结束这一小节之前，让我们来给出一个典型而又基本的关于二叉树的算法。这个算法可把一个二叉树复写到不同的内存单元中。

算法 C（复写一个二叉树） 设 HEAD是一个二叉树 T 的表头的地址(即. T是HEAD的左子树；LLINK（HEAD）是一个指向这个树的指针)。设 NODE(U) 是一个具有空左子树的节点。该算法给出 T 的一个拷贝，而且这个拷贝将成为 NODE(U) 的左子树，特别是，如果 NODE(U) 是一个空的二叉树的表头，这个算法则把空树形改变成为 T 的一个拷贝。

C1.〔初始化〕 置 P←HEAD，Q←U。转到 C4。

C2.〔右边有什么？〕 如果 NODE(P) 有一个非空的右子树，则置 R←AVAIL，并把 NODE(R)加进 NODE(Q)的右边（在步骤 C2 开始时，NODE(Q) 的右子树是空的)。

C3.〔复写 INFO〕 置 INFO(Q)←INFO(P)（这里的 INFO 表示有待复写的节点的所有部分)。

C4.〔左边有什么？〕 如果 NODE(P)有一个非空的左子树，则置 R←AVAIL，并且把NODE(R)加入到 NODE(Q)的左边(在步骤 C4 开始时，NODE(Q)的左子树是空的)。

C5.〔前进〕 置 P←P*，Q←Q*。

C6.〔测试完成没有？〕 如果 P = HEAD（或者等价地说，如果Q = RLINK(U)；假定NODE(U) 有一个非空的右子树，则该算法终止；否则转到步骤 C2。∎

这个简单的算法说明了树形遍历的一个典型的应用；这里的描述可应用于穿线的，非穿线的，或部分穿线的树形。步骤 C5 要求计算先根次序的后继 P* 和 Q*；对于非穿线的树，通常是通过一个辅助堆栈来完成的。算法 C 的正确性证明见于习题 29 中；在右穿线二叉树的情况下，相应于这个算法的一个 MIX 程序出现在习题 2.3.2-13 中。对于穿线的树，步骤 C2 和 C4 中的“加入”是利用算法 I 完成的。

下面的一些习题，的确包含了少量的与这一小节的题材有关的有趣内容。

习题

1.〔01〕 在二叉树（2）中，命 INFO(P) 表示存于 NODE(P) 的字母，那么 INFO(LLINK (RLINK (RLINK(T)))) 是什么呢？

2.〔11〕 分别以（a）先根次序、（b）对称次序和（c）后根次序，列出二叉树的节点。



3.〔20〕 下列命题是真还是假："一个二叉树的终端节点，在先根次序下，在中根次序下，以及在后根次序下，都出现在相同的相对位置上。"？

▶4.〔20〕 正文里定义了为遍历一个二叉树的三个基本次序；另有一种选择是将三个步骤处理如下：

a）访问根；

b）遍历右子树；

c）遍历左子树。

对所有的非空子树递归地使用这同一规则。这个新的次序与三个已经讨论过的次序，有什么简单的关系？

5.〔22〕 在类似于树形的"杜威记数法"的记号下，一个 0 和 1 的序列可以如下的标识一个二叉树的节点：根（如果存在的话）以序列"1"表示。以 α 表示的节点，其左、右子树的根（如果存在的话）分别用 $\alpha 0$ 和 $\alpha 1$ 表示。例如，（1）中的节点H将表示为"1110"（参照习题 2.3-15）。

试说明，借助于这个记号可以很方便地描述先根次序，中根次序，和后根次序。

6.〔*M*22〕 假设一个二叉树有 n 个节点，这些节点在先根次序下是 $u_1u_2\cdots u_n$，而在中根次序下是 $u_{p_1}u_{p_2}\cdots u_{p_n}$。试证明，在习题 2.2.1-2 的意义下，让 1，2，…，n 通过一个堆栈，即可得到排列 $p_1p_2\cdots p_n$。反之，试证明，用一个堆栈可得到的任何排列 $p_1p_2\cdots p_n$，在这种方式下对应于某个二叉树。

7.〔22〕 证明：如果给了我们一个二叉树节点的先根次序和中根次序，那么该二叉树结构即可构造出来。如果给了我们先根次序和后根次序（而不是中根次序），同一结果是否保持成立？或者，如果给了我们中根次序和后根次序，又将如何？

8.〔20〕 找出所有的二叉树，其节点在下列两种次序之下，恰好都以同样的顺序出现：（a）先根次序和中根次序；（b）先根次序和后根次序；（c）中根次序和后根次序。

9.〔*M*20〕 当一个具有 n 个节点的二叉树使用算法 T 来遍历时，试指出步骤 T1，T2，T3，T4，和 T5 每一个被实施了多少次（以 n 的函数形式来表示）？

▶10.〔20〕 如果二叉树有 n 个节点，当执行算法 T 时，一次可能在堆栈中的最大条款数是多少（如果堆栈正在被相继地存储，那么这个问题的答案，对于存储分配来说就是十分重要的了）？

11.〔*M*43〕 假定具有 n 个节点的所有二叉树都认为是同样可能的，试作为 n 的一个函数，来分析在执行算法 T 时最大堆栈大小的平均值。

12.〔22〕 设计一个类似于算法T的算法，它可在先根次序下遍历一个二叉树，并证明你的算法是正确的。

▶13.〔24〕 试设计一个类似于算法T的算法，它可在后根次序下遍历一个二叉树。

14.〔22〕 证明：如果一个具有 n 个节点的二叉树象（2）那样表示，则在这个表示中A链接的总个数可以表达为 n 的一个简单函数；这个数量并不依赖于树的形状。

15.〔19〕 注意在象（9）那样的一个穿线的树形表示中，除了表头之外，每一个节点恰好有一个从上边指向它的链接，也就是说，从它的“父亲”引出的链接。此外，某些节点还有从下边指向它的进一步的链接。例如，包含“C”的节点有两个从下边引上来的指针，节点“E”有一个。在指向一个节点的链接的个数与该节点的某些其它的基本性质之间，是否有任何简单的联系（在考虑树结构的变动时，当然是需要知道有多少链接指向一个给定的节点）？

▶16.〔22〕 图24中的图式有助于通过NODE(Q)附近的结构，给出二叉树中NODE(Q\$)位置的直观特征：如果NODE(Q)有一个非空的右子树，则在上边的图式中考虑Q=\$P，Q\$=P；NODE(Q\$)就是该右子树“最左”的节点；如果NODE(Q)有一个空的右子树，则在下边的图式中考虑Q=P；通过在这个树形中向上进行直到头一个向上的步骤向右转之后，便定出了NODE(Q\$)的位置。

为了借助于NODE(Q)附近的结构，来找出在一个二叉树中NODE(Q*)的位置，试给出一个类似的直观规则。

▶17.〔22〕 试给出一个类似于S的算法用于确定在一穿线的二叉树中的P*。假定这个树有一个像(7)，(8)，(9)中那样的表头。

18.〔24〕 许多处理树形的算法宁愿两次访问每一个节点，而不是访问一次。使用先根次序和中根次序的组合，这种组合我们称之为双重次序。在双重次序下，一个二叉树之遍历定义如下：如果二叉树是空的，则什么也不做；否则，

a）访问根，作为第一次；

b）在双重次序下遍历左子树；

c）第二次访问根；

d）在双重次序下遍历右子树。

例如，在双重次序下，遍历（1）给出序列

$$A_1B_1D_1D_2B_2A_2C_1E_1E_2G_1G_2C_2F_1H_1H_2F_2J_1J_2$$

其中 A_1 意味着“A”是头一次被访问，等等。

假如P指向树的一个节点，而且 $d=1$ 或2，那么，定义 $(P,d)^A=(Q,e)$：或者如果在双重次序下在第 d 次访问NODE(P)之后的下一步骤是第 e 次访问NODE(Q)；或者如果 (P,d) 是在双重次序下的最后步骤且 $(Q,e)=(HEAD,2)$，其中HEAD是表头的地址。我们还定义 $(HEAD,1)^A$ 为在双重次序下的第一个步骤。

试设计一个类似于T的算法，它在双重次序下遍历一个二叉树，再设计一个类似于S的算法，它计算 $(P,d)^A$。试讨论这些算法与习题12和17之间的关系。

19.〔24〕 试设计一个类似于算法S并用来计算在一个穿线的二叉树中的P#的算法。

20.〔23〕 修改算法T，使得它在一个链接的表中，而不是在连续的内存单元中，来保

存堆栈。

21.〔32〕设计一个算法，它不使用任何辅助的堆栈，在中根次序下遍历一个非穿线的二叉树。在正当实施这个算法时，允许以无论什么方式来改变树形节点的LLINK和RLINK场，仅仅假定这样一个条件，即在你的算法遍历这个树形之前和之后的两种情况下，二叉树都有它通常的表示〔例如，象（2）中那样〕。你也可以在每个节点中使用一个RTAG场（仅一位）作为临时的存储。

22.〔25〕针对习题21中的算法，试写出一个MIX程序，并同程序S和T比较执行时间。

23.〔22〕试设计类似于I的算法用于插入到一个右穿线的二叉树之右边和插入到其左边的算法。假定这些节点都包含有LLINK，RLINK和RTAG场。

24.〔*M*20〕如果T与T'的节点都是以对称次序而不是以先根次序给出的，则定理A是否仍然成立？

25〔*M*24〕设$\mathscr{T}$是二叉树的集合，并设$N(\mathscr{T})$是集合{info(u)| u是$\mathscr{T}$中某个T的一个节点}。假设在$N(\mathscr{T})$上已经定义了一个线性的次序关系（参照习题2.2.3-14）。给定$\mathscr{T}$中的任何树T，T'，现在让我们来定义$T\prec T'$当且仅当

1）T是空树；或

2）T与T'非空，且info(root(T))$\prec$info(root(T'))；或

3）T与T'非空，info(root(T))=info(root(T'))，左子树(T)$\preceq$左子树(T')，而且，左子树(T)不等价于左子树(T')；或

4）T与T'非空，info(root(T))=info(root(T'))，左子树（T）等价于左子树（T'），而且，右子树（T）$\prec$右子树（T'）。

试证明：(a)$T\preceq T'$和$T'\preceq T''$将导出$T\preceq T''$；(b)T等价于T'当且仅当$T\preceq T'$和$T'\preceq T$；(c)对$\mathscr{T}$中的任意T，T'，我们有$T\preceq T'$或$T'\preceq T$。〔于是，如果$\mathscr{T}$中等价的树形被认为是相等的，则关系$\prec$引起$\mathscr{T}$上的一个线性次序。这个次序有许多应用（例如，在代数表达式化简中）。当$N(\mathscr{T})$仅有一个元素时，即，如果每个节点的“info”都相同时，则我们就有等价性即等同于相似性的特殊情形〕。

26.〔*M*24〕考虑上题中所定义的次序$T\preceq T'$。试证明一个类似于定理A的定理，给出$T\preceq T'$的必要和充分条件，并利用在习题18中所定义的那种双重次序。

▶27.〔28〕试设计一个算法，它借助于习题25中所定义的关系，来测试两个给定的树形T与T'，看是否$T\prec T'$，$T\succ T'$，或T等价于T'，假定这两个树形都是右穿线的，并且每个节点都包含有场LLINK，RLINK，RTAG，INFO；不得利用辅助堆栈。

28.〔00〕在已经用算法C来复写一个树形之后，新的二叉树是等价于原来的树形呢，还是相似于它呢？

29.〔*M*25〕尽可能严格地证明算法C成立。

▶30.〔22〕试设计一个对一非穿线的树进行穿线的算法〔例如，把（2）变换成（9）〕。注意，当有可能不重复象T那样的遍历算法步骤时，总是利用记号P*，PS，等等。

31.〔23〕试设计一个算法，它“抹去”一个右穿线的二叉树，即，把该树形的所有除表头之外的节点还给AVAIL表，并使得表头标志着一个空的二叉树。假定每个节点都包

含有 LLINK，RLINK，RTAG 诸场；不要使用辅助堆栈。

32.〔21〕假设二叉树的每个节点有四个链接场：LLINK 和 RLINK，它们指向左、右子树或Λ，如同在一个非穿线的树情况中那样；以及 SUC 和 PRED，它们指向在对称次序下该节点的后继和前驱（于是，SUC(P)=P\$ 和 PRED(P)=\$P。这样的一个树，比一个穿线的树包含更多的信息）。试设计一个针对这样的一个树形进行插入的，类似于 I 那样的算法。

▶33.〔30〕对一个树形进行穿线，有多种方式！在每个节点上利用三个场 LTAG，LLINK，RLINK，考虑下面的表示：

LTAG(P)：就象在穿线的二叉树中那样的定义；

LLINK(P)：总是等于 P*；

RLINK(P)：如同在非穿线的二叉树中那样的定义。

试讨论针对这样一个表示的插入算法，并详细写出针对这种表示的复写算法，即算法 C。

34.〔22〕设 P 指向某个二叉树中的一个节点，并且设 HEAD 是一个空的二叉树表头的地址。试给出一个算法，它由 NODE(P)所在的无论什么树中，撤消 NODE(P)和 NODE(P)后的所有子树，并把以 NODE(P)作为其根的子树加入到 HEAD。假定问题里的所有二叉树都是右穿线的，且每个节点都包含有 LLINK，RTAG，RLINK 诸场。

35.〔40〕试以类似于我们对二叉树所下定义的方式，来定义一个三叉树（更一般地，是对于 $t \geqslant 2$ 的 t 进树），并阐述在本小节中所讨论的内容，哪些能以有意义的方式推广到 t 叉树中（包括在上边的习题中所提及的内容）。

36.〔*M*23〕习题 1.2.1-15 说明：字典次序可把一集合 S 的一个良序，推广成 S 的 n 重元素的一个良序。上边的习题 25 又说明：树节点中的信息的一个线性次序，利用类似的定义，可以推广到树的一个线性次序。如果关系 $\prec$ 使 $N(\mathscr{T})$ 成为良序，则习题 25 所推广的关系，是否为 $\mathscr{T}$ 的一个良序？

▶37.〔24〕（戴·弗格森）如果为了包含两个链接场和一个 info 场，必须用两个计算机字，那么对于有 n 个节点的树来说，表示（2）则要求有 $2n$ 个内存字。试设计一个使用较少空间的二叉树的表示方案，假定一个链接和一个 INFO 场能填入一个计算机字中。

2.3.2 树的二叉树表示

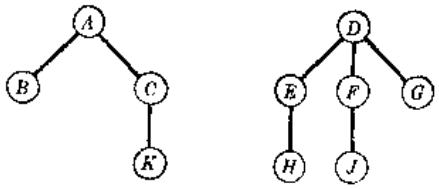
现在让我们从二叉树中掉过头来，讨论完全普通的树。先让我们回想一下，树与二叉树之间的基本区别，按照我们给它们所下的定义，区别在于：

1）一树决不会为空，即，它至少有一个节点，而且一树的每个节点可以有 0，1，2，3，…个儿子。

2）一个二叉树可以是空的，它的每个节点可以有 0，1，或 2 个儿子；我们还要对“左”儿子与“右”儿子加以区别。

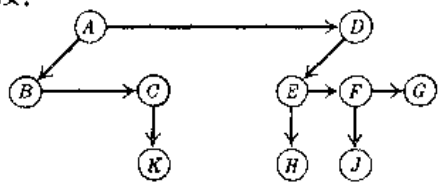
还须回想起，一个“森林”是 0 个或多个树的有序集合。一树的任何节点下面的直接诸子树，又形成一个森林。

有一个自然的方式来把任何森林表示成一个二叉树。考虑下列的两树之森林：



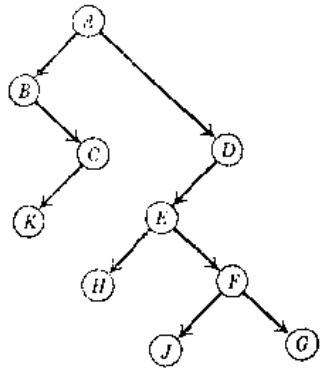
（1）

对应的二叉树可以这样来得到，即，把每个家族的儿子链在一起，并撤消除父亲到他的头一个儿子之外的所有垂直链接：



（2）

然后，把这个图式倾斜过来，并使之稍微弯曲，就能得到：



（3）

反之，通过逆转该过程，容易看出，任何二叉树都对应于一个唯一的树的森林。

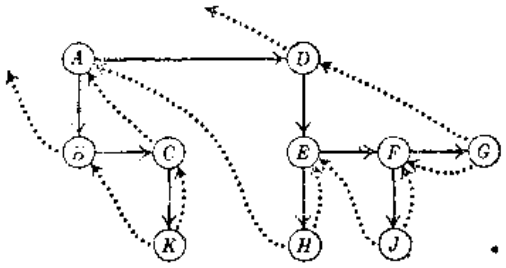
上述变换是极其重要的，它叫做森林与二叉树之间的自然对应（特别是，它给出了树与那些有一个根但没有右子树的二叉树之间的一个对应。我们也可以把事情稍微改变一下，让一个二叉树的表头来对应一个树的根，这样就得到一个具有 $n+1$ 个节点的树与具有 n 个节点的二叉树之间的一一对应）。

设 $F=(T_1, T_2, \cdots, T_n)$ 是树的一个森林。对应于 F 的二叉树 $B(F)$ 可以严格地定义如下：

a）如果 $n=0$，则 $B(F)$ 是空的。

b）如果 $n>0$，则 $B(F)$ 的根是 root(T_1)；$B(F)$ 的左子树是 $B(T_{11}, T_{12}, \cdots, T_{1m})$，其中 $T_{11}, T_{12}, \cdots, T_{1m}$ 是 root(T_1) 的子树；而且 $B(F)$ 的右子树是 $B(T_2, \cdots, T_n)$。这些规则精确地确定了从（1）到（3）的变换。

有时候，无须乎转动 45°，就象在（2）中那样画出二叉树的图式，也是很方便的。对应于（1）的穿线的二叉树是



（4）

（将上图转动 45°，与图 24 进行对照）。注意，右穿线的链接是从一个家族的最右的儿子连到它父亲。左穿线则没有这样一个自然的解释，因为在左边与右边之间缺乏对称性。

在上一小节中所表达的关于遍历的思想，也可以针对森林（同时也是针对着树）来变换一下陈述。没有简单的关于“中根次序”的仿照，因为在一个根的诸后裔当中，没有明显的位置来插入这个根。但是，先根次序和后根次序却可以用明显的方式更换之。给出任何非空的森林，可以如下地来定义遍历它的两个基本的方式：

先根次序	后根次序
a）访问头一个树的根；	a）遍历头一个树的子树（在后根次序下）；
b）遍历头一个树的子树（在先根次序下）；	b）访问头一个树的根；
c）遍历剩下的树（在先根次序下）。	c）遍历剩下的树（在后根次序下）。

为了理解这两个遍历方法的意义，试考虑下列通过嵌套的括弧来表达树形结构的记号：

$$(A(B,C(K)),\ D(E(H),\ F(J),\ G)) \qquad (5)$$

这个记号对应于森林（1）：我们通过在树的根中所写的信息，后边接以它的子树的表示法来表示一树；一个非空森林的表示，是它的诸树表示的一个带圆括弧的表，以逗号把各树表示分隔开。

如果在先根次序下遍历（1），则我们访问诸节点的次序是$ABCKDEHFJG$；这无非就是撤消了圆括弧和逗号的（5）。先根次序是列出一树之节点的一个自然方式：我们首先列出根，然后列出后裔。如果用图20(c)中那样的凹入来表示一个树，这些行将以先根次序出现。本书的节次本身（见图21）就是以先根次序出现的；这样，例如，2.3小节之后是2.3.1小节，然后是2.3.2，2.3.3，2.3.4，2.3.4.1，…，2.3.4.6，2.3.5，2.4，等等。

需要说明一下，先根次序是一个由来已久的概念，是很有趣的；也可以称作朝代次序。当一个国王，公爵，或伯爵等人死去后，头衔总是依次传给他的长子，然后是长子的后裔，最后，如果这一支的人全都死去，则以同样的方式传给这个家族的其他子孙们（根据英国的习惯，女儿与儿子地位相同也包括在家族中的子孙中）。在所有的理论上，我们可以提供所有贵族社会的一个家谱，并以先根次序写出其节点；然后，如果我们仅仅考虑现今活着的人们，我们就将得到宝座的承袭次序（除了由于让位而作的修改外）。

对于（1）中节点的后根次序是$BKCAHEJFGD$；这类似于先根次序，它对应于类似的圆括弧记法

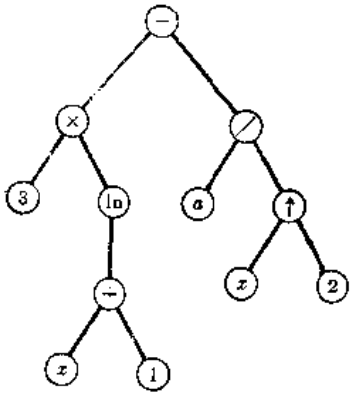
$$((B,(K)C)A,\ ((H)E,\ (J)F,\ G)D), \qquad (6)$$

其中，节点出现在它的后裔之后，而不是在其之前。

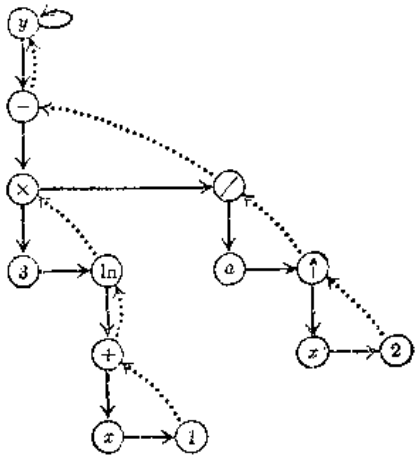
先根次序和后根次序的定义，非常精密地刻画了树与二叉树之间的自然对应，因为头一树的子树都对应到左二叉子树，而剩下的树都对应到右二叉子树。通过把这些定义与二叉树相应的定义进行比较，我们发现，在先根次序下遍历一个森林，正好等同于，在先根次序下遍历其相应的二叉树。在后根次序下遍历一个森林，正好等同于，在中根次序下遍历其相应的二叉树。因此，2.3.1小节中所展示的算法，可以无须改变地加以使用（注意，树的后根次序对应于二叉树的中根次序，而不是后根次序。这一点是很幸运的，因为我们已经看到，在后根次序下遍历二叉树是比较困难的）。

作为把这些方法应用于实际问题的一个例子，让我们来考虑代数公式的操作。把这样

的公式看成是树结构的表示，而不是看成一维或二维的符号配置，甚至也不是看成二叉树，是最适当的了。例如，公式 $y=3\ln(x+1)-a/x^2$ 有树表示



即是，



(7)

这里左边的图式给出了通常的树表示，象图 22 那样，其中二元运算符 +，−，×，/，和 ↑（后者表示乘幂）都有两个与它们的操作数相对应的子树；一元运算符 "ln" 和 "neg"（后者在这树中没有出现，它表示象 "$y=-x$" 中那样的取负值）有一子树，而且变量和常数是终端节点。在右边的图式中，我们已经示出了等价的右穿线的二叉树，包括作为这树的表头的一个附加的节点。表头有 2.3.1-(7) 中所述的形式。

说明这样一点是重要的，即，尽管（7）中左边的树与一个二叉树表面上很相象，但这里我们把它处理成一个树，而且通过在（7）右边所示的一个十分不同的二叉树来表示它。虽然我们可以直接以二叉树的结构为基础，来研制关于代数操作的程序——这些乃是代数公式的所谓 "三地址代码" 表示——但如果我们使用（7）中那样的代数公式的一般树表示，则在实际上就会出现许多简化，因为在一树中以后根次序遍历，是更为容易的。

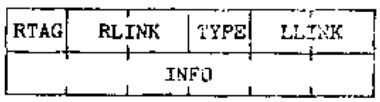
（7）中左边树的节点是

$$-\ \times\ 3\ \ln\ +\ x\ 1\ /\ a\ \uparrow\ x\ 2 \qquad \text{在先根次序下} \tag{8}$$

$$3\ x\ 1\ +\ \ln\ \times\ a\ x\ 2\ \uparrow\ /\ - \qquad \text{在后根次序下} \tag{9}$$

象（8）和（9）这样的代数表达式是非常重要的，而且由于形式（8）是由波兰逻辑学家鲁卡西维兹（Lukasiewicz）引进的，所以它们被称为 "波兰记号"。表达式（8）是公式(7)的前缀记号，而(9)是对应的后缀记号。在后边几章里，我们还将回到有关波兰记号的有趣课题上来；现在就暂且谈这一点，初步了解波兰记号是直接关系到树遍历的基本次序的。

假定我们将要处理的代数公式的树结构，有如下形式的节点：



(10)

这里的 RLINK 和LLINK 具有通常的意义，而且，对于穿线的链接，RTAG 是负的。用 TYPE场来区分不同类型的节点：TYPE＝0 意味着此节点表示一个常数，而 INFO 就是此常数的值；TYPE＝1 意味着此节点表示一个变量，而 INFO 就是这个变量的五个字母的字符名称；TYPE≥2 意味着此节点表示一个运算符，INFO 便是此运算符的字符名称，

用TYPE=2，3，4，…来区别+，−，×，/，等不同的运算符。我们在这里先不去管，在计算机内存中，如何来建立这个树形结构，因为在第10章中将详尽地来分析这个课题。现在让我们假定，这个树形已经在我们的计算机内存中了，至于输入和输出的问题将推迟到以后解决。

我们现在将讨论"经典的"代数操作的例子，即求一个公式关于变量x的微商。关于代数微分法的程序，属于第一批为计算机编制的符号操作程序，它们早在1952年就被使用了。求微商的过程说明了代数操作的许多技术，而且在科学应用中有重要的实用价值。

不熟悉数学演算的读者，可以把这个问题当作是一个由下列规则定义的公式操作的抽象习题：

$$D(x)=1 \tag{11}$$

$D(a)=0$，如果a是一个常数或是一个$\neq x$的变量 (12)

$$D(\ln u)=D(u)/u, \text{ 如果 } u \text{ 是任何公式} \tag{13}$$

$$D(-u)=-D(u) \tag{14}$$

$$D(u+v)=D(u)+D(v) \tag{15}$$

$$D(u-v)=D(u)-D(v) \tag{16}$$

$$D(u\times v)=D(u)\times v+u\times D(v) \tag{17}$$

$$D(u/v)=D(u)/v-(u\times D(v))/(v\uparrow 2) \tag{18}$$

$$D(u\uparrow v)=D(u)\times(v\times(u\uparrow(v-1)))+((\ln u)\times D(v))\times(u\uparrow v) \tag{19}$$

这些规则允许我们，对任何由上述运算符组成的公式y，来计算微商$D(y)$。

（和往常一样我们在这个算法中的主要兴趣在于这个过程在一台计算机内是如何进行的细节。在大多数计算机装置中，有许多高级语言和专用程序可供利用，这些计算机装置有类似于这些的大大地简化代数操作的机内设备。但现在这个例子的目的，是为了对基本的树形运算获得更多的经验）。

下列算法的思想背景，就是在后根次序下遍历树，在我们向前进行的同时，形成每个节点的微商，直到终于算出整个导数为止。利用后根次序就意味着，当我们达到一个运算符节点（象"+"）时，是在它的操作数已经被微商之后。规则（11）到（19）意味着，原来公式的每一个部分公式，迟早都应当被微商，所以我们倒不如就在后根次序下来进行微商。通过利用一个右穿线的树形，我们避免了在算法操作期间使用堆栈。另一方面，穿线的树表示也有缺点，即它需要复写子树〔例如，在关于$D(u\uparrow v)$的规则中，我们可能就需要对u和v各复写三次〕，然而在许多情况下，我们可以用列表表示来代替一个树，以避免这种复写；见2.3.5小节。

算法D（微分法） 如果Y是指向表示成上述那种公式的表头的地址，又若DY是一空树的表头的地址，则这个算法将使得NODE(DY)指向一个表示Y关于变量"X"的解析导数的树。

D1. 〔初始化〕置P←Y $（即，树形在后根次序下的头一个节点，它是相应的二叉树在中根次序下的头一个节点）。

D2. 〔微分〕置P1←LLINK(P)；而且如果P1≠A，则亦置Q1←RLINK(P1)。然后实施下边所述的程序DIFF[TYPE(P)]（程序DIFF[0]，DIFF[1]，等等，将形成

根P的树形的微商，并置指针变量Q为微商的根的地址。首先建立P1和Q1，为的是简化DIFF程序的说明)。

D3. 〔调整链接〕如果TYPE(P)表示一个二元运算符，则置RLINK(P1)←P2(关于说明，见下一个步骤)。

D4. 〔进到P\$〕置P2←P，P←P\$。现在，如果RTAG(P2)="+"，即NODE(P2)的右边有一个兄弟，则置RLINK(P2)←Q(这是本算法的技巧：暂时破坏树Y的结构，使得保存一个指向P2微商的链接以供将来使用。这个掉落的链接在步骤D3中又将重新插入。对于这一技巧的进一步讨论，见习题21)。

D5. 〔已完成?〕如果P≠Y，则返回步骤D2。否则置LLINK(DY)←Q和RLINK(Q)←DY，RTAG(Q)←"−"。

算法D中描述的过程，只是关于微商操作的基础程序，这些操作是通过在步骤D2中调用处理程序DIFF(0)，DIFF(1)，…来实现的。在许多方面，算法D就好象是一个解释系统或机器模拟程序的控制程序，很像1.4.3小节中所讨论的那样；但是，它是遍历一个树，而不是一个简单的指令序列。

现在让我们来考虑实际进行微商的程序。在下面的讨论中，语句"P指向一个树"意味着NODE(P)是一个以通常方式存储的树形的根，而RLINK(P)和RTAG(P)两者就这树而论是没有意义的。我们将使用一个树构造函数，通过把一些较小的树形合在一起，作成新的树形；命x表示某类节点、常数、变量，或是运算符，又命U和V表示指向树的指针；于是

TREE(x, U, V) 便做成新树，它以x为其根节点，并以U和V作为这个根的子树：
W⇐AVAIL，INFO(W)←x，LLINK(W)←U，RLINK(U)←V，
RTAG(U)←"+"，RLINK(V)←W，RTAG(V)←"−"

TREE(x, U) 类似地仅以一个子树做成一个新的树：W⇐AVAIL，INFO(W)←x，
LLINK(W)←U，RLINK(U)←W，RTAG(U)←"−"

TREE(x)做成一个以x为一终端根节点的新树：W⇐AVAIL，INFO(W)←x，
LLINK(W)←Λ

在所有的情况下，TREE的值都是W，一个指向刚刚构造的树的指针。读者应细心地研究上述定义，因为它们说明了树的二叉树表示。另一个函数COPY(U)，做出一个由U指出的树的复写，而且它的值是指向由此建立的树的指针。

基本函数TREE和COPY，便于逐步地构造出对于一个公式的导数的树。然而，在我们考察DIFF程序之前，我们还要考虑到，如果我们盲目地将规则(11)至(19)应用于象

$$y = 3\ln(x+1) - a/x^2$$

这样一个颇为简单的公式，将会发生什么情况。我们会得到

$$\begin{aligned} D(y) = {} & (0\cdot\ln(x+1) + 3((1+0)/(x+1)) \\ & -(0/x^2 - (a(1(2x^{2-1}) + ((\ln x)\cdot 0)x^2))/(x^2)^2) \end{aligned} \tag{20}$$

这是不能令人满意的。为了避免在答案中出现如此之多的冗余运算，还必须使我们的程序更加复杂些，使得它能认识到，加以或乘以0，乘以1，或1次乘方等特殊情况。有了这些简化将把(20)归约为

$$D(y)=3(1/(x+1))-((-(a(2x)))/(x^2)^2) \tag{21}$$

这是较为可取的，显然还不是十分令人满意的。什么是真正满意的答案尚未完全定义出来，因为不同的数学家喜欢以不同的方式表达公式；然而，(21) 显然还不象可能的那样简单。为了获得比公式 (21) 更好的实质性的改进，就有必要来研制代数的简化程序（见习题 17），例如它将把式 (21) 归约成

$$D(y)=3(x+1)^{-1}+2ax^{-3} \tag{22}$$

目前，我们暂且限于考虑能产生出 (21)，而不是 (22) 的程序。

零元运算符（常数和变量） 对于这些运算，NODE(P)是一个终端节点，而且在运算之前P1，P2，Q1 和Q的值是不相干的。

DIFF[0]：(NODE(P)是一个常数)置Q←TREE(0)。

DIFF[1]：(NODE(P)是一个变量)。如果 INFO(P)="X"，则置Q←TREE(1)；否则置Q←TREE(0)。

一元运算符（对数和取负值） 对于这些运算，NODE(P)只有一个儿子U，由P1指出；还有一个Q指向$D(U)$。运算之前P2和Q1的值是不相干的。

DIFF[2]：(NODE(P)是"ln")。如果 INFO(Q)≠0，则置Q←TREE("/", Q, COPY(P1))。

DIFF[3]：(NODE(P)是"neg")。如果 INFO(Q)≠0，则置Q←TREE("neg", Q)。

二元运算符（加法，减法，等等） 对于这些运算，NODE(P)有两个儿子U和V，分别通过P1和P2来指出之；Q1和Q分别指向$D(U)$和$D(V)$。

DIFF[4]：("+"运算)如果 INFO(Q1)=0，则置AVAIL⇐Q1。否则，如果INFO(Q)=0，则置AVAIL⇐Q和Q←Q1；否则置Q←TREE("+", Q1, Q)。

DIFF[5]：("−"运算)如果 INFO(Q)=0，则置AVAIL⇐Q和Q←Q1。否则，如果 INFO(Q1)=0，则置AVAIL⇐Q1并置Q←TREE("neg", Q)；否则置Q←TREE("−", Q1, Q)。

DIFF[6]：("×"运算)如果 INFO(Q1)≠0，则置Q1←MULT(Q1, COPY(P2))。然后，如果 INFO(Q)≠0，则置Q←MULT(COPY(P1), Q)。然后转到DIFF[4]。

这里 MULT(U, V) 是一个构造U×V的树形的新函数，但它也进行U或V是否等于"1"的测试：

如果 INFO(U)=1 且 TYPE(U)=0，则置AVAIL⇐U和MULT(U, V)←V；

如果 INFO(V)=1 且 TYPE(V)=0，则置AVAIL⇐V和MULT(U, V)←U；

否则置 MULT(U, V)←TREE("×", U, V)。

DIFF[7]：("/"运算)如果 INFO(Q1)≠0，则置

Q1←TREE("/", Q1, COPY(P2))

然后如果 INFO(Q)≠0，则置

Q←TREE("/", MULT(COPY(P1), Q), TREE("↑", COPY(P2), TREE(2)))。

然后转到 DIFF[5]。

DIFF〔8〕:(“↑”运算)见习题12。

在结束这一小节之前，我们来阐明，怎样仅仅以MIX机器语言作为基础，“白手起家”，但是却很容易把上述所有运算，都变换成一个计算机程序。

程序D（微分法） 下列的MIXAL程序实施算法D，以rI2≡P，rI3≡P2，rI4≡P1，rI5≡Q，rI6≡Q1。为方便起见，计算顺序重新安排了一下。

```
01    *DIFFERENTIATION IN A RIGHT-THREADED TREE
02    LLINK   EQU     4:5         场的定义，见(10)
03    RLINK   EQU     1:2
04    RLINKT  EQU     0:2
05    TYPE    EQU     3:3
06    *MAIN CONTROL ROUTINE       D1. 初始化
07    D1      STJ     9F          把整个过程处理作子程序
08            LD4     Y(LLINK)    P1←LLINK(Y)，准备找Y$
09    1H      ENT2    0,4         P←P1
10    2H      LD4     0,2(LLINK)  P1←LLINK(P)
11            J4NZ    1B          如果P1≠Λ，则重复
12    D2      LD1     0,2(TYPE)   D2. 微分
13            JMP     *+1,1       跳到DIFF[TYPE(P)]
14            JMP     CONSTANT    对应于DIFF[0]的开关表入口
15            JMP     VARIABLE          DIFF[1]
16            JMP     LN                 .
17            JMP     NEG                .
18            JMP     ADD                .
19            JMP     SUB                .
20            JMP     MUL                .
21            JMP     DIV                .
22            JMP     PWR               DIFF[8]
23    D3      ST3     0,4(RLINK)  D3. 调整链接。RLINK(P1)←P2
24    D4      ENT3    0,2         D4. 进到P$。P2←P
25            LD2     0,2(RLINKT) P←RLINKT(P)
26            J2N     1F          如果RTAG(P)="-"则跳
27            ST5     0,3(RLINK)  否则置RLINK(P2)←Q
28            JMP     2B          注意NODE(P$)将是终端
29    1H      ENN2    0,2
30    D5      ENT1    -Y,2        D5. 已完成?
31            LD4     0,2(LLINK)  P1←LLINK(P)，为步骤D2作准备
32            LD6     0,4(RLINK)  Q1←RLINK(P1)
33            J1NZ    D2          如果P≠Y则跳到D2
34            ST5     DY(LLINK)   否则置LLINK(DY)←Q
35            ENNA    DY
36            STA     0,5(RLINKT) RLINK(Q)←DY，RTAG(Q)←"-"
37    9H      JMP     *           从微分子程序出口
```

程序的后一部分包括基本子程序TREE和COPY。前者按照正在被构造的树的子树个数，

有三个入口 TREE 0，TREE 1，及TREE 2。不论使用的是该子程序的那个入口，rA 部包含着一个特殊常数的地址，这个特殊常数指出：什么类型的节点形成了这个正被构造的树的根；这些特殊的常数出现在行 105～124 中。

```
38    * BASIC SUBROUTINES FOR TREE CONSTRUCTION
39    TREE0   STJ     9F              TREE (rA) 函数
40            JMP     2F
41    TREE1   ST1     3F(0:2)         TREE (rA,rI1) 函数
42            JSJ     1F
43    TREE2   STX     3F(0:2)         TREE (rA,rX,rI1) 函数
44    3H      ST1     *(RLINKT)       RLINK(rX)←rI1, RTAG(rX)←"+"
45    1H      STJ     9F
46            LDXN    AVAIL
47            JXZ     OVERFLOW
48            STX     0,1(RLINKT)     RLINK(rI1)←AVAIL, RTAG(rI1)←"-"
49            LDX     3B(0:2)
50            STA     *+1(0:2)
51            STX     *(LLINK)        置下一个根节点的LLINK
52    2H      LD1     AVAIL           rI1←AVAIL
53            J1Z     OVERFLOW
54            LDX     0,1(LLINK)
55            STX     AVAIL
56            STA     *+1(0:2)        把根节点移到可利用的空间
57            MOVE    *(2)
58            DEC1    2               恢复rI1以指向根节点
59    9H      JMP     *               从TREE出来，结果在rI1
60    COPYP1  ENT1    0,4             COPY(P1)，COPY的特殊入口
61            JSJ     COPY
62    COPYP2  ENT1    0,3             COPY(P2),COPY的特殊入口
63    COPY    STJ     9F              COPY(rI1)函数：
 :            :                       (见习题13)
104   9H      JMP     *               从COPY出来，rI1指向新树形
105   CON0    CON     0               表示常数"0"的节点
106           CON     0
107   CON1    CON     0               表示"1"的节点
108           CON     1
109   CON2    CON     0               表示"2"的节点
110           CON     2
111   LOG     CON     2(TYPE)         表示"ln"的节点
112           ALF       LN
113   NEGOP   CON     3(TYPE)         表示"neg"的节点
114           ALF       NEG
115   PLUS    CON     4(TYPE)         表示"+"的节点
116           ALF       +
```

```
117   MINUS     CON     5 (TYPE)      表示 "-" 的节点
118             ALF       -
119   TIMES     CON     6 (TYPE)      表示 "×" 的节点
120             ALF       *
121   SLASH     CON     7 (TYPE)      表示 "/" 的节点
122             ALF       /
123   UPARROW   CON     8 (TYPE)      表示 "↑" 的节点
124             ALF       * *
```

程序的其余部分对应于微分程序 DIFF(0), DIFF(1), …; 我们已经把这些程序写成: 当处理了一个二元运算符之后, 便把控制返回到步骤 D3; 否则控制返回到步骤D4。

```
125   *DIFFERENTIATION ROUTINES
126   VARIABLE  LDX     1,2
127             ENTA    CON 1
128             CMPX    2 F           INFO(P) = X?
129             JE      * + 2         若然, 则调用 TREE(1)
130   CONSTANT  ENTA    CON 0         调用 TREE(0)
131             JMP     TREE 0
132   1 H       ENT 5   0,1           Q←新树形的单元
133             JMP     D 4           返回控制程序
134   2 H       ALF       X
135   LN        LDA     1,5
136             JAZ     D 4           如果INFO(Q) = 0, 则返回控制程序
137             JMP     COPYP 1       否则置rI1←COPY(P1)
138             ENTX    0,5
139             ENTA    SLASH
140             JMP     TREE 2        rI 1←TREE("/", Q, rI 1)
141             JMP     1 B           Q←rI 1, 返回控制
142   NEG       LDA     1,5
143             JAZ     D 4           如果INFO(Q) = 0, 则返回
144             ENTA    NEGOP
145             ENT 1   0,5
146             JMP     TREE 1        TREE ("neg", Q)
147             JMP     1 B           →Q, 返回控制
148   ADD       LDA     1,6
149             JANZ    1 F           转移, 除非INFO(Q 1) = 0
150   3 H       LDA     AVAIL         AVAIL←Q 1
151             STA     0,6 (LLINK)
152             ST 6    AVAIL
153             JMP     D 3           返回控制, 二元运算符
154   1 H       LDA     1,5
155             JANZ    1 F           转移, 除非INFO(Q) = 0
156   2 H       LDA     AVAIL         AVAIL←Q
157             STA     0,5(LLINK)
```

```
158                 ST5     AVAIL
159                 ENT5    0,6             Q←Q1
160                 JMP     D3              返回控制
161   1H            ENTA    PLUS            准备调用TREE ("+", Q1, Q)
162   3H            ENTX    0,6
163                 ENT1    0,5
164                 JMP     TREE2
165                 ENT5    0,1             Q←TREE ("±", Q1, Q)
166                 JMP     D3              返回控制
167   SUB           LDA     1,5
168                 JAZ     2B              如果INFO (Q) = 0, 则跳
169                 LDA     1,6
170                 JANZ    1F              跳, 除非INFO (Q1) = 0
171                 ENTA    NEGOP
172                 ENT1    0,5
173                 JMP     TREE1
174                 ENT5    0,1             Q←TREE ("neg", Q)
175                 JMP     3B              AVAIL←Q1 并返回
176   1H            ENTA    MINUS           准备调用TREE ("-", Q1, Q)
177                 JMP     4B
178   MUL           LDA     1,6
179                 JAZ     1F              如果INFO(Q1) = 0, 则跳
180                 JMP     COPYP2          否则置rI1←COPY(P2)
181                 ENTA    0,6
182                 JMP     MULT            MULT (Q1, COPY(P2))
183                 ENT6    0,1             →Q1
184   1H            LDA     1,5
185                 JAZ     ADD             如果INFO(Q) = 0, 则跳
186                 JMP     COPYP1          否则置rI1←COPY(P1)
187                 ENTA    0,1
188                 ENT1    0,5
189                 JMP     MULT            MULT(COPY(P1), Q)
190                 ENT5    0,1             →Q
191                 JMP     ADD
192   MULT          STJ     9F              MULT(rA, rI1)子程序
193                 STA     1F(0:2)         设rA=U, rI1=V
194                 ST2     8F(0:2)         保存rI2
195   1H            ENT2    *               rI2←U
196                 LDA     1,2             测试是否INFO(U) = 1
197                 DECA    1
198                 JANZ    1F
199                 LDA     0,2(TYPE)       及是否TYPE(U) = 0
200                 JAZ     2F
201   1H            LDA     1,1             若否, 则测试是否INFO(V) = 1
```

```
202             DECA    1
203             JANZ    1F
204             LDA     0,1(TYPE)       及是否TYPE(V) = 0
205             JANZ    1F
206             ST1     *+2(0:2)        若然，则互换U↔V
207             ENT1    0,2
208             ENT2    *
209     2H      LDA     AVAIL           AVAIL←U
210             STA     0,2(LLINK)
211             ST2     AVAIL
212             JMP     8F              结果是V
213     1H      ENTA    TIMES
214             ENTX    0,2
215             JMP     TREE2           结果是TREE("×", U, V)
216     8H      ENT2    *               恢复rI2的设置
217     9H      JMP     *               结果在rI1转出MULT。
```

其它两个程序 DIV 和 PWR 是类似的，它们已被留作习题（见习题 15 和 16）。

习题

▶1.〔20〕 正文里给出了$B(F)$，即对应于森林F的二叉树的一个形式定义。试给出该过程之逆的形式定义，即定义$F(B)$，对应于一个二叉树的森林。

▶2.〔20〕 我们已经在 2.3 节中对于森林，并在习题 2.3.1–5 中对于二叉树，定义了杜威记数法。于是，（1）中的节点“J”以“2.2.1”表示之，而在等价的二叉树（3）中它则以“11010”表示。如有可能，则请给出一个规则，能直接把树与二叉树之间的自然对应表达成杜威记数法之间的对应。

3.〔22〕 一个森林的节点的杜威记数法，与这些节点的先根次序和后根次序之间，是什么关系？

4.〔19〕 下列命题是真还是假：“一个树形的终端节点，在先根次序和后根次序下，皆以相同的相对位置出现”？

5.〔23〕 树与二叉树之间的另一个对应，可以通过命 RLINK（P）指向 NODE（P）最右边的儿子，LLINK（P）指向左边最近的兄弟来定义。设F是一个以这种方式对应于一个二叉树B的森林。试问B的节点的什么次序，对应于F的（a）先根次序，（b）后根次序？

6.〔25〕 设T是每个节点都有 0 个或 2 个儿子的一个非空树。如果我们认为T是一个普通的树，则它（通过自然的对应）对应于另一个二叉树T'。试问在T的节点的先根次序，中根次序和后根次序（象对二叉树那样来定义），与T'的节点的同样的三个次序之间，是否有任何简单的关系？

7.〔$M20$〕 如果我们说，每个节点都居于它在树形中的后裔之前，则可以认为一个森林是部分有序的。当这些节点以（a）先根次序，（b）后根次序，（c）反向的先根次序，（d）反向的后根次序列出时，它们是否已被拓扑分类？（如同在 2.2.3 节中定义的那样）。

8.〔*M*20〕 习题 2.3.1-25 说明，在一个二叉树的个别节点中所存的信息之间的一个次序，可如何推广成所有二叉树的一个线性次序。在自然对应下，同一个构造列出所有树的一个次序。试就树来重新陈述该习题的定义。

9.〔*M*21〕 说明，一个森林中非终端节点的总数，与对应的二叉树中等于A的右链接的总数，有一个简单的关系。

10.〔*M*23〕 设F是一个在先根次序下其节点为u_1，u_2，…，u_n的树形之森林，并设F'是一个在先根次序下其节点为u'_1，u'_2，…，$u'_{n'}$的森林。命$S(u)$表示节点u的次数（儿子的个数）。试借助于这些思想，来陈述并证明一个类似于定理 2.3.1A的定理。

11.〔20〕 试画出类似于在（7）中所示的对应于公式$y=e^{-x^2}$的树形。

12.〔*M*21〕 给出对于程序 DIFF〔8〕（"↑"运算）的说明，它在正文的计算中被省略了。

▶13.〔26〕 写出关于 COPY 子程序的一个 MIX 程序（它填入到正文程序里的 63-104 行之间）〔提示：以适当的初始条件，对于右穿线的二叉树的情况采用算法 2.3.1 C〕。

▶14.〔*M*21〕 为复写一个有n个节点的树形，习题 13 的程序要有多长？

15.〔23〕 写出一个如同正文里所描述的 DIFF〔7〕的，对应于 DIV 程序的 MIX 程序（该程序应加到正文中程序的 217 行之后）。

16.〔24〕 如习题 12 中所描述的 DIFF〔8〕，写出对应于 PWR 程序的一个 MIX 程序（在解答习题 15 之后，这个程序应该加到正文程序中）。

17.〔*M*10〕 写出一个例如能够约简（20）或（21）成为（22）的代数化简程序〔提示：对于每个节点包括一个新的场，它表示该节点的系数（对于被加项）或者它的指数（对于一乘积中的因子）。应用代数恒等式，例如以$v\ln u$来代替$\ln(u\uparrow v)$，以及当可能时通过使用等价的加法或乘法运算，来撤消运算－，/，↑和 neg（取负）。把＋和×变成为n元的而不是 2 元的运算符；通过对诸项的操作数在树次序（习题 8）下进行分类，归拼同类项；某些和及乘积现在将约简成 0 或 1，也许还可进行进一步的简化。另一种调整，如通过用乘积的对数来代替对数的和，也提示了简化〕。作为参考，请看琼·萨米特（J. Sammet）的综述性论文，*CACM* 9 (1966)，555～569。

18.〔*M*40〕 考虑由符号逻辑的运算符（比如说，AND，OR，和 NOT）组成的代数公式。试谋求若干不同的算法，来判定这样一个公式是否是一个重言式，即，一个对于它的变量的真假值的所有组合皆为真的公式〔例如，

((X AND Y) OR NOT X) OR NOT (Y AND Z)

就是一个重言式〕。对于所考虑的每一个算法，试分析它的计算效率。关于一个合理的算法的讨论，以及关于早期的著述的参考文献，请看由马·戴维斯（M. Davis）和希·普特南（H. Putnam）所写的论文，*JACM* 7 (1960)，201～215。另一个思想，是对于2^n位数量的布尔运算使用一台二进制计算机，其中n是变量的个数。

19.〔*M*35〕 一个自由格就是一个数学系统，它可以简单地定义成（为了本习题的目的）：由变量和两个抽象的二元运算符"∪"和"∩"组成的所有公式的集合。在自由格中的某些公式X与Y之间，通过下列规则，来定义一个关系"⊇"：

a）$X\cup Y\supseteq W\cap Z$，当且仅当，$X\cup Y\supseteq W$或$X\cup Y\supseteq Z$或$X\supseteq W\cap Z$或$Y\supseteq W\cap Z$

b) $X \cap Y \supseteq Z$，当且仅当，$X \supseteq Z$ 和 $Y \supseteq Z$

c) $Y \supseteq Y \cup Z$，当且仅当，$X \supseteq Y$ 和 $X \supseteq Z$

d) $x \supseteq Y \cap Z$，当且仅当，$x \supseteq Y$ 或 $x \supseteq Z$ 当 x 为一变量时

e) $X \cup Y \supseteq z$，当且仅当，$X \supseteq z$ 或 $Y \supseteq z$ 当 z 为一变量时

f) $x \supseteq y$，当且仅当，$x = y$ 当 x 和 y 为变量时

例如，我们推出 $a \cap (b \cup c) \supseteq (a \cap b) \cup (a \cap c) \not\supseteq a \cap (b \cup c)$。

试设计一个算法，这个算法，对于给定的自由格中的两个公式 X 和 Y，测试是否 $X \supseteq Y$。

▶20. [4/22] 证明，如果 u 和 v 是一棵树的节点，那么，u 是 v 的一个祖宗，当且仅当，在先根次序下 u 在 v 之前，而在后根次序下 u 在 v 之后。

21. [25] 算法D控制着对于二元运算符，一元运算符，以及“0元”运算符，即是对于其节点有次数 2，1，以及 0 之树的微分活动；但它并未明显地表明，对于三元运算符以及有更高次数的节点，如何来处理控制（例如，习题 17 提议，把加法和乘法做成有任何数目的操作数的运算符）。是否有可能以简单的方式推广算法D，使得它将处理次数大于 2 的运算符？

▶22. [M26] 假设 T 和 T' 是树。我们说 T 能嵌入到 T' 中，写成 $T \subseteq T'$，如果有着一个从 T 的节点到 T' 中的节点之一对一的函数 f，使得 f 既保持先根次序又保持后根次序（换句话说，对于 T 在先根次序下 u 在 v 之前，当且仅当，对于 T' 在先根次序下 $f(u)$ 在 $f(v)$ 之前；而且对于后根次序，这同一事实仍然成立。见图 25）。

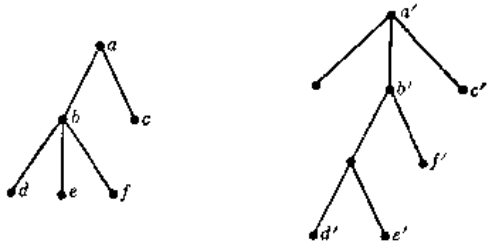


图25 一棵树嵌入到另一棵中（见习题22）

如果 T 有多于一个的节点，则命 $l(t)$ 是 root (T) 最左边的子树，而且 $r(t)$ 是 T 的剩下部分，即是去掉 $l(T)$ 的 T。试证明，T 能被嵌入到 T' 中，如果 T 只有一个节点，或者 T 和 T' 两者都有多于一个的节点，以及或 $T \subseteq l(T')$，或 $T \subseteq r(T')$，或 $l(T) \subseteq l(T')$ 且 $r(T) \subseteq r(T')$。逆命题是否也成立？

2.3.3 树的其它表示

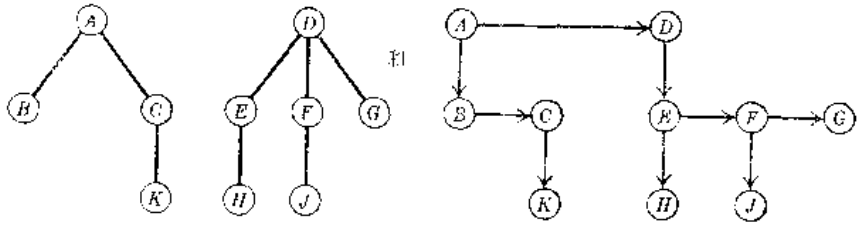
除了在上一小节中给出的 LLINK-RLINK（左儿子-右兄弟姐妹）方法之外，为了在一台计算机内表示树的结构，还有很多方法。通常，表示法选择得当，强烈地依赖于我们对于树形所要实施的运算，是什么类型的。在这一小节，我们将考虑已被证明是有用的一些可能的树形表示方法。

首先，我们可以使用顺序的存储法。如象在线性表中的情况那样，当我们要求一个树形结构的紧凑表示时，而且在程序执行期间，该结构在大小或形状上又不致遭受激烈的动态变化时，那么，这种分配方式就是最适当的了。在许多情况下，我们特别需要树结构的常数表，以便在一程序内访问之。而且，在内存中对这些树形所要求的形式，依赖于对这些表格进行考察的方式。

树（以及森林）的最普通的顺序表示，实质上就相当于省略 LLINK 场，而代之以连续的编址。例如，让我们再次来考察在上一小节中所考虑的森林

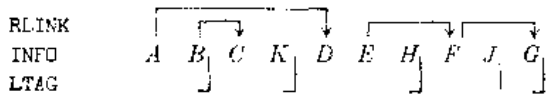
$$(A(B,\ C(K)),\ D(E(H),\ F(J),\ G)), \tag{1}$$

这森林有树形图式



(2)

先根次序的顺序表示，有着以先根次序出现的节点，在每个节点中有INFO，RLINK，和LTAG诸场：

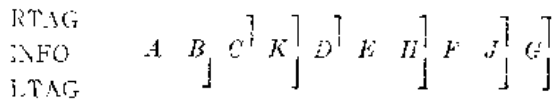


(3)

这里，非空的 RLINK 已经以箭头示出，而 LTAG="—"(对于终端节点) 则以 "⌋" 表示之。LLINK 是不必要的，因为它将或者是空，或者它将指向序列中的下一项。比较 (1) 与 (3) 是有启发的。

这个表示有许多有趣的性质。首先，一个节点的所有子树全都直接地出现于该节点之后，使得在原来森林内的所有子树全都出现在连续的区段中〔把这与 (1) 中及图 20(b) 中的 "嵌套的圆括弧" 加以比较〕。其次，注意在 (3) 中 RLINK 箭头决不彼此交叉：一般说来都是这样，因为在一个二叉树中，所有的在先根次序下介于 X 与 RLINK (X) 之间的节点，都位于 X 的左子树中，因此从树形的该部分将不会冒出向外的箭头。第三，我们可以发现 LTAG 场是多余的，这个场用于指出一个节点是否为终端的。因为，"⌋" 仅仅出现在森林的结尾以及恰好在每个向下指的箭头之前。

其实，这些论述说明，RLINK 场本身就几乎是多余的；我们真正需要的用以表示这个结构的一切，是 RTAG 和 LTAG。因此有可能从少得多的数据：



(4)

来导出 (3)。从左到右扫描 (4)，有着 RTAG≠"⌋" 的位置对应于必须添入的非空的 RLINK。每次我们通过一个有 LTAG="⌋" 的项，我们就应该完成最近的待完成的 RLINK 之配置（待完成的 RLINK 之单元因此可保存于一个堆栈上）。我们实质上又再次证明了定理 2.3.1A。

RLINK 或 LTAG 在 (3) 中是多余的这一事实，对我们的帮助很少乃至毫无帮助。除非我们正在顺序地扫瞄整个的森林，其时需要额外的计算来推导这失去的信息。因此在 (3) 中的完全的数据通常是需要的。然而，明显地有某些浪费的空间，因为对于这个具体的森林，有超过一半的 RLINK 场等于 A。有两种通常的方式来利用浪费的空间：

1) 把每个节点的 RLINK 填成紧跟在该节点下面的子树之后的地址。这个场现在通

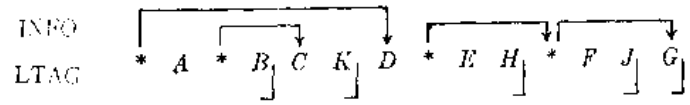
常都称作“SCOPE（辖域）”以代替 RLINK. 因为它表示每个节点的“影响”（后裔）的右边界。代替（3），我们将有



（5）

这些箭头仍不彼此交叉。而且，LTAG（X）=“−”即以条件 SCOPE（X）= X + c 为其特征，假定 c 是每个节点的字数。一个使用 SCOPE 思想的例子，见于习题 2.4-12 中。

2）通过撤消 RLINK 场来缩小每个节点的大小，并且在那些先前有着一个非空的 RLINK 的节点的紧前边，加上特殊的“链接”节点：



（6）

这里“*”表示这些特殊的链接节点，它们的 INFO 使它们以指向如箭头所示者之链接为特征。如果（3）的 INFO 与 RLINK 场占有大致相同的空间数量，那么，改变成（6）就赢得了化费较少内存的纯粹的收益，因为“*”节点的个数总是小于非“*”节点的个数。表示（6）有些类似于在一台如象 MIX 那样的单地址计算机上的一个指令序列，其中之“*”节点就相当于条件转移指令。

通过省略 RLINK 而不是 LLINK，又可以设计出类似于（3）的另一个顺序的表示。在这种情况下，我们以一种新次序来列出森林的节点，这种新次序由于使每个家族的成员都一起出现，所以可以称之为“家族次序”。对于任何森林的家族次序，可以递归地定义如下：

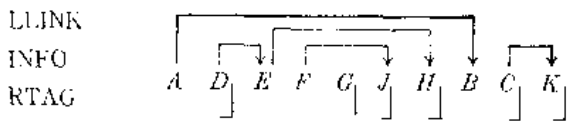
a）访问头一棵树的根；

b）遍历剩下的树（在家族次序下）；

c）遍历头一棵树之根的子树（在家族次序下）。

（把这与上一小节中的先根次序和后根次序的定义加以比较。家族次序等同于在对应的二叉树中的后根次序之逆）。

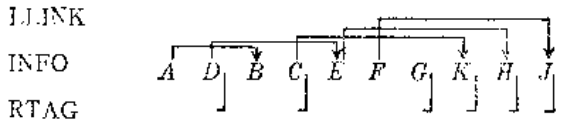
树（2）的家族次序的顺序表示是



（7）

注意，RTAG 的条款用来为诸家族划界线。家族次序从列出森林中所有树的根开始，然后继续列出诸家族，逐次地选择最近出现的节点之家族，但这种节点的家族须是尚未被列出的。由此可见，LLINK 箭头将决不交叉，而且可以类似的方式，把先根次序表示的其它诸性质照搬过来。

代替使用家族次序，我们也可以从左到右简单地列出诸节点，一次列一层。这叫做“层次次序”〔见杰·萨尔顿（G.Salton），*CACM* 5（1962），103～114〕，而且（2）的层次次序的顺序表示为



(8)

这就象是(7)，只不过诸家族的选择，乃是以先进先出的方式而不是以后进先出的方式。(7)或(8)都可认为是线性表的顺序表示对于树形的一种自然的推广。

读者将容易看出，怎样来设计遍历和分析如上顺序表示树的算法。因为，实质上，LLINK和RLINK的信息都是可资利用的，因此我们就象是已经有了一个完全地链接的树结构一样。

另一个顺序的方法，称为带次数的后根次序，与上述技术稍有不同。我们以后根次序列出节点，并且给出每个节点的次数(DEGREE)以替代链接：

DEGREE	0	0	1	2	0	1	0	1	0	3
INFO	*B*	*K*	*C*	*A*	*H*	*E*	*J*	*F*	*G*	*D*

(9)

这就足以表征树的结构。关于它的证明，请见习题2.3.2-10。这种次序，对于计算某些定义在一树之节点上的函数值，是有用的。例如，可见于下列的算法。

算法F(计算一个局部地定义于一树中的函数之值) 假设f是树节点的这样一个函数，即f在一个节点x上的值，仅仅依赖于x以及f在x的儿子处的值。下列算法，使用了一个辅助堆栈，它求出f在一个非空的森林中每个节点处的值。

F1.〔初始化〕置堆栈为空，并命P指向该森林在后根次序下的第一个节点。

F2.〔计算f〕置$d\leftarrow$DEGREE(P)(头一次到达这一步骤时，d将为0。一般地说，当我们到达这一步时，总有下述事实成立：即是堆栈顶上的d个条款是$f(x_d)$，…，$f(x_1)$——从栈顶往下——其中x_1，…，x_d是NODE(P)的从左到右数的儿子)。利用在堆栈中已找到的$f(x_d)$，…，$f(x_1)$的值，计算f(NODE(P))。

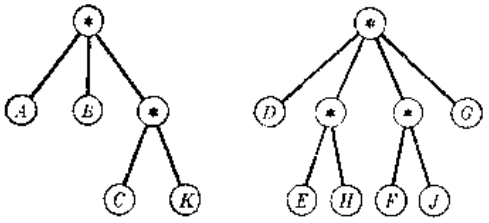
F3.〔更新堆栈〕撤消堆栈顶上的d个条款，然后把值f(NODE(P))放到堆栈顶上。

F4.〔前进〕如果P是在后根次序下的最后一个节点，则算法终止(于是从顶到底，堆栈包含着$f(\mathrm{root}(T_m))$，…，$f(\mathrm{root}(T_1))$，其中T_1，…，T_m是给定森林的树)。否则置P$\leftarrow$P\$〔在表示(9)中，简单地说这就是P$\leftarrow$P+1〕，并返回步骤F2。 ■

通过对被处理树形的大小使用归纳法，即可证明算法F的正确性(见习题17)。本算法与上一小节的微分算法(2.3.2D)有显著的类似性，微分算法计算的是一个密切相关类型的函数之值(见习题3)。在后缀记号下计算算术表达式之值的许多解释程序都使用了这同一个思想；在第8章中，我们将回过头来讨论这一课题。也可参看习题18，它给出了另一个类似算法F的重要过程。

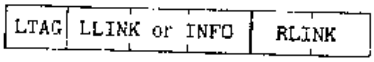
至此，我们已经看到了树和森林的各种各样顺序的表示。还有许许多多链接的表示形式，这是我们现在需要来考虑的。

头一个想法，与从(5)到(6)的变换有关：我们从所有非终端节点撤消INFO场，并且把这个信息作为一个新的终端节点放在原先节点的下面。例如，树(2)将变成



(10)

这种新的形式说明，我们可以假定（不失一般性）一棵树结构中的所有 INFO 全都出现于它的终端节点中。因此，在 2.3.2 小节的自然的二叉树表示中，LLINK 与 INFO 场是互斥的，而且它们在一个节点中可共享同一个场。一个节点可能有场



其中符号 LTAG 指示第二个场是否为一链接（请把这种表示比如说与 2.3.2 小节中（10）的双字格式进行比较）。通过把 INFO 从 6 个字节截成 3 个字节，我们就可以把每个节点填入到一个字中。然而要注意现在有着 15 个节点，而不是 10 个节点；森林（10）要占用内存的 15 个字，而（2）则要占用 20 个字；而且，后者有 60 个 INFO 字节，前者相应地则是 30 个字节。除非过多的 INFO 空间确已造成浪费，(10) 在内存空间上并没有赢得真正的权益；在（10）中 LLINK 被取代了，这是以在增加的节点中添加几乎同样数量的 RLINK 为代价而被撤消的。在习题 4 中，讨论了这两种表示之间差别的精确细节。

在树的标准二叉树表示中，LLINK 场可更精确地称作“LSON”场，因为它是从一父亲节点指向他最左边的儿子（*Leftmost* SON）的。通常，在树形中最左边的儿子是“最年青”的儿子，因此在一个家族的左边插入一节点要比在右边插入容易；所以，“LSON”这一缩写也可以想象成“最后的（*Last*）儿子”或“最小的（*least*）儿子”。

树结构的许多应用，还颇为经常地要求在树中作向上的访问，就象要求作向下的访问一样。穿线的树给了我们以向上进行的能力，但速度不快；有时候，在每一节点中有第三个链接 FATHER 就更好了。这就导致了三重链接的树，其中每个节点有着 LSON, RLINK，和 FATHER 等链接。图 26 示出了（2）的一个三重链接的树表示。关于使用三重链接的树的例子，请见 2.4 节。

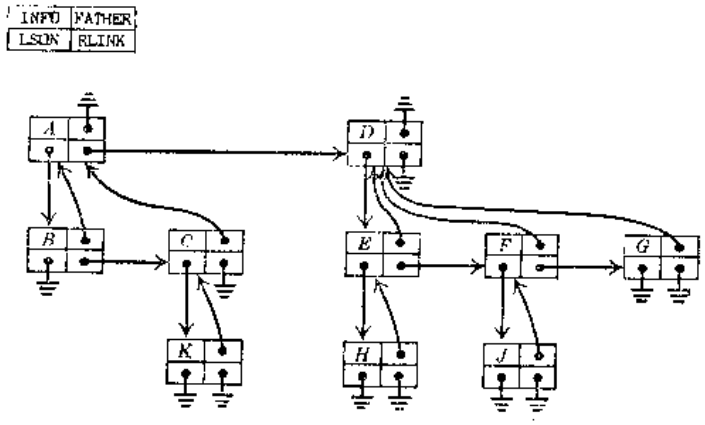


图26 三重链接的树

显然，FATHER 链接本身，独自就足以完全地确定任何有向树（或森林）。因为，如果我们知道了全部向上的链接，则我们就能画出这树的图式。除了根之外，每一个节点恰好都有一个父亲，但却可能有好几个儿子。所以给出向上的链接，比给出向下的链接要更为简单。那么，为什么在我们的讨论中，并未认为向上的链接更容易得多呢？回答当然是，向上的链接在大多数情况下是难以适用的，因为它很难迅速地告知，一个节点是否是终端节点，或者迅速地定出一个节点的任何一个儿子之位置，等等。然而，有一种非常重要的应用，对它来说，仅仅向上的链接就足够了；我们现在就着手来简略地研究一个精采的用于处理等价关系的算法，这个算法是由米·约·费希尔（M. J. Fischer）和伯·艾·加勒（B. A. Galler）给出的。

一个等价关系 "≡"，就是一个在对象的集合 S 的元素之间的关系，它对于 S 中的任何对象 x，y，和 z（不必是不同的），满足下列三个性质：

i）如果 $x \equiv y$ 且 $y \equiv z$，则 $x \equiv z$。（传递性）

ii）如果 $x \equiv y$，则 $y \equiv x$。（对称性）

iii）$x \equiv x$。（反身性）

（请把这与 2.2.3 小节中的"部分次序"关系之定义加以比较；等价关系十分不同于部分次序，尽管三条定义性质中竟有两条是相同的）。等价关系的例子有：关系 "="，对于整数的同余关系（modulo m），如 2.3.1 小节所定义的在树之间的相似关系，等等。

等价性问题就是：输入等价的元素对，然后来确定：根据那些已给的等价对，能否证明两个具体的元素是等价的。例如，假设 S 是集合 {1，2，3，4，5，6，7，8，9}，并假设我们已给了等价对：

$$1 \equiv 5,\quad 6 \equiv 8,\quad 7 \equiv 2,\quad 9 \equiv 8,\quad 3 \equiv 7,\quad 4 \equiv 2,\quad 9 \equiv 3 \tag{11}$$

由此推出，比如，$2 \equiv 6$，因为 $2 \equiv 7 \equiv 3 \equiv 9 \equiv 8 \equiv 6$。但我们不能得出 $1 \equiv 6$。事实上，(11)中的诸对把 S 分成了两类

$$\{1, 5\}\text{和}\{2, 3, 4, 6, 7, 8, 9\} \tag{12}$$

两个元素是等价的，当且仅当，它们是属于同一个类。不难证明：任何等价关系，把它的集合 S 划分成一些不相交的类（称为"等价类"），使得两个元素等价当且仅当它们属于同一个类。

因此，为解决等价性问题，无非就是要记住象(12)那样的等价类。我们可以从每个元素都独自在它的类中开始，于是：

$$\{1\}\ \{2\}\ \{3\}\ \{4\}\ \{5\}\ \{6\}\ \{7\}\ \{8\}\ \{9\} \tag{13}$$

现在如果我们给了关系 $1 \equiv 5$，则我们就拼拢{1，5}而成为一个类。在处理了头三个关系 $1 \equiv 5$，$6 \equiv 8$，和 $7 \equiv 2$ 之后，我们就把(13)改变成为

$$\{1, 5\}\ \{2, 7\}\ \{3\}\ \{4\}\ \{6, 8\}\ \{9\} \tag{14}$$

再利用等价对 $9 \equiv 8$ 来拼拢{6，8，9}，等等。

问题是，如何找出一个好方式，使在计算机内，来表示象(12)，(13)，和(14)这样的各种情况，并能使我们有效地实施下列操作：把一些类合并在一起，以及，测试两个给定的元素是否在同一个类中。以下的算法为此使用了树结构：S 的元素变成了森林的节点；而且根据迄今输入的等价对来判断，两个节点若是等价的，当且仅当它们属于同一个树。

这种测试是容易进行的，因为两个元素在同一个树中，当且仅当，它们是在同一个根元素之下。而且，通过简单地把一个树加入另一个树，以作为另一个树根的一个新的子树，就很容易把两个树合并成一个。

算法 E（处理等价关系） 设 S 是数{1，2，…，n}的集合，并设FATHER[1]，FATHER[2]，…，FATHER[n]都是整数变量。这个算法输入如同(11)那样的一组关系，并调整FATHER表格以表示一组树，使得两个元素就所给的关系说来是等价的，当且仅当它们属于同一个树形（注意，在更为一般的情况下，S的元素将是符号名称，而不是简单的从1到n的数；然后，如同在第6章中给出的一个检索程序，将设置对应于S的元素的诸节点，而且FATHER将是每个节点中的一个场。对于这种更为一般的情况的修改，是直接了当的）。

E1.〔初始化〕 置FATHER[k]←0对于$1 \leqslant k \leqslant n$〔这意味着，所有树开始时全都仅由一个根独自组成，如同(13)那样〕。

E2.〔输入新对〕 从输入取得下一对等价的元素“$j \equiv k$”。如果输入已穷尽，则算法终止。

E3.〔找根〕 如果FATHER[j]>0，则置j←FATHER[j]并重复这一步骤。如果FATHER[k]>0，则置k←FATHER[k]并重复这一步骤（经这一操作之后，j和k已被移到正待使其成为等价的两个树的根上．输入关系“$j \equiv k$”是多余的，当且仅当，我们现在有$j = k$）。

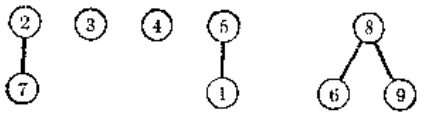
E4.〔合并树〕 如果$j \neq k$，则置FATHER[j]←k。返回步骤E2。∎

读者应对照输入(11)来试验这个算法。在处理了$1 \equiv 5$，$6 \equiv 8$，$7 \equiv 2$，和$9 \equiv 8$之后，我们将有

FATHER[k]:	5	0	0	0	0	8	2	0	8
k:	1	2	3	4	5	6	7	8	9

(15)

它表示树



(16)

在这一步之后，(11)剩下的关系将是更为有趣的；见习题9。

这类等价性问题，出现在许多应用之中。习题11中讨论了这类问题的更为一般的形式，当编译程序处理象FORTRAN这样的语言中的“等价说明”时，就出现了这种情况。

在计算机内存中表示树，还有其它种种方法。回顾我们在2.2节中，曾讨论过三个主要的表示线性表的方法：使用终端链接A的“直线的”表示，“循环地”链接的表，以及“双重”链接表。在2.3.1小节中所述的非穿线的二叉树表示，就相当于，LLINK和RLINK两者的“直线的”表示。通过在LLINK和RLINK方向上，独立地使用这三个方法当中的任何一个，就可能得到八个其它的二叉树表示。例如，图27示出了，当我们把循环链接用于两个方向上时，我们所能得到的情况。如果就象该图形这样，整个地使用循环的链接，那么就产生了一种所谓环形结构。在一些应用中，已经证明环形结构是十分灵活的。表示的选择是否得当，通常取决于在处理这些结构的算法中所需要的插入，删去，

和遍历之类型。读者既然已经考察了这一章中迄今所给出的例子，就会毫无困难地了解应何时使用和如何来处理这些内存表示中的任何一种。

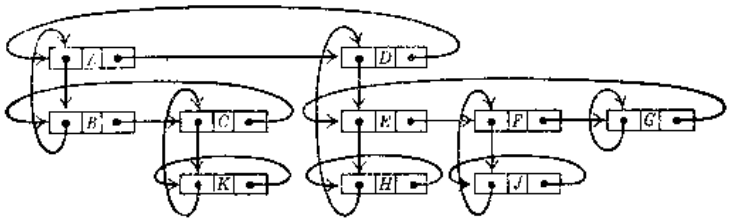


图27　环形结构

让我们用一个例子来结束这一小节。这个例子就是把修正后的双重链接的环形结构，应用到以前曾经考虑过的一个问题：关于多项式的算术。设给定了两个表示成循环表的多项式，算法2.2.4A实现了把一个多项式加到另一个多项式上去；在那一小节里，还给出了对多项式的其它运算的算法；然而，多项式限于至多三个变量。当涉及多个变量的多项式时，通常以树的结构来代替线性表更为恰当。

一个多项式或者是一个常数，或者有形式

$$\sum_{0\le j\le n} g_j x^{e_j}$$

其中 x 是一个变量，$n>0$，$0=e_0<e_1<\cdots<e_n$，而且 g_0，…，g_n 是仅含按字母次序小于 x 的那些变量之多项式；g_0，…，g_n 皆非 0。多项式的这个定义适用于如图 28 所指出的树表示。节点有六个场，在 MIX 的情况下，它可以填入到三个字中：

+	0	LEFT	RIGHT
+	EXP	UP	DOWN
CV			

(17)

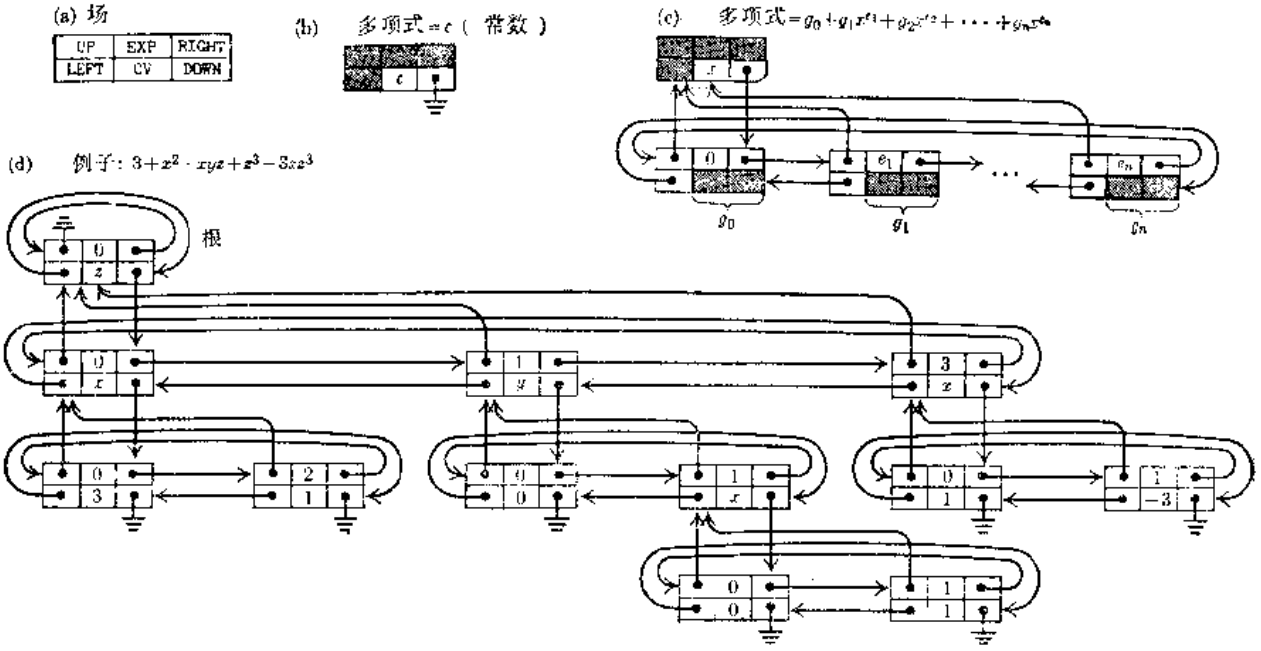


图28　利用四个方向的链接来表示多项式。节点的阴影部分表示与所考虑之上下文不相干的信息

这里 LEFT，RIGHT，UP，和 DOWN 是链接，EXP 是一个表示指数的整数，而 CV 或者是一个常数（系数）或者是一个变量的字符名称。

下列算法说明了，在这样一个四向链接的树形中之遍历，插入，和删去，所以它值得认真地加以研究。

算法 A（多项式的加法） 这个算法把多项式（P）加到多项式（Q）上，假定 P 和 Q 是指针变量，它们链接到如图 28 所示形式的不同的多项式树根上。在算法结束时，多项式（P）将不变，而多项式（Q）将包含和式。

A1.〔测试多项式类型〕 如果 DOWN(P)=Λ（即，如果 P 指向一个常数），则置 Q←DOWN(Q) 0 次或多次，直到 DOWN(Q)=Λ 为止，并转到 A3。如果 DOWN(P)≠Λ，那么，如若 DOWN(Q)=Λ 或如若 CV(Q)<CV(P)，则转到A2。否则如若 CV(Q)=CV(P)，则置 P←DOWN(P)，Q←DOWN(Q) 并重复这一步骤；如若 CV(Q)>CV(P)，则置 Q←DOWN(Q) 并重复这一步骤（步骤 A1 或许找到了多项式的两个匹配项，否则，便确定出必须把一个新变量插入到多项式(Q)的这一部分）。

A2.〔向下插入〕 置 R⇐AVAIL，S←DOWN(Q)。如果 S≠Λ，则置 UP(S)←R，S←RIGHT(S)，而且如果 EXP(S)≠0，则重复这一操作直到 EXP(S)=0 为止。置 UP(R)←Q，DOWN(R)←DOWN(Q)，LEFT(R)←R，RIGHT(R)←R，CV(R)←CV(Q)，以及 EXP(R)←0。最后，置 CV(Q)←CV(P) 以及 DOWN(Q)←R，并返回到 A1（我们在 NODE(Q)的紧下边已经插入了一个“虚拟”0 多项式，以便与在 P 的树内找到的对应多项式相匹配。在这个步骤中所完成的链接处理，是直接了当的，而且，利用“之前和之后”的图式，如同 2.2.3 小节所阐明的那样，很容易导出）。

A3.〔找到匹配项〕（这时，P 和 Q 指向给定多项式的对应项，所以已准备好进行加法）。置 CV(Q)←CV(Q)+CV(P)。如果这个和式为 0，以及如果 EXP(Q)≠0，则转到A8。如果 EXP(Q)=0，则转到 A7。

A4.〔向左推进〕（在成功地加了一项后，我们来找要相加的下一项）。置P←LEFT(P)。如果 EXP(P)=0，则转到 A6。否则置Q←LEFT(Q) 一次或多次，直到 EXP(Q)≤EXP(P)为止。然后如果 EXP(Q)=EXP(P)，则返回到步骤 A1。

A5.〔插入到右边〕 置 R⇐AVAIL。置 UP(R)←UP(Q)，DOWN(R)←Λ，LEFT(R)←Q，RIGHT(R)←RIGHT(Q)，LEFT(RIGHT(R))←R，RIGHT(Q)←R，EXP(R)←EXP(P)，CV(R)←0，以及 Q←R。返回步骤 A1（发现有必要在当前的行里，插入新的一项到 NODE(Q) 的紧右边，为的是匹配多项式（P）中对应的指数。如同在步骤 A2 中那样，“之前和之后”图式使上述操作一目了然）。

A6.〔向上转〕（多项式（P）的一行，现在已经完全遍历）。如果 UP(P)=Λ，则转到 A11；否则置 P←UP(P)。

A7.〔把 Q 上移到适当的层次〕 如果 UP(P)=Λ，则转到 A11。否则置Q←UP(Q) 0 次或多次直到 CV(UP(Q))=CV(UP(P))。返回步骤 A4。

A8.〔删去 0 项〕 置R←Q，Q←RIGHT(R)，S←LEFT(R)，RIGHT(S)←Q，LEFT(Q)←S，以及 AVAIL⇐R（出现消去，所以删去多项式（Q）的一个行元素）。如果现在 EXP(LEFT(P))=0 以及 Q=S，则转到 A9；否则返回 A4。

A9.〔删去常数多项式〕（由于消去，致使一个多项式约简成一个常数，所以删去多项式（Q）的一行）。置 R←Q，Q←UP(Q)，DOWN(Q)←DOWN(R)，CV(Q)←CV(R)，以及 AVAIL⇐R。置S←DOWN(Q)；如果S≠A，则置 UP(S)←Q、S←RIGHT(S)，而且如果EXP(S)≠0，则重复这一操作直到最终 EXP(S)= 0 为止。

A10.〔发现了0？〕 如果 DOWN(Q)= A，CV(Q)= 0，以及EXP(Q)≠0，则置P← UP(P) 并转到 A8；否则转到 A6。

A11.〔终止〕 置Q← UP(Q) 0 次或多次直到 UP(Q)= A 为止（于是把Q引导到树根上）。 ∎

如果多项式（P）有较少的项，而多项式（Q）有更多的项，那么这个算法实际上要比算法 2.2.4A 快得多，因为在加法过程中不必通过整个多项式（Q）。读者将发现，亲手来模拟算法A，把多项式 $xy-x^2-xyz-z^3+3xz^3$ 加到图 28 中的多项式上，是颇有启发的（这一情况并不是揭示算法A的效率，而是通过示明必须加以处理的困难情况，使得这个算法走遍了它的所有步骤）。关于算法A进一步的说明，请看习题 12 和 13。

我们在这里并不打算声称，图 28 中所示的表示，对于多个变量的多项式是"最好的"；在第 8 章中，我们将考虑多项式表示的另一种格式，和使用一个辅助堆栈的算术算法合在一起。这种算术算法与算法A相比，有概念简单的最大优点。我们对算法A的主要兴趣在于，它表达了对有多个链接的树进行处理的途径。

习题

▶1.〔20〕 如果我们在象（8）那样的层次次序的顺序表示中，仅有LTAG，INFO，以及 RTAG 场（没有 LLINK），则是否有可能重新构造 LLINK？（换言之，是否 LLINK 在（8）中是多余的，就象 RLINK 在（3）中那样？）

2.〔22〕（伯克斯（Burks），沃伦（Warren），和赖特（Wright），Math. Comp。8(1954)，46～50）以带次数的先根次序存储树(2)，将是

DEGREE	2	0	1	0	3	1	0	1	0	0
INFO	A	B	C	K	D	E	H	F	J	G

〔参照(9)，其中使用了后根次序〕。试设计一个类似于F的算法，通过在这个表示中从右到左地进行，来计算一个局部地定义的节点函数之值。

▶3.〔24〕 修改算法 2.3.2D，使得它遵循算法F的思想，但不象在步骤D3 中所做的那样，以不规则的方式来记录它们的位置，而是把它计算的导数作为中间结果放到堆栈上（参考习题 2.3.2-21）。该堆栈可以通过在每个导数的根中，使用 RLINK 场加以维护。

4.〔18〕 树(2)包含 10 个节点，其中有 5 个是终端的。以通常的二叉树方式表示这些树，就含有 10 个 LLINK 场及 10 个 RLINK场（每个节点一个）。这些树以形式（10）表示，其中 LLINK 和 INFO 在一个节点中共享同一个空间，则要求 5 个 LLINK 和 15 个 RLINK。在每种情况下都有 10 个 INFO 场。

给定具有 n 个节点的一个森林，其中有 m 个是终端节点。试比较使用这两种树表示方法所必须存储的 LLINK 和 RLINK 的总数。

5.〔16〕 如图26中所示的一个三重链接的树，在每个节点中含有 FATHER，LSON，

以及 RLINK 场，而且当在 FATHER，LSON，或 RLINK 场中没有提出适当的节点时，可自由地使用 A 链。试问：通过把"穿线的"链接放置到空的 LSON 和 RLINK 条款位置处，如同我们在 2.3.1 小节所做的那样，是推广这种表示成为一个穿线树的好办法吗？

▶6.〔24〕 假设一个有向森林的节点有三个链接场FATHER，LSON，以及 RLINK，但是仅有 FATHER 链接已被建立来表示该树结构。每个节点的 LSON 场都是 A，而 RLINK 场被设置成一个线性表，以某种次序简单地把节点链接在一起。链接变量 FIRST 指向头一个节点，而最后节点则有 RLINK = A。

试设计一个算法，它走遍这些节点并填入与 FATHER 链接相容的 LSON 和 RLINK 场，从而得到一个象图 26 中那样的一个三重链接的树表示。并且，恢复 FIRST使得它现在指向在这个表示中的头一个树根。

7.〔15〕 如果关系式 $9 == 3$ 在(11)中未予给出，则(12)中将出现什么类呢？

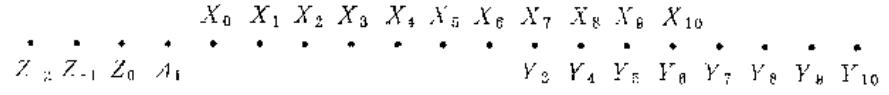
8.〔20〕 算法 E 建立了一个表示给定的等价对的树结构，但是正文中没有明确地提出如何能使用算法 E 的结果。试设计一个回答问题"$j == k$ 否？"的算法，假定 $1 \leqslant j \leqslant n$，$1 \leqslant k \leqslant n$，并假定对于某组等价式，算法 E 已经建立了 FATHER 表格。

9.〔21〕 给出一个类似于(15)的表格和一个类似于(16)的图式，指出在算法 E 从左到右地处理了(11)中的所有等价对之后，所出现的树。

10.〔25〕 在最坏的情况下，算法 E 可能花费 n^2 个步骤来处理 n 个等价式。试说明怎样来修改这个算法，使得最坏的情况不至这样糟。

▶11.〔24〕 ("等价说明")许多编译程序语言都提供了一种能把某些顺序存储的表格的内存单元，加以重叠的措施。程序员可对编译程序给出形如"X〔j〕≡Y〔k〕"的关系对，它意味着对所有的 s，变量X〔$j+s$〕将分配给与Y〔$k+s$〕相同的单元。对每个变量进一步给出可允许的下标范围："ARRAY X〔$l:u$〕"意味着在内存中将为表格条款X〔l〕，X〔$l+1$〕，…，X〔u〕腾出空间。对于变量的每一个等价类，编译程序将保留一个尽可能小的连续的内存单元区段，以容纳这些变量所允许的下标值的所有表格条款。

例如，假设我们有 ARRAY X〔0:10〕，ARRAY Y〔3:10〕，ARRAY A〔1:1〕，以及 ARRAY Z〔−2:0〕，加上等价式X〔7〕≡Y〔3〕，Z〔0〕≡A〔0〕，以及Y〔1〕≡A〔8〕。这样我们就必须为这些变量腾出 20 个连续的单元



注意单元A〔1〕之后的单元不是为诸数组中的任何一个所"允许的"下标值，但是无论如何它是被保留了。

这道题的任务是，要修改算法E，使得它可应用于刚才所述的更一般的情况。假定我们正在对这样一个语言来写编译程序，而且在我们的编译程序本身的内部表格中，对每个数组都有一个节点，包含 NAME，FATHER，DELTA，LBD诸场，以及 UBD。假定编译程序已经事先处理了所有的 ARRAY（数组）说明，使得一旦出现"ARRAY X〔$l:u$〕"以及如果 P 指向对于 X 的节点，则

$$\text{NAME}(P)=\text{"X"} \quad \text{FATHER}(P)=\Lambda \quad \text{DELTA}(P)=0$$
$$\text{LBD}(P)=l \quad \text{UBD}(P)=u$$

问题是如何设计一个处理等价说明的算法，使得在实施了这个算法之后，

FATHER(P)=A意味着：在内存中要为这个等价类保留单元X[LBD(P)]，…，X[UBD(P)]；

FATHER(P)=Q≠A意味着：单元X[k]等于单元Y[k+DELTA(P)]，其中NAME(Q)="Y"。

例如，在列出上边的等价关系以前，我们可以有节点

P	NAME(P)	FATHER(P)	DELTA(P)	LBD(P)	UBD(P)
α	X	A	0	0	10
β	Y	A	0	3	10
γ	A	A	0	1	1
δ	Z	A	0	-2	0

在处理了这些等价式之后，这些节点变成这样：

α	X	A	*	-5	14
β	Y	α	4	*	*
γ	A	δ	0	*	*
δ	Z	α	-3	*	*

("*"表示不相干的信息)。

要设计一个进行这个变换的算法。假定对你的算法的输入，有着形式(P, j, Q, k)，表示"X[j]=Y[k]"，其中NAME(P)="X"且NAME(Q)="Y"。务必要校验这些等价式是否矛盾；例如，X[1]=Y[2]与X[2]=Y[1]相矛盾。

12.〔21〕 在算法A开始时，变量P和Q指向两个树根。命P_0和Q_0表示在执行算法A之前P和Q的值。(a)在算法终止后，Q_0是否总是两个给定的多项式之和的根的地址？(b)在算法结束后，P和Q是否已恢复到它们原来的值P_0和Q_0？

▶13.〔*M*29〕 给出一个非形式的证明，证明在算法A的步骤A8开始处，我们总有EXP(P)=EXP(Q)和CV(UP(P))=CV(UP(Q))(这个事实对于真正了解该算法是很重要的)。

14.〔40〕 给出一个关于算法A的正确性的形式证明(或反驳之)。

15.〔40〕 试设计一个算法，来计算两个如图28中那样表示的多项式的乘积。

▶16.〔28〕 试设计一个这样的算法：给定了对应于先根次序的顺序表示的表格INFO1[j]，RLINK[j]，其中$1\leqslant j\leqslant n$，要形成对应于带次数的后根次序的表格INFO2[j]，DEGREE[j]，其中$1\leqslant j\leqslant n$。例如，按照(3)和(9)，你的算法应当把

j	1	2	3	4	5	6	7	8	9	10
INFO1[j]	A	B	C	K	D	E	H	F	J	G
RLINK[j]	5	3	0	0	0	8	0	10	0	0

变换成为

INFO2[j]	B	K	C	A	H	E	J	F	G	D
DEGREE[j]	0	0	1	2	0	1	0	1	0	3

17.〔*M*24〕 证明算法F的正确性。

▶18.〔25〕 算法F计算一个"由底向上"局部定义的函数的值，就是说，这种函数，在对一个节点求值之前，应该先对该节点的儿子求值。一个"由顶向下"局部定义的函数

f，则是一个这样的函数，即 f 在一个节点 x 处的值，仅仅依赖于 x 以及 f 在 x 的父亲处的值。试利用一个辅助堆栈，来设计一个类似于F的算法，它计算一个“由顶向下”的函数 f 在树的每个节点上的值（象算法F那样，你的算法应对(9)中那样以带次数的后根次序存储的树，有效地进行工作）。

19. 〔M28〕 当以随机的次序给出随机的等价对时，试对算法E的效率作一分析。特别是，在算法E已进入操作之后，节点在树中的平均层数将是多少？

2.3.4 树的基本数学性质

远在发明计算机之前，树结构就已经是多少年来广泛的数学研究的对象，而且对于它们，已经发现了许多饶有兴趣的事实。我们将在这一小节中，来综述树的数学理论。这不仅使我们对树结构的性质能有更深入的了解，而且对于计算机的算法也有重要的应用。

不搞数学的读者，我们建议他跳过前边几小节，径直来看2.3.4.5小节。这一小节讨论了，我们后边将要研究的一些应用中，经常出现的若干课题。

下边这部分材料，大多数选自称为图论的一个较大的数学领域。遗憾的是，这方面还没有标准的术语，因此作者只好遵循有关图论的现行书本上的通常做法，这就是说采用与其它关于图论的书本所使用的术语相类似但却不尽相同的名词。在下列诸小节中（事实上，贯穿全书也是如此），我们已经对一些重要的概念，力图选择简短的描述性的名词；这些名词，是从相当普遍地使用的而且与其它普通的术语没有尖锐矛盾的名词当中，挑选出来的。这里所用的专门术语是偏向于计算机的应用。一个电机工程师可能喜欢把我们称之为“自由树”的对象，径直称为“树”；但是我们却要用较为简短的“树”一词，来称呼一个已普遍地用于计算机著作中，在计算机应用中更为重要得多的概念。如果遵循图论方面某些作者的术语，我们就要说“有限的带标号的有根有序的树”而不是说“树”，或说“拓扑分叉的树木”而不是“二叉树”！

2.3.4.1 自由树 一个图形，一般地定义为一个由点（叫做顶点）组成的集合，连同一个联接某些对不同顶点的线（叫做边）的集合。至多有一条边联接任何一对顶点。两个顶点称为相邻的，如果有一条边联接着它们。如果 V 和 V' 是顶点，而且如果 $n\geq 0$，我们说 $(V_0, V_1, \cdots, V_n)$ 是一条从 V 到 V' 的长度为 n 的通路，乃指如果 $V=V_0$，对于 $0\leq k<n$，V_k 与 V_{k+1} 相邻，而且 $V_n=V'$。说该通路是简单的，是指如果 $V_0, V_1, \cdots, V_{n-1}$ 是不同的以及 $V_1, \cdots, V_{n-1}, V_n$ 是不同的。一个图形说成是连通的，指的是如果图形的任何两个顶点之间都有一条通路。一条回路是一条从一个顶点到它本身的长度为3或者更大的简单通路。

图29中解释了这些定义，它示出了一个有五个顶点和六条边的连通图形。顶点 C 与 A 相邻，但不与 B 相邻；从 B 到 C 有两条长度为2的通路，就是 (B, A, C) 和 (B, D, C)。有若干条回路，包括 (B, D, E, B) 在内。

一个自由树或“无根树”（图30），定义为一个没有回路的连通的图形。这个定义既可应用于无限的图形，也可应用于有限的图形。对于计算机的应用，我们当然是关注于有限的树形。关于定义自由树，有许多等价的方式；其中的若干个方式，出现于下列熟知的定理中：

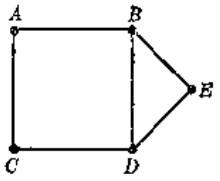


图29 图形

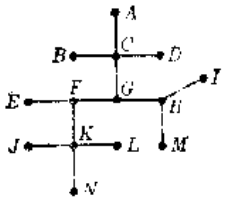


图30 自由树

定理 A 如果G是一个图形，则下列诸命题是等价的：

a）G是一个自由树。

b）G是连通的，但如果删去任何一条边，则得到的图形就不再是连通的了。

c）如果V和V'是G的不同顶点，则恰有一条从V到V'的简单通路。

而且，如果G是有限的，并恰好含有$n>0$个顶点，则以下的性质也等价于（a），（b），和（c）。

d）G不含回路而且有$n-1$条边。

e）G是连通的而且有$n-1$条边。

证明：（a）推出（b）。因为，如果删去边VV'后G仍然是连通的，则必有长度为2或更大的一条简单通路$(V, V_1, \cdots, V')$——见习题2——然则$(V, V_1, \cdots, V', V)$就将是G中的一条回路。

（b）推出（c）。因为，从V到V'至少有一条简单通路。而且如果有两条这样的通路$(V, V_1, \cdots, V')$和$(V, V_1', \cdots, V')$，则我们就能找到最小的k使得$V_k \neq V_k'$；删去$V_{k-1}V_k$边后将不能使该图形分解，因为仍将有从V_{k-1}到V_k的一条通路$(V_{k-1}, V_k', \cdots, V', \cdots, V_k)$，而它不使用这删去的边。

（c）推出（a）。因为，如果G含有一条回路$(V, V_1, \cdots, V)$，则从V到V_1就有两条简单通路。

为了证明（d）和（e）也等价于（a），（b），和（c），让我们首先证明一个辅助的结果：如果G是任何有限的图形，它没有回路但至少有一条边，那么至少有一个顶点，它恰巧邻接另外一个顶点。因为，我们取一个任意顶点V_1及一个相邻的顶点V_2；对于$k \geqslant 2$，或者V_k相邻于V_{k-1}而且不相邻于其它的，或者它相邻于一个我们可以称之为$V_{k-1} \neq V_{k-1}$的顶点。由于没有回路，所以$V_1, V_2, \cdots, V_{k+1}$必定都是不同的顶点，因此这个过程最终必然终止。

现在假定G是一个有$n>1$个顶点的树，而且命V_n仅仅相邻于另外一个顶点，即是V_{n-1}。如果我们删去V_n以及边$V_{n-1}V_n$，则剩下的图形G'是一个树，因为V_n除了作为头一个或最后一个元素外，不会出现于G的一条简单的通路中。这个论证证明（通过对n用归纳法），G有$n-1$条边，即（a）推出（d）。

假定G满足（d），且命V_n，V_{n-1}，G'就象上一段的那样。那么，图形G便是连通的，因为V_n连通到V_{n-1}，而V_{n-1}（通过对n用归纳法）连通到G'的所有其它顶点。于是，（d）推出（e）。

最后，假定G满足（e）。如果G含有一个回路，则我们可删去出现于该回路中的任

何一条边，而G仍将是连通的。因此我们就能以这种方式继续来删去各边，直到我们得到一个具有$n-1-k$条边而且无回路的连通的图形G'为止。但因为（a）推出（d），我们必须有$k=0$，即，$G=G'$。

自由树的思想可直接地应用来分析计算机的算法。在1.3.3小节里，我们曾经讨论过克希霍夫第一定律，在关于计算一算法中每一步骤所被执行的次数问题中的应用。我们已经看到，克希霍夫定律不能定全地确定每一步骤被执行的次数，但它减少了必须特殊解释的未知量的个数。树理论告诉我们，将留下多少个独立的未知量，而且给了我们一个系统的寻找它们的方法。

通过研究例子，是比较容易了解所遵循的方法的。所以，随着理论的被展开，我们将以一个例子来进行工作。图31示出了程序1.3.3A的一个抽象的框图，它从属于1.3.3小节中的"克希霍夫定律"的分析。图31中的每个方框表示计算部分，而方框内的字母或数字表示在一次运行这个程序期间，执行该计算的次数，其中使用的是1.3.3小节的记号。方框之间的箭头表示程序中一个可能的转移。这些箭头被标为$e_1, e_2, \cdots, e_{27}$。我们的目标是要找出由克希霍夫定律所能导出的，在数量A，B，C，D，E，F，G，H，I，J，K，L，P，Q，R，与S之间的所有关系，同时我们希望对一般的问题也能得到某些洞察（注意：在图31中已经作了某些简化，例如，C与E之间的方框已经标为"1"，而这事实上是克希霍夫定律的一个推论）。

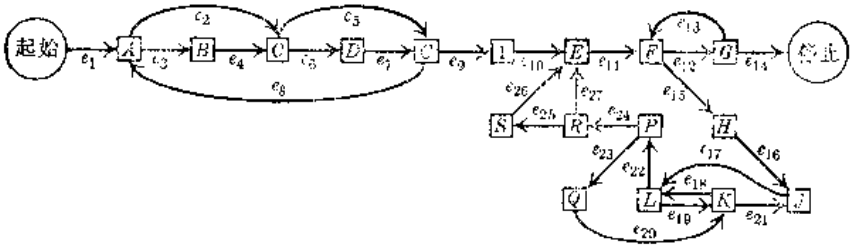


图31　程序1.3.3A的抽象框图

命E_i表示在执行所被研究的程序期间，分支e_i实行的次数；于是克希霍夫定律就是

"进入方框的诸E之和＝框中之值＝离开方框的诸E之和"　　(1)

例如，在标有K的方框的情况下，我们有

$$E_{18}+E_{20} \quad = \quad K = \quad E_{19}+E_{21} \qquad (2)$$

在以下的讨论中，我们将认为E_1，E_2，…，E_{27}是未知量，以代替A，B，…，S。

图31可以进一步进行抽象，使得它变成一个象图32那样的图形G。方框缩成为顶点，而箭头e_1，e_2，……现在表示图形的边（一个图形，严格地说，它的边是不隐含方向的，而且当我们谈及G的图论上的性质时，应当忽略箭头的方向。然而，对克希霍夫定律的应用，象我们后面将要看到的那样，就利用了箭头）。为方便起见，从"停止"顶点到"起始"顶点画了一条额外的边e_0，以使克希霍夫定律能统一地应用到图形的所有部分。图32也包括对图31的其它一些小的改动：增加了额外的顶点和边，把e_{13}分成两个部分e'_{13}和e''_{13}，以使其适合一个图形的基本定义（没有两条边联接相同的两个顶点）；e_{19}也以这种方式分开了。只要我们遇到任何带有引回到自身之箭头的顶点，则就应进行类似的修改。

图 32 中的某些边比其它边画得要粗些。这些边形成了该图形的一个连通所有顶点的自由子树。总能从框图引出的图形中找到一个自由子树。因为，这图形必然是连通的，并且由定理A的部分（b）可知，如果G是连通的，而且不是一个自由树，则我们就能删去某条边，而且得到的图形仍然是连通的；这一过程可以迭代下去，直到我们得到一子树为止。习题 9 中给出了为寻找一个自由子树的另一个算法。事实上，我们总可以首先放弃边e_0（它从“停止”向“起始”顶点进行），所以我们总可以假定e_0不出现在选择的子树中。

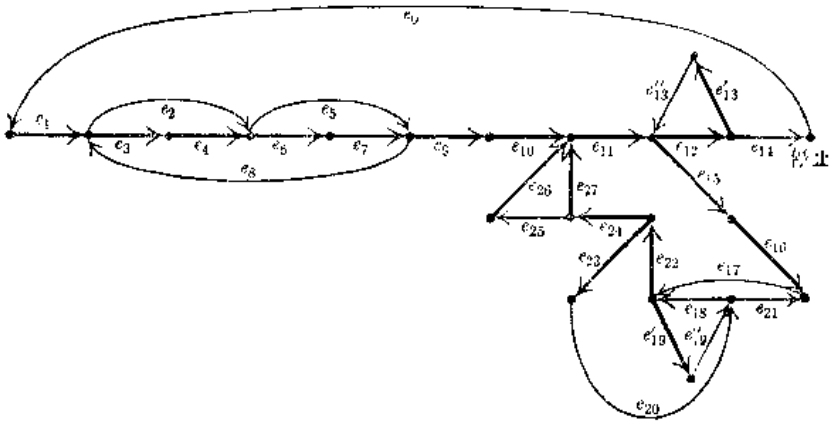


图32　对应于图31的图形，包括一个自由子树

命G'是图形G的一个以这种方式找到的自由子树，并考虑G的任何不在G'中的边VV'。我们现在可以注意定理A的一个重要推论：G'加上这条新的边VV'就包含着一条回路；而且事实上恰好有一条回路，它具有形式（V，V'，…，V），因为在G'中有一条唯一的从V'到V的简单通路。例如，如果G'是图 32 中所示的自由子树，而且如果我们加上边e_2，则我们就得到一条沿着e_2进行而后（以箭头所指的反方向）沿着e_4到e_3的回路。这条回路可以代数地写成为“$e_2-e_4-e_3$”，其中使用＋号和－号来表示回路是沿着箭头的方向还是相反的方向。

如果我们对不在自由子树中的每一条边，来进行这一过程，则我们就得到了所谓的基本回路；在图 32 的情况下，它们是

$$
\begin{aligned}
C_0&:\quad e_0+e_1-e_3+e_4+e_6+e_7+e_9+e_{10}+e_{11}+e_{12}+e_{14}\\
C_2&:\quad e_2-e_4-e_3\\
C_5&:\quad e_5-e_7-e_6\\
C_8&:\quad e_8-e_2+e_4+e_6+e_7\\
C''_{13}&:\quad e''_{13}-e_{12}+e'_{13}\\
C_{17}&:\quad e_{17}+e_{22}+e_{24}+e_{27}+e_{11}+e_{15}+e_{16}\\
C''_{19}&:\quad e''_{19}+e_{18}+e'_{19}\\
C_{20}&:\quad e_{20}+e_{18}+e_{22}+e_{23}\\
C_{21}&:\quad e_{21}-e_{16}-e_{15}-e_{11}-e_{27}-e_{24}-e_{22}-e_{18}\\
C_{25}&:\quad e_{25}+e_{26}-e_{27}
\end{aligned}
\tag{3}
$$

显然，不在自由子树中的边e_j将仅仅出现于诸条基本回路中的一条之中，即在C_j之中。

我们现在快要到这项建设的高潮了。每一条基本的回路代表着对克希霍夫定律的一个

解；例如，对应于 C_2 的解是命 $E_2=+1$，$E_3=-1$，$E_4=-1$，而所有其余诸 $E=0$。显然，在一个图形中，围绕一条回路的流量，总满足克希霍夫定律的条件（1）。而且克希霍夫方程是“齐次的”，所以对于（1）之解，作其和或差，就能得到另一个解。因此，我们可以得出这个结论，E_0，E_2，E_6，…，E_{25} 的值，在下列意义下，是独立的：

如果 x_0，x_2，…，x_{25} 是任意实数（对于不在自由子树 G' 中的每一个 e_j，各有一个 x_j），则克希霍夫方程（1）有一个解使得

$$E_0=x_0,\quad E_2=x_2,\cdots,\quad E_{25}=x_{25} \tag{4}$$

通过围绕回路 C_0 进行 x_0 次，围绕回路 C_2 进行 x_2 次，等等，即可找到这样一个解。进一步，我们推出，剩下的变量 E_1，E_3，E_4，…之值，完全依赖于 E_0，E_2，…，E_{25} 之值：

命题（4）中所述的解是唯一的 (5)

因为，如果对克希霍夫方程有两个解，使得 $E_0=x_0$，…，$E_{25}=x_{25}$，则我们就可从一个解减去另一个解并由此而得到一个解，使在其中 $E_0=E_2=E_5=\cdots=E_{25}=0$。但现在所有的 E_j 都必须为 0，因为容易看出，对于克希霍夫方程的非 0 解，当图形为一自由树时，是不可能的（见习题 4）。因此所假定的两个解必须相等。我们已经证明了克希霍夫方程的所有解，可以作为基本回路之多重的和而得到。

当把这些论述应用于图 32 时，借助于独立变量 E_0，E_2，…，E_{25}，我们就得到克希霍夫方程的下列的一般解：

$$\begin{aligned}
&E_1=E_0 && E_{14}=E_0\\
&E_3=E_0-E_2+E_8 && E_{15}=E_{17}-E_{21}\\
&E_4=E_0-E_2+E_8 && E_{16}=E_{17}-E_{21}\\
&E_6=E_0-E_5+E_8 && E_{18}=E''_{19}+E_{20}-E_{21}\\
&E_7=E_0-E_5-E_8 && E'_{19}=E''_{19}\\
&E_9=E_5 && E_{22}=E_{17}+E_{20}-E_{21}\\
&E_{10}=E_0 && E_{23}=E_{20}\\
&E_{11}=E_0+E_{17}-E_{21} && E_{24}=E_{17}-E_{21}\\
&E_{12}=E_0+E''_{13} && E_{26}=E_{25}\\
&E'_{13}=E''_{13} && E_{27}=E_{17}-E_{21}-E_{25}
\end{aligned} \tag{6}$$

为了得到这些等式，我们只要对于子树中的每一条边 e_j，来列出使 e_j 以适当符号出现在回路 C_k 中的所有 E_k〔这样，（6）中的系数矩阵就恰是（3）中的系数矩阵之转置〕。

严格说来，C_0 不应该称作一条基本回路，因为它包含特殊的边 e_0。我们可以把 C_0 减去边 e_0，称为从“起始”到“停止”的一条基本通路。我们的边界条件，即框图中的“起始”和“停止”框仅执行一次，就等价于关系式

$$E_0=1 \tag{7}$$

上边的讨论，说明怎样来得到克希霍夫定律的所有解；同样的方法也可以应用到电路中以代替程序框图（如同克希霍夫自己应用这一定律那样）。至此，自然要问，克希霍夫定律是否就是对于程序框图的情况所能给出的最有可能的方程组了，或者是否能更进一步地说：一个计算机程序从“起始”到“停止”的任何执行，对于每一边被遍历的次数，给予

了我们一组值 E_1，E_2，……，E_{27}，而且这些值遵从克希霍夫定律；但是还有没有不对应于任何计算机程序之执行的克希霍夫方程的解呢？（在这个问题上，我们不假定，除了框图以外，关于给定的计算机程序我们还知道些什么东西），如果有着满足克希霍夫条件，但不对应于实际的程序执行的解，则我们就能给出比克希霍夫定律更强的条件。对于电子线路的情况，克希霍夫自己给出了第二个定律：围绕一条基本回路的电压降之和必须为 0。这第二个定律不能应用于我们的问题。

如果诸 E 都对应于，从“起始”到“停止”的框图中的某条实际的通路，那么，显然还有它们必须要满足的更进一步的条件；它们必须是整数，而且事实上它们必须是非负整数。这并不是一个不足道的条件，因为我们并不能对独立变量 E_2，E_5，…，E_{25} 简单地赋予任意的非负整数值；例如，如果我们取 $E_2=2$ 以及 $E_8=0$，则可从（6），（7）求出 $E_3=-1$（于是，如果不是至少通过分支 e_8 一次的话，则图 31 的执行就不会两次通过分支 e_2）。所有的 E 都是非负整数的条件，也还是不够的；例如，考虑其中 $E_{19}=1$，$E_1=E_5=\cdots=E_{17}=E_{21}=E_{25}=0$ 的解；则除了经由 e_{19}，就没有办法进到 e_{18}。下列的条件是一个充分必要条件，它回答了在上一段中提出的问题：命 E_2，E_5，…，E_{25} 是任何给定的值，并按照（6），（7）来确定 E_1，E_3，…，E_{27}。假定所有的 E 是非负整数，且假定这样的图形是连通的，即它的边是使得 $E_j>0$ 的那些 e_j，而它的顶点是接触这种 e_j 的那些顶点。那么，就有一条从“起始”到“停止”的通路，其中边 e_j 恰好被遍历 E_j 次。在下一小节中（见习题 2.3.4.2-24）证明了这一事实。

现在让我们来综述上面的讨论：

定理 K 如果一个框图（如图 31 那样）包含 n 个方框（包括“起始”和“停止”）和 m 个箭头，那么，有可能找出 $m-n+1$ 条基本回路和一条从“起始”到“停止”的基本通路，使得任何从“起始”到“停止”的通路，等价于（就每边之被遍历的次数说来）这个基本通路的一次遍历，再加上这些基本回路中每一个唯一确定次数的遍历（基本通路和基本回路，可能包括某些与所标示之箭头相反的方向来遍历的边。习惯上，我们说这样的边是被遍历 -1 次）。

反之，对于基本通路和基本回路的任何这样的遍历，即其中每边被遍历的总次数是非负的，而且对应于正次数遍历的顶点和边形成一连通的图形，都至少有一条从“起始”到“停止”的等价的通路。 ∎

基本回路是通过挑出如图 32 中那样的一个自由子树来找出的；如果我们选择一个不同的子树，则一般说来，我们就得到一组不同的基本回路。有 $m-n+1$ 条基本回路这一事实，是从定理 A 得出的。我们从图 31 到图 32 所作的修改，在增加了 e_0 之后，就不再改变 $m-n+1$ 的值了，尽管它们可能增加 m 和 n 二者之值；这种作法可以加以推广，以至完全避免这些平凡的修改（见习题 8）。

定理 K 是鼓舞人心的，因为它说克希霍夫定律（它由 m 个未知数 E_1，E_2，…，E_m 的 n 个方程组成）仅有一个是“多余的”，即是，这 n 个方程允许我们来消去 $n-1$ 个未知数。然而要注意，贯穿在整个讨论中，未知变量是诸边被遍历的次数，而不是该框图的每个方框被进入的次数。习题 7 说明了怎样来构造另一个图形，它的边就对应于框图的方框，使得上述的理论可以用来导出在诸有关的变量中冗余者的真正数目。

习题

1.〔14〕试列出出现于图 29 中的从 B 到 B 的所有回路。

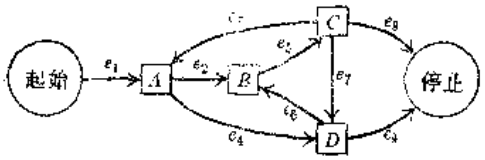
2.〔$M20$〕证明：如果 V 和 V' 是一个图形的顶点，而且如果有从 V 到 V' 的一条通路，则就有从 V 到 V' 的一条简单的通路。

3.〔15〕在图 32 中，从“起始”到“停止”的什么通路，等价于（在定理K的意义下）遍历一次基本通路加上遍历一次回路 C_2？

▶4.〔$M20$〕设 G' 是一个有限的自由树，在它的边 e_1，…，e_{n-1} 上已画出了箭头；设 E_1，…，E_{n-1} 是满足 G' 中克希霍夫定律（1）的数，试证明 $E_1=\cdots=E_{n-1}=0$。

5.〔20〕利用等式（6），借助于独立变量 E_2，E_5，…，E_{35} 来表达出现于图 31 的方框里边的量 A，B，…，S。

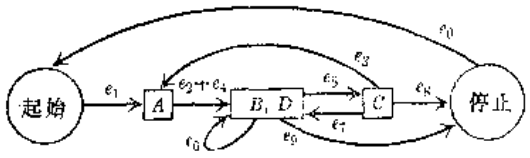
6.〔22〕试利用由边 e_1，e_2，e_3，e_4，e_9 所组成的自由子树，对框图



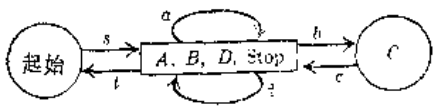
实施正文中的那种构作。什么是基本回路？借助于 E_5，E_6，E_7，和 E_8，来表达 E_1，E_2，E_3，E_4，E_9。

▶7.〔$M25$〕当克希霍夫第一定律用于程序框图时，我们通常仅仅是对顶点的流量（即框图的每一方框被执行的次数）感兴趣，而不是对正文中所分析的边的流量感兴趣。例如，在习题 6 的图形中，顶点的流量是 $A=E_2+E_4$，$B=E_5$，$C=E_3+E_7+E_8$，$D=E_6+E_9$。

如果我们把某些顶点合并在一起，把它们当作是一个“超顶点”来处理，我们就可以把对应于同一个顶点流量的边流量组合在一起。例如，若把顶点 B 与 D 合并在一起，则在上述框图中，边 e_5 与 e_4 就组合在一起了：



（这里，e_9 也已象在正文中那样，从“起始”到“停止”地被加上）。继续这一过程，我们可以组合 e_3+e_7，然后 $(e_3+e_7)+e_8$，然后 e_6+e_9，直到我们得到一个既约的框图，有边 $s=e_1$，$a=e_2+e_4$，$b=e_5$，$c=e_3+e_7+e_8$，$d=e_6+e_9$，$t=e_0$，即对于原来框图中的每个顶点恰好有一条边：



由作法看出，在既约的框图中，克希霍夫定律保持成立。新的边流量就是原来的顶点流量；因此，正文中的分析，可应用于既约的框图，来说明原来的顶点流量如何地彼此相关。

证明这个约化过程可以倒过来，在这样的意义下，即，在既约框图中满足克希霍夫定律的任何一组流量$\{a, b, \cdots\}$，可以“分开”成为原来的框图中的一组边流量$\{e_0, e_1, \cdots\}$。这些流量e_j满足克希霍夫定律并且组合起来可得到所给的流量$\{a, b, \cdots\}$；然而，它们之中的某些个可以为负（尽管在这里仅仅对一个具体的框图说明了约化过程，但你的证明应该是普遍地正确的）。

8.〔M22〕边e_{13}和e_{18}在图32中被分成为两部分，因为一个图形不允许有两条边联接同样的两个顶点。然而，如果我们来观察这种构造的最后结果，则分成两部分就显得非常不自然，因为$E'_{13}=E''_{13}$，$E'_{18}=E''_{18}$是在（6）中求得的两个关系，而E''_{13}，E''_{18}是两个独立变量。试阐明，怎样来推广这种构作，使得可以避免这种不自然地分开边的做法。

▶9.〔M27〕假设一个图形有n个顶点V_1，…，V_n和m条边e_1，…，e_m。每一条边由一对整数(a_k, b_k)来表示，它们给出被这条边造成相邻的顶点的号数。试设计一个算法，它接收输入对(a_1, b_1)，…，(a_m, b_m)打印出这些数对中形成一个自由树的一个子集；如果不可能，该算法要报告失败。要求想出一个高效的算法。

10.〔16〕一位为计算机设计线路的电子工程师发现，他有n个端点$T_1, T_2, \cdots, T_n$要求在所有时刻它们基本上都有相同的电压。为了实现这一点，他可以在任何一对端点之间焊接导线；这个思想就是要用足够的导线连接，使得从任何端点到其它的端点，都有一条经由导线的通路。试说明，为连接所有端点所需要的导线条数最少是$n-1$，而且用$n-1$条导线能实现所希望的连接当且仅当它们形成一个自由树时（把端点和导线理解成顶点和边）。

11.〔M27〕（罗·克·普赖姆（R. C. Prim），《贝尔系统技术杂志》（Bell System Tech. J.）36.1957, 1389～1401）考虑习题10的导线连接问题，今附加条件：对于每对$i<j$，给出一个代价$c(i, j)$，表示从端点T_i到T_j的导线的开销。试说明，下列的算法给出了一个最小代价的连通树：“如果$n=1$，则什么也不干。否则，重新对端点和代价编号，使得$c(n-1, n)=\min_{1\leqslant i<n} c(i, n)$；连通端点$T_{n-1}$到$T_n$；然后对于$1\leqslant j<n-1$，把$c(j, n-1)$变成$\min(c(j, n-1), c(j, n))$，并且利用这些新的代价，对$n-1$个端点$T_1$，…，$T_{n-1}$重复这个算法（本算法在下述条件下进行重复：每当随后要求在现在称为T_j与T_{n-1}的端点之间进行连通时，那么，如果在现在称为T_j与T_n的端点之间的连通更为便宜的话，则实际上就进行后一个连通。这样，在算法的剩下部分，T_{n-1}和T_n就被认为一个端点了）”。这个算法也可陈述如下：“从选定一个具体的端点着手；然后重复地做出从一个未选端点到一个已选端点之间可能最便宜的连通，直到全都选完为止”。

例如，考虑图33(a)，它示出在一个格子上的九个端点；设连通两个端点的代价是导线的长度，即是，它们之间的距离（读者不妨试验用手工来找极小代价的树，用直觉代替上述算法）。这个算法将首先连通T_8到T_9，然后T_6到T_8，T_5到T_8，T_2到T_6，T_1到T_2，T_3到T_1，T_7到T_3，而最后T_4或者到T_2或者到T_6。图33(b)示出了一个极小代价的树（导线长度为$7+2\sqrt{2}+2\sqrt{5}$）。

▶12.〔29〕习题11的算法不是以适宜于计算机直接实现的方式来叙述的。试以计算机程序能以相当的效率来实现这一过程的方式，重新陈述该算法。更加详尽地描述有待完成的操作。

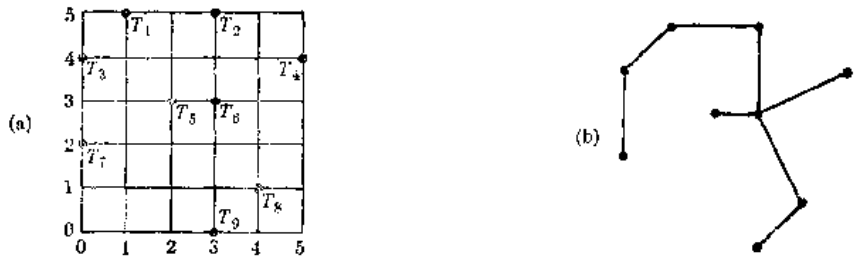


图33 最小代价的自由树(见习题11)

13. 〔*M*20〕设G是一个图形（可能是无限的），它不包含回路，但是如果把在G中不曾出现的任何边加到G上，则形成一条回路。试证明G是一个自由树。

14. 〔*M*24〕考虑一个有n个顶点和m条边的图形，采用习题9的记号。试证明：有可能把整数$\{1, 2, \cdots, n\}$的任何置换写成对换之乘积$(a_{k_1} b_{k_1})(a_{k_2} b_{k_2})\cdots(a_{k_t} b_{k_t})$，当且仅当，图形是连通的（因此，存在着由$n-1$个对换组成的集合，它生成$n$个元素的所有置换，但是由$n-2$个对换组成的集合将不能做到这一点）。

***2.3.4.2 有向树** 在上一小节中，我们看到，如果忽略框图中边上的箭头方向，则一个抽象的框图可以认为是一个图形。我们已经看到，“回路”和“自由子树形”等等图论的思想，关系到框图的研究。当每条边的方向给出更重要的意义时，要说的就更多了，在这种情况下，我们就有了所谓的“有向图形”或“图型”。

现在让我们形式地定义有向图形为：一个由顶点组成的集合和一个由弧组成的集合，每条弧从一顶点V引到一顶点V'。如果e是从V到V'的一条弧，则我们就说V是e的初始顶点，而V'是最终顶点，而且我们写$V=\text{init}(e)$，$V'=\text{fin}(e)$。不排除$\text{init}(e)=\text{fin}(e)$的情况（尽管在原先的图形中，它按边的定义是被排除的），而且若干不同的弧可以有相同的初始顶点和最终顶点。一个顶点V的输出次数是由它引出的弧的条数，即，使得$\text{init}(e)=V$的弧e的条数；类似地，V的输入次数是有$\text{fin}(e)=V$的弧数。

对于有向图形的通路，回路，等等概念，以对于普通图形的相应的定义，类似地定义之，但有某些必须考虑的重要的新专门术语。假设$e_1, e_2, \cdots, e_n$是弧（其中$n \geqslant 1$），则我们说$(e_1, e_2, \cdots, e_n)$是一条从V到V'的长度为n的有向通路，如果$V=\text{init}(e_1)$，$V'=\text{fin}(e_n)$，而且$\text{fin}(e_k)=\text{init}(e_{k+1})$对于$1 \leqslant k < n$。一条有向通路$(e_1, e_2, \cdots, e_n)$称作是简单的，如果$\text{init}(e_1), \cdots, \text{init}(e_n)$是不同的，而且$\text{fin}(e_1), \cdots, \text{fin}(e_n)$也是不同的。一条有向回路是从一个顶点到它自身的一条简单有向通路（注意一条有向回路能有长度1或2，但这在我们上一小节的“回路”定义中，是被排除的。读者能看出吗，这为什么讲得通？）。

作为这些直接了当定义的例子，我们可以参考上一小节的图31。标号“J”的方框是一个有着输入次数2（因为弧e_{16}，e_{21}）和输出次数1的顶点。序列$(e_{17}, e_{19}, e_{18}, e_{22})$是一条从J到P的长度为4的有向通路。这条通路不是简单的，因为，例如，$\text{init}(e_{19})=L=\text{init}(e_{22})$。该图式不包含长度为1的有向回路，但是$(e_{18}, e_{19})$是一条长度为2的有向回路。

一个有向图形说是强连通的，如果对于任何两个顶点$V \neq V'$，都有一条从V到V'的有向通路。有向图形说是有根的，如果至少有一个"根"，即，至少有一个顶点R，使得对于所有的$V \neq R$，从V到R都有一有向通路("强连通"总是意味着"有根"，但反之不成立。在上一小节中，如象图31那样的一个框图是有根的有向图形的一个例子，以R为"停止"顶点；通过从"停止"到"起始"的附加弧（图32），它就变成强连通的了)。

每一个有向图形G以一个明显的形式对应于普通的图形G_0，其中G_0有一条从V到V'的边，当且仅当，$V \neq V'$而且G有一条从V到V'或从V'到V的弧。通过把G中的通路和回路理解成G_0的通路和回路，我们可以谈论G中的（无向的）通路和回路。我们可以说，G是连通的（这是比"强连通"要弱得多，甚至比"有根"还要弱的性质)，如果对应的图形G_0是连通的，等等。

一个有向树（见图34)，其他作者有时称之为"有根树"，是这样一个有着一个确定的顶点R的有向图形：

a）每个顶点$V \neq R$恰是以$e(V)$表示的一条弧的初始顶点；

b）R不是任何弧的初始顶点；

c）R在上述定义的意义下是一个根（即，对于每一个顶点$V \neq R$，有一条从V到R的有向通路)。

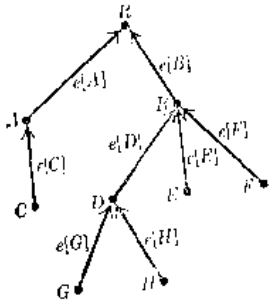


图34 有向树

立即可以推出，对于每一顶点$V \neq R$，从V到R有一条唯一的有向通路；而且，因此没有有向回路。

当有有限多个顶点时，容易看出，我们以前的"有向树"的定义(在2.3小节开始处)，是与刚才给出的新定义相一致的；顶点就对应于节点，而且弧$e(V)$就是从V到FATHER(V)的链接。

由性质（c)，对应于一个有向树的（无向）图形是连通的。而且，倘若$(V_0, V_1, \cdots, V_n)$是一条有$n \geqslant 3$的回路，而且如果V_0与V_1之间的边是$e(V_1)$，则V_1与V_2之间的边必然是$e(V_2)$，而且类似地对于$1 \leqslant k \leqslant n$，$V_{k-1}$与$V_k$之间的边必然是$e(V_k)$，这同不存在有向回路相矛盾。如果$V_0$与$V_1$之间的边是$e(V_0)$，则对于回路

$$(V_1, V_0, V_{n-1}, \cdots, V_1)$$

也可应用同样的论证。因此，没有回路；当忽略弧的方向时，一个有向树是一个自由树。

反之，重要的是要注意到，我们可以把刚才描述的过程逆过来。如果我们从任何非空的自由树形开始，例如在图30中的那样，则我们可以选择任何顶点作为根R，而且对这些边赋予方向。直观的思想是在顶点R处"提起"这图形并抖动它；然后赋予向上指的箭头。更形式地，这个规则是：

> 把边VV'改变成一条从V到V'的弧，当且仅当，从V到R的简单通路经由V'进行，即，当它有$(V_0, V_1, \cdots, V_n)$的形式时，其中$n > 0$，$V_0 = V$，$V_1 = V'$，$V_n = R$。

为了验证这样的构造是正确的，我们需要证明，每条边皆已赋予方向$V \leftarrow V'$或$V \rightarrow V'$；这是容易证明的，因为如果$(V, V_1, \cdots, R)$和$(V', V_1', \cdots, R)$是简单通路，则将有一条回路，除非$V = V_1'$或$V_1 = V'$。作为这个构造的一个推论，我们得出，在一个有

向树形中，诸弧的方向，只要知道哪一个顶点是根，就完全确定了，所以当已明确地指出根时，它们已不必在图式中示出。

现在我们来看三种类型的树之间的关系：（有序）树，就象在2.3小节中所定义的那样，它在计算机程序中是最为重要的；有向树（即无序树）；以及自由树。后两种类型的树是在研究计算机算法中出现的，但不及头一种类型那样经常。在这些类型的树结构之间，其实质性的差别仅仅在于：其所顾及的信息之总量。例如，图35示出三个不同的树，如果把它们认为是有序树的话（根在顶上）。作为有向树，则头一个与第二个是相等的，因为子树从左到右的次序是无所谓的；作为自由树，图35中的所有三个图形都是相同的，因为根是无所谓的。

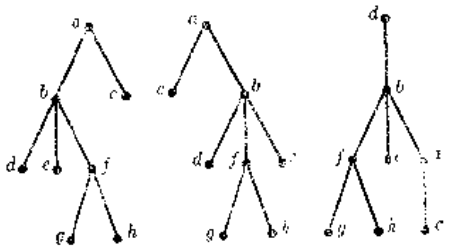


图35 三个树结构

在一个有向图形中的一条欧拉线路，是一条有向通路 $(e_1, e_2, \cdots, e_m)$，使得在这有向图形中的每一条弧恰恰出现一次，而且 $\mathrm{fin}(e_m)=\mathrm{init}(e_1)$。这就是“完全地遍历”有向图形的弧（欧拉线路，是从伦哈特·欧拉在1736年关于“在一个星期天闲逛时，不可能恰好一次地通过柯尼斯堡城中七座桥中的每一座”的著名讨论而得名的。他还对无向图形讨论了类似的问题。欧拉线路应当与“哈密顿线路”加以区别，后者是恰好一次地碰到每一个顶点的有向回路；见第7章）。

一个有向图形说是平衡的（见图36），如果每一个顶点 V 都有与其输出次数相同的输入次数，亦即，如果，以 V 作为它们的初始顶点的弧，与以 V 作为最终顶点的弧，恰恰一样多。这个条件与克希霍夫定律密切相关（见习题24）。显然，仅当一个有向图形是连通的和平衡的时，才有可能找到这图形的一条欧拉线路。假定没有孤立的顶点，即输入次数和输出次数两者都等于0的顶点。

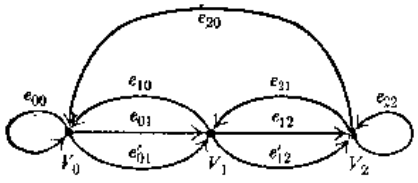


图36 平衡的有向图形

这一小节迄今为止，已经有了好几个定义（例如，有向图形，弧，初始顶点，最终顶点，输出次数，输入次数，有向通路，简单的有向通路，有向回路，有向树，欧拉线路，孤立顶点，以及是强连通的，有根的，以及平衡的诸性质），但尚缺乏联系这些概念的重要结果。现在我们已经为内容丰富的材料作好了准备。头一个基本的结果，是一个由欧·约·古德（I. J. Good）给出的定理〔《伦敦数学学会杂志》（J. London Math. Soc.）21(1947), 167～169〕，他证明欧拉线路是经常可能的，除非明显不可能：

定理G 一个有限的没有孤立顶点的有向图形具有一条欧拉线路，当且仅当，它是连通的和平衡的。

证明：假定 G 是平衡的，并设

$$P=(e_1, \cdots, e_m)$$

是一条不两次使用一条弧的最长可能的有向通路。那么，如果 $V=\mathrm{fin}(e_m)$，而且如果 k 是 V 的输出次数，则所有具有 $\mathrm{init}(e)=V$ 的 k 条弧都必然已经出现于 P 中，否则我们可以

加上 e 并获得一条更长的通路。但是如果 $\text{init}(e_j)=V$ 和 $j>1$，则 $\text{fin}(e_{j-1})=V$；因此，由于 G 是平衡的，我们就必须有

$$\text{init}(e_1)=V=\text{fin}(e_m)$$

否则 V 的输入次数将至少为 $k+1$。

现在通过 P 的循环排列得出，任何不在这通路中的弧 e，与任何在这通路中的弧，既没有共同的初始顶点，也没有共同的最终顶点。所以，如若 P 不是一条欧拉线路，则 G 就不是连通的。 ∎

在欧拉线路与有向树之间有一个重要的联系：

引理 E 设 $(e_1, \cdots, e_m)$ 是一个没有孤立顶点的有向图形 G 的一条欧拉线路。命 $R=\text{fin}(e_m)=\text{init}(e_1)$。对于每一顶点 $V\neq R$，命 $e[V]$ 是在这线路中从 V "最后出来"的，即

$$e[V]=e_j,\ \text{如果 } \text{init}(e_j)=V \text{ 而且 } \text{init}(e_k)\neq V \text{ 对于 } j<k\leqslant m \qquad (1)$$

则 G 的顶点连同弧 $e[V]$，形成一个有根 R 的有向树。

证明：有向树定义的性质（a）和（b）显然是满足的。由习题 7 我们仅仅需要证明，在 $e[V]$ 当中没有有向回路；但这是直接的，因为如果 $\text{fin}(e[V])=V'=\text{init}(e[V'])$，其中 $e[V]=e_j$ 且 $e[V']=e_{j'}$，则 $j<j'$。 ∎

这个引理，如果我们换个角度，来考虑"最先进入"到每个顶点的，那么，就可能会理解得更好：诸最先进入者形成一个无序树，其所有的弧的指向都背着 R。引理 E 有一个令人惊喜而又重要的逆定理，它是由塔·范·阿登尼-埃伦斯费特（T. van Aardenne-Ehrenfest）和尼·戈·德·布鲁因证明的[Simon Stevin 28(1951), 203～217]：

定理 D 设 G 是一个有限的，平衡的，有向的图形，且设 G' 是一个由 G 的顶点加上 G 的某些弧组成的有向树。设 R 是 G' 的根，并设 $e[V]$ 是 G' 的具有初始顶点 V 的弧。设 e_1 是 G 的任何具有 $\text{init}(e_1)=R$ 的弧。则 $P=(e_1, e_2, \cdots, e_m)$ 是一条欧拉线路，如果它是一条有向通路，而且对于它

i）没有被使用一次以上的弧，即当 $j\neq k$ 时 $e_j\neq e_k$。

ii）$e[V]$ 不用于 P 中，除非它是与规则（i）相一致的唯一选择；亦即，如果 $e_j=e[V]$ 而且如果 e 是一条具有 $\text{init}(e)=V$ 的弧，则对于某个 $k\leqslant j$，$e=e_k$。

iii）P 终止，仅当其按规则（1）已不能再继续下去时；即，如果 $\text{init}(e)=\text{fin}(e_m)$，则对于某个 k，有 $e=e_k$。

证明：由(iii)和定理 G 证明中的论证，我们必然有 $\text{fin}(e_m)=\text{init}(e_1)=R$。现在倘若 e 是一条不在 P 中出现的弧，则命 $V=\text{fin}(e)$。因为 G 是平衡的，由此得出 V 是某条不在 P 中的弧的初始顶点；而且如果 $V\neq R$，则 $e[V]$ 由条件(ii)必不在 P 中。现在对于 $e=e[V]$ 使用同样的论证，而且我们最终地发现 R 是某条不在这通路中的弧之初始顶点，与(iii)矛盾。 ∎

定理 D 的实质在于，它给了我们一个简单的方法：给定了一个平衡的有向图形之任何有向子树，来构造这图形中的一条欧拉线路（见习题 14 中的例子）。事实上，定理 D 允许我们来计算在一个有向图形中欧拉线路的确切条数；这一结果，以及在这一小节所提出思想的许多其它重要的结果，出现于以下的一些习题之中。

习题

1.〔$M20$〕 证明：如果V和V'是一有向图形的顶点，而且从V到V'有一有向通路，则从V到V'有一简单的有向通路。

2.〔15〕 在2.3.4.1小节的(3)中列出的十条“基本回路”中，哪些是该小节的有向图形（图32）中的有向回路？

3.〔16〕 画出一个连通的但无根的有向图形的图式。

▶4.〔$M20$〕 拓扑分类的概念，对任何有限的有向图形G，可以定义为顶点的一种线性排列，使得在对G中的所有边e的排序中，init(e)居于fin(e)之前(参照2.2.3小节，图6和图7)。并不是所有的有限有向图形都能拓扑地分类的；哪些能呢？(请用这一小节的术语来给出回答。)

5.〔$M21$〕 设G是一个有向图形，包含一条具有$\mathrm{fin}(e_n)=\mathrm{init}(e_1)$的有向通路$(e_1,\cdots,e_n)$。试利用在这一小节中定义的术语，来给出$G$不是一个有向树的证明。

6.〔$M21$〕 以下断言是真还是假：一个有根且不含回路和不含有向回路的有向图形，是一个有向树。

▶7.〔$M22$〕 以下断言是真还是假：一个满足有向树定义的性质(a)和(b)，而且无有向回路的有向图形，是一个有向树。

8.〔$HM40$〕 研究有向树的自同构群的性质，即是这样的群，它由顶点与弧的那样一些置换π组成，π使得$\mathrm{init}(e\pi)=\mathrm{init}(e)\pi$，$\mathrm{fin}(e\pi)=\mathrm{fin}(e)\pi$者。

9.〔18〕 通过对边赋予方向，以G作为根，画出对应于2.3.4.1小节的图30中的自由树之有向树。

10.〔22〕 一个具有顶点$V_1,\cdots,V_n$的有向树，通过使用一个表格$F[1],\cdots,F[n]$，在计算机内可表示如下：如果V_j是根，则$F[j]=0$；否则，如果弧$e[V_j]$从V_j引向V_k，则$F[j]=k$（于是，$F[1],\cdots,F[n]$就等同于在算法2.3.3E中所使用的“FATHER(父亲)”表格)。

正文说明了，如何通过选择任何所想要的顶点来作为根，以使一个自由树形能转化成为一个有向树。因而，有可能由一个有着根R的有向树开始，然后通过忽略弧的方向，而把它转化成为一个自由树，又最后赋予新的方向，来得到一个以任何确定的顶点作根的有向树。试设计一个实施这个变换的算法：以一个表格$F[1],\cdots,F[n]$开始，它表示一个有向树形，并给出一整数j，$1\leqslant j\leqslant n$，来设计变换F表格，使得它表示同样的自由树但以V_j作为根的算法。

▶11.〔28〕 利用习题2.3.4.1-9的假定，但以(a_k,b_k)表示一条其箭头从V_{a_k}指向V_{b_k}的边，试设计一个算法，使得它不仅如同在该算法中那样，打印出一自由树，而且还打印出基本回路〔提示：习题2.3.4.1-9之解答中所给出的算法，可以与上一题的算法组合在一起〕。

12.〔$M10$〕 在这里所定义的有向树，与在2.3节开头所定义的有向树之间的对应中，一个树形节点的次数，是否等于对应顶点的输入次数或输出次数？

▶13.〔$M24$〕 证明：如果R是一个有向图形G（可能无限）的一个根，则G包含一个

与G有相同顶点并以R为根的有向子树（作为一个推论，总是有可能象在2.3.4.1小节的图32中那样，来选择框图中的自由树，使得它实际上是一个有向子树；如果我们选择e''_{13}，e''_{10}，e_{20}，和e_{17}来代替e'_{13}，e'_{10}，e_{23}，和e_{15}，则这就是在该图式中的这种情况）。

14. [21] 设G是图36中所示的有向图形，并设G'是具有顶点V_0，V_1，V_2和弧e_{01}，e_{21}的有向子树。试以弧e_{12}开始，来找出满足定理D的条件的所有通路P。

15. [M20] 证明：连通的和平衡的有向图形，是强连通的。

▶16. [M24] 在一个称为“时钟”的流行的单人纸牌游戏中，一副普通的扑克牌的52张牌面朝下地分成13叠，每叠4张；12叠象一个时钟的12个小时那样排列在一个圆周上，而第13叠放在正中。单人游戏通过翻转正中这叠的顶点的纸牌开始，然后如果它的面值为k，则我们就把它放在第k叠一傍（1，2，…，13等价于A，2，…，10，J，Q，K）。继续通过翻转第k叠顶上的纸牌，并把它放到相应叠的傍边，等等，直至我们达到了，由于在所指定的叠上已不再有纸牌可供翻转，已不可能再继续下去为止（在这个游戏中，游戏者没有什么选择，因为上边的规则完全确定了他的动作）。当游戏终止时，如果所有的牌面朝上，则这游戏就得胜了〔参考：小阿·莫伊西(A. Moyse, Jr.)，《单人游戏150种》(150 Ways to play solitaire)（芝加哥：惠特曼，1950)〕。

说明：该游戏将获胜当且仅当以下的有向图形是一个有向树：顶点是$V_1, V_2, \cdots, V_{13}$；弧是$e_1, e_2, \cdots e_{13}$，其中e_j从V_j到V_k，如果在分牌之后k是第j叠底下的牌。

（特别是，如果第j叠的底下的牌是“j”，对于$j \neq 13$，则容易看出，这场游戏肯定失败，因为这张纸牌就决不能被翻过来。这道习题所证明的结果，给出了一种更快得多地进行这一游戏的方法！）

17. [M32] 假定一副纸牌是随机地来洗的，则单人时钟游戏（习题16中所述的）取胜的概率是什么？当游戏结束时，恰有k张纸牌仍然面朝下的概率又是什么？

18. [M30]（奥凯达(Okada)和奥诺迪拉(Onodera)，《山形大学报告》(Bull. Yamagata Univ.) 2 (1952), 89～117。）设G是一个有$n+1$个顶点$V_0, V_1, \cdots, V_n$和m条边$e_1, \cdots, e_m$的图形。通过对每条边赋予一个任意的方向，使G成为一个有向图形；然后构造一个$m \times (n+1)$矩阵A如下：

$$a_{ij} = \begin{cases} +1, & \text{如果 } \mathrm{init}(e_i) = V_j \\ -1, & \text{如果 } \mathrm{fin}(e_i) = V_j; \\ 0, & \text{否则。} \end{cases}$$

命A_0是A通过删去0列而得到的$m \times n$矩阵。

a）如果$m = n$，则证明：如果G不是一个自由树，则A_0的行列式等于0，而如果G是一个自由树形，则它等于± 1。

b）证明：对于一般的m，$A_0^T A_0$的行列式是G的自由子树的个数（即，从m条边中选择n条边使所得到的图形是一个自由树形的方式的数目）〔提示：利用（a）和习题1.2.3-16的结果〕。

19. [M31]（卡·威·博查德(C. W. Borchardt)，《纯粹与应用数学杂志》(Journal f. d. reine und angewandte Math.) 57 (1860), 111～121）设G是一个具有顶点V_0，V_1，…，V_n的有向图形。命A是一$(n+1) \times (n+1)$矩阵，其元素为

$$a_{ij}=\begin{cases} -k, & \text{如果 } i\neq j \text{ 且从 } V_i \text{ 到 } V_j \text{ 有 } k \text{ 条弧;} \\ t, & \text{如果 } i=j \text{ 且从 } V_i \text{ 到其它顶点有 } t \text{ 条弧。} \end{cases}$$

(由此得出，$a_{i0}+a_{i1}+\cdots+a_{in}=0$ 对于 $0\leqslant i\leqslant n$)。命 A_0 是这同一矩阵通过删去 0 行和 0 列而得到的矩阵。例如，如果G是图 36 的有向图形，则我们有

$$A=\begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix} \qquad A_0=\begin{pmatrix} 3 & -2 \\ -1 & 2 \end{pmatrix}$$

a) 证明：在特殊情况 $a_{00}=0$ 和 $a_{jj}=1$ (对于 $1\leqslant j\leqslant n$) 之下，而且如果G不包含从一顶点到它自身的弧，则G是一个有根 V_0 的有向树当且仅当 $\det A_0=1$；而且如果G不是一个有向树，则 $\det A_0=0$。

b) 证明：在一般情况下，$\det A_0$ 是以 V_0 为根的G的有向子树的个数（即，从G的弧中选择 n 条使得到的有向图形是一个以 V_0 为根的有向树之方式的数目）〔提示：对弧数应用归纳法〕。

20. 〔$M21$〕 如果G是 $n+1$ 个顶点 V_0, …, V_n 上的一个图形，命B是对于 $1\leqslant i$, $j\leqslant n$ 如下地定义的 $n\times n$ 矩阵：

$$b_{ij}=\begin{cases} t, & \text{如果 } i=j \text{ 且有 } t \text{ 条通过 } V_i \text{ 的边;} \\ -1, & \text{如果 } i\neq j \text{ 且 } V_i \text{ 与 } V_j \text{ 相邻;} \\ 0, & \text{否则。} \end{cases}$$

例如，如果G是 2.3.4.1 小节中图 29 的图形，其中 $(V_0,V_1,V_2,V_3,V_4)=(A,B,C,D,E)$，则我们得出

$$B=\begin{pmatrix} 3 & 0 & -1 & -1 \\ 0 & 2 & -1 & 0 \\ -1 & -1 & 3 & -1 \\ -1 & 0 & -1 & 2 \end{pmatrix}$$

证明：G的自由子树的个数是 $\det B$〔提示：利用习题 18 或 19〕。

21. 〔$HM38$〕 图 36 的有向图形的例子，不仅仅是平衡的，而且还是正则的，这意味着每个顶点都有与每个其它顶点一样的输入次数和输出次数。设G是一个有 $n+1$ 个顶点 $V_0,V_1,\cdots,V_n$ 的正则有向图形，其中每个顶点有等于 m 的输入次数和输出次数〔因此总共有 $(n+1)m$ 条弧〕。设G^*是有 $(n+1)m$ 个对应于G的弧的顶点的图形：设对应于G中从 V_j 到 V_k 的弧的G^*的顶点，以 V_{jk} 来标记之。在G^*中有一条弧从 V_{jk} 到 $V_{j'k'}$ 当且仅当 $k=j'$。例如，如果G是图 36 的有向图形，则G^*如图 37 所示。G中的一条欧拉线路是G^*中的一条哈密顿线路，反之亦然。

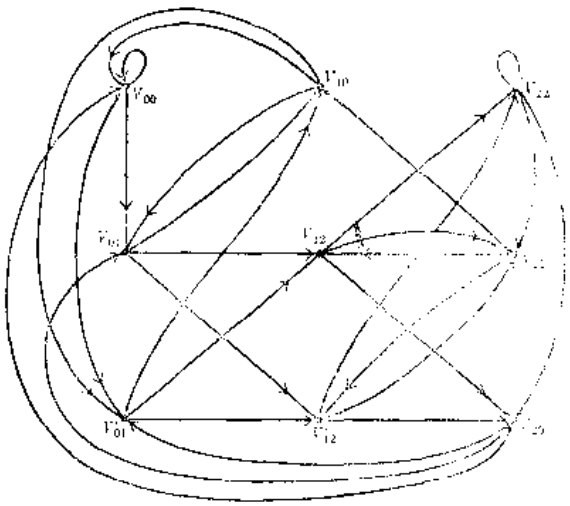


图37 对应于图36的弧有向图形(见习题21)

证明：G^*的有向子树的个数，是 $m^{(n+1)(m-1)}$ 乘G的有向子树形的个数〔提

示：利用习题 19〕。

▶22.〔$M26$〕 设G是一个具有顶点$V_1, V_2, \cdots, V_n$的平衡的有向图形，且无孤立顶点。命σ_j是V_j的输出次数。证明：G的欧拉线路的条数是

$$(\sigma_1+\sigma_2+\cdots+\sigma_n)T\prod_{1\leqslant j\leqslant n}(\sigma_j-1)!$$

其中T是以V_1为根的G的有向子树的个数〔注意：如果认为欧拉线路$(e_1, \cdots, e_m)$等于$(e_k, \cdots, e_m, e_1, \cdots, e_{k-1})$，则因子$(\sigma_1+\cdots+\sigma_n)$即可省略，它是$G$的弧数〕。

▶23.〔$M33$〕（尼·戈·德·布鲁因）对于每个小于m的非负整数的序列$x_1, \cdots, x_k$，命$f(x_1, \cdots, x_k)$是一个小于m的非负整数。定义一个无穷序列如下：$X_1=X_2=\cdots=X_k=0$；$X_{n+k+1}=f(X_{n+k}, \cdots, X_{n+1})$当$n\geqslant 0$。请问：对于$m^{m^k}$个可能的函数$f$中的多少个，这个序列是以极大长度$m^k$为周期的循环序列？〔提示：对所有$0\leqslant x_j<m$，构造一个具有顶点$(x_1, \cdots, x_{k-1})$和具有从$(x_1, x_2, \cdots, x_{k-1})$到$(x_2, \cdots, x_{k-1}, x_k)$的弧的有向图形；应用习题 21 和 22。〕

▶24.〔$M20$〕 设G是一个具有弧$e_0, e_1, \cdots, e_m$的连通的有向图形。设$E_0, E_1, \cdots, E_m$是一组对于G满足克希霍夫定律的正整数，即，对于每个顶点V，

$$\sum_{\mathrm{init}(e_j)=V}E_j=\sum_{\mathrm{fin}(e_j)=V}E_j$$

进一步假定$E_0=1$。试证明：在G中、从$\mathrm{fin}(e_0)$到$\mathrm{init}(e_0)$有一条有向通路，使得边e_0不出现于通路中，而且对于$1\leqslant j\leqslant m$，边e_j恰巧出现E_j次〔提示：把定理G应用于一个适当的有向图形〕。

▶25.〔26〕 试推广树的右穿线二叉树表示，来设计有向图形的一个计算机表示。用两个链接场 ALINK，BLINK 和两个一位的场 ATAG，BTAG；并把该表示设计成这样使得：（a）对于有向图形的每条弧（不是对每个顶点）有一个节点；（b）如果有向图形是一个以R为根的有向树，而且如果我们增加一条从R到一个新的顶点H的弧，则这个有向图形的表示实际上就等同于这个有向树的右穿线表示（同时对在每个家族中的儿子们赋予某个次序），使得 ALINK，BLINK，BTAG 分别等同于 2.3.2 小节中的 LLINK，RLINK，RTAG；而且（C）这表示在这样的意义下是对称的，就是说，把 ALINK，ATAG 与 BLINK，BTAG 进行交换，就等价于改变有向图形的所有弧的方向。

▶26.〔$HM39$〕（随机算法的分析）设G是顶点$V_1, V_2, \cdots, V_n$上的一个有向图形。假定G表示一个算法的框图，其中V_1是“起始”顶点，V_n是“停止”顶点（因此V_n是G的根）。假设G的每条弧e已经赋予一个概率$p(e)$，其中概率满足条件

$$0<p(e)\leqslant 1\qquad\sum_{\mathrm{init}(e)=V_j}p(e)=1\quad 1\leqslant j<n$$

考虑一条“随机通路”，它从V_1开始，而且它顺序地选择G的概率为$p(e)$的分支e，直至达到V_n为止；对分支所进行的每一步选择，不依赖于以前所有的选择。

例如，考虑习题 2.3.4.1-6 的图形，并对弧$e_1, e_2, \cdots, e_9$分别赋予概率 1，$\frac{1}{2}$，$\frac{1}{2}$，$\frac{1}{2}$，1，$\frac{3}{4}$，$\frac{1}{4}$，$\frac{1}{4}$，$\frac{1}{4}$。则通路“起始-A-B-C-A-D-B-C-停止”以概率$1\cdot\frac{1}{2}\cdot$

$1\cdot\frac{1}{2}\cdot\frac{1}{2}\cdot\frac{3}{4}\cdot 1\cdot\frac{1}{4}=\frac{3}{128}$被选取。

这样的随机通路称为马尔科夫（Markov）链，这是因俄国数学家安德烈·马尔科夫而得名的，他第一个对这种类型的随机过程进行了广泛的研究。这种情况可用作某些算法的一个模型，尽管，我们对通路的每一选择必须独立于其它选择这一要求，是一个很强的假定。这里我们所希望解决的问题，是对于这种类型的算法，来分析计算时间。

通过考虑 $n\times n$ 矩阵 $A=(a_{ij})$，能简化分析，其中 $a_{ij}=\Sigma p(e)$ 对于从 V_i 到 V_j 的所有弧 e 进行求和。如果没有这样的弧，则 $a_{ij}=0$。对于上边考虑的例子，矩阵 A 就是

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & \frac{3}{4} & 0 & 0 & \frac{1}{4} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

容易得出，$(A^k)_{ij}$ 是在 k 步之后，一条在 V_i 处开始的通路将达 V_j 处的概率。

对于上述类型的一个任意的有向图形 G，证明下列事实：（a）矩阵 $(I-A)$ 是非奇异的〔提示：说明，没有使得 $xA^n=x$ 的非 0 向量 x〕。（b）顶点 V_j 在通路中出现的平均次数是

$$(I-A)^{-1}_{jj}=\text{余因子}_{jj}(I-A)/\det(I-A),\quad \text{对于 } 1\leqslant j\leqslant n$$

〔于是在所考虑的例子中，我们发现顶点 A，B，C，D 被遍历的平均次数，分别是 $\frac{13}{6}$，$\frac{7}{3}$，$\frac{7}{3}$，$\frac{5}{3}$次〕。（c）V_j 在通路中出现的概率是

$$a_j=\text{余因子}_{j1}(I-A)/\text{余因子}_{jj}(I-A)$$

而且 $a_n=1$，因之通路以概率 1 在有限步内终止。（d）一条从 V_j 开始的随机通路将决不返回到 V_j 的概率是

$$b_j=\det(I-A)/\text{余因子}_{jj}(I-A)$$

（e）在通路中 V_j 恰巧出现 k 次的概率是

$$a_j(1-b_j)^{k-1}b_j\quad \text{对于 } k\geqslant 1\qquad 1\leqslant j\leqslant n$$

***2.3.4.3 “无限性引理”** 到现在为止，我们主要是关注于有限的树，即仅有有限多个顶点（节点）的树；但是我们所给出的关于自由树以及有向树的定义，也同样可应用于无限的图形中。无限的有序树，可以用许多方式来定义，例如，通过把“杜威记数法”推广到无限的数集，如同在习题 2.3-14 那样。甚至在计算机算法的研究中，有时候也需要知道无限树的性质（例如，为了通过反证法证明某个树不是无限的）。无限树的最基本的性质之一如下，它是由德·科尼格（D. König）首先以其完全的一般性给出陈述的：

定理 K（“无限性引理”） 在每个顶点都有有限次数的任何无限有向树中，有一条“由根发出的无限通路”，即，顶点 $V_0,V_1,V_2,\cdots$ 的一个无限序列，其中 V_0 是根而且对于所有

$j \geqslant 0$，$\mathrm{fin}(c[V_{j+1}]) = V_j$。

证明：我们由 V_0，即由有向树的根，开始来定义该通路。假定 $j \geqslant 0$ 而且假定已经选定 V_j 有无限多个后裔。由假设知 V_j 的次数是有限的，所以 V_j 有有限多个儿子 $U_1, \cdots, U_n$。这些儿子之中至少有一个必须有无限多个后裔，于是我们取 V_{j+1} 为 V_j 的这样一个儿子。

现在 $V_0, V_1, V_2, \cdots$，是一条由根发出的无限通路。■

学过微积分的学生可能会认识到，这里所用的论证，实质上就象是用于证明古典的布尔查诺（Bolzano）-维斯特拉斯（Weierstrass）定理“一个有界的无限实数集合有一个聚点”的论证。叙述定理K的一个方式，正如科尼格所发现的那样，就是：“如果人类决不会灭绝的话，现在活着的人当中必然存在一个人，他有决不灭绝的后裔谱系。”

大多数人们，当头一次遇到定理K时，他们都会觉得定理K是完全显然的。但经进一步思索和考虑更多的例子之后，就会认识到关于无限性引理的某些“奥妙”。尽管树的每个节点的次数是有限的，然而我们并没有假定它是有界的（对于所有顶点，小于某个数N），所以可能有次数越来越大的节点。如果我们不是过细地来考虑问题，则至少可以想象每个人的后裔都将最终地死去，尽管将有某些家族传到百万代，另一些传到十亿代，等等，等等。事实上，亨·威·沃森（H. W. Watson）曾经发表过一个“证明”：在生物概率的某些定律不确定地推行之下，将来将有无限多的人出生，但每个家族谱系将以概率1灭绝。他的论文〔J. Anthropological Inst. Gt. Britain and Ireland 4 (1874), 138～144〕实际上包含着一些重要的和影响深远的定理，尽管有使得他作出这一错误论断的少量疏忽，而且值得注意的是，他竟没有发现，他的结论在逻辑上是不一致的。

定理K的反方面可以直接地应用于计算机算法：“如果我们有一个这样的算法，它周期性地把它自己分成为有限多个子算法，而且如果子算法的每个链最终地终止，则这一算法自身也终止。”

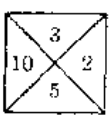
还有另一种方法来陈述。假设我们有一个集合 S，有限或无限，S 的每一元素是一个有限长度 $n \geqslant 0$ 的正整数序列 $(x_1, x_2, \cdots, x_n)$。如果我们赋予条件

i）如果 $(x_1, \cdots, x_n)$ 在 S 中，则对于 $0 \leqslant k \leqslant n$，$(x_1, \cdots, x_k)$ 也在 S 中。

ii）如果 $(x_1, \cdots, x_n)$ 在 S 中，则仅有有限多个 x_{n+1} 存在，使得 $(x_1, \cdots, x_n, x_{n+1})$ 也在 S 中。

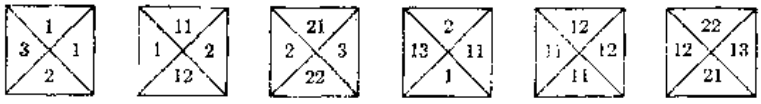
iii）不存在那样的无限序列 $(x_1, x_2, \cdots)$，它的所有的初始子序列 $(x_1, x_2, \cdots, x_n)$ 都在 S 中。那么，实质上，S 是一个实质上以杜威记数法来说明的有向树，而且定理K告诉我们，S 是有限的。

定理K的功效中最令人信服的例子之一，最近已由王浩给出，这与他的“骨牌问题”有关。一块骨牌的式样是一个分成四部分的正方形，每一部分之中有一个确定的数，例如，



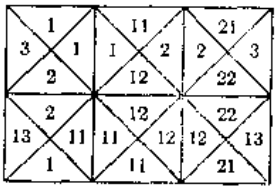
(1)

铺平面问题是：要采用一个有限种骨牌式样的集合，而每种式样的骨牌可以有无限多个，看看怎样把骨牌放到无限平面的每一正方形处（不许转动或反射骨牌式样），使得两块骨牌仅当它们在接触处有相同的数时才相邻。例如，我们可以用下列六个骨牌式样



(2)

以实际上唯一的方式，通过反复地重复矩形



(3)

来砌平面。读者容易验证，用下面三个骨牌式样



(4)

是不可能砌平面的。

王浩的发现[见 *Sci. Am.* 213(1965 年 11 月)，98～106]是：如果有可能来砌平面的右上象限，则就有可能来砌整个平面。这确实是出乎意料的，因为砌右上象限的方法，关系到沿着 x 轴和 y 轴的一个“边界”，而且它似乎对于怎样来砌平面的左上象限并没有给出提示（因为骨牌式样不许转动或反射）。我们不能单纯通过把上方象限的解向下移动并向左移去，来摆脱这边界问题，因为把解移动多于有限次是讲不通的。但是王浩的证明如下：右上象限之解的存在，意味着对于所有的 n，有一个办法来砌一个 $2n\times 2n$ 正方形。对于砌每一边上有偶数个方格的正方形问题，其所有解形成一个有向树，如果每个 $2n\times 2n$ 解 x 的儿子，是通过 x 镶边得到的可能的 $(2n+2)\times(2n+2)$ 解。这个有向树的根是 0×0 解；它的儿子是 2×2 解，等等。每个节点仅有有限多个儿子，因为砌平面的问题假定：仅仅给定了有限多个骨牌式样；因此按无限性引理，有一条由根发出的无限通路。这意味着有一个办法来砌整个平面！

习题

1.〔*M*10〕 正文提到一个包含正整数的有限序列的集合 S，并指出，“这个集合实质上是一个有向树”。这个有向树的根是什么？弧又是什么？

2.〔20〕 证明：如果允许转动骨牌式样，则总有可能来砌平面。

▶3.〔*M*23〕 如果，当给定了一个无限的骨牌式样的集合时，可能砌平面的右上象限，则是否总有可能砌整个平面？

4.〔*M*25〕（王浩）六个骨牌式样（2）导致对砌平面问题的一个“圆环”解，即是一个这样的解，其中某个矩形模式[就是（3）]被反复复制到整个平面上。

不加证明地假定，每当有可能以一个有限的骨牌式样集合来砌平面时，就有一个使用这些骨牌式样的圆环解。试利用这个假定连同无限性引理一起，来设计一个这样的算法：给定了任何有限的骨牌式样集合的说明，在有限步之内确定出是否有一个方法，用这些式样来砌平面。

5.〔*M*40〕 证明：利用下列的92种骨牌式样，有可能来砌平面，但没有在习题4定义意义下的“圆环”解。

为了简化这92种骨牌式样的说明，首先让我们引进某种记号。定义下列的“基本代码”：

$\alpha=(1,2,1,2)$　$\beta=(3,4,2,1)$　$\gamma=(2,1,3,4)$　$\delta=(4,3,4,3)$

$a=(Q,D,P,R)$　$b=(\ ,\ ,L,P)$　$c=(U,Q,T,S)$　$d=(\ ,\ ,S,T)$

$N=(Y,\ ,X,\ )$　$J=(D,U,\ ,X)$　$K=(\ ,Y,R,L)$　$B=(\ ,\ ,\ ,\ )$

$R=(\ ,\ ,R,R)$　$L=(\ ,\ ,L,L)$　$P=(\ ,\ ,P,P)$　$S=(\ ,\ ,S,S)$

$T=(\ ,\ ,T,T)$　$X=(\ ,\ ,X,X)$

$Y=(Y,Y,\ ,\ )$　$U=(U,U,\ ,\ )$　$D=(D,D,\ ,\ )$　$Q=(Q,Q,\ ,\ )$

骨牌式样现在是

$\alpha\{a,b,c,d\}$　〔4个式样〕

$\beta\{Y\{B,U,Q\}\{P,T\},\ \{B,U,D,Q\}\{P,S,T\},\ K\{B,U,Q\}\}$　〔21个式样〕

$\gamma\{\{\{X,B\}\{L,P,S,T\},\ R\}\{B,Q\},\ J\{L,P,S,T\}\}$　〔22个式样〕

$\delta\{X\{L,P,S,T\}\{B,Q\},\ Y\{B,U,Q\}\{P,T\},\ N\{a,b,c,d\},$

$J\{L,P,S,T\},\ K\{B,U,Q\},\ \{R,L,P,S,T\}\{B,U,D,Q\}\}$　〔45个式样〕

这些缩写意味着：基本代码将被逐个分量地放在一起，并在每个分量中以字母顺序进行分类。这样，$\beta Y\{B,U,Q\}\{P,T\}$代表六个式样βYBP，βYUP，βYQP，βYBT，βYUT，βYQT。式样βYQT在乘对应的分量并按次序分类后，为

$(3,4,2,1)(Y,Y,\ ,\ )(Q,Q,\ ,\ )(\ ,\ ,T,T)=(3QY,4QY,2T,1T)$

我们打算把这个式样对应于以下所示的骨牌式样，其中我们用符号串来代替骨牌式样的四个部分中的数。两个骨牌式样能彼此相邻地放置，仅当在它们接触的部分有同样的符号串。



一个“β类”的骨牌式样，指的是在它的如上给出的说明中有一个β者。为着手求解这道习题，注意β类的任何骨牌，都必须有α类的一块骨牌在它的左边和在它的右边，而且必须有δ类的一块在它的上边和下边。“αa”骨牌必须有“βKB”或“βKU”或“βKQ”在它的右边，而且然后必须来一块“αb”的骨牌，等等。

（上述构造，是一个由罗伯特·伯杰（Robert Berger）给出的类似形式的简化，他曾经接着证明，如果没有不正确的假定，则习题4中的一般问题，是不可能解决的。见Memoirs Amer. Math. Soc. 66(1966)）。

▶6.〔*M*23〕（奥托·施利耶尔（Otto Schreier））在一篇著名的论文〔Nieuw Archief voor Wiskunde（2）15（1927），212～216〕中，巴·利·范德瓦尔登（B. L. van der Waerden）证明了：

> “如果k和m是正整数，而且如果我们有正整数的k个集合$S_1,\cdots S_k$，使得每个正整数必须至少包含在这些集合之一中，则在这些集合S_j中至少有一个包含一长度为m的算术级数。”

(这后边的论断，指的是存在整数 a 和 $\delta>0$，使得 $a+\delta$，$a+2\delta$，…，$a+m\delta$ 全都在 S_j 中。) 如有可能，请利用这一结果和无限性引理来证明更强的命题：

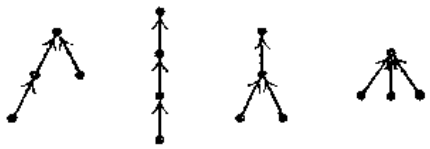
"如果 k 和 m 是正整数，那么，就有一个数 N，使得：如果我们有整数的 k 个集合 S_1，…，S_k，而且 1 与 N 之间的每个整数至少包含在这些集合之一中，则诸集合 S_j 中至少有一个包含一长度为 m 的算术级数。"

▶7. [M30] 如有可能，请利用习题 6 的范德瓦尔登定理和无限性引理，来证明更强的命题：

"如果 k 为一正整数，并且如果我们有整数的 k 个集合 S_1，…，S_k，使得每个正整数包含在至少这些集合之一中，则诸集合 S_j 中至少有一个包含一无限长的算术级数。"

▶8. [M39] (约·伯·克鲁斯科尔(J. B. Kruskal)，Trans. Am. Math. Soc. 95(1960)，210～225。) 如果 T 和 T' 是 (有限的有序的) 树，命记号 $T\subseteq T'$ 标记：T 能被嵌入 T'，如象在习题 2.3.2-22 中那样。试证：如果 $T_1, T_2, T_3, \cdots$ 是任何无限的树形序列，则存在整数 $j<k$，使得 $T_j\subseteq T_k$ (换句话说，不可能构造一无限的树形序列，使其中没有树"包含"序列中任何较前的树形。这一事实，可用来证明某些算法必然终止)。

***2.3.4.4 树的枚举** 树的数学理论，对于算法分析的某些最有启发性的应用，是与给出用于计算各种类型中不同树的数目的公式有关的。例如，如果我们要想知道，当有着四个不能区别的顶点时，能有多少不同的有向树可被构造，则我们发现恰有四种可性能：



(1)

作为我们的头一个枚举问题，让我们来确定具有 n 个顶点的在结构上不同的有向树的个数 a_n。显然，$a_1=1$。如果 $n>1$，则这树有一个根和各种各样的子树。假设有 j_1 个 1 个顶点的子树，有 j_2 个 2 个顶点的子树，等等。则我们可以有

$$\binom{a_k+j_k-1}{j_k}$$

种方式，来选择 a_k 个可能的 k 顶点树中的 j_k 个，因为重复是允许的(参照习题 1.2.6-60)；而且因此我们看出

$$a_n=\sum_{j_1+2j_2+\cdots=n-1}\binom{a_1+j_1-1}{j_1}\cdots\binom{a_{n-1}+j_{n-1}-1}{j_{n-1}},\quad \text{对于 } n>1 \tag{2}$$

如果我们考虑具有 $a_0=0$ 的生成函数 $A(z)=\sum_n a_n z^n$，则我们得出等式

$$\frac{1}{(1-z^r)^a}=\sum_j\binom{a+j-1}{j}z^{rj}$$

连同 (2) 一起，推出

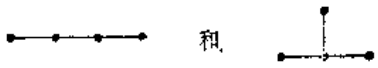
$$A(z)=z/(1-z)^{a_1}(1-z^2)^{a_2}(1-z^3)^{a_3}\cdots \tag{3}$$

这不是对于 $A(z)$ 的一个特别好的形式，因为它包含一个无限的乘积，而且系数 $a_1, a_2, \cdots$ 出现在右边；表示 $A(z)$ 的一种较好的方式，在习题 1 中给出了，而且这导出一个计算 a_n 的相当有效的公式 (见习题 2)；事实上，它也能用来推导对于大 n，a_n 的渐近特性(见

习题 1）。我们求得

$$A(z)=z+z^2+2z^3+4z^4+9z^5+20z^6+48z^7+115z^8$$
$$+286z^9+719z^{10}+1842z^{11}+\cdots \tag{4}$$

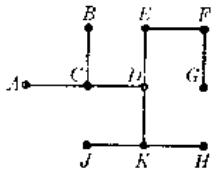
现在，我们实际上已经求出了有向树的个数。确定具有 n 个顶点的在结构上不同的自由树的个数是十分有趣的。对于四个顶点恰有两个不同的自由树，就是



(5)

因为当去掉方向后，(1) 的头两个和后两个有向树就变成相同的了。

我们已经看到，有可能来选择一个自由树的任何顶点 X 并以唯一的方式对边赋予方向，使得它变成一个以 X 作为根的有向树。对于一给定的顶点 X，一旦这一步已经完成，随即假设根 X 有 k 个子树，在这些分别的树中有 $s_1, s_2, \cdots, s_k$ 个顶点。显然，k 是通过 X 的弧数；而且 $s_1+s_2+\cdots+s_k=n-1$，比自由树中的顶点总数少 1。在这种情况下，我们说 X 的权是 $\max(s_1, s_2, \cdots, s_k)$。于是，在树



(6)

中，顶点 D 有权 3（每个从 D 引出的子树，都有九个剩下的顶点中的三个）。而顶点 E 有权 $\max(7, 2)=7$。一个具有极小的权数之顶点，称作该自由树的一个形心。

设 X 和 $s_1, s_2, \cdots, s_k$ 如上，且命 $y_1, y_2, \cdots, y_k$ 是从 X 伸出的子树的根。显然，y_1 的权至少是 $n-s_1=1+s_2+\cdots+s_k$，因为当 y_1 是假定的根时，在其经由 X 的子树中有 $n-s_1$ 个点。如果在 y_1 子树中有一个形心 Y，则我们有

$$\text{权}(X)=\max(s_1, s_2, \cdots, s_k)\geqslant \text{权}(Y)\geqslant 1+s_2+\cdots+s_k$$

而且这导出 $s_1>s_2+\cdots+s_k$。在这个讨论中，如果我们以 Y_j 代替 Y_1，则可以推导出类似的结果。所以，在一个顶点的诸子树中，至多有一个子树能包含一个形心。

这是一个很强的条件，因为它意味着，在一个自由树中至多有两个形心，而且如果存在两个形心，则它们是相邻的（见习题 9）。

反之，如果 $s_1>s_2+\cdots+s_k$，则在 y_1 的子树中有一个形心，因为

$$\text{权}(Y_1)\leqslant\max(s_1-1, 1+s_2+\cdots+s_k)\leqslant s_1=\text{权}(X)$$

而且在 $Y_2, \cdots, Y_k$ 子树中，所有节点的权至少是 s_1-1。我们已经证明：顶点 X 是一个自由树的唯一的形心，当且仅当，

$$s_j\leqslant s_1+\cdots+s_k-s_j \quad \text{对于} \quad 1\leqslant j\leqslant k \tag{7}$$

因此具有 n 个顶点，仅有一个形心的自由树的个数，是具有 n 个顶点的有向树个数减去不符合条件 (7) 的这样的有向树的个数；而后者实质上由一个具有 s_j 个顶点的有向树以及另一个具有 $n-s_j\leqslant s_j$ 个顶点的有向树组成。因此，具有一个形心的自由树的个数成为

$$a_n - a_1 a_{n-1} - a_2 a_{n-2} - \cdots - a_{\lfloor n/2 \rfloor} a_{\lceil n/2 \rceil} \tag{8}$$

一个具有两个形心的自由树，有偶数个顶点，而且每个形心的权是 $n/2$（见习题 10）。所以，如果 $n=2m$，则双形心的自由树的个数，是由 a_m 个事物中允许重复地选出 2 个的选择数，即是

$$\binom{a_m+1}{2}$$

于是，为得到自由树的总数，当 n 为偶数时，我们把 $\frac{1}{2}a_{n/2}(a_{n/2}+1)$ 加到（8）上。等式（8）的形式提示了一个简单的生成函数，而且，事实上，我们不难求得，对于结构上不同的自由树，其生成函数是

$$F(z)=A(z)-\frac{1}{2}A(z)^2+\frac{1}{2}A(z^2)=z+z^2+z^3+2z^4+3z^5+6z^6+11z^7$$
$$+23z^8+47z^9+106z^{10}+235z^{11}+\cdots \tag{9}$$

这个 $F(z)$ 与 $A(z)$ 之间的简单关系，主要归功于卡·约当（C.Jordan），他在 1869 年考虑了这个问题。

现在让我们回到枚举有序树的问题。关于计算机程序设计的算法，它是我们主要关心的问题。对于四个顶点，有五种结构上不同的有序树：



(10)

其中头两个，作为有向树是相同的，所以在上边的（1）中仅出现这两者之一。

在我们探讨不同的有序树结构的个数之前，首先让我们考虑二进树的情况，因为这更接近于实际的计算机表示，而且更易于研究。设 b_n 是具有 n 个顶点的不同二叉树的个数。按二叉树的定义，显然 $b_0=1$，而且对于 $n>0$，可能性的种数就是把一个具有 k 个节点的二叉树放到根的左边、又把另一个具有 $n-1-k$ 个节点的二叉树放到右边的方式的数目。所以

$$b_n=b_0b_{n-1}+b_1b_{n-2}+\cdots+b_{n-1}b_0 \quad n\geqslant 1 \tag{11}$$

按这个关系，显然，生成函数

$$B(z)=b_0+b_1z+b_2z^2+\cdots$$

满足方程

$$zB(z)^2=B(z)-1$$

求解这个二次方程，并利用 $B(0)=1$ 这个事实，我们便得到

$$B(z)=\frac{1}{2z}(1-\sqrt{1-4z})$$
$$=\frac{1}{2z}\left(1-\sum_{n\geqslant 0}\binom{\frac{1}{2}}{n}(-4z)^n\right)=\sum_{m\geqslant 0}\binom{\frac{1}{2}}{m+1}(-1)^m 2^{2m+1}z^m$$
$$=1+z+2z^2+5z^3+14z^4+42z^5+132z^6+429z^7+1430z^8+4862z^9$$
$$+16796z^{10}+\cdots \tag{12}$$

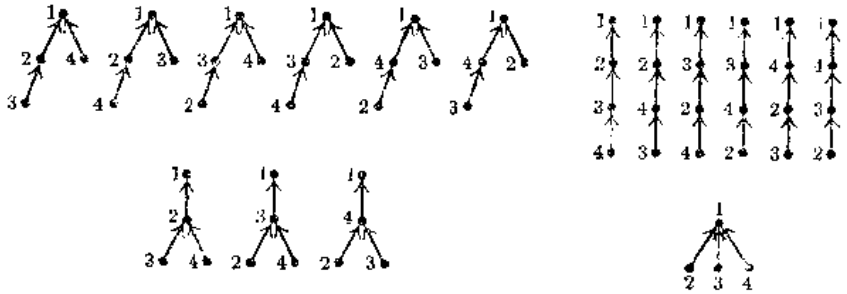
因此所求的答案是

$$b_n=\binom{-\frac{1}{2}}{n+1}(-1)^n 2^{2n+1}=\frac{1}{n+1}\binom{2n}{n} \tag{13}$$

按斯特林渐近公式，这近似于$4^n/n\sqrt{\pi n}+O(4^n n^{-5/2})$。等式（13）的某些重要推广出现于习题 11 和 32 中。

转回到我们关于具有 n 个节点的有序树的问题，我们可以看出，这与二叉树的个数，实质上是同一个问题。因为，在二叉树与森林之间，我们有一个标准的对应，而且一个树去掉它的根就成为一个森林。因此，具有 n 个顶点的（有序）树形的个数是 b_{n-1}，即具有 $n-1$ 个顶点的二叉树的个数。

以上所进行的枚举，假定了这些顶点是不可区分的点。如果我们在（1）中将诸顶点标记为 1，2，3，4，并选定 1 作为根，则我们现在就得到 16 个不同的有向树：



(14)

对于带标号的树，其枚举问题显然十分不同于以上所解决的问题。在这种情况下，这可以重新叙述如下："考虑来画从顶点 2，3，和 4 中的每一个，指向其它顶点的线；从每个顶点发出的线，有三种选择，所以全部共有 $3^3=27$ 种可能性。这 27 种当中有多少种将得出以 1 为根的有向树呢？"如我们已看到的，答案是 16。对于同一类问题，但是针对 n 个顶点的情况来说，一个类似的陈述如下："设 $f(x)$ 是一个整数值的函数，对所有的整数 $1\leqslant x\leqslant n$ 有 $f(1)=1$ 和 $1\leqslant f(x)\leqslant n$。我们称 f 为一'树函数'，是说如果对于所有的 x，$f^n(x)$，即 $f(f(\cdots(f(x))\cdots))$迭代 n 次，等于 1。问树函数有多少个？"这个问题，比如说，在考虑随机数的生成时就会被提出来。我们将发现，相当令人惊奇地是，平均说来，每 n 个这样的函数 f 当中恰好有一个是树函数。

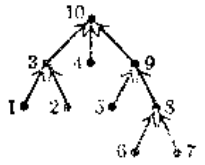
利用前几小节已提供的计算图形的子树个数的一般公式，可以很容易地导出对于这一枚举问题的解（见习题 12）。但是有一个更好的解决问题的方法，因为它给了我们一种新的和紧致的方式，来表示有向树结构。

现在让我们假设，我们已给了一个具有顶点 $\{1,\ 2,\ \cdots,\ n\}$ 和 $n-1$ 条弧的有向树，其中的弧，对于所有的 j，除了根之外，都是从 j 到 $f(j)$ 进行的。则至少有一个终端顶点（叶）；命 V_1 是最小号数的终端顶点。如果 $n>1$，则写下 $f(V_1)$ 并从树中删去 V_1 以及从 V_1 到 $f(V_1)$ 的弧；然后设 V_2 是在得到的树中最小号数的终端顶点。如果 $n>2$，则写下 $f(V_2)$ 并从此树删去 V_2 以及从 V_2 到 $f(V_2)$ 的弧；以这一方式继续进行，直到除根之外的所有顶点都被删去为止。得到的 $n-1$ 个数的序列

$$f(V_1),\ f(V_2),\ \cdots,\ f(V_{n-1}) \tag{15}$$

其中 $1\leqslant f(V_j)\leqslant n$，称为原来的有向树的典型表示。

例如，具有 10 个顶点的有向树



(16)

有典型表示 3，3，10，10，9，8，8，9，10。

这里的重要之点在于，我们能把这一过程逆过来，并且从任何 $n-1$ 个数的序列(15)返回到一个产生它的有向树。因为，如果我们有 1 到 n 之间的数的任何序列 x_1，x_2，…，x_{n-1}，则命 V_1 是最小的不出现于序列 x_1，…，x_{n-1} 之中的数，然后命 V_2 是最小的不出现于序列 x_2，…，x_{n-1} 中的$\neq V_1$ 的数，如此类推。以这一方式得到整数 1，2，…，n 的一个排列 V_1，V_2，…，V_n 之后，对于 $1\leqslant j\leqslant n$，画出从顶点 V_j 到顶点 x_j 的弧。这就给出了一个无有向回路的有向图的构造，而且按习题2.3.4.2-7，它是一个有向树形。显然，序列 x_1，x_2，…，x_{n-1}，与关于该有向树的序列（15）相同。

由于这一过程是可逆的，我们已经得到数〈1，2，…，n〉的 $n-1$ 元组，与在这些顶点上的有向树之间的一一对应。因此，具有 n 个带标号的顶点的不同的有向树有 n^{n-1} 个。如果我们确定一个顶点以作为根，则显然其个数并不关系到确定这一个顶点还是那一个顶点。所以在〈1，2，…，n〉上有 n^{n-2} 个不同的有着一个给定的根的有向树。这在（14）中计算出为$16=4^{4-2}$ 个树。从这一个信息，容易确定出具有带标号的顶点的自由树的个数（见习题 22）。一旦我们已经知道当不涉及标号时那个问题的答案（见习题 23），则带标号的顶点的有序树的个数也容易确定出来（见习题 23）。所以，对于带标号与不带标号的顶点来说，三种基本的树类的枚举问题，在这一小节里实际上都已解决了。

如果我们把生成函数的通常方法，应用于枚举带标号的有向树形的问题中，将会发生什么情况呢？那是很有趣的。为此我们大概会发现，最容易的是来考虑量 $r(n,q)$，即具有 n 个顶点，无有向回路，以及从 q 个指定的顶点中的每一个都发出一条弧的带标号的有向图形的个数。因此，具有一个确定根的带标号的有向树的个数，是 $r(n,n-1)$。在这个记号下，通过简单的计算论证，我们发现，对于固定的 m，

$$r(n,q)=\sum_{\substack{k,t\\ t-k=m}}\binom{q}{k}r(t,k)\,r(n-t,q-k),\quad \text{如果 } 0<m<n-q \tag{17}$$

$$r(n,q)=\sum_{k}\binom{q}{k}r(n-1,q-k),\quad \text{如果 } q=n-1 \tag{18}$$

如果我们把未指定的顶点分为 A 和 B 两组，A 中有 m 个顶点，B 中有 $n-q-m$ 个顶点；然后这 q 个指定的顶点又被分成 k 个顶点的一组——这 k 个顶点开始了引向 A 中的通路，与 $q-k$ 个顶点的一组——这 $q-k$ 个顶点开始了引向 B 中的通路，那么，就得出这两个关系中的头一个（17）。通过考虑根的次数为 k 的有向树，即得出关系（18）。

这些关系的形式指出，我们可以通过生成函数

$$G_m(z)=r(m,0)+r(m+1,1)z+\frac{r(m+2,2)z^2}{2!}+\cdots=\sum_k\frac{r(k+m,k)z^k}{k!}$$

有利地进行工作。在这些术语之下，等式（17）指出 $G_{n-q}(z)=G_m(z)G_{n-q-m}(z)$，而且，通过对 m 使用归纳法，我们发现 $G_m(z)=G_1(z)^m$。现在从等式（18），我们得到

$$G_1(z)=\sum_{n\geqslant 1}\frac{r(n,n-1)z^{n-1}}{(n-1)!}=\sum_{k\geqslant 0}\sum_{n\geqslant 1}\frac{r(n-1,n-1-k)z^{n-1}}{k!(n-1-k)!}$$

$$=\sum_{k\geqslant 0}\frac{z^k}{k!}\cdot G_k(z)=\sum_{k\geqslant 0}\frac{(zG_1(z))^k}{k!}=e^{zG_1(z)}$$

换句话说，置 $G_1(z)=w$，从超越方程

$$w=e^{zw} \tag{19}$$

解的系数中，就得到了我们的问题的解。

利用拉格朗日反演公式，即由 $z=\zeta/f(\zeta)$ 可导出

$$\zeta=\sum_{n\geqslant 1}\frac{z^n}{n!}g_n^{(n-1)}(0)$$

其中 $g_n(\zeta)=f(\zeta)^n$，当 f 在原点的邻域中解析且 $f(0)\neq 0$ 时（见习题33），可以求得这个方程的解。在这种情况下，我们可以置 $\zeta=zw$，$f(\zeta)=e^\zeta$，从而导出解

$$w=\sum_{n\geqslant 0}\frac{(n+1)^{n-1}}{n!}z^n \tag{20}$$

这与我们上边得到的答案毫无二致。

乔·尼·拉尼（G.N.Raney）已经证明，我们可以一种重要的方式来推广这一方法，以得到更一般的方程

$$w=y_1e^{z_1w}+y_2e^{z_2w}+\cdots+y_se^{z_sw}$$

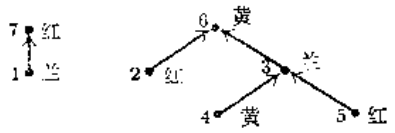
的一个明显的幂级数解，亦即，借助于 y_1，…，y_s 和 z_1，…，z_s 的一个幂级数来解 w。为了进行这一推广，让我们考虑整数的 s 维向量

$$\boldsymbol{n}=(n_1,n_2,\cdots,n_s)$$

为方便计，让我们写成

$$\Sigma\boldsymbol{n}=n_1+n_2+\cdots+n_s$$

假设我们有 s 种“颜色” C_1，C_2，…，C_s，来考虑这样的有向图形，其中每一顶点都被赋予一种颜色，例如，



(21)

命 $r(\boldsymbol{n},q)$ 是如下定义的这种画弧以及对顶点 $\{1,2,\cdots,n\}$ 着色的方式种数：

i）对于 $1\leqslant i\leqslant s$，恰有 n_i 个顶点是 C_i 色的（因此 $n=\Sigma\boldsymbol{n}$）；

ii）有 q 条弧，是从顶点 $\{1,2,\cdots,q\}$ 中的每一个引出来的；

iii）对于 $1 \leqslant i \leqslant s$，恰有 q_i 条弧引到颜色为 C_i 的顶点上（因此 $q = \Sigma q$）；

iv）无有向回路（因此 $q < n$）。

让我们称此为一（n，q）构造。

例如，如果 C_1 = 红，C_2 = 黄，C_3 = 兰，则（21）示出一个（(3，2，2)，(1，3，2)）构造。当仅有一种颜色时，就是我们已解决了的有向树形问题。

命 n 和 q 是固定的 s 元的非负整数的向量，且命 $n = \Sigma n$，$q = \Sigma q$。对于每个(n, q)构造和每一个数 k，$1 \leqslant k \leqslant n$，来定义一个由四件东西组成的典型表示：

a）一个数 t，具有 $q < t \leqslant n$；

b）n 个颜色的一个序列，其中有 n_i 个颜色 C_i；

c）q 个颜色的一个序列，其中有 q_i 个颜色 C_i；

d）对于 $1 \leqslant i \leqslant s$，集合{1，2，…，$n_i$}中的 q_i 个元素组成的一个序列。

典型表示定义如下：首先，按有向树的典型表示次序 V_1，V_2，…，V_q（如同以前所给出的那样）列出顶点{1, 2, …, q}，然后在 V_j 的下面写出由 V_j 引出的弧上的顶点号数 $f(V_j)$。命 $t = f(V_q)$；并命颜色序列（c）分别是顶点 $f(V_1)$，…，$f(V_q)$的颜色。命颜色序列（b）分别是顶点 k，$k+1$，…，n，1，…，$k-1$ 的颜色。最后，命(d)中的第 i 个序列为 x_{i1}，x_{i2}，…，x_{iq_i}，其中，如果序列 $f(V_1)$，…，$f(V_q)$ 的第 j 个 C_i 颜色的元素是序列 k，$k+1$，…，n，1，…，$k-1$ 的第 m 个 C_i 颜色的元素的话，则有 $x_{ij} = m$。

例如，考虑构造（21）并命 $k = 3$。我们开始列出 V_1，…，V_5 和在它们下面的 $f(V_1)$，…，$f(V_5)$ 如下：

1	2	4	5	3
7	6	3	3	6

因此，$t = 6$，且序列（c）分别是 7，6，3，3，6 的颜色，即红，黄，兰，兰，黄。序列（b）分别表示3，4，5，6，7，1，2 的颜色，即兰，黄，红，黄，红，兰，红。最后，为获得（d）中序列，处理如下：

颜色	在3，4，5，6，7，1，2中这一颜色的元素	在7，6，3，3，6中这一颜色的元素	用列2为列3编码
红	5，7，2	7	2
黄	4，6	6，6	2，2
兰	3，1	3，3	1，1

因此（d）序列是 2；2，2；以及 1，1。

由典型表示，我们便能依如下方式恢复原来的（n，q）构造与数 k；从(a)和(c)我们知道顶点 t 的颜色。关于这个颜色的（d）序列的最后元素，连同（b），告诉我们 t 在序列 k，…，n，1，…，$k-1$ 中的位置；因此，我们就知道了 k，还有所有顶点的颜色。然后（d）中的子序列连同（b）和（c）一起确定 $f(V_1)$，$f(V_2)$，…，$f(V_q)$；最后，如我们对有向树形已经做过的那样，来设置 V_1，…，V_q，就重新构造出有向图形。

这个典型表示的可逆性，允许我们来计算可能的（n，q）构造的个数。因为对于(a)有 $n-q$ 种选择，而对于（b）则有多项式系数

$$\binom{n}{n_1,\ \cdots,\ n_s}$$

种选择，而对于（c）有

$$\binom{q}{q_1,\ \cdots,\ q_t}$$

种选择，最后对于（d）有 $n_1^{q_{11}}n_2^{q_{12}}\cdots n_t^{q_{ts}}$ 种选择。再除以关于 k 的 n 种选择，我们便得到一般的结果

$$r(\boldsymbol{n},\boldsymbol{q})=\frac{n-q}{n}\cdot\frac{n!}{n_1!\cdots n_s!}\cdot\frac{q!}{q_1!\cdots q_t!}n_1^{q_{11}}n_2^{q_{12}}\cdots n_t^{q_{ts}} \tag{22}$$

进一步，我们能推导出类似于等式（17）和（18）的结果：

$$r(\boldsymbol{n},\boldsymbol{q})=\sum_{\substack{\boldsymbol{k},\boldsymbol{t}\\ \Sigma(\boldsymbol{t},\boldsymbol{k})=m}}\binom{\Sigma\boldsymbol{q}}{\Sigma\boldsymbol{k}}r(\boldsymbol{t},\boldsymbol{k})\,r(\boldsymbol{n}-\boldsymbol{t},\boldsymbol{q}-\boldsymbol{k}),\ \text{如果}\ 0<m<\Sigma(\boldsymbol{n}-\boldsymbol{q}) \tag{23}$$

其中约定 $r(\mathbf{0},\mathbf{0})=1$，和 $r(\boldsymbol{n},\boldsymbol{q})=0$ 如果任何 n_i 或 q_i 为负或如果 $\boldsymbol{q}>\boldsymbol{n}$：

$$r(\boldsymbol{n},\boldsymbol{q})=\sum_{1\leqslant i\leqslant s}\ \sum_{k}\binom{\Sigma\boldsymbol{q}}{k}r(\boldsymbol{n}-\boldsymbol{e}_i,\ \boldsymbol{q}-k\boldsymbol{e}_i),\ \text{如果}\ \Sigma\boldsymbol{n}=1+\Sigma\boldsymbol{q} \tag{24}$$

其中 $\boldsymbol{e}_i$ 是在位置 i 上有 1 而其它为 0 的向量。关系（23）的根据，是把顶点 $\{q+1,\cdots,n\}$ 分成为分别有 m 个和 $n-q-m$ 个元素的两部分；通过删去唯一的根并考虑剩下的结构，即导出第二个关系。我们现在得到以下的结果：

定理 R（乔治·尼·拉尼）《加拿大数学杂志（Canadian J.Math.）16（1964），755～762）。设

$$w=\sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \Sigma(\boldsymbol{n}-\boldsymbol{q})=1}}\frac{r(\boldsymbol{n},\boldsymbol{q})}{(\Sigma\boldsymbol{q})!}y_1^{n_1}\cdots y_s^{n_s}z_1^{q_1}\cdots z_s^{q_s} \tag{25}$$

其中 $r(\boldsymbol{n},\boldsymbol{q})$ 由（22）定义，又其中 $\boldsymbol{n}$，$\boldsymbol{q}$ 是 s 维的整数向量。则 w 满足恒等式

$$w=y_1e^{z_1w}+y_2e^{z_2w}+\cdots+y_se^{z_sw} \tag{26}$$

证明。由（23）并对 m 用归纳法，我们求得

$$w^m=\sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \Sigma(\boldsymbol{n}-\boldsymbol{q})=m}}\frac{r(\boldsymbol{n},\boldsymbol{q})}{(\Sigma\boldsymbol{q})!}y_1^{n_1}\cdots y_s^{n_s}z_1^{q_1}\cdots z_s^{q_s} \tag{27}$$

现在按（24），

$$\begin{aligned}w&=\sum_{1\leqslant i\leqslant s}\ \sum_{k}\ \sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \Sigma(\boldsymbol{n}-\boldsymbol{q})=1}}\frac{r(\boldsymbol{n}-\boldsymbol{e}_i,\ \boldsymbol{q}-k\boldsymbol{e}_i)}{k!(\Sigma\boldsymbol{q}-k)!}y_1^{n_1}\cdots y_s^{n_s}z_1^{q_1}\cdots z_s^{q_s}\\&=\sum_{1\leqslant i\leqslant s}\ \sum_{k}\frac{1}{k!}y_iz_i^k\sum_{\substack{\boldsymbol{n},\boldsymbol{q}\\ \Sigma(\boldsymbol{n}-\boldsymbol{q})=k}}\frac{r(\boldsymbol{n},\boldsymbol{q})}{(\Sigma\boldsymbol{q})!}y_1^{n_1}\cdots y_s^{n_s}z_1^{q_1}\cdots z_s^{q_s}\\&=\sum_{1\leqslant i\leqslant s}\ \sum_{k}\frac{1}{k!}y_iz_i^kw^k\end{aligned}$$

∎

以巧妙的生成函数之操作为基础的，关于树的枚举公式之概论，已由欧·约·吉尔德〔《剑桥哲学学会会报》（Proc.Cambridge Philos.Soc.）61(1965)，499～517；61(1968)，489〕给出。这篇重要的论文，包含了关于在本节中所导出的许多公式的广泛的推广。

习题

1.〔$M20$〕（乔·波利亚）证明

$$A(z)=z\cdot\exp\left(A(z)+\frac{1}{2}A(z^2)+\frac{1}{3}A(z^3)+\cdots\right)$$

〔提示：取（3）的对数〕

2.〔$HM24$〕（理·奥特（R.Otter））证明：数 a_n 满足下列条件：

$$na_{n+1}=a_1s_{n1}+2a_2s_{n2}+\cdots+na_ns_{nn}$$

其中

$$s_{nk}=\sum_{1\leqslant j\leqslant n/k}a_{n+1-jk}$$

〔这些公式对于计算 a_n 是有用的，因为 $s_{nk}=s_{(n-k)k}+a_{n+1-k}$〕

3.〔$M40$〕写出一个计算机程序，来确定具有 n 个顶点，$n\leqslant 100$ 的（无标号的）自由树和有向树形的个数（利用习题2的结果）阐述这些数的算术性质；关于它们的质因子，或它们模 p 的余数，可以说些什么？

▶4.〔$HM39$〕（乔·波利亚，1937）利用复变数理论，确定有向树的个数之近似值如下：（a）证明：在0与1之间有一个实数 α，对于它，$A(z)$ 有收敛半径 α 而且 $A(z)$ 对于所有使得 $|z|\leqslant\alpha$ 的复数 z 绝对收敛，并且有极大值 $A(\alpha)=a<\infty$〔提示：当一个幂级数有着非负的系数时，则它或者是整函数，或者有一个正的实奇异点；并通过利用习题1中的恒等式，证明 $A(z)/z$ 当 $z\to\alpha-$ 时是有界的〕。（b）设

$$F(z,\ w)=\exp\left(zw+\frac{1}{2}A(z^2)+\frac{1}{3}A(z^3)+\cdots\right)-w$$

证明：在 $(z,\ w)=(\alpha,\ \alpha/\alpha)$ 的一个邻域内，$F(z,\ w)$ 对每个各别的变量是解析的。（c）证明：在点 $(z,w)=(\alpha,\ \alpha/\alpha)$ 处，$\partial F/\partial w=0$；因此 $a=1$。（d）在点 $(z,w)=(\alpha,\ 1/\alpha)$ 处，证明

$$\frac{\partial F}{\partial z}=\beta=\alpha^{-2}+\sum_{k\geqslant 2}\alpha^{k-2}A'(\alpha^k),\ \text{且}\frac{\partial^2F}{\partial w^2}=\alpha$$

（e）当 $|z|=\alpha$ 且 $z\neq\alpha$ 时，证明 $\partial F/\partial w\neq 0$；因此 $A(z)$ 在 $|z|=\alpha$ 上仅有一个奇异点。（f）证明有一个比 $|z|<\alpha$ 还大的区域，其中

$$\frac{1}{z}A(z)=\frac{1}{\alpha}-\sqrt{2\beta}\sqrt{1-z/\alpha}+R(z)$$

这里 $R(z)$ 是 $\sqrt{z-\alpha}$ 的一个解析函数。（g）因此，证明

$$a_n=\frac{1}{\alpha^{n-1}n^{-3/2}\sqrt{\beta/2\pi}}+O(n^{-5/2}\alpha^{-n})$$

（注意：$1/\alpha=2.95576$，且 $\alpha\sqrt{\beta/2\pi}=0.43992$）

▶5.〔$M25$〕(阿·凯利(A.Cayley)) 设c_n是有着n个叶(即输入次数为0的顶点)且在每个其它的顶点上至少有两个子树的(无标号)有向树的个数。于是,鉴于两树



得出$c_3=2$。试对于生成函数

$$C(z)=\sum_{n\geq 1} c_n z^n$$

求出一个类似于(3)的公式。

6.〔$M25$〕 命一个"有向二叉树",就是一个其中每个顶点有输入次数为2或更小的有向树。试求出一个相当简单的关系,它对于具有n个顶点的不同的有向二叉树的个数,定义生成函数,并求出开头的一些值。

7.〔$HM40$〕 求出习题6的个数的近似值(见习题4)。

8.〔20〕 按照等式(9),具有6个顶点的自由树共有6个。画出它们,并指出它们的形心。

9.〔$M20$〕 在一个自由树中,一个顶点至多有一个子树包含一形心。试从这一事实出发,来证明在一个自由树中至多有两个形心;而且,如果有两个,则它们必然是相邻的。

▶10.〔$M22$〕 证明:一个具有n个顶点和两个形心的自由树,由两个具有$n/2$个顶点的自由树组成,这两个自由树通过一条边连接起来。反之,如果两个具有m个顶点的自由树通过一条边连接起来,则我们就得到一个具有$2m$个顶点和两个形心的自由树。

▶11.〔$M28$〕 正文导出了具有n个顶点的不同的二叉树的个数〔等式(13)〕。试推广这个结果,以求出具有n个顶点的不同的t进树的个数。(参照习题2.3.1:一个t进树形或者是一个空集,或者是由一个根和t个不相交的t叉树组成)。〔提示:利用1.2.9小节中的等式(21)。〕

12.〔$M20$〕 试利用行列式和习题2.3.4.2-19的结果,求出具有n个顶点的带标号的有向树的个数(也请看习题1.2.3-36)。

13.〔15〕 顶点1,2,…,10上的什么有向树,有典型表示3,1,4,1,5,9,2,6,5?

14.〔10〕 下列命题是真还是假:在一个有向树的典型表示中,最后的条款$f(V_n)$,总是该树的根。

15.〔21〕 试讨论存在于(如果有的话)2.2.3小节的拓扑分类算法与有向树的典型表示之间的关系。

16.〔25〕 试设计一个(尽可能高效的)算法,来把一个有向树的典型表示转换成为用"FATHER"链接的通常的计算机表示。

▶17.〔$M26$〕 命$f(x)$是一个整数值函数,其中对于所有的整数$1\leqslant x\leqslant m$,有$1\leqslant f(x)\leqslant m$。定义$x\equiv y$,如果对于某$r$,$s>0$有$f^r(x)=f^s(y)$,其中$f^0(x)=x$且$f^{s+1}(x)=f(f^s(x))$。试利用象这一小节中的枚举方法,来证明:使得对所有x和y有$x\equiv y$的函数的个数,是$m^{m-1}Q(m)$,其中$Q(m)$是1.2.11.3中定义的函数。

18.〔24〕 证明下列方法，是定义1到n的数的（$n-1$）元组，与具有n个带标号顶点的有向树之间的一一对应的又一种方式：命树的叶在递增次序下是V_1，…，V_{k_0}。命$(V_1, V_{k+1}, V_{k+2}, \cdots, V_q)$是从$V_1$到根的通路，并写下顶点$V_q$，…，$V_{k+2}$，$V_{k+1}$。然后命$(V_2, V_{q+1}, V_{q+2}, \cdots, V_r)$是已写出的$V_r$中从$V_2$出发的最短有向通路，并写下$V_r$，…，$V_{q+2}$，$V_{q+1}$。然后命$(V_3, V_{r+1}, \cdots, V_s)$是已写出的$V_s$中从$V_3$出发的最短有向通路，并写下$V_s$，…，$V_{r+1}$；如此类推。例如，树(16)将编码成10,3,3,10,10,9,9,8,8。试证明：这一过程是可逆的并具体画出具有顶点1，2，…，10和表示3，1，4,1,5,9,2,6,5的有向树。

19.〔$M24$〕 有多少个不同的具有n个顶点，且其中k个是叶(即其输入次数为0者)的带标号的有向树？

20.〔$M24$〕(约·赖尔登（J.Riordan))有多少个不同的具有n个顶点，且其中有k_0个输入次数为0，k_1个输入次数为1，k_2个输入次数为2，…的带标号的有向树？（注意，必然$k_0+k_1+k_2+\cdots=n$和$k_1+2k_2+3k_3+\cdots=n-1$）

▶21.〔$M21$〕试枚举带标号的且其中每个顶点的输入次数为0或2的有向树的个数(参照习题20和习题2.3-20)。

22.〔$M20$〕 具有n个顶点的带标号的自由树有多少种可能？(换言之，如果我们给出n个顶点，则有$2^{\binom{n}{2}}$个可能的有着这些顶点的图形，取决于$\binom{n}{2}$条可能的边中那些条被结合到图形中；试问这些图形中有多少个是自由树？)

23.〔$M21$〕 具有n个带标号的顶点的有序树有多少种可能？（给出一个包含阶乘的简单公式。）

24.〔$M16$〕 具有顶点1，2，3，4和根1的所有带标号的有向树如(14)中所示。如果我们列出具有这些顶点和这个根的所有带标号的有序树，则将有多少种呢？

25.〔$M20$〕 在等式(17)和(18)中出现的量$r(n, q)$，其值是什么？(请给出一个明显的公式；正文仅仅提到$r(n, n-1)=n^{n-2}$)

26.〔20〕 借助于这一小节末尾的记号，画出$((3, 2, 4), (2,3,2))$构造(类似于(21))；并求出数k，它对应于有着$t=8$，颜色序列“红，黄，兰，红，黄，兰，红，兰，兰”和“红，黄，兰，红，黄，兰，黄”，以及序列3,2；1，2，1；2，4的典型表示。

▶27.〔$M28$〕 设$U_1, U_2, \cdots, U_p, \cdots, U_q$；$V_1, V_2, \cdots, V_r$是一个有向图形的顶点，其中$1\leqslant p\leqslant q$。设$f$是任何从集合$\{p+1, \cdots, q\}$到集合$\{1, 2, \cdots, r\}$中的函数，并设该有向图形从$U_k$到$V_{f(k)}$恰好含有$q-p$条弧，其中$p<k\leqslant q$。试证明：增加$r$条附加的从每个$U$到一个$U$的弧，使得到的有向图形不含有有向回路的方式数目，是$q^{r-1}p$。请通过推广典型表示的方法，即建立起所有这种增加r条附加弧的方式，与所有的整数序列$a_1, a_2, \cdots, a_r$(其中$1\leqslant a_k\leqslant q$对于$1\leqslant k<r$且$1\leqslant a_r\leqslant p$)的集合之间的一一对应，来证明这一点。

28.〔$M22$〕 利用习题27的结果，来枚举顶点U_1，…，U_m，V_1，…，V_n上，使得所有的边都是从U_j到V_k（对于某个j和k）的，带标号的自由树的个数。

29.〔$HM26$〕 证明：如果$E_k(r, t)=r(r+kt)^{k-1}/k!$，而且如果$zx^t=\ln x$，则对于充分小的$|z|$和$|x-1|$，有

$$x^r=\sum_k E_k(r, t)z^k$$

〔利用等式 (18) 讨论中 $G_m(z)=G_1(z)^m$ 这个事实。〕在这个公式中，r 代表一个任意的实数〔注意：作为这个公式的一个推论，我们有恒等式

$$\sum_k E_k(r,t)E_{n-k}(s,t)=E_n(r+s,t)$$

这就导出了阿贝尔二项式定理（1.2.6小节的等式 16）。并同该小节的等式(31)进行比较〕。

30. 〔*M*23〕 设 n，x，y，z_1，…，z_n 是正整数。考虑 $x+y+z_1+\cdots+z_n+n$ 个顶点 r_i、s_{jk}、t_j（$1\leqslant i\leqslant x+y$，$1\leqslant j\leqslant n$，$1\leqslant k\leqslant z_j$）的一个集合，其中对于所有的 j 和 k，已经画出从 s_{jk} 到 t_j 的弧。按照习题27，从 t_1，…，t_n 中的每一个到另外的顶点画一条弧，使得得到的有向图形不含有向回路，其方式共有 $(x+y)(x+y-z_1+\cdots+z_n)^{n-1}$ 种。试利用这一事实，来证明胡尔维茨（Hurwitz）对于二项式定理的推广：

$$\sum x(x+\epsilon_1z_1+\cdots+\epsilon_nz_n)^{\epsilon_1+\cdots+\epsilon_n-1}y(y+(1-\epsilon_1)z_1+\cdots+(1-\epsilon_n)z_n)^{n-1-\epsilon_1-\cdots-\epsilon_n}$$
$$=(x+y)(x+y+z_1+\cdots+z_n)^{n-1}$$

其中的求和遍取 ϵ_1，…，ϵ_n 等于 0 或 1 的所有的 2^n 种选择。

31. 〔*M*24〕 对于有序树求解习题 5；也就是导出关于有 n 个终端节点的，并无次数为 1 的节点的，不带标号的有序树形个数的生成函数。

32. 〔*M*37〕（阿·厄尔德莱（A.Erdélyi）和艾·马·哈·埃思林顿(I.M.H.Etherington)，爱丁堡数学学报（Edinburgh Math.Notes）32 (1940), 7～12）具有 n_0 个 0 次，n_1 个 1 次，…，n_m 个 m 次节点，并且无高于 m 次的节点的(有序的无标号的)树，共有多少个？（对于这个问题的一个明显的解，可借助于阶乘给出，从而相当可观地推广了习题 11 的结果）。

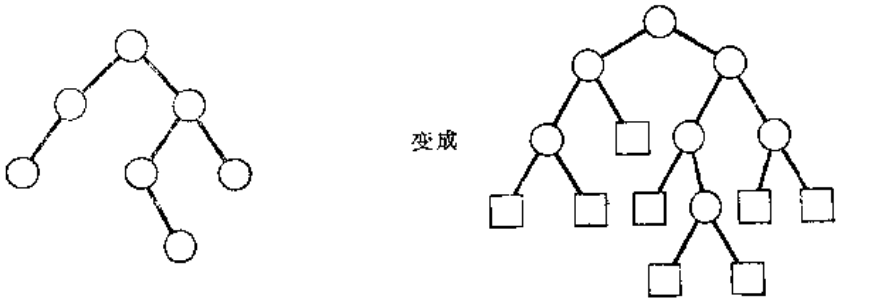
▶33. 〔*M*28〕 正文给出了一个以某些有向森林的枚举公式为基础的方程 $w=y_1e^{z_1w}+\cdots+y_re^{z_rw}$ 的明显的幂级数解。类似地，请证明：习题 32 的枚举公式导出方程

$$w=z_1w^{e_1}+z_2w^{e_2}+\cdots+z_rw^{e_r}$$

的一个明显的幂级数解，把 w 表达成为 z_1，…，z_r 的一个幂级数（这里 e_1，…，e_r 是固定的非负整数，其中至少有一个为 0）。

2.3.4.5 通路长度 在算法分析中，树形的“通路长度”概念极为重要，因为这个量通常直接关系到执行的时间。我们主要关心的是二叉树的情形，因为它与计算机表示密切相关。

在以下讨论中，我们要把每一个二叉树的图式加以扩充，即每当在原来的树中出现一个空的子树时，就增加特殊的节点，使得



(1)

后者称为一个扩充的二叉树。在以这种方式加上了方形的节点之后，有时处理起来就更加方便了。在后面几章中，我们将经常地遇到扩充的二叉树。显然，每一个圆形节点都有两

个儿子，而每一个方形节点都没有儿子（与习题 2.3-20 进行比较）。如果有 n 个圆形节点和 s 个方形节点，则我们有 $n+s-1$ 条边（因为这图式是一个自由树形），而且，如果用另一种方法计算，即通过计算儿子的个数，则我们看到有 $2n$ 条边。因此，显然

$$s = n + 1 \tag{2}$$

亦即，刚才增加的"外部"节点的个数，比我们原有的"内部"节点的个数大 1（对于另一个证明，见习题 2.3.1-14）。甚至当 $n=0$ 时，公式（2）也是正确的。

假定一个二叉树已经这一方式扩充。树的外部通路长度 E，则定义为从根到每个节点——遍取所有外部（方形）节点——的通路长度之和。内部通路长度 I，是对所有内部（圆形）节点求和的这种量。在（1）中，外部通路长度为 $E=3+3+2+3+4+4+3+3=25$，而内部通路长度为 $I=2+1+0+2+3+1+2=11$。这两个量总是通过公式

$$E = I + 2n \tag{3}$$

相互关连，其中 n 是内部节点的个数。

为证明公式（3），考虑删去一个与根之距离为 k 的内部节点 V，其中 V 的两个儿子都是外部的。量 E 随之就减小 $2(k+1)$，因为删去了 V 的儿子，然后它增大 k，因为 V 变成外部的了，所以 E 的纯变化是 $-k-2$。I 的纯变化是 $-k$，这样通过归纳法即可证明（3）。

不难看出，当我们有一个退化为线性结构的树时，内部通路长度是（因而外部通路长度也是）最长的；在这种情况下，内部通路长度是

$$(n-1)+(n-2)+\cdots+1+0=\frac{1}{2}(n^2-n)$$

可以证明，对所有二叉树的"平均"通路长度，实质上与 $n\sqrt{n}$ 成正比（见习题 5）。

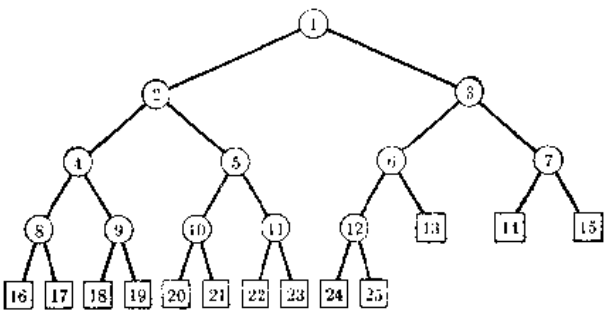
现在，再来考虑一个有着极小通路长度的具有 n 个节点的二叉树的问题：这样一个树形将是重要的，因为它将使各种各样的算法的计算时间减到极小。显然，仅有一个节点（根），与根的距离为 0；至多有两个节点与根的距离为 1，至多有四个节点，距离为 2，等等。所以，我们看到，内部通路长度，总是至少有序列

$$0,1,1,2,2,2,2,3,3,3,3,3,3,3,3,4,4,4,4,\cdots$$

的头 n 项之和那样大。这是和数 $\sum_{1\le k\le n}\lfloor\lg k\rfloor$，我们从习题 1.2.4-42 知道它是

$$(n+1)q-2^{q+1}+2,\quad q=\lfloor\lg(n+1)\rfloor \tag{4}$$

最优值（4）实质上有 $n\lg n$ 的形式；这个最优值，在一个看起来象下图这样的树（对于 $n=12$）中，显然是被达到了：



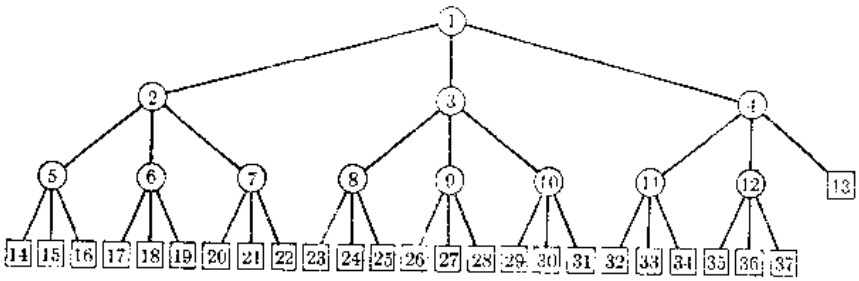
(5)

一个如同（5）这样的树，叫做具有 n 个内部节点的完全二叉树。在一般情况下，我们

可以把节点编号成 1，2，…，n；这种编号很有用，即节点 k 的父亲是节点 $\lfloor k/2 \rfloor$，节点 k 的儿子是节点 $2k$ 和 $2k+1$。外部节点则包含号码 $n-1$ 到 $2n+1$。

可以推知，一个完全的二叉树可以在顺序的内存单元中简单地表示之，使得该结构就隐含在这些节点的单元中。完全的二叉树明显地或隐含地出现在许多重要的计算机算法中，所以读者应该给以特殊的注意。

这些概念对于三叉，四叉，等等树，有重要的推广。我们定义一个 t 叉树为一些节点的一个集合，它或者是空的，或者由一个根和 t 个有序的不相交的 t 进树形组成（参照 2.3 节中二叉树的定义）具有 12 个内部节点的完全三叉树是



（6）

容易看出，怎样来把这个推广到具有内部节点 1，2，…，n 的完全 t 叉树：节点 k 的父亲是节点

$$\lfloor (k+t-2)/t \rfloor = \lceil (k-1)/t \rceil$$

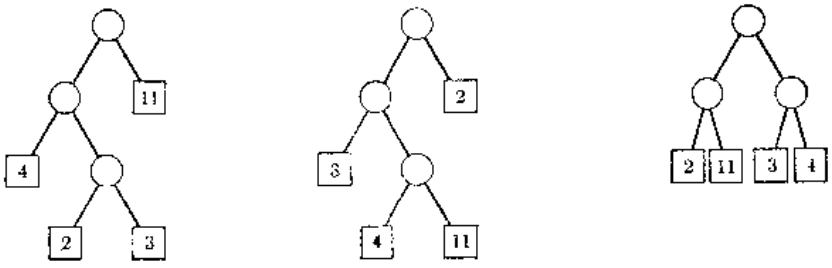
而且节点 k 的儿子是

$$t(k-1)+2,\quad t(k-1)+3,\quad \cdots,\quad tk+1$$

这个树在所有具有 n 个内部节点的 t 叉树当中，有极小的内部通路长度；它的内部通路长度是（见习题 8）

$$\left(n+\frac{1}{t-1}\right)q-\frac{(t^{q+1}-t)}{(t-1)^2},\quad q=\lfloor \log_t((t-1)n+1)\rfloor \qquad (7)$$

如果我们稍微转移一下我们的观点，则这些结果就有另一种重要的推广。假设我们给出了 m 个实数 $w_1, w_2, \cdots, w_m$；问题是要求一个具有 m 个外部节点的扩充的二叉树，而且要把数 $w_1, \cdots, w_m$ 与这些节点结合起来，使得和数 $\Sigma w_i l_i$ 为极小，其中 l_i 是从根出发的通路的长度，而且这个求和是对所有外部节点来进行的。例如，如果给定的数是 2，3，4，11，则我们可形成如下三个扩充的二叉树：



（8）

这里“加权”的通路长度 $\Sigma w_i l_i$ 分别是 34，53，和 40（注意，一个完全平衡的树形，当权数是 2，3，4，和 11 时，并不给出极小的加权的通路长度，尽管我们已经看到，它在特殊情况 $w_1=w_2=\cdots=w_m=1$ 之下给出了极小值）。

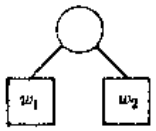
联系到不同的计算机算法，加权的通路长度有若干种解释；例如，我们可以把它应用于合并长度分别为 w_1，w_2，…，w_m 的已分类的序列（见第 5 章）。这个思想最直接了当的一个应用，是把一个二叉树考虑作为一个一般的检索过程，从根开始，然后来做某种测试；测试的结果把我们送到两个分支之一，在那里我们可以进行进一步的测试，等等。例如，如果我们要来判定四个不同的选择中哪一个是真的，而且如果这些可能性将分别以 $\frac{2}{20}$，$\frac{3}{20}$，$\frac{4}{20}$，和 $\frac{11}{20}$ 的概率为真，那么，在这种情况下，一个极小化加权的通路长度的树，就组成一个最优的检索过程〔这些(概率)是(8)中所示的权〕。

为求得一个具有极小的加权的通路长度之树形，有一个精巧的算法，业已由戴·赫夫曼(D. Huffman) 给出：首先找出两个最小的 w 值，比方说是 w_1 和 w_2。然后对 $m-1$ 个权 w_1+w_2，w_3，…，w_m 来求解这问题，并且将这个解中的节点



(9)

代之以

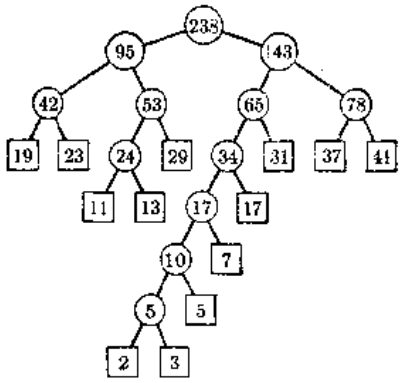


(10)

作为赫夫曼方法的一个例子，让我们来求对于权 2，3，5，7，11，13，17，19，23，29，31，37，41 的最优树。首先我们组合 2 + 3，并寻找对于 5，5，7，…，41 的解；然后，我们组合 5 + 5，等等。这个计算可综述如下：

2	3	5	7	11	13	17	19	23	29	31	37	41
	5	5	7	11	13	17	19	23	29	31	37	41
		10	7	11	13	17	19	23	29	31	37	41
			17	11	13	17	19	23	29	31	37	41
			17		24	17	19	23	29	31	37	41
					24	34	19	23	29	31	37	41
					24	34		42	29	31	37	41
						34		42	53	31	37	41
								42	53	65	37	41
								42	53	65		78
									95	65		78
									95			143
												238

因此下列的树对应于赫夫曼的造树法：



(11)

(在圆形节点内的号数，示出了在这个树与我们的计算之间的对应；还请看习题 9)。

通过对 m 用归纳法，不难证明，这个方法事实上极小化了加权的通路长度。假设 $m \geqslant 2$，而且 $w_1 \leqslant w_2 \leqslant w_3 \leqslant \cdots \leqslant w_m$，并假设我们已给出一个极小化加权通路长度的树（一个这样的树形肯定存在，因为仅有有限多个可能的具有 m 个终端节点的二叉树)。命 V 是一个与根有极大距离的内部节点。如果 w_1 和 w_2 不是已经附加于 V 的儿子的权，则我们可以把它们与已经有的值进行交换，而不增加这个加权的通路长度。于是就有一个极小化了加权通路长度的而且包含子树(10)的树。现在容易证明，这样一个树形的加权通路长度是极小的，当且仅当，以(9)代替(10)所得之树对于权 w_1+w_2，w_3，…，w_m 有极小的通路长度(见习题 9)。

一般说来，有许多个极小化 $\Sigma w_j l_j$ 的树。如果在整个构造过程中诸 w 都进行排序，而且当撤消 w_1，w_2 时，量 w_1+w_2 被排到比任何其它相同权的更靠后的位置上（即，被排到 w_k 与 w_{k+1} 之间，其中 $w_k \leqslant w_1+w_2 < w_{k+1}$)，那么，用赫夫曼方法构造的树，就在所有极小化 $\Sigma w_j l_j$ 的树当中，有最小的 $\max l_j$ 的值和 Σl_j 的值〔见尤金·西·施瓦茨 (Eugene S. Schwartz)，《信息和控制》(Information and Control) 7 (1964)，37～44〕。

赫夫曼方法同样可以推广到 t 叉树，就象二叉树一样（见习题 10)。赫夫曼方法的另一个重要的推广，将在 6.2.2 小节中讨论。关于通路长度的进一步讨论，出现于 5.3.1，5.4.9 小节和 6.3 节中。

习题

1.〔12〕 除了完全的二叉树(5)之外，还有没有任何其它的具有 12 个内部节点和极小通路长度的二叉树？

2.〔17〕 画出一个扩充的二叉树，它具有包含权 1，4，9，16，25，36，49，64，81，100 的终端节点，并有着极小的加权通路长度。

▶3.〔$M24$〕 一个具有 m 个外部节点的扩充的二叉树确定了一个通路长度 $l_1, l_2, \cdots, l_m$ 的集合，它描述了从根到分别的外部节点的通路长度。反之，如果我们给了一个数 l_1，l_2，…，l_m 的集合，则是否总有可能来构造一个扩充的二叉树，其中这些数是在某种次序下的通路长度？试证这是可能的，当且仅当，$\Sigma_{1 \leqslant j \leqslant m} 2^{-l_j} = 1$。

▶4.〔$M25$〕 (尤·西·施瓦茨) 假定 $w_1 \leqslant w_2 \leqslant \cdots \leqslant w_m$。试说明：有一个极小化 $\Sigma w_j l_j$ 的扩充的二叉树，而且对于它，在从左到右的次序下终端节点分别包含值 w_1，w_2，…，w_m

〔例如，树(11)不满足这一条件，因为权数是以次序 19，23，11，13，29，2，3，5，7，17，31，37，41 出现的。我们找到了一个权数以递增次序出现的树，而这对于赫夫曼造树法并不一定如此〕。

5.〔*HM*26〕 命

$$B(w,\ z)=\sum_{n,\,p\geqslant 0} b_{np}w^p z^n$$

其中 b_{np} 是具有 n 个节点和内部通路长度 p 的二叉树的个数〔于是：

$$B(w,\ z)=1+z+2wz^2+(w^2+4w^3)z^3+(4w^4+2w^5+8w^6)z^4+\cdots$$

$B(1,\ z)$是 2.3.4.4 小节的等式(12)中的函数 $B(z)$〕。(*a*)试求出一个表征 $B(w,z)$ 的函数关系。(*b*)利用(*a*)的结果，来确定具有 n 个节点的一个二叉树的平均内部通路长度，假定

$$\frac{1}{n+1}\binom{2n}{n}$$

个树形中的每一个都是同等可能的。(*c*)试求出这个量的近似值。

6.〔16〕 如果通过(1)中那种的"方形"节点来扩充一个 t 叉树，则对应于等式(2)，在方形与圆形节点的个数之间有什么关系？

7.〔*M*21〕 在一个 t 叉树中，其外部与内部通路长度之间有什么关系？〔参照习题 6；希望对等式(3)作一推广〕。

8.〔*M*23〕 证明等式(7)。

9.〔*M*21〕 出现于(11)的圆形节点中的号码，等于对应的子树的外部节点中的权数之和。试证明，所有圆形节点中值的和，等于加权的通路长度。

▶10.〔*M*26〕 (戴·赫夫曼)给定权 w_1，w_2，…，w_m。试说明怎样来构造一个具有极小的加权通路长度的 t 叉树。试对于权 1，4，9，16，25，36，49，64，81，100，来构造一个最优的三叉树。

11.〔16〕 在完全的二叉树(5)与习题 2.3.1-5 中所述的二叉树的"杜威记数法"之间，有无任何联系？

***2.3.4.6 历史和参考文献** 自从开天辟地以来，就有了树林；而几数世纪以来，树结构(特别是家族树)已被普遍地使用了。树的概念，作为一个正式地定义的数学实体，似乎首先出现在占·克希霍夫的著作中〔Annalen der Physik und Chemie 72 (1847)，497～508〕，他使用自由树，在一个电气网络中，联系以他的名字命名的定律，求出一组基本回路，实际上就如象我们在 2.3.4.1 小节所做过的那样。这概念大约在同一时候，也出现在卡·乔·克·冯·斯托特(K. G. Chr. von Staudt)的书《位置几何》(Geometrie der Lage)(pp. 20～21)中。"树"这一名称以及许多主要是讨论树形枚举的结果，十年之后开始出现于阿瑟·凯利的一系列论文中〔见《凯利数学论文集》(Collected Mathematical Papers of A. Cauley) 3 (1857)，242～246；4 (1859)，114～115；9 (1874)，202～204；9 (1875)，427～460；10(1877)，598～600；11(1881)，365～367；13(1889)，26～28〕。凯利还不知道以前的克希霍夫和冯·斯托特的工作；他的研究以代数公式的结构之研讨开始，而且后来它们主要地为化学中的异构体问题的应用所采纳。树结构也为卡·威·博查德〔Journal f. d. reine und angewandte Math. 57 (1860)，111～121〕；约·本·利斯汀

(J. B. Listing)〔Göttinger Abhandlungen, Math. Classe, 10(1862), 137～139〕；以及卡·约当〔Journal f. d. reine und angewandte Math. 70 (1869), 185～190〕独立地研究过。

无限性引理首先是由德尼斯·科尼格系统地阐述的〔《数学基础》Fundamenta Mathematicae 8(1926), 114～134〕，而且在他的经典著作《有限与无限图形的理论》(Theorie der endlichen und unendlichen Graphen)（莱比锡，1936）第6章中给了它一个突出的位置。一个称为"扇子定理"的类似结果，稍微更早地出现在卢·埃·简·希劳维尔(L. E. J. Brouwer)的著作中〔维汗德林根科学院（Verhandelingen Akad.）阿姆斯特丹12 (1919), 7〕，但这包含了更强得多的假设；对布劳威尔的工作的讨论，请见阿·海沃汀(A. Heyting)的《直观论》(Intuitionism)（1956）3.4节。

2.3.4.4小节中关于枚举不带标号的有向树的公式(3)，是由凯利在他关于树的头一篇论文中给出的。在他的第二篇论文中，他枚举了不带标号的有序树；一个等价的问题早在100年前已由约·冯·西格尼尔(J. von Segner)和伦·欧拉提出并解决(Novi Commentarii Academiæ Scientiarum Petropolitanæ 7 (1760), 13～15, 203～209)，而且它也是加·拉默(G. Lamé)，尤·卡塔兰(E. Catalan)，奥·罗德里格斯(O. Rodrigues)和雅·比尼特(J. Binet)在《数学杂志》(Journal de Mathematiques) 3, 4 (1838, 1839)中七篇论文的主题（也请参考习题2.2.1-4)。相应的数，现在普遍称为卡塔兰数。

关于带标号的自由树个数的公式n^{n-2}，是由卡·威·博查德作为他对某个行列式进行求值的一个付产品，于1860年发现的。凯利在1889年给出了对这个公式的一个独立的推导〔见上边的参考文献〕；他的讨论，是极为含糊的，但提示了带标号的有向树形与数的$(n-1)$元组之间的关系。揭示这样一个联系的明显的对应关系，是由海因茨·普里弗(Heinz Prüfer)首先发表的〔Arch. Math. u. Phys. 27 (1918), 142～144〕，而且完全独立于凯利先前的工作。关于这个课题已经发表有大量的论述；而且在约·韦·穆恩(J. W. Moon)《带标号的树之计数》(Counting Labelled Trees)（蒙特利尔：加拿大数学会议，1970）一书中有很漂亮的综述。

一篇非常重要的关于树的枚举和许多其它类型组合结构的论文，由乔·波利亚发表于《数学文集》(Acta Mafh.) 68 (1937), 145～253。关于图形的枚举问题的讨论和一篇出色的文献目录，请看弗兰克·哈拉里(Frank Harary)在《图论与理论物理》(Graph Theory and Theoretical Physics)（伦敦：科学出版社，1967）一书中(pp1～41)的综述。

通过重复地组合最小的权来极小化加权的通路长度的原理，是由戴·赫夫曼发现的〔Proc. IRE40 (1952), 1098～1101〕，这与设计极小化信息长度的代码有关。同一思想也独立地由塞恩·齐默恩曼(Seth Zimmerman)发表了〔*AMM*66 (1959), 690～693〕。

在2.3.4.1到2.3.4.5小节中，已就具体的课题方面，引述了若干值得注意的最近讨论树结构理论的论文。

关于树数学性质的进一步的讨论，请参看下列文献以及它们的文献目录：

Claude Berge, Théorie des graphes et ses applications, Dunod Paris, 1958;〔法〕C. 贝尔热，图的理论及其应用，李修睦译，上海科学技术出版社，1963；英译本译者

Alison Doig, The Theory of Graphs (London: Methuen, 1962). 第 16 章和 17 章。

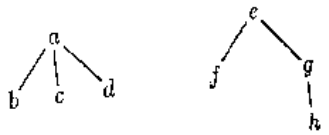
弗兰克·哈拉里,《图论》(Graph Theory)(马萨诸塞:爱迪生-韦斯利,1969)。第 4 章。

奥伊斯登·奥尔(Øystein Ore)《图形的理论》(Theory of Graphs)(美国数学学会,1962)第 4 章。

约翰·赖尔登,《组合分析引论》(Introduction to Combinatorial Analysis)(纽约:威利,1958)。第 6 章。

2.3.5 列表和废料收集

在 2.3 节开头,我们已把一个列表定义为"0 个或更多个的原子或列表的有限序列"。任何森林,是一个列表;例如,

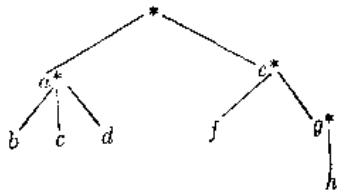


(1)

可以认为是列表

$$(a:(b,\ c,\ d),\ e:(f,\ g:(h)))\tag{2}$$

而对应的列表图式将是



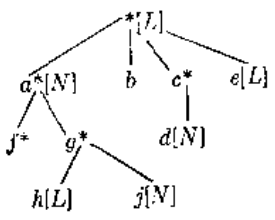
(3)

在这里读者应当复习早先给出的对于列表的介绍,特别是 2.3 节开始处的(3),(4),(5),(6),(7)。回忆一下,出现在上边(2)中的记号"a:"意味着:列表(b, c, d),除了它的结构信息,即它是三个原子 b, c, d 的一个列表之外,它是以属性"a"来"标记"的。这与我们一般的约定,即一树的每个节点可以包含除了它的结构联系之外的信息,是相容的。然而,正如我们在 2.3.3 节中对于树的讨论那样,完全可能而且有时也希望坚持所有的列表都不带标号,从而所有的信息都出现在原子中。

尽管任何森林都可以看成是一个列表,但反过来却不对。下列的列表也许比(2)和(3)更为典型,因为它说明了树结构的限制是如何被违背的:

$$L=(a:N,\ b,\ c:(d:N),\ e:L);\ N=(f:(\ ),\ g:(h:L,\ j:N))\tag{4}$$

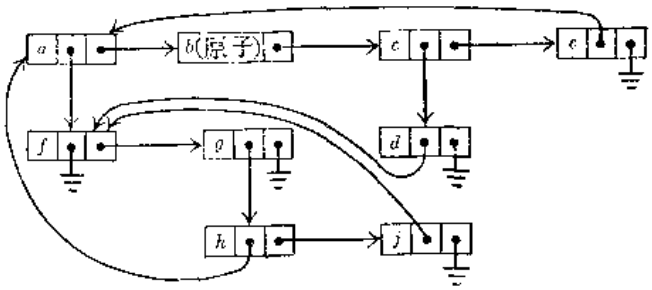
它可以表示为图式



(5)

〔参照 2.3 节的图式（7），这些图式的形式不必看得太认真了〕。

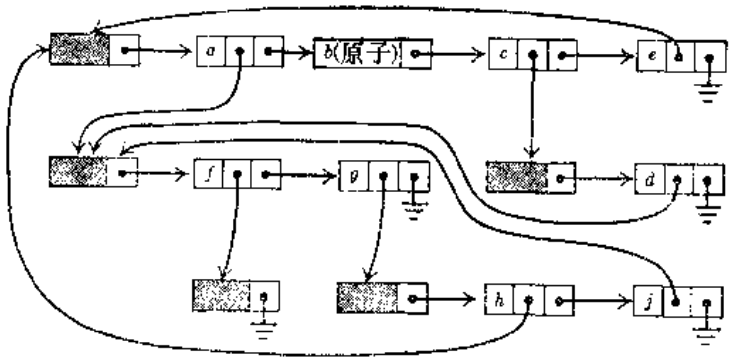
如同我们可以预期的那样，为在一台计算机内来表示列表结构，方法有许多种。按照用二叉树来表示一般树森林的同一基本思想，通常就得出这样一种表示方法：一个场 RLINK，比如说，用来指向一个列表的下一个元素，以及另一个场 DLINK，可以用来指向一个子列表的头一个元素。通过对 2.3.2 小节所述的内存表示作一自然的推广，我们将表示列表（5）如下：



（6）

遗憾得很，这个简单的思想，即便对于最普通的列表处理的应用，也并不十分合适。例如，假设我们有列表 $L=(A,\ a,\ (A,\ A))$，它包含着对另一个列表 $A=(b,\ c,\ d)$ 的三次访问。典型的列表操作之一，是消去 A 最左边的元素，使得 A 变成 $(c,\ d)$；但这要求三次变动 L 的表示，如果我们使用（6）中的技术的话。因为，每个指向 A 的指针就指着正要被删去的元素 b。考虑一会之后，读者将确认，仅仅因为 A 的头一个元素要被删去，就来改变对 A 的每一次访问的指针，是极其不希望的〔注意：在这个例子中，我们可以来试验这样一项技巧，就是，通过把整个元素 c 复写到以前由 b 所占据的位置，而后再删去老的元素 c，借以假定没有指向元素 c 的指针。但这项技巧，当 A 失去它的最后一个元素而变成空时，就不能进行工作了〕。

由于这个原因，表示方案（6）通常为另一方案所代替。方法类似，但使用了一个列表头来开始每个列表，如 2.2.4 小节中所介绍的那样。每个列表包含一个称为它的列表头的附加的节点，使得配置（6）将被表示成，比如说，图式（7）这样的形式



（7）

引进这些打头的节点实际上并不真正浪费内存空间，因为对于那些明显地不用的场〔即图式（7）中的带阴影的区域〕，一般有许多的用法。例如，作计访问次数的单元，或是指向该列表右末端的指针，或者是一个字符的名称，或者是一个辅助某遍历算法工作的“便笺式的”场，等等。

注意在我们原先的图式(6)中，包含 b 的节点是一个原子，而包含 f 的节点则确定一个空的列表。这两件事在结构上是相同的。因此读者有充分的理由来问，为什么我们不全然都说成是"原子"呢？不失一般性，我们可以把列表定义成仅仅是"0个或多个列表的一个有限序列"，但伴以我们通常的约定，即一列表的每个节点，可以含有除它的结构信息之外的数据。这个观点肯定是站得住脚的，而且它使得"原子"的概念相形见拙。不过，特别挑出原子来，正象我们已做过的那样，也是有充分的理由的，如果考虑到有效地利用计算机内存的话。因为，原子不须要经受为列表所需要的同一种类的通用处理。内存表示(6)说明，在一个原子节点 b 中，用于信息的空间可能比列表节点 f 中的更大；而且当列表头的节点也象(7)那样出现时，对于节点 b 与 f 的内存要求就有很大的区别。因此，引进原子的概念主要是为了便于有效地利用计算机的内存。典型的列表含有比我们例子所指出的更多的原子。例子(4)～(7)，企图说明可能有的复杂性，而不是通常的简单性。

一个列表本质上只不过是一个其元素可以包含指向其它列表的指针的线性表。我们希望对列表实施的普通操作，也就是通常对线性表所需要的那些（建立，毁坏，插入，删去，分开，连接），加上一些主要是与树结构有关的进一步的操作（复写，遍历，嵌套信息的输入与输出）。为此，对于内存中表示链接线性表的三项基本技术——即直接的，循环的，或双重链接的——任何一项，均可采用，不过对于所采用的不同算法有不同的效果。关于这三种类型的表示，图式（7）可以如（8）中所指出的那样，出现在内存中。

内存	直接链接			循环链接			双重链接			
单元	INFO	DLINK	RLINK	INFO	DLINK	RLINK	INFO	DLINK	LLINK	RLINK
010:	—	表头	020	—	表头	020	—	表头	050	020
020:	*a*	060	030	*a*	060	030	*a*	060	010	030
030:	*b*	原子	040	*b*	原子	040	*b*	原子	020	040
040:	*c*	090	050	*c*	090	050	*c*	090	030	050
050:	*e*	010	Λ	*e*	010	010	*e*	010	040	010
060:	—	表头	070	—	表头	070	—	表头	080	070
070:	*f*	110	080	*f*	110	080	*f*	110	060	080
080:	*g*	120	Λ	*g*	120	060	*g*	120	070	060
090:	—	表头	100	—	表头	100	—	表头	100	100
100:	*d*	060	Λ	*d*	060	090	*d*	060	090	090
110:		表头	Λ	—	表头	110	—	表头	110	110
120:	—	表头	130	—	表头	130	—	表头	140	140
130:	*h*	010	140	*h*	010	140	*h*	010	120	140
140:	*j*	060	Λ	*j*	060	120	*j*	060	130	120

(8)

这里的"LLINK"，在双重链接表示下用作一个指向左边的指针。注意在所有三种形式下，其 INFO 和 DLINK 场皆相同。

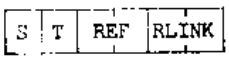
在这里已经不需要对这三种形式的任何一种，来重复列表处理的算法，因为算法思想与我们在这一章中已多次见过的那些相同。但是，应该注意到以下关于列表的几个要点，

它有别于我们以前处理的比较简单的情况：

1）在上述内存表示中隐含了原子节点与非原子节点是有区别的；而且，当正在使用的是循环表或双重链接表时，还需要把表头节点与其它节点区别开来，用以辅助对于该列表的遍历。因此每个节点一般地都包含一个 TYPE 场，它指出该节点所表示信息的种类。这个 TYPE 场也常用来区别各种类型的原子（例如，为了在打印或显示答案时用来区别字符数据，整型数据，或浮点数据）。

2）为了用 MIX 计算机以进行一般的列表操作，可能有多种方式来设计节点的格式。以下是其中的两个例子：

a）可能的单字格式，假定所有的 INFO 都出现在原子中：



(9)

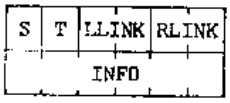
S(标志)："标志位"，用于"废料收集"（见下面）。

T(类型)：对于列表头 T = 0；对于子列表元素 T = 1；对于原子 T > 1。

REF：当 T = 0 时，REF 是一个"访问计数"(见下面)；当 T = 1 时，REF 指向问题中子列表的列表头；当 T > 1 时，REF 指向一个含有一个标志位和五个字节的原子信息的节点。

RLINK：如（8）那样，用作直接或循环链接的指针。

b）可能的双字格式：



(10)

S，T：与（9）相同。

LLINK，RLINK：如(8)，用作双重链接的通常指针。

INFO：与这个节点相结合的全字长信息；对于一个表头节点，这可以包括：一个访问计数，一个指向列表内部的运行指针，以便于线性遍历，以及一个字符名称，等等。当 T = 1 时，这个信息包括 DLINK。

3）显然，列表是非常一般的结构；似乎完全可以说，只要加上适当的约定，任何结构无论如何都可以表示为一个列表。鉴于列表的这种普遍性，已有大量的程序设计系统，被设计成便于进行列表的操作，而且在任何计算机装置中通常都有若干个这样的系统可供利用。这些系统都是在节点的某种通用格式的基础上，如象上面的（9）或（10)那样，为了列表操作的灵活性而设计的。实际上，很显然，这种通用格式，一般说来并不是适合于特殊应用的最佳格式，并且对特殊问题而言使用通用程序的处理时间，比起用手工"特制"的系统所花的时间，要显著地多。例如，很容易看出，在这一章里迄今我们所做过的几乎所有的应用中，如果改换了在每种情况下所给出的节点格式，都受到象（9）或（10）中一般列表表示的牵累。一个列表处理程序在其处理节点时，必须经常地检查场 T，然而迄今在我们的任何程序中是不需要的。效率的损失，在许多情况下，是以使用通用系统相对

地简化了程序设计和减少了调试时间为抵偿的。

4）列表处理算法与这一章以前给出的算法之间，有着一个极其显著的差别。因为一个单一的列表可能包含在许多其它的列表之中，所以什么时候一个列表应该被恢复到可利用的存贮池去，这是不清楚的。迄今我们所有的算法总是每当 NODE(X) 已不再需要时，就说“AVAIL⇐X”。但由于在一个程序运行的同时，一般的列表可能以一种不可预测的方式生长和消失，通常就十分难以确切地说出什么时候某个节点是多余的，因此，列表的维持可利用空间的问题，比起以前所考虑的简单情况来说，要困难得多。我们将在本小节的后面，集中讨论这个问题。

让我们想象，我们正在设计一个通用的将为数以百计的其他程序员所使用的列表处理系统。为了维持可利用的空间表，已经提出了两个主要的方法：使用访问计数器和废料收集。访问计数器技术是在每个节点中使用一个新的场，它包含着有多少个箭头指向这一节点的计数。随着程序的运行，这样一个计数是比较容易维持的，每当它变成 0 时，则所述的那个节点即成为可利用的。另一方面，废料收集技术，是在每个节点中增加一个新的一位的场，称之为“标志位”。支配这一情况的思想是，几乎所有的算法，都写成不把任何节点恢复成自由存储，并让程序高兴地一直运行下去，直到所有的可利用存储都用光为止。然后，一个“反复循环的”算法，利用标志位来把当前不被访问的节点恢复成可利用的存储，进而程序又继续运行。

这两种方法，无论那一种都不是完全令人满意的。访问计数器方法的主要缺点，在于它并不总能释放所有可利用的节点。它对于重迭的列表工作得很好；但对于递归的列表，象（4）中的例子 L 和 N，通过访问计数器的技术，是不会还给存储器的。它们的计数将是非 0 的（因为它们访问自身），甚至没有别的访问该运行程序的列表指向它们时也是如此。而且，访问计数器方法在每个节点中使用了好大一块空间(尽管由于计算机字的大小，有时这份空间总是办得到的)!

使用废料收集技术的困难，除了每一节点中损失一位之外，当几乎全部内存空间都在使用的时候，它的运行则非常缓慢；而且在这样的情况下，通过回收过程所找到的自由存储单元，其数量是微不足道的。超过了存储容量的那些程序（以及许多未调试的程序），在存储刚好被最后穷尽之前，经常要浪费大量时间来多次地调用废料收集程序，而且几乎毫无成效。关于这个问题的一个部份的解决办法，是让程序员确定一个这样的数 k，即当运行一次废料收集程序所找到的自由节点为 k 个或更少时，他就不希望再继续进行处理了。进一步的问题是，有时候难于准确地确定，在一个给定的阶段上什么样的列表不是废料；如果程序员已经使用任何非标准技术，或者已经在一些不寻常的位置中保留着任何指针值，则废料收集程序出错的机会就很大。由于下述事实，在计算机程序调试的历史中，会引起某些不可思议的事情：正当运行着一些前已多次工作过的程序期间，废料收集却在一个意外的时刻突然地闯入了。废料收集还要求程序员要在所有的指针场中始终保持正确的信息，尽管在那些决不为程序访问的场（例如，在一个排队中的后尾节点的链接，见习题 2.2.3-6）中，置以无意义的信息，通常也是方便的。我们还需要指出，废料收集对于“实时的”应用是不适宜的，因为即使废料收集程序不是频繁进行的，但在运行的时刻却要求大量的计算机时间（见习题 12）。

虽说废料收集对于每个节点都要求一个标志位，但也有可能把所有的标志位一起组装到另一个存储区域中，从而保留一份单独的表格，使得节点的单元与其标志位之间建立起一个适当的对应关系。在某些计算机上，这一思想可能导致另一种处理废料的方法，比在每个节点中设立一位标志的方法更具有吸引力。但在许多其它计算机上，它却使废料收集过程更为缓慢。

约·韦曾鲍姆（J. Weizenbaum）对于访问计数技术提出了一个有趣的修改。利用双重链接列表结构，把访问计数器仅仅放在每个列表的表头处。这样，当某些指针变量遍历一个列表时，它们并不被包括在对于个别节点的访问计数中；但由于程序员知道对于整个列表维持访问计数的规则，所以他（在理论上）就知道怎样才能避免去访问其访问计数已为0的任何列表。程序员也有能力于脆不顾访问计数而把某些列表恢复到可利用的存储。这些思想要求程序员谨慎从事；没有经验的程序员掌握这些思想，本来是有些危险的，而且由于可能访问已被抹去的节点致使程序调试更为困难。韦曾鲍姆方法最精采的部分，是对那些其访问计数刚刚变成0的列表的处理：这样一个列表被附加到当前可利用的空间表的末尾——这对于双重链接表是容易做到的——而且仅仅在所有以前可利用的单元已经用完之后，它才被考虑作可利用的空间；然后随着这个列表的个别节点变成可利用的，被它们访问的诸列表的访问计数器减1。这个抹去列表的延迟的动作，就运行时间而论是十分有效的；但是它却会助长不正确的程序正确地运行一会儿！有关的进一步细节，请见*CACM* 6(1963), 524～544。

废料收集的算法，由于种种原因，是十分值得注意的。首先，当我们要求“标志被一个给定节点直接或间接地访问的所有节点”时，这样一个算法还是有用的（例如，我们可能要求找出所有的被某个子程序直接地或间接地调用的子程序；参看习题2.2.3-26。还请看第7章中的“祖先算法”）。

废料收集一般分两个阶段进行。我们假定所有节点的标志位开始时皆为0（或者我们把它们都置成0）。现在，头一个阶段，是从那些可为主程序直接访问的节点开始，标出所有的非废料节点。第二个阶段对整个内存储池区域进行一遍顺序的扫瞄，把所有未标志的节点都放进自由空间的表中。标志阶段是最有趣的，我们将为之倾注我们的注意力。第二阶段则有许多变化，致使该阶段也不平凡；见习题9。

废料收集最有趣的特性，在于下列事实：每当这个算法运行的时候，我们仅有非常有限的可利用的存储，能用来控制我们的标志算法。这个使人感兴趣的问题要在以下的讨论中才能剖析清楚。困难在于，大多数人当他们头一次听到关于废料收集的思想时，是不能对它进行评价的，加之多年以来，对它也没有完善的解决办法。

下列的标志算法也许是最明显的一个：

算法A（标志） 设用作列表存储的全部内存是NODE(1), NODE(2), …, NODE(M)，又假设这些字或者是“原子”，或者包含两个场ALINK和BLINK。假定所有节点开始时都是未标志的。这个算法的目的就是，从一组“直接可访问的”节点开始，来标志所有这样的节点，即能由非原子节点中的ALINK和/或BLINK指针的一条链达到的节点。

A1. 〔初始化〕 标出所有“直接可访问的”节点，即由主程序中某些固定的单元所指

出的节点，它们被用作对所有的内存访问的一个起源。置K←1。

A2.〔NODE(K)导致另一个吗?〕 置K1←K+1。如果NODE(K)是一个原子或一个未标志者，则转到步骤A3。否则，如果NODE(ALINK(K))是未标志的，就去标出它，而且如果它不是一个原子，则置K1←min(K1,ALINK(K))。类似地，如果NODE(BLINK(K))是未标志的，则去标志它，而如果它不是一个原子，则置K1←min(K1,BLINK(K))。

A3.〔已完成吗?〕 置K←K1。如果K≤M，则返回步骤A2；否则算法终止。 ∎

贯穿在这个算法以及在该小节内紧接着的一些算法中，为方便起见，我们始终都将假定，不存在的节点"NODE(A)"是"已标志的"（例如，ALINK(K)或BLINK(K)在步骤A2中可以等于A)。

算法A的一种变形，是在步骤A1中置K1←M+1，从步骤A2中撤消操作"K1←K+1"，而代之以把步骤A3改变成为

"A3′.〔已完成?〕 置K←K+1。如果K≤M，则返回步骤A2。否则如果K1≤M，则置K←K1和K1←M+1并返回步骤A2。否则算法结束。"

要对算法A给出一个精确的分析，或者要确定它比刚才描述的变形更好还是更坏，都是非常困难的，因为眼前并没有什么有效的方式来描述输入的概率分布。但我们可以说，在最坏的情况下，它所花费的时间与nM成比例，其中n是它所标志的单元数；而且，一般地说，我们可以确信，当n很大时，它是非常缓慢的。为使得废料收集能成为一个可用的技术，算法A实在是太慢了。

另一个较明显的标志算法是，沿着所有的通路走下去，并随着我们的行进，把分支点记录于一个堆栈上：

算法B(标志) 这一算法能达到与算法A相同的效果，它使用STACK〔1〕，STACK〔2〕，…作辅助存储，来记住所有尚未走完的通路。

B1.〔初始化〕 命T是直接可访问的节点的个数；标志这些节点并把指向它们的指针放进STACK〔1〕，…，STACK〔T〕中。

B2.〔堆栈空?〕 如果T=0，则算法结束。

B3.〔撤消顶上的条款〕 置K←STACK〔T〕，T←T-1。

B4.〔检查链接〕 如果NODE(K)是一个原子，则返回B2。否则，如果NODE(ALINK(K))是未标志的，则标志它并置T←T+1，STACK〔T〕←ALINK(K)；如果NODE(BLINK(K))是未标志的，则标志它并置T←T+1，STACK〔T〕←BLINK(K)。返回B2。 ∎

显然，算法B的执行时间实质上同它所标志的单元个数成正比，这简直就是我们所期望的了；可惜，它并不能真正用于废料收集，因为没有位置来保持堆栈！并非没有根据，假定算法B中的堆栈增长到比如说，占内存大小的百分之五；当废料收集过程被调用到时，而所有可利用的空间已经用完了，就只能有固定数量（比较少）的单元用作这样的一个堆栈。大多数早期的废料收集过程，实质上都是以这个算法为基础的，而且如果特殊的堆栈空间已经用完，则整个程序即告终止。

通过把算法A与B组合在一起，使用一个固定大小的堆栈，就可能有一个稍微好一点

的方案。

算法C（标志） 这个算法与算法A和B有相同的效果，它使用H个单元的一张辅助表格STACK[0]，STACK[1]，…，STACK[H−1]。

在这个算法中，动作"把X插入堆栈"意义如下："置T←(T+1)mod H和STACK[K]←X。如果T=B，则置B←(B+1) mod H和K1←min(K1,STACK[B])。"(注意T指向堆栈当前的顶，而B指向当前底的下一个位置；STACK实质上是作为一个受限输入的双排队进行操作的)。

C1. [初始化] 置T←H−1，B←H−1，K1←M+1。标志所有直接可访问的节点，并逐次地把它们的单元插入到堆栈上（就象刚才上边所述的那样)。

C2. [堆栈空?] 如果T=B，则转到C5。

C3. [撤消顶上条款] 置K←STACK[T]，T←(T−1)mod H。

C4. [检查链接] 如果NODE(K)是一个原子，则返回C2。否则，如果NODE(ALINK(K))未标志，则标志它并把ALINK(K)插入到堆栈上。类似地，如果NODE(BLINK(K))未标志，就标志它并把BLINK(K)插入到堆栈上。返回C2。

C5. [扫描] 如果K1>M，则算法终止（变量K1表示这样的最小的单元：即是在该处存在有一个新的节点，它可能导致一个应该加以标志的节点)。否则，如果NODE(K1)是一个原子或未加标志的，则K1增加1并重复这个步骤。如果NODE(K1)已标志，则置K←K1，K1增加1，并转到C4。 ∎

如果当NODE(X)是一个原子时，不把X放到堆栈上的话，则该算法和算法B都能加以改进；这样的修改是直接了当的，而且它们已被略去，以免使这些算法不必要地复杂化。

算法C当H=1时实质上就是算法A，而当H=M时实质上就是算法B；显然，当H越来越大时，它就逐渐地更为有效。遗憾的是，算法C，由于与算法A同样的理由，而缺乏一个精确的分析；而且我们对于H应该有多大才能使得这个方法足够快，还没有很好的见解。我们说，在大多数应用中，为使算法C可用于废料收集，一个H=50的值就足够了，这也是似是而非的，并没有什么可以告慰的。

算法B和C都使用一个保留在顺序的内存单元中的堆栈。我们在这一章的早些时候已经看到，链接的内存技术适合于维持在内存中不连续的堆栈。这就提出了这样一种思想，即我们可以这样来维持算法B的堆栈，即设法将其散列于我们正在收集废料的同一个内存区域中。只要我们能给予废料收集程序一点位置以进行活动的话，这将是容易做到的。假如说，所有的列表都被表示成（9）中那样，只是列表头节点的REF场被用作废料收集之目的，以替代访问计数。那么，我们就能重新设计算法B，使得该堆栈得以维持在表头节点的REF场中。

算法D(标志) 这个算法与算法A，B，及C有同样的效果，但它假定节点有如上所述的S、T，REF，和RLINK场，以代替ALINK和BLINK。S场用作标志位，S(P)="−"意味着NODE(P)是标志了的。

D1. [初始化] 置TOP←A。然后对于每一个指向直接可访问列表的头的指针P(参照算法A的步骤A1)，如果S(P)="+"，则置S(P)←"−"，REF(P)←TOP，TOP←P。

D2. 〔堆栈空?〕 如果 TOP＝Λ，则算法终止。

D3. 〔撤消顶上条款〕 置P←TOP，TOP←REF(P)。

D4. 〔通过列表〕 置P←RLINK(P)；然后如果P＝Λ，或T(P)＝0，则转到D2.否则置S(P)←"−"。如果T(P)＞1，则置S(REF(P))←"−"(由此标志原子信息)。否则(T(P)＝1)，置Q←REF(P)；如果Q≠Λ且S(Q)＝"+"，则置S(Q)←"−"，REF(Q)←TOP，TOP←Q。重复步骤D4。 ▌

算法D可与算法B进行对照，它们十分相似，而且该算法的运行时间实质上也正比于标志的节点数。然而，不能无条件地推荐算法D，因为它的颇为温和的限制，对于一般的列表处理系统来说，还太严厉了。每当废料收集过程被调用进行工作时，这个算法，实质上要求所有的列表结构都是非常工整的〔如在(7)中那样的〕。但列表处理的算法，可暂时地许可列表结构不是太工整的，而且重要的是在这些短暂的期间内，象算法D那样的废料收集程序将不能使用。况且，有若干列表处理算法，它们竟恣意地在它们操作的期间大肆破坏列表中的链接场，纵然它们已被设计成在该算法业经完成之后，再次恢复工整的列表。当程序含有指向一个列表中间的指针时，在步骤D1中也还必须小心在意。

这些考虑，就把我们引导到算法E，它是1965年时由彼得·多伊奇(Peter Deutsch)和由赫伯特·肖尔(Herbert Schorr)及威·麦·韦特(W.M.Waite)独立地发现的一个卓越的标志方法。在这个算法中所用的假定，与算法A到D只有很少的差别。

算法E(标志) 假定给了一组节点，有下列诸场：

MARK(一位场)，

ATOM(另一个一位场)，

ALINK(指针场)，

BLINK(指针场)。

当ATOM＝0时，ALINK和BLINK场可以包含Λ或一个指向另一个相同格式节点的指针；当ATOM＝1时，ALINK和BLINK场的内容与本算法无关。

给定一个指针PO，该算法置NODE(PO)中的MARK场为1，同时对于从NODE(PO)通过链接具有ATOM＝MARK＝0的节点中的ALINK和BLINK指针所能达到的每一个其它节点中的MARK场也置成1。这个算法使用三个指针变量T，Q，和P，并且在其执行期间，以这样一种方式来修改诸链接和诸控制位，即所有的ATOM，ALINK，和BLINK场在完成之后都被恢复成它们原来的设置，尽管它们可能被暂时地改变。

E1. 〔初始化〕 置T←Λ，P←PO(在这个算法的余下部分，变量T有双重的意义：当T≠Λ时，它实质上指向一个如同算法D中那样的堆栈的顶；而且曾为T指向的节点，在当前占有NODE(T)的"人为的"堆栈链接的位置上，包含了一个等于P的链接)。

E2. 〔标志〕 置MARK(P)←1。

E3. 〔原子?〕 如果ATOM(P)＝1，则转到E6。

E4. 〔下降ALINK〕 置Q←ALINK(P)。如果Q≠Λ且MARK(Q)＝0，则置ATOM(P)←1，ALINK(P)←T，T←P，P←Q，并转到E2(这里暂时地改变

ATOM 场和 ALINK场，使得该列表结构在某些已标志的节点中能被相当激烈地改变。但这些改动将在步骤E6中被恢复)。

E5. 〔下降 BLINK〕 置 Q←BLINK(P)。如果Q≠A且 MARK(Q)= 0，则置 BLINK(P)←T，T←P，P←Q，并转到E2。

E6. 〔上升〕 (这一步骤恢复在步骤E4或E5中所做的链接转换；ATOM(T)之设置告知要恢复的是ALINK(T) 还是BLINK(T))。如果T = A，则算法终止。否则置 Q←T。如果 ATOM(Q)= 1，则置 ATOM(Q)←0，T←ALINK(Q)，ALINK(Q)←P，P←Q，并返回到E5。如果 ATOM(Q)= 0，则置T←BLINK(Q)，BLINK(Q)←P，P←Q，并返回到E6。 ∎

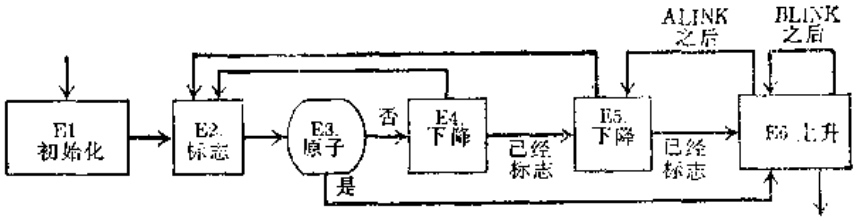


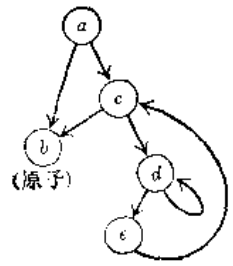
图38 算法E的框图

该算法的一个正在动作的例子，见于图 39，它说明了对于一个简单的列表结构所遇到的逐次的步骤。读者将发现，算法E是值得非常仔细地进行研究的；注意为了保持类似于算法D中那样的堆栈，其链接结构在步骤E4和E5中是怎样人为地改变的。当我们返回到一个以前的状态时，ATOM 场即用来告知 ALINK，BLINK 中的哪一个包含着这人为的地址。图 39 底下所示的“嵌套”，说明在执行算法E的期间，每个非原子节点是如何被访问了三次的〔于是，在步骤E2，E5，和E6的开始处，出现了同样的配置(T，P)〕。

对有待标志的节点数运用归纳法，可以证明算法E是正确的。同时，人们还证明，在该算法结束时，P = PO；欲求其详，请见习题 3。如果删去步骤E 3，并且取代特殊的测试“ATOM(Q)= 1”，而在步骤E4和E5中进行一些适当的动作，以及在步骤E 1 中进行一个“ATOM(PO)= 1”的测试的话，则算法E将会进行得更快。为简便起见，我们仅以其现有的形式叙述了该算法；刚才所指出的修改，出现在习题 4 的答案中。

算法E中所运用的思想，也可应用于废料收集以外的某些问题。事实上，在习题 2.3.1-21 中已经提到它对于树遍历的使用。读者还可以发现，把算法E与习题 2.2.3-7中所解决的较简单的问题加以对照，是有益的。

在我们已讨论的所有标志算法中，仅仅算法D可直接地应用于如(9)中所表示的列表。而其余算法，则全都要测试一个给定的节点是否为一个原子，而(9)的约定与这样的测试是不相容的，因为(9)的约定允许原子信息填入除标志位外的一个整字。然而，



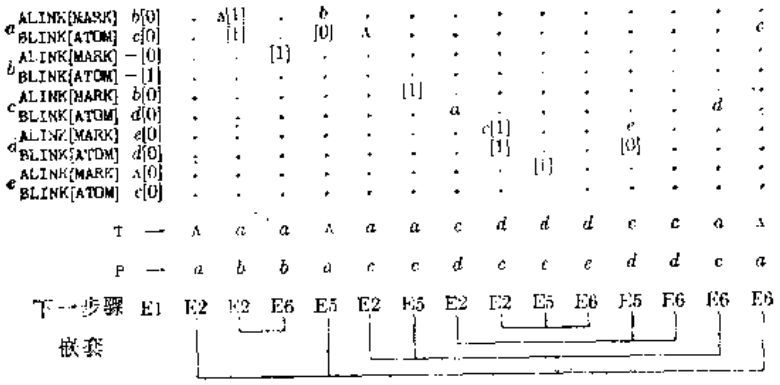


图39 一个通过算法E标志的结构（这个表格仅仅示出自前一步以来所已出现的变化。）

其余每个算法，均可加以修改，使得原子数据与字中的指针数据相区别时，其中的指针数据是链接到字的，而不是着眼于字本身，那么，这些算法将能进行工作。在算法A或C中，我们可以简单地避免标志原子字，直到所有的非原子字已经被适当地标志了为止；然后对所有数据的进一步扫瞄就足以来标志所有的原子字。算法B甚至更易于修改，因为我们仅仅需要不让原子字接近堆栈。采用算法E几乎同样简单，尽管当ALINK和BLINK两者都允许指向原子数据时，有必要在非原子节点中引进另一个一位的场。一般说来这是不难做到的（例如，当每个节点有两个字时，每个链接场的最低有效位便可用来存储临时信息）。

虽然算法E需要一个与其所标志的节点数成正比的时间，但这个比例常数不象在算法B中那样小；已知的最快的废料收集方法，是习题5中所讨论的，把算法B和E组合在一起的方法。

现在让我们试着对废料收集的有效性作某种定量的估计。这是针对本章前面的大多数例子中所用的“AVAIL←X”这一原则进行的。在先前每种情况下，我们均可省略所有关于把节点恢复到自由空间的特定的叙述，并且可以替换成一个废料收集程序。在专门应用中，与在一组通用的列表处理子程序中相反，对一个废料收集程序来进行程序设计和调试，是比我们所已使用的方法更为困难的，而且，当然，废料收集还要求在每一个节点中保留有额外的一位；但是，一旦这些程序已被写成和调试过，那么，所关心的则是这些程序的相对速度了。

已知的最好的废料收集程序有一个形式为c_1N+c_2M的执行时间，其中c_1和c_2是常

数，N是被标志的节点数，而M是内存中的节点总数。于是M－N是所找到的自由节点的个数，为把这些节点恢复成自由存储的时间是，每一个节点$(c_1N+c_2M)/(M-N)$。命$N=\rho M$；则这个数字即变成$(c_1\rho+c_2)/(1-\rho)$。所以如果$\rho=\frac{3}{4}$，即如果存储器是四分之三满，则每个自由节点恢复到存储，要花费$3c_1+4c_2$个时间单位；当$\rho=\frac{1}{4}$时，则对应的数量就只是$\frac{1}{3}c_1+\frac{4}{3}c_2$。如果我们不使用废料收集技术，则恢复每个节点的时间实质上是一个常数c_3，而且说不定c_3/c_1将非常之大。因此我们可以看出，当内存变满时，废料收集低效到什么程度，而当对于内存的需要很轻快时，相应地它又是如何之有效。

可以把废料收集，与某些其它的把单元恢复成自由存储的方法结合起来；这些思想都不是相互排斥的，而且某些系统既采用了访问计数器，又使用了废料收集方法，此外还允许程序员明确地抹去节点。其想法是，每当所有其它的恢复单元的方法都失败时，便使用废料收集作为“最后的一着”〔见约·韦普鲍姆的讨论，*CACM* 12(1969)，370～372〕。

列表的一个顺序的表示是可能的，这种表示，以更复杂的存储管理为代价，节省了许多的链接场；见威·詹·汉森（W．J．Hansen），*CACM* 12(1969)，499～506，以及C．J．切尼（C．J．Cheney），*CACM* 13(1970)，677～678。

习题

▶1．〔*M*21〕 在2.3.4小节中，我们看到：树，乃是一个有向图形的“经典的”数学概念的特殊情况。能否以图论的术语来叙述列表？

2．〔20〕 在2.3.1小节中，我们看到：在计算机内使用“穿线的”表示，能便利于树的遍历。能以类似的方式来对列表结构进行穿线吗？

3．〔*M*26〕 证明算法E的正确性〔提示：见算法2.3.1 T的证明〕。

4．〔28〕 试为算法E写出一个MIX程序，假定把节点表示成一个MIX字，且以MARK在（0:0）场〔＋＝0，－＝1〕，ATOM在（1:1）场，ALINK在(2:3)场，BLINK在（4:5）场，以及A＝0。并且，借助于有关的参数，来确定你的程序的执行时间(注意，在MIX计算机内，确定一个内存单元是包含－0还是＋0的问题，并不是完全不足道的，而这可能是你的程序中的一个因素)。

5．〔25〕（肖尔和韦特）给出如下一个把算法B和算法E结合起来的标志算法：采用算法E关于节点中诸场等的假定；然而，如同算法B那样，使用一个辅助的堆栈STACK〔1〕，STACK〔2〕，…，STACK〔N〕，而且仅当堆栈满了时，才使用算法E的机制。

6．〔00〕 这一小节末尾的定量讨论，说是废料收集花费近似于c_1N+c_2M个时间单位；这“c_2M”项是从那儿得来的？

7．〔24〕（罗·威·弗洛伊德）试设计一个类似于算法E的标志算法，其类似性在于：它不使用辅助堆栈，但是（a）它有一个更困难的任务要做，即每个节点仅含有MARK，ALINK，和BLINK场（因而没有ATOM场来提供附加的控制），不过（b）它也有一个更容易的任务要做，即它仅仅标志一个二叉树而不是一个一般的列表。这里ALINK和BLINK是一个二叉树中的LLINK和RLINK。

▶8.〔27〕（劳·彼·多伊奇）试设计一个类似于算法D和E的标志算法，它们的类似性在于：不使用辅助存储作为堆栈，但要修改这方法，使得它以可变大小的节点和以可变个数的有着下列格式的指针来进行工作：一个节点的头一个字有两个场MARK和SIZE；MARK场如同算法E中那样来处理，SIZE场包含一个数$n \geqslant 0$。这个数意味着在头一个字之后，有n个连续的字，每一个字含有两个场：MARK（它为0并且始终如此）和LINK（它为A或指向另一个节点的头一个字）。例如，一个具有三个指针的节点，应该由四个连续的字组成：

头一个字	MARK = 0	（将要被置成1）	SIZE = 3
第二个字	MARK = 0		LINK = 头一个指针
第三个字	MARK = 0		LINK = 第二个指针
第四个字	MARK = 0		LINK = 第三个指针

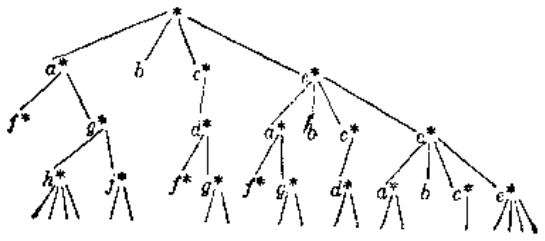
你的算法应该标志从一个给定的节点PO所能到达的所有节点。

▶9.〔28〕（丹·爱德华兹（D.Edwards））试对废料收集的第二阶段设计一个算法，在下列意义下它"使存储紧凑"：设NODE(1)，…，NODE(M)是一个字的节点，具有如算法E中所述的场MARK，ATOM，ALINK，以及BLINK。假定在所有不是废料的节点中MARK = 1。所求的算法应当在必要时，重新对已标志的节点进行定址，使得它们都出现在连续的单元NODE(1)，…，NODE(K)中，同时在必要时应该改变非原子节点的ALINK和BLINK场，使保持该列表的结构。

▶10.〔28〕试设计一个复写列表结构的算法，假定正在使用的是象（7）那样的内部表示〔于是，如果你的过程所要求的复写表头是（7）的左上角处节点的列表，则一组新的有着14个节点的，并且具有与（7）中所示结构和信息相等同的列表，将被建立起来〕。

假定该列表结构，利用象（9）中的S，T，REF，和RLINK场而存储于内存中；而且NODE(PO)是有待复写的列表的表头。进一步假定在每个列表头的节点中，REF场是Λ；为避免需要附加的存储空间，你的复写过程应当利用REF场（并且以后再次把它们恢复成Λ）。

11.〔M30〕任何列表结构，通过重复所有重迭的元素直到不再有剩余时为止，均可"完全地展开"成一个树结构；当列表是递归的时，这就给出一个无限的树。例如，列表（5）可展开成一个无限树，它的头四层为



试设计一个算法，来测试这两个列表结构的等价性，这种等价性的意义是：当把它们完全地展开时，它们有着相同的图式〔例如，在这个意义下，如果

$$A=(a:C, b, a:(b:D))$$
$$B=(a:(b:D), b, a:E)$$
$$C=(b:(a:C))$$
$$D=(a:(b:D))$$
$$E=(b:(a:C))$$

则列表A与B是等价的〕。

12.〔30〕(马·明斯基(M. Minsky))说明：在一个“实时的”应用中，例如一台计算机正控制着某部物理装置时，有可能可靠地使用废料收集方法。甚至对于执行每个列表操作所要求的极大执行时间，提出了严格的上限〔提示：如果相当谨慎，废料收集可安排与列表操作平行地工作〕。

2.4 多重链接结构

既然我们已经详尽地考察了线性表和树结构，则在一台计算机内表示结构信息的原理，已属了然。在这一节里，我们将来考察这些技术的另一个应用。这一次，是关于其中结构信息稍微复杂一些的典型情况：在“高级”应用中，通常同时出现多重类型的结构。

“多重链接结构”涉及到在每个节点中具有多个链接场的节点，而不象我们以前的大多数例子那样，仅仅有一个或两个链接场。我们已经见过某些多重链接的例子，例如，在2.2.5小节中所模拟的电梯系统和在2.2.3小节中的多变量多项式。

我们将看到，每个节点有多个不同类型的链接存在，伴随而来的算法未必比已研究的算法更难写或更难理解。我们还将讨论这样一个重要问题：“应有多少结构信息被明显的记录在内存中？”

我们将要考虑的问题，是同编写一个用于翻译COBOL和其它有关的语言的编译程序相联系而提出来的。一个使用COBOL的程序员，可以在若干层上对他的程序中的量给出字母名称；例如，他对卖出和买进可以有两个数据文件，并具有如下的结构：

```
1  SALES (卖出)
  2  DATA (日期)
    3  MONTH (月)
    3  DAY (日)
    3  YEAR (年)
  2  TRANSACTION (事务)
    3  ITEM (品名)
    3  QUANTITY (数量)
    3  PRICE (价格)
    3  TAX (税)
    3  BUYER (买者)
      4  NAME (姓名)
      4  ADDRESS (地址)
```

```
1  PURCHASES (买进)
  2  DATA (日期)
    3  DAY (日)
    3  MONTH (月)
    3  YEAR (年)
  2  TRANSACTION (事务)
    3  ITEM (品名)                    (1)
    3  QUANTITY (数量)
    3  PRICE (价格)
    3  TAX (税)
    3  SHIPPER (货主)
      4  NAME (姓名)
      4  ADDRESS (地址)
```

这个设置表示，在 SALES 中的每个项目由两个部分组成，即 DATE 和 TRANSACTION；DATE 进一步分成三个部分，而类似地 TRANSACTION 有五个小部分。类似的说明亦可应用于 PURCHASES。这些名称的相对次序指出，出现在文件的外部表示（例如穿孔卡片或磁带）中的数量的次序；注意在这个例子里，“DAY” 和 “MONTH”在两个文件中以相反的次序出现，一个 COBOL 程序员要进一步给出未在这个例子中示出的信息，告知每个信息项目占有多少空间以及它以什么格式出现，由于这些考虑在这节中与我们无关，所以我们将不进一步去涉及。

一个 COBOL 程序员首先描述他的文件的布局和在他程序中的其它变量，然后他给出处理这些量的算法。为了访问上述例子中的个别的变量，仅仅给出名称 DAY 是不充分的，因为还没有根据确定，这个叫做 DAY 的变量是在 SALES 文件里的呢，还是在 PURCHASES 文件里。因此给予 COBOL 程序员这样的能力，通过写“DAY OF SALES”来访问 SALES 项目的 DAY 部分。更完全些，他也可以写成

“DAY OF DATE OF SALES”

但是一般没有必要给出比避免二义性所必需的更多的限定。这样，

“NAME OF SHIPPER OF TRASACTION OF PURCHASES”

可以简写成为

“NAME OF SHIPPER”

因为仅有数据的一个部分被称为 SHIPPER。

COBOL 的这些规则，可以更精确地叙述如下：

a）每个名称的紧前面，是一个相关联的叫做它的“层号”的正整数。一个名称或者访问一个初等的项目，或者，它是一个由一个或多个项目组成的组的名称，这些项目的名称紧接其后。在后一种情况下，这个组的每个项目必须有相同的层号，该层号必须大于组名称的层号（例如，上例的 DATE 和 TRANSACTION 有层号 2，它大于 SALES 的层号 1）。

b）为访问一个名叫 A_0 的初等项目或者项目组，其一般形式为

$$A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_n$$

其中 $n \geqslant 0$，而且其中对于 $0 \leqslant j < n$，A_j 是某个项目的名称，这个项目直接或间接地包含在一个名称为 A_{j+1} 的组中。必须恰好有一个项目 A_0 满足这一条件。

c）如果同一个名称 A_0 出现在好几个位置上，则通过使用限定，必须有一个方法来访问这个名称的每一使用。

作为规则（c）的一个例子，数据构造

```
1 AA
  2 BB
    3 CC
    3 DD
  2 CC
```
(2)

将是不允许的，因为没有一个无二义性的方法来访问 CC 的第二个出现（见习题 4）。

COBOL 还有另一个特性，它关系到编译程序的编写和我们正在考虑的应用。这就是语言中的任选，它使得有可能一次就访问许多项目。一个 COBOL 程序员可以写

MOVE CORRESPONDING α TO β

它把具有对应名称的所有项目，从数据区域α移动到数据区域β。例如，COBOL 语句

MOVE CORRESPONDING DATE OF SALES TO DATE OF PURCHASES

意味着，SALES 文件中的 MONTH，DAY，和 YEAR 的值，要被移给 PURCHASES 文件中的变量 DAY，MONTH，YEAR（DAY 与 MONTH 的相对次序由此而被互换）。

这一节我们将要研究的问题，是设计三个适合于 COBOL 编译程序使用的算法，它们要做下列的事情：

操作 1．要加工如（1）那样的名称和层号的描述，把有关的信息放进编译程序内的表中，以供操作 2 和 3 使用。

操作 2．要确定一个已给的，如同规则（b）中那样的，限定的访问是否正确；而且当其正确时要确定对应数据项目的位置。

操作 3．要求出由一个“CORRESPONDING”语句所指出的所有对应的项目对偶。

我们假定，在我们的编译程序内存在着一个“符号表子程序”．它将把一个字母名称转换成一个指针，指向一个内存单元，其中包含着一个关于该名称的表条款（关于构造符号表算法的一些方法，在第 6 章中有详细的讨论）除了符号表之外，还有一张更大的表，它对于正在被编译的 COBOL 源程序中每个数据项目，包含着一个条款；我们将称这张表为数据表。

显然，在我们知道什么类型的信息有待存入该数据表之前，我们是不能为操作 1 设计算法的，而且该数据表的形式，取决于我们对何种信息需要施行操作 2 和 3。我们首先看操作 2 和 3。

为了确定 COBOL 访问

$$A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_n, \quad n \geqslant 0 \tag{3}$$

的意义，我们将首先在符号表中考察名称 A_0。应当有一系列的从符号表的条款到所有的关于这个名称的数据表条款的链接。然后对于每个数据表条款，我们将需要有一个链到包含这个条款的组项目的条款的链接。现在，如果有一个进一步的链接场，从数据表项目返回到符号表，那么，就不难看出象（3）这样的一个访问可以如何来处理。而且，为了定出由“MOVE CORRESPONDING”所指出的项目对偶，我们将需要有从组项目的数据表条款链到该组中的项目的某种类型的链接。

至此我们已经发现，在每个数据表条款中可能需要五个链接场：

PREV（链到以前具有同一名称的条款的链接，如果有这样的条款的话）；

FATHER（链到包含该项目的最小的组的链接，如果有这样的组的话）；

NAME（链到这个项目的符号表条款的链接）；

SON（链到一个组的头一个子项目的链接）；

BROTHER（链到包含这个项目的组中下一个子项目的链接）。

显然，象上边的 SALES 和 PURCHASES 那样的 COBOL 数据结构，实质上是一种树；而且，这里所出现的 FATHER，SON，和 BROTHER 链接，与我们以前的研究相类似（一树的通常的二叉树表示，由 SON 和 BROTHER 链接组成；加上 FATHER 链接就得到我们所谓的“三重链接树”。上边的五种链接，是由这些树链接，连同 PREV 和 NAME 一

起——对树结构赋以进一步的信息——而组成的)。

也许并不是所有这五种链接全都真是必要的，或者是充分的；但是，我们将首先在这样的试验性的假定之下，来试探设计我们的算法——即假定数据表条款将包含这五个链接场（加上与我们的问题不相干的进一步的信息)。作为所用的多重链接的一个例子，考虑两个 COBOL 数据结构

```
1 A                1 H
  3 B                5 F
    7 C                8 G
    7 D              5 B
  3 E                5 C                (4)
  3 F                  9 E
    4 G                9 D
                       9 G
```

它们将被表示成如（5）中所示的那样（具有用符号来表示的链接)。注意，符号表条款的 LINK 场，指向最近遇到的该所述符号名称的数据表条款。

我们要求的头一个算法，是构造这样一种形式的数据表的算法。注意由 COBOL 规则所允许的在层号选择上的灵活性，(4）中左边的结构完全等价于

```
1 A
  2 B
    3 C
    3 D
  2 E
  2 F
    3 G
```

因为层号不必是顺序的。

符号表

	LINK
A:	A1
B:	B5
C:	C5
D:	D9
E:	E9
F:	F5
G:	G9
H:	H1

（阴影表示在这里是无关的额外的信息）

数据表

	PREV	FATHER	NAME	SON	BROTHER
A1:	A	A	A	B3	H1
B3:	A	A1	B	C7	E3
C7:	A	B3	C	A	D7
D7:	A	B3	D	A	A
E3:	A	A1	E	A	F3
F3:	A	A1	F	G4	A
G4:	A	F3	G	A	A
H1:	A	A	H	F5	A
F5:	F3	H1	F	G8	B5
G8:	G4	F5	G	A	A
B5:	B3	H1	B	A	C5
C5:	C7	H1	C	E9	A
E9:	E3	C5	E	A	D9
D9:	D7	C5	D	A	G9
G9:	G8	C5	G	A	A

(5)

然而，可能有不予使用的层号序列；例如，如果（4）中D的层号改变成“6”(在任何一个位置)，则我们就得到一个没有意义的数据配置，它违背了关于一个组的所有项目必须有相同的层号，以及这个层号必须大于组名称的层号规则。因此，下边的算法要确保满足这样一些限制。

算法A（建造数据表） 这个算法，给出一个对应于上面(4）那样的COBOL数据结构的对偶（L、P）的序列，其中L是一个正整数“层号”，而P指向一个符号表条款。这个算法建造一个如上边的例（5）那样的数据表。当P指向一个以前尚未出现过的符号表的条款时，LINK（P）将等于A。这个算法使用一个辅助的堆栈，如同通常那样处理(或者如同2.2.2小节那样使用顺序的内存单元，或者如同2.2.3小节那样使用链接的分配)。

A1.〔初始化〕 把堆栈内容置为单个的条款(0，Λ)(整个算法之中，堆栈条款始终是对偶（L，P），其中L是一个整数，而P是一个指针；随着这个算法的进行，堆栈包含着层号和指向树中所有高于当前层的诸层上最后的数据条款。例如，在上边的例子中，在刚刚遇到对偶“3F”之前，堆栈由底到顶将包含着

$$(0,\Lambda)\quad(1,A1)\quad(3,E3))$$

A2.〔下一个项目〕 设(L，P)是来自输入的下一个数据项目。然而，如果输入已经穷尽，则这个算法就告终止。置Q⇐AVAIL（即，设Q是一个新的，我们能把下一个数据表条款放进去的节点的地址)。

A3.〔置名称链接〕 置

PREV（Q)←LINK（P)，LINK（P)←Q，NAME（Q)←P

(这就在NODE(Q）中妥善地为五个链接中的两个置位了。我们现在则要适当地来置FATHER，SON，以及BROTHER)。

A4.〔比较层号〕 命堆栈顶上的条款是(L1，P1)。如果L1＜L，则置SON（P1)←Q(或者，如果P1＝Λ，则置FIRST←Q，其中FIRST是指向头一个数据表条款的变量)并转到A6。

A5.〔撤消顶层〕 如果L1＞L，则撤消顶上的堆栈条款，命(L1，P1)是新的刚刚进到堆栈顶上的条款。重复步骤A5。如果L1＜L，则发出一个出错信号（在同一层上出现了混合的号数)。否则，当L1＝L时，则置BROTHER（P1)←Q，撤消这个顶上的堆栈条款，并命（L1，P1）是刚刚进到堆栈顶上的对偶。

A6.〔置族链接〕 置

FATHER（Q)←P1，SON（Q)←Λ，BROTHER（Q)←Λ

A7.〔加到堆栈〕 把（L，Q）放到堆栈顶上，并返回步骤A2。 ∎

通过引进一辅助堆栈，如同在步骤A1中所说明的，使得这个算法如此明晰，以致不需要进一步说明了。

下一个问题是，要定位对应于访问

$$A_0\ \text{OF}\ A_1\ \text{OF}\ \cdots\ \text{OF}\ A_n,\quad n\geqslant 0 \qquad (6)$$

的数据表条款。一个“好的”编译程序还将进行校验以确保这样一个访问是无二义性的。在这种情况下，一个适当的算法骤然呈现于眼前：我们所要做的一切，就是要遍查关于名称A_0的数据表条款，而且要确保这些条款中恰恰有一个，与所述的限定A_1，…，A_n相

匹配。

算法 B（校验一个限定的访问） 对应于访问（6），符号表子程序将求出指针$P_0, P_1, \cdots, P_n$，分别地指向关于$A_0, A_1, \cdots, A_n$的符号表条款。

这个算法的目的，是要检查$P_0, P_1, \cdots, P_n$；而且，或者确定出访问（6）有错，或者置变量Q成为被（6）访问的项目的数据表条款的地址。

B1.〔初始化〕 置$Q \leftarrow \Lambda$，$P \leftarrow \mathrm{LINK}(P_0)$。

B2.〔完成了？〕 如果$P = \Lambda$，则算法终止；这时如果(6)不对应于任何数据表条款，则Q将等于Λ。否则置$S \leftarrow P$和$k \leftarrow 0$（S是一个指针变量，它将凭借P沿着FATHER链接在树中向上运行；k是一个整数变量，它从0变到n。实际上，指针$P_0, \cdots, P_n$通常将被保存在一个链接表中，而且代替k，我们将代入一个遍历这个链接表的指针变量；见习题5）。

B3.〔匹配完成？〕 如果$k < n$，则转到B4。否则我们已经找到一个匹配的数据表条款；如果$Q \neq \Lambda$，则这是找到的第二个条款，所以发出一个出错状态信号。置$Q \leftarrow P$，$P \leftarrow \mathrm{PREV}(P)$，并转到B2。

B4.〔k增值〕 置$k \leftarrow k + 1$。

B5.〔上树〕 置$S \leftarrow \mathrm{FATHER}(S)$。如果$S = \Lambda$，则我们已经不能找到一个匹配；置$P \leftarrow \mathrm{PREV}(P)$并转到B2。

B6.〔A_k匹配？〕 如果$\mathrm{NAME}(S) = P_k$，则转到B3，否则转到B5。

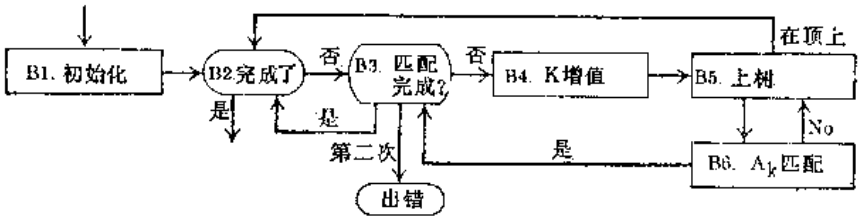


图40 校验一个COBOL访问的算法

注意这一算法并不需要SON和BROTHER链接。

我们所需要的第三个，也是最后的一个算法，是关于"MOVE CORRESPONDING"；而且在我们设计这样一个算法之前，我们对于所需要的，必须有一个精确的定义。COBOL语句

$$\text{MOVE CORRESPONDING } \alpha \text{ TO } \beta \tag{7}$$

其中α和β是（6）中那种对数据项目的访问，这是对于所有语句

$$\text{MOVE } \alpha' \text{ TO } \beta'$$

的集合的一个缩写，其中存在一个整数$n \geqslant 0$和n个名称$A_0, A_1, \cdots, A_{n-1}$，使得

$$\begin{aligned} \alpha' &= A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_{n-1} \text{ OF } \alpha \\ \beta' &= A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_{n-1} \text{ OF } \beta \end{aligned} \tag{8}$$

而且α'或者β'是一个初等项目（不是一个组项目）。我们还要求（8）示出完备的限定，即是，A_{j+1}是A_j的父亲，对于$0 \leqslant j \leqslant n-1$；$\alpha'$和$\beta'$必须在树中恰恰比$\alpha$和$\beta$低$n$层。

在我们的例子（4）中，

"MOVE CORRESPONDING A TO H"

是关于语句

MOVE B OF A TO B OF H
MOVE G OF F OF A TO G OF F OF H

的一个缩写。

这个要识别所有对应的对偶 α'，β' 的算法，尽管并不困难，但是十分有趣；我们在先根次序下，跑遍其根为 α 的树，同时在 β 树中寻找匹配的名称，而且跳过其中不可能出现对应元素的子树。(8) 的名称 A_0，…，A_{n-1} 遂以相反的次序 A_{n-1}，…，A_0 而被发现。

算法 C（找 CORRESPONDING 对偶） 给定了 PO 和 QO，它们分别指向对于 α 和 β 的数据表条款，这个算法逐步地找出所有的指向满足上述限制条件的项 (α', β') 的指针对偶 (P, Q)。

C1.〔初始化〕 置 P←PO，Q←QO（在这个算法的余下部分中，指针变量 P 和 Q 将走遍分别有根 α 和 β 的树）。

C2.〔初等的？〕 如果 SON(P)=A 或 SON(Q)=A，则输出(P, Q)以作为一个所求的对偶并转到 C5。否则置 P←SON(P)，Q←SON(Q)（在这一步骤，P 和 Q 指向满足 (8) 的项目 α' 和 β'，而且我们希望要 MOVE α' TO β' 当且仅当 α' 或 β'（或两者）是初等项目）。

C3.〔匹配名称〕（现在 P 和 Q 指向分别有形如

$$A_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_{n-1} \text{ OF } \alpha$$

和

$$B_0 \text{ OF } A_1 \text{ OF } \cdots \text{ OF } A_{n-1} \text{ OF } \beta$$

之限定的数据项目。目的是要通过检查这组 A_1 OF … OF A_{n-1} OF β 中的所有名称，来看看我们是否能使 $B_0=A_0$)。如果 NAME(P)=NAME(Q)，则转到 C2(已经找到了一个匹配)。否则，如果 BROTHER(Q)≠A，则置 Q←BROTHER(Q) 并重复步骤 C3（如果 BROTHER(Q)=A，则在这组中不出现匹配的名称，因而我们继续进行步骤 C4)。

C4.〔继续前进〕 如果 BROTHER(P)≠A，则置

P←BROTHER(P) 和 Q←SON(FATHER(Q))

并转回到 C3。如果 BROTHER(P)=A，则置

P←FATHER(P) 和 Q←FATHER(Q)

C5.〔完成了？〕 如果 P=PO，则算法终止；否则转到 C4。 ∎

在图 41 中，示出了这个算法的框图。关于这个算法的正确性的证明，通过对所涉及树的大小运用归纳法，即可容易地构造出来（见习题 9）。

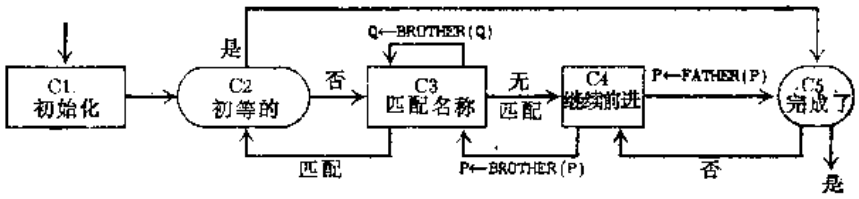


图41 "MOVE CORRESPONDING" 算法

在此，值得研究一下，算法B和C使用这五个链接场PREV，FATHER，NAME，SON，和BROTHER的方式。令人惊奇的特性是，在这样的意义下，这五个链接组成一个“完备的集合”，即当算法B和C遍历数据表而移动时，它们实际上只做极少量的工作；每当有必要访问另一个数据表条款时，它的地址就在我们的手指尖之下立即可加以取用；我们无需去寻找它。如果还有任何附加的链接信息可出现在这个表中，我们也很难想象，如何能够把算法B和C搞得更快些（见习题11）。

每个链接场，都可以看作是在那里设置的为使算法运行得更快的一条线索（当然，尽管建造这个表的算法，象算法A那样，因为它们要添入更多的链接，相应地运行较慢）。显然，上面建造的数据表包含着许多冗余信息。现在我们就来考虑，如果删去某些链接场，将会发生什么情况。

PREV链接，虽在算法C中不使用，但对算法B却极为重要，而且，如果不是漫长的检索有待完成，则它似乎是任何COBOL编译程序的一个实质性部分。因而，一个把所有同名的项目链在一起的场，似乎对有效性是很必须的。我们也许可以稍微修改一下这个技巧并采用循环链接而不是以Λ终止每一个链接表，但却没有什么理由来删去它，除非其它的链接场已被改变或被撤消。

FATHER链接，在算法B和C中都使用了，尽管在算法C中可予避免，如果我们在该算法中使用一个辅助堆栈，或者如果我们扩充BROTHER使之包括“穿线的”链接（参照2.3.2小节）。所以我们看出，FATHER链接仅仅在算法B中才以必要的方式被使用。如果BROTHER链接穿成线了，使得现有的BROTHER＝Λ项目将代之于BROTHER＝FATHER，则将有可能沿着这个BROTHER链接来对任何数据项目的父亲进行定址。所增加的链接场可以这样地区别出来，即或者是通过在每一个节点中有一个新的TAG场来说明一BROTHER链接是否为一穿线来区别，或者通过条件“BROTHER(P)＜P”来区别，如果诸数据表条款以出现的次序连续地保存在内存中的话。这将意味着在步骤B5中，必须进行简短的检索，因此该算法相应地就得慢些。

NAME链接仅仅在步骤B6和C3中，才为算法所使用。在两种情况下，我们均可以其它的方式来测试“NAME(S)＝P_k”，“NAME(P)＝NAME(Q)”，如果不出现NAME链接的话（参照习题10），这将显著地放慢算法B和算法C两者的内循环。在这里我们再一次看到，供链接用的空间与算法的速度之间的一个折衷（在COBOL编译程序中，当我们考虑MOVE CORRESPONDING的典型用法时，算法C的速度不是特别有意义的；但算法B应该是快的）。经验说明，对于在一个COBOL编译程序内的NAME链接，还发现有其它重要的使用，特别是在打印诊断信息时。

由于算法A是逐步地建造数据表的，而且决没有机会把它恢复到可利用的存储池中，我们通常发现，数据表条款是以数据项目在COBOL源程序中出现的顺序来占用连续的内存单元的。于是，在我们的例子（5）中，单元A1，B3，…将彼此挨着。数据表的这种顺序的本性导致了若干的简化；例如，每个节点的SON链接或者是Λ，或者指向直接紧接着的节点，所以SON可以缩减成一个1位的场。或者，SON还直被撤消而利于测试是否FATHER(P＋C)＝P，其中C是数据表中的节点大小。

这样，五个链接场并不全都是必要的，尽管从算法B和C的速度来看它们是有帮助的。

这种情况在大多数多重链接中，是相当典型的。

指出这样一点是有趣的，即至少有半打的编写 COBOL 编译程序的人们已经独立地得到这种使用五个链接（或五个中的四个，通常 SON 链接不出现）来维持一个数据表的同一种方式。这样一种技术是由小哈·威·劳森（H. W. Lawson, Jr.）头一次发表的（《ACM 国内会议文摘》(ACM National Conference Digest)，锡拉丘兹，纽约，1962)。但在1965年，戴维·达姆（David Dahm）介绍了仅仅使用两个链接场和使用数据表的顺序存储，又不大降速度，是达到算法 B 和 C 之效能的一个巧妙的技术；见习题 12 到 14。

习题

1.〔00〕 如果把 COBOL 的数据结构当作树形结构来考虑，那么COBOL程序员所列出的数据项目，是在先根次序之下、后根次序之下，还是这两者都不是呢？

2.〔10〕 试对算法 A 的运行时间作出说明。

3.〔22〕 PL/I 语言所接受的数据结构非常象 COBOL 中的数据结构，只是层号序列可以是任意的。例如，序列

1 A		1 A
3 B		2 B
5 C	等价于	3 C
4 D		3 D
2 E		2 E

一般说来，规则（a）被修改成“一个组中的项目必须有一个非递增的层号序列，其中所有的层号均大于组名称的层号”。当从 COBOL 的约定改变成 PL/I 的这个约定时，算法 A 将应作那些修改？

▶4.〔26〕 如果 COBOL 程序员违犯正文中所述的规则（C），则算法 A 并不检测这个错误。应该如何修改算法 A，使得它仅仅接受满足规则（C）的数据结构？

5.〔20〕 实际上，算法 B 可以给出符号表访问的一个链接表来作为输入，以代替我们所说的“P_0，P_1，…，P_m”命 T 是一个指针变量，使得

$$\mathrm{INFO}(T)\equiv P_0,\ \mathrm{INFO}(\mathrm{RLINK}(T))\equiv P_1,\ \cdots,$$

$$\mathrm{INFO}(\mathrm{RLINK}^m(T))\equiv P_m,\ \mathrm{RLINK}^{m+1}(T)=\Lambda$$

试说明怎样来修改算法 B，使得它可使用这样一个链接表来作输入。

6.〔23〕 PL/I 语言接受的是与 COBOL 非常相象的数据结构，但是未加规则（C）的限制；代替它的是这样一个规则：一个限定的访问（3）是无二义性的，如果它示出“完备的”限定，即如果 A_{j+1} 是 A_j 的父亲，对于 $0\leqslant j<n$，而且如果 A_n 没有父亲。规则（C）现在减弱成为一个这样的简单条件：没有一个组的两个项目具有相同的名称。（2）中的第二个“CC”将无二义地作为“CC OF AA”而被访问之；三个数据项目

1 A

2 A

3 A

就刚才所述的 PL/I 的约定而论，将作为“A”，“A OF A”，“A OF A OF A”访问之（注意：实际上，词“OF”在 PL/I 中以一个句号来代替，而且次序也被颠倒过来；“CC OF AA”

在PL/I中实际上写成"AA·CC",但这一点对于本习题的目的说来是不重要的)试说明:怎样来修改算法B使得它遵从PL/I的约定,即它将不把一个完备的限定作为二义性来考虑。

7.〔15〕 给定了(1)中的数据结构,则COBOL的语句"MOVE CORRESPONDING SALES TO PURCHASES"意味着什么?

8.〔10〕 在什么情况下,按照正文中的定义,

"MOVE CORRESPONDING α TO β"

与

"MOVE α TO β"

恰恰完全一样?

9.〔*M*23〕 证明算法C是正确的。

10.〔23〕 (a)如果在数据表的节点中没有NAME链接,则在步骤B6中怎样才能实现测试"NAME(S)=P_k"?(b)如果在数据表条款中没有NAME链接,则在步骤C3中如何能实现测试"NAME(P)=NAME(Q)"?(假定所有其它的链接均如正文中那样表示)。

▶11〔23〕 在正文的这些算法技巧中,附加些什么链接或者附加些什么改动,便能使算法B或算法C更快些?

12.〔25〕 (戴·迈·达姆)试考虑在顺序单元中来表示数据表,其中每个项目恰好有两个链接:

PREV (如正文中那样);

SCOPE (链接该组最后的初等项目的链接)

我们有SCOPE(P)=P当且仅当NODE(P)表示一个初等项目。例如,(5)的数据表将代之以

	PREV	SCOPE		PREV	SCOPE
A1:	Λ	G4	H1:	A	G9
B3:	Λ	D7	F5:	F3	G8
C7:	Λ	C7	G8:	G4	G8
D7:	Λ	D7	B5:	B3	B5
E3:	Λ	E3	C5:	C7	G9
F3:	Λ	G4	E9:	E3	E9
G4:	Λ	G4	D9:	D7	D9
			G9:	G8	G9

(与2.3.3小节的(5)作比较)注意:NODE(P)是在NODE(Q)之下的树的一部分,当且仅当,Q-P≤SCOPE(Q)。试设计一个算法,当数据表有着这种格式时,它实现了算法B的功能。

▶13.〔24〕 试给出一个算法,当数据表有着习题12中所示的格式时,它代替了算法A。

▶14.〔28〕 试给出一个算法,当数据表有着习题12中所示的格式时,它代替了算法C。

2.5 动态存储分配

我们已经看到,链接的使用是怎样蕴涵着表格不必顺序地放置在内存中的;许多表格

可以独立地在一个公共的"联营成存储池"的内存区域中，增长和收缩。然而，我们的讨论总是暗中假定，所有的节点都有同样的大小，即它们都占有同样数量的内存单元。

对于许许多多的应用，可以求得一个适当的妥协，使得对于所有的表格都使用统一的节点大小（例如，见习题2）。习惯上，不是简单地取所必需的最大的大小(因为对于较小的节点就会浪费空间)，而是来选择比较小的节点大小，并且来应用这样的原理——可称之为经典的内存链接原理："如果这里已没有存储该信息的余地，则我们就把它放到某个别处并且安置一个指向它的链接。"

然而，对于其它的许许多多应用，单一的节点大小是不合理的；我们往往希望能有可变大小的节点，来共享一个公共的内存区域。换句话说就是，我们需要这样的算法，它们要从一个更大的内存区域来保留和释放可变大小的内存块，其中这些块是由连续的内存单元组成的。这样的技术一般就称作"动态的存储分配"算法。

有时候，通常是在模拟程序中，我们要求对于大小比较小的节点（比如说一个到十个字）进行动态的存储分配；而在其它时候，通常是在"执行着的"控制程序中，我们主要是处理比较大块的信息。这两种观点就导致了对于动态存储分配的稍微不同的解决办法，尽管这些方法有许多是相通的。为了在这两种方法之间求得术语上的统一，在这一节里我们将一般地使用术语块和区域，而不使用"节点"，用来表示相连接的内存单元的集合。

A. 保留 图42示出了一个典型的"内存图"或"跳棋盘"，即是一份表示某个内存池当前状态的图表。在这种情况下，所示的内存被划分成53块"被保留的"，即在使用中的存储块，与21块不在使用中的"自由的"或"可利用的"块混合在一起。在动态存储分配业

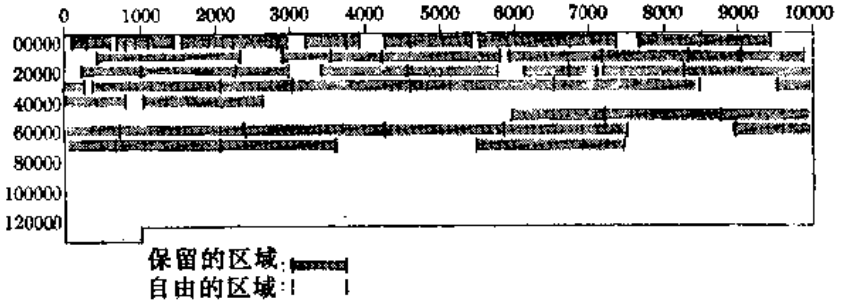


图42 内存图

经操作了一会儿之后，计算机内存看起来颇象这个样子；我们的头一个问题是

a）在计算机内，如何划分这种可利用的空间？

b）给定了这样一种可用空间的表示，为寻找一块 n 个连续的自由空间并保留之，怎样才是一个好的算法？

当然，对于问题（a）的答案是在某个地方保持一个可利用空间的表；而往往总是最好使用可利用空间本身来包含这样一个表（一个例外是当我们要为一个磁盘文件或者其它存储器分配存储时的情况，其中不一致的存取时间使得，还是来维护一个独立的可利用空间的目录倒更好些）。

这样，我们可以把可利用的段链接在一起；每一个自由存储区域的第一个字可以包含该块的大小以及下一个自由区域的地址。自由块可以大小递增或递减的次序链接在一起，

或者以内存地址的次序，或者以实际上随机的次序。

例如，考虑图 42，它示出一131，072 个字的大内存，其地址从 0 到 131071。如果我们以内存单元为次序把可利用的块链在一起，那么，我们就将有一个指向头一块自由块的变量 AVAIL（在这种情况下，AVAIL 将等于 0），而其它诸块将表示成如下：

单元	大小	链接	
0	101	632	
632	42	1488	
⋮	⋮	⋮	〔17 个类似的条款〕
73654	1909	77519	
77519	53553	Λ	〔最后链接的特殊标志〕

于是单元 0 到 100 形成头一个可利用的块；在图 42 中所示的保留区域 101-290 和 291-631 之后，我们还有一个自由空间在单元 632-673；等等。

至于问题（b），如果我们要求 n 个连续的字，则显然我们必须找出某个有 $m \geqslant n$ 个可利用字的块，并把它的大小缩减成 $m-n$（而且，当 $m=n$ 时，我们还必须从可利用空间的表中删去这个块）。可能有许多个具有 n 个或更多个单元的块，因而问题就变成为：应当选择哪一个区域呢？

对于这个问题，有两个原则的答案呈现在眼前：我们可以使用“最佳适应”方法或“首先适应”的方法。在前一种情况下，我们决定选择一个具有 m 个单元的区域，其中 m 是所出现的最小为 n 或更大的值。在能够作出一个决定之前，可能要检索整个可利用空间表。另一方面，“首先适应”方法，则简单地选择所遇到的有着 $\geqslant n$ 个字的头一个区域。

历史上，最佳适应方法已广泛地使用了多年；自然是一个好的决策，因为它节省下更大的可利用的区域供以后可能需要它们的时候使用。但是，最佳适应这一技术，也可能引起若干缺陷：它比较缓慢，因为它包含有相当长的检索；如果“最佳适应”不是由于其它的原因，比“首先适应”实质上更好些的话，那么，这个额外的检索时间是不值得的。更重要的是，最佳适应方法，增加了许多非常小的块，而小块的蔓延通常是不希望的。有些情况，首先适应技术确实比最佳适应方法更好；例如，假设仅仅给了我们两个大小为1300和1200的可利用的内存区域，并假设此后逐次地需要大小为 1000，1100、和 250 的块，那么：

内存需求	可利用的区域“首先适应”	可利用的区域“最佳适应”	
—	1300，1200	1300，1200	
1000	300，1200	1300，200	(1)
1100	300，100	200，200	
250	50，100	为难	

由于这些原因，首先适应方法还是可取的。

算法 A（首先适应方法） 命 AVAIL 指向头一个可利用的存储块，并假设具有地址 P 的每个可利用的存储块有着两个场：SIZE(P)，块中的字数；以及LINK(P)，指向下一个可利用块的指针。最后的指针是Λ。这个算法检索和保留N个字的一个块，或报告失败。

A1. 〔初始化〕 置Q←LOC(AVAIL)（贯穿这个算法，我们始终使用两个指针Q和P，

一般地这两者通过条件P＝LINK(Q) 相互关联。我们假定

$$LINK(LOC(AVAIL))=AVAIL)$$

A2.〔表之末尾？〕 置P←LINK(Q)。如果P＝A，则算法不成功地结束；没有余地供给一个有着N个连续字块。

A3.〔SIZE 够吗？〕 如果SIZE(P)≥N，则转到A4；否则置Q←P并返回步骤A2。

A4.〔保留N〕 置K←SIZE(P)－N。如果K＝0，则置LINK(Q)←LINK(P)(从该表撤消一个空闲的区域)；否则置SIZE(P)←K。这个算法成功地结束，从而保留了一个从单元P＋K开始的长度为N的区域。 ∎

这个算法的确够直接了当的了。然而，在技巧上只不过是做微小的变化，就可使它在运行速度方面得到显著的改进。这一改进是十分重要的，读者将会有这样的体验，亲自发现这个改进将是一大快事（见习题6）。

存储分配不论是要求小的N还是大的N，算法A都可加以使用。然而，让我们暂时假定，我们主要是对大的N值感兴趣。然后我们来说明在这个算法中，当SIZE(P)等于N＋1时，将发生什么情况：我们到达步骤A4。并把SIZE(P)减小成1。换句话说，一块大小为1的可利用的块刚才已被建立；这个块是如此之小，以致它实际上是无用的，而且它恰恰阻碍着这个系统。如果我们迳而保留这整个的N＋1个字的块，而不是省出这个多余的字，则我们反倒能更好地摆脱；往往是花费少量的内存字反倒更好，以避免处理某些烦琐的细节。当K很小时，类似的陈述也可应用到N－K个字的块上。

如果我们允许有保留稍微多于N个字的可能性，则就有必要记住已经保留了多少个字，以便稍后当这个块再次变成可利用时，就释放这整组的N＋K个字。这增加的簿记工作的要旨，意味着仅在一个“贴切的适应”已被找到的某些情况下，我们才在每一块中扩大空间，为的是使这系统更为有效；所以这个技巧看来并不特别地吸引人。然而，由于其它原因，往往发现需要一个特殊的控制字，作为每个可变大小块的头一个字。因此通常预期SIZE场出现在每块的头一个字中——不论这个块可利用与否——并不是不合理的。

根据这些约定，我们将把上面的步骤A4修改如下：

“A4′.〔保留≥N〕 置K←SIZE(P)－N。如果K＜c（其中c是一个小的正常数，被选择来反映我们出于想要节省时间而欣然愿意牺牲的存储数量），则置

$$LINK(Q)\leftarrow LINK(P)\text{ 和 }L\leftarrow P$$

否则置

$$SIZE(P)\leftarrow K,\ L\leftarrow P+K,\text{ 及 }SIZE(L)\leftarrow N$$

这个算法成功地结束，业已保留了一个从单元L开始的长度为N或更大的区域。”

建议大约用8或10来作为常数c的值，尽管很少有理论或实践的证据存在，来把这个选择与其它选择加以比较。当正在使用的是最佳适应方法时，K＜c的测试，甚至比起它在首先适应方法中所做的更为重要，因为更贴切的适应者（更小的K值）更可能出现，而且可利用块的块数对于该算法应该保持尽可能地小。

B. 释放 现在让我们来考虑相反的问题：当诸块已不再需要时，我们应该怎样把它们恢复到可利用空间的表中去？

通过使用“废料收集”（见2.3.5小节），也许会诱使我们草草了结这个问题；我们可

以遵循这样一个简单的决策，即直到空间用完为止什么也不干，然后检查所有当前正使用的区域，并且形成一个新的 AVAIL 表。

但是，我们并不希望对所有的应用都来推荐废料收集的思想。首先，如果我们要能保证当前所有正在使用的区域都很易于找出，则我们就需要相当“有纪律地”使用指针，而这种适度的纪律在这里所考虑的应用中通常是缺乏的。其次，如同我们在这以前所已见到的，当内存已接近充满时，废料收集即趋于变慢。

为什么废料收集不是令人满意的，还有另一个更为重要的原因，即有一个我们以前对这个技术的讨论中尚未遇到的现象：假设有两个相邻的内存区域，这两者都是可利用的，但由于废料收集的原理，它们当中的一个（以阴影示出的）不在 AVAIL 表中。



（2）

在这个图式中，左边和右边两端中的浓阴影区域是不可利用的。我们现在可以保留已知是可利用的区域中的一段：



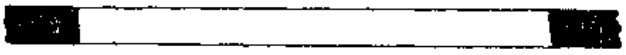
（3）

如果这时出现废料收集，则我们就有两个分开的自由区域.



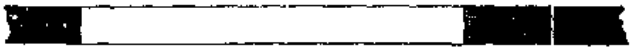
（4）

在可利用的与被保留的区域之间的界限，竟有一种保全下去的趋向，而且随着时间的延续，这一情况还逐渐地变坏。但是如果我们使用这样一种原理，即一旦诸块变成自由的，就把它们恢复到 AVAIL 表，而且把两个相邻的可利用区域融合在一起，那么，我们就把（2）融合成



（5）

而且我们就会得到



（6）

这比（4）要好得多！这种现象导致废料收集技术使内存比应有的更为破碎。

为了克服这一困难，可把废料收集与紧凑内存的过程结合起来，即每当废料收集完成时，就把所有的保留块移到连续的单元，从而所有的可利用块也就都会合在一起了。对照算法 A，则分配算法现在就变得十分平凡了，因为在所有时刻仅仅有一个可利用的块。即使这种技术要花时间来重新复写所有正在使用的单元，而且还要改变其中的链接场之值，但只要对指针有一个“纪律严明”的用法，以及只要在每一块中有一个链接场腾出供废料

收集算法使用，仍然可以相当高的效率来应用之（见习题33）。

由于许多应用并不满足这些为使废料收集可实行的要求，所以我们现在要研究把内存块恢复到可利用空间表的方法。在这些方法中，唯一的困难是融合问题：两个相邻的自由区域应当合并成一个。事实上，当以两个可利用的块作边界的一个区域变成自由时，所有这三个区域就应当合并成一个。依照这种方式，内存中就得到一个很好的平衡，即使存储区继续被保留和被释放一个长时间（关于这一事实的证明，见下面的“百分之五十”的规则）。

问题是要确定，在被恢复块的两边的区域，当前是否可利用；如果它们是可利用的，则我们就要适当地修改AVAIL表。这后一操作比起设想的要稍微困难些。

对这个问题的第一个解决办法，是以内存单元的递增顺序来维持AVAIL表。

算法B（通过已分类的表释放） 在算法A的假定下，并补充假定：AVAIL表已按内存单元分类（即如果P指向一个可利用的块，而且LINK(P)≠A，则LINK(P)>P），这个算法把由单元PO开始的N个连续单元的块，加到AVAIL表中。我们自然假定，这些N个单元中已经没有可利用的。

B1.〔初始化〕 置Q←LOC(AVAIL)（见上面步骤A1中的注记）。

B2.〔推进P〕 置P←LINK(Q)。如果P=A，或者如果P>PO，则转到B3；否则置Q←P并重复步骤B2。

B3.〔校验上界〕 如果PO+N=P（且P≠A），则置N←N+SIZE(P)并置LINK(PO)←LINK(P)。否则置LINK(PO)←P。

B4.〔校验下界〕 如果Q+SIZE(Q)=PO〔我们假定

$$SIZE(LOC(AVAIL))=0,$$

所以当Q=LOC(AVAIL)时这个测试总是失败〕，则置SIZE(Q)←SIZE(Q)+N以及LINK(Q)←LINK(PO)。否则置LINK(Q)←PO，SIZE(PO)←N。 ▮

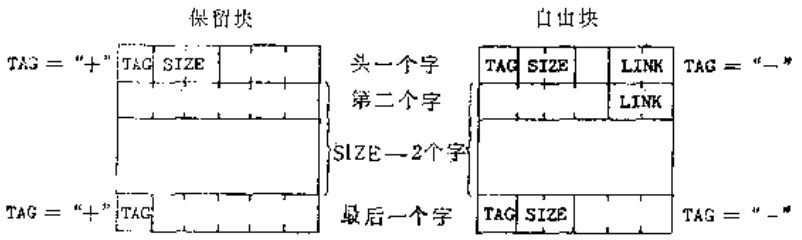
步骤B3和B4以下述事实为基础，进行了所要求的融合：Q<PO<P是三个连续的可利用区域的开始单元。

如果AVAIL表不是按单元的顺序来维持，则读者可以看到，对融合问题的“蛮干”解法将要求完全地检索整个AVAIL表；算法B把这个检索缩减到平均约为AVAIL表的一半（在步骤B2）。习题11说明，可以如何来修改算法B，使得平均仅需要检索AVAIL表的约三分之一。但显然，当AVAIL表很长时，所有这些方法，比起我们所要求于它们的，都嫌太慢了。难道就没有某种保留和释放存储的方法，使得我们不必对整个AVAIL表做这么多的检索吗？

我们现在就来考虑一种方法，当恢复存储时，它能消除所有的检索，而且当保留存储时，可以把它加以修改，如同在习题6中那样，几乎避免所有的检索。这种技术在每一个块的两端利用了一个TAG场，并在每块的头一个字中利用了一个SIZE场；当正在使用的块相当大时，这点“开销”是可以忽略不计的，但是当这些块的平均大小非常小的时候，也许就是一项太大的代价。习题19中描述的另一种方法，只要求用每块的第一个字中的一位，不过要牺牲点运行时间，而且程序也稍微复杂些。

总而言之，我们假定，当AVAIL表很长时，为了节省运行算法B的大量时间，我们

不在乎增加一点控制信息。我们所描述的方法将假定每块有如下的形式：



(7)

下列算法的思想，是要维持一个双重链接的 AVAIL 表，以便可以很方便地从该表的随机部分删去条款。在一个块两端的任何一端，TAG 场可用来控制融合的过程，因为我们能容易地判明两个相邻的块是否都是可利用的。

以熟悉的方式实现双重链接，即命头一个字中的 LINK 指向表中的下一个自由块，并且命第二个字中的 LINK 回过头来指向先前的块。于是，如果 P 是一个可利用块的地址，则我们总有

$$\mathrm{LINK}(\mathrm{LINK}(P)+1)=P=\mathrm{LINK}(\mathrm{LINK}(P+1)) \tag{8}$$

为了确保适当的“边界条件”，乃设置表头如下：



(9)

对于这种技术的一个“首先适应”的保留算法，可以象算法 A 那样进行设计，所以在这里我们就不予考虑了（参照习题 12）。这个方法的特点主要在于：在实质上固定的时间里，一个块可被释放的方式。

算法 C（使用边界标志的释放） 假定单元块有（7）中所示的形式，并且假定，如同上边所述，AVAIL 表是双重链接的。这个算法把从地址 PO 开始的单元块放入 AVAIL 表中。如果这个可利用的存储池从单元 m_0 遍及 m_1，包括 m_0 和 m_1 在内，则为了方便起见，这个算法假定

$$\mathrm{TAG}(m_0-1)=\mathrm{TAG}(m_1+1)=\text{“+”}$$

C1. 〔校验下界〕 如果 TAG(PO − 1) = “+”，则转到 C3

C2. 〔删去下区〕 置 P ← PO − SIZE(PO − 1)，然后置

P1 ← LINK(P)，P2 ← LINK(P + 1)，LINK(P1 + 1) ← P2

LINK(P2) ← P1，SIZE(P) ← SIZE(P) + SIZE(PO)，PO ← P

C3. 〔校验上界〕 置 P ← PO + SIZE(PO)。如果 TAG(P) = “+”，则转到 C5。

C4. 〔删去上区〕 置

P1 ← LINK(P)，P2 ← LINK(P + 1)，LINK(P1 + 1) ← P2

LINK(P2) ← P1，SIZE(PO) ← SIZE(PO) + SIZE(P)，P ← P + SIZE(P)

C5. 〔加到 AVAIL 表中〕 置

SIZE(P − 1) ← SIZE(PO)，LINK(PO) ← AVAIL

LINK(PO + 1) ← LOC(AVAIL)，LINK(AVAIL + 1) ← PO

AVAIL ← PO，TAG(PO) ← TAG(P − 1) ← “−” ■

算法C的这些步骤，乃是存储方案（7）的直接了当的推论；一个稍微长些的但却比较快些的算法，出现在习题15中。在步骤C5中，AVAIL是LINK(LOC(AVAIL))的一个缩写，如同（9）中所示。

C. "伙伴系统" 我们现在将研究适合于二进制计算机使用的，进行动态存储分配的另一种方法。这种方法在每块中都付出一位的"开销"，而且，要求所有块的长度将是1，2，4，8，或16等等；如果一个块的长度不是对应于某个整数k的2^k个字，则就选择下一个更高的2的次幂，并从而分配了额外的不用的空间。

这种方法的思想，就是保持每个大小为2^k的独立的可利用块的表，$0 \leq k \leq m$。参与分配的内存空间的整个池，由2^m个字组成；为方便起见，我们将假定它的地址是从0到2^m-1。原先，整个的2^m个字的块都是可利用的。过后，当要求一2^k个字的块，而且如果没有这样大小的块可利用时，就把更大的可利用块分成相等的两部分；最终，就出现正好大小为2^k的块。当一块分成两块时（其中的每一块都是原来块的一半），这两块即称为伙伴。过后，当这两个伙伴再次可利用时，它们又合并回来成为单一的块；于是，这过程遂可无限地维持下去，除非在某处我们用尽了空间。

为这个方法的实际可用性奠定基础的关键事实是，如果我们知道一个块的地址（即它的头一个字的内存单元），而且又知道该块的大小，我们就知道它的伙伴的地址。例如，开始于一个二进单元101110010110000的大小为16的块，其伙伴是开始于二进单元101110010100000的块。为了看出这为什么必然是对的，我们首先发现，随着这个算法的进行，大小为2^k的一个块的地址，是2^k的一个倍数。换句话说，在二进记号下这地址至少有k个0在右边。这项发现通过归纳法很容易证实；如果对于所有大小为2^{k+1}的块为真，则当这样一个块被二等分时这肯定地亦真。

因此，其大小比如说是32的一块，有一形如$xx\cdots x00000$的地址（其中x表示0或1）；如果把它破半，则新形成的伙伴块就有地址$xx\cdots x00000$和$xx\cdots x10000$。一般地，命$\text{buddy}_k(x)$=地址为x大小为2^k的块之伙伴的地址，则我们就有

$$\text{buddy}_k(x)=\begin{cases} x+2^k, & \text{若 } x \bmod 2^{k+1}=0 \\ x-2^k, & \text{若 } x \bmod 2^{k+1}=2^k \end{cases} \tag{10}$$

通过"异或"指令（有时候称为"选择求补"或"按位加"指令），很容易计算这个函数；而这种指令，在二进制计算机上通常均可找到；参照习题28。

伙伴系统在每块中利用一个一位的TAG场：

$$\begin{aligned} &\text{TAG(P)}=0，\text{如果具有地址P的块被保留} \\ &\text{TAG(P)}=1，\text{如果具有地址P的块可利用} \end{aligned} \tag{11}$$

除了在所有的块中都出现的TAG场之外，可利用的块还有两个链接场LINKF和LINKB，它们就是通常的双重链接表的向前和向后的链接；而且可利用的块还有一个KVAL场以指出k，当它们的大小为2^k时。以下的算法利用表格单元AVAIL[0]，AVAIL[1]，…，AVAIL[m]，它们分别作为大小为1，2，4 …，2^m的可利用存贮表的表头。这些表都是双重链接的，所以象通常那样，表头均含两个指针（见2.2.5小节）：

$$\begin{aligned}&\mathrm{AVAILF}[k]=\mathrm{LINKF}(\mathrm{LOC}(\mathrm{AVAIL}[k]))=\text{链到 AVAIL}[k]\text{表末尾的链接}\\&\mathrm{AVAILB}[k]=\mathrm{LINKB}(\mathrm{LOC}(\mathrm{AVAIL}[k]))=\text{链到 AVAIL}[k]\text{表头上的链接}\end{aligned}\tag{12}$$

开始。在分配任何存储之前，我们有

$$\mathrm{AVAILF}[m]=\mathrm{AVAILB}[m]=0$$
$$\mathrm{LINKF}(0)=\mathrm{LINKB}(0)=\mathrm{LOC}(\mathrm{AVAIL}[m])\tag{13}$$
$$\mathrm{TAG}(0)=1\qquad \mathrm{KVAL}(0)=m$$

（表示一个单一的开始于单元 0 的长度为 2^m 的可利用块），并且还有

$$\mathrm{AVAILF}[k]=\mathrm{AVAILB}[k]=\mathrm{LOC}(\mathrm{AVAIL}[k]),\ \text{对于}\ 0\leqslant k<m\tag{14}$$

（表示对于所有的 $k<m$，长度为 2^k 的可利用块的空表）。

从伙伴系统的这一描述，读者可以发现，由他自己亲自来设计用于保留和释放存储区域的必要算法，是颇有兴味的，而且还可以把他的解答与下面给出的算法加以比较。注意按此描述，在这个保留算法中，诸块可相当容易地分成两半。

算法 R（伙伴系统的保留） 这个算法，利用以上所说明的伙伴系统之组织，找出并保留一个 2^k 个单元的块，或报告失败。

R1.〔找块〕 命 j 是在 $k\leqslant j\leqslant m$ 的范围内最小的整数，对于它 $\mathrm{AVAILF}[j]\neq\mathrm{LOC}(\mathrm{AVAIL}[j])$，即对它来说，大小为 2^j 的可利用块的表不是空的。如果不存在这样的 j，则这个算法不成功地终止，因为没有足够大小的已知可利用块来满足需要。

R2.〔从表中撤消〕 置

$$L\leftarrow\mathrm{AVAILF}[j]\qquad \mathrm{AVAILF}[j]\leftarrow\mathrm{LINKF}(L)$$
$$\mathrm{LINKB}(\mathrm{LINKF}(L))\leftarrow\mathrm{LOC}(\mathrm{AVAIL}[j])\ \text{以及}\ \mathrm{TAG}(L)\leftarrow 0$$

R3.〔需要分开？〕 如果 $j=k$，则这算法终止（我们已经找到并保留了一个由地址 L 开始的可利用块）。

R4.〔分开〕 j 减 1。然后置

$$P\leftarrow L+2^j\quad \mathrm{TAG}(P)\leftarrow 1\quad \mathrm{KVAL}(P)\leftarrow j\quad \mathrm{LINKF}(P)\leftarrow\mathrm{LOC}(\mathrm{AVAIL}[j])$$
$$\mathrm{LINKB}(P)\leftarrow\mathrm{LOC}(\mathrm{AVAIL}[j]),\ \mathrm{AVAILF}[j]\leftarrow\mathrm{AVAILB}[j]\leftarrow P$$

（这就把一个大块分开并把不用的一半记入原来是空的 $\mathrm{AVAIL}[j]$ 表中。）返回步骤 R3。

算法 S（伙伴系统的释放） 利用以上阐明的伙伴系统的组织，这个算法把一开始于地址 L 的 2^k 个单元的块，恢复到自由存储中。

S1.〔伙伴可利用否？〕 置 $P\leftarrow \mathrm{buddy}_k(L)$。（见等式 (10)。）如果 $k=m$ 或如果 $\mathrm{TAG}(P)=0$，或者如果 $\mathrm{TAG}(P)=1$ 且 $\mathrm{KVAL}(P)\neq k$，则转到 S3。

S2.〔同伙伴结合〕 置

$$\mathrm{LINKF}(\mathrm{LINKB}(P))\leftarrow\mathrm{LINKF}(P)\qquad \mathrm{LINKB}(\mathrm{LINKF}(P))\leftarrow\mathrm{LINKB}(P)$$

（这就从 $\mathrm{AVAIL}[k]$ 表撤消块 P）然后置 $k\leftarrow k+1$，而且如果 $P<L$ 则置 $L\leftarrow P$。返回到 S1。

S3.〔放到表中〕 置

$$\mathrm{TAG}(L)\leftarrow 1\qquad \mathrm{LINKF}(L)\leftarrow\mathrm{AVAILF}[k]\qquad \mathrm{LINKB}(\mathrm{AVAILF}[k])\leftarrow L$$
$$\mathrm{KVAL}(L)\leftarrow k\qquad \mathrm{LINKB}(L)\leftarrow\mathrm{LOC}(\mathrm{AVAIL}[k])\qquad \mathrm{AVAILF}[k]\leftarrow L$$

（这就把块 L 放到 $\mathrm{AVAIL}[k]$ 表中）。

D. 诸方法之比较 对于这些动态的存储分配算法，要进行数学的分析，原来都是十分困难的。但是，有一个有趣的现象，那是相当容易来分析的，即“百分之五十的规则”：

如果算法A和B，以系统趋向于一个均衡状态这样一种方式，被连续地使用，其中在这个系统中平均有N个保留的块，每一块各具有独立的寿命，而且其中算法A中之量K取非0值（或者，更一般地，如同在步骤$A4'$中那样，取值$\geqslant c$）之概率为p，那末，可利用块的平均块数就近似地趋向于$\frac{1}{2}pN$。

这个规则告诉我们AVAIL表近似地将有多长。当数量p接近1时——如果c非常小而且如果块的大小很少彼此相等，则就将发生这种情况——我们将有大约为不可利用块之一半那么多的可利用块；因此就有“百分之五十规则”之称。

不难来导出这个规则。考虑以下的内存图：



这就说明，保留块分成了三类：

A：当释放时，可利用块的个数将减1；

B：当释放时，可利用块的个数将不变；

C：当释放时，可利用块的个数将加1。

现在命N是保留块的个数，并命M是可利用块的个数；命A，B，和C是上述标识类型的块数。我们有

$$\begin{aligned} N &= A + B + C \\ M &= \frac{1}{2}(2A + B + \epsilon) \end{aligned} \tag{15}$$

其中$\epsilon = 0$，1，或2，取决于下界和上界的条件。为导出百分之五十规则，我们置

$$M\text{加}1\text{的概率} = M\text{减}1\text{的概率}$$

（或者，更确切地说，在均衡期间，M于每个单位时间的平均变化被置成0）。这就导出

$$C = A + (1 - p)N ,$$

而且由(15)，同时假定ϵ是0（因为M和N都假定是相当大的），我们得到

$$N - 2M + A = A + (1 - p)N \tag{16}$$

从而得出百分之五十规则。这一推导事实上证明了，当M暂时地小于$\frac{1}{2}pN$时，则M将增加的概率就大于它将减小的概率，反之亦然。

除了这个有趣的规则外，我们对这些算法的性能的知识，几乎完全以蒙特卡罗实验为基础。读者将会发现，当他就一台特殊的机器和一个特殊的应用或一类应用，选择存储分配算法时，实施他自己的模拟实验是有启发的。作者就曾在刚要写这一小节之前，进行了若干次这样的实验（其实，在作出这个百分之五十规则的证明之前，在作这些实验的期间，就已经予示出这个规则）。这里，让我们简短地来检查一下这些实验的方法和结果。

基本的模拟程序运行如下，TIME开始时为0，而且内存区域开始时全都可利用：

P1. TIME增1。

P2. 释放系统中在TIME的当前值时所有被调度为要释放的块。

P3. 利用第3章的一些方法，计算以某些概率分布为基础的两个量S(一个随机的大小)和T(一个随机的"寿命")。

P4. 保留一个长度为S的新块，它在(TIME+T)时要被释放的。返回到P1。■

每当TIME是200的一个倍数时，就打印出关于保留和释放算法之性能的详细统计。S和T值的序列为每一对被测试的算法所使用。在TIME增加到超过2000之后，这个系统通常已达到一种或高或低的稳定状态，它给出了关于此后将无限地被保持的每一项指示。然而，由于可利用存储的总量以及在步骤P3中S和T的分布，分配算法有时候将不能找出足够的空间，模拟实验将因此而被终止。

命M是可利用的内存单元的总数，并命$\bar{S}$，$\bar{T}$表示步骤P3中S和T的平均值。容易看出，在任何给定的时间，一旦TIME充分大时，内存中不可利用字的预期个数是$\bar{S}\cdot\bar{T}$。当$\bar{S}\cdot\bar{T}$在实验中约大于$\frac{2}{3}M$时，通常往往是在真正需要M个内存字之前就出现内存溢出。当块的大小相对于M来说很小时，则内存有可能充满到百分之九十以上；但是当块的大小允许超过$\frac{1}{3}M$时(以至取更小的值时亦然)，则当事实上需要少于$\frac{1}{2}M$个单元时，内存即已趋向于"填满"。经验的证据强烈地提示，如果预期进行有效的操作，则大于$\frac{1}{10}M$的块大小不应为动态存储分配所使用。

实验以对于S的三种大小分布实施之：

($S1$) 均匀地选取在100与2000之间的整数。

($S2$) 分别以概率$\left(\frac{1}{2},\ \frac{1}{4},\ \frac{1}{8},\ \frac{1}{16},\ \frac{1}{32},\ \frac{1}{32}\right)$选定大小(1，2，4，8，16，32)

($S3$) 以相等的概率选择大小(10，12，14，16，18，20，30，40，50，60，70，80，90，100，150，200，250，500，1000，2000，3000，4000)。

时间分布T通常是一个随机的，对于固定的$t=10$，100，或1000，在1与t之间均匀地选取的整数。

实验也可以这样来进行，即在步骤P3中，在1与$\min\left(\left\lfloor\frac{4}{5}\cdot U\right\rfloor,\ 12500\right)$之间均匀地选择$T$，其中$U$是直到系统中的某个当前被保留的块下一次被调度释放为止，剩下的时间单位数。这种时间分布是用来模拟一个"几乎是后进先出的"行为；因为如果总是把T选择成$\leqslant U$，则存储分配系统就将退化成为只不过是一个不需要复杂的算法的堆栈操作(见习题1)。在这种情况下，T被选择成大于U的次数约为20%，所以我们就几乎有(而不是完全有)一个堆栈操作。当采用这种分布时，诸如A，B，和C这些算法，要比通常表现好得多。在整个AVAIL表中很少有多于两个的项目，而同时却有大约14块保留块。另一方面，当使用这种分布时，伙伴系统算法R和S是较慢的，因为它需要更经常地以一种类似堆栈的操作来分开和合并诸块。这个时间分布的理论性质，显得特别难推导(见习题32)。

这一节开头的图42，是在TIME=5000时的内存配置，用的是大小分布($S1$)和在1与100之间随机地选定的时间分布，利用的就是象上述算法A和B中那样的"首先适应"

的方法。对于这一实验，进入“百分之五十规则”之概率 p 实质上是 1，所以我们可以预期大约有保留块的一半那么多的可利用块。实际上，图 42 说明有 21 块可利用块和 53 块保留块。这并不否定百分之五十规则：例如，在 TIME＝4600 时有 25 块可利用块和 49 块保留块。图 42 中的配置只不过说明了百分之五十规则怎样地以统计的变化为条件。可利用块的块数范围一般是在 20 与 30 之间，而保留块的块数一般是在 45 与 55 之间。

图 43 示出了与图 42 相同的数据，但使用的是“最佳适应”而不是“首先适应”方法所得到的内存图。步骤 A4′ 中的常数 c 选择成16，以消除小块，而且结果概率 p 降低到约 0.7，并有更少的可利用区域。

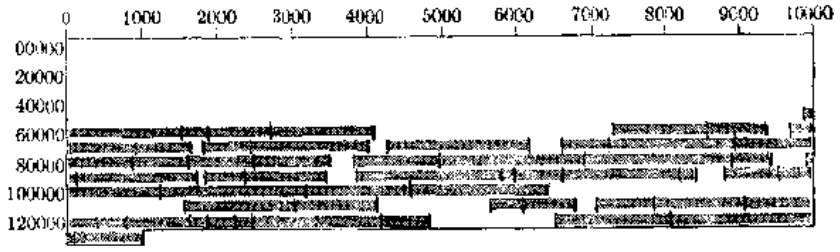


图43　用“最佳适应”方法得到的内存图（对照示出“首先适应”方法之图42，以及对于同一个存储要求序列示出“伙伴系统”之图44）。

当时间分布是从 1 到 1000 变化而不是从 1 到 100 时，就得到确切地类似于图 42 和 43 中所示的情况，而所有相应的数量都近似地乘以 10。例如，有 515 块保留块；并在图 42 的相应条件下有 240 块自由块，在图 43 的相应条件下有 176 块自由块。

在最佳适应与首先适应方法的所有实验中，后者总是显得更好些。当内存大小被穷尽时，在大多数情况下，在内存出现溢出之前，首先适应方法实际上比最佳适应方法更能长久地持续活动。

伙伴系统也可应用到导致图 42 和 43 的相同的数据上，从而得到图 44 作为其结果。这里，所有在 257 到 512 范围内的大小都被处理成 512，而所有在 513 与 1024 之间的大小都被提高到 1024，等等。平均说来，这意味着大约要求三分之四多的内存（见习题 21）；当然，伙伴系统对于象上边（$S2$）那样的大小分布，而不是（$S1$）那样的，要工作得更好。注意在图 44 中有大小为 2^9，2^{10}，2^{11}，2^{15}，2^{16}，和 2^{17} 的可利用块。

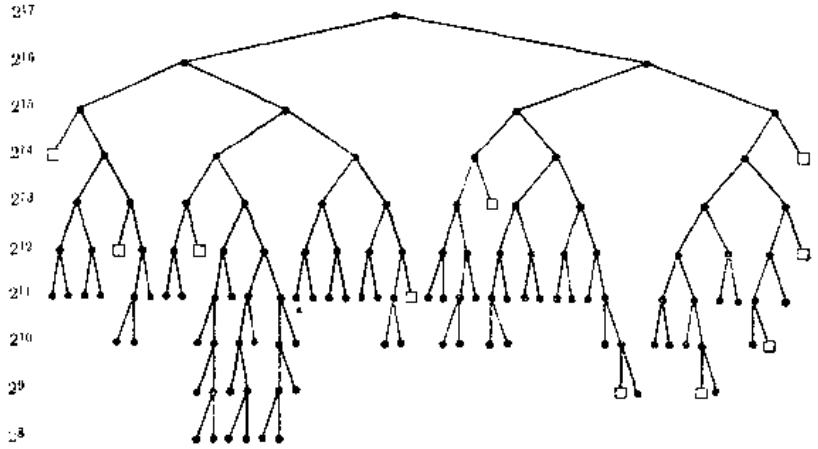


图44　用“伙伴系统”得到的内存图（这个树形结构指出了某些大块分成大小折半的“伙伴”。正方形表示可利用的块）

对伙伴系统的模拟说明，比最初预料要执行得好得多。显然，伙伴系统有时将允许两个相邻的区域都是可利用的，而不把它们合并成一个（设想它们不是"伙伴"）；但这种情况在图44中不出现，事实上，这在实践中是少见的。在出现内存溢出的情况下，内存百分之九十五被保留，而这反映了一种十分好的分配平衡。况且，很少有必要在算法R中分开诸块，或者在算法S中合并它们；在最普遍使用的诸层上，可利用块所属的树极象图44。为洞察在树之最低层上的这种特性，某些数学结果，已经由小保·沃·珀多姆（P. W. Purdom, Jr）和斯·麦·施蒂格勒（S. M. Stigler）得到，见*JACM* (17) 1970, 683～697。

还有另一件令人吃惊的事情，就是算法A在作了如习题6中所述的修改之后，所得到的出色的特性；平均说来，仅需要对于可利用块大小作2.8次检查〔使用大小分布（S1）和在1与1000之间均匀选取的时间〕，而且多一半的时间，仅仅需要极小的值，即一次迭代。尽管大约存在有250个可利用块这样一个事实，但这却是真的。对于未修改的算法A，同一实验证明，平均大约需要125次迭代（所以每次大约要检查AVAIL表的一半），而且20%的时间发现需要200次或更多次的迭代。

事实上，未修改的算法A的这种行为，可以作为百分之五十规则的一个推论而预测到。在平衡时，内存中包含最后$\frac{1}{2}$个保留块的部分，也将包含最后$\frac{1}{2}$个自由块；当一块被释放时，该部分将被卷入$\frac{1}{2}$的时间，从而它必须涉及$\frac{1}{2}$的分配以便维持平衡。当以任何其它的分数来代替"$\frac{1}{2}$"时，同样的论证亦成立（这些发现属于约·迈·罗布森（J. M. Robson））。

后面的习题包括对于两个主要算法的MIX程序（对于如习题12中的修改的算法A连同算法C，与伙伴系统进行比较），这些程序是由上述一个推论给出的，这里就是近似的结果：

	进行保留的时间	进行释放的时间
边界的TAG系统	$33+7A$	18, 29, 31, 或 34
伙伴系统	$19+25R$	$27+26S$

这里，$A \geqslant 1$是在检索一块足够大的可利用块时所需要的迭代次数；$R \geqslant 0$是一块分成二块的次数（在算法R中的$j-k$之初始差）；而$S \geqslant 0$是在算法S期间伙伴块被重新融合在一起的次数。模拟实验表明，在所述的假定下，用大小分布（S1）以及在1与1000之间选定的时间，平均说来，我们可以取$A=2.8$，$R=S=0.04$（当代换成如前所述的"几乎是后进先出"的时间分布时，则发现这个平均值为$A=1.3$，$R=S=0.9$）。这说明，两个方法都非常快，尽管伙伴系统在MIX的情况下比较起来更快些。记住，当块的大小不被限制为2的幂次时，伙伴系统就要求多出大约40%的空间。

对于习题33的废料收集和紧凑算法，其相应的时间估计是，大约要104个时间单位来找出一个自由节点，假定当内存近于半数充满时出现废料收集，并且假定诸节点有5个字的平均长度，又每个节点有2个链接。废料收集的正反两个方面，都已在2.3.5小节中讨论过了。当内存未被大量装入而且当满足适当的限制时，废料收集和紧凑是非常有效的；

例如，在MIX计算机上如果内存空间决不会充满到1/3以上，而且节点又是相当小的话，废料收集方法要比其它两个方法都要快。

同样的模拟技术也可应用于某些其它的存储分配算法。与这一小节的算法比较起来，其它的算法就显然太拙劣了，因此在这里仅仅简单地提及之：

a）对每个大小保存独立的AVAIL表。一个单一的自由块必要时偶而也分成两个更小的块，但并不试图再把这样的块重又合到一起。内存图破碎为越来越细的部分，直至濒于细如毫毛的怪状；一个与此相象的简单的方案，几乎相当于在分散的区域里来做独立的分配，对于每个块的大小都有一个区域。

b）企图来进行“二级”分配：把内存分成32个大段。用“蛮干”分配方法来保留1，2，或3份（很难再多了）相邻段的大块。每个这样的大块又被进一步分开以满足存储要求，直到在当前的大块内已不再剩有余地为止；而后保留另一大块供其后的分配使用。仅当每一大块内的所有空间都已成为可利用时，才把它恢复到自由存储。这个方法几乎总是非常快地用完存储空间。

尽管这一特殊的“二级”分配的方法，对于作者的模拟实验中所考虑的数据是失败的，却存在有其它一些情况（在实践中不经常出现），多级分配技巧可能有益（例如，考虑分若干阶段工作的比较大的程序，其中已经知道在某个子程序内仅仅需要某种类型的节点）。在同一程序中对于不同的节点类使用完全不同的分配策略，也可能是所希望的。通过“带区”来分配存储，在每个带区采用可能是十分不同的策略以及能一次释放整个带区，这样的思想是由道格拉斯·泰·罗斯(Douglas T. Ross)在CACM 10(1967)，481～492中予以讨论的。

关于动态存储分配进一步的实验结果，请参看布·兰德尔在CACM 12(1969)，365～369，372；保·沃·珀多姆，斯·麦·施蒂格勒和T. O. 奇亚姆（T. O. Cheam）在*BIT* 11(1971)，187～195；巴·赫·马戈林（B. H. Margolin）。理·佩·帕马利（R. P. Parmelee）和马沙特佐夫（M. Schatzoff）在《*IBM*系统杂志》(*IBM Systems J.*)10(1971)，283～304中的论文。

E. 溢出 当不再有空间可利用时，我们应该做些什么呢？假设，有一个要求，比如说，要求n个连续的字，当所有可利用的块都太小时。这种情况头一次发生时，通常有多于n个的可利用的单元存在，但它们不是连续的；“紧凑内存”（即是移动某些使用中的单元，使得所有可利用的单元都被挪到一起）将意味着我们可以继续处理。但紧凑是缓慢的；而且绝大多数情况是，当“首先适应”的方法穷尽所有余地时，不论进行多少次紧凑和再紧凑，实际上此后不久空间也就完全用光了。因此，除非是在与废料收集相联系的特殊情况下，如同习题33中那样，一般说来已不值得来写紧凑程序。如果预期会出现溢出，则可以使用从内存撤消某些项目并把它们转存到一个外部的存储装置上的某种方法，同时提供必要时再次把信息送回的条件。这就意味着，所有访问动态存储区域的程序，就其对其它块所作的访问来说，必须加以严格的限制，而且一般说来，要求特殊的计算机硬件（例如，为缺乏数据而中断，或自动地“分页”）在这些条件下能进行有效的操作。

为了判定哪些块是最可能撤消的候选者，就必须有某些判定过程。一种想法是维持一份保留块的双重链接表，每当其中某个块被访问时，就把它移动到该表的前头；这样，诸

块即以它们最近访问的次序，进行了有效的分类，而且在这个表的末尾的块是首先要撤消的块。通过把保留块放进一个循环表中，并且在每一块中包括一位“最近使用”位，就可以更简单地达到一种类似的效果；每当一块被访问时，该位即置成 1。当要撤消一个块的时候，一个指针沿着这个循环表移动，复置所有的“最近使用”位成 0，直到找到了一块自从指针上一次到达圆周上这个部分以来未曾使用的块为止。

约·迈·罗布森已经证明〔*JACM* 18(1971), 416～423〕未对保留块重新定址的动态存储分配策略，不可能保证有效地使用内存，即总是会有使这方法破产的病理情况。例如，甚至当诸块均被限制成为大小为 1 和 2 时，且内存仅仅 2/3 被填满，不论使用什么分配算法，仍可出现溢出！习题 36–40 中综述了罗布森的有趣结果。

习题

1.〔20〕 对于这一节的保留和释放算法，可以作些什么简化，如果存储要求总是以“后进先出”的方式出现，即直到其后所有被保留的块已经被释放之后，再没有保留块要释放了？

2.〔*HM*23〕（埃·沃尔曼(E. Wolman)）假设我们要为可变长度的项目选择固定节点的大小，且又假设当每个节点有长度 k 而一个项目有长度 l 时，就采用 $\lceil l/(k-b)\rceil$ 个节点来存这个项目(这里 b 是一个常数，表示每个节点包含诸如指向下一节点的链接等控制信息的 b 个字)。如果一个记录的长度 l，平均说来是 L，则 k 应如何选择，以使所要求的存储空间的平均数量为极小？(假定对于任何固定的 k，当 l 变化时，$(l/(k-b+1)) \bmod 1$ 的平均值等于 $\frac{1}{2}$)。

3.〔40〕 通过计算机模拟，来比较存储分配的最佳适应，首先适应，与“最坏适应”的方法；在后一种方法下，总是选择最大的可利用块。在内存的用法方面，有什么显著的差别？

4.〔22〕 为算法 A 编写一个 MIX 程序，特别要着眼于力求内循环运行快。假定 SIZE 场是 (4:5)，LINK 场是 (0:2)，以及 A < 0。

▶5.〔18〕 假设已知 N 在算法 A 中总是 100 或更大，则在修改的步骤 A4′ 中置 $c = 100$ 是个好主意吗？

▶6.〔23〕 在算法 A 已经反复地使用之后，有一种强烈的趋向，把 SIZE 小的块保存在 AVAIL 表的前头，以致通常在找到一个长度为 N 或更大的块之前，必须远远地进入这张表来进行检索。例如，注意在图 42 中，诸块的大小，对于保留块和自由块两者，从内存的开始到末尾，实质上是如何地在增加（在准备图 42 的同时，所用的 AVAIL 表已保持了按单元顺序的分类，如同算法 B 所要求的那样）。你能不能提出一个修改算法 A 的方法，使得(a) 短小的块不趋向于累积于一个特殊的区域，以及 (b) AVAIL 表仍然可以以内存单元递增的顺序来保持，以利于诸如算法 B 那样的一些算法？

7.〔10〕 例 (1) 说明有时候“首先适应”方法确乎是优越于“最佳适应”方法。试给出一个类似的例子，以示出一种“最佳适应”方法优于“首先适应”方法的情况。

8.〔21〕 说明怎样以一种简单的方式来修改算法 A，以得到关于“最佳适应”方法的

算法，来代替“首先适应”方法。

▶9.〔26〕 用“最佳适应”方法，以什么方式来设计一个保留算法，使其避免检索整个的 AVAIL 表？（试尽可能多地想出能减少必要检索的方法）。

10.〔22〕 说明怎样来修改算法 B，使得从单元 PO 开始的 N 个连续单元的块，成为可利用的，而无须假定这 N 个单元中的每一个当前是否可利用；事实上，假定这个正被释放的区域实际上可以与已经被释放的若干块相重叠。

11.〔*M*25〕 证明：在习题 6 的答案中所提议的对算法 A 的改进，也可用来实现对算法 B 的一点改进，它把平均检索长度由 AVAIL 表长度的一半减少到这个长度的 1/3（假定这个被释放的块将被插入到已分类的 AVAIL 表内的一个随机的位置处）。

▶12.〔20〕 请修改算法 A，使得它遵从（7）的约定，使用正文中所述的修改的步骤 A4′，并且也加入习题 6 中所作的改进。

13.〔21〕 为习题 12 的算法编写一个 MIX 程序。

14.〔21〕（a）如果 SIZE 场不出现在一自由块最后的字中，或者（b）如果 SIZE 场不出现在一保留块的头一个字中，则将对于算法 C 和习题 12 的算法产生什么影响？

▶15.〔24〕 以稍微长些的程序为代价，在取决于 TAG(PO−1), TAG(PO+SIZE(PO)) 是正还是负的四种情况中的每一种情况，除绝对必要外不改变任何更多的链接，试说明怎样来提高算法 C 的速度。

16.〔24〕 请利用习题 15 的思想，来为算法 C 编写一个 MIX 程序。

17.〔10〕 当不出现可利用的块时，（9）中的 LOC(AVAIL) 和 LOC(AVAIL)+1 的内容将是什么？

▶18.〔20〕 图 42 和 43 是利用相同的数据，以及实际上相同的算法（算法 A 和 B），而得到的，只是图 43 是通过把算法 A 加以修改来选择“最佳适应”以代替“首先适应”而作出的。为什么这就使得图 42 有一块大的可利用区域在内存的较高单元中，而在图 43 中则有一块大的可利用区域在较低的单元中呢？

▶19.〔24〕 假设内存中诸块都有（7）的形式，但在块的最后一个字中没有所需要的 TAG 或 SIZE 场。进一步假设，下述简单的算法正被用来再次地释放一块保留的块：“Q←AVAIL，LINK(PO)←Q，LINK(PO+1)←LOC(AVAIL)，LINK(Q+1)←PO，AVAIL←PO，TAG(PO)←“−”·”（该算法对于把相邻的区域融合成为一体方面，无所作为）。

试说明有可能来设计一个类似于算法 A 的保留算法，它在检索 AVAIL 表的同时对相邻的自由块进行必要的融合，同时避免对内存进行（2），（3）和（4）中那样不必要的分段。

20.〔00〕 为什么希望在伙伴系统中的 AVAIL[*k*]表都是双重链接的，而不是简单、直接的线性表？

21.〔*HM*25〕 考察当 n 趋于无穷时的比率 a_n/b_n，其中 a_n 是 1+2+4+4+8+8+8+8+16+16+…的头 n 项之和，而 b_n 是 1+2+3+4+5+6+7+8+9+10+…的头 n 项之和。

▶22.〔21〕 正文反复地说到，伙伴系统仅只允许使用大小为 2^k 的块，而且习题 21 表

明这可能导致所要求存储实质性地增加。但如果在伙伴系统方面需要一个 11 个字的块时，那为什么我们不能找一个 16 个字的块，并把它分成为一份 11 个字的片段连同两块大小分别为 4 和 1 的自由块呢？

23.〔05〕 二进地址为 011011110000，大小为 4 的块，其伙伴的二进地址是什么？当块的大小为 16 又将如何？

24.〔20〕 按照正文中的算法（大小为 2^m 的）、最大的块将没有伙伴，因为它表示存储的全部。这话对吗？如果定义 $\text{buddy}_m(0)=0$（即命这一块就是它自己的伙伴），便从而避免在步骤 S1 中对 $k=m$ 的测试？

▶25.〔22〕 鉴定下述的想法："在实际的情况下，使用伙伴系统的动态存储分配将不保留大小为 2^m 的一个块（因为这将填满整个内存），而且一般说来，有一个极大的大小为 2^m 的块，对于该块大小说来，已不再有更大的块可予保留了。因此，从这样大的块为可利用者开始，当结合后的块有大于 2^m 时，在算法 S 中还要把伙伴结合起来，将是浪费时间"。

▶26.〔21〕 试阐明伙伴系统如何可用于从内存单元 0 开始到 M－1 之间的动态存储分配，即便 M 不象正文中所要求的那样有 2^m 之形式。

27.〔24〕 为算法 R 写出一个 MIX 程序，并确定它的运行时间。

28.〔25〕 假定MIX是一台二进计算机，并有如下定义的一个新操作码 XOR（用 1.3.1 小节的记号）："C＝5，F＝5；对应单元M中每个等于 1 的位上，在寄存器 A 的对应位上取补（把 0 变成 1 或 1 变成 0）"。

试为算法 S 写出一个 MIX 程序，并确定它的运行时间。

29.〔20〕 能否把伙伴系统加以修改，以避免在每一保留块中设标志位 TAG？

30.〔*M48*〕 试给出一些合理的关于存储要求序列的分布，来分析伙伴系统的性质，特别是算法 R 和 S 的平均速度。

31.〔*M40*〕 能否设计一个类似于伙伴系统的存储分配系统——利用斐波那契序列来代替 2 的次幂？（这样，我们可以从 F_m 个可利用的字开始，而且可以把有 F_k 个字的一个可利用块分开成为长度分别为 F_{k-1} 和 F_{k-2} 的两个伙伴）。

32.〔*HM47*〕 试确定 $\lim_{n\to\infty}\alpha_n$，如果它存在的话，其中 α_n 是在一个如下定义的序列中 t_n 的平均值：命

$$g_k=\left\lfloor\frac{5}{4}-\min(10000,\ f(t_{k-1}-1),\ f(t_{k-2}-2),\cdots,\ f(t_1-(k-1)))\right\rfloor$$

其中如果 $x>0$，则有 $f(x)=x$；如果 $x\leqslant 0$，则 $f(x)=\infty$。量 t_k 取值 $1,2,\cdots,g_k$ 中之任何一个的概率为 $1/g_k$。（注意：某些有限的经验测试表明，α_n 可能近似于 14，但这不是很精确的）。

▶33.〔28〕（废料收集和紧凑）。假定内存单元 1 到 AVAIL－1 正被用作有如下形式的可变大小的节点的一个存储池：NODE（P）的头一个字含有场

SIZE(P)＝NODE(P) 中的字数；

T(P)＝NODE(P) 中链接场的个数；T(P)<SIZE(P)；

LINK(P)＝仅在废料收集期间使用的特殊的链接场。

在内存中紧接 NODE(P)之后的是节点 NODE(P＋SIZE(P))。假定，NODE(P)中仅

有的用作链到其它节点的链接场是LINK(P+1),LINK(P+2),…,LINK(P+T(P)),而且这些链接场中的每一个，或者是A，或者是另一个节点的头一个字的地址。最后，假定在程序中另有一个进一步的链接变量，称作USE，它指向这些节点之一。

试设计一个算法，它能：(a) 确定出所有的可由变量USE直接地或间接地访问的节点，(b) 改变所有链接使得保持结构关系，对于某个K，把这些节点移入内存单元1到K−1；以及(c) 置AVAIL←K。

例如，考虑内存的下列内容，其中INFO(L)表示单元L除LINK(L)外的内容：

1: SIZE = 2, T = 1	6: SIZE = 2, T = 0	AVAIL = 11,
2: LINK = 6, INFO = A	7: CONTENTS = D	USE = 3
3: SIZE = 3, T = 1	8: SIZE = 3, T = 2	
4: LINK = 8, INFO = B	9: LINK = 8, INFO = E	
5: CONTENTS = C	10: LINK = 3, INFO = F	

你的算法应当把这个变换成

1: SIZE = 3, T = 1	4: SIZE = 3, T = 2	AVAIL = 7,
2: LINK = 4, INFO = B	5: LINK = 4, INFO = E	USE = 1
3: CONTENTS = C	6: LINK = 1, INFO = F	

31. [29] 为习题33的算法编写一个MIX程序，并确定它的运行时间。

35. [22] 把这一小节的动态存储分配方法，与在2.2.2小节末尾讨论的可变大小的顺序表技术加以对比。

▶36. [20] 在加利福尼亚的好莱坞，有某个便餐台，一排含有23个席位。用餐者以一个或二人一伙进入这餐馆，女招待员指示他们在那里就座。试证明，如果没有单独来到的顾客被指定到号码为2，5，8，…，20中的任何一个席位上，则这位女招待员总能立即让人们入席而不必分开任何一对，假定任何时候都不会有多于16个顾客出现（各对保持在一起）。

▶37. [26] 继续习题36，证明当便餐台仅有22个席位时，女招待员不可能总是这样好地进行工作：不管她使用什么策略，总是可能碰到这样一种情况，就是两个朋友进入餐馆，而且仅有14人已入席，但是却没有两个相邻的座位空着。

38. [*M*21] (约迈·罗布森)习题36和37的便餐台问题可以推而广之，来说明任何不对保留块重新进行再定址的动态存储分配算法的最坏情况的性能。命$N(n, m)$是最小的使得任何要求分配和释放的序列均可不产生溢出而被处理的内存数量，假定所有块的大小都$\leqslant m$而且所要求的空间总数总不超过n。习题36和37证明了$N(16, 2)=23$；试对于所有的n，确定$N(n, 2)$的精确值。

39. [*HM*23] (约·迈·罗布森)利用习题38的记号，证明$N(n_1+n_2, m)\leqslant N(n_1, m)+N(n_2, m)+N(2m-2, m)$；因此对于固定的$m$，$\lim_{n\to\infty}N(n, m)/n=N(m)$存在。

40. [*HM*50] 继续习题39，如果它存在的话，确定$N(3)$，$N(1)$，以及$\lim_{m\to\infty}N(m)/\lg m$。

41. [*M*27] 这道习题的目的，是考虑伙伴系统的最坏情况的内存用法。例如，会出现这样一个特别坏的情况——如果我们从一个空的内存开始，并如下进行之：首先保留长度为1的$n=2^{r-1}$个块，它从单元0开始到$n-1$；然后对于$k=1, 2, \ldots, r$，释放全部

其开始单元可被 2^k 整除的块，并保留 $2^{r-k-1}n$ 个长度为 2^k 的块。这是从单元 $\frac{1}{2}(1+k)n$ 开始到 $\frac{1}{2}(2+k)n-1$ 为止的块。这个过程竟动用了如此之多的内存，达到曾被占用过内存的 $1+\frac{1}{2}r$ 倍！

证明最坏的情况实质上不会比以下这种情况更坏：当所有的请求是对于块的大小为 $1, 2, \cdots, 2^r$ 的时候，而且在任何时候所要求的总空间决不超过 n，其中 n 是 2^r 的一个倍数，那么，伙伴系统将不会溢出一个大小为 $(r+1)n$ 的内存区域。

2.6 历史和参考文献

保存在连续内存单元中的信息的线性表和矩形数组，从最早期的存储程序式计算机问世以来，就已被广泛地应用了，而且最早的关于程序设计的一些论文，就已经给出过遍历这些结构的基本算法。〔例如，见约·冯·诺依曼，《选集》5，113–116（写于 1946 年）；莫·文·威尔克斯，戴·约·惠勒、斯·吉尔：《为电子数字计算机准备程序》（马萨诸塞，雷丁：爱迪生–韦斯利，1951），子程序 V－1〕在使用变址寄存器之前，顺序线性表的操作，乃是通过对机器语言指令本身实施算术运算来完成的，而且这种类型的操作，成为早期追求一台这样的计算机的动机之一，即它的程序与程序所加工的数据，共享内存空间。

允许可变长的线性表，以这样一种方式来共享顺序的单元——即必要时向前或向后移动之，如同在 2.2.2 小节中所描述的那样，这种技术，显然是更晚得多的发明。迪吉提克（Digitek）公司的詹·邓拉普（J. Dunlap）于 1963 年，联系到设计一系列的编译程序，发明了这些技术。大约与此同时，这思想也独立地出现于 IBM 公司的 COBOL 编译程序的设计中，而且一套称为 CITRUS 的有关子程序的汇集，陆续地使用于各种装置中。这些技术一直保密，直到挪威的简·加威克独立地提出了它们之后松开；见 BIT 4（1964），137–140。

存线性表于非顺序单元中的思想，其起源似乎与设计带有磁鼓存储器的计算机有关。在执行了单元 n 中的指令之后，这样一种计算机通常不准备从单元 $n+1$ 中来得到它的下一条指令，因为磁鼓已经转过了这一点。这取决于正被实施的指令，下一条指令的最有利的位置，可能是 $n+7$ 或 $n+18$，等等；如果机器的指令最佳地而不是连续地来放置，则这种机器的操作速度可提高六到七倍〔至于谈到这些指令的最佳位置，这一有趣的问题，见作者发表在 JACM 8（1961），119～150 页的论文中的讨论〕。因此，这类机器的设计是在每条机器语言指令中提供了一个额外的地址场，作为指向下一条指令的链接。这样一种机器叫做“一加一地址计算机”，以区别于“一地址计算机” MIX。一加一地址计算机的设计，显然是在计算机程序内头一次出现链接表的思想，尽管我们在这一章中如此频繁使用的动态插入和删去操作，当时尚未被人们知晓。一加一编址是约·莫兹里（J. Mauchly）于 1946 年讨论的〔《电子计算机设计的理论和技术》（*Theory and techniques for the design of electronic computers*）4（宾夕法尼亚大学，1946）讲义 37〕。早期程序链接的另一个出现，见汉·彼·卢恩（H. P. Luhn）提议对外部检索使用“链接”的 1963 年

备忘录；参考6.4节。

当艾·纽厄尔、约·克·肖和赫·阿·西蒙开始他们的“由机器启发式地来解决问题”的研究时，链接的存储技术就真正地产生了。为编写寻求数理逻辑证明的程序，作为一项辅助工具，他们于1956年春天设计了头一个“表处理”语言IPL-Ⅱ(IPL表示Information Processing Language)。这已经是一个这样的系统，即它使用了链接并且包括了诸如可利用空间表这样一些重要概念，但是，当时堆栈的概念尚未完善地形成；IPL-Ⅲ是在一年之后设计的，它包含了堆栈的“下推”和“弹出”，作为重要的基本操作(为参考IPL-Ⅱ，见《IRE通报，信息论部分》(IRE Transactions on Information Theory)IT-2(1956年9月)61-70；《西部计算机会议会报》(Proc. Western Joint Comp. Conf.)(1957年2月)，218-240，关于IPL-Ⅲ的材料，第一次出现于1957年夏天密执安大学的课程讲义中)。

纽厄尔，肖和西蒙的工作，鼓励了许多其他的人们，来使用链接存储(当时这叫做NSS存储)，主要用于模拟人类思维的过程。这些技术逐渐被确认为计算机程序设计的基本工具；描述链接内存储对于“物质的”问题之有用性的第一篇论文，是约·韦·卡尔Ⅲ(J. W. Carr, Ⅲ)发表于CACM 2，2(Feb. 1959)，4-6上。卡尔在这篇论文中指出，在通常的程序设计语言中，可以很容易地处理链接表，而无须要求有灵巧的子程序或解释系统。还可看格·安·布劳维(G. A. Blaauw)的文章，《IBM研究和发展杂志》(IBM J. Res. and Dev.) 3 (1959)，288～301。

开始，在链接表中使用一个字的节点，但是大约1959年，一些不同团体的人们，逐渐发现了每个节点可有若干个连续的字和“多重链接”表的有用价值。专门讨论这一思想的头一篇论文，是道·泰·罗斯发表的，CACM 4 (1961)，147～150；当时他对于在这一章中已被称作节点(node)的对象，使用了“丛(plex)”这个词，但他随后又在不同的意义下来使用plex这一词，来表示与有关的遍历它们的算法相结合的一个节点类。

为访问节点内的场，所用的记号一般有两种：场的名称或者是在指针的名称之前，或者在其之后。于是，尽管我们在这一章中已经写成“INFO(P)”，另一些作者却写成“P.INFO”。在准备编写这一章时，这两种记号似乎同样都是可取的。但是，这里采用的记号更有其优越性，这就是，如果我们定义INFO和LINK数组，并且使用P作为下标，则它就立即可翻译成FORTRAN，COBOL等类似的语言。况且使用数学函数记号来描述一个节点的属性更显得自然些。注意“INFO(P)”在通常的数学表示中，读作“P的INFO，信息”，恰如$f(x)$被读作“x的f”。而另外一种记法P.INFO，因为它趋向于强调P，尽管它也可以读作“P的INFO”，但不太自然；INFO(P)显得更好的原因，显然在于这样一个事实，即当使用此记号时，P是变量，但INFO则有着一个固定的意义。照此类推，我们可以把一个向量$A=(A[1], A[2], \cdots, A[100])$考虑成为一个有100个名为1，2，…，100的场的节点。现在在我们的记号下，第2个场将作为“2(P)”来访问之，其中P指向向量A；但如果我们正在访问的是该向量的第j个元素，则我们发现把变量“j”写在第二位，写成$A[j]$更为自然些。类似地，把可变的量“P”写在记号INFO(P)中的第二位，似乎也更为适当。

也许，第一个认识到“准栈”(后进先出，LIFO)和“排队”(先进先出，FIFO)概念

是重要研究对象的人，是关注于降低所得税估定的成本会计员；关于存货计价的“LIFO”和“FIFO”方法的讨论，可参看任何中等会计教程。例如查·福·施拉特和威·约·施拉特的《成本结算》(*Cost Accounting*)(纽约：威利，1957)，第7章。于1947年，艾·马·图林提出了一个堆栈，称为反转存储，用于子程序的链接(见1.4.5小节)。毫无疑问，就保存在顺序内存单元中的堆栈来说，其简单的使用早已遍及于计算机程序设计之中，因为堆栈是一种简单和直观的概念。如上所述，以链接形式出现的堆栈程序设计，首先是IPL；堆栈一词是从IPL的术语中引出的(尽管“下推表”在IPL中是更正式的词汇)，在欧洲，它是由埃·怀·迪克斯特拉独立地引进的。“双排队”一词是由厄·贾·施韦普(E.J.Schweppe)创造的。

循环的和双重链接的表，其起源说不清楚；也许这些思想对于许多人来说是自然出现的。一种促使这些技术普及的强烈因素，是以它们为基础的一般列表处理系统的存在(主要是约·韦普鲍姆的《打结的列表结构》(Knotted List Structures)CACM5(1962)，161～165以及《对称的列表处理程序》(Symmetric List Processor)，CACM 6 (1963)，524～544)。

自从早期的计算机出现以来，有才干的程序员们已经各自地发展了种种进行编址和遍历多维信息数组的方法，从而就出现了另一部分未发表的计算机的民间创作。这一课题是由赫·赫勒曼（H.Hellerman）首先加以书面综述的，见CACM 5 (1962)，205～207。也可见约·克·高尔（J.C.Gower)，《计算机杂志》(Comp.J.)，4 (1962)，286～286。

明显地表示在计算机内存中的树结构，开始是用来处理代数公式的。由格·默·霍珀于1951年建立的A-1编译程序语言，使用以三地址代码写成的算术表达式；这三地址代码就等价于一个二叉树表示的INFO，LLINK，和RLINK。1952年，哈·乔·卡里美尼安(H.G.Kahrimanian)建立了为求以A-1编译程序语言表示的代数公式之微商的算法；见《自动程序设计讨论会》(*Symposium on Automatic Programming*)(美国华盛顿：海军研究所，1954年5月)，5～14。

从此以后，各种外观的树结构，联系到大量的计算机应用中，已由许多人独立地进行了研究；但是，对于树操作的基本技术（不是一般的列表操作）除了具体算法的详细说明之外，很少出现在出版物中。肯·尤·艾弗森和莱·罗·约翰逊（L.R.Johnson），联系到所有数据结构更一般的研究，第一次作出了一般的综述[《IBM公司研究报告》(IBM Corp. research reports) RC-390，RC-603，1961；见艾弗森的《一个程序设计语言》(A Programming Language)(纽约：威利，1962)，第3章]。也见杰·萨尔顿，CACM 5 (1962)，103～114。

穿线树的概念是由艾·J·珀利斯和查·桑顿给出的，CACM 3 (1960)，195～204。他们的文章也介绍了在各种次序下遍历树的重要思想，并给出了代数操作算法的大量例子。遗憾的是，这一篇重要的文章是匆忙抛出的，因此它包含许多印刷错误。珀利斯和桑顿的穿线树，按照我们的术语实质上只是“右穿线的树”；在两个方向下都穿线的二叉树，是由阿·沃·霍尔特（A.W.Holt）独立地发现的，见《树结构的数学与应用研究》(A Mathematical and Applied Investigation of Tree Structures)(宾夕法尼亚大学博士论文，1963)。树节点的后根次序和先根次序，被泽·波莱克（Z.Pawlak）称作“正常进行的次序”和“双向进行的次序”，《数学基础讨论会》(Colloquium on the Foundation of Mathematics)，等等（蒂豪尼，1962，由Akadémia Kiadó出版，布达佩斯，1965)，227～238。在上面

引证的艾弗森和约翰逊的文献中，先根次序又称作“子树次序”。安·格·奥廷格（A.G. Oettinger）描述了表示在树结构与对应的线性记号之间联系的图式方式，《哈佛大学关于数字计算机和它们的应用讨论会会报》(Proc. Harvard Symp. on Digital Computers and their Application)（1961年4月)203～224，索·戈尼（S.Gorn）介绍了通过次数在先根次序下来表示树，连同把这个表示与杜威记数法以及树的其它性质加以联系的有关算法，见《自动机的数学理论讨论会会报》(Proc. Symp. Math. Theory of Automata)（布鲁克林：多种技术研究所，1962），223～240。

在2.3.4.6小节中，回顾了树结构作为数学对象的历史，并给出了这一课题的参考文献。

当编写这一节时，关于信息结构最普遍的知识，乃是来自程序员对列表处理系统的揭示——列表处理系统在这个历史中有着非常重要的地位。第一个广泛使用的系统是IPL-V（IPL-Ⅱ的后继，迟至1959年才建立的）；IPL-V是一个解释系统，程序员得到一种关于列表操作的类似的机器语言。同时，FLPL（对于列表处理的一组FORTRAN子程序，它也是受到IPL的启发，但使用子程序调用以代替解释语言）为赫·格伦特(H.Gelernter)和其他人所建立。第三个系统LISP，为约·麦卡锡（J.McCarthy）所设计，也是在1959年。LISP截然不同于它的先驱者：它的程序都是表达成与“条件表达式”（见第8章）相结合的数学函数记号，然后变换成一种列表表示。自那以后，已有许多列表处理系统问世，它们当中在历史上最突出的是约·韦曾鲍姆的SLIP；这是一组以双重链接列表进行操作的，用于FORTRAN程序的子程序。

博布罗（Bobrow）和拉菲尔（Raphael）的一篇论文，CACM 7 (1964)，231～240，可以作为对IPL-V，LISP和SLIP的一个简短介绍，而且它还给出了对这些系统的比较。菲·梅·伍德沃德（P.M.Woodward）和D.P.詹金斯（D.P.Jenkins）对LISP给出了精采的介绍，Comp.J. 4 (1961)，47～53。也请见作者们对他们自己的系统的讨论，他们的每一篇论文都有着重要的历史意义：艾·纽厄尔和弗·麦·汤格（F.M.Tonge）的《IPL-V简介》(An introduction to IPL-V，CACM 3 (1960)，205～211；赫·格伦特詹·罗·汉森（J.R.Hansen）和卡·卢·格贝里茨（C.L.Gerberich）的《一个编译FORTRAN的列表处理语言》（A FORTRAN-Compiled List Processing Language)，JACM7(1960)，87～101；约翰·麦卡锡的《符号表达式的递归函数和它们由机器进行的计算》(Recursive functions of symbolic expressions and their computation by machine) CACM 3 (1960)，184～195；约·韦曾鲍姆的《对称列表处理器》(Symmetric List Processor)，CACM 6 (1963)，524～544。这后一篇论文包括一个用于SLIP中的所有算法的完备描述。近年来，关于这些系统也已经写出了一些书。

还出现了若干字符串处理系统。这些系统主要是讨论关于字符信息的可变长字符串的运算（寻找某些子串的出现，等等）。其中，历史上最重要的曾经是COMIT（维·休·英韦（V.H.Yngve)，CACM 6 (1963)，83～84）和SNOBOL（戴·杰·法伯（D. J.Farber)，拉·爱·格里斯沃尔德（R.E.Griswold)，和伊·保·波伦斯基(I.P.Polonsky). JACM 11 (1964)，21～30)。尽管字符串处理系统已得到广泛使用，而且它们主要是组成如同我们在这一章中所见到的这样一些算法，但是，它们在信息结构表示技术的历史上，

所起的作用相对说来比较小；这些系统的用户已经大大地摆脱通过计算机来实现的实际内部过程的细节。关于字符串操作技术的概述，见斯·埃·马德尼克（S.E.Madnick），CACM 10（1967），420-424。

关于列表处理的IPL-V和FLPL系统就共享列表问题而言既不使用废料收集，也不使用访问计数技术；代替的是，每个列表为一个列表"所固有"，又为访问这一列表的任何其它列表"所借用"，而且当一个列表的"持有者"允许时，可删去一个列表。因此，程序员被责成来确保，不得有列表借用任何正被删去的列表。关于列表的访问计数器技术，是由乔·埃·柯林斯（G.E.Collins）引进的，CACM 3（1960），655-657；也请见这篇文章的一个重要的后续，CACM 9（1966），578-588。废料收集第一次描述于上边引证的麦卡锡的论文中；还可见CACM 7（1964），38，以及柯恩（Cohen）和特里林（Trilling）的一篇论文，BIT 7（1967），22-30。

六十年代里，对链接处理的重要性日益增长认识，自然导致它们侵入到代数程序设计语言中——允许程序员选择适当的数据表示形式，而不必乞怜于汇编语言或付出完全是一般的列表结构的开销。在这一发展中的某些基本的步骤，有这样一些人的工作：查·安·里·霍尔〔《符号处理语言和技术》（Symbol Manepulation Language and Techniques），由丹·古·博比罗编，（阿姆斯特丹、北荷兰，1968），262-284〕，哈·威·劳森〔CACM 10(1967)，358-367〕，奥-约·达尔和克·尼加德〔CACM 9（1966），671-678〕，A·范·威因恩登（A.van Wijngaarden）等〔Numerische Math 14（1969），79-218〕。

关于动态存储分配算法的消息公开之前，它们就已经被使用若干年了。韦·T·康福特（W.T.Comfort）给出了一篇非常适合讨论的论述，CACM 7（1964），357-362（于1961年的一篇论文中）。在2.5节中所介绍的"边界标志"方法，是作者于1962年设计的，用于B-5000计算机的一个控制程序。"伙伴系统"首先是由哈·马柯维兹（H.Markowitz），联系到SCRIPT的程序设计系统，于1963年使用的，而且肯·诺尔顿（K.Knowlton）也独立地发现并发表之，CACM 8（1965），623-625；也见CACM 9（1966），616-625。关于动态存储分配的进一步讨论，见艾利夫（Iliffe）和乔德特（Jodeit）的论文，Comp.J. 5（1962），200-209；贝利（Bailey），巴尼特（Barnett），和伯利森（Burleson），CACM 7（1964），339-346；伯齐提斯（Berztiss），CACM 8（1965），512-513；以及道·泰·罗斯，CACM 10（1967），481-492。

玛丽·德英比里约（Mary d'Imperio）的《数据结构及其在存储器中的表示》（Data Structures and their Representation in Storage），《自动程序设计年鉴》（Annual Review in Automatic Programming）5（牛津：珀加孟出版社），1969），给出了一个关于信息结构与程序设计的一般关系的讨论。这篇论文，也是对于这一课题历史的一个有价值的指南，因为它包括有，联系到十二个列表处理和字符串处理系统所用结构的详细分析。也可见两个讨论会的会报，CACM 3（1960），183-234和CACM 9（1966），567-643，以了解进一步的历史细节（这些会报中若干独立的文章，在上边已引证过了）。

一份优秀的带注解的参考书目，已由琼·伊·萨米特编辑出来——《计算评论》（Comput.Rev.）7（1966年7－8月），B1—B31。它主要面向符号处理和代数公式处理的应用，但是与这一章的内容有着大量的联系。

在这一章中，我们已详尽地考察了信息结构的各种具体类型，而且（为免得我们见树不见林），还清点了一下我们所学过的内容以及从一个更宽阔的视野来简单地概括关于信息结构的一般课题。这也许是明智的：从节点（作为数据的元素）的基本思想开始，我们已经见过许多例子，它们都说明了这样的一个事实，即或者是隐含地（以节点在计算机内存中存放的相对次序为基础）或者是明显地（借助于节点中指向其它节点之链接）来表示结构关系是方便的。应当在计算机程序的表格之内予以表示的结构信息的数量，取决于对这些节点有待实施的操作。

由于教学法上的原因，我们大部分集注在信息结构与它们的机器表示之间的联系，而不是孤立地来讨论这些问题。然而，为了更深入的理解，从更抽象的观点，即"抽取"可通过对它们本身加以研究的若干层次的思想，来讨论这一课题是有帮助的。已经提出了这类值得注意的若干方法，我们特别推荐下列思想深邃的论文：乔·米利（G.Mealy）《对数据的另一瞥》（Another look at data），《美国信息处理联合会举办的计算机会议会报》（Proc.AFIPS Fall Jt.Comp.Conf.）31（1967），525-534；杰·厄尔利（J.Earley）《关于了解数据结构的问题》（Toward an understanding of data structures），CACM 14（1971）617-627；在奥-约·达尔、埃·怀·迪克斯特拉和查·安·里·霍尔的《结构程序设计》（Structured Programming）一书中（美国科学出版社，1972）83-174；查·安·里·霍尔，《关于数据结构的注记》（Notes on data structuring），罗伯特·威·恩格勒斯（Robert W.Engles），《数据库组织指导》（A tutorial on data-base organization），《自动程序设计年鉴》（Ann.Rev.in Automatic Programming）7（1972），3-63。

这一章中的讨论，并未以完全的一般性来概括信息结构的整个课题；至少有三个重要方面，在这里未曾涉及到：

a）通常有必要或希望对一个表格进行检索，以求出具有某个值的一个节点或一组节点，而这样的一项操作通常对于这个表格的结构有着深远的影响。这一情况将留待第6章详细剖析。

b）我们主要关注在一台计算机内的结构的内部表示，这显然仅仅是事情的一个方面，因为结构还必须表示成外部的输入和输出数据。在简单的情况下，外部结构实质上能以我们所曾经考虑过的同样的技术来处理；但在字符串与更复杂结构之间的转换，也是非常重要的。这些过程要在第9章和第10章中分析。

c）我们主要讨论了结构在一个高速随机存取内存中的表示。当使用缓慢的存储设备（例如，磁盘，磁鼓，磁带）时，我们发现所有的结构问题，都变得更为迫切；有效的算法和对于数据表示的有效方案，更是具有决定性的因素。通常有必要在存储设备的附近区域中试着设置一些彼此链接的"邻近节点"，等等；这些问题都是高度依赖于个别机器的特征，所以难于一般地进行讨论。所希望的是，这一章所处理的较简单的例子，将为读者解决有关不太理想的存储设备出现的更困难的问题作出准备。第5章和第6章详细地讨论了这些问题。

这一章处理的课题中总的含意是什么？也许，我们所能作出的最重要的结论是：我们所遇到的思想不仅限于计算机程序设计，它们更一般地可应用于日常的生活中。一组包含着一些场的节点，其中某些场指向其它的节点，对于各类结构关系来说，看来是一个非常

好的抽象模型。它说明，我们可如何从一些简单的结构来构造出复杂的结构，而且我们已经看到，为处理这一结构的相应的算法，可以一种自然的方式来设计。

因此，提出比我们现时所知道的更多的关于节点链接集合的理论，似乎是适当的。也许来探讨这样一套理论的最明显的途径，就是来定义新的一类处理链接结构的抽象机器或"自动机"。例如，这样一种自动机可以非正式地定义如下：有数 k，l，r，和 s，使得这自动机处理含有 k 个链接场和 r 个信息场的节点；它有 l 个链接寄存器和 s 个信息寄存器，这使它得以控制它所实现的过程。信息场和寄存器可以包含来自信息符号的某个给定集合中的任何符号；每个链接场和链接寄存器或者包含 Λ，或者指向一个节点。机器能够 (i)建立新的节点（置链到该节点的链接到一个寄存器中），(ii) 比较诸信息符号或链接值是否相等，(iii) 在寄存器与节点之间传送信息符号或链接值。仅仅是由链接寄存器指向的节点，才是直接地可存取的。适当地限制机器的行为，将使它等价于若干早先的自动机。

对于这样的设备需要解决的某些最有趣的问题将是，确定它们能多快地解决某些问题，或者它们需要多少节点以解决某些问题（例如，为翻译某种形式语言）。在写完这一章时，关于这方面的若干有趣的结果，已经得到了〔特别是山朱·哈特玛尼斯（J.Hartmanis）和理·埃·斯特恩斯(R.E.Stearns)〕，但仅对具有多重带和读/写头等等的特殊类型的图林机有效；由于图林机的模型相对说来是不现实的，这些结果对于实际的问题一般是不大可取的。这样一点是真实的，即为一个链接自动机所建立的节点的个数 n 趋向无限时，我们必须承认，我们不知道怎样来物理地构造这样一个设备，因为我们为预期该机器的各种操作不管 n 的大小如何，将要花费同样数量的时间；如果象在计算机的内存中那样，通过使用地址来表示链接，则有必要对节点个数来设置一个界限，因为链接场有一个固定的大小。因此当 n 趋向无限时，多带图林机是一个更为现实的模型。而且似乎有理由相信，如上所述的一个链接自动机，会导致比图林机更适当的算法复杂性理论，甚至当考虑到对于充分大的 n 的渐近公式时也是如此，因为该理论更象是与实际的 n 值相关的。

习题

1.〔*M*50〕 剖析正文中所定义的链接自动机的性质。

习 题 答 案

关于习题的说明

1．习题对于专门从事数学的读者，是一个普通的问题。

1．见威·贾·利维格（W. J. LeVeque），《数论集》(Topics in Number Theory) 2（马萨塞·雷丁：爱迪生-韦斯利，1956），第3章（注意：一位阅读过本书最初手稿的人报告说，他已发现了一个确实非常好的证明，但在他的抄本中却没有包括进去）。

1.1 节

1．$t \leftarrow a$，$a \leftarrow b$，$b \leftarrow c$，$c \leftarrow d$，$d \leftarrow t$。

2．第一次之后，变量m，n之值分别是以前的n，r之值；而且$n > r$。

3．**算法 F**（欧几里得算法）。给定两个正整数m和n，求它们的最大公约数。

F1．〔余数m/n〕 以n除m并命m是余数。

F2．〔它为0？〕 如果$m = 0$，则此算法以n为答案而终止。

F3．〔余数n/m〕 以m除n并命n是余数。

F4．〔它为0？〕 如果$n = 0$，则算法以答案m而终止；否则返回步骤F1。 ∎

4．由算法E，n =6099，2166，1767，399，171，57。答案=57。

5．它既不是有限的，也不是确定的，更不是能行的，还可能无输出；对格式而言，在步骤的号数之前没有字母，不出现有概括的短语，而且没有"∎"。

6．以$n = 5$来试验算法E，并计算步骤E1之被执行的次数。当m=1，2，3，4，5，分别得到2，3，4，3，1次；因此平均是$2.6 = T_5$。

7．除了个别情况外，在所有情况下，$n > m$。在这种情况下，算法E的头一次迭代仅仅是互换这两个数；所以$U_m = T_m + 1$。

8．命$A = \{a, b, c\}$，$N = 5$。这个算法将以串$a^{\gcd(m,n)}$结束。

j	θ_j	ϕ_j	b_j	a_j	
0	ab	（空）	1	2	撤消一个a和一个b，或转到2。
1	（空）	c	0	0	把c加到左端，返回到0。
2	a	b	2	3	把所有a变成b。
3	c	a	3	4	把所有c变成a。
4	b	b	0	5	如果还剩有b，则重复。

9．例如，我们可以说C_2表示C_1，如果有一个从I_1到I_2的函数g，有一个从Q_2到Q_1的函数h，把Ω_2写到Ω_1，和一个从Q_2到正整数的函数j，满足下列条件：

a）如果x在I_1中，则C_1从x产生输出y，当且仅当在Ω_2中存在一个y'，使得C_2从$g(x)$产生y'且$h(y') = y$。

b）如果 q 是在 Q_2 中，则 $f_1(h(q))=h(f_2^{j(q)}(q))$，其中 $f_2^{j(q)}$ 意思是函数 f_2 之被迭代 $j(q)$ 次。

例如，命 C_1 如同（2）中那样，并命 C_2 有 $I_2=\{(m,\ n)\}$，$\Omega_2=\{(m,\ n,\ d)\}$，$Q_2=I_2\cup\Omega_2\cup\{(m,\ n,\ a,\ b,\ 1)\}\cup\{(m,\ n,\ a,\ b,r,2)\}\cup\{(m,\ n,\ a,b,r.\ 3)\}\cup\{(m,n.a.\ b.\ r,\ 4)\}$。命 $f_2(m,\ n)=(m,\ n,\ m,\ n,\ 1)$；$f_2(m,\ n,\ d)=(m,\ n,\ d)$；$f_2(m,\ n,\ a,\ b,\ 1)=(m,\ n,\ a,\ b,\ a \bmod b,\ 2)$；$f_2(m,\ n,\ a,\ b,\ r,\ 2)=(m,\ n,\ b)$ 如果 $r=0$，否则 $(m,\ n,\ a,\ b,\ r,\ 3)$；$f_2(m,\ n,\ a,\ b,\ r,\ 3)=(m,\ n,\ b,\ b,\ r,\ 4)$；$f_2(m,\ n,\ a,\ b,\ r,\ 4)=(m,\ n,\ a,\ r,\ 1)$。

现在命 $h(m,\ n)=(m,\ n)=g(m,\ n)$；$h(m,\ n,\ d)=(d)$；$h(m,\ n\ a,\ b,\ 1)=(a,\ b,\ 0,\ 1)$ 如果 $a=m$，$b=n$，否则 $(a,\ b,\ b,\ 1)$；$h(m,\ n,\ a,\ b,\ r,\ 2)=(a,\ b,\ r,\ 2)$；$h(m,\ n,\ a,\ b,\ r,\ 3)=(a,\ b,\ r,\ 3)$；$h(m,\ n,\ a,\ b,\ r,\ 4)=h(f_2(m,\ n,\ a,\ b.\ r,\ 4))$；$j(m,\ n,\ a.\ b,\ r,\ 3)=j(m,\ n,\ a,\ b,\ r,\ 4)=2$，否则 $j(q)=1$。则 C_2 表示 C_1。

注意：试图以一种更简单的方式定义诸事项，例如，命 g 把 Q_1 映象到 Q_2 中，并且主张当 x_0，x_1，…是 C_1 中的一个计算序列时，则 $g(x_0)$，$g(x_1)$，…是 C_2 中计算序列的一个以 $g(x_0)$ 开始的子序列。但这是不适当的，例如，在上边的例子中，C_1 忘记了 m 和 n 原来的值，但 C_2 则不然。

如果 C_3 借助于函数 g，h，j 来表示 C_2，而且如果 C_2 借助于函数 g'，h'，j' 来表示 C_1，则 C_3 借助于函数 g''，h''，j'' 来表示 C_1，其中

$$g''(x)=g'(g(x)),\quad h''(x)=h(h'(x))$$

而且

$$j''(q)=\Sigma_{0\leqslant k<j(h'(q))}j'(q_k)$$

如果 $q_0=q$，$q_{k+1}=f_3^{j'(q_k)}(q_k)$。因此上述关系是传递的。我们可以说 C_2 直接地表示 C_1，如果函数 j 是有界的；这个关系也是传递的。关系“C_2 表示 C_1”产生一个等价关系，其中，两个计算方法显然地是等价的，当且仅当，它们计算其输入之同构的函数；关系“C_2 直接地表示 C_1”生成一个更有趣的等价关系，它也许吻合于“是实质上相同的算法”的直观想法。

1.2.1 小节

1.（a）证明 $P(0)$。（b）证明 $P(0)$，…，$P(n)$ 意味着 $P(n+1)$，当所有的 $n\geqslant 0$。

2. 当 $n=2$ 定理未曾证明；在证明的第二部分中，取 $n=1$；我们假定那里 $a^{-1}=1$。如果这个条件为真（即是，如果 $a=1$），则定理确实成立。

3. 正确的答案是 $1-1/n$。错误出现在对 $n=1$ 的证明中。这时左边的公式，或者可以假定是无意义的，或者可以假定是 0（因为有 $n-1$ 项）。

5. 如果 n 为质数，最简单地它是质数的乘积。否则由定义，n 有因子，所以当 $1<k$，$m<n$ 时，$n=km$。由于 k 和 m 两者小于 n，由归纳法它们可以写作质数的乘积；因此 n 在表示 k 和 m 中是质数的乘积。

6. 在图 3 的记号下，我们证明，$A5$ 意味着 $A6$。很显然，由于 $A5$ 意味着 $(a'-qa)m+$

$(b'-qb)n=(a'm+b'n)-q(am+bn)=c-qd=r$。

7. 解为 $1+2+\cdots+n$；或者，$n(n+1)/2$。

8. (a) 我们将证明 $(n^2-n+1)+(n^2-n+3)+\cdots+(n^2+n-1)$ 等于 n^3。这个和是 $(1+3+\cdots+(n^2+n-1))-(1+3+\cdots+(n^2-n-1))=((n^2+n)/2)^2-((n^2-n)/2)^2=n^3$。我们已经使用了等式 (2)；然而，要求的是一个归纳证明，所以应该采用另一个方法！对于 $n=1$，结果是明显的。设 $n\geqslant 1$；$(n+1)^3-(n-1)=n^3-n+2n$，所以对于 $n+1$ 的前头诸项都更大 $2n$；于是这对于 $n+1$ 的和，是这对于 n 的和加上 $2n+\cdots+2n$ (n 次)+$(n+1)^2+(n-1)-1$；这就等于 $n^3+2n^2+n^2+3n+1=(n+1)^3$。(b) 我们已经看到，对于 $(n+1)^3$ 的头一项比对于 n^3 的最后一项大 2。因此由等式 (2)，$1^3+2^3+\cdots+n^3=$ 由 1 开始的连续的奇数之和 $=$(项数)$^2=(1+2+\cdots+n)^2$。

10. 显然对于 $n=10$；如果 $n\geqslant 10$，则我们有 $2^{n+1}=2\cdot 2^n>\left(1+\frac{1}{10}\right)^3 2^n$，而且由归纳法，这就大于 $(1+1/n)^3n^3=(n+1)^3$。

11. $(-1)^n(n-1)/(4(n+1)^2+1)$。

12. 这一推广的唯一不平凡部分，是计算 E2 中整数 q。这可以通过反复的减法，把问题归结成确定 $u+v\sqrt{2}$ 是正的、负的或是零，而这后一个问题易于解决。

不难证明，每当 $u+v\sqrt{2}=u'+v'\sqrt{2}$ 时，我们必须有 $u=u'$ 和 $v=v'$，因为 $\sqrt{2}$ 是无理数。现在很清楚，1 与 $\sqrt{2}$ 没有公因子，如果我们按着这样的意义来定义因子，$u+v\sqrt{2}$ 整除 a ($u+v\sqrt{2}$) 当且仅当 a 是一个整数。

(注意：然而，如果我们把因子的概念推而广之：就是说 $u+v\sqrt{2}$ 整除 a ($u+v\sqrt{2}$)，当且仅当 a 有 $u'+v'\sqrt{2}$ 的形式，对于整数 u' 和 v' 那么，就有一个方法来推广算法 E，使得它总会终止：如果在步骤 E2 中我们有 $c=u+v\sqrt{2}$ 和 $d=u'+v'\sqrt{2}$，计算 $c/d=c(u'-v'\sqrt{2})/(u'^2-2v'^2)=x+y\sqrt{2}$，其中 x 和 y 是有理数。现在命 $q=u''+v''\sqrt{2}$，其中 u'' 和 v'' 是最接近 x 和 y 的整数；并命 $r=c-qd$。如果 $r=u''+v''\sqrt{2}$，从而得出 $|u''^2-2v''^2|<|u'^2-2v'^2|$，因此计算将终止。关于进一步的介绍，请参看“数论教科书中的”二次欧几里得整环)。

13. 把 “$T<3(n-d)+k$” 加到 $A3$，$A4$，$A5$，$A6$ 的断言中，其中 k 分别取 2，3，3，1 的值。

15. (a) 在 (iii) 中命 $A=S$；每一个非空的良序集合有一个“最小的”元素。

(b) 命 $x\prec y$，如果 $|x|<|y|$，或者如果 $|x|=|y|$ 以及 $x<0<y$。

(c) 否，所有正实数的子集不满足 (iii) (注意：使用“选择公理”，可以给出一个较为复杂的论证，以证明每个集合无论如何总能成为良序的；但是尚没有人能定义一个使实数成为良序的明显的关系)。

(d) 为对于 T_n 证明 (iii)，对 n 用归纳法：设 A 是 T_n 的一个非空子集并考虑 A_1，A 的头一个分量的集合。由于 A_1 是 S 的一个非空子集，而且 S 是良序的，A_1 含有一个最小的元素 x。现在考虑 A_x，即 A 的子集，其中之头一个分量等于 x；A_x 可以考虑作 T_{n-1} 的一个子集，如果它的头一个分量被隐去，所以由归纳法 A_x 包含一个最小的元素 $(x,x_2,\cdots,x_n)$，事实上它就是 A 的最小的元素。

(e) 否，尽管性质 (i) 和 (ii) 都成立。如果 S 至少含有两个不同的元素，$a<b$，

则集合（b），（a，b），（a，a，b），（a，a，a，b），（a，a，a，a，b），…没有最小的元素。另一方面，T可被良序，如果我们定义$(x_1, \cdots, x_n) \prec (y_1, \cdots, y_m)$每当$n < m$，或者$n = m$以及$(x_1, \cdots, x_n) \prec (y_1, \cdots, y_m)$在$T_n$中。

（f）设S通过$\prec$而被良序。如果存在这样一个无穷序列，则由该序列之成员所组成的集合A不满足性质（iii），因为在该序列中没有元素能是最小的。反之，如果$\prec$是一个满足（i）和（ii）但不满足（iii）的关系，设A是S的一个没有最小元素的非空子集。由于A非空，我们可以在A中找到x_1；由于x_1不是A的最小元素，故在A中有x_2使得$x_2 \prec x_1$；由于x_2还不是最小的元素，我们可以找到$x_3 \prec x_2$等等。

（g）命A是使得$P(x)$为假的所有的x的集合。如果A非空，则它含有一个最小的元素x_0。因此$P(y)$对于所有$y \prec x_0$为真。但这意味着$P(x_0)$为真，所以x_0不在A中（矛盾）。因此A必须为空，即$P(x)$总是为真。

1.2.2 小节

1. 不存在；因为如果r是一个正的有理数，则$r/2$是更小的正有理数。

2. 不是，如果出现有无限多个9的话；在这种情况下，根据等式（2），此数的十进展开式为$1 + .24000000\cdots$。

3. $-1/27$，但正文没有定义它。

4. 4。

6. 一个数的十进展开式是唯一的，所以$x = y$，当且仅当，$m = n$而且$d_i = e_i$对于$i = 1, 2, \cdots$。如果$x \neq y$，则人们可以把m与n，d_1与e_1，d_2与e_2，等等，加以比较，而且当出现头一次不等时，较大者即属于x，y中之较大者。

7. 可以对x用归纳法，首先证明对于x为正的定律，而后证明对于x为负的。这里把细节略去。

8. 通过对$n = 0, 1, 2\cdots$进行试验，我们求出当$n^m \leqslant u < (n+1)^m$时$n$的值。加以归纳假定$n$，$d_1$，…，$d_{k-1}$已经确定，$d_k$是使得

$$\left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k}\right)^m \leqslant u < \left(n + \frac{d_1}{10} + \cdots + \frac{d_k}{10^k} + \frac{1}{10^k}\right)^m$$

的数字。

9. $((b^p)^{q/n})^{n'} = (((b^{p/q})^{n/n})^n)^q = ((b^{p/q})^n)^q = ((b^{p/q})^q)^p = b^{pn}$，因此$(b^{p/q})^{n/n} = b^{pn/n}$，这就证明了第二个定律。我们利用第二个来证明头一个：$b^{p/q}b^{n/n} = (b^{1/qn})^{pn}(b^{1/qn})^{qn} = (b^{1/qn})^{pn+qn} = b^{p/q+n/n}$。

10. 如果$\log_{10}2 = p/q$，其中p，q为正，则$2^q = 10^p$。这是荒谬的，因为右边可为5整除，但左边却不能。

11. 无限多！不论给出多少位的x，我们将不知道是$10^x = 1.99999\cdots$还是$2.00000\cdots$。在这一点上，并没有什么神秘或不合理的；如果我们把$.444444\cdots$加到$.555555\cdots$，则在加法中就出现类似的情况。

12. 它们是满足等式（6）的d_1，…，d_8的仅有的值。

13. （a）首先用归纳法证明，如果$y > 0$，则$1 + ny \leqslant (1+y)^n$。然后置$y = x/n$，并

取 n 次根。（b） $x=b-1$， $n=10^k$。

14. 在（4）的第二个等式中置 $x=\log_b c$，并对该等式的两边取对数。

15. 通过把 "$\log_b y$" 移到等式的另一边，并且利用（10）来证明之。

16. $\ln x/\ln 10$。

17. 5；1；1；0；无定义。

18. 否，$\log_8 x=\lg x/\lg 8=\frac{1}{3}\lg x$。

19. 是，因为 $\lg x<(\log_{10}x)/.301<14/.301<47$。

20. 它们互为倒数。

21. $(\ln\ln x-\ln\ln b)/\ln b$。

22. 由附录B中所附的表，$\lg x=1.442695\ln x$；$\log_{10}x=.4342945\ln x$。误差是 $(1.442695-1.4342945)/1.442695=0.582\%$。

23. 取面积 $\ln y$ 的图形，以 x 除它的高度同时以 x 乘它的长度。这个变形保持它的面积，并使得它全等于左边的那块——当从 $\ln xy$ 撤消 $\ln x$ 时；因为在对于 $\ln xy$ 的图式中，于点 $x+xt$ 处的高度为 $1/(x+xt)=(1/(1+t))/x$。

24. 在出现 10 的任何地方代之以 2。

27. 对 k 用归纳法，证明

$$x^{n^k}(1-\eta)^{2^k-1}\leqslant 10^{n^k(n-b_1/2+\cdots+b_k/2^k)}x_k'\leqslant x^{2^k}(1+\epsilon)^{2^{k+1}-1}$$

并取对数。

28. **E1**. 置 $y\leftarrow 1$，$k\leftarrow 0$。

E2. 如果 $x=0$，则停止。

E3. 如果 $x<\log_b(1+2^{-k})$，则转到E5。

E4. 置 $x\leftarrow x-\log_b(1+2^{-k})$，$y\leftarrow y+2^{-k}y$，则转到E2。

E5. k 增1，转到E2。 ∎

在第 j 次执行步骤E1时，当我们置 $x\leftarrow x-\log_b(1+2^{-k})+\eta_j$ 和 $y\leftarrow y(1+2^{-k})(1+\epsilon_j)$ 对于某个小的误差 η_j 和 ϵ_j，就引起了计算误差。当算法终止时，我们有 $y=b^x\Pi(1+\epsilon_j)b^{\sum\eta_j}$。进一步的分析依赖于 b 和计算机的字长。注意在这种情况和习题26的两种情况下，如果底数为 e，则有可能稍微地改进误差估计，因为对于 k 的大多数值，表的记录 $\ln(1\pm 2^{-k})$ 可以高精度地给出来：它等于 $\pm 2^{-k}-\frac{1}{2}2^{-2k}\pm\frac{1}{3}2^{-3k}-\cdots$。

注意：对于三角函数，可以给出类似的算法；见约·E·梅吉特(J. E. Meggitt)，《IBM研究和发展杂志》(IBM J. Res. and Dev.) 6 (1962)，210~226；7(1963)，237~245。

29. e；3；1。

1.2.3 小节

1. $a_1+a_2+a_3$。

2. $-\frac{1}{1}+\frac{1}{3}+\frac{1}{5}+-\frac{1}{7}+\frac{1}{9}+\frac{1}{11}$；$\frac{1}{9}+\frac{1}{3}+\frac{1}{1}+\frac{1}{3}+\frac{1}{9}$。

3. 违背了对于 $P(j)$ 的规则：在第一个当中，$n^2=3$ 没有 n 出现；而在第二个当中，$n^2=4$ 出现有两个 n。

4. $(a_{11})+(a_{21}+a_{22})+(a_{31}+a_{32}+a_{33})$

$$=(a_{11}+a_{21}+a_{31})+(a_{22}+a_{32})+(a_{33})$$

5. 仅仅需要使用规则 $a\Sigma_{R(i)}x_i=\Sigma_{R(i)}(ax_i)$：

$$\left(\sum_{R(i)}a_i\right)\left(\sum_{S(j)}b_j\right)=\sum_{R(i)}a_i\left(\sum_{S(j)}b_j\right)=\sum_{R(i)}\left(\sum_{S(j)}a_ib_j\right)$$

7. 用等式（3）；交换两个极限，而且 a_0 与 a_e 之间的项必须从一个极限移到另一个。

8. 命 $a_{(i+1)i}=+1$，以及 $a_{i(i-1)}=-1$，对于所有 $i\geqslant 0$，且所有其它的 a_{ij} 为 0；命 $R(i)=S(i)=$"$i\geqslant 0$"。左边为 -1，右边为 $+1$。

9,10. 否，规则（d）的应用假定 $n\geqslant 0$（对于 $n=-1$ 结果是正确的，但微商不对）。

11. $(n+1)a$。

12. $\frac{7}{6}(1-1/7^{n+1})$。

13. $m(n-m+1)+\frac{1}{2}(n-m)(n-m+1)$；或者，$\frac{1}{2}(n(n+1)-m(m-1))$。

14. $\left(m(n-m+1)+\frac{1}{2}(n-m)(n-m+1)\right)\left(r(s-r+1)+\frac{1}{2}(s-r)(s-r+1)\right)$。

15,16. 关键步骤：

$$\sum_{0\leqslant j\leqslant n}jx^j=x\sum_{1\leqslant j\leqslant n}jx^{j-1}=x\sum_{0\leqslant j\leqslant n-1}(j-1)x^j$$

$$=x\sum_{0\leqslant j\leqslant n}jx^j-nx^{n+1}+x\sum_{0\leqslant j\leqslant n-1}x^j$$

17. S 中的元数。

18. $S'(j)=$"$1\leqslant j<n$"。$R'(i,j)=$"n 是 i 的一个倍数而且 $i>j$"。

19. a_n-a_{n-1}。

20. $(b-1)\Sigma_{3\leqslant k\leqslant n}(n-k)b^k+n+1=\Sigma_{0\leqslant k\leqslant n}b^k$；这个公式从（14）和习题 16 的结果得出。

21. 与（3）类似，加上这样一个约定，即存在一个整数 j_0，使得

$$\prod_{\substack{R(j)\\|j|>j_0}}a_j\neq 0$$

22. 对于（5），（7），仅仅把 Σ 换成 Π。而且，我们还有

$$\prod_{R(i)}(b_ic_i)=\Big(\prod_{R(i)}b_i\Big)\Big(\prod_{R(i)}c_i\Big);\quad\Big(\prod_{R(j)}a_j\Big)\Big(\prod_{S(j)}a_j\Big)$$

$$=\Big(\prod_{\substack{R(j)\\ 或S(j)}}a_j\Big)\Big(\prod_{\substack{R(j)\\ 且S(j)}}a_j\Big)$$

23. $0+x=x$ 和 $1\cdot x=x$。这就使得许多运算和等式更简单，例如，规则（d）及其在上题中之类似者。

25. 头一步和最后一步，没错。第二步，同时使用 i 于两种不同的目的。第三步，大概应该是 $\Sigma_{j<i\leqslant n}u_i$。

26. 关键步骤：

$$\prod_{0\leqslant i\leqslant n}\Big(\prod_{0\leqslant j\leqslant n}a_ia_j\Big)=\prod_{0\leqslant i\leqslant n}\Big(a_i^{n+1}\prod_{0\leqslant j\leqslant n}a_j\Big)$$

$$=\Big(\prod_{0\leqslant i\leqslant n}a_i^{n-1}\Big)\Big(\prod_{0\leqslant i\leqslant n}\Big(\prod_{0\leqslant j\leqslant n}a_j\Big)\Big)$$

$$=\Big(\prod_{0\leqslant i\leqslant n}a_i\Big)^{2n+2}$$

答案是

$$\Big(\prod_{0\leqslant i\leqslant n}a_i\Big)^{n+2}$$

28. $(n-1)/2n$。

29. a）$\Sigma_{0\leqslant k\leqslant j\leqslant i\leqslant n}a_ia_ja_k$。

b）命 $S_r=\Sigma_{0\leqslant i\leqslant n}a_i^r$。解答：$\frac{1}{3}S_3+\frac{1}{2}S_1S_2+\frac{1}{6}S_1^3$。

这个问题的一般解答，随着下标的个数变得更大时，可在 1.2.9 小节中找到，即等式(34)。

31. $n\sum_{1\leqslant j\leqslant n}a_jb_j-\Big(\sum_{1\leqslant j\leqslant n}a_j\Big)\Big(\sum_{1\leqslant j\leqslant n}b_j\Big)$。

33. 这可以通过对 n 用归纳法来证明，如果我们把该公式改写成

$$\frac{1}{x_n-x_{n-1}}\left(\sum_{1\leqslant j\leqslant n}\frac{x_j^r(x_j-x_{n-1})}{\prod_{1\leqslant k\leqslant n,k\neq j}(x_j-x_k)}-\sum_{1\leqslant j\leqslant n}\frac{x_j^r(x_j-x_n)}{\prod_{1\leqslant k\leqslant n,k\neq j}(x_j-x_k)}\right)$$

现在这些和的每一个都有了原来的和的形式，除了是对 $n-1$ 个元素求和之外，而且当 $0\leqslant r\leqslant n-1$ 时，按归纳法就很好地得出这些值。当 $r=n$，考虑恒等式

$$0=\sum_{1\leqslant j\leqslant n}\frac{\prod_{1\leqslant k<n}(x_j-x_k)}{\prod_{1\leqslant k<n,k\neq j}(x_j-x_k)}=\sum_{1\leqslant j\leqslant n}\frac{x_j^n-(x_1+\cdots+x_n)x_j^{n-1}+P(x_j)}{\prod_{1\leqslant k<n,k\neq j}(x_j-x_k)}$$

其中 $P(x_j)$ 是一个 $n-2$ 次的多项式；由对于 $r=0$，1，…，$n-1$ 之解答，我们就得到所求的答案。

注意：这里的公式，乃是关于“除法差分”之数值方法的基础。下面的另一个证明方

法，用到复变理论，它稍逊初等但却尤为漂亮：按残数定理，所给的和之值为

$$\frac{1}{2\pi i}\int_{|z|=R}\frac{z^r dz}{(z-x_1)\cdots(z-x_n)}$$

其中 $R>|x_1|$，…，$|x_n|$。被积函数的劳伦特展开式在 $|z|=R$ 上一致收敛；它是

$$z^{r-n}\left(\frac{1}{1-x_1/z}\right)\cdots\left(\frac{1}{1-x_n/z}\right)$$

$$=z^{r-n}+(x_1+\cdots+x_n)z^{r-n-1}+(x_1^2+x_1x_2+\cdots)z^{r-n-2}+\cdots$$

逐项进行积分，除了 z^{-1} 的系数之外，一切全都为 0。这个方法，给了我们对于任意整数 $r\geqslant 0$ 的一般公式：

$$\sum_{\substack{j_1+\cdots+j_n=r-n+1\\ j_1,\ \cdots,\ j_n\geqslant 0}} x_1^{j_1}\cdots x_n^{j_n}$$

34. 如果读者已经非常认真地来解决这个问题，则无须得到答案，大概它的目的就已经达到了。把分子看作 x 的多项式而不是看作 k 的多项式，这种倾向几乎占绝对优势。毫无疑问地，更易于证明颇为一般的结果

$$\sum_{1\leqslant k\leqslant n}\frac{\Pi_{1\leqslant r<n-1}(y_k-z_r)}{\Pi_{1\leqslant r\leqslant n,r\neq k}(y_k-y_r)}=1$$

这是 $2n-1$ 个变量的一个恒等式！

35. 如果 $R(j)$ 根本不成立，则这个值将是 $-\infty$。所述的规则（a）的类似性，是以恒等式 $a+\max(b,c)=\max(a+b,a+c)$ 为基础的。类似地，如果所有的 a_i，b_j 都非负，则我们有

$$\sup_{R(i)}a_i\ \sup_{S(j)}b_j=\sup_{R(i)}\ \sup_{S(j)}a_ib_j$$

规则（b），（c）不变；对于规则（d），我们得到更简单的形式

$$\sup\left(\sup_{R(j)}a_j,\ \sup_{S(j)}a_j\right)=\sup_{R(j)\text{或}S(j)}a_j$$

36. 从第 2，…，n 列减去第 1 列。把第 2，…，n 行加到第 1 行。结果是一个三角形的行列式。

37. 从第 2，…，n 列减去第 1 列。然后对于 $k=n$，$n-1$，…，2（按此顺序），从行 k 减去 x_1 乘行 $k-1$。我们现在从第 1 列提取因子 x_1，而且从列 $k=2$，…，n 提取因子 x_k-x_2，得到 $x_1(x_2-x_1)\cdots(x_n-x_1)$ 乘一个 $n-1$ 阶的范特蒙德行列式，这个过程按归纳法而如此继续。

另一个证明，利用“更高等的”数学：这行列式是一个总次数为 $1+2+\cdots+n$ 的关于变量 x_1，…，x_n 的多项式。如果 $x_i=0$ 或者如果 $x_i=x_j$（$i<j$），则它就为 0。而且 $x_1^1x_2^2\cdots x_n^n$ 的系数为 +1。这些事实就表征了它的值。一般地，如果一个矩阵的两行由于 $x_i=x_j$ 而变为相等者，则它们的差总是可为 x_i-x_j 所整除，而这种观察往往加快着行列式的计算（罗·威·弗洛伊德）。

38. 从第 2，…，n 列减去第 1 列，并从诸行和诸列提取因子 $(x_1+y_1)^{-1}\cdots(x_n+y_1)^{-1}(y_1-y_2)\cdots(y_1-y_n)$。现在从第 2，…，$n$ 行减去第 1 行，并提出因子 $(x_1-x_2)\cdots$

$(x_1-x_n)(x_1+y_2)^{-1}\cdots(x_1+y_n)^{-1}$，我们留下了 $n-1$ 阶的柯西行列式。

39. 命 I =单位矩阵，J =全 1 矩阵。由于 $J^2=nJ$，我们立即得出 $(xI+yJ)((x+ny)I-yJ)=x(x+ny)I$。

40. $$\sum_{1\leqslant t\leqslant n} b_{it}x_j^t=x_i\prod_{\substack{1\leqslant k\leqslant n\\ k\neq i}}(x_k-x_j)\Big/x_t\prod_{\substack{1\leqslant k\leqslant n\\ k\neq i}}(x_k-x_t)=\delta_{ij}。$$

41. 通过观察逆矩阵与余因子的关系，可立即得出。这里给出一个直接的证明也是有趣的。

$$\sum_{1\leqslant t\leqslant n}\frac{1}{x_i+y_t}b_{tj}=\sum_{1\leqslant t\leqslant n}\frac{\Pi_{k\neq t}(x_j+y_k-x)\Pi_{k\neq i}(x_k+y_t)}{\Pi_{k\neq j}(x_j-x_k)\Pi_{k\neq t}(y_t-y_k)}$$

当 $x=0$ 时。这是 x 的至多 $n-1$ 次的一个多项式。如果我们置 $x=x_j+y_s$，$1\leqslant s\leqslant n$，则这些项都为 0，除非 $s=t$ 时，所以这个多项式的值为

$$\prod_{k\neq i}(-x_k-y_s)\Big/\prod_{k\neq j}(x_j-x_k)=\prod_{k\neq i}(x_j-x_k-x)\Big/\prod_{k\neq j}(x_j-x_k)$$

由于这些次数至多为 $n-1$ 的多项式，在 n 个不同的 x 上一致，所以它们对于 $x=0$ 也一致，因此

$$\sum_{1\leqslant t\leqslant n}\frac{1}{x_i+y_t}b_{tj}=\prod_{k\neq i}(x_j-x_k)\Big/\prod_{k\neq j}(x_j-x_k)=\delta_{ij}$$

42. $n/(x+ny)$。

43. $1-\Pi_{1<k\leqslant n}(1-1/x_k)$。这是容易验证的，如果有 $x_t=1$ 的话，因为任何有一行或一列全为 1 的矩阵之逆，必然有这样的元素，即其和为 1。如果没有 x_i 等于 1，则象在习题 44 中那样把行 i 的元素加起来，并得到 $\Pi_{k\neq i}(x_k-1)/x_i\Pi_{k\neq i}(x_k-x_i)$。我们现在能使用习题 33，对 i 来求这个和，用 $r=0$（以 (x_i-1) 乘分子和分母）。

44. 在应用习题 33 之后，我们求出

$$c=\sum_{1\leqslant i\leqslant n}b_{ij}=\prod_{1\leqslant k\leqslant n}(x_k+y_t)\Big/\prod_{\substack{1\leqslant k\leqslant n\\ k\neq j}}(x_j-x_k)$$

而且

$$\sum_{1\leqslant j\leqslant n}c_j=\sum_{1\leqslant j\leqslant n}\frac{(x_j^n+(y_1+\cdots+y_n)x_j^{n-1}+\cdots)}{\Pi_{1\leqslant k\leqslant n,k\neq j}(x_j-x_k)}$$
$$=(x_1+x_2+\cdots+x_n)+(y_1+y_2+\cdots+y_n)$$

45. 命 $x_i=i$，$y_j=j-1$。由习题 44，逆的元素之和为 $(1+2+\cdots+n)+((n-1)+(n-2)+\cdots+0)=n^2$。由习题 38，这逆的元素是

$$b_{ij}=\frac{(-1)^{i+j}(i+n-1)!(j+n-1)!}{(i+j-1)(i-1)!^2(j-1)!^2(n-i)!(n-j)!}$$

这个量可以写成若干包含二项式系数的形式，例如

$$\frac{(-1)^{i+j}ij}{i+j-1}\binom{-i}{n}\binom{n}{i}\binom{-j}{n}\binom{n}{j}$$

$$=(-1)^{i+j}\binom{i+j-2}{i-1}\binom{i+n-1}{i-1}\binom{j+n-1}{n-i}\binom{n}{j}$$

从这后边的公式，我们看到，b_{ij}不仅是一个整数，而且它还可被 i，j，n，$i+j-1$，$i+n-1$、以及 $j+n-1$ 所整除。也许对于 b_{ij}的最好的公式是

$$(i+j-1)\binom{i+j-2}{i-1}^2\binom{-(i+j)}{n-j}\binom{-(i+j)}{n-j}$$

如果我们不认识希尔伯特矩阵乃是柯西矩阵的特殊情况，那么，对于这个问题的解决将是极为困难的；更一般的问题要比它的特殊情况，容易解决得多！这往往是明智的——把一个问题推广成为它的“归纳法闭包”，即是成为最小的推广，使得在一个企图用数学归纳法作出的证明中，所产生的全部子问题都属于这同一类中。在现在情况下，我们看到，柯西矩阵的余因子是柯西矩阵，但希尔伯特矩阵的余因子则不是希尔伯特矩阵〔关于进一步的介绍，见约·托德（J. Todd），J. Res. Nat. Bur. Stand. 65(1961)，19–22〕。

46. 对于任何整数 k_1，k_2，…，k_m，命 $\epsilon(k_1,\cdots,k_m)=\operatorname{sign}(\prod_{1\leqslant i<j\leqslant m}(k_j-k_i))$。如果 $(q_1,\cdots,q_m)$ 等于 $(k_1,\cdots,k_m)$，除了 k_i 与 k_j 已被互相交换的事实之外，那么，我们就有 $\epsilon(q_1,\cdots,q_m)=-\epsilon(k_1,\cdots,k_m)$。因此，我们有等式 $\det(B_{k_1\cdots k_m})=\epsilon(k_1,\cdots,k_m)\det(B_{j_1\cdots j_m})$，如果 $j_1\leqslant\cdots\leqslant j_m$ 是以递降顺序重新排列的数 $k_1,\cdots,k_m$。现在由行列式的定义，

$$\det(AB)=\sum_{\substack{1\leqslant q_1,\cdots,\\ q_m\leqslant m}}\epsilon(q_1,\cdots,q_m)\left(\sum_{1\leqslant k\leqslant n}a_{1k}b_{kq_1}\right)\cdots\left(\sum_{1\leqslant k\leqslant n}a_{mk}b_{kq_m}\right)$$

$$=\sum_{\substack{1\leqslant k_1,\cdots,\\ k_m\leqslant n}}a_{1k_1}\cdots a_{mk_m}\sum_{\substack{1\leqslant q_1,\cdots,\\ q_m\leqslant m}}\epsilon(q_1,\cdots,q_m)b_{k_1q_1}\cdots b_{k_mq_m}$$

$$=\sum_{\substack{1\leqslant k_1,\cdots,\\ k_m\leqslant n}}a_{1k_1}\cdots a_{mk_m}\det(B_{k_1\cdots k_m})$$

$$=\sum_{\substack{1\leqslant k_1,\cdots,\\ k_m\leqslant n}}\epsilon(k_1,\cdots,k_m)a_{1k_1}\cdots a_{mk_m}\det(B_{j_1\cdots j_m})$$

〔如上所述，其中诸 j 与诸 k 有关〕

$$=\sum_{\substack{1\leqslant j_1\leqslant\cdots\\ \leqslant j_m\leqslant n}}\det(A_{j_1\cdots j_m})\det(B_{j_1\cdots j_m})$$

最后，如果两个 j 相等，则 $\det(A_{j_1\cdots j_m})=0$。

1.2.4 小节

1. 3，−2，−1，0，5。

2. $\lfloor x\rfloor$。

3. 由定义，$\lfloor x\rfloor$ 是小于或等于 x 的最大整数；因此，$\lfloor x\rfloor$ 为一整数，$\lfloor x\rfloor\leqslant x$，而且 $\lfloor x\rfloor+1>x$。这后一性质，加上这样一个事实，即当 m，n 为整数时，$n<m$ 当且仅当 $n\leqslant m-1$，就容易导出命题（a）和（b）的一个证明。类似的论证可证明（c）

和（d）。最后，（e）和（f）只不过是这一道题的前几部分之组合。

1. $x-1<\lfloor x\rfloor\leqslant x$；所以$-x+1>-\lfloor x\rfloor\geqslant -x$；从而得出结果。

5. $\left\lfloor x-\frac{1}{2}\right\rfloor$。（$-x$舍入）的值将等同于$-$（$x$舍入），除了当$x \bmod 1=\frac{1}{2}$之外，其时负的值傍近0舍入，正的值则远离0舍入。

6.（a）为真；$\lfloor\sqrt{\lfloor x\rfloor}\rfloor=n$当且仅当$n^2\leqslant x<(n+1)^2$当且仅当$n^2\leqslant\lfloor x\rfloor<n(+1)^2$当且仅当$\lfloor\sqrt{\lfloor x\rfloor}\rfloor=n$。类似地，（b）为真。但（c）为假，例如，对于$x=1.1$。

7. 应用习题3和等式（4）。对于上限，不等式应是$\geqslant$，而后等式成立当且仅当或x或y是一个整数或者$x \bmod 1+y \bmod 1\geqslant 1$。

8. 1，2，5，-100。

9. -1，0，-2。

10. 0.1，0.01，-0.09。

11. $x=y$。

12. 所有的整数。

13. $+1$，-1。

14. 8。

15. 以z来乘等式（1）的两边；如果$y=0$，则这一结果也容易验证。

17. 作为一个例子，考虑定律A的乘法部分：我们有$a=b+qm$，$x=y+rm$对于某整数q，r；所以$ax=by+(br+yq+qrm)m$。

18. 我们有$a-b=kr$对于某个整数k，而且还有$kr\equiv 0$(modulo s)。因此，由定律B，$k\equiv 0$(modulo s)，所以$a-b=qsr$对于某个整数q。

20. 以a'来乘同余式的两边。

21. 按以前证明的习题，至少有一个这样的表示。如果有两个表示，$n=p_1\cdots p_k=q_1\cdots q_m$，则我们有$q_1\cdots q_m\equiv 0$(modulo p_1)；所以如果没有q等于p_1，则由定律B我们就可以全部消去诸q并得到$1\equiv 0$(modulo p_1)。后者是不可能的，因为p_1不等于1（这乃是我们不允许把1当作一个质数的主要原因）。所以某个q_j等于p_1，而且$n/p_1=p_2\cdots p_k=q_1\cdots q_{j-1}q_{j+1}\cdots q_m$。或者$n$是质数，这时结果显然为真，或者按归纳法，$n/p_1$的两个因子分解相同。

22. 如果$a=cd$，$m=nd$，则$an\equiv 0$但$n\not\equiv 0$(modulo m)如果$d>1$，$m\neq 0$。

24. 对于加法和减法定律A总是对的；定律C总是对的。

26. 如果b不是p的一个倍数，则b^2-1是，所以它的因子之一必须是。

27. 一个数与p^e互质，当且仅当它不是p的一个倍数。所以我们计算的那些不是p的倍数，并得到$\Phi(p^e)=p^e-p^{e-1}$。

28. 如果a、b与m互质，从而$(ab \bmod m)$也与m互质。因为整除后者和m的任何质数必然也整除a或b。现在简单地命$x_1,\cdots,x_{\varphi(m)}$是与$m$互质的数，并且观察$ax_1 \bmod m,\cdots,ax_{\varphi(m)} \bmod m$是在某一次序下的同样数等等。

29. 我们证明（b）：如果r，s互质而且如果k^2整除rs，则对于某个质数p，p^2整除rs，从而p整除（比如说）r且不能整除s；所以p^2整除r。我们看到，$f(rs)=0$，当且

仅当 $f(r)=0$ 或 $f(s)=0$。

30. 设 r、s 互质。这个想法是要证明，$\varphi(rs)$ 个与 rs 互质的数，恰恰就是 $\varphi(r)\varphi(s)$ 个不同的数 $(sx_i+ry_j)\bmod(rs)$，其中 x_1、…、$x_{\varphi(r)}$ 和 y_1、…、$y_{\varphi(s)}$ 是对于 r 和 s 的对应的值。

然后，我们求得 $\varphi(10^6)=\varphi(2^6)\varphi(5^6)=(2^6-2^5)(5^6-5^5)=400000$；$\varphi(p_1^{e_1}\cdots p_r^{e_r})=(p_1^{e_1}-p_1^{e_1-1})\cdots(p_r^{e_r}-p_r^{e_r-1})$；$\varphi(n)=n\prod_{p\backslash n,p\in\mathbb{P}}(1-1/p)$。另一个证明在习题 1.3.3-27。

31. rs 的因子可以唯一地写成 cd 的形式，其中 c 整除 r 且 d 整除 s。类似地，如果 $f(n)\geqslant 0$，我们发现函数 $\max_{d\backslash n}f(d)$ 是乘性的。参见习题 1.2.3-35。

33. 或者是 $n+m$，或者是 $n-m+1$ 为偶数，所以左边括弧内的数量之一是一个整数，从而习题 7 中的等式成立。

34. b 必须是一个 $\geqslant 2$ 的整数（置 $x=b$）。充分性的证明就象在习题 6 中那样。同一个条件对于 $\lceil\log_b x\rceil=\lceil\log_b\lceil x\rceil\rceil$ 是充分必要的。

更一般地，我们有下列由罗·麦克尔里斯(R. McEliece) 给出的良好的推断：设 f 是一个在区间 A 上定义的连续的和严格递增的函数，并假定 x 在 A 中意味着 $\lfloor x\rfloor$ 和 $\lceil x\rceil$ 两者都在 A 中。于是，关系 $\lfloor f(x)\rfloor=\lfloor f(\lfloor x\rfloor)\rfloor$ 对于 A 中所有的 x 成立，当且仅当，关系 $\lceil f(x)\rceil=\lceil f(\lceil x\rceil)\rceil$ 对于 A 中所有的 x 成立，当且仅当，我们有下列条件："$f(x)$ 是一个整数，意味着 x 是一个整数"。这个条件显然是必要的，因为如果 $f(x)$ 是一个整数，而且它等于 $\lfloor f(\lfloor x\rfloor)\rfloor$ 或 $\lceil f(\lceil x\rceil)\rceil$，则 x 必然等于 $\lfloor x\rfloor$ 或 $\lceil x\rceil$。反之，如果说 $\lfloor f(\lfloor x\rfloor)\rfloor<\lfloor f(x)\rfloor$，则由于连续性必然有某个 y 使得 $\lfloor x\rfloor<y\leqslant x$，对于它 $f(y)$ 为一整数，而 y 不可能是整数。

35. $$\frac{x+m}{n}-1=\frac{x+m}{n}-\frac{1}{n}-\frac{n-1}{n}<\frac{\lfloor x\rfloor+m}{n}-\frac{n-1}{n}\leqslant\left\lfloor\frac{\lfloor x\rfloor+m}{n}\right\rfloor\leqslant\frac{x+m}{n}$$

应用习题 3。用习题 4 给出一个类似的关于上限函数之结果。两个恒等式遂作为习题 34 中的麦克尔里斯定理之一特殊情况而得出。

36. 首先假定 $n=2t$。

$$\sum_{1\leqslant k\leqslant n}\lfloor k/2\rfloor=\sum_{1\leqslant k\leqslant n}\lfloor(n+1-k)/2\rfloor$$

因此

$$\sum_{1\leqslant k\leqslant n}\lfloor k/2\rfloor=\frac{1}{2}\left(\sum_{1\leqslant k\leqslant n}(\lfloor k/2\rfloor+\lfloor(n+1-k)/2\rfloor)\right)$$

$$=\frac{1}{2}\sum_{1\leqslant k\leqslant n}\lfloor(2t+1)/2\rfloor=t^2=n^2/4$$

（参照习题 33.）而且，如果 $n=2t+1$，我们有 $t^2+\lfloor n/2\rfloor=t^2+t=n^2/4-\frac{1}{4}$。对于第二个和，类似地得到 $\lceil n(n+2)/4\rceil$。

37. $\sum_{0\leqslant k<n}\left\{\frac{mk-x}{n}\right\}=\frac{m(n-1)}{2}+x$

命 $\{y\}$ 表示 $y \bmod 1$；必须减去

$$S=\sum_{0\leqslant k<n}\left\{\frac{mk-x}{n}\right\}$$

S 由 d 个相同的和之复写组成. 因为如果 $t=n/d$，

$$\left\{\frac{mk+x}{n}\right\}=\left\{\frac{m(k-t)+x}{n}\right\}$$

命 $u=m/d$；则

$$\sum_{0\leqslant k<t}\left\{\frac{mk+x}{n}\right\}=\sum_{0\leqslant k<t}\left\{\frac{x}{n}+\frac{uk}{t}\right\}$$

而且因为 t 与 u 互质，这个和可重新排列成

$$\left\{\frac{x \bmod d}{n}\right\}+\left\{\frac{x \bmod d}{n}+\frac{1}{t}\right\}+\cdots+\left\{\frac{x \bmod d}{n}+\frac{t-1}{t}\right\}.$$

最后，由于 $(x \bmod d)/n<1/t$，可消去这个和中的括弧，而且有

$$S=d\left(\frac{t(x \bmod d)}{n}+\frac{t-1}{2}\right)$$

应用习题 4，我们得到

$$\sum_{0\leqslant k<n}\left\lceil\frac{mk+x}{n}\right\rceil=\frac{(m+1)(n-1)}{2}-\frac{d-1}{2}+d\lceil x/d\rceil$$

这个公式将对 m 和 n 变成对称的，如果把它扩充到范围 $0\leqslant k\leqslant n$（以下述方法阐明对称性：即把被加项作为 k 的函数画出图形，然后关于直线 $y=x$ 进行反射）。

38. 对于 $x=0$ 等式成立，而且当 x 增加一个形如 k/n 的数时，两边都增加 1。

39. (f)的证明：考虑更一般的恒等式 $\prod_{0\leqslant k<n}2\sin\pi(x+k/n)=2\sin\pi nx$，它可以被证明如下：由于 $2\sin\theta=(e^{i\theta}-e^{-i\theta})/i=(1-e^{-2i\theta})e^{i\theta-i\pi/2}$，这恒等式乃是两个公式

$$\prod_{0\leqslant k<n}(1-e^{-2\pi i(x+k/n)})=1-e^{-2\pi inx}\quad\text{和}\quad\prod_{0\leqslant k<n}e^{\pi i(x-(1/2)+(k/n))}=e^{\pi i(nx-1/2)}$$

的一个推论。后者为真，因为函数 $x-\frac{1}{2}$ 是重迭的；前者为真，因为在多项式的因式分解 $z^n-\alpha^n=(z-\alpha)(z-\omega\alpha)\cdots(z-\omega^{n-1}\alpha)$，$\omega=e^{2\pi i/n}$ 之中，我们可以置 $z=1$。

40.（由尼·戈·德·布鲁因说明）如果 f 是重迭的，则 $f(nx+1)-f(nx)=f(x+1)-f(x)$ 对于所有 $n>0$。因此，如果 f 是连续的，则 $f(x+1)-f(x)=c$ 对于所有的 x，而且 $g(x)=f(x)-c\lfloor x\rfloor$ 是重迭的和周期的。现在

$$\int_0^1 e^{2\pi inx}g(x)dx=n^{-1}\int_0^1 e^{2\pi iy}g(y)dy$$

展开成傅里叶级数看出，$g(x)=\left(x-\frac{1}{2}\right)a$ 对于 $0<x<1$。由此推出 $f(x)=\left(x-\frac{1}{2}\right)a$。一般地，这个论证说明了，任何重迭的局部黎曼可积函数几乎处处都有形式

$\left(x-\frac{1}{2}\right)a+b\max(\lfloor x\rfloor,0)+c\min(\lfloor x\rfloor,0)$。

41. 我们要 $a_n=k$，当

$$\frac{k(k-1)}{2}<n\leqslant\frac{k(k+1)}{2}$$

因为 n 是一个整数，这就等价于

$$\frac{k(k-1)}{2}+\frac{1}{8}<n<\frac{k(k+1)}{2}+\frac{1}{8}$$

即 $k-\frac{1}{2}<\sqrt{2n}<k+\frac{1}{2}$。因此 $a_n=\left\lfloor\sqrt{2n}+\frac{1}{2}\right\rfloor$，即最接近于 $\sqrt{2n}$ 的整数。另外的正确答案为 $\lceil(-1+\sqrt{1+8n})/2\rceil$ 和 $\lfloor(1+\sqrt{8n-7})/2\rfloor$。

42. (a) 参照习题 1.2.7-10。(b) 给出的和为 $n\lfloor\log_b n\rfloor-S$，其中

$$S=\sum_{\substack{1\leqslant k<n\\ k+1\text{ 为 }b\text{ 的幂}}}k=\sum_{1\leqslant t\leqslant\log_b n}(b^t-1)=(b^{\lfloor\log_b n\rfloor+1}-b)/(b-1)-\lfloor\log_b n\rfloor。$$

43. $\lfloor\sqrt{n}\rfloor\left(n-\frac{(2\lfloor\sqrt{n}\rfloor+5)(\lfloor\sqrt{n}\rfloor-1)}{6}\right)$。

44. 当 n 为负时，这个和为 $n+1$。

45. $\lfloor mj/n\rfloor=r$ 当且仅当

$$\left\lceil\frac{rn}{m}\right\rceil\leqslant j<\left\lceil\frac{(r+1)n}{m}\right\rceil$$

而且我们求出，这给定的和因而是

$$\sum_{0\leqslant r<m}f(r)\left(\left\lceil\frac{(r+1)n}{m}\right\rceil-\left\lceil\frac{rn}{m}\right\rceil\right)$$

通过重新排列这后边的和，把具有特殊的 $\lceil rn/m\rceil$ 之值的那些项归在一起，而得出所述结果。第二个公式通过代入

$$f(x)=\binom{x+1}{k}$$

直接得到。

46. $\sum_{0\leqslant j<\alpha n}f(\lfloor mj/n\rfloor)=\sum_{0\leqslant r<\alpha m}\lceil rn/m\rceil(f(r-1)-f(r))+\lceil\alpha n\rceil f(\lceil\alpha m\rceil-1)$。

47. (a) 数 $2,4\cdots,p-1$ 是偶剩余 (modulo p)；因为 $2kq=p\lfloor 2kq/p\rfloor-(2kq)\bmod p$，数 $(-1)^{\lfloor 2kq/p\rfloor}(2kq)\bmod p$ 将是一偶剩余或一偶剩余减 p，而且每一个偶剩余显然地恰好出现一次。因此 $(-1)^{\sigma}q^{(p-1)/2}2\cdot4\cdots(p-1)\equiv2\cdot4\cdots(p-1)$。(b) 命 $q=2$。如果 $p=4n+1$，则 $\sigma=n$；如果 $p=4n+3$，$\sigma=n+1$。因此 $\left(\frac{2}{p}\right)=1,-1,-1,1$，分别地依照 $p\bmod 8=1,3,5,7$。(c) 对于 $k<p/4$，

$$\lfloor(p-1-2k)q/p\rfloor=q-\lceil(2k+1)q/p\rceil=q-1-\lfloor(2k+1)q/p\rfloor$$
$$\equiv\lfloor(2k+1)q/p\rfloor(\text{modulo }2)$$●

● 原著有误，已予改正。——译者

因此，我们可以以 $\lfloor q/p \rfloor$，$\lfloor 3q/p \rfloor$，… 来代替最后诸项 $\lfloor (p-1)q/p \rfloor$，$\lfloor (p-3) \times q/p \rfloor$，等等。(d) $\sum_{0<k<p/2}\lfloor kq/p \rfloor + \sum_{0<r<q/2}\lceil rp/q \rceil = \lceil p/2 \rceil(\lceil q/2 \rceil - 1) = (p+1) \times (q-1)/4$。而且 $\sum_{0<r<q/2}\lceil rp/q \rceil = \sum_{0<r<q/2}\lfloor rp/q \rfloor + (q-1)/2$。这个证明的思想回溯到戈·艾森斯坦 (G. Eisenstein)，Journal f. d. reine und angewandte Math. 28 (1844)，246～248；艾森斯坦在同一卷中，还给出了这个以及另外的互反定律的若干其它的证明。

48. (a) 当 $n<0$ 时，显然地这不总是成立的；当 $n>0$ 时这容易验证。(b) $\lfloor (n+2-\lfloor n/25 \rfloor)/3 \rfloor = \lceil (n-\lfloor n/25 \rfloor)/3 \rceil = \lceil (n+\lceil -n/25 \rceil)/3 \rceil = \lceil \lceil 24n/25 \rceil/3 \rceil =$ (参照习题 35) $\lceil 8n/25 \rceil = \lfloor (8n+24)/25 \rfloor$。

1.2.5 小节

1. 52!。为了好奇吧，这个数算出是 806 58175 17094 38785 71660 63685 64037 66975 28950 54408 83277 82400 00000 00000。(!)

2. $p_{nk} = p_{n(k-1)}(n-k+1)$。在放置了头 $n-1$ 个对象之后，仅剩有一个位置了，对于最后一个对象仅有一种选择。但这并不意味着最后一个对象在所有排列中都是相同的!

3. 53124, 35124, 31524, 31254, 31245; 42351, 41352, 41253, 31254, 31245。

4. 有 2568 个数字。头一个数字是 4 (因为 $\log_{10} 4 = 2\log_{10} 2 \approx .602$)。最低有效位数字为 0，而且事实上由等式 (8)，低位的 249 个数字全都是零! 霍·斯·尤勒 (H. S. Uhler) 使用一台台式计算器以及在若干年期间的极度的耐心，算出了 1000! 的精确值，而这个值出现在 Scripta Mathematica 21(1955), pp. 266-267。它以 402 38726 00770… 开始。

5. $(39902)(97/96) \approx 416 + 39902 = 40318$。

6. $2^{18} \cdot 3^8 \cdot 5^4 \cdot 7^2 \cdot 11 \cdot 13 \cdot 17 \cdot 19$。

8. 它是 $m^n m!/((n+m)!/n!) = n!$ 乘 $m^n/(m+1)\cdots(m+n)$。后一数量由于 $m/(m+k) \to 1$ 而趋于 1。

9. $\sqrt{\pi}$ 和 $-2\sqrt{\pi}$ (使用了习题10)。

10. 对，除非当 $x=0$ 或一负整数时。因为我们有

$$\Gamma(x+1) = x \lim_{m\to\infty} \frac{m^x m!}{x(x+1)\cdots(x+m)}\left(\frac{m}{x+m+1}\right)$$

11、12.
$$\begin{aligned}\mu &= (a_k p^{k-1} + \cdots + a_1) + (a_k p^{k-2} + \cdots + a_0) + \cdots + a_k \\ &= a_k(p^{k-1} + \cdots + p + 1) + \cdots + a_1 \\ &= (a_k(p^k - 1) + \cdots + a_0(p^0 - 1))/(p-1) \\ &= (n - a_k - \cdots - a_1 - a_0)/(p-1)\end{aligned}$$

13. 对于每一个 n，$1 \leqslant n < p$，如同在习题1.2.4-19中那样确定 n'。由定律1.2.4B，我们有唯一的这样一个 n'；而且 $(n')' = n$。因此我们能把诸数两两配对成两个一组，假定 $n' \neq n$。如果 $n' = n$，则我们有 $n^2 \equiv 1 \pmod{p}$；因此，如同在习题1.2.4-26中那样，$n=1$ 或 $n=p-1$。所以 $(p-1)! \equiv 1 \cdot 1 \cdot \cdots \cdot (-1)$，因为 $+1$ 和 $p-1$ 是

仅有的未配对的元素。

14. 不为 p 之倍数的数 $1, 2, \cdots, n$ 中，有 $\lfloor n/p \rfloor$ 个由 $p-1$ 个相继的元素组成的完备集合，由威尔逊定理，每一个都有同余于 (-1)(modulo p) 的乘积。而且还剩下 a_0，这部分同余于 $a_0!$(modulo p)，所以，不为 p 之倍数的那些因子的贡献是 $(-1)^{\lfloor n/p \rfloor} a_0!$。而为 p 之倍数的那些因子的贡献，则等同于 $\lfloor n/p \rfloor!$ 中的贡献；因此可以重复这个论证以获得所求的公式。

15. $(n!)^3$。有 $n!$ 项。每一项均有来自每行每列各一元素，所以每一项有值 $(n!)^2$。

16. 这些项不趋于 0，因为系数趋于 $1/e$。

17. 按等式 (15) 把伽马函数表达成极限。

18. $$\prod_{n\geqslant 1} \frac{n}{(n-1/2)} \frac{n}{(n+1/2)} = \frac{\Gamma\left(\frac{1}{2}\right)\Gamma\left(\frac{3}{2}\right)}{\Gamma(1)\Gamma(1)} = 2\Gamma\left(\frac{3}{2}\right)^2 。$$

19. (a) 变量的替换 $t = mt$。(b) 分部积分。(c) 归纳。

20. 〔为完全计，我们证明所述的不等式。从易于验证的不等式 $1+x \leqslant e^x$ 开始。置 $x = \pm t/n$ 并取 n 次幂以得到 $(1 \pm t/n)^n \leqslant e^{\pm t}$。因此由习题 1.2.1-9，

$$e^{-t} \geqslant (1-t/n)^n = e^{-t}(1-t/n)^n e^t \geqslant e^{-t}(1-t/n)^n(1+t/n)^n$$
$$= e^{-t}(1-t^2/n^2)^n \geqslant e^{-t}(1-t^2/n)〕。$$

现在，这给定的积分减 $\Gamma_m(x)$ 是

$$\int_m^\infty e^{-t}t^{x-1}dt + \int_0^m \left(e^{-t} - \left(1-\frac{t}{m}\right)^m\right)t^{x-1}dt$$

当 $m\to\infty$ 时，其中之头一项趋于 0，因为对于充分大的 t，$t^{x-1} < e^{t/2}$；而第二项小于

$$\frac{1}{m}\int_0^m t^{x+1}e^{-t}dt < \frac{1}{m}\int_0^\infty t^{x+1}e^{-t}dt \to 0$$

21. 如果 $c(n, j, k_1, k_2, \cdots)$ 表示相应的系数，则我们由微分求得

$$c(n+1, j, k_1, \cdots) = c(n, j-1, k_1-1, k_2, \cdots)$$
$$+(k_1+1)\,c(n, j, k_1+1, k_2-1, k_3, \cdots)$$
$$+(k_2+1)\,c(n, j, k_1, k_2+1, k_3-1, k_4, \cdots)$$
$$+\cdots$$

等式 $k_1+k_2+\cdots = j$ 和 $k_1+2k_2+\cdots = n$ 在这个归纳关系中被保持。我们能容易地由出现于 $c(n+1, j, k_1\cdots)$ 的等式右边中之每一项，分解出因子 $n!/k_1!(1!)^{k_1}k_2!(2!)^{k_2}\cdots$，而且我们保持有 $k_1+2k_2+3k_3+\cdots = n+1$（在证明中，假定有无限多个 k 是方便的，尽管很明显 $k_{n+1} = k_{n+2} = \cdots = 0$）。关于这些系数的一份表，见 1.2.9 小节末尾的参考文献。

刚才给出的解答，利用了标准的求和技术，但它并没有给出为什么这个公式有此形式之令人满意的揭示，也没有说明首先是如何发现它的。让我们使用一种组合的论证，来考查这个问题。为了方便起见，写 $w_j = D_u^j w$，$u_k = D_x^k u$。于是 $D_x(w_j) = w_{j+1}u_1$ 和 $D_x(u_k) = u_{k+1}$。由这两个规则以及求一乘积之导数的规则，我们求得

$$D_x^1 w = w_1u_1$$
$$D_x^2 w = (w_2u_1u_1 + w_1u_2)$$
$$D_x^3 w = ((w_3u_1u_1u_1 + w_2u_2u_1 + w_2u_1u_2) + (w_2u_1u_2 + w_1u_3)) \text{ 等等}$$

类似地，我们可以建立对应的集合分划的表示如下：

$$\mathscr{O}^1=\{1\}$$

$$\mathscr{O}^2=(\{2\}\{1\}+\{2,1\})$$

$$\mathscr{O}^3=((\{3\}\{2\}\{1\}+\{3,2\}\{1\}+\{2\}\{3,1\})+(\{3\}\{2,1\}+\{3,2,1\}))$$ 等等

形式上，如果 $a_1a_2\cdots a_j$ 是集合 $\{1,2,\cdots,n-1\}$ 的一个分划，则定义

$$\mathscr{O}a_1a_2\cdots a_j=(\{n\}a_1a_2\cdots a_j+(a_1\cup\{n\})a_2\cdots a_j+a_1(a_2\cup\{n\})\cdots a_j+\cdots$$
$$+a_1a_2\cdots(a_j\cup\{n\})$$

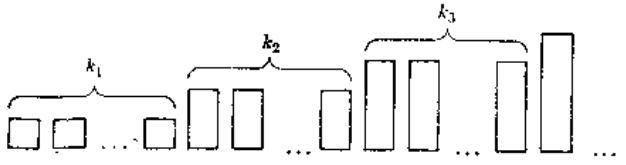
这个规则确切地平行于规则

$$D_x(w_ju_{r_1}u_{r_2}\cdots u_{r_j})=w_{j+1}u_1u_{r_1}u_{r_2}\cdots u_{r_j}+w_ju_{r_1+1}u_{r_2}\cdots u_{r_j}$$
$$+w_ju_{r_1}u_{r_2+1}\cdots u_{r_j}+\cdots+w_ju_{r_1}u_{r_2}\cdots u_{r_j+1}$$

如果我们让项 $w_ju_{r_1}u_{r_2}\cdots u_{r_j}$ 对应于一个具有 r_i 个元素在 a_i 中的分划 $a_1a_2\cdots a_j$，$1\leqslant i\leqslant j$。所以有一个从 $\mathscr{O}^n$ 到 D_x^nw 上的自然映象，进而容易看出，$\mathscr{O}^n$ 恰巧包括集合 $\{1,2,\cdots,n\}$ 的每个分划各一次（参照习题1.2.6-64）。

由这些观察，我们发现，如果我们归并 D_x^nw 中的同类项，则我们就得到一个项 $c(k_1,k_2,\cdots)w_ju_1^{k_1}u_2^{k_2}\cdots$ 的和，其中 $j=k_1+k_2+\cdots$ 和 $n=k_1+2k_2+\cdots$，而 $c(k_1,k_2,\cdots)$ 是把 $\{1,2,\cdots,n\}$ 分划成 j 个部分，恰使 k_t 个部分有着 t 个元素的分划数。

剩下就是来计算这分划数。考虑容量 t 的 k_t 个盒子的一个阵列：



放置 n 个不同的元素于这些盒子内的放法种数，是多项式系数

$$\binom{n}{11\cdots122\cdots233\cdots34\cdots}=n!/1!^{k_1}2!^{k_2}3!^{k_3}\cdots$$

为了得到 $c(k_1,k_2,k_3,\cdots)$，应以 $k_1!k_2!k_3!\cdots$ 来除这个数，因为在每个 k_t 的类中，诸盒子是不能互相区别的，而且可以以 $k_t!$ 种方法来进行排列而不影响这集合的分划（这个解答是由罗·麦克尔里斯给出的）。

伐敌奔的公式已经有若干种方式之推广。至于哪一种也许是最为宽广的推广，以及对其它有关著作的参考文献，请见欧·约·古德，《数理统计年刊》(Annals of Mathematical Statistics) 32(1961)，540～541。

22. 关于 $\lim_{n\to\infty}(n+x)!/n!n^x=1$ 的假设，对于整数 x 是正确的；例如，如果 x 为正，则这个量是 $(1+1/n)(1+2/n)\cdots(1+x/n)$，它确是趋于 1。如果我们还假定 $x!=x(x-1)!$，则这个假设就把我们引向直接结论

$$1=\lim_{n\to\infty}\frac{(n+x)!}{n!n^x}=x!\lim_{n\to\infty}\frac{(x+1)\cdots(x+n)}{n!n^x}$$

它就等价于课文中所给出的定义。

1.2.6 小节

1. n，因为每个组合忽略一项。

2. 1。

3. $\binom{52}{13}$。实际的数是 635013559600。

4. $2^4 5^2 7^2 17 \cdot 23 \cdot 41 \cdot 43 \cdot 47$。

5. $(10+1)^4 = 10000 + 4(1000) + 6(100) + 4(10) + 1$。

6. $r=-3$: 1 -3 6 -10 15 -21 28 -36 …
   $r=-2$: 1 -2 3 -4 5 -6 7 -8 …
   $r=-1$: 1 -1 1 -1 1 -1 1 -1 …

7. $\lfloor n/2 \rfloor$；或者，$\lceil n/2 \rceil$。由（3）显而易见，对于较小的值，二项式系数是严格地递增的，而后递减成 0。

8. 每行中的非 0 项，从左到右来读与从右到左来读是相同的。

9. 如果 n 为正或 0，则值为 1；如果 n 为负，则值为 0。

10. （a），（b），（f）直接由（e）得出，而（c），（d）由（a），（b）和等式（9）得出。因此，只须证明（e）就足够了。把 $\binom{n}{k}$ 考虑作一个由等式（3）给出的分数，具有分子和分母中的诸因子。分母中头 $k \bmod p$ 个因子没有 p，而在分子和分母中这些项显然同余于

$$\binom{n \bmod p}{k \bmod p}$$

的相应的诸项。它们相差 p 的倍数（当处理 p 之非倍数时，我们可以在分子和分母两者中都进行 modulo p 的工作，因为，如果 $a \equiv c$ 和 $b \equiv d$，而且 a/b，c/d 是整数，则 $a/b \equiv c/d$）。还剩下 $k - k \bmod p$ 个因子，它们各自落入 $\lfloor k/p \rfloor$ 个由 p 个连续值组成的类中。每一类恰恰包含一个 p 之倍数；一类中的其它 $(p-1)$ 个因子，(modulo p) 同余于 $(p-1)!$，所以它们在分子和分母中相消。剩下来研究分子和分母中的 $\lfloor k/p \rfloor$ 个 p 之倍数；我们以 p 来除这些中的每一个，而且留下二项式系数

$$\binom{\lfloor (n - k \bmod p)/p \rfloor}{\lfloor k/p \rfloor}$$

如果 $k \bmod p \leqslant n \bmod p$，则这就等于

$$\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor}$$

是所欲求；但如果 $k \bmod p > n \bmod p$，则这等于

$$\binom{\lfloor n/p \rfloor - 1}{\lfloor k/p \rfloor}$$

然而，另一个因子

$$\binom{n \bmod p}{k \bmod p}$$

则为 0。所以这个公式是一般地成立的〔也见内·雅·法因(N. J. Fine), AMM 54(1947), 589-592〕。

11. 如果

$$a = a_r p^r + \cdots + a_0$$
$$b = b_r p^r + \cdots + b_0$$
$$a + b = c_r p^r + \cdots + c_0$$

则这值（按照习题 1.2.5-12 和等式（5））是

$$(a_0 + \cdots + a_r + b_0 + \cdots + b_r - c_0 - \cdots - c_r)/(p-1)$$

进位使 c_j 减 p 并使 c_{j+1} 增 1，在这个公式中引起 +1 的净变化。

12. 由上两题中之任一题，n 必须比 2 的幂小 1。更一般地，$\binom{n}{k}$ 决不能为质数 p 所整除，$0 \leqslant k \leqslant n$，当且仅当，$n = ap^m - 1$，$1 \leqslant a < p$，$m \geqslant 0$。

14. $$24\binom{n+1}{5} + 36\binom{n+1}{4} + 14\binom{n+1}{3} + \binom{n+1}{2}$$
$$= \frac{n^2}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30} = \frac{n(n+1)\left(n+\frac{1}{2}\right)(3n^2+3n-1)}{15}$$

15. 用归纳法和等式（9）。

17. 我们可以假定 r，s 是正整数。而且对于所有 x，

$$\sum_n \binom{r+s}{n} x^n = (1+x)^{r+s} = \sum_n \binom{r}{n} x^n \sum_k \binom{s}{k} x^k$$
$$= \sum_n \sum_k \binom{r}{n-k} x^{n-k} \binom{s}{k} x^k$$
$$= \sum_n \left(\sum_k \binom{r}{n-k} \binom{s}{k} \right) x^n$$

所以 x^n 的系数必须相等。

21. 左边是一个 n 次的多项式，右边是一个 $m+n+1$ 次的多项式。我们已经在 $n+1$ 个点上一致。但并不足以证明它们相等（尽管当 $m=0$ 时，表明两边都是某个多项的倍式；而事实上在 $m=0$ 的情况下，我们发现，这个等式是一个关于 s 的恒等式，因为它是等式（11））。

22. 假定 $n > 0$。第 k 项是

$$\frac{1}{n!}\binom{n}{k} \prod_{0<j<k} (r - tk - j) \prod_{0<j<n-k} (n - 1 - r + tk - j)$$
$$= \frac{(-1)^{n-1}}{n!}\binom{n}{k} \prod_{0<j<k} (-r + tk + j) \prod_{k \leqslant j < n} (-r + tk + j)$$

而且这两个乘积给出 k 的一个 $n-1$ 次多项式，所以由等式(35)，对 k 之求和为零。

24. 证明通过对 n 用归纳法来进行。如果 $n \leqslant 0$，则恒等式是明显的。如果 $n > 0$，则我们通过对整数 $m \geqslant 0$，证明它对于 $(r, n-r+ni+m, i, n)$ 成立，利用上边的两

道习题和它对于 $n-1$ 的正确性。这就对于无限多的 s 确立了 (r, s, t, n) 的恒等性，而且作为 s 的一个多项式，它对所有 s 成立。

25. 利用比率测试和对于大的 k 值的直接了当的估计，我们可以证明收敛性（或者利用复变理论，我们知道这函数在 $x=1$ 的邻域中是解析的）。我们有

$$\begin{aligned}1&=\sum_{k,\,j}(-1)^j\binom{k}{j}\binom{r-jt}{k}\frac{r}{r-jt}w^k=\sum_j(-1)^j\frac{r}{r-jt}\sum_k\binom{k}{j}\binom{r-jt}{k}w^k\\&=\sum_j\frac{r(-1)^j}{r-jt}\sum_k\binom{r-jt}{j}\binom{r-jt-j}{k-j}w^k\\&=\sum_j(-1)^jA_j(r,t)(1+w)^{r-jt-j}w^j\end{aligned}$$

现在命 $x=1/(1+w)$，$z=-w/(1+w)^{1+t}$。这个证明是亨·沃·古尔德给出的〔*AMM* 63(1956)，84～91〕。也可看迈·斯卡尔斯基（M. Skalsky）给出的简单的但稍逊初等的推导，*AMM* 69 (1962)，404-405，和习题 2.3.4.4-33 中的更一般的公式。

26. 我们将通过恒等式

$$\sum_j(-1)^j\binom{k}{j}\binom{r-jt}{k}=t^k$$

开始，并如上进行。另一种方法是把我们的公式对于 z 求微商；我们得到

$$\sum_k kA_k(r,t)z^k=z\frac{d(x^r)}{dz}=\frac{(x^{t+1}-x^t)rx^r}{((t+1)x^{t+1}-tx^t)x}$$

由此我们能得到

$$\sum_k\left(1-\frac{t}{r}k\right)A_k(r,t)z^k$$

的值。

27. 对于等式 (26)，以 x^s 的级数来乘 $x^{t+1}/((t+1)x-t)$ 的级数，得到 $x^{t+s+1}/((t+1)x-t)$ 的级数，其中 z 的系数可以等于由 $x^{(t+s)+1}/((t+1)x-t)$ 产生的级数之系数。

28. 以 $f(r,s,t,n)$ 来表示左边，我们求得

$$\binom{r+s}{n}+t\cdot f(r-t-1,\ s+t,\ t,\ n-1)=f(r,s,t,n)$$

通过考虑恒等式

$$\sum_k\binom{r+tk}{k}\binom{s-tk}{n-k}\frac{r}{r+tk}+\sum_k\binom{r+tk}{k}\binom{s-tk}{n-k}\frac{tk}{r+tk}$$
$$=f(r,s,t,n)。$$

29. $(-1)^k\binom{n}{k}\Big/n!=(-1)^k/k!\,(n-k)!=(-1)^n\Big/\prod_{\substack{0\leqslant j\leqslant n\\ j\neq k}}(k-j)$。

30. 应用 (7) 和 (19) 来求得

$$\sum_{k\geqslant 0}\binom{-m-2k-1}{n-m-k}\binom{2k+1}{k+1}\frac{(-1)^{n-m}}{2k+1}$$

$$=\sum_{k\geqslant 1}\binom{-m-2k+1}{n-m-k+1}\binom{2k-1}{k}\frac{(-1)^{n-m}}{2k-1}$$

现在，如果我们加上 $k=0$ 的项，则我们即能以

$$r=-1,\quad s=m-2n-1,\quad t=-2,\quad n=n-m+1$$

来应用等式（26）。所以我们得到结果

$$\binom{-m}{n-m+1}(-1)^{n-m-1}+\binom{-m+1}{n-m+1}(-1)^{n-m}=\binom{-m}{n-m}(-1)^{n-m}=\binom{n-1}{n-m}$$

这个结果与我们以前的公式相同，当 n 为正时；但当 $n=0$，我们所得到的答案是正确的，而 $\binom{n-1}{m-1}$ 则不然。我们的推导还有进一步的收益，因为答案 $\binom{n-1}{n-m}$ 对于 $n\geqslant 0$ 和所有整数 m 都有效。

31.〔这个和式首先由 J. F. 帕夫（J. F. Pfaff）以一种相近的形式得到，Nova acta acad scient. Petr. 11（1797），38～57〕我们有

$$\sum_k\sum_j\binom{m-r+s}{k}\binom{n+r-s}{n-k}\binom{r}{m+n-j}\binom{k}{j}$$

$$=\sum_j\sum_k\binom{m-r+s}{j}\binom{n+r-s}{n-k}\binom{r}{m+n-j}\binom{m-r+s-j}{k-j}$$

$$=\sum_j\binom{m-r+s}{j}\binom{r}{m-n-j}\binom{m+n-j}{n-j}$$

把

$$\binom{m-n-j}{n-j}\quad\text{变成为}\quad\binom{m-n-j}{m}$$

并且再次应用（20），我们得到

$$\sum_j\binom{m-r+s}{j}\binom{r}{m}\binom{r-m}{n-j}=\binom{r}{m}\binom{s}{n}$$

32. 在（40）中以 $-x$ 代替 x。

33，34. 我们有

$$x^{\overline{n}}=n!\binom{x+n-1}{n}$$

这个等式因此可以变换成

$$\binom{x+y+n-1}{n}=\sum_k\binom{x+(1-z)k}{k}$$

$$\cdot\binom{y-1+nz+(n-k)(1-z)}{n-k}\frac{x}{x+(1-z)k}$$

此乃（26）之一情形。

35. 例如，我们证明头一个公式：

$$\sum_k (-1)^{n-k}\left((n-1)\left[\begin{matrix}n-1\\k\end{matrix}\right]+\left[\begin{matrix}n-1\\k-1\end{matrix}\right]\right)x^k$$

$$= (n-1)(n-1)!\binom{x}{n-1}+x(n-1)!\binom{x}{n-1}=n!\binom{x}{n}$$

36. 通过二项式公式（假定 n 是非负整数）我们分别得到 2^n 和 δ_{n0}。

37. 当 $n>0$ 时，是 2^{n-1}（奇数项和偶数项相消，所以每个等于总和的一半）。

38. 命 $\omega=e^{2\pi i/m}$。则

$$\sum_{0\leqslant j<m}(1+\omega^j)^n\omega^{-jk}=\sum_t\binom{n}{t}\sum_{0\leqslant j<m}\omega^{j(t-k)}$$

现在

$$\sum_{0\leqslant j<m}\omega^{rj}=m\delta_{(r \bmod m)0}$$

（这是一个几何级数的和），所以右边的和为 $m\sum_{t \bmod m=k}\binom{n}{t}$。左边原来的和为

$$\sum_{0\leqslant j<m}(\omega^{-j/2}+\omega^{j/2})^n\omega^{j(n/2-k)}=\sum_{0\leqslant j<m}\left(2\cos\frac{j\pi}{m}\right)^n\omega^{j(n/2-k)}$$

因为已知这个量是实数，我们可以取实数部分并得到所述公式。

39. $n!$；0，如果 $n\geqslant 2$，$+1$，在其它两种情况下（在第二个三角形中的行和，并不那么简单；我们将发现（习题64），这就给出把 n 个元素的集合分划为不相交集合方式的种数，即是，等价关系的个数）。

40.（c）的证明：通过分部积分，

$$B(x+1,y)=\frac{t^x(1-t)^y}{y}\Big|_0^1+\frac{x}{y}\int_0^1 t^{x-1}(1-t)^y dt$$

现在使用（b）。

41. 不论 m 跑遍整数值与否（由单调性）。当 $m\to\infty$，$m^xB(x,m+1)\to\Gamma(x)$，因此，$(m+y)^xB(x,m+y+1)\to\Gamma(x)$。而且 $(m/(m+y))^x\to 1$。

42. $1/kB(k,r-k+1)$。

43. $\int_0^1 dt/t^{1/2}(1-t)^{1/2}=2\int_0^1 du/(1-u^2)^{1/2}=2\arcsin u\Big|_0^1=\pi$。

45. 对于充分大的 r 值，我们有

$$\frac{1}{k\Gamma(k)}\sqrt{\frac{r}{r-k}}\,\frac{1}{e^k}\,\frac{(1-k/r)^k}{(1-k/r)^r}\to\frac{1}{\Gamma(k+1)}$$

46. $\sqrt{\frac{1}{2\pi}\left(\frac{1}{x}+\frac{1}{y}\right)}\left(1+\frac{y}{x}\right)^x\left(1+\frac{x}{y}\right)^y\cdot\binom{2n}{n}\approx 4^n/\sqrt{\pi n}$。

48. 这可以用归纳法证明之，利用事实

$$0=\sum_k\binom{n}{k}(-1)^k=\sum_k\binom{n}{k}\frac{(-1)^k k}{k+x}+\sum_k\binom{n}{k}\frac{(-1)^k x}{k+x}$$

或者，我们有

$$B(x, n-1)=\int_0^1 t^{x-1}(1-t)^n dt=\sum_k \binom{n}{k}(-1)^k \int_0^1 t^{x+k-1}dt$$

（事实上，当级数收敛时，对于非整数 n，所述的和也等于 $B(x, n+1)$）。

49. $\binom{r}{m}=\sum_k \binom{r}{k}\binom{-r}{m-2k}(-1)^{m-k}$，$m$ 为整数。

50. 第 k 个被加项是 $\binom{n}{k}(-1)^{n-k}(x-kz)^{n-1}x$。应用等式 (35)。

51. 右边是

$$\sum_k \binom{n}{n-k} x(x-kz)^{k-1} \sum_r \binom{n-k}{r}(x+y)^r(-x+kz)^{n-k-r}$$

$$=\sum_r \binom{n}{r}(x+y)^r \sum_k \binom{n-r}{n-r-k} x(x-kz)^{k-1}(-x+kz)^{n-k-r}$$

$$=\sum_r \binom{n}{r}(x+y)^r 0^{n-r}=(x+y)^n$$

同样的手段也可用来证明托勒甲和（习题34）

$$(x+y)^n=\sum_k \binom{n}{k} x(x-kz-1)^{\underline{k-1}}(y+kz)^{\underline{n-k}}$$

其中 $x^{\underline{n}}=x(x-1)\cdots(x-n+1)$。

对于阿贝尔公式的另一个出色的证明，来自于如下的事实，即是，很容易把它变换成在习题 2.3.4.4-29 中所导出的更为对称的恒等式：

$$\sum_k \binom{n}{k} x(x+kz)^{k-1} y(y+(n-k)z)^{n-k-1}=(x+y)(x+y+nz)^{n-1}。$$

阿·胡尔维兹尤进一步推广了阿贝尔定理〔Acta Mathematica 26 (1902), 199～203〕如下：

$$\sum x(x+\epsilon_1 z_1+\cdots+\epsilon_n z_n)^{\epsilon_1+\cdots+\epsilon_n-1}(y-\epsilon_1 z_1-\cdots-\epsilon_n z_n)^{n-\epsilon_1-\cdots-\epsilon_n}=(x+y)^n$$

其中 $\sum$ 是对 2^n 项求和（$\epsilon_1, \cdots, \epsilon_n$ 只能分别选择 0 或 1）。这是 $x, y, z_1, \cdots, z_n$ 的一个恒等式，而阿贝尔公式乃是 $z_1=z_2=\cdots=z_n$ 之特殊情况。胡尔维兹公式由习题 2.3.4.4-30 之结果推出。

52. $\sum_{k\geq 0}(k+1)^{-2}=\pi^2/6-1$〔马·洛·约·豪道斯（M. L. J. Hautus）发现，这个和对于所有复变量 x、y、z、r，每当 $z\neq 0$，就绝对收敛，因为对于充分大的 k，诸项总有 $1/k^2$ 的阶。这个收敛性在有界区域中是一致的；所以我们可以逐项来对这个级数求微商。如果 $f(x, y, r)$ 是这个和当 $z=1$ 时的值，则我们求出 $(\partial/\partial y) f(x, y, r)=rf(x, y, r-1)$ 和 $(\partial/\partial x) f(x, y, r)=rf(x-1, y+1, r-1)$。这些公式与 $f(x, y, r)=(x+y)^r$ 是一致的；但实际上后一等式似乎很少成立，如果有过的话，除非这个和是有限的。而且，关于 z 的导数几乎总是非 0 的〕。

54. 在“棋盘式样”中插入负号具体如下

$$\begin{pmatrix} 1 & -0 & 0 & -0 \\ -1 & 1 & -0 & 0 \\ 1 & -2 & 1 & -0 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

这就等价于以$(-1)^{i+j}$来乘a_{ij}。由等式(34)，这结果是所求的逆。

55. 象在上题中那样，在一个三角形矩阵内插入负号，得到另一个的逆（等式(34)）。

56. 012 013 023 123 014 024 124 034 134 234 015 025 125 035 135 235 045 145 245 345 016。通过固定c，a和b跑遍c件事物每次取2件的组合；通过固定c，b，a跑遍b件事物每次取1件的组合。类似地，我们可以以$0\leqslant a<b<c<d$来表达所有的数$n=\binom{a}{1}+\binom{b}{2}+\binom{c}{3}+\binom{d}{4}$；这个序列以0123 0124 0134 0234 1234 0125 0135 0235 …开始。

58. 用归纳法，由于$\binom{n}{k}_q=\binom{n-1}{k}_q+\binom{n-1}{k-1}_q q^{n-k}=\binom{n-1}{k}_q q^k+\binom{n-1}{k-1}_q$。由此推出(21)的"$q$推广"是

$$\sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(r-k)(n-k)}=\sum_k \binom{r}{k}_q \binom{s}{n-k}_q q^{(s-n+k)k}=\binom{r+s}{n}_q$$

这些系数在许多不同的应用中出现；参照5.1.2. 6.3节和作者在《组合理论杂志》(J. Combinatorial Theory)(A) 10(1971)，178～180中的注记。对于进一步的介绍，请看威·诺·贝利的经典小书《广义超几何级数》Generalized Hypergeometric Series(剑桥大学出版社，1935)，第8章。

59. $(n+1)\binom{n}{k}$。

60. $\binom{n+k-1}{k}$。这个公式容易记住，因为它是

$$\frac{n(n+1)\cdots(n+k-1)}{k(k-1)\cdots 1}$$

即是，象等式(2)，只是分子中的数往上升而不是往下降。欲证这个公式，有一个乖巧的方法，就是请注意，我们要来计算对于关系$1\leqslant a_1\leqslant a_2\leqslant\cdots\leqslant a_k\leqslant n$的整数解$(a_1,\cdots,a_k)$的个数。但这关系就等同于$0<a_1<a_2+1<\cdots<a_k+k-1<n+k$；而且对于$0<b_1<b_2<\cdots<b_k<n+k$的解的个数，是由集合$\{1,2,\cdots,n+k-1\}$中选取$k$件不同的事物之选择个数（这一技巧是由海·弗·谢勒克(H. F. Scherk)给出的《数学杂志》(Journal für Math.) 3(1828)，97；说来也怪，威·A·福斯特曼(W. A. Förstemann)在同一杂志上于vol.13(1835)，237中也给出，他说："人们几乎都认为这必然是早就知道了，但是我那儿也没有找到它，尽管我已经查考了这一方面的许多工作"！）通过使用生成函数，可以容易地推导出这个公式（参照习题1.2.9-16）。

61. 如果a_{nm}是所求的量，则按(42)，(43)，我们有$a_{nm}=na_{(n-1)m}+(-1)^n\delta_{nm}$。因此对于$n<m$，答案为0，而对于$n\geqslant m$，答案为$(-1)^m n!/m!$。通过反演(52)，也容易得到这同一个公式。

62. 利用习题31的恒等式，以$(m,n,r,s,k)\leftarrow(m+k,l-k,m+n,n-l,j)$：通过重新安排阶乘的符号，

$$\sum_k(-1)^k\binom{l+m}{l+k}\binom{m+n}{m+k}\binom{n+l}{n+k}$$

$$=\sum_{j,k}(-1)^k\binom{l+m}{l+k}\binom{l+k}{j}\binom{m-k}{l-k-j}\binom{m+n+j}{m+l}$$

$$=\sum_{j,k}(-1)^k\binom{2l-2j}{l-j+k}\frac{(m+n+j)!}{(2l-2j)!j!(m-l+j)!(n+j-l)!}$$

对k之求和现在为0，除非$j=l$。

这个恒等式的$l=m=n$之情形，是由阿·卡·狄克逊（A.C.Dixon）发表的，见《数学信使》（Messenger of Math.）20（1891），79～80。见珀·阿·麦克马汉（P.A.MacMahon）的论文《纯粹数学与应用数学季刊》（Quarterly Journal of Pure and Applied Math.）33（1902），274～288，以及约翰·杜格尔（John Dougall），《爱丁堡数学协会会报》（Proc.Edinburgh Math.Society）25(1906)，114～132。对应的q项式的恒等式是

$$\sum_k\binom{m-r+s}{k}_q\binom{n+r-s}{n-k}_q\binom{r+k}{m+n}_q q^{(m-r+s-k)(n-k)}=\binom{r}{m}_q\binom{s}{n}_q$$

$$\sum_k(-1)^k\binom{l+m}{l+k}_q\binom{m+n}{m+k}_q\binom{n+l}{n+k}_q q^{(3k^2-k)/2}=\frac{(l+m+n)!_q}{l!_q m!_q n!_q}$$

其中$n!_q=\prod_{1\le k\le n}(1+q+\cdots+q^{k-1})$。

64. 设$f(n,m)$是把$\{1,2,\cdots,n\}$分成m个部分的划分数。显然，$f(1,m)=\delta_{1m}$。如果$n>1$，则这种划分有两种：（a）元素n单独地形成该划分的一个集合；共有$f(n-1,m-1)$种方式来构造象这样的分划；（b）元素n与另外的元素一起出现，共有m种方式来把n插入到$\{1,2,\cdots,n-1\}$的任何m-划分中，因此有$mf(n-1,m)$种方式来构造象这样的划分。于是，我们结论$f(n,m)=f(n-1,m-1)+mf(n-1,m)$，而且由归纳法$f(n,m)=\{{n\atop m}\}$。

1.2.7 小节

1. 0；1；3/2。

2. 以上界$1/2^m$代替每项$1/(2^m+k)$。

3. $S_{2^m}^{(r)}\le\sum_{0\le k\le m}(1/2^{r-1})^k\cdot 2^{r-1}/(2^{r-1}-1)$为一上界。

4. （b）和（c）。

5. 9.78760 60360 44382…。

6. 用归纳法和等式1.2.6-42。

7. $T(m+1,n)-T(m,n)=1/(m+1)-1/(mn+1)-\cdots-1/(mn+n)<1/(m+1)-1/(mn+n)-\cdots-1/(mn+n)=1/(m+1)-n/(m+1)\le 0$。当$m=n=1$时拥有极大值，而当$m$和$n$取得充分大的值时，它趋于极小值。由等式（3），最大的下界

为 γ，这决不被真正地达到。对于这个结果之一推广，见于 *AMM* 70 (1963), 575～577。

8. 由斯特林近似公式，$\ln n!$ 近似于 $\left(n+\frac{1}{2}\right)\ln n - n + \ln\sqrt{2\pi}$；而 $\sum_{1\leqslant k\leqslant n} H_k$ 近似于 $(n+1)\ln n - n(1-\gamma)+\left(\gamma+\frac{1}{2}\right)$；其差近似于 $\gamma_n+\frac{1}{2}\ln n + .158$。

9. $-1/n$。

10. 把左边拆成两个和式，在第二个和式中把 k 变成 $k+1$。

11. $2-H_{n+1}/n-1/(n+1)$。

12. 1.000…正确到超过 300 位小数!

14. 见 1.2.3 小节，例 2。第二个和是 $\frac{1}{2}(H_{n+1}^2-H_{n+1}^{(2)})$。

15. $\sum_{1\leqslant j\leqslant n}(1/j)\sum_{j\leqslant k\leqslant n} H_k$ 可以借助于正文中的公式来求和；答案为 $(n+1)H_n^2-(2n+1)H_n+2n$。

16. $H_{2n+1}-\frac{1}{2}H_n$。

17. 取分母为 $(p-1)!$，此乃真正分母之一倍数，但不是 p 的倍数。我们仅须证明，对应的分子 $(p-1)!/1+(p-1)!/2+\cdots+(p-1)!/(p-1)$ 是 p 的倍数。取 modulo p，$(p-1)!/k\equiv(p-1)!k'$，其中 k' 可通过关系 $kk' \bmod p = 1$ 确定之。集合 $\{1',2',\cdots,(p-1)'\}$ 恰是集合 $\{1,2,\cdots,(p-1)\}$；所以分子同余于 $(p-1)!(1+2+\cdots+p-1)\equiv 0$。事实上，已经知道分子是 p^5 的倍数，当 $p>3$；见哈迪和赖特 (Wright)，《数论》(The Theory of Numbers)，7.8 节。

18. 当 $n=2^k m$ 时，其中 m 为奇数，这和等于 $2^{2k}m_1/m_2$，其中 m_1 和 m_2 两者都是奇数。*AMM* 67 (1960)，924～925。

19. 仅有 $n=0$ 和 $n=1$。对于 $n\geqslant 2$，命 $k=\lfloor\log_2 n\rfloor$。恰恰有一项，它的分母为 2^k。所以 $2^{k-1}H_n-\frac{1}{2}$ 是分母中仅含奇质数的诸项之和。如若 H_n 是一整数，则 $2^{k-1}H_n-\frac{1}{2}$ 将有一个等于 2 的分母。

20. 逐项展开被积函数。也见 *AMM* 69 (1962)，239，以及亨·沃·古尔德的论文，《数学杂志》(Mathematics Magazine) 34 (1961)，317～321。

21. $H_{n+1}^2-H_{n+1}^{(2)}$。

22. $(n+2)(H_{n+1}^2-H_{n+1}^{(2)})-2(n+1)H_n+2n$。

23. $\Gamma'(n+1)/\Gamma(n+1)=1/n+\Gamma'(n)/\Gamma(n)$，因为 $\Gamma(x+1)=x\Gamma(x)$。因此 $H_n=\gamma+\Gamma'(n+1)/\Gamma(n+1)$。函数 $H_n-\gamma$ 称作 *psi*（普赛）函数或双伽玛函数。对于有理数 n 的某些值，见于附录 B。

24. 它是

$$x\lim_{n\to\infty} e^{(H_n-\ln n)x}\prod_{1\leqslant k\leqslant n}\left(\left(1+\frac{x}{k}\right)e^{-x/k}\right)=\lim_{n\to\infty}\frac{x(x+1)\cdots(x+n)}{n^x n!}$$

注意：在上题中考虑的 H_n 之推广，因此等于 $H_n^{(r)}=\sum_{k\geqslant 0}(1/(k+1)^r-1/(k+1+x)^r)$，当

$r=1$；对于充分大的 r 值，亦可使用这同一思想。

1.2.8 小节

1. F_{k+2}；答案是 $F_{14}=337$ 对。

2. $\ln(\phi^{1000}/\sqrt{5})=1000\ln\phi-\frac{1}{2}\ln 5=480.40711$；$\log_{10}F_{1000}$ 是 $1/(\ln 10)$ 乘此数，或曰 208.64；因此 F_{1000} 是一个 209 位的数，打头一位是 4。

4. 0，1，5；此后 F_n 增加太快。

5. 0，1，12。

6. 用归纳法（这个等式对于负的 n 也成立，参照习题 8）。

7. 如果 d 是 n 的一个真因子，则 F_d 整除 F_n。现在 F_d 大于 1 且小于 F_n，假定 d 大于 2。唯一的没有大于 2 的真因子的非质数是 $n=4$（因为如果 $d=2$，则 $n/d\geqslant 2$），$F_4=3$ 是仅有的例外。

8. $F_{-1}=1$；$F_{-2}=-1$；$F_{-n}=(-1)^{n+1}F_n$，通过对 n 用归纳法。

9. (15) 不行。通过归纳的论证，即假定某件事对于 n 和更大者为真，证明它对于 $n-1$ 为真，得证其它的都成立。

10. 由等式(14)，当 n 为偶数时它大于，而当 n 为奇数时它小于。

11. 用归纳法；参照习题 9。这是习题 13（a）的一种特殊情形。

12. 如果 $\mathscr{G}(z)=\sum\mathscr{F}_nz^n$，则 $(1-z-z^2)\mathscr{G}(z)=z+F_0z^2+F_1z^3+\cdots=z+z^2G(z)$。因此，$\mathscr{G}(z)=G(z)+zG(z)^2$；由等式(17)，我们导出 $\mathscr{F}_n=((3n+3)/5)F_n-(n/5)F_{n+1}$。

13. (a) $a_n=rF_{n-1}+sF_n$。(b) 因为 $(b_{n+2}+c)=(b_{n+1}+c)+(b_n+c)$，我们可以考虑一个新的序列 $b'_n=b_n+c$。应用 (a) 于 b'_n，我们得到答案为 $cF_{n-1}+(c+1)F_n-c$。

14. $a_n=F_{m+1}F_{n-1}+(F_{m+2}+1)F_n-\binom{n}{m}-\binom{n+1}{m-1}-\cdots-\binom{n+m}{0}$。

15. $c_n=xa_n+yb_n+(1-x-y)F_n$。

16. F_{n+1}。用归纳法，以及 $\binom{n+1-k}{k}=\binom{n-k}{k}+\binom{(n-1)-(k-1)}{k-1}$。

17. $(x^{n+k}-y^{n+k})(x^{m-k}-y^{m-k})-(x^n-y^n)(x^m-y^m)=(xy)^n(x^{m-n-k}-y^{m-n-k})(x^k-y^k)$。现在置 $x=\phi$，$y=\hat{\phi}$，且以 $(\sqrt{5})^2$ 来除。

18. 它是 F_{2n+1}。

19. 命 $u=\cos 72^\circ$，$v=\cos 36^\circ$。我们有 $u=2v^2-1$；$v=1-2\sin^2 18^\circ=1-2u^2$，因此 $u+v=2(v^2-u^2)$，即是，$1=2(v-u)$，所以 $4v^2-2v-1=0$，$v=\frac{1}{2}\phi$。

20. $F_{n+2}-1$。

21. 乘以 x^2+x+1；解答是 $(x^{n+1}F_{n+1}+x^{n+2}F_n-x)/(x^2+x-1)$。如果分母为 0，则 x 为 $1/\phi$ 或 $1/\hat{\phi}$；于是解答是

$$((n+1)x^nF_{n+1}+(n+2)x^{n+1}F_n-1)/(2x+1)$$

22. F_{m+2n}；见下一题，$t=2$。

23. $\dfrac{1}{\sqrt{5}}\sum_k\binom{n}{k}(\phi^k F_t^k F_{t-1}^{n-k}\phi^m-\hat{\phi}^k F_t^k F_{t-1}^{n-k}\hat{\phi}^m)$

$$=\frac{1}{\sqrt{5}}(\phi^m(\phi F_t+F_{t-1})^n-\hat{\phi}^m(\hat{\phi}F_t+F_{t-1})^n)=F_{m+tn}$$

24. F_{n+1}（在头一行按余因子展开）。

25. $2^n\sqrt{5}F_n=(1+\sqrt{5})^n-(1-\sqrt{5})^n$。

26. 由费尔马定理，$2^{p-1}\equiv 1$；现在应用上一题和习题 1.2.6-10(b)。

27. 如果 $p=2$，则它为真。否则 modulo p，$F_{p-1}F_{p-1}-F_p^2=-1$；由上题和费尔马定理，$F_{p-1}F_{p+1}\equiv 0$。由于 $F_{p+1}=F_p+F_{p-1}$，因此仅仅 F_{p-1} 和 F_{p+1} 两者中的一个可为 p 的倍数。

28. $\hat{\phi}^n$. 注意：对于关系式 $a_{n+1}=Aa_n+B^n$，$a_0=0$，其解为：

$a_n=(A^n-B^n)/(A-B)$ 如果 $A\neq B$，$a_n=nA^{n-1}$ 如果 $A=B$。

29.

$\left(\left(\binom{n}{0}\right)\right)$	$\left(\left(\binom{n}{1}\right)\right)$	$\left(\left(\binom{n}{2}\right)\right)$	$\left(\left(\binom{n}{3}\right)\right)$	$\left(\left(\binom{n}{4}\right)\right)$	$\left(\left(\binom{n}{5}\right)\right)$	$\left(\left(\binom{n}{5}\right)\right)$
1	0	0	0	0	0	0
1	1	0	0	0	0	0
1	1	1	0	0	0	0
1	2	2	1	0	0	0
1	3	6	3	1	0	0
1	5	15	15	5	1	0
1	8	40	60	40	8	1

(b) 由 (6) 推出。

30. 对 m 用归纳法，这一命题对于 $m=1$ 是明显的：

a) $\displaystyle\sum_k\left(\left(\binom{m}{k}\right)\right)(-1)^{\lceil(m-k)/2\rceil}F_{n+k}^{m-2}F_k=F_m\sum_k\left(\left(\binom{m-1}{k-1}\right)\right)(-1)^{\lceil(m-k)/2\rceil}F_{n+k}^{m-2}\equiv 0$。

b) $\displaystyle\sum_k\left(\left(\binom{m}{k}\right)\right)(-1)^{\lceil(m+k)/2\rceil}F_{n+k}^{m-2}(-1)^kF_{m-k}$

$$=F_m\sum_k\left(\left(\binom{m-1}{k}\right)\right)(-1)^{\lceil(m-1-k)/2\rceil}F_{n+k}^{m-2}(-1)^m\equiv 0。$$

c) 由于 $(-1)^kF_{m-k}=F_{k-1}F_m-F_kF_{m-1}$，我们由 (a)，(b) 结论

$$\sum_k\left(\left(\binom{m}{k}\right)\right)(-1)^{\lceil(m+k)/2\rceil}F_{n+k}^{m-2}F_{k-1}\equiv 0$$

由于 $F_m\neq 0$。

d) 由于 $F_{n+k}=F_{k-1}F_n+F_kF_{n+1}$，由 (a) 和 (c) 推出结果。这个结果也可以通过使用习题 1.2.6-58 中的“q 项式定理”，以更一般的形式来进行证明。也可见约·赖尔登，Duke Math. J. 29 (1962)，5～12。

31. 习题 8 和 11。

32. 取 modulo F_n，斐波那契序列为 0，1，…，F_{n-1}，0，F_{n-1}，$-F_{n-2}$，…。

33. 如果已经知道契贝雪夫多项式，则人们即可利用它的性质来进行证明。直接地，我们得到 $\cos z=\frac{1}{2}(e^{iz}+e^{-iz})=-i/2$。然后利用 $\sin(n+1)z+\sin(n-1)z=2\sin(nz)\cos z$。

34. 证明：对于 F_{k_1} 仅有的可能的值，是小于或等于 n 的最大的斐波那契数；因此 $n-F_{k_1}$ 小于 F_{k_1-1}，而且由归纳法，有一个唯一的 $n-F_{k_1}$ 之表示。这个证明的轮廓，十分类似于唯一因子分解定理之证明。斐波那契数系的讨论是由埃·泽肯多夫（E. Zeckendorf）给出的〔见 Simon Stevin 29(1952), 190～195; Bull. de la Soc. Royale des Sciences de Liège 41 (1972), 179～182〕；习题 5.4.2-10 中讨论了推广。

35. 见乔·马·伯格曼（G. M. Bergman），《数学杂志》（Mathematics Magazine）31 (1957), 98～110。

36. 我们可以考虑无限的串 S_∞，因为 S_n（$n>1$）由 S_∞ 的头 F_n 个字母组成。没有双重的 a，没有三重的 b。S_n 含有 F_{n-2} 个 a，F_{n-1} 个 b。如果我们象在习题 34 中那样，以斐波那契数系来表达 $m-1$，那么，S_∞ 的第 m 个字母是 a，当且仅当 $k_r=2$。S_∞ 的第 k 个字母是 b，当且仅当 $\lfloor(k+1)\phi^{-1}\rfloor-\lfloor k\phi^{-1}\rfloor=1$；在头 k 个字母中 b 的个数因此是 $\lfloor(k+1)\phi^{-1}\rfloor$。

37. 〔斐波那契季刊 1 (Dec. 1963), 9～12.〕考虑习题 34 的斐波那契数系；如果在这个数系中 $n=F_{k_1}+\cdots+F_{k_r}>0$，则命 $\mu(n)=F_{k_r}$。命 $\mu(0)=\infty$，我们发现：（A）如果 $n>0$，则 $\mu(n-\mu(n))>2\mu(n)$。证明：$\mu(n-\mu(n))=F_{k_r-1}\geqslant F_{k_r+2}>2F_{k_r}$，因为 $k_r\geqslant 2$。（B）如果 $0<m<F_k$，则 $\mu(m)\leqslant 2(F_k-m)$。证明：命 $\mu(m)=F_j$；$m\leqslant F_{k-1}+F_{k-3}+\cdots+F_{j+(k-1-j) \bmod 2}=-F_{j-1+(k-1-j) \bmod 2}+F_k\leqslant-\frac{1}{2}F_j+F_k$。（C）如果 $0<m<\mu(n)$，则 $\mu(n-\mu(n)+m)\leqslant 2(\mu(n)-m)$。证明：这由（B）推出。（D）如果 $0<m<\mu(n)$，则 $\mu(n-m)\leqslant 2m$。证明：置 $m=\mu(n)-$（C）中之 m。

现在我们证明，如果有 n 张牌，而且如果在下轮中至多可取 q 张牌，那么，有一个获胜的取法，当且仅当 $\mu(n)\leqslant q$。证明：（a）如果 $\mu(n)>q$，则所有的取法留于位置 n'、q'，有 $\mu(n')\leqslant q'$〔这从上边的（D）推出〕。（b）如果 $\mu(n)\leqslant q$，则我们或者能在这次取法中获胜（如果 $q\geqslant n$），或者我们能做成一着停留于位置 n'，q' 的取法，有 $\mu(n')>q'$〔这从上边的（A）推出，我们的取法是取 $\mu(n)$ 张牌〕。可以看出，全部获胜的取法的集合，如果 $n=F_{k_1}+\cdots+F_{k_r}$，则是取走 $F_{k_j}+\cdots+F_{k_r}$，对于某个满足 $1\leqslant j\leqslant r$ 的 j，假定 $j=1$ 或 $F_{k_j-1}>2(F_{k_j}+\cdots+F_{k_r})$。

如果 $n=1000$，则斐波那契表示是 $987+13$；唯一幸运的夺取胜利的取法，是取 13 张牌。头一个选手总能获胜，除非 n 是一个斐波那契数。

这种很一般游戏的解答，已由艾·施文克（A. Schwenk）得到〔《斐波那契季刊》8 (1970), 225～234〕。

39. $(3^n-(-2)^n)/5$。

1.2.9 小节

1. $1/(1-2z)+1/(1-3z)$。

2. 由(6)得出，因为$\binom{n}{k}=n!/k!(n-k)!$。

3. $G'(z)=\ln(1/(1-z))/(1-z)^2+1/(1-z)^2$。由这以及$G(z)/(1-z)$的意义，我们有$\sum_{j\leq k\leq n-1}H_k=nH_n-n$；这与等式1.2.7-8一致。一般地，对于整数$m\geq 0$，有$(1-z)^{-m-1}\ln(1/(1-z))=\sum_{n\geq 0}(H_{n+m}-H_m)\binom{n+m}{m}z^n$。

4. 置$t=0$。

5. 由(11)，(22)，z^k的系数为

$$(n-1)!\sum_{0\leq j<k}\left\{{n\atop n-1}\right\}\binom{k}{j}$$

现在应用等式1.2.6-42和1.2.6-48(或者，微商并利用1.2.6-42)。

6. $(\ln(1/(1-z)))^2$，导数是调和数之生成函数的两倍；因此，这个和是$2H_{n-1}/n$。

8. $1/(1-z)(1-z^2)(1-z^3)\cdots$〔这是历史上生成函数的首批应用之一。有关伦·欧拉在十八世纪关于这个生成函数的有趣的研究，请看乔·波利亚，《数学的归纳与类似》(Induction and Analogy in Mathematics)(普林斯顿：普林斯顿大学印刷厂，1954)，第六章〕。

9. $\frac{1}{24}S_1^4+\frac{1}{4}S_1^2S_2+\frac{1}{8}S_2^2+\frac{1}{3}S_1S_3+\frac{1}{4}S_4$。

10. $G(z)=(1+x_1z)\cdots(1+x_nz)$。如同在导出等式(34)那样取对数，我们就有同样的公式，所不同的只是(24)代替(17)，而且答案也完全相同，除了以$-S_2$，$-S_4$，$-S_6$，…来代替S_2,S_4,S_6等等。我们有$a_1=S_1$，$a_2=\frac{1}{2}S_1^2-\frac{1}{2}S_2$，$a_3=\frac{1}{6}S_1^3-\frac{1}{2}S_1S_2+\frac{1}{3}S^3$，$a_4=\frac{1}{24}S_1^4-\frac{1}{4}S_1^2S_2+\frac{1}{8}S_2^2+\frac{1}{3}S_1S_3-\frac{1}{4}S_4$(参照习题9)。数$a_m$称为$x_j$的初等对称函数，而这里推导出来的公式称为牛顿恒等式。

11. 我们发现$z^2G'(z)+zG(z)=G(z)-1$。这个微分方程的解为$G(z)=(-1/z)e^{-1/z}(E_1(-1/z)+C)$，其中$E_1(x)=\int_x^\infty e^{-t}dt/t$. 而$C$是一个常数。这个函数在$z=0$的邻域内有非常病态的特性，而且$G(z)$没有幂级数展开式。其实，由于$\sqrt[n]{n!}\approx n/e$是无界的，在这种情况下生成函数不收敛；然而，它是上述函数的渐近展开，当$z<0$。

12. $\sum_{m,n\geq 0}a_{mn}w^mz^n=\sum_{m,n\geq 0}\binom{n}{m}w^mz^n=\sum_{n\geq 0}(1+w)^nz^n=1/(1-z-wz)$。

13. $\int_n^{n+1}e^{-st}f(t)dt=\frac{a_0+\cdots+a_n}{s}(e^{-sn}-e^{-s(n+1)})$。

把这些加在一起，我们求得$\mathbf{L}f(s)=G(e^{-s})/s$。

14. 参照习题1.2.6-38。

15. $G_n(z)=G_{n-1}(z)+zG_{n-2}(z)+\delta_{n0}$，所以我们求得$H(w)=1/(1-w-zw^2)$。因此，最终地，我们求得

$$G_n(z)=\left(\left(\frac{1+\sqrt{1+4z}}{2}\right)^{n+1}-\left(\frac{1-\sqrt{1+4z}}{2}\right)^{n+1}\right)/\sqrt{1+4z}$$

当$z\neq -1/4$时，$C_n(-1/4)=(n+1)/2^n$　对于$n>0$。

16. $G_n(z)=(1+z+\cdots+z^r)^n=\left(\frac{1-z^{r+1}}{1-z}\right)^n$〔注意 $r=\infty$ 的情况〕。

17. $\sum_k \binom{-w}{k}(-z)^k=\sum_k \frac{w(w+1)\cdots(w+k-1)}{k(k-1)\cdots 1}z^k=\sum_{n,k}\begin{bmatrix}k\\n\end{bmatrix}z^k w^n/k!$（或者，把它写成 $e^{w\ln(1/(1-z))}$ 而且首先按 w 的幂展开）。

18. (a)对于固定的 n 和变动的 r，由等式(27)，生成函数是

$$G_n(z)=(1+z)(1+2z)\cdots(1+nz)=z^{n+1}\left(\frac{1}{z}\right)\left(\frac{1}{z}+1\right)\left(\frac{1}{z}+2\right)\cdots\left(\frac{1}{z}+n\right)=\sum_k\begin{bmatrix}n+1\\k\end{bmatrix}z^{n+1-k}$$

因此，答案为

$$\begin{bmatrix}n+1\\n+1-r\end{bmatrix}$$

(b)类似地，由等式(28)，生成函数是

$$\frac{1}{1-z}\cdot\frac{1}{1-2z}\cdot\cdots\cdot\frac{1}{1-nz}=\sum_k\begin{Bmatrix}k\\n\end{Bmatrix}z^{k-n}$$

所以答案为

$$\begin{Bmatrix}n+r\\n\end{Bmatrix}$$

19. $$\sum_{n\geqslant 0}\left(\frac{1}{n+p/q}-\frac{1}{n+1}\right)x^{p-nq}=x^{p-q}\ln(1-x^q)-\sum_{1\leqslant k\leqslant q}\omega^{-kp}\ln(1-\omega^k x)=S_1+S_2+S_3$$

其中 $\omega=e^{2\pi i/q}$，而这里的 S_1,S_2,S_3 定义如下。

现在

$$\lim_{x\to 1^-}S_1=\lim_{x\to 1^-}x^{p-q}\ln\left(\frac{1-x^q}{1-x}\right)=\ln q$$

$$\lim_{x\to 1^-}S_2=\lim_{x\to 1^-}(x^{p-q}-1)\ln(1-x)=0$$

以及

$$\lim_{x\to 1^-}S_3=-\sum_{0<k<q}\omega^{-kp}\ln(1-\omega^k)$$

由恒等式

$$\ln(1-e^{i\theta})=\ln\left(2e^{i(\theta-\pi)/2}\cdot\frac{e^{i\theta/2}-e^{-i\theta/2}}{2i}\right)=\ln 2+\frac{1}{2}i(\theta-\pi)+\ln\sin\frac{\theta}{2}$$

我们可以把后一和数写成 S_4+S_5，其中

$$S_4=-\sum_{0<k<q}\omega^{-kp}\ln\sin\frac{k}{q}\pi=-\sum_{0<k<q/2}(\omega^{-kp}+\omega^{-(q-k)p})\ln\sin\frac{k}{q}\pi$$

$$=-2\sum_{0<k<q/2}\cos\frac{2\pi pk}{q}\ln\sin\frac{k}{q}\pi$$

而

$$S_5=-\sum_{0<k<q}\omega^{-kp}\left(\ln 2-\frac{i\pi}{2}+\frac{ik\pi}{q}\right)=\ln 2-\frac{i\pi}{2}-\frac{i\pi}{(\omega^{-p}-1)}$$

最后，

$$-\frac{i}{2}-\frac{i}{(\omega^{-p}-1)}=-\frac{i}{2}\left(\frac{1+\omega^p}{1-\omega^p}\right)=\frac{i}{2}\left(\frac{\omega^{p/2}+\omega^{-p/2}}{\omega^{p/2}-\omega^{-p/2}}\right)=-\frac{1}{2}-\cot\frac{p}{q}\pi$$

1.2.10 小节

1. $1/n$；这是$X[n]$为最大的概率。

2. $G''(1)=\sum k(k-1)p_k$，$G'(1)=\sum kp_k$。

3. min 0，ave6.49，max999，dev2.42(注意 $H_n^{(2)}$ 近似于$\pi^2/6$；见等式 1.2.7-7)。

4. $\binom{n}{k}p^kq^{n-k}$。

5. 平均值是36/5=7.2；标准离差是$6\sqrt{2}/5\approx 1.697$。

7. $A=k$ 的概率是p_{mk}。因为我们可以考虑这些值是 1，2，…，m。给定任何把 n 个位置划分成m个不相交集合的分划，有 $m!$ 种方式来分配这些数 1，…，m到这些集合中。算法M把这些值处理成就象是仅仅出现每个集合的最右元素；所以，p_{mk} 是对于任何固定的分划的平均值。例如，如果 $n=5$，$m=3$，一个分划是

$$\{X[1],\ X[4]\}\{X[2],\ X[5]\}\{X[3]\}$$

可能的排列是12312，13213，21321，23123，31231，32132。在每个分划中，我们以 $A=k$ 得到相同百分数的排列。

另一方面，如果给出更多的信息，则概率分布即改变。如果 $n=3$，$m=2$，则上述论证考虑六种可能性122，212，221，211，121，112；如果我们知道有两个 2 和一个 1，则仅仅要考虑这些可能性中的头三种。然而，这个解释与本习题的命题已不对题了。

8. $M(M-1)\cdots(M-n+1)/M^n=M!/(M-n)!M^n$。更大的$M$就更接近于这个概率为 1。

9. 命 q_{nm} 是恰有 m 个不同的值出现的概率；则从递推式

$$q_{nm}=\frac{M-m+1}{M}q_{(n-1)(m-1)}+\frac{m}{M}q_{(n-1)m}$$

我们导出

$$q_{nm}=M!\left\{{n\atop m}\right\}/(M-m)!M^n$$

也见习题1.2.6-64。

10. 这是对所有 m 求和的 $q_{nm}p_{mk}$；即是

$$\frac{1}{M^n}\sum_m\binom{M}{m}\left\{{n\atop m}\right\}\left[{m\atop k+1}\right]$$

这好象并不是关于平均值的一个简单公式，它比

$$H_M-\sum_{1\leqslant m\leqslant M}\left(1-\frac{m}{M}\right)^n m^{-1}=H_n+\sum_{1\leqslant k\leqslant n}\left(\binom{n}{k}-1\right)B_k M^{-k}k^{-1}$$

小于1。

11. 通过写出 e^{kt} 的幂级数。头一个恒等式是明显的。对于第二个，命 $u=1+M_1t+M_2t^2/2!+\cdots$；当 $t=0$ 时，我们有 $u=1$ 和 $D_u^k u=M_k$。而且，$D_u^j(\ln u)=(-1)^{j-1}(j-1)!/u^j$。

12. 由于这是一个乘积，我们加每项的半不变量。如果 $H(z)=z^n$，则 $H(e^t)=e^{nt}$，所以我们发现 $k_1=n$ 且所有其它的为0。因此，$\mathrm{mean}(G_1)=n+\mathrm{mean}(G)$ 而且所有其它的半不变量均不变(这是"半不变量"名称之由来)。

13. $$G_n(z)=\frac{\Gamma(n+z)}{\Gamma(z+1)n!}=\frac{1}{\Gamma(z+1)}\cdot\frac{(n+z)^z}{n+z}e^{-z}\left(1+\frac{z}{n}\right)^n\left(1+O\left(\frac{1}{n}\right)\right)$$
$$=\frac{n^{z-1}}{\Gamma(z+1)}\left(1+O\left(\frac{1}{n}\right)\right)$$

命 $z=z_n=\exp(t/\sigma_n)$。现在 $z_n\to 1$，所以当 $n\to\infty$ 时 $\Gamma(z+1)$ 项可以忽略。现在极限是

$$\exp(-t\mu_n/\sigma_n+(e^{t/\sigma_n}-1)\ln n)=\exp\left(t\left(\frac{\ln n-\mu_n}{\sigma_n}\right)+\frac{t^2\ln n}{2\sigma_n^2}+O\left(\frac{1}{\ln n}\right)\right)\to\exp\left(\frac{t^2}{2}\right)$$

14. $e^{-tpn/\sqrt{pqn}}(q+pe^{t/\sqrt{pqn}})^n=(qe^{-tp/\sqrt{pqn}}+pe^{tq/\sqrt{pqn}})^n$。把指数展开成幂级数，得到 $(1+t^2/2n+O(n^{-3/2}))^n=\exp(n\ln(1+t^2/2n+O(n^{-3/2})))=\exp(t^2/2+O(n^{-1/2}))\to\exp(t^2/2)$。

15. 等式(8)中的 $G(z)$。在这种方式下，关于概率的一个生成函数，总可以解释为一个数量的平均值。

16. (a) $\sum_{k\geqslant 0}e^{-\mu}(\mu z)^k/k!=e^{\mu(z-1)}$。(b) $\ln e^{\mu(e^t-1)}=\mu(e^t-1)$，所以所有半不变量均等于 μ。(c) $\exp(-tnp/\sqrt{npq})\times\exp(np(t/\sqrt{np}+t^2/2np+O(n^{-3/2})))=\exp(t^2/2+O\times(n^{-1/2}))$。

17. (a) $f(z)$，$g(z)$ 的系数非负而且 $f(1)=g(1)=1$。显然，$h(z)$ 享有同样的特征，因为 $h(1)=g(f(1))$，而且 h 的系数，是具有非负系数的 f，g 之系数的多项式。(b) 命 $f(z)=\sum p_k z^k$，其中 p_k 是某个事件产生一个 k 的"得分"的概率。命 $g(z)=\sum q_k z^k$，其中 q_k 是由 f 描述的事件恰恰发生 k 次的概率(事件的每一出现与其它的无关)。则 $h(z)=\sum r_k z^k$，其中 r_k 是出现的事件的得分之和等于 k 的概率(这是容易看出的，如果我们注意到 $f(z)^k=\sum s_t z^t$，其中 s_t 是在事件的 k 次独立出现中得到一总共的得分 t 的概率)。例子：如果 f 给出一个男人有 k 个男孩的概率，而且如果 g 给出在 n 代有 k 个男子的概率，则 h 就给出在 $n+1$ 代有 k 个男子的概率——假定无关性。(c) $\mathrm{mean}(h)=\mathrm{mean}(g)\mathrm{mean}(f)$；$\mathrm{var}(h)=\mathrm{var}(g)\mathrm{mean}^2(f)+\mathrm{mean}(g)\mathrm{var}(f)$。

18. 把$X[1],\cdots,X[n]$的选择考虑作一个过程，其中我们首先置所有的n，然后在这些n当中放置所有的$n-1$，…，最后在其中放置1。当我们在数$r+1$，…，n当中放置r时，从右到左的局部极大值的个数增加1，当且仅当，我们在最右端放置一个r。这以$k_r/(k_r+k_{r+1}+\cdots+k_n)$的概率出现。

1.2.11.1 小节

1. 0。

2. 每个O符号表示一个不同的近似量；因为左边可以是$f(n)-(\cdots f(n))=2f(n)$。我们充其量只能说是$O(f(n))-O(f(n))=O(f(n))$。为证明后者，注意，如果$|x_n|\leqslant M_1|f(n)|$和$|y_n|\leqslant M_2|f(n)|$，则$|x_n-y_n|\leqslant|x_n|+|y_n|\leqslant(M_1-M_2)|f(n)|$。

3. $n(\ln n)+\gamma n+O(\sqrt{n}\ln n)$。

4. $\ln a+(\ln a)^2/2n+(\ln a)^3/6n^2+O(n^{-3})$。

5. (a)取$M=|c_0|/r^m+|c_1|/r^{m-1}+\cdots+|c_m|$(参照等式(3)后边的正文)。(b)反驳：命$P(x)=1,m=1$；则$|1|>Mx$ 当$x<1/M$；因此对M的所有选择，这条件不成立。

6. O符号的可变的个数n，已为一单个的O符号所代替，错误地蕴含着一单个的M值对于每个项$|1|\leqslant Mn$都将满足。如我们所知道的，给定的和实际上是$O(n^3)$。最后的等式$\sum_{1\leqslant k\leqslant n}O(n)=O(n^2)$，完全正确。

7. 如果x是正数，则幂级数告诉我们$e^x>x^{m+1}/(m+1)!$，因此e^x/x^m的比率不能为任何M所限定。

8. 以e^n代替n，而且应用前面的习题。

9. 如果$|f(x)|\leqslant M|x|^m$，则$e^{|f(x)|}\leqslant e^{M|x|^m}=1+|x|^m(M+M^2|x|^m/2!+M^3|x|^{2m}/3!+\cdots)\leqslant 1+|x|^m(M+M^2r^m/2!+M^3r^{2m}/3!+\cdots)$。

10. “如果$f(x)=O(x^m)$，$|x|\leqslant r$，则存在一个正数r'使得$\ln(1+f(x))=O(x^m)$，$|x|\leqslant r'$。”证明。取r'，r''，使得$|f(x)|\leqslant r''<1$当$|x|\leqslant r'$时。于是$|\ln(1+f(x))|\leqslant|f(x)|+\frac{1}{2}|f(x)|^2+\frac{1}{3}|f(x)|^3+\cdots\leqslant|x|^m M\left(1+\frac{1}{2}r''+\frac{1}{3}r''^2+\cdots\right)$。

11. 以$m=1$，$x=\ln n/n$，应用等式(11)。这是正当的，因为$\ln n/n$保持以某个正的值r为界（对于充分大的n，它趋于0）。

1.2.11.2 小节

1. $x=(B_0+B_1x+B_2x^2/2!+\cdots)e^x-(B_0+B_1x+B_2x^2/2!+\cdots)$；应用等式1.2.9-11。

2. 为进行分部积分，函数$B_{m+1}(\{x\})$必须是连续的。

3. $|R_{2k}|\leqslant\left|\frac{B_{2k}}{(2k)!}\right|\int_1^n|f^{(2k)}(x)|dx$。

4. $\sum_{0\leqslant k<n}k^m=\frac{1}{1+m}n^{m+1}+\sum_{1\leqslant k\leqslant m}\frac{B_k}{k!}\frac{m!}{(m-k+1)!}n^{m-k+1}=\frac{1}{m+1}B_{m+1}(n)-\frac{1}{m+1}B_{m+1}$。

5. 由此得出

$$\kappa=\sqrt{2}\lim_{n\to\infty}\frac{2^{2n}(n!)^2}{\sqrt{n}\,(2n)!}$$

$$\kappa^2=\lim_{n\to\infty}\frac{2}{n}\,\frac{n^2(n-1)^2\cdots(1)^2}{\left(n-\frac{1}{2}\right)^2\left(n-\frac{3}{2}\right)^2\cdots\left(\frac{1}{2}\right)^2}=4\,\frac{2\cdot2\cdot4\cdot4\cdot}{1\cdot3\cdot3\cdot5\cdot}\cdots=2\pi$$

6. 假定 $c>0$ 并考虑 $\sum_{0\leqslant k<n}\ln(k+c)$。我们得出

$$\ln(c(c+1)\cdots(c+n-1))=(n+c)\ln(n+c)-c\ln c-n-\frac{1}{2}\ln c+\sum_{1<k\leqslant m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{(n+c)^{k-1}}-\frac{1}{c^{k-1}}\right)+R_{mn}$$

而且

$$\ln(n-1)!=\left(n-\frac{1}{2}\right)\ln n-n+\sigma+\sum_{1<k\leqslant m}\frac{B_k(-1)^k}{k(k-1)}\left(\frac{1}{n^{k-1}}\right)-\frac{1}{m}\int_n^\infty\frac{B_m(\{x\})dx}{x^m}$$

现在 $\ln\Gamma_{n-1}(c)=c\ln(n-1)+\ln(n-1)!-\ln(c\cdots(c+n-1))$；代入并令 $n\to\infty$，我们得到

$$\ln\Gamma(c)=-c+\left(c-\frac{1}{2}\right)\ln c+\sigma+\sum_{1<k\leqslant m}\frac{B_k(-1)^k}{k(k-1)c^{k-1}}-\frac{1}{m}\int_0^\infty\frac{B_m(\{x\})dx}{(x+c)^m}$$

这证明了，$\Gamma(c+1)=ce^{\ln\Gamma(c)}$ 有着等同于我们对 $c!$ 导出的展开式。

7. $A\cdot n^{n^2/2+n/2+1/12}e^{-n^2/4}$，其中 A 是一个常数（它是“格莱西尔(Glaisher)常数”1.2824271…）。为得到这个结果，对 $\sum_{1\leqslant k<n}k\ln k$ 应用欧拉求和公式。如果我们以

$$\exp(-B_4/2\cdot3\cdot4n^2-\cdots-B_{2t}/(2t-2)(2t-1)(2t)n^{2t-2}+O(1/n^{2t}))$$

来乘上边的答案，则得到一个更精确的公式。这个公式使得有可能来计算格莱西尔常数到六位小数，如果我们命 $t=3$，$n=4$。

1.2.11.3 小节

1. 分部积分。

2. 在积分中代入 e^{-t} 的级数。

3. 见等式1.2.9-11和习题1.2.6-48。

4. $1+1/u$ 作为 v 的一个函数是有界的，因为当 v 从 r 趋向无穷时，它趋于 0。以 M 代替它，得到的积分为 Me^{-rx}。

5. $f''(x)=f(x)\left(\frac{\left(n+\frac{1}{2}\right)\left(n-\frac{1}{2}\right)}{x^2}-\frac{(2n-1)}{x}+1\right)$ 对于 $0\leqslant x\leqslant n-1$ 有固定的正负号。所以 $|R|\leqslant\frac{1}{12}|f'(n-1)|+\frac{1}{12}\int_{n-1}^n|f''(x)|dx$。

6. 它是 $n^{n+\beta}\exp((n+\beta)(\alpha/n-\alpha^2/2n^2+O(n^{-3})))$，等等。

7. 被积函数作为 x^{-1} 的幂级数，有着 x^{-n} 的系数达 $O(u^{2n})$。在积分之后，x^{-3} 中的项是 $Cu^7/x^3=O(x^{-5/4})$，等等。为了得到答案中的 $O(x^{-2})$，我们可以抛弃具有 $4m-n\geqslant9$

的诸项u^m/x^m(参照下一题)。因此，展开$\exp(-u^2/2x)\exp(u^3/3x^2)\cdots$最终就导致答案

$$yx^{1/4}-\frac{y^3}{6}x^{-1/4}+\frac{y^5}{40}x^{-3/4}+\left(-\frac{y^4}{12}\right)x^{-1}-\frac{y^7}{336}x^{-5/4}-\frac{y^6}{36}x^{-3/2}$$

$$+\left(\frac{y^9}{3456}-\frac{y^5}{20}\right)x^{-7/4}+O(x^{-2})$$

8. 被积函数可被展开成形如$c_{mn}u^m/x^n$的项。这些项被积分成为$O(x^{(m+1)r-n})$。〔注意，如果$r>\frac{1}{2}$，则$e^{-u^2/2x}$的级数被积分成为对于充分大的x为发散之级数，因此假定$r\leqslant\frac{1}{2}$是必要的〕。乘两项$(u^{m_1}/x^{n_1})(u^{m_2}/x^{n_2})$，如果$(m_1+1)r-n_1\leqslant -s$，则我们有

$$(m_1+m_2+1)r-(n_1+n_2)=(m_1+1)r-n_1+(m_2+1)r$$

$$-n_2-r\leqslant -s+r-r=-s$$

因此我们可以展开$\exp(-u^2/2x)\exp(u^3/3x^2)\cdots$，并且在把所有的因子乘在一起以前，抛弃具有$(m+1)r-n\leqslant -s$的所有项；而且可以删掉所有具有$(p+1)r-p+1\leqslant -s$的项$\exp(u^p/px^{p-1})$，即是，具有$p>\lceil(s+2r)/(1-r)\rceil$的那些。

9. 我们可以假定$p\neq 1$，因为$p=1$已由定理A给出。我们也可以假定$p\neq 0$，因为该情况是平凡的。

情况1　$p<1$。代入$t=px(1-u)$而且然后$v=-\ln(1-u)-pu$。我们有$dv=((1-p+pu)/(1-u))du$，所以对于$0\leqslant u\leqslant 1$这个变换是单调的，而且我们得到一个形如

$$\int_0^\infty xe^{-xv}dv\left(\frac{1-u}{1-p+pu}\right)$$

的积分。带圆括弧的量是

$$\frac{1}{1-p}\left(1-\frac{v}{(1-p)^2}+\cdots\right)$$

答案因此是

$$\frac{p}{1-p}(pe^{1-p})^x\frac{e^{-x}x^x}{\Gamma(x+1)}\left(1-1/(p-1)^2x+O(x^{-2})\right)$$

情况2　$p>1$。这是$1-\int_{px}^\infty(\quad)$。在后边的积分中，代入$t=px(1+u)$，然后$v=pu-\ln(1+u)$，而且象在情况1那样进行。答案结果是，等于情况1的同一公式加1。注意$pe^{1-p}<1$，所以$(pe^{1-p})^x$非常之小。

10. $\frac{p}{p-1}(pe^{1-p})^xe^{-x}x^x\left(1-e^{-y}-\frac{e^{-y}(e^y-1-y-y^2/2)}{x(p-1)^2}+O(x^{-2})\right)$。

11. 首先，$xQ_x(n)+R_{1/x}(n)=n!\,(x/n)^ne^{n/x}$（情况$x=-1$在这里是有趣的！），我们得到

$$Q_x(n)=\frac{1}{1-x}-\frac{x}{(1-x)^2n}+O(n^{-2})$$

$$R_x(n)=\frac{1}{1-x}-\frac{x}{(1-x)^2n}+O(n^{-2})，\text{如果}x<1$$

$$Q_x(n)=\frac{1}{x}\,n!\left(\frac{x}{n}\right)^ne^{nx}-\frac{1}{x-1}+\frac{1}{(x-1)^2n}+O(n^{-2})$$

$$R_x(n)=n!\left(\frac{x}{n}\right)^n e^{n\alpha}-\frac{1}{x-1}+\left(\frac{x}{(x-1)^3 n}\right)+O(n^{-2}),\quad \text{如果 } x>1$$

这些公式当$|x|<1$时十分容易验证，而且$Q_x(n)$与$R_{1/x}(n)$之间的关系把这推广到$|x|>1$。剩下$x=-1$的情况；这要求对极限之更精湛的运用。关于渐近展开之更进一步的细节，以及它与第二类斯特林数之联系，见伦·卡里兹(L. Carlitz)，《美国数学协会会报》(Proc. Amer. Math. Soc.)16(1965)，248～252。

12. $\gamma\left(\frac{1}{2},\frac{1}{2}x^2\right)/\sqrt{2}$。

15. 以二项式定理展开被积函数，我们求得积分为$1+Q(n)$。

16. 把$Q(k)$写作一个和，并利用等式1.2.6-49交换求和的次序。

17. $S(n)=\sqrt{\pi n/2}+\frac{2}{3}-\frac{1}{24}\sqrt{\pi/2n}-\frac{4}{135}n^{-1}+\frac{49}{1152}\sqrt{\pi/2}n^{-\frac{3}{2}}+O(n^{-2})$〔注意$S(n+1)+P(n)=\Sigma_{k\geq 0}k^{n-k}k!/n!$，而$Q(n)+R(n)=\Sigma_{k\geq 0}n!/k!n^{n-k}$〕。

1.3.1 小节

1. 4；因而每个字节将含有$3^4=81$个不同的值。

2. 5，因为有5个字节总是足够了，但4个不够。

3. (0:2)；(3:3)；(4:4)；(5:5)。

4. 假定变址寄存器4含有一个大于或等于2000的值，使得经变址之后得到一个有效的内存地址。

5. "DIV-80, 3(0:5)"或者简单地"DIV-8, 3"。

6. (a)置rA成为 | - | 5 | 1 | 200 | 15 |。(b)置rI2成为−200。(c)置rX成为 | + | 0 | 0 | 5 | 1 | ? |。(d)无定义，因为我们不能把这样大的一个值装入一个变址寄存器。(e)置rX成为 | - | 0 | 0 | 0 | 0 | 0 |。

7. 设运算之前|rAX|这一数量为n，设V的数量为d，经运算之后，rA的数量为$\lfloor n/d\rfloor$，而且rX的数量为$n \bmod d$。以后rX的符号为rA先前的符号；如果rA和V先前的符号相同，则以后rA的符号为+，否则它为−。

8. rA ← | + | 0 | 617 | 0 | 1 |；rX ← | - | 0 | 0 | 0 | 1 | 1 |。

9. ADD, SUB, DIV, NUM, JOV, JNOV, INCA, DECA, INCX, DECX。

10. CMPA, CMP1, CMP2, CMP3, CMP4, CMP5, CMP6, CMPX(还有FCMP，浮点的比较)。

11. MOVE, LD1, LD1N, INC1, DEC1, ENT1, ENN1。

12. INC3 0, 3。

13. "JOV1000"除了执行时间之外毫无差别。"JNOV1001"在大多数情况下造成rJ的不同设置。"JNOV1000"造成异乎寻常的差别，因为它可能使计算机封锁于无限的循环之中。

14. NOP本身自不待言；ADD, SUB于$F=(0:0)$；HLT(取决于你如何解释本题的叙述)；任何具有地址和变址为0的移位；MOVE于$F=0$；JSJ具有地址等于指令单

元加1；任何INC或DEC指令，具有地址和变址为0；ENTi 0，i对于$1 \leqslant i \leqslant 6$；SLC或SRC，以10的倍数为地址。

15. 70；80；120(记录大小的5倍)。

16. (a)STZ 0；ENT1 1；MOVE 0(49)；MOVE 0(50)。如果已经知道字节大小等于100，则将仅仅需要一条MOVE指令，但我们并未获许对字节大小作出假定。(b)用100条STZ。

```
17.(a)STZ  0, 2     (b)          STZ    0
       DEC2 1                    ENT1   1
       J2NN 3000                 JMP    3004
                          (3003) MOVE   0(63)
                          (3004) DEC2   63
                                 J2NN   3003
                                 INC2   63
                                 ST2    3008(4:4)
                          (3008) MOVE   0
```

(一个稍快的但十分不合理的程序，使用993个STZ:JMP3995；STZ 1，2；STZ 2，2；…；STZ 993，2；J2N 3993；DEC2 993；J2NN3004；ENN1 0,2；JMP 3004，1。)

18. (如果你已经正确地遵循这些指令，则在进行ADD时将出现一个溢出，以后在寄存器A中有负0。)答案：溢出开关被置位，比较指示器置成EQUAL，rA置成 |-|30|30|30|30|30|，rX置成 |-|31|30|30|30|30|，rI1置成+3，而且内存单元0001，0002置成0(除非程序出现在0000～0015单元中)。

19. $24u = (2+1+2+2+1+1+1+2+2+1+2+2+3+1+1)u$。

```
20.(3990)   ENT1    0
            MOVE    3995    (对于MOVE的标准的F为1)
            MOVE    0(49)
   (3993)   MOVE    0(50)
            JMP     3993
   (3995)   HLT     0
```

21. (a)不能，除非它可以通过外部的手段被置成0(见习题26的“GO按钮”)，因为一个程序仅能通过由单元$N-1$的转移来置rJ←N。

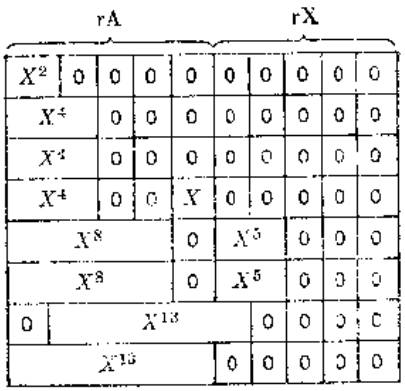
```
(b)         LDA   -1, 4
            LDX   3004
            STX   -1, 4
            JMP   -1, 4
   (3004)   JMP   3005
   (3005)   STA   -1, 4
```

22. 极小时间：如果b是字节大小，则$|X^{18}|<b^5$之假定意味着$X^2<b$，所以X^2可包含在一个字节内。下面是耶·南·帕特(Y. N. Patt)利用这一事实得到的巧妙解答。

```
   (3000)   LDA   2000
            MUL   2000(1:5)
            STX   3500(1:1)
```

```
                SRC     1
                MUL     3500
                STA     3501
                ADD     2000
                MUL     3501(1:5)
                STX     3501
                MUL     3501(1:5)
                SLAX    1
                HLT     0
(3500)          NOP     0
(3501)          NOP     0
```

空间 = 14；时间 = 54u。



按照在 4.6.3 小节提出的理论，至少有 5 个乘法是“必要的”，而这个程序却仅用了 4 个！而且事实上在下面还有一个甚至更好的解。

极小空间

```
(3000)          ENT4    12
                LDA     2000
(3002)          MUL     2000
                SLAX    5
                DEC4    1
                J4P     3002
                HLT     0
```

空间 = 7；时间 = 171u。

真正的极小时间：如罗·威·弗洛伊德所指出的，这些条件意味着$|X|\leqslant 6$，所以通过访问一个表，可以达到极小的执行时间：

```
(3000)          LD1     2000
                LDA     3500,1
                HLT     0
(3494)          (-6)^13
(3495)          (-5)^13
  ⋮
(3506)          (+6)^13
```

空间 = 16；时间 = 4u。

对于小的字节大小，单元$\pm 6^{13}$将不出现。

25. 由罗·计·狄克逊(R. D. Dixon)给出的下列解答，看来满足所有的条件：

```
(3000)          ENT1    4
(3001)          LDA     200
                SRA     0,1
                SRAX    1
                DEC1    1
                J1NN    3001
                SLAX    5
                HLT     0
```

24. (a) DIV3500，其中3500= | + | 1 | 0 | 0 | 0 | 0 | 。

(b) SRC 4；SRA 1；SLC 5。

25. 某些想法：(a)明显的事情，象更快速的存储器，更多的输入输出设备。(b) I 场可用作 J 寄存器变址，和/或多重变址（确定两个不同的变址寄存器），和/或“间接编址”（习题2.2.2-3，4.5）。(c)变址寄存器和 J 寄存器可以扩充成满5个字节；这意味着具有更高地址的单元，可以仅通过变址来访问，但这并不是那么受不了的。如果可利用如同(b)中那样的多重变址。(d)可以加上一项中断的能力（当出现某些条件时，所有寄存器被存入特殊的单元中——比如说单元－1，－2，…中——而且通过利用一个新的操作码，形成到一个控制程序之转移，这控制程序稍后又重新启动原来的程序，见习题1.4.4-18）。(e)在一个负的内存地址中，可以加上一个“实时时钟”。(f)对于MIX的二进制的型式，可以加上“逻辑操作”（见习题2.5-28和第7章）。(g)一个“执行命令”，指的是来执行在单元 M 处的指令，可以是C＝5的另一种变形。(h)C＝48，…，55的另一种变形，可以置CI+寄存器：M。

26. 下面的程序是迄今发现之最短者，对于它传输的卡片不依赖于字节大小。企图利用(2:5)场来获得卡片的7-10列，但这是不可能完成的，因为2×8＋5＝21。由于这个程序仅要求28条指令，故它可适应于纸带。

为使程序更易于理解，这里以符号语言介绍之，预先利用正文的下一小节的内容。

传输的卡片在1-10列有格式TRANSOnnnn，其中 nnnn 是为传移到开始执行的位置。

```
         BUFF    EQU     28              缓冲区区域是28-43      穿在卡片上的字符
                 ORIG    0
00       TEMP1   IN      16(16)          读入第一张卡片          0 06
01       READ    IN      BUFF(16)        读下一张卡片            Y 06
02               LDI     0 (0:0)         (ENT1 0)                I
03       ENTA    0                                               B =
04               JBUS    *(16)           等待输入完成            D 04
05               LD2N    BUFF+1(1:1)     —(计数+30)              Z IQ
06               STZ     BUFF+1(1:1)     清除(1:1)俾             Z I3
07               LDX     BUFF+1          可用(2:5)               Z EN
08       TEMP    NUM     0                                       E
09               STA     TEMP1           启始单元                EU
10               ENTA    30,2                —(计数)             OBB=
11       LOOP    STA     TEMP(1:1)                               H IU
12               LD3     TEMP1                                   EJ
13               JAZ     0,3             传输卡片                CA.
14               INTA    1,3             增加TEMP1               ACB=
15               STA     TEMP1                                   EU
16               LDA     BUFF+3,1(5:5)                           IA-H
17               DECA    25                                      V A =
18               STA     0,3             存符号                  CEU
```

19	LDA	BUFF + 2, 1		OAEH
20	LDX	BUFF + 3, 1		1AEN
21	NUM			E
22	STA	0, 3(1:5)	存数值	CLU
23	MOVE	0,1(2)	[INC12;]	ABG
24	LDA	TEMP(1:1)		II HI
25	DECA	1	减计数	A A =
26	JAP	LOOP		J B.
27	JMP	READ	准备新的卡片	A 9

1.3.2 小节

1. ENTX1000; STX X。

2. 行03中的指令 STJ 恢复这一地址（通过"*"来表示这样的指令的地址是方便的，这既因为它写起来简单，也因为它提供了对于程序中的出错条件之可识别的测试——在这里由于某些疏忽而未曾正确地进入一个子程序。某些人喜欢用"*-*")。

3. 从磁带设备 0 输入 100 个字；把这些中的极大值与最后一个交换；把剩下的 99 个中的极大值与这些中之最后一个交换；等等。最后这 100 个字将完全地分类成递增的序列，并且然后把结果写到磁带设备 1 上。

4. 非 0 单元：

3000:	+	0000	00	18	35
3001:	+	2051	00	05	09
3002:	+	2050	00	05	10
3003:	+	0001	00	00	49
3004:	+	0499	01	05	26
3005:	+	3016	00	01	41
3006:	+	0002	00	00	50
3007:	+	0002	00	02	51
3008:	+	0000	00	02	48
3009:	+	0000	02	02	55
3010:	-	0001	03	05	04
3011:	+	3006	00	01	47
3012:	-	0001	03	05	56
3013:	+	0001	00	00	51
3014:	+	3008	00	06	39
3015:	+	3003	00	00	39
3016:	+	1995	00	18	37
3017:	+	2035	00	02	52
3018:	-	0050	00	02	53
3019:	+	0501	00	00	53
3020:	-	0001	05	05	08

3021:	+	0000	00	01	05
3022:	+	0000	04	12	31
3023:	+	0001	00	01	52
3024:	+	0050	00	01	53
3025:	+	3020	00	02	45
3026:	+	0000	04	18	37
3027:	+	0024	04	05	12
3028:	+	3019	00	00	45
3029:	+	0000	00	02	05
0000:	+				2
1995:	+	06	09	19	22	23
1996:	+	00	06	09	25	05
1997:	+	00	08	24	15	04
1998:	+	19	05	04	00	17
1999:	+	19	09	14	05	22
2024:	+				2035	
2049:	+				2010	
2050:	+				3	
2051:	-				499	

（最后两个可以交换，相应地改变3001和3002）

5. 每个 OUT 都等候以前的打印机操作完成(从另一个缓冲区)。

6. (a)如果 n 不是质数，则由定义 n 有一个因子 d 满足 $1<d<n$。如果 $d>\sqrt{n}$，

则 n/d 是一个满足 $1<n/d<\sqrt{n}$ 的因子。(b)如果N不是质数，则N有一个质因子 d 满足 $1<d\leqslant\sqrt{N}$。这个算法已经验证了N没有 $\leqslant p=$PRIME[K]的质因子；而且 $N=pQ+R<pQ+p\leqslant p^2+p<(p+1)^2$。因此N的任何质因子 $>p+1>\sqrt{N}$。

我们还必须证明，当N为质数时，将有一个充分大的质数小于N，即是，第 $k+1$ 个质数 p_{k+1} 小于 $p_k^2+p_k$。这由“伯特兰(Bertrand)假设”推出：如果 p 是质数，则有一更大的质数 $<2p$。

7.(a)它参照行29的单元。(b)这个程序于是将失败；行14将参照行15而不是行25；行24将参照行15而不是行12。

8.打印100行。如果把这些行上的12000个字符衔接地排列起来，那它们将达到相当之远，而且将组成由5个空白，其后紧跟5个A，后边接着10个空白，其后紧跟5个A，后边接着15个空白，…，后边接着 $5k$ 个空白，其后紧跟5个A，后边接着 $5(k+1)$ 个空白，…，直到打印完1200个字符为止。倒数第二行以AAAAA和35个空白结尾；最后一行是全空白。总的意思是，用操作码来形成一项操作技巧。

9.在这表格中，场(4:4)是极大的F设置；(1:2)是校验程序的单元。

```
B          EQU     1(4:4)
BMAX       EQU     B-1
TABLE      NOP     GOOD(BMAX)
           ADD     FLOAT(5:5)
           SUB     FLOAT(5:5)
           MUL     FLOAT(5:5)
           DIV     FLOAT(5:5)
           HLT     GOOD
           SRC     GOOD
           MOVE    MEMORY(BMAX)
           LDA     FIELD(5:5)
           LD1     FIELD(5:5)
...
           STZ     FIELD(5:5)
           JBUS    MEMORY(19)
           IOC     GOOD(19)
           IN      MEMORY(19)
           OUT     MEMORY(19)
           JRED    MEMORY(19)
           JLE     MEMORY
           JANP    MEMORY
...
           JXNP    MEMORY
           ENNA    GOOD
...
           ENNX    GOOD
           CMPA    FLOAT(5:5)
```

```
                CMP1    FIELD(5:5)
...
                CMPX    FIELD(5:5)
BEGIN           LDA     INST
                CMPA    VALID(3:3)
                JG      BAD             I 场 > 6 ?
                LD1     INST(5:5)
                DEC1    64              C 场 ≥ 64?
                J1NN    BAD
                CMPA    TABLE+64,1(4:4) F 场 > F_max?
                JG      BAD
                LD1     TABLE+64,1(1:2) 转到特殊的
                JMP     0,1             程序
FLOAT           CMPA    VALID(1:4)      对算术操作允许的
                JE      GOOD            F = 6
FIELD           ENTA    0
                LDX     INST(4:4)       这是校验一个有效的
                DIV     =9=             部分场的
                STX     *+1(0:2)        一项技巧性
                INCA    0               的方法
                CMPA    =5=
                JG      BAD
MEMORY          LDX     INST(3:3)
                JXNZ    GOOD            如果 I = 0, 则
                LDX     INST(0:2)       保证这个
                JXN     BAD             地址是
                CMPX    =3999=          一个有效的
                JLE     GOOD            存贮单元
                JMP     BAD
VALID           CMPX    3999,6(6)
```

10. 对这个问题要领会的是，在一行（列）中可能有若干个位置上出现极小(极大)值，而且每一个都是一个潜在的马鞍点。

解法 1　在这个解法中，我们造一份其中出现有行极小的所有列的表，然后对这份表中每一列校验一个列的极大。

rX≡当前的 max 或 min; rI1 跟踪整个矩阵（从72跑到 0，除非找到了一个马鞍点); rI2≡rI1 的列下标; rI3≡极小表的大小。注意，在所有情况下，一个循环的终止条件是一个变址寄存器≤0。

```
*SOLUTION   1
A10         EQU     1008
LIST        EQU     1000
START       ENT1    72              在低下的右边的列开始
ROWMIN      ENT2    8               现在rI1是在行的第 8 列
```

```
2H          LDX     A10, 1      行极小的候选者
            ENT3    0           表空
4H          INC3    1
            ST2     LIST, 3     在表中置列下标
1H          DEC1    1           左移 1
            DEC2    1
            J2Z     COLMAX      对于行完了吗?
3H          CMPX    A10, 1
            JL      1B          rX仍是极小?
            JG      2B          新的极小?
            JMP     4B          极小的另一个出现
COLMAX      LD2     LIST, 3     从表中得到列
            INC2    64
1H          CMPX    A10, 2
            JL      NO          行min<列元素?
            DEC2    8
            J2P     1B          对于列完了吗?
YES         INC1    A10+8, 2    是; rI1←马鞍点的地址
            HLT
NO          DEC3    1           表空了吗?
            J3P     COLMAX      否; 再试
            J1P     ROWMIN      所有的行都已试完?
            HLT                 是; rI1 = 0, 无马鞍点。 ∎
```

解法 2　引进数学，给出一个不同的算法。

定理　命 $R(i)=\min_j a_{ij}$，$C(j)=\max_i a_{ij}$。元素 $a_{i_0j_0}$ 是一个马鞍点，当且仅当，$R(i_0)=\max_i R(i)=C(j_0)=\min_j C(j)$。

证明：如果 $a_{i_0j_0}$ 是一个马鞍点，则对于任何固定的 i，$R(i_0)=C(j_0)\geqslant a_{i_0j_0}\geqslant R(i)$；所以 $R(i_0)=\max_i R(i)$。类似地 $C(j_0)=\min_j C(j)$。反之，假定有给定的条件，则 $R(i_0)\leqslant a_{i_0j_0}\leqslant C(j_0)=R(i_0)$ 意味着 $a_{i_0j_0}=R(i_0)$，所以我们有一个马鞍点。 ∎

（说明这样一点可能是有趣的，就是，我们总有不等式

$\max_i \min_j a_{ij}=\min_j a_{i_0j}\leqslant \min_j \max_i a_{ij}$，对某 i_0；

所以，没有马鞍点的充分和必要条件是 $\max R(i)<\min C(j)$，即是，所有的 R 都小于所有的 C）。

以这个定理为基础的程序，先求最小的列极大，而后寻找一个相等的行极小（阶段 1：rI1 是列下标；rI2 跑遍矩阵。阶段 2：rI1 是可能的答案；rI2 跑遍矩阵；rI3 是行下标；rI4 是列下标）。

```
*SOLUTION 2
A10         EQU     1008
CMAX        EQU     1000
PHASE1      ENT1    8           在页 8 开始
3H          ENT2    64, 1
```

```
        JMP     2F
1H      CMPX    A10,2           rX 仍是极小?
        JGE     *+2
2H      LDX     A10,2           列中的新极小
        DEC2    8
        J2P     1B
        STX     CMAX+8,2        存列极大
        J2Z     1F              第一次?
        CMPA    A10,2           rA 仍为 min max?
        JLE     *+2
1H      LDA     A10,2
        DEC1    1               左移一列
        J1P     3B
PHASE2  ENT3    64              这时 rA = minC(j)
3H      ENT2    8,3             准备寻找一行
        ENT4    8
1H      CMPA    A10,2           a[i,j]≥minC(j)吗?
        JG      NO              这行中无马鞍点
        JL      2F
        CMPA    CMAX,4          a[i,j]=C(j)?
        JNE     2F
        ENT1    A10,2           可能的马鞍点
2H      DEC4    1               在行中向左移
        DEC2    1
        J4P     1B
        HLT                     找到了马鞍点
NO      DEC3    8
        ENT1    0               试另一行
        J3P     3B
        HLT                     r11=0;无马鞍点。
```

我们给读者留下这样的问题，就是来想出一个还要更好的解法；其中“阶段1”记录所有可能的这样的行，就是它们都是在“阶段2”中寻找的行的候选者。没有必要来寻找所有的行，仅仅是这样的 i_0 才是必要的；对于它存在着 j_0 使得 $a_{i_0 j_0}=C(j_0)=\min_j C(j)$。通常这仅只是一行。

在具有从{0，1，2，3，4}中随机地选择之元素的某些试验运行中，解法1近乎需要730 u 时间来运行，而解法2大约要花540 u。给定一个全0的矩阵，解法1用137 u 时间寻找出马鞍点，而解法2用524 u。

11. 假定一个 $m \times n$ 矩阵。(a) 由习题10答案中的定理，一个矩阵的所有马鞍点有相同的值。所以（在我们关于诸元素不同的假定下）至多有一个马鞍点。由对称性，所求的概率是：mn 乘以 a_{11} 为一马鞍点的概率。这后者为 $1/(mn)!$ 乘以对于 $a_{12}>a_{11},\cdots,a_{1n}>a_{11}$，$a_{11}>a_{21},\cdots,a_{11}>a_{m1}$ 的排列数；这是 $1/(m+n-1)!$ 乘以 $m+n-1$ 个元素的排

列的个数，其中头一个大于其次的（$m-1$）元素，并小于剩下的（$n-1$）个，即是$(m-1)!\ (n-1)!$。因此答案是

$$mn(m-1)!\ (n-1)!\ /(m+n-1)!\ =(m+n)\Big/\binom{m+n}{n}$$

在我们的情况下，这是$17\Big/\binom{17}{8}$，仅仅是1430中的一个机会。（b）在第二个假定之下，由于可以有多个马鞍点，因此必须使用完全不同的方法；事实上，或者是整个一行或者是整个一列，必须完全由马鞍点组成。这个概率等于：有一个马鞍点具有值0的概率，加上有一个马鞍点具有值1的概率。前者是至少有一列0的概率；后者是至少有一行1的概率。答案是$(1-(1-2^{-m})^n)+(1-(1-2^{-n})^m)$；在我们的情况下，是92474479623403623 1/18446744073709551616，大约是1/19.9分之一。一个近似的答案为$n2^{-m}+m2^{-n}$。

13.

```
        *CRYPTANALYST PROBLEM (CLASSIFIED)
UNIT    EQU   19          输入设备号
SIZE    EQU   14          输入区段大小
TABLE   EQU   1000        计数表
        ORIG  TABLE           (除了对于
        CON   -1              空格和星
        ORIG  TABLE+46        号的条款外，
        CON   -1              开始为0)
        ORIG  2000
BUF1    ORIG  *+SIZE      头一个缓冲区区域
        CON   -1          缓冲区末端的"哨兵"
        CON   *+1         每一个缓冲区参照另一个
BUF2    ORIG  *+SIZE      第二个缓冲区
        CON   -1          "哨兵"
        CON   BUF1        访问第一个缓冲区
BEGIN   IN    BUF1(UNIT)  输入第一区段
        ENT6  BUF2
1H      IN    0,6(UNIT)   输入下一区段
        LD6   SIZE+1,6    在进行这个输入期间，准备
        ENT5  0,6             处理以前的输入
2H      LDX   0,5         5个字符→rX
        JXN   1B          区段结尾?
1H      SLAX  1
        STA   *+1(2:2)    下一字符→r1
        ENT1  0
        LDA   TABLE,1
        JAN   2F          是特殊字符?
        INCA  1           更新表格条款
        STA   TABLE,1
1H      JXP   1B          rX中的任何非空格?
        INC5  1
        JMP   2B
```

主循环，应尽可能快地运行

```
2H      J1Z     1B          跳过一空格
        ENT1    1           发现星号
2H      LDA     TABLE,1
        JANP    1F          跳过0答案
        CHAR                转换成十进制
        JBUS    *(19)       等候打字机完成
        ST1     CHAR(1:1)
        STA     CHAR(4:5)
        STX     FREQ
        OUT     ANS(19)     打出一个答案
1H      CMP1    =60=
        INC1    1           直到累计60个
        JL      2B              字符代码
        HLT
ANS     ALF                 输出缓冲区
        ALF
CHAR    ALF     C    NN
FREQ    ALF     NNNNN
        ORIG    ANS+14      缓冲区的剩下部分为空白
        END     BEGIN       文字常数=60=在这里。 ■
```

对于这个问题，输出缓冲不是所希望的，因为对于每一行的输出，它至多可以节省7 u的时间，而这相对于输出这一行本身所需要的时间，乃是非常不足道的。关于字母频率的信息，见查尔斯·佩·伯恩（Charles P. Bourne）和唐纳德·弗·福特(Donald F. Ford)，《英文词中字母的统计研究》(A study of the statistics of letters in English words)，《信息和控制》(Information and Control) 4 (1961)，48～67。

14. 为使得这个问题更值得注意，下列的解法使用尽可能多的技巧，为的是减少执行时间。读者还能否再挤出点时间来呢？

```
*DATE OF EASTER
EASTER  STJ     EASTX
        STX     Y
        ENTA    0           E1.
        DIV     =19=
        INCX    1
        STX     G(0:2)
        LDA     Y           E2.
        MUL     =1//100+1=      (见下
        INCA    1               面)
        STA     C(1:4)
        MUL     =3//4+1=    E3.
        STA     XPLUS12(0:2)
        LDA     =8(1:1)=
        MUL     C           rA=8C
```

```
        INCA    680             680 = 5 + 27·25
        MUL     =1//25+1=       rA = Z + 32
XPLUS12 DECA    0
        STA     1F(0:2)         Z + 20 - X
        LDA     Y               E4.
        MUL     =1//4+1=
        ADD     Y
        SUB     XPLUS12(0:2)
        INCA    5
        STA     DPLUS12(0:2)
G       ENTA    0               F5.
        MUL     =11=
1H      INCX    0
        DIV     =30=
        JXNN    *+2             见习题 15
        INCX    30
        CMPX    =24=
        JE      1F
        CMPX    =25=
        JNE     2F
        LDA     G(0:2)
        DECA    11
        JANP    2F
1H      INCX    1
2H      DECX    20              E6. (24-N)
        CMPX    =3=
        JLE     *+2
        DECX    30
        STX     N(0:2)
        LDAN    N(0:2)          E7.
        ADD     DPLUS3
        SRAX    5
        DIV     =7=
        SLAX    5
N       INCA    0               31-N
        JANN    1F              E8.
        CHAR
        LDA     APRIL
        JMP     2F
1H      DECA    31
        CHAR
        LDA     MARCH
2H      JBUS    *(18)
        STA     MONTH
```

```
                STX     DAY(1:2)
                LDA     Y
                CHAR
                STX     YEAR
                OUT     ANS(18)          打印
EASTX           JMP     *
MARCH           ALF     MARCH
APRIL           ALF     APRIL
ANS             ALF
DAY             ALF     DD
MONTH           ALF     MMMMM
                ALF     ,
YEAR            ALF     YYYYY
C               CON     0                C乘以字节大小
                ORIG    *+20
BEGIN           LDX     =1950=           "驱动"
                JMP     EASTER             程序,
                LDX     Y                  使用
                INCX    1                  上述的
                CMPX    =2000=             子程序
                JLE     EASTER+1
                HLT
                END     BEGIN
```

在许多场合，从除法到乘法之变化的严格论证，可以以下列事实为基础，即是，在rA中的数并不太大（参照第12章）。这个程序对所有的字节大小进行工作。

16. 以比例数 $R_n=10^n r_n$ 来进行工作。$R_n(1/m)=R$ 当且仅当 $10^n\Big/\left(R+\frac{1}{2}\right)<m\leqslant 10^n\Big/\left(R-\frac{1}{2}\right)$；于是我们求得 $m_k=\lfloor 2\cdot 10^n/(2R-1)\rfloor$。

```
*SUM OF     HARMONIC    SERIES
BUF             ORIG    *+24
START           ENT2    0
                ENT1    3               〔5-n〕
                ENTA    20
OUTER           MUL     =10=
                STX     CONST           〔2·10^n〕
                DIV     =2=
                ENTX    2
                JMP     1F
INNER           STA     R
                ADD     R
                DECA    1
                STA     TEMP            〔2R-1〕
                LDX     CONST
```

```
          ENTA    0
          DIV     TEMP
          INCA    1
          STA     TEMP        [m_A + 1]
          SUB     M
          MUL     R
          SLAX    5
          ADD     S
          LDX     TEMP
3H        STA     S           部分和
          STX     M
          LDA     M
          ADD     M
          STA     TEMP
          LDA     CONST
          ADD     M           计算
          SRAX    5           ⌊(2·10^n + m)/2m⌋
          DIV     TEMP
          JAP     INNER       R > 0?
          LDA     S           答案
          CHAR
          SLAX    0, 1        适当的格式
          SLA     1
          INCA    40          小数点
          STA     BUF, 2
          STX     BUF+1, 2
          INC2    3
          DEC1    1
          LDA     CONST
          J1NN    OUTER
          OUT     BUF(18)
          HLT
          END     START
```

■

在 65595 u 加上输出时间，输出是

0006.16　　0008.449　　0010.7509　　0013.05362

```
18.  FAREY   STJ     9F          假设 rI1 包含 n，其中 n ≥ 1
             STZ     X           x_0 ← 0
             ENTX    1
             STX     Y           y_0 ← 1
             STX     X+1         x_1 ← 1
             STJ     Y+1         y_1 ← n
             ENT2    2           k ← 2
3H           LDX     Y-2, 2
             INCX    0, 1
```

```
        ENTA    0
        DIV     Y-1,2
        STA     TEMP            ⌊(y_{k-2}+x)/y_{k-1}⌋
        MUL     Y-1,2
        SLAX    5
        SUB     Y-2,2
        STA     Y,2             y_k
        LDA     TEMP
        MUL     X-1,2
        SLAX    5
        SUB     X-2,2
        STA     X,2             x_k
        CMPA    Y,2             测试是否 x_k<y_k。
        INC2    1               k←k+1
        JL      1B              若然，继续
9H      JMP     *               由子程序出口。 ∎
```

19. (a) 归纳法。(b) 设 $k\geqslant 0$ 且设 (*) 的右边以 X, Y 表示之。由部分(a), 我们有 $\gcd(X, Y)=1$ 而且 $X/Y>x_{k+1}/y_{k+1}$。所以，如果

$$X/Y\neq x_{k+2}/y_{k+2}$$

则由定义，我们有 $X/Y>x_{k+2}/y_{k+2}$。但这意味着

$$\frac{1}{Yy_{k+2}}=\frac{Xy_{k+2}-Yx_{k+1}}{Yy_{k+1}}=\frac{X}{Y}-\frac{x_{k+1}}{y_{k+1}}=\left(\frac{X}{Y}-\frac{x_{k+2}}{y_{k+2}}\right)+\left(\frac{x_{k+2}}{y_{k+2}}-\frac{x_{k+1}}{y_{k+1}}\right)\geqslant\frac{1}{Yy_{k+2}}+\frac{1}{y_{k+1}y_{k+2}}$$

$$=\frac{Y+y_{k+1}}{Yy_{k+1}y_{k+2}}>\frac{n}{Yy_{k+1}y_{k+2}}\geqslant\frac{1}{Yy_{k+1}}$$

对于法雷级数的更多的有趣性质，以及它的历史，请见戈·哈·哈迪和爱·梅·赖特,《数论》，牛津，第3章。

20.

```
* TRAFFIC SIGNAL PROBLEM
BSIZE   EQU     1(4:4)          字节大小
2BSIZE  EQU     2(4:4)          两倍字节大小
DELAY   STJ     1F              如果 rA 含有 n，则
        DECA    6                   这个子程序恰好
        DECA    2                   等候 max(n, 7) u，
        JAP     *-1                 不包括转到
        JAN     *+2                 这个子程序
        NOP                         的转移
1H      JMP     *
FLASH   STJ     2F          4   这个子程序使
        ENT2    8           5       相应的
        LDA     =49991=     7       DON'T WALK
1H      JMP     DELAY       8       灯闪亮
        DECX    0,1         9   使灯闭关
```

```
          LDA     =49996=     2
          JMP     DELAY       3
          INCX    0, 1        4   "DON'T WALK"
          DEC2    1           1
          J2Z     1F          2   重复八次
          LDA     =49993=     4
          JMP     1B          5
1H        LDA     =399992=        在出口之后置黄色灯 2 u
          JMP     DELAY       5
2H        JMP     *           6
WAIT      JNOV    *               Del Mar绿灯直到顺利通过
TRIP      INCX    BSIZE           Del Mar上的 DON'T WALK(不许通过)
          ENT1    2BSIZE
          JMP     FLASH
          LDX     BAMBER          在大街上黄灯
          LDA     =799995=
          JMP     DELAY       3   等候 8 秒
          LDX     AGREEN      5   大路上绿灯
          LDA     =799996=
          JMP     DELAY           等候 8 秒
          INCX    1               Berk'ly 上 DON'T WALK
          ENT1    2
          JMP     FLASH           进行闪亮循环
          LDX     AAMBER          大路上黄灯
          JOV     *+1             消去多余的通过
          LDA     =499994=
          JMP     DELAY           等候 5 秒
BEGIN     LDX     BGREEN          大街上绿灯
          LDA     =1799994=
          JMP     DELAY           等候至少
          JMP     WAIT                18秒
AGREEN    ALF     CABA            大路上绿灯
AAMBER    ALF     CBBB            大路上黄灯
BGREEN    ALF     ACAB            大街上绿灯
BAMBER    ALF     BCBB            大街上黄灯
          END     BEGIN
```

22.

```
*JOSEPHUS PROBLEM
N     EQU   24
M     EQU   11
X     ORIG  *+N
OH    ENT1  N-1          1          置每个小格成为
      STZ   X+N-1        1              序列中下一个
      ST1   X-1,1        N-1            人的号数
      DEC1  1            N-1
```

```
        J1P     * - 2          N - 1
        ENTA    1                1          (现在 rI1 = 0)
1H      ENT2    M - 2          N - 1        (假定M > 2)
        LD1     X, 1    (M - 2)(N - 1)      围绕圆圈进行
        DEC2    1       (M - 2)(N - 1)          计数
        J2P     * - 2   (M - 2)(N - 1)
        LD2     X, 1           N - 1        rI1≡幸运的人
        LD3     X, 2           N - 1        rI2≡不幸运的人
        CHAR                   N - 1        rI3≡下一个人
        STX     X, 2(4:5)      N - 1        存执行数
        NUM                    N - 1
        INCA    1              N - 1
        ST3     X, 1           N - 1        从圆圈取人
        ENT1    0, 3           N - 1
        CMPA    =N=            N - 1
        JL      1B             N - 1
        CHAR                     1          一人剩下
        STX     X, 1(4:5)        1          (他太危险了)
        OUT     X(18)            1          打印答案
        HLT                      1
        END     0B                          ∎
```

最后一人在位置 15 处。输出之前的总时间是 $(4(N-1)(M+3)+7)u$。可能有若干改进，例如，丹·英戈尔斯（D. Ingalls)建议要有三个字的代码组“DEC2 1; J2P NEXT; JMP OUT”，其中 OUT 修改了 NEXT 场，以便删去一组。

1.3.3 小节

1. (124)(365)。

2. $a \leftrightarrow c$，$c \leftrightarrow f$；$b \leftrightarrow d$。推广到任意排列是显然的。

3. $\begin{pmatrix} a & b & c & d & e & f \\ d & b & f & c & a & e \end{pmatrix}$。

4. $(a\ d\ c\ f\ e)$。

5. 12.（参照习题 20.）

6. 总的时间，对于每个空白字，其具有居前的非空白字“(”者，增加 $4u$，对于每个空白字，其具有居前的非空白字名称者，加 $5u$。初始的空格和诸循环之间的空格不影响执行的时间。空格的位置对程序 B 毫无影响。

7. $X=2$，$Y=29$，$M=5$，$N=7$，$U=3$，$V=1$。由等式 (18)，总时间为 $2165u$。

8. 是；那样，我们将记住排列的逆，即是，x_i 变成 x_j 当且仅当 $T(j)=i$（最后的循环形式于是将从右到左地由 T 表构造出来）。

9. 否。例如，给定 (6) 作为输入，程序 A 将产生“(ADG)(CEB)”作为输出，而程序 B 则将产生“(CEB)(DGA)”。这些答案是等价的，但并不同一，这是由于循环记号的不

唯一性所致。对于一循环所选择的头一个元素，就程序A或B而言，分别是（a）最左地可用的名称，或（b）从右到左所遇到的最后一个可用的不同名称。

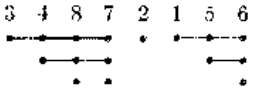
10.（1）克希霍夫定律推出 $A=1+C-D$；$B=A+J+P-1$；$C=B-(P-L)$；$E=D-L$；$G=E$；$Q=Z$；$W=S$。（2）解释：B=输入的字数$=16X-1$；C=非空白字的个数$=V$；$D=C-M$；$F=D-M$；F=查名字表时进行比较的次数；$H=N$；$K=M$；$Q=N$；$R=U$；$S=R-V$；$T=N-V$，由于名称彼此标记。（3）加起来，我们有$(4F+16Y+80X+21N-19M-9U-16V)u$，由于$F$肯定小于$16NX$，这比程序$A$要稍微好些。在所述情况下的时间为$983u$。

11. “反射之”；例如，$(acf)(bd)$的逆是$(db)(fca)$。

12.（a）在单元$L+mn-1$中的值，通过转置是固定的，所以我们可以略去不予考虑。否则，如果$x=n(i-1)+(j-1)<mn-1$，则$L+x$中的值应该变到单元$L+(mx)\bmod N=L+(mn(i-1)+m(j-1))\bmod N=L+m(j-1)+(i-1)$，由于$mn\equiv 1\ (\text{modulo}N)$和$0\leqslant m(j-1)+(i-1)<N$。（b）如果在每一存贮单元中有一位可利用（例如，符号位，或者一个浮点值的最低有效位），则我们就能利用象算法I那样的一个算法，随着我们移动诸元素而“标记”它们。这样，（a）置$x\leftarrow N-1$；（b）如果CONTENTS$(L+x)$已加标记，则转到（f），否则对它加标记，而且置$y\leftarrow$CONTENTS$(L+x)$；（c）置$x\leftarrow(mx)\bmod N$；（d）交换$y\leftrightarrow$CONTENTS$(L+x)$；（e）如果对y未加标记，就对它加标记并返回到（c）；（f）x减1，而且如果$x>0$则返回（b）。参考：马丁·弗·伯曼（Martin F. Berman），JACM 5 (1958)，383～384。如果无余地用于一个标记位，也许标记位可以记在一个辅助的表格中，或者否则，可以使用一张所有非单一元素的循环之代表的表：对于N的每个因子d，由于m与N互质，我们可以独立地转置那些其为d之倍数的元素，其含有x之循环的长度，当$\gcd(x,N)=d$时，是使得$m^r\equiv 1\ (\text{modulo}N/d)$的最小整数$r>0$。对于每个$d$，我们要来求$\varphi(N/d)/r$个代表，由这些循环的每一个找一个，某些数论的方法对于这一目的是可利用的，但它们并不是足够简单而真正地令人满意的。参考戈登·帕尔（Gordon Pall）和埃思尔·塞登（Esther Seiden），Math. Comp. 14(1960)，189～192。最后，有一个类似于算法J的方法，它更慢，但不需要辅助的存储。参考：P. F. 温德利（P. F. Windley），《计算机杂志》(Comp. J.) 2 (1959)，47～48；唐·欧·克努特，《国际信息处理联合会会议会报》(Proc. IFIP Congress)(1971)，1，19～27。

13. 用归纳法证明，在步骤$J2$的开始，$X[i]=+j$，当且仅当，$j>m$而且在π之下j变到i；$X[i]=-j$，当且仅当，在π^{k+1}之下i变到j，其中k是使得π^k把i变成一个$\leqslant m$的数之最小非负整数。

14. 以典型循环形式写出已给的排列之逆，并脱去圈括弧，数量$A-N$是大于一给定的元素的并直接在它右边的连续元素的个数之和。例如，如果原来的排列是(165)(3784)，则逆的典型形式是（3487)(2)(156)；建立阵列



而且数量 A 是“圆点”的个数，即 16。在第 k 个元素下边的圆点的个数，是在头 k 个元素当中自右到左的极小值的个数（即是，如上，在 7 下边有 3 个圆点，因为在 3487 中有 3 个自右到左的极小值）。因此，平均值是 $H_1+H_2+\cdots+H_n=(n+1)H_n-n$。

15. 如果线性表示的头一个字符是 1，则典型表示的最后字符是 1。如果线性表示的头一个字符是 $m>1$，则“$\cdots 1m\cdots$”出现在典型表示中。所以仅有的解是单个对象的排列。

16. 1324, 4231, 3214, 4213, 2143, 3412, 2413, 1243, 3421, 1324, $\cdots$。

17. (a) 循环为一个 m 循环的概率，是 $n!/m$ 除以 $n!H_n$，所以 $p_m=1/mH_n$。平均长度是 $p_1+2p_2+3p_3+\cdots=\Sigma_{1\leqslant m\leqslant n}(m/mH_n)=n/H_n$。(b) 由于 m 循环的总数是 $n!/m$，在 m 循环中元素的出现总数是 $n!$。由于对称性，每个元素与其它任何元素同样经常地出现。所以在 m 循环中 k 出现 $n!/n$ 次。因此，在这种情况下，$p_m=1/n$ 对于所有的 k 和 m；平均值是

$$\Sigma_{1\leqslant m\leqslant n}m/n=(n+1)/2$$

18. 见习题 22(e)。

19. $|P_{n0}-n!/e|=1/(n+1)!-1/(n+2)!+\cdots$，一个数值递降的交错级数，它小于 $1/(n+1)!\leqslant\frac{1}{2}-$。

20. 每个 m 循环可以以 m 种方式独立地写成；全部有 $\alpha_1+\alpha_2+\cdots$ 个循环，可以排列在另一个当中；所以答案是

$$(\alpha_1+\alpha_2+\cdots)!1^{\alpha_1}\cdot 2^{\alpha_2}\cdot 3^{\alpha_3}\cdots$$

21. 如果 $n=\alpha_1+2\alpha_2+\cdots$，则为 $1/(\alpha_1!1^{\alpha_1}\alpha_2!2^{\alpha_2}\cdots)$；否则为 0。

证明：在一行中，写出 α_1 个 1 循环，α_2 个 2 循环，等等，具有空的位置；例如，如果 $\alpha_1=1$，$\alpha_2=2$，$\alpha_3=\alpha_4=\cdots=0$，则我们将有“(−)(−−)(−−)”。在所有 $n!$ 种可能的方式下，填充空的位置；我们得到所希望形式的每个排列恰好 $\alpha_1!1^{\alpha_1}\alpha_2!2^{\alpha_2}\cdots$ 次。

22. (a) 如果 $k_1+2k_2+\cdots=n$，则 (ii) 中的概率为 $\Pi_{j\geqslant 0}f(w,\ j,\ k_j)$，假定它等于 $(1-w)w^n/k_1!1^{k_1}k_2!2^{k_2}\cdots$；因之

$$\Big(\prod_{j\geqslant 0}f(w,\ j,\ k_j)\Big)^{-1}\prod_{j\geqslant 0}f(w,\ j,\ k_j+\delta_{jm})$$

$$=\frac{f(w,\ m,\ k_m+1)}{f(w,\ m,\ k_m)}=\frac{w^m}{m(k_m+1)}$$

因此由归纳法

$$f(w,\ m,\ k)=\frac{1}{k!}\Big(\frac{w^m}{m}\Big)^k f(w,\ m,\ 0)$$

现在条件 (i) 意味着

$$f(w,\ m,\ k)=\frac{1}{k!}\Big(\frac{w^m}{m}\Big)^k e^{-w^m/m}$$

〔注意：因此以一“泊松分布”来选择 α_m，见习题 1.2.10-16。〕

$$(b)\quad \sum_{\substack{k_1+2k_2+\cdots=n\\ k_1,k_2,\cdots\geqslant 0}}\Big(\prod_{j\geqslant 0}f(w,\ j,\ k_j)\Big)=(1-w)w^n\sum_{\substack{k_1+2k_2+\cdots=n\\ k_1,k_2,\cdots\geqslant 0}}P(n;\ k_1,k_2,\cdots)$$

$$=(1+w)w^n$$

因此 $\alpha_1+2\alpha_2+\cdots\leqslant n$ 的概率为 $(1-w)(1+w+\cdots+w^n)=1-w^{n+1}$。

（c）ϕ的平均值为

$$\sum_{n\geqslant 0}\left(\sum_{k_1+2k_2+\cdots=n}\phi(k_1,\ k_2,\cdots)\Pr(\alpha_1=k_1,\ \alpha_2=k_2,\cdots)\right)$$

$$=(1-w)\sum_{n\geqslant 0}w^n\left(\sum_{k_1+2k_2+\cdots=n}\phi(k_1,\ k_2,\cdots)/k_1!1^{k_1}k_2!2^{k_2}\cdots\right)$$

（d）设$\phi(\alpha_1,\ \alpha_2,\cdots)=\alpha_2+\alpha_4+\alpha_6+\cdots$。线性组合$\phi$的平均值是$\alpha_2$，$\alpha_4$，$\alpha_6$，…的平均值之和；$\alpha_m$的平均值是

$$\sum_{k\geqslant 0}kf(w,\ m,\ k)=\sum_{k\geqslant 1}\frac{1}{(k-1)!}\left(\frac{w^m}{m}\right)^k e^{-w^m/m}=\frac{w^m}{m}$$

因此ϕ的平均值是

$$\frac{w^2}{2}+\frac{w^4}{4}+\cdots=(1-w)\left(\frac{1}{2}w^2+\frac{1}{2}w^3+\left(\frac{1}{2}+\frac{1}{4}\right)w^4+\cdots\right)$$

所求的答案是

$$\sum_{\substack{0\leqslant k\leqslant 2n\\ k\text{为偶}}}\frac{1}{k!}=\frac{1}{2}+H_{\lfloor n/2\rfloor}$$

（e）设z为一实数；设$\phi(\alpha_1,\ \alpha_2,\cdots)=z^{\alpha_m}$。$\phi$的平均值是

$$\sum_{k\geqslant 0}f(w,\ m,\ k)z^k=\sum_{k\geqslant 0}\frac{1}{k!}\left(\frac{w^m z}{m}\right)^k e^{-w^m/m}=e^{w^m(z-1)/m}$$

$$=\sum_{j\geqslant 0}\frac{w^{mj}}{j!}\left(\frac{z-1}{m}\right)^j=(1-w)\sum_{n\geqslant 0}w^n\left(\sum_{0\leqslant j\leqslant n/m}\frac{1}{j!}\left(\frac{z-1}{m}\right)^j\right)$$

$$=(1-w)\sum_{n\geqslant 0}w^nG_{nm}(z)$$

因此

$$G_{nm}(z)=\sum_{0\leqslant j\leqslant n/m}\frac{1}{j!}\left(\frac{z-1}{m}\right)^j;\quad p_{nmk}=\frac{1}{m^k k!}\sum_{0\leqslant j\leqslant n/m-k}\frac{(-1/m)^j}{j!}$$

（min 0，ave $1/m$，max n/m，dev $\sqrt{1/m}$），当$n>2m$时。

23. 常数λ是$\int_0^\infty\exp(-t-E_1(t))dt$，其中$E_1(x)=\int_0^x e^t dt/t$。见《美国数学协会会报》(Transaction of the American Math. Society)121(1966)，340～357；在这篇论文中还证明了许多其它的结果，特别是最短循环的平均长度近似于$e^{-\gamma}\ln n$之渐近表示的进一步的项尚犹未知。威廉·查·米切尔（William C. Mitchell）已经计算出λ的一个高精度的值=.62432 99885 43550 87099 29363 83100 83724 41796+〔Math. Comp.(1968)，411～415〕；在λ与经典的数学常数之间的关系，尚犹未知。

24. 见唐·欧·克努特，《国际信息处理联合会会议会报》(Proc. IFIP Congress)(1971)，1，19～27。

25. 一个通过对N用归纳法的证明，是以下列事实为基础的：当第N个元素为集合中之s个的一个成员时，它恰巧为和数提供

$$\binom{s}{0}-\binom{s}{1}+\binom{s}{2}-\cdots=(1-1)^s=\delta_{s0}$$

另一个通过对M用归纳法的证明，是以下列事实为基础的：在S_M中但不在$S_1\cup\cdots\cup S_{M-1}$中的元素之个数为

$$|S_M|-\sum_{1\leq j<M}|S_j\cap S_M|+\sum_{1\leq j<k<M}|S_j\cap S_k\cap S_M|-\cdots$$

26. 设$N_0=N$，并设

$$N_k=\sum_{1\leq j_1<\cdots<j_k\leq M}|S_{j_1}\cap\cdots\cap S_{j_k}|$$

则所求的公式为

$$N_r-\binom{r+1}{r}N_{r+1}+\binom{r+2}{r}N_{r+2}-\cdots$$

这可以由包含和排斥原理本身来证明，或者通过使用公式

$$\binom{r}{r}\binom{s}{r}-\binom{r+1}{r}\binom{s}{r+1}+\cdots=\binom{s}{r}\binom{s-r}{0}-\binom{s}{r}\binom{s-r}{1}\cdots\cdots=\delta_{sr}$$

如在习题25中那样。

27. 设S_j为在所述范围内m_j的诸倍数，且设$N=am_1\cdots m_t$。则

$$|S_j\cap S_k|=N/m_jm_k，\text{等等}$$

所以答案是

$$N-N\sum_{1\leq j\leq t}\frac{1}{m_j}+N\sum_{1\leq j<k\leq t}\frac{1}{m_jm_k}-\cdots=N\left(1-\frac{1}{m_1}\right)\cdots\left(1-\frac{1}{m_t}\right)$$

如果我们设$m_1,\cdots,m_t$为整除N的质数，则这也解决了习题1.2.4-30。

29. 当通过一个人时，就给他指定一个新的号数（由$n+1$开始）。于是，第k个被执行的人的号数是$2k$，而且对于号数为j，$j>n$的人，以前的号数为$(2j)\bmod(2n+1)$。

1.4.1 小节

1. 调用序列：JMP MAXN；或JMP MAX100，如果$n=100$。
   进入条件：对于MAXN的入口，rI3$=n$；假定$n\geqslant 1$。
   出口条件：与（4）相同。

2. MAX50 STJ EXIT
   ENT3 50
   JMP 2F

3. 进入条件：$n=$rI1如果rI1>0；否则$n=1$。
   出口条件：rA和rI2如同（4）中那样；rI1不变；rI3$=\min(0,$ rI1$)$；rJ$=$EXIT$+1$；CI不变如果$n=1$的话，否则根据极大值大于$X[1]$，等于$X[1]$且rI2>1，或等于$X[1]$而rI2$=1$，CI为大于，等于，或

小于。

（对于（9）的类似的习题，当然稍微要复杂些。）

```
4.    SMAX1    ENT1    1        r = 1
      SMAX     STJ     EXIT     一般的 r
               JMP     2F       象以前一样继续
      ...
               DEC3    0, 1     减小 r
               J>P     1B
      EXIT     JMP     *        出口
```

调用序列：JMP SMAX；或 JMP SMAX1 如果 $r=1$。

进入条件：rI3 $=n$，假定为正；对于 SMAX 的入口，rI1 $=r$，假定为正。

出口条件：$rA=\max_{0\le k<n,r}\mathrm{CONTENTS}(X+n-k_r)=\mathrm{CONTENTS}(X+\mathrm{rI2})$，$\mathrm{rI3}=-((-n)\bmod r)$。

5. 可以使用任何其它的寄存器。例如，

调用序列：ENTA　* + 2

　　JMP　　MAX100

进入条件：无。

出口条件：与（4）中相同。

除了头一条指令将为

"MAX100 STA EXIT(0:2)"

之外，代码与（4）中相同。

6.（由乔尔·戈尔堡（Joel Goldberg）和罗杰·M·艾伦斯（Roger M. Aarons）给出的解答。）

```
MOVE    STJ     3F
        STA     4F
        ST2     5F(0:2)
        LD2     3F(0:2)
        LDA     0, 2(0:3)
        STA     *+1(0:3)
        ENTA    *
        LD2N    0, 2(4:4)
        J2N     1F
        DECA    0, 2
        STA     2F(0:2)
        DEC1    0, 2
        ST1     6F(0:2)
2H      LDA     *, 2
6H      STA     *, 2
        INC2    1
        J2N     2B
1H      LDA     4F
```

```
5H      EXT2    *
3H      JMP     *
4H      CON     0
```

1.4.2 小节

1. 如果一个共行程序仅仅调用另一个一次，则它只不过是一个子程序；所以我们需要一个这样的应用，其中每个共行程序调用另一个至少在不同的两处。甚至于，通常容易设置某种类型的开关或者使用数据的某种性质，使得当进入到一个共行程序之内的一个固定的位置时，有可能转到两个所希望的位置之一——因而再次地，将需要的只不过是一个子程序。当它们之间访问的次数变得更大时，共行程序相应地变得更为有用。

2. 由 IN 找到的头一个字符将失去。

3. 几乎是真的，因为 IN 内的“CMPA＝10＝”就是这程序之唯一的比较指令，而且因为“．”的代码是40。但是如果最后的句号以一个重复的数字在前边，则测试将被忽略过去矣！（注意：最有效的程序也许是撤消行 40，44 和 48，而且在行 26 与 27 之间插入“CMPA PERIOD”。然而如果穿插于诸共行程序使用了比较指示器的状态时，则它必须作为部分共行程序的特征记录于程序的文件编制中）。

4.（a）在 IBM650 上，使用 SOAP 汇编语言，我们将有调用序列“LDD A”和“LDD B”；而且有链接“A STD BX AX”和“B STD AX BX”（用两条链接指令于内存更好些）。（b）在 IBM709 上，使用普通的汇编语言，调用序列将是“TSX A，4”或“TSX B，4”；链接指令将是

```
A     SXA   BX, 4          B     SXA   AX, 4
AX    AXT   1-A1, 4        BX    AXT   1-B1, 4
      TRA   1, 4                 TRA   1, 4
```

（c）在 CDC1604，调用序列将是“返回转移”（SLJ 4）到 A 或 B，而且链接比如说将是

```
B：SLJ A1；ALS 0
A：SLJ B1；SLJ B。
```

（d）大多数其它的机器类似于这三者之一。例如，系统/360将类似于709，或者我们将使用 BALR r，r 于短的共行程序中。

5. “STA HOLDAIN；LDA HOLDAOUT”在 OUT 与 OUTX 之间，以及“STA HOLDAOUT；LDA HOLDAIN”在 IN 与 INX 之间。

6. 在 A 之内写“JMP AB”以激活 B，“JMP AC”以激活 C。类似地在 B 和 C 内将使用单元 BA，BC，CA 和 CB。链接是

```
AB    STJ   AX      BC    STJ   BX      CA    STJ   CX
BX    JMP   B1      CX    JMP   C1      AX    JMP   A1
CB    STJ   CX      AC    STJ   AX      BA    STJ   BX
      JMP   BX            JMP   CX            JMP   AX
```

（注意：对于 n 个共行程序，将需要 $2(n-1)n$ 个单元以进行链接。如果 n 很大，当然可以使用一个为进行链接的“中心化程序”；一个使用 $3n+2$ 个单元的方法将不难想出。

但事实上，上边更快的方法仅需要 $2m$ 个单元，其中 m 是使得共行程序 i 转到共行程序 j 的对偶（i，j）的个数。当有许多共行程序，每一个都独立地转移到另一个时，我们通常处于这样一种状况，即其中控制序列是在外部的影响之下，如同 2.2.5 小节所讨论的那样)。

1.4.3.1 小节

1. 它仅被使用两次，两次都直接地接着调用 MEMORY，所以使它成为对 MEMORY 子程序的一个特殊的入口，而且使它置 $-R$ 于 rI2 中，将会稍微更有效些。

2.
```
SHIFT   J5N     ADDRERROR
        DEC3    5
        J3P     FERROR
        LDA     AREG
        LDX     XREG
        LD1     1F, 3(4:5)
        ST1     2F(4:5)
        J5Z     CYCLE
2H      SLA     1
        DEC5    1
        J5P     2B
        JMP     STOREAX
        SLA     1
        SRA     1
        SLAX    1
        SRAX    1
        SLC     1
1H      SRC     1
```

3.
```
MOVE    J3Z     CYCLE
        JMP     MEMORY
        SRAX    5
        LD1     I1REG
        LDA     SIGN1
        JAP     *+3
        J1NZ    MEMERROR
        STZ     SIGN1(0:0)
        CMP1    =BEGIN=
        JGE     MEMERROR
        STX     0,1
        LDA     CLOCK
        INCA    2
        STA     CLOCK
        INC1    1
        ST1     I1REG
        INC5    1
```

```
        DEC3    1
        JMP     MOVE
```

4. 仅仅在行 03 与 04 之间插入 "IN 0(16)" 和"JBUS *(16)"(当然，在另一台计算机上这将是相当不同的，因为将有必要转换到 MIX 的字符代码)。

5. 中央控制时间为 34u，如果需要变址加上 15u；GETV 子程序花去 52u，如果 $L\neq 0$ 则加上 5u；对于 LDA 或 LDX，进行真正的装入的额外时间为 11u，对于 LDi 为 13u，对于 ENTA 或 ENTX 为 21u，对于 ENTi 为 23u(如果M＝0，则对于后两个时间加上 2u)。总加起来，对于 LDA 我们有 97u 的总时间，而对于 ENTA 有 55u，对于变址加上 15u，而在某些其它的情况下加上 5u 或 2u。看起来，在这种情况下的模拟引起大约为 50:1 的速度之比（一个含有 178u 模拟时间的测试运行之结果，需要 8122u 之实际时间，比例为 47:1)。

7. IN 或 OUT 之执行，把一个与适当的输入设备相关联的变量，置成为希望进行传输的时间。"CYCLE" 控制程序在每个循环询问这些变量，来看看 CLOCK 是否已经超过它们中的每一个（或两个)；如是这样，就进行传输而且把变量置为"无限大"(当在这种方式下，有多于两个输入/输出设备必须加以处理时，则将有这样多的变量——因而把它们记存在一个使用链接的内存技术的分类的表中，将更好些；见 2.2.5 小节)。当模拟HLT时，需要小心地来完成输入/输出。

8. ● 错的；r16 可等于 BEGIN，如果我们从以前的行"归于失败"时。但那样将出现一个 MEMERROR，试图来 STZ 到 TIME 中！由行 254，我们总是有 0≤r16≤BEGIN。

1.4.3.2 小节

1. 把行 48 和 49 改变成下面的序列：

```
LEAVE   STX   3F
        ST1   2F(0:2)
        LD1   JREG(0:2)
        LDA   -1,1
        LDX   1F
        STX   -1,1
        JMP   -1,1
1H      JMP   *+1
        STA   -1,1
2H      ENT1  *
        LDX   3F
        LDA   AREG
LEAVEX  JSJ   *
3H      CON   0
```

当然，操作符"JSJ"在这里是特别有决定性意义的。

2.
```
        *     TRACE  ROUTINE
              ORIG   *+99
        BUF   CON    0
        ..... .....  .....      行 02-04
```

● 本题答案在原书中被误排到1.4.3.2小节的答案中。——译者注

```
        ST1             I1REG(0:2)
.....           ... ... ... ...    行  05-07
PTR     ENT1            -100
        JBUS            *(0)
        STA             BUF+1, 1(0:2)
.....           ... ... ... ...    行  08-11
        STA             BUF+2, 1
.....           ... ... ... ...    行  12-13
        LDA             AREG
        STA             BUF+3, 1
        LDA             I1REG(0:2)
        STA             BUF+4, 1
        ST2             BUF+5, 1
        ST3             BUF+6, 1
        ST4             BUF+7, 1
        ST5             BUF+8, 1
        ST6             BUF+9, 1
        STX             BUF+10, 1
        LDA             JREG(0:2)
        STA             BUF+1, 1(4:5)
        ENTA            8
        JNOV            1F
        ADD             BIG
1H      JL              1F
        INCA            1
        JE              1F
        INCA            1
1H      STA             BUF+1, 1(3:3)
        INC1            10
        J1N             1F
        OUT             BUF-99(0)
        ENT1            -100
1H      ST1             PTR(0:2)
.....           ... ... ... ...    行  14-31
I1REG   ENT1            *
.....           ... ... ... ...    行  32-35
        ST1             I1REG(0:2)
.....           ... ... ... ...    行  36-48
        LD1             I1REG(0:2)
.....           ... ... ... ...    行  49-50
B4      EQU             1(1:1)
BIG     CON             B4-8, B4-1(1:1)
```

一个写出最后的缓冲区并重绕磁带 0 的进一步的程序，应该在实现了所有的跟踪之后来

调用。

3．磁带是更快的；而且在跟踪时编辑这个信息成字符，将消耗太多的空间。而且这带的内容可有选择地打印。

4．如象在习题 6 中所希望的那样的一个真正的跟踪，将不被得到，因为违犯了正文中提到的限制(a)。跟踪 CYCLE 的头一次尝试，将引起回到跟踪 ENTER＋1 的循环。

6．提示：记录一份已经为外部程序改变了的跟踪区域内每个内存单元之值的表格。

7．这个例行程序将扫描程序，直至找到头一个转移指令（或条件转移指令）；在修改了该指令以及紧接着的一条之后，它将恢复寄存器并允许程序一口气来执行所有它的指令，直到该点为止〔这项技术可能失败，如果程序修改它自己的转移指令的话。为了实用的目的，我们可以把这样一种做法抛开，只是 STJ 除外，我们对它大概应该以不论任何方式来独立处理之〕。

1.4.4 小节

1．(a) 否，输入操作可能尚未完成。(b) 否，输入操作可能只是进行得快一点点，而这是太冒险了。

2．
```
                ENT1    2000
                JBUS    *(6)
                MOVE    1000(50)
                MOVE    1050(50)
                OUT     2000(6)
```

3．
```
WORDOUT         STJ     1F
                INC5    1
                LDX     0,5
                JXZ     2F
                OUT     -100,5(V)
                LD5     0,5
                ENT1    0,5
                MOVE    -1,1(50)
                MOVE    -1,1(50)
                ST5     CURRENT(0:2)
2H              STA     0,5
1H              JMP     *
*BUFFER         AREAS
                CON     0
OUTBUF1         ORIG    *+100
                CON     *+2
                CON     0
OUTBUF2         ORIG    *+100
                CON     OUTBUF1
```

在这程序的开始处，给出指令“ENT5 OUTBUF1-1”。在这程序的末尾，设置

```
CURRENT         OUT     *(V)            写出最后区段
```

```
OUT     OUTBUF 1 (V)}   (任选的：在后面的输入缓冲的
IOC     0 (V)       }   情况下写一个额外的区段，并重绕带)
```

4．如果计算时间恰巧等于输入/输出时间（这是最适当的情况），则计算机与外部设备两者同时运行，当它们独自地运行时，需要一半那样长的时间。形式地设C是整个程序的运行时间，设T是所需总的输入/输出时间；则通过缓冲的最好可能运行时间是$\max(C, T)$，而没有缓冲的运行时间是$C+T$时；当然$\frac{1}{2}(C+T)\leqslant\max(C, T)\leqslant C+T$。然而，有些设备，它有导致花去额外时间的“关闭罚款”，如果对于该设备的访问之间，出现一个太长的时间间隔的话；在这样一种情况下，可能有比2:1更好的比例。

5．最好的比例为$(n+1):1$。

6．
```
{IN     INBUF 1 (U)}     或     {IN     INBUF2(U) }
{ENT6   INBUF 2 +99}           {ENT6   INBUF 1 +99}
```
(可能恰恰在必要的情况下，冠以IOC 0(U)以重绕带)。

7．一个方法是使用共行程序：

```
INBUF1    ORIG    *+100
          CON     *+1
INBUF2    ORIG    *+100
          CON     INBUF1
1H        LDA     INBUF2+100,6
          JMP     MAIN
          INC6    1
          J6N     1B
WORDIN1   IN      INBUF2(U)
          ENN6    100
2H        LDA     INBUF1+100,6
          JMP     MAIN
          INC6    1
          J6N     2B
          IN      INBUF1(U)
          ENN6    100
          JMP     1B
WORDIN    STJ     MAINX
WORDINX   JMP     WORDIN1
MAIN      STJ     WORDINX
MAINX     JMP     *
```

加上几条利用特殊情况的指令，将使得这个程序实际上比（4）更快。

8．在图23所示的时间里，两个红的缓冲区已经填以行的映象。而且正在打印的是由NEXTR指示的一个。同时，程序正在RELEASE与ASSIGN之间进行计算。当程序进行ASSIGN（指定）时，由NEXTG指示的绿色缓冲区变成黄色；NEXTG顺时针移动，而且程序开始来填黄色缓冲区。当完成输出操作时，NEXTR顺时针移动，刚刚被打印的缓冲区转为绿色，而且剩下的红色缓冲区被打印。最后，这个程序RELEASE（释放）黄色缓冲区而且它也为随继的打印做好准备。

9，10，11：

时间	动作（$N=1$）	动作（$N=2$）	动作（$N=4$）
0	ASSIGN(BUF1)	ASSIGN(BUF1)	ASSIGN (BUF1)
1000	RELEASE, OUT BUF1	RELEASE,OUT BUF1	RELEASE,OUT BUF1
2000	ASSIGN(等候)	ASSIGN(BUF2)	ASSIGN(BUF2)
3000		RELEASE	RELEASE
4000		ASSIGN(等候)	ASSIGN(BUF3)
5000			RELEASE
6000			ASSIGN(BUF4)
7000			RELEASE
8000			ASSIGN(等候)
8500	指定 BUF1，输出停止	指定 BUF1,OUT BUF2	指定 BUF1,OUT BUF2
9500	RELEASE, OUT BUF1	RELEASE	
10500	ASSIGN (等候)	ASSIGN(等候)	
15500			RELEASE

等等。当$N=1$时，总的时间是 110000 u；当$N=2$时，它是 89000 u；当$N=3$时，它是 81500 u；而且当$N\geqslant 4$时，它是 76000 u。

12．下列代码应插入程序 B 中的“LD5-1，5”之前：

```
        STA   2F
        LDA   3F
        CMPA  15, 5 (5:5)
        LDA   2F
```

然后指令“JMP 1B”应当改变成

```
        JNE   1B
        JMP   COMPUTE
        JMP   *-1      (或 JMP COMPUTEX)
2H      CON   0
3H      ALF
```

13.

```
        JRED  CONTROL(U)
        J6NZ  *-1
```

14．如果$N=1$，则算法失败（可能涉及缓冲器，虽则输入/输出在进行中）；否则，这个构造将产生有两个黄色缓冲区之效果。如果计算程序一次要访问两个缓冲区，则这是有用的，虽然它冻结缓冲区的空间。一般地说，超过 RELEASE（释放）的 ASSIGN（指定）将是非负的，而且不大于N。

15.

```
U         EQU    0
V         EQU    1
BUF1      ORIG   *+100
BUF2      ORIG   *+100
BUF3      ORIG   *+100
TAPECPY   ENT1   99
          IN     BUF1(U)
1H        IN     BUF2(U)
```

```
OUT     BUF1(V)
IN      BUF3(U)
OUT     BUF2(V)
IN      BUF1(U)
OUT     BUF3(V)
DEC1    3
J1P     1B
OUT     BUF1(V)
HLT
END     TAPECPY
```

这是在图 26 中所指出之算法的一种特殊情况。

18. 部分的解答：在下列算法中，t 是一个变量，当输入/输出设备活动时被置为 0，而当它空闲时等于 1。

算法 A（ASSIGN，通常状态的子程序）。

这个算法与算法 1.4.4A 没有什么变化。

算法 R（RELEASE，通常状态的子程序）。

R1. n 增加 1。

R2. 如果 $t = 0$，则引起一个中断（使用 INT 操作符），它应当转到步骤 B2。

算法 B（缓冲区控制程序，它处理中断）。

B1. 如果 $n = 0$，则置 $t \leftarrow 0$ 并重新启动主程序。

B2. 置 $t \leftarrow 1$，并且从由 NEXTR 确定的缓冲区区域开始输入/输出。

B3. 重新启动主程序；一“输入/输出完成”条件，将中断到步骤 B4。

B4. 把 NEXTR 推进到顺时针方向的下一个缓冲区。

B5. n 减 1，并转到步骤 B1。 ∎

2.1 节

1.（a）SUIT (NEXT(TOP))=SUIT(NEXT (242))=SUIT(386)= 4。（b）A。

2. 每当 V 是一个链接变量时（否则 CONTENTS(V)就没有意义），它的值就不是 A。在象这里的上下文中，避免使用 LOC 是明智的。

3. 置 NEWCARD←TOP，而且如果 TOP≠A，则置 TOP←NEXT(TOP)。

4. **C1**. 置 X←LOC(TOP)。（为了方便起见，我们做一个适当的假定：TOP=NEXT (LOC(TOP))，即是，TOP 的值出现于保存它的单元的 NEXT 场中。这个假定与程序（5）是相容的，而且它使我们免得去写对于一空叠情况的一个专用程序。）

C2. 如果 NEXT(X)≠A，则置 X←NEXT(X) 并重复这一步骤。

C3. 置 NEXT(X)←NEWCARD，NEXT(NEWCARD)←A，TAG(NEWCARD) ←1。 ∎

5. **D1**. 置 X←LOC(TOP)，Y←TOP。（见上面的步骤 C1。由假设，Y≠A。贯穿下面的算法，在 Y=NEXT(X)的意义下，X 拖着 Y 后一步。）

D2. 如果 NEXT(Y)≠A，则 X←Y，Y←NEXT(Y)，并重复这一步骤。

D3. (现在NEXT(Y)=A，所以Y指向底下的卡片；而且X指向次末一张卡片。) 置NEXT(X)←A，NEWCARD←Y。 ∎

6. (b)和(d)。不是(a)！CARD是一个节点，不是一指向一个节点的链接。

7. 序列(a)给出NEXT(LOC(TOP))，在这种情况下它恒等于TOP的值；序列(b)是正确的。不应造成混乱；当X是一个数值变量时，考虑类似的例子：为把X带入寄存器A，我们写LDA X，而不写ENTA X，因为后者把LOC(X)带入寄存器。

8. 设rA=N，rI1=X。

```
ENTA    0                    B1.  N←0
LD1     TOP                       X←TOP
J1Z     *+1                  B2.  X=A吗?
INCA    1                    B3.  N←N+1
LD1     0,1(NEXT)                 X←NEXT(X)
J1NZ    *-2                                    ∎
```

9. 设rI2=X。

```
PRINTER  EQU   18          打印机的设备号
TAG      EQU   1:1
NEXT     EQU   4:5         场的定义
NAME     EQU   0:5
PBUF     ALF   PILE        在空叠的情况下
         ALF   EMPTY            打印的信息
         ORIG  PBUF+24
BEGIN    LD2   TOP         置X←TOP
         J2Z   2F          是空叠吗?
1H       LDA   0,2(TAG)    rA←TAG(X)
         ENT1  PBUF        对MOVE指令准备好
         JBUS  *(PRINTER)  等候打印机准备好
         JAZ   *+3         TAG=0 (卡片面朝上) 吗?
         MOVE  PAREN(3)    否：置圆括弧
         JMP   *+2
         MOVE  BLANKS(3)   是：置空格
         LDA   1,2(NAME)   rA←NAME(X)
         STA   PBUF+1
         LD2   0,2(NEXT)   置X←NEXT(X)
2H       OUT   PBUF(PRINTER) 打印此行
         J2NZ  1B          如果X≠A，则重复
DONE     HLT                    这打印循环
PAREN    ALF    (
BLANKS   ALF
         ALF   )
         ALF                                    ∎
```

2.2.1 小节

1. 是(在两端之一一致地插入所有的条款)。

2．为得到325641，作 SSSXXSSXSXXX（在下列习题的记号下）。不可能达到 154623 的次序，因为仅仅在 3 插入堆栈之前，从堆栈中撤消 2 时，2 才能在 3 之前。

3．一个可允许的序列是当我们从左到右来读时，其中 X 的个数决不超过 S 的个数这样的序列。

两个不同的可允许的序列必须给出不同的结果，因为如果两个序列在一处相一致，于该处一个序列为 S 而另一个为 X，则后一个序列输出一个符号；这个符号，在由前一个序列的 S 刚刚插入的符号之前，不可能被输出。

4．这个问题等价于许多其它有趣的问题，诸如二叉树的枚举，把圆括弧插入一个公式中的方法数，以及把一个多边形分成三角形的方法数，而且它早在 1759 年就出现于欧拉和西格纳（Segner）的笔记中（见 2.3.4.6 小节）。为了进一步参考，见阿·厄尔德莱和艾·马·哈·埃思林顿，爱丁堡数学学报，32（1940），1～12。

下面的精采的解法是由德·安德烈（D.André）（1878）给出的：显然有 $\binom{2n}{n}$ 个含 S 和 X 各 n 个的序列。剩下的是来计算非允许的序列数（它包含正确个数的 S 和 X，但是违背其它条件）。在任何非允许的序列中，定出使得 X 的个数超过 S 的头一个 X 的位置。然后，在引向并包括这个 X 的部分序列中，以 S 代替所有的 X 并以 X 代替所有的 S。结果是一个有（$n+1$）个 S 和（$n-1$）个 X 的序列。反过来，对于后一种类型的每一个序列，我们都能逆转这个过程，而且找出导致它的前一种类型的非允许序列。例如，序列 XXSXSSSXXSSS 必然来自 SSXSXXXXXSSS。这个对应说明，非允许序列的个数是 $\binom{2n}{n-1}$。因此 $a_n=\binom{2n}{n}-\binom{2n}{n-1}$。

利用同样的思想，我们可以解决概率论的更一般的“抽签问题”，它实际上是要枚举所有的具有给定个数的 S 和 X 的部分可允许序列。对于抽签问题的历史及其某些推广，见戴·埃·巴顿和科·林·马洛斯（C.L.Mallows）的易于理解的评述，《数理统计年鉴》(Annals of Math. Statistics)36(1965),236～260；也见习题 2.3.4.4-32 及 5.1.4 小节。

在这里，我们给出一个新的通过使用双重生成函数的方法，来解决抽签问题，因为这个方法使它得以解决诸如习题 11 那样更困难的问题。

设 g_{nm} 是长度为 n 的 S 和 X 的序列的个数，在这些序列中，如果我们从左边计数，则 X 的个数决不超过 S 的个数，而且总共是 S 比 X 多 m 个。于是 $a_n=g_{(2n)0}$。显然 g_{nm} 为 0，除非 $m+n$ 为偶数。容易找出为这些数所满足的递推关系，即是

$$g_{(n+1)m}=g_{n(m-1)}+g_{n(m+1)},\quad m\geqslant 0,\quad n\geqslant 0;\quad g_{0m}=\delta_{0m}。$$

我们建立双重生成函数 $G(x,z)=\sum_{n,m}g_{nm}x^mz^n$，而且设 $g(z)=G(0,z)$。上边的递推关系变换成为

$$\left(x+\frac{1}{x}\right)G(x,z)=\frac{1}{x}g(z)+\frac{1}{z}(G(x,z)-1)$$

即

$$G(x,z)=\frac{zg(z)-x}{z(x^2+1)-x}$$

这个等式，如果我们置 $x=0$，则不幸地什么也没有告诉我们，但是，我们可把分母分解成 $z(1-r_1(z)x)(1-r_2(z)x)$，其中

$$r_1(z)=\frac{1}{2z}(1+\sqrt{1-4z^2}),\quad r_2(z)=\frac{1}{2z}(1-\sqrt{1-4z^2})$$

(注意，$r_1+r_2=1/z$；$r_1r_2=1$。)我们现在以启发式进行；问题是要求$g(z)$的某个值，使其如上边的公式所给出的$G(x,z)$，有一个x和z的无穷幂级数展开。注意$r_2(z)$有这样一个展开，而且$r_2(0)=0$；而对于固定的z，值$x=r_2(z)$使得$G(x,z)$之分母为0。这就提示我们，可以选择$g(z)$使得当$x=r_2(z)$时分子亦为0，即，取$zg(z)=r_2(z)$。现在$G(x,z)$的等式简化为

$$G(x,z)=\frac{r_2(z)}{z(1-r_2(z)x)}=\sum_{n\geqslant 0}(r_2(z))^{n+1}x^nz^{-1}$$

由于这是一个满足原来等式的幂级数展开，我们必然能找到$g(z)$的正确选择。

$g(z)$的系数就是对我们问题的解答。实际上，我们能进一步进行而且对于$G(x,z)$的所有系数能推导出一个简单的形式：由二项式定理，

$$r_2(z)=\sum_{k\geqslant 0}z^{2k+1}\binom{2k+1}{k}\frac{1}{2k+1}$$

设$w=z^2$和$r_2(z)=zf(w)$，则在习题1.2.6-25的记号下$f(w)=\sum_{k\geqslant 0}A_k(1,-2)w^k$；因此

$$f(w)'=\sum_{k\geqslant 0}A_k(r,-2)w^k$$

我们现在有

$$G(x,z)=\sum_{n,m}A_m(n,-2)x^nz^{2m+n}$$

所以一般的解为

$$g_{(2n)(2m)}=\binom{2n+1}{n-m}\frac{2m+1}{2n+1}=\binom{2n}{n-m}-\binom{2n}{n-m-1};$$

$$g_{(2n+1)(2m+1)}=\binom{2n+2}{n-m}\frac{2m+2}{2n+2}=\binom{2n+1}{n-m}-\binom{2n+1}{n-m-1}。$$

5．如果$j<k$和$p_j<p_k$，则我们必须在p_k置于堆栈之前，把p_j取走；如果$p_j>p_k$，则我们必须把p_k留在堆栈上，直到p_j放到堆栈之后。把这两条规则结合起来，条件$i<j<k$和$p_j<p_k<p_i$是不可能的，因为它意味着p_j必须在p_k之前和在p_i之后离开，而p_i又出现在p_k之后。

反之，通过使用算法“对于$j=1,2,\cdots,n$输入0个或多个条款（按必要之个数），直到p_j头一次出现在堆栈中，然后输出p_j”，可以得到所欲的排列。仅当我们达到一个j，p_j不在栈的顶上但它被$k>j$的某个元素p_k所覆盖时，这个算法可能失败。由于这个堆栈的内容总在单调增加，我们有$p_j<p_k$。这个元素p_k可能到达那里，仅当对于某个$i<j$，它小于p_i。

爱德华·M·赖因戈尔德（Edward M.Reingold）已经发现了这个结果的一个有趣的推广：假设，除了堆栈之外，还有随机存取存储的m个单元，可暂存元素于其中。于是，一个排列是可达到的，当且仅当，它不包含子序列$xy_1\cdots y_{m+1}z_1\cdots z_{m+1}$使得对于所有的$i$，$j$，$y_i<z_j<x$。〔SIAM J. Computing 1 (1972), 350～353。〕

6．按排队的本性仅仅是平凡的一个12…n。

7．一个首先输出n的输入受限的双排队，必须以如同它的头n个操作那样的顺序，简单地把值12…n放到双排队上。一个首先输出n的输出受限的双排队，必须把值$p_1p_2\cdots p_n$放到它的双排队上，如同它的头n个操作那样。因此我们求出唯一的答案(a)4132(b)4213(c)4231。

8．当$n=4$时，无；当$n=5$时，有4个（见习题13）。

9．由逆转的操作，我们可以通过输出受限的双排队，得到任何输入受限的排列之逆的逆，且反之亦然。这就在两组排列之间建立一一对应关系。

10．(i)应当有n个X和n个结合的S和Q。(ii)如果我们从左边读的话，X的个数总不得超过结合的S和Q的个数。(iii)每当X的个数等于结合的S和Q的个数时(从左边读)，下一个字符必须是Q。(iv)两个操作XQ永远不得以这个次序相邻。

显然，规则(i)和(ii)是必要的。加上额外的规则(iii)和(iv)以消除二义性，由于当双排队为空时S就等同于Q，而且由于XQ总可为QX所代替。于是，任何可得到的序列至少对应于一个可允许的序列。

为了说明两个可允许的序列给出不同的排列，考虑直到一处皆相同的序列，而且一个序列有一个S，另一个有一个X或Q。由于按(iii)，双排队非空，显然通过两个序列得到不同的排列（相对于通过S移动元素的次序)。剩下的情况是序列A，B直到一处皆相同，而后序列A有Q，序列B有X。序列B在该处可能有更多的X，而按(iv)它们必须接以S，所以这两个排列还是不同的。

11．如象在习题4中那样进行，我们设g_{nm}是长度为n的部分可允许的序列的个数，它在双排队上保留m个元素，不以符号X结束；h_{nm}类似地定义，是那些以X结束的序列。我们有$g_{(n+1)m}=2g_{n(m-1)}(+h_{n(m-1)}$如果$m>1)$；$h_{(n+1)m}=g_{n(m+1)}+h_{n(m+1)}$。类似于习题4中的定义，定义$G(x,z)$和$H(x,z)$；我们有

$$G(x,z)=xz+2x^2z^2+4x^3z^3+(8x^4+2x^2)z^4+(16x^5+8x^3)z^5+\cdots$$

$$H(x,z)=z^2+2xz^3+(4x^2+2)z^4+(8x^3+6x)z^5+\cdots$$

如果$h(z)=H(0,z)$，则我们求得$z^{-1}G(x,z)=2xG(x,z)+x(H(x,z)-h(z))+x$，$z^{-1}H(x,z)=x^{-1}G(x,z)+x^{-1}(H(x,z)-h(z))$；因之

$$G(x,z)=\frac{xz(x-z-xh(z))}{x-z-2x^2z+xz^2}$$

如同在习题4中那样，我们试图选择$h(z)$以使分子与分母的一个因式相消。我们发现$G(x,z)=xz/(1-2r_2(z))$，其中

$$r_2(z)=\frac{1}{4z}(z^2+1-\sqrt{(z^2+1)^2-8z^2})$$

利用$b_0=1$的约定，所求的生成函数成为

$$\frac{1}{2}(3-z-\sqrt{1-6z+z^2})=1+z+2z^2+6z^3+22z^4+90z^5+\cdots$$

通过微商，我们找出便于计算的递归关系：$nb_n=3(2n-3)b_{n-1}-(n-3)b_{n-2}$，$n\geqslant 2$。

另一个解决这个问题的方法，是由沃·普拉特（V.Pratt）提出的，这方法使用对于串的集合的上下文无关文法（参照第11章)。具有产生式$S\to q^n(Bx)^n$，$B\to sq^n(Bx)^{n+1}B$，

对于所有 $n \geqslant 0$，以及 $B \to \epsilon$ 的无限文法，是无二义的，而且它允许我们来计算具有 n 个X的串的个数，如同在习题 2.3.4.4-31 中那样。

12. 如果 $|\alpha|<1$，则在 $\sqrt{1-w}\sqrt{1-\alpha w}=\sqrt{1-w}\sqrt{1-\alpha}+(1-w)^{3/2}\times\alpha/(\sqrt{1-\alpha w}+\sqrt{1-\alpha})$ 中 w^n 的系数是

$$(-1)^n\binom{\frac{1}{2}}{n}\left(\sqrt{1-\alpha}+O\left(\frac{1}{n}\right)\right)$$

由斯特林近似公式，这近似于 $-\frac{1}{2}\sqrt{(1-\alpha)\pi}\,n^{-3/2}$。现在 $1-6z+z^2=(1-(3+\sqrt{8})z)(1-(3-\sqrt{8})z)$。命 $w=(3+\sqrt{8})z$，我们求得 $a_n\sim 4^n/\sqrt{\pi n^3}$；$b_n\sim c(3+\sqrt{8})^n n^{-3/2}$，其中 $c=\frac{1}{2}\sqrt{(3\sqrt{2}-4)/\pi}\approx 0.139$。

13. 沃·普拉特已经发现，一个排列是不能得到的，当且仅当，对于某个 $k\geqslant 1$，它包含一个其相对数量分别为

5，2，7，4，…，$4k+1$，$4k-2$，3，$4k$，1　或5，2，7，4，…，$4k+3$，$4k$，1，$4k+2$，3

的子序列，或者为交换其后两个元素者，或者为交换其1和2者，或者为此两者兼有之。因此，对于 $k=1$，禁止的样式为52341，52314，51342，51324；5274163，5274136，5174263，5174236。

2.2.2 小节

1. M－1（不是M）。如果我们允许M个项目，象（6）和（7）那样，则将不可能通过检查R和F来区别一个全满的排队与一个空的排队，因为仅有M种可能性能被检测。放弃一个存储单元，比起去编写过于复杂的程序来，要更好些！

2. 从后尾删去：如果R＝F，则UNDERFOLW；Y←X〔R〕；如果R＝1，则R←M，否则R←R－1。在前头插入：置X〔F〕←Y；如果F＝1则F←M，否则F←F－1；如果F＝R则OVERFLOW。

3.（a）LD1 I；LDA BASE，7:1。这花去5个循环，而不是象在（8）中那样的4个或8个。

（b）解法1：LDA BASE，2:7，其中每个基地址通过 $I_1=0$，$I_2=1$ 来存储。解法2：如果希望以 $I_1=I_2=0$ 来存基地址，则我们可以写LDA X，7:1，其中单元X含有NOP BASE，2:7。第二个解法多花去一个循环，但是允许任何变址寄存器来使用“基表”。

（c）这等价于“LD4 X(0:2)”，而且花去同样的执行时间，但当X(0:2)包含－0时rI4将置成＋0。

4.（i）NOP*，7。(ii) LDA X，7:7(0:2)。(iii) 这是不可能的；代码LDA Y，7:7，其中单元Y含有NOP X，7:7者，破坏了对7:7的限制（见习题5）。(iv)LDA X，7:1连同辅助常数

X　NOP　*+1，7:2

```
NOP   *+1, 7:3
NOP   *+1, 7:4
NOP   0, 5:6
```

执行时间是6个单位。(v) INC6 X, 7:6, 其中X含有 NOP 0, 6:6。

5. (a) 考虑指令 ENTA 1000, 7:7 连同内存配置

单元	ADDRESS	I_1	I_2
1000:	1001	7	7
1001:	1004	7	1
1002:	1002	2	2
1003:	1001	1	1
1004:	1005	1	7
1005:	1006	1	7
1006:	1008	7	7
1007:	1002	7	1
1008:	1003	7	2

并连同 rI1=1, rI2=2。我们求得 1000, 7, 7=1001, 7, 7, 7=1004, 7, 1, 7, 7=1005, 1, 7, 1, 7, 7=1006, 7, 1, 7, 7=1008, 7, 7, 1, 7, 7=1003, 7, 2, 7, 1, 7, 7=1001, 1, 1, 2, 7, 1, 7, 7=1002, 1, 2, 7, 1, 7, 7=1003, 2, 7, 1, 7, 7=1005, 7, 1, 7, 7=1006, 1, 7, 1, 7, 7=1007, 7, 1, 7, 7=1002, 7, 1, 1, 7, 7=1002, 2, 2, 1, 1, 7, 7=1004, 2, 1, 1, 7, 7=1006, 1, 1, 7, 7=1007, 1, 7, 7=1008, 7, 7=1003, 7, 2, 7=1001, 1, 1, 2, 7=1002, 1, 2, 7=1003, 2, 7=1005, 7=1006, 1, 7=1007, 7=1002, 7, 1=1002, 2, 2, 1=1004, 2, 1=1006, 1=1007。(手工来作这个推导，一种也许更快的方法，将是逐次地计算在单元 1002, 1003, 1007, 1008, 1005, 1006, 1004, 1001, 1000中以此顺序确定的地址，但是看起来，一台计算机实质上将需要如上所示地进行求值)。作者试验过若干想象的方案，在计算地址的同时来改变内存内容，而且已经设计成使得在得到了最后的地址时再次恢复每一事项。类似的算法出现于2.3.5小节中。然而，这些企图并没有获得结果，而且看起来恰恰是没有足够的单元来存储必要的信息。

(b, c)设H, C是辅助寄存器，而且设N是一个计数器。对于单元L中的指令，为了得到有效的地址M，按如下方法来做：

A1.〔起始〕置H←0, C←L, N←0。(C将是“当前的”单元，H用来把各个变址寄存器的内容加到一起，而N测量间接寻址的“深度”)。

A2.〔检查地址〕 置M←ADDRESS(C)。如果$I_1(C)=j$, $1\leqslant j\leqslant 6$, 则置M←M+rIj。如果$I_2(C)=j$, $1\leqslant j\leqslant 6$, 则置H←H+rIj。如果$I_1(C)=I_2(C)=7$, 则置N←N+1, H←0。

A3.〔间接？〕 如果或$I_1(C)$或$I_2(C)$等于7, 则置C←M并转到A2。否则置M←M+H, H←0。

A4.〔降低深度〕 如果N>0, 则置C←M, N←N−1, 并转到A2。否则M是所求的答案。 ∎

这个算法将正确地处理任何情况，除了 $I_1=7$ 且 $1\leqslant I_2\leqslant 6$，以及在 ADDRESS 中地址的计算涉及 $I_1=I_2=7$ 的一种状况。效果就好象 I_2 为 0 一样。为了了解算法 A 的操作，考虑部分（a）的记号；在上述算法中，状态“L，7，1，2，3，5，7，7，7，7”通过 C 或 M＝L，N＝4（末尾 7 的个数），而且 H＝(rI1)＋(rI2)＋(rI3)＋(rI5)(后变址）来表示。在这道习题的部分（b）的解答中，计数器 N 不是 0 就是 1。

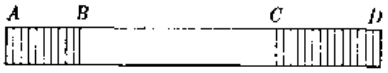
6.（c）引起 OVERFLOW。（e）引起 UNDERFLOW，如果这个程序继续，则它在最后的 I_2 上引起 OVERFLOW。

7. 否，因为 TOP[i]必须大于 OLDTOP[i]。

8. 对于一个堆栈，有用的信息出现在一端，而空信息出现在另一端：



其中 A＝BASE[j]，B＝TOP[j]，C＝BASE[$j+1$]。对于排队或双排队，有用的信息出现在一端，而空的信息出现在当中的某些地方：



或者有用的信息在当中而空信息在两端：



其中 A＝BASE[j]，B＝REAR[j]，C＝FRONT[j]，D＝BASE[$j+1$]。这两种情况分别通过条件 $B\leqslant C$，$B\geqslant C$ 区别之。因此以一种明显的方式来修改这些算法，以使空信息的间隔加宽或缩小（于是在溢出的情况下，即是当 $B=C$ 时，我们通过移动一部分而不移动另一部分，可做成 B 与 C 之间空的空间）。

9. 给定任何的序列说明 $a_1,a_2,\cdots,a_m$，对于每一对使得 $j<k$ 且 $a_j>a_k$ 的数对 (j,k)，要求有一个移动操作。因此，这样的数对的个数，就是所要求的移动的个数。现在想象所有的 n^m 个说明都已写出，而且对于 $j<k$ 的 $\binom{m}{2}$ 个数对 (j,k) 中的每一对，计算有多少个有着 $a_j>a_k$ 的说明。显然，这就是 $\binom{n}{2}$，即对于 a_j 和 a_k 之选择的个数，再乘以 n^{m-2}，即填充剩下的位置之方式的种数。因此在所有的说明当中，移动的总数为 $\binom{m}{2}\binom{n}{2}n^{m-2}$。以 n^m 来除这个数以得到平均值，即得等式(12)。

10. 如同在习题 9 中那样，我们求得预期的值是

$$\binom{m}{2}\sum_{1\leqslant j<k\leqslant n}p_jp_k=\frac{1}{2}\binom{m}{2}((p_1+\cdots+p_n)^2-(p_1^2+\cdots+p_n^2))$$

$$=\frac{1}{2}\binom{m}{2}(1-(p_1^2+\cdots+p_n^2))$$

对于这一模型，它与这些表的相对次序是什么根本无关！（立即说明为什么；如果我们考虑一个给定的序列 $a_1,\cdots,a_m$ 的所有可能的排列，则我们发现，对所有这些排列求和的移动总数，仅仅依赖于不同元素的对偶 $a_j\neq a_k$ 的个数）。

11. 按前面来计算，我们求得预期的数是

$$\frac{1}{n^m}\binom{n}{2}\sum_{0\leqslant s<m}\sum_{r\geqslant t}\binom{s}{r}(n-1)^{s-r}n^{m-s-2}(m-s-1)$$

这里 s 表示 $j-1$，在上面的答案之术语下，而 r 是 $a_1, a_2, \cdots, a_r$ 之中等于 a_j 的元素的个数。这个公式可以稍加简化，例如，通过写出与之对应的生成函数，直至我们达到

$$\frac{1}{2n^t}\sum_{0\leqslant k\leqslant m-t-2}\binom{t-1+k}{k}\binom{m-t-k}{2}\left(1-\frac{1}{n}\right)^{k+1},\quad \text{对于 } t\geqslant 0$$

是否还有一个更简单的方法来给出这个答案呢？显然没有了，因为生成函数为

$$\sum_m F_{tm}z^m=\binom{n}{2}\frac{1}{(1-nz)^s}\left(\frac{z}{1-(n-1)z}\right)^t$$

12. 如果 $m=2k$，则平均值是 2^{-2k} 乘以

$$\binom{2k}{0}2k+\binom{2k}{1}(2k-1)+\cdots+\binom{2k}{k}k+\binom{2k}{k+1}(k+1)+\cdots+\binom{2k}{2k}2k$$

后者之和为

$$\binom{2k}{k}k+2\left(\binom{2k-1}{k}2k+\cdots+\binom{2k-1}{2k-1}2k\right)=\binom{2k}{k}k+4k\cdot\frac{1}{2}\quad 2^{2k-1}$$

当 $m=2k+1$ 时，可使用一个类似的论证。答案为

$$\frac{m}{2}+\frac{m}{2^m}\binom{m-1}{\lfloor m/2\rfloor}$$

14. 设 $k_j=n/m+\sqrt{n}x_j$（这个思想是由尼·戈·德·布鲁因提出的）。斯特林近似公式系指，当 $k_1+\cdots+k_n=m$ 且当诸 x 一致地有界时，

$$n^{-m}\frac{m!}{k_1!\cdots k_n!}\max(k_1,\cdots,k_n)=\sqrt{2\pi}^{1-n}n^{n/2}\left(\frac{m}{n}+\sqrt{m}\max(x_1,\cdots,x_n)\right)$$
$$\times\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)\sqrt{m}^{1-m}\left(1+O\left(\frac{1}{\sqrt{m}}\right)\right)$$

对于满足这个条件的所有非负的 $k_1,\cdots,k_n$，后边这个量之和近似于一个黎曼积分；我们可以导出这个和的渐近性质，即是 $a_n(m/n)+c_n\sqrt{m}+O(1)$，其中

$$a_n=\sqrt{2\pi}^{1-n}n^{n/2}\int_{x_1+\cdots+x_n=0}\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)dx_2\cdots dx_n$$

$$c_n=\sqrt{2\pi}^{1-n}n^{n/2}\int_{x_1+\cdots+x_n=0}\max(x_1,\cdots,x_n)$$
$$\times\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)dx_2\cdots dx_n$$

因为能够证明，对于任何 ϵ，对应的和进入 a_n 和 c_n 的 ϵ 之内。

我们知道 $a_n=1$，因为对应的和可明确地进行求值。出现于 c_n 的表达式中的积分等于 nI_1，其中

$$I_1=\int_{\substack{x_1+\cdots+x_n=0\\ x_1\geqslant x_2,\cdots,x_n}}x_1\exp\left(-\frac{n}{2}(x_1^2+\cdots+x_n^2)\right)dx_2\cdots dx_n$$

我们可以作替换

$$x_1=\frac{1}{n}(y_2+\cdots+y_n),\quad x_2=x_1-y_2,\quad x_3=x_1-y_3,\cdots,\quad x_n=x_1-y_n$$

然后我们求得 $I_1=I_2/n$，其中

$$I_2=\int_{y_2,\cdots,y_n\geqslant 0}(y_2+\cdots+y_n)\exp\left(-\frac{Q}{2}\right)dy_2\cdots dy_n,$$

$Q=n(y_2^2+\cdots+y_n^2)-(y_2+\cdots+y_n)^2$。现在由对称性，$I_2$是（$n-1$）乘上这同一积分，但其中以$y_2$代替$(y_2+\cdots+y_n)$；因此$I_2=(n-1)I_3$，其中

$$I_3=\int_{y_2,\cdots,y_n\geqslant 0}(ny_2-(y_2+\cdots+y_n))\exp\left(-\frac{Q}{2}\right)dy_2\cdots dy_n$$

$$=\int_{y_3,\cdots,y_n\geqslant 0}\exp\left(-\frac{Q_0}{2}\right)dy_3\cdots dy_n;$$

这里Q_0是以0代替y_2后的Q。〔当$n=2$时，命$I_3=1$。〕现在设$z_j=\sqrt{n}\ y_j-(y_3+\cdots+y_n)/(\sqrt{2}+\sqrt{n})$，$3\leqslant j\leqslant m$。于是$Q_0=z_3^2+\cdots+z_n^2$，而且我们导出$I_3=I_4/n^{(n-2)/2}\sqrt{2}$，其中

$$I_4=\int_{y_3,\cdots,y_n\geqslant 0}\exp\left(-\frac{z_3^2+\cdots+z_n^2}{2}\right)dz_3\cdots dz_n$$

$$=\alpha_n\int\exp\left(-\frac{z_3^2+\cdots+z_n^2}{2}\right)dz_3\cdots dz_n$$

$$=\alpha_n(\sqrt{2\pi})^{n-2}$$

其中α_n为（$n-2$）维空间中由向量$(n+\sqrt{2n},\ 0,\cdots,0)-(1,1,\cdots,1),\cdots,(0,0,\cdots,n+\sqrt{2n})-(1,1,\cdots,1)$支起的“立体角”，除以整个空间的总立体角。因此

$$c_n=\frac{(n-1)\sqrt{n}}{2\sqrt{\pi}}\alpha_n$$

我们有

$$\alpha_2=1,\quad \alpha_3=\frac{1}{2},\quad \alpha_4=\frac{1}{\pi}\arctan\sqrt{2}\approx.304$$

而且

$$\alpha_5=\frac{1}{8}+\frac{1}{4\pi}\arctan\frac{1}{\sqrt{8}}\approx.206$$

〔c_3的值是由罗伯特·马·科泽尔卡（Robert M. Kozelka）求出的，《数理统计年鉴》（Annals of Math. Stat.）27（1956），507～512。但是，对于更高的n值，这一问题的解答，显然还从未在文献中出现过〕。

16. 不是，除非排队满足应用于原始方法（4），（5）的限制。

2.2.3 小 节

1. OVERFLOW 蕴含在操作 P⇐AVAIL中。

2.
```
INSERT  STJ   1F          存 "NOP T" 的单元
        STJ   9F          存出口单元
        LD1   AVAIL       rI1⇐AVAIL
        J1Z   OVERFLOW
        LD3   0,1 (LINK)
        ST 3  AVAIL
        STA   0, 1 (INFO) INFO(rI1)⇐Y
```

```
        1H      LD3     *(0:2)          rI3←LOC(T)
                LD2     0, 3            rI2←T
                ST2     0, 1 (LINK)     LINK (rI1)←T
                ST1     0, 3            T←rI1
        9H      JMP     *               ■
3.      DELETE  ST1     1F              存"NOP T"的单元
                ST1     9F              存出口单元
        1H      LD2     *(0:2)          rI2←LOC(T)
                LD3     0, 2            rI3←T
                J3Z     9F              是T = A?
                LD1     0, 3 (LINK)     rI1←LINK(T)
                ST1     0, 2            T←rI1
                LDA     0, 3(INFO)      rA←INFO(rI1)
                LD2     AVAIL           AVAIL←rI3
                ST2     0, 3 (LINK)
                ST3     AVAIL
                ENT3    2               准备第二个出口
        9H      JMP     *, 3            ■
4.      OVERFLOW ST1    9F              存rJ的内容
                ST1     8F(0:2)         保留rI1 的内容
                LD1     POOLMAX
                ST1     AVAIL           置AVAIL指新单元
                INC1    c
                ST1     POOLMAX         增加POOLMAX
                CMP1    SEQMIN
                JG      TOOBAD          存贮已超过?
                STZ     -c, 1 (LINK)    置LINK (AVAIL)←A
        9H      ENT1    *               取rJ的内容
                DEC1    2               减 2
                ST1     * + 2 (0:2)     存出口单元
        8H      ENT1    *               恢复rI1
                JMP     *               返回  ■
```

5．在前头的插入，实际上类似于基本的插入操作（8），但加上对于空排队的附加测试：P⇐AVAIL，INFO(P)←Y，LINK(P)←F；如果F＝A，则R←P；F←P。

为了从后尾删去，我们要找出，哪一个节点链接到NODE(R)，而且这必然是低效的，因为我们需要特地从F去检索。例如，这可以按如下步骤进行：

a）如果F＝A，则UNDERFLOW，否则置P←LOC(F)。

b）如果LINK(P)≠R，则置P←LINK(P)并重复这一步骤直到LINK(P)＝R。

c）置Y←INFO(R)，AVAIL⇐R，R←P．LINK(P)←A。

6．我们可以从(14)撤消操作LINK(P)←A，如果我们从(17)删去命令"F←LINK(P)"和"如果F＝A则置R←LOC(F)"；后者代之以"如果F＝R，则F←A且R←LOC(F)，否则置F←LINK(P)"。

这些改动的效果是，排队中的后尾节点的LINK场将包含决不为程序所询问的多余的信息。类似于此的技巧，节省了执行时间，而且它在实践中十分有用，尽管它违背了废料收集（见2.3.5小节）的基本假定之一，可它不能与这样的算法一起使用。

7.（要确保你的解答对于空的表进行工作。）

I1. 置P←FIRST，Q←A。

I2. 如果P≠A，则置R←Q，Q←P，P←LINK(Q)，LINK(Q)←R，并重复这一步骤。

I3. 置FIRST←Q。 ∎

实质上，我们是在把一些节点由一个堆栈弹出并把它们压入另一个堆栈。

8.	LD1	FIRST	1	P≡rI1←FIRST
	ENT2	0	1	Q≡rI2←Λ
	J1Z	2F	1	若表为空，则跳
1H	ENTA	0, 2	n	R≡rA←Q
	ENT2	0, 1	n	Q←P
	LD1	0, 2(LINK)	n	P←LINK(Q)
	STA	0, 2(LINK)	n	LINK(Q)←R
	J1NZ	1B	n	P≠Λ吗?
2H	ST2	FIRST	1	FIRST←Q。 ∎

时间是$(7n+6)u$。可达到的更快的速度为$(5n+$常数$)u$；参看习题1.1-3。

9.（a）是。（b）是的，如果考虑法定的血统的话（一个人的女儿不可以同他的父亲结婚，以至形成“我是我自己的爷爷”之怪事)●。（c）否（$-1\prec 1$且$1\prec -1$)。（d）但愿如此，或者否则将有一个循环的论证。（e）$1\prec 3$且$3\prec 1$。（f）命题有二义性。如果我们采用这样的见解，即是，为y调用的诸子程序要取决于哪一个子程序调用y，则我们就将结论：传递律不成立（例如，一个通用的输入/输出子程序可以调用每台现有输入/输出设备的不同处理程序，但通常在一单个的程序中又不需要全部的这些处理子程序。这乃是一个折磨许多自动程序设计系统的问题)。

10. 对于(i)，有三种情况：$x=y$；$x\subset y$且$y=z$；$x\subset y$且$y\subset z$。对于(ii)，有两种情况：$x=y$；$x\neq y$。这些情况的每一种均可平凡地处理之，(iii)也是。

11. “乘出”下列的数以得到全部52个解：13749(25+52)86+(1379+1397+1937+9137)(4258+4528+2458+5428+2548+5248+2584+5284)6+(1392+1932+1923+9123+9132+9213)7(458+548+584)6。

12. 例如：（a）在所有具有$k+1$个元素（$0\leqslant k<n$）的集合之前，列出所有具有k个元素的集合（以任何次序)。（b）通过这样的序列来表示一个子集合，即是，一个由0和1组成的序列，它说明哪一个元素在该子集合中。这给出了一个在所有子集合与（经由二进数系统的）整数0到2^n-1之间的对应关系。对应的次序就是一个拓扑序列。

13. D. J. 格莱弗曼（D. J. Kleffman）已经证明$\lim\limits_{n\to\infty} 2^{-n}\log f(n)=\lim\limits_{n\to\infty} 2^{-n}\log_{o<k<n}\binom{n}{k}$1〔待出版〕。

14. 如果$a_1\ a_2\cdots a_n$和$b_1b_2\cdots b_n$是两个可能的拓扑分类，则命j是使$a_j\neq b_j$的极小值；于是就有$a_k=b_j$和$a_j=b_m$对于某k，$m>j$。现在$b_j\not\leq a_j$因为$k>j$，而且$a_j\not\leq b_j$因为$m>j$，

● 翻译时对原意有所改动。——译者注

因此（iv）不成立。反之，如果仅有一个拓扑分类$a_1\ a_2\cdots a_n$，则我们必须有$a_j\prec a_{j+1}$，对于$1\leqslant j<n$；否则a_j与a_{j+1}将被交换过来。这个和传递性导出（iv）。

注意：下列另一个证明，也可对无穷集合进行。（a）每个偏序均可嵌入到一个线性次序中。因为如果我们有两个元素使得$x_0\not\preceq y_0$和$y_0\not\preceq x_0$，则我们通过规则"$x\preceq y$或$x\prec x_0$且$y_0\preceq y$"，即可生成另一个偏序。这后一个次序"包括"前一个，而且有$x_0\preceq y_0$。现在在通常方式下应用佐恩（Zorn）引理或超穷归纳法来完成证明。（b）显然，一个线性次序不可能被嵌入到任何不同的线性次序中。（c）一个有着两个如同在（a）中那样的不可比较的元素x_0和y_0的偏序，可以扩充成两个线性次序，其中分别有$x_0\preceq y_0$和$y_0\preceq x_0$，所以至少存在两个线性次序。

注意：其交集是一个给定的偏序之线性次序的最小个数，称为该部分次序的维数。这看来是一个重要的概念〔参看奥尔，《图论》(Theory of Graphs)(美国数学协会,1962)，第10章〕，但是还不知道为计算偏序之维数的有效算法。有可能在$O(n^5)$个步骤中来测试维数是否为2〔见第7章〕。

15. 如果S是有限的，则我们可以列出所有在给定的偏序下为真的关系$a\prec b$。通过逐次地撤消任何为其它所蕴涵的关系，一次一个，我们就作成一个非冗余的集合。问题是要证明，恰有一个这样的集合，而不论在什么顺序下我们来撤消这些冗余的关系。如果有两个非冗余的集合α和β，其中"$a\prec b$"出现在α中而不在β中，则对于某个$k\geqslant 1$，在β中有$k+1$个关系$a\prec c_1\prec\cdots\prec c_k\prec b$。但是有可能从$\alpha$导出$a\prec c_1$和$c_1\prec b$，而无须使用关系$a\prec b$（因为$b\preceq c_1$和$c_1\prec a$），因此在$\alpha$中关系$a\prec b$是冗余的。

对于无穷的集合S，当至多有一个非冗余的关系集合时，这个结果是不成立的。例如，如果S表示整数加上元素∞，而且我们定义：对于所有的n，有$n\prec n+1$和$n\prec\infty$，则不存在表征这个偏序的非冗余的关系集合。

16. 设S被拓扑分类为$x_{p_1}x_{p_2}\cdots x_{p_n}$，并把这个排列应用到行和列两者。

17. 如果在步骤T4中k从1增大到n，则输出为1932745860。如果在步骤T4中k从n减小到1，如同它在程序T中那样，则输出为9123745860。

18. 它们以分类的次序把诸条款链接在一起：QLINK〔0〕打头，QLINK〔QLINK〔0〕〕是第二个，以此类推；QLINK〔最后〕$=0$。

19. 在某些情况下将失败；当这个排队在步骤T5中仅含一个元素时，将置$F=0$（由此使排队变空），但其它的条款在步骤T6中可放进这个排队。因此，这个修改需在步骤T6中增加一项对$F=0$的测试。

20. 确实，以如下的方式，可使用一个堆栈：

步骤T4. 置T$\leftarrow$0。对于$1\leqslant k\leqslant n$，如果COUNT〔k〕为0，则作下列工作：置SLINK〔k〕$\leftarrow$T，T$\leftarrow k$。（SLINK〔k〕=QLINK〔k〕。）

步骤T5. 输出T的值。如果T$=0$，则转到T8；否则，置N$\leftarrow$N-1，P$\leftarrow$TOP〔T〕，T$\leftarrow$SLINK〔T〕。

步骤T6. 和以前一样，除了当COUNT〔SUC(P)〕减小成0时，我们置SLINK〔SUC(P)〕$\leftarrow$T和T$\leftarrow$SUC(P)。

21. 重复的关系仅使得这个算法稍微慢些，并且在存储池中花去更多的空间。关系

"$j \prec j$"将被处理作类似一个循环（例如，在对应的图中，一个箭头从一个方框到它自身）。

22. 为使程序"可靠"，我们将（a）检验 $0 < n <$（某个适当的极大值）；（b）检验每个关系 $j \prec k$，对于条件 $0 < j$，$k \leqslant n$；（c）确保关系个数不溢出存储池区域。

23. 在步骤 T5 的末尾，增加"TOP[F]←A"（于是，TOP[1]，…，TOP[n]总是指向未被消去的所有关系）。在步骤 T8，如果 N > 0，则打印"LOOP DETECTED IN INPUT:"，而且对于 $1 \leqslant k \leqslant n$，置QLINK[$k$]←P。现在增加以下的步骤：

T9. 对于 $1 \leqslant k \leqslant n$ 置 P←TOP[k]，TOP[k]←0，并实行步骤 T10（这将把 QLINK[j]置成对象 j 的前驱之一，对于还未输出的每个 j）。然后转到 T11。

T10. 如果 P≠A，置 QLINK[SUC(P)]←k，P←NEXT(P)，并重复这一步骤。

T11. 找一个使得 QLINK[k]≠0 的 k。

T12. 置 TOP[k]←1 以及 k←QLINK[k]。现在如果 TOP[k]=0，则重复这个步骤。

T13.（我们已经找到了一个循环的开端）。打印 k 的值，置 TOP[k]←0，k←QLINK[k]，而且如果TOP[k]=1，则重复这个步骤。

T14. 打印 k 的值（这个循环的开始和末尾）并停机（注意：这个循环已被往后打印；如果希望以向前的次序打印这个循环，则应在步骤 T12 与 T13 之间使用一个类似于习题 7 中的算法）。

24. 在正文的程序中插入三行：

```
08a    PRINTER    EQU    18
14a               ST6    NO
59a               STZ    X,1(TOP)      TOP[F]←Λ。
```

以下列诸行来代替行 74–75：

```
74              J6Z     DONE
75              OUT     LINE1(PRINTER)      打印循环的标志
76              LD6     NO
77              STZ     X,6(QLINK)          QLINK[k]←0
78              DEC6    1
79              J6P     *-2                 n ≥ k ≥ 1
80              LD6     NO
81    T9        LD2     X,6(TOP)            P←TOP[k]
82              STZ     X,6(TOP)            TOP[k]←0
83              J2Z     T9A                 P = A吗?
84    T10       LD1     0,2(SUC)            r1←SUC(P)
85              ST6     X,1(QLINK)          QLINK[r1]←k
86              LD2     0,2(NEXT)           P←NEXT(P)
87              J2P     T10                 P≠A吗?
88    T9A       DEC6    1
89              J6P     T9                  n ≥ k ≥ 1
90    T11       INC6    1
```

```
91                  LD6     X,6(QLINK)
92                  JAZ     *-2                 求k使QLINK(k)≠0
93          T12     ST6     X,6(TOP)            TOP(k)←k
94                  LD6     X,6(QLINK)          k←QLINK(k)
95                  LD1     X,6(TOP)
96                  J1Z     T12                 TOP(k)=0吗?
97          T13     ENTA    0,6
98                  CHAR                        转换k成字母
99                  JBUS    *(PRINTER)
100                 STX     VALUE               打印
101                 OUT     LINE2(PRINTER)
102                 JIZ     DONE                当TOP(k)=0时停止
103                 STZ     X,6(TOP)            TOP(k)←0
104                 LD6     X,6(QLINK)          k←QLINK(k)
105                 LD1     X,6(TOP)
106                 JMP     T13
107         LINE1   ALF     LOOP                标题行
108                 ALF     DETEC
109                 ALF     TED I
110                 ALF     N INP
111                 ALF     UT;
112         LINE2   ALF                         逐次的各行
113         VALUE   EQU     LINE2+3
114                 ORIG    LINE2+24
115         DONE    HLT                         计算结束
116         X       END     TOPSORT             ■
```

26. 解答之一，是象下边这样. 分两个阶段处理：

阶段1（我们为每个需要使用的子程序标志B＝1或2的同时，使用X表格作为一个（顺序的）堆栈）。

A0. 对于1≤J≤N置B(X[J])←B(X[J])+2，如果B(X[J])≤0。

A1. 如果N＝0，则转向阶段2；否则置P←X[N]而且N减1

A2. 如果|B(P)|＝1，则转向A1，否则置P←P＋1。

A3. 如果B(SUB1(P))≤0，则置N←N＋1，B(SUB1(P))←B(SUB1(P))＋2，X[N]←SUB1(P)。如果SUB2(P)≠0且B(SUB2(P))≤0，则对于SUB2(P)作一组类似的动作。转到A2。 ■

阶段2（我们通过表格并分配内存）。

B1. 置P←FIRST。

B2. 如果P＝A，则置N←N－1，BASE(LOC(X[N]))←MLOC，SUB(LOC(X[N]))←0，并结束本算法。

B3. 如果B(P)＞0，则置N←N＋1，BASE(LOC(X[N]))←MLOC，SUB(LOC(X[N]))←P，MLOC←MLOC＋SPACE(P)。

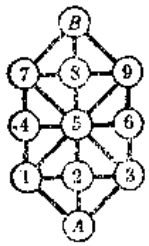
B4. 置P←LINK(P)并返回B2。 ■

27. 关于下列代码的注释，留给读者来作。

```
B       EQU     0:1
SPACE   EQU     2:3
LINK    EQU     4:5
SUB1    EQU     2:3
SUB2    EQU     4:5
BASE    EQU     0:3
SUB     EQU     4:5
A0      LD2     N
        J2Z     B1
1H      LD3     X,2
        LDA     0,3(B)
        JAP     *+3
        INCA    2
        STA     0,3(B)
        DEC2    1
        J2P     1B
        LD1     N
A1      J1Z     B1
        LD2     X,1
        DEC1    1
A2      LDA     0,2(1:1)
        DECA    1
        JAZ     A1
        INC2    1
A3      LD3     0,2(SUB1)
        LDA     0,3(B)
        JAP     9F
        INC1    1
        INCA    2
        STA     0,3(B)
        ST3     X,1
9H      LD3     0,2(SUB2)
        J3Z     A2
        LDA     0,3(B)
        JAP     A2
        INC1    1
        INCA    2
        STA     0,3(B)
        ST3     X,1
        JMP     A2
B1      ENT2    FIRST
```

```
        LDA     MLOC
        CMP     1F
B3      LDX     0,2(B)
        JXNP    B1
        INC1    1
        ST2     X,1(SUB)
        ADD     0,2(SPACE)
1H      STA     X+1,1(BASE)
B1      LD2     0,2(LINK)
B2      J2NZ    B3
        ST2     X+1,1(SUB)
```

28. 这里仅仅给出关于军事游戏的一点注释。设A是指挥“三个士兵”的选手。三个士兵之棋子开始于节点A 13上。在这项游戏中，A必须“捕捉”B，而且如果B有一个位置在第二次被重复，则我们可认为他是胜利者。然而，为了避免把这个游戏纯属过去的部分作为诸位置的构成部分，我们应该以如下方式修改算法：开始，标志位置157-4，789-B，359-6作为B走到“失败”的位置，而且应用所提议的算法。现在的想法，是让选手A仅仅走到B的“失败”位置。但是，他还必须额外地预防重复以前的走法。一个“好”的计算机游戏程序，将使用一个随机数生成程序，以在若干胜利的做法之间进行选择；当它们有多于一个时，一个明显的办法将使作为A选手的计算机，恰恰在那些使A对B置于“失败”的诸做法当中，随机地进行选择。但是，有一些有趣的情况，其实这个貌似有



“军事游戏”的盘

理的过程竟至于失败！例如，考虑由A来走位置258-7；这是一个胜利的位置。从这个位置，选手A可以试探移动到158-7（按照本算法，它是一个对于B乃“失败”的位置）。但是，当B移到158-B，这就迫使A移到258-B，而后B回到258-7；他就赢了，因为前边的位置已经重复！这个例子说明，本算法必须在作成每着移动之后，再被重新调用——从以前已经标有“失败”（如果A走）或“胜利”（如果B走）的每个位置开始。

作者已经发现，这项游戏作成一个非常令人满意的计算机表演程序。

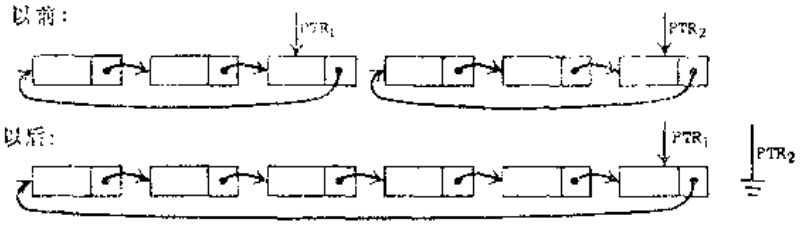
29.（a）如果FIRST=A，则没有什么可做的；否则置P←FIRST，而后反复地置P←LINK(P)以0次或多次，直到LINK(P)=A为止。最后置LINK(P)←AVAIL以及AVAIL←FIRST（而且也许还有FIRST←A）。（b）如果F=A，则没有什么可做的；否则置LINK(R)←AVAIL以及AVAIL←F（而且也许还有F←A，R←LOC(F)）。

2.2.4 小　　节

1. 否，它并无帮助，似乎倒是妨碍（如果有的话）（所述的约定不是与循环表原理

特别吻合的，除非我们置NODE(LOC(PTR))入此表以作为它的表头)。

2.



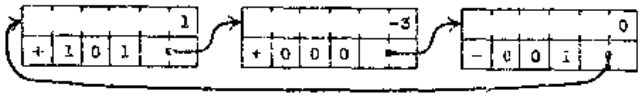
3. 如果 $PTR_1=PTR_2$，则唯一的效果是 $PTR_2\leftarrow A$。如果 $PTR_1\neq PTR_2$，则链接的交换把表分成两部分，就如同一个圆周通过在两点处切开而分成两部分一样；那么这个操作的第二部分使得 PTR_1 指向一个这样的循环表，即它是由那些节点组成的，如果在原来的表中，我们沿着链从PTR_1走到PTR_2，则这些节点已被遍历。

4. 设 HEAD 是表头的地址。为把 Y 压到堆栈上：置 P←AVAIL，INFO(P)← Y，LINK(P)←LINK(HEAD)，LINK(HEAD)← P。为把 Y 弹出堆栈：如果LINK(HEAD)=HEAD则 UNDERFLOW，否则置 P←LINK(HEAD)，LINK(HEAD)←LINK(P)，Y←INFO(P)，AVAIL⇐P。

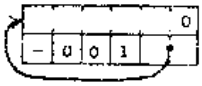
5. 如果 PTR≠A，则以 P← PTR，Q←LINK(PTR) 开始；然后置 R← LINK(Q)，LINK(Q)← P，P←Q，Q←R，并重复这个操作直到 P = PTR为止。

6.

a)



b)



7. 多项式中匹配的项，经扫过一遍本表即可找出，而不需要重复的随机检索。此外，递增的次序将与哨兵“－1”不相容。

8. 我们必须知道什么节点指向当前感兴趣的节点，如果我们正在删去这个节点或者在它的前头插入另一个节点的话。然而，有另外的方案：我们可删去 NODE(Q) 通过置Q2← LINK(Q)而后置NODE(Q)←NODE(Q2)，AVAIL⇐Q2；我们可以把 NODE(Q2) 插入NODE(Q) 的前头，通过首先交换NODE(Q2)⟷NODE(Q)，然后置 LINK(Q)←Q2，Q←Q2。这些灵活的技巧允许删去和插入，而不必知道哪个节点链接到 NODE(Q)；它们用于IPL早期的版本中。但它们有个缺点，就是在一个多项式的末尾的哨兵节点，有时将移动，而且其它的链接变量可能指向这个节点。

9. 对于 P = Q，算法A只不过加倍多项式（Q），恰如它所应有的那样。对于 P = M，算法M也给出预期的结果。对于 P = Q，算法M置多项式（P）← 多项式（P）乘以$(1+t_1)(1+t_2)\cdots(1+t_k)$，如果$M=t_1+t_2+\cdots+t_k$（尽管这并非立即地显然的）。当M = Q时，算法M令人惊奇地给出预期的结果。多项式（Q）← 多项式（Q）+ 多项式（Q）× 多项式（P），但当多项式（P）的常数项为－1时，这个计算失败。

10. 全然不变。唯一可能的差别将是在步骤 M2 中，撤消关于 A、B 或 C，可能个别地产生溢出的出错校验（这些出错校验未予确定，因为我们已经假定它们是不需要的）。换句话说，这一小节中的算法可以认为是对 $f(x^{b^2},x^b,x)$的运算，而不是对 $f(x,y,z)$的运算。

```
11.         COPY    STJ     9F      (注释留给读者去作)
            ENT3    9F
            LDA     1, 1
            1H      LD6     AVAIL
            J6Z     OVERFLOW
            LDX     1, 6 (LINK)
            STX     AVAIL
            STA     1, 6
            LDA     0, 1
            STA     0, 6
            ST6     1, 3 (LINK)
            ENT3    0, 6
            LD1     1, 1 (LINK)
            LDA     1, 1
            JANN    1B
            LD2     8F(LINK)
            ST2     1, 3 (LINK)
            9H      JMP     *
            8H      CON     0
```

12. 设所复写的多项式有 p 项。程序 A 运行 $(29p+13)u$，而且为使之成为一个公平的比较，我们应当加上建立一个 0 多项式的时间，例如，对于习题 14 为 $18u$。习题 11 的程序运行 $(21p+31)u$，大约是程序 A 运行时间的 $\frac{3}{4}$ 之多。

```
13.         ERASE   STJ     9F
                    LDX     AVAIL
                    LDA     1, 1(LINK)
                    STA     AVAIL
                    STX     1, 1(LINK)
            9H      JMP     *
14.         ZERO    STJ     9F
                    LD1     AVAIL
                    J1Z     OVERFLOW
                    LDX     1, 1(LINK)
                    STX     AVAIL
                    ENT2    0, 1
                    MOVE    1F(2)
                    ST2     1, 2(LINK)
        9H          JMP     *
        1H          CON     0
                    CON     -1(ABC)
15.     MULT        STJ     9F          子程序入口
                    LDA     5F          改变开关的状态
                    STA     SW1
                    LDA     6F
```

```
        STA     SW2
        STA     SW3
        JMP     *+2
2H      JMP     ADD             M2. 乘法循环
1H      LD4     1,1(LINK)       M1. 下一个乘子。M←LINK(M)
        LDA     1,4
        JANN    2B              如果ABC(M)≥0则到M2。
8H      LDA     7F              恢复开关的状态
        STA     SW1
        LDA     8F
        STA     SW2
        STA     SW3
9H      JMP     *               返回
5H      JNP     *+1             SW1的新状态
        LDA     0,1             COEF(P)
        MUL     0,4                 ×COEF(M)→rX
        LDA     1,1(ABC)        ABC(P)
        JAN     *+2
        ADD     1,4(ABC)            +ABC(M)，如果ABC(P)≥0
        SLA     2               移入rA的0:3场
        STX     0F              保存rX以供在SW2和SW3使用
        JMP     SW1+1
6H      LDA     0F              SW2，SW3的新状态
7H      LDA     1,1             SW1的通常状态
8H      LDA     0,1             SW2，SW3的通常状态
0H      CON     0               临时存储
```

16. 设 r 是多项式(M)的项数。这个子程序需要 $21pr+38r+29+27\Sigma m'+18\Sigma m''+27\Sigma p'+8\Sigma q'$，其中后边的几个求和指的是在程序A活动期间对应的数量。多项式(Q)的项数对于程序A的每次活动上升 $p'-m'$。如果我们作一个并非不合理的假设，即是 $m'=0$ 和 $p'=\alpha p$，其中 $0<\alpha<1$，则我们就得到等于 0、$(1-\alpha)pr$、αpr、以及 $rq'_0+\alpha p(r(r-1)/2)$ 各门的和，其中 q'_0 为 q' 在头一次迭代之值。总计为 $4\alpha pr^2+40pr+4\alpha pr+8q'_0r+38r+29$。这一分析指出，乘式应该比被乘式的项数要少些，因为我们要经常地跳过多项式(Q)中不匹配的项（关于更快的算法，见习题5.2.3-29)。

17. 实际上只有很小的优点；使用各种类型的表的加法和乘法子程序，实际上都是相同的。ERASE子程序（见习题13）的效率，显然是唯一重要的差别。

18. 设节点 x_i 的链接场包含 $\mathrm{LOC}(x_{i+1})\ominus\mathrm{LOC}(x_{i-1})$，其中"$\ominus$"表示减法或者"排斥的或"。两个相邻的表头被包括在循环表中，以助于使事件适当地开始（这项巧妙的技术，其起源却不清楚)。

2.2.5 小 节

1. 在左边插入Y：P←AVAIL；INFO(P)←Y；LLINK(P)←A；RLINK(P)←

LEFT；如果LEFT≠A，则LLINK(LEFT)←P．否则RIGHT←P；LEFT←P。置Y于左边并删去：如果LEFT＝A，则UNDERFLOW；P←LEFT；LEFT←RLINK(P)；如果LEFT＝A则RIGHT←A，否则 LLINK(LEFT)←A；Y←INFO(P)；AVAIL⇐P。

2．考虑逐次（在同一端）进行若干次删去的情况，在每次删去之后，我们必须知道下次要删什么。这意味着表中的链接之指向乃是背着表的该端。所以在两端删去就意味着链接必须在两个方向进行。

3．为了说明CALLUP对于CALLDOWN的无关性，注意，例如在表格1中的电梯在0393-0444的时间里不停在2层或3层上，尽管有人在等候着；这些人已经按了CALLDOWN，但如果他们已经按了CALLUP，则这电梯将停止。

为了说明CALLCAR与其它的无关性，注意在表1中，当门在1398的时间开始打开时，电梯已经判定为GOINGUP。如果 CALLCAR〔1〕＝CALLCAR〔2〕＝CALLCAR〔3〕＝CALLCAR〔4〕＝0，则根据步骤2，这时它在该点的状态应是NEUTRAL，但是实际上，CALLCAR〔2〕和 CALLCAR〔3〕已由电梯中的第7人和第9人置为1（如果我们想象对于所有层号都增加1的同一情况，则当门打开时，STATE＝NEUTRAL或是STATE＝GOINGUP之事实，将影响电梯是否也许将继续往下进行还是无条件地上行）。

4．如果有许多人离开同一层，则在所有这一时间里，STATE可以是NEUTRAL，而且当E9调用 DECISION 子程序时，在任何人抵达当前这一层之前．这可以置为一个新的状态。它确实是很少发生的（而且，它确实是作者在他进行的电梯实验期间所发现的最伤脑筋的现象）。

5．由门在1053的时间开始打开之时间，直到第7人在时间1183抵达为止，状态应是NEUTRAL，因为已经没有对第0层的呼叫。然后第7人将置CALLCAR〔2〕← 1并且状态将相应地变成为GOINGUP。

6．对于步骤M2和M4中的条件“FLOOR＝IN”，加上条件“如果OUT<IN则STATE≠GOINGUP；如果OUT>IN则STATE≠GOINGDOWN”。在步骤E4中，从QUEUE〔FLOOR〕接受人们，仅当他们站在电梯方向的前头时，除非STATE≠NEUTRAL（那时我们接受所有的到来者）；从QUEUE〔FLOOR〕未被接受的人们还应再次按CALLUP或CALLDOWN（因为在步骤M5中状态可改变）。

7．在行227，假定这个人是在WAIT表中。转移到M4A确保他停在那里。假定GIVEUPTIME不为0，而且它确实是100以至更多。

8．注释留给读者来作。

```
277     E8      DEC4    1
278             ENTA    61
279             JMP     HOLDC
280             LDA     CALL,4(3:5)
281             JAP     1F
282             ENT1    -2,4
283             J1Z     2F
284             LDA     CALL,4(1:1)
```

```
285             JAZ     E8
286     2H      LDA     CALL-1,4
287             ADD     CALL-2,4
288             ADD     CALL-3,4
289             ADD     CALL-4,4
290             JANZ    E8
291     1H      ENTA    23
292             JMP     E2A
9.  01  DECISION  STJ     9F                  存出口单元
    02            J5NZ    9F                  D1. 需要判断?
    03            LDX     ELEV1+2(NEXTINST)
    04            DECX    E1                  D2. 应该开门?
    05            JXNZ    1F                  如果电梯不在E1则跳转
    06            LDA     CALL+2
    07            ENT3    E3                  准备调度E3,如果
    08            JANZ    8F                      有对第2层的呼叫
    09            ENT1    -4                  D3. 有呼叫?
    10            LDA     CALL+4,1            寻找非0的呼叫变量
    11            JANZ    2F
    12  1H        INC1    1                   I1≡j-4
    13            J1NP    *-3
    14            LDA     9F(0:2)             所有CALL(j),j≠FLOOR,都为0
    15            DECA    E6B                 是否出口地址=行250?
    16            JANZ    9F
    17            ENT1    -2                  置j←-2
    18  2H        ENT5    4,1                 D4. 置STATE
    19            DEC5    0,4                 STATE←j-FLOOR
    20            J5NZ    *+2
    21            JANZ    1B                  j=FLOOR一般不允许
    22            JXNZ    9F                  D5. 电梯静止?
    23            J5Z     9F                  如果不在E1或如果j=2则跳转
    24            ENT3    E6                  否则调度E6
    25  8H        ENTA    20                  等候20个时间单位
    26            ST6     8F(0:2)             保存rI6
    27            ENT6    ELEV1
    28            ST3     2,6(NEXTINST)       置NEXTINST成E3或E6
    29            JMP     HOLD                调度这一活动
    30  8H        ENT6    *                   恢复rI6
    31  9H        JMP     *                   由子程序出口。
```

11. 开始，命 LINK[k] = 0，$1 \leqslant k \leqslant n$ 和 HEAD = −1。在改变 V[k]的一个模拟步骤期间，如果 LINK[k] ≠ 0 给出一个出错指示；否则置 LINK[k]←HEAD，HEAD←k 并置 NEWV[k]成为 V[k]的新值。在每个模拟步骤之后，置 k←HEAD，HEAD←−1，并反复下列操作 0 次或多次，直到 $k < 0$：置 V[k]←NEWV[k]，t←

LINK[k], LINK[k]$\leftarrow 0$, $k \leftarrow t$。

显然这个方法易于为散列的变量情况所采用，如果我们在每个与一个变址场V相关联的节点中，包括一个NEWV和LINK场。

12. WAIT表是从左到右删去的，但插入是从右到左地进行分类的（因为从这一边进行寻找，似乎要更短些）。而且当我们不知道正要删去的节点前驱或后继时，我们也在若干位置上从所有三个表中删去节点。仅仅是ELEVATOR表可以转换成一个单方向的表，而不失去很多有效性。

注意：在一个离散的模拟程序中，可能喜欢用一个非线性表来作为WAIT表，以减少进行"分入类中"所需要的时间。5.2.3小节讨论了维护优先排队，或"最小进，先出"表等诸如此类的一般问题。业已知道若干种方法，其中当在列表中仅有n个元素时，为进行插入或删去仅仅需要$O(\log n)$个操作，尽管当已知n很小时也不需要这样一个繁费苦心的方法。

2.2.6 小节

1. （注意下标从1变到n，而不是象在等式(5)中那样从0变到n。）LOC(A[0,0])$-2n$J$+2$K$=$LOC(A[J,K])，其中A[0,0]是一个假定的节点，实际上是不存在的。如果我们置J$=$K-1，则我们得到LOC(A[0,0])$+2n+2=$LOC(A[1,1])，所以可以以若干方式来表达这个答案。LOC(A[0,0])可以为负。

2. $$\mathrm{LOC}(\mathrm{A}[l_1,\cdots,l_k])=\mathrm{LOC}(\mathrm{A}[0,\cdots,0])+\sum_{1\leqslant r\leqslant k}a_r l_r$$
$$=\mathrm{LOC}(\mathrm{A}[l_1,\cdots,l_k])-\sum_{1\leqslant r\leqslant k}a_r l_r+\sum_{1\leqslant r\leqslant k}a_r,$$
其中 $a_r=\prod_{r<s\leqslant k}(u_s-l_s+1)$。

注意 有关对于出现于COBOL和PL/I语言中的结构之推广，以及为计算有关的常数的一个简单算法，见菲·德尔（P. Deuel），*CACM* 9 (1966), 344-347。

3. $1\leqslant k\leqslant j\leqslant n$当且仅当$0\leqslant k-1\leqslant j-1\leqslant n-1$；所以在对于下限0导出的所有公式中，分别地以$k-1$，$j-1$，$n-1$来代替$k$，$j$，$n$。

4. LOC(A[J,K])$=$LOC(A[0,0])$+n$J$-$J(J$-1)/2+$K。

5. 命AO$=$LOC(A[0,0])。假定J在r11和K在r12中，至少有两个解。(1)"LDA TA2,1:7"，这里单元TA2$+j$是"NOP $j+1*j/2+$AO, 2"；(2)"LDA C1,7:2"，这里单元C1包含"NOP TA,1:7"而且单元TA$+j$写着"NOP $j+1*j/2+$AO"。后者花去多一个的循环，但是不把这个表格限制于变址寄存器2。

6. (a) $$\mathrm{LOC}(\mathrm{A}[I,J,K])=\mathrm{LOC}(\mathrm{A}[0,0,0])+\binom{I+2}{3}+\binom{J+1}{2}+\binom{K}{1}$$
(b) $$\mathrm{LOC}(\mathrm{B}[I,J,K])=\mathrm{LOC}(\mathrm{B}[0,0,0])+\binom{n+3}{3}-\binom{n+3-I}{3}+\binom{n+2-I}{2}$$
$$-\binom{n+2-J}{2}+K-J$$
因此在这种情况下，所述的形式也是可能的。

7. $$\mathrm{LOC}(\mathrm{A}[l_1,\cdots,l_k])=\mathrm{LOC}(\mathrm{A}[0,\cdots,0])+\sum_{1\leqslant r\leqslant k}\binom{l_r+k-r}{1+k-r}。$$

见习题1.2.6-56。

8. 设对于 $0 \leqslant I \leqslant n+1$，$0 \leqslant J \leqslant n+1$，$0 \leqslant K \leqslant n+2$ 定义 X[I,J,K]。我们可以命 A[I,J,K]=X[I,J,K]，B[I,J,K]=X[J,I+1,K]，C[I,J,K]=X[I+1,K,J+1]，D[I,J,K]=X[K,I+1,J+1]，E[I,J,K]=X[J,K,I+1]，F[I,J,K]=X[K,J+1,I+2]。显然这些定义的每一个，都落在X数组的限定范围之内；我们必须证明没有重迭，而且如果我们在六个定义的每个当中，把X的下标更名成为 i、j、k，则这是容易做到的：然后我们分别地求得 $i \geqslant j \geqslant k$，$j > i \geqslant k$，$i \geqslant k > j$，$j \geqslant k > i$，$k > i \geqslant j$，$k > j > i$ 而且这些条件证明了没有重迭。然而，上述方法并不脱离填入的X数组：元素 X[n+1,J,K]和 X[J,K,n+2]对于 $0 \leqslant K \leqslant J \leqslant n+1$ 是不用的。当 $n \geqslant 2$，看起来不可能挑出任何比这更紧凑的而且在每个下标中仍然保持线性的编址了，尽管还未找到不可能性的证明。某些下标可以运行于负的值。当 $n=0$ 或 1 时，问题是平凡而没有趣味的。

上边的方法使用单元 X[0,0,0]到 X[n+1,n,n-1]，其间减去少量的空。全部共有 $(n-1)(n+2)(n+3)$ 个元素的这个数组，已经装入 $(n+2)(n+2)(n+3)-(n+4)$ 个连续的单元中。还有更好的解吗？

9. **G1**. 把指针变量 P1，P2，P3，P4，P5，P6分别置为表 FEMALE，A21，A22，A23，BLOND，BLUE 的头一个单元。以下假定，每个表的末端以链接A给出，而且A比任何其它的链接都要小。如果 P6=A，则停止（不幸，表是空的）。

G2.（下列诸动作可以做许多可能的排序；我们首先选择检查 EYES，然后依次进行检查 HAIR，AGE，SEX。）置 P5←HAIR(P5) 0 次或多次直到 P5<P6 为止。如果现在 P5<P6，则转到步骤 G5。

G3. 置P4←AGE(P4)，如果有必要就重复，直到 P4≤P6 为止。类似地对 P3 和 P2 也做同样的事情直到 P3≤P6 和P2≤P6 为止。如果现在 P4，P3，P2 全都小于 P6，则转到 G5。

G4. 置 P1←SEX(P1)直到 P1≤P6 为止。如果 P1=P6，则我们已经找到所希望的女孩中的一个，所以输出她的地址，P6(她的年龄可由 P2，P3，和 P4 的状态确定之)。

G5. 置 P6←EYES(P6)。如果 P6=A，则现在就停止；否则返回 G2。 ∎

这个算法是有趣的，但不是对于这样一种寻找来组织一个表的最好的方法。

10. 在试验了许多不同的似乎有效的方案而且分析了它们的效率之后，作者感到，似乎没有比把所有的人分成 n 个近乎相等的组更好的方法了，这里 n 依据可利用的空间总额尽可能地大，而分组的办法是这样的：一个人的特征确定他所在的组；然后对于所希望的特征来寻找相应组内的每个人（关于进一步的讨论，见 6.5 节）。

11. 至多 200+200+3·4·200=2800 个字。

12. VAL(QO)=c，VAL(PO)=b/a，VAL(P1)=d。

13. 在每一个表的末尾有一个哨兵是方便的，这个哨兵“比喻者”在某个对表进行编序的场中之“最低者”。已经能使用一个直接的单向的表，例如，通过在 BASEROW(I) 中仅保留 LEFT 链接和在 BASECOL 中保留 UP 链接，通过修改算法成为：S2，在置 J←COL(PO)之前测试是否 PO=A，而且如是则置 PO←LOC(BASEROW(IO)) 并转到 S3。S3，测试是否 QO=A，而且如果是，则结束。S4，类似于 S2 中的变化。S5，测试是否 P1=A，而且如果是，则把这处理成如同 COL(P1)<0 那样。S6，测试是否 UP(PTR(J))=A，而且如果是，则就象它的 ROW 场为负那样处理之。

这些修改使得算法更为复杂，而且并不节省内存空间，但表头中的 ROW 或 COL 场除外（在 MIX 的情况下，这全然不节省）。

14. 人们可以首先把在主要行中有非 0 元素的那些列链接在一起，以使所有其它的列，随着我们以每行为主要时可被跳过。主要列为 0 的那些行立即被跳过。

15. 命 rI1≡PIVOT；rI2≡P0；rI3≡Q0；rI4≡P；rI5≡P1．X；LOC(BASEROW(*i*))≡BROW＋*i*；LOC(BASECOL(*j*))≡BCOL＋*j*；PTR(*j*)≡BCOL＋*j*(1:3)。

```
01    ROW         EQU     0:3
02    UP          EQU     4:5
03    COL         EQU     0:3
04    LEFT        EQU     4:5
05    PTR         EQU     1:3
06    PIVOTSTEP   STJ     9F              子程序入口，rJ1 = PIVOT
07    S1          LD2     0,1(ROW)        S1. 初始化
08                ST2     I0              I0←ROW(PIVOT)
09                LD3     I,1(COL)
10                ST3     J0              J0←COL(PIVOT)
11                LDA     =1.0=           浮点常数 1
12                FDIV    2,1
13                STA     ALPHA           ALPHA←1/VAL(PIVOT)
14                LDA     =1.0=
15                STA     2,1             VAL(PIVOT)←1
16                ENT2    BROW,2          P0←LOC(BASEROW(I0))
17                ENT3    BCOL,3          Q0←LOC(BASECOL(J0))
18                JMP     S2
19    2H          ENTA    BCOL,1
20                STA     BCOL,1(PTR)     PTR(J)←LOC(BASECOL(J))
21                LDA     2,2
22                FMUL    ALPHA
23                STA     2,2             VAL(P0)←ALPHA×VAL(P0)
24    S2          LD2     1,2(LEFT)       S2. 处理主要行。P0←LEFT(P0)
25                LD1     1,2(COL)        J←COL(P0)
26                J1NN    2B              如果 J≥0，则处理 J
27    S3          LD3     0,3(UP)         S3. 找新行。Q0←UP(Q0)
28                LD4     0,3(ROW)        rI4←ROW(Q0)
29    9H          J4N     *               如果 rI4＜0，则出口
30                CWP4    I0
31                JE      S3              如果 rI4 = I0，则重复
32                ST4     I(ROW)          I←rI4
33                ENT4    BROW,4          P←LOC(BASEROW(I))
34    S4A         LD5     1,4(LEFT)       P1←LEFT(P)
35    S4          LD2     1,2(LEFT)       S4. 找新列。P0←LEFT(P0)
36                LD1     1,2(COL)        J←COL(P0)
37                CMP1    J0
```

```
38                      JE      S4              重复，如果 J = JO
39                      ENTA    6,1
40                      SLA     2               rA(0:3)←J
41                      JNN     S5
42                      LDAN    2,3             如果 J < 0，则
                                                    置 VAL(QO) ← - ALPHA × VAL(PO)
43                      FMUL    ALPHA
44                      STA     2,3
45                      JMP     S3
46      1H              ENT4    0,5             P ← P1
47                      LD5     1,4(LEFT)       P1 ← LEFT(P)
48      S5              CMPA    1,5(COL)        S5. 找 I,J 元素
49                      JL      1B              循环直到 COL(P1) ≤ J
50                      JE      S7              如果 =，则转 S7
51      S6              LD6     BCOL,4(PTR)     S6. 插入 I,J 元素。 rI5 ← PTR[J]
52                      LDA     J               rA(0:3) ← J
53      2H              ENT6    0,5             rI6 ← rI5
54                      LD5     0,6(UP)         rI5 ← UP(rI6)
55                      CMPA    1,5(COL)
56                      JL      2B              如果 COL(rI5) ← 6则跳转
57                      LD5     AVAIL           X ← AVAIL
58                      J5Z     OVERFLOW
59                      LDA     0,5(UP)
60                      STA     AVAIL
61                      LDA     0,6(UP)         UP(PTR[J])
62                      STA     0,5(UP)         → UP(X)
63                      LDA     1,4(LEFT)       LEFT(P)
64                      STA     1,5(LEFT)       → LEFT(X)
65                      ST1     1,5(COL)        COL(X) ← J
66                      LDA     I(ROW)
67                      STA     0,5(ROW)        ROW(X) ← I
68                      STZ     2,5             VAL(X) ← 0
69                      ST5     1,4(LEFT)       LEFT(P) ← X
70                      ST5     0,6(UP)         UP(PTR[J]) ← X
71      S7              LDAN    2,3             S7. 主要步骤。 - VAL(QO)
72                      FMUL    2,2               × VAL(PO)
73                      FADD    2,5                 + VAL(P1)
74                      JAZ     S8              如果失去意义，则到 S8
75                      STA     2,5             否则存入 VAL(P1)
76                      ST5     BCOL,1(PTR)     PTR[J] ← P1
77                      ENT4    0,5             P ← P1
78                      JMP     S4A             P1 ← LEFT(P)，到S4
79      S8              LD6     BCOL,1(PTR)     S8. 删去 I，J 元素。 rI6 ← PTR[J]
80                      JMP     * + 2
```

```
81    LD6    0,6(UP)      rI6←UP(rI6)
82    LDA    0,6(UP)
83    DECA   0,5          UP(rI6)=P1 吗?
84    JANZ   *-3          循环直到相等
85    LDA    0,5(UP)
86    STA    0,6(UP)      UP(rI6)←UP(P1)
87    LDA    1,5(LEFT)
88    STA    1,4(LEFT)    LEFT(P)←LEFT(P1)
89    LDA    AVAIL        AVAIL←P1
90    STA    0,5(UP)
91    ST5    AVAIL
92    JMP    M4A          P1←LEFT(P)，到S4
```

注意：用第4章的约定，行71—74实际上将被编码成

LDA2,3; FMUL 2,2; FCMP 2,5; JE S8; STA TEMP; LDA2,5; FSUB TEMP;

而且在单元0中还有一个适当的参数EPSILON。

18. $k=1$，主要列3，我们得到

$$\begin{pmatrix} \frac{1}{3} & \frac{2}{3} & \frac{1}{3} \\ -\frac{2}{3} & -\frac{1}{3} & -\frac{2}{3} \\ -\frac{1}{3} & -\frac{2}{3} & -\frac{1}{3} \end{pmatrix}$$

$k=2$，主要列1，我们得到

$$\begin{pmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{3}{2} & -\frac{1}{2} & 1 \\ -\frac{1}{2} & -\frac{1}{2} & 0 \end{pmatrix}$$

$k=3$，主要列2，我们得到

$$\begin{pmatrix} 0 & 1 & 0 \\ -2 & 1 & 1 \\ 1 & -2 & 0 \end{pmatrix}$$

经最后的置换之后，我们有答案

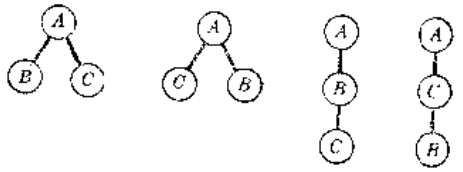
$$\begin{pmatrix} 1 & -2 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

20. $a_0=\mathrm{LOC}(A[1,1])-3$，$a_1=1$ 或 2，$a_2=3-a_1$。

2.3 节

1. 有三种方法来选择根。一旦已经选定了根，比如说 A，有三种方法把剩下的节点分成子树：$\{B\}$、$\{C\}$；$\{C\}$，$\{B\}$；$\{B,C\}$。在后边的情况下，有两种方法$\{B,C\}$作成

一个树，依赖于哪一个节点是它的根。因此，当A是根时，我们得到四个树：



而且全部有12个。在2.3.4.4小节中，对于任何节点数n，解决了这个问题；方式数一般为$(2n-2)!/(n-1)!$。

2. 习题1答案中的头两个树作为有向树而言是相同的，所以在这种情况下，我们仅得到9种不同的可能性。关于一般的解答，见2.3.4.4小节，其中证明了公式n^{n-1}。

3. 部分1：证明至少有一个这样的序列。设树有n个节点。当$n=1$时，结果是显然的，因为X必然是根。如果$n>1$，则这定义蕴含着有一个根X_1和子树$T_1,T_2,\cdots,T_m$；或者$X=X_1$，或者X是唯一的T_j之一成员。在后一种情况下，由归纳法，有一条通路$X_2,\cdots,X$，其中X是T_j的根，而且由于X_1是X_2的父亲，我们乃有一条通路$X_1,X_2,\cdots,X$。

部分2：证明至多有一个这样的序列。我们将通过归纳法证明，如果X不是树的根，则X有一个唯一的父亲（于是X_k确定X_{k-1}，确定X_{k-2}，等等）。如果树有一个节点，则没有什么可证的；否则X在唯一的T_j中。或者X是T_j的根，在这种情况下，由定义，X有唯一的父亲；或者X不是T_j的根，在这种情况下，由归纳法，X在T_j中有唯一的父亲，而且在T_j之外没有节点能成为X的父亲。

4. 真的（很遗憾）。

5. 4。

6. 设$\text{father}^0(X)$表示X，$\text{father}^1(X)$表示X的父亲，$\text{father}^2(X)=\text{father}(\text{father}(X))$＝$X$的祖父，$\text{father}^k(X)=\text{father}(\text{father}^{k-1}(X))=$“$X$的(曾)$^{k-2}$祖父”。堂兄弟关系的条件即是$\text{father}^{m+1}(X)=\text{father}^{n+1}(Y)$，但$\text{father}^m(X)=\text{father}^{m+n}(X)$；或者，如果$n>0$，则有可能是其中交换了$X$，$Y$的该同一条件。

7. 我们来观察一个在X与Y之间的非对称条件：使用习题6的条件，并且约定如果j或k之一（或二者都）为-1时，则$\text{father}^j(X)\neq\text{father}^k(Y)$。为了证明对于某个唯一的$m$和$n$，这个关系总是正确的，试考虑$X$和$Y$的杜威记数法，即是$1.a_1.\cdots.a_p.b_1.\cdots.b_q$和$1.a_1.\cdots.a_p.c_1.\cdots.c_r$，其中$p\geq 0$，$q\geq 0$，$r\geq 0$以及(如果$qr\neq 0$)$b_1\neq c_1$。任何一对节点的编号，都可以以这种形式写出，而且显然我们必须取$m=q-1$和$n+m=r-1$。

8. 没有真正是一个树的二叉树；这两个概念是完全独立的，尽管一个非空二叉树的图式可能看起来象是树似的。

9. A是根，因为我们习惯上把根放在顶上。

10. 嵌套集合的任何有限的集体，如下对应于正文中所定义的那样一个森林：设$A_1,\cdots,A_n$是不含于其它当中的集体之集合。对于固定的j，包含于A_j中的所有集合的子集体是嵌套的，而且因此我们可以假定这个子集体对应于一个以A_j为根的（无序）树。

11. 在一个嵌套的集体$\mathfrak{S}$中，命$X\equiv Y$，如果有某个$Z\in\mathfrak{S}$使得$X\cup Y\subseteq Z$。这一关系显然是反身的和对称的，而且事实上它是一个等价关系，因为$W\equiv X$，$X\equiv Y$意味着在$\mathfrak{S}$中有Z_1，Z_2同时$W\subseteq Z_1$，$X\subseteq Z_1\cap Z_2$，$Y\subseteq Z_2$。由于$Z_1\cap Z_2\neq\varnothing$，故或者$Z_1\subseteq Z_2$或者

$Z_2 \subseteq Z_1$，因此 $W \cup Y \subseteq Z_1 \cap Z_2 \in \mathfrak{S}$。现在如果$\mathfrak{S}$是一个嵌套的集体，则通过规则“$X$是$Y$的一个祖宗，而且$Y$是$X$的一个后裔，当且仅当，$X \supset Y$”定义一个对应于$\mathfrak{S}$的有向森林。$\mathfrak{S}$的每个等价类对应于一个有向树，它是对于所有$X$，$Y$使得$X \equiv Y$的一个有向森林（如此我们就推广了森林和树的定义，这些对于有限的集体是已经给出了的）。用这些术语，我们可以把X的层次定义作祖宗(X)的基数。类似地，X的次数是后裔(X)在嵌套集体中等价类的基数。我们说X是Y的父亲，以及Y是X的一个儿子，如果X是Y的一个祖宗，而且没有Z使得$X \supset Z \supset Y$。为了得到有序树和森林，试以某种特别的方式将上述的等价类编序之，例如通过将关系$\subseteq$嵌入到线性次序中。

例(a)：设 $S_{\alpha k} = \{x \mid x =$ 十进记号下的 $.d_1 d_2 d_3 \cdots$，其中 $\alpha =$ 十进记号下的 $.e_1 e_2 e_3 \cdots$，而且 $d_j = e_j$，如果 $j \bmod 2^k \neq 0\}$。集体 $\mathfrak{S} = \{S_{\alpha k} \mid k \geqslant 0,\ 0 \leqslant \alpha < 1\}$ 是嵌套的，并且给出一个树，具有无限多的层次，且对于每个节点具有不可数的次数。

例(b)，(c)：在平面中，代替借助于实数，定义这样的集合是方便的，而且这是充分的，因为在平面与实数之间有一一对应关系。命 $S_{\alpha mn} = \{(\alpha, \beta) \mid m/2^n \leqslant \beta < (m+1)/2^n\}$，且命 $T_\alpha = \{(\gamma, \beta) \mid \gamma \leqslant \alpha\}$。集体 $\mathfrak{S} = \{S_{\alpha mn} \mid 0 < \alpha < 1,\ n > 0,\ 0 \leqslant m < 2^n\} \cup \{T_\alpha \mid 0 < \alpha < 1\}$ 容易看出是嵌套的。$S_{\alpha mn}$ 的儿子是 $S_{\alpha(2m)(n+1)}$ 和 $S_{\alpha(2m+1)(n+1)}$，而且 T_α 有儿子 $S_{\alpha 00}$ 加上子树 $\{S_{\gamma mn} \mid \gamma < \alpha\} \cup \{T_\gamma \mid \gamma < \alpha\}$。所以每个节点有次数 2，而且每个节点有不可数的形如 T_α 的祖宗。这个构造是由理·比奇洛（R. Bjgelow）给出的。

注意：如果我们取实数的一个适当的良序，而且如果我们定义 $T_\alpha = \{(\alpha, \beta) \mid \gamma > \alpha\}$，则我们可以稍微地改进这个构造，得到一个嵌套的集体，其中每个节点有次数 2，有不可数的层次，以及 2 个儿子。

12. 我们对于部分编序附加另外一个条件（类似于“嵌套集合”的条件）以保证它对应于一个森林：如果 $x \preceq y$ 且 $x \preceq z$，则或者 $y \preceq z$ 或者 $z \preceq y$。为做成一个树，也断言对于所有的 x，存在一个节点 r 使得 $x \preceq r$。这给出一个如同正文中所定义的那样的无序树，当节点数有限时，其证明象证明习题10中的嵌套集合那样进行之。

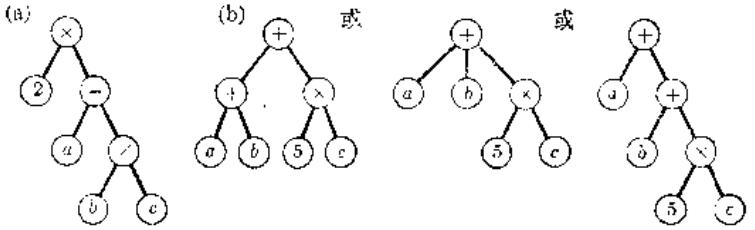
13. $a_1, a_1.a_2, \cdots, a_1.a_2.\cdots.a_k$。

14. 由于S是非空的，它包含一个元素“$1.a_1.\cdots.a_k$”，其中k尽可能小；如果$k > 0$，则我们也取a_k在S中尽可能小，而且我们立即看出，k必须为0，即是，S含有元素“1”。设这是根。所有其它的元素有$k > 0$，所以S的剩下的元素可以划分成集合 $S_j = \{1.j.a_2.\cdots.a_k\}$，$1 \leqslant j \leqslant m$，对于某个 $m \geqslant 0$。如果 $m \neq 0$，且 S_m 为非空集合，则通过如上的推理，我们发现对于每个 S_j，“$1.j$”是在 S_j 中，所以每个 S_j 是非空的。于是不难看出，集合 $S_j' = \{1.a_2.\cdots.a_k \mid 1.j.a_2.\cdots.a_k$ 是在 S_j 中$\}$ 满足与S所满足的相同的条件，所以由归纳法，每个 S_j 也形成一个树。

15. 设这根为“1”而且设α的左子树的根为$\alpha.0$；α的右子树的根可以命名为$\alpha.1$。例如在图19(a)中，King Christian IX出现在两个位置中，即1.0.0.0.0和1.1.0.0.1.0。为了简便，我们可以去掉小数点，而仅仅写10000和110010。注意：这个记号是由法国人高尔顿（Galton）给出的；见《自然继承》（Natural Inheritance)（麦克米伦，1889)，249。对于“家系”，使用F和M来代替0和1是更好记的；例如，Christian IX是Charles的MFFMF，即是Charles的母亲的父亲的父亲的母亲的父亲。0和1的约定由于另一个

原因，是有趣的，因为它向我们提供了一个重要的在一个二叉树的节点与以二进系统（即是在一台计算机内的内存地址）表示的正整数之间的对应关系。

16.



17. root(T)$=A$; root (T[2])$=C$; root(T[2, 2])$=E$。

18. L[5, 1, 1]= "(2)"。L[3, 1]是无意义的，因为L[3]是一个空列表。

19. $*$[L]　　L[2]= "(L)"; L[2, 1, 1]= "a"。



20.（直观上，通过撤消 b 树的所有终端节点，得到 b 树与二叉树之间的对应；请看 2.3.4.5 小节中重要的构造）。命具有一个节点的 b 树对应于空的二叉树；并命具有一个以上节点的 b 树——因此它由一个根 r 和 b 树形 T_1 和 T_2 组成，对应于具有根 r，左子树 T_1' 和右子树 T_2' 的二叉树，这里 T_1 和 T_2 分别对应于 T_1' 和 T_2'。

21. $1+0\cdot n_1+1\cdot n_2+\cdots+(m-1)\cdot n_m$。证明：树中的节点数为 $n_0+n_1+n_2+\cdots+n_m$，而且这也等于 1+(树中的儿子数)$=1+0\cdot n_0+1\cdot n_1+2\cdot n_2+\cdots+m\cdot n_m$。

2.3.1 小节

1. INFO(T)$=A$, INFO(RLINK(T))$=C$, 等等；答案为H。

2. 先根次序：1245367；对称次序：4251637；后根次序：4526731。

3. 命题为真（注意，例如节点 4, 5, 6, 7 在习题 2 中总以这种次序出现）。通过对二叉树的大小用归纳法，即可证明这一结果。

4. 它是后根次序的逆（用归纳法容易证明该事实）。

5. 例如在习题 2 的树中，先根次序为（用二进记号，在这种情况下它等价于杜威系统）1, 10, 100, 101, 11, 110, 111。如同在字典中那样，这可认为是从左到右的分类。

一般地说，节点将以先根次序来列出，如果它们被从左到右地以词典编辑法进行分类，而且把“空格”处理作小于 0 或 1。节点将以后根次序列出，如果它们以 $0<1<$“空格”的次序，按词典编辑法进行分类。对于中根次序，使用 $0<$“空格”<1。

6. $p_1p_2\cdots p_n$ 可由一堆栈得到这一事实，通过对 n 用归纳法，容易加以证明，或者事实上我们可以观察到，算法 T 精确地做着在其堆栈的动作中所要求做的那些工作（如同在习题 2.2.1-3 中的诸 S 和诸 X 的相应序列，与在二重次序下的下标那样的诸 1 和诸 2 的序列是同样的，见习题 18）。

反之，如果 $p_1p_2\cdots p_n$ 可由一个堆栈得到，而且如果 $p_k=1$，则 $p_1\cdots p_{k-1}$ 是 $\{2,\cdots k\}$ 的一个排列，而且 $p_{k+1}\cdots p_n$ 是 $\{k+1,\cdots,n\}$ 的一个排列，它们中每一个都可由堆栈得到，而且它们是对应于左和右子树的排列。现在用归纳法来进行证明。

7. 由先根次序，根就知道了；然后由中根次序，我们知道左子树和右子树；而且事实上我们知道在后诸子树中节点的先根次序与中根次序。因此，这树容易被构造出来（而

且事实上这是十分有趣的，来构造一个简单的算法，它以通常的方式把树链接在一起，从把节点以先根次序在 LLINK 中链接在一起且以中根次序在 RLINK 中链接在一起开始。类似地，后根次序与中根次序在一起表征这一结构。但先根次序与后根次序则不然，有着以 "AB" 作为先根次序和以 "BA" 作为后根次序的两个二叉树。如果一个二叉树的所有非终端节点都有两个非空的分枝，则它的结构即由先根次序与后根次序所表征。

8. (a)具有所有的 LLINK 为空的二叉树。(b)具有 0 个或 1 个节点的二叉树。(c)具有所有的 RLINK 为空的二叉树。

9. T1 一次，T2 $(2n+1)$ 次，T3 n 次，T4 $(n+1)$ 次，T5 n 次。这由归纳法或由克希霍夫定律导出，或者通过验证程序 T 导出。

10. 所有 RLINK 为空的一个二叉树，将使得在撤消任何一个节点之前，把所有 n 个节点的地址都放进堆栈。

11. 命 a_{nk} 是具有 n 个节点的二叉树的个数，其中算法 T 中的堆栈决不含有 k 个条款以上。如果 $g_k(z)=\sum_n a_{nk}z^n$，则我们求得 $g_1(z)=1/(1-z)$，$g_2(z)=1/(1-z/(1-z))=(1-z)/(1-2z)$，…，$g_k(z)=1/(1-zg_{k-1}(z))=q_{k-1}(z)/q_k(z)$，其中 $q_{-1}(z)=q_0(z)=1$，$q_{k+1}(z)=q_k(z)-zq_{k-1}(z)$；因此 $g_k(z)=(f_1(z)^{k+1}-f_2(z)^{k+1})/(f_1(z)^{k+2}-f_2(z)^{k+2})$，其中 $f_1(z)=\frac{1}{2}(1\pm\sqrt{1-4z})$。现在可以证明，$a_{nk}$ 是 $(1-u)(1+u)^{2n}(1-u^{k-1})/(1-u^{k+2})$ 中 u^n 的系数；因此 $s_n=\sum_{k\geq 1}k(a_{nk}-a_{n(k-1)})$ 是在 $(1-u)^2(1+u)^{2n}\sum_{j\geq 1}u^j/(1-u^j)$ 中 u^{n+1} 的系数，减去 a_{nn}。习题 5.2.2-52 中的技巧现在得到渐近序列

$$s_n/a_{nn}=\sqrt{\pi n}-1.5+\frac{11}{24}\sqrt{\frac{\pi}{n}}+O(n^{-3/2})$$

〔罗·赛·里德 (R.C.Read) 编的《图论与计算》(Graph Theory and Computing)(纽约：科学出版社，1972) 一书中尼·戈·德·布鲁因，唐·欧·克努特以及斯·奥·赖斯(S.O.Rice)的文章，15～22。〕

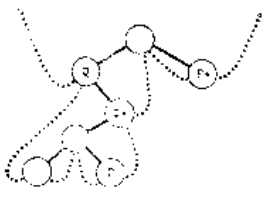
12. 在步骤 T2 与 T3 之间，代替在步骤 T4 与 T2 之间，访问节点 NODE(P)。为了证明，说明 "在步骤T2，以…原始值 $A[1]\cdots A[m]$ 开始"，实际上和正文中一样。

13. 〔S.阿拉乌杰(S.Arauja)的解〕设步骤 T1 到 T4 不变，但在步骤 T1 中给新变量 Q 的初值为 Λ。如果有的话，Q 将指向最后访问的节点。步骤 T5 变成两个步骤："T5〔右分支完了？〕如果 RLINK(P)=Λ 或 RLINK(P)=Q，转向 T6；否则置 $A\Leftarrow P$，$P\leftarrow$ RLINK(P) 并返回 T2。T6〔访问P〕"访问" NODE(P)，置 $Q\leftarrow P$，并返回 T4'。可应用类似的证明。

14. 由归纳法，总是恰有 $n+1$ 个 Λ 链接 (T 计入，当它为空时)。有 n 个非空的链接，T 计入，因此正文中关于空链居多的陈述是正确的。

15. 有一个指向一个节点的穿线的 LLINK，当且仅当，它有一个非空的右子树；有一个指向一个节点的穿线的 RLINK，当且仅当，它的左子树是非空的 (见图 24)。

16. 如果 LTAG(Q)="+"，则 $Q^*=$LLINK(Q)，即是往左边下一步。否则，通过在树中往上重复 (如果必要的话) 进行，直到头一次有可能往右下行而不走回头路为止，来得到 Q^*。典型的例子是在下列树中从 P 到 P^* 和从 Q 到 Q^* 的旅行：



17. 如果 LTAG(P)="+"，则置 Q←LLINK(P) 并终止；否则置Q← P，而且现在置 Q←RLINK(Q) 0 次或多次直到找到 RTAG(Q)="+" 为止，并且最后另外置 Q←RLINK(Q) 一次。

18. 通过插入步骤 T2a"访问 NODE(P) 头一次"，来修改算法T；而且在步骤 T5中，我们第二次访问 NODE(P)。

给定一个穿线的树，遍历是极为简单的：

$(P,1)^{A}=$(LLINK(P), 1) 如果 LTAG(P)="+"，否则 (P, 2)；

$(P,2)^{A}=$(RLINK(P), 1) 如果RTAG(P)="+"，否则 (RLINK(P), 2)。

在每种情况下，我们在树中至多移动一步；所以实际上，双重次序和 d 与 e 之值已被嵌入程序中而不详述了。

删掉所有的"头一次访问"恰巧给予我们算法T和S；删掉"第二次访问"给了我们习题 12 和 17 的解。

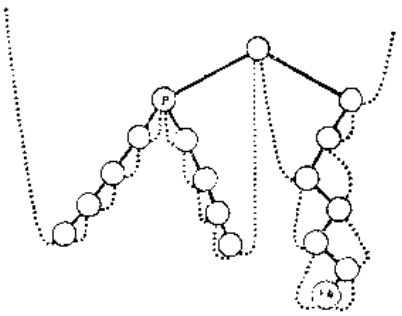
19. **E1**. 置Q← P。如果Q=HEAD，则转到E5。

E2. 如果 RTAG(Q)="+"，则置Q←RLINK(Q)并重复这一步骤。

E3. 置Q←RLINK(Q)。如果现在 LLINK(Q)=P，则转到E5；否则置Q←LLINK(Q)。

E4. 如果 RLINK(Q)≠P，则置Q←RLINK(Q) 并重复这个步骤；否则终止本算法。

E5. 如果 RTAG(Q)="+"，则用算法S置Q←Q S并重复这个步骤。



注意 似乎有比这个更有效的算法（例如考虑所示图形中的P和P #），而且对于在后根次序下遍历整个树而言，它劣于一个使用堆栈（象在习题 13 那样）的算法。

20. 行06-09代之以：

```
T3  ENT   0,6
    LD6   AVAIL
    J6Z   OVERFLOW
    LDX   0,6(LINK)
    STX   AVAIL
    ST5   0,6(INFO)
    ST4   0,6(LINK)
```

行 12-13 代之以：

```
    LD4   0,6(LINK)
```

```
        LD5   0,6(INFO)
        LDX   AVAIL
        STX   0,6(LINK)
        ST6   AVAIL
        ENT6  0,4
```

如果在行06处再加两行代码

```
T3      LD3   0,5(LINK)
        J3Z   T5              如果LLINK(P) = Λ，则转T5。
```

并在行10和11中作些适当的改变，则运行时间将从30n + a + 4减少到27a + 6n − 22个单位（如果我们置a =(n + 1)/2，则这同一设计将把程序T的运行时间减少到12a + 6n − 7，这是一点改进）。

21. 下列算法事实上可用作在三种次序中的任何一种之下的遍历，即使有"共享的"子树亦然：

V1.〔初始化〕 置P←LOC(T)，Q←T。如果Q=Λ，则终止本算法。

V2.〔先根次序访问〕 如果在先根次序下进行遍历，则访问NODE(Q)。

V3.〔转向左〕 置R←LLINK(Q)。如果R≠Λ，则置LLINK(Q)←P，P←Q，Q←R，并返回到V2。（假定RTAG(P)开始时为"+"。）

V4.〔中根次序访问〕 如果在中根次序下遍历，则访问NODE(Q)。

V5.〔转向右〕 置R←RLINK(Q)。如果R≠Λ，则置RTAG(Q)←"−"，RLINK(Q)←P，P←Q，Q←R，转到V2。

V6.〔后根次序访问〕 如果在后根次序下遍历，则访问NODE(Q)。

V7.〔上升〕 如果P=LOC(T)，则终止本算法。否则如果RTAG(P)="+"，则置R←LLINK(P)，LLINK(P)←Q，Q←P，P←R，并转到V4。否则置R←RLINK(P)，RTAG(P)←"+"，RLINK(P)←Q，Q←P，P←R，并转到V6。

同这一算法有关的一些算法，将在2.3.5小节中作进一步的讨论。实际上有可能不使用附加的RTAG位来解决这个问题，使用一个由约·迈·罗布逊给出的巧妙的思想。他保留一个指向一些节点的指针的附加的堆栈，这些节点有一个非空的左子树，并使得它们的右子树当前正被访问。使用节点中的链接场，其中有LLINK=RLINK=Λ，就有余地来维持这样一个堆栈：

劳·西科洛西（L.Siklo'ssy）已经发现了遍历一个树而不需要辅助堆栈又不需要改变内存中数据的一个有趣的方法，它是通过把习题2.2.4-18的方法推广到树来实现的。〔《信息处理文集》(Information Processing Letters) 1 (1972)，149～152。〕

22. 命rI4≡R，rI5≡Q，rI6≡P，并使用程序T和S的其它约定：

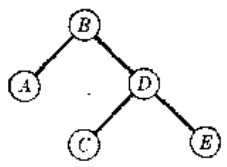
```
V1  ENT6  T              1        V1. 初始化. P←LOC(T).
    LD5   T(LLINK)       1        Q←T.
    J5NZ  V3             1
    JMP   DONE           0        对于空树的特殊出口
1H  ST6   0,5(LLINK)     a − 1    LLINK(Q)←P
2H  ENT6  0,5            n − 1    P←Q
```

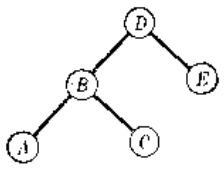
```
        ENT5  0,4          n - 1     Q←R
V3      LD4   0,5(LLINK)   n         V3. 转向左. R←LLINK(Q)
        J4NZ  1B           n         如果R≠Λ，则重复
V4      JMP   VISIT        n         V4. 中根次序访问
V5      LD5   1,5(RLINK)   n         V5. 转向右. R←RLINK(Q)
        J5Z   V7           n         如果R = Λ，则转V7
        ENNA  0,6          n - a
        STA   1,5(RLINKT)  n - a     RLINK(Q)←P，RTAG(Q)←"-"
        JMP   2B           n - a     P←Q，Q←R，转到V3
V7      ENT4  T,6          n         V7. 上升
        J4Z   DONE         n         如果P = LOC(T)则终止
        LD4   1,6(RLINKT)  n - 1     R←RLINKT(P)
        J4N   1F           n - 1     如果 RTAG(P) = "-" 则跳
        LD4   0,6(LLINK)   a - 1     R←LLINK(P)
        ST5   0,6(LLINK)   a - 1     LLINK(P)←Q
        ENT5  0,6          a - 1     Q←P
        ENT6  0,4          a - 1     P←R
        JMP   V4           a - 1
1H      ST5   1,6(RLINKT)  n - a     RLINK(P)←Q, RTAG(P)←"+"
        ENT5  0,6          n - a     Q←P
        ENN6  0,4          n - a     P←-R
        JMP   V7           n - a     ∎
```

运行时间是$23n-10$（奇怪啊，竟与a无关）．对于$n\neq 0$。所以执行时间堪与习题20相媲美。

23．插入右边：RLINKT(Q)←RLINKT(P)，RLINKT(P)←+Q，LLINK(Q)←Λ。插入左边，假定LLINK(P)=Λ：置LLINK(P)←Q，LLINK(Q)←Λ，RLINKT(Q)←-P。在P与LLINK(P)≠Λ之间，插入左边：置R←LLINK(P)，LLINK(Q)←R，而且如果RTAG(R)≠"-"则置R←RLINK(R)，重复地进行直到RTAG(R)="-"为止；最后，置RLINK(R)←Q，LLINK(P)←Q，RLINKT(Q)←-P。（如果我们知道一个节点F，它使得P=LLINK(F)或P=RLINK(F)，则对于后一种情况可以使用一更为有效的算法；例如，假定后者，我们可以置INFO(P)↔INFO(Q)，RLINK(F)←Q，LLINK(Q)←P，RLINKT(P)←-Q；这花费固定的时间量，但由于它使节点转遍内存，因此一般不推荐它。）

24．否：





25．我们首先对T中节点数用归纳法证明(b)，并类似地证明(c)。现在(a)分成为一些特殊情况；写$T\preceq_1 T'$如果（1）成立，$T\preceq_2 T'$如果（2）成立，等等。则$T\preceq_1 T'$

和$T'\preceq T''$意味着$T\preceq_1 T''$；$T\preceq_1 T'$和$T'\preceq T''$意味着$T\preceq_2 T''$；而剩下的两种情况通过对于T中的节点数用归纳法证明（a）来进行处理。

26. 如果T的双重次序是(u_1, d_1)，(u_2, d_2)，…，(u_{2n}, d_{2n})，其中之诸u是节点，诸d是1或2，则形成该树的"踪迹"$(v_1, s_1), (v_2, s_2), \cdots, (v_{2n}, s_{2n})$，这里$v_j=\text{info}(u_j)$，且$s_j=l(u_j)$或$r(u_j)$，依据$d_j=1$或2。现在$T\preceq T'$，当且仅当，$T$的（按这里所定义的）踪迹按词典编辑法前于或等于$T'$踪迹。形式上，这意味着或者$n\leqslant n'$且$(v_j, s_j)=(v'_j, s'_j)$，对于$1\leqslant j\leqslant n$；或者否则有一个$k$使得$(v_j, s_j)=(v'_j, s'_j)$，对于$1\leqslant j<k$；而且或者$v_k\preceq v'_k$或者$v_k=v'_k$以及$s_k<s'_k$。

27. **R1**.〔初始化〕 置P←HEAD，P′←HEAD′，即是，给定的右穿线二叉树的分别的表头。转到R3。

R2.〔校验INFO〕 如果INFO(P)≺INFO(P′)，则终止($T\prec T'$)；如果INFO(P)≻INFO(P′)，则终止（$T\succ T'$）。

R3.〔转向左边〕 如果LLINK(P)=A=LLINK(P′)，则转到R4；如果LLINK(P)=A≠LLINK(P′)，则终止（$T\prec T'$）；如果LLINK(P)≠A=LLINK(P′)，则终止（$T\succ T'$）；否则置P←LLINK(P)，P′←LLINK(P′)并转到R2。

R4.〔树结束？〕 如果P=HEAD（或者，等价地，如果P′=HEAD′），则终止（T等价于T'）。

R5.〔转向右边〕 如果RTAG(P)="−"=RTAG(P′)，则置P←RLINK(P)，P′←RLINK(P′)，并转到R4。如果RTAG(P)="−"≠RTAG(P)，则终止（$T\prec T'$）。如果RTAG(P)≠"−"=RTAG(P′)，则终止（$T\succ T'$）。否则，置P←RLINK(P)，P′←RLINK(P′)，并转到R2。 ∎

为证明这个算法的正确性（且因此以了解它是怎样工作的），人们可以对树T_0的大小用归纳法来证明下列命题成立："从步骤R2，以P和P′指向两个非空右穿线二叉树T_0和T'_0开始，如果T_0与T'_0不等价则算法将终止，指出是$T_0\prec T'_0$还是$T_0\succ T'_0$；如果T_0与T'_0等价，则这个算法将到达步骤R4，同时P和P′于是分别指向T_0和T'_0在对称次序下的后继节点。"

28. 等价而且相似。

29. 通过对T的大小用归纳法，证明下列命题是正确的："在步骤C2以P指向一个非空的二叉树T之根，且以Q指向一个有着非空的左和右子树的节点开始，在置INFO(Q)←INFO(P)和对NODE(Q)附加上NODE(P)的左和右子树的复写之后，这个过程将最终地到达步骤C6，而且P和Q分别指向树T和NODE(Q)的先根次序的后继节点。"

30. 假定（2）中的指针是（9）中的LLINK(HEAD)。

L1.〔初始化〕 置Q←HEAD，RLINK(Q)←Q。

L2.〔推进〕置 P←Q $（见以下）。

L3.〔穿线〕 如果RLINK(Q)=A，则置RLINK(Q)←P，RTAG(Q)←"−"，否则置RTAG(Q)←"+"。如果LLINK(P)=A，则置LLINK(P)←Q，LTAG(P)←"−"，否则置LTAG(P)← "+"。

L4.〔完成了？〕如果P≠HEAD，则置Q←P并返回到L2。 ∎

这个算法的步骤L2意味着，一个中根次序的遍历共行程序的活动类似于算法T，附

加额外的条件：算法T在其完全遍历树之后“访问”HEAD。这个记号在描述树的算法中是一个方便的简化，因为我们不需要一次又一次地重复算法T的堆栈机构。当然算法S在步骤L2期间不能使用，因为这个树尚未被穿线。但是，习题21的算法则在步骤L2中能被采用，而且这向我们提供了一种不需要任何辅助的堆栈来穿线一个树的非常巧妙的方法！

31.（a）置P←HEAD；（b）置Q←P$（例如，使用对于一个右穿线的树进行了修改的算法S）；（c）如果P≠HEAD，则AVAIL⇐P；（d）Q≠HEAD，则置P←Q并返回到（b）〔其它的减少内循环长度的解法显然是可能的，尽管基本步骤的次序也有某些重要性。上述的过程行得通，因为我们决不把一个节点复还到可利用的存储中去，直到算法S已经考察了它的LLINK和RLINK两者之后；如同在正文中所看到的那样，在一个完全的树遍历期间，每一个这些链接恰巧使用一次〕。

32. RLINK(Q)←RLINK(P)，SUC(Q)←SUC(P)，SUC(P)←RLINK(P)←Q，PRED(Q)←P，PRED(SUC(Q))←Q。

33. 把NODE(Q)刚好插入到NODE(P)的左边和下边是十分简单的：置LLINKT(Q)←LLINKT(P)，LLINKT(P)←+Q，RLINK(Q)←Λ。对于右边的插入是颇为困难的，因为它实际上要求寻找*Q，这可以与寻找Q#的困难相比拟（见习题19）；也许可以使用习题23中讨论的移动节点的技术。所以对于这种类型的穿线一般的插入更为困难。但是为算法C所要求的插入并不象一般的插入那样困难，而且事实上对于这种类型的穿线，复写的过程倒稍微更快些：

C1. 置P←HEAD，Q←U，转到C4（始终地使用正文中算法C的假定和原理）。

C2. 如果RLINK(P)≠Λ，则置R⇐AVAIL，LLINK(R)←LLINK(Q)，LTAG(R)←“−”，RLINK(R)←Λ，RLINK(Q)←LLINK(Q)←R。

C3. 置INFO(Q)←INFO(P)。

C4. 如果LTAG(P)=“+”则R⇐AVAIL，LLINK(R)←LLINK(Q)，LTAG(R)←“−”，RLINK(R)←Λ，LLINK(Q)←R，LTAG(Q)←“+”。

C5. 置P←LLINK(P)，Q←LLINK(Q)。

C6. 如果P≠HEAD，则转到C2。 ∎

这个算法是正确的，但显得似乎太简单了！

由于步骤C5中为计算P*，Q*的额外时间，对于穿线或右穿线的二叉树，算法C稍微长些。

在步骤C2和C4中适当地置RLINK(R)和RLINKT(Q)，同上述的复写方法相结合，可能以通常的方式对RLINK进行穿线或者置#P于RLINK(P)。

34. **A1**. 置Q←P，然后重复地置Q←RLINK(Q)0次或多次，直到RTAG(Q)=“−”为止。

A2. 置R←RLINK(Q)。如果LLINK(R)=P，则置LLINK(R)←Λ；否则置R←LLINK(R)，然后重复地置R←RLINK(R) 0次或多次直到RLINK(R)=P为止，于是最后置RLINKT(R)←RLINKT(Q)（这个步骤已从原来的树撤消了NODE(P)和它的子树）。

A3. 置RLINK(Q)←HEAD，LLINK (HEAD)←P，[illegible] (为想出和/或了解这个算法的关键，是构造好的“之前和之后”图式)。

36. 否；参看习题 1.2.1-15(c) 的答案。

37. 如果在表示 (2) 中

$$\mathrm{LLINK(P)=RLINK(P)=A}$$

则命

$$\mathrm{LINK(P)=A}$$

否则命 LINK(P)=Q，其中 NODE(Q) 对应于 NODE (LLINK(P)) 而且 NODE (Q+1) 对应于 NODE (RLINK(P))。条件 LLINK(P) 或 RLINK(P)=A 分别通过 NODE(Q) 或 NODE(Q+1) 中的一个哨兵来表示。这个表示使用 n 到 $2n-1$ 之间的内存位置；在所述假定下，(2) 将需要 18 个内存字，同现在的方案的 11 个相对照。在每种表示下，插入和删去操作有近乎相等的效率。但这个表示在与其它结构相结合时，不那么灵便。

2.3.2 小节

1. 如果B为空，则F(B)是一个空的森林。否则，F(B) 由一个树 T 加上森林F (右子树(B))组成，其中 root(T)=root(B) 而且子树 (T)=F (左子树(B))。

2. 在二进记号下的 0 的个数，是在十进记号下小数点的个数，而且进行对应的精确公式是

$$a_1.a_2.\cdots.a_k \leftrightarrow 1^{a_1}01^{a_2-1}0\cdots01^{a_k-1}$$

其中 1^a 表示在一行中有 a 个 1。

3. 把节点的杜威记数法按词典编辑法（如同在字典中那样，从左到右地)进行分类。对于先根次序，把较短的序列 $a_1.\cdots.a_k$ 放在它的扩展 $a_1.\cdots.a_k.\cdots.a_r$ 的前边，而对于后根次序，则放在它的扩展的后边（于是，如果我们正在对词，而不是对数的序列进行分类，则我们对于先根次序，将把字 cat，cataract 以通常的字典次序放置，但对于后根次序，则以相反于初始部分字的次序来放置“cataract，cat”)。对树的大小用归纳法，容易证明这些规则。

4. 通过对节点个数用归纳法，为真。

5. (a) 中根次序。(b) 后根次序。阐述这些算法的等价性之严格归纳证明，是颇有兴趣的。

6. 我们有先根次序 (T)=先根次序 (T')，后根次序 (T)=中根次序 (T')，即使 T 有着仅有一个儿子的节点；剩下的两个次序不处于任何的简单关系之中（例如，在一种情况下 T 的根出现于末尾，而在另一种情况下则差不多出现于中间)。

7. (a) 是；(b) 否；(c) 否；(d) 是。注意一个森林的颠倒的先根次序，等于左右颠倒的森林的后根次序 (在镜面反射的意义下)。

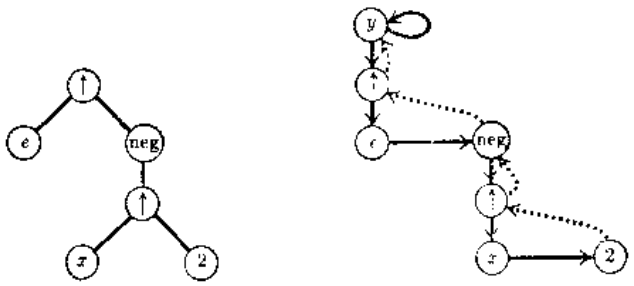
8. $T\preceq T'$ 意味着 info (root(T)) $\prec$info (root(T'))，或者这些 info 相等且或者是 (a)：root(T) 的子树为 T_1，…，T_m，而 root(T') 的子树为 T'_1，…，T'_n，其中有一个 k，使得对于 $1\leqslant j<k$，T_j 等价于 T'_j 但 $T_k\preceq$而且不等价于 T'_k；或者是 (b)：对于

(a) 的记号，T_j 等价于 T'_j，对于 $1 \leqslant j \leqslant n$ 以及 $n \leqslant n'$。

9. 在一个非空的森林中，非终端节点的个数，比其为Λ的右链接的个数小1。因为空的右链接对应于每个非终端节点的最右儿子，而且也对应于森林中最右树的根（这个事实给出了习题2.3.1-14的另一个证明，因为显然空的左链接的个数等于终端节点的个数）。

10. 这些森林是相似的，当且仅当，$n = n'$，而且 $s(u_j) = s(u'_j)$，$1 \leqslant j \leqslant n$；它们是等价的，当且仅当，此外 $\text{info}(u_j) = \text{info}(u'_j)$，$1 \leqslant j \leqslant n$。通过推广引理2.3.1P（取 $f(u) = s(u) - 1$），其证明类似于以前的证明。

11.



12. 如果 INFO(Q1)≠0，则进行下列操作：置R←COPY(P1)；然后如果 TYPE(P2)=0且INFO(P2)≠2，则置P←TREE（"↑"，R，TREE(INFO(P2)−1))；如果 TYPE(P2)≠0，则置R←TREE（"∧"，R，TREE（"−",COPY(P2),TREE(1)))；然后置Q1←MULT（Q1，MULT（COPY(P2)，R))。

如果 INFO(Q)≠0，则置Q←TREE（"X"，MULT（TREE（"ln"，COPY(P1)),Q)，TREE（"↑"，COPY(P1)，COPY(P2)))。最后转到 DIFF(4)。

13. 下列程序实现算法2.3.1C，以 rI1≡P，rI2≡Q，rI3≡R，并且在初始条件和终止条件中作适当的改变：

```
64                                          保留 rI3，rI2的内容
65                                          C1. 初始化
66                                          通过 RLINK(U) = A 建立
67                                              NODE(U)开始
68      8H  CON     0                       为进行初始化的常数 0
69      4H  LD1     0,1 (LLINK)             置 P←LLINK(P) = P *
70      1H  LD3     AVAIL                   R ⇐AVAIL
71          J3Z     OVERFLOW
72          LDA     0,3(LLINK)
73          STA     AVAIL
74          ST3     0,2(LLINK)              LLINK(Q)←R
75          ENNA    0,2
76          STA     0,3(RLINKT)             RLINKT(R)←−Q
77          INCA    8B                      rA←LDC（原始节点）−Q
78          ENT2    0,3                     置Q←R = Q *
79          JAZ     C3                      转到C3，头一次
80      C2  LDA     0,1                     C2. 右边有什么?
81          JAN     C3                      如果 RTAG(P) = "−" 则跳转
```

```
82          LD3   AVAIL          R ← AVAIL
83          J3Z   OVERFLOW
84          LDA   0,3(LLINK)
85          STA   AVAIL
86          LDA   0,2
87          STA   0,3(RLINKT)    置 RLINKT(R) ← RLINKT(Q)
88          ST3   0,2(RLINKT)    RLINKT(Q) ← +R
89    C3    LDA   1,1            C3. 复写 INFO.
90          STA   1,2            复写的 INFO 场
91          LDA   0,1 (TYPE)
92          STA   0,2 (TYPE)     复写的 TYPE 场
93    C4    STZ   0,2(LLINK)     C4. 左边有什么?
94          LDA   0,1(LLINK)
95          JANZ  4B             如果 LLINK(P) ≠ A，则跳转
96    C5    LD2   0,2(RLINKT)    C5. 推进. Q ← RLINKT(Q)
97          LD1   0,1(RLINK)     P ← RLINK(P)
98          J2P   C2             如果 RTAG(Q) = "+" 则跳转
99          ENN2  0,2            Q ← - Q
100   C6    J2NZ  C5             C6. 测试完成否?
101         LD1   8B(LLINK)      r11 ← 建立的头一个节点的地址
102   6H    ENT3  *              恢复变址寄存器
103   7H    ENT2  *
```

14. 命 a 是复写的非终端（运算符）节点的个数。在上边的程序中，各行的执行次数如下：64–67，1；69，a；70–79，$a+1$；80–81，$n-1$；82–88，$n-1-a$；89–95，n；96–98，$n+1$；99–100，$a+2$；101–103，1。总的时间为$(34n+5a+18)u$；其中大约有 1/5 是用来获得可利用的节点，2/5 用来遍历，以及 2/5 用来复写 INFO 和 LINK 信息。

15. 注释留给读者自行作出。

```
218         DIV   LDA   1,6
219               JAZ   1F
220               JMP   CONV2
221               ENTA  SLASH
222               ENTX  0,6
223               JMP   TREE2
224               ENT6  0,1
225         1H    LDA   1,5
226               JAZ   SUB
227               JMP   COPYP2
228               ST1   1F(0:2)
229               ENTA  CON2
230               JMP   TREE0
231               ENTA  UPARROW
```

```
232         1H      ENTX    *
233                 JMP     TREE
234                 ST1     1F(0:2)
235                 JMP     COPYP1
236                 ENTA    0,1
237                 ENT1    0,5
238                 JMP     MULT
239                 ENTX    0,1
240         1H      ENT1    *
241                 ENTA    SLASH
242                 JMP     TREE2
243                 ENT5    0,1
244                 JMP     SUB
```

16. 注释留给读者自行作出。

```
245         PWR     LDA     1,6
246                 JAZ     4F
247                 JMP     COPYP1
248                 ST1     R(0:2)
249                 LDA     0,3 (TYPE)
250                 JANZ    2F
251                 LDA     1,3
252                 DECA    2
253                 JAZ     3F
254                 INCA    1
255                 STA     CON0+1
256                 ENTA    CON0
257                 JMP     TREE0
258                 STZ     CON0+1
259                 JMP     5F
260         2H      JMP     COPYP2
261                 ST1     1F(0:2)
262                 ENTA    CON1
263                 JMP     TREE0
264         1H      ENTX    *
265                 ENTA    MINUS
266                 JMP     TREE2
267         5H      LDX     R(0:2)
268                 ENTA    UPARROW
269                 JMP     TREE2
270                 ST1     R(0:2)
271         3H      JMP     COPYP2
272                 ENTA    0,1
273         R       ENT1    *
```

```
274             JMP     MULT
275             ENTA    0,6
276             JMP     MULT
277             ENT6    0,1
278     4H      LDA     1,5
279             JAZ     ADD
280             JMP     COPYP1
281             ENTA    LOG
282             JMP     TREE1
283             ENTA    0,1
284             ENT1    0,5
285             JMP     MULT
286             ST1     1F(0:2)
287             JMP     COPYP1
288             ST1     2F(0:2)
289             JMP     COPYP2
290     2H      ENTX    *
291             ENTA    UPARROW
292             JMP     TREE2
293     1H      ENTX    *
294             ENTA    TIMES
295             JMP     TREE2
296             ENT5    0,1
297             JMP     ADD
```

20. 更一般地，设 u 和 v 是一个森林的任何节点。如果 u 是 v 的一个祖宗，则由归纳法立即有，u 在先根次序下前于 v，而在后根次序下后于 v。反之，假设在先根次序下，u 在 v 之前，而且后根次序下，u 在 v 之后；我们必须证明，u 是 v 的一个祖宗。如果 u 是头一个树的根，则这是显然的。如果 u 是头一个树的另一个节点，则 v 也必然是，因为在后根次序下 u 在 v 之后；所以可应用归纳法。类似地，如果 u 不在头一个树中，则 v 必不是两者之一，因为在先根次序下 u 在 v 之前。

21. 如果 NODE(P)是一个二元运算符，则指向它的两个操作数的指针为 P1=LLINK(P) 和 P2=RLINK(P1)=$P。算法D利用了事实 P2$=P，所以 RLINK(P1) 可以改变成Q1，即指向 NODE(P1) 之导数的指针，而后来在步骤D3中 RLINK(P1)被复位。对于三元运算，是难于推广这一技巧的；我们将有，比方说，P1=LLINK(P)，P2=RLINK(P1)，P3=RLINK(P2)=$P。现在在计算了导数之后，我们将暂时地置 RLINK(P1)←Q1，而且然后在计算了下一个导数Q2之后，我们可以置 RLINK(Q2)←Q1 和 RLINK(P2)←Q2 以及恢复 RLINK(P1)←P2。但这肯定是不漂亮的，而且它逐渐地增长以至运算符的次数变成更高。因此，在算法D中暂时地改变 RLINK(P1) 的设计，肯定地是一个技巧，而不是一项技术。为控制一个求微商之过程的更优越的方法，由于它可推广到更高次数的运算符，而且不依赖于孤立的技巧，因而可以以算法2.3.3F为基础，这在习题2.3.3-3中进行了详细的讨论。

22. 由定义立即推出，这个关系是传递的，即是，如果$T \subseteq T'$和$T' \subseteq T''$，则$T \subseteq T''$（事实上，容易看出这个关系是一个部分编序）。显然，如果我们设f是把诸节点变成它们自身的函数，则$l(T) \subseteq T$和$r(T) \subseteq T$。因此，如果$T \subseteq l(T')$或$T \subseteq r(T')$，则我们必然有$T \subseteq T'$。

假设f_l和f_r是分别地示出$l(T) \subseteq l(T')$和$r(T) \subseteq r(T')$的函数。设$f(u) = f_l(u)$如果u在$l(T)$中，$f(u) = \mathrm{root}(T')$如果u是root(T)，否则$f(u) = f_r(u)$。现在容易推知，f示出$T \subseteq T'$；例如，如果我们设$r'(T)$表示$r(T) - \mathrm{root}(T)$，则我们有先根次序$(T) = \mathrm{root}(T)$先根次序$(l(T))$先根次序$(r'(T))$；先根次序$(T') = f(\mathrm{root}(T))$先根次序$(l(T'))$先根次序$(r'(T'))$。

反之则不成立，例如，考虑图25中以b和b'为根的子树。

2.3.3 小节

1. 是，我们可以重新构造它们，就如同由（4）推导出（3）一样。但是互换LTAG和RTAG，LLINK和RLINK，而且使用排队以代替堆栈。

2. 在算法F中作下列的变动：步骤F1，变成“先根次序下森林的最后的节点”。步骤F2，在两处把“$f(x_d), \cdots, f(x_1)$”改变成为“$f(x_1), \cdots, f(x_d)$”。步骤F4，“如果P是先根次序下的头一个节点，则终止本算法（则从顶上到底下，堆栈含有$f(\mathrm{root}(T_1)), \cdots, f(\mathrm{root}(T_m))$，其中$T_1, \cdots, T_m$是从左到右，给定的森林的树）。否则置P←*P（在给定的表示下P←P−1），并返回到F2”。

3. 在步骤D1中，也置S←A（S是链接到堆栈顶上的一个链接变量）。步骤D2，比如说变成“如果NODE(P)一个一元运算符，则置Q←S，S←RLINK(Q)，P1←LLINK(P)；如果它表示一个二元运算符，则置Q←S，Q1←RLINK(Q)，S←RLINK(Q1)，P1←LLINK(P)，P2←RLINK(P1)。然后实施DIFF(TYPE(P))”。步骤D3变成“置RLINK(Q)←S，S←Q”。步骤D4变成“置P←P$”。如果我们假定S≡LLINK(DY)，则操作LLINK(DY)←Q可予以避免。这个技术显然可推广到三元的或更高阶的运算符。

4. 类似于（10）的表示用$n-m$个LLINK和$n+(n-m)$个RLINK。在两种形式的表示之间，总的链接个数之差为$n-2m$。当在一个节点中，LLINK和INFO场要求大约相同的空间数量以及当m稍大些时，亦即当非终端节点有稍大的次数时，配置（10）就更优越的。

5. 肯定地，包括穿线的RLINK是愚蠢的，因为RLINK穿线无论如何仅仅是指向FATHER。象在2.3.2-(4)中那样穿线的LLINK将是有用的，如果在树中有必要向左移动时，例如，如果我们要以颠倒的后根次序或以家族次序来遍历一个树时；但是这些运算如果没有穿线的LLINK，难度也并不甚大，除非这些节点有着非常高的次数。

6. **L1**. 置P←FIRST，FIRST←A。

L2. 如果P = A，则终止。否则置Q←RLINK(P)。

L3. 如果FATHER(P) = A，则置RLINK(P)←FIRST，FIRST←P；否则置RLINK(P)←LSON(FATHER(P))，LSON(FATHER(P))←P。

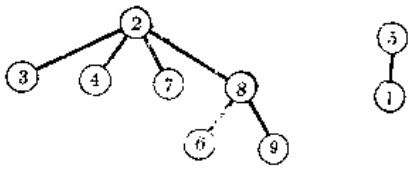
L4. 置P←Q并返回到L2。

7. {1, 5}{2, 3, 4, 7}{6, 8, 9}。

8. 实施算法E的步骤E3，然后测试是否 $j = k$。

9. FATHER[k]: 5 0 2 2 0 8 2 2 8

k: 1 2 3 4 5 6 7 8 9



10. 一个思想是把每个根节点的FATHER置为它的树的节点个数之负(这些值容易保持为最新的)；如果在步骤E4中|FATHER[j]|>|FATHER[k]|，则交换 j 和 k 的作用。这个技术（由马·道·麦基尔罗伊（M. D. McIlroy)给出）确保每个操作采取阶数 $\log n$ 个步骤。

为了获得更快的速度，我们可以使用下列的艾伦·特里特（Alan Tritter）提出的建议：在步骤E4中，对于在步骤E3中遇到的所有值 $x \neq k$，置 FATHER[x]←k。这意味着对树进行额外的通过，但它瓦解这些树，使将来的检索更快些（关于等价性问题的进一步的讨论，见第7章)。

11. 只须来定义变换，它是对于每个输入（P, j, Q, k)：

T1. 如果 FATHER(P)≠Λ，则置 $j \leftarrow j$ +DELTA(P)，P←FATHER(P)，并重复这一步骤。

T2. 如果 FATHER(Q)≠Λ，则置 $k \leftarrow k$ +DELTA(Q)，Q←FATHER(Q)，并重复这一步骤。

T3. 如果P=Q，则校验 $j = k$（否则在这个输入中已经造成一个错误，等价性是矛盾的)。如果P≠Q，则置 DELTA(Q)←$j - k$，FATHER(Q)←P，LBD(P)←min(LBD(P), LBD(Q)+DELTA(Q))，UBD(P)←max(UBD(P), UBD(Q)+DELTA(Q))。

注意：有可能允许"ARRAY X[l : u]"说明与等价性相混合而出现，或者，在其它的等价于它们之前，允许指定某些变量的地址，等等，在适当的条件下，这些是不难理解的。关于这个算法的进一步的发展，见CACM 7 (1964), 301-303, 506。

12. (a) 是（如果不需要这一条件，则将有可能避免出现于步骤A2和A9中对S的循环)。(b)是。

13. 决定性的事实是从P往上引出的UP链，总是提出与从Q往上引出的UP链同样的变量和同样的这些变量的指数，但后一个链接可以包括具有指数0的变量的额外步骤(这个条件贯穿大多数算法是成立的，除了在步骤A9和A10的执行期间外)。现在我们或从A3或从A10到达步骤A8，而且在每种情况下已经验证了 $E \setminus P(Q) \neq 0$。因此 $E \setminus P(P) \neq$

0，尤其由此得出P≠Λ，Q≠Λ，UP(P)≠Λ，UP(Q)≠Λ，而且得到习题中指出的结果。所以这个证明依赖于说明上述的UP链条件为算法的动作所保持。

16. INF01，RLINK表格连同在正文中为计算LTAG的提示，向我们提出了以通常方式表示的一个二叉树形的等价者。这一思想是在后根次序下遍历这个树，计算次数如下：

P1. 设R，D，和I是堆栈，它们开始时为空；然后置$R \Leftarrow n+1$，$D \Leftarrow 0$，$j \leftarrow 0$，$k \leftarrow 0$。

P2. 如果top(R)$>j+1$(即，如果LTAG="+"，如果该场出现的话)，转到P5。

P3. 如果I是空的，则终止本算法；否则置$i \Leftarrow I$，$k \leftarrow k+1$，INF02[k]←INF01[i]，DEGREE[k]⇐D。

P4. 如果RLINK[i]$=0$，则转到P3；否则删去R的顶上(它将等于RLINK[i])。

P5. 置top(D)←top(D)$+1$，$j \leftarrow j+1$，$I \Leftarrow j$，$D \Leftarrow 0$，而且如果RLINK[j]$\neq 0$，则置R⇐RLINK[j]。转到P2。

17. 我们(通过对在单个树T中的节点数用归纳法)证明，如果P是一个指向T的指针，而且如果堆栈开始时为空，步骤F2到F4将以堆栈上的单个的值f(root(T))结束。这对于$n=1$为真。如果$n>1$，则有$0<d=$DEGREE(root(T))个子树T_1，…，T_d；通过归纳法和一个堆栈的本性，而且由跟着root(T)的T_1，…，T_d组成后根次序，这个算法计算$f(T_1)$，…，$f(T_d)$，而后如所希望的，计算f(root(T))。这就得出算法F对于森林的正确性。

18. **G1**. 置堆栈为空，并设P指向树的根(在后根次序下的最后的节点)。求值f(NODE(P))。

G2. 把f(NODE(P))的DEGREE(P)个复写压入堆栈。

G3. 如果P是后根次序下的头一个节点，则终止本算法。否则置P←SP(这在(9)中将只不过是P←P−1)。

G4. 利用在堆栈顶上的值(它是f(NODE(FATHER(P))))求值f(NODE(P))。把这个值弹出堆栈，并返回G2。

注意：类似于这个的一个算法可以以先根次序为基础，而不是如同在习题2中那样以后根次序。事实上，家族次序或层次次序可予使用；在后种情况下，我们将使用排队来代替堆栈。

2.3.4.1 小节

1. (B, A, C, D, B)，(B, A, C, D, E, B)，(B, D, C, A, B)，(B, D, E, B)，(B, E, D, B)，(B, E, D, C, A, B)。

2. 设$(V_0, V_1, \cdots, V_n)$是从V到V'的最小可能长度的通路。如果现在$V_j=V_k$对于某$j<k$，则$(V_0, \cdots, V_j, V_{k+1}, \cdots, V_n)$是一条更短的通路。

3. (基本通路遍历e_3和e_4一次，但循环C_2遍历它们"−1"次，给出一个0的净总次数)。遍历下列诸边：e_1，e_2，e_6，e_7，e_9，e_{10}，e_{11}，e_{12}，e_{14}。

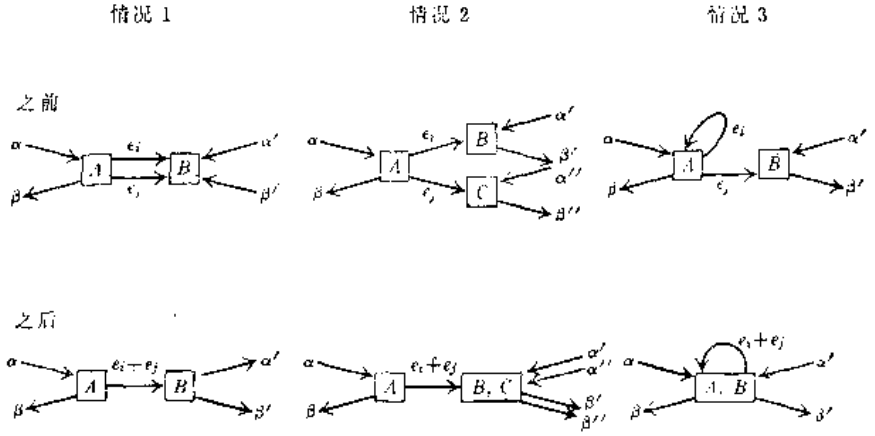
4. 如果不是，则命G''是通过删去使$E_j=0$的每条边e_j所得到的G'的子图形。于是G''是一个没有回路的有限图形，而且至少有一条边，所以由定理A的证明，至少有一个顶点V，它恰巧与一个其它的顶点V'相邻。设e_j是连接V到V'的边，则在顶点V处的

克希霍夫方程（1）为 $E_j=0$，与 G'' 的定义相矛盾。

5. $A=1+E_8$，$B=1+E_8-E_2$，$C=1+E_8$，$D=1+E_8-E_5$，$E=1+E_{17}-E_{21}$，$F=1+E''_{13}+E_{17}-E_{21}$，$G=1+E''_{13}$，$H=E_{17}-E_{21}$，$J=E_{17}$，$K=E''_{19}+E_{20}$，$L=E_{17}+E''_{19}+E_{20}-E_{21}$，$P=E_{17}+E_{20}-E_{21}$，$Q=E_{20}$，$R=E_{17}-E_{21}$，$S=E_{25}$。注意：在这种情况下，也有可能借助于 A，B，…，S 来解 E_2，E_5，…，E_{25}；因此有几个独立的解，说明为什么我们消去等式 1.3.3-(8) 中的六个变址。

6. 基本回路：$C_0=e_0-e_1+e_4+e_9$（基本通路是 $e_1+e_4+e_9$）；$C_5=e_5+e_3+e_2$；$C_6=e_6-e_2+e_4$；$C_7=e_7-e_4-e_8$；$C_8=e_8-e_9-e_4-e_3$。因此我们求得 $E_1=1$，$E_2=E_5-E_6$，$E_3=E_5-E_7-E_8$，$E_4=1+E_6-E_7-E_8$，$E_9=1-E_6$。

7. 在约化过程的每一步骤中组合两个在同一个方框开始的箭头 e_i 和 e_j，而且这足以证明这样的步骤可被逆转。于是经组合之后，我们已给出 e_i+e_j 的值，而且在组合之前，我们必须对于 e_i 和 e_j 赋以一致的值。实际上有三种不同的情况：



这里 A，B，C 代表顶点或超顶点，诸 α 和诸 β 代表除了 e_i+e_j 之外其它给定的流量；这些流量每一个可分布在若干条边当中，虽然示出的仅一条边。在情况 1 中（e_i 和 e_j 引向同一方框），我们可以任意地选择 e_i，然后 $e_j\leftarrow(e_i+e_j)-e_i$，在情况 2 中（$e_i$ 和 e_j 引向不同的方框），我们必须置 $e_i\leftarrow\beta'-\alpha'$，$e_j\leftarrow\beta''-\alpha''$。在情况 3 中（$e_i$ 是一个循环而 e_j 则不是），我们必须置 $e_i\leftarrow\beta'-\alpha'$，$e_j\leftarrow(e_i+e_j)-e_i$。在每种情况下，我们都已象所希望的那样，逆转了组合的步骤。

这一习题的结果实际上证明了，在既约的流程图中基本回路的个数，是必须加以测定，以确定所有其它的流量的顶点流量之极小个数。在给定的例子中，既约的流程图指出，仅有三个顶点流量（例如，a，c，d）需要加以测定，而习题 6 原来的流程图则有四个独立的边流量。在约化期间，每次情况 1 出现，我们都节省一个测定。

类似的约化过程可以以组合流入一个给定的方框的箭头为基础，而不是以那些流出的。可以证明，这将得出相同的既约流程图，除了超顶点将含有不同的名称之外。

这个习题的构造是以阿尔门·内哈彼蒂安（Armen Nahapetian）和弗·史蒂文森（F. Stevenson）所给出的思想为基础的。关于进一步的注释，见唐·欧·克努特和弗·史蒂文森，*BIT* 11(1973)。

8. 从一个顶点到它本身的每一条边，变成为全都通过它自身的一条“基本回路”。如

果有 $k+1$ 条边 e，e'，…，$e^{(k)}$ 在顶点 V 与 V' 之间，则做 k 条基本回路 $e'+e$，…，$e^{(k)}\pm e$（按照诸边于相反或相同方向进行来选定＋或－），而后就象仅仅有边 e 存在那样来进行。

实际上这种情况在概念上将更为简单，如果我们以这样一种方式来定义一个图形的话，即是，在顶点之间允许有多条边，而且允许边从一个顶点到它自身；通路和回路将借助于边而不是顶点来进行定义。事实上，这种类型的定义，已在关于有向图形的下一节中做出。

9．（下列的解是以我们可能打印出每条不与前面的诸边组成一条回路的边这样的思想为基础的）。利用算法 2.3.3E，而且在该算法的记号下，每一对偶 (a_i, b_i) 表示 a_i—b_i。唯一的改变是，如果在步骤E4中 $j\neq k$，则打印 (a_i, b_i)。

为证明这一算法是正确的，我们必须证明（a）这个算法不打印出那些形成一条回路的边，（b）如果 G 至少含有一个自由子树，则这个算法打印出 $n-1$ 条边。定义 $j\sim k$，如果存在一条从 V_j 到 V_k 的通路，或者如果 $j=k$。这显然是一个等价关系，而且 $j\sim k$，当且仅当，这个关系可从等价式 $a_1=b_1$，…，$a_m=b_m$ 导出。现在（a）是正确的，因为本算法不打印出与以前已打印的边形成一条回路的那些边；（b）是正确的，因为如果所有顶点都等价，则对于恰好一个 k，FATHER$[k]=0$。

10．如果端点已经连接在一起，则对应的图形必须在技术的意义下加以连接。导线的极小数显然将不含回路。所以我们必须有一个自由树。按定理A，一个自由树包含 $n-1$ 条导线，而一个具有 n 个顶点和 $n-1$ 条边的图形是一个自由树，当且仅当，它是连通的。

11．只须证明，当 $n>1$ 且 $c(n-1,\ n)$ 是 $c(i,\ n)$ 之极小者时，至少存在一个极小代价的树，其中 T_{n-1} 是连线到 T_n 的（因为，任何具有 $n>1$ 个端点和 T_{n-1} 连线到 T_n 的极小代价树，必须也是一个有 $n-1$ 个终端的极小代价树，如果我们认为 T_{n-1} 和 T_n 是“公共的”，利用这一算法中所指出的约定）。

为证明上述命题，假设我们有一个极小代价树，其中 T_{n-1} 不连线到 T_n。如果我们加上导线 $T_{n-1}T_n$，则我们就得到一条回路，而且在该回路中的任何其它的导线均可予以撤消。把接触 T_n 的另一条导线撤消，给了我们另一个树，它的总的代价不大于原来的，而且出现 $T_{n-1}T_n$。

12．保持两个辅助的表格 $a(i)$ 和 $b(i)$，对于 $1\leqslant i<n$，表示这样的事实，即是，从 T_i 到一个选定的端点的最便宜的连接是到 $T_{b(i)}$，而且它的代价是 $a(i)$；开始时 $a(i)=c(i,\ n)$ 和 $b(i)=n$。然后进行下列的操作 $n-1$ 次：求 i，使得 $a(i)=\min_{1\leqslant j<n}a(j)$；连接 T_i 到 $T_{b(i)}$；对于 $1\leqslant j<n$，如果 $c(i,\ j)<a(j)$，则置 $a(j)\leftarrow c(i,\ j)$ 和 $b(j)\leftarrow i$；并且置 $a(i)\leftarrow\infty$。

（避免使用∞是稍微有效的，代之以保持一份尚未选定的那些 j 的单路链接表。采用或不采用这个直接了当的改进，这一算法均花去 $O(n^2)$ 个操作）。见埃·怀·迪伊克斯特拉，Proc. Nederl. Akad. Wetensch. A-63(1960)，196-199。

13．我们必须证明 G 是连通的。如果 $V\neq V'$ 且 VV' 不是 G 的一条边，则把边 VV' 加到 G 上；这就引进了一条必然包含这新边的回路，所以它可以写成为 $(V, V', V_2, \cdots, V)$；因此在 G 中有一条从 V' 到 V 的通路。

14．如果对于某 $i\neq j$，没有回路从 V_i 到 V_j，则没有对换之乘积将把 i 移到 j。所

以如果生成了所有的排列，则这图形必连通。反之，如果它连通，则必要时撤去些边直到我们得到一个树为止。然后重新将顶点编号使得V_n仅与另一个顶点相邻，称之为V_{n-1}(见定理A的证明)。现在不同于$(n-1, n)$的对换，形成一个具有$n-1$个顶点的树；所以由归纳法，如果π是$\{1, 2, \cdots, n\}$上的任何排列，它使n保持固定，则π可以写成那些对换的乘积。如果π把n移到j，则$\pi(j, n-1)(n-1, n)=\rho$固定n；因此

$$\pi=\rho(n-1, n)(j, n-1)$$

可以写成为给定对换之一乘积。

2.3.4.2 小节

1. 设$(e_1, \cdots, e_n)$是一条从V到V'的最小可能长度的有向通路。现在如果$\mathrm{init}(e_j)=\mathrm{init}(e_k)$对于$j<k$，则$(e_1, \cdots, e_{j-1}, e_k, \cdots, e_n)$将是一条更短的通路；如果$\mathrm{fin}(e_j)=\mathrm{fin}(e_k)$对于$j<k$，则可应用同样的论证。因此$(e_1, \cdots, e_n)$是简单的。

2. 其中所有的符号皆相同的那些回路：C_0，C_8，C'_{13}，C_{17}，C'_{19}，C_{21}。

3. 例如，使用三个顶点A，B，C，连同从B到A和到C的弧。

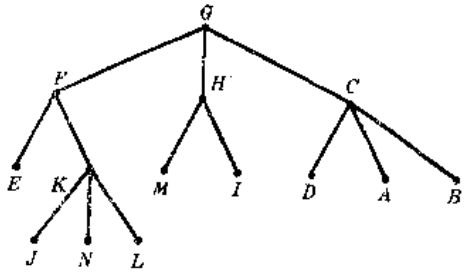
4. 如果没有有向回路，则算法2.2.3T把G拓扑分类。如果有一个有向回路，则拓扑分类显然是不可能的(取决于对这道习题如何解释，长度为1的有向回路可从上述讨论中排斥出去)。

5. 设k是最小整数，使得$\mathrm{fin}(e_k)=\mathrm{init}(e_j)$对于某$j\leqslant k$。则$(e_j, \cdots, e_k)$是一条有向回路。

6. 假的(在一技术细节方面)，仅仅因为从一个顶点到另一个顶点可能有若干条不同的弧。

7. 对于有限的有向图形为真：因为如果我们在任何顶点V开始并且沿着唯一可能的有向通路，则我们决不会两次遇到任何顶点，所以我们必然终于达到顶点R(唯一无后继的顶点)。对于无限的有向图形，这一结果显然是假的，因为我们可以有顶点R，V_1，V_2，V_3，…以及从V_j到V_{j+1}的弧，对于$j\geqslant 1$。

9. 所有的弧都向上指。



10. **G1**. 置$k\leftarrow F[j]$，$F[j]\leftarrow 0$。

G2. 如果$k=0$，则停止；否则置$m\leftarrow F[k]$，$F[k]\leftarrow j$，$j\leftarrow k$，$k\leftarrow m$，而且重复步骤G2。 ∎

11. 这个算法把算法2.3.3E与上题的方法结合起来，使得所有有向树的弧对应于有向图形中实际的弧；$S[j]$是一份辅助的表格，它告知一条弧是从j到$F[j]$($S[j]=+1$)还是从$F[j]$到j($S[j]=-1$)。开始时，$F[1]=\cdots=F[n]=0$。下列诸步

骤可用来处理每条弧(a, b)：

C1. 置$j \leftarrow a$，$k \leftarrow F[j]$，$F[j] \leftarrow 0$，$s \leftarrow S[j]$。

C2. 如果$k = 0$，则转到C3；否则置$m \leftarrow F[k]$，$t \leftarrow S[k]$，$F[k] \leftarrow j$，$S[k] \leftarrow -s$，$s \leftarrow t$，$j \leftarrow k$，$k \leftarrow m$，而且重复步骤C2。

C3.（现在a表现为它的树形的根）。置$j \leftarrow b$，而且然后，如果$F[j] \neq 0$，则重复地置$j \leftarrow F[j]$直到$F[j] = 0$为止。

C4. 如果$j = a$，则转到C5；否则置$F[a] \leftarrow b$，$S[a] \leftarrow +1$，打印(a, b)以作为一条属于自由子树的弧，而且终止。

C5. 打印"CYCLE"，后面紧跟"(a, b)"。

C6. 如果$F[b] = 0$，则终止。否则如果$S[b] = +1$，则打印"$+(b, F[b])$"，否则打印"$-(F[b], b)$"；置$b \leftarrow F[b]$并重复步骤C6。 ∎

12. 它等于输入次数，每个顶点的输出次数仅可为0或1。

13. 定义G之有向子树的序列如下：G_0仅是顶点R。G_{k+1}是G_k加上G的任何顶点V，不在G_k中但对于它有一条从V到V'的弧，其中V'在G_k中，加上对于每个这样的顶点的一条这样的弧$e[V]$。由归纳法立即得知，对所有$k \geqslant 0$，G_k是一个有向树，而且如果在G中从V到R有一长度为k的有向通路，则V在G_k中。因此G_∞，即在任何G_k中所有的V和$e[V]$之集合，就是所求的G的有向子树。

14. 在"词典编辑顺序"下，

$$(e_{12}, e_{10}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{22}, e_{21}), \quad (e_{12}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{01}),$$
$$(e_{12}, e_{20}, e'_{01}, e_{10}, e_{00}, e_{01}, e'_{12}, e_{22}, e_{21}), \quad (e_{12}, e_{20}, e'_{01}, e'_{12}, e_{22}, e_{21}, e_{10}, e_{00}, e_{01}),$$
$$(e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e_{10}, e_{01}, e'_{12}, e_{21}), \quad (e_{12}, e_{22}, e_{20}, e_{00}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{01}),$$
$$(e_{12}, e_{22}, e_{02}, e'_{01}, e_{10}, e_{20}, e_{01}, e_{12}, e_{21}), \quad (e_{12}, e_{22}, e_{02}, e'_{01}, e'_{12}, e_{21}, e_{10}, e_{20}, e_{01});$$

这八种可能性，来自于e_{00}或e'_{01}、e_{10}或e'_{12}、e_{01}或e_{21}中，哪个应当排在其它之前的独立选择。

15. 如果它是连通和平衡的，则它或者仅有一个顶点，或者有一条与所有顶点都接触的欧拉线路；两次围绕该线路，将第一次接触任何给定的顶点V，并且第二次接触任何其它给定的顶点V'。

16. 考虑具有顶点V_1，…，V_{13}，和对于每个在叠j中的k具有一条从V_j到V_k的弧的有向图形G。赢得游戏，就等价于在这个有向图形中寻找出一条欧拉线路（因为如果这个游戏获胜，则所翻转的最后一张牌必然来自于中心；这个图形就是平衡的）。现在如果这局游戏得胜，则由引理E，所述图形是一个有向树。反之，如果所述图形是一个有向树，则由定理D游戏获胜。

17. $\frac{1}{13}$。可以得到这个答案，如同作者头一次得到它那样。办法是以2.3.4.1小节的方法为基础，通过对特殊类型的有向树进行实际的枚举，而且应用生成函数，等等。从以下简单而直接的证明也容易得出，即定义翻转整付所有牌的次序如下：遵照游戏的规则直到停住为止，然后通过翻转头一张可利用的牌来"作弊"（找不空的头一叠，从叠1顺时针方向前进）并与以前一样继续下去，直到终于把所有的牌都翻转了为止。在翻转的次序下，诸牌处于完全随机的次序（因为一张牌的值，直到翻转它之前实未确定）。所以向

题恰恰是计算，在一付随机洗过的牌中的最后一张牌是王（大 k）的概率。更一般地说，当游戏玩完时，k 张牌仍然朝下的概率，是在随机的洗牌下最后的王(大 k)后面跟着 k 张牌的概率，这就是 $4!\binom{51-k}{3}48!/52!$。因此一个玩这种游戏的人，如果不作弊，则在每一局游戏中，平均恰恰翻转 42.4 张牌。注意：类似地可以证明，选手在上边描述的过程中将需要"作弊" k 次的概率，恰恰是由斯特林数 $\left[\begin{matrix}13\\k-1\end{matrix}\right]/13!$ 给出的（见1.2.10小节，等式(9)和习题 7；在习题 18 中考虑了更一般的纸牌的情况）。

18.（a）如果有一条回路 $(V_0, V_1, \cdots, V_k)$，其中必须有 $3\leqslant k\leqslant n$，则对应于这条回路的 k 条边的 A 的 k 行之和，带上相应的符号，是一全 0 的行；所以如果 G 不是一个自由树，则 A_0 的行列式为 0。

现在如果 G 是一个自由树，则我们可以认为它是一个具有根 V_0 的有序树，而且我们可以重新排列 A_0 的行与列，使得这些列处于先根次序之下，并且使得第 k 行对应于从第 k 个顶点(列)到它的父亲的边；于是这个矩阵是在对角线上为 ± 1 的三角矩阵，所以行列式为 ± 1。

（b）由比尼特-柯西公式（习题 1.2.3-46），我们有

$$\det A_0^T A_0=\sum_{1\leqslant i_1<\cdots<i_n\leqslant m}(\det A_{i_1\ldots i_n})^2$$

其中 $A_{i_1\ldots i_n}$ 表示一个由 A_0 的行 i_1，…，i_n 组成的矩阵（于是对应于 G 的 n 条边的一种选择）。现在由(a)得出结果。

19.（a）$a_{00}=0$ 和 $a_{jj}=1$ 的条件，恰是有向树之定义的条件(a)，(b)。如果 G 不是一个有向树，则有一个有向回路（由习题 7），而且这意味着，对应于这有向回路中之顶点的 A_0 的诸行，加成为一全 0 的行；因此 $\det A_0=0$。如果 G 是一个有向树，则对于每个家族的诸儿子指定一个任意的次序，并把 G 看成是一个有序树。现在排列 A_0 的行与列，直到它们对应于顶点的先根次序为止。由于象应用于列一样的排列已经应用于行，这行列式不变；而且得到的矩阵是在每一对角线位置上有 $+1$ 的三角矩阵。

（b）我们可以假定 $a_{0j}=0$ 对于所有的 j。因为没有从 V_0 发生的边能加入到一个有向子树中。我们还可以假定 $a_{jj}>0$ 对于所有的 $j\geqslant 1$，因为否则整个的第 j 行为 0，而且显然地没有有向子树。现在对弧数用归纳法：如果 $a_{jj}>1$，则设 e 是从 V_j 引出的某条弧；设 B_0 是一个类似 A_0 的矩阵，但删去了弧 e；且设 C_0 为类似 A_0 的矩阵，但删去了除了从 V_j 引出的 e 之外的所有的弧。

例：$A_0=\begin{pmatrix}3&-2\\-1&2\end{pmatrix}$，$j=1$，$e=$从 V_1 到 V_0 的弧；则 $B_0=\begin{pmatrix}2&-2\\-1&2\end{pmatrix}$，$C_0=\begin{pmatrix}1&0\\-1&2\end{pmatrix}$。于是 $\det A_0=\det B_0+\det C_0$，由于这些矩阵在除了行 j 之外的所有行上都一致，而且在该行上 A_0 为 B_0，C_0 之和。而且，G 之有向子树的个数，是不使用 e 的（即是，$\det B_0$，由归纳法）加上使用 e 的（即是，$\det C_0$）子树的个数。

博在德的这一重要定理，已经由西尔威斯特（Sylvester）不加证明地叙述了两次〔Journal f. d. reine und angewandte Math. 52(1856), 279; Quart. J. Math. 1 (1857), 55-56〕。

20. 利用习题 18，我们求得 $B=A_0^TA_1$。或者，利用习题 19，B 是以两条弧代替 G 的每条边（在每一方向各一条）得到的有向图形 G' 的矩阵 A_0；而且 G 的每个自由子树，唯一地对应于 G' 的一个以 V_0 为根的有向子树（通过根的选择来确定诸弧的方向）。

21. （这一结果，可以从正文中所引的范·阿登尼-埃伦费斯特和德·布鲁因的论文中所用的有趣的然而是颇为复杂的论证导出。下列推导不仅要简单些，而且它还可以推广来确定 G^* 的有向子树的个数，当 G 是任意的有向图形时；见 R·道森（R. Dawson）和欧·约·古德，Ann. Math. Stat. 28 (1957)，946～956；唐·欧·克努特，组合理论杂志 3 (1967)，309-314）。

如同在习题 19 中那样构造矩阵 A 和 A^*。对于在图 36 和 37 中的例子的图形 G，G^*，

$$A=\begin{pmatrix} 2 & -2 & 0 \\ -1 & 3 & -2 \\ -1 & -1 & 2 \end{pmatrix}$$

$$A^*=\begin{array}{c} \\ (00) \\ (10) \\ (10) \\ (01) \\ (01) \\ (21) \\ (12) \\ (12) \\ (22) \end{array}\begin{array}{c} \begin{array}{ccccccccc} (00) & (10) & (10) & (01) & (01) & (21) & (12) & (12) & (22) \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 3 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 3 & 0 & 0 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 3 & 0 & -1 & -1 & 0 \\ 0 & -1 & 0 & 0 & 0 & 3 & -1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & -1 & 3 & 0 & -1 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 3 & -1 \\ 0 & 0 & -1 & 0 & 0 & -1 & 0 & 0 & 2 \end{pmatrix} \end{array}$$

把不确定的 λ 加到 A 和 A^* 的左上角的元素上（在本例中，这给出 $2+\lambda$ 以代替 2）。如果 $t(G)$，$t(G^*)$ 是 G 和 G^* 的有向子树的个数，则我们有 $t(G)=(1/\lambda)(n+1)\det A$，$t(G^*)=(1/\lambda)m(n+1)\det A^*$（一个平衡的图形的有向子树的个数，对于任何给定的根是相同的，例如，通过习题 22 可知）。

如果我们把顶点 V_{jk} 对相等的 k 分组，则矩阵 A^* 可如上所示地划分之。设 $B_{kk'}$ 是由 V_{jk} 的行和 $V_{jk'}$ 的列所组成的 A^* 的子矩阵，对于所有的 j 和 k，使得 V_{jk} 和 $V_{jk'}$ 都在 G^* 中。通过把每个子矩阵的第 2，…，第 m 列加到头一列，并且然后从第 2，…，第 m 行减去每个子矩阵的头一行，矩阵 A^* 被变换成使得

$$B_{kk'}=\begin{pmatrix} a_{kk'} & * & \cdots & * \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \text{ 对于 } k\neq k' \qquad B_{kk}=\begin{pmatrix} a_{kk}+\lambda\delta_{k0} & * & \cdots & * \\ -\lambda\delta_{k0} & m & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ -\lambda\delta_{k0} & 0 & 0 & m \end{pmatrix}$$

这里“*”表示近乎不相干的值。由此推出，$\det A^*$ 是 $m^{m(n-1)}$ 乘

$$\begin{pmatrix} \lambda+a_{00} & * & * & \cdots & * & a_{01}\cdots a_{0m} \\ -\lambda & m & 0 & \cdots & 0 & 0 \\ \vdots & & & & & \\ -\lambda & 0 & 0 & \cdots & m & 0 \\ a_{10} & * & * & \cdots & * & a_{11}\cdots a_{1n} \\ \vdots & & & & & \\ a_{n0} & * & * & \cdots & * & a_{n1}\cdots a_{nn} \end{pmatrix}$$

的行列式。剩下的 * 全为 0，除了在每列中恰有一个 -1 之外。把最后 n 行加到顶上一行，而且以头一行展开这个行列式，以得到 $m^{n(m-1)+m-1}\times\det A-(m-1)m^{n\cdot m^{n-1}-m-2}\det A$。

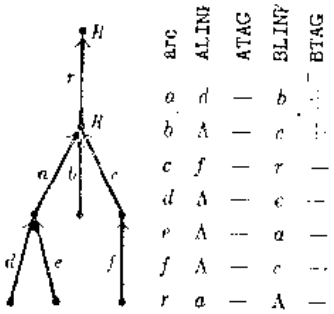
22. 总数是 $(\sigma_1+\cdots+\sigma_n)$ 乘以由一个给定的边 e_1 开始的欧拉线路的条数，其中 $\mathrm{init}(e_j)=V_1$。由引理 E，每条这样的线路，确定一个以 V_1 为根的有向子树，而且对于 T 个有向子树中的每一个，有 $\prod_{1\leqslant j\leqslant n}(\sigma_j-1)!$ 条满足定理 D 之三个条件的通路，对应于不同的次序，在此次序下诸弧 $\{e\mid \mathrm{init}(e)=V_j,\ e\neq e_j\}$ 皆进入 P（参看习题 14）。

23. 象在提示中那样构造具有 m^{k-1} 个顶点的有向图形 G_k，而且设 $(x_1,\cdots,x_k)$ 表示那里所述的弧。对于每一个有着极大周期长度的函数，通过命 $f(x_1,\cdots,x_k)=x_{k+1}$，如果弧 $(x_2,\cdots,x_k)$ 的后边接有 $(x_2,\cdots,x_{k-1})$，我们可以定义一条唯一对应的欧拉线路（如果一条欧拉线路仅仅是另一条的循环排列，则我们认为它们是相同的）。现在在习题 21 的意义下 $G_k=G_{k-1}^*$，所以 G_k 有 $m^{m^{k-1}-m^{k-2}}$ 倍于 G_{k-1} 之多的有向子树；由归纳法，G_k 有 $m^{m^{k-1}-1}$ 个有向子树，而且有 $m^{m^{k-1}-k}$ 个具有一个给定的根。因此，由习题 22，具有极大周期的函数之个数，即是，以一个给定的弧开始的 G_n 的欧拉线路之条数，是 $m^{-k}(m!)^{m^{k-1}}$（对于 $m=2$，这个结果属于 C·弗莱·萨姆特-玛丽（C. Flye Samte-Marie），Plutermediaire des Mathematicians 1 (1891)，107–110）。

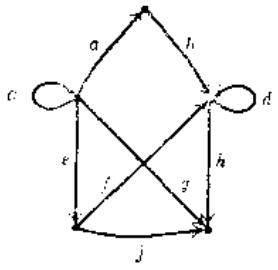
24. 定义一个新的有 E_j 个 e_j 的复写的有向图形，对于 $0\leqslant j\leqslant m$。这个图形是平衡的，且因之由定理 G 我们知道有一条欧拉线路 $(e_0,\cdots)$。通过从这条欧拉线路删去边 e_0，就得到所求的通路。

25. 对于有共同的初始顶点的弧的集合，指定一个次序，并且对于有共同的最终顶点的弧的集合，也指定一个次序。现在对于每条弧 e，设节点中表示 e 的场如下：如果 e' 是下一条弧（在所假定的编序下），对于它 $\mathrm{init}(e')=\mathrm{init}(e)$，则命 ALINK 指向 e'，且命 ATAG="+"；然而，如果 e 是具有这个初始顶点的最后一边弧（在所假定的编序之下），则命 ATAG="−"并命 ALINK 是一指向头一条使得 $\mathrm{init}(e)=\mathrm{fin}(e')$ 的弧 e' 的指针；如果不存在这样的 e'，则命 ALINK=Λ。以同样的规则定义 BLINK 和 BTAG，把 init 和 fin 的作用颠倒过来。

例：



arc	ALINK	ATAG	BLINK	BTAG
a	d	−	b	+
b	Λ	−	c	+
c	f	−	r	−
d	Λ	−	e	+
e	Λ	−	a	−
f	Λ	−	c	+
r	a	−	Λ	−



arc	ALINK	ATAG	BLINK	BTAG
a	c	+	b	−
b	a	−	d	+
c	e	+	a	−
d	h	+	f	+
e	g	−	f	−
f	j	+	h	−
g	c	−	h	+
h	b	−	j	+
j	e	−	Λ	−

注意：如果在这有向树表示中，我们加上从 H 到它自己的另一条弧，则我们就得到一种有趣的情况；我们或者是得到以在表头处互相交换 LLINK，LTAG，RLINK，RTAG 的标准约定 2.3.1-(7)，或者（如果新的弧在这编序下排在最后）是得到在与树的根相关联的节点中的标准约定，但 RTAG="+"除外。

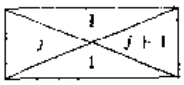
这一习题是以威·查·林奇（W. C. Lynch）告知作者的思想为基础的。揭示这个表示之进一步的性质，将是有趣的，例如，可以把树遍历算法与这一小节的欧拉线路之构造作一比较。

2.3.4.3 小节

1．根是空的序列；弧从 $(x_1, \cdots, x_n)$ 到 $(x_2, \cdots, x_{n-1})$。

2．取一个骨牌式样，并把它转动 180°以获得另一个骨牌式样；这两个式样通过 2×2 型的复制，以一种显然的方法来砌平面（不作进一步的转动）。

3．考虑骨牌式样



的集合，对于所有正整数 j。则右半平面可以以不可数多的方法来砌；但是，放在这平面之中心的方块，无论对距离放置什么有限的极限，它总可以向左继续。

4．系统地枚举所有可能的方法来砌一 $n \times n$ 方块，对于 $n = 1, 2, \cdots$，考虑在这些方块之内的圆环解。如果没有办法来砌这平面，则无限性引理告诉我们，有一个整数 n 没有 $n \times n$ 解。如果有一个办法来砌这个平面，则这假定告诉我们，存在一个 n，只有包含着一个产生这一圆环解的矩形的 $n \times n$ 解。因此，在两种情况下的每一种，这算法都将终止。（但正像下一习题所示，所述的假定是假的，而且事实上没有这样一种算法，它在有限步之内将确定是否存在一个方法，以给定的骨牌式样集合来砌这平面）。

5．通过注意这样一点开始，即在任何一个解中，我们需要以 2 × 2 组所复制的种类 $\begin{smallmatrix}\alpha\beta\\ \gamma\delta\end{smallmatrix}$。再者，步骤 1：仅仅考虑 α 正方形；我们证明，型式 $\begin{smallmatrix}\alpha b\\ cd\end{smallmatrix}$ 必须以 α 正方形的 2× 2 组来复制。步骤 $n \geqslant 1$：我们确定一个必然出现在一高度和宽度为 $2^n - 1$ 之交叉形式的区域里的型式。这些交叉的中间，有遍整个平面复制的型式 $\begin{smallmatrix}NaNb\\ NcNd\end{smallmatrix}$。

例如，在步骤 3 之后，我们将知道整个平面上 7×7 方块的内容，以单位长的小分条分开，每个是 8 个单位。在中心的类 Na 的 7×7 方块，有下列形式：

αα	βKQ	αb	βQP	αc	βBK	αb
γPJ	δNc	γRB	δQX	γLJ	δNb	γPB
αc	βDS	αd	βQTY	αc	βBS	αd
γPQ	δPJ	γPXB	δNa	γRQ	δRB	γRB
αα	βUK	αb	βDP	αα	βBK	αb
γTJ	δNc	γSB	δDS	γST	δNd	γTB
αc	βQS	αd	βUT	αc	βBS	αd

中间的列和中间的行，是刚刚在步骤 3 期间填入的"十字架"；其它四个 3×3 正方形在步骤 2 之后填入；这些恰好在 7×7 正方形的右边和下边的正方形，是在步骤 4 时有待填入的 15×15 交叉的一部分。

关于导致仅有 35 个骨牌式样之集合的类似的构造，只不过是有非圆环解的，见拉·米·鲁宾逊（R. M. Robinson），Inventions Math. 12(1971)，177–209。鲁宾逊也揭

示了一组六个多面体的形式，它仅仅非圆环地砌平面，甚至当允许转动和反射时也罢。

6. 设 k 和 m 是固定的。考虑一个有向树，它的每个顶点，对于某个 n，表示把 $\{1, \cdots, n\}$ 划分成不包含长度为 m 之算术级数的 k 个部分之一分划。划分 $\{1, \cdots, n+1\}$ 的一个节点，是划分 $\{1, \cdots, n\}$ 的一个节点的儿子，如果这两个划分在 $\{1, \cdots, n\}$ 上一致。如果有一条从根发出的无限通路，则我们将有一个方法来把所有整数划分成没有长度为 m 之算术级数的 k 个集合。因此，因无限性引理和范德瓦尔登定理，这个树是有限的（如果 $k=2$，$m=3$，则这个树可以很快地用手算来计算之，而且 N 的最小值为 9。见《纯粹数学研究》Studies in Pure Mathematics），L. 米尔斯基(L. Mirsky) 编(科学出版社，1971)，251-260，该文有关于范德瓦尔登怎样发现他的定理之证明的有趣的说明)。

7. 存在两个集合 S_0，S_1，它们把整数划分成使得其中任何一个都不包含任何无限的可计算的序列（参看习题 3.5-32），所以，特别没有无限的算术级数。由于没有办法把部分解置入一个其每个顶点均有有限次数的树中，因此不能应用定理K。

8. 命一"反例序列"是违背克鲁斯科尔定理的树的一个无限序列，如果存在这样的序列的话。假设定理为假；则命 T_1 是一个具有最小可能的节点数的树，它使得 T_1 可以成为一反例序列中的头一个树；如果已经选定了 $T_1, \cdots, T_j$，则命 T_{j+1} 是一个使得 $T_1, \cdots, T_j, T_{j+1}$ 成为一反例序列之开头的具有最小可能的节点数的树。这一过程定义了一反例序列 $\langle T_n \rangle$。这些 T 中刚好无根。现在我们非常细心地来考察这个序列：

(a) 假设有一个子序列 $T_{n_1}, T_{n_2}, \cdots$，对于它 $l(T_{n_1})$，$l(T_{n_2})$，…是一个反例序列。这是不可能的，因为否则 $T_1, \cdots, T_{n_1-1}, l(T_{n_1}), l(T_{n_2}), \cdots$ 将是一反例序列，与 T_{n_1} 的定义相矛盾。

(b) 由于(a)，仅仅有有限多的 j，对于它说来，对任何 $k>j$，$l(T_j)$ 不能被嵌入 $l(T_k)$ 中。因此，通过取 n_1 大于任何这样的 j，我们可以找到一个子序列，使得 $l(T_{n_1}) \subseteq l(T_{n_2}) \subseteq l(T_{n_3}) \subseteq \cdots$。

(c) 现在由习题 2.3.2-22 的结果，对于任何 $k>j$，$r(T_{n_j})$ 不可能嵌入 $l(T_{n_k})$ 中，否则 $T_{n_j} \subseteq T_{n_k}$。因此 $T_1, \cdots, T_{n_j-1}, r(T_{n_1}), r(T_{n_2}), \cdots$ 是一个反例序列。但这与 T_{n_1} 的定义相矛盾。

注意：克鲁斯科尔定理不象是简单地可由无限性引理推出的，尽管它们似乎有一种暧昧不清的关系；一般说来，当给定 $T_1, T_2, \cdots, T_n$ 时，有无限多的树 T，使得 $T_1 \not\subseteq T$，$T_2 \not\subseteq T$，…，$T_n \not\subseteq T$。

关于进一步的发展，见组合理论杂志(A)13(1972)，297-305。

2.3.4.4 小节

1. $$\ln A(z) = \ln z + \sum_{k\geqslant 1} a_k \ln\left(\frac{1}{1-z^k}\right) = \ln z + \sum_{k,t\geqslant 1} \frac{a_k z^{kt}}{t}$$

$$= \ln z + \sum_{t\geqslant 1} \frac{A(z^t)}{t}.$$

2. 通过微商，以及等置 z^n 的系数，我们得到恒等式

$$na_{n+1}=\sum_{k\geq 1}\sum_{d\mid k} da_d a_{n+1-k}$$

现在交换求和的次序。

1. (a) 至少对于$|z|<\frac{1}{4}$，$A(z)$肯定地收敛，因为a_n小于有序树b_{n-1}的个数。由于$A(1)$是无限的，而且所有系数为正，故有一正数$\alpha\leqslant 1$，使得对于$|z|<\alpha$，$A(z)$收敛，而且在$z=\alpha$处有一奇点。命$\psi(z)=(1/z)A(z)$；由于$\psi(z)>e^{zA(z)}$，我们看到$\psi(z)=m$意味着$z<\ln m/m$，所以$\psi(z)$是有界的，而且$\lim_{z\to\alpha-}\psi(z)$存在。于是$\alpha<1$，而且由阿贝尔极限定理$a=\alpha\cdot\exp\left(a+\frac{1}{2}A(\alpha^2)+\frac{1}{3}A(\alpha^3)+\cdots\right)$。

(b) $A(z^2)$，$A(z^3)$，…是解析的对于$|z|<\sqrt{\alpha}$，而且$\frac{1}{2}A(z^2)+\frac{1}{3}A(z^3)+\cdots$在一稍微更小的盘面里一致收敛。

(c) 如果$\partial F/\partial w=a-1\neq 0$，则隐函数定理意味着，在$(\alpha,a/\alpha)$的一个邻域内，有一解析函数$f(z)$使得$F(z,f(z))=0$。但这意味着$f(z)=(1/z)A(z)$，与$A(z)$在$\alpha$处奇异之事实相矛盾。

(d) 显然。

(e) $\partial F/\partial w=A(z)-1$且$|A(z)|<A(\alpha)=1$，因为$A(z)$的系数都为正。因此如同在(c)中那样，在所有这样的点处，$A(z)$是正则的。

(f) 邻近$(\alpha,1/\alpha)$，我们有等式$0=\beta(z-\alpha)+(\alpha/2)(w-1/\alpha)^2+$更高阶的项，其中$w=(1/z)A(z)$；所以这些按隐函数定理，$w$是$\sqrt{z-\alpha}$的一个解析函数。因而，有一区域$|z|<\alpha_1$减去一仿口$(\alpha,\alpha_1)$，其中$A(z)$具有所述形式(由于正号将最终地使系数成为负的，因而选定负号)。

(g) 所述形式的任何函数，渐近地有系数

$$\nu\frac{\sqrt{2\beta}}{\alpha^n}\binom{1/2}{n}\qquad\left(\text{注意}\binom{3/2}{n}=0\left(\frac{1}{n}\binom{1/2}{n}\right)\right)$$

(参看习题2.2.1-12)。关于进一步的细节，以及自由树之个数的近似值，见理·奥特，Ann. Math. (2)49(1948)，583–599。

5. $c_n=\sum_{j_1+2j_2+\cdots=n}\binom{c_1+j_1-1}{j_1}\cdots\binom{c_n+j_n-1}{j_n}-c_n,\quad n\geq 1$

因此

$$\begin{aligned}2C(z)+1+z&=(1-z)^{-c_1}(1-z^2)^{-c_2}(1-z^3)^{-c_3}\cdots\\&=\exp\left(C(z)+\frac{1}{2}C(z^2)+\cdots\right)\end{aligned}$$

我们求得$C(z)=z+z^2+2z^3+5z^4+12z^5+33z^6+90z^7+\cdots$。与$A(z)$没有明显的关系，虽然说可能存在某种关系，也是貌似有理的。

6. $zG(z)^2=2G(z)-2-zG(z^2)$；$G(z)=1+z+z^2+2z^3+3z^4+6z^5+11z^6+23z^7+\cdots$。见AMM56(1949)，697–699以查阅参考文献。

7. $g_n\sim c\alpha^n n^{-3/2}$，其中$c=.791603$，$\alpha=2.48325$。

8.



9. 如果有两个形心，则通过考虑从一个形心到另一个的一条道路，我们发现不可能有中间的点，所以任何两个形心是相邻的。一个树包含有三个互相相邻的顶点是不可能的，所以至多有两个。

10. 如果X，Y是相邻的，则命$s(X, Y)$是在X的Y子树中的顶点数。于是$s(X, Y)+s(Y, X)=n$。正文中的论证说明，如果Y是一个形心，则权$(X)=s(X, Y)$。因此如果X和Y两者都是形心，则权$(X)=$权$(Y)=n/2$。

借助于这个记号，继续进行正文中的论证，以证明如果$s(X, Y)\geqslant s(Y, X)$，则在X的Y子树中有一形心。所以如果两个具有m个顶点的自由树，通过X与Y之间的一条边连接之，则我们就得到一个自由树，其中$s(X, Y)=m=s(Y, X)$，而且必然有两个形心（即是X和Y）。

11. $zT(z)'=T(z)-1$；即是，$z+T(z)^{t-1}=T(z)^{t-1}$。由等式1.2.9-21，$T(z)=\sum_n A_n(1, -t)z^n$，所以$t$进树的个数是

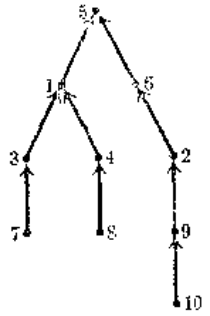
$$\binom{1+tn}{n}\frac{1}{1+tn}=\binom{tn}{n}\frac{1}{(t-1)n+1}$$

12. 考虑有向图形，它有一条从V_i到V_j的弧，对于所有的$i\neq j$。习题2.3.4.2-19的矩阵A_0，是一个组合的$(n-1)\times(n-1)$矩阵，具有$n-1$在对角线上，而-1在对角线外。所以其行列式为

$$(n+(n-1)(-1))n^{n-2}=n^{n-2}$$

是为具有一给定的根的有向树的个数（也可以利用习题2.3.4.2-20）。

13.

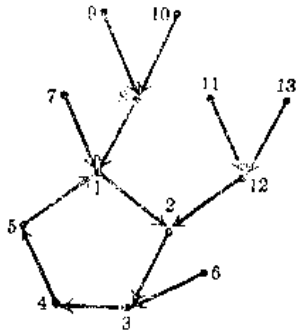


14. 真的，因为根将不变成一片叶，直到撤消所有其它分枝为止。

15. 在典型表示V_1，V_2，…，V_{n-1}中，$f(V_{n-1})$是考虑作一有向图形之有向树的拓扑分类，但这个次序一般地将不为算法2.2.3T所输出。可以改变算法2.2.3T，使得它确定V_1，V_2，…，V_{n-1}的值，如果步骤T6的“插入排队”操作，为一过程所代替的话。这个过程调整链接使得表的条款从前头到后尾以递增的次序出现，即是说，如果排队变成一优先排队的话。

17. 必须恰有一条回路x_1，x_2，…，x_k，其中$f(x_j)=x_{j+1}$和$f(x_k)=x_1$。我们将列举

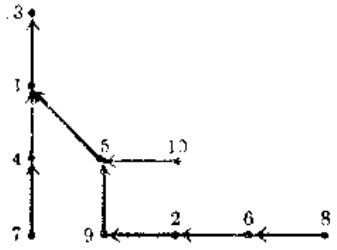
所有的 f，它有一条长度 k 的回路，使得每个 x 的迭代最终地进入这条回路。如同正文中那样定义典型表示 $f(V_1)$，$f(V_2)$，…，$f(V_{m-k})$；现在 $f(V_{m-k})$ 在回路中，通过写下回路的其余部分 $f(f(V_{m-k}))$，$f(f(V_{m-k}))$，等等，以获得一"典型表示"。例如，图形为



的具有 $m=13$ 的函数，导致表示 3，1，8，8，1，12，12，2，3，4，5，1。我们得到一个 $m-1$ 个数的序列，其中最后 k 个是不同的。反之，我们可以从任何这样的序列逆转这构造（假定 k 已知），因此恰有 $m(m-1)\cdots(m-k+1)m^{m-k-1}$ 个这样的具有一条 k 回路的函数（对于有关的结果，见习题 3.1-11）。

18. 为了从一个序列 s_1，s_2，…，s_{n-1} 重新构造这树，以 s_1 作为根开始并逐次地向这树附上那些指向 s_1，s_2，…的弧。如果顶点 s_k 在事先已经出现，则保留导向 s_{k+1} 的弧的初始顶点，但无名，否则给这个顶点以名称 s_k。在放置了所有 $n-1$ 条弧之后，通过使用未曾出现的数来给还没有名称的所有顶点起名，以这些无名顶点的建立次序，以递增次序给它们以名称。

例如，从 3，1，4，1，5，9，2，6，5 我们将构造



在这个方法与正文中的方法之间，没有简单的联系。有可能有更多的表示；见埃·哈·内维尔（E. H. Neville）的论文，《剑桥哲学协会会报》(Proc. Cambridge Phil. Soc.)49 (1953), 381-385。

19. 考虑这样树的典型表示。我们是问 $(x_1+\cdots+x_n)^{n-1}$ 有多少项，有 k_0 个指数 0，k_1 个指数 1，等等。这显然就是，这样一项的系数，乘以这样一些项的项数，即是，
$(n-1)!/(0!)^{k_0}(1!)^{k_1}\cdots(n!)^{k_n}$ 乘以 $n!/k_0!\,k_1!\cdots k_n!$。

21. 具有 $2m$ 个顶点的一个也没有；如果有 $n=2m+1$ 个顶点，则答案可从习题 20 对于 $k_0=m+1$，$k_2=m$ 得到，即是 $\binom{2m+1}{m}(2m)!/2^m$。

22. 确切地为 n^{n-2}；因为如果 X 是一个特殊的顶点，则自由树与以 X 为根的有向树一

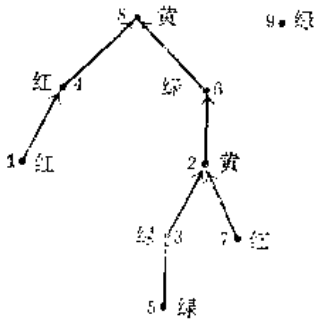
一对应。

23. 有可能对每一未标号的有序树，以 $n!$ 种方式进行标号，而且这些有标号的有序树，每一个都不相同。所以总数是 $n!\ b_{n-1}=(2n-2)!\ /(n-1)!$。

24. 对于另一个根，与对于一个给定的根一般多，所以答案一般是 $1/n$ 乘习题 23 中的答案；而且在此特殊情况下答案为 30。

25. 对于 $0\leqslant q<n$，$r(n,\ q)=(n-q)n^{r-1}$（在等式(22)中，特殊情况 $s=1$）。

26.



27. 给定一个从 $\{1,\ 2,\ \cdots,\ r\}$ 到 $\{1,\ 2,\ \cdots,\ q\}$ 的函数 g，使得加上从 V_k 到 $U_{g(k)}$ 的一些弧不引入有向回路。构造一个序列 $a_1,\ \cdots,\ a_r$ 如下：如果对于任何 $j\neq k$，没有从 V_j 到 V_k 的有向通路，则称顶点 V_k 是“自由的”。因为没有有向回路，故至少必有一个自由顶点。命 b_1 是使得 V_{b_1} 为自由顶点的最小整数；并假定已经选定 $b_1,\ \cdots,\ b_t$。命 b_{t+1} 是不同于 $b_1,\ \cdots,\ b_t$ 的最小整数，对于它，当 $1\leqslant k\leqslant t$ 时，通过删去从 V_{b_k} 到 $U_{g(b_k)}$ 的弧所得到的图形中，$V_{b_{t+1}}$ 是自由的。这个规则定义了整数 $\{1,\ 2,\ \cdots,\ r\}$ 的一个排列 $b_1,\ \cdots,\ b_r$。命 $a_k=g(b_k)$ 对于 $1\leqslant k\leqslant r$；这就定义了一个序列，使得 $1\leqslant a_k\leqslant q$ 对于 $1\leqslant k<r$，而且 $1\leqslant a_r\leqslant p$。

反之，如果给定了这样一个序列，则称顶点 V_k 是“自由的”。如果不存在这样的 j，使得 $a_j>p$ 和 $f(a_j)=k$。由于 $a_r\leqslant p$，至多有 $r-1$ 个非自由的顶点。命 b_t 是使 V_{b_t} 为自由顶点的最小整数；并且假设 $b_1,\ \cdots,\ b_t$ 已经选定，命 b_{t-1} 是不同于 $b_1,\ \cdots,\ b_t$ 的最小整数，对于它，$V_{b_{t+1}}$ 对于序列 $a_{t+1},\ \cdots,\ a_r$ 是自由的。这个规则定义了整数 $\{1,\ 2,\ \cdots,\ r\}$ 的一个排列 $b_1,\ \cdots,\ b_r$。命 $g(b_k)=a_k$ 对于 $1\leqslant k\leqslant r$；这就定义了一个函数，使得把从 V_k 到 $U_{g(b_k)}$ 的一些弧加进来不引进有向回路。

28. 命 f 是从 $\{2,\ \cdots,\ m\}$ 到 $\{1,\ 2,\ \cdots,\ n\}$ 的 n^{m-1} 个函数中的任一个，并考虑有向图形，具有顶点 $U_1,\ \cdots,\ U_m,\ V_1,\ \cdots,\ V_n$ 和从 U_k 到 $V_{f(k)}$ 对于 $1<k\leqslant m$ 的弧。以 $p=1$，$q=m$，$r=n$ 应用习题 29，以证明有 m^{n-1} 种方法把从诸 V 到诸 U 的更多的弧加进来，以得到根为 U_1 的有向树。因为在自由树的所求的集合与根为 U_1 的有向树的集合之间，有一个一一对应关系，故答案是 $n^{m-1}m^{n-1}$〔注意：这个构造可进行广泛的推广；见唐·克努特，《加拿大数学杂志》(Canadian J. Math.)20(1968)，1077–1086〕。

29. 如果 $y=x^t$，则 $(tz)\ y=\ln y$，而且我们看到，只须对于 $t=1$ 来证明这恒等式。现在如果 $zx=\ln x$，则我们通过习题 25 知道对于非负整数 m，$x^m=\Sigma_k F_k\ (m,\ 1)z^k$。因此

$$x^r = e^{zxr} = \sum_k \frac{(zxr)^k}{k!} = \sum_{j,k} \frac{r^k z^{k-j} E_j(k, 1)}{k!}$$

$$= \sum_{j,k} \frac{z^k}{k!} \binom{k}{j} j! \; E_j(k-j, 1) r^{k-j}$$

$$= \sum_k \frac{z^k}{k!} \sum_j \binom{k-1}{j} k^j r^{k-j} = \sum_k z^k F_k(r, 1)$$

30. 所述的每个图形定义一个集合 $C_x \subseteq \{1, \cdots, n\}$，这里 j 在 C_x 中，当且仅当，有一条从 t_j 到 r_j 的通路，对于某 $i \leqslant x$。对于一个给定的 C_x，所述的每个图形由两个独立的部分组成：对于 $i \leqslant x$ 和 $j \in C_x$，关于顶点 r_i, s_{jk}, t_j 的 $x(x+\epsilon_1 z_1 + \cdots + \epsilon_n z_n)^{\epsilon_1 + \cdots + \epsilon_n - 1}$ 个图形中的一个，其中 $\epsilon_j = 1$ 当且仅当 $j \in C_x$，加上关于剩下的顶点的 $y(y+(1-\epsilon_1)z + \cdots + (1-\epsilon_n)z_n)^{(1-\epsilon_1)+\cdots+(1-\epsilon_n)-1}$ 个图形中的一个。

31. $G(z) = z + G(z)^2 + G(z)^3 + G(z)^4 + \cdots = z + G(z)^2/(1 - G(z))$。因此 $G(z) = \frac{1}{4}(1 + z - \sqrt{1 - 6z + z^2})$。〔注意：等价于这个问题的另一个问题，是厄·施罗德（E. Schröder）提出和解决的，《数学杂志》(Zeitschrift für Mathematik)15 (1870), 361–376，他确定出在一个凸的 $(n+1)$ 角中插入非重叠的对角线的方法种数。对于 $n > 1$，这些数恰是习题 2.2.1–11 中所得到的值之一半，因为普拉特文法允许相关联的分析树的根节点有次数 1。习题 2.2.1–12 中计算了这个近似值〕。

32. 如果 $n_0 \neq 1 + n_2 + 2n_3 + 3n_4 + \cdots$ 则为 0（参照习题 2.3–21），否则为

$$(n_0 + n_1 + \cdots + n_m - 1)! \; / n_0! \; n_1! \; \cdots n_m!$$

为了证明这一结果，我们回想起，一个具有 $n = n_0 + n_1 + \cdots + n_m$ 个节点的未标号的树，通过在后根次序下节点的次数的序列 $d_1 d_2 \cdots d_n$ 来表征之（2.3.3 小节）。而且，这样一个次数的序列，对应于一个树，当且仅当，$\sum_{1 \leqslant j \leqslant k} (1 - d_j) > 0$ 对于 $0 < k \leqslant n$（波兰记号的这一重要性质，容易通过归纳法证明之；参照算法 2.3.3F，对于一个建立一树之函数 f，类似 2.3.2 小节的 TREE 函数）。特别是，d_1 必须为 0。因此我们问题的答案是，对于 $j > 0$，具有 j 的 n_j 个出现的序列 $d_2 \cdots d_n$ 的个数，即是多项式系数

$$\binom{n-1}{n_0 - 1, \; n_1, \; \cdots, \; n_m}$$

减去这样的序列 $d_2 \cdots d_n$ 的个数，对于这些序列，当某个 $k \geqslant 2$，有 $\sum_{2 \leqslant j \leqslant k} (1 - d_j) < 0$。

我们可以枚举后边的序列如下：命 t 是使得 $\sum_{2 \leqslant j \leqslant t} (1 - d_j) < 0$ 之极小者；则 $\sum_{2 \leqslant j < t} (1 - d_j) = -s$，其中 $1 \leqslant s < d_t$，而且我们可以形成子序列 $d_2' \cdots d_n' = d_{t-1} \cdots d_2 \, 0 \, d_{t+1} \cdots d_n$，对于 $j \neq d_t$ 它有 j 的 n_j 个出现，对于 $j = d_t$ 它有 j 的 $n_j - 1$ 个出现。现在讨论 $\sum_{2 \leqslant j \leqslant k} (1 - d_j')$：当 $k = n$ 时，等于 d_t；当 $k = t$ 时，等于 $d_t - s$；当 $k < t$ 时，小于 $d_t - s$（为证明这后一命题，注意

$$\sum_{2 \leqslant j \leqslant k} (1 - d_j') = \sum_{2 \leqslant j < t} (1 - d_j) - \sum_{2 \leqslant j < t \cdot k} (1 - d_j) \leqslant \sum_{2 \leqslant j < t} (1 - d_j)$$

$$= d_t - s - 1)。$$

由此得出，给定 s 和任何序列 $d_2' \cdots d_n'$，这个构造可加以逆转：因此有一个给定的 d_t 和 s

之值的序列 $d_2\cdots d_n$ 的个数，是多项式系数

$$\binom{n-1}{n_0,\ \cdots,\ n_j-1,\ \cdots,\ n_m}$$

因此，通过对所有可能的 d_s 和 s 之值进行求和，就得到了对应于这些树的序列 $d_2\cdots d_n$ 的个数：

$$\sum_{0\leqslant j<m}(1-j)\binom{n-1}{n_0,\ \cdots,\ n_j-1,\ \cdots,\ n_m}=\frac{(n-1)!}{n_0!\ n_1!\ \cdots n_m!}\sum_{0\leqslant j\leqslant m}(1-j)n_j$$

而且后面的和为 1。

这个结果的一个更为简单的证明，已经由乔·尼·拉尼给出(《美国数学协会会报》(Transactions of the American Math. Society)94(1960), 441-451)。如果 $d_1d_2\cdots d_n$ 是具有 j 的 n_j 个出现的任何序列，则恰有一个对应于一树的循环的重新配置 $d_k\cdots d_nd_1\cdots d_{k-1}$，也就是其中 k 使得 $\sum_{1\leqslant j<k}(1-d_j)$ 为极小时的极大者的重新配置。

以上两个方法中的每一个都可加以推广，以证明有 f 个树和 n_j 个次数 j 的节点的(有序和无标号的)森林个数是 $(n-1)!\ f/n_0!\ n_1!\ \cdots!\ n_m!$，假定条件 $n_0=f+n_2+2n_3+\cdots$ 被满足。

33. 考虑具有标号为 1 的 n_1 个节点，标号为 2 的 n_2 个节点，…，以致标号为 j 的每个节点的树的个数有次数 e_j。命这个数为 $c(n_1,\ n_2,\ \cdots)$，而且所说明的次数 e_1，e_2，…被看成是固定的。生成函数 $G(z_1,\ z_2,\ \cdots)=\sum c(n_1,n_2,\cdots)z_1^{n_1}z_2^{n_2}\cdots$ 满足恒等式 $G=z_1G^{e_1}+\cdots+z_rG^{e_r}$，因为 $z_jG^{e_j}$ 枚举其根标号为 j 的那些树。而且由前面习题的结果，

$$c(n_1,\ n_2,\ \cdots)=\begin{cases}\dfrac{(n_1+n_2+\cdots-1)!}{n_1!\ n_2!\ \cdots}, & \text{如果}(1-e_1)n_1+(1-e_2)n_2+\cdots=1;\\ 0, & \text{否则。}\end{cases}$$

更一般地，因为 G^f 枚举有这样标号的有序森林的个数，故对于整数 $f\geqslant 0$ 我们有

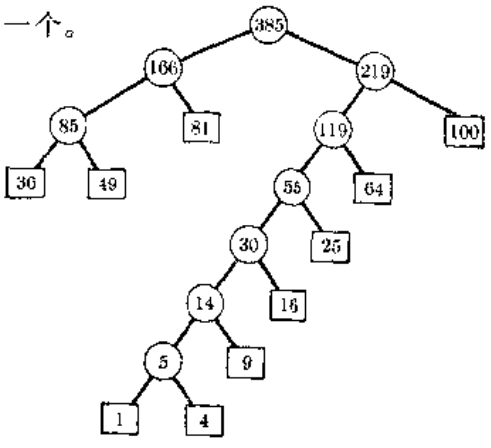
$$w^f=\sum_{f-(1-e_1)n_1+(1-e_2)n_2+\cdots}\frac{(n_1+n_2+\cdots-1)!\ f}{n_1!\ n_2!}z_1^{n_1}z_2^{n_2}\cdots。$$

当 $r=\infty$ 时，这些公式是有意义的，而且实际上它们等价于拉格朗日的反演公式。

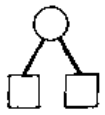
2.3.4.5 小节

1. 是，全部有 $\binom{8}{5}$，因为编号为 8，9，10，11，12 的节点，可以附加于 4，5，6 和 7 之下的 8 个位置中的每一个。

2.

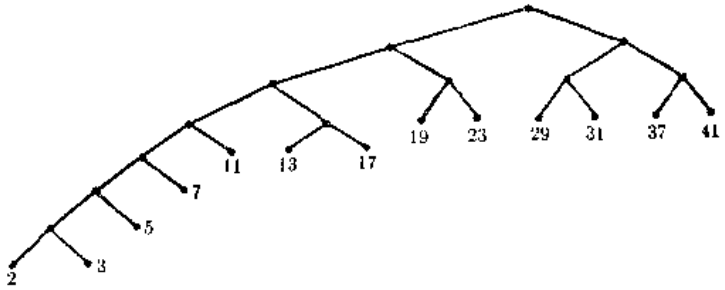


3. 通过对m用归纳法，这条件是必要的。反之，如果$\Sigma_{1\leqslant j\leqslant m}2^{-l_j}=1$，则我们要来构造一个具有这些通路长度的扩充的二叉树。如果$m=1$，则$l_1=1$而且这构造是平凡的。否则我们可以假定诸l是有序的，使得对于$1\leqslant q\leqslant m$的某q，$l_1=l_2=\cdots=l_q>l_{q+1}\geqslant l_{q+2}\geqslant\cdots\geqslant l_m>0$。现在$2^{l_1-1}=\Sigma_{1\leqslant j\leqslant m}2^{l_1-l_j-1}=\frac{1}{2}q+$整数，因此$q$为偶数。通过对$m$用归纳法，有一通路长度为$l_1-1$，$l_3$，$l_4$，…，$l_m$的树；取这样一个树并且以



来代替在l_1-1层的外部节点之一。

4. 首先，通过赫夫曼方法找一个树。如果$w_j<w_{j+1}$，则$l_j\geqslant l_{j+1}$，否则这个树将不是最优的。习题3答案中的构造，现在向我们给出了另一个树，具有这些相同通路长度而且具有适当顺序下的权数。例如，树(11)变成



见 CACM 7 (1964), 166–169。

5. (a) $b_{np}=\sum_{\substack{k+l=n-1\\r+s+n-1=p}}b_{kr}b_{ls}$。

因此，$zB(w,\ wz)^2=B(w,\ z)-1$。(b) 关于w取偏导数：

$$2zB(w,\ wz)(B_w(w,\ wz)+zB_z(w,\ wz))=B_w(w,\ z)$$

因此如果$H(z)=B_w(1,\ z)=\Sigma_n h_n z^n$，则我们求得$H(z)=2zB(z)(H(z)+zB'(z))$；而且这对于$B(z)$的已知的公式导出

$$H(z)=-\frac{1}{1-4z}-\frac{1}{z}\left(\frac{1-z}{\sqrt{1-4z}}-1\right),$$

所以

$$h_n=4^n-\frac{3n+1}{n+1}\binom{2n}{n}。$$

平均值是h_n/b_n。(c)近似地，这得出$n\sqrt{\pi n}-3n+O(\sqrt{n})$。

关于一些类似问题的解，见约翰·赖尔登，IBM 研究和发展杂志。4(1960), 473–478；艾·伦尼 (A. Renyi) 和乔·泽克勒斯 (G. Szekeres)，《澳大利亚数学协会杂志》(J. Australian Math. Soc.) 7 (1967), 497–507；以及习题 2.3.1–11。

6. $n+s-1=tn$。

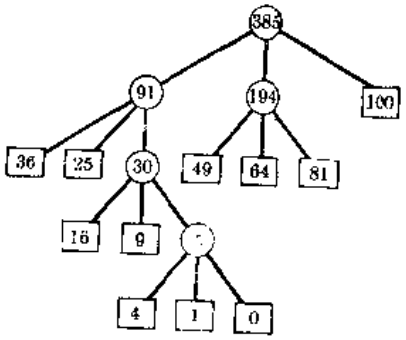
7. $E=(t-1)I+tn$。

8. $\sum_{1\le k\le n}\lfloor\log_t((t-1)k)\rfloor=nq-\sum_{\substack{0\le k\le n\\ \exists j.\,(t-1)k+1=t^j}}k$。 （部分求和）

这后一和数可以改写成 $\sum_{1\le j\le q}(t^j-1)/(t-1)$。

9. 对树大小用归纳法。

10. 必要时，通过加上额外的 0 权数，我们可以假定 $m \bmod (t-1)=1$。为得到具有极小的加权通路长度的 t 叉树，在每一步骤都组合最小的 t 个值，并以它们的和来代替它们。证明实质上与二叉树相同。所求的三叉树为



11. “杜威记号”是节点号数的二进表示。

2.3.5 小节

1. 一个列表结构是一个这样的有向图形，其中离开每个顶点的弧是有序的，并且有输出次数 0 的某些顶点被标记为“原子”。而且，有一个顶点 S，使得对于所有顶点 $V\neq S$，有一条从 S 到 V 的有向通路（对于逆于弧的方向，S 将是一个“根”）。

2. 不是在相同的方式下，因为在通常表示下的穿线链接，回过头来引向对于子列表不是唯一的“FATHER”。习题 2.3.4.2-25 中所讨论的表示也许可用（尽管在写作的时候，尚未剖析这个思想）。

3. 如果仅有 PO 有待加标记，则这算法肯定地能正确地工作。如果有 $n>1$ 个节点有待标记，则注明 ATOM(PO)=0，步骤 E4 而后置 ALINK(PO)←A 并执行以 ALINK(PO)代替 PO 和以 PO 代替 T 的算法。由归纳法（注意由于 MARK(PO)现在为 1，故由步骤 E4 和 E5，所有指向 PO 的链接都等价于 A），我们看到，最终地我们将标记以 ALINK(PO) 开始和不通过 PO 诸通路上的所有节点；然后我们将以 T=PO 和 P=ALINK(PO)达到步骤 E6。现在由于 ATOM(T)=1，故步骤 E6 恢复 ALINK(PO) 和 ATOM(PO)，而且我们到达步骤 E5。步骤 E5 置 BLINK(PO)←A，等等，而且一个类似的论证表明，我们将最终地标记以 BLINK(PO)开始和不通过 PO 或者从 ALINK(PO) 可以到达的节点诸通路上的所有节点。然后我们将以 T=PO，P=BLINK(PO) 到达步骤 E6，而且最后我们以 T=A，P=PO 到达步骤 E6。

4. 下列程序加上了在处理原子速度方面所提议的改进，这一改进是在正文中，在算法 E 的陈述之后提出的。

在算法的步骤E4和E5中，我们要来测试是否MARK(Q)=0。如果NODE(Q)=+0，则这是一种非常情况。通过置它等于-0而且就象它开始时就是-0那样来看待，就可以适当地加以处理，因为它有ALINK和BLINK两者均为Λ。这一简化在下面的计时的计算中未反映出来。

rI1≡P，rI2≡T，rI3≡Q，rX≡-1（为设置诸MARK）。

```
01    MARK    EQU   0:0
02    ATOM    EQU   1:1
03    ALINK   EQU   2:3
04    BLINK   EQU   4:5
05    E1      LD1   PO              1        E1. 初始化。P←PO
06            ENT2  0               1        T← Λ
07            ENTX  -1              1        rX←-1
08    E2      STA   0,1(MARK)       1        E2. 标志。MARK(P)←"-"
09    E3      LDA   0,1(ATOM)       1        E3. 原子?
10            JAZ   E4              1        如果 ATOM(P) = 0 则跳转
11    E6      J2Z   DONE            n        E6. 上升
12            ENT3  0,2             n-1      Q←T
13            LDA   0,3(ATOM)       t
14            JAN2  1F              t        如果 ATOM(T) = 1 则跳转
15            LD2   0,3(BLINK)      t2       T←BLINK(Q)
16            ST1   0,3(BLINK)      t2       BLINK(Q)←P
17            ENT1  0,3             t2       P←Q
18            JMP   E6              t2
19    1H      STZ   0,2(ATOM)       t1       ATOM(T)←0
20            LD2   0,3(ALINK)      t1       T←ALINK(Q)
21            ST1   0,3(ALINK)      t1       ALINK(Q)←P
22            ENT1  0,3             t1       P←Q
23    E5      LD3   0,1(BLINK)      n        E5. 下降 BLINK。Q←BLINK(P)
24            J3Z   E6              n        如果 Q = Λ 则跳转
25            LDA   0,3             n-b2
26            STX   0,3(MARK)       n-b2     MARK(Q)←"-"
27            JANP  E6              n-b2     如果 NODE(Q) 已被标志，则跳转
28            LDA   0,3(ATOM)       t2+a2
29            JANZ  E6              t2+a2    如果 ATOM(Q) = 1 则跳转
30            ST2   0,1(BLINK)      t2       BLINK(P)←T
31    E4A     ENT2  0,1             n-1      T←P
32            ENT1  0,3             n-1      P←Q
33    E4      LD3   0,1(ALINK)      n        E4. 下降 ALINK。Q←ALINK(P)
34            J3Z   E5              n        如果 Q = Λ 则跳转
35            LDA   0,3             n-b1
36            STX   0,3(MARK)       n-b1     MARK(Q)←"-"
37            JANP  E5              n-b      1 如果 NODE(Q) 已被标志，则跳转
```

38	LDA	0,3(ATOM)	$t1+a1$	
39	JANZ	E5	$t1+a1$	如果 ATOM(Q) = 1 则跳转
40	STX	0,1(ATOM)	$t1$	ATOM(P)←1
41	ST2	0,1(ALINK)	$t1$	ALINK(P)←T
42	JMP	E4A	$t1$	T←P，P←Q，转到E4。 ■

由克希霍夫定律，$t1+t2+1=n$。总的时间是$(34n+4t1+3a-5b-8)u$，其中n是被标志的非原子节点个数，a是被标志的原子个数，b是在被标志的非原子节点中所遇到的A链接的个数，而$b1$是我们下降一ALINK的次数（$0\leqslant t1<n$）。

5.（下面是“已知最快的”标志算法）。

S1. 置MARK(PO)←1。如果ATOM(PO)=1，则算法终止；否则置S←0，R←PO，T←A。

S2. 置P←BLINK(R)。如果P=A或MARK(P)=1，则转到S3。否则置MARK(P)←1。现在如果ATOM(P)=1，则转到S3；否则如果S<N则置S←S+1，STACK[S]←P，并转到S3；否则转到S5。

S3. 置P←ALINK(R)。如果P=A或MARK(P)=1，则转到S4。否则置MARK(P)←1。现在如果ATOM(P)=1，则转到S4；否则置R←P并返回S2。

S4. 如果S=0，则终止本算法；否则置R←STACK[S]，S←S−1，并转到S2。

S5. 置Q←ALINK(P)。如果Q=A或者MARK(Q)=1，则转到S6；否则置MARK(Q)←1。现在如果ATOM(Q)=1，则转到S6；否则置ATOM(P)←1，ALINK(P)←T，T←P，P←Q，转到S5。

S6. 置Q←BLINK(P)。如果Q=A或者MARK(Q)=1，则转到S7。否则置MARK(Q)←1。现在如果ATOM(Q)=1，则转到S7；否则置BLINK(P)←T，T←P，P←Q，转到S5。

S7. 如果T=A，则转到S3。否则置Q←T。如果ATOM(Q)=1，则置ATOM(Q)←0，T←ALINK(Q)，ALINK(Q)←P，P←Q，并返回S6。如果ATOM(Q)=0，则置T←BLINK(Q)，BLINK(Q)←P，P←Q，并返回S7。 ■

参考文献：CACM 10(1967)，501-506。

6. 从废料收集的第二阶段（或许也包括初始阶段，如果所有的标志位这时皆置为0）。

7. 删去步骤E2和E3，并删去E4中的“ATOM(P)←1”。在步骤E5中置MARK(P)←1，以及在步骤E6中用“MARK(Q)=0”，“MARK(Q)=1”分别代替当前的“ATOM(Q)=1”，“ATOM(Q)=0”。这个思想是仅在已经标志了左子树之后来设置MARK位。这个算法甚至当树已经重迭（共享的）子树时也能进行工作；但是，它对于所有的递归列表结构，诸如以NODE(ALINK(Q))作为NODE(Q)之一祖宗的那些结构，不能进行工作（注意一个已标志节点的ALINK决不改变）。

8. 解法1：类似于算法E，但更为简单。

F1. 置T←A，P←PO。

F2. 置MARK(P)←1，并置P←P+SIZE(P)。

F3. 如果MARK(P)=1，则转到F5。

F4. 置Q←LINK(P)。如果Q≠A而且MARK(Q)=0，则置LINK(P)←T，T←P，P←Q，并转到F2。否则置P←P－1并返回F3。

F5. 如果T=A，则停止。否则置Q←T，T←LINK(Q)，LINK(Q)←P，P←Q－1，并返回F3。 ■

一个类似的算法，它有时减轻内存的"开销"，而且它避免所有指向诸节点之中的指针，已经由拉勒斯-埃里克·托勒里（Lars-Erik Thorelli）提出，BIT12(1972)，555-568。

解法2：类似于算法D。对于这一解法，我们假定SIZE场大得足以包含一个链接地址。这样一个假定对于问题的陈述可能是不正当的，但是它让我们使用一个比头一个稍为更快的方法，当其可应用时。

G1. 置T←A，MARK(P0)←1，P←P0+SIZE(P0)。

G2. 如果MARK(P)=1，则转到G5。

G3. 置Q←LINK(P)，P←P－1。

G4. 如果Q≠A且MARK(Q)=0，则置MARK(Q)←1，S←SIZE(Q)，SIZE(Q)←T，T←Q+S。返回到G2。

G5. 如果T=A，则停止。否则置P←T并求使得MARK(Q)=1的Q=P，P－1，P－2，…的头一个值；置T←SIZE(Q)和SIZE(Q)←P－Q。返回到G2。 ■

9. **H1**. 置L←0，K←M+1，MARK(0)←1，MARK(M+1)←0。

H2. L增加1，而且如果MARK(L)=1则重复这一步骤。

H3. K减1，而且如果MARK(K)=0，则重复这一步骤。

H4. 如果L>K，则转到步骤H5；否则置NODE(L)←NODE(K)，ALINK(K)←L，MARK(K)←0，并返回H2。

H5. 对于L=1，2，…，K做下列各项：置MARK(L)←0。

如果ATOM(L)=0而且ALINK(L)>K，则置ALINK(L)←ALINK(ALINK(L))。

如果ATOM(L)=0而且BLINK(L)>K，则置BLINK(L)←ALINK(BLINK(L))。 ■

10. **Z1**. 〔初始化〕置F←P0，R⇐AVAIL，NODE(R)←NODE(F)，REF(F)←R（这里F和R是在所遇到的所有头部节点的REF场中建立起来的一个排队的指针）。

Z2. 〔开始新的列表〕置P←F，Q←REF(P)。

Z3. 〔向右推进〕置P←RLINK(P)。如果P=A，则转到Z6。

Z4. 〔复写一个节点〕置Q1⇐AVAIL，RLINK(Q)←Q1，Q←Q1，NODE(Q)←NODE(P)。

Z5. 〔转换子列表链接〕如果T(P)=1，则置P1←REF(P)，而且如果DREF(P1)=A则置REF(R)←P1，R⇐AVAIL，REF(P1)←R，NODE(R)←NODE(P1)，REF(Q)←R。如果T(P)=1而且REF(P1)≠A，则置REF(Q)←REF(P1)。转到Z3。

Z6. 〔移到下一列表〕置RLINK(Q)←A。如果REF(F)≠R，则置F←REF(REF(F))并返回Z2。否则置REF(R)←A，P←P0。

Z7. 〔最后的清扫〕置Q←REF(P)。如果Q≠A，则置REF(P)←A以及P←Q并重

复步骤 Z7。 ∎

当然，REF 场的这种用法使得不可能以算法D来进行废料收集；而且，算法D由于下述事实而被排斥，即是在复写期间诸表尚未很好地形成。

11．对于这个问题的答案，这里是一个直观的方法，它可以更加形式地写出：首先对于给定集合中的每一个列表，附加一个唯一的名称（例如，一个大写字母），在我们将要有的例子中．例如 $A=(a:C, b, a:F)$，$F=(b:D)$，$B=(a:F, b, a:E)$，$C=(b:G)$，$G=(a:C)$，$D=(a:F)$，$E=(b:G)$。现在造一个必须证明其为相等的列表对偶的表。对于这个表逐次地加上对偶，直到或者是由于我们有一个在头一层上不一致的对偶（于是原来给定的列表是不相等的），而发现一个矛盾为止，或者直到对偶的表不蕴含任何更进一步的对偶（于是原来给定的列表是相等的）为止。在本例子中，对偶的这个表原来仅含有给定的对偶 AB；然后它得到进一步的对偶 CF，EF（通过匹配 A 和 B），DG（从 CF），而后我们有一个自一致的集合。

为了证明这个方法的正确性，我们来观察，（a）如果它回答“不相等”的答案，则给定的列表是不相等的；（b）如果给定的列表是不相等的，则它回答“不相等”的答案；（c）它总是终止的。

12．当 AVAIL 表含有N个节点时，这里N是在下面的讨论中有待选定的确定的常数，起始另一个与主程序共享计算机时间的共行程序，而且做下列工作：（a）在AVAIL表上标记所有N个节点；（b）标记可为程序所访问的所有其它的节点；（c）把所有未标记的节点链接在一起以准备一个新的 AVAIL表，供当前的 AVAIL 表为空时使用，以及（d）在所有节点中把标志位复位。人们必须选择N和共享的时间比例，以便有一个确实的保证，俾在从 AVAIL 表采用N个节点之前，完成（a），（b），（c）和（d）诸操作，而主程序充分快地运行着。有必要在步骤（b）中多加用心，以确保包括所有“可为程序访问”的节点，当程序继续运行时；这里省略了细节。如果在（c）中形成的表有少于N个的节点，则由于内存空间可能成为穷尽了，因此势必终于停止下来〔见小盖伊·I·斯蒂尔（Guy L.Steele Jr.），CACM 18 (1975), 495～508, 以及P·韦德勒(P.Wadler), CACM 19 (1976)，关于进一步的信息，待出版〕。

2.4 节

1．先根次序。

2．实际上与所建立的Data Table的条款数成比例。

3．把步骤A5改变成：“A5′〔撤消顶层〕撤消顶上堆栈条款；而且如果在堆栈顶上的新的层号是≥L，则命（L1，P1）是在堆栈顶上的新的条款，而且重复这一步骤。否则置 BROTHER(P1)←Q并且然后命（L1, P1）为在堆栈顶上的新条款”。

4．规则（c）被违犯，当且仅当，有一个数据项，它的完备的限定 A_0 OF…OF A_n 也是对某个其它的数据项的一个 COBOL 访问。因此算法A将被扩充，随着把新的数据项加到 Data Table的同时，校验它的完备的限定是否包含在某个相同名称的以前的访问中，或者相同名称的某个以前的项的完备的限定是否也是对于这个新的数据项的一个访问。这实际上意味着以适当的方式，把算法B附加到算法A上。

5．作下列的改动：

步骤	被代替者	代替者
B1.	$P \leftarrow LINK(P_0)$	P←LINK(INFO(T))
B2.	$k \leftarrow 0$	K←T
B3.	$k < n$	RLINK(K)≠A
B4.	$k \leftarrow k+1$	K←RLINK(K)
B6.	$NAME(S) = P_k$	NAME(S)=INFO(K)

6．算法B的一个简单的修改，使得它仅仅寻找完备的访问（如果在步骤B3中，$k=n$ 和 FATHER(S)≠A，或者如果在步骤B6中 $NAME(S) \neq P_k$，则置P←PREV(P)并转到B2)。这个思想是首先通过这个修改的算法B；然后，如果Q仍然是A，则就来实施未修改的算法。

7．MOVE MONTH OF DATE OF SALES TO MONTH OF DATE OF PURCHASES.MOVE DAY OF DATE OF SALES TO DAY OF DATE OF PURCHASES.MOVE YEAR OF DATE OF SALES TO YEAR OF DATE OF PURCHASES.MOVE ITEM OF TRANSACTION OF SALES TO ITEM OF TRANSACTION OF PURCHASES.MOVE QUANTITY OF TRANSACTION OF SALES TO QUANTITY OF TRANSACTION OF PURCHASES.MOVE PRICE OF TRANSACTION OF SALES TO PRICE OF TRANSACTION OF PURCHASES.MOVE TAX OF TRANSACTION OF SALES TO TAX OF TRANSACTION OF PURCHASES.

8．当且仅当α和β是一个初等项（说明一下，在作者关于算法C的头一稿中，没有适当地处理这一情况．这可能是值得注意的，而且实际上，它把这算法弄得更复杂)。

9．“MOVE CORRESPONDINGα TO β”，如果α和β都不是初等的，则它等价于取遍分组α和β所共同的所有名称A的语句“MOVE CORRESPONDING *A* OF α TO *A* OF β”之集合（这是更优雅的叙述这定义的方式，比起正文中所给出的更为传统也更为烦琐的“MOVE CORRESPONDING”的定义)。我们可以验证算法C满足这个定义，用的是一个归纳法的证明，即是步骤C2到C5最终将以P＝PO和Q＝QO而终止。关于这个证明的进一步的细节，如同我们以前在一个“树的归纳法”里所已经多次做过的那样填充之（参照算法2.3.1T的证明)。

10．(a) 置S1←LINK(P_k)。然后重复地置S1←PREV(S1) 0次或多次直到或S1＝A (NAME(S)≠P_k) 或S1＝S (NAME(S)＝P_k) 为止。(b)置P1←P和然后置P1←PREV(P1) 0次或多次直到PREV(P1)＝A为止；对于变量Q1，Q进行类似的操作；而后测试是否P1＝Q1。或者，如果Data Table的条款是有序的，使得对于所有的P，PREV(P)＜P，则依赖于是否P＞Q，以及沿着更大的条款的PREV链来看看是否可遇到较小的条款，就可以以一种明显的方式来进行更快的测试。

11．通过增加一个新的链接场BROTHER1(P)≡SON(FATHER(P))，在步骤C4的速度方面将实现一点点改进。更有意义的是，我们可以修改SON和BROTHER的链接使得NAME(BROTHER(P))＞NAME(P)；这将相当大地加速步骤C3中的检索，因为

它仅仅需要一次扫描每个家族来寻找匹配的成员。因此，这将撤消出现在算法B或C中的仅有的“寻找”。用这样的解释，算法A和C很容易加以修改，而且读者可能会发现这是一个有趣的习题（然而，如果我们考虑MOVE CORRESPONDING语句的相对频率以及家族类的通常大小，则在真正的COBOL程序的翻译中，得到的加速将不是特别有意义的）。

12. 保持步骤B1，B2，B3不变；改变其它步骤成为：

B4. 置$k \leftarrow k+1$，$R \leftarrow \mathrm{LINK}(P_k)$。

B5. 如果$R=A$，则没有匹配；置$P \leftarrow \mathrm{PREV}(P)$并转到B2. 如果$R<S\leqslant \mathrm{SCOPE}(R)$，则置$S \leftarrow R$并转到B3. 否则置$R \leftarrow \mathrm{PREV}(R)$并重复步骤B5。 ∎

这一算法不适合于习题6的PL/I的约定。

13. 使用相同的操作，减去置NAME，FATHER，SON，以及BROTHER诸操作。每当撤消步骤A5中的堆栈顶的条款时，就置$\mathrm{SCOPE}(P1) \leftarrow Q-1$。当步骤A2中的输入穷尽时，简单地置$L \leftarrow 0$，并继续，然后如果在步骤A7中$L=0$，则这算法结束。

14. 下列算法，使用了一个辅助的堆栈（参照第8章），它有编了号的步骤以示出与正文的算法的一个直接对应。

C1. 置$P \leftarrow PO$，$Q \leftarrow QO$，并置堆栈的内容为空。

C2. 如果$\mathrm{SCOPE}(P)=P$或$\mathrm{SCOPE}(Q)=Q$，则输出(P, Q)以作为所求的对偶之一并转到C5. 否则把(P, Q)放入堆栈并置$P \leftarrow P+1$，$Q \leftarrow Q+1$。

C3. 确定P和Q是否指向具有相同名称的条款（参照习题10(b)）。如果是，则转到C2。如果不是，则命(P1，Q1)是堆栈顶上的条款；如果$\mathrm{SCOPE}(Q)<\mathrm{SCOPE}(Q1)$，则置$Q \leftarrow \mathrm{SCOPE}(Q)+1$并重复步骤C3。

C4. 命(P1，Q1)是堆栈顶上的条款。如果$\mathrm{SCOPE}(P)<\mathrm{SCOPE}(P1)$，则置$P \leftarrow \mathrm{SCOPE}(P)+1$，$Q \leftarrow Q1+1$，并且转回到C3。如果$\mathrm{SCOPE}(P)=\mathrm{SCOPE}(P1)$，则置$P \leftarrow P1$，$Q \leftarrow Q1$并且撤消堆栈顶上的条款。

C5. 如果堆栈为空，则算法结束。否则转到C4。 ∎

2.5 节

1. 在这样一个意外的环境下，可以使用类似堆栈的操作如下：设内存池区域是单元0到$M-1$，并设AVAIL指向最低的自由单元。为保留N个字，如果$\mathrm{AVAIL}+N>M$，则报告失败，否则置$\mathrm{AVAIL} \leftarrow \mathrm{AVAIL}+N$。为释放这N个字，只须置$\mathrm{AVAIL} \leftarrow \mathrm{AVAIL}-N$。

类似地，循环的类似排队的操作适宜于先进先出的规则。

2. 对于长度l的一个条款，存储空间的数量是$k\lceil l/(k-b)\rceil$，它有平均值$kL/(k-b)+(1-\alpha)k$，其中α假定是1/2，与k无关。当$k=b+\sqrt{2bL}$时，这个表达式取极小值（对于k的实数值）。所以选择k为刚好大于或刚好小于这个值的一个整数，无论那一个总给出$kL/(k-b)+\frac{1}{2}k$的最低值。例如，如果$b=1$和$L=10$，则$k \approx 1+\sqrt{20}=5$或6；两者都同样好。关于这问题的大量细节，请见JACM 12 (1965)，53-70。

4. rI1≡Q，rI2≡P。

```
A1    LDA     N                rA←N
```

```
        ENT2    AVAIL           P←LOC(AVAIL)
A2A     ENT1    0,2             Q←P
A2      LD2     0,1(LINK)       P←LINK(Q)
        J2N     OVERFLOW        如果P=A，则无单元了
A3      CMPA    0,2(SIZE)
        JG      A2A             如果N>SIZE(P)，则跳转
A4      SUB     0,2(SIZE)       rA←N-SIZE(P)=K
        JANZ    *-3             如果K≠0则跳转
        LDX     0,2(LINK)       LINK(P)
        STX     0,1(LINK)       →LINK(Q)
        STA     0,2(SIZE)       SIZE(P)←K
        LD1     0,2(SIZE)       最优的结束
        INC1    0,2             置rI1←P+K
```

5. 大概不是。刚才在单元P之前不可利用的存储区域，将随继变成可利用的，而且它的长度将增加数量K；99的增量将是不可忽视的。

6. 这个思想是每次在AVAIL表的不同部分试行检索。我们可以使用一个“遨游指针”，例如称它为ROVER，对它处理如下：在步骤A1，置Q←ROVER。在步骤A4之后，置ROVER←LINK(Q)。在步骤A2，在算法A的一个具体执行期间当头一次P=A时，置Q←LOC(AVAIL)并重复步骤A2。当第二次P=A时，这个算法将以失败而结束。在这种方式下，ROVER将趋于指向AVAIL表中的随机的一点，而且大小将更为平衡。在程序开始时，置ROVER←LOC(AVAIL)；在程序中，别的每处也有必要置ROVER成为LOC(AVAIL)，只要其地址等于ROVER之当前内容的那些区段，是由AVAIL表取出时。

7. 2000，1000对于大小800，1300的要求〔最坏的适合获得成功，而最好的适合反倒失败的这样的一个例子，已经由R.J.韦兰（R.J.Weiland）构造出来〕。

8. 在步骤A1，还置R←A。在步骤A2，如果P=A则转到A6。在步骤A3，转到A5而不是A4。加上新的步骤如下：

A5.〔较好的适合？〕如果R=A或M>SIZE(P)，则置R←Q和M←SIZE(P)。然后置Q←P并返回A2。

A6.〔找到一个？〕如果R=A，则本算法以失败而结束。否则置Q←R，P←LINK(Q)，并转到A4。 ∎

9. 显然，如果我们如此幸运地找出SIZE(P)=N，则我们有一个“最好的适合”，而且不需要进行进一步的检索（当仅有很少不同的区段大小时，这颇为经常地出现）。如果正使用的是类似于算法C中的一个“边界标志”，则有可能以分类的次序来维持这AVAIL表，所以平均说来，检索的长度可能减少到这表长度的1/2以至更少。但最好的解法是如同在6.2.3小节中所述的那样，把AVAIL表造入一个平衡的树结构中，如果预期它是很长的话。

10. 作如下的变动：

步骤B2，将“P>PO”改成“P≥PO”。

步骤B3，插入“如果PO+N≥P（而且P≠A），则置P←LINK(P)并重复步

骤 B3。”

步骤 B4，将“Q＋SIZE(Q)＝P0”改作“Q＋SIZE(Q)≥P0”；而且将“SIZE(Q)←SIZE(Q)＋N”改作“SIZE(Q)← P0＋N－Q”。

11. 如果 P0 大于 ROVER，则在步骤 B1 中我们可以置Q←ROVER 以代替Q←LOC(AVAIL)。如果在 AVAIL 表中有 n 个条款，则步骤 B2 的迭代的平均数是 $(2n+3)(n+2)/6(n+1)=\frac{1}{3}n+\frac{5}{6}-O\left(\frac{1}{n}\right)$。例如如果 $n=2$，则我们得到 9 种同等可能的情况，其中 P1 和 P2 指向两个现存的可利用区段：

	P0 < P1	P1 < P0 < P2	P2 < P0
ROVER = P1	1	1	2
ROVER = P2	1	2	1
ROVER = LOC(AVAIL)	1	2	3

这个图表说明了在每种情况下所需要的迭代数。平均是 $\frac{1}{9}\left(\binom{2}{2}+\binom{3}{2}+\binom{4}{2}+\binom{3}{2}+\binom{2}{2}\right)=\frac{1}{9}\left(\binom{5}{3}+\binom{4}{3}\right)=\frac{14}{9}$。

12. **A1**. 置 P←ROVER，F←0。

A2. 如果 P＝LOC(AVAIL) 而且 F＝0，则置P←AVAIL，F←1，并复步骤 A2。

如果 P＝LOC(AVAIL) 而且 F≠0，则算法以失败而结束。

A3. 如果 SIZE(P)≥N，则转到 A4；否则置 P←LINK(P) 并返回 A2。

A4. 置 ROVER←LINK(P)，K←SIZE(P)－N。如果K＜c（其中 c 是一个必须等于 2 或更大的常数），则置 LINK(LINK(P＋1))←ROVER，LINK(ROVER＋1)←LINK(P＋1)，L←P；否则置 L←P＋K，SIZE(P)←SIZE(L－1)←K，TAG(L－1)←“－”，SIZE(L)←N。置 TAG(L)←TAG(L＋SIZE(L)－1)←“＋”。∎

13. r1≡P，rX≡F，r12≡L。

```
LINK    EQU     4:5
SIZE    EQU     1:2
TSIZE   EQU     0:2
TAG     EQU     0:0
A1      LDA     N               rA← N
        SLA     3               移入 SIZE 场
        ENTX    0               F← 0
        LD1     ROVER           P←ROVER
        JMP     A2
A3      CMPA    0,1(SIZE)
        JLE     A4              如果N≤SIZE(P)则跳转
        LD1     0,1(LINK)       P←LINK(P)
A2      ENT2    -AVAIL,1        r12←P-LOC(AVAIL)
        J2NZ    A3
```

```
        JXNZ    OVERFLOW        F≠0?
        ENTX    1               置F←1
        LD1     AVAIL(LINK)     P←AVAIL
        JMP     A2
A4      LD2     0,1(LINK)
        ST2     ROVER           ROVER←LINK(P)
        LDA     0,1(SIZE)       rA≡K←SIZE(P)-N
        SUB     N
        CMPA    =c=
        JGE     1F              如果K≥c则跳转
        LD3     1,1(LINK)       rI3←LINK(P+1)
        ST2     0,3(LINK)       LINK(rI3)←ROVER
        ST3     1,2(LINK)       LINK(ROVER+1)←rI3
        ENT2    0,1             L←P
        LD3     0,1(SIZE)       rI3←SIZE(P)
        JMP     2F
1H      STA     0,1(SIZE)       SIZE(P)←K
        LD2     0,1(SIZE)
        INC2    0,1             L←P+K
        LDAN    0,1(SIZE)       rA←-K
        STA     -1,2(TSIZE)     SIZE(L-1)←K, TAG(L-1)←"-"
        LD3     N               rI3←N
2H      ST3     0,2(TSIZE)      TAG(L)←"+", 且置SIZE(L)←rI3
        INC3    0,2
        STZ     -1,3(TAG)       TAG(L+SIZE(L)-1)←"+"
```

14. (a) 在步骤C2中，需要这个场来定位区段的开始。它可以用一个指向这区段的头一个字的链接来加以代替。(b) 由于经常有必要保留N个以上的字（例如，如果$K=1$），因而需要这个场，而且，当这个区段被随继释放时，必须知道保留的数量。

15, 16. rI1≡PO, rI2≡P1, rI3≡F, rI4≡B, rI6≡$-N$。

```
D1      LD1     PO              D1.
        LD2     0,1(SIZE)
        ENN6    0,2             N←-SIZE(PO)
        INC2    0,1             P1←PO+N
        LD5     0,2(TSIZE)
        J5N     D4              如果TAG(P1)="-"则转D4
D2      LD5     -1,1(TSIZE)     D2.
        J5N     D7              如果TAG(PO-1)="-"则转D7
D3      LD3     AVAIL(LINK)     D3.置F←AVAIL
        ENT4    AVAIL           B←LOC(AVAIL)
        JMP     D5              转D5
D4      INC6    0,5             D4.N←N+SIZE(P1)
        LD3     0,2(LINK)       F←LINK(P1)
        LD4     1,2(LINK)       B←LINK(P1+1)
```

```
        CMP2    ROVER          (这部分是由于
        JNE     * + 3          习题12的ROVER的特性:
        ENTX    AVAIL          如果P1=ROVER, 则
        STX     ROVER              置ROVER←LOC(AVAIL).)
        DEC2    0,5            P1←P1+SIZE(P1)。
        LD5     -1,1(TSIZE)
        J5N     D6             如果TAG(P0-1)="-"则转D6
D5      ST3     0,1(LINK)      D5.LINK(P0)←F
        ST1     1,1(LINK)      LINK(P0+1)←B
        ST1     1,3(LINK)      LINK(F+1)←P0
        ST1     0,4(LINK)      LINK(B)←P0
        JMP     D8             转D8
D6      ST3     0,4(LINK)      D6.LINK(B)←F
        ST4     1,3(LINK)      LINK(F+1)←B
D7      INC6    0,5            D7.N←N+SIZE(P0-1)
        INC1    0,5            P0←P0-SIZE(P0-1)
D8      ST6     0,1(TSIZE)     D8.SIZE(P0)←N, TAG(P0)←" "
        ST6     -1,2(TSIZE)    SIZE(P1-1)←N, TAG(P1-1)←"-"  ■
```

17. 两个 LINK 场都等于 LOC(AVAIL)。

18. 算法A保留一个大区段的高末端。当存储完全可利用时，头一个适合的方法实际上是通过保留高序单元开始的，但一旦这些变成再次可利用的时，它们就不是重新保留的，因为一个"适合"通常已经在低单元找到；于是在内存的低末端之初始的大的区段，很快地消失于"头一个适合"。然而一个大的区段很少是"最好的适合"的，所以在内存的开始，最好的适合的方法保留一个大的区段。

19. 利用习题12的算法，但从步骤A4删去对于SIZE(L-1)，TAG(L-1)，以及TAG(L+SIZE(L)-1)的访问；并且在步骤A3的开始插入下列诸动作："置P1←P+SIZE(P)。如果TAG(P1)="-"，则置LINK(LINK(P1)+1)←LINK(P1+1)，LINK(LINK(P1+1))←LINK(P1)，SIZE(P)←SIZE(P)+SIZE(P1)，并重复步骤A3。否则："

显然（2），（3），（4）的情况这里不能出现；对存储分配唯一的实际效果是，这里的检索将趋向于比习题12中的更长，而且有时K将小于C，尽管实际上在这个的前头还有另一个我们还不知道的可利用的区段。

（一个选择是采取解除内循环A3，而且仅仅在步骤A4中，在最后的分配之前或者在内循环中，当算法否则就将以失败结束时，来进行解除。这个选择要求进行一个模拟研究，来看看它是否是一个改进）。

20. 当在解除循环期间，发现了一个伙伴可利用时，我们要把这个区段从它的AVAIL[k]表中撤消，但我们不知道修改哪一些链接，除非（a）我们作了一个可能很长的检索，或者（b）这个表是双重链接的。

21. 如果 $n=2^k\alpha$，其中 $1\leqslant\alpha\leqslant 2$，则 a_n 是 $2^{2k+1}\left(\alpha-\frac{2}{3}\right)+\frac{1}{3}$，而且 b_n 是 $2^{2k-1}\alpha^2+2^{k-1}\alpha$。比率 a_n/b_n 对于大 n 实际上是 $4\left(\alpha-\frac{2}{3}\right)/\alpha^2$，当 $\alpha=1$ 和 2 时它取其极小值 4/3，

而当 $\alpha = 1 - \frac{1}{3}$ 时取其极大值 3/2。所以 a_n/b_n 不趋向极限，它在这两个极端之间振荡。

22. 这个思想要求在 11 个字区段的若干字中，有一个 TAG 场，而不仅在头一个字中。这是一个切实可行的思想，假定这些额外的 TAG 位可以抽出，而且它显得特别适合于计算机硬件中使用。

23. 011011110100；011011100000。

24. 这在程序中引进了一个故障；当 TAG(0)= 1 时我们可能到达步骤 S1；因为 S2可能返回到 S1。为使它能够工作，在步骤 S2 中，在 "L←P" 之后，加上 "TAG(L)← 0"（换成假定 TAG(2^m) = 0 更为容易）。

25. 这一思想十分正确（注意，鉴定未必就是否定）。对于 $n < k \leqslant m$，表头 AVAIL 可能被消去；如果在步骤 R1，S1 中把 "m" 改变成 "n"，则可使用正文中的算法。初始条件 (13)，(14) 应该改变成指出大小 2^n 的 2^{m-n} 个区段，以代替大小 2^m 的一个区段。

26. 使用M的二进表示，我们可容易地修改初始条件 (13)，(14)，使得所有的内存单元被划分成大小为 2 之乘幂的一些区段。在算法 S 中，每当 P ≥ M时，TAG(P) 就应认为是 0。

27. r[1]≡ k，r[2]≡ j，r[3]≡ $j - k$，r[4]≡ L，LOC(AVAIL[j])=AVAIL+ j；假定对于 $0 \leqslant j \leqslant m$，有一辅助的表格 TWO[j]=2^j，存于单元TWO+ j 中。进一步假定 TAG= +，− 表示 TAG= 0，1；TAG(LOC(AVAIL[j]))="−"，但作为一哨兵，TAG(LOC(AVAIL[$m+1$]))="+"。

```
00    KVAL    EQU    5:5
01    TAG     EQU    0:0
02    LINKF   EQU    1:2
03    LINKB   EQU    3:4
04    TLNKF   EQU    0:2
05    R1      LD1    K                  1      R1. 找区段_
06            ENT2   0,1                1        j ← k
07            ENT3   0                  1
08            LD4    AVAIL,2(LINKF)     1
09    1H      ENT5   AVAIL,  2          1 + R
10            DEC5   0,4                1 + R
11            J5NZ   R2                 1 - R    如果 AVAILF[ j ]≠LOC(AVAIL
                                                  [ j ])则跳转
12            INC2   1                  R        增加 j
13            INC3   1                  R
14            LD4N   AVAIL,2(TLNKF)     R
15            J4NN   1B                 R        j ≤ m 吗?
16            JMP    OVERFLOW
17    R2      LD5    0,4(LINKF)         1        R2. 从表中撤消
18            ST5    AVAIL,2(LINKF)     1        AVAILF[ j ]←LINKF(L)
19            ENTA   AVAIL,2            1
20            STA    0,5(LINKB)         1        LINKB(L)←LOC(AVAIL[ j ])
```

```
21                STZ     0,4(TAG)          1       TAG(L)←0
22    R3          J2Z     DONE              1       R3.需要分开吗?
23    R4          DEC3    1                 R       R4.分开
24                DEC2    1                 R       减小 j
25                LD5     TWO,2             R       rI5←P
26                INC5    0,4               R       P←L+2^j
27                ENNA    AVAIL,2           R
28                STA     0,5(TLNKF)        R       TAG(P)←1, LINKF(P)←LOC
                                                    (AVAIL[j])
29                STA     0,5(LINKB)        R       LINKB(P)←LOC(AVAIL[j])
30                ST5     AVAIL,2(LINKF)    R       AVAILF[j]←P
31                ST5     AVAIL,2(LINKB)    R       AVAILB[j]←P
32                ST2     0,5(KVAL)         R       KVAL(P)←j
33                J3P     R4                R       转到R3
34    DONE        ...                               ■
```

28. rI1≡k，rI5≡P，rI4≡L；假定 TAG(2^m)＝"+"。

```
01    S1    LD4     L                 1         S1.是可利用的伙伴吗?
02          LD1     K                 1
03    1H    ENTA    0,4               1+S
04          XOR     TWO,1             1+S       rA←buddy_k(L)
05          STA     TEMP              1+S
06          LD5     TEMP              1-S       P←rA
07          LDA     0,5               1-S
08          JANN    S3                1+S       如果 TAG(P)=0 则跳转
09          CMP1    0,5(KVAL)         B+S
10          JNE     S3                B+S       如果 KVAL(P)≠k 则跳转
11    S2    LD2     0,5(LINKF)        S         S2.与伙伴结合
12          LD3     0,5(LINKB)        S
13          ST3     0,2(LINKF)        S         LINKF(LINKB(P))←LINKF(P)
14          ST2     0,3(LINKB)        S         LINKB(LINKF(P))←LINKB(P)
15          INC1    1                 S         增加 k
16          CMP1    TEMP              S
17          JL      1B                S
18          ENT4    0,5               A         如果L<P，则置L←P
19          JMP     1B                A
20    S3    LD2     AVAIL,1(LINKF)    1         S3.置入表中
21          ENNA    AVAIL,1           1
22          STA     0,4(0:4)          1         TAG(L)←1,LINKB(L)←LOC(AVAIL[k])
23          ST2     0,4(LINKF)        1         LINKF(L)←AVAILF[k]
24          ST1     0,4(KVAL)         1         KVAL(L)←k
25          ST4     0,2(LINKB)        1         LINKB(AVAILF[k])←L
26          ST4     AVAIL,1(LINKF)    1         AVAILF[k]←L                ■
```

29. 是的，但仅仅牺牲某检索，或者（更好地）想办法组装 TAG 位的一个附加的表

格（谨建议在算法S的运行期间不把伙伴联合到一起，而仅仅在算法R中这样做，如果没有足够大的区段满足这个要求的话；但这大概会导致一个很坏地分段的内存）。

33. **G1**.〔清除诸 LINK〕置P←1，并重复操作LINK(P)←A，P←P+SIZE(P)直到P=AVAIL为止（这仅仅在每个节点的头一个字中，把LINK场置成A；在大多数情况下，我们可以假定，这个步骤是不必要的，因为LINK(P)在以下的步骤G9中置成A，并且通过存储分配程序，它可被置成A）。

G2.〔初始化标记阶段〕置TOP←USE，LINK(TOP)←AVAIL，LINK(AVAIL)←A（如同在算法2.3.5D中那样，TOP指向一个堆栈的顶上）。

G3.〔弹出堆栈〕置P←TOP，TOP←LINK(TOP)。如果TOP=A，则转到G5。

G4.〔放置新链接到堆栈〕对于$1\leqslant k\leqslant T(P)$，进行下列操作：置Q←LINK(P+$k$)，而且如果Q≠A，LINK(Q)=A，则置LINK(Q)←TOP，TOP←Q。然后返回到G3。

G5.〔初始化下一阶段〕（现在P=AVAIL，而且加标记阶段已经完成，使得每个可访问的节点的头一个字，都有一非空的LINK。现在我们为了加快后面步骤的速度，希望把相邻的不可访问的节点联结起来，而且对于可访问的节点指定新的地址）。置Q←1，LINK(AVAIL)←Q，SIZE(AVAIL)←0，P←1（单元AVAIL正用作一个哨兵，以标记在随继的阶段中一个循环之结束）。

G6.〔指定新的地址〕如果LINK(P)=A，则转到G7。否则如果SIZE(P)=0，则转到G8。否则置LINK(P)←Q，Q←Q+SIZE(P)，P←P+SIZE(P)，并重复这一步骤。

G7.〔解除可利用的区域〕如果LINK(P+SIZE(P))=A，则增加SIZE(P)以SIZE(P+SIZE(P))并重复这一步骤。否则置P←P+SIZE(P)并返回G6。

G8.〔转换全部链接〕（现在每个可访问的节点的头一个字中的LINK场，都包含着节点将被移动到的地址）。置USE←LINK(USE)，以及AVAIL←Q。然后置P←1，并重复下列操作直到SIZE(P)=0为止：如果LINK(P)≠A，则置LINK(Q)←LINK(LINK(Q))对于$P<Q\leqslant P+T(P)$；然后不顾LINK(P)的值，置P←P+SIZE(P)。

G9.〔移动〕置P←1，并且重复下列操作直到SIZE(P)=0为止：置Q←LINK(P)，而且如果Q≠A，则置LINK(P)←A以及NODE(Q)←NODE(P)；然后无论Q=A或否，置P←P+SIZE(P)（操作NODE(Q)←NODE(P)意味着移动SIZE(P)个字；我们总是有$Q\leqslant P$，所以以从最小单元到最大单元的次序，来移动这些字是安全的）。

〔这个方法称为“LISP2废料收集程序”。另外一个稍微更复杂的紧凑的算法，已经为B.K.哈登（B.K.Haddon）和W.M.韦特（W.M.Waite）所描述，《计算机杂志》(Comp.J.) 10(1967)，162–165。也见B.韦格布赖特(B.Wegbreit)，《计算机杂志》16(1972)，204～208；D.A.扎弗（D.A.Zave），《信息处理文集》(Inf.Proc.Letters) 3(1975)，167–169〕。

34. 设TOP≡r11，Q≡r12，P≡r13，k≡r14，SIZE(P)≡r15。进一步假定Λ=0，和LINK(0)≠0以简化步骤G4。省略去步骤G1。

```
01    LINK    EQU     4:5
02    INFO    EQU     0:3
03    SIZE    EQU     1:2
04    T       EQU     3:3
05    G2      LD1     USE             1       G2. 初始化标记阶段。TOP←USE
06            LD2     AVAIL           1
07            ST2     0,1(LINK)       1       LINK(TOP)←AVAIL
08            STZ     0,2(LINK)       1       LINK(AVAIL)←Λ
09    G3      ENT3    0,1             a+1     G3. 弹出堆栈。P←TOP
10            LD1     0,1(LINK)       a+1     TOP←LINK(TOP)
11            J1Z     G5              a+1     如果TOP=Λ则转到G5
12    G4      LD4     0,3(T)          a       G4. 把新链接放入堆栈。k←T(P)
13    1H      J4Z     G3              b+a     k=0?
14            INC3    1               b       P←P+1
15            DEC4    1               b       k←k-1
16            LD2     0,3(LINK)       b       Q←LINK(P)
17            LDA     0,2(LINK)       b
18            JANZ    1B              b       如果LINK(Q)≠Λ则跳转
19            ST1     0,2(LINK)       a-1     否则置LINK(Q)←TOP,
20            ENT1    0,2             a-1     TOP←Q
21            JMP     1B              a-1
22    G5      ENT2    1               1       G5. 初始化下一阶段。Q←1
23            ST2     0,3             1       LINK(AVAIL)←1, SIZE(AVAIL)←0
24            ENT3    1               1       P←1
25            JMP     G6              1
26    1H      ST2     0,3(LINK)       a       LINK(P)←Q
27            INC2    0,5             a       Q←Q+SIZE(P)
28            INC3    0,5             a       P←P+SIZE(P)
29    G6      LDA     0,3(LINK)       a+1     G6. 指定新地址
30    G6A     LD5     0,3(SIZE)       a+c+1
31            JAZ     G7              a+c+1   如果LINK(P)=Λ则跳转
32            J5NZ    1B              a+1     如果SIZE(P)≠0则跳转
33    G8      LD1     USE             1       G8. 转换全部链接
34            LDA     0,1(LINK)       1
35            STA     USE             1       USE←LINK(USE)
36            ST2     AVAIL           1       AVAIL←Q
37            ENT3    1               1       P←1
38            JMP     G8P             1
39    1H      LD6     0,6(SIZE)       d
40            INC6    0,6             d       rI5←rI5+SIZE(P+SIZE(P))
41    G7      ENT6    0,3             c+d     G7. 解除可利用区域
42            INC6    0,5             c+d     rI6←P+SIZE(P)
43            LDA     0,6(LINK)       c+d
```

44		JAZ	1B	$c+d$	如果LINK(rI6) = A则跳转
45		ST5	0,3(SIZE)	c	SIZE(P)←rI5
46		INC3	0,5	c	P←P+SIZE(P)
47		JMP	G6A	c	
48	2H	DEC4	1	b	$k \leftarrow k-1$
49		INC2	1	b	Q←Q+1
50		LD6	0,2(LINK)	b	
51		LDA	0,6(LINK)	b	
52		STA	0,2(LINK)	b	LINK(Q)←LINK(LINK(Q))
53	1H	J4NZ	2B	$a+b$	如果$k \neq 0$则跳转
54	3H	INC3	0,5	$a+b$	P←P+SIZE(P)
55	G8P	LDA	0,3(LINK)	$1+a+c$	
56		LD5	0,3(SIZE)	$1+a+c$	
57		JAZ	3B	$1+a+c$	LINK(P) = A吗?
58		LD1	0,3(T)	$1+a$	$k \leftarrow T(P)$
59		ENT2	0,3	$1+a$	Q←P
60		J5NZ	1B	$1+a$	跳转,除非SIZE(P) = 0
61	G9	ENT3	1	1	G9.移动。P←1
62		ENT1	1	1	为MOVE指令置rI1
63		JMP	G9P	1	
64	1H	ST2	0,3(LINK)	a	LINK(P)←A
65		ST5	*+1(1:4)	a	
66		MOVE	0,3(*)	a	NODE(rI1)←NODE(P),rI1←rI1+SIZE(P)
67	3H	INC3	0,5	$a+c$	P←P+SIZE(P)
68	G9P	LDA	0,3(LINK)	$1+a+c$	
69		LD5	0,3(SIZE)	$1+a+c$	
70		JAZ	3B	$1+a+c$	如果LINK(P) = A则跳转
71		J5NZ	1B	$1+a$	跳转,除非SIZE(P) = 0

注意在行66中，我们假定每个节点的大小都充分小，使得它能以一单条MOVE指令来进行移动；对于大多数情况说来，当这种类型的废料收集可应用时，这似乎是一个公正的假定。

这个程序的总运行时间为$(44a+17b+2w+25c+8d+47)u$，其中a是可访问的节点数，b是其中链接场的个数，c是不以一不可访问节点领先的那些不可访问节点的个数，d是以一不可访问节点领先的那些不可访问节点的个数，而w是在可访问节点中的总字数。如果内存含有n个节点，其中有ρn个是不可访问的，则我们即可估计$a=(1-\rho)n$，$c=(1-\rho)\rho n$，$d=\rho^2 n$。例：5字的节点（平均地说），每个节点有两个链接场(平均地说)，以及1000个节点的内存。则当$\rho=1/5$时，发现每个可利用的节点花费374u；当$\rho=\frac{1}{2}$时，花费104u；而当$\rho=\frac{4}{5}$时，仅花33u。

36．一个顾客将可能在16个座席1，3，4，6，…，23中之一入座，如果进来一对顾客，则对他俩必须有位置，否则至少有两个人在座席(1，2，3)，至少两个在(4，5，6)，…，至少两个在(19，20，21)，以及至少一个在22或23，所以至少有15人已入座。

37. 头 16 个单人的进入者，她请他们入座。在已占用的席位之间有 17 个空席位的'间隙'——在每一端计一个间隙，在相邻的已占席位之间假定为长度 0 的一个间隙。空席位的总数，即是全部 17 个间隙之和，为 6。假设这些间隙的 x 个有奇数的长度；则 $6-x$ 个空间可利用来坐各对（注意 $6-x$ 为偶且$\geqslant 0$）。现在从左到右，顾客 1，3，5，7，9，11，13，15 中的每一个，其两边有着一个偶的间隙者，用完了他的饭并离席而去；每个奇的间隙至多防止这 8 个用餐者之一留下，因此至少 $8-x$ 人留下。可是仅仅还有 $6-x$ 个空位可供成对者入座。但是现在 $(8-x)/2$ 对成对者进入。

38. 这些论证易于推广；$N(n,2)=\lfloor(3n-1)/2\rfloor$ 对于 $n\geqslant 1$〔当女招待员使用头一个适合的策略以代替最优者时，罗布森已经证明，席位的必要和充分之个数为 $\lfloor(5n-2)/3\rfloor$〕。

39. 把内存分成大小为 $N(n_1, m)$，$N(n_2, n)$，$N(2m-2, m)$ 的三个独立的区域。为处理一个对于空间的要求，对该区域使用有关的最优策略，把每个区段放进头一个所述的容量不被超过的区域。这不能失败，因为如果我们对于 x 个单元的要求不能满足，则我们必然至少已经占有了 $(n_1-x+1)+(n_2-x+1)+(2m-x-1)>n_1+n_2-x$ 个单元。

现在如果 $f(n)=N(n,m)\div N(2m-2,m)$，则我们有半加性定律 $f(n_1+n_2)\leqslant f(n_1)+f(n_2)$。此后 $\lim f(n)/n$ 存在。〔证明：$f(a+bc)\leqslant f(a)+bf(c)$；因此 $\lim\sup_{n\to\infty} f(n)/n=\max_{0<a<c}\lim\sup_{b\to\infty} f(a+bc)/(a+bc)\leqslant f(c)/c$ 对于所有的 c；从此 $\lim\sup_{n\to\infty} f(n)/n\leqslant\lim\inf_{n\to\infty} f(n)/n$〕。所以 $\lim N(n,m)/n$ 存在。

〔由习题 38 我们得知 $N(2)=3/2$。对于任何 $m>2$，$N(m)$ 的值是未知的；不难证明，对于仅仅两个区段大小 1 和 b，乘法因子是 $2-1/b$；因此 $N(3)\geqslant 1\frac{2}{3}$。罗布森的方法意味着 $N(3)\leqslant 1\frac{11}{12}$，而且 $2\leqslant N(4)\leqslant 2\frac{1}{6}$〕。

40. 通过使用下列的策略，罗布森已经证明 $N(2^r)\leqslant 1+r$；分配给大小 k 的每个区段，其中 $2^m\leqslant k<2^{m+1}$，即头一个可利用的，在 2^m 的一个倍数开始的 k 个单元的区段。

当所有区段的大小都被限制在处于集合 $\{b_1, b_2, \cdots, b_n\}$ 之中时，命 $N(\{b_1, b_2, \cdots, b_n\})$ 表示乘法因子，所以 $N(n)=N(\{1, 2, \cdots, n\})$。罗布森和 S. 克罗达尔 (S. Krogdahl) 已经发现 $N(\{b_1, b_2, \cdots, b_n\})=n-(b_1/b_2+\cdots+b_{n-1}/b_n)$，每当对于 $1<i\leqslant n$，b_i 是 b_{i-1} 的一个倍数时；其实，罗布森已经建立了精确的公式 $N(2^r m, \{1, 2, 4, \cdots, 2^r\})=2^r m\left(1+\frac{1}{2}r\right)-2^r+1$。因此，特别地，$N(n)\geqslant 1+\frac{1}{2}\lfloor\lg n\rfloor$。他也推出了上限 $N(n)\leqslant 1.221\ln n+O(1)$，而且他试探推测 $N(n)=H_n$。一般地说，如果 $N(\{b_1, b_2, \cdots, b_n\})$ 等于 $n-(b_1/b_2+\cdots+b_{n-1}/b_n)$，则将得出这个推测，但遗憾的是并非这种情况，因为罗布森已经证明 $N(\{3, 4\})\geqslant 1\frac{4}{15}$。

41. 考虑维护大小 2^k 的区段；对于大小 1，2，4，$\cdots$，2^{k-1} 的要求，将周期地要求来分开大小 2^k 的一个新区段，或者将恢复该大小的一个区段。通过对 n 用归纳法我们可以证明，为被如此分开的诸区段所消耗的总的存储，决不超过 kn；因为在每次要求来分开大小 2^{k+1} 的一个区段之后，我们在已分开的 2^k 区段中至多正使用 kn 个单元，而且在未分开的区段中至多正使用 n 个单元。

这个论证可被强化以证明，$a_r n$ 个单元就足够了，其中 $a_0 = 1$ 和 $a_r = 1 + a_{r-1}(1 - 2^{-r})$；我们有

$k =$	0	1	2	3	4	5
$a_k =$	1	$1\frac{1}{2}$	$2\frac{1}{8}$	$2\frac{55}{64}$	$3\frac{697}{1021}$	$4\frac{19559}{32768}$

反之，对于 $r \leqslant 5$，可以证明，一个伙伴系统有时要求象 $a_r n$ 个之多的单元，如果把步骤 R1 和 R2 的机构加以修改，以选择最坏可能的可利用的 2^l 区段去分开，来代替头一个这样的区段。

罗布森关于 $N(2^r) \leqslant 1 + r$ 的证明（见习题 40），容易加以修改来证明，一个这样的"最左"的策略，将决不需要 $\left(1 + \frac{1}{2}r\right)n$ 个以上的单元来为大小 1，2，4，…，2^r 的区段分配空间，因为大小 2^k 的区段将决不被放置在 $\geqslant \left(1 + \frac{1}{2}k\right)n$ 的单元。尽管这个算法似乎非常象伙伴系统，但原来是，伙伴系统将没有这个好处，即使我们把步骤 R1 和 R2 加以修改以选择分开最好可能的可利用的 2^l 区段。例如，对于 16 和 $r = 3$，考虑下列内存的"抽点打印"序列：

11111111	11111111	00000000	00000000
10101010	10101010	2-2-2-2-	00000000
11110000	11110000	2-110000	00000000
11111111	11110000	11110000	00000000
10101010	10102-2-	10102-2-	00000000
10001000	10002-00	10002-00	4---4---
10000000	10000000	10000000	4---0000

这里 0 表示一个可利用的单元，而 k 表示一个 k 区段的开始。在一个类似的方式下，有一系列的操作，每当 n 是 16 的一个倍数时，它迫使大小 8 的 $\frac{3}{16}n$ 个区段成为 $\frac{1}{8}$ 满，而另外的 $\frac{1}{16}n$ 成为 $\frac{1}{2}$ 满。如果 n 是 128 的倍数，则大小 8 的 $\frac{9}{128}$ 个区段的一个随继的要求，将需要 2.5 n 个以上的内存单元（伙伴系统允许诸 1 溜进 8 区段中的 $\frac{3}{16}n$ 个，因为在一个决定性的时间里没有其它可利用的诸 2 来分开；"最左"的算法保持所有有限的诸 1）。

附 录 A

记 号 索 引

在下列公式中，未作进一步说明的字母有下列的意义：

j、k　整数值的算术表达式

m、n　非负整数值的算术表达式

x，y，z　实数值的算术表达式

f　实数值的函数

P　指针值的表达式，即是，或是A或是在一台计算机内的一个地址

S，T　集合或多重集合

α　符号串

形式符号	意　义	节次参考
NODE(P)	地址为P，P≠A，的节点(由它们的场名称独立地识别的变量的组)	2.1
F(P)	场名称是F的NODE(P)中的变量	2.1
CONTENTS(P)	地址是P的计算机“字”的内容	2.1
LOC(V)	在一台计算机内变量V的地址	2.1
A_n	线性数组A的第n个元素	
A_{mn}	矩阵数组A的行m列n中的元素	
$A[n]$	等价于A_n	1.1
$A[m,n]$	等价于A_{mn}	1.1
$V \leftarrow E$	把表达式E的值给变量V	1.1
$U \leftrightarrow V$	交换变量U和V的值	1.1
P←AVAIL	把指针变量P的值置为一个新节点的地址，或者如果没有一个新节点的余地了，则就给出内存溢出的信号	2.2.3
AVAIL←P	NODE(P)恢复成自由存储；所有它的场都失去它们的身份	2.2.3
top(S)	在一非空的堆栈S顶上的节点	2.2.1
$X \leftarrow$ S	从S弹出到X；置$X \leftarrow$top(S)；然后从非空堆栈S中删去top(S)	2.2.1
S $\leftarrow X$	把X压入S；插入由X表示的值或值组以作为堆栈S顶上的一个新的条款	2.2.1
$(B \Rightarrow E_1; E_2)$	条件表达式：表示$E1$如果B为真，表示$E2$如果B为假	8.1
δ_{jk}	克罗内克尔δ：($j = k \Rightarrow 1$；0)	1.2.3
$\sum_{R(k)} f(k)$	使得k为整数且关系$R(k)$为真的所有$f(k)$之和	1.2.3
$\prod_{R(k)} f(k)$	使得k为整数且关系$R(k)$为真的所有$f(k)$之乘积	1.2.3

续

形式符号	意义	节次参考
$\min_{R(k)} f(k)$	使得 k 为整数且关系 $R(k)$ 为真的所有 $f(k)$ 之极小值	1.2.3
$\max_{R(k)} f(x)$	使得 k 为整数且关系 $R(k)$ 为真的所有 $f(k)$ 之极大值	1.2.3
$j \backslash k$	j 整除 k：$k \bmod j = 0$	1.2.1
$S \setminus T$	集合差：$\{a \mid a$ 在 S 中，而 a 不在 T 中$\}$	
$\gcd(j,k)$	j 与 k 的最大公因子：$(j = k = 0 \Rightarrow 0;\ \max_{d \backslash j, d \backslash k} d)$	1.1
$\det(A)$	正方矩阵 A 的行列式	1.2.3
A^T	矩形数组 A 的转置：$A^T[j,k] = A[k,j]$	1.2.3
α^R	α 的左右反转	
x^y	x 的 y 次方，x 为正	1.2.2
x^k	x 的 k 次方：$\left(k \geqslant 0 \Rightarrow \prod_{0 \leqslant j < k} x;\ 1/x^{-k}\right)$	1.2.2
$x^{\overline{k}}$	x 上面 k：$\left(k \geqslant 0 \Rightarrow x(x+1)\cdots(x+k-1) = \prod_{0 \leqslant j < k} (x+j);\ 1/(x+k)^{\overline{-k}}\right)$	1.2.6
$x^{\underline{k}}$	x 下面 k：$(-1)^k (-x)^{\underline{k}} = \left(k \geqslant 0 \Rightarrow x(x-1)\cdots(x-k+1) = \prod_{0 \leqslant j < k} (x-j);\ 1/(x-k)^{\underline{-k}}\right)$	1.2.6
$n!$	n 的阶乘：$1 \cdot 2 \cdot \cdots \cdot n = n^{\underline{n}}$	1.2.5
$\binom{x}{k}$	二项式系数：$(k < 0 \Rightarrow 0;\ x^{\underline{k}} / k!)$	1.2.6
$\binom{n}{n_1, n_2, \cdots, n_m}$	多项式系数，$n = n_1 + n_2 + \cdots + n_m$	1.2.6
$\left[\begin{matrix} n \\ m \end{matrix}\right]$	第一类型的Stirling数：$\sum_{0 < k_1 < k_2 < \cdots < k_{n-m} < n} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\left\{\begin{matrix} n \\ m \end{matrix}\right\}$	第二类型的Stirling数：$\sum_{0 \leqslant k_1 \leqslant k_2 \leqslant \cdots \leqslant k_{n-m} \leqslant m} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\{a \mid R(a)\}$	使得关系 $R(a)$ 为真的所有 a 的集合	
$\langle a_1, \cdots, a_n \rangle$	集合或多重集合 $\{a_k \mid 1 \leqslant k \leqslant n\}$	
$\{x\}$	在所要求的是一实数值，而不是一个集合的上下文中，表示分数部分：$x \bmod 1$	1.2.11.2
$\lvert S \rvert$	基数：S 中元素之个数	
$\lvert x \rvert$	x 的绝对值：$(x < 0 \Rightarrow -x;\ x)$	

形式符号	意义	节次参考
$\lvert\alpha\rvert$	α 的长度	
$\lfloor x \rfloor$	x 的底限，最大整数函数：$\max_{k \leqslant x} k$	1.2.4
$\lceil x \rceil$	x 的顶限，最小整数函数：$\min_{k \geqslant x} k$	1.2.4
$x \bmod y$	mod函数：（$y = 0 \Rightarrow x$；$x - y \lfloor x / y \rfloor$）	1.2.4
$x \equiv y \pmod{z}$	同余关系：$x \bmod z = y \bmod z$	1.2.4
$\log_b x$	以 b 为底，x 的对数（实的正的 $b \neq 1$）：$x = b^{\log_b x}$	1.2.2
$\ln x$	自然对数：$\log_e x$	1.2.2
$\lg x$	x 的二进对数：$\log_2 x$	1.2.2
$\exp x$	x 的指数：e^x	1.2.2
$\langle X_n \rangle$	无限序列 $X_0, X_1, X_2, \cdots$（这里 n 是一个字母，它是符号的一部分）	1.2.9
$f'(x)$	f 在 x 处的导数	1.2.9
$f''(x)$	f 在 x 处的二次导数	1.2.10
$f^{(n)}(x)$	第 n 次导数：（$n = 0 \Rightarrow f(x)$；$g'(x)$ 其中 $g(x) = f^{(n-1)}(x)$）	1.2.11.2
$H_n^{(x)}$	$1 + 1/2^x + \cdots + 1/n^x = \sum_{1 \leqslant k \leqslant n} 1/k^x$	1.2.7
H_n	调和数：$H_n^{(1)}$	1.2.7
F_n	斐波那契数：（$n \leqslant 1 \Rightarrow n$；$F_{n-1} + F_{n-2}$）	1.2.8
B_n	贝努利数	1.2.11.2
$B(x, y)$	贝塔函数	1.2.6
$\operatorname{sing}(x)$	x 的符号：（$x = 0 \Rightarrow 0$；（$x > 0 \Rightarrow +1$；-1））	
$\zeta(x)$	黎特函数：$H_\infty^{(x)}$ 当 $x > 1$ 时	1.2.7
$\Gamma(x)$	伽马函数：$\gamma(x, \infty)$；$(x-1)!$ 当 x 为一正整数时	1.2.5
$\gamma(x, y)$	不完全的伽马函数	1.2.11.3
γ	欧拉常数	1.2.7
e	自然对数的底：$\sum_{k \geqslant 0} 1/k!$	1.2.2
∞	无穷：大于任何数	
Λ	空的链接（指针不指向地址）	2.1
ϵ	空串（长度为 0 的串）	
$\emptyset$	空的集合（无元素的集合）	
ϕ	黄金比，$\frac{1}{2}(1 + \sqrt{5})$	1.2.8
$\varphi(n)$	欧拉的 φ 函数：$\sum_{\substack{0 \leqslant k < n \\ \gcd(k, n) = 1}} 1$	1.2.4
$p(n)$	n 的分划数	1.2.1
$x \sim y$	x 近似地等于 y	1.2.5
$O(f(n))$	当 $n \to \infty$ 时 $f(n)$ 的大 O	1.2.11.1
$O(f(x))$	对于小 x（或者对于在某个确定范围内的 x），$f(x)$ 的大 O	1.2.11.1

续

形式符号	意义	节次参考
$(\min x_1, \text{ave}x_2, \max x_3, \text{dev}x_4)$	一个随机变量，有极小值x_1，平均（"预期的"）值x_2，极大值x_3，标准偏差x_4	1.2.10
mean(g)	由生成函数g所表示的概率分布的平均值	1.2.10
var(g)	由生成函数g所表示的概率分布的方差：$g''(1)+g'(1)-g'(1)^2$	1.2.10
P*	在一二叉树中NODE(P)的先根次序的后继者之地址	2.3.1
P⊕	在一二叉树中NODE(P)的中根次序的后继者之地址	2.3.1
P#	在一二叉树中NODE(P)的后根次序的后继者之地址	2.3.1
*P	在一二叉树中NODE(P)的先根次序的先驱之地址	2.3.1
⊕P	在一二叉树中NODE(P)的中根次序的先驱之地址	2.3.1
#P	在一二叉树中NODE(P)的后根次序的先驱之地址	2.3.1
■	算法，程序，或证明的结束	1.1
␣	一个空格	1.3.1
rA	MIX的寄存器（累加器）A	1.3.1
rX	MIX的寄存器（扩充）X	1.3.1
r1,…,r16	MIX的（变址）寄存器I1,…,I6	1.3.1
rJ	MIX的（跳转）寄存器J	1.3.1
(L:R)	MIX字的部分场，$0 \leqslant L \leqslant R \leqslant 5$	1.3.1
OP ADDRESS,I(F)	MIX指令的记号	1.3.1, 1.3.2
u	MIX中的时间单位	1.3.1
*	MIXAL中的"自身"	1.3.2
0F,1F,2F,…,9F	在MIXAL中"向前"的局部符号	1.3.2
0B,1B,2B,…,9B	在MIXAL中"向后"的局部符号	1.3.2
0H,1H,2H,…,9H	在MIXAL中"这里"的局部符号	1.3.2

附 录 B

数值数量表格

表 1

在标准的子程序中和在计算机程序的分析中，经常使用的数量（到40位小数）。

$\sqrt{2}$ = 1.41421 25623 73095 04880 16887 24209 69807 85697 -

$\sqrt{3}$ = 1.73205 08075 68877 29352 74463 41505 87236 69428 +

$\sqrt{5}$ = 2.23606 79774 99789 69640 91736 68731 27623 54406 +

$\sqrt{10}$ = 3.16227 76601 68379 33199 88935 44432 71853 37196 -

$\sqrt[3]{2}$ = 1.25992 10498 94873 16476 72106 07278 22835 05703 -

$\sqrt[3]{3}$ = 1.44224 95703 07408 38232 16383 10780 10958 83919 +

$\sqrt[3]{2}$ = 1.18920 71150 02721 06671 74999 70560 47591 52930 -

ln 2 = 0.69314 71805 59945 30941 72321 21458 17656 80755 -

ln 3 = 1.09861 22886 68109 69139 52452 36922 52570 46475 -

ln10 = 2.30258 50929 94045 68401 79914 54684 36420 76011 +

1/ln2 = 1.44269 50408 88963 40735 99246 81001 89213 74266 +

1/ln10 = 0.43429 44819 03251 82765 11289 18916 60508 22944 -

π = 3.14159 26535 89793 23846 26433 83279 50288 41972 -

$1^\circ = \pi/180$ = 0.01745 32925 19943 29576 92369 07684 88612 71344 +

$1/\pi$ = 0.31830 98861 83790 67153 77675 26745 02872 40689 +

π^2 = 9.86960 44010 89358 61883 44909 99876 15113 53137 -

$\sqrt{\pi} = \Gamma(1/2)$ = 1.77245 38509 05516 02729 81674 83341 14518 27975 +

$\Gamma(1/3)$ = 2.67893 85347 07747 63365 56929 40974 67761 41287 -

$\Gamma(2/3)$ = 1.35411 79391 26400 41694 52880 28154 51378 55193 -

e = 2.71828 18284 59045 23536 02874 71352 66249 77572 +

$1/e$ = 0.36787 94411 71442 32159 55237 70161 46086 74458 +

e^2 = 7.38905 60989 30650 22723 04274 60575 00781 31803 +

γ = 0.57721 56649 01532 86060 65120 90082 40243 10422 -

ln π = 1.14472 98858 49400 17414 34273 51353 05871 16473 -

ϕ = 1.61803 39887 49894 84820 45868 34365 63811 77203 +

e^γ = 1.78107 24179 90197 98523 65041 03107 17954 91696 +

$e^{\pi/4}$ = 2.19328 00507 38015 45655 97696 59278 73822 34616 +

sin 1 = 0.84147 09848 07896 50665 25023 21630 29899 96226 -

cos 1 = 0.54030 23058 68139 71740 09366 07442 97660 37323 -

$\zeta(3)$ = 1.20205 69031 59594 28539 97381 61511 44999 07650 -

ln ϕ = 0.48121 18250 59603 44749 77589 13424 36842 31352 -

1/ln ϕ = 2.07808 69212 35027 53760 13226 06117 79576 77422 -

- ln ln 2 = 0.36651 29205 81664 32701 24391 58232 66916 94543 -

表 2

在标准的子程序中和在计算机程序的分析中经常用到八进记数。出现于等号右边的每个量的名称，以十进记数给出。

0.1 =	0.06314 63146 31463 14631 46314 63146 31463 14631 1632
0.01 =	0.00507 53412 17270 24365 60507 53412 17270 24365 6051
0.001 =	0.00040 61115 64570 65176 76355 44264 16254 02030 4467
0.0001 =	0.00003 21556 13530 70414 54512 75170 33021 15002 3522
0.00001 =	0.00000 24761 32610 70664 36041 06077 17401 56063 3442
0.000001 =	0.00000 02061 57364 05536 66151 55323 07746 44470 2603
0.0000001 =	0.00000 00153 27745 15274 53644 12741 72312 20354 0215
0.00000001 =	0.00000 00012 57143 56106 04303 47374 77341 01512 6333
0.000000001 =	0.00000 00001 04560 27640 46655 12262 71426 40124 2174
0.0000000001 =	0.00000 00000 06676 33766 35367 55653 37265 34642 0163
$\sqrt{2}$ =	1.32404 74631 77167 46220 42627 66135 46725 12575 1744
$\sqrt{3}$ =	1.56663 65641 30231 25163 54453 50265 60361 34073 4222
$\sqrt{5}$ =	2.17067 36334 57722 47602 57471 63003 00563 55620 3202
$\sqrt{10}$ =	3.12305 40726 64555 22444 02242 57101 41466 33775 2253
$\sqrt[3]{2}$ =	1.20505 05746 15345 05342 10756 65334 25574 22415 0303
$\sqrt[3]{3}$ =	1.34233 50444 22175 73134 67363 76133 05334 31147 6012
$\sqrt[3]{10}$ =	1.14067 74050 61556 12455 72152 64430 60271 02755 7314
ln 2 =	0.54271 02775 75071 73632 57117 07316 30007 71366 5364
ln 3 =	1.06237 24752 55006 05227 32440 63065 25012 35574 5534
ln10 =	2.23273 06735 52524 25405 56512 66542 56026 46050 5071
1/ln 2 =	1.34252 16624 53405 77027 35750 37766 40644 35175 0435
1/ln10 =	0.33626 75425 11562 41614 52325 33525 27655 14756 0622
π =	3.11037 55242 10264 30215 14230 63050 56006 70163 2112
$1^\circ = \pi/180$ =	0.01073 72152 11224 72344 25603 54275 63351 22056 1155
$1/\pi$ =	0.24276 30155 62344 20251 23760 47257 50765 15156 7007
π^2 =	11.67517 14467 62153 71322 25561 15466 30021 40654 3410
$\sqrt{\pi} \cdot \Gamma(1/2)$ =	1.61337 61106 64736 65247 47035 40510 15273 34470 1776
$\Gamma(1/3)$ =	2.53347 35234 51013 61316 73106 47644 54653 00106 6605
$\Gamma(2/3)$ =	1.26523 57112 14154 74312 54572 37655 60126 23231 0245
e =	2.55760 52130 50535 51246 52773 42542 00171 72363 0166
$1/e$ =	0.27426 53066 13167 46761 52726 75436 02440 52371 0336
e^2 =	7.30714 45615 23355 33460 63507 35040 32664 25356 5022
γ =	0.44742 14770 67666 06172 23215 74376 01002 51313 2552
ln π =	1.14206 40443 47503 36413 65374 52661 52410 37511 4606
ϕ =	1.47433 57156 27751 23701 27634 71401 10271 66710 1501
e^γ =	1.61772 13452 61152 65761 22477 36553 53327 17554 2126
$e^{\pi/4}$ =	2.14275 31512 16162 52370 35530 11342 53523 44307 0217
sin 1 =	0.65665 24436 04414 73402 03067 23614 11612 07474 1451
cos 1 =	0.42450 50037 32406 42711 07022 14666 27320 70675 1232
$\zeta(3)$ =	1.14735 00023 60014 20470 15613 42561 31715 10177 0662
ln ϕ =	0.36630 26256 61213 01145 13700 41004 52264 30700 4065
1/ln ϕ =	2.04776 60111 17144 41512 11436 16575 00355 43630 4065
$-\ln\ln 2$ =	0.27351 71233 67265 63650 17401 56637 26334 31455 5701

表 1 和表 2 包含了若干至今未发表过的由小约翰·W·伦奇 (John W. Wrench, Jr) 用一台式计算器计算出来的 40 位数字的值。

对于在这个表中找不到的常数的高精度的值，请看 J·彼得斯 (J. Peters)，《从 1

到100000的数的十位对数》(Ten Place Logarithms of the Numbers from 1 to 100000)，第1卷的附录（纽约：F·昂加尔图书公司，1957)；以及由米·阿布拉莫维茨和艾·安·斯特岗编的《数学函数手册》(Handbook of Mathematical Functions)，(美国首都华盛顿：美国政府出版局，1964)，第一章。

关于另一个基本常数的40位值，见习题1.3.3-23的答案。

表 3

调和数，贝努利数，和斐波那契数的值，对于小的一些 n 的值的。

n	H_n	B_n	F_n	n
0	0	1	0	0
1	1	−1/2	1	1
2	3/2	1/6	1	2
3	11/6	0	2	3
4	25/12	−1/30	3	4
5	137/60	0	5	5
6	49/20	1/42	8	6
7	363/140	0	13	7
8	761/280	−1/30	21	8
9	7129/2520	0	34	9
10	7381/2520	5/66	55	10
11	83711/27720	0	89	11
12	86021/27720	−691/2730	144	12
13	1145993/360360	0	233	13
14	1171733/360360	7/6	377	14
15	1195757/360360	0	610	15
16	2436559/720720	3617/510	987	16
17	42142223/12252240	0	1597	17
18	14274301/4084080	43867/798	2584	18
19	275295799/77597520	0	4181	19
20	55835135/15519504	−174611/330	6765	20
21	18858053/5173168	0	10946	21
22	19093197/5173168	854513/138	17711	22
23	444316699/118982864	0	28657	23
24	1347822955/356948592	−236364091/2730	46368	24
25	34052522467/8923714800	0	75025	25

对于任何 x，命 $H_x=\sum_{n\geq 1}\left(\frac{1}{n}-\frac{1}{n+x}\right)$。于是

$$H_{1/2}=2-2\ln 2,$$

$$H_{1/3}=3-\frac{1}{2}\pi\Big/\sqrt{3}-\frac{3}{2}\ln 3,$$

$$H_{2/3}=\frac{3}{2}+\frac{1}{2}\pi\Big/\sqrt{3}-\frac{3}{2}\ln 3,$$

$$H_{1/4}=4-\frac{1}{2}\pi-3\ln 2,$$

$$H_{3/4}=\frac{4}{3}+\frac{1}{2}\pi-3\ln 2,$$

$$H_{1/5}=5-\frac{1}{2}\pi\phi\sqrt{\frac{2+\phi}{5}}-\frac{1}{2}(3-\phi)\ln 5-\left(\phi-\frac{1}{2}\right)\ln(2+\phi),$$

$$H_{2/5}=\frac{5}{2}-\frac{1}{2}\pi/\phi\sqrt{2+\phi}-\frac{1}{2}(2+\phi)\ln 5+\left(\phi-\frac{1}{2}\right)\ln(2+\phi),$$

$$H_{3/5}=\frac{5}{3}+\frac{1}{2}\pi/\phi\sqrt{2+\phi}-\frac{1}{2}(2+\phi)\ln 5+\left(\phi-\frac{1}{2}\right)\ln(2+\phi),$$

$$H_{4/5}=\frac{5}{4}+\frac{1}{2}\pi\phi\sqrt{\frac{2+\phi}{5}}-\frac{1}{2}(3-\phi)\ln 5-\left(\phi-\frac{1}{2}\right)\ln(2+\phi),$$

$$H_{1/6}=6-\frac{1}{2}\pi\sqrt{3}-2\ln 2-\frac{3}{2}\ln 3,$$

$$H_{5/6}=\frac{6}{5}+\frac{1}{2}\pi\sqrt{3}-2\ln 2-\frac{3}{2}\ln 3,$$

而且，一般地，当 $0<p<q$ 时（参照 1.2.9-19），

$$H_{p/q}=\frac{q}{p}-\frac{1}{2}\pi\cot\frac{p}{q}\pi-\ln 2q+2\sum_{1\leqslant n<q/2}\cos\frac{2\pi np}{q}\ln\sin\frac{n}{q}\pi。$$

名词和姓名中英对照表

A register of MIX — MIX 的 A 寄存器
A-1 compiler — A-1 编译程序
Aardenne-Ehrenfest, Tatiana van — 阿登尼-埃伦费斯特，塔尼亚纳·范
Aarons, Roger M. — 艾伦斯，罗杰 M
Abel, Niels Henrik — 阿贝尔，尼亚斯·亨里克
  binomial formula generalized — 推广的二项公式
  limit theorem — 极限定理
Abramowitz, Milton — 阿布拉莫维茨，米尔顿
ACE computer, Pilot — 试验性的（飞行者）ACE 计算机
Adams, Charles William — 亚当斯，查尔斯·威廉
ADD — 加指令
Add to list; see Insertion — 加到表中，见插入
Addition of polynomials — 多项式加法
Address; A number used to identify a position in memory — 地址：用来标识存储器的位置的一个数
  field of MIXAL line — MIXAL 语句的地址场
  of node — 节点地址
  portion of MIX instruction — MIX指令的地址部分
Address transfer operator of MIX — MIX的地址传送操作符
Adjacent vertices of a graph — 图画的相邻顶点
Agenda, see Priority queue — 议事日程，见优先排队
Ahrens, Wilhelm Ernst Martin Georg — 阿伦斯，威廉·厄恩斯特·马丁·乔
al-Khwārizmī, Abu Ja 'far Mohammed ibn Mûsā — 阿科瓦里茨米，阿布·贾法·穆哈默德·伊本·摩萨
Alanen, Jack David, — 艾伦宁，杰克·戴维
ALF (alphabetic data) — ALF（字母的数据）
Algebraic formulas, manipulation of — 代数公式的处理
  differentiation — 微商的处理
  representation as trees — 表示为树形的处理
  simplification — 简化的处理
Algorithm, origin of word — 算法，词的起源
Algorithms — 算法
  analysis of, — 算法的分析
  communication of — 算法的通讯
  effective — 能行的算法
  equivalence between — 算法间的等价性
  form of in this book — 算法在本书中的形式
  hardware-oriented — 面向硬件的算法
  how to read — 如何读算法
  proof of — 算法的证明
  properties of — 算法的性质
  random paths in — 算法中的随机通路
  set theoretical definition — 算法的集合论定义
  theory of — 算法论
Along order — 进行的次序
Alphametric character; A letter, digit, or special character symbol — 字母数字字符：一个字母，数字，或特殊字符

codes for MIX — MIX 的字母数字代码
AMM, *American Mathematical Monthly*, the official journal of the Mathematical Association of America, Inc — AMM，美国数学月刊，美国数学协会机关刊物，综合性的
Analysis of algorithms — 算法的分析
Analytical Engine — 分析机器
Ancestor, in a tree structure — 在一树结构中的祖先
André, Antoine Désiré — 安德烈，安托万·德西利
Anticipated input, see Buffering — 抢先输入，见缓冲
Antisymmetric relation — 反对称关系
Apex of tree — 树形的顶点
Apostol, Tom Mike — 阿波斯托尔，汤姆·迈克
Arborescence, see Oriented trees — 分叉，见有向树
Arc in a directed graph — 一有向树中的弧
Arc-digraph — 弧有向树
Area of memory — 存储区域
Arguments of subroutines — 子程序的自变量
Arithmetic, Addition, subtraction, multiplication, and division — 算术运算：加、减、乘、除
  fixed point — 定点算术运算
  floating-point — 浮点算术运算
  operators of MIX — MIX 的操作符
  polynomial — 多项式算术运算
  scaled decimal — 带比例的十进算术运算
Arithmetic expressions, see Algebraic formulas — 算术表达式，见代数公式
Arithmetic progression, sum of — 算术级数的和
Array, A table which usually has a k-dimensional rectangular structure — 数组，通常有一k维矩阵结构的表
  one dimensional, see Linear list — 一维数组，见线性表
  sequential allocation — 顺序分配
  represented as trees — 数组表示成树
  tetrahedral, see Binomial number system — 四面体数组，见二项式数系
  two dimensional, see Matrix — 二维数组，见矩阵
Arrows, used to represent links in diagrams — 箭头，用来表示图式中的连接
Assembly language, A language which is intended to facilitate the construction of programs in machine language by making use of symbolic and mnemonic conventions to denote machine language instructions. — 汇编语言，利用符号和记忆名约定来表示机器语言指令，在机器语言中以图简化程序构造的一种语言
  for MIX — MIX 的汇编语言
Assembly program — 汇编程序
ASSIGN a buffer — 指定一个缓冲区
Assignment operation — 赋值运算
Asterik ("*") in assembly language — 在一汇编语言中的星号
Asymmetric relations — 反对称关系
Asymptotic values, Functions which express the limiting behavior approached by numerical quantities — 近似值：表示数值量所趋向的极限行为的函数
Atom (in a List) — (在一个列表中的) 原子
Automata theory — 自动机理论
Automaton, An abstract machine which is formally defined in some manner, often intended to be a model of some aspects of actual computers (plural: — 自动机：以某种方式形式地定义的抽象机器，它经常试图用作实际计算机某一方面的模型（这个词的复数形式是Automata）

Automata)
AVAIL stack; Available space list — AVAIL 堆栈；可利用的空间表
Available space list — 可利用的空间表
  history — 可利用的空间表的历史
  variable size blocks — 可变大小的区段
Average value of a probability distribution — 一个概率分布的平均值
Babbage, Charles — 巴贝奇，查理斯
Bachmann, Paul Gustav Heinrich — 巴克曼，保罗·古斯塔夫·亨里奇
Backus, John Warner — 巴克斯，约翰·沃纳
Bailey, Michael John — 贝利，迈克尔·约翰
Bailey, Wilfrid Norman — 贝利，威尔弗雷德·诺曼
Balanced directed graph — 平衡的有向图通
Ball, Walter William Rouse — 鲍尔，沃尔特·威廉·劳斯
Ballot problem — 抽签问题
Barnett, Michael Peter — 巴尼特，迈克尔·彼得
Base address — 基地址
Bead, see Node — 珠子，见节点
Before and after diagram — 之前和之后的图式
Bell, Eric Temple — 贝尔，埃里克·坦普尔
Bellman, Richard Ernest — 贝尔曼，理查德·欧内斯特
Bennett, John Makepeace — 贝内特，约翰·梅克皮斯
Berge, Claude — 伯奇，克劳德
Berger, Robert — 伯杰，罗伯特
Bergman, George Mark — 伯格曼，乔治·马克
Berman, Martin Fredric — 伯曼，马丁·弗雷德里克
Bernoulli, James (= Jakob = Jacques) — 贝努利，詹姆斯（=雅各布 =雅克）
  numbers — 贝努利数
  numbers, table — 贝努利数表
  polynomials — 贝努利多项式
Bertrand, Joseph Louis Francois — 伯特兰，约瑟夫·路易斯·弗朗索斯
  postulate — 伯特兰假设
Berztiss, Alfs Teodors — 伯齐提斯，阿尔夫斯·杰阿多尔斯
Best-fit method of storage allocation — 存储分配的"最适合"的方法
Beta function — 贝塔函数
Bhāskara Acharya — 巴斯加拉·阿凯里雅
"Big-oh" notation — 大○记号
Bigelow, Richard Henry — 比奇洛，理查德·亨利
Binary computer; A computer which manipulates numbers primarily in the binary (radix 2) number system — 二进计算机；主要以二进数系进行运算（进制为2）的一台计算机
Binary logarithm — 二叉树
Binary trees — 二叉树
  complete — 完备的二叉树
  copying of — 二叉树的复写
  correspondence to trees and forest — 二叉树同树和森林的对应
  definition of — 二叉树的定义
  "Dewey" notation for — 二叉树的杜威记号
  enumeration of — 二叉树的枚举
  equivalent — 二叉树的等价
  erasing of — 二叉树的抹去
  extended — 扩充的二叉树
  oriented — 有向的二叉树
  path length of — 二叉树的通路长度

right-threaded
representation of
similar
threaded
traversal of
Binet, Jacques Phillipe Marie
Binomial coefficients
combinatorial interpretation
defined
generalized
generating functions
history
sums involving
table of
Binomial distribution
Binomial number system
Binomial theorem
Abel's generalization
generalization of
Hurwitz's generalization
Bit, "Binary digit", either zero or unity
BIT, Nordisk Tidskrift for Informations-behandling, a journal published by Regnecentralen, Copenhagen, Denmark
Blaauw, Gerrit Anne
Blikle, Andrzej Jacek
Block of memory
Blocking of record
Bobrow, Daniel Gureasko
Bolzano, Bernhard, theorem
Boncompagni, Prince Baldassarre
Bocthroyd, John
Borchardt, Carl Wilhelm
Bottom of stack
Bottom-up process
Boundary tag method of storage allocation
Bourne, Charles Percy
Branch instruction, A conditional "jump" instruction
Branch node of tree
Brother, in a tree structure
BROTHER link in tree
Brouwer, Luitzen Egbertus Jan
Brute force
Buddy system for storage allocation
Buffering of input-output
history
swapping
Buniakovskii, Viktor Yakovlevich
Burke, John, Peerage
Burks, Arthur Walter
Burleson, Peter Barrus
Burroughs B220
Burroughs B5000-B5500

右穿线的二叉树
二叉树的表示
相似的二叉树
穿线的
二叉树的遍历
比尼特，雅克·菲利普·玛丽
二项式系数
二项式系数的组合解释
确定的二项式系数
推广的二项式系数
二项式系数的生成函数
二项式系数的历史
有关二项式系数的和
二项式系数表
二项分布
二项数系
二项式定理
二项式定理的阿贝尔推广
二项式定理的推广
二项式定理的赫维茨推广
比特：二进数字，0或1
BIT：丹麦哥本哈根计算中心出版的杂志
布劳维，格里特·安妮
布利格尔，安德烈·杰西克
存储区的区段
记录的分块
博布罗，丹尼尔·格里斯戈
布尔查诺，伯恩哈特定理
邦康帕尼，巴尔萨斯·波达萨利
布思罗伊德，约翰
博查德，卡尔·威廉
堆栈的底下
由底向上的过程
存储分配的边界标志方法
伯恩，查尔斯·珀西
转移指令：一个有条件的"跳转"指令
树的分枝节点
一树结构中的兄弟
树中的 BROTHER（兄弟）链
布劳威尔，卢特曾·埃格伯杜斯·简
暴力，蛮干，指使算
存储分配的伙伴系统
输入-输出的缓冲技术
输入-输出缓冲的历史
转储
布尼雅柯夫斯基，维克托·亚柯夫列维奇
伯克，约翰，皮拉吉
伯克斯，阿塞·沃尔特
伯利森，彼得·巴拉斯
巴尔斯 B220（计算机）
巴尔斯 B5000-B5500（计算机）

Byte: Basic unit of data, usually associated with alphameric characters in MIX
Byte size in MIX: The number of distinct values that might be stored in a byte
CACM: Communications of the ACM, a publication of the Association for Computing Machinery
Cajori, Florian
Calendar
California Institute Technology (Caltech)
Call: To activate another routine in a program
Call sequence
Canonical cycle notation for permutation
Canonical representation of oriented trees
Car: LISP terminology for the first component of a List; analogous to INFO and DLINK or to ALINK
Card format for MIXAL programs
Cards, playing
Carlitz, Leonard
Carlyle, Thomas
Carr, John Weber, Ⅲ
Cassini, Jean Dominique
Catalan, Eugène Charles
  numbers
Cauchy, Augustin Louis
Cayley, Arthur
CDC G20
CDC 1604
Cdr: LISP terminology for the remainder of a List with its first component deleted; analogous to RLINK or to BLINK
Ceiling function
Cell: A word of the computer memory
Cellar
Centroid of a free tree
Chain: A word used by some authors to denote a linked linear list
Chain rule for diffrentiation see Faà di Bruno's formula
Chainning: A word used by some authors in place of "linking"
Channel: A data-transmission device connected to a computer
CHAR (convert to characters)
Character code of MIX
Characteristic function of a probability distribution
Chebyshev, Pafnutii L'vovich
  inequality
  polynomials
Checkerboard
Checkerboarding, see Fragmentation
Cheney, C. J.
Chess

字节：数据的基本单位，通常与字母数字字符相关联
MIX 中的字节
MIX 中的字节大小在一个字节中可以存储的不同值的个数
CACM ACM 通讯，（美国）计算机协会的刊物
卡乔里，弗洛里安
日历
加里弗尼亚理工学院（简称 Caltech）
调用：激活一程序中的另一个子程序
调用序列
排列的典型循环记法
有向树形的典型表示
Car：LISP 语言的术语，用于表示一个列表的头一个分量，类似于 INFO 和 DLINK 或 ALINK

MIXAL 程序的卡片格式
扑克牌
卡里兹，伦纳德
卡莱尔，托马斯
卡尔，约翰·韦伯Ⅲ
卡西尼，琼·多米尼克
卡塔兰，尤金·查尔斯
  卡塔兰数
柯西，奥古斯丁·路易斯
凯利，阿瑟
CDC G20（计算机）
CDC 1604计算机
Cdr：LISP 语言的术语，用来表示删去一个列表的头一个分量之后的剩下部分；类似于 RLINK 或 BLINK
"天棚"函数
单元：计算机存储器的一个字
地窖
一自由树的形心
链：某些作者用来表示一个链接的线性表列的词
微分的链接规则，见佛故奋公式

链接：为某些作者用来代替"链接"的另一名词的词
通道：连接到一台计算机的一数据传输设备

CHAR（转换成字符）的程序
MIX 的字符代码
概率分布的特征函数
契贝雪夫，帕芙奴蒂·勒沃维奇
  契贝雪夫不等式
  契贝雪夫多项式
跳棋盘
跳棋盘化，见分段
切尼，C·J.
国际象棋

Chung, Kai Lai 钟开来
CI: The comparison indicator of MIX CI: MIX 的比较指示器
Circle of buffers 缓冲区的回圈
Circuit, Eulerian, in a directed graph 在一个有向图画中的欧拉回路
Circuit, Hamiltonian, in a directed graph 在一个有向图画中的哈密顿回路
Circular definition, see Definition, circular 循环的定义，见下边 D 字首
Circular linkage 循环链接
Circular list 循环的表
Circular storage 循环存储
Circulating shift 循环移位
CITRUS 子程序名
Clavius, Christopher, S. J. 克拉维尤斯，克里斯托弗，S.J.
Clock, real time 实时钟
Clock, simulated 模拟钟
Clock, solitaired game 单人游戏钟
Closed subroutine, see Subroutine 闭子程序，见子程序
CMPA (compare A) CMPA 指令（比较 A）
CMPX (compare X) CMPX 指令（比较 X）
CMP1 (compare 1) CMP1 指令（比较 1）
COBOL: "Common Business-Oriented Language" COBOL: "公共的面向商务的语言"
Coding: Synonym for "programming", but with even less prestige Associated 编码：是程序设计的同义词，但有时声势略小些
Cofactor of element in square matrix: Determinant of the matrix obtained by replacing this element by unity and replacing all other elements having the same row or column by zero 正方矩阵元素的余因式：通过以 1 代替此元素，并以 0 代替和它同行或同列元素而得到矩阵的行列式
Cohen, Jacques 科恩，雅克
Coin tossing 投掷硬币
Combinations of n objects taken k at a time n 个事物一次取 k 个的组合
with repetitions permitted 允许有重复的 n 个事物一次取 k 个的组合
with restricted repetitions 有有限制重复的 n 个事物一次取 k 个的组合
Combinatorial matrix 组合矩阵
Comfort Webb T. 康福特，韦布·T
COMIT COMIT 语言
Command: Synonym for "instruction" 命令：指令的同义词
Comment in assembly language 汇编语言中的注释
Comp. J.: The Computer Journal, published by the British Computer Society Comp. J.: 计算机杂志，由英国计算机协会出版
Compacting memory 紧凑内存
Comparison indicator of MIX MIX 的比较指示器
Comparison operators of MIX MIX 的比较操作符
Compiler: Program which translates programming languages 编译程序：翻译程序设计语言的程序
Complete binary tree 完备的二叉树
Compound interest 复利
Computational method 计算方法
Compute: To process data 计算：处理数据
Computer: A data processor 计算机：一台数据处理器
Computer language, see Assembly language, Machine language, Programming language 计算机语言，见汇编语言，机器语言，程序设计语言
CON (constant) CON（常数）
Concatenation of strings 字符串的连接
Conditional expression 条件表达式

Congruence
Connected directed graph
  strongly
Connected graph
Conservative law, see Kirchhoff's law
Constants in assembly language
Construction of trees
CONTENTS
Continuous simulation
Convergence: An infinite sequence $\langle X_n \rangle$ converges if it approaches a limit as n approaches infinity; an infinite sum or product is said to "converge" or to "exist" if it has a value according to the conventions of mathematical calculus; see Eq. 1.2.3-3 and exercise 1.2.3-21
  of power series
Conversion operators of MIX
Convolution of probability distributions: The distribution obtained by adding two independent variables
Conway, Melvin Edward
Copy a data structure: To duplicate a structured object by producing another distinct object having the same data values and structural relationships
  binary tree
  linear list
  List
  tree
  two-dimensional linked list
Coroutine
  history
  linkage
Correspondence between binary trees and forests
Cousins
Coxeter, Harold Scott Macdonald
Critical path time
Crossword puzzle
Cumulants of probability distribution
Cycle: Path from vertex to itself
  detection of
  fundamental
  in directed graph
  in graph
  in permutation
  in random permutation
  notation for permutations
  oriented, in directed graph
  singleton
Dahl, Ole-Johan
Dahm, David Michael
Data (originally plural of the word "datum", but now used as singular or plural): Representation in a precise, formalized language of some facts or con-

同余
连通的有向图画
  强连通的有向图画
连通的图画
守恒定律，见克希霍夫定律
汇编语言中的常数
树的构造
CONTENTS 字段（场），即"内容"（场）字段
连续的模拟
收敛性：一无穷序列 $\langle X_n \rangle$ 收敛，如果当 n 趋于无穷时，它趋于一极限；一无穷和或乘积说是收敛或存在，如果按照微积分的约定，它有值；见等式 1.2.3-3 和习题 1.2.3-21
  幂级数的收敛性
MIX 的转换操作符
概率分布的卷积：通过把两个独立变量相加得到的分布

康伟，梅尔文·爱德华
一个数据结构的复写，通过产生具有相同数据值和结构关系的另一个不同对象以重复一个有结构的对象
  复写二叉树
  复写线性表列
  复写列表
  复写树
  复写二维链接的表列
并行程序
  并行程序的历史
  并行程序的链接
二叉树和森林之间的对应
堂兄弟
考克斯特，哈罗德·斯科特·麦克唐纳
临界通路时间
一种纵横填字字谜
概率分布的累加
回路：从顶点到它本身的通路
  回路的探测
  基本回路
  在有向图形中的回路
  图形中的回路
  排列中的回路
  在随机排列中的回路
  排列的回路记号
  在有向图形中分有向回路
  单一回路
达尔，奥利·约喻
达姆，戴维·迈克尔
数据（原来是 datum 一词的复数，但现在用作复数或单数）：在某些事实或概念的精确的形式化的语言中的表示，通

cepts, often numeric or alphabetic values, in a manner which can be manipulated by a computational method
packed
Data structure: A table of data including structural relationships
linear list structure
List structure
multilinked structures
orthogonal lists
tree structure
Daughter, see son
David, Florence Nightingale
Davis, Martin
de Bruijn, Nicolaas Govert
de La Loubère, Simon
de Moivre, Abraham
De Morgan, Augustus
Debugging: Detecting and removing bugs (error)
DECA (decrease A)
Decimal computer: A computer which manipulates numbers primarily in the decimal (radix ten) number system
DECX (decrease X)
DECI (decrease 1)
Defined symbol, in assembly language
Definition, circular, see Circular definition
Degree, of node in tree
of vertex in directed graph
Deletion of node: Removing it from a data structure and possibly returning it to available storage
from available space list, see Reservation
from deque
from doubly linked list
from linear list
from queue
from stack
from tree
from two-dimensional list
Demuth, Howard B
Deque: Double ended queue
deletion from
input-restricted
insertion into
linked allocation
output-restricted
sequential allocation
Derivative of a formula
Descendant in a tree structure
Determinant of a square matrix
Deuel, Phillip DeVere, Jr
Deutsch, Laurence Peter
Dewey, Melvil notation for binary

常是可由一计算方法可加处理的形式下的数值或字符值

集（组）装的数据
数据结构：包括结构关系的一个数据表
线性表列结构
列表结构
多重链接结构
正交表
树结构
女儿，见儿子
戴维，弗洛伦斯·奈廷格尔
戴维斯，马丁
德布鲁因，尼古拉斯戈维特
德·拉·劳比，西蒙
德·莫弗勒，阿伯拉罕
德·摩根，奥古斯杜斯
排错（调试）：探测和消除错误
DECA 指令（减 A）
十进计算机：一台主要是以十进制运算（进制为10）的计算机

DECX 指令（减 X）
DECI 指令（减 1）
在汇编语言中确定的符号
定义，循环，见 C字首循环定义
树中节点的次数
有向图形中顶点的次数
节点的删去：从一个数据结构消去它，可能把它恢复成可利用的存储
从可利用的空间表删去，见 Reservation（保留）
从双排队删去
从双链接表删去
从线性表删去
从排队删去
从堆栈删去
从树删去
从二维表删去
德穆思，霍华德·B
双排队：两个端点的排队
从双排队删去
输入受限的双排队
插入双排队
双排队的链接分配
输出受限双排队
双排队的顺序分配
一个公式的推导
在一个树结构中的随子
正方矩阵的行列式
小德尔，菲利普·德弗里
多伊奇，劳伦斯·彼得
杜威，梅尔威尔

trees (due to Galton)
for trees
Diagrams of structural information
before-and after
List structures
tree structures
Dickson, Leonard Eugene
Difference of function
divided
Differentiation, algebraic
chain rule for, see Faà di Bruno's formula
Digamma function
Digit: One of the symbols used in radix notation; usually a decimal digit, one of the symbols 0, 1, ..., or 9
Digraph, see Directed graph
Dijkstra, Edsger Wybe
Dilworth, Robert Palmer
Dimension of a partial ordering
d'Imperio, Mary E.
Directed graph
as flow chart
balanced
connected
regular
representation of
rooted
strongly connected
Discrete system simulation
synchronous
Disjoint sets: Set with no common elements
Disk files
Distribution: A specification of probabilities which govern the value of a random variable
binomial
normal
Poisson
uniform
Distributive law
DIV (divide)
Divided differences
Division converted to multiplication
Divisor: x is a divisor of y if $y \bmod x = 0$; it is a proper divisor if in addition $1 < x < y$
Dixon, Alfred Cardew
Dixon, Robert Dan
DLINK: Link downward
Doig, Alison
Domino problem
Double order for traversing tree
Doubly linked list
Dougall, John
Drum

对于二叉树的记号（归功于高尔顿）
对于树的性质记号
结构信息的图式
之前和之后图式
列表结构图式
树结构图式
迪克森，伦纳德·尤金
函数的差
除法差分
代数微分
关于微分的链接规则，见法底布鲁诺公式
双伽玛函数
数字：在进制记号中所使用的诸符号之一；通常的十进数字，是符号 0，1，…，9 之一
图型，见有向图形
迪伊克斯特拉，埃德斯盖尔·韦比
迪尔沃思，罗伯特·帕尔默
偏序的维数
德莫比里约，玛丽·E
有向图形
有向图形作为流程图
平衡的有向图形
连通的有向图形
正则的有向图形
有向图形的表示
有根的有向图形
强连通的有向图形
离散系统的模拟
同步的离散系统模拟
不相交的集合：无公共元素的集合
磁盘文件
分布：支配一个随机变量的值的概率的描述
二项式分布
正则分布
泊松分布
均匀分布
分配律
DIV（除）指令
除法差分
除法转换为乘法
因式：x 是 y 的一个因式，如果 $y \bmod x = 0$；如果加上 $1 < x < y$，则它是真因式
狄克逊，艾尔弗雷德·卡迪尤
狄克逊，罗伯特·丹
DLINK：向下的链接
多伊格，艾莉森
骨牌问题
遍历树的双重次序
双重链接的表
杜格尔，约翰
鼓

Dull, Brutus Cyclops 达尔，布鲁特斯·赛格罗普斯
Dummy variable 哑(虚拟)变量
Dunlap, James Robert 邓拉普，詹姆斯·罗伯特
Dynamic storage allocation 动态存储分配
  history 动态存储分配的历史
  running time estimates 动态存储分配运行时间估计
Dynastic order, see Preorder 朝代次序，见先根次序
Earley, Jackson Clark 厄尔利，杰克逊·克拉克
Easter date 复活节的日期
Edge in a graph 一图形中的边
Edwards, Daniel James 爱德华兹，丹尼尔·詹姆斯
Effective algorithm 能行算法
Eisenstein, Ferdinand Gotthold 艾森斯坦，弗迪南德·戈德霍尔德
Elementary symmetric functions 初等对称函数
Elevator (lift) system 电梯系统
Embedding of partial order into linear order, see Topological sorting 偏序嵌入到线性序中，见拓扑分类
Embedding of tree in another tree 把树嵌入到另一个树中
END END 语句
End of file 文件结束
Endorder, see Postorder 末端次序，见后根次序
Engles, Robert William 恩格尔斯，罗伯特·威廉
English letter frequencies 英文字母的频率
ENNA (enter negative A) ENNA 指令（负的进入A）
ENNX (enter negative X) ENNX 指令（负的进入X）
ENN1 (enter negative 1) ENN1 指令（负的进入1）
ENTA (enter A) ENTA 指令（进入A）
Entity, see Node 实体，见节点
Enumeration of tree structure 树结构的枚举
  history 树结构枚举的历史
Epictetus 伊壁鸠鲁学说
EQU (equivalent to) EQU 语句（等价于）
Equivalence algorithm (Algorithm 2.3.3E) 等价算法（算法 2.3.3E）
Equivalence between two algorithms 两个算法之间的等价性
Equivalence classes 等价类
Equivalence declaration 等价性说明
Equivalence relation 等价关系
Equivalent of a MIXAL symbol 一个 MIXAL 符号的等价物
Equivalent trees 等价树
Erase a data structure; To return all its nodes to available storage 抹去一个数据结构；把它的所有节点恢复成可利用的存储
  binary tree 抹去二叉树
  linear list 抹去线性表
  List 抹去列表
Erdélyi, Arthur 厄尔德莱，阿瑟
Erdwinn, Joel Dyne 厄尔德文，乔尔·戴恩
Errors, avoiding 避免错误
  computational 计算的错误
  detection of 错误的探测
Etherington, Ivor Malcolm Haddon 埃恩林顿，艾弗·马尔科姆·哈登
Ettingshausen, Andreas von 埃廷当欣，安德烈亚斯·冯
Euclides (=Euclid) 欧几里得
  algorithm for gcd 求最高公因的欧几里得算法

Euclidean domain
Euler, Leonhard
  constant
  summation formula
  theorem of
  totient function $\varphi(n)$
Eulerian circuit
Evaluate tree function
Evans, Arthur, Jr
Exchange operation
Exclusive or
Execution time, methods for determining
  for MIX instructions
Exercises, notes on
Exit; Place where control leaves a routine
Exits from subroutines, multiple
Expected value of a probability distribution; The average or "mean" value
Exponential integral; $E_1(x)$
Exponents, Laws of
Expressions, arithmetic, see Algebraic formula
Extended binary tree
External path length
Faà di Bruno, Francesco, formula of
Factorial
Factorial powers
Factorization into primes
FADD (floating add)
Fail-safe program
Family-order sequential representation of trees
Family tree
Farber, David Jack
Farey, John, series
Father, in a tree structure
FATHER link in tree
FCMP (floating compare)
FDIV (floating divide)
Ferguson, David Elton
Fermat, Pierre de
  theorem
Feynman, Richard Phillips
Fibonacci, Leonardo
  number system
  numbers; elements of the Fibonacci sequenci
  numbers, generating function
  Quarterly
  sequence
  string sequence
Fibonomial coefficients
Field; A designated portion of a set of data, usually consisting of contiguous (adjacent) symbols; e.g. a field of a punched card is usually a set of adjacent column position

欧几里得整区（环）
伦哈特，欧拉
  欧拉常数
  欧拉求和公式
  欧拉定理
  欧拉计数函数$\varphi(n)$
欧拉线路
计算树函数
小埃文斯，阿瑟
交换运算
排斥的或，异或
用于确定执行时间的一些方法
  MIX 指令的执行时间
关于习题的说明
出口；控制离开一个子程序的位置
从一些子程序的多个出口
一个概率分布预期的值；平均或"均"值
幂积分：$E_1(x)$
指数律
算术表达式，见代数公式
推广的二叉树
外部通路长度
弗朗西斯科，伐放奈的公式
阶乘
阶乘的幂
分解成质数
FADD 指令（浮点加）
不安全的程序
树的家族次序的串行表示
家族树
法伯，戴维·杰克
法雷，约翰级数
在一树结构中的父亲
树中的 FATHER 链接
FCMP 指令（浮点比较）
FDIV 指令（浮点除）
弗格森，戴维·埃尔顿
费尔马，皮尔·德
  费尔马定理
范曼，理查德·菲利普斯
斐波那契，伦纳德
  斐波那契数系
  斐波那契数；斐波那契序列的元素
  斐波那契数的生成函数
  斐波那契季刊
  斐波那契序列
  斐波那契字符串序列
斐波那契系数
场；一个数据集合的指定部分，通常由连续的（相邻的）符号组成，例如一穿孔卡片的场通常是由相邻的列位置的集合

partial, of MIX word — MIX 字的部分场
within a node — 一节点内的场
within a node, notations for — 在一节点内，场的记号
FIFO, see Queue — FIFO，即先进先出，见排队
Fifty percent rule — 百分之五十的规则
Final vertex of arc — 弧的最终的顶点
Fine, Nathan Jacob — 法因，内森·雅各布
First-fit method of storage allocation — 存储分配的"头一个适合"的方法
First-in-first out, see Queue — 先进先出，见排队
Fischer, Michael John — 费希尔，迈克尔·约翰
Fixed element of permutation — 排列的固定元素
Fixed point arithmetic — 定点算术
Flag, see Sentinel — 标志，见哨兵
Floating point arithmetic — 浮点算术
Floating-point operators of MIX — MIX 的浮点运算符
Floor function — "地板"函数
Flow chart — 框图（或日流程图）
Floyd, Robert W — 弗洛伊德，罗伯特·W
FLPL — FLPL 语言
FMUL (floating multiply) — FMUL 指令（浮点乘）
Förstemann, Wilhelm — 福斯特曼，威廉
Ford, Donald Floyd — 福特，唐纳德·弗洛伊德
Forecasting — 预报
Forest: Zero or more trees, see Trees — 森林：零个或多个树，见树
Formulas algebraic, see Algebraic formula — 代数公式，见A字首代数公式
logical — 逻辑公式
FORTRAN — FORTRAN 语言
Foster, Frederic Gordon — 福斯特，弗雷德里克·戈顿
Fractional part — 小数部分
Fragmentation problem — 断片问题
Franklin, Joel Nick — 富兰克林，乔尔·尼克
Free lattice — 自由格
Free storage, see Available space — 释放存储，见可利用空间
Free trees — 自由树
enumeration — 自由树的枚举
minimum cost — 极小代价的自由树
Front of queue — 排队的前沿
FSUB (floating subtract) — FSUB 指令（浮点减）
Fundamental cycles in graph — 图形中的基本回路
Furch, Robert — 弗兹，罗伯特
Future reference (in MIXAL) — (MIXAL 中的）未来的访问
restriction on — 对未来的访问的限制
Galler, Bernard Aaron — 加勒，伯纳德·艾伦
Galton, Francis — 高尔顿，弗朗西斯
Games, solution of — 一些游戏的解
Gamma function — 伽玛函数
incomplete — 不完备的伽玛函数
Garbage collection — 废料收集
Gardner, Martin — 加德纳，马丁
Garwick, Jan Vaumund — 加威克，简·沃蒙德
Gaskell, Robert Eugene — 加斯克尔，罗伯特·尤金
Gauss, Karl Friedrich — 高斯，卡尔·弗里德里克
reduction algorithm for matrix inversion — 矩阵求逆的归约算法

gcd, Greatest common divisor — 最高公因
Gelernter, Herbert Leo — 格伦特，赫伯特·利奥
Generating functions — 生成函数
  asymptotic values from — 生成函数的渐近值
  for a discrete probability distribution — 离散概率分布的生成函数
Genuys, Francois — 吉奴伊斯，弗朗索伊斯
Geometric progression, sum of — 几何级数的和
Gerberich, Carl Luther — 格伯利兹，卡尔·卢瑟
Gill, Stanley — 吉尔，斯坦利
Glaisher, James Whitbread Lee, constant — 格拉西尔，詹姆斯·惠特布雷德·李常数
Gredenko, Boris Vladimirovich — 格涅坚科，鲍利斯·弗拉基米罗维奇
GO button of MIX — MIX 的GO·按钮
Goldbach, Christian — 戈德巴赫，克里斯琴
Goldberg, Joel — 戈德堡，乔尔
Golden ratio — 黄金比率
Goldstine, Herman Heine — 戈德斯坦，赫尔曼·海因
Golomb, Solomon Wolf — 戈龙卜，索罗门·沃尔夫
Goncharov, Vasilii Leonidovich — 贡查罗夫，瓦西里·利奥尼多维奇
Good, Irving John — 古德，欧文·约翰
Gorn, Saul — 戈尼，索尔
Gould, Henry Wadsworth — 古尔德，亨利·沃兹沃思
Gower, John Clifford — 高尔，约翰·克利福德
Graph — 图形
  connected — 连通图形
  directed, see Directed graph — 有向图形，见D字首
Graphical display — 图形显示
Greatest common divisor — 最高公因
Greatest integer function, see Floor function — 最大整数函数，见"地板"函数
Grid — 格子
Griswold, Ralph Edward — 格里斯沃尔德，拉尔夫·爱德华
Haddon, Bruce Kenneth — 哈登，布鲁斯·肯尼特
Halayudha — 哈拉尤达
Hamilton, Dennis Eugene — 哈密尔顿，丹尼斯·尤金
Hamilton, Sir William Rowan, circuit — 哈密尔顿，威廉·罗恩爵士线路
Hamlet, Prince of Denmark — 哈姆莱特，丹麦王子
Hansen, Wilfred James — 汉森，威尔弗雷德·詹姆斯
Harary, Frank — 哈拉里，弗兰克
Hardware oriented algorithms — 面向硬件的算法
Hardy, Godfrey Harold — 哈迪，戈弗雷·哈罗德
Harmonic numbers — 调和数
  generating function — 调和数生成函数
  table — 调和数表
Harmonic series — 调和级数
Harrison, Michael Alexander — 哈里森，迈克尔·阿历山大
Hartmanis, Juris — 哈特马尼斯，朱里斯
Hautus, Matheus Lodewijk Johannes — 豪道斯，马恩斯·洛德威克·约翰尼斯
Head of list — 表头
Hellerman, Herbert — 赫勒曼，赫伯特
Henkin, Leon Albert — 亨金，利昂·艾伯特
Herbert, George — 赫伯特，乔治
Hermite, Charles — 赫密特，查尔斯
Heyting, Arend — 海沃汀，阿伦
Hilbert, David, matrix — 希尔伯特，戴维矩阵

HLT (halt)
Hoare, Charles Antony Richard
Holmes, Thomas Sherlock Scott
Holt Hopfenberg, Anatol Wolf
Honeywell H800
Hopper, Grace Murray
Huffman, David Albert
Hurwitz, Adolf
  generalized binomial formula
IBM 650
IBM 701
IBM 705
IBM 709
IBM 7070
Identity permutation
Iff: If and only if
Iliffe, John Kenneth
Illiac I
IN (input)
In-degree of vertex
INCA (increase A)
Incident matrix
Inclusion and exclusion principle
Incomplete gamma function
INCX (increase X)
INC1 (increase 1)
Indentation
Index: A number which indicates a particular element of an array (sometimes called a "subscript")
Index register
  modification of MIX instruction
Indirect addressing
Induction, mathematical
  generalized
Infinite series: A sum over infinitely many values
Infinite trees
Infinity lemma
Information: The meaning associated with data, the facts or concepts represented by data; often used also in a narrower sense as a synonym for "data", or in a wider sense to include any concepts which can be deduced from data
Information structure, see Data structure
Ingalls, Daniel Henry Holmes
Ingerman, Peter Zilahy
Initial vertex of arc
Inorder for binary tree
Input
  anticipated
  buffering
  operators of MIX
Input restricted deque
Insertion of node: Entering it into a data structure

HLT 指令（停机）
霍尔，查尔斯·安东尼·理查德
霍姆斯，托马斯·舍洛克·斯库特
霍尔特·霍普夫伯格，阿纳托尔·沃尔夫
霍尼韦尔公司H800计算机
霍珀，格雷斯·默里
赫夫曼，戴维·艾伯特
胡尔维兹，阿道夫
  胡尔维兹广义的二项式公式
IBM 公司650计算机
IBM 公司701计算机
IBM 公司705计算机
IBM 公司709计算机
IBM 公司7070计算机
恒等排列
当且仅当
艾利夫，约翰·肯尼特
伊利阿克Ⅰ计算机
IN 指令（输入）
顶点的输入次数
INCA 指令（增A）
作弧矩阵
包括和排除原理
不完备的伽玛函数
INCX 指令（增X）
INC1 指令（增1）
凹入
下标：用于指出一数组的具体元素的数（有时称为"足标"）
变址寄存器
  MIX 指令的变址寄存器的修改
间接编址
数学归纳法
  广义的数学归纳法
无穷级数，对于无穷多值的一个求和
无穷树
无穷性引理
信息：与数据相关联的意义，为数据所表示的事实或概念；通常在狭窄的意义下也用作数据的同义语，或者在更广的意下用作可由数据导出的任何概念
信息结构，见数据结构
英戈尔斯，丹尼尔·亨利·霍姆斯
英格曼，彼得·齐拉希
弧的起始顶点
二叉树的中根次序
输入
  抢先输入
  输入缓冲
  MIX 的输入操作等
输入受限双排队
节点的插入：把它放入一个数据结构中

into doubly linked list
into linear list
into linked list
into tree
into two dimensional list
onto stack
instruction, machine language: A code which, when interpreted by the circuitry of a computer, causes the computer to perform some action
in MIX
symbolic form
INT (interrupt)
Integer
Integration
related to summation
Interchange of values
Interchanging the order of summation
Interest compound
Interlock time: Delay of one part of a system while another part is busy completing some action
Internal path length
Interpreter (interpretive routine)
Interrupt
Inverse (modulo m)
Inverse of matrix
Inverse of permutation
Inversion problem
Invert a linked list
I/O: Input or output
IOC (input output control)
IPL
Irons, Edgar Towar
Irreflexive relation
Isolated vertex
Iverson, Kenneth Eugere
I1 register of MIX
I-register of MIX
JACM: Journal of the ACM, a publication of the Association for Computing Machinery
Jacquard, Joseph Marie, loom
JAN (jump A negative)
JANN (jump A nonnegative)
JANP (jump A nonpositive)
JANZ (jump A nonzero)
JAP (jump A positive)
Jarden, Dov
JAZ (jump A zero)
JBUS (jump busy)
JE (jump on equal)
Jenkins, D. P
JG (jump on greater)
JGE (jump on greater or equal)
JL (jump on less)

把它放入双重链接表中
把它放入线性表中
把它放入链接表中
把它放入树中
把它放入二维表中
把它放到堆栈上
机器语言指令：一个代码，当由一计算机线路解释它时，引起计算机实施某一动作
MIX 的指令
指令的符号形式
INT 指令（中断）
整数
积分
同求和有关的积分
诸值的互换
交换求和的次序
复利
互锁时间：在一部分忙于完成某个动作时，延迟系统的另一部分
内部通路长度
解释程序
中断
(modulo m) 下的逆
矩阵的逆
排列的逆
反演问题
颠倒一个链接表
输入/输出
IOC（输入 输出控制）
IPL 语言
艾恩斯，埃德加·托沃
非反身关系
孤立点
艾弗森，肯尼特·尤金
MIX 的 I 1 寄存器
MIX 的 I- 寄存器
JACM：ACM 杂志，（美国）计算机器协会刊物
雅克卡德，约瑟夫·玛丽，卢默
JAN 指令（A为负跳）
JANN 指令（A非负跳）
JANP 指令（A非正跳）
JANZ 指令（A非零跳）
JAP 指令（A为正跳）
贾登，达夫
JAZ 指令（A为0跳）
JBUS 指令（忙碌时跳）
JE 指令（相等时跳）
詹金斯，D. P
JG 指令（大于时跳）
JGE 指令（大于或等于跳）
JL 指令（小于时跳）

JLE (jump on less or equal)
JMP (jump)
JNE (jump on not equal)
JNOV (jump on no overflow)
Jodeit, Jane G.
Johnson, Lyle Robert
Joke
Jordan, Camille
Jordan, Karoly (= Charles)
Jordan, Wilhelm. reduction algorithm for matrix inversion
Josephus, Flavius, problem
JOV (jump on overflow)
JRED (jump ready)
JSJ (jump, save J)
Jump operator of MIX
Jump trace
JXN (jump X negative)
JXNN (jump X nonnegative)
JXNP (jump X nonpositive)
JXNZ (jump X nonzero)
JXP (jump X positive)
JXZ (jump X zero)
J1N (jump 1 negative)
J1NN (jump 1 nonnegative)
J1NP (jump 1 nonpositive)
J1NZ (jump 1 nonzero)
J1P (jump 1 positive)
J1Z (jump 1 zero)
Kahn, Arthur B
Kahrimanian, Harry George
Kaucky, Josef
Kepler, Johann
Kilner, Joyce
King, James Cornelius
Kirchhoff, Gustav Robert
  law of conservation of flow
Knopp, Konrad
Knotted List
Knowlton, Kenneth Charles
Knuth, Donald Ervin
Knuth, Ervin Henry
Knuth, Jill Carter
Kolmogorov, Andrei Nikolaevich
König, Denes
Kozelka, Robert Marvin
Kramp, Christian
Krogdahl, Leopold. delta notation
Kruskal, Joseph Bernard
Kummer, Ernst Eduard
La Loubere, Simon de
Labeled trees, enumeration of
Lagrange, Joseph Louis, Comte

JLE 指令（小于或等于跳）
JMP 指令（跳）
JNE 指令（不等时跳）
JNOV 指令（无溢出时跳）
乔德特，简·G
约翰逊，莱尔·罗伯特
笑话
乔丹，卡米尔
乔丹，卡罗里（=查尔斯）
约旦，威廉关于矩阵求逆的简化算法
约瑟夫，弗雷维厄斯问题
JOV 指令（溢出时跳）
JRED 指令（准备好跳）
JSJ 指令（跳，保存 J）
MIX 的跳转操作符
跳转踪迹
JXN 指令（X为负跳）
JXNN 指令（X非负时跳）
JXNP 指令（X非正跳）
JXNZ 指令（X非0跳）
JXP 指令（X为正跳）
JXZ 指令（X为0跳）
J1N 指令（1为负跳）
J1NN 指令（1为非负跳）
J1NP 指令（1非为跳）
J1NZ 指令（1非0跳）
J1P 指令（1为正跳）
J1Z 指令（1为0跳）
卡恩，阿瑟·B
卡里美尼安，哈里·乔治
考克夫，约瑟夫
开普勒，约翰
基尔纳，乔伊斯
金，詹姆斯·科尼利厄斯
克希霍夫，古斯塔夫·罗伯特
  克希霍夫流线守恒定律
诺普，康拉德
打结的列表
诺尔顿，肯尼特·查尔斯
克努特，唐纳德·欧文
克努特，欧文·亨利
克努特，吉尔·卡特
柯尔莫哥罗夫，安德烈·尼古拉耶维奇
科尼格，德尼斯
科泽尔卡，罗伯特·马文
克兰普，克里斯琴
克罗格达尔，利奥波德·德尔塔记号
克鲁斯科尔，约瑟夫·伯纳德
库莫尔，厄恩斯特·埃杜瓦德
拉·劳伯，西蒙·德
带标号树的枚举
拉格朗日，约瑟夫·路易斯，科莫特

identity
inversion formula
Lamé, Gabriel
Language: A set of strings of symbols, usually accompanied by conventions for assigning a "meaning" to each string in the set
Laplace, Pierre Simon, marquis de transform
Large programs, writing
Last-in-first-out, see Stack
almost
Lattice, free
Lawson, Harold Wilbur, Jr
LDA (load A)
LDAN (load A negative)
LDX (load X)
LDI (load I)
LDIN (load I negative)
Least-recently used replacement
Left subtree in a binary tree
Legendre, Adrien Marie
symbol
Leibnitz (=Leibniz), Gottfried Wilhelm Freiherr von
Leiner, Alan L.
Leonardo of Pisa
Letter frequencies in English
Level of node in tree
Level-order sequential representation of tree
LeVeque, William Judson
Lévy, Paul
Lexicographic order
L'Hospital, Guillaume Francois Antoine de, marquis de Sainte-Mesme, rule of
Liberation of reserved storage
LIFO, see Stack
Lilius, Aloysius
Lineal chart
Linear lists
Linear ordering
embed partial ordering into, see Topological sorting
of trees
Linear recurrence
Link
diagram of
field, purpose of
manipulation, avoiding errors in
null
Link variable
Linkage: Manner of setting links
circular
coroutine
double
orthogonal

拉格朗日恒等式
拉格朗日反演公式
拉姆，加布里埃尔
语言：符号串的集合，通常伴之以对集合中每个串赋加意义的一些约定
拉普拉斯，皮埃尔·西蒙，马奎斯·德·拉普拉斯变换
编写大型程序
后进先出，见堆栈
几乎后进先出
自由格
小劳森，哈罗德，威尔伯
LDA 指令（装入A）
LDAN 指令（负A装入A）
LDX 指令（装入X）
LDI 指令（装入I）
LDIN 指令（负的装入I）
替换最近最少使用的
二叉树中的左子树
勒让德，艾德里安·玛丽
勒让德符号
莱布尼兹，戈特弗里德·威廉·弗雷赫尔·冯
莱因尼尔，艾伦·L
比萨的伦纳德
英语中的字母频率
树中节点的层数
树的层次次序的串行表示
莱维克，威廉·贾德森
利维，波尔
辞典编排次序
洛必达，吉尔劳默·弗朗科伊斯·安托万·德马奎斯·德·圣特-梅斯莫规则
保留存储的释放
后进先出，见堆栈
利里留斯，阿洛伊修斯
直系图
线性表
线性次序
把偏序嵌入线性序中，见拓扑分类
树的线性次序
线性递归
链接
链接的图式
链接场的目的
在链接处理中避免出错
空链接
链接变量
链接：设置链接的方式
循环链接
链接共协程序
双重链接
正交链接

straight 直接链接
subroutine 子程序链接
two way 双向链接
Linked allocation of tables 表格的链接分配
array 数组的链接分配
contrasted to sequential allocation 链接分配同顺序分配对照
linear list 线性表的链接分配
tree structurec 树结构的链接分配
Linked-memory philosophy 链接内存原理
Linking automaton 链接自动化
LISP LISP语言
List; Ordered sequence of zero or more elements 表：0个或多个元素的有序序列
circular 循环表
doubly linked 双重链接表
linear 线性表
of available space, see Available space list 可利用空间表，见A字首
List(capital-List)structure 列表结构
copying 复写列表结构
diagrams of 列表的图式
distinguished from lists 同表的区别
equivalence between 列表间的等价性
notation for 列表的记号
representation 列表的表示
List head 表头
List processing system 列表处理系统
Listing, Johann Benedict 利斯汀，约翰·本尼迪克
Literal constants in MIXAL MIXAL中的文字常数
LLINK; Link to the left LLINK：对左边的链接
in binary tree 二叉树中的LLINK
in doubly linked list 在双重链接表中的LLINK
Lloyd, Stuart Phinney 劳埃德，斯图尔特·菲尼
Loading operators of MIX MIX的装入操作符
Loading routine 装入程序
LOC LOC语句
Local symbols in MIXAL MIXAL中的局部符号
Locally defined function in tree 树中局部地定义的功能
Location; The memory address of a computer word or node; or the memory cell itself 单元：一个计算机字或节点的内存地址，或内存单元本身
Location counter in MIXAL MIXAL中的地址计数器
Location field of MIXAL line MIXAL行的地址场
Logarithm 对数
binary 二进制对数
common 常用对数
natural 自然对数
power series 幂级数的对数
Logical formulas 逻辑公式
Loop detection 循环探测
Loopstra, B.J. 卢普斯特拉，B.J.
Lovelace, Ada Augusta, countess of 艾达·奥古斯塔 洛夫莱斯伯爵夫人
LSON LSON链接（左边的儿子）
LTAG LTAG场，指出第一场是否链接
Lucas, Edouard 卢卡斯，埃道尔德
Luhn, Haus Peter 卢恩，汉斯·彼得

Lukasiewicz, Jan	鲁卡西维兹，简
Lunch counter problem	便餐台问题
Lynch, William Charles	林奇，威廉·查尔斯
Machine language: A language which directly governs a computer's actions, as it is interpreted by a computer's circuitry	机器语言：当由一计算机线路解释时，支配一计算机动作的语言
symbolic, see Assembly language	符号机器语言，见汇编语言
MacMahon, Maj. Percy Alexander	麦克马洪，梅伊·珀西·阿历山大
Macro instruction: Specification of a pattern of instructions and/or pseudo-operators which may be frequently repeated within a program	宏指令：在一个程序内可能经常被重复的一组指令或伪操作符的详细说明
Madnick, Stuart Elliot	马德尼克，斯图尔特·埃利奥特
Magic square	纵横图
Magnetic tape	磁带
Mallows, Colin Lingwood	马洛斯，科林·林伍德
Margolin, Barry Herbert	马戈林，巴里·赫伯特
Mark I calculator	马克 I 计算器
Marking algorithms: Algorithms which "mark" all nodes that are accessible from some given nodes	作标记算法：标记由某些给定的节点可访问的所有节点的算法
Markov, Andrei Andreevich (the elder)	马尔科夫，安德列·安德烈维奇（老的）
process	马尔科夫过程
Markov, Andrei Andreevich (the younger)	马尔科夫，安德列·安德列维奇（年轻的）
Markowitz, Harry Max	马柯维兹，哈里·马克斯
Math. Comp.: Mathematics of Computation, a journal published by the American Mathematical Society	《计算数学》杂志，美国数学协会出版的
Mathematical induction	数学归纳法
generalized	推广的数学归纳法
Matrix	矩阵
Cauchy	柯西矩阵
combinatorial	组合矩阵
determinant of	矩阵的行列式
Hilbert	希尔伯特矩阵
incidence	伴随矩阵
inverse of	矩阵的逆
multiplication	矩阵乘法
representation of	矩阵的表示
sparse	稀疏矩阵
transpose of	矩阵的转置
tridiagonal	对角矩阵
triangular	三角矩阵
Vandermonde	范德蒙德矩阵
Matrix, Dr. Irvine, Joshua	马特里克，欧文·乔舒亚博士
Mauchly, John William	莫兹里，约翰·威廉
Maurolico, Francesco	莫洛里科，弗朗西斯科
Maximum, algorithm to find	求极大值的算法
McCall's	麦考尔的
McCarthy, John	麦卡锡，约翰
McCracken, Daniel Delbert	麦克拉肯，丹尼尔·戴尔伯特
McFliece, Robert James	麦克尔里斯，罗伯特·詹姆斯
McIlroy, Malcolm Douglas	麦基尔洛伊，马尔科姆·道格拉斯
McNeley, John Louis	麦克尼利，约翰·路易斯
Mealy, George	米利，乔治

Mean (average) of a probability distribution — 概率分布的均值
Meek, H. V. — 米克，H. V.
Meggitt, John E. — 梅吉特，约翰·E
Memory: Part of a computer system used to store data — 存储器：用来存储数据的一个计算机系统的部分
  cell of — 存储单元
  types of — 存储的类型
  update — 更新内存
Memory map — 存储映象
Merging — 合并
Merner, Jack Newton Forsythe — 默纳，杰克·牛顿·福赛
Metcalfe, Howard Hurtig — 梅特卡夫，霍华德·赫迪格
Military game — 军事游戏
Miller, Kenneth William — 米勒，肯尼特·威廉
Minimum path length — 极小通路长度
Minimum wire length — 极小导线长度
Minsky, L. — 明斯基，L
Mitchell, William Charles — 米切尔，威廉·查尔斯
MIX computer — MIX计算机
  assembly language for — MIX计算机的汇编语言
  extensions to — MIX计算机的扩充
  instructions, form of — MIX计算机的指令形式
  instructions, summary — MIX计算机指令，小结
  simulator of — MIX计算机的模拟程序
MIXAL: MIX Assembly Language — MIXAL：MIX的汇编语言
Mixed radix number system — 混合进制系统
Mock, Owen Russell — 莫克，欧文·拉塞尔
  mod — 模
  modulo — 模
Moments of probability distribution — 概率分布的矩
Moritor routine, see Trace routine — 监督程序，见跟踪程序
Monte Carlo method: Experiments with random data — 蒙特卡罗方法：以随机数进行实验
Moon, John Wesley — 穆恩，约翰·韦斯利
Mordell, Louis Joel — 莫德尔，路易斯·乔尔
Morrison, Emily — 莫里森，埃米莉
Morrison, Philip — 莫里森，菲利普
Mother, see Father — 母亲，见父亲
Mouse algorithm — 鼠探算法
MOVE — MOVE指令
MOVE CORRESPONDING in COBOL — 在COBOL中的MOVE CORRESPONDING
Moyse, Alphonse, Jr — 小莫伊西，阿方斯
MUG: MIX User's Group — MUG：MIX用户组
MUL (multiply) — MUL指令（乘法）
Multilinked structures — 多重链接结构
Multinomial coefficient — 多项式系数
Multinomial theorem — 多项式定理
Multipass algorithm — 多遍扫描算法
Multiple: x is a multiple of y if y is a divisor of x, i.e., $x = ky$ for some integer k. — 倍数：x 是 y 的倍数，如果 y 是 x 的因子，即对于某个整数 k，有 $x = ky$
Multiple entrances to subroutines — 对子程序的多个入口
Multiple exits from subroutines — 从子程序的多个出口
Multiple precision arithmetic — 多精度算术
Multiplication of permutation — 排列的乘法
Multiplication of polynomial — 多项式的乘法

Multiplicative function 乘性函数
Multiset, Analogous to a set, but elements may appear more than once 多重集，和一个集合类似，但其元素可以出现一次以上
Multiway decisions 多路判断
Nahapetian, Armen 阿尔门·内和彼辛安
Napier, John 内皮尔，约翰
National Science Foundation 国家科学基金会
Natural correspondence between binary trees and forests 二叉树和森林之间的自然对应
Natural logarithm 自然对数
Naur, Peter 瑙尔，彼得
Negative, Less than zero(not zero) 负，小于0(非0)
Nested parenthesis 嵌套的括弧
Nested sets 嵌套的集合
Nesting store 嵌套存储
Network, see Graph 网络，见图形
Neville, Eric Harold 内维尔，埃里克·哈罗德
Newell, Allen 纽厄尔，艾伦
Newton, Sir Isaac 牛顿，艾萨克爵士
identities 牛顿恒等式
Nicomachus of Gerasa 希腊的尼科梅彻斯
Nil link, see Null link 空链接
Niven, Ivan 尼文，伊凡
Noah 诺亚
Node, Basic component of data structures 节点，数据结构的基本
address of 节点的地址
diagram of 节点的图式
link to 对节点的链接
notations for field 场的节点记号
size of 节点的大小
variable-size 可变大小的节点
NODE NODE场
Node variable 节点变量
Nonnegative, Zero or positive 非负，零或正
NOP(no operation) NOP指令（空操作）
Normal distribution 正态分布
Notations, index to 记号索引
Notes on the exercise 关于习题的说明
Null link 空链接
in tree 树中的空链接
NUM(convert to numeric) NUM指令（转换成数值）
Number, definition 数的定义
Number system, A language for representing numbers 数系，表示数的一种语言
binomial 二项式数系
decimal 十进数系
Fibonacci 斐波那契数系
mixed-radix 混合进制数系
phi 斐（Φ）
Number theory, elementary 初等数论
Nygaard, Kristen 尼加德，克里斯顿
O-notation O记号
O'Beirne, Thomas Hay 奥贝恩，托马斯·海
Oettinger, Anthony Gervin 奥廷格，安东尼·杰文

Office of Naval Research 海军研究所
Okada, Satio 奥凯达，萨修俄
Oldenburg, Henry 奥尔登伯，亨利
One-plus-one address computer 一加一地址计算机
One-way equalities 单向相等性
One-way linkage, see Straight linkage 单向链接，见直接链接
  Circular linkage 循环链接
Onodera, Rikio 奥诺迪拉，赖基俄
Oper subroutine, see Macro instruction 开子程序，见宏指令
Operation code field, of MIX instruction MIX指令的操作码场
of MIXAL line MIXAL语句的操作码场
Optimal search procedure 最优检索过程
Order of succession to throne 宝座之沿袭次序
Ordered tree 有序树
Ordering, A transitive relation between objects of a set 次序，一个集合的对象之间的一个传递关系
  lexicographic 辞典编辑次序
  linear 线性次序
  linear, of trees 树的线性次序
  partial 部分序
  well 良序
Ore, Øystein 奥尔，奥伊斯登
Oresme, Nicole 奥雷斯姆，尼科尔
Oriented binary tree 有向二叉树
Oriented cycle in directed graph 有向图形的有向弧环
Oriented path in directed graph 有向图形的有向通路
Oriented trees 有向树
  canonical representation 有向树的典型表示
  enumeration 有向树的枚举
  representation of 有向树的表示
  root changed in 有向树中改变根
ORIG (origin) ORIG伪指令
Orthogonal lists 正交表
Otter, Richard Robert 奥特，理查德·罗伯特
OUT (output) OUT指令(输出)
  buffering 输出缓冲
  operators of MIX MIX的输出操作符
Output-restricted deque 输出受限双排队
Overflow toggle of MIX MIX的溢出开关
Packed data, Data which has been compressed into a small space, e.g., by putting two or more elements of data into the same word of memory 集装数据，已经被压缩到一个小空间中的数据，例如，把两个或多个数据元素置入内存同一字中
Paging 分页
Pall, Gordon 帕尔，戈登
Parallelism, see Discrete system simulation 并行性，见离散系统的模拟
Parameters of subroutines 子程序的参数
Parker, William Wayne 帕克，威廉·韦恩
Parmelee, Richard Paine 帕米利，理查德·佩因
Partial field designations in MIX MIX中的部分场的指定
Partial fractions 部分分式
Partial ordering 偏序
Partitions of a set 一个集合的划分
Partitions of an integer 一个整数的分划

Pascal, Blaise — 帕斯卡尔，布莱斯
  triangl, see — 帕斯卡尔三角，见二项式系数
  Binomial coefficients
Pass, in a program — 一个程序中的趟
Path, in a graph or directed graph — 在一个图形或有向图形中的通路
  oriented — 有向通路
  random — 随机通路
  simple — 简单通路
Patt, Yale Nance — 帕特，耶尔·南斯
Pawlak, Zdzislaw — 波莱克，泽吉斯洛
PDP-4 — PDP-4计算机
Pedigree — 家系
Peripheral device: An I/O compenent of a computer system — 外部设备，一个计算机系统的输入输出设备
Perlis, Alan J — 珀利斯，艾伦·J
Permanent of a square matrix — 正方矩阵的永久式
Permutations — 排列
  inverse of — 排列的逆
  multiplication of — 排列的乘法
  notations for — 排
PERT network — PERT 网络
Peters, Johann(=Jean)Theodor — 彼得斯，约翰(=琼)·西奥多
Peterson, William Wesley — 彼得森，威廉·沃尔利
Phi, see Golden ratio — 斐，见黄金比率
  number system — 斐数系
Phidias — 菲迪亚斯
Philco S2000 — 菲尔科S 2000计算机
Pile — 派尔
Pilot ACE computer — 飞行者 ACE 计算机
Pisano, Leonardo — 皮萨诺，伦纳德
Pivot step — 主要步骤
PL/I — PL/I语言
PL/MIX — MIX计算机的PL/I 语言
Plane tree, see Ordered tree — 平面树，见有序树
Playing cards — 扑克牌
Plex — 从
Pointer, see Link — 指针，见链接
Pointer variable: A variable whose values are links — 指针变量：其值为一些链接的变量
Poisson, Siméon Denis, distribution — 泊松，西米恩·丹尼斯分布
Polish notation, see Prefix notation — 波兰记号，见前缀记号
  Postfix notation — 后缀记号
Polonsky, Ivan Paul — 波伦斯基，伊凡·保尔
Polya, György(=George) — 波利亚，乔治
Polynomials — 多项式
  addition of — 多项式的加法
  Bernoulli — 贝努利多项式
  Chebyshev — 契比雪夫多项式
  differences of — 多项式的差
  multiplication of — 多项式的乘法
  representation of — 多项式的表示
Pool of available nodes, see Available space list — 可利用节点池，见可利用空间表
Pooled buffers — 联营缓冲
Pop up a stack: Delete its top element — 弹出堆栈：删去其顶上元素

Positive; Greater than zero(not zero) 正的；大于0的（非0）
Postfix notation 后缀记号
Posting a new item, see Insertion 插入一个新条款
Postorder for binary tree 二叉树的后根次序
Postorder for tree 树的后根次序
Postorder with degrees, representation of trees 树表示，带次数的后根次序
Power of number 数的幂
  factorial 阶乘的幂
Power series; Sum of the form $\Sigma_{k\geq 0}a_k z^k$, see Generating function 幂级数，形如 $\Sigma_{k\geq 0}a_k z^k$ 的和 见生成函数
  convergence of 幂级数的收敛性
  manipulation of 幂级数的处理
Pratt, Vaughan Ronald 普拉特，沃恩·罗纳德
Prefix notation 前缀记号
Preorder for binary tree 二叉树的前缀记号
Preorder sequential representation of trees 树的前根次序的顺序表示
  with degrees 带次数的前根次序
Prim, Robert Clay 普赖姆，罗伯特·克莱
Prime numbers 质数
  algorithm to compute 计算质数的算法
  factorization into 因式分解成质数
Printer 打印机
Prinz, D. G. 普林兹，D. G
Priority queue 优先非队
Probability distribution; A sepicification of probabilities which govern the value of a random variable 概率分布；支配一个随机变量的值的概率的描述
  average("mean")value of 概率分布的均值
  generating function for 概率分布的生成函数
  variance of 概率分布的方差
Procedure, see Subroutine 过程，见子程序
Procedure for reading this set of books 阅读这部书的步骤
Program; Representation in some precise, formalized language of a computational method 程序；一个计算方法在某个精确的形式化语言下的表示
Programming language; A precise, formalized language in which programs are writen 程序设计语言；用于写程序的一个精确的形式化语言
Programs, hints for construction of 对构造程序的提示
Progression, arithmetic, sum of 算术级数的和
Proof of algorithms 算法的证明
Proper divisor, see Divisor 真因子，见因子
Propositional calculus 命题演算
Prüfer, Heinz 普里费，海因茨
Pseudo-operator; A construction in a programming language which is used to control the translation of that language into machine language 伪操作符；在一个程序设计语言中，用来控制把该语言翻译成机器语言的一个构造
Psi function 普函数
Purdom, Paul Walton, Jr. 小珀多姆，保尔·沃尔顿
Push down list, see Stack 下推表，见堆栈
Push down onto a stack; Insert a new top element 下推到一个堆栈，插入一个新的顶上元素
Putnam, Hilary 普特南
q-binomial theorem q-二项式定理
q-nomial coefficients q项式系数
Quadratic Euclidean domain 二次欧几里得整环
Quadratic reciprocity law 二次倒数律

Qualification of names — 名称的选定
Quasi-parallel processing, see Discrete system simulation — 拟并行处理，见离散系统模拟
Queue — 排队
  deletion from front — 从前端删去
  insertion at rear — 在后端插入
  linked allocation — 链接分配
  sequential allocation — 顺序分配
Quick, Jonathan Horatio — 奎克，乔纳森·霍雷肖
Rāmānujan Aiyangar, Srinivāsa — 拉马纳逊·艾扬格尔，斯里尼瓦萨
Ramus, Christian — 拉马斯，克里斯蒂
Randell, Brian — 兰德尔，布赖恩
Random path — 随机通路
Raney, George Neal — 拉尼，乔治·尼尔
Raphael, Bertram — 拉菲尔，伯特尼
Rational number — 有理数
RCA 601 — RCA601 计算机
Read, Ronald Cedric — 里德，罗纳德·塞德里克
Reading: Doing input — 读：进行输入
Real number — 实数
Real time — 实时
Reallocate sequentially stored tables — 对顺序存储表进行再分配
Rear of queue — 排队后部
Recipe — 菜谱
Reciprocity formulas — 互反公式
Recomp I — 雷科姆普 I
Record: A set of data that is input or output at one time, see also Node — 记录：作为一次输入或输出的数据集合，也见节点
Records, blocking of — 记录的分区
Rectangular arrays — 矩形数组
Recurrence relation: A rule which defines each element of a sequence in terms of the preceding elements — 递归关系：借助于前边的元素定义一个序列的每个元素的一个规则
Recursive definition — 递归定义
Recursive List — 递归表
Recursive use of subroutine — 子程序的递归使用
Ref, see Link — Ref 场，见链接
Reference, counter technique — 访问计数器技术
Reflexive relation — 反身关系
Registers: Portions of a computer's internal circuitry in which data is processed, the most accessible data kept in a machine appears in its registers — 寄存器：一台计算机的内部线路部分，要处理的数据就在其中。保持于一机器中的大多数可访问数据出现于它的寄存器中
  of MIX — MIX的寄存器
  saving and restoring contents of — 保存和恢复寄存器的内容
Regular directed graph — 正则的有向图画
Relation: A property which holds for certain sets (usually odreded pairs) of elements, for example, "<" is a relation defined for ordered pairs (x, y) of integers, and the property "$x<y$" holds if and only if x less than y. — 关系：对某些集合的元素（通常是对有序的对偶）成立的一个性质；例如，"<" 是对于整数的有序对 (x, y) 定义的一个关系，且性质 $x<y$ 成立当且仅当 x 小于 y
  antisymmetric — 反对称关系
  asymmetric — 非对称关系
  equivalence — 等价关系
  irreflexive — 非反身关系

symmetric 对称关系
transitive, see Ordering 传递关系，见次序
Relatively prime integers 互质整数
RELEASE a buffer 释放一个缓冲
Remove from structure, see Deletion 从结构中撤消，见删去
Renyi, Alfred 伦尼，艾尔弗雷德
Repacking 改组
Replacement operation 替代运算
Replicative function 重迭函数
Representation (inside a computer) (在一计算机内的) 表示
methods for choosing 选择表示的方法
of algebraic formulas 代数公式的表示
of arrays 数组的表示
of binary trees 二叉树的表示
of deques 双排队的表示
of directed graphs 有向图形的表示
of forests 森林的表示
of Lists 列表的表示
of oriented trees 有向树的表示
of polynomials 多项式的表示
of queues 排队的表示
of stacks 堆栈的表示
of trees 树的表示
Reservation of free storage 自由存储的保留
Reversion storage 反转存储
Rice, Stephan Oswald 赖斯，斯蒂芬·奥斯瓦德
Riemann, George Friedrich Berchard 黎曼，乔治·弗里德里克·伯恩哈德
Right subtree in a binary tree 二叉树中的右子树
Right-threaded tree structure 右穿线树结构
Ring structure 环形结构
Riordan, John 赖尔登，约翰
RLINK, Link to the right RLINK 场对右边的链接
in binary tree 二叉树中的 RLINK
in doubly linked list 双重链接表中的 RLINK
in List 列表中的RLINK
in tree see BROTHER link 树中的RLINK，见BROTHER 链接
Robertson, James Chalmers 罗伯逊，詹姆斯·查默斯
Robinson, Raphael Mitchel 鲁宾逊，拉菲尔·米切尔
Robson, John Michael 罗布逊，约翰·迈克尔
Rodrigues, Banjamin Olinde 罗德里格斯，班杰明·奥林德
Roll 案卷
Root of number 数的根
Root of tree 树的根
change of 树的根的变动
Rooted directed graph 有根有向树
Rooted tree, see Oriented tree 有根树，见有向树
Ross, Douglas Taylor 罗斯，道格拉斯·泰勒
Rothe, Heinrich August 罗塞，海因里希·奥古斯特
Rounding 舍入
Row major order 行优先次序
RTAG RTAG场
Running time, see Execution time 运行时间，见执行时间
Russell, Lawford John 拉塞尔，劳福德·约翰

Saddle point	马鞍点
Salton, Gerard Anton	萨尔顿，杰勒德·安东
Sammet, Jean Elaine	萨米特，琼·伊莱恩
Satterthwaite, Edwin Hallowell, Jr.	小萨特思韦特，埃德温·霍洛韦尔
Scaled decimal arithmetic	带比例的十进算术
Schatzoff, Martin	沙佐夫，马丁
Scherk, Heinrich Ferdinand	谢尔克，海因里希·费迪南德
Schlatter, Charles Fordemwalt	施拉特，查尔斯·福登沃尔特
Schlatter, William Joseph	施拉特，威廉·约瑟夫
Schorr, Herbert	肖尔，赫伯特
Schorr-Kon, Jacques J.	肖尔·康，雅克·J
Schorre, Dewey Val	肖尔，杜威·维尔
Schreier, Otto	施利耶尔，奥托
Schröder, Ernst	施罗德，恩恩斯特
Schützenberger, Marrel Paul	舒特贝尔杰，马勒尔·保罗
Schwartz, Eugene Sidney	施瓦茨，尤金·西德尼
Schwartz, Hermann Amandus, inequality: $(\Sigma a_k b_k)^2 \leq (\Sigma a_k^2)\ (\Sigma b_k^2)$ [due to Buniakovskii, 1859] see Lagrange's identity	施瓦茨，赫尔曼·阿曼达斯不等式：$(\Sigma a_k b_k)^2 \leq (\Sigma a_k^2)(\Sigma b_k^2)$[归功于布尼雅科夫斯基1859]，见拉格朗日恒等式
Schwenk, Allen John	施文克，艾伦·约翰
Schweppe, Earl Justin	施韦普，厄尔·贾斯廷
SCOPE link	SCOPE 链接
Scroll	卷轴
Segner, Johann Andreas von	西格纳，约翰·安德烈·冯
Seiden, Esther	塞登，埃斯特
Selfridge, John L.	塞尔弗利齐，约翰·L
Semaphore	信号灯
Semi-invariants of a probability distribution	一个概率分布的半不变量
Sentinel: A special value placed in a table, e.g., to mark the boundaries of the table, designed to be easily recognizable by the accompanying program	哨兵：放置在一个表格的特殊值，例如，用于标志此表格的边界，以便伴随的程序容易地认识
Sequential(consecutive)allocation of tables	表格的顺序(连续的)分配
array	数组的顺序分配
contrasted to linked allocation	顺序分配同链接分配对照
linear list	线性表的顺序分配
tree structures	树结构的顺序分配
Series, infinite: A infinite sum	无穷级数：一个无穷的求和
Sets, partition of	集合的分划
Shaw, Christopher Joseph	肖，克里斯托弗·约瑟夫
Shaw, John Clifford	肖，约翰·克利福德
Shelf	架子
Shell, Donald Lewis	谢尔，唐纳德·刘易斯
Shepp, Lawrence Alan	谢普，劳伦斯·艾伦
Shift operators of MIX	MIX的移位操作符
Shih chieh, Chu	朱世杰
Sibling, see Brother	兄弟，见B字首兄弟
Siklóssy, Laurent	西克洛西，劳伦特
Sister, see Brother	姐妹，见兄弟
Similar trees	相似的树
Simon, Herbert Alaxander	西蒙，赫伯特·阿历山大
Simple oriented path	简单的有向通路
Simplification, algebraic	代数简化
SIMSCRIPT	SIMSCRIPT语言

SIMULA
Simulated time
Simulation, Imitation of some system
  continuous
  discrete
  of one computer on another
  of one computer on itself
Singleton cycle of permutation
Skalsky, Michael
SLA(shift left A)
SLAX(shift left AX)
SLC(shift left AX circularly)
SLIP
Smallest-in-first-out
SNOBOL
Solitaire(patience)game
Son, in a tree structure
Sorting
  topological
Sparse matrix
Speedcoding
SRA(shift right A)
SRAX(shift right AX)
SRC(shift right AX circularly)
STA(store A)
Stack
  deletion("poping")
  insertion("pushing")
  linked allocation
  pointer to
  sequential allocation
Standard deviation of probability distribution: The square root of the variance, an indication of how much a random quantity may be expected to deviate from its mean value
Stearns, Richard Edwin
Stegun, Irene Anne
Stevenson, Francis Robert
Stickelberger, Ludwig
Stigler, Stephen Mack
Stirling, James
  approximation
Stirling numbers
  combinatorial interpretations
  generating functions
  tables of
STJ(store J)
Storage allocation: Choosing memory cells in which to store data, see Available space list, Dynamic storage allocation Linked allocation, Sequential allocation
Storage mapping function: The function whose value is the location of an array node, given the indi-

SIMULA语言
模拟时间
模拟：某个系统的模仿
  连续的模拟
  离散的模拟
  在另一台计算机上模拟一台计算机
  在一台计算机本身上模拟一台计算机
排列的单个循环
斯卡尔斯基，迈克尔
SLA指令（左移A）
SLAX（左移AX）
SLC指令（循环左移AX）
SLIP程序设计语言
最小的进先出
SNOBOL语言
单人（独玩）游戏
在一个树结构中的儿子
分类
  拓扑分类
稀疏矩阵
快速编码
SRA指令（右移A）
SRAX指令（右移AX）
SRC指令（循环右移AX）
STA指令（存储A）
堆栈
  堆栈的删去（"弹出"）
  堆栈的插入（"下推"）
  堆栈的链接分配
  堆栈的指针
  堆栈的顺序分配
概率分布的标准离差，方差的平方根，给出一个随机变量对它的均值可以有多大偏离的一个标志
斯特恩斯，理查德·埃德温
斯特根，艾琳·安妮
史蒂文森，弗朗西斯·罗伯特
斯蒂克勒贝格，路德维格
施蒂格勒，斯蒂芬·麦克
斯特林，詹姆斯
  斯特林近似公式
斯特林数
  斯特林数的组合解释
  斯特林数的生成函数
  斯特林数表
STJ指令（存储J）
存储分配：选择用于存储数据的存储单元，见可利用的空间表，动态存储分配，链接分配，顺序分配。
存储映象函数：给定节点的指标，其值为此数组节点的单元的函数

ces of that node
Store, British word for "memory" 存储，英文称作 memory
Storing operators of MIX MIX的存储操作符
Straight linkage 直接链接
String, A finite sequence of zero or more symbols, see Linear list 串，0个或多个符号的有限序列，见线性表
concatenation 串的连接
manipulation 串的处理
Strongly connected directed graph 强连通的有向图形
Structure, how to represent, see Representation 如何表示结构，见表示
Stuart, Alan 斯图尔特，艾伦
STX (store X) STX指令（存储X）
STZ (store zero) STZ指令（存储零）
ST1 (store 1) ST1指令（存储1）
SUB (subtract) SUB指令（减法）
Subroutine 子程序
allocation 子程序的分配
closed, see Subroutine 闭子程序，见子程序
history 子程序的历史
linkage 子程序链接
open, see Macro instruction 开子程序，见宏指令
Subscript see Index 下标，见I字首
Substitution operation 替代运算
Subtree order 子树的次序
Subtrees 子树
enumeration of 子树的枚举
free (spanning) 自由（遍及的）子树
Summation 求和
by part 部分求和
Euler's formula 欧拉求和公式
interchange of order 交换求和次序
of arithmetic progression 算术级数求和
of binomial coefficients 二项式系数求和
of geometric progression 几何级数求和
relation to integration 求和同整数的关系
Swapping buffers 转储缓冲
Swift, Charles James 斯威夫特，查尔斯·詹姆斯
Swift, Jonathan 斯威夫特，乔纳森
Switching table 开关表
Sylvester, James Joseph 西尔威斯特，詹姆斯·约瑟夫
Symbol manipulation, A general term for data processing, usually applied to nonnumeric processes such as manipulation of strings or algebraic formulas 符号处理，用于数据处理的一般术语，通常可应用于诸如串或代数公式的非数值处理
Symbol table algorithms 符号表算法
Symbolic machine language see Assembly language 符号机器语言，见语言
Symmetric function, elementary 初等对称函数
Symmetric order for binary tree 二叉树的对称性顺
Inorder 中根次序
Symmetric relation 对称关系
Synchronous discrete simulation 同步离散模拟
Syntactical algorithms 语法算法
System, A set of objects or processes which are interconnected or which interact with each other 系统，彼相互连接或彼此相互作用的对象或处理的集合

System/360 系统/360
Szekeres, George 塞克特斯，乔治
Szpilrajn, Edward 斯皮尔拉因，爱德华
Table-driven program, see Interpreter, Switching table 驱动表格程序，见解释程序，开关表格
Tables, arrangement of, inside a computer, see Representation 表格，一台计算机内的排列，见表示
Tables of numerical quantities 数值量表
Tag field in tree node, see LTAG, RTAG 树形节点中的加标场，见LTAG, RTAG
Tape 带
Taussky, Olga 陶斯基，奥尔加
Tautology 重言式
Taylor, Brook, formula with remainder 台劳，布鲁克带余项的公式
Temp storage; Part of memory used to hold a value for a comparatively short time while other values occupy the registers 临时存储：在其它值占据寄存器时，用来保持一个值于相对地较短的时间
Terminal node of tree 树的终端节点
Terminology 术语
Ternary tree 三进树
Tetrahedral array, see Binomial number system 四面体数组，见二项式数系
Theory of automata 自动机理论
Theory of algorithms 算法论
Thiele, Thorvald Nicolai 蒂利，索瓦尔德·尼古拉
Thorelli, Lars-Erik 托雷利，拉斯·埃里克
Thread an unthread tree 对一未穿线的树进行穿线
Thread links 穿线链接
Thread trees, 穿线的树
  compared to unthread 穿线树同未穿线的加以比较
  insertion into 插入到穿线树
  list head in 穿线树中的表头
Three-address code 三地址代码
Tiling the plane 砌平面
Time, simulated 模拟时间
Time taken by program, see Execution time 程序花费的时间，见执行时间
Timer, see Clock 计时器，见时钟
Todd, John 托德，约翰
Todd, Olga Taussky 托德，奥尔加·陶斯基
Tonge, Frederic McLanahan 汤格，弗雷德里克·麦克拉纳汉
Top of stack 堆栈的顶
Top down process 由顶向下的处理
Topological sorting 拓扑分类
Torelli, Gabriele 托勒里，加布里埃尔
Totient function Φ(n) 计数函数Φ(n)
Trace routine 跟踪程序
Traffic signal 交通信号
Transfer instruction; A "jump" instruction 转移指令；一个"跳转"指令
Transitive relation, see Ordering 传递关系，见次序
Transpose of matrix 矩阵的转置
Traversal of tree structure 树结构的遍历
Tree function, evaluation of 树函数的计算
Trees 树
  binary, see Binary trees 二叉树，见B字首
  comparison of different types 不同类型树的比较

complete t-ary 完备的 t 进树
construction of 树的构造
copying of 树的复写
definition of 树的定义
deletion from 从树中删去
Dewey notation for 树的杜威记号
diagrams of 树的图式
embedding of 树的嵌入
enumeration of 树的枚举
equivalent 等价树
erasing of 抹去树
free, see Free trees 自由树，见F字首，
history 树的历史
index notation for 树的下标记号
infinite 无穷树
insertion into 插入到树
labeled, enumeration of 带标号树的枚举
linear ordering for 树的线性次序
linked allocation for 树的链接分配
mathematical theory of 树的数学理论
m-tuply rooted, see Forest m 重有根的树，见森林
ordered, see Trees 有序树，见树
oriented, see Oriented trees 有向树，见O字首
representation of 树的表示
right-thread 右穿线的树
sequential allocation for 树的顺序分配
similar 相似树
t-ary t 进树
ternary 三进树
threaded, see Threaded trees 穿线的树，见T字首
traversal of 树的遍历
triply linked 三重链接
unordered, see Oriented trees 无序树，见有向树
unrooted, see Free trees 无根树，见自由树
Triangular matrix 三角矩阵
Tridiagonal matrix 三对角矩阵
Trigonometric functions 三角函数
Trilling, Laurent 特里林，劳伦特
Triply linked tree 三重链接的树形
Tritter, Alan Levi 特里特，艾伦·利瓦伊
Turing, Alan Mathison 图林，艾伦·马西森
machine 图林机
Twain, Mark (= Clemens, Samuel Langhorne) 马克，吐温（=克莱门斯，塞缪尔·兰格恩）
Two-way linkage 双向链接
Uhler, Horace Scudder 尤勒，霍勒斯·斯卡德
UNDERFLOW 下溢标志
Uniform distribution, A probability distribution in which every value is equal probable 均匀分布，每个值都有同样可能性的一个概率分布
UNIVAC 1 UNIVAC 1计算机
UNIVAC 3 UNIVAC 3计算机
UNIVAC 1107 UNIVAC 1107计算机
UNIVAC SS80 UNIVAC SS80计算机
Unpacking 散开

Update-memory 现代化存储
van Aardenne Ehrenfest, Tatiana 范·阿登尼-埃伦费斯特，塔尼亚纳
van der Waerden, Bartel Leendert 范·德·瓦尔登，伯特尔·利恩德特
Vandermonde, Alexandre Théophile 范特蒙德，阿历山大·西奥菲尔
Varga, Richard Steven 瓦尔加，理卡德·史蒂文
Variable, A quantity in a program which may posses different values as the calculation proceeds 变量：在一个程序中，随着计算的进行，可以具有不同的值的量
link or pointer 链或或指针变量
Variable-size *nodes* 可变大小的节点
Variance of a probability distribution 一个概率分布的方差
Vauvenargues, Luc de Clapiers marquis de 沃维纳格斯，卢克·德·克拉皮尔斯·马奎斯·德
Vector, see Linear Lists 向量，见线性列表
Vertex in a graph 一个图形中的顶点
isolated 孤立顶点
Victorius of Aquitania 阿基坦的维克托里尤斯
Virtual machine 虚拟机器
Visit a node 访问一个节点
von Ettingshausen, Andreas 冯·埃廷肖欣，安德烈亚斯
von Neumann, John 冯·诺依曼，约翰
von Staudt, Karl Georg Christian 冯·斯托特，卡尔·乔治·克里斯琴
W value(in MIXAL) (MIXAL中的) W-值
Wait list, see Agenda 等候表，见议事日程（A字首）
Waite, William McCastline 韦特，威廉·麦卡斯特林
Wallis, John 沃里斯，约翰
product 沃里斯乘积
Wang, Hao 王浩
Waring, Edward 华林，爱德华
Warren, Don W. 沃伦，唐·W
Watson, Rev. Henry William 沃森，贾维·亨利·威廉
Webster, Noah, dictionary 韦伯斯特，诺亚词典
Wegner, Peter 韦格纳，彼得
Weierstrass, Karl, theorem 维斯特拉斯，卡尔定理
Weight of vertex in free tree 自由树中顶点的权
Weighted path length 带权的通路长度
Weiland, Richard Joel 韦兰，理查德·乔尔
Weizenbaum, Joseph 韦森鲍姆，约瑟夫
Well-ordering 良序
Wheeler, David John 惠勒，戴维·约翰
Whinihan, Michael James 惠尼罕，迈克尔·詹姆斯
Whirlwind I 沃尔温德
Whitworth, William Allen 惠特沃思，威廉·艾伦
Wilkes, Maurice Vincent 威尔克斯，莫里斯·文森特
Wilson, Sir John, theorem 威尔逊，约翰爵士定理
Windley, P. F. 温德利，P. F
Wire length, minimum 极小的导线长度
Wirth, Niklaus Emil 沃思，尼克劳斯·埃米尔
Wolman, Eric 沃尔曼，埃里克
Wolontis, Vidar Michael 沃伦提斯，维达尔·迈克尔
Woodger, Michael 伍杰，迈克尔
Woods, M. L. 伍兹，M. L.
Woodward, Philip Mayre 伍德沃德，菲利普·梅恩
Word; Addressable unit of computer memory 字；计算机存储器的可寻址的单元
Word size, for MIX, The number of different values MIX的字的大小：可以存入五个字节中的不同值的个数

that might be stored in five bytes
Wordsworth, William 沃德沃思，威廉
Worst-fit method of storage allocation 存储分配的"最坏的适合"的方法
Wrench, John William, Jr 小伦奇，约翰·威廉
Wright, Edward Maitland 赖特，爱德华·梅特兰
Wright, Jesse Bowdle 赖特，杰西·鲍德尔
Writing; Doing output 写；进行输出
Writing large programs 写大型程序
X-register X寄存器
XDS 920 XDS 920计算机
XOR(exclusive) XOR指令（排斥的或）
Yngve, Victor Hesse 英韦，维克托·海塞
Yo-yo list 啃啃表
Youden, William Wallace 尤登，威廉·华莱士
Young, Rosalind Cecily Hildegard 扬，罗沙琳德·西西莉·希尔德加德
Zeckendorf, Edouard 泽肯多夫
Zeta function 泽塔函数
Zimmerman, Seth 齐默尔曼，塞思