计算机程序设计技巧

第三卷 排序和查找

(美) D.E.克努特 著
曹胜文 高连霞 译
闫振群 等校

JISUANJI C GY

制作信息和申明

图书名称：计算机程序设计技巧（第三卷 排序和查找）

The Art of Computer Programming（Vol.3 Sorting and Searching）

作者：D.E.Knuth

译者：管纪文 苏运霖 译　　审校：陆汝钤 校

出版：国防工业出版社（Addison Wesley 国际出版公司授权）

制作版本：1.1

制作时间：2001 年 11 月 25 日

制作形式：pdf 文件

制作声明：

本电子文档及相关文档，只为本人私人阅读方便而制作，以及极小范围作教学交流使用，不供网路传播。如果不慎流传，并有人通过某些特殊途径偶然获取到本文档，为尊重知识产权，特此敬告：版权读物，如需要请购买正式印刷版，本电子文档请勿保留，立即删除，更不要在网路传播！

否则，凡有未经通知本人、未经同意许可，私自传播者，一概跟本人无关，自负其责，特此申明。

联系方式：BookFTP.bbs@bbs.nju.edu.cn or BookFTP@21cn.com

73.87221
257
3:2

计算机程序设计技巧

（第三卷　排序和查找）

〔美〕D.E.克努特　著
管纪文　苏运霖　译
陆汝钤　等　校

国防工业出版社
1986.8.1

内容简介

《计算机程序设计技巧》是计算机科学方面的一部重要著作，受到了国际上有关方面的普遍重视。

本书共分七卷。主题是"算法分析"，即特定的计算机算法性质的理论。全书详尽地介绍了若干"非数值分析"领域中大量采用的各种优秀方法和技巧，并从数学上给予了严格的论证和比较分析。书中附有大量习题和答案，标明了难易程度及数学概念的使用。这部较完整的书，可供从事计算机科学、计算数学、计算技术诸方面的工作人员参考、研究和借鉴，也可作为高等院校有关专业的一部基本的教学参考书。

本卷是全书的第三卷——《排序和查找》，分"排序"和"查找"两章。这是无论在系统软件中还是在应用软件中都要遇到的两个极其重要的问题。本卷详细评价了在这两方面现有的重要技术或算法，指出了使用各种技术的条件，理论与实践并重。可作为大型信息管理系统、数据库组织、算法复杂性分析等课程的教科书或教学参考书。

THE ART OF COMPUTER PROGRAMMING
Volume 3/Sorting and Searching
D.E.Knuth
1973 by Addison-Wesley Publishing Company, Inc.

*

计算机程序设计技巧
(第三卷　排序和查找)
〔美〕D.E.克努特　著
管纪文　苏运霖　译
陆汝钤　等　校

*

国防工业出版社　出版
新华书店北京发行所发行　各地新华书店经售
国防工业出版社印刷厂印装

*

787×1092 1/16　印张 41 1/8　957千字
1984年8月第一版　1984年8月第一次印刷　印数：00,001—20,000册
统一书号：15034·2606　定价：5.85元

译 者 序

经过反复修改、润饰加工，D. E. 克努特的长篇巨著《计算机程序设计技巧》第三卷中译本终于问世了。我们高兴地向广大读者，特别是计算机科学工作者，大专院校计算机专业的师生，以及从事与计算机应用有关的工作的同志们，推荐这本书。

这一卷集中讨论两个问题：排序和查找。这两个问题，无论在系统软件中，还是在应用软件中，都是要遇到的极其重要的课题。特别是对于数据库的维护和从数据库查找信息，具有直接的重要性。作者阐述、讨论和分析了排序和查找中所使用的较为重要的各种算法，有利于读者掌握这方面的技术发展水平。

这里有必要说明，我们翻译时所使用的蓝本，是爱迪生·韦斯利出版社（Addison-Wesley Publishing Company. Inc）1973 年出版的版本。后来作者向我们提供了两份更正和补遗。一份是 1976 年作的，一份是 1978 年作的。我们根据这两份材料，对内容进行了处置：按作者所提出的更正，改正了书中的错误；在相应的位置添加了作者所提出的补遗。最近，我们又收到作者提供的 1981 年的更正和补遗。由于时间关系，我们未能按这份更正和补遗重新加工全书。但对于一些重要的改动，尤其是更正部分，则全都吸收进来。这样就使本书实际上成了 1981 年的最新版本。所以我们也可以说，虽然本书的出版拖了一些时间，却并未因此而使本书过时。

关于译文的术语问题，由于国内对计算机的术语正力求统一，但是仍有一些还未达到一致，我们力求采用已被普遍接受的术语。而对于还未有统一译法的术语，则依据我们的理解来给出译名，不拘泥于已有的一些译法。这样做，并非想标新立异，而只是想尽我们之所能，更加准确地反映著者的原义。因此，同一个词，在不同时候，对于不同问题，可能采用不同的译法。如 uniform 一词，在涉及排序时，译为“一致”；而在涉及查找和杂凑时，则译为“均匀”。又如 binary，当同 search 相连时，译为“二分查找”；同 tree 相连，译为“二叉树”；同 search tree 相连，译为“二分查找树”，等等。另外，为了顺口，我们把“最大(小)者进先出”译为“最大（小）者先出”。原书中规定，$\log_2=\lg$，在书中两者混杂出现，为求统一，我们全部把它改为 $\log_2$。

在翻译过程中，一直得到电子工业部计算机工业管理局、中国科学院北京计算技术研究所、数学研究所、华北计算技术研究所等单位一些同志的热情鼓励和支持。数学研究所的陆汝钤同志主校了全书，周龙骧、陆维明、赵立平同志也分别承担了本书的部分校对工作。谨向这些同志表示衷心的感谢。

由于译者水平所限，译文中肯定难免还有缺点或错误。我们欢迎广大读者给以批评指正。

一九八二年八月

38335

前　言

烹调法成了一种艺术，一门高深的科学；厨师是有教养的人。
——泰特斯·利维尤斯（TITUS LIVIUS），*Ab Urbe Condita* XXXIX. vi
（罗伯特·伯顿(Robert Burton)，*Anatomy of Melancholy* 1.2.2.2）

本书形成了第二章关于信息结构内容的自然续篇，因为它把线性有序数据的概念加入到别的基本结构的思想中去了。本书书名“排序和查找”可能给人以印象，仿佛它仅仅是为那些关心通用排序程序编制或信息检索应用的系统程序员编写的。然而，事实上排序和查找的领域还提供了一个理想的园地，以讨论范围广泛而且很重要的一般性问题：

如何发现好的算法？

如何改进给定的算法或程序？

如何从数学上分析算法的效率？

对于相同的应用，人们如何在不同的算法之间进行合理的选择？

在什么意义之下，可以证明某个算法是“最好”的？

计算理论同实际考虑如何相互影响？

象带、鼓或盘这样的外存，如何能有效地用于大型数据库？

其实，我确信，程序设计的每一个重要方面，实际上都离不开排序或查找！

本卷由全套书的第五章和第六章组成。第五章讨论的是排序；这是一个相当大的课题，已经把它主要划分成两大部分，即内部排序和外部排序。还有补充的几节，提出了关于排列（5.1节）和关于最优排序算法（5.3节）的辅助理论。第六章讨论了在表或文件中查找特定项目的问题；这又细分成顺序查找、通过比较键查找、按数字性质查找、“杂凑”查找等几种方法，而后考虑了比较困难的辅键查找的问题。在这两章之间存在着令人惊奇的相互影响以及强烈的类似，从而把这些课题联系在一起。除了在第二章中所讨论的信息结构以外，这里还讨论了信息结构的两种重要的变形，即优先队（5.2.3节）和表示成平衡树的线性表（6.2.3节）。

建议不熟悉本套书第一卷的读者去参看一下记号索引（附录B），因为这里的某些记号约定，不同于一般的用法。

我已经使用本书作为大学三年级至研究生程度的数据结构方面的第二门课程的教材，但省略了大部分数学内容。本书的数学部分可作为大学四年级或研究生算法分析课程的适当教材，特别要着重5.1、5.2.2、6.3和6.4节。关于计算复杂性的研究生课程，也可以以4.3.3、4.6.3、4.6.4、5.3和5.4.4节作为基础。

计算机科学的飞速发展，已经使本书的出版推迟了几乎三年之久。这是因为，排序和查找的许多方面已经高度地发展起来了。我要特别感激国家科学基金会、海军研究所、国防分析研究所、国际商用机器公司以及挪威全国科学研究委员会给与我连续不断的研究

支持。

许多人对于我筹备这一卷的出版给予了帮助，特别是爱德华·A.本德（Edward A. Bender）、克拉克·A.克兰(Clark A. Crane)、戴维·E.弗格森(David E. Ferguson)、罗伯特·W.弗洛伊德（Robert W. Floyd)、罗纳德·L.格雷厄姆（Ronald L. Graham)、利奥尼达斯·吉巴斯（Leonidas Guibas)、约翰·霍普克洛弗特（John Hopcroft)、理查德·M.卡普（Richard M. Karp)、加里·D.克诺特（Gary D. Knott)、鲁道夫·A.克鲁塔尔（Rudolph A. Krutar)、申林（Shen Lin)、沃恩·R.普拉特（Vaughan R. Pratt)、斯蒂芬·O.里斯（Stephan O. Rice)、理查德·P.斯坦利（Richard P. Stanley)、J. A. 范·德·普尔（J. A. Van der Pool）及小约翰·W.伦奇（John W. Wrench, Jr.)。此外，还有斯坦福和伯克利的学生们，他们曾校对了原稿。

D. E. 克努特　1972年9月

于挪威奥斯陆

一位作家享有某些公认的特权，我认为，这件事的好处没有理由加以怀疑；特别是，在我还不被人们理解的地方，后面蕴涵着某种非常有用和意义深远的东西。

——乔纳坦·斯威夫特(JONATHAN SWIFT)

(*Tale of a Tub*, Preface, 1704)

关于习题的说明

这部书的习题既适合于作课堂作业，也可用于自学。任何人只想通过单纯的阅读，而不把所学的知识应用于特定问题，并鼓励自己对已经阅读过的东西反复进行思考，就想掌握一门学问，如果说不是不可能的，那也是困难的。其次，我们自己发现的东西，一定也是我们学得最好的东西。因此，习题就成为这本著作的主要部分之一；我力求使这些习题尽可能地丰富多采，并尽量选择解决起来有趣味的问题。

在许多书里，容易的习题随机地混杂在非常难的习题当中。这有时是不合适的，因为读者在认真着手一个问题之前，首先要想到需要花多少时间去解决它（否则他可能干脆跳过所有的问题）。这种情况的一个典型例子是理查德·贝尔曼（Richard Bellman）所著的"Dynamic Programming"一书；这是一本重要的开创性的书，其中，在若干章的末尾，在"习题和研究题"的标题下，把一组问题收集在一起，使极为浅显的问题混杂在深奥的还未解决的问题当中。据说曾有人问过贝尔曼博士怎样把习题同研究题区分开，他回答说："如果你能解决，它就是一个习题；否则它就是一个研究题。"

在这一类型的书里同时包括研究题和非常容易的习题，是有充分道理的。因此，为使读者免于陷入确定哪些是习题哪些是研究题的困境，我特意提供了分数以指出难易的程度。这些分数的一般意义如下：

分数	解释
00	一个极其容易的习题，如果已经理解了书上的材料，就能立即回答它，而且几乎总是可以只用心算，不必动手。
10	一个简单的问题，它要你想一想刚刚读过的材料，但它决不是难题，至多在一分钟之内就可能做出。为了得到解答，可能要用纸和笔。
20	一个一般水平的题目，它测验你对课文内容的理解程度，但大约化上十五到二十分钟的时间，就可能得到完全的答案。
30	一个中等难度和/或复杂程度的问题，可能需要两个小时以上的工作才能满意地解决。
40	确实是一个困难或冗长的问题，在学校里，它或许适合于当作学期论文的题目。预计一个学生要用相当长的时间，才能解决这个问题，但解答是并不显然的。
50	一个研究题，它（就作者在写书时所知）是尚未被满意地解决的问题。如果读者已经发现了这个问题的答案，就请你把它写出来加以发表；而且，本书的作者将乐于尽可能快地获知关于这个解答的消息（假定它是正确的）！

对这些分数进行"对数的"比例插值，其它分数的意义就明白了。例如，分数 17 表示一个比一般水平的题目要简单的题目。分数为 50 的问题，如果随后被某个读者解决了，则在本书再版时就可能变成 45 分。

作者力求认真地评定准确的分数。但是提出问题的人要确切地知道这个问题对于另外一个人的难度如何，确实是有困难的；而且每个人有可能较其他人更适应于某种类型的问题。希望分数能成为表示难易程度的一个好的估计，但它只应被当作一般的准绳，而不应被看作绝对的指示。

本书是为具有不同程度的数学训练和素养的读者撰写的。因而，有些习题是仅供数学水平较高的读者利用的。因此，如果习题所涉及的数学概念或出题动机已经超过了仅关心程序设计算法的读者所需要的程度，则在分数前写上M。如果习题的解答需要有微积分或者本书中未提供的其它高等数学的知识，习题就标上字母“HM”，“HM”的标示并不一定意味着困难。

有些习题以一个黑箭头“▶”打头；这表示这些问题是特别有益和特别值得推荐的。当然，并不期望读者或学生来做所有的习题，所以才把那些也许是最有价值的标示出来。这并不意味着贬低其它的习题，每个读者至少应当试图解决所有分数为10以下的那些习题；而且箭头可以帮助你决定哪些具有较高分数的问题应该优先考虑。

大多数习题的解答已在答案部分中给出。请明智地使用它们；在你未独立地作出真正的努力来解决问题之前，请不要翻阅答案，除非你没有时间来做这个具体的问题。在你得到了自己的解答或者对问题进行了认真的努力后，你可能会感到答案是有益的和有帮助的。给出的答案通常是十分简短的，而且假定你自己事先已经进行了认真的努力，因而已将细节略去。有时解答给出的信息比想要的少；但通常都要多给一些。你的答案很可能比我们这里给出的要更好些，或者你可能发现在发表的解答中有某些错误；在这样的情况下，作者希望尽快地知道详细的情节。在本书再版时，作者将把改进了的答案，连同解答者的名字一起在适当的地方给出来。

当做一个习题时，如未明文禁止，一般地你可以利用前面习题的答案。我在指定分数时已经考虑到这一点了；因此，尽管习题$n+1$把习题n的结果作为特例，习题$n+1$的分数有可能要比习题n的分数低。

代码一览表

		00	即刻可解的
		10	简单的（一分钟）
		20	一般水平的（一刻钟）
▶	推荐的	30	中等难度的
M	面向数学的	40	学期论文
HM	需要“高等数学”	50	研究题

习题

▶1.〔00〕 分数“$M20$”意味着什么？

2.〔10〕 一本教科书中的习题对读者有什么价值？

3.〔$M50$〕 证明当n是一个整数且$n>2$时，方程$x^n+y^n=z^n$无正整数x、y、z的解。

每天两小时的练习……将足以保持适合于他的工作的一个速度。

——M. H. 马洪（M. H. MAHOH）“*The Handy Horse Book*”(Edinburgh, 1865)

目　录

第5章　排　序

*5.1　排列的组合性质 …… 9
*5.1.1　反序 …… 9
*5.1.2　多重集合的排列 …… 18
*5.1.3　路段 …… 28
*5.1.4　图表和对合 …… 39
5.2　内部排序 …… 59
5.2.1　通过插入进行排序 …… 65
5.2.2　通过交换进行排序 …… 85
5.2.3　通过选择进行排序 …… 114
5.2.4　通过合并进行排序 …… 130
5.2.5　通过分布进行排序 …… 139
5.3　最优排序 …… 147
5.3.1　极少比较排序 …… 148
*5.3.2　极少比较合并 …… 162
*5.3.3　极少比较选择 …… 171
*5.3.4　排序网络 …… 181
5.4　外部排序 …… 204
5.4.1　多路合并和替代选择 …… 207
5.4.2　多阶段合并 …… 220
5.4.3　级联合并 …… 237
5.4.4　向后读带 …… 248
5.4.5　交替排序 …… 258
5.4.6　关于带合并的实际考虑 …… 263
*5.4.7　外部基数排序 …… 285
*5.4.8　双带排序 …… 290
5.4.9　磁盘和磁鼓 …… 297
5.5　小结、历史和文献目录 …… 311

第6章　查　找

6.1　顺序查找 …… 324
6.2　通过键比较进行查找 …… 334
6.2.1　查找一个有序的表 …… 334
6.2.2　二叉树查找 …… 348
6.2.3　平衡的树 …… 373
6.2.4　多路树 …… 391
6.3　数字查找 …… 398
6.4　杂凑 …… 420
6.5　利用辅助键的检索 …… 456
习题答案 …… 475
附录A　数值量表 …… 611
1. 基本常数(十进制) …… 611
2. 基本常数(八进制) …… 612
3. 调和数、贝努利数、斐波那契数 …… 613
附录B　记号索引 …… 615
名词和姓名中英对照表 …… 619

第5章 排　　序

没有什么比带头建立事物的新秩序更难掌握，更担风险，更成败莫测的了。

——尼科罗·麦克谢尔里（NICCOLÔ MACHIAVELLI)(The Prince, 1513)

"但是你不能及时地查出全部那些执照号"德雷克争辩说。

"没有必要，保罗。我们只要列一张表并把重复的找出来"。

——佩里·梅森（PERRY MASON)(The Case of the Angry Mourner, 1951)

"树排序"计算机——应用这种新的'计算机方法'来进行种类研究，你就能快速地确认260种以上美国、阿拉斯加和加拿大的不同的树种，甚至棕榈、沙漠的树，以及其它外来树种。

要想排序，你只须插入这个针就行了。

——"Catalog of Edmund Scientific Company"（1964）

在这一章里，我们将研究在程序设计中经常出现的一个课题：从递增或递减的次序重新排列条款。我们设想，倘若字典中的词不是以字母的顺序排列，那么使用这样的字典将是何等困难；同样地，存在计算机存储器中的条款的次序，对于处理这些条款的算法的速度及简便性，也经常有深远的影响。

尽管英语词典里把"sorting"一词定义为按照种类或类型分开或安排事物的过程，但是计算机程序员习惯于在更为特殊的意义下来使用它，即把事物排成递增或递降的次序。这过程也许应称作ordering，而不是sorting；但任何试图把它称为"ordering"的人很快就会导致混乱，因为这个词已被附加了许多不同的意义。例如，考虑下边的句子："由于我们只有两台磁带机处于工作状态（working order），我被要求（was ordered）去以快速订货的形式（in short order）订购（order）更多的磁带机，以便（in order）以快若干数量级（order）的速度对数据进行排序（order)"。在数学术语中，"order"的意义太多了〔群的阶（order），置换的阶（order），分支点的级数（order），次序（order）的关系，等等〕。所以我们发现，"order"一词可能导致混乱。

某些人建议使用"排顺序"（sequencing）这个词作为排序过程的适当名称；但是这个词通常缺乏恰当的涵义，特别是当出现相等的元素时。而且，它有时同其它术语冲突。我们承认"Sorting"本身也是一个被滥用了的词〔"在对那一类（sort）数据进行排序sorting）以后，他有几分(sort of)不快（out of sorts)。"〕，但它已经在计算的用语中确定地建立起来了。因此，我们将把"排序"（sorting)一词主要地用于"排成有序"的严格意义之下，而不去作进一步的辩解。

排序的某些最重要的应用是：

a）解决"一起性"的问题，即把具有相同标识的所有条款联在一起。假设我们有在随机次序下的10,000个条款，其中许多有相等的值；而且假设我们要重新排列这个文件，使得

具有相等值的所有条款出现在相邻的位置上。这实际上是在该词的旧的意义下的“sorting”问题；通过在该词的新意义下对文件排序，可以容易地解决这个问题，使得这些值按递增的次序 $v_1 \leqslant v_2 \cdots \leqslant v_{10,000}$ 排好（在这个过程中可能达到的效率说明了为什么“sorting”原来的意义已经改变）。

b）如果两个或更多个文件已经排成相同的次序，则有可能通过顺序地对它们扫描一趟，从中找出所有匹配的条款，而无须返回去查。这是佩里·梅森（Perry Mason）用来帮助解决一个谋杀案件的原理（见本章开头的引用语）。从开始到末尾顺序地存取一个信息表，而不是在表中随机地来回跳动，通常是经济得多的，除非这个表很小，足以存入到高速随机存取的存储器中。排序使得有可能对大型文件使用顺序存取，从而有可能取代直接寻址。

c）如同我们将在第六章中要见到的，排序也有助于查找，因此它也有助于使计算机的输出更适合人们的需要。事实上，按字母顺序排序的一份清单，往往看上去十分可信，即使有关的数字信息计算得不正确也是如此。

尽管排序传统上大多用于商业数据处理，但它实际上是程序员应当掌握的运用广泛的一项基本工具。在习题 2.3.2-17 中，我们已经讨论了它在简化代数公式中的用途，以下的习题说明各种各样的典型应用。

展示排序多样性最初的大型软件系统之一，是计算机科学公司 1960 年研制的拉克（Larc）科学编译程序。这个用于扩充的 FORTRAN 语言的优化编译程序，大量地使用了排序，使得各种编译算法同源程序的有关部分都以适当的顺序出现。头一趟是一个词法扫描，它把 FORTRAN 源代码分成单个的记号，每个表示一个标识符（例如变量名称），或者一个常数，或者一个操作符等等。每个记号被赋以若干顺序号；当对名字和适当的顺序号进行排序时，一个给定标识符的所有使用就联系在一起了。例如，确定标识符是否为一个函数名、参数或数组变量的“定义条款”，被给予一个低的顺序号，以使它们在对应于一个给定标识符的诸记号当中首先出现；这样，既便于校验一个标识符的使用是否有冲突，又便于相对于 EQUIVALENCE 说明符来分配存储，等等。以这种方式收集在一起的每个标识符的信息现在附加在每个记号上；这样在高速存储器中就不需要保留标识符的“符号表”了。更新过的记号然后按另一个序列号排序，它基本上是把源程序恢复到它原来的顺序，只是该序列是特殊设计的，它把算术表达式变成为更方便的“波兰前缀”形式。排序也用于编译的后期，以便于进行循环优化，把错误信息合并到清单当中，等等。简言之，编译程序被设计成，使得实际上可以通过顺序地处理存于辅助鼓内的文件而完成工作，因为适当的顺序号，是以这样一种方式附加在数据上的，即数据能被排成各种方便的序列。

排序的另一个更明显的应用出现于文件编辑程序中，其中每行以一个键号加以标识。当一个用户打入附加的信息和修改时，在存储器中不需要有整个文件。改变的行可以随后进行排序（它们通常是在相当程度上排成某种次序的），然后同原来的文件合并，这就导致了在多道程序设计情况下使用内存的相当有效的方式〔参考 C. C. Foster，Comp. J. 11 (1968)，134-137〕。

计算机厂家们估计，把他们所有的顾客都考虑在内，在他们的计算机上，运行时间的

百分之二十五以上是花在排序上。有许多计算机装置，排序竟用去计算机时间的一半以上。由这些统计，可以得出结论：或者（i）排序有许多重要的应用，或者（ii）当不应当进行排序时，有许多人却这样干了，或者（iii）低效的排序算法正被普遍地使用着。真实情形可能包括所有三种可能中的某一些。但无论如何，可以看出排序是值得作为一个实际问题而认真地加以研究的。

即使排序几乎无用，无论如何也仍然有许多值得对它进行研究的理由！业已发现的巧妙的算法表明，排序本身就是一个值得剖析的有趣课题，在这方面有许多有魅力的悬而未决的问题（也有不少已经解决了）。

从一个更广泛的方面着眼，我们也会发现，排序算法构成了一般地如何着手解决计算机程序设计问题的有趣的实例研究。我们将说明数据结构操作的许多重要的原理。我们将考察各种排序技术的演化，以图向读者指出，他自己如何可以去发现同样的思想（如果他先于任何其他人碰到了这个问题的话）。通过外推这一实例研究，我们可以学到许多有助于对其它计算机问题设计好算法的有关思想。

排序技术也对包含在算法分析中的一般思想提供了极好的说明，这就是，用来确定算法的性能特征，使得在相匹敌的诸方法之间可以进行巧妙选择的一些思想。专长数学的读者，将在这一章中找到用于估价计算机算法的速度和解决复杂的递归关系的许多有启发的技术。另一方面，已经对内容作了适当的编排，使得那些没有数学爱好的读者也能安稳地跳过这些计算。

在往下进行之前，应该稍微更清楚地来定义我们的问题，并且介绍某些术语。给定有待排序的N个项目

$$R_1,\ R_2,\cdots,\ R_N$$

我们称这些项目为记录，并称N个记录的整个集合为一个文件。每个记录R_i有一个键K_i，它支配着排序的过程。在一个记录中，除这个键外，还可能有另外的信息，这额外的“辅助信息”，除非它属于同一记录，否则对于排序过程没有影响。

在诸键上确定一个次序关系“$<$”，使得对于任意三个键值a、b、c，下列条件成立：

i）$a<b$，$a=b$，$b<a$三个可能性中恰有一个可能性成立（这是三分律）。

ii）如果$a<b$，而且$b<c$，则$a<c$（传递律）。

这两个性质表征了线性次序的数学概念，也称作全序。任何满足（i）和（ii）的关系“$<$”都可以按本章中即将提出的大多数方法进行排序，尽管有些排序技术要求通常次序下的数值或字母键。

排序的目标，是确定记录的一个排列$p(1)p(2)\cdots p(N)$，它以非递降的次序来放置诸键：

$$K_{p(1)}\leqslant K_{p(2)}\leqslant\cdots\leqslant K_{p(N)} \tag{1}$$

如果我们提出进一步要求，即具有相同键的记录保留它们原来的相对次序，即如果

$$p(i)<p(j)\quad 每当K_{p(i)}=K_{p(j)}且i<j时 \tag{2}$$

则该排序说成是稳定的。

在某些情况下我们将要求在存储中对记录进行实际的重新排列，使其键成为有序的，

而在其它情况下，仅仅需要一张辅助表就够了，该表以某种方式指明这个排列，以使这些记录能按其键的次序存取。

少数排序方法假定了值“∞”和“－∞”两者之一或两者的存在性，它们分别地定义为大于或小于所有键：

$$-\infty<K_j<\infty,\quad 1\leqslant j\leqslant N \tag{3}$$

这些值被用作人为的键，也用作哨兵指示器，（3）中排除了相等的情况；如果出现相等，可以对这些算法进行修改，使得它们仍然能够工作，但通常要损失一些优美性和有效性。

排序一般可以分为内部排序和外部排序。在内部排序中，记录被保留在计算机的高速随机存取存储器中；当记录比内存储器一次所能容纳的还多时，就使用外部排序。内部排序在构造和存取数据方面具有更多的灵活性，而外部排序告诉我们如何应付比较严格的存取约束。

利用一个不坏的通用排序算法，对N个记录进行排序所需要的时间，大约同$N\log_2N$成比例；我们对数据大约扫描$\log_2N$“趟”，如同在5.3.1节将看到的，这是至少要花的时间。于是，如果把记录的数目加倍，则将花略多于两倍的时间对它们进行排序，而所有其它事情都相同。（实际上，当N趋于无穷时，如果键不同的话，则为进行排序所需时间的一个更好的估计是$N(\log_2N)^2$，因为键的大小至少以$\log_2N$的速度增长；但对于实用来说，N实际上决不会趋于无穷。）

习题——第一组

1.〔$M20$〕 用三分律和传递律证明，当假定排序稳定时，则排列$p(1)\,p(2)\cdots p(N)$是唯一确定的。

2.〔21〕 假定某文件中的每个记录R_j包含两个键，一个“主键”K_j和一个“次键”k_j，每一个键集合中各定义了一个线性次序$<$，则我们可以用通常的方式在诸键对(K,k)之间定义一个“词典编辑次序”：

$$(K_i,\ k_i)<(K_j,k_j)\quad \text{如果}\ K_i<K_j\ \text{或如果}\ K_i=K_j\ \text{且}\ k_i<k_j$$

一个名叫A的先生取这个文件，首先按主键对它进行排序，得到n组记录，对于每组记录，有相同的主键

$$K_{p(1)}=\cdots=K_{p(i_1)}<K_{p(i_1+1)}=\cdots=K_{p(i_2)}<\cdots<K_{p(i_{n-1}+1)}=\cdots=K_{p(i_n)}$$

其中$i_n=N$。然后，他按各组的次键对n个组$R_{p(i_{j-1}+1)},\cdots,R_{p(i_j)}$中的每一个进行排序。

B先生取同一个原始文件，首先按次键进行排序；然后对得到的文件按主键进行排序。

C先生取同一个原始文件，按主键和次键$(K_j,\ k_j)$的词典编辑顺序对它进行单次排序操作。

问这三个人是否得到相同的结果？

3.〔$M25$〕 设$<$是$K_1,\cdots,K_N$上的一个关系，它满足三分律，但不满足传递律。证明，即使没有传递律，也有可能以一种稳定的方式对记录进行排序，且满足条件（1）和(2)；事实上，至少有三个满足这些条件的排列方式！

4.〔15〕 布·赛·达尔（B. C. Dull）先生（一个MIX程序员）要想知道存在单元A中的数是否大于、小于、或等于存在单元B中的数，所以，他写

LDA A
SUB B

并测试寄存器A是正，是负，或是0，他犯了什么严重的错误？他本应如何做？

5.〔17〕 按下列说明写出多精度键比较的MIX子程序：

调用序列：JMP COMPARE

进入条件：rI1$=n$；CONTENTS (A$+k$)$=a_k$，CONTENTS (B$+k$)$=b_k$，对于 $1\leqslant k\leqslant n$；假定 $n\geqslant 1$

出口条件：CI$=+1$，如果 $(a_n,\cdots,a_1)>(b_n,\cdots,b_1)$；
CI$=0$，如果 $(a_n,\cdots,a_1)=(b_n,\cdots,b_1)$；
CI$=-1$，如果 $(a_n,\cdots,a_1)<(b_n,\cdots,b_1)$；
rX和rI1可能受了影响。

这里，关系 $(a_n,\cdots,a_1)<(b_n,\cdots,b_1)$ 表示从左到右的词典编辑次序；即，对于 $n\geqslant k>j$，有一个下标 j，使得 $a_k=b_k$，而 $a_j<b_j$。

6.〔30〕 单元A和B分别存放两个数 a 和 b。证明，有可能写出一个MIX程序，它计算并保存 $\min(a,\ b)$ 于单元 c 中，而不使用任何转移运算符。（警告：由于你不可能测试是否出现算术溢出，所以确保不论对于什么 a 和 b 的值，都不会出现溢出，是明智的。）

7.〔M27〕 在 N 个独立地一致分布于0和1之间的随机变量已经排为非减顺序之后，这些变量中第 r 个最小者 $\leqslant x$ 的概率是多少？

习题——第二组

下列每一个习题都指出了计算机程序员可能面临的问题。假定仅有较少量的内存可资利用，而以大约6台磁带机作为补充（对于排序说来这些磁带机足够了），试给出解决问题的一个“好”方法。

8.〔15〕 给你一条含有100万个数据字的磁带，你如何确定在该带中有多少不同的数据字？

9.〔18〕 你是美国国内税收服务局，你收到了来自各单位的数百万“信息”表格，报告它们已向人民支付了多少钱；同时，又从人民那里收到了数百万“税收”表格，报告他们已经得到了多少收入。你如何发现没有报告他们所有收入的人？

10.〔M25〕（转置一个矩阵） 给你一条含有一百万字的磁带，这些字代表按行顺序存储的1000×1000阶矩阵的元素：$a_{1,1}$，$a_{1,2}$，…，$a_{1,1000}$，$a_{2,1}$，…$a_{2,1000}$…$a_{1000,1000}$。你如何建立一条带，其中元素是按列的次序$a_{1,1}$ $a_{2,1}$…$a_{1000,1}$ $a_{1,2}$…$a_{1000,2}$…$a_{1000,1000}$存储的？（试试看，至多只对数据扫描10趟。）

11.〔M26〕 给定一个 N 个字的大型文件，你如何把它“洗”成一个随机的排列？

▶12.〔24〕 某大学有约1000名教职员和500个委员会；期望每个教职员至少是两个委员会的成员。你的任务是编制一张打印得很漂亮的各委员会的成员表。为此，以随机的顺序给你大约1500张卡片，穿孔如下：

教职员卡片 列1，空白；列2-18，名字，后边接着若干空白；列19-20，姓的首字母；列21-23，头一个委员会号码；列24-26，第二个委员会号码；…；列78-80，第二十

个委员会号码（如果需要），或空白。

委员会卡片　列1，“*”，列2-77，委员会名称；列78-80，委员会号码。

你怎么办？（稍微详细地述说你的方法。）

13.〔20〕 你正在用两个计算机系统进行工作，它们对字母数字字符的“整理序列”（即顺序关系）有不同的约定。你如何使一台计算机按另一台计算机使用的次序对字母数字文件进行排序？

14.〔18〕 给你一份出生在美国的附有其出生州名的人的名单，名单上的人很多，你如何计算出每个州出生的人数（假设在这个名单中没有人出现一次以上）。

15.〔20〕 为了易于改动大型FORTRAN程序，你需要设计一个“交叉引用”程序；这样一个程序以FORTRAN程序作为输入，并连同一个索引一起把它们打印出来，索引说明程序中每个标识符（即每个名字）的各次使用。你如何来设计这个程序？

16.〔33〕 （图书馆卡片排序） 按字母顺序的目录卡方式随图书馆不同而稍有变化；下面的“按字母顺序的”清单指出了“美国图书馆协会关于目录卡归档的规则”〔the *American Library Association Rules for Filing Catalog Cards* (Chicago, 1942)〕中建议的许多规程：

卡片的正文	注　释
R. Accademia nazionale deilincei, Rome	略去外国的正位（英国除外）
1812; ein historischer roman.	一千八百一十二年
Bibliothèque d'histoire révolutionnaire.	象法语中的空格那样处理撇号
Bibliothèque des curiosités.	略去字母上的重音符号
Brown, Mrs. J. Crosby	略去军阶名称
Brown, John	后边有日期的名字放在后边没有日期的名字之后
Brown, John, mathematician	…后边没有日期的名字按描述性的词来区别
Brown, John. of Boston	
Brown, John, 1715-1766	对同样的名字按生卒年份排列
BROWN, JOHN, 1715-1766	“写谁”放在“谁写”之后
Brown, John, d. 1811	有时生卒年份必须进行估计
Brown, Dr. John, 1810-1882	略去军阶名称
Brown-Williams, Reginald Makepeace	连字号作空格处理
Brown America.	书名紧接着复合名字
Brown & Dallison's Nevada directory.	英文中&变成“and”
Brownjohn, Alan	
Den', Vladimir Éduardovich, 1867-	略去名字中的撇号
The den.	略去开始的冠词
Den Lieben süssen Mädeln.	…假定它处于主格
Dix, Morgan, 1827-1908	名字放在其它字之前
1812 overture.	一千八百一十二年
Le XIXe siécle fran, cais.	十九世纪
The 1847 issue of U. S. stamps.	一千八百四十七年
1812 overture.	一千八百一十二年
I am a mathematician.	（诺伯·维纳语）

卡片的正文	注　释
IBM journal of research and development.	首字母像是一个字母的词
ha-I ha-ehad.	略去开始的冠词
Ia; a love story.	略去题目中的标点符号
International Business Machines Corporation	
al-khuwārizmī, Muhammad ibn Mūsā, *fl*. 813-846	略去阿拉伯名字中开始的"al-"
Labour; a magazine for all workers.	重新写成"Labor"
Labor research association	
Labour, See labor	交叉引用卡片
McCall's cookbook	略去英语中的撇号
McCaarthy, John, 1927-	Mc = Mac
Machine-independent computer programming.	连字符作空格处理
MacMahon, Maj. Percy Alexander, 1854-1929	忽略军阶符号
Mrs. Dalloway.	"Mrs." = "Mistress"
Mistress of mistresses.	
Royal society of London	
St. Petersburger Zeitung.	"St." = "Saint"，甚至在德文中
Saint-Saënf, Camiue, 1835-1921	连字符作空格处理
Ste. Anne des Monts, Quebec	圣
Seminumerical algorithms.	
Uncle Jom's cabin.	
U. S. Bureau of the census.	"U. S." = "United States"
Vandermonde, Alexander Théophile, 1735-1796	
Van Valkenburg, Mac Elwyn, 1921-	略去姓氏中前缀后面的空格
Von Neumann, John, 1903-1957	
The whole art of legerdemain.	
Who's afraid of Virginia Woolf?	略去英语中的撇号
Wijngaarden, Adriaan Van, 1916-	姓不要以小写字母开始

（这些规则大多数都有若干例外，而且还有许多其它规则这里未予说明。）

如果给你用计算机对大量图书馆卡片进行排序的任务，并最终维护这样一个很大的卡片文件，要是你没有机会去改变卡片归档的这些长期有效的规定，那么你应怎样安排数据，使能便于进行排序和合并操作？

17. 〔*M*21〕（离散对数）你知道，p是一个（相当大的）质数，a是一个模p的本原根。因此，对于$1 \leqslant b < p$的所有b，有唯一的n使得$a^n \bmod p = b$，$1 \leqslant n < p$，这个n称作关于a，b modulo p的指数。给定b，你怎样在少于$O(n)$步内找出n？

〔提示：设$m = \lceil \sqrt{p} \rceil$，试对$0 \leqslant n_1$，$n_2 < m$求解$a^{mn_1} \equiv ba^{-n_2}$ (modulo p)。〕

18. 〔*M*25〕〔E. T. 帕克（E. T. Parker）〕欧拉猜测，方程

$$u^6 + v^6 + w^6 + x^6 + y^6 = z^6$$

除开至少有四个变量为0时的显然解外，没有非负整数的解u，v，w，x，y，z。同时，他猜测，对于所有的$n \geqslant 3$

$$x_1^n + \cdots + x_{n-1}^n = x_n^n$$

没有非显然的解。但是这个猜测已被计算机发现的恒等式 $27^5+84^5+110^5+133^5=144^5$ 所否定〔见L. J.兰德（L. J. Lander）、T. R. 帕金（T. R. Parkin）和J. L. 塞尔弗里奇（J. L. Selfridge），Math. Comp. 21(1967)，446-459.〕；你能否想出一个方法，其中排序有助于找到当 $n=6$ 时欧拉猜测的反例？

▶19.〔24〕 给定一个包含大量不同的30位二进字 $x_1, \cdots, x_N$ 的文件，什么是找出其中所有补码对（x_i，x_j）的好方法？（两个字称为互补的，如果其中一个是0的地方另一个是1，且反之亦然；于是，当它们被处理作二进制数时，互补的充要条件是其和为 $(11\cdots1)_2$。

▶20.〔25〕 给定一个含有1000个30位二进字 $x_1, \cdots, x_{1000}$ 的文件，你如何编制一份所有对偶（x_i，x_j）的表，使得除至多两个二进数位外，$x_i=x_j$？

21.〔22〕 你如何去寻找五个字母的变异单词？诸如：CARET，CARTE，CATER，CRATE，REACT，TRACE；CRUEL，LUCRE，ULCER；DOWRY，ROWDY，WORDY。〔除值得注意的集合

APERS，ASPER，PARES，PARSE，PEARS，PRASE，PRESA，RAPES，REAPS，SPARE，SPEAR

（我们也把法文字APRÈS加进去）外，人们可能希望知道是否有包括十个或更多的五字母变异单词的任何集合。〕

22.〔M28〕 给定大量的有向图说明，为了按同构进行分组，可用什么方法？（如果在有向图的顶点之间有一一对应，在它们的弧之间也有一一对应，并保持顶点和弧之间的邻接关系，则有向图称为是同构的。）

23.〔30〕 在4096个人的某人群中，每个人有大约100个相识者。一个文件已经列出所有相识的对偶（关系是对称的，即，如果 x 同 y 相识，则 y 同 x 相识。因此，这文件大约含有200,000个条款）。你怎样设计一个算法，对于给定的 k，列出这群人当中所有 k 个人的集体？（一个集体是互相认识的一组人，集体中每个人都与其他人相识。）假定不存在太大的集体。

24.〔30〕 具有不同名字的三百万人头尾相接地排在一起，从纽约排到加里福尼亚。给每个参加者发一张纸条，在这张纸条上写下他自己的名字，以及紧挨着他西边的那个人的名字。在最西端的那个人不知道怎么做，就把纸条扔了；剩下的2,999,999张纸条放到一个大篮子中，并送到首都华盛顿的国家档案局。在那里篮子中的内容完全"洗"乱了，之后转移到磁带上。

这时，一个信息科学家发现，在磁带上有足够的信息能重新构造原来顺序下人们的名单。而且一个计算机科学家发现了，扫描整个数据带不到1000次，即可进行重新构造的一个方法，此方法只对带文件作顺序存取和使用少量的随机存取存储器。他是怎样做的？

〔换言之，对于 $1\leqslant i<N$ 给定随机次序下的诸对偶（x_i，x_{i+1}），其中，x_i 各不相同，若所有操作限于适合使用磁带的串行技术，如何得到序列 $x_1, x_2, \cdots, x_N$？这是当没有容易的方法来确定两个给定的键中哪一个在前时的排序问题；我们已经把这个问题当作习题2.2.3-25的一部分。〕

*5.1 排列的组合性质

有限集合的一个排列就是把它的元素排成一行的一种排法。在研究排序算法中，排列具有特殊的重要性，因为它们表示了未排序的输入数据。为了研究不同排序方法的有效性，我们要能计算这样的排列数目，它们引起一个排序过程的某一步执行一定的次数。

当然，在第一、二和第三章中我们已有好几次遇到排列了。例如，在 1.2.5 节中，讨论了构造 n 个对象的 $n!$ 个排列的两个基本理论方法；在 1.3.3 节，分析了涉及排列的轮换结构和乘法性质的某些算法；在 3.3.2 节，研究了它们的“上运行”和“下运行”。这一节的目的，是研究排列的某些其它性质，并考虑允许出现相等元素的一般情况。在进行这一研究期间，我们将大量地学习组合数学。

排列的性质就其本性来说是非常有趣的，因而，不把其内容分散在整个一章中，而在一节系统地研究较为合适。对于不专长数学的读者，以及急于埋头研究排序技术的读者，则建议他们直接跳到 5.2 节，因为这一节实际上同排序的直接联系较少。

*5.1.1 反序

设 $a_1, a_2, \cdots, a_n$ 是集合 $\{1, 2, \cdots, n\}$ 的一个排列，如果 $i<j$，且 $a_i>a_j$，则对偶 (a_i, a_j) 称为排列的一个“反序”；例如，排列 3 1 4 2 有三个反序：(3, 1)，(3, 2) 和 (4, 2)。每个反序是“违反排序的”一对元素，所以没有反序的唯一的排列是已排好序的排列 1 2 … n。同排序的这种联系，是我们对反序如此感兴趣的主要原因，尽管我们已经使用过这个概念来分析动态存储分配的算法（见习题2.2.2-9）。

反序的概念是G. 克拉默(G. Cramer) 于 1750 年在研究他的解线性方程组的著名规则时引进的〔*Intr. à l' Analyse des Lignes Courbes algébriques* (Geneva, 1750), 657-659; 参考 Thomas Muir, *Theory of Determinants* I (1906), 11-14〕。实质上，他以下列方式定义了 $n\times n$ 阶矩阵的行列式：

$$\det\begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ \vdots & \vdots & & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nn} \end{pmatrix} = \sum(-1)^{I(a_1a_2\cdots a_n)} x_{1a_1}x_{2a_2}\cdots x_{na_n}$$

对所有排列 $a_1a_2\cdots a_n$ 求和，其中，$I(a_1a_2\cdots a_n)$ 是排列中的反序个数。

令 b_j 为位于 j 左边但大于 j 的元素个数，就得到排列 $a_1a_2\cdots a_n$ 的反序表 $b_1b_2\cdots b_n$（换言之，b_j 是第二个分量为 j 的反序的个数）。例如，排列

$$5\ 9\ 1\ 8\ 2\ 6\ 4\ 7\ 3 \tag{1}$$

有反序表

$$2\ 3\ 6\ 4\ 0\ 2\ 2\ 1\ 0 \tag{2}$$

因为 5 和 9 在 1 的左边；5，9，8 在 2 的左边；等等，全部共有 20 个反序。由定义，我们总有

$$0\leqslant b_1\leqslant n-1,\quad 0\leqslant b_2\leqslant n-2,\quad \cdots,\quad 0\leqslant b_{n-1}\leqslant 1,\quad b_n=0 \tag{3}$$

关于反序，也许最重要的事实是马歇尔·霍尔(Marshall Hall)的发现，即一张反序表唯一地确定相应的排列〔见 *Proc · Symp · Applied Math*.6(American Math.Society,

1956)，203]。我们可以通过逐次地确定元素 n，$n-1$，…，1（按这个次序）的相对位置，从满足（3）的任何反序表 $b_1b_2\cdots b_n$ 返回到产生它的唯一的排列。例如，我们可以构造对应于（2）的排列如下：写下数字 9，则因 $b_8=1$，所以 8 跟着 9，因 $b_7=2$，7 在 8 和 9 之后，因 $b_6=2$，6 跟在已经写下的头两个数之后，所以到此有

9 8 6 7

继续把 5 置于左边，因 $b_5=0$；置 4 在 4 个数之后；置 3 在 6 个数之后(即在最右边)；我们就有

5 9 8 6 4 7 3

类似地插入 2 和 1 就得到（1）。

这个对应是重要的，因为我们经常能把借助于排列叙述的问题翻译成借助于反序表叙述的一个等价问题，而后一问题可能比较易于解决。例如，考虑最简单的问题：$\{1,2,\cdots,n\}$ 可能有多少排列？回答一定是可能的反序表个数，而这是容易枚举的。因为对于 b_1 有 n 种选择，对 b_2 独立地有 $n-1$ 种选择，……，对 b_n 有一种选择，所以全部共有 $n(n-1)\cdots 1=n!$ 种选择。由于诸 b 是完全相互独立的，而诸 a 必须是彼此不同的，所以，反序容易计算。

在 1.2.10 节，我们分析了从右到左读一个排列时，该排列的局部极大值个数的问题；换言之，我们要计算有多少个元素大于它们的所有后继（例如，（1）中从右到左的极大值是 9，8，7 和 3）。这个数就是使 $b_j=n-j$ 的数目。因为 b_1 将以 $1/n$ 的概率等于 $n-1$，而 b_2 将独立地以 $1/(n-1)$ 的概率等于 $n-2$，等等，显然，由反序的考虑，从右到左的极大值的平均个数是

$$\frac{1}{n}+\frac{1}{n-1}+\cdots+1=H_n$$

类似地，也容易导出对应的生成函数。

同特定的排序算法相联系，在这一章的稍后部分，给出了反序表的其它应用。

如果我们交换一个排列的两个相邻元素，则容易看出，反序的总数将增 1 或减 1。图 1 示出了 $\{1,2,3,4\}$ 的24个排列，及其因互换相邻元素而各异的排列的连线；沿线往下使反序的数目增 1。因此，一个排列 π 的反序数是图 1 中从 1 2 3 4 到 π 的向下通路的长度；所有这样的通路必须有相同的长度。

注意，这个图形可以看作三维的立体“截八面体”，它有八个六边形的面和六个正方形的面。这是阿基米德（Archimedes）讨论过的均匀多面体之一（见习题 10）。

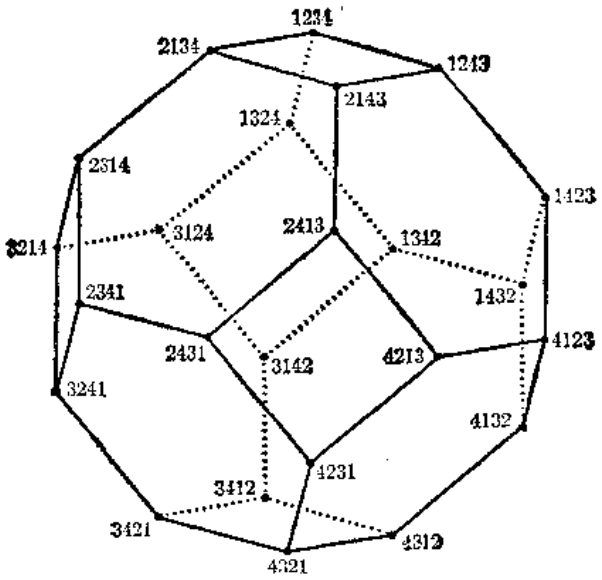


图 1　截八面体，它说明当交换排列的相邻元素时反序的变化

读者应不至于把一个排列● 的“反序”同一个排列的“逆”相混淆。回想一下我们可

● 即置换。——译注

以两行的形式写一个排列：

$$\begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ a_1 & a_2 & a_3 & \cdots & a_n \end{pmatrix} \tag{4}$$

这个排列的逆 $a'_1a'_2a_3\cdots a'_n$，是通过交换这两行，而后对各列进行排序，使得新的顶上那行按递增次序排列所得到的排列：

$$\begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_n \\ 1 & 2 & 3 & \cdots & n \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & \cdots & n \\ a'_1 & a'_2 & a'_3 & \cdots & a'_n \end{pmatrix} \tag{5}$$

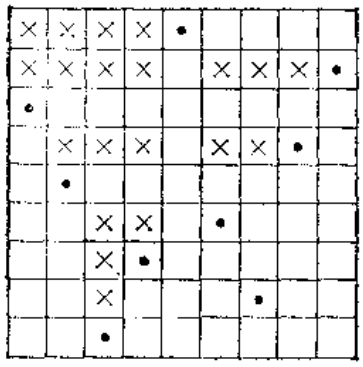
例如，由于

$$\begin{pmatrix} 5 & 9 & 1 & 8 & 2 & 6 & 4 & 7 & 3 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 5 & 9 & 7 & 1 & 6 & 8 & 4 & 2 \end{pmatrix}$$

所以，5 9 1 8 2 6 4 7 3 的逆是 3 5 9 7 1 6 8 4 2。

定义逆的另一种方法是说 $a'_j = k$ 当且仅当 $a_k = j$。

排列的逆首先是由H.A.罗瑟(H.A.Rothe)〔在K.F.Hindenburg(ed.), *Sammlung Combinatorisch-analytischer Abhandlungen* 2(Leipzig, 1800), 263-305中〕定义的。他指出了逆和反序之间一个有趣的关系：一个排列的逆恰与排列本身有同样多的反序。罗瑟对这个问题的证明不是最简单的，但它是有教益而且还是相当漂亮的。我们构造一个 $n \times n$ 的棋盘，并且每当 $a_i = j$ 时就在 i 行 j 列处点一个圆点。然后，在那些下边（在同一列）和右边（在同一行）同时都有圆点的所有正方形中打上×。例如，对于 5 9 1 8 2 6 4 7 3 的图式是



容易看出，b_j 是 j 中×的个数，所以×的个数就是反序的个数。现在如果我们把这个图式转置一下（交换行与列），就得到对应原排列的逆的图式；因此，在两种情况下，×的个数（反序的个数）是相同的。罗瑟利用这一事实证明，当对矩阵进行转置时，矩阵的行列式不变。

某些排序算法的分析涉及如下知识：有多少种 n 个元素的排列恰有 k 个反序。让我们用 $I_n(k)$ 表示此数；表 1 列出了这个函数开头的一些值。

表1　具有k个反序的排列

n	$I_n(0)$	$I_n(1)$	$I_n(2)$	$I_n(3)$	$I_n(4)$	$I_n(5)$	$I_n(6)$	$I_n(7)$	$I_n(8)$	$I_n(9)$	$I_n(10)$	$I_n(11)$
1	1	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0	0	0	0	0	0
3	1	2	2	1	0	0	0	0	0	0	0	0
4	1	3	5	6	5	3	1	0	0	0	0	0
5	1	4	9	15	20	22	20	15	9	4	1	0
6	1	5	14	29	49	71	90	101	101	90	71	49

通过考虑反序表$b_1b_2\cdots b_n$，显然有$I_n(0)=1$，$I_n(1)=n-1$，而且有对称性质

$$I_n\left(\binom{n}{2}-k\right)=I_n(k) \tag{6}$$

此外，由于每一个b可独立于其它诸b来选择，故不难看出，生成函数

$$G_n(z)=I_n(0)+I_n(1)z+I_n(2)z^2+\cdots \tag{7}$$

满足$G_n(z)=(1+z+\cdots+z^{n-1})G_{n-1}(z)$；因此，它具有比较简单的形式

$$(1+z+\cdots+z^{n-1})\cdots(1+z)(1)=(1-z^n)\cdots(1-z^2)(1-z)/(1-z)^n \tag{8}$$

由这个生成函数，我们可以容易地推广表1，而且可以验证，该表中锯齿形线下边的数满足

$$I_n(k)=I_n(k-1)+I_{n-1}(k),\quad 对于\ k<n \tag{9}$$

（这一关系对锯齿形的上边不成立。）更复杂的论证表明（见习题14），事实上，我们有公式

$$I_n(2)=\binom{n}{2}-1,\qquad n\geqslant 2;$$

$$I_n(3)=\binom{n+1}{3}-\binom{n}{1},\qquad n\geqslant 3;$$

$$I_n(4)=\binom{n+2}{4}-\binom{n+1}{2},\qquad n\geqslant 4;$$

$$I_n(5)=\binom{n+3}{5}-\binom{n+2}{3}+1,\qquad n\geqslant 5;$$

一般地说，$I_n(k)$的公式包含有大约$1.6\sqrt{k}$项：

$$I_n(k)=\binom{n+k-2}{k}-\binom{n+k-3}{k-2}+\binom{n-k-6}{k-5}-\binom{n+k-8}{k-7}-\cdots$$

$$+(-1)^j\left(\binom{n+k-u_j-1}{k-u_j}+\binom{n+k-u_j-j-1}{k-u_j-j}\right)+\cdots,\quad n\geqslant k \tag{10}$$

其中，$u_j=(3j^2-j)/2$就是所谓的“五边形数”。

如果把 $G_n(z)$ 除以 $n!$，则我们就得到了在 n 个元素的随机排列中，反序个数的概率分布的生成函数，即乘积

$$g_n(z)=h_1(z)h_2(z)\cdots h_n(z) \tag{11}$$

其中，$h_k(z)=(1+z+\cdots+z^{k-1})/k$，是小于 k 的一个随机非负整数的一致分布的生成函数。由此得出

$$\begin{aligned}\mathrm{mean}(g_n)&=\mathrm{mean}(h_1)+\mathrm{mean}(h_2)+\cdots+\mathrm{mean}(h_n)\\&=0+\frac{1}{2}+\cdots+\frac{n-1}{2}=\frac{n(n-1)}{4}\end{aligned} \tag{12}$$

$$\begin{aligned}\mathrm{var}(g_n)&=\mathrm{var}(h_1)+\mathrm{var}(h_2)+\cdots+\mathrm{var}(h_n)\\&=0+\frac{1}{4}+\cdots+\frac{n^2-1}{12}=\frac{n(2n+5)(n-1)}{72}\end{aligned} \tag{13}$$

所以反序的平均数是相当大的，约为 $\frac{1}{4}n^2$；标准离差也相当大，大约是 $\frac{1}{6}n^{3/2}$。

为了使反序研究得到一个有趣的结论，我们来讨论 P. A. 麦克马洪的著名发现〔*Amer. J. Math*, 35(1913), 281-322〕。让我们把排列 $a_1a_2\cdots a_n$ 的指数定义为使得 $a_j>a_{j+1}$，$1\leqslant j<n$ 的所有角标 j 之和。例如，5 9 1 8 2 6 4 7 3 的指数是 $2+4+6+8=20$。凑巧，在这种情况下，指数和反序个数相同。如果列出 $\{1,2,3,4\}$ 的 24 个排列，即

排列	指数	反序
1 2 3 4	0	0
1 2 4 3	3	1
1 3 2 4	2	1
1 3 4 2	3	2
1 4 2 3	2	2
1 4 3 2	5	3
2 1 3 4	1	1
2 1 4 3	4	2
2 3 1 4	2	2
2 3 4 1	3	3
2 4 1 3	2	3
2 4 3 1	5	4
3 1 2 4	1	2
3 1 4 2	4	3
3 2 1 4	3	3
3 2 4 1	4	4
3 4 1 2	2	4
3 4 2 1	5	5
4 1 2 3	1	3
4 1 3 2	4	4
4 2 1 3	3	4
4 2 3 1	4	5
4 3 1 2	3	5
4 3 2 1	6	6

则我们看到，有给定指数 k 的排列的个数，等同于有 k 个反序的排列的个数。

乍一看，这个事实几乎是显然的，但是进一步仔细研究，就觉得非常不可思议，而且显然没有简单的直接证明。麦克马洪给出了巧妙的间接证明如下：设 $J(a_1a_2\cdots a_n)$ 是排列 $a_1a_2\cdots a_n$ 的指数，并令对 $\{1,2,\cdots,n\}$ 的所有排列取和的

$$H_n(z)=\sum z^{J(a_1a_2\cdots a_n)} \tag{14}$$

是对应的生成函数，我们希望证明 $H_n(z)=G_n(z)$。为此，将定义以非负整数的 n 元组 $(q_1, q_2, \cdots, q_n)$ 为一方，和以有序 n 元组对偶

$$((a_1, a_2, \cdots, a_n), (p_1, p_2, \cdots, p_n))$$

为另一方之间的一一对应，其中，$a_1, a_2, \cdots a_n$ 是 $\{1,2,\cdots,n\}$ 的一个排列，且 $p_1\geqslant p_2\cdots\geqslant p_n\geqslant 0$。这个对应将满足条件

$$q_1+q_2+\cdots+q_n=J(a_1a_2\cdots a_n)+(p_1+p_2+\cdots+p_n) \tag{15}$$

对非负整数的所有 n 元组$(q_1, q_2, \cdots, q_n)$求和的生成函数 $\sum z^{q_1+q_2+\cdots+q_n}$是$Q_n(z)=1/(1-z)^n$，而对于满足 $p_1\geqslant p_2\geqslant\cdots\geqslant p_n\geqslant 0$ 的整数的所有 n 元组 $(p_1, p_2, \cdots, p_n)$ 求和的生成函数 $\sum z^{p_1+p_2+\cdots+p_n}$，如习题 15 所示，是

$$P_n(z)=1/(1-z)(1-z^2)\cdots(1-z^n) \tag{16}$$

依据 (15)，我们正要建立的一一对应关系将证明$Q_n(z)=H_n(z)P_n(z)$，即

$$H_n(z)=Q_n(z)/P_n(z)=G_n(z)$$

我们用一个“排序”算法来定义所希望的对应关系。由一个空表开始，对于 $k=1, 2, \cdots, n$ (按此顺序)，以如下方式把 q_k 插入这张表：$k-1$ 步之后，该表包含 $p_1, p_2, \cdots, p_{k-1}$，其中 $p_1\geqslant p_2\geqslant\cdots\geqslant p_{k-1}$；而且假定已经定义了 $\{n, n-1, \cdots, n-k+2\}$ 上的一个排列 $a_1a_2\cdots a_{k-1}$。设 j 是使 $p_j>q_k\geqslant p_{j+1}$ 的唯一整数；如果$q_k\geqslant p_1$，则令 $j=0$，而如果 $p_{k-1}>q_k$，则令 $j=k-1$。现在把 q_k 插入表中的 p_j 和 p_{j+1} 之间，把整数 $(n-k+1)$ 插入排列中的 a_j 和 a_{j+1} 之间。当对于所有的 k 这个步骤已经做完时，我们就得到 $\{1, 2, \cdots, n\}$的一个排列 $a_1a_2\cdots a_n$ 以及使得 $p_1\geqslant p_2\geqslant\cdots\geqslant p_n\geqslant 0$ 的 n 元组$(p_1, p_2, \cdots, p_n)$；且

$$p_j>p_{j+1} \qquad 每当 \qquad a_j>a_{j+1} 时$$

最后，对于使得 $a_j>a_{j+1}$ 的每个 j，$1\leqslant j<n$，从$p_1, \cdots, p_j$ 的每一个中减去 1。这样得到的对偶 $((a_1, a_2, \cdots, a_n), (p_1, p_2, \cdots, p_n))$ 满足(15)。

例如，假设 $n=6$ 和$(q_1, \cdots, q_6)=(3, 1, 4, 0, 0, 1)$，则做法如下：

k	$p_1\cdots p_k$	$a_1\cdots a_k$
1	3	6
2	3 1	6 5
3	4 3 1	4 6 5
4	4 3 1 0	4 6 5 3
5	4 3 1 0 0	4 6 5 2 3
6	4 3 1 1 0 0	4 6 1 5 2 3

最后的调整给出 $(p_1, \cdots, p_6)=(2, 1, 0, 0, 0, 0)$。

不难验证，这个过程是可逆的；所以这便建立了所希望的对应，同时证明了麦克马洪定理。在 5.1.4 节我们将会遇到类似的一一对应算法。

习题

1. 〔10〕 排列 2 7 1 8 4 5 9 3 6 的反序表是什么？什么样的排列有反序表 5 0 1 2 1 2 0 0？

2. 〔M15〕 约瑟夫 (Josephus) 问题的解，如习题 1.3.2-22 中所述，是$\{1, 2, \cdots, n\}$的一个排列；在那里给出的例子 $(n=8, m=4)$，其解为 5 4 6 1 3 8 7 2。对应这个排列的反序表是 3 6 3 1 0 0 1 0。试给出对于 n 个人，每第 m 个人被处决的一般约瑟夫问题中，反序表元素 $b_1b_2\cdots b_n$ 的一个简单的递归关系。

3. 〔18〕 如果排列 $a_1a_2\cdots a_n$ 对应于反序表 $b_1b_2\cdots b_n$，则对应于反序表

$$(n-1-b_1)(n-2-b_2)\cdots(0-b_n)$$

的排列 $\bar{a}_1\bar{a}_2\cdots\bar{a}_n$ 是什么？

▶4.〔20〕 试设计一个适合于计算机实现的算法，它根据满足(3)的一个给定的反序表 $b_1b_2\cdots b_n$ 构造出排列 $a_1a_2\cdots a_n$。〔提示：考虑一项链接的存储器技术。〕

5.〔33〕 在典型的计算机上习题4的算法要求大致与 n^2 成比例的执行时间；是否有可能设计一个算法，其运行时间的阶比 n^2 有本质上的改进？

▶6.〔26〕 设计一个算法，它根据 $\{1, 2, \cdots, n\}$ 的一个给定排列 $a_1a_2\cdots a_n$ 计算其反序表 $b_1b_2\cdots b_n$，它的典型计算机上的运行时间基本上同 $n\log_2 n$ 成比例。

7.〔20〕 除了本书定义的特殊反序表 $b_1b_2\cdots b_n$ 外，还可以定义若干其它类型的与 $\{1, 2, \cdots, n\}$ 的给定排列 $a_1a_2\cdots a_n$ 相对应的反序表；在本题中将考虑在应用中出现的三种其它类型的反序表。

设 c_j 是其头一个分量为 j 的反序的个数，即 j 右边小于 j 的元素个数〔对应于(1)我们有表 0 0 0 1 4 2 1 5 7；显然 $0 \leqslant c_j < j$〕。令 $B_j = b_{a_j}$，$C_j = c_{a_j}$。

证明，对于 $1 \leqslant j \leqslant n$，有 $0 \leqslant B_j < j$ 和 $0 \leqslant C_j \leqslant n-j$；再证明，当给定 $c_1c_2\cdots c_n$ 或 $B_1B_2\cdots B_n$ 或 $C_1C_2\cdots C_n$ 之一时，可唯一地确定排列 $a_1a_2\cdots a_n$。

8.〔$M24$〕 继续使用习题7的记号，设 $a'_1a'_2\cdots a'_n$ 是 $a_1a_2\cdots a_n$ 的逆，并设对应的反序表为 $b'_1b'_2\cdots b'_n$、$c'_1c'_2\cdots c'_n$、$B'_1B'_2\cdots B'_n$ 和 $C'_1C'_2\cdots C'_n$。尽你所能，找出在 a_j，b_j，c_j，B_j，C_j，a'_j，b'_j，c'_j，B'_j，C'_j 之间的许多有趣的关系。

9.〔$M21$〕 在习题7的记号下，证明：当且仅当对于 $1 \leqslant j \leqslant n$，有 $b_j = C_j$ 时，$a_1a_2\cdots a_n$ 是一个卷积（即它自己的逆）。

10.〔$HM20$〕 把图1当作一个三维的多面体，如果它的所有边都有单位长度，试问截八面体的直径（顶点1234和顶点4321之间的距离）是多少？

11.〔$M25$〕 (a)如果 $\pi = a_1a_2\cdots a_n$ 是 $\{1, 2, \cdots, n\}$ 的一个排列，设 $E(\pi) = \{(a_i, a_j) \mid i < j, a_i > a_j\}$ 是它的反序集合，并设

$$\overline{E}(\pi) = \{(a_i, a_j) \mid i > j,\ a_i > a_j\}$$

是那些"非反序"，证明 $E(\pi)$ 和 $\overline{E}(\pi)$ 是传递的。(有序对的一个集合 S 称为是传递的，如果每当 (a, b) 和 (b, c) 都在 S 中时，则 (a, c) 在 S 中。) (b) 反之，设 E 是 $T = \{(x, y) \mid 1 \leqslant y < x \leqslant n\}$ 的任何传递的子集，它的余集 $T \backslash E$ 也是传递的，证明存在一个使 $E(\pi) = E$ 的排列 π。

12.〔$M28$〕 继续使用上题的记号，证明：如果 π_1 和 π_2 是排列，且如果 E 是包含 $E(\pi_1) \cup E(\pi_2)$ 的最小传递集合，则 $\overline{E}$ 是传递的〔因此，如果我们说每当 $E(\pi_1) \subseteq E(\pi_2)$ 时，π_1 在 π_2"上面"，则就定义了排列的一个格；在两个给定的排列"上面"，有唯一的"最低"排列。图1是当 $n = 4$ 时格的图式〕。

13.〔$M23$〕 众所周知，行列式展开中有一半的项有+号，有一半的项有−号。换言之，当 $n \geqslant 2$ 时，具有奇数个反序的排列和具有偶数个反序的排列一样多。试证明，一般情况下，当 $n \geqslant m$ 时，不管整数 t 为何，反序个数同余于 t modulo m 的排列数目为 $n!/m$。

14.〔$M24$〕〔F·富兰克林(F·Franklin)〕把 n 分成 k 个不同部分的分划是一个表达式示 $n = p_1 + p_2 + \cdots + p_k$，其中，$p_1 > p_2 > \cdots > p_k > 0$。例如，把7分成不同部分的分划是 7，$6+1$，$5+2$，$4+3$，$4+2+1$。设 $f_k(n)$ 是把 n 分成 k 个不同部分的分划数；试证明：除非对于某个非负整数 j，n 的形式为 $(3j^2 \pm j)/2$，我们总有 $\Sigma_k(-1)^k$

$\times f_k(n)=0$；否则，和数是$(-1)^j$。例如，当$n=7$时，和数是$-1+3-1=1$，因为$7=(3\times 2^2+2)/2$。〔提示：把一个分划表示成点的一个阵列，对于$1\leqslant i\leqslant k$，置p^i个点于第i行中，试求使得$p_{j+1}<p_j-1$的最小的j，并把头j行最右边的点圈出。如果$j<p_k$，则这j个点通常可被移走，把它们转45°，并作为新的第$k+1$行放下。另一方面，如果$j\geqslant p_k$，则第k行的点通常可移走，把它们转45°，并放置到圈出点的右边（见图2）。这一过程在大多数情况下，把有奇数个行的分划同有偶数个行的分划配对了，所以在和式中仅须考虑未配对的分划。〕

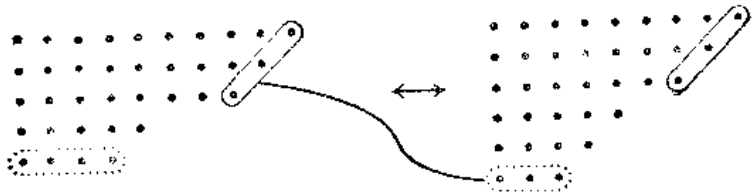


图2　具有不同部分的分划之间的富兰克林对应

注意：作为一个推论，我们得到欧拉公式

$$(1-z)(1-z^2)(1-z^3)\cdots=1-z-z^2+z^5+z^7-z^{12}-z^{15}+\cdots$$
$$=\sum_{-\infty<j<\infty}(-1)^j z^{(3j^2+j)/2}$$

由于通常的分划（它们的各部分不必是不同的）的生成函数是$\sum p(n)z^n=1/(1-z)(1-z^2)\times(1-z^3)\cdots$，因此，我们得到分划数的一个不显然的递归关系式，$p(n)=p(n-1)+p(n-2)-p(n-5)-p(n-7)+p(n+12)+p(n+15)-\cdots$。

15.〔M23〕证明（16）是对于至多分成n个部分的分划的生成函数。即，证明在$1/(1-z)(1-z^2)\cdots(1-z^n)$中，$z^m$的系数是写$m=p_1+p_2+\cdots+p_n$的方式个数，其中$p_1\geqslant p_2\geqslant\cdots\geqslant p_n\geqslant 0$。〔提示：如在习题14中那样画点，证明在$n$元组$(p_1,\ p_2,\ \cdots,\ p_n)\times(p_1\geqslant p_2\geqslant\cdots\geqslant p_n\geqslant 0)$和序列$(P_1,\ P_2,\ P_3,\ \cdots)$（$n\geqslant P_1\geqslant P_2\geqslant P_3\geqslant\cdots\geqslant 0$）之间有一一对应，并具有性质$p_1+p_2+\cdots+p_n=P_1+P_2+P_3+\cdots$。换言之，分成至多$n$个部分的分划对应于分成不超过$n$的部分的分划。〕

16.〔M25〕（L.欧拉）借助分划来解释等式两边，以证明下列恒等式：

$$\prod_{k\geqslant 0}\frac{1}{(1-q^kz)}=\frac{1}{(1-z)(1-qz)(1-q^2z)\cdots}$$
$$=1+\frac{z}{1-q}+\frac{z^2}{(1-q)(1-q^2)}+\cdots=\sum_{n\geqslant 0}z^n\Big/\prod_{1\leqslant k\leqslant n}(1-q^k)$$

$$\prod_{k\geqslant 0}(1+q^kz)=(1+z)(1+qz)(1+q^2z)\cdots$$
$$=1+\frac{z}{1-q}+\frac{z^2q}{(1-q)(1-q^2)}+\cdots$$
$$=\sum_{n\geqslant 0}z^nq^{n(n-1)/2}\Big/\prod_{1\leqslant k\leqslant n}(1-q^k)$$

17. 〔20〕 在本节末定义的麦克马洪对应中，使 $(p_1, p_2, p_3, p_4)=(0,0,0,0)$ 的24个四元组 (q_1, q_2, q_3, q_4) 是什么？

18. 〔$M30$〕〔T. 希巴德 (T. Hibbard)，$CACM$ 6(1963)，210〕 令 $n>0$，并设 2^n 个 n 位二进整数 X_0，…，X_{2^n-1}的序列已经随机地生成，其中每个数的每一个二进位独立地以概率 p 等于 1。考虑序列 $X_0\oplus 0$，$X_1\oplus 1$，…，$X_{2^n-1}\oplus(2^n-1)$，其中，$\oplus$为对二进表示取"异或"运算。于是，如果 $p=0$，则该序列为 0，1，…2^n-1，如果 $p=1$，则该序列为 2^n-1，…，1，0；当 $p=\frac{1}{2}$时，这个序列的每个元素是 0 和 2^n-1 之间的一个随机整数。对于一般的 p，这是生成有偏向的反序个数的随机整数序列的一个有用的方法，尽管整体上序列的元素分布在下述意义上是一致的，即每个 n 位二进整数有相同的分布。

确定在这样一个序列中反序的平均数（作为概率 p 的一个函数）。

19. 〔$M36$〕〔D. 福塔 (D. Foata)〕 给出麦克马洪指数定理的直接证明：找一个明确的一一对应，它把一个指数为 k 的 n 个元素的排列，同有 k 个反序和相同最右元素的排列对应起来。

20. 〔$M43$〕 下面这一属于雅各比 (Jacobi) 的著名恒等式〔*Fundamenta Nova Theoriæ Functionum Ellipticorum*) (1829)，§ 64〕是涉及椭圆函数的许多值得注意的关系式的基础

$$\prod_{k\geqslant 1}(1-u^kv^{k-1})(1-u^{k-1}v^k)(1-u^kv^k)$$

$$=(1-u)(1-v)(1-uv)(1-u^2v)(1-uv^2)(1-u^2v^2)\cdots$$

$$=1-(u+v)+(u^3v+uv^3)-(u^6v^3+u^3v^6)+\cdots$$

$$=1+\sum_{n\geqslant 1}(-1)^n(u^{(n+1)n/2}v^{(n-1)n/2}+u^{(n-1)n/2}v^{(n+1)n/2})$$

例如，若置 $u=z$，$v=z^2$，则我们得到习题 14 的欧拉公式。如果置 $z=\sqrt{u/v}$，$q=\sqrt{uv}$，则得到

$$\prod_{k\geqslant 1}(1-q^{2k-1}z)(1-q^{2k-1}z^{-1})(1-q^{2k})=\sum_{-\infty<n<\infty}(-1)^nz^nq^{n^2}$$

雅各比恒等式有一个类似于习题14中特殊情况的富兰克林 (Franklin) 证明那样的组合证明吗？(于是我们要考虑"复数分划"

$$m+ni=(p_1+q_1i)+(p_2+q_2i)+\cdots+(p_k+q_ki)$$

其中，p_j+q_ji 是不同的非零复数；p_j 和 q_j 是满足$|p_j-q_j|\leqslant 1$ 的非负整数。雅各比恒等式指出，k 为偶数的这种表示的个数和 k 为奇数的这种表示的个数相等，只有 m 和 n 是相邻的三角数时为例外！）复数分划还有什么其它值得注意的性质？

▶21. 〔$M25$〕 (G. D. 克诺特) 证明：在习题 2.2.1-5 或 2.3.1-6 的意义和习题 7 的记号下，当且仅当对于 $1\leqslant j<n$，$C_j\leqslant C_{j+1}+1$ 时，排列 $a_1\cdots a_n$ 可通过一个堆栈得到。

22. 〔$M28$〕〔C. 迈耶(C. Meyer)〕 当 m 与 n 互质时，我们知道，序列 (m mod n)

$(2m \bmod n)\cdots((n-1)m \bmod n)$ 是$\{1, 2, \cdots, n-1\}$的一个排列。证明这个排列的反序的个数可以用狄德金（Dedekind）和来表达（参看 3.3.3 节）。

*5.1.2 多重集合的排列

至今我们已经讨论了元素集合的排列；这仅仅是多重集合排列概念的一种特殊情况（一个多重集合和一个集合一样，只是它可以具有重复的相同元素。在 4.6.3 节中已经讨论了多重集合的某些基本性质）。

例如，考虑多重集合

$$M=\{a, a, a, b, b, c, d, d, d, d\} \tag{1}$$

它包含 3 个 a，2 个 b，1 个 c 和 4 个 d。我们也可以用另外的方式来表示元素多重性，即

$$M=\{3\cdot a, 2\cdot b, c, 4\cdot d\} \tag{2}$$

M的一个排列是它的元素排成一行的一种排法，例如

$$c\ a\ b\ d\ d\ a\ b\ d\ a\ d$$

从另一个观点来看，我们又称它为包含 3 个 a，2 个 b，1 个 c 和 4 个 d 的字母串。

M有多少可能的排列？如果我们把M的元素都当作不同的，通过给它们置下标 a_1，a_2，a_3，b_1，b_2，c_1，d_1，d_2，d_3，d_4，则将有 $10!=3,628,800$ 个排列。但是当去掉这些下标时，它们当中有许多实际上是相同的。事实上，M的每个排列恰好出现 $3!2!1!4!=288$ 次。因为我们可以由M的任何排列开始，以 $3!$种方式对 a 置下标，（独立地）以 $2!$种方式对 b 置下标，以 1 种方式对 c 置下标，以及以 $4!$ 种方式对 d 置下标，因此 M 的真正排列个数为

$$\frac{10!}{3!2!1!4!}=12,600$$

一般说来，通过相同的论证，我们能看到，任何多重集合的排列个数是多项式系数

$$\binom{n}{n_1,\ n_2,\ \cdots}=\frac{n!}{n_1!n_2!\cdots} \tag{3}$$

其中 n_1 是一类元素的个数，n_2 是另一类元素的个数，等等；且 $n=n_1+n_2+\cdots$ 是元素的总数。

一个集合的排列个数在古代就知道了。犹太哲学神秘主义的最早文字著作，即犹太的圣经书（约公元 100 年），就给出了头七个阶乘的正确值，在给出此值后，该书说“继续往下就会得到嘴不能说和耳不能听的数了”。〔*Sefer Yezirah*, *ed. by* R. Mordecai Atia (Jerusalem: Sh. Monson, 1962) verse 52 (pp. 107-108)；也参考 Solomon Gandz, *Studies in Hebrew Astronomy and Mathematics* (New York: Ktav, 1970), 494-496。圣经书是以猜想在七个行星、双发音法的七个谐音、人头上的七窍、以及创世的七天之间的重要关系为基础的。〕这是已知的历史上最早的排列枚举。其次出现于印度的经典著作 *Anuyogadvāra-sutra*（约公元 500 年）规则 97 中，它给出了六个元素的排列个数的公式

$$6\times5\times4\times3\times2\times1-2$$

这些排列既不是递增顺序，也不是递减顺序的。〔见 G. Chakravarti, *Bull. Calcutta*

Math. Soc. 24 (1932), 79-88。*Anuyogadvarā-sutra* 是耆那教规中一些书当中的一本，耆那教是在印度兴盛起来的一个宗教派别。〕

多重集合的相应规则似乎最早出现于 *Lilārati* of Bhāscara Ācharya（约公元1150年）一书270-271节。巴斯卡拉（Bhāscara）相当简洁地指出这个规则，而且仅以两个简单的例子{2, 2, 1, 1}和{4, 8, 5, 5, 5}来说明它。他的著作的随后的那些英译本并没有都完全正确地给出这个规则，尽管有点疑问：巴斯卡拉是否已经知道他所谈论的问题。他对20个数48555+45855+…的和曾经给出了有趣的公式

$$\frac{(4+8+5+5+5)\times 120\times 11111}{5\times 6}$$

当只有一个元素被重复时，计算排列数的正确规则是由日耳曼耶稣会学者阿塔那修斯·柯切尔(Athanasius Kircher)在他关于音乐的长篇论文〔*Musurgia Universalis* 2 (Rome, 1650) 5-7〕中发现的。他对一个给定的音符集合所能作成的曲调数目很感兴趣，所以，他设想了所谓的"音乐算术"。在论文的18～21页上，对于若干个m和n的值，他正确地给出了多重集合$\{m\cdot C,\ n\cdot D\}$的排列个数，尽管他没有揭示除$n=1$之外的计算方法。

随后，一般的规则（3）出现于琼·普雷斯提特(Jean Prestet)的论文 *Elèments de Mathematies* (Paris, 1675), 351-352中。该书是在西方世界写得非常早的解说组合数学的书之一。普雷斯提特正确地叙述了一般多重集合的规则，但是仅以简单的情况$\{a, a, b, b, c, c\}$来说明它。他特别指出，除以阶乘之和的方法，已行不通了，他认为这种除法是柯切尔（Kircher）规则的自然的推广。几年之后，约翰·瓦利斯（John Wallis）在"*Treatise of Algebra* 2"（Oxford, 1685）117-118页给出了对这一规则的更详细的讨论。

1965年，多明尼库·福塔引进了称为"插入乘积"的一项新颖的思想，使得有可能把关于通常排列的许多已知结果推广到多重集合排列的一般情况〔见 *Publ. Inst. Statistique*, Univ. Paris, 14 (1965), 81-241；也见 *Lecture Notes in Math.* 85 (Springer, 1969)〕。假定这个多重集合的元素是在某种方式下线性有序的，我们就可以考虑两行的表示法，例如

$$\begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix} \tag{4}$$

其中，上面一行包含以非减次序排好序的M的元素，而下面一行是排列本身。两个多重集合排列α和β的插入乘积$\alpha\top\beta$定义如下：（a）以两行表示法表达α和β；（b）并列这些两行的表示，以及（c）把这些列排序，使上行成为非减次序。这个排序应在这样一种意义下是"稳定的"，即当上面一行的对应元素相等时，下面一行元素自左至右的次序应被保持。例如，$c\ a\ d\ a\ b\top b\ d\ d\ a\ d=c\ a\ b\ d\ d\ a\ b\ d\ a\ d$，因为

$$\begin{pmatrix} a & a & b & c & d \\ c & a & d & a & b \end{pmatrix}\top\begin{pmatrix} a & b & d & d & d \\ b & d & d & a & d \end{pmatrix}=\begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix} \tag{5}$$

容易看出，插入乘积运算是可结合的，即

$$(\alpha\top\beta)\top\gamma=\alpha\top(\beta\top\gamma) \tag{6}$$

而且它也满足消去律

$$\pi \top \alpha = \pi \top \beta \quad 蕴涵 \quad \alpha = \beta$$
$$\alpha \top \pi = \beta \top \pi \quad 蕴涵 \quad \alpha = \beta \tag{7}$$

有一个“恒等元素”

$$\alpha \top \epsilon = \epsilon \top \alpha = \alpha \tag{8}$$

其中 ϵ 是零排列，即空集的“排法”。一般情况下，交换律不成立（见习题 2），但我们有

$$\alpha \top \beta = \beta \top \alpha \quad 如果\alpha和\beta没有公共字母 \tag{9}$$

按类似的方式，我们可以把轮换的概念推广到元素重复的情况；令

$$(x_1 \quad x_2 \quad \cdots \quad x_n) \tag{10}$$

表示以一种稳定的方式根据其顶部的元素把

$$\begin{pmatrix} x_1 & x_2 & \cdots & x_n \\ x_2 & x_3 & \cdots & x_1 \end{pmatrix} \tag{11}$$

的列进行排序所得到的两行形式的排列。例如

$$(d\ b\ d\ d\ a\ c\ a\ a\ b\ d) = \begin{pmatrix} d & b & d & d & a & c & a & a & b & d \\ b & d & d & a & c & a & a & b & d & d \end{pmatrix} = \begin{pmatrix} a & a & a & b & b & c & d & d & d & d \\ c & a & b & d & d & a & b & d & a & d \end{pmatrix}$$

所以，排列（4）实际上是一个轮换。可以这样述说一个轮换，比方说“d 变为 b 变为 d 变为 d 变为……变为 d 变回来”。注意，这些一般的轮换不具有通常轮换的所有性质：$(x_1x_2\cdots x_n)$ 不总是和 $(x_2\cdots x_nx_1)$ 一样。

在 1.3.3 节就注意到，一个集合的每个排列可以唯一地（在不计次序的意义下）表示为不相交轮换的一个乘积，其中排列的“乘积”由一种合成规则定义。容易看出，不相交轮换的乘积完全等同于它们的插入乘积；这提示我们，能够推广以前的结果，把一个多重集合的任何排列（在某种意义下）唯一地表示为轮换的插入乘积。事实上，至少有两种自然的方式来做到这一点，每一种都有重要的应用。

等式（5）说明了把 $c\ a\ b\ d\ d\ a\ b\ d\ a\ d$ 分解为较短排列的插入乘积的一条途径；让我们考虑求出一个给定排列 π 的所有分解 $\pi = \alpha \top \beta$ 的一般问题。当研究这个分解因子问题时，考虑一个特殊的排列是有帮助的，例如考虑

$$\pi = \begin{pmatrix} a & a & b & b & b & b & b & c & c & c & d & d & d & d & d \\ d & b & c & b & c & a & c & d & a & d & d & b & b & b & d \end{pmatrix} \tag{12}$$

如果我们能以形式 $\alpha \top \beta$ 来写这个排列，其中 α 至少含字母 a 一次，则 α 的两行记号的上面一行中，最左边的 a 必须出现于字母 d 之上，所以 α 至少也包含字母 d 的一次出现。如果现在观察 α 上面一行中最左边的 d，则同样看到它必须出现于字母 d 之上。所以，α 必须至少含有两个 d。考察第二个 d，我们看到：α 也必须含有一个 b。在仅仅假定 α 是 π 的一个含有字母 a 的左因子之下，有部分结果

$$\alpha = \begin{pmatrix} a & & b & & d\ d & \\ & \cdots & & \cdots & & \cdots \\ d & & & & d\ b & \end{pmatrix} \tag{13}$$

以同样方式继续进行，发现（13）上面一行中的 b 必须出现于字母 c 之上，等等。最后，这个过程将再次达到字母 a，并且可以认为这个 a 就是头一个 a，如果我们选定这样做的话。刚才作的论证本质上证明了（12）的含有字母 a 的任何左因子 α 有形式（$d\ d\ b\ c\ d\ b\ b\ c\ a$）$\top\alpha'$，其中，$\alpha'$ 是某个排列。(把 a 写在这个轮换的最后而不是最前边比较方便。由于仅有一个 a，这是允许的。）类似地，如果假定 α 含有字母 b，则我们就会导出 $\alpha=(c\ d\ d\ b)\top\alpha''$，其中 α'' 是某个排列。

一般说来，这个论证表明，如果我们有任何因子分解 $\alpha\top\beta=\pi$，其中 α 含有一个给定的字母 y，则有形为

$$(x_1\cdots x_n\,y),\quad n\geqslant 0,\quad x_1,\ \cdots,\ x_n\neq y \tag{14}$$

的唯一轮换，它是 α 的一个左因子。当给定 π 和 y 时，这个轮换容易确定；它是 π 的含有字母 y 的最短左因子。这项发现的推论之一是下面的定理：

定理A 设多重集合 M 的元素对于关系“$<$”线性有序。M 的每个排列 π 可唯一地表示为下列插入乘积：

$$\pi=(x_{11}\cdots x_{1n_1}y_1)\top(x_{21}\cdots x_{2n_2}y_2)\top\cdots\top(x_{t1}\cdots x_{tn_t}y_t)\quad t\geqslant 0 \tag{15}$$

满足下列两个条件：

$$y_1\leqslant y_2\leqslant\cdots\leqslant y_t;$$
$$y_i<x_{ij},\ \text{对于}\ 1\leqslant j\leqslant n_i,\quad 1\leqslant i\leqslant t \tag{16}$$

(换句话说，每个轮换中最后的元素小于其它元素，并且最后元素的序列是按非减次序排列的)。

证明 如果 $\pi=\epsilon$，则我们通过令 $t=0$ 而得到这样一个因子分解。否则令 y_1 是被排列的最小元素；而且如上面的例子那样，我们来确定 π 的含 y_1 的最短左因子 $(x_{11}\cdots x_{1n_1}y_1)$。现在，$\pi=(x_{11}\cdots x_{1n_1}y_1)\top\rho$，其中 ρ 为某个排列；通过对长度施行归纳法，可以写出

$\rho=(x_{21}\cdots x_{2n_2}y_2)\top\cdots\top(x_{t1}\cdots x_{tn_t}y_t)$，$t\geqslant 1$，它满足（16）的条件，这就证明了这样一个因子分解的存在性。

现在来证明满足（16）的表示（15）是唯一的。显然，$t=0$，当且仅当 π 是零排列 ϵ。当 $t>0$ 时，（16）意味着 y_1 是被排列的最小元素，而且 $(x_{11}\cdots x_{1n_1}y_1)$ 是含有 y_1 的最短左因子。因此，$(x_{11}\cdots x_{1n_1}y_1)$ 是唯一确定的；由消去律（7）和归纳法可知，这个表示是唯一的。

例如，若 $a<b<c<d$，则满足给定条件（12）的“规范的”因子分解，是

$$(d\ d\ b\ c\ d\ b\ b\ c\ a)\top(b\ a)\top(c\ d\ b)\top(d) \tag{17}$$

说明这样一点是重要的，即我们实际上可以把这个表示中的圆括弧和 $\top$ 都去掉，也不致引起二义性！每个轮换恰在剩下的最小元素头一次出现之后结束。所以这个构造把排列

$$\pi'=d\ d\ b\ c\ d\ b\ b\ c\ a\ b\ a\ c\ d\ b\ d$$

同原来的排列

$$\pi=d\ b\ c\ b\ c\ a\ c\ d\ a\ d\ d\ b\ b\ b\ d$$

联系在一起。当 π 的两行表示有形如 $\begin{smallmatrix}y\\x\end{smallmatrix}$ 的一个列时（其中 $x<y$），相关联的排列就有一对

应的相邻元素对偶$\cdots yx\cdots$。例如，我们的例子中，排列π有形如$\begin{smallmatrix}d\\b\end{smallmatrix}$的三个列，$\pi'$就有对偶$db$的三次出现。一般地说，这个构造建立了下列值得注意的定理：

定理B 设M是一个多重集合，在M的排列之间存在一个一一对应，使得如果π对应于π'，则下列条件成立：

a）π'最左边的元素等于π最左边的元素；

b）对所有满足$x<y$的排列后的元素对偶（x，y），在π的两行表示法中的列$\begin{smallmatrix}y\\x\end{smallmatrix}$出现的次数等于在$\pi'$中$y$直接在$x$之前的次数。

当M是一个集合时，这本质上就等同于我们在1.3.3节结尾处讨论的“不寻常的对应”，只有一些不重要的变化。定理B中的更一般的结果对于枚举特殊类型的排列是十分有用的，因为以两行约束为基础来解决一个问题，比起以一个相邻对偶约束为基础来解决等价的问题通常要更容易些。

P. A. 麦克马洪在他卓越的书“*Combinatory Analysis*” 1 (Cambridge Univ. Press, 1915)，168-186中考虑了这种类型的问题。他对M仅含两种不同类型的元素——比如说a和b——的特殊情况，给出了定理B的一个构造性证明。对于这种情况，他的构造本质上同这里所给出的是一样的，尽管他的表达形式十分不同。对于三种不同元素a，b，c的情况，麦克马洪给出了定理B的一个复杂的非构造性的证明；一般的情况是由福塔于1965年首先证明的。

作为定理B的一个非显然的例子，让我们来求恰巧含有

字母a的A次出现；
字母b的B次出现；
字母c的C次出现；
相邻的字母对偶ca的k次出现；
相邻的字母对偶cb的l次出现；
相邻的字母对偶ba的m次出现 (18)

的字母a，b，c的串数。本定理告诉我们这等同于形如

$$\begin{pmatrix} \overbrace{a\cdots a}^{A} & \overbrace{b\cdots b}^{B} & \overbrace{c\cdots c}^{C} \\ \underbrace{\sqcup\cdots\sqcup}_{A-k-m\text{个}a} & \underbrace{\sqcup\cdots\sqcup}_{m\text{个}a} & \underbrace{\sqcup\cdots\sqcup}_{k\text{个}a} \end{pmatrix} \tag{19}$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{C\text{个}c}$$

$$\underbrace{\qquad\qquad\qquad}_{B-l\text{个}b} \qquad \underbrace{\qquad}_{l\text{个}b}$$

的两行数组的个数。这些a可以以

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k}$$

种方式放置在第二行中，然后诸 b 可以
以

$$\binom{B+k}{B-l}\binom{C-k}{l}$$

种方式放置在剩余的位置上。剩下的空位置必须以诸 c 来填上；因此所求的数目是

$$\binom{A}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k}{B-l}\binom{C-k}{l} \tag{20}$$

让我们回到求出一个给定排列的所有因子分解的问题。是否存在一个“质”排列，即除开它本身和 ϵ 外，没有插入乘积因子的排列？定理A之前的讨论很快地给我们导出了结论，即，一个排列是质的当且仅当它是一个无重复元素的轮换。因为，如果它是这样一个轮换，则我们的论证证明，除开 ϵ 和它本身之外没有左因子。而且如果一个排列含有一个重复的元素 y，则它就有一个其中 y 仅出现一次的非显然的轮换左因子。

如果一个排列不是质的，则我们就可以进一步把它分解成越来越小的片段，直到它表达为质排列的一个乘积为止。而且如果忽略可交换的因子的次序，我们还能够证明这个因子分解是唯一的。

定理C 一个多重集合的每个排列都可以写成一个乘积

$$\sigma_1 \top \sigma_2 \top \cdots \top \sigma_l, \quad l \geqslant 0 \tag{21}$$

其中，每个 σ_j 是没有重复元素的一个轮换。在下面的意义下，这个表示是唯一的，即同一排列的任何两个这样的表示可以通过逐次地交换相邻的不相交轮换对偶的办法彼此转换。

“不相交轮换”这一术语意味着没有公共的元素。作为该定理的一个例子，我们可以验证，排列

$$\begin{pmatrix} a & a & b & b & c & c & d \\ b & a & a & c & d & b & c \end{pmatrix}$$

恰有五种分解为质因子的方式，即

$$\begin{aligned}(a\ b)\top(a)\top(c\ d)\top(b\ c) &= (a\ b)\top(c\ d)\top(a)\top(b\ c)\\ &= (a\ b)\top(c\ d)\top(b\ c)\top(a)\\ &= (c\ d)\top(a\ b)\top(a)\top(b\ c)\\ &= (c\ d)\top(a\ b)\top(b\ c)\top(a)\end{aligned} \tag{22}$$

证明 我们必须证明，所述的唯一性成立。通过对排列的长度用归纳法，只需证明：如果 ρ 和 σ 是没有重复元素的不相等的轮换，且如果

$$\rho \top \alpha = \sigma \top \beta$$

则 ρ 和 σ 是不相交的，而且对于某个排列 θ，有

$$\alpha = \sigma \top \theta, \quad \beta = \rho \top \theta$$

如果 y 是轮换 ρ 的任何元素，则含有元素 y 的 $\sigma \top \beta$ 的任何左因子必然含有 ρ 作为一个左因子。所以如果 ρ 和 σ 有一个公共的元素，则 σ 是 ρ 的一个倍数；因此 $\sigma = \rho$（因为它们是质的），这同我们的假定矛盾。故含有 y 且同 σ 没有公共元素的轮换，必然是 β 的一个左因子。通过使用消去律（7），这个证明就完成了。

作为定理C的一个例子，我们考虑由A个a，B个b和C个c组成的多重集合$M=\{A\cdot a,\ B\cdot b,\ C\cdot c\}$的排列。设$N(A,\ B,\ C,\ m)$是$M$的排列个数，其两行表示法不含有形如$\begin{smallmatrix}a & b & c\\ a, & b, & c\end{smallmatrix}$的列，而恰含有$m$个形如$\begin{smallmatrix}a\\ b\end{smallmatrix}$的列。由此得出，形如$\begin{smallmatrix}a\\ c\end{smallmatrix}$的列恰有$A-m$个，形如$\begin{smallmatrix}c\\ b\end{smallmatrix}$的恰有$B-m$个，形如$\begin{smallmatrix}c\\ a\end{smallmatrix}$的有$C-B+m$个，形如$\begin{smallmatrix}b\\ c\end{smallmatrix}$的有$C-A+m$个，以及形如$\begin{smallmatrix}b\\ a\end{smallmatrix}$的有$A+B-C-m$个，因此

$$N(A,B,C,m)=\binom{A}{m}\binom{B}{C-A+m}\binom{C}{B-m} \tag{23}$$

定理C告诉我们，可以以另一种方式计算这些排列：由于排除了形如$\begin{smallmatrix}a & b & c\\ a, & b, & c\end{smallmatrix}$的列，这一排列仅有的可能的质因子为

$$(a\ b),\ (a\ c),\ (b\ c),\ (a\ b\ c),\ (a\ c\ b) \tag{24}$$

这些轮换中的每一对至少有一公共字母，所以分解成质因子是完全唯一的。如果轮换$(a\ b\ c)$在因子分解中出现k次，则以前的假设意味着$(a\ b)$出现$m-k$次，$(b\ c)$出现$C-A+m-k$次，$(a\ c)$出现$C-B+m-k$次，以及$(a\ c\ b)$出现$A+B-C-2m+k$次。因此$N(A,\ B,\ C,\ m)$是这些轮换的排列个数（一个多项式系数），对k求和：

$$\begin{aligned}&N(A,\ B,\ C,\ m)\\ &=\sum_k \frac{(C+m-k)!}{(m-k)!(C-A+m-k)!(C-B+m-k)!\,k!\,(A+B-C-2m+k)!}\\ &=\sum_k \binom{m}{k}\binom{A}{m}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A}\end{aligned} \tag{25}$$

同（23）比较，发现下列恒等式必须成立：

$$\sum_k \binom{m}{k}\binom{A-m}{C-B+m-k}\binom{C+m-k}{A}=\binom{B}{C-A+m}\binom{C}{B-m} \tag{26}$$

这原来是在习题 1.2.6-31 中遇到的恒等式，即

$$\sum_j \binom{M-R+S}{j}\binom{N+R-S}{N-j}\binom{R+j}{M+N}=\binom{R}{M}\binom{S}{N} \tag{27}$$

其中，$M=A+B-C-m$，$N=C-B+m$，$R=B$，$S=C$，以及$j=C-B+m-k$。

类似地，可以计算$\{A\cdot a,\ B\cdot b,\ C\cdot c,\ D\cdot d\}$的排列个数，其中各种类型的列的个数指定如下：

$$\begin{array}{lcccccccc}\text{列的类型：} & \begin{smallmatrix}a\\ d\end{smallmatrix} & \begin{smallmatrix}a\\ b\end{smallmatrix} & \begin{smallmatrix}b\\ a\end{smallmatrix} & \begin{smallmatrix}b\\ c\end{smallmatrix} & \begin{smallmatrix}c\\ b\end{smallmatrix} & \begin{smallmatrix}c\\ d\end{smallmatrix} & \begin{smallmatrix}d\\ a\end{smallmatrix} & \begin{smallmatrix}d\\ c\end{smallmatrix}\\ \text{次数：} & r & A-r & q & B-q & B-A+r & D-r & A-q & D-A+q\end{array} \tag{28}$$

(这里$A+C=B+D$) 于是存在s，使这种排列的质因子分解中可能出现的轮换是(见习题12)：

轮换：	$(a\ b)$	$(b\ c)$	$(c\ d)$	$(d\ a)$	$(a\ b\ c\ d)$	$(d\ c\ b\ a)$
次数：	$A-r-s$	$B-q-s$	$D-r-s$	$A-q-s$	s	$q-A+r+s$

(29)

在这种情况下，轮换（$a\ b$）和（$c\ d$）可相互交换，而且（$b\ c$）和（$d\ a$）亦然，所以必须计算不同的质因子分解的个数。结果是（见习题 10），总有一个唯一的因子分解，使得（$c\ d$）之后不会紧跟（$a\ b$），且（$d\ a$）之后不会紧跟（$b\ c$）。因此，由习题 13 的结果，有恒等式

$$\sum_{s,\ t}\binom{B}{t}\binom{A-q-s}{A-r-s-t}\binom{B+D-r-s-t}{B-q-s}$$
$$\times\frac{D!}{(D-r-s)!(A-q-s)!\,s!(q-A+r+s)!}$$
$$=\binom{A}{r}\binom{B+D-A}{D-r}\binom{B}{q}\binom{D}{A-q}$$

从公式两边去掉因子$\binom{D}{A-q}$并简化阶乘，就化简了颇显复杂的五个参数的二项式系数恒等式

$$\sum_{s,\ t}\binom{B}{t}\binom{A-r-t}{s}\binom{B+D-r-s-t}{D+q-r-t}\binom{D-A+q}{D-r-s}\binom{A-q}{r+t-q}$$
$$=\binom{A}{r}\binom{B+D-A}{D-r}\binom{B}{q} \tag{30}$$

利用（27）可实现对s求和，而且对t的和数也容易计算；所以，在完成所有这些之后，我们未能有幸发现尚不知如何导出的任何恒等式。但是至少我们已经学习了以两种不同的方式计算某些类型的排列，而且这些计算技术对于往后的问题是一个很好的训练。

习题

1.〔$M05$〕 真或假：设M_1和M_2是多重集合，如果α是M_1的一个排列，β是M_2的一个排列，则$\alpha\top\beta$是$M_1\cup M_2$的一个排列。

2.〔10〕 在（5）中计算了$c\ a\ d\ a\ b$与$b\ d\ d\ a\ d$的插入乘积；试求当交换因子时得到的插入乘积$b\ d\ d\ a\ d\top c\ a\ d\ a\ b$。

3.〔$M13$〕（9）的逆成立否？换言之，如果α和β在插入乘积中可交换，则它们必须没有公共字母吗？

4.〔$M11$〕 当$a<b<c<d$时，在定理A的意义下，(17) 中给出了（12）的规范因子分解。试求当$d<c<b<a$时对应的规范因子分解。

5.〔$M23$〕 定理B的条件（b）要求$x<y$；如果把这个关系减弱为$x\leqslant y$，则会发生什么情况？

6.〔$M15$〕 问这样的串有多少？它恰含有m个a，n个b，又无其它字母，而且恰在k个a的紧前头都有一个b？

7.〔$M21$〕 满足条件（18）且以字母a打头的字母a，b，c的串有多少？以字母b打头的呢？以字母c打头的呢？

▶8.〔20〕 求把（12）分解成两个因子$\alpha\top\beta$的所有因子分解。

9.〔33〕 试写出把一个给定的多重集合的排列分解成定理A和定理C所述的形式的计算机程序。

▶10.〔$M30$〕 真或假：尽管根据定理C分解质因子不真正唯一，但我们可以以下列方式确保唯一性："存在质排列集合的线性次序$\prec$，使得一个多重集合的每个排列有唯一的质因子分解$\sigma_1\top\sigma_2\top\cdots\top\sigma_n$，它满足条件：对于$1\leqslant i<n$，当$\sigma_i$与$\sigma_{i+1}$可交换时，$\sigma_i\preceq\sigma_{i+1}$。

11.〔$M26$〕 设σ_1，σ_2，…，σ_t是没有重复元素的轮换。如果$i<j$且σ_i至少同σ_j有一个公共字母，则说$x_i\prec x_j$。这样就定义了t个对象$\{x_1\cdots x_t\}$上的偏序$\succeq$。证明定理C和"拓扑排序"（2.2.3节）概念之间的下列联系：$\sigma_1\top\sigma_2\top\cdots\top\sigma_t$的不同质因子分解的数目是对给定的偏序进行拓扑排序的方式个数。（例如对应于（22），我们发现有五种方式对次序$x_1\prec x_3$，$x_2\prec x_4$，$x_1\prec x_4$拓扑排序。）反之，给定t个元素的任何偏序，就有一轮换集合

$$\{\sigma_1,\ \sigma_2,\ \cdots,\ \sigma_t\}$$

以所述的方式定义偏序。

12.〔$M16$〕 证明（29）是（28）假设的一个推论。

13.〔$M21$〕 证明不含有相邻的字母对偶ca和db的$\{A\cdot a,\ B\cdot b,\ C\cdot c,\ D\cdot d,\ E\cdot e,\ F\cdot f\}$的排列个数是

$$\sum_t\binom{D}{A-t}\binom{A+B+E+F}{t}\binom{A+B+C+E+F-t}{B}\binom{C+D+E+F}{C,\ D,\ E,\ F}$$

14.〔$M30$〕 本节中定义一般排列π的"逆"π^{-1}的方式，提示另一种定义的方法，就是交换π的两行表示的行，然后对这些列做稳定的排序，以便把顶上的行变成为非减次序。例如，如果$a<b<c<d$，则这个定义意味着$c\ a\ b\ d\ d\ a\ b\ d\ a\ d^{-1}=a\ c\ d\ a\ d\ a\ b\ b\ d\ d$。

试剖析这个求逆操作的性质；例如，它同插入乘积是否有任何简单的关系？我们能计算使$\pi=\pi^{-1}$的排列的数目吗？

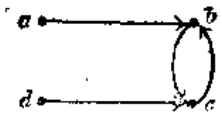
15.〔$M25$〕 证明多重集合

$$\{n_1\cdot x_1,\ n_2\cdot x_2,\ \cdots,\ n_m\cdot x_m\}$$

的排列$a_1\cdots a_n$其中，$x_1<x_2<\cdots<x_n$，$n_1+n_2+\cdots+n_m=m$是一个轮换，当且仅当且具有顶点$\{x_1,\ x_2,\ \cdots,\ x_m\}$和从$x_j$到$a_{n_1+\cdots+n_j}$的诸弧的有向图恰好包含一条有向回路。在后一种情况下，以轮换形式表示排列的方式个数是有向回路的长度。例如，对应

$$\begin{pmatrix} a & a & a & b & b & c & c & c & d & d \\ d & c & b & a & c & a & a & b & d & c \end{pmatrix}$$

的有向图是



而把这个排列表示成一个轮换的两种方式是（$b\ a\ d\ d\ c\ a\ c\ a\ b\ c$）和（$c\ a\ d\ d\ c\ a\ c\ b\ a\ b$）。

16.〔$M35$〕 在排列一个集合的特殊情况下，我们在上节建立了排列的反序的生成函数，即等式（8）。试证明，一般情况下，如果排列一个多重集合，则反序的生成函数是"z-多项式系数"

$$\binom{n}{n_1,\ n_2,\ \cdots}_z=\frac{n!_z}{n_1!_z n_2!_z\cdots},\ \text{其中}\ m!_z=\prod_{1\leqslant k\leqslant m}(1+z+\cdots+z^{k-1})$$

〔参考（3）；等式 1.2.6-37 定义了 z-二项式系数。〕

17.〔$M24$〕 利用习题 16 建立的生成函数，求出在一个给定的多重集合的随机排列中反序个数的平均值和标准离差。

18.〔$M30$〕(P. A. 麦克马洪) 上一节定义了一个排列 $a_1a_2\cdots a_n$ 的指数；而且证明了一个集合的指数为 k 的排列个数等同于有 k 个反序的排列个数。对于一个给定的多重集合的排列，同样的结果成立否？

19.〔$HM28$〕 定义一个排列 π 的 Möbius 函数 $\mu(\pi)$ 如下：如果 π 含有重复的元素，则 $\mu(\pi)$ 为零；否则，如果 π 是 k 个质因子的乘积，则 $\mu(\pi)$ 为 $(-1)^k$。〔同通常的 Möbius 函数的定义（习题 4.5.2-10）进行比较。〕（a）证明如果 $\pi\neq\epsilon$，则对所有为 π 的左因子的排列 λ 求和（即对某个 ρ，$\lambda\top\rho=\pi$），我们有

$$\Sigma\mu(\lambda)=0$$

（b）给定 $x_1<x_2<\cdots<x_m$ 和 $\pi=x_{i_1}x_{i_2}\cdots x_{i_n}$，其中，当 $1\leqslant j\leqslant n$ 时，$1\leqslant i_j\leqslant m$。证明

$$\mu(\pi)=(-1)^n\epsilon(i_1i_2\cdots i_n)$$

其中，$\epsilon(i_1i_2\cdots i_n)=\operatorname{sign}\prod_{1\leqslant j<k\leqslant n}(i_k-i_j)$。

20.〔$HM33$〕 (D. 福塔) 设 (a_{ij}) 是任意实数矩阵，在上题（b）部分的记号下，定义 $\nu(\pi)=a_{i_1j_1}\cdots a_{i_nj_n}$，其中，$\pi$ 的两行表示法为

$$\begin{pmatrix} x_{i_1} & x_{i_2} & \cdots & x_{i_n} \\ x_{j_1} & x_{j_2} & \cdots & x_{j_n} \end{pmatrix}$$

这个函数在计算一个多重集合排列的生成函数时是有用的，因为对于多重集合

$$\{n_1\cdot x_1,\ \cdots,\ n_m\cdot x_m\}$$

的所有排列 π 求和的 $\Sigma\nu(\pi)$ 将是满足某些限制的排列个数的生成函数。例如，如果对于 $i=j$，取 $a_{ij}=z$，以及对于 $i\neq j$，取 $a_{ij}=1$，则 $\Sigma\nu(\pi)$ 是对于"不动点"(上面和下面条款相同的列）个数的生成函数。为了同时研究所有多重集合的 $\Sigma\nu(\pi)$，我们考虑函数

$$G=\Sigma\pi\nu(\pi)$$

它对于包含元素 x_1，…，x_m 的多重集合的所有排列集合 $\{x_1, \cdots, x_m\}^*$ 中的所有 π 求和，我们考察 G 中 $x_1^{n_1}\cdots x_m^{n_m}$ 的系数。

在 G 的这个公式中，我们把 π 处理作诸 x 的乘积。例如，当 $m=2$ 时，有

$$\begin{aligned} G &= 1 + x_1\nu(x_1) + x_2\nu(x_2) + x_1x_1\nu(x_1x_1) + x_1x_2\nu(x_1x_2) \\ &\quad + x_2x_1\nu(x_2x_1) + x_2x_2\nu(x_2x_2) + \cdots \\ &= 1 + x_1a_{11} + x_2a_{22} + x_1^2a_{11}^2 + x_1x_2a_{11}a_{22} \\ &\quad + x_1x_2a_{21}a_{12} + x_2^2a_{22}^2 + \cdots \end{aligned}$$

于是 G 中 $x_1^{n_1}\cdots x_m^{n_m}$ 的系数就是对于 $\{n_1\cdot x_1, \cdots, n_m\cdot x_m\}$ 的所有排列 π 求和的 $\Sigma\nu(\pi)$。不难看出，这个系数也是在表达式

$(a_{11}x_1+\cdots+a_{1m}x_m)^{n_1}(a_{21}x_1+\cdots+a_{2m}x_m)^{n_2}\cdots(a_{m1}x_1+\cdots+a_{mm}x_m)^{n_m}$ 中 $x_1^{n_1}\cdots x_m^{n_m}$ 的系数。

这个习题的目的是要证明 P. A. 麦克马洪在他的"*Combinatory Analysis*" 1 (1915) 第 3 节中所谓的"主要定理"，即证明公式

$$G=1/D$$

其中

$$D=\det\begin{pmatrix} 1-a_{11}x_1 & -a_{12}x_2 & \cdots & -a_{1m}x_m \\ -a_{21}x_1 & 1-a_{22}x_2 & & -a_{2m}x_m \\ \vdots & & & \vdots \\ -a_{m1}x_1 & -a_{m2}x_2 & \cdots & 1-a_{mm}x_m \end{pmatrix}$$

例如，如果对于所有的 i 和 j，$a_{ij}=1$，则这个公式给出

$$G=1/(1-(x_1+x_2+\cdots+x_m))$$

而 $x_1^{n_1}\cdots x_m^{n_m}$ 的系数正好就是 $(n_1+\cdots+n_m)!/n_1!\cdots n_m!$。

为了证明主要定理，证明：(a) $\nu(\pi\top\rho)=\nu(\pi)\nu(\rho)$；(b) 在习题 19 的记号下，$D=\Sigma\pi\mu(\pi)\nu(\pi)$，该式右边对在

$$\{x_1, \cdots, x_m\}^*$$

中的所有排列 π 进行求和；(c) 因此 $D\cdot G=1$。

*5.1.3 路段

在 3.3.2 节中，曾考虑过在一个排列中的"上运行"，作为测试一个序列的随机性的一项方法。如果在一个排列 $a_1a_2\cdots a_n$ 的两端各放一根垂直的线段，而且每当 $a_j>a_{j+1}$ 时，也在 a_j 与 a_{j+1} 之间放一根直线，则路段就是诸对线之间的区段。例如，排列

$$|3\ 5\ 7|1\ 6\ 8\ 9|4|2|$$

有四个路段。我们已确定了在 $\{1, 2, \cdots, n\}$ 的随机排列中长度为 k 的路段的平均个数，以及长度为 j 和 k 的路段的个数的协方差。路段在排序算法的研究中是重要的，因为它们表示数据已排序的诸段，所以现在将再次研究路段这个课题。

让我们使用记号

$$\left\langle {n \atop k} \right\rangle \tag{1}$$

来代表{1, 2, …, n}的排列中，恰有 k 个递增路段的个数。数$\left\langle {n \atop k} \right\rangle$在许多场合中出现，而且它们通常被叫做欧拉数，因为欧拉在他著名的书 *Institutions calculi differentialis* (St. Petersburg: 1755) 485-487 中讨论了它们〔Euler: *Opera Omnia* (1) 10 (1913), 373-375〕；它们应该与习题 5.1.4-22 中讨论的所谓"欧拉数"相区别。

通过把元素 n 插入到所有可能的位置上，我们可以使用对{1, …, $n-1$}的任何给定的排列来形成 n 个新的排列。如果原来的排列有 k 个路段，则恰有 k 个新的排列将有 k 个路段；剩下的 $n-k$ 个排列将有 $k+1$ 个路段，因为除非把元素 n 放置在一个现有路段的末端，不然路段的数目是会增加的。例如，从 3 1 2 4 5 形成的六个排列是

6 3 1 2 4 5, 3 6 1 2 4 5, 3 1 6 2 4 5,

3 1 2 6 4 5, 3 1 2 4 6 5, 3 1 2 4 5 6;

除第二个和最后一个外，它们都有三个路段，而不是两个。因此有递归关系

$$\left\langle {n \atop k} \right\rangle = k\left\langle {n-1 \atop k} \right\rangle + (n-k+1)\left\langle {n-1 \atop k-1} \right\rangle, \quad 整数\ n \geqslant 1;\ 整数\ k \tag{2}$$

按约定，置

$$\left\langle {0 \atop k} \right\rangle = \delta_{1k} \tag{3}$$

表示零排列有一个路段。读者可以发现，把（2）同斯特林数的递归关系进行比较是有趣的（等式1.2.6-42）。表1列出了对于小的 n 的欧拉数。

表1　欧拉数

n	$\left\langle {n \atop 0} \right\rangle$	$\left\langle {n \atop 1} \right\rangle$	$\left\langle {n \atop 2} \right\rangle$	$\left\langle {n \atop 3} \right\rangle$	$\left\langle {n \atop 4} \right\rangle$	$\left\langle {n \atop 5} \right\rangle$	$\left\langle {n \atop 6} \right\rangle$	$\left\langle {n \atop 7} \right\rangle$
0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0
2	0	1	1	0	0	0	0	0
3	0	1	4	1	0	0	0	0
4	0	1	11	11	1	0	0	0
5	0	1	26	66	26	1	0	0
6	0	1	57	302	302	57	1	0
7	0	1	120	1191	2416	1191	120	1

在表1中可以观察到若干规律。由定义，我们有

$$\left\langle {n \atop 0} \right\rangle + \left\langle {n \atop 1} \right\rangle + \cdots + \left\langle {n \atop n} \right\rangle = n! \tag{4}$$

$$\left\langle {n \atop 0} \right\rangle = 0, \quad \left\langle {n \atop 1} \right\rangle = 1 \tag{5}$$

$$\left\langle {n \atop n} \right\rangle = 1, \quad 对于\ n \geqslant 1 \tag{6}$$

也有一个对称性规则

$$\left\langle {n \atop k} \right\rangle = \left\langle {n \atop n+1-k} \right\rangle, \quad n \geqslant 1 \tag{7}$$

它从这样一个事实得出，即每个有 k 个路段的排列 $a_1a_2\cdots a_n$ 对应于有 $n+1-k$ 个路段的排列 $a_n\cdots a_2a_1$。

欧拉数另一个重要的性质是公式

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{m+k-1}{n} = m^n, \quad n \geqslant 0 \tag{8}$$

可以用排序的思想证明它：考虑 m^n 个使得 $1 \leqslant a_i \leqslant m$ 的序列 $a_1a_2\cdots a_n$。我们可以把任何这样的序列以一种稳定的方式排成非减的次序，得到关系

$$a_{i_1} \leqslant a_{i_2} \leqslant \cdots \leqslant a_{i_n} \tag{9}$$

其中 $i_1i_2\cdots i_n$ 是 $\{1, 2, \cdots, n\}$ 的一个唯一确定的排列，使得 $a_{i_j}=a_{i_{j+1}}$ 意味着 $i_j<i_{j+1}$；换言之，$i_j>i_{j+1}$ 意味着 $a_{i_j}<a_{i_{j+1}}$。如果排列 $i_1i_2\cdots i_n$ 有 k 个路段，则将证明对应的序列 $a_1a_2\cdots a_n$ 的个数是 $\binom{m+n-k}{n}$，而且如果我们以 $n+1-k$ 来代替 k，就证明了（8）。

例如，如果 $n=9$ 且 $i_1i_2\cdots i_n=3\ 5\ 7\ 1\ 6\ 8\ 9\ 4\ 2$，则我们要计算使得

$$1 \leqslant a_3 \leqslant a_5 \leqslant a_7 < a_1 \leqslant a_6 \leqslant a_8 \leqslant a_9 < a_4 < a_2 \leqslant m \tag{10}$$

的序列 $a_1a_2\cdots a_n$ 的个数，是使得

$$1 \leqslant b_1 < b_2 < b_3 < b_4 < b_5 < b_6 < b_7 < b_8 < b_9 \leqslant m+5$$

的序列 $b_1b_2\cdots b_9$ 的个数。因为我们能令 $b_1=a_3$，$b_2=a_5+1$，$b_3=a_7+2$，$b_4=a_1+2$，$b_5=a_6+3$ 等等。选择诸 b 的方式的个数，就是由 $m+5$ 件事物中选择 9 件的方式数，即 $\binom{m+5}{9}$；类似的证明对于一般的 k 和 n，以及对于任何具有 k 个路段的任何排列 $i_1i_2\cdots i_n$ 也有效。

由于（8）的两边都是 m 的多项式，故可以用任何实数 x 来代替 m，从而得到用相继的二项式系数表示乘方的一种有趣方法：

$$x^n = \left\langle {n \atop 1} \right\rangle \binom{x}{n} + \left\langle {n \atop 2} \right\rangle \binom{x+1}{n} + \cdots + \left\langle {n \atop n} \right\rangle \binom{x+n-1}{n}, \quad n \geqslant 1 \tag{11}$$

例如，

$$x^3 = \binom{x}{3} + 4\binom{x+1}{3} + \binom{x+2}{3}$$

这是欧拉数的关键性质，这种性质使得欧拉数在离散数学的研究中很有用。

在（11）中置 $x=1$，就再次证明了

$$\left\langle {n \atop n} \right\rangle = 1$$

因为二项式系数除了最后一项外都消失了。置 $x=2$，得到

$$\left\langle {n \atop n-1} \right\rangle = 2^n - n - 1, \quad n \geqslant 1 \tag{12}$$

置 $x=3, 4, \cdots$ 即知关系（11）完全确定了数 $\left\langle {n \atop k} \right\rangle$，并且导出了最初由欧拉给出的一个公式：

$$\begin{aligned}\left\langle {n \atop k} \right\rangle &= k^n - (k-1)^n\binom{n+1}{1} + (k-2)^n\binom{n+1}{2} - \cdots + (-1)^k 0^n\binom{n+1}{k} \\ &= \sum_{0 \leqslant j \leqslant k} (-1)^j (k-j)^n \binom{n+1}{j}, \quad n \geqslant 0, \quad k \geqslant 0\end{aligned} \tag{13}$$

现在研究路段的生成函数。如果置

$$g_n(z)=\sum_k \left\langle {n \atop k} \right\rangle z^k/n! \tag{14}$$

则 z^k 的系数是 $\{1, 2, \cdots, n\}$ 的一个随机排列恰有 k 个路段的概率。由于 k 个路段的可能性恰巧等同于 $n+1-k$ 个，故路段的平均数必须是 $\frac{1}{2}(n+1)$，因此 $g_n'(1)=\frac{1}{2}(n+1)$。习题 2（b）证明，对于 $g_n(z)$ 在点 $z=1$ 的各次导数，有一个简单的公式

$$g_n^m(1)=\left\{ {n+1 \atop n+1-m} \right\} \Big/ \binom{n}{m}, \quad n \geqslant m \tag{15}$$

例如，当 $n \geqslant 2$ 时，方差 $g_n''(1)+g_n'(1)-g_n'(1)^2$ 成为 $\frac{1}{12}(n+1)$，它表明均值有一个比较稳定的分布（已在等式 3.3.2-18 中求得了同样的量，在那里它被称作协方差 (R_1', R_1')）。由于 $g_n(z)$ 是一个多项式，可以使用公式（15）来导出泰勒级数展开式

$$\begin{aligned} g_n(z) &= \frac{1}{n!} \sum_{0 \leqslant k \leqslant n} (z-1)^{n-k} \cdot k! \left\{ {n+1 \atop k+1} \right\} \\ &= \frac{1}{n!} \sum_{0 \leqslant k \leqslant n} z^{k+1}(1-z)^{n-k} k! \left\{ {n+1 \atop k+1} \right\} \end{aligned} \tag{16}$$

这公式的第二式从第一式得出，因为由对称性条件（7）

$$g_n(z)=z^{n+1} g_n\left(\frac{1}{z}\right), \quad n \geqslant 1 \tag{17}$$

斯特林数递归式

$$\left\{ {n+1 \atop k+1} \right\}=(k+1)\left\{ {n \atop k+1} \right\}+\left\{ {n \atop k} \right\}$$

给出了两个稍微更简单些的表示

$$\begin{aligned} g_n(z) &= \frac{1}{n!} \sum_{0 \leqslant k \leqslant n} z(z-1)^{n-k} k! \left\{ {n \atop k} \right\} \\ &= \frac{1}{n!} \sum_{0 \leqslant k \leqslant n} z^k(1-z)^{n-k} k! \left\{ {n \atop k} \right\} \end{aligned} \tag{18}$$

因此，超生成函数

$$g(z, x)=\sum_{n \geqslant 0} g_n(z) x^n = \sum_{k, n \geqslant 0} \left\langle {n \atop k} \right\rangle z^k x^n / n! \tag{19}$$

等于

$$z \sum_{n, k \geqslant 0} \frac{((z-1)x)^n}{(z-1)^k} \left\{ {n \atop k} \right\} \frac{k!}{n!} = z \sum_{k \geqslant 0} \left(\frac{e^{(z-1)x}-1}{z-1} \right)^k = \frac{z(1-z)}{e^{(z-1)x}-z} \tag{20}$$

这是欧拉讨论的另一个关系。

欧拉数进一步的性质可以在 L. 卡利兹（L. Carlitz）的综述性论文〔*Math. Magazine* 33（1959），247-260〕中找到；也可见 J. 赖尔登（J. Riordan）的 "*Introduction to Combinatorial Analysis*"（New York：Wiley，1958）38-39，214-219，234-237。

现在考虑路段的长度；平均说来，一个路段的长度等于多少？在 3.3.2 节，已经研究了给定长度路段的预期数目；路段的平均长度近似为 2，同长度为 n 的一个随机排列中大约有 $\frac{1}{2}(n+1)$ 个路段这一事实相一致。为了应用于排序算法，现在换一个角度来考察，对于 $k=1, 2, \cdots$，考虑在排列中从左到右第 k 个路段的长度。

例如，一个随机排列 $a_1a_2\cdots a_n$ 的头一个（最左边的）路段有多长？它的长度总是 $\geqslant 1$，而且恰有一半的次数它的长度 $\geqslant 2$（即当 $a_1<a_2$ 时）。恰有六分之一的次数它的长度 $\geqslant 3$（当 $a_1<a_2<a_3$ 时）。而且一般说来，对于 $1\leqslant m\leqslant n$，它的长度 $\geqslant m$ 的概率是 $q_m=1/m!$。因此，它的长度恰恰等于 m 的概率是

$$p_m=q_m-q_{m+1}=1/m!-1/(m+1)!, \qquad \text{对于 } 1\leqslant m<n \tag{21}$$

$$p_n=1/n!$$

因此平均长度为

$$\begin{aligned} p_1+2p_2+\cdots+np_n &= (q_1-q_2)+2(q_2-q_3)+\cdots+(n-1)(q_{n-1}-q_n)+nq_n \\ &= q_1+q_2+\cdots+q_n=\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{n!} \end{aligned} \tag{22}$$

如果令 $n\to\infty$，则极限为 $e-1=1.71828\cdots$，而且对于有限的 n，这个值为 $e-1-\delta_n$，其中 δ_n 十分小

$$\delta_n=\frac{1}{(n+1)!}\left(1+\frac{1}{n+2}+\frac{1}{(n+2)(n+3)}+\cdots\right)<\frac{e-1}{(n+1)!}$$

因此，为了实用，不妨研究在不同的数的随机无穷序列

$$a_1,\ a_2,\ a_3,\ \cdots$$

中的路段；这里我们所说的"随机"，是指在这个序列中，头 n 个元素的 $n!$ 种相对次序的每一种都是同等可能的。因此，在一个随机无穷序列中头一个路段的平均长度为 $e-1$。

只要稍微地加强这项方法，就能确定在一个随机序列中第 k 个路段的平均长度。设 q_{km} 是头 k 个路段的总长度 $\geqslant m$ 的概率；则 q_{km} 是 $1/m!$ 乘以路段数 $\leqslant k$ 的 $\{1, 2, \cdots, m\}$ 的排列数

$$q_{km}=\left(\left\langle {m \atop 1} \right\rangle+\cdots+\left\langle {m \atop k} \right\rangle\right)\Big/ m! \tag{23}$$

头 k 个路段总长度是 m 的概率为 $q_{km}-q_{k(m+1)}$。因此，如果 L_k 表示第 k 个路段的平均长度，则求得

$$\begin{aligned} &L_1+\cdots+L_k=\text{头 } k \text{ 个路段的平均总长度} \\ &=(q_{k1}-q_{k2})+2(q_{k2}-q_{k3})+3(q_{k3}-q_{k4})+\cdots \\ &=q_{k1}+q_{k2}+q_{k3}+\cdots \end{aligned}$$

减去 $L_1+\cdots+L_{k-1}$ 并利用（23）中的 q_{km} 的值即得所求的公式

$$L_k=\frac{1}{1!}\left\langle {1 \atop k} \right\rangle+\frac{1}{2!}\left\langle {2 \atop k} \right\rangle+\frac{1}{3!}\left\langle {3 \atop k} \right\rangle+\cdots=\sum_{m\geqslant 1}\left\langle {m \atop k} \right\rangle\frac{1}{m!} \tag{24}$$

由于除当 $k=1$ 外 $\left\langle {0 \atop k} \right\rangle = 0$，故 L_k 可证明是生成函数 $q(z,1)-z$（见等式 19）中 z^k 的系数，所以有

$$L(z)=\sum_{k\geqslant 0} L_k z^k = \frac{z(1-z)}{e^{z-1}-z} - z \tag{25}$$

由欧拉公式（13）得到 L_k 作为 e 的一个多项式的表示

$$\begin{aligned}
L_k &= \sum_{m\geqslant 0} \sum_{0\leqslant j\leqslant k} (-1)^{k-j} \binom{m+1}{k-j} \frac{j^m}{m!} \\
&= \sum_{0\leqslant j\leqslant k} (-1)^{k-j} \sum_{m\geqslant 0} \binom{m}{k-j} \frac{j^m}{m!} + \sum_{0\leqslant j\leqslant k} (-1)^{k-j} \sum_{m\geqslant 0} \binom{m}{k-j-1} \frac{j^m}{m!} \\
&= \sum_{0\leqslant j\leqslant k} \frac{(-1)^{k-j} j^{k-j}}{(k-j)!} \sum_{n\geqslant 0} \frac{j^n}{n!} + \sum_{0\leqslant j\leqslant k} \frac{(-1)^{k-j} j^{k-j-1}}{(k-j-1)!} \sum_{n\geqslant 0} \frac{j^n}{n!} \\
&= k \sum_{0\leqslant j\leqslant k} e^j \frac{(-1)^{k-j} j^{k-j-1}}{(k-j)!}
\end{aligned} \tag{26}$$

这个 L_k 的公式首先由 B. J. 加斯纳（B. J. Gassner）得到〔见 CACM 10(1967)，89-93〕。特别是，有

$$L_1 = e-1 \approx 1.71828\cdots;$$
$$L_2 = e^2-2e \approx 1.95249\cdots;$$
$$L_3 = e^3-3e^2+\frac{3}{2}e \approx 1.99579\cdots$$

所以预期第二个路段较头一个长，而且平均地说，第三个还得更长！乍一看，这似乎是使人惊奇的，但可以联想到由于第二个路段的头一个元素趋于变小(它引起第一个路段终止)，故第二个路段就有一个较好的机会变得更长些。第三个路段的头一个元素甚至比第二个的更小。

在替代一选择排序理论中，数 L_k 有其重要性（5.4.1节），所以详细研究它们的值是有意义的。由小 J. W. 伦奇计算的表 2 示出了 L_k 的头 18 个值到 15 位十进数。在上段中的讨论可能使我们首先猜测 $L_{k+1}>L_k$，但是事实上这些值是向后和向前摆动的。注意，L_k 迅速地趋于极限值 2；看看超越数 e 的这些单一多项式多么迅速地收敛到有理数 2 是很有意

表 2　第 k 个路段的平均长度

k	L_k			k	L_k		
1	1.71828	18284	59045 +	10	2.00000	00012	05997 +
2	1.95249	24420	12560 −	11	2.00000	00001	93672 +
3	1.99579	13690	84285 −	12	1.99999	99999	99909 +
4	2.00003	88504	76806 −	13	1.99999	99999	97022 −
5	2.00005	75785	89716 +	14	1.99999	99999	99719 +
6	2.00000	50727	55710 −	15	2.00000	00000	00019 +
7	1.99999	96401	44022 +	16	2.00000	00000	00006 +
8	1.99999	98889	04744 +	17	2.00000	00000	00000 +
9	1.99999	99948	43434 −	18	2.00000	00000	00000 −

思的！从数值分析的观点来看，多项式（26）也是颇为有趣的，因为它们提供了当接近相等的数相减时有效位数丢失的一个精采的例子；利用19位浮点算术，加斯纳得到不正确的结论 $L_{12}>2$，而伦奇已经注意到，用 42 位浮点算术来计算 L_{28} 时仅仅正确到 29 位有效数字。

L_k的渐近性质可用复变数理论的简单原理来确定。仅当 $e^{z-1}=z$，即(令$z=x+iy$)当

$$e^{x-1}\cos y=x \qquad 和 \qquad e^{x-1}\sin y=y \tag{27}$$

时（25）的分母为 0，图 3 示出这两个方程的叠印图。我们注意到，它们在 $z=z_0$，z_1，$\bar{z}_1$，z_2，$\bar{z}_2$，…处相交，这里，$z_0=1$，

$$z_1=(3.08884\ 30156\ 13044-)+(7.46148\ 92856\ 54255-)i \tag{28}$$

而且虚部$\mathscr{I}(z_{k+1})$对于很大的 k 粗略地等于$\mathscr{I}(z_k)+2\pi$。

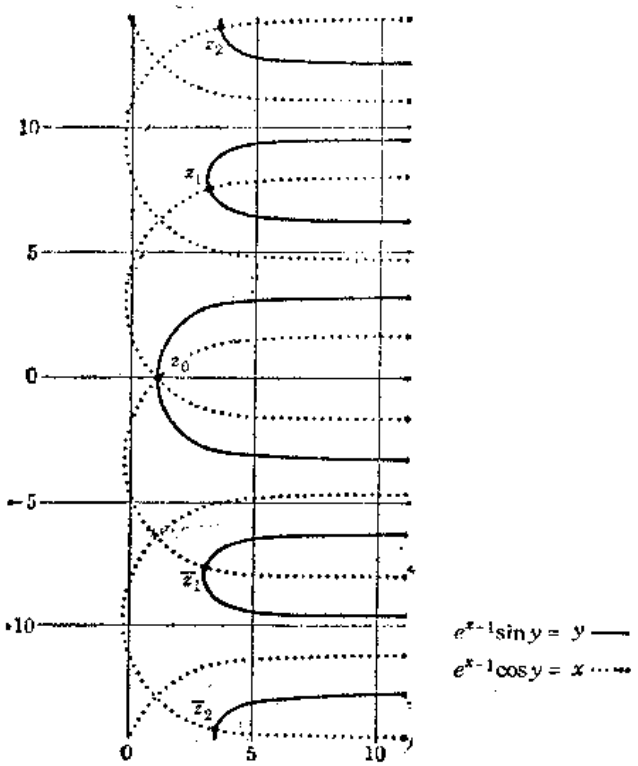


图 3　$e^{z-1}=z$ 的根

由于

$$\lim_{z\to z_k}\left(\frac{1-z}{e^{z-1}-z}\right)(z-z_k)=-1, \qquad 对于\ k>0$$

且对 $k=0$，此极限为-2，故函数

$$R_m(z)=L(z)+\frac{2}{z-z_0}+\frac{z_1}{z-z_1}+\frac{\bar{z}_1}{z-\bar{z}_1}+\frac{z_2}{z-z_2}+\frac{\bar{z}_2}{z-\bar{z}_2}$$

$$+\cdots+\frac{z_m}{z-z_m}+\frac{\bar{z}_m}{z-\bar{z}_m}$$

在$|z|<|z_{m+1}|$的复平面中没有奇异点，因此 $R_m(z)$有一个幂级数展开式$\Sigma_k\rho_k z^k$，当 $|z|<$

$|z_{m+1}|$时，它绝对收敛；由此得出当$k\to\infty$时，$\rho_k M^k\to 0$，其中$M=|z_{m+1}|-\epsilon$。$L(z)$的系数是

$$\frac{2}{1-z}+\frac{1}{1-z/z_1}+\frac{1}{1-z/\bar{z}_1}+\cdots+\frac{1}{1-z/z_m}+\frac{1}{1-z/\bar{z}_m}+R_m(z)$$

的系数

$$L_n=2+2r_1^{-n}\cos n\theta_1+2r_2^{-n}\cos n\theta_2+\cdots+2r_m^{-n}\cos n\theta_m+O(r_{m+1}^{-n}) \tag{29}$$

若令

$$z_k=r_k e^{i\theta_k} \tag{30}$$

这说明了L_n的渐近性质。我们有

$$\begin{aligned}
&r_1=8.07556\ 64528\ 89526-,\\
&\theta_1=1.17830\ 39784\ 74668+;\\
&r_2=14.35457-, \qquad \theta_2=1.31269-;\\
&r_3=20.62073+, \qquad \theta_3=1.37428-;\\
&r_4=26.88795+, \qquad \theta_4=1.41050-
\end{aligned} \tag{31}$$

所以对于L_n-2的主要贡献乃归功于r_1和θ_1，而且（29）的收敛是十分迅速的。所述r_1和θ_1的值是由小J.W.伦奇确定的。进一步的分析〔W.W.Hooker，*CACM* 12（1969），411-413〕证明，当$m\to\infty$时，$R_m(z)\to -z$；因此当$n>1$时，级数$2\Sigma_{k\geq 0}r_k^{-n}\cos n\theta_k$实际上收敛于$L_n$。

可以对概率进行更仔细的考察，以确定对于第k个路段长度和对头k个路段的总长度的整个概率分布（见习题9，10，11）。和数$L_1+L_2+\cdots+L_k$结果是渐近于$2k-1/3$。

在结束本节之前，让我们来考虑：当在排列中允许出现相等的元素时路段的性质，十九世纪著名的英国天文学家西蒙·纽科姆（Simon Newcomb）对做一种与这个问题有关的游戏很感兴趣。他把一副扑克牌一张一张地放在桌上，只要牌的面值是以非减次序出现的，他就把它们堆成一堆，但每当待堆放的下一张牌的面值小于前者时，他就开始一个新的堆，他想要知道，在整副牌以这种方式分发完毕之后，形成给定堆数的概率。

因此，西蒙·纽科姆的问题就是，求在一个多重集合的随机排列中，路段的概率分布问题。尽管已经看过怎样来解决当所有的牌都有不同的面值时的特殊情况，但一般的回答是比较复杂的（见习题12）。这里，我们将满足于推导出现于这个游戏中的平均堆数。

首先假定有m种不同类型的牌，每种类型恰有p张。例如，通常的桥牌，如果不考虑花色时，有$m=13$和$p=4$。P.A.麦克马洪发现了应用于这种情况的值得注意的对称性〔*Combinatory Analysis*1,（Cambridge，1915），212-213〕：有$k+1$个路段的排列个数等同于有$mp-p-k+1$个路段的排列个数。当$p=1$时，这个关系易于验证（等式7），但对于$p>1$时，它是十分令人惊奇的。

通过把具有$k+1$个路段的每个排列对应于有$mp-p-k+1$个路段的另一个排列，以建立排列之间的一一对应，就能够证明这对称性。在往下阅读之前，鼓励读者亲自动手试一试来发现这一对应。

显然，没有很简单的对应；麦克马洪的证明是以生成函数而不是以一种组合构造为基础的。但福塔的对应（定理5.1.2B）提供了一个有用的简化，因为它告诉我们，在具有$k+1$个路段的排列和其两行表示法恰含有k个$x<y$的列$\binom{x}{y}$的排列之间，有一一对应。

假设给定的多重集合是$\{p\cdot 1,\ p\cdot 2,\ \cdots,\ p\cdot m\}$，考察其两行表示法为

$$\begin{pmatrix} 1 & \cdots & 1 & 2 & \cdots & 2 & \cdots & m & \cdots & m \\ x_{11} & \cdots & x_{1p} & x_{21} & \cdots & x_{2p} & \cdots & x_{m1} & \cdots & x_{mp} \end{pmatrix} \tag{32}$$

的排列，可以把这个排列和同一多重集合的另一个排列结合起来

$$\begin{pmatrix} 1 & \cdots & 1 & 2 & \cdots & 2 & \cdots & m & \cdots & m \\ x'_{11} & \cdots & x'_{1p} & x'_{m1} & \cdots & x'_{mp} & \cdots & x'_{21} & \cdots & x'_{2p} \end{pmatrix} \tag{33}$$

其中$x'=m+1-x$。如果(32)含有形如$\begin{smallmatrix} y \\ x \end{smallmatrix}$且$x<y$的$k$个列，则(33)含有$(m-1)p-k$个这样的列；因为仅仅需要考虑$y>1$的情况，而且$x<y$等价于$x'\geqslant m+2-y$。由于(32)对应于具有$k+1$个路段的一个排列，而(33)对应于具有$mp-p-k+1$个路段的一个排列，而且由于把(32)变为(33)的变换是可逆的〔它把(33)变回(32)〕，麦克马洪的对称性条件已经建立。关于这个构造的一个例子，见习题14。

由于对称性，在一个随机排列中路段的平均数必须是$\frac{1}{2}[(k+1)+(mp-p-k+1)]$ $=1+\frac{1}{2}p(m-1)$。例如，利用一副标准扑克牌，由西蒙·纽康姆的单人游戏得到的平均堆数将是25（所以它不象是一种有吸引力的单人游戏）。

给定任意多重集合$\{n_1\cdot x_1,\ n_2\cdot x_2,\ \cdots,\ n_m\cdot x_m\}$，其中诸$x$是不同的，利用一个相当简单的论证，实际上可以一般地确定路段的平均数。设$n=n_1+n_2+\cdots+n_m$，并想象这个多重集合的所有排列$a_1a_2\cdots a_n$都已经写下来；我们来考察；对于每个固定的i值，a_i大于a_{i+1}的机会如何，这里$1\leqslant i<n$。$a_i>a_{i+1}$的次数恰是$a_i\neq a_{i+1}$的次数之半；而且不难看出，$a_i=a_{i+1}=x_j$恰有$Nn_j(n_j-1)/n(n-1)$次，其中N是排列的总数。因此，$a_i=a_{i+1}$恰有

$$\frac{N}{n(n-1)}(n_1(n_1-1)+\cdots+n_m(n_m-1))=\frac{N}{n(n-1)}(n_1^2+\cdots+n_m^2-n)$$

次，而且$a_i>a_{i+1}$恰有

$$\frac{N}{2n(n-1)}(n^2-(n_1^2+\cdots+n_m^2))$$

次。因为在每个排列中定有一个路段在a_n处结束，因此若对i求和并加上N，则得到在所有N个排列当中路段的总数

$$N\left(\frac{n}{2}-\frac{1}{2n}(n_1^2+\cdots+n_m^2)+1\right) \tag{34}$$

除以N即给出所求的平均路段数。

由于在“顺序统计”的研究中路段是重要的，因此有相当大量的著作讨论它们，包括这里未予考虑的若干其它类型的路段在内。关于另外的信息，请看F.N.戴维(F.N. David)和D.E.巴顿(D.E.Barton)著的书“*Combinatorial Chance*”(London:Griffin, 1962)第10章；以及由D.E.巴顿与C.L.马洛斯(C.L.Mallows)合写的综述论文“*Annals of Math. Statistics*” 36 (1965), 236-260。欧拉数与排列的进一步联系，出现于D.福塔和M.P.舒曾伯杰(M.P.Schützenberger)的“*Théorie Geometrique des Polynômes Euleriens*” *Lecture Notes in Math.* (Berlin Springer, 1970)94页中。

习题

1. 〔$M26$〕 推导欧拉公式（13）。

2. 〔$M22$〕 （a）推广正文中用来证明（8）的思想，考虑恰含 q 个不同元素的那些序列 $a_1a_2\cdots a_n$，以证明

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k-1}{n-q} = \left\{ {n \atop q} \right\} q!$$

（b）用这一恒等式来证明，当 $n \geqslant m$ 时有

$$\sum_k \left\langle {n \atop k} \right\rangle \binom{k}{m} = \left\{ {n+1 \atop n+1-m} \right\} (n-m)!$$

3. 〔$HM25$〕 计算和数 $\sum_k \binom{n}{k}(-1)^k$。

4. 〔$M21$〕

$$\sum_k (-1)^k \left\{ {n \atop k} \right\} k! \binom{n-k}{m}$$

的值等于多少？

5. 〔$M20$〕 当 p 是质数时，求 $\binom{p}{k} \bmod p$ 的值。

▶6. 〔$M21$〕 B. C. 达尔先生注意到，由等式（4）和（13）

$$n! = \sum_{k \geqslant 0} \left\langle {n \atop k} \right\rangle = \sum_{k \geqslant 0} \sum_{j \geqslant 0} (-1)^{k-j} \binom{n+1}{k-j} j^n$$

首先对 k 进行求和，他发现对于所有的 $j \geqslant 0$，$\sum_{k \geqslant 0}(-1)^{k-j}\binom{n+1}{k-j} = 0$；因此，对于所有的 $n \geqslant 0$，$n! = 0$，他搞错了没有？

7. 〔$HM40$〕 由(14)给出的路段的概率分布，是否渐近于正态的（参考1.2.10-13）？

8. 〔$M24$〕 （P · A · 麦克马洪）证明一个充分长的排列的头一个路段的长度为 l_1，第二个的长度为 l_2，…，以及第 k 个的长度 $\geqslant l_k$ 的概率，是

$$\det \begin{pmatrix} 1/l_1! & 1/(l_1+l_2)! & 1/(l_1+l_2+l_3)! & \cdots & 1/(l_1+l_2+l_3+\cdots+l_k)! \\ 1 & 1/l_2! & 1/(l_2+l_3)! & \cdots & 1/(l_2+l_3+\cdots+l_k)! \\ 0 & 1 & 1/l_3! & \cdots & 1/(l_3+\cdots+l_k)! \\ \vdots & \vdots & & & \vdots \\ 0 & 0 & \cdots & 1 & 1/l_k! \end{pmatrix}$$

9. 〔$M30$〕 设 $h_k(z) = \sum p_{km} z^m$，其中 p_{km} 是在一个随机（无穷）序列中头 k 个路段的总长度为 m 的概率。试求 $h_1(z)$、$h_2(z)$ 以及超生成函数 $h(z, x) = \sum_k h_k(z) x^k$ 的“简单”表达式。

10. 〔$HM30$〕 对于大的 k，求上题中的分布 $h_k(z)$ 的均值和方差的渐近特性。

11. 〔$M40$〕 设 $H_k(z) = \sum p_{km} z^m$，其中 p_{km} 是在一个随机（无穷）序列中第 k 个路段的长度为 m 的概率，试借助熟悉的函数来表达 $H_1(z)$、$H_2(z)$ 以及超生成函数 $H(z, x) = \sum_k H_k(z) x^k$。

12.〔$M33$〕（P. A. 麦克马洪）通过证明恰有 k 个路段的 $\{n_1\cdot 1,\ n_2\cdot 2,\ \cdots,\ n_m\cdot m\}$ 的排列数是

$$\sum_{0\leqslant j\leqslant k}(-1)^j\binom{n+1}{j}\binom{n_1-1+k-j}{n_1}\binom{n_2-1+k-j}{n_2}\cdots\binom{n_m-1+k-j}{n_m}$$

其中，$n=n_1+n_2+\cdots+n_m$，把等式（13）推广到一个多重集合的排列上。

13.〔05〕 如果西蒙·纽康姆的单人游戏用一副标准桥牌来进行，忽略面值，但令（♣ < ♦ < ♥ < ♠，问平均的堆数等于多少？

14.〔$M17$〕 排列 3 1 1 1 2 3 1 4 2 3 3 4 2 2 4 4 有 5 个路段；试按照前文关于麦克马洪对称性条件的构造求对应的有 9 个路段的排列。

15.〔$M21$〕（交错路段） 十九世纪经典的组合分析的著作都没有像我们这样考虑排列中的路段这个课题，但有若干文章涉及了交错地递增或递减的路段，例如 5 3 2 4 7 6 1 8 被认为有 4 个路段，即

5 3 2， 2 4 7， 7 6 1， 1 8

（按照 $a_1<a_2$ 或 $a_1>a_2$，头一个路段将是递增的或递减的；于是 $a_1a_2\cdots a_n$ 和 $a_n\cdots a_2a_1$ 以及 $(n+1-a_1)(n+1-a_2)\cdots(n+1-a_n)$ 都有相同的交错路段数。）当排列 n 个元素时，这种类型的路段的最大个数为 $n-1$。

试求在集合 $\{1,\ 2,\ \cdots,\ n\}$ 的一个随机排列中交错路段的平均数。〔提示：考虑（34）的证明。〕

16.〔$M30$〕 继续上一题，设 $\langle\langle {n \atop k}\rangle\rangle$ 是恰有 k 个交错路段的 $\{1,\ 2,\ \cdots,\ n\}$ 的排列数，试求一个递归关系，借助它可以计算一张 $\langle\langle {n \atop k}\rangle\rangle$ 的表，并求生成函数 $G_n(z)=\sum_k\langle\langle {n \atop k}\rangle\rangle z^k/n!$ 的对应递归关系。试利用后一递归关系来发现在 $\{1,\ 2,\ \cdots,\ n\}$ 的一个随机排列中交错路段个数方差的一个简单公式。

17.〔$M25$〕 若每个 a_j 是 0 或 1，则在所有 2^n 个序列 $a_1a_2\cdots a_n$ 当中，其恰有 k 个路段者（即，出现 $k-1$ 次 $a_j>a_{j+1}$）有多少？

18.〔$M27$〕 若每个 a_j 是在 $0\leqslant a_j\leqslant n-j$ 范围中的一个整数，则在所有 $n!$ 个序列 $a_1a_2\cdots a_n$ 中，恰有 k 个路段者（即，出现 $k-1$ 次 $a_j>a_{j+1}$）有多少？

▶19.〔$M26$〕（J·赖尔登）（a）在 $n\times n$ 的棋盘上可以有多少方式来放置 n 个不相拼（即，在同一行或同一列中没有两个）的车，使得在主对角线的给定一边内恰有 k 个？（b）在 $n\times n$ 的棋盘的主对角线给定一边内放置 k 个不相拼的车，可以有多少种方式？

例如，图 4 表示把八个不相拼的车放在标准棋盘上，其中，有三个车放在主对角线下不带阴影的部分，这是 15619 种放置方式之一，也是把三个不相拼的车放在一个三角棋盘上的 1050 种方式之一。

▶20.〔$M21$〕 如果在读一个排列的时候，必须从左到右扫描 k 次，才能在非减的次序下读出它的全部元素，则称这个排列是要求 k 次阅读的。例如，排列

4 9 1 8 2 5 3 6 7

要求 4 次阅读：头一次阅读时，得到 1，2，3；第二次阅读得到 4，5，6，7；然后 8，然

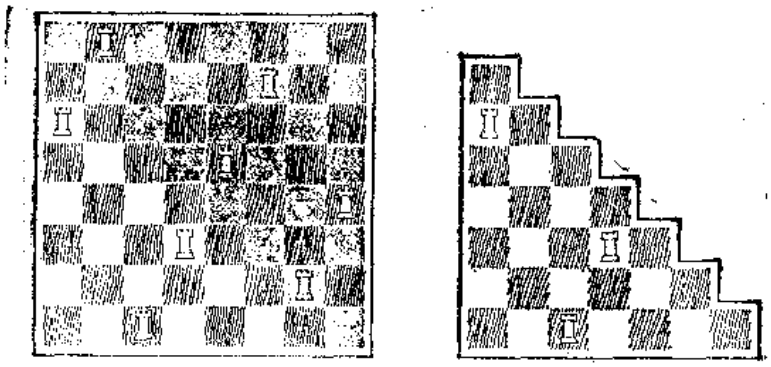


图 4　在一个棋盘上不相拼的车，且在主对角线之下有给定的个数

后 9。试求路段和阅读之间的联系。

21.〔$M22$〕 如果$\{1, 2, \cdots, n\}$的排列 $a_1a_2\cdots a_n$ 在习题 20 的意义下，有 k 个路段并要求 j 次阅读，则排列 $a_n\cdots a_2a_1$ 将如何？

22.〔$M26$〕〔L·卡利兹，D. P. 罗塞尔（D. P. Roselle），以及 R. A. 斯科维尔（R. A. Scoville）〕 试证，如果 $rs<n$，则不存在具有 $n+1-r$ 个路段，且要求 s 次阅读的$\{1, 2, \cdots, n\}$的排列；但如果 $n\geqslant n+1-r\geqslant s\geqslant 1$，$rs\geqslant n$，则这样的排列就存在。

23.〔$HM42$〕（沃尔特·韦斯布卢姆（Walter Weisblum））在排列 $a_1a_2\cdots a_n$ 中，每当一个区段停止单调时，即在该处放置一条竖线，这就得到此排列的“长路段”；长路段的递增或递减取决于它们的头两个元素的次序，所以每个长路段的长度（可能最后一个除外）都$\geqslant 2$。例如，7 5 | 6 2 | 3 8 9 | 1 4 有四个长路段。试求一个无穷排列的头两个长路段的平均长度，并证明长路段的极限长度是

$$\left(1+\cot\frac{1}{2}\right)\Big/\left(3-\cot\frac{1}{2}\right)\approx 2.4202$$

24.〔$M30$〕 如同习题 5.1.1-18 那样产生的序列中路段的平均数（作为 p 的一个函数）是多少？

*5.1.4　图表和对合

为了完成对于排列的组合性质的概述，我们将讨论把排列同整数阵列（称为图表）联系起来的某些值得注意的关系。杨氏$(n_1, n_2, \cdots, n_m)$形图表（其中 $n_1\geqslant n_2\geqslant\cdots\geqslant n_m\geqslant 0$）是把 $n_1+n_2+\cdots+n_m$ 个不同的整数排成一个阵列，其中第 i 行有 n_i 个元素，各行的左端是对齐的，行中的条款从左到右处于递增的次序，而且每列的条款从顶到下也是递增的。例如

1	2	5	9	10	15
3	6	7	13		
4	8	12	14		
11					

(1)

是一个杨氏(6, 4, 4, 1)形图表。这种排法是阿尔弗雷德·杨(Alfred Young)于 1900年为了研究排列的矩阵表示而引进的一个辅助手段〔例如，请见 D. E. Rutherford, *Substitutional Analysis* (New York: Hafner, 1968)〕。为了简便起见，把“杨氏图表”简称为“图表”。

一个对合是这样一种排列，它是自己的逆，例如，{1，2，3，4}的对合共有10个

$$\begin{pmatrix}1&2&3&4\\1&2&3&4\end{pmatrix}\begin{pmatrix}1&2&3&4\\2&1&3&4\end{pmatrix}\begin{pmatrix}1&2&3&4\\3&2&1&4\end{pmatrix}\begin{pmatrix}1&2&3&4\\4&2&3&1\end{pmatrix}\begin{pmatrix}1&2&3&4\\1&3&2&4\end{pmatrix}$$
$$\begin{pmatrix}1&2&3&4\\1&4&3&2\end{pmatrix}\begin{pmatrix}1&2&3&4\\1&2&4&3\end{pmatrix}\begin{pmatrix}1&2&3&4\\2&1&4&3\end{pmatrix}\begin{pmatrix}1&2&3&4\\3&4&1&2\end{pmatrix}\begin{pmatrix}1&2&3&4\\4&3&2&1\end{pmatrix}\qquad(2)$$

"对合"这一术语起源于古典的几何问题；这里考虑的是一般意义下的对合，这是由H. A. 罗瑟，在他引进逆的概念的时候，首先研究的（见5.1.1节）。

我们同时讨论图表和对合可能显得有点奇怪，但是在这两个表面上无关的概念之间，实际上有着非同寻常的联系：{1，2，…，n}的对合的个数和由元素{1，2，…，n}所能形成的图表数相同。例如，由{1，2，3，4}恰能形成10张图表，即

$$\begin{array}{|c|c|c|c|}\hline 1&2&3&4\\\hline\end{array}\quad\begin{array}{|c|c|c|}\hline 1&3&4\\\hline 2\\\cline{1-1}\end{array}\quad\begin{array}{|c|c|}\hline 1&4\\\hline 2\\\cline{1-1} 3\\\cline{1-1}\end{array}\quad\begin{array}{|c|c|}\hline 1&3\\\hline 2\\\cline{1-1} 4\\\cline{1-1}\end{array}\quad\begin{array}{|c|c|c|}\hline 1&2&4\\\hline 3\\\cline{1-1}\end{array}$$
$$\begin{array}{|c|c|}\hline 1&2\\\hline 3\\\cline{1-1} 4\\\cline{1-1}\end{array}\quad\begin{array}{|c|c|c|}\hline 1&2&3\\\hline 4\\\cline{1-1}\end{array}\quad\begin{array}{|c|c|}\hline 1&3\\\hline 2&4\\\hline\end{array}\quad\begin{array}{|c|c|}\hline 1&2\\\hline 3&4\\\hline\end{array}\quad\begin{array}{|c|}\hline 1\\\hline 2\\\hline 3\\\hline 4\\\hline\end{array}\qquad(3)$$

分别对应于10个对合（2）。

对合和图表之间的这一联系决不是显然的，而且大概没有非常简单的办法来证明它。我们将要讨论的证明包含一个非常有趣的图表构造算法，它有若干其它意外的性质；该算法是以把新元素插入图表的一个特殊过程为基础的。

例如，假设要对图表

$$\begin{array}{|c|c|c|c|c|c|}\hline 1&3&5&9&12&16\\\hline 2&6&10&15\\\cline{1-4} 4&13&14\\\cline{1-3} 11\\\cline{1-1} 17\\\cline{1-1}\end{array}\qquad(4)$$

插入元素8。所使用的方法是，开始把8放入行1，放在原来为9所占据的小格上，因为9是在该行中大于8的最小元素，元素9就被"撞"到行2，在那里它代替10。然后10把13从行3"撞"到行4；而且由于行4没有大于13的元素，这一过程遂以把13插入到行4的右端而终止。至此这个图表已经变换成

$$\begin{array}{|c|c|c|c|c|c|}\hline 1&3&5&8&12&16\\\hline 2&6&9&15\\\cline{1-4} 4&10&14\\\cline{1-3} 11&13\\\cline{1-2} 17\\\cline{1-1}\end{array}\qquad(5)$$

这个过程的精确描述在算法 I 中给出，同时还证明此过程始终保持图表性质。

算法 I（插进图表） 设 $P=(P_{ij})$ 是一张正整数的图表，又设 x 是不在 P 中的一个正整数。这个算法把 P 变换成为另外一张图表，它除了含有 P 原来的元素外，还含有 x。这张新的图表除了在第 s 行第 t 列增加一个新的位置外，形式和旧的图表相同，这里 s 和 t 是由本算法所确定的量。

（在本算法中带圆括弧的注释用来证明它的正确性，因为容易归纳地验证这些注释是正确的，以及在整个过程中阵列 P 保持为一图表。为了方便起见，假定这图表已被镶了边，在顶上和左边有一些 0 并在右边和底下有一些∞，使得对于所有的 i，$j\geqslant 0$，P_{ij} 都有定义。如果定义关系

$$a\lesssim b \quad 当且仅当\ a<b\ 或\ a=b=\infty \tag{6}$$

则这些图表不等式即可以下列方便的形式加以表达

$$P_{ij}=0\ 若\ i=0\ 或\ j=0$$

$$P_{ij}\lesssim P_{i(j+1)}\ 和\ P_{ij}\lesssim P_{(i+1)j}\ 对所有\ i,\ j\geqslant 0 \tag{7}$$

语句“$x\notin P$”意味着或 $x=\infty$ 或对于所有 i，$j\geqslant 0$，$x\neq P_{ij}$。）

I1.〔输入 x〕 置 $i\leftarrow 1$，置 $x_1\leftarrow x$，并置 j 为使得 $P_{ij}=\infty$ 的最小值。

I2.〔求 x_{i+1}〕（这时 $P_{(i-1)j}<x_i<P_{ij}$ 且 $x_i\notin P$） 如果 $x_i<P_{i(j-1)}$，则 j 减 1 并重复这一步骤。否则，置 $x_{i+1}\leftarrow P_{ij}$ 且置 $r_i\leftarrow j$。

I3.〔代以 x_i〕（现在 $P_{i(j-1)}<x_i<x_{i+1}=P_{ij}\lesssim P_{i(j+1)}$，$P_{(i-1)j}<x_i<x_{i+1}=P_{ij}\lesssim P_{(i+1)j}$，且 $r_i=j$。） 置 $P_{ij}\leftarrow x_i$。

I4.〔$x_{i+1}=\infty$ 吗？〕（现在 $P_{i(j-1)}<P_{ij}=x_i<x_{i+1}\lesssim P_{i(j+1)}$，$P_{(i-1)j}<P_{ij}=x_i<x_{i+1}\lesssim P_{(i+1)j}$，$r_i=j$，且 $x_{i+1}\notin P$。）若 $x_{i+1}\neq\infty$，i 增 1，并返回步骤 I2。

I5.〔确定 s，t〕 置 $s\leftarrow i$，$t\leftarrow j$，并终止此算法。（此时，满足条件

$$P_{st}\neq\infty,\ P_{(s+1)t}=P_{s(t+1)}=\infty \tag{8}$$

注意这个算法定义了一个“碰撞序列”

$$x=x_1<x_2<\cdots<x_s<x_{s+1}=\infty \tag{9}$$

和一个辅助的列标序列

$$r_1\geq r_2\geq\cdots\geq r_s=t \tag{10}$$

对于 $1\leqslant i\leqslant s$，元素 P_{ir_i} 已经从 x_{i+1} 变成为 x_i。例如，当把 8 插到（4）中时，碰撞序列是 8，9，10，13，∞，且辅助序列为 4，3，2，2。本可用另一种形式这个算法，使得它使用更少的临时存储（仅仅需要记住 j、x_i 和 x_{i+1} 的当前值），但还是引入了序列（9）和（10），为的是能够证明关于这个算法的一些有趣的事情。

关于算法 I，将要使用的关键事实在于它能够被“逆转运行”；给定在步骤 I5 中确定的 s 和 t 的值，能够倒过来把 P 转换成为它原来的形式，并确定和消去被插入的元素 x。例如，考虑（5）并假设告诉我们元素 13 是在原为空格的位置上，则 13 必然被 10 从行 3 撞下的，因为 10 是在该行中小于 13 的最大元素；类似地，10 必然是被 9 从行 2 撞下

的，而9必然是被8从行1撞下的。于是，可以从（5）倒回到（4）。下列算法详细地说明这一过程：

算法D（从一图表删去） 给定一个图表P以及满足（8）的正整数s，t，本算法把P变换成另一个具有几乎相同形状的图表，但在第s行第t列处具有∞。由本算法所确定的一个元素x被从P删去。

（如同在算法I中一样，这里加上了带圆括弧的断言，以便于证明在整个过程中P保持为图表。）

D1.〔输入s，t〕 置$j \leftarrow t$，$i \leftarrow s$，$x_{j+1} \leftarrow \infty$。

D2.〔求x_i〕（这时$P_{ij} < x_{i+1} \lesssim P_{(i+1)j}$且$x_{i+1} \notin P$。）如果$P_{i(j+1)} < x_{i+1}$，则$j$增加1并重复这一步骤。否则，置$x_i \leftarrow P_{ij}$且$r_i \leftarrow j$。

D3.〔代以x_{i+1}〕（现在$P_{i(j-1)} < P_{ij} = x_i < x_{i+1} \lesssim P_{i(j+1)}$，$P_{(i-1)j} < P_{ij} = x_i < x_{i+1} \lesssim P_{(i+1)j}$，且$r_i = j$。）置$P_{ij} \leftarrow x_{i+1}$。

D4.〔$i = 1$吗？〕（现在$P_{i(j-1)} < x_i < x_{i+1} = P_{ij} \lesssim P_{i(j+1)}$，$P_{(i-1)j} < x_i < x_{i+1} = P_{ij} \lesssim P_{(i+1)j}$，且$r_i = j$。）如果$i > 1$，则$i$减1并返回步骤D2。

D5.〔确定x〕 置$x \leftarrow x_1$并终止这个算法（现在$0 < x < \infty$）。

在算法I和算法D中出现的带圆括弧的断言不仅是证明这些算法保持图表结构的有用方式，也有利于验证算法I和D恰恰互相为逆。给定某个图表P和某个正整数$x \notin P$，如果首先实施算法I，它将插入x，并确定满足（8）的正整数s，t；对此结果应用算法D，则将重新算出x并将恢复P。反过来，给定某个图表P和某个满足（8）的整数s，t，如果首先实施算法D，它将修改P，删去某个正整数x；对此结果应用算法I，将重新算出s、t并将恢复P。原因是步骤I3和D4的带圆括弧的断言是相同的，步骤I4和D3的断言也一样，而且这些断言唯一地表征j的值。因此辅助序列（9）、(10)在每种情况下都是一样的。

现在已作好准备来证明图表的一个基本性质。

定理A 在$\{1, 2, \cdots, n\}$的所有排列的集合与由$\{1, 2, \cdots, n\}$形成的图表的有序对偶(P, Q)的集合之间，有一对一的对应，其中P和Q有相同的形状。

（这个定理的一个例子出现于下边的证明内。）

证明 宜于证明一个稍微更一般的结果。给定任何两行的阵列

$$\begin{pmatrix} q_1 & q_2 \cdots q_n \\ p_1 & p_2 \cdots p_n \end{pmatrix}, \quad \begin{matrix} q_1 < q_2 < \cdots < q_n \\ p_1, p_2, \cdots, p_n \text{不同} \end{matrix} \tag{11}$$

我们将构造两个对应的图表P和Q，其中P的元素是$\{p_1, \cdots, p_n\}$，而Q的元素是$\{q_1, \cdots, q_n\}$，并且P的形状即是Q的形状。

设P和Q开始是空的。然后，对于$i = 1, 2, \cdots, n$（以这个次序）作下列操作：利用算法I把p_i插入图表P；然后置$Q_{st} \leftarrow q_i$，其中s和t确定P中的新被填入的位置。

例如，如果给定的排列是$\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$，我们得到

$$
\begin{array}{lll}
 & P & Q \\
\text{插入 7：} & \begin{array}{|c|}\hline 7 \\ \hline\end{array} & \begin{array}{|c|}\hline 1 \\ \hline\end{array} \\
\text{插入 2：} & \begin{array}{|c|}\hline 2 \\ \hline 7 \\ \hline\end{array} & \begin{array}{|c|}\hline 1 \\ \hline 3 \\ \hline\end{array} \\
\text{插入 9：} & \begin{array}{|c|c|}\hline 2 & 9 \\ \hline 7 \\ \cline{1-1}\end{array} & \begin{array}{|c|c|}\hline 1 & 5 \\ \hline 3 \\ \cline{1-1}\end{array} \\
\text{插入 5：} & \begin{array}{|c|c|}\hline 2 & 5 \\ \hline 7 & 9 \\ \hline\end{array} & \begin{array}{|c|c|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline\end{array} \\
\text{插入 3：} & \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 \\ \cline{1-1}\end{array} & \begin{array}{|c|c|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 \\ \cline{1-1}\end{array}
\end{array}
\tag{12}
$$

于是对应于$\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$的图表（$P, Q$）是

$$
P = \begin{array}{|c|c|}\hline 2 & 3 \\ \hline 5 & 9 \\ \hline 7 \\ \cline{1-1}\end{array}, \qquad Q = \begin{array}{|c|c|}\hline 1 & 5 \\ \hline 3 & 6 \\ \hline 8 \\ \cline{1-1}\end{array} \tag{13}
$$

按这种构造法，显然P和Q总有相同的形状；其次，由于总是以递增的次序在Q的周围增加元素，故Q是一个图表。

反之，给定两个相同形状的图表P和Q，我们可以求相应的两行阵列（11）如下。设Q的元素是

$$q_1 < q_2 < \cdots < q_n$$

对于$i = n, \cdots, 2, 1$（以这个次序），令p_i是当应用算法D于P，并且用使得$Q_{st} = q_i$的值s、t时，被消去的元素x。

例如，这个构造将以（13）开始并将逐次地做（12）的反运算直到P为空，这就得到$\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$。

由于算法1和D彼此互逆，故已经描述的两个构造彼此互逆，而且建立了一一对应。

在定理A的证明中，定义的对应关系有许多惊人的性质，现在就着手来推导其中的某些。读者不妨亲手试试习题1中的例子，以便在往下阅读之前能熟悉这个构造。

一旦已经从行1把一个元素撞下到行2中，它就不再影响行1了；而且由“被撞”元素序列来构造行2，3，…的方法恰恰就象由原来的排列来构造行1，2，…一样。这些事实提示我们，可以用另一种方式来考察定理A的构造，即仅专注于P和Q的开头几行。例如，排列$\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$引起在行1中发生下列动作(参考(12))：

$$\begin{aligned} &1：插入7，置Q_{11}\leftarrow 1。\\ &3：插入2，撞下7。\\ &5：插入9，置Q_{12}\leftarrow 5。\\ &6：插入5，撞下9。\\ &8：插入3，撞下5。\end{aligned} \tag{14}$$

于是P的头一行是2 3，而Q的头一行是1 5，其次，P和Q中剩下的行是对应于“被撞下”的两行阵列

$$\begin{pmatrix} 3 & 6 & 8 \\ 7 & 9 & 5 \end{pmatrix} \tag{15}$$

的图表。为了研究行1上构造的特性，可以考察进入该行的一个给定列的诸元素。我们说(q_i, p_i)相对于两行阵列

$$\begin{pmatrix} q_1 & q_2 \cdots q_n \\ p_1 & p_2 \cdots p_n \end{pmatrix}, \quad \begin{matrix} q_1<q_2<\cdots<q_n \\ p_1, p_2, \cdots, p_n \text{ 不同} \end{matrix} \tag{16}$$

来说是在类t中，如果从一个空表格P开始，在逐次应用算法Ⅰ于$p_1, p_2, \cdots, p_i$以后，$p_i=p_{1t}$的话（记住算法Ⅰ总是把给定的元素插入到行1中）。

容易看出，当且仅当p_i有$i-1$个反序，即，$p_i=\min\{p_1, p_2, \cdots, p_i\}$是“自左到右的极小值”时，$(q_i p_i)$在类1中。如果删去(16)中属于类1的诸列，则就得到另一个两行阵列

$$\begin{pmatrix} q'_1 & q'_2 \cdots q'_m \\ p'_1 & p'_2 \cdots p'_m \end{pmatrix} \tag{17}$$

使得当且仅当(q, p)相对于(17)来说是在类t中时，它相对于(16)来说是在类$t+1$中。从(16)进行到(17)的操作表示撤销行1最左的位置。这给了一个确定这些类的系统方法。例如，在$\begin{pmatrix} 1 & 3 & 5 & 6 & 8 \\ 7 & 2 & 9 & 5 & 3 \end{pmatrix}$中，自左到右极小的元素是7和2，所以类1是$\{(1, 7), (3, 2)\}$；在剩下的阵列$\begin{pmatrix} 5 & 6 & 8 \\ 9 & 5 & 3 \end{pmatrix}$中，所有的元素都是极小的，所以类2是$\{(5, 9), (6, 5), (8, 3)\}$。在“被撞下”的阵列(15)中，类1是$\{(3, 7), (8, 5)\}$，而类2是$\{(6, 9)\}$。

对于任何固定的t值，类t的元素可以标记成

$$(q_{i_1}, p_{i_1}), \cdots, (q_{i_k}, p_{i_k})$$

其中

$$q_{i_1}<q_{i_2}<\cdots<q_{i_k}$$
$$p_{i_1}>p_{i_2}>\cdots>p_{i_k} \tag{18}$$

因为随着插入算法的进行，图表位置 P_{1t} 取的值是递减序列 $p_{i_1}, \cdots, p_{i_k}$。在构造结束时，有

$$P_{1t}=p_{i_k},\ Q_{1t}=q_{i_1} \tag{19}$$

P 和 Q 的第 2，3，…诸行是通过“被撞下来的”两行阵列定义的，这个阵列包含

$$\begin{pmatrix} q_{i_2}, & q_{i_3}, & \cdots, & q_{i_k} \\ p_{i_1}, & p_{i_2}, & \cdots, & p_{i_{k-1}} \end{pmatrix} \tag{20}$$

诸列，加上以类似方式从其它类形成的其它列。

这些考察导致了通过手算来算出 P 和 Q 的一个简单的方法（见习题 3），而且它们也为我们提供了证明一个颇为意外的结果的手段：

定理 B 如果排列

$$\begin{pmatrix} 1 & 2\cdots n \\ a_1 & a_2\cdots a_n \end{pmatrix}$$

对应于定理 A 的构造中的图表（P，Q），则逆排列对应于（Q，P）。

这项事实是十分惊人的，因为 P 和 Q 在定理 A 中是通过完全不同的方法形成的，而且一个排列的逆是通过反复变幻两行阵列的诸列而得到的。

证明 假设有一个两行阵列（16）；交换这两行，并重排诸列，使得新的上边一行以递增次序出现，就得到“逆”阵列

$$\begin{pmatrix} q_1 & q_2\cdots q_n \\ p_1 & p_2\cdots p_n \end{pmatrix}^{-1}=\begin{pmatrix} p_1 & p_2\cdots p_n \\ q_1 & q_2\cdots q_n \end{pmatrix}=\begin{pmatrix} p'_1 & p'_2\cdots p'_n \\ q'_1 & q'_2\cdots q'_n \end{pmatrix},\quad \begin{matrix} p'_1<p'_2<\cdots<p'_n \\ q'_1, q'_2, \cdots q'_n \text{ 不同} \end{matrix} \tag{21}$$

我们将证明这项操作对应于在定理 A 的构造中交换 P 和 Q。

习题 2 已重新叙述了如何确定类的注释，使得（q_i，p_i）的类不依赖于 q_1，q_2，…，q_n 是以递增次序出现的这一事实。由于得到的条件在诸 q 和诸 p 中是对称的，故操作(21)不破坏类的结构；如果（q，p）相对于（16）来说是处于类 t 中，则（p，q）相对于(21)来说就处于类 t 中。如果因此把后一类 t 的元素排成

$$p_{i_k}<\cdots<p_{i_2}<p_{i_1}$$
$$q_{i_k}>\cdots>q_{i_2}>q_{i_1} \tag{22}$$

〔参考(18)〕则如同在（19）中一样，就有

$$P_{1t}=q_{i_1},\ Q_{1t}=p_{i_k} \tag{23}$$

而且诸列

$$\begin{pmatrix} p_{i_{k-1}}\cdots p_{i_2} & p_{i_1} \\ q_{i_k} \quad \cdots q_{i_3} & q_{i_2} \end{pmatrix} \tag{24}$$

如同在（20）中一样进入到“撞下的”阵列中。因此 P 和 Q 的前几行被交换。而且（21）的“撞下的”两行阵列是（16）的“撞下的”两行阵列的逆，所以，通过对图表中的行数用归纳法，就完成了这一证明。

推论 由 $\{1, 2, \cdots, n\}$ 所能形成的图表的数目是 $\{1, 2, \cdots, n\}$ 上的对合的数目。

证明 如果 π 是对应于 (P, Q) 的一个对合，则 $\pi=\pi^{-1}$ 对应于 (Q, P)；因此 $P=Q$。反之，如果 π 是对应于 (P, P) 的任何排列，则 π^{-1} 也对应于 (P, P)，因此 $\pi=\pi^{-1}$。所以，在对合 π 和图表 P 之间有一一对应。

显然，一个图表的左上角的元素总是最小的。这提示了对一个数集进行排序的一种可能的途径：首先，可以通过反复地使用算法 I，把它们放置到一个图表当中；这就把最小的元素放到角上。然后删去这个最小的元素，并重新安排剩余的元素，使得它们形成另一个图表；然后删去新的最小元素；等等。

因此，让我们来考虑当从图表

1	3	5	8	12	16
2	6	9	15		
4	10	14			
11	13				
17					

(25)

删去角元素时，将出现什么情况。如果删去 1，则 2 必须取代它的位置。然后可以向上移动 4 到 2 原来所在的位置，但不能把 11 移动到 4 原来所在的位置；而是移动 10，然后用 13 代替 10。如此下去，就得到如下的过程：

算法 S （删去角元素） 给定一个图表 P，本算法删去 P 的左上角元素，并且移动其它的元素，使得 P 仍保持图表的性质。这里沿用算法 I 和 D 的记号约定。

S1.〔初始化〕 置 $r \leftarrow 1$，$s \leftarrow 1$。

S2.〔完成？〕 如果 $P_{rs}=\infty$，则此过程完成。

S3.〔比较〕 如果 $P_{(r+1)s} \lesssim P_{r(s+1)}$，则转到步骤 S5。（考察恰在空格右方和下方的元素，并移动其中较小者。）

S4.〔左移〕 置 $P_{rs} \leftarrow P_{r(s+1)}$，$s \leftarrow s-1$，并返回 S3。

S5.〔上移〕 置 $P_{rs} \leftarrow P_{(r+1)s}$，$r \leftarrow r+1$，并返回 S2。

容易证明，在算法 S 已经删去了 P 的角元素之后，P 仍然是一个图表（见习题 10）。所以如果重复算法 S 直到 P 为空，则就能以递增的次序读出它的元素。可惜，这并不是一个象将看到的其它方法一样有效的排序算法；它的极小运行时间与 $n^{1.5}$ 成比例，而使用树形而不是使用图表结构的类似算法，却有阶为 $n \log_2 n$ 的执行时间。

尽管算法 S 不能导致一个特别有效的排序算法，但它却有某些有趣的性质。

定理 C （M. P. 舒曾伯杰） 如果 P 是由排列 $a_1, a_2, \cdots, a_n$ 通过定理 A 的构造形成的图表，而且如果 $a_i=\min\{a_1, a_2, \cdots, a_n\}$，则算法 S 把 P 变成为对应于 $a_1 \cdots a_{i-1} a_{i+1} \cdots a_n$ 的图表。

证明 见习题13。

在应用算法 S 于一图表之后，把删去的元素放置到刚刚空出来的位置 P_{rs} 中，并对它

加圆括弧以指出它实际上并不是这个图表的一部分。例如，在应用这一过程于（25）之后，将有

2	3	5	8	12	16
4	6	9	15		
10	13	14			
11	(1)				
17					

再应用两次，得出

4	5	8	12	16	(2)
6	9	14	15		
10	13	(3)			
11	(1)				
17					

继续进行，直到所有元素都已加了圆括弧，然后去掉这些圆括弧，给出

17	15	14	13	11	2
16	10	6	4		
12	5	3			
9	1				
8					

(26)

它同原来的图表（25）有同样的形状。这个结构可以称作一个对偶图表，因为它除了使用“对偶次序”（逆转<和>的作用）这一点外，其它都象一个图表。我们以符号 P^s 表示以这种方式由 P 形成的对偶图表。

由 P^s 可以唯一地确定 P；事实上，通过应用完全相同的算法，可以从 P^s 得到原来的图表 P（逆转次序，因为 P^s 是一个对偶图表）。例如，应用这个算法于（26）两步，给出

15	14	13	11	2	(16)
12	10	6	4		
9	5	3			
8	1				
(17)					

终于将重新产生出（25）来！这个值得注意的事实是下一定理的推论之一。

定理D （C. 申斯迪德（C. Schensted），M. P. 舒普伯杰）

设

$$\begin{pmatrix} q_1 & q_2 \cdots q_n \\ p_1 & p_2 \cdots p_n \end{pmatrix} \tag{27}$$

是对应于图表（P, Q）的两行阵列。

a）对诸 q，而不是对诸 p 应用对偶（逆转）次序，则两行阵列

$$\begin{pmatrix} q_n \cdots q_2 & q_1 \\ p_n \cdots p_2 & p_1 \end{pmatrix} \tag{28}$$

对应于（P^T, $(Q^s)^T$）。

（像通常一样，"T" 表示对行和列进行转置的操作；P^T 是一个图表，而 $(Q^s)^T$ 是一个对偶图表，因为诸 q 的次序被逆转了。）

b）对诸 p 而不是对诸 q 应用对偶次序，则两行阵列（27）对应于（$((P^s)^T$, $Q^T)$。

c）对诸 p 和诸 q 同时应用对偶次序，则两行阵列（28）对应于（P^s, Q^s）。

证明　还不知道这个定理的简单证明。关于情况（a）对应于（P^T, X）这一事实，在习题 6 中有证明，其中 X 是某个对偶图表；因此由定理B，情况（b）对应于（Y, Q^T），这里 Y 也是某个对偶图表，且必然有 P^T 的形状。

设 $p_i = \min\{p_1, \cdots, p_n\}$；由于 p_i 是对偶次序下"最大的"元素，它出现于 Y 的外围上，而且它不撞下定理A的构造中的任何元素。于是，如果利用对偶次序，逐次地插入 $p_1, \cdots, p_{i-1}$, $p_{i+1}, \cdots, p_n$，则就得到 $Y - \{p_i\}$，即除去了 p_i 的 Y。由定理C，如果利用正常的次序，逐次地插入 $p_1, \cdots, p_{i-1}, p_{i+1}, \cdots, p_n$，则得到对 p 使用算法S而得到的图表 $d(P)$。对 n 用归纳法，$Y - \{p_i\} = (d(p)^s)^T$。但由于按操作 s 的定义

$$(P^s)^T - \{p_i\} = (d(p)^s)^T \tag{29}$$

以及由于 Y 有和 $(P^s)^T$ 同样的形状，必然有 $Y = (P^s)^T$。

这就证明了（b），而且通过应用定理B而得出（a）。相继地应用（a）和（b），可证明情况（c）对应于（$((P^T)^s)^T$, $((Q^s)^T)^T$）；而这就是（P^s, Q^s），因为由操作 s 的行、列对称性可知 $(P^s)^T = (P^T)^s$。

特别是，这个定理证实了关于图表插入算法的两个惊人的事实：如果逐次插入不同的元素 $p_1, \cdots, p_n$ 到一个空图表中可产生图表 P，则按相反的次序插入 $p_n, \cdots, p_1$ 就产生转置图表 P^T。而如果不仅以次序 $p_n, \cdots, p_1$ 插入诸 p，并且在插入过程中也交换〈和〉以及0和∞的作用，则就得到对偶图表 P^s。读者不妨用一些简单的例子来试验这些过程。这些巧合的不寻常的属性可能使我们猜测：在舞台后边有某些魔法在操纵！还不知道关于这些现象的简单的解释；看起来甚至没有显然的方式来证明下列事实：情况（c）对应于与 P 和 Q 有相同形状的图表。

G. de B. 鲁滨逊（G. de B. Robinson）〔*American J. Math*, 60(1938), 745-760, *Sec* 5〕以有些含混和不同的形式，给出了定理A中的对应关系，作为解决群论中颇为困难的问题的一部分。不难验证，他的构造实质上等同于这里所给出的构造。他未加证明地叙述了定理B。许多年后，C. 申斯迪德独立地重新发现了这个对应关系，他所指出的这一对应关系实质上是以已经使用的形式给出的〔*Canadian J. Math.* 13（1961），179-191〕。

他也证明了定理D（a）的“P”部分。M·P·舒曾伯杰〔*Math. Scand.* 12 (1936), 117-128〕证明了定理B和定理D（a）的“Q”部分，由它就得出了（b）和（c）。有可能把这个对应关系推广到多重集合的排列；申斯迪德考虑了 $p_1, \cdots, p_n$ 不必是不同的情况，而克努特研究了“最终”推广到诸 p 和诸 q 两者皆可以包含重复元素的情况〔*Pacific J. Math.* 34 (1970), 709-727〕。

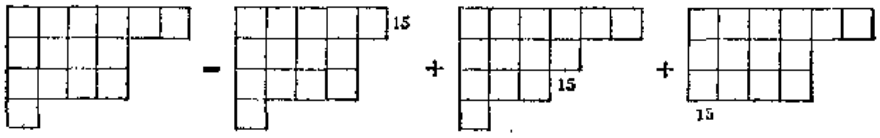
现在我们转到一个有关的问题：具有一个给定形状 $(n_1, n_2, \cdots, n_m)$ 的由 $\{1, 2, \cdots, n\}$ 形成的图表共有多少，其中 $n_1+n_2+\cdots+n_m=n$？如果用 $f(n_1, n_2, \cdots, n_m)$ 来表示这个数，则它必然满足关系

$$f(n_1, n_2, \cdots, n_m)=0 \text{ 除非} n_1 \geqslant n_2 \geqslant \cdots \geqslant n_m \geqslant 0 \tag{30}$$

$$f(n_1, n_2, \cdots, n_m, 0)=f(n_1, n_2, \cdots, n_m) \tag{31}$$

$$\begin{aligned} f(n_1, n_2, \cdots, n_m) &= f(n_1-1, n_2, \cdots, n_m)+f(n_1, n_2-1, \cdots, n_m) \\ &\quad +\cdots+f(n_1, n_2, \cdots, n_m-1) \quad \text{如果} n_1 \geqslant n_2 \geqslant \cdots \geqslant n_m \geqslant 1 \end{aligned} \tag{32}$$

后边的递归式得自这样一个事实，即撤销一个图表的最大元素后总是得到另一个图表；例如，形状为（6，4，4，1）的图表的个数是 $f(5,4,4,1)+f(6,3,4,1)+f(6,4,3,1)+f(6,4,4,0)=f(5,4,4,1)+f(6,4,3,1)+f(6,4,4)$，因为 $\{1, 2, \cdots, 15\}$ 上形状为（6，4，4，1）的每个图表均由插入元素 15 到形状为（5，4，4，1），（6，4，3，1）或（6，4，4）的一个图表的适当位置而形成。如下图。



(33)

满足这些关系的函数 $f(n_1, n_2, \cdots, n_m)$ 有一种相当简单的形式

$$f(n_1, n_2, \cdots, n_m)=\frac{\Delta(n_1+m-1, n_2+m-2, \cdots, n_m)\, n!}{(n_1+m-1)!\,(n_2+m-2)!\cdots n_m!},$$

$$\text{对于 } n_1+m-1 \geqslant n_2+m-2 \geqslant \cdots \geqslant n_m \tag{34}$$

式中 Δ 表示“判别式的平方根”函数

$$\Delta(x_1, x_2, \cdots, x_m)=\det\begin{pmatrix} x_1^{m-1} & x_2^{m-1} & \cdots & x_m^{m-1} \\ \vdots & & & \\ x_1^2 & x_2^2 & & x_m^2 \\ x_1 & x_2 & & x_m \\ 1 & 1 & \cdots & 1 \end{pmatrix}=\prod_{1 \leqslant i < j \leqslant m}(x_i-x_j) \tag{35}$$

公式（34）是 G. 弗罗比尼厄斯（G. Frobenius）在解决群论中的一个等价问题时导出的〔*Sitzungsberichte Preuss. Akad. der Wissenchaften* (Berlin, 1900), 516-534, Sec 3〕，并且利用了一个颇为深入的群论上的论证。麦克马洪独立地给出了一个组合的证明〔*Philosophical Tran. A*-209 (London, 1909), 153-175〕。这个公式可以通过归纳法来建立，因为关系（30）和（31）是容易证明的，而且在习题 17 的恒等式中置 $y=-1$ 就得出（32）。

定理A给出了与图表数的这个公式有关的一个值得注意的恒等式。如果对所有形状求和，则有

$$n!=\sum_{\substack{k_1\geqslant\cdots\geqslant k_n\geqslant 0\\ k_1+\cdots+k_n=n}} f(k_1,\cdots,k_n)^2=n!^2\sum_{\substack{k_1\geqslant\cdots\geqslant k_n\geqslant 0\\ k_1+\cdots+k_n=n}}\frac{\Delta(k_1+n-1,\cdots,k_n)^2}{(k_1+n-1)!^2\cdots k_n!^2}$$

$$=n!^2\sum_{\substack{q_1>q_2>\cdots>q_n\geqslant 0\\ q_1+\cdots+q_n=(n+1)n/2}}\frac{\Delta(q_1,\cdots,q_n)^2}{q_1!^2\cdots q_n!^2}$$

因此

$$\sum_{\substack{q_1+\cdots+q_n=(n+1)n/2\\ q_1,\ \cdots,\ q_n\geqslant 0}}\frac{\Delta(q_1,\cdots,q_n)^2}{q_1!^2\cdots q_n!^2}=1 \tag{36}$$

在后面这个和式中，已经除去了不等式 $q_1>q_2>\cdots>q_m$，因为被加数是诸 q 的对称函数，当 $q_i=q_j$ 时它为 0。一个类似的恒等式在习题 24 中出现。

利用“钩子”的思想，可以用一种更有趣的方式来表达求图表个数的公式。对应于图表中的每个小格，可定义钩子为小格本身加上位于其下边和右边的诸小格。例如，图 5的阴影区域是对应于行 2、列 3 的小格（2，3）的钩子；它包含六个小格。图 5 的每个小格已经填以它的钩子的长度。

12	11	8	7	5	4	1
10	9	6	5	3	2	•
9	8	5	4	2	1	•
6	5	2	1	•		
3	2	•				
2	1	•				

图 5　钩子和钩子的长度

如果图表的形状是 $(n_1,n_2,\cdots,n_m)$，且 $n_m\geqslant 1$，则最长的钩子的长度是 n_1+m-1。对钩子长度作进一步的考察表明，行 1 包含所有长度 n_1+m-1，$n_1+m-2,\cdots,1$，但不包括 $(n_1+m-1)-(n_m)$，$(n_1+m-1)-(n_{m-1}+1),\cdots,(n_1+m-1)-(n_2+m-2)$。例如，在图 5 中，行 1 的钩子长度是12，11，8，…，1 而不包括 10，9，6，3，2；这些例外对应于五个不存在的钩子，即从不存在的诸小格（6，3），（5，3），（4，5），（3，7），（2，7）通到小格（1，7）的钩子。类似地，行 j 包含有所有的钩子长度 $n_j+m-j,\cdots,1$，但不包括 $(n_j-m-j)-(n_m),\cdots,(n_j-m-j)-(n_{j+1}-m-j-1)$。由此得出所有钩子长度的乘积等于

$$(n_1+m-1)!\cdots(n_m)!/\Delta(n_1+m-1,\cdots,n_m)$$

这恰是等式（34）中的内容，所以有

定理 H　〔J. S. 弗雷姆（J. S. Frame），G. de B. 鲁滨逊，R. M. 思罗尔（R. M. Thrall）〕　$\{1,2,\cdots,n\}$上的有一个确定形状的图表数是 $n!$ 除以钩子长度的乘积。

既然是如此简单的结果，它应该有一个简单的证明；但是（像大多数已知的关于图表的事实一样），尚不知更为直接的证明。一个图表的每个元素是它的钩子中的最小者；如果随机地填图表，则小格（i，j）包含对应的钩子的极小元素的概率是钩子长度的倒数。对于所有的 i 和 j 求这些概率的乘积即给出定理H；但这个论证是荒谬的，因为这些概率远不是独立的。定理C的所有已知的证明都是根据一种平凡的归纳论证，它并没有真正阐明为什么这个定理是正确的（因为它没真正使用钩子的性质）。

定理H同树的枚举有着有趣的联系，在第二章中已考虑了树的枚举问题。我们已经注意到，具有 n 个节点的二叉树对应于用一个栈所能得到的排列，而且这类排列对应于诸 S

和诸X的那些序列 $a_1a_2\cdots a_{2n}$，其中当自左向右阅读时，S的个数决不少于X的个数（见习题2.3.1-6和2.2.1-3）。后面的那些序列以自然的方式对应于形状为（n，n）的图表；我们在行1放置使得 $a_i=$ S的下标 i，而在行2中放置使得 $a_i=$ X的下标 i。例如，序列

S S S X X S S X X S X X

对应于图表

1	2	3	6	7	10
4	5	8	9	11	12

(37)

在这个图表中，当且仅当自左向右读X的个数决不超过S的个数时，关于列的规定成立。由定理H，形如（n，n）的图表的数目是

$$(2n)!/(n+1)!n!$$

而这是二叉树的数目（同等式2.3.4.4-13一致）。而且，如果对于 $n\geqslant m$ 使用形状为（n，m）的图表，那么这个论证解决了在习题2.2.1-4的答案中考虑的更为一般的“抽签”问题。所以，定理H包括了某些颇为复杂的枚举问题作为简单的特殊情况。

元素｛1，2，…，$2n$｝上的形状为（n，n）的任何图表 A 按麦克马洪提出的如下方式对应于相同形状的两个图表（P，Q），〔*Combinatory Analysis* 1（1915），130-131〕。设 P 由在 A 中出现的元素｛1，…，n｝组成；Q 由剩下的元素转动180°，并分别以 n，$n-1$，…，1代替 $n+1$，$n+2$，…，$2n$ 形成。例如，(37) 分成为

1	2	3	6
4	5		

和

		7	10
8	9	11	12

对后者进行转动和更名就得到

$P=$

1	2	3	6
4	5		

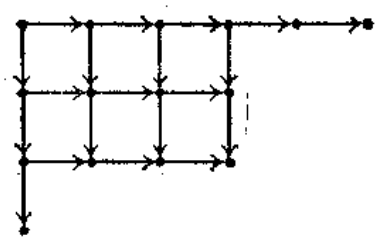
$Q=$

1	2	4	5
3	6		

(38)

反之，任何一对至多两行的各有 n 个小格的图表，都以这样一种方式对应于形状为（n，n）的图表。因此由习题7，不包含递减子序列 $a_i>a_j>a_k$，（$i<j<k$）的｛1，2，…，n｝的排列 $a_1a_2\cdots a_n$ 的个数为具有 n 个节点的二叉树的个数。D.罗登姆（D.Rotem）发现了在这样的排列和二叉树之间的一个有趣的一一对应，它较之这里已经使用的经由算法I的迂回（兜圈）方法更直接（*Inf. Proc. Letters* 4（1975），58-61〕。类似地，在二叉树和对于 $i<j<k$ 没有 $a_j>a_k>a_i$ 情形的排列之间，有着颇为直接的对应（见习题2.2.1-5）。

填写一个（6，4，4，1）形的图表的方式数，显然是把标号｛1，2，…，15｝以如下方式放置到有向图的顶点的方式数



(39)

这种放置的方式就是每当 $u \to v$ 时，顶点 u 的标号小于顶点 v 的标号。换言之，它是在2.2.3节的意义下，对偏序（39）进行拓扑排序的方式数。

一般说来，我们可以对不包含有向回路的任何有向图问同样的问题。如果有把定理H推广到任意有向图的某个简单的公式，那就好了；但并非所有的图都象对应于图表的图一样有这样令人兴奋的性质。本节末尾的习题中讨论了某些其它类的有向图，对于它们说来标号问题有一个简单的解。也还有一些习题，说明某些有向图没有对应于定理H的简单公式。例如，加标号的方式数不总是 $n!$ 的一个因子。

为了完成我们的研究，计算可以由 n 个不同的元素形成的图表总数；并以 t_n 来表示这个数。由定理B的推论，t_n 是 $\{1, 2, \cdots, n\}$ 的对合数。当且仅当一个排列的轮换形式仅由一元轮换（不动点）和二元轮换（对换）所组成时，这个排列是它自己的逆。由于 t_n 个对合中的 t_{n-1} 个有一元轮换（n），且它们中的 t_{n-2} 个有二元轮换（$j\ n$）（对于固定的 $j<n$），得到公式

$$t_n = t_{n-1} + (n-1)t_{n-2} \tag{40}$$

这是罗瑟于1800年设计出来的，以便对于小的 n 造出 t_n 表。

用另一种方式来计数时，假设有 k 个二元轮换和（$n-2k$）个一元轮换，有 $\binom{n}{2k}$ 种方法来选择不动点，且多项式系数 $(2k)!/(2!)^k$ 是把其它元素排成 k 个不同的对换的方式数；除以 $k!$ 使得对换是不加区别的，即得到

$$t_n = \sum_{k \geq 0} t_n(k), \quad t_n(k) = \frac{n!}{(n-2k)!\,2^k k!} \tag{41}$$

可惜，如所周知，这个和数已不能进一步简化，所以可用两个间接的方法以得到对于 t_n 的更好的理解：

a）可以找到生成函数

$$\sum_n t_n z^n/n! = e^{z+z^2/2} \tag{42}$$

见习题25。

b）可以确定 t_n 的渐近特性，这是一个有教益的问题，因为它包括某些在其它方面对我们有用的一般性技术，所以将以分析 t_n 的渐近特性来结束这一节。

分析（41）的渐近特性的头一步是判明它对于和数的主要贡献，由于

$$\frac{t_n(k+1)}{t_n(k)} = \frac{(n-2k)(n-2k-1)}{2(k+1)} \tag{43}$$

可以看到，从 $k=0$ 直到 k 近似于 $\frac{1}{2}(n-\sqrt{n})$，这些项逐渐地增加，然后又逐渐地减

少，当 k 近似于 $\frac{1}{2}n$ 时，它们减成为 0。主要的贡献显然来自 $k=\frac{1}{2}(n-\sqrt{n})$ 的附近。通常都把主要贡献安排在值为 0 处，所以写

$$k=\frac{1}{2}(n-\sqrt{n})+x \tag{44}$$

并把 $t_n(k)$ 的大小看作 x 的函数。

在 $t_n(k)$ 中可用斯特林近似公式，即等式 1.2.11.2-18，来摆脱阶乘。为此，把 x 限制在

$$-(n^{\epsilon+1/4})\leqslant x\leqslant n^{\epsilon+1/4} \tag{45}$$

的范围内是方便的（我们马上就会见到），比如说其中 $\epsilon=0.001$，使得一个误差项可以包括进去。稍微麻烦的计算——它实际上应该由计算机来完成——给出公式

$$\begin{aligned}t_n(k)=\exp\Big(&\frac{1}{2}n\ln n-\frac{1}{2}n+\sqrt{n}-\frac{1}{4}\ln n-2x^2/\sqrt{n}-\frac{1}{4}\\&-\frac{1}{2}\ln\pi-\frac{4}{3}x^3/n+2x/\sqrt{n}+\frac{1}{3}/\sqrt{n}-\frac{4}{3}x^4/n\sqrt{n}\\&+O(n^{5\epsilon-3/4})\Big)\end{aligned} \tag{46}$$

(45) 中对 x 所作的限制是合理的，因为可以置 $x=\pm n^{\epsilon+1/4}$ 来给出所有被舍弃的项的上界。即

$$\begin{aligned}e^{-2n^{2\epsilon}}\exp\Big(&\frac{1}{2}n\ln n-\frac{1}{2}n+\sqrt{n}-\frac{1}{4}\ln n-\frac{1}{4}\\&-\frac{1}{2}\ln\pi+O(n^{3\epsilon-1/4})\Big)\end{aligned} \tag{47}$$

而且如果乘以 n，则就得到所有被舍弃的项之和的上界。当 x 属于范围 (45) 时，这个上界的阶小于所要计算的各项的阶，因为因子 $\exp(-2n^{2\epsilon})$ 比起 n 的任何多项式来都要小得多。

显然，可以从这个和式中删去因式

$$\exp\left(\frac{1}{2}n\ln n-\frac{1}{2}n+\sqrt{n}-\frac{1}{4}\ln n-\frac{1}{4}-\frac{1}{2}\ln\pi+\frac{1}{3}/\sqrt{n}\right) \tag{48}$$

剩下的任务是在 $x=\alpha,\ \alpha+1,\ \cdots,\ \beta-2,\ \beta-1$ 的区间上对

$$\begin{aligned}&\exp\left(-2x^2/\sqrt{n}-\frac{4}{3}x^3/n+2x/\sqrt{n}-\frac{4}{3}x^4/n\sqrt{n}+O(n^{5\epsilon-3/4})\right)\\&=\exp\left(\frac{-2x^2}{\sqrt{n}}\right)\cdot\left(1-\frac{4}{3}\cdot\frac{x^3}{n}+\frac{8}{9}\cdot\frac{x^6}{n^2}\right)\cdot\left(1+2\frac{x}{\sqrt{n}}+2\frac{x^2}{n}\right)\\&\quad\cdot\left(1-\frac{4}{3}\frac{x^4}{n\sqrt{n}}\right)\cdot(1+O(n^{9\epsilon-3/4}))\end{aligned} \tag{49}$$

求和，其中 $-\alpha$ 和 β（不必是整数）近似地等于 $n^{\epsilon+1/4}$。通过对求和区间的变换，欧拉求和公式，即等式 1.2.11.2-10，可以写成

$$\begin{aligned}\sum_{\alpha\leqslant x<\beta}f(x)=&\int_\alpha^\beta f(x)dx-\frac{1}{2}f(x)\Big|_\alpha^\beta+\frac{1}{2}B_2\cdot\frac{f'(x)}{1!}\Big|_\alpha^\beta+\cdots+\\&\frac{1}{m+1}B_{m+1}\cdot\frac{f^{(m)}(x)}{m!}\Big|_\alpha^\beta+R_{m+1}\end{aligned} \tag{50}$$

这里 $|R_m|\leqslant(4/(2\pi)^m)\int_\alpha^\beta|f^{(m)}(x)|dx$。如果设 $f(x)=x^t\exp(-2x^2/\sqrt{n})$，其中 t 是一个固定的非负整数，欧拉求和公式将给出当 $n\to\infty$ 时 $\Sigma f(x)$ 的一个渐近级数，因为在这种情况下

$$f^{(m)}(x)=n^{(t-m)/4}g^{(m)}(n^{-1/4}x),\quad g(y)=y^te^{-2y^2} \tag{51}$$

且 $g(y)$ 是与 n 无关的一个很整齐的函数；导数 $g^{(m)}(y)$ 是 y 的多项式乘以 e^{-2y^2}，因此

$$R_m=O(n^{(t+1-m)/4})\int_{-\infty}^{+\infty}g^m(y)|dy=O(n^{(t+1-m)/4})$$

如果在(50)的右边以 $-\infty$ 和 $+\infty$ 来代替 α 和 β，则在每项中至多造成阶为 $O(\exp(-2n^\epsilon))$ 的一个误差。于是

$$\sum_{\alpha\leqslant x\leqslant\beta}f(x)=\int_{-\infty}^{\infty}f(x)dx+O(n^{-m}),\ \text{对于所有}\ m\geqslant 0 \tag{52}$$

只有对此待定的 $f(x)$，这个积分才是真正有意义的！这个积分不难求值（见习题26），所以可以乘出公式（49）并求和，这给出

$$\sqrt{\frac{\pi}{2}}\,n^{1/4}\left(1-\frac{1}{24}n^{-1/4}+O(n^{-1/2})\right)$$

$$t_n=\frac{1}{\sqrt{2}}n^{n/2}e^{-n/2+\sqrt{n}-1/4}\left(1+\frac{7}{24}n^{-1/2}+O(n^{-3/4})\right) \tag{53}$$

实际上，在这里 O 项应该在指数中有一个 9ϵ，但从处理方法中容易看出，如果取得更精确一些，这个 9ϵ 就将消失。原则上，这个方法可加以推广，以得到对于任意 k 的 $O(n^{-k})$，而不是 $O(n^{-3/4})$。t_n 的这一渐近级数，首先是由莫泽（Moser）和怀曼（Wyman）确定的（使用不同的方法），见"Canadian J. Math" 7 (1955)，159-168。关于进一步的例子和用于推导（53）的技术的推广，见 5.2.2 节的结语。

习题

1.〔16〕 什么图表（P, Q）对应于定理A的构造中的两行阵列

$$\begin{pmatrix}1&2&3&4&5&6&7&8&9\\6&4&9&5&7&1&2&8&3\end{pmatrix}$$

什么两行阵列对应于图表

$$P=\begin{array}{|c|c|c|}\hline 1&4&7\\\hline 2&8\\\cline{1-2} 5&9\\\cline{1-2}\end{array}\qquad Q=\begin{array}{|c|c|c|}\hline 1&3&7\\\hline 4&5\\\cline{1-2} 8&9\\\cline{1-2}\end{array}$$

2.〔M21〕 证明：当且仅当 t 是使得

$$p_{i_1}<p_{i_2}<\cdots<p_{i_t}=p,\quad q_{i_1}<q_{i_2}<\cdots<q_{i_t}=q$$

的下标 i_1, i_2, …, i_t 的最大个数时，（q, p）属于相对于（16）而言的类 t。

▶3.〔M24〕 证明在定理A的证明中所定义的对应关系也可通过构造诸如下的一张表来得到，

行 0	1	3	5	6	8
行 1	7	2	9	5	3
行 2	∞	7	∞	9	5
行 3		∞		∞	7
行 4					∞

这里行 0 和行 1 即是给定的两行阵列。对于 $k \geqslant 1$，行 $k+1$ 通过下列步骤由行 k 形成：

a）置 $p \leftarrow -\infty$。

b）设第 j 列是具有下列性质的诸列中最左边的列，即：该列的第 k 行包含一个 $< p$ 的整数，但第 $k+1$ 行是空白。如果不存在这样的列，且如果 $p=\infty$，则第 $k+1$ 行就完成了；如果不存在这样的列且 $p<\infty$，则返回（a）。

c）把 p 插入到第 j 列的第 $k+1$ 行中，然后置 p 等于第 k 行第 j 列处的条款，并且返回（b）。

一旦这张表已经以这样的方式构造出来，P 的第 k 行就由这张表的第 k 行中那些不在该表的第（$k+1$）行中的整数所组成；Q 的第 k 行由在这张表的第 0 行中那样一些整数所组成，它们所在的列的第 $k+1$ 行均含有 ∞。

4.〔$M26$〕（M. P. 舒曾伯杰） 设 π 是具有 k 个不动点的一个对合，证明在定理 B 的推论的证明中，对应于 π 的图表恰有 k 个奇数长度的列。

▶5.〔$M30$〕 设 $a_1 \cdots a_{j-1} a_j \cdots a_n$ 是不同元素的一个排列，且 $1 < j \leqslant n$。通过交换 a_{j-1} 和 a_j 得到的排列 $a_1 \cdots a_{j-2} a_j a_{j-1} a_{j+1} \cdots a_n$，称为“可允许的”，如果

i）$j \geqslant 3$ 且 a_{j-2} 位于 a_{j-1} 与 a_j 之间；或者

ii）$j < n$ 且 a_{j+1} 位于 a_{j-1} 与 a_j 之间。

例如，对于排列 1　5　4　6　8　3　7 恰可实施三个可允许的交换：可以交换 1 和 5，因为 $1<4<5$；可以交换 8 和 3，因为 $3<6<8$（或者因为 $3<7<8$）；但不能交换 5 和 4，或 3 和 7。

a）证明一个可允许的交换，不改变由这个排列通过逐次地插入元素 $a_1, a_2, \cdots, a_n$ 到开始时为空的图表中去而形成的图表 P。

b）反之，证明有相同 P 图表的任何两个排列，通过由一个或多个可允许的交换组成的序列，均可以彼此转换。〔提示：令 P 的形状是（n_1，n_2，…，n_m），证明对应于 P 的任何排列，通过一系列可允许的交换均可转换成为“规范排列” $P_{m1} \cdots P_{mn_m} \cdots P_{21} \cdots P_{2n_2} P_{11} \cdots P_{1n_1}$。〕

▶6.〔$M22$〕 设 P 是对应于排列 $a_1 a_2 \cdots a_n$ 的图表；利用习题 5 来证明 P^T 是对应于 $a_n \cdots a_2 a_1$ 的图表。

7.〔$M20$〕（C. 申斯迪德） 设 P 是对应于排列 $a_1 a_2 \cdots a_n$ 的图表。证明 P 中的列的数目是一个递增的子序列 $a_{i_1} < a_{i_2} < \cdots < a_{i_c}$ 的最大长度 c，其中 $i_1 < i_2 < \cdots < i_c$；P 中行的数目是一个递减子序列 $a_{j_1} > a_{j_2} > \cdots > a_{j_r}$ 的最大长度 r，其中 $j_1 < j_2 < \cdots < j_r$。

8.〔$M18$〕〔P. 厄尔多斯（P. Erdos），G. 斯泽克勒斯（G. Szekeres）〕 证明含有多于 n^2 个元素的任何排列都有一个长度大于 n 的单调子序列；但却有 n^2 个元素的排列，它无长度大于 n 的单调子序列。〔提示：见前一习题。〕

9.〔$M24$〕 继续习题 8，对于没有长度大于 n 的单调子序列的$\{1, 2, \cdots, n^2\}$的排列，求出一个“简单的”精确个数的公式。

10.〔$M20$〕 证明若 P 开始时是一图表，则当算法 S 终止时，P 仍是一图表。

11.〔20〕 仅仅给定算法 S 终止后的 r 和 s 之值，是否有可能按原来的条件恢复 P？

12.〔$M24$〕 如果反复地使用算法 S 删去其形状如 $(n_1, n_2, \cdots, n_m)$ 的一个图表 P 的所有元素，问步骤 S3 被执行多少次？对于 $n_1+n_2+\cdots+n_m=n$ 的所有形状，这个量的最小值是多少？

13.〔$M28$〕 证明定理 C。

14.〔$M43$〕 找出定理 D 的（c）部分的一个更直接的证明。

15.〔$M20$〕 多重集合$\{l\cdot a, m\cdot b, n\cdot c\}$有多少具有这样性质的排列，即当从左到右读这个排列时，c 的数目决不超过 b 的数目，而 b 的数目决不超过 a 的数目（例如，$a\ a\ b\ c\ a\ b\ b\ c\ a\ c\ a$ 是这样一个排列）？

16.〔$M08$〕 由（39）表示的偏序可以用多少种方式按拓扑排序？

17.〔$HM25$〕 设

$$g(x_1, x_2, \cdots, x_n; y)=x_1\Delta(x_1+y, x_2, \cdots, x_n)$$
$$+x_2\Delta(x_1, x_2+y, \cdots, x_n)+\cdots+x_n\Delta(x_1, x_2, \cdots, x_n+y)$$

证明

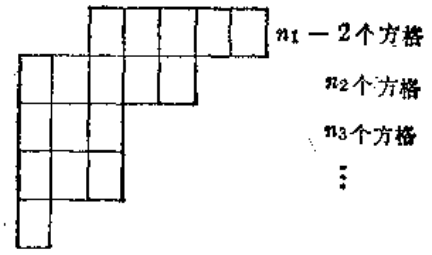
$$g(x_1, x_2, \cdots, x_n; y)=\left(x_1+x_2+\cdots+x_n+\binom{n}{2}y\right)\Delta(x_1, x_2, \cdots, x_n)$$

〔提示：多项式 g 是齐次的（所有的项都有相同的总次数）。而且，它是对诸 x 反对称的交换 x_i 和 x_j 时改变 g 的符号）。〕

18.〔$HM30$〕 推广习题 17，当 $m\geqslant 0$ 时计算和式

$$x_1^m\Delta(x_1+y, x_2, \cdots, x_n)+x_2^m\Delta(x_1, x_2+y, \cdots, x_n)$$
$$+\cdots+x_n^m\Delta(x_1, x_2, \cdots, x_n+y)$$

19.〔$M40$〕 求填写一个阵列的方式种数的公式，这个阵列和一个图表一样，但在行 1 的左边除去两个方格；例如



就是这样一个形状。（行和列，如同在通常的图表中那样，是按递增顺序排列的。）

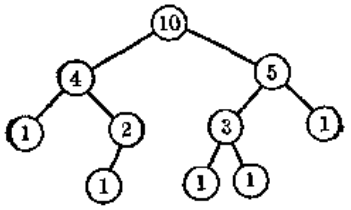
换言之，$\{1, 2, \cdots, n_1+\cdots+n_m\}$上形状为

$$(n_1, n_2, \cdots, n_m)$$

的 $f(n_1, n_2, \cdots, n_m)$ 图表中，头行兼有元素 1 和元素 2 者有多少？

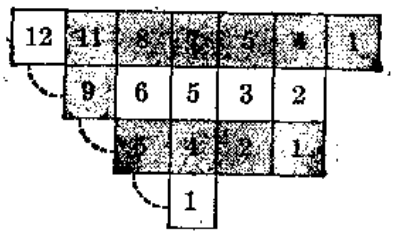
▶20.〔$M24$〕 对具有元素$\{1, 2, \cdots, n\}$的一个给定二叉树的节点进行标号，使得

每个节点的标号小于它的后裔的标号。证明这样加标号的方式种数是$n!$除以“子树的长度”（即在每个子树中节点的个数）的乘积（参考定理H）。例如，对



的节点进行标号的方式种数是 $10!/10\cdot4\cdot5\cdot1\cdot2\cdot3\cdot1\cdot1\cdot1\cdot1=9\cdot8\cdot7\cdot6$。

21.〔$HM31$〕（R. M. 思罗尔） 设$n_1>n_2>\cdots>n_m$确定一个“移位图表”的形状，其中行$i+1$从行i的右边一个位置开始。例如，形状（7，5，4，1）的移位图表有图式

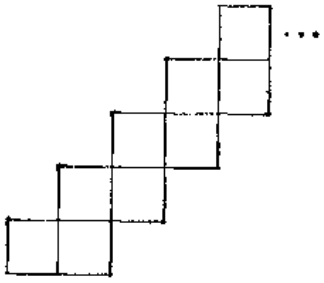


现在把整数$1,2,\cdots,n=n_1+n_2+\cdots+n_m$放置到形为（$n_1$，$n_2$，…，$n_m$）的移位图表中，使得行和列都处于递增的次序。证明：这种填移位图表的方式种数等于$n!$除以“广义钩子长度”的乘积；在上述图式中，已经用阴影部分标出对应于行1列2中小格的长度为11的广义钩子。（阵列左边的“倒楼梯”部分中的钩子，其形状为转了90°的U形，而不是L形。）于是以行和列的递增次序来填写上图的方式有

$$17!/12\cdot11\cdot8\cdot7\cdot5\cdot4\cdot1\cdot9\cdot6\cdot5\cdot3\cdot2\cdot5\cdot4\cdot2\cdot1\cdot1$$

种。

▶22.〔$HM30$〕（D. 安德烈） 把数$\{1,2,\cdots,n\}$放置到n个格子的阵列



中去，使得行和列处于递增的次序，问能有多少种方式A_n？求生成函数$g(z)=\sum A_nz^n/n!$。

23.〔$M39$〕 以集合$\{1,2,\cdots,N\}$的元素填入形为（n_1，n_2，…，n_m）的一个阵列，并且允许重复。问能有多少种方式，使得行是非减的，而列是严格递增的？例如简单的m行图形（1，1，…，1）可以$\binom{N}{m}$种方式来填入；1行的形状（m）可以$\binom{m+N-1}{m}$种方

式填入；图形（2，2）可以$\frac{1}{3}\cdot\binom{N+1}{2}\binom{N}{2}$种方式填入。

24.〔$M28$〕 证明

$$\sum_{\substack{q_1+\cdots+q_n=t\\0\leqslant q_1,\cdots,q_n\leqslant m}}\binom{m}{q_1}\cdots\binom{m}{q_n}\Delta(q_1,\ \cdots,\ q_n)^2$$

$$=n!\binom{nm-(n^2-n)}{t-\frac{1}{2}\cdot(n^2-n)}\binom{m}{n-1}\binom{m}{n-2}\cdots\binom{m}{0}\Delta(n-1,\ \cdots,\ 0)^2$$

〔提示：证明$\Delta(k_1+n-1,\ \cdots,\ k_n)=\Delta(m-k_n+n-1,\ \cdots,\ m-k_1)$；以类似于(38)的方式分解一个$n\times(m-n+1)$的图表；并如同推导（36）中那样处理这个和式。〕

25.〔$M20$〕 为什么（42）是对合的生成函数？

26.〔$HM21$〕 当t是一个非负整数时，计算$\int_{-\infty}^{\infty}x^t\exp(-2x^2/\sqrt{n})dx$。

27.〔$M24$〕 设Q是$\{1,\ 2,\ \cdots,\ n\}$上的杨氏图表，元素i在行r_i和列c_i中。当$r_i<r_j$时，我们说i在j"之上"。

a）证明：对于$1\leqslant i<n$，当且仅当$c_i\geqslant c_{i+1}$时i在$i+1$之上。

b）给定Q使得（P,Q）对应于排列

$$\begin{pmatrix}1 & 2\cdots n\\ a_1 & a_2\cdots a_n\end{pmatrix}$$

证明i在$i+1$之上，当且仅当$a_i>a_{i+1}$。（因此只要知道Q，就能确定排列中路段的个数，这一结果是属于M.P.舒曾伯杰的。）

c）证明，对于$1\leqslant i<n$，在Q中i在$i+1$之上，当且仅当在Q^S中$i+1$是在i之上。

28.〔$M43$〕 证明$\{1,\ 2,\ \cdots,\ n\}$的一个随机排列的最长递增子序列的平均长度近似于$2\sqrt{n}$（这是对应于定理A中行1的平均长度）。

29.〔$M50$〕 为了看看二维图表的多少性质可加以推广，试研究三维阵列。

30.〔$M42$〕（M.P.舒曾伯杰） 证明：由P到P^S所进行的操作可应用于任何有限的偏序集，而不仅仅是对于一个图表，后者仅是上述广泛应用的一个特殊情况：以整数$\{1,\ 2,\ \cdots,\ n\}$对一个偏序集的元素进行标号，使得这偏序同标号相协调。试通过逐次地删去标号1，2，…，同时以类似于算法S的方式移动其它标号，并且在释放的位置上放置（1），（2）…，求出一个类似于（26）的对偶的加标号的方法。证明：当以逆转的数值次序对对偶标号重复此操作时，得到原来的标号；并剖析这个操作的其它性质。

31.〔$HM30$〕 设x_n是在一个$n\times n$的棋盘上放置n个互不冲突的车的方式种数，并规定，这种布排在对角线的反射和180°的转动下都不改变。于是，$x_4=6$。试求x_n的渐近特性。

5.2 内部排序

让我们通过一项小实验，来开始有关好的“排序者队伍”的讨论。你将如何解决下面的程序设计问题？

“存储单元$R+1$，$R+2$，$R+3$，$R+4$和$R+5$包含五个数，试写出一个计算机程序，使得当必要时，它重新把这些数排成为递增次序的。”（如果你已经熟悉某些排序方法，就请你尽量暂时忘掉它们；想象你头一次着手解决这个问题，没有任何关于怎样做的预先的知识。）

在进一步往下阅读之前，要求你构造一个这个问题的解。

……

在解决上述问题时所花费的时间将在你继续阅读这一章时获得补偿。可能，你的解是下列类型之一：

A．一个插入排序　逐个考察诸项，每一个新的项被插入到相对于以前已排好序的诸项的适当位置上（这是许多玩桥牌者每抓一张牌时对他们手牌排序的方式）。

B．一个交换排序　如果发现两项是次序颠倒的，则就交换它们。重复这一过程直到不需要再作进一步的交换为止。

C．一个选择排序　首先找到最小的（或许最大的）项，并设法把它同余下的分开来；然后选择下一个最小的（或最大的），等等。

D．一个枚举排序　每个项都同其它的项进行比较；对比它小的键的个数进行计数，以确定该项的最后位置。

E．一个专用排序　如果像问题中所述的那样，是对五个元素进行排序，则它工作得很好，但不能容易地推广到较大的项数。

F．一种懒散的态度　在这种态度下你忽略了上面的建议并且决定全然不去解决这个问题。遗憾，现在你已经读得太远了，而且已经错过了时机。

G．一项新的超级排序技术　它肯定是对已知方法的改进（请立刻通知作者）。

如果这个问题比如说有1000项，而不只是五项，你可能已经发现了将在后边述及的更为精巧的技术。但无论如何，当着手解决一个新问题时，先找某些相当明显的解决方法，然后再试图来改进它，往往是明智的。上边的情况A、B和C导致了重要的几类排序技术，它们是已述的简单想法的深化。

已经发明了许多不同的排序算法。我们将在本书讨论其中大约25个算法。这样颇使人惊讶的众多的方法，实际上还只是至今已经想出的算法的一小部分；所作的讨论将略去许多现在已被废弃的方法，或者仅仅简单地提及它们。为什么会有这么多的排序方法呢？在计算机的程序设计中常有“为什么会有这样多的x方法呢？”的问题，其中x属于某个问题集合。现在的问题只不过是这个一般问题的特殊情况；而答案是：每种方法都有各自的优点和缺点，所以它对于某些数据和硬件配置，在性能上会超过其它的方法。可惜，还不知道“最好的”排序方法；如果是对于特定的机器，特定的目的，对特定的对象进行排序，则有许多最好的方法。用拉迪亚德·基普林（Rudyard Kipling）的话说：“有六十九种构造部落安置的方式，而且它们的每一种都是对的。”

一个好的想法是学习每种排序方法的特征，俾能对具体的应用作出一种明智的选择。幸而，学习这些算法并不是一项艰难的任务，因为它们都以有趣的方式相互关联着。

在本章开始时，已定义了将在排序研究中使用的基本术语和记号：记录

$$R_1,\ R_2,\ \cdots,\ R_N \tag{1}$$

有待按其键 K_1，K_2，…，K_N 的非减次序进行排序，实质上要求找出一个排列 $p(1)\ p(2)\cdots p(N)$ 使得

$$K_{p(1)} \leqslant K_{p(2)} \leqslant \cdots \leqslant K_{p(N)} \tag{2}$$

在本节中，我们讨论内部排序。此时，有待排序的记录个数足够小，以致整个过程都能在一台计算机的高速存储器中实现。

在某些情况下，将要求把这些记录在存储器中实际上重新排列，使得它们的键排成次序。但在另外的情况下，也可能只需要有指明这个排列的某种辅助表就够了。如果每个记录和／或键要占相当多的计算机存储器的字，则构造一个新的指向记录的链接地址表，并处理这些链接地址，而不是到处移动庞大的记录，通常更好些。这种方法称为地址表排序（见图 6）。如果键很短，但是记录的附属信息很长，则为了获得更高的速度，这个键即可用作链接地址，这就是所谓键排序。其它的排序方案利用了包括在每个记录中的一个辅助链接场；链接的方式是使这些记录最终被链接在一起以形成一个直接的线性表，每个链接指向下一个记录，这就是所谓表排序（见图 7）。

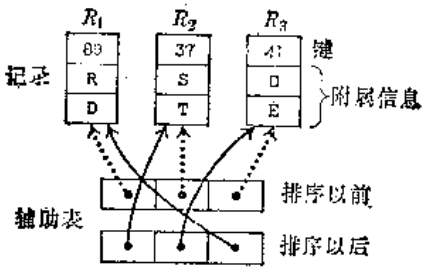


图 6　地址表排序

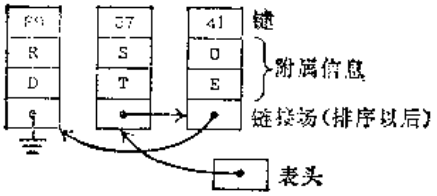


图 7　表排序

在用地址表方法或表方法进行排序之后，诸记录可像所希望的那样重新安排成递增的顺序。有若干方法来做到这一点，只要求有足够容纳一个记录的附加存储空间即可（见习题 10 到 12）；或者，可以简单地把这些记录都移到一个能容纳所有记录的新区域。后一方法通常比头一个方法快两倍，但它几乎要求两倍的存储空间。在许多应用中，全然不需要移动记录，因为对于随后的寻址操作而言，使用链接场通常已足够了。

我们将通过四个方面来说明将要“深入”考察的所有排序方法，即借助于

a）算法的一个英语语言描述；

b）一个框图；

c）一个 MIX 程序；

d）应用于一个给定数集的排序方法的例子。

〔只要适当，后边的例子将讨论 16 个数的某个集合，它是作者于 1963 年 3 月 19 日利用一副十进的骰子随机地选择的，参考习题 3.1-1（c）〕。

为了方便起见，在给出 MIX 程序时，通常都假定键是数值的并且能放到一个单字中去；有时，甚至把键限制为一个字的一部分。次序关系<将是通常的算术次序；记录将只由键组成，而没有附属的信息。这些假定使得程序更短和更容易理解。至于一般的情况(例如，使用地址表排序)，怎样来修改这些程序，应当是显然的。将对 MIX 程序给出每个排序算法运行时间的分析。

通过计数进行排序 作为研究内部排序方法的一个简单例子，考虑在本节开头提出的"计数"思想。这个简单的方法是以这样一个思想为基础的，即在最后排好序的序列中第 j 个键确实大于（$j-1$）个其它键。换言之，如果知道某个键确实超过 27 个其它键，则在排序之后对应的记录应当进入位置 28。所以，这个思想是比较每对键，计算有多少个键小于每一个特定的键。

进行这些比较的明显方法是

$$\text{对于 } 1 \leqslant i \leqslant N \text{ (对于 } 1 \leqslant j \leqslant N \text{ (比较 } K_j \text{ 和 } K_i))$$

但容易看出，这些比较中有一半以上是多余的，因为没有必要把一个键同它自己进行比较，也没有必要比较 K_a 和 K_b 然后比较 K_b 和 K_a。我们只需要比较

$$\text{对于 } 1 < i \leqslant N \text{ (对于 } 1 \leqslant j \leqslant i \text{ (比较 } K_j \text{ 和 } K_i))$$

因此导出了下列的算法。

算法 C（比较计数） 本算法通过维持一张辅助表 COUNT〔1〕，…，COUNT〔N〕，对于小于一个给定键的键个数进行计数，来实现用键 K_1，…，K_N 对记录 R_1，…，R_N 进行排序。算法结束时，COUNT〔j〕+1 确定记录 R_j 的最后位置。

C1.〔清诸 COUNT〕 把 COUNT〔1〕至 COUNT〔N〕都置成 0。

C2.〔对 i 进行循环〕 对 $i=N$，$N-1$，…，2 实施步骤 C3；然后结束此算法。

C3.〔对 j 进行循环〕 对 $j=i-1$，$i-2$，…，1 实施步骤 C4。

C4.〔比较 K_i，K_j〕 如果 $K_i<K_j$，则 COUNT〔j〕加 1，否则 COUNT〔i〕增 1。

注意，此算法不涉及记录的移动。它类似于地址表排序，因为 COUNT 表确定了这些记录最后的安排；但是由于 COUNT〔j〕告诉我们往何处移动 R_j，而不是指出哪一个

表 1 通过计数进行排序（算法 C）

键：	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
COUNT（初始）：	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
COUNT($i=N$):	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	12
COUNT($i=N-1$):	0	0	0	0	2	0	2	0	0	0	0	0	0	0	13	12
COUNT($i=N-2$):	0	0	0	0	3	0	3	0	0	0	0	0	0	11	13	12
COUNT($i=N-3$):	0	0	0	0	4	0	4	0	1	0	0	0	9	11	13	12
COUNT($i=N-4$):	0	0	1	0	5	0	5	0	2	0	0	7	9	11	13	12
COUNT($i=N-5$):	1	0	2	0	6	1	6	1	3	1	2	7	9	11	13	12
. . . . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .	. . .
COUNT($i=2$):	6	1	8	0	15	3	14	4	10	5	2	7	9	11	13	12

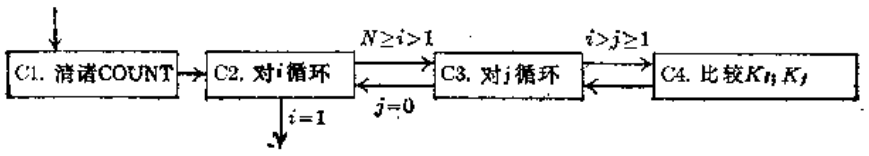


图 8 算法 C：比较计数

记录应当被移动到 R_j 的位置，故它与地址表排序略有不同（因此在 COUNT 表中确定了排列 $p(1)$，…，$p(N)$ 的"逆"，见 5.1.1 节）。

在描述这个算法以前所进行的讨论中，没有考虑相等的键的可能性。这可能是一个严重的疏忽，因为如果相等的键对应于相等的 COUNT，则这些记录的最后的重新安排将是颇为麻烦的。幸而，如同习题 2 所示，不管出现多少相等的键，算法 C 仍给出正确的结果。

程序 C（比较计数） 以下是算法C的 MIX 实现。假定对于 $1 \leqslant j \leqslant N$，$R_j$存于单元 INPUT+ j 中，而 COUNT[j] 存于单元 COUNT+ j 中；r11= i；r12— j；rA= K_i—R_j；rX=COUNT[i]。

1	START	ENT1	N	1	C1. 清零COUNT
2		STZ	COUNT,1	N	COUNT[i]←0
3		DEC1	1	N	
4		J1P	*-2	N	$N \geqslant i > 0$
5		ENT1	N	1	C2. 对 i 进行循环
6		JMP	1F	1	
7	2H	LDA	INPUT,1	$N-1$	
8		LDX	COUNT,1	$N-1$	
9	3H	CMPA	INPUT,2	A	C4. 比较K_i、K_j
10		JGE	4F	A	如果$K_i \geqslant K_j$则转移
11		LD3	COUNT,2	B	COUNT[j]
12		INC3	1	B	+1
13		ST3	COUNT,2	B	→COUNT[j]
14		JMP	5F	B	
15	4H	INCX	1	$A-B$	COUNT[i]+1→COUNT[i]
16	5H	DEC2	1	A	C3. 对 j 循环
17		J2P	3B	A	
18		STX	COUNT,1	$N-1$	
19		DEC1	1	$N-1$	
20	1H	ENT2	-1,1	N	$N \geqslant i > j > 0$
21		J2P	2B	N	

这个程序的运行时间是 $13N+6A+5B-4$ 个单位，其中N是记录个数；A是从N个对象的集合中选择两个的数目，即$\binom{N}{2}=(N^2-N)/2$；B 是满足 $j<i$ 且 $K_j>K_i$的下标对偶的数目。因此，B是排列 K_1，…，K_N的反序数；这是在 5.1.1 节深入分析过的量，在那里发现（等式 5.1.1-12,13）对于随机次序下的不相等的键

$$B=(\min 0, \operatorname{ave}(N^2-N)/4,\ \max(N^2-N)/2,\ \operatorname{dev}\sqrt{N(N-1)(N+2.5)/6})。$$

因此程序C要求的时间单位在 $3N^2+10N-4$ 和 $5.5N^2+7.5N-4$ 之间，而平均运行时间在这两个极值的正中间。例如，表 1 中的数据有$N=16$，$A=120$，$B=41$，所以程序C将用 1129 个单位时间进行排序。关于程序C的一个改型，请看习题 5，它有稍微不同的时间特征。

在这个运行时间中占支配地位的因子 N^2 说明，当 N 很大时，算法C 不是一个有效的方法。记录数加一倍，就会使运行时间增加四倍。由于这个方法要求比较所有不同的键对偶 (K_i, K_j)，因此没有显见的方法来使它摆脱对于 N^2 的依赖性。在这一章稍后将看到，利用"分划交换"技术，平均运行时间可以减少到 $N\log_2 N$。我们对于算法C 的主要兴趣在于它的简便性，而不在于它的有效性；它作为一个例子，表明将要描述的更为复杂（和更为有效）的方法的风格。

通过计数进行排序，还有另外一个方法，从有效性的观点看，它是十分重要的；它主要应用于有许多相同的键出现，且所有的键都落入范围 $u \leqslant K_j \leqslant v$ 的情况. 其中 $(v-u)$ 很小。这些假定看来是十分强的限制，但是事实上将看到这一思想有不少的应用；例如，如果把这个算法应用于键的头几位数字，而不是整个键，则这个文件将被部分地排序，而且完成这项任务将是相当简单的。

为了了解其中蕴含的原理，假设所有的键位于 1 和 100 之间。当第一遍扫描这个文件时，可以统计有多少个 1，2，…，100 出现，而在第二遍扫描时，就可以把这些记录移到输出区域中的适当位置。下列算法给出此过程的细节：

算法 D（分布计数） 假定所有键都是在 $u \leqslant K_j \leqslant v$ 范围中的整数，其中 $1 \leqslant j \leqslant N$。本算法通过一张辅助表 COUNT〔$u$〕，…，COUNT（$v$）对记录 R_1，…，R_N 进行排序。算法结束时，这些记录以所希望的次序移到一个输出区域 S_1，…，S_N 中去。

D1.〔清诸 COUNT〕 把 COUNT〔u〕至 COUNT〔v〕全部清成 0。

D2.〔对 j 进行循环〕 对于 $1 \leqslant j \leqslant N$ 实施步骤 D3；然后转到 D4。

D3.〔COUNT〔K_j〕增值〕 COUNT〔K_j〕的值增 1。

D4.〔累加〕（这时 COUNT〔i〕是等于 i 的键的个数）。对于 $i = u+1$，$u+2$，…，v，置 COUNT〔i〕←COUNT〔i〕+COUNT〔$i-1$〕。

D5.〔对 j 进行循环〕（这时 COUNT〔i〕是小于或等于 i 的键的个数，特别是 COUNT〔v〕$=N$。）对 $j=N$，$N-1$，…，1 实施步骤 D6；然后终止这个算法。

D6.〔输出 R_j〕 置 i ←COUNT〔K_j〕，$S_i \leftarrow R_j$ 及 COUNT〔K_j〕← $i-1$。

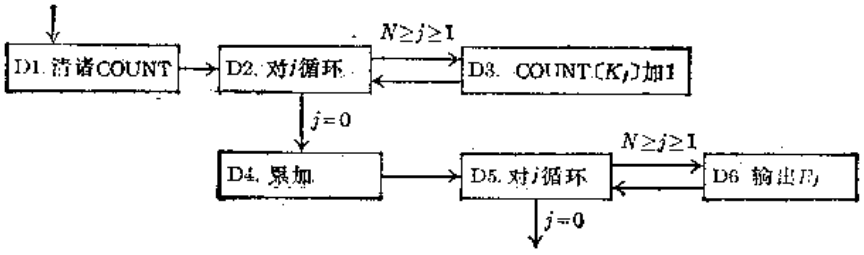


图 9 算法D：分布计数

这个算法的一个例子在习题 6 中讨论；在习题 9 中有一个 MIX程序。在上述条件下，这个排序过程是非常快的。

如同算法C这样通过比较计数进行的排序，首先是由E. H. 弗兰德（E. H. Friend）发表的〔*JACM*3（1956），152〕，尽管他没有宣称这是他自己的发明。象在算法D中那样的分布排序，是由H. 西沃德（H. Seward）于1954年首先提出的，他把这个方法同后边将

进行讨论的基数排序技术一起使用（见5.2.5节），该方法也由W.福伊尔齐格（W.Feurzeig）以“Mathsort”（数学排序）为名发表了，见*CACM*3（1960），601。

习题

1.〔15〕 如果在步骤C2中，i从2变到N，而不是从N变到2，算法C是否仍然有效？如果在步骤C3中j从1变到$i-1$又将如何呢？

2.〔21〕 证明当出现相等的键时，算法C仍将正确地工作。如果$K_j=K_i$且$j<i$，则R_j在最后的编序下是在R_i之前还是之后出现？

▶3.〔21〕 如果在步骤C4中的测试从“$K_i<K_j$”变成为“$K_i\leqslant K_j$”，则算法C是否仍将正确工作？

4.〔16〕 写出一个MIX程序，它“完成”由程序C着手的排序；你的程序应把这些键以递增的次序传送到单元OUTPUT＋1至OUTPUT＋N。你的程序需要多少时间？

5.〔22〕 下列一组修改是否改进程序C？

新添一行8a：INCX 0,2

改变行10：JGE 5F

改变行14：DECX 1

删去行15

6.〔18〕 用手算来模仿算法D，说明当对16个记录5T，0C，5U，0O，9.，1N，8S，2R，6A，4A，1G，5L，6T，6I，7O，7N进行排序时的中间结果，这里数字是键，而字母信息附属于这些记录。

7.〔13〕 算法D是一个“稳定的”排序方法吗？

8.〔15〕 如果在步骤D5中j从1变到N，而不是从N变到1，则算法D是否仍将正确地工作？

9.〔23〕 为算法D写出一个类似程序C和习题4的MIX程序来。你的程序的运行时间（作为N和（$v-u$）的函数）等于多少？

10.〔25〕 试设计一个有效的算法，当给定R_1，…，R_N的值以及$\{1，…，N\}$的排列$p(1)$，…，$p(N)$后，它分别以（$R_{p(1)}$，…，$R_{p(N)}$）代替N个量（$R_1,\cdots,R_N$）。试避免使用过多的存储空间。（如果希望在地址表排序之后，在存储器中重新安排这些记录，而不要求存储$2N$个记录的空间，就会出现这个问题。）

11.〔*M*27〕 为习题10的算法写出一个MIX程序，并分析它的效率。

▶12.〔25〕试设计一个有效的算法，它适合于在完成表排序（图7）之后，把记录R_1，…，R_N按排好的次序重新排列。试避免使用过多的存储空间。

▶13.〔27〕 算法D要求用于存放$2N$个记录R_1，…，R_N和S_1，…，S_N的空间，证明：如果用新的紧凑的过程来代替步骤D5和D6，则有可能仅需要存放N个记录$R_1,\cdots,R_N$的空间。（于是这个问题就成为在步骤D4之后，根据值

$$\mathrm{COUNT}[u],\ \cdots,\ \mathrm{COUNT}[v]$$

设计一个适当地重新排列R_1，…，R_N的算法，而无须使用附加的存储空间；这实质上是习题10中所考虑的问题的推广。）

5.2.1 通过插入进行排序

一类重要的排序技术是以5.2节开头处提到的“玩桥牌者”的方法为基础的。在考察记录 R_j 之前，假定以前的记录 R_1，…，R_{j-1} 已经排好序；然后把 R_j 插入到前已排好序的诸记录的适当位置，这个基本主题可以有若干有趣的变形。

直接插入 最简单的插入排序也是最显然的。假定 $1<j\leqslant N$，而且已经把记录 R_1，…，R_{j-1} 重新排列好，使得

$$K_1\leqslant K_2\leqslant\cdots\leqslant K_{j-1}$$

（记住：贯穿于本章，K_j 皆表示 R_j 的键部分。）把新键 K_j 依次地和 K_{j-1}，K_{j-2}，…进行比较，直到发现 R_j 应当插入到记录 R_i 和 R_{i+1} 之间；然后把记录 R_{i+1}，…，R_{j-1} 向上移动一格，并把新的记录放置到位置 $i+1$ 处。如下列算法所示的那样，宜于把比较和移动操作组合在一起，互相穿插，由于 R_j “被安放到适当的层次中去”，这种排序方法通常称为筛选或陷入技术。

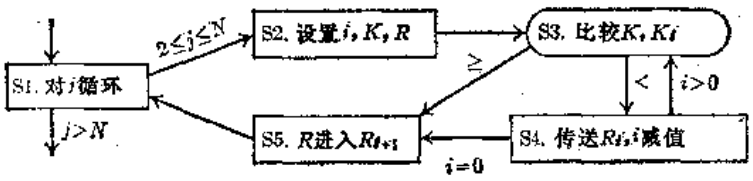


图10 算法S：直接插入

算法S（直接插入排序） 适当地重新安排记录 R_1，…，R_N；在完成排序之后，它们的键将是有序的，即有 $K_1\leqslant\cdots\leqslant K_N$。

S1.〔对 j 进行循环〕 对于 $j=2$，3，…，N 实施步骤S2到S5；然后终止本算法。

S2.〔给 i，K，R 赋值〕 置 $i\leftarrow j-1$，$K\leftarrow K_j$，$R\leftarrow R_j$（在下列步骤中，将通过按 i 的递减次序比较 K 和 K_i，来把 R 插入正确的位置。）

S3.〔比较 K，K_i〕 如果 $K\geqslant K_i$，则转向步骤S5（我们已找到了记录 R 的所求的位置）。

S4.〔移动 R_i，i 减值〕 置 $R_{i+1}\leftarrow R_i$，然后 $i\leftarrow i-1$。如果 $i>0$，则转回到步骤S3。（如果 $i=0$，K 是至今找到的最小的键，所以记录 R 属于位置1。）

S5.〔R 进入 R_{i+1}〕 置 $R_{i+1}\leftarrow R$。

表1表明，十六个作例子的数，是如何通过算法S进行排序的。这个方法在计算机上是很容易实现的；事实上，下列的MIX程序是书中最短的还不错的排序程序。

程序S（直接插入排序） 有待排序的记录在单元INPUT+1至INPUT+N中，它们按一个全字长的键在同一区域中就地排序。

rI1≡ $j-N$；rI2≡ i；rA≡ R ≡ K；假定 $N\geqslant 2$。

01	START	ENT1	2-N	1	*S1. 对 j 进行循环，j←2*
02	2H	LDA	INPUT+N,1	$N-1$	*S2. 给 i，K，R 赋值*
03		ENT2	N-1,1	$N-1$	$i\leftarrow j\leftarrow 1$

04	3H	CMPA	INPUT,2	$B+N-1-A$	S3. 比较K，K_i
05		JGE	5F	$B+N-1-A$	如果$K \geqslant K_i$，则转到S5
06	4H	LDX	INPUT,2	B	S4. 移动R_i，i减值
07		STX	INPUT+1,2	B	$R_{i-1} \leftarrow R_i$
08		DEC2	1	B	$i \leftarrow i-1$
09		J2P	3B	B	如果$i>0$，则转到S3
10	5H	STA	INPUT+1,2	$N-1$	S5. R进入R_{i+1}
11		INC1	1	$N-1$	
12		J1NP	2B	$N-1$	$2 \leqslant j \leqslant N$

这个程序的运行时间是$9B+10N-3A-9$个单位，其中N是被排序的记录数，A是在步骤S4中i减小到0的次数，而B是移动的次数。显然，A是对于$1<j \leqslant N$，$K_j<(K_1, \cdots, K_{j-1})$的次数；这是自左到右的极小值的个数，即在1.2.10节中被细致地分析的量。略加考虑又使我们看出B也是一个熟悉的量：对于固定的j，移动的次数即是K_j的反序的个数。所以，B是排列$K_1, K_2, \cdots, K_N$的反序的个数。因此，由等式1.2.10-16和5.1.1-12，13

$$A=(\min 0, \operatorname{ave} H_{N-1}, \max N-1, \operatorname{dev}\sqrt{H_n-H_n^{(2)}});$$

$$B=(\min 0, \operatorname{ave}(N^2-N)/4, \max(N^2-N)/2, \operatorname{dev}\sqrt{N(N-1)(N+2.5)/6});$$

而且，假定输入键是不同的并且是随机排列的，则程序S的平均运行时间是$(2.25N^2+7.75N-3H_N-6)\mu$，习题33说明了如何对此稍作改进。

表1的例子包含有16项，有两个自左至右的极小值，即087和061；而且同上一节我们所见到的那样，有41个反序。因此$N=16$，$A=2$，$B=41$，而总共的排序时间是514个时间单位。

表1　直接插入的例子

```
503:087
087 503:512
087 503 512:061
061 087 503 512:908
061 087 503 512 908:170
061 087 170 503 512 908:897
. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
061 087 154 170 275 426 503 509 512 612 653 677 765 897 908:703
061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908
```

二叉插入和两路插入　在一个直接插入排序期间，在处理第j个记录时，平均说来要把它的键大约同$\frac{1}{2}j$个预先已排好序的键进行比较，因此所实施的比较的总数大约是$\frac{1}{2}(1+2+\cdots+N) \approx \frac{1}{4}N^2$；当$N$适当大时，这就已经非常之大了。在6.2.1节，将研究"二分查找"技术，该技术使我们能够在仅仅进行$\log_2 N$次仔细选择的比较之后，就指出在哪里插入第j项。例如，当插入第64个记录时，可以由对K_{64}和K_{32}进行比较

开始。如果是小于，则就把它同K_{16}进行比较，但如果是大于，则就把它同K_{48}进行比较，等等。于是仅仅做六次比较之后，就可知道R_{64}应插入的位置。插入所有N项所作的比较总数就大约是$N\log_2 N$，这是对于$\frac{1}{4}N^2$的实质性的改进。而6.2.1节表明，相应的程序并不比直接插入程序复杂许多，这个方法叫二叉插入；它是早在1946年由约翰·莫恩利在计算机排序的第一个公开讨论中述及的。

可惜，二叉插入也有困难，它只解决了问题的一半。在已经发现记录R_j应插入到哪里之后，仍然需要移动大约$\frac{1}{2}j$个预先已排好序的记录，以便为R_j腾出位置，所以总共的运行时间实质上仍然同N^2成比例。某些计算机（例如IBM705）有一个内部的"滚进"指令，它以高速度来进行这样的移动操作，但当N增加时，对N^2的依赖性终将突出出来。例如，H.纳格勒（H. Nagler）的分析（*CACM* 3（1960），618-620）指出，当每个记录的长度为80个字符时，在705上对多于约$N=128$个记录的排序不应推荐二叉插入；类似的分析对其它的机器也适用。

当然，一个灵巧的程序员可以想出各种方法来减少所需移动的数量；头一个这样的技巧，如表2所示，是早在50年代时就被提出的。表中排序的头一项被放置在一个输出区域的中心，而且通过向右或向左（就看哪种最方便）移动腾出空间。此法比普通二叉插入节省一半运行时间，其代价是程序稍微复杂一点。使用此法时，还可以不必使用比N个记录所需要的更多的空间（见练习6）；但对于这个"两路"插入的方法，将不作更详细的叙述，因为已经提出了更有趣得多的方法。

表2　两路插入

∧503
087 503∧
∧087 503 512
061 087 503 512∧
061 087∧ 503 512 908
061 087 170 503 512 ∧908
061 087 170∧ 503 512 897 908
061 087 170 275 503 512 897 908

谢尔方法　如果有这样的一个排序算法，它一次只把诸项目移动一个位置，则它的平均运行时间至少同N^2成比例。因为在这个排序过程中每个记录都必须平均遍历大约1/3 N个位置（见习题7）。因此，如果要对直接插入作实质性的改进，就需要一种新原理，它使这些记录作长距离的跳跃，而不只是一些短促的小步挪动。

这样一个方法是由唐纳德·L.谢尔（Donald. L. Shell）于1959年提出的[*CACM* 2（July，1959），30-32]，我们称它为减少增量的排序。表3说明了这方法背后的一般想法；首先把这16个记录分成为8组，两两一组，即（R_1，R_9），（R_2，R_{10}），…，（R_8，R_{16}）。分别对每组记录进行排序，使我们进到表3的第2行，这称为"第一次扫描"。注意，154已同512交换了位置；908和897两者都跳到右边去了。现在把这些记录分成四组，每四个

表 3　减少增量的排序(8，4，2，1)

	503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703
8-排序	
	503 087 154 061 612 170 765 275 653 426 512 509 908 677 897 703
4-排序	
	503 087 154 061 612 170 512 275 653 426 765 509 908 677 897 703
2-排序	
	154 061 503 087 512 170 612 275 653 426 765 509 897 677 908 703
1-排序	
	061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908

一组，即 (R_1, R_5, R_9, R_{13})，…，$(R_4, R_8, R_{12}, R_{16})$，并再次分别对每组进行排序；这"第二次扫描"使我们进到了第 3 行。第三次扫描对于各有八个记录的两个组进行排序，然后第四次扫描通过对所有16个记录进行排序，来完成整个过程。每个中间排序过程，或者处理比较短的文件，或者处理相当好地编序了的文件，所以每个排序操作可以使用直接插入；这些记录势必快速地收敛到它们的最终目标。

增量 8，4，2，1 的序列不是一成不变的；任何序列 h_t，h_{t-1}，…，h_1 都可以使用，只要最后的增量 $h_1=1$ 就行。例如，表 4 示出了用增量 7，5，3，1 进行排序的相同数据。有些序列要比其它序列更好些，后面将讨论增量的选择。

算法 D（减少增量的排序）　就地排列记录 R_1，…，R_N；在完成排序之后，它们的键将是有序的：$K_1 \leqslant \cdots \leqslant K_N$。用一个辅助的增量序列 h_t，h_{t-1}，…，h_1 来控制这个排序过程，其中 $h_1=1$；适当选择增量可以显著地减少排序时间，当 $t=1$ 时，这个算法退化为算法 S。

D1.〔对 s 进行循环〕　对于 $s=t$，$t-1$，…，1 实施步骤 D2，然后终止这个算法。

D2.〔对 j 进行循环〕　置 $h \leftarrow h_s$，并对 $h<j \leqslant N$ 实施步骤 D3 到 D6。(我们将使用直接插入方法来对隔开 h 个位置的元素进行排序，使得对于 $1 \leqslant i \leqslant N-h$，$K_i \leqslant K_{i+h}$。步骤 D3 到 D6 实际上分别和算法 S 中的步骤 S2 到S5 相同。)

D3.〔给 i，K，R 赋值〕　置 $i \leftarrow j-h$，$K \leftarrow K_j$，$R \leftarrow R_j$。

D4.〔比较 K，K_i〕　如果 $K \geqslant K_i$，则转到步骤 D6。

D5.〔移动 R_i，i 减值〕　置 $R_{i+h} \leftarrow R_i$，然后 $i \leftarrow i-h$。如果 $i>0$，则转回步骤 D4。

D6.〔R 进入 R_{i+h}〕　置 $R_{i+h} \leftarrow R$。

对应的 MIX 程序不比直接插入程序长很多，下列代码中的08～19行是从程序 S 直接翻译过来的成为算法 D 这个更一般的结构的一部分。

程序 D（减少增量的排序）　我们假定这些增量被存储于一个辅助表中，h_s 在单元 $H+s$ 中；所有的增量都小于 N。寄存器分配：rI1≡$j-N$；rI2≡i；rA≡R≡K；rI3≡s；rI4≡h。注意，这个程序是自修改的，为的是要得到内部循环的有效执行。

01	START	ENT3	T	1	D1. 对 s 进行循环。$s \leftarrow t$
02	1H	LD4	1H, 3	T	D2. 对 j 进行循环。$h \leftarrow h_s$
03		ENT1	INPUT, 4	T	修改主循环中三条
04		ST1	6F (0:2)	T	指令的地址
05		ST1	7F (0:2)	T	
06		ENN1	−N, 4	T	$r1 \leftarrow N-h$
07		ST1	4F (0:2)	T	
08		ENT1	1−N, 4	T	$j \leftarrow h+1$
09	2H	LDA	INPUT+N, 1	$NT-S$	D3. 给 i, K, R 赋值
10	4H	ENT2	N−H, 1	$NT-S$	$i \leftarrow j-h$（被修改的指令）
11	5H	CMPA	INPUT, 2	$B+NT-S-A$	D4. 比较 K, K_i
12		JGE	7F	$B+NT-S-A$	如果 $K \geqslant K_i$，则转到 D6
13		LDX	INPUT, 2	B	D5. 移动 R_i, i 减值
14	6H	STX	INPUT+H, 2	B	$R_{i+h} \leftarrow R_i$（被修改的指令）
15		DEC2	0, 4	B	$i \leftarrow i-h$
16		J2P	5B	B	如果 $i>0$，则转到 D4
17	7H	STA	INPUT+H, 2	$NT-S$	D6. R 进入 R_{i+h}（被修改的指令）
18	8H	INC1	1	$NT-S$	$j \leftarrow j+1$
19		J1NP	2B	$NT-S$	如果 $j \leqslant N$，则转到 D3
20		DEC3	1	T	
21		J3P	1B	T	$t \geqslant s \geqslant 1$

***谢尔方法的分析** 为了选择好的用于算法D中的增量序列 h_t, …, h_1，需要把运行时间作为这些增量的函数来分析。这导致了某些更迷惑人的数学问题，至今犹未完全解决；还没有人能够确定：对于很大的 N 值，最好的增量序列是什么？但是关于谢尔减少增量排序的特性，已经知道了大量有趣的事实，我们将在这里作一概述。细节在以下的习题中给出〔不专长于数学的读者可以跳过下面数页，自公式（8）处继续阅读〕。

程序D的频率计数指出有五个因素确定执行时间：文件大小 N；扫描次数（即增量的个数）$T=t$；增量的和

$$S=h_1+h_2+\cdots+h_t;$$

比较的次数 $B+NT-S-A$；以及移动的次数 B。如同在程序S中分析的那样，A 是在中间的排序操作中遇到的自左到右的极小值的数目，B 是诸子文件中的反序数。对运行时间起支配作用的因素是 B，所以我们将把大部分注意力集中于此。为了进行分析，将假定键是不同的，且开始时是处于随机的顺序之下。

我们把步骤D2的操作称为"h 排序"，于是，谢尔方法由 h_t 排序，接着是 h_{t-1}, …，最后是 h_1 排序所组成。对于 $1 \leqslant i \leqslant N-h$，满足 $K_i \leqslant K_i+h$ 的文件，称为"h 有序"的文件。

当恰有两个增量 $h_2=2$ 和 $h_1=1$ 时，首先考虑直接插入的最简单的推广。在第二次扫描期间，有一个2有序的键序列 K_1, K_2, …, K_N。容易看出，对于 $1 \leqslant i \leqslant n-2$，使得 $a_i \leqslant a_{i+2}$ 的 $\{1, 2, \cdots, n\}$ 的排列 $a_1a_2\cdots a_n$ 的个数是

$$\binom{n}{\lfloor n/2 \rfloor}$$

因为若任选$\lfloor n/2\rfloor$个元素放入偶数号位置$a_2a_4\cdots$中，剩下的$\lceil n/2\rceil$个元素放入奇数号位置中，则由每一次这种选择我们正好得到一个2有序排列。在一个随机文件已经2排序之后，每种2有序排列都是同等可能的。在所有这样的排列当中，反序的平均个数是多少呢？

设A_n是在$\{1, 2, \cdots, n\}$的所有2有序排列当中的反序总数。显然$A_1=0$，$A_2=1$，$A_3=2$；通过考虑下面六种情况

1 3 2 4　1 2 3 4　1 2 4 3　2 1 3 4　2 1 4 3　3 1 4 2

我们发现$A_4=1+0+1+1+2+3=8$。为了一般地研究A_n，考虑对于$n=15$，图11所示的"格子框图"。$\{1, 2, \cdots, n\}$的一个2有序排列可以表示作从左上角的点(0, 0)到右下角的点$(\lceil n/2\rceil, \lfloor n/2\rfloor)$的一条通路，如果根据在这个排列中，$k$出现于奇数或偶数位置而分别作向下或向右的第$k$步通路的话。这个规则确定了2有序排列同格子框图的角到角的n步通路之间的一一对应。例如，图11中由黑线所示的通路对应于排列

2 1 3 4 5 6 7 10 8 11 9 12 14 13 15　　(1)

而由黑点标出的楼梯通路对应于完全排序的排列。对格子框图的仔细研究就会发现通路和反序之间的自然联系，即楼梯通路与一给定通路之间的区域等于相应排列的反序数。于是，排列（1）有六个反序，对应于图11中的六个非阴影的方格。

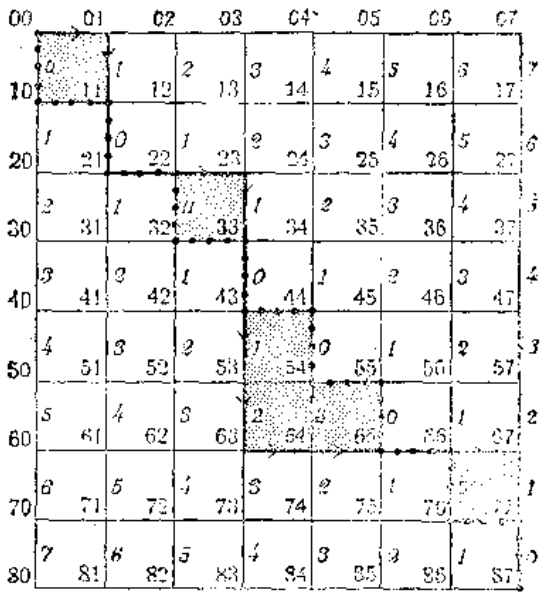


图11　在2有序和一个格子通路之间的对应关系。斜体数字是权，它对应于2有序排列中的反序数

图中的斜体数字提供了比较反序的一个等价的方式，即作为沿给定通路的"权"之和：由（i, j）到（$i+1$, j）的线有权$|i-j|$。当$a\leqslant a'$和$b\leqslant b'$时，从（a, b）到(a', b')的有关通路的条数是把$b'-b$条水平线同$a'-a$条垂直线混合起来的方式种数，即

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

因此，考察所有包含从（i, j）到（$i+1$, j）的垂直线段的通路，与这些通路相应的排列数是

$$\binom{i+j}{i}\binom{n-i-j-1}{\lfloor n/2\rfloor-j}$$

乘以相关联的权并对所有线段求和，给出

$$A_{2n}=\sum_{\substack{0\leqslant i\leqslant n\\0\leqslant j\leqslant n}}|i-j|\binom{i+j}{i}\binom{2n-i-j-1}{n-j}$$

$$A_{2n+1}=\sum_{\substack{0\leqslant i\leqslant n\\0\leqslant j\leqslant n}}|i-j|\binom{i+j}{i}\binom{2n-i-j}{n-j} \tag{2}$$

在这些和中的绝对值符号使得此计算有些棘手，但是习题14说明 A_n 有惊人的简单形式 $\lfloor n/2\rfloor 2^{n-2}$。因此，在一个随机的 2 有序排列中的平均反序数是

$$\lfloor n/2\rfloor 2^{n-2}\Big/\binom{n}{\lfloor n/2\rfloor}$$

由斯特林近似公式，这渐近于 $\sqrt{\pi/128}\,n^{3/2}\approx 0.15n^{3/2}$。容易看出反序的极大个数是

$$\binom{\lfloor n/2\rfloor+1}{2}\approx\frac{1}{8}n^2$$

如同在习题15中那样，考察生成函数

$$h_1(z)=1,\ h_2(z)=1+z,$$
$$h_3(z)=1+2z,\ h_4(z)=1+3z+z^2+z^3,\ \cdots, \tag{3}$$

以便更加仔细地研究反序的分布，是有教益的。这样一来，我们就发现了标准离差也同 $n^{3/2}$ 成比例。所以，这个分布相对于平均值来说不是非常稳定的。

现在来考虑，当增量是 h 和 1 时，算法 D 的一般的两次扫描的情况：

定理H 在 $\{1,\ 2,\ \cdots,\ n\}$ 的一个 h 有序的排列中，反序的平均数是

$$f(n,\ h)=\frac{2^{2q-1}q!q!}{(2q+1)!}\left(\binom{h}{2}q(q+1)+\binom{r}{2}(q+1)-\frac{1}{2}\binom{h-r}{2}q\right) \tag{4}$$

其中，$q=\lfloor n/h\rfloor$，$r=n \bmod h$。

这个定理是道格拉斯·H. 亨特给出的（Douglas H. Hunt [Bachelor's thesis, Princeton University (April, 1967)]。注意，当 $h\geqslant n$ 时，这一公式正确地给出 $f(n,\ h)=\frac{1}{2}\binom{n}{2}$。

证明 一个 h 有序的排列包含长度为 $q+1$ 的 r 个排好序的子序列和长度为 q 的 $h-r$ 个排好序的子序列。每个反序都来自一对不同的子序列，而且在一个随机的 h 有序的排列中，任一对不同子序列都定义一个随机的 2 有序的排列。因此，反序的平均个数等于在每对不同子序列之间反序的平均个数之和，即

$$\binom{r}{2}\frac{A_{2q+2}}{\binom{2q+2}{q+1}}+r(h-r)\frac{A_{2q+1}}{\binom{2q+1}{q}}+\binom{h-r}{2}\frac{A_{2q}}{\binom{2q}{q}}=f(n,\ h)$$

推论 如果增量序列 $h_t,\cdots,\ h_1$ 满足条件

$$h_{s+1} \bmod h_s=0 \qquad 对于\ t>s\geqslant 1 \tag{5}$$

则在算法 D 中移动操作的平均次数是

$$\sum_{t\geqslant s\geqslant 1}(r_s f(q_s+1,\ h_{s+1}/h_s)+(h_s-r_s)f(q_s,\ h_{s+1}/h_s)) \tag{6}$$

其中 $r_s=N \bmod h_s$，$q_s=\lfloor N/h_s\rfloor$，$h_{t+1}=Nh_t$，f 在（4）中定义。

证明　h_s 排序的过程由对长度为 q_{s+1} 的 r_s 个 (h_{s+1}/h_s) 有序的子文件，以及长度为 q_s 的 (h_s-r_s) 个这样的子文件进行直接插入排序所组成。整除性条件意味着，在每种 (h_{s+1}/h_s) 有序排列都是同等可能的意义下，每个子文件都是"随机的"(h_{s+1}/h_s) 有序的排列，因为我们假定原来的输入是不同元素的一个随机排列。

当增量分别为 h 和 1 时，在这个推论中的条件（5）对于两次扫描的谢尔排序总是满足的，如果 $q=\lfloor N/h\rfloor$ 且 $r=N \bmod h$，则程序D中的量 B 的平均值将为

$$rf(q+1,\ N)+(h-r)f(q,\ N)+f(N,\ h)$$
$$=\frac{r}{2}\binom{q+1}{2}+\frac{h-r}{2}\binom{q}{2}+f(N,\ h)$$

若取一次近似，函数 $f(n,\ h)$ 等于 $(\sqrt{\pi}/8)n^{3/2}h^{1/2}$（把它同图 12 的光滑曲线相对照）。因此两次扫描的程序D的运行时间近似同 $2N^2/h+\sqrt{\pi}N^3\sqrt{h}$ 成比例。而 h 的最好选择近似于 $\sqrt[3]{16N/\pi}\approx 1.72\sqrt[3]{N}$；而且通过这样选择的 h，我们得到同 $N^{5/3}$ 成比例的平均运行时间。

于是，可以仅仅通过使用具有两个增量的谢尔方法，就能对直接插入法作出实质性的改进，由 $O(N^2)$ 到 $O(N^{1.667})$。显然当使用更多的增量时，甚至可以做得更好。习题 18讨论了当 t 固定且当诸 h 受整除性条件限制时 $h_t,\cdots,h_1$ 的最佳选择，对于很大的 N，运行时间减少到 $O(N^{1+\epsilon+\epsilon/2})$，其中 $\epsilon=1/(2^t-1)$。我们不能通过使用上述公式突破 $N^{1.5}$ 的壁垒，因为最后的扫描总是要对这个和提供

$$f(N,\ h_2)\approx(\sqrt{\pi}/8)N^{3/2}h_2^{1/2}$$

个反序。

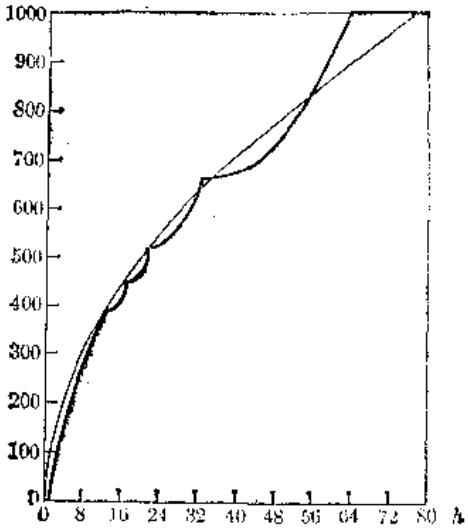


图12　n 个元素（$n=64$）的 h 有序文件中的反序平均数 $f(n,h)$

但是凭直观感觉就知道，当增量 $h_t,\cdots,h_1$ 不满足整除性条件（5）时，甚至能做得更好。例如，8 排序继之以 4 排序，再继之以 2 排序的方法不允许在奇位置和偶位置的键之间有任何相互作用；因此，最后的 1 排序扫描面临 $O(N^{3/2})$ 个反序。相反，7 排序继之 5排序，再继之以 3 排序，以这样的方式交叉掺杂，最后的 1 排序扫描就不会遇到多于 $2N$ 个反序〔见习题 26〕！真的，这里出现了一个令人吃惊的现象：

定理K　如果一个 k 有序的文件被 h 排序，则它保持为 k 有序的。

于是，首先是 7 排序，然后是 5 排序的一个文件，变成为既是 7 有序的，又是 5 有序的。而如果对它进行 3 排序，则结果是 7，5 和 3 有序的。这个值得注意的性质的例子在表 4 中。

证明　习题 20 表明，这个定理是从下列事实得出的：

引理L　设 m、n、r 是非负整数，且 $x_1,\cdots,x_{m+r}$ 和 $y_1,\cdots,y_{n+r}$ 是任意数列，满足

$$y_1\leqslant x_{m+1},\quad y_2\leqslant x_{m+2},\ \cdots,\quad y_r\leqslant x_{m+r} \tag{7}$$

如果诸 x 和诸 y 被独立地排序，使得 $x_1\leqslant\cdots\leqslant x_{m+r}$ 和 $y_1\leqslant\cdots\leqslant y_{n+r}$，则关系（7）仍将

表 4　减少增量排序(7，5，3，1)

	503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703
7-排序	
	275 087 426 061 509 170 677 503 653 512 154 908 612 897 765 703
5-排序	
	154 087 426 061 509 170 677 503 653 512 275 908 612 897 765 703
3-排序	
	061 087 170 154 275 426 512 503 653 612 509 765 677 897 908 703
1-排序	
	061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908

成立。

证明　已知除m个之外所有的x都高于（即大于或等于）某个y，这里，不同的x高于不同的y。设$1\leqslant j\leqslant r$，由于x_{m+j}在排序之后高于$m+j$个x，它至少高于j个y；所以它高于最小的j个y；因此在排序之后$x_{m+j}\geqslant y_j$。

定理K表明，通过互质增量来进行排序是合意的，但它并不直接导致对在算法D中所作移动次数的精确估计。由于既是h有序又是k有序的$\{1, 2, \cdots, n\}$的排列个数，并不总是$n!$的一个因子，我们可以看到，定理K并不提供一切详情；在进行k排序和h排序之后，某些k和h有序的文件，比起其它文件更为经常出现。而且，对于一般的增量h_t，…，h_1并没有显然的方法找出算法D的“最坏情况”。因此，在一般情况下，对于这个算法的分析，至今使人们困惑不已。本质上，所知的仅仅是在某些情况下的极大运行时间的近似渐近形式：

定理P　当$h_s=2^s-1$，$1\leqslant s\leqslant t=\lfloor\log_2 N\rfloor$时，算法D的运行时间是$O(N^{3/2})$。

证明　只需对于第s次扫描中的移动次数B_s给出下界，使得$B_t+\cdots+B_1=O(N^{3/2})$就可以了。在头$t/2$次扫描期间（对于$t\geqslant s\geqslant t/2$），可以使用显然的界$B_s=O(h_s(N/h_s)^2)$，而对于随后的扫描，可以使用习题23的结果，$B_s=O(Nh_{s+2}h_{s+1}/h_s)$；因此，$B_t+\cdots+B_1=O(N(2+2^2+\cdots+2^{t/2}+2^{t/2}+\cdots+2))=O(N^{3/2})$。

这个定理是A. A. 佩珀诺夫（A. A. Паперное）和Г. В. 斯塔西维奇(Г. В. Штаневич）给出的。见“Проблемы Передаци Информации” 1, 3 (1965, 81-98。它给出了对于这个算法的极长运行时间的上限，而不只是平均运行时间的一个界限。由于当诸h满足整除性约定（5）时，极长运行时间是N^2阶的，这个结果是不平凡的；而且习题24表明：指数3/2已不能再降低了。

1969年沃恩·普拉特（Vaughan Pratt）发现了定理P的一个有趣的改进：如果被选择的增量是形如2^p3^q的所有数的集合，它们都是小于N的，则算法D的运行时间是$N(\log_2 N)^2$阶的。在这种情况下，还可以对这个算法作若干重要的简化。可惜，普拉特的方法需要相当多的扫描次数，所以除非N十分之大，否则它并不是选择增量的最好的方法；见习题30和31。

让我们考虑程序D的总的运行时间$(9B+10NT+13T-10S-3A+1)u$。表5说明当$N=8$时，相对于各种增量序列的平均运行时间；表中的每项数据，都可以由上面那些

公式，或者由习题 19 中的公式计算出来，只有增量 5 3 1 或 3 2 1 例外；在这种情况下，已对所有 8! 种排列作了详尽彻底的研究。注意，对于这个小的N值，簿记操作占了运行时间的大部分，以致当 $t=1$ 时得到最好的结果；因此，对于$N=8$，使用简单的直接插入更好（当$N=8$时，程序 S 的平均运行时间仅仅是 191.85 u）。奇怪，当 $h_2=6$ 时，出现最好的两次扫描算法，因为一个大的S值在这里比一个小的B值更为重要。类似地，三个增量 3 2 1 使平均移动次数减到极小，但它们并不导致最好的三次扫描序列。在这里，记下使移动次数增到极大的某些“最坏”的排列可能是有趣的，因为这种排列的一般构造仍然是未知的：

$h_3=5$，$h_2=3$，$h_1=1$；8 5 2 6 3 7 4 1（19 次移动）

$h_3=3$，$h_2=2$，$h_1=1$；8 3 5 7 2 4 6 1（17 次移动）

表 5　当$N=8$时，对算法 D 的分析

增量	A平均	B平均	S	T	MIX时间
1	1.718	14.000	1	1	204.85 u
2 1	2.667	9.657	3	2	235.91 u
3 1	2.917	9.100	4	2	220.16 u
4 1	3.083	10.000	5	2	217.75 u
5 1	2.601	10.000	6	2	210.00 u
6 1	2.735	10.667	7	2	206.60 u
7 1	1.718	12.000	8	2	208.85 u
4 2 1	3.500	8.324	7	3	272.32 u
5 3 1	3.301	8.167	9	3	251.60 u
3 2 1	3.320	7.829	6	3	278.50 u

随着N的增大，情况也就大不一样。表 6 指明了当$N=1000$时各种增量序列的近似移动次数。最初的几项满足整除性的限制（5），因而可使用公式（6）；通过经验测试得到了其它增量序列的近似平均值。生成了 1000 个元素的五个随机文件，而且它们中的每一个都由每一个增量序列加以排序。

在这些数据中，某些型式是明显的，但是算法 D 的行为仍然是十分模糊的。谢尔最初建议使用增量$\lfloor N/2\rfloor$，$\lfloor N/4\rfloor$，$\lfloor N/8\rfloor$，…，但当N的二进表示包含一长串 0 时，这是不理想的。拉扎勒斯（Lazarus）和弗朗克（Frank）[*CACM* 3(1960), 20-22]建议使用实质上相同的序列，但必要时加 1，以使所有增量都成为奇数。希巴德（Hibbard）[*CACM* 6(1963), 206-213]建议使用形如2^k-1的增量；佩珀诺夫和斯塔西维奇建议使用形如2^k+1的增量。表 6 中研究的其它自然序列包含数$(2^k-(-1)^k)/3$、$(3^k-1)/2$以及斐波那契数。

对于形如2^k+1的增量，最少的移动次数大约是 6600；但重要的是要认识到，尽管移动次数渐近地主宰运行时间，但它并不是唯一要考虑的因素。由于程序 D 花费$9B+10NT+\cdots$个单位时间，我们看到，减少一次扫描大约相当于减少$\frac{10}{9}N$次移动；我们宁愿增加1100次移动，以减少一次扫描，因此比如说，由大于$\frac{1}{3}N$的h_t开始，看来并不明智，因为一个很大的增量不能使随后的移动次数减少到足以补偿头一次扫描的程度。

表 6　当$N=1000$时，对算法D的分析

增量										A平均	B平均	T	增量														A平均	B平均	T
									1	6.485	249750	1					513	257	129	65	33	17	9	5	3	1	575	6600	10
								17	1	64.66	41666.5	2						257	129	65	33	17	9	5	3	1	450	6800	9
							60	6	1	158.1	26360.6	3							129	65	33	17	9	5	3	1	300	7600	8
						140	20	4	1	262.1	21912.5	4								65	33	17	9	5	3	1	200	9400	7
					256	64	16	4	1	361.9	20458.9	5									33	17	9	5	3	1	125	13000	6
				576	192	48	16	4	1	419.2	20087.6	6					683	341	171	85	43	21	11	5	3	1	500	7050	10
			729	243	81	27	9	3	1	378.3	18533.2	7						341	171	85	43	21	11	5	3	1	500	7300	9
512	256	128	64	32	16	8	4	2	1	492.5	16434.7	10									255	63	15	7	3	1	400	8600	6
	500	250	125	62	31	15	7	3	1	525	7600	9									257	65	17	5	3	1	400	8700	6
	501	251	125	63	31	15	7	3	1	575	7200	9									341	85	21	5	3	1	425	9300	6
	511	255	127	63	31	15	7	3	1	550	7100	9	610	377	233	144	89	55	34	21	13	8	5	3	2	1	550	7400	14
		255	127	63	31	15	7	3	1	450	7300	8								377	144	55	21	8	3	1	475	8800	7
			127	63	31	15	7	3	1	300	8150	7								365	122	41	14	5	2	1	450	8300	7
				63	31	15	7	3	1	200	9700	6									364	121	40	13	4	1	450	9200	6
					31	15	7	3	1	125	13600	5										121	40	13	4	1	275	9900	5

1971 年斯坦福大学的詹姆斯·彼得森（James Peterson）和戴维·L. 拉塞尔（David L. Russell）对算法D进行了大量实验；他们发现当 $100 \leqslant N \leqslant 60000$ 时，平均移动次数 B 可以用下列公式相当好地近似：

$1.09N^{1.27}$ 或 $.30N(\ln N)^2-1.35N\ln N$　对于增量 $2^k+1, \cdots, 9, 5, 3, 1$；

$1.22N^{1.26}$ 或 $.29N(\ln N)^2-1.26N\ln N$　对于增量 $2^k-1, \cdots, 15, 7, 3, 1$；

$1.12N^{1.23}$ 或 $.36N(\ln N)^2-1.73N\ln N$　对于增量 $(2^k\pm 1)/3, \cdots, 11, 5, 3, 1$；

$1.66N^{1.25}$ 或 $.33N(\ln N)^2-1.26N\ln N$　对于增量 $(3^k-1)/2, \cdots, 40, 13, 4, 1$。

例如当 $N=20000$ 时，相对于各类增量，我们分别得到 $B\approx 31000$，33000，35000，39000。表 7 示出了在这些实验中三个实验所得到的每一次扫描的移动次数的统计分析。奇怪的是，函数形式 $\alpha N(\ln N)^2+\beta N\ln N$ 和 βN^α 两者看来都很好地拟合观察的数据，尽管对于较小的 N 值，指数形式远比对数好得多。对于增量 2^k-1 进行了进一步的试验，让 N 的范围高达 250000；对于 $N=250000$ 的 45 回试验得出 $B\approx 7900000$，观察到的标准离差大约是 50,000。对于范围 $100\leqslant N\leqslant 250,000$ 的"最好拟合"公式可证明分别为 $1.21N^{1.26}$ 和 $.39N(\ln N)^2-2.33N\ln N$。由于指数形式的系数大致相同，而对数形式的系数则颇为剧烈地改变，故有理由猜测指数形式给出对于谢尔方法行为的真正近似。

表 7　每遍扫描的移动；当 $N=20000$ 时的例子

h_s	B_s	h_s	B_s	h_s	B_s
4095	19160	4097	19550	3280	25210
2047	15115	2049	14344	1093	28324
1023	15369	1025	15731	364	35177
511	18891	513	18548	121	47158
255	22306	257	21827	40	62110
127	27400	129	27814	13	83324
63	35353	65	33751	4	71599
31	34677	33	34303	1	34666
15	51054	17	48044		
7	40382	9	35817		
3	24044	5	19561		
1	16789	3	9628		
		1	13277		

这个经验数据并不是十全十美的。并且已经发现，对于什么增量序列在算法 D 中是最好的，没有理由作武断的断言。因为增量 $(3^k-1)/2$ 不会引起多得多的移动，而且因为它们要求的扫描数仅为其它形式增量所要求扫描数的 5/8，这个证据指出有理由以下列方式选择增量：

$$\text{设 } h_1=1,\ h_{s+1}=3h_s+1 \quad \text{而且当 } h_{t+2}\geqslant N \text{ 时以 } h_t \text{ 停止} \tag{8}$$

表插入　现在我们放下谢尔方法并考虑对于直接插入的其它类型的改进。对一个给定算法进行改进的最重要的一般方式之一是仔细地检验它的数据结构。因为，为避免不必要的操作而进行的数据结构的再组织，常常导致实质性的改进。2.4 节中有这个一般想法的进一步讨论，那里研究了一个颇为复杂的算法；我们考虑如何把它应用于象直接插入这样非常简单的算法。对于算法 S，什么是最适当的数据结构呢？

直接插入包括两个基本的操作：

i）扫描一个有序文件来找出小于或等于一个给定键的最大键；

ii）插入一个新的记录到一个有序文件的指定部分。

这个文件显然是一个线性表，而算法S通过使用顺序分配（2.2.2节）来处理这个表；因此，为完成每一个插入操作，必须移动大约一半记录。另一方面，我们知道，链接分配（2.2.3节）对插入比较理想，因为仅仅需要改变较少的链接；而且另一个操作，即顺序扫描，对链接分配也大致象顺序分配一样容易进行。由于总是从同一方向来扫描这张表，故只需要单路链接。因此得出结论，对于直接插入"适用的"数据结构是单路链接的线性表。不难修正算法S，以便按递增的次序来扫描这个表：

算法L（表插入） 假定记录$R_1,\cdots,R_N$包含键$K_1,\cdots K_N$，且"链接场"$L_1,\cdots,L_N$能容纳数字0到N；在这个文件的开始处，一个人为的记录R_0中有另一个链接场L_0。本算法设置链接场，使得这些记录以递增的顺序链接在一起。于是，如果$p(1)\cdots p(N)$是使得$K_{p(1)}\leqslant\cdots\leqslant K_{p(N)}$的"稳定的"排列，则这个算法将得出

$$L_0=p(1);\ L_{p(i)}=p(i+1),\ 对于\ 1\leqslant i\leqslant N;\ L_{p(N)}=0 \qquad (9)$$

L1.〔对j进行循环〕 置$L_0\leftarrow N$，$L_N\leftarrow 0$。（L_0起表"头"的作用，0起空链的作用；因此这个表实质上是循环的。）对于$j=N-1,N-2,\cdots,1$；执行步骤L_2到L_5；然后终止本算法。

L2.〔给p，q，K赋值〕 置$p\leftarrow L_0$，$q\leftarrow 0$，$K\leftarrow K_j$。（在下列步骤中我们将通过把K按递增次序同以前的键进行比较，把R_j插入到链接表中的适当位置。变量p和q起指向这个表的当前位置的指针作用，而且$p=L_q$，所以q比p晚一步。）

L3.〔比较K，K_p〕 如果$K\leqslant K_p$则转向步骤L5。（已经在这个表的R_q和R_p之间，为记录R找到了所要求的位置。）

L4.〔碰撞p，q〕 置$q\leftarrow p$，$p\leftarrow L_q$。如果$p>0$，则转回到步骤$L3$。（如果$p=0$，则K是至今发现的最大的键；因此记录R列入这个表的末端，在R_q和R_0之间。）

L5.〔插入表中〕 置$L_q\leftarrow j$、$L_j\leftarrow p$。

这个算法是重要的，不仅仅因为它是简单的排序方法，而且还因为它经常作为其它表处理算法的一部分出现。表8示出了当对16个例数进行排序时开头几步的情况。

表8 表插入的例子

j:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
K_j:	—	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
L_j:	16	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	0
L_j:	16	—	—	—	—	—	—	—	—		—	—	—	—	—	0	15
L_j:	14	—	—	—	—	—	—	—	—	—	—	—	—	—	16	0	15

程序L（表插入） 我们假定K_j存于INPUT+j(0:3)中，L_j存于INPUT+j(4:5)中。rI1≡j，rI2≡p；rI3≡q；rA(0:3)≡K。

```
01    KEY     EQU    0:3
02    LINK    EQU    4:5
```

03	START	ENT1	N	1	L1. 对 j 进行循环，$j \leftarrow N$
04		ST1	INPUT(LINK)	1	$L_0 \leftarrow N$
05		STZ	INPUT+N(LINK)	1	$L_N \leftarrow 0$
06		JMP	6F	1	转去使 j 减值
07	2H	LD2	INPUT(LINK)	$N-1$	L2. 给 p，q，K 赋值，$p \leftarrow L_0$
08		ENT3	0	$N-1$	$q \leftarrow 0$
09		LDA	INPUT,1	$N-1$	$K \leftarrow K_j$
10	3H	CMPA	INPUT,2(KEY)	$B+N-1-A$	L3. 比较 K，K_p
11		JLE	5F	$B+N-1-A$	如果 $K \leqslant K_p$，则转 L5
12	4H	ENT3	0,2	B	L4. 碰撞 p，q，$q \leftarrow p$
13		LD2	INPUT,3(LINK)	B	$p \leftarrow L_q$
14		J2P	3B	B	如果 $p>0$，则转 L3
15	5H	ST1	INPUT,3(LINK)	$N-1$	L5. 插入表中，$L_q \leftarrow j$
16		ST2	INPUT,1(LINK)	$N-1$	$L_j \leftarrow p$
17	6H	DEC1	1	N	
18		J1P	2B	N	$N>j \geqslant 1$

这个程序的运行时间是 $7N+14N-3A-6$ 个单位，其中 N 是文件的长度，A 是自右到左极大值的个数，B 是原来排列中反序的个数。（参考程序 S 的分析。注意程序 L 不重新排列内存中的这些记录；这可以如习题 5.2-12 中那样，以大约 $20N$ 的额外时间单位完成之。）程序 S 需要 $(9B+10N-3A-9)$ u 时间单位。而且由于 B 大约是 $\frac{1}{4}N^2$，我们可以看出，用作链接场的额外存储空间已经使我们节省了大约22%的执行时间。通过周密的程序设计，还可以再节省 22%（见习题 33），但运行时间保持同 N^2 成比例。

兹对至今所作的工作做一总结　我们从一个简单和自然的排序算法 S 开始，它进行大约 $\frac{1}{4}N^2$ 次比较和 $\frac{1}{4}N^2$ 次移动。一方面，通过考虑二叉插入来进行改进，后者大约进行 $N\log_2 N$ 次比较和 $\frac{1}{4}N^2$ 次移动。通过"两路插入"稍稍改变数据结构，使移动次数减少到大约 $\frac{1}{8}N^2$。谢尔的减少增量排序技术对于在实用范围中的 N，使比较和移动次数减少到大约 $N^{1.3}$；当 $N\to\infty$ 时，这个数可以被减少到阶为 $N(\log_2 N)^2$。对算法 S 进行改进的另一条途径是使用一个链接的数据结构，结果就得到表插入方法，它大约进行 $\frac{1}{4}N^2$ 次比较，0 次移动和 $2N$ 次改变链接。

能不能把这些方法的最好特性结合起来，使之既象二叉插入那样把比较次数减少到 $N\log_2 N$ 的阶数，又象表插入那样减少移动的次数？答案是肯定的，方法是采取一个树形结构的排列。这个可能性首先是由 D. J. 惠勒（D. J. Wheeler）于 1957 年阐明的，他建议使用两路插入，直到有必要移动某些数据时为止；然后，不去移动这些数据，而是插入指向另一内存区域的指针，而且同样的技术又递归地应用到所有有待插入到这个新内存区域去的项目上。惠勒最初的方法〔见 A. S. Douglas, *Comp. J.* 2(1959), 5〕是顺序和链接的内存的一个复杂的组合，亦带有可变大小的节点；对于我们的 16 个例数，将形成图 13 的树。使用二叉树，一个类似的但较为简单的树插入方案是由 C. M. 伯纳斯-李（C. M. Berners-Lee）大约于 1958 年独立地设计出来的〔见 *Comp. J.* 3(1960), 174, 184〕。

由于这个方法和它的改进对于查找和排序都是十分重要的，我们将在第六章详细地进行讨论。

还有另一条改进直接插入法的途径，是考虑一次插入若干个项目。例如，如果我们有一个 1000 个项目的文件，而且如果它们中的 998 个项目已经排好序，则算法 S 还要再对整个文件进行两次扫描（首先插入 R_{999}，然后插入 R_{1000}）。如果我们首先比较 R_{999} 和 R_{1000}，看哪个更大，然后通过对这个文件的一次考察来插入它们俩，则很明显可以节省时间。这种类型的组合操作，大约包含 $\frac{3}{2}N$ 个比较和移动（参考习题3.4.2-5），而不是每次大约包含 $\frac{1}{2}N$ 个比较和移动的两次扫描。

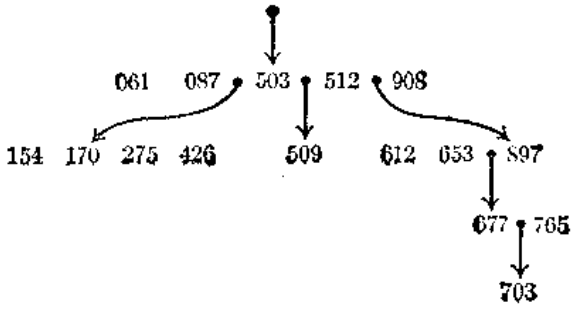


图13 惠勒树插入方案的例子

换句话说，一般说来，对于要求长时间的查找操作，进行“分批处理”是一个好的想法，使得多个操作可以一起完成。如果把这个想法引向其自然的结论，则就会重新发现通过合并进行排序的方法，它相当重要，以后要用单独一节对它进行讨论。

地址计算排序 确实，至今已经详细地讨论了对于简单的直接插入方法的所有可能的改进途径；但让我们再次来考察一下！假设某人给了你一张白纸，并说他正要向你读某些词，你的任务是把这些词以字母顺序排列，并把在这张纸上排好序的表给他。那么，如果他告诉你以 A 开头的一个词，你势必把它写在这页纸上靠顶部的地方，而一个以 Z 开头的词，大概将放在靠底部处，等等。当按作者的姓名次序把书放在一个书架上时，使用的也是一种类似的技术，如果这些书是以随机次序放在地板上给你的：你一边判断它最好的位置，一边适当地放置每一本书。借以减少你所要作的比较和移动的次数（如果你的书架容量稍大于所需要的容量的话，整个过程会更有效些）。这个办法首先由伊萨克（Isaac）和辛格尔顿（Singleton）提出用于计算机的排序，见 *JACM* 3 (1956)，169-174，而且已由克朗马尔（Kronmal）和塔塔尔（Tartar）进一步加以发展，见 *Proc. ACM Nat'l Conf.* 21 (1966)，331-337。

地址计算排序通常要求同 N 成比例的附加存储空间，或者用来保留足够的空间使得不需要过多的移动，或者用来保留辅助的表格，用这些表格帮助处理键分布的不规则性（见“分布计数”排序，算法 5.2D，它是地址计算的一种形式）。如果如同在表插入方法中那样把这个附加的存储空间提供给链接场，那么，也许可以最好地使用它们。由此，还可以避免为输入和输出开设独立的区域；一切都能够在同一个存储区域中完成。

基本的想法是来推广表插入，我们将保持若干张表，而不仅是一张。每张表被用于键的某些范围。我们作重要的假定，即假定这些键都是相当均匀地分布的，而不是不规则地“聚集起来的”：把键的所有可能值的集合划分成 M 部分，并假定一个给定的键落入一个给定部分的概率是 $1/M$。然后，提供 M 个表头的附加存储，而且每一个表就象在简单的表插入中那样来处理。

这里不必详细地给出算法；这个方法只是简单地以把所有表头置为 Λ 开始。当每个

新项目进入时，首先决定它的键落入M个部分中的哪一个，然后就象在算法L中那样把它插入到对应的表中。

为了说明这个方法，假设用于例子中的16个键被分成$M=4$个范围0-249，250-499，500-749，750-999。随着排序的进行，得到下列的配置。

	4个项目进入	8个项目进入	12个项目进入	最后的状态
表1：	061，087	061，087，170	061，087，154，170	061，087，154，170
表2：		275	275，426	275，426
表3：	503，512	503，512	503，509，512，653	503，509，512，612 653，677，703
表4：		897，908	897，908	765，897，908

由于使用了链接存储，所以就不存在由于可变长度的表引起的存储分配问题。

程序M（多表插入） 在本程序中，作象在程序L中一样的假定，但是这些键必须是非负的，于是

$$0 \leqslant \mathrm{KEY} < (\mathrm{BYTESIZE})^3$$

通过把每个键乘以一个适当的常数，这个程序把上述范围分为M个相等的部分。表头在单元HEAD+1到HEAD+M中。

```
01   KEY     EQU    1:3
02   LINK    EQU    4:5
03   START   ENT2   M                    1
04           STZ    HEAD,2               M        HEAD[p]←A
05           DEC2   1                    M
06           J2P    *-2                  M        M≥p≥1
07           ENT1   N                    1        j←N
08   2H      LDA    INPUT,1(KEY)         N
09           MUL    =M(1:3)=             N        rA←⌊M·K_j/BYTESIZE^3⌋
10           STA    *+1(1:2)             N
11           ENT3   0                    N        q←-rA
12           INC3   HEAD+1-INPUT         N        q←LOC(HEAD[q])
13           LDA    INPUT,1              N        K←K_j
14           JMP    4F                   N        去给p赋值
15   3H      CMPA   INPUT,2(KEY)         B+N-A
16           JLE    5F                   B+N-A    若K≤K_p，则去插入
17           ENT3   0,2                  B        q←p
18   4H      LD2    INPUT,3(LINK)        B+N      p←LINK(q)
19           J2P    3B                   B+N      如果不是表的末端则转移
20   5H      ST1    INPUT,3(LINK)        N        LINK(q)←LOC(R_j)
21           ST2    INPUT,1(LINK)        N        LINK(LOC(R_j))←p
22   6H      DEC1   1                    N
23           J1P    2B                   N        N≥j≥1
```

这个程序是对一般的M写的，但是，倒不如把M固定在某个合适的值上；例如，可以选择$M=$BYTESIZE，使得这些表头都能通过一条MOVE指令来清零，并用一条LD3 INPUT,1(1:1)指令即可代替08至11行的乘法序列。在程序L和程序M之间最值得注

意的差别在于这样一个事实，即当不需进行比较时，程序M必须考虑一个空表的情况。

用M个表节省了多少时间？程序M的总运行时间是$7B+31N-3A+4M+2$个单位，其中M是表的个数，N是被排序记录的个数，而A、B分别计算自右到左的极大值个数和每个表的键当中的反序个数。(同本节的其它时间分析相反，在这里把一个非空排列的最右元素处理作一个"自右到左的极大值"，而不是忽略它。)我们已经对于$M=1$研究了A和B，当时它们的平均值分别是H_N和$\left(\frac{1}{2}\right)\binom{N}{2}$。按关于键分布的假定，一个给定的表在排序结束时恰包含n个项目的概率是"二项式"概率

$$\binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n} \tag{10}$$

因此在一般情况下A和B的平均值是

$$A_{\text{ave}}=M\sum_n\binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n}H_n \tag{11}$$

$$B_{\text{ave}}=M\sum_n\binom{N}{n}\left(\frac{1}{M}\right)^n\left(1-\frac{1}{M}\right)^{N-n}\binom{n}{2} \tag{12}$$

由定理 1.2.7A

$$\sum_n\binom{N}{n}(M-1)^{-n}H_n=\left(1-\frac{1}{M}\right)^{-N}(H_N-\ln M)+\epsilon$$

$$0<\epsilon=\sum_{n>N}\frac{1}{n}\left(1-\frac{1}{M}\right)^{n-N}<\frac{M-1}{N+1}.$$

因此

$$A_{\text{ave}}=M(H_N-\ln M)+\delta,\quad 0<\delta<\frac{M^2}{N+1}\left(1-\frac{1}{M}\right)^{N+1} \tag{13}$$

(这个公式当$M\approx N$时实际是没有用的。当$M=N/2$时，A_{ave}的渐近行为的一个更详细的讨论，在习题 5.2.2-57 中)。使用恒等式

$$\binom{N}{n}\binom{n}{2}=\binom{N}{2}\binom{N-2}{n-2}$$

(它是等式 1.2.6-20 的一个特殊情况)(11)中的和即可容易地计算出来，因而得到

$$B_{\text{ave}}=\frac{1}{2M}\binom{N}{2} \tag{14}$$

(B的标准离差见习题 37。)因此对于固定的M，当$N\to\infty$时，程序M的总运行时间是

$$\begin{aligned}
\min\quad & 31N+M+2\\
\text{ave}\quad & 1.75N^2/M+31N-3MH_N+3M\ln M+4M\\
&\qquad -3-1.75N/M+2\\
\max\quad & 3.50N^2+24.5N+4M+2
\end{aligned} \tag{15}$$

注意，当M不太大时，我们把平均时间加快M倍；$M=10$大约要比$M=1$快十倍！然而，极长时间比平均时间要大得多；这重申了关于键分布十分均等的假定，因为当所有记录都

堆积到同一个表时，出现了最坏的情况。

如果置$M=N$，则平均运行时间近似于$34.36N$；当$M=\frac{1}{2}N$时，它稍大些，近似于$34.52N$；而当$M=\frac{1}{10}N$时，它近似于$48.04N$。（注意这些MIX时间单位中有$10N$个是花费在乘法指令中的！）仅仅假定这些键相当好地散布在它们的范围中，我们实现了一个阶为N的排序方法。

习题

1.〔10〕 算法S是一个"稳定的"排序算法吗？

2.〔11〕 如果把步骤S3中的关系"$K\geqslant K_i$"改为"$K>K_i$"，则算法S是否仍将正确地排序？

▶3.〔30〕 程序S是不是用MIX所能编写的最短的排序程序，换句话说，是否有实现同样效果的一个更短的程序？

▶4.〔M20〕 求出程序S的极小和极大运行时间（作为N的一个函数）。

▶5.〔M27〕 给定$\{1,2,\cdots,N\}$的一个随机排列作为输入，求程序S总运行时间的生成函数$g_N(z)=\sum_{k\geqslant 0}p_{Nk}z^k$，其中$p_{Nk}$是程序S恰好花$k$个单位时间的概率。此外对给定的$N$，计算运行时间的标准离差。

6.〔23〕 表2说明的两路插入方法，似乎意味着除了有一个能容纳N个记录的输入区域之外，还有能容纳达$2N+1$个记录的一个输出区域。说明两路插入可以仅仅使用足以存放$N+1$个记录的空间来完成，同时包括输入和输出在内。

7.〔M20〕 如果$a_1a_2\cdots a_n$是$\{1,2,\cdots,n\}$的一个随机排列，问什么是$|a_1-1|+|a_2-2|+\cdots+|a_n-n|$的平均值（这是$n$乘上在一个排序过程中某记录所走过的平均纯距离）？

8.〔10〕 算法D是一个"稳定的"排序算法吗？

9.〔20〕 对应于表3和4，量A和B是多少？程序D的总运行时间是多少？试讨论在这种情况下谢尔方法相对于直接插入的优点。

▶10.〔22〕 如果以步骤D3开始时$K_j\geqslant K_{j-h}$，则算法D指定了许多什么贡献也没有的动作。说明怎样修改程序D，使得可以避免这些多余的操作，并讨论这样一个修改的优点。

11.〔M10〕 在象图11中的一个格中，对应于排列1 2 5 3 7 4 8 6 9 11 10 12的通路是什么？

12.〔M20〕 证明在一个格通路和楼梯通路之间的区域（如图11中所示）等于对应2有序排列中的反序的个数。

▶13.〔M22〕 说明怎样把权放置到一个格的水平线段上，而不是垂直线段上，使得在格的一条通路上水平的权之和，是对应的2有序排列中反序的数目。

14.〔M24〕 (a) 说明在由等式(2)所定义的和中，$A_{2n+1}=2A_{2n}$。(b) 如果我们置$r=-s-1$，$t=1$，习题1.2.6-26的一般恒等式简化为

$$\sum_k\binom{2k+s}{k}z^k=\frac{1}{\sqrt{1-4z}}\left(\frac{1-\sqrt{1-4z}}{2z}\right)^s$$

通过考虑和式 $\sum_n A_{2n}z^n$，证明

$$A_{2n}=n\cdot 4^{n-1}$$

▶15.〔$HM33$〕 设 $g_n(z)$、$\bar{g}_n(z)$、$h_n(z)$、$\bar{h}_n(z)$ 是对从（0，0）到（n，n）的长度为 $2n$ 的所有格通路求和的 Σ_n 通路的总权数，其中对通路的顶点有若干限制：对于 $h_n(z)$，没有限制，但对于 $g_n(z)$，这条通路必须回避所有使 $i>j$ 的顶点（i，j）；$\bar{h}_n(z)$ 和 $\bar{g}_n(z)$ 的定义与此类似，但要排除对于 $0<i<n$ 的所有顶点（i，i）。于是

$$g_0(z)=1,\ g_1(z)=z,\ g_2(z)=z^3+z^2;\ \bar{g}_1(z)=z,\ \bar{g}_2(z)=z^3;$$
$$h_0(z)=1,\ h_1(z)=z+1,\ h_2(z)=z^3+z^2+3z+1;$$
$$\bar{h}_1(z)=z+1,\ \bar{h}_2(z)=z^3+z$$

试求出定义这些函数的递归关系，并使用递归关系来证明

$$h_n''(1)+h_n'(1)=\frac{7n^3+4n^2+4n}{30}\binom{2n}{n}$$

（因此容易找出{1，2，…，$2n$}的一个随机2有序排列中反序个数方差的精确公式；它渐近于$\left(\frac{7}{30}-\frac{\pi}{16}\right)n^3$）。

16.〔$M24$〕 求{1，2，…，n}的一个 h 有序排列中反序的极大个数的公式。当增量满足整除性条件（5）时，在算法D中最多能有多少次移动？

17.〔$M21$〕 说明，若对 $t\geqslant s\geqslant 1$ 有 $N=2^t$ 和 $h_s=2^{t-1}$，则存在{1，2，…，N}的唯一排列，它使由算法D执行的移动操作个数取极大值。试求描述这个排列的一个简单方式。

18.〔$HM24$〕 对于很大的 N，和数（6）可被估计为

$$\frac{1}{4}\ \frac{N^2}{h_t}+\frac{\sqrt{\pi}}{8}\left(\frac{N^{3/2}h_t^{1/2}}{h_{t-1}}+\cdots+\frac{N^{3/2}h_2^{1/2}}{h_1}\right)$$

当 N 和 t 固定且 $h_t=1$ 时，$h_t,\cdots,h_1$ 的什么实数值使这个表达式取极小值？

▶19.〔$M25$〕 在程序D的计时分析中，当增量满足整除性条件（5）时，量 A 的平均值是多少？

20.〔$M20$〕 证明定理K可从引理L推得。

21.〔$M25$〕 设 h 和 k 是互质的正整数，证明：当 a 和 b 是非负整数时，不能表示成 $ah+bk$ 的形式的最大整数，是 $hk-h-k$。因此，如果一个文件是 h 有序和 k 有序的，则当 $j-i\geqslant(h-1)(k-1)$ 时，有 $K_i\leqslant K_j$。

22.〔$M30$〕 证明所有 $\geqslant 2^h(2^k-1)$ 的整数均可表示成 $a_0(2^k-1)+a_1(2^{k+1}-1)+a_2(2^{k+2}-1)+\cdots$ 的形式，其中，诸 a_j 是非负整数。但 $2^h(2^k-1)-1$ 则不能这样表示。而且，恰有 $2^{k-1}(2^h+h-3)$ 个正整数是不能以这样的形式表示的。

▶23.〔$M22$〕 证明如果 h_{s+2} 和 h_{s+1} 互质，则当算法D使用增量 h_s 时，移动次数是 $O(Nh_s+2h_{s+1}/h_s)$。提示，见习题21。

24.〔$M42$〕 证明，在指数3/2已不能再降低的意义下，定理P是最好的。

▶25.〔$M22$〕 {1，2，…，n}有多少排列既是3有序的又是2有序的？在这样一个

排列中反序的极大个数是多少？在所有这样的排列中反序的总数是什么？

26.〔$M35$〕 如果N个元素的一个文件是3有序、5有序和7有序的，则它能有多于N个反序吗？

27.〔$M50$〕 给定N，h_t，h_{t-1}，…，h_1，求一个有效的算法，来构造使算法D中移动次数取极大值的排列。

28.〔15〕 从程序D的观点看，并考虑总运行时间，表6中所示的哪个增量序列是最好的？

29.〔$M42$〕 对于$N=1000$以及各种t值，试求$h_t,\cdots,h_1$的经验值，它在这样一种意义下使程序D所作的平均移动次数取极小值，即如果诸h中的任何一个在改变，而其它被固定，则移动的平均次数就增加。

30.〔$M23$〕（沃·普拉特） 如果增量集合是$\{2^p3^q|2^p3^q<N\}$，试证明扫描次数近似于$\frac{1}{2}(\log_2 N)(\log_3 N)$，而每次扫描的移动次数至多是$N/2$。事实上，如果对于任何一次扫描$K_{j-h}>K_j$，则将总有$K_{j-2h}$，$K_{j-2h}\leqslant K_j<K_{j-h}\leqslant K_{j+h}$，$K_{j+2h}$，所以可以简单地交换$K_{j-h}$和$K_j$并使$j$增加$2h$，这节省了算法D的两次比较。提示：见习题25。

▶31.〔25〕 为普拉特的排序算法写一个MIX程序（习题30）。试借用程序D中的那些量A，B，S，T，N，来表达它的运行时间。

32.〔10〕 如果表8中的表插入排序一直进行到完成为止，则L_0，L_1，…，L_{18}的最后内容将是什么？

▶33.〔25〕 试求改进程序L的一种方法，使得主宰它的运行时间的是$5B$而不是$7B$，其中B是反序的数目。试讨论对于程序S的相应的改进。

34.〔$M10$〕 验证公式（10）。

35.〔18〕 假设MIX的字节大小是100，表8中十六个键的例子实际上是503000，087000，512000，…，703000。当$M=4$时试对于这组数据确定出程序L和M的运行时间。

36.〔$HM16$〕 如果一个程序的运行近似地需要$A/M+B$个时间单位，并使用$C+M$个内存单元，则怎样选择M可给出最佳的空间和时间的乘积？

37.〔$M25$〕 设$g_n(z)$是n个对象的一个随机排列中反序的生成函数（参考等式5.1.1-11）。设$g_{NM}(z)$是程序M中量B的对应生成函数。证明

$$\sum_{N\geqslant 0} g_{NM}(z)M^N w^N/N! = \left(\sum_{m\geqslant 0} g_n(z)w^n/n!\right)^M$$

并使用这个公式导出B的方差。

38.〔$HM23$〕〔R. M. 卡普（R. M. Karp）〕 设$F(x)$是一个概率分布的分布函数，且$F(0)=0$及$F(1)=1$。若键K_1，K_2，…，K_N是从这个分布中随机地独立选择的，而且$M=cN$，其中c是常数，$N\to\infty$，试证明当F充分光滑时，程序M的运行时间是$O(N)$（当$\lfloor MK\rfloor=j-1$时，一个键K被插入到表j中；这以$F(j/M)-F((j-1)/M)$的概率出现。正文中仅讨论了$F(x)=x$，$0\leqslant x\leqslant 1$的情况）。

39.〔$M25$〕（图书馆中的混乱） 一个图书馆的临时用户常常把书放回到书架上错误

的地方。有一种方法可以度量出现于一个图书馆中的混乱，即如果从一个位置上拿出一本书，并把它插入到另一个位置上去，为了使得所有的书都恢复到正确的次序，最少需要这样做多少次？

于是设 $\pi=a_1a_2\cdots a_n$ 是 $\{1,2,\cdots,n\}$ 的一个排列。一个"删去插入操作"把 π 变成

$$a_1\cdots a_{i-1}a_{i+1}\cdots a_ja_ja_{j+1}\cdots a_n \text{ 或者 } a_1\cdots a_ja_ia_{j+1}\cdots a_{i-1}a_{i+1}\cdots a_n$$

(对某个 i 和 j)。设 $\mathrm{dis}(\pi)$ 是将 π 排成有序的删去-插入操作的极小数目，试问能否利用 π 的较简单的特征，来表达 $\mathrm{dis}(\pi)$。

40. 〔40〕 试对减少增量排序的下述变形进行试验，目的是加快"内部循环"：对于 $s=t,\ t-1,\cdots,2$，以及对于 $j=h_s+1,\ h_s+2,\ \cdots,\ N$，如果 $K_{j-h_s}>K_j$，则交换 $R_{j-h_s}\leftrightarrow R_j$（于是，交换的结果不会扩散出去；不进行 $K_{j-h_s}:K_{j-2h_s}$ 的比较。这些记录不总是 h_s 有序的，但这些预扫描将有助于减少反序的数目）。最后应用直接插入以完成这个排序。

5.2.2 通过交换进行排序

现在回过来讨论在 5.2 节开头提到的第二类排序算法："交换"或"换位"方法，它系统地交换反序的元素对偶，直到不再存在这样的对偶为止。

直接插入算法 5.2.1S 的过程，可以看成是一个交换方法：我们基本上把每个新记录 R_j 同它左边的相邻者进行交换，直到它已经插入到适当的位置为止。所以，把排序方法分成各种类型，诸如"插入"、"交换"、"选择"等等，其界线并不总是很分明的。在这一节将讨论以交换为主要特征的四种类型的排序方法。它们是：交换选择（"气泡排序"）；合并交换（巴切尔的平行排序）；分划交换（霍尔的"快速排序"）；以及基数交换。

气泡排序 也许通过交换进行排序的最明显的方法是比较 K_1 和 K_2，如果这些键不是顺序的，就交换 R_1 和 R_2；然后对记录 R_2 和 R_3，R_3 和 R_4 等等进行相同的工作。在这一系列操作期间，具有较大键的记录势必向右移动，而且事实上具有最大键的记录将移动到成为 R_N 为止。重复这个过程，就得到在位置 R_{N-1}、R_{N-2} 等处的适当记录，使得所有记录最终地被排好序。

图 14 示出了对于十六个键 503 087 512…703 进行的这个排序方法。垂直地而不是水平地表示数的文件，并以 R_N 在顶部和 R_1 在底部，是适宜的。这个方法称为"气泡排序"，因为较大的元素"上浮"到它们适当的位置，同"下沉排序"（即直接插入）相反。在下沉排序中，元素下沉到一个适当的位置。气泡排序还有比较乏味的名字，例如称为"交换选择"或"扩散"等。

在每次扫过这文件之后，不难看出，上边的所有记录，包括最后被交换的那个记录在内，都必然处于它们的最后位置上，所以在随后的扫描中无须考察它们。图 14 中的水平线根据这个观点示出了这个排序的过程，例如注意，在第四趟之后，已知有多于五个元素处于最后位置。在最后一趟扫描时已全然不实施任何交换了。通过这些考察，我们就容易系统地阐述这个算法了。

算法 B（气泡排序） 适当地重新排列记录 $R_1,\cdots,R_N$；在排序完成之后，它们的键便是有序的，$K_1\leqslant\cdots\leqslant K_N$。

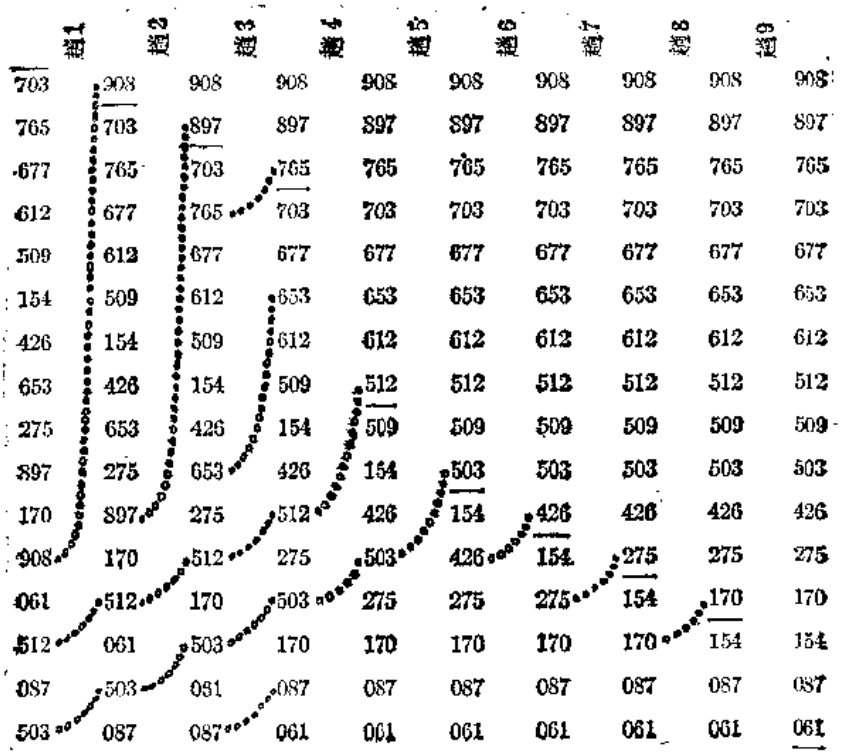


图14　进行中的气泡排序

B1.〔给 BOUND 赋初值〕　置 BOUND←N。(BOUND 是尚不知其记录是否已处于其最后位置上的最高下标；于是，这表示此刻我们尚一无所知。)

B2.〔对 j 进行循环〕　置 $t \leftarrow 0$，对 $j = 1, 2, \cdots$, BOUND-1 执行步骤 B3，然后转到步骤 B4。(如果 BOUND＝1，这意味着直接转到 B4。)

B3.〔比较/交换 $R_j : R_{j+1}$〕　如果 $K_j > K_{j+1}$，则交换 $R_j \leftrightarrow R_{j+1}$，并置 $t \leftarrow j$。

B4.〔还要交换?〕　如果 $t = 0$，则此算法终止。否则置 BOUND←t，并返回到步骤 B2。

程序 B（气泡排序）　如同该章中前边的 MIX 程序一样，我们假定有待排序的项目是在单元 INPUT＋1 到 INPUT＋N 中。rI1＝t；rI2＝j。

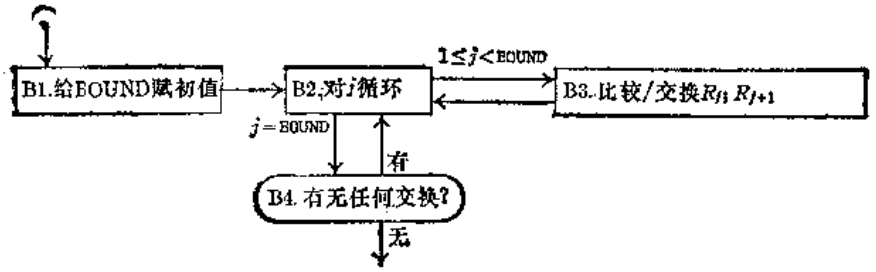


图15　气泡排序的框图

01	START	ENT1	N	1	$B1$. 给BOUND赋初值 $t+N$
02	1H	ST1	BOUND(1:2)	A	BOUND← t
03		ENT2	1	A	$B2$. 对 j 进行循环
04		ENT1	0	A	$t\leftarrow 0$
05		JMP	BOUND	A	如果 $j>$BOUND，则出口
06	3H	LDA	INPUT,2	C	$B3$. 比较1交换R_j: R_{j+1}
07		CMPA	INPUT+1,2	C	
08		JLE	2F	C	如果$K_j\leqslant K_{j+1}$，则不交换
09		LDX	INPUT+1,2	B	R_{j+1}
10		STX	INPUT,2	B	$\rightarrow R_j$
11		STA	INPUT+1,2	B	(旧的R_j)$\rightarrow R_{j+1}$
12		ENT1	0,2	B	$t\leftarrow j$
13	2H	INC2	1	C	$j\leftarrow j+1$
14	BOUND	ENTX	−*,2	$A+C$	rX← j − BOUND(被修改的指令)
15		JXN	3B	$A+C$	$1\leqslant j<$BOUND
16	4H	J1P	1B	A	$B4$. 还要交换？如果$t>0$，则转B2

气泡排序的分析　分析一下算法B的运行时间是很有益处的。在计时中涉及了三个量：扫描的趟数A，交换的次数B和比较的次数C。如果输入的键是不同的，并处于随机的次序，则可以假定它们形成1，2，…，n的一个随机排列。从反序表的想法（5.1.1节）可推出一个容易的方法，来描述在一个气泡排序中的每次扫描的效果。

定理Ⅰ　设a_1，a_2，…，a_n是{1，2，…，n}的一个排列，并设b_1，b_2，…，b_n是对应的反序表。如果气泡排序算法B的一次扫描把$a_1a_2\cdots a_n$改变成排列$a'_1a'_2\cdots a'_n$，则从$b_1b_2\cdots b_n$中把每个非零项减1，就得到对应的反序表$b'_1b'_2\cdots b'_n$。

证明　如果有一个较大的元素居于a_i前边，则居于前边的最大元素就同它进行交换，所以b_{a_i}减1。另一方面，如果没有较大的元素居于a_i前边，它就决不同一个较大的元素交换，所以b_{a_i}保持为零。

于是，通过研究在诸趟扫描中的反序表序列，即可看出在进行气泡排序期间发生的情况。例如，对应于图14的逐次的反序表是：

3 1 8 3 4 5 0 4 0 3 2 2 3 2 1 0

第1趟扫描

2 0 7 2 3 4 0 3 0 2 1 1 2 1 0 0

第 2 趟扫描

1 0 6 1 2 3 0 2 0 1 0 0 1 0 0 0

第 3 趟扫描

$$0\ 0\ 5\ 0\ 1\ 2\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \tag{1}$$

等等。因此如果 $b_1b_2\cdots b_n$ 是输入排列的反序表，我们必然有

$$A=1+\max(b_1,b_2,\cdots,b_n) \tag{2}$$

$$B=b_1+b_2+\cdots+b_n \tag{3}$$

$$C=c_1+c_2+\cdots+c_A \tag{4}$$

其中 c_j 是在第 j 趟扫描开始时 BOUND-1 的值。借助于反序表来表示，有

$$c_j=\max\{b_i+i\mid b_i\geqslant j-1\}-j \tag{5}$$

(见习题 5)。在例（1）中，因此有 $A=9$，$B=41$，$C=15+14+13+12+7+5+4+3+2=75$。对于图 14，总的 MIX 排序时间是 $960u$。

B（在一个随机排列中的反序的总数）的分布是非常熟知的，所以只剩下 A 和 C 有待分析。

$A\leqslant k$ 的概率是 $1/n!$ 乘上没有 $\geqslant k$ 的分量的反序表数目，当 $1\leqslant k\leqslant n$ 时即是 $k^{n-k}k!$。因此恰好要求 k 次扫描的概率是

$$A_k=\frac{1}{n!}(k^{n-k}k!-(k-1)^{n-k+1}(k-1)!) \tag{6}$$

由此可计算均值 ΣkA_k；作部分求和，它就是

$$A_{\text{ave}}=n+1-\sum_{0\leqslant k\leqslant n}\frac{k^{n-k}k!}{n!}=m+1-P(n) \tag{7}$$

其中 $P(n)$ 是在 1.2.11 节等式（24）中求出的渐近值为 $\sqrt{\pi n/2}-\frac{2}{3}+O(1/\sqrt{n})$ 的函数。E. H. 费兰德（E. H. Friend）在 $JACM$ 3(1956)，150 中未加证明地指出了公式（7）；霍华德·B. 德穆斯（Howard B. Demuth）给出了一个证明〔Ph. D. thesis(Stanford University: October，1956)，64-68〕。关于 A 的标准离差，见习题 7。

比较的总数 C 要更难处理些，我们将仅仅考虑 C_{ave}。对于固定的 n，命 $f_j(k)$ 是对于 $1\leqslant i\leqslant n$，使得 $b_i<j-1$ 或 $b_i+i-j\leqslant k$ 的反序表 $b_1\cdots b_n$ 的数目；于是

$$f_j(k)=(j+k)!(j-1)^{n-j-k},\quad 0\leqslant k\leqslant n-j \tag{8}$$

(见习题 8)。（5）中 C_j 的平均值是 $(\Sigma k(f_j(k)-f_j(k-1)))/n!$；进行部分求和，然后对 j 求和，即导出公式

$$C_{\text{ave}}=\binom{n+1}{2}-\frac{1}{n!}\sum_{\substack{1\leqslant j\leqslant n\\0\leqslant k\leqslant n-j}}f_j(k)=\binom{n+1}{2}-\frac{1}{n!}\sum_{0\leqslant r<s\leqslant n}s!r^{n-s} \tag{9}$$

这里渐近值不容易确定。因而将在本节末再回过头来研究它。

为了总结对于气泡排序的分析，以上和以下所导出的公式可以写成如下的结果：

$$A=(\min 1,\ \text{ave}N-\sqrt{\pi N}/2+O(1),\ \max N) \tag{10}$$

$$B=(\min 0,\ \text{ave}\ \frac{1}{4}(N^2-N),\ \max\frac{1}{2}(N^2-N)) \tag{11}$$

$$C=(\min N-1,\ \text{ave}\ \frac{1}{2}(N^2-N\ln N-(\gamma+\ln 2-1)N)+O(\sqrt{N}),$$

$$\max\frac{1}{2}(N^2-N)) \tag{12}$$

在每种情况下，当输入已处于有序时即出现极小值；所以 MIX 运行时间是 $8A+7B+8C+1=(\min 8N+1,\ \text{ave}5.75N^2+O(N\ln N),\ \max 7.5N^2+0.5N+1)$。

气泡排序的精化 为分析气泡排序做了大量的工作，尽管用于这些计算的技术是有教益的，但结果却是令人失望的，因为它告诉我们，气泡排序并不是非常好的。同直接插入相比（算法5.2.1S），气泡排序需要相当复杂的程序，而且花费大约两倍长的时间！

为了改进气泡排序，可以给出某些建议。例如，在图14中，第4趟扫描中的头一个比较是多余的，第5趟扫描中的头两个和在第6趟及第7趟扫描中的头三个也是多余的。还要注意，每次扫描时元素向左移动决不能多于一步；所以，如果开始时最小的项目正好靠近右端，则我们就被迫进行最大次数的比较。这就提出了“鸡尾混合排序”，它以相反的方向交替地进行扫描（见图16）。这个方法稍微减少了进行比较的平均次数。在这方面 K. E. 艾弗森（K. E. Iverson）〔A Programming Language（Wiley，1962），218-219〕已经进行了有趣的考察：如果 j 是对于在相反方向的两次连续扫描下 R_j 和 R_{j+1} 彼此不交换的一个下标，则 R_j 和 R_{j+1} 必须都处于它们最后的位置，而且它们都不必进行任何随后的比较。例如，从左到右遍历 4 3 2 1 8 6 9 7 5 得出 3 2 1 4 6 8 7 5 9；在 R_4 和 R_5 之间没有出现交换，从右到左遍历后一个排列，R_4 仍然小于（新的）R_5，所以可以立即得出结论，R_4 和 R_5 无须进行任何进一步的比较。

703	908	908	908	908	908	908	908
765	703	765	897	897	897	897	897
677	765	703	765	765	765	765	765
612	677	677	703	703	703	703	703
509	612	612	677	677	677	677	677
154	509	509	612	612	653	653	653
426	154	426	509	509	612	612	612
653	426	653	426	653	509	512	512
275	653	275	653	426	512	509	509
897	275	897	275	512	426	503	503
170	897	170	512	275	503	426	426
908	170	512	170	503	275	275	275
061	512	154	503	170	170	170	170
512	061	503	154	154	154	154	154
087	503	087	087	087	087	087	087
503	087	061	061	061	061	061	061

图16 “鸡尾混合排序”

但是这些精化中没有一个能导出比直接插入更好的算法（而且我们已经知道，对于很大的 N，直接插入是不适当的）。另一个想法是消除大多数的交换，因为在一个交换期间大多数元素都只不过左移一步，故通过移动下标的起点，换一种方式考察这个数组，就可以得到同样的效果！但得到的算法也不比后面将要研究的直接选择算法5.2.3S更好。

简言之，气泡排序除开它迷人的名字和导致了某些有趣的理论问题这一事实之外，似乎没有什么值得推荐的。

巴切尔平行法 如果想要有一个其运行时间比N^2的阶数快的交换算法，就需要选择某个不相邻的键对偶（K_i，K_j）进行比较；否则，就需要象原来排列的所有反序那样多的交换，而反序的平均数为$\frac{1}{4}(N^2-N)$。考察可能的交换，巧妙地编排比较序列的方法是1964年由K.E.巴切尔（K.E.Batcher）发现的〔Proc. AFIPS. Spring Joint Computer Conference 32 (1968), 307-314〕。他的方法并不是十分显然的，由于只进行相当小的比较，因此需要相当错综的论证，才能证明它是正确的。我们将讨论两个证明，一个在这一节中，另一个在5.3.4节中。

巴切尔的排序方案同谢尔的排序有些相似，但是比较是在一种新奇的途径下完成的，以使交换的扩散成为不必要。于是可以把表1与表5.2.1-3进行类比；巴切尔方法达到了8，4，2和1排序的效果。但是这些比较都不重叠。因为巴切尔的算法实质上是合并排好序的子序列对偶，故可以称作“合并交换排序”。

算法M（合并交换） 适当地重新排列记录$R_1,\cdots,R_N$，在完成排序之后，它们的键将是有序的：$K_1\leqslant K_2\leqslant\cdots\leqslant K_N$。假定$N\geqslant 2$。

M1.〔给p赋初值〕 置$p\leftarrow 2^{t-1}$，其中$t=\lceil \lg N\rceil$是使得$2^t\geqslant N$的最小整数（将对$p=2^{t-1}$，2^{t-2}，…，1实施步骤M2到M5）。

M2.〔给q，r，d赋初值〕 置$q\leftarrow 2^{t-1}$，$r\leftarrow 0$，$d\leftarrow p$。

M3.〔对i进行循环〕 对于使得$0\leqslant i<N-d$和$i\wedge p=r$的所有i，执行步骤M4。然后转到步骤M5。（这里$i\wedge p$指的是i和p的二进表示的“逻辑与”；对每个二进位来说，除非i和p两数在该位上的值都为1，否则结果为零。例如$13\wedge 21=(1101)_2\wedge(10101)_2=5$。这时，$d$是$p$的奇数倍，而且$p$是2的一个乘方，使得$i\wedge p\neq(i+d)\wedge p$；由此得出，步骤M4的动作可以以任意次序，甚至同时对所有有关的i来完成。）

M4.〔比较/交换$R_{i+1}:R_{i+d+1}$〕 如果$K_{i+1}>K_{i+d+1}$，则交换$R_{i+1}\longleftrightarrow R_{i+d+1}$。

M5.〔对q进行循环〕 如果$q\neq p$，则置$d\leftarrow q-p$，$q\leftarrow q/2$，$r\leftarrow p$，并返回步骤M3。

M6.〔对p进行循环〕（这时，排列$K_1K_2\cdots K_N$是p有序的）置$p\leftarrow\lfloor p/2\rfloor$。如果$p>0$，则返回到M2。

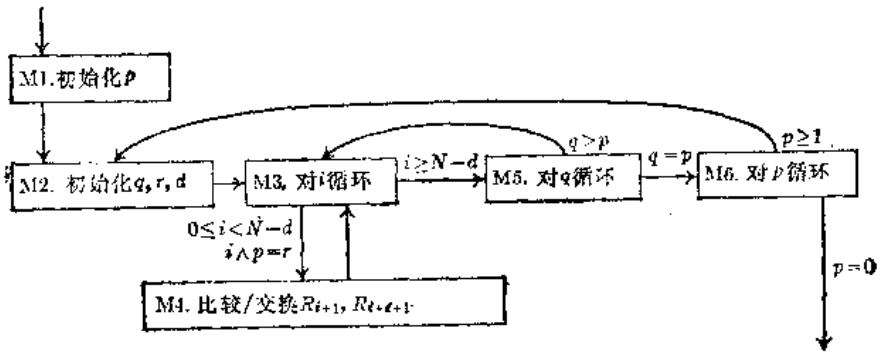


图17 算法M

表1对于$N=16$说明了本方法。注意，这个算法实质上先对R_1，R_3，R_5，…和R_2，R_4，R_6…独立地进行排序；然后，对$p=1$实施步骤M 2到M 5，为的是把两个排好序的序列合并在一起，从而完成对N个元素的排序。

为了证明在算法M中描述的比较/交换的奇妙序列实际上能对所有可能的输入文件$R_1R_2\cdots R_N$排序，仅仅需要证明，当$p=1$时步骤M 2到M 5将合并所有的2有序文件$R_1R_2\cdots R_N$。为此，可以使用5.2.1节的格盘通路方法（参考图11）；$\{1, 2, \cdots, N\}$的每个2有序排列唯一地对应于在一个格盘图示中从（0,0）到（$\lceil N/2 \rceil$、$\lfloor N/2 \rfloor$）的一条通路。图18（a）示出了$N=16$的一个例子，对应于排列1 3 2 4 10 5 11 6 13 7 14 8 15 9 16 12。当对于$p=1$，$q=2^{t-1}$，$r=0$，$d=1$执行步骤M 3时，其效果是比较（可能还有交换）$R_1:R_2$，$R_3:R_4$，等等。这个操作对应于格盘通路的一个简单变换，即在必要时相对于对角线“折叠”该通路，以使它决不落在对角线的上面部分（见图18（b）和习题10中的证明）。步骤M 3以后的叠代有$p=r=1$，以及$d=2^{t-1}-1$，$2^{t-2}-1$，…，1；它们的效果是比较/交换$R_2:R_{2+d}$，$R_4:R_{4+d}$，等等。而且也有一个简单的格盘解释：把

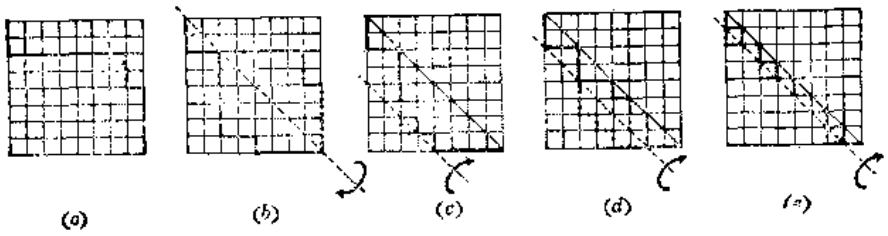


图18　巴切尔方法的一个几何解释，$N=16$

表1　合并交换排序（巴切尔方法）

p	q	r	d	
				503 087 512 061 908 170 897 275 653 426 154 509 612 677 765 703
8	8	0	8	
				503 087 154 061 612 170 765 275 653 426 512 509 908 677 897 703
4	8	0	4	
				503 087 154 061 612 170 765 275 653 426 512 509 908 677 897 703
4	4	4	4	
				503 087 154 061 612 170 512 275 653 426 765 509 908 677 897 703
2	8	0	2	
				154 061 503 087 512 170 612 275 653 426 765 509 897 677 908 703
2	4	2	6	
				154 061 503 087 512 170 612 275 653 426 765 509 897 677 908 703
2	2	2	2	
				154 061 503 087 512 170 612 275 653 426 765 509 897 677 908 703
1	8	0	1	
				061 154 087 503 170 512 275 612 426 653 509 765 677 897 703 908
1	4	1	7	
				061 154 087 503 170 512 275 612 426 653 509 765 677 897 703 908
1	2	1	3	
				061 154 087 275 170 426 503 509 512 653 612 703 677 897 765 908
1	1	1	1	
				061 087 154 170 275 426 503 509 512 612 653 677 703 765 897 908

这条通路相对于对角线下方 $\frac{1}{2}(d+1)$ 个单位处的一条线作“折叠”，见图18（c）和（d）；最后得到了图18（e）中的通路，它对应于一个完全排好序的排列。这就完成了关于巴切尔算法是正确的一个“几何证明”；我们可以称它为通过折叠进行排序！

算法M的一个MIX程序出现于习题12中。可惜，为控制比较序列所需要的簿记工作的量是相当大的，所以这个程序比我们所看到的其它方法效率都更低。但有一个重要的特性可补救它的不足：在允许进行并行计算的计算机或逻辑网络上，由步骤M3的迭代所确定的所有比较/交换都可以同时地完成。通过这样的并行操作，排序在 $\frac{1}{2}\lceil\log_2 N\rceil(\lceil\log_2 N\rceil+1)$ 步内完成，因此这大体同任何已知的一般方法一样快。例如，1024个元素使用巴切尔方法可在仅仅55个平行步骤内加以排序。最接近的竞争者是普拉特方法（见习题5.2.1-30），它使用40或73步，这看我们如何计数；如果我们打算允许重叠的比较，但不允许重叠的交换，则普拉特方法只需要40个比较/交换循环对1024个元素进行排序。关于进一步的注释，见5.3.4节。

“快速排序” 巴切尔方法中的比较序列是预先确定的；每次比较相同的键对偶，而不管关于这个文件从以前的比较中已经知道些什么。在气泡排序中，尽管算法B有限地利用了以前的知识来减少它在这个文件右端的工作，但大体上仍是如此。现在我们转到一种十分不同的策略，它使用每个比较的结果来确定下一次要比较什么键。这样一种策略对于并行计算是不适当的，但在顺序地工作的计算机上却是十分有成效的。

下列方法的基本思想是取一个记录，比如说R_1，并把它移动到在排了序的文件中它将占据的最后位置，比如说位置 s。在确定这最后的位置的同时，也重新排列其它的记录使得带有较大键的记录都位于 s 的右边，带有较小键者则位于 s 的左边。这样一来文件就将以这样一个方式被划分，即原来的排序问题归结成两个较简单的问题，就是对$R_1\cdots R_{s-1}$排序，并且（独立地）对$R_{s+1}\cdots R_N$排序。可以应用同一技术到这些子文件中的每一个上，直到这一作业完成时为止。

有若干个方法来把文件分划成左和右子文件；下面由R. 塞奇维克（R. Sedgewick）给出的方案，似乎是最好的，其原因当分析此算法时就会清楚了：保持两个指针 i 和 j，而且开始时 $i=2$ 和 $j=N$。如果 R_i 在分划之后最终要成为左子文件的一部分（通过比较K_i和K_1我们可知道这一点），则 i 增1，并且继续进行直到遇到一个属于右子文件的记录R_i为止。类似地，j 减1直到遇到属于左子文件的一个记录R_j为止。如果 $i<j$，则交换R_i同R_j；然后以同样方式处理下一记录，“两头点蜡”直到 $i\geqslant j$。通过交换R_j同R_1，最终地完成这个分划。例如，试看对于十六个数的文件所发生的情况（为了指出 i 和 j 的位置，K_i和K_j已经以黑体字标出）。

给定：	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703	减少 j
第1次交换：	**154**	087	512	061	908	170	897	275	653	426	**503**	509	612	677	765	703	增加 i
第2次交换：	154	087	**503**	061	908	170	897	275	653	426	**512**	509	612	677	765	703	减少 j
第3次交换	154	087	**426**	061	908	170	897	275	653	**503**	512	509	612	677	765	703	增加 i
第4次交换：	154	087	426	061	**503**	170	897	275	653	908	512	509	612	677	765	703	减少 j
第5次交换：	154	087	426	061	275	170	897	**503**	653	908	512	509	612	677	765	703	增加 i
第6次交换：	154	087	426	061	275	170	**503**	897	653	908	512	509	612	677	765	703	减少 j

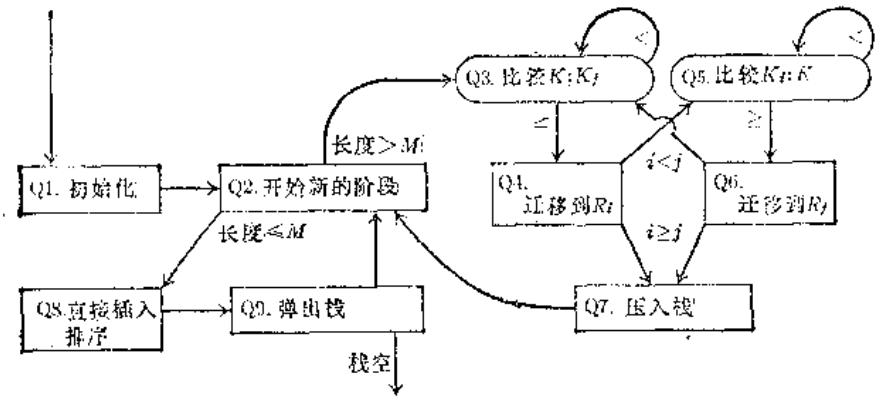


图19　分划交换排序（“快速排序”）

表2示出了怎样通过这种方法，用11个阶段把我们的示例文件完全排好序。方括弧指出了仍然需要加以排序的子文件；在一台计算机内，这些子文件可以通过两个变量 l 和 r（当前考察的子文件的边界）以及附加对偶（l_k，r_k）的一个栈来表示。每次分划这个文件，就把最大的子文件压入栈，并对另一个子文件着手进行工作，直到得到极短的非常容易排序的短文件为止；这个过程确保栈将决不包含多于大约 $\log_2 N$ 个项目，如习题20中所示那样。

表2　“快速排序”

																(l, r)	栈
【503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703】	(1,16)	—
【154	087	426	061	275	170】	503	〔897	653	908	512	509	612	677	765	703〕	(1,6)	(8,16)
【061	087】	154	〔426	275	170〕	503	〔897	653	908	512	509	612	677	765	703〕	(1,2)	(4,6)(8,16)
061	087	154	【426	275	170】	503	〔897	653	908	512	509	612	677	765	703〕	(4,6)	(8,16)
061	087	154	【170	275】	426	503	〔897	653	908	512	509	612	677	765	703〕	(4,5)	(8,16)
061	087	154	170	275	426	503	【897	653	908	512	509	612	677	765	703】	(8,16)	—
061	087	154	170	275	426	503	【703	653	765	512	509	612	677】	897	908	(8,14)	—
061	087	154	170	275	426	503	【677	653	612	512	509】	703	765	897	908	(8,12)	—
061	087	154	170	275	426	503	【509	653	612	512】	677	703	765	897	908	(8,11)	—
061	087	154	170	275	426	503	509	【653	612	512】	677	703	765	897	908	(9,11)	—
061	087	154	170	275	426	503	509	【512	612】	653	677	703	765	897	908	(9,10)	—
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	—	—

刚才描述的排序过程可以称作分划交换排序，它是由C. A. R. 霍尔(C. A. R. Hoare)提出来的。在所有已经发表过的排序方法中，他的有趣的论文〔*Comp. J.* 5(1962), 10-15〕中所讨论的方法是研究得最深的。霍尔把他的方法称为“快速排序”；这样一个名称并非滥用，因为，例如，表2中的整个排序过程仅需要进行48次比较（除了花费47次比较的二叉插入之外，这是至今所见到的最小值）。在每个阶段中，所有的比较都相对于同一个键进行，所以这个键可以保存在一个寄存器中。而且，数据移动的量是十分合理的；表2中的计算仅仅进行17次交换，其中大多数都只不过是简单的“半交换”（简单的传送），因为一个键就驻留在寄存器中，并且在一个阶段结束之前，不必进行存储。

额外的簿记（用来控制 i、j 和栈）并不困难，但它使得快速排序分划过程最适合于

很大的N；因此宜于象如下算法中那样，以一种特殊的方式来对短的子文件排序。

算法Q（分划交换排序） 适当地重新安排记录$R_1, \cdots, R_N$。在排序完成之后，它们的键将是有序的，即$K_1 \leqslant \cdots \leqslant K_N$。需要有至多$\log_2 N$个项目的一个辅助栈作为临时存储。本算法遵循上边正文中所述的"快速排序"分划过程，但为了提高效率已稍作修改：

a）假定存在人为的键$K_0 = -\infty$和$K_{N+1} = +\infty$使得

$$K_0 \leqslant K_i \leqslant K_{N+1} \quad 对于 1 \leqslant i \leqslant N \tag{13}$$

（允许相等）

b）具有M个或更少个元素的子文件通过直接插入进行排序，其中$M \geqslant 1$是一个参数，它应象下边正文中所述那样进行选择。

c）在特殊的阶段中（允许指针i，j交叉），进行一次或两次额外的比较，使得主要的比较循环能够尽可能地快。

d）交换具有相等键的记录，尽管并非绝对必须这样做（这一思想归功于R. C. 辛格尔顿，当出现相等的元素时，它有助于划分近乎一半的子文件，见习题18）。

Q1.〔初始化〕 如果$N \leqslant M$，转到步骤Q9。否则置栈为空，并置$l \leftarrow 1$，$r \leftarrow N$。

Q2.〔开始新阶段〕（现在希望对子文件$R_l \cdots R_r$进行排序；基于算法的本性，有$r \geqslant l + M$，而且对于$l \leqslant i \leqslant r$，$K_{l-1} \leqslant K_i \leqslant K_{r+1}$。）置$i \leftarrow l$，$j \leftarrow r + 1$；并置$K \leftarrow K_l$，$R \leftarrow R_l$。（下文讨论了对$K$和$R$另外的选择，它们可能是更好的。）

Q3.〔比较$K_i : K$〕（此时这个文件已被重排使得

$$对于 l - 1 \leqslant k \leqslant i，K_k \leqslant K，对于 j \leqslant k \leqslant r + 1，K \leqslant K_k \tag{14}$$

且$l \leqslant i < j$。）i增1；然后如果$K_i < K$，则重复该步骤（由于$K_j \geqslant K$，此迭代必定以$i \leqslant j$结束）。

Q4.〔比较$K : K_j$〕 j减1；然后如果$K < K_j$，则重复此步骤（由于$K \geqslant K_{l-1}$，该迭代必定以$j \geqslant i - 1$结束）。

Q5.〔比较$i : j$〕（这时，除对于$k = i$和$k = j$外，(14)成立；而且$K_i \geqslant K \geqslant K_j$，$r \geqslant j \geqslant i - 1 \geqslant l$。）如果$j \leqslant i$，交换$R_i \longleftrightarrow R_j$，并转到步骤Q7。

Q6.〔交换〕 交换$R_i \longleftrightarrow R_j$并转回步骤Q3。

Q7.〔压入栈〕（现在子文件$R_l \cdots R_i \cdots R_r$已经被分划，使得对于$l - 1 \leqslant k \leqslant i$，$K_k \leqslant K_i$且对于$j \leqslant k \leqslant r + 1$，$K_j \leqslant K_k$。）如果$r - j \geqslant j - l > M$，则把$(j + 1, r)$插到栈顶上，置$r \leftarrow j - 1$，并转到Q2。如果$j - l > r - j > M$，则把$(l, j - 1)$插入到栈顶上，置$l \leftarrow j + 1$，并转到Q2。（在栈上的每个项目$(a, b)$，是在某个将来的时刻对子文件$R_a \cdots R_b$排序的一个请求。）否则如果$r - j > M \geqslant j - l$，置$l \leftarrow j + 1$并转到Q2；或者如果$j - l > M \geqslant r - j$，置$r \leftarrow j - 1$并转到Q2。

Q8.〔弹出栈〕 如果这个栈不空，则撤销它的顶部的项目(l', r')，置$l \leftarrow l'$，$r \leftarrow r'$，并返回步骤Q2。

Q9.〔直接插入排序〕 对于$j = 2, 3, \cdots N$，如果$K_{j-1} > K_j$，则进行下列操作：置$K \leftarrow K_j$，$R \leftarrow R_j$，$i \leftarrow j - 1$；然后$R_{i+1} \leftarrow R_i$并且$i \leftarrow i - 1$，一次或多次直到$K_i \leqslant K$；然后置$R_{i+1} \leftarrow R$。（这是如同习题5.2.1-10和33所提议的那样修改过了的算法5.2.1S。如果$M = 1$，则步骤Q9可以省略。警告：最后用直接插入可以掩盖在步骤Q1～Q8中的

失误；仅当发现超负荷时才可能告诉你在哪些步骤中究竟有什么错误！）

对应的MIX程序是相当长的，但并不复杂。事实上，大部分代码是专门用于步骤Q7的，它仅仅以一种非常直截了当的方式来摆弄那些变量。

程序Q （分划交换排序） 有待排序的记录在单元INPUT＋1到INPUT＋N中；假定单元INPUT和INPUT＋N＋1分别包含MIX中最小和最大可能的值。栈保存在单元STACK＋1、STACK＋2，…中；关于栈所需单元的精确数目，见习题20。rI1≡l，rI2≡r，rI3≡i，rI4≡j，rI6≡堆栈的大小，rA≡K≡R。我们假定$N>M$。

	A	EQU	2:3		栈单元的头一个分量
	B	EQU	4:5		栈单元的第二个分量
01	START	ENT1	1	1	Q1. 初始化，$l \leftarrow 1$
02		ENT2	N	1	$r \leftarrow N$
03		ENT6	0	1	置栈为空
04	2H	ENTX	0,2	$2A+1$	Q2. 开始新阶段
05		DECX	M,1	$2A+1$	$rX \leftarrow r-l-M$
06		JXN	8F	$2A+1$	如果子文件的长度≤M则转Q8
07		ENT3	0,1	A	$i \leftarrow l$
08		ENT4	0,2	A	$j \leftarrow r$
09		LDA	INPUT,3	A	$K \leftarrow K_l$
10		JMP	3F	A	转到Q3
11	0H	LDX	INPUT,3	B	
12		STX	INPUT,4	B	$R_j \leftarrow R_l$
13		DEC4	1	$C'-A$	$j \leftarrow j-1$
14	3H	CMPA	INPUT,4	C'	Q3. 比较$K:K_j$
15		JL	*-2	C'	如果<，则j减值，并重复
16	4H	ENTX	0,3	$B+A$	Q4. 迁移到R_l
17		DECX	0,4	$B+A$	
18		JXNN	7F	$B+A$	如果$i \geq j$则转Q7
19		LDX	INPUT,4	$B+X$	
20		STX	INPUT,3	$B+X$	$R_i \leftarrow R_j$
21		INC3	1	C''	$i \leftarrow i+1$
22	5H	CMPA	INPUT,3	C''	Q5. 比较$K_i:K$
23		JG	*-2	C''	如果<，则i增值并重复
24	6H	ENTX	0,3	$B+X$	Q6. 迁移到R_l
25		DECX	0,4	$B+X$	
26		JXN	0B	$B+X$	如果$i<j$则转Q3
27		ENT3	0,4	X	$i \leftarrow j$
28	7H	STA	INPUT,3	A	Q7. 压入栈，$R_i \leftarrow R$
29		ENTA	0,2	A	
30		DECA	0,3	A	
31		DECA	0,3	A	
32		INCA	0,1	A	

33		INC6	1	A	rI6←rI6 + 1
34		JANN	1F	A	如果 $r-i \geqslant i-l$ 则转移
35		ST1	STACK,6(A)	A'	
36		DEC3	1	A'	
37		ST3	STACK,6(B)	A'	$(l,\ i-1)$ →栈
38		ENT1	2,3	A'	$l \leftarrow i+1$
39		JMP	2B	A'	到Q2
40	1H	INC3	1	$A-A'$	
41		ST3	STACK,6(A)	$A-A'$	
42		ST2	STACK,6(B)	$A-A'$	$(i+1,\ r)$⇒栈
43		ENT2	−2,3	$A-A'$	$r \leftarrow i-1$
44		JMP	2B	$A-A'$	转到Q2
45	8H	DEC2	0,1	$A+1$	Q8. 直接插入排序
46		J2NP	9F	$A+1$	如果 $r \leqslant l$，则转到Q9
47		ENT4	1,1	$A+1-L$	$j \leftarrow l+1$
48	1H	LDA	INPUT,4	D	$R \leftarrow R_j$
49		ENT3	−1,4	D	$i \leftarrow j-1$
50		JMP	* +4	D	跳入循环
51		LDX	INPUT,3	E	
52		STX	INPUT+1,3	E	$R_{i+1} \leftarrow R_i$
53		DEC3	1	E	$i \leftarrow i-1$
54		CMPA	INPUT,3	$D+E$	
55		JL	*−4	$D+E$	如果 $K<K_i$，则转移
56		STA	INPUT+1,3	D	$R_{i+1} \leftarrow R$
57		INC4	1	D	$j \leftarrow j+1$
58		DEC2	1	D	
59		J2P	1B	D	重复 $r-l$ 次
60	9H	LD1	STACK,6(A)	$A+1$	Q9. 弹出栈
61		LD2	STACK,6(B)	$A+1$	(l,r)⇐栈
62		DEC6	1	$A+1$	rI6←rI6 − 1
63		J6NN	2B	$A+1$	如果栈不空，则转Q2,

"快速排序"的分析 用程序Q表示的计时信息，不难利用克希霍夫（Kirchhoff）守恒定律（参考1.3.3节）和压入栈的每一个信息终归又被撤销这一事实导出。克希霍夫定律应用于步骤Q 2还表明

$$A=1+(S'+A'')+(S-S'+A'')+S=2S+1+A''+A''' \tag{15}$$

因此总共的运行时间为

$$24A+11B+4C+3D+8E+7N+9S\text{单位}$$

其中

A＝分划阶段数；

B＝在步骤Q 6中的交换数；

C＝在进行分划时所作比较数；

D＝在直接插入（步骤Q 9）期间$K_{j-1}>K_j$的次数；

E＝被直接插入消去的反序数；

S＝一个条款压入栈的次数。 (16)

通过分析这六个量，就能对参数M进行精巧的选择，M是用来确定直接插入和分划之间的"阈值"的。这个分析是特别有教益的，因为这个算法比较复杂；揭示这个复杂性是阐明重要技术的一个好方法。然而，非数学方面的读者，请跳到等式（25）。

象在这一章中大多数其它的分析那样，假定有待排序的这些键是不同的；习题18指出，这些键之间的相等性并不严重地损害算法Q的效率，而且事实上它们似乎还有助于提高效率。由于这个方法仅依赖于键的相对次序，也可以假定，它们只不过是在某种次序下的$\{1, 2, \cdots, N\}$。

我们可以通过考虑最初分划阶段的行为来着手解决这个问题，这个最初的阶段使我们头一次到Q 7去。一旦已经实现了这个分划，则如果原来的文件是在随机次序下的，$R_1 \cdots R_{j-1}$和$R_{j+1} \cdots R_N$两个子文件也将是在随机次序下，因为在这些子文件中元素的相对次序对分划算法没有影响。因此随后的分划的贡献可以通过对N用归纳法确定（这是一个重要的发现，因为某些违背这个性质的其它算法已经证明是相当慢的；见*Computing Surveys* 6 (December, 1974)。

设s是头一个键K_1的值，并假定$K_1, \cdots K_s$中恰有t个大于s（记住正被排序的键是整数$\{1, 2, \cdots, N\}$）。

如果$s=1$，则容易看出在分划的头一阶段发生了什么情况：步骤Q 3被执行N次，然后步骤Q 4使我们转去执行Q 7。所以在此情况下头一个阶段的贡献是$A=1$，$B=0$，$C=N$，$X=0$。当$s>1$时一个类似的但稍微复杂的论证（见习题21）说明，一般地，头一阶段对于总运行时间的贡献是

$$A=1,\quad B=t,\quad C=N+1-\delta_{s1},\quad X=h,\quad \text{对于}\ 1\leqslant s\leqslant N \tag{17}$$

对此，还要加上稍后阶段的贡献，这些阶段分别对具有$s-1$和$N-s$个元素的子文件进行排序。

如果假定原始的文件是随机次序的，则现在已有可能写下定义A，B，…，X的概率分布的生成函数公式（见习题22）。但为了简单起见，这里仅仅考虑这些量的平均值A_N，B_N，…，X_N（作为N的函数）。例如，考虑在分划过程中出现的比较平均次数C_N。当$N\leqslant M$时，$C_N=0$。由于任意给定的s值以$1/N$的概率出现，我们有

$$\begin{aligned} C_N &= \frac{1}{N}\sum_{1\leqslant s\leqslant N}(N+1-\delta_{s1}+C_{s-1}+C_{N-s}) \\ &= N+1-\frac{1}{N}+\frac{2}{N}\sum_{0\leqslant k<N}C_k \quad \text{对于}\quad N>M \end{aligned} \tag{18}$$

对于其它的量，A_N，B_N，…，X_N类似的公式成立（见习题23）。

有一个简单的方法来解形如

$$x_n=f_n+\frac{2}{n}\sum_{0\leqslant k<n}x_k \quad n\geqslant m \tag{19}$$

的递归关系。头一步是脱去和式号：

$$(n+1)x_{n+1}=(n+1)f_{n+1}+2\sum_{0\leqslant k\leqslant n}x_k$$

$$nx_n=nf_n+2\sum_{0\leqslant k\leqslant n}x_k$$

可以把两者相减，得到

$$(n+1)x_{n+1}-nx_n=g_n+2x_n,\quad 其中\ g_n=(n+1)f_{n+1}-nf_n$$

现在递归式有了简单得多的形式

$$(n+1)x_{n+1}=(n+2)x_n+g_n\qquad 对于\ n\geqslant m \tag{20}$$

任何形为

$$a_nx_{n+1}=b_nx_n+g_n \tag{21}$$

的递归关系可以约化为一个和式，只要以“求和因子” $a_0a_1\cdots a_{n-1}/b_0b_1\cdots b_n$ 乘以两边即可；我们得到

$$y_{n+1}=y_n+c_n,\quad 其中\ y_n=\frac{a_0\cdots a_{n-1}}{b_0\cdots b_{n-1}}x_n,\quad c_n=\frac{a_0\cdots a_{n-1}}{b_0b_1\cdots b_n}g_n \tag{22}$$

在情况（20）中，求和因子就是 $n!/(n+2)!=1/(n+1)(n+2)$，所以我们发现简单的关系

$$\frac{x_{n+1}}{n+2}=\frac{x_n}{n+1}+\frac{(n+1)f_{n+1}-nf_n}{(n+1)(n+2)},\quad n\geqslant m \tag{23}$$

是（19）的一个推论。

例如，如果我们置 $f_n=1/n$，则对于所有 $n\geqslant m$，得到意外的结果 $x_n/(n+1)=x_m/(m+1)$。如果我们置 $f_n=n+1$，则对于所有 $n\geqslant m$，得到

$$\begin{aligned}x_n/(n+1)&=2/(n+1)+2/n+\cdots+2/(m+2)+x_m/(m+1)\\&=2(H_{n+1}-H_{m+1})+x_m/(m+1)\end{aligned}$$

把这两个解组合在一起，并且对于 $n\leqslant M$ 置 $m=M+1$ 和 $x_n=0$，给出公式

$$\begin{aligned}C_N&=(N+1)\left(2H_{N+1}-2H_{M+2}+1-\frac{1}{(M+1)(M+2)}\right)\\&\approx 2(N+1)\ln\binom{N+1}{M+2},\quad 对于 N>M\end{aligned} \tag{24}$$

在 6.2.2 节，将证明 C_N 的标准离差渐近于 $\sqrt{(21-2\pi^2)/3}\,N$；这与（24）相比是相当小的。

以类似的方式，可以求出其它的量（见习题 23）；我们有

$$\begin{aligned}A_n&=2(N+1)/(M+2)-1,\\B_N&=\frac{1}{6}(N+1)\left(2H_{N+1}-2H_{M+2}+1-\frac{6}{M+2}\right)+\frac{1}{2},\\D_N&=(N+1)M(M-1)/(M+2)(M+1),\\E_N&=\frac{1}{6}(N+1)M(M-1)/(M+2),\\L_N&=4(N+1)/(M+2)(M+1),\\x_N&=(N+1)/(M+2)-\frac{1}{2},\quad 对于 N>M。\end{aligned} \tag{25}$$

上面的讨论说明，通过使用以前仅仅应用于较简单情况的那些技术，有可能对相当复杂的程序的平均运行时间进行精确的分析。

公式（24）和（25）可以用来确定在一台具体的计算机上M的“最好”的值。在MIX的情况下，程序Q要求$37A+14B+4C+12D+8E-L+8X+15$个时间单位；对于$N>M$，该值平均为$\frac{1}{3}(38(N+1)H_N+(N+1)f(M))-19$个单位，其中

$$f(M)=4M-38H_{M+2}+43+\frac{84}{M+2}+\frac{48}{(M+2)(M+1)} \tag{26}$$

我们要选择M，使得$f(M)$是一个极小值。在这种情况下

$$f(M)-f(M-1)=4-\frac{38}{M+2}-\frac{84}{(M+2)(M+1)}-\frac{96}{(M+2)(M+1)M}$$

而且要求出M，使得$f(M)-f(M-1)\leqslant 0$，$f(M+1)-f(M)\geqslant 0$；容易求出解$M=9$。当$M=9$时，对很大的N，程序Q的平均运行时间近似为$12.67(N+1)\ln N-1.92N-14.59$。

考虑到程序Q只要很少的存储空间，所以平均说来，它是十分快的。但是，算法Q的最坏的情况是什么呢？有无某些输入，它不能有效地处理呢？对于这个问题，答案是十分使人头痛的：如果原来的文件已经是有次序的，$K_1<K_2<\cdots<K_N$，则每个“分划”操作都几乎是无用的，因为它仅使子文件的大小减少一个元素！所以这种情况（它应是所有情况中最易于排序的）使得快速排序根本不快；它的运行时间同N^2，而不是同$N\log_2 N$成比例（见习题25）。和已经见到的其它排序方法不同，算法Q偏爱一个无次序的文件！

霍尔在他原来的论文中曾建议用两种方法来弥补这种情况，其原理是选择更好的比较键K（它是支配这个分划的）。他的建议之一是在Q2的最后部分中在l和r之间选择一个随机整数q；可以在该步中把指令“$K\leftarrow K_l$”变成为

$$K\leftarrow K_q,\quad R\leftarrow R_q,\quad R_q\leftarrow R_l \tag{27}$$

按照等式（25），这样的随机整数平均说来仅仅需要计算$2(N+1)/(M+2)-1$次，所以增加的运行时间并不要紧；而且随机选择给出了避免出现最坏情况的好保证。

霍尔的第二个建议是考察文件的小样品，并选择这个样品的一个中值。这个方法已为R. C. 辛格尔顿[*CACM* 12(1969)，185-187）所遵循，他建议命K_q是三个值

$$K_l,\quad K\lfloor(l+r)/2\rfloor,\quad K_r \tag{28}$$

的中值。辛格尔顿过程把比较的次数从$2N\ln N$减少到大约$\frac{12}{7}N\ln N$（见习题29）。在这种情况下可以证明B_N渐近于$C_N/5$，而不是$C_N/6$，所以这个中值方法稍微增加了花费在传送数据中的时间数量；因此总运行时间大约减少百分之八（详细的分析见习题56）。最坏的情况仍然是N^2阶的，但是这样缓慢的行为很少出现。

W. D. 弗雷泽（W. D. Frazer）和A. C. 麦凯勒（A. C. Mckellar）[*JACM* 17(1970)，496-507]已经建议取一个更大的由2^k-1个记录组成的样品，其中把k选择成使$2^k\approx N/\ln N$。这个样品可以用通常的快速排序方法进行排序，然后通过对这个文件进行k次扫

描把样品插入到剩下的记录当中（把它划分成 2^k 个子文件，以样品的元素为界）。最后对这些子文件排序。当N在一个实用的范围内时，这样一个“样品排序”过程所需要的平均比较次数，大体等同于在辛格尔顿中值方法中的次数，但是当$N\to\infty$时，它减少到渐近值$N\log_2 N$。

基数交换 现在讨论另一种方法，它十分不同于以前所看到的任何排序方案；它利用键的二进制表示，所以仅仅适用于二进制计算机。这个方案不是比较两个键，而是检查这些键个别的二进位，看看它们是 0 还是 1。在其它方面，它有交换排序的特性，而且事实上，它颇类似于快速排序。由于它依赖于基数 2 的表示，故我们称它“基数交换排序”。这个算法可粗略描述如下：

1）按键的最高二进位对序列排序，使得有前导 0 的所有键都出现在有前导 1 的所有键之前。这个排序首先寻找有前导 1 的最左边的键 K_i，以及有前导 0 的最右边的键 K_j；然后交换 R_i 和 R_j 并重复这个过程直到 $i>j$。

2）设 F_0 是具有前导 0 的诸元素，F_1 是其余的元素。对 F_0 应用基数交换排序方法（现在从左边第二位开始而不是从最高位开始），直到 F_0 整个地被排序为止；然后对 F_1 同样这样做。

例如，表 3 示出了对于我们的 16 个随机数如何进行基数交换排序，这些数已被转换成为八进制记号。表中的第一阶段示出初始的输入，在对第一位进行交换之后，我们到达阶段 2。阶段 2 按第 2 位对头一组排序，而阶段 3 按第 3 位进行工作（读者可用心算把八进记号转换成 10 位二进制数）。当按第 4 位进行排序后到达阶段 5 时，发现剩下的每一组只有一个元素，所以文件的这部分已不必进一步考察。记号“^{4}(0232 0252)”指的是下一步要按左起第 4 位对子文件 0232 0252 进行排序。对本例来说，当按第 4 位进行排序时，没有进展；需要进到第 5 位，才能把这些条款区分开来。

表 3 中所示的整个排序过程花费 22 个阶段，稍多于快速排序的相应的数目（表 2）。类似地，按位检查的数目为 82 次，是相当高的；但将看到，假定这些键是一致分布的，则对于很大的N来说，按位检查的数目实际上小于快速排序所做的比较数目。表 3 中的交换总数是 17 次，它是完全适度的。注意，尽管被排序的是 10 位数，但这里按位检查决没有超过第七位。

如同在快速排序中那样，可以使用一个堆栈来记住正在等候的子文件的“边界行信息”。一种方便的办法是不去首先对最小的子文件排序，而是简单地从左到右进行排序，因为在这种情况下，栈的大小决不能超过正被排序的键的位数。在下列算法中，栈项目（r，b）用来表示将被按第 b 位进行排序的一个子文件的右边界；左边界不必记录在这个栈中，由于这一过程的自左到右的本性，它是隐含的。

算法 R（基数交换排序） 适当地重新排列记录 $R_1,\cdots,R_N$；在排序完成之后，它们的键将是有序的，$K_1\leqslant\cdots\leqslant K_N$。假定每个键都是一个$m$位的二进数，例如 $(a_1a_2\cdots a_m)_2$；第 i 个最高位 a_i，称为这个键的“位 i”。需要一个至多能存下$m-1$个项目的辅助栈作临时存储。本算法实质上遵循上文中所描述的基数交换分划过程；对它的效率可以有若干改进，如同下文和习题所描述的那样。

表 3　基数-交换排序

阶段																	l	r	b	枝
1	[1]【0767	0127	1000	0075	1614	0252	1601	0423	1215	0652	0232	0775	1144	1245	1375	1277】	1	16	1	—
2	[2]【0767	0127	0775	0075	0232	0252	0652	0423】	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	1	8	2	(16,2)
3	[3]【0252	0127	0232	0075】	[3]〔0775	0767	0652	0423〕	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	1	4	3	(8,3)(16,2)
4	[4]【0075	0127】	[4]〔0232	0252〕	[2]〔0775	0767	0652	0423〕	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	1	2	4	(4,4)(8,3)(16,2)
5	0075	0127	[4]【0232	0252】	[5]〔0775	0767	0652	0423〕	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	3	4	4	(8,3)(16,2)
6	0075	0127	[5]【0232	0252】	[6]〔0775	0767	0652	0423〕	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	3	4	5	(8,3(16,2)
7	0075	0127	0232	0252	[7]【0775	0767	0652	0423】	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	5	8	3	(16,2)
8	0075	0127	0232	0252	0423	[4]【0767	0652	0775】	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	5	8	4	(16,2)
9	0075	0127	0232	0252	0423	0652	[5]【0767	0775】	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	7	8	5	(16,2)
10	0075	0127	0232	0252	0423	0652	[6]【0767	0775】	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	7	8	6	(16.2)
11	0075	0127	0232	0252	0423	0652	[7]【0767	0775】	[2]〔1215	1601	1614	1000	1144	1245	1375	1277〕	7	8	7	(16,2)
12	0075	0127	0232	0252	0423	0652	0767	0775	[2]【1215	1601	1614	1000	1144	1245	1375	1277】	9	16	2	—
13	0075	0127	0232	0252	0423	0652	0767	0775	[3]【1215	1277	1375	1000	1144	1245】	[3]〔1614	1601〕	9	14	3	(16.3)
14	0075	0127	0232	0252	0423	0652	0767	0775	[4]【1144	1000】	[4]〔1375	1277	1215	1245〕	[3]〔1614	1601〕	9	10	4	(14.4)(16.3)
15	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	[4]【1375	1277	1215	1245】	[3]〔1614	1601〕	11	14	4	(16.3)
16	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	[6]【1245	1277	1215】	[5]〔1375〕	[3]〔1614	1601〕	11	13	5	(14,5)(16.3)
17	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	[6]【1277	1245】	[6]〔1375〕	[3]〔1614	1601〕	12	13	6	(14.5)(16.3)
18	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[3]【1614	1601】	15	16	3	—
19	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[4]【1614	1601】	15	16	4	—
20	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[5]【1614	1601】	15	16	5	—
21	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[6]【1614	1601】	15	16	6	—
22	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	[7]【1614	1601】	15	16	7	—
23	0075	0127	0232	0252	0423	0652	0767	0775	1000	1144	1215	1245	1277	1375	1614	1601	17	—	—	—

R1.〔初始化〕 置栈为空，并置 $l \leftarrow 1$， $r \leftarrow N$， $b \leftarrow 1$。

R2.〔开始新阶段〕（现在希望按位 b 对子文件 $R_l \leqslant \cdots \leqslant R_r$ 排序；由于这个算法的本性，我们有 $l \leqslant r$。）如果 $l = r$，则转到步骤 R10(由于已有一个单字文件被排好序)。否则置 $i \leftarrow l$， $j \leftarrow r$。

R3.〔检查 K_i 中的 1〕 考察 K_i 的位 b，如果它是 1，则转到步骤 R6。

R4.〔i 增值〕 i 增加 1，如果 $i \leqslant j$，则返回步骤 R3；否则，转到步骤 R8。

R5.〔检查 K_{j+1} 中的 0〕 考察 K_{j+1} 的位 b，如果它为 0，则转到步骤 R7。

R6.〔j 减值〕 j 减 1。如果 $i \leqslant j$，则转到步骤 R5；否则，转到步骤 R8。

R7.〔交换 R_i，R_{j+1}〕 交换记录 $R_i \leftrightarrow R_{j+1}$；然后转到步骤 R4。

R8.〔检查特殊情况〕 （这时已经完成了一个分划阶段；$i = j + 1$，键 $K_l, \cdots, K_j$ 的位 b 是 0，而键 K_i，…，K_r 的位 b 是 1。）b 增加 1。如果 $b > m$，则转到步骤 R10，其中 m 是在这些键中总的位数。（在这样一种情况下，子文件 $R_l \cdots R_r$ 已被排好序。如果在文件中不可能出现相等的键，则不必进行这个检查。）否则，如果 $j < l$ 或 $j = r$，则返回步骤 R2（所有考察的位都为 1 或都为 0）。否则，如果 $j = l$，则 j 增加 1 并且转到步骤 R2（仅有一个位为 0）。

R9.〔压入栈〕 在栈顶插入项目（r， b）；然后置 $r \leftarrow j$ 并转到步骤 R2。

R10.〔弹出栈〕 如果栈是空的，则已经完成了排序；否则置 $l \leftarrow r + 1$，取出栈顶的项目（r'， b'），置 $r \leftarrow r'$， $b \leftarrow b'$，并返回步骤 R2。

程序 R（基数交换排序） 下列 MIX 代码基本上使用和程序Q同样的约定。我们有 rI1≡ $l - r$， rI2≡ r， rI3≡ i， rI4≡ j， rI5≡ $m - b$，rI6≡栈的大小，但对于某些（下边指出的）指令保持 rI3≡ $i - j$ 或 rI4≡ $j - i$ 是方便的。由于基数交换的二进制属性，这个程序使用操作 SRB（二进制右移 AX）、JAE（A 为偶数时转移）以及 JAO（A 为奇数时转移），这些操作已在 4.5.2 节中定义。假定 $N \geqslant 2$。

01	START	ENT6	0	1	R1. 初始化，置栈为空
02		ENT1	1 − N	1	$l \leftarrow 1$
03		ENT2	N	1	$r \leftarrow N$
04		ENT5	M − 1	1	$b \leftarrow 1$
05		JMP	1F	1	转 R2(省略检验 $l = r$)
06	9H	INC6	1	S	R9. 压入栈〔rI4 = $l - j$〕
07		ST2	STACK,6(A)	S	
08		ST5	STACK,6(B)	S	(r， b)⇒栈
09		ENN1	0,4	S	rI1 ← $l - j$
10		ENT2	−1,3	S	$r \leftarrow j$
11	1H	ENT3	0,1	A	R2. 开始新阶段〔rI3 = $i - j$〕
12		ENT4	0,2	A	$i \leftarrow l$， $j \leftarrow r$〔rI3 = $i - j$〕
13	3H	INC3	0,4	C'	R3. 检查 K_i 是否为 1
14		LDA	INPUT,3	C'	
15		SRB	0,5	C'	rA的个位 ← K_i 的位 b
16		JAE	4F	C'	如果它为 0，则转到 R4

17	6H	DEC4	1,3	$C''+X$	R6. j 减值，$j \leftarrow j-1$，〔rI4 $=j-i$〕
18		J4N	8F	$C''+X$	如果 $j<i$ 则转到R8，〔rI4 $=j-i$〕
19	5H	INC4	0,3	C''	R5. 检查K_{j+1}是否为0
20		LDA	INPUT+1,4	C''	
21		SRB	0,5	C''	rA的个位$\leftarrow K_{j+1}$的位 b
22		JAO	6B	C''	如果它为1，则转到R6
23	7H	LDA	INPUT+1,4	B	R7. 交换R_i，R_{i+1}
24		LDX	INPUT,3	B	
25		STX	INPUT+1,4	B	
26		STA	INPUT,3	B	
27	4H	DEC3	−1,4	$C'-X$	R4. i 增值，$i \leftarrow i+1$，〔rI3 $=i-j$〕
28		J3NP	3B	$C'-X$	如果 $i \leqslant j$ 则转R3，〔rI3 $=i-j$〕
29		INC3	0,4	$A-X$	rI3$\leftarrow i$
30	8H	J5Z	OF	A	R8. 检验特殊情况〔rI4 = 未知〕
31		DEC5	1	$A-G$	如果 $b=m$ 则转到R10，否则 $b \leftarrow b-1$
32		ENT4	−1,3	$A-G$	rI4$\leftarrow j$
33		DEC4	0,2	$A-G$	rI4$\leftarrow j-r$
34		J4Z	1B	$A-G$	如果 $j=r$，则转到R2
35		DEC4	0,1	$A-G-R$	rI4$\leftarrow j-l$
36		J4N	1B	$A-G-R$	如果 $j<l$，则转到R2
37		J4NZ	9B	$A-G-L-R$	如果 $j \neq l$，则转到R9
38		INC1	1	K	$l \leftarrow l+1$
39	2H	J1NZ	1B	$K+S$	$l \neq r$，则转移
40	OH	ENT1	1,2	$S+1$	R10. 弹出栈
41		LD2	STACK,6(A)	$S+1$	
42		DEC1	0,2	$S+1$	
43		LD5	STACK,6(B)	$S+1$	栈⇒(r，b)
44		DEC6	1	$S+1$	
45		J6NN	2B	$S+1$	如果栈非空，则转到R2

这个基数交换程序的运行时间依赖于：

A＝遇到的 $l<r$ 的阶段数；

B＝交换数；

$C=C'+C''$＝位检查的数目；

G＝在步骤R8中 $b>m$ 的次数；

K＝在步骤R8中 $b \leqslant m$，$j=l$ 的次数；

L＝在步骤R8中 $b \leqslant m$，$j<l$ 的次数；

R＝在步骤 R8 中 $b \leqslant m$，$j = r$ 的次数；

S＝元素压入栈的次数；

X＝在步骤 R6 中 $j < i$ 的次数。 (29)

由柯希霍夫定律，$S = A - G - K - L - R$；所以总运行时间为 $27A + 8B + 8C - 23G - 14K - 17L - 19R - X + 13$ 个单位。如同习题 34 所示，以一个较复杂的程序为代价，位检查循环可以进行得更快些。每当 $r - l$ 充分小时，可以象在算法Q中那样，通过使用直接插入来提高基数交换的速度；但我们将不详细讲述这些精化措施。

为了分析基数交换的运行时间，可以使用两种类型的输入数据：

(i) 假定 $N = 2^m$ 并且待排序的键就是随机顺序下的整数 $0, 1, 2, \cdots, 2^m - 1$；或

(ii) 假定 $m = \infty$（无限地精确）而且有待排序的键是在 $[0, 1)$ 中独立地一致分布的实数。

情况(i)的分析相对来讲是容易的，所以已留作读者的习题（见习题 35）。情况(ii)相对来讲是困难的，所以它也留作一个习题。下表示出了对于这些分析结果的粗略近似：

量	情况（i）	情况（ii）
A	N	αN
B	$\frac{1}{4} N \log_2 N$	$\frac{1}{4} N \log_2 N$
C	$N \log_2 N$	$N \log_2 N$
G	$\frac{1}{2} N$	0
K	0	$\frac{1}{2} N$
L	0	$\frac{1}{2}(\alpha - 1)N$
R	0	$\frac{1}{2}(\alpha - 1)N$
S	$\frac{1}{2} N$	$\frac{1}{2} N$
X	$\frac{1}{2} N$	$\frac{1}{4}(\alpha + 1)N$

(30)

这里 $\alpha = 1/(\ln 2) = 1.4427$。注意尽管情况（ii）花费的阶段要多 44%，但是交换、位检查和栈访问的平均次数，对于两种情况说来实质上是相同的。平均说来，我们的 MIX 程序，在情况（ii）下为对 N 个项目排序花费了近 $14.4 N \ln N$ 个时间单位，而且使用习题 34 的建议还可以减少到大约 $11.5 N \ln N$；对于程序Q对应的数字是 $12.7 N \ln N$，使用辛格尔顿三个取中的建议，它可以减少到大约 $11.7 N \ln N$。

于是，当对一致分布的数据排序时，平均说来，基数交换排序和快速排序花费大约同样长的时间。在某些机器上，它实际上比快速排序还稍微更快些。习题 53 指出对于非一致分布的数据这个过程减慢到什么程度。重要的是要注意，整个分析是以诸键都不相同这个假定为根据的；当出现相等的键时，如上定义的基数交换就不是特别有效的。因为在 b 变成 $> m$ 之前，它白白浪费一些阶段来分开相等的键的集合。弥补这个缺陷的一个似乎有理

的方法，在习题 40 的答案中提出。

基数交换和快速排序两者实质上都是以分划的思想为基础的。诸记录被交换，直到这文件被分成两个部分为止：一个左边的子文件，其中所有的键都$\leqslant K$（对某个K），以及一个右边的子文件，其中所有的键都$\geqslant K$。快速排序把这个文件中的一个实际的键选为K，而基数交换则实质上以二进表示为基础选择一个人为的键K。从历史上看，基数交换是由 P.希尔德布兰特（P.Hildebrandt）、H.伊斯比兹（H.Isbitz）、H.赖辛（H.Rising）以及 J.施瓦茨（J.Schwartz）〔JACM 6 (1959), 156-163〕发现的，大约比快速排序早一年。其它的分划方案也是可能的；例如，当已知所有的键位于 u 和 v 之间时，约翰·麦卡锡建议置$K\approx\frac{1}{2}(u+v)$。王义孝已经提议把诸如（28）的三个键值的均值用作分划的阈值；他证明了对一致分布的随机数据排序所需要的比较次数近似为 $1.082\,n\log_2 n$。

M.H.范·埃姆登(M.H.van Emden)提出了另外一种分划策略〔*CACM* 13 (1970), 563-567〕：他不是预先选择K，而是随着分划的进行，记住$K'=\max(K_i,\cdots,K_l)$和$K''=\min(K_j,\cdots,K_r)$，"学会"如何选一个好的K。我们可以令 i 增值，直到遇到一个大于K'的键为止，然后令 j 减值，直到遇到一个小于K''的键为止，然后交换和/或调整K'和K''。对于这个"区间交换排序"方法的经验测试指出，它需要进行大约 $1.64N\ln N=1.14N\log_2 N$ 次比较；这是本书所讨论的唯一的对于其行为还没有找到适当理论说明的排序方法。

把基数交换推广到大于 2 的基数问题，在 5.2.5 节中讨论。

***渐近方法** 交换排序算法的分析导致了某些特别有教益的数学问题，它使我们更多地学会如何研究诸函数的渐近行为。例如，在对气泡排序的分析中（等式 9）遇到过函数

$$W_n=\frac{1}{n!}\sum_{0\leqslant r<s\leqslant n}s!\,r^{n-s} \tag{31}$$

它的渐近值是什么呢？

我们可以照搬对对合数等式 5.1.4-41 的研究方式；读者将发现在进一步阅读之前复习一下 5.1.4 节末尾的讨论是有帮助的。

检查（31）可知，$s=n$ 的贡献大于 $s=n-1$，等等；这提示以 $n-s$ 代替 s。事实上，立即发现，使用替换 $t=n-s+1$，$m=n+1$ 把（31）变为

$$\frac{1}{m}W_{m-1}=\frac{1}{m!}\sum_{1\leqslant t<m}(m-t)!\sum_{0\leqslant r<m-t}r^{t-1} \tag{32}$$

是最方便的。内层的求和可由欧拉求和公式得到熟知的渐近级数

$$\begin{aligned}\sum_{0\leqslant r<N}r^{t-1}&=\frac{N^t}{t}-\frac{1}{2}(N^{t-1}-\delta_{t1})+\frac{B_2}{2!}(t-1)(N^{t-2}-\delta_{t2})+\cdots\\&=\frac{1}{t}\sum_{0\leqslant j\leqslant k}\binom{t}{j}B_j(N^{t-j}-\delta_{tj})+O(N^{t-k})\end{aligned} \tag{33}$$

（见习题 1.2.11.2-4），因此问题归结为研究形如

$$\frac{1}{m!}\sum_{1\leqslant t<m}(m-t)!(m-t)^t t^k \qquad K\geqslant -1 \tag{34}$$

的和。如同在 5.1.4 节那样，可以证明每当 t 大于 $m^{1/2+\epsilon}$ 时，这个求和数的值是可以忽略的 $O(\exp(-n^5))$，因此可以置 $t=0(m^{1/2+\epsilon})$，而且用斯特林近似代替这些阶乘：

$$\frac{(m-t)!(m-t)^t}{m!}=\sqrt{1-\frac{t}{m}}\exp\left(\frac{t}{12m^2}-\left(\frac{t^2}{2m}+\frac{t^3}{3m^2}+\frac{t^4}{4m^3}+\frac{t^5}{5m^4}\right)+O(m^{-2+6\epsilon})\right)$$

我们因此对

$$r_k(m)=\sum_{1\leqslant t<m}e^{-t^2/2m}t^k,\quad k\geqslant -1 \tag{35}$$

的渐近值感兴趣。（这个和也可以扩展到整个 $1\leqslant t<\infty$ 的范围而不会改变它的渐近值，因为如上所述 $t>m^{1/2+\epsilon}$ 的那些值是可以忽略的。）

设 $g_k(x)=x^k e^{-x^2}$ 和 $f_k(x)=g_k(x/\sqrt{2m})$。当 $k\geqslant 0$ 时，欧拉求和公式告诉我们

$$\sum_{0\leqslant t<m}f_k(t)=\int_0^m f_k(x)dx+\sum_{1\leqslant j\leqslant p}\frac{B_j}{j!}(f_k^{(j-1)}(m)-f_k^{(j-1)}(0))+R_p,$$

$$R_p=\frac{(-1)^{p+1}}{p!}\int_0^m B_p(\{x\})f_k^{(p)}(x)dx=\left(\frac{1}{\sqrt{2m}}\right)^p O\left(\int_0^m|g_k^{(p)}(y)|dy\right)$$

$$=O(m^{-p/2}); \tag{36}$$

因此每当 $k\geqslant 0$ 时，通过实质上和以前使用的同样的思想，就能得到一个 $r_k(m)$ 的渐近级数。但当 $k=-1$ 时，这个方法就失灵了，因为 $f_{-1}(0)$ 无定义；我们不能仅仅从 1 到 m 求和，因为当下限为 1 时，余数不给出越来越小的 m 的乘方（这是问题的症结，而且读者在进一步阅读之前应确保自己了解这个问题）。

为了摆脱这个困境，可以定义 $g_{-1}(x)=(e^{-x^2}-1)/x$，$f_{-1}(x)=g_{-1}(x/\sqrt{2m})$；然后可以以简单的方式从 $\Sigma_{0\leqslant t<m}f_{-1}(t)$ 得到 $f_{-1}(0)=0$ 和 $r_{-1}(m)$。等式（36）现在对 $k=-1$ 成立，而且剩下的积分是"熟知的"，由习题 43

$$\frac{2}{\sqrt{2m}}\int_0^m f_{-1}(x)dx=2\int_0^m\frac{e^{-x^2/2m}-1}{x}dx=\int_0^{m/2}\frac{e^{-y}-1}{y}dy$$

$$=\int_0^1\frac{e^{-y}-1}{y}dy+\int_1^{m/2}\frac{e^{-y}}{y}dy-\ln(m/2)$$

$$=-r-\ln(m/2)+O(e^{-m/2})$$

现在已经有了足够的事实和公式来作出答案，如习题 44 所示。

$$W_n=\frac{1}{2}m\ln m+\frac{1}{2}(r+\ln 2)m-\frac{2}{3}\sqrt{2\pi m}+\frac{31}{36}+O(n^{-1/2}),\quad m=n+1 \tag{37}$$

这就完成了对于气泡排序的分析。

对于基数交换排序的分析，需要知道当 $n\to\infty$ 时有限和

$$U_n=\sum_{k\geqslant 2}\binom{n}{k}(-1)^k\frac{1}{2^{k-1}-1} \tag{38}$$

的渐近值。这个问题要比至今已经遇到的任何其它渐近问题更难些；幂级数展开，欧拉求和公式等等的初等方法，证明是不适用的。N. G. 德·布鲁因（N. G. de Bruijn）已经提出了下列推导。

为了摆脱在（38）中大因子 $\binom{n}{k}(-1)^k$ 的抵消作用，我们把和重新写成一个无穷级数

$$U_n=\sum_{k\geqslant 2}\binom{n}{k}(-1)^k\sum_{j\geqslant 1}\left(\frac{1}{2^{k-1}}\right)^j=\sum_{j\geqslant 1}(2^j(1-2^{-j})^n-2^j+n) \tag{39}$$

如果置 $x=n/2^j$，求和的数是

$$2^j(1-2^{-j})^n-2^j+n=\frac{n}{x}\left(\left(1-\frac{x}{n}\right)^n-1+x\right)$$

当 $x\leqslant n^\epsilon$ 时，有

$$\left(1-\frac{x}{n}\right)^n=\exp\left(n\ln\left(1-\frac{x}{n}\right)\right)=\exp(-x+x^2O(n^{-1})) \tag{40}$$

而这提示以

$$T_n=\sum_{j\geqslant 1}(2^je^{-n/2^j}-2^j+n) \tag{41}$$

逼近（39）。为了论证这个逼近，有 $U_n-T_n=X_n+Y_n$，其中

$$\begin{aligned}
x_n&=\sum_{\substack{j\geqslant 1\\ 2^j<n^{1-\epsilon}}}(2^j(1-2)^{-j})^n-2^je^{-n/2^j}) && \text{〔即 } x>n^\epsilon \text{ 的诸项〕}\\
&=\sum_{\substack{j\geqslant 1\\ 2^j<n^{1-\epsilon}}}O(ne^{-n/2^j}) && \text{〔因为 } 0<1-2^{-j}<e^{-2^{-j}}\text{〕}\\
&=O(n\log_2 ne^{-n^\epsilon}) && \text{〔因为有 } O(\log_2 n) \text{ 项〕}
\end{aligned}$$

以及

$$\begin{aligned}
Y_n&=\sum_{\substack{j\geqslant 1\\ 2^j\geqslant n^{1-\epsilon}}}(2^j(1-2^{-j})^n-2^je^{-n/2^j}) && \text{〔} x\leqslant n^\epsilon \text{ 的诸项〕}\\
&=\sum_{\substack{j\geqslant 1\\ 2^j\geqslant n^{1-\epsilon}}}\left(e^{-n/2^j}\left(\frac{n}{2^j}\right)O(1)\right) && \text{〔由（40）〕}
\end{aligned}$$

以下的讨论将揭示，后一个和是 $O(1)$；因此 $U_n-T_n=O(1)$（见习题 47）。

至今还未曾应用实际上不同于以前使用过的任何技术；但是 T_n 的研究需要一种以复变理论的简单原理为基础的新思想。如果 x 是任何正数，则有

$$e^{-x}=\frac{1}{2\pi i}\int_{1/2-i\infty}^{1/2+i\infty}\Gamma(z)x^{-z}dz=\frac{1}{2\pi}\int_{-\infty}^{\infty}\Gamma\left(\frac{1}{2}+it\right)x^{-(1/2+it)}dt \tag{42}$$

为了证明这个恒等式，考虑图 20(a) 中所示的积分路径，其中 N、N' 和 M 是很大的。沿着这条回路的积分值是内部残数的和，即

$$\sum_{0\leqslant k<M}x^{-k}\lim_{z\to -k}(z+k)\Gamma(z)=\sum_{0\leqslant k<M}x^{-k}\frac{(-1)^k}{k!}$$

在顶线上的积分是$O\left(\int_{-\infty}^{1/2}|\Gamma(t+iN)|x^{-t}dt\right)$，而且我们有熟知的界限

$$\Gamma(t+iN)=O(|t+iN|^{t-1/2}e^{-\pi N/2}) \qquad 当N\to\infty时$$

〔关于伽玛函数的性质，例如可见A. Erdèlyi et al., Higher Transcendental Functions 1 (New York: McGraw-Hill, 1953), Chapter 1.〕因此顶线积分是完全可以忽略的，$O\left(e^{-\pi N/2}\int_{-\infty}^{1/2}(N/xe)^t dt\right)$。底线积分也有一种类似的无关痛痒的行为。对于沿着左边线的积分，利用下列事实

$$\Gamma\left(-\frac{1}{2}+it-M\right)=\Gamma\left(-\frac{1}{2}+it\right)\Big/\left(-M+-\frac{1}{2}+it\right)\cdots\left(-1+-\frac{1}{2}+it\right)$$
$$=\Gamma\left(-\frac{1}{2}+it\right)O\left(1/(M-1)!\right)$$

因此左边的积分是$O(x^{M-1/2}/(M-1)!)\int_{-\infty}^{\infty}\left|\Gamma\left(\frac{1}{2}+it\right)\right|dt$。因而当$M$，$N$，$N'\to\infty$时，仅仅右边的积分残存，而这就证明了（42）。事实上，如果以任何正数代替"$\frac{1}{2}$"，则（42）成立。

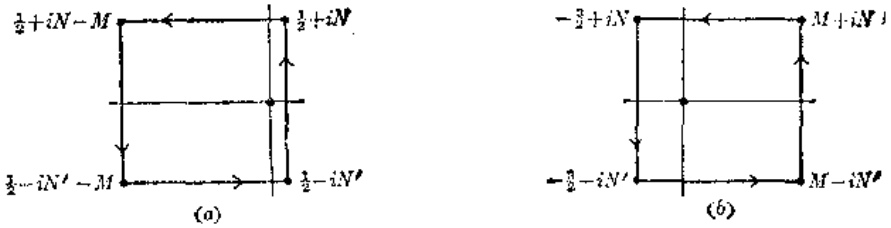


图20　对于伽玛函数恒等式的积分回路

同样的论证可用来推导涉及伽玛函数的许多其它有用的关系。我们可以以z的其它函数代替x^{-z}；或者可以用其它的量代替常数$\frac{1}{2}$。例如

$$\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)x^{-z}dz=e^{-x}-1+x \tag{43}$$

这是在T_n公式（41）中关键性的量：

$$T_n=n\sum_{j\geq 1}\frac{1}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)(n/2^j)^{-1-z}dz \tag{44}$$

这个和可以放置到积分里边，因为收敛性是绝对好的。我们有

$$\sum_{j\leq 1}(n/2^j)^w=n^w\sum_{j\leq 1}(1/2^w)^j=n^w/(2^w-1)，当\ \Re(W)>0\ 因为\ |2^w|=2^{\Re(w)}>1，$$

因此

$$T_n=\frac{n}{2\pi i}\int_{-3/2-i\infty}^{-3/2+i\infty}\frac{\Gamma(z)n^{-1-z}}{2^{-1-z}-1}dz \tag{45}$$

剩下的是计算后边的积分。

这次沿着远向右边扩展的一条路径进行积分，如图20(b)所示。如果$2^{iN}\neq 1$，则顶线积分是$O\left(n^{1/2}e^{-\pi N/2}\int_{-3/2}^{M}|M+iN|^t dt\right)$而当$N$和$N'$比$M$大得多时，底线积分同样是

可以忽略的。右线积分是$O\left(n^{-1-M}\int_{-\infty}^{\infty}|\Gamma(M+it)|dt\right)$。固定$M$并命$N, N'\to\infty$，证明$-T_n/n$是$O(n^{-1-M})$加上在$-3/2<\Re_z(z)<M$范围内的残数之和。命$M\to\infty$；$\Gamma(z)$在$z=-1$和0处有单极点，$n^{-1-z}$没有极点，而且当$z=-1+2\pi ik/\ln 2$时$1/(2^{-1-z}-1)$有单极点。

在$z=-1$处的双重极点是最难处理的。可以使用熟知的关系

$$\Gamma(z+1)=\exp(-\gamma z+\zeta(2)z^2/2-\zeta(3)z^3/3+\zeta(4)z^4/4-\cdots)$$

其中$\zeta(s)=1^{-s}+2^{-s}+3^{-s}+\cdots=H_\infty^{(s)}$，来导出当$w=z+1$很小时的下列展开式：

$$\Gamma(z)=\frac{\Gamma(w+1)}{W(w-1)}=-w^{-1}+(\gamma-1)+O(w)$$

$$n^{-1-z}=1-(\ln n)w+O(w^2)$$

$$1/(2^{-1-z}-1)=-W^{-1}/\ln 2-\frac{1}{2}+O(w)$$

在$z=-1$处的残数是这三个公式的乘积中W^{-1}的系数，即$\frac{1}{2}-(\ln n+\gamma-1)/\ln 2$。加上其它残数即得公式

$$T_n/n=-\frac{\ln n+\gamma-1}{\ln 2}-\frac{1}{2}+f(n)+\frac{2}{n}+O(n^{-M}) \tag{46}$$

其中，对任意大的n，$f(n)$是颇奇怪的函数

$$f(n)=\frac{2}{\ln 2}\sum_{k\geq 1}\Re(\Gamma(-1-2\pi ik/\ln 2)\exp(2\pi ik\log_2 n)) \tag{47}$$

注意，$f(n)=f(2n)$。$f(n)$的平均值为0，因为每项的平均值为0。（根据4.2.4节中关于浮点数的结果，我们可以假定$(\log_2 n)\bmod 1$是一致分布的。）而且由于$|\Gamma(-1+it)|=|\pi/(t(1+t^2)\sinh\pi t)|^{1/2}$，不难证明

$$|f(n)|<0.000000173$$

于是对于实际应用，可以放心地忽略$f(n)$。然而在理论上，U_n的渐近展开不能没有它：这就是为什么U_n是比较难分析的函数。

总之，已经证明

$$U_n=n\log_2 n+n\left(\frac{\gamma-1}{\ln 2}-\frac{1}{2}+f(n)\right)+O(1) \tag{48}$$

这个伽玛函数方法的其它例子出现在习题51-53和6.3节中。

习题

1. 〔$M20$〕 设$a_1\cdots a_n$是$\{1,\cdots,n\}$的一个排列，并设i和j是使得$i<j$和$a_i>a_j$的下标。令$a_1'\cdots a_n'$是由$a_1\cdots a_n$通过交换a_i和a_j得到的排列。问$a_1'\cdots a_n'$能有比$a_1\cdots a_n$更多的反序吗？

▶2. 〔$M25$〕 (a)对排列3 7 6 9 8 1 4 5 2进行排序的极小交换次数是多少？(b)一般地说，给定$\{1,\cdots,n\}$的任何排列$\pi=a_1\cdots a_n$，设$\mathrm{xch}(\pi)$是把π排序成为递增次序的极小交换次数。试借助于π的"较简单的"特征来表达$\mathrm{xch}(\pi)$（参考习题5.2.1-39）。

3. 〔10〕 气泡排序算法B是一个"稳定"的排序算法吗？

4. 〔$M23$〕 如果在步骤B 4中$t=1$，则实际上可以立即终止算法B。因为随后的

步骤B 2已没有什么有用的事要做。当对一个随机排列排序时，在步骤B 4中将出现 $t=1$ 的概率是多少？

5.〔$M25$〕 设 $b_1b_2, \cdots, b_n$ 是排列 $a_1, a_2, \cdots, a_n$ 的反序表。试证，对于 $0 \leqslant r \leqslant \max(b_1, \cdots, b_n)$，在气泡排序的 r 次扫描之后，BOUND 的值是 $\max\{b_i+i \mid b_i \geqslant r\}-r$。

6.〔$M22$〕 设 $a_1, a_2, \cdots, a_n$ 是 $\{1 \cdots n\}$ 的一个排列，并设 $a'_1 \cdots a'_n$ 是它的逆。试证，对 $a_1 \cdots a_n$ 进行气泡排序的扫描次数是 $1+\max(a'_1-1, a'_2-2, \cdots, a'_n-n)$。

7.〔$M28$〕 计算气泡排序的扫描次数的标准离差，并用 n 和函数 $P(n)$ 来表达它〔参考等式（6）和（7）〕。

8.〔$M24$〕 推导等式（8）。

9.〔$M48$〕 试分析在鸡尾混合排序算法〔shic〕中的扫描次数和比较次数。注意：在习题 5.4.8-9 中可找到部分信息。

10.〔$M26$〕 设 $a_1a_2 \cdots a_n$ 是 $\{1, 2, \cdots, n\}$ 的一个2有序排列。（a）对应的格点通路第 a_i 步的端点坐标是什么（参考图 11）？（b）证明 $a_1:a_2$，$a_3:a_4$，…的比较/交换，对应于按对角线折叠这条通路，如图 18（b）所示。（c）证明当 $d=2m-1$ 时，$a_2:a_2+d$，$a_4:a_4+d$，…的比较/交换，对应于按对角线之下 m 个单位处的一条直线折叠形成的通路，如图 18（c）、（d）和（e）所示。

11.〔$M25$〕 $\{1, 2, \cdots, 16\}$ 的什么排列使由巴切尔算法所完成的交换次数取极大值？

12.〔24〕 假定 MIX 是一台具有操作 AND、SRB 的二进制计算机，写出算法M的一个 MIX 程序。为把表 1 中的 16 个记录排序，你的程序花费多少时间？

13.〔10〕 巴切尔方法是一个“稳定的”排序算法吗？

14.〔$M21$〕 设 $c(N)$ 是通过巴切尔方法对 N 个元素排序所进行的键比较次数；这是步骤M4被执行的次数。（a）证明当 $t \geqslant 1$ 时有 $c(2^t)=2c(2^{t-1})+(t-1)2^{t-1}$。（b）试求作为 T 的一个函数的 $c(2^t)$ 的简单表达式。提示：考虑序列 $x_t=c(2^t)/2^t$。

15.〔$M38$〕 本题的任务是来分析习题 4 的函数 $c(N)$，并求出当 $N=2^{e_1}+2^{e_2}+\cdots+2^{e_r}, e_1>e_2>\cdots>e_r \geqslant 0$ 时关于 $c(N)$ 的一个公式。（a）设 $a(N)=c(N+1)-c(N)$。证明 $a(2n)=a(n)+\lfloor \log_2(2n) \rfloor$，以及 $a(2n+1)=a(n)+1$；因此

$$a(N)=\binom{e_1+1}{2}-r(e_1-1)+(e_1+e_2+\cdots+e_r)$$

（b）设 $x(n)=a(n)-a(\lfloor n/2 \rfloor)$，于是 $a(n)=x(n)+x(\lfloor n/2 \rfloor+x(\lfloor n/4 \rfloor)+\cdots$。设 $y(n)=x(1)+x(2)+\cdots+x(n)$；$z(2n)=y(2n)-a(n)$，$z(2n+1)=y(2n+1)$。证明 $c(N+1)=z(N)+2z(\lfloor N/2 \rfloor)+4z(\lfloor N/4 \rfloor)+\cdots$。（c）证明 $y(N)=N+(\lfloor N/2 \rfloor+1)(e_1-1)-2^{e_1}+2$。（d）现在使 r 保持固定，综合以上所得，并借助指数 e_j 求出关于 $c(N)$ 的一个公式。

16.〔$HM46$〕 求出当应用巴切尔方法于 $N=2^t$ 个不同的按随机次序排列的元素时，所需的平均交换次数的渐近值。

▶17.〔20〕 K_0 和 K_{N+1} 的值就是在（13）中假设的值，在算法Q中何处用了这一事实？

▶18.〔20〕 说明当所有的输入键都相等时，算法Q中的计算如何进行。如果步骤Q3和Q4中的“<”号改成“≤”，将发生什么情况？

19.〔15〕 如果使用一个队（先进先出），而不是使用栈（后进先出），问算法Q是否仍将正确地工作？

20.〔M20〕 作为M和N的函数，在算法Q中最多能有多少元素同时在栈中？

21.〔20〕 说明为什么算法Q的分划阶段要花费（17）中所确定的比较、传送等次数。

22.〔M25〕 命p_{kN}是当应用算法Q于$\{1, 2, \cdots, N\}$的一个随机排列时，(16)中的量A将等于k的概率；并命$A_N(z)=\sum_k p_{kN}z^k$是对应的生成函数。试证对于$N\leqslant M$，$A_N(z)=1$，而对于$N>M$，$A_N(z)=z(\sum_{1\leqslant j\leqslant N}A_{j-1}(z)A_{N-j}(z))/N$。试求确定其它概率分布$B_N(z)$、$C_N(z)$、$D_N(z)$、$E_N(z)$、$L_N(z)$、$X_N(z)$的类似递归关系。

23.〔M24〕 命A_N、B_N、D_N、E_N、L_N、X_N是当对$\{1, 2, \cdots, N\}$的一个随机排列排序时，(16)中对应量的平均值，试求对应于（18）的这些量的递归关系，并且解这些递归关系以得到（25）。

24.〔M21〕 算法Q明显地作了比它所需要的还要多一点的比较。因为我们在步骤Q3中能有$i=j$，而且甚至在Q4中有$i>j$。如果当$i\geqslant j$时，我们避免进行所有的比较，则平均说来，所执行的比较次数c_N是多少？

25.〔M20〕 当输入的键依次是数1，2，…，N时，问在程序Q的计时中量A、B、C、D、E、L和X的精确值是什么（假定$N>M$）？

▶26.〔M21〕 构造一个输入文件，它使程序Q甚至比它在习题25中进行得还要慢（试找出一种真正坏的情况）。

27.〔M28〕（R.塞奇维克）考虑算法Q的最好情况：寻找$\{1, 2, \cdots, 23\}$的一个排列，当$N=23$和$M=3$时，对它排序所花时间最少。

28.〔M26〕 求类似于（20）的递归关系，它被辛格尔顿修改过的算法Q的平均比较次数所满足（选择s作为$\{K_1, K_{\lfloor(N+1)/2\rfloor}, K_N\}$的中值，而不是$s=K_1$）。

29.〔HM40〕 求在辛格尔顿“三值取中”方法中比较次数的渐近值（参考习题28）。

30.〔25〕〔P.沙克尔顿（P.Shackleton）〕 当对多个字的键排序时，许多排序方法随着文件接近它的最后次序而逐渐慢下来，因为相等的和接近相等的键需要检查几个字才能确定适当的字典编辑次序（见习题5-5）。通常实践中出现的文件都包含接近相等的键，所以这个现象对排序时间可能有重大的影响。算法Q有这个困难；算法R则没有，因为它每次仅检查一位（尽管由于其它原因，相等的和接近相等的键可能大大地增加算法R的运行时间）。

试说明怎样推广算法Q以避免这个困难；在一个已知对于所有键其前导的k个字都为常数值的子文件中，仅需检查键的第（$k+1$）个字。

▶31.〔20〕（C.A.R.霍尔） 假设不是对整个文件排序，而只要确定一个给定的n个元素集合的第m个最小者。证明“快速排序”可适合于这一目的，它避免了为进行一次完全排序所需要的许多计算。

32.〔M40〕 利用习题31的方法求出寻找n个元素的第m个最小者所需要进行的平均键比较次数C_{nm}的简单的“封闭形式”的表达式（为简便起见，如同习题24中那样，避

免作对于 $i \geqslant j$ 的一切比较)。通过霍尔方法求 $2m-1$ 个元素中的中值所需要的平均比较次数 $C_{(2m-1)m}$ 的渐近行为如何?

▶33.〔20〕 试设计一个算法，它重新排列一个给定表中的所有数，使得所有负数都在所有非负数之前（这些条款不必完全排序，仅需分开负的和非负的)。你的算法应使用最小可能的交换次数。

34.〔20〕 如何加快基数交换（在步骤 R3 到 R6 中）的位检查循环?

35.〔$M23$〕 分析频率 A、B、C、G、K、L、R、S 和 X 的值，它们是在利用"情况（i）的输入"的基数交换排序中出现的。

36.〔$M27$〕 给定数的序列 $\langle a_n \rangle = a_0, a_1, a_2, \cdots$，通过规则

$$\hat{a}_n = \sum_k \binom{n}{k} (-1)^k a_k$$

定义它的二项式转换 $\langle \hat{a}_n \rangle = \hat{a}_0, \hat{a}_1, \hat{a}_2, \cdots$。（a）证明 $\langle \hat{\hat{a}}_n \rangle = \langle a_n \rangle$。（b）求序列（1）; $\langle n \rangle$ 的二项式转换；对于固定的 m，求 $\left\langle \binom{n}{m} \right\rangle$ 的二项式转换；对于固定的 a，求 $\langle a^n \rangle$ 的二项式转换；对于固定的 a 和 m，求 $\left\langle \binom{n}{m} a^n \right\rangle$ 的二项式转换。（c）已知序列 $\langle x_n \rangle$ 满足关系

$$x_n = a_n + 2^{1-n} \sum_{k \geqslant 2} \binom{n}{k} x_k, \quad 对于\ n \geqslant 2; \quad x_0 = x_1 = a_0 = a_1 = 0$$

证明

$$x_n = \sum_{k \geqslant 2} \binom{n}{k} (-1)^k \frac{2^{k-1} \hat{a}_k}{2^{k-1} - 1} = a_n + \sum_{k \geqslant 2} \binom{n}{k} (-1)^k \frac{\hat{a}_k}{2^{k-1} - 1}$$

37.〔$M28$〕 在习题 36 的意义下，确定所有使得 $\langle \hat{a}_n \rangle = \langle a_n \rangle$ 的序列 $\langle a_n \rangle$。

▶38.〔$M30$〕 当基数交换应用于"情况（ii）的输入"时，求（29）中诸量的平均值 A_N、B_N、C_N、G_N、K_N、R_N 和 X_N。试借助 N 和函数

$$U_n = \sum_{k \geqslant 2} \binom{n}{k} \frac{(-1)^k}{2^{k-1} - 1} \qquad V_n = \sum_{k \geqslant 2} \binom{n}{k} \frac{(-1)^k k}{2^{k-1} - 1} = n(U_n - U_{n-1})$$

表达你的答案〔提示；见习题 36〕。

39.〔20〕 (30) 中所示的结果指出，当应用基数交换于随机输入时，大约花费 1.44 N 个阶段。证明快速排序决不需要多于 N 个阶段；并说明为什么基数交换往往需要这么多。

40.〔21〕 说明怎样修改算法 R，使得它对包含许多相等键的文件排序时，也以相当不错的效率进行工作。

41.〔23〕 阐述范·埃姆登的"区间交换排序"精确描述其算法。

42.〔$M43$〕 分析习题 41 的区间交换算法。

43.〔$HM21$〕 证明 $\int_0^1 y^{-1}(e^{-y} - 1)dy + \int_1^\infty y^{-1}e^{-y}dy = -\gamma$〔提示：考虑 $\lim_{a \to 0^+} y^{a-1}$〕。

44.〔$HM24$〕 如正文中所建议的那样，推导（37）。

45. 〔$HM20$〕 说明当 $x>0$ 时，为什么（43）为真。

46. 〔$HM20$〕 给定正整数 s，$0<a<s$，问 $(1/2\pi i)\int_{a-i\infty}^{a+i\infty}\Gamma(z)n^{s-z}dz/(2^{s-z}-1)$ 的值是多少？

47. 〔$HM20$〕 证明 $\sum_{i\geqslant 1}(n/2^i)e^{-n/2^i}$ 是 n 的一个有界函数。

48. 〔$HM24$〕 试求在习题 38 中定义的量 V_n 的渐近值，利用类似正文研究 V_n 的一个方法，求其诸项直到 $O(1)$ 为止。

49. 〔$HM24$〕 展开 U_n 的渐近公式（47）直到 $O(n^{-1})$。

50. 〔$HM24$〕 当 m 为任何大于 1 的固定数时，求函数

$$U_{mn}=\sum_{k\geqslant 2}\binom{n}{k}(-1)^k\frac{1}{m^{k-1}-1}$$

的渐近值（当 m 是大于 2 的一个整数时，这个量出现于推广基数交换的研究中，以及 6.3 节的“检索结构内存”的查找算法中）。

▶51. 〔$HM28$〕 证明解决渐近问题的伽玛函数可以用来代替欧拉求和公式，以推导 $r_k(m)$ 的渐近展开（参考等式（35））。这给了对所有 k 研究 $r_k(m)$ 的一个统一的方法，而无须依赖于正文中介绍的 $g_k(x)=(e^{-x^2}-1)/x$ 的技巧）。

52. 〔$HM35$〕（N. G. 德·布鲁因）和数

$$S_n=\sum_{t\geqslant 1}\binom{2n}{n+t}d(t)$$

的渐近行为是什么？其中 $d(t)$ 是 t 的因子的个数。（于是 $d(1)=1$，$d(2)=d(3)=2$，$d(4)=3$，$d(5)=2$，等等。这个问题的出现同分析一个树遍历算法有关，见习题 2.3.1-11。）试求 $S_n\Big/\binom{2n}{n}$ 的值，到 $O(n^{-1})$ 的项为止。

53. 〔$HM42$〕 当输入数据由〔0，1〕中的无穷精度二进数组成，且它们诸位中的每一个都独立地以概率 p 等于 1 时，试分析基数交换所执行的位检查和交换的平均次数。（正文中仅仅讨论了 $p=\frac{1}{2}$ 的情况；所使用的方法可以推广到任意的 p。）特别是，考虑 $p=1/\phi=0.61803\cdots$ 的情况。

54. 〔$HM24$〕〔S. O. 里斯（S. O. Rice）〕 证明 U_n 可被写成

$$U_n=(-1)^n\frac{n!}{2\pi i}\oint_C\frac{dz}{z(z-1)\cdots(z-n)}\frac{1}{2^{z-1}-1}$$

其中 C 是包围点 2，3，…，n 的小的封闭曲线。试把 C 变成以原点为中心的任意大的圆，推导出诸收敛级数

$$U_n=n(H_{n-1}-1)/(\ln 2)-\frac{1}{2}n+2+\frac{2}{\ln 2}\sum_{m\geqslant 1}\Re(B(n+1,\ -1+ibm))$$

其中，$b=2\pi/(\ln 2)$，$B(n+1,\ -1+ibm)=\Gamma(n+1)\Gamma(-1+ibm)/\Gamma(n+ibm)$ $=n!/\prod_{0\leqslant k\leqslant n}(k-1+ibm)$。

▶55.〔22〕 说明应该如何修改程序Q，使得分划的元素是三个键（28）的中值。

56.〔44〕 当程序已经修改成如习题55中那样取三个元素的中值时，试分析出现于算法Q的运行时间中诸量的平均行为〔参考习题29〕。

▶57.〔$HM24$〕 试求当 $M=N/\alpha$，且 $N\to\infty$而α保持固定时，出现于多表插入中自右到左的极大值个数（等式5.2.1-11）的渐近值。利用"指数积分"函数$E_1(z)=\int_z^\infty e^{-t}dt/t$，来表达你的答案，展开到$O(N^{-1})$的项为止。

5.2.3 通过选择进行排序

另一个重要的排序技术类型是以重复选择的思想为基础的。最简单的选择方法也许如下：

i）求最小的键；传送对应的记录到输出区域；然后以值"∞"（假定它高于任何实际的键）代替这个键。

ii）重复步骤i）。这次将选出第二个最小的键，因为最小的键已为∞所代替。

iii）继续重复步骤i）直到已经选择了N个记录为止。

注意，这样一个选择方法要求在排序开始之前所有的输入项目均已出现，而且它顺序地逐个产生最后的输出。这实质上正好与插入法相反，在那里输入被顺序地接收，但在完成排序之前并不知道任何最后的输出。

某些计算机（例如，那些具有循环磁波存储的）有一个高速操作的"求最小者"指令。这使得当N不太大时，通过上述方法进行选择排序特别有吸引力。

在上述方法中，每选择一个新记录需要$N-1$次比较，而且它在内存中也需要一个分开的输出区域。有一个显然的方法来改进这种状况，同时避免使用∞：我们可以取这个已选出的值，把它同当前占有它正当位置的记录进行交换，从而把它移到它的正确位置处。然后，在以后的选择中，就不需要再考虑该位置了。由这个思想产生了头一个选择排序算法。

算法S（直接插入排序） 适当重新安排记录R_1，…，R_N；在完成排序后，它们的键将是有序的$K_1\leqslant\cdots\leqslant K_N$。排序是以上边指出的方法为基础的，但改为首先选最大元素，紧接着选择第二个最大的等等，这样做证明是更为方便的。

S1.〔对j进行循环〕 对$j=N$，$N-1$，…2执行步骤S2和S3。

S2.〔找$\max(K_1,\cdots K_j)$〕 查一遍$K_j,K_{j-1},\cdots K_1$，以找出一极大者，设它为K_i。

S3.〔同R_j进行交换〕 交换记录$R_i\longleftrightarrow R_j$。（现在诸记录$R_j$，…，$R_N$都处于它们的最后位置处）。

表1示出了本算法应用于16个示例键的情况；在查找期间（步骤S2）作为极大值被选出的元素，以黑体标出。

表 1　直接选择排序

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	087	512	061	703	170	897	275	653	426	154	509	612	677	765	908
503	087	512	061	703	170	765	275	653	426	154	509	612	677	897	908
503	087	512	061	703	170	677	275	653	426	154	509	612	765	897	908
503	087	512	061	612	170	677	275	653	426	154	509	703	765	897	908
503	087	512	061	612	170	509	275	653	426	154	677	703	765	897	908
…															
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

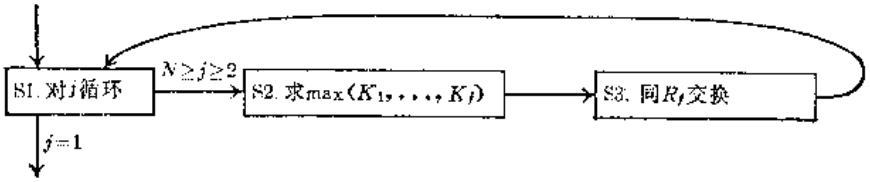


图21　直接选择排序

对应的 MIX 程序是十分简单的：

程序 5.2.3S（直接插入排序）　同本章中以前的程序一样，单元 INPUT＋1 到 INPUT＋N 中的诸记录按一个全字长键就地排序；rA≡当前极大值，rI1≡$j-1$，rI2≡k（查找的下标），rI3≡i，假定$N\geqslant 2$。

01	START	ENT1	N－1	1	S1. 对 j 进行循环，$j\leftarrow N$
02	2H	ENT2	0,1	$N-1$	S2. 找max($K_1,\cdots,K_j$)，$k\leftarrow j-1$
03		ENT3	1,1	$N-1$	$i\leftarrow j$
04		LDA	INPUT,3	$N-1$	rA←K_i
05	8H	CMPA	INPUT,2	A	
06		JGE	*＋3	A	如果$K_i\geqslant K_k$，则转移
07		ENT3	0,2	B	否则，置 $i\leftarrow k$
08		LDA	INPUT,3	B	rA←K_i
09		DEC2	1	A	$k\leftarrow k-1$
10		J2P	8B	A	如果 $k>0$，重复
11		LDX	INPUT＋1,1	$N-1$	S3. 同R_j进行交换
12		STX	INPUT，3	$N-1$	$R_i\leftarrow R_j$
13		STA	INPUT＋1,1	$N-1$	$R_j\leftarrow$rA
14		DEC1	1	$N-1$	
15		J1P	2B	$N-1$	$N\geqslant j\geqslant 2$

这个程序的运行时间依赖于项目数N；比较的次数A；以及“自右至左的极大”数B。容易看出，不论输入键的值为何

$$A=\binom{N}{2}=\frac{1}{2}N(N-1) \tag{1}$$

因此仅仅B是可变的。尽管直接选择很简单，这个量B也是不易精确地分析的。习题 3 到 6 指出

$$B=(\min 0,\ \mathrm{ave}(N+1)H_N-2N,\ \max\lfloor N^2/4\rfloor) \tag{2}$$

在这种情况下，极大值显得特别有趣（B的标准离差尚未确定）。

于是程序S的平均运行时间是$2.5N^2+3(N+1)H_N+3.5N-11$个单位。它仅比直接插入（程序5.2.1S）稍慢些。把算法S同气泡排序（算法5.2.2B）进行比较是有趣的，因为气泡排序可以看作是一个选择算法，它有时一次选择出多于一个元素。由于这一原因，气泡排序通常比直接选择要少做些比较，而且可能显得还优越一些；但事实上，程序5.2.2B比程序S慢两倍多！气泡排序逊色的原因是它进行那么多的交换，而直接选择包含非常少的数据移动。

直接选择的精化 有什么途径来改进用于算法S中的选择方法呢？例如，步骤S2中对于极大值的查找，是否有一个快得多的方法呢？对于后一个问题的回答是否定的！

引理M 任何以比较元素对偶为基础的在n个元素中寻找极大者的算法，必须至少作$n-1$次比较。

证明 如果比较少于$n-1$次，则至少有两个元素，它们决未被发现是小于任何其它元素的。因此，就不会知道这两个元素中哪一个是较大的，从而也就不能确定极大者。

于是，任何寻找最大元素的选择过程必须至少进行$n-1$步；也许所有以n次重复的选择为基础的排序方法，其步骤的阶数都注定为n^2？幸而，引理M仅适用于头一个选择步骤。随后的选择可以利用前已获得的信息。例如，习题8指出，对算法S稍加改变就把平均次数削减了一半。

考虑表1中的16个数；节省重复选择时间的一个方法是把它们看做四个四元组。我们可以从确定每个组的最大元素开始，这些最大元素的键是

512, 908, 653, 765

这四个元素的最大者是908，因而也是整个文件的最大者。为了获得第二个最大者，仅仅需要考察包含908的组中的其它三个元素；{170, 897, 275}中最大者是897，而且

512, 897, 653, 765

的最大者是897。类似地，为了得到第三个最大的元素，先确定{170, 275}中的最大者，然后找

512, 275, 653, 765

的最大者，等等。在头一个之后的每一个选择，至多花费6个附加的比较。一般说来，如果N是一个完全平方，则可以把文件分成为$\sqrt{N}$个$\sqrt{N}$元组；在头一次选择之后，每个选择至多在以前选择了项目的组中花费$\sqrt{N}-1$次比较，加上在"各组尖子"当中的$\sqrt{N}-1$次比较。这一思想被称为二次选择；它的总共执行时间是$O(N\sqrt{N})$，这比阶N^2要好得多。

二次选择首先是由E. H. 弗兰德发表的[*JACM* 3 (1956), 152-154]，他指出同一思想可推广到三次、四次选择等等。例如，三次选择把文件划分成$\sqrt[3]{N}$个大组，每一大组包含$\sqrt[3]{N}$个小组，每个小组包含$\sqrt[3]{N}$个记录；执行时间与$N\sqrt[3]{N}$成比例。如果把这个思想引到它最终的结论，则就得到了弗兰德所谓的以一个二叉树结构为基础的"n次选择"。这个方法的执行时间与$N\log_2 N$成比例；我们称它为树选择。

树选择 借助典型的"淘汰锦标赛"中的对垒，容易理解树选择排序的原理。例如，

考虑图 22 中乒乓球比赛的结果：吉姆打败唐，乔打败杰克，然后在下轮中乔打败吉姆等等。

图 22 示出乔是八个选手中的优胜者，而且为确定这事实须做 8 − 1 = 7 次比赛（即比较）。狄克不一定是第二个最好的选手；被乔打败的任何人，包括头一轮的失败者杰克在内，都可能是第二个最好者。可以通过杰克和吉姆比赛，以及该场比赛的优胜者同狄克比赛，来确定第二个最好的选手；由于我们已经从前几场比赛中记住了这个结构，为找出第二个最好的人只须进行两场附加的比赛。

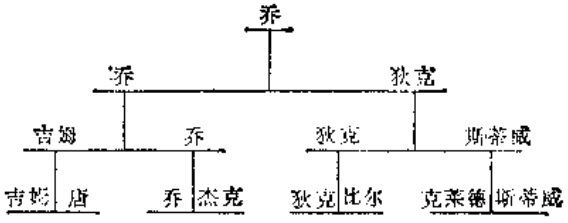


图22 一场乒乓球锦标赛

一般说来，可以在树的根处“输出”选手，然后用极其弱的选手来代替它。替入这个弱的选手意味着原来第二个最好的选手现在将是最好的，所以如果重新计算树上边数层的优胜者，则他将出现于根处。为此目的，树中仅有一条通路必须改变，所以为选择下一个最好的选手，只需要作少于$\lceil \log_2 N \rceil$个进一步的比较。这样一个选择排序的总共时间大约同$N \log_2 N$成比例。

图 23 示出了对 16 个例数进行的树选择排序过程。注意，为了知道在哪里插入下一个“$-\infty$”就必须知道根处的键是从哪里来的。因此树的分枝节点实际上包含一个指针或下标，以确定有关键的位置，而不是键本身。由此推知，我们需要能存放N个输入记录，$N-1$个指针字，以及N个输出记录的存储空间。（当然，如果输出送到磁带或磁盘上，那么就不需要在高速存储器中保留输出记录了）。

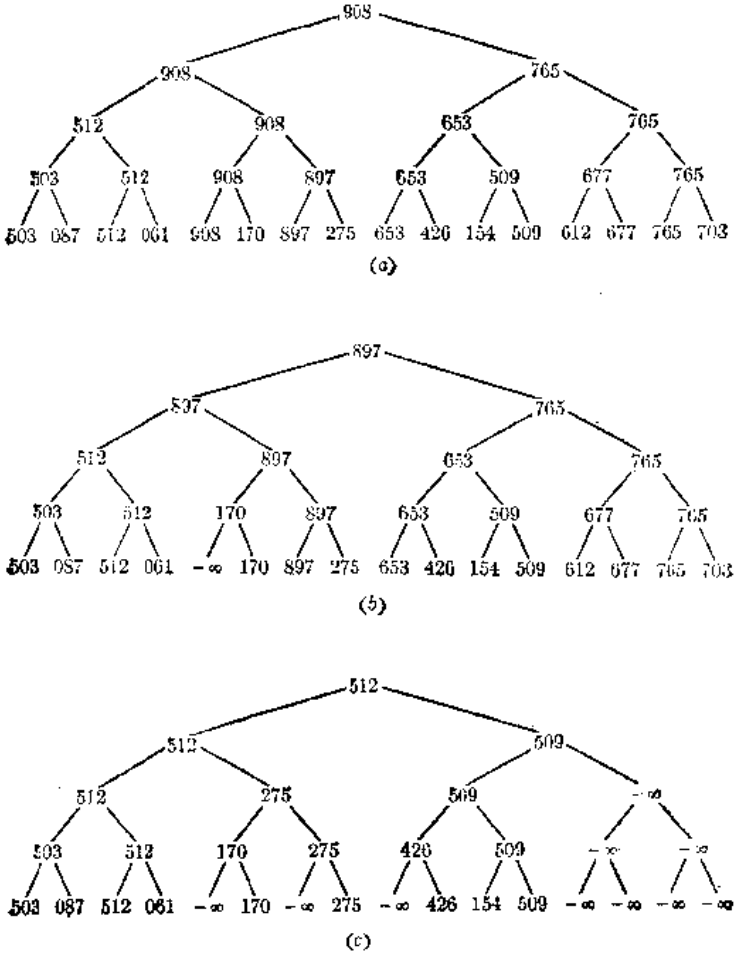


图23 树选择排序的一个例子

（a）初始配置；（b）908以$-\infty$代替，而且第二个最高的元素上移至根处；（c）在输出了908，897，765，703，677，653，612之后的配置。

读者在这里首先要搞明白树选择，然后再往下阅读，这样才能来评价我们将要讨论的一些重大改进。例如，现在应该能容易地做出习题 10。

修改树选择的一个方

法，实质上是由 K. E. 艾弗森〔*A Programming Language*，(Wiley，1962)，223-227〕引进的，通过下列方法“向前看”就可以去掉对指针的需要：当在树的底层中比赛的优胜者被上移时，在底层处立即可以用“$-\infty$”来代替他；而且当一个优胜者从一个分枝上移到另一分枝时，他留下的位置即可由下边两个键中的较大者来占领（此较大者最后总是要占据这个位置的）。重复这个操作直到做不下去为止就把图 23(a)变成图 24。

一旦以这样的方法建起了树，就可以通过一个“由顶向下”的方法来进行排序，以代替图 23 的“由底向上”的方法；我们输出这个根，上移他的最大后裔，上移后者的最大后裔，等等。这个过程开始看来不大象乒乓球锦标赛，而更象团体选拔系统。

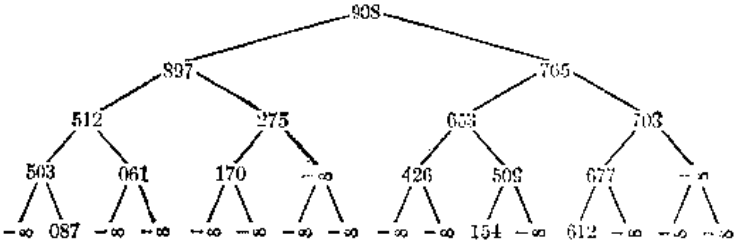


图24　应用于排序的彼得原理，每个人在这个层次中都上升到他失败的层上

读者应能看出，这个由顶向下的方法有一个优点，它可以避免$-\infty$和$-\infty$的多余比较。(由底向上的方法在排序的稍后阶段将发现到处都是$-\infty$，而由顶向下的方法则在每个阶段中存进一个$-\infty$后便停止修改树)。

图 23 和 24 是具有 16 个终端结点的完备二叉树（参考 2.3.4.5 节），而且以如图 25 所示的连续单元来表示这样一株树是方便的。注意，号码为 k 的节点的父亲是节点 $\lfloor k/2 \rfloor$，而它的儿子是节点 $2k$ 和 $2k+1$。这就导出了由顶向下方法的另一个优点，因为由顶向下从节点 k 到节点 $2k$ 和 $2k+1$，比起由底向上从节点 k 到节点 $k \ominus 1$ 和 $\lfloor k/2 \rfloor$（这里依据 k 是偶数或奇数，$k \ominus 1$ 分别表示 $k+1$ 或 $k-1$）往往要更简便些。

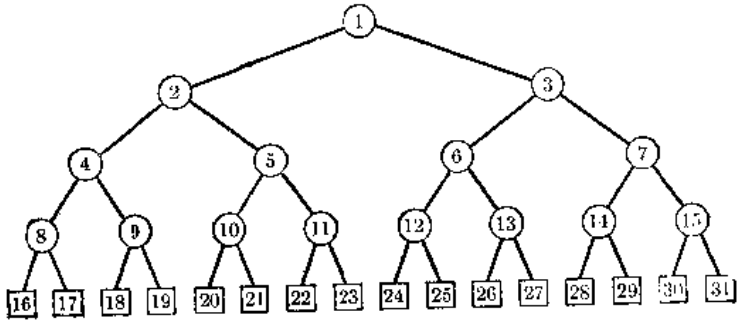


图25　一株完备的二叉树的顺序存储分配

至今树选择的例子都或多或少地假定了 N 是 2 的一个乘方；但是实际上 N 可以是任意的，因为很容易对任何的 N 构造具有 N 个终端结点的完备二叉树。

现在遇到了决定性的问题：能否全然不使用“$-\infty$”来实施由顶向下的方法？如果图 24 的重要信息全存在完备的二叉树单元 1 到 16 中，而不要包含$-\infty$的无用“空穴”，岂不是很好吗？某些研究指出，确有可能达到这一目标，不仅仅是消去$-\infty$，而且有可能对 N 个记录就地排序，而无须使用任何辅助的输出区域。这是一个重要的排序算法，它的发现者 J. W. J. 威廉斯（J. W. J. Williams）为之取名“堆排序”〔*CACM* 7 (1964)，347-348〕。

堆排序　我们称键 K_1，K_2，…，K_N 的一个文件为一个“堆”，如果

$$K_{\lfloor j/2 \rfloor} \geqslant K_j \quad 对于\ 1 \leqslant \lfloor j/2 \rfloor < j \leqslant N \tag{3}$$

例如$K_1 \geqslant K_2$，$K_1 \geqslant K_3$，$K_2 \geqslant K_4$，等等；这恰是在图24中成立的条件，特别是，它意味着最大的键出现在“堆的顶部”

$$K_1 = \max(K_1, K_2, \cdots, K_N) \tag{4}$$

如果能够设法把一个任意的输入文件转换成一个堆，则就可以使用一个如上所述的“由顶向下”的选择过程，来得到一个有效的排序算法。

R.W.弗洛伊德已经提出了建立堆的有效方法〔*CACM* 7 (1964)，701〕。假定已经有能力来安排这个文件，使得

$$K_{\lfloor j/2 \rfloor} \geqslant K_j \quad 对于\ l < \lfloor j/2 \rfloor < j \leqslant N \tag{5}$$

其中l是某个$\geqslant 1$的数。(在原来的文件中，这个条件对于$l = \lfloor N/2 \rfloor$“空虚地”成立，因为不存在满足条件$\lfloor N/2 \rfloor < \lfloor j/2 \rfloor < j \leqslant N$的下标$j$）不难看出应该怎样变换这个文件，使得（5）中的不等式可以推广到$\lfloor j/2 \rfloor = l$的情况，使它在以节点l为根的整个子树中都成立。因此可以将l减1，直到最终达到条件（3）。威廉斯和弗洛伊德的这些思想导致了下面的新颖算法，值得仔细地研究：

算法H（堆排序） 适当地重新安排记录$R_1, \cdots, R_N$；在完成了排序之后，它们的键将是有序的，$K_1 \leqslant \cdots \leqslant K_N$。首先，重新安排文件，使得它形成一个堆，然后反复地撤销堆顶，并把它传送到适当的最后位置。假定$N \geqslant 2$。

H1.〔初始化〕 置$l \leftarrow \lfloor N/2 \rfloor + 1$，$r \leftarrow N$。

H2.〔l或r减值〕 如果$l > 1$，则置$l \leftarrow l - 1$，$R \leftarrow R_l$，$K \leftarrow K_l$。(如果$l > 1$，则处于把输入文件变换为一个堆的过程中；而如果$l = 1$，则键$K_1 K_2 \cdots K_r$已组成一个堆。）否则，置$R \leftarrow R_r$，$K \leftarrow K_r$，$R_r \leftarrow R_1$，以及$r \leftarrow r - 1$；如果这使得$r = 1$，则置$R_1 \leftarrow R$，并且终止这个算法。

H3.〔准备“筛选”〕 置$j \leftarrow l$。(这时，对于$r < k \leqslant N$，我们有

$$K_{\lfloor j/2 \rfloor} \geqslant K_j \quad 对于\ l < \lfloor j/2 \rfloor < j \leqslant r \tag{6}$$

而且记录R_k处于它最后的位置。步骤H_3～H_8称为“筛选”算法；它们的效果等价于置$R_l \leftarrow R$然后重新安排$R_l, \cdots, R_r$，使得条件（6）对于$\lfloor j/2 \rfloor = l$也成立。)

H4.〔向下进行〕 置$i \leftarrow j$和$j \leftarrow 2j$。(在下列步骤中，我们有$i = \lfloor j/2 \rfloor$。）如果$j < r$，则一直前进到步骤H5；如果$j = r$，则转到步骤H6；而如果$j > r$，则转到H8。

H5.〔找“较大的”儿子〕 如果$K_j < K_{j+1}$，则置$j \leftarrow j + 1$。

H6.〔大于K？〕 如果$K \geqslant K_j$，则转到步骤H8。

H7.〔上移它〕 置$R_i \leftarrow R_j$，并返转到步骤H4。

H8.〔存储R〕 置$R_i \leftarrow R$(这终止了起始于步骤H3的“筛选”算法)。返回到步骤H2。

由于l和r的运动，堆排序有时被描述作“⊲”算法。上三角形表示当$r = N$且l减小到1时堆的建立阶段，而下三角形表示当$l = 1$且r减小到1时的选择阶段。表2示出了对十六个例数的堆排序过程（表中的每行示出在步骤H2之后的状态，方括号表示位置l和r）。

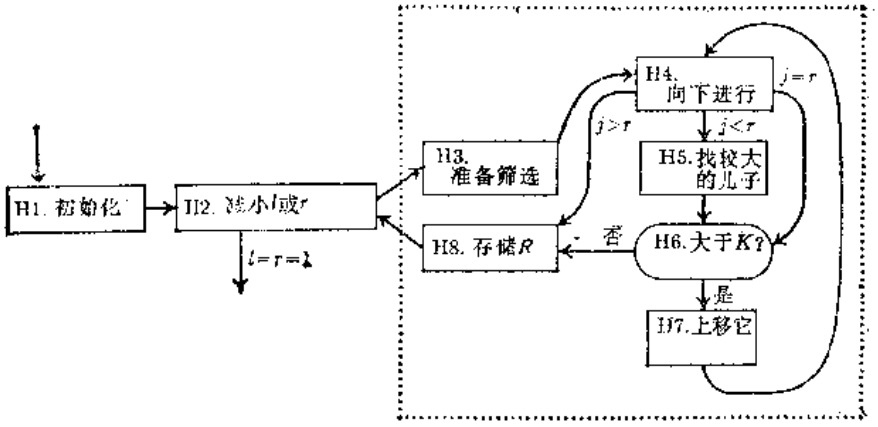


图26 堆排序，虚线包括的是“筛选”算法

表 2 堆排序的例子

K_1	K_2	K_3	K_4	K_5	K_6	K_7	K_8	K_9	K_{10}	K_{11}	K_{12}	K_{13}	K_{14}	K_{15}	K_{16}	l	r	K
503	087	512	061	908	170	897	[275	653	426	154	509	612	677	765	703]	8	16	275
503	087	512	061	908	170	[897	703	653	426	154	509	612	677	765	275]	7	16	897
503	087	512	061	908	[170	897	703	653	426	154	509	612	677	765	275]	6	16	170
503	087	512	061	[908	612	897	703	653	426	154	509	170	677	765	275]	5	16	908
503	087	512	[061	908	612	897	703	653	426	154	509	170	677	765	275]	4	16	061
503	087	[512	703	908	612	897	275	653	426	154	509	170	677	765	061]	3	16	512
503	[087	897	703	908	612	765	275	653	426	154	509	170	677	512	061]	2	16	087
[503	908	897	703	426	612	765	275	653	087	154	509	170	677	512	061]	1	16	503
[908	703	897	653	426	612	765	275	503	087	154	509	170	677	512]	908	1	15	061
[897	703	765	653	426	612	677	275	503	087	154	509	170	061]	897	908	1	14	512
[765	703	677	653	426	612	512	275	503	087	154	509	170]	765	897	908	1	13	061
[703	653	677	503	426	612	512	275	061	087	154	509]	703	765	897	908	1	12	170
[677	653	612	503	426	509	512	275	061	087	154]	677	703	765	897	908	1	11	170
[653	503	612	275	426	509	512	170	061	087]	653	677	703	765	897	908	1	10	154
[612	503	512	275	426	509	154	170	061]	612	653	677	703	765	897	908	1	9	087
[512	503	509	275	426	087	154	170]	512	612	653	677	703	765	897	908	1	8	061
[509	503	154	275	426	087	061]	509	512	612	653	677	703	765	897	908	1	7	170
[503	426	154	275	170	087]	503	509	512	612	653	677	703	765	897	908	1	6	061
[426	275	154	061	170]	426	503	509	512	612	653	677	703	765	897	908	1	5	087
[275	170	154	061]	275	426	503	509	512	612	653	677	703	765	897	908	1	4	087
[170	087	154]	170	275	426	503	509	512	612	653	677	703	765	897	908	1	3	061
[154	087]	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	2	061
[061]	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908	1	1	061

程序H(堆排序) 通过算法H，对单元INPUT+1到INPUT+N的诸记录进行排序，并对寄存器赋值如下：rI1≡$l-1$，rI2≡$r-1$，rI3≡i，rI4≡j，rI5≡$r-l$，rA≡K≡R，rX≡R_j。

01	START	ENT1	N/2	1	$H1$. 初始化，$l \leftarrow \lfloor N/2 \rfloor + 1$
02		ENT2	$N-1$	1	$r \leftarrow N$
03	1H	DEC1	1	$\lfloor N/2 \rfloor$	$l \leftarrow l-1$

04		LDA	INPUT+1,1	$\lfloor N/2 \rfloor$	$R \leftarrow R_l$, $K \leftarrow K_l$
05	3H	ENT4	1,1	P	$H3$. 准备"筛选"，$j \leftarrow l$
06		ENT5	0,2	P	
07		DEC5	0,1	P	rI5$\leftarrow r-j$
08		JMP	4F	P	转H4
09	5H	LDX	INPUT,4	$B+A-D$	$H5$. 找"较大的"儿子
10		CMPX	INPUT+1,4	$B-A-D$	
11		JGE	6F	$B+A-D$	如果$K_j \geqslant K_{j+1}$则转移
12		INC4	1	C	否则，置$j \leftarrow j+1$
13		DEC5	1	C	
14	9H	LDX	INPUT,4	$C+D$	rX$\leftarrow R_j$
15	6H	CMPA	INPUT,4	$B+A$	$H6$. 大于K?
16		JGE	8F	$B+A$	如果$K \geqslant K_j$，则转到H8
17	7H	STX	INPUT,3	B	$H7$. 上移它，$R_i \leftarrow R_j$
18	4H	ENT3	0,4	$B+P$	$H4$. 向下进行，$i \leftarrow j$
19		DEC5	0,4	$B+P$	rI5$\leftarrow$rI5$-j$
20		INC4	0,4	$B+P$	$j \leftarrow j+j$
21		J5P	5B	$B+P$	如果$j<r$，则转到$H5$
22		J5Z	9B	$P-A+D$	如果$j=r$，则转到$H6$
23	8H	STA	INPUT,3	P	$H8$. 存储R，$R_i \leftarrow R$
24	2H	JIP	1B	P	$H2$. 减值r
25		LDA	INPUT+1,2	$N-1$	如果$l=1$，则置$R \leftarrow R_r$, $K \leftarrow K_r$
26		LDX	INPUT+1	$N-1$	
27		STX	INPUT+1,2	$N-1$	$R_r \leftarrow R_1$
28		DEC2	1	$N-1$	$r \leftarrow r-1$
29		J2P	3B	$N-1$	如果$r>1$，则转到$H3$
30		STA	INPUT+1	1	$R_1 \leftarrow R$

这个程序的长度大约仅是程序S的两倍，但当N很大时，它更为有效。它的运行时间依赖于

$P=N+\lfloor N/2 \rfloor-2$，筛选扫描的次数；

A　使键K最终地固定在堆的一个内部节点上的筛选扫描次数；

B　在筛选期间被提升的键总数；

C　步骤H5中$j \leftarrow j+1$的次数；

D　步骤H4中$j=r$的次数。

这些量在下边进行分析；实际上，相当地接近它们的平均值

$$A \approx 0.349N,\quad B \approx N\log_2 N-1.87N$$
$$C \approx \frac{1}{2}N\log_2 N-0.9N,\quad D \approx \ln N \tag{7}$$

例如，当$N=1000$时，对于随机输入的四项实验分别给出$(A, B, C, D)=(371, 8055, 4056, 12)$，$(351, 8072, 4087, 14)$，$(341, 8094, 4087, 8)$，$(340, 8108, 4083, 13)$。因此，总

运行时间 $7A+14B+4C+20N-2D+15\lfloor N/2\rfloor-28$，平均说来近似于 $16N\log_2 N+0.2N$ 个单位。

观察表 2 就使我们确信，堆排序是非常有效的；在把大的键储藏在右边之前左移之！当 N 很小时，它确实是不可思议的排序方法；表 2 中 16 个键的排序时间是 1068 u，而简单的直接插入法（程序 5.2.1 S）仅花费 514 u。直接选择法（程序 S）花费 853 u。

对于较大的 N，程序H比较有效。由于程序H、谢尔减小增量排序（程序 5.2.1 D）和霍尔快速排序（程序 5.2.2 Q）这三个程序都通过键的比较进行排序，而且较少或不用辅助存储，读者不妨拿程序H同其余两者进行比较。当 $N=1000$ 时，近似的平均运行时间是

对于堆排序　160000 u

对于谢尔排序　130000 u

对于快速排序　80000 u

(MIX是今天大多数计算机的典型代表，但特殊的机器当然将产生稍微不同的相对数值。)当 N 变得更大时，堆排序将优于谢尔排序，但渐近公式 $16N\log_2 N\approx 23.08N\ln N$ 将决不超过快速排序的 $12.67N\ln N$。习题 18 中所讨论的对堆排序的修改，将提高在许多计算机上的处理速度，但即使经过改进也仍然赶不上快速排序。

另一方面，快速排序仅仅是平均地有效的，而且它的最坏情况是 N^2 阶的。堆排序具有有趣的性质，它的最坏情况并不比平均情况坏多少：我们总有

$$A\leqslant 1.5N,\quad B\leqslant N\lfloor\log_2 N\rfloor,\quad C\leqslant N\lfloor\log_2 N\rfloor \tag{8}$$

所以不论输入数据的分布如何程序H花费的时间单位将不多于 $18N\lfloor\log_2 N\rfloor+38N$。堆排序是已经看到的头一个保证具有 $N\log_2 N$ 阶的排序方法。下面 5.2.4 节中讨论的合并排序也有这个性质，但它要求较多的存储空间。

最大者先出　在第二章中我们已经看到，线性表通常可以按照对其所实施的插入和删去操作的特性而合理地划分为不同的类型。在每次删去都是撤销表中最年轻的项目这一意义下，一个栈有“后进先出”的特性（所谓最年轻的项目是指在所有当前存在的项目中最新插入的项目）。而一个简单的队，在每次删去都是撤销剩下的最老的项目这一意义下，则有“先进先出”的特性。在更复杂的情况下，例如 2.2.5 节的电梯模拟，需要一个“最小者先出”表，其中每次删去都是撤销有最小键的项目。这样一个表可以称为一个优先队，因为每个项目的键反映了它迅速离开这个表的相对能力。选择排序是优先队的特殊情况，其中我们在做了 N 次插入之后，接着做 N 次删去。

优先队有广泛的应用。例如，某些数值迭代方案是以重复选择符合某项测试准则的最大（或最小）值的项目为基础的。选中的项目的参数被改变，并根据参数的新值，按新的测试值重新插入到表中。操作系统通常利用优先队进行作业调度。习题 15、29 和 36 提出了优先队其它典型的应用。在后面数章中将出现许多其它的例子。

应如何实现优先队呢？显然的方法之一是维持一个排好序的表，其中按键的次序排列诸项目。于是，插入一个新项目实质上就是我们在 5.2.1 节插入排序的研究中所讨论的同样问题。另一个处理优先队的甚至更为显然的方法，是让元素表的次序任意，每当需要删去具有最大（或最小）键的元素时，就把这个元素找出来。这两个显然的方法，麻烦都在

于：当在表中有N项时，插入或删去要求阶为N的步骤。所以当N很大时，它们是非常消耗时间的。

在威廉斯关于堆排序的最初论文中，他指出把堆应用于大的优先队是很理想的，因为我们可以在$O(\log_2 N)$步中向一个堆插入或从一个堆删去元素；而且，堆的所有元素都紧凑地放置在连续的存储单元中。算法II的选择阶段是一系列按最大者先出原则进行的删去步骤：为删去最大的元素K_1，我们撤销它，并"筛选"出K_N进入$N-1$个元素的新堆中。（如果如同在电梯模拟中那样要一个最小者先出算法，则显然可以改变堆的定义，使得在（3）中"≥"变成为"≤"；为了方便起见，在这里只考虑最大者先出的情况。）一般地说，如果要删去最大的项目，然后插入一个新的元素x，则可以以$l=1$，$r=N$，和$K=x$来进行筛选过程。如果希望插入一个元素x，而无须预先删去堆中的元素，则可以使用习题16的"由底向上"的过程。

优先队的一个链接表示 把优先队表示为链接二叉树的一个有效方法是1976年由克拉克·A.克兰（Clark A. Crane）提出的，他的方法要求在每个记录中有两个链接场以及一个小的计数场，它比起堆来有如下优点：

1）当优先队被处理作一个栈时，插入和删去操作是更为有效的（它们花费的时间是固定的，与队大小无关）。

2）这些记录决不移动，仅仅改变指针。

3）有可能仅在$O(\log_2 N)$步之内把总共有N个元素的两个分离的优先队合并成一个优先队。

稍经修改后的克兰方法如图27，它示出特殊类型的二叉树结构。每个节点包含一个KEY（键）场，一个DIST（距离）场，以及两个链接场LEFT（左）和RIGHT（右）。DIST场的值总是等于从该节点到此树的一个叶（即一个空的Λ节点）的最短通路长度。如果认为DIST(Λ)＝0和KEY(Λ)＝$-\infty$，则在这株树中的KEY和DIST场满足下列性质：

$$\mathrm{KEY(P)} \geqslant \mathrm{KEY(LEFT(P))},\ \mathrm{KEY(P)} \geqslant \mathrm{KEY(RIGHT(P))} \tag{9}$$

$$\mathrm{DIST(P)} = 1 + \min(\mathrm{DIST(LEFT(P))},\ \mathrm{DIST(RIGHT(P))}) \tag{10}$$

$$\mathrm{DIST(LEFT(P))} \geqslant \mathrm{DIST(RIGHT(P))} \tag{11}$$

关系（9）类似于堆的条件（3），它确保树根有最大的键；而关系（10）恰是上面所述的DIST场的定义。关系（11）是有趣的新方法：它意味着到一个叶的最短通路总可以通过向右移动而得到。我们说，具有这种性质的树是一个左倾树，因为它有如此强烈地"倾向于"左边的趋势。

由这些定义，显然，DIST(P)＝n意味着在P下面至少存在2^n个叶节点；否则，将有一条更短的通路从P通到一片叶子。于是，如果在一个左倾树中有N个节点，则从根向下往右边走的通路至多包含$\lfloor \log_2(N+1) \rfloor$个节点。通过遍历这一通路，就可能把一个新点插入到优先队中（见题32）；因此，在最坏情况下仅仅需要$O(\log_2 N)$步。当树是线性的时（所有RIGHT链接都是Λ）出现最好的情况，而当树完全平衡时出现最坏的情况。

为了撤销在根处的节点，我们只需合并它的两株子树。合并两个分别由P和Q所指向

的不相交的左倾树的操作，从概念上说是简单的：如果KEY(P)≥KEY(Q)，则把P取作根，把Q同P的右子树合并；然后修改DIST(P)，必要时互换LEFT(P)与RIGHT(P)。这一过程的详细描述是不难想出来的（见习题32）。

优先队技术的比较 当节点的数目N很小时，最好使用一种直截了当的线性表示方法，来保存一个优先队；但当N很大时，$\log_2 N$的方法显然快得多。因此，很大的优先队一般被表示成堆或左倾树。在6.2.3节，将通过**平衡树**讨论表示线性表的另一个方法，而这就导致了适合于表示优先队的第三个$\log_2 N$方法。因此对这三项技术作一比较是适当的。

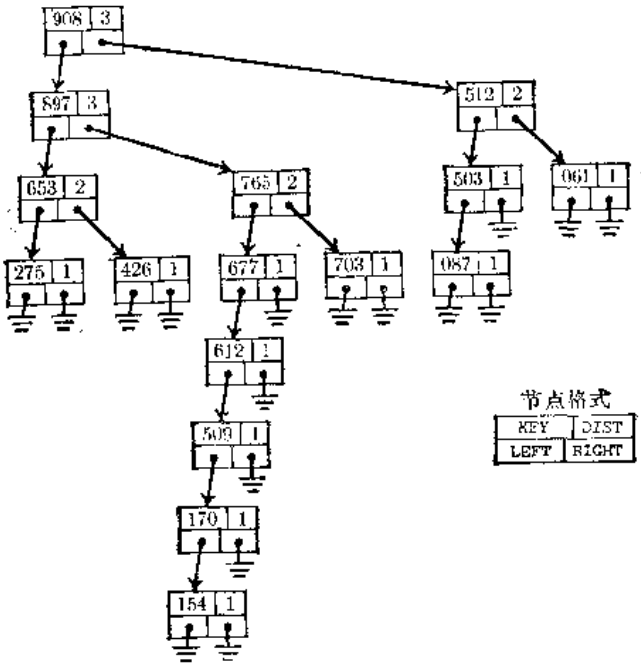


图27 表示为一株左倾树的优先队

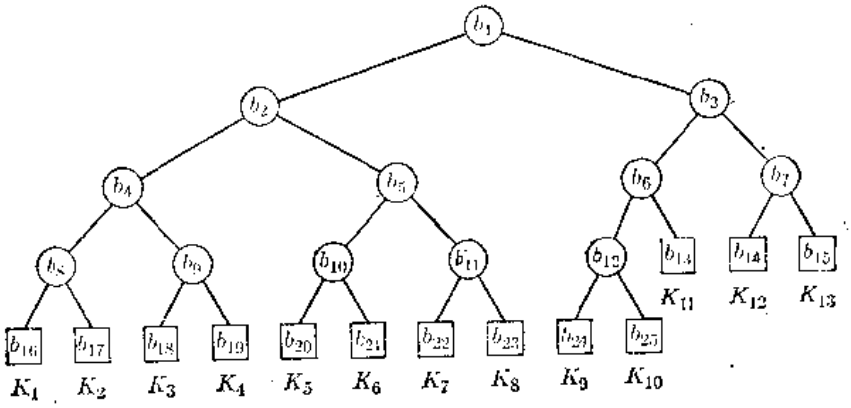
已经看出，左倾树操作一般比堆操作稍微快些，但堆花费的存储空间较少。平衡树大约花费和左倾树同样多的空间，也许稍微少些；其操作比堆慢些，而且程序设计更复杂，但平衡树结构在若干方面是相当灵活的。当使用一个堆或一株左倾树时，我们不易预测，对于具有相等键的两个项目将发生什么；不可能保证，具有相等键的项目将以后进先出或先进先出的方式被处理，除非键扩展成包括附加的“插入的顺序编号”场，使得实际上不出现相等的键。另一方面，通过平衡树，能容易地作出关于相等键的一致的约定，并且也能做诸如“在y之前（或之后）直接插入x”等操作。平衡树是对称的，以致能在任何时刻删去最大的元素或者最小的元素，而堆和左倾树必然倾向于这边或那边（然而，见习题31，它说明怎样构造对称的堆）。平衡树既可用作查找，也可用作排序；而且，能颇为快速地从一株平衡树撤销连续的元素块区。但是两株平衡树不可能以少于$O(N)$步进行合并，而左倾树则可仅以$O(\log_2 N)$步加以合并。

总之，堆使用极小的存储；左倾树对于合并不相交的优先队是很好的；而如有必要，可以合理的代价利用平衡树的灵活性。

当队中的元素个数多少保持为常量时，5.4.1节中叙述了处理优先队的另一方法。

一种合意的特殊情况 如果已知一个优先队的所有元素包含在某个固定的集合$\{K_1, K_2, \cdots K_N\}$中，其中$K_1<K_2<\cdots<K_N$，那么，有一个简单得多的方法来实现一个有效的表示。事实上，这个方案仅仅要求存储器的$2N-1$个二进位，而且诸操作十分容易编程序。

这个思想是使用有N个外部节点的完备二叉树，而且对于$1\leqslant j\leqslant 2N-1$，在每个节点中用存储器的一个二进位$b_j$。例如，当$N=13$时，我们有



外部节点隐含地同从左到右处于递增顺序的键相关联；例如，b_{18}同K_5相关联，而b_{15}同K_{11}相关联。由假定，优先队的内容是$\{K_1, \cdots, K_N\}$的某个子集S。表示S的方法是：当且仅当在某个b_j之下有一个相关联的键出现时，即置b_j为1。于是如果$S=\{K_5, K_7, K_8\}$，将有$b_1=b_2=b_4=b_6=b_9=b_{11}=b_{18}=b_{22}=b_{23}=1$，而其它的$b_j$是0。事实上，我们有类似图23的一个锦标赛，而且0代替了$-\infty$。

容易看出怎样在$\log_2 N$步的数量级内找出S的最大元素：首先置$j \leftarrow 1$，然后重复地置$j \leftarrow 2j+b_{2j+1}$直到$j \geqslant N$；然后b_j将是同所要的键相关联的外部节点。(如果S可以是空的，也应在这个算法的结尾测试$b_j=1$。注意b_1决不需要，因而可以省略它。)

同样，容易找S的最小元素，或插入或删去元素。所有这些操作将花费$\log_2 N$步的数量级（注意，这是优先队极大长度的对数，而不是实际长度的对数；然而，差别可能不是甚殊的)。

表示上述优先队的方法是卢瑟·C. 阿贝尔（Luther C.Abel）给出的〔Ph.D thesis (Univ.of Illinois, 1972)，106-109〕。

***堆排序的分析** 算法H还未被完备地分析过，但是它的若干性质不难推导出来。因此将通过稍微详细地研究一个堆的构造，来结束这一节。

图28示出了具有26个元素的一个堆的形状；每个节点都已根据它在此堆中的下标用二进记号标出。图中的星号表示所谓的特殊节点，它位于从1到N的通路上。

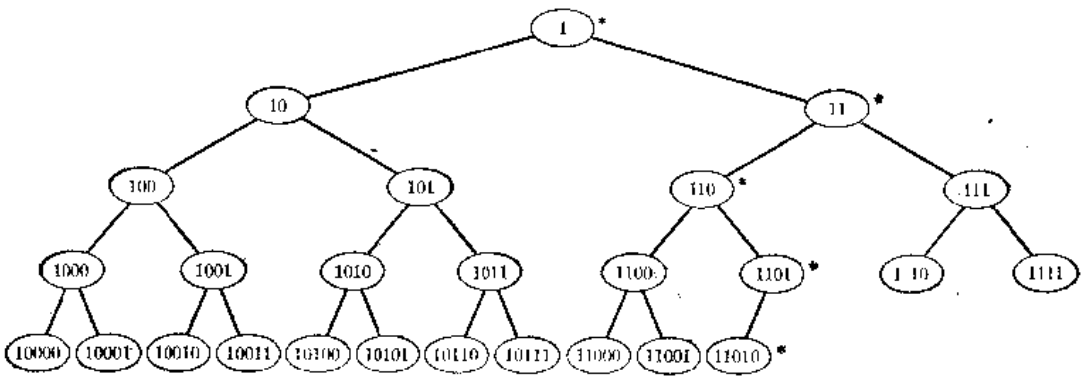


图28 一个$26=(11010)_2$个元素的堆的图示

堆的最重要的属性之一是它的子树大小的集合。例如，在图 28 中以 1，2，…，26 为根的子树的大小分别是

$$26^*, 15, 10^*, 7, 7, 6^*, 3, 3, 3, 3,$$
$$3, 3, 2^*, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1^* \tag{12}$$

星号表示以特殊节点为根的特殊子树；习题 20 说明，如果N的二进表示是

$$N=(b_n b_{n-1}\cdots b_1 b_0)_2,\quad n=\lfloor \log_2 N \rfloor \tag{13}$$

则特殊子树的大小总是

$$(1b_{n-1}\cdots b_1 b_0)_2,\ (1b_{n-2}\cdots b_1 b_0)_2,\ \cdots,\ (1b_1 b_0)_2,\ (1b_0)_2,\ (1)_2 \tag{14}$$

非特殊的子树总是完全平衡的，所以它们的大小总是形如 2^k-1。习题 21 说明非特殊子树的大小恰由

$$\lfloor (N-1)/2 \rfloor 个 1,\ \lfloor (N-2)/4 \rfloor 个 3,\ \lfloor (N-4)/8 \rfloor 个 7, \cdots, \tag{15}$$
$$\lfloor (N-2^{n-1}) \rfloor / 2^n \rfloor 个\ (2^n-1)$$

所组成。例如，图 28 含有 12 株大小为 1 的，6 株大小为 3 的，两株大小为 7 的以及一株大小为 15 的非特殊子树。

设s_l是其根为 l 的子树的大小，M_L是所有这些大小的多重集合$\{s_1, s_2, \cdots, s_N\}$。利用(14) 和 (15)，对于任何给定的$N$，可以容易地计算出$M_N$。习题 5.1.4-20 告诉我们，把整数$\{1, 2, \cdots, N\}$排列成一个堆的方式总数是

$$N!/s_1 s_2 \cdots s_N = N!/\prod_{s\in M_N} s \tag{16}$$

例如，放置 26 个字母$\{A, B, C, \cdots, Z\}$到图 28 中使得竖线保持字母顺序，其方式个数为

$$26!/(26\cdot 10\cdot 6\cdot 2\cdot 1\cdot 1^{12}\cdot 3^6\cdot 7^2\cdot 15^1)$$

我们现在可以来分析算法H中的堆建立阶段，即在步骤H2 中出现条件 $l=1$ 之前的阶段。幸而，可以把对堆建立的研究归结为对于独立的筛选操作的研究，这是由下面的定理导出的。

定理H 如果算法H的输入是$\{1, 2, \cdots, N\}$的随机排列，则堆建立阶段可以提供$N!/(\pi_{s\in M_N} s)$种堆，每种堆都是同等可能的。而且，在此阶段执行的$\lfloor N/2 \rfloor$个筛选操作在下列意义下是均匀的，即当到达步骤H8 时，i 的所有s_i种值都是同等可能的。

证明 可以应用数值分析家们也许称之为“向后分析”的方法；给定以 l 为根的筛选操作的一个可能结果$K_1\cdots K_N$，我们看到，恰有这个文件的s_l个以前的配置$K'_1\cdots K'_N$，将筛选出这结果。这些以前的配置的每一个都有不同的K'_l值；因此向后看去，恰有$\{1, 2, \cdots, N\}$的$s_l s_{l+1}\cdots s_N$个输入排列，在位置 l 的筛选完成之后，它们产生配置$K_1\cdots K_N$。

情况 $l=1$ 是典型的：设$K_1\cdots K_N$是一个堆，$K'_1\cdots K'_N$是一个文件，当 $l=1$，$K=K'_1$时，该文件通过筛选操作变换成$K_1\cdots K_N$。如果$K=K_i$，则必定有 $K'_i=K_{\lfloor i/2 \rfloor}$，$K'_{\lfloor i/2 \rfloor}=K_{\lfloor i/4 \rfloor}$，等等，而对于不在从 1 到 i 的通路上的所有 j，$K'_j=K_j$。反之，对于每个 i，这个构造产生一个文件$K'_1\cdots K'_N$，使得（a）筛选操作把$K'_1\cdots K'_N$变换成$K_1\cdots K_N$，以及(b) 对于 $2\leqslant \lfloor j/2 \rfloor < j \leqslant N$，$K_{\lfloor j/2 \rfloor}\geqslant K_j$。因此，恰能有$N$个这样的文件$K'_1\cdots K'_N$，而

且筛选操作是均匀的。(这个定理证明的一个例子出现于习题 22 中。)

考虑程序H的分析中的量A, B, C, D,可以看到，在大小为s的一株子树上的均匀筛选操作，对A的平均值的贡献是$\lfloor s/2 \rfloor / s$；对B的平均值的贡献是

$$\frac{1}{s}(0+1+1+2+\cdots+\lfloor \log_2 s \rfloor)=\frac{1}{s}\left(\sum_{1\leqslant k\leqslant s}\lfloor \log_2 k \rfloor\right)$$

$$=\frac{1}{s}((s+1)\lfloor \log_2 s \rfloor-2^{\lfloor \log_2 s \rfloor+1}+2)$$

(见习题 1.2.4-42)；而且根据s是偶数还是奇数，它对D的平均值的贡献是$2/s$或0。对C的相应的贡献更难确定些，所以把它留给读者(见习题 26)。对于所有筛选操作进行求和，我们求出在堆建立期间A的平均值是

$$A'_N=\sum_{s\in M_N}\lfloor s/2 \rfloor / s \tag{17}$$

对于B、C和D，类似的公式也成立。因此不难精确地计算这些平均值；下表给出了典型的结果：

N	A'_N	B'_N	C'_N	D'_N
99	19.18	68.35	42.95	0.00
100	19.93	69.39	42.71	1.84
999	196.16	734.66	464.53	0.00
1000	196.94	735.80	464.16	1.92
9999	1966.02	7428.18	4695.54	0.00
10000	1966.82	7429.39	4695.06	1.97
10001	1966.45	7430.07	4695.84	0.00
10002	1967.15	7430.97	4695.95	1.73

从渐近的意义上说，可以忽略M_N中特殊子树的大小，例如可以求出

$$A'_N=\frac{N}{2}\cdot\frac{0}{1}+\frac{N}{4}\cdot\frac{1}{3}+\frac{N}{8}\cdot\frac{3}{7}+\cdots+O(\log_2 N)$$

$$=\left(1-\frac{1}{2}\alpha\right)N+O(\log_2 N) \tag{18}$$

其中

$$\alpha=\sum_{k\geqslant 1}\frac{1}{2^k-1}=1.60669\ \ 51524\ \ 15291\ \ 76378\ \ 33015\ \ 23190\ \ 92458\ \ 04805\cdots \tag{19}$$

(这个值已由小 J. W. 伦奇(J. W. Wrench, Jr.)用习题 27 的级数变换计算出来。)对于很大的N，可以使用近似公式

$$\begin{aligned}
A'_N&\approx 0.1967N+0.3(N\text{偶}),\quad 0.1967N-0.3(N\text{奇});\\
B'_N&\approx 0.74403N-1.3\ln N;\\
C'_N&\approx 0.47034N-0.8\ln N;\\
D'_N&\approx 1.8\pm 0.2(N\text{偶}),\quad 0\ (N\text{奇})
\end{aligned} \tag{20}$$

极小值和极大值也容易确定。为建立这个堆，只需要$O(N)$步(参考习题 23)。

这实质上处理了算法H的堆建立阶段。但是，选择阶段是另一回事，还未向读者交代！

命 A''_N、B''_N、C''_N 和 D''_N 表示当 N 个元素被堆排序时在选择阶段中 A、B、C 和 D 的平均值。对于随机输入，算法H的特性相当接近于经验确定的平均值

$$
\begin{aligned}
&A''_N \approx 0.152N;\\
&B''_N \approx N\log_2 N - 2.61N;\\
&C''_N \approx \frac{1}{2}N\log_2 N - 1.4N;\\
&D''_N \approx \ln N \pm 2
\end{aligned}
\tag{21}
$$

但是，对于这些常数还没有找到适当的理论说明。通过考虑个别的筛选操作，不难导出（8）中所给的上界。但是，当把这个算法作为一个整体考虑时，C 的上界大概将降低到约 $\frac{1}{2}N\log_2 N$。

习题

1.〔10〕 直接选择（算法S）是一个“稳定的”排序方法吗？

2.〔15〕 为什么在算法S中首先选择最大的键，然后次大的，等等。而不是首先求最小的，然后次小的，等等？为什么前一个做法更有效？

3.〔M21〕（a）证明，如果算法S以 $\{1, 2, \cdots, N\}$ 的一个随机排列开始，则步骤S2和S3的头一次迭代产生以 N 结尾的 $\{1, 2, \cdots, N-1\}$ 的一个随机排列（换言之，在 $K_1 \cdots K_{N-1}$ 中 $\{1, 2, \cdots, N-1\}$ 的每种排列的出现都是同等可能的）。（b）因此，如果 B_N 表示程序S中量 B 的平均值，给定随机排列的输入，则我们就有 $B_N = H_N - 1 + B_{N-1}$〔提示：参考等式1.2.10-16〕。

▶4.〔M25〕 当 $i=j$ 时算法S的步骤S3什么也不做；在进行步骤S3之前测试是否 $i=j$，是一个好想法吗？对于随机输入，在步骤S3中条件 $i=j$ 出现的平均次数是多少？

5.〔20〕 当输入是 $N \cdots 3\ 2\ 1$ 时，在程序S的分析中量 B 的值是什么？

6.〔M29〕（a）设 $a_1\ a_2 \cdots a_N$ 是 $\{1, 2, \cdots, N\}$ 的一个排列，它有 C 个循环和 I 个反序，以及当由程序S进行排序时有 B 个自右到左的极大值变化。证明 $2B \leqslant I + N - C$〔提示：见习题5.2.2-1〕。（b）证明 $I + N - C \leqslant \lfloor n^2/2 \rfloor$，因此 B 决不能超过 $\lfloor n^2/4 \rfloor$。

7.〔M46〕 假定随机输入，试求作为 N 的一个函数的程序S中量 B 的方差。

▶8.〔24〕 证明，如果在步骤S2中查找 $\max(K_1, \cdots, K_j)$ 时，按自左到右的次序 K_1，K_2，…，K_j 来考察键，而不是象在程序S中那样按 $K_j, \cdots, K_2$，K_1 的次序来考察，则通常有可能减少步骤S2的下一次迭代所需要的比较次数。试根据这个想法，写出一个MIX程序。

9.〔M25〕 对于随机输入，由习题8的算法执行比较的平均次数是多少？

10.〔12〕 在原来的16个项目中的14项已经输出之后，图23中树的配置将是怎样的？

11.〔10〕 在元素908输出之后，图24的树的配置将如何？

12.〔20〕 当使用图23的“由底向上”方法把 2^n 个元素的一个文件排成有序时，$-\infty$ 同 $-\infty$ 将比较多少次？

13.〔20〕（J.W.J.威廉斯）算法H的步骤H4区分$j<r$，$j=r$和$j>r$三种情况。证明，如果$K\geqslant K_{r+1}$，则将有可能简化步骤H4，从而只需作一个两路分支。问应怎样修改步骤H2，才能确保$K\geqslant K_{r+1}$的条件贯穿于堆排序过程？

14.〔10〕证明简单队是优先队的特殊情况（说明如何把键赋予诸元素，使得最大者先出的过程等价于先进先出）。一个栈也是优先队的一种特殊情况吗？

15.〔$M22$〕〔B.A.查特勒斯（B.A.Chartres）〕设计一个高速算法，它构造一张$\leqslant N$的质数的表，并利用一个优先队来避免除法操作。〔提示：命优先队中最小的键，是比上一个作为质数候选者的奇数要大的最小非质奇数。试使队中元素个数极少。〕

16.〔20〕设计一个有效的算法，它把一个新的键插入到n个元素的一个给定的堆中，产生$n+1$个元素的一个堆。

17.〔20〕习题16的算法可代替算法H中的"减小l到1"的方法，以用来建立堆。当它们以相同的输入文件开始时，这两个方法是否建立相同的堆？

18.〔21〕（R.W.弗洛伊德）在堆排序的选择阶段，键K势必是十分小的，所以步骤H6中几乎所有的比较都发现$K<K_j$。说明如何修改这个算法，使得在计算的主循环中K不同K_j进行比较，从而把比较的平均次数削减一半。

19.〔21〕试设计一个算法，它删去长度为N的堆中一个给定的元素，产生长度为$N-1$的一个堆。

20.〔$M20$〕证明（14）给出一个堆中的特殊子树的大小。

21.〔$M24$〕证明（15）给出一个堆中的非特殊子树的大小。

▶22.〔20〕在算法H的堆建立阶段，{1，2，3，4，5}的什么排列变换成5 3 4 1 2？

23.〔$M28$〕（a）证明在一个筛选算法中扫描的长度B决不超过$\lfloor\log_2(r/l)\rfloor$。（b）根据（8），在算法H的任何具体应用中，$B$决不能超过$N\lfloor\log_2 N\rfloor$。在所有可能的输入文件中，试求$B$的极大值（作为$N$的一个函数）。（你必须证明，存在一个输入文件，使$B$取到这个极大值。）

24.〔$M24$〕推导出B_N'（在算法H的堆建立阶段的扫描总长度）的标准离差的精确公式。

25.〔$M20$〕当$l=1$和$r=N$时，如果$N=2^{n+1}-1$，则在筛选扫描期间对C贡献的平均值是多少？

26.〔$M30$〕求解习题25，（a）对于$N=26$，（b）对于一般的N。

27.〔$M25$〕（小J.W.伦奇）证明$\sum_{n\geqslant 1}x^n/(1-x^n)=\sum_{n\geqslant 1}x^{n^2}(1+x^n)/(1-x^n)$（置$x=\frac{1}{2}$即得一个用于计算（19）的非常迅速地收敛的级数）。

28.〔35〕以完备的三叉树而不是二叉树为基础，来剖析三叉堆的思想。三叉堆排序比二叉堆更快吗？

29.〔26〕〔W.S.布朗（W.S.Brown）〕设计一个用于多项式或幂级数$(a_1x^{i_1}+a_2x^{i_2}+\cdots)(b_1x^{j_1}+b_2x^{j_2}+\cdots)$的乘法的算法，其中，答案$c_1x^{i_1+j_1}+\cdots$的系数按照输入系数相乘的次序来生成〔提示：使用一个适当的优先队〕。

30.〔$M48$〕 当一个文件被堆排序时，C还能超过$\frac{1}{2}N\log_2 N$吗？ C的极大值是多少？极小值是多少？

31.〔37〕（J.W.J.威廉斯） 证明，如果两个堆以一种适当的方式“背对背”地放在一起，则有可能维持一种结构，使得在任何时候都可以用$O(\log_2 N)$步删去最小的或最大的元素（这样一种结构可以叫做一个优先双队）。

32.〔21〕 设计一个算法，它把表示作左倾树的两个不相交的优先队合并成一个（特别是，如果给定的两队之一只包含单个元素，则你的算法将把它插入到另一个队中）。

33.〔15〕 在表示作一个左倾树的优先队中，为什么要通过合并两个子树，而不是象在一个堆中那样把节点“提升”到顶上，来完成撤销根的操作？

34.〔$M47$〕 如果不考虑KEY的值，有多少种可能的N个结点的左倾树？〔这个序列开始是1，1，2，4，8，17，38，…；是否有一个简单的渐近公式呢？〕

35.〔26〕 如果UP链接被加到一个左倾树上（参考6.2.3节中三重链接树的讨论），则有可能从优先队内删去任何节点P如下：用合并后的LEFT(P)和RIGHT(P)代替P；然后调整P诸祖宗的DIST场，直到达到根或者达到一个其DIST场未被改变的节点为止。

试证明：如果在这株树中有N个节点，即使这株树可以包含非常长的向上通路，这个过程也决不要求改变多于$O(\log_2 N)$个DIST场。

36.〔18〕（最近最少使用的页的替换） 许多操作系统都使用下列类型的算法：对一个节点集合施加两个操作，“使用”一个节点，以及用一个新节点替换最近最少“使用”的节点。试问什么样的数据结构有利于确认最近最少“使用”的节点？

5.2.4 通过合并进行排序

合并（或者整理）意味着把两个或多个有序文件组成一个单一的有序文件。例如：我们可以合并两个文件503 703 765和087 512 677得到087 503 512 677 703 765。实现这一合并的一个简单方法是比较两个最小的项目，输出最小的，而后重复这同一过程。以

$$\begin{cases} 503 \quad 703 \quad 765 \\ 087 \quad 512 \quad 677 \end{cases}$$

开始，我们得到

$$087 \begin{cases} 503 \quad 703 \quad 765 \\ 512 \quad 677 \end{cases}$$

然后

$$087 \quad 503 \begin{cases} 703 \quad 765 \\ 512 \quad 677 \end{cases}$$

等等。当两个文件之一已被取尽时，需要稍注意些；在下列算法中有这一过程的详细描述：

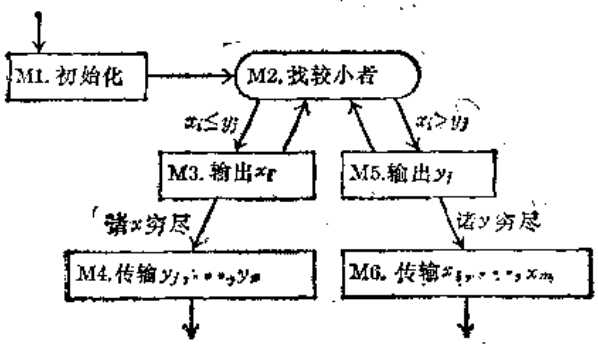


图29　合并$x_1 \leqslant \cdots \leqslant x_n$和$y_1 \leqslant \cdots \leqslant y_n$

算法 M　（两路合并）　本算法把有序文件 $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_m$ 和 $y_1 \leqslant y_2 \leqslant \cdots \leqslant y_n$ 合并成一个单一文件 $z_1 \leqslant z_2 \leqslant \cdots \leqslant z_{m+n}$。

M1.〔初始化〕　置 $i \leftarrow 1$，$j \leftarrow 1$，$k \leftarrow 1$。

M2.〔找较小者〕　如果 $x_i \leqslant y_j$，则转到步骤 M3，否则转到 M5。

M3.〔输出 x_i〕　置 $z_k \leftarrow x_i$，$k \leftarrow k+1$，$i \leftarrow i+1$。如果 $i \leqslant m$，则返回 M2。

M4.〔传送 $y_j, \cdots, y_n$〕　置 $(z_k, \cdots, z_{m+n}) \leftarrow (y_j, \cdots, y_n)$ 并终止此算法。

M5.〔输出 y_j〕　置 $z_k \leftarrow y_j$，$k \leftarrow k+1$，$j \leftarrow j+1$。如果 $j \leqslant n$，则返回 M2。

M6.〔传送 $x_i, \cdots, x_m$〕　置 $(z_k, \cdots, z_{m+n}) \leftarrow (x_i, \cdots, x_m)$ 并终止此算法。

我们将在 5.3.2 节看到，当 $m \approx n$ 时，这个直截了当的过程实质上是在一台通常的计算机上进行合并的“最好”的方法。（另一方面，当 m 比 n 小得多时，却有可能设计一个更有效的合并算法，尽管一般说来它们是比较复杂的。）通过在输入文件的末端设置人工的“哨兵”元素 $x_{m+1} = y_{n+1} = \infty$，就可以在不降低很多效率的前提下使算法M稍简单些，使其恰恰在输出∞之前停止。关于算法M的分析，见习题 2。

算法M的总工作量基本上与 $m+n$ 成比例，所以显然，比起排序来，合并是较为简单的问题。而且，我们可以把排序问题归结为合并，因为有可能继续合并越来越长的子文件直到一切都是有序的为止。可以把这当作排序思想的一个推广：把一个新元素插入到一个有序的文件是合并的特殊情况 $n=1$！如果要加速这个插入过程，则可以考虑一次插入若干个元素，“成批”处理它们，而这自然地导致合并排序的一般思想。从历史上看，合并排序是用于计算机排序早期的方法之一；它早在 1945 年就由约翰·冯·诺依曼提出来了（见 5.5节）。

我们将在 5.4 节颇为详细地研究合并的问题，并考虑外部排序算法；本节，只专注于在一个高速随机存取存储器内的合并排序问题。

表 1 说明了一种合并排序，它类似于在快速排序、基数交换等方法中使用过的扫描过程的形式，“两头点蜡”：同时从左边和右边考察输入数据，并向中间靠拢。现在暂时忽略这个表的顶上那行，而考虑从行 2 到行 3 的变换。在左边，有递增的路段 503　703　765；在右边向左读，有路段 087　512　677。合并这两个序列得到 087　503　512　677　703　765，它被放置在行 3 的左边。然后行 2 中的键 061　612　908 同 170　509　897 合并，结果（061　170　509　612　897　908）被记录到行 3 的右端。最后，154　275　426　653

同 653 合并（及时发现了重叠，使它不致为害），这个结果被放到左边。本表的行 2 以同样方式由行 1 的原始输入数据形成。

表 1　自然的两路合并排序

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	765	061	612	908	154	275	426	653	897	509	170	677	512	087
087	503	512	677	703	765	154	275	426	653	908	897	612	509	170	061
061	087	170	503	509	512	612	677	703	765	897	908	653	426	275	154
061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

表 1 中垂直的线表示路段之间的边界。这些垂线都是所谓的“下坡线”，它们前边的元素大于后边的元素。在这个文件的中间，当我们从两个方向读到同一个键时，一般来说，要出现一个含混的状态；但如果象在下列算法中那样稍加注意，就不致有问题了。由于这个方法利用了在它的输入中自然地出现的“路段”，故习惯上称为“自然的”合并。

算法 N（自然的两路合并排序）　使用两个存储区域，对记录 $R_1, \cdots, R_N$ 排序，每个存储区域都能容纳 N 个记录。为了方便起见，我们假定，第二个区域的记录是 $R_{N+1}, \cdots, R_{2N}$，实际上 R_{N+1} 当然不必相邻于 R_N。$R_{N+1}, \cdots, R_{2N}$ 的初始内容是无所谓的。在完成排序之后，键将是有序的：$K_1 \leqslant \cdots \leqslant K_N$。

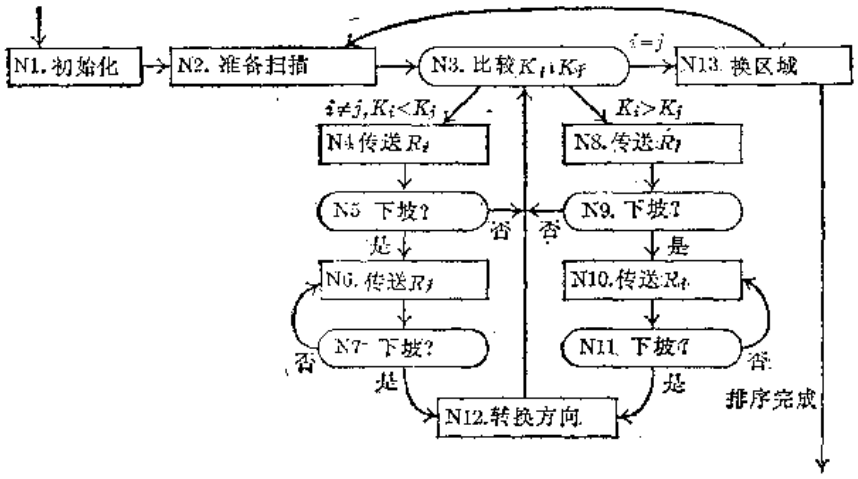


图30　合并排序

N1.〔初始化〕　置 $s \leftarrow 0$（当 $s=0$ 时，我们把记录从 $(R_1, \cdots, R_N)$ 区域传送到 $(R_{N+1}, \cdots, R_{2N})$ 区域；当 $s=1$ 时，以相反方向进行。）

N2.〔准备进行扫描〕　如果 $s=0$，则置 $i \leftarrow 1$，$j \leftarrow N$，$k \leftarrow N+1$，$l \leftarrow 2N$；如果 $s=1$，则置 $i \leftarrow N+1$，$j \leftarrow 2N$，$k \leftarrow 1$，$l \leftarrow N$（变量 i，j，k，l 指向正在读的“源文件”和正在写的“目标文件”的当前位置）。置 $d \leftarrow 1$，$f \leftarrow 1$（变量 d 给出输出的当前方向；如果将来还要进行扫描，则置 f 为 0）。

N3.〔比较 $K_i:K_j$〕　如果 $K_i > K_j$，则转到步骤 N8。如果 $i=j$，则置 $R_k \leftarrow R_i$，并转到 N13。

N4.〔传送 R_i〕（步骤 N4-N7 类似于算法M的步骤 M3-M4）置 $R_k \leftarrow R_i$，$k \leftarrow k+d$。

N5.〔下坡？〕 i 加 1。然后如果 $K_{i-1} \leqslant K_i$，则返回到步骤 N3。

N6.〔传送 R_j〕 置 $R_k \leftarrow R_j$，$k \leftarrow k+d$。

N7.〔下坡？〕 j 减 1。然后如果 $K_{j+1} \leqslant K_j$，则返回步骤 N6；否则转到步骤 N12。

N8.〔传送 R_j〕（步骤 N8-N11 对偶于步骤 N4-N7）置 $R_k \leftarrow R_j$，$k \leftarrow k+d$。

N9.〔下坡？〕 j 减 1。然后如果 $K_{j+1} \leqslant K_j$，则返回到步骤 N3。

N10.〔传送 R_i〕 置 $R_k \leftarrow R_i$，$k \leftarrow k+d$。

N11.〔下坡？〕 i 增加 1，然后如果 $K_{i-1} \leqslant K_i$，则返回到步骤 N10。

N12.〔转换方向〕 置 $f \leftarrow 0$，$d \leftarrow -d$，并交换 $k \leftrightarrow l$。返回到步骤 N3。

N13.〔换区域〕 如果 $f=0$，则置 $s \leftarrow s-1$，并返回步骤 N2。否则排序完成：如果 $s=0$，则置 $(R_1, \cdots, R_N) \leftarrow (R_{N+1}, \cdots, R_{2N})$。（如果有时把 $(R_{N+1}, \cdots, R_{2N})$ 作为输出是可接受的，则不必执行后一复写操作。）

这个算法包含一个巧妙的特性，将在习题 5 中说明它。

在 MIX 上把算法 N 编成程序是不困难的，但是我们无须构造整个程序，就可以推导它的特性的基本事实。在随机条件下，输入中的递增路段数大约是 $\frac{1}{2}N$，因为 $K_i > K_{i+1}$ 的概率是 $\frac{1}{2}$；在稍微不同的假设下，关于路段数的详细信息已经在 5.1.3 节中导出。每次扫描减少一半的路段数（对于象习题 6 那样的不寻常情况除外）。所以扫描数通常将大约是 $\log_2 N$。每次扫描都需要传送全部 N 个记录，而且按习题 2，大部分时间都花费在步骤 N3、N4、N5、N8、N9 上。如果假定键相等的概率很低，我们可以估算内部循环的时间如下：

步骤	操作	时间
N3	CMPA, JG, JE	$3.5u$
或者 N4	STA, INC	$3u$
或者 N5	INC, LDA, CMPA, JGE	$6u$
或 N8	STX, INC	$3u$
或 N9	DEC, LDX, CMPX, JGE	$6u$

于是在每次扫描中对于每个记录大约花费 $12.5u$，在平常情况和最坏情况下，总共的运行时间都渐近于 $12.5N\log_2 N$。这比快速排序的平均时间慢些，而就其花费两倍多的存储空间来说，不应认为它比堆排序好，因为程序 5.2.3H 的渐近运行时间是 $16N\log_2 N$。

在算法N中完全通过“下坡”来确定路段之间的边界线。这可能有优点，即一个主要是递增或主要是递减次序的输入文件可以非常快地被处理。但它减慢了主循环的计算。也可以不去判断下坡，而是人为地决定路段的长度，就说在输入中所有路段的长度都是 1，在第一次扫描之后（最后的路段可能除外）所有路段的长度都是 2，…，在 k 次扫描之后所有路段（可能除开最后的路段）的长度都是 2^k。相对于算法N中的“自然”合并而言，称这为直接两路合并。

直接两路合并非常类似于算法N，而且它实质上有同样的流程图；但区别还是不小的，因而值得再次写出整个算法：

算法S（直接两路合并排序） 如同在算法N中一样，使用两个存储区域对记录 $R_1, \cdots, R_N$ 排序。

S1.〔初始化〕 置 $s \leftarrow 0$，$p \leftarrow 1$。（关于变量 s，i，j，k，l，d 的意义见算法N。这里 p 表示在当前的扫描中有待合并的递增路段的大小；q 和 r 记住在一个路段中未合并的项目数。）

S2.〔准备进行扫描〕 如果 $s=0$，则置 $i \leftarrow 1$，$j \leftarrow N$，$k \leftarrow N$，$l \leftarrow 2N+1$；如果 $s=1$，则置 $i \leftarrow N+1$，$j \leftarrow 2N$，$k \leftarrow 0$，$l \leftarrow N+1$。然后置 $d \leftarrow 1$，$q \leftarrow p$，$r \leftarrow p$。

S3.〔比较 $K_i:K_j$〕 如果 $K_i > K_j$，则转到步骤S8。

S4.〔传送 R_i〕 置 $k \leftarrow k+d$，$R_k \leftarrow R_i$。

S5.〔路段结束？〕 置 $i \leftarrow i+1$，$q \leftarrow q-1$。如果 $q>0$，则转回到步骤S3。

S6.〔传送 R_j〕 置 $k \leftarrow k+d$。然后如果 $k=l$，则转到步骤S13；否则置 $R_k \leftarrow R_j$。

S7.〔路段结束？〕 置 $j \leftarrow j-1$，$r=r-1$。如果 $r>0$，则转回到步骤S6；否则转向S12。

S8.〔传送 R_j〕 置 $k \leftarrow k+d$，$R_k \leftarrow R_j$。

S9.〔路段结束？〕 置 $j \leftarrow j-1$，$r \leftarrow r-1$。如果 $r>0$，则转回步骤S3。

S10.〔传送 R_i〕 置 $k \leftarrow k+d$。然后如果 $k=l$，则转到步骤S13；否则置 $R_k \leftarrow R_i$。

S11.〔路段结束？〕 置 $i \leftarrow i+1$，$q \leftarrow q-1$。如果 $q>0$，则转回到步骤S10。

S12.〔转换方向〕 置 $q \leftarrow p$，$r \leftarrow p$，$d \leftarrow -d$，并交换 $k \leftrightarrow l$。返回到步骤S3。

S13.〔换区域〕 置 $p \leftarrow p+p$。如果 $p<N$，则置 $s \leftarrow 1-s$，并返回到S2。否则排序完成；如果 $s=0$，则置

$$(R_1, \cdots, R_N) \leftarrow (R_{N+1}, \cdots, R_{2N})$$

（当且仅当 $\lceil \log_2 N \rceil$ 是奇数时，才执行后一个复写操作，而不管输入的分布如何。因此，有可能事先预测排序输出的位置，从而往往就免去了复写。）

作为本算法的例子，见表2。稍微使人惊奇的是，当 N 不是2的一个乘方时，这个方法工作得很好；正在合并的路段不全都是长度为 2^k 的，对于这些例外情况却还无明文规定（见习题8）！以前的下坡判断已经被 q 或 r 的减值运算以及结果是否为0的判断所代替；这把MIX的渐近运行时间减少到 $11N\log_2 N$ 单位，比起我们由算法N所能达到的稍微快些。

表2 直接两路合并排序

503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
503	703	512	677	509	908	426	897	653	275	170	154	612	061	765	087
087	503	703	765	154	170	509	908	897	653	426	275	677	612	512	061
061	087	503	512	612	677	703	765	908	897	653	509	426	275	170	154
061	087	154	179	275	426	503	509	512	612	653	677	703	765	897	908

实际上，把算法 S 同直接插入结合起来是值得的，我们可以利用直接插入，代替算法 S 的头四趟扫描，来对16个项目的小组排序，从而避免短的合并造成的相当浪费的簿记操作。如在快速排序情况下所看到的，这种多个方法的结合并不影响渐近的运行时间，但它仍然不失为相当大的改进。

现在从数据结构的观点来研究算法N和S。为什么需要$2N$个记录单元而不是N个呢？原因是比较简单的：我们处理的是可变大小的四个表（对于每次扫描要两个“源”表和两个“目标”表）；而对于每一对顺序地分配的表，我们使用的是2.2.2节中所讨论的标准的“一起生长”的思想。但是存储空间的一半总是没有用上，稍事考虑就看出，实际上应该对这四个表都使用链接分配。如果对N个记录的每一个都加上一个链接场，利用简单的链接操作而全然不必移动记录，就可以做到合并算法所需要的每一件事！加N个链接场，一般来说比加另外N个记录所需要的空间要好，而且减少记录的移动也节约时间。因此，我们应当考虑象下述那样的合并算法：

算法L（表合并排序） 假定记录R_1，…，R_N包含键K_1，…，K_N，而且链接场L_1，…，L_N能容纳数$-(N+1)$到$(N+1)$。在文件的开始和结尾的人为记录R_0和R_{N+1}中，有两个辅助的链接场L_0和L_{N+1}。本算法是一个“表排序”，它设置链接场，使得诸记录以递增次序被链接在一起。在排序完成之后，L_0是具有最小键记录的下标；而且对于$1\leqslant k\leqslant N$，L_k是R_k后面的记录下标，如果R_k是具有最大键的记录，则$L_k=0$（见等式5.2.1-9）。

在本算法的过程中，R_0和R_{n+1}作为其子表正被合并的两个线性表的“表头”。一个负的链接表示被排好序的一个子表的结尾；一个零链接表示整个表的结尾。假定$N\geqslant 2$。

记号“$|L_s|\leftarrow p$”指的是“把L_s置成p或$-p$，但要保留L_s以前的符号。”这个操作很适合于MIX，但可惜并不适于大多数计算机；有可能直截了当地修改这个算法，以得到一个对于大多数机器都同样有效的方法。

L1.〔准备两个表〕 置$L_0\leftarrow 1$，$L_{N+1}\leftarrow 2$，对于$1\leqslant i\leqslant N-2$置$L_i\leftarrow -(i+2)$，并置$L_{N-1}\leftarrow L_N\leftarrow 0$。（我们已经分别建立了包含$R_1$，$R_3$，$R_5$，…及$R_2$，$R_4$，$R_6$…的两个表；负的链接指出每个有序的“子表”仅由一个元素组成。利用在初始数据中可能出现的次序，可以用另一种方法执行这一步，见习题12。）

L2.〔开始新的扫描〕 置$s\leftarrow 0$，$t\leftarrow N+1$，$p\leftarrow L_s$，$q\leftarrow L_t$。如果$q=0$，则本算法终止。（在每次扫描期间，p和q都遍历正被合并的表；s通常指向当前子表的最近被处理的记录，而t指向上一次输出的子表的结尾。）

L3.〔比较$K_p:K_q$〕 如果$K_p>K_q$，则转L6。

L4.〔推进p〕 置$|L_s|\leftarrow p$，$s\leftarrow p$，$p\leftarrow L_p$。如果$p>0$，则返回L3。

L5.〔完成子表〕 置$L_s\leftarrow q$，$s\leftarrow t$。然后置$t\leftarrow q$且$q\leftarrow L_q$，一次或多次，直到$q\leqslant 0$为止。最后转向L8。

L6.〔推进q〕（步骤L6和L7对偶于L4和L5） 置$|L_s|\leftarrow q$，$s\leftarrow q$，$q\leftarrow L_q$。如果$q>0$，则返回L3。

L7.〔完成子表〕 置$L_s\leftarrow p$，$s\leftarrow t$。然后置$t\leftarrow p$和$p\leftarrow L_p$，一次或多次，直到$p\leqslant 0$为止。

L8.〔扫描结束？〕（这时，$p\leqslant 0$和$q\leqslant 0$，因为两个指针都分别移动到它们的子表的结尾。）置$p\leftarrow -p$，$q\leftarrow -q$。如果$q=0$，则置$|L_s|\leftarrow p$，$|L_t|\leftarrow 0$并返回步骤L2。否则返回L3。

执行这个算法的一个例子见表3，其中我们可以看到每次遇到步骤L2时的键接情况。

使用习题5.2-12的方法，即可在这个算法的结尾处重新安排记录 R_1，…，R_N，以使它们的键有序。在表合并和稀疏多项式的加法（见算法2.2.4A）之间，有着有趣的类似性。

表 3　表合并排序

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
K_j	—	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703	—
L_j	1	−3	−4	−5	−6	−7	−8	−9	−10	−11	−12	−13	−14	−15	−16	0	0	2
L_j	2	−6	1	−8	3	−10	5	−11	7	−13	9	12	−16	14	0	0	15	4
L_j	4	3	1	−11	2	−13	8	5	7	0	12	10	9	14	16	0	15	6
L_j	4	3	6	7	2	0	8	5	1	14	12	10	13	9	16	0	15	11
L_j	4	12	11	13	2	0	8	5	10	14	1	6	3	9	16	7	15	0

现在来构造算法 L 的一个 MIX 程序，从速度的观点以及空间的观点看表操作是否有利：

程序L（表合并排序）　为方便起见，假定记录都是一个字长的，且 L_j 在单元INPUT＋j 的（0：2）场中，K_j 在（3：5）场中；rI1≡p，rI2≡q，rI3≡s，rI4≡t，rA≡R_q，$N\geqslant 2$。

```
01   L       EQU     0:2                          场名的定义
02   ABSL    EQU     1:2
03   KEY     EQU     3:5
04   START   ENT1    N-2                 1        L1. 准备两个表
05           ENNA    2,1                 N-2
06           STA     INPUT,1(L)          N-2      L_i ← -(i+2)
07           DEC1    1                   N-2
08           J1P     *-3                 N-2      N-2 ≥ i > 0
09           ENTA    1                   1
10           STA     INPUT(L)            1        L_0 ← 1
11           ENTA    2                   1
12           STA     INPUT+N+1(L)        1        L_{N+1} ← 2
13           STZ     INPUT+N-1(L)        1        L_{N-1} ← 0
14           STZ     INPUT+N(L)          1        L_N ← 0
15           JMP     L2                  1        转 L2（见行43）
16   L3Q     LDA     INPUT,2             C''+B'   L3. 比较 K_p:K_q
17   L3P     CMPA    INPUT,1(KEY)        C
18           JL      L6                  C        如果 K_q < K_p 则转 L6
19   L4      ST1     INPUT,3(ABSL)       C'       L4. 推进 p. |L_s| ← p
20           ENT3    0,1                 C'       s ← p
21           LD1     INPUT,1(L)          C'       p ← L_p
22           J1P     L3P                 C'       如果 p > 0 则转 L3
23   L5      ST2     INPUT,3(L)          B'       L5. 完成子表，L_s ← q
24           ENT3    0,4                 B'       s ← t
25           ENT4    0,2                 D'       t ← q
26           LD2     INPUT,2(L)          D'       q ← L_q
```

27		J2P	* -2	D'	如果 $q>0$，则重复
28		JMP	L8	B'	转L8
29	L6	ST2	INPUT, 3(ABSL)	C''	L6. 推进 q，$[L_s]\leftarrow q$
30		ENT3	0, 2	C''	$s\leftarrow q$
31		LD2	INPUT, 2(L)	C''	$q\leftarrow L_q$
32		J2P	L3Q	C''	如果 $q>0$ 则转L3
33	L7	ST1	INPUT, 3(L)	B''	L7. 完成子表，$L_s\leftarrow p$
34		ENT3	0, 4	B''	$s\leftarrow t$
35		ENT4	0, 1	D''	$t\leftarrow p$
36		LD1	INPUT, 1(L)	D''	$p\leftarrow L_p$
37		J1P	* -2	D''	如果 $p>0$ 则重复
38	L8	ENN1	0, 1	B	L8. 扫描完成？ $p\leftarrow -p$
39		ENN2	0, 2	B	$q\leftarrow -q$
40		J2NZ	L3Q	B	如果 $q\neq 0$ 则转L3
41		ST1	INPUT, 3(ABSL)	A	$[L_s]\leftarrow p$
42		STZ	INPUT, 4(ABSL)	A	$[L_t]\leftarrow 0$
43	L2	ENT3	0	$A+1$	L2. 开始新的扫描， $s\leftarrow 0$
44		ENT4	N + 1	$A+1$	$t\leftarrow N+1$
45		LD1	INPUT(L)	$A+1$	$p\leftarrow L_s$
46		LD2	INPUT + N + 1(L)	$A-1$	$q\leftarrow L_t$
47		J2NZ	L3Q	$A+1$	如果 $q\neq 0$ 则转L3

利用在这以前已见过多次的技术（见习题13、14），就可以推导这个程序的运行时间：平均说来，它近似于 $(10N\log_2 N+4.92N)u$，并具有阶为$\sqrt{N}$的很小的标准离差。习题15说明，以较长的程序为代价，运行时间还可以少到大约$9N\log_2 N$。

因此，在内部合并时，链接存储技术显然胜过顺序分配：只需要较小的存储空间，而程序运行要快大约10%到20%。L.J.伍德鲁姆（L.J.Woodrum）〔*IBM Systems J*.8(1969)，189-203〕和A.D.伍德尔（A.D.Woodall）〔*Comp*，*J*.13(1970).110-111〕已经发表了类似的算法。

习题

1.〔20〕 把算法M推广到输入文件 $x_{i_1}\leqslant\cdots\leqslant x_{i_{m_i}}$ 的一个 k 路合并，其中 $i=1, 2, \cdots, k$。

2.〔*M*24〕 假定在 n 个 y 之中插入 m 个 x 的$\binom{m+n}{m}$种安排，每一种都是同等可能的，试求在算法M中步骤M2被执行次数的平均值和标准离差。问这个量的极大值和极小值是多少？

3.〔*M*20〕（更新） 给定记录 $R_1, \cdots, R_M$ 和 $R'_1, \cdots, R'_N$，它们的键是不同的和有序的，就是说 $K_1<\cdots<K_M$和$K'_1<\cdots<K'_N$，如果我们要求：当第一个文件的记录R_i的键也出现于第二个文件中时，则抛弃R_i，试说明如何修改算法M以得到一个合并的文件。

4.〔21〕 正文中已经注意到，合并排序可以认为是插入排序的一个推广，试说明合并排序如同图23中所指出的那样，也同树选择排序密切相关。

▶5.〔21〕 证明在步骤N6或N10中，i 决不会等于 j（因此在这些步骤中都不必判

断是否应转向N13)。

6.〔22〕 求{1，2，…，16}的一个排列K_1，K_2，…，K_{16}，使得

$K_2>K_3$，$K_4>K_5$，$K_6>K_7$，$K_8>K_9$，$K_{10}<K_{11}$，$K_{12}<K_{13}$，$K_{14}<K_{15}$

而且算法N仅仅用两次扫描完成对这个文件的排序。(因为有八个或更多的路段，预期在头一次扫描之后至少有四个路段，在第二次扫描之后有两个路段，而通常至少经过三次扫描才能完成排序。如何能通过仅仅两次扫描来完成？)

7.〔16〕 给出算法S所需要的精确扫描次数的一个公式（作为N的一个函数)。

8.〔12〕 在算法S中，让变量q和r表示当前正在处理的路段中尚未合并的诸元素的长度，q和r两者开始都等于p，但路段并不都是这个长度的。问这怎么能行得通呢？

9.〔24〕 写出算法S的一个MIX程序，利用类似于程序L中的量A，B'，B''，C'，…确定指令频率。

10.〔25〕〔D.A.贝尔（D.A.Bell)〕 说明顺序分配的直接两路合并可以通过至多$\frac{3}{2}N$个存储单元完成，而不是象在算法S中那样需要$2N$个。

11.〔21〕 算法L是一个“稳定的”排序方法吗？

▶12.〔22〕 利用开始时存在的递增路段，修正算法L的步骤L1，使得两路合并是“自然的”。(特别是，如果输入已经有序，则在你的步骤L1进行之后，步骤L2应立即终止这个算法)。

▶13.〔M34〕 以这一章中其它分析的风格，给出对程序L的平均运行时间的分析：解释量A，B，B'，…，并且说明怎样计算它们的精确平均值。为对表3中的16个数排序，程序L要用多长时间？

14.〔M24〕 设N的二进表示是$2^{e_1}+2^{e_2}+\cdots+2^{e_t}$，其中$e_1>e_2>\cdots>e_t\geqslant 0$，$t\geqslant 1$。证明由算法L执行的键比较极大次数是$1-2^{e_t}+\sum_{1\leqslant k\leqslant t}(e_k+k-1)2^{e_k}$。

15.〔20〕 算法L的手工模拟揭示，它有时作一些多余的动作；步骤L4和L6中的赋值$|L_s|\leftarrow p$，$|L_s|\leftarrow q$有一半的次数是不必要的，因为在每次由步骤L4（或L6）返回到L3时，我们有$L_s=p$（或q)。如何能改进程序L，来消除这些多余的赋值？

16.〔28〕 设计一个象算法L那样的表合并算法，但它以三路合并为基础。

17.〔20〕（J.麦卡锡） 设N的二进表示如同习题14中那样，并假定给了我们N个记录，这些记录分布在t个有序子文件中，其大小分别为2^{e_1}，2^{e_2}，…，2^{e_t}。试说明当附加一个新的第（$N+1$）个记录且$N\leftarrow N+1$时，怎样维持这一事态。(得到的算法可以叫做一个“联机”合并排序。)

18.〔40〕〔M.A.克朗洛德（M.A.Kronrod)〕 给定仅含两个路段的N个记录的一个文件

$$K_1\leqslant\cdots\leqslant K_M \text{ 和 } K_{M+1}\leqslant\cdots\leqslant K_N$$

有可能在一个随机存取存储器中用$O(N)$个操作对这个文件排序吗？而且不论M和N的大小如何，只许使用少量固定的附加存储空间（本节描述的所有合并算法都使用与N成比例的额外存储空间)。

19.〔26〕 考虑当$n=5$时，如图31中所示的具有n个“栈”的铁路转辙器网络；这样一个网络有同N次扫描排序算法有关的某些问题。在习题2.2.1-2到2.2.1-5中我们考虑过

一个栈的网络。如果N列火车从右边进入，则我们发现，在一个栈的情况下，这些列车的$N!$个排列中仅有很少几种在左边出现。

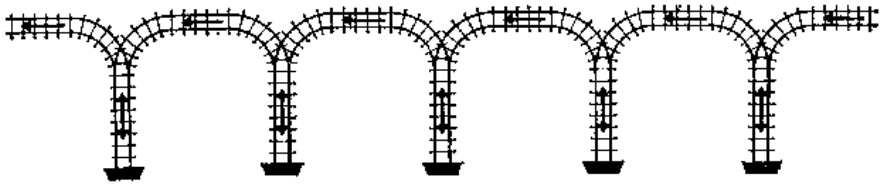


图31 具有五个“栈”的铁路网络

在n个栈的网络中，假定有2^n列车从右边进入。试证明这些列车的$2^n!$种可能的排列中的每一种，通过适当的操作序列都可在左边得到。(必要时每个栈都足够大，以便能容纳所有的列车。)

20.〔47〕 在习题2.2.1-4的记号下，通过n个栈的铁路网络至多能产生N个元素的a_N^n个排列；因此为得到所有$N!$个排列所需要的栈数至少是$\log_2 N!/\log_2 a_N \approx \log_4 N$。习题19则说明至多需要$\lceil \log_2 N \rceil$个栈。当$n \to \infty$时，所需要栈数的真实增长率是多少？

21.〔23〕〔A.J.史密斯（A.J.Smith)〕 说明怎样扩充算法L，使得除排序外，它还计算输入排列中出现的反序个数。

5.2.5 通过分布进行排序

我们现在要讲一类有趣的排序方法，在5.4.7中将看出它实质上恰是合并的对立面。熟悉穿孔卡片设备的读者对于在卡片排序机上使用的，以键的数字为基础的有效过程是很清楚的；同样的思想也可用来进行计算机的程序设计，而且它一般地被称作“基数排序”、“数字排序”或“钱袋排序”。

假设要对52张扑克牌进行排序。我们可以定义

$$A < 2 < 3 < 4 < 5 < 6 < 7 < 8 < 9 < 10 < J < Q < K$$

作为面值的一个排序，而对于花色，我们可以定义

$$\clubsuit < \diamondsuit < \heartsuit < \spadesuit$$

如果（i）一张牌的花色小于另一张牌的花色，或（ii）花色相同，但它的面值较另一张牌小（这是对象的有序偶之间的一个字典编辑次序的特殊情况，参考习题5-2。)，则我们称这张牌是位于另一张牌之前。于是

$$A\clubsuit < 2\clubsuit < \cdots < K\clubsuit < A\diamondsuit < \cdots < Q\spadesuit < K\spadesuit$$

可以用已经讨论过的任何方法来对这些牌进行排序；人们通常使用多少类似于基数交换思想的一种技术。对这些纸牌排序的一种自然的方法是先按它们的花色分成四堆，然后拨弄每一堆直到它们变成有序的为止。

但是还有一种更好的方式来玩这圈牌！首先把这些牌仰面分成13堆，每种面值一堆。然后通过把A牌放在下边，2朝上放在它们的上面，然后3放在上面，等等，最后把K面

朝上放在上边，来收集这些堆。翻转这些牌使它们面朝下并且再次分堆，这次按四种花色分成四堆。把这样得到的四堆放在一起，黑梅花放在底下，然后方块、红桃和黑桃，这副牌就成为完全有序的了。

同样的思想也可应用于数和字母数据的排序。为什么它有效呢？因为（在扑克牌的例子中）如果两张牌在最后分堆时位于不同的堆中，则它们有不同的花色，所以具有较低花色的一张在最下面。但是如果两张纸牌有相同的花色（因此它们分在同一堆中），则由于前边的排序它们已处于适当的次序了。换言之，当我们第二序扫描这些纸牌时，对于这四堆的每一堆，面值将处于递增的次序。同样的证明可被抽象出来，以指出任何字典编辑次序也可以用这种方式进行排序；关于其细节，见本章开始处习题 5-2 的答案。

刚才描述的排序方法并非一目了然，也不清楚是谁最先发现它如此有效。1923 年，IBM的造表机器公司分部发表的题为《简化的存货清单》的19页小册子，介绍了在他们的电动排序机上形成乘积和的一种有趣的数字计划方法：例如，假设我们要用23～25列上穿孔的数来乘 1 ～10列上穿孔的数，并对大量的卡片把这些乘积加起来。可以首先对第25列进行排序，然后使用造表机求量 a_1，a_2，…，a_9，其中 a_k 是对于在第 25 列中有 k 的所有卡片的 1 ～10列求和的总数。然后可以对第24列进行排序，求类似的总和 b_1，b_2，…，b_9；然后对第23列，得到 c_1，c_2，…，c_9。容易看出所求的乘积和是

$$a_1+2a_2+\cdots+9a_9+10b_1+20b_2+\cdots+90b_9+100c_1+200c_2+\cdots+900c_9$$

这种穿孔卡片的造表方法自然导致发现首先对最低数字排序的基数排序方法，所以它大概后来已为机器的操作员所熟知。这个排序原理的头一个公开的参考文献出现在 L.J.科姆里（L.J.Comrie）关于穿孔卡片设备的早期讨论中〔*Transactions of the Office Machinery Users' Assoc., Ltd* (1929), 25-37, esp.p.28〕。

为了处理在一台计算机内部的基数排序，我们必须决定如何处理这些堆。假定有 M 个堆，我们可以留出 M 个存储区域，把来自输入区域的每个记录移到它的适当的堆区域中。但这是难以令人满意的，因为每个区域都必须足够地大以容纳 N 个项目，而这将要求 $(M+1)N$ 个记录的空间。这样一个过分的存储要求当初曾使大多数人都否定在计算机内部作基数排序的想法。直到 H.H.西沃德（H.H.Seward）〔Master's thesis, M.I.T. Digital Computer Laboratory Report R-232(Cambridge, Mass; 1954), 25-28〕指出，可以仅用 $2N$ 个记录的区域和 M 个计数场，来达到同样的效果，情况才有所改变。只需对这些数据进行初步扫描，以计算 M 堆中的每一堆将有多少个元素；这确切地告诉我们怎样为这些堆分配存储。在“分布排序”算法5.2D 中，已经利用了同样的思想。

于是基数排序可以按如下方式进行：首先按键的最低位数字（在基数 M 的记法下）进行分布排序，把取自输入区域的记录移到一个辅助区域。然后对于下一个最低位，进行另一次分布排序，把记录移回到原来的输入区域，等等，直到最后（对最高位）的扫描把所有的记录排成所希望的次序为止。

如果有一台具有12位数字键的十进制计算机，并且 N 相当大，则可以选择 $M=1000$（考虑三个十进数字作为一个1000进制的数字）；排序将在四次扫描中完成，而不论 N 的大小如何。类似地，如果有一台二进制计算机和一个40位的键，则可以置 $M=1024$，并且在四次扫描中完成排序。实际上，每次“扫描”由三部分组成（计数，分配，移动）；弗兰德

〔JACM 3 (1956)151〕已经建议，在第 k 次扫描中移动记录的同时，为第 $k+1$ 次扫描累加计数，这样，就能以多用 M 个存储单元为代价，把扫描中的两件事情组合在一起。

表 1 说明，对于 $M=10$，这样一个基数排序如何被应用于我们的16个例数。一般来说，对于这样小的 N 基数排序没有什么用，举这样一个小例子的目的是要说明这个方法的充分性而不是其效率。

表1　基 数 排 序

输入区域内容：	503	087	512	061	908	170	897	275	653	426	154	509	612	677	765	703
对于个位数字分布的计数：					1	1	2	3	1	2	1	3	1	1		
以这些计数为基础的存储分配：					1	2	4	7	8	10	11	14	15	16		
辅助区域的内容：	170	061	512	612	503	653	703	154	275	765	426	087	897	677	908	509
对于十位数字的分布的计数：					4	2	1	0	0	2	2	3	1	1		
以这些计数为基础的存储分配：					4	6	7	7	7	9	11	14	15	16		
输入区域的内容：	503	703	908	509	512	612	426	653	154	061	765	170	275	677	087	897
对于百位数字的分布的计数：					2	2	1	0	1	3	3	2	1	1		
以这些计数为基础的存储分配：					2	4	5	5	6	9	12	14	15	16		
辅助区域内容：	061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908

然而，一个机灵的“现代化”的读者将注意到，对于存储分布进行数字计数的整个思想，都受到顺序数据表示的“旧”思想的束缚；我们知道，**链接**分配是特别设计用来处理可变大小的多重表的，所以排序自然应选择链接的数据结构。由于串行地遍历每一堆，因此需要的仅仅是从每个项目到它的后继者的一条单链。而且，决不需要移动这些记录，仅需调整这些链接并轻快地向下通过这个表。需要的存储数量是 $(1+\epsilon)N+2\epsilon M$ 个记录，其中 ϵ 是一个链接场所取的空间数量。这个过程的形式细节是相当有趣的，因为它提供了一个典型的数据结构操作的卓越例子，它综合了顺序的和链接的分配：

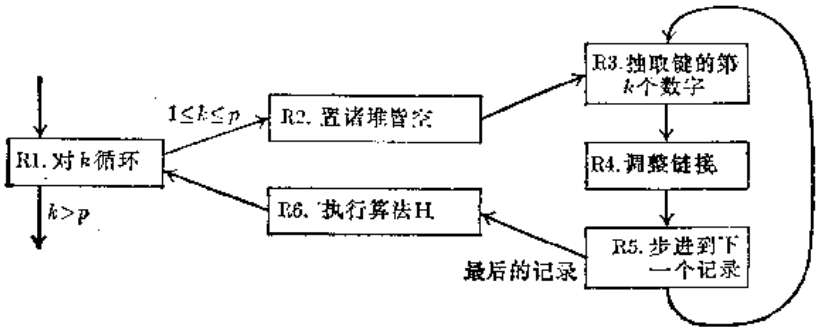


图32　基数表排序

算法R（基数表排序）　假定记录 $R_1 \cdots R_N$ 中的每个都包含一个LINK 场，它们的键是 p 元组

$$(a_p,\ \cdots,\ a_2,\ a_1),\quad 0 \leqslant a_i < M$$

其中，次序按字典编辑的顺序定义，即

$$(a_p,\ \cdots,\ a_2,\ a_1) < (b_p,\ \cdots,\ b_2,\ b_1)$$

当且仅当对某个 j，$1 \leqslant j \leqslant P$，我们有

$$\text{对所有 } i > j,\ a_i = b_i \text{ 但 } a_j < b_j$$

特别是，键可以想成是以基数 M 的表示法写成的数

$$a_p M^{p-1} + \cdots + a_2 M + a_1$$

在这种情况下，字典编辑次序相当于非负数的正常次序。键也可以是字母串，等等。

排序使用M“堆”记录，以模拟一个卡片排序机的动作方式。诸堆事实上都是第二章意义上的队，因为我们把它们链接在一起，使得它们以先进先出的方式被遍历。对于每个堆，都有两个指针变量 TOP〔i〕和 BOTM〔i〕，$0 \leqslant i \leqslant M$，而且如同在第二章那样，我们假定

$$\mathrm{LINK(LOC(BOTM([}i\mathrm{])))} \equiv \mathrm{BOTM[}i\mathrm{]}$$

R1.〔对k进行循环〕 开始时，置$p \leftarrow \mathrm{LOC}(R_N)$，这是指向最后记录的一个指针。然后对于$k=1$，2，…，$p$执行步骤 R2 到 R6（步骤 R2 到 R6 组成一趟“扫描”），然后这个算法终止，以p指向具有最小键的记录，LINK(p) 指向具有次小键的记录，然后 LINK(LINK(P))，等等；在最后记录中的 LINK 将是 Λ。

R2.〔置诸堆皆空〕 对于$0 \leqslant i \leqslant M$，置 TOP〔$i$〕←LOC(BOTM〔$i$〕) 和 BOTM〔$i$〕←Λ。

R3.〔抽取键的第k位数字〕 命由P访问的记录中的键 KEY(P) 是 (a_p，…，a_2，a_1)；置$i \leftarrow a_k$，a_k是这个键的第k个最低位。

R4.〔调整链接〕 置 LINK(TOP〔i〕)←P，然后置 TOP〔i〕←P。

R5.〔步进到下一个记录〕 如果$k=1$（头一次扫描）且如果对于某个$j \neq 1$，$\mathrm{P}=\mathrm{LOC}(R_j)$，则置$\mathrm{P} \leftarrow \mathrm{LOC}(R_{j-1})$并返回到 R3。如果$k>1$（随后的扫描），则置P←LINK(P)，如果P≠Λ则返回到 R3。

R6.〔执行算法H〕（现在已把所有的元素都分配到诸堆上。）实施以下的算法 H，它把各堆“钩在一起”，形成一个表，以准备进行下次扫描。然后，置$P \leftarrow$-BOTM〔0〕，这是指向已钩起来的表的头一个元素的指针。（见习题 3）

算法 H（队的挂钩）给定M个队，它们已各自按照算法 R 的约定链接，本算法至多调整M个链接来建立一个单队，以 BOTM〔0〕指向头一个元素，并置堆 0 在堆 1 之前，……，在堆$M-1$之前。

H1.〔初始化〕 置$i \leftarrow 0$。

H2.〔指向堆的顶部〕 置$P \leftarrow$TOP〔i〕。

H3.〔下一堆〕 i加 1，如果$i=M$，则置 LINK(P)←Λ，并终止此算法。

H4.〔堆为空?〕 如果 BOTM〔i〕=Λ，则回到 H3。

H5.〔把诸堆链在一起〕 置 LINK(P)←BOTM〔i〕，返回到 H2。

图 33 示出了以$M=10$对我们的 16 个例数进行排序的情况，共三次扫描，图中给出每次扫描后诸堆栈的内容。步骤 R3 和 R5 是逐趟变形的，只要找到了处理这种变形的适当途径，则算法 R 的 MIX 程序是非常易于编制的。下列程序通过重叠两条指令，无须牺牲内部循环中的任何速度即可解决此问题。注意 TOP〔i〕和 BOTM〔i〕可以组装进同一个字中。

程序 R（基数表排序） 假定单元 INPUT+1到 INPUT+N 中的输入有$P=3$个分量(a_3，a_2，a_1)，分别地存储在（1:1），（2:2）和（3:3）场中（这就假定了M是小于或等于 MIX 的字节大小的）。每个记录的（4:5）场是它的 LINK。对于$0 \leqslant i < M$，命 TOP〔i〕≡PILES+i（1:2）和 BOTM〔i〕≡PILES+i（4:5）。不难把单元 INPUT 链接起

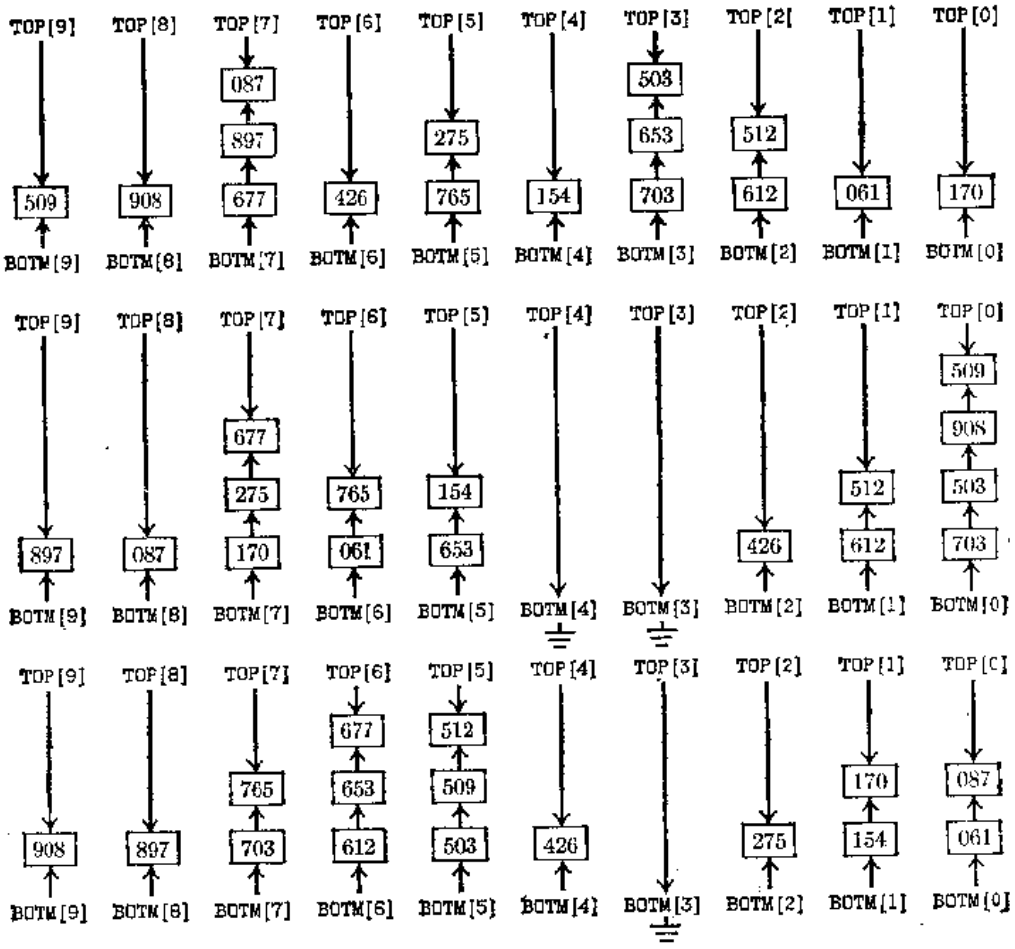


图33 使用链接分配的基数排序：在每次扫描之后十个堆的内容

来，使得 LOC(BOTM[i])=PILES+ i −INPUT；因此，为了避免负链接，我们要求 PILES 表处于比 INPUT 表较高的单元。变址寄存器被赋值如下：rI1≡P，rI2≡ i ，rI3≡ $3-k$ ，rI4≡TOP[i]；在算法H期间，rI2≡ $i-M$ 。

01	LINK	EQU	4:5		
02	TOP	EQU	1:2		
03	START	ENT1	N	1	*R*1. 对 k 进行循环，P←LOC(R_N)
04		ENT3	2	1	k←1
05	2H	ENT2	M − 1	3	*R*2. 置诸堆皆空
06		ENTA	PILES-INPUT, 2	3*M*	LOC(BOTM[i])
07		STA	PILES, 2(TOP)	3*M*	→TOP[i]
08		STZ	PILES, 2(LINK)	3*M*	BOTM[i]←Λ
09		DEC2	1	3*M*	
10		J2NN	* −4	3*M*	$M>i\geqslant 0$
11		LDA	R3SW, 3	3	
12		STA	3F	3	修改第 k 次扫描的指令
13		LDA	R5SW, 3	3	
14		STA	5F	3	

15	3H	[LD2	INPUT,1(3:3)]		R3. 抽取键的第 k 位数字
16	4H	LD4	PILES,2(TOP)	$3N$	R4. 调整链接
17		ST1	INPUT,4(LINK)	$3N$	LINK(TOP[i])←P
18		ST1	PILES,2(TOP)	$3N$	TOP[i]←P
19	5H	[DEC1	1]		R5. 步进到下一个记录
20		J1NZ	3B	$3N$	如果扫描结束转到R3
21	6H	ENN2	M	3	R6. 执行算法H
22		JMP	7F	3	以 i ← 0 转到H2
23	R3SW	LD2	INPUT,1(1:1)	N	当 k = 3时，R3的指令
24		LD2	INPUT,1(2:2)	N	当 k = 2时，R3的指令
25		LD2	INPUT,1(3:3)	N	当 k = 1时，R3的指令
26	R5SW	LD1	INPUT,1(LINK)	N	当 k = 3时，R5的指令
27		LD1	INPUT,1(LINK)	N	当 k = 2时，R5的指令
28		DEC1	1	N	当 k = 1时，R5的指令
29	9H	LDA	PILES+M,2(LINK)	$3M-3$	H4. 堆为空?
30		JAZ	8F	$3M-3$	如果BOTM[i]=A，则转到H3
31		STA	INPUT,1(LINK)	$3M-3-E$	H5. 把诸堆链在一起 LINK(P)← BOTM[i]
32	7H	LD1	PILES+M,2(TOP)	$3M-E$	H2. 指向堆的顶部
33	8H	INC2	1	$3M$	H3. 下一堆，$i \leftarrow i+1$
34		J2NZ	9B	$3M$	如果 $i \neq M$ 转到H4
35		STZ	INPUT,1(LINK)	3	LINK(P)←A
36		LD1	PILES(LINK)	3	P←BOTM[0]
37		DEC3	1	3	
38		J3NN	2B	3	$1 \leqslant k \leqslant 3$

程序R的运行时间是 $32N+48M+38-4E$，其中 N 是输入的记录个数，M 是基数(堆数)，E 是出现的空堆个数。这非常适合于同以类似假定为基础的其它程序（程序5.2.1M，5.2.4L）进行对比。这个程序的 p 趟变形● 将花费 $(11p-1)N+O(pM)$ 个单位时间；计时中的关键因素是内部循环，它包含五次对存储的访问和一次转移。在一台典型的计算机上，有 $M=b^r$ 和 $p=\lceil i/r \rceil$，其中 i 是在这些键中基数 b 的数字个数；增加 r 将减少 p，所以这个公式可用来确定 r "最好的"值。

计时中唯一的变量是 E，即在步骤H4中发现的空堆个数。如果认为基数为 M 的 M^N 个数字序列中的每一个都是同等可能的，则从3.3.2D中对于"扑克检验"的研究知道，每次扫描有 $M-r$ 个空堆的概率是

$$\frac{M(M-1)\cdots(M-r+1)}{M^N}\left\{ {N \atop r} \right\}$$

其中 $\left\{ {N \atop r} \right\}$ 是第二类斯特林数。由习题5

$$E=\left(\min\ \max(M-N,\ 0)p,\ \mathrm{ave}M\left(1-\frac{1}{M}\right)^N p,\ \max(M-1)p\right)$$

● 本程序限于 $p=3$，推广到一般的 p 就是"p 趟变形"。——译注

"流水线"或"吞嚼"数据的计算机近年来与日俱增，这些机器有多个运算器以及"先行控制"线路，使得存储的访问可以和计算高度重叠，但是它们的效率在出现条件转移指令时显著地下降，除非这个转移几乎总是走同一条路。一个基数排序的内部循环，很适合于这种机器，因为它是典型的"吞嚼"数据形式的一个直接迭代计算。因此，假定N不太小而键不太长的话，比起在这样的机器上的任何其它已知的内部排序方法来，基数排序通常要更为有效。

当然，基数排序在键极其长时不是非常有效的。例如，想象要按 80 列键来对卡片排序；在头五列中，很少有两张卡片具相同的键，所以头 75 次扫描起不了多少作用。在对基数交换排序的分析中，当从左边而不是从右边处理键时，已发现并不需要检查键的许多位。因此，重新考虑从最高位数字（MSD）而不是从最低位数字（LSD）开始的基数排序的想法。

我们已经说过，首先排最高位数字的基数方法自然地浮现在眼前；事实上，不难看出，为什么邮局使用这样一种方法来对邮件排序。大量的信件可以排序成按不同地理区域分开的邮包；每一个包包含较少量的信件，这些信件可独立于其它的包而分成越来越细的地理区域（其实，在它们进一步被排序之前，已被传递到更接近于它们的目的地）。这个"分而治之"的原理是十分有吸引力的，而它对于穿孔卡片的排序不特别适用的唯一原因，是它毕竟花费太多的时间去跟非常小的堆打交道。算法R是相对有效的，尽管它首先考虑最低有效位数字，因为决没有多于M个堆，而且它们仅仅需要钩在一起p次。另一方面，利用链接内存，并象算法 5.2.4L 那样以负的链接来表示堆之间的边界，则不难设计首先处理最高有效位的方法（见习题10）。

最好的折衷也许已经由 M. D. 麦克拉伦（M. D. MacLaren）提出来了〔*JACM* 13 (1966)，404-411)〕，他推荐如算法R中那样的首先处理最低位数字的排序，但仅仅应用于最高的那些数字位。这不能完全把文件排好序，但它通常都把文件搞成为非常接近有序的，而仅保留很少的反序；因此可用直接插入来完成。对算法 5.2.1M的分析也可应用于这种情况，所以，如果键是均匀分布的，在对前导的p个数字进行排序之后，将在文件中平均剩下

$$\frac{1}{4}N(N-1)M^{-p}$$

个反序（见等式 5.2.1-14 和习题 5.2.1-38）。麦克拉伦已经算出每排好一个项目所需的存储访问的平均次数，以及M和p的最优选择（假定M是 2 的乘方，以及这些键都是均匀分布的，且$N/M^p \leqslant 0.1$，则对于均匀性的偏离是可以容忍的）是由下表给出的：

$N=$	100	1000	5000	10000	50000	100000
最好的$M=$	32	128	256	512	1024	1024
最好的$p=$	2	2	2	2	2	2
$\bar{\beta}(N)=$	19.3	18.5	18.2	18.2	18.1	18.0

这里，$\bar{\beta}(N)$ 表示每排好一个项目所需的平均存储访问次数；如果取$p=2$和$M>\sqrt{N}$，当$N\to\infty$时它是有界的，则平均的排序时间实际上是$O(N)$而不是$N\log_2 N$阶的。这个方法是对于多重表插入（算法 5.2.1M）的一个改进，后者实质上是$p=1$的情况。对于一个部分地用表排好序的文件的最后重排，习题 12 给出麦克拉伦的一个有趣的方法。

也有可能使用算法 5.2D和习题 5.2-13 的方法来避免链接场，使得除了记录本身所需要的空间外，仅仅需要 $O(\sqrt{N})$ 个存储单元。如果输入记录是均匀分布的，则平均排序时间同N成比例。

习题

▶1.〔20〕 习题 5.2-13 的算法说明，对于仅有N个记录（以及M个计数场）而不是$2N$个记录的区域，怎样来做一个分布排序。这能引出一项对于表 1 中说明的基数排序算法的改进吗？

2.〔13〕 算法R是一个“稳定的”排序方法吗？

3.〔15〕 说明为什么在算法II中，虽然堆 0 可以是空的，但 BOTM〔0〕指向钩起的队中的头一个记录。

▶4.〔23〕 算法R把M个堆链接在一起成为(先进先出)队，试剖析链接诸堆成为一些栈的思想（图 33 中的箭头将向下进行而不是向上，且 BOTM 表是不必要的）。证明：如果诸堆以适当的次序钩在一起，则有可能得到一个有效的排序方法。这能构成一个更简单或一个更快的算法吗？

5.〔M24〕 设 $g_{MN}(z)=\sum p_{MNk}z^k$，其中 p_{MNk} 是在一个随机基数排序扫描把N个元素分成M个堆之后，恰出现有k个空堆的概率。(a) 证明 $g_{M,N+1}(z)=g_{MN}(z)+((1-z)/M)g'_{MN}(z)$。(b) 使用此关系式求出这个概率分布的平均值和方差的简单表达式(作为M和N的一个函数)。

6.〔20〕 为使程序R能对 8 个字节的键而不是三个字节的键排序，对它需要作些什么改动？假定 K_i 的诸最高位字节存储在单元 KEY+ i (1:5) 中，而三个最低位字节就象目前这样存在单元 INPUT+ i (1:3) 中，在进行了这些改动之后，该程序的运行时间是多少？

7.〔20〕 试讨论算法R和基数交换排序（算法 5.2.2R）之间的相似性和区别。

8.〔20〕 正文中讨论的基数排序算法假定，被排序的所有键都是非负的，当这些键是以 2 的补码或 1 的补码记号表示的数时，对诸算法应作什么改动？

9.〔20〕 继续习题 8，当键是以带正负号的量表达的数时，对诸算法应当作些什么改动？

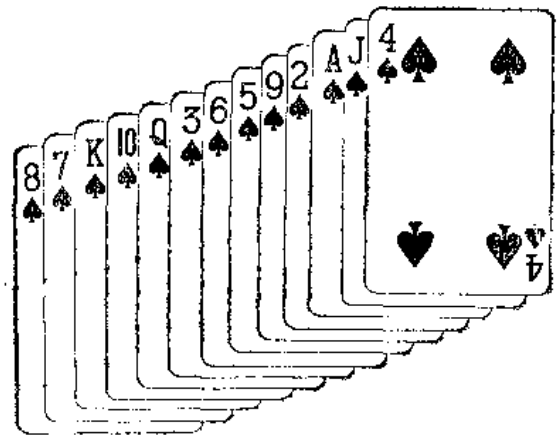
10.〔30〕 设计一个使用链接内存的，有效的首先处理最高位数字的基数排序算法(随着子文件长度的减小，减小M并对非常短的子文件使用一个非基数的方法是明智的)。

11.〔16〕 表 1 中所示的 16 个输入数以 41 个反序开始；在排序完成之后，当然已无剩下的反序。如果省略第一次扫描，仅仅对十位和百位数字进行基数排序，则在这个文件中将出现多少反序？如果省略第一次扫描和第二次扫描，则将出现多少反序？

12.〔24〕 (M.D.麦克拉伦) 假设算法R已应用于实际键的仅仅p个前导数字；于是当我们以链接的次序来读它时，文件已接近于有序了，但是其头p位数字相等的那些键可能是无次序的。试设计一个算法，它重新适当地安排记录，使得它们的键都成为有序的，$K_1\leqslant K_2\leqslant\cdots\leqslant K_N$。〔提示：文件被完全排序的特殊情况出现于习题 5.2-12 的答案中；有可能不影响效率而将它与直接插入组合起来，因为在文件中保留了少量的反序。〕

13.〔40〕 请实现在本节结尾处的正文中提出的内部排序方法，写一个子程序，它用 $O(N)$ 个时间单位对随机数据排序，且仅使用 $O(\sqrt{N})$ 个附加存储单元。

14.〔22〕 扑克牌序列



可以在两次扫描中排序成从顶到底递增的次序 A 2 … J Q K，且只使用两个堆作为中间存储：把这些牌面朝下分成分别含 A 2 9 3 10 和 4 J 5 6 Q K 7 8（从底到顶）的两堆；然后把第二堆放在第一堆的上面，翻过来使这些牌面朝上，把这些牌分成两堆，A 2 3 4 5 6 7 8，9 10 J Q K。把这两堆组合在一起，让它们面朝上，你就大功告成了。

证明上边的纸牌序列不可能在两次扫描中排序成为从顶到底的递减次序 K Q J … 2 A，即使你使用多达三个堆作为中间存储也罢。（处理必须总从纸牌的顶部进行，当它们被处理时，把这些纸牌翻成面朝下。由顶到底相应于图中的自右到左。）

15.〔M25〕 当所有纸牌必须面朝上而不是面朝下地处理时，考虑习题 14 的问题。于是，可用一次扫描把递增次序转换成递减次序。问需要多少次扫描？

> 一旦有分析机存在，它就必将支配将来的科学进程。
>
> 每当在它的帮助下得到任何结果时，问题就出现了——通过什么计算过程可以在最短时间内用同一机器达到这些结果呢？
>
> ——查尔斯·巴贝其
> (Charles Babbage)(1864)

5.3 最优排序

我们已经分析了内部排序的大量方法，现在应转来考虑一个更广泛的问题：什么是进行排序的最好方法？能否对可能达到的极大排序速度给出极限，使得最灵巧的程序员也不能超过它呢？

当然，并没有进行排序的最好方式；必须精确地定义什么叫“最好”，但是，不存在一种最好的方式来定义“最好”。在 4.3.3、4.6.3 和 4.6.4 节中，曾经讨论过算法理论最优性的类似问题，其中考虑了高精度的乘法和多项式计算，在每种情况下，都有必要阐述“最好的”算法的一种相当简单的定义，以便给出使问题得以处理的足够的结构。而且在每种情况下，都遇到了如此困难的因而尚未完全解决的有趣问题。对于排序说来情况也是

这样，已经获得了某些非常有趣的发现，但仍然有许多困难的问题犹未解决。

关于排序的固有复杂性的研究通常是针对如下一些问题，即当把 n 个项目排序时、或者把 m 个项目和 n 个项目合并时，或者选择一个未编序的 n 个项目集合的第 t 个最大者时，我们如何把对诸键的比较次数极小化？5.3.1、5.3.2 和 5.3.3 节一般地讨论了这些问题，而 5.3.4 节则在有趣的限制下讨论了类似的问题，其中所做的限制是：比较的型式必须基本上预先加以固定，某些同最优排序有关的其它类型的有趣的理论问题，出现在 5.3.4 节的习题中，以及 5.4.4 节外部排序的讨论中。

5.3.1 极少比较排序

将 n 个元素排序所需要的键比较极小次数，显然为 0，因为我们已经看到基数方法全然不作比较。事实上，有可能编写出能进行排序的 MIX 程序，使得它们全然不包含条件转移指令（见本章开始的习题 5-6）！我们也已经看见了若干排序方法，它们实质上是以键的比较为基础的，然而它们的运行时间实际上取决于其它的因素，如数据移动、簿记操作等等。

因此，计算比较次数显然不是测量一个排序方法有效性的唯一方式。但是，仔细地研究比较次数，无论如何是一桩乐事。因为这个课题的理论研究会给我们带来大量的洞察排序过程本性的有用知识，也会帮助我们增进在其它情况下解决可能碰到的更为普遍的问题的能力。

为了排除完全不做比较的基数排序方法，如同本章开始时所讨论的那样，我们将把讨论局限于仅仅以键之间的一个抽象线性次序关系“<”为基础的排序技术。为了简便起见也把讨论限制在不同的键的情况，从而对任何 K_i 和 K_j 的比较，只有两种可能的结果：或者 $K_i>K_j$，或者 $K_i<K_j$。（至于把本理论扩充到允许有相等键的一般情况，见习题 3 到12。）

通过比较进行排序的问题，也可以以其它等价的方式来加以表达。给定 n 个不同的砝码和一台天平，可以提出，当天平的每个盘子仅能容纳一个砝码时，按重量的大小顺序完全排列这些砝码所需要的最少称重量次数问题。或者，给出在一项锦标赛中的一组 n 个选手，可以问及，假定选手们的实力可以线性地排序（没有平局）时，足以把所有竞赛者排名次的最少比赛次数的问题。

所有满足上述限制的 n 个元素的排序方法，都可借助图 34 所示那样的扩展二叉树结构来表示。每个内部节点（画作一个圆圈）包含两个下标“$i:j$”，表示 K_i 同 K_j 的一个比较。这个节点的左子树表示当 $K_i<K_j$ 时所需采取的动作，而右子树则表示当 $K_i>K_j$ 时所要采取的动作。该树的每个外部节点（画做一个方框）包含 $\{1, 2, \cdots, n\}$ 的一个排列 $a_1a_2\cdots a_n$，表示已经建立起次序

$$K_{a_1}<K_{a_2}<\cdots<K_{a_n}$$

这一事实（如果我们考察从根到这个外部节点的路径，则对于 $1\leqslant i<n$ 的 $n-1$ 个关系 $K_{a_i}<K_{a_{i+1}}$ 中的每一个，都将是在这条路径上某个比较 $a_i:a_{i+1}$ 或 $a_{i+1}:a_i$ 的结果）。

例如图 34 表示首先比较 K_1 和 K_2 的一个排序方法；如果 $K_1>K_2$，则它就（通过右子树）去进行 K_2 同 K_3 的比较，然后如果 $K_2<K_3$，则它就比较 K_1 和 K_3；最后，如果

$K_1>K_2$，便知道了 $K_2<K_3<K_1$。一个实在的排序算法通常也将在这个文件中移动键，但我们在这里仅对比较感兴趣，所以忽略了所有的数据移动。在这株树中 K_i 同 K_j 的比较总是意味着原来的键 K_i 同 K_j，而不是在这些记录已经被重排之后当前可能占有文件的第 i 个和第 j 个位置的键。

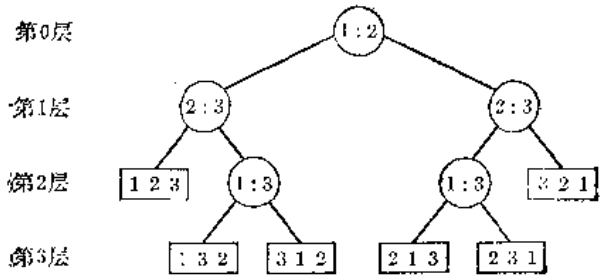


图34　对三个元素排序的一株比较树

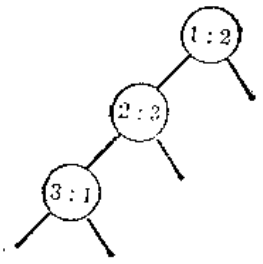


图35　多余比较的例子

有可能做了多余的比较：例如，在图35中，没有理由比较3:1，因为 $K_1<K_2$ 和 $K_2<K_3$ 意味着 $K_1<K_3$。没有排列能对应于图35中节点3:1的左子树，所以算法的该部分将永不执行！由于我们的兴趣在于把比较的次数极小化，故可以假定没有作多余的比较；因此有一个扩展的二叉树结构，其中每个外部节点都对应一个排列。输入键的所有排列都是可能的，而且每个排列都定义从根到外部节点的唯一路径；由此推知，在一株对 n 个元素进行比较而没有多余比较的比较树中，恰有 $n!$ 个外部节点。

最好的最坏情况　自然地出现的头一个问题，是找出使所作的最大比较次数为极小的比较树（后面将考虑平均的比较次数）。

命 $S(n)$ 是足以将 n 个元素排序的极小比较次数。如果一个比较树的所有内部节点都在 $<k$ 的层次内，则显然在这树中至多可以有 2^k 个外部节点。因此，令 $k=S(n)$，就有

$$n!\leqslant 2^{S(n)}$$

由于 $S(n)$ 是一个整数，我们可以重写这个公式以得到下界

$$S(n)\geqslant\lceil\log_2 n!\rceil \tag{1}$$

注意，按斯蒂林近似公式

$$\lceil\log_2 n!\rceil=n\log_2 n-n/(\ln 2)+\frac{1}{2}\log_2 n+O(1) \tag{2}$$

所以大约需要 $n\log_2 n$ 次比较。

关系（1）通常称为“信息论下界”，因为信息论的鉴赏家将说在排序过程中获得了 $(\log_2 n!)$ 个“比特的信息”；每次比较至多产生一个“比特的信息”。象图34那样的树也称作“询问”，它们的数学性质在克劳德·皮卡德（Claude Picard）的书“*Théorie des questionnaires*” Paris：Gauthier-Villars，1965）中已作过探究。

在所见过的所有排序方法中，需要最少比较的三个方法是：二叉插入（参考5.2.1节），树选择（参考5.2.3节），以及如算法5.2.4L中反映的直接两路合并。容易看出二叉插入的极大比较次数是

$$B(n)=\sum_{1\leqslant k\leqslant n}\lceil\log_2 k\rceil=n\lceil\log_2 n\rceil-2^{\lceil\log_2 n\rceil}+1 \tag{3}$$

(参考习题 1.2.4-42)，而算法 5.2.4L 的极大比较次数在习题 5.2.4-14 中给出。我们发现（见 5.3.3 节），树选择所作的比较或和二叉插入，或和两路合并有同样的界限，取决于树是怎样建立的。在所有三种情况下，都达到了 $n\log_2 n$ 的渐近值；把 $S(n)$ 的下界和上界结合起来，就证明了

$$\lim_{n\to+\infty}\frac{S(n)}{n\log_2 n}=1 \tag{4}$$

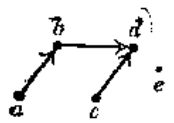
于是我们有了一个 $S(n)$ 的近似公式，但是还希望得到更精确的信息。下表给出了对于小的 n，上述量的准确值。

$n=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\lceil\log_2 n!\rceil=$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49
$B(n)=$	0	1	3	5	8	11	14	17	21	25	29	33	37	41	45	49	54
$L(n)=$	0	1	3	5	9	11	14	17	25	27	30	33	38	41	45	49	65

这里 $B(n)$ 和 $L(n)$ 分别对应于二叉插入和表合并。对所有的 n，可以证明 $B(n)\leqslant L(n)$（见习题 2）。

从上表可以看出 $S(4)=5$，但 $S(5)$ 可以是 7 或 8。这把我们带回到 5.2 节开始处所叙述的一个问题：什么是对五个元素进行排序的最好方法？能否仅仅使用七次比较，对五个元素进行排序？

回答是肯定的，但找出这种方法并不特别容易。开始时，就象用合并对四个元素排序一样，首先比较 $K_1:K_2$，然后 $K_3:K_4$，然后把每对的较大者拿来比较，这就产生了一个可以被表示为



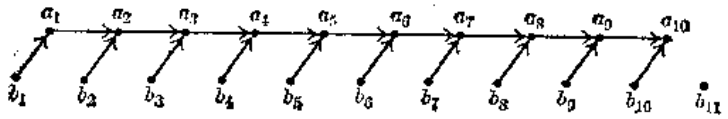
(5)

的图式，以示 $a<b<d$ 和 $c<d$。(用这种有向图来表示元素之间的已知次序关系，是方便的。当且仅当在该有向图中有一条从 x 到 y 的路径时 x 小于 y)。这时，我们把第五个元素 $K_5=e$ 插入到 $\{a,\ b,\ d\}$ 当中的适当位置，只需要比较两次，因为可以首先同 b 进行比较，而后同 a 或 d 进行比较，这有下列四种可能：

(6)

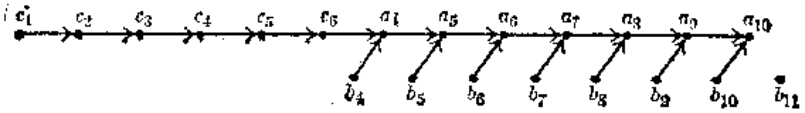
而且在每种情况下，都可以以另两次比较把 c 插入小于 d 的剩下的元素当中。H. B. 德穆斯首先发现了对五个元素排序的这两种方法〔Ph. D. thesis, Stanford University (Oct. 1956), 41-43〕。

合并插入 小莱斯特·福特（Lester Ford，Jr）和塞尔默·约翰逊（Selmer Johnson）已经发现了上述方法的一项合意的推广。由于它包含了合并的某些方面及插入的某些方面，我们称它为合并插入。例如，考虑对 21 个元素排序的问题。开始先比较十对 $K_1:K_2$，$K_3:K_4$，…，$K_{19}:K_{20}$，然后对这些对中十个较大元素进行排序，并利用合并插入。结果，得到类似于（5）的图形



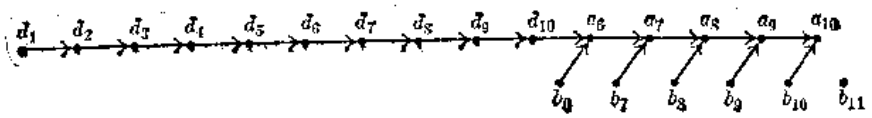
（7）

下一步是在$\{b_1, a_1, a_2\}$中插入 b_3，然后把 b_2 插入到小于 a_2 的其它元素当中；得到图形



（8）

上边一行的元素称为主链。利用三次比较（首先 b_5 同 c_4 进行比较，然后与 c_2 或 c_6 比，等等）可以把 b_5 插入到主链中的适当位置。然后 b_4 可以用另外三个步骤移入主链中，结果得到：



（9）

下一步是关键性的；知道该干什么吗？仅仅利用四次比较，我们就把 b_{11}（而不是 b_7）插入到主链。然后，也能把每个 b_{10}，b_9，b_8，b_7，b_6（以这个次序）插入到主链中的适当位置，每个至多使用四次比较。

这里涉及的对比较次数的仔细的计算，说明 21 个元素至多用 $10+22+2+2+3+3+4+4+4+4+4+4=66$ 步即可排序。由于

$$2^{65}<21!<2^{66}$$

我们也知道，在任何情况下，少于 66 次将是不可能的；因此

$$S(21)=66 \tag{10}$$

（二叉插入要求74次比较）。

一般地说，n 个元素的合并插入按如下方式进行：

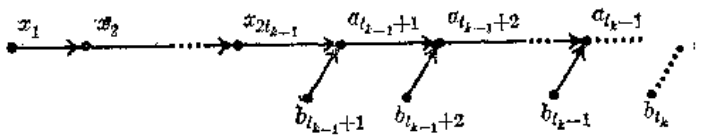
i）对于$\lfloor n/2 \rfloor$个不相交的元素对，进行逐对比较（如果 n 为奇数，则剩下一个）。

ii）通过合并插入，将步骤 i）中找到的$\lfloor n/2 \rfloor$个较大的数进行排序。

iii）象在（7）中那样命名元素 $a_1, a_2, \cdots, a_{\lfloor n/2 \rfloor}, b_1, b_2, \cdots b_{\lceil n/2 \rceil}$，其中 $a_1 \leqslant a_2 \leqslant \cdots \leqslant a_{\lfloor n/2 \rfloor}$ 且 $b_i \leqslant a_i$ 对于 $1 \leqslant i \leqslant \lfloor n/2 \rfloor$；称 b_1 和诸 a 为“主链”。以下列次序用二叉插入把诸 b_j 插入到主链中，其中 $j \leqslant \lceil n/2 \rceil$：

$$b_3,\ b_2;\ b_5,\ b_4;\ b_{11},\ b_{10}, \cdots,\ b_6; \cdots;\ b_{t_k},\ b_{t_k-1}, \cdots,\ b_{t_k+1}; \cdots \tag{11}$$

我们希望定义序列 $(t_1, t_2, t_3, t_4, \cdots)=(1, 3, 5, 11, \cdots)$，它以这样一种方式出现于（11）中，即 $b_{t_k}, b_{t_{k-1}}, \cdots, b_{t_{k-1}+1}$ 中的每一个都可以用至多 k 次比较插入到主链中，推广（7）、（8）和（9），我们得到图形



其中达到并包括 $a_{t_{k-1}}$ 的主链包含 $2t_{k-1}+(t_k-t_{k-1}-1)$ 个元素。这个数必然小于2^k；我们最好的赌注是把它置成等于 2^k-1，使得

$$t_{k-1}-t_k=2^k \tag{12}$$

由于 $t_1=1$，为了方便起见我们可以置 $t_0=1$，通过对一个几何级数求和得到

$$t_k=2^k-t_{k-1}=2^k-2^{k-1}+t_{k-2}=\cdots=2^k-2^{k-1}+\cdots+(-1)^k2^0=(2^{k+1}+(-1)^k)/3 \tag{13}$$

（奇怪的是，这同一序列出现在我们为计算两个整数的最大公因子的一个算法研究中，参考习题4·5·2-27。）

令$F(n)$是通过合并插入将 n 个元素排序所需要的比较次数，显然。

$$F(n)=\lfloor n/2\rfloor+F(\lfloor n/2\rfloor)+G(\lceil n/2\rceil) \tag{14}$$

其中G表示包含在步骤(iii)中的工作量。如果 $t_{k-1}\leqslant m\leqslant t_k$，则通过部分求和我们有

$$G(m)=\sum_{1\leqslant j<k} j(t_j-t_{j-1})+k(m-t_{k-1})=km-(t_0+t_1+\cdots+t_{k-1}) \tag{15}$$

置

$$w_k=t_0+t_1+\cdots+t_{k-1}=\lfloor 2^{k+1}/3\rfloor \tag{16}$$

使得 $(w_0, w_1, w_2, w_3, w_4, \cdots)=(0, 1, 2, 5, 10, 21, \cdots)$，习题13证明

$$F(n)-F(n-1)=k \quad \text{当且仅当 } w_k<n\leqslant w_{k+1} \tag{17}$$

并且后一条件等价于

$$\frac{2^{k-1}}{3}<n\leqslant\frac{2^{k+2}}{3}$$

$$k+1<\log_2(3n)\leqslant k+2$$

因此

$$F(n)-F(n-1)=\left\lceil \log_2\left(\frac{3}{4}n\right)\right\rceil \tag{18}$$

（这个公式属于 A. 哈迪安（A. Hadian）〔Ph. D. thesis, Univ. of Minnesota (1969), 38-42〕。）由此得出，$F(n)$有相当简单的表达式

$$F(n)=\sum_{1\leqslant k\leqslant n}\left\lceil \log_2\left(\frac{3}{4}k\right)\right\rceil \tag{19}$$

它十分类似于二叉插入的对应公式（3）。这个和的“封闭形式”出现于习题14中。

利用等式(19)不难构造$F(n)$的一个表；我们有

$n=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
$\lceil\log_2 n!\rceil=$	0	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49	
$F(n)=$	0	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50	

$n=$ 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33

$\lceil \log_2 n! \rceil=$ 53 57 62 66 70 75 80 84 89 94 98 103 108 113 118 123

$F(n)=$ 54 58 62 66 71 76 81 86 91 96 101 106 111 116 121 126

注意对于 $1 \leqslant n \leqslant 11$ 和 $20 \leqslant n \leqslant 21$，$F(n)=\lceil \log_2 n! \rceil$，所以我们知道对于这些 n 合并插入是最优的：

$$S(n)=\lceil \log_2 n! \rceil=F(n) \text{对于} 1 \leqslant n \leqslant 11 \text{和} 20 \leqslant n \leqslant 21 \tag{20}$$

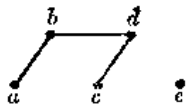
雨果·斯坦豪斯（Hugo Steinhaus）在他著名的书"*Mathematical Snapshots*"（Oxford University Press, 1950）的第二版38～39页中提出了求 $S(n)$ 的问题。他描述了二叉插入的方法——它在下述意义下是对 n 个对象排序的最好的方法，即：如果先把前 $n-1$ 个排好序，再考虑第 n 个；而且，他猜测，一般说来，二叉插入是最优的。若干年后〔Calcutta Math. Soc. Golden Jubilee Commemoration 2(1959)。323-327〕他报告说，他的两个同事S. 特里布拉（S. Trybuta）和成平，"最近"已否定了他的猜测，并且已经在 $n \leqslant 11$ 的情况下确定了 $S(n)$。特里布拉和成平可能已经独立地发现了合并插入的方法，不久之后福特和约翰逊就发表了这个方法〔AMM66(1959)，387-389〕。

在发现了合并插入之后，头一个未知的 $S(n)$ 值是 $S(12)$，表1示出了12!十分接近于 2^{29}，所以不大可能只用29步对12个元素进行排序。因此，麦克·韦尔（Mark Wells）进行了一次穷举查找（在一台Maniac Ⅱ计算机上大约花了60小时），他发现 $S(12)=30$〔Proc IFIP Congress 65 2(1965)，497-498〕，于是证实合并插入过程对于 $n=12$ 也是最优的。

表1　阶乘值的二进制表示

1	= 1!
10	= 2!
110	= 3!
11000	= 4!
1111000	= 5!
1011010000	= 6!
1001110110000	= 7!
1001110110000000	= 8!
1011000100110000000	= 9!
1101110101111100000000	= 10!
1001100001000101010000000000	= 11!
1110001000110011111100000000000	= 12!
1011100110010100011001100000000000	= 13!
1010001001101111110110010100000000000000	= 14!
1001100000111011101110111010110000000000000	= 15!
1001100000111011101110111010110000000000000000	= 16!

***一个稍更深入的分析**　为了更仔细地研究 $S(n)$，让我们更严密地考察例如（5）那样的偏序图式。在进行了若干次比较之后，可以借助于一个有向图来表示已经获得的知识。鉴于<关系的传递性，这个有向图不包含回路，所以可以把所有的弧都画成从左到右；因此不妨去掉图中的箭头。于是（5）变为



(21)

如果G是这样一个有向图，命$T(G)$是同G一致的排列的数目，即，把整数$\{1, 2, \cdots, n\}$赋予G各顶点的方式种数，使得对G中的任意$x \to y$，顶点x上的数均小于顶点y上的数。例如，同(21)一致的一种排列是$a=1$，$b=4$，$c=2$，$d=5$，$e=3$。在5.1.4节中已经对各种G研究了$T(G)$，其中我们发现，$T(G)$表示G可以被拓扑地排序的方式种数。

如果G是在进行了k次比较之后可以得到的n个元素的一个图，我们定义G的效率为

$$E(G)=\frac{n!}{2^k T(G)} \tag{22}$$

（这个思想属于黄光明和林身。）严格地说，效率并不单独是图G的一个函数，它依赖于在一个排序过程中得到G的方式，但是在叙述时不妨略去这一点。在元素i和j之间进行了另一次比较之后，得到两个图G_1和G_2，一个是对于情况$K_i<K_j$的，一个是对于情况$K_i>K_j$的。显然

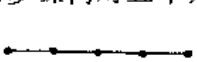
$$T(G)=T(G_1)+T(G_2)$$

如果$T(G_1)\geqslant T(G_2)$，则有

$$T(G)\leqslant 2T(G_1)$$

$$E(G_1)=\frac{n!}{2^{k+1}T(G_1)}=\frac{E(G)T(G)}{2T(G_1)}\leqslant E(G) \tag{23}$$

因此，每次比较导致了一个具有更小或相等效率的图；不可能通过作进一步的比较来改进效率。

注意，当G全然没有弧时，有$k=0$及$T(G)=n!$，所以初始的效率是1。而且，当G是表示排序最后结果的一个图时，G看起来象一条直线，且$T(G)=1$。于是，例如，如果要寻找一个排序过程，在七步或更少的步骤内对五个元素排序，则必须得到其效率为$5!/(2^7\times 1)=120/128=15/16$的线性图  。由此得出在排序过程中出现的所有图必然有$\geqslant 15/16$的效率；如果出现任何低效率的图，则至少它的后裔之一也将是低效率的，而且最终将达到其效率$<\frac{15}{16}$的一个线性图。一般地说，这个论证证明，所有对应于n个元素排序过程的树节点的图，都必然有$\geqslant n!/2^l$的效率，其中$l+1$是树的层数。这是证明$S(n)\geqslant\lceil\log_2 n!\rceil$的另一个方法，尽管这个论证同我们以前所说的实际上并没有多少不同。

图（21）的效率为1，因为$T(G)=15$，而且由于G已经在三次比较中得到，为了看看下次应当比较什么顶点，可以形成比较矩阵

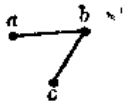
$$C(G)=\begin{array}{c} \\ a \\ b \\ c \\ d \\ e \end{array}\begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 15 & 10 & 15 & 11 \\ 0 & 0 & 5 & 15 & 7 \\ 5 & 10 & 0 & 15 & 9 \\ 0 & 0 & 0 & 0 & 3 \\ 4 & 8 & 6 & 12 & 0 \end{pmatrix} \end{array} \tag{24}$$

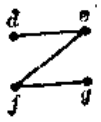
其中 C_{ij} 是通过把弧 $i \to j$ 加到 G 得到的图 G_1 的 $T(G_1)$。例如，如果比较 K_a 和 K_e，则同 G 相一致的15个排列分裂成 $K_e<K_a$ 的 $C_{ea}=6$ 个和 $K_e<K_a$ 的 $C_{ea}=9$ 个，后一个图的效率是 $15/(2\times 9)=5/6<15/16$，所以它不可能导致一个七步排序过程。下一个比较必然是 $K_b:K_e$，为的是保持 $\geqslant\frac{15}{16}$ 的效率。

当考虑图的"连通分量"时，"效率"的概念是特别有用的。例如考虑图

$G=$

它有两个分量

$G'=$ 

和 $G''=$ 

由于没有连接 G' 和 G'' 的弧，所以它由完全在 G' 之内进行的某些比较和完全在 G'' 之内进行的其它比较形成。一般地说，假定 $G=G'+G''$ 没有 G' 和 G'' 之间的弧，其中 G' 和 G'' 分别有 n' 和 n'' 个顶点，容易看出

$$T(G)=\binom{n'+n''}{n'}T(G')\,T(G'') \tag{25}$$

因为 G 的每个一致的排列都通过选择 n' 个元素赋给 G'，然后独立地在 G' 和 G'' 之内构造一致的排列而得到。如果在 G' 内已经进行了 k' 次比较，在 G'' 内进行了 k'' 次比较，则有基本的结果

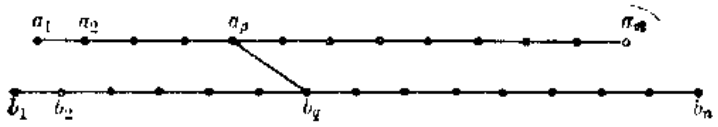
$$E(G)=\frac{(n'+n'')!}{2^{k'+k''}T(G)}=\frac{n'!}{2^{k'}T(G')}\cdot\frac{n''!}{2^{k''}T(G'')}=E(G')\,E(G'') \tag{26}$$

说明一个图的效率以一种简单的方式同其分量的效率有关。因此，我们可以限于考虑仅有一个分量的图。

现在假定 G' 和 G'' 是一个分量的图，并且假设要通过比较 G' 的一个顶点 x 和 G'' 的一个顶点 y 而把它们钩到一起。要知道这个效率是什么，为此，需要一个可以通过

$$\binom{p}{m}<\binom{q}{m} \tag{27}$$

表示的函数，定义为同图



(28)

一致的排列的数目，于是$\binom{p \quad < \quad q}{m \qquad n}$等于$\binom{m+n}{m}$乘以一个概率数，该概率数是$m$个数的一个集合的第$p$个最小者小于独立地选择的$n$个数的集合的第$q$个最小者的概率。习题17证明，可以借助于二项式系数以两种方式表达$\binom{p \quad < \quad q}{m \qquad n}$。

$$\binom{p \quad > \quad q}{m \qquad n}=\sum_{0\leqslant k<q}\binom{m-p+n-k}{m-p}\binom{p-1+k}{p-1}$$

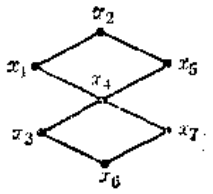
$$=\sum_{p\leqslant j\leqslant m}\binom{n-q+m-j}{n-q}\binom{q-1+j}{q-1} \tag{29}$$

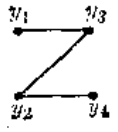
（顺便说一句，从代数上看来，如下一件事决不是明显的：这些二项式系数乘积的两个和居然是相等的。）我们也有公式

$$\binom{p \quad < \quad q}{m \qquad n}+\binom{q \quad < \quad p}{n \qquad m}=\binom{m+n}{n} \tag{30}$$

$$\binom{q \quad < \quad p}{n \qquad n}=\binom{m+1-p \quad < \quad n+1-q}{m \qquad\qquad n} \tag{31}$$

为了确定起见，现在考虑两个图

$G'=$ 

$G''=$ 

(32)

通过直接枚举，不难证明$T(G')=42$和$T(G'')=5$；所以如果G是以G'和G''作为分量的11个顶点的图，则按等式(25)，我们有$T(G)=\binom{11}{4}\cdot 42\cdot 5=69300$。如果要知道对于每个$i$和$j$有多少$x_i<y_j$，这是待列出的惊人的排列的数目。但是，这个计算可以在不到一个小时之内用手算出如下：构造矩阵$A(G')$和$A(G'')$，这里A_{ik}是使得x_i(或y_i)等于k的G'(或G'')的一致排列的个数。于是，其中x_i小于y_i的G的排列个数是$A(G')$的(ip)元素乘以$\binom{p \quad < \quad q}{7 \qquad 4}$乘以$A(G'')$的$(jq)$元素，对$1\leqslant p\leqslant 7$和$1\leqslant q\leqslant 4$进行求和。换言之，我们要构造矩阵乘积$A(G')\cdot L\cdot A(G'')^T$，其中$L_{pq}=\binom{p \quad < \quad q}{7 \qquad 4}$。这得到

$$\begin{pmatrix} 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 21 & 16 & 5 & 0 & 0 & 0 & 0 \\ 0 & 0 & 12 & 18 & 12 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \\ 0 & 5 & 10 & 12 & 10 & 5 & 0 \\ 0 & 0 & 0 & 0 & 5 & 16 & 21 \end{pmatrix} \begin{pmatrix} 210 & 294 & 322 & 329 \\ 126 & 238 & 301 & 325 \\ 70 & 175 & 265 & 315 \\ 35 & 115 & 215 & 295 \\ 15 & 65 & 155 & 260 \\ 5 & 29 & 92 & 204 \\ 1 & 8 & 36 & 120 \end{pmatrix} \begin{pmatrix} 2 & 3 & 0 & 0 \\ 2 & 2 & 0 & 1 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 3 & 2 \end{pmatrix}$$

$$= \begin{pmatrix} 48169 & 42042 & 66858 & 64031 \\ 22825 & 16005 & 53295 & 46475 \\ 48169 & 42042 & 66858 & 64031 \\ 22110 & 14850 & 54450 & 47190 \\ 5269 & 2442 & 27258 & 21131 \\ 22825 & 16005 & 53295 & 46475 \\ 5269 & 2442 & 27258 & 21131 \end{pmatrix}$$

于是钩起 G' 和 G'' 的"最好的"方法是把 x_1 同 y_2 加以比较；当 $x_1<y_2$ 时有42042种情况，当 $x_1>y_2$ 时有$69300-42042=27258$种情况。（由对称性，我们也可以比较 x_3 和 y_2，x_5 和 y_3，或 x_7 和 y_3，导出实际上相同的结果。）对于 $x_1<y_2$ 得到的图效率是

$$\frac{69300}{84084}E(G')E(G'')$$

它不是太好的；因此，在任何排序方法中把 G' 和 G'' 钩起来大概会是一种坏的想法！这个例子的要点是，无须多余的计算就能够作出这样一个判断来。

这些思想可用来独立地证实麦克·韦尔斯关于 $S(12)=30$ 的证明。从包含一个顶点的图开始，可以重复地尝试用这样一种方式对图 G 之一或对一对图分量 G' 和 G'' 增加一个比较，使得两个由此得到的图只有12个或更少的顶点，且效率$\geqslant 12!/2^{29}\approx 0.89221$。只要这是可能的，就取所得到的最低效率的图，并且把它加到我们的集合中，除非它同构于集合中已有的一个图（或者同构于通过颠倒次序所得到的这样一个图的对偶图）。如果得到的两个图有同样的效率，则就任意地选择它们之一。图36中示出了以这一方式得到的头24个图及其效率。

在整个过程中，计算机一共生成了1594个图。因为不能得到图 ，故可以断定 $S(12)>29$。看来我们也可以通过类似的实验，花费不太长的时间以推导出 $S(22)>70$，因为$22!/2^{70}\approx 0.952$表明必须有极高的效率才能在70步之内进行排序。（在用12个或更少的顶点建立的1594个图中，仅有92个有这样的高效率。）

这些中间结果强烈地提示 $S(13)=33$，所以当 $n=13$ 时该合并插入不是最优的。但是至今还没有人发现存在任何 n 使得 $S(n)<F(n)$。

肯定应有可能证明 $S(16)<F(16)$，因为比起首先通过 $S(10)$ 次比较对十个元素排序，然后用二叉插入逐个插入其它六个元素来，$F(16)$ 所需的比较次数决不会更少。必然存在一种办法来改进这样一种方法！

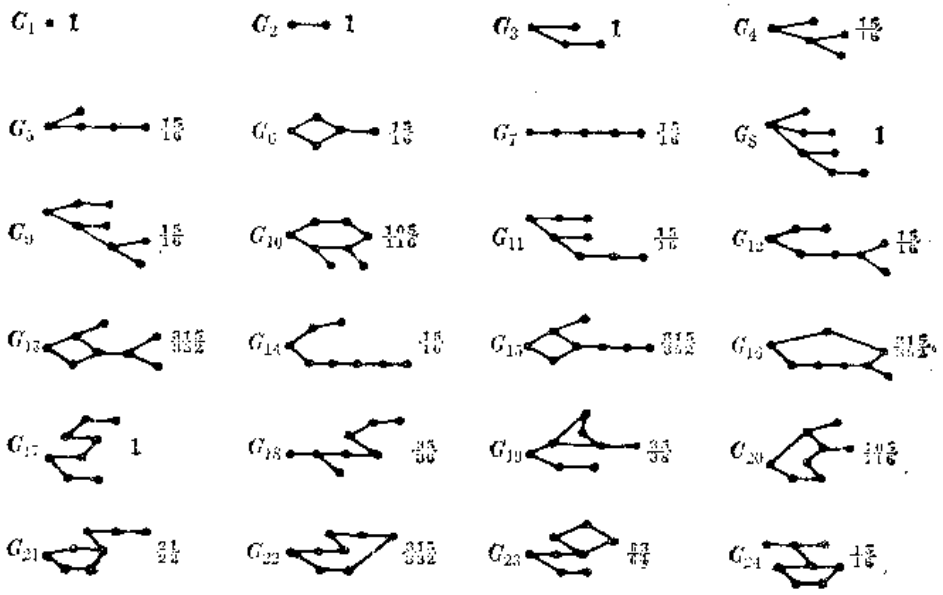


图36 在 $S(12)>29$ 的一个长证明的最初几步中得到的某些图和它们的效率

比较的平均次数 至今已经考虑了一些过程，在它们的最坏情形尚不坏这个意义下它们是最好的；已经寻找过“极小化极大”的过程，对于比较的极大次数而言它取极小。现在让我们寻找一个“极小化平均值”的过程，它对比较的平均次数取极小。假定输入是随机的，于是每种排列都是同等可能的。

如图34所示再次考虑一个排序过程的树表示。在该树中的平均比较次数是

$$\frac{2+3+3+3+3+2}{6}=2\frac{2}{3}$$

（对于所有的排列取平均。）一般地说，在一个排序方法中平均的比较次数是树的外部路径长度除以$n!$（回想一下，外部路径长度是从根到每个外部节点的距离之和；见2.3.4.5节）。从2.3.4.5节的一些考虑容易看出，如果一个具有N个外部节点的二叉树在$q-1$层有2^q-N个外部节点，而在q层有$2N-2^q$个节点，其中$q=\lceil\log_2 N\rceil$（根在0层），则此树具有极小的外部路径长度，因此极小的外部路径长度是

$$(q-1)(2^q-N)+q(2N-2^q)=(q+1)N-2^q \tag{33}$$

极小路径长度也能由另一种有趣的方式加以表征：当且仅当有一个数l，使得一株扩展的二叉树的所有外部节点都出现在l层和$l+1$层上时，此树有极小的外部路径长度（见习题20）。

如果置$q=\log_2 N+\theta$，其中$0\leqslant\theta<1$，则极小外部路径长度的公式变为：

$$N(\log_2 N+1+\theta+2^\theta) \tag{34}$$

函数$1+\theta-2^\theta$示于图37中；对于$0<\theta<1$它是正的但是非常小，决不超过

$$1-(1+\ln\ln 2)/(\ln 2)=0.08607\ 13320\ 55934+ \tag{35}$$

于是，通过将(34)除以N得到的极小可能平均路径长度，决不小于$\log_2 N$也决不大于$\log_2 N$

+0.0861（这个结果首先是由 A. 格利森（A. Gleason）在一个未发表的备忘录中得到的(1956)）。

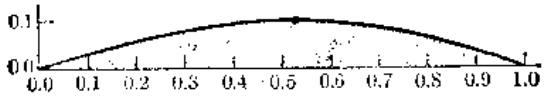


图37 函数 $1+\theta-2^{\theta}$

现在如果置$N=n!$，则得到在任何排序方案中平均比较次数的下界，注意，这个下界是 $\log_2 n!+O(1)=n\log_2 n-n/(\ln 2)+O(\log n)$

设 $\overline{F}(n)$是由合并插入算法执行的平均比较次数；我们有

$n=1$ 2 3 4 5 6 7 8

下界(33)= 0 2 16 112 832 6896 62368 619904

$n!\overline{F}(n)=$ 0 2 16 112 832 6912 62784 623232

于是合并插入对于 $n\leqslant 5$ 在两种意义下都是最优的，但是对于 $n=6$，它平均有6912/720 =9.6次比较，而我们的下界表明平均6896/720=9.577777…次比较是可能的。片刻的思考就可明白为什么这是正确的：采用合并插入法，只需八次比较，即可把六个元素的某些"幸运的"排列排好序，所以比较树在三个层次上而不是在两个层次上出现外部节点。这使得整个路径长度变长了。习题24说明有可能构造一个六元素的排序过程。这个过程在每种情况下都要求九或十次比较；由此得出，平均来说，这个方法比合并插入优越。而且在其最坏情况下，也不比合并插入更坏。

当 $n=7$ 时，Y. 基萨里（Y. Césari)〔thesis(Paris 1968)〕，P. 37〕已经说明，没有排序方法能达到外部路径长度的下界62368（利用习题22的结果，有可能用手算来证明这个事实)。另一方面，他已经构造了一些过程，当 $n=9$ 或10时，它们达到下界(33)。一般地说，使平均比较次数极小化的问题，实质上比确定$S(n)$的问题更为困难。甚至可能对于某个n，所有使平均比较次数极小化的方法在它们最坏的情况下要求进行 $s(n)$次以上的比较。

习题

1.〔20〕（a）用二叉插入法，（b）用直接的两路合并法，画出对四个元素进行排序的比较树。这些树的外部路径长度是多少？

2.〔$M24$〕 证明$B(n)\leqslant L(n)$，并求使等式成立的所有的n。

3.〔$M22$〕 当允许键之间的等式时，在对三个元素排序时有13种可能的结果：

$$K_1=K_2=K_3 \qquad K_1=K_2<K_3 \qquad K_1=K_3<K_2$$
$$K_2=K_3<K_1 \qquad K_1<K_2=K_3 \qquad K_2<K_1=K_3$$
$$K_3<K_1=K_2 \qquad K_1<K_2<K_3 \qquad K_1<K_3<K_2$$
$$K_2<K_1<K_3 \quad K_2<K_3<K_1 \quad K_3<K_1<K_2 \quad K_3<K_2<K_1$$

令 P_n 表示在对n个元素排序，而且允许钩连时可能的结果数目，于是，$(P_0, P_1, P_2, P_3, P_4, P_5, \cdots)=(1, 1, 3, 13, 75, 541, \cdots)$，证明母函数 $P(z)=\sum_{n\geqslant 0}P_n z^n/n!$ 等于 $1/(2-e^z)$。提示：证明

$$P_n=\sum_{k>0}\binom{n}{k}P_{n-k}\quad 当\ n>0$$

4.〔$HM27$〕（O. A. 格罗斯（O. A. Gross））确定当 $n\to\infty$ 时，习题3的数 P_n 的渐近值。〔也许可以提示：考虑 $\cot z$ 的部分分式展开。〕

5.〔16〕 当键可以相等时，每个比较可以有三种结果而不是两种结果：$K_i<K_j$，$K_i=K_j$，$K_i>K_j$。这类一般情况的排序算法，可表示成扩展的三叉树，其中每个内部节点 $i:j$ 有三株子树；左边、中间和右边子树分别对应于比较的三种可能的结果。

试画出一株扩展的三叉树，它定义了允许有相等键的 $n=3$ 时的排序算法。这株树应有13个外部节点，对应于习题3中列出的13种可能的结果。

▶6.〔$M22$〕 设每个比较如同在习题5中那样有三种结果，$S'(n)$ 是对 n 个元素排序并确定诸键之间所有等式所需要的最小比较数。正文中的"信息论"论证方法可以容易地被推广来证明 $S'(n)\geqslant\lceil\log_3 P_n\rceil$，其中 P_n 是在习题3和4中所研究的函数；但请证明，事实上 $S'(n)=S(n)$。

7.〔20〕 在习题5对四个元素排序的意义下，当已知所有键都是0或1时，画出扩展的三叉树。（于是，如果 $K_1<K_2$ 和 $K_3<K_4$，则我们知道 $K_1=K_3$ 和 $K_2=K_4$！）假定 2^4 种输入都是同等可能的，请使用极小的平均比较次数。注意找出所有的等式。例如，当你只知道 $K_1\leqslant K_2\leqslant K_3\leqslant K_4$ 时切勿停止排序。

8.〔26〕 当已知所有的键皆为 -1、0或 $+1$ 时，象习题7中那样，对于四个元素排序，画出一株扩展的三叉树，假定 3^4 种输入都是同等可能的，请使用极小的平均比较次数。

9.〔$M20$〕 当象在习题7中那样，对 n 个元素进行排序，且又知道所有键皆为0或1时，试问在最坏情况下极小的比较次数是多少？

▶10.〔$M25$〕 当象在习题7中那样，对 n 个元素进行排序，又知道所有键皆为0或1时，试问作为 n 的函数，极小平均比较次数是多少？

11.〔$HM25$〕 当象在习题5中那样，对 n 个元素进行排序，又知道所有键都是集合 $\{1,2,\cdots,m\}$ 上的数时，命 $S_m(n)$ 是在最坏的情况下需要的极小比较数〔于是，由习题6，$S_n(n)=S(n)$〕，试证明对于固定的 m，当 $n\to\infty$ 时，$S_m(n)$ 渐近于 $n\log_2 m$。

12.〔$M25$〕〔W. G. 布里西尤斯（W. G. Bouricius） 约1954〕 假设可以出现相等的键，但是只需要对元素 $\{K_1,K_2,\cdots,K_n\}$ 排序，找出排列 $a_1a_2\cdots a_n$，使得 $K_{a_1}\leqslant K_{a_2}\leqslant\cdots K_{a_n}$；我们不需要知道 K_{a_j} 和 $K_{a_{j+1}}$ 是否相等。

如果一株比较树在上述意义下对一个键序列排序，当 $K_i=K_j$ 时它在节点 $i:j$ 之下任取一分枝（这株树是二叉的，而不是三叉的），则这种排序称为强排序。

a）证明没有多余比较的一个比较树排序，当且仅当它对每个无相同键的序列进行排序时，它也对每个键进行强排序。

b）证明：当且仅当一株比较树对0和1的每个序列进行强排序时，它也对每个键序列进行强排序。

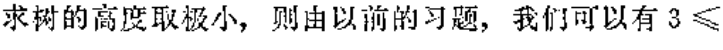
13.〔$M28$〕 证明(17)。

14.〔$M24$〕 试求和数(19)的一个“封闭形式”。

15.〔$M21$〕 确定$B(n)$和$F(n)$的渐近行为，精确到$O(\log_2 n)$。〔提示：证明在这两种情况下，n的系数都包含图37中所示的函数。〕

16.〔$HM26$〕 (黄光明和林身) 证明当$n\geqslant 22$时$F(n)>\lceil\log_2 n!\rceil$。

17.〔$M20$〕 证明(29)。

18.〔20〕 如果图36中开始的过程产生了具有效率$121/2^{29}$的图  ，则这将证明$S(12)=29$吗？

19.〔40〕 以下列带启发性的探索规则进行实验，来判定在设计一株比较树时下次应比较哪一对元素：在对$\{K_1,\cdots,K_n\}$进行排序的每一个阶段，对于$1\leqslant i<n$，命u_i是作为迄今所作比较的结果而得到的$\leqslant K_i$的键数，v_i是已知为$\geqslant K_i$的键数，按上升的u_i/v_i对诸键重新编号，使得$u_1/v_1\leqslant u_2/v_2\cdots\leqslant u_n/v_n$，现在对于某个使$|u_iv_{i+1}-u_{i+1}v_i|$取极小的$i$，比较$K_i:K_{i+1}$(比起(24)中所用的完全比较矩阵，本方法所用的信息要少得多。但它在许多情况下似乎给出最优的结果)。

▶20.〔$M26$〕 证明：当且仅当存在一个数l，使得一株扩展的二叉树的所有外部节点都出现于第l层和第$l+1$层上时（或许，所有的节点都在同一层上），该树有极小的外部路径长度。

21.〔$M21$〕 一株扩展的二叉树的高度是它的外部节点的极大层次数。如果x是一株扩展的二叉树的内部节点，命$t(x)$是x以下的外部节点数，并命$l(x)$表示x的左子树的根。如果x是外部节点，则命$t(x)=1$。试证一株扩展二叉树在具有相同节点数的所有二叉数中具有极小高度的充分必要条件是：对于所有内部节点

$$|t(x)-2t(l(x))|\leqslant 2^{\lceil\log_2 t(x)\rceil}-t(x)$$

22.〔$M24$〕 继续习题21，证明一株二叉树在具有同样节点数的所有二叉树当中，拥有极小外部路径长度的充分必要条件是对所有内部节点x

$|t(x)-2t(l(x))|\leqslant 2^{\lceil\log_2 t(x)\rceil}-t(x)$和$|t(x)-2t(l(x))|\leqslant t(x)-2^{\lfloor\log_2 t(x)\rfloor}$

〔于是，例如，如果$t(x)=67$则我们必定有$t(l(x))=32$，33，34，或35。如果只要求树的高度取极小，则由以前的习题，我们可以有$3\leqslant t(l(x))\leqslant 64$。〕

23.〔10〕 正文证明〔参考〔34〕〕，任何对n个元素进行排序的方法，其平均比较次数至少是$\lceil\log_2 n!\rceil\approx n\log_2 n$，但是多重表插入（算法5.2.1M）平均仅仅花费$O(n)$个时间单位，为什么能这样？

24.〔27〕 (C.皮卡德) 试求一株使得所有外部节点都出现于10层和11层的对六个元素排序的树。

25.〔11〕 如果有一个对七个元素排序的过程，它达到等式(34)所预测的极小平均比较次数，则在13层上将有多少外部节点？

26.〔$M42$〕 试求对七个元素进行排序的一个过程，它的平均比较次数取极小。

▶27.〔20〕 假设已知配置($K_1<K_2<K_3$，$K_1<K_3<K_2$，$K_2<K_1<K_3$，$K_2<K_3<K_1$，$K_3<K_1<K_2$，$K_3<K_2<K_1$)分别以概率(.01，.25，.01，.24，.25，.24)出现，试找出一株比较树，它以最小的平均比较次数对这三个元素排序。

28.〔40〕 写出一个 MIX 程序，它在最少时间内对五个单字长的键进行排序，并停机(关于基础规则见 5.2 节开头)。

29.〔M25〕〔S. M. 蔡斯（S. M. Chase)〕 设 $a_1a_2\cdots a_n$ 是 $\{1, 2, \cdots, n\}$ 的一个排列，证明任何一个仅仅以诸 a 之间的比较为基础来判断这个排列是偶还是奇（亦即它是否有偶的或奇的反序数）的算法，必须至少进行 $n\log_2 n$ 次比较。即使这个算法仅有两个可能的结果亦然。

30.〔M23〕（最优交换排序）在 5.2.2 节中定义的每一个交换排序算法都可表示为一株比较交换树；亦即一二叉树结构，其内部节点具形式 $i:j$，$i<j$，它可解释作下列的操作："如果 $K_i\leqslant K_j$，则通过取这株树的左分枝继续进行；如果 $K_i>K_j$，则通过交换记录 i 和 j，而后取这株树的右分枝继续进行"。当遇到一个外部节点时，$K_1\leqslant K_2\leqslant\cdots\leqslant K_n$ 必然为真。因此，一株比较交换树与一株比较树的区别在于：它指明数据移动以及比较操作。

命 $S_e(n)$ 表示在最坏的情况下，借助于一株比较交换树对 n 个元素排序所需要的极小比较交换数。证明 $S_e(n)\leqslant S(n)+n-1$。

31.〔M28〕 继续习题 30，证明 $S_e(5)=8$。

32.〔M42〕 继续习题 31，对于 $n>5$ 的小的值研究 $S_e(n)$。

33.〔M30〕（T·N·希巴德）阶为 x 和分辨度为 δ 的实值的查找树是一株扩展的二叉树，其中所有的节点都包含一个非负的实值，使得（i）在每个外部节点中的值都 $\leqslant\delta$；(ii) 在每个内部节点的值都 $\leqslant$ 它的两个儿子值的和；(iii) 根的值是 x。这样一株树的加权路径长度，定义为每个外部节点的层次乘以它包含的值再对所有外部节点求和。

证明一株阶为 x 和分辨度为 1 的实值查找树，在具同样的阶和分辨度的所有这样的树中，具有极小的加权路径长度的充分必要条件是：在 (ii) 中的等式成立，且对于包含于兄弟节点中的所有值偶 x_0 和 x_1，下列进一步的条件成立：(iv) 不存在整数 $k\geqslant 0$ 使得 $x_0<2^k<x_1$ 或 $x_1<2^k<x_0$；（v）$\lceil x_0\rceil-x_0+\lceil x_1\rceil-x_1<1$。(特别是，如果 x 是整数，则条件（v）意味着在这株树中的所有值都是整数，且条件（iv）等价于习题 22 的结果。)

并且证明对应的极小加权路径长度是 $x\lceil\log_2 x\rceil+\lceil x\rceil-2^{\lceil\log_2 x\rceil}$。

34.〔M50〕 对于无穷多个 n 确定 $S(n)$ 的精确值。

*5.3.2 极少比较合并

现在考虑一个相关的问题：什么是把 m 个元素的一个有序集合和 n 个元素的一个有序集合合并的最好方法？以

$$A_1<A_2<\cdots<A_m$$
$$B_1<B_2<\cdots<B_n \tag{1}$$

表示有待合并的元素，如同在 5.3.1 节中那样，假定这 $m+n$ 个元素是不同的。诸 A 可以 $\binom{m+n}{m}$ 种方式出现于诸 B 当中，所以，已经使用于排序问题的论证立即告诉我们，至少需要

$$\left\lceil\log_2\binom{m+n}{m}\right\rceil \tag{2}$$

次比较。如果置 $m=\alpha n$ 并设 $n\to\infty$，而 α 是固定的，则斯特林近似公式告诉我们

$$\log_2\binom{\alpha n+n}{\alpha n}=n((1+\alpha)\log_2(1+\alpha)-\alpha\log_2\alpha)-\frac{1}{2}\log_2 n+O(1) \qquad (3)$$

正常的合并过程，即算法 5.2.4M，在它最坏的情况下，花费$m+n-1$次比较。

命$M(m,\ n)$表示类似于$S(n)$的函数，亦即总是足以把m件事物同n个事物合并的极小比较次数，按我们刚才所作的观察

$$\left\lceil\log_2\binom{m+n}{m}\right\rceil\leqslant M(m,\ n)\leqslant m+n-1 \text{ 对于所有} m,\ n\geqslant 1 \qquad (4)$$

公式（3）说明这个下界可以如何远离上界，当$\alpha=1$（即$m=n$）时，下界是$2n-\frac{1}{2}\log_2 n+O(1)$，所以上下界两者都有正确数量级，但是它们之间的差可以任意大，当$\alpha=0.5$(即$m=\frac{1}{2}n$)时，下界为

$$\frac{3}{2}n\left(\log_2 3-\frac{2}{3}\right)+O(\log n)$$

它大约是 $\log_2 3-\frac{2}{3}\approx 0.918$ 乘上界。而且，随着α减小，这些界限就越离越远，因为标准的合并算法主要是对$m=n$的文件来设计的。

当$m=n$时，合并的问题有相当简单的解；由此可知，是（4）的下界有问题，而不是上界有毛病。下列定理是由 R. L. 格雷厄姆（R. L. Graham）和 R. M. 卡普大约于 1968 年独立地发现的：

定理 M 当$m\geqslant 1$时，$M(m,\ m)=2m-1$。

证明 考虑把$A_1<\cdots<A_m$同$B_1<\cdots<B_m$合并的任何算法。当它比较$A_i:B_j$时，如果$i<j$取分枝$A_i<B_j$；如果$i\geqslant j$取分枝$A_i>B_j$。合并最终地必须以配置

$$B_1<A_1<B_2<A_2<\cdots<B_m<A_m \qquad (5)$$

结束，因为这同所采取的所有分枝一致，而且$2m-1$个比较$B_1:A_1$，$A_1:B_2$，$B_2:A_2$，…，$B_m:A_m$中的每一个都必须已经明显地作出，否则至少会有两个配置同已知的事实相一致，例如，如果A_1未与B_2进行比较，则配置

$$B_1<B_2<A_1<A_2\cdots<B_m<A_m$$

是与（5）不能相区别的。

简单修改这个证明，就得出伴随的公式

$$M(m,\ m+1)=2m \qquad \text{对于} m\geqslant 0 \qquad (6)$$

构造下界 定理M表明，“信息论”的下界（2）可以同真正的下界相距任意远；于是，用于证明定理M的技术给了我们发现下界的另一个办法，这样一个证明技术通常被看作是制造一个敌手，一个有害的人，他试图使这些算法缓慢地进行。当用于合并的一个算法比较$A_i:B_j$时，这个敌手是这样确定比较的结局的：就好象他要使这个算法沿着较为困难的道路走下去，如同在定理M的证明中那样。如果能够发明一个适当的敌手，就能够确保，每一个正确的合并算法将必须作更大量的比较。〔某些人已经使用“神谕”或“精灵”这样一些字眼来代替“敌手”（魔鬼），但在这里避免使用这样的术语是可取的，因为在递归函数论中“神谕”（预言）有着十分不同的含义，而在人工智能的语言内“精灵”似乎仍有不同的外形。〕

我们将利用受限的敌手，它在确定某些比较的结果时，其权力是受到限制的，在一个受限的敌手影响下的一个合并方法不知道这些限制，所以它仍然进行必要的比较，尽管比较的结果已被预先确定。例如，在关于定理M的证明中，通过条件（5）限制了所有的结果，然而合并算法不能利用这一事实来免去任何一次比较。

在以下的讨论中，我们将使用的限制可应用于文件的左端和右端。左边的限制通过下列符号表征：

· （意思是没有左边的限制），

\ （意思是所有的结论都必须同 $A_1<B_1$ 一致），

/ （意思是所有的结论都必须同 $A_1>B_1$ 一致）；

右边的限制以下列符号表征：

· （意思是没有右边的限制），

\ （意思是所有的结论都必须同 $A_m<B_n$ 一致），

/ （意思是所有的结论都必须同 $A_m>B_n$ 一致）。

有九种类型的敌手，以 $\nabla M\phi$ 来表示，其中 ∇ 是一个左限制而 ϕ 是一个右限制。例如，一个"\M\"敌手必须指出 $A_1<B_j$ 和 $A_i<B_n$；一个"·M·"敌手是不受限制的。对于某个小的 m 和 n，某些类型的受限制的敌手是不可能有的；当 $m=1$ 时，我们显然不能有一个"\M/"的敌手。

现在让我们来构造用于合并的一个颇为复杂但非常聪明的敌手，它不总产生最优的结果，但它给出了包含大量有趣情况的下界。给定 m、n 和左边及右边的限制 ∇ 和 ϕ，假设要求敌手指出 A_i 或 B_j 哪一个大。这个敌手一般地使用六个策略，它把此问题归结为较少的 $m+n$ 的情况。

策略 $A(k, l)$，适用于 $i\leqslant k\leqslant m$ 以及 $1\leqslant l\leqslant j$。比如说 $A_i<B_j$，而且要求随后的操作把 $\{A_1, \cdots, A_k\}$ 同 $\{B_1, \cdots, B_{l-1}\}$ 以及把 $\{A_{k+1}, \cdots, A_m\}$ 同 $\{B_l, \cdots, B_n\}$ 合并。如果 $p\leqslant k$ 和 $q\geqslant l$，则比较 $A_p:B_q$ 所得的回答将为 $A_p<B_q$；如果 $p>k$ 和 $q<l$，则回答将是 $A_p>p_q$；如果 $p\leqslant k$ 和 $q<l$，则它们将通过（k，$l-1$，∇，·）敌手加以处理；如果 $p>k$ 和 $q\geqslant l$，则通过（$m-k$，$n+1-l$，·，ϕ）敌手加以处理。

策略 $B(k, l)$，适用于 $i\leqslant k\leqslant m$ 和 $1\leqslant l\leqslant j$。比如说 $A_i<B_j$，并且要求随后的操作把 $\{A_1, \cdots, A_k\}$ 同 $\{B_1, \cdots, B_l\}$，以及把 $\{A_{k+1}, \cdots, A_m\}$ 同 $\{B_l, \cdots, B_n\}$ 加以合并，约定 $A_k<B_l<A_{k+1}$。（注意，B_l 出现于有待合并的两个表中。条件 $A_k<B_l<A_{k+1}$ 确保合并一个组不会给出有助于合并另一个组的信息。）在将来比较 $A_p:B_q$ 时，如果 $p\leqslant k$ 和 $q\leqslant l$，将得到结果 $A_p<B_q$；如果 $p>k$ 和 $q\leqslant l$，结果将为 $A_p>B_q$；如果 $p\leqslant k$ 和 $q\leqslant l$，它们将通过一个（k，l，∇，\）敌手来处理；如果 $p>k$ 和 $q\geqslant l$，则通过一个（$m-k$，$n+1-l$，/，ϕ）敌手来处理。

策略 $C(k, l)$，适用于 $i<k\leqslant m$ 和 $1\leqslant l\leqslant j$。比如说 $A_i<B_j$，并且要求随后的操作把 $\{A_1, \cdots, A_k\}$ 同 $\{B_1, \cdots, B_{l-1}\}$ 和 $\{A_k, \cdots, A_m\}$ 同 $\{B_l, \cdots, B_n\}$ 合并，约定 $B_{l-1}<A_k<B_l$。（类似于策略 B，交换 A、B 的作用。）

策略 $A'(k, l)$，适用于 $1\leqslant k\leqslant i$ 和 $j\leqslant l\leqslant n$。比如说 $A_i<B_j$，并且要求把 $\{A_1, \cdots, A_{k-1}\}$ 同 $\{B_1, \cdots, B_l\}$，和 $\{A_k, \cdots, A_m\}$ 同 $\{B_{l+1}, \cdots, B_n\}$ 合并。（类似于策略 A。）

策略 $B'(k, l)$，适用于 $1 \leqslant k \leqslant i$ 和 $j < l \leqslant n$。比如说 $A_i > B_j$，并且要求把 $\{A_1, \cdots, A_{k-1}\}$ 同 $\{B_1, \cdots, B_l\}$ 和 $\{A_k, \cdots, A_m\}$ 同 $\{B_l, \cdots, B_n\}$ 合并，且有 $A_{k-1} < B_l < A_k$。(类似于策略 B。)

策略 $C'(k, l)$，适用于 $1 \leqslant k < i$ 和 $j \leqslant l \leqslant n$。比如说 $A_i > B_j$，并且要求把 $\{A_1, \cdots, A_k\}$ 同 $\{B_1, \cdots, B_l\}$ 以及 $\{A_k, \cdots, A_m\}$ 同 $\{B_{l+1}, \cdots, B_n\}$ 合并，且有 $B_l < A_k < B_{l+1}$。(类似于策略 C。)

由于这些限制，上述策略不可能用于下列情况：

策略	当出现下列情况时不能采用
$A(k, 1)$，$B(k, 1)$，$C(k, 1)$	$\nabla = /$
$A'(1, l)$，$B'(1, l)$，$C'(1, l)$	$\nabla = \backslash$
$A(m, l)$，$B(m, l)$，$C(m, l)$	$\phi = /$
$A'(k, n)$，$B'(k, n)$，$C'(k, n)$	$\phi = \backslash$

命 $\nabla M\phi(m, n)$ 表示合并的极大下界，它是通过上述的某个敌手得到的，当头一个比较数是 $A_i : A_j$ 时，每个策略在其适用的场合给了我们关于这九个函数的一个不等式，有：

$A(k, l)$：$\nabla M\phi(m, n) \geqslant 1 + \nabla M.(k, l-1) \div .M\phi(m-k, n+1-l)$

$B(k, l)$：$\nabla M\phi(m, n) \geqslant 1 + \nabla M\backslash(k, l) + /M\phi(m-k, n+1-l)$

$C(k, l)$：$\nabla M\phi(m, n) \geqslant 1 + \nabla M/(k, l-1) + \backslash M\phi(m+1-k, n+1-l)$

$A'(k, l)$：$\nabla M\phi(m, n) \geqslant 1 + \nabla M.(k-1, l) + .M\phi(m+1-k, n-l)$

$B'(k, l)$：$\nabla M\phi(m, n) \geqslant 1 + \nabla M\backslash(k-1, l) + /M\phi(m+1-k, n+1-l)$

$C'(k, l)$：$\nabla M\phi(m, n) \geqslant 1 + \nabla M/(k, l) + \backslash M\phi(m+1-k, n-l)$

对于固定的 i 和 j，这个敌手将采取一个策略，该策略使得右端给出的下界取极大值；于是，定义 $\nabla M\phi(m, n)$ 为对于 $1 \leqslant i \leqslant m$ 和 $1 \leqslant j \leqslant n$ 所取的这些下界的极小值。当 m 或 n 为 0 时，$\nabla M\phi(m, n)$ 为 0。

例如，假设 $m = 2$ 和 $n = 3$，而且我们的敌手是不受限制的。如果头一个比较是 $A_1 : B_1$，则这个敌手可以采取策略 $A'(1, 1)$，此时需要 $.M.(0, 1) + .M.(2, 2) = 3$ 个进一步的比较。如果头一个比较是 $A_1 : B_2$ 则这个敌手可以采取策略 $B(1, 2)$，此时需要 $.M\backslash(1, 2) + /M.(1, 2) = 4$ 个进一步的比较。不管首先比较哪一对 $A_i : B_j$，这个敌手总可以保证至少必须进行三次进一步的比较，因此 $.M.(2, 3) = 4$。

用手来进行这些计算是不容易的，但是一台计算机却可以相当快地作出 $\nabla M\phi$ 函数表。有某些明显的对称性，例如

$$/M.(m, n) = .M\backslash(m, n) = \backslash M.(n, m) = .M/(n, m) \tag{7}$$

借助于它我们可以把九个函数减少成只有四个 $.M.(m, n)$，$/M.(m, n)$，$/M\backslash(m, n)$ 以及 $/M/(m, n)$ 表 1 指出对于所有 m，$n \leqslant 10$ 所得到的值；我们的合并的敌手已经以这样的方式定义，即

$$.M.(m, n) \leqslant M(m, n) \quad 对于所有 m, n \geqslant 0 \tag{8}$$

这个关系包括定理M作为一个特殊情况，因为当 $|m-n| \leqslant 1$ 时我们的敌手将使用该定理的简单策略。

现在考虑为 M 函数所满足的某些简单关系：

表1　由“敌手”得到的对于合并的下界

	$.M.(m,n)$											$/M.(m,n)$										
	1	2	3	4	5	6	7	8	9	10	n	1	2	3	4	5	6	7	8	9	10	
1	1	2	2	3	3	3	3	4	4	4		1	2	2	3	3	3	3	4	4	4	1
2	2	3	4	5	5	6	6	6	7	7		1	3	4	4	5	5	6	6	7	7	2
3	2	4	5	6	7	7	8	8	9	9		1	3	5	6	7	7	8	8	9	9	3
4	3	5	6	7	8	9	10	10	11	11		1	4	5	7	8	9	9	10	10	11	4
5	3	5	7	8	9	10	11	12	12	13		1	4	6	8	9	10	11	12	12	13	5
6	3	6	7	9	10	11	12	13	14	15		1	4	6	8	10	11	12	13	14	14	6
7	3	6	8	10	11	12	13	14	15	16		1	4	7	9	10	12	13	14	15	16	7
8	4	6	8	10	12	13	14	15	16	17		1	5	7	9	11	13	14	15	16	17	8
9	4	7	9	11	12	14	15	16	17	18		1	5	8	10	11	13	15	16	17	18	9
10	4	7	9	11	13	15	16	17	18	19		1	5	8	10	12	14	15	17	18	19	10
m																						m
	$/M\backslash(m,n)$											$/M/(m,n)$										
1	$-\infty$	2	2	3	3	3	3	4	4	4		1	1	1	1	1	1	1	1	1	1	1
2	$-\infty$	2	4	4	5	5	6	6	7	7		1	3	3	4	4	4	4	5	5	5	2
3	$-\infty$	2	4	6	6	7	8	8	8	9		1	3	5	5	6	6	7	7	8	8	3
4	$-\infty$	2	5	6	8	8	9	10	10	11		1	4	5	7	7	8	9	9	9	10	4
5	$-\infty$	2	5	7	8	10	10	11	12	13		1	4	6	7	9	9	10	11	11	12	5
6	$-\infty$	2	5	7	9	10	12	13	14	14		1	4	6	8	9	11	11	12	13	14	6
7	$-\infty$	2	5	8	10	11	12	14	15	16		1	4	7	9	10	11	13	14	15	15	7
8	$-\infty$	2	6	8	10	12	13	15	16	17		1	5	7	9	11	12	14	15	16	17	8
9	$-\infty$	2	6	9	10	12	14	16	17	18		1	5	8	9	11	13	15	16	17	18	9
10	$-\infty$	2	6	9	11	13	15	16	18	19		1	5	8	10	12	14	15	17	18	19	10
	1	2	3	4	5	6	7	8	9	10	n	1	2	3	4	5	6	7	8	9	10	

$$M(m,\ n)=M(n,\ m) \tag{9}$$

$$M(m,\ n)\leqslant M(m,\ n+1) \tag{10}$$

$$M(k+m,\ n)\leqslant M(k,\ n)+M(m,\ n) \tag{11}$$

$$M(m,\ n)\leqslant \max(M(m,\ n-1)+1,\ M(m-1,\ n)+1) \tag{12}$$

$$M(m,\ n)\leqslant \max(M(m,\ n-2)+1,\ M(m-1,\ n)+2),$$
$$\text{对于}\ m\geqslant 1,\ n\geqslant 2 \tag{13}$$

如果首先比较 $A_1:B_1$，由通常的合并过程即得出关系（12）。通过首先比较 $A_1:B_2$ 即可类似地导出关系（13）；如果 $A_1>B_2$，则需要$M(m,\ n-2)$ 次进一步的比较，但如果 $A_1<B_2$，则我们可以把 A_1 插入到它应有的位置，并把$\{A_2,\ \cdots,\ A_m\}$同$\{B_1,\ \cdots,\ B_n\}$合并。推而广之，通过首先比较 $A_1:B_k$，而且如果 $A_1<B_k$ 便使用二叉查找，我们可以看出

$$M(m,\ n)\leqslant \max\ (M(m,\ n-k)+1,\ M(m-1,\ n)+1+\lceil \log_2 k\rceil)$$
$$\text{对于}\ m\geqslant 1,\ n\geqslant k \tag{14}$$

结果是，对于所有m，$n\leqslant 10$，$M(m,\ n)=.M.(m,\ n)$，所以表1实际上给出了合并的最优值。这可以通过使用（9）～(14）以及在习题8、9和10中给出的$(m,\ n)=(2,\ 8)$，$(3,\ 6)$ 和（5.9）的特殊构造得到证明。

另一方面，我们的敌手并不总是给出最好可能的下界；最简单的例子是$m=3$，$n=11$。其时$.M.(3,\ 11)=9$，但是$M(3,\ 11)=10$。为了看出在这种情况下这个“敌人”在

哪儿“出错”，我们必须研究他作此判断的理由；进一步仔细检查揭示出，如果$(i, j)\neq(2, 6)$，则这个敌手可以找到一个要求 10 次比较的策略；但当$(i, j)=(2,6)$时，没有策略胜过策略$A(2, 4)$，使下界成为$1+.M.(2, 3)+.M.(1, 8)=9$。通过把$\{A_1, A_2\}$同$\{B_1, B_2, B_3\}$以及$\{A_3\}$同$\{B_4, \cdots, B_{11}\}$合并来完成整个合并是必要的但不是充分的，所以在这种情况下下界并不是最佳的。

类似地，可以证明$.M.(2, 38)=10$，而$M(2, 38)=11$，所以我们的敌手并没有好到足以解决$m=2$的情况，但是有一类无穷多个值，对于这类值说来它是杰出的：

定理K

$$M(m, m+2)=2m+1 \quad 对于 m\geqslant 2$$
$$M(m, m+3)=2m+2 \quad 对于 m\geqslant 4$$
$$M(m, m+4)=2m+3 \quad 对于 m\geqslant 6$$

证明　事实上可以以$.M.$代替M来证明这个结果；对于小的m，这些结果已经通过计算机得到，所以可以假定m是足够大的。也可以假定，头一个比较是$A_i:B_j$，其中$i\leqslant\lceil m/2\rceil$。如果$j\leqslant i$则使用策略$A'(i, i)$，对$d$用归纳法，$d\leqslant 4$，得到

$$\begin{aligned}.M.(m, m+d)&\geqslant 1+.M.(i-1, i)+.M.(m+1-i, m+d-i)\\&=2m+d-1\end{aligned}$$

如果$j>i$，则使用策略$A(i, i+1)$，对m用归纳法，得到

$$\begin{aligned}.M.(m, m+d)&\geqslant 1+.M.(i, i)+.M.(m-i, m+d-i)\\&=2m+d-1\end{aligned}$$

定理K的头两个部分是由黄光明和林身于 1969 年得到的。读者可能已经注意到刚才证明的三个公式的一种模式；保罗·斯托克迈耶（Paul Stockmeyer）和姚储枫已经证明，一般地这个模式成立，即由以上的策略所导出的下界足以得到值$M(m, m+d)=2m+d-1$，其中$m\geqslant 2d-2$〔*SIAM J. Computing* 9 (1980), 85—90〕。

上界　现在来考虑$M(m, n)$的上界；好的上界对应于有效的合并算法。

当$m=1$时，合并问题等价于一个插入问题，而且$B_1, \cdots, B_n$之中有$n+1$个位置是A_1可以落进去的。对于这种情况，容易看出，具有$n+1$个外部节点的任何一株扩展的二叉树均对应于某个合并方法的树（见习题 2）！因此，可以选择一株最优二叉树，它实现信息论的下界

$$1+\lfloor\log_2 n\rfloor=M(1, n)=\lceil\log_2(n+1)\rceil \tag{15}$$

当然，二叉查找（6.2.1节）是达到这个值的一个简单方法。

$m=2$的情况是极为有趣的，但相当难。它已经为R. L. 格雷厄姆、黄光明以及林身完全解决了（见习题11, 12, 13）；我们有

$$M(2, n)=\left\lceil\log_2\frac{7}{12}(n+1)\right\rceil+\left\lceil\log_2\frac{14}{17}(n+1)\right\rceil \tag{16}$$

已经看到，当$m=n$时，通常的合并过程是最优的，而当$m=1$时，颇为不同的二叉查找过程是最优的。我们需要的是居于中间的方法，它把通常的合并算法同二叉查找组合在一起，并且保留两者最好的特性，公式（14）提出了由黄光明和林身给出的〔*SIAM. J. Computing* 1 (1972), 31-39〕下列合并算法：

算法H（二叉合并）

H1. 如果m或n为0，则停止。如果$m\leqslant n$，则置$t\leftarrow\lfloor\log_2(n/m)\rfloor$。如果$m>n$，则置$t\leftarrow\lfloor\log_2(m/n)\rfloor$并转向H4。

H2. 比较$A_m:B_{n+1-2^t}$。如果A_m是较小的，则置$n\leftarrow n-2^t$并返回步骤H1。

H3. 利用二叉查找（它恰恰要求t次进一步的比较），把A_m插入到它在$\{B_{n+1-2^t},\cdots,B_n\}$中的应有位置。如果$k$是使得$B_k<A_m$的极大值，则置$m\leftarrow m-1$和$n\leftarrow k$。返回H1。

H4.（步骤H4和H5同H2和H3类似，只是交换了m、n、A、B的作用。）如果$B_n<A_{m+1-2^t}$，则置$m\leftarrow m-2^t$并返回步骤H1。

H5. 插入B_n到它在诸A中的应有位置。如果k是使得$A_k<B_n$的极大值，则置$m\leftarrow k$和$n\leftarrow n-1$，返回H1。

作为这个算法的一个例子，表2示出三个键{087，503，512}同十三个键{061，154，…，908}合并的过程；在这个例子中，需要进行八次比较。

表2　二叉合并的例子

A（黑体元素被比较）	B	输出
{087 503 512}	{061 154 170 275 426 509 612 653 677 703 765 897 908}	
{087 503 512}	{061 154 170 275 426 509 612 653 677}	{703 765 897 908}
{870 503 512}	{061 154 170 275 426 509 612}	{653 677 703 765 897 908}
{087 503 512}	{061 154 170 275 426 509 612}	{653 677 703 765 897 908}
{087 503}	{061 154 170 275 426 509}	{512 612 653 677 703 765 897 908}
{087 503}	{061 154 170 275 426 509}	{512 612 653 677 703 765 897 908}
{087}	{061 154 170 275 426}	{503 509 512 612 653 677 703 765 897 908}
{087}	{061}	{154 170 275 426 503 509 512 612 653 677 703 765 897 908}
	{061	087 154 170 275 426 503 509 512 612 653 677 703 765 897 908}

设$H(m,\ n)$是黄和林的算法所要求的极大比较数。为了计算$H(m,\ n)$，可以假定在步骤H3中$k=n$且在步骤H5中$k=m$，因为通过对m用归纳法，不难证明对所有的$n\geqslant m-1$，$H(m-1,\ n)\leqslant H(m-1,\ n+1)$，于是当$m\leqslant n$时，有

$$H(m,\ n)=\max[H(m,\ n-2^t)+1,\ H(m-1,\ n)+t+1],$$
$$\text{对于 } 2^t m\leqslant n<2^{t+1}m \tag{17}$$

以$2n+\epsilon$代替n，$\epsilon=0$或1，得到

$$H(m,\ 2n+\epsilon)=\max[H(m,\ 2n+\epsilon-2^{t+1})+1,\ H(m-1,\ 2n+\epsilon)+t+2],$$
$$\text{对于 } 2^t m\leqslant n<2^{t+1}m$$

对n用归纳法，得出

$$H(m,\ 2n+\epsilon)=H(m,\ n)+m,\ \text{对于 } m\leqslant n,\ \epsilon=0\text{ 或 }1 \tag{18}$$

容易看出，当$m\leqslant n<2m$时，$H(m,\ n)=m+n-1$；因此重复地应用（18）即得一般的公式

$$H(m,\ n)=m+\lfloor n/2^t\rfloor-1+tm,\ \text{对于 } m\leqslant n,\ t=\lfloor\log_2(n/m)\rfloor \tag{19}$$

这隐含了，对所有的$n\geqslant m$有$H(m,\ n)\leqslant H(m,\ n+1)$证实了我们关于H3这一步的归纳假设。

置$m=\alpha n$和$\theta=\log_2(n/m)-t$，当$n\to\infty$时，给出

$$H(\alpha n,\ n)=\alpha n(1+2^\theta-\theta-\log_2\alpha)+O(1) \tag{20}$$

等式 5.3.1-35 我们知道，$1.9139<1+2^{\theta}-\theta\leqslant 2$；因此，(20) 可以同信息论的下界(3)相比较，黄和林已经证明（见习题17）

$$H(m,\ n)<\left\lceil\log_2\binom{m+n}{m}\right\rceil+\min(m,\ n) \tag{21}$$

堪称为“二元合并”的黄和林算法并不总是给出最优的结果，但它有很大的长处，即它可以相当容易地编成程序。当$m=1$时，该算法归结为“非中心的二叉查找”，当$m\approx n$时，它归结为通常的合并过程，所以它是这两个方法之间的一种卓越的折衷。而且，在许多情况下它是最优的（见习题 16）。

公式（18）提示，M函数本身可以满足

$$M(m,\ n)\leqslant M(m,\ \lfloor n/2\rfloor)+n \tag{22}$$

这实际上是真的（见习题 19）。$M(m,\ n)$ 的表提示了若干其它似乎正确的关系，诸如

$$M(m+1,\ n)\geqslant 1+M(m,\ n)\geqslant M(m,\ n+1),\ 对于\ m\leqslant n \tag{23}$$

$$M(m+1,\ n+1)\geqslant 2+M(m,\ n) \tag{24}$$

但这些不等式尚未得到证明。

习题

1.〔15〕 试求$M(m,\ n)$ 和 5.3.1 节中定义的函数S之间有趣的关系。〔提示：考虑$S(m+n)$。〕

▶2.〔22〕 当$m=1$时，没有多余的比较的每个合并算法都定义了一株具有$\binom{m+n}{m}=n+1$个外部节点的扩展二叉树。试证明：反过来，每一株具有$n+1$个外部节点的扩展二叉树都对应于$m=1$的合并算法。

3.〔$M24$〕 证明对所有n，$.M.(1,\ n)=M(1,\ n)$。

4.〔$M44$〕 对所有m和n，$.M.(m,\ n)\geqslant\left\lceil\log_2\binom{m+n}{m}\right\rceil$吗？

5.〔$M30$〕 证明$.M.(m,\ n)\leqslant .M\backslash(m,\ n+1)$。

6.〔$M26$〕 定理K的证明要求计算机验证大量的情况，怎样能大大地减少这种情况的数目？

7.〔21〕 证明（11）。

▶8.〔24〕 通过找出一个算法，它使用至多六次比较把两个元素同其他八个元素合并起来，来证明$M(2,\ 8)\leqslant 6$。

9.〔27〕 证明至多用七步便可以把三个元素同六个元素合并起来。

10.〔33〕 证明五个元素可以在至多 12 步中同九个元素合并起来。〔提示：根据敌手的经验提议先比较 $A_1:B_2$，然后如果 $A_1<B_2$，则试验 $A_5:A_8$。〕

11.〔$M40$〕（黄光明，林身）对于$k\geqslant 0$命$g_{2k}=\left\lfloor 2^k\cdot\frac{17}{14}\right\rfloor$，$g_{2k+1}=\left\lfloor 2^k\cdot\frac{12}{7}\right\rfloor$，于是有$(g_0,g_1,\ g_2,\cdots)=(1,\ 1,\ 2,\ 3,\ 4,\ 6,\ 9,\ 13,\ 19,27,38,54,\ 77,\cdots)$。证明，在最坏的情况下，它花费七次以上的比较把两个元素同g_t个元素合并；但是两个元素可以在至多七步中同g_t-1个元素合并。〔提示：证明，如果$n\geqslant g_{t-1}$且如果要在t次比较中合并$\{A_1,\ A_2\}$和$\{B_1,\ B_2,\cdots,B_n\}$，则头一步以比较$A_2:B_{g_{t-1}}$为最好。〕

12.〔M21〕 设 $R_n(i,j)$ 是为对不同的对象 $\{\alpha, \beta, X_1, X_2, \cdots, X_n\}$ 排序所需要的最少比较数，给定关系

$$\alpha<\beta,\ X_1<X_2<\cdots<X_n,\ \alpha<X_{i+1},\ \beta>X_{n-j}$$

（当 $i\geqslant n$ 或 $j\geqslant n$ 时条件 $\alpha<X_{i+1}$ 或 $\beta>X_{n-j}$ 为空，因此 $R_n(n,n)=M(2,n)$。）

显然，$R_n(0,0)=0$。证明：当 $0\leqslant i\leqslant n$，$0\leqslant j\leqslant n$，$i+j>0$ 时

$$R_n(i,j)=1+\min\Big(\min_{1\leqslant k\leqslant i}\max(R_n(k-1,j),R_{n-k}(i-k,j))$$

$$\min_{1\leqslant k\leqslant j}\max(R_n(i,k-1),R_{n-k}(i,j-k))\Big)$$

13.〔42〕（R. L. 格雷厄姆）说明习题 12 中的递归关系的解可以表达如下，通过规则

$$G(x)=\begin{cases}1 & 0<x\leqslant\frac{5}{7}\\ \frac{1}{2}-\frac{1}{8}G(8x-5), & \frac{5}{7}<x\leqslant\frac{3}{4}\\ \frac{1}{2}G(2x-1), & \frac{3}{4}<x\leqslant 1\\ 0 & 1<x<\infty\end{cases}$$

对于 $0<x<\infty$，定义函数 $G(x)$。

（见图 38）由于 $R_n(i,j)=R_n(j,i)$ 以及 $R_n(0,j)=M(1,j)$，可以假定 $1\leqslant i\leqslant j\leqslant n$。命 $p=\lfloor\log_2 i\rfloor$，$q=\lfloor\log_2 j\rfloor$，$r=\lfloor\log_2 n\rfloor$，并命 $t=n-2^r+1$。于是

$$R_n(i,j)=p+q+S_n(i,j)+T_n(i,j)$$

其中 S_n 和 T_n 是为 0 或 1 的函数：

$$S_n(i,j)=1\quad 当且仅当\ q<r\ 或(i-2^p\geqslant u\ 且\ j-2^r\geqslant u)$$

$$T_n(i,j)=1\quad 当且仅当\ p<r\ 或(t>\frac{6}{7}2^{r-2}\ 且\ i-2^r\geqslant v)$$

其中 $u=2^pG(t/2^p)$ 且 $v=2^{r-2}G(t/2^{r-1})$。

（这可能是需要解决的最麻烦的递归关系！）

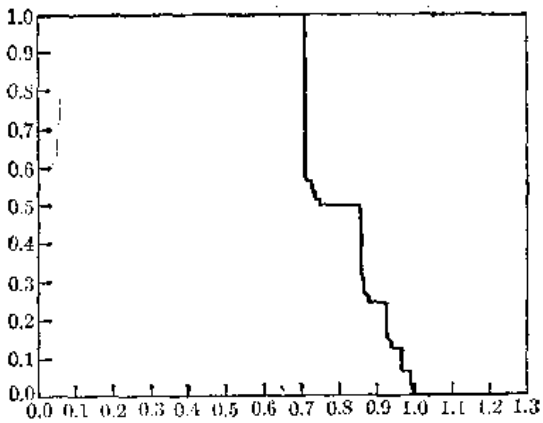


图38 格雷厄姆的函数（见习题13）

14.〔41〕（黄光明）对于 $k\geqslant 3$ 设 $h_{3k}=\lfloor(43/28)\cdot 2^k\rfloor-1$，$h_{3k+1}=h_{3k}+3\cdot 2^{k-2}$，$h_{3k+2}=\lfloor(17/7)\cdot 2^k-6/7\rfloor$，且设初始值被定义成使得 $(h_0, h_1, h_2, \cdots)=(1, 1, 2, 2, 3, 4, 5, 7, 9, 11, 14, 18, 23, 29, 38, 48, 60, 76, 97, 121, 154\cdots)$。证明对所有 t，$M(3,h_t)>t$ 和 $M(3,h_t-1)\leqslant t$，由此确定对于所有 n，$M(3,n)$ 的准确值。

15.〔12〕 二叉合并算法的步骤H1可能要求计算 $\lfloor\log_2(n/m)\rfloor$，说明怎样无须用除

法或对数的计算，即可容易地计算之。

16.〔18〕 当 $1 \leqslant m \leqslant n \leqslant 10$ 时，对于什么样的 m 和 n，黄和林的二叉合并算法是最优的？

17.〔$M25$〕 证明（21）。〔提示：这个不等式不是非常严格的。〕

18.〔$M40$〕 研究二叉合并所使用的平均比较次数。

▶19.〔23〕 证明 M 函数满足（22）。

20.〔20〕 证明如果对所有 $m \leqslant n$，$M(m, n+1) \leqslant M(m+1, n)$，则对所有 $m \leqslant n$，$M(m, n+1) \leqslant 1 + M(m, n)$。

21.〔$M47$〕 证明或否定（23），（24）。

22.〔$M50$〕 研究为合并 m 个事物和 n 个事物所需要的极小平均比较次数。

23.〔$HM30$〕〔E，莱因戈尔德（E. Reingold）〕设 $\{A_1, A_2, \cdots, A_n\}$ 和 $\{B_1, B_2, \cdots, B_n\}$ 是各包含 n 个元素的集合。试考虑一个算法，它试图仅仅通过比较元素的相等性，来测试这两个集合的相等性。于是，此算法提出对某个 i 和 j "$A_i = B_j$ 吗？" 的问题，而且它依赖答案为是或否。

试通过定义一适当的敌手，证明任何这样的算法在它最坏的情况下必须至少进行 $\frac{1}{2} n(n+1)$ 次比较。

*5.3.3 极少比较选择

当我们寻找最好的过程以选择 n 个元素的第 t 个最大者时，出现了一类类似的有趣问题。

这个问题的历史可回溯到C. L. 道奇森（C. L. Dodgson）牧师关于草地网球锦标赛的有趣的（尽管是严肃的）试验，它出现在 *St, James's Gazette*, August 1, 1883, pp. 5-6 上。道奇森当然比刘易斯·卡罗尔（Lewis Carroll）更有名，他关心的是在网球锦标赛中过去（而且现在仍然如此）颁发奖金的不公平方式。例如，考虑图 39 中所示的标号为 01, 02, …, 32 的 32 个选手之间进行的一场典型的"淘汰赛"。在"决赛"中，选手 01 击败了 05，所以显然选手 01 是冠军而且他应获头等奖。但在如下的问题上却出现了不公正，即选手 05 通常获得二等奖，尽管他可能不是第二好的选手。在这场锦标赛中，即使你比一半的选手都要差，却仍有可能获得二等奖。事实上，如同道奇森所注意到的，第二个最好的选手只有当他在锦标赛开始时和冠军处于不同的两半中时才能获二等奖；若有 2^n 个选手，则这以 $2^{n-1}/(2^n-1)$ 的概率出现，所以不好的选手几乎有一半的机会可获得二等奖！如果半决赛的失败者（图 39 中的 25 和 17 号选手）竞争三等奖，则第三个最好者接受三等奖更是极少可能的。

因此，道奇森提出用传递性排次序的方法来设计一个锦标赛，以确定真正的第二和第三好的选手（换言之，如果选手 A 击败选手 B，而选手 B 击败选手 C，则假定 A 将击败 C）。他设计了一个过程，在这个过程中，允许失败者进一步比赛，直到确实知道他们劣于三个其他选手为止。图 40 中示出了道奇森方案的一个例子，它是有待同图 39 配合进行的补充比赛。该方案尽量把迄今具相等记录的选手配对比赛，并且避免两个已被同一个选手击败的选手进行比赛。例如，在第一轮中，16 败于 11 和 13 败于 12；在第二轮中 16 败于

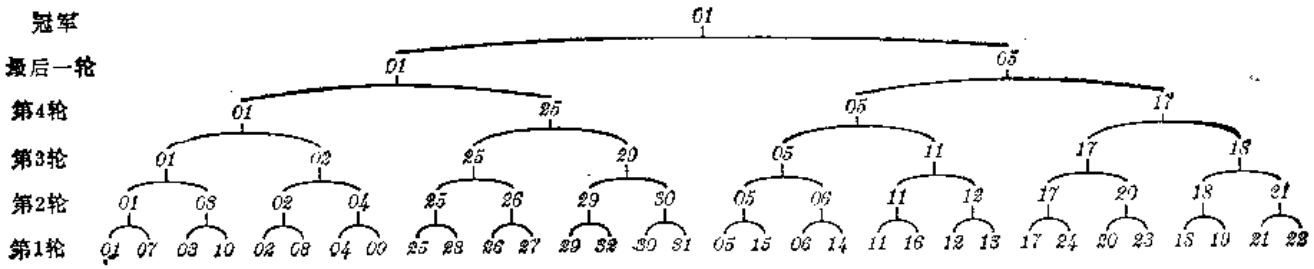


图39　32个选手的淘汰赛

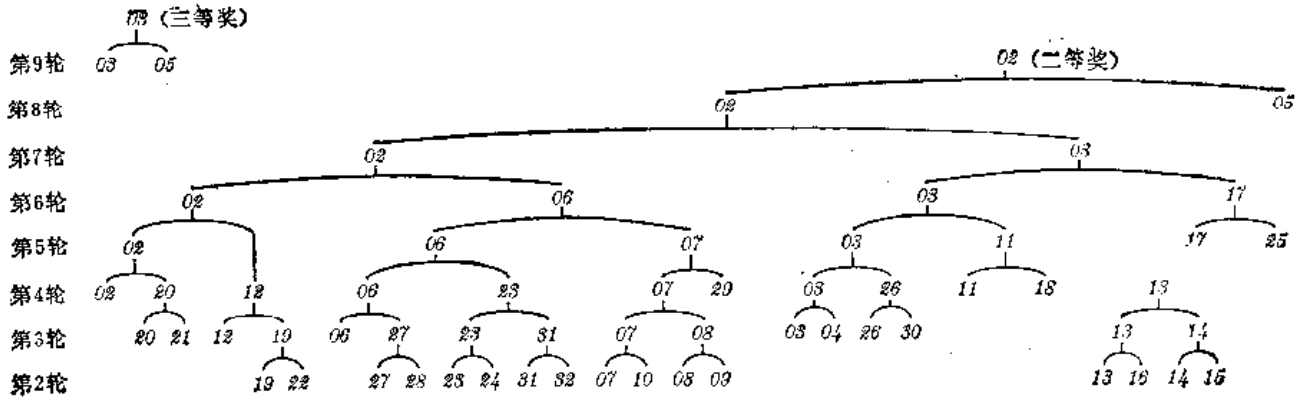


图40　刘易斯·卡罗尔的草地锦标赛（同图39合在一起进行）

13，由此就取消了 16，因为已经知道他低于 11、12 和 13 了。在第三轮中，不允许 19 同 21 进行比赛。因为他们俩都为 18 所击败，故我们不能自动地取消 19 对 21 的失败者。

假如我们能报导刘易斯·卡罗尔的锦标赛确实是最优的，那就好了。但可惜情况并非如此。他在 1883 年 7 月 23 日的日记中指出，他用了大约六个小时来排方案，他觉得“现在我们的（网球）赛期已过去了这么多，因此比起写出来，倒不如立即就进行比赛更好些。”他的过程中比较次数不是最节省的，而且该过程没有足够精确地描述出来，从而可以被称为一个算法。另一方面，从平行计算的观点看，它有某些颇为有趣的方面，而且对于一个网球锦标赛说来，它似乎是一个卓越的计划，因为他取得了某些戏剧性的效果；例如，他确定，两个决赛选手应该不参加第五轮比赛，而参加在第六轮中和第七轮中进行的附加比赛。但是锦标赛的指挥者大概认为这个提案太逻辑化了，因此看来从未试用过卡罗尔的系统，而代之使用一个“种子”方法，来把最好的选手分在树的不同部分。

在 1929～1930 年的数学研究班中，雨果·斯坦豪斯提出了当共有 $n \geqslant 2$ 个选手时，为确定在一次锦标赛中的头一个和第二个最好的选手，所需要的网球比赛的极小场次问题。J. 施赖尔（J. Schreier）〔*Mathesis Polska* 7 (1932)，154-160〕给出了至多需要 $n-2+\lceil \log_2 n \rceil$ 场比赛的一个过程，基本上使用了我们称作树选择排序方法中的头两个阶段同样的方法（参考 5.2.3 节，图 23），避免涉及 $-\infty$ 的多余比较。施赖尔还宣称，$n-2+\lceil \log_2 n \rceil$ 是最好的，但他的证明如同 J. 斯鲁贝克基（J. Stupecki）试图另给的证明一样〔*Colloquium Mathematicum* 2 (1951)，286-290〕是不正确的。三十二年之后，S. S. 基斯里特森（S. S. Kislitsyn）最后发表了一个正确的但相当复杂的证明〔Сибирский Мат Журнал，5 (1964) 557-564〕。

命 $V_t(n)$ 表示为确定 n 个元素的第七个最大者所需要的极小比较次数，$1 \leqslant t \leqslant n$，并命 $W_t(n)$ 是为了确定最大者，第二个最大者，…以及第 t 个最大者等全部 t 个元素所需要的极小比较次数。由对称性，我们有

$$V_t(n)=V_{n+1-t}(n) \tag{1}$$

而且显然有

$$V_1(n)=W_1(n) \tag{2}$$

$$V_t(n) \leqslant W_t(n) \tag{3}$$

$$W_n(n)=W_{n-1}(n)=S(n) \tag{4}$$

在 5.2.3 节中我们已经发现

$$V_1(n)=n-1 \tag{5}$$

事实上，有一个关于这个事实的惊人地简单的证明，因为在一次锦标赛中，除了冠军外，每个选手都必然至少输了一场！通过扩展这一思想，并使用一个“敌手”，就可以不太困难地证明施赖尔—基斯里特森定理：

定理 S 当 $n \geqslant 2$ 时，$V_2(n)=W_2(n)=n-2+\lceil \log_2 n \rceil$。

证明 假设有 n 个选手参加了一次锦标赛。这个锦标赛通过某种给定的过程确定了第二个最好的选手，并命 a_j 是输了 j 场或更多场比赛的选手数。于是，比赛的总场数是 $a_1+a_2+a_3+\cdots$。如果不同时确定出冠军来的话，我们不能确定第二个最好的选手（参考习题

2），所以用以前的论证说明 $a_1 = n - 1$。为完成这个证明，将证明定有某个比赛结果的序列，使得 $a_2 \geqslant \lceil \log_2 n \rceil - 1$。

假设在锦标赛结束时，冠军已经迎战（并击败了）p 个选手；这些人当中有一个是第二个最好者，而且其他的必然至少已经另外输了一场，所以 $a_2 \geqslant p - 1$。因此可以通过这样一个方法来完成这个证明，即通过构造一个敌手，该敌手规定冠军至少必须同 $\lceil \log_2 N \rceil$ 个其他人较量，他由此来判定比赛的结果。

设此敌手在下列情况下断言 A 比 B 更好，即，如果 A 以前未败过，而 B 至少输了一次，或者如果两者都未败过而此时 B 比 A 赢得少些。在其它情况下，敌手可以作出同某个偏序相一致的任何判断。

考虑整个锦标赛的结果，这次锦标赛的角逐已经为这样一个敌手所决断。即当且仅当 $A = B$ 或 A 强过头一个击败 B 的选手时我们说“A 强于 B”（按这个说法，对每个选手来说，只有第一次败绩是关键的，他随后的比赛则被忽略。按照敌手的规则，任何首先击败另一个人的人都必须以前未败过）。由此得出，赢得头 p 场的人，在这 p 场比赛的基础上至多强过 2^p 个选手（这对于 $p = 0$ 是显然的，而对于 $p > 0$，第 p 次比赛是迎战以前败过了的或至多强过 2^{p-1} 个选手的某个人）。冠军强于每个人，所以他必须至少赢得了 $\lceil \log_2 n \rceil$ 场比赛。

于是，找第二个最大者的问题在极小化极大的意义下完全解决了。事实上，习题 6 说明，当已知某集合中诸元素的一个任意偏序时，有可能给出求该集合的第二个最大元素所需要的极小比较次数的简单公式。

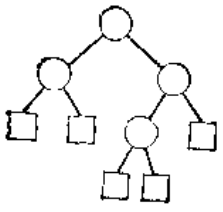
如果 $t > 2$ 如何？ 在上边引证的文章中，基斯里特森考虑过 t 的更大的值并证明

$$W_t(n) \leqslant n - t + \sum_{n+1-t<j\leqslant n} \lceil \log_2 j \rceil \text{对于 } n \geqslant t \tag{6}$$

对于 $t = 1$ 和 $t = 2$，已经看到，在这个公式中等式实际上成立；对于 $t = 3$，它能稍加改进（见习题 21）。

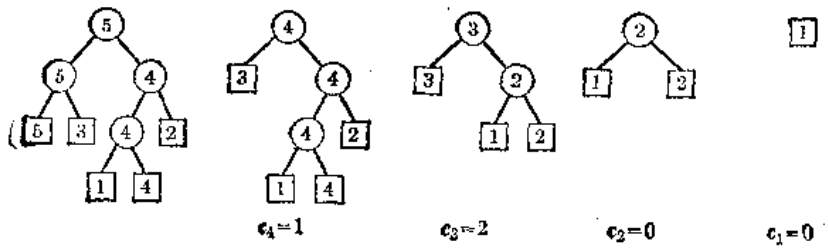
我们将说明树选择的头 t 个阶段至多需要 $n - t + \sum_{n+1-t<j\leqslant n} \lceil \log_2 j \rceil$ 次比较（忽略涉及 $-\infty$ 的所有比较），以证明基斯里特森定理。有意思的是，当 $t = n - 1$ 和 $t = n$ 时，（6）的右边等于 $B(n)$（见等式 5.3.1-3）；因此，树选择和二元插入在排序问题上给出同样的上界，尽管它们是十分不同的方法。

命 α 是具有 n 个外部节点的扩展二叉树，并命 π 是 $\{1, 2, \cdots, n\}$ 的一个排列。以对称的次序从左到右地把 π 的元素放进外部节点，并象在树选择中那样按着淘汰锦标赛的规则填入内部节点。当得到的树被施以重复的选择操作时，它就定义了一个序列 $c_{n-1}, c_{n-2}, \cdots, c_1$，其中 c_j 是当元素 $j + 1$ 已被 $-\infty$ 代替时为把元素 j 引到树的根部所需要的比较次数。例如，如果 α 是树



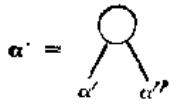
（7）

而且如果 $\pi = 5\ 3\ 1\ 4\ 2$，则逐次得到如下的树



如果 π 换成 3 1 5 4 2，则序列 $c_4c_3c_2c_1$ 将是 2110。容易看出 c_1 总是零。

命 $\mu(\alpha, \pi)$ 是由 α 和 π 确定的多重集合 $\{c_{n-1}, c_{n-2}, \cdots, c_1\}$。如果



而且如果元素 1 和 2 不同时在 α' 中出现，也不同时在 α'' 中出现，则容易看出对于适当的排列 π' 和 π''

$$\mu(\alpha, \pi) = (\mu(\alpha', \pi') + 1) \uplus (\mu(\alpha'', \pi'') + 1) \uplus \{0\} \tag{8}$$

其中 $(\mu + 1)$ 表示对 μ 的每个元素加 1 得到的多重集合。另一方面，如果元素 1 和元素 2 都出现于 α' 中，则有

$$\mu(\alpha, \pi) = (\mu(\alpha', \pi') + \varepsilon) \uplus (\mu(\alpha'', \pi'') + 1) \uplus \{0\}$$

其中 $(\mu + \varepsilon)$ 表示把 μ 的某些元素加 1 而其它元素加 0 得到的某个多重集合。当 1 和 2 都出现于 α'' 中时，类似的公式成立。我们说多重集合 μ_1 高于 μ_2，如果 μ_1 和 μ_2 两者有相同的元素个数，而且对于所有的 k，μ_1 的第 k 个最大的元素大于或等于 μ_2 的第 k 个最大的元素；而且，在下述意义下定义 $\mu(2)$ 为"最高的" $\mu(\alpha, \pi)$：对所有的排列 π，$\mu(\alpha)$ 都高于 $\mu(\alpha, \pi)$，而对于某个 π，$\mu(\alpha) = \mu(\alpha, \pi)$。上边的公式表明

$$\mu(\square) = \emptyset, \quad \mu(\underset{\alpha' \quad \alpha''}{\bigcirc}) = (\mu(\alpha') + 1) \uplus (\mu(\alpha'') + 1) \uplus \{0\} \tag{9}$$

因此，$\mu(\alpha)$ 是从 α 的根到它的内部节点的所有距离的多重集合。

只要读者遵循上面的思路，现在就能看出，我们已为基斯里特森定理（6）的证明作好了准备；因为 $W_t(n)$ 小于或等于 $n-1$ 加 $\mu(\alpha)$ 的 $t-1$ 个最大元素，其中 α 是在树选择排序中使用的任何树，当

$$\begin{aligned}\mu(\alpha) &= \{\lfloor \log_2 1 \rfloor, \lfloor \log_2 2 \rfloor, \cdots, \lfloor \log_2 (n-1) \rfloor\} \\ &= \{\lceil \log_2 2 \rceil - 1, \lceil \log_2 3 \rceil - 1, \cdots, \lceil \log_2 (n-1) \rceil - 1\}\end{aligned} \tag{10}$$

时，可以取 α 为具有 n 个外部节点的完备二叉树（参考 2.3.4.5 节）。当考虑这个多重集合的 $t-1$ 个最大的元素时，就得到公式（6）。

基斯里特森定理给出了对于 $W_t(n)$ 的一个好的上界；他注意到 $V_3(5) = 6 < W_3(5) = 7$，但是一般地说，他未能得到 $V_t(n)$ 的一个更好的界。A·哈迪安和M·索

贝尔发现了使用替代选择，而不用树选择（见5.4.1节），来解决这个问题的一个办法。他们的公式〔Univ of Minnesota, Dept. of Statistics, report 121 (May, 1969)〕

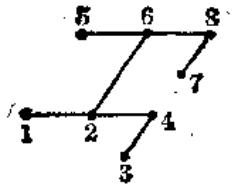
$$V_t(n) \leqslant n - t + (t-1)\lceil \log_2(n+2-t) \rceil, \quad n \geqslant t \tag{11}$$

与（6）的区别在于：在（6）的和中的每个元素都已换成最小的元素。

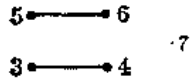
通过使用下列构造，即可证明哈迪安和索贝尔定理（11）：首先建立对于 $n-t+2$ 个项目的一次淘汰锦标赛的二叉树（这花费 $n-t+1$ 次比较）。最大的项目大于 $n-t+1$ 个其它项目，所以它不可能是第七个最大的。不管它出现在树的哪一个外部节点处，都以保留的 $t-2$ 个元素之一代替它，在由此所得的 $n-t+2$ 个元素中，找出其最大的元素；这至多要求 $\lceil \log_2(n+2-t) \rceil$ 次比较，因为仅仅需要重新计算树中的一条路径。对于保留的每个元素重复这个操作，总共重复 $t-2$ 次。最后，以 $-\infty$ 代替当前最大的元素，并确定剩下的 $n+1-t$ 个元素中的最大者；这至多要求 $\lceil \log_2(n+2-t) \rceil - 1$ 次比较，而且它把原来集合的第 t 个最大元素引向树的根。把这些比较加起来就得到（11）。

在关系（11）中，每当 $n+1-t$ 给出更好的值时（如同 $n=6$，$t=3$ 时那样），我们在式子右边当然将以 $n+1-t$ 来代替 t，奇怪的是，这个公式对于 $V_7(13)$ 给出了比对于 $V_6(13)$ 更小的界。对于 $n \leqslant 6$，（11）中的上界是精确的，但当 n 和 t 变大时，有可能得到对于 $V_t(n)$ 的好得多的估计。

例如，下列漂亮的方法（由戴维·G.多林（David G. Doren 给出）可以用来说明 $V_4(8) \leqslant 12$。设元素为 $X_1, \cdots, X_8$；首先比较 $X_1:X_2$ 和 $X_3:X_4$ 以及两个胜利者，并对 $X_5:X_6$ 和 $X_7:X_8$ 以及它们的胜利者也同样进行比较。重新标出这些元素，使得 $X_1<X_2<X_4>X_3$，$X_5<X_6<X_8>X_7$，然后比较 $X_2:X_6$；由对称性，假定 $X_2<X_6$，于是我们有配置



（现在 X_1 和 X_8 已脱离比赛，我们必须求 $\{X_2, \cdots, X_7\}$ 的第三个最大者。）比较 $X_2:X_7$ 并丢弃较小者；在最坏的情况下，我们有 $X_2<X_7$，而且必须求出



的第三个最大者。这可在另外 $V_3(5)-2=4$ 次比较中完成，因为（11）的 $V_3(5)=6$ 过程是通过比较两个不相交的元素对开始的。

这种类型的其它技巧可用来产生表1所示的结果；尚未发明一般的方法，对 $V_4(9)$、$V_5(10)$ 及其对偶 $V_6(9)$、$V_6(10)$ 所示的值可作进一步的改进。注意 $V_3(10) \leqslant 14$，故（11）在 $t=3$ 时并不总给出等式。$V_4(7)=10$ 这一事实表明，当 $n=7$ 时，（11）中的偏离已经等于2。

戴维·盖·柯克帕特里克（David G. Kirkpatrick）已经得到了对于选择问题的相当好的下界〔Ph. D. thesis, U. of Toronto, 1974〕他构造了一个敌手，它证明

$$V_t(n) \geqslant n + t - 3 + \sum_{0 \leqslant j \leqslant t-2} \lceil \log_2((n+2-t)/(t+j)) \rceil \quad n \geqslant 2t-1 \qquad (12)$$

柯克帕特里克证明了对于所有的 $n \geqslant 50$，$V_3(n) = n + \lceil \log_2((n-1)/2.5) \rceil + \lceil \log_2((n-1)/4) \rceil$，并由此确立了当 $t=3$ 时的精确行为（参考习题 22）。

表 1　对于 $V_t(n)$ 已知的最好上界

n	$V_1(n)$	$V_2(n)$	$V_3(n)$	$V_4(n)$	$V_5(n)$	$V_6(n)$	$V_7(n)$	$V_8(n)$	$V_9(n)$	$V_{10}(n)$
1	0									
2	1	1								
3	2	3	2							
4	3	4	4	3						
5	4	6	6	6	4					
6	5	7	8	8	7	5				
7	6	8	10	10*	10	8	6			
8	7	9	11	12	12	11	9	7		
9	8	11	12	14	15*	14	12	11	8	
10	9	12	14*	15	16**	15**	15	14*	12	9

* 在这些情况下，习题10-12给出改进等式（11）的构造。

** 见 K. Noshita, Trans. of the IECE of Japan, E59, 12 (DEC. 1976), 17-18。

一个线性方法　当 n 是奇数而且 $t = \lceil n/2 \rceil$ 时，第 t 个最大的（以及第 t 个最小的）元素称作中值。按照（11），我们可以在 $\approx \frac{1}{2} n \log_2 n$ 次比较中求出 n 个元素的中值；尽管只要求少得多的信息，但这大约只比排序快一倍。若干年来许多人一致地致力于寻求当 t 和 n 很大时对于（11）的改进。最后，在 1971 年，曼纽尔·布卢姆（Manuel Blum）发现了仅仅需要 $O(n \log \log n)$ 个步骤的一个方法。布卢姆解决这个问题的途径，提示了一类新技术，它导致了R．里夫斯特（R．Rivest）和R．塔里简（R．Tarjan）给出的下列构造〔J. Comp. and Sys. Sci. 7 (1973), 448-461〕：

定理L　当 $n \geqslant 32$ 时，对于 $1 \leqslant t \leqslant n$，$V_t(n) \leqslant 15n - 163$。

证明　当 n 很小时，这定理是显然的，因为对于 $32 < n \leqslant 2^{10}$ 有 $V_t(n) \leqslant S(n) \leqslant 10n \leqslant 15n - 163$。通过加上至多 13 个虚构的“$-\infty$”元素，对于某个整数 $q \geqslant 73$，我们可以假定 $n = 7(2q+1)$。下列方法可以用来选择第 t 个最大的元素：

步骤 1　把这些元素分成每组 7 个元素的 $2q+1$ 组，对每组排序。这至多花费 $13(2q+1)$ 次比较。

步骤 2　求在步骤 1 中得到的 $2q+1$ 个中值元素的中值，称它为 x。通过对 q 用归纳法可证，这至多花费 $V_{q+1}(2q+1) \leqslant 30q - 148$ 次比较。

步骤 3　不同于 x 的 $n-1$ 个元素现在已经分成为三个集合（见图 41）：

$4q+3$ 个大于 x 的元素（区域B）；

$4q+3$ 个小于 x 的元素（区域C）；

$6q$ 个元素，它们同 x 的关系是未知的（区域A、D）。

通过进行 $4q$ 次附加的比较，我们就能确切地说出区域A和D中哪些元素小于 x（首先拿

每个三元组的中间元素来和 x 比较)。

步骤 4　对某个 r，现在找到了 r 个大于 x 的元素和 $n-1-r$ 个小于 x 的元素。如果 $t=r+1$，则 x 是答案；如果 $t<r+1$，则需要求 r 个大的元素中第 t 个最大的；如果 $t>r+1$，则需要求 $n-1-r$ 个小元素中的第 $(t-1-r)$ 个最大的。要点是 r 和 $n-1-r$ 两者都小于或者等于 $10q+3$（区域A和D加上B或C的大小）。因此，通过对 q 用归纳法可证，这个步骤至多需要 $15(10q+3)-163$ 次比较。

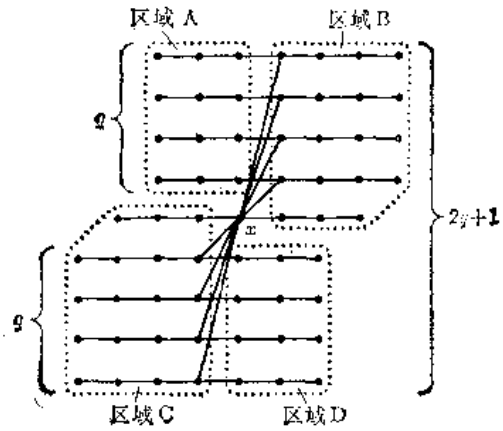


图41　里夫斯特和塔里简的选择算法（$q=4$）

比较的次数至多为

$$
\begin{aligned}
&13(2q+1)+30q-148+4q\\
&\quad+15(10q+3)-163\\
&=15(14q-6)-163
\end{aligned}
$$

由于我们是从至少 $14q-6$ 个元素开始的，证明就完成了。

定理L证明了选择总可在"线性时间"内完成，即证明了，$V_t(n)/n$ 是有界的。当然，这个证明所使用的方法稍微有些粗糙，因为它放过了步骤 4 中的好的信息。关于这一问题的研究已经得到更漂亮的界；例如，A. 舍恩哈格（A. Schönhage），M. 彼得森（M. Peterson）及N. 皮普宾格(N. Pippenger)〔J. Comp. Sys. Sci. 13(1976)，184-199〕已经证明，为求平均值所需要的极大比较次数至多是 $3n+O(n\log_2 n)^{5/4}$。另一方面，V. 昔拉特已经得到对于这个问题的 $1.75n$ 的渐近下界（参看 *Proc. IEEE Conf. Switching and Automata Theory* 14(1973)，70-81〕；习题 25 是他的结果的一个推广。

平均数　除了把极大比较次数极小化外，我们还可以要求一个算法，对于随机次序下的对象，使得平均比较次数极小化。和通常一样，这个问题要难得多，而且甚至对于 $t=2$ 的情况也尚未解决。克劳德·皮卡德在他的"*Theorie des Questionnaires*"（1965）一书中提到了这个问题，而米尔顿·索贝尔作了深入的剖析〔*Univ. of Minnesota, Dept. of Statistics, reports* 113 *and* 114 (November, 1968)〕。

索贝尔构造了图 42 的过程，它平均仅仅使用 $6\frac{1}{2}$ 次比较就求得了六个元素的第二个最大者。在最坏的情况下，需要进行 8 次比较，而这比 $V_2(6)=7$ 要坏；事实上，D. 霍伊所作的一个穷尽的计算机查找已经证明，对于这个问题的最好的过程，如果限定至多作 7 次比较，则平均使用 $6\frac{26}{45}$ 次比较。于是，在同时地极小化极大值和极小化平均值两种意义下，似乎没有求六个元素的第二个最大元素的最优过程。

命 $\bar{V}_t(n)$ 表示为求 n 个元素的第七个最大者所需要的极小平均比较数。如同D. 霍伊所计算的那样，表 2 示出了对于小的 n 的一些精确值。

R. W. 弗洛伊德在 1970 年发现，n 个元素的中值，平均说来，可以仅仅通过 $\frac{3}{2}n+O(n^{2/3}\log_2 n)$ 次比较求得（见习题 13）。他证明了，事实上

$$\overline{V}_t(n) \leqslant n + t + f(n);\ 其中 \lim_{n \to \infty} f(n)/n = 0 \tag{13}$$

表 2　进行选择所需的极小平均比较次数

n	$\overline{V}_1(n)$	$\overline{V}_2(n)$	$\overline{V}_3(n)$	$\overline{V}_4(n)$	$\overline{V}_5(n)$	$\overline{V}_6(n)$	$\overline{V}_7(n)$
1	0						
2	1	1					
3	2	$2\frac{2}{3}$	2				
4	3	4	4	3			
5	4	$5\frac{4}{15}$	$5\frac{13}{15}$	$5\frac{4}{15}$	4		
6	5	$6\frac{1}{2}$	$7\frac{7}{18}$	$7\frac{7}{18}$	$6\frac{1}{2}$	5	
7	6	$7\frac{149}{210}$	$8\frac{509}{630}$	$9\frac{32}{105}$	$8\frac{509}{630}$	$7\frac{149}{210}$	6

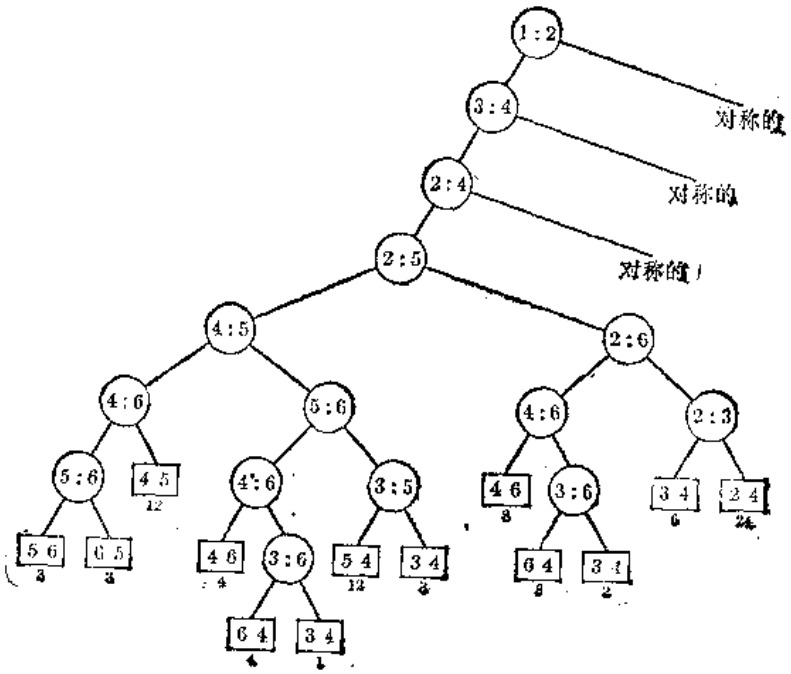


图42　选择$\{X_1, X_2, X_3, X_4, X_5, X_6\}$的第二个最大者的过程，平均使用 $6\frac{1}{2}$次比较。每个"对称的"分枝和它的兄弟相同，只是名字按某种适当的方式排列过。当已知X_j是第二个最大者和X_k是最大者时，外部节点包含"$j\ k$"；写在每个外部节点下的数字表示通向这个节点要做多少次排列。

根据对 $t=2$ 的一种索贝尔构造的推广，戴维·W.马图拉（David W.Matula）〔待出〕已用另一种方法证明

$$\overline{V}_t(n)\leqslant n+t\lceil\log_2 t\rceil(11+\ln\ln n) \tag{14}$$

因此，对于固定的 t，平均的工作量可减少到仅仅用 $n+O(\log_2\log_2 n)$ 次比较。对于 $V_t(n)$ 还没有得到真正令人满意的下界。

习题

1.〔15〕 在刘易斯·卡罗尔的锦标赛中（图 39 和 40），为什么选手 13被淘汰，尽管在第三轮中他赢了？

▶2.〔M25〕 证明，在通过一系列的比较，找出了 n 个元素的第 t 个最大者之后，我们还知道哪 $t-1$ 个元素是大于它的，哪 $n-t$ 个元素是小于它的。

3.〔M21〕 证明，对于 $1\leqslant t\leqslant n$ $V_t(n)\geqslant V_t(n-1)+1$。

4.〔M20〕 证明，$W_t(n)\geqslant\lceil\log_2 n^{\underline{t}}\rceil$，其中 $n^{\underline{t}}=n(n-1)\cdots(n+1-t)$。

5.〔10〕 证明 $W_3(n)\leqslant V_3(n)+1$。

6.〔M26〕（R.W.弗洛伊德）给定 n 个不同的元素 $\{X_1,\cdots,X_n\}$ 以及某些（i，j）对的关系 $X_i<X_j$，我们希望找出第二个最大的元素。如果已经知道对 $j\neq k$ 有 $X_i<X_j$ 及 $X_i<X_k$，则 X_i 不可能是第二个最大的元素，所以它可被消去。得到的关系有一个如同

的形式，即，可以由一个向量（l_1，l_2，…，l_m）表示的 m 组元素；第 j 组包含 l_j+1 个元素，已知它们当中的一个大于其它的。例如，上边的配置可以由向量（3，2，5，0，1）来描述；当不知道任何关系时，n 个分量都是 0。

命 $f(l_1,l_2,\cdots,l_m)$ 是为找出这样一个偏序集合的第二个最大的元素所需要的极小比较数，证明

$$f(l_1,l_2,\cdots,l_m)=m-2+\lceil 2^{l_1}+2^{l_2}+\cdots+2^{l_m})\rceil。$$

〔提示：须证明，最好的策略总是比较两个最小组的最大元素，直到把 m 化到 1 为止；对 $l_1+l_2+\cdots+l_m+2m$ 使用归纳法。〕

7.〔M20〕 证明（8）。

8.〔M21〕 基斯里特森公式（6）是以使用具有 n 个外部节点的完备二叉树进行树选择排序为基础的。对任何 t 和 n，有没有一个以某个其它树为基础的给出更好界的树选择方法。

▶9.〔20〕 使用哈迪安和索贝尔的替代选择方法〔参阅（11）〕，画出在至多六步中即找出五个元素的中值的比较树。

10.〔38〕（儿平野下）证明可以在至多 14 步之内求出九个元素的中值，而这 14 步的头七步和多林的方法一样。

11. 〔21〕（哈迪安和索贝尔）证明：$V_3(n)\leqslant V_3(n-1)+2$。〔提示：由去掉$\{X_1, X_2, X_3, X_4\}$之最小者开始。〕

12. 〔35〕 证明可以在至多 10 步之内求出七个元素的中值。

13. 〔$HM28$〕（R. W. 弗洛伊德）证明，如果使用一个递归地定义的方法，从找出$\{X_1,\cdots, X_{n^{2/3}}\}$的中值元素开始，则可以通过平均$\frac{3}{2}n+O(n^{2/3}\log_2 n)$次比较来找出$\{X_1,\cdots, X_n\}$的中值。

▶14. 〔$M20$〕（M. 索贝尔）设$U_t(n)$是为找出n个元素的第t个最大者（但无须知道它们的相对次序）所需要的极小比较数。证明$U_2(5)\leqslant 5$。

15. 〔22〕（I. 波尔（I. Pohl））假设我们对空间（而不是时间）的极小化感兴趣。如果每个元素都填入一个字，而且如果这些元素逐个地输入到一个寄存器中，则为了计算n个元素的第t个最大者，至少需要多少存储单元？

16. 〔25〕（I. 波尔）证明，至多使用$\lceil\frac{2}{3}n\rceil-2$次比较，便可以找出$n$个元素的集合的极大和极小；而且这个比较数已不可能被降低。〔提示：这样一个算法的任何阶段都可表示作一个四元组（a, b, c, d），其中a元素尚未被比较，b已经获胜过但是尚未失败过，c已经失败过而尚未获胜过，d有胜有负。试构造一个适当的敌手。〕

17. 〔20〕（R. W. 弗洛伊德）试证使用至多$\lceil\frac{3}{2}n\rceil-k-l+\sum_{n+1-k<j\leqslant n}\lceil\log_2 j\rceil+\sum_{n+1-l<j\leqslant n}\lceil\log_2 j\rceil$次比较，就有可能依次地选择$n$个元素的一个集合的$k$个最大和$l$个最小的元素。

18. 〔$M20$〕 如果在定理 L 的证明中使用了大小为 5，而不是 7 的组，则将得到什么定理？

19. 〔$M44$〕 把表 2 扩充到$n=8$。

20. 〔$M47$〕 当$n\to\infty$时，$\overline{V}_2(n)-n$的渐近值是多少？

21. 〔25〕（林身）证明当$k\geqslant 3$时$W_3(2^k+2)\leqslant 2^k+2k$。

22. 〔$M47$〕 对所有n确定$V_3(n)$和$W_3(n)$的值。

23. 〔$M49$〕 当$n\to\infty$时，$V_{\lceil n/2\rceil}$的渐近值是多少？

24. 〔$M48$〕 当$n\to\infty$时，$\overline{V}_{\lceil n/2\rceil}(n)$的渐近值是多少？

25. 〔$M32$〕（A. 舍恩哈盖（A. Schönhage, 1974）（a）在习题 14 的记号下，证明对于$n\geqslant 3$，$U_t(n)\geqslant\min(2+U_t(n-1), 2+U_{t-1}(n-1))$。提示：构造一个敌手，该敌手的规则是：只要当前的偏序不是由分量 0 或 0 − 0 组成，便从n归约到$n-1$。（b）类似地，证明对于$n\geqslant 5$，$U_t(n)\geqslant\min(2+U_t(n-1), 3+U_{t-1}(n-1), 3+U_t(n-2))$，通过构造涉及分量 ∘, ∘—∘, ∘>∘, ∘=∘ 的一个敌手进行。（c）因此对于$1\leqslant t\leqslant n/2$，有$U_t(n)\geqslant n+t+\min(\lfloor(n-t)/2\rfloor, t)-3$。（d）当用$V$或$W$代替$U$时，也可应用在（a）和（b）中的不等式，由此证明表 1 中若干项目的最优性。

5.3.4 排序网络

在这节中，我们将研究一种受限制类型的排序，由于它的应用和丰富的理论基础使它特别有趣。新的限制是坚持齐性的比较序列。齐性的定义是：每次比较K_i和K_j后，随后

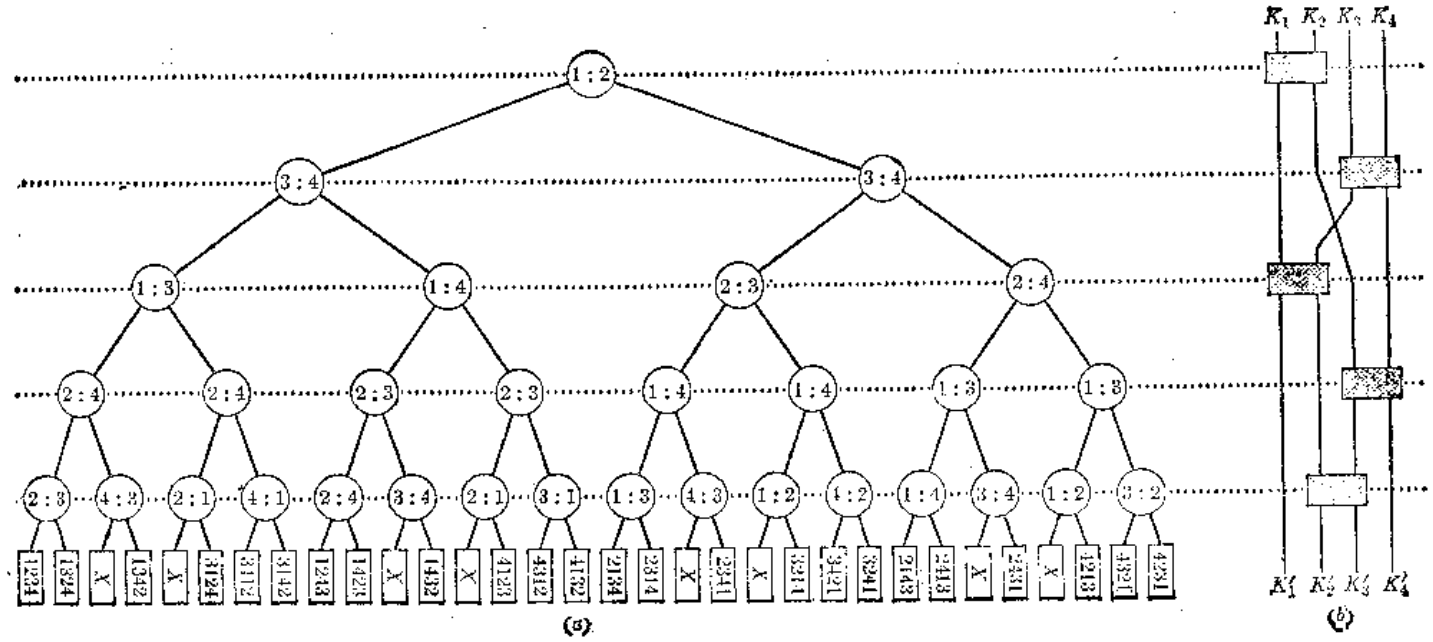


图 43

（a）齐性的比较树；（b）对应的网络。

的比较对于 $K_i<K_j$ 和 $K_i>K_j$ 是完全一样的，仅将 i 和 j 互换即可。图 43（a）示出了一株满足这个齐性条件的比较树（注意，对于每层都进行同样多比较，所以在进行了 m 次比较后，有 2^m 个结果；由于 $n!$ 不是 2 的幂，由此得出，某些分枝必须进行多余的比较，以确保这株树的所有对应的分枝都恰当地排好序。这里“多余”的意思是：它们的子树之一实际上是不可能出现的。）

由于这样一株树被从顶部到下部的每条路径唯一地确定，这样一个方案最容易表示成图 42（b）中那样的一个网络。在这样一个网络中，方框表示“比较模块”，它有两个输入（表示成从上边进入到这个模块的直线）和两个输出（表示引向下边的直线）；左边的输出是两个输入的较小者，而右边的输出是较大者。在这个网络的下部，K_1' 是 $\{K_1, K_2, K_3, K_4\}$ 的最小者，K_2' 是第二个最小者，等等。不难证明，任何排序网络在上述的意义下都对应齐性的比较，而任何齐性树都对应于比较模块的一个网络。

附带说一句，我们要指出，从工程的观点来看，比较模块是相当容易制造的。例如，假设直线包含二进制数，其中每一个单位时间有一个二进位数进入每个模块，最先进入的是最高位。每个比较模块都有三个状态，并按如下方式进行：

时间 t		时间（$t+1$）	
状态	输入	状态	输出
0	0 0	0	0 0
0	0 1	1	0 1
0	1 0	2	0 1
0	1 1	0	1 1
1	x y	1	x y
2	x y	2	y x

开始时，所有的模块都在状态 0 并且输出 0 0。一个模块只要它的输入不同就进入状态 1 或状态 2。如果在直线 K_1' 和 K_4' 上附加一个适当的延迟元件，则在时间 t 时在图 43（b）顶上开始送来的数，将被排好次序，并从 $t+3$ 时开始由底下输出。

为了展开排序网络的理论，不妨以一种如图 44 中所示的稍微不同的方式来表示它们。这里，数从左边进入，并用两条直线间的垂直连接表示比较模块；每个比较块必要时交换其输入量，使得在通过这个比较块之后较大的数出现在下边的线上。在这个图形的右边所有的数都是从上到下有序的。

我们以前关于最优排序的研究，集中在使比较次数极小化上，而很少或者根本不注意它引起的任何数据移动或可能需要的判定结构的复杂性。在这方面，排序网络有某些优点，因为数据可以保留在 n 个地点，而且判定结构是“直线”式的；不需要记住以前比较的结果——因为方案是预先就固定好的。排序网络的另一个重要的优点是，有可能重叠若干个操作，同时地执行它们（在一台适当的机器上）。例如，当允许同时的非重叠的比较时，图43和 44 中的五步可以叠合成为三步，因为头两个和第二个两个可以联合起来；在本节稍后将剖析这个排序网络

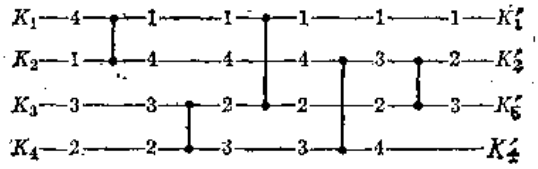


图44　当对序列〈4，1，3，2〉排序时表示图43的网络的另一种方式

的性质。因此，网络可能是非常有用的，尽管还不完全清楚对很大的 n 能否构造有效的 n 个元素的排序网络；我们将会发现，为了保持齐性的判定结构，需要许多附加的比较。

当给定一个 n 元素的网络时，存在两种简单的方法来构造一个 $n+1$ 个元素的排序网络，或用插入原理，或用选择原理。图 45(a) 说明在头 n 个元素已经排序之后，如何把第 $n+1$ 个元素插入到它应有位置中去；而这个图的 (b) 部分说明在对剩下的一些元素排序之前，如何来选择最大的元素。重复应用图 45(a)，就得到了类似于直接插入排序的网络（算法 5.2.1S），而重复应用图 45(b)，就产生了类似于气泡排序的网络（算法 5.2.2B）。图 46 给出了相应的六元素的网络。有意思的是，当允许同时操作时，两个方法都归结为同一个“三角形”的有 $2n-3$ 个阶段的过程（图 47）。

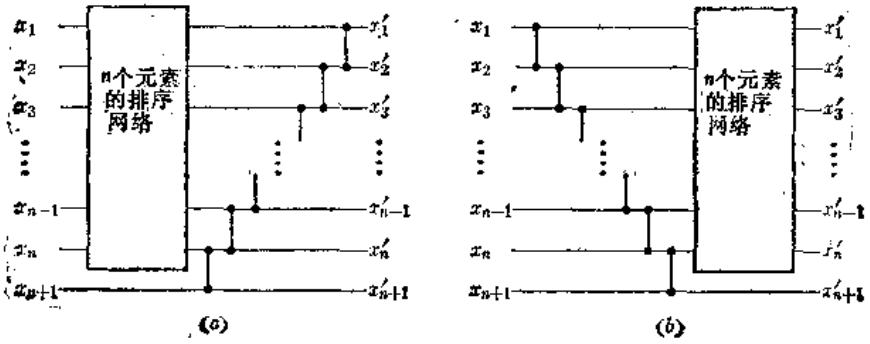


图45　从 n 排序器构造($n+1$)排序器

(a) 插入；(b) 选择。

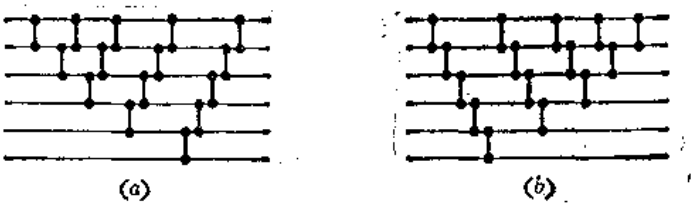


图46　通过重复应用图 45 得到的类似于初等内部排序图式的网络

(a) 直接插入；(b) 气泡排序。

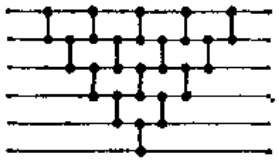


图47　在并行的情况下直接插入＝气泡排序！

容易证明：图 43 和图 44 的网络将把四个数的任何集合排成为有序的，因为头四个比较器把最小和最大的元素放置到正确的位置，最后的比较器把剩下的两个元素顺序排序。但是要知道一个给定的网络是否对所有可能的输入序列进行排序，并不总是那样容易的；例如：

都是正确的四个元素的排序网络，但是它们正确性的证明却非显然。考虑 n 个不同元素的所有排列，得到 $n!$ 个 n 元素网络，逐一试验之，这样肯定够了。但是事实上，我们可以通过少得多的试验获得之：

定理Z （0—1原理）如果具有 n 个输入线的一个网络把0和1的所有 2^n 个序列排成为非减次序，则它将把任何 n 个数的任意序列排成为非减的次序。

证明 （这是布里西尤斯定理的特殊情况，习题5.3.1-12）如果 $f(x)$ 是任何单调函数，且每当 $x \leqslant y$ 时，$f(x) \leqslant f(y)$，而且如果一个给定的网络把 $\langle x_1, \cdots, x_n \rangle$ 变换成 $\langle y_1, \cdots, y_n \rangle$，则容易看出这个网络将把 $\langle f(x_1), \cdots, f(x_n) \rangle$ 变换成 $\langle f(y_1), \cdots, f(y_n) \rangle$。如果对某个 i，$y_i > y_{i+1}$，则考察单调函数 f，它把所有 $< y_i$ 的数变成0，以及把所有 $\geqslant y_i$ 的数变成1；这定义了诸0和1的一个序列 $\langle f(x_1), \cdots, f(x_n) \rangle$，它不被这个网络所排序。因此，如果所有的0—1序列被排序，则对于 $1 \leqslant i < n$，我们有 $y_i \leqslant y_{i+1}$。

在构造排序网络中，0—1原理是十分有帮助的。作为一个非显然的例子，我们可以推导巴切尔的“合并交换”（算法5.2.2M）的一种推广形式。其思想是通过独立地对头 m 个元素和最后 n 个元素排序来实现对全部 $m+n$ 个元素的排序。然后对结果应用 (m, n) 合并网络。一个 (m, n) 合并网络可以归纳地构造如下：

a）如果 $m=0$ 或 $n=0$，则这个网络是空的。如果 $m=n=1$，则这个网络是单个比较模块。

b）如果 $m\ n>1$，命有待合并的诸序列是 $\langle x_1, \cdots, x_m \rangle$ 和 $\langle y_1, \cdots y_n \rangle$。合并“奇序列” $\langle x_1, x_3, \cdots, x_{2\lceil m/2 \rceil - 1} \rangle$ 和 $\langle y_1, y_3, \cdots y_{2\lceil n/2 \rceil - 1} \rangle$，得到排序结果 $\langle v_1, v_2, \cdots, v_{\lceil m/2 \rceil + \lceil n/2 \rceil} \rangle$；合并“偶序列” $\langle x_2, x_4, \cdots, x_{2\lceil m/2 \rceil} \rangle$ 和 $\langle y_2, y_4, \cdots, y_{2\lceil n/2 \rceil} \rangle$，得到结果 $\langle w_1, w_2, \cdots, w_{\lceil m/2 \rceil + \lceil n/2 \rceil} \rangle$。最后，应用比较交换操作

$$w_1 : v_2,\ w_2 : v_3,\ w_3 : v_4, \cdots,\ w_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor} : v^* \tag{1}$$

到序列

$$\langle v_1, w_1, v_2, w_2, v_3, w_3, \cdots, v_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor}, w_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor}, v^*, v^{**} \rangle \tag{2}$$

上，这个结果将是排好序的，(1)（这里如果 m 和 n 都是偶数则 $v^* = v_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor + 1}$ 不存在，而除非 m 和 n 都是奇数，$v^{**} = v_{\lfloor m/2 \rfloor + \lfloor n/2 \rfloor + 2}$ 也不存在；（1）指出比较模块的总数是 $\lfloor (m+n-1)/2 \rfloor$。）

巴切尔的 (m, n) 合并网络称为奇偶合并。按照这些原理构造的一个 $(4, 7)$ 合并示于图48中。

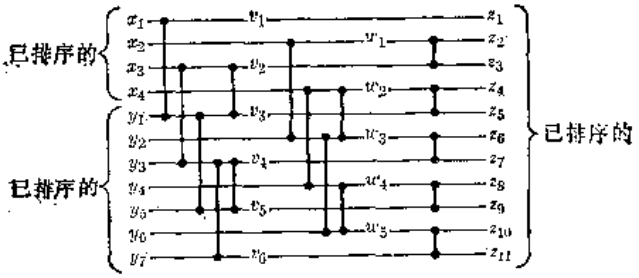


图48 当 $m=4$ 和 $n=7$ 时的奇偶合并

为了证明这个相当奇异的合并过程实际上是行得通的，当 $mn>1$ 时，我们使用0—1原理，对所有的0和1的序列来测试它。在初始的 m 排序和 n 排序后，对某个 k 和 l，序列 $\langle x_1, \cdots, x_m \rangle$ 将由 k 个0后边接上 $m-k$ 个1组成，而序列 $\langle y_1, \cdots, y_n \rangle$ 将是 l 个0后边接上 $n-l$ 个1。因此序列 $\langle v_1, v_2, \cdots \rangle$ 将恰由 $\lceil k/2 \rceil + \lceil l/2 \rceil$ 个0，后边接一些1组成；而 $\langle w_1, w_2, \cdots, \rangle$ 将由 $\lfloor k/2 \rfloor + \lfloor l/2 \rfloor$ 个0，后边接一些1组成。现在这里的要点是

$$(\lceil k/2 \rceil + \lceil l/2 \rceil) - (\lfloor k/2 \rfloor + \lfloor l/2 \rfloor) = 0,\ 1 \text{ 或 } 2 \tag{3}$$

如果这个差是0或1，则序列（2）已经有序了，如果这个差是2，则（1）中的比较交

换之一就把问题全都解决了。这就完成了证明。(注意，0—1 原理把合并问题从$\binom{m+n}{m}$种情况的考虑归结为仅仅 $(m+1)(m+1)$ 种，它们由两个参数 k 和 l 表示。)

设 $C(m, n)$ 是对于 m 和 n 在奇偶合并中使用的比较模块个数，不计初始的 m 排序和 n 排序；我们有

$$C(m, n)=\begin{cases} mn, & \text{如果 } mn\leqslant 1 \\ C(\lceil m/2\rceil, \lceil n/2\rceil)+C(\lfloor m/2\rfloor, \lfloor n/2\rfloor) \\ \quad +\lfloor (m+n-1)/2\rfloor, & \text{如果 } mn>1 \end{cases} \tag{4}$$

一般地说，这不是 m 和 n 的特别简单的函数，但是注意 $C(1, n)=n$ 和

$$C(m+1, n+1)-C(m, n)=1+C(\lfloor m/2\rfloor+1, \lfloor n/2\rfloor+1)-C(\lfloor m/2\rfloor, \lfloor n/2\rfloor), \text{ 如果 } mn\geqslant 1$$

可以导出关系

$$C(m+1, n+1)-C(m, n)=t+2+\lfloor n/2^{t+1}\rfloor, \quad \text{如果 } n\geqslant m\geqslant 1 \text{ 和 } t=\lfloor \log_2 m\rfloor \tag{5}$$

因此

$$C(m, m+r)=B(m)+m+R_m(r) \quad \text{对于 } m\geqslant 0, \ r\geqslant 0 \tag{6}$$

其中 $B(m)$ 是等式 5.3.1-3 的“二叉插入”函数 $\sum_{1\leqslant k\leqslant m}\lceil \log_2 k\rceil$，而其中 $R_m(r)$ 表示下列级数的头 m 项之和

$$\left\lfloor \frac{r+0}{1}\right\rfloor+\left\lfloor \frac{r+1}{2}\right\rfloor+\left\lfloor \frac{r+2}{4}\right\rfloor+\left\lfloor \frac{r+3}{4}\right\rfloor+\left\lfloor \frac{r+4}{8}\right\rfloor+\cdots+\left\lfloor \frac{r+j}{2^{\lfloor \log_2 j\rfloor+1}}\right\rfloor+\cdots \tag{7}$$

特别是，当 $r=0$ 时，我们有重要的特殊情况

$$C(m, n)=B(m)+m \tag{8}$$

进而，如果 $t=\lceil \log_2 m\rceil$，则

$$R_m(r+2^t)=R_m(r)+1\cdot 2^{t-1}+2\cdot 2^{t-2}+\cdots+2^{t-1}\cdot 2^0+m=R_m(r)+m+t\cdot 2^{t-1}$$

因此 $C(m, m+2^t)-C(m, n)$ 有一个简单的形式，并且

$$C(m, n)=\left(\frac{t}{2}+\frac{m}{2^t}\right)n+O(1), \text{ 对固定的 } m, \ n\to\infty, \ t=\lceil \log_2 m\rceil \tag{9}$$

$O(1)$ 项是 n 的一个最终周期函数，其周期长度为 2^t。按等式(8)和习题 5.3.1-15，当 $n\to\infty$ 时，$C(n, n)$ 的渐近值是 $n\log_2 n$。

极小比较网络 命 $\hat{S}(n)$ 是 n 个元素的排序网络中所需要的比较器的极小个数；显然 $\hat{S}(n)\geqslant S(n)$，其中 $S(n)$ 是在一个不加限制的排序过程中所需要的极小比较次数(参考 5.3.1 节)。我们有 $\hat{S}(4)=5=S(4)$，所以当 $n=4$ 时，新的限制并不引起效率的损失；当 $n=5$ 时，结果已经是 $\hat{S}(5)=9$ 而 $S(5)=7$。确定 $\hat{S}(n)$ 的问题似乎比确定 $S(n)$ 的问题更为困难；甚至连 $\hat{S}(n)$ 的渐近行为也还不知道。

回顾这个问题的历史是有趣的，因为在这里迈出每一步都是很艰难的。大约在 1954 年，P. N. 阿姆斯特朗(P. N. Armstrong)、R. J. 纳尔逊(R. J. Nelson)以及 D. J.

奥康纳（D. J. O'Connor）首先剖析了排序网络〔见U. S Patent3029413〕；他们证明 $\hat{S}(n+1)\leqslant\hat{S}(n)+n$。用其专利代办人的话说，"通过使用这个技巧，有可能使用较少的双线排序开关来设计经济的 n 线排序开关"。他们给出了对于 $4\leqslant n\leqslant 8$ 的特殊构造，分别地使用5，9，12，18和19个比较器。随后，纳尔逊同R. C. 博斯（R. C. Bose）合作，在1960年以前，设计了一个用于构造排序网络的一般过程，证明对于所有的 n，$\hat{S}(2^n)\leqslant 3^n-2^n$；因此 $\hat{S}(n)=O(n^{\log_2 3})=O(n^{1.585})$。博斯和纳尔逊在 *JACM* 9（1962），282-296上发表了他们有趣的方法，文中他们推测它是最好的。T. N. 席巴德〔*JACM* 10(1963)，142-150〕发现了一个类似的但稍简单些的方法，它使用相同的比较数。由此更增强了这个推测。

1964年，R. W. 弗洛伊德和D. E. 克努特发现了解决这个问题的新方法，导致了形如 $\hat{S}(n)=O(n^{1+c/\sqrt{\log_2 n}})$ 的一个渐近界值。K. E. 巴切尔独立地发现了上边概述的一般的合并技巧；使用了由

$$c(1)=0,\quad C(n)=C(\lceil n/2\rceil)+C(\lfloor n/2\rfloor+C(\lceil n/2\rceil,\ \lfloor n/2\rfloor)\quad 对于\ n\geqslant 2 \tag{10}$$

定义的比较器个数，他证明（见习题5.2.2-4）

$$C(2^t)=(t^2-t+4)2^{t-2}-1$$

而且由此得出 $\hat{S}(n)=O(n(\log n)^2)$。巴切尔、弗洛伊德及克努特都过了一段时间才发表他们的构造〔*Notices of the Amer. Math. Soc.* 14 (1967)，283；*Proc. AFIPS Spring Joint Computer Conference*)32(1968)307-314〕。

若干人已经找出一些方法来改进巴切尔的合并交换的构造所用的比较次数；下表说明当前已知的 $\hat{S}(n)$ 的最好上界：

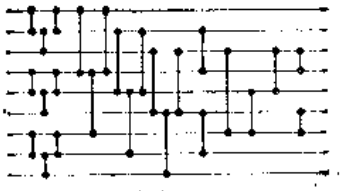
$n=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$c(n)=$	0	1	3	5	9	12	16	19	26	31	37	41	48	53	59	63
$\hat{S}(n)\leqslant$	0	1	3	5	9	12	16	19	25	29	35	39	46	51	56	60

(11)

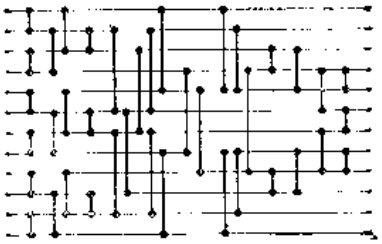
由于对 $8<n\leqslant 16$，$\hat{S}(n)<C(n)$，故对所有 $n>8$，合并交换是非最优的。当 $n\leqslant 8$ 时，合并交换在比较器个数方面等价于博斯和纳尔逊的构造。弗洛伊德和克努特于1964～1966年证明了当 $n\leqslant 8$ 时，$\hat{S}(n)$ 的上列值是精确的〔见*A Survey of Combinatorial Theory*（North-Holland，1972）163-172〕；当 $n>8$ 时 $\hat{S}(n)$ 的值仍然是未知的。

图49示出了获得上述 $\hat{S}(n)$ 值的构造。以有趣的三路合并为基础的 $n=9$ 的网络，是1964年由R. W. 弗洛伊德建立的；它的正确性可以通过使用习题27中描述的一般原理来证实。1969年，A. 瓦克斯曼（A. Waksman）把输入当作{1，2，…，10}的排列，并且试图尽可能减少在每个阶段中每一直线上出现的可能的数值的数目，同时保留某种对称性，由此发现了 $n=10$ 的网络。

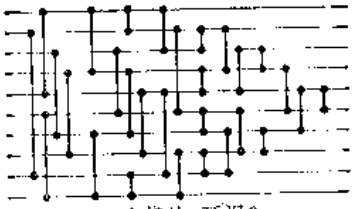
1969年，G. 夏皮罗（G. Shapiro）建立了对于16个元素的62个比较器的排序网络，这是颇为令人吃惊的。因为巴切尔的方法（63次比较）似乎是当 n 为2的乘方时的最好情况。在获知夏皮罗的构造之后不久，M. W. 格林（M. W. Green）通过求出在图49中的包含60个比较的排序器，又进一步增加了人们的惊异；格林的构造的头一部分是相当容



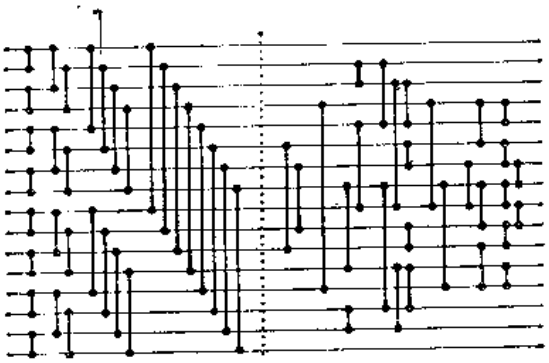
$n=9$　25个模块，延迟10



$n=12$　39个模块，延迟10



$n=10$　29个模块，延迟9



$n=16$　60个模块，延迟10

图49　有效的排序网络

易理解的；在对虚线左边进行了 32 次比较/交换之后，这些直线可以以$\{a, b, c, d\}$的 16 个子集按下面的方式来标号：每当 s 是 t 的一个子集时，标号为 s 的直线包含一个数，这个数小于或等于标号为 t 的直线的内容。习题 32 中进一步讨论了这里的排序状态。然而，在格林的网络的随后的层次上所作的比较变得越来越神秘，而且至今未有人看出怎样来推广这个构造，以便相应地得到对于 n 的更高值的有效网络。

夏皮罗和格林也发现了 $n=12$ 的网络。当 $n=11$，13，14 或 15 时，通过把 $n+1$ 网络中最底下的线和触及该线的所有比较器一起撤销，就可以建立好的网络。

当前已知的 $n\to\infty$的最好的网络见 R.L. 德赖斯代尔(R.L. Drysdale) Ⅱ 1973 年在诺克斯(Knox)学院的优秀的大学学年设计；他的网络要求$\frac{1}{4}n(\log_2 n)^2-(371/960)n\log_2 n+O(n)$个比较器；特别是，他的构造产生 $\hat{S}(256)\leqslant 3657$〔见 *SIAM J. Computing* 4 (1975)，264-270〕。

极小时间网络　在排序网络的物理实现中，以及在并行计算机上，有可能同时进行非重叠的比较交换；因此自然地提出了延迟时间极小化的问题。稍事考虑就可看出，如果把

一条路径定义为任何自左到右的路线，这个路线可能在比较器处改换所走的直线，则一个排序网络的延迟时间，等于通过网络的任何“路径”上的比较次数中之极大者。可以在每个比较器上放置一个序列编号，来指出它可以执行的最早时刻；这个编号比在它之前的各比较器的序列编号的极大值大 1（见图 50(a)；此图的（b）部分示出重画的同一网络，其中每一个比较完成于最早可能的时刻)。

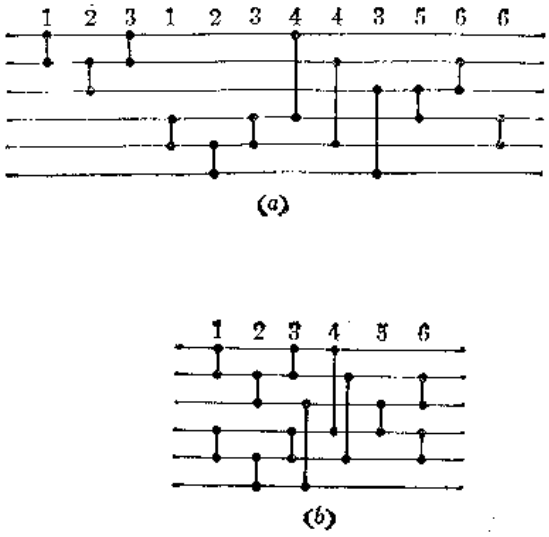


图50　在最早可能的时刻进行每个比较

上面所描述的巴切尔奇偶合并网络花费 $T_B(m,\ m)$ 个时间单位，其中$T_B(m,\ 0)=T_B(0,\ n)=0$，$T_B(1,\ 1)=1$，且

$$T_B(m,\ n)=1+\max(T_B(\lfloor m/2\rfloor,\ \lfloor n/2\rfloor),\ T_B(\lceil m/2\rceil,\ \lceil n/2\rceil)\quad 对于\ mn\geqslant 2$$

我们可以使用这些关系，通过归纳法来证明 $T_B(m,\ n+1)\geqslant T_B(m,\ n)$；因此对于$mn\geqslant 2$，$T_B(m,\ n)=1+T_B(\lceil m/2\rceil,\ \lceil n/2\rceil)$，并由此得出

$$T_B(m,\ n)=1+\lceil\log_2\max(m,\ n)\rceil\qquad 对于\ mn\geqslant 1 \tag{12}$$

由此，习题 5 证明，巴切尔排序方法的延迟时间为

$$\binom{1+\lceil\log_2 n\rceil}{2} \tag{13}$$

命 $\hat{T}(n)$ 是在 n 个元素的任何排序网络中，可达到的极小延迟时间。有可能来改进上面描述的某些网络，使得它们有较少的延迟时间，但不使用更多的比较器，如同图 51 中对于 $n=6$ 和 $n=9$ 以及习题 7 对于 $n=10$ 所示的那样。如果如同图 51 中对于 $n=10$、12 和 16 的值得注意的网络那样，我们增加一个或两个额外的模块，则还可以达到更小的延迟时间。这些构造，对于小的 n，产生了 $\hat{T}(n)$ 的下列上界

$$\begin{array}{lcccccccccccccccc} n= & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 & 16 \\ \hat{T}(n)\leqslant & 0 & 1 & 3 & 3 & 5 & 5 & 6 & 6 & 7 & 7 & 8 & 8 & 9 & 9 & 9 & 9 \end{array} \tag{14}$$

对于 $n \leqslant 8$，这里给出的值已知是精确的（见习题 4）。图 51 中的网络值得仔细研究，因为它们的排序能力决不是一眼能看出来的；这是在 1969～1971 年间由夏皮罗（对于 $n=6$，9，12）和 D. 范·沃勒斯（D. Van Voorhis）（n：10，16）发现的。

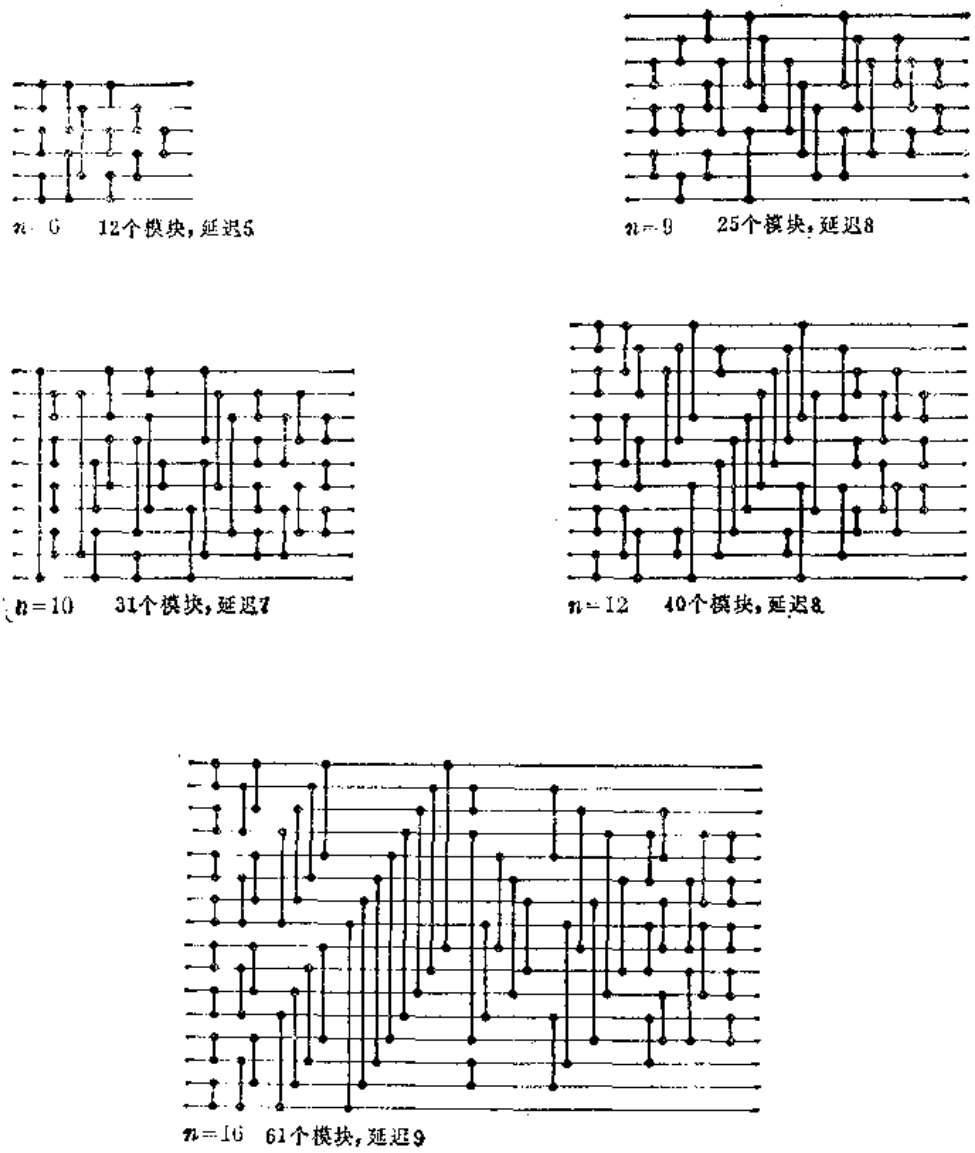


图51 当并行地执行比较时，非常快的排序网络

合并网络 命 $\hat{M}(m, n)$ 表示在一个网络中所需要的极小比较模块数，这个网络把 m 个元素 $x_1 \leqslant \cdots \leqslant x_m$ 同 n 个元素 $y_1 \leqslant \cdots \leqslant y_n$ 加以合并以形成有序序列 $z_1 \leqslant \cdots \leqslant z_{m+n}$。现在，还没有发现比上述描述的奇偶合并更优越的合并网络，因此（6）中的函数 $C(m, n)$ 就表示对于 $\hat{M}(m, n)$ 已知的最好上界。

R. W. 弗洛伊德已经发现了求这个合并问题的下界的一种有趣的方法。

定理F 对于所有 $n \geqslant 1$，$\hat{M}(2n, 2n) \geqslant 2\hat{M}(n, n) + n$。

证明 考虑具有 $\hat{M}(2n, 2n)$ 个比较模块的网络，它有能力对所有的输入序列（z_1，…，z_{4n}）排序，使得 $z_1 \leqslant z_3 \leqslant \cdots z_{4n-1}$ 和 $z_2 \leqslant z_4 \leqslant \cdots \leqslant z_{4n}$。可以假定每个模块都对某个 $i < j$ 以（$\min(z_i, z_j)$，$\max(z_i, z_j)$）代替（z_i，z_j）（见习题 16）。因此诸比较器可以分成三类：

a） $i \geqslant 2n$ 和 $j \leqslant 2n$

b） $i > 2n$ 和 $j < 2n$

c） $i \leqslant 2n$ 和 $j > 2n$

类（a）必须至少包含 $\hat{M}(n, n)$ 个比较器，因为 z_{2n+1}，z_{2n+2}，…，z_{4n} 当开始合并时可能已经处于它们最后的位置了；类似地，在类（b）中至少有 $\hat{M}(n, n)$ 个比较器，进而，输入序列〈0，1，0，1，…，0，1〉说明类（c）至少包含 n 个比较器，因为 n 个 0 必须从 $\{z_{2n+1}, \cdots, z_{4n}\}$ 移到 $\{z_1, \cdots, z_n\}$。

重复使用定理F可证明 $\hat{M}(2^n, 2^m) \geqslant \frac{1}{2}(m+2)2^m$；因此 $\hat{M}(n, n) \geqslant \frac{1}{2} n \log_2 n + O(n)$。没有网络限制的合并，只需要 $M(n, n) = 2n - 1$ 个比较；因此我们已经证明，通过网络进行合并本质上比通常的合并更困难。奇偶合并表明，$\hat{M}(n, n) \leqslant C(n, n) = n \log_2 n + O(n)$，这样，就知道了 $\hat{M}(n, n)$ 的渐近行为，顶多只差一个因子 2（当 $n \leqslant 5$ 时 $\hat{M}(n, n)$ 的精确值是已知的，见习题 9）。姚期智和姚储枫已经证明 $\hat{M}(2, n) = C(2, n) = \left\lceil \frac{3}{2} n \right\rceil$，而且对于 $m \leqslant n$，$\hat{M}(m, n) \geqslant \frac{1}{2} n \log_2 (m+1)$〔JACM 23 (1976)，566-571〕。

双调排序 当允许进行同时的比较时，在等式（12）中已经看到，当 $1 \leqslant m \leqslant n$ 时，奇偶合并使用 $\lceil \log_2(2n) \rceil$ 个延迟时间单位。巴切尔已经想出了用于合并的另一类型的网络，称作双调排序器。尽管它要求更多的比较模块，但它把延迟时间降低到 $\lceil \log_2(m+n) \rceil$。

我们说 p 个数的一个序列〈z_1，…，z_p〉是双调的，如果对于某个 k，$1 \leqslant k \leqslant p$，$z_1 \geqslant \cdots \geqslant z_k \leqslant \cdots \leqslant z_p$（请对照通常的“单调”序列的定义）。一个 p 阶的双调排序器是一个有能力把长度为 p 的任何双调序列排成为非减次序的比较网络。把 $x_1 \leqslant \cdots \leqslant x_m$ 同 $y_1 \leqslant \cdots \leqslant y_n$ 合并的问题，是双调排序问题的一个特殊情况，因为合并可以通过对序列〈x_m，…，x_1，y_1，…y_n〉应用一个 $m+n$ 阶的双调排序器来完成。

注意，若序列〈z_1，…，z_p〉是双调的，则它的所有子序列也都是双调的；在巴切尔发现了奇偶合并网络之后不久他就发现了：我们可以以任何类似的方式构造一个阶为 p 的双调排序器，方法是首先独立地对双调子序列〈z_1，z_3，z_5，…〉和〈z_2，z_4，z_6，…〉进行排序，然后比较和交换 $z_1 : z_2$，$z_3 : z_4$（它的证明见习题 10）。如果 $C'(p)$ 是对应的比较模块数，则我们有

$$C'(p) = C'(\lceil p/2 \rceil) + C'(\lfloor p/2 \rfloor) + \lfloor p/2 \rfloor \quad 对于\ p \geqslant 2 \tag{15}$$

而且延迟时间显然是 $\lceil \log_2 p \rceil$。图 52 示出了以这种方式构造的 7 阶双调排序器：它可以用作一个具有 3 个延迟单位的（3，4）或（2，5）合并网络，对于 $m = 2$ 和 $n = 5$ 的奇偶合并，比较器少一个，但多一级延迟。

2^k 阶的巴切尔双调排序器是特别有趣的：它由 k 层组成，每层有 2^{k-1} 个比较器。如果

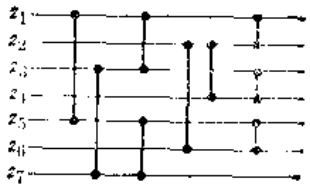


图52　7阶巴切尔双调排序器

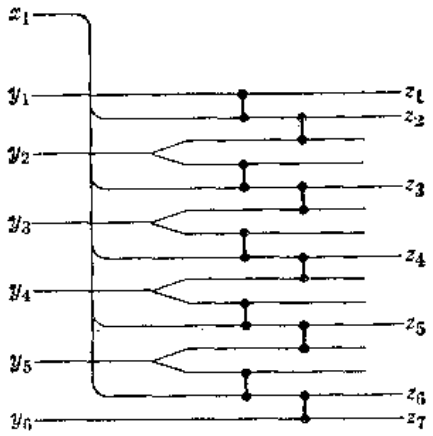


图53　把一个项目同六个其它项目合并，用多重扇形展开，以便达到极小延迟时间

我们对输入线按 z_0，z_1，$\cdots z_{2^k-1}$ 编号，则当且仅当 i 和 j 仅仅在它们的二进位表示的第 l 个最高位上不同时，元素 z_i 才和 l 层上的 z_j 进行比较。这个简单的结构导致了平行排序网络，它象合并交换（算法 5.2.2M）那样快，但是实现要容易得多（见习题 11 和 13）。

当 $m=n$ 时，不难证明，奇偶合并和巴切尔的双调排序器都提供了在任何合并网络中可以达到的绝对极小的延迟时间。一个（n，n）合并网络的第 n 个最小的输出必须依赖于所有的 $2n$ 个输入，而且如果它可以在 l 步中计算，则最多涉及 2^l 个输入；因此 $2^l \geqslant 2n$，$l \geqslant \lceil \log_2(2n) \rceil$。

当 $m<n$ 时，一个（m，n）合并网络的第 n 个输出依赖于 $2m+1$ 个输入（参考习题 29）。所以，在这种情况下同样的论证给出了用于合并的极小延迟时间 $\lceil \log_2(2m+1) \rceil$。巴切尔已经说明〔Report GER-14122（Akron，Ohio：Goodyear Aerospace Corporation，1968）〕，如果在这个网络中允许"多重的扇形展开"，即分开诸直线使得在同一时间里同一个数可以输入到许多模块中，则这个极小延迟时间是可以达到的。例如图 53 示出了他给出的 $n=6$ 时能在仅仅两个延迟级中把一个项目同 n 个其它项目合并的网络。当然，具有多个扇形展开的网络不可能符合我们的约定，很容易看出：没有多重扇形展开的任何（1，n）合并网络必须有 $\log_2(n+1)$ 或更多的延迟时间（见习题 14）。

选择网络　也可以使用网络来解决 5.3.3 节的问题。命 $\hat{U}_t(n)$ 表示在一个网络中所需要的比较器的极小个数，这些比较器把 n 个不同输入中的 t 个最大者送入 t 个指定的输出线；允许它们在这些输出线上以任何次序出现。命 $\hat{V}_t(n)$ 表示为了把 n 个不同输入中的第 t 个最大者送入一个指定的输出直线所需要的比较器极小个数；命 $\hat{W}_t(n)$ 表示把 n 个不同输入中的 t 个最大者以非减次序送到 t 个指定的输出线所需要的比较器极小个数。不难看出（参考习题 17）

$$\hat{U}_t(n) \leqslant \hat{V}_{(t)}(n) \leqslant \hat{W}_{(t)}(n) \tag{16}$$

首先假设有 $2t$ 个元素 $\langle x_1, \cdots, x_{2t} \rangle$，我们希望选择最大的 t 个；В. Э. 阿历克谢耶夫（В. Э. Алекцеев）〔Кибернемика 5，5（1969），99-103〕已经注意到，要做到这一点我们可以首先对 $\langle x_1, \cdots, x_t \rangle$ 和 $\langle x_{t+1} \cdots, x_{2t} \rangle$ 排序，然后比较和交换

$$x_1 : x_{2t}, \quad x_2 : x_{2t-1} \quad \cdots, \quad x_t : x_{t+1} \tag{17}$$

由于这些对中没有一对能包含一个以上的最大的 t 个元素之一，（为什么？）阿历克谢耶夫过程必定能选择最大的 t 个元素。

如果要选择 nt 个元素中的 t 个最大者，则可应用这个过程 $n-1$ 次（每次消去 t 个元素）；因此

$$\hat{U}_t(nt)\leqslant(n-1)(2\hat{S}(t)+t) \tag{18}$$

阿历克谢耶夫也对这个选择问题推导出一个有趣的下界。

定理A $\hat{U}_t(n)\geqslant(n-t)\lceil\log_2(t+1)\rceil$。

证明　考虑选择最小的 t 个元素这样一个等价的问题是最方便的。可以把数（l，u）附加到一个比较器网络的每条直线上，如图 54 中所示，其中 l 和 u 分别表示当输入是{1，2，…，n}的一个排列时在该位置可以出现的极小值和极大值。命 l_i 和 l_j 是进行比较 x_i：x_j 之前在直线 i 和 j 上的下界，并命 l'_i 和 l'_j 是进行比较之后对应的下界。显然，$l'_i=\min(l_i,\ l_j)$，习题 24 证明了这个并非显而易见的关系

$$l'_j\leqslant l_i+l_j \tag{19}$$

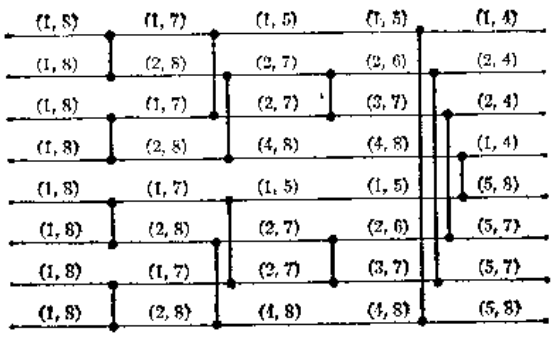


图54　分开最小的四个与最大的四个（在这些直线上的数用于定理A的证明）

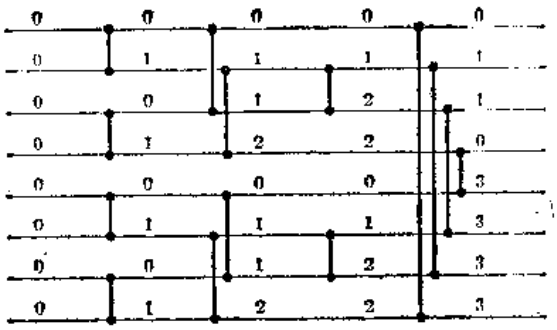


图55　对于图 54 的网络的另一种解释

现在让我们重新以另一种方式解释网络的操作（见图 55）：假定所有的输入线都含有零，而且每个“比较器”把它的输入中的较小者置于上边的直线，并把较大者加 1 置于下边的直线。得到的数〈m_1，m_2，…，m_n〉在整个网络中均有性质

$$2^{m_i}\geqslant l_i \tag{20}$$

因为这在开始时成立，而且由于（19），通过每个比较器后它还继续成立。进而

$$m_1+m_2+\cdots+m_n$$

的最后值是在这个网络中比较器的总数，因为每个比较器都对这个和加 1。

如果这个网络选择最小的 t 个数，则诸 l_i 中有 $n-t$ 个$\geqslant t+1$；因此诸 m_i 中有 $n-t$ 个必然$\geqslant\lceil\log_2(t+1)\rceil$

当 $t=1$ 和 $t=2$ 时定理A的下界证明是精确的（见习题 19）。表 1 给出了对于小的 t 和 n，$\hat{U}_t(n)$、$\hat{V}_t(n)$ 和 $\hat{W}_t(n)$ 的某些值。

表1　在选择网络中所需要的比较（$\hat{U}_t(n)$，$\hat{V}(n)$，$\hat{W}_t(n)$）

	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
$n=1$	(0,0,0)					
$n=2$	(1,1,1)	(0,1,1)				
$n=3$	(2,2,2)	(2,3,3)	(0,2,3)			
$n=4$	(3,3,3)	(4,5,5)	(3,5,5)	(0,3,5)		
$n=5$	(4,4,4)	(6,7,7)	(6,7,8)	(4,7,9)	(0,4,9)	
$n=6$	(5,5,5)	(8,9,9)	(8,10,10)	(8,10,12)	(5,9,12)	(0,5,12)

习题——第一组

下列习题中的若干题深入探讨了排序网络理论，此处引进某些记号是方便的。我们命$[i:j]$代表一个比较/交换模块。具有n个输入和r个比较模块的一个网络写作$[i_1:j_1][i_2:j_2]\cdots[i_r:j_r]$，其中每个$i$和$j$都$\leqslant n$；为了简略起见，我们称它为$n$-网络。一个网络称作标准的，如果对于$1\leqslant q\leqslant r$，$i_q<j_q$。于是例如，图44描绘了一个标准的4-网络，以比较器序列[1:2][3:4][1:3][2:4][2:3]表示之。

正文中对于画网络图形的约定仅仅表示标准的网络；所有比较器$[i:j]$都以从i到j的一条直线表示，其中$i<j$。当须画出非标准网络时，可以使用从i到j的一个箭头，表示较大的数转到箭头的端点。例如，图56示出了对于16个元素的非标准网络，它的比较符是[1:2][4:3][5:6][8:7]等。习题11证明图56是一个排序网络。

如果$x=\langle x_1,\ \cdots,\ x_n\rangle$是一个$n$向量，$\alpha$是一个$n$网络，则由这个网络产生的数向量$\langle(x\alpha_1),\cdots,(x\alpha_n)\rangle$，写作$x\alpha$。为了简洁，令$a\vee b=\max(a,\ b)$，$a\wedge b=\min(a,\ b)$，$\bar{a}=1-a$。于是当$i\neq k\neq j$时，$(x[i:j])_i=x_i\wedge x_j$，$(x[i:j])_j=x_i\vee x_j$，以及$(x[i:j])_k=x_k$。当且仅当对于所有的$x$和对于$1\leqslant i\leqslant n$，$(x\alpha)_n\leqslant(x\alpha)_{i+1}$时，我们说$\alpha$是一个排序网络。

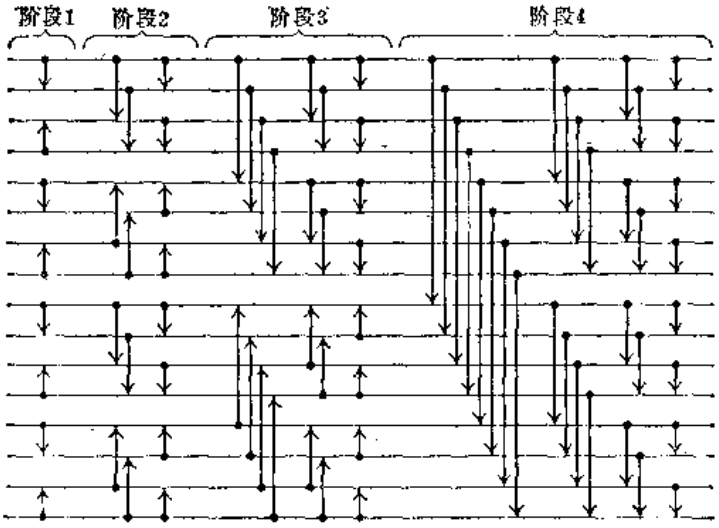


图56　以双调排序为基础的一个非标准排序网络

符号$e^{(i)}$代表在位置i处为1，其它处为0的一个向量；于是$(e^{(i)})_j=\delta_{ij}$。符号D_n代表所有由0和1组成的2^n个n维向量的集合，而P_n代表由$\{1,\ 2,\cdots,\ n\}$的诸排列构成的所有$n!$个向量的集合。向量$\langle x_1\wedge y_1,\ \cdots,\ x_n\wedge y_n\rangle$和$\langle x_1\vee y_1,\ \cdots,\ x_n\vee y_n\rangle$写成$x\wedge y$和$x\vee y$，而且如果对于所有的$i$，$x_i\leqslant y_i$，则写成$x\leqslant y$。于是$x\leqslant y$当且仅当$x\vee y=y$当且仅当$x\wedge y=x$。如果$x$和$y$在$D_n$中，且对某个$i$，$x=y\vee e^{(i)}\neq y$，则说$x$覆盖$y$。最后，对于$D_n$中所有$x$，设$\nu(x)$是$x$中1的个数，$\zeta(x)$是0的个数；于是

$\nu(x)+\xi(x)=n$。

1.〔20〕 当$m=3$和$n=5$时，试画出一个奇偶合并的网络图形。

2.〔22〕 证明V.普拉特的排序算法（5.2.1-30）导致对于n个元素的一个排序网络，它有近似于（$\log_2 n$）（$\log_2 n$）的延迟级。试画出对于$n=12$的对应的网络。

3.〔M20〕（K.E.巴切尔）试求$C(m, m-1)$和$C(m, m)$之间的一个简单关系。

▶4.〔M23〕 证明$\hat{T}(6)=5$。

5.〔M21〕 证明（13）是（10）中概述的排序网络所需的延迟时间。

6.〔28〕 命$T(n)$是通过进行同时的不相交的比较（没有网络限制）排序所需要的极小阶段数。每组这样的比较可表示为包含对偶集合$i_1:j_1$，$i_2:j_2$，…，$i_r:j_r$的一个节点，其中i_1，j_1，i_2，j_2，…，i_r，j_r是不同的，在这个节点之下有2^r个分支，它们分别表示下列情况

$$\langle K_{i_1}<K_{j_1},\ K_{i_2}<K_{j_2},\ \cdots,\ K_{i_r}<K_{j_r}\rangle$$

等等

$$\langle K_{i_1}>K_{j_1},\ K_{i_2}<K_{j_2},\ \cdots,\ K_{i_r}<K_{j_r}\rangle$$

试证明$T(5)=T(6)=5$。

7.〔25〕 证明如果图49 $n=10$的网络中最后的比较器恰被移到倒数第二个和倒数第三个比较器的前头，则这网络仍能完成排序。

8.〔M20〕 证明对于m_1，m_2，$n_1, n_2\geqslant 0$，$\hat{M}(m_1+m_2, n_1+n_2)\geqslant\hat{M}(m_1, n_1)+\hat{M}(m_2, n_2)+\min(m_1, n_2)$。

9.〔M25〕（R.W.弗洛伊德）证明$\hat{M}(3,3)=6$，$\hat{M}(4,4)=9$，$\hat{M}(5,5)=13$。

10.〔M22〕 证明巴切尔的双调排序器，如同在（15）之前的注释中所定义的那样，是有效的。〔提示：只须证明，所有的由k个1，后边接以l个0，后边再接以$n-k-l$个1组成的序列都能被排序。〕

11.〔M23〕 证明阶为2^p的巴切尔双调排序器将不仅对满足$z_0\geqslant\cdots\geqslant z_k\leqslant\cdots z_{2^p-1}$的序列（$z_0$，$z_1$，…，$z_{2^p-1}$）排序，也将对满足$z_0\leqslant\cdots\leqslant z_k\geqslant\cdots\geqslant z_{2^p-1}$的任何序列排序。〔推论，图56中的网络将对16个元素排序，这是因为虚线分开的每个阶段由双调排序器或逆序的双调排序器组成，后者应用于已经以相反的方向排好序的序列。〕

12.〔M20〕 证明或否定：如果x和y是同样长度的双调序列，则$x\vee y$和$x\wedge y$也是。

▶13.〔24〕〔H.S.斯通（H.S.Stone）〕 证明对于2^t个元素的一个排序网络，可以仿照图57中对于$t=4$所示的型式加以构造。在这个方案中t^2个步骤的每一步都由头2^{t-1}个元素和最后2^{t-1}个元素的一个“完全的洗牌”组成，接着是对2^{t-1}对相邻元素执行的同时的操作。这些操作或是“0”（无操作），或“+”（一个标准比较模块）或“−”（反序的比较模块）。这个排序分t个阶段进行，每阶段又分t步；在最后阶段，所有的操作都是“+”。在阶段s（$s<t$），我们做其中所有操作都是“0”的$t-s$步，后边接之以s步，其中在第q步内的操作交替地由$2q$个“+”接之以$2q$个“−”组成，$q=1, 2, \cdots, s$。

〔注意，这个排序方案可以通过相当简单的设备予以实施。这个设备的线路执行一个

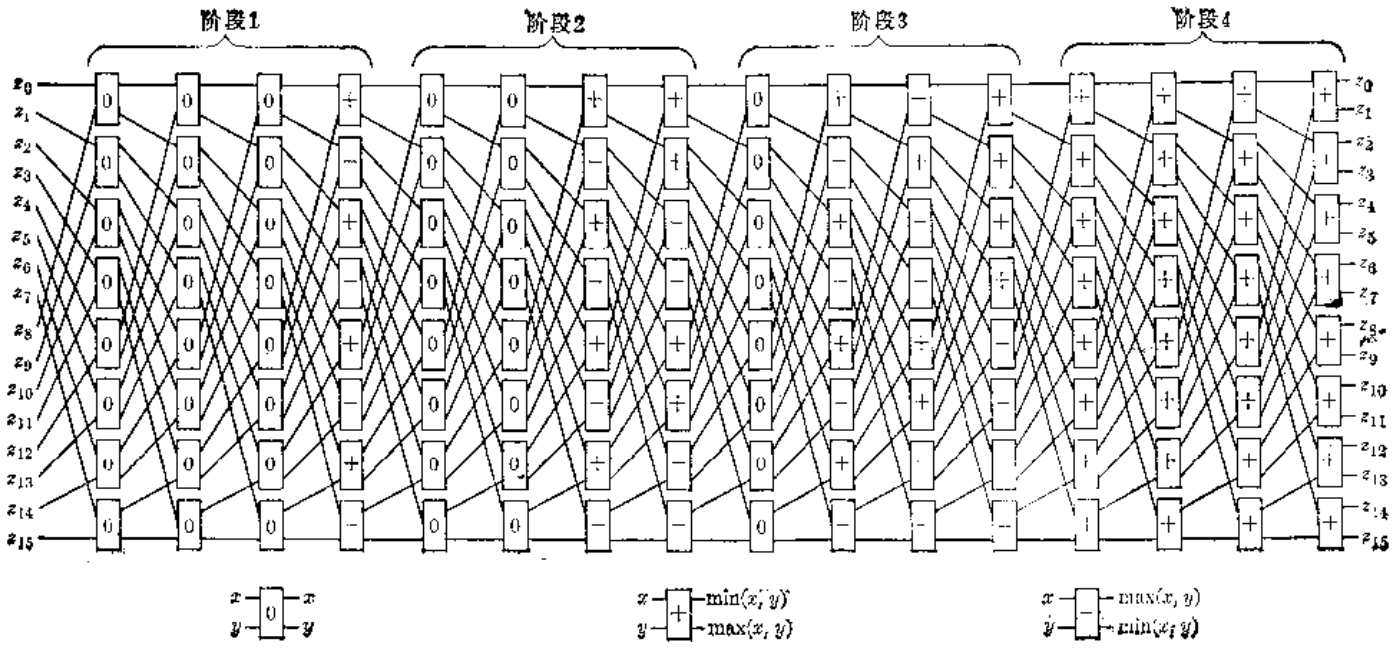


图57　用"完全的洗牌"对16个元素排序

"洗牌和操作"步骤，以及把输出线反馈到输入。图 57 中的头三步当然可以消去；保留它们仅仅是为了使这个型式更清楚。斯通指出，在若干其它算法中也出现有相同型式的"洗牌操作"，诸如在快速傅里叶变换中（参考 4.6.4 节）。〕

14.〔$M20$〕 证明任何没有多重扇形展开的（1，n）合并网络都至少必须有$\lceil\log_2(n+1)\rceil$个延迟级。

15.〔20〕 试求四个元素的一个非标准排序网络，它只有五个比较模块。

16.〔$M22$〕 证明：下列算法把任何一个排序网络$[i_1:j_1]$，…，$[i_r:j_r]$都变换成一个标准的排序网络。

T1. 命 q 是使得 $i_q>j_q$ 的最小下标，如果这样的下标不存在，就停止。

T2. 对于 $q\leqslant s\leqslant r$，在所有比较器$[i_s:j_s]$中把 i_q 的所有出现改为 j_q，并把 j_q 的所有出现改为 i_q。返回 T1。

例如，网络[4:1][3:2][1:3][2:4][1:2][3:4]首先变换成[1:4][3:2][4:3][2:1][4:2][3:1]，然后[1:4][2:3][4:2][3:1][4:3][2:1]，然后[1:4][2:3][2:4][3:1][2:3][4:1]等等。直到得到标准网络[1:4][2:3][2:4][1:3][1:2][3:4]为止。

17.〔$M25$〕 命 D_{tn} 是恰有 t 个 1 的所有$\binom{n}{t}$个 0 和 1 的序列$\langle x_1,\ \cdots,\ x_n\rangle$的集合。证明：$\hat{U}_t(n)$是在对 D_{tn} 的所有元素排序的一个网络中所需要的比较器的极小个数；$\hat{V}_t(n)$ 是为排序 $D_{tn}\cup D_{(t-1)n}$ 所需要的比较器的极小个数；而 $\hat{W}_t(n)$ 是为对$\bigcup_{0\leqslant k\leqslant t}D_{kn}$排序所需要的比较器的极小个数。

▶18.〔$M20$〕 证明求 $2t-1$ 个元素的中值的一个网络至少需要 $(t-1)\lceil\log_2(t+1)\rceil+\lceil\log_2 t\rceil$ 个比较模块。〔提示：见定理 A 的证明。〕

19.〔$M22$〕 证明：对所有 $n\geqslant 2$，$\hat{U}_2(n)=2n-4$ 和 $\hat{V}_2(n)=2n-3$。

20.〔24〕 证明 $\hat{V}_3(5)=7$。

21.〔$M15$〕 设 α 是任何 n-网络，并设 x 和 y 是 n-向量。证明 $x\leqslant y$ 蕴涵 $x\alpha\leqslant y\alpha$。

22.〔$M15$〕 证明：如果 x 和 y 是实数的 n-向量，则 $x\cdot y\leqslant(x\alpha)\cdot(y\alpha)$（这里 $x\cdot y$ 是内积 $x_1y_1+\cdots+x_ny_n$）。

23.〔$M1$〕 设 α 是一个 n-网络。证明存在一个排列 $p\in P_n$，使得 $(p\alpha)_i=j$ 当且仅当在 D_n 中存在向量 x，y 使得 x 覆盖 y，$(x\alpha)_i=1$，$(y\alpha)_i=0$ 和 $\zeta(y)=j$。

▶24.〔$M21$〕（V. E. 阿历克谢耶夫）设 α 是一个 n-网络，而且对于 $1\leqslant k\leqslant n$，命 $l_k=\min\{(p\alpha)_k|\ p$ 在 P_n 中$\}$，$U_k=\max\{(p\alpha)_k|\ p$ 在 P_n 中$\}$，表示可出现于输出线 k 中值的下界和上界，对于网络 $\alpha'=\alpha[i:j]$类似地定义 l'_k 和 u'_k。证明 $l'_i=l_i\wedge l_j$，$l'_j\leqslant l_i+l_j$，$u'_i\geqslant u_i+u_j-(n+1)$，$u'_i=u_i\vee u_j$。〔提示：给定 D_m 中的向量 x 和 y，有 $(x\alpha)_i=(y\alpha)_j=0$，$\zeta(x)=l_i$，$\zeta(y)=l_j$，求出 D_m 中的一个向量 z，使 $(z\alpha')_i=0$，$\zeta(z)\leqslant l_i+l_j$。〕

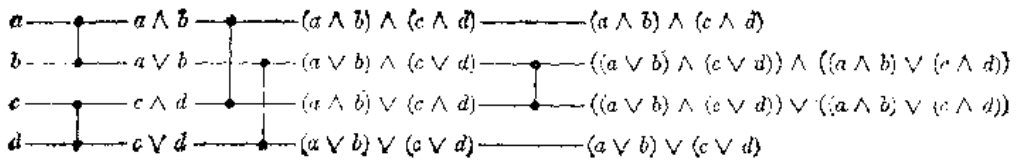
25.〔$M30$〕 设 l_k 和 u_k 如习题 24 所定义的那样。证明集合$\{(p\alpha)_k|\ p$ 在 P_n 中$\}$包括 l_k 和 u_k 之间以及 l_k 和 u_k 本身的所有整数。

26.〔$M24$〕（R. W. 弗洛伊德）设 α 是一个 n-网络。证明集合 $D_n\alpha=\{x\alpha|x$ 在 D_m 中$\}$可以由集合 $P_n\alpha=\{p\alpha|\ p$ 在 P_n 中$\}$确定；反之，$P_n\alpha$ 可由 $D_n\alpha$ 确定。

▶27.〔$M20$〕 设 x 和 y 是向量，并设 $x\alpha$ 和 $y\alpha$ 被排序，证明 $(x\alpha)_i\leqslant(y\alpha)_j$，当且

仅当每从 y 中任取 j 个元素，我们都可以从 x 中选择 i 个元素，使得每个选定的 x 元素$\leqslant$某个选定的 y 元素。利用这一原理证明，如果对任何矩阵先按行，后按列排序，则诸行仍保持有序。

▶28.〔$M20$〕 下列图形说明，有可能利用输入元素写出在一个排序网络中所有直线上的内容的公式：



利用交换律 $x\wedge y=y\wedge x$，$x\vee y=y\vee x$，结合律 $x\wedge(y\wedge x)=(x\wedge y)\wedge z$，$x\vee(y\vee z)=(x\vee y)\vee z$，分配律 $x\wedge(y\vee z)=(x\wedge y)\vee(x\wedge z)$，$x\vee(y\wedge z)=(x\vee y)\wedge(x\vee z)$ 吸收律 $x\wedge(x\vee y)=x\vee(x\wedge y)=x$ 以及等幂律 $x\wedge x=x\vee x=x$，可以把这个网络右端的公式分别地简化成为 $(a\wedge b\wedge c\wedge d)$，$(a\wedge b\wedge c)\vee(a\wedge b\wedge d)\vee(a\wedge c\wedge d)\vee(b\wedge c\wedge d)$，$(a\wedge b)\vee(a\wedge c)\vee(a\wedge d)\vee(b\wedge c)\vee(b\wedge d)\vee(c\wedge d)$，$a\vee b\vee c\vee d$。

证明，一般地，$\{x_1, \cdots, x_n\}$ 的第 t 个最大元素由“初等对称函数”

$\sigma_t(x_1,\cdots,x_n)=\bigvee\{x_{i_1}\wedge x_{i_2}\wedge\cdots\wedge x_{i_t}\mid 1\leqslant i_1<i_2<\cdots<i_t\leqslant n\}$ 给出。它有 $\binom{n}{t}$ 项被$\vee$在一起。于是求极小花费的排序网络的问题，等价于用极小数量的“与/或”线路计算初等对称函数的问题，其中在每级上我们用 $\phi\wedge\psi$ 和 $\phi\vee\psi$ 代替两个量 φ 和 ψ。

29.〔$M20$〕 设 $x_1\leqslant x_2\leqslant x_3$ 和 $y_1\leqslant y_2\leqslant y_3\leqslant y_4\leqslant y_5$，又设 $z_1\leqslant z_2\leqslant\cdots\leqslant z_8$ 是把诸 x 和诸 y 合并的结果，应用运算符$\wedge$和$\vee$，求出以 x 和 y 表达的每一个 z 的公式。

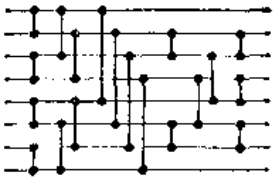
30.〔$HM24$〕 证明：使用习题 28 中的恒等式，可把任何包含$\wedge$和$\vee$和独立变量 $\{x_1, \cdots, x_n\}$ 的公式，归结为一个“正则”的形式 $\tau_1\vee\tau_2\cdots\vee\tau_k$，其中 $k\geqslant 1$，每个 τ_i 形为 $\wedge\{x_j \mid j$ 在 S_i 中$\}$，其中 S_i 是 $\{1, 2, \cdots, n\}$ 的一个子集，而且当 $i\neq j$ 时，集合 S_i 不包括在 S_j 中。证明两个这样的正则形式对于所有的 $x_1, \cdots, x_n$ 都相等的充要条件是它们恒等（准确到次序）。

31.〔$M24$〕 （R. 狄德金 1897）设 δ_n 是在习题 30 的意义下关于 $x_1, \cdots, x_n$ 的不同正则形式的个数。于是 $\delta_1=2$，$\delta_2=4$ 和 $\delta_3=18$，问 δ_4 是多少？

32.〔$M28$〕 （M. W. 格林）设 $G_1=\{00, 01, 11\}$，G_{n+1} 是所有串 $\theta\phi\psi\omega$ 的集合，其中 θ，φ，ψ，ω 的长度为 2^{n-1} 且 $\theta\varphi$，$\psi\omega$，$\theta\psi$ 及 $\varphi\omega$ 在 G_n 中。设 α 是由图 49 中所示 16 排序器的头四级组成的网络。证明 $D_{16\alpha}=G_4$，并且证明，它恰有 δ_4+2 个元素（参考习题 31）。

▶33.〔$M22$〕 并非习题 31 中的 $(x_1, \cdots, x_n)$ 的函数的所有 δ_n 都可以出现于比较器网络中。事实上，证明函数 $(x_1\wedge x_2)\vee(x_2\wedge x_3)\vee(x_3\wedge x_4)$ 不可能成为关于 $\langle x_1, \cdots x_n\rangle$ 的任何比较器网络的输出。

34.〔23〕 下图是否是一个排序网络？



35.〔20〕 证明对于 $1 \leqslant i < n$，任何标准排序网络都必须至少包含每个“相邻的”比较器〔$i:i+1$〕一次。

36.〔22〕 图 47 的网络仅包含相邻的比较〔$i:i+1$〕；我们称这个网络为本原的。(a)证明 n 个元素的一个本原排序网络至少有$\binom{n}{2}$个比较器。〔提示：考虑一个排列的逆。〕(b)（R. W. 弗洛伊德，1964）设 α 是 n 个元素的一个本原网络，又设 x 是对于某个 $i<j$ 使得 $(x\alpha)_i>(x\alpha)_j$ 的一个向量。证明 $(y\alpha)_i>(y\alpha)_j$，其中 y 是向量 $\langle n_1, n-1, \cdots, 1\rangle$。(c) 作为 (b) 的推论，一个本原网络是一个排序网络的充要条件是，它把单个向量 $\langle n, n-1, \cdots, 1\rangle$ 排序！

37.〔M22〕 对于 $n \geqslant 3$ 的 n 个数的奇偶转置排序，是如图 58 所示的一个排成类似砖块式的，具有 $\frac{1}{2}n(n-1)$ 个比较器的，深度为 n 层的网络（当 n 是偶数时有两种可能性）。这种排序在硬件中是特别容易实现的，因为仅仅交替地执行两类操作。证明，这样一个网络是一个有效的排序网络〔提示：见习题 36〕。

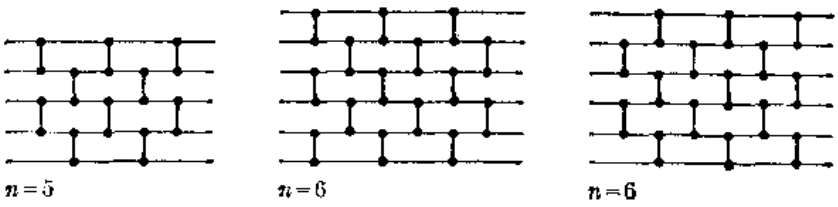


图58 奇偶转置排序

38.〔29〕 我们可以用另一种方式来解释排序网络，让每条线载有 m 个数的一个多重集合，而不是单个数；在这种解释之下，操作〔$i:j$〕分别地以 $x_i \curlywedge x_j$ 和 $x_i \curlyvee x_j$ 代替 x_i 和 x_j，即 $2m$ 个数 $x_i \uplus x_j$ 中最小的 m 个和最大的 m 个（当 $m=2$ 时，对于这种解释的图示，见图 59）。如果 a 和 b 是各含 m 个数的多重集合，则当且仅当 $a \curlywedge b = a$ 时我们说 $a \ll b$（等价地，$a \curlyvee b = b$；a 的最大元素小于或等于 b 的最小者）。于是 $a \curlywedge b \ll a \curlyvee b$。

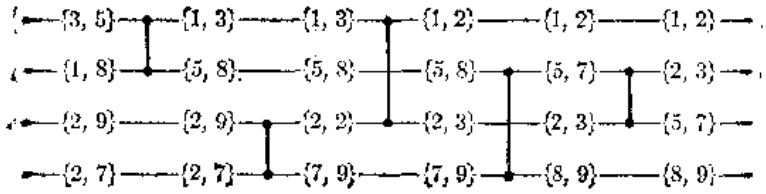


图59 图44中的排序网络的另一种解释：每个比较模块执行一个合并操作

设 α 是一个 n-网络，又设 $x=\langle x_1, \cdots, x_n\rangle$ 是一个向量，其中每个 x_i 是 m 个元素的一个多重集合。证明，在上述解释中，如果 $(x\alpha)_i$ 不是 $\ll(x\alpha)_j$，则在 D_n 中有一个向量 y，使得 $(y\alpha)_i=1$ 和 $(y\alpha)_j=0$。〔因此，如果我们以 m 路合并来代替比较，则 n 个元素的一个排序网络变成 mn 个元素的一个排序网络，图 60 示出了使用这种办法由四元排序器构造的八元排序器。〕

▶39.〔$M23$〕 在习题 38 的记号下，证明 $(x\curlywedge y)\curlywedge z=x\curlywedge(y\curlywedge z)$ 和 $(x\curlyvee y)\curlyvee z=x\curlyvee(y\curlyvee z)$；然而 $(x\curlyvee y)\curlywedge z$ 不总等于 $(x\curlywedge z)\curlyvee(y\curlywedge z)$，而且 $(x\curlywedge y)\curlyvee(x\curlywedge z)\curlyvee(y\curlywedge z)$ 不总等于 $x\uplus y\uplus z$ 中间的 m 个元素。对于这些中间元素，试借助于 x，y，z 和 $\curlywedge$ 及 $\curlyvee$ 运算，求出一个正确的公式。

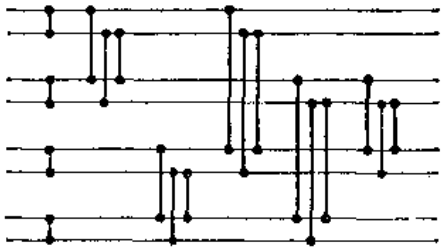


图60 通过使用合并解释，从一个 4-排序器构造一个 8-排序器

▶40〔$M25$〕 （R·L·格雷厄姆）我们说比较器 $[i:j]$ 在网络 $\alpha_1[i:j]\alpha_2$ 中是多余的，如果对于所有向量 x，$(x\alpha_1)_i\leqslant(x\alpha_1)_j$，或对所有向量 x，$(x\alpha_1)_i\geqslant(x\alpha_1)_j$。证明，如果 α 是具有 r 个非多余比较器的网络，则至少有不同下标的 r 个不同的有序对 (i, j)，使得对于所有向量 x，$(x\alpha)_i\leqslant(x\alpha)_j$。（因此没有多余的比较器的一个网络至多包含 $\binom{n}{2}$ 个模块。）

▶41.〔$M27$〕 （V.E.阿历克谢耶夫）设 $\alpha=[i_1:j_1]\cdots[i_r:j_r]$ 是一个 n-网络；对于 $1\leqslant s\leqslant r$，我们定义 $\alpha^s=[i'_1:j'_1]\cdots[i'_{s-1}:j'_{s-1}][i_s:j_s]\cdots[i_r:j_r]$，其中 i'_k 和 j'_k 是从 i_k 和 j_k 通过把 i_s 改换为 j_s，以及把 j_s 改换为 i_s（对于它们的所有出现）而得到的。例如，如果 $\alpha=[1:2][3:4][1:3][2:4][2:3]$，则 $\alpha^4=[1:4][3:2][1:3][2:4][2:3]$。

a）证明 $D_n\alpha=D_n(\alpha^s)$。

b）证明 $(\alpha^s)^t=(\alpha^t)^s$。

c）α 的共轭是任一形如 $(\cdots((\alpha^{s_1})^{s_2})\cdots)^{s_k}$ 的网络。证明 α 至少有 2^{r-1} 个共轭。

d）设当 $x\in D_n\alpha$ 时 $g_\alpha(x)=1$，否则 $g_\alpha(x)=0$；又设

$$f_\alpha(x)=(\bar{x}_{i_1}\vee x_{j_1})\wedge\cdots\wedge(\bar{x}_{i_r}\vee x_{j_r})$$

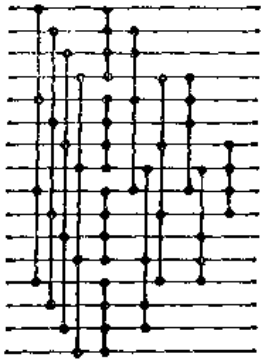
证明 $g_\alpha(x)=\vee\{f_{\alpha'}(x)\mid\alpha'$ 是 α 的一个共轭$\}$。

e）设 G_α 是具有顶点 $\{1, \cdots, n\}$ 以及弧 $i_s\to j_s$，$1\leqslant s\leqslant r$ 的有向图。证明：α 是一个排序网络的充要条件是，对于 $1\leqslant i<n$ 和对于共轭于 α 的所有 α'，$G_{\alpha'}$ 有从 i 到 $i+1$ 的一条有向路径。〔这个条件是颇值得注意的，因为 G_α 不依赖于 α 中的比较器的次序。〕

42.〔25〕 （D.范·沃勒斯）证明：$\hat{S}(n)\geqslant\hat{S}(n-1)+\lceil\log_2 n\rceil$。

43.〔23〕 一个排列网络是一个模块序列 $[i_1:j_1]\cdots[i_r:j_r]$，其中每个模块 $[i:j]$ 都可以通过外部控制来设置，使其输入或不经改变地通过或交换 x_i 和 x_j（不论 x_i 和 x_j 的值如何），适当设置模块，即可在输出线上得到输入量的任意排列。每个排序网络显然是一个排列网络，但反之不真。求仅有八个模块的五元素排列网络。

44.〔46〕 研究这样的排序网络的性质，它是由 m 排序器模块而不是由 2 排序器组成的。（例如，G·夏皮罗已经构造了下列网络，它使用 14 个 4 排序器对 16 个元素排序。这是最好的可能吗？有没有有效的方法对所有的 m，用 m 排序器模块对 m^2 个元素进行排序？）



45.〔48〕 求出具有少于$C(m, n)$个比较器的一个(m, n)合并网络，或证明它不存在。

46.〔48〕 试求具有少于$\lceil \log(m+n) \rceil$个延迟级的一个(m, n)合并网络，或证明它不存在。

47.〔48〕 研究一类排序方案，这类方案可以如同图 57 中那样，通过完全的洗牌网络实现，但利用“0”、“+”和“−”运算的其它型式。

48.〔HM49〕 剖析习题 38 中定义的$\wedge$和$\vee$运算的性质。能否以某种精细的方式来表征这个代数中的所有恒等式，或者以有限的一组恒等式来推导它们？在这方面，像$x \wedge x \wedge x = x \wedge x$或$x \wedge (x \vee (x \wedge (x \vee y))) = x \wedge (x \vee y)$这样的恒等式，它们仅对$m \leqslant 2$成立，是意思不大的。请仅仅考虑对所有$m$为真的恒等式。

49.〔M49〕 习题 6 中定义的函数$T(n)$具有何种渐近行为？是否存在某个n，使$T(n) < \hat{T}(n)$？

50.〔50〕 试求$\hat{S}(n)$对于某个$n > 8$的准确值。

51.〔M50〕 证明$\hat{S}(n)$的渐近值不是$O(n \log_2 n)$。

习题——第二组

下列习题涉及了同排序有关的若干不同类型的最优性问题。前面的少数问题，是以P.N.阿姆斯特朗和R.J.纳尔逊早在 1954 年研究过的气泡排序有趣的“多头”推广为基础的〔见U.S.Patents 3029413, 3034102〕。设$1 = h_1 < h_2 < \cdots < h_m = n$是一个递增的整数序列；我们称它做长度为$m$和间距为$n$的一个“头序列”，并用它来定义一种特殊类型的排序方法。记录$R_1, \cdots, R_N$的排序分若干次扫描进行，每次扫描由$N+n-1$个步骤组成。在第j步上$(j = 1-n, 2-n, \cdots N-1)$，应考察并在必要时重新排列记录$R_{j+h[1]}, R_{j+h[2]}, \cdots, R_{j+h[m]}$，使得它们的键成为有序的。(这时我们说$R_{j+h[1]}, \cdots, R_{j+h[m]}$是“在读写头之下”。当$j+h[k] < 1$或$> N$时，不考虑记录$R_{j+h[k]}$；从效果上看键$K_0$，$K_{-1}$，$K_{-2}$，…被处理做$-\infty$，而$K_{N+1}$，$K_{N+2}$，…被处理做$+\infty$。因此当$j \leqslant -h[m-1]$或$j > N-h[2]$时，步骤$j$实际上是显然的)。

例如，下表示出了当$m=3$，$N=9$以及$h_1=1$，$h_2=2$，$h_3=4$时一个排序的一次扫描。

$K_{-1}K_{-2}K_0K_1K_2K_3K_4K_5K_6K_7K_8K_9K_{10}K_{11}K_{12}$

$j=-3$ ___ 3 1 4 5 9 2 6 8 7

$j=-2$ ___ 3 1 4 5 9 2 6 8 7

$j=-1$	$_\ \underline{3}\ 1\ \underline{4}\ 5\ 9\ 2\ 6\ 8\ 7$
$j=0$	$\underline{1}\ 3\ 4\ \underline{5}\ 9\ 2\ 6\ 8\ 7$
$j=1$	$1\ \underline{3}\ \underline{4}\ 5\ \underline{9}\ 2\ 6\ 8\ 7$
$j=2$	$1\ 3\ \underline{2}\ \underline{4}\ 9\ \underline{5}\ 6\ 8\ 7$
$j=3$	$1\ 3\ 2\ \underline{4}\ \underline{6}\ 5\ \underline{9}\ 8\ 7$
$j=4$	$1\ 3\ 2\ 4\ \underline{5}\ \underline{6}\ 9\ \underline{8}\ 7$
$j=5$	$1\ 3\ 2\ 4\ 5\ 6\ \underline{7}\ 8\ \underline{9}$
$j=6$	$1\ 3\ 2\ 4\ 5\ 6\ \underline{7}\ 8\ 9\ _$
$j=7$	$1\ 3\ 2\ 4\ 5\ 6\ 7\ \underline{8}\ \underline{9}\ _$
$j=8$	$1\ 3\ 2\ 4\ 5\ 6\ 7\ 8\ \underline{9}\ _\ _$

注意，当$m=2$，$h_1=1$及$h_2=2$时，这个多头的方法简化成为气泡排序(算法5.2.2B)。

52.〔21〕〔詹姆斯·达冈迪伊（James Dugundji）〕 证明如果对于某个k，$1\leqslant k\leqslant m$，有$h(k+1)=h[k]+1$，则上边定义的多头排序器将在有限次扫描中最终完成对任何输入文件的排序。但是如果对于$1\leqslant k<m$，$h[k+1]\geqslant h[k]+2$，则这个输入可能永远排不了序。

▶53.〔30〕（阿姆斯特朗和纳尔逊） 设对$1\leqslant k<m$，$h[k+1]\leqslant h[k]+k$，又设$N\geqslant n-1$，证明经头一次扫描后最大的$n-1$个元素总是被送到它们最后目的地。〔提示：使用0—1原理；如果对一些0和1进行排序，且1的个数少于n，证明不可能所有的头都读出1，除非所有的0都在这些头的左边。〕

证明当这些头满足给定条件时，排序将在至多$\lceil(N-1)/(n-1)\rceil$次扫描中完成。是否有一个输入文件，它需要这么多遍扫描？

54.〔26〕 如果$n=N$，证明：当且仅当对于$1\leqslant k<m$，$h[k+1]\leqslant 2h[k]$时，头一次扫描可以保证把最小的键放置到位置R_1。

55.〔34〕（J.霍普克洛夫特） N个元素的一个“完全排序器”，是$N=n$的一个总在一次扫描中完成的多头排序器。习题53证明，序列$\langle h_1, h_2, h_3, h_4, \cdots, h_m\rangle=\langle 1, 2, 4, 7, \cdots, +\binom{m}{2}\rangle$给出对于$N=\binom{m}{2}+1$个元素的使用$m=(\sqrt{8N-7}+1)/2$个头的完全的排序器。例如，头序列$\langle 1, 2, 4, 7, 11, 16, 22\rangle$是对于22个元素的完全排序器。

证明，事实上头序列$\langle 1, 2, 4, 7, 11, 16, 23\rangle$是对于23个元素的一个完全排序器。

56.〔49〕 给定m，研究使m头完全排序器存在的最大N。$N=O(m^2)$吗？

57.〔23〕（V.普拉特） 若对$1\leqslant k\leqslant m$，每个头h_k都在位置2^{k-1}上，现要对0和1的序列$z_1z_2\cdots z_{2^{m-1}}$进行排序，其中当且仅当j是2的一个乘方时$z_j=0$，问需要多少次扫描？

58.〔24〕（一致排序） 5.3.1节中图34的树在第一级的两个分支中进行了2:3的比较，而在第二级的每个分支中比较1:3(除非该比较是多余的)。一般地说，我们可以考虑在这种意义下一致的所有排序算法类：假定$M=\binom{N}{2}$个对偶$\{(a, b)\mid 1\leqslant a<b\leqslant N\}$已经安排成一个序列

$$(a_1, b_1)(a_2, b_2), \cdots, (a_m, b_m)$$

我们可以逐个地对未知结果的对进行比较；$K_{a_1}:K_{b_1}$，$K_{a_2}:K_{b_2}$，…，诸对偶(a, b)共

有$M!$个排列，其中每个都定义了一个一致的算法。一致排序的概念是H. L. 比尤斯（H. L. Beus）给出的[JACM 17（1970），482-495]，他的工作已经提示了下列的几个习题。

借助于图论来形式地定义一致排序是方便的。设G是顶点$\{1, 2 \cdots, N\}$上的有向图，没有弧，对于$i=1, 2, \cdots, M$，我们对G加上弧如下：

情况1　G包含从a_i到b_i的一条路径。把弧$a_i \to b_i$加到G中。

情况2　G包含从b_i到a_i的一条路径。把弧$b_i \to a_i$加到G中。

情况3　G不包含从a_i到b_i或b_i到a_i的路径。比较$K_{a_i}=K_{b_i}$，如果$K_{a_i} \leqslant K_{b_i}$，则把弧$a_i \to b_i$加到$G$中；如果$K_{a_i}>K_{b_i}$，则把弧$b_i \to a_i$加到$G$中。

我们主要关心由一个一致排序算法所作的键比较次数，而不是关心用什么方法避免多余的比较；图G不必明显地构造，这里使用它仅仅是因为它有助于定义一致排序的概念。

我们也将考虑有限制的一致排序，其中在上述的情况1、2和3中，仅仅计算长度为2的路径。（一个有限制的一致排序算法可能会进行某些多余的比较，但习题59说明在有限制的情况下分析要更简单些。）

证明当对偶序列按字典编辑次序排列

$$(1, 2)(1, 3)(1, 4)\cdots(1, N)(2, 3)(2, 4)\cdots(2, N)\cdots(N-1, N)$$

时，有限制的一致算法和一致算法是一样的。证明，事实上，当诸键不同且快速排序的多余比较如习题5.2.2-24中那样被消除时（忽略在快速排序中实在地进行的比较的次序，仅仅考虑哪些关键字对被比较过），这两个算法都等价于“快速排序”（算法5.2.2Q）。

59. [M38]　如同在习题58中那样，给定一个对偶序列$(a_1b_1)\cdots(a_mb_m)$，设c_i是使得$j<k<i$且使(a_ib_i)，(a_jb_j)，(a_kb_k)形成一个三角形的对偶(j, k)的数目。(a)证明通过有限制的一致排序算法所做的平均比较次数是$\sum_{1\leqslant i\leqslant m}2/(c_i+2)$。(b)使用(a)和习题58的结果来确定快速排序执行的非多余比较的平均次数。(c)合并排序引起了(但不等价于)下列的对偶序列：

$$(1,2)(3,4)(5,6)\cdots(1,3)(1,4)(2,3)(2,4)(5,7)\cdots(1,5)(1,6)(1,7)(1,8)\cdots(2,5)\cdots$$

试问以这个序列为基础的一致方法，平均说来，比快速排序执行更多还是更少的比较？

60. [M29]　在最坏的情况下，快速排序进行$\binom{N}{2}$次比较。是否所有有限制的一致排序算法（在习题57的意义下）在其最坏的情况下，都执行$\binom{N}{2}$次比较？

61. [M48]（H. L. 比尤斯）就所有（有限制的）一致排序算法而言，快速排序是否有极小的平均比较次数？

62. [25]　霍华德·B. 德穆思的博士论文“Electronic Data Sorting”（Stanford University：October，1956）也许是详细讨论计算复杂性问题的头一篇论文。德穆思考虑了排序设备的若干抽象模型，并建立了每个模型可达到的平均和极大执行时间的下界和上界。他的最简单的模型“循环无逆存储”（图61），是本习题的主题。

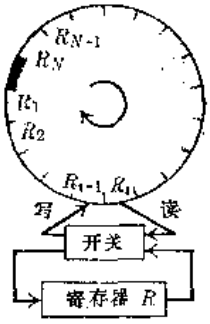


图61　一种设备，对它来说气泡排序是最优的

考虑一台机器，它在若干次扫描中对R_1，R_2，…

R_N 排序，其中每次扫描都包含下列$N+1$步：

步骤1　置$R \leftarrow R_1$（R是机器内部的一个寄存器）。

步骤i　对于$1 < i \leqslant N$

　　或者是（a）置$R_{i-1} \leftarrow R$，$R \leftarrow R_i$；

　　或者是（b）置$R_{i-1} \leftarrow R_i$，R保持不变。

步骤$N+1$　置$R_N \leftarrow R$。

问题是找出一种方式，每次在方案（a）和（b）中进行选择，以便把进行排序所需要的扫描次数极小化。

证明对这个模型来说，"气泡排序"技术是最优的。换句话说，证明每当$R \leqslant R_i$时选择方案（a），以及每当$R > R_i$时选择方案（b）的这个策略，将达到极小的扫描次数。

5.4　外部排序

现在该是我们来研究当有待排序的记录数大于计算机的高速内部存储器所能容纳的数量时，出现的一些有趣问题的时候了。外部排序与内部排序十分不同，因为对外部文件进行存取的有效技术受到相当严格的限制，尽管在两种情况下排序问题都是把一个给定文件排成非减的次序。对外部排序来说，必须把数据结构安排成使得相当慢的外部存储设备（带、盘、鼓、等等），能快速地满足排序算法的要求。因而，我们已经研究过的大多数内部排序技术（插入、合并、选择），实际上对于外部排序是无用的，从而有必要重新考虑全部的问题。

例如，假设要对一个有5000个记录R_1，$R_2 \cdots R_{5000}$的文件排序，而且每个记录R_i是20个字长的（尽管键K_i不必这么长）。如果这些记录中一次只能有1000个装入一台具体计算机的内部存储器中，则应怎么做呢？

一个相当明显的解决方法是，先独立地对五个子文件$R_1 \cdots R_{1000}$，$R_{1001} \cdots R_{2000}$，…，$R_{4001} \cdots R_{5000}$排序，然后把得到的子文件合并在一起。幸而，这个合并过程仅仅使用非常简单的数据结构，即线性表，它们以和栈或队相同的顺序方式被访问；因此合并可以毫无困难地在最少费用的外部存储设备上完成。

刚才描述的过程——先内部排序，再"外部合并"——使用是非常普遍的，我们将把对于外部排序的研究主要集中在这个主题的各种变形上。

通过初始的内部排序阶段产生的记录递增序列，在关于排序的公开著作中通常称作串；这个术语是相当广泛的，可惜，它同在计算机科学的其它分支中另一种更广泛的用法相抵触，在这种更广泛的用法下，串是符号的任意序列。关于排列的研究，已经给一个文件的排好序的段落取了一个美好的名称，即习惯上它被称作递增的路段或简单地说是路段，因此，我们将一致地使用"路段"这个词来描述一个文件的已排好序的部分。用这种方法，就可能把"路段的串"和"串的路段"两者区别开来而不致引起混淆。（当然，"一个程序的运行●"更是指另一件事，我们不可能面面具到。）

● 英语中路段和运行都是run。——译注

现在首先考虑使用磁带作为辅助存储时的外部排序过程。遵循在算法5.2.4N、S和L中使用的中心思想，也许，用带进行合并的最简单和最具吸引力的方法是平衡的2路合并。在这个过程中，我们使用四条“工作带”。在头一阶段，由内部排序产生的递增路段被交替地放置在带1和带2上，直到输入被穷尽为止。然后带1和带2被重绕到它们开始的位置，从这些带上取出路段再次合并，得到其长度为原来路段长度两倍的新路段；这些新路段也象它们形成时那样交替地写到带3和带4上(如果带1包含的路段比带2多一个，则假定一个长度为0的额外的虚拟路段出现在带2上)，然后所有的带都被重绕，而且带3和带4的内容被合并成交替地记录到带1和带2上的四倍长度的路段。这个过程继续进行，每次把路段的长度加倍，直到仅仅剩下一个路段（即完全排好序的文件）为止。如果在内部排序阶段产生了S个路段，而且如果$2^{k-1}<S\leqslant 2^k$，则这个平衡的2路合并过程对所有数据恰好进行了$k=\lceil\log_2 S\rceil$次合并扫描。

例如，在上述通过容量为1000的一个内部存储对5000个记录排序的情况下，有$S=5$。排序过程的初始分布阶段按如下方式把五个路段放置到带上：

带1　$R_1\cdots R_{1000}$；$R_{2001}\cdots R_{3000}$；$R_{4001}\cdots R_{5000}$

带2　$R_{1001}\cdots R_{2000}$；$R_{3001}\cdots R_{4000}$　　(1)

带3　(空)

带4　(空)

然后，头一次合并扫描，按照它读带1和带2的顺序，在带3和带4上产生更长的路段如下：

带3　$R_1\cdots R_{2000}$；$R_{4001}\cdots R_{5000}$　　(2)

带4　$R_{2001}\cdots R_{4000}$

(一个虚拟的路段已经隐式地加在带2的结尾，使得带1上的最后的路段$R_{4001}\cdots R_{5000}$仅仅复写到带3上。）在所有的带被重绕之后，对于数据的下一遍扫描产生

带1　$R_1\cdots R_{4000}$　　(3)

带2　$R_{4001}\cdots R_{5000}$

(再次简单地复写路段$R_{4001}\cdots R_{5000}$；但是如果我们以8000个记录开始，则这时带2就将包含$R_{4001}\cdots R_{8000}$了。）最后在另一轮重绕之后，在带3上产生$R_1\cdots R_{5000}$，排序乃告完成。

平衡的合并能容易地推广到对于任何$T\geqslant 3$的T条带的情形。选择任意数P，$1\leqslant P<T$，而把T条带分成两“边”，P条带在左边，$T-P$条带在右边。初始的路段尽可能均匀地分配到左边的P条带上；然后从左到右进行P路合并，紧接着从右到左进行（$T-P$）路合并，等等，直到排序完成为止。P的最好选择通常认为是$\lceil T/2\rceil$（见习题3.4)。

平衡的两路合并是$T=4$，$P=2$时的特殊情况。用更多条带重新考虑上面的例子，取$T=6$和$T=3$。现在的初始分布是

带1　$R_1\cdots R_{1000}$；$R_{3001}\cdots R_{4000}$

带2　$R_{1001}\cdots R_{2000}$；$R_{4001}\cdots R_{5000}$　　(4)

带3　$R_{2001}\cdots R_{3000}$

第一遍合并扫描产生了

带 4　$R_1 \cdots R_{2000}$

带 5　$R_{2001} \cdots R_{5000}$　　(5)

带 6　(空)

(在带 3 上已经假定有一个虚拟路段。) 第二遍合并扫描完成了这个工作，把 $R_1 \cdots R_{5000}$ 放置到带 1 上。在这个特殊的情况下 $T=6$ 实际上和 $T=5$ 是一样的，因为第 6 条带仅当 $S \geqslant 7$ 时才使用。

三路合并实际上比两路合并要求更多的计算机处理时间，但是比起读、写和重绕带所需要的时间来，一般地均可忽略之。我们可以仅仅考虑带移动的数量，而得到一个相当好的运行时间的估计。(4) 和 (5) 中的例子只要求对数据扫两次，而 $T=4$ 时须扫三次。相比之下，$T=6$ 时这个合并所花费的时间仅仅约为前者的三分之二。

平衡合并十分简单，但是如果我们更仔细地观察，则立即发现，它不是处理上边讨论的特殊情形的最好途径！不从 (1) 进行到 (2) 并重绕所有的带，而是在带 3 和带 4 分别地包含 $R_1 \cdots R_{2000}$ 和 $R_{2001} \cdots R_{4000}$ 之后，即停止第一遍合并扫描，且使带 1 作好读 $R_{4001} \cdots R_{5000}$ 的准备，然后重绕带 2、3、4，最后通过在带 2 上进行三路合并来完成这个排序，则在这个过程中从带上读记录的总数量，同在平衡方案中的 5000＋5000＋5000＝15000 相对照，将仅仅是 4000＋5000＝9000，一个灵巧的计算机应有能力做到这一点！

其实，当有五个路段和四条带时，甚至能做得更好，办法是把它们作如下分布：

带 1　$R_1 \cdots R_{1000}$；$R_{3001} \cdots R_{4000}$

带 2　$R_{1001} \cdots R_{2000}$；$R_{4001} \cdots R_{5000}$

带 3　$R_{2001} \cdots R_{3000}$

带 4　(空)

然后对带 4 进行三路合并，紧接着进行带 3 和带 4 的一次重绕，再紧接着进行在带 3 上的三路合并，仅仅通过读 3000＋5000＝8000 个记录就完成了排序。

而且，当然，如果有六条带，则可以把初始的路段放置到带 1 到带 5 上，并且通过对带 6 进行五路合并，在一次扫描中就完成排序。这些考虑说明简单的平衡合并不是最好的，故考虑改进的合并型式是有趣的。

本章的以下部分，将更深刻地研究外部排序。在 5.4.1 节，我们考虑“内部排序”阶段，它产生初始的路段；其中特别有趣的是“替代选择”技术。它利用在大多数数据中存在的次序，来产生其长度实际上大大超过内部存储容量的初始路段。5.4.1 节也讨论了适合于多路合并的数据结构。

5.4.2 节到 5.4.5 节讨论了最重要的合并型式。当我们学习这些型式的特征时，最好先不忙于去跟那些烦人的磁带机和有待排序的实际数据打交道，而是先有一个相当朴素的磁带排序的概念。例如，可以冒失地（如上边所做的那样）假定，原来的输入记录在初始分布阶段神秘地出现；事实上，这些输入记录可能占用一条带，而且也可能甚至充满若干卷带，因为带并非无限长的！最好是，在对经典的合并样式获得原则性的理解之前，忽略这种过于实际的考虑。然后 5.4.6 节讨论强烈地影响合并型式选择的现实生活的约束。这个讨论使我们又回到实际中来。5.4.6 节中，使用在实际中出现的各种各样的假定，来比

较 5.4.2 到 5.4.5 节的基本合并型式。

5.4.7 和 5.4.8 节中讨论了不是以合并为基础的解决外部排序的某些其它方法。最后，5.4.9 节通过讨论诸如磁盘和磁鼓等这样的海量存储的重要排序问题，完成了对于外部排序的综述。

习题

1.〔15〕 正文建议首先进行内部排序，紧接着进行外部合并，为什么不能去掉内部排序阶段，简单地从一开始就把记录合并成越来越长的路段？

2.〔10〕 当举例的记录 $R_1R_2\cdots R_{5000}$ 使用一个 3 带平衡方法以 $P=2$ 进行排序时，相当于（1）到（3）的带的内容序列将是什么？请把这同 4 带合并比较；在初始的路段分布好之后，对所有的数据要做多少次扫描？

3.〔20〕 说明当 $P^k(T-P)^{k-1}<S\leqslant P^k(T-P)^k$ 时，应用于 S 个初始路段的平衡的（P，$T-P$）路合并花费 $2k$ 次扫描；而当 $P^k(T-P)^k<S\leqslant P^{k+1}(T-P)^k$ 时，它花费 $2k+1$ 次扫描。

给出对于（a）当 $T=2P$ 时，作为 S 的函数的扫描的精确次数，和（b）对于一般的 P 和 T，当 $S\to\infty$ 时，扫描的近似次数的一个简单公式。

4.〔HM15〕 对于 $1\leqslant P<T$，P 的什么值使得 $P(T-P)$ 取极大？

5.4.1 多路合并和替代选择

在 5.2.4 节中，我们研究了以两路合并为基础的内部排序方法，即把两个有序的序列合并成一个有序序列的过程。不难将其推广成 P 路合并的思想，其中 P 个输入路段被合并成一个输出路段。

假定，我们已经给定 P 个递增的路段，即，其键按非减次序排列的记录。把它们合并的明显方式，是考察每个路段的头一个记录，选择其中键为最小的那个；输出此记录，同时从输入中撤消它并重复这一过程。在任何给定的时刻，我们都仅需考虑 P 个键（每个输入路段一个键），并选择最小的。如果有两个或者更多个最小的键，那就任意选择一个。

当 P 不太大时，通过简单地进行 $P-1$ 次比较来求当前诸键的最小者，是一种方便的选择方法。但当 P 比如说是 8 或更大时，通过使用 5.2.3 节所描述的那样一株选择树，我们就可以节省工作量；于是一旦已经建立了树，每次就将仅仅需要大约 $\log_2 P$ 次比较。

例如，考虑四路合并的情形，用两级选择树：

步骤 1.

$$087\begin{cases}087\begin{cases}087 & 503 & \infty\\ 170 & 908 & \infty\end{cases}\\ 154\begin{cases}154 & 426 & 653 & \infty\\ 612 & \infty\end{cases}\end{cases}$$

步骤 2.

$$087\quad 154\begin{cases}170\begin{cases}503 & \infty\\ 170 & 908 & \infty\end{cases}\\ 154\begin{cases}154 & 426 & 653 & \infty\\ 612 & \infty\end{cases}\end{cases}$$

步骤 3.　087　154　170 $\begin{cases} 170 \begin{cases} 503 & \infty \\ 170 & 908 \end{cases} & \infty \\ 426 \begin{cases} 426 & 653 \\ 612 & \infty \end{cases} & \end{cases}$

⋮

步骤 9.　087　154　170　426　503　612　653　908　$\infty \begin{cases} \infty \begin{cases} \infty \\ \infty \end{cases} \\ \infty \begin{cases} \infty \\ \infty \end{cases} \end{cases}$

在这个例子中，每个路段的末尾都放置一个附加的键“∞”，这是一种结束合并的很好的方法。由于外部合并一般都涉及非常长的路段，故添加带有∞键的记录后，无论是数据的长度或合并的工作量都不会有实质性的增加，而且这样的“哨兵”记录经常用作一项界定一个文件上的路段的有用的技术。

在上述过程开始之后，每一步都把最小的元素换成同一路段中的随后的元素，并且改变在选择树中对应的路径。于是，该树在步骤 1 中包含 087 的三个位置在步骤 2 中被改变；在步骤 2 中包含 154 的三个位置在步骤 3 中被改变，等等。在选择树中用另一个键来代替一个键的过程，称为替代选择。

我们可以从几个方面来考察这种四路合并。从一种观点看，它等价于象联立子程序那样并发执行的三个两路合并，选择树中的每一个节点，都代表一个包含在并发合并过程中的序列。选择树实质上也象一个优先队一样操作，这个队具有“最小者先出”的规则。

如同在 5.2.3 节一样，我们可以使用一个堆代替一株选择树，以实现优先队（当然堆将安排成最小元素出现在顶上，而不是最大的元素在顶上，颠倒等式 5.2.3-3 的顺序）。由于一个堆并没有固定的大小，因此我们避免使用“∞”键；当堆成空时，合并完成。另一方面，应用外部排序时通常都要处理相当长的记录和键，故堆中填入指向键的指针以代替键本身；下边我们将看到，选择树可以以这样一种方便的方式由指针表示，在这种情况下，它们可能优越于堆。

“失利者”的树　图 62 示出了具有 12 个外部（方框）节点和 11 个内部（圆圈）节点的完备二叉树；如果把这株树看作是选择最小键的一次锦标赛的话，则外部节点中已填入键，而内部节点中已经填入“胜利者”。每个节点上边小号字数字标明了为完备的二叉树分配连续存储位置的传统方法。

当最小的键 061 用图 62 中选择树的另一个键来代替时，为确定选择树的新状态我们需要考察键 512、087 和 154，而非其它现存的键。把这树看作一场锦标赛，这三个键就是在竞赛中同 061 对垒的失利者。这提示，我们在这株树的内部节点中真正应该存储的是每次对垒的失利者，而不是胜利者；这样，为更新这株树所需要的信息就很容易拿到了。

图 63 示出了和图 62 同样的树，但它表示失利者而不是胜利者。在这株树的顶部已经附加了一个额外的数 0，指出这场锦标赛的冠军。注意，除了冠军之外每个键都恰巧是一次比赛的失败者（参考 5.3.3 节），所以每个键都在一个外部节点中出现一次和在一个内

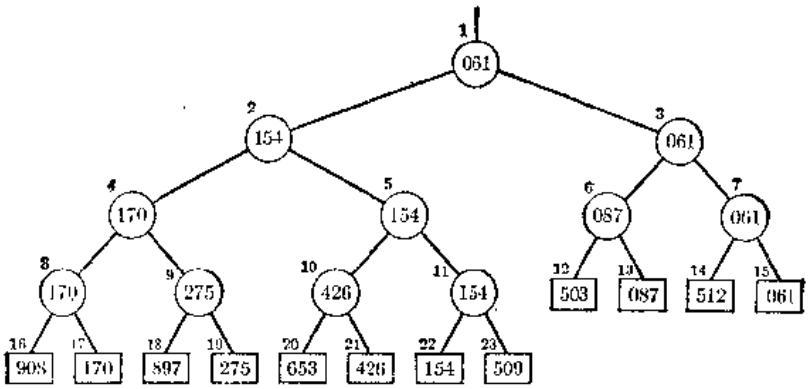


图62　使用一个其节点编号成1到23的二叉树来选择最小键的一次锦标赛

部节点中出现一次。

实际上，图63底部的外部节点表示存于计算机存储器中的相当长的记录，而内部节点表示指向这些记录的指针。注意，P路合并恰巧调用P个外部节点和P个内部节点，每个都在相邻的组中，于是这本身就提示了若干有效的存储分配方法。不难看出，如何利用替代选择的“面向失利者”的树；我们将在本节的稍后部分稍微详细地讨论这个算法。

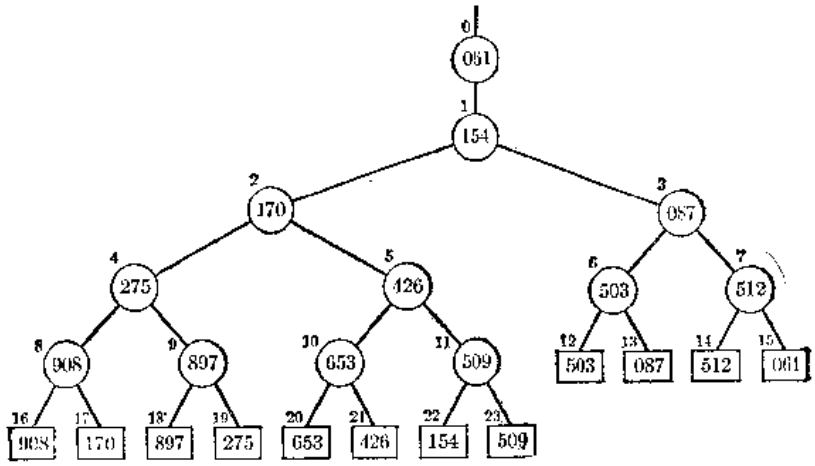


图63　和图62相同的锦标赛，但示出失利者而不是胜利者；冠军出现于最顶上

由替代选择所产生的初始路段　如果基本上用输入数据本身进行P路合并，则替代选择技术也可以用于外部排序的第一阶段！在这种情况下，把P取得相当大，使得内部存储基本上已被充满。当一个记录输出时，它就被下一个输入记录所代替。如果新记录的键小于刚才输出的键，则不能把它包括在当前的路段中；否则可以通常的方式把它送入选择树中，并且它将形成当前正在产生的路段的一部分。于是每个路段就能包含多于P个记录，尽管任何时候在选择树中决没有多于P个记录。表1示出$P=4$的这一过程：带圆括号的数将要包括进随后的路段中。

形成初始路段的这一重要方法，首先是由哈罗德·H·西沃德描述的〔Moster's Thesis, Digital Computer Laboratory Report R-232(Mass.Inst.of Technology, 1954), 29-30〕，他给出的理由使人们确信，当应用于随机数据时，这些路段将包含1.5 P个以上的记录，1950年左右A.I.杜米（A.I.Dumey）在谈到工程研究协会设计的一部特殊

表 1　四路替代选择的例子

存储内容				输出
503	087	512	061	061
503	087	512	908	087
503	170	512	908	170
503	897	512	908	503
(275)	897	512	908	512
(275)	897	653	908	653
(275)	897	(426)	908	897
(275)	(154)	(426)	908	908
(275)	(154)	(426)	(509)	(路段结束)
275	154	426	509	154
275	612	426	509	275
		等等		

排序设备时，也提出了这个思想，但他没有发表。“替代选择”这一名称，是由 E.H. 弗朗德杜撰的〔JACM 3 (1956)，154〕，他解释说：“提不出所产生序列的预期长度的公式，但是实验提示$2P$是一个合理的预测。”

为了说明$2P$确是预期的路段长度，E.F.穆尔（E.F.Moore）发现了一种聪明的方式，他把这个情况同在一个圆形轨道上的扫雪机作了比较（U.S.Patent，2983904，(1961)，cols3-4)。考虑图 64 中所示的情况；雪片均匀地落在一条圆形的路上，一台孤独的扫雪机不断地清扫雪。一旦已经把雪扫出路外，它就从这个系统消失。可以通过实数x来指明路标，$0 \leqslant x < 1$；落在位置x的雪片表示其键为x的输入记录，而扫雪机表示替代选择的输出。扫雪机的基本速度同它所遇到的雪的高度成反比，且情况是完全平衡的，即在路上雪的总量在所有时刻都恰巧是P。每当扫雪机通过点 0 时，便在输出中形成一个新的路段。

在这个系统已经运行一段时间之后，显然，直观上它将趋于一个稳定状态，在这个状态下，扫雪机以恒速运行（由于这个轨道的圆形对称性）。这意味着当遇到扫雪机时，雪处于常数高度中，而且如图 65 所示，这高度在扫雪机前面线性地降低。由此推出，在一个循环（即路段长度）中所扫除的雪的体积是任何一个时候存在的量（即P）的两倍。

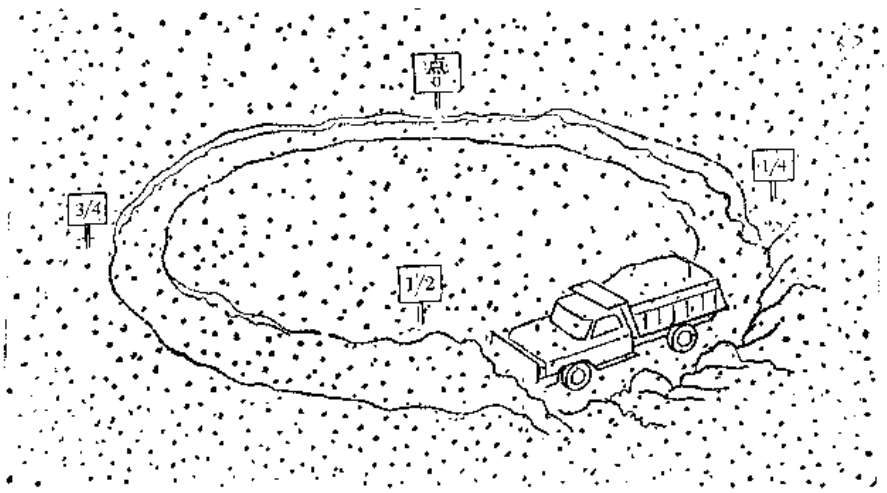


图64　在环形圆圈上连续不停运行的扫雪车

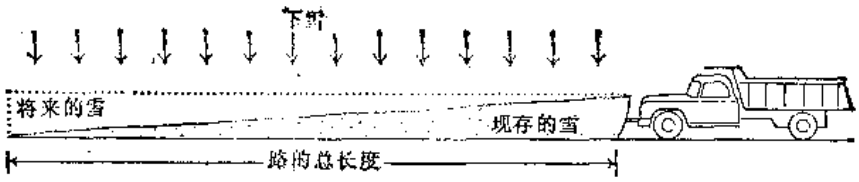


图65 横断面，示出当这个系统处于它的“稳定状态”时机犁前雪的高度的变化

在许多商业应用中输入数据不是完全随机的，它已经有某种程度的既定次序；因此由替代选择产生的路段很可能包含甚至$2P$个以上的记录。我们将看到，外部合并排序所需要的时间在很大程度上受初始分布阶段所产生的路段数的支配，因而替代选择显得特别可取；其它类型的内部排序由于存储大小的限制，将产生大约二倍之多的初始路段。

现在让我们详细考虑由替代选择建立初始路段的过程。下列算法是由约翰·R.沃尔特斯（John R. Walters）、詹姆斯·佩因特（James Painter）及马丁·扎尔克（Martin Zalk）给出的，他们在1958年把它用于菲尔戈（Philco）2000的合并路段程序中。它加入了一个很好的方法，通过相当简单和一致的逻辑来建立选择树的初态，并区分了属于不同路段的记录，以及清理最后一个路段（由替代选择产生的最后路段的适当处理，是带点窍门的，对于程序员说来它势必是一个令人困惑的难点）。基本的思想是把每个键都当作一个对偶（S，K），其中K是原来的键，而S是这个记录所属的路段的编号，当这样扩展的键按字典编辑次序排列，并以S作主键，以K作辅键时，我们就得到由替代选择产生的输出序列。

以下的算法使用了包含P个节点的一个数据结构来表示选择树；假定第j个节点$X[j]$包含从$\mathrm{LOC}(x[j])=L_0+cj$开始的$c$个字，$0\leqslant j<P$，它既表示图63中的内部节点号$j$又表示外部节点号$P+j$。在每个节点中有若干命名了的场：

KEY＝存储在这个外部节点中的键字；

RECORD＝存储在这个外部节点中的记录（包括KEY作为一个子场）；

LOSER＝指向存储在这个内部节点中的“失利者”的指针；

RN＝由LOSER指出的记录路段号；

FE＝指向这株树中在这个外部节点上方的内部节点的指针；

FI＝指向这株树中在这个内部节点上方的内部节点的指针。

例如，当$P=12$时，图63的内部节点号5和外部节点号17都将通过场KEY＝170、LOSER＝L_0+9c（外部节点号21的地址）、FE＝L_0+8c　FI＝L_0+2c在$X[5]$中表示出来。

场FE和FI有常数值，所以它们不必明显地出现于存储器中；然而，外部排序的初始阶段，有时跟不上输入/输出设备的速度，故通常值得把这些多余的值作为数据存起来，而不是每次重新计算它们。

算法R（替代选择）这个算法顺序地从一个输入文件读入记录，并把它们顺序地写在一个输出文件上，产生RMAX个路段，除最后的路段外，每个路段的长度均大于或等于P。共有$P\geqslant 2$个节点$Y[0]$，…，$X[P-1]$，每个节点所含的场如上所述。

R1. 〔初始化〕置RMAX←0，RC←0，LASTKEY←∞，Q←LOC($X[0]$)，

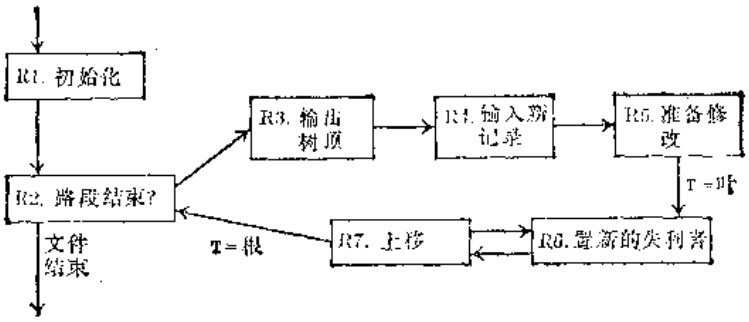


图66 用替代选择作初始路段

以及 RQ←0。(RC 是当前路段的编号，而 LASTKEY 是最后输出记录的键。LASTKEY 的初值应当大于任何可能的键；参考习题8。)对于任何 $0 \leqslant j < P$，当 J=LOC($X[j]$) 时，置$X[j]$的初始内容如下：

$$\mathrm{LOSER}(J) \leftarrow J;\ \mathrm{RN}(J) \leftarrow 0;$$

$$\mathrm{FE}(J) \leftarrow \mathrm{LOC}(X[\lfloor (P+j)/2 \rfloor]);\ \mathrm{FI}(J) \leftarrow \mathrm{LOC}(X[\lfloor j/2 \rfloor])$$

(LOSER(J) 和 RN(J) 的赋值都是人为的办法，为的是通过考虑一个虚构的编号为 0 的路段来建立树的初态。该路段决不输出，这是技巧，见习题10。)

R2. 〔路段结束？〕如果 RQ=RC，则继续转到步骤R3。(否则 RQ=RC+1，此时我们刚刚完成编号为 RC 的路段；一个合并型式所要求的对于排序的随后的扫描的任何特殊动作将在这时完成。)如果 RQ>RMAX，则停业，否则置 RC←RQ。

R3. 〔输出树顶〕(现在Q指向"冠军"，而 RQ 是它的路段号。) 如果 RQ≠0，则输出 RECORD(Q) 并置 LASTKEY←KEY(Q)。

R4. 〔输入新记录〕 如果输入文件穷尽了，则置 RQ←RMAX+1 且继续转到步骤 R5。否则置 RECORD(Q) 为输入文件的下个记录。如果 KEY(Q)<LASTKEY (于是这个新记录不属于当前的路段)，则置 RQ←RQ+1，然后如果 RQ>RMAX，则置 RMAX←RQ。

R5. 〔准备更新〕 (现在Q指向一个新记录，其路段号是 RQ。) 置 T←FE(Q)(T 是一个指针变量，它将沿树上移)。

R6. 〔置新的失利者〕 如果 RN(T)<RQ 或者如果 RN(T)=RQ 并且 KEY(LOSER(T))<KEY(Q)，则对换 LOSER(T)←→Q，RN(T)←→RQ。(变量Q和 RQ 记住当前的胜利者和它的路段号)。

R7. 〔上移〕 如果 T=LOC($X[1]$) 则返回到R2，否则置 T←FI(T)并返回R6。

算法R中的输入和输出一次只涉及一个记录，然而实际上最好是读和写相当大块的记录。因此，存储器中有某些输入和输出缓冲区，它们实际上在幕后降低P的大小，我们在5.4.6节说明这一点。

E.H.费朗德〔JACM 3 (1956)，154〕提出扩展的替代选择，使得每当一个输入键小于 LASTKEY (于是它不加入当前的路段)，但是大于或等于已经写到带上的最后的键(于是它毕竟可以加入当前的路段)时，它将被插入到输出缓冲区中。进而，某些计算机有"分散读"和"集中写"的设备，可以从非连续单元输出或输入到非连续单元，这导致

把缓冲区同选择树重叠起来的某些技术。

***路段的延迟重新组成** 使用我们将称之为自由度的概念，R.J.丁斯莫尔（R.J.Dinsmore）〔CACM 3(1965).48〕已经提出了改进替代选择的一项非常有趣的方法。如同我们已经看到的，磁带上在一个路段之内的每组记录是按非减次序排好的，于是它的头一个元素是最低的而最后元素是最高的。在通常的替代选择过程中，一个路段内每组的最低元素决不小于该路段中前面那组的最高元素；这是"1个自由度"。丁斯莫尔提议把这条件放松成为"m个自由度"，其中每组的最小元素可以小于前一组的最高元素，只要它不小于本路段前的m个不同的组中的最高元素就行。如同以前一样，在单个组内的记录是有序的，但相邻的组不必是有序的。

例如，假设每组恰有两个记录；下列组序列是具有三个自由度的一个路段：

$$|08\quad 50|06\quad 90|17\quad 27|42\quad 67|51\quad 89| \qquad (1)$$

该路段将要包含的下一个组，必须以一个不小于{50，90，27，67，89}的第三个最大元素（即67）的元素开始。如果仅有两个自由度的话，序列（1）将不是一个路段，因为17既小于50，也小于90。

具有m个自由度的一个路段，当它在排序的下一个阶段被读入时，可以被"重新组成"，使得对于所有实用的目的说来，它都是在通常意义下的一个路段。我们从把m个组读到m个缓冲区开始，对它们进行m路合并；当穷尽了一个缓冲区时，以第$m+1$个组代替它，等等。用这种办法，我们可以重新把这一路段恢复成一单个序列，因为每个新近读的头一个字都必须大于或等于刚穷尽的组的最后一个字（免得它小于在它之前的m个不同组中的最高元素）。这个重新组成路段的方法，实际上就象对于所有的输入组都使用一台磁带机的m路合并一样！重新组成过程的工作方式象一个联立子程序，它被调用来一次发送路段的一个记录。我们可以对从不同的磁带机上来的具有不同自由度的路段加以重新组成，并且合并得到的路段，所有这些都在同一个时间进行，这种方法实际上就象本节开始所说明的，四路合并可以想象作一次进行若干两路合并一样。

这个有独创性的思想尚未彻底分析过。但T.O.埃斯皮里得（T.O.Espelid）已经说明如何来扩充扫雪机的类似性，以得到对于这个行为的一个近似公式〔BIT16(1976)，133-142〕。按照这个同经验测试很一致的公式，路段长度将大约是$2P+b(m-1.5)(2P+b(m-2))/(2P+b(2m-3))$。当$b$是组区大小且$m\geqslant 2$时，这样一种增加尚不足以证明由此而增加的复杂性是合理的；另一方面，在第二个排序阶段，如果有足够的地方可安排相当大量的缓冲区，它可能是有利的。

***自然的选择** W·D·弗雷泽和黄泽权已经剖析了增加由替代选择产生的路段长度的另一种方式，他们的想法是基本上象算法R那样做。但当在步骤R4中KEY(Q)<LASTKEY时，新的RECORD(Q)不保留在这株树中，而是被输出到一个外部库中，并读入另一个新的记录，这个过程延续到这个库被一定数量的记录P'充满为止；这时，当前路段的剩下部分从该树中输出，而存于库中的项目即用作下一路段的输入。

这个方法一般能产生比替代选择更长的路段，因为它避免了来自下个串的"死的"记录，不让它们来充斥该树；但它需要额外的时间以从库进行输入和输出到库中。当$P'>P$时，某些记录有可能两次入库，但当$P'\leqslant P$时，这将决不可能发生。

弗雷泽和黄泽权对他们的方法进行了广泛的实验测试，得知当P相当大（比如说$P\geqslant 32$）和$P'=P$时，随机数据的平均路段长度由eP给出，其中$e=2.718$是自然对数底。由于这个现象，以及由于这个方法是对于简单的替代选择的一种渐进改良，自然地使他们称呼他们的方法为自然的选择。

通过再次考虑图 64 的扫雪机并且应用初等微积分，可以证明路段长度的“自然”定律。设L是道路的长度，并设$x(t)$是在时间$t(0\leqslant t\leqslant T)$时扫雪机的位置。当雪暂时停止而扫雪机回到它开始的位置（清除了在它的道路上剩下的P个单位的雪）时，假定在时间T时库是满的。除了“平衡条件”外，这个情况和以前是一样的；代替在所有时刻道路上有P个单位的雪，我们有P个单位的雪在扫雪机之前，而且库（在扫雪机之后）增加成$P'=P$个单位。如果在一时间区间dt内扫雪机前进dx，则$h(x,\ t)dx$记录是输出，这里$h(x,\ t)$是在时间t和位置$x=x(t)$（以适当单位测量）下雪的高度；因此，对所有x，$h(x,\ t)=h(x,\ 0)+Kt$。由于在存储器中的记录数保持不变，$h(x,\ t)$也是作为在扫雪机前头之输入的记录数，即$Kdt(L-x)$，其中K是落雪的速度（见图67）。于是

$$\frac{dx}{dt}=\frac{K(L-x)}{h(x,\ t)} \tag{2}$$

幸而，结果证明每当$X=X(t)$和$0\leqslant t\leqslant T$时，$h(x,\ t)$是等于KT的常数。因为在扫雪机通过点$X(t)$后，雪稳定地降落到该位置达$T-t$个时间单位，在它返回前还要加上t个时间单位。换句话说，在已达到稳定状态的前提下，每个旅程都是一样的，扫雪机看到在它的旅程中所有的雪都处于同样的高度，因此清扫的雪的总量（路段长度）是KTL；而存储器中雪的数量是在时间T后清除的数量，即是$KT(L-X(T))$。使得$X(0)=0$的（2）的解是

$$X(t)=L(1-e^{-t/T}) \tag{3}$$

因此$P=KTLe^{-1}=$(路段长度)$/e$；而这就是我们所要证明的。

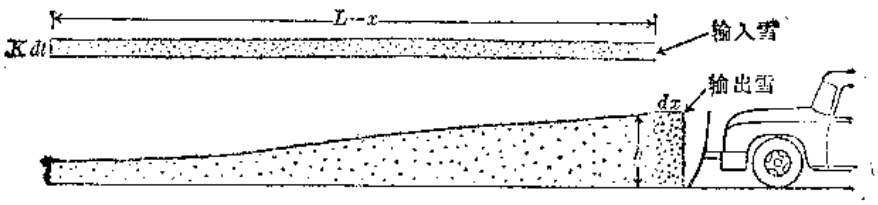


图67　同量的雪被输入和输出；扫雪机在时间dt内移动dx

习题 21-23 说明，这个分析可以推广到一般的P'的情况；例如，当$P'=2P$时，平均路段长度得知是$e^{\theta}(e-\theta)P$，其中$\theta=\frac{1}{2}\left(e-\sqrt{e^2-4}\right)$，这大概是还没有人能立即猜出的结果；表 2 说明了路段长度对于库的大小的依赖性；通过查这张表，在一个给定的计算机环境中自然选择的有用性即可估计出来。

如同T.C.丁(T.C.Ting)和Y.W.王（Y.W.Wang)在Comp. J.20 (1977)298-301中所讨论的那样，延迟路段—重新组成和自然选择的思想可以联合在一起。

表 2　自然选择产生的路段长度

库　大　小	路段长度	参　　数	库大小	路段长度	参　　数
1.00000P	2.71828P	1.00000	0.38629P	2.00000P	0.69315
2.00000P	3.53487P	1.43867	1.30132P	3.00000P	1.15881
3.00000P	4.16220P	1.74773	2.72294P	4.00000P	1.66862
4.00000P	4.69445P	2.01212	4.63853P	5.00000P	2.16714
5.00000P	5.16369P	2.24938	21.72222P	10.00000P	4.66667
10.00000P	7.00877P	3.17122	5.29143P	5.29143P	2.31329

在习题22中定义了"参数" $k+\theta$。

***替代选择的分析**　现在让我们转回到没有辅助库的替代选择的情况。扫雪机的类比，给了我们在"极限情况下"的由替代选择所得到的平均路段长度的一个相当好的估计，但是通过应用我们在 5.1.3 节中做的关于排列中路段的研究，有可能获得算法 R 的许多更精确的信息。为此目的，假定输入文件是 0 和 1 之间的独立随机实数的一个任意长的序列，设

$$g_P(z_1,\ z_2,\ \cdots,\ z_k)=\sum_{l_1,l_2\cdots,l_k\geqslant 0} a_P(l_1,\ l_2,\ \cdots,\ l_k)z_1^{l_1}z_2^{l_2}\cdots z_k^{l_k}$$ ●

是由在这样一个文件上的 P 路替代选择所产生的路段长度的母函数，其中 $a_P(l_1,\ l_2,\ \cdots,\ l_k)$ 是头一个路段长度为 l_1，第二个长度为 l_2，…，第 k 个长度为 l_k 的概率。下列"独立性定理"是一个基本的定理，因为它把这个分析归结为 $P=1$ 的情况：

定理K　$g_P(z_1,\ z_2,\ \cdots,\ z_k)=g_1(z_1,\ z_2,\ \cdots,\ z_k)^P$

证明　设输入键是 X_1，X_2，X_3，…。按照它们在这株树中所处的外部节点的位置，算法 R 把它们划分为 P 个子序列；包含 X_n 的子序列由值 X_1，…，X_n 确定。因此每一个子序列是 0 和 1 间独立的随机数的一个独立序列。进而，替代选择的输出恰是对这些子序列进行一次 P 路合并所应得到的；一个元素属于一个子序列的第 j 个路段的充要条件是，它属于由替代选择产生的第 j 个路段（因为在步骤 R4 中，LASTKEY 和 KEY(θ) 属于同一个子序列）。

换句话说，我们也完全可以假定，算法 R 正被应用于 P 个独立的随机输入文件，而且步骤 R4 从对应于外部节点 0 的文件读下一个记录；在这个意义下，该算法等价于一个 P 路合并，以"下坡"来标志诸路段的结束。

于是，当且仅当诸子序列有长度分别为 $(l_{11},\ \cdots,\ l_{1k})$，…，$(l_{P1},\ \cdots,\ l_{Pk})$ 的路段时，输出中有长度为 $(l_1,\ \cdots,\ l_k)$ 的路段。其中 l_{ij} 是满足 $\sum_{1\leqslant i\leqslant P} l_{ij}=l_j$ 的某些非负整数，$1\leqslant j\leqslant k$。由此得出

$$a_P(l_1,\ \cdots,\ l_k)=\sum_{\substack{l_{11}+\cdots+l_{P1}=l_1\\ \vdots \\ l_{1k}+\cdots+l_{Pk}=l_k}} a_1(l_{11},\ \cdots,\ l_{1k})\cdots a_1(l_{P1},\ \cdots,\ l_{Pk})$$

而这等价于所求的结果。

我们已经在 5.1.3 节讨论了当 $P=1$ 时，第 k 个路段的平均长度 L_k，在表5.1.3-2中已列出了这些值。定理K蕴涵对于一般的 P，第 k 个路段的平均长度是当 $P=1$ 时的平均长度的 P 倍，即 L_kP；而且方差也是 P 倍。所以路段长度的标准离差同 $\sqrt{P}$ 成比例。这些

● 此式原文错，P 应作为 g 和 a 的下标出现。——译注

结果是 1958 年左右由 B · J · 加斯纳首先导出的。

于是，对于随机数据，由算法 R 产生的头一个路段大约包含 $(e-1)P=1.718P$ 个记录。第二个路段，大约是 $(e^2-2e)P=1.952P$ 个记录。第三个，大约是 $1.996P$；而随继的路段，将非常接近于包含 $2P$ 个记录，直到我们得到最后两个路段（见习题14）为止。这些路段长度的大多数标准离差都近似于 $(4e-10)P\approx 0.934\sqrt{P}$〔CACM6 (1963), 685-687〕。进而，习题 5.1.3-10 说明，头 k 个路段的总长度将相当接近于 $\left(2k-\frac{1}{3}\right)P$，并有 $\left(\left(-\frac{2}{3}k+\frac{2}{9}\right)P\right)^{1/2}$ 的标准离差，在习题 5.1.3-9 和 11 中导出了母函数 $g_1(z, z, \cdots, z)$ 和 $g_1(1, 1, \cdots, 1, z)$。

上边的分析已经假定输入文件是无限长的，但定理 K 的证明表明，在至少包含 $l_1+\cdots+l_k+P$ 个元素的任何随机输入序列中，都将得到相同的概率 $a_p(l_1, \cdots, l_k)$。所以上述结果在小的标准离差的观点下，对于比如说大小为 $N>(2k+1)P$ 的文件是可应用的。

我们将看到某些应用，其中合并型式要求某些路段是递增的，而某些是递减的，由于在一个递增的路段结束时，在存储器中累积的剩余往往包含平均比随机数更小些的数，故顺序方向的改变减少了路段的平均长度。例如，考虑一台扫雪机，在它每次到达一条直路的终点时必须跑一个 U 形转弯；它将非常快速地通过刚才扫过的区域。当颠倒方向时，随机数据的路段长度将在 $1.5P$ 和 $2P$ 之间变动（见习题24）。

习题

1. 〔10〕 在这节开头的四路合并例子中，步骤 4 是什么？

2. 〔12〕 如果键 061 以 612 代替，对于图 63 的树将作什么变动？

3. 〔16〕 (E. F. 摩尔) 当应用四路替代选择于下列一串单词时，产生的输出是什么？

fourscore and seven years ago our fathers brought forth on this continent a new nation conceived in liberty and dedicatd to the proposition that all men are created equal

(使用通常的字母顺序，把每个词处理作一个键。)

4. 〔16〕 把四路自然选择应用于习题 3 的句子，使用容量为 4 的一个库。

5. 〔00〕 真或假：仅当 P 是 2 的乘方时或者是两个 2 的乘方之和时，使用一个树的替代选择才有效。

6. 〔15〕 算法 R 指明，P 必须 $\geqslant 2$；应如何稍微修改这个算法，以使它对于所有 $P\geqslant 1$ 成立？

7. 〔17〕 当全然没有输入时，算法 R 做什么？

8. 〔20〕 算法 R 利用了一个人为的键 "∞"，它必须大于任何可能的键。证明：要是真有一个键等于 ∞ 的话，这个算法可能失误，并说明在真正的 ∞ 的实现并不方便的情况下，如何来修改这个算法？

▶9. 〔23〕 你将怎样修改算法 R，使得某些指定的路段（依赖于 RC）按递增次序输出，而其它的按递减次序输出？

10. 〔23〕 在步骤R1中 LOSER 指针的初态通常不对应于任何实际的锦标赛，因为外部节点$P+j$不会处于内部节点j之下的子树中，说明为什么算法R仍然有效。〔提示：如果在步骤R1中{LOSER (LOC (X〔0〕), …, LOSER (LOC (X〔$P-1$〕))} 被置成{LOC(X〔0〕), …, LOC (X〔$P-1$〕)}的任何一个排列，则这个算法是否有效？〕

11. 〔M25〕 真或假：假定输入是随机的，在步骤R4中 KEY(Q)<LASTKEY 的概率近似于$\frac{1}{2}$。

12. 〔M46〕 详细分析算法R的每一部分被执行的次数，例如，在步骤R6中所作的交换有多频繁？

13. 〔13〕 为什么由替代选择产生的第二个路段通常都大于头一个路段？

▶14〔HM25〕 用扫雪机的类比来估计由输入数据的一个长序列通过替代选择所产生的最后两个路段的平均长度。

15. 〔20〕 真或假：由替代选择产生的最后路段决不包含多于P个记录，请讨论你的答案。

16. 〔M26〕 试求一个文件$R_1R_2\cdots R_N$将由P路替代选择在一次扫描中完全排序的一个“简单的”必要和充分条件。当输入是{1，2，…，N}的随机排列时，发生这种情况的概率（作为P和N的函数）是什么？

17. 〔20〕 当输入键处于递减次序下，$K_1\geqslant K_2\geqslant\cdots\geqslant K_N$时，由算法R产生的输出是什么？

▶18. 〔22〕 如果算法R再次应用于由算法R产生的一个输出文件，则将发生什么情况？

19. 〔HM22〕 用扫雪机的类比来证明，由替代选择产生的头一个路段的长度近似为$(e-1)P$个记录。

20. 〔HM24〕 当$P=P'$时，由自然选择产生的头一个路段近似地有多长？

▶21. 〔HM23〕 试确定当$P'<P$时由自然选择产生的路段的近似长度。

22. 〔HM40〕 这一习题的目的是来确定当$P'>P$时，在自然选择中得到的平均路段长度。命$x=k+\theta$是一个$\geqslant 1$的实数，其中$k=\lfloor x\rfloor$和$\theta=x \bmod 1$。并考虑函数$F(x)=F_k(\theta)$，其中$F_k(\theta)$是由母函数

$$\sum_{k\geqslant 0} F_k(\theta)z^k=e^{-\theta z}/(1-ze^{1-z})$$

定义的多项式，于是$F_0(\theta)=1$，$F_1(\theta)=e-\theta$，$F_2(\theta)=e^2-e-e\theta+\frac{1}{2}\theta^2$，等等。

假设一台扫雪机在时间0启动，开始模拟自然选择的过程，并假设在T个时间单位之后，恰有P个雪花已经落在它后面。这时，第二台扫雪机开始相同的旅程，它在时间$t+T$时所占的位置就是第一台扫雪机在时间t时所占的位置。最后，在时间xT，恰有P'个雪花落在头一台扫雪机之后，它在一瞬间把剩下一段路全扫完，随之即告消失。

利用这一模型来表示自然选择的过程，说明当

$$P'/P=k+1+e^{\theta}\Big(xF(x)-\sum_{0\leqslant j\leqslant x}F(x-j)\Big)$$

时，得到一个长度等于 $e^{\theta}F(\nu)P$ 的路段。

23.〔$HM35$〕 上题分析了当记录按同一次序写入和读出库（即先进先出）时的自然选择。如果以完全随机的次序读入上一路段存进库中的内容，就象库中的记录在两个路段之间已整个地重“洗”了一样，试求应得到的近似路段长度。

24.〔$HM39$〕 本题的目的是分析因偶然改变替代选择中路段的方向所引起的效果。

a）设 $g_P(z_1, z_2, \cdots, z_k)$ 是如同在定理 K 中那样定义的一个母函数，但同时，k 个路段中的每个都已被确定是递增还是递减的。例如，我们可以说所有奇数编号的路段是递增的，所有偶数编号的路段是递减的。试说明对于这种类型的 2^k 个函数的每一个，都有定理K成立。

b）作为（a）的推论，我们可以假设 $P=1$。我们也可以假定，输入是 0 和 1 之间的独立随机数的一个一致分布序列。设

$$a(x, y)=\begin{cases} e^{1-x}-e^{y-x} & \text{如果 } x\leqslant y \\ e^{1-x} & \text{如果 } x>y \end{cases}$$

设 $f(x)dx$ 是某个递增路段以 x 开始的概率，证明 $\left(\int_0^1 a(x, y)f(x)dx\right)dy$ 是紧接着的路段以 y 开始的概率。〔提示：对已知的 x 和 y，对于每个 $n\geqslant 0$，考虑 $x\leqslant X_1\leqslant\cdots\leqslant X_n>y$ 的概率。〕

c）考虑以概率 p 改变方向的路段；换句话说，在头一个路段后每个路段的方向，有 $q=(1-p)$ 的机会被随机地选择成和以前的路段一样，而选择相反方向的机会是 p（于是当 $p=0$ 时，所有的路段都有相同的方向；当 $p=1$ 时，这些路段都改变方向；而当 $p=\frac{1}{2}$ 时，这些路段是独立地随机的）。设

$$f_1(x)=1,\ f_{n+1}(y)=p\int_0^1 a(x, y)f_n(1-x)dx+q\int_0^1 a(x, y)f_n(x)dx$$

说明当第（$n-1$）个路段递增时第 n 个路段以 x 开始的概率是 $f_n(x)dx$，当第（$n-1$）个路段递减时第 n 个路段以 x 开始的概率是 $f_n(1-x)dx$。

d）求对于“稳态”方程

$$f(y)=p\int_0^1 a(x, y)f(1-x)dx+q\int_0^1 a(x, y)f(x)dx,$$

$$\int_0^1 f(x)dx=1$$

的一个解。〔提示：证明 $f^{\prime\prime}(x)$ 是独立于 x 的。〕

e）证明在（c）中的序列 $f_n(x)$ 相当迅速地收敛到（d）中的函数。

f）证明，以 x 开始的一个递增路段的平均长度是 e^{1-x}。

g）最后，把上述这些结果归结到一起，以证明下列定理：在替代选择中，如果连续一系列路段的方向是独立地以概率 p 逆转的，则平均路段长度趋于 $(6/(3+p))P$（关于这定理的 $p=1$ 的情况，首先是由克努特导出的〔$CACM$ 6 (1963)，685-688〕；$p=\frac{1}{2}$ 的情况首先由 A. G. 康赫姆（A. G. Konheim）于1970 年加以证明。）

25.〔$HM40$〕 考虑下列过程：

N1. 读一个记录到容量仅为一个字的“库”中，然后读另一个记录R并命K是它的键。

N2. 输出这个库，置LASTKEY为它的键，并置这个库为空。

N3. 如果K<LASTKEY，则输出R和置LASTKEY←K并转到N5。

N4. 如果库非空，则转到N2；否则把R送入库中。

N5. 读入新记录R，并设K是它的键。转到N3。

这实际上等价于$P=1$和$P'=1$或2的自然选择（取决于你是选定在库满的时刻来弄空它，还是在它大约溢出了的时刻弄空它）。唯一的区别是它产生递减的路段，而且它决不停止。对于本题的目的说来，后边的异常情况是方便的和无害的假定。

如同在习题24中那样进行，设$f_n(x, y)dydx$是恰在第n次执行步骤N2之后(LASTKEY, K)分别取值(x, y)的概率。证明存在一个变量的函数$g_n(x)$，使得当$x<y$时$f(x, y)=g_n(x)$，而当$x>y$时$f_n(x, y)=g_n(x)-e^{-y}(g_n(x)-g_n(y))$。这个函数$g_n(x)$由关系式$g_1(x)=1$，

$$g_{n+1}(x)=\int_0^x e^u g_n(u)du+\int_0^x dv(v+1)\int_v^1 du((e^v-1)g_n(u)+g_n(v))$$

$$+x\int_x^1 dv\int_v^1 du((e^v-1)g_n(u)+g_n(v))$$

来定义。进一步说明，第n个路段的预期长度是

$$\int_0^1 dx\int_0^x dy(g_n(x)(e^y-1)+g_n(y))\left(2-\frac{1}{2}y^2\right)+\int_0^1 dx(1-x)g_n(x)e^x$$

〔注意：这个方程的“稳定状态”的解显得非常复杂；它的数值解已经为J.麦克纳（J. McKenna）得到，他说明这些路段长度趋于2.61307209的极限值，定理K不能应用于自然选择，所以$P=1$的情况对其它的P行不通。〕

26.〔$M33$〕把习题25中的算法当作$P'=1$时自然选择的定义，对任何$r\geqslant 0$，求当$P'=r$时头一个路段的预期长度如下

a）证明头一个路段长度为n的概率是

$$(n+r)\begin{bmatrix} n+r \\ n \end{bmatrix}\Big/(n+r+1)!$$

b）通过规则

$$\left[\!\left[\begin{matrix} 0 \\ m \end{matrix}\right]\!\right]=\delta_{m0},\quad \left[\!\left[\begin{matrix} n \\ m \end{matrix}\right]\!\right]=(n+m-1)\left(\left[\!\left[\begin{matrix} n-1 \\ m \end{matrix}\right]\!\right]+\left[\!\left[\begin{matrix} n-1 \\ m-1 \end{matrix}\right]\!\right]\right)\text{对 } n>0$$

定义“2阶斯特林数”$\left[\!\left[\begin{matrix} n \\ m \end{matrix}\right]\!\right]$，证明

$$\begin{bmatrix} n+r \\ n \end{bmatrix}=\sum_{0\leqslant k\leqslant r}\binom{n+r}{k+r}\left[\!\left[\begin{matrix} r \\ k \end{matrix}\right]\!\right]$$

c）证明头一个路段的平均长度因此是$c_r e-r-1$，其中

$$c_r = \sum_{0 \leqslant k \leqslant r} \left[\left[\begin{matrix} r \\ k \end{matrix}\right]\right](r+k+1)/(r+k)!$$

27.〔25〕 正文仅仅考虑了所有有待排序的记录都有一个固定大小的情况，对于可变长的记录，替代选择应如何做才好？

5.4.2 多阶段合并

现在我们已经看到可以如何构造初始路段，我们将考虑各种型式，这些型式可被用来把这些路段分布到带上，并把它们合并在一起直到仅剩下一个路段为止。

从假定有三条带 T1，T2 和 T3 可资利用开始；在 5.4 节开始处所描述的“平衡合并”技术，可以对 $P=2$ 和 $T=3$ 使用，这时它采取如下的形式：

B1. 交替地在 T1 带和 T2 带上分布路段。

B2. 把 T1 和 T2 的路段合并到 T3 上；然后如果 T3 仅包含一个路段时便停止。

B3. 把 T3 的路段交替地复写到 T1 和 T2 上，然后返回到 B2。

如果初始分布扫描产生 S 个路段，则头一个合并扫描将在 T3 上产生 $\lceil S/2 \rceil$ 个路段，第二个合并扫描将产生 $\lceil S/4 \rceil$ 个，等等。于是，如果说 $17 \leqslant S \leqslant 32$，则我们将有 1 趟分配扫描，5 趟合并扫描，以及 4 趟复写扫描。一般地说，若 $S>1$，对于所有数据的扫描次数是 $2\lceil \log_2 S \rceil$。

在此过程中的复写扫描是不需要的，因为它们并不减少路段数。如果我们使用一种两阶段的过程，则有一半的复写可以避免。

A1. 在带 T1 和 T2 上交替地分布初始路段。

A2. 把 T1 和 T2 上的路段合并到 T3 上；然后如果 T3 仅包含一个路段，则停止。

A3. 把 T3 上一半的路段复写到 T1 上。

A4. 把 T1 和 T3 的路段合并到 T2 上；然后如果 T2 仅含一个路段，则停止。

A5. 把 T2 上一半的路段复写到 T1，返回 A2。

对数据的扫描次数已经减少到 $\frac{3}{2}\lceil \log_2 S \rceil + \frac{1}{2}$，因为步骤 A3 和 A5 仅仅作“一次扫描的一半”；因此节省了大约25%的时间。

如果我们从 T1 上的 F_n 个路段和 T2 上的 F_{n-1} 个路段开始，这里 F_n 和 F_{n-1} 是连续的斐波那契数，则实际上可以完全消去复写。例如，考虑 $n=7$，$S=F_n+F_{n-1}=13+8=21$ 的情况：

	T1的内容	T2的内容	T3的内容	注释
阶段1.	1,1,1,1,1,1,1,1,1,1,1,1	1,1,1,1,1,1,1,1		初始分布
阶段2.	1,1,1,1,1	—	2,2,2,2,2,2,2,2	把 8 个路段合并到 T 3
阶段3.	—	3,3,3,3,3	2,2,2	把 5 个路段合并到 T 2
阶段4.	5,5,5	3,3	—	把 3 个路段合并到 T 1
阶段5.	5	—	8,8	把 2 个路段合并到 T 3
阶段6.	—	13	8	把 1 个路段合并到 T 2
阶段7.	21	—	—	把 1 个路段合并到 T 1

例如，若把每个初始路段的相对长度定为1，则“2，2，2，2，2，2，2，2”表示相对长度为2的8个路段，斐波那契数在这个图表中是无所不在的！

只有阶段1和7对数据进行了完全的扫描，阶段2仅仅处理初始路段的16/21；阶段3仅处理15/21，等等，因此如果我们假定初始路段都有近似相等的长度，则“扫描”总数就成为 $(21+16+15+15+16+13+21)/21=5\frac{4}{7}$。通过比较，以上的两阶段的过程将要求8次扫描以对21个初始的路段进行排序。我们将看到，一般地，这“斐波那契”型式近似地要求 $1.046\log_2 S+0.99$ 次扫描，使得它同一个四带平衡合并相匹敌，然而它只要求三条带。

同样的思想可以推广到T条带，对任何$T\geqslant 3$，使用（$T-1$）路合并。例如我们将看到，四条带的情形只要求对数据进行大约 $.703\log_2 S+0.96$ 次扫描。推广的形式涉及推广的斐波那契数。考虑下边六条带的例子：

	T1	T2	T3	T4	T5	T6	处理的初始路段
阶段1.	1^{31}	1^{30}	1^{28}	1^{24}	1^{16}	—	31+20+28+24+16=129
阶段2.	1^{15}	1^{14}	1^{12}	1^{8}	—	5^{16}	16×5=80
阶段3.	1^{7}	1^{6}	1^{4}	—	9^{8}	5^{8}	8×9=72
阶段4.	1^{3}	1^{2}	—	17^{4}	9^{4}	5^{4}	4×17=68
阶段5.	1^{1}	—	33^{2}	17^{2}	9^{2}	5^{2}	2×33=66
阶段6.	—	65^{1}	33^{1}	17^{1}	9^{1}	5^{1}	1×65=65
阶段7.	129^{1}	—	—	—	—	—	1×129=129

这里 1^{31} 代表相对长度为1的31个路段，等等；这里已经从头到尾使用了五路合并。这一般的型式是由 R. L. 吉尔斯塔德（R. L. Gilstad）提出的〔Proc. AFIPS Eastern Jt. Computer Conf. 18(1960)，143-148〕。他把它称作多阶段合并。三条带的情况已经较早地为 B. K. 贝兹（B. K. Betz）发现〔未发表的小册子，Minneapolis-Honeywell Regulator Co. (1956)〕。

为了如同上边的例子中那样进行多阶段的合并工作，我们在每个阶段之后，需要使诸路段在带上构成“完全的斐波那契分布”。通过由底向上读上面的表，可以看到当$T=6$时，头七个完全的斐波那契分布是{1，0，0，0，0}，{1，1，1，1，1}，{2，2，2，2，1}，{4，4，4，3，2}，{8，8，7，6，4}，{16，15，14，12，8}和{31，30，28，24，16}，我们面临的一些大问题是：

1. 这些完全的斐波那契分布所基于的规则是什么？
2. 如果S不对应于一个完全的斐波那契分布，则我们怎么办？
3. 我们应该怎样设计初始的分布扫描，使得它在带上产生所希望的配置？
4. 一个T带多阶段合并要求对数据进行多少“次”扫描？其中扫描次数是S（初始路段的个数）的一个函数。

我们将依次地讨论这四个问题，首先给出“容易的答案”而后进行更加仔细的分析。

完全的斐波那契分布可以通过周期地转动带的内容来向后运行这种型式而得到。例如，当$T=6$时，我们有下列的路段分布：

级	T1	T2	T3	T4	T5	总数	最后的输出将在
0	1	0	0	0	0	1	T1
1	1	1	1	1	1	5	T6
2	2	2	2	2	1	9	T5
3	4	4	4	3	2	17	T4
4	8	8	7	6	4	33	T3
5	16	15	14	12	8	65	T2
6	31	30	28	24	16	129	T1
7	61	59	55	47	31	253	T6
8	120	116	108	92	61	497	T5
……							
n	a_n	b_n	c_n	d_n	e_n	t_n	$T(k)$
$n+1$	a_n+b_n	a_n+c_n	a_n+d_n	a_n+e_n	a_n	t_n+4c_n	$T(k-1)$ (1)
……							

（在初始的分布之后，带 6 将总是空的。）

从级 n 到级 $n+1$ 进行的规则表明，在每一级中都有条件

$$a_n \geqslant b_n \geqslant c_n \geqslant d_n \geqslant e_n \tag{2}$$

成立。事实上，从（1）容易看出

$$\begin{aligned}
e_n &= a_{n-1} \\
d_n &= a_{n-1}+e_{n-1}=a_{n-1}+a_{n-2} \\
c_n &= a_{n-1}+d_{n-1}=a_{n-1}+a_{n-2}+a_{n-3} \\
b_n &= a_{n-1}+c_{n-1}=a_{n-1}+a_{n-2}+a_{n-3}+a_{n-4} \\
a_n &= a_{n-1}+b_{n-1}=a_{n-1}+a_{n-2}+a_{n-3}+a_{n-4}+a_{n-5}
\end{aligned} \tag{3}$$

其中 $a_0=1$，对于 $n=-1$，-2，-3，-4 我们令 $a_n=0$。

第 p 阶斐波那契数 $F_n^{(p)}$ 定义为

$$\begin{aligned}
&F_n^{(p)}=F_{n-1}^{(p)}+F_{n-2}^{(p)}+\cdots+F_{n-p}^{(p)} \quad 对于\ n \geqslant p \\
&F_n^{(p)}=0 \quad 对于\ 0 \leqslant n \leqslant p-2,\ F_{p-1}^{(p)}=1
\end{aligned} \tag{4}$$

换句话说，我们开始有 $p-1$ 个 0，然后是 1，然后每个数是前边 p 个值之和。当 $p=2$ 时，这是通常的斐波那契数列。对于更大的 p 值，这个序列似乎首先是由 V. 施莱格尔（V. Schlegel）在“El Progreso Mathematico” 4(1894)，173-174上加以研究的。施莱格尔导出了母函数

$$\sum_{n \geqslant 0} F_n^{(p)} z^n=\frac{z^{p-1}}{1-z-z^2-\cdots-z^p}=\frac{z^{p-1}-z^p}{1-2z+z^{p+1}} \tag{5}$$

等式（3）表明在一个六带多阶段合并期间，在T1上的路段数是第五阶斐波那契数 $a_n=F_{n+4}^{(5)}$。

一般地说，如果我们置 $P=T-1$，则 T 条带的多阶段合并分布将以同样方式对应于第 P 阶斐波那契数。对于 $1 \leqslant k \leqslant P$，第 k 条带在完全的第 n 级分布中得到

$$F_{n+p-2}^{(P)}+F_{n+p-3}^{(P)}+\cdots+F_{n+k-2}^{(P)}$$

个初始路段，因此在所有带上的初始路段总数是

$$t_n=PF_{n+p-2}^{(P)}+(P-1)F_{n+p-3}^{(P)}+\cdots+F_{n-1}^{(P)} \tag{6}$$

这样就解决了“完全的斐波那契分布”的问题。但如果对于任意 n，S 不恰巧等于 t_n，则我们应做什么呢？而且开头在这些带上我们怎样得到路段呢？

当 S 不是完全的（有少数值是这样的）时候，我们还可以象在平衡的 P 路合并中那样做，加上人工的“虚拟路段”后使得可以假想 S 仍然是完全的。有若干种方式来附加虚拟路段，我们尚不准备分析这样做的“最好”的方式。我们将首先讨论分布的方法和指定虚拟路段的方法，它并非严格地是最优的，尽管它有简便的长处而且似乎比所有其它同样简单的方法都要好些。

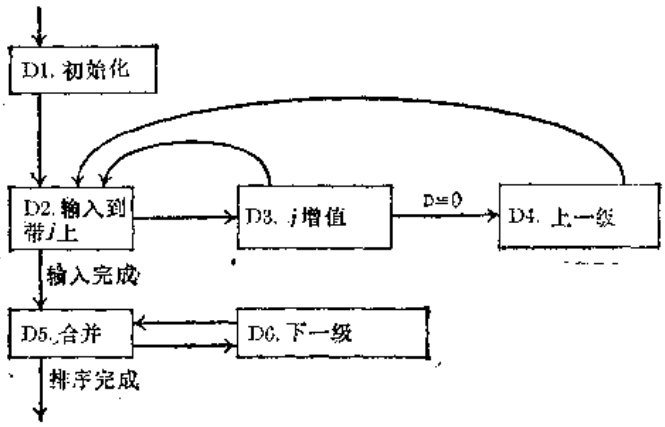


图68　多阶段的合并排序

算法D　（具有“水平”分布的多阶段合并排序）这个算法把初始路段疏散到带上，一次一个路段，直到初始路段的供给穷尽为止。然后它确定这些带如何被合并。假定有 $T=P+1\geqslant 3$ 台可利用的磁带机，同时使用 P 路合并。带 T 可以用来保存输入，因为它不接收任何初始的路段。建议如下一些表格：

$A[j]$，$1\leqslant j\leqslant T$：我们正在力求的完全斐波那契分布。

$D[j]$，$1\leqslant j\leqslant T$：假定应出现在设备编号为 j 的逻辑带开始处的虚拟路段个数。

TAPE$[j]$，$1\leqslant j\leqslant T$：对应于逻辑带设备号 j 的物理带设备号（处理“逻辑带设备号”是方便的。对它分配物理带设备号是随算法的进行而变的）。

D1.〔初始化〕　对于 $1\leqslant j\leqslant T$，置 $A[j]\leftarrow D[j]\leftarrow 1$ 和 TAPE$[j]\leftarrow j$。置 $A[T]\leftarrow D[T]\leftarrow 0$ 和 TAPE$[T]\leftarrow T$。然后置 $l\leftarrow 1$，$j\leftarrow 1$。

D2.〔输入到带 j〕　写一个路段到编号为 j 的带上，并且 $D[j]$ 减 1。然后如果输入已穷尽，则重绕所有的带并转到步骤 D5。

D3.〔j 增值〕　如果 $D[j]<D[j+1]$，则 j 增加 1，并且返回到 D2。否则如果 $D[j]=0$，则转到 D4。否则置 $j\leftarrow 1$，并且返回到 D2。

D4.〔升一级〕　置 $l\leftarrow l+1$，$a\leftarrow A[1]$，然后对于 $j=1$，2，…，P（在这个次序下）置 $D[j]\leftarrow a+A[j+1]-A[j]$ 和 $A[j]\leftarrow a+A[j+1]$。（见（1）并注意 $A[P+1]$ 总是 0。这时我们将有 $D[1]\geqslant D[2]\geqslant\cdots\geqslant D[T]$）。

现在置 $j\leftarrow 1$ 并且返回到 D2。

D5.〔合并〕　如果 $l=0$，则排序完成而且输出在 TAPE$[1]$上，否则，把 TAPE$[1]$，…，TAPE$[P]$的路段合并到 TAPE$[T]$上，直到 TAPE$[P]$ 成为空的而且 $D[P]=0$ 为止。对于每一个被合并的路段，合并过程应该如下操作：如果对所有 j，$1\leqslant j\leqslant P$，$D[j]>0$，则 $D[T]$ 增加 1 且对于 $1\leqslant j\leqslant P$，每个 $D[j]$ 减 1；否则从每一个使得 $D[j]=0$ 的 TAPE$[j]$合并一个路段，而且对每个其它的 j，$D[j]$ 减 1。（于是虚拟路段被想象为在带的开始而不是在末尾。）

D6.〔降一级〕 置 $l \leftarrow l-1$。重绕 TAPE〔P〕和 TAPE〔T〕(实际上 TAPE〔P〕之重绕在步骤 D5 中，在输入了它的最后一组路段之后就可以开始)。然后置 (TAPE〔1〕, TAPE〔2〕, …, TAPE〔T〕)←(TAPE〔T〕, TAPE〔1〕, …, TAPE〔$T-1$〕), (D〔1〕, D〔2〕, …, D〔T〕)←(D〔T〕, D〔1〕, …, D〔$T-1$〕), 并返回到步骤 D5。

在这个算法的步骤 D3 中如此简洁地叙述的分布规则，是打算尽可能在每条带上设置相等个数的虚拟路段。图69示出了当我们在一个六带的排序中从级 4 (33个路段) 进行到级 5 (65 个路段) 时分布的次序；如果仅仅有，比如说，53 个初始的路段，则所有编号为 54 和更高的路段都将被处理作虚拟的（这些路段实际上被写到这条带的末尾，但是最好把它们想象成写到开始处，因为已经假定虚拟路段都是在开始处)。

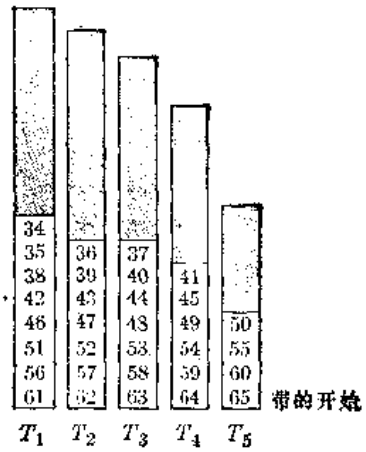


图69 当从级 4 进行到级 5 时，路段 34 到 65 被分布于诸带上的次序（见(1)的完全分布表)。阴影区域表示当到达级 4 时已被分布好的头 33 个路段

我们现在已经讨论了上边列出的头三个问题，剩下要考虑的是数据“扫描”的次数。把六带的例子同表（1）进行比较，我们看到，当 $S=t_6$ 时，处理的初始路段的总数是 $a_5t_1+a_4t_2+a_3t_3+a_2t_4+a_1t_5+a_0t_6$，初始的分布扫描不计在内。习题 4 导出母函数

$$a(z)=\sum_{n\geqslant 0} a_n z^n=\frac{1}{1-z-z^2-z^3-z^4-z^5}$$

$$t(z)=\sum_{n\geqslant 1} t_n z^n=\frac{5z+4z^2+3z^3+2z^4+z^5}{1-z-z^2-z^3-z^4-z^5} \qquad (7)$$

由此得出，一般说来当 $S=t_n$ 时处理的初始路段数准确地等于在 $a(z)\cdot t(z)$ 中 z^n 的系数，加上 t_n(对于初始的分布扫描)。这使得有可能如习题 5～7 所示来计算多阶段合并的渐近行为，并得到表 1 中所示的结果。

表 1 多阶段的合并排序的近似行为

带	阶段	扫描	扫描/阶段百分比	增长率
3	$2.078\ln S+0.672$	$1.504\ln S+0.992$	72	1.6180340
4	$1.641\ln S+0.364$	$1.015\ln S+0.965$	62	1.8392868
5	$1.524\ln S+0.078$	$0.863\ln S+0.921$	57	1.9275620
6	$1.479\ln S-0.185$	$0.795\ln S+0.864$	54	1.9659482
7	$1.460\ln S-0.424$	$0.762\ln S+0.797$	52	1.9835828
8	$1.451\ln S-0.642$	$0.744\ln S+0.723$	51	1.9919642
9	$1.447\ln S-0.838$	$0.734\ln S+0.646$	51	1.9960312
10	$1.445\ln S-1.017$	$0.728\ln S+0.568$	50	1.9980295
20	$1.443\ln S-2.170$	$0.721\ln S-0.030$	50	1.9999981

在表 1 中，“增长率”是 $\lim_{n\to\infty} t_{n+1}/t_n$，此即路段的个数在每一级增长的近似因子。“扫描”表示每个记录被处理的平均次数，即 $(1/S)$ 乘上在分布和合并阶段处理的初始路段的总数。对于完全分布来说，当 $S\to\infty$ 时，存在一个 $\varepsilon>0$，使得在每种情况下所述的

扫描和阶段数都正确到$O(S^{-1})$。

图70示出了当算法D用来处理不完全数的情况时，每个记录被合并的平均次数（作为S的一个函数）。注意，对于三条带，恰恰在完全分布之后就出现了相对低效率的“峰值”，但当有四条或更多的带时，这种现象就很少见了。使用八条或更多的带对六条或七条带的改进相当小。

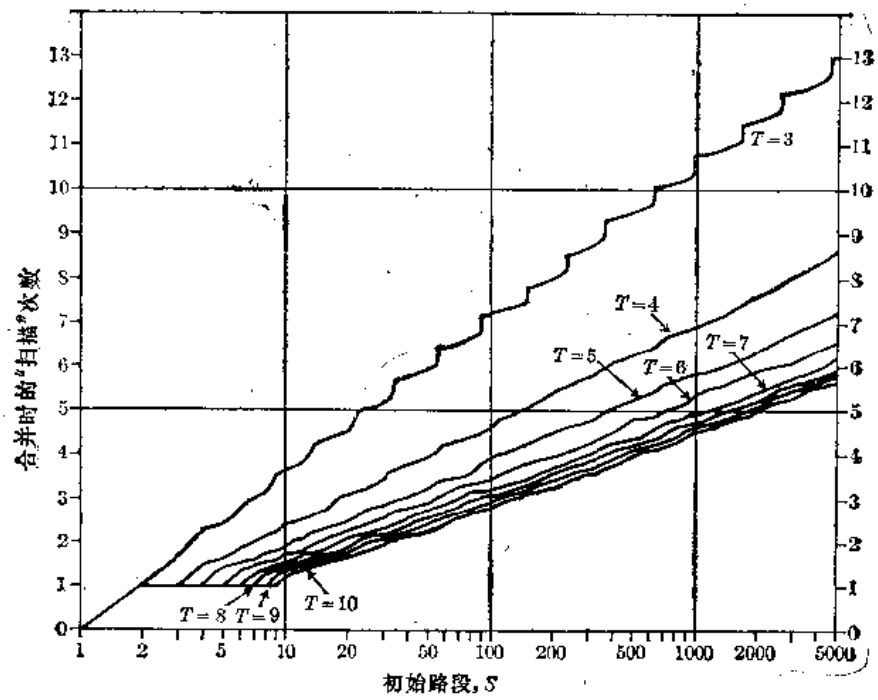


图70　使用算法D的多阶段合并的效率

***更仔细的考查**　在要求k次扫描的一个平衡的合并中，在排序过程中每个记录恰巧被处理k次。但是多阶段的过程却并非不偏不倚；某些记录可能得到比其它记录多许多次的处理，而且如果把虚拟路段安排到经常被处理的位置，则我们就能赢得速度。

因此让我们更仔细地研究多阶段的分布，不象在（1）中那样仅仅考查在每个带上的路段数，我们把每一路段同它的合并数，即在整个多阶段排序期间将被处理的次数，结合起来，就得到下列的表，以代替（1）：

级	T1	T2	T3	T4	T5	
0	0	—	—	—	—	
1	1	1	1	1	1	
2	21	21	21	21	2	
3	3221	3221	3221	322	32	
4	43323221	43323221	4332322	433232	4332	(8)
5	5443433243323221	544343324332322	54434332433232	544343324332	54434332	
·····	·····	·····	·····	·····	·····	
n	A_n	B_n	C_n	D_n	E_n	
$n+1$	$(A_n+1)B_n$	$(A_n+1)C_n$	$(A_n+1)D_n$	$(A_n+1)E_n$	A_n+1	
·····	·····	·····	·····	·····	·····	

如果我们从n级分布开始，A_n是a_n个值的串，表示T1上每个路段的合并数；B_n是对

于T2的对应的串，等等。记号“$(A_n+1)B_n$”意味着“A_n中的所有值增加1，后边紧接以B_n”。

图71(a)示出了出现在末端的A_5，B_5，C_5，D_5，E_5，示出每个路段的合并数如何出现在带上；注意，例如，在每条带开始的路段将被处理五次，而在T1末尾的路段将仅仅被处理一次。多阶段合并的这种区别对待的情况，使得把一个虚拟路段放置在带的开始比放在带的末尾要好得多。图71(b)示出了在五级多阶段合并中路段分布的一个最佳次序，把每个新路段放置到具有最小合并次数的位置上。注意，算法D(图69)并不是十分好的，因为在所有“3”的位置被用完之前，它填充了某些“4”的位置。

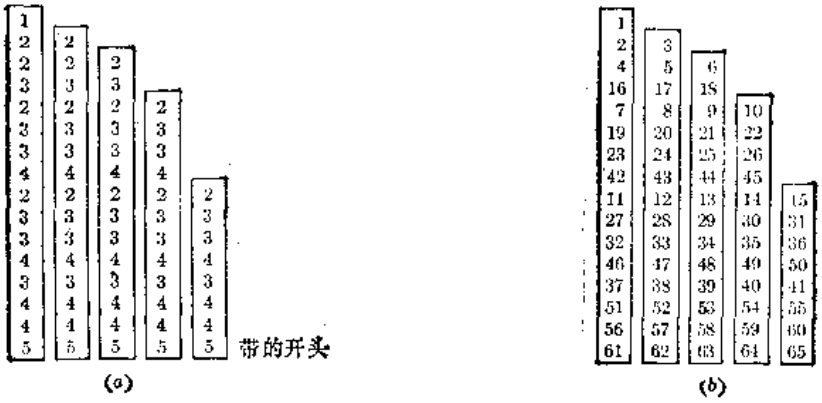


图71　对于六条带的第五级多阶段的分布的分析
(a) 合并数；(b) 最优分布的次序。

递归关系（8）表明，B_n，C_n，D_n，E_n的每一个都是A_n的初始子串。事实上，我们可以使用（8）来推导公式

$$\begin{aligned}
E_n&=(A_{n-1})+1\\
D_n&=(A_{n-1}A_{n-2})+1\\
C_n&=(A_{n-1}A_{n-2}A_{n-3})+1\\
B_n&=(A_{n-1}A_{n-2}A_{n-3}A_{n-4})+1\\
A_n&=(A_{n-1}A_{n-2}A_{n-3}A_{n-4}A_{n-5})+1
\end{aligned}\tag{9}$$

这推广了仅仅处理这些串的长度的公式（3）。进而，从定义诸A的规则可以看出，在每级的开头出现的实际上是相同的结构；我们有

$$A_n=n-Q_n\tag{10}$$

其中Q_n是由法则

$$Q_n=Q_{n-1}(Q_{n-2}+1)(Q_{n-3}+2)(Q_{n-4}+3)(Q_{n-5}+4)，\text{对于 } n\geqslant 1；$$
$$Q_0=\text{‘0’}，\ Q_n=(\text{空})\quad \text{对于 } n<0\tag{11}$$

定义的a_n个值的一个串。由于Q_n以Q_{n-1}开始，我们可以考察无穷的串Q_∞；它的头a_n个元素都等于Q_n，这个串Q_∞实际上表征了在多阶段分布中所有的合并数。在六条带的情况下。

$$\begin{aligned}
Q_\infty=&0112122312232334122323342334344 12\\
&2323342334344523343445344545122 32\cdots
\end{aligned}\tag{12}$$

习题11包含了对于这个串的一种有趣的解释。

设 A_n 是串 $m_1m_2\cdots m_{q_n}$，命 $A_n(x)=x^{m_1}+x^{m_2}+\cdots+x^{m_{q_n}}$ 是相应的计算每个合并数出现的次数的母函数；而且类似地定义 $B_n(x)$，$C_n(x)$，$D_n(x)$，$E_n(x)$。例如，$A_4(x)=x^4+x^3+x^3+x^2+x^2+x^2+x^2+x=x^4+3x^3+3x^2+x$。关系（9）告诉我们，对于 $n\geqslant 1$，有

$$
\begin{aligned}
E_n(x)&=x\,(A_{n-1}(x))\\
D_n(x)&=x\,(A_{n-1}(x)+A_{n-2}(x))\\
C_n(x)&=x\,(A_{n-1}(x)+A_{n-2}(x)+A_{n-3}(x))\\
B_n(x)&=x\,(A_{n-1}(x)+A_{n-2}(x)+A_{n-3}(x)+A_{n-4}(x))\\
A_n(x)&=x\,(A_{n-1}(x)+A_{n-2}(x)+A_{n-3}(x)+A_{n-4}(x)+A_{n-5}(x))
\end{aligned}
\tag{13}
$$

其中 $A_0(x)=1$，而且对于 $n=-1,\ -2,\ -3,\ -4$，$A_n(x)=0$。因之

$$
\sum_{n\geqslant 0}A_n(x)z^n=\frac{1}{1-x\,(z+z^2+z^3+z^4+z^5)}
$$

$$
=\sum_{k\geqslant 0}x^k(z+z^2+z^3+z^4+z^5)^k \tag{14}
$$

考虑所有带上的路段，我们设

$$
T_n(x)=A_n(x)+B_n(x)+C_n(x)+D_n(x)+E_n(x),\quad n\geqslant 1 \tag{15}
$$

由（13）我们立即有

$$
T_n(x)=5A_{n-1}(x)+4A_{n-2}(x)+3A_{n-3}(x)+2A_{n-4}(x)+A_{n-5}(x)
$$

因此

$$
\sum_{n\geqslant 1}T_n(x)z^n=\frac{x\,(5z+4z^2+3z^3+2z^4+z^5)}{1-x\,(z+z^2+z^3+z^4+z^5)} \tag{16}
$$

（16）的形式说明，容易计算出 $T_n(x)$ 的系数：

	z	z^2	z^3	z^4	z^5	z^6	z^7	z^8	z^9	z^{10}	z^{11}	z^{12}	z^{13}	z^{14}
x	5	4	3	2	1	0	0	0	0	0	0	0	0	0
x^2	0	5	9	12	14	15	10	6	3	1	0	0	0	0
x^3	0	0	5	14	26	40	55	60	57	48	35	20	10	4
x^4	0	0	0	5	19	45	85	140	195	238	260	255	220	170
x^5	0	0	0	0	5	24	69	154	294	484	703	918	1088	1168

(17)

这个表的诸列给出了 $T_n(x)$；例如，$T_4(x)=2x+12x^2+14x^3+5x^4$。在头一行之后，这个表中的每个项目都是位于前边一行中对应项目左边的五个项目之和。

在一个“完全的”第 n 级分布中，路段的数目是 $T_n(1)$，而且当这些路段被合并时处理的总数是微商 $T'_n(1)$，现在

$$
\sum_{n\geqslant 1}T'_n(x)z^n=\frac{5z+4z^2+3z^3+2z^4+z^5}{(1-x\,(z+z^2+z^3+z^4+z^5))^2} \tag{18}
$$

在（16）和（18）式中置 $x=1$，就可得出一个同我们早先的结论相一致的结果，这个结论就是：对于一个完全的第 n 次分布，合并的处理是 $a(z)\,t(z)$ 中 z^n 的系数〔参考（7）〕。

我们可以使用函数 $T_n(x)$ 来确定，当以最优方式加上虚拟路段时所涉及的工作量有多少。命 $\Sigma_n(m)$ 是在第 n 级分布中最小的 m 个合并数之和。通过考查（17）的诸列，很容易计算出这些值来，而且我们发现 $\Sigma_n(m)=$

	$m=1$	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
$n=1$	1	2	3	4	5	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n=2$	1	2	3	4	6	8	10	12	14	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
$n=3$	1	2	3	5	7	9	11	13	15	17	19	21	24	27	30	33	36	∞	∞	∞	∞
$n=4$	1	2	4	6	8	10	12	14	16	18	20	22	24	26	29	32	35	38	41	44	47
$n=5$	1	3	5	7	9	11	13	15	17	19	21	23	25	27	29	32	35	38	41	44	47
$n=6$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	33	36	39	42	45	48
$n=7$	2	4	6	8	10	12	14	16	18	20	23	26	29	32	35	38	41	44	47	50	53

(19)

例如，如果我们希望用一个第 3 级分布来对17个路段排序，则处理的总数量是 $\Sigma_3(17)=36$；但是如果我们使用第 4 级或第 5 级分布和虚拟路段的最优位置，则在合并阶段的处理总数量就只有 $\Sigma_4(17)=\Sigma_5(17)=35$。使用第 4 级是更好的，尽管17对应于一个“完全的”第 3 级分布！其实，当 S 变得很大时，已经证实级的最优个数要比在算法 D 中所使用的多得多。

习题14表明有一个非减的数序列 M_n，使得级 n 对于 $M_n\leqslant S<M_{n+1}$ 是“最优”的，但对于 $S\geqslant M_{n+1}$ 则不然。在六条带的情况下，我们刚刚计算的 $\Sigma_n(m)$ 的表说明

$$M_0=0,\ M_1=2,\ M_2=6,\ M_3=10,\ M_4=14$$

上边的讨论仅仅处理了六条带的情况，但是显然，同样的思想可应用于对于任何 $T\geqslant 3$ 的 T 条带的多阶段的合并；在所有适当的位置中，我们都简单地以 $P=T-1$ 代替 5。表 2 示出了对于各种 T 值所得到的序列 M_n。表 3 和图72指出了在作成虚拟路段的一个最优

表 2　对应于最优级的路段数

线路	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
1	2	2	2	2	2	2	2	2	M_1
2	3	4	5	6	7	8	9	10	M_2
3	4	6	8	10	12	14	16	18	M_3
4	6	10	14	14	17	20	23	26	M_4
5	9	18	23	29	20	24	28	32	M_5
6	14	32	35	43	53	27	32	37	M_6
7	22	55	76	61	73	88	35	41	M_7
8	35	96	109	154	98	115	136	44	M_8
9	56	173	244	216	283	148	171	199	M_9
10	90	280	359	269	386	168	213	243	M_{10}
11	145	535	456	779	481	640	240	295	M_{11}
12	234	820	1197	1034	555	792	1002	330	M_{12}
13	378	1635	1563	1249	1993	922	1228	1499	M_{13}
14	611	2401	4034	3910	2486	1017	1432	1818	M_{14}
15	988	4959	5379	4970	2901	4397	1598	2116	M_{15}
16	1598	7029	6456	5841	10578	5251	1713	2374	M_{16}
17	2574	14953	18561	19409	13097	5979	8683	2576	M_{17}
18	3955	20583	22876	23918	15335	6499	10069	2709	M_{18}
19	6528	44899	64189	27557	17029	30164	11259	15787	M_{19}

表 3　在最优的多阶段合并期间处理的初始路段

S	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
10	36	24	19	17	15	14	13	12	
20	90	60	49	44	38	36	34	33	
50	294	194	158	135	128	121	113	104	
100	702	454	362	325	285	271	263	254	
1000	10371	6680	5430	4672	4347	3872	3739	3632	
5000	63578	41286	32905	28620	26426	23880	23114	22073	
S	(1.51	0.951	0.761	0.656	0.589	0.548	0.539	0.488)	$\cdot S\ln S$
	+(− .11	+ .14	+ .16	+ .19	+ .21	+ .20	+ .02	+ .18)	$\cdot S$

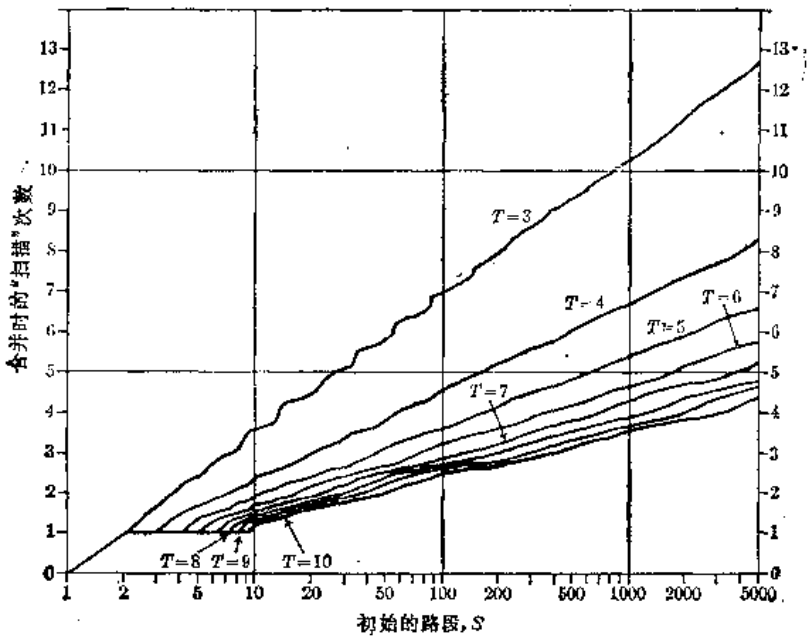


图72　具有最优初始分布的多阶段合并的效率（参考图70）

分布之后，被处理的初始路段总数。（在表 3 的下部出现的公式应当有保留地采用，因为它们是在变程 $1\leqslant S\leqslant 5000$ 上"最小平方符合"的（对于 $T=3$ 为 $1\leqslant S\leqslant 10000$）；这导致了有些反常的行为，因为给定的 S 的范围并非对于所有的 T 都是同样有利的。当 $S\to\infty$ 时，在一个最优的多阶段分布之后处理的初始路段数渐近于 $S\log_T S$，但是收敛于此渐近极限值的过程是极端缓慢的。）

表 4 表明算法 D 的分布方法堪与表 3 的最优分布结果相匹敌。显然，当 S 和 T 变得很大时，算法 D 并不是非常接近最优的；但是还不清楚在这两种情况下如何能比算法 D 做得更好而又不致搞得过分复杂，特别是，如果我们事先并不知道 S 的情况时就更是如此。幸而我们很少碰上很大的 S（见5.4.6节），所以算法 D 实际上并不是太坏的；事实上，它相当好。

多阶段排序首先是由 W. C. 卡特（W. C. Carter）从数学上加以分析的〔*Proc. IFIP Congress* (1962), 62-66〕。关于最优虚拟路段安排的以上许多结果，原来是由 B. 萨克曼（B. Sackman）和 T. 辛格（T. Singer）给出的〔A vector model for merge sort analysis)，在ACM Sort Symposium (November, 1962) 上宣读的未发表的论文，21pp.〕。

表 4　在标准多阶段合并时处理的初始路段

S	$T=3$	$T=4$	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$
10	36	24	19	17	15	14	13	12
20	90	62	49	44	41	37	34	33
50	294	194	167	143	134	131	120	114
100	714	459	393	339	319	312	292	277
1000	10730	6920	5774	5370	4913	4716	4597	4552
5000	64740	43210	36497	32781	31442	29533	28817	28080

萨克曼后来提出在算法D中使用的分布的水平方法。唐纳德·谢尔〔CACM 14（1971），713-719；15（1972），28〕独立地发展了这个理论，指出了关系（10），并对于若干不同的分布算法进行了详细的研究。德里克·A. 扎夫（Derek A. Zave）〔SIAM J. Computing 6(1977)，1-39〕已经作出了进一步的有教益的发展和改进，母函数（16）首先是由W. 伯克（W. Burge）研究的〔*Proc. IFIP Congress*（1971），待出〕。

重绕时间为何？　迄今，我们都以"被处理的初始路段"作为比较带合并策略的有效性的唯一测度。但这节开始的例子中在阶段2到6的每一阶段之后，计算机都必须等候两条带重绕；在下一个阶段可以进行之前，先前的输出带和新的当前输出带两者都必须重新定位到开头处。这会引起一个相当大的迟延，因为先前的输出带一般都包含了相当百分比的正被排序的记录（见表1的"扫描/阶段"列）。在所有这些重绕操作期间，让计算机闲得无聊是令人遗憾的，但如果我们使用一种不同的合并形式，那就可以用其它带来完成有用的工作。

简单地修改多阶段的过程，就能解决这个问题，尽管它至少要用五条带〔见Y. Cesari, thesis, U. of Paris (1968)，25-27，其中把这个思想归功于J. 卡伦（J. Caron）〕，在卡伦的方案里，每当那两条带重绕时，都把$T-3$条带上的路段合并到另一条带上。

例如，考虑六条带和49个初始路段的情况。在下表中，R表示在这个阶段进行重绕，而且假定75包含原来的输入：

阶段	T1	T2	T3	T4	T5	T6	写的时间	重绕时间
1	1^{11}	1^{17}	1^{13}	1^{8}	—	(R)	49	17
2	(R)	1^{9}	1^{5}	—	R	3^{8}	$8\times3=24$	$49-17=32$
3	1^{6}	1^{4}	—	R	3^{5}	R	$5\times3=15$	max (8, 24)
4	1^{2}	—	R	5^{4}	R	3^{4}	$4\times5=20$	max (13, 15)
5	—	R	7^{2}	R	3^{3}	3^{2}	$2\times7=14$	max (17, 20)
6	R	11^{2}	R	5^{2}	3^{1}	—	$2\times11=22$	max (11, 14)
7	15^{1}	R	7^{1}	5^{1}	—	R	$1\times15=15$	max (22, 24)
8	R	11^{1}	7^{0}	—	R	23^{1}	$1\times23=23$	max (15, 15)
9	15^{1}	11^{1}	—	R	33^{0}	R	$0\times33=0$	max (20, 23)
10	(15^{0})	—	R	49^{1}	(R)	(23^{0})	$1\times49=49$	14

这里所有的重绕时间实际上都是重叠的，只有阶段9（这是一个"虚拟阶段"，它为最后的合并作准备）以及在初始分布阶段之后（当所有带都被重绕时）除外。如果t是合并一个初始路段中全体记录的时间，且如果r是重绕一个初始路段的时间，则这个过程花费大约$182t+40r$加上初始分布和最后重绕花费的时间。对于使用算法D的标准多阶段法，对

应数字是$140t+104r$，当$r=\frac{3}{4}t$时它要差一些，而当$r=\frac{1}{2}t$时则稍微好些。

我们对于标准多阶段法已说过的每件事情，都可适用于卡伦的多阶段法；例如，序列a_n现在满足的不是（3）而是递归式

$$a_n=a_{n-2}+a_{n-3}+a_{n-4} \tag{20}$$

读者将发现，象我们分析标准的多阶段法那样来分析这个方法是有教益的，因为它将增进对于这两种方法的理解（比如，见习题19和20）。

表5给出了关于卡伦多阶段法的一些事实，它们类似于表1中的通常多阶段法的相应事实。注意，卡伦的方法就处理的路段数以及重绕的时间而言，实际上比对八条或更多带的多阶段法优越，尽管它进行了$T-3$路合并而不是$T-1$路合并！

表5　卡伦多阶段合并排序的近似行为

带	阶　段	扫　描	扫描/阶段百分比	增 长 率
5	$3.556\ln S+0.158$	$1.463\ln S+1.016$	41	1.3247180
6	$2.616\ln S-0.166$	$0.951\ln S+1.014$	36	1.4655712
7	$2.337\ln S-0.472$	$0.781\ln S+1.001$	33	1.5341577
8	$2.216\ln S-0.762$	$0.699\ln S+0.980$	32	1.5701473
9	$2.156\ln S-1.034$	$0.654\ln S+0.954$	30	1.5900094
10	$2.124\ln S-1.290$	$0.626\ln S+0.922$	29	1.6013473
20	$2.078\ln S-3.093$	$0.575\ln S+0.524$	28	1.6179086

一个高阶的合并并不必然地意味着一个有效的排序，在我们认识到这一点之前，上述事实似乎是一种怪事。作为一个极端的例子，考虑把一个路段放到T1上而且把n个路段放到T2，T3，T4，T5上；如果我们交替地进行五路合并到T6和T1上，直到T2，T3，T4，T5变空为止，则处理时间是$(2n^2+3n)$个初始路段长度，实际上和S^2而不是和$S\log_2 S$成比例，尽管自始至终地进行了五路合并。

带的分开　重绕时间的有效重叠是在许多应用中产生的一个问题，而不只限于排序中，而且有一个通常可资利用的一般的方法。考虑一个迭代过程，它以如下方式使用两条带：

	T1	T2
阶段1	输出1	—
	重绕	—
阶段2	输入1	输出2
	重绕	重绕
阶段3	输出3	输入2
	重绕	重绕
阶段4	输入3	输出4
	重绕	重绕

等等，其中“输出k”意味着写出第k个输出文件，而“输入k”就意味着读入它。当使

用三条带时，如同C．韦塞特（C. Weisert）所建议的〔CACM5（1962），102〕那样，可以免去重绕的时间。

	T1	T2	T3
阶段1	输出1.1	—	—
	输出1.2	—	—
	重绕	输出1.3	—
阶段2	输入1.1	输出2.1	—
	输入1.2	重绕	输出2.2
	重绕	输入1.3	输出2.3
阶段3	输出3.1	输入2.1	重绕
	输出3.2	重绕	输入2.2
	重绕	输出3.3	输入2.3
阶段4	输入3.1	输出4.1	重绕
	输入3.2	重绕	输出4.2
	重绕	输入3.3	输出4.3

等等，这里“输出$k.j$”意味着，写出第k个输出文件的第j个三分之一，而“输入$k.j$”就意味着读入它。实际上，如果重绕速度至少是读写速度的两倍，则所有的重绕时间都将被消去。这样一个过程，其中每个阶段的输出都被分到一些带上，称为“带的分开”。

R．L．麦卡勒斯特（R. L. McAllester）〔*CACM* 7（1964），158-159〕已经证明，带的分开导致一个把多阶段合并中的重绕时间加以重叠的有效方法。他的方法可以用于四条或更多的带，而且它进行$T-2$路合并。

再次假定有六条带，让我们来设计一个合并型式，它的操作如下：分开在每级上的输出，其中“I”，“O”，和“R”分别表示输入、输出和重绕。

级	T1	T2	T3	T4	T5	T6	输出路段的个数
7	I	I	I	I	R	O	u_7
	I	I	I	I	O	R	v_7
6	I	I	I	R	O	I	u_6
	I	I	I	O	R	I	v_6
5	I	I	R	O	I	I	u_5
	I	I	O	R	I	I	v_5
4	I	R	O	I	I	I	u_4
	I	O	R	I	I	I	v_4
3	R	O	I	I	I	I	u_3
	O	R	I	I	I	I	v_3
2	O	I	I	I	I	R	u_2
	R	I	I	I	I	O	v_2
1	I	I	I	I	R	O	u_1
	I	I	I	I	O	R	v_1
0	I	I	I	R	O	I	u_0
	I	I	I	O	R	I	v_0

(21)

为了以T4上有一个路段和所有其它的带都变空告终，我们需要有

$$
\begin{aligned}
&v_0=1\\
&u_0+v_1=0\\
&u_1+v_2=u_0+v_0\\
&u_2+v_3=u_1+v_1+u_0+v_0\\
&u_3+v_4=u_2+v_2+u_1+v_1+u_0+v_0\\
&u_4+v_5=u_3+v_3+u_2+v_2+u_1+v_1+u_0+v_0\\
&u_5+v_6=u_4+v_4+u_3+v_3+u_2+v_2+u_1+v_1
\end{aligned}
$$

等等；一般说来，要求对于所有的 $n\geqslant 0$

$$u_n+v_{n+1}=u_{n-1}+v_{n-1}+u_{n-2}+v_{n-2}+u_{n-3}+v_{n-3}+u_{n-4}+v_{n-4} \tag{22}$$

这里我们认为，对所有 $j<0$　　$u_j=v_j=0$。

这些方程没有唯一的解；其实，如果令所有的 u 都为 0，则我们就浪费了一条带而得到了通常的多阶段合并！但是如果我们选择 $u_n\approx v_{n+1}$，则重绕时间将令人满意地重叠。

麦卡勒斯特建议取 $u_n=v_{n-1}+v_{n-2}+v_{n-3}+v_{n-4}$, $v_{n-1}=u_{n-1}+u_{n-2}+u_{n-3}+u_{n-4}$，使得序列

$$\langle x_0,\ x_1,\ x_2,\ x_3,\ x_4,\ x_5,\ \cdots\rangle=\langle v_0,\ u_0,\ v_1,\ u_1,\ v_2,\ u_2,\ \cdots\rangle$$

满足一致递归式 $x_n=x_{n-3}+x_{n-5}+x_{n-7}+x_{n-8}$。然而可以证明，若命

$$
\begin{aligned}
v_{n+1}&=u_{n-1}+v_{n-1}+u_{n-2}+v_{n-2}\\
u_n&=u_{n-3}+v_{n-3}+u_{n-4}+v_{n-4}
\end{aligned} \tag{23}
$$

则更好，这个序列不仅在它的合并时间方面稍好些，而且它还有很大的长处，即对应的合并时间可从数学上加以分析。〔麦卡勒斯特的选择是极其难以分析的，因为不同长度的路段可以在同一个阶段中出现；我们将看到这对于（23）则不然。〕

在（21）的型式中按反方向进行，我们可以导出每级上每条带的路段数，并得到下列的排序方案：

级	T1	T2	T3	T4	T5	T6	写的时间	重绕时间
	1^{23}	1^{21}	1^{17}	1^{10}	—	1^{11}	82	23
7	1^{19}	1^{17}	1^{13}	1^{6}	R	$1^{11}4^{4}$	$4\times 4=16$	$82-23$
	1^{13}	1^{11}	1^{7}	—	4^{6}	R	$6\times 4=24$	25
6	1^{10}	1^{8}	1^{4}	R	4^{9}	$1^{8}4^{4}$	$3\times 4=12$	10
	1^{6}	1^{4}	—	4^{4}	R	$1^{4}4^{4}$	$4\times 4=16$	36
5	1^{5}	1^{3}	R	$4^{4}7^{1}$	4^{8}	$1^{3}4^{4}$	$1\times 7=7$	17
	1^{2}	—	7^{3}	R	4^{5}	4^{4}	$3\times 7=21$	23
4	1^{1}	R	$7^{3}13^{1}$	$4^{3}7^{1}$	4^{4}	4^{3}	$1\times 13=13$	21
	—	13^{1}	R	$4^{2}7^{1}$	4^{3}	4^{2}	$1\times 13=13$	34
3	R	$13^{1}19^{1}$	$7^{2}13^{1}$	$4^{1}7^{1}$	4^{2}	4^{1}	$1\times 19=19$	23
	19^{1}	R	$7^{1}13^{1}$	7^{1}	4^{1}	—	$1\times 19=19$	32
2	$19^{1}31^{0}$	$13^{1}19^{1}$	$7^{1}13^{1}$	7^{1}	4^{1}	R	$0\times 31=0$	25
	R	19^{1}	13^{1}	7^{0}	—	31^{1}	$1\times 31=31$	19

1	$19^1 31^0$	19^1	13^1	7^0	R	$31^1 52^0$	$0\times 52=0$	
	$19^1 31^0$	19^1	13^1	—	52^0	R	$0\times 52=0$	max (36,31,23)
0	$19^1 31^0$	19^1	13^1	R	$52^0 82^0$	$31^1 52^0$	$0\times 82=0$	
	(31^0)	(19^0)	—	82^1	(R)	(52^0)	$1\times 82=82$	0

在输入带T5被重绕期间（82个单位），在第2级的上半阶段（25个单位），以及在第1级和第0级的最后的“虚拟合并”阶段（36个单位），出现非重叠的重绕。所以我们可以估算这个时间是$273t+143r$，算法D相应的量是$268t+208r$，几乎总要低些。

不难看出（参考习题23），在每个阶段，路段的输出长度逐次为

$$4,\ 4,\ 7,\ 13,\ 19,\ 31,\ 52,\ 82,\ 133,\ \cdots \tag{24}$$

这是一个满足规则

$$t_n=t_{n-2}+2t_{n-3}+t_{n-4} \tag{25}$$

的序列，如果认为当$n\leqslant 0$时$t_n=1$，通过象我们对于标准的多阶段法〔参考（8）〕所做的那样，来考察合并数的串，我们也可以分析虚拟路段的最优设置：

级	T1	T2	T3	T4	T6	最后输出在
1	1	1	1	1	—	T5
2	1	1	1	—	1	T4
3	21	21	2	2	1	T3
4	2221	222	222	22	2	T2
5	23222	23222	2322	23	222	T1
6	333323222	33332322	333323	3333	2322	T6
·····	·····	·····	·····	·····	·····	·····
n	A_n	B_n	C_n	D_n	E_n	T(k)
$n+1$	$(A''_nE_n+1)B_n$	$(A''_nE_n+1)C_n$	$(A''_nE_n+1)D_n$	A''_nE_n+1	A'_n	T($k-1$)
·····	·····	·····	·····	·····	·····	·····

(26)

其中$A_n=A'_nA''_n$，且A''_n由A_n的最后u_n个合并数组成。上述从n级进行到$n+1$级的规则对于满足（22）的**任何**方案都是正确的。当我们用（23）来定义诸u和诸v时，可以下列颇简单的方式来表达串A_n，…，E_n〔参考（9）〕：

$$\begin{aligned}
A_n&=(W_{n-1}W_{n-2}W_{n-3}W_{n-4})+1\\
B_n&=(W_{n-1}W_{n-2}W_{n-3})+1\\
C_n&=(W_{n-1}W_{n-2})+1\\
D_n&=(W_{n-1})+1\\
E_n&=(W_{n-2}W_{n-3})+1
\end{aligned} \tag{27}$$

其中

$$\begin{aligned}
&W_n=(W_{n-3}W_{n-4}W_{n-2}W_{n-3})+1 && \text{对于 } n>0\\
&W_0=\text{‘0’和}W_n=(\text{空}) && \text{对于 } n<0
\end{aligned} \tag{28}$$

从这些关系，容易对六条带的情况进行详细的分析。

一般说来，当有$T\geqslant 5$条带时，我们令$P=T-2$，而且通过规则

$$\begin{aligned}
v_{n+1}&=u_{n-1}+v_{n-1}+\cdots+u_{n-r}+v_{n-r}\\
u_n&=u_{n-r-1}+v_{n-r-1}+\cdots+v_{n-P}+v_{n-P} \qquad \text{对于 } n\geqslant 0
\end{aligned} \tag{29}$$

定义序列$\langle u_n\rangle$，$\langle v_n\rangle$，其中$r=\lfloor P/2\rfloor$；$v_0=1$且对$n<0$，$u_n=v_n=0$。如果$w_n=u_n+v_n$，则我们有

$$w_n=w_{n-2}+\cdots+w_{n-r}+2w_{n-r-1}+w_{n-r-2}+\cdots+w_{n-P},\quad n>0 \tag{30}$$

$w_0=1$；且当$n<0$时$w_n=0$。$n+1$级的带上的初始分布是：置$w_n+w_{n-r}+\cdots+w_{n-P+k}$个路段到带$k$上，$1\leqslant k\leqslant P$，以及置$w_{n-1}+\cdots+w_{n-r}$个路段到带$T$上；带$T-1$用于输入。然后当带$T-1$正被重绕时，$u_n$个路段被合并到$T$上，在$T$正重绕时，$v_n$个路段被合并到$T-1$上；当$T-2$正重绕时，$u_{n-1}$个路段合并到$T-1$上，等等。

当S不太小时，表6示出了这一过程的近似行为。"扫描/阶段"列近似地指出在一个阶段的每一半期间，整个文件中有多少正在被重绕，以及在每个完全的阶段，这文件近似地有多少正在被写出。对于六条或更多的带，带分开方法是比标准多阶段法要好的，而且或许对于五条带也是这样，至少对于很大的S是这样。

当$T=4$时，上述过程实际上就等价于平衡的两路合并，没有重叠重绕的时间，因为对所有的n，w_{2n+1}都将成为0。所以略加修改，置$v_2=0$，$u_1=0$，$v_1=0$，$u_0=0$，$v_0=1$，和对于$n\geqslant 2$，$v_{n+1}=u_{n-1}+v_{n-1}$，$u_n=u_{n-2}+v_{n-2}$，即得到$T=4$的表6的诸项。这导出了一个非常有趣的方案（见习题25和26）。

表6　用带分开的多阶段合并的近似行为

带	阶　段	扫　描	扫描/阶段百分比	增长率
4	$2.855\ln S+0.000$	$1.443\ln S+1.000$	50	1.4142136
5	$2.078\ln S+0.232$	$0.929\ln S+1.002$	45	1.6180340
6	$2.078\ln S-0.170$	$0.752\ln S+1.024$	36	1.6180340
7	$1.958\ln S-0.408$	$0.670\ln S+1.007$	34	1.6663019
8	$2.008\ln S-0.762$	$0.624\ln S+0.994$	31	1.6454116
9	$1.972\ln S-0.987$	$0.595\ln S+0.967$	30	1.6604077
10	$2.013\ln S-1.300$	$0.580\ln S+0.941$	29	1.6433803
20	$2.069\ln S-3.164$	$0.566\ln S+0.536$	27	1.6214947

习题

1.〔16〕 图69示出了路段34到65通过算法D被分布到五条带上的次序，问路段1到33以什么次序分布？

▶2.〔21〕 真或假：在算法D中的两个合并阶段之后，即在我们第二次达到步骤D6时，所有的虚拟路段都已经消失。

▶3.〔22〕 证明在步骤D4的结果处，条件D〔1〕≥D〔2〕≥…≥D〔T〕总是满足的。说明为什么在下述意义之下，这个条件是重要的：即否则步骤D2和D3就不能有效工作。

4.〔$M20$〕 导出母函数（7）。

5.〔$HM26$〕〔小E·P·迈尔斯（E. P. Miles）1960〕 对于所有$p\geqslant 2$，证明多项式$f_p(z)=z^p-z^{p-1}-\cdots-z-1$有$p$个不同的根，它们当中恰有一个绝对值大于1。〔提示：考虑多项式$z^{p+1}-2z^p+1$。〕

6.〔$HM24$〕 本题的目的是考虑如何来编制表1、5和6。假定我们有一个合并型式，其性质以下列方式由多项式$p(z)$和$q(z)$表征：（1）出现在要求n个合并阶段

的一个"完全分布"中的初始路段个数是 $p(z)/q(z)$ 中 z^n 的系数。(2)在这 n 个合并阶段里处理的初始路段个数是 $p(z)/q(z)^2$ 中 z^n 的系数。(3)有 $q(z^{-1})$ 的一个"最大"的根 α，使得 $q(\alpha^{-1})=0$, $q'(\alpha^{-1})\neq 0$, $p(\alpha^{-1})\neq 0$，而且 $q(\beta^{-1})=0$ 蕴涵着 $\beta=\alpha$ 或 $|\beta|<|\alpha|$。

证明存在一个数 $\varepsilon>0$，使得如果 S 是在一个要求 n 个合并阶段的完全分布中的路段数，而且如果在这些阶段中处理了 ρS 个初始路段，则我们有 $n=a\ln S+b+O(S^{-\varepsilon})$, $\rho=c\ln S+d+O(S^{-\varepsilon})$，其中

$$a=(\ln\alpha)^{-1},\quad b=-a\ln\left(\frac{p(\alpha^{-1})}{-q'(\alpha^{-1})}\right)-1\quad c=a\frac{\alpha}{-q'(\alpha^{-1})},$$

$$d=\frac{(b+1)\alpha-p'(\alpha^{-1})/p(\alpha^{-1})+q''(\alpha^{-1})/q'(\alpha^{-1})}{-q'(\alpha^{-1})}$$

7. 〔$HM22$〕 设 α_p 是习题5中多项式 $f_p(z)$ 的最大的根，问当 $p\to\infty$ 时，α_p 的渐近行为是什么？

8. 〔$M20$〕 E·内特托（E. Netto, 1901） 设 $N_m^{(p)}$ 是把 m 表达成整数 $\{1, 2, \cdots, p\}$ 的一个有序和的方式数。例如，当 $p=3$ 和 $m=5$ 时，有13种方式，即 $1+1+1+1+1=1+1+1+2=1+1+2+1=1+1+3=1+2+1+1=1+2+2=1+3+1=2+1+1+1=2+1+2=2+2+1=2+3=3+1+1=3+2$。证明 $N_m^{(p)}$ 是一个广义裴波那契数。

9. 〔$M20$〕 设 $K_m^{(p)}$ 是 m 个0和1的这样的序列的个数，即其中没有 p 个相邻的1出现。例如，当 $p=3$ 和 $m=5$ 时，有24种方式：00000, 00001, 00010, 00011, 00100, 00101, 00110, 01000, 01001, …, 11011。证明 $K_m^{(p)}$ 是一个广义裴波那契数。

10. 〔$M27$〕 （广义裴波那契数系统） 证明每一个非负整数 n 都可以唯一地表示为不同的 p 阶裴波那契数 $F_j^{(p)}$ 之和，$j\geqslant p$，而且这个表示不使用连续的 p 个裴波那契数。

11. 〔$M24$〕 证明（12）中的串 Q_∞ 的第 n 个元素等于用第五阶裴波那契数表示 $n-1$ 时其中不同裴波那契数的个数（参考习题10）。

12. 〔$M20$〕 试求矩阵

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

的乘方与（1）中的完全裴波那契分布之间的联系。

▶13. 〔22〕 证明完全裴波那契分布的下列相当奇特的性质：当最后的输出将在 T 号带上时，则在每个其它带上的路段数都是奇数；当最后的输出将是在不同于 T 的某条带上时，则在该带上的路段数将是奇的，而它在其它带上将是偶的〔参考（1）〕。

14. 〔$M35$〕 设 $T_n(x)=\sum_{k\geqslant 0}T_{nk}x^k$，其中 $T_n(x)$ 是在(16)中定义的多项式。(a)证明对于每个 k，都有一个数 $n(k)$，使得 $T_{1k}\leqslant T_{2k}\leqslant\cdots\leqslant T_{n(k)k}>T_{n(k)+1,k}\geqslant\cdots$。(b) 若 $T_{n'k'}<T_{nk'}$ 且 $n'<n$，证明对所有的 $k\geqslant k'$ $T_{n'k}\leqslant T_{nk}$。(c)证明有一个非减的序列 $\langle M_n\rangle$，使得当 $M_n\leqslant S<M_{n+1}$ 时 $\Sigma_n(S)>\min_{j\geqslant 1}\Sigma_j(S)$，但当 $S\geqslant M_{n+1}$ 时 $\Sigma_n(S)>\min_{j\geqslant 1}\Sigma_j(S)$，〔参考(19)〕。

15. 〔$M43$〕 证明或否定，$\Sigma_{n-1}(m)<\Sigma_n(m)$ 蕴涵着 $\Sigma_n(m)\leqslant\Sigma_{n+1}(m)\leqslant\Sigma_{n+2}(m)\leqslant\cdots$。〔这样一个结果将大大地简化表2的计算。〕

16.〔$M43$〕 确定具有最优虚拟路段分布的多阶段合并的渐近行为。

17.〔32〕 证明或否定：通过把一个路段（在一条适当的带上）加到S个初始路段的分布上，来形成$S+1$个初始路段的分布，这个方法可用来散布一个最优多阶段分布的路段。

18.〔30〕 如果我们坚持初始路段至多被放置在$T-1$条带上，则在处理的初始路段个数极小的意义下，最优多阶段分布是否产生最好的合并型式？（忽略重绕时间。）

19.〔21〕 试对于六条带上的卡伦多阶段排序，构造类似于（1）的一张表。

20.〔$M24$〕 对于六条带上的卡伦多阶段排序，什么母函数对应于（7）和（16）？类似于（9）和（27）的什么关系定义合并数的中？

21.〔11〕 在（26）中的第7级上，将出现什么？

22.〔$M21$〕 序列（24）的每一项近似地等于前面两项之和。这种现象对于序列中剩下的数是否成立？叙述并证明关于$t_n-t_{n-1}-t_{n-2}$的一个定理。

▶23.〔29〕 如果把（23）变成$v_{n+1}=u_{n-1}+v_{n-1}+u_{n-2}$，$u_n=v_{n-2}+u_{n-3}+v_{n-3}+u_{n-4}+v_{n-4}$，则对于（25）、（27）和（28）应作什么改变？

24.〔$M41$〕 当把v_{n+1}定义为$u_{n-1}+v_{n-1}+\cdots+u_{n-P}+v_{n-P}$的头$q$项之和时，试对各种$P=T-2$及$0\leqslant q\leqslant 2P$计算带分开的多阶段过程的渐近行为（正文仅仅处理$q=2\lfloor P/2\rfloor$的情况；参考习题23）。

25.〔19〕 说明：本节末尾提到的在四条带上的带分开多阶段合并将如何对32个初始路段排序（给出逐个阶段的分析，象在正文中82个路段的六条带的例子那样）。

26.〔$M21$〕 分析当$S=2^n$和当$S=2^n+2^{n-1}$时，在四条带上的分带多路合并的行为（见习题25）。

27.〔23〕 在一个完全分布中一旦初始路段已经分布到一些带上，则多阶段策略只不过是“合并直到空”。我们从所有非空输入带合并路段，直到它们之一变成空的为止；然后使用该带作为下一输出带，并且让以前的输出带作为输入之一。

是否不论初始路段如何分布，只要我们至少把它们分布到两条带上，则这个合并直到空的策略就总能完成？（当然，一条带将保持为空，以便它能作为头一条输出带。）

28.〔$M26$〕 上边的习题定义了一族相当广泛的合并型式，说明在下列意义下多阶段法是其中最好的：如果有六条带，而且如果我们考虑所有这样的初始分布类（a，b，c，d，e），在这个类中合并直到空的策略要求不超过n个阶段来进行排序，则$a+b+c+d+e\leqslant t_n$，其中t_n是对于多阶段排序（1）的相应的值。

29.〔$M47$〕 习题28说明，在极少阶段的意义下，多阶段分布是在所有“合并直到成为空”的型式中最优的，但是在极少扫描的意义下它也是最优的吗？

设a和b互质，而且假定$a+b$是裴波那契数F_n。证明或否定下列的R.M.卡普的猜想：在以分布（a，b）开始的合并直到空型式期间处理的初始路段数，大于或等于$((n-5)F_{n+1}+(2n+2)F_n)/5$（当$a=F_{n-1}$，$b=F_{n-2}$时，达到此数）。

30.〔42〕 对于带分开多阶段合并，试编制类似表2的一张表。

5.4.3 级联合并

另一个称为“级联合并”的基本型式，实际上在多阶段法之前就发现了〔B.K.Betz

and W. C. Carter, *ACM Nat'l Conference* 14 (1959), Paper 14]。沿用5.4.2节所给出的记号，下表给出在六条带和190个初始路段的情况下使用该法的概要。

	T1	T2	T3	T4	T5	T6	处理的初始路段
扫描1	1^{55}	1^{50}	1^{41}	1^{29}	1^{15}	—	190
扫描2	—	1^{5}_{*}	2^{9}	3^{12}	4^{14}	5^{15}	190
扫描3	15^{5}	14^{4}	12^{3}	9^{2}	5^{1}_{*}	—	190
扫描4	—	15^{1}_{*}	29^{1}	41^{1}	50^{1}	55^{1}	190
扫描5	190^{1}	—	—	—	—	—	190

一个级联合并象多阶段法一样，以诸带上的路段的一个“完全的分布”开始，尽管完全分布的规则有些不同于5.4.2节中的那些。表中的每一行表示对所有数据的一次完全的扫描。例如，第二遍扫描是通过对T1，T2，T3，T4，T5进行一次五路合并到T6，直到T5成为空为止（这把相对长度为5的15个路段放置到T6上），然后对T1，T2，T3，T4进行一次四路合并到T5，然后进行三路合并到T4，然后进行两路合并到T3，最后从T1进行一路合并（一个复写操作）到T2。第三遍扫描与此类似，首先进行一次五路合并直到一条带变成空的为止，然后进行一次四路合并，等等。〔也许，本书的这节应编号为5.4.3.2.1，而不是5.4.3!〕

显然复写操作是不必要的，故可以省略它们。然而，实际上，在六条带的情况下，这个复写仅仅花费总时间的很小的一部分。在上面的表中以星号标出的项目，就是简单地被复写的那些；处理的950个路段中仅有25个是这种类型的。大多数时间都花费在五路合并和四路合并上。

若与多阶段法进行比较，则第一个印象可能觉得级联型式是一种相当拙劣的选择，因为标准多阶段法始终使用$T-1$路合并，而级联则使用$T-1$路，$T-2$路，$T-3$路，等等。但事实上，在六条或更多的带上，它渐近地比多阶段法更好！如同我们已经在5.4.2节中发现的，一个高阶的合并并不保证高效性。同5.4.2节中的表类似，表1示出了级联合并的性能特征。

表1　级联合并排序的近似行为

带	扫描（有复写）	扫描（没有复写）	增长率
3	$2.078 \ln S + 0.672$	$1.504 \ln S + 0.992$	1.6180340
4	$1.235 \ln S + 0.754$	$1.102 \ln S + 0.820$	2.2469796
5	$0.946 \ln S + 0.796$	$0.897 \ln S + 0.800$	2.8793852
6	$0.796 \ln S + 0.821$	$0.773 \ln S + 0.808$	3.5133371
7	$0.703 \ln S + 0.839$	$0.691 \ln S + 0.822$	4.1481149
8	$0.639 \ln S + 0.852$	$0.632 \ln S + 0.834$	4.7833861
9	$0.592 \ln S + 0.861$	$0.587 \ln S + 0.845$	5.4189757
10	$0.555 \ln S + 0.869$	$0.552 \ln S + 0.854$	6.0547828
20	$0.397 \ln S + 0.905$	$0.397 \ln S + 0.901$	12.4174426

不难导出对于六条带的一个级联合并的“完全分布”。它们是

级	T1	T2	T3	T4	T5
0	1	0	0	0	0
1	1	1	1	1	1
2	5	4	3	2	1
3	15	14	12	9	5

4	55	50	41	29	15
5	190	175	146	105	55
· · ·

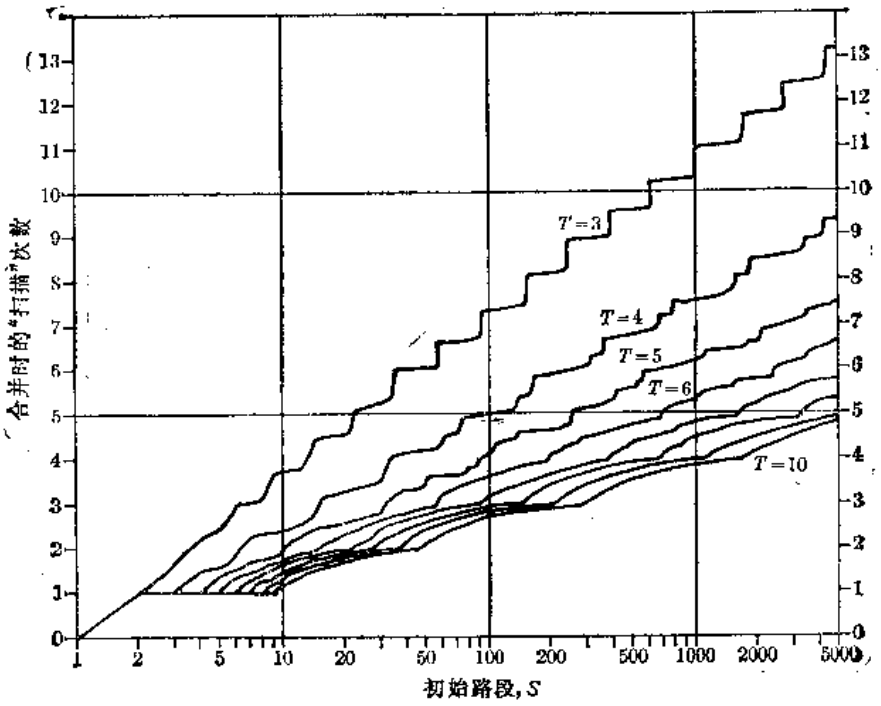


图74　具有算法D的分布的级联合并的效率

$$D_1 \leqslant a+b+c+d,\ D_2 \leqslant a+b+c,\ D_3 \leqslant a+b,$$
$$D_4 \leqslant a_1,\ D_5=0,\ D_1 \geqslant D_2 \geqslant D_3 \geqslant D_4 \geqslant D_5 \qquad (2)$$

我们现在可以想像虚拟路段出现在诸带上的任何方便的位置。假定头一趟合并扫描通个五路合并产生 a 个路段，然后通个四路合并产生 b 个路段，等等。而我们的目标是安排这些虚拟路段以便用复写来代替合并。我们不妨按如下方式来做头一趟合并扫描：

1．如果$D_4=a$，则每个D_1，D_2，D_3，D_4都减去 a，并假定T 5 就是合并的结果。如果$D_4<a$，从带T 1到T 5合并 a 个路段，合并时使用的这 5 条带上的虚拟路段要尽可能地少，并使得D_1，D_2，D_3，D_4的新值满足

$$D_1 \leqslant b+c+d,\ D_2 \leqslant b+c,\ D_3 \leqslant b,\ D_4=0;\ D_1 \geqslant D_2 \geqslant D_3 \geqslant D_4 \qquad (3)$$

例如，如果D_2原来$\leqslant b+c$，则在这一步我们不使用来自它的虚拟路段，而如果 $b+c<D_2 \leqslant a+b+c$，则我们恰巧使用其中的D_2-b-c个。

2．（这一步类似于第一步，但已"移位"）如果$D_3=b$，则每个D_1，D_2，D_3都减去 b 而且设想T 4 就是合并的结果。如果$D_3<b$，则从带T 1 到T4 合并 b 个路段，必要时减少虚拟路段数以使得

$$D_1 \leqslant b+c,\ D_2 \leqslant b,\ D_3=0,\ D_1 \geqslant D_2 \geqslant D_3$$

3．等等。

弗格森的分布诸路段到诸带的方法，可以通过考察在（1）中从第 3 级进行到第 4 级的过程来说明。假设"逻辑的"带（T1，…，T5）分别包含（5，9，12，14，15）个路段，而且我们最终要把它变成为（55，50，41，29，15）。这个过程可以概括为如下所示。

	加到T1	加到T2	加到T3	加到T4	加到T5	“节省的数量”
步(1,1)	9	0	0	0	0	15+14+12+5
步(2,2)	3	12	0	0	0	15+14+9+5
步(2,1)	9	0	0	0	0	15+14+5
步(3,3)	2	2	14	0	0	15+12+5
步(3,2)	3	12	0	0	0	15+9+5
步(3,1)	9	0	0	0	0	15+5
步(4,4)	1	1	1	15	0	14+5
步(4,3)	2	2	14	0	0	12+5
步(4,2)	3	12	0	0	0	9+5
步(4,1)	9	0	0	0	0	5

我们首先放置9个路段于T1上，然后置(3,12)到T1和T2上，等等。如果在比如说步骤(3,2)期间输入已经穷尽，则“节省的数量”是15+9+5，意味着通过分配虚拟路段而避免了15个路段的5路合并，9个路段的两路合并，以及5个路段的一路合并。换句话说，在头一合并阶段出现于第3级的路段中，有15+9+5个不予处理。

以下的算法详细地定义了这一进程。

算法C (具有特殊分布的级联合并排序) 这个算法取初始的路段并把它们分散到诸带上，一次一个路段，直到提供的初始路段被穷尽为止。然后它确定诸带如何被合并，假定有$T\geqslant 3$台可供利用的磁带机，同时至多利用$T-1$路合并，并避免不必要的一路合并。带T可用来保留输入，因为它不接受任何初始的路段。

下列诸表是必需的。

A[j]，$1\leqslant j\leqslant T$：我们最近达到的完全级联分布。

AA[j]，$1\leqslant j\leqslant T$：我们正在争取的完全级联分布。

D[j]，$1\leqslant j\leqslant T$：假定在设备号为j的逻辑带上要出现的虚拟路段数。

M[j]，$1\leqslant j\leqslant T$：在设备号为j的逻辑带上希望的极大虚拟路段数。

TAPE[j]，$1\leqslant j\leqslant T$：对应于逻辑带设备号j的物理带设备号。

C1.〔初始化〕 对于$2\leqslant k\leqslant T$置A[k]←AA[k]←D[k]←0；并置A[1]←1，D[1]←1。对于$1\leqslant k\leqslant T$置TAPE[k]←k。最后，置$i\leftarrow T-2$，$j\leftarrow 1$，$k\leftarrow 1$，$l\leftarrow 0$，$m\leftarrow 1$，并转到步骤C5(这一处理手段是通过给控制变量赋适当的值，以便直接跳入内循环，从而一举启动整个过程的一种办法)。

C2.〔开始新的一级〕 (我们刚才已经达到一个完全的分布，而且由于输入还没有完，我们必须为下一级做好准备。) l增1，对于$1\leqslant k\leqslant T$置A[k]←AA[k]，然后对于$k=1,2,\cdots\cdots,T-1$，以这个顺序置AA[$T-k$]←AA[$T-k+1$]+A[k]。置(TAPE[1],TAPE[2],…,TAPE[$T-1$])←(TAPE[$T-1$],…,TAPE[2],TAPE[1])，并对于$1\leqslant k\leqslant T$置D[k]←AA[$k+1$]。最后置$i\leftarrow 1$。

C3.〔开始第i个子级〕 置$j\leftarrow i$(变量i和j表示在上面所示例子中的“步骤[i,j]”。

C4.〔开始步骤(i,j)〕 置$k\leftarrow j$和$m\leftarrow$A[$T-j-1$]。如果$m=0$和$i=j$，则置$i\leftarrow T-2$并返回C3，如果$m=0$和$i\neq j$，则返回C2(变量m表示有待写到TAPE[k]

上的路段数；$m=0$ 仅当 $l=1$ 时才出现)。

C5.〔输入到 TAPE〔k〕〕 写一个路段到号码为 TAPE〔k〕的带，而且D〔k〕减1。然后如果输入已穷尽，则重绕所有的带并转到步骤C7。

C6.〔推进〕 m 减1。如果 $m>0$，则返回C5；否则 k 减去1；如果 $k>0$，则置 $m\leftarrow$A〔$T-j-1$〕$-$A〔$T-j$〕，并返回C5。否则 j 减1；如果 $j>0$，则转到C4。否则 i 加1；如果 $i<T-1$，则返回C3。否则转回C2。

C7.〔准备合并〕 (这时初始分布已经完成，而且A、AA、D和 TAPE 诸表描述了诸带的现状。) 对于 $1\leqslant k<T$ 置M〔k〕$\leftarrow$AA〔$k+1$〕，并置 FIRST$\leftarrow 1$ (变量 FIRST 只在头一趟合并扫描期间非0)。

C8.〔级联〕 如果 $l=0$，则停止；排序完成并输出在 TAPE〔1〕上。否则，以 $p=T-1$，$T-2$，…，1 这个次序，进行从 TAPE〔1〕，…，TAPE〔p〕到 TAPE〔$p+1$〕的 p 路合并如下：如果 $p=1$，则通过简单地重绕 TAPE〔2〕，模拟一路合并，然后交换 TAPE〔1〕$\leftrightarrow$TAPE〔2〕。否则如果 FIRST$=1$ 且D〔$p-1$〕$=$M〔$p-1$〕，则通过简单地交换 TAPE〔p〕$\leftrightarrow$TAPE〔$p+1$〕，重绕 TAPE〔p〕，并从每个D〔1〕，…，D〔$p-1$〕，M〔1〕，…，M〔$p-1$〕都减去M〔$p-1$〕模拟 p 路合并。否则从每个 M〔1〕，…，M〔$p-1$〕减去M〔$p-1$〕。然后从每个满足 $1\leqslant j\leqslant p$ 且使D〔j〕$\leqslant$M〔j〕的 TAPE〔j〕合并一个路段，从每个满足 $1\leqslant j\leqslant p$ 且使D〔j〕$>$M〔j〕的D〔j〕减1；并且置输出路段到 TAPE〔$p+1$〕上。继续进行这一动作直到 TAPE(p)成为空为止。然后重绕 TAPE〔p〕和TAPE〔$p+1$〕。

C9.〔下降一级〕 l 减1，置 FIRST$\leftarrow 0$，且置 (TAPE〔1〕，…，TAPE〔T〕)$\leftarrow$(TAPE〔T〕，…，TAPE〔1〕)(这时，所有的D和M皆为0，而且将保持如此)。返回C8。

这个算法的C1～C6诸步作出分布，而C7～C9诸步进行合并；这两个部分彼此是相当独立的，而且将有可能在相同的存储单元中来保存M〔k〕和 AA〔$k+1$〕。

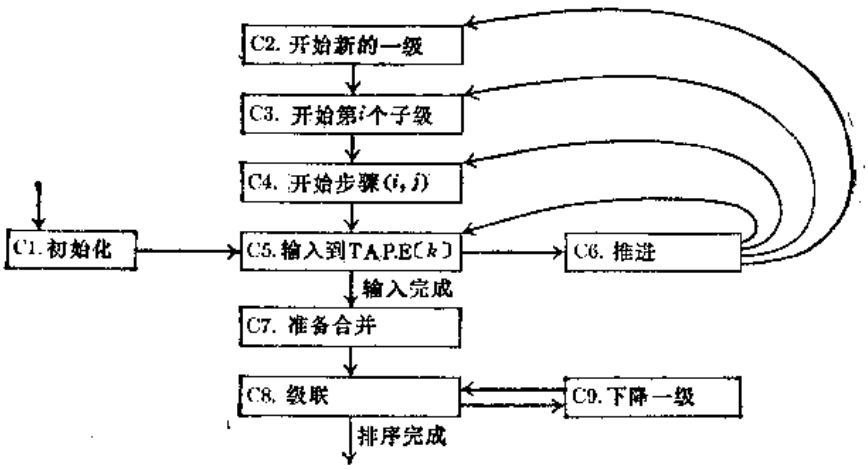


图75 具有特殊分布的级联合并

***级联合并的分析** 比起多阶段的情况来，级联合并是有些难以分析的，但由于出现了许多引人注意的公式，因而这个分析是特别有趣的。建议喜欢离散数学的读者，在进一步阅读之前，自己先来研究级联分布，因为这些数具有那么多不寻常的性质，去发现它们

是一件乐事！这里我们将讨论许多分析方法中的一种，重点放在去发现这些结果的方法上。

为了方便起见，让我们考虑六条带的情况，并寻求可推广到所有T的公式。关系式（1）导致了头一个基本的型式：

$$\left.\begin{aligned}
&a_n=a_n=\binom{0}{0}a_n\\
&b_n=a_n-e_{n-1}=a_n-a_{n-2}=\binom{1}{0}a_n-\binom{2}{2}a_{n-2}\\
&c_n=b_n-d_{n-1}=b_n-a_{n-2}-b_{n-2}=\binom{2}{0}a_n-\binom{3}{2}a_{n-2}+\binom{4}{4}a_{n-4}\\
&d_n=c_n-e_{n-1}=c_n-a_{n-2}-b_{n-2}-c_{n-2}=\binom{3}{0}a_n-\binom{4}{2}a_{n-2}+\binom{5}{4}a_{n-4}-\binom{6}{6}a_{n-6}\\
&e_n=d_n-b_{n-1}=d_n-a_{n-2}-b_{n-2}-c_{n-2}-d_{n-2}=\binom{4}{0}a_n-\binom{5}{2}a_{n-2}+\binom{6}{4}a_{n-4}-\binom{7}{6}a_{n-6}\\
&\quad+\binom{8}{8}a_{n-8}
\end{aligned}\right\}\quad(4)$$

设$A(z)=\sum\limits_{n\geqslant 0}a_nz^n$，…，$E(z)=\sum\limits_{n\geqslant 0}e_nz^n$，并定义多项式

$$\begin{aligned}
q_m(z)&=\binom{m}{0}-\binom{m+1}{2}z^2+\binom{m+2}{4}z^4-\cdots\\
&=\sum_{k\geqslant 0}\binom{m+k}{2k}(-1)^kz^{2k}\\
&=\sum_{0\leqslant k\leqslant m}\binom{2m-k}{k}(-1)^{m-k}z^{2m-2k}
\end{aligned}\qquad(5)$$

（4）的结果可概括如下：

$B(z)-q_1(z)A(z)$，…，$E(z)-q_4(z)A(z)$简化为对应于边界条件的诸有限和，即对于小的n，出现于（4）中但不在$A(z)$中的a_{-1}，a_{-2}，a_{-3}，…的值。为了提供适当的边界条件，让我们向后往负的方向来运行递归式直到－8级：

n	a_n	b_n	c_n	d_n	e_n
0	1	0	0	0	0
−1	0	0	0	0	1
−2	1	−1	0	0	0
−3	0	0	0	−1	2
−4	2	−3	1	0	0
−5	0	0	1	−4	5
−6	5	−9	5	−1	0
−7	0	−1	6	−14	14
−8	14	−28	20	−7	1

(在7条带上，这张表也是类似的，只是奇数n的项右移一列。）序列a_0，a_{-2}，$a_{-4}\cdots=1$，1，2，5，14，…对于计算机科学家说来是司空见惯的，因为它同如此多的递归算法相关联而出现（例如习题2.2.1-4和方程2.3.4.4-13）；我们因此猜测在T带的情况下

$$a_{-2n}=\binom{2n}{n}\frac{1}{n+1}\quad 对于\ 0\leqslant n\leqslant T-2$$
$$a_{-2n-1}=0\qquad 对于\ 0\leqslant n\leqslant T-3 \tag{6}$$

为了验证这个选择是正确的，只要证明（6）和（4）对于第0级和第1级产生正确的结果就行了。在第1级上，这是显然的，而在第0级上，我们要对于$0\leqslant m\leqslant T-2$，验证

$$\binom{m}{0}a_0-\binom{m+1}{2}a_{-2}+\binom{m+2}{4}a_{-4}-\binom{m-3}{6}a_{-6}+\cdots$$
$$=\sum_{k\geqslant 0}\binom{m+k}{2k}\binom{2k}{k}\frac{(-)^k}{k+1}=\delta_{m0} \tag{7}$$

幸而这个和可以通过标准的技术来求值（它是1.2.6节的正文中基本例子之一的“问题2”）。

现在我们可以计算$B(z)-q_1(z)A(z)$等的系数。例如，考虑$D(z)-q_3(z)A(z)$中z^{2m}的系数；由1.2.6节中“问题3”的结果，它是

$$\sum_{k\geqslant 0}\binom{3+m+k}{2m+2k}(-)^{m+k}a_{-2k}=\sum_{k\geqslant 0}\binom{3+m+k}{2m+2k}\binom{2k}{k}\frac{(-1)^{k+m}}{k+1}$$
$$=(-1)^m\left(\binom{2+m}{2m-1}-\binom{3+m}{2m}\right)$$
$$=(-1)^{m+1}\binom{2+m}{2m}$$

因此我们已经推出

$$\begin{aligned}A(z)&=q_0(z)A(z)\\B(z)&=q_1(z)A(z)-q_0(z)\\C(z)&=q_2(z)A(z)-q_1(z)\\D(z)&=q_3(z)A(z)-q_2(z)\\E(z)&=q_4(z)A(z)-q_3(z)\end{aligned} \tag{8}$$

进而我们有$e_{n+1}=a_n$；因此$zA(z)=E(z)$，而且

$$A(z)=q_3(z)/(q_4(z)-z) \tag{9}$$

借助于q多项式，现在已经推导出诸母函数了，因而我们想要对诸q有更好的了解，习题1.2.9-15在这方面是有用的，因为它给了我们一个“封闭的”形式，它可以写成

$$q_m(z)=\frac{((\sqrt{4-z^2}+iz)/2)^{2m+1}+((\sqrt{4-z^2}-iz)/2)^{2m+1}}{\sqrt{4-z^2}} \tag{10}$$

如果我们现在置$z=2\sin\theta$则一切都简化了：

$$q_m(2\sin\theta)=\frac{(\cos\theta+i\sin\theta)^{2m+1}+(\cos\theta-i\sin\theta)^{2m+1}}{2\cos\theta}=\frac{\cos(2m+1)\theta}{\cos\theta} \tag{11}$$

(这种巧合导致我们猜想多项式 $q_m(z)$ 在数学上是已知的；确实，看一下适当的表将发现 $q_m(z)$ 基本上是第二类的契比雪夫多项式，即是通常记号下的 $(-1)^m U_{2m}(Z/2)$。)

我们现在可以确定（9）中分母的根：$q_4(2\sin\theta)=2\sin\theta$ 归结为

$$\cos 9\theta = 2\sin\theta\cos\theta = \sin 2\theta$$

当 $\pm 9\theta = 2\theta + \left(2n-\frac{1}{2}\right)\pi$ 时，我们能得到这个关系式的解；而且若 $\cos\theta=0$，则所有这样的 θ 都提供（9）中分母的根（当 $\cos\theta=0$ 时，$q_m(\pm 2)=\pm(2m+1)$ 决不等于 ± 2）。因此得到下列八个不同的根：

$$q_4(z)-z=0 \quad 当\ z=2\sin\frac{-5}{14}\pi,\ 2\sin\frac{-1}{14}\pi,\ 2\sin\frac{3}{14}\pi,$$

$$2\sin\frac{7}{22}\pi,\ 2\sin\frac{-3}{22}\pi,\ 2\sin\frac{1}{22}\pi,\ 2\sin\frac{-5}{22}\pi,\ 2\sin\frac{9}{22}\pi$$

时。由于 $q_4(z)$ 是 8 次多项式，这已枚举了所有的根。这些值的头三个值使得 $q_3(z)=0$，所以 $q_3(z)$ 和 $q_4(z)-z$ 有一个三次多项式作为公因式。如果我们展开（9）为部分分式，则其余五个根支配了 $A(z)$ 的系数的渐近行为。

在考虑一般的 T 带的情况下，命 $\theta_k=(4k+1)\pi/(4T-2)$。$T$ 带级联分布数的母函数 $A(z)$ 取形式

$$\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\frac{\cos^2\theta_k}{1-z/(2\sin\theta_k)} \tag{12}$$

(见习题 8)；因此

$$a_n=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos^2\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n \tag{13}$$

等式（8）现在导致类似的公式

$$\begin{aligned} b_n&=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 3\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n \\ c_n&=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 5\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n \\ d_n&=\frac{4}{2T-1}\sum_{-T/2<k<\lfloor T/2\rfloor}\cos\theta_k\cos 7\theta_k\left(\frac{1}{2\sin\theta_k}\right)^n \end{aligned} \tag{14}$$

等等。习题 9 说明对于所有 $n\geqslant 0$ 而不仅仅对于大的 n，这些等式都成立。在每个和式中，$k=0$ 的项高于所有其余的项，特别是当 n 相当大时，因此“增长率”是

$$\frac{1}{2\sin\theta_0}=\frac{2}{\pi}T-\frac{1}{\pi}+\frac{\pi}{48T}+O(T^{-2}) \tag{15}$$

W·C·卡特首先分析了级联排序〔*Proc. IFIP Congress*(1962).62-66〕，他得到了对于小的 T 的数值结果，戴维·E·弗格森也进行了分析〔见 *CACM*>(1964)297〕，他发现了增长率的渐近行为中的头两项。1964年夏R. W. 弗洛伊德发现了增长率的明显的形式 1/(2

$\times \sin\theta_0$)，于是对于所有的T都可以使用准确的公式。G. N. 拉尼（G. N. Raney）独立地进行了级联数的一个深入细致的分析〔*Canadian J. Math.* 18(1966), 332-349〕，他以与排排序毫无关系的完全不同的方式来研究它们。拉尼观察了图 73 的"对角线比率"原理，而且导出了这些数的许多其他有趣性质。弗洛伊德和拉尼在他们的证明中使用了矩阵处理（见习题 6）。

级联排序的修改 如果再加上一条带，则在级联排序期间，就有可能覆盖几乎所有的重绕时间。例如，我们可以把 T1～T5 合并到 T7，然后合并 T1～T4 到 T6，然后合并 T1～T3 到 T5（它现在已被重绕），然后合并 T1～T2 到 T4，而当已经重绕了 T4 上的比较短的数据时，即可开始下趟扫描。从对级联的上述分析中，即可预测这一过程的效率（关于进一步的信息见 5.4.6 节）。

在 *CACM* 6 (1963), 585-587 中，D. E. 克努特提出了一个"折衷合并"的方案，它包括多阶段和级联两者作为特殊情况，每个阶段都由$T-1$路，$T-2$路，…，P路合并组成，其中P是 1 和$T-1$之间的任何固定数。当$P=T-1$时，这是多阶段的。而当$P=1$时，它是"纯粹的级联"。当$P=2$时，它是没有复写阶段的级联。C. E. 拉德克（C. E. Radke）已经对这个方案进行了分析〔*IBM System J.* 5(1966), 226-247〕。W. H. 伯奇也进行了分析〔*Proc IFIP Congress*(1971)〕。伯奇找到了对于每个（P, T）折衷合并的母函数$\Sigma T_n(x)z^n$，并推广了等式 5.4.2-16；他发现，如果把初始路段数作为S的函数，当$S\to\infty$时，要求初始路段数为最少（利用一个宜截了当的分布方案，并忽略重绕时间），则对于$F=(3, 4, 5, 6, 7, 8, 9, 10, 11)$ P的最好值分别为（2，3，3，4，4，4，3，3，4）。当T增加时P的这些值更接近于级联而不是更接近于多阶段；由此得出，折衷合并决不实质性地好于级联。另一方面，如同 5.4.2 节所述，在选择最优的级数和最优的虚拟路段分布的情况下，纯粹的多阶段法似乎在所有折衷合并中是最好的。不幸的是，最优分布比较难以实现。

索·L. 约翰逊（Th. L. Johnsen）〔*BIT* 6 (1966), 129-143〕已经研究了平衡的和多阶段合并的组合；M. A. 戈茨（M. A. Goetz）已经提出平衡合并的一个重绕重叠的变种〔*Digital Computer User's Handbook*, ed. by M. Klerer and G. A. Korn (New York: McGraw-Hill, 1967), 1.311-1.312〕；而且还可以想象出许多其它混合的方案。

习题

1.〔10〕 利用表 1，试比较级联合并和5.4.2 节中所述的多阶段法的"带分开"方案，问哪一个更好？（忽略重绕时间。）

▶2.〔22〕 比较在三条带上利用算法 C 的级联排序和在三条带上利用算法 5.4.2D 的多阶段的排序。它们之间有什么类似性和差别？

3.〔20〕编制一张表，它说明当在六条带上利用算法 C 对 100 个初始路段排序时所发生的情况。

4.〔*M*20〕（G. N. 拉尼） 一个"第n级级联分布"是如下定义的多重集合（在六条带的情况下）：{1，0，0，0，0}是第 0 级级联分布；而且如果{a，b，c，d，e}是第n级级联分布，则{$a+b+c+d+e$，$a+b+c+d$，$a+b+c$，$a+b$，a}是第$n+1$级级联分布（因为一个多重集合是未排序的，从单单一个第n级的分布可以形成达

5！种不同的第（$n+1$）级分布)。(a）证明互质整数的任何多重集合$\{a, b, c, d, e\}$都是对于某个n的第n级级联分布。(b）证明，在下列意义下，对于级联排序定义的分布是最优的，即如果$\{a, b, c, d, e\}$是使$a \geqslant b \geqslant c \geqslant d \geqslant e$的任何第$n$级分布，则我们有$a \leqslant a_n$，$b \leqslant b_n$，$c \leqslant c_n$，$d \leqslant d_n$，$e \leqslant e_n$，其中$\{a_n, b_n, c_n, d_n, e_n\}$是在（1）中定义的分布。

▶5.〔20〕 证明(1)中定义的级联数满足定律 $a_k a_{n-k}+b_k b_{n-k}+c_k c_{n-k}+d_k d_{n-k}+e_k e_{n-k}=a_n$，对于$0 \leqslant k \leqslant n$。

〔提示：通过考虑在一个完整的级联排序的第k趟扫描期间共输出了多少各种长度的路段，来解释这个关系式。〕

6.〔$M20$〕 求一个5×5矩阵Q，使得对于所有$n \geqslant 0$，Q^n的头一行包含六条带的级联数a_n，b_n，c_n，d_n，e_n。

7.〔$M20$〕 假定级联合并被应用于c_n个初始路段的一个完全分布，试求当禁止一路合并时所节省下的工作量的公式。

8.〔$HM23$〕 推导(12)。

9.〔$HM26$〕 推导(14)。

▶10.〔$HM28$〕 不使用型式（4）来开始级联数的研究，而由恒等式

$$e_n=a_{n-1}=\binom{1}{1}a_{n-1}$$

$$d_n=2a_{n-1}-e_{n-2}=\binom{2}{1}a_{n-1}-\binom{3}{3}a_{n-3}$$

$$c_n=3a_{n-1}-d_{n-2}-2e_{n-2}=\binom{3}{1}a_{n-1}-\binom{4}{3}a_{n-3}+\binom{5}{5}a_{n-5}$$

等开始，命

$$r_m(z)=\binom{m}{1}z-\binom{m+1}{3}z^3+\binom{m+2}{5}z^5-\cdots$$

用这些r个多项式来表达$A(z)$，$B(z)$，等等。

11.〔$M38$〕 设

$$f_m(z)=\sum_{0 \leqslant k \leqslant m}\binom{\lfloor (m+K)/2 \rfloor}{K}(-1)^{\lceil K/2 \rceil}z^K$$

证明，T带级联数的母函数$A(z)$等于$f_{T-3}(z)/f_{T-1}(z)$，其中这个表达式中的分子和分母没有公因式。

12.〔$M40$〕 证明在下面的意义下，弗格森的分布方案是最优的。即在头一次扫描期间任何满足（2）的分配虚拟路段的方法都不能减少需要处理的初始路段个数（假定在这个扫描期间使用步骤C7～C9的策略)。

13.〔40〕 正文提议通过增加一条额外的带来覆盖大多数重绕时间。试剖析这个思想，(例如，正文的方案包含等候T4重绕；如果从下次扫描的头一个合并阶段省略T4，是否将更好些?)

5.4.4 向后读带

许多磁带机都有从被写入的方向相反的方向来读带的能力。我们迄今遇到的合并型总是以“向前”的方向把信息写到带上，然后重绕带，向前读，而且再次重绕它（因此带文件象队那样，以先进先出的方式来操作）。而向后读允许我们消除上面的两个重绕操作：即我们向前写带和向后读它（在这种情况下，文件象栈那样处理，因为它们以后进先出的方式被使用）。

平衡的、多阶段的、以及级联合并型式都能够适应向后读。主要差别是，当我们向后读和向前写时，合并颠倒了路段的顺序。如果两个路段在带上有递增的次序，则我们在向后读时可以合并它们，但这产生了递降的次序。这样产生的递降的次序，在下次扫描时将随继变成为递增的次序，所以合并算法必须有能力来处理递增或递降次序的路段。一个头一次向后读的程序员会经常感到他自己是倒立着的。

作为向后读的一个例子，考虑利用平衡合并在四条带上合并 8 个初始路段。这些操作可以综述如下：

	T1	T2	T3	T4	
扫描 1	$A_1A_1A_1A_1$	$A_1A_1A_1A_1$	—	—	初始分布
扫描 2	—	—	D_2D_2	D_2D_2	合并到 T3 和 T4
扫描 3	A_4	A_4	—	—	合并到 T1 和 T2
扫描 4	—	—	D_8	—	最后合并到 T3

这里 A_r 代表相对长度为 r 的路段，如果象我们以前的例子那样向前读这条带，则它的次序是递增的；D_r 是长度为 r 的递降的对应路段记号。在第 2 趟扫描期间递增的路段变成递降的：在输入时它们表现为递降的，因为我们在向后读。在第 3 次扫描时，它们又转换了方向。

注意，上边的过程以按递降次序出现在带 T3 上的结果而结束。如果这是不适当的（依赖于输出是否向后读，还是被卸下并取走以供将来使用），则我们可以把它复写到另一条带上，同时逆转方向。一种较快的方式将是在第 3 次扫描后重绕 T1 和 T2，并且在第 4 遍扫描时产生 A_8。还有一种更快的方式是在第一次扫描期间从八个递降路段开始，因为这将交换所有的 A 和 D。然而，对于 16 个初始路段的平衡的合并将要求初始路段是递增的；而且由于我们通常预先并不知道将形成多少初始路段，因此有必要来选择一个一致的方向，所以，在第 3 次扫描之后重绕的思想，大概是最好的。

级联合并以相同的方式进行。例如，考虑在四条带上对 14 个初始路段排序：

	T1	T2	T3	T4
第 1 遍扫描	$A_1A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1$	$A_1A_1A_1$	—
第 2 遍扫描	—	D_1	D_2D_2	$D_3D_3D_3$
第 3 遍扫描	A_6	A_5	A_3	—
第 4 遍扫描	—	—	—	D_{14}

又是如此：如果我们恰在最后一趟扫描之前重绕 T1，T2，T3，则我们可以产生 A_{14} 而不是 D_{14}。注意，在所有的一路合并都已经明显地执行的意义下，这是一个“纯粹的”级联

合并，如果刚才象在算法 5.4.3.C● 那样禁止复写操作，则在第 2 次扫描后我们将面临情况

$$A_1 \qquad — \qquad D_1D_2 \qquad D_3D_3D_3$$

而且已不可能继续三路合并，因为我们不可能合并处于相反方向的路段！如果我们重绕 T1 而且在下一个合并期间向前读它（同时向后读 T3 和T4），则可避免复写 T1 到 T2 的操作。但在合并之后再次需要重绕 T1，所以这技巧以两次重绕来交换一次复写。

因此算法 5.4.3.C 的分布方法对于向后读并不象向前读那样有效；每当初始路段数通过一个“完全”的级联分布数时，所需要的时间数量有一颇为急剧的跳跃。另一种散布的技术可用来给出在完全的级联分布之间的一个更为光滑的转换（见习题17）。

向后读的多阶段法 乍一看去（甚至第二次和第三次看去！）多阶段合并方案似乎完全不适合于向后读。例如，假设有 13 个初始路段和三条带：

	T1	T2	T3
阶段 1.	$A_1A_1A_1A_1A_1$	$A_1A_1A_1A_1A_1A_1A_1A_1$	—
阶段 2.		$A_1A_1A_1$	$D_2D_2D_2D_2D_2$

现在我们停住；我们可以重绕 T2 或 T3，然后向前读它，同时向后读其它带，但这将得到颇为混杂的情况，且通过向后读我们获益颇小。

一种挽回局势的巧妙的想法是交替每条带上路段的方向。然后，合并即可在完满的同步下进行：

阶段1.	$A_1D_1A_1D_1A_1$	$D_1A_1D_1A_1D_1A_1D_1A_1$	—
阶段2.	—	$D_1A_1D_1$	$D_2A_2D_2A_2D_2$
阶段3.	$A_3D_2A_3$	—	D_2A_2
阶段4.	A_3	D_5A_5	—
阶段5.	—	D_5	D_8
阶段6.	A_{13}	—	—

这一原理是由 R. L. 吉尔斯塔德（R. L. Gilstad）在他关于多阶段合并的开创性论文中简略地提到的，他又在 *CACM*6(1963)，220-223 中作了更详细的描述。

对于任何数目的带上的多阶段合并，ADA…技术都能有效地工作；因为我们可以证明，在每个阶段诸 A 和诸 D 都将被适当地同步，为此仅仅假定初始的分布扫描在每条带上产生交替的诸 A 和诸 D，而且每条带都以 A 结束（或者每条带以 D 结束）；因为在一个阶段期间写到输出文件上的最后路段，和由输入文件所取来的诸最后路段的方向相反，因此下一阶段总发现它的路段处于适当的方向。进而我们在习题 5.4.2-13 中已经看到，大多数完全的斐波那契分布都要求在一条带上有奇数个路段（最后的输出带），而在其他的带上都有偶数个路段。如果把 T1 标志成最后的输出带，则我们因此能够保证，如果 T1 以 A 开始，而且剩下的诸带都以 D 开始，则所有的带都以 A 路段告终。可以使用类似于算法 5.4.2D 的一个分布方法，但修改成使在每级上的分布以T 1作为最后的输出带（我们跳过级 1，$T+1$，$2T+1$，…，因为在这些级中，起始为空的带是最后的输出带）。例如，在六条带的情况下，我们可以使用下列的分布数代替 5.4.2-(1)：

● 原书误为5.4.3.D。——译注

级	T1	T2	T3	T4	T5	总共	最后的输出将是在
0	1	0	0	0	0	1	T1
2	1	2	2	2	2	9	T1
3	3	4	4	4	2	17	T1
4	7	8	8	6	4	33	T1
5	15	16	14	12	8	65	T1
6	31	30	28	24	16	129	T1
8	61	120	116	108	92	497	T1

（1）

于是，T1 总得到奇数个路段，而 T2 到 T5 得到偶数个路段，路段按递降次序排列，这是为了在分配虚拟路段时可有灵活性。这样一种分布有其优点，即最后的输出带是预先知道的，而不管有待出现的初始路段数是多少。结果（见习题 3），当使用这个方案时，输出将总是以递增的次序出现在 T1 上。D. T. 古德温（D. T. Good-Win）和 J. L. 维恩（J. L. Venn）已经提出了用于处理向后读多阶段分布的另一个方法〔*CACM*，7（1964），315〕。若每条带上都以一个 D 路段开始，我们几乎可以象在算法 5.4.2D 中那样分布路段。当耗尽输入时，除非已经达到对全部奇数编号的带的分布了，否则即想象有一个虚拟路段在唯一的“奇数”带开始处。其它的虚拟路段想象成都在诸带的末端，或者组成为位于中间的对偶，下面的习题 5 中分析了虚拟路段的最优放置问题。

最优合并型式 迄今我们已经讨论了关于合并到带上的各种型式，但没有问及“最好”的方法。要确定最优的型式似乎是十分困难的，特别是在向前读的情况下，其中重绕时间同合并时间的相互作用难以处理。另一方面，当通过向后读和向前写完成合并时，所有重绕实质上都被消去，而且有可能得到相当好的最优合并型式的特性。理查德·M. 卡普已经引进某些非常有趣的方法来解决这个问题，我们将通过讨论他提出的理论来结束这一节。

首先，我们需要一种更令人满意的描述合并型式的方式来代替上边使用过的颇为神秘的“带内容”图。卡普已经提出了两种方法来做这件事，即合并型式的向量表示和树表示。表示的这两种形式实际上都是有用的，所以我们将依次来描述它们。

一个合并型式的向量表示，由一系列“合并向量” $y^{(m)},\cdots,y^{(1)}y^{(0)}$ 所组成，它们的每一个都有 T 个分量。以如下方式用 $y^{(i)}$ 来表示倒数第 i 个合并步：

$$y_j^{(i)}=\begin{cases}+1 & \text{如果第 } j \text{ 号带是对于合并的一个输入}\\ 0 & \text{如果第 } j \text{ 号带在合并中未用}\\ -1 & \text{如果第 } j \text{ 号带得到合并的输出}\end{cases} \qquad (2)$$

于是，$y^{(i)}$ 恰有一个分量是 -1，而其余的分量为 0 和 1。最后的向量 $y^{(0)}$ 是特殊的：它是单位向量，如果最后的排序输出出现在第 j 号设备处，则该向量在位置 j 处有 1，其余处则为 0。这些定义蕴涵着，向量和

$$v^{(i)}=y^{(i)}+y^{(i-1)}+\cdots+y^{(0)} \qquad (3)$$

表示倒数第 i 个合并步之前带上路段的分布。特别是，$v^{(m)}$ 说明初始分布扫描安置多少个路段到每条带上。

在本节中最先以表格形式描述的三种合并型式有下列向量表示：

平衡的($T=4$, $S=8$)	级联 ($T=4$, $S=14$)	多阶段($T=3$, $S=13$)
$v^{(7)}=(4,4,0,0)$	$v^{(13)}=(6,5,3,0)$	$v^{(12)}=(5,8,0)$
$y^{(7)}=(+1,+1,-1,0)$	$y^{(13)}=(-1,+1,+1,-1)$	$y^{(12)}=(+1,+1,-1)$
$y^{(6)}=(+1,+1,0,-1)$	$y^{(9)}=(+1,+1,+1,-1)$	$y^{(11)}=(+1,+1,-1)$
$y^{(5)}=(+1,+1,-1,0)$	$y^{(8)}=(+1,+1,+1,-1)$	$y^{(10)}=(+1,+1,-1)$
$y^{(4)}=(+1,+1,0,-1)$	$y^{(7)}=(+1,+1,-1,0)$	$y^{(9)}=(+1,+1,-1)$
$y^{(3)}=(-1,0,+1,+1)$	$y^{(6)}=(+1,+1,-1,0)$	$y^{(8)}=(+1,+1,-1)$
$y^{(2)}=(0,-1,+1,+1)$	$y^{(5)}=(+1,-1,0,0)$	$y^{(7)}=(-1,+1,+1)$
$y^{(1)}=(+1,+1,-1,0)$	$y^{(4)}=(-1,+1,+1,+1)$	$y^{(6)}=(-1,+1,+1)$
$y^{(0)}=(0,0,1,0)$	$y^{(3)}=(0,-1,+1,+1)$	$y^{(5)}=(-1,+1,+1)$
	$y^{(2)}=(0,0,-1,+1)$	$y^{(4)}=(+1,-1,+1)$
	$y^{(1)}=(+1,+1,+1,-1)$	$y^{(3)}=(+1,-1,+1)$
	$y^{(0)}=(0,0,0,1)$	$y^{(2)}=(+1,+1,-1)$
		$y^{(1)}=(-1,+1,+1)$
		$y^{(0)}=(1,0,0)$

把这些向量倒着编号，即从 $y^{(m)}$ 开头而以 $y^{(0)}$ 结尾，似乎并不方便，但是这种特殊的观点证明对发展这一理论是有利的。为了寻找一个最优的方法，一个好办法就是以排好序的输出开始而且想象把它拆开到各条带上，然后再拆开这些，等等，同时，以同它们在排序过程中实际出现的顺序相颠倒的顺序，考虑依次的分布 $v^{(0)}$，$v^{(1)}$，$v^{(2)}$，…。实质上，这就是我们在对多阶段和级联合并进行分析时，已经采取的方法。

每个合并型式显然有一个向量表示。反之，容易看到，向量序列 $y^{(m)}\cdots y^{(1)}y^{(0)}$ 对应于一个实在的合并型式的充要条件是下列三点：

i）$y^{(0)}$ 是一个单位向量。

ii）对于 $m\geqslant i\geqslant 1$，$y^{(i)}$ 恰有一个分量等于 -1，所有其余的分量为 0 或 $+1$。

iii）对于 $m\geqslant i\geqslant 1$，$y^{(i)}+\cdots+y^{(1)}+y^{(0)}$ 的所有分量非负。

一个合并型式的树表示给出了同样信息的另一种图象。我们构造一株树，对于每个初始路段有一个外部“叶”节点，对于被合并的每个路段有一个内部节点，构造的方式是：每个内部节点的后裔是那样一些路段，该内部节点是从这些路段制作出来的。每个内部节点都以对应的开始时的步骤号来作标号，并且象在向量表示中那样，步骤是向后编号的；进而，每个节点上面的边以路段所在的带名来标出。例如，上列的三种合并型式有图 76 中所描述的树表示，其中我们称带为 A，B，C，D，以代替 T1，T2，T3，T4。

这个表示以方便的形式揭示了合并型式的许多有关的性质；例如，如果树的 0 级(根)上的路段是递增的，则 1 级上的路段必定是递降的，在 2 级上的那些必定是递增的，等等；一个初始路段是递增的，当且仅当对应的外部节点在一个偶数号级上。进而，在合并期间处理的初始路段的总数（不包括初始分布）恰等于这株树的外部路段长度，因为在 k 级上的每个初始路段都恰被处理 k 次。

每种合并型式都有一个树表示，但不是每个树都定义一种合并型式。一株其内部节点已经以数 1 到 m 标号，而其边已经以带名标号的树，表示一种正确的向后读的合并型式的

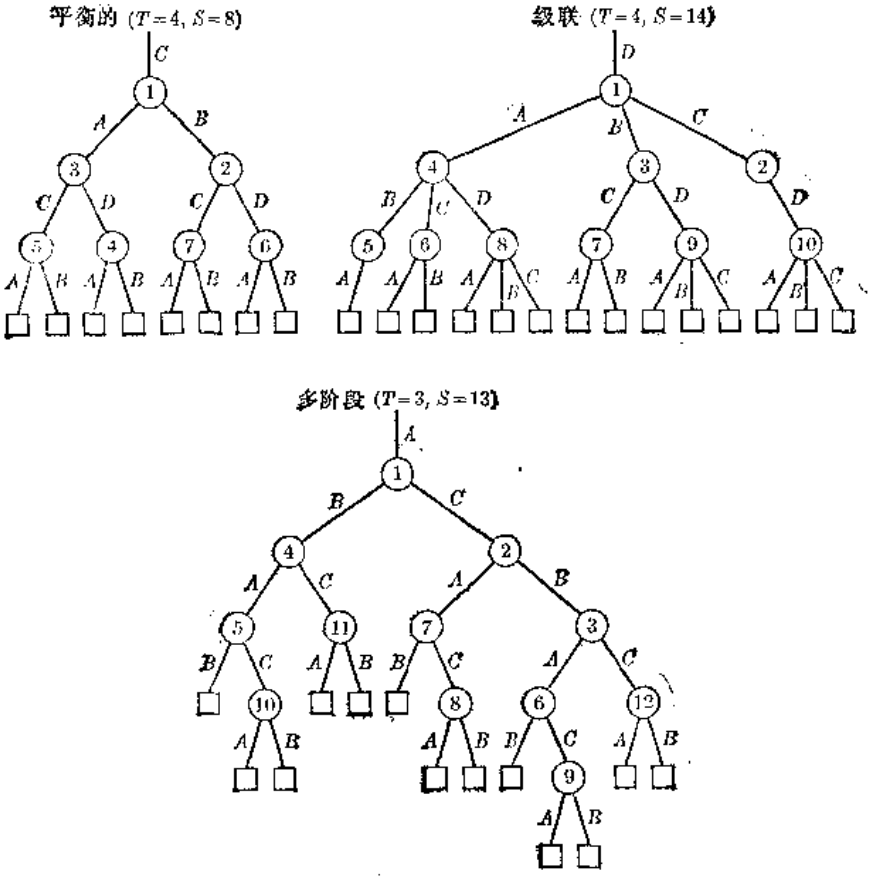
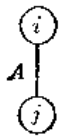


图76　三种合并型式的树表示

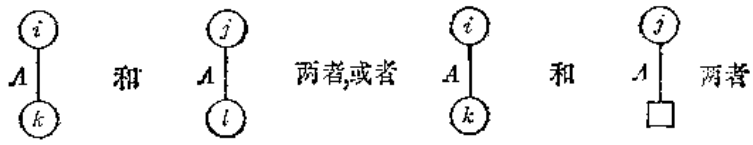
充要条件是

a）凡邻接于同一内部节点的两边，带名都不相同。

b）如果 $i>j$，且如果 A 是一个带名，则这树不包含下列构形：



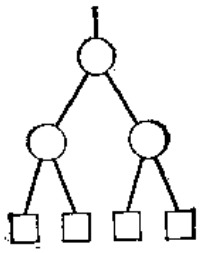
c）如果 $i<j<k<l$，而且如果 A 是一个带名，则这株树不包含下列构形



（4）

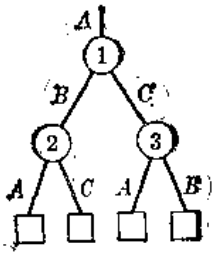
条件（a）是自明的，因为在一个合并中的输入和输出的带必须是不同的；类似地，（b）是显然的。“无交叉”条件（c）反映了后进先出的限制，它表征了在带上向后读的操作：在步骤 k 中形成的一个路段，必须比该同一带上以前形成的先撤销；因此（4）中构形是不可能的。不难验证，任何满足条件（a）、（b）、（c）的带标号的树确实对应于一种向后读的合并型式。

如果有T台磁带机，则条件（a）蕴涵着每个内部节点的次数是$T-1$或更小。并不总能对所有这样的树都附加上适当的标号；例如，当$T=3$时没有其树的形状为



（5）

的合并型式。如果我们能以适当的方式附加步骤号和带名称，则这个形状将导致一个最优的合并型式，因为它是在有四个外部节点的一株树中达到极小外部路段长度的唯一途径。但由于图式的对称性，实质上仅有一种方式按照条件（a）和（b）来加标号，这是

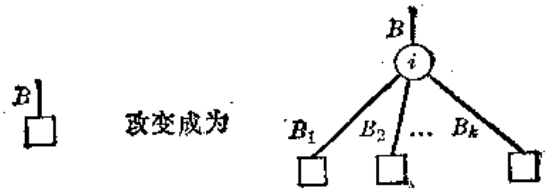


（6）

但这同条件（c）相冲突。按照上述条件利用T个带或较少的带名称可以加上标号的一种图形，称为T后进先出（T-Lifo）树。

还有一种方式，可以刻划产生于合并形式的所有带标号树，即考虑怎样能“长出”所有这样的树。从某个带名称，比如说A，以及以树秧

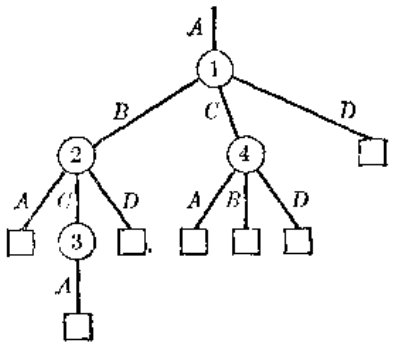
开始，在树生长过程中的第i步是选择不同的带名称B，B_1，B_2，…，B_k，并把对应于B的最近形成的外部节点



（7）

这个“最后形成，首先生长”的规则，精确地说明了树表示可以怎样从向量表示直接地构造出来。

要确定严格最优的T带合并型式，即从具有给定外部节点数的所有T后进先出树中选出具有极小路径长度的树，似乎是十分困难的。例如，下列并非显然的型式，可证明是向后读、合并四条带上的七个初始路段的一种最优方式。



(8)

为达到最优，一个一路合并实际上是必不可少的！（见习题8。）另一方面，对于任何固定的T，要给出渐近地最优的构造，倒并不那么困难。

设$K_T(n)$是在具有n个外部节点的一株T后进先出树中可达到的极小外部路径长度。由2、3、4、5节中展示的理论，不难证明

$$K_T(n)\geqslant nq-\lfloor((T-1)^q-n)/(T-2)\rfloor,\quad q=\lceil\log_{T-1}n\rceil \tag{9}$$

因为这是具有n个外部节点且所有节点的次数$<T$的任何树的极小外部路径长度，现在确切地知道的$K_T(n)$的值比较少。这里是某些上界，它们可能是准确的：

$$\begin{array}{lcccccccccccccccc} n= & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\ K_3(n)\leqslant & 0 & 2 & 5 & 9 & 12 & 16 & 21 & 25 & 30 & 34 & 39 & 45 & 50 & 56 & 61 \\ K_4(n)\leqslant & 0 & 2 & 3 & 6 & 8 & 11 & 14 & 17 & 20 & 24 & 27 & 31 & 33 & 37 & 40 \end{array} \tag{10}$$

卡普已经发现，任何其内部节点的次数$<T$的树都是准T后进先出的，就是说，只要改变某些外部节点成为一路合并，它就变成T后进先出的了。事实上，构造一组适当的标号是相当简单的。命A是一个特殊的带名称，则构造方法如下：

步骤1. 假定特殊名称A仅用在一个分枝的最左边的边上，则以同上述条件（a）相一致的任何方式，把诸带名称附加到树图式的诸边上。

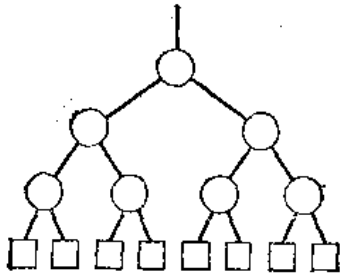
步骤2. 每当$B\neq A$时以

代替形如

的每一外部节点。

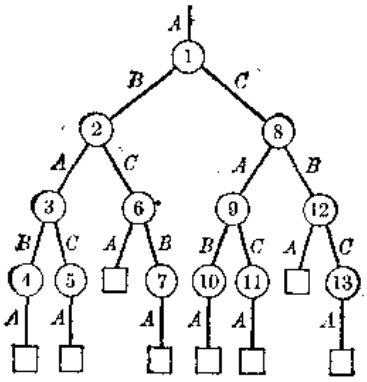
步骤3. 按先序为树的内部节点编号，结果就是满足条件（a）、（b）、（c）的一株带标号树。

例如，如果我们从树



(11)

和三条带开始，则这个过程可以赋标号如下：



(12)

不难验证，卡普的构造满足“最后形成，最先生长”的规则，这是由于先序的本性所致（见习题 12）。

注意，这个构造的结果是一个合并型式，它所有的初始路段都出现于带 A 上。这提示了以下的分布和排序方案，我们可以称之为先序合并。

P1. 分布初始路段到带 A 上，直到输入被穷尽为止。设 S 是初始路段的总数。

P2. 利用具有 S 个外部节点的一株极小路径长度的 $T-1$ 叉树，进行上述构造，得到一株其外部路径长度在（9）中下限的 S 倍之内的 T 后进先出树。

P3. 按照这一型式合并诸路径。

这个方案将在任何所希望的带上产生它的输出。但它有一个严重的缺陷——读者看出什么毛病了吗？问题在于合并型式要求开始时在带 A 上的某些路段是递增的，某些是递减的，取决于对应的外部节点是出现在奇数级上还是出现在偶数级上。这个问题，通过在恰好需要之前，把应是递减的诸路段，复写到一条或多条辅助带上，就可无须预先知道 S 而解决。然后，用初始路段的长度来表达，处理的总数量为

$$S\log_{T-1}S+O(S)$$

于是，当 $S\to\infty$ 时，先序合并肯定好于多阶段或级联合并。其实，它是渐近地最优的，因为（9）表明，$S\log_{T-1}S+O(S)$ 是我们在 T 条带上所能希望达到的最好者。另一方面，对于在实践中通常出现的较小的 S 值，先序合并是相当低效的；当 S 相当小时，多阶段法或级联方法是更简单和更快的。也许有可能想出一种简单的分布和合并方案，对于小的 S 值，它足以同多阶段法和级联法匹敌，而对于大的 S 它是渐近地最优的。

以下的第二组习题表明卡普如何以一种类似的方式陈述了向前读合并的问题。在这种情况下，这个理论证明是更为复杂的，尽管已经发现了某些非常有趣的结果。

习题——第一组

1.〔17〕 在向前读合并时，用一个键为 $+\infty$ 的人造“哨兵”来标志带上每个路段之结尾往往是方便的。试问当向后读时，如何修改这项作法？

2.〔20〕 象（1）的一个数组的诸列将总是非减的吗？或者，是否有这种可能，当我们从一级进行到下一级时，将要从某条带“减”去一些路段？

▶3.〔20〕 证明：如果 T1 原来从 $ADA\cdots$ 开始，而 T2 到 T5 以 $DAD\cdots$ 开始，则在（1）中描述的多阶段分布方法，当排序完成时，将总是在带 T1 上产生一个 A 路段。

4.〔22〕 在以递增次序分布所有路段之后，作向后读多阶段合并是一个好的想法吗？想象所有的“D”位置开始时都填以虚拟路段。

▶5.〔23〕 当使用向后读多阶段合并时，合并数串将有什么公式，以代替5.4.2节的（8），（9），（10）和（11）？通过画出象图71（a）的一个图形，示出在六条带上第五级分布的合并数。

6.〔07〕 什么是其树表示为（8）的合并型式的向量表示？

7.〔16〕 画出由下列向量序列定义的向后读合并型式的树表示：

$v^{(33)}=(20, 9, 5)$	$y^{(22)}=(+1,-1,+1)$	$y^{(10)}=(+1,+1,-1)$
$y^{(33)}=(+1,-1,+1)$	$y^{(21)}=(-1,+1,+1)$	$y^{(9)}=(+1,-1,+1)$
$y^{(32)}=(+1,+1,-1)$	$y^{(20)}=(+1,+1,-1)$	$y^{(8)}=(+1,+1,-1)$
$y^{(31)}=(+1,+1,-1)$	$y^{(19)}=(-1,+1,+1)$	$y^{(7)}=(+1,+1,-1)$
$y^{(30)}=(+1,+1,-1)$	$y^{(18)}=(+1,+1,-1)$	$y^{(6)}=(+1,+1,-1)$
$y^{(29)}=(+1,-1,+1)$	$y^{(17)}=(+1,+1,-1)$	$y^{(5)}=(-1,+1,+1)$
$y^{(28)}=(-1,+1,+1)$	$y^{(16)}=(+1,+1,-1)$	$y^{(4)}=(+1,-1,+1)$
$y^{(27)}=(+1,-1,+1)$	$y^{(15)}=(+1,+1,-1)$	$y^{(3)}=(-1,+1,+1)$
$y^{(26)}=(+1,-1,+1)$	$y^{(14)}=(+1,-1,+1)$	$y^{(2)}=(+1,-1,+1)$
$y^{(25)}=(+1,+1,-1)$	$y^{(13)}=(+1,-1,+1)$	$y^{(1)}=(-1,+1,+1)$
$y^{(24)}=(+1,-1,+1)$	$y^{(12)}=(-1,+1,+1)$	$y^{(0)}=(1, 0, 0)$
$y^{(23)}=(+1,-1,+1)$	$y^{(11)}=(+1,+1,-1)$	

8.〔23〕 证明当$S=7$和$T=4$时，（8）是向后读合并的一种最优方式，而避免一路合并的所有方法都比它差。

9.〔M22〕 证明下界（9）。

10.〔41〕 利用一台计算机，编制$K_T(n)$的准确值的表。

▶11.〔20〕 真或假：对于只使用T-1路合并的任何向后读合并型式，在每条带上只能有交替的$ADAD\cdots$的路段，如果两个相邻的路段以相同次序出现，那就不成了。

12.〔22〕 证明：卡普的先序构造将总是产生一个满足条件（a）、（b）、和（c）的带标号的树。

13.〔16〕 在先序仍然给出内部节点正确标号的前提下，尽可能多地撤销一路合并，以使（12）更为有效。

14.〔40〕 试设计一个算法，它进行先序合并，而无须明显地表示在步骤P2和P3中的树，要求此算法仅仅用$O(\log_2 S)$个内存单元来控制合并型式。

15.〔M39〕 正文中的卡普先序构造，产生在若干终端节点处具有一路合并的树。证明当$T=3$时，有可能构造渐近最优的3后进先出树，其中自始至终使用两路合并。

换言之，设$\hat{K}_T(n)$是所有的具有n个外部节点，且使得每个内部节点的次数为$T-1$的T后进先出树的极小外部路径长度，证明：$\hat{K}_3(n)=n\log_2 n+O(n)$。

16.〔M46〕 在习题15的记号下，当$n\equiv 1$（modulo$T-1$）时，是否对于所有的$T\geqslant 3$，$\hat{K}_T(n)=n\log_{T-1} n+O(n)$？

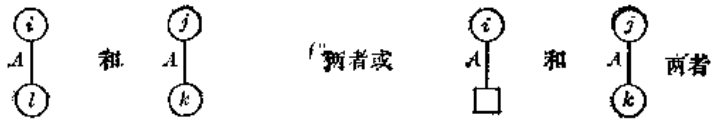
▶17.〔28〕（理查德·D.普拉特（Richard D. Pratt））为了在向后读级联合并中

实现递增次序，我们将坚持偶数次的合并扫描次数；这提示了有些不同于算法 5.4.3C 的初始分布技术。(a)改变5.4.3-1,使得它仅仅示出要求偶数次合并扫描的完全分布。(b)试设计一初始分布方案，它内插于这些完全分布之间。(于是，如果初始路段数落在几个完全分布之间，则希望两次合并某些路段，但不是所有的路段，以便达到一个完全的分布。)

18.〔46〕 假设对于某个$T \geqslant 3$，有T个磁带机可利用，而且T1含N个记录而其余的带为空。问是否有可能在少于$O(N\log_2 N)$步内颠倒T1上记录的顺序，而无须向后读？(当然，如果允许向后读，则操作是显然的。)对于要求阶为$N\log_2 N$步的这样一类算法，请参照习题 5.2.5-14。

习题——第二组

下列习题发展了向前读带的带合并理论；在这种情况下，每条带起一个队的作用，而不是一个栈的作用。一个合并型式可以完全像正文中那样表示成向量$y^{(m)}\cdots y^{(1)}y^{(0)}$的一个序列。但当我们把向量表示转换为一个树表示时，我们把“最后形成，首先生长”改为“首先形成，首先生长”，于是非法的构形（4）将改变为



(4′)

使用类似于向后读情况下T后进先出的术语，可把一株通过加标号来表示T条带上的一个向前读合并的树，称为T先进先出树。

当带可以向后读时，它们是非常好的栈。但不幸的是，它们不是非常好的通用的队。如果以先进先出的方式，随机地写和读，则我们浪费了大量的时间把带移来移去。甚至更坏的是，我们不久将跑出带的末尾！我们会遇到 2.2.2-(4) 和 (5) 中超出内存的队的同样问题，但是 2.2.2-(6) 和 (7) 中的解不能应用到带上。因为它们并不是圆形的循环。因此，如果我们可以对一株树加以标号，使得它对应的合并型式使每条带都遵循特殊的排队规则“写，重绕，读所有的记录，重绕；写，重绕，读所有的记录，重绕，等等”，则称这株树为强T-先进先出的。

▶19.〔22〕 (R. M. 卡普) 试求一株不是 3 先进先出的二叉树。

▶20.〔22〕 利用类似于 (4′) 的关于非法的带标号构形的相当简单的规则，建立强“T先进先出”的条件。

21.〔18〕 画出习题 7 中用向量定义的向前读合并型式的树表示，这株树是强 3 先进先出的吗？

22.〔28〕 (R. M. 卡普) 证明具有完全分布的多阶段和级联合并的树表示，除了标示内部节点的诸数外，对向后读及向前读两者的情况完全一样。

23.〔24〕 (R. M. 卡普) 如果没有随后用作输入带的输出带，即：如果不存在i，j，k，使$q \geqslant i > k \geqslant r$和$y_j^{(i)} = -1$和$y_j^{(k)} = +1$，则我们称一个合并型式的一段$y^{(q)}\cdots y^{(r)}$是一个时期，本题的目的是来证明级联合并使有相同的带数和初始路段数的所有合并型式的时期数极小化。

为叙述方便，我们定义某些记号。如果v和w是T向量，使得存在一个合并型式，它

在它的头一时期把w约化为v，则写之为$v \to w$（于是存在一个合并型式$y^{(m)}\cdots y^{(l)}$使得$y^{(m)}\cdots y^{(l+1)}$是一个时期，$w=y^{(m)}+\cdots+y^{(0)}$且$v=y^{(l)}+\cdots+y^{(0)}$）。如果v和w是T向量，使得对于$1\leqslant k\leqslant T$，v的最大k个元素之和$\leqslant w$的最大k个元素之和，则写之为$v \preceq w$。于是，例如，$(2,1,2,2,2,1)\preceq(1,2,3,0,3,1)$，因为$2\leqslant 3$，$2+2\leqslant 3+3$，…，$2+2+2+2+1+1\leqslant 3+3+2+1+1+0$。最后，如果$v=(v_1,\cdots,v_T)$，则命$C(v)=(s_T,s_{T-2},s_{T-3},\cdots,s_1,0)$，其中$s_k$是$v$的最大$k$个元素之和。

a）证明$v\to C(v)$。b）证明$v\preceq w$蕴涵着$C(v)\preceq C(w)$。c）假定习题24的结果，证明级联合并使时期的数目取极小。

24.〔$M35$〕 在习题23的记号下，证明$v\to w$蕴涵着$w\preceq C(v)$。

25.〔$M36$〕（R.M.卡普）如果没有既用作输入也用作输出的带，即，如果不存在i，j，k，使$q\geqslant i$，$k\geqslant r$且$y_j^{(i)}=+1$和$y_j^{(k)}=-1$，则我们称合并型式的一个段$y^{(q)}\cdots y^{(r)}$是一个阶段。本题的目的是来研究使阶段数极小化的合并型式。如果在一个阶段中w可以被约化为v（参考习题23中引进的类似记号），则写之为$v\Rightarrow w$。我们设$D_k(v)=(s_k+t_{k+1},\ s_k+t_{k-2},\ \cdots,\ s_k+t_T,\ 0,\cdots,0)$，其中$t_j$表示$v$的第$j$个最大的元素，且$v_k=t_1+\cdots+t_k$。（a）证明对于$1\leqslant k\leqslant T$，$v\Rightarrow D_k(v)$。（b）证明对于$1\leqslant k\leqslant T$，$v\preceq w$蕴涵着$D_k(v)\preceq D_k(w)$。（c）证明$v\Rightarrow w$蕴涵着对于某个$k$，$1\leqslant k<T$，$w\preceq D_k(v)$。（d）因此，在$q$个阶段和$T$条带的情况下，能使被排序的初始路段数达到极大的一个合并型式，可以通过一个整数序列$k_1k_2\cdots k_q$表示，使得初始分布是$D_{k_q}(\cdots(D_{k_2}(D_{k_1}(u)))\cdots)$，其中$u=(1,0,\cdots,0)$。这个极小阶段策略，有一个强的$T$先进先出表示，而且它也属于习题22中型式的类型。当$T=3$时，它就是多阶段合并。而对于$T=4,5,6,7$，它是平衡的合并的一种变种。

26.〔$M46$〕（R.M.卡普）对于所有$T\geqslant 4$和对于所有充分大的q，习题25中提到的最优序列$k_1k_2\cdots k_q$总是等于$1\lceil T/2\rceil\lfloor T/2\rfloor\lceil T/2\rceil\lfloor T/2\rfloor\cdots$，吗？

5.4.5 交替排序

合并排序的一个略为不同的方法，是由谢尔登·索贝尔（Sheldon Sobel）在$JACM$ 9 (1962)，373-375上介绍的。他不是从一个分布扫描（其中所有的初始路段被分散到诸条带上）开始，而是提出了在分布和合并之间交替执行的一个算法，使得在考察全部输入数据之前，就已经进行了许多排序。

例如，假设有五条带可供合并时利用，索贝尔方法将16个初始路段排序如下：

	操作	T1	T2	T3	T4	T5	"代价"
阶段1.	分布	A_1	A_1	A_1	A_1	—	4
阶段2.	合并	—	—	—	—	D_4	4
阶段3.	分布	—	A_1	A_1	A_1	D_4A_1	4
阶段4.	合并	D_4	—	—	—	D_4	4
阶段5.	分布	D_4A_1	—	A_1	A_1	D_4A_1	4
阶段6.	合并	D_4	D_4	—	—	D_4	4
阶段7.	分布	D_4A_1	D_4A_1	—	A_1	D_4A_1	4
阶段8.	合并	D_4	D_4	D_4	—	D_4	4
阶段9.	合并	—	—	—	A_{16}	—	16

这里如象在5.4.4节中那样，我们使用A_r和D_r分别代表相对长度为r的递增和递降的路段。本方法开始在四条带上各写一个初始路段，并且把它们（向后读）合并到第五条带上。分布再次继续，这次诸带循环地向右移一个位置，而第二个合并产生另一个路段D_4。在以这种方式形成四个D_4后，一个附加的合并建立A_{16}。我们可以继续建立另三个A_{16}，把它们合并成一个D_{64}。如此类推直到穷尽输入为止。预先不必知道输入的长度。

当初始路段的个数S是4^m时，不难看出这个方法恰处理每个记录$m+1$次（在分布期间一次和在合并期间的m次）。当S处于4^{m-1}和4^m之间时，我们可以假定存在虚拟路段，把S提升成4^m，因此总共的排序时间实际上将等于对于所有的数据进行$\lceil \log_4 S \rceil + 1$次扫描。这恰是通过对八条带的一个平衡排序所达到的；一般地说，具有T条工作带的交替排序等价于具有$2(T-1)$条带的平衡的合并，因为它进行对数据的$\lceil \log_{T-1} S \rceil + 1$次扫描。当$S$是$T-1$的一个乘方时，这是任何$T$带方法所能达到的最好效果，因为它达到等式5.4.4-9中的下界。另一方面，当S是$(T-1)^{m-1}+1$时，即恰巧比$T-1$的一个乘方大1时，这个方法几乎浪费整个一趟扫描。

习题2示出，如何通过使用一个特殊的结尾例程，来解决S的非完整乘方这个难题。1966年丹尼斯·L. 本彻（Dennes L. Bencher）发现了进一步的改进，他称他的过程是"交叉合并"〔见H. Wedekind, Datenorganisation (Berlin: W. de Gruyter, 1970), 166；和U.S. Patent 354000 (November 10, 1970)〕。主要的思想是延迟合并直到已经获得关于S的更多的知识为止。我们将讨论本彻创立的方案的稍加修改了的形式。

这个改进的交替排序进行如下：

	操作	T1	T2	T3	T4	T5	"代价"
阶段1.	分布	—	A_1	A_1	A_1	A_1	4
阶段2.	分布	—	A_1	A_1A_1	A_1A_1	A_1A_1	3
阶段3.	合并	D_4	—	A_1	A_1	A_1	4
阶段4.	分布	D_4A_1	—	A_1	A_1A_1	A_1A_1	3
阶段5.	合并	D_4	D_4	—	A_1	A_1	4
阶段6.	分布	D_4A_1	D_4A_1	—	A_1	A_1A_1	3
阶段7.	合并	D_4	D_4	D_4	—	A_1	4
阶段8.	分布	D_4A_1	D_4A_1	D_4A_1	—	A_1	3
阶段9.	合并	D_4	D_4	D_4	D_4	—	4

这时我们不把D_4合并到A_{16}中（除非输入将要穷尽），仅仅在实施

阶段15.	合并	D_4D_4	D_4D_4	D_4D_4	D_4	—	4

之后，我们将得到

阶段16.	合并	D_4	D_4	D_4	—	A_{16}	16

在再生成三个D_4之后，将出现第二个A_{16}

阶段22.	合并	D_4D_4	D_4D_4	D_4	—	$A_{16}D_4$	4
阶段23.	合并	D_4	D_4	—	A_{16}	A_{16}	16

等等（参看阶段1～5）。可以看出本彻方案的优点，例如，如果仅有五个初始路段，则在习题2中修改过的交替排序将进行4路合并（在阶段2），紧接着做一个二路合并，总共代价为$4+4+1+5=14$。而本彻方案则做一个二路合并（在阶段3），紧接着做一个

四路合并，总共代价为 4 + 1 + 2 + 5 = 12。(两个方法都含一个小的附加代价，即在最后的合并之前有一单位的重绕。)

以下的算法B中有本例方法的一个精确的描述。可惜，它似乎是比代码更难以理解的过程。向一台计算机说明这项技术，比之于向一位计算机科学家说明更容易！这部分地是由于它是一个递归方法，它已经表达成迭代的形式而且然后稍加“优化”了；读者可以发现，要真正了解它，有必要跟踪整个算法若干次。

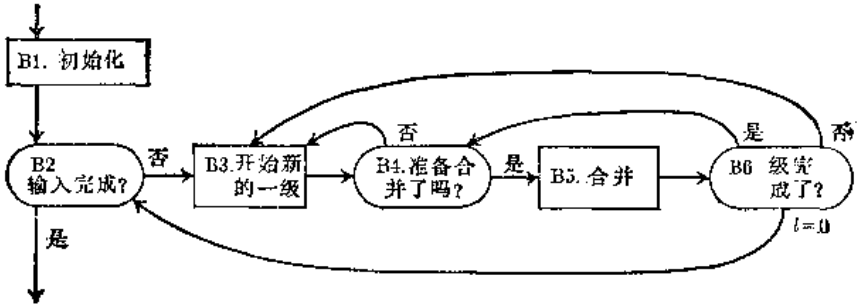


图77 具有一个交叉分布的交替排序

算法 B（具有交叉分布的交替排序） 这个算法取诸初始路段并把它们分散到诸带上，偶尔中断分布过程以便合并某些带的内容。这个算法使用 P 路合并，并假定有 $T = P - 1 \geqslant 3$ 台磁带机可利用（用于保持输入数据的磁带机不计在内）。这些磁带机必须允许以向前和向后来进行读入，而且以数 0，1，…，P 标记。下列的表格是必需的：

D〔j〕，$0 \leqslant j \leqslant P$ 假设出现在带 j 末端的虚拟路段号。

A〔l，j〕，$0 \leqslant l \leqslant L$，$0 \leqslant j \leqslant P$ 这里 L 是足够大的一个数，使输入的初始路段数不超过 P^{L+1}。当 A〔l，j〕$= k \geqslant 0$ 时，名义上长度为 P^k 的一个路段出现在带 j 上，这对应于算法操作的“级 l”。如果 k 为偶；这个路段是递增的，如果 k 为奇，则是递降的。当 A〔l，j〕$= -1$ 时，级 l 不使用带 j。

“写一初始路段到带 j 上”这一语句，是下列操作的简述：

置A〔l，j〕$\leftarrow 0$，如果输入已穷尽，则D〔j〕增1；否则在带 j 上写一初始路段(以递增的次序)。

“合并到带 j 上”这一语句是下列操作的简述：

如果对于所有 $i \neq j$，D〔i〕> 0，则对所有 $i \neq j$，D〔i〕减1且D〔j〕加1。否则从所有使D〔i〕$= 0$ 的带 i（$i \neq j$）取一个路段合并到带 j 上，而且对于所有其它 $i \neq j$，D〔i〕减1。

B 1.〔初始化〕 对于 $0 \leqslant j \leqslant P$，置D〔$j$〕$\leftarrow 0$。置A〔0，0〕$\leftarrow -1$，$l \leftarrow 0$，$q \leftarrow 0$。然后对于 $1 \leqslant j \leqslant P$，写一个初始路段到带 j 上。

B 2.〔输入完成？〕（这时带 q 是空的，而且其它带至多每条包含一个路段。）如果还有输入，则进行步骤B3。但如果输入已穷尽，就重绕所有使得A〔0，j〕为偶的带 j（$j \neq q$），然后在刚才重绕的带上向前读，而在其它的带上向后读，以此方式合并到带 q。这就完成了排序，且按递增次序排列的输出就在带 q 上。

B3.〔开始新的级〕 置 $l \leftarrow l+1$，$r \leftarrow q$，$s \leftarrow 0$ 且 $q \leftarrow (q+1) \bmod T$。对于 $1 \leqslant j \leqslant T-2$，各写一初始路段到带 $(q+j) \bmod T$ 上（于是在除开带 q 和 r 的每条带上，各写一个初始路段）。置 $A[l, q] \leftarrow -1$ 和 $A[l, r] \leftarrow -1$。

B4.〔准备好合并了吗？〕 如果 $A[l-1, q] \neq s$，则转回步骤 B3。

B5.〔合并〕 （这时对所有的 $j \neq q$，$j \neq r$，$A[l-1, q] = A[l, j] = s$。）合并到带 r（见上面这个操作的定义。）然后置 $s \leftarrow s+1$，$l \leftarrow l-1$，$A[l, r] \leftarrow s$ 以及 $A[l, q] \leftarrow -1$，置 $r \leftarrow (2q-1) \bmod T$。（一般说来，当 s 为偶时我们有 $r = (q-1) \bmod T$，当 s 为奇时 $r = (q+1) \bmod T$。）

B6.〔本级完成了吗？〕 如果 $l = 0$，则转到 B2，否则如果对于所有 $j \neq q$ 和 $j \neq r$，$A[l, r] = s$，则转到 B4，否则返回 B3。

我们可以使用一个“递归归纳法”风格的证明来证明这个算法是正确的，正如对算法 2.3.1T 所做的那样。假设我们以 $l = l_0$，$q = q_0$，$s_+ = A[l_0, (q_0+1) \bmod T]$，以及 $s_- = A[l_0, (q_0-1) \bmod T]$ 开始步骤 B3；并进而假定 $s_+ = 0$ 或 $s_- = 1$ 或 $s_+ = 2$ 或 $s_- = 3$ 或……。用归纳法能够验证这个算法最终将达到 B5 而无须改变 A 的第 0 到第 l_0 诸行，而且 $l = l_0 + 1$，$q = q_0 \pm 1$，$r = q_0$ 及 $s = s_+$ 或 s_-，这里如果 $s_+ = 0$ 或（$s_+ = 2$ 和 $s_- \neq 1$）或（$s_+ = 4$ 和 $s_- \neq 1$，3）或……时，我们选择正号 +，而如果（$s_- = 1$ 和 $s_+ \neq 0$）或（$s_- = 3$ 和 $s_+ \neq 0$，2）或……时，我们选择负号 −。这里简述的证明并不是很优美的，因为这个算法是以一种更适合于实现而不是验证的形式来陈述的。

图 78 借助于每个记录被合并的平均次数（作为初始路段数 S 的函数），示出了算法 B 的效率，其中假定初始路段都近似地等长（对于多阶段和级联排序对应的图出现在图 70 和 74 中），在编制这张图时，已经使用了习题 3 中提到的一点改进。

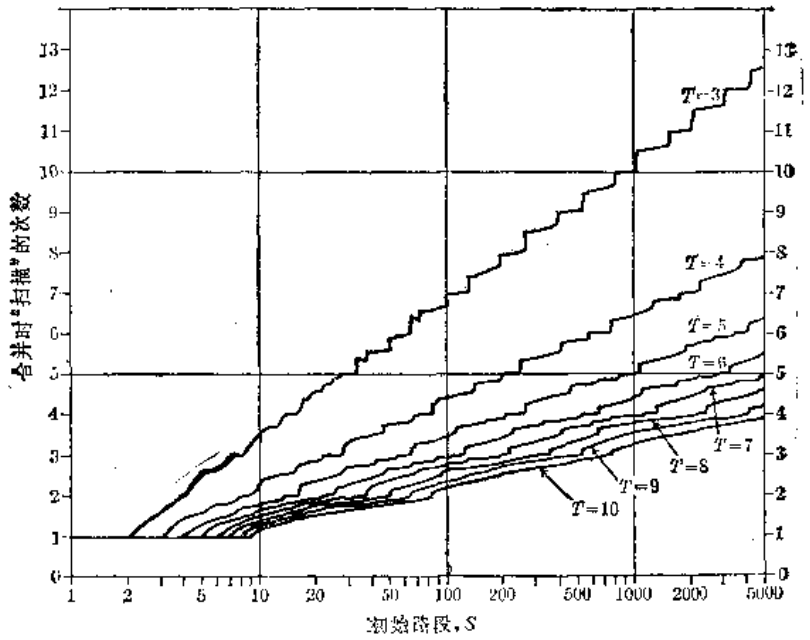


图78 交替排序的效率，其中使用了算法B和习题 3 的技术

向前读 交替排序型式看来要求向后读的能力，因为我们在合并新近输入的短路段时，也需要在某处存入长路段。然而，M. A. 戈茨〔Proc. AFIPS Spring Jt. Comp. Conf.

25(1964)，599-607〕已经发现了仅仅使用向前读和简单的重绕就能实现交替排序的方法。他的方法同这一章中我们已经见到过的其它方案有根本的不同。因为

a）数据有时写在带的前端，同时知道在带中间的现存数据未被破坏。

b）所有的初始字符串都有一固定的极大长度。

条件a）违背了我们已经假定的作为向前读的特征的“先进先出”的性质，但是只要在路段之间保留有充分数量的空白带而且在适当的时候忽略“奇偶错”的话，则这仍能可靠地实现。b）同有效地使用替代选择看来不大相容。

从作为一个算法而不是作为一部物理设备而获得专利的观点来看，戈茨的向前读交替排序是第一批这类算法之一。这是颇成问题的。〔U. S. Patent 3380029(April 23, 1968)〕；除非能把他驳倒，否则这意味着，如果未获得专利允许，在一个程序中使用这个算法是不合法的。本彻的向后读交替排序技术，已被IBM公司在若干年后当作专利〔因此，充分享受发现一个新算法的欢乐和愉快的时代已经过去！因为程序设计很类似于一部机器的制造，而且由于现在计算机程序很值钱，因而专利算法是不可避免的。当然把新技术完全保密的人的灵魂，比起在一段有限时间内作为专利，之后再把算法公开发表的人，要坏得多。〕

戈茨方法的中心思想，是把事情安排成使得每条带以相对长度为1的一个路段开始，紧接着是相对长度为P的，然后相对长度为P^2的，等等。例如，当$T=5$时，排序开始如下，利用“.”指出在每条带上当前读写头的位置：

	操作	T1	T2	T3	T4	T5	“代价”	注释
阶段1.	分布	$.A_1$	$.A_1$	$.A_1$	$.A_1$	$A_1.$	5	〔T5尚未重绕〕
阶段2.	合并	$\mathbf{A}_1.$	$\mathbf{A}_1.$	$\mathbf{A}_1.$	$\mathbf{A}_1.$	$A_1A_4.$	4	〔现在全都重绕〕
阶段3.	分布	$.A_1$	$.A_1$	$.A_1$	$A_1.$	$.A_1A_4$	4	〔T4尚未重绕〕
阶段4.	合并	$\mathbf{A}_1.$	$\mathbf{A}_1.$	$\mathbf{A}_1$	$A_1A_4.$	$\mathbf{A}_1.A_4$	4	〔现在全都重绕〕
阶段5.	分布	$.A_1$	$.A_1$	$A_1.$	$.A_1A_4$	$.A_1A_4$	4	〔T3尚未重绕〕
阶段6.	合并	$\mathbf{A}_1.$	$\mathbf{A}_1.$	$A_1A_4.$	$\mathbf{A}_1.A_4$	$\mathbf{A}_1.A_4$	4	〔现在全都重绕〕
阶段7.	分布	$.A_1$	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	〔T2尚未重绕〕
阶段8.	合并	$\mathbf{A}_1.$	$A_1A_4.$	$\mathbf{A}_1.A_4$	$\mathbf{A}_1.A_4$	$\mathbf{A}_1.A_4$	4	〔现在全都重绕〕
阶段9.	分布	$A_1.$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	$.A_1A_4$	4	〔T1尚未重绕〕
阶段10	合并	$A_1A_4.$	$\mathbf{A}_1.A_4$	$\mathbf{A}_1.A_4$	$\mathbf{A}_1.A_4$	$\mathbf{A}_1.A_4$	4	〔不重绕〕
阶段11.	合并	$A_1A_4A_{16}.$	$\mathbf{A}_1\mathbf{A}_4.$	$\mathbf{A}_1\mathbf{A}_4.$	$\mathbf{A}_1\mathbf{A}_4.$	$\mathbf{A}_1\mathbf{A}_4.$	16	〔现在全都重绕〕

等等。在阶段1期间，当T2接受它的输入时，T1被重绕，然后当T3接受它的输入时，T2被重绕，等等。最后，当输入穷尽时，虚拟路段将开始出现，有时有必要想象成它们已经以全长明显地写到带上。例如，如果$S=18$，则在带T4和T5上的诸A_1在阶段9期间都将是虚拟的；当阶段10期间从T2和T3合并到T1时，我们将向前跳到T4和T5，因为我们必须转到T4和T5上的诸A_4那里，以便为阶段11做准备。另一方面，T1上的虚拟路段A_1则不必明显地出现。因此，这“压轴戏”有一点技巧。

下节有这个方法的进一步的例子。

习题

1.〔22〕 正文说明了对于$T=5$和$S=16$的索贝尔交替排序。试给出一个算法的精确描述，该算法推广这个过程，对$T=P+1\geqslant 3$带上的$S=P^b$个初始路段进行排序。

力求找出能非常简单地加以描述的一个算法。

2.〔24〕 在索贝尔原先的方法中，如果$S=6$，则我们可以假托$S=16$并假设存在11个虚拟路段。这样在正文的例子中，阶段3将置诸虚拟路段A_0到T4和T5上，阶段4将把T2和T3上的诸A_1合并为T1上的一个D_2，阶段5～8将不做什么；而阶段9将在T4上产生A_5。更好的是在阶段3之后，就重绕T2和T3，然后通过三路合并立即在T4上产生A_6。

说明怎样来修改习题1的算法，使得当S不是P的完全乘方时，可得到与此类似的一个改进了的结果。

▶3.〔24〕 假定有九个初始路段，编制一张说明$T=3$时算法B之行为的图表。证明这个过程在一处显然是低效的，并建议对算法B的校正，使之弥补这一情况。

4.〔21〕 步骤B3置$A[l, q]$和$A[l, r]$成为-1。说明这两个操作之一总是多余的，因为对应的A表中的项是决不会被查找的。

5.〔M25〕 设S是出现在算法B的输入中的初始路段数，S的哪些值使得在步骤B2中不需要重绕？

5.4.6 关于带合并的实际考虑

麻烦的事情现在来了：至今我们已经讨论了各类合并型式，现在是来看看它们如何真正地应用于计算机和磁带的实际配置，并且以有意义的方式来比较它们的时候了。对于内部排序的研究表明，仅仅通过计算它执行的比较的数目，还不能适当地判断一个排序方法的效率；类似地，也不能仅仅通过知道它对于数据扫描的次数，就适当地评价一个外部排序方法。

在这一节里，将讨论典型磁带机的特征，及其对初始分布和合并的影响。特别是，我们将研究缓冲区分配的某些方案，及其对于运行时间的相应影响。我们也将简单地考虑排序生成器程序的构造。

带如何工作 不同的厂家所提供的磁带机的特征有很大不同。为方便起见，我们将定义一个假想的MIXT磁带机，对于在编写本书时，正被制造的设备来说，它是相当典型的。通过研究怎样来设计对于MIXT带的一个排序算法，就比较容易了解在其它具体的磁带机上应如何处理了。

MIXT可在每一英寸带上读和写800个字符，其速度为每秒75英寸。这意味着，当带运行时，每1/60毫秒，即$16\frac{2}{3}$微秒读或写一个字符。〔当前销售的实际磁带机的密度从每英寸200个字符到1600个字符不等，而其速度达每秒$37\frac{1}{2}$英寸到150英寸，所以它们的有效速度是MIXT的1/8倍到4倍。实际上，许多排序都是在比这里所考虑的速度要缓慢的、较小而又便宜的设备的商业环境中完成的。另一方面，磁带机也许不久就会急剧地变化，使得现在的假定过时。我们这里主要关心的，不是得到一个特定的答案；而是以一种合理的方式来学习如何把理论同实践结合在一起。〕

我们不能忘记的一个重要事实是：具体的带都不是无限长的。每卷带的长度不超过2400英尺。因此每卷MIXT带至多有23000000左右个字符的空间；而且读一遍大约花

23000000/3600000≈6.4 分钟。如果必须对更大的文件排序，则一般最好每次只排一整卷，而后再合并排好序的卷，以避免过多的带处理。这意味着，出现在我们已经研究过的合并型式中的初始路段数目 S 不可能太大。决不会有 $S>5000$，即使对于一个非常小的只产生 5000 个字符长的初始路段的内存也是如此。因此给出当 $S\to\infty$ 时诸算法的渐近效率公式主要在学术上有意义。

数据分块区记录在带上（图 79），而每个读/写指令传送一个块区。块区通常称为“记录”，但我们将避免使用这一术语，因为它同我们在对一个由“记录”组成的文件排序这句话中所表示的意义是相冲突的。在 50 年代所写的早期的许多排序程序中，这种区分是没有必要的，因为每个块区写一个记录；但是我们将看到，在带上的每个块区内多放几个记录通常是有利的。

在相邻的块区之间有一个 480 个字符位置长的块区间隔，为的是允许带在个别的读或写指令之间停止和启动。块区间隔使得每卷带的字符数减少，这依赖于每个块区的字符数（见图 80）；同样，每秒传输的平均字符数也减少了，因为带是以相当固定的速度移动的。

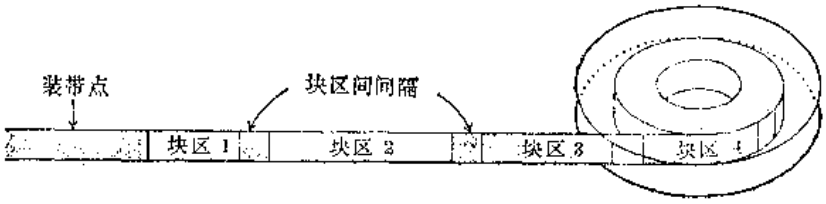


图79　带有可变大小的块区的磁带

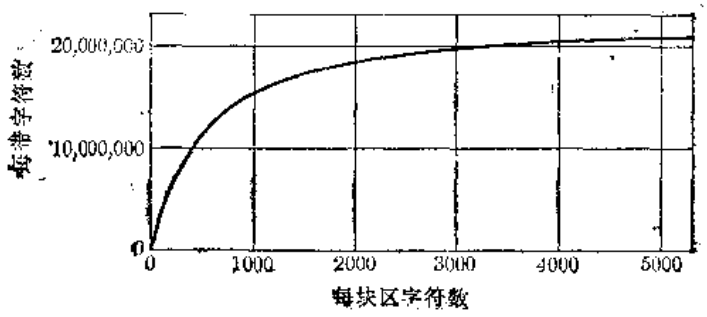


图80　作为块区大小的函数，每卷 MIXT 带的字符数

许多“旧式的”计算机都有相当小的固定的块区大小。例如，第一章中所定义的MIX总是读和写 100 字的块区；因为这表示每块区大约 500 个字符，而每个间隔 480 个字符，几乎浪费一半的带！现在大多数机器都允许块区大小可变，所以下面我们将讨论选择适当的块区大小的问题。

在一个读或写操作末了，磁带机以全速“滑过”间隔的头 66 个（左右的）字符。如果在这期间开始对同一条带作下一个操作，则带的运动无中断地继续下去。但如果下一个操作不能相当快地到来，则这条带将停止，而且它也需要一些时间来加速到全速以进行下一个操作。停启时间的延迟加在一起是 5 毫秒，停止 2 毫秒和启动 3 毫秒（见图 81）。因此，如果我们错过了继续全速读的机会，则对运行时间的影响实质上和在块区间隔中有 780 个字符（而不是 480 个字符）是一样的。

现在我们考虑重绕操作。可惜，对于给定的字符数 n，一般难于表征重绕所需的确切时间。在某些机器上，有一个“高速的重绕”，它仅当 n 大于 5 百万左右时才可以应用；对于较小的 n 值，重绕则以通常的读/写速度进行。在其它机器上，有一个特殊的发动机，所有重绕操作都使用它；它逐渐地加快带卷达到每分钟某一转速，然后当该停止时即自动刹

车。而实际的带速对于全卷都在变化。为了简便起见，我们将假定 MIXT 需要 max(30, n/150) 毫秒重绕 n 个字符位置（包括间隔在内），写它们大约要花两倍半的时间。这是对于许多实际磁带机行为的相当好的近似，在实际的磁带机中，读写时间同重绕时间之比一般都在 2 和 3 之间，但是这还未充分地模拟出在许多其它机器上出现的综合的低速和高速重绕的效果（见图 82）。

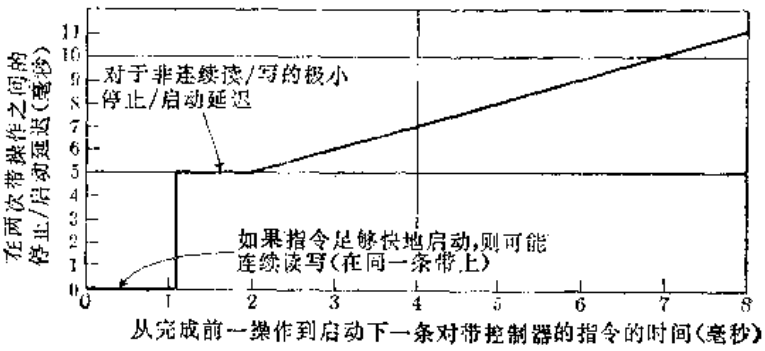


图81　怎样计算停止/启动延迟时间
（加到用于读或写块区及间隔的时间上）

初始的装带和/或重绕都把一条带定位在“装点”处，而在装点处开始的任何读或写操作都需要额外的 110 毫秒的时间。当这条带不在装点处时，可以向后读；在一个向前的操作之后的任何向后操作或者在一个向后操作之后的任何向前操作，都须附加 32 毫秒的时间。

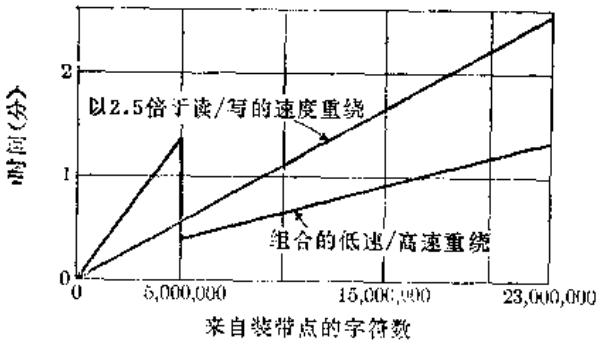


图82　对于两个通常使用的重绕技术的近似运行时间

最后，我们必须考虑输入和输出的同时性能被允许到什么程度。由于经济上的原因，常常把若干台磁带机接在一个带控制器上，在某个时刻里它仅能处理一台或两台磁带机，因为在它和计算机之间只有有限数目的数据通路。有时控制器没有能力在同一个时刻处理一台以上的磁带机，但它们通常却有能力在一台磁带机上读而同时在另一台磁带机上写。更稀少些的是能同时在两台磁带机上读的控制器，而作者从未见过可以同时在两台磁带机上写的控制器。重绕是一种特殊情况：在一个 MIXT 的重绕已经启动 30 毫秒之后，磁带机就同它的控制器“切断”关系了。这控制器又可以自由地同其它磁带机打交道了。这样一来，可以有大量的磁带机同时进行重绕。

几乎所有的机器都允许输入和输出同计算并发地处理，尽管许多计算机当输入/输出在进行时由于“周期共享”而运行得慢了 20% 到 40% 以上。

再论合并　现在让我们再次考察 P 路合并的处理，同时着重于考察输入和输出的活动，并假定 $P+1$ 台磁带机正在用于输入文件和输出文件。我们的目标是使输入/输出操作尽可能多地彼此重叠以及同程序的计算重叠，以使整个合并时间极小化。

考虑下列特殊情况是有教益的：在这种情况下，对于可能的同时性设置了严格的限制。假设

a）在任何一个时刻，至多一条带可写。

b）在任何一个时刻，至多一条带可读。

c）仅当读和写操作同时被启动后，读、写和计算可同时进行。

结果，尽管加上了这三个条件，$2P$个输入缓冲区和2个输出缓冲区的一个系统，就足以使带保持它实际上最快的速度来运行，除非计算机非常之慢。注意（a）实际上不是一个限制，因为仅有一条输出带。其次，输入的数量等于输出的数量，所以平均说来，在任何给定的时刻仅有一条带正在读；如果条件（b）不满足，则必定将有全然不出现输入的时期。于是，如果我们保持输出带忙碌，则就能使合并的时间极小化。

一项称为预测或预报的重要技术给了我们所希望的效果。在我们正在进行一个P路合并的同时，一般地有P个"当前的输入缓冲区"，它们被用作数据源：其中的某些比其余的更满些，这依赖于它们的数据已有多少被扫描。如果大约在同一时间它们全都变空了，则在可以往下进行之前，将需要大量地读，除非我们事先已经预测到这个意外事件。幸而，通过简单地考察在每个缓冲区中的最后一个记录，总有可能说出哪个缓冲区将首先变空，最后记录有最小键的缓冲区将总是头一个变空，不管任何其它键的值是什么。所以我们总是知道哪一个文件将是下一条输入指令的源文件。下列算法详细地叙述了这个原理。

算法F（浮动缓冲区的预报）本算法在$P\geqslant 2$时控制长输入文件的P路合并期间的缓冲区安排。假定输入带和文件编号成1，2，…，P，这个算法使用$2P$个输入缓冲区$I[1]$，…，$I[2P]$；两个输出缓冲区O[0]和O[1]；以及下列辅助表格：

A[j]，$1\leqslant j\leqslant 2P$：如果I[j]可利用作输入则为0，否则为1；

B[i]，$1\leqslant i\leqslant P$：包含迄今从文件i读入的最后块区的缓冲区；

C[i]，$1\leqslant i\leqslant P$：当前用来存放来自文件i的输入的缓冲区；

L[i]，$1\leqslant i\leqslant P$：迄今从文件i读入的最后一个键；

S[j]，$1\leqslant j\leqslant 2P$：当I[j]变成空时使用的缓冲区。

这里描述的算法并不终止，"终止它"的适当方式在以下讨论。

F1.〔初始化〕 对于$1\leqslant i\leqslant P$，从带i读头一个块区到缓冲区I[i]，置A[i]←1，A[$P+i$]←0，B[i]←i，C[i]←i，并置L[i]为在I[i]中最后一个记录的键，然后找m使得L[m]=min{L(1)，…，L(P)}；并置t←0，k←$P-1$，开始从带m读到缓冲区I[k]。

F2.〔合并〕 合并来自缓冲区I[C[1]]，…，I[C[P]]的记录到O[t]，直到O[t]满了为止。如果在这处理期间一个输入缓冲区，比如说I[C[i]]变空而O[t]仍未满，则置A[C[i]]←0，C[i]←S[C[i]]，并继续合并。

F3.〔完成输入/输出〕 等候直到以前的读（或读/写）操作完成。然后置A[k]←1，S[B[m]]←k，B[m]←k，并置L[m]成为I[k]中最后记录的键。

F4.〔预报〕 求m使得L[m]=min{L[1]，…，L[P]}，并求k使得A[k]=0。

F5.〔读/写〕 开始从带m读到缓冲区I[k]，并从缓冲区O[t]写到输出带，然后置t←$t-1$并返回到F2。▌

图84的例子示出了当带上的每个块区仅含两个记录，$P=2$时预报如何进行，图示的是每次我们达到步骤F2的开始处输入缓冲区的内容。算法F实际上形成P个缓冲区队，而以C[i]指向第i个队的前头，B[i]则指向队尾。同时，S[j]指向缓冲区I[j]的后继；这些指针如图84中的箭头所示，第1行说明了初始化之后的状态：对于

每个输入文件有一个缓冲区，而另一个块区正从文件1读入（因为03<05）。第2行示出在合并了第一个块区之后的状态：我们正在输出包含“01 02”的一个块区，并正在从文件2输入下一个块区（因为05<09）。注意，在第3行，四个缓冲区中的三个实际上都被文件2占用，因为我们正从该文件读，而且在它的队中已经有一个满缓冲区和一个部分满缓冲区。这种“浮动缓冲区”的安排是算法F的一个重要的特性，因为如果选择文件1而不是文件2作为第3行的输入，那么我们就会无法进行到第4行。

为了证明算法F是正确的，必须证明两件事情：

i）总存在一个输入缓冲区可以利用（即在步骤F 4中总能找到一个k）。

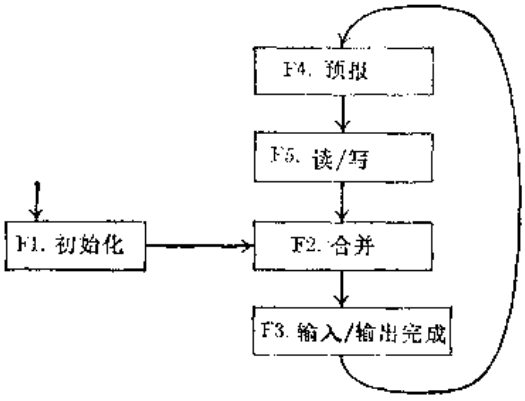


图83　浮动缓冲区的预报

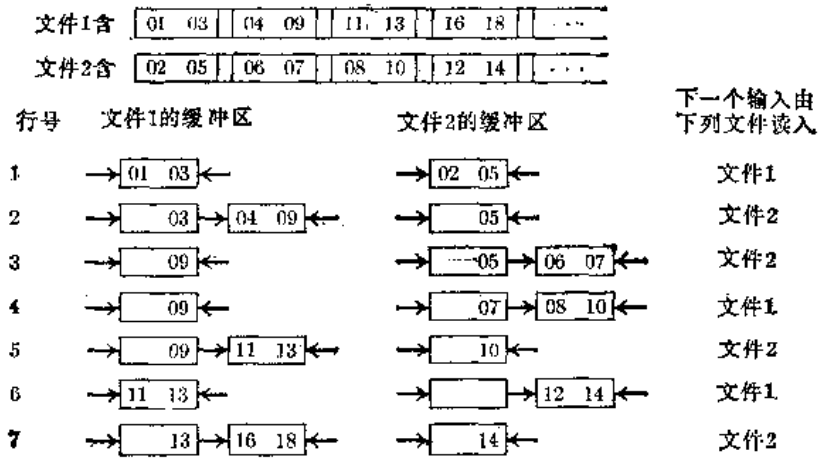


图84　按照算法F的缓冲区排队情况

ii）如果在合并时一个输入缓冲区被穷尽，则它的后继者已经出现在内存中（即在步骤F 2中的S[C[i]]是有意义的）。假设i）为假，于是在我们到达步骤F 4的某个时刻所有缓冲区都不能利用。每次到达该步骤时，在所有缓冲区中未处理的数据总数量恰恰是P个缓冲负载，即如果重新定位这些数据，则它们恰好足够填满P个缓冲区，因为我们正在以相同的速度输入和输出数据。某些缓冲区仅仅是部分地满了；但对于每个文件至多有一个缓冲区是部分地满的，所以至多有P个缓冲区是部分地满的。由假设，所有$2P$个缓冲区都不可利用，所以其中至少有P个必须是完全满的。这仅当P个是满的和P个是空的才会发生，否则我们将有太多的数据。但在任何一个时刻都至多只能有一个缓冲区是不可利用的和空的，因此（i）不能为假。

假设（ii）为假，于是我们在内存中没有对于某个文件的未处理的记录，但当前的输出缓冲区还不满。由预报原理，所有其它的文件不能有多于一个的数据块，因为我们只有

在任何其它文件上的缓冲区都被穷尽以前需要某个块区时才读入该块区。因此未处理的记录总共至多等于$P-1$个块区；加上未填满的输出缓冲区，将导出在内存中的数据量少于P个缓冲负载，矛盾。

这个论证确立了算法F的正确性，而且它也指出了出现病态情况的可能性，以及在这种情况下本算法尚能通过而不致出大问题。我们还未提及的一个重要奥妙，即有关相等的键的可能性，在习题5中讨论。

如果刚刚读入的块区是一个路段的最后块区，则巧妙地结束算法F的一种方式，是当刚才读入的块区是某一路段的最后一个块区时，在步骤F3中置$L[m]$为∞(通常都以某种特殊的方式来指出一个路段的结束)。在读了所有文件上的所有数据之后，我们将终于发现在步骤F4中所有的L都等于∞；在一般情况下，这时就可以开始读每个文件上的下一个路段的头一个块区，并随着最后的$P+1$个块区的输出，开始下一个合并阶段的初始化。

因此我们能使输出带实际上以全速进行，而无须在同一个时间读一条以上的带。在步骤F1中出现了对于这一规则的例外情形，那里为使诸事都从开始处进行，倒是一次读若干条带有利；但步骤F1通常都可安排成同计算的以前部分重叠。

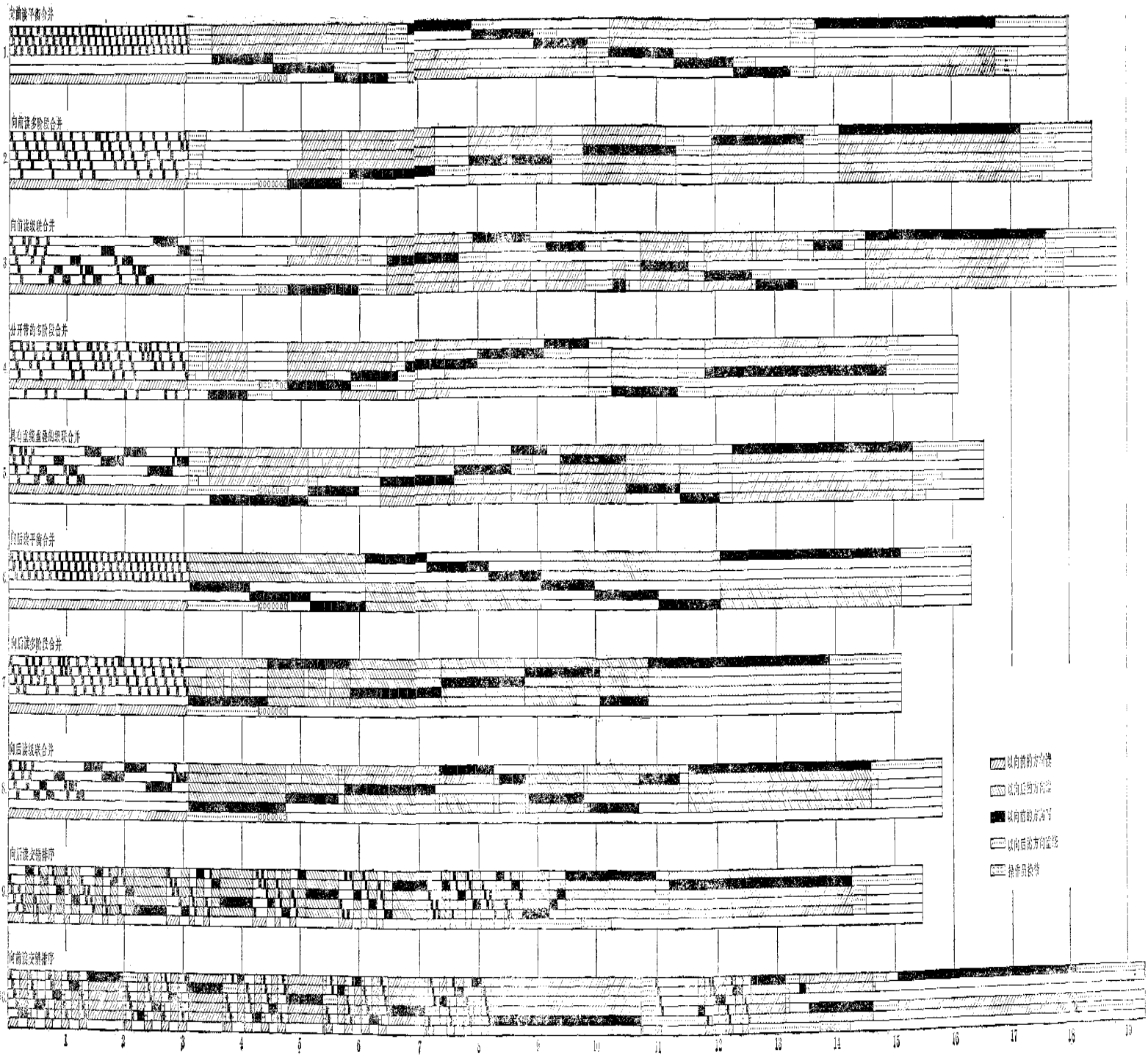
考察每个块区最后的记录，来预测哪一个缓冲区将首先变空的思想，是由F.E.霍尔伯顿(F.E.Holberton)于1953年发现的，这项技术首先由E.H.弗兰德发表[JACM3(1956), 144-145, 165]。他的颇为复杂的算法使用3P个输入缓冲区，每个输入文件专用三个缓冲区；算法F通过利用浮动缓冲区改善了这个状态，允许任何一个文件一次要求多达$P+1$个输入缓冲区，但总数决不多于2P个。

在这一节的末尾讨论了少于2P个输入缓冲区的合并。某些计算机有一种“分散读一集中写”的能力，它允许从内存中非连续的单元进行输入/输出；这样一个特性的含意已经超过了本书的范围。

合并型式的特性比较 现在让我们使用关于带和合并所知道的知识，来比较在5.4.2节到5.4.5节所研究的各种合并型式的有效性。当应用于一个特定的“无偏向”的例子时，给出每个方法的细节是非常有教益的，因此，我们就来考察一个文件的排序问题，该文件的每个记录包含100个字符，同时，内存中有100000个字符位置可以用来存储数据(不计程序和它的辅助变量，以及一个选择树中的链接所需要的相对少量的空间)。输入以随机次序出现在带上，每块区5000个字符，而输出以同样的格式出现。除了包含输入带的磁带机外，还有五条“空白带”可用。

有待排序的记录总数是100000，但排序算法事先并不知道这个信息。

图表A综述了把十个不同的合并方案应用到这批数据时发生的动作。考察这个重要的图解的最好方式是想象你正在观看排序的进行：从左到右缓慢地扫描每行，假想你真正能看六条带的读、写、重绕和/或向后读，如同在图式中所指出的那样。在P路合并期间，输入带被移动的次数将仅仅是输出带移动次数的$1/P$。注意，当原来的输入带已经完全读入（又已重绕和“锁带”）时，图A假定一个熟练的计算机操作员仅在30秒内卸下它并且以一条空白带代替它。在例2、3和4中，这是计算机空闲地等候操作员工作的“关键通路时间”；但在余下的例子中，卸带和重装操作是同其它处理重叠的。


向前读平衡合并
向前读多阶段合并
向前读级联合并
分开带的多阶段合并
具有重缓盖隧的级联合并
向后读平衡合并
向后读多阶段合并
向后读级联合并
向后读交错排序
向前读交错排序
以向前的方向读
以向后的方向读
以向前的方向写
以向后的方向写
排序的数字
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19

图表A 带的合并图解

例1. 向前读的平衡合并 让我们回顾这个问题的说明，记录的长度是 100 个字符，在每个时刻有足够的内存来保持1000个记录，而且在输入带上每个块区包含5000个字符（50个记录）。全部共有100000个记录（=10000000字符=2000个块区）。

对于中间文件我们自由地选择块区的大小，一个六带的平衡的合并使用 3 路合并，所以算法F的技术要求 8 个缓冲区，我们因此可以使用每个含1000/8=125个记录（=12500个字符）的块区。

初始分布扫描可以利用替换选择（5.4.1节），而且为了保持带光滑地运行，我们可以使用每个有50个记录的两个输入缓冲区，加上每个有125个记录的两个输出缓冲区，这就在替换选择树中保留了650个记录的空间，因此大多数初始路段大约将有1300个记录长(10或11个块区)，结果在图表A中产生了78个初始路段，最后一个要短些。

上述头一趟合并扫描中有九个路段合并到带 4，而不是在带 4、带 5 和带 6 之间交替进行。这使得在计算机操作员装入一条空白带到第 6 台磁带机上的同时有可能来做有用的工作；因为一旦已经完成了初始分布就知道了路段的总数，这个算法知道$\lceil S/9 \rceil$个路段应该被合并到带 4，然后$\lceil (S-3)/9 \rceil$个到带 5，然后$\lceil (S-6)/9 \rceil$个到带 6。

利用5.4.2节引进的记号，对于这个例子的整个排序过程可以下列方式综述如下：

1^{26}	1^{26}	1^{26}	—	—	—
—	—	—	3^9	3^9	3^8
9^3	9^3	$9^2 6^1$	—	—	—
—	—	—	27^1	27^1	24^1
78^1	—	—	—	—	—

例2. 向前读多阶段合并 图表A中的第二个例子按照算法5.4.2D进行多阶段合并。在此情况下，我们进行五路合并，使得存储器分成为每个有83个记录的12个缓冲区。在初始替换选择期间，我们有两个各50个记录的输入缓冲区和两个各83个记录的输出缓冲区，在树中保留 734 个记录；所以这次初始路段的长大约是1468个记录（17或18个块区）。所示的状态表明得到$S=70$个初始路段，最后两个实际上分别仅有四个块区和一个块区长。合并型式可综述为：

$0^{13}1^{18}$	$0^{13}1^{17}$	$0^{13}1^{15}$	$0^{12}1^{12}$	$0^8 1^8$	—
1^{15}	1^{14}	1^{12}	1^8	—	$0^8 1^4 2^1 5^3$
1^7	1^6	1^4	—	4^8	$1^4 2^1 5^3$
1^3	1^2	—	8^4	4^4	$2^1 5^3$
1^1	—	$16^1 19^1$	8^2	4^2	5^2
—	34^1	19^1	8^1	4^1	5^1
70^1	—	—	—	—	—

多奇怪，多阶段实际上比不甚复杂的平衡合并多花大约25秒钟！关于此，有两个主要的原因：

1）平衡合并在这种情况下特别走运，因为$S=78$是如此接近于 3 的一个完全乘方。如果产生的是82个初始路段，则平衡的合并将要花费一趟额外的扫描。

2）多阶段合并在换输入带时浪费了30秒钟，而且在它等候重绕操作完成的同时总共花去 5 分钟以上。对比之下平衡的合并需要比较少的重绕时间。在多阶段合并的第二个阶

段，由于可假定带 6 上的 8 个虚拟路段在该带重绕时即存在，因此节省了13秒，但是并不出现其它的重绕重叠，因此，多阶段法失败了，尽管它要求相当少的读写时间。

例3．向前读级联合并 这一情况类似于前例，但是使用算法5.4.3C。合并过程可综合如下：

1^{14}	1^{16}	1^{12}	1^{14}	1^{15}	—
1^{5}	1^{9}	—	1^{14}	1^{15}	$1^{8}2^{3}3^{6}$
$5^{1}6^{3}$	5^{8}	$5^{3}6^{2}$	—	1^{1}	2^{2}
—	12^{1}	6^{1}	18^{1}	18^{1}	16^{1}
70^{1}	—	—	—	—	—

（记住，通过扫描图表A，观看每一个例子的进行情况。）

例4．带分开的多阶段合并 在5.4.2节末尾描述的这一过程，允许重叠大多数重绕时间。它使用四路合并，所以我们把内存分成10个100个记录的缓冲区；在替换选择树中有700个记录，所以结果形成了72个初始路段。最后的路段又是非常短的。使用了一个类似于算法5.4.2D的分布方案，之后接着一简单但稍微特别的设置虚拟路段的方法：

1^{21}	1^{19}	1^{16}	1^{8}	—	$0^{2}1^{9}$
$0^{2}1^{17}$	$0^{2}1^{15}$	$0^{2}1^{11}$	$0^{2}1^{14}$	—	$0^{2}1^{9}4^{4}$
1^{13}	1^{11}	1^{7}	—	$0^{2}4^{4}$	$0^{2}1^{9}4^{4}$
1^{10}	1^{8}	1^{4}	—	$0^{2}4^{4}3^{2}4^{1}$	$1^{8}4^{4}$
1^{6}	1^{4}	—	4^{4}	$0^{2}4^{4}3^{2}4^{1}$	$1^{4}4^{4}$
1^{5}	1^{3}	—	$4^{4}3^{1}$	$0^{2}4^{4}3^{2}4^{1}$	$1^{2}4^{4}$
1^{2}	—	$3^{1}7^{2}$	$4^{4}3^{1}$	$4^{2}3^{2}4^{1}$	4^{4}
1^{1}	—	$3^{1}7^{2}13^{1}$	$4^{3}3^{1}$	$4^{1}3^{2}4^{1}$	4^{3}
—	13^{1}	$3^{1}7^{2}13^{1}$	$4^{2}3^{1}$	$3^{1}4^{1}$	4^{2}
—	$13^{1}14^{1}$	$7^{2}13^{1}$	$4^{1}3^{1}$	$3^{1}4^{1}$	4^{1}
18^{1}	$13^{1}14^{1}$	$7^{1}13^{1}$	3^{1}	4^{1}	—
18^{1}	14^{1}	13^{1}	—	—	27^{1}
—	—	—	72^{1}	—	—

结果表明，这是图表A中的所有不向后读的例子中最好的运行时间。因为 S 决不会达到非常之大，故有可能研究出一个更为复杂的算法，它以一个甚至更好的方式来设置虚拟路段（参见等式5.4.2-26）。

例5．具有重绕重叠的级联合并 尽管支配这个过程的算法要简单得多，它却几乎象以前的例子一样快地运行。对于初始分布，我们象在算法5.4.3C中那样简单地使用级联排序方法，但用 $T=5$ 来代替 $T=6$。然后每个"级联"的每个阶段错开诸带的运行，使得我们通常不写带，除非它已经得到机会被重绕过了。这个型式可简略地写为

1^{21}	1^{22}	1^{19}	1^{10}	—	—
1^{4}	1^{7}	—	—	$1^{5}2^{3}3^{5}$	4^{16}
7^{2}	—	8^{3}	$7^{2}8^{2}$	—	4^{1}
—	26^{1}	—	8^{1}	22^{1}	16^{1}
72^{1}	—	—	—	—	—

例6．向后读的平衡合并 这象例子1，但消去了所有的重绕：

A_1^{26}	A_1^{26}	A_1^{26}	—	—	—
—	—	—	D_3^9	D_3^9	D_3^8
A_9^3	A_9^3	$A_9^2A_6^1$	—	—	—
—	—	—	D_{14}^1	D_{27}^1	D_{27}^1
A_{78}^1	—	—	—	—	—

由于在例1中重绕比较少，这个方案不比向前读的情况好很多。事实上，尽管$S=78$是较好的值，但它实际比带分开的多阶段稍慢。

例7．向后读多阶段合并 在这个例子中，仅使用了六条带中的五条，为的是消去重绕时间和更换输入带。于是，仅仅使用了四路合并，而且缓冲区的分配就象在例4和例5那样。使用了象算法5.4.2D那样的分布，但路段的方向是交替的，带1固定为最后的输出带。首先把一个递增的路段写到带1上；然后把诸递减的路段写到带2，3，4上；然后把诸递增的路段写到带2，3，4上；然后把诸递减的路段写到带1，2，3上等等。每次转换方向时，替换选择通常产生一个更短的路段，所以结果形成了77个初始路段而不是在例4和例5中的72个。

这个过程得到了（22，21，19，15）个路段的一个分布，而下一个完全的分布是(29，56，52，44)。习题5.4.4-5注明了怎样生成合并数串，要使生成的数串可以用最优的方式放置虚拟路段；由于一个带卷的有限性，确保了S决不太大，所以这样一个过程在实际上是能行的。因此，图表A中的例子就是用这种设置虚拟路段的方法构造的（见习题7），在所有图解例子中这证明是最快的。

例8．向后读级联合并 如同在例7中那样，这里仅仅使用5条带。这个过程遵循算法5.4.3C，使用重绕和向前读来避免一路合并（因为重绕在MIXT磁带机上比读入要快两倍以上）。所以分布和例6一样。这个型式可以简单地综述如下，使用↓来表示重绕：

A_1^{21}	A_1^{22}	A_1^{19}	A_1^{10}	—
$A_1^4\downarrow$	$A_1^7\downarrow$	—	$D_1^3D_2^2D_3^5$	D_4^{10}
$A_8A_7^4$	A_5^3	A_9^4	—	$D_4^1\downarrow$
—	D_{17}	$A_9\downarrow$	D_{25}	D_{21}
A_{72}	—	—	—	—

例9．向后读交替排序 对于$T=5$的交替排序（算法5.4.5B）可以使用如同在例子4.5.7和8那样的缓冲区分配，因为它进行四路合并。然而，替换选择不以同样的方式进行，因为恰在进入每个合并阶段之前要输出一个长度为700的（而不是1400左右的）路段，为的是清内存。因此，在这个例子中，产生85个路段，而不是72个。在这个过程中的某些关键步骤是

—	A_1	A_1A_1	A_1A_1	A_1A_1
D_4	—	A_1	A_1	A_1
D_4D_4	D_4D_4	D_4D_4	D_4	—
D_4	D_4	D_4	—	A_{16}
D_4	$A_{16}D_4D_4$	$A_{16}D_4$	$A_{16}D_4A_1$	A_{16}
D_4	$A_{16}D_4D_4$	$A_{16}D_4A_1$	$A_{16}D_4$	A_{16}
—	$A_{16}D_4$	$A_{16}D_4$	A_{16}	$A_{16}A_{13}$
—	$A_{16}D_4$	A_{16}	$A_{16}A_4$	$A_{16}A_{13}$
—	A_{16}	$A_{16}A_4$	$A_{16}A_4$	$A_{16}A_{13}$
D_{37}	—	$A_{16}\downarrow$	$A_{16}\downarrow$	$A_{16}\downarrow$
—	A_{85}	—	—	—

例10. 向前读交替排序 在这个最后的例子中不使用替代选择，因为所有的初始路段都必须是相同长度的。因此每当需要一个初始路段时，占满整个内存的1000个记录被内部排序；这使得 $S=100$。在这过程中某些关键步骤是

A_1	A_1	A_1	A_1	A_1
—	—	—	—	A_1A_4
……	……	……	……	……
A_1	A_1	A_1	A_1	A_1A_4
—	—	—	A_1A_4	$\cancel{A_1}A_4$
—	—	—	A_1A_4	A_1A_4
……	……	……	……	……
A_1	A_1A_4	A_1A_4	A_1A_4	A_1A_4
A_1A_4	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$
$A_1A_4A_{16}$	—	—	—	—
……	……	……	……	……
—	A_1A_4	A_1A_4	A_1A_4	$A_1A_4A_{16}A_{64}$
A_4	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4$	$\cancel{A_1}A_4A_{16}A_{64}$
A_4A_{16}	—	—	—	$\cancel{A_1}\cancel{A_4}A_{16}A_{64}$
$\cancel{A_4}A_{16}$	A_4	—	—	$\cancel{A_1}\cancel{A_4}A_{16}A_{64}$
—	—	—	A_{36}	$\cancel{A_1}\cancel{A_4}\cancel{A_{16}}A_{64}$
A_{100}	—	—	—	—

这个例程乃是所有程序中最慢的，部分地是因为它不使用替换选择，但主要地是因为它的相当笨拙的结尾（一个两路合并）。

估计运行时间 现在让我们来看看如何算出使用 MIXT 带的一种排序方法的近似执行时间。如果不进行详细的模拟，我们也能预测图表A中的结果吗？

传统地用来比较不同的合并型式的一种方法是叠印图，如同我们在图70、74和78所看见的那样。在假定每个初始路段的长度大抵相同的条件下，这些图给出数据的有效扫描次数（作为初始路段个数的函数）(见图85)。但这并不是非常现实的比较，因为我们已经看到，不同的方法导致不同数目的初始路段；其次，由于块区间隔的疏密不同，“开销时间”也不一样，而且重绕时间也有相当大的影响。所有这些同机器有关的特性，使我们不可能编制出在独立于机器的基础上对诸方法进行正确比较的图表。另一方面，图85向我们指出，除了平衡的合并之外，扫描的有效次数可以通过形如 $\alpha \ln S+\beta$ 的光滑曲线相当好地逼近。因此，在任何特定的情况下，都可以通过研究运行时间的近似公式，来对各个方法进行相当好的比较。我们的目标当然是要找出简单而且充分现实的公式来。

现在我们借助于下列参数，尝试建立起这样的一些公式：

N = 有待排序的记录数，

C = 每个记录的字符数，

M = 在内存中可以利用的字符位置数（假定为 C 的倍数），

τ = 读或写一个字符的秒数，

$\rho\tau$ = 重绕一个字符的秒数，

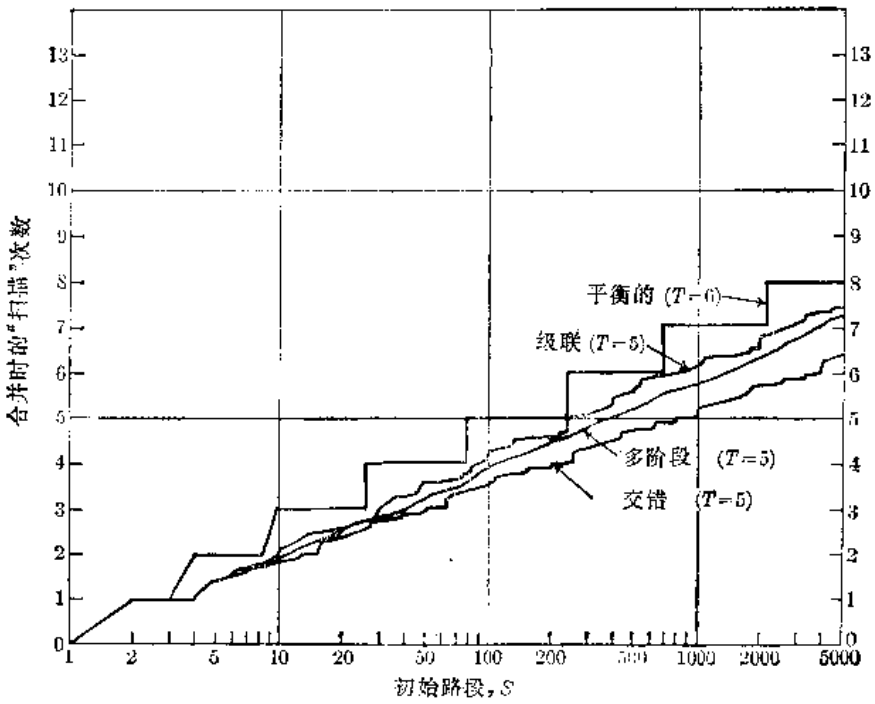


图85 比较合并型式的一种欠妥的方法

$\sigma\tau$＝停止-启动时间延迟的秒数，

γ＝每个块区间隔的字符数，

δ＝操作员为卸掉和替换输入带所需秒数，

B_i＝在未排序的输入中每个块区的字符数，

B_0＝在排好序的输出中每个块区的字符数。

对于MIXT我们有 $\tau=1/60000$，$\rho=2/5$，$\sigma=300$，$\gamma=480$。上面讨论的应用例子有 $N=100000$，$C=100$，$M=100000$，$B_i=B_0=5000$，$\delta=30$。这些参数通常都是影响排序时间的关键性的机器和数据特征（尽管重绕时间通常都是以比较复杂的一个表达式给出的，而不只是一个简单的比率 ρ）。给定了上述参数和一个合并型式，我们将进一步计算诸如下面的一些量：

P＝在这个型式中合并的极大阶数，

P'＝在替代选择树中的记录个数，

S＝初始路段个数，

$\pi=\alpha\ln S+\beta$＝读和写每个字符的近似平均次数，未计及初始分布或最后合并，

$\pi'=\alpha'\lg S+\beta'$＝在中间合并阶段对每个字符进行重绕的近似平均次数，

B＝在中间合并阶段每个块区的字符数，

ω_i，ω，ω_0＝"开销比"，读或写一个字符所需有效时间（由于间隔和停止/启动）除以硬件的时间 τ。

图表A的诸例子，已经按照公式

$$B=\left\lfloor\frac{M}{C(2P+2)}\right\rfloor C \qquad (1)$$

选出块区和缓冲区的大小，使得在同算法F的缓冲方案相一致的前提下能尽可能地大（为了避免在最后扫描期间的麻烦，P应该是够小，使得（1）能使$B \geqslant B_0$）。因此，在替代选择期间，树的大小是

$$P' = (M - 2B_i - 2B)/C \tag{2}$$

对于随机的数据，利用5.4.1节的结果，初始路段的个数S可以估计成

$$S \approx \left\lceil \frac{N}{2P'} + \frac{7}{6} \right\rceil \tag{3}$$

假定$B_i < B$，以及在分布期间输入带可以以全速进行（见以下），则大约花费$NC\omega_i\tau$秒来分布初始路段，其中

$$\omega_i = (B_i + \gamma)/B_i \tag{4}$$

在合并时，缓冲方案允许同时读、写和计算，但输入带之间频繁的转换意味着必须增加由停止/启动带来的时间花费；因此我们置

$$\omega = (B + \gamma + \sigma)/B \tag{5}$$

而合并时间近似于

$$(\pi + \rho\pi')NC\omega\tau \tag{6}$$

这个公式稍微地侵占了重绕时间，因为ω包括了停止/启动时间，但其它的考虑（诸如重绕的互锁及从装带点起读造成的额外花费）通常对此都能作出补偿。假定$B_0 \leqslant B$，则最后的合并扫描是通过开销比

$$\omega_0 = (B_0 + \gamma)/B_0 \tag{7}$$

来加以限制的。

我们可以估计最后的合并和重绕的运行时间为

$$NC(1 + \rho)\omega_0\tau$$

在实践中，由于存在不相等的块区长度，它可能要花稍微长些的时间（输入和输出不象在算法F中那样同步），但运行时间对于所有的合并型式都将大概相同。

在讨论对于个别型式的更专用的公式之前，让我们试图来论证上述所作的两个假定。

a）替代选择能够跟上输入带吗？在图表A的例子中，它大概能够，因为它大约花费算法5.4.1R的内循环的10次迭代来选择下一个记录，而且我们有$C\omega_i\tau > 1667$微秒来做这件事。只要在编替代选择循环的程序时多加斟酌，在许多机器上（但不是所有机器上）都是做得到的。注意，这一情况在合并时不是那么太关键：每个记录的计算时间，几乎总是小于在一个P路合并期间每个记录的带运行时间，因为P不太大。

b）如同在（1）中那样，我们果然应选择B为缓冲区的最大容量吗？一个很大的缓冲区会降低（5）中的开销比ω；但它也会增加内部路段S的个数，因为P'减小了。哪一个因素更重要，并不是一眼就能看出来的。把合并时间看成$x = CP'$的函数，我们可以把它表示成

$$\left(\theta_1 \ln\left(\frac{N}{x} + \frac{7}{6}\right) + \theta_2\right)\left(\frac{\theta_3 - x}{\theta_4 - x}\right) \tag{8}$$

其中θ_1、θ_2、θ_3、θ_4是某些适当的常数，且$\theta_3 > \theta_4$。求x的微商，即可看出存在某个N_0，使得对于所有$N \geqslant N_0$，牺牲缓冲区尺寸来增加x并不值得。例如，在图表A的排序应用中，

N_i大约等于10000；当对多于10000个记录排序时，大尺寸的缓冲区是有利的。

然而要注意，当S超过P的一个乘方时，平衡合并的扫描次数急剧跃升。如果预先知道N的一个近似，则缓冲区大小应该被选择成使得S尽可能地稍小于P的一个乘方。例如，图表A的头一行的缓冲区大小是12500；由于$S=78$，这是非常令人满意的，但如果S是82，则减少一点缓冲区尺寸将要好得多。

10个例子的公式 回到图表A，让我们试图给出一些公式，这些公式逼近十个方法中每个的运行时间。在大多数情况下，只要我们确定了中间合并扫描的次数$\pi=\alpha\ln S+\beta$和中间重绕扫描的次数$\pi'=\alpha'\ln S+\beta'$，则基本的公式

$$NC\omega_i\tau+(\pi+\rho\pi')NC\omega\tau+(1+\rho)NC\omega_0\tau \qquad (9)$$

是对于整个排序时间的充分好的近似。有时有必要对（9）附加进一步的校正；每一个方法的细节可以叙述如下：

例1. 向前读平衡合并 对于在$2P$条带上的P路合并可以使用公式$\pi=\lceil\ln S/\ln P\rceil-1$，$\pi'=\lceil\ln S/\ln P\rceil/P$。

例2. 向前读多阶段合并 我们可以取$\pi'\approx\pi$，因为每个阶段的后面通常都跟有大约和以前的合并长度相同的重绕。由表5.4.2-1我们得到，在六条带的情况下值$\alpha=0.795$，$\beta=0.864-2$（这“-2”是由于表中的项除了中间的扫描外，还包括初始和最后的扫描）。在初始分布之后重绕输入带的时间，即$\rho NC\omega_i\tau+\delta$，应该加到（9）上。

例3. 向前读级联合并 表5.4.3-1给出了值$\alpha=0.773$，$\beta=0.808-2$。估计重绕时间比较困难；或许置$\pi'\approx\pi$是足够精确的。如同例2一样，我们需加初始重绕时间到（9）上。

例4. 带分开多阶段合并 表5.4.2-5给出$\alpha=0.752$，$\beta=1.024-2$。除开初始化（$\rho NC\omega_i\tau+\delta$）和接近末尾的两个阶段（$2\rho NC\omega\tau$乘百分之36）外重绕时间几乎重叠。我们也可以从$\beta$减去0.18，因为头半个阶段为初始重绕所重叠。

例5. 具重绕重叠的级联合并 这里对于$T=5$我们使用表5.4.3-1，得到$\alpha=0.897$，$\beta=0.800-2$。几乎所有的非重叠的重绕都恰出现在初始分布之后和每两路合并之后，在一个完全的初始分布之后，最长的带包含数据的大约$1/g$，其中g是“增长率”。在每个两路合并之后，在六条带的情况下重绕的数量是$d_k d_{n-k}$（参考习题5.4.3-5），而且可以证明，在T条带的情况下，在两路合并之后的重绕量近似于该文件的

$$(2/(2T-1)(1-\cos(4\pi/(2T-1)))$$

在我们的情况下（$T=5$）这是该文件的$\frac{2}{9}(1-\cos 80^\circ)\approx 0.183$，而且它出现的次数是$0.946\ln S+0.796-2$。

例6. 向后读平衡合并 除大多数的重绕已被消去外，这和例1类似。从向前变成为向后这一方向变化，引起某些延迟，但是这些延迟并不重要。在最后扫描之前，将有50-50机会需要重绕，所以我们可以取$\pi'=1/(2P)$。

例7. 向后读多阶段合并 由于在这个情况下的替代选择产生大约每P次改变一次方向的路段，我们必须以S的另一个公式来替换（3）。一个相当好的近似（参考5.4.1-24）是$S=\lceil N(3+1/P)/6P'\rceil+1$。消去了所有的重绕时间，而表5.4.2-1给出$\alpha=0.863$，$\beta=0.921-2$。

例8. 向后读级联合并 由表5.4.3-1我们有$\alpha=0.897$，$\beta=0.800-2$。重绕时间可以估计成在该表中的"有复写的扫描"减去"无复写的扫描"后所得差额的两倍，假如重绕必须在最后的合并之前以得到递增的次序，则加上$1/(2P)$。

例9. 向后读交替排序 在这种情况下，替代选择要被启动和停止许多次；每次分布$P-1$到$2P-1$个路段，长度平均为P；因此，路段的平均长度近似于$P'(2P-4/3)/P$，故我们可以估计$S=\lceil N/((2-4/(3P))P')\rceil+1$。用一点时间来作从合并变为分布和从分布变为合并的转换；这近似于从输入带读入P'个记录的时间，即$P'C\omega_i\tau$，且它大约出现S/P次。重绕时间和合并时间可以象例6中那样估计。

例10. 向前读交替排序 这个方法不易于分析，因为在输入已穷尽后最后执行的"清除"阶段不象先前的一些阶段那样有效。若忽略这个麻烦的方面，只简单地要求作额外一次扫描，我们就可以通过置$\alpha=1/\ln P$，$\beta=0$，及$\pi'=\pi/P$来估计合并时间。在这种情况下路段的分布有些不同，因为未用替代选择；我们置$P'=M/C$及$S=\lceil N/P'\rceil$。注意，通过在开销中增加大约为$(M+2B)/M$的一个额外因子，有可能在分布期间细安排计算和读写，使它们重叠起来。在现在的情况下并不需要习题9中提到的"方式转换"时间，因为它已为重绕所重叠。所以在这个情况下估计的排序时间是（9）加上$2BNC\omega_i\tau/M$。

表1 排序的时间估计一览

例子	P	B	P'	S	ω	α	β	α'	β'	(9)	加到(9)中	估计数总和	实际数总和
1	3	12500	650	79	1.062	0.910	−1.000	0.303	0.000	1064		1064	1076
2	5	8300	734	70	1.094	0.795	−1.136	0.795	−1.136	1010	$\rho NC_{\omega i\tau}+\delta$	1113	1103
3	5	8300	734	70	1.094	0.773	−1.192	0.773	−1.192	972	$\rho NC_{\omega i\tau}+\delta$	1075	1127
4	4	10000	700	73	1.078	0.752	−0.944	0.000	0.720	844	$\rho NC_{\omega i\tau}+\delta$	947	966
5	4	10000	700	73	1.078	0.897	−1.200	0.173	0.129	972		972	992
6	3	12500	650	79	1.062	0.910	−1.000	0.000	0.167	981		981	980
7	4	10000	700	79	1.078	0.863	−1.079	0.000	0.000	922		922	907
8	4	10000	700	73	1.078	0.897	−1.200	0.098	0.117	952		952	949
9	4	10000	700	87	1.078	0.721	−1.000	0.000	0.125	846	$P'SC_{\omega i\tau}/P$	874	828
10	4	10000	—	100	1.078	0.721	0.000	0.180	0.000	1095	$2BNC_{\omega i\tau}/M$	1131	1158

表1表明，在这些例子中这些估计并不太坏，尽管在一些情况下，有五十秒左右的差异。例2和例3中的公式指出六条带时级联合并比多阶段还要好些，然而实际上却是多阶段更好。原因是象图85(它示出五条带的情况）这样的图象对于多阶段算法更接近于直线；如果有$14\leqslant S\leqslant 15$和$43\leqslant S\leqslant 55$，即近乎"完全"的级联数15和55，则在六条带上级联比多阶段更好，但算法5.4.2D的多阶段分布对于所有其它的$S\leqslant 100$同样好或更好。当$S\to\infty$时级联将优于多阶段，但S并不真正地趋于∞。例9中的估计不是也是由于类似的情况所致；多阶段优于交替，尽管渐近理论告诉我们，对于很大的S交替将是最好的。

杂记 现在宜于对带的合并来作一些随机的观察：

1）上述公式表明，排序实际上是N乘C的函数，而不是独立的N和C的函数。除了一些较小的考虑（例如把B取作C的一个倍数）外，我们的公式表明，对每个含10个字

符的 100 万个记录排序和对每个含 100 个字符的 10 万个记录排序，所花时间大约是同样的。虽然在公式中未予揭示，实际上却可能有差别，这差别是由于在替代选择期间为链接诸场所用的空间造成的。无论如何，键的大小很难铸成任何差别，除非键弄得那么长和那么复杂，以致内部计算已不能跟上带了。

对于长的记录和短的键，会诱人来“分离出”这些键，首先对它们排序，而后设法重新安排这些记录。但这个思想看来未必行之有效，它仅仅延迟了苦恼，因为最后的重排过程大约要花费和通常的合并排序相同的时间。

2）我们已经看到，在上述假定之下，为对 100000 个 100 个字符的记录排序，要花费 15 到 19 分的时间。但在一部卡片排序机上，做同样工作将花费多长的时间呢？这个问题是有实际意义的，因为卡片排序机比计算机要便宜。假定每个记录可以被压缩到一张 80列的卡片上，而字符键占六列，且在排序中每一列平均需要 $1\frac{2}{3}$ 次扫描，则我们必须在机器上运行每张卡片大约 10 次。以每分钟 1000 张卡片计，将花费 1000 分钟，这几乎是 17 个小时（在这期间，有相当的可能，某些卡片将在机器里偶而被洗乱或塞挤到机器中去）。

3）在编写一个有待重复使用的排序例程时，非常小心地来估计它的运行时间，并且把理论同实际观察到的性能进行比较，是明智的。由于排序理论已经相当成熟，因此也就知道这个过程会显露出现存系统上输入/输出硬件或软件的缺陷；但服务工作慢于它应该有的速度，而且直到排序例程运行得实在太慢时，方才引起人的注意！

4）某些计算机系统有两“组”磁带机，它们以这样的方式接到分开的“通道”上，即仅对属于不同组的带，允许同时进行读和写。平衡合并特别适合于这样的配置。例如，考虑每组各有三条带的六条带的情形，并假设我们要对 $T=6$ 进行多阶段合并。在一个五路合并期间，两条输入带将在错误的组里，所以大约有五分之二的输入时间不能同输出重叠。这近似地增加了百分之四十的排序时间，使得甚至在向后读的情况下，平衡合并结果比多阶段更好。

5）我们已经对“随机”文件进行过替代选择的分析，但在实践中真正出现的文件经常都有大量的现存的次序（事实上，有时人们确实在对已经有次序的文件排序）。因此经验证明，替代选择比其它类型的内部排序更可取，这比我们公式所指出的还要突出。在向后读多阶段排序的情况下，这个优点不太明显，因为必须产生一些递减的路段；确实，R. L. 吉尔斯塔德（他首先发表了多阶段合并）原来由于这个原因而拒绝向后读的技术。但他后来注意到，交替的方向仍将挑出长的递增路段。而且向后读多阶段乃是仅有的既适用于递减输入文件也适用于递增输入文件的标准技术。

6）替代选择的另一个优点是它允许同时进行读、写和计算。如果我们仅仅以一种显然的方式进行内部排序——填满内存，对它进行排序，然后当开始装入下一个负载时写出它——则分布扫描将大约花费两倍之长的时间！

我们已经讨论过的，唯一适合于同时读、写和计算的其它内部排序，就是堆排序（这个思想已经用于编制图表A的例 10）。为了方便起见，假设内部存储保持 1000 个记录，并假设在带上每个块区保持 100 个。命 $B_1B_2\cdots B_{10}$ 表示分成 10 个块区，每个块区各 100 个记录的内存的内容，我们可以进行如下：

步骤 0　填满内存，并使 $B_1 \cdots B_{10}$ 的元素满足堆（在根处具有最小元素）的诸不等式。

步骤 1　使 $B_1 \cdots B_{10}$ 成为一个堆，然后选择出最小的 100 个记录，并且把它们移到 B_{10}。

步骤 2　写出 B_{10}，同时选择 $B_1 \cdots B_9$ 的最小的 100 个记录，并把它们移到 B_9。

步骤 3　读进 B_{10}，并写出 B_9，同时选择 $B_1 \cdots B_8$ 的最小的 100 个记录，并把它们移到 B_8。

……

步骤 9　读进 B_4，并写出 B_3，同时选择 $B_1 B_2$ 的最小的 100 个记录，并把它们移到 B_2，同时使堆的诸不等式在 $B_5 \cdots B_{10}$ 中成立。

步骤10　读入 B_3，并写出 B_2，同时对 B_1 排序并使堆的诸不等式在 $B_4 \cdots B_{10}$ 中成立。

步骤 11　读入 B_2，并写出 B_1，同时使堆的诸不等式在 $B_3 \cdots B_{10}$ 中成立。

步骤 12　读入 B_1，同时使堆的诸不等式在 $B_2 \cdots B_{10}$ 中成立，返回步骤 1。

7）我们已经假定要排序的记录的个数事先是未知的。实际上在大多数计算机应用中，有可能始终掌握住全部文件中记录的个数，且可以假定计算机系统有能力告诉我们 N 的值。这将有多大的帮助呢？可惜，不很多！我们已经看到，替代选择是非常有利的，但是它导致了一个不可预测的初始路段个数。在一个平衡合并中，我们可以使用关于 N 的信息，以这样的方式来设置缓冲区的大小 B，即使得 S 大概将恰恰小于 P 的一个乘方；而在具有最优的虚拟路段设置的多阶段分布中，我们可以使用关于 N 的信息，来判断使用什么级别最好（参考表 5.4.2-2）。

另一方面，不使用替代选择，我们可以应用在 5.4.4 节结尾处描述的先序合并，使它成为在适当的时刻以适当的方向分布初始路段的一个交替排序（而不是象在那里描述的从"带 A"来取它们）。在所有不使用替代选择进行内部排序的方法当中，这个方法实际上是最优的，因为它按照最好可能的 P 叉树来合并；但它实际上比以替代选择为基础的方法运行要慢得多。

8）磁带机看来是计算机最不可靠的部分。典型地说，维修工大部分是在维修磁带机而不是机器的任何其它部件，并且计算机操作员必须学会如何在一条带失误之后重新开始作业。作者还未曾见到过一个计算站有十台或更多的磁带机都能同时很好地工作的！因此，直到知道整个排序已经满意地完成之前，原来的输入带决不应被破坏。这应该认为是公理。"操作员的卸带时间"是图表 A 的一些例子中的烦恼，但是，鉴于在一个长排序期间某些事情可能出错，把输入抹掉是太冒险了。

9）当从向前写变成向后读时，只要不把最后的缓冲负载写到带上，我们可以节省某些时间；因为它总是仍旧要读进来的！图表 A 表明这项技巧实际上节省的时间比较少，除非是交替排序，在那里经常颠倒方向。

10）如果我们有大量的磁带机可利用，那种尽可能多地使用它们以便得到一个"高阶合并"的方法，并不总是最好的。例如，一个更高阶的合并蕴涵着一个较小的块区大小；而且当 P 很大时，在 $\log_P S$ 和 $\log_{P+1} S$ 之间的百分比之差，并不是很大的。还要考虑到可怜的计算机操作员，他必须安装所有那些空白带。另一方面，习题 12 描述了利用附加的

磁带机的一种有趣的方式，即把它们组合在一起以便重叠输入/输出时间而不必增加合并的阶。

11）象 MIX 这样的机器上，它们都有固定的颇小的块区大小，在合并时几乎都不需要使用任何内存。于是交替排序变得更有吸引力，因为在合并时就可能保持替代选择树于内存中。事实上，在这种情况下，我们可以（如科林·J.贝尔（Colin J.Bell）在 1962 年所提议的）改进交替排序，每次从工作带合并时，同时合并一个新的初始路段到输出中。

12）我们已经观察到，多卷文件每次只应该对一卷排序，以避免过多的带处理。这有时称为“卷时间”应用。实际上，如果比较细心地编程序，则在六条带上的一个平衡排序，能够同时对三个满卷排序，直到最后的合并时间为止。

为了对相当大量的已经各自排好序的磁带卷进行排序，极小路径长度的合并树将是最快的（参考 5.4.4 节）。这个构造首先是由 E. H. 弗兰德作出的〔*JACM* 3(1956)166-167〕。然后W. H. 伯奇〔*Information and Control* 1 (1958)181-197〕指出，如果我们忽略带的处理时间，则合并给定（可能不等）长度路段的一种最优方式，可通过构造用路段长度做为权，具有极小加权路径长度的一株树来得到（参考 2.3.4.5 节和 5.4.9 节）。但是多卷文件大概应该保持在一个磁盘上或其它的大存储器上，而不是在带上。

13）我们的讨论已经冒失地假定我们有对磁带机输入/输出指令的直接控制，而且没有复杂的系统接口来阻止我们象带的设计者所希望的那样有效地使用带。这些理想的假定，使我们洞察了带合并的问题，而且还可以使我们对于操作系统的正确设计有所了解，但应认识到，多道程序设计和多重处理可使情况更为复杂。

14）这一节所研究的论题，是由 E. H. 弗兰德〔*JACM* 3 (1956), 134-168〕, W. 佐贝尔比尔（W. Zoberbier)〔*Elektron. Datenverarb.* 5 (1960), 28-44〕以及M. A. 戈茨〔*Digital Computer User's Handbook* (New York: McGraw-Hill, 1967) 1.292-1.320〕首先在印行的著作中讨论的。

小结 我们可以以如下方式总结在比较各种不同的合并型式中所学到的东西。

定理 A 难于判断哪个合并型式在一个给定的情况下是最好的。

在图表 A 中已经看到的诸例说明，100000 个随机地排序的 100 个字符的记录（或 1 百万个 10 个字符的记录），在现实的假定之下，可以怎样利用六条带来进行排序。这么多数据填满大约一半的带，而且它能在大约 15 分到 19 分钟内进行；然而可利用的磁带机有相当大的差异，而且这样一个作业的运行时间，在不同的机器上，可以小到 4 分钟，大到两小时。在我们的例子中路段的初始分布和内部排序使用了大约 3 分钟时间；最后的合并和重绕输出带使用大约 $4\frac{1}{2}$ 分钟；而合并的中间阶段花费大约 $7\frac{1}{2}$ 到 $11\frac{1}{2}$ 分钟。

给定不能向后读的六条带，在我们的假定之下，最好的排序方法是“带分开的多级合并”（例 4）；而对于允许向后读的带，最好的方法乃是具有一个复杂的虚拟路段设置的向后读多阶段法（例 7）。交替排序（例 9）仅次于它。在这两种情况下级联合并提供了一个更简单的方案，它仅仅稍微慢些（例 5 和 8）。在向前读的情况下，一个直截了当的平衡合并（例 1）惊人地有效，部分地是因为这个特定例子对它说是一种幸运，但部分地也是因为它花费较小的重绕时间。

如果我们有不同数目的可利用的带，则情况将稍有改变。

排序生成器 由于数据和装备特征是千变万化的，几乎不可能写一个单一的外部排序程序，使它在非常多的应用中都是令人满意的。而且也颇难编写能真正有效地处理带的一个程序。因此排序软件的编制是一项特别有挑战性的工作。一个排序生成器是这样一个程序，它根据描述数据格式和硬件配置的参数，产生出适合于某类特定用途的机器代码。这样一个程序通常都依赖于诸如COBOL或PL/1这样的高级语言，或者它可以写成通过宏汇编程序加以使用的一组宏程序。

一个排序生成器通常提供的特性之一，是插入“特种编码”的能力，这是有待加入到排序例程的第一遍和最后一遍扫描的特殊指令。头一遍扫描的特种编码常常用来编辑输入记录，通常把它们缩小或扩展成为易于排序的形式。例如，假设输入记录按一个九字符的键来进行排序，该键以月一日一年的格式表示日期：

JUL041776 OCT311517 NOV051605 JUL141789 NOV201917

在头一次扫描时，三个字母的月代码可在一份表中查出，而月代码可以用数来代替，其中最高有效字段在左边：

17760704 15171031 16051105 17890714 19171120

这就减小了记录的长度并且使随后的比较要简单得多（还可以换成一个甚至更紧凑的代码）。最后扫描的特种编码，则可用来恢复原来的格式，和/或对文件进行其它希望的变动，和/或计算输出记录的某个函数。我们已经研究过的合并算法，是以这种方式组织的，即易于把最后的扫描同其它合并阶段区别开来，注意，当出现特种编码时，必须至少对文件进行两次扫描，即使它开始时已经是有序的亦然。改变记录大小的特种编码，可使交替排序难以使它的某些输入/输出操作相重叠。

排序生成器也照顾到诸如磁带标识的约定等系统的细节，而且它们通常提供“杂凑总和”或其它校验以确保不丢失或改变数据。有时还有在适当的位置停止排序及过后恢复执行的规定。最优秀的生成器允许记录有动态变化的长度〔参考D. J. Waks, *CACM* 6 (1963), 262-272〕。

***使用较少缓冲区的合并** 我们已经看到，在P路合并期间，为跟上带的急速移动，$2P+2$个缓冲区已足够了。现在在结束本节之前，让我们从数学上来分析一下在少于$2P+2$个缓冲区时的合并时间。

两个输出缓冲区显然是可取的，因为我们可以在从一个中写出的同时在另一个中形成下一个输出块区。因此我们完全可以忽略输出的问题，而仅仅专注于输入。

假设有$P+Q$个输入缓冲区，其中$1 \leqslant Q \leqslant P$。如同L. J. 伍德隆姆所提议的，我们将使用这种状态的下列近似模型〔*IBM System J.* 9 (1970), 118-144〕：每读带上一个块区需用一个时间单位。在这时间里，没有输入缓冲区变成空的概率为p_0，有一个变成空的概率为p_1，有两个或更多变成空的概率为$p \geqslant 2$，等等，当完成一条带的读入时，我们处于$Q+1$个状态之一：

状态0 Q个缓冲区为空；我们使用在这节中早些时候说明的预报技术，开始从适当的文件读一个块区进入Q个缓冲区之一。在一个时间单位之后，我们以概率p_0转到状态1，否则我们留在状态0。

状态 1 $Q-1$ 个缓冲区为空；预报适当的文件，我们开始读入 $Q-1$ 个缓冲区之一。在一个时间单位之后，我们以概率 p_0 转到状态 2，以概率 p_1 转入状态 1，以概率 $p_{\geqslant 2}$ 转入状态 0。

⋮

状态 $Q-1$ 一个缓冲区为空；预报适当的文件，我们开始读入此缓冲区。在一个时间单位之后，我们以概率 p_0 转到状态 Q，以概率 p_1 转到状态 $Q-1$，…，以概率 p_{Q-1} 转到状态 1，和以概率 $p_{\geqslant Q}$ 转到状态 0。

状态 Q 所有缓冲区都被填满。读带停止平均 μ 个时间单位，而后转到状态 $Q-1$。

我们以状态 0 开始。这个状态的模型对应于一个马尔科夫过程（参考习题 2.3.4.2-26），它可以以下列有趣的方式通过母函数来进行分析：设 z 是一个任意参数，而且假定，每当我们有读带机会时，决定读带的概率是 z，而决定结束算法的概率是 $1-z$。现在命 $g_Q(z)=\sum_{n\geqslant 0}a_n^{(Q)}z^n(1-z)$ 是在这样一个过程中出现状态 Q 的平均次数；由此得出 $a_n^{(Q)}$ 是当恰好读了 n 个块区时状态 Q 出现的平均次数。于是 $n+a_n\mu$ 是输入＋计算的平均总时间。如果如同在（$2P+2$）个缓冲区的算法中那样，我们有完全的重叠，则总共的时间将仅仅是 n 个单位，所以 $a_n\mu$ 表示“读挂起”的时间。

对于 $0\leqslant i$，$j\leqslant Q+1$，命 A_{ij} 是在这过程中我们从状态 i 转到状态 j 的概率，其中 $Q+1$ 是一个新的“被停止”状态。例如，对于小的 Q，A 矩阵取如下的形式：

$$Q=1:\begin{pmatrix} p_{\geqslant 1}z & p_0z & 1-z \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

$$Q=2:\begin{pmatrix} p_{\geqslant 1}z & p_0z & 0 & 1-z \\ p_{\geqslant 2}z & p_1z & p_0z & 1-z \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$Q=3:\begin{pmatrix} p_{\geqslant 1}z & p_0z & 0 & 0 & 1-z \\ p_{\geqslant 2}z & p_1z & p_0z & 0 & 1-z \\ p_{\geqslant 3}z & p_2z & p_1z & p_0z & 1-z \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

习题 2.3.4.2-26 b 告诉我们 $g_Q(z)=$ 余因式$_{Q_0}(I-A)/\det(I-A)$。于是例如当 $Q=1$ 时，我们有

$$g_1(z)=\det\begin{pmatrix} 0 & -p_0z & z-1 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}\Bigg/\det\begin{pmatrix} 1-p_{\geqslant 1}z & -p_0z & z-1 \\ -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$=\frac{p_0z}{1-p_1z-p_0z}=\frac{p_0z}{1-z}=\sum_{n\geqslant 0}np_0z^n(1-z)$$

所以 $a_n^{(1)}=np_0$，这当然是早就能想得到的，因为当 $Q=1$ 时问题是很简单的。当 $Q=2$ 时（见习题 14）类似的计算给出不大显然的公式

$$a_n^{(2)}=\frac{p_0^2n}{1-p_1}-\frac{p_0^2(1-p_1^n)}{(1-p_1)^2} \tag{10}$$

一般地说，我们可以证明，当 $n\to\infty$ 时 $a_n^{(Q)}$ 的形式是 $\alpha^{(Q)}n+O(1)$，其中常数 $\alpha^{(Q)}$ 不是太难计算的（见习题 15）。结果是 $\alpha^{(3)}=p_0^3/((1-p_1)^2-p_0p_2)$。

鉴于合并的本性，我们有相当的理由来假定 $\mu=1/P$ 以及一个二项式分布

$$p_k=\binom{P}{k}\left(\frac{1}{P}\right)^k\left(\frac{P-1}{P}\right)^{P-k}$$

例如，当 $P=5$ 时，我们有 $p_0=.32768$，$p_1=.4096$，$p_2=.2048$，$p_3=.0512$，$p_4=.0064$，以及 $p_5=.00032$；因此 $\alpha^{(1)}=0.328$，$\alpha^{(2)}=0.182$，以及 $\alpha^{(3)}=0.127$。换句话说，如果我们使用 5 + 3 个输入缓冲区代替 5 + 5 个输入缓冲区，则可以预期一个额外的大约 0.127/5 ≈2.5%的“读挂起”时间。

当然，这个模型仅仅是一个非常粗略的近似，我们知道，当 $Q=P$ 时，全然没有挂起时间，但这个模型说有。对于较小的 Q，额外的读挂起时间大约恰恰抵消通过较大的块区所节省的开销，所以简单的 $Q=P$ 的方案似乎是合理的。

习题

1.〔13〕 假设在没有块区间的间隔时，每条带正好包含 23000000 个字符，求当带上的每个块区包含 n 个字符时，每条带的准确字符数的一个公式。

2.〔15〕 说明为什么图 84 的第 6 行中文件 2 的头一个缓冲区完全是空的？

3.〔20〕 如果仅有 $2P-1$ 个输入缓冲区而不是 $2P$ 个，则算法 F 能正常工作吗？若能，则证明之；若不能，则给出一个使它不对的例子。

4.〔20〕 如何改变算法 F，使得当 $P=1$ 时它也能够工作？

▶5.〔21〕 当等键出现在不同的文件中时，在预报过程中就必须非常小心。说明为什么，并通过更精确地定义算法 F 的合并和预报操作来说明怎样克服这个困难？

6.〔22〕 为了把算法 5.4.3C 转换成对于 $T+1$ 条带的具有重绕重叠的级联合并的一个算法，对算法 5.4.3C 应该做些什么改动？

▶7.〔26〕 图表 A 例 7 中的初始分布产生

$$(A_1D_1)^{11}\quad D_1(A_1D_1)^{10}\quad D_1(A_1D_1)^{9}\quad D_1(A_1D_1)^{7}$$

于带 1-4 上，这里 $(A_1D_1)^7$ 指的是

$A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1A_1D_1$。说明如何以“最好”的方式来插入诸 A_0 和 D_0（所谓最好，指的是在合并时所处理的初始路段总数为极小），使分布变成为

$$A(DA)^{14}\qquad (DA)^{25}\qquad (DA)^{26}\qquad (DA)^{22}$$

提示：为了保持奇偶性，有必要把许多 A_0 和 D_0 作为相邻的对偶插入。每个初始路段的合并数，可以象在习题 5.4.4-5 中那样来计算；由于相邻的路段总有相邻的合并数，就会出现某些简化。

8.〔20〕 图表 A 说明，路段初始分布的大多数方案（对于级联合并的初始分布例外），都趋向于把相继的路段置到不同的带上。如果相继的路段放到相同的带上，则我们可以节省停止/启动时间；如果因此而修改分布算法，以减少带的转换次数，是一个好的想法吗？

▶9.〔22〕 如果我们对于排序都用$T=6$条带，而不是象在例7中那样用$T=5$条带，试估计在图表A中的向后读多阶段算法将会多长，避免使用输入带是明智的吗？

10.〔$M23$〕 使用5.4.2和5.4.3节的分析，证明在一个标准的六条带的多阶段或级联合并期间，每次重绕的长度很少超过文件的大约54%(初始和最后的重绕除外，它们席卷整个文件)。

11.〔23〕 通过修改表1中适当的项，试估计，如果我们有一个联合的低速/高速重绕，则图表A的头九个例子将花费多长时间？假定带未装满四分之一时$\rho=1$，并假定更满的带的重绕时间近似于五秒加上当$\rho=\frac{1}{5}$时花费的时间。试改变例8，使得它使用有复写的级联合并，因为在这种情况下重绕和向前读比复写更慢〔提示：使用习题10的结果〕。

12.〔40〕考虑把六条带划分成三对带，每对在$T=3$的多阶段合并中起着一条带的作用。每对的一条带将包含块区1，3，5…而另一条带将包含块区2，4，6，…；这样一来，合并时，实质上我们总有两条输入带和两条输出带在活动，这等效于加倍了合并的速度。(a)试找出一个适当的方式把算法F推广到这一情况。应有多少个缓冲区？(b)如果用这个方法对100000个100个字符的记录排序，考虑向前读和向后读两种情况，试估计总共的运行时间将是多少？

13.〔12〕 按照算法5.4.5B中的定义，一个五带交替排序可否用来对四个满卷的输入数据排序，直到最后合并时为止？

14.〔19〕 推导(10)。

15.〔$HM29$〕 证明$g_Q(z)=h_Q(z)/(1-z)$，其中$h_Q(z)$是单位圆里边没有奇点的z的一个有理函数；因此当$n\to\infty$时，$a_n^{(Q)}=h_{(Q)}(1)n+O(1)$，特别证明

$$h_5(1)=\det\begin{pmatrix}0 & -p_0 & 0 & 0\\ 0 & 1-p_1 & -p_0 & 0\\ 0 & -p_2 & 1-p_1 & -p_0\\ 1 & 0 & 0 & 0\end{pmatrix}\Big/\det\begin{pmatrix}1 & -p_0 & 0 & 0\\ 1 & 1-p_1 & -p_0 & 0\\ 1 & -p_2 & 1-p_1 & -p_0\\ 0 & 0 & -1 & 1\end{pmatrix}$$

16.〔$M46$〕 利用一个更精确的模型，比正文中更仔细地分析具$P+Q$个输入缓冲区的合并。

17.〔41〕假定可利用3、4或5条带时，象在图表A中那样画出图表，详细研究对100,000个100字符的记录的排序问题。

*5.4.7 外部基数排序

以前诸节已经讨论了通过合并进行带排序的过程；但是还有另外一种用带进行排序的方式，它的依据是机械的卡片排序机（参考5.2.5节）所用的基数排序原理。这个方法有时称为分布排序、列排序、袋排序、数字排序、分开排序等；可以证明，它实际上恰与合并相对立！

例如，假设有四条带和仅有八种可能的键：0，1，2，3，4，5，6，7。如果输入数据在带1上，则开始我们可以把所有的偶键传送到T3，把所有的奇键送到T4：

	T1	T2	T3	T4
给定	{0, 1, 2, 3, 4, 5, 6, 7}	—	—	—
第1趟扫描	—	—	{0, 2, 4, 6}	{1, 3, 5, 7}

现在我们重绕，并读T3而后读T4，把{0, 1, 4, 5}置于T1上，{2,3,6,7}置于T2上：

第2趟扫描	{0, 4}{1, 5}	{2, 6}{3, 7}	—	—

(记号"{0, 4}{1, 5}"表示一个文件，这个文件包含某些记录，它们的键都是0或4，后边跟着键都是1或5的记录。注意，T1现在包含的诸键的中间二进数字均是0）在再次重绕和分布0，1，2，3到T3以及4，5，6，7到T4之后，有

第3趟扫描			{0}{1}{2}{3}	{4}{5}{6}{7}

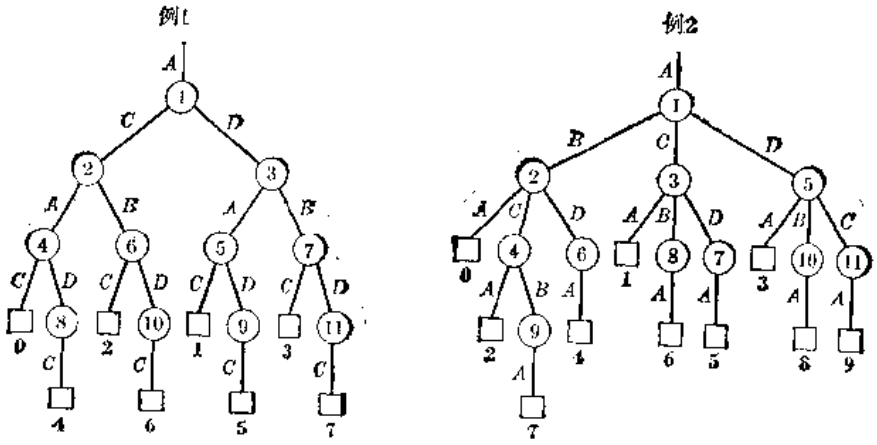
现在我们可以通过复写T4到T3的末尾来结束。一般地，如果键的范围是从0到2^k-1，则我们可以利用k趟扫描，以类似的方式对文件排序，后边跟之以最后的"收集"阶段，它从一条带复写大约一半的数据到另一条带。若用六条带，则可以以类似的方式，使用基数3的表示，在k趟扫描中对从0到3^k-1的键排序。

也可以使用部分扫描的方法。例如，假设有十种可能的键{0，1，…9}，考虑由R. L. 阿申赫斯特（R. L. Ashenharst)给出的下列过程〔Theory of Switching 7 (Harvard Univ. Comp. Laboratory: May, 1954)，Ⅰ.1-Ⅰ.76〕：

	T1	T2	T3	T4	
给定	{0, 1, …, 9}	—	—	—	
阶段1	—	{0, 2, 4, 7}	{1, 5, 6}	{3, 8, 9}	1.0趟
阶段2	{0}	—	{1, 5, 6}{2, 7}	{3, 8, 9}{4}	0.4趟
阶段3	{0}{1}{2}	{6}{7}	—	{3,8,9}{4}{5}	0.5趟
阶段4	{0}{1}{2}{3}	{6}{7}{8}	{9}	{4}{5}	0.3趟
收集	{0}{1}{2}{3}{4}…{9}				0.6趟
					2.8趟

如果每种键值出现的机会大约是十分之一，则上述过程只化费2.8趟扫描来对十个键排序，而头一个例子则需要3.5次扫描来对仅仅八个键排序。因此我们发现，无论是基数排序还是合并，由于所使用的分布型式巧妙不同，其结果可以是很不一样的。

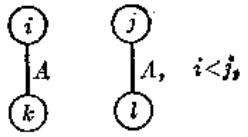
上述例子中的分布型式可以方便地表示成树结构：



这些树形的圆圈式的内部节点编号成1，2，3，…，它们对应于这一过程的步骤1，2，3，…。所有树枝旁分别标以带名A，B，C，D（代替T1，T2，T3，T4），以便说明这些记录的去处。正方形的外部节点表示一个文件的仅仅含有一个键的部分，该键以黑体示于此节点之下。在正方形节点上边的树枝带有输出带的名字（在头一个例子中为C，在第二个例子中为A）。

于是，例1中步骤3的组成是：从带B读入并写诸1和诸5到带A上，写诸3和诸7到带B上。如果我们假定每个键都同样经常地出现，则不难看出，所执行的扫描趟数等于树的外部路径长度除以外部节点数。

由于带的顺序性，以及向前读的先进先出的规则，我们不能简单地使用任何有标号的树来作为一个分布型式的基础。在例1的树中，在步骤2和步骤3期间数据写到带A上；在我们使用步骤3期间所写的数据之前，必须首先使用步骤2期间所写的数据。一般地说，如果在步骤i和j期间写到一个带上，其中$i<j$，则我们必须首先使用在步骤i期间写的数据；当树包含形如



的两个分枝时，必须有$k<l$，并且不能在步骤k和l之间往带A上写任何东西，因为我们必须在读和写之间进行重绕。

已经作了5.4.4节的习题的读者，现在会立即觉察到，可以表示在T条带上向前读基数排序的树，恰恰就是“强T先进先出”树，它表征了T条带上的向前读的合并排序（见习题5.4.4-20）！唯一的差别是，我们在这里考虑的树的所有外部节点都有相同的带标号。只要假定一个最后的“收集阶段”，这个阶段把所有记录传送到一条输出带上，我们就可以去掉这个限制；或者也可以附加这个限制到T先进先出树的诸规则上，为此只须要求合并排序的初始分布扫描能明显地在对应的合并树中表示出来。

换句话说，每一个合并型式对应于一个分布型式，而每一个分布型式对应于一个合并型式。稍经考虑后就知道为什么是这样的：我们按反方向做合并排序，首先“拆开”最后的输出成为一些子文件，这些子文件又拆开成其它的，等等。最后，我们将把这个文件拆开成为S个路段。实现这样一个型式的充要条件是，对应的基数排序分布型式对于S个键说来是可能的。合并与分布之间的这种对偶性几乎是完全的；它仅在一个方面不成立，即输入带的内容需要保护几次。

本节开始时所讨论的八个键的例子，显然对偶于四条带上的一个平衡合并。部分扫描的十个键的例子，对应于下列十路段的合并型式（如果去掉复写阶段——即树中的步骤6-11——的话）：

	T1	T2	T3	T4
初始分布	1^4	1^3	1^1	1^2
树步骤5	1^3	1^2	—	$1^2 3^1$
树步骤4	1^2	1^1	2^1	$1^2 3^1$

树步骤 3	1^1	—	$2^1 3^1$	$1^1 3^1$
树步骤 2	—	4^1	3^1	3^1
树步骤 1	10^1	—	—	—

如果把这同基数排序进行比较，则我们看到，这些方法实际上都有相同的构造，但在时间上是颠倒的，同时带的内容也从后面颠倒到前头：$1^2 3^1$(两个长度皆为 1 的路段，后边接有长度为 3 的一个路段）对应于{3， 8， 9}{4}{5}(两个子文件，每个包含 1 个键，前面还有包含 3 个键的一个子文件)。

换一种方式，我们原则上也可以构造一个基数排序，使其对偶于多阶段合并、级联合并，等等。例如，在 5.4.2 节开始所说明的在三条带上的21个路段的多阶段合并，对应于下列有趣的基数排序：

	T1	T2
给 定	{0，1，…20}	—
阶段1.	—	{0，2，4，5，7，9，10，12，13，15，17，18，20}
阶段2.	{0，5，10，13，18}	—
阶段3.	{0，5，10，13，18}{1，6，11，14，19}{2，7，12，15，20}	{3，8，16}{4，9，17}
阶段4.	—	{3，8，16}{4，9，17}{5，10，18}{6，11，19}{7，12，20}
阶段5.	{8}{9}{10}{11}{12}	—
阶段6.	{8}{9}{10}{11}{12}…{20}	{0}{1}…{7}

	T3
给定	—
阶段1.	{1，3，6，8，11，14，16，19}
阶段2.	{1，3，6，8，11，14，16，19}{2，4，7，9，12，15，17，20}
阶段3.	—
阶段4.	{0，13}{1，14}{2，15}
阶段5.	{0，13}{1，14}{2，15}{3，16}…{7，20}
阶段6.	—

这里用来决定在每个步骤中哪些键进到哪些带上的分布规则，似乎是玄妙的，但是事实上它与斐波那契数系统有着一种简单的联系!(见习题 2)。

向后读 基数排序和合并之间的对偶性也可应用于向后读的算法。我们已经在 5.4.4 节定义了“T 后进先出树”，容易看出，它们对应于基数排序，也对应于合并排序。

实际上，在1946年约翰·莫兹利就已考虑过向后读的基数排序，那是一篇关于排序的最早发表的论文之一（参照 5.5 节)；莫兹利实际给出了下列构造：

	T1	T2	T3	T4
给 定	—	{0，1，2，…，9}	—	—
阶段1.	{4，5}	—	{2，3，6，7}	{0，1，8，9}
阶段2.	{4，5}{2，7}	{3，6}	—	{0，1，8，9}
阶段3.	{4，5}{2，7}{0，9}	{3，6}{1，8}	—	—
阶段4.	{4，5}{2，7}	{3，6}{1，8}	{9}	{0}
……				
阶段8.	—	—	{9}{8}{7}{6}{5}	{0}{1}{2}{3}{4}
最后的收集	—	—	—	{0}{1}{2}…{9}

这个方案并不是最有效的，但它却是有趣的，因为它说明，在1946年即已考虑过在基数排序中使用部分扫描方法。但是直到大约1960年以前它们还未出现在关于合并的著作中。

向后读分布型式的一个有效的构造已经由A.贝斯（A.Bayes)提出〔*CACM* 11(1968) 491-493〕；给定$P+1$条带和S个键，把这些键分成为P个子文件，每一个含$\lceil S/P \rceil$或$\lceil S/P \rceil$个键，并递归地应用这个过程于每个子文件。当$S<2P$时，一个子文件应仅由最小的键所组成，而且它应被写到输出文件中（R.M.卡普的一般先序构造包括这个方法作为其特殊情况，卡普的方法出现在5.4.4节末尾)。

向后读使合并稍微复杂化了，因为它颠倒了路段的次序，对于基数排序有相应的影响；它使得结果是稳定的或"不稳定的"取决于在树中达到什么级。在一个向后读的基数排序(其中某些外部节点在奇数级上，而某些在偶数级上)之后，具有相等键的不同记录的相对次序，对于某些键将和原始次序相同，但对于其它键，则将和原来的次序相反（参照习题6）。

交替合并排序也有它们在对偶意义下的配对物。在一个交替基数排序中，我们继续分开键，直到达到仅有一个键的子文件为止，或者直到子文件小到足以在内部进行排序为止。对这样的子文件排序并把它们写到输出带上，然后继续分下去。例如，如果有三条工作带和一条输出带，并且如果键是二进制数，则我们可以开始把形如"0x"的键放在带T1上，把键"1x"放到T2上，如果T1接收一个以上的内存负载，则再次扫描它，并且置"00x"到T2上和"01x"到T3上，现在如果"00x"子文件短到足以在内部进行排序，则排序之并输出其结果，然后继续来处理"01x"子文件。这样一个方法被E.H.弗兰德称为"级联伪基数排序"〔*JACM* 3(1956)，157-159〕；H.纳格勒进一步发展了这个方法〔*JACM* 6(1959)，459-468〕，他给它以精采的名称"两头蛇排序"。而C.H.高德特（C.H.Gaudette）也发展了这一方法〔*IBM Tech. Disclosure Bull.* 12（*April*，1970)，1849-1853〕。

基数排序胜过合并吗？ 对偶性原理的一个重要结果是基数排序通常都低劣于合并排序，这是由于替代选择技术使合并排序具有肯定的优势所致；没有明显的方式来安排基数排序，使得我们能利用同时处理一个以上内存负载的内部排序。问题在于交替基数排序将经常产生稍微小于一个内存负载的子文件，所以分布型式对应于一株树，其外部节点较使用合并和替代选择时出现的外部节点多得多。因此，树的外部路径长度（即排序时间）就增加了（参照习题5.3.1-33)。

然而，外部基数排序有某些重要的应用。例如，假设我们有一个包含某个大公司所有职员名字的文件，名字以字母顺序出现；这个公司有10个部门，而且希望按部门来对文件进行排序，同时在每个部门中仍保留职员的字母顺序。如果文件很长，则这就是应用一个稳定的基数排序的理想情况，因为属于每个部门的记录数大概将多于通过替代选择产生的初始路段中所得到的记录个数。一般说来，如果键的值范围如此之小，以致具有同一键的记录集合相当内存大小的两倍以上，则使用一项基数排序技术是明智的。

在5.2.5节我们已经看到，在某些高速计算机上，内部基数排序优于合并排序，因为基数排序算法的"内循环"避免了复杂的分枝。如果外存特别快，则这样的机器不可能足够快速地合并数据以赶上输入/输出设备。因此，可以证明在这种情况下基数排序优于合并，特别是，如果已知键是一致地分布时。

习题

1.〔20〕 在5.4节的开始处，我们定义了具有参数P，$1\leqslant P<T$的一般的T条带

的平衡合并，证明这对应于一个使用一个混合基数系统的基数排序。

2.〔$M28$〕 正文说明了对于21个键的三带多阶段基数排序。试把它推广到F_n个键的情况；说明在每个阶段的末了，什么键出现在什么带上〔提示：考虑斐波那契数系统，习题1.2.8-34〕。

3.〔$M40$〕 把习题2的结果推广到四条带或更多带上的多阶段基数排序（参照习题5.4.2-10）。

4.〔$M20$〕 阿申赫斯特分布型式是不向后读的在四条带上对10个键进行排序的最好方式，因为在所有"强4先进先出树"中，与这种型式相应的树有极小的外部路径长度。试证明这点。（于是我们忽略重绕时间，则它实质上是最好的方法。）

5.〔15〕 画出对应于10个键的莫兹利向后读基数排序的4后进先出树。

▶6.〔20〕 某个文件包含两位数字的键00，01，…，99。在执行了对于个位数字的莫兹利基数排序之后，交换T2带和T4带的作用，我们可以对十位数字重复同一方案。问在T2带上诸键最终将以什么次序出现？

7.〔21〕 对偶性原理也可以应用于多卷文件吗？

*5.4.8 双带排序

由于我们需要三条带来实现一个没有过多的磁带运动的合并过程，因此探索如何只用两条带就实现一个合理的外部排序是有趣的。

H. B. 德穆思于1956年提出的一个方法，是综合替代选择和气泡排序的一种排序。假定输入在带T1上，开始读$P+1$个记录到内存中。现在输出其键为最小的记录到带T2上，并且代之以下一个输入记录，利用一个选择树或$P+1$个元素的优先队，可继续输出当前内存中具最小键的记录。当最终穷尽输入时，这个文件的最大的P个键将出现在内存中；以递增次序输出它们。现在重绕这两条带并且通过从T2读和往T1写来重复这个过程；每趟这样的扫描，都至少安排好又一组P个记录的位置。程序中可加进一个简单的检验，以确定整个文件何时排好序。至多需要$\lceil (N-1)/P \rceil$趟扫描。

稍经考虑后表明，这个过程的每一趟扫描，实际上就等价于气泡排序（算法5.2.2B）的P趟连续扫描！如果一个元素有P个或更多的反序，则当它输入时，它将比树中的一切都小，所以它将立即被输出（因此失去P个反序）。如果一个元素有少于P个反序，则它将进入到选择树中，而且将在所有比它大的键之前先被输出（由此失去它的全部反序）。当$P=1$时，由定理5.2.2I，这恰是气泡排序的情况。

因此扫描的总趟数将是$\lceil I/P \rceil$，其中I是各元素反序的极大个数，按5.2.2节中发展的理论，I的平均值是$N-\sqrt{\pi N/2}+\frac{2}{3}+O(1/\sqrt{N})$。

如果文件不比内存容量大很多，或者它几乎一开始就是排好序的，则这个P阶的气泡排序将是相当快的；事实上，甚至当有额外的磁带机可利用时，使用它也可能是有利的，因为它可能还用不着花费计算机操作员安装第三条带所花的那么多时间！另一方面，对相当长的随机排序文件，它将运行得十分慢，因为它的运行时间近似于同N^2成比例。

让我们来看看对于5.4.6节的100000个记录的例子，这个方法如何实现？我们需要明

智地选择P，为的是补偿同时进行读、写和计算时的块区间间隔。由于这个例子假定每个记录的长度是 100 个字符，有100,000个字符将填入内存，故可以通过置

$$100(P+1)+4B=100000 \tag{1}$$

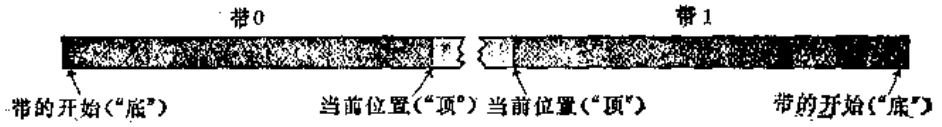
为大小为B的两个输入缓冲区和两个输出缓冲区提供位置。利用 5.4.6 的记号，每趟扫描的运行时间将大约是

$$NC\omega\tau(1+\rho),\quad w=(B+\gamma)/B \tag{2}$$

由于扫描数同P成反比，我们需要选择B为 100的倍数，它把量ω/P极小化。初等微积分说明，当B近似于$\sqrt{24975\gamma+\gamma^2}-\gamma$时出现极小，所以我们取$B=3000$，$P=879$。在上述公式中置$N=100000$，就看出扫描的次数$\lceil I/P\rceil$将大约是 114，而总共的运行时间将近似于8.57小时（为方便起见，假定初始输入和最后的输出也是$B=3000$）。这大约相当于满卷数据的0.44倍；一个满卷将大约有五倍之长。我们可做某些改进，即周期地中断算法的运行，把有最大键的诸记录写到一条辅助带上并把它卸下，因为一旦这样一些记录已按顺序放置了，所需要的只不过是向前和向后复写它们罢了。

快速排序的应用 另一个以近乎顺序的方式遍历数据的内部排序方法是分划交换或快速排序过程算法5.2.2Q，我们能否使它适应两条带呢〔N. B. 约希(N. B. yoash)，*CACM* 8 (1965)，649〕？

利用向后读，即不难看到这可以如何完成。假设把两条带编号成 0 和 1，而且想象文件的安排如下：



每一条带用作一个栈；把它们象这样放在一起，就有可能把文件看作一个线性表，其中，通过从一个栈复写到另一个栈，我们可以左右移动当前的位置。下列的递归子例程定义了一个适当的排序过程：

· SORT00 〔对 0 带顶部的子文件排序，并且把它记回到 0 带上。〕

如果这个子文件在内存中放得下，则对它内部排序并把它送回带上。否则就从这个子文件选择一个记录R，命它的键为K。在 0 带上向后读，复写其键$>K$的所有记录，在 1 带的顶部形成一个新的子文件。现在在 0 带上向前读，复写其键$=K$的所有记录到 1 带上。然后再次向后读，把键$<K$的所有记录复写到 1 带上。通过对于$<K$的键执行SORT10 而完成排序，然后把$=K$的键复写到 0 带上，最后对于$>K$的键执行 SORT10。

· SORT01 〔对 0 带顶部的子文件排序而且把它写到 1 带上。〕 和SORT00一样，但是最后的“SORT10”改成为“SORT11”，之后紧接着复写$\leqslant K$的键到 1 带上。

· SORT10 〔对在 1 带顶部的子文件排序并把它写到 0 带上。〕 和 SORT01一样，只是交换 0 和 1，以及交换$<$和$>$。

· SORT11 〔对 1 带顶部的子文件排序并且把它送回到 1 带上。〕 和 SORT00一样，交换 0 和 1 以及交换$<$和$>$。

这些子例程的递归性，可以毫无困难地通过在诸带上存以适当的控制信息，来加以处理。

如果我们假定数据是按随机次序排列的，同时有相同键的概率可予忽略，则对于这个算法的运行时间可估计如下：设M是装满内部存储的记录个数，命X_N是当$N>M$时在应用SORT00或SORT11于N个记录的一个子文件时平均所读的记录个数，且命Y_N是对于SORT01或SORT10对应的量，于是我们有

$$X_N=\begin{cases}0 & \text{如果}N\leqslant M;\\ 3N+1+\dfrac{1}{N}\sum\limits_{0\leqslant k<N}(Y_k+Y_{N-1-k}), & \text{如果}N>M;\end{cases}$$

$$Y_N=\begin{cases}0 & \text{如果}N\leqslant M;\\ 3N+2+\dfrac{1}{N}\sum\limits_{0\leqslant k<N}(Y_k+X_{N-1-k}+k), & \text{如果}N>M,\end{cases}\tag{3}$$

这些递归式的解（见习题2）表明，当$N\to\infty$时，在外部分划阶段期间读带的总数量，平均说来，将是$6\cdot\frac{2}{3}N\ln N+O(N)$，从等式5.2.2-25我们也知道，内部排序阶段的平均数将是$2(N+1)/(M+2)-1$。

如果把这一分析应用于5.4.6节的100,000记录的例子，并且利用25000个字符的缓冲区，以及假定对于n个记录的子文件，$n\leqslant M=1000$个记录，排序时间是$2nC\omega t$，则我们得到近似于103分钟的平均排序时间（如同在图表A中那样包括最后的重绕时间）。于是快速排序方法平均说来还不坏，当然，它的最坏情况证明甚至比上面讨论过的气泡排序更为糟糕！

基数排序 基数交换方法（算法5.2.2 R）可以以类似的方式适用于两条带的排序，因为它很象快速排序。使这两种方法都有效的“技巧”，是不止一次地读一个文件的思想，这是我们以前的带算法所决不做的事情。

同样的技巧亦可应用于在两条带上进行一种通常的“首先比较最低位有效数字”的基数排序。给定T1带上的输入数据，我们把所有其键在二进记号下以0结尾的记录都复写到T2上；然后在重绕T1之后，再次读它，同时复写其键以1结尾的记录。现在两条带都被重绕，而且作类似的一对扫描，在扫描时交换T1和T2的作用，并使用第二个最低位有效二进数字。这时，T1将包含其键为$(\cdots00)_2$的所有记录，紧接着是其键为$(\cdots01)_2$的，然后是键为$(\cdots10)_2$的，然后是$(\cdots11)_2$的。如果键是6位长，则为了完成这个排序，我们仅仅需要对文件进行26次扫描。

对于某些审慎地选择的数b，这样一种基数排序可仅应用于键开头的b位；如果诸键是一致地分布的，这将使反序的个数减少大约2^b的一个因子，所以可以用较少次扫描的P路气泡排序来完成这一作业。

A.И.尼基廷（А.И.Никитин）和Л.Н.肖尔莫夫（Л.Н.Шолмов）〔Кибернетика 2，6 (1966)，79–84〕已经提出了对于两条带分布排序的一个新奇但稍更复杂的方法。计算其前头几位为各种可能的组合的键的个数，并根据这些计数构造人为的键，$\kappa_1,\kappa_2,$

…，κ_M，使得对于每个 i，位于 κ_i 和 κ_{i+1} 之间的实际键的个数是在预先确定的极限 P_1 和 P_2 之间。于是，M 位于 $\lceil N/P_2 \rceil$ 和 $\lceil N/P_1 \rceil$ 之间。如果头几位的计数未给出充分的信息以确定这样的 κ_1，κ_2，…，κ_M，就进行再一次或更多次的扫描，以便对于某些最高有效位的组合，算出各种较低有效位型式的出现频率。在构造了人为的键 κ_1，κ_2，…，κ_M 的表之后，$2\lceil \log_2 M \rceil$ 次进一步的扫描足以来完成这个排序。(这个方法要求同 N 成比例的存储空间，所以当 $N \to \infty$ 时它不能用于外部排序。实际上我们将不对多卷文件使用此技术，所以 M 将总是比较小的，而且人为键一定能在内存中放下)。

更多带的模拟 F. C. 亨尼 (F. C. Hennie) 和 R. E. 斯特恩斯 (R. E. Stearns) 已经想出了仅在两条带上模拟 k 条带的一般技术，其方法是，所需要的带运动仅以 $O(\log_2 L)$ 的因子增加，其中 L 是在任一条带上遍历的极大距离〔*JACM* 13 (1966)，533-546〕。他们的构造，如同 R. M. 卡普所提出的下列方法那样，在排序的情况下，可稍作简化。

利用两条带 T1 和 T2，我们来模拟一个通常的四条带的平衡合并。首先，T1 用来保存被模拟的带的内容，其方式由图86给出；我们想象，对于每一条被模拟的带，数据写到四个"道"上

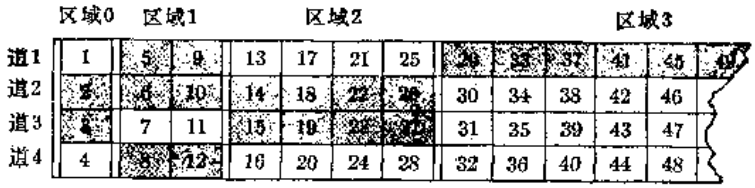


图86 在亨尼·斯特恩斯构造中带1的安排，非空白的区域带有阴影

(事实上，带没有这样的"道"，块区 1，5，9，13，…被想象作道 1，而块区 2，6，10，14，…被想象作道 2，等等)。另一条带 T2，仅用作辅助存储，以帮助 T1 移动其数据。

每个道的块区组成一些区域，每个区域分别包含 1，2，4，8，…，2^k，…个块区，在每个道上的区域 k，或者正好装满 2^k 个数据块区，或者完全是"空白"。例如，在图86中，道 1 在区域 1 和 3 中有数据；道 2 在区域 0，1，2 有数据；道 3 在区域 0 和 2；道 4 在区域 1；其余的区域都是空白。

假设我们正在从道 1 和道 2 把数据合并到道 3。计算机的内存中有两个缓冲区用于两路合并的输入，加上第三个缓冲区用作输出，当道 1 的输入缓冲区变空时，我们可以按如下方式重新填满它：找出道 1 上第一个非空的区域，比如说，区域 k，复写它的第一个块区到输入缓冲区；然后复写其它 2^k-1 个块区的数据到 T2 上，并把它们移动到道 1 的区域 0，1，…$k-1$ (区域 0，1，…，$k-1$ 现在被填满而区域 k 是空白)。每当道 2 的输入缓冲区变空时，一个类似的过程用来重新填满道 2 的输入缓冲区。当输出缓冲区准备好要写到道 3 时，我们颠倒这一过程，扫描整个 T1 以找出道 3 上的第一个空白区域，比如说区域 k，同时从区域 0，1，…，$k-1$ 复写数据到 T2 上。由输出缓冲区的内容加以扩充的 T2 上的数据，现在用来填满道 3 的区域 k。

这一过程要求有能力写到带 1 的中间，而不破坏该带上随后的信息，如同在向前读交替排序的情况 (5.4.5节) 那样。如果采用适当的预防措施，就有可能可靠地做到这一点。

为把道 1 的 2^l-1 个块区送进内存所需要的磁带运动的数量是 $\sum_{0\le k<l} 2^{l-1-k}\cdot c\cdot 2^k = cl2^{l-1}$，$c$ 是某个常数 (因为在每 2^k 步中我们只扫描到区域 k 一次)。于是，每趟合并扫描需要 $O(N\log_2 N)$ 步。由于在平衡合并中有 $O(\log_2 N)$ 趟扫描，故最坏情况下的完全排

序时间肯定是$O(N(\log_2 N)^2)$，这在渐近意义下比快速排序的最坏情况好得多。

但是，如果我们把这个方法应用到5.4.6节的100,000个记录的例子上，它实际上并不是非常好的，因为送往带1的信息将超出一卷带的内容。即使忽略这一事实，而且即使使用关于读/写/计算重叠以及块区间间隔长度等最优的假定，我们会发现，为完成这一排序，仍将需要大约37小时！所以这个方法只有纯学术的意义；当N是在一个实用的范围时，$O(N(\log_2 N)^2)$中的常数太大了，不能令人满意。

一条带排序 我们能否只用一条带呢？不难看出，上面描述的P阶气泡排序能转化成一条带的排序，但结果可能是极坏的。

H.B.德穆思[Ph.D. thesis (Stanford University, 1956), 85]发现，具有有限内存的一台计算机，当它在带上运动有限距离时，它所能减少的一个排列的反序个数，不多于一个有限的数量。因此每个单带的排序算法，平均说来至少必须花费dN^2个时间单位(对于某个正的常数d，这个常数依赖于计算机的配置)。

R.M.卡普已经以一种非常有趣的方式来推进这个课题，并且发现了用一条带进行排序的一个最优方式是怎样的。要讨论卡普的算法，宜于把这个问题改变如下：利用单部电梯在诸层楼之间运送人们的最快方式是什么？

考虑一个具有n层的建筑物，在每层上只能容下c个人。这个建筑物没有门、窗、楼梯，但它有可以停在每层上的电梯。在建筑物中有cn个人，每层楼恰好有c个人要去。电梯至多能容b个人，而且它从第i层到第$i+1$层要花费一个时间单位。如果要求电梯在第一层开始和结束的话，我们希望找出使所有的人都到达适当的层上的最快方法。

不难看出这个电梯问题和单带的排序问题之间的联系：人是记录而建筑物是带，楼层是带上个别的块区，而电梯是计算机的内存。一个计算机的程序比一个电梯操作员有更多的灵活性（例如，它可以对人们复制复分或暂时把他们劈成在不同层上的两部分，等等）；但下列的解法不必做这样的操作，而以能想象到的最快时间来解决这个问题。

卡普的算法需要下列两张辅助表：

u_k，$1\leqslant k\leqslant n$：在$\leqslant k$的层上其目的地$>k$的人数；

d_k，$1\leqslant k\leqslant n$：在$\geqslant k$的层上其目的地$<k$的人数；　　(4)

当电梯空时，对于$1\leqslant k<n$，我们总有$u_k=d_{k+1}$，因为在每层上有c个人；在$\{1,\cdots,k\}$层上等待乘电梯的人数，必须等于在$\{k+1,\cdots,n\}$上相应的人数。由定义，$u_n=d_1=0$。

显然，对于$1\leqslant k<n$，电梯至少必须进行$\lceil u_k/b\rceil$次从层k到层$k+1$的旅行，因为仅b个旅客可以在每次旅行中上楼。类似地，它必须至少进行$\lceil d_k/b\rceil$次从第k层到$k-1$层的旅行，因此对于任何正确的调度，电梯至少需花费

$$\sum_{1\leqslant k\leqslant n}(\lceil u_k/b\rceil+\lceil d_k/b\rceil) \qquad (5)$$

个时间单位，卡普发现，当$u_1,\cdots,u_{n-1}$非0时，这个下限实际上可以达到。

定理K 如果对于$1\leqslant k<n$，$u_k>0$，则存在一个电梯调度，它在极小时间(5)之下载运每一个人到他的目的地。

证明 假定在建筑物中有额外的b个人；他们开始在电梯中，并且他们的目标层人为

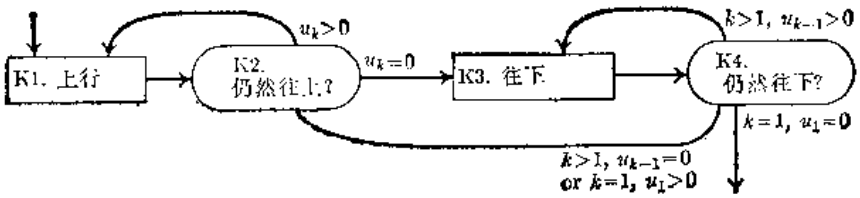


图87　卡普的电梯算法

地置成0，电梯可以按照下列算法来进行操作，以k（当前的层）等于1开始：

K1.〔上行〕 请当前在电梯中或在第k层上的$b+c$个人中，有最大目的地的那b个人上电梯，而其余的人留在k层上。

设现在电梯上有u个人的目的地$>k$，另外d个人的目的地$\leqslant k$（结果$u=\min(b, u_k)$；如果$u_k<b$，则可能要运送某些人离开他们的目的地。这表示他们为共同的好事作出牺牲）。u_k减u，d_{k+1}加d而后k加1。

K2.〔仍然往上?〕 如果$u_k>0$，则返回步骤K1。

K3.〔下行〕 请当前在电梯或在k层的$b+c$个人当中，那些具有最小目的地的人进电梯，而其余的停留在k层上。

设现在电梯中有u个人的目的地$\geqslant k$，另外d个人的目的地$<k$。（结果将总是$u=0$和$d=b$，但这里所述的算法仍用一般的u和d来描述，为的是要使证明稍更清楚些。）d_k减d，u_{k-1}加u，而后k减1。

K4.〔仍然往下?〕 如果$k>1$和$u_{k-1}>0$，则返回步骤K3。如果$k=1$和$u=0$则此算法结束。（每个人已经移到他的目的地而b个"额外者"也回到电梯中。（否则返回到步骤K2。

图88示出这个算法的一个例子，这是一座九层建筑物且$b=3$，$c=2$。注意，尽管电梯走的是最小的距离，6人中仍有1人被暂时送离他的目的地。在步骤K4中测试u_{k-1}的思想是关键性的，它使得算法有效，我们将看到的这一点。

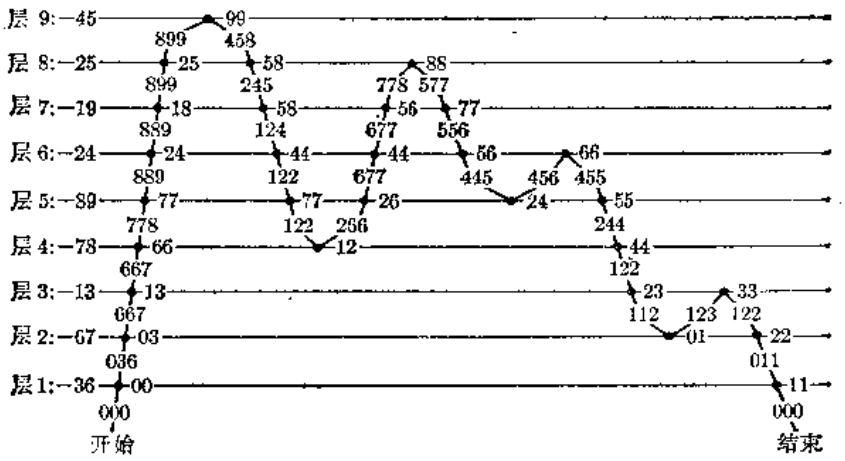


图88　利用一个小的缓慢的电梯来重新安排人们的最佳方式（每个人以他的目的层号来表示）

为了证明这个算法的正确性，我们注意，如果我们认为在电梯里的人是在"当前"层k上，则步骤K1和K3总是保持u和d表（4）的内容是最新的。现在有可能用归纳法证明下列性质在每个步骤的开始时均成立：

$$u_l = d_{l+1}, \qquad 对于\ k \leqslant l < n \tag{6}$$

$$u_l = d_{l+1} - b,\ 对于\ 1 \leqslant l < k \tag{7}$$

$$如果\ u_l = 0\ 和\ k \leqslant l < n\ 则\ u_{l-1} = 0 \tag{8}$$

进而，在步骤K1的开始，在$\leqslant k$层上且目的地$> k$的所有人当中，具有最高目的地的$\min(u_k,\ b)$个人，是在电梯中或在k层上。在步骤K3的开始，在$\geqslant k$层且目的地$< k$的所有人当中，具有最低目的地的$\min(d_k,\ b)$个人，是在电梯中或在k层上。这些条件也可以通过考虑我们怎样来达到步骤K1或K3，而归纳地验证(参照习题5)。

从这些性质得出，在步骤K1和K3中的带圆括号的注释是正确的。因此，步骤K1的每一次执行都使$\lceil u_k/b \rceil$减少1，而保持$\lceil d_{k-1}/b \rceil$不变；步骤K3的每次执行都使$\lceil d_k/b \rceil$减少1而使$\lceil u_{k-1}/b \rceil$保持不变。因此这个算法必定在有限步之内结束，而且每个人由于(6)和(8)必然处于他的目的地处。

当$u_k = 0$和$u_{k-1} > 0$时，我们有一个"断开"的状态；尽管没有人要从$\leqslant k$的层上升到$\geqslant k+1$的层，电梯还是必须升到$k+1$层以便重新安排那儿的人们。不失一般性，我们可以假定$u_{n-1} > 0$；于是每个正确的电梯调度都必须至少包括

$$2 \sum_{1 \leqslant k < n} \max(1,\ \lceil u_k/b \rceil) \tag{9}$$

次移动，因为我们要求电梯返回到层1，达到这一下限的调度是容易构造出来的(习题4)。

习题

1. 〔17〕 正文中讨论的P阶气泡排序，仅仅使用向前读和重绕。能否把这个算法修改来利用向后读的好处？

2. 〔M26〕 求出在(3)中定义的数X_N、Y_N的明显的"封闭形式"的解〔提示：研究等式5.2.2-19的解〕。

3. 〔38〕 是否有一个两条带的排序方法，它仅以键(不是数字性质的)比较为基础，当对N个记录排序时，它的带运动在最坏的情况下为$O(N\log_2 N)$〔快速排序平均说来达到这一点，但在最坏的情况下不行，亨尼·斯特恩斯方法(图86)则达到$O(N \times (\log_2(N)^2))$〕？

4. 〔M23〕 在电梯问题中，假设有下标p、q，且$q \geqslant p+2$，$u_p > 0$，$u_q > 0$且$u_{p-1} = \cdots = u_{q-1} = 0$。说明怎样构造至多需要(9)时间单位的一个调度。

▶5. 〔M23〕 真或假：在定理K的算法的步骤K1之后，电梯上的人谁也不比$< k$的层上的任何人有更小的目的地。

6. 〔M30〕 (R. M. 卡普) 把电梯问题推广到下列情形：对于$1 \leqslant j \leqslant n$，开始时在$j$层上有$c_j$个乘客，以及有$c'_j$个乘客其目的地是$j$层，证明存在一个调度，它花费

$$2 \sum_{1 \leqslant k < n} \max(1,\ \lceil u_k/b \rceil,\ \lceil d_{k+1}/b \rceil)$$

个时间单位，且在任何时刻决不允许在层j上有多于$\max(c_j,\ c'_j)$个乘客。〔提示：如果必要，引进虚构的人，使得对于所有的j，$c_j = c'_j$〕。

7. 〔M40〕 (R. M. 卡普) 推广习题6的问题，用一个客车通过的道路网络来代

替电梯所遵循的线性通路，其中假定该网络形成任意的自由树。这个客车容量有限，而且所希望的是，客车只行驶极小的距离，就把旅客运送到他们的目的地。

8.〔$M32$〕 设在正文中讨论的电梯问题里，$c=1$。问 n 层上的 n 个人有多少种排列使得在（4）中，对于 $1\leqslant k\leqslant n$，有 $u_k\leqslant 1$〔例如，3 1 4 5 9 2 6 8 7 就是一种这样的排列〕？

▶9.〔$M25$〕 找出在5.2.2节的图16中描述的"鸡尾混合排序"和在 $c=1$ 时（4）的数 u_1，u_2，…，u_n 之间的重要联系。

10.〔20〕 你将如何仅仅用两条带来对多卷文件排序？

5.4.9 磁盘和磁鼓

迄今我们一直把带当作外部排序的工具，但是通常还有更为灵活的大容量存储设备可用。尽管这样的"海量存储"或者"直接存取存储"设备形式五花八门，但它们可以粗略地以如下一些性质来表征：

(i) 它不必花费非常长时间来访问所存储信息的任何特定部分；

(ii) 在内存和外存之间可以迅速地传输连续的字的块区。

磁带满足（ii）但不满足（i），因为它要花费长时间来从带的一端达到另一端。某些设备满足（i）但不满足（ii），例如海量磁心存储，它对于每个字的存取时间大约比内存速度慢10倍。

在为一种外部存储设备编写大量程序之前，首先应当仔细地研究它的特性；但是技术的变化如此迅速，以致在这里不可能对于所有可用的硬件类型进行完备的讨论。因此，我们将仅仅考虑某些典型的存储设备，这些设备展示了对于排序问题的有用方法。

满足（i）和（ii）的最普遍的外部存储类型之一，是一个磁盘文件或磁盘组模块（见图89）。数据被存放在一些急速地转动的圆盘上，这些圆盘上涂了磁性材料；一个象梳子样的存取臂对于每个盘面包含一个或多个"读/写头"，它用来存储和检索信息。每个盘面分成称作道的同心的环，使得每当这个盘完成一次转动时，整个道的数据就通过一次读/写头。存取臂可以伸进和伸出，从一个道到另一个道地移动读/写头；但这种运动花费时间。可以不必重新定位存取臂就能读或写的一组道，称为一个柱面或一个数据层。例如，图 89 示出了一个磁盘文件，它的每个平面恰有一个读/写头；虚线表示一个柱面，它由当前正被读/写头扫描的所有道所组成。

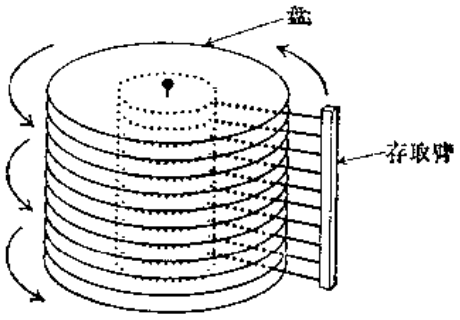


图89 一个盘组模块

为了叙述更确切一点，我们考虑假想的MIXTEC磁盘组设备，对于它

1 个道＝5000个字符

1 个柱面＝20个道

1 个磁盘设备＝200个柱面

这样一个磁盘设备含有 2 千万个字符，略少于一条磁带上所能存储的数据量。在某些机器

上，接近中心的道的字符比接近边缘的道要少；这势必使程序设计更为复杂，幸而，MIXTEC避免了这样的问题。

在一个磁盘文件上为读和写所需要的时间实质上是三个量之和。

- 寻找时间（为把存取臂移动到适当的柱面所需时间）。
- 等待时间（在读/写头达到正确的位置之前的转动延迟）。
- 传输时间（在数据通过读/写头时的转动延迟）。

在MIXTEC设备上为从柱面 i 转移到柱面 j 所需要的寻找时间是 $25+\frac{1}{2}|i-j|$ 毫秒。如果 i 和 j 是在1和200之间随机地选择的整数，则 $|i-j|$ 的平均值是 $2\binom{201}{3}/200^2\approx 66.7$，所以平均寻找时间大约是60毫秒。MIXTEC盘每25毫秒转动一圈，所以等待时间平均约12.5毫秒。n 个字符的传输时间是 $(n/5000)\times 25$毫秒 $=5n$ 微秒（这大约是5.4.6节的例子所用MIXT带的传输速度的 $3\frac{1}{3}$ 倍）。

因此，对于排序来说，MIXTEC盘和MIXT带之间的主要差别是

a）带仅可被顺序地存取；

b）个别的磁盘操作势必要求相当多的开销（寻找时间＋相对于停止/启动时间的等待时间）；

c）盘传输速度比较快。

通过使用更灵巧的带的合并型式，能够对缺点（a）作些补偿。我们现在的目标是考虑能补偿缺点（b）的某些灵巧的磁盘排序算法。

克服等待时间 让我们首先考虑使延迟极小化的问题。这延迟是由这样一个事实引起的，当启动一条输入输出指令时，磁盘往往并不正好在适当的位置上。我们不能使磁盘旋转得更快，但可以应用某些技巧，来减少或甚至消除所有的等待时间。加上更多的存取臂显然是有帮助的，但这将是一个昂贵的硬件修改。下面是某些"软件"的思想：

1）如果一次读或写同一个柱面的若干道，则我们就避免了除了头一次以外所有道上的等待时间（以及寻找时间）。一般地说，通常都有可能以这样一种方式来同步计算时间和磁盘运动的时间，即可以无须等待延迟而执行一系列的输入输出指令。

2）考虑读半个数据道的问题(图90)：如果当头在点 A 时开始读命令，则没有等待延迟，而用于读的总时间恰是传输时间 $\frac{1}{2}\times 25$毫秒。如果当头在点 B 时开始这命令，则我们需要转 $\frac{1}{4}$ 圈以用于等待和 $\frac{1}{2}$ 圈用于传输，总共 $\frac{3}{4}\times 25$毫秒。当头一开始位于 C 时出现最有趣的情况：通过适当的硬件和软件，我们不需要浪费 $\frac{3}{4}$ 的转动用于等待延迟。立即可开始读，读到输入缓冲区的第二半；然后在 $\frac{1}{2}\times 25$毫秒的中止之后，可以继续读到这个缓冲区的头一半，于是当再次到达点 C 时，指令已告完成。以一种类似的方式，我们可以确保总共的等待＋传送时间决不超过转一圈的时间，而不管磁盘的初始位置如何。通过这个方案减少了平均等待延迟时间，如果我们正在读或写一个道的指定部分 x 时（0

$< x \leqslant 1$），可由延迟半圈减少到延迟$\frac{1}{2}(1-x^2)$圈。当正在读或写整个一个道（$x=1$）时，这个技术消除了所有等待时间。

鼓：无寻找的情况 某些外部存储设备，由于对每个道都有一个读/写头，这就消除了寻找时间。如果图90的技术被应用到这样的设备上，则寻找时间和等待时间就都减少成0，其中假定我们总是一次读或写一个道；这时传输时间是唯一的限制因素，这是一种理想情况。

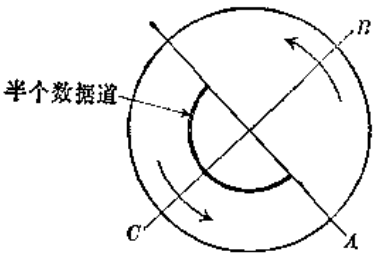


图90 读半个数据道时等待时间的分析

让我们再次考虑5.4.6节的应用实例，用有100,000个字符的内存对100,000个每个有100个字符的记录排序。有待排序的数据总量填满了一个MIXTEC盘的一半。通常不可能同时在单个磁盘设备上进行读和写；我们将假定有两个磁盘可以利用，使得读和写可以彼此重叠。事实上，暂时我们将假定“盘”实际上是鼓，它包含4000个每个含5000个字符的道，而且不需要寻找时间。

应使用什么排序算法呢？合并的方法是一个相当自然的选择，内部排序的其它方法不见得更利于在盘上实现，也许基数技术除外，但是5.4.7节的考虑表明，对于一般应用，基数排序通常比合并要低劣（不难看出，该节的对偶性定理既可应用于带，也可应用于盘）。

为了开始对于这个问题的一个合并排序，我们可以使用替代选择，用两个5000个字符的输入缓冲区和两个5000字符的输出缓冲区。事实上，如果用从选择树来的记录代替当前输入缓冲区中的记录，则有可能把上述缓冲区减少成三个5000字符的缓冲区。这把85000个字符（850个记录）送到一株选择树，所以对于我们所举例子数据的一次扫描将形成大约60个初始路段（见等式5.4.6.3）。如果假定内部处理足够快，能赶上输入输出的速度（每500微秒一个记录移动到输出缓冲区），则这次扫描仅仅花费大约50秒的时间。如果有待排序的输入出现在一条MIXT带上，而不是出现在一个鼓上，则这个扫描将慢一些，这是由带的速度所致。

对于两个鼓和全道的读/写，不难看到，如果我们命P尽可能地大，则P路合并的总传输时间即为极小。可惜，我们不能对所有初始路段简单地做一个60路的合并，因为在内存中没有60个缓冲区空间（少于5000个字符的一个缓冲区将导致不必要的等待时间）。如果进行P路合并，从一个鼓到其它鼓地传送所有数据，使得读和写重叠，则合并扫描的次数是$\lceil \log_P 60 \rceil$，所以如果$8 \leqslant P \leqslant 59$，我们就可以在两次扫描中来完成这项工作。最小的这样的P减少了内部计算的量，所以我们选择$P=8$；如果已经形成65个初始路段，则将取$P=9$。如果已经形成了82个或更多的初始路段，则将取$P=10$，但由于仅仅有18个输入缓冲区和2个输出缓冲区的空间，所以在合并期间将有挂起的可能性（见算法5.4.6F）；在这样一种情况下，可能更宜于对一小部分数据作部分的扫描，并且把初始路段的数目减到81或更少。

在我们的假定之下，两个合并扫描都将花费大约50秒，所以在这种理想的情况下完成整个的排序仅用$2\frac{1}{2}$分。（加上用于管理操作、初始化等等的几秒钟）。这比5.4.6节中所

考虑的最好的六带排序快六倍；速度提高的原因是：改进了外部/内部传输速度（快$3-\frac{1}{2}$倍），更高阶的合并（我们不能做八路的带合并，除非有九条或更多的带），以及输出保持在盘上这一事实（不必要最后的重绕，等等）。如果初始的输入和排序的输出要求在MIXT带上，而鼓仅用作合并，则相应的排序时间将大约是8.2分钟。

如果仅仅有一个鼓可利用而不是两个鼓，则输入输出时间将花费两倍之长，因为读和写必须分开来完成（事实上，输入输出操作将花费三倍时间，因为我们将冲掉初始的输入数据；在这样一种情况下，如果硬件不提供对已写信息的自动校验，则宜于在每一个写操作之后紧接着一个"向后读校验"操作，免得某些输入数据丢失以致无法检索）。但由于我们可以使用部分扫描的方法，这方法处理一部分数据记录的次数多于另一部分，因此就可以挽回一些时间上的损失。两个鼓的情况需要偶数次或奇数次地处理所有数据，但一个鼓的情况可以使用更一般的合并型式。

我们在5.4.4节就注意到，合并型式可以通过树来表示，而且对应于一个合并型式的传输时间同其树的外部路径长度成比例。仅仅某些树（T后进先出或在强T先进先出）可以用作有效的带合并的型式，因为某些路段在进行合并过程中会掩蔽在一条带的中间。但是在盘或鼓上，所有树都定义了可用的合并型式，只要它们内部节点的层次不是太大(即，与可利用的内部存储的大小相容)。

因此，我们可以通过选择一株具有极小外部路径长度的树来使传输时间极小化，诸如一株完备的P叉树，这里P尽可能地大。由等式5.4.4-9，如果这样一株树有S个外部节点(叶)，则它的外部路径长度等于

$$qS-\lfloor(P^q-S)/(P-1)\rfloor,\quad q=\lceil\log_P S\rceil \tag{1}$$

设计一个按照完备的P叉树型式来合并的算法，是特别容易的。例如，见图91，它示出了$P=3$，$S=6$的情况。首先，如果必要的话，我们加上"虚拟路段"，使得$S\equiv 1$ (modulo$P-1$)；然后，按照"先进先出"的规则来组合路段，在每个阶段，把队前头P个"最老的"的路段合并成为一个路段，并把该路段放在后边。

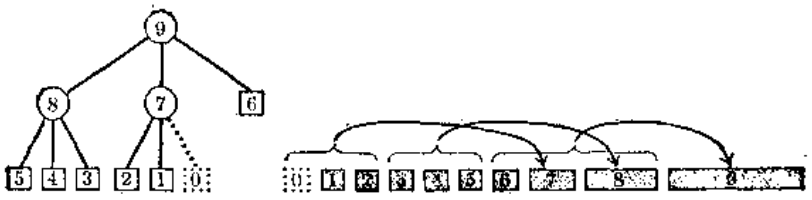


图91 具有六片叶的完备的三叉树及其相应的合并型式

如果所有的初始路段都有相同的长度，则完备的P叉树给出一个最优的型式，但是如果某些路段比其余的长，则我们常常可以做得更好。这种一般情况的最优型式，可以毫无困难地使用赫夫曼方法来加以构造（习题2.3.4.5-10），它可以用合并的语言叙述如下："首先加上$(1-S)\bmod(P-1)$个长度为0的虚拟路段，然后反复地把P个最短的现存路段合并在一起直到剩下一个路段为止。"当所有路段都有相同长度时，这个方法就归结为前面所述的先进先出规则。

在100,000个记录的例子中，我们可以作九路合并，因为内存中放得下18个输入缓冲区和两个输出缓冲区，而且算法5.4.6F将重叠所有的计算时间。如果所有初始路段都有相同的长度，则完备的具有60片叶的9叉树对应于具有$1\frac{29}{30}$趟扫描的一个合并型式。因此，若在每次写之后使用"向后读校验"，则使用一个鼓的总排序时间，约等于7.4分。用更高的P值可以稍稍减少这个运行时间；但由于可能出现"读挂起"，因此情况不很清楚，因为缓冲区可能变得太满或太空。

寻找时间的影响 以上的讨论表明，对于磁鼓来构造"最优"合并型式是比较容易的，因为寻找时间和等待时间实际上都可以没有。但当使用磁盘时，通常寻找信息的时间比起读信息的时间要长，所以寻找时间对于排序策略有相当大的影响。减少合并的阶数P，就可能使用更大的缓冲区，以减少所需的寻找时间；这通常能弥补较小的P值所引起的额外的传输时间。

寻找时间同存取臂通过的距离有关。我们可以努力把事情安排成使得这个距离成为极小。例如，首先在柱面内对诸记录排序可能是明智的。然而大型的合并要求在柱面之间作大量的跳跃（参考习题2）。其次，现代操作系统的多道程序设计的能力意味着，一个用户很少对磁盘存取臂的位置有很多的实际控制；使寻找时间极小化的各种方案通常都仅在周末时才行得通！因此，我们通常认为假定每一个磁盘命令都包含一个"随机"的寻找是合理的。

我们的目标是发现一株树（即一个合并型式），它在寻找时间和传输时间之间实现最好的平衡；为此目的，需要相对于一个特定的硬件配置来评价任何特定树的好坏。例如，考虑图92中的树；我们要估价为进行相应的合并它将花费多长时间，以便能把这株树同其它树进行比较。

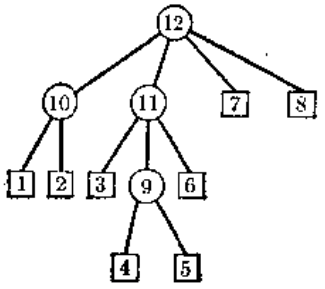


图92 外部路径长度为16而叉数路径长度为52的一株树

在下面的讨论中，我们将对磁盘合并作某些简单的假定，以便说明某些一般的思想。假定：（1）为了读或者写n个字符要花费$72.5+0.005n$毫秒；（2）内存中有100,000个字符的位置可用作工作存储；（3）为把每个字符从输入传输到输出，需要平均0.004毫秒的计算时间；（4）在读、写或计算之间没有重叠；以及（5）用作输出的缓冲区大小不必和下次扫描时用作读数据的缓冲区大小一样。对于这些简单假定之下排序问题的分析，将为我们转向更复杂的情况提供某些知识。

如果进行一个P路合并，则我们可以把内部工作存储分成为$P+1$个缓冲区域，P个作为输入和一个作为输出，而每个缓冲区都有$B=100,000/(P+1)$个字符。假定正在合并的文件总共包含L个字符；于是我们将做近似于L/B个输出操作和大约相同数量的输入操作，所以在我们的假定之下总共的合并时间将近似于

$$2\left(72.5\frac{L}{B}+0.005L\right)+0.004L=(0.00145P+0.01545)L\text{毫秒} \qquad (2)$$

换言之，L个字符的P路合并将花费大约$(\alpha P+\beta)L$个单位时间，其中α、β是依赖于寻找时间、等待时间、计算时间以及内存大小的常数。这个公式导致了一个有趣的构

造一个好的磁盘合并型式的方法。例如，考虑图92，并假定所有的初始路段（以正方形"叶"节点表示）长度都为L_0，则在节点9和10处的合并各须花费（$2\alpha+\beta$)($2L_0$）个单位时间，在节点11处的合并花费（$3\alpha+\beta$)($4L_0$）个，在节点12的最后合并花费（$4\alpha+\beta$）×($8L_0$）个。因此，总共的合并时间为($52\alpha+16\beta$)L_0单位。这里的系数"16"是我们所熟知的，它只不过是该树的外部路径长度。然而，α的系数"52"是一个新的概念，它可以称为树的叉数路径长度；它是（对于所有叶节点取的)从叶到根的通路上内部节点叉数的和。例如，在图92中，叉数路径长度是

$$(2+4)+(2+4)+(3+4)+(2+3+4)+(2+3+4)$$
$$+(3+4)+(4)+(4)=52$$

如果$\mathcal{T}$是任意树，则命$D(\mathcal{T})$、$E(\mathcal{T})$分别表示它的叉数路径长度和外部路径长度，我们的分析可以综述如下：

定理H 如果对于L个字符进行一个P路合并所需要的时间是（$\alpha P+\beta$)L，且如果有S个等长路段有待合并，则最好的合并型式对应于一株树$\mathcal{T}$，它使$\alpha D(\mathcal{T})+\beta E(\mathcal{T})$对所有的$S$片叶的树取极小。

（这个定理含蓄地包含在一篇未发表的论文中，乔治·U.哈巴德（George U.Hubbard）在1963年的ACM全国会议上介绍过它。）

命α和β是固定的常数；如果一株树对于具有相同的叶数的所有树$\mathcal{T}$有极小的$\alpha D(\mathcal{T})+\beta E(\mathcal{T})$值，则我们说这株树是最优的。不难看出，一株最优树的所有子树是最优的。因而我们可以通过把具有$<n$片叶的最优树合在一起，构造出具有n片叶的最优树。

定理K 对于$1\leqslant m\leqslant n$，通过规则

$$A_1(1)=0 \tag{3}$$

$$A_m(n)=\min_{1\leqslant k\leqslant n/m}(A_1(k)+A_{m-1}(n-k)) \quad 对于\ 2\leqslant m\leqslant n \tag{4}$$

$$A_1(n)=\min_{2\leqslant m\leqslant n}(\alpha mn+\beta n+A_m(n)) \quad 对于\ n\geqslant 2 \tag{5}$$

定义数$A_m(n)$的序列，则在所有具有n片叶的树$\mathcal{T}$中$A_1(n)$是$\alpha D(\mathcal{T})+\beta E(\mathcal{T})$的极小值。

证明 等式（4）蕴涵若$A_m(n)$是对于所有使得$n_1+\cdots+n_m=n$的正整数n_1，…，n_m，和式$A_1(n_1)+\cdots+A_1(n_m)$的极小值。对n用归纳法即得结果。

递归关系(3)、(4)、(5)也可用来构造最优树本身，命$k_m(n)$是在$A_m(n)$的定义中使极小值出现的一个值，则我们可以通过在根处联合$m=k_1(n)$个子树，构造具有n片叶的一株最优树；这些子树是分别具有$k_m(n)$，$k_{m-1}(n-k_m(n))$，$k_{m-2}(n-k_m(n)-k_{m-1}(n-k_m(n)))$，…片叶的最优树。

例如，表1说明了当$\alpha=\beta=1$时的这个构造。在该表的右边是对应的最优树的紧凑说明；例如，当$n=22$时，条目"4:9:9"意味着，具有22片叶的最优树$\mathcal{T}_{22}$，可以通过把$\mathcal{T}_4$、$\mathcal{T}_9$和$\mathcal{T}_9$组合在一起来得到（见图93)，最优树不是唯一的；例如，5:8:9同4:9:9一样地好。

对（2）的推导表明，使用$P+1$个相等的缓冲区域时，关系式$\alpha\leqslant\beta$总成立。当要求寻找时间极小化，而不考虑传输时间时，即出现表1和图93中的极限情况$\alpha=\beta$。

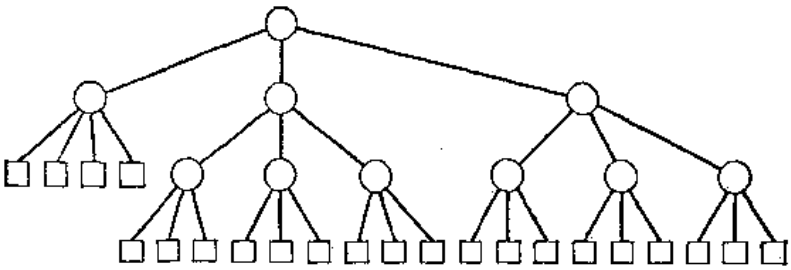


图93　当在定理H中$\alpha=\beta$时，合并22个相等长度的初始路段的一种最优方式。在与正文中的等式（2）同样的假定之下，这个型式使寻找时间极小化

当$\alpha=\beta=1$时最优树特征$A_m(n)$，$h_m(n)$

表1	$m=1$	$m=2$	$m=3$	$m=4$	$m=5$	$m=6$	$m=7$	$m=8$	$m=9$	$m=10$	$m=11$	$m=12$	树的说明	
$n=1$	0,0												—	$n=1$
$n=2$	6,2	0,1											1:1	$n=2$
$n=3$	12,3	6,1	0,1										1:1:1	$n=3$
$n=4$	20,4	12,1	6,1	0,1									1:1:1:1	$n=4$
$n=5$	30,5	18,2	12,1	6,1	0,1								1:1:1:1:1	$n=5$
$n=6$	42,5	24,3	18,1	12,1	6,1	0,1							3:3	$n=6$
$n=7$	52,3	32,3	24,1	18,1	12,1	6,1	0,1						1:3:3	$n=7$
$n=8$	62,3	40,4	30,2	24,1	18,1	12,1	6,1	0,1					2:3:3	$n=8$
$n=9$	72,3	50,4	36,3	30,1	24,1	18,1	12,1	6,1	0,1				3:3:3	$n=9$
$n=10$	84,3	60,5	44,3	36,1	30,1	24,1	18,1	12,1	6,1	0,1			3:3:4	$n=10$
$n=11$	96,3	72,4	52,3	42,2	36,1	30,1	24,1	18,1	12,1	6,1	0,1		3:4:4	$n=11$
$n=12$	108,3	82,4	60,4	48,3	42,1	36,1	30,1	24,1	18,1	12,1	6,1	0,1	4:4:4	$n=12$
$n=13$	121,4	92,4	70,4	56,3	48,1	42,1	36,1	30,1	24,1	18,1	12,1	6,1	3:3:3:4	$n=13$
$n=14$	134,4	102,5	80,4	64,3	54,2	48,1	42,1	36,1	30,1	24,1	18,1	12,1	3:3:4:4	$n=14$
$n=15$	147,4	114,5	90,5	72,3	60,4	54,1	48,1	42,1	36,1	30,1	24,1	18,1	3:4:4:4	$n=15$
$n=16$	160,4	124,7	102,4	80,4	68,3	60,1	54,1	48,1	42,1	36,1	30,1	24,1	4:4:4:4	$n=16$
$n=17$	175,4	134,8	112,4	90,4	76,3	66,2	60,1	54,1	48,1	42,1	36,1	30,1	4:4:4:5	$n=17$
$n=18$	190,4	156,9	122,4	100,4	84,3	72,3	66,1	60,1	54,1	48,1	42,1	36,1	4:4:5:5	$n=18$
$n=19$	205,4	168,9	132,5	110,5	92,3	80,3	72,1	66,1	60,1	54,1	48,1	42,1	4:5:5:5	$n=19$
$n=20$	220,4	168,9	144,4	120,5	100,4	88,3	78,2	72,1	66,1	60,1	54,1	48,1	5:5:5:5	$n=20$
$n=21$	236,5	180,9	154,4	132,4	110,4	96,3	84,3	78,1	72,1	66,1	60,1	54,1	4:4:4:4:5	$n=21$
$n=22$	252,3	192,10	164,4	142,4	120,4	104,3	92,3	84,1	78,1	72,1	66,1	60,1	4:9:9	$n=22$
$n=23$	266,3	204,11	174,5	152,4	130,4	112,3	100,3	90,2	84,1	78,1	72,1	66,1	5:9:9	$n=23$
$n=24$	282,3	216,12	186,5	162,5	140,4	120,4	108,3	96,3	90,1	84,1	78,1	72,1	5:9:10	$n=24$
$n=25$	296,3	229,12	196,7	174,4	150,5	130,4	116,3	104,4	96,1	90,1	84,1	78,1	7:9:9	$n=25$

回到最初的应用，我们尚未考虑如何首先得到初始的路段；如果没有读/写/计算的重叠，则替代选择就失去了它的某些优点。也许我们将填满整个内存，然后排序并输出结果；每一个这样的输入和输出操作都只需要一次寻找。或者也许拿出比如说内存的百分之二十作为一个综合的输入/输出缓冲区，并进行替代选择倒更好。这要求五倍的寻找时间（一个额外的60秒左右时间！），但它使初始路段的数目从100减少到64；如果输入文件原来就是相当有序的，则减少就更为显著。

如果我们决定不使用替代选择，则对于$S=100$，$\alpha=0.00145$，$\beta=0.01545$的最优

树〔参考(2)〕是相当平凡的：它只不过是对数据在两次扫描中完成的一个10路合并。如果允许对内部排序花费30秒（比如说，100次快速排序），则初始分配扫描大约花费$2\frac{1}{2}$分，每次合并扫描将近花费5分，总共是12.4分。如果我们决定使用替代选择，则对于$S=64$的最优树也同样是平凡的（两个8路合并扫描）；初始分布扫描大约花费$3\frac{1}{2}$分，每次合并扫描大约花费$4\frac{1}{2}$分，估计总共的时间为12.6分。别忘了，这两个方法实际上都废弃了所有的读/写/计算的重叠，为的是有更大的缓冲区（减少寻找时间）。估计这些时间时都没有包括为"向后读检验"操作可能需要的时间。

实际上，最后的合并扫描往往十分不同于其它几次扫描；例如，输出常常被展开和/或写到带上。在这样的情况下，树型式应该利用在根处的另一种最优性准则来加以选择。

***分配缓冲区的另一种方式** 戴维·E.弗格森〔CACM 14（1971），476-478〕已经指出，如果我们不把所有缓冲区都作成相同大小，则寻找时间即可减少。同样的思想大约在同一个时间也出现在其他人的文章中〔S.J.Waters，Comp.J.14(1971)，109-112；Ewing S.Walker，Software Age 4（August-September，1970），16-17〕。

假设我们正在对具有相等长度L_0的路段进行四路合并，且内存可容纳M个字符。如果把内存划分成相同的缓冲区大小$B=M/5$，则对于每个输入文件需要L_0/B次寻找，对于输出需要$4L_0/B$次寻找，加起来为$8L_0/B=40L_0/M$次寻找。但如果我们用四个大小为$M/6$的输入缓冲区和一个大小为$M/3$的输出缓冲区，则只需要大约$4\times(6L_0/M)+4\times(3L_0/M)=36L_0/M$次寻找！在两种情况下传输时间是相同的，所以在后一种情况下，我们并不受任何损失。

一般地说，假设要把长度为L_1，…，L_P的排序文件合并成为长度为$L_{P+1}=L_1+\cdots+L_P$的一个排序文件，而且假定一个大小为B_k的缓冲区被用于第k个文件。于是

$$B_1+\cdots+B_P+B_{P+1}=M \tag{6}$$

其中M是可利用的内存的总容量，寻找的次数将近似为

$$\frac{L_1}{B_1}+\cdots+\frac{L_P}{B_P}+\frac{L_{P+1}}{B_{P+1}} \tag{7}$$

现在让我们在条件（6）的约束下使这个量极小化，为方便起见假定B_k不必为整数。如果对B_j增加δ而对B_k减少相同的小量，则寻找次数的改变是

$$\frac{L_j}{B_j+\delta}-\frac{L_j}{B_j}+\frac{L_k}{B_k-\delta}-\frac{L_k}{B_k}=\left(\frac{L_k}{B_k(B_k-\delta)}-\frac{L_j}{B_j(B_j+\delta)}\right)\delta$$

所以如果$L_j/B_j^2\neq L_k/B_k^2$，这个分配就能改进。因此，仅当

$$\frac{L_1}{B_1^2}=\cdots=\frac{L_P}{B_P^2}=\frac{L_{P+1}}{B_{P+1}^2} \tag{8}$$

时我们得到极小的寻找次数。由于存在一个极小，因此当

$$B_k=\sqrt{L_k}M/(\sqrt{L_1}+\cdots+\sqrt{L_{P+1}}),\quad 1\leqslant k\leqslant P+1 \tag{9}$$

时它应该出现，因为这些是$B_1,\cdots,B_{P+1}$的同时满足（6）和（8）的仅有的值。把这些值代入（7），就给出寻找总次数的相当简单的公式

$$(\sqrt{L_1}+\cdots+\sqrt{L_{P+1}})^2/M \tag{10}$$

它可以同当所有缓冲区长度都相等时得到的数 $(P+1)(L_1+\cdots+L_{P+1})/M$ 相对照。按习题 1.2.3-31，得到的改进为 $\sum_{1\leqslant j<k\leqslant P+1}(\sqrt{L_j}-\sqrt{L_k})^2/M$。可惜，不能象在定理K中那样容易地利用公式（10）确定最优合并形式（参照习题 14）。

进一步的考虑 如果使用两个磁盘设备，一个用来读和一个用来写，则用一个额外的输出缓冲区我们即可重叠某些写的时间。如果使用“预报”技术（即，考察每个输入缓冲区中最后的键，以便确定哪一个缓冲区将首先变空），则又可重叠某些读的时间。可惜，一个象算法 5.4.6F 的简单同步过程，并不提供连续的输入/输出，因为寻找时间的长短可以很不一样，这意味着输入和输出操作不会在同一时间结束。重叠的准确数值难以估计，因此如同在定理H中那样模拟合并过程以求出近似于运行时间的常数 α 和 β，可能是最好的。

如果我们使用“浮动”的输入缓冲区（不专门供给一个特定的文件），则它们就必须都有相同的大小，所以就不能使用上述的缓冲区分配技术，只有把输出缓冲区选成大于输入缓冲区这一点尚可应用。如果使用 $2P$ 个缓冲区而不是 P 个，则重叠了大量的输入时间但加倍了寻找时间，所以我们可能毫无所得；也许 $P+1$ 或 $P+2$ 个输入缓冲区倒是最好的。

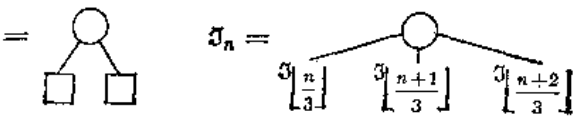
现在读者应当清楚了：对于最优磁盘排序，至今未研究出界线分明的策略；可利用的选择数大大超过了理论上已经分析过的策略数。尽管在这节中探讨的理论提供了某些有用的知识，但仍尚待进行大量的实验。

***更深入地考察最优树** 虽然在实际的情况下参数 α 和 β 都满足 $0\leqslant\alpha\leqslant\beta$，考察在定理H和K中的极端情况 $\beta=0$ 还是有趣的，什么样的具有 n 片叶的树有最小的叉数路径长度？奇怪，结果证明三路合并在这种情况下是最好的。

定理 L 具有 n 片叶子的一株树的叉数路径长度决不小于

$$f(n)=\begin{cases}3qn+2(n-3^q), & \text{如果 } 2\cdot 3^{q-1}\leqslant n\leqslant 3^q\\ 3qn+4(n-3^q), & \text{如果 } 3^q\leqslant n\leqslant 2\cdot 3^q\end{cases} \tag{11}$$

由规则

$\mathcal{T}_1=$ 

$$\mathcal{T}_n=\mathcal{T}_{\lceil\frac{n}{3}\rceil}\quad\mathcal{T}_{\lceil\frac{n+1}{3}\rceil}\quad\mathcal{T}_{\lceil\frac{n+2}{3}\rceil} \tag{12}$$

定义的三叉树 $\mathcal{T}_n$ 有极小的叉数路径长度。

证明 重要的是，注意 $f(n)$ 是一个凸函数，即

$$f(n+1)-f(n)\geqslant f(n)-f(n-1)\qquad \text{对于所有 } n\geqslant 2 \tag{13}$$

这个性质的正确性由下列引理得出。

引理 C 在正整数上定义的一个函数 $g(n)$ 满足

$$\min_{1\leqslant k\leqslant n}(g(k)+g(n-k))=g(\lfloor n/2\rfloor)+g(\lceil n/2\rceil),\quad n\geqslant 2 \tag{14}$$

的充要条件是它是凸的。

证明 如果对某个 $n\geqslant 2$，$g(n+1)-g(n)<g(n)-g(n-1)$，则我们有 $g(n+1)+g(n-1)<g(n)+g(n)$，同（14）矛盾。反之，如果（13）对 g 成立，且 $1\leqslant k<n-k$，则由凸性我们有 $g(k+1)+g(n-k-1)\leqslant g(k)+g(n-k)$。

引理C的证明的后面部分可以推广到任何的$m \geqslant 2$，以证明每当g为凸时

$$\min_{\substack{n_1+\cdots+n_m=n \\ n_1,\cdots,m_m\geqslant 1}} (g(n_1)+\cdots+g(n_m)) = g(\lfloor n/m \rfloor + g(\lfloor n+1/m \rfloor)+\cdots+g(\lfloor n+m-1/m \rfloor) \tag{15}$$

令

$$f_m(n) = f(\lfloor n/m \rfloor)+f(\lfloor (n+1)/m \rfloor) + \cdots + f(\lfloor (n+m-1)/m \rfloor) \tag{16}$$

通过证明$f_3(n)+3n=f(n)$及对于所有$m \geqslant 2$，$f_m(n)+mn \geqslant f(n)$便完成了定理L的证明（见习题11）。

如果最优树总是象在定理L中那样干净利落地表征，那该有多好。但是在表1中对于$\alpha=\beta$我们已经看到的结果表明，函数$A_1(n)$并不总是凸的。事实上，表1已经足以否定关于最优树的大多数简单的猜测！然而，在一般情况下，我们可以部分地挽救定理L：M.施卢姆伯杰（M. Schlumberger）和J.武维列明（J. Vaillemin）已经证明，总可以避免大阶数的合并。

定理M 象在定理H中那样给定α和β，则存在一株最优的树，其中每个节点的叉数至多为

$$d(\alpha,\beta)=\left\lceil \min_{k\geqslant 1}\left(k+\left(1+\frac{1}{k}\right)\left(1+\frac{\beta}{\alpha}\right)\right)\right\rceil \tag{17}$$

证明 设$n_1,\cdots,n_m$是正整数，使得$n_1+\cdots+n_m=n$，$A(n_1)+\cdots+A(n_m)=A_m(n)$，且$n_1\leqslant\cdots\leqslant n_m$，并假设$m\geqslant d(\alpha,\beta)+1$，令$k$是使（17）为极小的值；我们将证明

$$\alpha n(m-k)+\beta n+A_{m-k}(n)\leqslant \alpha nm+\beta n+A_m(n) \tag{18}$$

因此（4）中的极小值总能对于某个$m\leqslant d(\alpha,\beta)$达到。

按定义，由于$m\geqslant k+2$，我们必然有

$$\begin{aligned} A_{m-k}(n) &\leqslant A_1(n_1+\cdots+n_{k+1})+A_1(n_{k+2})+\cdots+A_1(n_m)\leqslant \alpha(n_1 \\ &\quad +\cdots+n_{k+1})(k+1)+\beta(n_1+\cdots+n_{k+1})+A_1(n_1) \\ &\quad +\cdots+A_1(n_m)=(\alpha(k+1)+\beta)(n_1+\cdots+n_{k+1}) \\ &\quad +A_m(n)\leqslant(\alpha(k+1)+\beta)(k+1)n/m+A_m(n) \end{aligned}$$

故现在容易得出（18）。（小心地检查这个证明看出，就某些最优的树必须有叉数为$d(\alpha,\beta)$的节点说来，（17）是“最好的”；见习题13。）

定理K中的构造要求$O(N^2)$个存储单元和$O(N^2\log_2 N)$个步骤来计算$A_m(n)$，其中$1\leqslant m\leqslant n\leqslant N$；定理M表明仅仅需要$O(N)$个单元和$O(N^2)$个步骤。施卢姆伯杰和武维列明已经发现了最优树的若干非常有趣的性质〔Acta Informatica 3(1975)，25-36〕。而且$A_1(n)$的渐近值可如习题9中所示那样算出。

链的使用 M.A.戈茨〔CACM 6 (1963), 245-248〕已经提出了通过把个别的道链接在一起，来避免输出时的寻找时间的一种有趣的方法。他的思想要求一组相当独特的磁盘存储管理例程，但它可应用于除排序之外的许多问题，因而它可能是一项非常有价值的通用技术。

概念是简单的：我们不在磁盘的圆柱面上顺序地分配道，而是把这些道链接在一起并且建立一张可用空间的表，对于每个圆柱面都有一张这样的表。当到了输出一道信息的时候，我们就把它写到当前的柱面上（即存取臂当时所在的位置），除非该圆柱面已经满了。这样一来，一般来说寻找时间就不需要了。

困难在于我们不能在一个道本身之内存储对下一个道的链接，因为所需的信息在当时是未知的（如果合适的话，我们可以存储一个对以前的道的链接，而且在下次扫描时向后读文件）。可以保留一张独立的表，记载每个文件的诸道间的链接信息，也许部分地就保留在盘本身上。通过使用二进位表可以紧凑地表示可用空间表，用 1000 位指明 1000 个道的可用性或不可用性。

键排序有邦助吗？ 当记录是长的而键是短的时，建立一个只是由键和指明它们原始文件位置的顺序编号所组成的新文件，是非常诱人的。在对这个键文件排序之后，我们可以用逐个相连的数 1，2，…来代替诸键；于是新文件就可以通过原始的文件位置来进行排序，而且我们能方便地说明怎样整理和最后重新排列记录。这个过程可列表表示如下：

a）原始文件	$(K_1, I_1)(K_2, I_2)\cdots(K_N, I_N)$	长
b）键文件	$(K_1, 1)\ (K_2, 2)\cdots(K_N, N)$	短
c）排好序的（b）	$(K_{p_1}, p_1)(K_{p_2}, p_2)\cdots(K_{p_N}, p_N)$	短
d）编辑好的（c）	$(1, p_1)(2, p_2)\cdots(N, p_N)$	短
e）排好序的（d）	$(q_1, 1)(q_2, 2)\cdots(q_N, N)$	短
f）编辑好的（a）	$(q_1, I_1)(q_2, I_2)\cdots(q_N, I_N)$	长

这里 $p_j = k$ 当且仅当 $q_k = j$；（c）、（e）中的两个排序过程比较快（也许就是内部排序），因为诸记录不是非常长的。在阶段（f）中，我们已经把这个问题归结为对一个其键简单地是数 $\{1, 2, \cdots, N\}$ 的文件排序的问题；现在每个记录都恰当地指定了它有待移去的地方。

在阶段（f）之后尚留下的外部重排问题，乍看起来似乎是显然的；但事实上它是颇为困难的，而且还没有找到真正好的（比排序要好得多）的算法。我们当然可以在 N 步之内来进行重排，一次移动一个记录；对于充分大的 N，这比一个排序方法的 $N\log_2 N$ 要好，但是 N 决不是这么大的，而 N 个寻找是不可思议的。

基数排序方法可有效地用于已编辑好的记录上，因为它们的键有一个完全一致的分布。在许多具有高速外部设备的高速计算机上，八路分布的处理时间比八路合并的处理时间要快得多；因此一个分布排序可能是最好的过程（见 5.4.7 节，也见习题 17）。

另一方面，仅仅为了进行一次完全排序而先进行一次键排序似乎是浪费！外部重排问题意外地困难的一个原因，已为 R. W. 弗洛伊德所发现，他找出了为在一个磁盘文件上重排记录所需要的寻找次数的非显然的下限〔Complexity of Computer Computations (New York: Plenum, 1972), 105-109〕。

借助于 5.4.8 节的电梯问题来描述弗洛伊德的结果是方便的；这一次我们要求一个使停止次数取极小而不是旅行距离取极小的电梯调度（使停止次数取极小严格地说并不等价于求极小——寻找重排算法，因为每次停止都把输入电梯同从电梯输出联合起来；但是停止极小化准则和这里的基本思想非常接近。

我们将使用“离散熵”函数

$$F(n)=\sum_{1<k\leqslant n}(\lceil\log_2 k\rceil+1)=B(n)+n-1=n\lceil\log_2 n\rceil-2^{\lceil\log_2 n\rceil}+n \tag{19}$$

其中$B(n)$是“二叉插入”函数（见等式5.3.1-3）。由等式5.3.1-33，$F(n)$是具n片叶子的一株二叉树的极小外部路径长度，且

$$n\log_2 n\leqslant F(n)\leqslant n\log_2 n+0.0861n \tag{20}$$

因为$F(n)$是凸的且满足$F(n)=n+F(\lfloor n/2\rfloor)+F(\lceil n/2\rceil)$，通过上述的引理C我们知道

$$F(n)\leqslant F(k)+F(n-k)+n,\quad 0\leqslant k\leqslant n \tag{21}$$

从F的外部路径长度特征来看，这个关系也是明显的；这个关键性的事实下面还要用到。

如同在5.4.8节中那样，我们将假定电梯装b个人，每层保持c个人，且有n层。令s_{ij}是当前在层i其目的地是层j的人数，在建筑物中人员的任何配置的集中率定义为和式$\sum_{1\leqslant i,j\leqslant n}F(s_{ij})$。

例如，假定$b=c=n=6$并且开始时36个人按如下方式散布在各层中：

```
␣␣␣␣␣␣
123456  123456  123456  123456  123456  123456
```
(22)

电梯是空的，停在层1；“␣”表示一个空位置。在每一层上，对每种可能的目的地，都有一人相应。故所有s_{ij}为1且集中率为0。如果现在电梯运送6个人到层2，则我们有配置

```
                123456
␣␣␣␣␣␣ 123456  123456  123456  123456
```
(23)

且集中率变成$6F(0)+24F(1)+6F(2)=12$。设电梯运1，1，2，3，3，4到层3：

```
                112334
␣␣␣␣␣␣ 245566  123456  123456  123456  123456
```
(24)

则集中率变成$4F(2)+2F(3)=18$。当每个人最终被运送到他或她的目的地时，集中率将是$6F(6)=96$。

弗洛伊德发现，在每次停止时，集中率的增加决不能多于$b+c$，因为一组s个相同目的地的人同一组s'个类似的人合在一起，使集中率增进$F(s+s')-F(s)-F(s')\leqslant s+s'$。因此我们有下列结果。

定理F 设t是在上述定义下的人的初始配置的集中率，则为了把他们全部运到他们的目的地，电梯至少必须停

$$\lceil(nF(c)-t)/(b+c)\rceil$$

次。

把这个结果翻译成磁盘术语，设有N个记录，且每个块区有B个记录，并假设内部存储可以一次保存M个记录。每一次读盘都把一个块区读到内存，每一次写盘都储存一个块区到盘中，而s_{ij}是块区i中属于块区j的记录的数目。如果$N\geqslant B^2$，则存在这样的初始配置，其中所有$s_{ij}\leqslant 1$，所以为了重排诸记录至少需要$(N/B)F(B)/M\approx(N\log_2 B)/M$个读块区操作（当$B$很大时，因子$\log_2 B$使得这个下限成为非显然的，但当$B=1$时这

个下限肯定可以加以改进)。

习题

1.〔$M22$〕 正文说明了一个方法，用这个方法，为读一个道的一部分（大小为x所需要的平均等待时间，就从$\frac{1}{2}$减少到$\frac{1}{2}(1-x^2)$圈转动。当只有一个存取臂时，这是可能得到的最小值。如果有两个存取臂，相距180°，则对应的极小平均等待时间是什么（假定在任何一个时刻仅有一个臂可传送数据)?

2.〔$M30$〕（A·G·康海姆） 本题的目的是研究在合并“正交地”分配于圆柱面的文件时，一个盘的存取臂必须移动多远？假设有P个文件，每个含L个记录块区，并且假定每个文件的头一个块区都出现在圆柱面1上，第二个出现在圆柱面2上，等等。在每个块区中最后那些键的相对次序支配了在合并期间存取臂的运动，因此我们可以下列数学上易处理的方式来表示这种情况：考虑PL个有序对偶的集合

$$\begin{array}{cccc}
(a_{11},1) & (a_{21},1) & \cdots & (a_{p1},1) \\
(a_{12},2) & (a_{22},2) & \cdots & (a_{p2},2) \\
\vdots & \vdots & & \vdots \\
(a_{1L},L) & (a_{2L},L) & \cdots & (a_{pL},L)
\end{array}$$

其中集合$\{a_{ij}|1\leqslant i\leqslant P,\ 1\leqslant j\leqslant L\}$是由在某种顺序下的数$\{1,2,\cdots,PL\}$组成的、而且对于$1\leqslant j<L$，$a_{ij}<a_{i,j+1}$（行表示圆柱面，列表示输入文件)。按它们的头一个分量对这些对偶排序并设得到的序列为$(1,j_1)(2,j_2)\cdots(PL,j_{PL})$。证明，如果$a_{ij}$的$(PL)!/L!^P$种选择的每一种是同等可能的，则

$$|j_2-j_1|+|j_3-j_2|+\cdots+|j_{PL}-j_{PL-1}|$$

的平均值是
$$(L-1)\left(1+(P-1)2^{2L-2}\Big/\binom{2L}{L}\right)$$

〔提示：见习题5.2.1-14〕 注意，当$L\to\infty$时，这个值渐近地等于$\frac{1}{4}(P-1)L\sqrt{\pi L}$。

3.〔$M15$〕 假设内存太小使得10路合并是不可行的，问怎样能修改递归关系（3)、(4)、(5)，使在所有的具有n片叶子且没有叉数大于9的内部节点的树中，$A_1(n)$是$\alpha D(\mathcal{T})+\beta E(\mathcal{T})$的极小值?

▶4.〔$M21$〕（a）平方根缓冲区分配的一种修正形式是，所有P个输入缓冲区都有相等长度，但输出缓冲区的大小应当选择成使寻找时间成为极小。若对一个L字符的P路合并使用这种修正形式，试导出对应于（2）的一个运行时间公式。

（b）证明定理K中的构造可以被修改，以便得到一个合并型式，这个合并型式按照（a）中所得的公式来看是最优的。

5.〔$M20$〕 当我们正在使用两个磁盘，使在一个上读同在另一个上写重叠时，我们不能使用如图93那样的合并型式，因为某些叶在偶数级上，而某些叶却在奇数级上。试说明怎样修改定理K的结构，以便产生出在所有叶都出现于偶数级上或所有的叶都出现于奇数级上这样的约束条件下为最优的树。

▶6.〔22〕 找出一株树，它在习题5的意义下当$n=23$和$\alpha=\beta=1$时为最优。(你

可能希望使用一台计算机。)

▶7.〔M24〕 当初始路段不都是同样长度的时候，最好的合并型式（在定理 H 的意义下）使 $\alpha D(\mathcal{T})+\beta E(\mathcal{T})$ 成为极小，其中 $D(\mathcal{T})$ 和 $E(\mathcal{T})$ 现在表示加权的路径长度：权 $w_1,\cdots,w_n$（对应于初始路段的长度）被附加于树的每片叶上，而叉数的和与路径长度被乘以适当的权。例如，如果 $\mathcal{T}$ 是图 92 的树，则我们将有 $D(\mathcal{T})=6w_1+6w_2+7w_3+9w_4+9w_5+7w_6+4w_7+4w_8$，$E(\mathcal{T})=2w_1+2w_2+2w_3+3w_4+3w_5+2w_6+w_7+w_8$。

证明总存在一最优的型式，其中，对于某个 k，最短的 k 个路段首先被合并。

8.〔49〕 是否有一个这样的算法，对于某个 c，仅取 $O(n^c)$ 个步骤，它就能找出对于给定的 α、β 和权 $w_1,\cdots,w_n$，在习题 7 意义下的最优树来。

9.〔HM39〕（L. 海亚菲尔（L. Hyafil），F. 普鲁斯克尔（F. Prusker），J. 武维列明）证明：对于固定的 α 和 β，当 $n\to\infty$ 时

$$A_1(n)=\left(\min_{m\geqslant 2}\frac{\alpha m+\beta}{\log_2 m}\right)n\log_2 n+O(n)$$

这里项 $O(n)\geqslant 0$。

10.〔HM44〕（L. 海亚菲尔，F. 普鲁斯克尔，J. 武维列明）证明：当 α 和 β 是固定的时，如果 m 使习题 9 中的系数成为极小，则对于所有充分大的 n，$A_1(n)=\alpha mn+\beta n+A_m(n)$。

11.〔M29〕 在（11）和（16）的记号下，证明对于所有的 m，$n\geqslant 2$，$f_m(n)+mn\geqslant f(n)$，并确定使等式成立的所有 m 和 n。

12.〔25〕 证明对所有 $n>0$，存在具有 n 个叶和极小叉数路径长度（11）的一株树，其所有的叶都在同一级上。

13.〔M24〕 证明对于 $2\leqslant n\leqslant d(\alpha,\beta)$，在定理 H 的意义下唯一最好的合并型式是一个 n 路合并（参考（17））。

14.〔40〕 如果利用缓冲区分配的平方根方法，则对于图 92 中的合并型式的寻找时间将同 $(\sqrt{2}+\sqrt{4}+\sqrt{1}+\sqrt{1}+\sqrt{8})^2+(\sqrt{1}+\sqrt{1}+\sqrt{2})^2+(\sqrt{1}+\sqrt{2}+\sqrt{1}+\sqrt{4})^2+(\sqrt{1}+\sqrt{1}+\sqrt{2})^2$ 成正比；这是 $(\sqrt{n_1}+\cdots+\sqrt{n_m}+\sqrt{n_1+\cdots+n_m})^2$ 关于每个内部节点的和，其中分别与诸节点对应的子树有 $(n_1,\cdots,n_m)$ 片叶。试写出一个计算机程序，它生成有 1，2，3，…片叶的极小寻找时间的树，并利用这个公式来估计寻找时间。

15.〔M22〕 证明，如果电梯开始时为空，且如果 $nF(c)\neq t$，则定理 F 可稍加改进：在这样的情况下，至少需要停 $\lceil(nF(c)+b-t)/(b+c)\rceil$ 次。

16.〔23〕（R. W. 弗洛伊德） 试求一电梯的调度表，它把（22）中的所有人都运送到他们的目的地而停的次数不超过 12 次（配置（23）显示了一次停止，而不是两次停止之后的情形）。

17.〔25〕（B. T. 贝内特（B. T. Bennet）和阿·查·麦凯勒，1971）考虑以具有 10 个键的一个文件作说明的下列键排序方法：

a）原始的文件：(50, I_0)(08, I_1)(51, I_2)(06, I_3)(90, I_4)(17, I_5)(89, I_6)(27, I_7)(65, I_8)(42, I_9)

b）键文件：(50, 0)(08, 1)(51, 2)(06, 3)(90, 4)(17, 5)(89, 6)(27, 7)(65, 8)(42, 9)

c）排好序的（b）：(06,3)(08,1)(17,5)(27,7)(42,9)(50,0)(51,2)(65,8)(89,6)(90,4)

d）箱子分配：(3,3)(3,9)(2,1)(2,5)(2,7)(2,8)(1,0)(1,2)(1,4)(1,6)

e）排好序的(d)：(1,0)(2,1)(1 2)(3,3)(1,4)(2,5)(1,6)(2,7)(2,8)(3,9)

f）用（e）把（a）分配到诸箱中：

箱1：$(50,I_0)(51,I_2)(90,I_4)(89,I_6)$

箱2：$(08,I_1)(17,I_5)(27,I_7)(65,I_8)$

箱3：$(06,I_3)(42,I_9)$

g）先读箱3，然后箱2，最后箱1，得到如下的替代选择结果：$(06,I_3)(08,I_1)(17,I_5)(27,I_7)(42,I_9)(50,I_0)(51,I_2)(65,I_8)(89,I_6)(90,I_4)$

在步骤（d）中箱号的分配通过对（c）作替代选择，从右到左地，而且以第二个分量递降的次序来进行。箱号是路段号。上述例子使用在选择树中仅有两个元素的替代选择；在（d）和（g）中应用相同尺寸的树进行替代选择。注意，箱的内容不是事先排好序的。

证明这个方法将完成排序，即，（g）中的替代选择将仅仅产生一个路段。（这项技术减少了通常的用分布来进行键排序时所需要的箱数，特别是如果输入已经相当有次序的情况下。）

▶18.〔25〕 某些硬件/软件系统提供给程序员一个虚拟存储：程序可以被写成如同有一个非常大的、能包含所有数据的内存那样。这个内存被划分成页，在任何一个时刻，仅仅很少的页在实际的内存当中，其余的则在盘或鼓上。但程序员不必去关心这样的细节，因为系统会照管每件事情；当需要时，新的页就自动地被装入内存中。

虚拟存储技术的出现似乎使得外部排序方法成为过时了，因为利用对于内部排序所发展的技术，这个作业即可简单地完成。试讨论这种情况；问在什么方式下，一个精工巧制的外部排序方法要胜过一个应用通用分页技术的内部排序方法？

19.〔$HM25$〕（R.W.弗洛伊德）证明当 $n=b^k$，其中 k 固定而 $b\to\infty$ 时，定理F的下界可以改进成为

$$\frac{(k+1)nb\log_2 b+nb/\ln 2}{b+c}\left(1+O\left(\frac{\log_2 b}{b}\right)\right)$$

而对于 b 固定 $n\to\infty$ 时，也可改进成为 $nb+O(n/\log_2 n)$，这是在某个初始的配置必定至少要求这么多次停止的意义下给出的。〔提示：计算在 s 次停止之后可以排序的配置数。〕

5.5 小结、历史和文献目录

现在已经接近这极为冗长的一章的结尾，我们最好“排序出”已经研究过的最为重要的事实。

用于排序的一个算法，是一个这样的过程，它重新安排一个文件的诸记录，使得诸键处于递增的次序下。这有序的排列是有用的，因为它把相同键的记录弄到一块，允许有效地处理按同一键排好序的多重文件，这导致了有效的检索算法，而且使计算机的输出看上去更为可靠。

当所有的记录都填入计算机的高速内存中时，就使用内部排序。我们对于内部排序已经研究了两打以上的算法，其详细程度各不相同；而且也许，如果并不知道对于这个问题有这样多不同的方法，我们反倒快活些！学习所有这些技术是有趣的，但是现在我们却面临着在一个给定的情况下，应该实际决定使用哪一种方法的令人烦恼的前景。

如果不论是对于哪一种应用或不论正在使用的是什么计算机，仅仅有一两种排序方法，它比所有其它的排序方法都好，那么，事情倒是很好的。但是事实上，每种方法都有它本身特殊的优点。例如，气泡排序（算法 5.2.2B）并没有明显可取的特性，因为总有一个更好的方法来做它所能做的；但即使是这样一项技术，在作适当的推广以后，证明对于两条带的排序是有用的（参考 5.4.8 节）。因此我们发现，几乎所有的算法都值得记住，因为在某些应用中，它们是最好的。

以下简短的总结，给出了对于内部排序所已遇到过的最有意义的算法的要旨。和通常一样，N表示在给定的文件中的记录数。

1．分布计数（算法 5.2D）当键的变化范围较小时，是非常有用的。它是稳定的（不影响具有相同键的记录的次序），但要求为计数器和 $2N$ 个记录所需的存储空间。习题 5.2-13 中给出了在损失稳定性的代价下节省N个记录空间的一项修改。

2．直接插入（算法 5.2.1S）对于编程序说来是最简单的方法，不需要额外的空间，而且对于较小的N十分有效（比如说$N\leqslant 25$）。对于大的N，它慢得不能容忍，除非输入是接近于有序的。

3．递减增量的排序（算法 5.2.1D（谢尔方法））也十分容易编程序，而且使用极小的内存空间；它对于适当大的N（比如说$N\leqslant 1000$）相当有效。

4．表插入（算法 5.2.1L）使用和直接插入相同的基本思想，所以它仅适合于小的N。如同以下所述的其它表排序方法一样，它通过处理链接减少了移动长记录的耗费；当记录有可变长度或者是其它数据结构的一部分时，特别有利。

5．地址计算技术当键有一个已知的（通常是一致的）分布时是很有效的。这个方法主要的变种是多表插入（算法 5.2.1M），以及麦克拉伦的组合基数插入方法（在 5.2.5 节的结尾处讨论的），后者仅用$O(\sqrt{N})$个额外的内存单元来完成。

6．合并交换（算法 5.2.2M（巴切尔方法））和它的“表姐妹”双调排序（习题 5.3.4-10），当大量的比较可同时进行时很有用。

7．分划交换（算法 5.2.2Q（霍尔方法，普通称为快速排序））大概是内部排序中最有用的通用技术，因为它需要非常少的内存空间，而且在大多数计算机上它的平均运行时间少于它的对手的平均运行时间。然而，在最坏的情况下，它能运行得非常之慢，所以每当可能是非随机的数据时，应当小心地选择分划的元素。如同习题 5.2.2-55 中所建议的，选择三个元素的中间值，使最坏的情况很可能不发生，同时也对平均运行时间有所改进。

8．直接选择（算法 5.2.3-S）当有特殊硬件可用来高速寻找一个表的最小元素时，是一个特别合适的简单方法。

9．堆排序（算法 5.2.3H）需要极小的内存，而且保证十分快地运行；它的平均时间和极大时间都大体上是快速排序平均运行时间的两倍。

10．表合并（算法 5.2.4L）是一个表排序，它和堆排序相象，保证甚至在最坏的情

况下也是相当快的，而且对于相等的键它是稳定的。

11．基数排序（使用算法 5.2.5R）特别适合于那样键的一种表排序，这些键或者比较短，或者是按一个非通常的字典顺序整理的序列，或者，也可以使用分布计数方法（见上面的 1）以代替链接；这样一个过程需要 $2N$ 个记录空间，加上一个计数器表，但是它的内部循环的简单形式，使它特别适用于超高速的“吞噬数据”的有先行控制的计算机。警告：基数排序不应对小 N 使用。

12．合并插入（见 5.3.1 节）在一个“直接式编码”的例程中，特别适合于非常小的 N。例如，在需要对大量五个或六个记录组排序的一种应用中，它将是适当的方法。

13．混合的方法（把上述的一个或多个技术组合在一起）也是可能的。例如，合并插入可用于快速排序中出现的短的子文件排序。

14．最后，一个未命名的出现在习题 5.2.1-3 的答案中的方法，似乎对应于最短的程序。但它的平均运行时间同 N^3 成正比，这就使它成为本书中最慢的排序例程！

表 1 综述了当为MIX进行程序设计时，这些方法的速度和空间特征。重要的是要认识到，这个表中的数字仅仅是相对的排序时间的粗略表示；它们仅应用于一台计算机上，而且关于输入数据所作的假定对所有的程序来说并不完全一致。和这个表相当的一些比较表已为许多作者给出，而且没有两个人给出相同的结论来！另一方面，这些时间估计至少给出当对一个字的记录排序时，对每个算法可能预期的大体速度，因为MIX是一台相当典型的计算机。

表 1 中的“空间”列，给出了每个算法所使用的辅助存储数量的某些信息，它以记录长度为单位，其中“ϵ”表示记录中链接场部分所需的长度。于是，例如，$N(1+\epsilon)$ 意味着这个方法要求 N 个记录加上 N 个链接场的空间。

表 1 中出现的渐近平均时间和极大时间仅仅给出了当 N 很大时占支配地位的前导项，且假定随机输入；c 表示一个未确定的常数。这些公式可能常常会引起误解，所以特地对两个特殊的输入数据序列，列出了程序实际的总运行时间。$N=16$ 的情况指的是 16 个键，它出现在 5.2 节的许多例子中；$N=1000$ 的情况指的是由

$$K_0=0;\ K_{n+1}=(3141592621K_n+2113148651) \mod 10^{10}$$

所定义的序列 $K_1, K, \cdots, K_{1000}$。一个相当“高质量”的MIX程序已经用来表示表中的每个算法，通常还加上了在习题中所建议的改进。这些程序运行时的字节大小是 100。

外部排序技术不同于内部排序，因为它们必须使用比较原始的数据结构，并且着重强调把它们的输入/输出时间极小化。5.4.6 节综述了对于带合并已经发展的有趣的方法，5.4.9 节讨论了盘和鼓的使用。

当然，排序并不是事情的全部。在研究所有这些排序技术时，我们已经学习了大量的有关如何处理数据结构、如何处理外存、以及如何分析算法等问题；而且，也许我们甚至已经学会了一点如何来发现新的算法的本领。

早期的发展 对于今天的排序技术起源的探索，把我们带回到 19 世纪，那时发明了头一批用于排序的机器。美国每十年进行一次全国所有居民的人口普查，而到 1880 年左右处理庞大的人口普查数据的问题已经变得非常尖锐；事实上，独身者（不同于结了婚的）的总数总不能在当年就造出表，即使所需要的信息都已经收集好了也是如此。赫尔曼·

表1　利用MIX计算机对内部排序方法进行的比较

方法	参考	稳定否?	MIX程序的长度	空间	运行时间 平均	运行时间 极大	运行时间 $N=10$	运行时间 $N=1000$	注
比较计数	习题5.2-5	是	22	$N(1+\epsilon)$	$4N^2+10N$	$5.5N^2$	1065	3992432	c
分布计数	习题5.2-9	是	26	$2N+1000\epsilon$	$22N+10010$	$22N$	10362	32010	a
直接插入	习题5.2.1-33	是	10	$N+1$	$1.5N^2+9.5N$	$3N^2$	412	1491928	
递减增量	程序5.2.1-D	否	21	$N+\epsilon\log_3 N$	$15N^{1.25}+10\log_3(N/3)$	$cN^{1.6}$	567	137502	d，h
表插入	习题5.2.1-33	是	19	$N(1+\epsilon)$	$1.25N^2+13.25N$	$2.5N^2$	433	1248615	b，c
多表插入	程序5.2.1M	否	18	$N+\epsilon(N+100)$	$0.0175N^2+18N$	$3.5N^2$	645	35246	b，c，f，i
合并交换	习题5.2.2-12	否	36	N	$3.7N(\log_2 N)^2$	$cN(\log_2 N)^2$	1284	379104	h
分划交换	程序5.2.2Q	否	63	$N+2\epsilon\log_2 N$	$11.67N\ln N-1.94N$	$\geqslant 2N^2$	470	81486	
3个取中快速排序	习题5.2.2-55	否	91	$N+2\epsilon\log_2 N$	$10.63N\ln N+2.11N$	$\geqslant N^2$	487	74574	e
基数交换	程序5.2.2R	否	45	$N+68\epsilon$	$11.43N\ln N+23.9N$	$272N$	1135	137614	g，i，j
直接选择	程序5.2.3S	是	15	N	$2.5N^2+3N\ln N$	$3.25N^2$	853	2525287	j
堆排序	程序5.2.3H	否	30	N	$23.08N\ln N+0.2N$	$<26N\ln N$	1068	159714	h，j
表合并	程序5.2.4L	是	44	$N(1+\epsilon)$	$14.43M\ln N+4.92N$	$14.4N\ln N$	761	104716	b，c，j
基数表排序	程序5.2.5R	是	36	$N+\epsilon(N+200)$	$32N+4838$	$32N$	4250	36838	b，c

a：仅三个数字的键

b：仅六个数字（即三字节）键

c：输出不重新排列，通过链接或计数器隐式地指明最后的序列

d：如同在5.2.1-8中那样选择增量，5.2.1-29中出现稍微好些的序列

e：$M=10$ 利用 SRB，对于用 DIV 的样式，平均运行时间增加 $\frac{13}{7}N$

f：$M=100$（字节大小）

g：$M=34$　因为 $2^{34}>10^{10}>2^{33}$

h：平均时间以经验估计为基础，因为这一理论不完备

i：平均时间以一致分布的键的假定为基础

j：正文中和伴随这一程序的习题提出的进一步改进，将减少运行时间

霍勒里斯（Herman Hollerith），人口统计局的一名20岁的职员，发明了一台巧妙的电动制表机，以满足更好地进行统计收集的需要，他的大约100台机器成功地用于制造1890年的人口名册。

图94示出了霍勒里斯最初的用电池驱动的装置；我们的主要兴趣在于右边的"排序盒"，它已被打开以示出26个小室的一半。操作员插入一张$6\frac{5}{8}$英寸×$3\frac{1}{4}$英寸的穿孔卡片到"夹具上"上，然后放下手柄；这使得凡是在卡片中穿有孔的地方，上面板上用弹簧驱动的针同在下面板上的水银容器相接触。相应的被接通了的线路将引起在控制台面板上的诸相关转盘前进一个单位；然后，26个排序盒盖之一推开。这时，操作员重新打开夹具，把卡片放到打开的小室，并关上盖。据说，一个人在6个半小时的一个工作日中通过这个机器能运行19071张卡片，大约每分钟49张卡片（一个典型的操作员的工作速度大约仅为这个速度的$\frac{1}{3}$左右）！

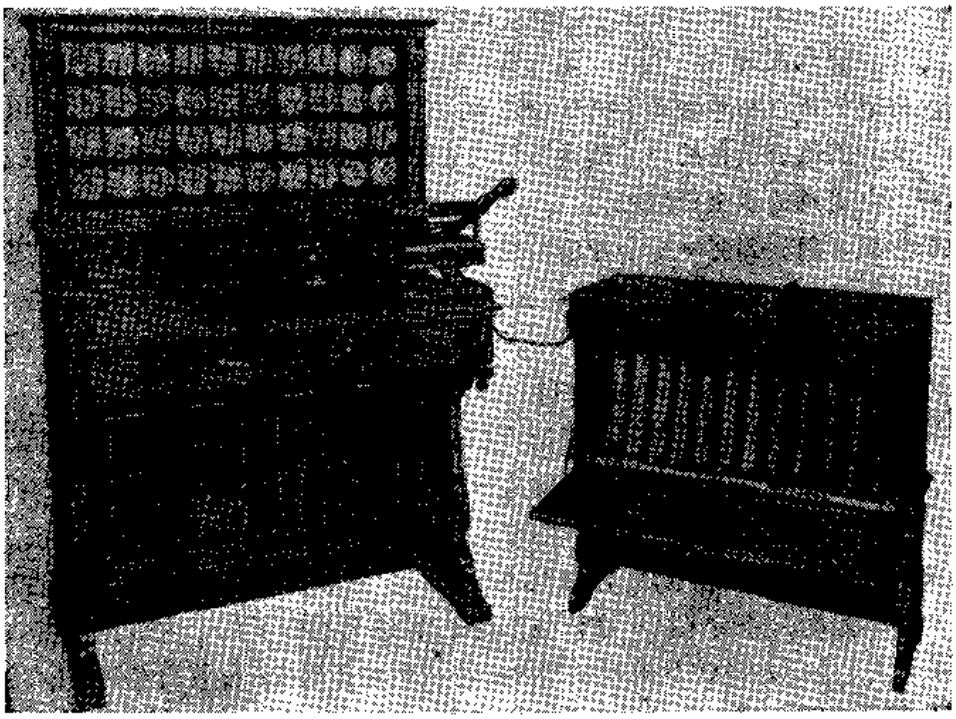


图94　霍勒里斯最初的制表机和排序机

（蒙IBM档案室同意采用此照片）

人口继续无情地增长，而原来的制表排序机已经不足以处理1900年的人口了。所以霍勒里斯又发明了另一台机器来避开另一次数据处理的危机。他的新设备（1901年和1904年的专利）有一个自动的喂卡（片）器，事实上它很象现代的卡片排序机。关于霍勒里斯早期机器的故事，已由利昂·E. 特鲁斯德尔（Leon E.Truesdell）有趣而详尽地加以叙述，见"The Development of Punch Card Tabulation"（Washington: U.S. Bureau of the Census, 1965）一书；也见当时的报导"Columbia College School of Mines Quarterly" 10(1889)238-255；"J.Franklin Inst." 129(1890)300-306；"The

Electrical Engineer"12 (Nov.11, 1891)521-530; "J.Amer.statistical Assn."2(1891), 330-341, 4 (1895), 365; "J.Royal Statistical Soc." 55(1892)326-327; "Allgemeines Statistisches Archiv" 2 (1892), 78-126; "J.Soc.Statistique de Paris"33(1892), 87-96; "U.S.Patents" 395781(1889), 685608(1901), 777209(1904)。霍勒里斯和人口统计局以前的另一名职员詹姆斯·鲍尔斯（James Powers）接着又创办互相竞争的公司，它们最终分别成为IBM和雷明顿·兰德（Remington Rand）公司的一部分。

霍勒里斯的排序机当然是现在用于数字计算机中基数排序方法的基础。他的专利提到，两列数值项的排序是每列独立地进行的，但他没有说是否应当首先考虑个位列或者十位列。首先使用个位列的这个不明显的技巧，大概首先是由某些不知名的机器操作员发现的，并且传给了其他人（参考 5.2.5 节）；它出现在最早期流行的IBM排序机手册中（1936），这个自右至左技术的头一个已知的出处是在罗伯特·法因德勒（Robert Feindler）所写的一本书，"Das Hollerith-Lochkarten-Verfahren"(Berlin: Reimar Hobbing, 1929), 126-130 中；大约在同一时间，它也为L.J.科姆里（L.J.Comrie）在一篇论文中提到，见"Trans. of the Office Machinery Users' Assoc"(London, 1930), 25-37。偶然地，科姆里成了作出重要发现的头一个人，这一发现就是，制表机尽管最初是为统计和结算应用而设计的，但它们可以有效地用于科学计算中。他的文章特别有趣，因为它给出了 1930 年在英国可买到的制表设备的详细描述。那时的排序机每分钟处理 300 到 400 张卡片，而且可以以每月九英磅的租金租用之。

合并的思想可回溯到另一部卡片滚动机器：整理机，这是一项晚得多的发明(1938)。通过它的两个馈送站，可以在仅仅一次扫描中就把两叠排好序的卡片合并成一叠；进行这一工作的技术，在头一本IBM整理机手册（April, 1939）中就清楚地说明了。〔参考 James W.Bryce, U.S.Patent 2189024(1940)〕。

然后计算机登上了舞台，而且终于把排序也包括在其发展当中。事实上有证据表明，排序例程曾经是为存储程序计算机写出的头一个程序。EDVAC的设计者们对于排序特别感兴趣，因为它集中体现了计算机潜在的非数值应用；他们认识到，一组令人满意的指令代码，不仅应有能力表达用于解差分方程的程序，它还必须有足够的灵活性来处理算法中的组合"判定"问题。约翰·冯·诺依曼因此于 1945 年编制了用于内部合并排序的程序，为的是检验他建议的EDVAC计算机的某些指令代码的适用性；有效的专用排序机的存在，提供了一个自然的标准，通过这种标准就可以评价他所提议的计算机组织的优点。本书作者在"Computing Surveys" 2 (1970)247-260 一文中已经描述了这一有趣的发展细节；关于冯诺依曼当初的排序程序的最后"润饰"了的形式，也见"Von Neumann's Collected Works 5"(New York: Macmillan, 1963)196-214。

在德国，K.朱斯（K.Zuse）独立地于 1945 年构造了用于直接插入排序的一个程序，作为在他的"Plankalkül"（计划计算）语言中线性列表操作的最简例子之一（这一开创性的工作推迟了近 30 年才发表；见"Berichte der Gesellschaft für Math. und Datenv"63(1972), Part 4, 84-85。

由于为早期计算机设计的内存容量是很有限的，使得很自然地既要考虑内部排序也要考虑外部排序。由穆尔电气工程学校（Moore School Electrical Engineering）的J.P.

埃克特（J. P. Eckert）和 J. W. 莫茨利所编写的报告“Progress Report on the EDVAC”(September30, 1945）指出，装有磁线或磁带设备的一台计算机可以模拟卡片机的操作，达到更快的排序速度。这个进展报告叙述了平衡的两路基数排序，以及平衡的两路合并（称为“整理”），同时使用四个磁线或磁带机，“每秒至少读或写 5000 个脉冲”。

1946 年在穆尔学校举行的计算的专题讨论会上，约翰·莫茨利作了“排序和整理”的演说，他的演讲稿是头一份公开发表的关于计算机排序的讨论〔Theory and techniques for the design of electronic digital computers, ed. by G. W. Patterson, 3 (1946), 22.1-22.20〕。莫茨利以一个有趣的评论来开始他的介绍：“要求一台机器把计算和排序的能力结合在一起，似乎很象要求一个设备既能用作开罐启子，又能用作自来水笔。”然后他说，有能力进行复杂的数学运算的机器必须也有能力对数据排序和分类，而且他说明排序可能甚至在数值计算中也有用。他描述了直接插入和二叉插入，说明前一种方法平均使用大约$N^2/4$个比较，而后一种方法决不需要多于$N\log_2 N$个比较。然而，二叉插入需要一个颇为复杂的数据结构，他接着说明两路合并仅仅使用对表的顺序存取就达到了同样低的比较次数。他的讲演稿的后半部分专门讨论了部分扫描的基数排序方法，即模拟四条带上的数字卡片排序，同时每位数字使用少于四次的扫描（参考 5.4.7 节）。

不久以后，埃克特和莫茨利创立了一个公司，这个公司生产了某些最早的电子计算机，BINAC(供军事应用）以及UNIVAC(供商业应用)。美国人口统计局再次在这个发展中发挥了作用，接受了头一台UNIVAC。这时，还全然不清楚计算机在经济上是有利的；用计算机排序可以比卡片机快，但它们要昂贵得多。因此，由弗朗西斯·E. 霍尔伯顿领导的UNIVAC程序员们，作出相当大的努力设计高速的外部排序例程，并且他们早期的程序也影响了硬件的设计。按照他们的估计，在UNIVAC上一亿个 10 个字的记录可以在 9000 个小时（即 375 天）之内完成排序。

UNIVAC I　正式交货于 1951 年 7 月，它的内存为有 1000 个字长为 12 字符（72个二进制位）的字。它被设计成以每秒钟 500 个字的速度，在带上读写 60 个字长的块区；可以向前或向后读，而且可以同时进行读/写/计算。1948 年，霍尔伯顿夫人想出了一个有趣的方法，使用六个输入缓冲区，以读、写和计算的完全重叠来进行两路合并：设每个输入文件都有一个“当前的缓冲区”和两个“辅助的缓冲区，”有可能以这样一种方式来进行合并，即每当输出一个块区时，两个当前的输入缓冲区包含着总共恰恰相当一个块区的未处理的记录。因此，在形成每个输出块区时，恰有一个输入缓冲区变空。故我们可以安排成，在所有时刻，当要读入一个辅助缓冲区时，四个辅助缓冲区中的另外三个都是满的。这个方法比算法 5.4.6F 的预报方法要稍微快些，因为它不需要在开始下一个输入之前，检查已输入的结果〔参考“Collation Methods for the UNIVAC System”(Eckert-Mauchly Computer Corp. 1950), 2 vols〕。

这个工作的最高峰就是一个排序生成程序，它是自动程序设计领域中研制的头一个重要的“软件”例程。用户说明记录的大小、在每个记录的一部分场中至多五个键的位置以及标志文件结束的“哨兵（标志）”键，而排序生成程序将产生对于一卷文件的有版权的排序程序。这个程序的头一遍扫描是使用比较计数（算法 5.2C）的 60 字块区的一个内部排序；然后进行一些平衡的两路合并扫描，向后读并避免如上所述的带互锁〔参考“Ma-

ster Generating Routine for 2-way Sorting"(Eckert-Mauchly Div. of Remington Rand, 1952)；这个报告的第一个草稿的标题是"两路整理的主预制例程"(Master Prefabrication Routine for 2-way Collation)！也见F. E. Holberton"Symposium on Automatic Programming"(office of Naval Research, 1954), 34-39]。

1952年左右，内部排序的许多方法在程序设计界已广为流传，但未充分展开理论上的研究。丹尼尔·戈登堡(Daniel Goldenberg)["Time analyses of various methods of sorting data", Digital Computer Laboratory memo M-1680 (Mass. Inst. of Tech., October17, 1952)]对沃温德(Whirlwind)计算机编写了五个不同方法的程序，并对每个程序都就最好的情况和最坏的情况进行了分析。当对于8位键的100个15位字进行排序时，他发现最快的方法是使用一份256字的表，把每个记录存入对应于它的键的唯一位置，然后压缩这份表。但这项技术有一个明显的缺点，因为每当后继的记录有相同的键时，它将冲掉一个记录。他所分析的其它四种方法被排成如下次序：直接两路合并优于基数2排序优于直接选择优于气泡排序。

这些结果，由哈罗德·H. 西沃德在他1954年的硕士论文["Information sorting in the application of electronic digital computers to business operations"Digital Computer Lab. report R-232 (Mass. Inst. of Tech. May24, 1954; 60 pp)]中作了推广。西沃德引进了分布计数和替代选择的思想；他证明在一个随机排列中的头一个路段的平均长度是$e-1$，而且他分析了在带上以及在各种类型的海量存储上的内部排序以及外部排序。

一篇甚至更值得注意的论文——事实上是一篇博士论文，是霍德华·B. 德穆思于1956年写的["Electronic Data Sorting"(Stanford University, October, 1956), 92pp]。这篇论文有助于奠定计算复杂性理论的基础。它利用循环的、线性的以及随机存取的存储器，考虑了排序问题的三个抽象模型；并对每个模型提出了最优的或接近于最优的方法(参考习题5.3.4-6)。尽管德穆思的论文并没有立即产生实际的结果，但它却建立了如何把理论同实践相联系的重要思想。

这样一来，排序的历史已经同计算中的许多"第一"紧密地联系在一起：第一批数据处理机器，第一批存储程序，第一个软件，第一批缓冲方法，算法分析和计算复杂性的第一个工作。

迄今所述的有关计算机的文献，实际上都未出现在"公开的著作中"；事实上，计算的大多数早期的历史，都出现在比较难以得到的报告中，因为在那时仅有比较少的人跟计算机打交道。关于排序文献的头一次印行是在1955-1956年，用的是三篇重要的综述性论文的形式。

头一篇综述性的文章是由J. C. 霍斯金(J. C. Hesken)编写的[Porc. Eastern joint Computer Conference 8 (1955), 39-55]。他以敏锐的观察开始："为降低每个输出单位的价格，人们通常都在增加他们的操作规模。但在这些条件下，排序的单位费用，不是降低了，而是增加了。"霍斯金综述了在计算机上进行排序的方法，以及所有可利用的现在已在销售的专用设备。他的54项条款的参考文献目录大多数是以厂家的小册子为基础的。

E. H. 弗兰德的内容丰富的论文 "Sorting on Electric Computer Systems"〔JACM 3 (1956), 134-168〕，是排序发展中的一个主要的里程碑。尽管自 1956 年以来已经发展了许多技术，但这篇论文在许多方面直到今天仍然值得注意。弗兰德对于相当多的内部和外部排序算法给出了细心的描述，而且他对于缓冲技术和磁带机的特征给以特殊的注意。他引进了某些新的方法（例如，树的选择，两头蛇排序以及预报），而且展示了较老的方法的某些数学性质。

大约在同一个时间里出现的第三篇关于排序的综述性文章，是由 D. W. 戴维斯(D. W. Davis)〔Proc. Inst. Elect. Engineers 103B, Supplement 1(1956), 87-93〕提出的。在随后数年中，以下列作者发表了许多值得注意的综述：D. A. 贝尔〔Comp. J. 1(1958), 71-77〕；A. S. 道格拉斯〔Comp. J. 2 (1959) 1-9〕；D. D. 麦克拉肯 (D. D. McCracken)，H. 韦斯 (H. Weiss)，以及李载华〔Programming Business New York: Wiley, 1959), Chapter 15, pp. 298-332〕；I·弗洛里斯〔JACM 8(1961), 48-80〕；K. E. 艾弗森〔A Programming Language(New York: Wiley, 1962), Chapter 6, 176-245〕；C. C. 戈德里布 (C. C. Gotlieb)〔CACM6(1963), 194-201〕；T. N. 希巴德〔CACM6 (1963), 206-213〕；M. A. 戈茨〔Digital Computer User's Handbook, ed. by M. Klerer and G. A. Korn (New York: McGraw-Hill, 1967), Chapter 1.10, pp 1.292-1.320〕。1962 年 11 月ACM发起了关于排序的一次讨论会；在该讨论会上宣读的大多数论文都发表在CACM的刊物上（1963 年 5 月），并且它们乃是当时技术发展水平的很好代表。C. C. 格德里布关于现代排序生成程序的综述，T. N. 希巴德关于极小存储内部排序的综述，以及 G. U. 哈巴德关于磁盘文件排序的早期剖析，都是在这个文集中特别值得注意的文章。

整个这个时期发现了一些新的排序方法：地址计算（1965），合并插入（1959），基数交换（1959），级联合并（1959），谢尔的递减增量排序（1959），多阶段合并（1960），树插入（1960），交替排序（1962），霍尔快速排序（1962），威廉的堆排序（1964），巴彻尔的合并交换（1964）。在本章各有关小节中，在描述方法的同时已经追述了每个算法的历史，60 年代后期，相应的理论得到了深入的发展。

当写这一章时，作者所考察的有关排序的全部文章的一份完整的参考文献目录，出现于Computing Reviews13(1972), 283-289中。

习题

1. 〔05〕 试通过叙述定理 5.4.6A的一个推广，来总结这一章的内容。

2. 〔20〕 在表 1 信息的基础上，在MIX计算机上使用时，对于六个数字的键，什么是最好的表排序方法？

3. 〔37〕 （在极小存储中的稳定排序）我们说一个排序算法需要极小存储，如果除了存储N个记录所需的空间外，它的变量仅使用$O((\log_2 N)^2)$ 位存储空间。这个算法必须在下列意义下是一般的，即必须对所有的N都有效，而不仅仅对于一个具体的N值有效，其中假定当实际调用这个算法来进行排序时，已经有充分数量的随机存取内存可资利用。

我们已经研究过的许多排序方法，都违背了这个极小存储的要求；特别是，使用N个

链接场是禁止的，快速排序（算法 5.2.2Q）满足极小存储要求，但它最坏情况的运行时间竟同N^2成正比。堆排序（算法 5.2.3H）是我们所研究过的唯一使用极小存储$O(N\log_2 N)$的算法，尽管利用习题 5.2.4-18 的思想，还可描述另一个这样的算法。

我们已经研究过的、以一种稳定的方式对键排序的、最快的一般算法是表合并排序（算法 5.2.4L），但它不使用极小存储。事实上，我们已经见到的稳定的极小存储排序算法都是$O(N^2)$的方法（直接插入，气泡排序以及直接选择的一种变种）。

设计一个稳定的极小存储排序算法，它在最坏的情况下仅需$O(N(\log_2 N)^2)$个时间单位。〔提示：有可能在$O(N\log_2)N$个时间单位内实行稳定的极小存储合并。〕

如果我能把过去几年内所产生的大量关于排序的材料的要旨都加以排序，编排出逻辑次序来，那我就心满意足了。

——J.C.霍金斯（1955）

第6章 查 找

让我们来查看这个记录。

——艾尔·史密斯〔(AL Smith) 1928〕

这一章可以给出更大的标题"信息的存储和检索"；另一方面，它也可以简单地称做"查表"，我们关心的是在一台计算机的存储中收集信息的过程，以及随后尽可能快地进行对该信息的检索。有时，我们掌握的数据中有一些实际上是没有用处的，可能最明智的办法是把这些数据的大部分忘掉或销毁；但在其它情况下，重要的是，要以能进行快速检索的方式保留和组织给定的数据。

这一章大部分篇幅都致力于研究一个非常简单的查找问题：怎样根据某种确定的标记来找出存储好的数据。例如，在一个数值应用中，给定了x和一张f值表后要求$f(x)$；在一个非数值应用中，可能要找一个给定的俄文单字的英语翻译。

通常，我们将假设，N个记录的一个集合已经存储好，问题是要找出我们所要的记录。如同排序那样，假定每个记录都包括称为它的键的一个特殊场，取这个名字也许是因为许多人每天都要花很多时间来找他们的钥匙，键就象钥匙●。我们一般都要求N个键互不相同，以便每一个键唯一地标识它的记录。所有记录的集合称为一份表或一个文件，"表"一词在这里通常用来表示一个小文件，而"文件"通常用来表示一个大的表。一个大的文件或一组文件经常称作一个数据库。

这里给出的查找算法中有一个变元K，问题是要寻找以K为其键的那一个记录。在查找完成之后，有两种可能性：或者这个查找是成功的，已找到含有K的唯一记录，或者是不成功的，已确定K无处可找。在不成功的查找之后，有时希望送入包含K的一个新记录到这份表中；做这项工作的方法称为"查找和插入"算法。名叫"联想存储"的某些硬件设备，以模拟人脑功能的方式自动地解决查找的问题；而我们将研究在一部通常的通用计算机上来进行查找的一些技术。

尽管查找的目标是寻找存储在同K相关联的记录中的信息，但这一章中的算法一般都忽略除键本身以外的一切。实际上，一旦我们定出K的地址，便可找出相关联的数据；例如，如果K出现在TABLE+i中，则相关联的数据（或指向该数据的一个指针）可在单元TABLE+i+1中，或在单元DATA+i中等等。因此，找到K之后，不妨把应该做什么等细节都忽略掉。

查找是许多程序中最消耗时间的一部分，因而用一个好的查找方法来替代一个坏的方法，常常会使运行速度大大提高。事实上，只要数据或数据结构安排得好，则查找常可完全免去。即，我们总能确切地知道所需要找的信息究竟在何处。拉链存储是实现这一点的一个普通的方法；例如，使用双重链接表就不必查找一个给定项目的前驱和后继。如果允

● 在英语中键和钥匙是同一个词Key。——译注

许我们自由地选择键，则又有避免查找的另一方法，因为我们也可以令诸键为数{1，2，…，N}；然后包含K的记录可以简单地放置在单元TABLE+K中。在2.2.3节讨论的拓扑排序算法中，为消除查找，这两项技术都曾用过。然而还有许多情况需要进行查找（例如，如果拓扑排序算法中的对象是符号名而不是号码），所以重要的是要有有效的查找方法。

查找方法可用若干种方式分类。我们可以把它们分为内部查找和外部查找，就同把第五章的排序算法分为内部排序和外部排序一样。或者可以把查找方法分为静态查找和动态查找，其中“静态”指的是表的内容实际上不变（于是重要的是极小查找时间而不考虑建立表所需要的时间），而“动态”指的是这个表经常要受到插入的影响（也许还要进行删除）。第三种可能的方案是，按照查找方法是以键之间的比较为基础还是以键的数字性质为基础来分类，就象用比较进行排序和用分布进行排序之间的区别那样。最后，我们还可以根据使用实际的键还是使用变换了的键来对查找方法分类。

这一章的组织，实质上是后两种分类方式的综合。6.1节考虑查找的“硬找”方法，即顺序方法，然后6.2节讨论了一种改进的方法，它以键之间的比较为基础，根据字母或数字的顺序作出判断。6.3节讨论数字查找，6.4节讨论一类重要的方法，称作杂凑技术，它是以实际键的算术变换为基础的。每一节都在静态和动态两种情况下，既讨论内部查找也讨论外部查找；并且每节都指出各种算法的相对优点和缺点。

查找和排序两者是不能截然分开的。例如，考虑下列问题：

给定两组数，$A=\{a_1, a_2, \cdots, a_m\}$和

$B=\{b_1, b_2, \cdots, b_n\}$，确定是否$A\subseteq B$

这本身提示了三种解，即

1．把诸a_i顺序地和诸b_j相比，直到找到一个相同的为止。

2．把诸b_j记入一个表中，然后查找每个a_i。

3．对诸a和诸b排序，然后对两个文件作一次顺序扫描，校验适当的条件。

上述每一种解法，在m和n值的一个特定范围内都是有吸引力的。解1要花大约c_1mn时间单位，c_1是某个常数；而解3将花大约$c_2(m\log_2 m+n\log_2 n)$时间单位，c_2是某个（更大的）常数。用一个适当的杂凑方法，解2将花大约c_3m+c_4n个时间单位，c_3和c_4是是某个（仍然很大的）常数。由此得出，对于非常小的m和n，解1是好的，但当m和n增大时，解3很快将变成更好。最后，解2变为可取的，直到n超过了内存大小为止，然后，解3往往又变得优越，至到n变得非常大为止。于是我们有这样一种情况：排序有时是对查找的一个好替换，而查找有时又是排序的一个好替换。

复杂些的查找问题通常都可以归结成这里所考虑的较简单的情况。例如，假设键可以是稍有拼错的字；而我们可能要在这种错误的情况下仍能找到正确的记录。如果我们作一个文件的两个副本，一个副本中键以通常的字母顺序出现，另一个则按字母从右到左排好（好象单词是倒过来拼那样），则拼错的查找变元大概将同这两个文件之一中的一个项，一致到它长度的一半或多一半。因此6.2节和6.3节的查找方法可被用来寻找大概想要的键。

一个有关的问题已受到了相当大的注意，它联系到飞机票预约系统及与人名有关的其它应用，在这些应用中，经常有可能由于拙劣的手迹或声音传输而拼错名字。目标是要把变元转换成某个代码，这个代码有助于把同一名字的所有变形都转换成同一形式。下述的

"探测"(Soundex) 方法，最初是由玛格丽特·K.奥德尔(Margaret K. Odell) 和罗伯特·C.拉塞尔 (Robert C. Russell) 提出的〔参考 U.S. Patents 1261167 (1918)，1435663 (1922)〕，常常用来对姓氏编码：

1. 保留名字的头一个字母，并且省略在其余位置中出现的所有 a，e，h，i，o，u，w，y。

2. 把下列数字赋给头一个字母之后的剩下的字母

b，f，p，v → 1	l → 4
c，g，j，k，q，s，x，z → 2	m，n → 5
d，t → 3	r → 6

3. 如果具有相同代码的两个或多个字母在原来的名字（在步骤 1 前）中相邻，则省略除头一个以外的所有字母。

4. 通过增加尾部零（如果少于三个数字），或者省略最右边的数字（如果有三个以上的数字），转换成"字母，数字，数字，数字"的形式。

例如，Euler，Gauss，Hilbert，Knuth，Lloyd 以及 Lukasiewicz 这些名字的代码分别是 E460，G200，H416，K530，L300，L222。当然，这个系统会把有些不同的名字或类似的名字变成同一形式；Ellery，Ghosh，Heilbronn，Kant，Ladd 以及 Lissajous 就得到同样的六型代码。而另一方面，某些有关的名字，如 Rogers 和 Rodgers 或 Sinclair 和 St. Clair 或 Tchebysheff 和 Chebyshev，则仍保持互异。但总的说来，探测代码大大地增加了在伪装的代码中找出名字的机会。〔更多的介绍，参看 C. P. Bourne and D. F. Ford, *JACM* 8 (1961)，538-552；Leon Davidson，*CACM* 5 (1962)，169-171；*Federal Population Censuses* 1790-1890 (Washington，D. C.：National Archives，1971)，90.〕。

当使用象探测这样的方案时，不必放弃所有键都不同这样的假定；我们可以造出具有等价代码的所有记录的表组，把每一个表当成一个单位来处理。

大数据库势必使检索过程更为复杂，因为人们通常要把每个记录的许多不同的场都当作可能的键，当只知道部分键信息时，也有能力来定出项目的位置。例如，给定关于戏剧演员的一个大文件，舞台监督可能希望找出 25 岁到 35 岁之间，具有跳舞天才和法国口音的所有还没有角色的女演员；给定垒球统计的一个大文件，一个体育专栏作家，可能希望确定 1964 年在对左手的投掷者进行晚场比赛的第七个回合期间，由芝加哥的怀特·索克 (White Sox) 跑完的总圈数。给定关于任何事情的一个大型数据文件后，人们会任意地提出复杂的问题。确实，我们可以把整个图书馆看成一个数据库，而查找者可能要寻找已发表的有关情报检索的每份资料。在下面的 6.5 节中，将介绍用于此类多重属性检索问题的技术。

在对查找进行详细研究之前，回顾一下历史似乎是有益的。在计算机出现以前，就编辑出版了对数表、三角函数表等许多书，使得数学计算可为查表所代替，最后，这些表被誊写到穿孔卡片上，并用于同整理机、排序机以及复写穿孔机相联系的科学问题中。但一经出现了存储程序的计算机，人们马上就发现：每次重新计算 $\log_2 x$ 或 $\cos x$ 以代替在一张表中查出答案要更合算些。

尽管在计算机出现伊始，排序问题就受到了相当大的注意，但是对于查找的算法，工

作却做得比较少。由于内存小和只有磁带这样的顺序介质能用来存储大型文件，因此查找或是极为容易，或是几乎不可能。

但是自 50 年代起，越来越大的随机存取存储器的发展，终于导致了这样一个认识，即查找就其本身的性质说来就是一个有趣的问题。在抱怨了好几年早期计算机的空间局限性之后，程序员们突然面对着非常大的内存容量，以致竟不知道如何有效使用它。

关于查找问题最初的一些综述文章如下：A. I. Dumey, *Computers & Automation* 5, 12 (December 1956), 6-9; W. W. Peterson, *IBM J. Research & Development* 1 (1957), 130-146; A. D. Booth, *Information and Control* 1 (1958), 159-164; A. S. Douglas, *Comp. J.* 2 (1959), 1-9。后来，下面两篇文章对查找问题进行了更广泛的讨论：Kenneth E. Iverson, *A Programming Language* (New York: Wiley, 1962) 133-158, 和 Werner Buchholz, *IBM Systems J.* 2 (1963), 86-11。

如我们将要看到的，在 60 年代初期，引进了许多有趣的新的基于树结构的查找方法；而关于查找的研究现在仍然活跃地开展着。

6.1 顺序查找

"从起点开始往下查，直到你找到了正确的键为止；然后停止"。这个顺序过程是进行查找的明显方法，这个方法也构成了我们讨论查找问题的一个有用的起点，因为许多更错综复杂的算法都是以它为基础的。我们将看到，顺序查找除了简单之外，还包含了某些非常有趣的思想。

这个算法可以更精确地阐述如下：

算法 S（顺序查找） 给定其键分别为 K_1, K_2, …, K_N 的记录 R_1, R_2, …, R_N 的一个表，这个算法查找一个给定的变元 K，我们假定 $N \geqslant 1$。

S1.〔初始化〕 置 $i \leftarrow 1$。

S2.〔比较〕 如果 $K = K_i$，则此算法成功地结束。

S3.〔前进〕 i 增加 1。

S4.〔文件结尾？〕 如果 $i \leqslant N$，则返回 S2，否则此算法以失败告终。

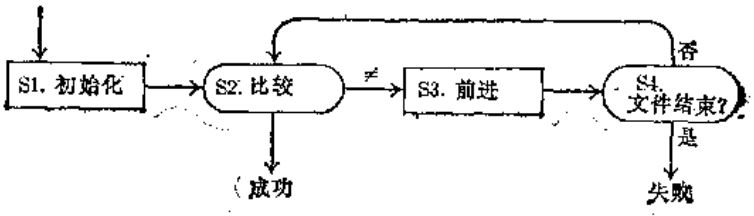


图 1 顺序查找

注意，这个算法可以以两种不同的方式结束，即成功（已定出所需键的位置），或不成功（已证实给定的变元不在该表中）。这一章中大多数其余的算法都同样如此。

一个 MIX 程序可立即写出来：

程序 S（顺序查找） 假定 K_i 出现于单元 KEY + i 中，记录 R_i 的剩余部分出现在单元 INFO + i 中，下列程序使用 rA = K，rI1 = $i - N$。

```
01   START   LDA    K          1    S1. 初始化
02           ENT1   1 - N      1    i ← 1
03   2H      CMPA   KEY + N,1  C    S2. 比较
```

04		JE	SUCCESS	C	如果$K=K_i$，转出
05		INC1	1	$C-S$	$S3$. 前进
06		J1NP	2B	$C-S$	$S4$. 文件结尾?
07	FAILURE	EQU	*	$1-S$	如果不在表中，则转出

现在，在单元SUCCESS处指令“LDA INFO+N，1”将把所需信息带到rA中。

对这个程序的分析是直截了当的；它表明算法S的运行依赖于两个因素

C＝键比较的次数；

S＝1，如果成功；0，如果不成功。

程序S花费$5C-2S+3$个时间单位。如果这个查找成功地找到$K=K_i$，则我们有$C=i$，$S=1$；因此总的时间为$(5i+1)u$。另一方面，如果查找不成功，则我们有$C=N$，$S=0$，总的时间为$(5N+3)u$。如果每个输入键都以相同的概率出现，则在一次成功的查找中，C的平均值将是

$$\frac{1+2+\cdots+N}{N}=\frac{N+1}{2} \tag{2}$$

当然标准离差肯定是相当大的，约为$0.289N$（见习题1）。

上述算法肯定是所有程序员都熟习的，但是，很少人知道，它并不总是进行顺序查找的一个正确的方式！只要作一些修改就会使这个方法更快，除非记录表很短。

算法Q（快速顺序查找） 这个算法和算法S相同，唯一的区别是它假定在这个文件的结尾存在一个“虚拟”记录R_{N+1}。

Q1.〔初始化〕 置$i\leftarrow 1$，并置$K_{N+1}\leftarrow K$。

Q2.〔比较〕 如果$K=K_i$，则转到Q4。

Q3.〔前进〕 i增加1，并返回Q2。

Q4.〔文件结尾?〕 如果$i\leqslant N$，则算法成功地结束；否则以失败告终（$i=N+1$）。

程序Q（快速顺序查找） rA≡K，rI1≡$i-N$。

01	BEGIN	LDA	K	1	$Q1$. 初始化
02		STA	KEY+N+1	1	$K_{N+1}\leftarrow K$
03		ENT1	-N	1	$i\leftarrow 0$
04		INC1	1	$C+1-S$	$Q3$. 前进
05		COMPA	KEY+N,1	$C+1-S$	$Q2$. 比较
06		JNE	*-2	$C+1-S$	如果$K_i\neq K$转到Q3
07		J1NP	SUCCESS	1	$Q4$. 文件结尾?
08	FAILURE	EQU	*	$1-S$	若不在表中则转出

用量C和S来分析程序S，运行时间已减少到$(4C-4S+10)u$，在一个成功的检索中$C\geqslant 6$时，这是一个改进；在一个不成功的检索中，当$N\geqslant 8$时它也是一个改进。

从算法S到算法Q的过渡利用了一个重要的“加速”原理：当程序中的一个内循环要测试两个或多个条件时，则应该力图将其减少为仅仅一个条件。

另一项技术将使程序Q还要快。

程序Q′（更快的顺序查找） rA≡K，rI1≡$i-N$。

01	BEGIN	LDA	K	1	Q1. 初始化
02		STA	KEY+N+1	1	$K_{N+1} \leftarrow K$
03		ENT1	-1-N	1	$i \leftarrow -1$
04		INC1	2	$\lfloor (C-S+2)/2 \rfloor$	Q3. 前进(两步)
05		CMPA	KEY+N,1	$\lfloor (C-S+2)/2 \rfloor$	Q2. 比较
06		JE	4F	$\lfloor (C-S+2)/2 \rfloor$	如果$K=K_i$则转到Q4
07		CMPA	KEY+N+1,1	$\lfloor (C-S+1)/2 \rfloor$	Q2. 比较(第二个)
08		JNE	3B	$\lfloor (C-S+1)/2 \rfloor$	如果$K \neq K_{i+1}$则转到Q3
09		INC1	1	$(C-S) \bmod 2$	i 前进
10	4H	JINP	SUCCESS	1	Q4. 文件结尾?
11	FAILURE	EQU	*	$1-S$	如不在表中则转出

内循环分成了两个；这避免了大约一半的“$i \leftarrow i+1$”指令，所以它使运行时间减少成

$$3.5C-3.5S+10+\frac{(C-S) \bmod 2}{2}$$

个单位。当正被查找的是很大的表时，我们已经节省了程序S运行时间的百分之三十；许多现存的程序都能以这种方式改进。

如果我们知道键是递增次序的；则宜于稍微改变一下算法：

算法T（在有序表中顺序查找） 给定一个其键为递增次序 $K_1<K_2<\cdots<K_N$ 的记录 R_1，R_2，…，R_N 的表，这个算法查找一个给定的变元K。为了方便，也为了速度，本算法假定有一个其键为 $K_{N+1}=\infty>K$ 的虚拟记录 R_{N+1}。

T1.〔初始化〕 置 $i \leftarrow 1$。

T2.〔比较〕 如果$K \leqslant K_i$，则转到T4。

T3.〔前进〕 i 加1，并返回T2。

T4.〔相等？〕 如果$K=K_i$，则此算法成功地终止，否则，它以失败告终。

如果所有的输入键都是同等可能的，则当查找成功时，这个算法所花的时间实质上和算法Q的平均运行时间相同。但当查找不成功时它比算法Q快大约两倍，因为该算法可以更快地确定一个记录不存在。

上述每一个算法都使用下标表示表的项，借助于这些下标来描述这些方法是很方便的，但是同样的查找过程也可用于一个使用链式表示的表中，这是因为数据是被顺序地遍历的（见习题2、3和4）。

存取频率 至今我们一直假定每个变元都同样经常出现，这并不总是一个现实的假定；在一般情况下，键 K_i 将以概率 p_i 出现，其中，$p_1+p_2+\cdots+p_N=1$。为进行一个成功的查找所需要的时间实际上同比较的次数C成正比，C的平均值为

$$\overline{C}_N=p_1+2p_2+\cdots+Np_N \tag{3}$$

若我们能以任意次序把记录放置到表中，则当

$$p_1 \geqslant p_2 \geqslant \cdots \geqslant p_N \tag{4}$$

时，即当最经常使用的记录出现在接近开始处时量 $\overline{C}_N$ 最小。

现在让我们考虑若干概率分布，以了解当诸记录以（4）中所确定的“最优”方式排

列时，可能节省多少时间？若 $p_1=p_2=\cdots=p_N=1/N$，则公式（3）简化为 $\overline{C}_N=(N+1)/2$；我们已经在等式（2）中导出这个值。另一方面，假设

$$p_1=\frac{1}{2},\quad p_2=\frac{1}{4},\quad \cdots,\quad p_{N-1}=\frac{1}{2^{N-1}},\qquad p_N=\frac{1}{2^{N-1}} \tag{5}$$

则由习题7，$\overline{C}_N=2-2^{1-N}$；若诸记录在表中以正确的次序出现，则对于这个分布，比较的平均数小于2。

另一个可供考察的概率分布是

$$p_1=Nc,\quad p_2=(N-1)c,\quad \cdots,\quad p_N=c$$

其中

$$c=2/N(N+1) \tag{6}$$

这个“楔形的”分布不象（5）那样厉害地偏离一致分布。在这种情况下，我们发现

$$\overline{C}_N=c\sum_{1\leqslant K\leqslant N}k\cdot(N+1-k)=\frac{N+2}{3} \tag{7}$$

即此种最优排列比起诸记录以随机顺序出现时节省了大约三分之一的查找时间。

当然，（5）和（6）中的概率分布是相当人为的，而且它们可能决不是现实的非常好的近似。一个更为典型的分布是“吉甫（Zipf）定律”

$$p_1=c/1,\quad p_2=c/2,\quad \cdots,\quad p_N=c/N,\ \text{其中}\ c=1/H_N \tag{8}$$

这个分布是由G. K. 吉甫(G. K. Zipf)得到的，他发现在用自然语言写的文章中第 n 个最常用的单字似乎近似地以与 $1/n$ 成正比的概率出现〔*The Psychology of Language*(Boston, Mass:Houghton Mifflin, 1935); *Human Behavior and the Principle of Least Effort* (Reading, Mass:Addison-Wesley, 1949)〕。他发现当大城市按递减的人口数排列时，人口统计表中也有同样的现象。如果吉甫定律适用于一个表中键的频率，则我们立即有

$$\overline{C}_N=N/H_N \tag{9}$$

查找这样一个文件大约是查找按随机次序排列记录的同一文件速度的 $\frac{1}{2}\ln N$ 倍。

〔参考 A. D. Booth et al., *Mechanical Resolution of Linguistic Problems* (New York: Academic Press, 1958), 79.〕。

对于现实分布的另一个近似，是在商业应用中已被普遍地观察到的“80-20”估计规则〔参考 W. P. Heising *IBM System J.* 2 (1963), 114-115〕。这个规则指出，80%的事务涉及一个文件最常用的20%；而对这20%应用同样的规则，则使得64%的事务涉及最常用的4%，等等，换言之

$$\frac{p_1+p_2+\cdots+p_{.20n}}{p_1+p_2+p_3+\cdots+p_n}\approx .80\quad \text{对所有}\ n \tag{10}$$

当 n 是5的倍数时，恰好满足这个规则的一个分布是

$$p_1=c,\quad p_2=(2^\theta-1)c,\quad p_3=(3^\theta-2^\theta)c,\quad \cdots,\quad p_N=(N^\theta-(N-1)^\theta)c \tag{11}$$

其中

$$c=1/N^\theta,\qquad \theta=\frac{\log_2 .80}{\log_2 .20}=0.1386 \tag{12}$$

因为在这种情况下，对于所的 n，$p_1+p_2+\cdots+p_n=cn^\theta$。要分析（11）中的概率不太容易；然而，由于我们有 $n^\theta-(n-1)^\theta=\theta n^{\theta-1}(1+O(1/n))$，所以有一个更简单的近似满足 80-20 规则的分布，即

$$p_1=c/1^{1-\theta},\quad p_2=c/2^{1-\theta},\quad \cdots,\quad p_N=c/N^{1-\theta},\quad 其中\ c=1/H_N^{(1-\theta)} \tag{13}$$

如前，这里 $\theta=\log_2 \cdot 80/\log_2 \cdot 20$，且 $H_N^{(r)}$ 是阶为 S 的第 N 个调和数，即 $1^{-r}+2^{-r}+\cdots+N^{-r}$。注意，这个概率分布非常类似于吉甫定律（8）；当 θ 从 1 变成 0 时，这个概率分布从一个一致分布变成一个吉甫分布$\left(确实，吉甫发现在个人收入的分布中\ \theta\approx\frac{1}{2}\right)$。应用（3）到（13）得出

$$\overline{C}_N=H_N^{(-\theta)}/H_N^{(1-\theta)}=\frac{\theta N}{\theta+1}+O(N^{1-\theta})\approx 0.122\,N \tag{14}$$

是关于 80-20 定律的平均比较数（见习题 8）。

由 E. S. 施瓦茨（E. S. Schwartz）进行的单字频率的研究〔见 *JACM* 10（1963）422 页上有趣的图〕，提示了对于吉甫规则的一个改进

$$p_1=c/1^{1+\theta},\quad p_2=c/2^{1+\theta},\quad \cdots,\quad p_N=c/N^{1+\theta},\quad 其中\ c=1/H_N^{(1+\theta)} \tag{15}$$

θ 是一个小的正值〔参照（13）；θ 的符号已经被反过来了〕，在这种情况下

$$\overline{C}_N=H_N^{(\theta)}/H_N^{(1+\theta)}=N^{1-\theta}/(1-\theta)\zeta(1+\theta)+O(N^{1-2\theta}) \tag{16}$$

当 $N\to\infty$ 时它大大地小于（9）。

一个“自组织”的文件 上述对于概率的计算看起来挺不错，但是在大多数情况下并不知道概率是多少，我们可以在每个记录中建立一个计数器来记住它被存取的次数，在这些计数的基础上重新分配记录；上边导出的公式告诉我们，这个过程通常能节省不少运行时间。但是或许我们不想提供那么多的存储空间来作计数场，因为我们可以更好地利用这些空间（例如，利用在这一章稍后将要说明的非顺序查找技术）。

有一种简单的方案，可用来把各记录按一种相当好的次序存放而无须使用附加的计数场。尽管并不知道这个方案的来源，但它已经使用多年了：每当一个记录被成功地检索（定址）时，便把它移到表的开头。若这个表是一个链接的线性表，则此过程是容易实现的，特别是因为正被移到开头处的这个记录通常都有待于进行某种重大更新。

这个“自组织”技术的思想背景是，当需要查找常用的项目时，应在相当接近于表的开头处找到它们。若假定 N 个键分别以概率 $\{p_1,\ p_2,\ \cdots,\ p_N\}$ 出现，而每次查找又完全独立于先前的查找，则可以证明，在这样一个自组织的文件中为寻找一个项目所需要的平均比较数，趋于极限值

$$\overline{C}_N=1+2\sum_{1\leqslant i<j\leqslant N}\frac{p_ip_j}{p_i+p_j}=\frac{1}{2}+\sum_{i,j}\frac{p_ip_j}{p_i+p_j} \tag{17}$$

（见习题 11）。例如，若当 $1\leqslant i\leqslant N$ 时有 $p_i=1/N$，则自组织表总是以完全随机的次序出现，且这个公式简化为上面所熟悉的公式 $(N+1)/2$。

现在让我们看看当键的概率服从吉甫定律（8）时，自组织过程的效果如何。由等式 1.2.7-8、3，有

$$\overline{C}_N=\frac{1}{2}+\sum_{1\leqslant i,j\leqslant N}\frac{(c/i)(c/j)}{c/i+c/j}=\frac{1}{2}+c\sum_{1\leqslant i,j\leqslant N}\frac{1}{i+j}$$

$$=\frac{1}{2}+c\sum_{1\leqslant i\leqslant N}(H_{N+i}-H_i)=\frac{1}{2}+c\sum_{1\leqslant i\leqslant 2N}-2c\sum_{1\leqslant i\leqslant N}H_i$$

$$=\frac{1}{2}+c((2N+1)H_{2N}-2N-2(N+1)H_N+2N)$$

$$=\frac{1}{2}+c(N\ln 4-\ln N+O(1))\approx 2N/\log_2 N \tag{18}$$

当N相当大时，这比$\frac{1}{2}N$要好得多，而且它仅仅为最优排列中所得到比较次数的大约$\ln 4\approx 1.386$倍（参考9）。

有关实际编译程序符号表的计算经验指出，自组织方法的效果甚至比上述公式所预料的更好，因为逐次的查找并不是独立的（有一小批键趋向于成串出现）。

约翰·麦卡帕（John McCabe）〔*Operations Research* 13（1965），609-618〕首先分析了这个自组织方案，他建立了（17）。有关的结果，见小G.沙伊（G.Schay，Jr.）和F.W.多尔（F.W.Dauer），*SIAM J.Appl.Math.* 15（1967），874-888。

麦卡帕还引进了另一个有趣的方案。其中，每一个成功地找到的、已不在表开头的键，只是简单地与前面的键互换，而不是总被移到头前去。他猜想，假定是独立查找，这一方法的极限平均查找时间决不超过（17）。事实上，后来罗纳德·L.吕弗斯（Ronald L.Rivest）证明，除了当$N\leqslant 2$时或当所有非零概率都相等之外（这是当然的），此换位方法渐近地使用严格地比移至前头方法少的比较（待发表）。

对于不等长记录的磁带查找 现在让我们把问题提升到一个新的难度：假设正在查找的表存储在磁带上，并且个别记录有可变的长度。例如，在一个旧式的操作系统中，“系统库带”就是这样一个文件；标准的系统程序，诸如编译程序、汇编程序、装入程序、报告生成程序等，都是这条带上的“记录”而大多数用户作业一开始就要从头至尾查找此带，以便读入所需的程序。这种体制使我们以前对算法S的分析已不可应用，因为步骤S3在每次执行时所花费的时间是变化的，因此，比较次数并不是唯一重要的准则。

设L_i是记录R_i的长度，p_i是记录被访问的概率，则这个查找方法的平均运行时间近似地同

$$p_1L_1+p_2(L_1+L_2)+\cdots+p_N(L_1+L_2+L_3+\cdots+L_N) \tag{19}$$

成正比。当$L_1=L_2=\cdots=L_N=1$时，此式化简为（3），这种情况已经研究过了。

把最经常需要的记录放在带的开头似乎是合乎逻辑的；但有时这却是个坏主意！例如，假定带仅包含两个程序A和B；对A的需要是对B的需要的两倍，而A的长度却是B的四倍，于是，$N=2$，$p_A=\frac{2}{3}$，$L_A=4$，$p_B=\frac{1}{3}$，$L_B=1$。如果我们首先把A放在带上，按照上面所述的“合乎逻辑的”原理，则平均运行时间是$\frac{2}{3}\times 4+\frac{1}{3}\times 5=\frac{13}{3}$；但如果我们使用一个“不合逻辑”的想法，首先放置$B$，则平均运行时间就被减少到$\frac{1}{3}\times$

$1+\frac{2}{3}\times 5=\frac{11}{3}$。

在一条库带上程序的最优排列可以确定如下：

定理S 设L_i和p_i定义如上，则当且仅当

$$p_1/L_1\geqslant p_2/L_2\geqslant\cdots\geqslant p_N/L_N \tag{20}$$

时表中记录的排列为最优，换言之，当且仅当（20）成立时，对于$\{1,2,\cdots,N\}$的所有排列$a_1a_2\cdots a_N$，$p_{a_1}L_{a_1}+p_{a_2}(L_{a_1}+L_{a_2})+\cdots+p_{a_N}(L_{a_1}+\cdots+L_{a_N})$的极小值等于(19)。

证明 假设R_i和R_{i+1}在带上互换；则代价（19）就从

$$\cdots+p_i(L_1+\cdots+L_{i-1}+L_i)+p_{i+1}(L_1+\cdots+L_{i+1})+\cdots$$

变成

$$\cdots+p_{i+1}(L_1+\cdots+L_{i-1}+L_{i+1})+p_i(L_1+\cdots+L_{i+1})+\cdots$$

纯变化为$p_iL_{i+1}-p_{i+1}L_i$。因此，如果$p_i/L_i<p_{i+1}/L_{i+1}$，则这样一个交换将改进平均运行时间，因而给定的排列就不是最优的。由此得出，（20）在任何最优的排列中成立。

反之，假定（20）成立；我们需要证明排列是最优的。刚才给出的论证表明，这个排列是“局部地最优的”，其意义是交换相邻元素不会导致改进；但可以设想有一个长的复杂的交换序列，它导出一个更好的“全局优化”。我们将考虑两个证明，一个证明使用计算机科学，一个使用一项数学技巧。

第一个证明 假定（20）成立。我们知道，通过相邻记录的一系列交换，记录的任何排列可以被“排”成次序$R_1R_2\cdots R_N$。每次交换都对某个$i<j$以$\cdots R_iR_j\cdots$代替$\cdots R_jR_i\cdots$，所以它使查找时间减少了非负量$p_iL_j-p_jL_i$。因此次序$R_1R_2\cdots R_N$必定使查找时间成为极小。

第二个证明 把每个概率p_i代之以

$$p_i(\epsilon)=p_i+\epsilon^i-(\epsilon^1+\epsilon^2+\cdots+\epsilon^N)/N \tag{21}$$

其中ϵ是极其小的正数。当ϵ充分小时，决不可能有$x_1p_1(\epsilon)+\cdots+x_Np_N(\epsilon)=y_1p_1(\epsilon)+\cdots+y_Np_N(\epsilon)$除非$x_1=y_1\cdots$，$x_N=y_N$；特别是，等式在（20）中将不成立。现在考虑记录的$N!$个排列；至少其中有一个是最优的，且知道它满足（20）；但是由于没有等式，故仅有一个排列满足（20）。因此只要ϵ充分小，（20）就唯一地表征了概率为$p_i(\epsilon)$的表中诸记录的最优排列。由连续性，当ϵ为0时，同一排列必然也是最优的（在与组合最优化有关的问题中，这种“解结”类型的证明通常是有用的）。

定理S是由W. E. 史密斯（W. E. Smith）给出的，见“*Naval Reseach Logistics Quarterly*” 3（1956），59-66。下面的习题包含了最优文件排列进一步的结果。

文件的压缩 如果我们能把数据组装起来使它耗费较少的空间，那么，在磁带上和其它外部存储设备上的顺序查找会进行得更快；因此考虑文件的另一种表示方式是一个好主意。我们并不总需要明显地存储键。

例如，假设我们要一张小于100万的所有质数的表，用来分解12位数字的数（参考4.5.4节）。共有78498个这样的质数；所以如果对每一个质数都使用20个二进位，则这个文件就有1,569,960个二进位长。这显然太浪费了，因为我们可以有一份一百万个二进位的表，用每位来表示对应的数是否是质数。由于所有质数（除2外）都是奇数，这个文件事实上被缩短到500,000个二进位。

缩短这个文件的另一种方式，是列出质数的间隔大小，而不是列出质数本身。按照表1，对于所有小于1,357,201的质数，差 $(p_{k+1}-p_k)/2$ 都小于64，所以我们只需要存储78496个表示间隔距离/2的数字（每个占六位二进数），便能表示3和1000000之间的所有质数；这使文件大约有471,000个二进位长。对这个间隔使用可变长度的二进制码，可以实现进一步的压缩（参考6.2.2节）。

表1　连续质数间的记录间隔

间隔 $(p_{k+1}-p_k)$	p_k	间隔 $(p_{k+1}-p_k)$	p_k
1	2	96	360653
2	3	112	370261
4	7	114	492113
6	23	118	1349533
8	89	132	1357201
14	113	148	2010733
18	523	154	4652353
20	887	180	17051707
22	1129	210	20831323
34	1327	220	47326693
36	9551	222	122164747
44	15683	234	189695659
52	19609	248	191912783
72	31397	250	387096133
86	155921	282	436273009

对于所有的 $j<k$，此表列出了当 $p_{k+1}-p_k$ 超出 $p_{j+1}-p_j$ 时前者的值。更详细的介绍见R. P. Brent, Math. Comp. 27 (1973), 959—963。

习题

1. 〔$M20$〕 当所有的键都有相同的查找概率时，在对一个有 N 个记录的表进行成功的顺序查找中，所作比较次数的标准离差是多少？

2. 〔16〕 利用链接存储记号代替下标记号，重新叙述算法S的步骤（如果P指向表中的一个记录，则假定KEY（P）是键，INFO(P）是相关的信息，并假定LINK(P）是指向下一个记录的指针，FIRST指向头一个记录，最后的记录指向Λ)。

3. 〔16〕 写出习题2中算法的一个MIX程序，如果用（1）中的量 C 和 S 来表达，你的程序运行时间是多少？

▶4. 〔17〕 算法Q的思想是否可不用下标记号而用链接存储记号表示(参考习题2)？

5. 〔20〕 当 C 很大时，程序Q′显然比程序Q快得多。但是否有任何小的 C 和 S 值，使程序Q′实际上比程序Q花的时间多？

6. 〔20〕 对程序Q′添加三条以上的指令，使它的运行时间减少到大约（$3.33C$ + 常数）u。

7. 〔$M20$〕 试利用“二进的”概率分布（5）计算比较的平均数（3）。

8. 〔$HM22$〕 当 $x\neq 1$ 时，试求 $n\to\infty$ 时 $H_n^{(x)}$ 的一个渐近级数。

▶9. 〔$M23$〕 正文指出，由（11）和（13）所给出的概率分布大致是相等的，而且利用（13）的平均比较数是 $\theta N/(\theta+1)+O(N^{1-\theta})$。当使用（11）的概率时平均比较数

是否也等于 $\theta N/(\theta+1)+O(N^{1-\theta})$？

10.〔$M20$〕（4）确定了在一个顺序表中诸记录最好的排列，什么是最坏的排列？指出在最坏的排列中的平均比较数与最好的排列中的平均比较数有一个简单的关系。

11.〔$M30$〕 本题的目的是分析正文中自组织文件的特性。首先我们需要定义一些相当复杂的记号：设 $f_m(x_1, x_2, \cdots, x_m)$ 是所有不同的有序积 $x_{i_1}x_{i_2}\cdots x_{i_k}$ 的无穷和，其中 $1\leqslant i_1, \cdots, i_k\leqslant m$ 且每个 $x_1, x_2, \cdots, x_m$ 都出现于每一项中，例如

$$f_2(x, y)=\sum_{j, k\geqslant 0}(x^{1-j}y(x+y)^k+y^{1-j}x(x+y)^k)$$
$$=\frac{xy}{1-x-y}\left(\frac{1}{1-x}+\frac{1}{1-y}\right)$$

给定 n 个变量 $\{x_1, x_2, \cdots, x_n\}$ 的一集合 X，令

$$P_{nm}=\sum_{1\leqslant j_1<\cdots<j_m\leqslant n} f_m(x_{j_1}, \cdots, x_{j_m});\quad Q_{nm}=\sum_{1\leqslant j_1<\cdots<j_m\leqslant n}\frac{1}{1-x_{j_1}-\cdots-x_{j_m}}$$

例如，$P_{32}=f_2(x_1, x_2)+f_2(x_1, x_3)+f_2(x_2, x_3)$ 和 $Q_{32}=1/(1-x_1-x_2)+1/(1-x_1-x_3)+1/(1-x_2-x_3)$。

我们约定置 $p_{n0}=Q_{n0}=1$。

a）假定正文中的自组织文件已以概率 p_i 响应对项目 R_i 的需求，在这个系统运行了一段长时间之后，试证 R_i 将以极限概率 $p_iP_{N-1,m-1}$ 成为从前端起的第 m 个项目，其中

$$X=\{p_1, \cdots, p_{i-1}, p_{i+1}, \cdots, p_N\}$$

b）对 $m=1, 2, \cdots$，把（a）的结果加起来，我们就得到恒等式

$$P_{nn}+P_{n,n-1}+\cdots+p_{n0}=Q_{nn}$$

证明由此可得

$$P_{nm}+\binom{n-m+1}{1}p_{n,m-1}+\cdots+\binom{n-m+m}{m}p_{n0}=Q_{nm}$$

$$Q_{nm}-\binom{n-m+1}{1}Q_{n,m-1}+\cdots+(-1)^m\binom{n-m+m}{m}Q_{n0}=p_{nm}$$

c）计算 R_i 与表前端的极限平均距离 $d_i=\sum_{m\geqslant 1}mp_iP_{N-1,m-1}$；然后求值

$$\overline{C}_N=\sum_{1\leqslant i\leqslant n}p_id_i$$

12.〔$M23$〕 当查找的键有二进概率分布（5）时，用（17）计算为查找自组织文件所需要的平均比较数。

13.〔$M27$〕 用（17）计算概率分布（6）的 $\overline{C}_N$。

14.〔$M21$〕给定两个实数序列 $\langle x_1, x_2, \cdots, x_n\rangle$ 和 $\langle y_1, y_2, \cdots, y_n\rangle$，问什么样的下标排列 $a_1a_2\cdots a_n$ 将使 $\sum x_iy_{a_i}$ 成为极大？成为极小？

▶15.〔$M22$〕 正文说明，当仅寻找一个程序时，怎样在一条"系统库带"上最优地安排诸程序。但对于一条子程序库带，则用另一组假定更为适合，因为我们希望把一个用户程序中调用的诸子程序一起从带上装入内存。

对于这种情况，我们假定需要子程序 i 的概率是 p_i，而不论是否需要其它子程序，于

是，例如全然不需要任何子程序的概率是 $(1-P_1)(1-P_2)\cdots(1-P_N)$；而查找恰在装入第 i 个子程序之后结束的概率是$P_i(1-P_{i+1})\cdots(1-P_N)$。如果$L_i$是子程序 i 的长度，则平均的查找时间实质上将同

$$L_1P_1(1-P_2)\cdots(1-P_N)+(L_1+L_2)P_2(1-P_3)\cdots(1-P_N)+\cdots$$
$$+(L_1+L_2+\cdots L_N)P_N$$

成正比。在这些假定下，子程序在带上的最优排列是什么？

16.〔M22〕〔H. 里塞尔（H. Riesel）〕 一个程序员要测试：是否给定的 n 个条件同时全部为真（例如，他可能要测试是否 $x>0$ 和 $y<z^2$ 两者都成立，而且首先应测试哪一个条件并不是一目了然的）。假设花费T_i个时间单位测试条件 i，而该条件为真的概率是p_i，且同所有其它条件的结果无关。试问他应在什么顺序下来进行测试？

17.〔M23〕（W. E. 史密斯）假设你要做 n 个作业，第 i 个作业花费 T_i 个时间单位，截止时间为D_i。换言之，第 i 个作业应该在至多过了D_i个时间单位之后完成。试问为处理这些作业，什么样的调度 $a_1a_2\cdots a_n$ 将使极大的延迟

$$\max(T_{a_1}-D_{a_1},\ T_{a_1}+T_{a_2}-D_{a_2},\ \cdots,\ T_{a_1}+T_{a_2}+\cdots T_{a_n}-D_{a_n})$$

极小化？

18.〔M30〕（连接查找）假定N个记录被放置在一个线性数组$R_1\cdots R_N$中，记录R_i被查找的概率为p_i。若每次查找都在刚才离开处开始，则这个查找过程称为“连接的”。如果诸连续的查找都是独立的，则所需的平均时间为$\sum_{1\leqslant i,j\leqslant N}p_ip_jd(i,j)$，其中$d(i,j)$表示从位置 i 处开始并在位置 j 处结束的一次查找所需的时间。若把这个模型应用到例如磁盘文件的寻找时间上，则 $d(i,j)$ 是为从圆柱面 i 扫视到圆柱面 j 所需的时间。

本题的目的，是对于连接的查找定出各记录的最优设置，其中规定 $d(i,j)$ 是$|i-j|$的一个递增函数，即对于$d_1<d_2<\cdots<d_{N-1}$均有 $d(i,j)=d_{|i-j|}$(d_0 的值是不相干的)。证明，在这种情况下，在所有$N!$个排列当中，各记录被最优排列的充分必要条件是 $p_1\leqslant p_N\leqslant p_2\leqslant p_{N-1}\leqslant\cdots\leqslant p_{\lfloor N/2\rfloor+1}$ 或者 $p_N\leqslant p_1\leqslant p_{N-1}\leqslant p_2\leqslant\cdots\leqslant p_{\lceil N/2\rceil}$（于是，概率的一种“管风琴”排列是最好的，如图 2 所示）。提示：考虑任一排列，其中概率分别为 $q_1q_2\cdots q_ksr_k\cdots r_2r_1t_1t_2\cdots t_m$，这里 $m\geqslant 0$，$k>0$，$N=2k+m+1$。证明若把它改排为 $q'_1q'_2\cdots q'_ksr'_k\cdots r'_2r'_1t_1\cdots t_m$，则效果更好，其中$q'_i=\min(q_i,\ r_i)$，$r'_i=\max(q_i,\ r_i)$，除非对所有 i，$q'_i=q_i$ 和 $r'_i=r_i$ 或对所有 i，j，$q'_i=r_i$，$r'_i=q_i$和$t_j=0$。当 s 不出现且$N=2k+m$时亦然。

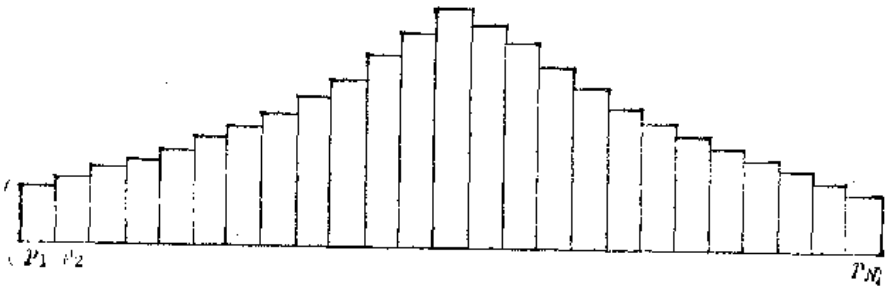


图 2　概率的“管风琴排列”使一个连接查找的平均寻找时间极小化

19.〔M20〕 继续习题18，当函数 $d(i,j)$ 有这样一个性质：即对于所有 $i\neq j$，$d(i,j)+d(j,i)=c$ 时，什么是连接查找的最优排列？〔例如，在没有向后读能力的

带上就出现这种情况。因为我们可能不知道应该往哪个方向进行查找；譬如，对于 $i<j$，我们有，$d(i,j)=a+b(L_{i+1}+\cdots+L_j)$，$d(j,i)=a+b(L_{j+1}+\cdots+L_N)+r+b(L_1+\cdots+L_i)$，其中，$r$ 为重绕时间。〕

20.〔M28〕 继续习题18，对于 $d_1<d_2<\cdots$，当函数 $d(i,j)=\min(d_{|i-j|}, d_{n-|i-j|})$ 时，什么是连接查找的最优排列？〔例如：在两路链接的循环表中，或者在一个两路移位寄存器的存储设备上就出现这种情况。〕

21.〔M28〕 考虑一个 n 维立方体，其顶点坐标为 $(d_n,\cdots,d_1)$，且 $d_i=0$ 或 1；两个顶点称为相邻的，如果它们恰有一坐标不同。假设有 2^n 个数 $x_0\leqslant x_1\leqslant\cdots\leqslant x_{2^n-1}$ 的一个集合，以使 $\Sigma|x_i-x_j|$ 成为极小这样一种方式，赋于 2^n 个顶点，其中，x_i 和 x_j 是赋给相邻顶点的两个数，求和即对所有这样的 x_i 和 x_j 进行。试证明：若每个 x_i 都被赋给一个这样的顶点，该顶点坐标的二进表示恰为 i，则上述极小值被达到。

22.〔20〕 假设你要查找一个大型文件，不是查找相等的，而是找出最接近于一个给定键的1000个记录。最接近的意义是：对于某个给定的距离函数 d，这1000个记录的 $d(K_i,K)$ 值为最小。试问对于这样一个顺序查找，什么数据结构最为适合？

6.2 通过键比较进行查找

在这一节中，我们将讨论以键的一个线性次序（例如，字母顺序或数值顺序）为基础的查找方法，在对给定的变元 K 和表中的一个键 K_i 进行比较之后，这个查找以三种不同的方式继续，分别取决于 $K<K_i$，$K=K_i$，或 $K>K_i$。6.1节的顺序查找方法实际上局限于两路判断（$K=K_i$ 和 $K\neq K_i$）。但如果我们摆脱顺序存取的限制，则就有可能有效地利用一个次序关系。

6.2.1 查找一有序的表

如果某人递给你一本大电话簿，而且请你找一个人的名字，此人的电话号码是795-6841，那你该做什么呢？处理这个问题，没有比6.1节的顺序方法更好的方法了。（然而，一个机灵的私人侦探可能试图拨这个号并想猜出谁接电话；或者他在电话公司有一个朋友，能够查找一个按号码而不是按名字分类的特殊目录。）问题在于通过当事者的名字而不是通过号码来找一个项目要容易得多，尽管在电话目录中包含了这两种情况下所需要的所有信息。当必须查找一个大型文件时，顺序扫描几乎是办不到的，而一个次序关系却大大地简化了这项工作。

使用已经讨论过的许多排序方法（第5章），把文件重新排为有序的，使其便于查找，这不会太困难。当然，如果只需对这个表查找一次，则进行顺序查找比对这个文件进行完整的排序要快；但如果要在同一文件中进行重复查找，那么，把它排成顺序就更好些。因此在这一节中，我们将集中研究这样一些方法，当对表的各项作随机存取时，这些方法适于查找其键为有序的表

$$K_1<K_2<K_3<\cdots<K_N$$

在这样的表中，比较 K 和 K_i 后，我们有

- $K<K_i$〔不必考虑 $R_i,R_{i+1},\cdots,R_N$〕

或 · $K=K_i$〔查找完成〕

或 · $K>K_i$〔不必考虑$R_1, R_2, \cdots, R_i$〕除非 i 靠近该表末尾，这三种情况均获实质性进展，这就是为什么次序能引出一个有效的算法的原因。

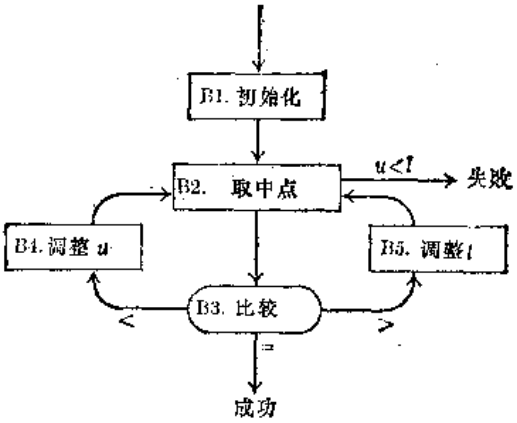


图 3　二分查找

二分查找　也许想到的头一个这样的方法是从K同表的中间键比较开始：这个探查的结果指出下一次应该查找表的哪一半，并可再次使用相同的步骤，比较K和选中的一半的中间键，等等。在至多进行了 $\log_2 N$次比较之后，或者找到这个键，或者确认它不存在。这个步骤有时称为“对数查找”或“二等分法”，但最普遍的是叫二分查找。

尽管二分查找的思想是相当直截了当的，但其细节可能有些技巧，而且有许多好的程序员在他们头几次试用时竟把它弄错了。这个算法的最普遍的正确形式之一如下，其中利用两个指针 l 和 u 来指出当前查找的下限和上限：

算法B（二分查找）　给定其键为递增次序$K_1<K_2<\cdots<K_N$的记录$R_1, R_2, \cdots, R_N$的一个表，本算法查找一个给定的变元K。

B1.〔初始化〕　置 $l \leftarrow 1$，$u \leftarrow N$。

B2.〔取中点〕(这时我们知道如果K在此表中，则它满足$K_l \leqslant K \leqslant K_u$。下面习题 1 中有关于这种情况的一个更精确的陈述。）如果 $u<l$，则这个算法以失败告终。否则，置 $i \leftarrow \lfloor (l+u)/2 \rfloor$，这是该表区域的近似中点。

B3.〔比较〕　如果$K<K_i$，则转到B4；如果$K>K_i$，则转到B5；如果$K=K_i$，则算法成功地结束。

B4.〔调整 u〕　置 $u \leftarrow i-1$，并返回B2。

B5.〔调整 l〕　置 $l \leftarrow i+1$，并返回B2。

图 4 说明了该二分查找算法的两种情况，第一个是查找变元653，它在表中出现，而后查找400，它不存在。方括号指出 l 和 u，下边划线的键表示K_i。在这两个例子中，进行了四次比较之后，查找结束。

a）查找 653

〔061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908〕
061	087	154	170	275	426	503	509	〔512	612	653	677	703	765	897	908〕
061	087	154	170	275	426	503	509	〔512	612	653〕	677	703	765	897	908
061	087	154	170	275	426	503	509	512	612	〔653〕	677	703	765	897	908

b）查找 400

〔061	087	154	170	275	426	503	509	512	612	653	677	703	765	897	908〕
〔061	087	154	170	275	426	503〕	509	512	612	653	677	703	765	897	908
061	087	154	170	〔275	426	503〕	509	512	612	653	677	703	765	897	908
061	087	154	170	〔275〕	426	503	509	512	612	653	677	703	765	897	908
061	087	154	170	275〕	〔426	503	509	512	612	653	677	703	765	897	908

图 4　二分查找的例子

程序B（二分查找） 如同在6.1节的程序中那样，我们这里假定K_i是出现在单元KEY+i中的全字长键。下列代码使用rI1≡l，rI2≡u，rI3≡i。

01	START	ENT1	1	1	*B*1. 初始化 $l \leftarrow 1$
02		ENT2	N	1	$u \leftarrow N$
03		JMP	2F	1	转B2
04	5H	JE	SUCCESS	$C1$	如果$K=K_i$则转移
05		ENT1	1,3	$C1-S$	*B*5. 调整 l，$l \leftarrow i+1$
06	2H	ENTA	0,1	$C+1-S$	*B*2. 取中点
07		INCA	0,2	$C+1-S$	rA←l+u
08		SRB	1	$C+1-S$	rA←⌊rA/2⌋ (rX也改变)
09		STA	TEMP	$C+1-S$	
10		CMP1	TEMP	$C+1-S$	
11		JG	FAILURE	$C+1-S$	如果$u<l$，则转移
12		LD3	TEMP	C	$i \leftarrow$中点
13	3H	LDA	K	C	*B*3. 比较
14		CMPA	KEY,3	C	
15		JGE	5B	C	如果$K \geqslant K_i$则转移
16		ENT2	-1,3	$C2$	*B*4. 调整u，$u \leftarrow i-1$
17		JMP	2B	$C2$	转B2

这个步骤不象我们见过的其它算法那样同MIX十分"光滑地"融成一体，因为MIX不允许在变址寄存器中作许多算术运算。运行时间为$(18C-10S+12)u$，其中$C=C1+C2$是所作的比较数（即步骤B3被执行的次数），而且$S=0$或1依赖于结果成功与否。注意，这个程序的08行"右移一个二进位"仅在二进制的MIX机器上是合法的；对于一般的字节大小，这条指令应该以"MUL=1// 2+1="代替，从而把运行时间增加到$(26C-18S+20)u$。

树表示 为了真正理解在算法B中所发生的事情，最好把它想象成一株二分判定树，如同图5所示，图中$N=16$。

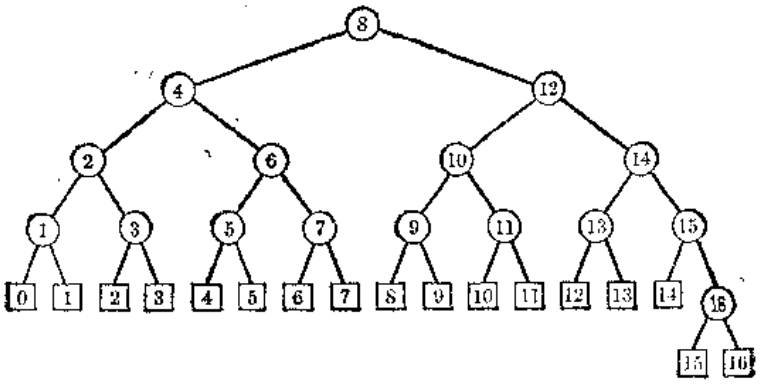


图5 $N=16$时二分查找的一株二叉树

当N为16时，这个算法所作的头一个比较是$K:K_8$，通过图中的根结点⑧来表示。然后，如果$K<K_8$，则这个算法沿着左子树，比较K与K_4；类似地，如果$K>K_8$，则使用

右子树。一个不成功的查找将通向编号为[0]到[N]的一个“外部”方形节点；例如，我们达到节点[6]当且仅当$K_6<K<K_7$。

在N个记录上进行二分查找的二叉树可以构造如下：如果$N=0$，则这个树简单地是[0]，否则根节点是

$$\lceil N/2 \rceil$$

左子树是$\lceil N/2 \rceil-1$个节点对应的二叉树，右子树是$\lceil N/2 \rceil$个节点对应的二叉树，而且所有节点号都增加$\lceil N/2 \rceil$。

类似地，任何借助于比较法查找长度为N的一个有序表的算法，都可以表示成一株二叉树，其中节点由1到N编号（除非算法作多余的比较）。反之，任何二叉树都对应一个查找有序表的有效方法；我们简单地用对称方法从左到右给诸节点编号

$$\boxed{0}\quad ①\quad \boxed{1}\quad ②\quad \boxed{2}\quad \ldots\quad \boxed{N-1}\quad Ⓝ\quad \boxed{N} \tag{1}$$

如果在算法B中输入的查找变元是K_{10}，则此算法进行$K>K_8$，$K<K_{12}$，$K=K_{10}$的比较。这对应于图5中从根到⑩的通路。类似地，对其它键来说，算法B的行为对应于从这株树的根导出的其它通路。因此，构造对应于算法B的二叉树的方法，使我们易于对N用归纳法证明下列结果：

定理B 如果$2^{k-1}\leqslant N<2^k$，则利用算法B进行一个成功的查找需要作（$\min 1, \max k$）次比较。如果$N=2^k-1$，则一个不成功的查找需要k次比较；如果$2^{k-1}\leqslant N<2^k-1$，则一个不成功的查找需要$k-1$次或k次比较。

对二分查找的进一步分析 （对数学没有兴趣的读者，请跳到等式(4)）树表示也向我们表明了怎样以一种简单的方式来计算平均比较数。令C_N是在一次成功的查找中的平均比较数，其中假定N个键中每一个都有同等可能成为变元；并令C'_N是在一次不成功的查找中的平均比较数，其中假定键之间和端点值外部的$N+1$个区间每一个都是同等可能的，于是由内部和外部路径长度的定义，我们有

$$C_N=1+\frac{\text{树的内部路径长度}}{N}$$

$$C'_N=\frac{\text{树的外部路径长度}}{N+1}$$

我们在等式2.3.4.5-3中已经看到，外部路径的长度总是比内部路径长度大$2N$，因此，在C_N和C'_N之间有一个相当意外的关系

$$C_N=\left(1+\frac{1}{N}\right)C'_N-1 \tag{2}$$

这个公式是希巴德给出的[JACM 9 (1962), 16-17]，它对适用于二叉树的所有查找方法都成立。即，对于所有基于非冗余比较的方法都成立。C_N的方差可用C'_N的方差来表示(见习题25)。

由上面的公式我们看出，通过比较进行查找的一种“最好的”方式，是在所有具有N

个内部节点的二叉树中，具有极小外部路径长度的树。幸而，可以证明，在这个意义下，对于所有的N，算法B是最优的；因为我们已经看到，当且仅当一株二叉树的所有外部节点都出现在至多两个相邻的级上时，该二叉树具有极小路径长度。由此得出，对应算法B的外部路径长度为

$$(N+1)(\lfloor \log_2 N \rfloor + 2) - 2^{\lfloor \log_2 N \rfloor + 1} \tag{3}$$

（见等式5.3.1-33）由这一公式和（2）我们就能计算精确的平均比较数，其中假定所有查找变元都是同等可能的。

$$N = 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9\ 10\ 11\ 12\ 13\ 14\ 15\ 16$$

$$C_N = 1 \quad 1\frac{1}{2} \quad 1\frac{2}{3} \quad 2 \quad 2\frac{1}{5} \quad 2\frac{2}{6} \quad 2\frac{3}{7} \quad 2\frac{5}{8} \quad 2\frac{7}{9} \quad 2\frac{9}{10} \quad 3 \quad 3\frac{1}{12} \quad 3\frac{2}{13} \quad 3\frac{3}{14} \quad 3\frac{4}{15} \quad 3\frac{6}{16}$$

$$C'_N = 1 \quad 1\frac{2}{3} \quad 2 \quad 2\frac{2}{5} \quad 2\frac{4}{6} \quad 2\frac{6}{7} \quad 3 \quad 3\frac{2}{9} \quad 3\frac{4}{10} \quad 3\frac{6}{11} \quad 3\frac{8}{12} \quad 3\frac{10}{13} \quad 3\frac{12}{14} \quad 3\frac{14}{15} \quad 4 \quad 4\frac{2}{17}$$

一般地说，如果$k = \lfloor \log_2 N \rfloor$，则我们有（参考等式5.3.1-34）

$$\begin{aligned} C_N &= k + 1 - (2^{k+1} - k - 2)/N = \log_2 N - 1 + \epsilon + (k+2)/N \\ C'_N &= k + 2 - 2^{k+1}/(N+1) = \log_2 N + \epsilon' \end{aligned} \tag{4}$$

其中$0 \leqslant \epsilon$，$\epsilon' < 0.0861$。

总结：算法B决不做多于$\lfloor \log_2 N \rfloor + 1$次比较，它在一次成功的查找中平均进行大约$\log_2 N - 1$次比较。再没有比这更好的基于比较的查找方法了。程序B的平均运行时间近似为

$$\begin{aligned} &(18\log_2 N - 16)\,u \quad \text{对于一次成功的查找} \\ &(18\log_2 N + 12)\,u \quad \text{对于一次不成功的查找} \end{aligned} \tag{5}$$

这里的前提是假定这个查找的所有结果都是同等可能的。

一个重要的变形 不使用上述查找中的三个指针l、i和u，而仅使用如下两个量是有趣的，即当前的位置i和它的变化速度δ；在每次不相等的比较之后，我们可以置$i \leftarrow i \pm \delta$和$\delta \leftarrow \delta/2$（近似地）。这样做是可能的，但正如下面的算法中那样，对其细节需极端小心才成，简单化的解决方法会引起失误！

算法U（均匀的二分查找） 给定一个其键处于递增次序$K_1 < K_2 < \cdots < K_N$的记录R_1，$R_2 \cdots$，R_N的表，本算法查找变元K。如果K为偶数，则算法有时将涉及一个虚拟键K_0，K_0应被置成$-\infty$（或小于K的任意值）。我们假定$N \geqslant 1$。

U1.〔初始化〕 置$i \leftarrow \lceil N/2 \rceil$，$m \leftarrow \lfloor N/2 \rfloor$。

U2.〔比较〕 如果$K < K_i$，则转到U_3；如果$K > K_i$，则转到U_4；如果$K = K_i$，则算法成功地结束。

U3.〔i减值〕（我们已经认准包含m或$m-1$个记录的一个查找区间；i恰指向该区间的右端）如果$m = 0$，则这个算法以失败告终。否则置$i \leftarrow i - \lceil m/2 \rceil$；然后置$m \leftarrow$

$\lfloor m/2 \rfloor$并返回U_2。

U4.〔i增值〕（我们已经认准包含m或$m-1$个记录的一个查找区间；i恰指向该区间的左端）如果$m=0$，则这个算法以失败告终。否则置$i \leftarrow i+\lceil m/2 \rceil$；然后置$m \leftarrow \lfloor m/2 \rfloor$并返回$U_2$。

图6示出当$N=10$时查找对应的二叉树。当查找失败时，此算法可能在结束前作一次冗余的比较，如图中阴影节点所示。我们可以称这个查找过程为均匀的，因为在级l上每个节点的号码与它在$l-1$级上的相应祖宗的号码之差，是一个常数δ。

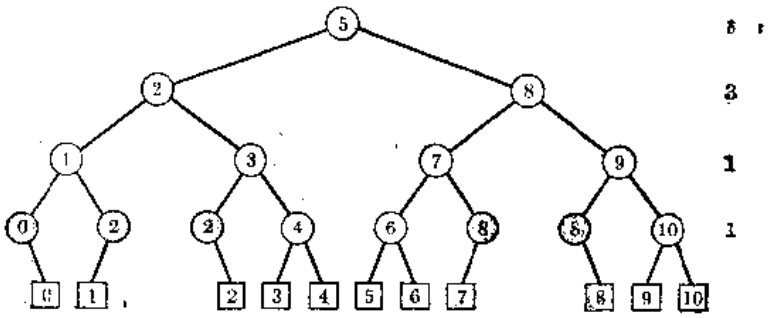


图6　当$N=10$时一株“均匀的”二分查找树

算法U的理论根据可阐述如下：假如我们有长度为$n-1$的一个区间要查找；同中间的元素（n是偶数）或者同两个中间的元素之一（n为奇数）进行比较后，剩下了长度为$\lfloor n/2 \rfloor-1$和$\lceil n/2 \rceil-1$的两个区间。在重复这一过程k次之后，我们得到了2^k个区间，其中最小者长度为$\lfloor n/2^k \rfloor-1$，最大者长度为$\lceil n/2^k \rceil-1$。因此，在同一级上两个区间的长度至多差1；这就使我们有可能选择一个适当的“中间”元素，而无须记住精确的长度。

算法U的主要优点是，我们全然不必保持m的值，只需要查一张在树的各级中使用的各δ值的短表。于是，这个算法简化成下列过程，它在二进计算机上或在十进计算机上同样适用。

算法C（均匀二分查找）这个算法与算法U相象，但它使用一个辅助表来代替涉及m的计算。这个表的项目是

$$\mathrm{DELTA}[j]=\left\lceil \frac{n+2^{j-1}}{2^j} \right\rceil \text{对于} 1 \leqslant j \leqslant \lfloor \log_2 N \rfloor+2 \qquad (6)$$

C1.〔初始化〕置$i \leftarrow \mathrm{DELTA}[1]$，$j \leftarrow 2$。

C2.〔比较〕如果$K<K_i$，则转到C3；如果$K>K_i$则转到C4；如果$K=K_i$，此算法成功地结束。

C3.〔i减值〕如果$\mathrm{DELTA}[j]=0$，则此算法以失败告终。否则，置$i \leftarrow i-\mathrm{DELTA}[j]$，$j \leftarrow j+1$，并转到C2。

C4.〔i增值〕如果$\mathrm{DELTA}[j]=0$，则此算法以失败告终，否则，置$i \leftarrow i+\mathrm{DELTA}[j]$，$j \leftarrow j+1$，并转到C2。

习题8证明，这个算法仅当N是偶数时才涉及人为的键$K_0=-\infty$。

程序C（均匀二分查找）这个程序以rA≡K，rI1≡i，rI2≡j，rI3≡DELTA[j]，利用算法C，所作的事和程序B相同。

01	START	ENT1	N + 1/2	1	C1. 初始化
02		ENT2	2	1	$j \leftarrow 2$

03		LDA	K	1	
04		JMP	2F	1	
05	3H	JE	SUCCESS	$C1$	如果$K=K_i$则转移
06		J3Z	FAILURE	$C1-S$	如果DELTA[i]=0，则转移
07		DEC1	0,3	$C1-S-A$	$C3$. i减值
08	5H	INC2	1	$C-1$	$j \leftarrow j+1$
09	2H	LD3	DELTA,2	C	$C2$. 比较
10		CMPA	KEY,1	C	
11		JLE	3B	C	如果$K \leqslant K_i$则转移
12		INC1	0,3	$C2$	$C4$. i增值
13		J3NZ	5B	$C2$	如果DELTA[i]$\neq 0$则转移
14	FAILURE	EQU	*	$1-S$	如果不在表中则转出

在一次成功的查找中，这个算法对应的二叉树与算法B的二叉树有相同的内部路径长度，所以平均比较次数C_N和以前一样。在一次不成功的查找中，算法C总是恰好进行$\lfloor \log_2 N \rfloor + 1$次比较。程序C的总运行时间在左、右分支之间不是十分对称的，因为$C1$的权比$C2$要大。但习题9表明，$K<K_i$同$K>K_i$的机会大体上相同；因此程序C近似地花费

$$(8.5\log_2 N - 6)u \quad \text{对于一次成功的查找}$$
$$(8.5\lfloor \log_2 N \rfloor + 12)u \quad \text{对于一次不成功的查找} \tag{7}$$

这个速度相当于程序B的两倍多，这里不使用二进计算机的任何特殊性质，然而对于程序B的运行时间（5）还假定MIX有一条“右移二进位”指令。

L. E. 沙尔(L. E. Shar)于1971年提议对二分查找作另一种修改。这在某些计算机上实现起来会更快一些，因为在头一步之后它是均匀的，而且它不需要表。头一步是比较K和K_i，其中，$i=2^k$，$k=\lfloor \log_2 N \rfloor$。如果$K<K_i$，则我们使用诸$\delta$等于$2^{k-1}, 2^{k-2}, \cdots, 1, 0$的一个均匀查找。另一方面，如果$K>K_i$，则重置$i$为$i'=N+1-2^l$，其中，$l=\lceil \log_2(N+1-2^k) \rceil$，并假设头一个比较实际上是$K>K_{i'}$，且使用诸$\delta$等于$2^{l-1}, 2^{l-2}, \cdots, 1, 0$的一个均匀查找。

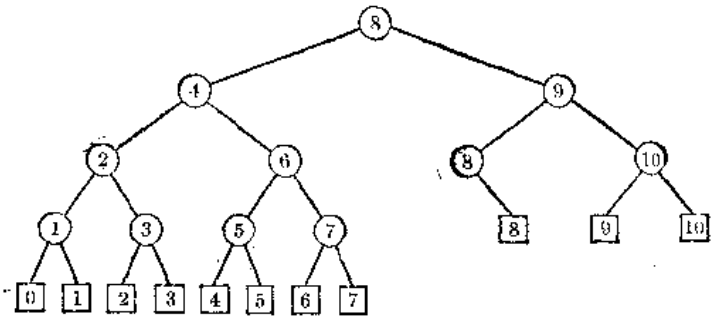


图7　当$N=10$时，沙尔的准均匀二叉树

图7示出了当$N=10$时沙尔的方法。和以前的算法一样，它的比较次数不会超过$\lfloor \log_2 N \rfloor + 1$；尽管有时某些步骤是多余的，但从平均比较次数来看，它仍不失为最好的算法之一（参考习题12）。

二分查找还有另一种修正，当N非常大时，它提高了所有上述方法的速度，这个方法在习题23中讨论，更快的一个方法见习题24。

斐波那契查找 在多阶段合并中，我们已经看到，斐波那契数可以起到和 2 的乘方相类似的作用。在查找中也出现类似的现象，在这里斐波那契数为我们提供了二分查找的另一个方案，从而得到在某些计算机上更可取的方法，因为它仅包含加法和减法，没有除以 2 的除法。我们所要讨论的过程，应同一个确定单峰函数极大值位置的，重要的数值“斐波那契查找”过程区别开来〔后者参考 *Fibonacci Quarterly* 4 (1966)，265-269〕，名称的相似性已经导致了一些混乱。

乍一看，斐波那契查找技术似乎非常神秘，如果我们简单地把程序拿来并试图看看它在干些什么的话，它似乎是在变魔术。但只要把对应的查找树画出来，则神秘感立即就消失了。因此，我们将通过考察“斐波那契树”来开始对这个方法进行研究。

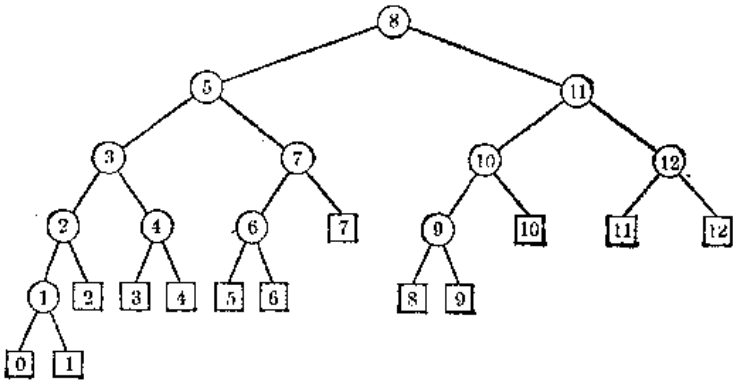


图 8　6 阶斐波那契树

图 8 示出了 6 阶的斐波那契树，注意，与我们已经考虑过的其它树相比，它看起来更象现实生活中的灌木，这也许是因为许多自然过程都满足一种斐波那契规律的缘故。一般地说，阶数 k 的斐波那契树有 $F_{k+1}-1$ 个内部（圆形）节点和 F_{k+1} 个外部（正方形）节点，这株树可构造如下：

如果 $k=0$ 或 $k=1$，则此树就是$\boxed{0}$。

如果 $k\geqslant 2$，则树根为$\boxed{F_k}$；左子树是阶数为 $k-1$ 的斐波那契树；右子树是阶数为 $k-2$ 且所有编号都增加 F_k 的斐波那契树。

注意，除外部节点外，每个内部节点的两个儿子的编号和他们的父亲的编号相差同一个数，这个数就是一个斐波那契数。例如，在图 8 中，$5=8-F_4$ 和 $11=8+F_4$。当差是 F_j 时，左下分枝对应的斐波那契差数是 F_{j-1}，而右下分枝的差数为 F_{j-2}，例如，$3=5-F_3$，而 $10=11-F_2$。

如果我们把这些观察同识别外部节点的一个适合的机制结合起来，则就得到下列方法：

算法F （斐波那契查找）给定一个其键为递增次序 $K_1<K_2<\cdots<K_N$ 的记录 $R_1, R_2, \cdots, R_N$ 的表，本算法查找一个给定的变元 K。

为了叙述方便，这个算法假定 $N+1$ 是一个完全的斐波那契数 F_{k+1}，只要适当地初始化（见习题 14），就不难使这个方法对于任何 N 都有效。

F1.〔初始化〕 置 $i\leftarrow F_k$，$p\leftarrow F_{k-1}$，$q\leftarrow F_{k-2}$（在整个算法中，p 和 q 总是相继的斐波那契数）。

F2.〔比较〕 如果 $K<K_i$，则转到步骤 F3；如果 $K>K_i$，则转到 F4；而如果 $K=K_i$，则这算法成功地结束。

F3.〔i 减值〕 如果 $q=0$，则这算法以失败告终，否则置 $i\leftarrow i-q$，并置（p，

$q)\leftarrow(q,\ p-q)$，然后返回 F2。

F4〔i 增值〕 如果 $p=1$，则这算法以失败告终，否则置 $i\leftarrow i+q$，$p\leftarrow p-q$，然后 $q\leftarrow q-p$，并返回 F2。

以下的 MIX 实现，通过设置两个内循环而赢得了速度，一个内循环把 p 放 rI2 中，q 放在 rI3 中，而另一个则把这两个寄存器恰好颠倒过来；这便简化了步骤 F3。事实上，程序在寄存器中真正保留的是 $p-1$ 和 $q-1$，而不是 p 和 q，为的是简化步骤 F4 中的判断"$p=1$?"。

程序 F（斐波那契查找）rA≡K，rI1≡i，(rI2, rI3)≡$p-1$，(rI3, rI2)≡$q-1$。

01	START	LDA	K	1	$F1$. 初始化
02		ENT1	F_k	1	$i\leftarrow F_k$
03		ENT2	$F_{k-1}-1$	1	$p\leftarrow F_{k-1}$
04		ENT3	$F_{k-2}-1$	1	$q\leftarrow F_{k-2}$
05		JMP	F2A	1	转到步骤 F2
06	F4A	INC1	1,3		
07		DEC2	1,3		
08		DEC3	1,2		
09	F2A	CMPA	KEY,1		
10		JL	F3A		
11		JE	SUCCESS		
12		J2NZ	F4A		
13		JMP	FAILURE		
14	F3A	DEC1	1,3		
15		DEC2	1,3		
16		J3NN	F2B		
17		JMP	FAILURE		
18	F4B	INC1	1,2	$C2-S-A$	$F4$. i 增值 $i\leftarrow i+q$
19		DEC3	1,2	$C2-S-A$	$p\leftarrow p-q$
20		DEC2	1,3	$C2-S-A$	$q\leftarrow q-p$
21	F2B	CMPA	KEY,1	C	$F2$. 比较
22		JL	F3B	C	如果 $K<K_i$ 则转到 F3
23		JE	SUCCESS	$C2$	如果 $K=K_i$ 则转出
24		J3N2	F4B	$C2-S$	如果 $p\neq 1$ 则转到 F4
25		JMP	FAILURE	A	如果不在表中则转出
26	F3B	DEC1	1,2	$C1$	$F3$. i 减值，$i\leftarrow i-q$
27		DEC3	1,2	$C1$	$p\leftarrow p-q$
28		J2NN	F2A	$C1$	如果 $q>0$ 则交换寄存器
29		JMP	FAILURE	$1-S-A$	如果不在表中则转出

习题 18 中分析了这个程序的运行时间，图 8 表明，而且分析证明，通常取左分枝的机会比右分枝稍多些。设 C、$C1$ 和 $(C2-S)$ 分别为步骤 F2、F3 和 F4 被执行的次数，则我们有

$$C=(\text{ave}\phi k/\sqrt{5}+O(1),\ \max k-1)$$
$$C1=(\text{avc}\,k/\sqrt{5}+O(1),\ \max k-1) \tag{8}$$
$$C2-S=(\text{ave}\phi^{-1}k/\sqrt{5}+O(1),\ \max\lfloor k/2\rfloor)$$

于是取左分枝的次数大约是取右分枝次数的 $\phi=1.618$ 倍（这是我们能够猜想得到的一个事实，因为每个探测都把剩下的区间分成为两部分，且左边部分大约是右边部分的 ϕ 倍那么大）。因此程序F总的平均运行时间近似于

$$(6\phi k/\sqrt{5}-(2+22\phi)/5)u\approx(6.252\log_2 N-4.6)u \quad \text{对于一次成功的查找} \tag{9}$$

$$(6\phi k/\sqrt{5}+(58/(27\phi))/5)u\approx(6.252\log_2 N+5.8)u \quad \text{对于一次不成功的查找}$$

程序F比程序C稍微快些，尽管最坏情况下运行时间（大约 $8.6\log_2 N$）稍微慢些。

内插查找 让我们暂时忘记计算机，来考虑人们如何真正地进行一次查找。有时日常生活会给我们提示好的算法。

想象你自己在一部字典中查一个字，你大概不是从查中间的页开始，然后查 1/4 或 3/4 处的页，等等，如同在二分查找中那样。你更不大可能去使用斐波那契查找！

如果你所要的字是以字母A开始的，那你或许会从靠近字典前头的地方开始。事实上，许多字典都有“书边索引”，它标出以一个固定字母开始的单字的起始页。这个书边索引技术不难用到计算机上去，而且它将加速查找，6.3 节中将剖析这一技术。

即使在找到了查找的起始点之后，你的动作也还不太象我们已经讨论过的方法。如果你注意到，所要找的字按字母次序比你正在查找的页上的字大很多，则在你开始进行下次查找之前，就要翻过好多页。这和上面的一些算法大不相同，那些算法并没有对“大得多”和“稍微大些”加以区别。

这些考虑提示了一个堪称为“内插查找”的算法：当知道 K 位于 K_l 和 K_u 之间时，我们可以把下一次探测的位置选在位于 l 和 u 之间大约 $(K-K_l)/(K_u-K_l)$ 这一点上。这里假定键是数值的，而且键在整个这一区间里以大体不变的方式增值。

内插查找渐近地优于二分查找；如果表中的键是随机分布的，则二分查找的每一步把查找工作量从 n 降到 $\frac{1}{2}n$，而内插查找则把 n 降到 $\sqrt{n}$。因此，可以证明，平均说来，内插查找花费 $\log_2\log_2 N$ 步（见习题 22）。

可惜，计算机的模拟实验表明，内插查找并不把比较数减少到足以补偿所需的多余计算，当查找存储在高速存储器内的一份表时，除非 N 很大，否则 $\log_2\log_2 N$ 和 $\log_2 N$ 之间的差别并不很大，而且典型的文件也不是充分地随机的。仅仅在应用于外部存储设备的外部查找时，内插查找才获得了有限的成功（注意，用手查阅字典实际上是外部的，而不是一个内部的查找）。我们稍后将讨论外部查找。

历史和参考文献 已知最早的把一长串项目排成顺序以便于查找的例子，是大约公元前 200 年就编制成的值得注意的巴比伦人的艾娜基比特-安奴（Inakibit-Anu）倒数表，这个陶土做的表看来是三个这样的表的头一个，它包含 500 个以上高精度的六十进制数及其倒数，并把它们排成了词典顺序。例如，该表包括有下列项的序列：

01	13	09	34	29	08	08	53	20	49	12	27							
01	13	14	31	52	30				49	09	07	12						
01	13	43	40	48					48	49	41	15						
01	13	48	40	30					48	46	22	59	25	25	55	33	20	
01	14	04	26	40					48	36								

对这样500个项进行排列的任务，在当时可利用的技术下，确实是非凡的〔关于进一步的细节，见D. E. Knuth，*CACM* 15 (1972)，671-677)〕。

把数值排成次序是相当自然的，但是字母或字之间的次序关系并不是那样容易看出的。然而对于单个字母的一个整理好的序列已经出现在最古老的字母表中。例如，许多圣经赞美诗都有这样的诗句，它遵循一个严格的字母序列，头一个诗句以 *a* 开始，第二个以 *b* 开始，等等；这有助于记忆。终于，字母的标准序列为闪米特人和希腊人用来表示数；例如，α，β，γ分别表示1，2，3。

但把字母表的次序用于整个的字，似乎是一项更晚些的发明；有时我们可能会把它想成是显然的，但这必须教给孩子们，而且在历史上的某个时候，曾也有必要教给成年人！在爱琴岛上已经发现了大约公元300年前的若干表，给出了某些迷信的宗教中的人名；这些表按字母排列，但仅按头一个字母，因此只表示了自左至右进行基数排序的第一次扫描。公元134-135年间的某些希腊文稿包含了一些分类账的片断，它列出了按头两个字母排序的纳税人的名称。阿波罗尼厄斯索非斯塔（Apollonius Sophista）在他的长而和谐的荷马诗中（公元一百年），把字母顺序用于头两个字母上，而且通常也用于随后的字母上。还有一些更完美的字母顺序化的例子是非常有名的，最突出的有盖伦（Galen）的《希波克拉底释注》(Hippocratic Glosses)(大约公元200年)，但这些都是非常罕见的。因此，在圣人伊西多拉斯（St. Isidorus）的《词源》(Etymologiarum)(大约公元630年，第十册）中，仅按它们的头一个字母来排列词序；而《释注大全》(Corpus Glossary)（约公元725年）仅使用每个字的头两个字母。这后两部著作也许是中世纪编纂的最大的非数值方面的数据文件。

在乔瓦尼·第·吉诺瓦（Giovanni di Genoa）的《万灵药》(Catholicon)（1286年)一书前，我们还没有见到过对正确的字母顺序的专门描述，在他的前言中，乔瓦尼说明

*amo*在 *bibo* 之前
*abeo*在 *adeo* 之前
*amatus*在 *amor* 之前
*imprudens*在 *impudens* 之前
*iusticia*在 *iustus* 之前
*polisintheton*在 *polissenus* 之前

（由此给出了按第一个，第二个，…，第六个字母排列的情况的例子）“余类推”。他述说了为想出这些规则所需要的紧张的努力。“因此，好读者，我请求你，不要把我这巨大的劳动和这个次序当作是不值钱的。”

在发明印刷术之前，L. W. 戴利对字母顺序进行过详细研究〔见Lloyd W. Daly，Collection Latomus 90 (1967)，100pp.〕，他发现了某些有趣的旧手稿，它们显然是一些草稿纸，上面有按头几个字母排序的一些单字（见该专著的87-90页）。

第一本英文字典，〔见 Robert Cawdrey, Table Alphabeticall (London, 1604)〕包含下列的指示：

如果你想要查找的字是以（a）开始的，则在这个表的开头处查找，而如果是以（u）开始，则在靠近这个表的末尾处查找。又，如果字以（ca）开始，则在字母（c）的开头处查找，而如果以（cu）开始，则在靠近该字母的末尾处查找，如此类推。说明这样一点是有趣的，考德雷在编写他的字典的过程中，看来也曾自学过怎样来编排字母；在最初的一些页上，出现过许多放错位置的字，而最后部分的字母顺序就好得多了。

约翰·莫茨利在也许是在第一部出版的关于非数值程序设计方法的书中，首先提出了二分查找〔*Theory and techniques for the design of electronic digital computers*, ed. by G. W. Patterson, 1 (1946), 9.7-9.8; 3 (1946), 22.8-22.9〕。这个方法自 50 年代以来变成了“众所周知的”了，但是似乎还没有人对N不具有 2^n-1 这种特殊形式时作过详细讨论〔见 A. D. Booth, *Nature* 176(1955), 565; A. I. Dumey, *Computer and Automation* 5 (December, 1956), 7，其中二分查找称为“第二十个问题”; Daniel D. McCracken, *Digital Computer Programming* (Wiley, 1957), 201-203; 以及 M. Halpern, *CACM* 1, 2 (*February*, 1958), 1-3〕。

D. H. 莱默（D. H. Lehmer）〔*Proc. Symp. Appl. Math.* **10**(1960), 180-181〕显然是第一个发表对于所有N都有效的一个二分查找算法的人，第二个人是 H. 博顿布鲁奇（H. Bottenbruch）〔*JACM* 9 (1962), 214〕。他介绍了算法 B 的一种有趣的变形，它避免了在最后结束之前单作一次相等判断：在步骤 $B2$ 中利用 $i \leftarrow \lceil (l+u)/2 \rceil$ 代替 $\lfloor (l+u)/2 \rfloor$，每当$K \geqslant K_i$ 时他置 $l \leftarrow i$；然后在每一步中 $u-l$ 都减值。最后，当 $l=u$ 时，我们有$K_l \leqslant K \leqslant K_{l+1}$，而且再做一次比较即可判断这个查找是否成功（他假定开始时$K \geqslant K_1$）。这个思想在许多计算机上稍微加速了内循环，而且同样的原理可以为在这一节中讨论的所有算法所使用；但是由于式（2），一次成功的查找平均将要求大约再做一次迭代。由于内循环仅执行大约 $\log_2 N$ 次，因此在这一次额外的迭代与一次更快的循环之间的折衷并不节省时间，除非N非常大（见习题 23）。另一方面，当表含重复键时，博顿布鲁奇的算法将发现一个给定键的最右出现，而这一性质有时很重要。

K. E. 艾弗森〔*A Programming Language* (Wiley, 1962), 141〕给出了算法B的过程，但是没有考虑到不成功查找的可能性。D. E. 克努特〔*CACM* 6 (1963), 556-558〕介绍了算法 B 作为自动画框图系统所用的一个例子。均匀二分查找算法 C，是 1971 年斯坦福大学的 A. K. 钱德拉（A. K. Chandra）对作者建议的。

斐波那契查找是由 D. E. 弗格森〔*CACM* 3 (1960), 648〕发明的，但框图和分析有错。斐波那契树（没有标号）早在许多年前就出现了，它作为珍品收藏在雨果·史泰因豪斯的普及读物《数学万花镜》●〔*Mathematical Snapshots* (New York: Stechert, 1938)〕第一版中；弗格森把这棵树倒画着，因而看起来象一棵真实的树，树的右分枝长度是左分枝的两倍，使得所有叶子都在同一级上出现。

W. W. 彼德森提出了内插查找〔*IBM J. Res. & Devel.* 1 (1957), 131-132〕；他给出

● 裘光明译，中国青年出版社，1953年，第23页。——译注

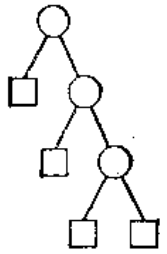
了当键选自一个均匀分布时，对于平均比较数的一个理论估计，但这个估计似乎同实际的模拟经验并不一致。

习题

▶1.〔21〕 证明：如果在二分查找的步骤B2中$u<l$，则我们有$u=l-1$和$K_u<K<K_l$（由约定，假定$K_0=-\infty$和$K_{N+1}=+\infty$，但这些人为的键在本算法中是用不上的，所以它们不必在实际的表中出现）。

▶2.〔22〕 如果我们（a）把步骤B5改为“$l\leftarrow i$”来代替“$l\leftarrow i+1$”，则算法B是否仍将有效地工作？（b）把步骤B4改为“$u\leftarrow i$”以代替“$u\leftarrow i-1$”呢？（c）这两个变化都作呢？

3.〔15〕 什么查找方法对应于树在一次成功的查找中，平均比较次数是多少？在一次不成功的查找中，平均比较次数又是多少？



4.〔20〕 如果使用程序6.1S（顺序查找），一次查找恰花费638个单位时间，则使用程序B（二分查找）须花费多长时间？

5.〔M24〕 对于N的什么值，程序B确实平均慢于一个顺序查找（程序6.1Q′）（假定这个查找是成功的）？

6.〔28〕（E.K.艾弗森）习题5建议我们最好采用一种“混合”方法，即当剩下区间的长度小于某个谨慎地选择的值时，就从二分查找变成顺序查找。试对这样一个查找写出有效的MIX程序并确定最适于“换马”的值。

▶7.〔M22〕 如果我们改变步骤U1使得：（a）i和m都置成等于$\lfloor N/2\rfloor$；（b）i和m都置成等于$\lceil N/2\rceil$，试问算法U是否仍将正确地工作〔提示：假设头一步是“置$i\leftarrow 0$，$m\leftarrow N$（或$N+1$），转到U4”〕？

8.〔M20〕（a）算法C中增量的和$\Sigma_{0\leqslant j\leqslant\lfloor\log_2 N\rfloor+2}$DELTA〔$j$〕是什么？（b）在步骤C2中能出现的极小和极大值是什么？

9.〔M26〕 作为N和S的一个函数，求程序C的频率分析中$C1$、$C2$和A的平均值的精确公式。

10.〔20〕 是否有这么一个$N>1$的值，使算法B和C是精确地等价的，即对于所有的查找变元它们都执行相同的比较序列。

11.〔21〕 说明怎样写出一个近似地含有$7\log_2 N$条指令和大约$4.5\log_2 N$个单位运行时间的对应于算法C的MIX程序。

12.〔20〕 画出对应于$N=12$时沙尔方法的二分查找树。

13. 〔$M24$〕 对于 $1 \leqslant N \leqslant 16$，试编出沙尔方法所作的平均比较次数表，并且既考虑成功的查找也考虑不成功的查找。

14. 〔21〕 说明怎样推广算法F，使它对所有 $N \geqslant 1$ 都可以应用。

15. 〔21〕 图9示出在斐波那契原来的兔子问题中（参考1.2.8节）兔子的线性图表。问在这个图表和正文中讨论的斐波那契树之间是否有一个简单的关系？

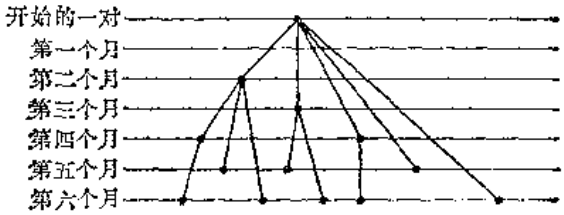


图9 按照斐波那契规则繁殖的兔子对

16. 〔19〕 对于 k 的什么值，k 阶斐波那契树定义了一个最优查找过程，即平均说来所做的比较次数最少？

17. 〔$M21$〕 由习题1.2.8-34（或习题5.4.2-10）我们知道，每个正整数 n 均可唯一地表示为斐波那契数之和 $n = F_{a_1} + F_{a_2} + \cdots + F_{a_r}$，其中 $r \geqslant 1$，对于 $1 \leqslant j < r$，$a_j \geqslant a_{j-1} + 2$，且 $a_r \geqslant 2$。试证明在 k 阶斐波那契树中，由根到节点 n 的路径长度为 $k + 1 - r - a_r$。

18. 〔$M30$〕 作为 k、F_k、F_{k+1} 和 S 的一个函数，求在程序F的频率分析中 $C1$、$C2$ 和 A 的平均值的精确公式。

19. 〔$M42$〕 对于习题14中建议的算法的平均运行时间进行详尽的分析。

20. 〔$M22$〕 在一个二分查找中需要的比较次数近似为 $\log_2 N$，而在斐波那契查找中它大约为 $(\phi/\sqrt{5})\log_\phi N$。本题的目的是证明这些公式都是一个更一般结果的特殊情况。

设 p 和 q 是正数，且 $p + q = 1$。考虑一个查找算法。对于它，给定递增的 N 个数的一份表，首先把变元同第（pN）个键比较，然后在诸较小的块区中迭代这个过程$\left(\text{二分查找有 } p = q = \frac{1}{2}\text{；斐波那契查找有 } P = 1/\phi,\ q = 1/\phi^2\right)$。

如果 $C(N)$ 表示查找长度为 N 的一份表所需要的平均比较次数，则它近似地满足关系式

$$C(1) = 0;\quad C(N) = 1 + pC(pN) + qC(qN) \qquad \text{对于} N > 1$$

这是由于在头一次比较之后，该查找归结为一个 pN 个元素的查找的概率（大约）为 p，归结为一个 qN 个元素的查找的概率为 q 的缘故。当 N 很大时，我们可以忽略由于 pN 和 qN 不恰为整数这一事实所引起的小阶次的影响。

a）证明，适当选择 b，$C(N) = \log_b N$ 恰满足这些关系，对于二分和斐波那契查找，b 的这个值同早先导出的相一致。

b）一个人争辩如下："在这个算法中，被扫描的区间长度除以 $1/p$ 的概率是 p；这个区间大小除以 $1/q$ 的概率是 q。因此平均说来该区间除以 $p \cdot (1/p) + q \cdot (1/q) = 2$，所以这个算法和二分查找一样好，而不论 p 和 q 是什么。"他的论证有错误吗？

21. 〔20〕 画出对应于 $N = 10$ 时内插查找的二叉树。

22. 〔$M43$〕（姚期智和姚储枫）证明：只要适当地定义内插查找，则把它应用于已经排好序的 N 个独立的均匀随机键时，它在渐近意义下只要求平均 $\log_2\log_2 N$ 次比较。而

且，对于这样的表的所有查找算法平均必须花费近似$\log_2\log_2 N$次比较。

▶23.〔25〕 在本节末尾提出的H·博顿布鲁奇的二分查找方法，算法末尾之前避免作相等判断（在该算法运行期间，我们知道$K_l \leqslant K < K_{u+1}$，而且在$l=u$之前，不检查相等的情形）。这样一个技巧将使程序B对大的N运行得更快些，因为"JE"指令可从内循环中撤销。（然而，这一思想事实上并不实际，因为$\log_2 N$通常都很小；为了补偿在一次成功的查找时所需要的额外迭代，要$N>2^{36}$才行！）

证明对应于一个二叉树的每个查找算法，均可修改成如下一个查找算法，这个查找算法在树的内部节点处使用两路分枝（<和≥），代替在正文讨论中所用的三路分枝(<，=，或>)。特别是，说明在这种情况下如何修改算法C。

▶24.〔23〕 完备的二叉树，是一种在连续单元中表示一株极小路径长度树的方便的方式。（参考2.3.4.5节）试设计以这个表示为基础的一个有效查找方法。〔提示：在二分查找中是否有可能用乘以2来代替除以2？）

▶25.〔M25〕 假设一株二叉树对于$k=0,1,\cdots$在k级上有a_k个内部节点和b_k个外部节点（根在0级），于是在图8中我们有$(a_0, a_1, \cdots, a_5)=(1,2,4,4,1,0)$和$(b_0, b_1, \cdots, b_5)=(0,0,0,4,7,2)$。(a) 证明有一个把生成函数$A(z)=\sum_k a_k z^k$和$B(z)=\sum_k b_k z^k$联系起来的简单的代数关系。(b) 一株二叉树的成功查找的概率分布，具有生成函数$g(z)=zA(z)/N$；而不成功的查找的生成函数是$h(z)=B(z)/(N+1)$。（于是在正文的记号下，我们有$C_N=\mathrm{mean}(g)$，$C'_N=\mathrm{mean}(h)$，且等式(2)给出了这些量之间的一个关系式。）试求$\mathrm{var}(g)$和$\mathrm{var}(h)$之间的一个关系式。

26.〔22〕 证明斐波那契树同三条带上的多阶段合并排序有关。

27.〔M30〕〔H. S. 斯通和约翰·林(John Linn)〕 考虑一个查找过程，它同时使用k部处理机，且仅以键的比较为基础。于是在查找的每一步，确定k个下标$i_1, \cdots, i_k$，同时进行k个比较；如果对于某个j，$K=K_{i_j}$，则该查找成功地结束，否则，根据2^k种可能的结果$K<K_{i_j}$或$K>K_{i_j}$，$(1 \leqslant j \leqslant k)$来进行下一步骤。

证明，当$N\to\infty$时，这样一个过程近似地至少平均花费$\log_{k+1}N$步，其中假定这个表的每个键作为一个查找变元是同等可能的。（因此，比起一部处理机的二分查找来，在速度上可能的增加仅是因子$\log_2(k+1)$，而不是我们可能期望的因子k。在这个意义下，赋予每个处理机以不同的独立的查找问题而不是把它们合并成单一的查找，是更有效的。）

6.2.2 二叉树查找

由上一节我们知道，一株隐含的二叉树结构有助于了解二分查找和斐波那契查找的特性。对于任何给定的N值，与二分查找相对应的树，达到了借助键比较来查找一份表所需比较数的理论极小值。但是，上一节的方法主要适用于固定长度的表，这是因为，记录的顺序分配使得插入和删除相当费时。如果这个表动态地变化，则我们花费的维护时间可能比在二分查找中查找它们时节省的时间还多。

使用一个明显的二叉树结构，不但能有效地查找表，而且还能迅速地插入和删除记录。因此，我们实际上就具有一个对于查找和排序两者都有用的方法。这种灵活性的获得，是通过对表的每个记录附加两个链接场来达到的。

查找一个增长着的表的技术，通常称为符号表算法，因为汇编程序和编译程序以及其它系统程序一般都使用这样的方法以记住用户定义过的符号。例如，在一个编译程序内的每个记录的键，可以是某个 FORTRAN 或 ALGOL 程序中表示某变量的符号标识符，而该记录的其余部分可以包含那个变量的类型及存储分配的信息。或者说，键可以是MIXAL程序中的一个符号，而该记录的其余部分包含着那个符号的等价物。本书描述的树查找和插入程序可有效地用作符号表算法，特别适用于一些希望按字母顺序打印出符号表的场合。6.3和6.4节描述了其它符号表算法。

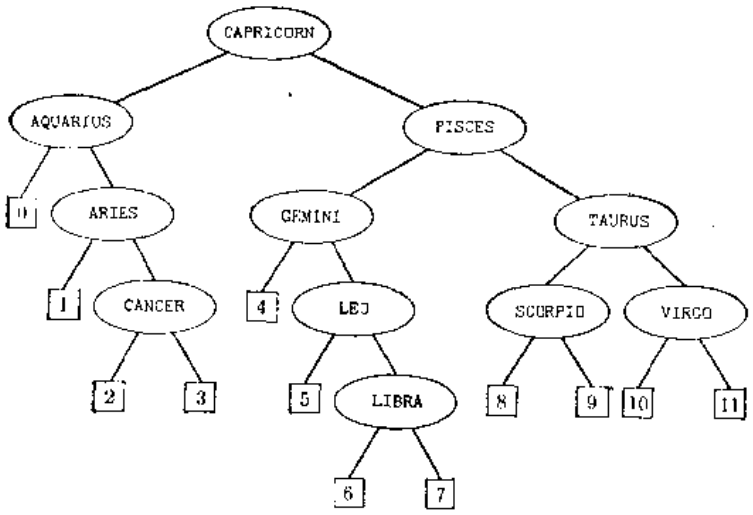


图10　一株二分查找树

图10示出包含黄道十一天宫名称的二分查找树。如果我们现在在根处或树的顶点处开始查找第12个名字人马座（SAGITTARIUS），则发现它大于摩羯座（CAPRICORN），所以我们向右移；它大于双鱼座（PISCES），所以再次右移；它小于金牛座（TAURUS），所以左移；它还小于天蝎座（SCORPIO），所以我们达到外部节点 $\boxed{8}$ 。这个查找是不成功的；现在我们可以把人马座链接到树中以代替外部节点 $\boxed{8}$ ，这就把它插入到了最后的查找位置。这样一来，这个表就能增长而无须移动任何现有的记录。图10就是以一株空树开始，把 CAPRICORN，AQUARIUS，PISCES，ARIES，TAURUS，GEMINI，CANCER，LEO，VIRGO，LIBRA，SCORPIO 诸键依次插入树中而形成的。

图10中根的左子树的所有键，在字符次序上都小于 CAPRICORN，而右边子树中的所有键，在字符次序上都大于它。对于每个节点的左子树和右子树，类似的命题也成立。由此得出，如果我们以对称次序来遍历这株树（参看2.3.1节），那么诸键严格地从左到右组成序列

AQUARIUS，ARIES，CANECR，CAPRICORN，GEMINI，
LEO，…，VIRGO

这是因为，对称次序的原则是：对每个节点，先遍历该节点的左子树，接着是该节点本身，然后遍历右子树。

下列算法详尽地叙述了查找和插入的过程。

算法 T(树查找和插入) 给定一个形成如上所述二叉树的记录表，本算法查找一个给定的变元K。如果K不在表中，则在树的适当位置插入包含K的一个新节点。

假定树的节点至少包含下列诸场：

KEY(P)=存于 NODE(P)中的键；

LLINK(P)=指向 NODE(P)的左子树的指针；

RLINK(P)=指向 NODE(P)的右子树的指针。

空子树（图10中的外部节点）以空指针 Λ 表示，变量 ROOT指向此树的根。为方便起见，我们假定此树不是空的（即，ROOT≠Λ)。因为当 ROOT=Λ 时，应该做什么操作是显然的。

T1.〔初始化〕 置P←ROOT（指针变量P将沿树下移)。

T2.〔比较〕 如果K<KEY(P)，则转到 T3；如果K>KEY(P)则转到 T4；如果K=KEY(P)，则查找成功地结束。

T3.〔左移〕 如果 LLINK(P)≠Λ，则置 P←LLINK(P)并转回 T2，否则转到T5。

T4.〔右移〕 如果 RLINK(P)≠Λ，则置 P←RLINK(P)，并转回 T2。

T5.〔插入树〕（这个查找是不成功的；我们现在将把K置入树中。）置Q⇐AVAIL，即一个新节点的地址。置 KEY(Q)←K，LLINK(Q)←RLINK(Q)←Λ（实际上，新节点的其它场也应予初始化)。如果K小于 KEY(P)，则置 LLINK(P)←Q，否则置RLINK(P)←Q（这时我们可置P←Q，并成功地结束此算法)。

这个算法很便于用机器语言实现。例如，我们可以假定，树节点形为

+	0	LLINK	RLINK
KEY			

(1)

也许后边还跟有 INFO 的一些附加字。利用可用存储空间的一个 AVAIL 表，如同在第二章中那样，我们可以写出下列的 MIX 程序：

程序 T（树查找和插入） rA≡K，rI1≡P rI2≡Q。

```
LLINK   EQU     2:3
RLINK   EQU     4:5
START   LDA     K               1       T1. 初始化
        LD1     ROOT            1       P←ROOT
        JMP     2F              1
4H      LD2     0,1(RLINK)      C2      T4. 右移，Q←RLINK(P)
        J2Z     5F              C2      如果Q=Λ 则转T5
1H      ENT1    0,2             C-1     P←Q
2H      CMPA    1,1             C       T2. 比较
        JG      4B              C       如果K>KEY(P) 则转到T4
        JE      SUCCESS         C1      如果K=KEY(P) 则转出
        LD2     0,1(LLINK)      C1-S    T3. 左移，Q←LLINK(P)
```

13		J2NZ	1B	$C1-S$	如果Q≠A则转到T2
14	5H	LD2	AVAIL	$1-S$	T5. 插入树中
15		J2Z	OVERFLOW	$1-S$	
16		LDX	0,2(RLINK)	$1-S$	
17		STX	AVAIL	$1-S$	Q←AVAIL
18		STA	1,2	$1-S$	KEY(Q)←K
19		STZ	0,2	$1-S$	LLINK(Q)←RLINK(Q)←Λ
20		JL	1F	$1-S$	是K<KEY(P)吗?
21		ST2	0,1(RLINK)	A	RLINK(P)←Q
22		JMP	*+2	A	
23	1H	ST2	0,1(LLINK)	$1-S-A$	LLINK(P)←Q
24	DONE	EQU	*	$1-S$	插入后转出

这个程序的前13行进行查找；后11行进行插入。查找阶段的运行时间是 $(7C+C1-3S+4)u$，其中

C =所作比较数；

$C1=K\leqslant$KEY(P)的次数；

$S=1$ 如果查找是成功的，否则为0。

平均说来，我们有 $C1=\frac{1}{2}(C+S)$，因为 $C1+C2=C$，且 $C1-S\approx C2$；所以运行时间大约是 $(7.5C-2.5S+4)u$，这优于使用一株隐含树的二分查找算法（参考程序6.2.1C）。通过象在程序6.2.1F中那样复写代码，我们可以消去程序T的行08，使运行时间减少到 $(6.5C-2.5S+5)u$。如果查找是不成功的，则程序的插入阶段额外耗费时间14 u 或15 u。

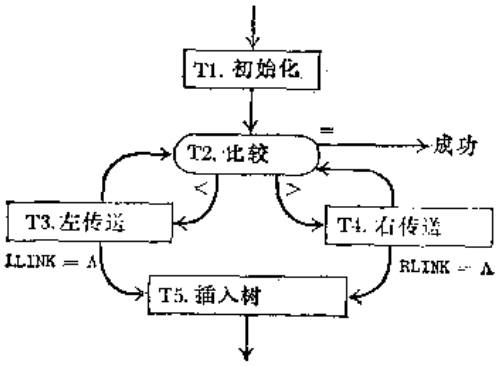


图11 树查找和插入

算法T很容易修改为适用于可变长的键和可变长的记录。例如，如果我们以后进先出方式顺序地分配可利用的空间，则不难建立大小可变的节点：（1）中的头一个字可指出大小。由于以树为基础的符号表算法能有效地使用存储，在编译程序、汇编程序以及装入程序中使用这种方法是特别有吸引力的。

但最坏情况如何呢？ 当程序员们第一次看到算法T时，通常是怀疑它的。如果图10的键是以字符顺序 AQUARIUS, …, VIRGO 而不是以历法顺序 CAPRICORN, …, SCORPIO 记入树中的，则算法将构造出一个实质上确定一个顺序查找的蜕化树（所有LLINKS将是空的）。类似地，如果键以古怪的次序

AQUARIUS, VIRGO, ARIES, TAURUS, CANCER, SCORPIO, CAPRICORN, PISCES, GEMINI, LIBRA, LEO

出现，则我们得到同样坏的“Z”字形弯弯曲曲的树。（请试一下！）

另一方面，对图10中的树来说，在一次成功的查找中，平均仅需要 $3\frac{2}{11}$ 次比较；这

仅仅比最好的二叉树中所能达到的极小平均比较数3稍大一点。

当我们有了一株相当平衡的树时，查找时间大体上与$\log_2 N$成正比。但当我们有一株蜕化树时，查找时间大体却与N成正比。习题2.3.4.5-5证明，如果把每株N节点的二叉树都当作是同等可能的，则平均查找时间将大约同$\sqrt{N}$成正比。对于算法T我们实际上能预期什么特性呢？

幸而，可以证明，如果诸键是以随机次序插入树中的，则树查找将仅需要大约$2\ln N \approx 1.386\log_2 N$次比较。平衡树是普遍的，而蜕化树则是非常稀少的。

关于这一事实，出人意外，有一个简单的证明。我们假定，在N个键的$N!$种可能的次序中，每一种都有同等可能用作插入序列以构造一株树。找出一个键所需要的比较次数，恰比把此键记入树所需要的比较次数多1。因此，如果C_N是一次成功的查找中的平均比较次数，且C'_N是在一次不成功的查找中的平均比较次数，则我们有

$$C_N = 1 + \frac{C'_0 + C'_1 + \cdots + C'_{N-1}}{N} \tag{2}$$

但是，在内部和外部路径长度之间的关系式告诉我们

$$C_N = \left(1 + \frac{1}{N}\right)C'_N - 1 \tag{3}$$

这就是等式6.2.1-2。把它同(2)合在一起得到

$$(N+1)C'_N = 2N + C'_0 + C'_1 + \cdots + C'_{N-1} \tag{4}$$

这个递归式容易解出。从式(4)减去

$$NC'_{N-1} = 2(N-1) + C'_0 + C'_1 + \cdots + C'_{N-2}$$

得到

$$(N+1)C'_N - NC'_{N-1} = 2 + C'_{N-1}$$

$$C'_N - C'_{N-1} + 2/(N+1)$$

由于$C'_0 = 0$，这意味着

$$C'_N = 2H_{N+1} - 2 \tag{5}$$

应用式(3)并进行简化即得到所希望的结果

$$C_N = 2\left(1 + \frac{1}{N}\right)H_N - 3 \tag{6}$$

下面的习题6-8给出更详细的信息；有可能计算C_N和C'_N精确的概率分布，而不仅仅是平均值。

树插入排序 算法T是为了进行查找而设计的，但它也可用作内部排序算法的基础；事实上，我们可以把它看作表插入算法5.2.1L的一种自然推广。如果正确地编制程序，则它的运行时间将只比我们在第5章中所讨论的某些最好的算法略慢一点。在把所有的键构造成一株树之后，一个对称的树遍历过程（算法2.3.1T）将按先后次序访问这些记录。

然而，需要预先提醒一下。注意，如果在步骤T2中K=KEY(P)，则需要完成一些不同的事情，因为我们正在排序而不是在进行查找。一个解决方法是把K=KEY(P)当成同K>KEY(P)完全一样，这就得到了一个稳定的排序方法（但要注意，相等的键在

树中未必是相邻的，它们只不过是在对称的次序下相邻而已）。如果出现许多重复的键，则这个方法会使树变得极不平衡，而且程序将减慢。另一种想法是对每一个节点，都保持一张表，表上记着具有相同键的所有记录；这虽需要另一个链接场，但当出现大量相同的键时能更快地排序。

于是，如果我们仅对排序感兴趣，而对查找不感兴趣，则算法T是不坏的；但对排序还有更好的方式。另一方面，如果有把查找和排序组合起来的应用，则我们可热情地推荐树方法。

说明这一点是有趣的，即，在树插入排序的分析与分划交换（"快速排序"）的分析之间有着密切的关系，尽管这些方法表面上却毫无类似之处。如果我们把N个键逐次地插入到开始为空的一株树中，则除了少数例外，我们所作的键之间的平均比较次数同算法5.2.2Q所作的相同。例如，在树插入中每个键同K_1进行比较，而后，小于K_1的每个键同小于K_1的第一个键进行比较，等等；在快速排序中，每一个键同头一个分划元素K进行比较，而后，小于K的每个键须同小于K的一个特殊元素进行比较，等等。在这两种情况下所作的平均比较数都是NC_N（然而，算法5.2.2Q实际上稍微多做了一点比较，为的是加速内循环）。

删去 有时我们要使计算机忘记它所知道的表中的一项。删去一个"叶"节点（一个其两个子树皆为空的节点），或删去其中LLINK＝Λ或RLINK＝Λ的一个节点，是容易的；但当LLINK和RLINK都是非空指针时，就要做些特殊处理，因为我们不能一次指向两条路。

例如，再次考虑图10；我们如何删去摩羯座（CAPRICORN）呢？一个解决方法是删去下一个节点，它总有一个空的LLINK，然后重新把它插入到我们真正要删去的节点的位置上。例如，在图10中，我们可以删去双子座（GEMINI），然后以双子座(GEMINI)来代替摩羯座（CAPRICORN）。这个操作保持了表中各项必要的自左到右的次序。下列算法给出了做这一操作的一般途径的详细描述。

算法D(树删去) 设Q是一个变量，它指向如算法T中所表示的二分查找树的一个节点。本算法删去该节点，保留一株二分查找树（实际上，在树的某个节点中，我们将有Q＝ROOT或Q＝LLINK(P)或Q＝RLINK(P)。这个算法在存储中重置Q的值，以反映删去）。

D1.〔RLINK为空吗？〕 置T←Q。如果RLINK(T)＝Λ，则置Q←LLINK(T)并转到$D4$。（例如，如果对于某个P，Q＝RLINK(P)，则这意味着我们将置RLINK(P)←LLINK(T)，等等）。

D2.〔找后继者〕 置R←RLINK(T)，如果LLINK(R)＝Λ，则置LLINK(R)←LLINK(T)，Q←R，并转到$D4$。

D3.〔找空的LLINK〕 置S←LLINK(R)，然后，如果LLINK(S)≠Λ，则置R←S并重复这一步骤直到LLINK(S)＝Λ为止（这时，S将等于Q＄，即Q的对称的后继者）。最后，置

LLINK(S)←LLINK(T)，LLINK(R)←RLINK(s)，RLINK(S)←RLINK(T)Q←S。

D4.〔释放此节点〕 置AVAIL⇐T（即把删去的节点送回到可用存储空间中）。

希望读者通过删去图10中的宝瓶座（AQUARIUS）、巨蟹座（CANCER）和摩羯座（CAPRICORN）来试验这个算法；每种情况是稍微不同的。一个机灵的读者可能已经注意到，对于RLINK(T)≠Λ，LLINK(T)=Λ的情况没有进行任何特殊的判断；我们将把这一情况的讨论推迟到稍后进行，因为这一算法有着某些非常有趣的性质。

由于算法D左右两边很不对称，有理由认为一连串的随机删去和插入将使树失去平衡，从而使我们已经作出的有效性估计成为不正确的。但是事实上，删去绝不会使树蜕化！

定理H（T.N.希巴德，1962） 通过算法D从一株随机树删去一个随机元素之后，得到的树仍是随机的。

〔不是学数学的读者，请跳到(10)。〕 当然，这个定理的叙述是非常含混的。我们可以更精确地概述这一情况如下：设$\mathcal{T}$是n个元素的一株树，且$P(\mathcal{T})$是当由算法T以随机次序插入$\mathcal{T}$的键时，$\mathcal{T}$出现的概率。有些树的出现概率可能大些，而另一些树的概率可能小些。命$Q(\mathcal{T})$是通过算法T以随机次序插入$n+1$个元素，而后通过算法D随机地选择其中一个元素并把它删去时，$\mathcal{T}$将出现的概率。在计算$P(\mathcal{T})$时，我们假定键的$n!$种排列是同等可能的；在计算$Q(\mathcal{T})$时，假定原来的n个键和准备删去的键的$(n+1)\cdot(n+1)!$种排列是同等可能的。此定理指出，对所有$\mathcal{T}$，$P(\mathcal{T})=Q(\mathcal{T})$。

证明 我们面对着这样一个事实，即诸排列是同等可能的，而诸树不一定，因此我们将通过把排列当作随机对象来证明这个结果。下面，给出从排列中删去的定义，而后再证明“从一个随机排列中删去一个随机元素后仍保留一个随机排列”。

设$a_1a_2\cdots a_{n+1}$是$\{1,2,\cdots,n+1\}$的一个排列；我们要定义删去a_i的操作，以便得到$\{1,2,\cdots,n\}$的一个排列$b_1b_2\cdots b_n$。这个操作应该对应于算法T和D，使得如果我们首先通过插入序列a_1，a_2，…，a_{n+1}构造一株树，然后删去a_i，并对剩下的键按1到n的次序重新编号时，便得到由$b_1b_2\cdots b_n$构造的树。

幸而，不难定义这样一个删去操作。有两种情况：

情况1：$a_i=n+1$，或对某个$j<i a_i+1=a_j$（这实质上是条件“RLINK$(a_i)=\Lambda$”）。从这个序列撤销a_i，并把每个大于a_i的元素减1。

情况2：对某个$j>i a_i+1=a_j$。以a_j代替a_i，从a_j原来的位置撤销a_j，并把每个大于a_i的元素减1。

例如：假设我们有排列4 6 1 3 5 2。如果把有待删去的元素画上圆圈，则有

④6 1 3 5 2＝4 5 1 3 2	4 6 1③5 2＝3 5 1 4 2
4⑥1 3 5 2＝4 1 3 5 2	4 6 1 3⑤2＝4 5 1 3 2
4 6①3 5 2＝3 5 1 2 4	4 6 1 3 5②＝3 5 1 2 4

因为有$(n+1)\cdot(n+1)!$个可能的删去操作，所以如果能够证明$\{1,2,\cdots,n\}$的每一排列恰是$(n+1)^2$次删去的结果，则定理得证。

设$b_1b_2\cdots b_n$是$\{1,2,\cdots,n\}$的一个排列。我们将定义$(n+1)^2$次删去，其中满足$1\leqslant i$，$j\leqslant n+1$的每一对偶i，j对应一次删去，如下：

如果$i<j$，则删去是

$$b'_1 \ldots b'_{i-1} \textcircled{b_i} b'_{i+1} \ldots b'_{j-1} (b_i + 1) b'_j \ldots b'_n \tag{7}$$

由这里起，b'_k 表示 b_k 或 $b_k + 1$，依赖于 b_k 是否小于划了圈的元素。这个删去对应于情况 2。

如果 $i > j$，则删去是

$$b'_1 \ldots b'_{i-1} \textcircled{b_j} b'_i \ldots b'_n; \tag{8}$$

这个删去满足情况 1 的定义。

最后，如果 $i = j$，则我们又有情况 1 的删去，即

$$b'_1 \ldots b'_{i-1} \textcircled{n+1} b'_i \ldots b'_n. \tag{9}$$

例如，设 $n = 4$ 并考虑映射为 3 1 4 2 的25次删去

	$i=1$	$i=2$	$i=3$	$i=4$	$i=5$
$j=1$	⑤3 1 4 2	4③1 5 2	4 1③5 2	4 1 5③2	4 1 5 2③
$j=2$	③4 1 5 2	3⑤1 4 2	4 2①5 3	4 2 5①3	4 2 5 3①
$j=3$	③1 4 5 2	4①2 5 3	3 1⑤4 2	3 1 5④2	3 1 5 2④
$j=4$	③1 5 4 2	4①5 2 3	3 1④5 2	3 1 4⑤2	4 1 5 3②
$j=5$	③1 5 2 4	4①5 3 2	3 1④2 5	4 1 5②3	3 1 4 2⑤

划圈的元素总在位置 i 处，且对于固定的 i 我们已经构造了 $n+1$ 个不同的删去，每个 j 对应 $n+1$ 个删去；因此对每个排列 $b_1 b_2 \cdots b_n$ 已经构造了 $(n+1)^2$ 个不同的删去。因为仅仅可能有 $(n+1)^2 n!$ 个删去，故我们必然已经找到了它们的全部。

定理H的证明不仅告诉我们删去的结果，而且也有助于我们分析一次删除中的平均运行时间。习题12证明，当从一个随机表中删去一个随机元素时，平均说来，我们可期望执行步骤D2的次数稍少于这时间的一半。

现在考虑步骤D3中循环执行的频繁程度：假设我们正在删去 l 级上的一个节点，而且在对称次序下直接跟随的外部节点在 k 级上。例如，如果我们从图10删去摩羯座（CAPRICORN），则有 $l = 0$ 和 $k = 3$，因为节点⑭在级 3 上。如果 $k = l + 1$，则在步骤D1中有 RLINK$(T) = \Lambda$；而如果 $k > l + 1$，则我们将在步骤 D3 中置 S←LLINK(R) 恰好 $k - l - 2$ 次，l 的平均值是（内部路径长度）$/N$；k 的平均值是（外部路径长度－到最左外部节点的距离）$/N$。到最左外部节点的距离是在插入序列中自左至右极小值的个数，所以，按1.2.10节的分析，该距离的平均值为 H_N。由于外部路径长度减去内部路径长度是 $2N$，故 $k - l - 2$ 的平均值是 $-H_N/N$。这个值加上 $k - l - 2$ 为 -1 的平均次数●，我们看出，在一个随机删去中，步骤D3中的操作S←LLINK(R)平均仅被实施

$$\frac{1}{2} + \frac{\frac{1}{2} - H_N}{N} \tag{10}$$

● 即 $k - l - 2$ 的值有多少次为 -1。——译注

次。这使人放心了，因为最坏的情况可能是相当慢的（见习题11）。

尽管在我们已经叙述过的精确形式下，定理H是严格地正确的，但它不能如我们可能预期的那样，应用到后边跟有插入的删除序列上。在删除之后树的形状是随机的，但在一个给定树形中值的相对分布可以变化，结果是，在删除之后的头一次随机插入实际上破坏了这些形状的随机性。这个山加里·克诺特在1972年首先发现的令人吃惊的事实，必须看成是可信的（参考习题15）。经验的证据强烈地提示，在重复的删去和插入之后，路径长度趋于减少，所以偏离随机性似乎是正确的；探讨这种行为的理论仍然缺乏。

如上所述，尽管当LLINK(T)=A时删除变得很容易，但算法D对此不作检验。我们可以在D1和D2之间增加一个新步骤，即

D $1\frac{1}{2}$．〔LLINK是空的吗？〕 如果LLINK=A，则置Q←RLINK(T)并转到D4。

习题14说明，带有这额外步骤的算法D，在路径长度的意义下，使树至少保持象在原来的算法D中那样好，而且有时结果甚至会更好。于是，利用这修改过的算法D的一个插入和删除序列，将得到比随机树理论所预期的实际上更好的树：随时间的推移，查找和插入的平均计算时间趋向于减少。

访问的频率 至今我们已经假定，每个键具有同等可能作为一个查找变元。在一种更为一般的情况下，设P_k是要查找第K个插入元素的概率，其中$P_1+\cdots+P_N=1$，则等式(2)的一个直截了当的修改——如果我们保留随机次序的假定使树的形状保持随机的话——表明，在一次成功的查找中平均比较次数将是

$$1+\sum_{1\leqslant k\leqslant N}P_k(2H_K-2)=2\sum_{1\leqslant k\leqslant N}P_kH_k-1 \tag{11}$$

（参考等式(5)）

例如，如果概率遵守吉甫定律等式6.1-8，并按重要性递减的次序插入键时，则平均比较次数减少为

$$H_N-1+H_N^{(2)}/H_N \tag{12}$$

（见习题18）。这大约等于按相等频率分析所预期的比较数的一半，而且它比我们使用二分查找做的比较数要少。

例如，图12示出当把31个最常用的英文字按频率递减的次序记入时所得到的树。对每个字亦给出其相对频率〔参考Cryptanalysis by H. F. Gaines (New York: Dover, 1956), P226〕。在这株树中，每次成功查找的平均比较次数是4.042；而利用算法6.2.1 B或C的对应的二分查找，将需要4.393次比较。

最优二分查找树 上述考虑自然地提出了这样的问题，即如要求以给定的频率查找键表，则最好的树是什么？例如，图13中画出了31个最常用的英文字的最优树；对于一次成功的查找，它平均只需要3.437次比较。

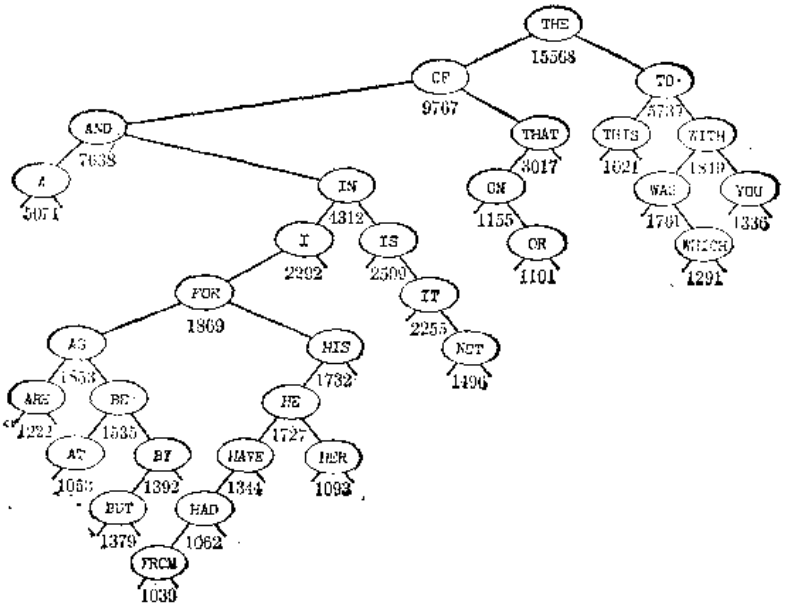


图12　31个最常用的英文字，以频率减少的次序插入

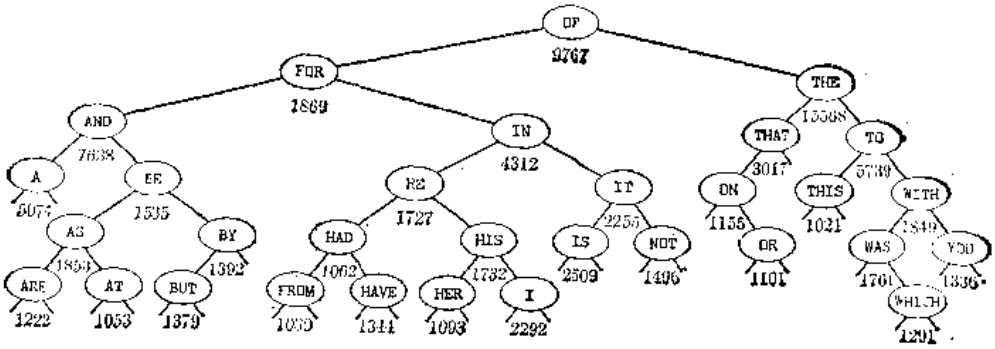
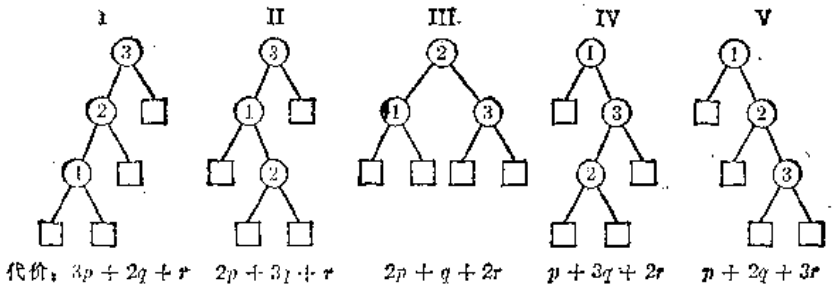


图13　31个最常用英文字的最优查找树

现在，我们来剖析最优树的问题。例如，当$N=3$时，假定键$K_1<K_2<K_3$的概率分别为p，q，r，则有五种可能的树形：



(13)

图14标出了使每株树是最优的p，q，r的范围；如果我们随机地选择p，q，r，则平衡的树大约有45%的可能是最好的（见习题21）。

可惜，当N很大时，有

$$\binom{2N}{N}\Big/(N+1)\approx 4^N/(\sqrt{\pi}N^{3/2})$$

株二叉树，所以我们不可能全部试验它们并看出哪一株是最好的。因此我们来更仔细地研究一下最优二分查找树的性质，以便发现找出它们的一个更好的方法。

至今，我们只研究了一次成功的查找的概率，实际上，通常还必须考虑不成功的情况。例如，图13中的31个字仅考虑了典型的英文课本中大约36%的情况；其余64%的情况肯定将影响最优查找树的结构。

因此，让我们以如下的方式来提出问题：给定 $2n+1$ 个概率 $p_1, p_2, \cdots, p_n$ 和 $q_0, q_1, \cdots, q_n$，其中

p_i＝查找变元是 K_i 的概率；

q_i＝查找变元处于 K_i 和 K_{i+1} 之间的概率

（由约定，q_0 是查找变元小于 K_1 的概率，q_n 是查找变元大于 K_n 的概率。）于是，$p_1+p_2+\cdots+p_n+q_0+q_1+\cdots+q_n=1$，我们要寻找一株二叉树，它使查找中预期的比较次数

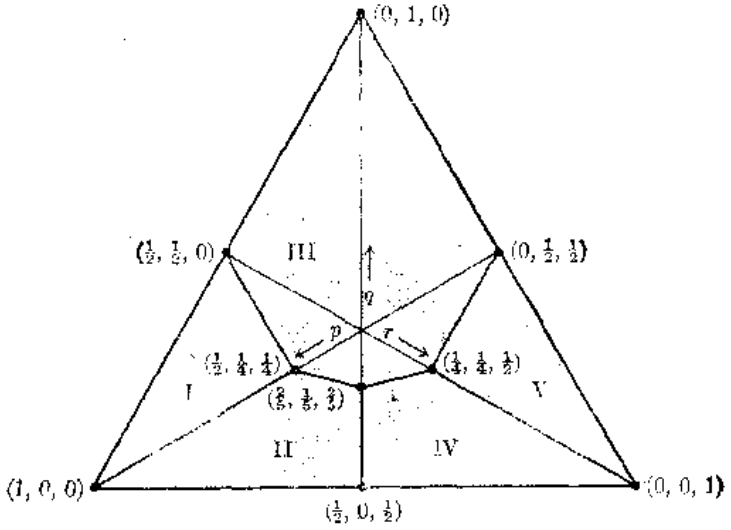
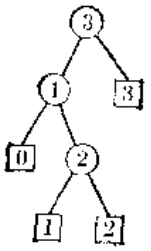


图14　如果（K_1, K_2, K_3）的相对频率是（p，q，r），则此图指出（13）的五株树中哪一株是最好的。虽然有三个坐标，但 $p+q+r=1$ 使这个图形变成二维的

$$\sum_{1\leqslant j\leqslant n} p_j(\text{级}(\textcircled{J})+1)+\sum_{0\leqslant k\leqslant n} q_K\,\text{级}(\boxed{K}) \tag{14}$$

极小化。其中，$\textcircled{J}$是在对称次序下的第 j 个内部节点，$\boxed{K}$是第$(K+1)$个外部节点，根的级为0。于是，对于二叉树



(15)

预期的比较次数是 $2q_0+2p_1+3q_1+3p_2+3q_2+p_3+q_3$。我们称此为树的代价；具有极小代价的树称为是最优的。在这个定义之下，无须要求诸 p 和 q 的和为1，对任何给定的"权"序列（$p_1, \cdots, p_n$；$q_0, \cdots, q_n$），我们可寻求相应的极小代价的树。

我们已经在2.3.4.5节中研究了构造具有极小加权路径长度树的霍夫曼过程；但该方法要求所有的 p 皆为0，且在它产生的树上，其外部节点的权（$q_0, \cdots, q_n$）通常不是严格地按从左到右的对称次序排列的。因此，我们需要另外的方法。

有一个原理能帮助我们，这就是：一株最优树的所有子树都是最优的。例如，如果(15)对于权（p_1，p_2，p_3；q_0，q_1，q_2，q_3）是一株最优树，则这个根的左子树必然是对于（p_1p_2；$q_0q_1q_2$）为最优的；对一株子树的任何改进必然导致对整株树的改进。

这一原理提示了一个计算过程，它系统地寻找越来越大的最优子树。我们在5.4.9节已经使用同样的思想来构造最优的合并形式；一般的方法称作“动态规划”，我们将在第7章作进一步的考虑。

设$c(i,j)$是具有权$(p_{i+1},\cdots,p_j;\ q_i,\cdots,q_j)$的一株最优子树的代价；且$W(i,j)=p_{i+1}+\cdots+p_j+q_i+\cdots+q_j$是所有那些权的和，$c(i,j)$和$W(i,j)$定义于$0\leqslant i\leqslant j\leqslant n$上。由此得出

$$c(i,i)=0$$

$$c(i,j)=W(i,j)+\min_{i\leqslant k<j}(c(i,k-1)+c(k,j)),\ 对于\ i<j \tag{16}$$

因为具有根ⓚ的一株树可能达到的极小代价是$W(i,j)+c(i,k-1)+c(k,j)$。当$i<j$时，设$R(i,j)$是使(16)中的极小值能达到的所有k的集合；这个集合确定了诸最优树的可能的根。

等式(16)使我们能对$j-i=1,2,3,\cdots,n$计算$c(i,j)$；大约有$\frac{1}{2}n^2$个这样的值，且对大约$\frac{1}{6}n^3$个k值实现了极小化操作。这意味着我们能利用$O(n^2)$个存储单元，在$O(n^3)$个时间单位中确定一株最优树。

实际上，如果我们利用一个“单调性”的性质，则即可从运行时间中消去一个因子n。设$r(i,j)$表示$R(i,j)$的一个元素；我们不必计算整个集合$R(i,j)$，只计算一个代表元素就够了。一旦找到$r(i,j-1)$和$r(i+1,j)$，则习题27的结果证明，当权非负时，我们总可以假定

$$r(i,j-1)\leqslant r(i,j)\leqslant r(i+1,j) \tag{17}$$

由于(16)中只需考察$r(i+1,j)-r(i,j-1)+1$个而不是$j-i$个k值，这就限制了极小值的查找范围。当$j-i=d$时，总工作量便以缩短的级数

$$\sum_{\substack{d\leqslant j\leqslant n\\ i=j-d}}(r(i+1,j)-r(i,j-1)+1)$$

$$=r(n-d+1,n)-r(0,d-1)+n-d+1<2n$$

为界，因此总的运行时间已减少到$O(n^2)$。

下面的算法详细地描述了这一过程。

算法K（求最优的二分查找树）给定$2n+1$个非负的权$(p_1,\cdots,p_n;\ q_0,\cdots,q_n)$。本算法构造二叉树$t(i,j)$，它在上述定义的意义下，对于权$(p_{i+1},\cdots,p_j;\ q_i,\cdots,q_j)$有极小的代价。要计算三个数组，即

$c[i,j]$，对于$0\leqslant i\leqslant j\leqslant n$，$t(i,j)$的代价

$r[i,j]$，对于$0\leqslant i\leqslant j\leqslant n$，$t(i,j)$的根

$w[i,j]$，对于$0\leqslant i\leqslant j\leqslant n$，$t(i,j)$的总权

算法的结果由数组r指明：如果$i=j$，则$t(i,j)$为空；否则它的左子树是$t(i,r[i,j]-1)$，它的右子树是$t(r[i,j],j)$。

K1.〔初始化〕 对于$0\leqslant i\leqslant n$置$c[i,i]\leftarrow 0$，和$w[i,i]\leftarrow q_i$，对于$j=i+1,\cdots,n$置$w[i,j]\leftarrow w[i,j-1]+p_j+q_j$。然后，对于$1\leqslant j\leqslant n$置$c[j-1,$

$j\rceil \leftarrow w\lceil j-1, j\rceil$以及$r\lceil j-1, j\rceil \leftarrow j$(这确定所有一个节点的最优树)。

K2. 〔对d循环〕 对$d=2, 3, \cdots, n$执行步骤K3，然后结束该算法。

K3. 〔对j循环〕 (我们已经确定了少于d个节点的最优树，这个步骤确定所有d个节点的最优树。) 对于$j=d, d+1, \cdots, n$进行步骤$K4$。

K4. 〔求$c\lceil i, j\rceil$, $r\lceil i, j\rceil$〕 置$i \leftarrow j-d$，然后置$c\lceil i, j\rceil \leftarrow w\lceil i, j\rceil + \min_{r\lceil i, j-1\rceil \leqslant k \leqslant r\lceil i+1, j\rceil}(c\lceil i, k-1\rceil + c\lceil k, j\rceil)$。并且置$r\lceil i, j\rceil$为使极小值出现的一个$k$值(习题22证明，$r\lceil i, j-1\rceil \leqslant r\lceil i+1, j\rceil$) ▌

作为算法K的一个例子，考虑图15，它是以“上下文中的键”(KWIC)索引应用为基础的。ACM杂志头十卷中所有论文的标题，被排序并编制成一个重要的词汇索引，其中每个标题的每个字都有一行。然而有些字，如“THE”和“EQUATION”被认为并不包含多少信息，故从索引中省去。在图15的内部节点上标出了这些特殊字和它们出现的频率。注意，象“对于某个新问题的一个方程的解”这样的标题，由于完全不提供任何信息，所以根本不出现在索引中！KWIC索引的思想，来源于H. P. Luhn, Amer. Documentation 11(1960), 288-295(见W. W. Youden, JACM 101 1963), 583-646，其中有完全的KWIC索引)。

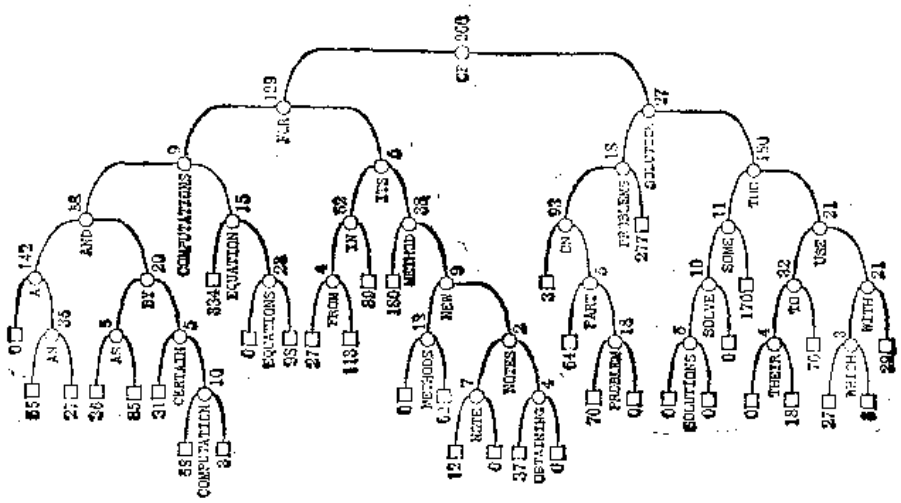


图15 用于一个KWIC索引的一株最优二分查找树

当为排序准备一个KWIC索引文件时，我们可能希望用一株二分查找树，以便检验每个特定的字是否应被编入索引。其它的字落在两个未编索引的字之间，其频率在图15的外部节点上标出；例如，在1954-1963年期间出现在JACM的标题中的字母处于“PROBLEMS”和“SOLUTION”之间的恰有277个字。

图15示出了$n=35$时，由算法K得到的最优树。对于$j=1, 2, \cdots, 35$, $r\lceil 0, j\rceil$的计算值是(1, 1, 2, 3, 3, 3, 3, 8, 8, 8, 8, 8, 8, 11, 11, $\cdots$, 11, 21, 21, 21, 21, 21, 21)；对于$i=0, 1, 2, \cdots, 34$, $r\lceil i, 35\rceil$的值是(21, 21, $\cdots$, 21, 25, 25, 25, 25, 25, 25, 26, 26, 26, 30, 30, 30, 30, 30, 30, 30, 33, 33, 33, 35, 35)。

“中间性频率”q_j，对于最优树结构有一个值得注意的影响。图16(a)示出了置q_j为

0时所得到的最优树。类似地，内部频率 p_i 是重要的：图16(b)示出了当 p_i 被置成0时的最优树。考虑全部频率的集合，平均说来，图15的树仅需要4.75个比较，而图16的树则分别要求5.29和5.32个比较（在这个例子中，一个直接的二分查找将比图16的树更好）。

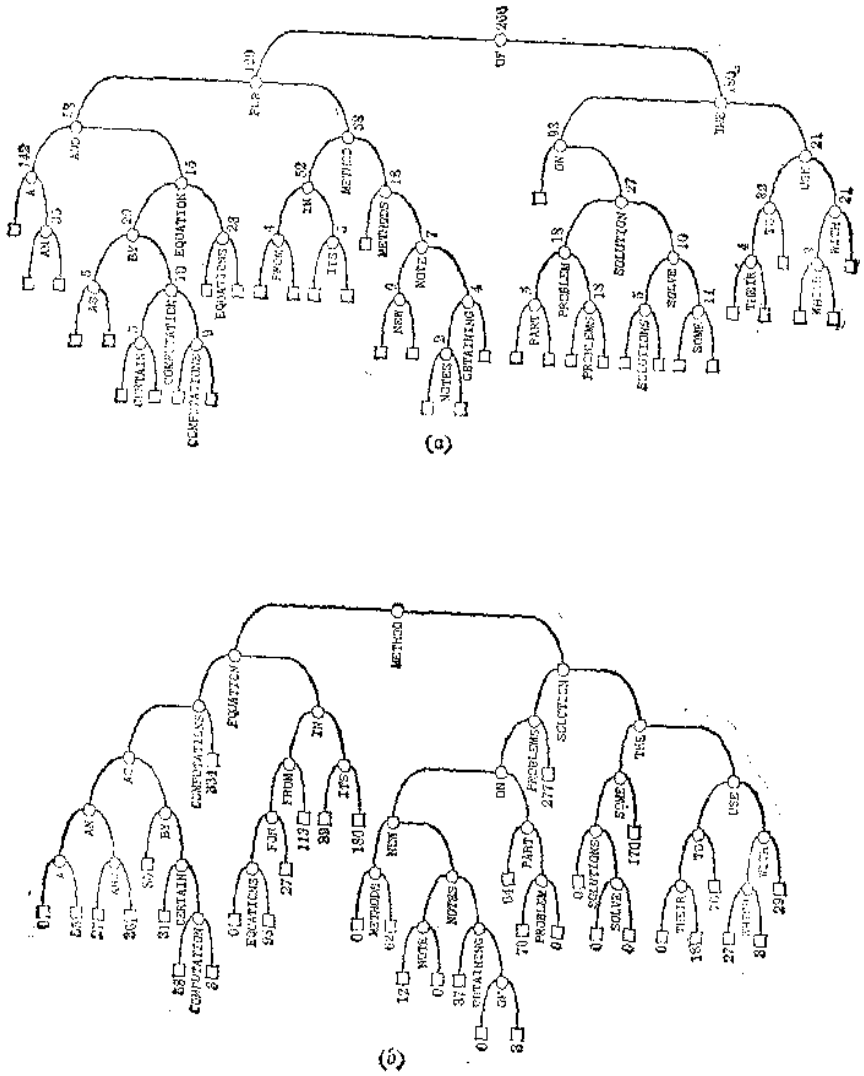


图16　在图15的一半数据基础上的最优二分查找树
（a）外部频率为0；（b）内部频率为0。

由于算法K要求时间和空间同n^2成正比，因此当n变得很大时，使用它就成为不实际的了。当然，鉴于在这一章稍后有待讨论的其它查找技术，我们对于很大的n可能不真正使用二叉树；但仍然假定，要在n很大时找一棵最优的或接近于最优的二叉树。

我们已经看到，以递降的频率顺序插入键的思想，平均说来可以产生相当好的树；但由于不利用q_i个权，它也可能非常之坏（见习题20），而且它不是经常地接近于最优。另一个方法是选择根k，使得到的极大子树权$\max(w(0, k-1), w(k, n))$尽可能小。这个方法也许是相当差的，因为可能选择一个具有非常小的p_k的节点作为根；然而，P. J. 拜尔（Paul J. Bayer）已经证明，得到的树总有一个接近于最优加权路径的长度（见习题36）。

正如W. A. 沃克（W. A. Walker）和C. C. 戈特利布（C. C. Gotlieb）〔Graph Theory and Computing (Academic Press, 1972)〕所提议的，把这两个方法组合起来，可以得到一个更令人满意的过程：设法使左边和右边的权相等，但准备把根向左或向右移动几步，以找出具有相当大的p_k的一个节点。图17指出为什么这个方法是有道理的：如果对图15的KWIC数据，把$c(0, k-1)+c(k, n)$画作k的一个函数，则我们看到结果对于p_k的数量级是十分敏感的。

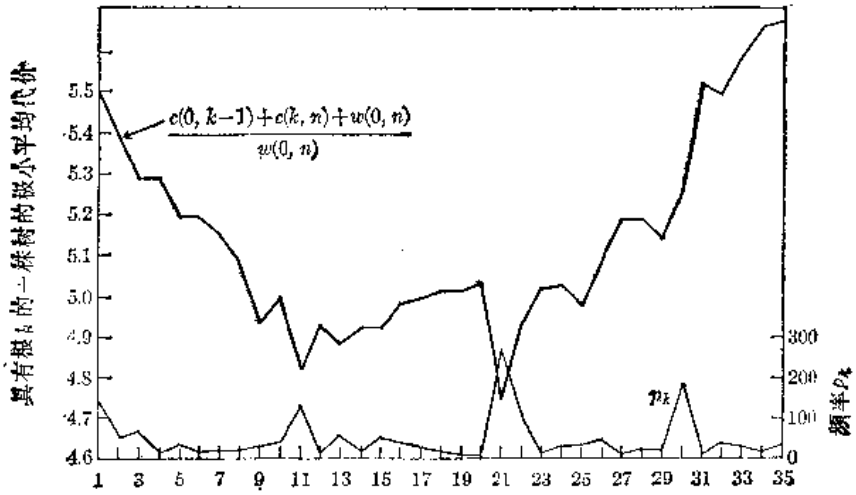


图17　作为根k的一个函数的代价特性

当n很大时，这样一个"由顶向下"的方法，可用来选择根，然后再处理左子树和右子树。当我们达到一株充分小的子树时，就可以应用算法K。这样的方法产生相当好的树（据报告与最优的树相差不到百分之二或三），并且它只需要$O(n)$个空间单位和$O(n \log_2 n)$个时间单位。事实上，M. 弗雷德曼（M. Fredman）已经证明，如果使用正确的数据结构，$O(n)$个时间单位就足够〔ACM. Symp. Theory of Comp. 7(1975), 240-244〕。

***胡-塔克算法**　在所有的p都为0的情况下，胡德强和A. C. 塔克（A. C. Tucker）已经发现了一个卓越的"由底向上"的方法来构造最优树；如果使用适当的数据结构，则他们的方法需要$O(n)$个空间单位和$O(n \log_2 n)$个时间单位，并且构造出一株实际上最优的树（而不仅仅是近似的）。

加西亚-瓦克斯（Garsia-Wachs）的算法刊登在SIAM J. Computing, Dec. 1977, PP. 622 ff上；但是现在胡德强、克莱特曼（Kleitman）和玉木（Tamaki）找到了一个显然更好的方法。他们的文章发表在SIAM J. Appl. Math. 37(1979)，246-256。

胡-塔克算法可描述如下：

·阶段1　组合　以写在外部节点内的权的“工作序列”

$$\boxed{q_0} \quad \boxed{q_1} \quad \boxed{q_2} \quad \cdots \quad \boxed{q_n} \tag{18}$$

开始。然后重复地把对于 $i<j$ 的两个权 q_i 和 q_j 组合成一个权 q_i+q_j，从工作序列删去包含 q_j 的节点，并以内部节点

$$\textcircled{q_i+q_j} \tag{19}$$

代替包含 q_i 的节点。每次被组合的对偶（q_i，q_j）是唯一的，它满足下列条件：

i）在 q_i 和 q_j 之间无外部节点出现（这是最重要的规则，它把本算法同霍夫曼方法区别开来）。

ii）和数 q_i+q_j 对满足规则（i）的所有（q_i，q_j）为极小。

iii）下标 i 对满足规则（i）、（ii）的所有（q_i，q_j）为极小。

iv）下标 j 对满足规则（i）、（ii）、（iii）的所有（q_i，q_j）为极小。

·阶段2　级的指定　当阶段1结束时，在工作序列中留下一个节点，以级数0来标志它。然后以颠倒的次序反做阶段1的步骤，标记相应树的级号；如果（19）的级为 l，则含有 q_i 和 q_j 的诸节点（q_i 和 q_j 是形成节点（19）的）以级号 $l+1$ 标记之。

·阶段3　重新组合　现在我们有了外部节点和级号的工作序列

$$\underset{l_1}{\boxed{q_0}} \quad \underset{l_2}{\boxed{q_1}} \quad \underset{l_3}{\boxed{q_2}} \quad \cdots \quad \underset{l_n}{\boxed{q_n}}$$

抛弃用于阶段1和阶段2的内部节点，并按照下列新规则，通过组合权（q_i，q_j）建立起一些新的节点：

i′）含有 q_i 和 q_j 的节点在工作序列中必须是相邻的。

ii′）级 l_i 和 l_j 必须都是在所有剩下的级当中极大的。

iii′）下标 i 对满足（i′）、（ii′）的所有（q_i，q_j）必须为极小。

指定新节点（19）的级为 l_i-1。在这阶段形成的二叉树，其外部节点的权（从左到右）为 $q_0, q_1, \cdots, q_n$，且在所有具备这种性质的二叉树中，它有极小的加权路径长度。

图18给出此算法的一个例子。权 q_i 是字符␣，A，B，…，Z在英文课文中出现的相对频率。在阶段1，组合J和K的频率，形成的第一个节点是⑥；然后形成节点㊵（组合P和Q），然后

⑰，⑱，㉖，㉟，㊱，㊳，㊶，㊽，㊹，㊼，㊼，㊸；

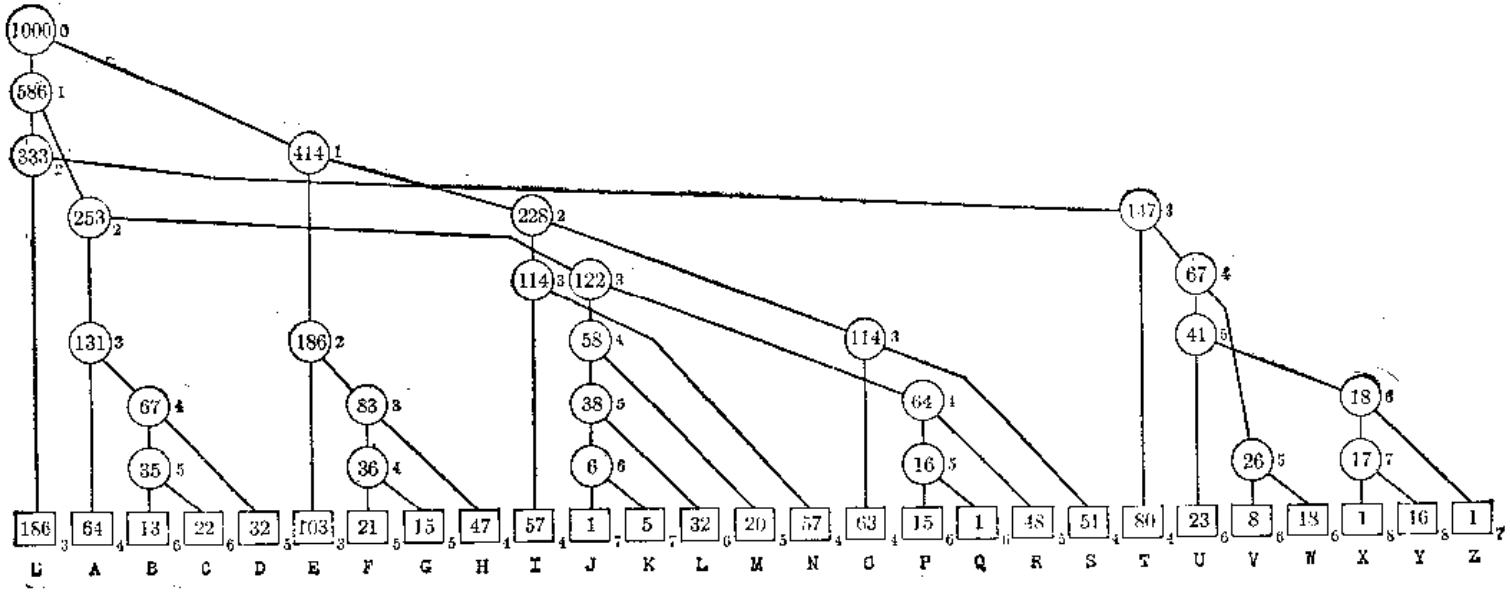


图18　胡一塔克算法应用于字母频率数据：阶段 1 和 2

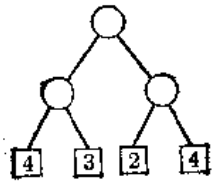
这时我们有工作序列

$$\boxed{186}\ \boxed{64}\ \textcircled{67}\ \boxed{103}\ \textcircled{83}\ \boxed{57}\ \textcircled{58}\ \boxed{57}\ \boxed{63}\ \textcircled{64}\ \boxed{51}\ \boxed{80}\ \textcircled{67} \tag{20}$$

根据规则（i），仅当不相邻的权被内部节点分开时，才能去组合它们；所以我们能组合 57＋57，然后 63＋51，然后 58＋64 等等。

在阶段 2 期间指定的级号，出现于图 18 中每个节点的右边。在阶段 3 期间的重新组合产生如图 19 所示的树；注意，在此树中的情况不同于图 18，因为图 18 不保持自左到右的次序。但是图 19 和图 18 有相同的代价，因为在两种情况下，外部节点都出现在相同的级上。

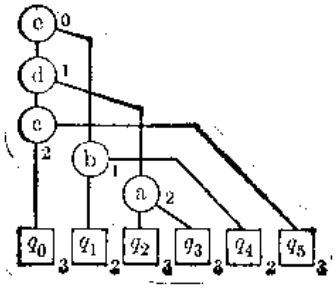
考虑一个简单的例子，其中诸权是 4，3，2，4；容易证明唯一的最优树是



(21)

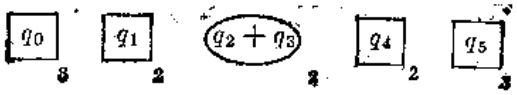
这个例子表明，两个最小的权 2 和 3，在一株最优树中不应该总是被组合在一起，即使当它们相邻时也如此；某些重新组合阶段是需要的。

至于胡-塔克算法的证明，已超出了本书的范围；还不知道有简单的证明，而且十分可能的是将永远找不到简单的证明！为了说明这种情况固有的复杂性，请注意阶段 3 必须把所有节点组成一株树，而这样做的可能性并不明显。例如，假设阶段 1 和 2 是通过组合节点ⓐ，ⓑ，ⓒ，ⓓ，ⓔ（按照这个次序）来构造树



(22)

这符合规则（i）。然后阶段 3 在形成



之后将停止下来，因为两个级 3 的节点不相邻！规则（i）本身不能保证阶段 3 一定能够进行，并且有必要证明在阶段 1 中造不出（22）这样的图形来。

当实现胡-塔克算法时，我们可以建立和使用那些不被外部节点分开的各节点权集合

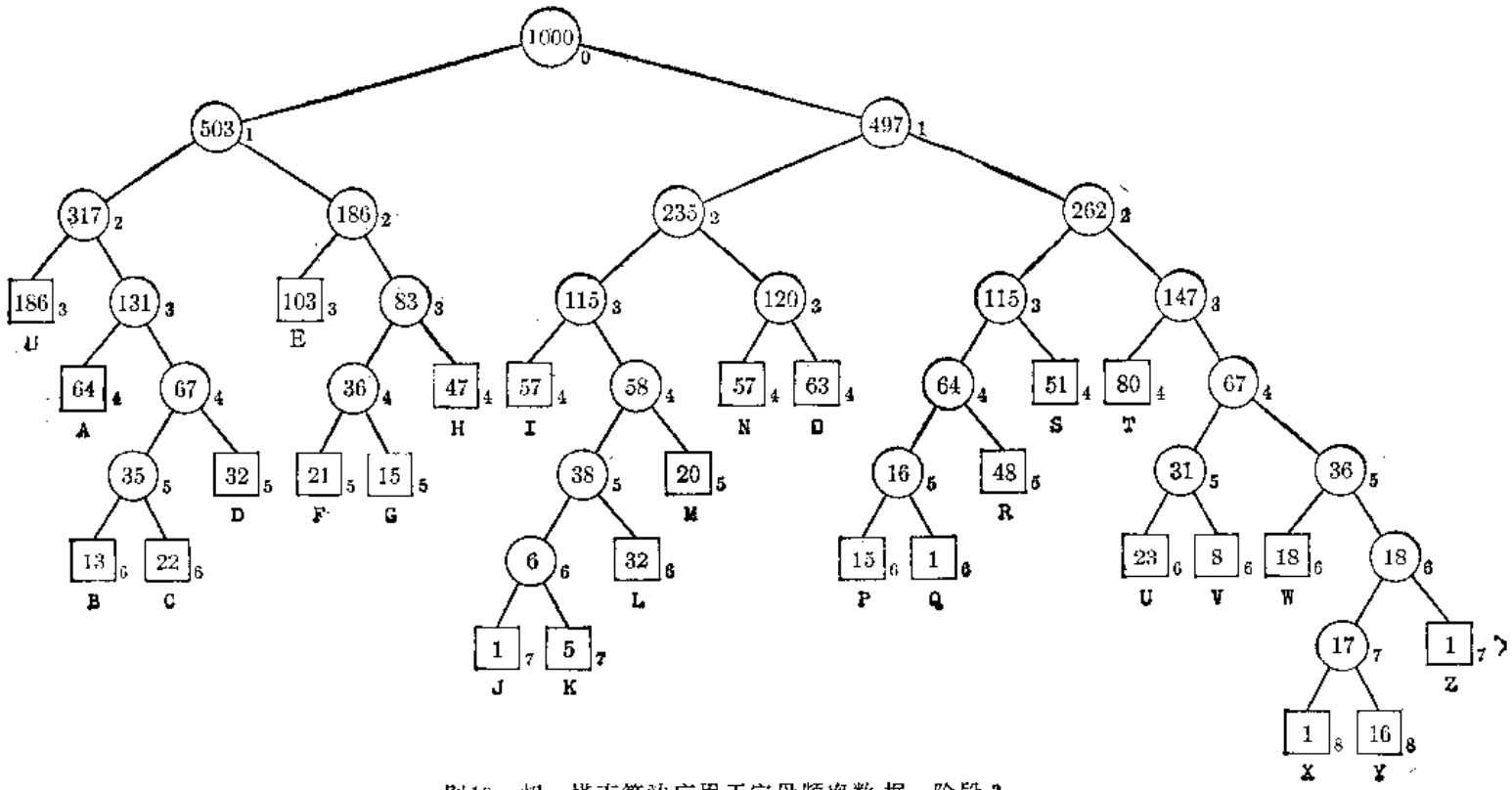


图19　胡一塔克算法应用于字母频率数据：阶段 3

的诸优先队。例如，(20) 可以表示成一些优先队，它们分别包含

$$\begin{array}{cccccccc} & 64 & 57 & 57 & 51 & & & \\ 64 & & & & 57 & & 51 & 67 \\ & 67 & 83 & 57 & 63 & & & \\ 186 & & & & 63 & & 80 & 80 \\ & 103 & 103 & 58 & 64 & & & \end{array} \tag{23}$$

再加上一些其它信息，如其中哪些是外部节点，以及按规则 (iii) 和 (iv) 解结时遵循的自左到右的次序。另一个“主”优先队，可以记住在其它队中两个最小元素之和。新节点 $57+57$ 的建立，引起上述优先队中的三个被合并了。当优先队被表示作左倾树(参考 5.2.3 节)时，阶段 1 的每个组合步骤至多需要 $O(\log_2 n)$ 个操作；因此当 $n\to\infty$ 时，$O(n\log_2 n)$ 个操作就足够了。当然对于小的 n，使用一个比较宜截了当的 $O(n^2)$ 的实现方法是更有效的。

图 19 中的最优二叉树，不仅对于查找，而且对于编码理论也有一个有趣的应用：用 0 表示树中的一个左分枝，用 1 表示一个右分枝，我们就得到下列可变长的代码字：

⊔	000	I	1000	R	11001	
A	0010	J	1001000	S	1101	
B	001100	K	1001001	T	1110	
C	001101	L	100101	U	111100	
D	00111	M	10011	V	111101	(24)
E	010	N	1010	W	111110	
F	01100	O	1011	X	11111100	
G	01101	P	110000	Y	11111101	
H	0111	Q	110001	Z	1111111	

于是，象“RIGHT ON”这样的一个消息将编码为串

1100110000110101111110000010111010

注意：虽然代码字的长度是可变的，但不难从左向右进行译码，这是因为树结构告诉我们，一个代码字何时结束和另一个代码字何时开始。这种编码保持了信息的字母次序，并且每个字母平均使用了约 4.2 个二进位。因此，这个代码可用来压缩数据文件，而不必破坏字母信息的字典次序(尽管如果我们忽略字母排列的限制，它还可以减小到每个字母 4.1 个二进位，但是，每个字母 4.2 个二进位，这个数字对所有二叉树代码来说是极小的。如果对字母对偶而不是对单个字母进行编码，则可以达到进一步的减少，同时保证字母的次序)。

E. N. 吉尔伯特 (E. N. Gilbert) 和 E. F. 穆尔已经导出了查找树的极小加权路径长度的一个有趣的近似上界。

定理 G 如果 $p_1=p_2=\cdots=p_n=0$，则一株最优二分查找树的加权路径长度位于

$$\sum_{0\le i\le n} q_i\log_2(Q/q_i) \text{ 和 } 2Q+\sum_{0\le i\le n} q_i\log_2(Q/q_i)$$

之间，其中 $Q=\sum_{0\le i\le n} q_i$。

证明 为取得下界，我们对 n 使用归纳法。如果 $n>0$，则对某个 k，加权外部路径长度至少是

$$Q+\sum_{0\leqslant i<k} q_i\log_2(Q_1/q_i)+\sum_{k\leqslant i\leqslant n} q_i\log_2((Q-Q_1)/q_i)$$

$$\geqslant \sum_{0\leqslant i\leqslant n} q_i\log_2(Q/q_i)+f(Q_1)$$

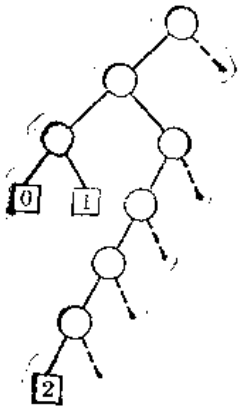
其中

$$Q_1=\sum_{0\leqslant i<k} q_i$$

$$f(Q_1)=Q+Q_1\log_2 Q_1+(Q-Q_1)\log_2(Q-Q_1)-Q\log_2 Q$$

函数 $f(Q_1)$ 是非负的，且当 $Q_1=\frac{1}{2}-Q$ 时取极小值 0。

为得到上界，我们可以假定 $Q=1$。对于 $0\leqslant i\leqslant n$，设 e_0，…，e_n 是使 $2^{-e_i}\leqslant q_i<2^{1-e_i}$ 的整数，用二进制记号表示分数 $\Sigma_{0\leqslant k<i}q_k+\frac{1}{2}q_i$，取其最高的 e_i+1 位二进数字，用以构造 0 和 1 的代码字 C_i。习题 35 证明当 $i\neq j$ 时，C_i 决不是 C_j 的一个初始子串；由此得出，我们可以构造对应于这些代码字的一株二分查找树。例如，当诸 q 是图19中的字母频率时，这个构造就给出 $C_0=0001$，$C_1=00110$，$C_2=01000001$，$C_3=0100011$，等等；这株树开始形状为



(25)

（这些代码右端的诸冗余位常可消去。）由这个一般步骤构造的二叉树的加权路径长度是

$$\leqslant \sum_{0\leqslant i\leqslant n}(e_i+1)q_i<\sum_{0\leqslant i\leqslant n} q_i(2+\log_2(1/q_i))$$

上面证明的第一部分，容易推广至证明每株二叉树的加权路径长度至少必须是 $\Sigma_{0\leqslant i\leqslant n}q_i\log_2(Q/q_i)$，而不论权是否要求处于从左到右的顺序（这个奠基性的结果是由克劳德·香农（Claude Shannon）给出的）。因此，自左到右的限制并不会使极小树增加多于两个额外级的代价，即两倍于权的总数。

历史和参考文献 这一节的树查找法是在 50 年代由若干人独立地发现的。在 1952 年 8 月印出的未公开的纪念性文章中，A. I. 杜米描述了树插入的原始形式：

“考虑其中可存储 2^n 个项目的一个磁鼓，每一个项目有一个二进地址，遵循下列程序：

1. 读入头一个项目并把它存于地址 2^{n-1} 中，即存储位置的一半处。
2. 读入下一项目，把它同头一个进行比较。

3．如果它更大，则把它放置在 $2^{n-1}+2^{n-2}$ 的位置处，如果它更小，则把它放置在 2^{n-2} 处…。”

树插入的另一种早期形式是由 D. J. 惠勒提出的，他实际上考虑了类似于我们将在 6.2.4 节讨论的多路分枝。一个二叉树插入技术也由 G. M. 伯纳斯-李独立地想出来了〔见 Comp. J. 2 (1959)，5〕。

P. F. 温德利（P. F. Windley）〔*Comp. J.* 3 (1960)，84-88〕、A. D. 布思和 A. J. T. 科林（A. J. T. Colin〔*Information and Contnol* 3 (1960)，327-334〕和托马斯·N. 纳希巴德〔*JACM* 9 (1962)，13-28〕首先发表了树插入的描述。所有这三篇文章的作者似乎都已经彼此独立地发展了这个方法，并且都给出了对于平均比较数（6）的稍微不同的证明。他们还讨论了这个算法的不同方面：温德利给出了对树插入排序的一个详细的讨论；布思和科林讨论了使前 2^{n-1} 个元素形成一株完全平衡的树这种预先处理的效果（见习题 4）；希巴德提出了删去的思想并说明了树插入分析和快速排序分析之间的联系。

最优二分查找树的思想，最先是针对 $p_1=\cdots p_n=0$ 的特殊情况，在研究类似（24）的字母二进编码时提出来的。E. N. 吉尔伯特和 E. F. 穆尔的一篇非常有趣的文章〔*Bell System Tech. J.*〕38 (1959)，933-968〕讨论了这个问题，以及它与其它编码问题的关系。在其它情况下，吉尔伯特和穆尔发现，利用类似算法 K 的一个算法，但不必要求单调性关系（17），可以在 $O(n^3)$ 步内构造出一株最优树。K. E. 艾弗森〔*A programming Language* (Wiley，1962)，142-144〕独立地考虑了其它情况，即当所有 q 都为 0 时的情况。他建议，如果把根选择成使左子树和右子树的概率尽可能地相等，则将得到一株最优树。D. E. 克努特〔*Acta Information* 1 (1971)，14-25，270〕随后考虑了一般的 p 和 q 个权的情况，并证明了这个算法可减少到 $O(n^2)$ 步；他还阐述了编译程序应用中的一个例子，其中，树中的键是在一种类 ALGOL 语言中的“保留字”。对于 $p=0$ 的情况，胡德强研究他自己的算法已经好些年了；由于这个问题的复杂性，关于这个算法正确性的一个严格证明不大好找，但是 1969 年在 A. C. 塔克的参与下他终于得到了一个证明〔*SIAM J. Applied Math.* 21(1971)，514-532〕。

习题

1．〔15〕 算法 T 仅仅叙述了非空树的情形，为了使它对于空树也能正确地工作，应作些什么变化？

2．〔20〕 修改算法 T 使它对右穿线树有效（参考 2.3.1 节；在这样的树中对称遍历更容易一些）。

▶3．〔20〕 在 6.1 节中，我们看到，对顺序查找算法 6.1S 稍作修改，就能达到加快的效果（算法 6.1Q）。类似的技巧可以用来加速算法 T 吗？

4．〔*M*24〕（A. D. 布思和 A. J. T. 科林）给出随机顺序下的 N 个键，假设我们使用前 2^n-1 个键构造一株完全的平衡树，同时，对于 $0\leqslant k<n$ 置 2^k 个键于 k 级上；然后使用算法 T 来插入剩下的键。试问在一次成功的查找中，平均比较次数是多少？〔提示：修改等式（2）。〕

▶5．〔*M*25〕 有 $11!=39916800$ 种不同的次序把名字 CAPRICORN、AQUARIUS

等插入到一株二分查找树中。（a）这些排列中有多少将产生图 10？（b）这些排列中有多少将产生一株蜕化的树，其中，每个节点上的 LLINK 或 RLINK 为 Λ？

6.〔$M26$〕 设 P_{nk} 是 $\{1, 2, \cdots, n\}$ 的这样一些排列 $a_1a_2\cdots a_n$ 的数目，它们使得如果使用算法 T 逐次地把 a_1，a_2，…，a_n 插入到一株空树中，则当插入 a_n 时，恰好要做 k 次比较（在这个问题中，将忽略插入 a_1，a_2，…，a_{n-1} 时所要做的比较。在正文的记号下，我们有 $C'_{n-1}=\Sigma(kP_{nk})/n!$。这是对一株含有 $n-1$ 个元素的树的不成功的查找中所作的平均比较次数）。

a）证明 $P_{n+1,k}=2P_{n,k-1}+(n-1)P_{n,k}$〔提示：考虑在这株树中 a_{n+1} 是否落在 a_n 之下〕。

b）试求母函数 $G_n(z)=\Sigma_kP_{nk}z^k$ 的一个简单公式，并使用你的公式按斯特林数来表达 P_{nk}。

c）这个分布的方差是多少？

7.〔$M30$〕〔S. R. 阿罗拉（S. R. Arora）和 W. T. 登特（W. T. Dent）〕在以随机顺序把 n 个元素插入到一株开始时为空的树之后，为找出第 m 个最大的元素，所需要做的平均比较次数是多少？

8.〔$M38$〕 用算法 T 由 n 个随机排序的键构造一株树，以 $p(n, k)$ 表示这株树的内部路径长度是 k 的概率（内部路径长度，是在构造树的过程中，由树插入排序所作的比较数）。（a）试求定义相应的母函数的递归关系。（b）计算这个分布的方差〔1.2.7 节中的若干习题在这里是有用的！〕

9.〔41〕 我们已经证明了，当以随机顺序插入键时，树的查找和插入仅需要大约 $2\ln N$ 次比较；但实际上，顺序不能是随机的。试作一经验研究，看看树插入实际上对于一个编译程序和/或汇编程序内的符号表的适用程度如何。在典型的大型程序中的标识符，是否会产生一株相当好的平衡二分查找树？

▶10.〔22〕 假设一个程序员对这个算法的排序性质不感兴趣，但他希望输入将以非随机的顺序进入。试讨论，通过把输入弄成“看起来象是”以随机次序进入而使他仍能使用树查找的一些方法。

11.〔22〕 当从大小为 N 的一株树中删去一个节点时，在步骤 D3 中实施 S←LLINK（R）的最大可能次数是多少？

12.〔$M22$〕 当从 N 个项目的一株随机树作一随机删去时，平均说来，步骤 D1 转到 D4 的次数是多少（见定理 H 的证明）？

▶13.〔$M23$〕 如果通过算法 D 删去一株随机树的根，则得到的树仍然是随机的吗？

▶14.〔22〕 证明带有附加步骤 D1$\frac{1}{2}$ 的算法 D 产生的树的路径长度，决不多于没有这个附加步骤时产生的树的路径长度。试找出步骤 D1$\frac{1}{2}$ 实际上减少路径长度的一种情况。

15.〔$M47$〕 当把新步骤 D1$\frac{1}{2}$ 加到算法 D 中时，在一长串的插入和删去之后，所影响的算法 T 的平均运行时间是多少？

▶16.〔25〕 删去操作是可交换的吗？即是说，如果使用算法 D 删去 X 和 Y，则得到

的树是否和使用算法D删去Y和X所得到的一样？

17.〔25〕 证明，如果算法D中的左和右的作用完全被颠倒，则容易推广这个算法，使得它从右穿线树删去一个给定的节点，同时保持必要的穿线（参见习题2）。

18.〔$M21$〕证明由吉甫定律可推出（12）。

19.〔$M23$〕 当输入概率满足等式6.1-11、12中定义的“80-20”定律时，式（11）近似的平均比较次数是多少？

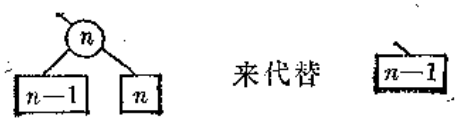
20.〔$M20$〕 假设我们以频率递减的次序$p_1 \geqslant p_2 \geqslant \cdots \geqslant p_N$，把键插入到一株树当中，这株树会比最优查找树明显地坏吗？

21.〔$M20$〕 如果p、q、r是随机选择的概率，并受到条件$p+q+r=1$的约束，则（13）中的树Ⅰ、Ⅱ、Ⅲ、Ⅳ和Ⅴ分别为最优的概率是多少（考虑图14中诸区域的相对面积）？

22.〔$M20$〕 证明，当实施算法K的步骤K4时，$r[i, j-1]$决不大于$r[i+1, j]$。

▶23.〔$M23$〕 对于$n=40$的情况，以及权$p_1=9$ $p_2=p_3=\cdots=p_{40}=1$，$q_0=q_1=\cdots=q_{40}=0$，求一株最优的二分查找树（不要使用计算机）。

24.〔$M25$〕 给定$p_n=q_n=0$，并设其它权均非负，证明$(p_1, \cdots, p_n; q_0, \cdots, q_n)$的一株最优树，可以通过在$(p_1, \cdots, p_{n-1}; q_0, \cdots, q_{n-1})$的任何最优树中以



得到。

25.〔$M20$〕 设A和B是实数的非空集合，以$A \leqslant B$表示下列性质：

（$a \in A$，$b \in B$，且$b < a$）蕴含（$a \in B$，且$b \in A$）。

（a）证明这个关系对于非空集合是传递的。（b）证明或否定：$A \leqslant B$当且仅当$A \leqslant A \cup B \leqslant B$。

26.〔$M22$〕 设$(p_1, \cdots, p_n; q_0, \cdots, q_n)$是非负的权，其中$p_n+q_n=x$，证明当$x$从0变化到$\infty$，而$(p_1, \cdots, p_{n-1}; q_0, \cdots, q_{n-1})$保持不变时，一株最优二分查找树的代价$c(0, n)$是具有整数斜率的“凸的、连续的、分段线性的”$x$的函数。换言之，证明存在正整数$l_0 > l_1 \cdots > l_m$和实常数$0=x_0<x_1<\cdots<x_m<x_{m+1}=\infty$，以及$y_0<y_1<\cdots y_m$，使得$0 \leqslant h \leqslant m$，$x_h \leqslant x \leqslant x_{h+1}$时，$c(0, n)=y_h+l_h x$。

27.〔$M33$〕 本题的目的是，证明每当权$(p_1, \cdots, p_n; q_0, \cdots, q_n)$非负时，最优二分查找树的根$R(i, j)$的集合满足

$$R(i, j-1) \leqslant R(i, j) \leqslant R(i+1, j) \quad 对于 j-i \geqslant 2$$

这里使用了习题25中定义的关系。这个证明是通过对$j-i$使用归纳法进行的；我们的任务是证明，假定$n \geqslant 2$且对于$j-i<n$，上面的关系成立，则$R(0, n-1) \leqslant R(0, n)$〔由左右的对称性，得出$R(0, n) \leqslant R(1, n)$〕。

a）证明如果$p_n=q_n=0$，则$R(0, n-1) \leqslant R(0, n)$（见习题24）。

b）设 $p_n+q_n=x$，在习题 26 的记号下，设 R_h 是当 $x_h<x<x_{h+1}$ 时最优根的集合 $R(0,\ n)$，并设 R'_h 是当 $x<x_h$ 时最优根的集合，证明

$$R'_0\leqslant R_0\leqslant R'_1\leqslant R_1\leqslant\cdots\leqslant R'_m\leqslant R_m$$

因此，由（a）部分和习题 25，对于所有的 x，我们有 $R(0,\ n-1)\leqslant R(0,\ n)$。

〔提示：考虑 $x=x_h$ 的情况并假定两株树

都是最优的，且 $s<r$ 和 $l\geqslant l'$。利用归纳法假设来证明，有一株根为 r 的最优树使得 $\boxed{n}$ 在 l' 级上，以及一株根为 s 的最优树，使得 $\boxed{n}$ 在 l 级上〕。

28.〔24〕 使用某种宏汇编语言来定义一个“最优二分查找”宏，其参数是一个最优二叉树的嵌套说明。

29.〔40〕 在图 12 的频率数据下，对于 31 个最常用的英文字，什么是最坏的二分查找树？

30.〔M41〕 证明或否定，最优二分查找树的代价满足 $c(i,\ j)+c(i+1,\ j-1)\geqslant c(i,\ j-1)+c(i+1,\ j)$。

31.〔M20〕（a）如果（22）中的权 $(q_0,\cdots,q_5)$ 分别为（2，3，1，1，3，2），试问这株树的加权路径长度是多少？（b）具有这个权序列的最优二分查找树的加权路径长度是多少？

▶32.〔M22〕（胡德强和A. C. 塔克）证明，在胡—塔克算法的第一阶段形成的新节点的权 q_i+q_j，是以非减次序建立的。

33.〔M41〕 为了找出使程序T的运行时间极小化的二分查找树，我们把量 $7C+C1$ 极小化，而不是简单地把比较次数 C 极小化。试提出一个算法，当这株树中的左、右分枝代价不同时，它找出最优的二分查找树。〔顺便指出，当右代价是左代价的两倍且节点频率都相等时，斐波那契树是最优的，参考L. E. 斯坦菲尔（L. E. Stanfel），*JACM* 17（1970），508-517〕。在不能一次作三路比较的机器上，算法T的一个程序将要在步骤T2中作两次比较，一次是关于等于的比较，一次是关于小于的比较；B·希尔（B. Sheil）和V. R. 普拉特已经发现，这些比较不一定涉及同一个键，所以，看来最好有一株这样的二叉树，其内部节点确定一个相等判断或一个小于判断，但不一定两者兼备。这种情况与上面所述的问题不同，对它进行探讨是很有意思的。

34.〔41〕 试写出胡—塔克算法的一个程序，它只使用 $O(n)$ 个存储单元和 $O(n\log_2 n)$ 个时间单位。

35.〔M23〕 证明在定理G的证明中构造的代码字有这样一个性质，即当 $i\neq j$ 时，C_i 决不以 C_j 开始。

36.〔M40〕（保罗·J. 拜尔（Paul J. Bayer））推广定理G的上界，证明任何具有非负权的最优二分查找树的代价至多是总计的权 $S=\Sigma_{1<i<n}p_i+\Sigma_{0<j<n}q_i$ 乘以 $H+2$，其中

$$H=\Sigma_{1<i<n}(p_i/s)\log_2(s/p_i)+\Sigma_{0<i<n}(q_i/s)\log_2(s/q_i)$$

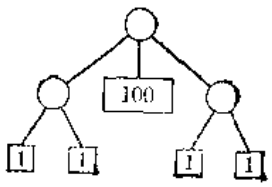
事实上，一个由顶向下过程，将产生满足这个界的一株二分查找树，这个过程反复地按如下原则选择根，即取极大子树权中的极小者。进一步证明，最优二分查找树的代价$\geqslant S$乘$H-\log_2(2H/e)$。

37.〔$M25$〕（胡德强和陈国财）设$n+1=2^m+k$，其中$0\leqslant k\leqslant 2^m$。有$\binom{2^m}{k}$株二叉树，其中所有外部节点都出现在$m$级和$m+1$级上。证明，在所有这些树当中，如果对充分大的$M$把胡—塔克算法应用于权（$M+q_0$，…，$M+q_n$）上，则我们得到对于权序列（$q_0$，…，$q_n$）具有极小加权路径长度的一株树。

38.〔$M35$〕（陈国财）证明，在满足$p_1+\cdots+p_n+q_0+\cdots+q_n=1$的所有概率集合（$p_1,\cdots,p_n$；$q_0,\cdots,q_n$）当中，当$p_i=0$（对所有$i$），$q_j=0$（对所有偶数的$j$）以及$q_j=1/\lceil n/2\rceil$（对所有奇数$j$）时，出现最昂贵的极小代价树。〔提示：给定任意概率（$p_1,\cdots,p_n$；q_0，…，q_n），设$c_0=q_0$，对于$1\leqslant i\leqslant n$，$c_i=p_i+q_i$，且$S(0)=\varnothing$；又对于$1\leqslant r\leqslant\lceil n/2\rceil$，令$S(r)=S(r-1)\cup\{i,j\}$，其中$c_i+c_j$是在所有使得$i,j\notin S(r-1)$的$i<j$中之极小者，且对于所有$i<k<j$，$k\in S(r-1)$。试构造在$q+1$级上有$S(n+1-2^q)$的外部节点和在$q$级上有其它外部节点的二叉树$T$，其中$q=\lfloor\log_2 k\rfloor$。证明这株树的代价$\leqslant f(n)$，其中$f(n)$是所说“最坏的”概率的最优查找树的代价。〕

39.〔$M30$〕（黄泽权和张系国）考虑一个方案，这个方案也是通过T构造出二分查找树，唯一区别是，当节点数达到形如2^n-1的一个数时，这株树被重新组织为一株完全平衡的均匀树，且对于$0\leqslant k<n$，有2^k个节点在级k上。证明，在构造这样的树时所作的比较次数平均为$N\log_2 N+O(N)$（不难证明，重新组织所需要的时间量为$O(N)$）。

40.〔$M50$〕能否推广胡—塔克算法，以找出其中每个节点至多有叉数t的最优树？例如，当$t=3$以及权的序列是（1，1，100，1，1）时，最优树为



6.2.3 平衡的树

我们刚刚学习了树插入算法，当输入数据是随机的时，这个算法将产生好的查找树，但是仍然有出现一株令人讨厌的蜕化树的可能性。也许我们可以想出一个算法，它总能使树成为最优的；但不幸的是，似乎这非常困难。另一个思想是，记住总的路径长度，而且当它的路径长度，比如说，超过$5N\log_2 N$时就完全重新组织树。但在构造这株树的过程中，这个方法可能要求大约$\sqrt{N/2}$次重新组织。

1962年两名俄国数学家Г. М. 阿德尔森-维尔斯基（Г. М. Адэлсон-Вэлиский）和Э. М. 兰迪斯（Э. М. Ландиш），对维持一株好查找树的问题，发现了一个非常漂亮的解〔Доклад Академии Наук СССР 146(1962)，263-266；英文翻译在*Soviet Math.*

31259-1263上]。他们的方法仅仅要求每个节点增加额外两个二进位，并且决不使用多于$O(\log_2 N)$个操作来查找树或插入一个项目。事实上，我们将看到，他们的方法也导致了一项好的表示任意长度N的线性表的一般技术，使得下列每个操作仅以$O(\log_2 N)$时间单位就可完成：

i）找出具有一给定键的一个项目。

ii）给定k，找出第k个项目。

iii）把一个项目插入到一个确定的位置。

iv）删去一个确定的项。

如果我们对线性表使用顺序分配，则操作（i）和（ii）是高效的，但操作（iii）和（iv）却花费阶为N的步骤；另一方面，如果我们使用链接分配，则操作（iii）和（iv）是高效的，但是操作（i）和（ii）花费阶为N的步骤。线性表的一种树表示可以以$O(\log_2 N)$步骤来做所有四个操作。并且它也可能相当高效地来做其它标准操作，例如，我们可以在$O(\log_2(M+N))$步内把一个M个元素的表同一个N个元素的表链接起来。

实现所有这些的方法，涉及到我们称之为“平衡树”的概念。上一段是给平衡树做广告，它把平衡树宣扬成好似包治百病的万灵药，而使所有其它数据表示形式都相形见绌；当然，我们对于平衡树也应当有一种平衡的态度！在不涉及所有上面四个操作的应用中，我们可能达到少得多的开销和更简单的程序设计。此外，除非N相当大，否则平衡树并没有优点；例如，如果有花费$20\log_2 N$时间单位的一个高效方法，和花费$2N$时间单位的一个低效方法，则我们将使用低效方法，除非N是大于1024的。另一方面，N也不应该太大；平衡树主要适合于数据的内部存储，而对于6.2.4节中的外部直接存取文件，我们将研究更好的方法。由于内存储器随着时间的推移似乎变得越来越大，因此平衡树也就越来越重要。

一株树的高度定义为它最大的级，即从根到一个外部节点的最长路径的长度。一株二叉树称为平衡的，如果每个节点左子树的高度同它的右子树的高度之差不超过±1。图20画出了具有17个内部节点和高度为5的一株平衡树；每个节点的平衡因子用+、或－表示，根据右子树的高度减左子树的高度是＋1、0或－1而定。图8（6.2.1节）中的斐波那契树是另一株高度为5的平衡二叉树，它仅有12个内部节点；在该树中平衡因子大多是－1。图10(6.2.2节）中的黄道天宫树不是平衡的，因为在AQUARIUS和GEMINI

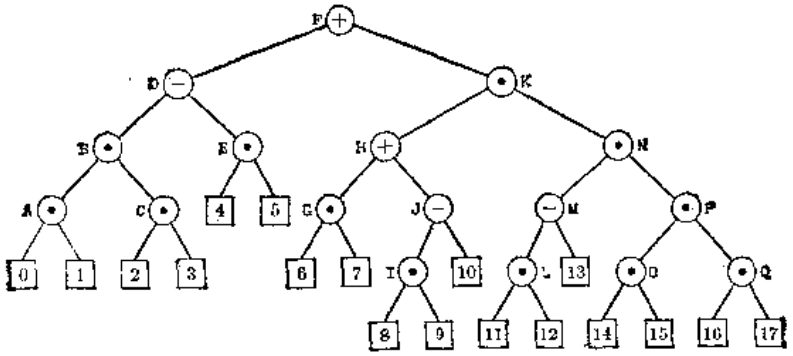


图20　一株平衡的二叉树

节点处不符合关于子树的高度限制。

平衡性这一定义表示了最优二叉树（要求所有外部节点都在两个相邻的级上）和任意二叉树（没有限制）之间的折衷。因此自然要问，一株平衡树比最优树差多少。答案是，它的查找路径长度决不会比最优树长百分之四十五以上。

定理A （阿德尔森·维尔斯基和兰迪斯）具有N个内部节点的一株平衡树的高度总是处于$\log_2(N+1)$和$1.4404\log_2(N+2)-0.328$之间。

证明 高度为h的一株二叉树显然不能有多于2^h个外部节点；所以$N+1\leqslant 2^h$，即在任何二叉树中$h\geqslant\lceil\log_2(N+1)\rceil$。

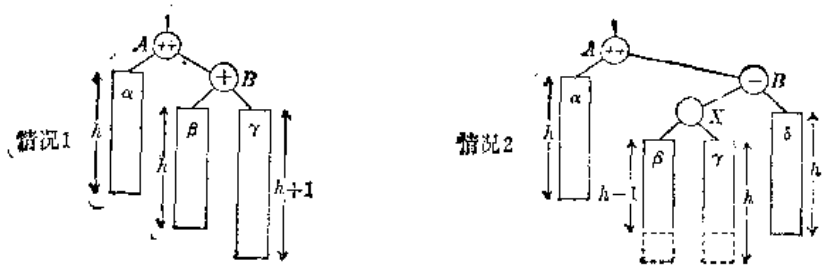
为了求h的极大值，我们把这个问题换一个提法，问在高度为h的一株平衡树中可能的极小节点数是多少。设T_h是具有最少节点的这样一株树；则这个根的子树之一，比如说左子树，高度为$h-1$，而另一株子树高度为$h-1$或$h-2$。由于我们要求T_h的节点数为极小所以可以假定这个根的左子树是T_{h-1}，而右子树是T_{h-2}。这段论证表明，阶为$h+1$的斐波那契树，在高度为h的所有可能的平衡树当中，节点数最少（见6.2.1节中斐波那契树的定义）。于是

$$N\geqslant F_{h+2}-1>\phi^{h+2}/\sqrt{5}-2$$

如同在定理4.5.3F的推论中那样，所述结果得证。

这个定理的证明表明，在一株平衡树的查找中，仅当这株树至少包含$F_{27}-1=196,417$个节点时，才需要25次以上的比较。

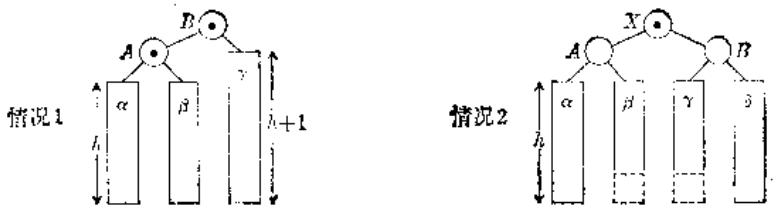
现在考虑当利用树插入（算法6.2.2T）把一个新节点插入到一株平衡树中时，将发生什么情况。在图20中，如果这个新节点位于[4]、[5]、[6]、[7]、[10]或[13]处，则这株树将是平衡的。但如果新节点落在别处，则需要作某些调整。问题发生在当我们有平衡因子为+1的一个节点时，这个节点的右子树在插入之后就更高了。或者，对偶地，如果平衡因子是−1，则左子树就更高了。不难看到，实际上只有如下两种情况会引起麻烦：



（1）

（如果我们反演这些图式，左右交换，则会出现另外两种实际上相同的情况）。在这些图式中，大的矩形α、β、γ、δ分别表示有图中所示高度的子树。当一个新元素刚把节点B的右子树的高度从h增加到$h+1$时出现情况1，当新元素增加B的左子树高度时出现情况2。在第二种情况下，我们或者有$h=0$（所以X本身是新节点），或者节点X有高度分别为（$h-1$，h）或（h，$h-1$）的两株子树。

在上述两种情况下，简单的变换将恢复平衡，同时保持树节点的对称次序：



（2）

在情况1中，我们简单地向左“转动”树，把B加到A而不是加到B，这个变换好象是对一个代数公式应用结合律似的，以（α β）γ来代替α（β γ）。在情况2中，我们使用双重转动，首先向右转动（x，B），然后向左转动（A，x）。在两种情况下，都只需要改变树的少量链接。而且，新树的高度是$h+2$，这恰是在插入之前的高度；因此原先在节点A上面的树的剩余部分（如果有的话），总是保持平衡的。

例如，如果我们把一个新节点插入到图20的位置12，则在一个转动之后（情况1）得到图21所示的平衡树。注意，若干平衡因子已经改变。

这个插入步骤的细节可以以若干方式给出，乍一看去，为了记住哪一些节点将受影响，似乎需要一个辅助的栈，但下面的算法用了一点小小的技巧，就避免了使用栈，从而赢得了速度。

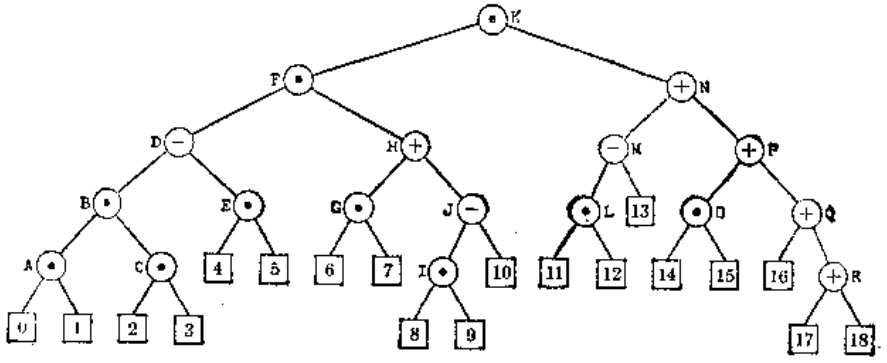


图21　图20的树，在插入了新的键R之后已重新平衡过

算法A　（平衡树的查找和插入）给定一个记录表，它形成如上所述的一株平衡的二叉树，本算法查找一个给定的变元K。如果K不在表中，则把一个包含K的新节点插入到树的适当位置，必要时重新平衡这株树。

如同在算法6.2.2T中那样，假定树的节点含有KEY、LLINK和RLINK诸场，我们还有一个新的场。

B（P）＝NODE（P）的平衡因子

即右子树的高度减去左子树的高度；这个场的内容不是＋1、0就是－1。一个特殊的表头节点也出现在树的顶上，在单元HEAD中；RLINK（HEAD）的值是指向树根的指针，而LLINK（HEAD）用来记住树的整个高度（这个算法实际上不需要知道高度。但在下面讨论的连接过程中它是有用的）。我们假定这株树是非空的，即RLINK（HEAD）≠Λ。

为了叙述方便，本算法使用记号LINK（a，P）作为$a=-1$时LLINK(P)，以及

$a=+1$ 时 RLINK（P）的同义语。

A1.〔初始化〕 置T←HEAD，S←P←RLINK（HEAD）（指针变量P将沿树下移，S将指向可能需要重新平衡的位置，且T总是指向S的父亲）。

A2.〔比较〕 如果$K<$KEY（P），则转到A3；如果$K>$KEY(P)，则转到A4；如果$K=$KEY(P)，则查找成功地结束。

A3.〔左移〕 置Q←LLINK(P)。如果Q＝Λ，则置Q⇐AVAIL和LLINK(P)←Q并转到步骤A5。否则，如果B(Q)≠0，则置T←P和S←Q。最后，置P←Q并返回到步骤$A2$。

A4.〔右移〕 置Q←RLINK(P)。如果Q＝Λ，则置Q⇐AVAIL和RLINK(P)←Q并转到步骤A5。否则，如果B(Q)≠0，则置T←P和S←Q。最后，置P←Q并返回到步骤A2（本步骤的最后部分可以同步骤A3的最后部分组合在一起）。

A5.〔插入〕（我们刚刚把一个新节点NODE(Q)链接到这株树中，而且它的场需要初始化。）置KEY（Q)←K，LLINK（Q)←RLINK(Q)←Λ，B(Q)←0。

A6.〔调整平衡因子〕（现在在S和Q之间的诸节点上的平衡因子需要从0变成为±1。）如果$K<$KEY(S)，则置R←P←LLINK(S)，否则置R←P←RLINK(S)。然后，重复地做下列操作0次或多次，直到P＝Q为止：如果$K<$KEY(P)，则置B(P)←－1和P←LLINK(P)；如果$K>$KEY(P)，则置B(P)←＋1和P←RLINK(P)。（如果K＝KEY(P)，则置P＝Q并且我们可以进到下一步。）

A7.〔平衡动作〕 如果$K<$KEY(S)，则置a←－1，否则a←＋1。现在出现若干情况：

i）如果B(S)＝0（这株树已经长得更高），则置B(S)←a，LLINK(HEAD)←LLINK(HEAD)＋1并结束此算法。

ii）如果B(S)＝－a（这株树更平衡了），则置B(S)←0并结束此算法。

iii）如果B(S)＝a（这株树失去了平衡），则如果B(R)＝a转到步骤$A8$，如果B(R)＝－a，转到步骤A9。

（情况（iii）对应于当a＝＋1时（1）中叙述的情况；S和R分别指向节点A和B，且LINK(－a，S）指向α，等等。）

A8.〔单转动〕 置P←R，LINK(a，S)←LINK(－a，R)，LINK(－a，R)←S，B(S)←B(R)←0，转到A10。

A9.〔双转动〕 置P←LINK(－a，R)，LINK(－a，R)←LINK(a，P)，LINK(a，P)←R，LINK(a，S)←LINK(－a，P)，LINK(－a，P)←S。现在置

$$(B(S),\ B(R))\leftarrow\begin{cases}(-a,\ 0)\ \text{如果}B(P)=a\\(0,\ 0)\ \text{如果}B(P)=0\\(0,\ a)\ \text{如果}B(P)=-a\end{cases}\qquad(3)$$

然后置B(P)←0。

A10.〔最后一步〕（我们已经完成了重新平衡的变换，把（1）变成（2），并且P指向新的根且T指向旧的根的父亲。）如果S＝RLINK(T)则置RLINK(T)←P，否则置LLINK(T)←P。

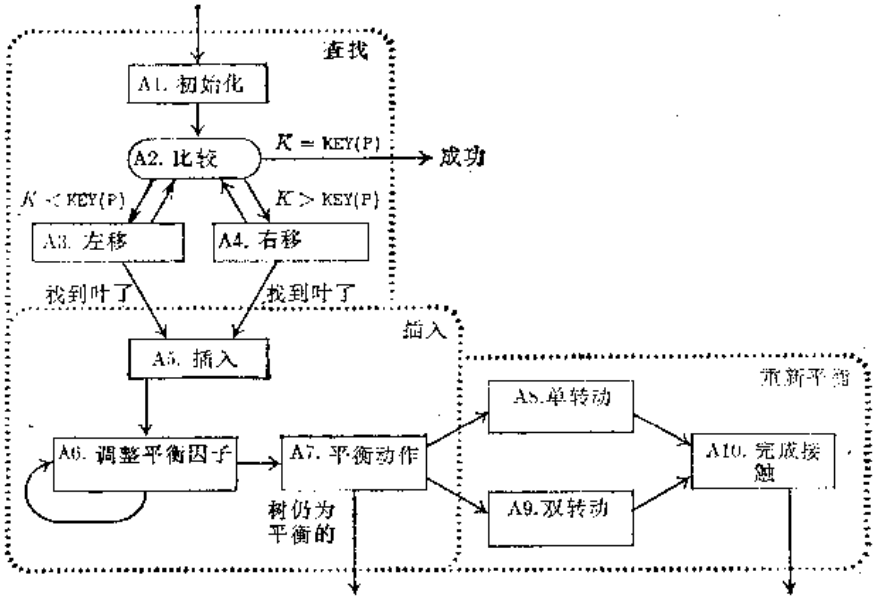
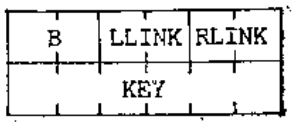


图22　平衡树的查找与插入

这个算法是相当长的，但它分为三个简单的部分：步骤 A1-A4 进行查找，步骤 A5-A7 插入一个新节点，而必要时步骤 A8-A10 重新平衡这株树。实际上，如果这株树是穿线的（参考习题6.2.2-2），则可使用同样的方法，因为平衡动作决不需要对穿线链接作困难的变动。

我们知道，这个算法花费大约 $C\log_2 N$ 个时间单位（C 是某个常数）。但重要的是要知道 C 的近似值，使得我们能了解 N 应当多大，使用平衡树才是值得的。下列的 MIX 实现能增进我们对这个问题的理解。

程序 A（平衡树的查找和插入）对于算法A，这个程序使用形如



（4）

的树节点，rA≡K，$rI1$≡P，$rI2$≡Q，$rI3$≡R，$rI4$≡S，$rI5$≡T，重复步骤 A7-A9 的代码，使得 a 的值隐式地（不是显式地）出现在程序中。

```
01   B       EQU     0:1
02   LLINK   EQU     2:3
03   RLINK   EQU     4:5
04   START   LDA     K              1       A1. 初始化
05           ENT5    HEAD           1       T←HEAD
06           LD2     0,5(RLINK)     1       Q←RLINK(HEAD)
07           JMP     2F             1       转A2且 S←P←Q
08   4H      LD2     0,1(RLINK)     C2      A4. 转移,Q←RLINK(P)
09           J2Z     5F             C2      如果Q = Λ 则转A5
10   1H      LDX     0,2(B)         C-1     rX←B(Q)
11           JXZ     *+3            C-1     如果B(Q) = 0 则转移
```

12		ENT5	0,1	$D-1$	T←P
13	2H	ENT4	0,2	D	S←Q
14		ENT1	0,2	C	P←Q
15		CMPA	1,1	C	*A*2. 比较
16		JG	4B	C	如果K>KEY(P)则转到A4
17		JE	SUCCESS	$C1$	如果K=KEY(P)则转出
18		LD2	0,1(LLINK)	$C1-S$	*A*3. 左移Q←LLINK(P)
19		J2NZ	1B	$C1-S$	如果Q≠Λ则转移
20	5H	LD2	AVAIL	$1-S$	*A*5. 插入
21		J2Z	OVERFLOW	$1-S$	
22		LDX	0,2(RLINK)	$1-S$	
23		STX	AVAIL	$1-S$	Q⇐AVAIL
24		STA	1,2	$1-S$	KEY(Q)←K
25		STZ	0,2	$1-S$	LLINK(Q)←RLINK(Q)←Λ
26		JL	1F	$1-S$	K<KEY(P)吗?
27		ST2	0,1(RLINK)	A	RLINK(P)←Q
28		JMP	*+2	A	
29	1H	ST2	0,1(LLINK)	$1-S-A$	LLINK(P)←Q
30	6H	CMPA	1,4	$1-S$	*A*6. 调整平衡因子
31		JL	*+3	$1-S$	如果K<KEY(S)则转移
32		LD3	0,4(RLINK)	E	R←RLINK(S)
33		JMP	*+2	E	
34		LD3	0,4(LLINK)	$1-S-E$	R←LLINK(S)
35		ENT1	0,3	$1-S$	P←R
36		ENTX	-1	$1-S$	rX←-1
37		JMP	1F	$1-S$	转到比较循环
38	4H	JE	7F	$F2+1-S$	如果K=KEY(P)则转到*A*7
39		STX	0,1(1:1)	$F2$	B(P)←+1(它原是+0)
40		LD1	0,1(RLINK)	$F2$	P←RLINK(P)
41	1H	CMPA	1,1	$F+1-S$	
42		JGE	4B	$F+1-S$	如果K≥KEY(P)则转移
43		STX	0,1(B)	$F1$	B(P)←-1
44		LD1	0,1(LLINK)	$F1$	P←LLINK(P)
45		JMP	1B	$F1$	转到比较循环
46	7H	LD2	0,4(B)	$1-S$	*A*7. 平衡动作rI2←B(S)
47		STZ	0,4(B)	$1-S$	B(S)←0
48		CMPA	1,4	$1-S$	
49		JG	A7R	$1-S$	如果K≥KEY(S)则转到 $a=+1$子程序
50	A7L	J2P	DONE		
51		J2Z	7F		
52		ENT1	0,3		
53		LD2	0,3(B)		

54		J2N	A8L		
55	A9L	LD1	0,3(RLINK)		
56		LDX	0,1(LLINK)		
57		STX	0,3(RLINK)		
58		ST3	0,1(LLINK)		
59		LD2	0,1(B)		
60		LDX	T1,2		
61		STX	0,4(B)		
62		LDX	T2,2		
63		STX	0,3(B)		
64	A8L	LDX	0,1(RLINK)		
65		STX	0,4(LLINK)		
66		ST4	0,1(RLINK)		
67		JMP	A8R1		
68	A7R	J2N	DONE	$1-S$	如果rI2 = $-a$ 则转出
69		J2Z	6F	$G+J$	如果B(S)原是0则转移
70		ENT1	0,3	G	P←R
71		LD2	0,3(B)	G	rI2←B(R)
72		J2P	A8R	G	如果rI2 = a 则转到A8
73	A9R	LD1	0,3(LLINK)	H	A9. 双转动
74		LDX	0,1(RLINK)	H	LINK(a, P←LINK($-a$, R))
75		STX	0,3(LLINK)	H	→LINK($-a$, R)
76		ST3	0,1(RLINK)	H	LINK(a, P)←R
77		LD2	0,1(B)	H	rI2←B(P)
78		LDX	T2,2	H	$-a$ 0或0
79		STX	0,4(B)	H	→B(S)
80		LDX	T1,2	H	0, 0或a
81		STX	0,3(B)	H	→B(R)
82	A8R	LDX	0,1(LLINK)	G	A8. 单转动
83		STX	0,4(RLINK)	G	LINK(a, S)←LINK($-a$, P)
84		ST4	0,1(LLINK)	G	LINK($-a$, P)←S
85	A8R1	STZ	0,1(B)	G	B(P)←0
86	A10	CMP4	0,5(RLINK)	G	A10. 最后一步
87		JNE	*-3	G	如果RLINK≠S 则转移
88		ST1	0,5(RLINK)	$G2$	RLINK(T)←P
89		JMP	DONE	$G2$	转出
90		ST1	0,5(LLINK)	$G1$	LLINK(T)←P
91		JMP	DONE	$G1$	转出
92		CON	+1		
93	T1	CON	0		(3)的表
94	T2	CON	0		
95		CON	-1		
96	6H	ENTX	+1	$J2$	rX←+1
97	7H	STX	0,4(B)	J	B(S)←a

98		LDX	HEAD(LLINK)	J	LLINK(HEAD)
99		INCX	1	J	+1
100		STX	HEAD(LLINK)	J	→LLINK(HEAD)
101	DONE	EQU	*	$1-S$	插入完成

平衡树插入的分析 〔非数学的读者，请跳到（10）。〕为了计算出算法A的运行时间，我们希望知道下列问题的答案：

- 在这个查找期间做了多少次比较？
- 节点S和Q相距多远（换言之，在步骤A6中需要多少次调整）？
- 单转动或双转动的频繁程度如何？

利用定理A不难导出最坏情况的运行时间的上界。但实际上我们当然要知道平均的性能。因为这个算法看起来颇为复杂，所以尚未能从理论上确定其平均性能，但是已经得到了某些有趣的经验结果。

首先，我们可以问起具有 n 个内部节点和高度 h 的平衡二叉树的数目 B_{nh}。不难从关系式

$$B_0(z)=1,\ B_1(z)=z,\ B_{h+1}(z)=zB_h(z)(B_h(z)+2B_{h-1}(z)) \tag{5}$$

计算对于小 h 的母函数 $B_h(z)=\sum_{n\geq 0}B_{nh}z^n$（见习题6）。于是

$$B_2(z)=2z^2+z^3$$
$$B_3(z)=4z^4+6z^5+4z^6+z^7$$
$$B_4(z)=16z^7+32z^8+44z^9+\cdots+8z^{14}+z^{15}$$

一般地，对于 $h\geq 3$，$B_h(z)$ 的形式为

$$2^{F_{h+2}-1}z^{F_{h+2}-1}+2^{F_{h+1}-2}L_{h-1}z^{F_{h+2}}+\text{复杂的项}+2^{h-1}z^{2h-2}+z^{2h-1} \tag{6}$$

其中 $L_k=F_{k+2}+F_{k-1}$（这个公式推广了定理A）。具有高度为 h 的平衡树的总数 $B_h=B_h(1)$，满足递归式

$$B_0=B_1=1,\ B_{h+1}=B_h^2+2B_hB_{h-1} \tag{7}$$

所以 $B_2=3$，$B_3=3\cdot 5$，$B_4=3^2\cdot 5\cdot 7$，$B_5=3^3\cdot 5^2\cdot 7\cdot 23$；而且，一般地说

$$B_h=A_0^{F_h}\cdot A_1^{F_{h-1}}\cdots A_{h-1}^{F_1}\cdot A_h^{F_0} \tag{8}$$

其中 $A_0=1$，$A_1=3$，$A_2=5$，$A_3=7$，$A_4=23$，$A_5=347$，…。$A_h=A_{h-1}B_{h-2}+2$。事实上，序列 B_h 和 A_h 增长非常迅速，它们都是“双重指数型的”。习题7表明有一个实数 $\theta\approx 1.43684$，使得

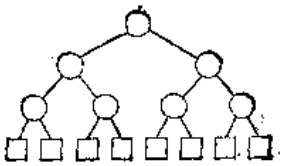
$$B_h=\lfloor\theta^{2^h}\rfloor-\lfloor\theta^{2^{h-1}}\rfloor+\lfloor\theta^{2^{h-2}}\rfloor-\cdots+(-1)^h\lfloor\theta^{2^0}\rfloor \tag{9}$$

如果我们考虑 B_h 株树的每一株都是同等可能的，则习题8表明在一个高度为 h 的树中节点的平均数是

$$B_h'(1)/B_h(1)\approx(0.70118)2^h \tag{10}$$

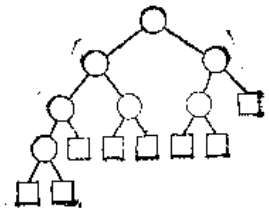
这表明具有 n 个节点的一株平衡树的高度，比起 $\log_6 n$ 来，通常更接近于 $\log_2 n$。

可惜，这些结果没有一个与算法A有关，因为该算法的机制使某些树比其它树出现机会更多些。例如，考虑 $N=7$ 的情况，其中可能有17株平衡树，有 $7!=5040$ 种可能的次序来插入7个键，并且完全平衡的“完备”树



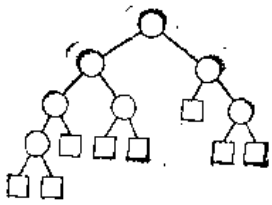
(11)

出现 2160 次。相反，斐波那契树



(12)

仅出现 144 次，而类似的树



(13)

则出现 216 次。〔以任意四节点的平衡树代替（12）和（13）的左子树，而后左右反演，得出 16 株不同的树；由（12）产生的八株出现 144 次，而由（13）产生的那些每株出现216次，（13）比（12）更为普遍，这是有些令人惊奇的。〕

以这样高的概率得到完全平衡树这一事实，连同（10）——它对应于相等概率的情况——使我们深信：一株平衡树的平均查找时间大约为 $\log_2 N + c$ 次比较，c 为某个小的常数。经验的检验指出，除非N很小，否则为插入第N个项所需要的平均比较次数近似于 $1.01\log_2 N + 0.1$。

为了研究算法A的插入和重新平衡阶段的性能，我们可以如在图 23 中所示那样，把平衡树的外部节点分成若干类。从一个外部节点向上走的通路可以用＋和－的序列予以描述（右链接为＋，左链接为－）；我们逐个写下链接说明直到达到第一个有非零平衡因子的节点为止。如果没有这样的节点，便直到达到根为止。然后，依据在把一个内部节点插入给定位置时，新树是平衡的还是不平衡的，来写A或B。例如，从③往上的通路是＋＋－B，意思是“右链接，右链接，左链接，不平衡”。以A结束的一个说明，在插入一个新节点之后不需要重新平衡；以＋＋B或－－B结束的说明，需要一次单转动；以＋－B或－＋B结束的说明，需要一次双转动。当k个链接出现于说明中时，步骤A6 恰好需要调整$k-1$个平衡因子。于是，这些说明给出了支配步骤A6 到A10 的运行时间的必要事实。

对于 $100 \leqslant N \leqslant 2000$ 的随机数的经验检验，表 1 给出了各种类型通路的近似概率；显然，当$N \to \infty$时这些概率迅速地趋于极限值若把输入的 $10!$个排列看成是同等可能的，则表 2 给出了当$N=10$ 时对应于表 1 的精确概率。

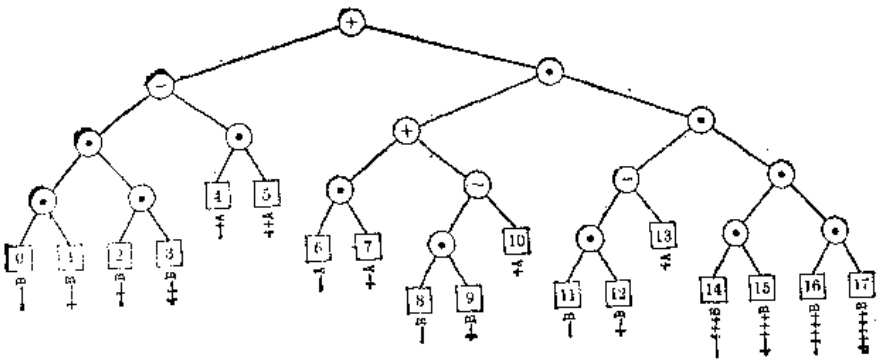


图23　分类码，它们确定插入后算法A的性能

表1❶　插入第N项的近似概率

路径长度k	没有重新平衡	单转动	双转动
1	0.144	0.000	0.000
2	0.153	0.144	0.144
3	0.093	0.048	0.048
4	0.058	0.023	0.023
5	0.036	0.010	0.010
≥ 5	0.051	0.008	0.007
平均2.78	0.535	0.233	0.232

❶ 作者在1978年的更正和补遗中指出，他将重新计算这个表，因为 0.144应是 0.143；因此也将修改下面的讨论并将引用习题11。——译注

表2　插入第10项的精确概率

路径长度	不重新平衡	单转动	双转动
1	1/7	0	0
2	6/35	1/7	1/7
3	4/21	2/35	2/35
4	0	1/21	1/21
平均247/105	53/105	26/105	26/105

从表1我们看到，k以.144＋.153＋.144＋.144＝.585的概率≤2；于是，步骤A6几乎百分之六十的时间是十分简单的。在该步骤中平衡因子从0变成±1的平均次数大约是1.8。在步骤A7到A10中，平衡因子从±1变成0的平均次数是.535＋2(.233＋.232)＝1.5；于是，插入一个新节点平均大约增加0.3个不平衡的节点。这同由算法A构造的随机树中，发现有68%的节点不平衡这一事实是一致的。

C.C.福斯特已经提出了算法A的特性的一个近似模型〔*Proc. ACM Nat. Conf.* 20 (1965)，192-205〕。这个模型不是严格地准确的，但它对于剖析该算法的本性来说，是相当接近于真实的。我们假定，p是在由算法A构造的一株大树中，一个给定节点的平衡因子为0的概率；于是平衡因子是＋1的概率为$\frac{1}{2}(1-p)$，是－1的概率也为$\frac{1}{2}(1-p)$。我们进一步假设（不作论证）所有节点的平衡因子都是独立的。于是，步骤A6恰把$k-1$个平衡因子置为非0的概率是$p^{k-1}(1-p)$，所以k的平均值是$p/(1-p)$。我们需要转动部分树的概率是1/2，插入一个新节点平均增加p个平衡节点；这个数在步骤A5

中被增加1，在步骤A6中增加$-p/(1-p)$，在步骤A7中增加$-\frac{1}{2}$，在步骤A8或A9中增加$-\frac{1}{2}\cdot 2$，所以我们有

$$p=1-p/(1-p)+\frac{1}{2}+1$$

对p求解就得到了同表1相当一致的结果：

$$p=\frac{9-\sqrt{41}}{4}\approx 0.649;\quad p/(1-p)\approx 2.851 \tag{14}$$

程序A查找阶段（行01-19）的运行时间是

$$10C+C1+2D+2-3S \tag{15}$$

其中C，$C1$，S和本章前面算法中的相同，D是在查找路径上所遇到的不平衡节点数。经验检验表明，我们可以取$D\approx\frac{1}{3}C$，$C1\approx\frac{1}{2}(C+S)$，$C+S\approx 1.01\log_2 N+0.1$，所以平均查找时间近似地为$11.3\log_2 N+3-13.7S$个单位。（如果查找比插入做得更经常，则我们当然可以使用对于查找的一个独立的、更快的程序，因为没有必要去查看平衡因子；因而，一次成功的查找的平均运行时间大约仅仅是$(\log_2 N+3)u$，而最坏的运行时间事实上将比算法6.2.2T所得到的平均运行时间要好。）

当查找不成功时，程序A的插入阶段（行20-45）的运行时间是$8F+26+$（0，1或2）个单位。表1的数据指出，平均说来，$F\approx 1.8$。重新平衡阶段（行46-101）所花费的时间，依赖于我们是增加总的高度，还是简单地不要重新平衡就转出程序，或者做一个单或双转动，而分别花费26.5、8、27.5或45.5(± 0.5)个单位。头一种情况几乎不出现，而其它情况出现的概率近似为.535、.233、.232，所以程序A的插入—重新平衡部分加在一起，其平均运行时间大约是$63u$。

这些数字指出，即使程序长一点，在内存中维持一株平衡树还是相当快的。如果输入数据是随机的，则在6.2.2节的简单树插入算法中，大约每次插入还快出$50u$；可以保证，即使对于非随机输入数据，平衡树算法仍是可靠的。

比较程序A同程序6.2.2 T的一个方法，是考虑后者的最坏情况。如果我们研究以递增的次序插入N个键到开始时为空的一株树中所需要的时间，则原来的程序A在$N\leqslant 26$时比它慢，而在$N\geqslant 27$时比它快。

线性表表示 现在我们转到本节开始时所作的断言，即平衡树可以以这样一种方式来表示线性表，使得我们既可以快速地插入诸项（克服顺序分配的困难），也可以对表项日实施随机存取（克服链接分配的困难）。

这想法就是在每个节点中引进一个新的称为RANK的场，它指出该节点在其子树内的相对位置，即，1加上它左子树中节点的个数，图24标出了图23的二叉树的RANK值。我们可以整个地消去KEY场；或者，如果需要，也可以同时有KEY和RANK两个场，使得有可能通过它们的键值或者它们在表中的相对位置来查找项日。

利用这样一个RANK场，通过位置进行查找，是对我们在一直研究的查找算法的一项直截了当的修改。

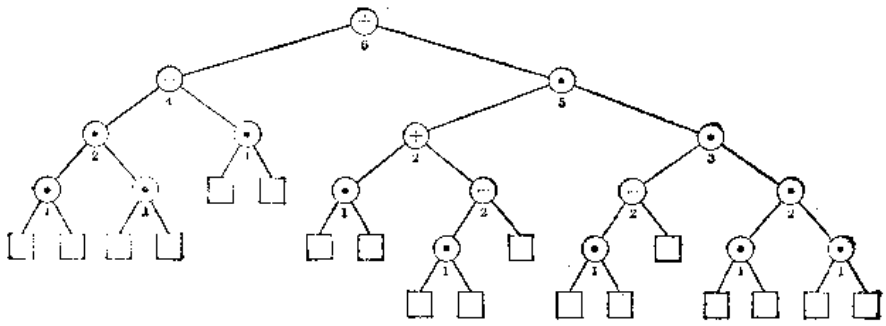


图24 用于通过位置进行查找的RANK场

算法B（通过位置进行树查找） 给定表示成一株二叉树的线性表，给定k，本算法寻找该表的第k个元素(在对称次序下树的第k个节点)。假定二叉树有LLINK场和RLINK场，以及如算法A中那样的一个表头，加上如前所述的一个RANK场。

B1.〔初始化〕 置M←k，P←RLINK（HEAD）

B2.〔比较〕 如果P＝A，则此算法以失败告终(仅当k大于树中的节点数，或$k\leqslant 0$时，这才可能发生)。否则，如果M<RANK(P)，则转到B3；如果M>RANK(P)，则转到B4；如果M＝RANK(P)，则本算法成功地结束（P指向第k个节点)。

B3.〔左移〕置P←LLINK(P) 并返回B2。

B4.〔右移〕 置M←M—RANK(P) 和P←RLINK(P) 并返回B2。

在本算法中唯一有趣之点是在步骤B4中M的操作。我们可以以类似的方式修改插入过程，尽管其细节是有些技巧的：

算法C(平衡树按位置插入) 给定表示成一株平衡二叉树的线性表，并给定k和指向一个新节点的指针Q，本算法恰在此表的第k个元素之前插入该新节点。如果$k=N+1$，则这个新节点恰被插入到此表的最末元素之后。

假定二叉树是非空的并有LLINK、RLINK和B场及一个表头，如算法A中那样，再加上如前所述的一个RANK场。本算法仅是算法A的改写；其差别只是它使用和更新RANK场而不是KEY场。

C1.〔初始化〕 置T←HEAD，S←P←RLINK（HEAD)，U←M←k。

C2.〔比较〕 如果M≤RANK(P)，则转到C3，否则转到C4。

C3.〔左移〕 置RANK(P)←RANK(P)+1（我们将插入一个新节点到P的左边)，置R←LLINK(P)，如果R＝A，则置LLINK(P)←Q，并转到C5。否则如果B(R)≠0，则置T←P，S←R以及U←M。最后，置P←R并返回到C2。

C4.〔右移〕 置M←M—RANK(P)和R←RLINK(P)。如果R＝A，则置RLINK(P)←Q并转到C5。否则如果B(R)≠0，则置T←P，S←R以及U←M。最后，置P←R并返回C2。

C5.〔插入〕 置RANK(R)←1，LLINK(Q)←RLINK(Q)←A，B(Q)←0。

C6.〔调整平衡因子〕 置M←U（这恢复了当P是S时M以前的值；所有RANK场现在都已适当地赋值)。如果M<RANK(S)，则置R←P←LLINK(S)，否则置R←

P←RLINK(S) 和M←M—RANK(S)。然后重复做下列操作直到P＝Q为止：如果M<RANK(P)，则置B(P)←－1以及P←LLINK(P)；如果M>RANK(P)，则置B(P)←＋1和M←M—RANK(P) 及P←RLINK(P)（如果M＝RANK(P)，则P＝Q而且我们可以进到下一步）。

C7.〔平衡动作〕 如果U<RANK(S)，则置a←－1，否则置a←＋1。现在出现若干情况：

i）如果B(S)＝0，则置B(S)←a，LLINK(HEAD)←LLINK(HEAD)＋1，并结束此算法。

ii）如果B(S)＝－a，则置B(S)←0并结束此算法。

iii）如果B(S)＝a，则如果B(R)＝a即转到步骤C8，如果B(R)＝－a即转到步骤C9。

C8.〔单转动〕 置P←R，LINK(a，S)←LINK(－a，R)，LINK(－a，R)←S，B(S)←B(R)←0，如果a＝＋1，则置RANK(R)←RANK(R)＋RANK(S)；如果a＝－1，则置RANK(S)←RANK(S)-RANK(R)，转向C10。

C9.〔双转动〕 进行步骤A9（算法A）的所有操作。然后，如果a＝＋1，则置RANK(R)←RANK(R)-RANK(P)，RANK(P)←RANK(P)＋RANK(S)；如果a＝－1，则置RANK(P)←RANK(P)＋RANK(R)，然后RANK(S)←RANK(S)-RANK(P)。

C10.〔最后一步〕 如果S＝RLINK(T)则置RLINK(T)←P，否则置LLINK(T)←P。

***删去、连接、等等** 对于平衡树和维持平衡还可做许多其它事情，但这些算法都是十分冗长的，它们的细节已超过了这本书的范围。这里我们只讨论一般的思想，有兴趣的读者不难自己补充其细节。

如果采用正确的方法，则删去的问题可以在$O(\log_2 N)$步内解决〔C. C. Foster, "A Study of AVL Trees", Goodyear Aerospace Corp. report GER-12158(April, 1965)〕。首先，我们可以把任意节点的删去简单地归结为一个节点P的删去，对这个节点P说来，如同在算法6.2.2D中那样，LLINK(P)或RLINK(P)是A。这个算法还应加以修改使它造一张确定到节点P的通路的指针表，即

$$(P_0 a_0),\ (P_1, a_1), \cdots,\ (P_l,\ a_l) \tag{16}$$

其中，P_0＝HEAD，a_0＝＋1；对于$0 \leqslant i < l$，LINK(a_i，P_i)＝P_{i+1}，P_l＝P；而且LINK(a_l，P_l)＝A。当我们往下查找树时，这张表可放置在一个辅助栈上。删去节点P的过程置LINK(a_{l-1}，P_{l-1})←LINK(－a_l，P_l)，而且我们必须调整节点P_{l-1}处的平衡因子。现在，假定需要调整节点P_k处的平衡因子，其原因是这个节点的a_k子树高度刚刚减少了；应当使用下列调整手续：如果k＝0，则置LLINK(HEAD)←LLINK(HEAD)-1并结束此算法，因为整个树高度减少了。否则考察平衡因子$B(P_k)$：有三种情况：

i）B(P_k)＝a_k。置B(P_k)←0，k减1，并对k的新值重复此调整手续。

ii）B(P_k)＝0。置B(P_k)为－a_k并结束删去算法。

iii）B(P_k)＝－a_k。需要重新平衡！

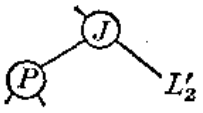
需要重新平衡的这些情况，几乎同我们在插入算法中遇到的一样；再次考虑（1），A是节点P_k B是节点LINK$(-a_k, P_k)$，即与删去的分枝相反的分枝。唯一的新特征是节点B可以是平衡的；这导致一个新的情况3，它和情况1类似，但β的高度是$h+1$。在情况1和2中，如同在（2）中那样的重新平衡，意味着减少高度，所以我们置LINK(a_{k-1}, P_{k-1})为（2）的根，k减1，并对这个新的k值重新开始调整。在情况3中，我们做一个转动，这就保持了A和B两者的平衡因子非0，而不改变整个高度；因此，在使LINK(a_{k-1}, P_{k-1})指向节点B之后，我们就结束这个算法。

删去和插入之间的重要差别是：删去可能需要多达$\log_2 N$次转动，而插入所需的转动决不多于一次。如果我们试图删去一株斐波那契树最右的节点（见6.2.1节图8），那么，这个原因就显得很清楚了。但是经验检验表明，每次删去平均只需作0.21次转动。

对于线性表表示使用平衡树，也提示了需要一个连接算法，以便把整株树L_2插入到树L_1的右边，而不破坏平衡。克拉克·A.克兰在假设高度（L_1）≥高度（L_2）的前提下，已经想出了用于连接的漂亮算法；其它情况是类似的。删去L_2的头一个节点，称它为"交界节点"J，并令L_2'代表新树$L_2\backslash\{J\}$，现在顺着L_1的诸右链接往下走，直到到达一个节点P，使得

$$\text{高度}(P)-\text{高度}(L_2')=0\text{或}1$$

为止；这总是可能的，因为每往下走一级，高度就改变1或2，然后以



来代替

，并进而调整L_1，这里把新节点J想象为是刚由算法A插入的。

克兰也解决了更困难的求逆问题，把一个表分成为两部分，使得它们的连接还是原来的表。例如，考虑把图20中的表分开来得到两个表，一个含{A，…，I}，另一个含{J，…，Q}；这需要对于诸子树作大量的重组工作。一般地说，当我们需要在某个给定的节点P处来分开一株树时，到P的通路有些象图25中那样。我们希望构造一株左树，它以对称次序包含α_1，P_1，α_4，P_4，α_6，P_6，α_7，P_7，α，P诸节点，以及一株右树，它包含β，P_8，β_8，P_5，β_5，P_3，β_3，P_2，β_2。这可以通过一系列的连接来做：首先在α的右边插入P，然后利用P_8做为交界节点连接β和P_8，利用P_7做为交界节点连接α_7和αP，利用P_6连接α_6和$\alpha_7 P_7 \alpha P$，利用P_5连接$\beta P_8 \beta_8$和β_5，等等；在到P的通路上的节点$P_8, P_7, \cdots, P_1$均被用做交界节点。克兰已经证明，当原来的树包含N个节点时，这个分开算法仅花费$O(\log_2 N)$个时间单位；主要原因是利用一个给定的交界点的连接要花费$O(k)$步，其

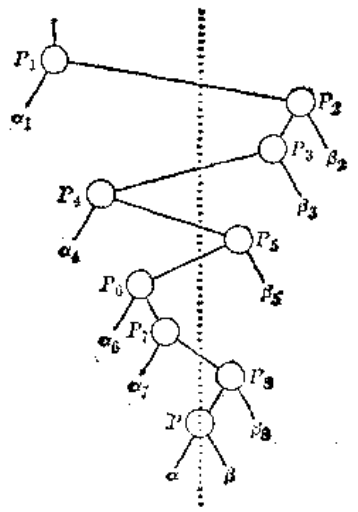


图25 分开一个表的问题

中 k 是正被连接的树之间的高度差，而且 k 的值必须累计起来，这些值实际上形成了正被构造的左树和右树两者一个收缩的级数。

所有这些算法都可以利用 KEY 或 RANK 场或者两者都用（尽管在连接的情况下，L_2 的键必须全都大于 L_1 的键）。为了通用的目的，通常可取的是，使用一株三重链接的树，它具有 UP 链以及 LLINK 和 RLINK，连同一个新的一位二进位场，这个场确定一个节点是其父亲的左儿子还是右儿子。三重链接树表示稍微简化了这些算法，并且使我们有可能确定树中的节点而无须明确地跟踪通到该节点的通路；给定 P，我们可以写出一个子程序删去 NODE(P)，或者删去在对称次序下跟随 P 的 NODE(P$)，或者求出包含 NODE(P) 的表，等等。在三重链接树的删去算法中不必构造表 (16)，因为 UP 链提供了我们所需要的信息。当然，一株三重链接树要求，当插入、删去和转动正被实施时，要改变稍微多的链接。使用一株三重链接树来代替一株双重链接树，类似于使用两路链接代替一路链接的情形：我们可以从任意点开始，或向前进行或向后进行。以三重链接平衡树为基础的表算法的完整的描述，出现在克拉克·A. 克兰的博士论文中（Stanford University, 1972）。

平衡树的变形 近来已有人提出组织树的某些其它方法，以保证对数阶的存取时间，眼前，这些方法还未得到非常透彻的研究。可能会证明，它们在某些计算机上，要比平衡树优越。

例如，C. C. 福斯特〔*CACM* 16(1973)，513-517〕已经研究了广义平衡树，当我们允许子树的高度差至多为 k 时出现这种树。这样的结构可称作 $HB[k]$ 树（意思是"高度平衡"），而通常的平衡树表示其特殊情况 $HB[1]$。P. L. 卡尔通（P. L. Karlton）等人已经讨论了 $HB[k]$ 树的经验测试，见 *CACM* 19(1976)，23-28。

加权平衡树的有趣概念，已经为 J. 尼弗杰尔特（J. Nievergelt），E. 莱因戈尔德和黄泽权所研究。我们不去考虑树的高度，而是约定所有节点的子树必须满足

$$\sqrt{2}-1<\frac{\text{左权}}{\text{右权}}<\sqrt{2}+1 \tag{17}$$

其中，左权和右权分别累计左子树和右子树中外部节点的数目。有可能证明，仅仅利用如算法 A 中（见习题 25）进行重新平衡的单转动和双转动，在插入之下仍能维持权的平衡。然而，在一次插入期间可能需要做许多重新平衡。在增加查找时间的条件下，减少重新平衡的次数，有可能放松 (17) 的条件。

乍一看，权平衡树似乎比以前单纯的平衡树需要更多的内存。但是事实上，它们有时要求反倒稍少些！对于线性表表示，如果每个节点已有一个 RANK 场，则这恰巧是左权。而当我们沿树下移时，有可能记住对应的右权。然而，为了维持权平衡所需要的管理操作，看来要花费比算法 A 更多的时间，而每个节点省去两位二进数位，似乎还抵不上所增加的麻烦。

平衡树的另一种有趣的变形，称作"2-3 树"，它是由约翰·霍普克洛甫特于 1970 年提出的〔见 Aho, Hopcroft and Ullman, The Design and Analysis of Computer Algorithms (Reading, Mass: Addison-Wesley, 1974), Chapter 4〕。这个思想要求在每个节点处作 2 路分支或 3 路分支，并约定所有外部节点都出现在相同的级上。每个内部

节点包含一个或两个键，如图 26 所示。

插入到一株 2-3 树要比插入到一株平衡树易于说明：如果我们要把一个新的键插入到仅含一个键的节点中，则只要简单地把它插入作为第二个键。另一方面，如果这个节点已经含有两个键，则我们就把它分为两个各有一个键的节点，而把中间的键插入到父母节点处。如果父母节点已有两个键，这又可以引起父母节点以相同的方式被划分。图 27 示出了把一个新的键插入到图 26 的 2-3 树的过程。

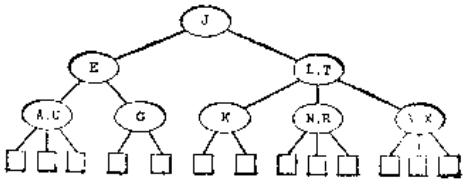


图26　一株3-2树

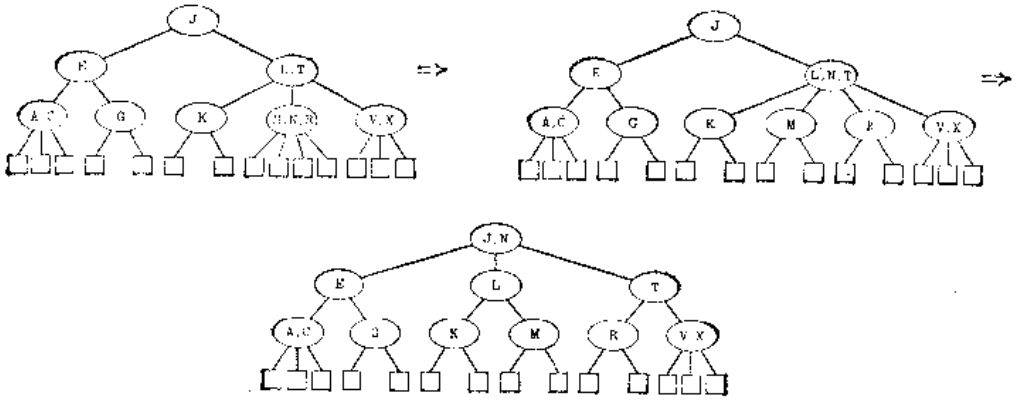


图27　把新键“M”插入图26的3-2树中

霍普克洛夫特已经发现，正象在平衡树的情况下那样，对于 2-3 树也可以相当直截了当的方式进行删去、连接和分开等操作。

R.拜尔(R.Bayer)〔*Proc. ACM-SIGFIDET Workshop*(1971),219-235〕已经提议使用 2-3 树的一种有趣的二叉树表示。见图 28，它给出图 26 的二叉树表示；在每个节点中，用一个二进位来区分诸“水平”的 RLINK 和诸“垂直”的RLINK。注意，正如任何二分查找树那，树的键以对称方式从左到右出现。结果表明，当我们如图 27 那样插入一个新键时，在这样一株二叉树上需要实施的转换，恰恰是把一个新键插入到一株平衡树时所用的单转动和双转动，不过我们只需要一个单转动和一个双转动（不是在算法A和C中的左右反演）。

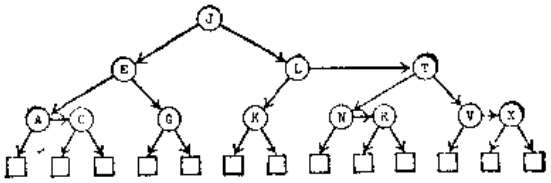


图28　图26的3-2树表示成一株二分查找树

习题

1.〔01〕 在（1）的情况 2 中，为什么通过简单地交换A和B的左子树来恢复平衡并不是一种好的想法？

2.〔16〕 如果我们以B（S）＝0 达到步骤 A7，试说明为什么这株树会高出一级来。

▶3.〔M25〕 证明：具有N个内部节点的一株平衡树决不包含多于$(\phi-1)N+$

0.61803N个其平衡因子非0的节点。

4.〔M22〕 证明或否定：在具有$F_{n+1}-1$个内部节点的所有平衡树当中，阶数为n的斐波那契树有最大的内部路径长度。

▶5.〔M25〕 证明或否定：如果算法A用来把N个键插入到一株开始时为空的树中，且这些键以递增的次序到达，则所产生的树总是最优的（即在所有N个节点的二叉树当中，它有极小的内部路径长度）。

6.〔M21〕 证明等式（5）定义了高度为h的平衡树的母函数。

7.〔M27〕（N. J. A.斯隆（N. J. A. Sloane）和A. V. 阿霍（A. V. Aho））证明高度为h的平衡树个数的著名公式（9）。〔提示：设$C_n=B_n+B_{n-1}$，且利用如下事实，即对于很大的n，$\log_2(C_{n+1}/C_n^2)$非常小。〕

8.〔M24〕（L. A. 基兹德尔（L. A. Khizder）〕 证明存在一个常数β，使得当$h\to\infty$时，$B_h'(1)/B_h(1)-2^n\beta\to 1$。

9.〔M47〕 具有n个内部节点的平衡二叉树的近似株数$\Sigma_{h\geqslant 0}B_{nh}$等于多少？近似平均高度$\Sigma_{h\geqslant 0}hB_{nh}/\Sigma_{h\geqslant 0}B_{nh}$等于多少？

10.〔M48〕 在算法A中，假定诸项以随机次序被插入，当插入第N项时，间所作的渐近平均比较次数是多少？

▶11.〔M24〕（马克·R·布朗（Mark R. Brown）〕 证明当$n\geqslant 6$时，由算法A构造的n个内部节点的一株随机平衡树中，属于+A、−A、++B、+−B、−+B、−−B等类型的任何一种外部节点的平均个数恰为$(n+1)/14$。

▶12.〔24〕 当把八个节点插入到一株平衡树中时，程序A可能的极大运行时间是多少？对于这个插入，可能的极小运行时间是多少？

13.〔10〕 为什么比较好的办法是如正文中定义的那样使用RANK场，而不是简单地存储每个节点的下标作为它的键（称第一个节点“1”，第二个节点“2”，等等）？

14.〔11〕 算法6.2.2T和6.2.2D能否如本节平衡树算法那样，适用于线性表？

15.〔18〕（C. A. 克兰） 假设一个有序线性表被表示为一株二叉树，且在每个节点处有KEY和RANK场。试设计一个算法，它在树上查找一个给定的键K，并确定K在表中的位置；即，它求得数M，使得K是第M个最小的键。

▶16.〔20〕 如果利用正文中的删去算法，从图20中同时删去节点E和根节点F，便得到一株平衡树，试画出这株树。

▶17.〔21〕 如果利用正文中建议的连接算法，把斐波那契树（12）连接到图20中树（a）右边，（b）左边，便得到两株平衡树，试画出这两株树。

18.〔21〕 如果利用正文中所建议的分开算法，把图20分成两部分{A，…，I}和{J，…，Q}便得到两株平衡树，试画出这两株树。

▶19.〔26〕 找出一个方法，转换一株给定的平衡树，使得在根处的平衡因子不是−1。你的转换应该保持节点的对称次序，而且不管原有树的大小如何，它都应该在O(1)的时间单位中产生另一株平衡树。

20.〔40〕 剖析利用一类平衡树的思想，这些树的节点的平衡因子限于是0或+1（于是B场的长度可减小到一个二进数位）。对于这样的树是否有一个相当有效的插入过程？

▶21.〔30〕 试设计一个算法，它在$O(N)$步骤内构造最优的N个节点的二叉树（在习题5的意义下）。你的算法应该是“联机的”，即是说，它逐个地以递增的次序输入节点，并随即构造部分树，而不必预先知道N的最后的值。(当重新构造一株失去平衡的树时，或者当合并两株树的键成为一株树时，使用这样一个算法是适当的。)

22.〔M20〕 对于加权平衡树，类似于定理A的结果是什么？

23.〔M20〕（E.莱因戈尔德）（a）证明存在其权平衡（左权)/(右权）为任意小的平衡树。(b）证明存在其左子树和右子树的高度差为任意大的加权平衡树。

24.〔M22〕（E.莱因戈尔德） 证明，如果我们把条件（17）增强到

$$\frac{1}{2}<\frac{\text{左权}}{\text{右权}}<2$$

则满足这一条件的仅有的二叉树是具有2^n-1个内部节点的完全平衡树（在这样的树中，左权和右权在所有节点处都完全相等)。

25.〔27〕（J.尼弗杰尔特，E.莱因戈尔德，黄泽权） 证明有可能设计加权平衡树的插入算法，使得条件（17）成立，并且使得每个插入至多进行$O(\log_2 N)$次转动。

26.〔40〕 试剖析对于$t>2$的平衡t叉树的性质。

▶27.〔M23〕 试估计在有N个内部节点的2-3树中进行查找所需要的极大比较次数。

28.〔41〕 编制2-3树算法的有效实现方案。

29.〔M47〕 试分析在随机插入下，2-3树的平均特性。

30.〔26〕（E.麦克里特（E.McCreight)） 2.5节讨论了动态存储分配的若干策略，包括“最好的适合”(从满足要求的所有那些区域当中选择尽可能小的可利用区域)和“第一个适合”(在所有满足要求的那些区域当中选择具有最小地址的可利用区域)。证明，如果以一种适当的方式把可利用空间链接在一起成为一株平衡树，则有可能只需要$O(\log_2 n)$个时间单位来进行（a）最好的适合和（b）第一个适合的分配，其中n是可利用区域数（在2.5节中给出的算法，花费阶为n的步骤)。

31.〔34〕（M.L.弗雷德曼） 试想出线性列表的这样一个表示——它具有如下的性质：给定m，在位置$m-1$和m之间插入一个新节点花费$O(\log_2 n)$个时间单位。

> 塞缪尔按部落一个一个地考虑以色列的民族，并且抽签选中了本杰明的部落。
> 然后他按家族一个一个地考虑本杰明的部落，并且抽签选中马特里的家族。
> 然后他按人一个一个地考虑马特里的家族，并且抽签选中基斯的儿子索尔。
> 但当他们寻找索尔时，却找不到他。
>
> ——1塞缪尔（Samuel)10:20-21

6.2.4 多路树

我们一直在讨论的树查找方法主要用于内部查找，即所要考察的表是完全包含在计算机高速内存中的。下面我们考虑，当从一个非常大的文件中查找信息时进行外部查找的问题。这些非常大的文件，出现在直接存取存储装置，例如盘或鼓上（5.4.9节有关于盘和鼓的介绍)。

如果我们选择一种适当的方法来表示树，那么树结构本身很适合于外部查找。考虑图

29 中所示的大型二分查找树，并想象它已经存在一个磁盘文件中（树的 LLINK 和 RLINK 现在是磁盘地址而不是内存地址）。如果我们以一个朴实的方式查找这株树，简单地应用对内部树查找已经学过的算法，则大约要做 $\log_2 N$ 次磁盘访问，才能完成查找。当 N 是 100 万时，这意味着我们需要做 20 次左右的寻找。但假如把这个表分成许多 7 个节点的"页"，如图 29 中虚线所示，如果我们一次访问一页，则仅需要前述次数的三分之一，所以查找大约快三倍。

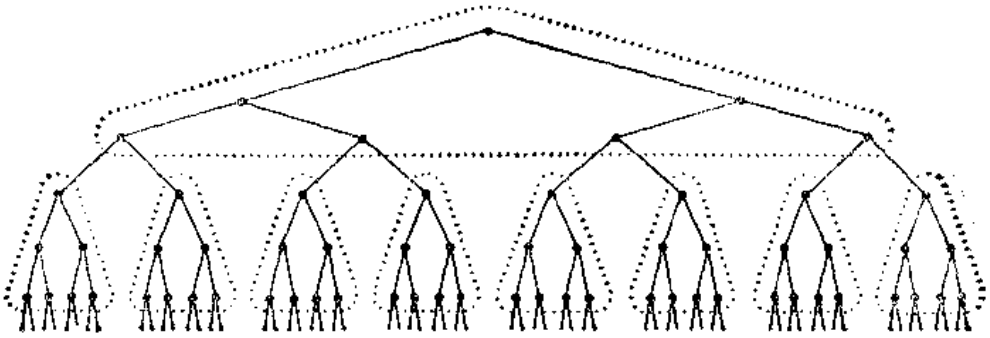


图29 一株大型二分查找树可以分成许多"页"

以这一方式把节点组合成页，实际上把一株二叉树变成了八叉树，在每个页节点处有 8 路分支。如果我们让页更大一些，在每次磁盘访问之后有 128 路分支，则在仅仅寻找三页之后，我们就可以在一个百万键的表中找出任何所希望的键。根页可以常驻内存，因此只需对盘进行两次访问，而且在任一时刻，内存中的键数都不超过 254 个。

当然我们不要使页任意大，因为，内存的大小有限，并且它需花费更长的时间来读入一个更大的页。例如，假设它花费 72.5+0.05m 毫秒来读允许 m 路分支的一个页，每页的内部处理时间约为 $a+b\log_2 m$，其中 a 相对 72.5 毫秒来说是很小的，因此，为查找一个大表所需要的时间总量近似地正比于 $\log_2 N$ 倍的

$$(72.5+0.05m)/\log_2 m+b$$

当 $m\approx 307$ 时，这个量取极小值；实际上，极小值是非常"广"的，200 和 500 之间的所有 m 都接近于达到一个最优值。事实上，根据外部存储设备的具体特征，以及表中记录的长度，将有一个类似的好的 m 值范围。

W. I. 兰道尔（W. I. Landauer）〔*IEEE. Trans.* EC-12(1963)，863-871〕提议，在构造一株 m 叉树时，首先要使第 l 级接近充满，然后才能往第 $l+1$ 级上放东西。这个方案需要一个相当复杂的转动方法，因为可能需要对整株树作重大的改变才能插入一个新的项目；兰道尔假设，我们需要比插入或删去项目更为经常得多地查找树中的项目。

当一个文件存在磁盘上，且插入或删去不多时，则一株三级树是适当的。其中分枝的第一级确定使用什么柱面，分枝的第二级确定在该柱面上的适当的磁道，第三级包含记录本身。这个方法称为索引顺序文件组织〔参考 *JACM* 16(1969)，569-571〕。

R. 芒茨（R. Muntz）和 R. 乌热加里斯（R. Uzgalis）〔*Proc. Princeton Conf. on Inf. Sciences and Systems* 4(1970)，345-349〕已经提议修改树查找和插入方法，即算法 6.2.2T，他们尽量让所有插入都对那些与其父亲节点属于相同页的节点进行；如果

该页已满，则只要可能，就开始一个新的页。如果页数无限，且如果数据以随机的次序到达，则可以证明，平均页存取次数近似为 $H_N/(H_m-1)$，仅比我们在最好的 m 叉树中所得到的稍微多一点（见习题 10）。

B 树 1970 年 R. 拜尔和 E. 麦克里特已经发现了借助多路树分枝研究外部查找的一个新方法〔*Acta Informatica* (1972), 173-189〕，而且大约在同一时间 M. 考夫曼（M. Kaufman）也独立地发现了同一方法〔未发表〕。他们的思想，是以一类称为 B 树的功能很强的新型数据结构为基础的，它使得在最坏的情况下，仍有可能利用比较简单的算法，以"有保证的"效率来查找和更新一个大型文件。

阶 m 的一株 B 树，是满足下列性质的一株树：

i）每个节点的儿子个数 $\leqslant m$。

ii）除了根和叶之外，每个节点的儿子个数 $\geqslant m/2$。

iii）根至少有两个儿子（除非它是一片叶）。

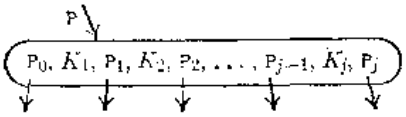
iv）所有叶都出现在同一级上，而且不带信息。

v）具有 k 个儿子的非叶节点含有 $k-1$ 个键。（通常，一片"叶"是一个终端节点，它没有儿子。因为叶不带有信息，故我们可以把它们看做实际上不在树内的外部节点，因而 A 是指向一片叶的指针。）

图30示出阶 7 的一株 B 树，每个节点（除根和叶外）都有 $\lceil 7/2 \rceil$ 到 7 之间个儿子，所以它包含 3、4、5 或 6 个键。允许根节点包含 1 到 6 个键；在现在情况下，它有 2 个键。所有的叶都在级 3 上。注意，（a）键从左到右以递增的次序出现，这是对称次序概念的一种自然的推广；（b）叶的片数恰比键的个数大 1。

阶 1 或 2 的 B 树显然是没有兴趣的，所以我们仅仅考虑 $m \geqslant 3$ 的情形。注意，在接近 6.2.3 节末尾处定义的"2-3"树是阶 3 的 B 树，反过来，阶 3 的 B 树是一株 2-3 树。

包含 j 个键和 $j+1$ 个指针的一个节点可以表示作



（1）

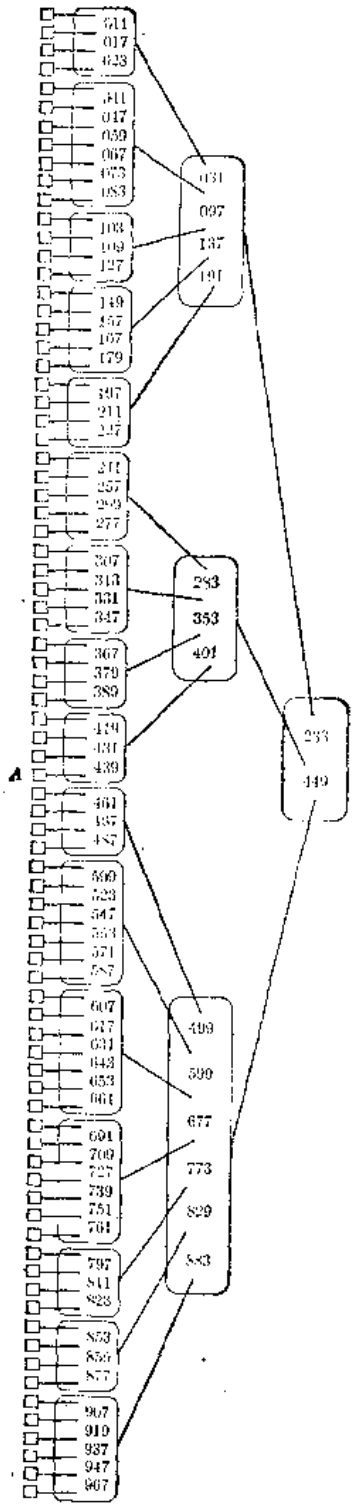


图30 阶为 7 的一株 B 树，所有叶都在第 3 级上

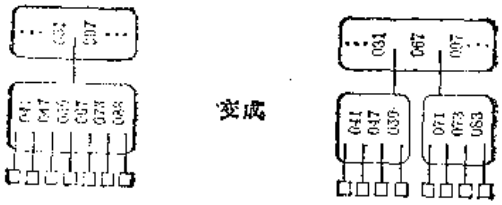
其中 $K_1<K_2<\cdots<K_j$，P_i 指向包含 K_i 和 K_{i+1} 之间的键的子树。因此在一株 B 树中的查找是十分直截了当的：在节点（1）已经进入内存之后，在键 K_1，K_2，…，K_j 当中查

找给定的变元（当 j 很大时，大概作一个二分查找；但当 j 很小时，顺序查找是最好的）。如果查找成功，则我们已经找到了所希望的键；但如果由于变元位于 K_j 和 K_{j+1} 之间而使查找不成功，则取由 P_j 指出的节点，并继续这个过程。如果变元小于 K_1，则使用指针 P_0，如果变元大于 K_j 则使用 P_j。如果 $P_j=\Lambda$，则查找是不成功的。

B 树的优点是，插入也十分简单。例如，考虑图 30，每片叶对应一个新的插入可能发生的位置。如果我们要插入新的键 337，则只需把适当的节点从

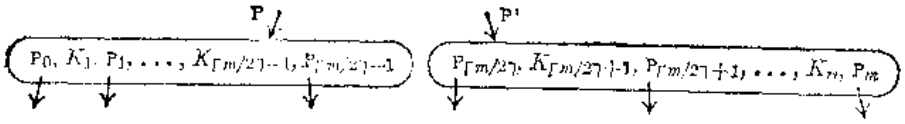
（2）

另一方面，如果我们要插入新的键 071，就没有空间了，因为在级 2 上的对应节点已经“满”了。这种情况可以通过把节点分成两部分来处理，每部分有三个键，并把中间的键送到级 1 上：



（3）

一般说来，如果我们要把一新项目插入到阶为 m 的 B 树中去，当所有叶都在级 l 上时，则把新的键插入到级 $l-1$。如果该节点现在含 m 个键，使得它有形式（1），且 $j=m$，则把它分成两个节点



（4）

并且把键 $K\lceil m/2\rceil$ 插入到原来节点的父亲处（于是，父亲节点中的指针 P 为序列 P，$K\lceil m/2\rceil$，P′ 所代替）。这个插入可能引起父亲节点包含 m 个键，而且如果这样，则它应以同样的方式分裂（参考图 27，它画出了 $m=3$ 的情况）。如果我们需要分裂根节点，根节点是没有父亲的，则只需建立包含单个键 $K\lceil m/2\rceil$ 的一个新根节点，在这种情况下这株树长高了一级。

这个插入步骤几乎保持 B 树的一切性质；为了鉴赏这一思想的巧妙性，读者应该作习题 1。注意，这株树或多或少地从顶部增长，而不是从下面减少，因为仅当根分裂时它的高度才增加。

从 B 树删去，仅仅比插入稍稍复杂点而已（见习题 7）。

关于性能的上限 现在让我们看看，当对阶为 m 的一株 B 树进行查找时，在最坏的情况下，有多少节点要被访问。假设有 N 个键，且 $N+1$ 片叶出现在 l 级上，则在级 1，2，3，…上的节点数至少是 2，$2\lceil m/2\rceil$，$2\lceil m/2\rceil^2$，…；因此

$$N+1 \geqslant 2\lceil m/2 \rceil^{l-1} \tag{5}$$

换句话说，

$$l \leqslant 1+\log_{\lceil m/2 \rceil}\left(\frac{N+1}{2}\right) \tag{6}$$

这意味着，例如，如果$N=1,999,998$且$m=199$，则l至多为3。由于我们在一次查找期间至多存取l个节点，故这个公式保证了运行时间十分小。

当插入一个新的键时，可能需要分裂多达l个节点。然而，需要分裂的节点的平均数要小得多。因为在整株树被构造时出现的分裂总数，恰好比树中内部节点总数少l。如果有p个内部节点，则至少有$1+(\lceil m/2 \rceil-1)(p-1)$个键；因此

$$p \leqslant 1+\frac{N-1}{\lceil m/2 \rceil-1} \tag{7}$$

由此得出，在构造一株N个键的树时，每作一次插入，我们需要分裂一个节点的次数，平均少于$1/(\lceil m/2 \rceil-1)$!

改进和变形 只要稍微突破常规，就有若干方法改进上面定义的基本B树结构。

首先，我们注意到，在$l-1$级节点中所有指针都是A，而其它级中的指针皆非A。这是值得注意的空间浪费。所以，我们可以通过消去所有的A和对每个"底"节点使用一个不同的m值以节省时间和空间。使用两个不同的m并不搅乱插入算法，因为被分裂的那个节点的两半，保留在和原来节点相同的级上。我们事实上可以通过要求在$l-i$级上的所有非根节点，都有$m_i/2$到m_i个儿子，来定义阶为m_1，m_2，m_3，…的一株广义B树；这样一株B树在每级上都有不同的m，而插入算法实质上仍和从前一样有效。

为把上一段中的这个思想进一步推广，我们可以在树的每一级上，使用一个完全不同的节点格式，而且我们也可以在叶中存储信息。有时，键仅构成一个文件中记录的很小部分，在这样的情况下，在靠近树根的分枝节点处存整个记录，乃是一个错误；这会使m对于有效的多路分枝说来太小了。

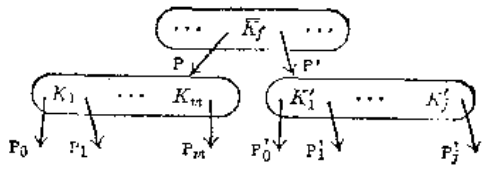
我们因此可以重新考虑图30，同时想象文件的所有记录现在都存在叶中，而且只有少量的键被复写在分枝节点中。在这种解释之下，最左的叶包含其键≤011的所有的记录；标志为A的叶包含其键满足$439<K\leqslant 449$的所有记录；等等。在这种解释之下，叶节点的增长和分裂就如同分枝节点那样，只是一个记录决不会从一片叶传到下一级去。于是，诸叶的容量至少总是填得半满的。每当一片叶分裂时，一个新的键便进入树的非叶部分。如果每片叶链接到它在对称次序下的后继，则我们就获得了以有效和方便的方式，既是顺序地，又是随机地来遍历文件的能力。

S．P．戈斯和M．E．森科（M．E．Senko）〔*JACM* 16(1969)，569-579〕的某些计算提示，把叶做得相当大，比如说大到大约10个连续的页那样长，可能是一个好的想法。通过对每片叶在已知的键范围内的线性内插，我们可以猜测哪10个叶大概包含一个给定的查找变元。如果猜测是错误的，则我们就损失了时间，但是经验指出，这个损失比我们通过减少树的大小所节省的时间要小。

T．H．马丁〔未发表〕指出，奠定B树的基础的思想，也可以用于可变长的键。我们不必对每个节点的儿子设置$[m/2,\ m]$的界限，而只规定每个节点中装载的数据应该至

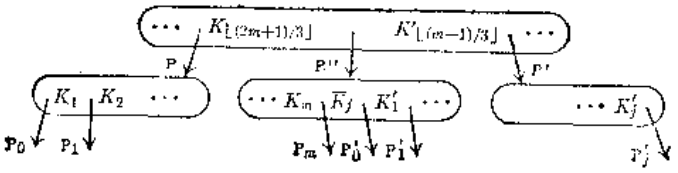
少是半满的。尽管每个节点中键的确切数目依赖于键的长短，插入和分开的机制仍然工作得很好。然而，不应该允许键过长，否则它们可能会碍事的（见习题5）。

对于基本B树的另一个重要的修改是拜尔和麦克里特提出的“溢出”思想。这个思想尽可能避免分裂节点，而代之以使用一个局部转动，来改进插入算法。假设有一个节点，它由于包含m个键和$m+1$个指针而过满了；我们不去分裂它，而是首先在右边考察它的兄弟节点，这个兄弟节点比如说有j个键和$j+1$个指针。在父亲节点中，有一个键$\overline{K}_i$，它按下图分开这两个兄弟的键



（8）

如果$j<m-1$，则一个简单的重新排列就使分裂成为不必要的了：我们在左节点中保留$\lfloor(m+j)/2\rfloor$个键，在父亲节点中以$\overline{K}_{\lfloor(m+j)/2\rfloor+1}$来代替$\overline{K}_i$，并把剩下的$\lceil(m+j)/2\rceil$个键（包括$\overline{K}_i$在内）以及对应的指针放置到右节点中。于是这个满了的节点就“流动”到它的兄弟节点处。另一方面，如果兄弟节点已经满了（$j=m-1$），则我们可以把这两个节点分裂为三个节点，其中每个约装满三分之二，分别包含$\lfloor(2m-2)/2\rfloor$、$\lfloor(2m-1)/3\rfloor$以及$\lfloor 2m/3\rfloor$个键：



（9）

如果原来的节点没有右兄弟，则我们实际上可以同样的方式考虑它的左兄弟（如果原来的节点既有右兄弟又有左兄弟，则我们完全可以不分裂出一个新节点，除非左兄弟和右兄弟两者全都是满的）。最后，如果有待分裂的原来的节点全然没有兄弟，则它必然是根；我们可以改变B树的定义，允许根包含$2\lfloor(2m-2)/3\rfloor$那么多个键，使得当根分裂时，产生每个有$\lfloor(2m-2)/3\rfloor$个键的两个节点。

上一段中所有技术的效果，是产生一类优良的树，比如说，阶为m的一株B^*树，可以定义如下：

i）除了根以外每个节点至多有m个儿子。

ii）每个节点，除了根和叶以外，有$\geqslant(2m-1)/3$个儿子。

iii）根至少有两个和至多有$2\lfloor(2m-2)/3\rfloor+1$个儿子。

iv）所有的叶都出现在同一级上。

v）具有K个儿子的一个非叶节点，包含$K-1$个键。

重要的变化是条件（ii），它断言，我们至少利用了每个节点中三分之二的可用空间。这个变化不仅有效地使用了空间，而且也使查找过程加快，因为在（6）和（7）中我们可以用“$\lceil(2m-1)/3\rceil$”代替“$\lceil m/2\rceil$”。

由于在B树中的叉数可以低到2，也许读者已经发生怀疑了：为什么我们要花费整整

一次盘访问于仅仅一个 2 路判定上呢?! 一个简单的缓冲方案，即所谓“最近最小使用的页替换”，可以消除这种反对意见；我们可以在内存中保持若干个缓冲区的信息负载，使得当对应的叶已经出现时可以避免输入命令。在这个方案中，查找或插入算法发出“虚拟的读”命令，仅当必要的叶不在内存中时，它才被翻译成真正的输入指令；当这缓冲区已经被读过而且可能已被算法所修改时，发出随继的“释放”命令。当需要一个真正的读入时，便选择最早释放的缓冲区；如果该缓冲区的内容在他们被读入以后发生了变化，则我们写出该缓冲区，然后就把所希望的页读入到这个选定的缓冲区中。

由于树中的级数相对于缓冲区数来说一般地较小，这个分页的方案将确保根页总在内存中；而且如果根只有 2 个或 3 个儿子，则第一级的页面大概也将驻留于内存中。可以加上某些特殊的机制，以确保至少有某一数量的接近于根的页总存在。注意，最近最少使用的方案，意味着在一个插入期间可能需要分裂的页，当需要时，便自动地在内存中了。

E. 麦克里特的某些经验表明，这个思想是十分成功的。例如，他发现，对于 10 页的缓冲区和 $m=\lceil 2\rceil$，以递增次序插入 100,000 个键的过程仅需要 22 条实际的读命令和 857 条实际的写命令；于是，大多数活动都在内存中进行。而且这株树仅仅包含 835 个节点，只比极小可能的值 $\lceil 100000/(m-1)\rceil=834$ 大 1；因此存储利用接近百分之百。对于这个实验，他使用了溢出技术，但如同（4）中那样仅通过两路节点分裂，而不是象在（9）中那样的 3 路分裂（见习题 3）。

在另一个实验中，再次通过 10 个缓冲区和 $m=\lceil 2\rceil$ 以及溢出技术，他以随机次序把 5000 个键插入到初始为空的一株树中。在作了 2762 个实际的读和 2739 个实际的写之后，便产生了具有 48 个节点的一株 2 级树（百分之 87 的存储利用）。随后的 1000 个随机查找需要 786 个实际的读。在作了 2743 个实际的读和 2800 个实际的写之后，没有溢出功能的同样实验，产生了具有 62 个节点的一株 2 级树（百分之 67 的存储利用）；1000 个相继的随机查找需要 836 次实际的读。这不仅表明分页方案是有效的，而且表明在判定分裂一个节点之前，局部地处理溢出是明智的。

姚期智已经证明，对于很大的 N 和 m，在没有溢出功能的随机插入之后的平均节点数将是 $N/(m\ln 2)+O(N/m^2)$，所以存储利用将近似为 $\ln 2\approx 69.3\%$（*Acta Informatica*, 9 (1978), 171-181）。

如果适当地实现这些算法，许多独立的用户有可能同时存取和更新一个大的 B 树的不同部分而不会出现“死锁”，见 B. Samadi, *Inf. Proc. Letters* 5(1976), 107-112。

习题

1. 〔10〕 在把键 613 插入到图 30 中之后，得到什么样的阶为 7 的 B 树?（请勿用“溢出”技术。）

2. 〔15〕 做习题 1，但使用溢出技术和如同（9）中那样的 3 路分裂。

▶3. 〔28〕 假如我们以递增的次序把键 1，2，3，…插入到初始为空的阶为 101 的 B 树中，（a）当不使用溢出时；（b）当使用溢出和如（4）那样的仅仅 2 路的分裂时；（c）当使用一株阶为 101 的 B^* 树、溢出以及如同（9）中的 3 路分裂时；问哪一个键首先引起诸叶出现在级 4 上?

4.〔21〕（拜尔和麦克里特）说明，怎样插入到一株广义的B树中，使得除根和叶之外的所有节点都保证至少有$\frac{3}{4}m-\frac{1}{2}$个儿子。

5.〔21〕假设某个节点表示1000个字符的外存位置，如果每个指针花费5个字符，且键是可变长的，其长度在5到50个字符之间，但总是5的倍数，则在一次插入期间，分裂一个节点之后，该节点中至少有多少字符位置被占用（只考虑类似于正文中对固定长的B树所述的那种简单的分裂过程，而不使用"溢出"；放过使剩下的两部分最接近相等的键。

6.〔22〕如过通过位置，而不是通过键的值来查找一个线性表的项目，则B树的思想是否仍然有效（参考算法6.2.3B）？

7.〔22〕试设计B树的一个删去算法。

8.〔23〕试设计B树的一个连接算法（参考6.2.3节）。

9.〔30〕试讨论，一个组织成如一株B树那样的大型文件，怎样以不同页的用户很少彼此干扰的方式，同时为大量用户作多路存取和更新？

10.〔HM37〕考虑芒茨和乌热加里斯所建议的树插入的推广，其中每页可保持M个键。在把N个随机项目插入到这样一株树中，使得它有$M+1$个外部节点之后，设$b_{Nk}^{(j)}$是具有如下性质的一次不成功查找的概率，该查找需要k次页访问，并结束于一个外部节点，该节点的父亲节点属于一个具有j个键的页，如果$B_N^{(j)}(z)=\sum b_{Nk}^{(j)}z^k$是对应的母函数，试证明

$$B_1^{(j)}(z)=\delta_{j1}z$$

$$B_N^{(j)}(z)=\frac{N-j-1}{N+1}B_{N-1}^{(j)}(z)+\frac{j+1}{N+1}B_{N-1}^{(j-1)}(z),\quad 对于\ 1<j<M$$

$$B_N^{(1)}(z)=\frac{N-2}{N+1}B_{N-1}^{(1)}(z)+\frac{2z}{N+1}B_{N-1}^{(M)}(z)$$

$$B_N^{(M)}(z)=\frac{N-1}{N+1}B_{N-1}^{(M)}(z)+\frac{M+1}{N+1}B_{N-1}^{(M-1)}(z)$$

试求每次不成功查找的平均页访问次数$C'_N=\sum_{1\leqslant j\leqslant M}B_N^{(j)}(1)$的渐近特性。〔提示：借助矩阵

$$W(z)=\begin{pmatrix} -3 & 0 & \cdots & 0 & 2z \\ 3 & -4 & \cdots & 0 & 0 \\ 0 & 4 & \cdots & 0 & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & -M-1 & 0 \\ 0 & 0 & \cdots & M+1 & -2 \end{pmatrix}$$

来表达这个递归式，并把C'_N同$W(1)$中的N次多项式联系起来。〕

6.3 数字查找

我们可以不把一个查找方法建立在键之间比较的基础上，而是利用它们的数字序列或字母字符序列的表示。例如，考虑一部大型字典中的"书边标目"；由一个给定的字的第一个字母，我们可以立即定出包含所有以该字母开始的字的那些页的位置。

如果我们推广书边标日的思想，则它的逻辑结论之一就是如表 1 所说明的以重复“下标”为基础的一个查找方案。假设我们要检验一个给定的查找变元，看它是否是英语中最普通的 31 个字之一（参考 6.2.2 节中的图 12 和 13），这个数据在表 1 中被表示成所谓的“trie”(检索)结构，这个名称是由 E. 弗雷德金（E. Fredkin）〔*CACM* 3(1960)，490-500〕提议的，因为它是信息·检索（*re trie val*）的一部分。一个检索结构实际上是一株 M 叉树，它的节点是 M 维向量，其分量对应于数字或字符，第 l 级上的每个节点表示以 l 个字符的某个序列开始的所有键的集合；这个节点根据第 $l+1$ 个字符确定一个 M 路分枝。

表 1　31个最普通的英文字的一个检索结构

	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
U	—	A	—	—	—	I	—	—	—	—	HE	—
A	(2)	—	—	—	(10)	—	—	—	WAS	—	—	THAT
B	(3)	—	—	—	—	—	—	—	—	—	—	—
C	—	—	—	—	—	—	—	—	—	—	—	—
D	—	—	—	—	—	—	—	—	—	HAD	—	—
E	—	—	BE	—	(11)	—	—	—	—	—	—	THE
F	(4)	—	—	—	—	—	OF	—	—	—	—	—
G	—	—	—	—	—	—	—	—	—	—	—	—
H	(5)	—	—	—	—	—	—	(12)	WHICH	—	—	—
I	(6)	—	—	—	HIS	—	—	—	WITH	—	—	THIS
Θ	—	—	—	—	—	—	—	—	—	—	—	—
J	—	—	—	—	—	—	—	—	—	—	—	—
K	—	—	—	—	—	—	—	—	—	—	—	—
L	—	—	—	—	—	—	—	—	—	—	—	—
M	—	—	—	—	—	—	—	—	—	—	—	—
N	NOT	AND	—	—	—	IN	ON	—	—	—	—	—
O	(7)	—	—	FOR	—	—	—	TO	—	—	—	—
P	—	—	—	—	—	—	—	—	—	—	—	—
Q	—	—	—	—	—	—	—	—	—	—	—	—
R	—	ARE	—	FROM	—	—	OR	—	—	—	HER	—
Φ	—	—	—	—	—	—	—	—	—	—	—	—
Π	—	—	—	—	—	—	—	—	—	—	—	—
S	—	AS	—	—	—	IS	—	—	—	—	—	—
T	(8)	AT	—	—	—	IT	—	—	—	—	—	—
U	—	—	BUT	—	—	—	—	—	—	—	—	—
V	—	—	—	—	—	—	—	—	—	HAVE	—	—
W	(9)	—	—	—	—	—	—	—	—	—	—	—
X	—	—	—	—	—	—	—	—	—	—	—	—
Y	YOU	—	BY	—	—	—	—	—	—	—	—	—
Z	—	—	—	—	—	—	—	—	—	—	—	—

例如，表 1 的检索结构有 12 个节点；节点（1）是根，我们在这里考察头一个字符。如果第一个字符是N，则这张表告诉我们，我们的字必然是 NOT（否则它不在表中）。另一方面，如果第一个字母是W，则节点（1）告诉我们转到节点（9）；以同一方式考察第二个字母，节点（9）指出第二个字母应该是A、H或I。

表 1 中节点向量是按照 MIX 字符代码重新排列过的。这意味着一个检索结构的查找将是十分快的，因为我们仅仅通过使用我们的键的字符作为下标，来取出一个阵列中的字。通过下标进行快速多路判断的技术，称为“检表”（Table Look-At），以示区别于“查表”（Table Look-Up）〔见 P. M. Sherman，*CACM* 4 (1961)，172-173，175〕。

算法 T（检索结构查找） 给定一个形成M叉检索结构的记录表，本算法查找一个给定的变元K。这个检索结构的节点，都是其下标从 0 变到$M-1$的向量；这些向量的每一分量或者是一个键或者是一个链接（可能是空的）。

T1.〔初始化〕 置链接变量 P，使得它指向检索结构的根。

T2.〔分枝〕 置k为输入变元K的从左到右的下一个字符（如果这个变元已经完全扫描过了，则置k为一个“空白”或字结束符号。字符应该表示为在范围$0 \leqslant k < M$中的数）。设X为 NODE(P) 中编号为k的表项目，如果X是一个链接，则转到 T3；但如果X是一个键，则转到 T4。

T3.〔前进〕 如果$X \neq \Lambda$，则置$P \leftarrow X$并返回步骤 T2；否则算法以失败告终。

T4.〔比较〕 如果$K = X$，则算法成功地结束，否则它以失败告终。

注意，如果这个查找是不成功的，则已经找到了最长的匹配，这个性质在应用上有时是有用的。

为了比较这个算法和这一章中其它算法的速度，我们可以写出一个短的 MIX 程序，同时假定字符都是字节而且键的长度至多是五个字节。

程序 T（检索结构的查找） 这个程序假定所有键都表示成一个 MIX 字，当键少于五个字符时，空白字符在右边。由于我们使用 MIX 字符代码，因此假定查找变元的每个字节都包含小于 30 的一个数。链接被表示作一个节点字的 0:2 字段中的一个负数。rI1=P，rX=K的未扫描部分。

01	START	LDX	K	1	*T*1. 初始化
02		ENT1	ROOT	1	P←指向检索结构的根的指针
03	2H	SLAX	1	*C*	*T*2. 分枝
04		STA	* + 1 (2:2)	*C*	摘出下一个字符k
05		ENT2	0,I	*C*	Q←P + k
06		LDIN	0,2(0:2)	*C*	P←LINK(Q)
07		JIP	2B	*C*	*T*3. 前进。如果P是一个≠Λ的链接则转 T2
08		LDA	0,2	1	*T*4. 比较。rA←KEY(Q)
09		CMPA	K	1	
10		JE	SUCCESS	1	如果rA = K，则成功地转出
11	FAILURE	EQU	*		如果不在检索结构中则转出

这个程序的运行时间是 $8C+8$ 个单位，其中C是考察的字符数。由于$C \leqslant 5$，故这个查找决不花费多于 48 个时间单位。

如果我们现在要比较这个程序（使用表 1 的检索结构）同程序 6.2.2T（使用图 13 的最优二分查找树）的效率，则可以作如下的观察：

1．检索结构花费更多的内存空间；我们使用360个字来表示仅仅31个键，而二分查找树只使用62个内存字（然而，习题 4 表明，通过省去某些无用的位置，实际上可以把表 1 的检索结构放在仅仅 49 个字中）。

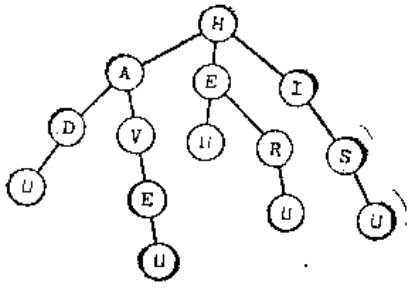
2．对两个程序来说，每次成功的查找都花费 26 个时间单位。但是一次不成功的查找在检索结构中将进行得更快些，而在二分查找树中则要慢些。对于这个数据来说查找的不

成功次数多于成功的次数，所以从速度观点看，检索结构是更可取的。

3．如果我们应用的对象是图 15 的 KWIC 索引而不是 31 个最普通的英文字，则由于这个数据的特性，检索结构就失去它的优越性了。例如，一个检索结构为了区别 COMPUTATION 和 COMPUTATIONS 竟需要作 12 次迭代(在这种情况下，若换一种方式构造检索结构，使它从右到左而不是从左到右扫描字，则倒更好些)。

检索结构存储的思想，是首先由雷内·德·拉·布赖恩戴斯(Rene. de. la Braindais)发表的〔*Proc. Western Joint Computer Conf.* 15(1959)，295-298〕。他指出，如果我们对每个节点向量都使用一个链接表，则可以以运行时间为代价来节省内存空间，因为大多数向量的元素趋向于空。实质上，这个思想在于通过图 31 中所示的森林来代替表 1 的检索结构。在这样一个森林中，查找是通过如下方式进行的：首先找到与第一个字符匹配的根，然后找到与第二个字符匹配的该根的儿子根，等等。

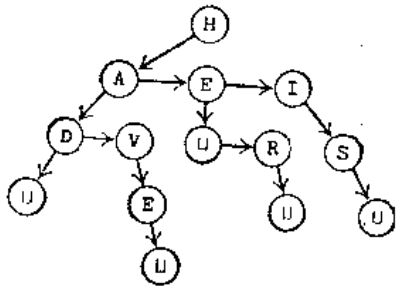
在这篇论文中，德·拉·布赖恩戴斯并不完全象表 1 或图31中所示的那样停止树的分叉；而是继续一个字符挨一个字符地表示每个键，直到达到字结束的限定符为止。于是，他实际上使用了



(1)

来代替图 31 中的“H”树。这个表示要求更多的存储，但是它使得可变长数据的处理特别容易。如果我们对每个字符使用两个链接场，则动态插入和删去即可以用一种简单的方式来处理。此外，还有许多应用，在这些应用中查找变元以一种“未压缩过的形式”，即以每个字放一个字符的形式出现，而且这样一株树无须在进行查找之前先压缩数据。

如果我们使用通常的方法把树表示作二叉树，则（1）就变成二叉树



(2)

(在图 31 的完全森林表示中，我们也应有一个从H向右引到其邻近根 I 的一个指针。)在这株二叉树中进行查找时，我们把变元中的一个字符同树中的字符加以比较，沿着诸 RLINK 直到找到一个匹配为止；然后取 LLINK 并以同一方式来处理变元的下一个字符。

对于这样一株二叉树，我们或多或少地通过比较法进行查找，用相等-不相等的分枝代替了小-大的分枝。6.2.1节的初等理论告诉我们，为了区别N个键，平均说来，必须

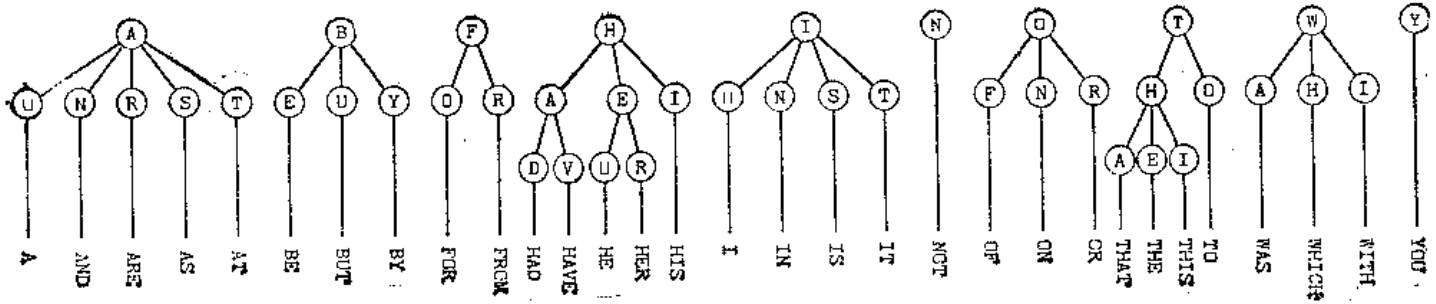


图31　表1的检索结构转换为一片森林

至少作 $\log_2 N$ 次比较；当查找象图 31 那样的一株树时，所作的平均查找次数，至少必须同使用 6.2 节的技术进行一次二分查找时所作的平均检验次数一般多。

另一方面，表 1 中的检索结构一次就能进行整个M路分枝；我们将看到，如果输入数据是随机的，则对于很大的N的平均查找时间，大约只要 $\log_M N=\log_2 N/\log_2 M$ 次迭代。我们还将看到，类似算法 T 中的一个“纯粹”检索结构方案，需要总数约$N/\ln N$个节点对N个随机输入进行区别；因此总共的空间数量同 $MN/\ln M$成正比。

从这些考虑显然可看出，检索结构的思想仅在树的头几级中有益。把两种策略混合使用，即对头一些字符使用检索结构，而后又转到某种其它的技术上，我们可以获得更好的性能。例如，小 E. H. 萨森古思(E. H. Sussenguth), Jr.〔*CACM* 6(1963), 272-279〕曾提议把逐个字符的方案一直用到树的那样的部分；其中，比如说，文件中的键至多是六个。然后，我们就能顺序地扫视剩下的键的短表。我们将看到，这个混合策略能减少约六分之一的检索结构节点数，而运行时间并没有很大的变化。

对于英语的一种应用 基本的检索结构和字符查找策略，有许许多多变形。为了了解其中的某些策略，我们考虑一种设想的大型应用：假设我们要在计算机的内存中存储一部相当完备的英语字典。当然，为了这一目的，将需要一个相当大的内存，比如说，50000 字。我们的目标是要找出一种紧凑的方法来表示字典，而且还要保持相当快的查找。

这显然不是一个简单的任务；可以想象得到的是：它既要求我们充分了解字典内容，又要求我们有相当的程序设计技巧。现在，让我们想象我们正面临着这个重大的课题。

一部典型的大学字典包含100000字以上；这比这里所考虑的还要大些，但是看一看这样的字典，就使我们知道一些大致会出现的情况。如果尝试应用检索结构存储方法，则我们立即会注意到，可以作出重大的简化。例如，假设不考虑专有名字和略写，则如果一个字的第一个字母是 b，则第二个字母一定不会是字符 b、c、d、f、g、j、k、m、n、p、q、s、t、v、w、x 或 z 中的任何一个（字 bdellium 除外，可以把它从我们的字典中排除掉！）。事实上，在以 c、d、f、g、h、j、k、l、m、n、p、q、r、t、v、w、x、z 开头的字中，也排除以上 17 种作为第二个字母的可能性，但以 ct、cz、dw、fj、gn、mn、nt、pf、pn、pt、tc、tm、ts、tz、zw开头的一些字，以及以 kn、ps、tw 开始的相当数量的字例外。利用这一事实的一种方法，是对字母进行编码（例如，有一张 26 个字的表且执行等价于“LD1 TABLE, 1”的动作），使得上面所列的辅音字母 b, c, d, …, z 都被转换为大于剩下字母 a、e、h、i、l、o、r、u、y 数值码的一种特殊表示。这样一来，一个检索结构的许多节点都可以被缩短成为一个 9 路分枝，而对于很少出现的例外字母使用另一个“换档”的单元。这将在英语检索结构的许多部分节省内存空间，而不仅是节省第二个字母的位置。

两个字母的 $26^2=676$ 种组合中，实际上只有 309 种出现在一部典型大学字典中字的首部；而且，在这 309 对字母中，有 88 对是 15 个或少于 15 个字的开始字母（这 88 个稀少的对的典型例子，是 aa, ah, aj, ak, ao, aq, ay, az, bd, bh, …, xr, yc, yi, yp, yt, yu, yw, za, zu, zw；大多数人都不能从这些类型中说出一个以上的字来）。当出现一个稀少类型时，我们可以从检索结构转移到一种类似于顺序查找的其它方案上去。

另一种降低一部字典存储要求的方法，是利用前缀。例如，如果我们要查一个以re-、

pre-、anti-、trans-、dis-、un-等开头的字，则可望去掉前缀而查这个字的剩下部分。这样一来，我们可以去掉如 reapply、recompute、redecorate、redesign、redeposit等许多字；但仍然需要保留如 remainder、requirement、retain、remove、readily等字，这是因为当去掉前缀“re-”之后，它们的意义就不易推断了。于是，我们应该首先查出这个字，仅当第一次查找失败时，才考虑删去前缀。

后缀的使用甚至比前缀更为重要。让每个名词和动词都出现两次，即以单数形式和复数形式同时出现，这肯定地是浪费的；而且还有许多其它类型的后缀。例如，下列的结尾可以加到许多动词词干上，来做成一类相关的字：

-e	-es	-ing
-ed	-s	-ings
-edly	-able	-ingly
-er	-ible	-ion
-or	-ably	-ions
-ers	-ibly	-ional
-ors	-ability	-ionally
	-abilities	

（这些后缀中有许多本身又是由后缀组成的）如果我们应用这些后缀于词干

comput-
calculat-
search-
suffix-
translat-
interpret-
confus-

上，即可组成许许多多的字；这大大地增加了字典的容量。当然，也形成许多不是字的东西，例如“compution”，词干 computat 似乎也和 comput 一样是必要的。但这不会引起问题，因为这样的组合无论如何不会出现在输入中；而如果某个作者杜撰 computedly 一字，则我们有已经为他准备好的翻译。注意，尽管“confusability”这个词只出现少数字典中；但大多数人都能理解它；我们的字典将给出一个适当的解释。如果正确地处理了前缀和后缀，则我们的字典甚至能够只通过给出动词词干“establish”来导出“antidisestablishmentarianism”的意义。

当然，必须注意，每个字的意义是由其词干和后缀适当地确定的；有些字不是这样，这些例外的情况应当加进字典当中，使得在我们试图查一个后缀之前，就能找到它们。例如，“socialism”和“socialist”之间的类似性，完全不象“organism”和“organist”之间那样！

这些都是可用来减少所需内存数量的技巧。但是，在一个简单的系统中，我们怎样紧凑地表示诸方法的这种“大杂烩”呢？答案是，把字典想象为以特殊机器语言为特殊的解释系统写成的程序(参考1.4.3节)；一个检索结构的每个节点内的诸项，都可以想象作一组指令。例如，在表1中我们有两类“指令”，而且程序下使用符号位作为“操作码”；负号意味着转移到另一个节点，并前进到下一个字符去执行下一条指令，而正号表示变元同

一个特定的键作比较。

就我们的字典应用说来，在解释语言中可以有下列类型的操作码：

• Test n，α，β "如果变元的下一个字符的编码值为 $k \leqslant n$，则转到单元 $\alpha+k$ 以寻找下一条指令；否则转到单元 β"。

• Compare n，α，β "把变元中剩下的字符同存储在单元 $\alpha, \alpha+1, \cdots, \alpha+n-1$ 中的几个字比较，如果在 $\alpha+k$ 单元中找到一个匹配，则查找以'意义' $\beta+k$ 成功地结束。但如果未找到匹配，则以失败告终"。

• split α，β "扫描到这一点的字，是一个可能的前缀或词干。转到单元 α 去执行下一条指令继续查找。如果该查找是不成功的，则通过查找该变元中剩下的字符，犹如它们是一个新变元那样继续进行查找。如果这第二次查找是成功的，则把找到的这个"意义"同'意义 β'组合在一起"。

检验操作（Test）实际上是检索结构概念，而比较操作（compare）则表示顺序查找的一种变形。分裂操作（split）既处理前缀，又处理后缀。若干其它操作，可以根据英语的进一步特性来想象。还可能从每条指令中消去参数 β，以便进一步节省内存空间，因为内存可安排成使得在每种情况下 β 都为 α、n 所隐含，或隐含在指令的单元中。

关于字典组织的其它内容，见 Sydney M. Lamb and William H. Jacobsen, Jr., *Mechanical Translation* 6 (1961), 76–107; Eugene S. Schwartz, *JACM* 10 (1963), 413–439; E. J. Galli and H. Yamada, *IBM Systems J.* 6 (1967), 192–207 等等有兴趣的论文。

二进的情形 现在考虑 $M=2$ 的特殊情况，在这种情况下，我们每次扫描查找变元的一个二进位。已经提出了两种有趣的方法，它们特别适合于这一情况。

第一种方法，我们将称之为数字树查找，它是由 E. G. 科夫曼（E. G. Coffman）和 J. 伊夫（J. Eve）〔*CACM* 13 (1970), 427–432, 436〕给出的。这个思想同 6.2.2 节树查找算法中所做的完全一样，在该节点中存储全键，但使用该变元的二进位（而不是用比较的结果）来支配在每步中是取左分枝还是取右分枝。图 32 表示，当我们以递减频率的次序插入 31 个最常用英文字时，用这种方法所构成的树。为了给这个图提供二进数据先把这些字表达成 MIX 的字符代码然后再把字符代码转换成每个字节 5 个二进位的二进数。例如，字 WHICH 表示为"11010 01000 01001 00011 01000"。

为了在图 32 中查找 WHICH 这个字，我们首先把它同该树根处的 THE 进行比较。因为没有匹配，且 WHICH 的第一个二进位是 1，我们向右移而同 OF 作比较。因为还不匹配，且因 WHICH 的第二个二进位是 1，我们向右移，而同 WITH 进行比较，等等。

把图 32 和 6.2.2 节中的图 12 作一对照是有趣的，因为后一株树是以同一方式形成的，而且是用比较而不是用用于转移的键二进位形成的。如果我们考虑给定的频率，则图32的数字查找树要求对每次成功的查找平均进行3.42次比较；这比图12所需要的4.04次比较稍微好些，尽管每次比较花费的时间可能不同。

算法 D（数字树查找） 给定一个记录表，它形成如上所述的一株二叉树。本算法查找给定的变元 K，如果 K 不在表中，则把包含有 K 的新节点插入这株树的适当位置中。

本算法假定，这株树是非空的且其节点如算法 6.2.2T 中那样，有 KEY、LLINK 和

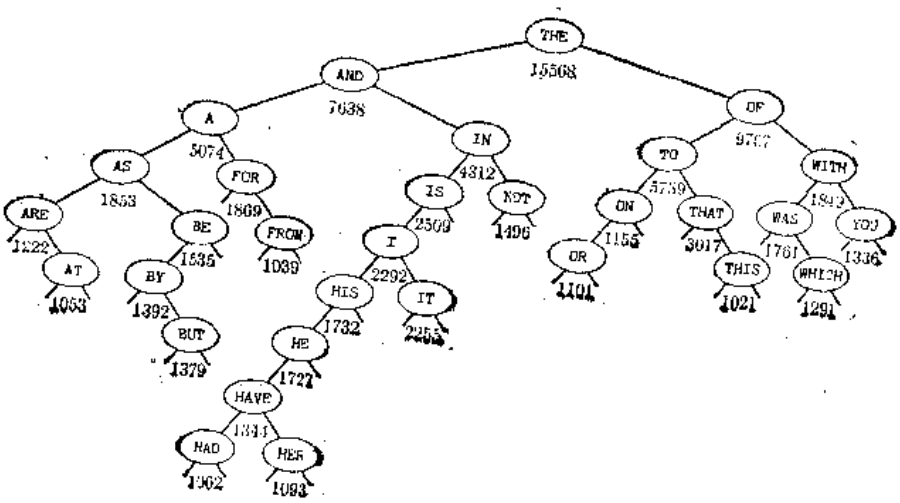


图32 对于以递减频率次序插入的31个最常用英语单字的一株数字查找树

RLINK 场。事实上，读者可以验证这两个算法几乎是相同的。

D1.〔初始化〕 置P←ROOT，且$K1 \leftarrow K$。

D2.〔比较〕 如果K=KEY(P)，则这个查找成功地结束，否则置b为$K1$的第一个二进位，并且把$K1$左移一位（由此删去该二进位并在右边引入一个0）。如果$b=0$，则转到D3，否则转到D4。

D3.〔左移〕 如果LLINK(P)≠Λ，则置P←LLINK(P)并转回到D2，否则转到D5。

D4.〔右移〕 如果RLINK(P)≠Λ，则置P←RLINK(P)并转回到D2。

D5.〔插入树中〕 置Q⇐AVAIL，KEY(Q)←K，LLINK(Q)←RLINK(Q)←Λ。如果$b=0$则置LLINK(P)←Q，否则置RLINK(P)←Q。

尽管算法6.2.2T的树查找本质上是二进的，但不难看出，现在的算法都可以被扩充成对任何$M \geqslant 2$的一个M叉数字查找（见习题13）。

唐纳德·R·莫里森（Donald R. Morrison）〔*JACM* 15（1968），514-534〕已经发现一个非常好的方法形成以键的二进表示为基础的N节点查找树，而无须在节点中存储键。他的方法，称为"Patricia"（是Pratical Algorithm To Retrieve Information Coded In Alphanumeric字头拼写成的）●，特别适合于处理极其长的、可变长的键，例如存储在一个大型文件中的标题或短句。

Patricia的基本思想是要构造一个二进的检索结构，但在每个节点中都包含了在作下一个检验之前可以跳过的二进位的位数，以避免单路分枝。有若干方法来说明这一思想，也许最便于说明的要算图33中所示的了。我们有一个二进位的TEXT阵列，它通常都十分冗长；它可以作为一个外部直接存取文件存储，因为每次查找仅访问TEXT一次。有待存入我们表中的每个键，都由文本中的一个开始位置来确定，并且总可把它想象成要由这个开始位置起直至达到文本的末尾为止（Patricia不查找键和变元之间的严格相等性，而

● 意指检索按字母数字编码的信息的实用算法。——译注

是确定是否存在一个以变元开头的键)。

图 33 中所描述的情况包含 7 个键，在每个字处都开始一个键，即“THIS IS THE HOUSE THAT JACK BUILT”(这是杰克建造的房子)以及“IS THE HOUSE THAT JACK BUILT”(是杰克建造的房子吗)以及……以及“BUILT”(建造)。有一个重要的限制，即没有任何一个键可以是另一个键的前缀；如果我们以一个在别处都不出现的唯一的文本结束代码(在现在情况下是“·”)，来结束文本，则这个限制即可满足。同样的限制也隐含在算法 T 的检索结构中，在那里“⊔”是结束代码。

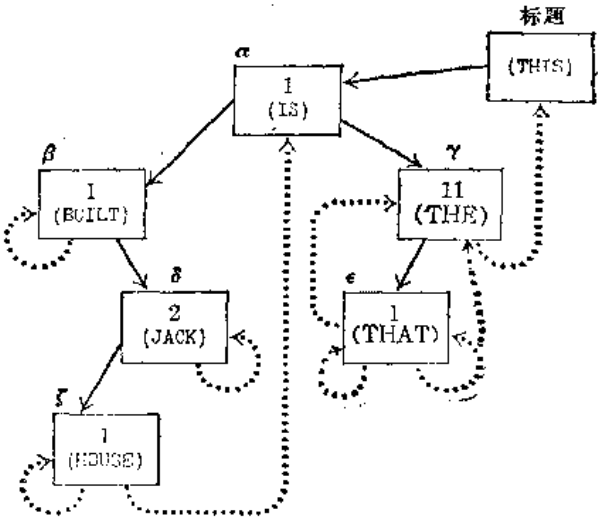


图33 Patricia树和TEXT的一例子

Patricia 用以查找的这株树，应当包含在随机存取存储器中，或被安排在 6.2.4 节所建议的一些页上。它由一个表头和$N-1$个节点所组成，这些节点都包含若干个场：

KEY 指向文本的一个指针。如果文本含C个字符，则这个场的长度必须至少有$\log_2 C$个二进位。在图 33 中，每个节点内所示的字，实际上都可由指向文本的指针来表示，例如，该节点不是包含“(JACK)”，而将包含数 24 (它指出“JACK BUILT”的开始位置)。

LLINK 和 RLINK 在树内的指针。这些场的长度至少必须是 $\log_2 N$ 个二进位。

LTAG 和 RTAG 两个单二进位的场，它们分别表示 LLINK 和 RLINK 是指向这个节点的儿子还是祖宗的指针。图 33 中的虚线对应于其 TAG 二进位为 1 的指针。

SKIP 如下所述，它是一个数，说明当查找时应跳过多少二进位。这个场应该足够大，使得如果在两个不同的键中出现相同的 k 位子串时，则场中应能容下最大的这样的 k；实际上，我们通常可以假定 k 不太大，如果它超过了 SKIP 场的大小，则可给出一个错误指示。SKIP 场在图 33 中表示为每个节点内的数。

表头仅含 KEY、LLINK 和 LTAG 场。

在 Patricia 树中的查找是如下进行的：假设我们正在查寻字 THE (二进位的型式为 10111 01000 00101)。我们从寻找根节点 α 的 SKIP 字段开始，它告诉我们检查这个变元的头一个二进位。该位是 1，所以向右移。下一个节点 γ 的 SKIP 字段告诉我们寻找这个变元的第 $1+11=12$位。该位为 0，所以左移。下一个节点 ϵ 的 SKIP 字段告诉我们寻找第 $(12+1)$ 位，该位是 1；现在找到 RTAG＝1，所以回到节点 γ，它让我们参考 TEXT。我们所走过的查找通路将对其二进位型式是 1 ×××× ××××× ×01…的任何变元出现，因此必须检验看看它是否匹配以该型式开始的唯一键。

另一方面，假设我们寻找一个或所有以 TH 开始的键。这个查找过程开始同上面所述的过程一样，但它最后试图查找10位变元的(不存在的)第12位。这时，我们在当前节点所确定的点处(在现在情况下为 γ)把变元同 TEXT 作比较；如果它不匹配，则此变元就

不是任何键的开头；但如果它匹配了，则此变元就是每一个由节点Y和其后裔中的虚线所表示的键的开头。

这个过程可以更精确地叙述如下：

算法P(*Patricia*) 给定一个TEXT阵列和一株具有如上所述KEY、LLINK、RLINK、LTAG、RTAG以及SKIP场的树，本算法确定在这个TEXT中是否有一个以特定变元K开始的键（如果对于$r \geqslant 1$，存在r个这样的键，则随后有可能在$O(r)$步内来确定所有它们的位置；见习题14）。我们假定至少存在一个这样的键。

P1.〔初始化〕 置P←HEAD和$j \leftarrow 0$（变量P是一个沿此树下移的指针，而j是一个计数器，它将标记变元的二进位的位置），置$n \leftarrow K$中二进位的个数。

P2.〔左移〕 置Q←P和P←LLINK(Q)。如果LTAG(Q)=1，则转到P6。

P3.〔跳过二进位〕（这时我们知道，如果匹配成功，则不论K的前j个二进位同哪一个键匹配，它们都同在KEY(P)处开始的键匹配。）置$j \leftarrow j+$SKIP(P)，如果$j>n$，转到P6。

P4.〔检验二进位〕（这时我们知道，如果匹配成功，则无论K的前$j-1$个二进位同哪一个键匹配，它们都同在KEY(P)处开始的键匹配。）如果K的第j个二进位为0，则转到P2，否则转到P5。

P5.〔右移〕 置Q←P和P←RLINK(Q)，如果RTAG(Q)=0，则转到P3。

P6.〔比较〕（这时我们知道，如果匹配成功，则无论K同哪一个键匹配，它都同在KEY(P)处开始的键匹配。）K同TEXT阵列中位于KEY(P)处开始的键作比较。如果它们相等（直到n位，即K的长度为止），则这个算法成功地结束；如果不相等，则它以失败告终。

习题15说明了首先可以怎样来构造Patricia树。我们也可以把新的内容加到文本中和插入新的键，只要新的文本材料总是以唯一的限定符（例如，一个文本结束符号后面接一个序列号）结尾即可。

Patricia有一点技巧，而只有仔细地阅读，才能揭示它的所有美妙之处。

算法的分析 在结束本节之前，让我们对检索结构、数字查找树以及Patricia进行一番数学研究。这些分析的主要结果，在最后综述。

我们首先考虑二叉树的情况，即$M=2$的检索结构的情况。图34表示，当第5章排序例子的十六个键被处理作10位的二进数时，所形成的二进检索结构〔这些键均以八进制记号示出，例如1144表示10位数（1001100100）$_2$〕。如同在算法T中那样，我们使用检索结构来存储键的前导二进位的信息，直到达到了键被唯一确定的第一个位置为止；然后此键被全部重新记录。

如果把图34同表5.2.2-3进行比较，便可显示出检索结构存储同基数交换排序之间的令人惊异的关系（而且，这个关系也许是很显然的）。图34的22个节点精确地对应于表5.2.2-3中的22个分划阶段，而且在先序下的第p个节点对应于阶段p。在一个分划阶段中二进位的检索数，等于在对应的节点和其子检索结构内的键的个数；因此，我们可以叙述下列结果。

定理T 如果按如上所述把N个不同的二进数放置到一个二进检索结构中，则(i)这

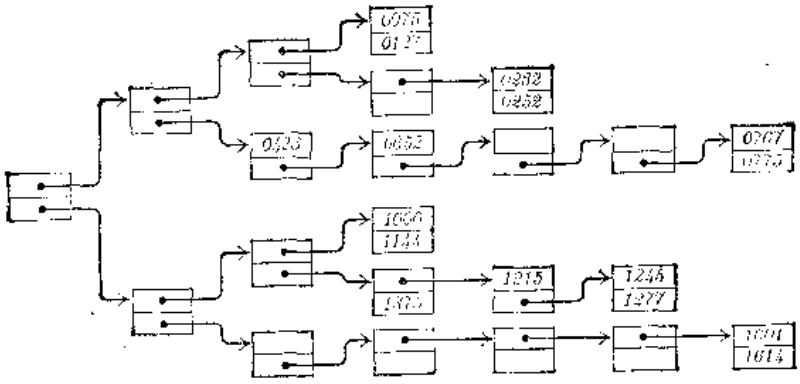


图34　一个随机二叉检索结构的例子

个检索结构的节点数就等于把这些数按基数交换排序所需要的分划阶段数；以及（ii）借助于算法T检索一个键所需要的二进位探查的平均次数，等于$1/N$乘以按交换排序所需要的二进位探查数。

由于这个定理，我们可以利用在5.2.2节中为进行基数交换而研制的全部数学机器。例如，如果假定，我们的键是在0与1之间一致分布的无限精度的随机实数，那么，为进行检索所需要的二进位探查数将是$\log_2 N+\gamma/(\ln 2)+1/2+f(N)+O(N^{-1})$，而检索结构的节点数将是$N/(\ln 2)+Ng(N)+O(1)$。这里，$f(N)$和$g(N)$是可以忽略的复杂函数，因为它们的值总小于$10^{-6}$（见习题5.2.2-38，48）。

当然，还有较多的工作犹待完成，因为需要把二进检索结构推广到M进检索结构。我们在这里只描述研究的起点，而把有教益的细节留作习题。

设A_N是在包含N个键的随几M进检索结构中的平均节点数，于是$A_0=A_1=0$，且对于$N\geqslant 2$，我们有

$$A_N=1+\sum_{k_1+k_2+\cdots+k_M=N}\left(\frac{N!}{k_1!\cdots k_M!}M^{-N}\right)(A_{k_1}+\cdots+A_{k_M}) \tag{3}$$

因为$N!M^{-N}/k_1!\cdots k_M!$是k_1个键在第一个子检索结构中，…，k_M个键在第M个子检索结构中的概率，利用对称性然后对$k_2,\cdots,k_M$求和，这个等式可改写成

$$A_N=1+M^{1-N}\sum_{k_1+\cdots+k_M=N}\left(\frac{N!}{k_1!\cdots k_M!}\right)A_{k_1}$$

$$=1+M^{1-N}\sum_k\binom{N}{k}(M-1)^{N-k}A_k\quad 对于N\geqslant 2 \tag{4}$$

类似地，如果C_N表示为找出检索结构中所有N个键所需要的二进位探查的平均总和数，则我们发现$C_0=C_1=0$且

$$C_N = N + M^{1-N}\sum_k \binom{N}{k}(M-1)^{N-k}C_k \quad 对于N \geqslant 2 \tag{5}$$

习题 17 说明了怎样处理这种类型的一般递归式，而习题 18-25 给出了对应的随机检索结构的理论〔对于 A_N 的分析，首先是由 L. R. 约翰逊（L. R. Johnson）和 M. H. 麦克安德鲁（M. H. McAndrew）联系到一个等价的面向硬件的排序算法从另一种观点解决的，见 *IBM J. Res. and Devel.* 8（1964），189-193.〕。

如果我们现在回到数字查找树的研究上，则会发现这些公式都是类似的，而尚不同的是，不容易看出怎样导出渐近特性来。例如，如果 $\overline{C}_N$ 表示当在一株二叉数字查找树中找出全部N个键时所作的二进位探查的平均总数，则和上面一样，不难导出 $\overline{C}_0 = \overline{C}_1 = 0$，且

$$\overline{C}_{N+1} = N + M^{1-N}\sum_k \binom{N}{k}(M-1)^{N-k}\overline{C}_k \quad 对于N \geqslant 0 \tag{6}$$

这几乎同等式（5）相同；但是在这个等式左边出现$N+1$而不是N，这足以改变递归式的整个特征，因此我们用来研究（5）的诸方法都不顶用了。

现在让我们首先考虑二进制的情况。图35示出，当已按第 5 章的例子所用的次序插入16个示例性的键时，对应于图34的数字查找树。如果我们要确定在一次成功的随机查找中所作的二进位探查平均次数，则它恰巧是这株树的内部路径长度除以N，因为我们需要 l 个二进的探查来找出级 l 上的一个节点。然而，要注意的是，在一次不成功的随机查找中所作的二进位探查的平均次数，并非简单地同这株树的外部路径长度相关，因为不成功的查找更可能出现在靠近根的外部节点处；例如，达到图35中节点0075的左子分枝的概率是 1/8（假定是无限精确度的键），而节点 0232 的左子分枝的概率仅为$\frac{1}{32}$（因此，当诸键一致分布时，数字查找树一般来说比算法6.2.2T的二分查找树能更好地维持平衡）。

我们可以用母函数描写一个数字查找树的有关特征。如果在级 l 上有 a_l 个内部节点，则考虑母函数 $a(z)=\sum a_l z^l$；例如，对应于图 35 的母函数为 $a(z)=1+2z+4z^2+5z^3+4z^4$。如果在级 l 上有 b_l 个外部节点，且如果 $b(z)=\sum b_l z^l$，则由习题 6.2.1-25 有

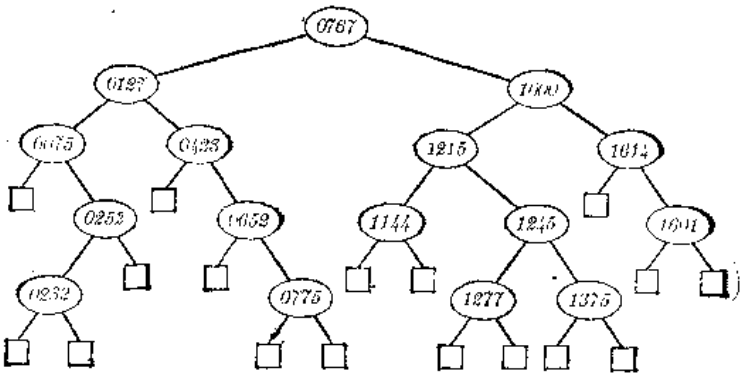


图35　由算法D构造的一株随机数字查找树

$$b(z)=1+(2z-1)a(z) \tag{7}$$

例如，$1+(2z-1)(1+2z+4z^2+5z^3+4z^4)=3z^3+6z^4+8z^5$。在一次随机的成功查找中，所做的二进位探查的平均次数是 $a'(1)/a(1)$，因为 $a'(1)$ 是树的内部路径长度，而 $a(1)$ 是内部节点数。在一次随机的不成功的查找中，所做的二进位探查的平均次数是 $\sum lb_l 2^{-l}=\frac{1}{2}b'\left(\frac{1}{2}\right)=a\left(\frac{1}{2}\right)$，因为我们是以 2^{-l} 的概率在级 l 上的一个给定外部节点处结束的。比较次数等同于二进位探查的次数，在一次成功的查找中此数要加 1。例如，在图35中，平均来说，一次成功的查找将花费 $2\frac{9}{16}$ 次二进位探查和 $3\frac{9}{16}$ 次比较；一次不成功查找的二进位探查和比较次数都将是 $3\frac{7}{8}$。

现在设 $g_N(z)$ 是具有 N 个节点的树的"平均" $a(z)$；换言之，$g_N(z)$ 是对于具有 N 个内部节点的所有二进数字查找树的求和式 $\sum p_T a_T(z)$，其中 $a_T(z)$ 是 T 的内部节点的母函数，p_T 是当使用算法D插入 N 个随机数时 T 出现的概率，那么，在一次成功的查找中二进位探查的平均次数将是 $g'_N(1)/N$，在一次不成功的查找中为 $g_N\left(\frac{1}{2}\right)$。

我们可以模拟树的构造过程计算出 $g_N(z)$。如果 $a(z)$ 是有 N 个节点的一株树的母函数，则我们可以通过在任何一个外部节点位置上作下一次插入，而形成 $N+1$ 株树，这个插入以 2^{-l} 的概率进到级 l 上一个给定的外部节点；因此，$N+1$ 株新树的母函数乘以出现的概率再求和为 $a(z)+b\left(\frac{1}{2}z\right)=a(z)+1+(z-1)a\left(\frac{1}{2}z\right)$。对于所有具有 N 个节点的树取平均值，就得

$$g_{N+1}(z)=g_N(z)+1+(z-1)g_N\left(\frac{1}{2}z\right);\quad g_0(z)=0 \tag{8}$$

外部节点对应的母函数 $h_N(z)=1+(2z-1)g_N(z)$，是更为容易给出的，因为（8）等价于公式

$$h_{N+1}(z)=h_N(z)+(2z-1)h_N\left(\frac{1}{2}z\right);\quad h_0(z)=1 \tag{9}$$

重复地应用这个规则，我们求得

$$\begin{aligned}h_{N+1}(z)&=h_{N-1}(z)+2(2z-1)h_{N-1}\left(\frac{1}{2}z\right)+(2z-1)(z-1)h_{N-1}\left(\frac{1}{4}z\right)\\&=h_{N-2}(z)+3(2z-1)h_{N-2}\left(\frac{1}{2}z\right)+3(2z-1)(z-1)h_{N-2}\left(\frac{1}{4}z\right)\\&\quad+(2z-1)(z-1)\left(\frac{1}{2}z-1\right)h_{N-2}\left(\frac{1}{8}z\right)\end{aligned}$$

等等，于是最后我们有

$$h_N(z)=\sum_k\binom{N}{k}\prod_{0\leqslant j<k}(2^{1-j}z-1) \tag{10}$$

$$g_N(z)=\sum_{k\geqslant 0}\binom{N}{k+1}\prod_{0\leqslant j<k}(2^{-j}z-1) \tag{11}$$

例如，$g_4(z)=4+6(z-1)+4(z-1)\left(\frac{1}{2}z-1\right)+(z-1)\left(\frac{1}{2}z-1\right)\left(\frac{1}{4}z-1\right)$这些公式使得有可能把我们正在寻找的量表达为乘积的和：

$$\overline{C}_N=g_N'(1)=\sum_{k\geqslant 0}\binom{N}{k+2}\prod_{1\leqslant j\leqslant k}(2^{-j}-1) \tag{12}$$

$$g_N\left(\frac{1}{2}\right)=\sum_{k\geqslant 0}\binom{N}{k+1}\prod_{1\leqslant j\leqslant k}(2^{-j}-1)=\overline{C}_{N+1}-\overline{C}_N \tag{13}$$

这个 $\overline{C}_N$ 的公式满足（6），这绝对不是显然的！

可惜，由于 $2^{-j}-1$ 是负的，这些表达式不便于计算或求出一个渐近展开式；我们得到相当大的项并有大量的消去。应用习题 5.1.1-16 的分划恒等式，可以得到 $\overline{C}_N$ 的一个更有用的公式。我们有

$$\begin{aligned}\overline{C}_N&=\left(\prod_{j\geqslant 1}(1-2^{-j})\right)\sum_{k\geqslant 0}\binom{N}{k+2}(-1)^k\prod_{l\geqslant 0}(1-2^{-l-k-1})^{-1}\\
&=\left(\prod_{j\geqslant 1}(1-2^{-j})\right)\sum_{k\geqslant 0}\binom{N}{k+2}(-1)^k\sum_{m\geqslant 0}(2^{-k-1})^m\prod_{1\leqslant r\leqslant m}(1-2^{-r})^{-1}\\
&=\sum_{m\geqslant 0}2^m\left(\sum_k\binom{N}{k}(-2^{-m})^k-1+2^{-m}N\right)\prod_{j\geqslant 0}(1-2^{-j-m-1})\\
&=\sum_{m\geqslant 0}2^m((1-2^{-m})^N-1+2^{-m}N)\sum_{m\geqslant 0}(-2^{-m-1})^n2^{-n(n-1)/2}\\
&\quad\times\prod_{1\leqslant r\leqslant n}(1-2^{-r})^{-1}\end{aligned} \tag{14}$$

乍一看去，这似乎并不是对于等式（12）的一个改进，但它有个很大的优点，即它对于每个固定的 n 皆迅速收敛。对于等式 5.2.2-38、39 中检索结构的情况，出现一种完全类似的状态；事实上，如果我们仅仅考虑（14）中 $n=0$ 的项，则其数量是$N-1$加上在一个二进检索结构中的二进位探查数。我们现在可以继续用同以前完全一样的方式，得到这个渐近值；见习题27〔上面的推导在很大程度上是以 A. J. 康海姆和 D. J. 纽曼(D. J. Newman)所提议的方法为基础的，见 *Discrete Mathematics* 4 (1973)，57-63〕。

最后让我们对 Patricia 进行一点数学考察。在这个情况下，二叉树就象是在同一些键上的对应的二进检索结构，但是被挤压在一起了（因为 SKIP 场消去了 1 路分枝），因此有$N-1$个内部节点和N个外部节点。图36示出了对应于图34的检索结构中 16 个键的Patricia树。在每个分枝节点中所示的数是 SKIP 的数量；尽管外部节点并不明显地出现，但键还是用节点来标记（实际上每个外部节点都用一个带标志的链接代替。该链接通到一个访问TEXT的内部节点处。为了分析的目的，我们可以假定如所示那样存在外部节点。

由于通过 Patricia 进行的成功查找结束于外部节点处，故在一次随机的成功查找中所作的二进探查的平均数将是外部路径长度除以N。如果我们对于外部节点，象上边那样构造母函数 $b(z)$，则这将是 $b'(1)/b(1)$。通过 Patricia的一次不成功的查找，也结束在一个外部节点处，但级 l 上的外部节点均以概率 2^{-l} 加权，所以二进位探查的平均数是 $\frac{1}{2}b'\left(\frac{1}{2}\right)$。例如，在图 36 中，我们有 $b(z)=3z^3+8z^4+3z^5+2z^6$；平均说来，每次

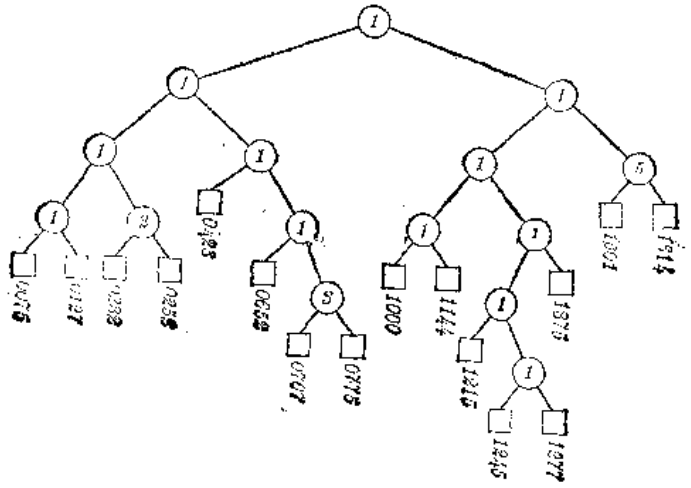


图36　Patricia构造代替图34的树

成功的查找有 $4\frac{1}{4}$ 个二进位探查，每次不成功的查找有 $3\frac{25}{32}$ 个。

设 $h_N(z)$ 是由 N 个外部节点，利用一致分布的键构造的一株 Patricia 树的"平均" $b(z)$，递归关系

$$h_n(z)=2^{1-n}\sum_k\binom{n}{k}h_k(z)(z+\delta_{kn}(1-z)),\quad h_0(z)=0,\quad h_1(z)=1 \tag{15}$$

似乎没有简单的解。幸而平均的外部路径长度 $h_n'(1)$ 有一个简单的递归式，因为

$$h_n'(1)=2^{1-n}\sum_k\binom{n}{k}h_k'(1)+2^{1-n}\sum_k\binom{n}{k}k(1-\delta_{kn})$$

$$=n-2^{n-1}n+2^{1-n}\sum_k\binom{n}{k}h_k'(1) \tag{16}$$

由于这有（6）的形式，我们就可以使用已经建立的用来解 $h_n'(1)$ 的诸方法，其结果恰巧比在一个随机的二进检索结构中对应的二进位的探查数小 n。于是对于随机数据，SKIP 场为每次成功的查找节省一个二进位探查（见习题31）。典型的实际数据的冗余性将导致更大的节省。

我们可以推算用 Patricia 进行的一次随机的不成功查找的二进位探查平均数，从而得到递归式

$$a_n=1+\frac{1}{2^n-2}\sum_{k<n}\binom{n}{k}a_k,\quad 对于\ n\geqslant 2;\quad a_0=a_1=0 \tag{17}$$

这里，$a_n=\frac{1}{2}h_n'\left(\frac{1}{2}\right)$。这不是我们已经研究过的任何递归式的形式，而且也不容易把它转换成过去见过的那种递归式。事实上，已经证明了这个解包含贝努利数：

$$\frac{na_{n-1}}{2}-n+2=\sum_{2\leqslant k<n}\binom{n}{k}\frac{B_k}{2^{k-1}-1} \tag{18}$$

这个公式大概是我们需要解决的最难的渐近问题；它的解在习题 34 中，这是对我们以前做过的许多事情的有益的回顾，并带有某些稍微不同的意味。

分析的综述　作为本节所有复杂的数学推导的结果，下列事实也许是最值得注意的：

（a）为把N个随机键存进一个M进检索结构中所需要的节点数(检索结构中的分枝终止于键数$\leqslant s$的子文件)近似地是$N/(s\ln M)$（这个近似对于很大的N，小的s，以及小的M是正确的）。由于一个检索结构包含M个链接场，故我们若选择$s=M$，则仅需要大约$N/\ln M$个链接场。

（b）在所有考虑过的方法中，一次随机查找期间考察的数字或字符个数，近似为$\log_M N$。当$M=2$时，各种分析给了我们下列更精确的二进位探查数的近似值：

	成功的	不成功的
检索结构查找	$\log_2 N+1.33275$	$\log_2 N-0.10995$
数字树查找	$\log_2 N-1.71665$	$\log_2 N-0.27395$
Patricia	$\log_2 N+0.33275$	$\log_2 N-0.31875$

（这些近似都可用基本数学常数来表达，例如0.31875代表$(\ln\pi-\gamma)/(\ln 2)-1/2$。）

（c）这里的“随机”数据意味着M进数字是一致分布的，即把键想象成以M进记号表示的0与1之间的实数。数字查找方法同键进入文件的次序无关（算法D除外，它是唯一对次序稍微敏感的）；但它们对于数字的分布非常敏感。例如，如果0的二进位比1的二进位更普遍，则这株树将变得比上面分析的随机数据的情况显得更为倾斜（习题5.2.2-53探讨了一个例子，说明当数据如此偏向时发生的情况）。

习题

1.〔00〕 如果一株树有叶，则一个检索结构有什么？

2.〔20〕 利用算法T的约定，试设计把一个新的键插入到一个M进检索结构中去的算法。

3.〔21〕 利用算法T的约定，试设计从一个M进检索结构删去一个键的算法。

▶4.〔21〕 表1中360个项大多数都是空白（空链接），但是我们可以通过使一些空的项同非空的项进行重叠，把这张表压缩成49个项如下：

项	位置	项	位置
…	1	HER	13
BE	2	…	14
(4)	3	…	15
(3)	4	HIS	16
(17)	5	I	17
(1)	6	HAD	18
(7)	7	BY	19
(20)	8	ARE	20
THE	9	HAVE	21
ON	10	WAS	22
TO	11	WHICH	23
FOR	12	HE	24

项	位置
…	25
IN	26
IS	27
(3)	28
THAT	29
WITH	30
A	31
(14)	32
FROM	33
IT	34
…	35
(18)	36
THIS	37

项	位置
OR	38
BUT	39
AND	40
AS	41
YOU	42
OF	43
…	44
(19)	45
(1)	46
NOT	47
AT	48
…	49

(表1的节点（1），（2），…，（12）分别在这个压缩表内的位置20，19，3，14，1，17，1，7，3，20，18，4处开始。)

证明，如果以压缩的表代替表1，则程序T仍将有效，但不十分快。

▶5.〔$M26$〕（Y.N.帕特（Y.N.Patt））考察图31的树，在那里每族中的字母都是按字符顺序排列的。这种顺序不是必要的，如果我们在构造象（2）那样的二叉树表示之前，重新安排族内节点的顺序，则可得到更快的查找。从这个观点看，图31的什么样的重新排列是最优的（利用图32的频率假定，并且去找这样的森林，当它表示成一株二叉树时，它使成功的查找时间极小化)？

6.〔15〕 如果通过算法D以递增次序插入十五个4位二进键0001，0010，0011，…，1111，则将得到什么样的数字查找树（在根处以0001开始，而后进行十四次插入)？

▶7.〔$M26$〕 如果以另一种次序插入习题6的15个键，则我们得到一株不同的树，在这些键的所有15!个可能的排列当中，哪一个是最坏的？所谓最坏指的是它产生具有最大内部路径长度的一株树。

8.〔20〕考虑下列对于算法D的修改，它有消去变量$K1$的作用：在步骤D_2中在两个位置上都把“$K1$”改成“K”，并从步骤D_1删去操作“$K1 \leftarrow K$”。试问得到的算法对于查找和插入是否仍将正确？

9.〔21〕 写出算法D的一个MIX程序，并把它同程序6.2.2T作比较。你可以使用诸如SLB（左移AX二进位)。JAE（如果A为偶则转移）等二进操作；你也可以使用习题8的思想，如果它有帮助的话。

10.〔23〕 给定一个文件，其中所有键都是n位二进数，并给定一个查找变元$K=b_1b_2\cdots b_n$，假设我们要在文件中所有以二进位型式$b_1b_2\cdots b_k$开始的键中求这样的k的极大值。如果这文件表示成（a）一株二进查找树（参考算法6.2.2T）（b）一个二进检索结构（参考算法T)，（c）一个二进数字查找树(参考算法D)，则我们如何能有效地进行？

11.〔21〕 如果不作改变，试问算法6.2.2D是否可用来从一株数字查找树删去一个节点？

12.〔25〕 在由算法D构造的一株随机数字查找树中删去一个随机元素之后，得到的树是否仍是随机的（参考习题 11 和定理6.2.2H）？

13.〔20〕（M进数字查找） 说明算法T和D怎样才能被组合成为一个扩充的算法，当$M=2$时它实际上同算法D一样。如果对于$M=30$使用你的算法，则表 1 将作什么改变？

▶14.〔25〕 试设计一个有效的算法，在算法P成功地结束之后，它就接着做下去，以确定出K出现于 TEXT 中的所有位置。

15.〔28〕 试设计一个有效的算法，它能用来构造为 Patricia 所使用的树，或者用来把新的 TEXT 访问插入到一株现存的树当中，你的插入算法至多只能使用 TEXT 阵列两次。

16.〔22〕 为什么对 Patricia 作出这样的限制，即任何键不能是另一个键的前缀，是行得通的？

17.〔M25〕 通过推广习题 5.2.2-36 的技术，借助二项式变换，找出一种表达递归式

$$x_0=x_1=0;\quad x_n=a_n+m^{1-n}\sum_k\binom{n}{k}(m-1)^{n-k}x_k,\quad n\geqslant 2$$

解的方式。

18.〔M21〕 借助类似习题 5.2.2-38 中定义的函数U_n和V_n，利用习题 17 的结果，表达（4）和（5）的解。

19.〔HM23〕 对于固定的$s\geqslant 0$和$m>1$，求当$n\to\infty$时函数

$$K(n,s,m)=\sum_{k\geqslant 2}\binom{n}{k}\binom{k}{s}\frac{(-1)^k}{m^{k-1}-1}$$

的渐近值到$O(1)$〔在习题 5.2.2-50 中已经解决了$s=0$的情况，而在习题 5.2.2-48 中已经解决了$s=1$，$m=2$的情况〕。

▶20.〔M30〕 如果在M进检索结构存储中，当我们达到键数$\leqslant s$的子文件时，即使用顺序查找（算法T是$s=1$时的特殊情况）。试应用上面一些习题的结果来分析（a）检索结构节点的平均数，（b）在一次成功的查找中数字或字符探查的平均数，以及（c）在一次成功的查找中所作的平均比较次数。对于固定的M和s，把你的答案叙述成当$N\to\infty$时的渐近公式；对于（a）的答案应该精确到$O(1)$的范围内，而对（b）和（c）的答案应该精确到$O(N^{-1})$的范围内〔当$M=2$时，这个分析也可应用于修正的基数交换排序上，在这个排序中，其大小$\leqslant s$的子文件通过插入进行排序〕。

21.〔M25〕 在含有N个键的一个随机M进检索结构中，有多少个节点在表的项 0 中有一个空的指针？（例如，在表 1 的 12 个节点中，有 9 个在“␣”位置中有空指针。本题中的“随机”象通常一样意味着键的字符一致地分布在 0 与$M-1$之间。）

22.〔M25〕 在含有N个键的一个M进检索结构中对于$l=0,1,2,\cdots$，有多少检索结构节点位于级l上。

23.〔M26〕 在含有N个随机键的一个M进检索结构中，在一次不成功的查找期间，平均作多少个数字探查？

24.〔M30〕 考虑一个M进检索结构，它已经表示作一片森林（参考图31）。试找出

对于下列两项精确的和渐近的表达式（a）在森林中节点的平均数，（b）在一次随机的成功查找中执行的“P←-RLINK(P)”的平均次数。

▶25.〔$M24$〕 在本节中对于渐近值的数学推导已经十分困难，同时涉及复变理论，这是因为希望得到比渐近特性的第一项还要多的结果(而第二项实质上是复杂的)。本题的目的是证明，对于导出某些较弱的结果，初等方法已够用了。（a）由归纳法证明对于$N\geqslant 1$，（4）的解满足$A_N\leqslant M(N-1)/(M-1)$，（b）设$D_N=C_N-NH_{N-1}/(\ln M)$，其中$C_N$由（5）定义。证明$D_N=O(N)$；因此$C_N=N\log_M N+O(N)$。〔提示：用（a）和定理1.2.7A。〕

26.〔23〕 确定无穷乘积

$$\left(1-\frac{1}{2}\right)\left(1-\frac{1}{4}\right)\left(1-\frac{1}{8}\right)\left(1-\frac{1}{16}\right)\cdots$$

的值，精确到五位十进数，用手计算。〔提示：参考习题5.1.1-16。〕

27.〔$HM31$〕 由（14）给出的$\overline{c}_N$精确到$O(1)$的渐近值是什么？

28.〔$HM26$〕 对一般的$M\geqslant 2$，当在一株随机的M进数字查找树中进行查找时，试求数字探查的渐近平均次数，既考虑成功的也考虑不成功的查找，并给出你的答案，精确到$O(N^{-1})$。

29.〔$M46$〕 在一株M进数字查找树中，试求那些所有的M链接均为空的节点的渐近平均数（我们可以通过消去这样的节点来节省内存空间，参考习题13〕。

30.〔$M24$〕 证明在（15）中定义的Patricia的母函数可以表达成相当吓人的形式

$$n\sum_{m\geqslant 1}z^m\left(\sum_{\substack{a_1+\cdots+a_m=n-1\\ a_1,\cdots,a_m\geqslant 1}}\binom{n-1}{a_1,\ \cdots,\ a_m}\frac{1}{(2^{a_1}-1)(2^{a_1+a_2}-1)\cdots(2^{a_1+\cdots+a_m}-1)}\right)$$

〔于是，如果对$h_n(z)$有一个简单的公式，则我们将有能力来简化这个相当笨拙的表达式。〕

31.〔$M21$〕 解递归式（18）。

32.〔$M21$〕 在具有$N-1$个内部节点的一株随机Patricia树中所有的SKIP场之和的平均值是什么？

33.〔$M30$〕 证明（18）是对递归式（17）的一个解。〔提示：考虑母函数$A(z)=\sum_{n\geqslant 0}a_nz^n/n!$。〕

34.〔$HM40$〕 本题的目的是求（18）的渐近特性。

（a）证明，如果$n\geqslant 2$

$$\frac{1}{n}\sum_{2\leqslant k<n}\binom{n}{k}\frac{B_k}{2^{k-1}-1}=\sum_{j\geqslant 1}\left(\frac{1^{n-1}+2^{n-1}+\cdots+(2^j-1)^{n-1}}{2^{j(n-1)}}-\frac{2^j}{n}+\frac{1}{2}\right)$$

（b）证明（a）中的被加数可由$1/(e^x-1)-1/x+1/2$来逼近，其中$x=n/2^j$；得到的和等于原来的和$+O(n^{-1})$。

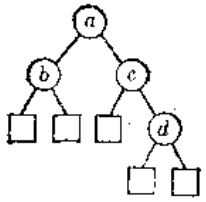
（c）证明

$$\frac{1}{e^x-1}-\frac{1}{x}+\frac{1}{2}=\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\zeta(z)\Gamma(z)x^{-z}dz,\quad 对于实\ x>0$$

（d）因此这个和等于

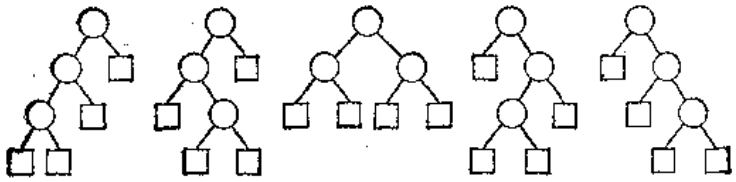
$$\frac{1}{2\pi i}\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\frac{\zeta(z)\Gamma(z)n^{-z}}{2^{-z}-1}dz+O(n^{-1})$$

计算这个积分。



▶35.〔$M20$〕 在五个键上的 Patricia 树将是且有如图所示的 SKIP 场a，b，c，d的概率是多少？（假定诸键都有独立的随机二进数，把答案作为 a，b，c，d 的函数给出。）

36.〔$M25$〕 兹有具有三个内部节点的五株二叉树。如果我们考虑在各种算法下，对于随机数据这些树中的每一株作为查找树出现的频率，则我们发现下列不同的概率：



查找树（算法 6.2.2T）	$\frac{1}{6}$	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{6}$	$\frac{1}{6}$
数字树查找（算法D）	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{8}$	$\frac{1}{8}$
Patricia（算法P）	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{3}{7}$	$\frac{1}{7}$	$\frac{1}{7}$

（注意数字查找树往往比其它树更经常地成为平衡的。）在习题 6.2.2-5 中，我们发现在树查找算法中，一株树的概率是 $\pi(1/s(x))$，其中乘积对所有内部节点 x 进行，而且 $s(x)$ 是在以 x 为根的子树中内部节点的个数，在（a）算法D；（b）算法P的情况下，试求一株树的概率的类似公式。

37.〔$M22$〕 考虑在级 l 上具有 b_l 个外部节点的一株二叉树。正文发现在数字查找树中，不成功的查找的运行时间不直接同外部路径长度 Σlb_l 有关，而是基本上同修正过的外部路径长度 Σlb_l2^{-l} 成正比。证明或否定：对于具有N个外部节点的所有树说来，最小的修正过的外部路径长度出现的条件是，所有外部节点都出现在至多两个相邻的级上〔参考习题 5.3.1-20〕。

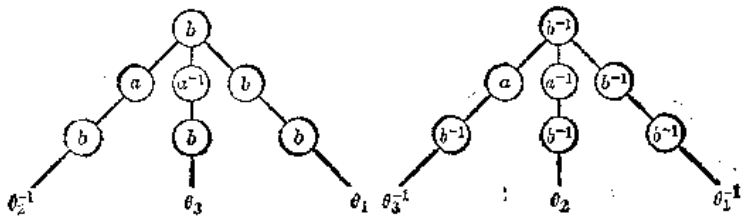
38.〔$M40$〕 给定 α 和 β，试建立一个算法，来找使 $\alpha\cdot$(内部路径长度)$+\beta\cdot$(修正过的外部路径长度）取极小值的 n 个节点的树（参考习题37)。

39.〔$M43$〕 建立一个算法，来找类似于 6.2.2 节所考虑的最优二分查找树的最优数字查找树。

40.〔25〕 设 $a_0a_1a_2\cdots$是一个周期二进序列，对于所有 $k\geqslant 0$，皆有 $a_{N+k}=a_k$。证明有一种方法以 $O(N)$个内存单元来表示这种类型的任何固定序列，使得下列操作可以在仅仅 $O(n)$ 步内完成：给定任何二进型式 $b_0b_1\cdots b_{n-1}$，确定这个型式在周期中出现的频繁程度（即，问有多少 p 值同时满足下列两个条件：即 $0\leqslant p<N$，且对于 $0\leqslant k<n$，有 $b_k=a_{p+k}$)。（假定每个内存单元足够大，能放下 0 和N之间的任何整数)。

41. 〔$HM28$〕 这是对群论的一个应用，设G是字母$\{a_1, \cdots, a_n\}$上的自由群，即，所有串$\alpha = b_1 \cdots b_r$的集合，其中每个b_i是a_j或a_j^{-1}之一，且没有相邻的对$a_j a_j^{-1}$或$a_j^{-1} a_j$出现。α的逆是$b_r^{-1} \cdots b_1^{-1}$，两个这样的串的相乘方法是连接它们并消去相邻的互逆的对。设H是通过串$\{\beta_1 \cdots \beta_t\}$生成的$G$的子群，即，可以写成诸$\beta$及它们的逆的乘积的$G$的所有元素的集合。可以证明（见 Marshall Hall, The Theory of Group (New York: Macmillan, 1959), Chapter 7）我们总可以求满足"尼尔森性质"的H的生成元$\theta_1, \cdots, \theta_m$，且$m \leqslant p$；尼尔森性质说，$\theta_i$的中间字符（如果它有偶数的长度，则至少$\theta_i$的两个中心字符之一）在表达式$\theta_i \theta_j^e$或$\theta_j^e \theta_i (e = \pm 1)$中决不会被消掉，只有$j = i$和$e = -1$是例外。这个性质意味着，有一个简单的算法，可以检验G的任意元素是否在H中：利用$2n$个字母$a_1, \cdots, a_n, a_1^{-1}, \cdots, a_n^{-1}$在一个面向字符的查找树中记录$2m$个键$\theta_1, \cdots, \theta_m, \theta_1^{-1}, \cdots, \theta_m^{-1}$。设$\alpha = b_1 \cdots b_r$是$G$的一个给定的元素；如果$r = 0$，则$\alpha$显然在$H$中，否则查找$\alpha$中能匹配一个键的最长前缀$b_1 \cdots b_k$。如果以$b_1 \cdots b_k$开始的键数超过一个，则$\alpha$不在$H$中；否则设唯一的这样的键是$b_1 \cdots b_k c_1 \cdots c_l = \theta_i^e$，以$\theta_i^{-e} \alpha = c_e^{-1} \cdots c_1^{-1} b_{k+1} \cdots b_r$代替$\alpha$。如果这个$\alpha$的新值比旧的更长（即如果$l > k$），则$\alpha$不在$H$中；否则对$\alpha$的新值重复此过程。尼尔森性质意味着这个算法总能结束。如果α最终归结成空串，则我们可以把原来的α的表示重新构造成为诸θ的一个乘积。

例如，设$\{\theta_1, \theta_2, \theta_3\} = \{bbb, b^{-1}a^{-1}b^{-1}, ba^{-1}b\}$且$\alpha = bbabaab$，森林



可用于上述算法来导出$\alpha = \theta_1 \theta_3^{-1} \theta_1 \theta_3^{-1} \theta_2^{-1}$。试实现这个算法，以诸$\theta$为你的程序的输入。

42. 〔23〕（前压缩和后压缩） 当二进键的一个集合被用作下标以分划一个较大的文件时，我们不必存储完全的键。例如，如果使用图 34 的 16 个键，则只要已经给出足够位的数字来唯一地识别它们，就可以在右边截断这些键：0000, 0001, 00100, 00101, 010, …, 1110001。这些截断了的键被用来把一个文件划分成 17 部分，其中例如第五部分由所有的以 0011 或 010 开始的键组成，而最后部分包含以 111001, 11101 或 1111 开始的所有键。在截断后的键中，如果我们去掉所有同上一个键相同的前导数字，则可以表示得更紧凑：0000, ＊＊＊1, ＊＊100, ＊＊＊＊1, ＊10, …, ＊＊＊＊＊＊1。跟随＊后边的二进位总是 1，所以它可以去掉。一个较大的文件将有许多＊，而且我们仅仅需要存储＊的个数和随后的二进位的值。（这项压缩技术是由 A. 赫勒（A. Heller）和 R. L. 约翰逊（R. L. Johnson）告知作者的。）

试证明，在压缩的文件中，除开＊和跟随＊的二进位 1 外，二进位的总数总等于键的二进检索结构中的节点数。

（因此，在整个下标中，这样的二进位的平均总数大约是$N/(\ln 2)$，即每个键仅 1.44 个二进位。由于我们仅需表示检索结构，因此还可能作进一步压缩；参考定理2.3.1A）

6.4 杂　　凑

至今我们已经考虑的查找方法，是以给定的变元K同表中的键作比较，或使用它的数字，来支配一个转移过程为基础的。第三种可能性是避免来回搜查，这就是通过对K作某种算术运算，计算一个函数$f(K)$，它是K的位置和表中的相关数据。

例如，再次考虑31个英文字的集合，在6.2.2节和6.3节中我们已经对于这些英文字施以各种查找策略。表1示出一个短的MIX程序，它把这31个键的每一个都变成为在-10和30之间唯一的数$f(K)$。如果我们把这个MIX程序所用的方法同已经考虑的其它方法（例如，二分查找，最优树查找，检索结构存储，数字树查找）作比较，则发现，从空间和速度两方面看，它都是更好的，只有二分查找使用的内存稍微少些。事实上，对于图12的频率数据，使用表1的程序，一次成功的查找的平均时间，大约仅$1.78u$，而且存储31个键仅需要41个表单元。

可惜，发现这样的函数$f(K)$并不是很容易的。从一个31个元素的集合到一个41元素的集合有$41^{31}\approx10^{50}$种可能的函数，而且它们当中仅仅$41\cdot40\cdots11=41!/10!\approx10^{15}$个能对每个变元给出不同的值；于是每一千万个函数当中仅仅有一个是适用的。

避免重复值的函数是异常稀少的，甚至对于一个相当大的表也是这样。例如，著名的"生日悖论"断言，如果有23个以上的人在一个房间里，则他们当中两个人有相同的出生月日的可能性很大。换言之，如果我们选择一个随机函数，它映射23个键到大小为365的一张表中，则两个键不映射到同一个单元的概率仅仅是0.4927（少于二分之一）。不相信这一结果的怀疑者们不妨在他们参加的下一次大型集会上查查各人的生日有无相同的

表1　把一组键转

指令	A	AND	ARE	AS	AT	BE	BUT	BY	FOR	FROM	HAD	HAVE	HE	HER
											给定一个特殊的键K，在其			
LD1N K(1:1)	-1	-1	-1	-1	-1	-2	-2	-2	-6	-6	-8	-8	-8	-8
LD2 K(2:2)	-1	-1	-1	-1	-1	-2	-2	-2	-6	-6	-8	-8	-8	-8
INC1 -8,2	-9	6	10	13	14	-5	14	18	2	5	-15	-15	-11	-11
J1P *+2	-9	6	10	13	14	-5	14	18	2	5	-15	-15	-11	-11
INC1 16,2	7	•	•	•	•	16	•	•	•	•	2	2	10	10
LD2 K(3:3)	7	6	10	13	14	16	14	18	2	5	2	2	10	10
J2Z 9F	7	6	10	13	14	16	14	18	2	5	2	2	10	10
INC1 -28,2	•	-18	-13	•	•	•	9	•	-7	-7	-22	-1	•	1
J1P 9F	•	-18	-13	•	•	•	9	•	-7	-7	-22	-1	•	1
INC1 11,2	•	-3	3	•	•	•	•	•	23	20	-7	35	•	•
LDA K(4:4)	•	-3	3	•	•	•	•	•	23	20	-7	35	•	•
JAZ 9F	•	-3	3	•	•	•	•	•	23	20	-7	35	•	•
DEC1 -5,2	•	•	•	•	•	•	•	•	•	9	•	15	•	•
J1N 9F	•	•	•	•	•	•	•	•	•	9	•	15	•	•
INC1 10	•	•	•	•	•	•	•	•	•	19	•	25	•	•
9HLDA K	7	-3	3	13	14	16	9	18	23	19	-7	25	10	1
CMPA TABLE,1	7	-3	3	13	14	16	9	18	23	19	-7	25	10	1
JNE FAILURE	7	-3	3	13	14	16	9	18	23	19	-7	25	10	1

〔生日诗论看来源于H. Davenport未发表的工作；参考W. W. R. Ball, *Math Recreations and Essays*(1939), 45, 另见R. von Mises, *Istanbul Üniversitesi Fen Fakültesi Mecmuasi* 4 (1939), 145-163, 以及W. Feller, *An Introduction to Probability Theory* (New York: Wiley, 1950), section2.3〕。

另一方面，表1中所用的方法是相当灵活的〔参考M. Greniewski and W. Turski, *CACM* 6 (1963), 322-323〕，而且对于一张中等规模的表，在进行了大约一天的工作之后，就能找到一个适当的函数。事实上，解决象这样的难题是颇为有趣的。有关适用技术的一个讨论，见R. Sprugnoli, *CACM* 20(1977), 841-850, 22(1979), 104。

当然，这个方法有一个严重的缺陷，因为这张表的内容必须是预先知道的；增多一个键都可能使一切破产，以致几乎需要从0开始。如果我们放弃唯一性的思想，允许不同的键产生相同的 $f(K)$ 值，而且在计算了 $f(K)$ 之后，再使用一种专门的方法解决任何二义性，则我们就可以得到一个更多样化的方法。

这些考虑导致了通常称作杂凑或散列存储技术的一类流行的查找方法。动词"杂凑"意味着把某些事物剁碎或把它弄得乱七八糟；杂凑的思想是把键的某些内容打乱，并且使用这种部分的信息作为查找的基础。我们计算一个杂凑函数 $h(K)$ 而且使用这个值作为查找开始的地址。

生日诗论告诉我们，很可能有不同的键 $K_i \neq K_j$，杂凑成相同的值 $h(K_i)=h(K_j)$。这种情况称作冲突，而且若干有趣的方法已被想出来处理冲突问题。为了使用一个散列表，一个程序员必须作两个几乎独立的决定：他必须选择一个杂凑函数 $h(K)$ 和解决冲突的方法。现在我们依次考虑这个问题的两个方面。

换成唯一的地址

HIS	I	IN	IS	IT	NOT	OF	ON	OR	THAT	THE	THIS	TO	WAS	WHICH	WITH	YOU
行了这条指令后rH的内容																
-8	-9	-9	-9	-9	-15	-16	-16	-16	-23	-23	-23	-23	-26	-26	-26	-28
-8	-9	-9	-9	-9	-15	-16	-16	-16	-23	-23	-23	-23	-26	-26	-26	-28
-7	-17	-2	5	6	-7	-18	-9	-5	-23	-23	-23	-15	-33	-26	-25	-20
-7	-17	-2	5	6	-7	-18	-9	-5	-23	-23	-23	-15	-33	-26	-25	-20
18	-1	29	•	•	25	4	22	30	1	1	1	17	-16	-2	0	12
18	-1	29	5	6	25	4	22	30	1	1	1	17	-16	-2	0	12
18	-1	29	5	6	25	4	22	30	1	1	1	17	-16	-2	0	12
12	•	•	•	•	20	•	•	•	-26	-22	-18	•	-22	-21	-5	8
12	•	•	•	•	20	•	•	•	-26	-22	-18	•	-22	-21	-5	8
•	•	•	•	•	•	•	•	•	-14	-6	2	•	11	-1	29	•
•	•	•	•	•	•	•	•	•	-14	-6	2	•	11	-1	29	•
•	•	•	•	•	•	•	•	•	-14	-6	2	•	11	-1	29	•
•	•	•	•	•	•	•	•	•	-10	•	-2	•	•	-5	11	•
•	•	•	•	•	•	•	•	•	-10	•	-2	•	•	-5	11	•
•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	21	•
12	-1	29	5	6	20	4	22	30	-10	-6	-2	17	11	-5	21	8
12	-1	29	5	6	20	4	22	30	-10	-6	-2	17	11	-5	21	8
12	-1	29	5	6	20	4	22	30	-10	-6	-2	17	11	-5	21	8

杂凑函数 为使事情更加明显，我们在这一节始终假定，杂凑函数最多取M个不同的值，且对于所有的键

$$0 \leqslant h(K) < M \tag{1}$$

实际上出现在真正文件中的键，通常都有大量的冗余性；我们必须小心寻找一个杂凑函数，它拆散几乎相同的成群的键，以便减少冲突的次数。

从理论上说，不可能定义这样一个杂凑函数，它由实际文件中的非随机数据建立随机的数据。但实际上，通过使用如同在第三章讨论的简单算术运算，不难产生出随机数据的一个相当好的拟态。而且事实上，通过进一步利用实际数据的非随机性质，来构造出一个比真正随机的键还少产生冲突的杂凑函数，我们常常甚至可以做得更好。

例如，考虑在一台十进制计算机上的 10 位数字键的情况。一个可考虑的杂凑函数将，比如说，命$M=1000$，以及命$h(K)$是从 20 位数字的乘积$K \times K$的接近于中间的某处选出的三位数字。这似乎将产生出在 000 和 999 之间相当好的散开的值，而且冲突概率较低。事实上，通过实际数据的实践表明，假定键中没有大量的前导或后尾的 0，这个“平方取中”的方法并不坏；但恰如我们在第三章中发现的，平方取中的方法不是一个特别好的随机数生成程序。还有更安全和更明智的处理方法。

对典型文件所作的广泛实验已经表明，有两大类杂凑函数十分好。一是以除法为基础的，而另一是以乘法为基础的。

除法方法是特别容易的；我们只需使用模M的余数：

$$h(K) = K \bmod M \tag{2}$$

在这种情况下，M的某些值显然比其它值要好得多。例如，如果M是一个偶数，则当K是偶数时$h(K)$将是偶数，当K是奇数时，$h(K)$也是奇数，在许多文件中这将导致一种相当大的偏向。若M是计算机基数的乘方，那将是更坏的，因为$K \bmod M$将仅是K的最低位上的一些数字（而同其它数字无关）。类似地，我们可以论证，M大概也不应是 3 的乘方；因为如果诸键都是字符的，则两个彼此仅仅是字母排列不同的键，在数值上可能仅仅差 3 的一个倍数（这是由于 $10^n \bmod 3 = 4^n \bmod 3 = 1$ 而出现的现象）。一般地说，我们要避免能整除$r^k \pm a$的M值，其中k和a是较小的数且r是字符集合的基数（通常$r=64, 256$或 100），因为对这样一个M值求模的余数，往往只是键中数字的叠加。这样一个考虑提示我们选择M为一个质数使得$r^k \equiv \pm a$ (moduloM)。已经发现，对于小的k和a。这种选择实际上在所有情况下都是十分令人满意的。

例如，在 MIX 计算机上，我们可以选择$M=1009$，通过序列

$$\begin{array}{lll} \text{LDX} & K & \text{rX} \leftarrow K \\ \text{ENTA} & 0 & \text{rA} \leftarrow 0 \\ \text{DIV} = 1009 = \text{rX} \leftarrow K \bmod 1009 & & \end{array} \tag{3}$$

计算$h(K)$。

乘法的杂凑方案也同样容易做出，但要稍难描述些，因为我们必须想象是对分数进行运算而不是对整数。设w是计算机字的大小，对于 MIX，它通常是 10^{10} 或 2^{30}；我们可以认为一个整数A是分数A/w，这里我们想象小数点位于这个字的左边。这方法就是来选择与w互质的某个数常数A，而且命

$$h(K)=\left\lfloor M\left(\left(\frac{A}{w}K\right)\bmod 1\right)\right\rfloor \tag{4}$$

在这种情况下，我们通常设M在一台二进计算机上是 2 的乘方，于是$h(K)$由乘积AK的较低一半的一些前导二进位组成。

在 MIX 代码中，如果我们设$M=2^m$且采用二进制，则乘法杂凑函数就是

```
LDA    K    rA←K
MUL    A    rAX←AK                    (5)
ENTA   0    rAX←AK mod w
SLB    m    rAX 的 m 个二进位移到左边
```

现在$h(K)$出现在寄存器A中。由于 MIX 的乘法和移位指令相当慢，故这个序列恰巧花费和（3）一样长的计算；但是在许多机器上乘法比除法要快得多。

在某种意义下这个方法可以认为是（3）的推广，因为例如我们可以取A为$w/1009$的一个近似值；乘以一个常数的倒数，通常要比除以该常数更快些。注意，（5）几乎就是“平方取中”方法，但有一点重要的区别：我们将看到，乘以一个适当的常数有许多可论证的好性质。

乘法方案好的特征之一在于，在（5）中不损失信息；在（5）完成之后，仅仅给定rAX的内容，我们即可再次确定K。原因在于A与w互质，所以欧几里得算法可用来求一常数A'，使得$AA' \bmod w=1$；这意味着$K=(A'(AK \bmod w)) \bmod w$。换言之，如果$f(K)$表示在（5）中的 SLB 指令之前寄存器$X$的内容，则

$$K_1 \neq K_2 \quad \text{意味着} \quad f(K_1) \neq f(K_2) \tag{6}$$

当然，$f(K)$在 0 到$w-1$的范围内取值，所以它不象一个杂凑函数那样好，但作为一个扰乱函数是非常有用的。所谓扰乱函数就是满足（6）和趋向于把键随机化的一个函数。如果键的次序无关重要，则在同 6.2.2 节的树查找函数相联系时，这样一个函数可能是非常有用的，因为当键以递增的次序进入树时，它消除了退化的危险性。如果实际的键的二进位有偏向时，则一个扰乱函数在同 6.3 节的数字树查找算法相联系时也是有用的。

乘法杂凑方法的另一个特征是，它很好地利用了在许多文件中存在的非随机性。实际的键集合通常有一种算术级数的倾向，其中$\{K, K+d, K+2d, \cdots, K+td\}$全都出现在文件中；例如考虑象{PART 1，PART 2，PART 3}或者{TYPEA, TYPEB, TYPEC}这样的字符名称。乘法杂凑方法把一个算术级数转换成不同杂凑值的近似的算术级数$h(K)$，$h(K+d)$，$h(K+2d)$，…，同时减少在随机状态下我们所预期的冲突次数。除法方法也有同样的性质。

图 37 在一种特别有趣的情况下，说明了乘法杂凑的这个方面。假定A/w近似地是黄金比$\phi^{-1}=(\sqrt{5}-1)/2=0.6180339887$；则通过考察$h(0)$，$h(1)$，$h(2)$，…诸相继的值的特性，可以研究$h(K)$，$h(K+1)$，$h(K+2)$…的特性。这提示了下面的实验：由线段$[0, 1]$开始，逐次地标出点$\{\phi^{-1}\}$，$\{2\phi^{-1}\}$，$\{3\phi^{-1}\}$，…，其中$\{x\}$代表$x$的小数部分（即$x-\lfloor x \rfloor = x \bmod 1$）。如同在图 37 中所示，这些点彼此之间分得很开；事实上，每个最新增加的点都落入最大的剩下的区间之一，而且按黄金比分割(这个现象首先是由 J. 奥德菲尔德（J. Oderfeld）猜测，并且由 S. 斯威尔兹科夫斯基（S.

Świerczkowski）证明的〔见 Fundamenta Math. 46(1958), 187-189〕。在这个证明中，斐波那契数起着重要作用。

黄金比的这个值得注意的性质实际上是原来由雨果·斯坦豪斯猜测并且首先由维拉·杜兰·索斯（Vera Turán Sós）证明的〔*Acta Math. Acad. Sci. Hung.* 8 (1957), 461-471; *Ann. Univ. Sci. Budapest. Eötvös Sect. Math.* 1 (1958), 127-134〕：

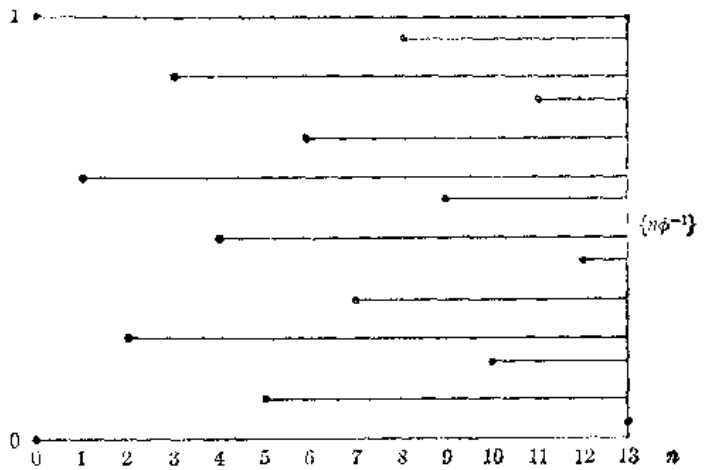


图37 斐波那契杂凑

定理S 设 θ 是任意无理数。当把点 $\{\theta\}$，$\{2\theta\}$，…，$\{n\theta\}$ 放置在线段〔0，1〕中时，形成的 $n+1$ 个线段至多有三种不同的长度。而且，下一点 $\{(n+1)\theta\}$ 将落在最大的现存线段之一中。

于是，点 $\{\theta\}$，$\{2\theta\}$，…，$\{n\theta\}$ 非常均匀地散列在 0 与 1 之间。如果 θ 是有理的，则同样的定理也成立，但我们必须对长度为 0 的线段给出一个适当的解释，这长度为 0 的线段是在 n 大于或等于 θ 的分母时出现的。定理 S 的一个证明，连同关于这种状态的基础结构的一个详细分析，出现于习题 8 中；可以证明，具有给定长度的诸线段是以先进先出的方式建立和破坏的。当然，某些 θ 比其它的 θ 更好，因为例如接近于 0 或 1 的一个值将以许多小段和一大段开始。习题 9 证明，在 0 与 1 之间的所有数 θ 当中，两个数 ϕ^{-1} 和 $\phi^{-2}=1-\phi^{-1}$ 导致"最一致地分布"的序列。

上面的理论提示了斐波那契杂凑，其中我们选择常数 A 是最接近于 $\phi^{-1}w$ 的与 w 互质的整数。例如，如果 MIX 是十进计算机，则取

$$A=\boxed{+}\boxed{61}\boxed{80}\boxed{33}\boxed{98}\boxed{87} \qquad (7)$$

这个乘数将非常好地撒开象 LIST 1，LIST 2，LIST 3 这样的字符键。但注意在第四个字符位置中有一个算术级数的情况，如同在键 SUM 1 ⊔，SUM2⊔，SUM3⊔中那样；其效果就象以 $\theta=\{100A/w\}=.80339887$ 代替 $\theta=.6180339887=A/w$ 来使用定理 S 那样。尽管这个 θ 值不完全象 ϕ^{-1} 那样好，但得到的结果仍然是不错的。另一方面，如果这个级数出现在第二个字符位置上，如同在 $A1$⊔⊔⊔，$A2$⊔⊔⊔，$A3$⊔⊔⊔中那样，则有效的 θ 是 .9887，而这可能太接近于 1 了。

因此，我们通过用

$$A=\boxed{+}\boxed{61}\boxed{61}\boxed{61}\boxed{61}\boxed{61} \qquad (8)$$

这样一个乘数代替（7），可以做得更好；这样一个乘数将分散在任何字符位置中互异的

连续的键序列。不幸的是，这个选择遇到了另外的问题，有点类似于除以$r^k \pm 1$时的困难；诸如XY和YX这样的键将趋于杂凑到相同的单元！摆脱这个困难的一种方式，是更仔细地考察定理S背后的基础结构；当键的级数很短时，仅仅θ的连分数表示的开头几个部分商是起作用的，而且小的部分商对应于好的分布性质。因此我们发现θ的最好的值位于范围

$$\frac{1}{4} < \theta < \frac{3}{10}, \quad \frac{1}{3} < \theta < \frac{3}{7}, \quad \frac{4}{7} < \theta < \frac{2}{3}, \quad \frac{7}{10} < \theta < \frac{3}{4}$$

之中。可以找到A的一个值，使得它的每个字节都位于一个好的范围之中，而且不太接近于其它字节的值或这些值的补，例如，可以考虑把

$$A = \boxed{\div \mid 61 \mid 25 \mid 42 \mid 33 \mid 71} \tag{9}$$

作为这样一个乘数（关于乘法杂凑的这些思想大部分是由R.W.弗洛伊德给出的）。

一个好的杂凑函数应满足下列两个要求：

a）它的计算应该是非常快的。

b）它应该使冲突极小化。

性质（a）有些和机器有关，而性质（b）则与数据有关。如果诸键是真正随机的，则我们可以从其中简单地抽出一些二进位，并使用这些二进位作为杂凑函数；但实际上，我们几乎总是需要有依赖于键的所有二进位的一个杂凑函数，以便满足（b）。

至今我们已经考虑了怎样杂凑一个字的键。多字的或可变长的键可以通过上述方法的多精度扩充来处理，但是一般宜于把几个字组成一个单个的字，然后如上面那样进行单个乘法或除法，这样可以提高速度。这个组合可以通过mod w加法或通过在一台二进计算机上的“异或”来完成；这两个操作都有这样一个优点，它们都是可逆的，也就是，它们都依赖于两个变元的所有二进位，而且由于异或避免了算术溢出，它有时是更可取的。注意，这两种操作是可交换的。例如（X，Y）和（Y，X）都将杂凑相同的地址；G.D.诺特（G.D.Knotl）已经建议在进行加法或异或之前作一个循环移位以避免这个问题。

已经提出了许许多多的杂凑方法，但是还没有证明出这些方法中哪一个比上面叙述的简单的除法和乘法更优越。关于某些其它方法及其用于实际文件时性能的详细统计，见V.Y.卢姆（V.Y.Lum），P.S.T.延（P.S.T.Yuen）以及M.多德（M.Dodd），*CACM*，14(1971)，228-239等人的文章。

在已经试验过的所有其它杂凑方法中，也许最有趣的是一项以代数编码理论为基础的技术；其思想类似于上面的除法方法，但我们除以一个多项式modulo 2而不是除以一个整数(如同在4.6节所观察到的，这个操作类似于除法，就如同加法类似于异或那样)。对于这个方法，M应该是2的一个乘方，比如说，$M=2^m$，而且我们利用一个m次多项式$P(x)=x^m+p_{m-1}x^{m-1}+\cdots+p_0$。一个$n$个数字的二进键$K=(k_{n-1}\cdots k_1k_0)_2$可以看作是多项式$K(x)=k_{n-1}x^{n-1}+\cdots+k_1x+k_0$，而且利用多项式算术运算modulo 2计算余式

$$K(x) \bmod P(x) = h_{m-1}x^{m-1}+\cdots+h_1x+h_0$$

则$h(K)=(h_{m-1}\cdots h_1h_0)_2$。如果适当地选择$P(x)$，则这个杂凑函数即可以保证避免在近

乎相等的键之间的冲突。例如，如果 $n=15$，$m=10$，

$$P(x)=x^{10}+x^8+x^5+x^4+x^2+x+1 \tag{10}$$

则可以证明，每当 K_1 和 K_2 是两个不同的在少于七个二进位位置中互异的键时，$f(K_1)$ 不等于 $f(K_2)$。（关于这个方案的进一步信息见习题 7；当然它比较适合于以硬件或微程序设计实现，而不适合于用软件实现。）

已经发现，当调试一个程序时，使用一个常数杂凑函数 $h(K)=0$ 是方便的，因为所有的键都将被存储在一起；稍后可以换成一个有效的 $h(K)$。

通过“拉链”解决冲突 我们已经注意到，某些杂凑地址由于为多个键所共享，可能要出麻烦。解决这个问题的也许最显然的方法，是保持M个链接表，对于每个可能的杂凑码用一个。每个记录应包括一LINK 场，而且还将有M个表头，比如说以从 1 到M来进行编号。在杂凑了这个键之后，我们只需在号码为 $h(K)+1$ 的表中进行顺序查找（参考习题 6.1-2，这一情况非常类似于多重表的插入排序，即程序 5.2.1M）。

图 38 示出了当$M=9$时，对于七个键的序列

$$K=\text{EN, TO, TRE, FIRE, FEM, SEKS, SYV} \tag{11}$$

（这就是挪威文中的数 1 到 7）这一简单的拉链方案，这些键的杂凑码分别为

$$h(K)+1=3,\ 1,\ 4,\ 1,\ 5,\ 9,\ 2 \tag{12}$$

第一个表有两个元素，而另外有三个表是空的。

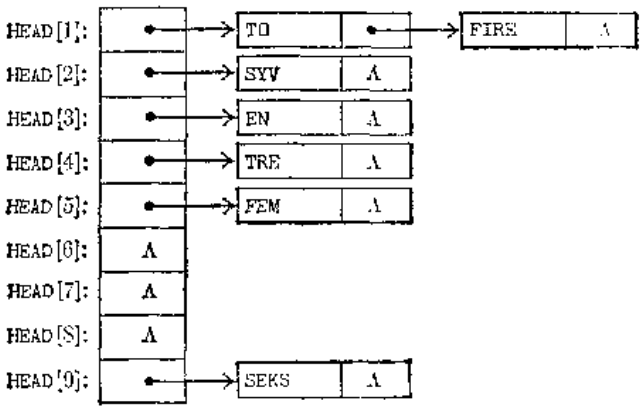


图38 分开的拉链

拉链是十分快的，因为这些表都是很短的。如果把 365 人聚集在一个房间里，则大概将有许多对的人有相同的生日，但是具有任何给定生日的人的平均数将仅仅是 1！一般说来，如果有N个键和M个表，则表的平均大小是N/M；于是杂凑使顺序查找所需的工作数量大约减少了一个因子M。习题 34 给出了一个精确的公式。

这个方法是我们以前讨论的技术的一个直截了当的组合，所以不需要描述对于拉链散列表的详细算法。以键的顺序来保持各别的表通常是一个好的想法，它使得插入和不成功的查找都能更快地进行。于是如果我们使表成为递增的，则图 38 的 TO 和 FIRE 节点将互相交换，而且所有的 Λ 链都将被指向一个空记录的指针所代替，该记录的键为∞（参考算法6.1T）。或者，我们可以利用 6.1 节中讨论的“自组织文件”的概念，把诸表按各键最新出现的时间排序，而不是按键的值排序。

为了保证速度，我们将乐于使M相当大。但当M很大时，许多表都将是空的，而且M个表头的许多空间都将浪费掉。这就提示了另外一个当记录很小时可用的方法：我们可以把表头和记录的存储重叠起来，只提供M个记录和M个链接而不是N个记录和$M+N$个链接所需的空间。有时有可能对所有数据作一趟扫描，找出哪一些表头将被使用，然后作另一趟扫描把所有的“溢出”记录都插入到空的位置上。但是这往往是不实际的或不可能

的，而且我们宁可用一种每个记录只处理一次，即只在它第一次进入系统时处理一次的技术。下面由F.A.威廉斯〔F.A.Williams〕给出的算法〔*CACM* 2，6(June1959)，21-24〕是解决这个问题的一种方便的方法。

算法C（拉链散列表的查找和插入）本算法在一个M节点的表中查找一个给定的键K。如果K不在表中，且这个表不满，则插入K。

表的诸节点，以TABLE〔i〕来表示，$0 \leqslant i \leqslant M$，而且它们有两种不同的类型，即空的和已占用的。一个已占用的节点含有一个键场KEY〔i〕，一个链接场LINK〔i〕，以及可能还有其它的场。

这个算法利用了一个杂凑函数$h(K)$，还使用一个辅助变量R来帮助寻找可用空间。当表空时，$R=M+1$，而当进行插入时，对于在$R \leqslant i \leqslant M$范围中的所有$i$，TABLE〔$i$〕是已占用的。约定TABLE〔0〕总是空的。

C1.〔杂凑〕 置$i \leftarrow h(K)+1$（现在$1 \leqslant i \leqslant M$）。

C2.〔有一个表吗？〕 如果TABLE〔i〕是空的，则转到C6(否则TABLE〔i〕已占用；我们将考察在这里开始的已占用节点的表列)。

C3.〔比较〕 如果$K=$KEY〔i〕，则这个算法成功地结束。

C4.〔进到下一个〕 如果LINK〔i〕$\neq 0$，则置$i \leftarrow$LINK〔i〕并转回到步骤C3。

C5.〔找空节点〕（这次查找不成功，要找表中的一个空位置。）R减值一次或多次直到找到使得TABLE〔R〕为空的R值为止。如果$R=0$，则这个算法以溢出（没有空节点）结束；否则置LINK〔i〕$\leftarrow R$，$i \leftarrow R$。

C6.〔插入新键〕 标记TABLE〔i〕为一个已占用的节点，并作KEY〔i〕$\leftarrow K$和LINK〔i〕$\leftarrow 0$。

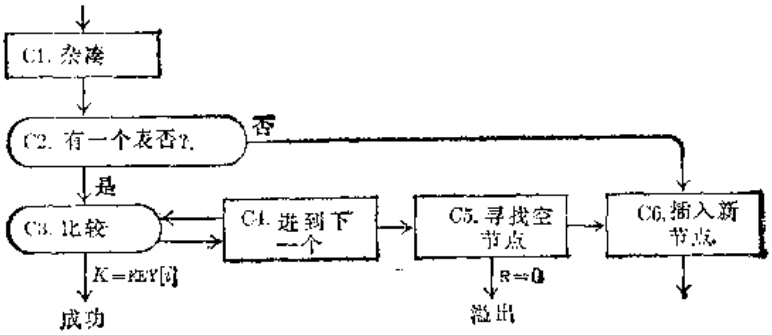


图39 拉链散列表的查找和插入

这个算法允许若干表结合起来，使得在诸记录插入到表中之后，就不必移动了。例如，见图40，其中SEKS出现在含有TO和FIRE的表中，因为后者已经插入到位置9上。

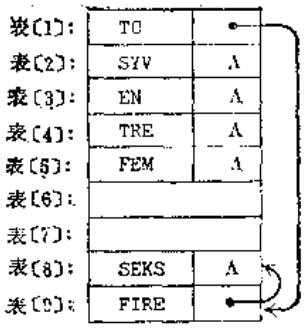


图40 结合起来的拉链

为了了解这个算法同这一章的其它算法比较时能匹敌到何种程度，我们可以写出下列MIX程序。以下分析指出，已占用单元的表一般是短的，而且程序设计已经考虑到这一点。

程序C（拉链散列表的查找和插入） 为方便起见，假定键仅有三个字节长，且节点被表示如下：

空的	−	1	0	0	0	0
已占用的	+	LINK		KEY		

(13)

假定表的大小M是质数；TABLE[i]被存储在TABLE+i之中，rI1≡i，rA≡K。

```
01   KEY     EQU     3:5
02   LINK    EQU     0:2
03   START   LDX     K                 1        C1. 杂凑
04           ENTA    0                 1
05           DIV     =M=               1
06           STX     *+1(0:2)          1
07           ENT1    *                 1        i ← h(k)
08           INC1    1                 1        +1
09           LDA     K                 1
10           LD2     TABLE,1(LINK)     1        C2. 有一个表否?
11           J2N     6F                1        如果TABLE[i]为空则转到C6
12           CMPA    TABLE,1(KEY)      A        C3. 比较
13           JE      SUCCESS           A        如果K=KEY[i]，则转出
14           J2Z     5F                A−S1     如果LINK[i]=0，则转到C5
15   4H      ENT1    0,2               C−1      C4. 进到下一个
16           CMPA    TABLE,1(KEY)      C−1      C3. 比较
17           JE      SUCCESS           C−1      如果K=KEY[i]则转出
18           LD2     TABLE,1(LINK)     C−1−S2
19           J2NZ    4B                C−1−S2   如果LINK[i]≠0，则转出
20   5H      LD2     R                 A−S      C5. 找空节点
21           DEC2    1                 T        R←R−1
22           LDX     TABLE,2           T
23           JXNN    *−2               T        重复直到TABLE[R]为空
24           J2Z     OVERFLOW          A−S      如果未剩下空节点则转出
25           ST2     TABLE,1(LINK)     A−S      LINK[i]←R
26           ENT1    0,2               A−S      i←R
27           ST2     R                 A−S      修改内存中的R
28   6H      STZ     TABLE,1(LINK)     1−S      C6. 插入新的键
29           STA     TABLE,1(KEY)      1−S      KEY[i]←K
```

这个程序的运行时间依赖于

C＝查找时所探查的表的项数

A＝1如果初始探查发现一个已占用的节点

S＝1如果成功，否则为0

T＝寻求可用空间时所探查的表项数

这里$S=S1+S2$，其中如果第一次试验成功则$S1=1$。程序C的查找阶段总共的运行时间是$(7C+4A+17-3S+2S1)u$，而当$S=0$时一个新键的插入还须花费附加的

$(8A+4T+4)u$。

假设这个程序开始时表中有N个键，而且设

$$\alpha=N/M=\text{这个表的负载因子} \tag{14}$$

如果杂凑函数是随机的，则在一次不成功的查找中A的平均值显然是α，而且习题39证明在一次不成功的查找中C的平均值是

$$C'_N=1+\frac{1}{4}\left(\left(1+\frac{2}{M}\right)^N-1-\frac{2N}{M}\right)\approx 1+\frac{1}{4}(e^{2\alpha}-1-2\alpha) \tag{15}$$

于是，当这个表为半满时，在一次不成功的查找当中所作探查的平均次数大约是$\frac{1}{4}(e+2)\approx 1.18$；甚至当这个表快要完全满时，在插入最后的项之前所作探查的平均次数，将仅仅大约是$\frac{1}{4}(e^2+1)\approx 2.10$。如同在习题40中所证明的那样，标准离差也是小的。这些统计证明，当杂凑函数是随机的时，即使算法偶尔允许诸表结合起来，这些表仍是短的。如果杂凑函数是坏的或我们极不走运，当然C可以象N那样大。

在一次成功的查找中，我们总有$A=1$。在一次成功的查找期间探查的平均次数，可以计算如下：若假定每个键都同等可能，则求$C+A$对前N次不成功的查找之和并除以N。于是我们得到

$$C_N=\frac{1}{N}\sum_{0\leqslant k<N}\left(C'_k+\frac{k}{M}\right)=1+\frac{1}{8}-\frac{M}{N}\left(\left(1+\frac{2}{M}\right)^N-1-\frac{2N}{M}\right)+\frac{1}{4}\frac{N-1}{M}$$

$$\approx 1+\frac{1}{8\alpha}(e^{2\alpha}-1-2\alpha)+\frac{1}{4}\alpha \tag{16}$$

这是在一次随机的成功的查找中探查的平均次数。甚至从一个满的表中找出一个项，平均也才仅仅需要1.80次探查！类似地（见习题42），知$S1$的平均值是

$$S1_N=1-\frac{1}{2}((N-1)/M)\approx 1-\frac{1}{2}\alpha \tag{17}$$

乍一看去，步骤C5可能显得是低效的，因为它要顺序地查找一个空位置。但是实际上当一个表正被构造时，在步骤C5中所作的表探查的总数决不超过表中的项目数；所以我们每作一次插入平均至多需要一次探查！习题41证明，在一次随机的不成功的查找中T近似于αe^α。

有可能对算法C进行修改，使得任何两个表都不结合在一起，但是这样一来记录就要搬家了。例如，考虑在我们刚要把SEKS插入到位置9之前，图40的状况；为了保持表是分开的，必须移动FIRE，而且为了这一目的，还要发现哪个节点是指向FIRE的。如同D·E·弗格森所建议的，通过杂凑FIRE和往下查找它的表，我们无须提供两路链接就能解决这个问题，因为这些表都是短的；习题34证明，当诸表不相结合时，平均的探查次数，减少成

$$C'_N=1+\frac{N(N-1)}{2M^2}\approx 1+\frac{1}{2}\alpha^2 \quad \text{（不成功的查找）} \tag{18}$$

$$C_N=1+\frac{N-1}{2M}\approx 1+\frac{1}{2}\alpha \quad \text{（成功的查找）} \tag{19}$$

但这一对（15）和（16）的改进尚不足以构成改变这个算法的理由。

另一方面，巴特勒·兰普森（Butler Lampson）已经发现，如果我们不把这些表结合在一起，则在使用拉链方法时，大多数用于链接的空间都可以节省。这导致了在习题13中讨论的一个有趣的算法。这个方法在每一项中引入一个标志位；因此在一次不成功的查找中所需平均探查数稍微减少，从（18）减少成为

$$\left(1-\frac{1}{M}\right)^N+\frac{N}{M}\approx e^{-\alpha}+\alpha \tag{18'}$$

注意，当$N>M$时，可以使用拉链，所以溢出不是一个严重的问题。当诸表如同算法C中那样结合起来时，我们可以把额外的一些项目链接成一个辅助的存储池；L.吉巴斯（L. Guibas）已经证明，在此情况下，为插入第（$M+L+1$）项需作的探查平均次数为$\left(L/2M+\frac{1}{4}\right)((1+2/M)^M-1)+1/2$。然而，通常喜欢使用把头一个冲突的元素放进一个辅助存储区域的另一方案，并且仅当这个辅助区域已经满时才允许诸表结合在一起；见习题43。

以“开式寻址法”解决冲突 解决冲突的另一个方法，是完全撤销链，简单地逐个考察表的各项，直到或者发现键K或者发现一个空的位置。这个思想就是确定某种规则，通过它，每个键K确定一个“探查序列”，即表中的一系列位置，每当要插入或查出K时这些位置即被探查。如果我们使用由K确定的探查序列，在查找K时，遇到一个空缺未用的位置，则可以得出结论，K不在表中，因为在每次处理K时用的是同一个探查序列。此类通用的方法为W. W. 彼得森称作开式寻址法〔IBM J. Research & Development 1 (1957), 130-146〕。

称为线性探查的最简单的开式寻址方案，使用如同下列算法中那样的循环探查序列

$$h(K),\ h(K)-1,\ \cdots,\ 0,\ M-1,\ M-2,\ \cdots,\ h(K)+1 \tag{20}$$

算法L（开式散列表查找和插入） 这个方法在有M个节点的表中寻找一个给定的键K，如果K不在表中且表不满，则插入K。

以TABLE〔i〕，$0\leqslant i<M$，表示表的节点，它们有两种不同的类型，即空的和已占用的。每个已占用的节点包含称为KEY〔i〕的一个键，可能还有其它的场。一个辅助变量N用以记住占用了多少个节点；把N看作是表的一部分，而且当插入一个新键时，N的值增1。

本算法利用一个杂凑函数$h(K)$，并使用线性探查序列（20）查表。对该序列的修改将在下面讨论。

L1.〔杂凑〕 置$i\leftarrow h(K)$（现在$0\leqslant i<M$）。

L2.〔比较〕 如果KEY〔i〕$=K$，则这个算法成功地结束。否则，如果TABLE〔i〕为空，则转回L4。

L3.〔进至下一个〕 置$i\leftarrow i-1$；如果现在$i<0$，则置$i\leftarrow i+M$。转回到步骤L2。

L4.〔插入〕（查找是不成功的） 如果$N=M-1$，则算法以溢出结束（这个算法认为当$N=M-1$时，而不是当$N=M$时表是满的，见习题15）。否则置$N\leftarrow N+1$，标记TABLE〔i〕为已占用的，且置KEY〔i〕$\leftarrow K$。

图41示出，当分别使用杂凑码2，7，1，8，2，8，1，通过算法L插入七个示

例性键（11）时发生的情况：最后三个键，FEM、SEKS和SYV，已经从它们开始的位置 $h(K)$ 转移了。

0	FEM
1	TRE
2	EN
3	
4	
5	SYV
6	SEKS
7	TU
8	FIRE

图41 线性开式寻址法

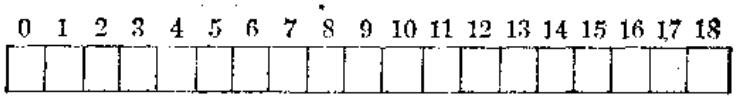
程序L（开式散列表的查找和插入） 这个程序处理全字长的键；但不允许值为0的键，因为0被用作标志表中的一个空位置（或者，我们可以要求键是非负的，让空位含－1）。假定表的大小M是质数，而且对于 $0 \leqslant i < M$ 把 TABLE〔i〕存在单元 TABLE＋i 中。为了加快内循环，假定单元 TABLE-1 包含0，假定单元 VACANCIES 包含值 $M-1-N$；且 rA $=K$，rI1$=i$。

为了加速这个程序的内循环，已经从循环中撤销了判断"$i<0$"，而仅仅保留了步骤L2和L3的必要部分。查找阶段的总共运行时间为 $(7C+9E+21-4S)u$，而在不成功查找之后的插入额外增加$8u$。

```
01   START  LDX    K              1          L1. 杂凑
02          ENTA   0              1
03          DIV    =M=            1
04          STX    *+1(0:2)       1
05          ENT1   *              1          i ← h(K)
06          LDA    K              1
07          JMP    2F             1
08   8H     INC1   M+1            E          L3. 进行到下一个
09   3H     DEC1   1              C+E-1      i ← i-1
10   2H     CMPA   TABLE,1        C+E        L2. 比较
11          JE     SUCCESS        C+E        如果K=KEY〔i〕则转出
12          LDX    TABLE,1        C+E-S
13          JXNZ   3B             C+E-S      如果TABLE〔i〕非空，则转L3
14          JN     8B             E+1-S      如果i=-1，则以i←M转L3
15   4H     LDX    VACANCIES      1-S        L4. 插入
16          JXZ    OVERFLOW       1-S        如果N=M-1，则溢出，并转出
17          DECX   1              1-S
18          STX    VACANCIES      1-S        N增加1
19          STA    TABLE,1        1-S        TABLE〔i〕←K
```

如同在程序C中那样，变量C表示探查的次数，S指出这次查找是否成功。我们可以忽略变量E，它仅当对 TABLE〔－1〕已经作了一次假的探查时为1，因为它的平均值是 $(C-1)/M$。

线性探查的经验表明，在表开始变满之前，本算法工作得十分好；但是这个过程终于减慢下来，而且长时间拖拉的查找逐渐地频繁起来。这只要考察$M=19$和$N=9$的假设的散列表，便可以明白原因：



(21)

带阴影的方格表示已占用的位置。有待插入到表中的下一个键K将进入到十个空的空间之

一，但是这些空位并不是同等可能的；事实上，如果$11 \leqslant h(K) \leqslant 15$，$K$将被插入到位置11中，同时仅当$h(K)=8$时它才进入到位置8。因此位置11的可能性，五倍于位置8；长的表列倾向于变得更长。

这个现象本身不足以看成是线性探查的一个相对拙劣的特性，因为在算法C中也出现类似的情况（一个长度为4的表列在算法C下，其增长的可能性是长度为1的表列的4倍）。当在（21）中象4或16这样的单元被填入内容时，真正的问题出现了：即将两个分开的表列组合起来，而在算法C中的表列一次决不增长一步以上。因而，当N趋于M时，线性探查的性能迅速退化。

我们将在这一节的稍后证明，算法L所需要的平均探查次数近似于

$$C'_N \approx \frac{1}{2}\left(1+\left(\frac{1}{1-\alpha}\right)^2\right) \qquad \text{（不成功的查找）} \tag{22}$$

$$C_N \approx \frac{1}{2}\left(1+\frac{1}{1-\alpha}\right) \qquad \text{（成功的查找）} \tag{23}$$

其中$\alpha=N/M$是该表的负载因子。所以当表的满载程度低于百分之75时，即使程序C处理的是不切实际地短的键，程序L仍然几乎和程序C一样快。另一方面，当α趋于1时，对于程序L我们可以作的最好的评价是，它缓慢但稳当地运行着。事实上，当$N=M-1$时，在表中仅有一个空位置，所以在一次不成功的查找中平均的探查次数是$(M+1)/2$；我们也将证明当表满时，在一次成功的查找中平均的探查次数近似于$\sqrt{\pi M/8}$。

当表快满时，堆集现象使线性探查变得代价高昂。而如果连续的键值出现可能性很大，则使用除法杂凑更加剧了这种堆集现象，因为这些键将有连续的杂凑码。乘法杂凑将令人满意地破坏这种堆集现象。

防止连续杂凑码问题的另一个方法是在步骤L3中置$i \leftarrow i-c$，而不是$i \leftarrow i-1$。c为任何正值都行，只要它与M互质，因为在这种情况下，探查序列仍将考察表中每一个位置，这样一种变化将使得程序L稍微慢些。它并不改变堆集现象，因为距离为C的记录组集仍将形成；等式（22）和（23）仍可应用，但是连续的键$\{K, K+1, K+2, \cdots\}$的出现现在实际上是一种帮助，而不是一个障碍。

尽管c的一个固定值并不减少堆集现象，但通过命c依赖于K，我们可以很好地改进这一情况！这一思想导致了对算法L的一项重要的修改，这是由盖·德·巴尔拜因(Gay de Balbine）首先发现的〔Ph. D. thesis, Calif. Inst. of Technology(1968), 149-150〕。

算法D（使用两次杂凑的开式寻址法） 这个算法和算法L几乎一样，但它以稍微不同的方式，通过使用两个杂凑函数$h_1(K)$和$h_2(K)$来探查表。和通常一样，$h_1(K)$产生0与$M-1$之间（含0和$M-1$）的一个值；但$h_2(K)$必须产生与M互质的1与$M-1$之间的一个值（例如，如果M是质数，则$h_2(K)$可以是1和$M-1$之间的任何值（包括1和$M-1$在内）；或者如果$M=2^m$，则$h_2(K)$可以是1和2^m-1之间的任何奇数值）。

D1.〔第一次杂凑〕 置$i \leftarrow h_1(K)$。

D2.〔第一次探查〕 如果TABLE〔i〕是空的，则转到D6。否则，如果KEY〔i〕$=K$，则这算法成功地结束。

D3.〔第二次杂凑〕 置$c \leftarrow h_2(K)$。

D4.〔前进到下一个〕 置 $i \leftarrow i-c$；如果现在 $i<0$，则置 $i \leftarrow i+M$。

D5.〔比较〕 如果 TABLE〔i〕是空的，则转到 D6，否则如果 KEY〔i〕$=K$，则这个算法成功地结束。否则转回 D4。

D6.〔插入〕 如果 $N=M-1$，则这个算法以溢出结束。否则置 $N \leftarrow N+1$，标志 TABLE〔i〕为已占用的，且置 KEY〔i〕$\leftarrow K$。

关于计算 $h_2(K)$ 的若干可能性已被提出。如果 M 是质数，且 $h_1(K)=K \bmod M$，则我们可以设 $h_2(K)=1+(K \bmod (M-1))$；但由于 $M-1$ 是偶数，因而命 $h_2(K)=1+(K \bmod (M-2))$ 将更好。这提示选择 M 使得 M 和 $M-2$ 好象1021和1009那样是"孪生质数"。另外，我们可以置 $h_2(K)=1+(\lfloor K/M \rfloor \bmod (M-2))$，因为商 $\lfloor K/M \rfloor$ 作为计算 $h_1(K)$ 的副产品，可在一个寄存器中利用之。

如果 $M=2^m$，且我们正在使用乘法杂凑，则只需左移 m 个以上的二进位和"或"上一个1，即可计算出 $h_2(K)$，于是代码序列（5）之后应接上

```
ENTA 0        清 rA
SUB  m        rAX 左移 m 位          (24)
OR=1=         rA←rA∨1
```

这要比除法方法更快些。

在上面提议的每一项技术当中，在下述意义下 $h_1(K)$ 和 $h_2(K)$ 是"独立的"，即不同的键对于 h_1 和 h_2 将有相同的值的概率是 $O(1/M^2)$ 而不是 $O(1/M)$。经验测试表明，具有独立的杂凑函数的算法 D 的性能，基本上也就是当键随机地插入到表中时所需要的探查次数；实际上没有如同算法 L 中那样的"拥挤"或"堆集"。

如同加里·诺特于1968年所建议的那样，也有可能让 $h_2(K)$ 依赖于 $h_1(K)$；例如，如果 M 是质数，则我们可以命

$$h_2(K)=\begin{cases} 1 & \text{如果 } h_1(K)=0 \\ M-h_1(K) & \text{如果 } h_1(K)>0 \end{cases} \tag{25}$$

这将比作另一个除法更快，但是我们将看到，它却引起一定数量的二次堆集，因为增加了两个或更多个键走同一条通路的机会，因此需要稍多些探查。以下导出的公式，可用来确定在杂凑时所获得的好处，是否超过探查时间上的损失。

算法 L 和 D 是非常类似的，但是也还有一定的差别，因此比较它们相应的 MIX 程序的运行时间是有教益的。

程序 D（使用两次杂凑的开式寻址法） 由于这个程序大体上很类似于程序 L，因此就不加注释。rI2$=c-1$。

```
01    START    LDX     K              1
02             ENTA    0              1
03             DIV     =M=            1
04             STX     *+1(0:2)       1
05             ENTI    *              1
06             LDX     TABLE,1        1
07             CMPX    K              1
```

08		JE	SUCCESS	1
09		JXZ	4F	$1-S1$
10		SRAX	5	$A-S1$
11		DIV	=M-2=	$A-S1$
12		STX	*+1(0:2)	$A-S1$
13		ENT2	*	$A-S1$
14		LDA	K	$A-S1$
15	3H	DEC1	1,2	$C-1$
16		JNN	*+2	$C-1$
17		INC1	M	B
18		CMPA	TABLE,1	$C-1$
19		JE	SUCCESS	$C-1$
20		LDX	TABLE,1	$C-1-S2$
21		JXNZ	3B	$C-1-S2$
22	4H	LDX	VACANCIES	$1-S$
23		JXZ	OVERFLOW	$1-S$
24		DECX	1	$1-S$
25		STX	VACANCIES	$1-S$
26		LDA	K	$1-S$
27		STA	TABLE,1	$1-S$

在这个程序中的频率计数A，C，$S1$，$S2$，与上面程序C中的解释相同。另一个变量B平均将是$\frac{1}{2}(C-1)$。(如果我们限制$h_2(K)$的范围为，比如说，$1\leqslant h_2(K)\leqslant\frac{1}{2}M$，则$B$大约将仅仅是$\frac{1}{4}(C-1)$；速度的这一提高大约将不足于弥补在探查次数方面的显著增加。）当在这个表中有$N=\alpha M$个键时，在一次不成功的查找中，A的平均值当然是α，而在一次成功的查找中$A=1$。如同在算法C中那样，在一次成功的查找中$S1$的平均值是$1-\frac{1}{2}((N-1)/M)\approx 1-\frac{1}{2}\alpha$。探查的平均次数难以精确地确定，但经验测试证明以下导出的公式同“一致探查”相当符合，即当$h_1(K)$和$h_2(K)$独立时，有

$$C'_N=\frac{M+1}{M+1-N}\approx(1-\alpha)^{-1}\qquad\text{(不成功的查找)}\tag{26}$$

$$C_N=\frac{M+1}{N}(H_{M+1}-H_{M+1-N})\approx-\alpha^{-1}\ln(1-\alpha)\qquad\text{(成功的查找)}\tag{27}$$

当$h_2(K)$如同在(25)中那样依赖于$h_1(K)$时，则二次堆集使这些公式增加到

$$C'_N=\frac{M+1}{M+1-N}-\frac{N}{M+1}+H_{M+1}-H_{M+1-N}+O(M^{-1})\approx(1-\alpha)^{-1}-\alpha-\ln(1-\alpha)\tag{28}$$

$$C_N=1+H_{M+1}-H_{M+1-N}-\frac{N}{2(M+1)}-(H_{M+1}-H_{M+1-N})/N+O(N^{-1})\approx 1-\ln(1-\alpha)-\frac{1}{2}\alpha\tag{29}$$

(见习题44)。注意，随着表变满，即当$N=M$时，C_N的这些值分别趋向于$H_{M+1}-1$和

$H_{M+1}-\frac{1}{2}$；这比我们在算法 L 中所发现的更好些，但不如拉链方法那样好。

由于每次探查花费时间比算法 L 稍少，故双杂凑仅当表变满时才有利。图42比较了程序 L、程序 D以及修改后的程序 D的平均运行时间，修改后的程序 D包含二次堆集，并以三条指令

ENN2	$-1-M$, 1	$c \leftarrow M-i$	
J1NZ	$*+2$		(30)
ENT2	0	如果 $i=0$ $c \leftarrow 1$	

来代替行10～13中 $h_2(K)$ 的相当慢的计算。程序 D 总共花费 $8C+19A+B+26-13S-17S1$ 时间单位；在一次成功的查找中（30）的修改大约节省这些时间单位中的 $15(A-S1)\approx 7.5\alpha$。在这种情况下，二次堆集比独立的两次杂凑更为可取。

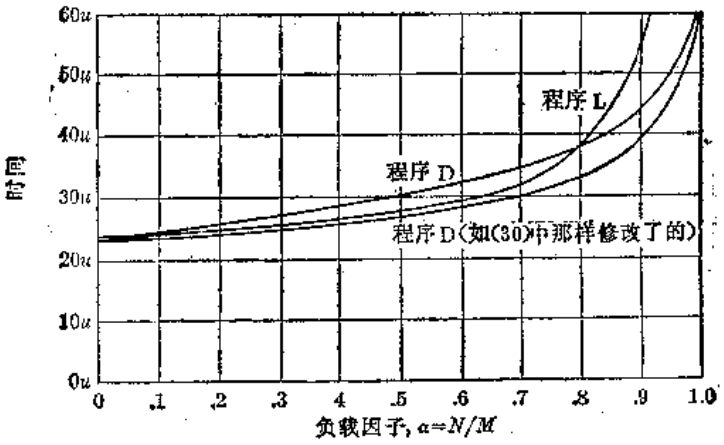


图42 通过三个开式寻址方案进行成功查找的运行时间

在一台二进计算机上，我们可以以另一种方式来加速 $h_2(K)$ 的计算，比如说如果 M 是一个大于 512 的质数，则以

AND	=511=	rA←rA mod 512	
STA	$*+1$ (0:2)		(31)
ENT2	*	$c \leftarrow$ rA+1	

来代替行10～13。这一思想（它是由贝尔（Bell）和卡曼（Kaman）提出的，*CACM* 13 (1970)，675-677，他们独立地发现了算法 D）无须用任何除法就避免了二次堆集。

已经提出许多其它的探查序列来改进算法 L，但看来没有一个比算法 D优越，可能习题20中描述的方法除外。

布伦特的变形 理查德·P·布伦特（Richard P. Brent）已经发现了一个修改算法 D 的方法，使得当表变满时平均的成功查找时间保持有界。他的方法是以这样一个事实为基础的，即在许多应用中，成功的查找比插入要更频繁得多；因此，他建议在插入一项目时，多做一些工作，包括移动表中的记录，以便减少期望的查找时间〔*CACM*15(1972)〕。

例如，图43示出了在一个典型的 PL/1 过程中，每个标识符真正被查到的次数。这一思想指出，使用一个杂凑表记住变量名字的 PL/1 编译程序，将查找许多名字五次以上，

但插入只一次。类似地，贝尔和卡曼发现，一个COBOL编译程序在编译一个程序时，使用它的符号表算法10988次，但只对该表作了735次插入；就是每作一次不成功的查找平均大约要作14次成功的查找。有时一个表实际上仅被建立一次（例如，在一个汇编程序中的符号操作码表），而此后纯粹用作检索。

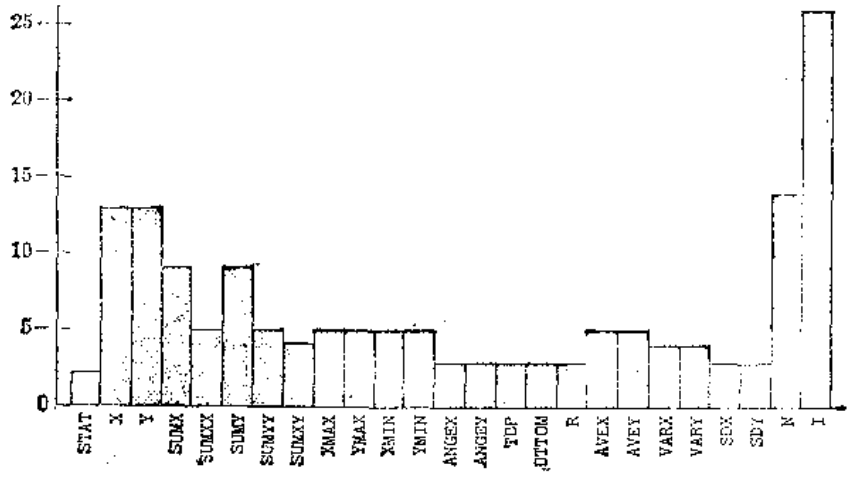


图43　一个编译程序典型地查找变量名的次数，诸名字按照它们头一次出现的次序自左到右地列出

布伦特的思想是把算法D中的插入过程改变如下。假设一次不成功的查找要探查单元

$$p_0,\ p_1,\ \cdots p_{t-1},\ p_t$$

其中 $p_j=(h_1(K)-jh_2(K))\bmod M$，且TABLE〔$p_t$〕为空。如果 $t\leqslant 1$，则我们象通常那样把K插入到位置 p_t 中；但如果 $t\geqslant 2$，则计算 $c_0=h_2(K_0)$，其中 $K_0=$KEY〔p_0〕，而且看TABLE〔$(p_0-c_0)\bmod M$〕是否为空。如果为空，则置它为TABLE〔p_0〕，而后把K插入到位置 p_0 中，这增多了一步对于 K_0 的查找时间，但它减少了 $t\geqslant 2$ 步对于K的查找时间，所以结果还是合算的。类似地，如果TABLE〔$(p_0-c_0)\bmod M$〕是已占用的，而且 $t\geqslant 3$，则我们尝试TABLE〔$(p_0-2c_0)\bmod M$〕；如果这也是满的，则计算$c_1=h_2$(KEY〔p_1〕)和试验TABLE〔$(p_1-c_1)\bmod M$〕；等等。一般地说，命 $c_j=h_2$(KEY〔p_j〕)且$p_{j,k}=(p_j-kc_j)\bmod M$；如果我们发现TABLE〔$p_{j,k}$〕对于使得 $j+k<r$ 的所有下标 j，k是已占用的，而且如果 $t\geqslant r+1$，则考察TABLE〔$p_{0,r}$〕，TABLE〔$p_{1,r-1}$〕，…，TABLE〔$p_{r-1,1}$〕。如果头一个可用空间出现于位置 $p_{j,r-j}$ 处，则我们就置TABLE〔$p_{j,r-j}$〕←TABLE〔p_j〕，而且把K插入位置 p_j 中。

布伦特的分析指出，正如后面的图44中所示的那样，每次成功的查找的平均探查数是减少了，极大值大约为2.49。

布伦特的变形并不减少在不成功查找中探查的次数 $t+1$，它保持等式(26)所指出的水平，当表满时，趋于$\frac{1}{2}(M+1)$。每次插入需要计算的平均次数 h_2是$\alpha^2+\alpha^5+\frac{1}{3}\alpha^6+\cdots$，按照布伦特的分析，最终趋于$\sqrt{M}$的阶；而且在判断如何来进行插入时额外被探查的表中位置数大约是$\alpha^2+\alpha^4+\frac{4}{3}\alpha^5+\alpha^6+\cdots$。

E. G. 马拉兹（E. G. Mallach）〔Comp. J20(1977)，137-140〕已经以布伦特的变形的

改进作了实验，而加斯顿·H. 康内特(Gaston H. Gonnet)和 J. 伊恩·芒罗(J. Ian Munro)已经得到了一些进一步的结果〔SIAM J. Computing 8 (1979)，463-478〕。

删去 许多计算机程序员都高度地信任算法，他们惊奇地发现从散列表删去记录的一种显然的方式居然失效。例如，如果我们想从图41删去键 EN，则不能简单地把表中这个位置标成空的，因为这将使另一个键 FEM 突然地被忘记！（还记得吗，EN 和 FEM 两者是杂凑到同一单元的。当考察 FEM 时，我们将发现一个空的位置，从而表示一次不成功的查找。）由于诸表的接合，对于算法 C必出现类似的问题；试想象从图40中删去 TO 和 FIRE 两者。

一般地说，我们可以通过置一个特殊的代码值于对应的单元中来处理删去，使得有三类表的项：空的、已占用的以及已删去的。当查找一个键时，我们将跳过已删去的单元，就象它们是已占用的那样。如果查找是不成功的，则这个键可被插入到头一个删去的或者空的位置中。

但是这一思想仅当删去是非常稀少时才是可行的，这是因为一旦表的项变为已占用的，则它们就决不再次地变空。在进行了一长串重复的插入和删去的序列之后，所有可用空间都终将消失，而且每次不成功的查找将花费M个探查！其次每个探查的时间将增加，因为我们要测试 i 是否已经恢复成它在步骤 D4 中的开始值；而且在一次成功的查找中探查的次数将趋于从 C_N 上升到 C'_N。

当正在使用的是线性探查（即算法 L）时，如果我们愿做某些额外工作，则在删去时可避免上述麻烦情况。

算法 R(用线性探查时的删去) 假定通过算法 L 已经构造了一个开式散列表，本算法从一个给定的位置 TABLE〔i〕删去记录。

R1.〔弄空一个单元〕 标记 TABLE〔i〕为空，且置 $j \leftarrow i$。

R2.〔i 减值〕 置 $i \leftarrow i-1$，如果这使得 i 成为负的，则置 $i \leftarrow i+M$。

R3.〔检查 TABLE〔i〕〕 如果 TABLE〔i〕是空的，则这算法结束。否则置 $r \leftarrow h$(KEY〔i〕)，这个键原来的杂凑地址现在存在位置 i 中。如果 $i \leqslant r < j$ 或如果 $r < j < i$ 或 $j < i \leqslant r$（换言之，如果 r 循环地位于 i 和 j 之间），则转回到 R2。

R4.〔移动一个记录〕 置 TABLE〔j〕←TABLE〔i〕并返回步骤 R1。

习题22表明这个算法不引起性能上的蜕化，即，在等式（22）和（23）中预测的平均探查次数保持相同（定理 6.2.2H 中证明了对于树插入的一个较弱的结果）。但是算法 R 的正确性与这样一个事实密切相关，即它是用于线性探查的，而且不可能有类似的对算法 D 可用的删去过程。

当然，当对每个可能的杂凑值使用分开的表并实行拉链时，删去不会引起问题，因为它仅仅是从一个链接的线性表的删去。习题23中讨论了算法C中的删去。

算法 R可能移动表中的某些项，但如果有其它某处是指向这些项的，则这种移动是不希望发生的。通过采用废料回收中使用的某些思想，有可能有另外一个删去的方法（参考 2.3.5节）：对于每个键我们可以保持一个“访问计数”说明有多少其它的键同它接触；然后当某个未占用单元的访问计数为 0 时，就有可能把它转换成空状态。另一种方法是，每当已经累计有太多的删去的项时，我们可以扫视整个的表，同时转换所有未占用的位置成

为空状态，而后考察所有剩余的键，为的是找出哪些未占用的位置真正需要“删去”状态。这个过程最初是由 T. 冈吉（T. Gunji）和 E. 戈托（E. Goto）提出的（待发表），它避免使用浮动，且对任何杂凑技术都有效。

对算法的分析 了解一个杂凑方法的平均特性是特别重要的，因为每当进行杂凑时，我们只能依靠概率的法则。这些算法最坏的情况几乎都不可想象地坏，所以我们需要得到保证：平均特性还是很好的。

在进行对线性探查等等的分析之前，让我们考虑这一状况的非常近似的模型，即所谓的均匀的杂凑（参考 W. W. Peterson，IBM J. Research & Development (1957)，135-136）。在这个模型中，我们假定诸键所占的表中位置是随机的，使得N个已占用单元及$M-N$个空单元的$\binom{M}{N}$种可能的配置是同等可能的。这个模型忽略一次堆集和二次堆集的任何影响；表中每个单元的占用基本上同其它的单元无关。对于这一模型，为插入第$N+1$项恰恰需要r次探查的概率，是$r-1$个给定的单元已占用和另一个单元是空的这样的配置数，除以$\binom{M}{N}$。即

$$P_r=\binom{M-r}{M-r+1}\Big/\binom{M}{N}$$

因此，对于均匀的杂凑的平均探查次数是

$$\begin{aligned}C'_N&=\sum_{1\leqslant r\leqslant M} rP_r=M+1-\sum_{1\leqslant r\leqslant M}(M+1-r)P_r\\&=M+1-\sum_{1\leqslant r\leqslant M}(M+1-r)\binom{M-r}{M-N-1}\Big/\binom{M}{N}\\&=M+1-\sum_{1\leqslant r\leqslant M}(M-N)\binom{M+1-r}{M-N}\Big/\binom{M}{N}\\&=M+1-(M-N)\binom{M+1}{M-N+1}\Big/\binom{M}{N}\\&=M+1-(M-N)\frac{M+1}{M-N+1}=\frac{M+1}{M-N+1}，\text{对于 } 1\\&\leqslant N<M\end{aligned}\tag{32}$$

（我们曾基本上解决了习题3.4.2-5中同随机抽样相联系的同一个问题）。置$\alpha=N/M$，C'_N的这一精确公式近似地等于

$$\frac{1}{1-\alpha}=1+\alpha+\alpha^2+\alpha^3+\cdots\tag{33}$$

这个级数有一种粗略的直观解释：我们需要一次以上的探查的概率为α，二次以上的探查的概率为α^2，等等。对于一次成功的查找对应的平均探查次数是

$$\begin{aligned}C_N&=\frac{1}{N}\sum_{0\leqslant k<N}C'_k=\frac{M+1}{N}\left(\frac{1}{M+1}+\frac{1}{M}+\cdots+\frac{1}{M-N+2}\right)\\&=\frac{M+1}{N}(H_{M+1}-H_{M-N+1})\approx\frac{1}{\alpha}\log\frac{1}{1-\alpha}\end{aligned}\tag{34}$$

如同上边所说的，广泛的测试表明，在所有的实际应用中，具有两个独立的杂凑函数的算法D的性能实质上都象均匀杂凑一样。事实上，利奥·吉巴斯（Leo Guibas）和恩德雷·森默勒迪（Endre Szemerédi）在证明这困难的定理方面已经成功，即双杂凑当$M\to\infty$时其极限近似地等价于均匀的探索〔J. Comp. Syst. Sci. 16(1978)，226-274〕。

这就完成了对均匀杂凑的分析。为了研究线性探查和其它类型的解决冲突的办法，我们需要以不同的更现实的方式来建立这个理论。为此，我们采用的概率模型假定：M^N个可能的"杂凑序列"的每一个

$$a_1a_2\cdots a_N \qquad 0\leqslant a_j<M \tag{35}$$

是同等可能的，其中a_j表示插入到表中的第j个键的初始杂凑地址。给定任何特殊的查找方法，在一次成功的查找中探查的平均次数将如同上面那样以C_N表示；假定这是为求第K个键所需要的探查的平均次数（平均是对所有$1\leqslant k\leqslant N$取的），而且每个键都是同等可能的。类似地，当第N个键被插入时，所需要的探查的平均次数，对所有的序列（35）来说是同等可能的，将以C'_{N-1}表示之；这是在以表中的$N-1$个元素开始的不成功的查找中，探查的平均次数。当使用开式寻址法时

$$C_N=\frac{1}{N}\sum_{0\leqslant k<N}C'_k \tag{36}$$

于是我们可以象在（34）中所做的那样从一个量导出另一个量。

严格地说，即使在这个更为精确的模型中也有两个不足之处。首先，不同的杂凑序列不是全都同等可能的，因为诸键本身是不同的。这使得$a_1=a_2$的概率稍微小于$1/M$；但这个差别通常是可忽略的，因为所有可能的键的集合相对于M说来是非常大的（见习题24）。其次，一个好的杂凑函数将利用典型数据的非随机性，使它甚至更不能有$a_1=a_2$；结果，我们对于探查次数的估计将是悲观的。在这个模型中的另一点不精确性，已在图43中指出：较早出现的键（除某些例外者外），比后来出现的键更可能被查找，因此我们对于C_N的估计更加悲观，而这些算法的实际运行情况应该比我们的分析所预言的稍好些。

在打过这些招呼之后，我们已经准备好了对于线性探查作一个"精确"的分析●。设$f(M,\ N)$是在算法L插入了诸键之后，使得表的位置0成为空的杂凑序列（35）的个数。线性探查的循环对称性意味着位置0为空的可能性恰如其它位置一样，所以它为空的概率是$1-N/M$；换言之，

$$f(M,\ N)=\left(1-\frac{N}{M}\right)M^N \tag{37}$$

由约定，我们也置$f(0,\ 0)=1$。

现在设$g(M,\ N,\ k)$是使得这算法保持位置0为空、位置1到k是已占用的、位置$k+1$为空的杂凑序列（35）的个数，我们有

$$g(M,\ N,\ k)=\binom{N}{k}f(k+1,\ k)f(M-k-1,\ N-k) \tag{38}$$

● 在这里作者禁不住想要插进一段关于历史的说明：我于1962年在开始撰写《计算机程序设计撰巧》之后不久，就已写出了下面的推导。由于这是我所深感满意的分析过的头一个非平凡的算法，因此它对于这些书的结构已经有了强烈的影响。但是我当时却未曾预料到，居然在过了十年多之后，这些推导才能付印。

因为所有这样的杂凑序列都是由两个子序列组成的，一个序列（包含 k 个元素 $a_i \leqslant k$）保持位置 0 为空，并保持位置 1 到 k 是已占用的，另一个序列（包含 $N-k$ 个元素 $a_j \geqslant k+1$）保持位置 $k+1$ 为空；前一种类型有 $f(k+1,\ k)$ 个子序列，后一种类型有 $f(M-k-1,\ N-k)$ 个子序列，而且有 $\binom{N}{k}$ 种方式来散开两个这样的子序列。最后，设 P_k 是当插入第 $N+1$ 个键时，恰巧需要 $k+1$ 个探查的概率；由此得出（见习题25）

$$P_k = M^{-N}(g(M,\ N,\ k) + g(M,\ N,\ k+1) + \cdots + g(M,\ N,\ N)) \tag{39}$$

现在，$C'_N = \sum_{0 \leqslant k \leqslant N} (k+1)P_k$；把这个等式同（36）～（39）放在一起并简化之，即得下列结果。

定理 K 假定所有 M^N 个杂凑序列（35）是同等可能的，则算法 L 所需要的平均探查次数是

$$C_N = \frac{1}{2}(1 + Q_0(M,\ N-1)) \qquad \text{（成功的查找）} \tag{40}$$

$$C'_N = \frac{1}{2}(1 + Q_1(M,\ N)) \qquad \text{（不成功的查找）} \tag{41}$$

其中

$$\begin{aligned} Q_r(M,\ N) &= \binom{r}{0} + \binom{r+1}{1}\frac{N}{M} + \binom{r+2}{2}\frac{N(N-1)}{M^2} + \cdots \\ &= \sum_{k \geqslant 0} \binom{r+k}{k} \frac{N}{M} \cdot \frac{N-1}{M} \cdots \frac{N-k+1}{M} \end{aligned} \tag{42}$$

证明 计算的细节在习题 27 中给出。

在这个定理中出现的看起来稍微有些奇怪的函数 $Q_r(M,\ N)$，实际上并不难处理。我们有

$$N^k - \binom{k}{2} N^{k-1} \leqslant N(N-1)\cdots(N-k+1) \leqslant N^k$$

因此如果 $N/M = \alpha$，则

$$\sum_{k \geqslant 0} \binom{r+k}{k} \left(N^k - \binom{k}{2} N^{k-1}\right) / M^k \leqslant Q_r(M,\ N) \leqslant \sum_{k \geqslant 0} \binom{r+k}{k} N^k / M^k$$

$$\sum_{k \geqslant 0} \binom{r+k}{k} \alpha^k - \frac{\alpha}{M} \sum_{k \geqslant 0} \binom{r+k}{k} \binom{k}{2} \alpha^{k-2} \leqslant Q_r(M,\ \alpha M) \leqslant \sum_{k \geqslant 0} \binom{r+k}{k} \alpha^k$$

即

$$\frac{1}{(1-\alpha)^{r+1}} - \frac{1}{M}\binom{r+2}{2} \frac{\alpha}{(1-\alpha)^{r+3}} \leqslant Q_r(M,\ \alpha M) \leqslant \frac{1}{(1-\alpha)^{r+1}} \tag{43}$$

当 M 很大，且 α 不太接近于 1 时，这个关系给了我们对于 $Q_r(M,\ N)$ 的一个好的估计（下界是比上界更好的近似）。当 α 趋于 1 时，这些公式就变成没有用了，但幸而 $Q_0(M,\ M-1)$ 是函数 $Q(M)$，它的渐近特性在 1.2.11.3 节中已被详细地研究过了。而且，$Q_1(M,\ M-1)$ 就等于 M（见习题50）。

分析线性探查的另一个方法，已为小 G. 沙伊和 W. G. 斯普鲁特（W. G. Spruth）得到〔*CACM* 5 (1962), 459-462〕。尽管他们的方法仅产生对于定理 K 中的精确公式的一个

近似，但它却对这个算法给出了进一步的分析，所以在这里我们将简单地概述一下。首先，让我们考虑由 W. W. 彼得森于 1957 年首先指出的线性探查的一个令人惊异的性质。

定理 P 通过算法L进行的一次成功的查找中，探查的平均次数同键被插入的次序无关；它仅依赖于杂凑成每个地址的键的数目。

换言之，一个杂凑序列 a_1，a_2，…，a_N 的任何重新排列，产生另一个杂凑序列，它的诸键相对于杂凑地址的平均偏移和原序列一样。（如同早先指出的，我们假定在这个表中的所有键都有相同的重要性。如果某个键比其它键更频繁地被访问，则可以扩充本定理并证明，通过定理 6.15 的方法，以频率的递减的次序插入它们，是一种最优的安排。）

证明 只需证明，对于杂凑序列 $a_1a_2\cdots a_k$，为插入键所需要的探查的总次数，和对于 $a_1\cdots a_{i-1}a_{i+1}a_ia_{i+2}\cdots a_n$，$1\leqslant i<N$ 所需要的总次数是相同的。显然，除开在第二个序列中的第 $i+1$ 个键落入到在第一个序列中由第 i 个键占据的位置外，两者没有差别。但第 i 个和第 $i+1$ 个仅仅交换了位置，所以对于第 $i+1$ 个的探查次数减少的数量，等同于对第 i 个的探查次数增加的数量。

定理P告诉我们，一个杂凑序列 $a_1a_2\cdots a_N$ 的平均查找长度，可由数 $b_0b_1\cdots b_{M-1}$ 确定之，其中 b_j 是等于 j 的诸 a 的个数。从这个序列我们可以确定“进位序列” $c_0c_1\cdots c_{M-1}$，其中 c_j 是这样的一些键的个数，在这些键被插入时单元 j 和 $j+1$ 都被探查。这个序列通过规则

$$c_j=\begin{cases}0, & \text{如果 } b_j=c_{(j+1)\bmod M}=0\\ b_j+c_{(j-1)\bmod M}-1, & \text{否则}\end{cases}\tag{44}$$

确定。例如，设 $M=10$，$N=8$ 以及 $b_0\cdots b_9=0\ 3\ 2\ 0\ 1\ 0\ 0\ 0\ 2$；则 $c_0\cdots c_9=2\ 3\ 1\ 0\ 0\ 0\ 0\ 1\ 2\ 3$，因为一个键需要从位置 2 “进位”到位置 1，三个键需要从位置 1 进到位置 0，其中的两个从位置 0 进到位置 9，等等。我们有 $b_0+b_1+\cdots+b_{M-1}=N$，而且为了查找 N 个键所需要的平均探查次数是

$$1+(c_0+c_1+\cdots+c_{M-1})/N\tag{45}$$

规则（44）似乎是用 c 来定义自己的一种循环定义，但实际上每当 $N<M$ 时，所述的方程就有唯一的一个解（见习题32）。

沙伊和斯普鲁特使用了这一思想，借助于 $b_j=k$ 的概率 p_k，来确定 $c_j=k$ 的概率 q_k（这些概率同 j 无关）。于是

$$\begin{aligned}q_0&=p_0q_0+p_1q_0+p_0q_1\\ q_1&=p_2q_0+p_1q_1+p_0q_2\\ q_2&=p_3q_0+p_2q_1+p_1q_2+p_0q_3\end{aligned}\tag{46}$$

等等，因为例如，$c_j=2$ 的概率是 $b_j+c_{(j+1)\bmod M}=3$ 的概率。设 $B(z)=\Sigma p_kz^k$ 和 $C(z)=\Sigma q_kz^k$ 是这些概率分布的母函数；等式（46）等价于

$$B(z)C(z)=p_0q_0+(q_0-p_0q_0)z+q_1z^2+\cdots=p_0q_0(1-z)+zC(z)$$

由于 $B(1)=1$，我们可以写 $B(z)=1+(z-1)D(z)$，而且由此得出

$$C(z)=\frac{p_0q_0}{1-D(z)}=\frac{1-D(1)}{1-D(z)}\tag{47}$$

因为 $C(1)=1$。根据（45），为进行查找所需要的平均探查次数，将是

$$1+\frac{M}{N}C'(1)=1+\frac{M}{N}\frac{D'(1)}{1-D(1)}=1+\frac{M}{2N}\frac{B''(1)}{1-B'(1)} \tag{48}$$

由于我们假定了每一个杂凑序列 $a_1\cdots a_N$ 是同等可能的，所以有

$$\begin{aligned} p_k &= \Pr(\text{对于固定的 } j\text{，恰有 } k \text{ 个 } a_i \text{ 等于 } j) \\ &= \binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k} \end{aligned} \tag{49}$$

因此

$$B(z)=1+\left(\frac{z-1}{M}\right)^N,\quad B'(1)=\frac{N}{M},\quad B''(1)=\frac{N(N-1)}{M^2} \tag{50}$$

而且根据（48），平均探查次数将是

$$C_N=\frac{1}{2}\left(1+\frac{M-1}{M-N}\right) \tag{51}$$

读者能否看出，这答案为什么不同于定理K中的结果（参考习题33）？

***最优性考虑** 我们已经看到若干关于开式寻址法的探查序列的例子，因而自然要问：能否找到一个在某种意义下是“最优”的序列？这一问题已经由J.D.厄尔曼（J.D. Ullman）以下列有趣的方式提出来〔*JACM* 19（1972），569-575〕：我们不去计算一个“杂凑地址” $h(K)$，而是把每个键K都映射到$\{0, 1, \cdots, M-1\}$的一整个排列中，这排列表示对K使用的探查序列。对于$M!$个排列的每一个赋予一个概率，而且提出一个广义的杂凑函数，它以指定的概率来选择相应的列。问题是：“在其对应的平均探查次数 C_N 或 C'_N 为极小这样的意义下，对于诸排列选定什么样的概率，将给出最好的性能？”

例如，如果对每个排列选定 $1/M!$ 的概率，则容易看出我们恰巧有在（32）、（34）中已经分析过的均匀杂凑的特性。然而，厄尔曼已经找到了对于$M=4$，$N=2$的一个例子，对于它 C'_N 小于由均匀杂凑得到的值 $\frac{5}{3}$。他的构造是，对于除下列6个排列之外的所有排列，都选定概率为0：

排列	概率	排列	概率
0 1 2 3	$(1+2\epsilon)/6$	1 0 3 2	$(1+2\epsilon)/6$
2 0 1 3	$(1-\epsilon)/6$	2 1 0 3	$(1-\epsilon)/6$
3 0 1 2	$(1-\epsilon)/6$	3 1 0 2	$(1-\epsilon)/6$

(52)

粗略地说，第一种选择喜欢2和3，但第二种选择喜欢0或1。为插入第三项所需要的平均探查次数是 $\frac{5}{3}-\frac{1}{9}\epsilon+O(\epsilon^2)$。所以我们通过取$\epsilon$为一个小的正值，可以改进均匀杂凑。

然而，对于这些概率对应的 C'_1 的值是 $\frac{23}{18}+O(\epsilon)$，它大于5/4（均匀杂凑的值）。厄尔曼已经证明，如果把概率选得使 $C'_N<(M+1)/(M+1-N)$ 对于某个N成立，则定有某个 $n<N$，使 $C'_n>(M+1)/(M+1-n)$；你不可能在所有时候都胜过均匀杂凑。

实际上，一次成功的查找的探查次数 C_N，是比 C'_N 更好的量度。（52）中的排列对于任何的N都不能改进。似乎有理由猜测，对概率的任何选定，都不能使 C_N 小于均匀的值 $((M+1)/N)\times(H_{M+1}-H_{M+1-N})$。

这个猜测似乎非常难以证明，特别是因为有许多方法可选定概率以达到均匀杂凑的效

果；我们不需要对每个排列选定 $1/M!$。例如，对 $M=4$ 的下列选定就等价于均匀杂凑：

排列	概率	排列	概率
0 1 2 3	1/6	0 2 1 3	1/12
1 2 3 0	1/6	1 3 2 0	1/12
2 3 0 1	1/6	2 0 3 1	1/12
3 0 1 2	1/6	3 1 0 2	1/12

(53)

同时选定其它 16 个排列的概率为 0。

下列定理表征了产生均匀杂凑特性的所有选定。

定理 U 下列两件事是等价的：

（1）当 $0<N<M$ 时，选定诸排列的相应概率，使得在进行 N 次插入之后，在空单元和被占用单元的 $\binom{M}{N}$ 种配置中，每种都是同等可能的。

（2）对所有的 N 和所有的 N 个元素的集合，考察所有这样的排列，这些排列的头 N 个元素即某个 N 个元素集合的成员，对这些排列选定的概率之和为 $1/\binom{M}{N}$。

例如，对于以数 $\{0, 1, 2\}$ 的某种顺序开始的 $3!(M-3)!$ 个排列中每个排列选定的概率之和必然是 $1\Big/\binom{M}{3}=3!(M-3)!/M!$。注意，这个定理的条件在（53）中成立。

证明 设 $A\subseteq\{0, 1, \cdots, M-1\}$，且设 $\Pi(A)$ 是其头 $\|A\|$ 个元素为 A 的成员的所有排列之集合，且设 $S(A)$ 是对于这些排列选定的概率之和。设 $P_k(A)$ 是在执行开式寻址法过程中头 $\|A\|$ 次插入占用由 A 确定的位置，且最后一次插入恰需 k 次探查的概率；设 $P(A)=P_1(A)+P_2(A)+\cdots$。本证明对 $N\geqslant 1$ 用归纳法，假定对于满足 $\|A\|=n<N$ 的所有集合 A 有

$$P(A)=S(A)=1\Big/\binom{M}{N}$$

设 B 是任意 N 个元素的集合，则

$$P_k(B)=\sum_{\substack{A\subseteq B\\ \|A\|=k}}\ \sum_{\pi\in\Pi(A)}\Pr(\pi)P(B/\{\pi_k\})$$

其中 $\Pr(\pi)$ 是对排列 π 选定的概率，而 π_k 是它的第 k 个元素。由归纳法

$$P_k(B)=\sum_{\substack{A\subseteq B\\ \|A\|=K}}\frac{1}{\binom{M}{N-1}}\sum_{\pi\in\Pi(A)}\Pr(\pi),$$

它等于

$$\binom{N}{k}\Big/\binom{M}{N-1}\binom{M}{k}\qquad \text{如果 } k<N$$

因此

$$P(B)=\frac{1}{\binom{M}{N-1}}\left(S(B)+\sum_{1\leqslant k<N}\frac{\binom{N}{k}}{\binom{M}{k}}\right)$$

当且仅当$S(B)$有正确的值时它等于$1\Big/\binom{M}{N}$。

外部查找 杂凑技术很有助于对磁盘或磁鼓这样的直接存取存储设备进行外部查找。在这样的应用中，如同在6.2.4节中那样，我们要把对文件的存取次数极小化，这对于算法的选择有两项主要的影响：

1）花费较多的时间计算杂凑函数是值得的，因为坏的杂凑函数带来的损失，比为进行一项仔细的工作所需要的额外时间代价要大得多。

2）通常都把记录组织成桶，以便每次从外存取出若干个记录。

文件通常分成为每个包含b个记录的M个桶。现在除非b个以上的键有相同的杂凑地址，不然冲突不会造成问题。下列三种解决冲突的方法似乎是最好的：

A）用分开的表列拉链 如果有多于b个记录落到同一桶当中，则可在第一个桶的末尾插入一个通向“溢出”记录的链接。这些溢出记录放在一个特殊的溢出区域中。一般来说，在溢出区域中建立桶并没有优点，因为溢出较少出现；于是，额外的记录通常被链接在一起，使得一个表列的第（$b+k$）个记录需要$1+k$次存取。一种好的想法是在一个磁盘文件的每个柱面上都保留某些溢出空间，使得大多数存取是对于相同柱面进行的。

尽管处理溢出的这个方法似乎是低效的，但溢出的次数从统计上说是够小，以致平均查找时间非常好。见表2和表3，它示出了当M，$N\to\infty$时，作为固定负载因子α

$$\alpha=N/Mb \tag{54}$$

的函数的平均存取次数。奇怪的是，当$\alpha=1$时，不成功的查找的渐近存取次数竟随b而增加。

表2 通过分开拉链进行一次不成功查找的平均存取数

桶大小，b	负载因子α 10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0048	1.0187	1.0408	1.0703	1.1065	1.1488	1.197	1.249	1.307	1.3
2	1.0012	1.0088	1.0260	1.0581	1.1036	1.1638	1.238	1.327	1.428	1.5
3	1.0003	1.0038	1.0162	1.0433	1.0898	1.1588	1.252	1.369	1.509	1.6
4	1.0001	1.0016	1.0095	1.0314	1.0751	1.1476	1.253	1.394	1.571	1.7
5	1.0000	1.0007	1.0056	1.0225	1.0619	1.1346	1.249	1.410	1.620	1.7
10	1.0000	1.0000	1.0001	1.0041	1.0222	1.0773	1.201	1.426	1.773	2.0
20	1.0000	1.0000	1.0000	1.0001	1.0028	1.0234	1.113	1.367	1.898	2.3
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0007	1.018	1.182	1.920	2.7

表3 通过分开拉链进行一次成功查找的平均存取数

桶大小，b	负载因子α 10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0500	1.1000	1.1500	1.2000	1.2500	1.3000	1.350	1.400	1.450	1.5
2	1.0063	1.0242	1.0520	1.0883	1.1321	1.1823	1.238	1.299	1.364	1.4
3	1.0010	1.0071	1.0216	1.0458	1.0806	1.1259	1.181	1.246	1.319	1.4
4	1.0002	1.0023	1.0097	1.0257	1.0527	1.0922	1.145	1.211	1.290	1.3
5	1.0000	1.0008	1.0046	1.0151	1.0358	1.0699	1.119	1.186	1.286	1.3
10	1.0000	1.0000	1.0002	1.0015	1.0070	1.0226	1.056	1.115	1.206	1.3
20	1.0000	1.0000	1.0000	1.0000	1.0005	1.0038	1.018	1.059	1.150	1.2
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.015	1.083	1.2

B）通过接合表列进行拉链　我们可修改算法C，使之适用于外部文件，以代替分开的溢出区域，使用可利用空间的双重拉链表列，可把所有还未满的桶链接在一起。在这个方案中，每个桶包含一个计数，它表明桶中有多少位置为空，仅当计数变成0时，这个桶才从双重链接表中撤销。可用"活动指针"分配溢出（参考习题2.5-6），使得不同的表列趋于使用不同的溢出桶。对于一个磁盘文件的每个柱面上的桶设置各自的可利用空间表列，可能是一种好想法。

这个方法还未曾分析过，但它将被证明是十分有用的。

C）开式寻址法　我们也可以不用链，而使用"开式"方法。当考虑外部查找时，线性探查大概比随机探查更好，因为通常可以选择增量c使它把连续存取间的等待延迟时间极小化。可推广上面作出的线性探查的近似理论模型，以考虑桶的影响，它表明，除非这个表已经变得非常满，否则线性探查确实是令人满意的。例如，参见表4；当负载因子是百分之90且桶的大小是50时，在一次成功的查找中平均存取次数只是1.04。这实际上比对于相同大小的桶使用拉链方法（A）所需要的1.08次存取更好！

表4　线性探查在一次成功查找中的平均存取次数

桶大小，b	负载因子α 10%	20%	30%	40%	50%	60%	70%	80%	90%	95%
1	1.0556	1.1250	1.2143	1.3333	1.5000	1.7500	2.167	3.000	5.500	10.5
2	1.0062	1.0242	1.0553	1.1033	1.1767	1.2950	1.494	1.903	3.147	5.6
3	1.0009	1.0066	1.0231	1.0450	1.0872	1.1584	1.286	1.554	2.378	4.0
4	1.0001	1.0021	1.0058	1.0227	1.0497	1.0984	1.190	1.386	2.000	3.2
5	1.0000	1.0007	1.0039	1.0124	1.0307	1.0663	1.136	1.289	1.777	2.7
10	1.0000	1.0000	1.0001	1.0011	1.0047	1.0154	1.042	1.110	1.315	1.8
20	1.0000	1.0000	1.0000	1.0000	1.0003	1.0020	1.010	1.036	1.144	1.4
50	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.001	1.005	1.040	1.1

对于方法（A）和（C）的分析涉及某些非常有趣的数学；我们在这里仅仅概述其结果，因为细节已在习题49和55中加以研究。这些公式涉及到同定理K的Q函数密切有关的两个函数，即

$$R(\alpha,\ n)=\frac{n}{n-1}+\frac{n^2\alpha}{(n+1)(n+2)}+\frac{n^3\alpha^2}{(n+1)(n+2)(n+3)}+\cdots \tag{55}$$

以及

$$\begin{aligned} t_n(\alpha)&=e^{-n\alpha}\left(\frac{(\alpha n)^n}{(n+1)!}+2\frac{(\alpha n)^{n+1}}{(n+2)!}+3\frac{(\alpha n)^{n+2}}{(n+3)!}+\cdots\right)\\ &=\frac{e^{-n\alpha}n^n\alpha^n}{n!}(1-(1-\alpha)R(\alpha,\ n)) \end{aligned} \tag{56}$$

若用这些函数来表示，则在一次不成功的查找中通过拉链方法（A）所作的平均存取次数当M，$N\to\infty$时是

$$C'_N=1+\alpha b t_b(\alpha)+O\left(\frac{1}{M}\right) \tag{57}$$

而在一次成功的查找中对应的次数是

$$\begin{aligned} C_N=1&+(e^{-b\alpha}b^b\alpha^b/2b!)(2+(\alpha-1)b+(\alpha^2+(\alpha-1)^2(b\\ &-1))R(\alpha,\ b))+O(1/M) \end{aligned} \tag{58}$$

这些是在表 2 和表 3 中所示的量。

由于拉链方法A需要一个分开的溢出区，我们需要估计将出现多少溢出。溢出的平均次数将是$M(C'_N-1)=Nt_b(\alpha)$，因为C'_N-1是在任何给定的表列中平均的溢出次数。因此表 2 可用来推导出所需要的溢出空间的数量。对于固定的α，当$M\to\infty$时，溢出的总数的标准离差将大致同$\sqrt{M}$成正比。

C'_N和C_N的渐近值出现在习题 53 中，但当b很小或α很大时，这些近似不是非常好的；幸而，$R(\alpha, n)$的级数收敛速度甚至当α很大时也颇快。所以这些公式可以没有多大困难地精确计算出来。由斯特林近似公式和 1.2.11.3 节中对函数$R(n)=R(1, n)-1$的分析，当$b\to\infty$时，在

$$\max C'_N=1+\frac{e^{-b}b^{b+1}}{b!}=\sqrt{\frac{b}{2\pi}}+1+O(b^{-1/2}) \tag{59}$$

$$\max C_N=1+\frac{e^{-b}b^b}{2b!}(R(b)+1)=\frac{5}{4}+\sqrt{\frac{2}{9\pi b}}+O(b^{-1}) \tag{60}$$

的条件下，对于$\alpha=1$出现极大值。

在通过线性探查所作的一次成功的外部查找中，平均的存取次数有相当简单的表达式

$$C_N\approx 1+t_b(\alpha)+t_{2b}(\alpha)+t_{3b}(\alpha)+\cdots \tag{61}$$

它可以理解如下：考查所有N个键的平均总访问次数是NC_N，此即$N+T_1+T_2+\cdots$，其中T_k是需要多于k次访问的键的平均个数。定理P指出，我们可以以任何顺序记入键而不影响C_N，并由此得出，如果我们有大小为kb的M/k个桶，则T_k是在拉链方法中出现的溢出记录的平均个数，如果按我们上边所说的，也就是$Nt_{kb}(\alpha)$。关于等式(11)的进一步论证在习题 55 中。

查尔斯·A. 奥尔森（Charles A. Olson）已经精采地论述了在外部散列表的设计中所涉及的实用性的考虑，参见*Proc. ACM Nat'l Conf.* 24(1969)，537-549。他列举了若干实践过的例子，并且指出，如果文件受到经常的插入/删去活动的影响而不浮动记录，则溢出记录的次数将大量增加；而且他给出了对于这一情况的分析，这是在J. A. 德·佩伊斯特（J. A. de Peyster）的参与下得到的。

诸方法比较 我们现在已经研究了大量的用于查找的技术；怎样选出正适合于给定应用的一个方法呢？很难以几句话概括在选择一个查找方法时所涉及的有关"折衷"的细节，但是相对于查找速度和所需的存储空间说来，下列诸事似乎是具有第一位重要性的。

图 44 概括了这一节的分析，并表明了解决冲突的各种方法导致的不同的探查次数，但是这并未说出事情的全部，因为在不同的方法中每次探查的时间也不同，而且方法的变化对于运行时间有相当大的影响（如同我们在图 42 中见到的）。线性探查比图 44 中所示的其它方法更经常地访问表，但是它的优点是简单，而且线性探查也不坏得可怕：当表有90%满时，为把一个随机项放置在表中，算法L平均需要少于 5.5 次的探查（然而，当表90%满时，通过算法L插入一个新的项所需要的平均探查次数是 50.5。）

图 44 表明，按探查次数来说，拉链方法是十分经济的，但因为链接场需要额外的存储空间，故有时开式寻址法对于小记录更有吸引力。例如，如果我们需要在容量为 500 的

一个拉链散列表和容量为1000的一个开式散列表之间进行选择，则后者显然是更可取的。因为当存有500个记录时，它的查找效率比较高，而且它有能力吸收两倍的数据。另一方面，有些记录的大小和格式实际上无需额外的代价就能为链接场提供空间（参见习题65）。

怎样把杂凑方法同我们在本章中已研究过的其它查找策略作比较呢？从速度的观点看，我们可以论证，当记录个数很大时，杂凑法更好些，因为如果假设表不会太满，则当$N\to\infty$时，一个杂凑方法的平均查找时间保持有界。例如，当表有90%满时，对于一次成功的查找，程序L将仅仅花费大约55个时间单位；当N大于600左右时，这比我们见到的最快的MIX二分查找程序要强，而代价仅为多花11%的存储空间（见习题6.2.1-24）。而且，二分查找仅适合于固定的表，而一个散列表却允许有效的插入。

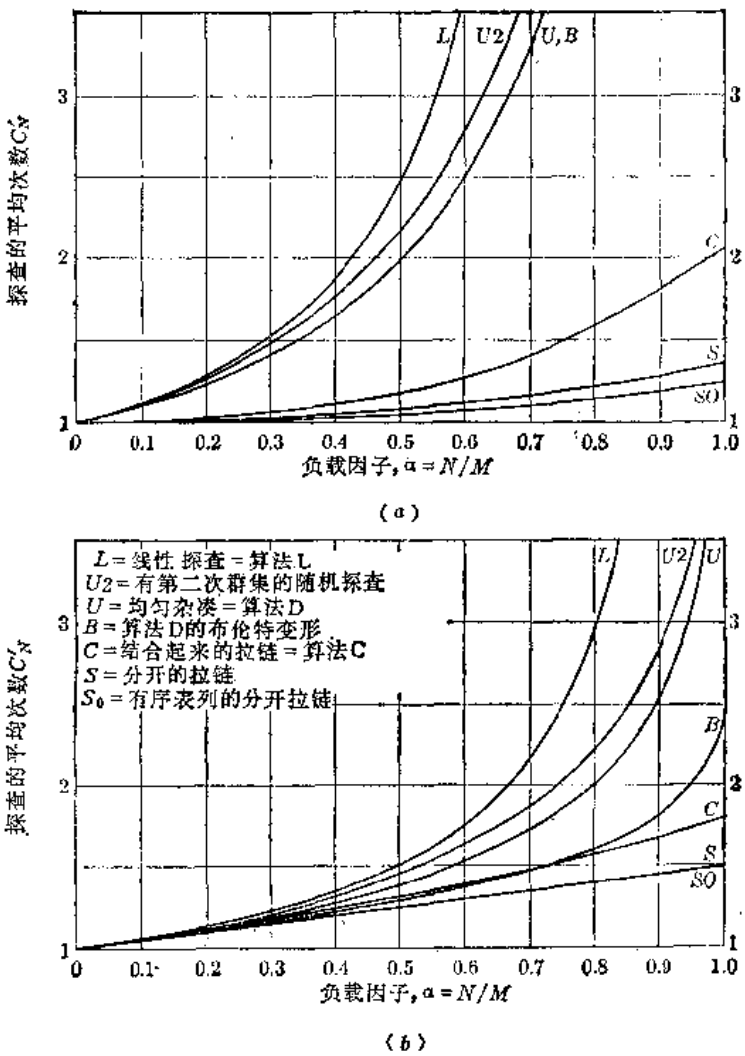


图44 解决冲突诸方法的比较，当$M\to\infty$时平均的探查次数的极限值（a）不成功的查找；（b）成功的查找。

我们也可以把程序L同允许动态插入的面向树的查找方法进行比较，当N大于约90时，对于90%满的表，程序L比程序6.2.2T更快；而当N大于约75时，它也比程序6.3D更快（习题6.3-9）。

对于成功的查找说来，这一章中仅有一个查找方法是有效且实际上没有存储开销的，这就是算法D的布伦特变形。他的方法允许我们把N个记录放到大小为$M=N+1$的一个表中，而且平均以大约$2\frac{1}{2}$次探查找出任何记录，链接场不需要额外的空间等等；然而，对于不成功的查找将是非常缓慢的，需要大约$\frac{1}{2}N$次的探查。

因此，杂凑有若干优点。另一方面，在三个重要的方面，散列表的查找比我们已经讨论的其它方法要差：

a）在散列表中一次不成功的查找之后，我们仅仅知道所希望的键不存在。以比较为

基础的查找方法总是产生更多的信息，使得有可能找出$\leqslant K$的最大键和/或$\geqslant K$的最小键；这在许多应用中（例如对一个存储好的函数值表进行内插）是重要的。也有可能使用以比较为基础的算法，来找出处于两个给定的值K和K'之间的所有键。利用6.2节的树查找方法可很容易地以递增的次序来输出一个表的内容，而无须单独排序，有时这是很有用的。

b）散列表的存储分配通常都有些困难；我们需要提供某一个存储区域来存杂凑表，但应当分配多少空间可能是不明显的。如果我们提供了太多的内存，则可能浪费其它表列或其它计算机用户范围内的存储；但如果不提供足够的空间，则这个表会溢出。当散列表溢出时，可能最好是"重新杂凑"它，即分配一个更大的空间，并改变杂凑函数，把每个记录重新插入更大的表中。F. R. A. 霍普古德（F. R. A. Hopgood）〔*Comp. Bulletin* 11 (1968)，297-300〕已经建议，当表变成百分之α满时，重新杂凑这个表，以d_0M代替M；通过使用上述的分析和数据的特征，可以适当选择参数α_0和d_0，使得有可能确定一个较有利的临界点，重新杂凑即从此点开始。（注意，拉链的方法并不导致任何麻烦的溢出，所以它不需要重新杂凑；但当M是固定的且N变得很大时，查找时间同N成正比。）反之，树查找和插入算法不需要这样费力的重新杂凑；树不会增长得比需要的更大。在一个虚拟的存储环境中，我们大概应该使用树查找或数字树查找，而不是去建立一个很大的散列表，因为散列表几乎在每次杂凑一个键时，就需要取一个新的页进来。

c）最后，当我们使用杂凑方法时，需要对概率论有巨大的信赖，因为杂凑法仅仅平均来说是有效的，而最坏的情况则是可怕的！与随机数生成程序的情况类似，我们决不完全保证当把一个杂凑函数应用于一个新的数据集合时，它能工作得很好。因此散列存储对于某些实时应用，如空间交通控制，这类生命攸关的事将是不适当的。6.2.3节和6.2.4节的平衡树算法是更为安全的，因为它们的查找时间的上界是有保证的。

历史 杂凑的思想似乎是由H. P. 吕恩创立的，他于1953年1月写了一篇IBM的内部备忘录，建议使用拉链；这也是拉链线性表的最初应用之一。他提出了在外部查找中以使用包含一个以上元素的桶为好。不久以后，A. D. 林（A. D. Lin）把吕恩的分析推进一步，并且提出了使用"退化地址"处理溢出的技术；例如，假定存在10000个初级桶，1000个二级桶，100个三级桶，等等，则初级桶2748产生的溢出置于二级桶274中；从该桶产生的溢出转到三级桶27中，等等。吕恩最初提出的杂凑函数是字符数字的，例如，把键数字的相邻对加起来并mod10，使得31415926被压缩成4548。

大约同一时间，杂凑的思想独立地出现于IBM的另一伙成员当中。他们是：吉恩·M. 阿姆达尔（Gene M. Amdahl）、伊莱恩·M. 贝姆（Elaine M. Boehme）、N. 罗彻斯特（N. Rochester）以及阿瑟·L. 塞缪尔（Arthur L. Samuel），他们编制了对于IBM701的一个汇编程序。为了处理冲突问题，阿姆达尔创立了线性探查的开式寻址法思想。

杂凑代码首先是由阿诺德·J. 杜米第一个在公开的文献中描述的，参见"*Computers and Automation*" 5, 12 (December 1956)，6-9。他第一个提出除以一个质数和使用余数作为杂凑地址的思想。杜米有趣的论文提到了拉链但没有开式寻址法。苏联的A. P. 厄索夫(A. P. Ershow)于1957年独立地发现了线性开式寻址法〔*Doklady Akad. Nauk SSSR* 118 (1958)，427-430〕；他发表了关于探查次数的经验结果，正确地猜测到，当$N/M<2/3$时，每一成功的查找的平均探查次数<2。

W·W·彼得森的一篇经典的文章（见 *IBM J. Research & Development* 1 (1957), 130-146），乃是第一篇讨论在大型文件中的查找问题的重要文章。彼得森一般地定义了开式寻址法，分析了均匀杂凑的性能，而且对于各种桶的大小给出了关于线性开式寻址法的各种经验统计，说明当删去项目时出现的性能上的蜕化。六年之后，沃纳·巴克霍尔兹〔*IBM System J.* 2 (1963), 86-111〕发表了另一篇关于这个课题的广泛的综述，他给出了关于杂凑函数的一个特别好的讨论。

到这时为止，线性探查是出现于文献中的唯一的一类开式寻址法方案，还有一个通过独立的杂凑函数进行重复的随机探查方案是由另外一些人独立地提出来的（见习题48）。在之后数年中杂凑广泛地被使用了，但是几乎没有发表任何更多的东西，然后罗伯特·莫里斯（Robert Morris）写了一篇关于这个课题的非常有影响的综述〔*CACM* 11 (1968), 38-44），其中他介绍了（有二次堆集的）随机探查的思想。莫里斯的文章引起了一阵研究热潮，并以算法D及其改进而达到高潮。

说明这样一点是有趣的，具有目前的意义的"杂凑"（hashing）一词，在60年代末期以前，似乎完全没有见诸文字当中，尽管那时在世界的若干地区它已经变成了普遍的术语。这个字似乎曾经出在H·赫勒曼（H. Hellerman）的书"*Digital Computer System Principles*"（New York: McGraw-Hill, 1967），第152页中。在撰写这一节时，我所研究的近60篇有关的文献当中，这一词唯一地出现于1961年由W. W. 彼得森所写的一篇未发表的备忘录中。在60年代中期，不知怎么回事，"杂凑"(to hash) 这一动词竟魔术般地变成了关于键转换的标准术语，但是在1967年以前却没有人敢很轻率地公开使用这样一个不庄重的词。

习题

1.〔20〕 假定K的字节1，2，3包含少于30个字母字符代码，当达到表1中的指令9H时，r11的内容可以多大和多小？

2.〔20〕 找出可以附加到表1而又无须改变程序的相当常用的英文字。

3.〔23〕 说明为什么对任何常数a，以下列五条指令开始的程序

```
LD1  K(1:1)或 LD1N K(1:1)
LD2  K(2:2)或 LD2N K(2:2)
     INC1  a, 2
     LD2   K(3:3)
     J2Z   9F
```

都不能用来代替表1中更复杂的程序，因为对于给定的键将不能产生唯一的地址。

4.〔M30〕 为了使得很可能有三个人有相同的生日，应该有多少人参加一个晚会？

5.〔15〕 B. C. 达尔曾经用一台十进的MIX计算机写一个FORTRAN编译程序，而且他需要一个符号表来记住正在被编译的FORTRAN程序中的变量名字。这些名字的长度限制成至多10个字符。他决定使用$M=100$的一个散列表，而且使用快速杂凑函数$h(K)=K$的最左字节。这是一个好的想法吗？

6.〔15〕 把（3）的头两条指令改变成为LDA K；ENTX 0，是明智的吗？

7.〔$HM30$〕（多项式杂凑） 本题的目的是考虑如象（10）那样的多项式$P(x)$的构造，它把n位键转换成为m位地址，任何两个不相同的键，如果其区别不超过七位，则它们一定被杂凑成不同的地址。给定n和$t\leqslant n$，并给定一个整数k，使得n整除α^k-1，我们将构造一个其次数m是n，t和k的函数的多项式（通常在必要时增大n使得把k选择成相当小）。

设S是使得$\{1, 2, \cdots, t\}\subseteq S$的整数的最小集合，而且设对所有$j\in S$，$(2j)\bmod n\in S$。例如，当$m=15$，$k=4$，$t=6$时，我们有$S=\{1, 2, 3, 4, 5, 6, 8, 10, 12, 9\}$。现在定义多项式$P(x)=\pi_{j\in S}(x-\alpha^j)$，其中$\alpha$是有限域$GF(2^k)$中阶为$n$的一个元素，而且$P(x)$的系数是在这个域中计算的。$P(x)$的次数$m$是$S$的元素的个数。因为当$\alpha^j$是$P(x)$的一个根时，$\alpha^{2j}$也是它的一个根，由此得出$P(x)$的系数$P_i$满足$P_i^2=P_i$，所以它们全都为0或1。

试证明，如果$R(x)=r_{n-1}x^{n-1}+\cdots+r_1x+r_0$是任意非0多项式modulo 2，而且至多有$t$个非0的系数，则$R(x)$不是$P(x)$的一个倍数modulo 2〔由此得出，对应的杂凑函数的行为就象已宣告的那样。

8.〔$M34$〕（三距离定理） 设0是0和1之间的无理数，在4.5.3节的记号下它的连分式表示是$\theta=/a_1, a_2, a_3, \cdots/$。设$q_0=0$，$p_0=1$，$q_1=1$，$p_1=0$，对于$k\geqslant 1$有$q_{k+1}=a_kq_k+q_{k-1}$，$P_{k+1}=a_kP_k+P_{k-1}$。设$\{x\}$表示$x\bmod 1=x-\lfloor x\rfloor$，且设$\{x\}^+$表示$x-\lceil x\rceil+1$。随着点$\{\theta\}$，$\{2\theta\}$，$\{3\theta\}$，…逐次插入到区间〔0，1〕中，而把线段按它们出现的方式进行编号，使得一个给定长度的第一段是编号0，第二段是1，等等。试证明下列命题全为真：长度为$\{t\theta\}$且编号为s的区间，其左端点为$\{s\theta\}$，右端点为$\{(s+t)\theta\}^+$，其中$t=rq_k+q_{k-1}$和$0\leqslant r<a_k$且k为偶数以及$0\leqslant s<q_k$。长度为$1-\{t\theta\}$且编号为s的区间，其左端点为$\{(s+t)\theta\}$，右端点为$\{s\theta\}^+$，其中$t=rq_k+q_{k-1}$，和$0\leqslant r<a_k$且k为奇数及$0\leqslant s<q_k$。每个正整数n均可唯一地表示成$n=rq_k+q_{k-1}+s$，其中$k\geqslant 1$，$1\leqslant r\leqslant a_k$，$0\leqslant s<q_k$。借助这个表示，在点$\{n\theta\}$被插入之前，已有的$n$个区间为

长度$\{(-1)^k(rq_k+q_{k-1})\theta\}$的头$s$个区间（编号$0, \cdots, s-1$）；

长度$\{(-1)^kq_k\theta\}$的头$n-q_k$个区间（编号为$0, \cdots, n-q_k-1$）；

长度$\{(-1)^k((r-1)q_k+q_{k-1})\theta\}$的最后$q_k-s$个区间（编号为$s, \cdots, q_k-1$）插入$\{n\theta\}$的操作撤销编号为$s$的最后一种类型的区间，并把它转换成编号为$s$的第一种类型的区间，以及编号为$n-q_k$的第二种类型的区间。

9.〔$M30$〕 当我们逐次地把点$\{\theta\}$，$\{2\theta\}$，…插入到区间〔0，1〕中时，定理S断言每个新的点总是分割剩下的最大的区间之一。如果把此区间〔a，c〕分成两个部分〔a，b〕，〔b，c〕，而其中的一个部分大于另一部分的两倍长，即若$b-a>2(c-b)$或$c-b>2(b-a)$，则我们称它为一个坏的分割。

证明除非$\theta\bmod 1=\phi^{-1}$或ϕ^{-2}，对于某个$\{n\theta\}$将出现坏的分割，而当$\theta\bmod 1=\phi^{-1}$或ϕ^{-2}时，将决不产生坏的分割。

10.〔$M38$〕（R. L. 格雷厄姆） 证明或反驳以下$3d$距离的猜测：如果$\theta, \alpha_1, \cdots, \alpha_d$是实数且$\alpha_1=0$，$n_1, \cdots, n_d$是正整数，此外如果对于$0\leqslant n<n_i$，$1\leqslant i\leqslant d$，点

$\{n\theta+\alpha_j\}$被插入到区间[0，1]中，则得到的$n_1+\cdots+n_d$（可能为空）个区间至多有$3d$个不同的长度。

11.〔16〕 成功的查找通常比不成功的查找要更为频繁。如果因此而把程序C的行12～13同行10～11加以交换，这是否为一个好想法？

▶12.〔21〕 证明程序C可被重写成使得在内部循环中仅有一个条件转移。把修改了的程序同原来的程序进行比较。

▶13.〔24〕（略写的键） 设$h(K)$是一个杂凑函数，且设$q(K)$是K的一个函数，使得一旦已经给定$h(K)$和$q(K)$，便可确定K。例如，在除法杂凑中，我们可以设$h(K)=K \bmod M$及$q(K)=\lfloor K/M \rfloor$；在乘法杂凑中可以设$h(K)$是$(AK/w) \bmod 1$的前几个二进位，而$q(K)$是其它的二进位。

试证明，当使用拉链而不使用重叠的表列时，在每个记录中我们仅需存储$q(K)$而不是K（这几乎节省了链接场所需要的空间）。试修改算法C使其避免重叠的表列，以允许这样略写的键，而且对于"溢出"的记录还不使用辅助存储单元。

14.〔24〕（E.W.埃尔科克（E.W.Elcock））证明有可能让一个很大的散列表同任意多个其它拉链表共享内存。设这个表列区域的每个字有一个2位二进位的TAG字段以及两个称作LINK和AUX的链接字段，并有下列的解释：

TAG(P)＝0表示可用空间表列中的一个字；LINK(P)指向该表中的下一项，AUX(P)不用。TAG(P)＝1表示正在使用的任何字，其中P不是散列表中任何键的杂凑地址；单元P中字的其它字段可以有任何需要的格式。

TAG(P)＝2表示P至少是一个键的杂凑地址；AUX(P)指向确定所有这样键的一个链接表，LINK(P)指向这个表存储器中的另一个字。在处理任何表期间，每当访问带有TAG(P)＝2的一个字时，有必要重复地置P←LINK(P)，直到达到具有TAG(P)≤1的一个字为止。（为了效率，我们也可以然后改变以前的链，使得今后不需要一再地跳过这些散列表的项。）

说明在这类的一个组合表中，为插入和检索键，怎样定义一个适当的算法。

15.〔16〕 为什么算法L和算法D当$N=M-1$而不是当$N=M$时警告溢出是一个好想法？

16.〔10〕 程序L指出，K不应为0。但当K为0时，它是否就真正不能工作了？

17.〔15〕 当$h_1(K)\neq 0$时，为什么不简单地在（25）中定义$h_2(K)=h_1(K)$？

▶18.〔21〕 作为程序D的行10～13的一个替换，（31）比（30）好些还是坏些？试根据A，$S1$和C的平均值，给出你的答案。

19.〔40〕 试凭经验测试以下列方式限制算法D中$h_2(K)$的范围的效果：a）对于$r=1, 2, 3, \cdots 10$，$1\leqslant h_2(K)\leqslant r$；b）对于$\rho=\frac{1}{10}, \frac{2}{10}, \cdots, \frac{9}{10}$，$1\leqslant h_2(K)\leqslant \rho M$。

20.〔M25〕（R.克鲁塔（R.Krutar））改变算法D如下以免去杂凑函数$h_2(K)$：在步骤D3中置$c\leftarrow 0$；且在步骤D4的开始置$c\leftarrow c+1$。试证明，如果$M=2^m$，则对应的探查序列$h_1(K)$，$(h_1(K)-1) \bmod M$，…，$(h_1(K)-\binom{M}{2}) \bmod M$将是$\{0, 1, \cdots, M$

-1)的一个排列。假定这个方法的特性与具有二次堆集的随机探查一样，则当把它编成MIX程序时，该程序与图42中考虑的三个程序比起来是好还是坏？

▶21.〔20〕 假设我们希望从由算法D构造的一个表中删去一个记录，并如同正文中所建议的那样，标记它为"删去的"。试问，我们也应该把用以支配算法D的变量N减值吗？

22.〔27〕 证明算法R使表保持好象它未曾在开头的位置被插入过KEY(i)那样。

▶23.〔33〕 试设计类似于算法R的一个算法，用于从通过算法C构造的一个拉链散列表中删去项。

24.〔M20〕 假设所有可能出现的键的集合有MP个元素，其中恰有P个键杂凑到一个给定的地址（在实际情况下，P是非常大的；例如当诸键为任意的十进数字时，如果$M=10^3$，则我们有$P=10^7$）假定$M\geqslant 7$且$N=7$。如果从所有可能的键的集合中随机地选择七个不同的键，则得到的杂凑序列为1 2 6 2 1 6 1（即$h(K_1)=1$，$h(K_2)=2$，…，$h(K_7)=1$）的精确概率（作为M和P的函数）等于多少？

25.〔M19〕 说明等式（39）为什么是正确的？

26.〔M20〕 利用线性探查时，有多少杂凑序列产生如（21）那样的已占用单元的型式。

27.〔M27〕 完成定理K的证明〔提示：设

$$s(n,x,y)=\sum_{k\geqslant 0}\binom{n}{k}(x+k)^{k+1}(y-k)^{n-k-1}(y-n)$$

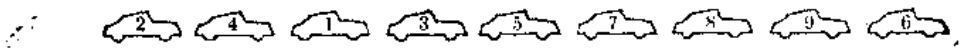
使用阿贝尔二项式定理，即等式1.2.6-16，来证明$s(n,\ x,\ y)=x(x+y)^n+ns(n-1,\ x+1,\ y-1)$〕。

28.〔M30〕 过去的计算机比今天的慢得多，有可能观看指示灯的闪烁，并了解算法L运行多快。当表开始填满时，某些项将非常快地被处理，而其它的则花费大量的时间。

这个经验提示，当使用线性探查时C'_N的标准离差是相当高的。试求一个公式，它借助于在定理K中定义的θ_r函数表达方差，并估计当$N=\alpha M$且$M\to\infty$时的方差。

29.〔M21〕（停车问题）某单行道有排成一行的m个停放车辆的位置，其编号为1到m。一个人和他的打瞌睡的妻子驱车过街，突然他妻子醒过来并命令他立即停车，他顺从地把车停在第一个可利用的位置；但如果没有剩下位置了，则他可以不必后退（即如果在他车子达到位置k时，他的妻子才醒过来，但位置K，$K+1$，…，m全都满了），他表示歉意并继续行驶。

事实上，假设对n辆不同的小汽车发生此事，其中第j个妻子刚巧在停车位置a_j时醒过来。假定这条街道开始时是空的，而且在停车之后没有人离开，试问有多少个序列$a_1\cdots a_n$使所有的小汽车都能安全地停放？例如，当$m=n=9$和$a_1\cdots a_9=3\ 1\ 4\ 1\ 5\ 9\ 2\ 6\ 5$时，这些小汽车停放如下：



〔提示：使用对线性探查的分析。〕

30.〔$M28$〕（约翰·赖尔登）在习题29的停车问题中，当$n=m$时，试证明所有的小汽车都能停放的充要条件是存在$\{1, 2, \cdots, n\}$的一个排列$p_1p_2\cdots p_n$，使得对所有的j，$a_j\leqslant P_j$。

31.〔$M40$〕在习题29的停车问题中，当$n=m$时，解的个数已证明是$(n+1)^{n-1}$；而且从习题2.3.4.4-22我们知道这和具有$(n+1)$个带标号顶点的自由树的个数相同！试求在停车序列和树之间的有趣联系。

32.〔$M26$〕证明每当$b_0, b_1, \cdots, b_{M-1}$是和数小于$M$的非负整数时，方程组(44)有唯一解$(c_0, c_1, \cdots, c_{M-1})$。试设计出求这个解的算法。

▶33.〔$M23$〕说明(51)为什么仅仅是由算法L所作的真正的平均探查次数的一个近似。

▶34.〔$M22$〕本题的目的是研究：当诸表列象(38)那样保持分开时，在一个拉链的散列表中的平均探查次数。(a)求P_{NK}，即当M^N个杂凑序列(35)同等可能时，一个给定的表列有长度k的概率。(b)求母函数$P_N(z)=\sum_{k\geqslant 0}P_{Nk}z^k$。(c)借助于这个母函数，表达出一次成功的查找所需的平均探查次数。(d)推导出在一次不成功的查找中的平均探查次数，并考虑采用下列约定的数据结构变形：(i)杂凑总是对一个表头进行(参考图38)；(ii)杂凑是对表的一个位置进行(参考图40)，但除了一个表的开头之外所有的键进入到一个分开的溢出区域；(iii)杂凑是对表的一个位置进行而且所有项都出现于杂凑表中。

35.〔$M24$〕继续习题34，当各表列按照它们的键值依次地排列时，在一次不成功的查找当中平均探查次数是多少？考虑数据结构(i)、(ii)和(iii)。

36.〔$M23$〕继续习题34(d)，当查找不成功时，使用数据结构(i)和(ii)，求探查次数的方差。

▶37.〔$M29$〕公式(19)给出当查找成功时在分开的拉链中探查的平均次数；这个量的方差是多少？

38.〔$M32$〕（树杂凑）一个机灵的程序员可以尝试使用二分查找树以代替拉链方法中的线性列表，由此把算法6.2.2T与杂凑结合起来。试分析通过这个复合的算法，对于成功和不成功的查找两者所需要的平均探查次数。〔提示：参看式5.2.1-11。〕

39.〔$M27$〕设$c_N(k)$是当算法C应用于所有M^N个杂凑序列(35)时形成的长度为k的列表总数。求关于数$c_N(k)$的一个递归关系使得有可能确定对于和

$$S_N=\sum_k\binom{k}{2}c_N(k)$$

的一个简单公式。S_N与用算法C作不成功查找时所需平均探查数的关系如何？

40.〔$M33$〕公式(15)给出在一次不成功的查找中由算法C所使用的平均探查次数，这个量的方差是多少？

41.〔$M40$〕分析T_N，即在通过算法C插入第$N+1$个项目时R值减1的平均次数。

▶42.〔$M20$〕推导(17)。

43.〔$M44$〕试分析使用大小为$M'\geqslant M$的一个表对算法C进行的一项修正。在这项

修正中仅仅前M个单元被用于杂凑，所以在步骤C5中找到的前$M'-M$个空的节点将在这个表的额外单元中。对于固定的M'，在$1\leqslant M\leqslant M'$范围中选择什么样的M方能导致最好的性能？

44.〔M43〕（有二次堆集的随机探查） 这个习题的目标是确定具有探查序列

$$h(K),\ (h(K)+p_1)\bmod M,\ (h(K)+p_2)\bmod M,\ \cdots,\ (h(K)+p_{M-1})\bmod M$$

的开式寻址方案中预期的探查次数，其中$p_1p_2\cdots p_{M-1}$是依赖于$h(K)$的一个随机选择的$\{1,\ 2,\ \cdots,\ M-1\}$的排列。换言之，具有相同$h(K)$值的所有键都遵循相同的探查序列，而且具有这一性质的M个探查序列的$(M-1)!^M$种可能的选择都是同等可能的。

这一情况可以通过对开始时是空的大小为m的线性阵列实施的下列实验来精确地模拟。进行如下操作n次：

> 以概率p，占用最左的空位置，否则(即以概率$q=1-p$)，选择表中除最左者外的任意位置，而且这$m-1$个供选择的位置是同等可能的。如果选出的位置是空的，则占用它；否则选择任何空的位置（包括最左边的）并占用它，同时认为每一个空的位置是同等可能的。

例如，当$m=5$和$n=3$时，在上述的实验之后的阵列配置将为（已占用，已占用，空，已占用，空）的概率是：

$$\frac{7}{192}qqq+\frac{1}{6}pqq+\frac{1}{6}qpq+\frac{11}{64}qqp+\frac{1}{3}ppq+\frac{1}{4}pqp+\frac{1}{4}qpp$$

(这个过程对应于当$p=1/m$时的有二次堆集的随机探查，因为我们可以把表的项重新编号使得一个特殊的探查序列是0，1，2，…而所有其它的都是随机的。)

试求出在这个阵列左边的已占用位置的平均数的一个公式（即，在上例中的2）。并求当$p=1/m$，$n=\alpha(m+1)$和$m\to\infty$时，这些量的渐近值。

45.〔M43〕 当探查序列以$h_1(K)$，$(h_1(K)+h_2(K))\bmod M$开始，且随后的探查仅仅依赖于$h_1(K)$和$h_2(K)$的随机选择时，求解具有三次堆集的习题44的类似情况（于是，具有这一性质的$M(M-1)$个探查序列的$(M-2)!^{M(M-1)}$种可能的选择，被认为是同等可能的)。这个过程渐近地等价于均匀探查吗？

46.〔M42〕 试确定使用探查序列

$$h(K),\ 0,\ 1,\ \cdots,\ h(K)-1,\ h(K)+1,\ \cdots,\ M-1$$

的开式寻址方法的C'_N和C_{N_c}

47.〔M25〕 当探查序列是

$$h(K),\ h(K)-1,\ h(K)+1,\ h(K)-2,\ h(K)+2,\ \cdots$$

时，试求为开式寻址所需要的平均探查数。这个探查序列曾一度被提出，因为当M为偶数时，连续探查之间的所有距离都是不同的。〔提示：找窍门，这个问题很容易。〕

▶48.〔M21〕 给定互相独立的随机杂凑函数$h_n(K)$的一个无穷序列，试分析探查位置$h_1(K)$，$h_2(K)$，$h_3(K)$，…的开式寻址方法。注意，有可能探查同一个位置两次，例如如果$h_1(K)=h_2(K)$，但这是很少可能的。

49.〔HM24〕 推广习题34到每个桶有b个记录的情况，对于具有分开的表列的拉

链，确定其平均的探查次数（即文件的访问次数）C_N 和 C'_N，其中假定在一次不成功的查找中含有 k 个元素的一个表列需要 $\max(1,\ k-b+1)$ 次探查。我们不象在习题 34 中那样使用精确的概率 P_{Nk}，而使用泊松近似

$$\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k}=\frac{N}{M}\cdot\frac{N-1}{M}\cdots\frac{N-k+1}{M}\left(1-\frac{1}{M}\right)^N\left(1-\frac{1}{M}\right)^{-k}\frac{1}{k!}=e^{-\rho}\rho^k/k!+O(M^{-1})$$

当 $M\to\infty$ 时，它对于 $N=\rho M$ 和 $k\leqslant\sqrt{M}$ 成立。

50.〔$M20$〕 证明，在（42）的记号下，$Q_1(M,\ N)=M-(M-N-1)Q_0(M,N)$。〔提示：首先证明 $Q_1(M,\ N)=(N+1)Q_0(M,\ N)-NQ_0(M,\ N-1)$〕

51.〔$HM16$〕 借助于在（42）中定义的函数 Q_0，来表达在（55）中定义的函数 $R(\alpha,\ n)$。

52.〔$HM20$〕 证明 $Q_0(M,\ N)=\int_0^\infty e^{-t}(1+t/M)^N dt$。

53.〔$HM20$〕 证明函数 $R(\alpha,\ n)$ 可借助于不完备的伽马函数表达，并使用习题 1.2.11.3-9的结果求出当 $n\to\infty$ 时，对于固定的 $\alpha<1$，$R(\alpha,\ n)$ 的渐近值直到 $O(n^{-2})$。

54.〔40〕 当算法C已经象正文中所述那样经过修改后用来进行外部查找时，试实验它的特性。

55.〔$HM43$〕 推广在定理P之后讨论的沙伊—斯普鲁特模型到大小为 b 的 M 个桶的情况，证明 $C(z)$ 等于 $Q(z)/(B(z)-z^b)$，其中 $Q(z)$ 是次数为 b 的多项式且 $Q(1)=0$。证明探查的平均次数是

$$1+\frac{M}{N}C'(1)=1+\frac{1}{b}\left(\frac{1}{1-q_1}+\cdots+\frac{1}{1-q_{b-1}}-\frac{1}{2}\cdot\frac{B''(1)-b(b-1)}{B'(1)-b}\right)$$

其中 $q_1,\ \cdots,\ q_{b-1}$ 是 $Q(z)/(z-1)$ 的根。以泊松近似式 $P(z)=e^{b\alpha(z-1)}$ 代替二项式概率分布 $B(z)$，其中 $\alpha=N/bM$，使用拉格朗日求逆公式（参考等式2.3.4.4-9和习题4.7-8），简化你的答案成为等式（61）。

56.〔$M48$〕 推广定理K，得到对于使用大小为 b 的桶的线性探查之精确分析。

57.〔$M47$〕 若均匀地选定各探查序列的概率，是否能使 C_N 的值相对于所有的开式寻址方法来说成为极小。

58.〔$M21$〕（S.C.约翰逊（S.C.Johnson））求出在定理U的意义下等价于均匀杂凑的 $\{0,\ 1,\ 2,\ 3,\ 4\}$ 的 10 个排列。

59.〔$M25$〕 证明，如果选定排列的一个概率，它在定理U的意义下等价于均匀杂凑，则当 M 充分大时，对于任何固定的指数 a，具有非 0 概率的排列数超过 M^a。

60.〔$M48$〕 如果一个开式寻址方案恰好使用 M 个探查序列，其中每个序列从 $h(k)$ 的一个可能的值开始，而每一个可能的值都以 $1/M$ 的概率出现，则称此方案为单杂凑方案。

问（在 C_N 为极小的意义下）最好的单杂凑方案是否渐近地比在习题 44 中分析的随机方案更好？

61. 〔$M46$〕 在习题 46 中分析的方法，是否是最坏的单杂凑方案（参考习题60)？

62. 〔$M49$〕 当在习题 44 的记号下，增量 $p_1p_2\cdots p_{M-1}$ 对于所有的K都固定时，单杂凑方案的性能如何？（这样的方法的例子，是线性探查和在习题 20 和 47 中考虑的序列。）

63. 〔$M25$〕 如果在散列表中反复地进行随机的插入和删去，则平均需要作多少次独立的插入，才能使所有M个位置至少被占用过一次？

64. 〔$M46$〕 试分析算法 R 的预期的特性。平均将实施步骤 R4多少次？

▶65. 〔20〕 (可变长的键)在散列表的许多应用中，处理的是任意长字符串的键。在这样的情况下，我们不能象在本节的程序中那样，把键简单地存入表中。在 MIX 计算机上，在散列表中处理可变长键的好方法应是怎样的？

▶66. 〔25〕 (奥勒·安布尔 (Ole Amble)) 在开式杂凑表中插入键时能否利用它们的数值或字母顺序，使得用算法 L 或算法 D 进行查找时，只要一遇到小于查找变元的键，便知查找已失败？

67. 〔$M25$〕 (姚期智) 证明在习题 62 意义下的所有固定排列的单杂凑满足不等式 $C'_N \geqslant \frac{1}{2}(1+1/(1-\alpha))$。〔提示：证明一个不成功的查找恰好花费 k 次探查的概率是 $p_k \leqslant (M-N)/M$。〕

为了保险，她做了对于专用名字的一次杂凑。

——格兰特·艾伦 (Grant Allen) "The Tents of Shem", ch.26(1889)

HASH, x，这个字没有定义——没有人知道杂凑是什么。

——安布罗斯·比尔斯 (Ambrose Bierce) (The Devil's Dictionary, 1906)

6.5 利用辅助键的检索

我们已经完成了对于"主键"查找，即唯一地确定文件中一个记录的键查找的研究。但有时也有必要不是按主键而是按记录中其它场的值进行查找；这些其它的场通常称为"辅助键"或记录的"属性"。例如，在一个包含某大学中学生信息的注册文件中，可能希望查找来自俄亥俄州不是专攻数学或统计学的二年级学生；或者查找所有未婚的说法语的研究生中的女生；等等。

一般地说，我们假定，每个记录包含有若干属性，而我们要查找其某些属性具有某些值的所有记录。指定所需记录的说明称作一个查询。查询通常限于下列三种类型：

a) 简单查询，它给出一个特定属性的一个特定的值；例如，MAJOR=MATHEMATICS；或者 RESIDENCE.STATE=OHIO。

b) 范围查询，它给出一个特定属性值的一个特定的范围；例如，COST<$18.00；或者21≤AGE≤23。

c) 布尔查询，它由与操作 AND，OR，NOT 相结合的上述类型所组成；例如

(CLASS=SOPHOMORE)AND(RESIDENCE·STATE=OHIO)
AND NOT((MAJOR=MATHEMATICS)OR(MAJOR=STATISTICS))

对于这三类查询，发现有效查找技术的问题，已经是十分困难的了。因此，通常就不再考虑更复杂类型的查询问题。例如，一个铁路公司可以有记载它的所有货车当前状态的

一个文件；但不允许直接提象“找出在距西雅图 500 英里内所有空冷藏车”这类查询．除非“同西雅图的距离”是保存在每个记录内的一个属性而不是由其它属性导出的一个复杂函数。而且使用除 *AND*, *OR* 和 *NOT* 之外的逻辑量词，将引进仅仅受提出查询者的想象所限制的进一步的复杂性；例如，给定垒球统计的一个文件，我们可以问及在晚场比赛中最长的连续的击中表演。这些例子是复杂的，但它们仍可以通过扫描一个适当排列的文件来处理；其它的查询甚至更困难，例如，求在五个或更多的属性上有相同值的所有记录对（而不必确定哪一些属性必须匹配）。这样的查询可以认为属于一般程序设计的任务而超出了本书讨论的范围，但通常它们可被分成属于这里所考虑的问题类型的一些小问题。

在我们开始研究辅助键检索的各种技术之前，重要的是考虑这个课题的经济效益。尽管大量的应用都能塞进上面概述的三类查询的一般框框里，但是这些应用中有许多并不真正适合于我们将要研究的复杂技术，而且它们中的某些用人工做比用机器做倒更好些。计算机已经把科学计算的速度提高了 10^7 或 10^8 倍，但这相对于信息处理问题来说，它们所提供的效率上的这一增益却还差得远。在涉及大量的数据时，今天的计算机仍然只能以机械的（而不是电子的）速度进行工作；所以，当计算机系统代替一个人工系统时，其性能价格比并没有急剧地增加。我们不能因为计算机执行某些其它任务那么好而对它抱过高的奢望。

人们攀登珠穆朗玛峰，“因为它是客观存在”，而且因为已经发展了使得攀登成为可能的工具；类似地，当面对着数据的高山时，人们试图在一个“联机实时”环境中，利用一台计算机找出对他们所能想象到的最为困难的查询的回答，而无须顾及代价。所期望的计算是可能的，但未必对每一个人的应用都合适。

例如，考虑下列对于辅助键检索的简单方法：在累积了一些查询之后，我们可以顺序地进行对整个文件的查找，检索出所有有关的记录（累积指的是在处理任何查询之前，先积累起一批这样的查询）。如果文件不太大并且这些查询不需要立即处理的话，这个方法是十分令人满意的。甚至对于磁带文件也可以使用，而且它仅仅在计算机有空闲时才使用它，所以就设备代价说来，它往往非常经济。而且它甚至能处理上面讨论的“到西雅图的距离”这样的计算查询。

简化辅助键检索的另一个简单方式，是提供给人们适当的打印好的索引信息，让他们自己做一部分工作。这通常是最合理和最经济的方式（当然，假定当打印新的索引时，旧的索引已被回炉）。

对某些应用来说，上述简单方案不能令人满意，这些应用都涉及非常大型的文件，对于这样的文件，查询的快速回答很重要。例如，如果不断地有好几个用户同时查询文件，或者如果这些查询是机器而不是人所生成的，就会出现这种情况。在本节中，我们的目标是要了解在有关文件结构的各种假定之下，在通常的计算机上，辅助键的效率有多高。

为了处理这一问题，已经提出了大量好的想法，但是（如同读者将能从所有这些预先警告式的注释中猜测到的那样）这些方法决不意味着它们都象主键检索算法那样好。由于文件及应用的多样性，我们不可能对已经考虑到的所有可能性给出完备的讨论，也不可能分析每个算法在典型环境中的特性。本节的剩下部分介绍已经提出的一些基本方法，至于在每一个具体情况下什么样的技术组合是最适当的，这个问题留给读者自己去想。

反文件 用于辅助键检索的第一类重要的技术是“反文件”的思想。这并不意味着文件被上下颠倒，它指的是记录和属性的作用被逆转。我们不是列出一个给定文件的各属性，而是列出有一个给定属性的各记录。

在日常生活中，经常地遇到反文件（在其它名称之下）。例如，对应于俄英字典的反文件是一部英俄字典。对应于本书的反文件是附于本书末尾的索引。财会人员传统地使用“双项簿记”，其中所有的交易同时记入现金账和顾客账，于是当前的现金状况和顾客的债务都很容易查阅。

一般地说，一个反文件并不是独立的，它要同原来的未反过来的文件一起使用。它提供了重复的冗余信息以便加速辅助键的检索。每个反文件由一些反表列组成，即其某个属性具有给定的值的所有记录的表。

正象别的表一样，在一台计算机内的反表可以用许多方式来表示，而且不同的表示方式适用于不同的场合。某些辅助键场仅仅有两个值（例如，“性别”属性），而对应的反表却十分长；但是其它场一般都有大量的值，而重复较少（例如，“电话号码”属性）。

例如，想象我们要在一本电话簿中存储信息，使得所有的项都可以根据名字、电话号或居住地址进行检索。一个简单的解决办法，是作成三个分开的文件，分别面向对于每种类型键的检索。另一个思想是组合这些文件，例如可以建立三个杂凑表，分别用作拉链方法的表头。在后一种方案中，文件的每个记录将是这三个表中的一个元素，因而它将包含三个链接场；这就是下面要进一步讨论的所谓多重表方法。第三种可能性是模拟图书馆卡片目录，在那里作者卡片、书名卡片、以及主题卡片分别按字母顺序排在一起，用类似的方式可把三个文件组合成一个超级文件。

考察本书索引所用的格式，即可得到关于反表表示的进一步的思想。在有些辅助键场中，每个属性值一般有五个左右的项，此时我们只需作一个简短的顺序表，其中首先列出键的值，后面跟以记录的位置（类似于一本书的索引中的页数）。如果有关的记录趋向于连续地堆集，则一个“范围说明”码（例如，页 200 到 214）可能是有用的。如果文件的记录经常要重新分配，则在反文件中使用主键来代替辅助键可能更好，使得当位置改变时也不必进行更改；例如，对于“圣经”段节的引用总是通过章和节给出，而对于某些书的索引都是基于节数而不是按页数给出的。

这些思想当中没有一个特别适合于象“性别”这样两个值的属性。在这种情况下，当然仅仅需要一个反表，因为非男性必然是女性而且反之亦然。如果每个值同大约一半的文件项有关，则反表将惊人地长，但是，在一台二进计算机上可以使用一种二进位串表示法，每个二进制位确定一个具体记录的值，即能相当好地解决这个问题。例如二进位串 0100101110…可能意味着，文件中的头一个记录指的是一个男子，第二个记录指的是一个女子，下两个记录指的都是男的，等等（参考 6.1 节末尾关于质数的讨论）。

上面的方法足以处理对特定属性值的简单查询。稍作推广即可以处理范围查询，不过必须用某个以比较为基础的查找方案（6.2 节）来代替杂凑。

对于象“(MAJOR=MATHEMATICS) AND (RESIDENCE.STATE=OHIO)”这样的布尔查询，我们需要涉及两个反文件。这可以用若干种方法来进行；例如，如果两个表都是有序的，则对每个表进行一次扫描将选出所有共同的项。或者，我们可以选出最

短的表并考查它的每个记录，校验其它的属性；但这个方法仅对 AND 有效，而对 OR 无效，而且它对于外部文件是没有吸引力的，因为它要对许多不满足查询要求的记录进行访问。

同样的考虑表明，上面所描述的多重表组织，对于在一个外部文件上的布尔查询是低效的，因为它涉及许多不必要的访问。例如，想象如果本书的索引被组织成一个多重表的形式，则会发生什么情况。这意味着，索引的每个项都将仅仅指出提到该主题的最后一页；然后在每一页上，对于该页的每个主题，又有对于该主题的上一次出现的进一步索引。为了找出有关"〔算法的分析〕和〔(外部排序)或(外部查找)〕"的所有页面，必须翻动许多页。若按另一种方法，即对反文件做简单操作，去找出满足查询的页的最小子集，即可仅仅查看实际出现该索引的两个页面，而解决同样的查询问题。

当一个反表被表示作一个二进位串时，简单查询的布尔组合当然是容易实现的，因为计算机可以用相当高的速度对二进位串进行操作。在某些混合查询中，某些属性被表示作由记录号码排成的顺序表，而其它属性表示作二进位串，此时不难把顺序的表转换成二进位串，然后对这些二进位串实施布尔操作。

这里，给出一个假设的应用的定量例子，可能是有帮助的。如同 5.4.9 节所述，假定我们有每个为 40 个字符的 1,000,000 个记录，而且文件存储在两个 MIXTEC 磁盘上。文件本身因此填满两个磁盘组，反表大概将填更多个磁盘组。每道包含 5000 个字符＝30,000 个二进位，所以一个具体属性的反表至多花费 34 道（这个极大道数当二进位串的表示为最短时出现）。假设有一个稍微复杂些的查询，它涉及 10 个反表的一个布尔组合；在最坏的情况下，我们将要从反文件读 340 道的信息，总共花费 340×25 毫秒＝8.5 秒。平均的等待延迟将大约是这个时间的一半，但通过小心的程序设计有可能消除这个等待。通过把每个二进位串表列的第一个道存入一个柱面中，而每个表列的第二个道存于下一个柱面当中等等，将消去大多数寻找时间，所以我们可以估计寻找时间至多大约是 34×26 毫秒≈0.9 秒（如果包含有两个磁盘组，则是这数的两倍）。最后，如果 k 个记录满足查询，则对于每个记录，它将花费大约 k×(60 毫秒（寻找）+12.5 毫秒（等待）+0.2 毫秒（读））的额外时间来取出它以便进行下一步处理。于是，为处理这个稍为复杂的查询，总计的预期时间的最优估计＜(10＋0.073 k)秒。这与在同样的假定下但不使用任何反表以最高速度读整个文件的 210 秒时间形成对照。

这个例子表明空间的最优化在磁盘存储中同时间最优化是紧密相关的；处理反表的时间大致就是寻找和读它们所需的时间。

上面的讨论或多或少假定，当我们查询文件时，它不增长或收缩；如果需要经常更改，则应该做什么呢？在许多应用中，只需累积一批更改的要求，当不需要回答查询时，在空闲的时间里处理它们即可。或者，如果更改的文件优先级较高，则 B 树（6.2.4 节）的方法是有吸引力的。反表的整个集合可被做成一个巨大的 B 树，同时对叶作特殊的约定使得分枝节点包含键的值，而诸叶既包含键又包含记录指针的表。

我们在上面的讨论中已经对另外一点作了注释，这就是关于范围查询的布尔组合的不同课题。例如，假设文件记录涉及北美诸城市，而且查询问及满足

$$(21.49^\circ \leqslant \text{纬度} \leqslant 37.41^\circ)\ \text{AND}\ (70.34^\circ \leqslant \text{经度} \leqslant 75.72^\circ)$$

的所有城市。对于这样的"正交范围查询"似乎没有真正好的数据结构（查阅地图可看出，许多城市满足这个纬度范围，而且许多城市满足经度范围，但是很难有任何城市处于这两个范围之中）。也许最好的方法是相当粗略地分划所有可能的纬度和经度的值，且每个属性仅仅有很小几类（例如，通过在下一个较小的5°的倍数处截断），然后对于每一（纬度，经度）类组合有一个反表，这就好象有对于每个局部区域各一页的地图。利用5°的区间，上边的查询将参考八个页，即（20°，70°），（25°，70°），…，（35°，75°）。现在需要对于这些页中的每一页处理范围查询，或者通过在这个页内进行一个更细的分划，或者通过直接查阅诸记录本身，采用哪种方法依赖于对应于该页的记录个数。在某种意义下，这是在每个内部节点处具有二维分枝的一个树结构。

J. L. 本特利（J. L. Bentley）和 R. A. 芬克尔（R. A. Finkel）已经提出处理正交范围查询的另一个方法，用的是所谓"四叉树"结构。这种树的每个节点表示地图的一个矩形区域，还包含来自该区域的记录之一；有四株子树，对应于与给定的键坐标有关的原来矩形的四个象限〔Acta Informatica 4（1974），1-10〕。最近，利用二分查找树，本特利已经对他的思想作了改进，在他的二分查找树上，当分枝时偶数级上的节点对照 x 坐标，而奇数级上的节点对照 y 坐标。结果，本特利的"二维二分查找树"的平均路径长度恰好和在6.2.2节中分析的一维查找树的平均路径长度完全一样。

对于正交范围查询的另一个树结构方法已经由布鲁斯·麦克纳特（Bruce McNutt）提出。例如，假设给定 x 的值，我们希望处理类似于"什么是最接近于点 x 的城市？"的查询。麦克纳特提出的二分查找树的每个节点对应于一个城市 y 和一个"测试半径" r；这个节点的左子树对应于和 y 之间的距离 $\leqslant r+\delta$ 的所有城市 z，而右子树对应于距离 $\geqslant r-\delta$ 的城市。这里 δ 是一个给定的容差；和 y 之间的距离在 $r-\delta$ 和 $r+\delta$ 之间的诸城市必须同时记入到两株子树中。这样一个"邮局树"中的查找，就有可能确定在一个给定点周围 δ 距离内的所有城市的位置（见图45）。

1972年，麦克纳特和爱德华·普林（Edward Pring）利用在随机次序下的美国大陆部分中231个人口最稠密的城市，作为一个数据库的例子，进行以这个思想为基础的若干实验。他们以有规律的方式，让测试半径收缩，当往左进行时以 $0.67r$ 代替 r，当向右进行时以 $0.57r$ 代替 r，仅当连续第二次取右分枝时 r 保持不变。结果是在 $\delta=20$ 英里的树中需要610个节点，而在 $\delta=35$ 英里的树中要求1600个节点。图45中示出这两株中较小的那一株的顶部诸级。（在这株树的剩下的级中，奥兰多（佛罗里达）出现在杰克逊维尔和迈阿密两个城市之下。某些城市十分频繁地出现；例如，马萨诸塞的布罗克顿有17个节点！）

复合属性 有可能把两个或多个属性组合成一个超级属性。例如，一个"（CLASS，MAJOR）属性"由一个大学的注册文件的 CLASS 场和 MAJOR 场组合而成。这样一来，用不相交的短的表的并代替较长的表的交，常常可满足查询的需要。

V. Y. 卢姆进一步发展了属性组合的思想〔CA（M 13（1970），660-665〕，他建议从左到右地按字典顺序排列组合属性的反表，且复写许多份，并以一种巧妙方式排列组合中的各属性。例如，假设我们有三个属性 A, B, C；则可以形成三个组合

$$(A, B, C),\ (B, C, A),\ (C, A, B) \tag{1}$$

而且对于其中的每一个，构造有序的反表（于是在第一个表中，诸记录以它们 A 值的顺序

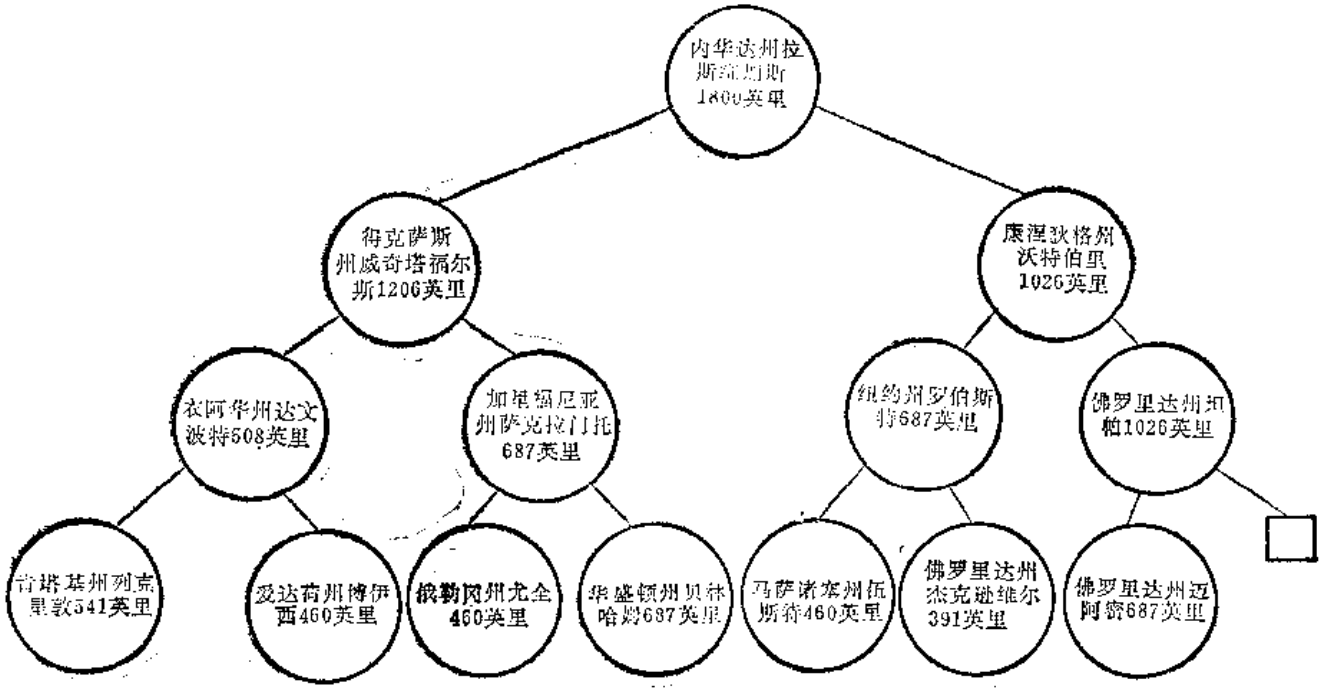


图45 "邮局树"例子的顶部诸级。为了查找一个给定点 x 附近的所有城市，由根点开始：如果 x 在拉斯维加斯1800英里之内则向左，否则就向右；然后重复这过程直到遇到一个终端节点。树构造的方法确保在这个查找期间将会找到所有距 x 20英里之内的城市。

出现，而所有具相同A值的记录以B值的顺序，然后再按C值的顺序出现)。这种组织有可能满足以三个属性的任何组合为基础的查询；例如，A和C有特定值的所有记录都将连续地出现于第三个表中。

类似地，由四个属性，我们可以形成六个组合属性

$$(A, B, C, D),\ (B, C, D, A),\ (B, D, A, C),\ (C, A, D, B),$$
$$(C, D, A B),\ (D, A, B, C) \qquad (2)$$

它们足以回答有关这些属性中一个、两个、三个或四个值的简单查询的所有组合。有一个一般的过程可从n个属性构造出$\binom{n}{k}$个组合属性，其中$k \leqslant \frac{1}{2} n$，这使得有$\leqslant k$或$\geqslant n-k$个属性值的特定组合的所有记录，都将在组合的一个属性表中连续地出现（见习题1）。当某些属性只有有限个值时，我们只需较少的组合就可以了。例如，如果D只是一个二值的属性，则通过把D放在（1）的前边可得到三个组合

$$(D, A, B, C),\ (D, B, C, A),\ (D, C, A, B) \qquad (3)$$

这几乎将和（2）一样地好，而且只有一半的冗余，因为不依赖于D的诸查询，可以只通过考察这些表之一的两个位置来进行处理。

二进属性 考虑所有属性都是二值的这一特殊情况是有教益的。在某种意义下，这是把属性组合起来的对立面，因为我们可以把任何值表示成为二进数，而且把这个数的个别的二进位认为是分开的属性。表1示出了涉及“是否”属性的一个典型文件；在这个文件中，诸记录代表某些甜饼的食谱，而诸属性确定使用哪些成分。例如，杏仁风味薄脆饼是用黄油、面粉、牛奶、坚果仁以及颗粒糖制造的。如果我们把表1想象作0和1的矩阵，则该矩阵的转置，就是在二进位串形式下的“反”文件。

表1的右边一列用来指出很少出现的特殊项目，比起为每个项目都设一列来，这样的方式可以更有效地进行编码；“玉米淀粉”这一列可以类似地处理。对偶地，我们可以发现对于“面粉”这一列进行编码的更有效的方式，因为面粉在除蛋白甜饼之外的每一行当中都出现。但现在我们把这些考虑放在一边，而且简单地忽略“特殊成分”一栏。

让我们定义二进属性文件中的一个基本查询为：对于在某些列中值为0和在其它列中值为1，以及在剩下的列中值为任意的所有记录的请求。利用“*”来代表一个任意的值，我们可以表示任何基本查询为诸0，1和*的序列。例如，考虑喜欢某种椰子饼干的人，但他却厌恶巧克力，讨厌茴香，以及不喜欢提纯香精；他可以描述查询如下

$$* 0 * * * * 0 * * 1 * * * * * * * * * * * * * * * * * 0 \qquad (4)$$

现在表1指出甜美条恰是这种东西。

在考虑为基本查询而组织文件的一般问题之前，先来考察没有指定0值，而仅仅指定1值和*值这种重要的特殊情况。这可以称为包含查询，因为它问及包括某个属性集合的所有记录，这里我们假定诸1表示出现的属性而0表示不出现的属性。例如，在表1的食谱中，既要求发酵粉又要求发酵苏打的，是浇糖姜汁饼和旧式家常糖饼。

在某些应用中，提供包含查询这种特殊功能就足够了。例如，在许多人工卡片文件系统的情况下就出现这种情形，诸如“带槽口边的卡片”或“特性卡片”。一个对应于表1的带槽口边卡片系统，对于每种花色将有一张卡片，而且对于每一成分都有穿出的孔（见图46）。为了处理一个包含查询，卡片文件排成一叠而穿针放置在每个列中对应于所需属

表1　具有二项属性的一个文件

	多香果	茴香籽	发酵粉	发酵苏打	黄油	小豆蔻	巧克力	桂皮	丁香	橙子	咖啡	玉米淀粉	枣	蛋白	蛋黄	面粉	姜汁	柠檬汁	柠檬皮	牛奶	糖浆	肉豆蔻	坚果仁	燕麦片	葡萄干	盐	红糖	颗粒糖	面糖	提纯香精	特殊成分
杏仁风味薄脆饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	0	1	0	0	—
苹果酱香味方饼	0	0	0	1	1	0	0	1	1	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	1	0	0	1	1	1	苹果酱
香蕉燕麦片家常小甜饼	0	0	0	1	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	0	1	0	1	香蕉
巧克力薄片小甜饼	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—
椰子杏仁饼	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—
奶油乳酪小甜饼	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1	奶油乳酪
美味梅脯条饼	0	0	0	0	1	0	0	0	0	1	0	1	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	桔子梅脯
双料巧克力糖饼	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	—
甜美条饼	0	0	1	0	1	0	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	1	—
带馅半圆形卷饼	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	0	1	1	0	0	0	1	0	0	1	0	1	0	1	—
芬斯卡·卡洛(一种食品名)	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	0	提纯杏仁
浇糖姜汁饼	0	0	1	1	1	0	0	1	1	0	0	0	0	1	1	1	1	0	0	1	1	0	0	0	0	0	1	0	1	0	醋
胡桃葡萄干甜饼	0	0	0	1	1	0	0	1	0	0	1	0	0	1	1	1	0	0	0	0	0	1	0	0	1	1	1	0	0	0	杏
宝石家常甜饼	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	葡萄肉冻
环形甜薄饼	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	色拉油
波纹套圈饼	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	1	0	0	0	0	1	0	0	—
圆腿肉面包	1	0	0	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	1	0	1	1	0	0	1	1	0	1	0	蜂蜜
蛋白甜饼	0	0	0	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	蜜饯樱桃
摩拉维亚香味家常甜饼	1	0	0	1	1	0	0	1	1	0	0	0	0	1	0	1	1	0	0	0	1	1	0	0	0	1	1	0	1	0	—
燕麦枣椰子条饼	0	0	0	1	1	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	1	1	0	1	1	1	0	0	—
川式家常糖饼	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	0	1	酸奶
花生奶油风车饼	0	0	0	1	1	0	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	0	0	1	花生黄油
裙状燕尾饼	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	1	—
佩弗纽斯	1	1	1	0	0	1	0	1	1	0	1	0	0	1	1	1	0	0	1	0	0	1	1	0	0	1	1	0	1	0	香橼，肉豆蔻干皮，胡椒
苏格兰燕麦脆饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	1	0	1	—
星状脆饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	—
春饼	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	1	0	1	1	0	—
斯普里兹家常甜饼	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	—
瑞典套圈饼	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	—
瑞士桂皮香脆片饼	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0	0	1	1	1	0	0	—
太妃条	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1	—
香精果仁冰镇家常饼	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	0	0	1	0	1	0	1	—

参考：*McCall's Cook Book*(New York, Random House), Chapter 9。

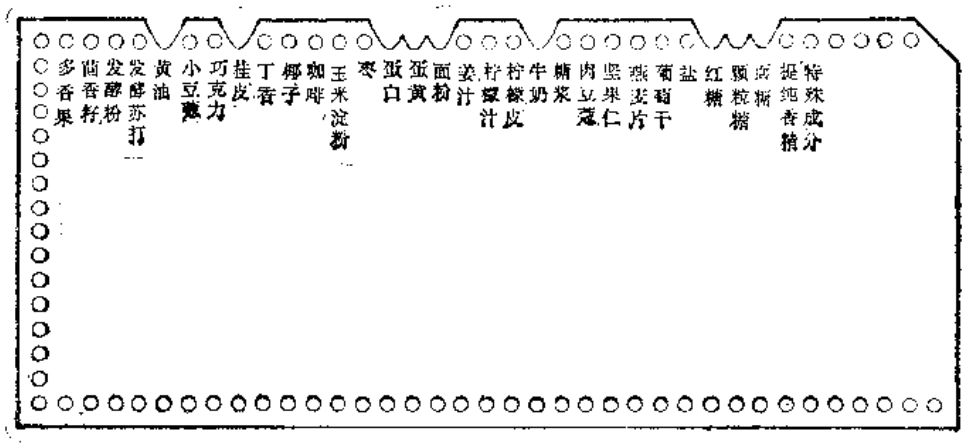


图46 一张带槽口边的卡片

性的位置上。在提起穿针之后，有适当属性的所有卡片都将落下。

一个特性卡片系统以类似的方式对反文件进行工作。在这种情况下，每个属性有一张卡片，对于具有该属性的每个记录，孔就穿在这张卡片的指定位置上。因此，一张通常的80列卡片可用来告知 $12\times80=960$ 个记录中，哪一个有给定的属性。为了处理一个包含查询，先选出特定属性的特性卡片并把它们放在一起；然后，光线穿过对应于所要的记录的所有位置。这个操作类似于上面所说明的求颠倒的二进位串之交来处理布尔查询的情况。

叠加的编码 我们对于这些人工卡片系统有特殊兴趣的原因在于已经想出了一些精巧的方案来节省带槽口边的卡片上的空间；同样的原理可应用于计算机文件的表示上。叠加的编码是一项类似于杂凑的技术，而且在发现杂凑之前若干年，实际上就发明了。其思想是把诸属性映射到一个 n 位场中的随机 k 位代码中，并且对于记录中所出现的每一个属性的编码进行叠加。对某个属性集合进行的包含查询可以被转换成对于对应的叠加的二进位码的包含查询。额外的少量记录就可满足这个查询。但是这种“假情报”的个数在统计上是可以控制的〔参考 Calvin N. Mooers, Amer. Chem. Soc. Meeting 112 (September, 1947), 14E-15E; American Documentation 2 (1951), 20-32〕。

作为叠加编码的一个例子，我们再次考虑表1，但是不考虑象发酵粉、蛋、面粉这样的基本成分，而仅仅考虑调味品。表2表明，如果我们对于每一个调味品的属性都指定10个二进位场中的2位随机代码，并叠加这些编码，则将发生什么情况。例如，巧克力薄片小甜饼的项目，是通过叠加巧克力、坚果仁和香精的代码而得到的：

$$0010001000\vee0000100100\vee0000001001=0010101101$$

这些代码的叠加，也产生了某些假的属性，在这种情况下即多香果，蜜饯樱桃，葡萄肉冻，花生黄油，以及胡椒；这些将引起在某些查询中出现假情报。（而且它们也提出了建立一个称为假情报家常饼的新品种！）

叠加的编码实际上在表2中不能非常有效地工作，因为该表是有着大量属性存在的一个小例子。事实上，苹果酱香味方饼对于每个查询将是一个假情报，因为它是通过叠加7个代码得到的，而结果是覆盖了全部十个位置；而佩弗纽斯饼甚至更坏，它通过叠加12个代码得到！另一方面，表2在某些方面却工作得惊人地好；例如，在查询“提纯香精”

表 2　叠加编码的一个例子

个别调味品的代码			
提纯杏仁	100001	枣	1000000100
多香果	0000100001	姜　汁	0000110000
荜茇籽	0000011000	蜂　蜜	0000000011
苹果酱	0010010000	柠檬汁	1000100000
杏	1000010000	柠檬皮	0011000000
香　蕉	0000100010	肉豆蔻干皮	0000010100
蜜饯樱桃	0000101000	糖　浆	1001000000
小豆蔻	1000000001	肉 豆 蔻	0000010010
巧克力	0010001000	坚 果 仁	0000100100
桂　皮	1000000010	桔　子	0100000100
香　橼	0100000010	花生黄油	0000000101
丁　香	0001100000	胡　椒	0010000100
樱　子	0001010000	梅　脯	0010000010
咖　啡	0001000100	葡萄干	0101000000
葡萄肉冻	0010000001	提纯香精	0000001001
叠加的代码			
杏仁风味薄脆饼	0000100100	圆筒肉面包	1011110111
苹果酱香味方饼	1111111111	蛋白甜饼	1000101100
香蕉燕麦片家常小甜饼	1000111111	摩拉维亚香味家常甜饼	1001110011
巧克力薄片小甜饼	0010101101	燕麦枣椰子条饼	1000100100
椰子杏仁饼	0001111101	旧式家常糖饼	0000011011
奶油乳酪小甜饼	0010001001	花生黄油风车饼	0010001101
美味梅脯条饼	0111110110	裙状燕尾饼	0000001001
双料巧克力情饼	0010101100	佩佛纽斯	1111111111
奶油条饼	0001111101	苏格兰燕麦脆饼	0000001001
带馅半圆形卷饼	1011101101	星状脆饼	0000000000
芬斯卡·卡客饼	0100100101	香饼	0011011000
浇糖姜汁饼	1001110010	斯普里兹甜饼	0000001001
胡桃葡萄干甜饼	1101010110	瑞典套圈饼	0000000000
宝石家常甜饼	0010101101	瑞士桂皮香脆饼	1000000010
环形甜薄饼	1000001011	太妃条饼	0010111101
波纹套圈饼	1011100101	香精果仁冰镇家常饼	0000101101

时，只有佩弗纽斯饼的记录作为假情报而出现。

如果我们有，比如说一个 32 位的场以及 $\binom{32}{3}=4960$ 种不同属性的组合，其中允许每个记录具有达六个属性，而且每个属性通过指定 32 个二进位中的 3 位来编码，则出现更适合的叠加编码的例子。在这种情况下，如果假定每个记录有六个随机选择的属性，则在一个包含查询中假情报的概率

$$
\begin{aligned}
&\text{在一个属性上是} && .07948358\\
&\text{在两个属性上是} && .00708659\\
&\text{在三个属性上是} && .00067094\\
&\text{在四个属性上是} && .00006786\\
&\text{在五个属性上是} && .00000728\\
&\text{在六个属性上是} && .00000082
\end{aligned}
\tag{5}
$$

于是，如果有M个不真正满足两属性查询的记录，则其中大约有.007M个叠加的代码将虚假地符合两个指定属性的所有代码位（这些概率在习题4中计算）。在反文件中所需的总的二进位仅仅是记录数的32倍，不到在原来的文件中确定属性所需要的二进位数的一半。

如果使用小心地选择的非随机代码，则如同W.H.考茨（W.H.Kautz）和R.C.卒格尔顿所证明的，有可能使用叠加代码而不出现任何假情报，见IEEE Transactions, IT-10（1964），363-377；关于它们的构造之一，见习题16。

马尔科姆·C.哈里逊（Malcolm C.Harrison）〔CACM 14（1971），777-779〕已经发现，叠加的编码可被用来加速文本查找。假定我们要在一个长的文本中找出一个特定字符串的所有出现，又不想象在算法6.3P那样构造很大的表；而且假定，这文本被分成为例如每个有50个字符的行$c_1c_2\cdots c_{50}$。哈里逊建议通过把49个对偶c_1c_2，c_2c_3，…，$c_{49}c_{50}$中的每一个，比方说，杂凑成为0到127之间的一个数，则行$c_1c_2\cdots c_{50}$的"标记"是128个二进位$b_0b_1\cdots b_{127}$的串，其中$b_i=1$当且仅当对某个j，$h(c_jc_{j+1})=i$。

如果我们现在要在一个称为HAYSTACK的大型文件中查找字$NEEDLE$的所有出现，则只需考察所有其标记在h(NE)，h(EE)，h(ED)，h(DL)和h(LE)的位置上包含二进位1的行。假定杂凑函数是随机的，一个随机行在它的标记中包含所有这些二进位的概率仅仅是0.00341（参考习题4）；因此，五个反表的二进位串的交将迅速查出包含NEEDLE的所有行，连同少量的假情报在一起。

随机性的假定在这个应用中并不非常合理，因为典型的文件有如此多的冗余性；二进的图，即在字中的双字母组合，其分布是有强烈倾向性的。例如，拼齐包含有一个空白字符的所有对偶c_jc_{j+1}大概将是非常有帮助的，因为空白通常比任何其它符号更为常见得多。

伯顿·H·布卢姆（Burton H.Bloom）已经提出了叠加编码对于查找问题的另一个有趣的应用〔CACM 13（1970），422-426〕；他的方法实际上适用于主键的检索，然而在这节中讨论它却是最适当的。想象一个大型数据库的查找，而且在这个大型数据库中，如果查找是不成功的，就不需要进行计算。例如，我们有可能需要校验某人的信贷定额或通行证号等等，如果他的记录在文件中没有出现，则无需作进一步的研究。类似地，在用计算机排字时，我们可以有一个简单的算法，它正确地连接大多数的字，但对大约50,000个例外的字它是失败的；如果我们没有发现例外文件中的字，则就可以放心地使用这个简单的算法。

在这种情况下，有可能在内存中保持一个二进位表，使得在文件中不出现的大多数键可以被认出是不存在的，而无须进行对外部存储的任何访问。原因是：设内部二进位表是$b_0b_1\cdots b_{m-1}$，其中M是相当大的，对于文件中的每个键K_j，计算k个独立的杂凑函数$h_1(K_j)$，…$h_k(K_j)$，并且置相应的k个b成为1（这k个值不必是不同的）。于是当且仅当对于某一个j和l，$h_l(K_j)=i$时$b_i=1$。现在来确定一个查找变元K是否在外部文件中，首先测试对于$1\leqslant l\leqslant k$，是否有$b_{h_l}(K)=1$；如果不是，则不需要访问外存，但如果是，则若k和M选择适当，一个通常的查找大概将找出K。当在文件中有N个记录时，一次假情报的机会近似地为$(1-e^{-kN/M})^k$。在某种意义下，布卢姆的方法把整个文件处理成为一个记录，以主键作为属性，而且在一个庞大的M-进场中存放叠加编码。

理查德·A.古斯塔夫森（Richard A.Gustafson）已经发现了叠加编码的另一种变

形〔Ph. D. thesis (Univ. South Carolina, 1969)〕。假设我们有N个记录，而且每个记录具有从 10,000 种可能的属性的集合中选出的六个属性。例如，这个记录可以代表技术论文，而属性可以是描述文章的关键词。设h是一个杂凑函数，它把每个属性映射到 0 和 15之间的一个数。如果一个记录有属性a_1，a_2，…，a_6，则古斯塔夫森建议把这个记录映射成为 16 个二进位的数$b_0b_1\cdots b_{15}$，其中$b_i=1$当且仅当对某个j，$h(a_j)=i$；而且，如果这个方法仅导致诸b中的k个等于1，这里$k<6$，则另外的$6-k$个1由某种随机方法提供之（不必依赖于记录本身）。在$\binom{16}{9}=8008$种十六个二进位的代码中恰有六个1出现，而且碰巧大约有$N/8008$个记录将被映射到每个值上。我们可以保持诸记录的8008个表，利用一个适当的公式直接计算对应于$b_0b_1\cdots b_{15}$的地址。事实上，如果在位置$0\leqslant p_1<p_2<\cdots<p_6$中出现1，则函数

$$\binom{p_1}{1}+\binom{p_2}{2}+\cdots+\binom{p_6}{6}$$

将把每个串$b_0b_1\cdots b_{15}$转换成数 0 和 8007 之间的唯一的数，我们在习题 1.2.6-56、2.2.6-7 中已经看到了这一点。

现在，如果要找出具有三个具体属性A_1，A_2，A_3的所有记录，则我们计算$h(A_1)$，$h(A_2)$，$h(A_3)$；假设这三个值是不同的，我们仅仅需要考察存储在$\binom{13}{3}=286$个表中，且其二进制代码$b_0b_1\cdots b_{15}$在这三个位置中包含1的诸记录。换言之，在查找中仅仅需要考察$286/8008\approx3.5\%$的记录。

组合杂凑 莫定刚刚描述的古斯塔夫森方法的思想是找出某种方法把记录映射到内存单元，使得仅有较少的单元同一个具体查询相关联。但是当各记录只有较少的属性时，他的方法仅可应用于包含查询。另一类映射设计来处理象（4）那样由诸0，1和*组成的任意基本查询，它是由罗纳德·L. 里夫斯特于 1971 年发现的〔见 *SIAM J. Computing* 5 (1976)，19-50〕。

首先，假定我们要编一部“交叉字谜字典”，用于所有具有六个字母的英文字；一个典型的查询可以是，比如说，查阅所有具有形式 N**D*E 的字，而得到的回答是{NEEDLE, NIDDLE, NODDLE, NOODLE, NUDDLE}。这个问题可巧妙地解决如下：建立2^{12}个表，把字 NEEDLE 翻成表号

$$h\ (\mathrm{N})\ h\ (\mathrm{E})\ h\ (\mathrm{E})\ h\ (\mathrm{D})\ h\ (\mathrm{L})\ h\ (\mathrm{E})$$

这里h是一个杂凑函数，它把每个字母变成一个二进位值。把 6 个二进位对放在一起，即得一12 位的表地址，因此，为了回答查询N**D*E，仅需查看 4096 张表中的 64 张表。

类似地假设我们有 1,000,000 个记录，每个记录含 10 个辅助键，其中每个辅助键有相当大量的可能的值。我们可以把辅助键为（K_1，K_2，…，K_{10}）的诸记录映射到 20 位的数

$$h\ (K_1)\ h\ (K_2)\cdots h\ (K_{10}) \tag{6}$$

上，其中h是把每个辅助键映成 2 位值的杂凑函数，（6）表示这 10 个二进位对偶的并列。这个方案把 1,000,000 个记录映射成为$2^{20}=1,048,576$个可能的值，而且可以把总共的映射当作具有$M=2^{20}$的一个杂凑函数；拉链可用来解决冲突。如果我们要检索其任何五个

辅助键具有特定值的所有记录，则仅需要考察 2^{10} 个表（对应于（6）中的五个未确定的二进位对偶），所以平均仅仅需要考察大约 $1000=\sqrt{N}$ 个记录。(一个类似的方法已由有泽诚提出，见 *J. Inf. Proc. Soc. Japan* 12 (1971)，163-167，B. 德易尔(B. Dwyer) 也提出了（未发表）。德易尔提出使用比（6）更灵活的映象，就是

$$(h_1(K_1)+h_2(K_2)+\cdots+h_{10}(K_{10}))\bmod M$$

其中M是任何合适的数，而 h_i 可能是 w_iK_i 形式的任意杂凑函数，其中 w_i 是"随机"的。

里夫斯特已经进一步发展了这个思想，使得在许多情况下，我们有下列的状况。假定有$N\approx 2^n$个记录，每个有m个辅助键。每个记录以这样一种方式映射到一个 n 位的杂凑地址上，即任何不指定 k 个键的值的查询近似地对应于 $N^{k/m}$ 个杂凑地址。在这节中（除古斯塔夫森的方法外），我们已经讨论过的所有其它的方法，都需要用阶为N的步骤进行检索，尽管比例常数是小的；对于足够大的N，里夫斯特的方法将是更快的，而且它不需要反文件。

但是在我们能应用这项技术之前，要定义一个适当的映象。这里是具有小参数的一个例子，当 $m=4$ 和 $n=3$ 以及所有辅助键的值都是二进数时；我们可以把 4 位记录映射成为八个地址如下：

$$\begin{array}{ll} *\,0\,0\,0\to 1 & 0\,1\,*\,0\to 5 \\ *\,1\,1\,1\to 2 & 1\,0\,*\,1\to 6 \\ 0\,*\,0\,1\to 3 & 0\,0\,1\,*\to 7 \\ 1\,*\,1\,0\to 4 & 1\,1\,0\,*\to 8 \end{array} \tag{7}$$

对这个表的考查揭示，对应于查询 0 * * * 的所有记录，都映射到单元 1，2，3，5 和 7；而且类似地，任何具有三个 * 的基本的查询恰恰对应到五个单元；具有两个 * 的基本查询每个对应于三个单元；而具有一个 * 的基本查询对应于一个或两个单元，平均为（8×1+24×2)/32=1.75，于是有

查询中未确定的二进位数	要查找的地址个数
4	$8=8^{4/4}$
3	$5\approx 8^{3/4}$
2	$3\approx 8^{2/4}$
1	$1.75\approx 8^{1/4}$
0	$1\approx 8^{0/4}$

(8)

当然，这是一个小例子，通过硬算更容易处理它。但它导致非显然的应用，因为当 $m=4r$ 和 $n=3r$ 时我们也可以应用它：通过把键分成每个 4 位的 r 个组，而且对每组应用（7），把 $4r$ 位的记录映射成为 $2^{3r}\approx N$ 个地址。得到的映射有所希望的性质：在m个二进位中对 k 个不加指定的一个查询，将近似地对应到 $N^{k/m}$ 个单元（见习题 6）。

由里夫斯特设计的某些进一步的映射出现于习题 7 中；它们可用来同（7）组合起来以产生在 $1\leqslant m/n\leqslant 2$ 的情况下的杂凑函数。实际上，可使用大小为 b 的桶而且我们将取$N\approx 2^nb$；为剖析简便计，在上述讨论中使 $b=1$。

里夫斯特还提出了用于处理基本查询的另一个简单的技术。假设我们有，比如说，每个有 30 个二进位的$N\approx 2^{10}$ 个记录，这里我们希望回答象（4）那样任意的 30 个二进位

的基本查询。于是可以简单地把这 30 个二进位分成三个 10 个二进位的场，并保持三个大小为$M=2^{10}$的杂凑表。每个记录分存于对应于它的二进位配置的各表列的三个场中。在适当条件下，每个表列将包含大约一个元素。给定一个带有 k 个未指定二进位的基本查询，则至少这些场之一将有 $\lfloor k/3 \rfloor$或更少的二进位是未指定的；因此我们至多需要考察 $2^{\lfloor k/3 \rfloor} \approx N^{k/30}$ 个表列，以找出对于这一查询的所有回答。或者，我们可以在已选择的场中使用用于处理基本查询的任何其它技术。

广义的检索结构 里夫斯特也还提出以类似 6.3 节中的检索结构的数据结构为基础的另一个方法。我们可以用一个广义的二分检索结构的每个内部节点指示它表示记录的哪个二进位。例如在表 1 的数据中，可以让检索结构的根表示香草精；于是左子检索结构将对应于不含香草精的 16 个食物品种，而右子检索结构将表示使用香草精的 15 个食物品种。这 16-15 的划分正好把文件分成两半；我们以类似的方式处理每个子文件。当一个子文件变成适当小时，即以一个终端节点表示它。

为处理一个基本的查询，我们从检索结构的根开始。一个广义检索结构的根指明一个属性，根据其值为 0 或 1 而分别地查找左或右子检索结构；如果在该二进位置上查到 *，则两个子检索结构都查找。里夫斯特已经证明，如果指定 k/m 个属性，则平均查找时间仍按 $N^{k/m}$ 增长。

***平衡的文件方案** 对于信息检索的另一个组合方法，是以“平衡的不完备的区组设计”为基础的，这是进行过相当大量研究的课题。尽管这一课题从数学的观点看来是非常有趣的，但不幸的是，与上面描述的其它方法比较，还不能证明它是非常有用的。这里将给出简短的理论介绍，以便指出结果的特色，希望某些读者可以想出一个好方法，使这些思想变成为实用的。

斯坦纳（Steiner）三元系统是把 v 个对象排列成无次序的三元组，方式是每一对对象恰在一个三元组中出现。例如，当 $v=7$ 时，基本上仅有一个斯坦纳三元系统，即

三元组	包括的数对
{1, 2, 4}	{1, 2}, {1, 4}, {2, 4}
{2, 3, 5}	{2, 3}, {2, 5}, {3, 5}
{3, 4, 6}	{3, 4}, {3, 6}, {4, 6}
{4, 5, 0}	{0, 4}, {0, 5}, {4, 5}
{5, 6, 1}	{1, 5}, {1, 6}, {5, 6}
{6, 0, 2}	{0, 2}, {0, 6}, {2, 6}
{0, 1, 3}	{0, 1}, {0, 3}, {1, 3}

(9)

由于有 $\frac{1}{2}v(v-1)$对对象，且每个三元组有三个对，总共必然有$\frac{1}{6}v(v-1)$个三元组，而且由于每个对象必须同 $v-1$ 个其它的对象配对，每一对象必然恰恰出现于 $\frac{1}{2}v(v-1)$个三元组中。这些条件意味着除非$\frac{1}{6}v(v-1)$和$\frac{1}{2}v(v-1)$是整数，否则斯坦纳三元系统就不能存在，这等价于说 v 是奇数而且不同余于 2modulo3，即

$$v \bmod 6 = 2 \text{ 或 } 3 \tag{10}$$

反之，T. P. 柯克曼（T. P. Kirkman）于 1847 年证明，斯坦纳三元系统对所有使得(10)

成立的 $v \geqslant 1$ 存在。他的有趣的构造在习题 10 中给出。

斯坦纳的三元组系统可用来减少组合的反文件索引的冗余性。例如，再次考虑表 1 家常饼食谱的文件，并把最右边的一列转换成第 31 个属性，当需要任何特殊成分时为 1 否则为 0。假定我们要回答关于属性对偶的所有包含查询，例如，“什么花色既使用榔子又使用葡萄干？”可以对这$\binom{31}{2}=465$种可能的查询之每一种，都建立一个反表。但结果将花费大量的空间。因为（比如说）佩弗纽斯饼将在$\binom{17}{2}=136$个表中出现，而且具有所有 31 种属性的一个记录将在每个表中出现！一个斯坦纳三元组系统在这种情况下即可用来作稍许改进。有一个 31 个对象的斯坦纳三元组系统，具有 155 个三元组，而且每个对象对偶恰恰出现于这些三元组之一中。我们可以为每个三元组 $\{a, b, c\}$ 配备四个表，在这四个表中，一个是对于有属性 a，b，$\bar{c}$（即非 c）的所有记录的，另一个表是对于有属性 a，$\bar{b}$，c 的，第三个是对于 $\bar{a}$，b，c，第四个是对于有所有三个属性 a，b，c 的。这就保证任何记录都不会被包括在 155 个以上的反表中，而且每当一个记录有对应于系统的某个三元组的三个属性时，它就节省了空间。

三元组系统是区组对象数在三个以上的区组设计的特殊情况。例如，有一个方法排列 31个对象成为六元组，使得每对对象恰恰出现于一个六元组中。

$$
\begin{array}{lllll}
\{1, 5, 17, 22, 23, 25\} & \{7, 11, 23, 28, 29, 0\} & \{13, 17, 29, 3, 4, 6\} & \{20, 24, 5, 10, 11, 13\} & \{26, 30, 11, 16, 17, 19\} \\
\{2, 6, 18, 23, 24, 26\} & \{8, 12, 24, 29, 30, 1\} & \{14, 18, 30, 4, 5, 7\} & \{21, 25, 6, 11, 12, 14\} & \{27, 0, 12, 17, 18, 20\} \\
\{3, 7, 19, 24, 25, 27\} & \{9, 13, 25, 30, 0, 2\} & \{15, 19, 0, 5, 6, 8\} & \{22, 26, 7, 12, 13, 15\} & \{28, 1, 13, 18, 19, 21\} \\
\{4, 8, 20, 25, 26, 28\} & \{10, 14, 26, 0, 1, 3\} & \{16, 20, 1, 6, 7, 9\} & \{23, 27, 8, 13, 14, 16\} & \{29, 2, 14, 19, 20, 22\} \\
\{5, 9, 21, 26, 27, 29\} & \{11, 15, 27, 1, 2, 4\} & \{17, 21, 2, 7, 8, 10\} & \{24, 28, 9, 14, 15, 17\} & \{30, 3, 15, 20, 21, 23\} \\
\{6, 10, 22, 27, 28, 30\} & \{12, 16, 28, 2, 3, 5\} & \{18, 22, 3, 8, 9, 11\} & \{25, 29, 10, 15, 16, 18\} & \{0, 4, 16, 21, 22, 24\} \\
 & & \{19, 23, 4, 9, 10, 12\} & &
\end{array}
\tag{11}
$$

（这个设计通过 mod31 加法由第一个区组形成。为了验证它有所述的性质，注意对于 $i \neq j$，30 个值 $(a_i - a_j)\bmod 31$ 是不同的。其中 $(a_1, a_2, \cdots, a_6) = 1, 5, 17, 22, 23, 25$）。为了求出包含给定对 (x, y) 的六元组，选择 i 和 j 使得 $a_i - a_j \equiv x - y$ (modulo31)；现在，如果 $k = (x - a_1)\bmod 31$，则我们有 $(a_i + k)\bmod 31 = x$ 和 $(a_j + k)\bmod 31 = y$）。

我们可以利用上述设计存储反表，使得没有任何一个记录出现 31 次以上。相应于具有 a，b，c，d，e，f 诸属性中两个或两个以上属性的记录的各种可能性，每个六元组 $\{a, b, c, d, e, f\}$ 同 57 个表相关联，即 $(a, b, \bar{c}, \bar{d}, \bar{e}, \bar{f})$，$(a, \bar{b}, c, \bar{d}, \bar{e}, \bar{f})$，…，$(a, b, c, d, e, f)$，而且对于每个 2-属性包含查询的答案是适当的六元组的 16 个适当的表的析取。对于这个设计，佩弗纽斯饼将存于 31 个区组的 29 个中，因为如果我们把列编号成 0 到 30，则该记录在除了 $\{19, 23, 4, 9, 10, 12\}$和$\{13, 17, 29, 3, 4, 6\}$两个区组之外的所有区组中，有六个属性中的两个。

当我们希望处理基本查询而不是包含查询时，一个类似的思想可用来减少复合反文件的冗余性，例如，假设我们有具有五个辅助键 K_1，K_2，K_3，K_4，K_5 的记录，每一个键有四个可能的值$\{0, 1, 2, 3\}$。给定 a 和 b 且 $i \neq j$，为了回答关于 $K_i = a$ 和$K_j = b$ 的记录的查询，我们可以对所有 160 种这样的查询建立反表；每个记录将出现在这些反表的 10 个当中。一种选择是利用下列在所谓“人为正交拉丁方”的组合型式基础上作成的有序五元组的配置：

$$
\begin{array}{llll}
(0,\ 0,\ 0,\ 0,\ 0) & (1,\ 0,\ 1,\ 2,\ 3) & (2,\ 0,\ 2,\ 3,\ 1) & (3,\ 0,\ 3,\ 1,\ 2) \\
(0,\ 1,\ 1,\ 1,\ 1) & (1,\ 1,\ 0,\ 3,\ 2) & (2,\ 1,\ 3,\ 2,\ 0) & (3,\ 1,\ 2,\ 0,\ 3) \\
(0,\ 2,\ 2,\ 2,\ 2) & (1,\ 2,\ 3,\ 0,\ 1) & (2,\ 2,\ 0,\ 1,\ 3) & (3,\ 2,\ 1,\ 3,\ 0) \\
(0,\ 3,\ 3,\ 3,\ 3) & (1,\ 3,\ 2,\ 1,\ 0) & (2,\ 3,\ 1,\ 0,\ 2) & (3,\ 3,\ 0,\ 2,\ 1)
\end{array}
\tag{12}
$$

这里如果我们考察五个分量的任意两个分量，则将看到，所有 16 种可能的有序对值在这些分量中恰出现一次。我们可以把这个配置的区组（a，b，c，d，e）同至少满足条件 $K_1=a$，$K_2=b$，…，$K_6=e$ 中的两个的记录联系起来。这样，16 个区组中的每一个将同 1024 个可能的记录中的 376 个相关联，所以平均冗余性从 10 降低到 $16\times376/1024=5\frac{7}{8}$。

小马歇尔·霍尔的书 *"Combinatorial Theory"* (Waltham, Mass: Blaisdell, 1967) 详细地论述了区组设计·拉丁方等等的理论。尽管这些组合配置是非常漂亮的，但是说到对于信息检索的应用，至今仅有的只是减少当使用复合的反表时引起的冗余性，而且戴维·K.乔（David. K. Chow）〔*Information and Control* 15 (1969), 377-396〕已经发现，甚至不使用组合设计也可以得到同样的效果。

简史和参考文献目录 关于辅助键检索技术的头一篇公开的论文，是由 L. R. 约翰逊发表的，见 *CACM* 4 (1961), 218-222。大约在同一时候，诺亚·S.普里韦斯（Noah S. Prywes）、H. J. 格雷（H. J. Gray）、W. I. 兰多尔（W. I. Landauer）、D. 莱夫科维茨（D. Lefkowitz）和 S. 利特文（S. Litwin）独立地发展了多重表系统，参考 *IEEE Trans. on Communication and Electronics* 68(1963), 488-492。另一篇相当早的并对后来的工作有影响的著作是 D. R. 戴维斯（D. R. Davis）和 A. D. 林（A. D. Lin）的，*CACM*, 8 (1965), 243-246。

自从那以后，出现了关于这个课题的大量文献，但大多数都涉及用户接口和程序设计语言的考虑，这些已不再属于本书的范围。除了已经引证的论文外，下列已发表的论文，在作者写这节时发现是最有用的：杰克·明克尔（Jack Minker）及杰罗姆·萨伯尔（Jerome Sable），*Ann. Rev. of Information Science and Technology* 2 (1967), 123-160；罗伯特·E.布莱尔（Robert E Bleier），Proc. *ACM Nat. Conf.* 22 (1967) 41-49；杰罗姆·A.菲尔德曼（Jerome A. Feldman）和保罗·D.罗夫纳（Paul. D. Rovner）*CACM* 12 (1969), 439-449；伯顿·H.布卢姆，Proc. *ACM Nat. Conf.* 24 (1969) 83-9；H. S.赫伯斯和 L. H. 蒂尔（L. H. Thiel），*Information Storage and Retrieval* 6 (1970) 137-153；文森特·Y.卢姆和回龄（Huei Ling），*Proc. ACM Nat. Conf.* 26 (1971), 349-356；C. P. 伯恩在 *"Methods of Information Handling"* (New York: Wiley, 1963), Chapter 5 中，给出了关于人工卡片文件系统的一个很好的综述。"平衡的文件方案"最初是由 C. T. 阿伯拉罕（C. T. Abraham），S. P. 戈斯（S. P. Ghosh）以及 D. K. 拉伊-查奥杜里于 1965 年发展起来的。也许关于这一工作及后来扩展的最好的综述出现在拉 R. C. 博斯和加里·G. 科克（Gary. G. Koch）的论文中，*SIAM J. Appl. Math* 17 (1969) 1023-1214。

在这一节里我们已经讨论了大量的在十分不同的情况下有益的有趣思想。由于这许多

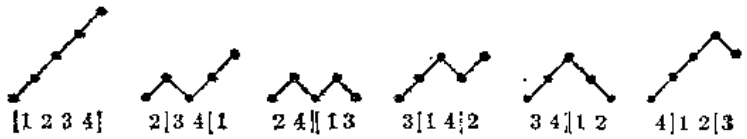
技术是在写这一节之前不久发明的，大概不久就会有进一步的发展。

说明这一点是有趣的：就是人脑在辅助键检索方面比计算机更好些；事实上，人们较易从片断的信息来识别面貌或表情等等，而计算机基本上还完全不能做这些工作。因此，有可能在某一天发现一种崭新的机器设计方法，一举解决辅助键的检索问题，从而使得整个这一节的内容过时。

习题

▶1.〔*M*27〕 设 $0 \leqslant k \leqslant \frac{1}{2}n$。证明下列构造产生$\binom{n}{k}$个 $\{1, 2, \cdots, n\}$ 的排列，使得对于 $t \leqslant k$ 或 $t \geqslant n-k$，$\{1, 2, \cdots, n\}$的每个 t 元子集至少作为一个排列的前 t 个元素出现：考虑平面中从（0，0）到（n，r）的一条通路，这里 $r \geqslant n-2k$，在这条通路中第 i 步是从（$i-1$，j）到（i，$j+1$）或者到（i，$j-1$）；仅当 $j \geqslant 1$ 时才允许后一种可能性，因此这条通路决不会跑到X轴下边。恰有$\binom{n}{k}$个这样的通路。对于每条这种类型的通路，利用开始时为空的三个表列构造一个排列如下：对于 $i=1, 2, \cdots, n$，如果这条通路的第 i 步往上去，就把号 i 放到表B中；如果这步往下，就把 i 放到表A中，而且把当前表B中最大的元素放到表C中。得到的排列是表A，然后表B，然后表C的最后内容，每个表处于递增的次序。

例如，当 $n=4$ 和 $k=2$ 时，由这个过程确定的六条通路和排列为



（垂直线示出表A、B和C之间的分界线。这六个排列对应于（2）中的复合属性）。

提示：通过从（0，0）到（n，$n-2t$）的一条通路表示每 t 个元素的子集S。如果 $i \notin S$，它的第 i 个步骤从（$i-1$，j）进行到（i，$j+1$），如果 $i \in S$，则进行到（i，$j-1$）。把每条这样的通路转换成有上述特殊形式的一条适当通路。

2.〔*M*25〕（萨克蒂·P.戈斯）试求用于访问记录的一个表 $r_1r_2\cdots r_l$ 的最小可能长度 l，使得对三个二进值的辅助键进行的任意包含查询 * * 1，* 1 *，1 * *，* 1 1，1 * 1，1 1 *，1 1 1 的所有回答的集合，都将出现于连续的单元 $r_j\cdots r_j$ 中。

3.〔19〕 在表2中，什么包含查询将引起（a）旧式家常糖饼（b）燕麦枣椰条饼在假情报中出现？

4.〔*M*30〕 在一个 n 位的场中共有$\binom{n}{k}$个 k 位代码，设每个记录都从中随机选择 r 个不同的属性，且查询包括 q 个不同的但除此以外是随机的属性，试求这种情况下（5）中概率的精确公式（如果这些公式没有简化，也不要吃惊）。

5.〔40〕 当对子串的查找使用哈里逊技术时，实验各种方法来避免正文的冗余性。

▶6.〔*M*20〕 指明 k 个二进位的 m 位基本查询的总数是 $s=\binom{m}{k}2^k$。如果象在（7）中那样的一个组合杂凑函数把这些查询分别地转化成为$l_1l_2, \cdots, l_s$ 个位置，则$L(k)=(l_1+l_2+\cdots+l_s)/s$ 是每个查询的平均位置数〔例如，在（71）中我们有$L(3)=1.75$〕。

一个杂凑函数映射前 m_1 位，另一个杂凑函数映射剩下的 m_2 位，它们形成（m_1+m_2）

位场上的一个复合的杂凑函数，其中$L_1(k)$和$L_2(k)$是每个查询对应的平均位置数，试借助于L_1和L_2，求对于复合函数，表达$L(k)$的一个公式。

7.〔M24〕（R. L. 里夫斯特）如同上题中定义的那样，对于下列复合的杂凑函数，求函数$L(k)$：

(a) $m=3$, $n=2$	(b) $m=4$, $n=2$
0 0 * → 1	0 0 * * → 1
1 * 0 → 2	* 1 * 0 → 2
* 1 1 → 3	* 1 1 1 → 3
1 0 1 → 4	1 0 1 * → 3
0 1 0 → 4	* 1 0 1 → 4
	1 0 0 * → 4

8.〔M47〕进一步发展象（7）那样有用的复合杂凑函数类。

9.〔M20〕证明当$v=3^n$时，所有形如

$$\{(a_1\cdots a_{k-1}\,0\,b_1\cdots b_{n-k})_3,\ (a_1\cdots a_{k-1}\,1\,c_1\cdots c_{n-k})_3,\ (a_1\cdots a_{k-1}2d_1\cdots d_{n-k})_3\},\quad 1\leqslant k\leqslant n$$

的三元组的集合形成一个斯坦纳三元组系统，其中诸a诸b诸c和诸d取遍0，1和2的所有使得$b_i+c_i+d_i\equiv 0$ (modulo3)的组合。

10.〔M32〕（托马斯·P. 柯克曼，*Cambridge and Dublin Math. Journal* 2 (1847), 191-204）我们说阶为v的一个柯克曼三元组系统是把$v+1$个对象$\{x_0, x_1, \cdots, x_v\}$变成为三元组的一个安排，使得对于$i\neq j$，每个对偶$\{x_i, x_j\}$恰出现于一个三元组中，但是对于$0\leqslant i<v$，$v$个对偶$\{x_i, x_{(i+1)\bmod v}\}$决不出现于同一个三元组中。例如

$$\{x_0, x_2, x_4\},\quad \{x_1, x_3, x_4\}$$

是阶为4的一个柯克曼三元组系统。

a）证明仅当$v \bmod 6=0$或4时一个柯克曼三元组系统才能存在。

b）给定v个对象$\{x_1, \cdots, x_v\}$上的一个斯坦纳系统S，证明下列构造产生$2v+1$个对象上的另一个斯坦纳系统S'以及阶为$2v-2$的一个柯克曼三元组系统K'：S'的三元组是S的那些三元组加上

i）$\{x_i, y_i, y_k\}$，其中$j+k\equiv i$ (modulo v)及$j<k$，$1\leqslant i$，j，$k\leqslant v$；

ii）$\{x_i, y_j, z\}$，其中$2j\equiv i$ (modulo v)，$1\leqslant i$，$j\leqslant v$。

K'的三元组是S'的那些三元组减去含y_1和/或y_v的那些。

c）给定$\{x_0, x_1, \cdots, x_v\}$上的一个柯克曼三元组系统，其中$v=2u$，证明下列构造产生$2v+1$个对象的一个斯坦纳三元组系统$S'$，以及阶为$2v-2$的一个柯克曼三元组系统$K'$：$S'$的三元组是$K$的三元组加上

i）$\{x_i, x_{(i+1)\bmod v}, y_{i+1}\}$，$0\leqslant i<v$；

ii）$\{x_i, y_j, y_k\}$，$j+k\equiv 2i+1$ (modulo $v-1$)，$1\leqslant j<k-1\leqslant v-2$，$1\leqslant i\leqslant v-2$；

iii）$\{x_i, y_j, y_v\}$，$2j\equiv 2i+1$ (modulo $v-1$)，$1\leqslant j\leqslant v-1$，$1\leqslant i\leqslant v-2$；

iv）$\{x_0, y_{2j}, y_{2j+1}\}$，$\{x_{v-1}, y_{2j-1}, y_{2j}\}$，$\{x_v, y_j, y_{v-j}\}$，对于$1\leqslant j<u$；

v）$\{x_v, y_u, y_v\}$。

K'的三元组是S'的三元组减去包含y_1和/或y_{v-1}的那些。

d）利用上面的结果证明阶为 v 的柯克曼三元组系统对于形如$6k$或 $6k+4$ 的所有 $v \geqslant 0$ 存在，且 v 个对象的斯坦纳三元组系统对于形如 $6k+1$ 或 $6k+3$ 的所有 $v \geqslant 1$ 存在。

11.〔$M25$〕 正文描述了同包含查询相联系时斯坦纳三元组系统的用法。为了把这推广到所有基本的查询中去，定义下列概念是自然的：阶为 v 的一个互补三元组系统是把 $2v$ 个对象 $\{x_1, \cdots, x_v, \bar{x}_1, \cdots, \bar{x}_v\}$ 编成下面这样一些三元组的一个安排，即每对对象恰在一个三元组中一起出现，但互补的对 $\{x_i, \bar{x}_i\}$ 决不一起出现。例如

$$\{x_1, x_2, x_3\}, \{x_1, \bar{x}_2, \bar{x}_3\}, \{\bar{x}_1, x_2, \bar{x}_3\}, \{\bar{x}_1, \bar{x}_2, x_3\}$$

是阶为 3 的一个互补三元组系统。

证明对于所有不是形如$3k+2$的 $v \geqslant 0$，都存在阶为 v 的互补三元组系统。

12.〔$M23$〕 继续习题11，构造阶为 7 的一个互补四元组系统。

13.〔$M25$〕 构造类似于习题 9 的三元组系统的具有 $v=4^n$ 个元素的四元组系统。

14.〔25〕 试讨论从四叉树、本德利的二维查找树以及类似于图 45 的邮局树删去节点的问题。

15.〔$HM30$〕（P. 埃利亚斯（P. Elias））给定 m 个二进位记录的一个很大集合，假设我们要来找一个记录，在大多数二进位一致的意义下，它最接近于一个给定的查找变元。假定给定了 2^n个元素的一个 m 个二进位的 t-误差校正码，而且每个记录已经"杂凑到"对应于最接近的码字的 2^n个表之一，试设计用于有效地解决这个问题的一个算法。

16.〔25〕（W. H. 考茨和 R. C. 卒格尔顿）证明阶为 v 的一个斯坦纳三元组系统可用以构造每个有 v 个二进位的 $v(v-1)/6$ 个代码字，使得任一代码字不包含在其它两个代码字的叠加码中。

习 题 答 案

"我已经回答了三个问题，而这就够了"，他的父亲说"不要装腔作势！你是不是以为我能整天地听这样的废话？滚开，否则我将一脚把你踢下楼去！"——刘易斯·卡罗尔(Lewis Carroll)（《艾丽斯漫游奇境记》❶，第5章。）

关于习题的说明

1．对于一个有数学修养的读者的一个普通的问题。

2．见W. J. 莱维克(W. J. Le Veque)"*Topics in Number Theory*"2(Reading, Mass: Addison-Wesley, 1956)，第三章。（注意：阅读过这本书原始手稿的一个人报告说，他发现了一个真正值得注意的证明，但要把这个证明写下来则嫌他那本的页边太小了。）

第五章

1．设$p(1)$，…，$p(n)$和$q(1)$，…，$q(n)$是满足这些条件的不同的排列，且设i是使$p(i)\neq q(i)$的极小值，则对某个j，k，i有$p(i)=q(j)$和$q(i)=p(k)$。由于$K_{p(i)}\leqslant K_{p(k)}=K_{q(i)}\leqslant K_{q(j)}=K_{p(i)}$，我们有$K_{p(i)}=K_{q(i)}$；因此由稳定性$p(i)<p(k)=q(i)<q(j)=p(i)$，矛盾。

2．是的，如果排序操作都是稳定的话（如果它们不稳定，则我们不能这么说）。A先生和C先生肯定有相同的结果；而且B先生也是，因为稳定性表明，在他的结果中当主键相等时，次键是按非减次序排列的。

形式上，假定B先生对次键进行排序之后，得到结果$R_{p(1)}$，…，$R_{p(N)}=R'_1$，…，R'_N，然后在对主键进行排序之后得到$R'_{q(1)}$，…，$R'_{q(N)}=R_{p(q(1))}$，…，$R_{p(q(N))}$；我们来证明：对于$1\leqslant i<N$，$(K_{p(q(i))},\ k_{p(q(i))})\leqslant(K_{p(q(i+1))},\ k_{p(q(i+1))})$。如果$K_{p(q(i))}\neq K_{p(q(i+1))}$，则我们有$K_{p(q(i))}<K_{p(q(i+1))}$；又如果$K_{p(q(i))}=K_{p(q(i+1))}$，则$K'_{q(i)}=K'_{q(i+1)}$，因此$q(i)<q(i+1)$，因此$k'_{q(i)}\leqslant k'_{q(i+1)}$，即$k_{p(q(i))}\leqslant k_{p(q(i+1))}$。

3．我们总能把具有相等键的记录送到一起，并保持它们的相对次序，在进一步的操作中把这些记录组当作一个单位来处理；因此，我们可以假定，所有的键都是不同的。设$a<b<c<a$；则我们可以把这些记录安排成头三个键为abc，bca，或cab。现在如果$N-1$个不同的键可以加以排序，则N个也能；因为如果$K_1<\cdots<K_{N-1}>K_N$，则对于某个i总有$K_{i-1}<K_N<K_i$，或者$K_N<K_1$。

4．溢出是可能的，而且它能导致一个假的相等指示。他应该编写，例如LDA A；CMPA B并查看比较指示器（不可能通过减法来做全字长的比较，这是许多计算机上的一

❶ 人民文学出版社，1956年。——译注

个问题；这是在 MIX 的指令系统中包括 CMPA，…，CMPX 的主要原因)。

5.

```
COMPARE  STJ    9F
1H       LDX    A,1
         CMPX   B,1
         JNE    9F
         DEC1   1
         J1P    1B
9H       JMP    *
```

6. 解法 1，以恒等式 $\min(a, b)=\frac{1}{2}(a+b-|a-b|)$ 为基础：

```
LDA    A                          SRAX   1
SRAX   5                          ADD    AB1
DIV    =2=                        ENTX   1
STA    A1        a = 2a1 + a2     SLAX   5
STX    A2        |a2| ≤ 1         MUL    AB2
LDA    B                          STX    AB3       (a2 - b2)sign(a - b)
SRAX   5                          LDA    A2
DIV    =2=                        ADD    B2
STA    B1        b = 2b1 + b2     SUB    AB3
STX    B2        |b2| ≤ 1         SRAX   5
LDA    A1                         DIV    =2=
SUB    B1        不可能溢出       ADD    A1
STA    AB1       a1 - b1          ADD    B1        不可能溢出
LDA    A2                         SUB    AB1(1:5)
SUB    B2                         STA    C
STA    AB2       a2 - b2
```

解法 2，以变址能以一种巧妙的方式引起交换这一事实为基础：

```
LDA    A
STA    C
STA    TA
LDA    B
STA    TB
```

现在重复下列代码 k 次，这里 $2^k>10^{10}$：

```
LDA    TA        LDA    TB        INC1   0,2
SRAX   5         SRAX   5         INC1   0,2
DIV    =2=       DIV    =2=       INC1   0,2
STX    TEMP      STX    TEMP      LD3    TMIN,1
LD1    TEMP      LD2    TEMP      LDA    0,3
STA    TA        STA    TB        STA    C
```

(它从右到左地扫描 a 和 b 的二进表示。) 最后，表的形式是

```
        HLT
        CON     C     -1     -1
        CON     B      0     -1
        CON     B     +1     -1
        CON     A     -1      0
TMIN    CON     C      0      0
        CON     B      1      0
        CON     A     -1      1
        CON     A      0      1
        CON     C      1      1
```

7．由取舍原理● （习题 1.3.3-26)得到$\sum_j\binom{r+j-1}{j}(-1)^j\binom{N}{r+j}x^{r+j}$。这是一个“β分布”$r\binom{N}{r}\int_0^x t^{r-1}(1-t)^{N-r}dt$。

8．将它进行排序，然后进行计数（某些排序方法可用来方便地筛除其键在文件的其它地方被重复的记录)。

9．给每一个人指定一个标识号，这个号必须在所有涉及他的表格中出现。以这个标识号为键，分别对信息表格和赋税的表格排序。以 R_1，…，R_N 表示排好序的赋税表格，其键为 $K_1<\cdots<K_N$（对于相等的键不应该有两张赋税表格)。增加一个新的其键为∞的记录，这是第（$N+1$）个记录并且置 $i\leftarrow 1$。然后对于信息文件中的每个记录，校验它是否已经报告如下：设K表示正在处理的信息表上的键。

a）如果$K>K_i$，则K增加1，并且重复这个步骤。

b）如$K<K_i$或者如果$K=K_i$而且信息不反映到赋税表格 R_i 上，则标出一个错误。试验来做这全部工作而不浪费纳税人的钱。

10．一个方法是附加键（j，i）到条目 $a_{i,j}$ 上，而且利用字典编辑顺序来排序，然后删去键（当能够给出重新编序的一个简单公式时，一个类似的思想可以用来得到对于信息的想要的任何重新编序)。

在这个问题所考虑的特殊情况下，“平衡的两路合并排序”方法以这样一种简单的方式来处理键，即不必明显地在带上写出它们来。给定一个 $n\times n$ 矩阵，我们可以这样做：首先在带 1 上放置奇数编号的行，在带 2 上放置偶数编号的行，等等，得到：

带 1：$a_{11}a_{12}\cdots a_{1n}a_{31}a_{32}\cdots a_{3n}a_{51}a_{52}\cdots a_{5n}\cdots$

带 2：$a_{21}a_{22}\cdots a_{2n}a_{41}a_{42}\cdots a_{4n}a_{61}a_{62}\cdots a_{6n}\cdots$

然后反绕这些带，而且同步地处理它们，以得到

带 3：$a_{11}a_{21}a_{12}a_{22}\cdots a_{1n}a_{2n}a_{51}a_{61}a_{52}a_{62}\cdots a_{5n}a_{6n}$

带 4：$a_{31}a_{41}a_{32}a_{42}\cdots a_{3n}a_{4n}a_{71}a_{81}a_{72}a_{82}\cdots a_{7n}a_{8n}$

反绕这些带，并且同步地处理它们，以得到

带 1：$a_{11}a_{21}a_{31}a_{41}a_{12}\cdots a_{42}\cdots a_{4n}a_{9\,1}\cdots$

带 2：$a_{51}a_{61}a_{71}a_{81}a_{52}\cdots a_{82}\cdots a_{8n}a_{13\,1}\cdots$

等等，直到$\lceil\log_2 n\rceil$遍扫描之后得到所希望的转置。

● 取舍原理又译为包括和排除原理。——译注

11．一个方法是附加随机的不同的键值，按这些键进行排序，然后抛弃这些键（参考习题 10；在 3.4.2 节中讨论了为得到一个随机的样品的一个类似方法）。另外一项技术需要差不多同样多的工作量但显然不太强求随机数生成程序的精确度，它附加一个在范围 $0 \leqslant K_i < N-i$ 内的随机数到 R_i 上，然后利用习题 5.1.1-5 的技术重新安排。

12．例如，准备 84 个字符的记录 $a_1a_2\cdots a_{84}$ 如下。对于每张委员会的卡片 $c_1\cdots c_{80}$，令 $a_1a_2a_3 \leftarrow c_{78}c_{79}c_{80}$，$a_4 \leftarrow$ 空白，$a_5\cdots a_{84} \leftarrow c_1\cdots c_{80}$。对于每张教职员卡片 $f_1\cdots f_{80}$，以及对于 $0 \leqslant k < 20$，在这些卡片的每个非空的场 $f_{21+3k}f_{22+3k}f_{23+3k}$，准备一个记录，填入 $a_1a_2a_3 \leftarrow f_{21+3k}f_{22+3k}f_{23+3k}$，$a_4\cdots a_{22} \leftarrow f_2\cdots f_{20}$，$a_{23}\cdots a_{21+j} \leftarrow f_1\cdots f_j$，$a_{22+j} \leftarrow$ “，”，$a_{24+j} \leftarrow f_{19}$，$a_{25+j} \leftarrow$ “．”，$a_{27+j} \leftarrow f_{20}$，以及（如果 f_{20} 为非空白的）$a_{28+j} \leftarrow$ “．”；其它位置是空白。这里 j 是 $\leqslant 18$ 且 f_j 为非空白的最大整数。（如果在整理序列中空白不是最低的，则委员会记录上的 a_4 和在教职员记录上在 $a_4\cdots a_{22}$ 中间的所有空白字符都应该被改变成为在整理序列中最低的字符。）

现在按字母顺序对这些 84 个字符的记录进行排序。在排序之后，依次地处理每个 84 个字符的记录如下：如果 a_4 是空白，则开始一个新页并打印 $a_4\cdots a_{84}$。如果 a_4 是非空白，则以 19 个空白打头打印 $a_{23}\cdots a_{84}$。（显然这个办法还需要进一步加细，以查出那些没有名字的委员会编号，及那些未参加足够数量的委员会的教职员名字。这个习题说明了通过对一个不被打印的键进行排序来打印清单的一个相当一般的方法。通过以一种同步的方式来处理委员会记录和教职员记录，分别对这两个文件排序，并且打印出它们，将有可能保持这两种记录的分开处理；但是分开可能并不足以节省进行额外的努力的时间。）

13．通过一个字符转换表，你能设计一个按字典顺序的比较程序，它模拟在其它机器上所用的次序。或者，你可以建立人工的键，它不同于实际的字符但是给出所希望的次序。后一种方法有这样一个优点：它仅仅需要做一次。但是它花费较多的空间而且要求整个键的转换。前一种方法通常可以仅仅转换键字的一个或两个字母，就确定了比较的结果；在排序的后边诸阶段，这个比较将仅仅在几乎相等的键之间进行。因此在前一种方法中，在转换字母之前校验字母的相等性也许是有优点的。

14．对于这个问题，仅需扫视这个文件一次并保持 50 个左右个别的计数。但如用“城市”来代替“州”，而且城市的总数十分大，则对于城市的名称进行排序将是一个好的想法。

15．象在习题 14 中一样它依赖于问题的大小。如果高速的存储器能放得下交叉引用的全部记录，则最好的方法大概是使用一个符号表算法（第六章），而且每个标识符与一个引用表的表头相联系。对于较大的问题，建立一个记录文件，为放置到索引中去的每个交叉引用条目都设置一个记录，并且对此文件进行排序。

16．每张卡片上设一个键，若以通常简单的方式按字典顺序对键进行排序，就能定义所要的次序。这个键应由图书馆职员提供，而且当它首次进入这个系统时附加到卡片数据上。若用 MIX 字符代码来定义整理序列，则一个可能的键使用以下的双字母代码把字彼此分开：

␣␣	键的结束
␣．	交叉引用的结束

⊔,	姓的结束
⊔(	多个姓的连字号
⊔)	作者名的结束
⊔+	地名的结束
⊔—	标题的结束
⊔*	书的题目的结束
⊔/	词之间的空格

对于给定的例子，我们将得出如下结果（仅仅给出头 25 个字符）：

ACCADEMIA⊔/NAZIONALE⊔/DEI
ACHTZEHNHUNDERT⊔/ZWOLF⊔/E
BIBLIOTHEQUE⊔/D⊔/HISTOIRE
BIBLIOTHEQUE⊔/DES⊔/CURIOS
BROWN⊔,J⊔/CROSBY⊔)⊔⊔
BROWN⊔,JOHN⊔)⊔⊔
BROWN⊔,JOHN⊔)MATHEMATICIA
BROWN⊔,JOHN⊔)OF⊔/BOSTON⊔⊔
BROWN⊔,JOHN⊔)1715⊔⊔
BROWN⊔,JOHN⊔)1715⊔-⊔⊔
BROWN⊔,JOHN⊔)1761⊔⊔
BROWN⊔,JOHN⊔)1810⊔⊔
BROWN⊔(WILLIAMS⊔,REGINALD
BROWN⊔/AMERICA⊔*⊔⊔
BROWN⊔/AND⊔/DALLISONS⊔/NE
BROWN:OHN⊔,ALAN⊔)⊔⊔
DEN⊔,VLADIMIR⊔/EDVARDOVIC
DEN⊔*⊔⊔
DEN⊔/LIEBEN⊔/SUSSEN⊔/MADE
DIX⊔,MORGAN⊔)1827⊔⊔
DIX⊔/HUIT⊔/CENT⊔/DOUZE⊔/O
DIX⊔/NEUVIEME⊔/SIECLE⊔/FR
EIGHTEEN⊔/FORTY⊔/SEVEN⊔/I
EIGHTEEN⊔/WELVE⊔/OVERTUR
I⊔/AM⊔/A⊔/MATHEMATICIAN⊔*
I⊔/B⊔/M⊔/JOURNAL⊔OF⊔/RES
J⊔/HA⊔/EHAD⊔*⊔⊔
JA⊔/A⊔/LOVE⊔/STORY⊔*⊔⊔
INTERNATIONAL⊔/BUSINESS⊔/
KHUWARIZMI⊔,MUHAMMAD⊔/IBN
LABOR⊔*⊔⊔
LABOR⊔/RESEARCH⊔/ASSOCIAT
LABOUR⊔.⊔⊔
MACCALLS⊔/COOKBOOK⊔*⊔
MACCARTHY⊔,JOHN⊔)1927⊔⊔
MACHINE⊔/INDEPENDENT⊔/COM
MACMAHON⊔,PERCY⊔/ALEXANDE
MISTRESS⊔/DALLOWAY⊔*⊔⊔
MISTRESS⊔/OF⊔/MISTRESSES⊔
ROYAL⊔/SOCIETY⊔/OF⊔/LONDO
SAINT⊔/PETERSBURGER⊔/ZEIT
SAINT⊔/SAENS⊔,CAMILLE⊔)18
SAINTE⊔/ANNE⊔/DES⊔/MONTS⊔
SEMINUMERICAL⊔/ALGORITHMS
UNCLE⊔/TOMS⊔/CABIN⊔*⊔⊔
UNITED⊔/STATES⊔/BUREAU⊔/O
VANDERMONDE⊔,ALEXANDER⊔/T
VANVALKENBURG⊔,MAC⊔/ELWYN
VONNEUMANN⊔,JOHN⊔)1903⊔⊔
WHOLE⊔/ART⊔/OF⊔/LEGERDEMA
WHOS⊔/AFRAID⊔/OF⊔/VIRGINI
WIJNGAARDEN⊔,ADRIAAN⊔/VAN

这个辅助的键应该接有卡片数据，以便正确地区别具有相同辅助键（例如 Sir John=John）的不相等卡片。

17.（D. 香克斯（D. Shanks） 给出的一个思想） 对于 $0\leqslant n<m$ 准备两个文件，一个包含 $a^{mn} \bmod p$，而另一个包含 $(ba^{-n}) \bmod p$。对这些文件进行排序而且寻找一个共同条目。

注意：这使工作量从 $O(p)$ 减少到 $O(\sqrt{p}\log_2 p)$。是否有进一步减少的可能？通过检验 $b^{(p-1)/2} \bmod p=1$ 或 $(p-1)$，在 $\log_2 p$ 步内不难确定 n 是偶数还是奇数。一般说

来，如果f是$p-1$的任何因子，以及d是$\gcd(f, n)$的任何因子，则通过在一长度为f/d的表中查找$b^{(p-1)/f}$的值，我们可以类似地确定$(n/d) \bmod f$。如果$p-1$有质因子$g_1 \leqslant g_2 \leqslant \cdots g_t$，而且$g_t$很小，则用$q_1, \cdots, q_t$作为基数，我们可通过自右至左地逐个确定$n$的各位这一方法来迅速地求出$n$的混合基数表示（这一思想是由R.L.西尔弗（R.L. Silver提出的）。

约翰·M.波拉德（John M.Pollard）〔*Math. Comp.* 32(1978)，918-924〕根据随机映射理论，已经发现了在大约$O(\sqrt{p})$个步骤内，用很小的内存来解决这个问题的一个漂亮的方法。也见习题4.5.4-39的逐渐加快的方法。

18．例如，对于$u \leqslant v \leqslant w$，$x \leqslant y \leqslant z$我们可以造包含$(u^6+v^6+w^6) \bmod W$和$(z^6-x^6-y^6) \bmod W$的值的两个文件，这里$W$是计算机的字长。对文件进行排序并且把重复的挑出来，然后对重复者施以进一步的检验。(某些对小质数求模的同余式也可用来对u，v，w，x，y，z设置进一步的限制。)

19．一般地说，有如下方法找出满足$x_i+x_j=c$的所有数对(x_i, x_j)（这里c是给定的）：对文件进行排序，使得$x_1<x_2<\cdots<x_n$。置$i \leftarrow 1$，$j \leftarrow N$，然后重复下列操作直到$j \leqslant i$：

如果$x_i+x_j=c$，则输出(x_i, x_j)和(x_j, x_i)，置$i \leftarrow i+1$，$j \leftarrow j+1$；

如果$x_i+x_j<c$，则置$i \leftarrow i+1$；

如果$x_i+x_j>c$，则置$j \leftarrow j-1$。

最后，如果$j=i$而且$2x_i=c$，则输出(x_i, x_i)。这个过程就象习题16和17的情况那样，我们实质上造两个有序文件，一个包含$x_1, \cdots, x_N$，另一个包含$c-x_N, \cdots, c-x_1$，而且校验重复。但第二个文件在这种情况下不需要明显地形成。当c为奇数时，另一个方法是对一个诸如（x奇$\Rightarrow x$，x偶$\Rightarrow c-x$）的键进行排序。

当给定t和两个排好序的文件$x_1 \leqslant \cdots \leqslant x_m$，$y_1 \leqslant \cdots \leqslant y_n$时，一个类似的算法可用来求$\max\{x_i+x_j \mid x_i+x_j \leqslant c\}$或者比如说，来求$\min\{x_i+y_j \mid x_i+y_j>t\}$。

20．某些方案是：（a）对于满足$1 \leqslant i<j \leqslant 1000$的499500个对偶$i$，$j$的每一个，置$y_1 \leftarrow x_i \oplus x_j$，$y_2 \leftarrow y_1 \wedge (y_1-1)$，$y_3 \leftarrow y_2 \wedge (y_2-1)$；然后，当且仅当$y_3=0$时，打印$(x_i, x_j)$，这里$\oplus$表示“异或”，$\wedge$表示“与”。（b）由每个原来的字$x_i$形成31个项，其中包括$x_i$本身及在一个位置上不同于$x_i$的另外30个字。这样得到一个共有31000个项的文件。对这个文件进行排序，而且考察重复性。（c）对于

i）它们的头10个二进位一致的所有字对；

ii）它们中间10个二进位（但不是头10个二进位）中一致的所有字对；

iii）它们后10个二进位（但既不是头10个二进位也不是中间10个二进位）一致的所有字对。

作类似于（a）的测试。这包含三个数据排序，并且每次使用一个确定的10个二进位的键。如果原来的字是随机分布的，则在三种情况的每一种中预期的对偶数目都小于500。

21．首先准备好包含所有五个字母的英文单词的一个文件。(别忘了把诸如-ED、-ER、-ERS、-S的后缀附加到较短的字上)。现在取每个五字母单词α，并把它的字母排为递增次序，得到有序的五个字母的序列α'。最后把所有对偶（α'，α）进行排序，来把

所有变异单词弄到一起。

金·D·吉布森（Kim. D. Gibson）于 1967 年所作的实验指出，变异字的第二个最长的集合为 LEAST，SLATE，STALE，STEAL，TAELS，TALES，TEALS。〔使用较大的字典，还可增加 ASTEL 即一个裂片；LEATS，水的汇合；STELA，'stele' 的另一拼法；TESLA，磁感应的一个单位，来扩大这个集合。而且我们也可以计入 de Staël 夫人！参考 H. E. 达德尼（H. E. Dudeney），"300 *Best Word Puzzles*"，ed. by Martin Garder（N. Y.：Chas Scribner's Sons，1968），*Puzzle* 194）。

已找到的头一个和最后一个集合分别是：ALBAS，BALAS，BALSA BASAL 和 STRUT STURT，TRUST。非预期的集合 ADDER，DARED，DREAD 和 ALGOR，ARGOL，GORAL，LARGO 对于计算机科学家可能是感兴趣的。

一个较快的处理方法是计算 $f(\alpha)=(x+a_1)(x+a_2)(x+a_3)(x+a_4)(x+a_5) \bmod m$，其中 $a_1 \cdots a_5$ 是 α 中各字母的数值代码，而 m 是计算机的字长。这里 x 是任何固定的值，在开始进行计算之前，它可以"随机地"选择。对文件（$f(\alpha)$，α）进行排序将把变异字弄到一起；然后当 $f(\alpha)=f(\beta)$ 时，我们必须确保，有一个真正的变异字，且 $\alpha'=\beta'$。$f(\alpha)$ 的值可以比 α' 更快地计算，对于文件中大多数字 α，用这个方法可以不必确定 α'。

注意：当我们要把所有有相等的 n 字键（$a_1a_2\cdots a_n$）的记录集合弄到一起时，可以使用一个类似的技术。假设除了把具有相等键的记录弄到一起外，我们不关心文件的次序，则对于单字的键（$a_1x^{n-1}+a_2x^{n-2}+\cdots+x_n$）$\bmod m$ 而不是对原来的 n 字键进行排序，有时更快些。

22．求图形的同构不变量（即，在同构的有向图上取相等值的函数）并对它们进行排序，来把"明显的非同构"的图形彼此分开。同构不变量的例子是：（a）以（a_i，b_i）表示顶点 v_i，这里 a_i 是它的输入次数，b_i 是它的输出次数；然后把对偶（a_i，b_i）排为字典编辑顺序。得到的文件是同构不变量。（b）用（a_i，b_i，a_j，b_j）表示从 v_i 到 v_j 的一条弧，并且把这四元组排成字典编辑顺序。（c）把有向图分开成为连通分量（参考算法 2.3.3 E），确定每个分量的不变量，并且以某种方式把这些分量按它们的不变量的次序排列。也见习题 21 中的讨论。

在按它们的不变量对有向图进行排序之后，一般仍然有必要进行第二次测试看是否具有相等不变量的有向图确是同构的。这些不变量对于这些测试也是有帮助的。在树的情况下，有可能来找出"特征的"或"规范的"不变量，它们完全地表征树，使我们不必进行第二次测试〔见 H. J. Scoins，*Machine Intelligence* 3（1969），43-60〕。

23．一个方法是形成包含所有三个人的集体的一个文件，然后把它变换为含有所有的四个人的集体的文件，等等；如果没有很大的集体，则这个方法将是十分令人满意的另一方面，如果有 n 个人的集体，则至少有 $\binom{n}{k}$ 个 k 个人的集体，所以甚至当 n 仅仅是20左右时，这个方法就可能失败。

以（a_1，…，a_{k-1}）的形式给定一个文件，它列出所有 $k-1$ 个人的集体，这里 $a_1<\cdots<a_{k-1}$，则找出 k 个人的集体的方法如下，（i）建立一个新文件，对于每对分别有形式

$(a_1, \cdots, a_{k-2}, b)$，$(a_1, \cdots, a_{k-2}, c)$ 且有 $b<c$ 的（$k-1$）个人的集体，新文件将含有项（b，c，a_1，…，a_{k-2}）；(ii) 按它的头两个分量对这个文件进行排序；(iii) 这个新文件的每个项（b，c，a_1，…，a_{k-2}）(它是匹配原来给定文件中的一对（b，c）的)，输出 k 个人的集体（a_1，…，a_{k-2}，b，c）。

24. 作输入文件的另一个复份；按照头一个分量对一个复份进行排序，按第二个分量对另一个进行排序。顺序地扫描这些文件，即可对于 $1 \leqslant i \leqslant n-2$ 建立含有所有对偶 (x_i, x_{i+2}) 的一个新文件，并确认 (x_{N-1}, x_N)。对偶 $(N-1, x_{N-1})$ 和 (N, x_N) 应写到另一个文件上。

归纳地继续这个过程，假定对于 $1 \leqslant i \leqslant N-t$，文件 F 以随机次序包含所有对偶 (x_i, x_{i+t})，而且对于 $N-t<i \leqslant N$ 文件 G 以第二个分量的顺序，包含所有对偶 (i, x_i)。设 H 是文件 F 的一个复份，而且通过头一个分量对 H 排序，通过第二分量对 F 排序。现在扫描 F，G 和 H，同时建立两个新文件 F' 和 G' 如下。如果文件 F，G，H 的当前记录分别为（x，x'），（y，y'），（z，z'）则：

i）如果 $x'=z$，则输出（x，z'）到 F'，且推进文件 F 和 H。

ii）如果 $x'=y'$，则输出（$y-t$，x）到 G'，且推进文件 F 和 G。

iii）如果 $x'>y'$，则推进文件 G。

iv）如果 $x'>z$，则推进文件 H。

当文件被取尽时，通过第二个分量对 G' 排序而且把 G 同它合并；然后以 $2t$ 代替 t，以 F' 代替 F，以 G' 代替 G。

于是 t 取值 2，4，8，…；对于固定的 t 我们扫描数据 $O(\log_2 n)$ 次来对它进行排序；因此扫描的总数是 $O((\log_2 n)^2)$。最后 $t \geqslant N$，此时 F 是空的；然后我们只需按 G 的头一个分量对 G 进行排序即可。

这个巧妙的方法归功于诺曼·哈迪（Norman Hardy）。通过在识别 x_i 的那趟扫描上，同时识别 x_{N+1-i}，就可以某种复杂性为代价，节省一个阶段。

5.1.1 节

1. 2 0 5 2 2 3 0 0 0；2 7 3 5 4 1 8 6。

2. $b_1=(m-1) \bmod n$；$b_{j+1}=(b_j+m-1) \bmod (n-j)$。

3. $\bar{a}_j=a_{n+1-j}$（"反演的排列"）。这一思想为 O·特昆姆（O. Terquem）[*Journ. de Math.*（1）3（1838），559-560] 用来证明在一个随机排列中反序的平均数是 $\frac{1}{2}\binom{n}{2}$。

4. **C1**. 置 $x_0 \leftarrow 0$。(对于 $1 \leqslant j \leqslant n$，有可能让 x_j 在以下的诸步骤中同 b_j 共享存储)

   C2. 对于 $k=n$，$n-1$，…，1（以这一次序）进行下列工作：置 $j \leftarrow 0$；然后置 $j \leftarrow x_j$ 恰 b_k 次；然后置 $x_k \leftarrow x_j$ 且 $x_j \leftarrow k$。

   C3. 置 $j \leftarrow 0$。

   C4. 对于 $k=1$，2，…，n（以这一次序）进行下列工作：置 $a_k \leftarrow x_j$，然后置 $j \leftarrow x_j$。

节省存储空间的问题见习题 5.2-12。

5．命α是非负整数的有序对的一个串$[m_1, n_1]\cdots[m_k, n_k]$；以$|\alpha|=k$表示$\alpha$的长度。命$\epsilon$表示空串（长度为0）。考虑在这种有序对上递归地定义的二元操作：

$$\epsilon\circ\alpha=\alpha\circ\epsilon=\alpha;$$

$$([m, n]\alpha)\circ([m', n']\beta)=\begin{cases}[m, n](\alpha\circ[m'-m, n']\beta)), & \text{如果}m\leqslant m'\\ [m', n'](([m-m'-1, n]\alpha)\circ\beta), & \text{如果}m>m'\end{cases}$$

由此得出为求$\alpha\circ\beta$的值所需要的时间与$|\alpha\circ\beta|=|\alpha|+|\beta|$成比例。其次，我们可以证明$\circ$是可以结合的，而且$[b_1, 1]\circ[b_2, 2]\circ\cdots\circ[b_n, n]=[0, a_1][0, a_2]\cdots[0, a_n]$。左边表达式可以在$\lceil\log_2 n\rceil$次扫描中计算出来，每次扫描都结合一些串对，总共为$O(n\log_2 n)$个步骤。

例子：从（2）开始，我们来求$[2, 1]\circ[3, 2]\circ[6, 3]\circ[4, 4]\circ[0, 5]\circ[2, 6]\circ[2, 7]\circ[1, 8]\circ[0, 9]$的值。头一次扫描把它归结为$[2, 1][1, 2]\circ[4, 4][1, 3]\circ[0, 5][2, 6]\circ[1, 8][0, 7]\circ[0, 9]$。第二次扫描把它归结为$[2,1][1,2][1,4][1,3]\circ[0, 5][1, 8][0, 6][0, 7]\circ[0, 9]$。

第三次扫描得到

$$[0, 5][1, 1][0, 8][0, 2][0, 6][0, 4][0, 7][0, 3]\circ[0, 9]$$

第四次扫描得出（1）。

动机：一个诸如$[4, 4][1, 3]$的串表示"␣␣␣␣4␣3"，这里"␣"表示稍后有待填入的一个很大的数。注意，同习题2一起，我们得到对于约瑟夫斯问题的一个算法，它是$O(n\log n)$而不是$O(mn)$部分地回答了在习题1.3.2-22中产生的问题。

以一种直截了当的方式使用平衡树，即得到对于这个问题的另一个$O(n\log_2 n)$的解，这种解法使用一个随机存取的存储器。

6．由$b_1=b_2=\cdots=b_n=0$开始，对于$k=\lfloor\log_2 n\rfloor, \lfloor\log_2 n\rfloor-1, \cdots, 0$，进行如下：对于$0\leqslant s\leqslant n/2^{k+1}$置$x_s\leftarrow 0$，然后对于$j=1, 2, \cdots, n$进行如下：置$r\leftarrow\lfloor a_j/2^k\rfloor \bmod 2$，$s\leftarrow\lfloor a_j/2^{k+1}\rfloor$（这些实质上是位的抽取）；如果$r=0$，则置$b_{a_j}\leftarrow b_{a_j}+x_s$，如果$r=1$则置$x_s\leftarrow x_s+1$。

另一个答案出现于习题5.2.4-21当中。

7．$B_j<j$且$C_j\leqslant n-j$，因为a_j左边有少于j个元素，而右边有$n-j$个元素。为了从$B_1, B_2, \cdots, B_n$重新构造$a_1a_2\cdots a_n$，由元素1开始；对于$k=2, \cdots, n$，对每个$\geqslant k-B_k$的元素加1并附加$k-B_k$于右边（参考1.2.5节的方法2）。对C可用一个类似的过程，或者，我们可以使用下列习题的结果。

8．$b'=C$，$c'=B$，$B'=c$，$C'=b$，因为$a_1\cdots a_n$的每对反序(a_i, a_j)对应于$a'_j\cdots a'_n$的反序(j, i)。某些进一步的关系：（a）$c_j=j-1$当且仅当对于所有的$i<j$有$b_i>b_j$；（b）$b_j=n-j$当且仅当对于所有的$i>j$有$c_i>c_j$；（c）$b_j=0$当且仅当对于所有的$i>j$有$c_i-i<c_j-j$；（d）$c_j=0$当且仅当对于所有的$i<j$有$b_i+i<b_j+j$；（e）$b_i\leqslant b_{i+1}$当且仅当$c_i\geqslant c_{i+1}$；（f）$a_j=j+C_j-B_j$，$a'_j=j+b_j-c_j$。

9．$b=C=b'$等价于$a=a'$。

10．$\sqrt{10}$。（对截八面体建立坐标的一种方式是命下列向量$(1, 0, 0)$，$(0, 1, 0)$，

$\frac{1}{2}(1, 1, \sqrt{2})$，$\frac{1}{2}(1, -1, \sqrt{2})$，$\frac{1}{2}(-1, 1, \sqrt{2})$，$\frac{1}{2}(-1, -1, \sqrt{2})$分别代表对偶 21，43，41，31，42，32 的相邻的交换。这些向量的和给出（1，1，$2\sqrt{2}$）作为顶点 4321 与 1234 之间的差。）

一个更对称的解是通过

$$\Sigma\{\mathbf{e}_u-\mathbf{e}_v|(u, v)\text{是}\pi\text{的一个反序}\}$$

来表示四维中的顶点 π，这里 $\mathbf{e}_1=(1, 0, 0, 0)$，$\mathbf{e}_2=(0, 1, 0, 0)$，$\mathbf{e}_3=(0, 0, 1, 0)$，$\mathbf{e}_4=(0, 0, 0, 1)$。于是 1 2 3 4 $\leftrightarrow(0, 0, 0, 0)$；1 2 3 4 $\leftrightarrow(0, 0, -1, 1)$；…；4 3 2 1 $\leftrightarrow(-3, -1, 1, 3)$。所有的点都位于三维子空间$\{(w, x, y, z)|w+x+y+z=0\}$上；两个相邻顶点之间的距离为$\sqrt{2}$。等价地（参考习题 8 中（i））我们可以通过向量（a'_1，a'_2，a'_3，a'_4）表示 $\pi=a_1a_2a_3a_4$，这里 $a'_1a'_2a'_3a'_4$ 是逆排列。（这个以排列作为坐标的截八面体的 4 维表示，以及 n 维推广，C·霍华德·欣顿曾在 *"The Fourth Dimension"* (London, 1904), Chapter 10 中讨论过。）

把无数截八面体拼接起来可以"最简单的"方式充填三维空间〔见 H. Steinhaus, *Mathematical Snapshots* (Oxford, 1960), 200-203; C. S. Smith, *Scientific American* 190 (January, 1954), 58-64〕。关于 14 个阿基米德体的图示（即非棱形多面体，它的面是正规的多面形），见 W. W. Rouse, *"Mathematical Recreations and Essays"*, rev. by H. S. M. Coxeter (Macmillan, 1939), 129-140; H. Martin Cundy and A. P. Rollett, *Mathematical Models* (Oxford, 1952), 94-109。

11. (a) 显然。(b) 由诸顶点（1，2，…，n）及如果 $x>y$ 且（x，y）$\in E$ 或 $x<y$ 且（y，x）$\in E$时由 $x\to y$ 形成的弧，构造一个有向图。如果没有有向回路，则这个有向图可以拓扑地进行排序，而且得到的线性次序是所希望的排列。如果有一个有向的回路，则它的长度至少为 3，因为没有长度为 1 或 2 的，而且由于一个更长的回路 $a_1\to a_2\to a_3\to a_4\to\cdots\to a_1$ 可以被缩短（或者 $a_1\to a_3$ 或者 $a_3\to a_1$）。但是长度为 3 的一条有向回路包含 E 或 $\overline{E}$ 的两条弧，这证明了 E 或 $\overline{E}$ 毕竟不是传递的。

12. 〔C. 伯奇 (C. Berge), *Principles de Combinatoire* (Paris, 1968), 114-117.〕假设（a，b）$\in\overline{E}$，（b，c）$\in\overline{E}$，（a，c）$\notin\overline{E}$。则对于某个 $k\geqslant 1$，我们有 $a=x_0>x_1>\cdots>x_k=c$，这里（x_i，x_{i+1}）$\in E(\pi_1)\cup E(\pi_2)$，其中 $0\leqslant i<k$。考虑这种类型的一个反例，其中 k 为极小。由于（a，b）$\notin E(\pi_1)$ 和（b，c）$\notin E(\pi_1)$，我们有（a，c）$\notin E(\pi_1)$，而且类似地（a，c）$\notin E(\pi_2)$；因此 $k>1$。但是如果 $x_1>b$，则（x，b）$\subset\overline{E}$，同 k 的极小性矛盾，而（x_1，b）$\in E$ 意味着（a，b）$\in E$。类似地如果 $x_1<b$ 则我们发现（b，x_1）$\in\overline{E}$ 和（b，x_1）$\in\overline{E}$ 和（b，x_1）$\in E$ 都是不可能的。

13. 对于反序表中 b_1，…，b_{m-1}，b_{m+1}，…，b_n 的任何固定的选择，当 b_m 跑遍它所有可能的值 0，1，…，$m-1$ 时，总和 $\Sigma_i b_i$ 将恰取每一 mod n lo m 剩余一次。

14. 题中提示的构造把分划的对偶互相转换，两个情况 $j=k=p_k$ 和 $j=k=p_k-1$ 除外。在例外的情况下，n 分别为 $(2j-1)+\cdots+j=(3j^2-j)/2$ 及 $(2j)+\cdots+(j-1)=(3j^2+j)/2$，而且有具有 j 个部分的一个唯一的不成对的分划〔欧拉原来的 i，在 *Novi Comment. acad. sc. Pet.* 5(1754)75-83上，也是非常有趣的，他通过简单的运算证明

了无穷乘积等于 s_1，其中对于 $n\geqslant 1$　$s_n=1-z^{2n-1}-z^{3n-1}s_{n+1}$〕。

15．转置点的图式，以从诸 p 进行到诸 P。容易得到诸 P 的生成函数，因为我们首先选择任意数量的 1（生成函数 $1/(1-z)$），然后独立地选择任意数量的 2（生成函数 $(1-z^2)$），…，最后任意数量的 n。

16．在头一个恒等式中 z^nq^m 的系数是把 m 分划成至多 n 部分的个数。在第二个恒等式中，它是把 m 分成 n 个不同的非负部分的个数：即 $m=p_1+p_2+\cdots+p_n$，其中 $p_1>p_2>\cdots>p_n\geqslant 0$。这和 $m-\binom{n}{2}=q_1+q_2+\cdots+q_n$ 相同，其中 $q_1\geqslant q_2\geqslant\cdots\geqslant q_n\geqslant 0$，对应关系为 $q_i=p_i-n+i$〔*Commentariiacademiæ scientiarum Petropolitanæ* 13(1741)，64-93〕。

17．

0000	0010	0100	0001
1011	1021	1201	2101
0101	0110	0210	2010
1101	1110	1210	2110
1001	1010	1100	2100
2102	2120	2210	3210

18．令 $q=1-p$。对于所有反序 α 求和的和数 $\Sigma Pr(\alpha)$，也可以通过对 k 求和来计算，这里 $0\leqslant k<n$ 是具下列性质的最左边的二进位位置的精确个数，在这些位置中，i 和 j 之间有等式成立且对于 $i<j$ 在一个反序 $X_i\oplus i>X_j\oplus j$ 中的 X_i 和 X_j 之间也有等式成立。由此我们得到公式 $\Sigma_{0\leqslant k\leqslant n}2^k(p^2+q^2)^k(p^2 2^{n-k-1}2^{n-k-1}+2pq2^{n-k-1}(2^{n-k-1}-1))$；求和并简化，得到 $2^{n-1}(p(2-p)(2^n-(p^2+q^2)^n)/(2-p^2-q^2)+(p^2+q^2)^n-1)$。

19．〔*Proc. Amer. Math. Soc.* 19(1968)，236-240。〕假设 $n>1$，给定一个排列 $a_1\cdots a_n$。通过归纳法，我们可以假设 $a_1\cdots a_{n-1}$ 对应于 $b_1\cdots b_{n-1}$，这里下标 $(a_1\cdots a_{n-1})=$ 反序 $(b_1\cdots b_{n-1})$ 而且 $a_{n-1}=b_{n-1}$。情况 1，$a_{n-1}<a_n$。令 $b_1\cdots b_{n-1}=\alpha_1x_1\cdots\alpha_rx_r$，其中 $\alpha_1,\cdots,\alpha_r$ 是 $>a_n$ 的元素的（可能为空的）串，而且 $x_1,\cdots,x_r$ 是 $<a_n$ 的元素。然后我们把 $a_1\cdots a_n$ 弄成 $x_1\alpha_1\cdots x_r\alpha_ra_n$。情况 2，$a_{n-1}>a_n$。和情况 1 相同，只是交换“<”和“>”。这个变换显然是可逆的，因为可以通过比较开头和最后的元素来区分情况 1 和情况 2。

例子：2 7 1 8 3 6 4 弄成为 7 2 8 1 6 3 4；因此 2 7 1 8 3 6 4 5 不变。

20．见 E. M. 赖特（E. M. Wright），*J. London Math. Soc.* 40(1965)，55-57，以及 J. 佐尔诺斯基（J. Zolnowsky），*Discrete Math.* 9(1974)，293-298。

21．把 C_j 解释成第 j 次输出后栈上的元素个数。

22．$\lfloor mj/n\rfloor-\lfloor mi/n\rfloor-\lfloor m(j-i)/n\rfloor=0$ 或 1；并且当且仅当 $mj \bmod n>mi \bmod n$ 时它才是零。因此，反序的个数为 $\Sigma_{0<i\leqslant j<n}(\lfloor mj/n\rfloor-\lfloor mi/n\rfloor-\lfloor m(j-i)/n\rfloor)=\Sigma_{0<r<n}\lfloor mr/n\rfloor(r-(k-r)-(k-r-1))$，可以把它变换成 $\frac{1}{4}(n-1)(n-2)-\frac{1}{4}n\sigma(m,n,0)$〔*J. für die reine und angew. Math.* 198(1957)，162—166.〕。

5.1.2 节

1．假的（由于一项相当重要的技术细节）。如果你说是“真的”，你大概不知道在 4.6.3 节中给出的 $M_1\cup M_2$ 的定义，它具有这样的性质：当 M_1 和 M_2 为集合时，$M_1\cup M_2$ 也是集合。实际上 $\alpha\top\beta$ 是 $M_1\uplus M_2$ 的一个排列。

2. $b\ c\ a\ d\ d\ a\ d\ a\ d\ b$。

3. 肯定不是，因为我们可以有 $\alpha=\beta$（然而，唯一因子分解定理证明，没有太多的可能性）。

4. $(d)\top(b\ c\ d)\top(b\ b\ c\ a\ d)\top(b\ a\ b\ c\ d)\top(d)$。

5. 对偶…xx…出现的个数，总是或者等于 $\begin{smallmatrix}x\\x\end{smallmatrix}$ 的列数，或者 $\begin{smallmatrix}x\\x\end{smallmatrix}$ 的列数减 1。若 x 是最小的元素时，则当且仅当 x 不是排列中的头一个时诸出现的个数相等。

6. 计算两行阵列的相关数是容易的：$\binom{m}{k}\binom{n}{k}$。

7. 利用定理 B 的(a)部分，类似于(20)的推导给出

$$\binom{A-1}{A-k-m-1}\binom{B}{m}\binom{C}{k}\binom{B+k}{B-l}\binom{C-k}{l}$$

$$\binom{A-1}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k-1}{B-l-1}\binom{C-k}{l}$$

$$\binom{A-1}{A-k-m}\binom{B}{m}\binom{C}{k}\binom{B+k-1}{B-l}\binom{C-k}{l}$$

8. 完全的质因子分解为 $(d)\top(b\ c\ d)\top(b)\top(a\ d\ b\ c)\top(a\ b)\top(b\ a\ d)\top(d)$，它是唯一的，因为没有相邻的对偶可交换。所以有 8 个解，即 $\alpha=\epsilon$，(d)，$(d)\top(b\ c\ d)$，…。

10. 假的，但在一些有趣的情况下为真。给定质数的任何线性次序，至少有一个所述形式的因子分解，因为每当违犯这个条件时，我们可以作一个交换来减少因子分解中的"反序"数目。所以条件不成立的原因仅仅是某些排列有多于一个这样的因子分解。

设 $\rho\sim\sigma$ 意味着 ρ 同 σ 可交换。对于所述的因子分解的唯一性说来，下列条件是必要和充分的：

$$\rho\sim\sigma\sim\tau \text{ 且 } \rho\prec\sigma\prec\tau \text{ 意味着 } \rho\sim\tau$$

证明：如果 $\rho\sim\sigma\sim\tau$ 和 $\rho\prec\sigma\prec\tau$ 且 $\rho\not\sim\tau$，则我们将有两个因子分解 $\sigma\top\tau\top\rho=\tau\top\rho\top\sigma$；因此条件是必要的。反之，为了证明它对于唯一性是充分的，令 $\rho_1\top\cdots\top\rho_n=\sigma_1\top\cdots\top\sigma_n$ 是满足条件的两个不相同的因子分解，我们可以假定 $\sigma_1\prec\rho_1$，因此对于某个 $k>1$ 有 $\sigma_1=\rho_k$；其次，对于 $1\leqslant j<k$，$\sigma_1\sim\rho_j$。由于 $\rho_{k-1}\sim\sigma_1=\rho_k$，我们有 $\rho_{k-1}\prec\sigma_1$；因此 $k>2$。设 j 使得 $\sigma_1\prec\rho_j$，而且对于 $j<i<k$，有 $\rho_i\prec\sigma_1$，则 $\rho_{j+1}\sim\sigma_1\sim\rho_j$ 和 $\rho_{j+1}\prec\sigma_1\prec\rho_j$ 意味着 $\rho_{j+1}\sim\rho_j$；因此 $\rho_j\prec\rho_{j+1}$，矛盾。

因此，若给定在质元素的一个集合 S 上的一个有序关系，它满足上述条件，而且如果知道一个排列 π 的所有质因子属于 S，则我们可以得出结论说 π 有所述类型的一个唯一的因子分解。例如，当 S 是 (29) 中循环的集合时，这样的条件成立。

但是，不能用这种办法对所有质元素的集合排序，因为如果有 $(a\ b)\prec(d\ e)$，则我们被迫定义

$$(a\ b)\prec(d\ e)\succ(b\ c)\prec(e\ a)\succ(c\ d)\prec(a\ b)\succ(d\ e)$$

矛盾。(也请看下一习题。)

11. 我们希望证明，如果$p(1)\cdots p(t)$是$(1,\cdots,t)$的一个排列，则当且仅当$\sigma_{p(1)}\top\cdots\top\sigma_{p(t)}=\sigma_1\top\cdots\top\sigma_t$时排列$x_{p(1)}\cdots x_{p(t)}$被拓扑地排序；而且如果$x_{p(1)}\cdots x_{p(t)}$和$x_{q(1)}\cdots x_{q(t)}$是不同的拓扑排序，则我们对于某个$j$有$\sigma_{p(j)}\neq\sigma_{q(j)}$。第一个性质可由下列事实推出：$x_{p(1)}$成为一个拓扑排序中的头一个的充要条件是$\sigma_{p(1)}$与$\sigma_{p(1)-1}\cdots\sigma_1$可交换(但不同于它们)；而且这个条件意味着$\sigma_{p(2)}\top\cdots\top\sigma_{p(t)}=\sigma_1\top\cdots\top\sigma_{p(1)-1}\top\sigma_{p(1)+1}\top\cdots\top\sigma_t$，所以我们可以使用归纳法。第二个性质可由下列事实推出，如果j是使$p(j)\neq q(j)$中之极小的，则由拓扑排序的定义，我们有，比如说$p(j)<q(j)$以及$x_{p(j)}\not<x_{q(j)}$；$\sigma_{p(j)}$同$\sigma_{q(j)}$没有公共的字母。

为了得到一个任意的偏序，设循环σ_k由所有使得$x_i<x_j$且$i=k$或$j=k$的有序对偶(i,j)组成；这些有序对偶以某种任意次序作为循环的元素出现。于是对于偏序$x_1\prec x_3$，$x_2\prec x_4$，$x_1\prec x_4$的循环将是$\sigma_1=((1,3)(1,4))$，$\sigma_2=((2,4))$，$\sigma_3=((1,3))$，$\sigma_4=((2,4)(1,4))$。

12. 不能形成其它循环。因为，例如，原来的排列不包括$\begin{smallmatrix}a\\c\end{smallmatrix}$列。如果$(a\ b\ c\ d)$出现$s$次，则$(a\ b)$必然出现$A-r-s$次，因为有$A-r$个$\begin{smallmatrix}a\\b\end{smallmatrix}$列，而且仅仅两类循环贡献这样的列。

13. 在两行的记号下，首先放置形如$\begin{smallmatrix}d\\a\end{smallmatrix}$的$A-i$个列，然后放置其它的$i$个$a$于第二行中，然后放置诸$b$，最后放置剩下的字母。

14. 由于在π^{-1}的两行记号下，任意给定的字母下边的元素，是处于非减次序的，我们不总有$(\pi^{-1})^{-1}=\pi$；但是$(((\pi)^{-1})^{-1})^{-1}=\pi^{-1}$是真的。事实上，恒等式

$$(\alpha\top\beta)^{-1}=((\alpha^{-1}\top\beta^{-1})^{-1})^{-1}$$

对于所有的α，β成立。

给定一多重集合，其不同的字母为$x_1<\cdots<x_m$，注意它们的每一个都有形如$\beta_1\top\cdots\top\beta_m$的唯一质因子分解，这里$\beta_j$有0个或多个质因子$(x_j)\top\cdots\top(x_j)\top(x_jx_{k_1})\top\cdots\top(x_jx_{k_t})$，$j<k_1\leqslant\cdots\leqslant k_t$，我们可以此表征它的自逆排列。例如，$(a)\top(a\ b)\top(a\ b)\top(b\ c)\top(c)$是一个自逆排列，$(m\cdot a,\ n\cdot b)$的自逆排列个数因此是$\min(m,n)+1$；$(l\cdot a,\ m\cdot b,\ n\cdot c)$的自逆排列个数是不等式$x+y\leqslant l$，$y+z\leqslant m$，$z+x\leqslant n$的非负整数解$x$，$y$，$z$的个数。一个集合的自逆排列的个数在5.1.4节中考虑。

在$(n_1\cdot x_1,\cdots,n_m\cdot x_m)$的所有排列中，在它们的两行记号中有$n_{ij}$次$x_i$出现于$x_j$之上者，其个数为$\pi_i n_i!/\pi_{i,j}n_{ij}!$，和在两行记号中有$n_{ij}$次$x_i$在$x_j$之下出现的个数相同。因此，通过给出一个适当的一一对应，应该有另外的方法来定义一个多重集合排列的逆。

15. 见定理2.3.4.2 D，撤消该有向图的一条弧，定能得到一棵有向树。

16. 如果多重集合是$(n_1\cdot x_1,\ n_2\cdot x_2\cdots)$，且$x_1<x_2<\cdots$，则诸$x_j$的反序表的项必然有形式$b_{j1}\leqslant\cdots\leqslant b_{jn_j}$，其中$b_{jn_j}$(最右边$x_j$的反序个数)至多是$n_{j+1}+n_{j+2}+\cdots$。所以反序表的第$j$部分的生成函数就是诸分划的生成函数，这些分划至多分成n_j部分，其中任一部分不超出$n_{j+1}+n_{j+2}+\cdots$。分成至多m个部分，其中任一部分不超过n的分划的生成函数，是z的二项式系数$\binom{m+n}{m}_z$；这容易通过归纳法证明，而且借助于西尔威斯特(Sylvester)

〔*Amer. J. Math.* 5 (1882), 268-269〕给出的巧妙的构造，或者波利亚（Polya）〔*Elemete der Mathematik*) 26 (1971), 102-109〕给出的另一个巧妙的构造，也可以证明。对于 $j=1,2,\cdots$ 把诸生成函数相乘，即给出所求的结果。

17．设 $h_n(z)=(n!z)/n!(1-z)^n$；所求的生成函数是 $g(z)=h_n(z)/h_{n_1}(z)h_{n_2}(z)\cdots$。由等式 5.1.1-12，$h_n(z)$ 的平均值是 $\frac{1}{2}-\left[\frac{n}{2}\right]$，所以 g 的平均值是

$$-\frac{1}{2}\left(\binom{n}{2}-\binom{n_1}{2}-\binom{n_2}{2}\cdots\right)=\frac{1}{4}(n^2-n_1^2-n_2^2\cdots)$$

18．是的。习题 5.1.1-17 的构造可以直截了当地加以推广。或者，我们可以推广 5.2.1 节中给出的证明，构造以 m 元组 $(q_1,\cdots,q_m)$ 为一方，n 元组的有序对 $((a_1,\cdots,a_n),(p_1,\cdots,p_n))$ 为另一方，这两者之间的一一对应，其中 q_j 是含有 n_j 个非负整数的一个多重集合，$a_1\cdots a_n$ 是 $(n_1\cdot 1,\cdots n_m\cdot m)$ 的一个排列，而且 $p_1\geqslant\cdots\geqslant p_n\geqslant 0$。这个对应如同以前一样定义，以非增的形式插入 q_j 的诸元素；它满足条件

$$(\Sigma_{q1})+\cdots+(\Sigma_{qm})=J(a_1\cdots a_n)+(p_1+\cdots+p_n)$$

其中 Σ_{qj} 是 q_j 的元素之和〔关于在这个证明中所用技术的进一步推广（连同在 5.1.3 节的等式（8）的推导），请看 D. E. 克努特 *Math. Comp.* 24(1970), 955-961，也可以看理查德·P. 斯坦利（Richard P. Stanley）的内容丰富的长文，(*Memoirs Amer. Math. Soc.*) 119, (1972), 104pp.〕。

19．(a) 设 $S=\{\sigma\mid\sigma$ 是质元素，σ 是 π 的左因子$\}$。如果 S 有 k 个元素，π 的使 $\mu(\lambda)\neq 0$ 的诸左因子 λ 恰巧是 s 子集的 2^k 个插入（请看定理 C 的证明）；因此 $\Sigma\mu(\lambda)=\pi_{\sigma\in S}(1+\mu(\sigma))=0$，因为 $\mu(\sigma)=-1$ 而且 s 非空。(b) 如果对于某个 $j\neq k$，有 $i_j=i_k$，则显然 $\epsilon(i_1\ldots i_n)=\mu(\pi)=0$，否则当 $i_1\cdots i_n$ 有 r 个反序时，$\epsilon(i_1\cdots i_n)=(-1)^r$；当 $i_1\cdots i_n$ 有 s 个偶循环时这就是 $(-1)^s$；当 $i_1\cdots i_n$ 有 t 个循环时它是 $(-1)^{n+t}$。

20．(a) 根据插入的定义，显然。(b) 根据定义

$$\det(b_{ij})=\sum_{1\leqslant i_1,\cdots,i_m\leqslant m}\epsilon(i_1\cdots i_m)b_{1i_1}\cdots b_{mi_m}$$

置 $b_{ij}=\delta_{ij}-a_{ij}x_j$，这变成为

$$\sum_{n\geqslant 0}\ \sum_{1\leqslant i_1,\cdots,i_n<m}(-1)^n x_{i_1}\cdots x_{i_n}\mu(x_{i_1}\cdots x_{i_n})\vee(x_{i_1}\cdots x_{i_n})$$

因为 $\mu(\pi)$ 通常为 0。

(c) 如果我们把诸 x 的乘积看作是不可交换变量的序列，则利用习题19(a)可证明，$D\top G=1$，这里用了自然的代数约定 $(\alpha+\beta)\top\pi=\alpha\top\pi+\beta\top\pi$

5.1.3 节

1．我们只须证明对于 $x=k$，当 $k\geqslant 1$ 时，这个值使（11）成立。公式变成为

$$\begin{aligned}k^n&=\sum_{0\leqslant j\leqslant r\leqslant k}(-1)^j(r-j)^n\binom{n+1}{j}\binom{n+k-r}{n}\\&=\sum_{0\leqslant s\leqslant k}s^n\sum_{0\leqslant j\leqslant k-s}(-1)^j\binom{n+1}{j}\binom{n+k-s-j}{n}\end{aligned}$$

当 $s<k$ 时，对 j 的求和可扩展到范围 $0\leqslant j\leqslant n+1$ 上，而且它为 0（j 的 n 次多项式的第 $n+1$ 次差分）。

2．(a)包含元素（1，2，…，q）的每一个至少一次的序列 $a_1a_2\cdots a_n$ 的个数为 $\left\{{n \atop q}\right\}q!$（参考习题1.2.6-64)；满足类似于(10)的式子(对于 $m=q$)的这类序列的个数为 $\binom{n-k}{n-q}$，因为我们必须选择 $n-q$ 个可能的＝符号。(b) 对 $m=n-q$ 和 $m=n-q-1$ 分别求(a)的结果，并相加之。

3．由(20)

$$\sum\frac{x^n}{n!}\sum\left\langle{n \atop k}\right\rangle(-1)^k=\sum g_n(-1)x^n=-2/(e^{-2x}+1)$$

$$=-\frac{1}{x}\left(\frac{(-2x)}{e^{-2x}-1}-\frac{(-4x)}{e^{-4x}-1}\right)$$

$$=-\frac{1}{x}\sum_{n\geqslant 0}\frac{B_nx^n}{n!}((-2)^n-(-4)^n)$$

因此结果是 $(-1)^nB_{n+1}2^{n+1}(2^{n+1}-1)/(n+1)$。或者，因为 $-2/(e^{-2x}+1)=-(1+\tanh x)$，当 n 是奇数时，我们可以把答案表示成 $(-1)^{(n+1)/2}T_n$，这里 T_n 表示由公式

$$\tan z=T_1z+T_3z^3/3!+T_5z^5/5!+\cdots$$

所定义的正切数。当 $n>0$ 为偶数时，由(7)，这和显然为 0。

注意，由(18)，我们得到新奇的恒等式

$$\sum_k\left\{{n \atop k}\right\}k!\left(-\frac{1}{2}\right)^k=2B_{n+1}(1-2^{n+1})/(n+1)$$

4．$(-1)^{n+m}\left\langle{n \atop m+1}\right\rangle$(考虑(18)中 z^{m+1} 的系数)。

5．首先我们通过公式(13)，对于 $1\leqslant k\leqslant p$，求得 $\left\langle{p-1 \atop k}\right\rangle \bmod p=1$。由定义的递归式(2)，对于 $1\leqslant k\leqslant p$ 有 $\left\langle{p \atop k}\right\rangle \bmod p=1$。

6．首先对 k 求和是不允许的，因为对于任意大的 j 和 k，这些项是非 0 的，而且绝对值之和是无穷大。

作为较简单的出错例子，设当 $|j-k|\neq 1$ 时 $a_{jk}=0$，并设 $a_{j(j+1)}=+1$，$a_{(j+1)j}=-1$，则 $\sum_{j\geqslant 0}(\sum_{k\geqslant 0}a_{jk})=\sum_{j\geqslant 0}(\delta_{j0})=+1$，而 $\sum_{k\geqslant 0}(\sum_{j\geqslant 0}a_{jk})=\sum_{k\geqslant 0}(-\delta_{k0})=-1$。

7．是的〔David and Barton, *Combinatorial chance*(1962), 150-154〕。

8．〔(*Combinatorical Analysis*)1(1915), 190〕由取舍原理。例如，$1/(l_1+l_2)!l_3!(l_4+l_5+l_6)!$ 是 $x_1<\cdots<x_{l_1+l_2}$，$x_{l_1+l_2+1}<\cdots<x_{l_1+l_2+l_3}$ 以及 $x_{l_1+l_2+l_3+1}<\cdots<x_{\Sigma l}$ 的概率。

N. G. 德·布鲁因已经给出了用来计算其路段长度分别为 $l_1,\cdots,l_k$ 的 $\{1,\cdots,n\}$ 的排列之个数的一个简单的 $O(n^2)$ 算法，*Nieuw Arehief voor Wiskunde*(3)18(1970), 61-65。

9．在 (23) 中 $p_{km}=q_{km}-q_{k(m+1)}$。因为 $\sum_{k,m}q_{km}z^mx^k=g(x,z)/(1-x)$，我们有 $h(z,x)=\sum h_k(z)x^k=g(x,z)/(1-x)-(g(x,z)-g(x,0))/z(1-x)=$

$(z-1)x/(ze^{(x-1)z}-x)+x/z(1-x)$。于是 $h_1(z)=l^2-(e^z-1)/z$；$h_2(z)=(e^{2z}-ze^z)+e^z-(e^{2z}-1)/z$。

10．设 $M_n=L_1+\cdots+L_n$ 是均值；则 $\Sigma M_n x^n=h'(1,x)$ 这里是对 z 求导数的，比如说这是 $x/(e^{x-1}-x)-x/(1-x)=M(x)$，由残数定理

$$\frac{1}{2\pi i}\oint M(z)z^{-n-1}dz=M_n-2\left(n+\frac{1}{3}\right)+1+\frac{z_1^{-n}}{z_1-1}+\frac{\bar{z}_1^{-n}}{\bar{z}_1-1}$$

这里我们沿着半径为 r 的一个圆周积分，$|z_1|<r<|z_2|$(注意在 $z=1$ 处的二重极点)。其次，这个积分的绝对值 $<\oint|M(z)|r^{-n-1}dz=O(r^{-n})$。在越来越大的圆周上积分就给出收敛的级数 $M_n=2n-\frac{1}{3}+\Sigma_{k\geqslant1}2\Re(1/z_k^n(1-z_k))$

为了确定方差，我们有 $h''(1,x)=-2h'(1,x)+2x(x-1)e^{x-1}/(e^{x-1}-x)^2$。一个类似于已对均值使用过的论证（但这次是对三重极点）证明了 $h''(1,x)$ 的系数渐近于 $4n^2+\frac{4}{3}n-2M_n$ 加些较小的项；由此得方差的渐近公式 $\frac{2}{3}n+2/9$(加上指数较小的项)。

11．$P_{kn}=\Sigma_{t_1\geqslant1,\ldots,t_k\geqslant1}D(t_1,\ldots,t_{k-1},n,1)$，其中 $D(l_1,l_2,\ldots,l_k)$ 是习题 8 的麦克马洪行列式。按它头一行计算这个行列式，我们求出 $P_{kn}=c_0P_{(k-1)n}+c_1P_{(k-2)n}+\cdots+c_{k-2}P_{1n}-E_k(n)$，其中 c_j 和 E_k 定义如下：

$$c_j=(-1)^j\sum_{t_1,\ldots,t_{j-1}\geqslant1}\frac{1}{(t_1+\cdots+t_{j+1})!}=(-1)^j\sum_{m\geqslant0}\binom{m}{j}\frac{1}{(m+1)!}$$

$$=(-1)^j\sum_{r,m\geqslant0}\binom{-1}{j-r}\binom{m+1}{r}\frac{1}{(m+1)!}=-1+e\left(\frac{1}{0!}-\frac{1}{1!}\right.$$

$$\left.+\cdots+(-1)^k\frac{1}{k!}\right)$$

$$E_1(n)=-1/n!+1/(n+1)!;\quad E_2(n)=1/(n+1)!;$$

$$E_k(n)=(-1)^k\sum_{m\geqslant0}\binom{m}{k-3}\frac{1}{(n+2+m)!},\quad k\geqslant3$$

设 $P_{0n}=0$，$C(z)=\Sigma c_jz^j=(e^{1-z}-1)/(1-z)$，并设

$$E(z,x)=\sum_{n,k}E_{k+1}(n)z^nx^k=1-e^z+\frac{(e^{1-x}-1)x^2}{(1-x)^2}$$

$$-\frac{x^2(e^{1-x}-e^z)}{(1-x)(1-x-z)}+\frac{e^z-1-z}{z(1-x)}$$

我们已经导出的递归关系等价于公式 $C(x)H(z,x)=H(z,x)/x+E(z,x)$。因此 $H(z,x)=\Sigma(z,x)x(1-x)/(xe^{1-x}-1)$。展开这个幂级数给出 $H_1(z)=h_1(z)$(见习题 9)；$H_2(z)=eh_1(z)+1-e^z$。

〔注意：头三个路段的生成函数是由克努特导出的，*CACM* 6(1963)，685—688。巴顿(*Barton*) 和马洛斯 (*Mallows*)，在 "*Ann. Math. Statistic*" 36(1965)，249 页上指出了公式 $1-H_{n+1}(z)=(1-H_n(z))/(1-z)-L_nh_1(z)$ 和公式(25)。解决这个问题的另一个方法在习题 23 中说明。由于相邻的路段是不独立的，在这里所解决的问题和习题 9 的

更简单（但大概更有用）的结果之间，没有简单的关系。〕

12．〔*Combinatorical Analysis* 1 (1915), 209-211〕 把多重集合放置到 t 个可区别的盒子中的方式数为

$$N_t=\binom{t+n_1-1}{n_1}\binom{t+n_2-1}{n_2}\cdots\binom{t+n_m-1}{n_m}$$

因为有$\binom{t+n_1-1}{n_1}$种方法来放置 1，等等。如果要求没有空的盒子，则取舍原理告诉我们方式数为

$$M_t=N_t-\binom{t}{1}N_{t-1}+\binom{t}{2}N_{t-2}-\cdots$$

设 P_k 是有 k 个路段的排列数；如果我们在诸路段之间放置 $k-1$ 条垂直的线，并且在 $n-k$ 个剩下的位置上的任何地方放置 $t-k$ 条另外的垂直线，则就得到把多重集合分成 t 个非空的可区别部分的 M_t 个方法之一。因此

$$M_t=P_t+\binom{n-t+1}{1}P_{t-1}+\binom{n-t+2}{2}P_{t-2}+\cdots$$

等置两个 M_t 的值，就可借助于 $N_1, N_2, \cdots$ 逐次地确定 $P_1, P_2, \cdots$。（我们希望见到一个更直接的证明。）

13．$1+\frac{1}{2}13\times 3=20.5$。

14．由福塔对应式，给定的排列对应于

$$\begin{pmatrix}1&1&1&1&2&2&2&2&3&3&3&3&4&4&4&4\\3&1&1&2&3&4&3&2&1&1&3&4&2&2&4&4\end{pmatrix}$$

由（33）式这对应于

$$\begin{pmatrix}1&1&1&1&2&2&2&2&3&3&3&3&4&4&4&4\\2&4&4&3&3&3&1&1&4&4&2&1&2&1&2&3\end{pmatrix}$$

而且这对应于具有 9 个路段的 2 3 4 2 3 4 1 4 2 1 4 3 2 1 3 1。

15．交替路段的个数是使得 $1<j<n$ 且 $a_{j-1}<a_j>a_{j+1}$ 或 $a_{j-1}>a_j<a_{j+1}$ 的 j 的个数。对于固定的 j，这个概率为$\frac{2}{3}$；因此对于 $n\geqslant 2$，平均是 $1+\frac{2}{3}(n-2)$。

16．当新的元素 n 被插入到所有可能位置时，$\{1, 2, \cdots, n-1\}$ 上的每个有 k 个交替路段的排列产生带有 k 个这样路段的 k 个排列，带有 $k+1$ 个这样路段的 2 个排列，以及带有 $k+2$ 个路段的 $n-k-2$ 个排列，因此

$$\left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle=k\left\langle\!\left\langle {n-1 \atop k} \right\rangle\!\right\rangle+2\left\langle\!\left\langle {n-1 \atop k-1} \right\rangle\!\right\rangle+(n-k)\left\langle\!\left\langle {n-1 \atop k\quad 2} \right\rangle\!\right\rangle$$

命$\left\langle\!\left\langle {1 \atop k} \right\rangle\!\right\rangle=\delta_{k0}$，$G_1(z)=1$ 是方便的。于是

$$G_n(z)=\frac{z}{n}\left((1-z^2)G'_{n-1}(z)+(2+(n-2)z)G_{n-1}(z)\right)$$

微分导出 $x_n=G_n'(1)$ 的递归式

$$x_n=\frac{1}{n}(x_{n-1}(n-2)+2n-2)$$

而这对于 $n\geqslant 2$ 有解 $x_n=\frac{2}{3}n-\frac{1}{3}$。另一个微分导致对于 $y_n=G_n''(1)$ 的递归式

$$y_n=\frac{1}{n}\left(y_{n-1}(n-4)+\frac{8}{3}n^2-\frac{26}{3}n+6\right)$$

置 $y_n=\alpha n^2+\beta n+\gamma$ 并对 α，β，γ 求解，得到 $y_n=\frac{4}{9}n^2-\frac{14}{15}n+\frac{11}{90}$，其中 $n\geqslant 4$。

因此 $\mathrm{Var}(g_n)=\frac{1}{90}(16n-29)$，$n\geqslant 4$。

这些均值和方差的公式是 J. 比内米（J. Bienayme）给出的，他未加证明地指出了它们〔*Bull. Soc. Math. de France*) 2 (1874), 153-154; *Comptes Rendus* 81(Acad. Sciences, Paris, 1875) 417-423，也请看 458 页上 Betrand 的注记〕。$\left\langle\!\left\langle {n \atop k} \right\rangle\!\right\rangle$ 的递归关系是 D. 安德烈给出的〔*Comptes Rendus* 97 (Acad. Sciences, Paris, 1883), 1356-1358; *Annales scientifiques de l'Ecole normale supérieure*(3), (Paris, 1884), 121-134〕。安德烈说明对于 $n\geqslant 4$，$g_n(-1)=0$，即具有偶数个交替路段的排列的个数是 $n!/2$。他也证明了均值的公式，并确定了具有极大的交替路段个数的排列的数目（见习题 5.1.4-22）。可以证明

$$G_n(z)=\left(\frac{1+z}{2}\right)^{n-1}(1-w)^{n+1}g_n\left(\frac{1-w}{1+w}\right),\quad w=\sqrt{\frac{1-z}{1+z}},\quad n\geqslant 2$$

其中 $g_n(z)$ 是递增路段的生成函数 (18)〔见 David and Barton, *Combinatorial chance* (London: Griffin, 1962), 157-162〕。

17. $\binom{n+1}{2k-1}$；$\binom{n}{2k-2}$ 个以 0 结尾，$\binom{n}{2k-1}$ 个以 1 结尾。

18. 设给定的序列是 $C_1C_2\cdots C_n$，一个如同我们在习题 5.1.1-7 中已考虑过的反序表。如果在这个序列中有 k 个路段，则在对应的排列中也有 k 个，因此答案是 $\left\langle {n \atop k} \right\rangle$。

19. (a) 由定理 5.1.2B 的对应结果得 $\left\langle {n \atop k+1} \right\rangle$。(b) 有 $(n-k)!$ 种方式把 $n-k$ 个不相拼的车放置到棋盘上；因此答案是 $1/(n-k)!$ 乘 $\sum_{j\geqslant 0}a_{nj}\binom{j}{k}$，其中 $a_{nj}=\left\langle {n \atop j+1} \right\rangle$ 是当 $j=k$ 时的 (a) 部分的解。由习题 2，这导出 $\left\{ {n \atop n-k} \right\}$。

由 E. A. 本德（E. A. Bender）给出的关于这个结果的一个直接证明，把使 $\{1,2,\cdots,n\}$ 分成为 k 个不相交的非空子集的每一分划与 $(n-k)$ 个车的一种排列联系起来：设分划是

$$\{1,2,\cdots,n\}=\{a_{11},a_{12},\cdots,a_{1n_1}\}\cup\cdots\cup\{a_{k1},a_{k2},\cdots,a_{kn_k}\}$$

其中对于 $1\leqslant j\leqslant n_i$，$1\leqslant i\leqslant k$ 有 $a_{ij}<a_{i(j+1)}$。对于 $1\leqslant j\leqslant n_i$，$1\leqslant i\leqslant k$，对应的排列把这些车放置到 $a_{i(j+1)}$ 行 a_{ij} 列中。例如，图 4 中所示图形对应于分划 $\{1,3,8\}\cup\{2\}\cup\{4,6\}\cup\{5\}\cup\{7\}$。

20．阅读的次数是逆排列中路段的个数。头一个路段对应于头一次阅读，等等。

21．它有 $n+1-k$ 个路段并需要 $n+1-j$ 次阅读。

22．〔*J. Combinatorical Theory* 1 (1966)，350～374〕 如果 $rs<n$，则某次阅读将挑出 $t>r$ 个元素，$a_{i_1}=j+1,\cdots,a_{i_t}=j+t$，其中 $i_1<\cdots<i_t$。对于在 $i_k\leqslant m\leqslant i_{k+1}$ 范围内的所有 m，我们不可能有 $a_m>a_{m+1}$，所以这个排列至少在 $t-1$ 个位置上有 $a_m<a_{m+1}$；因此它至多有 $n-t+1$ 个路段。

但是如果 $rs\geqslant n$，则考虑排列

$$(tr)\cdots(2r)(r)\cdots(3)((t-1)r+2)\cdots(r+2)(2)$$
$$((t-1)r+1)\cdots(r+1) \qquad (1)$$

其中 $\lceil n/r\rceil=t\leqslant s$，而且省略了 $>n$ 的元素。它有 $n+1-r$ 个路段，而且它需要 $n+1-r$ 次阅读。通过独立地重新排列 $\{kr+r,\cdots,kr+1\}$ 诸组，有可能把阅读的次数调整成任何所希望的较小的值 $s\geqslant t$。

23．〔*SIAM Review* 3 (1967)，121-122.〕 假设无穷排列由取自于一致分布的独立样品组成。设 $f_k(x)dx$ 是第 k 个最长的以 x 开始的概率，而且设 $g(u,x)dx$ 是当上一个路段以 u 开始时后一个长路段以 x 开始的概率，因此有 $f_1(x)=1$，$f_{k+1}(x)=\int_0^1 f_k(u)\,g(u,x)\,du$。我们有 $g(u,x)=\sum_{m\geqslant 1}g_m(u,x)$，其中 $g_m(u,x)=P_r(u<X_1<\cdots<X_m>X$ 或 $u>X_1>\cdots>X_m<x)=P_r(u<X_1<\cdots<X_m)+P_r(u>X_1>\cdots>X_m)-P_r(u<X_1<\cdots<X_m<x)-P_r(u>X_1>\cdots>X_m>x)=(u^m+(1-u)^m+|u-x|^m)/m!$ 因此 $g(u,x)=e^u+e^{1-u}-1-e^{|u-x|}$。由此，$f_2(x)=2e-1-e^x-e^{1-x}$。可以证明 $f_k(x)$ 趋向于极限值 $\left(2\cos\left(x-\frac{1}{2}\right)-\sin\frac{1}{2}-\cos\frac{1}{2}\right)\Big/\left(3\sin\frac{1}{2}-\cos\frac{1}{2}\right)$。以 x 开始的一个路段的平均长度是 $e^x+e^{1-x}-1$，因此第 k 个最长的路段的长度 LL_k 是 $\int_0^1 f_k(x)(e^x+e^{1-x}-1)dx$；$LL_1=2e-3\approx 2.4365$；$LL_2=3e^2-8e+2\approx 2.4209$。类似的结果见5.4.1节。

24．如同前面一样进行论证，结果是

$$1+\sum_{0\leqslant k<n}2^k(p^2+q^2)^k(p^2+2pq(2^{n-k-1}-1+q^2((2pq)^{n-k-1}-1)/(2pq-1)));$$

求和，并简化之得出

$$2^n(p^2+q^2)^n\left(p(p-q)/(p^2+q^2-pq)-\frac{1}{2}\right)+(2pq)^n pq^3/(p^2+q^2)$$
$$\times(p^2+q^2-pq)+q^2/(p^2+q^2)+2^{n-1}$$

5.1.4 节

1．

1	2	3	8
4	5	7	
6	9		

1	3	5	8
2	4	9	
6	7		

；

$$\begin{pmatrix}1&3&4&5&7&8&9\\5&9&2&4&8&1&7\end{pmatrix}.$$

2．当把 p_i 插入到列 t 中时，设列 $t-1$ 中的元素是 p_j。则 (q_j,p_i) 是在类 $t-1$

中，$q_j<q_i$，而且 $p_j<p_i$，所以由归纳法，存在具有这个性质的足标 $i_1,\cdots,i_t$。反之，如果 $q_j<q_i$ 和 $p_j<p_i$ 而且如果 (q_j, p_j) 在类 $t-1$ 中，则当插入 p_i 时列 $t-1$ 包含一个 $<p_i$ 的元素，所以 (q_i, p_i) 在类 t 中。

3．当插入 p_i 时，这些列是"冲撞序列"（9）。本题中的行 1 和 2 反映了对于行 1 的操作，参考(14)。如果我们撤销其行 2 有∞的项的一些列，则如同在（15）中那样，行 0 和行 2 组成"撞下的阵列"。从行 k 进行到行 $k+1$ 的所述方法，恰是正文中确定类的算法。

4．设共有 t 类；这些类中恰有 k 个有奇数个元素，因为一个类中的元素的形式为

$$(p_{i_k}, p_{i_1}),\ (p_{i_{k-1}}, p_{i_2}),\cdots,\ (p_{i_1}, p_{i_k})$$

（见（18）和(22)）。"撞下的两行阵列恰有 $t-k$ 个不动点，这是由于构造它的方法所致；因此，由归纳法，这个图表减去它的头一行后，有 $t-k$ 个奇数长度的列。所以在头一行中的 t 个元素导致整个图表中 k 个奇数长度的列。

5．（a）对图表的大小用归纳法；利用一个例子分析，首先考虑对于行 1 的影响，然后考虑对于由行 1 撞下来的元素序列的影响。（b）可允许的交换可以模拟算法 I 的操作，其中图表表示成在这个算法之前和之后的一个规范排列。例如，通过一个可允许的交换序列（参考(4)和(5)），我们可以把

17 11 4 13 14 2 6 10 15 1 3 5 9 12 16 8

变换成为

17 11 13 4 10 14 2 6 9 15 1 3 5 8 12 16

6．可允许的交换是左右对称的，而且当颠倒插入次序时，P 的规范排列显然地成为 P^T。

7．列的数目，即行 1 的长度，是类的数目（习题 2）。行数是 P^T 的列数，所以习题 6（或定理 D）完成了这个证明。

8．对于多于 n^2 个元素，对应的 P 图表必须或者多于 n 行或者多于 n 列。但有 $n\times n$ 的图表〔这一结果最初是在"*Compositio Math*" 2(1935) 463-470 中证明的〕。

9．这样的排列同形如（$n, n, \cdots, n$）的图表有一一对应关系，所以由(34)，答案为

$$\left(\frac{n^2!\Delta(2n-1, 2n-2, \cdots, n)}{(2n-1)!(2n-2)!\cdots n!}\right)^2=\left(\frac{n^2!}{(2n-1)(2n-2)^2\cdots n^n(n-1)^{n-1}\cdots 1^1}\right)^2$$

对于这个问题，存在这样一个简单公式，真是惊人。我们也可以计算没有长于 m 的递增子序列和没有长于 n 的递减子序列的 $\{1, 2, \cdots, mn\}$ 的排列数。

10．我们归纳地证明在步骤 S3 中，$P_{(r-1)s}$ 和 $P_{r(s-1)}$ 都小于 $P_{(r+1)s}$ 和 $P_{r(s+1)}$。

11．我们当然也需要知道原来曾是 P_{11} 的元素。然后有可能利用颇类似于算法 S 的一个算法，来进行恢复。

12．$$\binom{n_1+1}{2}+\binom{n_2+2}{2}+\cdots+\binom{n_m+m}{2}-\binom{m+1}{3}$$

极小值是习题 1.2.4-41 的序列 1，2，2，3，3，3，4，4，4，4 …的头 n 项之和；这个和近似于 $\sqrt{8/9}\,n^{3/2}$（因为按照习题 28 中引用的表格，n 个元素的图表的大多数都相当接近这个下界，平均次数确实几乎是 $O(n^{3/2})$，但这尚未证明）。

13．假设排列的元素是$\{1, 2, \cdots, n\}$，因而$a_i=1$；并且假设$a_j=2$。情况1，$j<i$。则1撞下2，所以对应于$a_1\cdots a_{i-1}a_{i+1}\cdots a_n$的图表的行1是$P^s$的行1；而且"撞下"的排列除了它的最小元素2外，是以前被撞下的排列，所以我们可以对n用归纳法。情况2，$j>i$。对P^T应用情况1，并利用习题6及$(P^T)^s=(P^s)^T$这一事实。

15．如同在(37)中那样，例中的排列对应于图表

1	2	5	9	11
3	6	7		
4	8	10		

因此此数为$f(l, m, n)=(l+m+n)!(l-m+1)(l-n+2)(m-n+1)/(l+2)!(m+1)!(n)!$，当然，假定$l\geqslant m\geqslant n$。

16．由定理H，80080。

17．由于g对诸x是反对称的，当$x_i=x_j$时它为0，所以对于所有的$i<j$它可以被x_i-x_j所整除。因此$g(x_1, \cdots, x_n; y)=h(x_1, \cdots x_n; y)\Delta(x_1, \cdots, x_n)$，这里$h$必须是总次数为1的对$x_1, \cdots, x_n, y$齐次的，而且对于$x_1, \cdots, x_n$是对称的；所以对于仅仅依赖于$n$的某个$a$，$b$，$h(x_1, \cdots, x_n; y)=a(x_1+\cdots+x_n)+by$。我们通过置$y=0$可以计算$a$；通过对$y$求偏微商，然后置$y=0$可以计算$b$。我们有

$$\frac{\partial}{\partial y}\Delta(x_1, \cdots, x_i+y, \cdots, x_n)|_{y=0}=\frac{\partial}{\partial x_i}\Delta(x_1, \cdots, x_n)=\Delta(x_1, \cdots, x_n)\sum_{j\neq i}\frac{1}{x_i-x_j}$$

最后

$$\sum_i\sum_{j\neq i}(x_i/(x_i-x_j))=\sum_i\sum_{j<i}(x_i/(x_i-x_j)+x_j/(x_j-x_i))=\binom{n}{2}$$

18．它必须是$\Delta(x_1, \cdots, x_n)\cdot(b_0+b_1y+\cdots+b_my^m)$，其中每个$b_k$是诸$x$的次数为$m-k$的齐次对称多项式。我们有

$$\frac{\partial^k}{k!\partial y^k}\Delta(x_1, \cdots, x_i+y, \cdots, x_n)|_{y=0}=\Delta(x_1, \cdots, x_n)\Sigma 1/\Pi_l(x_i-x_{j_l})$$

对于不同下标$j_1, \cdots, j_k\neq i$的所有$\binom{n-1}{k}$种选择进行求和。现在在$b_k=\Sigma x_i^m/\Pi_l(x_i-x_j)$中，我们可以组合那些有一个给定的下标集合$\{i, j_1, \cdots, j_k\}$的$k+1$项的组；例如，当$k=2$时，我们组合形如$a^m/(a-b)(a-c)+b^m/(b-a)(b-c)+c^m/(c-a)(c-b)$的三项的集合。每个这样的组的和如同在习题1.2.3-33中那样计算；它是在$1/(1-x_iz)(1-x_{j_1}z)\cdots(1-x_{j_k}z)$中$z^{m-k}$的系数。我们因此求得

$$b_k=\sum_j\binom{n-j}{k+1-j}\Sigma s(p_1, \cdots, p_j)$$

其中$s(p_1, \cdots, p_j)$是对于不同的下标$i_1, \cdots, i_j\in\{1, \cdots, n\}$，由形如$x_{i_1}^{p_1}\cdots x_{i_j}^{p_j}$的所有不同项所组成的单项对称函数；而第二个和是对于把$m-k$分成恰好j个部分的所有分划进行的，即$p_1\geqslant\cdots\geqslant p_j\geqslant 1$，$p_1+\cdots+p_j=m-k$（这个结果是同E·A·本德一起得到的）。

当$m=2$时答案是$\left(s(2)+(n-1)s(1)y+\binom{n}{3}y^2\right)\Delta(x_1, \cdots, x_n)$；对于$m=3$我们得到$\left(s(3)+((n-1)s(2)+s(1, 1)y+\binom{n-1}{2}s(1)y^2+\binom{m}{4}y^3\right)\Delta(x_1, \cdots, x_n)$；等等。

另一个表达式，作为

$$\left(\binom{n}{k+1}z^k-\binom{n-1}{k+1}a_1z^{k+1}+\binom{n-2}{k+1}a_2z^{k+2}-\cdots\right)\Big/(1-a_1z+a_2z^2-\cdots)$$

中 z^m 的系数给出 b_k，其中 $a_l=\Sigma_{1\leqslant i_1<\cdots<i_l\leqslant n}x_{i_1}\cdots x_{i_l}$ 是一个"初等对称函数"。乘以 y^k 并对 k 进行求和即给出答案，它是

$$\frac{1}{y^z}\left(\frac{(1+z(y-x_1))\cdots(1+z(y-x_n))}{(1-zx_1)\cdots(1-zx_n)}-1\right)\Delta(x_1,\cdots,x_n)$$

中 z^m 的系数。

19．设转置图表的形状是 $(n'_1,n'_2,\cdots,n'_r)$；答案是 $\frac{1}{2}f(n_1,n_2,\cdots,n_m)$ 乘 $(\Sigma(n_i^2)-\Sigma(n_j'^2))/n(n+1)+1$，其中 $n=\Sigma n_i=\Sigma n'_j$。（利用关系式 $\Sigma in_i=n+\frac{1}{2}\Sigma n_j'^2$，这个公式可以以不大对称的形式来加以表达。）

注：W. 菲特（W. Feit）研究了作为两个图表形状之差 $(n_1,\cdots,n_m)/(l_1,\cdots,l_m)$ 的阵列，得到了把整数（1，2，…，n）放置到这种阵列中的方式个数的一般公式，其中 $0\leqslant l_i\leqslant n_i$，且 $n=\Sigma n_i-\Sigma l_i$〔*Proc. Amer. Math. Soc.* 4 (1953)，740-744〕。这个数是 $n!\det(1/((n_j-j)-(l_j-i))!)$。

20．在定理H之后的讨论中靠不住的论证，实际上对于这一情况是正确的（对应的概率都是独立的）。

21．〔*Michigan Math. J.*) 1 (1952), 81-88〕设 $g(n_1,\cdots,n_m)=(n_1+\cdots+n_m)!\Delta(n_1,\cdots,n_m)/n_1!\cdots n_m!\sigma(n_1,\cdots,n_m)$，其中 $\sigma(x_1,\cdots,x_n)=\Pi_{1\leqslant i<j\leqslant m}(x_i+x_j)$。为证明 $g(n_1,\cdots,n_m)$ 是填充移位图表的方式个数，我们必须证明 $g(n_1,\cdots,n_m)=g(n_1-1,\cdots,n_m)+\cdots+g(n_1,\cdots,n_{m-1})$。对应于习题 17 的这个恒等式是 $x_1\Delta(x_1+y,\cdots,x_n)/\sigma(x_1+y,\cdots,x_n)+\cdots+x_n\Delta(x_1,\cdots,x_n+y)/\sigma(x_1,\cdots,x_n+y)=(x_1+\cdots+x_n)\Delta(x_1,\cdots,x_n)/\sigma(x_1,\cdots,x_n)$，同 y 无关；因为如果象在习题 17 中那样计算微商，则我们求得 $2x_ix_j/(x_j^2-x_i^2)+2x_jx_i/(x_i^2-x_j^2)=0$。

22．〔*Journ. de Math.* (3) 7 (1881), 167-184.〕（这是对于长度为 2 的所有路段习题 5.1.3-18 的一种特殊情况，也许最后一个路段除外。）当 $n>0$ 时，元素 n 必出现在一行的最右边的位置之一上；一旦已经把它放到第 k 行最右边的框中，我们便有 $\binom{n-1}{2k-1}A_{2k-1}A_{n-2k}$ 种方式来完成这一工作。设

$$h(z)=\sum_{k\geqslant 1}A_{2n-1}z^{2n-1}/(2n-1)!=\frac{1}{2}(g(z)-g(-z))$$

则

$$h(z)g(z)=\sum_{k,n\geqslant 1}\binom{n}{2k-1}A_{2k-1}A_{n-2k+1}z^n/n!=\left(\sum_{n\geqslant 1}A_{n+1}z^n/n!\right)-1$$
$$=g'(z)-1$$

以 $-z$ 代替 z 并且相加，得到 $h(z)^2=h'(z)-1$；因此，$h(z)=\tan(z)$。置 $k(z)=g(z)-h(z)$，我们有 $h(z)k(z)=k'(z)$；因此 $k(z)=\sec(z)$ 和 $g(z)=$

$\sec(z)+\tan(z)=\tan(1/2\,z+1/4\,\pi)$。因此系数 A_{2n} 是欧拉数 E_{2n}；系数 A_{2n+1} 是正切数 T_{2n+1}。这些数的表出现于 *Math. Comp.* 21(1967)，663-688 中。

23. 假设 $m=N$，必要时把一些 0 加到这个图形上；如果 $m>N$ 和 $n_m>0$，则方式数显然为 0。当 $m=N$ 时，这个答案是

$$\det\begin{pmatrix} \binom{n_1+m-1}{m-1} & \binom{n_2+m-2}{m-1} & \cdots & \binom{n_m}{m-1} \\ \vdots & & & \vdots \\ \binom{n_1+m-1}{0} & \binom{n_2+m-1}{0} & \cdots & \binom{n_m}{0} \end{pmatrix}$$

证明：我们可以假定 $n_m=0$，因为如果 $n_m>0$ 则这个阵列的头 n_m 列的第 i 行必然填的是 i，而且我们可以考虑剩下的图形 $(n_1-n_m,\ \cdots,\ n_m-n_m)$。通过对 m 用归纳法，方式数转为

$$\sum_{\substack{n_2\leqslant k_1\leqslant n_1\\ \vdots \\ n_m\leqslant k_{m-1}\leqslant n_{m-1}}} \det\begin{pmatrix} \binom{k_1+m-2}{m-2} & \binom{k_2+m-3}{m-2} & \cdots & \binom{k_{m-1}}{m-2} \\ \vdots & & & \vdots \\ \binom{k_1+m-2}{0} & \binom{k_2+m-3}{0} & \cdots & \binom{k_{m-1}}{0} \end{pmatrix}$$

其中 n_j-k_j 表示在行 j 中诸 m 的个数。对于每个 k_j 的求和可以独立地进行，由此得

$$\det\begin{pmatrix} \binom{n_1+m-1}{m-1}-\binom{n_2+m-2}{m-1} & \binom{n_2+m-2}{m-1}-\binom{n_3+m-3}{n-1} & \cdots & \binom{n_{m-1}+1}{m-1}-\binom{n_m}{m-1} \\ & & & \vdots \\ & & & \vdots \\ \binom{n_1+m-1}{1}-\binom{n_2+m-2}{1} & \binom{n_2+m-2}{1}-\binom{n_3+m-3}{1} & \cdots & \binom{n_{m-1}+1}{1}-\binom{n_m}{1} \end{pmatrix}$$

即是所求的答案，因为 $n_m=0$。通过行运算，这个答案可以转化为一个范特蒙特行列式，同时给出公式 $\Delta(n_1+m-1,\ n_2+m-2,\ \cdots,\ n_m)/(m-1)!(m-2)!\cdots 0!$〔这一习题的答案，同群论中一个等价的问题相联系，它出现在 D. E. Littlewood 的 Theory of Group Characters (Oxford, 1940)，189 中。〕

25. 一般地说，如果 u_{nk} 是 $\{1,2,\cdots,n\}$ 上没有长度 $>k$ 的循环的排列个数，则 $\sum u_{nk}z^n/n!=\exp(z+z^2/2+\cdots+z^k/k)$；这可以通过乘 $\exp(z)\times\cdots\times\exp(z^k/k)$ 来证明，由此得到

$$\sum_n z^n\left(\sum_{j_1+2j_2+\cdots+kj_n=n} 1/1^{j_1}j_1!2^{j_2}j_2!\cdots\right)$$

参考习题 1.3.3-21。类似地 $\exp(\sum_{s\in S} z^s/s)$ 是相应的诸排列的母函数，这些排列中循环的长度全是某给定集合 S 的元素。

26. 由伽玛函数积分（习题 1.2.5-20，$t=2x^2/\sqrt{n}$）可知，从 0 到 ∞ 的积分为 $n^{(t+1)/4}\Gamma((t+1)/2)/2^{(t+3)/2}$。所以当 t 为奇数时从 $-\infty$ 到 $+\infty$ 的积分是 0，否则为 $n^{(t+1)/4}\sqrt{\pi}\,t!/2^{(3t+1)/2}(t/2)!$。

27. (a) 如果 $r_i<r_{i+1}$，而且 $c_i<c_{i+1}$，则 $i<Q_{r_i c_{i+1}}<i+1$ 是不可能的。如果 $r_i>r_{i+1}$ 和 $c_i<c_{i+1}$ 则显然 $r_i\neq r_{i+1}$，所以得到一个类似的矛盾。(b) 通过对 $a_1\cdots a_i$ 的图表中的行数用归纳法，证明 $a_i<a_{i+1}$ 意味着 $c_i<c_{i+1}$，且 $a_i>a_{i+1}$ 意味着 $c_i\geqslant c_{i+1}$（考虑行 1 及"撞下的"序列）。(c) 这从定理 D(c) 得出。

28. 这一结果是 A. M. 维尔希克（A. M. Вершик）和 C. B. 克洛夫（C. B. Кэров）给出的见"Дакл. Акаб. Наук. ссср" 233(1977)，1024-1028。也可见 B. F. 洛根（B. F. Logan）和 L. A. 谢波（L. A. Shepp）"Advances in Math" 26(1977)，206-222。

29. $\binom{n}{p}\Big/p!$ 是长度为 p 的递增子序列的平均个数。由习题 8 和 29，最大的递增子序列有长度 $\geqslant e\sqrt{n}$ 或 $\leqslant\sqrt{n}/e$ 的概率为 $O(1/\sqrt{n})$；这一结果是同 J. D. 狄克森（J. D. Dixon）一起得到的。

30. (Discrete Math, 2 (1972), 73-94)。)

31. $x_n=a_{\lfloor n/2\rfloor}$，其中 $a_0=1$，$a_1=2$，$a_n=2a_{n-1}+(2n-2)a_{n-2}$；$\sum a_n z^n/n!=\exp(2z+z^2)=(\sum t_n z^n/n!)^2$；$x_n\sim\exp\left(\frac{1}{4}n\ln n-\frac{1}{4}n+\sqrt{n}-\frac{1}{2}-\frac{1}{2}\ln 2\right)$。

5.2 节

1. 是，i 和 j 可以以任意顺序跑遍 $1\leqslant j<i\leqslant N$ 的值的集合。这表明当记录被读入时，可以同时进行计数。

2. 在这一章开始时给出的定义的意义下，排序是"稳定的"；因为这个算法实质上是按不同键对 $(K_1, 1)$，$(K_2, 2)$，…，(K_N, N) 按字典顺序进行排序的（如果我们把每个键想像作通过它在文件中的位置向右边扩充，则不出现相等的键，而且是"稳定"的）。

3. 它能排序。但不是一种"稳定"的方式；如果 $K_j=K_i$ 且 $j<i$，则 R_j 将在排好了的顺序中出现在 R_i 之后，这个变化也将使程序 C 更慢地进行。

4.	ENT1	N	1
	LD2	COUNT, 1	N
	LDA	INPUT, 1	N
	STA	OUTPUT+1, 2	N
	DEC1	1	N
	J1P	*−4	N

5. 运行时间改变 $N-1-A+B$ 单位，而且这在绝大多数情况下是个改进。

6. $u=0$，$v=9$。

在 D1 之后，COUNT = 0 0 0 0 0 0 0 0 0 0

在 D2 之后，COUNT = 2 2 1 0 1 3 3 2 1 1

在D4之后，COUNT＝2 4 5 5 6 9 12 14 15 16

在D5期间，COUNT＝2 3 5 5 5 8 9 12 15 16 $j=8$

OUTPUT＝— — — 1G — 4A — — 5L 6A 6T 6I 7O 7N — —

在D5之后，OUTPUT＝0C 0O 1N 1G 2R 4A 5T 5U 5L 6A 6T 6I 7O 7N 8S 9

7．是（注意在步骤D6中COUNT[K_j]被减值，而且j减值）。

8．它能排序，但不是以一种“稳定”的方式（参考习题7）。

9．设 $M=v-u$；Loc(R_j)≡INPUT＋j；Loc(COUNT[j])≡COUNT＋j；Loc(S_j)≡OUTPUT＋j；rI1＝i；rI2＝j；rI3≡$i-v$，K_j。假定|u|，|v|能在两个字节中容下。

```
M       EQU     V - U
KEY     EQU     0:2                     (在字节3:5中的随从信息)
1H      ENN3    M          1            D1. 清诸COUNT
        STZ     COUNT - V,3   M + 1     COUNT[v - k]← 0
        INC3    1          M + 1
        J3NP    * - 2      M + 1        u ≤ i ≤ v
2H      ENT2    N          1            D2. 对 j 进行循环
3H      LD3     INPUT,2(KEY)  N         D3. COUNT[K_j]增值
        LDA     COUNT,3    N
        INCA    1          N
        STA     COUNT,3    N
        DEC2    1          N
        J2P     3B         N            N ≥ j > 0
        ENN3    M - 1      1            D4. 累计
        LDA     COUNT + U  1            rA← COUNT[i - 1]
4H      ADD     COUNT + V,3   M         COUNT[i - 1] + COUNT[i]
        STA     COUNT + V,3   M         →COUNT[i]
        INC3    1          M
        J3NP    4B         M            u ≤ i ≤ v
5H      ENT2    N          1            D5. 对 j 进行循环
6H      LD3     INPUT,2(KEY)  N         D6. 输出R_j
        LD1     COUNT,3    N            i ← COUNT[K_j]
        LDA     INPUT,2    N            rA→R_j
        STA     OUTPUT,1   N            S_i←rA
        DEC1    1          N
        ST1     COUNT,3    N            COUNT[K_j]← i - 1
        DEC2    1          N
        J2P     6B         N            N ≥ j > 0
```

运行时间是（$10M+22N+10$）u。

10．为了避免使用N个额外“标志”位[见1.3.3节和Cybernetics1(1965)，95]并

保持运行时间基本上同N成比例，我们可以使用下列以排列的循环结构为基础的算法：

P1.〔对 i 进行循环〕 对于 $1 \leqslant i \leqslant N$做步骤 P2；然后结束这个算法。

P2.〔$p(i)=i$ 吗?〕 如果 $p(i) \neq i$，则进行步骤 P3 到 P5。

P3.〔开始循环〕 置 $t \leftarrow R_i$，$j \leftarrow i$。

P4.〔修正 R_j〕 置 $k \leftarrow p(j)$，$R_j \leftarrow R_k$，$p(j) \leftarrow j$，$j \leftarrow k$。

P5.〔结束循环〕 置 $R_j \leftarrow t$，$p(j) \leftarrow j$。 这个算法改变 $p(i)$，因为排序的应用允许我们假定 $p(i)$ 存储在内存中。另一方面，存在象矩阵转置这样一些应用，其中 $p(i)$ 是有待计算（为了节省内存空间而不造表）的 i 的函数。在这样一种情况下，我们可以使用下列的方法，对于 $1 \leqslant i \leqslant N$实施步骤 B1 到 B3：

B1. 置 $k \leftarrow p(i)$。

B2. 如果 $k > i$，则置 $k \leftarrow p(k)$ 并重复这个步骤。

B3. 如果 $k < i$，则什么也不做；但如果 $k = i$（这意味着 i 是在它的循环中最小的）则我们把包含 i 的循环排列如下：置 $t \leftarrow R_i$，然后当 $p(k) \neq i$ 时，重复地置 $R_k \leftarrow R_{p(k)}$，以及 $k \leftarrow p(k)$；最后 $R_k \leftarrow t$

这个算法类似于 J. 布思罗伊德（J. Boothroyd）〔*Comp. J.* 10(1967)，310〕的过程。但只要较少的数据移动；I. D. G 麦克劳德（I. D. G. Macleod）〔*Australian Comp. J.* 2 (1970)，16-19.〕已经提出某些改进。对于随机排列，习题 1.3.3-14 中的分析表明，步骤 B2 平均要实施 $(N+1)H_N - N$步。参考 D. E. 克努特〔*Proc IFIP Congress* 1971. 1，19-27〕。可以设计类似的算法来用 $(R_1, \cdots, R_N)$ 代替 $(R_{p(1)}, \cdots, R_{p(N)})$。例如，如果习题 4 中的重新排列是以 OUTPUT=INPUT 来加以完成的。

11. 命 rI1≡ i；rI2≡ j；rI3≡ k；rX≡ t。

1H	ENT1	N	1	*P*1. 对 i 进行循环
2H	CMP1	P,1	N	*P*2. $P(i)=i$ 吗?
	JE	8F	N	如果 $P(i)=i$ 则转移
3H	LDX	INPUT,1	$A-B$	*P*3. 开始循环，$t \leftarrow R_i$
	ENT2	0,1	$A-B$	$j \leftarrow i$
4H	LD3	P,2	$N-A$	*P*4. 修正R_j。$k \leftarrow p(j)$
	LDA	INPUT,3	$N-A$	
	STA	INPUT,2	$N-A$	$R_j \leftarrow R_k$
	ST2	P,2	$N-A$	$P(j) \leftarrow j$
	ENT2	0,3	$N-A$	$j \leftarrow k$
	CMP1	P,2	$N-A$	
	JNE	4B	$N-A$	如果 $P(j) \neq i$ 则重复
5H	STX	INPUT,2	$A-B$	*P*5. 结束循环 $R_j \leftarrow t$
	ST2	P,2	$A-B$	$P(j) \leftarrow j$
8H	DEC1	1	N	
	J1P	2B	N	$N \geqslant i \geqslant 1$

运行时间是 $(17N-5A-7B+1)u$，这里A是排列 $p(1)\cdots p(N)$ 中的循环个数，且B是不动点（1—循环）的个数。由等式 1.3.3-21 和 28，对于 $N \geqslant 2$，$A=(\min 1,\ \text{ave}\ H_N,\ \max N,\ \text{dev}\sqrt{H_N - H_N^{(2)}})$；$B=(\min 0,\ \text{ave}\ 1,\ \max N,\ \text{dev}\ 1)$。

12. 下列"直接"方法是由M·D·麦克拉伦给出的(为方便起见假定对于$1 \leqslant P \leqslant N$，$0 \leqslant LINK(P) \leqslant N$，这里$\Lambda \equiv 0$)。

M1.〔初始化〕 置$P \leftarrow HEAD$，$k \leftarrow 1$。

M2.〔完成了?〕 如果$P = \Lambda$（或者等价地，如果$k = N + 1$），算法结束。

M3.〔确保$P \geqslant k$〕 如果$P < k$，置$P \leftarrow LINK(P)$，并重复这一步骤。

M4.〔交换〕 交换R_k和$R[P]$（假定LINK（k）和LINK(P)在这个过程中也交换)，然后置$Q \leftarrow LINK(k)$，$LINK(k) \leftarrow P$，$P \leftarrow Q$，$k \leftarrow k + 1$，并返回步骤M2。

麦克拉伦方法有效的一个证明，可以以下列性质的归纳性验证为基础。这个性质在步骤M2开始时成立：在序列P，LINK(P)，LINK(LINK(P))，…，Λ中$\geqslant k$的项是$a_1, a_2, \cdots, a_{N+1-k}$，这里$R_1 \leqslant \cdots \leqslant R_{k-1} \leqslant R_{a_1} \leqslant \cdots \leqslant R_{a_{N+1-k}}$是这些记录所应有的最后顺序。而且对于$1 \leqslant j < k$有$LINK(j) \geqslant j$，使得$LINK(j) = \Lambda$意味着$j \geqslant k$。

分析上述算法是十分有趣的；它的突出性质之一是它可反向运行，即从LINK(1)，…，LINK(N)最后的值重新构造链接的原来集合。满足$j \leqslant LINK(j) \leqslant N$的$N!$种可能的输出配置的每一种，恰巧对应于$N!$种可能的输入配置之一。如果$A$是步骤M3中$P \leftarrow LINK(P)$的次数，则$N - A$是在这算法结束处使得$LINK(j) = j$的$j$的个数；这种情况当且仅当$j$是它循环中最大者时才出现；因此$N - A$是排列中的循环个数，而且$A = (\min 0,\ \text{ave}\, N - H_N,\ \max N - 1)$。

13. **D5′**. 置$r \leftarrow N$。

D6′. 如果$r = 0$，则停止。否则如果$COUNT[K_r] < r$，则置$r \leftarrow r - 1$并重复这一步骤；如果$COUNT[K_r] = r$，则$COUNT[K_r]$和r都减1，并重复这一步骤。否则置$R \leftarrow R_r$，$j \leftarrow COUNT[K_r]$，$COUNT[K_r] \leftarrow j - 1$。

D7′. 置$S \leftarrow R_j$，$k \leftarrow COUNT[K_j]$，$COUNT[K_j] \leftarrow k - 1$，$R_j \leftarrow R$，$R \leftarrow S$，$j \leftarrow k$。然后如果$j \neq r$，则重复这一步骤；如果$j = r$，则置$R_j \leftarrow R$，$r \leftarrow r - 1$，并转回到D′6。

为了证明这是正确的，注意在步骤D′6开始处，所有满足$j > r$的不在它们最后的安放位置处的记录R_j都必须左移；当$r = 0$时，不可能有任何这样的记录，因为某个东西必须右移。这个算法是漂亮的，但对于相等的键不"稳定"；它同定理5.1.2B中的福塔构造密切相关。

5.2.1 节

1. 是，相等的元素决不彼此交错地移动。

2. 是，但当出现相等的元素时，运行时间将是较慢的，而排序将不是稳定的。

3. 下列8行被猜测为最短的MIX排序程序，尽管就速度来说，它是不值得推荐的。我们假定数出现在位置1，…，N上（即，INPUT EQU 0；否则需要多一行代码)。

```
2H      LDA     0,1     A
        CMPA    1,1     B
        JLE     1F      B
```

```
        MOVE    1,1     A
        STA     0,1     A
START   ENT1    N       A+1
1H      DEC1    1       B+1
        J1P     2B      B+1
```

注：为了估计这个程序的运行时间，注意A是反序的个数，B是反序表的相当简单的函数，而且（假定在随机顺序下的不同输入）它的生成函数是

$$z^{N-1}(1+z)(1+z^2+z^{2+1})$$

$$(1+z^3+z^{3+2}+z^{3+2+1})\cdots\left(1+z^{N-1}+z^{2N-3}+\cdots+z^{\binom{N}{2}}\right)\Big/N!$$

B的平均值是$N-1+\sum_{1\leqslant k\leqslant N}(k-1)(2k-1)/6=(N-1)(4N^2+N+36)/36$；因此这个程序的平均运行时间大约是$\frac{7}{9}N^3u$。

4．在习题5.1.1-7的意义下，考虑给定的输入排列的反序表$B_1\cdots B_N$。A比等于$j-1$的B_j的个数少1，而B是诸B_j之和。因此当输入的排列是$N\ \cdots 2\ 1$时，$B-A$和B都取极大值；当输入$1\ 2\cdots\ N$时，两者取极小值，因此当$A=0$，$B=0$时可达到时间的下限，即$(10N-9)u$；当$A=N-1$，$B=\binom{N}{2}$时出现极大值，这就是$(4.5N^2+2.5N-6)u$。

5．生成函数是$9B-3A$的生成函数乘上z^{10N-9}。通过象在上题中那样考虑反序表，并回想起反序表中各项是彼此独立的，所求的生成函数是$z^{10N-9}\Pi_{1\leqslant j\leqslant N}((1+z^9+\cdots+z^{9j-18}+z^{3j-12})/j)$。方差成为$2.25N^3+3.375N^2-32.625N+36H_N-9H_N^{(2)}$。

6．把输入区域当作一个循环表，而且位置N同位置1相邻。根据上一次插入的元素是落到已排序元素中心的右边还是左边，而从当前的未排序元素段的左边或右边来取新的待插入的元素。过后，通常将需要“转动”这个区域，对于某个固定的k，把每个记录顺着这个循环移动k个位置；这可在$\gcd(N,\ k)$次扫描中完成(对于$1\leqslant i\leqslant\gcd(N,k)$循环地排列$R_i$，$R_{i+k}$，…，$R_{N+i-k}$；参考W. Fletcher and R. Silver，*CACM* 9 (1966)，326)。

7．$|a_j-j|$的平均值是

$$\frac{1}{n}(|1-j|+|2-j|+\cdots+|N-j|)=\frac{1}{n}\left(\binom{j}{2}+\binom{n-j+1}{2}\right)$$

对j求和给出

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

8．否；例如，考虑键2 1 1 1 1 1 1 1 1 1 1。

9．对于表3，$A=3+0+2+1=6$，$B=3+1+4+21=29$；在表4中，$A=4+2+2+0=8$，$B=4+3+8+10=25$；因此程序D的运行时间分别为$786u$和$734u$。尽管移动的次数已经从41减少到25，但运行的时间不能同程序S相比，因为当$N=16$时，有四次扫描的簿记时间是浪费掉的。当对16个项目排序时用两次扫描效果更

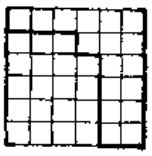
好；一个两次扫描的程序D大约在$N=13$时开始胜过程序S，尽管在短时间内它们是几乎等同的（而对这样小的N，程序长度是相当重要的）。

10．把“INC1 INPUT；ST1 3F(0:2)”插入到行07和行08之间，并把行10～17改变成为：

3H	CMPA	INPUT + N - H, 1	$NT-S$
	JGE	8F	$NT-S$
4H	ENT2	N - H, 1	$NT-S-C$
5H	LDX	INPUT, 2	B
6H	STX	INPUT + H, 2	B
	DEC2	0, 4	B
	J2NP	7F	B
	CMPA	INPUT, 2	$B-A$
	JL	5B	$B-A$
7H	STA	INPUT + H, 2	$NT-S-C$

这纯增了4条指令，却节省了$3(C-T)$个时间单位，这里C是$K_j \geqslant K_{j-h}$的次数。在表3和表4中，节省的时间分别近似于87和88；经验表明，$C/(NT-S)$的值当$h_{s+1}/h_s \approx 2$时大约是0.4，而当$h_{s+1}/h_s \approx 3$时大约是0.3，所以这项改进是值得的（另一方面，对于程序S作类似的改变是不理想的，因为在该情况下的节省仅仅同$\log_2 N$成比例，除非已知输入是相当好地排了序的）。

11．



12．把↓→变成为⇉总是使反序数改变±1，依赖于这个变动是在对角线上面还是下面而异。

13．把权$|i-j|$放到从$(i, j-1)$到(i, j)的线段上。

14．(a) 在A_{2n}的和式中交换i和j，并把这两个和式加起来。(b) 取这个结果的一半，我们看到

$$A_{2n}=\sum_{0 \leqslant i \leqslant j}(j-i)\binom{i+j}{i}\binom{2n-i-j}{n-j}$$

因此两次应用所述恒等式即得$\sum A_{2n}z^n = z/(1-4z)^2$。

以上证明是由伦纳德·卡利兹向作者提出的。另一个证明可以根据水平和垂直权之间的相互作用进行（参考习题13）；利用习题5.2.2-16的答案中的恒等式，并令$f(k)=k$，可得另一个证明；但是却未发现有关公式$A_n=\lfloor n/2 \rfloor 2^{n-2}$的简单的组合推导。

15．对于$n>0$

$$\bar{g}_n(z)=z^{n-1}g_{n-1}(z);\quad \bar{h}_n(z)=\bar{g}_n(z)+z^{-n}\bar{g}_n(z)$$

$$g_n(z)=\sum_{1 \leqslant k \leqslant n}\bar{g}_k(z)g_{n-k}(z);\quad h_n(z)=\sum_{1 \leqslant k \leqslant n}\bar{h}_k(z)h_{n-k}(z)$$

命$G(w, z)=\sum_n g_n(z)w^n$，我们发现$wzG(w, z)G(wz, z)=G(w, z)-1$。从这个

表示我们可以导出，如果 $t=\sqrt{1-4w}=1-2w-2w^2-4w^3-\cdots$，则我们有 $G(w,1)=(1-t)/(2w)$；$G_t(w,1)=1/(wt)-(1-t)/(2w^2)$；$G'(w,1)=1/(2t^2)-1/(2t)$；$G_w(w,1)=2/(wt^5)-2/(w^2t)+(1-t)/w^3$；$G_1'(w,1)=2/t^4-1/t^3$；且 $G''(w,1)=1/t^3-(1-2w)/t^4+10w^2/t^5$。此处下边的撇号表示相对于头一个参数的微商，而上边的撇号表示相对于第二参数的微商。类似地，由公式

$$w(zG(wz,z)+G(w,z))H(w,z)=H(w,z)-1$$

我们导出

$$H'(w,1)=w/t^4,\quad H''(w,1)=-w/t^3-w/t^4+2w/t^5+(2w^2+20w^3)/t^7$$

这里概述的公式的处理是用手算的，但它本应由计算机来算。原则上，所有分布阶段都可以以这样的方式得到。

生成函数 $g_n(z)$ 也表示了 Σ_n 内部路径长度，这里求和对所有 n 个节点的树进行；参见习题 2.3.4.5-5。

说明这样一点是有趣的，即 $G(w,z)=F(-wz,z)/F(w,z)$ 其中 $F(z,q)=\sum_{n\geqslant 0}z^nq^{n^2}/\pi_{1\leqslant k\leqslant n}(1-q^k)$——是把 $p_1+\cdots+p_n$ 分成 n 部分的分划的生成函数，其中对于 $1\leqslant j<r$，有 $p_j>p_{j+1}+2$，且 $p_n>0$（参考习题 5.1-16）。

16. 当 $h=2$ 时，显然地对于通过格子图式右上角的通路出现极大值，即是

$$\binom{\lfloor n/2\rfloor+1}{2}$$

对于一般的 h，对应的数是

$$\bar{f}(n,h)=\binom{h}{2}\binom{q+1}{2}+\binom{r}{2}(q+1)$$

这里的 q 和 r 在定理 H 中定义：因为具有 $a_{i+j}h=$（当 $1\leqslant i\leqslant r$ 时为 $n+1-i(q+1)+j$，当 $r<i<h$ 时为 $n+1-r-iq+j$）的排列，使 $\binom{h}{2}$ 对排好序的子序列中的每一对之间的反序个数极大化。如果我们在（6）中以 $\bar{f}$ 代替 f，便得到极大的移动次数。

17. 有 $\binom{n+1}{2}$ 个反序的 $\{1,2,\cdots,2n\}$ 的唯一的二有序排列是 $(n+1)1(n+2)2\cdots(2n)n$。递归地利用这一思想，我们得到通过对序列 $(2^t-1)^R\cdots 1^R0^R$ 的每个元素加 1 来定义的排列，这里 R 表示把一个整数写成为一个 t 位二进数的操作，然后逆转这些数字从左到右的顺序！

18. 取出一个公共因子并命 $h_{t+1}=4N/\pi$；当 $h_1=1$ 时我们要把 $\sum_{t\geqslant s\geqslant 1}h_{s+1}^{1/2}/h_s$ 极小化。微商之得到 $h_s^3=4h_{s+1}^3h_{s-1}$，它有 $h_s=2^{s-2+s/(2^s-1)}h_{s+1}^{(2^s-1-1)/(2^s-1)}$ 的解。开始时给出的估计极小值成为 $(1-2^{-t})\pi^{(2^{t-1}-1)/(2^{t-1})}N^{1+(2^{t-1}+1)/(2^{t-1})}/2^{1+(t-1)/(2^{t-1})}$
当 $t\to\infty$ 时它迅速地趋于 $N\sqrt{\pi N}/2$。

当 $N=1000$ 时“最优的” h 的典型例子是（也看表 6）：

$$h_3=57.64,\quad h_2=6.13,\quad h_1=1,$$

$h_5=284.5,\quad h_4=67.23,\quad h_3=16.34,\quad h_2=4.028,\quad h_1=1;$

$h_{10}=9165,\quad h_9=12294,\quad h_8=7120,\quad h_7=2709,\quad h_6=835.5,\ \cdots,$

$h_2=3.97,\quad h_1=1$

19. 命 $g(n,h)=H_r-1+\sum_{r<j\leqslant h}q/(qj+r)$，这里 q 和 r 在定理H中定义；然后在（6）中以 g 代替 f。

20.（把它写出来，是比理解它更为困难的。）假设一个 k 有序文件 $R_1,\cdots,R_N$ 已经进行了 h 排序，并设 $1\leqslant i\leqslant N-k$；我们要来证明 $K_i\leqslant K_{i+k}$。求 u,v 使得 $1\leqslant u,v\leqslant h$；$i\equiv u$，且 $i+k\equiv v$ (modulo h)；并对于 $m=(u-v-k)/h$，$r=\lfloor(N-k+h-u)/h\rfloor$，$n+r=\lfloor(N+h-u)/h\rfloor$，$x_r=K_{v+(j-1)h}$，$y_j=K_{u+(j-1)h}$ 应用引理L。

21. 如果 $ah+bk=hk-h-k$，那么 $a \bmod k=k-1$ 并且 $b \bmod h=h-1$；因此 $ah+bk\geqslant(k-1)h+(h-1)k>hk-h-k$，矛盾。反之，如果 $n\geqslant(h-1)(k-1)$，则选择 a 使得 $0\leqslant a<k$ 且 $ah\equiv n$ (modulo k)。于是 $(n-ah)/k$ 是一个非负整数，b；因此 n 是可表示的。

稍微加强这个论证，即可看出在所述形式中恰有 $\frac{1}{2}(h-1)(k-1)$ 个正整数是不可表示的（见R. Z. Norman，*AMM*，67(1960)，594）。

22. 为了避免麻烦的记号，考虑 $h=4$，这代表了一般情况。设 n_k 是可用 $15a_0+31a_1+\cdots$ 的形式表示的最小数，该形式同余于 k (modulo15)；则我们容易发现

$k=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$n_k=$	15	31	62	63	94	125	126	127	158	189	190	221	252	253	254

因此 $239=2^4(2^4-1)-1$ 是最大的不可表示的数，而且不可表示的数的总数是

$$x_4=(n_1-1+n_2-2+\cdots+n_{14}-14)/15=(2+4+4+6+8+8)+8$$
$$+(10+12+12+14+16+16)+16=2x_3+8\cdot 9$$

一般，$x_h=2x_{h-1}+2^{h-1}(2^{h-1}+1)$。

对于另一个问题，答案分别是 $2^{2h}+2^h+2$ 和 $2^{h-1}(2^h+h-1)+2$。

23. N 个数的每一个在它的子文件中至多有 $\lceil(h_{i+2}-1)(h_{i+1}-1)/h_i\rceil$ 个反序。

24.（同V. 普拉特一起得到的解。）构造 $\{1,2,\cdots,N\}$ 的"h 违例排列"如下。以 $a_1\cdots a_N$ 个空白开始；然后对于 $j=2,3,4,\cdots$ 作步骤 j：每当 $(2^h-1)j-i$ 是如同习题22中那样可表示的一个正整数时，用在这个排列中还未出现的最小数从左到右地填入诸空白位置 a_i。继续进行，直到所有位置都填满为止。例如对于 $N=20$ 这个2违例排列是

6 2 1 9 4 3 12 7 5 15 10 8 17 13 11 19 16 14 20 18

对于所有的 $k\geqslant h$ 的 h 违例排列是（2^k-1）有序的。当 $2^h<j\leqslant N/(2^h-1)$ 时，在步骤 j 期间恰好填入 2^h-1 个位置；这些位置中的第 $k+1$ 个至少使对排列进行（$2^{k-1}-1$）排序所需要移动次数增加 $2^{k-1}-2k$ 次。因此当 $N=2^{k+1}(2^h-1)$ 时，为对具有增量 $h_i=2^i-1$ 的 h 违例排列进行排序所需移动次数 $>2^{3k-4}>\frac{1}{128}N^{3/2}$。普拉特在他的博士论文（Stanford University，1972）中已经把这个构造推广到包括（8）在内的一大类类似的序列上。

25. F_{n+1} 这一结果是由H·B·曼（H. B. Mann）得出的，"*Econometrica*"13，(1945)，

256〕；因为这个排列必须由 1 …或 2 …开头，至多有$\lfloor n/2 \rfloor$个反序；而且反序的总数是

$$\frac{n-1}{5}F_n+\frac{2n}{5}F_{n-1}$$

(见习题 1.2.8-12)。注意，F_{n+1}个排列可以方便地由点和破折号的“莫尔斯代码”序列表示，这里破折号对应于反序；于是当 $n=4$ 时，排列

1 2 3 4，　1 2 4 3，　1 3 2 4，　2 1 3 4，　2 1 4 3

分别对应于长度为 4 的莫尔斯代码序列

· · · ·，· · —，· — ·，— · ·，——

因此我们已求出了长度为 n 的所有莫尔斯代码序列当中破折号的总数。

在 3-排序之后再来一个 2-排序不会产生一个“随机的” 3-及 2-有序排列，因此反序的总数不向我们提供关于在算法 D 中得到的平均数的任何确切信息。

26. 是的；一个最短的例子是 4 1 3 7 2 6 8 5，它有 9 个反序。一般地说，对于 $-1\leqslant s\leqslant 1$，构造 $a_{3k+s}=3k+4s$ 产生 3-有序，5-有序，7-有序的一些文件，并有近似于$\frac{4}{3}N$个反序。当$N \bmod 3=2$时，这个构造是最好的。

28. 255 63 15 7 3 1。但可能还有更好的序列，见习题 29。

29. 根据 1971 年 C. 特里波里特（C. Tribolet）的实验，被选中的值是 373 137 53 19 7 3 1（$B_{ave}\approx 7100$）及 317 101 31 11 3 1（$B_{ave}=8300$）〔这些值中的头一个需要 12 64 57 u 的排序时间，而当利用增量（8）对相同的数据进行排序时为 1 3 1 0 0 2 u〕。一般说来，特里波里特建议命 h_n 是最接近于 $N^{(t-1)/t}$ 的质数。1972 年谢尔比·西格尔（Shelby Siegel）的实验指出，对于$N\leqslant 10000$，在这样的方法下最好的增量数是 $t\approx\frac{4}{3}\ln(N/5.75)$。

小罗伯特·L. 汤姆林森（Robert L. Tomlinson）找到的另一个好的序列，是 199 79 31 11 5 1（$B_{ave}\approx 7600$）。

根据卡罗利·M. 麦克纳米所作的广泛测试，最好的 3 增量序列似乎是 45 7 1（$B_{ave}\approx 17600$）。对于 4 增量，在她的测试中 91 23 7 1 是优胜者（$B_{ave}\approx 11500$），但是一个颇为广泛的增量范围也大体给出相同的性能。

30. 在三角区域$\{x\ln 2+y\ln 3<\ln N,\ x\geqslant 0\ \ y\geqslant 0\}$中的整数点的个数是$\frac{1}{2}(\log_2 N)(\log_3 N)+O(\log_2 N)$。由定理 K，在我们进行 h 排序时，这个文件已经是 $2h$ 有序的和 $3h$ 有序的；因此习题 25 在此适用。

31.

```
01   START   ENT3    T                    1
02   1H      LD4     H,3                  T
03           ENN2    -INPUT-N,4           T
04           ST2     6F(0:2)              T
05           ST2     7F(0:2)              T
06           ST2     4F(0:2)              T
07           ENT2    0,4                  T
```

08		JMP	9F	T
09	2H	LDA	INPUT+N,1	$NT-S-B+A$
10	4H	CMPA	INPUT+N-H,1	$NT-S-B+A$
11		JGE	8F	$NT-S-B+A$
12	6H	LDX	INPUT+N-H,1	B
13		STX	INPUT+N, 1	B
14	7H	STA	INPUT+N-H,1	B
15		INC1	0,4	B
16	8H	INC1	0,4	$NT-B+A$
17		J1NP	2B	$NT-B+A$
18		DEC2	1	S
19	9H	ENT1	-N,2	$T+S$
20		I2P	8B	$T+S$
21		DEC3	1	T
22		J3P	1B	T

在程序D中A是自左至右的极小值的个数，在此意义下，这里A是自右到左的极大值的个数；这两个量都有相同的统计特性。在内循环中的简化已经把运行时间减少到$7NT+7A-2S+1+15T$单位，奇怪，竟同B无关！

当$N=8$时，增量是6，4，3，2，1，而且我们有$A_{ave}=3.892$，$B_{ave}=6.762$；平均总的运行时间是276.24u（参考表5）。A和B两者在排列7 3 8 4 5 1 6 2中都取极大值。当$N=1000$时，有40个增量，972，864，768，729，…，8，6，4，3，2，1；类似于表6中的经验测试给出$A\approx875$，$B\approx4250$，而且总的时间大约是268000u（大约是采用习题28增量的程序D的两倍之长）。

我们不在一个辅助表中存储增量，而是很方便地在一台二进机器上如下地生成它们：

P1. 置$m\leftarrow2^{\lceil\log_2N\rceil-1}$，小于$N$的2的最大幂。

P2. 置$h\leftarrow m$。

P3. 使用h作为一次排序扫描的增量。

P4. 如果h是偶数，则置$h\leftarrow h+h/2$；然后如果$h<N$，则返回到P3。

P5. 置$m\leftarrow\lfloor m/2\rfloor$，而且如果$m\geqslant1$则返回P2。尽管增量不以递减次序来生成，但这里确定的次序足以使排序算法是有效的。

32. 4 12 11 13 2 0 8 5 10 14 1 6 3 9 16 7 15。

33. 可以作出两种类型的改进。首先，假定人为的键K_0是∞，这样我们可以不必测试p是否>0（例如，这个思想已经用于算法2.2.4A当中）其次，一个标准的“优化”技术：我们可以设置两份内循环，一个对P，另一个对q作寄存器赋值；这就避免了$q\leftarrow p$的赋值（这个思想已经用于习题1.1-3中）。

于是，我们假定单元INPUT在它的（0：3）场中包含最大可能的值，而且以下列程序代替程序L中的行07和它后面的诸行：

07	8H	LD3	INPUT,2(LINK)	B'	$P\leftarrow L_q$（这里$P\equiv$r[3]，$q\equiv$r[2]）
08		CMPA	INPUT,3(KEY)	B'	
09		JG	4F	B'	如果$K>K_p$则转L4且$q\leftrightarrow P$

10	7H	ST1	INPUT,2(LINK)	N'	$L_q \leftarrow j$
11		ST3	INPUT,1(LINK)	N'	$L_j \leftarrow P$
12		JMP	6F	N'	转去使 j 减值
13	4H	LD2	INPUT,3(LINK)	B''	$P \leftarrow L_q$(这里 p =rI2、 q =rI3)
14		CMPA	INPUT,2(KEY)	B''	
15		JG	8B	B''	如果$K>K_p$则转L4，且$q \leftarrow P$
16	5H	ST1	INPUT,3(LINK)	N''	$L_q \leftarrow j$
17		ST2	INPUT,1(LINK)	N''	$L_j \leftarrow p$
18	6H	DEC1	1	N	$j \leftarrow j-1$
19		ENT3	0	N	$q \leftarrow 0$
20		LDA	INPUT,1	N	$K \leftarrow K_1$
21		J1P	4B	N	$N \geqslant j \geqslant 1$

这里 $B'+B''=B+N-1$，$N'+N''=N-1$，所以总共的运行时间为 $5B+14N+N'-3$ 单位（N 是在其右边具有奇数个较小元素的元素个数）。所以它的统计数字是（min 0，ave $\frac{1}{2}N+\frac{1}{4}H_{\lfloor N/2\rfloor}-\frac{1}{2}H_N$，max$N-1$）。

∞的技巧也可以加速程序 S；J. H. 霍尔柏林（J. H. Halperin）建议的以下代码使用了这一思想以及 MOVE 指令，从而使运行时间减少到 $(6B+11N-10)u$，其中假定 INPUT+N+1 单元中已经包含有最大可能的单字值：

01	START	ENT2	N－1	1
02	2H	LDA	INPUT,2	$N-1$
03		ENT1	INPUT,2	$N-1$
04		JMP	3F	$N-1$
05	4H	MOVE	1,1(1)	B
06	3H	CMPA	1,1	$B+N-1$
07		JG	4B	$B+N-1$
08	5H	STA	0,1	$N-1$
09		DEC2	1	$N-1$
10		J2P	2B	$N-1$

加倍内循环还将另外节省 $B/2$ 左右的时间单位。

34．有$\binom{N}{n}$个N种选择的序列，其中给定的表被选择 n 次；每个这样的序列出现的概率是 $(1/M)^n(1-1/M)^{N-n}$，因为给定的表以 $1/M$ 的概率被选择。

35．程序L：$A=3$，$B=41$，$N=16$，时间$=496u$。程序M：$A=2+1+1+3=7$，$B=2+0+3+3=8$，$N=16$，时间$=549u$（因为有 16 个乘法。但是习题 33 中改进的程序L仅花费 $358u$）。

36．为了使 $AC/M+BM$ 取得极小值，我们需要$M=\sqrt{AC/B}$，所以M是恰恰在这个量之上或之下的整数之一（在程序M的情况下，我们将选择M同N成比例）。

37．可如同在习题 34 中那样证明。所述的恒等式等价于

$$g_{NM}(z)=M^{-N}\sum_{n_1+\cdots+n_M=N}\left(\frac{N!}{n_1!\cdots n_M!}\right)g_{n_1}(z)\cdots g_{n_M}(z)$$

把其中某些母函数列成表，以指出M增长的趋势，可能是有趣的：

$$g_{41}(z)=(216+648z+1080z^2+1296z^3+1080z^4+648z^5+216z^6)/5184$$
$$g_{42}(z)=(945+1917z+1485z^2+594z^3+135z^4+81z^5+27z^6)/5184$$
$$g_{43}(z)=(1704+2264z+840z^2+304z^3+40z^4+24z^5+8z^6)/5184$$

所述恒等式对 z 微商两次，然后置 $z=1$ 得到

$$G''_{NM}(1)=M^{-N}(M(M-1)\sum_{m,n}\binom{N}{m,n,N-m-n}\times(M-2)^{N-m-n}g'_m(1)g'_n(1)$$
$$+M\sum_n\binom{N}{n}(M-1)^{N-n}g''_n(1))$$

利用公式

$$g''_n(1)=\frac{3}{2}\binom{n}{4}+\frac{5}{3}\binom{n}{3}$$

这些和式没有造成什么特殊的困难，而且我们发现

$$G''_{NM}(1)=\left(\frac{3}{2}\binom{N}{4}+\frac{5}{3}\binom{N}{3}\right)M^2$$

最后，由于

$$G'_{NM}(1)=\frac{1}{2}\binom{N}{2}\Big/M,\quad G'_{NM}(1)^2=\left(\frac{3}{2}\binom{N}{4}+\frac{3}{2}\binom{N}{3}+\frac{1}{4}\binom{N}{2}\right)\Big/M^2$$

方差是

$$\left(\frac{1}{6}\binom{N}{3}+\frac{(2M-1)}{4}\binom{N}{2}\right)\Big/M^2$$

38. $\sum_{j,n}\binom{N}{n}p_j^n(1-p_j)^{N-n}\binom{n}{2}=\binom{N}{2}\sum_j p_j^2$；置 $p_j=F(j/M)-F((j-1)/M)$，且 $F'(x)=f(x)$，当 F 有相当好的特性时，这收敛于 $\binom{N}{2}/M$ 乘 $\int_0^1 f(x)^2dx$。

39. (R. W. 弗洛伊德的解。) 一个删去插入操作实质上仅仅移动 a_i。在一个这样的操作序列中，未移动的元素保留它们的相对次序，因此如果 π 的排序可以通过 k 个删去插入来完成则它有长度为 $n-k$ 的递增子序列；而且反之亦然。因此 $\mathrm{dis}(\pi)=n-(\pi$ 的最长递增子序列的长度)。(M. L. 弗雷德曼 (M. L. Fredman) 已经证明，为计算这个长度所需要的极小比较次数是 $n\log_2 n-n\log_2\log_2 n+O(n)$〔*Discrete Math*, 11(1975), 29-36〕。

5.2.2 节

1. 否，它少 $2m+1$ 个反序，其中 $m\geqslant 0$ 是使得 $i<k<j$ 且 $a_i>a_k>a_j$ 的元素 a_k 的个数 (因此所有的交换排序方法最终都将收敛于一个排好序的排列)。

2. (a) 6。(b)〔A. Cayley Philos. Mag. 34 (1849), 527-529.〕考虑 π 的循环表示：在同一循环中的任何元素交换，都使循环数加 1；在不同循环中的任何元素交换都使循环数减 1 (这实际上是习题 2.2.4-3 的内容)。一个完全排好序的排列通过有 n 个循环

来表征。因此 xch(π) 是 n 减 π 中的循环个数。

〔算法 5.2.35 恰恰进行 xch(π) 个交换，见习题 5.2.3-4。〕

3. 是，相等的元素决不能彼此交叉地移动。

4. 在反序表中 $b_1>b_2$，…，bn 的概率，即

$$\left(\sum_{1\leqslant k<n} k!\cdot k^{n-k-1}\right)\Big/ n!=\sqrt{\pi/2n}+O(n^{-1})=\text{可忽略的。}$$

5. 我们可以假定 $r>0$。设 $b'_i=(b_i\geqslant r\Rightarrow b_i-r+1;\ 0)$ 是在 $r-1$ 次扫描之后的反序表。如果 $b'_i>0$，则元素 i 就居于 b'_i 个较大的元素之后，这些较大的元素中的最大者至少将往上冒到 b'_i+i 的位置（由于 i 个元素 $\leqslant i$）。而且如果元素 j 是有待交换的最右者，则在第 r 次扫描之后，我们有 $b'_j>0$ 和 $\text{BOUND}=b'_j+j-1$。

6. 解法 1：一个在它最后位置右边并离此最右位置最远的元素，在每次扫描中都左移一步，最后一个元素除外。解法 2（更高级的）：由习题 5.1.1-8，答案（f），对于 $1\leqslant i\leqslant n$，$a'_i-i=b_i-c_i$，其中 $c_1c_2\cdots c_n$ 是对偶的反序表。如果 $b_j=\max(b_1,\ \cdots,\ b_n)$ 则 $c_j=0$。

7. $(2(n+1)(1+P(n)-P(n+1))-P(n)-P(n)^2)^{1/2}=\sqrt{(2-\pi/2)n}+O(1)$。

8. 当 $i<k+2$ 时，对于 b_i 有 $j+k-i+1$ 种选择；当 $k+2\leqslant i<n-j+2$ 时有 $j-1$ 种选择；而且当 $i\geqslant n-j+2$ 时有 $n-i+1$ 种选择。

10. (a) 如果 $i=2k-1$，则从 $(k-1,\ a_i-k)$ 到 $(k,\ a_i-k)$。若 $i=2k$，则从 $(a_i-k,\ k-1)$ 到 $(a_i-k,\ k)$。(b) 步骤 a_{2k-1} 在对角线的上部当且仅当 $k\leqslant a_{2k-1}-k$ 当且仅当 $a_{2k-1}\geqslant 2k$ 当且仅当 $a_{2k-1}>a_{2k}$ 当且仅当 $a_{2k}\leqslant 2k-1$ 当且仅当 $a_{2k}-k\leqslant k-1$ 当且仅当步骤 a_{2k} 在对角线的上部。交换它们就对换了水平的和垂直的步骤。(c) 步骤 a_{2k+d} 在对角线之下距离至少是 m 的地方当且仅当 $k+m-1\geqslant a_{2k+d}-(k+m)+m$ 当且仅当 $a_{2k+d}<2k+m$，当且仅当 $a_{2k}\geqslant 2k+m$ 当且仅当 $a_{2k}-k\geqslant k+m$ 当且仅当步骤 a_{2k} 在对角线之下距离至少是 m 的地方（如果 $a_{2k+d}<2k+m$ 且 $a_{2k}<2k+m$，则至少有 $(k+m)+k$ 个元素小于 $2k+m$；这是不可能的。如果 $a_{2k+d}\geqslant 2k+m$ 且 $a_{2k}\geqslant 2k+m$，则 $\geqslant$ 中有一个必是 $>$；但我们不能把 $\leqslant 2k+m$ 的所有元素都填入少于 $(k+m)+k$ 个位置中。因此 $a_{2k+d}<a_{2k}$ 当且仅当 $a_{2k+d}<2k+m$ 当且仅当 $2k+m\leqslant a_{2k}$。这是一个相当意外的结果！）

11. 考虑格子图式，即得 16 10 13 5 14 6 9 2 15 8 11 3 12 4 7 1（61 个交换）。当 $N=64$ 时这一情况变得更为复杂；R. 塞奇威克（R. Sedgwick）已经说明怎样来计算最坏情况的排列，而且他还证明了极大交换次数是 $1-\log_2\log_2 N/\log_2 N+O(1/\log_2 N)$ 乘以比较次数〔SIAM J. Computing，待出版〕。

12. 在这个程序当中，$\text{TT}\equiv 2^{t-1}$，$p\equiv$ rI1，$r\equiv$ rI2，$i\equiv$ rI3，$i+d-N\equiv$ rI4；假定 $N\geqslant 2$。

01	START	ENT1	TT	1	*M*1. 初始化 p，$p\leftarrow 2^{t-1}$
02		ST1	P	1	
03	2H	ENT2	TT	T	*M*2. 初始化 q，r，d
04		ST2	Q(1:2)	T	$q\leftarrow 2^{t-1}$

05		ENT2	0	T	$r \leftarrow 0$
06		ENT4	0,1	T	rI4$\leftarrow P$
07	3H	DEC4	N	A	$M3$. 对 i 进行循环
08		ENT3	0	A	$i \leftarrow 0$
09	8H	ENTA	0,3	$C+E$	(这时rI4<0)
10		AND	P	$C+E$	rA$\leftarrow i \wedge p$
11		DECA	0,2	$C+E$	
12		JAZ	4F	$C+E$	如果 $i \wedge p = r$ 则转移
13		INC3	0,1	D	否则 $i \leftarrow i + p$
14		INC4	0,1	D	
15		J4NN	5F	D	如果 $i + d \geqslant$ 则出循环
16	4H	LDA	INPUT+1,3	C	$M4$.比较/交换
17		CMPA	INPUT+N+1,4	C	$R_{i+1}:R_{i+d+1}$
18		JLE	*+4	C	
19		LDX	INPUT+N+1,4	B	
20		STX	INPUT+1,3	B	
21		STA	INPUT+N+1,4	B	
22		INC3	1	C	$i \leftarrow i + 1$
23		INC4	1	C	
24		J4N	8B	C	如果 $i + d < N$ 则重复循环
25	5H	ENT2	0,1	A	$M5$. 对 q 进行循环 $r \leftarrow p$
26	Q	ENT4	*	A	rI4$\leftarrow q$
27		ENTA	0,4	A	
28		SRB	1	A	
29		STA	Q(1:2)	A	$q' \leftarrow q/2$
30		DEC4	0,1	A	rI4$\leftarrow q-p$
31		J4P	3B	A	如果 $q \neq p$ 则转向M3
32	6H	ENTA	0,1	T	$M6$. 对 p 进行循环
33		SRB	1	T	
34		STA	P	T	
35		LD1	P	T	$p \leftarrow \lfloor p/2 \rfloor$
36		JANZ	2B	T	如果 $p \neq 0$ 则转向M2

运行时间依赖于六个量，其中仅有一个依赖于输入数据（剩下五个仅是 N 的函数）：$T=t$，即“主循环”的数目；$A=t(t+1)/2$，扫描的次数或“次循环”的数目；$B=$（可变的）交换次数；$C=$比较次数；$D=$在步骤M2中 $i \wedge p \neq r$ 的次数；$E=$在步骤M2中 $i \wedge p \neq r$ 和 $i+p+d \geqslant N$ 的次数。当 $N=2^t$ 时，可以证明 $D=2^t(t-2)+2+E$ 及 $E=\binom{t}{2}$。对于表1，$T=4$，$A=10$，$B=3+0+1+4+0+0+8+0+4+5=25$，$C=63$，$D=40$，$E=6$，所以总运行时间是 $11A+6B+13C+3D+5E+13T+3=1284u$。

13．否，算法Q、R都不是。

14．（a）当 $p=1$ 时，我们对于最后的合并，进行 $(2^{t-1}-0)+(2^{t-1}-1)+(2^{t-1}-$

$2)+(2^{t-1}-4)+\cdots+(2^{t-1}-2^{t-2})=(t-1)2^{t-1}+1$ 次比较。（b）$x_t=x_{t-1}+\frac{1}{2}(t-1)+2^{-t}=\cdots=x_0+\sum_{0\leqslant k<t}\left(-\frac{1}{2}k+2^{-k-1}\right)=-\frac{1}{2}\binom{t}{2}+1-2^{-t}$，因此 $c(2^t)=2^{t-2}(t^2-t+4)-1$。

15.（a）考虑使 $i+d=N$ 的比较次数；然后对 r 使用归纳法。（b）如果 $b(n)=c(n+1)$，则我们有 $b(2n)=a(1)+\cdots+a(2n)=a(0)+a(1)+a(1)+\cdots+a(n-1)+a(n)+x(1)+x(2)+\cdots+x(2n)=2b(n)+y(2n)-a(n)$；类似地，$b(2n+1)=2b(n)+y(2n+1)$。（c）参考习题 1.2.4-42。（d）对于 $(z(N)+2z(\lfloor N/2\rfloor)+\cdots)-a(N)$ 的一个颇为费劲的计算，并利用诸如

$$\sum_{0\leqslant k\leqslant n}2^k(n-k)=2^{n+1}-n-2,\quad \sum_{0\leqslant k\leqslant n}2^k\binom{n-k}{2}=2^{n+1}-\binom{n+2}{2}-1$$ 的

公式，即得

$$c(N)=N\left(-\frac{1}{2}\binom{e_1}{2}+2e_1-1\right)-2^{e_1}(e_1-1)-1$$
$$+\sum_{1\leqslant j\leqslant r}2^{e_j}\left(e_1+\cdots+e_{j-1}-j(e_1-1)+\frac{1}{2}\binom{e_1-e_j}{2}\right)$$

16. 考虑如同图 11 和 18 中从（0，0）到（n，n）的 $\binom{2n}{n}$ 条格子通路，而且如果 $i\geqslant j$ 便附加权 $f(i-j)$，如果 $i<j$ 则附加权 $f(j-i+1)$ 到从（i，j）到（$i+1$，j）的边上；这里 $f(k)$ 是在二进展开 $k=(\cdots b_2b_1b_0)_2$ 中二进位 $b_r\neq b_{r+1}$ 的个数。当 $N=2n$ 时在最后合并中的交换总次数为

$$\sum_{0\leqslant j\leqslant i<n}(2f(j)+1)\binom{2i-j}{i-j}\binom{2n-2i+j-1}{n-i-1}$$

R. 塞奇维克已经把这个和简化成为

$(1/2)n\binom{2n}{n}+2\sum_{k\geqslant 1}\binom{2n}{n-k}\sum_{0\leqslant j<k}f(j)$ 并且使用伽玛函数的方法得到渐近公式

$\binom{2n}{n}((1/4)n\log_2 n+(\log_2(\Gamma(1/4)^2/2\pi)+1/4-(\gamma+2)/(4\ln 2)+\delta(n))n+O(\sqrt{n}\log_2 n))$，其中 $\delta(n)$ 是 $\log_2 n$ 的一个周期函数且有 .0005 的数量界限；因此当 $n\to\infty$ 时，平均说来，大约四分之一的比较导致交换〔SIAM J. Computing，待发表〕。

17. 当我们对一个 $r=N$ 且 K_l 是最大键的子文件排序时，检查 K_{N+1}。当自左至右的极小值落在位置 R_1 时，在步骤Q9 期间，检查 K_0。

18. 在离开Q5 之前，步骤Q3 和Q4 仅作 i 和 j 的一个改变；$R_l\cdots R_r$ 的分划过程在步骤Q7 中以 $j=\lceil(l+r)/2\rceil$ 结束，并且尽可能完善地分开这个子文件。定量地说，我们以 $A=1$，$B=\lfloor(N-1)/2\rfloor$，$C=N+(N \bmod 2)$ 代替（17）；除非 $B\approx\frac{1}{2}C$，这

实质上使我们处于此算法的最好情况（参看习题27）。如果把步骤Q3和Q4的“<”号变为“≤”，则本算法将不再排序；即使在（13）中我们取“<”号，它将交换R_0和R_1，那样第三个分划阶段将把原来的R_0移到位置R_2去，等等，即成为一个真正的灾难。

19．是，其它文件可以以任意顺序来处理（而无须增加存储区域的大小）；但若用一个栈则最易于进行工作。

20．$\lfloor \log_2(N+1)/(M+2) \rfloor$。（若$N=2^k(M+1)-1$且若划分子文件时所有的子文件皆被完全二等分，则此时出现最坏的情况。）

21．在步骤Q6中恰有t个记录移到区域$R_{s+1}\cdots R_N$中，因此$B=t$。划分阶段以$j=s$结束，因此$C-C'=N+1-s$是j减少的次数。当诸键不同时，在步骤Q7中我们也有$i=s+1$，因为$i=j$蕴涵$K_j=K$；于是$C'=s$。

22．由于$A_{s-1}(z)A_{N-s}(z)$是在独立地对大小为$s-1$和$N-s$的随机独立有序文件排序之后，A值的母函数，因此容易得出$A_N(z)$的所述关系。类似地，我们对于$N>M$得到关系式

$$B_N(z)=\sum_{1\leqslant s\leqslant N}\ \sum_{0\leqslant k\leqslant s} b_{stN}z^t B_{s-1}(z)B_{N-s}(z)$$

$$C_N(z)=\frac{1}{N}\sum_{1\leqslant s\leqslant N} z^{N+1}C_{s-1}(z)C_{N-s}(z)$$

$$D_N(z)=\frac{1}{N}\sum_{1\leqslant s\leqslant N} D_{s-1}(z)D_{N-s}(z)$$

$$E_N(z)=\frac{1}{N}\sum_{1\leqslant s\leqslant N} E_{s-1}(z)E_{N-s}(z)$$

$$L_N(z)=\frac{1}{N}\sum_{1\leqslant s\leqslant N} L_{s-1}(z)L_{N-s}(z)$$

$$X_N(z)=\frac{1}{N}\sum_{1\leqslant s\leqslant N}(1+(z-1)h_{sN})X_{s-1}(z)X_{N-s}(z)$$

这里b_{stN}是s和t在长度为N的一个文件中有给定值的概率，即

$$\binom{s-1}{t}\binom{N-s}{t}\Big/N\binom{N-1}{s-1}$$

它是（$1/N!$）乘（$s-1$）！种对$\{1,\cdots,s-1\}$进行排列的方法乘以（$N-s$）！种对$\{s+1,\cdots,N\}$进行排列的方法乘每边上具有t个被移动元素的$\binom{s-1}{t}\binom{N-s}{t}$种型式；而且$h_{sN}$是当$N$和$s$已经给定时$h=1$的概率，即（$s-1$）/（$N-1$）。对于$0\leqslant N\leqslant M$，我们有$B_N(z)=C_N(z)=X_N(z)=1$；$L_N(z)=1$，但$L_0(z)=L_1(z)=z$；$D_N(z)=z^{N-1}$，但$D_0(z)=1$；而且$E_N(z)$是$\Pi_{1\leqslant k\leqslant N}((1+z+\cdots+z^{k-1})/k)$。

23．当$N>M$时，$A_N=1+(2/N)\sum_{0\leqslant k<N}A_k$；$B_N=\sum_{0\leqslant t<s\leqslant N}b_{stN}(t+B_{s-1}$

$+B_{N-s})=(1/N)\sum_{1\leqslant s\leqslant N}((s-1)(N-S)/(N-1)+B_{s-1}+B_{N-s})=(N-2)/6$ $+(2/N)\sum_{0\leqslant k<N}B_k$[参考习题22]；$D_N=(2/N)\sum_{0\leqslant k<N}D_k$；$E_N$和$L_N$是类似的；$X_N=(1/N)\sum_{1\leqslant s\leqslant N}(h_{sN}+X_{s-1}+X_{N-s})=-\frac{1}{2}+(2/N)\sum_{0\leqslant k<N}X_k=-\frac{1}{2}-A_N$。这些递归式的每一个对于某个函数$f$。有形式(20)。

24. 递归式$C_N=N-1+(2/N)\sum_{0\leqslant k<N}C_k$，$N>M$，有解$(N+1)(2H_{N+1}-2H_{M+2}+1-4/(M+2)+2/(N+1))$，$N>M$(所以我们节省了大约$4N/M$次比较，但每次比较要花费较长的时间)。

25. (对于$s=1$重复地利用(17)。)$A=N-M$，$B=0$，$C=\binom{N+1}{2}-\binom{M+1}{2}$，$D=M-1$，$E=0$，$L=N-M+\delta_{M1}$，$X=0$。

26. 实际上你不能比对$123\cdots(N-M)N(N-1)\cdots(N-M+1)$排序做得更糟：更微妙的答案$N(M-1)(M-2)\cdots1M(M+1)\cdots(N-1)$同样是一种坏的情况。这仅仅比习题25坏一点，因为它使$D=M-1$，$E=\binom{M}{2}$。

27. 12 2 3 1 8 6 7 5 9 10 11 4 16 14 15 13 20 18 19 17 21 22 23，它要求$546u$。可以证明，当通过分划来分开子文件直到达到$3M+2$的大小时，出现$N=3(M+1)2^k-1$的最坏情况；然后分成三分以避免压入栈的开销。我们有$A=3\cdot2^k-1$，$C=\left(k+\frac{5}{3}\right)(N+1)$，$S=2^k-1$，$B=D=E=0$。(一般的$M$和$N$的最坏情况的特性作成一个有趣的但复杂的模式。)

28. 递归式

$$C_N=n+1+\frac{2}{\binom{n}{3}}\sum_{1\leqslant k\leqslant n}(k-1)(n-k)C_{k-1}$$

可以被转换成

$$\binom{n}{3}C_n-2\binom{n-1}{3}C_{n-1}+\binom{n-2}{3}C_{n-2}=2(n-1)(n-2)+2(n-2)C_{n-2}$$

29. 一般地说，考虑递归式

$$C_n=n+1+\frac{2}{\binom{n}{2t+1}}\sum_{1\leqslant k\leqslant n}\binom{k-1}{t}\binom{n-k}{t}C_{k-1}$$

当$2t+1$个元素的均值支配分划时就发生这种情况。命$C(z)=\Sigma_nC_nz^n$，递归式可转换成$(1-z)^{t+1}C^{(2t+1)}(z)/(2t+2)!=1/(1-z)^{t+2}+C^{(t)}(z)/(t+1)!$。命$f(x)=C^{(t)}(1-x)$；则$p_t(\theta)f(x)=(2t+2)!/(x)^{t+2}$，其中$\theta$表示算子$x(d/dx)$，$p_t(x)=(t-x)^{\underline{t-1}}-(2t+2)^{\underline{t+1}}$。对于$\alpha\neq\beta$，$(\theta-\alpha)g(x)=x^\beta$，的通解是$g(x)=x^\beta/(\beta-\alpha)+Cx^2$；对于$\alpha=\beta$是$g(x)=x^\beta(\ln x+C)$。我们有$p_t(-t-2)=0$；所

以微分方程通解是

$$C^{(t)}(z)=(2t+2)!\ln(1-z)/p_t'(-t-2)(1-z)^{t+2}+\sum_{0\leqslant j\leqslant t}c_j(1-z)^{\alpha_j}$$

这里 α_0，$\cdots\alpha_t$ 是 $p_t(x)=0$ 的根，而且常数 c_j 依赖于初值 C_t，$\cdots$，C_{2t}。由手边的恒等式

$$\frac{1}{(1-z)^{m+1}}\ln\left(\frac{1}{1-z}\right)=\sum_{m\geqslant 0}(H_{n+m}-H_m)\binom{n+m}{m}z^n,\quad m\geqslant 0$$

导出惊人地简单的封闭形式的解

$$C_n=(H_{n+1}-H_{t+1})(n+1)/(H_{2t+2}-H_{t+1})+\left(\sum_{0\leqslant j\leqslant t}c_j(-\alpha_j)^{\overline{n-t}}\right)/n!$$

由此容易导出渐近公式。（前导项 $n\ln n/(H_{2t+2}-H_{t+1})$ 是由 M. H. 范·埃姆登（M. H. van. Emden〔*CACM* 13 (1970)，563-567〕）发现的，他利用了一项信息论的方法。事实上，假设我们希望分析任何具下列性质的分划过程，它使得对于 $0\leqslant x\leqslant 1$，当 $N\to\infty$ 时左子文件以渐近概率 $\int_0^x f(x)dx$ 至多包含 xN 个元素；范·埃姆登已经证明，为了完整地对文件排序所需要的平均比较次数是 $\sim n\ln n/\alpha$，其中 $\alpha=-1/\int_0^1(f(x)+f(1-x)x\ln x\,dx$。这个公式既可应用于基数交换方法，也可用于快速排序和各种其它的方法。又见 H. Hurwitz，*CACM* 14 (1971), 99-102。）

30. 每个子文件都可以由四个量（l，r，k，X）来标识，这里 l 和 r 是边界（指当前的），k 表示在整个子文件中已知具有相等的键的字数，X 是诸键的第（$k+1$）个字的下界。采用非负的键，我们开始时有（l，r，k，X）=(1，N，0，0)。当分划一个文件时，我们命 K 是测试键 K_g 的第（$k+1$）个字。如果 $K>X$，则分划把所有 $\geqslant K$ 的键划在右边，所有 $<K$ 的键划在左边。（每次仅仅考察键的第（$k+1$）个字；通过分划产生的子文件的标识分别是（l，$i-1$，k，X）和（i，r，k，K）。）但如果 $K=X$ 则分划把所有 $>K$ 的键划在右边，所有 $\leqslant K$〔实际上 $=K$〕的键划在左边，通过分划产生的子文件的标识分别是（l，i，$k+1$，0）以及（$i+1$，r，k，K）。在两种情况下，都不能保证 R_j 是在它最后的位置，因为我们没有考察第（$k+2$）个字。为了适当处理边界条件，应作显然的进一步的修改。通过增加第五个“上界”分量，这个方法可以成为对左边和右边是对称的。

31. 通过一个正常的分划过程使 R_1 最后落到 R_k 上，如果 $k=m$，则停止；如果 $k>m$，则使用同样的技术来求右边子文件的第（$m-k$）个最小的元素，如果 $k<m$ 则找左边子文件的第 m 个最小元素〔*CACM* 4 (1961)，321-322；14(1971)39-45〕。

32. 递归式是 $C_{11}=0$ 和当 $n>1$ 时，$C_{nm}=n+1+(A_{nm}+B_{nm})/n$，其中对于 $1\leqslant m\leqslant n$

$$A_{nm}=\sum_{1\leqslant s<m}C_{n-s,m-s}\quad 和\quad B_{nm}=\sum_{m<s\leqslant n}C_{s-1,m}$$

由于 $A_{n+1,m+1}=A_{nm}+C_{nm}$ 和 $B_{n+1,m}=B_{nm}+C_{nm}$，因此我们可以首先求量 $D_n=(n+1)\times C_{n+1,m+1}-C_{nm}$ 的一个公式，然后对它们求和以得到答案 $2\Big((n+1)H_n-(n+2-m)H_{n+1-m}$

$-(m+1)H_m+n+\frac{5}{3}\Big)-\frac{1}{3}\delta_{mn}-\frac{1}{3}\delta_{m1}-\frac{2}{3}\delta_{mn}\delta_{m1}$。当 $n=2m-1$ 时，它变成为 $4m(H_{2m-1}-H_m)+4m-4H_m+\frac{4}{3}(1-\delta_{m1})=(4+4\ln 2)m-4\ln m-4\gamma-\frac{5}{3}+O(m^{-1})\approx 3.39$〔参见D. E. Knuth, *Proc. IFIP Congress* (1971), 19-27〕。

33. 象在基数交换的第一阶段那样进行，并且利用正负号代替二进位1。

34. 只要我们在每个阶段已经找到至少一个二进位0和至少一个二进位1，即在每个阶段中进行了头一次交换之后，即可以避免作是否 $i\leqslant j$ 的判断，这在程序R中节省了接近 $2C$ 的时间单位。

35. $A=N-1$, $B=(\min 0,\ \mathrm{ave}\frac{1}{4}N\log_2 N,\ \max\frac{1}{2}N\log_2 N)$, $C=N\log_2 N$, $G=\frac{1}{2}N$, $K=L=R=0$, $S=\frac{1}{2}N-1$, $X=\Big(\min 0,\ \mathrm{ave}\frac{1}{2}(N-1),\ \max N-1\Big)$。〔注意这样一点是有趣的：一般来说，量 A, C, G, K, L, R 和 S 仅依赖于文件中的键的集合，而不依赖于它们开始时的次序，仅仅 B 和 X 受键的初始顺序的影响。〕

36. (a) $\Sigma\binom{n}{k}\binom{k}{j}(-1)^{k+j}a_j=\Sigma\binom{n}{j}\binom{n-j}{k-j}(-1)^{k-j}a_j=\Sigma\binom{n}{j}\delta_{nj}a_j=a_n$。(b) $\langle\delta_{n0}\rangle$; $\langle-\delta_{n1}\rangle$; $\langle(-1)^n\delta_{nm}\rangle$; $\langle(1-a)^n\rangle$; $\left\langle\binom{n}{m}(-a)^m(1-a)^{n-m}\right\rangle$。(c) 把有待证明的关系式写成 $x_n=y_n=a_n+z_n$，由(a)部分我们有 $y_n=a_n+z_n$，而且 $2^{1-n}\Sigma_{k\geqslant 2}\binom{n}{k}y_k=z_n$，所以 y_n 和 x_n 满足相同的递归式〔关于这个结果的某些推广，见习题53和6.3-17。直接证明 $\hat{x}_n=\hat{a}_n 2^{n-1}/(2^{n-1}-1)$ 看来并不容易。〕

37. $\left\langle\Sigma_m c_m\binom{n}{2m}2^{-n}\right\rangle$，其中 c_0, $c_1, c_2, \cdots$ 为任意常数序列。〔这个答案尽管是正确的。但并不立即揭示出 $\langle 1/(n+1)\rangle$ 和 $\langle n-\delta_{n1}\rangle$ 是这样的序列！形式为 $\langle a_n+\hat{a}_n\rangle$ 的序列总是自对偶的。注意，借助于生成函数 $A(z)=\Sigma a_n z^n/n!$ 我们有 $\hat{A}(z)=e^z A(-z)$；因此 $A=\hat{A}$ 等价于说，$A(z)e^{-z/2}$ 是一个偶函数。〕

38. 产生大小为 s 的一个左文件和大小为 $N-s$ 的一个右文件的一个分划阶段，对总的运行时间作出如下的贡献：

$$A=1,\quad B=t,\quad C=N,\quad K=\delta_{s1},\quad L=\delta_{s0},\quad R=\delta_{sN},\quad X=h.$$

这里 t 是 $K_1, \cdots, K_s$ 诸键中二进位 b 等于1的键的个数，而且 h 是 K_{s+1} 的二进位 b；如果 $s=N$，则 $h=0$（参考(17)）。这导致诸如

$$B_N=2^{-N}\sum_{0\leqslant t\leqslant s\leqslant N}\binom{s}{t}\binom{N-s}{t}(t+B_s+B_{N-s})$$

$$=\frac{1}{4}(N-1)+2^{1-N}\sum_{s\geqslant 2}\binom{N}{s}B_s,\ \text{对于}N\geqslant 2,\ B_0=B_1=0$$

的递推方程（参考习题23）。用习题36的方法来解这些递归式，得出公式 $A_N=V_N-U_N+1$, $B_N=\frac{1}{4}(U_N+N-1)$, $C_N=V_N+N$, $K_N=N/2$, $L_N=R_N=\frac{1}{2}(V_N-U_N-N)$

$+1$，$X_N=\frac{1}{2}(A_N-L_N)$。显然 $G_N=0$。

39．快速排序的每个阶段至少把一个元素放入到它最后的位置，但在基数交换期间则不一定（参考表3）。

40．如果我们在步骤R2中每当 $r-l<M$ 时即转到直接插入，则这个问题不出现，除非有多于 M 个相等的元素出现。如果后者是一种可能的前景，则在步骤R8中每当 $j<l$ 或 $j=r$ 时，我们可以判断是否 $K_l=\cdots=K_r$。

43．当 $a\to 0+$ 时由习题1.2.7-24，$\int_0^1 y^{a-1}(e^{-y}-1)dy+\int_1^\infty y^{a-1}e^{-y}dy=\Gamma(a)-1/a=(\Gamma(a+1)-\Gamma(1))/a\to\Gamma'(1)=-\gamma$。

44．对于 $k\geqslant 0$，我们有 $r_k(m)\sim\frac{1}{2}(2m)^{(k+1)/2}\Gamma((k+1)/2)-\Sigma_{j\geqslant 0}(-1)^jB_{k+2j+1}/((k+2j+1)j!(2m)^j)$。当 $k=-1$ 时(36)中来自 $f_k^{(j-1)}(m)$ 的贡献与 H_{m-1} 的展开式中类似的项相消，而且我们有 $r_{-1}(m)=H_{m-1}+(1/\sqrt{2}m\Sigma_{t\geqslant 0}f_{-1}(t)\sim\frac{1}{2}(\ln(2m)+\gamma)-\Sigma_{j\geqslant 1}(-1)^jB_{2j}/(2j)j!(2m)^j$。因此由(33)的项 N_t/t 对 W_{m-1} 的贡献得自 $m\Sigma_{t\geqslant 1}t^{-1}\exp(-t^2/2m)(1-t^2/3m^2+t^3/18m^4)(1-t^4/4m^3)(1-t/2m-t^2/8m^2)+O(m^{-1/2})=\frac{1}{2}m\ln m+\frac{1}{2}(\ln 2+\gamma)m-\frac{5}{12}\sqrt{2\pi m}+\frac{4}{9}+O(m^{-1/2})$。项 $-\frac{1}{2}N^{t-1}$ 贡献 $-\frac{1}{2}\Sigma_{t\geqslant 1}\exp(-t^2/2m)(1-t^3/3m^2)(1-t/2m)(1+t/m)+O(m^{-1/2})=-\frac{1}{4}\times\sqrt{2\pi m}-\frac{1}{6}$。项 $\frac{1}{2}\delta_{t1}$ 产生 $-\frac{1}{2}$。最后项 $-\frac{1}{2}(t-1)B_2N^{t-2}$ 贡献 $\frac{1}{12}m^{-1}\Sigma_{t\geqslant 1}t\exp\times(-t^2/2m)+O(m^{-1/2})=\frac{1}{12}+O(m^{-1/2})$。

45．用来推导（42）的论证对于（43）也是有效的但应略去在 $z=-1$ 和 $z=0$ 处的残数。

46．如同我们对于（45）所做的那样，我们得到 $(s-1)!/\ln 2+f_s(n)$，这里

$$f_s(n)=\frac{2}{\ln 2}\sum_{k\geqslant 1}\mathscr{R}(\Gamma(s-2\pi ik/\ln 2)\exp(2\pi ik\log_2 n))$$

（注意：对于整数 $s\geqslant 0$，$|\Gamma(s+it)|^2=(\Pi_{0\leqslant k<s}(k^2+t^2))\pi/(t\sinh\pi t)$，所以我们可以给出 $f_n(s)$ 的界。）

47．事实上，对于所有 $s>0$，$\Sigma_{j\geqslant 1}e^{-n/2^j}(n/2^j)^s$ 等于习题46中的积分。

48．利用中间恒等式

$$1-e^{-x}=\frac{-1}{2\pi i}\int_{-1/2-i\infty}^{-1/2+i\infty}\Gamma(z)x^{-z}dz$$

我们沿用正文中的方法，但 $1-e^{-x}$ 起着 $e^{-x}-1+x$ 的作用：$V_{n+1}/(n+1)=$

$(-1/2\pi i)\int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty}\Gamma(z)n^{-z}dz/(2^{-z}-1)+O(n^{-1})$，而且在习题 26 的记号下，这个积分等于 $\log_2 n+\gamma/(\ln 2)-\frac{1}{2}-f_0(n)$〔于是在习题 38 中 A_N 是 $N(1/(\ln 2)-f_0(N-1)-f_{-1}(N))+O(1)$〕。

49. 等式 (40) 的右边可以改进成为 $e^{-x}\left(n/x+\frac{1}{2}x+x^3O(n^{-1})\right)$。其效果是减 $\frac{1}{2}$ 乘习题 47 中的和，以 $2-\frac{1}{2}(1/(\ln 2)+f_1(n))+O(n^{-1})$ 代替(47)中的 $O(1)$。("2" 从(45) 中的 "$2/n$" 得出。)

50. $U_{mn}=n\log_m n+n((\gamma-1)/(\ln m)-\frac{1}{2}+f_{-1}(n)+m/(m-1)-1/(2\ln m)-\frac{1}{2}f_1(n)+O(n^{-1})$，其中 $f_s(n)$ 象在习题 46 中那样定义，但以 $\ln m$ 和 $\log_2 m$ 来代替 $\ln 2$ 和 $\log_2$〔注意：对于 $m=2$，3，4，5，10，100，1000，10^6，我们分别有 $f_{-1}(n)<.0000001725$，.00041227，000296，.00085，.00627，.068，.153，.341〕。

51. 设 $N=2m$。我们可以推广和式 (35) 到所有 $t\geqslant 1$，那时它等于

$$\sum_{t\geqslant 1}(1/2\pi i)\int_{a-i\infty}^{a+i\infty}\Gamma(z)(t^2/N)^{-z}t^k dz$$

$$=(1/2\pi i)\int_{a-i\infty}^{a+i\infty}\Gamma(z)N^z\zeta(2z-k)dz$$

条件是 $a>(k+1)/2$。所以我们需要知道截塔函数的性质，当 $\Re(w)\geqslant -q$ 且 $|w|\to\infty$ 时 $\zeta(w)=O(|w|^{q+1})$；因此如果仅考虑残数，则我们可以随心所欲地向左移动积分的边。因子 $\Gamma(z)$ 在 0，-1，-2，…处有极点，$\zeta(2z-k)$ 仅在 $z=(k+1)/2$ 处有一个极点。在 $z=-j$ 处的残数是 $N^{-j}(-1)^j\zeta(-2j-k)/j!$ 而且 $\zeta(-n)=(-1)^{n+1}B_{n+1}/(n+1)$。在 $z=(k+1)/2$ 处的残数是 $-\frac{1}{2}\Gamma((k+1)/2)N^{(k+1)/2}$。但当 $k=-1$ 时，在 $z=0$ 处有一个双重极点；$\zeta(z)=1/(z-1)+\gamma+O(|z-1|)$，所以在这种情况下 0 点处的残数为 $\gamma+\frac{1}{2}\ln N-\frac{1}{2}\gamma$。我们因此得到在习题 44 的答案中所述的渐近级数。

52. 置 $x=t/n$；则

$$\binom{2N}{n+t}\Big/\binom{2n}{n}=\exp(-2n(x^2/1\cdot 2+x^4/3\cdot 4+\cdots)+(x^2/2+x^4/4+\cdots)$$
$$-(1/6n)(x^2+x^4+\cdots)+\cdots)$$

对于不同的 k 现在可以借助于 $\sum_{t\geqslant 1}t^k d(t)e^{-t^2/n}$ 来表达所求的和数。沿用习题 51 的方法，由于 $\zeta(z)^2=\sum_{t\geqslant 1}d(t)t^{-z}$，我们希望来计算当 $k\geqslant 0$ 时 $\Gamma(z)n^z\zeta(2z-k)^2$ 的残

数。在 $z=-z$ 处，这个残数是 $n^{-j}(-1)^j(B_{2j+k+1}/(2j+k-1))^2/j!$，而在 $z=(k+1)/2$ 处，这个残数是 $n^{(k+1)/2}\Gamma((k+1)/2)\left(\gamma+\frac{1}{4}\ln n+\frac{1}{4}\psi((k+1)/2)\right)$。其中 $\psi(z)=\Gamma'(z)/\Gamma(z)=H_{z-1}-\gamma$；于是，例如，当 $k=0$ 时对所有 M 有 $\Sigma_{t\geqslant 1}e^{-t^2/n}d(t)=\frac{1}{4}\sqrt{\pi n}\ln n+\left(\frac{3}{4}\gamma-\frac{1}{2}\ln 2\right)\sqrt{\pi n}+\frac{1}{4}+O(n^{-M})$。对于 $S_n\Big/\binom{2n}{n}$，把 $\left(\frac{1}{32}\ln n+\frac{3}{32}\gamma+\frac{1}{24}-\frac{1}{16}\ln 2\right)\sqrt{\pi/n}+O(n^{-1})$ 加到这个量上(参考习题 1.2.7-23，1.2.9-19)。

53. 设 $q=1-p$，推广习题 36(c)，如果

$$x_n=a_n+\sum_{k\geqslant 2}\binom{n}{k}(p^kq^{n-k}+q^kp^{n-k})x_k$$

则

$$x_n=a_n+\sum_{k\geqslant 2}\binom{n}{k}(-1)^k\hat{a}_k(p^k+q^k)/(1-p^k-q^k)$$

因此我们可以象以前那样求 B_N 和 C_N(B_N 中的因子 $\frac{1}{4}$ 应以 pq 代替)。U_N 的渐近性质可基本上如正文中那样考察，而且

$$T_n=\sum_{r\geqslant 1,\ s\geqslant 0}\binom{r}{s}(e^{-np^sq^{r-s}}-1+np^sq^{r-s})$$

$$=(1/2\pi i)\int_{-3/2-i\infty}^{-3/2+i\infty}\Gamma(z)n^{-z}(p^{-z}+q^{-z})dz/(1-p^{-z}-q^{-z})$$

$$=(n/h_p)(\ln n+\gamma-1+h_p^{(2)}/2h_p-h_p+f(n))+O(1)$$

其中

$$h_p=-(p\ln p+q\ln q),\quad h_p^{(2)}=p(\ln p)^2+q(\ln q)^2$$

而且

$$f(n)=\Sigma\Gamma(z)n^{-1-z}/h_p$$

这里求和对所有使得 $p^{-z}+q^{-z}=1$ 的复数 $z\neq -1$ 进行。后一点集一般地似乎是难于分析的；但当 $p=\phi^{-1}$，$q=\phi^{-2}$ 时，解为 $z=(-1)^{k+1}+k\pi i/(\ln\phi)$。主项 $(n\ln n)/h_p$ 也可以从习题 29 的解答中所引的范·埃姆登的一般公式得到。对于 $p=\phi^{-1}$，我们有 $1/h_p=1.521$ 而 $1/h_{1/2}=1.4427$。

54. 命 C 是半径为 $\left(M+\frac{1}{2}\right)b$ 的一个圆，于是当 $M\to\infty$ 时在 C 上的积分变为 0 (U_N 的渐近形式现在可以用一种新方法即展开 $\Gamma(n+1)/\Gamma(n+ibm)$ 来导出。当 f 有相当好的特性时，本题的方法可应用于形如 $\Sigma_k\binom{n}{k}(-1)^kf(k)$ 的所有和式上！)。

55. 用来代替程序Q的行 04～06，后边再接上 "STA INPUT+1，2"（参考 (27)

```
21H  ENTA   0,2                 STA   INPUT,3          JGE   5F
     INCA   0,3                 STX   INPUT,2          CMPX  INPUT,4
     SRB    1            5H     LDA   INPUT,4          JG    5B
     STA    *+1(0:2)           JMP   6F               LDA   INPUT,3
     ENT4   *            4H     LDA   INPUT,3          LDX   INPUT,4
     LDA    INPUT,2             LDX   INPUT,2          STX   INPUT,3
     LDX    INPUT,3             STX   INPUT,3          JMP   6F
     CMPA   INPUT,3             JMP   5F          5H   LDX   INPUT,4
     JL     1F           3H     STX   INPUT,2          STX   INPUT,2
     CMPA   INPUT,4             LDX   INPUT,4     6H   LDX   INPUT+1,2
     JLE    3F                  STX   INPUT,3          STX   INPUT,4
     CMPX   INPUT,4             JMP   6F               ENT4  2,2
     JG     4F           1H     CMPA  INPUT,4          ENT5  0,3
```

之后的注释)；并把行 22 的指令改成为“STX INPUT+1，2”。如果二进移位不能用，则头三条指令须改成“ENTX 0，2；INCX 0，3；ENTA 0；DIV=2=”。

这一程序实质上交换 R_{l+1} 和 $R_{\lfloor(l+r)/2\rfloor}$ 并对三个记录 R_l，R_{l+1}，R_r 进行排序，然后应用通常的分划到 $R_{l+1}\cdots R_{r-1}$ 上。通过简单地把中间元素放在 rA 中，把 R_l 传送到中间元素以前的位置，并且如同它现在这样来使用程序Q。但这样一个方法有坏结果，因为它需要 N^2 步的阶来对文件 $N(N-1)\cdots 1$ 进行排序。(这一惊奇的结果，首先是由D. B. 科德里克（D. B. Coldrick）发现的，它要你相信——试试看！)上边推荐的这一技术，是属于R. 塞奇维克的，看来避免了这种“简单的最坏情况”的异常，而且运行也更快。

56. 通过命 $y_n=nx_n$，$u_n=ny_{n+1}-(n+2)y_n$，$v_n=nu_{n+1}-(n-5)u_n$；我们可以解递归式 $\binom{n}{3}x_n=b_n+2\Sigma_{1<k\leqslant n}(k-1)(n-k)x_{k-1}$，$n>m$。由此得出对于 $n>m$，$v_n=6(b_{n+2}-2b_{n+1}+b_n)$。例如：对于 $n\leqslant m$，命 $x_n=\delta_{n1}$，且设 $b_n=0$。则对于所有的 $n>m$，$v_n=0$，因此 $n^5u_{n+1}=m^5u_{m+1}$。由于 $y_{m+1}=12/m$，$y_{m+2}=12/(m+1)$，我们最终求得对于 $n>m$，$x_n=\frac{48}{7}(n+1)/m(m+1)(m+2)+\frac{36}{7}(m-1)^4/n^6$。一般地，当 b_n 恒等于 0 时，命 $f_n=(12/(n+1)(n-2))\Sigma_{1\leqslant k\leqslant n}(k-1)(n-k)x_{k-1}$；对于 $n>m$ 的解是 $x_n=(n+1)((m+1)f_{m+2}-(m-4)f_{m+1})/7(m+1)(m+2)+((m+1)f_{m+2}-(m+3)f_{m+1})m^5/7n^6$。当对于 $n\leqslant m$，$b_n=\binom{n}{3}/n^p$ 和 $x_n=0$ 时，这个解对于 $n>m$ 是

$$x_n/(n+1)=(p-3)(p-2)/(p-6)(p+1)(n+1)^{p+1}$$
$$+12/7(p+1)(m+2)^{p+1}-12(m+1-p)^{6-p}/7(p-6)(n+1)^7$$

但当 $p=-1$ 时，$x_n/(n+1)=\frac{12}{7}(H_{n-1}-H_{m+2})+\frac{37}{49}+\frac{12}{49}(m+2)^7/(n+1)^7$。

当 $p=6$ 时，$x_n/(n+1)=-\frac{12}{7}(H_{n-6}-H_{m-5})/(n+1)^7+\frac{12}{49}/(m+2)^7+\frac{37}{49}/(n+1)^7$。

如同在习题 21～23 那样论证，我们发现头一个分划阶段现在分别地对（A，B，C）贡献（1，t，$N-1$），其中 t 如前一样定义但在做了习题 55 中的重新安排之后。在这个

新的假定之下，我们求得 $b_{s,N}=6\binom{s-2}{t}\binom{N-s-1}{t}\Big/N\binom{N-1}{s-1}$，因此在这个问题当中上面的递归式以下列方式出现：

量	对于$N\leqslant M$的值	对于$N>M$的$b_N/\binom{N}{3}$	对于$N>M$的解
A_N	0	1	$(N+1)\left(\frac{12}{7}/(M+2)\right)-1+O(N^{-6})$
B_N	0	$(N-4)/5$	$(C_N-3A_N)/5$
C_N	0	$N-1$	$(N+1)\left(\frac{12}{7}(H_{N+1}-H_{M+2})+\frac{37}{49}-\frac{24}{7}/(M+2)\right)+2+O(N^{-6})$
D_N	$N-H_N$	0	$(N+1)\left(1-\frac{12}{7}H_{M+1}/(M+2)-\frac{4}{7}/(M+2)\right)+O(N^{-6})$
E_N	$N(N-1)/4$	0	$(N+1)\left(\frac{6}{35}M-\frac{17}{35}+\frac{6}{7}/(M+2)\right)+O(N^{-6})$

类似地，$S_N=\frac{3}{7}(N+1)(5M+3)/(2M+3)(2M+1)-1+O(N^{-6})$。在习题55中程序总共的平均运行时间是 $53\frac{1}{2}A_N+11B_N+4C_N+3D_N+8E_N+9S_N+7N$；$M=9$ 的选择比 $M=10$ 的选择还要好些，并产生近似于 $10\frac{22}{35}N\ln N+2.11N$ 的平均时间。若以DIV代替SRB，则运行时间增加 $11A_N$ 并取 $M=10$。

57.

$$\sum_{n>N} n^{-1}(1-\alpha/N)^{n-N}=-N^{-1}+\sum_{k\geqslant 0}(N+k)^{-1}(1-\alpha/N)^k$$

的渐近级数可以通过限制 k 为 $O(N^{1+\epsilon})$ 来得到，把 $(1-\alpha/N)^k$ 展开成 $e^{-\alpha k/N}$ 乘 $(1-k\alpha^2/2N^2+\cdots)$，并使用欧拉求和公式，它成为 $e^{\alpha}E_1(\alpha)(1+\alpha^2/2N)-(1+\alpha)/2N+O(N^{-2})$。因此5.2.1-11的渐近值是 $N(\ln\alpha+\gamma+E_1(\alpha))/\alpha+(1-e^{-\alpha}(1+\alpha))/2\alpha+O(N^{-1})$〔对于 $\alpha=1$，2，10，N 的系数分别是0.7966，0.6596，0.2880〕。

注意：由习题43，$\ln\alpha+\gamma+E_1(\alpha)=\int_0^{\alpha}(1-e^{-t})t^{-1}dt$。

5.2.3 节

1．否；但使用∞的方法（正如恰在算法S之前所描述的那样）是稳定的。

2．如果我们从较高的下标到较低的下标，来扫描在内存中顺序地存放的一个线性表，则这种遍历方法通常都稍微快些，因为判断一个下标是否为0通常比起判断它是否超过 N 更容易些（由于同一原因，步骤S2的查找从 j 往下运行到1；但见习题8！）

3．(a) 对于 $Na_2\cdots a_{N-1}a_1$，$a_1Na_3\cdots a_{N-1}a_2$，…，$a_1a_2\cdots a_{N-2}Na_{N-1}$，$a_1\cdots a_{N-1}N$ 等输入，出现排列 $a_1\cdots a_{N-1}N$。(b) 如同在1.2.10节中所示那样，在步骤S2的头一次迭代期间，极大值改变的平均次数是 H_N-1〔因此，B_N 可以从等式1.2.7-8求得〕。

4．如果输入是{1，2，…，N}的一个排列，则在步骤S3中 $i=j$ 的次数恰比排列中的轮换个数少1（其实，不难证明，步骤S2和S3只不过从它的轮换中撤销元素 j；因

此S3仅当j是它的轮换中最小元素时才什么也不干)。由等式1.3.3-21，平均说来，我们可以节省步骤S3的$N-1$次执行中的H_N-1次。

于是，在步骤S3之前插入一个额外的判断"$i=j$?"是低效的。然而我们可以不去判断i和j的值，而是稍微地加长S2的程序，重复一部分代码，使得如果在查找极大值期间初始的猜测K_j不改变，则决不会遇到S3；这将使程序S稍微快一点。

5. $(N-1)+(N-3)+\cdots=\lfloor N^2/4\rfloor$。

6. (a) 在步骤S3中如果$i\neq j$，则该步使反序数减少$2m-1$，其中m比$K_{i-1}\cdots K_{j-1}$诸键中那些处于K_i和K_j之间键的个数大1；显然m不少于上一个步骤S2对B的贡献。现在应用习题4的观察，把轮换同条件$i=j$联系起来。(b) 通过逐次的交换反序的相邻元素，每个排列均可从N，…2，1得到(以相反的顺序应用把排列排成递减次序的交换)。每个这样的操作使I减1，并且使C变化±1。因此，没有任何排列有$I-C$的一个值，这个值超过$N\cdots 2\ 1$的相应值〔由习题5，不等式$B\leqslant\lfloor N^2/4\rfloor$是最好的〕。

7. 〔当$N=5$时，母函数是$\frac{1}{120}(1+4z+15z^2+30z^3+37z^4+26z^5+7z^6)$。〕

8. 假定我们已经记住了$\max(K_1,\ \cdots,\ K_{i-1})$，则可以在位置$K_i$处开始步骤S2的下一次迭代。记住所有这些辅助信息的一个方法是使用一个链接表$L_1\cdots L_N$，使得只要K_k是黑体的，则K_{L_k}是上一个黑体元素；$L_1=0$。〔我们以某些冗余的比较为代价，也可以减少所用的辅助存储。〕

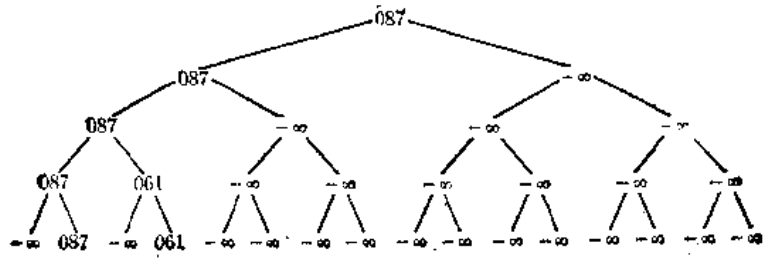
下列的MIX程序使用地址修改，使得内循环加快。rI1$\equiv j$，rI2$\equiv k-j$，rI3$\equiv i$，rA$\equiv K_i$。

01	START	ENT1	N	1	$j\leftarrow N$
02		STZ	LINK+1	1	
03		JMP	9F	1	
04	1H	ST1	6F(0:2)	$N-D$	修改循环中的地址
05		ENT4	INPUT,1	$N-D$	
06		ST4	7F(0:2)	$N-D$	
07		ENT4	LINK,1	$N-D$	
08		ST4	8F(0:2)	$N-D$	
09	7H	CMPA	INPUT+J,2	A	〔修改地址〕
10		JGE	*+4	A	如果$K_i\geqslant K_k$则转移
11	8H	ST3	LINK+J,2	$N+1-C$	否则$L_k\leftarrow i$〔修改地址〕
12	6H	ENT3	J,2	$N+1-C$	$i\leftarrow k$〔修改地址〕
13		LDA	INPUT,3	$N+1-C$	
14		INC2	1	A	$k\leftarrow k+1$
15		J2NP	7B	A	如果$k\leqslant j$则转移
16	4H	LDX	INPUT,1	N	
17		STX	INPUT,3	N	$R_i\leftarrow R_j$
18		STA	INPUT,1	N	$R_j\leftarrow$以前的R_i
19		DCE1	1	N	$j\leftarrow j-1$

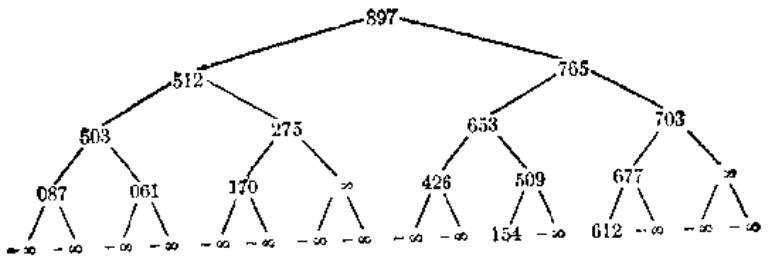
20		ENT2	0,3	N	rI2←i
21		LD3	LINK,3	N	$i \leftarrow L_i$
22		J3NZ	5F	N	如果 $i>0$ 则 k 将在 i 处开始
23	9H	ENT3	1	C	否则 $i \leftarrow 1$
24		ENT2	2	C	k 将在 2 处开始
25	5H	DEC2	0,1	$N-1$	
26		LDA	INPUT,3	$N+1$	rA←K_i
27		J2NP	1B	$N+1$	如果 $k \leqslant j$ 则转移
28		J1P	4B	$D+1$	如果 $j>0$ 则转移

9. $N-1+\sum_{N\geqslant k\geqslant 2}((k-1)/2-1/k)=\frac{1}{2}\binom{N}{2}+N-H_N$ [C 和 D 的平均值分别为 H_N+1 和 $H_N-\frac{1}{2}$；因此这个程序的平均运行时间为 $(1.25N^2+31.75N-15H_N+14.5)u$] 程序H要好得多。

10.



11.



12. 对于每个分枝节点中的 $-\infty$ 一次，共 2^n-1 次。

13．如果$K \geqslant K_{r+1}$，则当$j=r$时步骤H4可以转到步骤H5(除非$K_r < K_{r+1}$，步骤H5什么也不干，这时步骤H6无论如何将转到H8)。为确保在整个算法中$K \geqslant K_{r+1}$，我们可以以$K_{N+1} \leqslant \min(K_1, \cdots, K_N)$开始；在步骤H2中不置$R_r \leftarrow R_1$，而置$R_{r+1} \leftarrow R_{N+1}$和$R_{N-1} \leftarrow R_1$；在$r=1$之后还置$R_2 \leftarrow R_{N-1}$(这项技巧既不加快这个算法也不使程序有任何缩短)。

14．当插入一个元素时，给它一个键，这个键小于（或大于）所有以前指定的键，以达到一个简单队（或栈）的效果。

15．为了有效起见，下列的解是有一点技巧的，它避免了3的所有倍数〔CACM 10 (1967)，570〕。

a）置$p[1] \leftarrow 2$，$p[2] \leftarrow 3$，$k \leftarrow 2$，$n \leftarrow 5$，$d \leftarrow 2$，$r \leftarrow 1$，$t \leftarrow 25$，而且在优先队中置（25，10，30）。（在这个算法中，$p[i]$＝第i个质数；k＝至今为止找到的质数个数；n＝质数候选者；d＝到下个候选者的距离；r＝在这个队中元素的个数；$t=p[r+2]^2$，即我们将对之增加r的下一个n。队中项的形式为（u，v，$6p$），其中p是u，$v=2p$或$4p$的最小质因子，而且$u+v$不是3的一个倍数。）

b）设（q, q', q''）是有最小的头一个分量的队元素。在队中以（$q+q'$，$q''-q'$，q''）代替它（这表示必须加以排除的$q''/6$的下一倍数。）若$n>q$，重复这一步骤直到$n \leqslant q$为止。

c）如果$n>N$，则结束本算法。否则，若$n<q$，则置$k \leftarrow k+1$，$p[k] \leftarrow n$，$n \leftarrow n+d$，$d \leftarrow 6-d$，并重复这一步骤。

d）（现在$n=q$不是质数。）如果$n=t$，则置$r \leftarrow r+1$，$u \leftarrow p[r+2]$，$t \leftarrow u^2$，并且按照$u \bmod 3=2$或$u \bmod 3=1$把（$t, 2u, 6u$）或（t，$4u$，$6u$）插入队中。

e）置$n \leftarrow n+d$，$d \leftarrow 6-d$，并返回（b）。

于是这个计算开始如下：

队的内容	找到的质数
(25, 10, 30)	5, 7, 11, 13, 17, 19, 23
(35, 20, 30) (49, 28, 42)	29, 31
(49, 28, 42) (55, 10, 30)	37, 41, 43, 47
(55, 10, 30) (77, 14, 42) (121, 22, 66)	53

如果把队保持为一个堆，则我们可以在$O(N \log_2 N)$步内求出所有$\leqslant N$的质数；这个堆集的长度至多是$\leqslant \sqrt{N}$的质数的个数。伊拉托斯森斯的筛（如同在习题4.5.4-8中实现的那样）是需要相当多随机存取存储的一个$O(N \log_2 \log_2 N)$方法。

16．步骤1．置$K \leftarrow$有待插入的键；$j \leftarrow n+1$。

步骤2　置$i \leftarrow \lfloor j/2 \rfloor$。

步骤3　如果$i=0$或$K_i \geqslant K$，则置$K_j \leftarrow K$并且终止此算法。

步骤4　置$K_j \leftarrow K_i$，$j \leftarrow i$并返回步骤2。

17．文件1 2 3在算法H中进入堆3 2 1，但在习题16中进入堆3 1 2。（注：后一个建立堆的方法有阶为$N \log N$的最坏情况；但是经验测试已表明，在建立堆的期间，对于随机输入，步骤2的迭代次数仅大约为$2.27N-32$。）

18．删去步骤H6，并以下述步骤代替H8：

H8′．〔向后移动〕 置 $j \leftarrow i$，$i \leftarrow \lfloor j/2 \rfloor$ 或 $j=1$。

H9′．〔K合适吗？〕如果$K \leqslant K_i$，则置$R_i \leftarrow R$并返回到H2。否则置$R_j \leftarrow R_i$，并返回H8′。 这个方法实质上与习题 16 相同，但堆具有不同的开始位置。文件的实际修改是和算法H中的一样的。关于这个方法的经验测试表明，在选择阶段期每过筛一次，$K_j \leftarrow K_i$出现的次数分别以概率（.848，.135，.016）为（0，1，2）。这个方法使程序稍微长些，但是把它的渐近速度改进成为 $13N\log_2 N+O(N)$。把一个变址寄存器的值取半的指令在这儿很有用。

19．如同在习题 18 的修正的过筛算法那样进行，且$K=K_N$，$l=1$、$r=N-1$，并在步骤H3 中以一个给定的 j 值开始。

20．对于 $0 \leqslant k \leqslant n$，其二进表示为 $(b_n \cdots b_k a_1 \cdots a_q)_2 (q \geqslant 0)$ 的$\leqslant N$的正整数的个数显然为 $\sum_{0 \leqslant q \leqslant k} 2^q + (b_{k-1} \cdots b_0)_2 + 1 = (1b_{k-1} \cdots b_0)_2$。

21．设 $j=(c_r \cdots c_0)_2$ 是在范围 $\lfloor N/2^{k-1} \rfloor = (b_n \cdots b_{k+1})_2 < j < (b_n \cdots b_k)_2 = \lfloor N/2^k \rfloor$ 中。则 s_j 是其二进表示形式为 $(c_r \cdots c_0 a_1 \cdots a_q)_2 (q \geqslant 0)$ 的$\leqslant N$的正整数的个数，即是 $\sum_{0 \leqslant q < k} 2^q = 2^{k+1}-1$。因此大小为 $2^{k-1}-1$ 的非特殊的子树的个数为

$$\lfloor N/2^k \rfloor - \lfloor N/2^{k-1} \rfloor - 1 = \lfloor (N-2^k)/2^{k+1} \rfloor$$

〔对于后一恒等式，以 $n=2$ 和 $x=N/2^{k+1}$使用习题 1.2.4-38 中的重复律。〕

22．在 $l=1$ 之前，五种可能性是 5 3 4 1 2，3 5 4 1 2，4 3 5 1 2，1 5 4 3 2，2 5 4 1 3。这些可能性的每一种$a_1a_2a_3a_4a_5$在 $l=2$ 之前导致三种可能的排列 $a_1a_2a_3a_4a_5$，$a_1a_4a_3a_2a_5$，$a_1a_5a_3a_4a_2$。

23．（a）在B次迭代之后，$j \geqslant 2^B l$；因此 $2^B l \leqslant r$。

（b）$\sum_{1 \leqslant l \leqslant N} \lfloor \log_2(N/l) \rfloor = (\lfloor N/2 \rfloor - \lfloor N/4 \rfloor) + 2(\lfloor N/4 \rfloor - \lfloor N/8 \rfloor) + 3(\lfloor N/8 \rfloor - \lfloor N/16 \rfloor) + \cdots = \lfloor N/2 \rfloor + \lfloor N/4 \rfloor + \lfloor N/8 \rfloor + \cdots = N - \nu(N)$，这里 $\nu(N)$ 是N的二进表示中 1 的个数。又由习题 1.2.4-42，我们有 $\sum_{1 \leqslant r \leqslant N} \lfloor \log_2 r \rfloor = \lfloor N\log_2 N \rfloor - 2^{\lfloor \log_2 N \rfloor + 1} + 2$。由定理H我们知道，关于$B$的这个上限在堆的建立阶段是最好的。而且注意这样一点是有趣的，就是有唯一的一个包含诸键$\{1, 2, \cdots, N\}$的堆，使得在算法H的整个选择阶段K恒等于 1（例如，当$N=7$，即堆是 7 5 6 2 4 3 1 时；不难从N过渡到$N+1$）。这个堆给出堆排序的选择阶段B的极大值（连同D的极大值$\lfloor N/2 \rfloor$），所以对于整个排序，B的最好的上界是$N-\nu(N)+N\lfloor \log_2 N \rfloor - 2^{\lfloor \log_2 N \rfloor + 1} + 2$。

24．$\sum_{1 \leqslant k \leqslant N} \lfloor \log_2 k \rfloor^2 = (N+1-2^n)n^2 + \sum_{0 \leqslant k < n} k^2 2^k = (N+1)^2 n^2 - 6 - (2n-3)2^{n+1}$，其中 $n=\lfloor \log_2 N \rfloor$（参考习题 4.5.2-23）；因此最后过筛的方差是 $\beta_N = ((N+1)n^2 - (2n-3)2^{n+1} - 6)/N - ((N+1)n+2-2^{n+1})^2/N^2 = O(1)$。

B'_N 的标准离差是 $(\sum_{s \in M_N} \beta_s)^{1/2} = O(\sqrt{N})$。

25．过筛是“均匀”的，而且每个比较$K_j : K_j+1$有$\frac{1}{2}$的概率得出$<$。在这种情况

下对于C的平均贡献仅仅是对于A和B的平均贡献之和的一半，即是$((2n-3)2^{n-1}+\binom{n}{2})/(2^{n+1}-1)$。

26. (a)$\left(\frac{10}{25}+\frac{1}{2}+1\frac{3}{9}+\frac{1}{2}+1\frac{1}{2}+1\frac{2}{5}+2\frac{1}{2}+\frac{1}{2}+1\frac{1}{2}+1\frac{1}{2}+2\frac{1}{2}+1\frac{1}{2}+2+2+3+0+1+1+2+1+2+2+3+1+2+2\right)/26$。 (b) $\left(\sum_{1\leqslant k\leqslant N}\nu(k)-N+\frac{1}{2}\lfloor N/2\rfloor-\frac{1}{2}n+\sum_{1\leqslant k<n}\min(\alpha_{k-1},\alpha_k-\alpha_{k-1}-1)/(\alpha_k-1)\right)/N$，这里$\nu(k)$是在$k$的二进表示中二进位1的个数，且$\alpha_k=(1b_k\cdots b_0)_2$。如果$N=2^{e_1}+2^{e_2}+\cdots+2^{e_t}$，且$e_1>e_2>\cdots>e_t\geqslant 0$，则可以证明$\sum_{0\leqslant k\leqslant N}\nu(k)=\frac{1}{2}((e_1+2)2^{e_1}+(e_2+4)2^{e_2}+\cdots+(e_t+2t)2^{e_t})+t-N$。

27. 一般说来，兰伯特（Lam bert）级数$\sum_{n\geqslant 1}a_nx^n/(1-x)^n=\sum_{N\geqslant 1}(\sum_{d/N}a_d)x^N=\sum_{m\geqslant 1}x^{m^2}(a_m+\sum_{k\geqslant 1}(a_m+a_{m+k})x^{km})$。

〔注意当$a_n=n$和$x=\frac{1}{2}$时，我们得到关系

$$\beta=\sum_{n\geqslant 1}\frac{1}{2^n-1}=\sum_{n\geqslant 1}2^{-n^2}\left(n\left(\frac{2^n+1}{2^n-1}\right)+\frac{2^n}{(2^n-1)^2}\right)=2$$

74403 38887 59488…

这个常数在（20）中出现，其中我们有$B'_N\sim(\beta-2)N$和$C'_N\sim\left(\frac{1}{2}\beta-\frac{1}{4}\alpha-\frac{1}{2}\right)N$。〕

28. 节点k的儿子是节点$3k-1$、$3k$和$3k+1$。类似于程序H的一个MIX程序花费近似于$21\frac{2}{3}\log_3N\approx 13.7\log_2N$个时间单位。利用习题18的思想，这个数降低成为$18\frac{2}{3}N\log_3N\approx 11.8N\log_2N$，尽管除以3将增加一个很大的$O(N)$项。

31. （J. 埃迪霍弗（J. Edighoffer）给出的解）设A是使得$A[2\lfloor i/2\rfloor]\leqslant A[2i]$和$A[2\lfloor i/2\rfloor-1]\geqslant A[2i-1]$（$1<i\leqslant n$）的$2n$个元素的数组；其次我们要求对于$1\leqslant i\leqslant n$，$A[2i-1]\geqslant A[2i]$（对于所有的$i$，这后一条件成立的充要条件是，它对于$n/2<i\leqslant n$成立，这是由于堆的结构所致）。这“孪生”的堆包含$2n$个元素，为处理奇数个数的元素，我们只须甩开一个元素。适当的修改这节中其它算法，可以用来保持孪生的堆，推算这些细节是有趣的。

32. 设P、Q指向给定的优先队；为了简洁起见，下列算法使用约定DIST(Λ)＝0。

M1.〔初始化〕 置R←Λ。

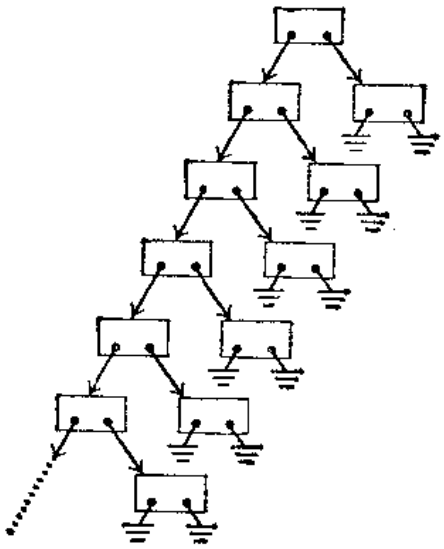
M2.〔表合并〕 如果Q＝Λ，则置D←DIST(P)并转到M3，如果P＝Λ则置P←Q，D←DIST(P)并转到M3，否则如果KEY(P)≥KEY(Q)，则置T←RIGHT(P)，RIGHT(P)←R，R←P，P←T并重复步骤M2。如果KEY(P)<KEY(Q)则置T←

RIGHT(Q)，RIGHT(Q)←R，R←Q，Q←T并重复步骤M2(这一步骤实质上合并给定树的两个“右表”，它暂时把向上的指针插入到诸RIGHT场中)。

M3.〔完成？〕 如果R＝A则结束此算法；P指向答案。

M4.〔修正诸DIST〕 置Q←RIGHT(R)。如果DIST(LEFT(R))＜D，则置D←DIST(LEFT(R))＋1，RIGHT(R)←LEFT(R)，LEFT(R)←P；否则置D←D＋1，RIGHT(R)←P。最后置DIST(R)←D，R←Q，P←R，并返回M3。

33．我们可能要在一株垂下一边的树中，引出大约一半的节点，诸如



34．设这数为L_N。不难证明$L_{m+n}\leqslant L_mL_n$，因此（取对数）$\lim L_n^{1/n}$存在（这是由D. A. 克拉尔纳（D. A. Klarner）提出的。E. 落格（E. Logg）所作的某些计算提示，$L_{n+1}/L_n\geqslant L_n/L_{n-1}$可能对所有的$n$成立）。

35．设删去的节点的DIST场是d_0，并设合并后的子树的DIST场是d_1。如果$d_0=d_1$，则我们根本不必向上走。如果$d_0>d_1$而且如果向上走n级，则P的诸祖宗的诸新DIST场必须分别是$d_1+1, d_1+2, \cdots, d_1+n$。如果$d_0<d_1$，则向上的通路只能向左拐。

36．最简单的是使用一个双重链接表来代替一般的优先队；当“使用”一个节点时，把它移动到这个表的一端，而且从另一端删去诸节点〔见6.1节中关于“自组织文件”的讨论〕。

5.2.4 节

1．以$i_1=\cdots=i_k=1$，$j=1$开始。重复地找$\min(x_{1i_1}, \cdots, x_{ki_k})=x_{ri_r}$，并置$z_j\leftarrow x_{ri_r}$，$j\leftarrow j+1$，$i_r\leftarrow i_{r+1}$(在这种情况下，$x_{i(m_i+1)}=\infty$的使用是一项决定性的妙着)。

当k相当大时，最好是象在5.2.3节中所讨论的那样，在适合于重复选择的一株树结构中保持键$x_1i_1, \cdots x_{ki_k}$使得在每次找到极小值之后仅仅需要作$\lfloor \log_2 k \rfloor$次比较，即可找出新的极小值。其实，这是在一个优先队中“最小者先出”原理的一个典型应用。诸键可以保持作一个堆，而且完全可以避免∞。进一步的讨论请看5.4.1节。

2．设C是比较的次数；我们有$C=m+n-S$，其中S是在步骤M4或M6中传送的

元素个数。容易看出对于 $1 \leqslant s \leqslant m+n$，$S \geqslant s$ 的概率是

$$q_s = \left(\binom{m+n-s}{m} + \binom{m+n-s}{n}\right) \Big/ \binom{m+n}{n}$$

当 $s > m+n$ 时，$q_s = 0$，因此 S 的均值是 $\mu_{mn} = q_1 + q_2 + \cdots = m/(n+1) + n/(m+1)$〔参考习题 3.4.2-5, 6〕，而且方差是 $\sigma^2_{mn} = (q_1 + 3q_2 + 5q_3 + \cdots) - \mu^2_{mn} = m(2m+n)/(n+1)(n+2) + (m+2n)n/(m+1)(m+2) - \mu^2_{mn}$。于是

$$C = (\min\ \min(m, n),\ \text{avc}\, m + n - \mu_{mn},\ \max m + n - 1,\ \text{dev}\, \sigma_{mn})$$

当 $m = n$ 时，这个平均值首先由 H. 纳格勒计算出来，CACM3(1960)，618-620；它渐近于 $2n - 2 + O(n^{-1})$ 并有 $\sqrt{2} + O(n^{-1})$ 的标准离差。于是 C 徘徊地接近于它的极大值。

3. **M2′**. 如果 $K_i < K'_j$ 则转到M3′；如果 $K_i = K'_j$，则转到M7′；如果 $K_i > K'_j$，则转到M5′。

M7′. 置 $K''_k \leftarrow K'_j$，$k \leftarrow k+1$，$i \leftarrow i+1$，$j \leftarrow j+1$。如果 $i > M$，转到M4′；否则如果 $j > N$ 转到M6′；否则返回M2′。

(对于算法M的其它步骤也作了适当的修改。此外，如果我们在这个文件的末尾插入人为的键 $K_{M+1} = K'_{N+1} = \infty$，则许多特殊情况就会消失。)

4. 随着时间的推移，出现在选择树的一个固定的内部节点处的元素序列，是通过合并出现于该节点的儿子处的元素序列得到的（5.2.3 节中的这个讨论是以选择最大元素为基础的，但是它同样也可以应用于颠倒次序的情况）。所以在树选择中进行的操作，实质上和合并中进行的那些操作相同，但它们以不同的序列而且使用不同的数据结构来实施。

习题 1 中指出了合并和树选择之间的另一个关系。注意，一些单元素文件的 N 路合并是一个选择排序；请把（A, B, C, D）的四路合并和先是（A, B），（C, D）然（AB, CD）的两路合并比较。

5. 在步骤N6 中，我们总有 $K_i < K_{i+1} \leqslant K_j$；在N10 中，$K_j < K_{j+1} < K_l$。

6. 2 6 4 10 8 14 12 16 15 11 13 7 9 3 5 1。在一次扫描之后我们有1 2 5 6 7 8 13 14 16 15 12 11 10 9 4 3(预期的两个下坡消失)。这种可能性是由 D. A. 贝尔说明的，Comp. J. 1(1958)，74。类似于此的怪例，使我们几乎没有希望来对于算法N作一个精细的分析。

7. $\lceil \log_2 N \rceil$，这里 $N > 1$（考虑 p 必须加倍多少次，直到它 $\geqslant N$）。

8. 如果 N 不是 $2p$ 的一个倍数，则在这趟扫描中有一个短路段，而且它总是接近于中间：设它的长度是 t，我们有 $0 \leqslant t < p$，步骤S12 处理的是短路段和空路段“合并”，或 $t = 0$ 的情况，否则，我们基本上有 $x_1 \leqslant x_2 \leqslant \cdots \leqslant x_p | y_t \geqslant \cdots \geqslant y_1$。如果 $x_p \leqslant y_t$ 则左边的路段首先穷尽，而且传送了 x_p 之后，步骤S6 将使我们达到步骤S13。另一方面如果 $t = 0$ 或 $x_p > y_t$，则右边将被人为地穷尽，但在步骤S3 中 $K_j = x_p$ 将决不 $< K_i$！于是，在所有的情况下，S6 将最终地使我们到达S13。

10. 例如，如果 $j \geqslant n$，则算法可以把 $x_{j+1}, \cdots, x_{j+m}$ 同 $x_{j+m+1} \cdots x_{j+m+n}$ 无冲突地合并到

一个阵列的$x_1 \cdots x_{m+n}$诸位置上。留点心我们就可以进一步利用这个思想，使得整个排序仅需要$N+2^{\lfloor \log_2 N \rfloor+1}$个单元。但是这个程序相对于算法S说来稍微复杂些〔Comp. J. 1 (1958)，75；也见Л. С. Ложинский Кибернетика 1，3 (1965)，58-62〕。

11．是的。例如，这可以通过考虑与习题4提出的树选择的关系来看到。但显然算法N和S不是稳定的。

12．置$L_0 \leftarrow 1$，$t \leftarrow N+1$；然后对于$p=1, 2, \cdots, N-1$作下列工作：

如果$K_p \leqslant K_{p+1}$，则置$L_p \leftarrow p+1$；否则置$L_t \leftarrow -(p+1)$，$t \leftarrow p$。最后置$L_t \leftarrow 0$，$L_N \leftarrow 0$，$L_{N+1} \leftarrow -|L_{N+1}|$。

(保持了稳定性。扫描次数是$\lceil \log_2 r \rceil$，其中r是输入中递增路段的个数；5.1.3节分析了r的精确分布。我们可以得出结论：当使用链接分配时，“自然”合并是比“直接”合并更可取的，但使用顺序分配时它是低劣的。

13．对于$N \geqslant 3$的运行时间是$(11A+6B+3B'+9C+2C''+4D+5N+9)u$，其中$A$是扫描次数；$B=B'+B''$是所实施的子文件合并操作的次数，这里$B'$是其中$p$子文件首先被穷尽的合并次数；$C=C'+C''$是所实施的比较次数，其中$C'$是满足$K_p \leqslant K_q$的比较的次数；$D=D'+D''$是当另一个子文件已被穷尽时在诸子文件中剩下的元素个数，其中D'是属于q子文件的元素的个数。

算法L对诸子文件进行一系列的合并，这些子文件的大小(m, n)，可如下确定：设在二进制记号下$N-1=(b_k \cdots b_1 b_0)_2$。对于$(m, n)=(2^j, 2^j)$，$0 \leqslant j < k$，有$(b_k \cdots b_{j+1})_2$个“通常的”合并；而且对于$0 \leqslant j \leqslant k$，每当$b_j=1$时，对于$(m, n)=(2^j, 1+(b_{j-1} \cdots b_0)_2)$有“特殊的”合并。例如当$N=14$时，有六个通常的(1，1)合并，三个通常的(2，2)合并，一个通常的(4，4)合并，以及处理大小为(1，1)，(4，2)，(6，8)的子文件的特殊的合并。

由此得出，不论输入分布如何，我们有$A=\lceil \log_2 N \rceil$，$B=N-1$，$C'+D''=\sum_{0 \leqslant j \leqslant k} b_j 2^j \left(1+\frac{1}{2}-j\right)$，$C''+b'=\sum_{0 \leqslant j \leqslant k} b_j \left(1+2^j\left(\frac{1}{2}-j+b_{j+1}+\cdots+b_k\right)\right)$；因此仅仅$B', C', D'$需要进一步分析。

如果对于算法L的输入是随机的，则每个合并操作都满足习题2的条件，而且同其它合并的特性无关；所以B', C', D'的分布是每个子文件合并时它们各自的分布的卷积。这样的合并的平均值是$B'=n/(m-n)$，$C'=mn/(n+1)$，$D'=n/(m+1)$。对于所有有关的(m, n)把这些加起来，就得精确的平均值。

当然，当$N=2^k$时，我们有最简单的情况：$B'_{\text{ave}}=\frac{1}{2}B$，$C'_{\text{ave}}=\frac{1}{2}C_{\text{ave}}$，$C+D=kN$，而且$D_{\text{ave}}=\sum_{1 \leqslant j \leqslant k}(2^{k-j} \cdot 2^j/(2^{j-1}+1)=\alpha' N+O(1)$，其中$\alpha'=\sum_{n \geqslant 0} 1/(2^n+1)=\alpha+\frac{1}{2}-\frac{1}{2}\sum_{n \geqslant 0} 1/(4^n-1)=1.26449\ 97803\ 48444\ 20919\ 13197\ 47255\ 49848\ 25577$—可以象在习题5.2.3-27中那样计算到很高的精确度。这个特殊情况是由A. 格利森〔未发表，1956〕和H. 纳格勒〔CACM3(1960)，618-620〕首先分析的。

在表3中，我们有$A=4$，$B'=6$，$B''=9$，$C'=22$，$C''=22$，$D'=10$，$D''=10$，总共

时间＝761 u（若象在习题 5.2.1-33 中那样改进程序 5.2.1 L 则仅花费 433 u，所以我们看到当N很小时，合并不是特别有效的）。

14．在习题 13 中置$D=B$以使C取得极大值。

15．设已知L_r等于p或q，构作步骤L_3，L_4，L_6的另一复份〔只需改动寄存器的名字即可。若从内循环中撤销赋值$s\leftarrow p$（或$s\leftarrow q$），还可以作出进一步的改进！例如，把行 20，21 改变成为“LD3 INPUT，1(L)”而且以rI3之值为P，rI1之值为S，以及L_r等于P来继续运行。根据（p，q，r）相对于（rI1，rI2，rI3）的不同排列，以及根据L_r的不同情况，我们可以构作内循环的 12 个复份，从而把平均运行时间减少到 $8N\log_2 N+O(N)$〕。

16．（这个结果将比算法 L 稍快，参考习题 5.2.3-28。）

17．把新的记录当作长度为 1 的子文件。如果最小的两个子文件有相同长度，即重复地合并之（得到的排序算法实质上和算法 L 相同，但在不同的相对时间合并诸子文件）。

18．是的，但它似乎是一项复杂的工作。已知的最简单的方法是使用下列巧妙的构造〔Доклоб Акаб. Hayk CCCP 186(1969)，1256-1258〕：设 $n\approx\sqrt{N}$，把这个文件分成为$m+2$个“段”$Z_1\cdots Z_mZ_{m+1}Z_{m+2}$，其中$Z_{m+2}$包含（$N \bmod n$）个记录，而每一个其它的段恰包含$n$个记录。把$Z_{M+1}$的诸记录同包含$R_M$的段进行交换；现在这个文件有$Z_1\cdots Z_mA$的形式，其中$z_1\cdots z_m$中的每一个恰巧包含$n$个排序好的记录，而且这里$A$是包含$s$个记录的一个辅助区域，其中$n\leqslant s<2n$。

找出具有最小前导元素的段，而且把整个该段同z_1交换（这花费$O(m+n)$个操作），然后找出具有次小前导元素的段，而且把它同z_2进行交换，等等。最后，通过$O(m(m+n))=O(N)$个操作，我们重新安排了m个段，使得它们的前导元素是有序的。而且，根据对于该文件原来的假定，在$z_1\cdots z_m$中的每个键现在都有少于n个反序。

我们利用下列技巧，可以合并Z_1和Z_2：把Z_1同A的头n个元素A'进行交换；然后以通常方式合并Z_2和A'，但被输出时与Z_1Z_2的元素相交换。例如，如果$n=3$，且$x_1<y_1<x_2<y_2<x_3<y_3$，则我们有

	段 1	段 2	辅助区域
初始值的内容：	$x_1x_2x_3$	$y_1y_2y_3$	$a_1a_2a_3$
交换z_1：	$a_1a_2a_3$	$y_1y_2y_3$	$x_1x_2x_3$
交换x_1：	$x_1a_2a_3$	$y_1y_2y_3$	$a_1x_2x_3$
交换y_1：	$x_1y_1a_3$	$a_2y_2y_3$	$a_1x_2x_3$
交换x_2：	$x_1y_1x_2$	$a_2y_2y_3$	$a_1a_3x_3$
交换y_2：	$x_1y_1x_2$	$y_2a_2y_3$	$a_1a_3x_3$
交换x_3：	$x_1y_1x_2$	$y_2x_3y_3$	$a_1a_3a_2$

（在辅助区域的第n个元素被交换过后，这个合并便完成了；这个方法通常会打乱辅助记录的原来位置。）

上述技巧用来合并Z_1和Z_2，然后Z_2和Z_3，…，Z_{m-1}和Z_m，总共需要$O(mn)=O(N)$个操作。由于没有多于n个反序的元素，这个文件的$Z_1\cdots Z_m$部分已经完全被排好序。

为了最后的“清理”，我们通过在$O(s^2)=O(N)$步内进行插入来对$R_{N+1-2s}\cdots R_N$排序；

这把S个最大元素带到A中。然后利用上述技巧，以辅助存储区域A（但整个地交换右和左，小于和大于的作用），合并$R_1 \cdots R_{N-2S}$和$R_{N+1-2S} \cdots R_{N-S}$。最后，我们通过插入来对$R_{N+1-S} \cdots R_N$排序。

19．我们可以把输入的列车编号，使得它们最后的排列依次为 1 2 …2^n；所以这实质上是一个排序问题。首先把2^{n-1}列车移过$n-1$个栈，以递减的次序来放置它们。然后把它们放置到第n个栈，使得最小的在顶上。然后使其它2^{n-1}列车通过$n-1$个栈，以递增的顺序放置它们，并使它们停在第n个栈之前。最后以显然的形式把两个序列合并在一起。

20．关于进一步的信息，请见R. E. Tarjan，JACM 19(1972)，341-346。

5.2.5 节

1．否。因为基数排序全然无效，除非在头一次扫描之后，分布排序是稳定的（但所提出的分布排序可用于首先最高位数字的基数排序方法中，为此需要推广基数交换，如同在正文最后一段中所提议的那样）。

2．恰恰相反，它是“反稳定的”；具有相同键的元素以相反的次序出现，因为头一遍扫描从R_N到R_1遍历诸记录（这证明是方便的，因为程序R的行28和行20把Λ同0等置起来，但是当然没有必要向后进行头一次扫描）。

3．如果堆0非空，则BOTM[0]已经指向头一个元素，如果它是空的，则我们置P←LOC(BOTM[0])而且过后使LINK(P)指向头一个非空堆的底。

4．当剩下偶数次扫描时，首先取堆0（由顶到底），接着取堆1，…，堆$(M-1)$；这个结果相对于至今所考虑的数字来说将是有序的。当剩下奇数次扫描时，首先取第$M-1$堆然后取第$M-2$堆，…，第0堆，这个结果相对于至今所考察的数字来说将是逆序的（这个规则看来是由E. H. 弗兰首先提出的〔*JACM*，3 (1956)，156，165-166〕。也见唐纳德·W. 约翰逊所写的论文〔*Library Resources and Technical sorvices*3 (1959)，300-310〕，他提出了把基数排序作为对图书馆卡片排列成序的一个手工方式。约翰逊说及，在伯克利的加利福尼亚大学图书馆发现，基数排序使得有可能仅仅花费4.8小时来把2000张卡片用手工排列成字母顺序，而用以前的一些方法，则要花费8.5小时。

5．(a) 在放置了第$N+1$个元素之后，有k个空堆的充分必要条件是(i)第$(N+1)$个元素以概率$((k+1)/M \times p_{MN(k+1)}$落入原来为空的堆中，或者(ii)它以$((M-k)/M)p_{MNk}$的概率落入一个非空的堆中。递归式

$$p_{M,N+1,k}=\frac{k+1}{M}\ p_{M,N,k+1}+\frac{M-k}{M}p_{M,N,k}$$

等价于所述的公式。(b) 通过对n用归纳法可证第n次导数满足$g_{M,N+1}^{(n)}(z)=(1-n/M)g_{M,N}^{(n)}(z)+((1-z)/M)g_{M,n}^{(n+1)}(z)$。置$z=1$，我们求出$g_{M,N}^{(n)}(1)=(1-n/M)^N M^n$，因为$g_{M0}(z)=z^M$。因此$\mathrm{mean}(g_{MN})=(1-1/M)^N M$，$\mathrm{var}(g_{MN})=(1-2/M)^N M(M-1)+(1-1/M)^N M-(1-1/M)^{2N}M^2$。

（注意，程序R中E的母函数是$(g_{MN}(z))^p$。）

6．把行04改变成为“ENT37”并把表R3SW和R5SW改变成为：

```
R3SW    LD2     KEY,    1(1:1)
        LD2     KEY,    1(2:2)
```

```
        LD2     KEY,      1(3:3)
        LD2     KEY,      1(4:4)
        LD2     KEY,      1(5:5)
        LD2     INPUT,    1(1:1)
        LD2     INPUT,    1(2:2)
        LD2     INPUT,    1(3:3)
R5SW    LD1     INPUT,    1(LINK)
            …… (再重复上一行6次)
        DEC1    1
```

通过在每处把“3”改成“8”，可以求得新的运行时间；对于$p=8$，它总计为$(11p-1)N+16pM+12p-4E+2$。

7．设R＝基数排序，RX＝基数交换。某些重要的类似性和差别是：RX从最高位数字进行到最低位，而R则从另一条途径进行。这两个方法都通过数字检查来进行排序，而不作键的比较。RX总有$M=2$（但见习题1）。R的运行时间几乎总是不变的，而RX对于数字的分布是敏感的。在两种情况下运行的时间都是$O(N\log_2 K)$，其中K是键的范围，但RX的比例常数是较高的；另一方面，当键按它们的前导数字来说是均匀分布时，无论K的大小如何，RX总有$O(N\log_2 N)$的平均运行时间。R需要链接场，而RX在“极小的存储”下运行。R的内循环更适合于“流水线”计算机。

8．在最后的扫描中，诸堆应以另一个次序被钩在一起；例如，如果$M=256$，则堆$(10000000)_2$首先来到，然后堆$(10000001)_2$，…，堆$(11111111)_2$，堆$(00000000)_2$，堆$(00000001)_2$，…，堆$(01111111)_2$。通过修改算法H，或（在表1中）通过在最后一次扫描时改变存储分配策略，即可更容易地进行挂钩顺序的改变。

9．如同在习题5.2.2-33中那样，我们可以首先把负的键同正的键分开；或者我们可以在头一次扫描时把键改成补码记号。或者，在最后一次扫描之后，可以把正键同负键分开，颠倒后者的顺序，但习题5.2.2-33的方法已不再能用了。

11．若没有头一次扫描，这个方法仍将成功地进行排序，因为（碰巧）503已在509之前。没有头两次扫描，反序数将是$1+1+0+0+0+1+1+1+0+0=5$。

12．在步骤M4(习题5.2-12)中交换R_k同$R[P]$之后，我们可以比较K_k同K_{k-1}。如果K_k较小，则把它同K_{k-2}，K_{k-3}，…，作比较，直到求出$K_k\geqslant K_j$为止。然后置$(R_{j+1}, \cdots, R_{k-1}, R_k)\leftarrow(R_k, R_{j+1}, \cdots, R_{k-1})$而不改变LINK场。设置一个人为的键$K_0$是方便的；它$\leqslant$在这个文件左边的所有其它键。

14．如果在习题5.1.3-20的意义下，卡片原来的排列需要读k次，而且如果对于每次扫描我们使用m个堆，则必须至少进行$\lceil\log_m k\rceil$次扫描（考虑从一个已排好序的卡片组到原来的卡片组的逆过程；每次扫描至多使读的次数增加一个因子m）。给定的排列需要4次递增的读，10次递减的读，所以递减的次序需要具有两个堆的4次扫描或者具有三个堆的3次扫描。

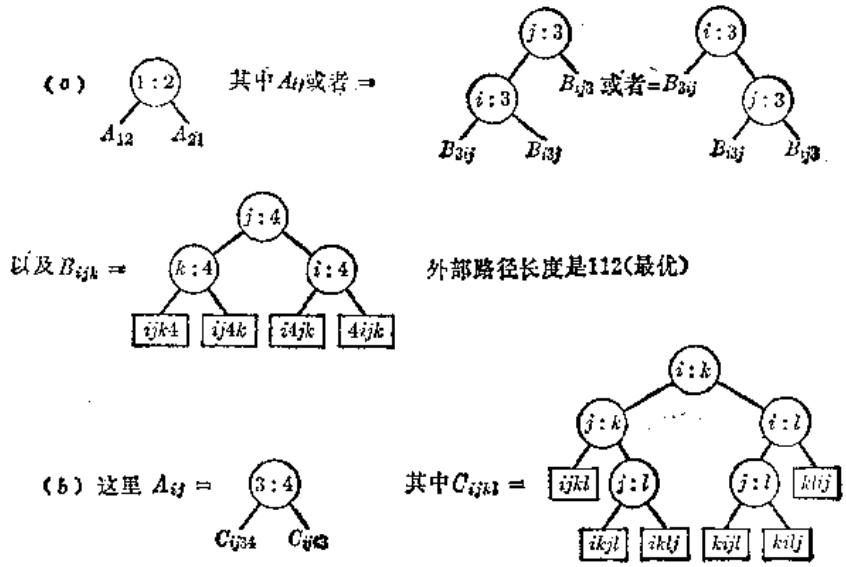
反之，这个最优的扫描次数可被达到；根据卡片是在第n次被读的，重新把它们编号成为0到$k-1$，并且使用一个基数排序（在基数m下最低位数字先开头）〔参考*Martin Gardner*的*Sixth Book of Mathematical Games*(San Francisco: W.H.Freeman, 1971),

111-112.]。

15. 设有 k 次读和 m 个堆，每次扫描时次序都被颠倒；如果按一个次序读 k 次，则在相反的次序下读的次数为 $n+1-k$。极小扫描次数或者是大于等于 $\log mk$ 的最小偶数，或者是大于等于 $\log_m(n+1-k)$ 的最小奇数(反向进行时，在一次扫描之后顶多有 m 次递减的读，在两次扫描之后，顶多有 m^2 次递增的读，等等)。这个例子可在 $\min(2,5)=2$ 次扫描中被排成递增次序，在 $\min(3,4)=3$ 次扫描中被排成递减次序，同时仅使用两个堆。

5.3.1 节

1. (a)



外部路径长度是112(最优)

2. 在习题 5.2.4-14 的记号下

$$\begin{aligned}I_r(n)-B(n)&=\sum_{1\le k\le t}((e_k+k-1)2^{e_k}-(e_1+1)2^{e_k})+2^{e_1+1}-2^{e_t}\\&=2^{e_1}-2^{e_t}-\sum_{2\le k\le t}(e_1-e_k+2-k)2^{e_k}\\&\ge 2e^1-(2^{e_1-1}+\cdots+2^{e_1-t+1}+2^{e_t})\ge 0\end{aligned}$$

当且仅当对于某个 $k>j\ge 0$ 有 $n=2^k-2^j$ 时，等式成立。

3. 当 $n>0$ 时，最小键恰恰出现 k 次的结果数为 $\binom{n}{k}P_{n-k}$。于是，对于 $n>0$，$2P_n=\sum_k\binom{n}{k}P_{n-k}$，而且我们有 $2P(z)=e^zP(z)+1$ (参考等式 1.2.9-10)。

另一个证明从这样一个事实得出：即 $P_n=\sum_{k\ge 0}\left\{{n \atop k}\right\}k!$，由于 $\left\{{n \atop k}\right\}$ 是把 n 个元素划分成 k 个非空部分的方式数，而且这些部分可以 $k!$ 种方式排列。于是，由等式 1.2.9-23，$\sum_{n\ge 0}P_nz^n/n!=\sum_{k\ge 0}(e^z-1)^k=1/(2-e^z)$。

还有另一个证明，或许最有趣，它得自于：我们以一种“稳定的”方式，把这些元素排成序列，使得当且仅当 $K_i<K_j$ 或 ($K_i=K_j$ 且 $i<j$) 时 K_i 居于 K_j 之前。在所有 P_n 个结果当中，如果排列 $a_n\cdots a_1$ 包含 k 个路段，则一个给定的安排 $K_{a_1}\cdots K_{a_n}$ 现在恰恰出现 2^{k-1} 次；因此 P_n 可以借助于欧拉数 $P_n=\sum_k\left\langle{n \atop k}\right\rangle 2^{k-1}$ 来表达。当 $z=2$ 时等式 5.1.3-20 给出了所求的结果。

这个母函数是由 A. 凯利〔*Phil. Mag.* 18(1859)374-378〕在枚举一个不精确定义的树

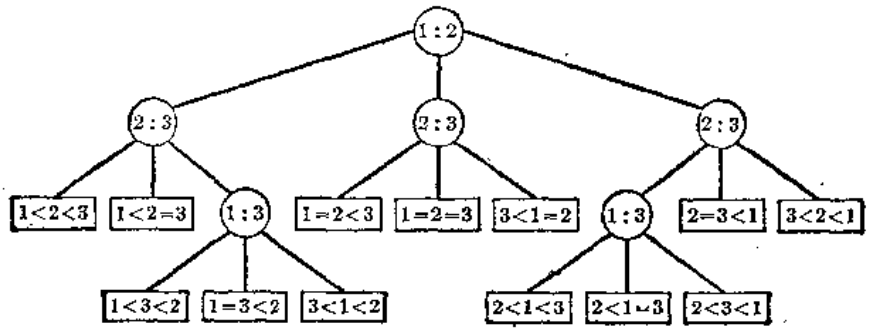
类时得到的。也见 J. Touchard, *Ann. Soc. Sci, Bruxelles* 53 (1933), 21-31。P_1, …, P_{14} 的一张表出现于 O. A. Gross, *AMM* 69(1962), 4-8; 格罗斯(Gross)给出了有趣的公式 $P_n=\sum_{k\geqslant 1}k^n/2^{1+k}$, $n\geqslant 1$。

4. 表示

$$2P(z)=\frac{1}{2}(1-i\cot(i(z-\ln 2)/2))$$
$$=\frac{1}{2}-\frac{1}{z-\ln 2}-\sum_{k\geqslant 1}\left(\frac{1}{z-\ln 2-2\pi ik}+\frac{1}{z-\ln 2+2\pi ik}\right)$$

产生一个收敛的级数 $P_n/n!=-\frac{1}{2}(\ln 2)^{-n-1}+\sum_{k\geqslant 1}\Re(\ln 2+2\pi jk)^{-n-1}$。

5.



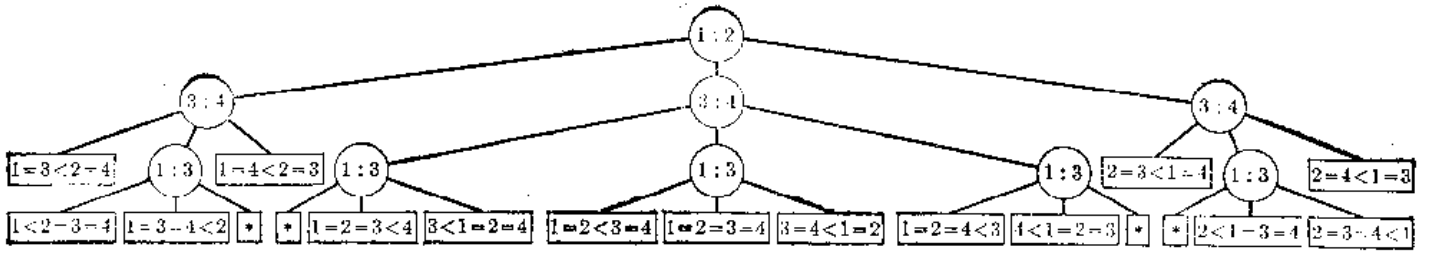
6. $S'(n)\geqslant S(n)$, 因为诸键可以全都不同, 于是我们必须证明 $S'(n)\leqslant S(n)$。给定一个在不同的键上花费 $S(n)$ 步的排序算法, 我们可以通过定义=分枝恒同于<分枝来消除冗余性, 从而构造一般情况的排序算法。当出现一个外部节点时, 我们知道所有的相等关系, 因为有 $K_{a_1}\leqslant K_{a_2}\leqslant\cdots\leqslant K_{a_n}$, 而且已经对 $1\leqslant i<n$ 明显地比较过 $K_{a_i}:K_{a_{i+1}}$。

M. 佩特森(M. Paterson) 发现, 如果键的多重集是 $\{n_1, \cdots, n_m\}$, 则比较的次数可减少成为 $n\log_2 n-\sum n_i\log_2 n_i+O(n)$; 见 *SIAM J. Computing* 5 (1976), 2。

9. 如果所有的键是相等的, 则我们至少需要 $n-1$ 次比较才能发现所有键相等这一事实。反之, $n-1$ 次总足够了, 因为在把 K_1 同所有其它键作了比较之后, 我们总能导出最后的排序。

10. 设 $f(n)$ 为所希望的函数, 并设 $g(n)$ 是当 $k>0$ 且恰有 k 个元素有已知的值(0 或 1)时为对 $n+k$ 个元素排序所需要的极小平均比较次数。则 $f(0)=f(1)=g(0)=0$, $g(1)=1$; 且对于 $n\geqslant 2$ 有 $f(n)=1+\frac{1}{2}f(n-1)+\frac{1}{2}g(n-2)$, $g(n)=1+\min\left(g(n-1), \frac{1}{2}g(n-1)+\frac{1}{2}g(n-2)\right)=1+\frac{1}{2}g(n-1)+\frac{1}{2}g(n-2)$(于是, 最好的策略是, 只要可能, 就来比较两个未知的元素)。由此得出对于 $n\geqslant 2$, $f(n)-g(n)=\frac{1}{2}(f(n-1)-g(n-1))$, 和对于 $n\geqslant 0$, $g(n)=\frac{2}{3}\left(n+\frac{1}{3}\left(1-\left(-\frac{1}{2}\right)^n\right)\right)$。因此答案是: 对于 $n\geqslant 1$, $\frac{2}{3}n+\frac{2}{9}-\frac{2}{9}\left(-\frac{1}{2}\right)^n-\left(\frac{1}{2}\right)^{n-1}$(这个精确公式可以同信息论的下限 $\log_3(2^n-1)\approx 0.6309\,n$ 相对照)。

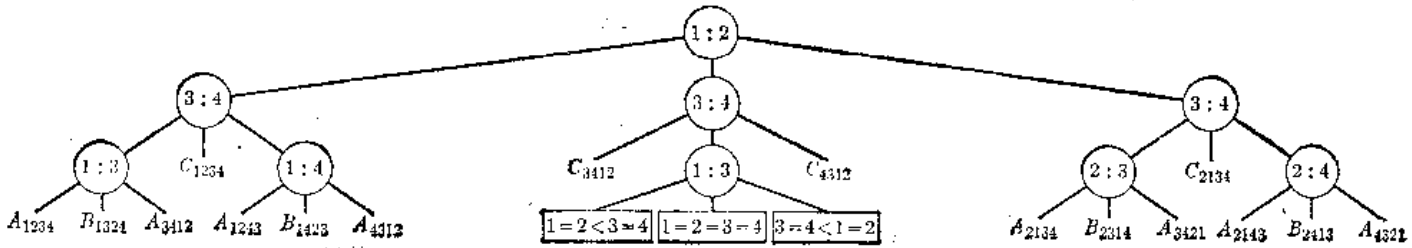
7.



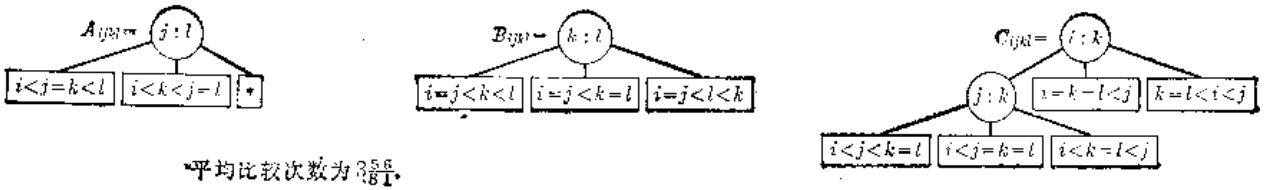
(* 表示一种不可能的情况)

平均比较次数为 $(2+3+3+2+3+3+3+6+3+3+3+2+3+3+2)/16 = 2\frac{3}{4}$.

8.



其中



平均比较次数为 $3\frac{56}{81}$.

11. 二叉插入证明，对于 $n \geqslant m$，$S_m(n) \leqslant B(m)+(n-m)\lceil \log_2(m+1)\rceil$。另一方面，$S_m(n) \geqslant \lceil \log_2 \Sigma_{1 \leqslant k \leqslant m}\binom{n}{k} k!\rceil$，而且这渐近于 $n \log_2 m$（参考等式 1.2.6-49）。

12. (a) 如果没有冗余的比较，则当实际上相等的键第一次被比较时，我们可以任意地指定一个次序，因为不能从以前所做的比较中导出次序。(b) 假定这株树对每个 0 和 1 的序列强排序；我们将证明，它对 $\{1, 2, \cdots, n\}$ 的每个排列也强排序。假若不然，于是有一个排列，对于这个排列，它宣称 $K_{a_1} \leqslant K_{a_2} \leqslant \cdots \leqslant K_{a_n}$，然而事实上对于某个 i，有 $K_{a_i} > K_{a_{i+1}}$。用 0 代替所有的 $< K_{a_i}$ 的元素，用 1 代替所有 $\geqslant K_{a_i}$ 的元素，由假设，当我们采取导致 $K_{a_1} \leqslant K_{a_2} \leqslant \cdots \leqslant K_{a_n}$ 的路径时，这个方法又应该能进行排序，矛盾。

13. 如果 n 为偶数，则 $F(n)-F(n-1)=1+F(\lfloor n/2\rfloor)-F(\lfloor n/2\rfloor-1)$，所以我们必须证明 $w_{k-1} < \lfloor n/2\rfloor \leqslant w_k$；由于 $w_{k-1}=\lfloor w_k/2\rfloor$ 这是明显的。如果 n 为奇数，则 $F(n)-F(n-1)=G(\lceil n/2\rceil)-G(\lfloor n/2\rfloor)$，所以我们必须证明 $t_{k-1} < \lceil n/2\rceil \leqslant t_k$；由于 $t_{k-1}=\lceil w_k/2\rceil$ 这是显然的。

14. 由习题1.2.4-42，和数为 $n\left\lceil \log_2\left(\frac{3}{4}n\right)\right\rceil-(w_1+\cdots+w_j)$，其中 $w_j < n \leqslant w_{j+1}$。后一个和数是 $w_{j-1}-\lfloor j/2\rfloor-1$。因此 $F(n)$ 可写成为 $n\left\lceil \log_2\left(\frac{3}{4}n\right)\right\rceil-\lfloor 2^{\lfloor \log_2(6n-1)/3\rfloor}+\left\lfloor -\frac{1}{2}-\log_2(6n)\right\rfloor$（以及许多其它方式）。

15. 如果 $\left\lceil \log_2\left(\frac{3}{4}n\right)\right\rceil=\log_2\left(\frac{3}{4}n\right)+\theta$，则 $F(n)=n\log_2 n-(3-\log_2 3)n+n(\theta+1-2^{\theta})+O(\log_2 n)$。如果 $\lceil \log_2 n\rceil=\log_2 n+\theta$，则 $B(n)=n\log_2 n-n+n(\theta+1-2^{\theta})+O(\log_2 n)$〔注意 $\log_2 n!=n\log_2 n-n/\ln 2+O(\log_2 n)$；$1/(\ln 2)\approx 1.443$；$3-\log_2 3\approx 1.415$〕。

17. $b_k < a_p < b_{k+1}$ 的情况数为

$$\binom{m-p+n-k}{m-p}\binom{p-1+k}{p-1}$$

而 $a_j < b_q < a_{j+1}$ 的情况数是

$$\binom{n-q+m-j}{n-q}\binom{q-1+j}{q-1}$$

18. 否，因为我们仅仅考虑在每个比较之下树的效率较低的分枝。较有效的分枝之一可以证明是更难于处理的。

20. 设 L 是其上出现有外部节点的极大的级，而设 l 是极小的这样的级。如果 $L \geqslant l+2$，则我们可以从级 L 撤销两个节点，并把它们放置在级 l 处的一个节点之下；这使外部路径长度减少 $l+2L-(L-1+2(l+1))=L-l-1 \geqslant 1$。反之，如果 $L \leqslant l+1$，就设在级 $l+1$ 上有 k 个外部节点而在级 l 上有 $N-k$ 个节点，这里 $0 < k \leqslant N$。由习题 2.3.4.5-3，$k2^{-l}+(N-k)2^{-l-1}=1$；因此，$N+k=2^{l+1}$。不等式 $N < 2^{l+1} \leqslant 2N$ 现在表明 $l=\lfloor \log_2 N\rfloor$；这定义了 k 并给出外部路径长度(33)。

21. 设 $r(x)$ 是 x 的右子树的根，我们需要 $\lceil \log_2 t(l(x))\rceil \leqslant \lceil \log_2 t(x)\rceil-1$ 和 $\lceil \log_2 t(r(x))\rceil \leqslant \lceil \log_2 t(x)\rceil-1$。头一个条件等价于 $2t(l(x))-t(x) \leqslant 2^{\lceil \log_2 t(x)\rceil}$

$-t(x)$，而第二个条件等价于 $t(x)-2t(l(x))\leqslant 2^{\lceil \log_2 t(x)\rceil}-t(x)$。

22. 由习题20，条件 $\lfloor \log_2 t(l(x))\rfloor$，$\lfloor \log_2 t(r(x))\rfloor \geqslant \lfloor \log_2 t(x)\rfloor -1$ 和 $\lceil \log_2 t(l(x))\rceil$，$\lceil \log_2 t(r(x))\rceil \leqslant \lceil \log_2 t(x)\rceil -1$ 是必要和充分的。如同在习题 21 中那样，我们可以证明它们等价于所述条件〔参考 Martin Sandelius，*AMM* 68(1961)，133-144〕。关于它的一个推广见习题33。

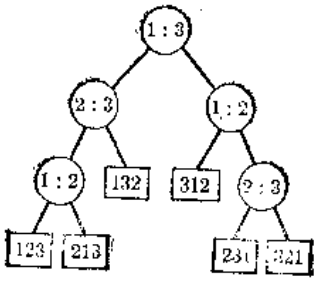
23. 多重表插入假设，键被均匀地分布在一个已知的范围内，所以它不是满足本节中所考虑的限制的“纯粹比较”的方法。

24. 首先如同对五个元素排序那样进行，直到进行了五次比较之后，我们达到(6)中的配置之一。在头三种情况下，再用两次比较完成对五个元素的排序，然后插入第六个元素 f。在后一种情况下，首先比较 $f:b$，把 f 插入到主链中，然后插入 c〔*Theorie des Questionnaires*，p.116。〕

25. 由于 $N=7!=5040$ 和 $q=13$，在级12上将有 $8192-5040=3652$ 个外部节点，且在级13上有 $5040-3152=1888$ 个外部节点。

26.（在迄今已知的最好方法中，外部路径长度为62416）。

27.



是用两次比较识别两个最经常的排列的唯一方式，不过头一次比较产生.27-.73的分裂。

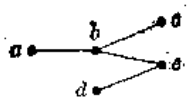
28. 伦阙（Lun Kwan）已经构造了一个其平均运行时间为 38.925 u 的 873 行的程序。它的极大运行时间为 43 u；后者看来是最优的，因为它是 7 次比较，7 次测试，6 次装入，5 次存储的时间。

29. 我们必须至少作 $S(n)$ 次比较，因为不可能知道一个排列是偶的还是奇的，除非已经作了足够的比较来唯一地确定它。因为我们可以假定已经作了足够的比较，来把问题缩小到取决于 a_i 是否小于 a_j（对于某个 i 和 j）的两种可能性；这两种可能性之一是偶的，另一种是奇的。〔另一方面，对于这个问题有一个 $O(n)$ 算法，这个算法简单地计算轮换的数目，而且全然不使用比较；见习题5.2.2-2。〕

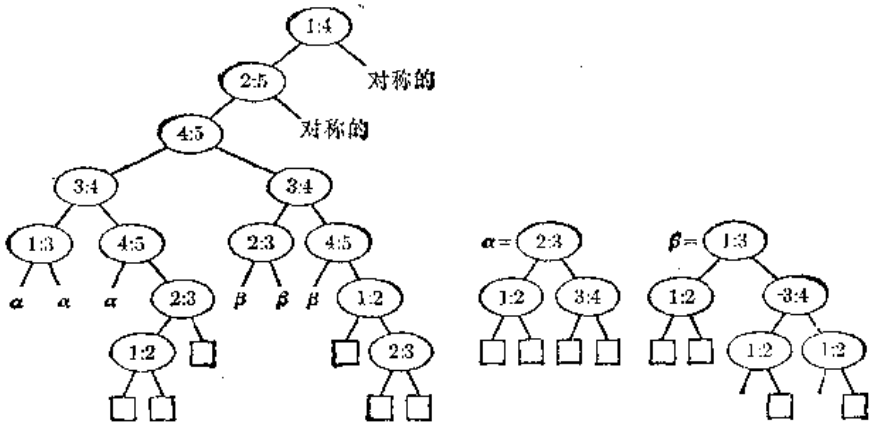
30. 由高度为 $S(n)$ 的最优比较树开始，从顶到底，重复地在标号为 $i:j$ 的节点的右子树中交换 $i\leftrightarrow j$。把这个结果解释为一株比较—交换树，每个终端节点定义一个唯一的排列，这个排列可以通过至多 $n-1$ 次另外的比较—交换（由习题 5.2.2-2）来进行排序。

〔比较—交换树的思想是由 T. N. 希巴德给出的。〕

31. 至少需要 8 个，因为高度为 7 的每一株树在 4 步之后，将在某个分枝产生（或它的对偶），其中 $a\neq 1$。这个配置不能用 3 次另外的比较/交换操作来完成排序。另一方



面，下列树达到了所希望的界(或许也是极小的平均比较/交换次数)：



33．可应用于阶为 x 和分辨率为 1 的任何树的简单操作，均可被应用来产生其加权路径长度不更大的另一株树，而且使得（a）对于某一个 k，所有外部节点都位于级 k 和 $k-1$ 上；（b）至多一个外部节点是非整数的，而且如果它出现，则它位于级 k 上。任何这样的树的加权路径长度都有所述的值，所以这必定是极小的。反之，如果(iv)和(v)对任何实值查找树成立，则通过归纳法有可能证明，加权路径长度有所述的值，因为一株树的加权路径长度可以借助于一个简单的公式，通过它的根的两个子树的加权路径长度来表达。

5.3.2 节

1．$S(m+n) \leqslant S(m)+S(n)+M(m, n)$。

2．在对称次序下为第 k 个内部节点对应于比较 $A_i:B_k$。

3．策略 $B(1, l)$ 不比策略 $A(1, l+1)$ 更好，而策略 $B'(1, l)$ 不比 $A'(1, l-1)$ 更好；因此我们必须解递归式

$$.M.(1, n)=\min_{1 \leqslant j \leqslant n} \max\left(\max_{1 \leqslant l \leqslant j}(1+.M.(1, l-1)), \max_{j \leqslant l \leqslant n}(1+.M.(1, n-l))\right), \quad n \geqslant 1 .M.(1, 0)=0$$

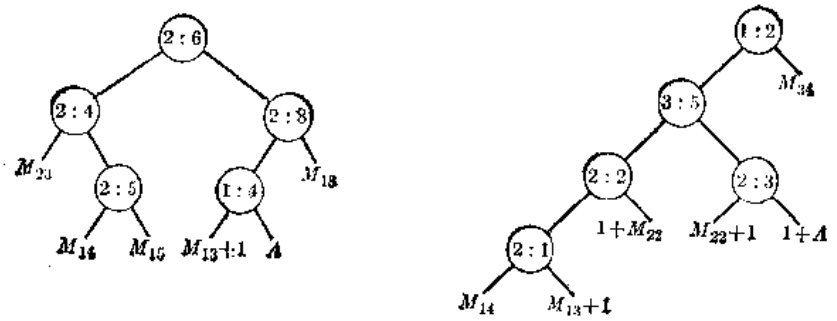
。不难验证 $\lceil \log_2(n+1) \rceil$ 满足这个递归式。

6．当 $j=i+1$ 时，除去 $i \leqslant 2$ 的情况外，可以使用策略 $A'(i, i+1)$。而当 $j \geqslant i+2$ 时，我们可以使用策略 $A(i, i+2)$。

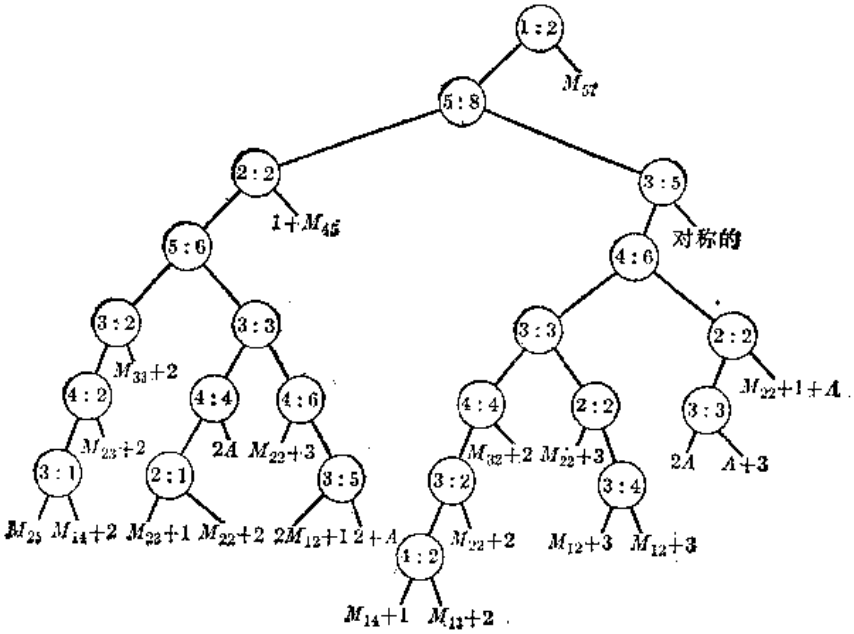
7．为了在 n 个其它元素当中插入 $k+m$ 个元素，可独立地插入 k 个元素及 m 个元素(当 k 和 m 很大时，本过程尚可改进，见习题19)。

8．在下列图式中，$i=j$ 表示比较 $A_i:B_j$，M_{ij} 表示用 $M(i, j)$ 步把元素 i 同元素 j 合并，而 Λ 表示用三步将型式 或 排序。

9.



10.



11．如同在提示中那样，设 $n=g_t$。我们可以假定 $t\geqslant 6$。不失一般性，可设 $A_2:B_j$ 为头一次比较。如果 $i>g_{t-1}$，则结果 $A_2<B_j$ 将需要 $\geqslant t$ 个另加的步骤。如果 $j\leqslant g_{t-1}$，则结果 $A_2<B_j$ 将不成问题，所以仅仅需要研究 $A_2>B_j$ 的情况，而且当 $j=g_{t-1}$ 时我们得到最多的信息。如果 $t=2k+1$，则可能要把 A_2 同 $>B_{g_{t-1}}$ 的 $g_t-g_{t-1}=2^{k-1}$ 个元素合并在一起，而把 A_1 同 g_{t-1} 个其它元素合并在一起，但这要求 $k+(k+1)=t$ 个另外的步骤。另一方面，如果 $n=g_t-1$，则我们可以把 A_2 同 $2^{k-1}-1$ 个元素合并在一起，然后把 A_1 同 n 个元素合并在一起，这共需 $(k-1)+(k+1)$ 个另外的步骤，因此 $M(2,g_t-1)\leqslant t$。

$t=2k$ 的情况要困难得多；注意 $g_t-g_{t-1}\geqslant 2^{k-2}$。在 $A_2>B_{g_{t-1}}$ 之后，假设我们比较 $A_1:B_j$。如果 $j>2^{k-1}$，则结果 $A_1<B_j$ 要求 $k+(k-1)$ 个另外的比较（太多）。如果 $j\leqslant 2^{k-1}$，则我们可以如同以前一样论证 $j=2^{k-1}$ 给出最多的信息。在 $A_1>B_{2^{k-1}}$ 之后，下一

个同 A_1 的比较也可以同 $B_{2^{k-1}+2^{k-2}}$ 进行，然后同 $B_{2^{k-1}+2^{k-2}+2^{k-3}}$ 比；因为 $2^{k-1}+2^{k-2}+2^{k-3}>g_{k-1}$，我们就剩下把 $\{A_1, A_2\}$ 同 $n-(2^{k-1}+2^{k-2}+2^{k-3})$ 个元素合并在一起了。当然，我们不必马上进行同 A_1 的任何比较；而代之以比较 $A_2:B_{n+1-j}$。如果 $j\leqslant 2^{k-3}$，则考虑 $A_2<B_{n+1-j}$ 的情况，如果 $j>2^{k-3}$，则我们考虑 $A_2>B_{n+1-j}$。后一种情况要求至少再有 $(k-2)+(k+1)$ 个步骤。继续进行，我们发现仅有的可能有成果的路线是 $A_2>B_{g_{k-1}}$，$A_2<B_{n+1-2^{k-2}}$，$A_1>B_{2^{k-1}}$，$A_1>B_{2^{k-1}+2^{k-2}}$，$A_1>B_{2^{k-1}+2^{k-2}+2^{k-3}}$，但是在这种情况下我们恰恰剩下了 g_{k+5} 个元素！反之，如果 $n=g_k-1$，则这路线是行得通的〔*Acta Informatica* 1(1971)，145-158.〕。

12. 头一个比较必须是 $\alpha:X_k(1\leqslant k\leqslant i)$，或者(对称地) $\beta:X_{n-k}(1\leqslant k\leqslant j)$。在前一种情况下，结果 $\alpha<X_k$ 要我们继续做 $R_n(k-1, j)$ 次另外的比较；结果 $\alpha>X_k$ 要我们继续解决 $\alpha<\beta$，$Y_1<\cdots<Y_{n-k}$，$\alpha<Y_{i-k+1}$，$\beta>Y_{n-k-j}$ 的排序问题，其中 $Y_r=X_{r+k}$。

13. 〔*Computers in Number Theory* (New York: Academic Press, 1971)，263-269。〕

14. 〔*SIAM J. Computing* 9/1980)，298-320。〕

15. 将 m 加倍直到它超过 n 为止，这需要 $\lfloor\log_2(n/m)\rfloor+1$ 次加倍。

16. 当它是一个以上时，为除去(2, 8)，(3, 6)，(3, 8)，(3, 10)，(4, 8)，(4, 10)，(5, 9)，(5, 10)之外的所有的(m, n)。

17. 假定 $m\leqslant n$ 且设 $t=\log_2(n/m)-\theta$，则 $\log_2\binom{m+n}{m}>\log_2 n^m-\log_2 m!\geqslant m\log_2 n-(m\log_2 m-m+1)=m(t+\theta)+m-1=H(m, n)+\theta m-\lfloor 2^\theta m\rfloor\geqslant H(m, n)+\theta_m-2^\theta m\geqslant H(m, n)-m$ (对于 $1\leqslant k<m$，不等式 $m!\leqslant m^m 2^{1-m}$ 由 $k(m-k)\leqslant(m/2)^2$ 推出)。

19. 首先合并 $\{A_1, \cdots, A_m\}$ 和 $\{B_2, B_4, \cdots, B_{2\lfloor n/2\rfloor}\}$，然后对于 $1\leqslant i\leqslant\lceil n/2\rceil$，我们必须把诸奇元素 B_{2i-1} 插入到诸 A 的 a_i 当中，其中 $a_1+a_2+\cdots+a_{\lceil n/2\rceil}\leqslant m$。后一操作对于每个 i 只要 $\leqslant a_i$ 个操作，所以至多 m 次另外的比较就可完成这个工作。

20. 应用(12)。

23. 对手保持一个其元素 x_{ij} 开始时全为 1 的 $n\times n$ 矩阵 X。当这个算法问及是否 $A=B_j$ 时，对手置 x_{ij} 成为 0，回答"否"，但如 X 的不变式恰恰变成 0 了，则回答"是"(这是必须的，免得此算法立即结束！)，并从 X 删去行 i 和列 j；得到的 $(n-1)\times(n-1)$ 矩阵将有一个非 0 的不变式。对手继续照此办理，直到只剩下 0×0 矩阵。

如果不变式要变成零，我们可以重新安排诸行与列使得 $i=j=1$ 并且使此矩阵的所有 1 全都在对角线上，但当 $x_{11}\leftarrow 1$ 时它的不变式仍变成零；这时对于所有的 $k>1$，我们必定有 $x_{1k}x_{k1}=0$。由此得出，当对手第一次回答"是"时至少已删去 n 个零，而当第二次回答"是"时已删去 $n-1$ 个零，等等。仅仅在对非冗余的问题接受了 n 个"是"的回答之后，以及在提出了至少 $n+(n-1)+\cdots+1$ 个问题之后，这个算法才会结束〔*JACM* 19 (1972)，649-659〕。类似的论证表明，当 $\|A\|=m\leqslant n=\|B\|$ 时需要 $n+(n-1)+\cdots+(n-m+1)$ 个问题才能确定 $A\subseteq B$。

5.3.3 节

1. 游戏者11失利于05，所以知道13比05，11，12要坏。

2．设 x 是第 t 个最大者，且 S 是使得所作比较不足以证明 $x<y$ 或 $y<x$ 的所有元素 y 的集合。存在同所作的所有比较相一致的一些排列，其中 S 的所有元素都小于 x；因为我们可以约定 S 的所有元素都小于 x 并把得到的偏序嵌入到一个线性序当中。类似地存在有一些一致的排列，其中 S 的所有元素都大于 x。因此我们不知道 x 的秩，除非 S 是空的。

3．一个对手可以认为头一个比较的失利者是所有游戏者当中最坏的〔R．W．弗洛伊德已使用一个类似的方法证明，表 1 中的值对所有 $n\leqslant 7$ 是准确的〕。

4．至少有 n^t 个结果。

5．事实上，由习题 2，$W_t(n)\leqslant V_t(n)+S(t-1)$。

6．命 $g(l_1, l_2, \cdots, l_m)=m-2+\lceil\log_2(2^{l_1}+2^{l_2}+\cdots+2^{l_m})\rceil$，并假定当 $l_1+l_2+\cdots+l_m+2m<N$ 时，$f=g$。我们将证明，当 $l_1+l_2+\cdots+l_m+2m=N$ 时，$f=g$。可以假定 $l_1\geqslant l_2\geqslant\cdots\geqslant l_m$。只有几种可能的方法来作头一次比较：

策略 $A(j, k)$，对于 $j<k$　比较组 j 的最大元素和组 k 的最大元素，这给出了关系

$$\begin{aligned} f(l_1, \cdots, l_m) &\leqslant 1+g(l_1, \cdots, l_{j-1}, l_{j+1}, l_{j+1}, \cdots, l_{k-1}, l_{k+1}, \cdots, l_m) \\ &= g(l_1, \cdots, l_{j-1}, l_j, l_{j+1}, \cdots, l_{k-1}, l_j, l_{k+1}, \cdots, l_m) \\ &\geqslant g(l_1, \cdots, l_m) \end{aligned}$$

策略 $B(j, k)$，对于 $l_k>0$　比较组 j 的最大元素同组 k 的小元素之一，这给出关系

$$f(l_1, \cdots, l_m)\leqslant 1+\max(\alpha, \beta)=1+\beta$$

其中

$$\alpha=g(l_1, \cdots, l_{j-1}, l_j+1, \cdots, l_m)\leqslant g(l_1, \cdots, l_m)-1$$

$$\beta=g(l_1, \cdots, l_{k-1}, l_k-1, l_{k+1}, \cdots, l_m)\geqslant g(l_1, \cdots, l_m)-1$$

策略 $C(j, k)$，对于 $j\leqslant k$，$l_j>0$，$l_k>0$　比较取自组 j 的一个小元素同取自组 k 的一个小元素。对应的关系是

$$f(l_1, \cdots, l_m)\leqslant 1+g(l_1, \cdots, l_{k-1}, l_k-1, l_{k+1}, \cdots, l_m)\geqslant g(l_1, \cdots, l_m)$$

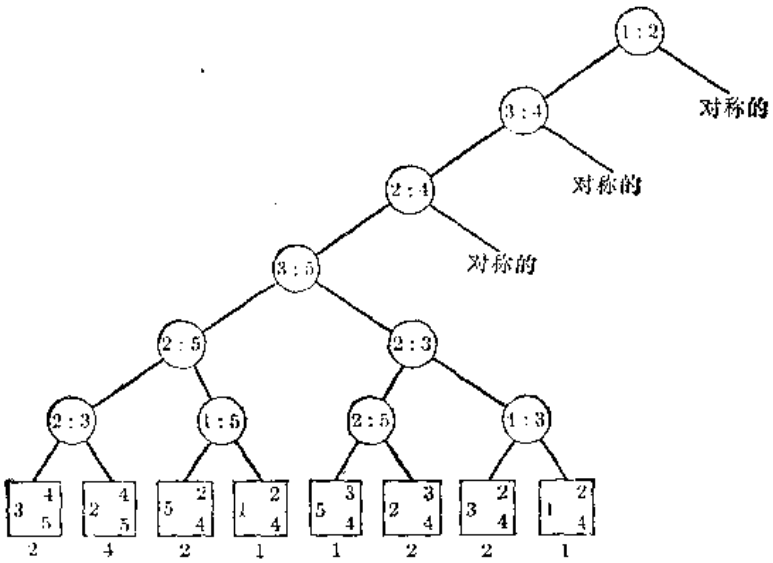
通过对于所有这些策略取右端极小值，即求出 $f(l_1, \cdots, l_m)$ 的值；因此 $f(l_1, \cdots, l_m)\geqslant g(l_1, \cdots, l_m)$。当 $m>1$ 时，策略 $A(m-1, m)$ 表明 $f(l_1, \cdots, l_m)\leqslant g(l_1, \cdots, l_m)$，因为当 $l_1\geqslant\cdots\geqslant l_m$ 时，$g(l_1, \cdots, l_{m-1}, l_m)=g(l_1, \cdots, l_{m-1}, l_m)$（证明：当 M 是 2^b 的一个正倍数时，对于 $0\leqslant a\leqslant b$ 有 $\lceil\log_2(M+2^a)\rceil=\lceil\log_2(M+2^b)\rceil$）。当 $m=1$ 时，使用策略 $C(1, 1)$。

〔S．S．基斯里特森的论文确定最优的策略 $A(m-1, m)$ 并以封闭的形式求出 $f(l, l, \cdots, l)$ 的值；f 的一般公式和这个简化了的证明是弗洛伊德于1970年发现的。〕

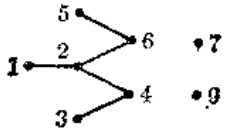
7．对于 $j>1$，如果 $j+1$ 在 α' 中，则 c_j 是 1 加上为选择 α' 的次一个最大元素所需要的比较数。如果 $j+1$ 在 α'' 中，则情况类似；而且 c_1 总为 0，因为这树在端点处样子总是相同的。

8．换言之，是否存在一株具有 n 个外部节点的扩充的二叉树，使得从根到 $t-1$ 个最远的内部节点的距离之和，小于完备的二叉树的相应的和？回答是否，因为不难证明，对于所有的 α，$\mu(\alpha)$ 的第 k 个最大的元素 $\geqslant\lfloor\log_2(n-k)\rfloor$。

9．（所有路径都使用六次比较。但这个过程不是最优的。）



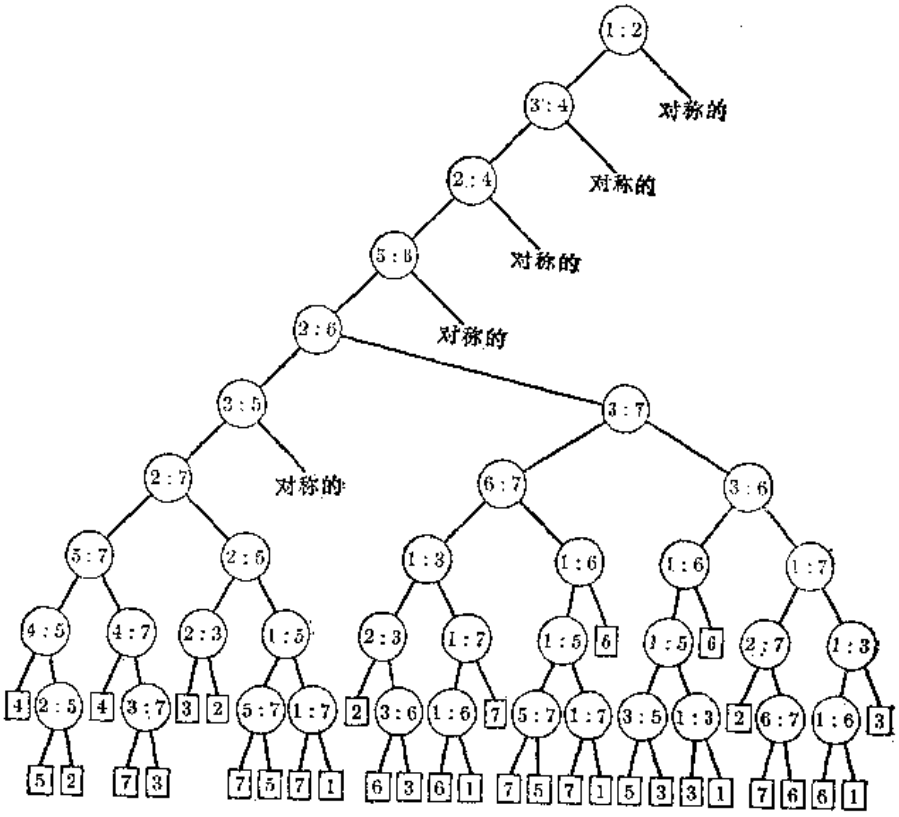
10．在进行了多琳（Doren）方法中的头 t 次比较后，我们可以抛弃X_8，因此必须求



当中的第四个最大者。这比我们在多琳方法中的第四步之后所知道的信息要多，所以至多还需要 12-4＝ 8 次比较。

11．在抛弃了$\{X_1, X_2, X_3, X_4\}$的最小者之后，我们有配置•⟷•加上 $n-3$ 个孤立的元素；这些当中的第三个最大者可以在$V_5(n-1)-1$个另外的步骤之后求得。

12．(通过使用尝试法手工地来找，并且使用习题 6 来帮助找出有成果的路线。)



13．在求得头 $f(n)$ 个元素的中间元素，比如说 X_j 之后，把它同其它每个元素进行比较；对于某个 k，这把诸元素分成近乎 $n/2-k$ 个小于 X_j 的和 $n/2+k$ 个大于 X_j 的两部分，剩下的是来求较大的集合当中第 $|k|$ 个最大或最小的元素，这需要 $n/2+O(|k|\log_2 n)$ 次另外的比较。$|k|$ 的平均值（考虑诸点一致地分布于〔0，1〕中）是 $O(1/\sqrt{n})+O(n/\sqrt{f(n)})$。设 $T(n)$ 是当 $f(n)=n^{2/3}$ 时的平均比较数；$T(n)-n=T(n^{2/3})-n^{2/3}+n/2+O(n^{2/3})$，由此得出结果。

注意到这样一点是有趣的，即当 $n=5$ 时，这个方法平均仅需要 $5\frac{13}{15}$ 次比较，比习题 9 的树还稍好些。

14．一般地说，由于习题 2，通过求 $\{X_1, \cdots, X_{n-1}\}$ 中的第 t 个最大者，并把它同 X_n 作比较，可以在 $U_t(n)\leqslant V_t(n-1)+1$ 次比较中求出第 t 个最大者。柯克帕特里克（Kirkpatrick）的对手实际上证明（12）是 $U_t(n+1)-1$ 的下界。

15．$\min(t, n+1-t)$。假定 $t\leqslant n+1-t$，如果当头 t 个字被读入时我们不把它们全部保存起来，则由于尚未知随后的值，我们可能忘记第 t 个最大者。反之，t 个单元已经足够了，因为我们可把新输入的元素同以前的第 t 个最大元素进行比较，当且仅当寄存器中的值是较大者时把它存起来。

16．这个算法由 $(a, b, c, d)=(n, 0, 0, 0)$ 开始并以 $(0, 1, 1, n-2)$ 终止，如果对手避免“出人意外”的结果，则在每次比较之后，唯一可能的转换是从 (a, b, c, d) 到本身或者到

$(a-2, b+1, c+1, d)$，如果 $a\geqslant 2$；

$(a-1, b, c+1, d)$ 或 $(a-1, b+1, c, d)$ 如果 $a\geqslant 1$；

$(a, b-1, c, d+1)$，如果 $b\geqslant 2$；

$(a, b, c-1, d+1)$，如果 $c\geqslant 2$。

由此得出，为了从 (a, b, c, d) 得到 $(0, 1, 1, a+b+c+d-2)$，需要 $\left\lceil\frac{3}{2}a\right\rceil+b+c-2$ 次比较〔CACM15(1972)，462-464〕。

17．首先对于最大者使用(6)，然后对于最小者使用(6)，注意其中有 $\lfloor n/2\rfloor$ 次比较对两者是公共的。

18．对于所有充分大的 n，$V_t(n)\leqslant 18n-151$。

21．设元素为 $\{x_1, \cdots, x_n, y, z\}$，其中 $n=2^k$。利用基斯里特森的方法，在 2^k+k-2 次比较中，求两个最大的 x，比如说 $b<a$；并且比较 $y:z$（比如说 $y<z$）。现在，如果 $z<b$，则继续基斯里特森方法，用 $k-1$ 次比较求 x 中第三个最大者，并把它同 z 作比较。另一方面，如果 $z>b$，则很容易在至多再有三步内完成它。

22．一般说来，当 $2^r\cdot 2^k<n-2-t\leqslant(2^r+1)\cdot 2^k$ 和 $t<2^r\leqslant 2t$ 时，这个由大小为 2^k 的 $t-1$ 个淘汰树开始的过程所作的比较比（11）少 $\lfloor(t-1)/2\rfloor$ 次，因为至少用来找（ii）中极小值的这许多个比较在（iii）中可“重新使用”〔关于进一步的改进，见叶志坚，CACM19(1976)，501-508.〕。

25．(a) 设两种类型分量的顶点被标记为 a；$b<c$。对于对非冗余的比较作用如下：情况 1，$a:a$，作出一个任意的决断。情况 2，$x:b$，比如说 $x>b$；此后对这个特定的

b 所作的所有比较 $y:b$，都将得到 $y>b$，否则这些比较便通过对于 $U_t(n-1)$ 的一个对手来加以判定，其总数将为 $\geqslant 2+U_t(n-1)$ 次比较。这个约化可被缩写成为"令 $b=\min$；$2+U_t(n-1)$"。情况3，$x:c$，令 $c=\max$；$2+U_t(n-1)$。

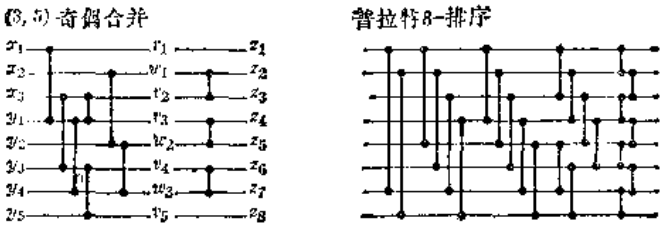
(b) 设新类型的顶点被标记为 d_1，$d_2<e$；$f<g<h>i$。情况1，$a:a$ 或 $c:c$，任意决断。情况2，$a:c$，比如说 $a<c$。情况3，$x:b$，令 $b=\min$；$2+U_t(n-1)$。情况4，$x:d$，令 $d=\min$；$2+U_t(n-1)$。情况5，$x:e$，令 $e=\max$；$3+U_t(n-1)$。情况6，$x:f$，令 $f=\min$；$2+U_t(n-1)$。情况7，$x:g$，令 f 和 $g=\min$；$3+U_t(n-2)$。情况8，$x:h$，令 $h=\max$；$3+U_{t-1}(n-1)$。情况9，$x:i$，令 $i=\min$；$2+U_t(n-1)$。

(c) 对于 $t=1$，我们有 $U_t(n)=n-1$，所以不等式成立。对于 $1<t\leqslant n/2-1$，使用归纳法和(b)。对于 $t=(n-1)/2$，使用归纳法和(a)。对于 $t=n/2$，$U_t(n-1)=U_{t-1}(n-1)$；使用归纳法和(a)。习题14现在产生了对于中间值的如下下界：

$$V_t(2t-1)\geqslant 3t+\lfloor t/2\rfloor-4$$

5.3.4 节

1.（当 m 为奇数时，最好在 v_m 后边接上 v_{m+1}，w_{m+1}，v_{m+2}，…，而不是像在图中那样接上 w_{m+1}，v_{m+1}，w_{m+2}，…。）



2. 对于 $n=8$ 见上面的图式。

3. 对于 $m\geqslant 1$，$C(m,\ m-1)=C(m,\ m)-1$。

4. 如果 $\hat{T}(6)=4$，则因为 $\hat{S}(6)=12$，每次将有三个比较动作，但是接着撤销底线和它的四个比较器将给出 $\hat{S}(5)\leqslant 8$，矛盾〔同样的论证产生 $\hat{T}(7)=\hat{T}(8)=6$〕。

5. 如果 $n\geqslant 2$，设 $f(n)=f(\lceil n/2\rceil)+1+\lceil\log_2\lceil n/2\rceil\rceil$；由习题1.2.4-34, 35，$f(n)=(1+\lceil\log_2\frac{1}{2}n\rceil)+(1+\lceil\log_2\frac{1}{4}n\rceil)+\cdots$。

6. 我们可以假定每个阶段作 $\lfloor n/2\rfloor$ 次比较（额外的比较不影响证明）。因为 $\hat{T}(6)=5$，只须证明 $T(5)=5$。当 $n=5$ 时，在两个阶段之后，我们不能避免偏序 或 ，它们的排序不可能再用两个阶段就完成。

7. 假定输入键是 $\{1,2,\cdots,10\}$，关键的事实在于，在进行了头16次比较之后，行2～5不可能包含8或9，它们也不可能同时包含6和7。

8．定理F的直截了当的推广。

9．由习题8，$\hat{M}(3,3)\geqslant\hat{S}(6)-2\hat{S}(3)$；$\hat{M}(4,4)\geqslant\hat{S}(8)-2\hat{S}(4)$；$\hat{M}(5,5)\geqslant 2\hat{M}(2,3)+3$；而且$\hat{M}(2,3)\geqslant\hat{S}(5)-\hat{S}(2)-\hat{S}(3)$。

10．提示见定理Z中的证明方法。然后证明，偶子序列中0的个数减奇子序列中0的个数为±1。

11．（M. W. 格林给出的解。）在下列意义下，这个网络是对称的，即，只要z_i同z_j进行比较，则就有一个相应的比较$z_{2^p-1-j}:z_{2^p-1-i}$。任何有能力对序列$\langle z_0,\ \cdots,\ z_{2^p-1}\rangle$排序的对称网络，也将对序列$\langle -z_{2^p-1},\ \cdots,\ -z_0\rangle$排序。

巴彻尔已经发现，这个网络确实能对一个双调序列的任何循环移位$\langle z_j, z_{j+1}, \cdots, z_{2^p-1}, z_0, \cdots, z_{j-1}\rangle$排序。这是0－1原理的一个推论。

12．$x\vee y$是（考虑0－1序列），但是，$x\wedge y$不是（考虑$\langle 3,1,4,5\rangle\wedge\langle 6,7,8,2\rangle$）。

13．一个完全的洗牌有以z_j代替z_i的效果，这里j的二进表示就是由i的二进表示向右循环转动一位得到的（参考习题3.4.2-13）。如果洗的是比较器而不是诸行；则这使得比较符的头一列作用于对偶$z[i]$和$z[i\oplus 2^{t-1}]$，下一列作用于$z[i]$和$z[i\oplus 2^{t-2}]$，…，第t列作用于$z[i]$和$z[i\oplus 1]$，第$t+1$列再次作用于$z(i)$和$z[i\oplus 2^{t-1}]$，等等。这里"$\oplus$"指二进表示的异或。这表明图57等价于图56；在s个阶段之后，我们得到2^s个元素的一些组，它们的次序交替地是正序和反序状。

14．在l个级之后，输入x_1至多可以处在2^l个不同的位置，在合并完成之后，x_1可处于$n+1$个不同的位置。

15．$[1:4][3:2][1:3][2:4][2:3]$。

16．这过程显然终止。步骤T2的每一次执行有交换第i_q个和第j_q个输出的作用，所以这个算法的结果，是以某种方式排列输出行。由于得到的（标准）网络不改变输入$\langle 1,2,\cdots,n\rangle$，输出行必然被排成它们原来的位置。

17．通过习题16的算法使网络标准化；然后考虑输入序列$\langle 1,\ 2,\ \cdots,\ n\rangle$，我们看到标准选择网络必然把$t$个最大的元素放入到$t$个最高编号的行中；而且$\hat{V}_t(n)$网络必然把第$t$个最大者放入行$n+1-t$。应用0－1原理。

18．定理A的证明表明$\hat{V}_t(n)\geqslant(n-t)\lceil\log_2(t+1)\rceil+\lceil\log_2 t\rceil$。

19．网络$[1:n][2:n]\cdots[1:3][2:3]$用$2n-4$个比较器选择最小的两个元素；对于$\hat{V}_2(n)$加$[1:2]$。下界从定理A得出（参考习题18）。

20．首先注意，当$n\geqslant 4$时，$\hat{V}_3(n)\geqslant\hat{V}_3(n-1)+2$；由对称性，头一个比较器可以假定是$[1:n]$；在这之后，必然出现一个网络去选择$\langle x_2,\ x_3,\ \cdots,\ x_n\rangle$中的第三个最大者，而另一个比较器接触行1。另一方面，$\hat{V}_3(5)\leqslant 7$，因为四个比较器找到$\{x_1,\ x_2,\ x_3,\ x_4\}$的min和max，而剩下的是对三个元素排序。

21．对α的长度用归纳法，因为$x_i\leqslant y_i$和$x_j\leqslant y_j$蕴涵$x_i\wedge x_j\leqslant y_i\wedge y_j$以及$x_i\vee x_j\leqslant y_i\vee y_j$。

22．对α的长度用归纳法。因为$(x_i\wedge x_j)(y_i\wedge y_j)+(x_i\vee x_j)(y_i\vee y_j)\geqslant x_iy_i+x_jy_j$（因此，$\nu(x\wedge y)\leqslant\nu(x\alpha\wedge y\alpha)$，这项研究是由W. 肖克利（W.Shockly）给出的）。

23. 设 $x_k=1$ 当且仅当 $p_k\geqslant j$，$y_k=1$ 当且仅当 $p_k>j$；则 $(x\alpha)_k=1$ 当且仅当 $(p\alpha)_k\geqslant j$，等等。

24. 对于 l_i' 这个公式是显然的，对于 l_j'，如同在提示中那样取 $z=x\wedge y$，而由习题21，应知 $(z\alpha)_i=(z\alpha)_j=0$。由习题23可知，加上额外的一些1到 z，即可看出存在一个排列 p，它使得 $(p\alpha')_j\leqslant\zeta(z)$。通过颠倒次序得出 u_i'，u_j' 的关系式。

25. (H.夏皮罗（H.Shepiro）给出的解。）设 p 和 q 是排列且 $(p\alpha)_k=l_k$ 和 $(q\alpha)_k=u_k$。通过一系列的步骤，每步交换相邻的一对整数（i，$i+1$），有可能把 p 变换成 q；输入中这样一种交换对第 k 个输出的影响至多是 ±1。

26. 存在有一一对应，它把 $P_n\alpha$ 的元素〈p_1，…，p_n〉对应到"覆盖的序列"$x^{(0)}$覆盖 $x^{(1)}$覆盖…覆盖$x^{(n)}$，其中$x^{(i)}$在$D_n\alpha$中；在这个对应中，当且仅当$p_i=i$ 时 $x^{(i-1)}=x^{(i)}\vee e^{(i)}$。例如，〈3，1，4，2〉对应于序列〈1，1，1，1〉覆盖〈1，0，1，1〉覆盖〈1，0，1，0〉覆盖〈0，0，1，0〉覆盖〈0，0，0，0〉。

27. 如果 x 和 y 表示其行是排好序的一个矩阵的不同的列，使得对所有的 i，$x_i\leqslant y_i$，而且如果 x_α 和 y_α 表示对这些列排序的结果，则所述的原理表明，对于所有 i，$(x\alpha)_i\leqslant(y\alpha)_i$，因为我们可以选择和 y 中任何 i 个给定的元素位于同一些行中的 x 的 i 个元素〔这一原理已在正文中使用，以证明谢尔的不变性性质即定理5.2.1K。关于这个思想的进一步的发展，见于戴维·盖尔（David Gale）和R.M.卡普有趣的文章中。"J.Computer and System Science" 6(1972)，103-115。关于列排序不弄乱排好序的行这一事实看来是在研究表格的处理时首先发现的；参考〔Hermann Boerner Dartellung von Gruppen (Springer，1955)137.〕。

28. 如果$\{x_{i_1}$，…，$x_{i_t}\}$是 t 个最大的元素，则$x_{i_1}\wedge\cdots\wedge x_{i_t}$是第 t 个最大的，如果$\{x_{i_1}$，…，$x_{i_t}\}$不是 t 个最大的元素，则 $x_{i_1}\wedge\cdots\wedge x_{i_t}$ 小于第 t 个最大的元素。

29. $(x_1\wedge y_1$，$(x_2\wedge y_1)\vee(x_1\wedge y_2)$，$(x_3\wedge y_1)\vee(x_2\wedge y_2)\vee(x_1\wedge y_3)$，$y_1\vee(x_3\wedge y_2)\vee(x_2\wedge y_3)\vee(x_1\wedge y_4)$，$y_2\vee(x_3\wedge y_3)\vee(x_2\wedge y_4)\vee(x_1\wedge y_5)$，$y_3\vee(x_3\wedge y_4)\vee(x_2\wedge y_5)\vee x_1$，$y_4\vee(x_3\wedge y_5)\vee x_2$，$y_5\vee x_3)$。

30. 应用分配律和结合律把任何公式化为$\wedge$的$\vee$的公式；然后用交换律、等幂律和吸收律导出规范形式。S_i正好是那样一些集合S，使得当（$x_j=1$ 当且仅当 $j\in S$）时这个公式为1，而对于S的任何真子集S'，当（$x_j=1$ 当且仅当 $j\in S'$）时这公式为0。

31. $\delta_4=166$。伦·丘奇（R.Church）〔Duke Math.J.6(1940)，732-734〕求出 $\delta_5=7579$，M.沃德（M.Ward）〔Bull. Amer.Math.Soc.52（1946)，423〕求出 $\delta_6=7828352$，而W.F.伦农（W.F.Lunnon）〔未发表，1969〕求出 $\delta_7=2208061288136$；这最后一个值还未曾被独立地校验过，显然对于 δ_n 没有简单的公式；D.克莱特曼(D.Kleitman)〔Proc. Amer.Math.Soc.21(1969)，677-682〕利用一个极端复杂的论证证明

$$\log_2\delta_n\sim\binom{n}{\lfloor n/2\rfloor}$$

32. G_{n+1} 也是所有串 $\theta\psi$ 的集合，其中 θ 和 ψ 在 G_n 中，而且作为0和1的向量 $0\leqslant\psi$。由此得出，G_n是所有0和1的串 $z_0\cdots z_{2^n-1}$ 的集合，其中每当在0-1向量的意义下，i 的二进表示"$\leqslant$" j 的二进表示时 $z_i\leqslant z_j$。G_n 的每个元素 z_0，…，z_{2^n-1}，除了00…0和11…1

之外，在 $f(x_1, \cdots, x_n)=z[(x_1\cdots x_n)_2]$ 的对应之下，表示从 D_n 到 $\{0, 1\}$ 的一个∧-∨函数 $f(x_1, \cdots, x_n)$。

33．如果这样一个网络存在，则对于某个函数 f，我们将有 $(x_1\wedge x_2)\vee(x_2\wedge x_3)\vee(x_3\wedge x_4)=f(x_1\wedge x_2, x_1\vee x_2, x_3, x_4)$ 或 $f(x_1\wedge x_3, x_2, x_1\vee x_3, x_4)$ 或…或 $f(x_1, x_2, x_3\wedge x_4, x_3\vee x_4)$。选择 $\langle x_1, x_2, x_3, x_4\rangle=\langle x, \bar{x}, 1, 0\rangle$，$\langle x, 0, \bar{x}, 1\rangle$，$\langle x, 1, 0, \bar{x}\rangle$，$\langle 1, x, \bar{x}, 0\rangle$，$\langle 1, x, 0, \bar{x}\rangle$，$\langle 0, 1, x, \bar{x}\rangle$ 表明不存在这样一个函数 f。

34．是；在证明了这点之后，你已经准备好着手处理图49中的 $n=16$ 的网络。

35．否则，其中仅有 i 和 $i+1$ 放错地方的排列将永远不被排序了。设 D_k 是在一个标准的排序网络中比较器 $[i:i-k]$ 的个数。则 $D_1+2D_2+D_3\geqslant 2(n-3)$，因为对于 $1\leqslant i\leqslant n-3$，从 $\{i, i-1\}$ 到 $\{i-2, i-3\}$ 必然有两个比较器。类似地 $D_1+2D_2+\cdots+kD_k+(k-1)D_{k-1}+\cdots+D_{2k-1}\geqslant k(n-2k+1)$，这是一个由 J·M·波拉德给出的公式。也可以证明：$2D_1+D_2\geqslant 3n-4$；如果我们对于所有的 j 删去形如 $[j:j+1]$ 的头一些比较器，则对于 $1\leqslant i\leqslant n-2$ 必然至少还剩下一个比较器位于 $\{i, i+1, i+2\}$ 之内。类似地 $kD_1+(k-1)D_2+\cdots+D_k\geqslant S(k+1)(n-k)+k(k-1)$。

36．(a) 每个相邻的比较器都使反序数减少 0 个或 1 个，而且 $\langle n, n-1, \cdots, 1\rangle$ 有 $\binom{n}{2}$ 个反序。(b) 设 $\alpha=\beta[p:p+1]$，对 α 的长度用归纳法，进行论证如下：如果 $p=i$，则 $j>p+1$，而且 $(x\beta)_p>(x\beta)_j$，$(x\beta)_{p+1}>(x\beta)_j$；因此 $(y\beta)_p>(y\beta)_j$ 和 $(y\beta)_{p+1}>(y\beta)_j$。如果 $p=i-1$，则或者 $(x\beta)_p$ 或者 $(x\beta)_{p+1}$ 要 $>(x\beta)_j$；因此或者 $(y\beta)_p$ 或者 $(y\beta)_{p+1}>(y\beta)_j$。如果 $p=j-1$ 或 j，论证是类似的。对于其它的 p，论证是显然的。〔如果 α 是一个原始的排序网络，则 α^R 也是（比较器是颠倒顺序的）。关于(c)的推广及另一个证明，见N. G. de Bruijn, Discrete Mathematics 9 (1974), 333-339。〕

37．只须证明：如果每个比较器被一个交换操作所代替，则我们便得到一个"反演网络"，它把 $\langle x_1, \cdots, x_n\rangle$ 转换成为 $\langle x_n, \cdots, x_1\rangle$。但在这种解释之下，不难来跟踪 x_k 的路线（注意，排列 $\pi=(1, 2)(3, 4)\cdots(2n-1, 2n)(2, 3)(4, 5)\cdots(2n-2, 2n-1)=(1, 3, 5, \cdots, 2n-1, 2n, 2n-2, \cdots, 2)$ 满足 $\pi^{n/2}=(1, 2n)(2, 2n-1)\cdots(n-1, n)$)。1954年 H·西沃德简略地提到了奇偶转置排序；它已经为A·格拉斯里(A. Grasselli)〔IRE Trans. EC-11(1962), 483〕和为考茨（Kautz）等〔IEEE Trans. C-17 (1968), 443-451〕所讨论。关于这个网络的反射性实际上老早就由 H. E. 达德尼在他的"青蛙难题"之一给出了〔Amusements in Mathematics(1917), 193〕。

38．设 u 是 $(x\alpha)_j$ 的最小元素，并设 $y^{(0)}$ 是 D_n 中任何一个这样的向量，它使得 $(y^{(0)})_k=0$ 蕴涵着 $(x\alpha)_k$ 包含一个 $\leqslant u$ 的元素，$(y^{(0)})_k=1$ 蕴涵着 $(x\alpha)_k$ 包含一个 $>u$ 的元素。如果 $\alpha=\beta[p:q]$，则有可能找到一个满足同样条件，但是以 β 代替 α 的并且使得 $y^{(1)}[p:q]=y^{(0)}$ 的向量 $y^{(1)}$，从 $(y^{(0)})_j=1$，$(y^{(0)})_j=0$ 开始，我们最终有满足所需条件的向量 $y=y^{(r)}$。

杰勒德·鲍德特（Gerard Baudet）和戴维·史蒂文森（David Stevenson）已经发现，把习题37和38结合在一起，产生出在 k 个处理机上只需 $(n\log_2 n)/k+O(n)$ 个比较的简单的排序方法：首先对大小为 $\leqslant\lceil n/k\rceil$ 的 k 个子文件排序，然后使用 k 阶的"奇偶转置合并"在 k 次扫描中合并它们（IEEE. Trans. C-27 (1978). 84-87)。

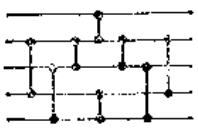
39. $(x \veebar y) \veebar z$ 和 $x \veebar (y \veebar z)$ 两者都表示多重集合 $x \uplus y \uplus z$ 的最大的 m 个元素；$(x \wedge\!\!\!\wedge y) \wedge\!\!\!\wedge z$ 和 $x \wedge\!\!\!\wedge (y \wedge\!\!\!\wedge z)$ 表示最小的 m 个。如果 $x = y = z = \{0, 1\}$，则 $(x \wedge\!\!\!\wedge z) \veebar (y \wedge\!\!\!\wedge z) = (x \wedge\!\!\!\wedge y) \veebar (x \wedge\!\!\!\wedge z) \veebar (y \wedge\!\!\!\wedge z) = \{0, 0\}$，而（$\{0, 0, 0, 1, 1, 1\}$ 的中间元素）$= (x \veebar y) \wedge\!\!\!\wedge z = \{0, 1\}$。三个元素的排序网络和习题38的结果意味着 $x \uplus y \uplus z$ 的中间元素可以表达成 $((x \veebar y) \wedge\!\!\!\wedge z) \veebar (x \wedge\!\!\!\wedge y)$ 或 $((x \wedge\!\!\!\wedge y) \veebar z) \wedge\!\!\!\wedge (x \veebar y)$ 或者在这些表达式中排列 x，y，z 后所得到的任何其它公式（对于中间元素似乎没有"对称"的公式）。

40. 设 $\alpha' = \alpha[i : j]$，并设 k 是一个 $\neq i$，j 的指标。如果对所有 x，$(x\alpha)_i \leqslant (x\alpha)_k$，则 $(x\alpha')_i \leqslant (x\alpha')_k$；如果对所有 x，$(x\alpha)_k \leqslant (x\alpha)_i$ 且 $(x\alpha)_k \leqslant (x\alpha)_j$，则当 α 被代之以 α' 时同样的事实成立；如果对于所有的 x，$(x\alpha)_k \leqslant (x\alpha)_i$，则 $(x\alpha')_k \leqslant (x\alpha')_j$。这样一来我们看到，$\alpha'$ 至少与 α 有一样多的已知关系，如果 $[i : j]$ 不是冗余的，则还多加上一个〔Bell System Tech. J. 49(1970)，1627-1614〕。

41. (a) 对于 $i_s \neq k \neq j_s$，设 $y_{i_s} = x_{j_s}$，$y_{j_s} = x_{i_s}$，$y_k = x_k$；则 $y\alpha^s = x\alpha$。(b) 这是显然的，除非集合 $\{i_s, j_s, i_t, j_t\}$ 仅有三个不同的元素；假设 $i_s = i_t$。于是如果 $s < t$，则头 $s-1$ 个比较器在 $(\alpha^t)^t$ 和 $(\alpha^t)^s$ 两者中分别要由 (j_s, j_t, i_s) 代替 (i_s, j_s, j_t)。(c) $(\alpha^t)^s = \alpha$ 和 $\alpha^t = \alpha$，所以我们可以假设 $s_1 > s_2 > \cdots > s_k > 1$。(d) 设 $\beta = \alpha[i : j]$；则 $g_\beta(x_1, \cdots, x_n) = (\bar{x}_i \vee x_j) \wedge (g_\alpha(x_1, \cdots, x_i, \cdots, x_j, \cdots, x_n) \vee g_\alpha(x_1, \cdots, x_j, \cdots, x_i, \cdots, x_n))$，迭代这个恒等式就产生出此结果。(e) 当且仅当在 G_α 中没有从 i 到 j 的路径时 $f_\alpha(x) = 1$，其中 $x_i > x_j$。如果 α 是一个排序网络，则 α 的共轭也是；而且对于满足 $x_i > x_{i+1}$ 的所有 x，$f_\alpha(x) = 0$。取 $x = e^{(i)}$；这表明对于某个 $k_1 \neq i$，G 有从 i 到 k_1 的一条弧。如果 $k_1 \neq i+1$，则 $x = e^{(i)} \vee e^{(k_1)}$ 表明对于某个 $k_2 \notin \{i, k_1\}$ G 有从 i 或 k_1 到 k_2 的一条弧。如果 $k_2 \neq i+1$，则以同样方式继续进行直到找到 G 中从 i 到 $i+1$ 的一条路径。反之，如果 α 不是一个排序网络，则设 x 是满足 $x_i > x_{i+1}$ 的一个向量，且有 $g_\alpha(x) = 1$。某个共轭 α' 有 $f_{\alpha'}(x) = 1$，所以 $G_{\alpha'}$ 不可能有从 i 到 $i-1$ 的路径〔一般地，对于所有 x，$(x\alpha)_i \leqslant (x\alpha)_j$ 的充要条件是，对于所有同 α 共轭的 α'，有从 i 到 j 的一条有向路径〕。

42. 从某个输入到最大的输出（考虑定理A中的 m_n），必然存在有长度为 $\lceil \log_2 n \rceil$ 或更长的一条路径；当该输入被置成为 ∞ 时，在这条路径上的诸比较器有一个预先确定的特性，而且剩下的网络必然是一个（$n-1$）排序器〔IEEE Trans on Computers C-21(1972)，612-613〕。

43.



〔如果 $P(n)$ 表示在一个排列网络中所需要的极小开关数，则显然 $P(n) \geqslant \lceil \log_2 n! \rceil$。通过稍微推广L. J. 戈尔茨坦（L. J. Goldstein）和 S. W. 莱博霍尔兹（S. W. Leibholz）给出的一个构造，见 IEEE Trans. EC-16(1967)，637-641。人们能够证明 $P(n) \leqslant P(\lfloor n/2 \rfloor) + P(\lceil n/2 \rceil) + n - 1$，因此，对所有的 n，$P(n) \leqslant B(n)$，其中 $B(n)$ 是等式5.3.1-3的

二叉插入函数；M·W·格林已经证明$P(5)=8$（未发表）。〕

48．等价地，由定理Z，我们应找到满足在区间〔0，1〕中的有理数x，y上的操作$x \vee y=\min(x+y,\ 1)$，$x \wedge y=\max(0,\ x+y-1)$的所有恒等式〔这好比是由装满到$x$的杯子中倒出尽可能多的液体到装满到$y$的杯子中的操作，这是J.M.勃拉德指出的〕。所有这些恒等式可由卢卡谢维茨的多值逻辑通过一个四个公理的公理系统和一个推导规则得到，见Rose和Rosser，Trans. Amer. Math. Soc. 87(1958)，1-53。

52．如果$h[k+1]=h[k]+1$，而且文件不是有序的，则在下次扫描时对于它必然发生某件事情；由习题5.2.2-1，这减少了反序的数目，因此这个文件最终将成为排好序的。但是如果对于$1 \leqslant k<m$，$h[k+1] \geqslant h[k]+2$，则如果最小的键开始时在R_2中，那么它将决不能移动到其适当的位置去。

53．我们使用提示，并认为$K_{N+1}=K_{N+2}=\cdots=1$。如果在步骤j中$K_{h[1]+j}=\cdots=K_{h[m]+j}=1$，而且如果对于某个$i>h[1]+j$，$K_i=0$，则我们必然有$i<h[m]+j$，因为1的个数少于$n$。假设$k$和$i$是使得$h[k]+j<i<h[k+1]+j$以及$K_i=0$的极小值，设$s=h[k+1]+j-i$；我们有$s<h[k+1]-h[k] \leqslant k$。在第$j-s$步中，必定至少有$k+1$个0已被读写头扫描过，因为$K_i=K_{h[k+1]+j-s}$在该步被置成0；$s$步以后，在$K_{h[1]+j}$和$K_i$（包含）之间，至少剩下$k+1-s \geqslant 2$个0，同$i$的极小性矛盾。

第二次扫描放好其次$n-1$个元素，等等。如果我们从排列$N(N-1)\cdots 21$开始，则头一次扫描把它变成为$(N+1-n)(N-n)\cdots 1(N+2-n)\cdots(N-1)N$，因为每当$1 \leqslant h[1]+j$和$h[m]+j \leqslant N$时，$K_{h[1]+j}>K_{h[m]+j}$；因此这个界是最好的。

54．假设$h[k+1]-s>h[k]$和$h(k) \leqslant s$；如果最小的键在R_{n-s}处开始，则它在位置R_i（对于$i>1$）处结束，因此$h[k+1] \leqslant 2h[k]$是必要的；由下述定理$t=0$的特殊情况，它也是充分的。

定理 如果$n=N$，以及如果$K_1\cdots K_N$是$\{1,\ 2,\ \cdots,\ n\}$的一个排列，且如果对于$1 \leqslant k<m$和$0 \leqslant i \leqslant t$，$h[k+1] \leqslant h[k]+h[k-i]+i$，则对于$1 \leqslant i \leqslant t+1$，一次排序扫描将置$K_i=i$（约定，当$k \leqslant 0$时设$h[k]=k$）。

证明 对t用归纳法；如果到步骤t时键$t+1$不在读写头下边，则我们可以假定对于某个$s>0$，它出现在位置$R_{h[k+1]+t-s}$中，其中$h[k+1]-s<h[k]$；因此$h[k-t]+t-s>0$。但如果我们考察步骤$t-s$，则这是不可能的，因为按假定这个步骤将把元素$t+1$放置到位置$R_{h[k+1]+t-s}$中，尽管至少有$t+1$个较低的头正在工作。

（这个条件对于$t=0$，1是必要的；但对于$t=2$则不然。）

55．如果数$\{1,\ \cdots,\ 23\}$正在被排序，则上道习题中的定理表明$\{1,2,3,4\}$找到了它们真正的目的地。当诸0和诸1被排序时，有可以验证，在步骤-2，-1和0中，不可能当所有的读写头都读0时，所有不处于这些头之下的位置都包含1；因此上题的证明可被推广以表明$\{5,6,7\}$找到了它们真正的目的地。最后，由习题53中的论证，$\{8,\cdots,23\}$必然被排序。

57．当$r \leqslant m-2$时，读写头把串$0^p1^101^301^70\cdots01^{2^r-1}01^q$变成为$0^{p+1}1^101^3\,01^7\,0\cdots 01^{2^r-1}\,01^{2^r-1+q}$；因此需要$m-2$次扫描。〔当读写头在位置1，2，3，5，…，$1+2^{m-2}$处时，

普拉特发现了一个类似的结果：串$0^p1^a01^{2^b-1}01^{2^{b+1}-1}0\cdots01^{2^r-1}01^q$，$1\leqslant a\leqslant 2^{b-1}$，变成$0^{p+1}1^{a-1}01^{2^b-1}01^{2^{b+1}-1}0\cdots01^{2^r-1-1}01^{2^r+q}$，因此对于这种头的序列，在最坏情况下至少需要$m-\lceil\log_2 m\rceil-1$次扫描。后一个头序列是特别有趣的，因为它已经用作P.N.阿姆斯特朗发明的非常精巧的排序设备的基础〔参考U.S.Patent 3399383〕普拉特猜测，对于所有输入，这些输入序列提供了真正最坏的情况。〕

58．在快速排序期间，每个键K_2，$\cdots K_N$都同K_1作过比较；设$A=\{i\mid K_i<K_1\}$，$B=\{j\mid K_j>K_1\}$。随后的操作独立地对A和B快速排序；在快速排序和有限制的一致算法中，对于A中的i和B中的j，禁止所有$K_i:K_j$的比较，而无限制的一致算法不禁止其它比较。

在这种情况下，我们甚至可以进一步限制这个算法，省略情况1和情况2使得仅当明显地做比较时，才把弧加到G中，而且当测试冗余性时仅考虑长度为2的路径。解决这个问题的另一个办法，是考虑6.2.2节的等价树插入排序算法，它正好以相同的次序进行与一致算法相同的比较。

59．(a) K_{a_j}同K_{b_j}进行比较的概率，是c_j个其它的特定键不处于K_{a_j}和K_{b_j}之间的概率；这是随机地从$\{1,2,\cdots,c_j+2\}$中选择的两个数是相连的概率，也就是

$$(c_j+1)\Big/\binom{c_j+2}{2}$$

(b) c_j的头$n-1$个值为0，然后是$(n-2)$个1，$(n-3)$个2，等等；因此平均值是$2\sum_{1\leqslant k\leqslant n}(n-k)/(k+1)=2\sum_{1\leqslant k\leqslant n}((n+1)/(k+1)-1)=2(n+1)\times(H_{n+1}-1)-2n$。(c)合并的"两部分构成的"本性表明，对于这个序列来说有限制的一致算法和一致算法是相同的。对包含顶点N的对偶来说，诸c分别等于$0,1,\cdots,N-2$；所以平均比较次数同快速排序相同。

60．否；当$N=5$时，设有以$(1,5)(1,2)(2,3)(3,4)(4,5)$结束的对偶序列会要求作10次比较〔一个有趣的研究问题：对于所有的N，试找出一个（有限制的）一致排序方法，使其最坏情况尽可能地好〕。

62．每次扫描时一个项至多失去一个反序，所以极小的扫描次数至少是在输入排列中任何项的极大反序数。气泡排序策略达到这个界，因为每次扫描都使每个有反序的项的反序计数减1（参考5.2.2-1）。可能需要另外的扫描来确定排序是否完备，但是这个习题的行文允许我们忽略这样的考虑。

也许不幸的是，通过自动机研究计算复杂性得到的头一个结果，是确定了一个排序的方法的"最优性"，而从程序设计的观点看来，它竟如此之低劣！同随机数生成的情况类似，当从一个特定的观点看来某些生成程序是"最优的"，因而被建议广泛采用时，实际上却是倒退了好几步。教益在于最优性的结果通常大量地依赖于抽象模型；尽管这些结果是非常有趣的，但在实用中必须明智地来应用它们。

〔德穆斯曾考虑对于一个r寄存器机器（保存r的一个因子）和对于一个类图灵机的推广。在这种机器中扫描方向可随意地在左右和右左之间摆动。他发现，这后一种类型的机器可以作直接插入以及鸡尾混合排序；但是任何这样的1-寄存器机器平均必须通过至少

$-\frac{1}{4}N^2$个步骤，因为每步至多使总的反序数减少1。最后他考虑了r-寄存器随机存取机器以及极小-比较排序的问题。〕

5.4 节

1．我们可以省略内部排序阶段，但是一般地这样将会慢得多，因为它将增加每块数据在外存上读和写的次数。

2．诸路段如同在（1）中那样分布，然后带3被置为$R_1 \cdots R_{2000}$；$R_{2001} \cdots R_{4000}$；$R_{4001} \cdots R_{5000}$。在重绕所有的带之后，一个"一路合并"置T_1和T_2分别成为（2）中T_3和T_4的内容，然后T_1和T_2被合并成T_3，而信息被复写回去并再次被合并，总共进行五次扫描。一般地说，这个过程和四条带的平衡合并类似，但是在每次合并扫描之间有复写扫描，所以共实施了两倍减1次扫描。

3．(a)$\lceil \log_P S \rceil$，(b) $\log_B S$，其中$B=\sqrt{P(T-P)}$称为"有效的合并能力"。当$T=2P$时，有效能力是P；当$T=2P-1$时，有效能力是$\sqrt{P(P-1)}=P-\frac{1}{2}-P^{-1}+o(P^{-2})$，比$\frac{1}{2}T$稍小些。

4．$\frac{1}{2}T$。如果T是奇数而且P一定是一个整数，则$\lceil T/2 \rceil$和$\lfloor T/2 \rfloor$给出相同的极大值。按照习题3，最好有$P \geqslant T-P$，所以对于平衡合并我们将选择$P=\lceil T/2 \rceil$。

5.4.1 节

1．

$$087 \quad 154 \quad 170 \quad 426 \begin{cases} 503 \begin{cases} 503 & \infty \\ 908 & \infty \end{cases} \\ 426 \begin{cases} 426 & 653 & \infty \\ 612 & \infty \end{cases} \end{cases}$$

2．路径 (061)—(154)—(087)—(512)—[061] 将被改变成为 (087)—(154)—(512)—(612)—[612]。

(沿着这条路径我们实际上进行了从下到上的一个"气泡排序"！)

3．and fourscore our seven years/ago brought fathers forth on this/a conceived continent in liberty nation new the to/and dedicated men proposition that/all are created equal。

4．(这个问题稍微含混些；按这种解释，在储藏库快溢出之前我们不清除内存。)
and fourscore on our seven this years/ago brought continent fathers forth in liberty nation new to/a and conceived dedicated men proposition that the/all are created equal。

5．假的。对于所有$P \geqslant 1$，定义具有P个外部节点的完备的二叉树。

6．在步骤R6的开始处插入"如果T=LOC(X[0])，则转到R2，否则"并从步骤R7删去类似的短句。

7．没有输出，且RMAX一直等于0。

8．如果头一个实际的键是∞，则这记录将丢失。为了避免∞，我们可以置一个开关，开始时避免在步骤R4中包含LASTKEY的测试。然后当头一次在步骤R3中$RQ\neq 0$时，该开关被改变使得R4测试LASTKEY，而R3不再测试RQ。

9．例如，假定当前的路段是递增的，而下一个将是递减的，则算法R的诸步骤除开下列一处要改动外，将正确地工作：在步骤R6中，如果RN(T)=RQ>RC，则颠倒对于KEY(LOSER(T))和KEY(Q)的测试。

当RC改变时，步骤R4和R6的键测试应当适当地改变。

10．设$\cdot j$=LOC($X[j]$)。如果我们首先置LOSER($\cdot 0$)←Q以及RN($\cdot 0$)←RQ，则算法R的机制确保了每当我们达到步骤R3时下列条件为真：LOSER($\cdot 0$)，…，LOSER($\cdot(P-1)$)的值是$\{\cdot 0, \cdot 1, \cdots, (P-1)\}$的一个排列；而且存在诸指针$\{$LOSER($\cdot j$)|RN($\cdot j$)$=0\}$的一个排列，它对应于一个实际的比赛。换言之，当R($\cdot j$)=0时，KEY(LOSER($\cdot j$))的值，因此还有LOSER($\cdot j$)的值，是无关紧要的；我们可以在这样的"失利者"之间进行内部排列。在P步之后，所有RN($\cdot j$)都将非0，所以整个树将是一致的（对于提示的回答是"是的"）。

人们可能抱怨此算法比较未曾初始化的KEY值。如果这样的行为太使人震惊，可以通过在比如说步骤R1中置所有KEY值成为0而避免它。

11．真（两个键都取自于定理K的证明中的相同的子序列）。

13．当头一个路段已经结束时，保留在内存中的键一般比平均值更小，因为它们不使它进入头一个路段当中，于是第二个路段可以输出更多的较小的键。

14．假定在扫雪车达到它的"稳定状态"后，它处于随机点u，$0\leqslant u<1$，雪突然停止，则倒数第二个路段包含$(1+2u-u^2)P$个记录，而最后的路段包含u^2P个记录。对它乘以du进行积分，就得出倒数第二个路段的平均时间是$\left(2-\frac{1}{3}\right)P$个记录，最后路段的平均时间是$\frac{1}{3}P$个记录。

15．假的，最后的路段可以是任意长的；但是仅当输入被穷尽，而所有内存中的记录都属于同一个路段这种相当稀少的情况下才有可能。

16．当且仅当每个元素都有少于P个反序（参考5.1.1，5.4.8）。考察反序表，当$N\leqslant P$时概率为1，当$N\geqslant P$时，概率为$P^{N-P}P!/N!$（然而在实际的做法中，一次扫描的排序并不太罕见，因为即使当人们怀疑一个文件是否有序时，作为一种预防措施，人们也倾向于把这个文件排序）。

17．除开长度为P的最后者外，所有的都恰有$\lceil N/P\rceil$个路段（"最坏的情况"）。

18．在第二次扫描时没有什么变化，因为能够证明，对于$1\leqslant k\leqslant P$，一个路段的第k个记录至少小于前一个路段的$P+1-k$个记录（然而，似乎没有简单的方法来表征当$P'>P$时，P路替代选择后边接以P'路替代选择的结果）。

19．如同在（2）的推导中那样论证$h(x, t)dx=KLdt$，这一次对于所有x有$h(x, t)=1+Kt$，而且$P=IL$。这蕴涵若$x(t)=L\ln((I+Kt)/I)$，使得当$x(T)=L$时，我们有$KT=(e-1)I$，故从$t=0$起的落雪量是$(e-1)IL=(e-1)P$。

20. 如同在习题 19 中那样，我们有 $(I+Kt)dx=K(L-x)dt$；因此 $x(t)=LKt/(I+Kt)$。积雪量是 $IL=P=P'=\int_0^T x(t)Kdt=L(KT-I\ln((I+KT)/I))$，因此 $KT=\alpha I$，其中 $\alpha\approx 2.1461$ 是 $1+\alpha=e^{\alpha-1}$ 的根。路段长度是在 $0\leqslant t\leqslant T$ 期间下雪的总量，即 $KLT=\alpha P$。

21. 如同在正文中那样进行，但在每个路段之后，在扫雪机再次开始工作之前等候 $P-P'$ 个雪花落下。这意味着 $h(x(t),\ t)$ 现在是 KT_1 而不是 KT，其中 T_1-T 是额外的落雪所花费的时间数量。路段长度是 LKT，$x(t)=L(1-e^{-t/T_1})$，$P=LKT_1e^{-T/T_1}$，以及 $P'=\int_0^T x(t)Kdt=P+LK(T-T_1)$。换言之，当对于 $0\leqslant\theta\leqslant 1$，$P'=(1-(1-\theta)e^{\theta})P$ 时，得到长度为 $e^{\theta}P$ 的一个路段。

22. 对于 $0\leqslant t\leqslant(\kappa-1)T$，$dx\cdot h=Kdt(x(t+T)-x(t))$，而对于 $(\kappa-1)T\leqslant t\leqslant T$，$dx\cdot h=Kdt(L-x(t))$，其中 h 在扫雪犁的位置处被看作恒等于 KT。由此得出，对于 $0\leqslant j\leqslant k$，$0\leqslant u\leqslant 1$，$t=(\kappa-j-u)T$，我们有 $x(t)=L(1-e^{\kappa-j}F_j(u)/F(\kappa))$。路段长度是 KTL，这即是在稳定状态下扫雪机接连两次离开 0 点的时间之间下雪的数量；P 是在每次扫雪机的最后速度突变期间清除的数量，即 $KT(L-x(\kappa T))=KTLe^{\kappa}/F(\kappa)$；而且 $P'=\int_0^{\kappa T}x(t)Kdt$ 可以证明有所述形式。

〔注：结果是，对于 $k=0$，所述公式也正确。当 $k\geqslant 1$ 时，进入雪堆两次的每个路段的元素个数是 $P''=\int_0^{(\kappa-1)T}x(t)Kdt$，而且容易证明（路段长度）$=P'+P''=(e-1)P$，这是由弗雷泽和黄泽权所指出的一个现象。$F_k(\theta)$ 的母函数如此类似于习题 5.1.3-11 中的母函数，这是否巧合呢？〕

23. 设 $P=pP'$ 且 $q=1-p$。在积雪中的最初的 pP' 个雪花已经以随机次序在开头移走之后，头 T_1 个时间单位的落雪来自积雪中剩下的 qP' 个雪花；而且当旧的积雪扫光时，雪再次均匀落下，我们选择 T_1 使得 $KLT_1=qP'$。对于 $0\leqslant t\leqslant T_1$，$h(x,\ t)=(p+qt/T_1)g(x)$，其中 $g(x)$ 是雪从位置 x 落入雪堆的高度；对于 $T_1\leqslant t\leqslant T$，$h(x,\ t)=g(x)+(t-T_1)K$。对于 $0\leqslant t\leqslant T_1$，$g(x(t))$ 是 $(q(T_1-t)/T_1)\times g(x(t))+(T-T_1)K$；而对于 $T_1\leqslant t\leqslant T$，$g(x(t))=(T-t)K$。因此对于 $0\leqslant t\leqslant T$，$h(x(t),\ t)=(T-T_1)K$，而且 $x(t)=L(1-\exp(-t/(T-T_1)))$。总的路段长度是 $(T-T_1)KL$；从雪堆重新又“循环”回去（参考习题 22）的总数量是 T_1KL；而且在时间 T 之后清除的总数量是 $P=KT(L-x(T))$。

所以当雪堆的大小为 $(1+(s-1)e^{t}/s)P$ 时，这个习题的假定给出长度为 $(e^{t}/s)P$ 的路段。这比习题 22 的结果坏得多，因为雪堆的内容在习题 22 的情况下是以一个更有利的次序来使用的。

（$h(x(t),\ t)$ 在这样多的问题当中都是常数这一事实是不足为怪的，因为它等价于说在系统的一个稳定状态期间，得到的每个路段的诸元素是一致分布的。）

24. (a) 证明基本上是同样的，每个子序列的路段和输出路段有相同的方向。(b) 所述的概率是路段的长度为 $n+1$ 而且它为 y 所跟随的概率；当 $x>y$ 时，它等于 $(1-x)^n/n!$，而当 $x\leqslant y$ 时，它是 $(1-x)^n/n!-(y-x)^n/n!$。(c) 归纳法。例如，如果

第 n 个路段是递增的，则第 $n-1$ 个是递减的概率为 p，所以头一个积分可应用。(d)我们求得 $f'(x)=f(x)-c-pf(1-x)-qf(x)$，因此 $f''(x)=-2pc$，它最终导致 $f(x)=c(1-qx-px^2)$，$c=6/(3+p)$。(e) 如果 $p>eq$，则 pe^x+qe^{1-x} 对于 $0\leqslant x\leqslant 1$ 是单调递增的，而且 $\int_0^1 |pe^x+qe^{1-x}-e^{1/2}|dx=(p-q)(e^{1/2}-1)^2<0.43$。如果 $q\leqslant p<eq$，则 pe^x+qe^{1-x} 处于 $2\sqrt{pqe}$ 和 $p+qe$ 之间，所以 $\int_0^1 |pe^x+qe^{1-x}-\frac{1}{2}(p+qe+2\sqrt{pqe})|dx\leqslant\frac{1}{2}(\sqrt{p}-\sqrt{qe})^2<0.4$；而如果 $p<q$，则我们可以使用一个对称的推理。于是，对于所有的 p 和 q，有一个常数 C，使得 $\int_0^1 |pe^x+qe^{1-x}-C|dx<0.43$。设 $\delta_n(x)=f_n(x)-f(x)$，则 $\delta_{n+1}(y)=(1-e^{y-1})\int_0^1 (pe^x+qe^{1-x}-C)\delta_n(x)\cdot dx+p\int_0^{1-y} e^{y-1+x}\delta_n(x)\,dx+q\int_y^1 e^{y-x}\delta_n(x)dx$；因此如果 $\delta_n(y)\leqslant\alpha_n$，则 $|\delta_{n+1}(y)|\leqslant(1-e^{y-1})(1.43)\alpha_n<0.91\alpha_n$。(f) 对于所有 $n\geqslant 0$，$(1-x)^n/n!$ 是路段长度 $>n$ 的概率。(g) $\int_0^1 (pe^x+qe^{1-x})f(x)dx=6/(3+p)$。

26. (a) 考虑具有 $n+r+1$ 个元素和 n 个自左至右极小值的排列数，其中最右的元素不是最小的。(b) 借助于记号索引中关于斯特林数的定义，利用

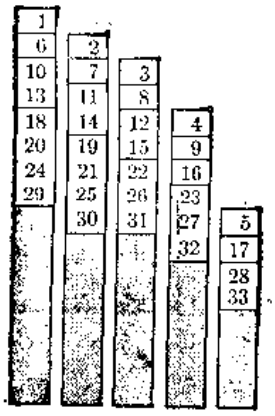
$$\sum_{1\leqslant k<n}\left[\begin{matrix} k \\ k-r \end{matrix}\right]k=\left[\begin{matrix} n \\ n-r-1 \end{matrix}\right]$$

的事实。(c) 把 $r+1$ 加到均值上，并且利用 $\sum_{n\geqslant 0}\left[\begin{matrix} n+r \\ n \end{matrix}\right](n+r)/(n+r+1)!=1$ 这一事实，得到 $\sum_{n\geqslant 0}\left[\begin{matrix} n+r \\ n \end{matrix}\right]/(n+r-1)!$。

27. 对于多路合并，问题比较小，因为 p 保持为常数，而且在每个文件上诸记录被顺序地处理；但当形成初始路段时，我们最好依照记录的长度来改变内存中记录的个数。利用如 2.5 节所述的动态分配策略，我们能保持装满整个内存那样多记录的一个堆。M. A. 戈茨〔*Proc. AFIPS Joint Computer Conf.* 25 (1964), 602-604〕已经提出了另一个方法．把每个记录分成为被链接在一起的固定大小的部分；它们在树叶处占有空间，但是仅仅前导部分参加比赛。

5.4.2 节

1.



2．在头一个合并阶段之后，所有剩下的虚拟路段都在带T上，而且它们至多有$a_n-a_{n-1}\leqslant a_{n-1}$个。因此在第二次合并阶段期间它们全都不出现。

3．我们有（D[1]，D[2]，…，D[T]）$=(a_n-a_{n-p},\ a_n-a_{n-p+1},\cdots,\ a_n-a_n)$，所以这个条件从诸$a$是非减的这一事实得出。这一条件对于这个算法的正确性是重要的，因为在步骤D2和D3中D[$j+1$]减值的次数决不比D[j]减值的次数更多。

4．由于（3），$(1-z-\cdots z^5)a(z)=1$，而且$t(z)=\sum_{n\geqslant 1}(a_n+b_n+c_n+d_n+e_n)z^n=(z+\cdots+z^5)a(z)+(z+\cdots+z^4)a(z)+\cdots+za(z)=(5z+4z^2+3z^3+2z^4+z^5)\cdot a(z)$[参考（16）]。

5．设$g_p(z)=(z-1)f_p(z)=z^{p+1}-2z^p+1$，而且设$h_p(z)=z^{p+1}-2z^p$，设$\phi^{-1}>\epsilon>0$；鲁彻（Rouche）定理告诉我们，倘若在圆上$|h_p(z)|>|h_p(z)-g_p(z)|=1$，则$h_p(z)$和$g_p(z)$在圆$|z|=1+\epsilon$内有相同个数的根。我们有$|h_p(z)|\geqslant(1+\epsilon)^p(1-\epsilon)>(1+\phi^{-1})^2(1-\phi^{-1})=1$。因此$g_p$有绝对值$\leqslant 1$的$p$个根。它们是不同的，因为$\gcd(g_p(z),\ g_p'(z))=\gcd(g_p(z),\ (p+1)z-2p)=1$[$AMM$67(1960)，745-752]。

6．设$c_0=-\alpha p(\alpha^{-1})/q'(\alpha^{-1})$，则对于某个$R>|\alpha|^{-1}$，$p(z)/q(z)-c_0/(1-\alpha z)$在$|z|\leqslant R$中解析；因此在$p(z)/q(z)$中$z^n$的系数是$c_0\alpha^n+O(R^{-n})$。于是，$\ln S=n\ln\alpha+\ln c_0+O(\alpha R)^{-n})$；而且$n=(\ln S/\ln\alpha)+O(1)$蕴涵着$O((\alpha R)^{-n})O(S^{-1})$。类似地，设$c_1=\alpha^2p(\alpha^{-1})/q'(\alpha^{-1})^2$，$c_2=-\alpha p'(\alpha^{-1})/q'(\alpha^{-1})^2+\alpha p(\alpha^{-1})q''(\alpha^{-1})/q'(\alpha^{-1})^3$，并考察$p(z)/q(z)^2-c_1/(1-\alpha z)^2-c_2/(1-\alpha z)$。

7．设$\alpha_p=2(1-t)$，$z=2^{-p-1}$，则$z=t(1-t)^p$且这个级数（算法4.7R）的反演给出用z表示的t的一个幂级数，产生渐近展开$\alpha_p=2-2^{-p}-p2^{-2p-1}+O(p^2 2^{-3p})$。

注：由此得出，当p增加时，习题6中的量ρ变成近似于$\log_4 S$。类似地，当带的量很大时，对于表5和表6两者，系数C趋于$1/((\phi+2)\ln\phi)$。

8．显然，对于$m<0$，$N_0^{(p)}=1$，$N_m^{(p)}=0$，而且考虑对于头一个和式的不同可能性，当$m>0$时，我们有$N_m^{(p)}=N_{m-1}^{(p)}+\cdots+N_{m-p}^{(p)}$。因此$N_m^{(p)}=F_{m+p-1}^{(p)}$。[Lehrbuch der Combinatorik Leipzig:Teubner，1901)，136-137]。

9．如果有0的话，考虑最左边的那个0的位置；我们求得$K_m^{(p)}=F_{m+p}^{(p)}$。

注：在这样的0和1的序列和习题8中考虑的$m+1$的表示之间，有一个简单的一一对应：把一个0放置在这个序列的右端，并观察所有0的位置。

10．引理：如果$n=F_{j_1}^{(p)}+\cdots+F_{j_m}^{(p)}$是这样一个表示，而且$j_1>\cdots>j_m\geqslant p$，则我们有$n<F_{j_1+1}^{(p)}$。证明：如果$m<p$，则结论是明显的；否则设$k$是使$j_k>j_{k+1}+1$的极小者；我们有$k<p$，而且由归纳法$F_{j_{k+1}}^{(p)}+\cdots+F_{j_m}^{(p)}<F_{j_k-1}^{(p)}$，因此$n<F_{j_1}^{(p)}+\cdots+F_{j_k-1}^{(p)}\leqslant F_{j_1+1}^{(p)}$。

现在可对n用归纳法证明所述的结果。如果$n>0$，则设j是使得$F_j^{(p)}\leqslant n$的极大者。引理表明，n的每个表示必须由$F_j^{(p)}$加上$n-F_j^{(p)}$的一个表示组成。由归纳法，$n-F_j^{(p)}$有所希望形式的一个唯一表示，而且这个表示不包括所有的$F_{j-1}^{(p)},\cdots,F_{j-p+1}^{(p)}$，因为$j$是极大者。

注：在习题1.2.8-34中已经考虑了$p=2$的情况，它是由E.泽肯多夫（E.Zeckendorf）给出的[参考Simon Stevin 29(1952)，190-195]。从n的表示进到$n+1$的表示，

有一个简单的算法，它加上 0 和 1 的序列 $c_j\cdots c_1c_0$，使该序列满足 $n=\Sigma c_jF_{j+p}^{(p)}$。例如，如果 $p=3$，则我们考察右边的诸数字，把…0 变成为…1，…01 变成为…10，…011 变成为…100；然后如果需要，就进位到左边，以“…1000…”代替“…0111…”。见习题 9 中按此次序列出的 0 和 1 的序列。一个类似的数系，已为W. C. 林奇（W. C. Lunch）所研究〔*Fibonacci Quarterly* 8 (1970), 6-22〕，他发现了一个非常有趣的方法，使它既支配一个多阶段排序的分布阶段，也支配合并阶段。

12. 第 k 次幂在它的诸行上逐次包含对于级 $k-4$ 到 k 的完全分布，最大的元素在右边。

13. 对级用归纳法。

14. (a) $n(1)=1$，所以假定 $k>1$。定律 $T_{nk}=T_{n-1,\ k-1}+\cdots+T_{n-p,\ k-1}$ 表明 $T_{nk}\leqslant T_{n+1,k}$ 当且仅当 $T_{n-p,k-1}\leqslant T_{n,k-1}$。设 r 是任意正整数，并设 n' 是使 $T_{n'-r,k-1}>T_{n',k-1}$ 之极小者；则对于所有 $n\geqslant n'$，$T_{n-r,k-1}\geqslant T_{n,k-1}$，因为它对于 $n\geqslant n(k-1)+r$ 是显然的，而且否则 $T_{n-r,k-1}\geqslant T_{n'-1,k-1}\geqslant T_{n',k-1}\geqslant T_{n,k-1}$。(b) 对于 $r=n-n'$ 的同样论证表明，$T_{n'k'}<T_{nk'}$ 蕴涵对于所有 $j\geqslant 0$，$T_{n'-j,k'}\leqslant T_{n-j,k'}$；因此这个递归式蕴涵对所有 $j\geqslant 0$ 和 $k\geqslant k'$，$T_{n'-j,k}\leqslant T_{n-j,k}$。(c) 设 $l(S)$ 是使得 $\Sigma_n(S)$ 取它的极小值的最小的 n。当且仅当对所有 S，$l(S)\leqslant l(S+1)$ 时存在所希望的序列 M_n。假设 $n=l(S)>l(S+1)=n'$，使得 $\Sigma_n(S)<\Sigma_{n'}(S)$ 和 $\Sigma_n(S+1)\geqslant\Sigma_{n'}(S+1)$。有某个最小的 s' 使得 $\Sigma_n(s')<\Sigma_{n'}(s')$，而且我们有 $m=\Sigma_n(s')-\Sigma_n(s'-1)<\Sigma_{n'}(s')-\Sigma_{n'}(s'-1)=m'$，所以 $\Sigma_{1\leqslant k\leqslant m}T_{n'k}<s'\leqslant\Sigma_{1\leqslant k\leqslant m}T_{nk}$；因此有某个 $k'\leqslant m$，使得 $T_{n'k'}<T_{nk'}$。类似地，我们有 $l=\Sigma_n(S+1)-\Sigma_n(S)>\Sigma_{n'}(S+1)-\Sigma_{n'}(S)=l'$；因此 $\Sigma_{1\leqslant k\leqslant l'}T_{n'k}\geqslant S+1>\Sigma_{1\leqslant k\leqslant l'}T_{nk}$。由于 $l'\geqslant m'>m$，有某个 $k>m$，使得 $T_{n'k}>T_{nk}$，但这同 (b) 部分矛盾。

15. 这个定理已为D. A. 扎夫所证明，他的论文已在正文中引用。

16. D. A. 扎夫已经证明，输入(和输出)记录的个数是 $S\log_{T-1}S+\frac{1}{2}S\log_{T-1}\log_{T-1}S+O(S)$。

17. 设 $T=3$；$A_{11}(x)=6x^6+35x^7+56x^8+\cdots$，$B_{11}(x)=x^6+15x^7+35x^8+\cdots$，$T_{11}(x)=7x^6+50x^7+91x^8+64x^9+19x^{10}+2x^{11}$。对于 $S=144$ 的最优分布要求 T2 上的 55 个路段，而这使得 $S=145$ 的分布不能是最优的。D. A. 扎夫已经研究了这种类型的接近最优的过程。

18. 设 $S=9$，$T=3$，而且考虑下列两个型式

最优多阶段：				或者：			
T1	T2	T3	代价	T1	T2	T3	代价
0^21^6	0^21^3	—		0^11^6	0^11^3	—	
1^3	—	0^22^3	6	1^3	—	0^12^3	6
—	1^23^1	2^2	5	—	1^13^2	2^1	7
3^2	3^1	—	6	3^1	3^2	—	3
3^1	—	6^1	6	—	3^1	6^1	6
—	9^1	—	$\frac{9}{32}$	9^1	—	—	$\frac{9}{31}$

（还有另一个方法来改进“最优”的多阶段，这就是重新考虑在每个合并阶段的输出带上，应在哪里出现虚拟路段。例如，合并 $0^2 1^5$ 和 $0^2 1^3$ 的结果可以认为是 $2^1 0^1 2^1 0^1 2^1$，而不是 $0^2 2^3$。于是，仍然遗留着许多未解决的最优性问题。）

19.

级	T1	T2	T3	T4	总共	最后输出在
0	1	0	0	0	1	T1
1	0	1	1	1	3	T6
2	1	1	1	0	3	T5
3	1	2	1	1	5	T4
4	2	2	2	1	7	T3
5	2	4	3	2	11	T2
6	4	5	4	2	15	T1
7	5	8	6	4	25	T6
……						
n	a_n	b_n	c_n	d_n	t_n	T(k)
$n+1$	b_n	c_n+a_n	d_n+a_n	a_n	t_n+2a_n	T($k-1$)

20. $a(z)=1/(1-z^2-z^3-z^4)$，$t(z)=(3z+3z^2+2z^3+z^4)/(1-z^2-z^3-z^4)$。$\sum_{n\geq 1}T_n(x)z^n=x(3z+3z^2+2z^3+z^4)/(1-x(z^2+z^3+z^4))$。$D_n=A_{n-1}+1$，$C_n=A_{n-1}A_{n-2}+1$，$B_n=A_{n-1}A_{n-2}A_{n-3}+1$，$A_n=A_{n-2}A_{n-3}A_{n-4}+1$。

21. 333343333332322　3333433333323　33334333333　3333433　333323　T5

22. 当 $n \bmod 3=1$ 时 $t_n-t_{n-1}-t_{n-2}=2$，否则它是 -1（这个类斐波那契关系是由 $1-z^2-2z^3-z^4=(1-\phi z)(1-\hat{\phi}z)(1-\omega z)(1-\hat{\omega}z)$ 这一事实得出的，其中 $\omega^3=1$）。

23. 在第 n 个合并阶段的头一半期间，路段长度不是（25），而是 S_n；在第二半为 t_n，其中

$$S_n=t_{n-2}+t_{n-3}+S_{n-3}+S_{n-4},\quad t_n=t_{n-2}+S_{n-2}+S_{n-3}+S_{n-4}$$

这里，我们认为对所有 $n\leqslant 0$，$S_n=t_n=1$〔一般地，如果 u_{n+1} 是 $u_{n-1}+\cdots+u_{n-p}$ 的头 $2r$ 项的和，则我们有 $S_n=t_n=t_{n-2}+\cdots+t_{n-r}+2t_{n-r-1}+t_{n-r-2}+\cdots+t_{n-p}$；如果 v_{n-1} 是头 $2r-1$ 项之和，则我们有 $S_n=t_{n-2}+\cdots+t_{n-r-1}+S_{n-r-1}+\cdots+S_{n-p}$，$t_n=t_{n-2}+\cdots+t_{n-r}+S_{n-r}+\cdots+S_{n-p}$〕。

代替（27）和（28）的是，$A_n=(U_{n-1}V_{n-1}U_{n-2}V_{n-2}U_{n-3}V_{n-3}U_{n-4}V_{n-4})+1$，…，$D_n=(U_{n-1}V_{n-1})+1$，$E_n=(U_{n-2}V_{n-2}U_{n-3})+1$；$V_{n+1}=(U_{n-1}V_{n-1}U_{n-2})+1$，$U_n=(V_{n-2}U_{n-3}\cdot V_{n-2}U_{n-4}V_{n-4})+1$。

25.

1^{16}	1^8	—	1^8
1^{12}	1^4	R	$1^8 2^4$
1^8	—	2^4	R
……			
R	$8^1 16^1$	8^1	8^0
16^0	R	8^1	—
16^1	16^1	8^0	R
R	16^1	—	24^0

16^1	16^1	R	$24^0 32^0$
16^0	16^0	32^1	(R)

26．当 2^n 个路段被排序时，在合并时处理了 $n\cdot 2^n$ 个初始的路段；每一半的阶段（除少数例外）合并 2^{n-2} 个和重绕 2^{n-1} 个。当 2^n+2^{n-1} 个初始路段被排序时，在合并时处理了 $n\cdot 2^n+(n-1)\cdot 2^{n-1}$ 个；每一半阶段（除少数例外）合并 2^{n-3} 个或 2^{n-1} 个并且重绕 $2^{n-1}+2^{n-2}$ 个。

27．当且仅当诸分布数的最大公因子是 1 时它有效。例如，设有六条带；如果我们分布 (a, b, c, d, e) 于 T1 到 T5，其中 $a\geqslant b\geqslant c\geqslant d\geqslant e>0$，则头一阶段产生一个分布 $(a-e, b-e, c-e, d-e, e)$，而且 $\gcd(a-e, b-e, c-e, d-e, e)=\gcd(a, b, c, d, e)$（一个数集的任何公因子也整除另一数集的公因子）。这个过程减少每个阶段的路段数，直到 $\gcd(a, b, c, d, e)$ 个路段都在同一条带上为止。

〔注意，如同习题 18 中所示，在某些虚拟路段的配置下这些非多阶段的分布有时要比多阶段优越。这个性质是由 B．萨克曼大约于 1963 年首先发现的。〕

28．由 $(1, 0, 0, 0, 0)$ 开始，并且进行下列操作恰好 n 次，我们即得所有这样的 (a, b, c, d, e)：$\{a, b, c, d, e\}$ 中选择 x，并把 x 加到 (a, b, c, d, e) 的其它四个元素中的每一个上。

为证明 $a+b+c+d+e\leqslant t_n$，按归纳法，我们将证明，如果 $a\geqslant b\geqslant c\geqslant d\geqslant e$，则总有 $a\leqslant a_n$，$b\leqslant b_n$，$c\leqslant c_n$，$d\leqslant d_n$，$e\leqslant e_n$。假定这对级 n 成立，它也必对 $n+1$ 成立，因为级 $n+1$ 的诸分布是 $(b+a, c+a, d+a, e+a, a)$，$(a+b, c+b, d+b, e+b, b)$，$(a+c, b+c, d+c, e+c, c)$，$(a+d, b+d, c+d, e+d, d)$，$(a+e, b+e, c+e, d+e, e)$。

30．下表是由 J．A．莫顿森（J. A. Mortenson）计算出来的：

级	$T=5$	$T=6$	$T=7$	$T=8$	$T=9$	$T=10$	
1	2	2	2	2	2	2	M_1
2	4	5	6	7	8	9	M_2
3	4	5	6	7	8	9	M_3
4	8	8	10	12	14	16	M_4
5	10	14	18	17	20	23	M_5
6	18	20	26	27	32	31	M_6
7	26	32	46	47	56	42	M_7
8	44	53	74	82	92	92	M_8
9	68	83	122	111	138	139	M_9
10	112	134	206	140	177	196	M_{10}
11	178	197	317	324	208	241	M_{11}
12	290	350	401	488	595	288	M_{12}
13	466	566	933	640	838	860	M_{13}
14	756	917	1371	769	1064	1177	M_{14}

5.4.3 节

1. 表 5.4.2-6 中，比较了处理每个记录的平均次数，当有 6、7 或 8 条带时，带分开的多阶段排序是最好的。

2. 当初始路段数是一个斐波那契数时，这两个方法实际上是等同的，但是在其它情况下多阶段分布虚拟路段的方式更好些。级联算法把 1 放在 T1 上，然后放 1 于 T2 上，放1于 T1 上，放 2 于 T2 上，放 3 于 T1 上，放 5 于 T2 上，等等，而且当 $p=2$ 时，在步骤 C8 中决不会有 $D[p-1]=M[p-1]$。事实上，所有虚拟路段都在一条带上，而这是比算法 5.4.2 D 中的方法更低效的。

3.（在步骤（3.3）期间把 12 个路段放置在 T3 上之后，分布停止。）

T1	T2	T3	T4	T5	T6
1^{26}	1^{21}	1^{24}	1^{14}	1^{16}	—
1^{5}	—	1^{12}	$1^{2}2^{7}$	1^{15}	$2^{2}4^{12}$
8^{4}	$6^{2}9^{3}$	5^{2}	6^{3}	1^{1}	—
—	9^{1}	23^{1}	17^{1}	25^{1}	26^{1}
100^{1}	—	—	—	—	—

4. 归纳法（参考习题 5.4.2-28）。

5. 当有 a_n 个初始路段时，第 k 趟扫描输出长度为 a_k 的 a_{n-k} 个路段，然后输出长度为 b_k 的 b_{n-k} 个路段，等等。

6. $$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

7. $e_2e_{n-2}+e_3e_{n-3}+\cdots+e_ne_0$ 个初始路段长度（参考习题 5），也可以写成为 $a_1a_{n-3}+a_2\cdot a_{n-4}+\cdots+a_{n-2}a_0$；它是 $A(z)^2-A(z)$ 中 z^{n-2} 的系数。

8. $A(z)$ 的分母有不同的根和大于分子的次数，因此将 $A(z)=\sum q_3(\rho)/(1-\rho z)\cdot\rho(1-q_4'(\rho))$ 对于 $q_4(\rho)=\rho$ 的所有根求和。在计算 $q_3(\rho)$ 和 $q_4'(\rho)$ 中，ρ 的特殊形式是有帮助的。

9. 鉴于 $q_m(2\sin\theta_k)$ 的值，根据（8）和（12），对所有大的 n 这些公式成立。为证明对于所有 n 它们都成立，我们需要知道对于 $0\leqslant m<r$，$q_{m-1}(z)$ 是当 $q_{r-1}(z)q_m(z)$ 除以 $q_r(z)-z$ 时的商。这可以通过以下几种方法来证明：（a）使用（10）并注意相消减少了 $q_{r-1}(z)q_m(z)-q_r(z)q_{m-1}(z)$ 的次数；或者（b）习题 5 的结果蕴涵当 $z\to\infty$ 时，$A(z)^2+B(z)^2+\cdots+E(z)^2\to 0$；或者（c）对于 $B(z)$，$C(z)$ 等等的分子，可以求得一个明显的公式。

10. $E(z)=r_1(z)A(z)$；$D(z)=r_2(z)A(z)-r_1(z)$；$C(z)=r_3(z)A(z)-r_2(z)$；$B(z)=r_4(z)A(z)-r_3(z)$；$A(z)=r_5(z)A(z)+1-r_4(z)$。于是 $A(z)=(1-r_4(z))/(1-r_5(z))$〔注意，$r_m(2\sin\theta)=\sin(2m\theta)/\cos\theta$；$r_m(z)$ 是契比雪夫多项式 $(-1)^{m+1}U_{2m-1}(z/2)$〕。

11. 证明 $f_m(z)=q_{\lfloor m/2\rfloor}(z)-r_{\lceil m/2\rceil}(z)$ 和 $f_m(z)f_{m-1}(z)=1-r_m(z)$，然后用习题 10 的结果。分母的这一显式首先是由戴维. E. 弗格森发现的。

5.4.4 节

1. 当写出一个递增的路段时，在输出该路段之前首先写一个含有 $-\infty$ 的“哨兵”记录（而如果这个输出将来是要被从后向前读的，例如在最后的扫描时，则 $+\infty$ 的哨兵也应写在这个路段的末端）。对于递减的路段，交换 $-\infty$ 和 $+\infty$ 的作用。

2. 级 $n+1$ 上的最小数，等于级 n 上的最大数；因此不管在任何特定的行中我们排列诸数的方式如何，诸列是非减的。

3. 事实上，在合并过程中，在 T2～T6 上的头一个路段将总是递减的，而在 T1 上将总是递增的（由归纳法）。

4. 在第二和第三阶段要求若干个“复写”操作；近似的额外代价是 $(\log_2 2)/(\log_2 \rho)$ 次扫描，其中 ρ 是表 5.4.2-1 中的“增长率”。

5. 如果 α 是一个串，则令 α^R 表示它的左—右反向串。

级	T1	T2	T3	T4	T5
0	0	—	—	—	—
1	1	1	1	1	1
2	12	12	12	12	2
3	1232	1232	1232	232	32
4	12323432	12323432	232343	2323432	3432
…	……	……	……	……	……
n	A_n	B_n	C_n	D_n	E_n
$n+1$	$B_n(A_n^R+1)$	$C_n(A_n^R+1)$	$D_n(A_n^R+1)$	$E_n(A_n^R+1)$	A_n^R+1

2				
3	2			
4	3	2		
3	4	3		
4	3	4	2	
5	4	3	3	
4	5	4	4	
3	4	5	3	
2	3	4	4	2
3	2	3	5	3
4	3	2	4	4
3	4	3	3	3
2	3	4	2	4
3	2	3	3	5
2	3	2	4	4
1	2	3	3	3

$$E_n=A_{n-1}^R+1$$
$$D_n=A_{n-2}^R A_{n-1}^R+1$$
$$C_n=A_{n-3}^R A_{n-2}^R A_{n-1}^R+1$$
$$B_n=A_{n-4}^R A_{n-3}^R A_{n-2}^R A_{n-1}^R+1$$
$$A_n=A_{n-5}^R A_{n-4}^R A_{n-3}^R A_{n-2}^R A_{n-1}^R+1=n-Q_n$$

其中

$$Q_n^R=Q_{n-1}(Q_{n-2}+1)(Q_{n-3}+2)(Q_{n-4}+3)(Q_{n-5}+4),\quad n\geqslant 1$$

$Q_0=$‘0’，而且对于 $n<0$，Q_n 是零。我们可以不用 5.4.2-12，而考虑一个双向无穷的串，在接近于它的中心处，串包含 Q_1^R，Q_2，Q_3^R，Q_4，Q_5^R 等等。

这些串 A_n，B_n 等等，包含和 5.4.2 节中对应的串相同的项，但是按另一种次序。注意，相邻的合并数总差 1。当且仅当一个初始路段是 A 时，它的合并数是偶数，是 D 时它的合并数为奇数。诸如算法 5.4.2D 那样的简单分布方案在把虚拟路段放置于高合并数位

置中时，不是十分有效的；因此，在阶段1和阶段2之间计算 Q_n 大概是有利的，为的是帮助控制虚拟路段的设置（假定 S 足够小，使得内存能毫无问题地容下 Q_n）。

6. $y^{(4)}=(+1,\ +1,\ -1,\ +1)$

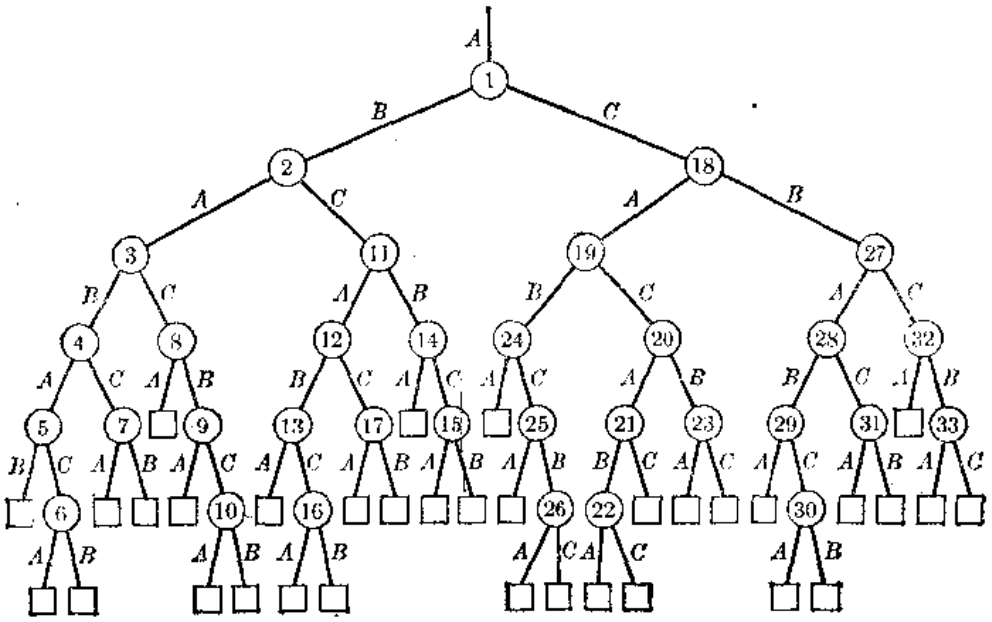
$y^{(3)}=(+1,\ 0,\ -1,\ 0)$

$y^{(2)}=(+1,\ -1,\ +1,\ +1)$

$y^{(1)}=(-1,\ +1,\ +1,\ +1)$

$y^{0}=(1,\ 0,\ 0,\ 0)$

7.



顺便说一句，34显然是这样一个最小的斐波那契数 F_n，对于它，多阶段排序不产生三条带上的 F_n 个初始路段的最优向后读合并。这株树的外部路径长度是178，它比多阶段胜2。

8. 对于 $T=4$，具有外部路径长度13的树不是 T 后进先出的，而且具有外部路径长度14的每株树包括一个一路合并。

9. 按习题2.3.4.5-6的结果，我们可以考虑一株完备的（$T-1$）叉树；"最后"的内部节点的次数处于2和 $T-1$ 之间。当有 $(T-1)^2-m$ 个外部节点时，它们中的 $\lfloor m/(T-2)\rfloor$ 个在级 $q-1$ 上，其余的则在级 q 上。

11. 对初始路段数用归纳法，可证其为真。如果存在一个具有 S 个路段的正确分布，而且有两个相邻的路段取同一方向，则有一个 $<S$ 个路段的分布；但当 $S=1$ 时不存在。

12. 条件（a）、（b）是明显的。如果对于某条带名 A 和某个 $i<j<k$，（4）中的两个配置之一存在，则由前序的定义，节点 j 必然是在节点 i 之下和节点 k 左边的一株子树中。因此"$j-l$"的情况不能存在，而且 A 必是"特殊"名，因为它出现在一支外部分枝上。但这同这样一个事实矛盾，就是：假定特殊名在节点 i 之下最左边的分枝上。

13．现在编号为4，7，11，13的节点可以是外部的，以代替一路合并（这给出比多阶段树高1的外部路径长度）。

15．设带名为A、B或C。我们将构造若干种类的树，在植物学上这些树由它们的根和叶（外部节点）结构来进行标识：

类型$r(A)$，　根A。

类型$s(A, C)$，根A，无C叶。

类型$t(A)$，　根A，无A叶。

类型$u(A, C)$，根A，无C叶，无复合B叶。

类型$v(A, C)$，根A，无C叶，无复合A叶。

类型$w(A, C)$，根A，无A叶，无复合C叶。

一片“复合叶”是这样一片叶，它的兄弟不是一片叶，我们可以首先生成一株$s(B, C)$类型的左子树，然后生成如$r(C)$类型的右子树来生成一个3后进先出类型的$r(A)$树。类似地，类型$s(A, C)$从类型$s(B, C)$和$t(C)$得出；类型$u(A, C)$从类型$v(B, C)$和$w(C, B)$得出；类型$v(A, C)$从类型$u(B, C)$和$w(C, A)$得出。我们可以生成一个3后进先出类型的$t(A)$树，其左子树是$u(B, A)$，其右子树是类型$s(C, A)$，办法是首先让左子树生长，但它的（非复合）C叶和它的右子树除外；而这时，左子树仅有A叶和B叶，所以我们可以生出整个树的右子树，然后长出左子树的A叶，最后生成左右子树。类似地，一株$w(A, C)$类型的树可以从一个$u(B, A)$和一个$v(C, A)$构造出来〔习题7的树是一株以这样的方式构造的$r(A)$树〕。

令$r(n)$，…，$w(n)$表示按上述过程构造出来的相应类型的所有n叶树的极小外部路径长度，我们有$r(1)=s(1)=u(1)=0$，$r(2)=t(2)=w(2)=2$，$t(1)=v(1)=w(1)=s(2)=u(2)=v(2)=\infty$；而且对于$n\geqslant 3$

$$r(n)=n+\min_k(s(k)+r(n-k)),\quad u_n=n+\min_k(v(k)+w(n-k)),$$

$$s(n)=n+\min_k(s(k)+t(n-k)),\quad v_n=n+\min_k(u(k)+w(n-k)),$$

$$t(n)=n+\min_k(u(k)+s(n-k)),\quad w_n=n+\min_k(u(k)+v(n-k))$$

由此得出，对所有的n，$r(n)\leqslant s(n)\leqslant u(n)$，$s(n)\leqslant v(n)$，以及$r(n)\leqslant t(n)\leqslant w(n)$；进而，$s(2n)=t(2n+1)=\infty$（后者是不证自明的）。

令$A(n)$是由规则$A(1)=0$，$A(2n)=2n+2A(n)$，$A(2n+1)=2n+1+A(n)+A(n+1)$定义的函数；则对所有$n\geqslant 2$，$A(2n)=2n+A(n-1)+A(n+1)-(0$或$1)$。设$C$是一个常数，使得对于$4\leqslant n\leqslant 8$。

i）n为偶数蕴涵$w(n)\leqslant A(n)+C_n-1$。

ii）n为奇数蕴涵$u(n)$和$v(n)\leqslant A(n)+C_n-1$。

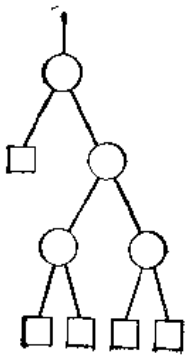
（这实际上对于所有$C\geqslant\frac{5}{6}$都有效。）则通过适当选择k为$\lfloor n/2\rfloor\pm 1$，归纳法论证表明，对于所有$n\geqslant 4$，这些关系成立。但是$A(n)$是当$T=3$时（9）中的下界，且$r(n)\leqslant\min(u(n), v(n), w(n))$，因此我们已经证明$A(n)\leqslant\hat{K}_3(n)\leqslant r(n)\leqslant A(n)+\frac{5}{6}n-1$〔常数$\frac{5}{6}$在这里可以减低〕。

17．

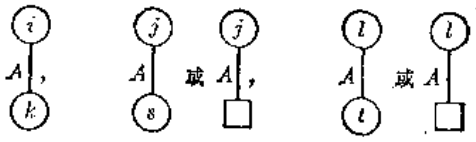
级	T1	T2	T3	T4	T5
0	1	0	0	0	0
1	5	4	3	2	1
2	55	50	41	29	15
⋯⋯					
n	a_n	b_n	c_n	d_n	e_n
	$4a_n+4b_n+3c_n+2d_n+e_n$		$2a_n+2b_n+2c_n+2d_n+e_n$		
$n+1$	$5a_n+4b_n+3c_n+2d_n+e_n$	$3a_n+3b_n+3c_n+2d_n+e_n$		$a_n+b_n+c_n+d_n+e_n$	

为了在初始分布期间从级 n 到达级 $n+1$，分别插入具有（4，4，3，2，1）个路段的 k_1 个“子级”，具有（4，3，3，2，1）个路段的 k_2 个“子级”，具有（3，3，2，2，1）个路段的 k_3 个“子级”，具有（2，2，2，1，1）个路段的 k_4 个“子级”，具有（1，1，1，1，0）个路段的 k_5 个“子级”；以上每对括号内的 5 组路段被分别附加到带（T1，T2，…，T5）上，其中 $k_1\leqslant a_n$，$k_2\leqslant b_n$，$k_3\leqslant c_n$，$k_4\leqslant d_n$，$k_5\leqslant e_n$〔如果 $(k_1,\ \cdots,\ k_5)=(a_n,\ \cdots,\ e_n)$，则我们已经达到级 $n+1$〕。必要时加虚拟路段以充满一个子级，然后从（T1，…，T5）合并 $k_1+k_2+k_3+k_4+k_5$ 个路段到 T6，从（T1，…，T4）合并 $k_1+\cdots+k_4$ 个到 T_5，…，从 T1 合并 k_1 个到 T2；然后从（T2，…，T6）合并 k_1 个到 T1，从（T3，…，T6）合并 k_2 个到 T2，…，从 T6 合并 k_5 个到 T5〔这个方法大约于 1962 年使用于 UNIVAC Ⅱ 排序程序，并且在该年的 ACM 排序讨论会上宣读〕。

19.

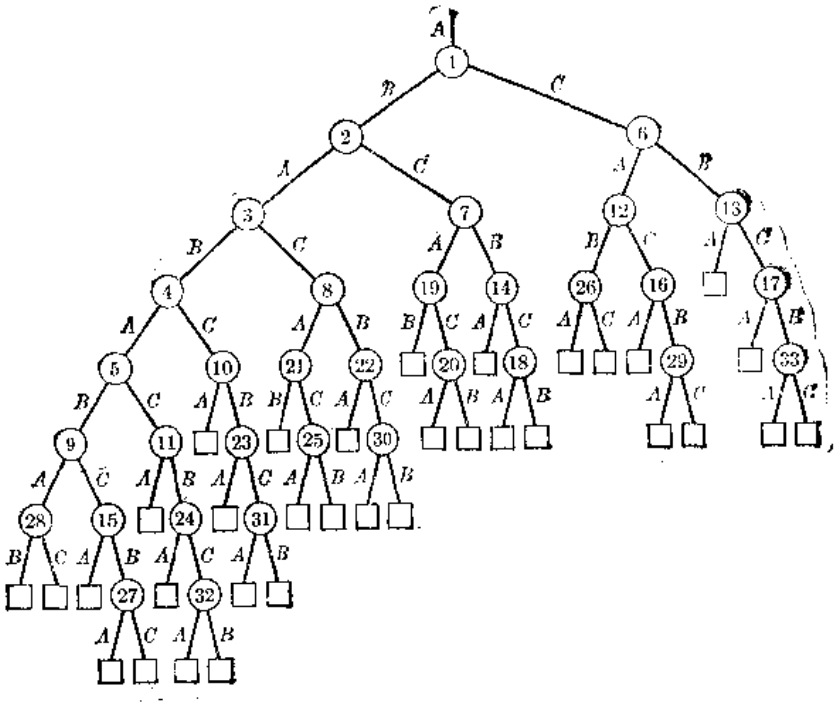


20. 一个强 T 先进先出树有一组 T 先进先出标号，其中不存在分别具有如下形式



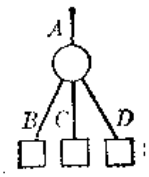
的三个分枝，这里 A 是某个带名，且 $i<j<k<l<s$。非形式地说，当我们“长出”一个 A 时，在建立任何新的 A 之前，必须长出所有其它的 A。

21.



它是非常弱的先进先出。

22. 对于通过以例如



(对于某些固定的带名A，B，C，D）逐次地代替例如

的所有出现，形成的任何树表示会出现这种情况。因为所有这些出现为相同型式所代替，后进先出或先进先出次序在这个树结构中并不造成差别。

借助于向量模型叙述这个条件：每当（$y^{(k+1)}\neq y^{(k)}$ 或 $k=m$）且 $y_i^{(k)}=-1$ 时，我们有 $y_i^{(k)}+\cdots+y_i^{(1)}+y_i^{(0)}=0$。

23.（a）假定 $v_1\leqslant v_2\leqslant\cdots\leqslant v_T$；"级联"阶段 $(1,\cdots,1,-1)^{v_T}(1,\cdots,1,-1,0)^{v_{T-1}}\cdots(1,-1,0,\cdots,0)^{v_2}$ 把 $C(v)$ 放入 v 中。（b）直接地，因为对于所有 k，$C(v)_k\leqslant C(w)_k$。（c）如果在 q 个阶段得到 v，则对于某个单位向量 u，以及某些其它向量 $u^{(1)}$，…我们有 $u\to u^{(1)}\to\cdots\to u^{(q)}=v$。因此 $u^{(1)}\preceq C(u)$，$u^{(2)}\preceq C(C(u))$，…，$v\preceq C^q(u)$。因此 $v_1+\cdots+v_T$ 小于或等于 $C^q(u)$ 的元素之和；而后者在级联合并中得到〔这个定理推广了习题5.4.3-4的结果。不幸，象这里所定义的"阶段"的概念，似乎没有任何实际的意义〕。

24. 设 $y^{(m)}\cdots y^{(l+1)}$ 是把 w 约化为 v 的一个阶段。如果对于某个 $k<i-1$，$y_j^{(i)}=-1$，$y_j^{(i-1)}=0$，…，$y_j^{(k+1)}=0$，以及 $y_j^{(k)}=-1$，则我们可以插入 $y^{(k)}$ 于 $y^{(i)}$ 与 $y^{(i-1)}$ 之间。重复这个操作直到在每列中所有的（-1）都是相邻的为止。然后如果 $y_j^{(i)}=0$ 和 $y_j^{(i-1)}\neq 0$，则有可能置 $y_j^{(i)}\leftarrow 1$；最后，每个列由一些 $+1$，后边接上一些 -1，再接上一些 0 组成，我们已经构造了一个阶段，对于某个 $w'\succeq w$，它把 w' 归结为 v。这个阶段对各列进行排列，它具有形式 $(1,\cdots,1,-1)^{a_T}\cdots(1,-1,0,\cdots,0)^{a_2}(-1,0,\cdots,0)^{a_1}$。$T-1$ 个关系的序列

$$
\begin{aligned}
(x_1,\cdots,x_T)&\preceq(x_1+x_T,\cdots,x_{T-1}+x_T,\ 0)\\
&\preceq(x_1+x_{T-1}+x_T,\cdots,x_{T-2}+x_{T-1}+x_T,\ x_T,\ 0)\\
&\preceq(x_1+x_{T-2}+x_{T-1}+x_T,\cdots,x_{T-3}+x_{T-2}+x_{T-1}+x_T,\ x_{T-1}+x_T,\ x_T,\ 0)\\
&\preceq\cdots\\
&\preceq(x_1+x_2+x_3+\cdots+x_T,\ x_3+\cdots+x_T,\cdots,x_{T-1}+x_T,\ x_T,\ 0)
\end{aligned}
$$

表明诸 a 的最好选择是 $a_T=v_T$，$a_{T-1}=v_{T-1}$，…，$a_2=v_2$，$a_1=0$。而且如果排列诸列使得 $v_1\leqslant\cdots\leqslant v_T$，则结果是最好的。

25.（a）假设 $v_{T-k+1}\geqslant\cdots\geqslant v_T\geqslant v_1\geqslant\cdots\geqslant v_{T-k}$，并使用 $(1,\cdots,1,-1,0,\cdots,0)^{v_{T-k+1}}\cdots(1,\cdots,1,0,\cdots,0,-1)^{v_T}$。（b）对于 $1\leqslant l\leqslant T-k$，$D_k(v)$ 的 l 个最大元素之和是 $(l-1)s_k+s_{k+l}$。（c）如果在一个使用 k 个输出带的阶段中 $v\Rightarrow w$，则我们显然可以假定该阶段有 $(1,\cdots,1,-1,0,\cdots,0)^{a_1}\cdots(1,\cdots,1,0,\cdots,0,-1)^{a_k}$ 的形式，且其它 $T-k$ 条带的每一条用作每个操作的输入。选择 $a_1=v_{T-k+1}$，…，$a_K=v_T$ 是最好的。（d）见习题 22(c)。我们总有 $k_1=1$；而且 $k=T-2$ 总是胜过 $k=T-1$，因为我们假定至少 v 的一个分量为 0。因此对于 $T=3$，有 $k_1\cdots k_q=1^q$ 和初始分布 $(F_{q+1},F_q,0)$。对于 $T=4$，找到的不占优势的策略及其对应的分布是

$q=2$　12(3, 2, 0, 0)

$q=3$　121(5, 3, 3, 0)；122(5, 5, 0, 0)

$q=4$　1211(8, 8, 5, 0)；1222(10, 10, 0, 0)；1212(11, 8, 0, 0)

$q=5$　12121(19, 11, 11, 0)；12222(20, 20, 0, 0)；12112(21, 16, 0, 0)

$q=6$　122222(40, 40, 0, 0)；121212(41, 30, 0, 0)

$q\geqslant 7$　$12^{q-1}(5\cdot 2^{q-3},\ 5\cdot 2^{q-3},\ 0,\ 0)$

故对于 $T=4$ 和 $q\geqslant 6$，极小阶段合并同平衡合并相似，只是在末尾处稍有曲折（从(3, 2, 0, 0) 进行到 (1, 0, 1, 1) 而不是 (0, 0, 2, 1))。

当 $T=5$ 时，不占优势的策略对于 $q=2n\geqslant 2$ 是 $1(32)^{n-1}2$，$1(32)^{n-1}3$；对于 $q=2n+1\geqslant 3$ 是 $1(32)^{n-1}32$，$1(32)^{n-1}22$，$1(32)^{n-1}23$（所列的头一个策略在它的分布中有最多的路段)。在六条带上，它们是 13 或 14，142 或 132 或 133，1333 或 1423，然后对于 $q\geqslant 5$ 为 13^{q-1}。

5.4.5 节

1. 下列算法为一个表 $A[L-1]\cdots A[1]A[0]$ 所控制，这个表基本上表示基数 P 记号下的一个数。当对这个数重复加 1 时，“进位”告诉我们何时合并。诸带被编号成从

0到P。

01.〔初始化〕 置（$A[L-1]$，…，$A[0]$）←（0，…，0）和$q\leftarrow 0$（在本算法期间，q将等于（$A[L-1]+\cdots+A[0]$）$\bmod T$）。

02.〔分布〕 以递增次序写出带q上的一个初始路段，置$l\leftarrow 0$。

03.〔加1〕 如果$l=L$，则停止；输出在带（$-L$）$\bmod T$上，当且仅当L是偶数时次序为递增的。否则置$A[l]\leftarrow A[l]+1$，$q\leftarrow(q+1)\bmod T$。

04.〔进位？〕 如果$A[l]<P$，则返回02。否则合并到带（$q-l$）$\bmod T$上，置$A[l]\leftarrow 0$和$q\leftarrow(q+1)\bmod T$，l增1，并且返回03。

2．随时记住在每条带上有多少个路段。当输入穷尽时，必要时增加虚拟路段，直到达到在每条带上至多有一个路段和至少一条带是空的为止。然后在另外一个合并中完成排序，必要时首先重绕某些带（有可能从A表导出诸路段的方向）。

3．

OP	T0	T1	T2	OP	T0	T1	T2
分布	—	A_1	A_1A_1	分布	D_2A_1	A_1	A_4
合并	D_2	—	A_1	合并	D_2	—	A_4D_2
分布	D_2A_1	—	A_1	合并	—	A_4	A_4
合并	D_2	D_2	—	分布	—	A_4	A_4A_1
分布	D_2	D_2A_1	A_1	复写	—	A_4D_1	A_4
合并	D_2D_2	D_2	—	复写	—	A_4	A_4A_1
合并	D_2	—	A_4	合并	D_5	—	A_4

这时T2将被重绕，而最后的合并将完成此排序。为了避免无用的复写操作，以下列B4′代替B4：

B4′〔准备好合并了？〕如果A〔$l-1$，q〕$=s$，则转到步骤B5。否则如果输入被穷尽，且A〔$l-1$，q〕$-s$是一个正偶数而且A〔$l-1$，$(2r-q)\bmod T$〕$\neq$A〔$l-1$，q〕$-q$，则置$s\leftarrow$A〔$l-1$，q〕，并转B5。否则转回到步骤B3。

〔这种改变避免了路段简单地向后和向前移的状态。步骤B5中头一个带圆括号的注解不再是严格地正确的，但是这个算法仍然正确地工作，因为在级l上将不查看A表的条款。受此影响的最小的S是P^3+1。当P很大时，这种改变几乎不造成大的差别，但在某些环境下，它避免使计算机看起来太笨了。这个算法还应该加以改变，使之更有效地处理$S=1$的情况。〕

4．事实上，我们不必在步骤B1中置$A[0,0]$，也不必在步骤B3和B5中置$A[l,q]$〔但是在步骤B3中置$A[l,r]$是必须的〕。

5．对于某个$k>0$，$P^{2k}-(P-1)P^{2k-2}<S\leqslant P^{2k}$。

5.4.6 节

1．$\lfloor 23000480/(n+480)\rfloor n$。

2．在所示时刻，该缓冲区中的所有记录都已经移至输出。步骤F2坚持，在合并时在测试“输入缓冲区是否满？”之前测试“输出缓冲区是否满？”，否则我们将会遇到麻烦（除非作了习题4的改变）。

3．否，例如，如果对于 $1 \leqslant i \leqslant P$，文件 i 包含键 i，$i+P$，$i+2P, \cdots$，则我们可以达到 P 个缓冲区 $1/P$ 满、和 $P-1$ 个缓冲区全满的状态。这个例子表明，为了连续进行输出，即使允许同时读，$2P$ 个输入缓冲区是必要的，除非我们重新分配内存作部分缓冲区或者以某种方式使用“散列读”〔实际上，如果诸块包含少于 $P-1$ 个记录，则仅需少于 $2P$ 个缓冲区，但这是不可能的。较老的计算机将读一个数据块到独立于计算机内存的一个特殊的硬件缓冲区中，而在这个缓冲区的内容被传送到内存时，计算将停止。在这样一种情况下，内存中的 $2P-1$ 个“软件”缓冲区将是足够的〕。

4．早一点给 S 赋值（在步骤 F1 和 F4 而不是 F3）。

5．例如，如果所有文件的所有键都相等，则在预报时我们不能简单地作任意的决断；预报必须同合并处理所作的决断相容。一种安全的方式，是在步骤 F1 和 F4 中求最小的 m，即每当 $i<j$ 时认为取自文件 $C[i]$ 的一个记录小于在文件 $C[j]$ 上有相同键的所有记录（实质上，文件号被附加在键上）。

6．在步骤 C1 中，也置 TAPE$[T+1] \leftarrow T+1$。在步骤 C8 中，应该合并到 TAPE$[p+2]$ 上而不是 TAPE$[P+1]$ 上。在步骤 C9 中，置（TAPE[1]，…，TAPE$[T+1]$）$\leftarrow$（TAPE[1]，…，TAPE[1]）。

7．在图表 A 中使用的方法是 $(A_1D_1)^4A_0D_0(A_1D_1)^2A_0D_0(A_1D_1)^3A_0, D_1(A_1D_1)^4A_0D_0(A_1D_1)^3A_0D_0\alpha A_1D_0A_0$，$D_1A_0D_0(A_1D_1)^3A_0D_0\alpha A_1D_1A_0$，$D_1A_1D_1\alpha A_1D_1A_0$，其中 $\alpha=(A_0D_0)^2A_1D_1A_0D_0(A_1D_1)^2(A_0D_0)^7A_1D_1(A_0D_0)^3A_1D_1A_0D_0$。头一个合并阶段写 $D_0A_0D_0A_1D_1A_4D_4A_0D_0A_1D_1A_1D_1A_4D_4A_0D_0A_1D_1A_0D_0(A_1D_1)^4$ 于带 5 上；其次写 $A_4D_4A_4D_4A_1D_1A_4D_4A_0D_0A_1D_1A_1D_1A_7$ 于带 1 上；其次写 $D_{13}A_4D_4A_0D_0A_{10}$ 于带 4 上。最后的阶段是

$A_4D_4A_4$	—	$D_{19}A_3D_3A_{12}$	$D_{13}A_4D_4A_4$	D_0A_3
A_4	$D_{23}A_{11}$	$D_{19}A_3$	$D_{13}A_4$	—
—	D_{23}	D_{19}	D_{13}	D_{22}
A_{77}	—	—	—	—

8．否。因为至多节约 S 次停止/启动，而且因为输入带（不是输出带）的速度总是要支配初始分布时间的，图表 A 中使用的分布方案的其它优点远远抵消了这微小的缺点。

9．$P=5$，$B=8300, B'=734, S=\lceil(3+1/P)N/6P'\rceil+1=74$，$\omega=1.094$，$\alpha=0.795$，$\beta=-1.136$，$\alpha'=\beta'=0$；（9）$=855$ 秒，我们对它们附加初始重绕的时间后总共为 958 秒。合并中节省大约一分钟，不足以补偿由于初始重绕和换带所损失的时间（除非也许我们处于一个多道程序设计环境中）。

10．在标准多阶段合并期间，重绕超过该文件的大约 54%（表 5.4.2-1 中的“扫描/阶段”行），而且由习题 5.4.3-5 和等式 5.4.3-13，在标准级联合并期间最长的重绕近似地包括文件的 $a_ka_{n-k}/a_n \approx (4/(2T-1))\cos^2(\pi/(4T-2)) < \frac{4}{11}$。

11．仅仅初始和最后的重绕利用到“高速”的性能，因为当这个磁带卷包含整个举例文件时它仅仅比 10/23 满一点。利用例 8 中的 $\pi=\lceil .946\ln S-1.204\rceil$，$\pi'=1/8$，我们得到对于例 1-9 的下列估计的总数，分别为：

1115，1296，1241，1008，1014，967，891，969，856

12．（a）使用$4P+4$个缓冲区的一个显然的解决方法，就是简单地从成对的带同时读和写。但是注意，三个输出缓冲区是足够的（在一个给定的时刻，我们从一个缓冲区执行第二半写，从另一缓冲区执行头一半写，而输出到第三个缓冲区），而且这对于输入缓冲区状态提出了相应的改进。使用一项稍微减弱些的“预报”技术可以证明，$3P$个输入缓冲区和3个输出缓冲区是必要的和充分的。J. 休（J. Sue）提出了一个更简单和更优越的方法，它把一个“向前看的键”加到每一个块区中，以指明后继块的最后键。J. 休的方法要求$2P+1$个输入缓冲区和$4P$个输出缓冲区，而且这是对算法F的一种直截了当的修改。

（b）在这种情况下，由于α的值很大，我们必须对数据进行五到六次扫描，它消除了双重快速合并的优点。这个思想在八条或九条带上实现时效果要好得多。

13．否，例如考虑正好在$A_{16}A_{16}A_{16}A_{16}$之前的状态。但可处理两个满卷。

14．

$$\frac{\det\begin{pmatrix} 0 & -p_0z & 0 & z-1 \\ 0 & 1-p_1z & -p_0z & z-1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}{\det\begin{pmatrix} 1-p\geqslant 1^z & -p_0z & 0 & z-1 \\ -p\geqslant 2^z & 1-p_1z & -p_0z & z-1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}}$$

15．A矩阵有形式

$$A=\begin{pmatrix} B_{10}z & B_{11}z & \cdots & B_{1n}z & 1-z \\ \vdots & & & & \vdots \\ B_{n0}z & B_{n1}z & \cdots & B_{nn}z & 1-z \\ 0\cdots 0 & 1 & 0 & 0 & \\ 0\cdots 0 & 0 & 0 & 0 & \end{pmatrix},\quad \begin{matrix} B_{10}+B_{11}+\cdots+B_{1n}=1 \\ \vdots \\ B_{n0}+B_{n1}+\cdots+B_{nn}=1 \end{matrix} \tag{11}$$

因此

$$\det(I-A)=\det\begin{pmatrix} 1-B_{10}z & -B_{11}z & \cdots & -B_{1,n-1}z & -B_{1n}z \\ \vdots & & & & \\ -B_{n0}z & -B_{n1}z & \cdots & 1-B_{n,n-1}z & -B_{nn}z \\ 0 & 0 & & -1 & 1 \end{pmatrix}$$

而我们可以把所有列都加到头一列上，然后分出因子$(1-z)$。结果$g_0(z)$有$h_0(z)/(1-z)$的形式，且$\alpha^{(0)}=h_0(1)$，因为$h_0(1)\neq 0$，且对于$|z|<1$ $\det(I-A)\neq 0$。

5.4.7 节

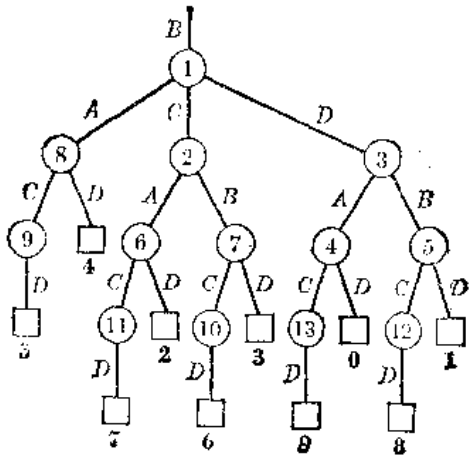
1．在基数交替地为P和$T-p$的数系中从最低位有效数字到最高位有效数字进行排序（如果把数字对偶组合在一起，则我们实际上有$P\cdot(T-P)$的一个纯基数，例如，如果$P=2$和$T=7$，则这个数系是以一种简单的方式同十进记号相关的“双五进制”）。

2．如果K是0和F_{n-1}之间的一个键，则设F_n-1-K的斐波那契表示是$a_{n-2}F_{n-1}+\cdots+a_1F_2$，其中$a_j$为0或1，而且不出现两个连续的1。在阶段$j$之后，在$a_{j-1}\cdots a_1$的递减次序下，带$j+1$（modulo 3）包含具有$a_j=0$的那些键，而带$j-1$（modulo 3）包含具有$a_j=1$的那些键。

3．〔想象具有两个袋子“0”和“1”的一个卡片排序机，并且考虑F_n张已经在$n-2$列穿上键$a_{n-2}\cdots a_1$的卡片的排序过程。把这些排成递减次序的惯常步骤是从最低位有效数字开始。这可以简化，因为我们知道每次扫描末尾时在“1”袋中的每件事物下次扫描时将转入“0”袋中。〕

4．如果在级2上有一个外部节点，则我们不能够构造这样一株好的树。否则，在级3上至多有三个外部节点，在级4上六个，因为每个外部节点应当都出现于相同的带上。

5．



6．09, 08, …, 00, 19, …, 10, 29, …, 20, 39, …, 30, 40, 41, …, 49, 59, …, 50, 60, 61, …, 99。

7．是；首先分布诸记录于越来越小的子文件上，直到得到一些单卷的文件，这些单卷的文件可以独立地排序。这对偶于下列过程：首先对诸单卷文件进行排序，而后把它们合并成越来越大的多卷文件。

5.4.8 节

1．是。文件的次序和选择树的次序交替地递增和递减，而我们实际上有一个阶P的鸡尾混合排序；参考习题9。

2．设$Z_N=Y_N-X_N$，并注意到$(N+1)NZ_{N+1}=N(N-1)Z_n+N^2+N$，即可解$Z_N$的这个递归式；

因此

$$Z_N=\frac{1}{3}(N+1)+\binom{M+2}{3}/N(N-1),\quad 对于N>M$$

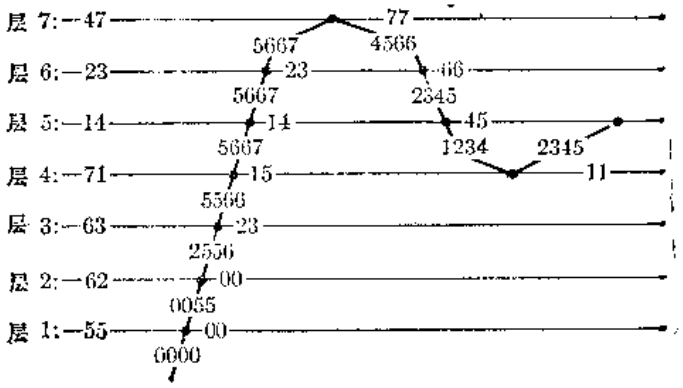
现在消去Y_N并得到

$$\frac{X_N}{N+1}=6\frac{2}{3}(H_{N+1}-H_{M+2})+2\left(\frac{1}{N+1}-\frac{1}{M+2}\right)$$
$$-\frac{2}{3}\binom{M+2}{3}\left(\frac{1}{(N+1)N(N-1)}-\frac{1}{(M+2)(M+1)M}\right)$$
$$+\frac{3M+4}{M+2},\quad N>M$$

3．是；利用类似于定理 5.3.3 的构造并用它来分划文件，即可用 $O(N)$ 步求一个中间元素。由 R. W. 弗洛伊德和 A. J. 史密斯给出的另一个有趣的方法，是在 $O(N)$ 个时间单位内合并两个 N 项的路段如下：把诸项连同它们之间的空格，布在诸带上，然后对每个空格逐个地用一个指明刚好居于该空格之前的项的最后位置的数，填到该空格内。

4．有可能把对于层 1，…，$p+1$ 的一个调度和对于层 q，…，n 的一个调度结合在一起：当前边的调度首先达到层 $p+1$ 时，向上到 q 层并进行后一个调度（利用当前的电梯内容就好象它们是在定理K的算法中额外的“0”个人那样）。在完成了该调度之后，回到层 $p+1$ 并且恢复以前的调度。

5．考虑 $b=4$，$c=2$ 及算法的下列行为



现在 2（在电梯里）小于 3（在层 3 上）。

〔在构造了这样一个例子之后，读者应该能看出怎样来揭示在定理K的证明中所需要的较弱的性质。〕

6．设 i，j 是使 $c_i<c'_i$ 和 $c_j>c'_j$ 的极小者。引进一个新的人，他要从 i 层到 j 层去。这对于任何 k 都不会增加 $\max(u_k,\ d_{k+1},\ 1)$ 或 $\max(c_k,\ c'_k)$。继续进行这过程直到对于所有 j，$c_j=c'_j$。现在注意若在步骤 K1 和 K3 中以 c_k 代替 c，则正文中的算法仍有效。

8．设这个数是 p_n，且设 Q_n 是使得对于 $1\leqslant k\leqslant n$，$u_k=1$ 的排列数，则 $p_n=Q_1p_{n-1}+Q_2p_{n-2}+\cdots+Q_nP_0$，$P_0=1$。可以证明，对于 $n\geqslant 2$，$Q_n=3^{n-2}$（见以下），因此用母函数可推出

$$\sum P_nz^n=(1-3z)/(1-4z+2z^2)=1+z+2z^2+6z^3+20z^4+\cdots$$
$$2P_n=(2+\sqrt{2})^{n-1}+(2-\sqrt{2})^{n-1}$$

为了证明 $Q_n=3^{n-2}$，考虑三进序列 $x_1x_2\cdots x_n$，使得 $x_1=2$，$x_n=0$ 且对于 $1<k<n$，有 $0\leqslant x_k<2$。下列规则定义了在这样的序列和所希望的排列 $a_1a_2\cdots a_n$ 之间的一一对应：

$$a_k=\begin{cases}\max\{j\mid(j<k\text{ 且 }x_j=0)\text{ 或 }j=1\}, & \text{如果 } x_k=0\\ k, & \text{如果 } x_k=1\\ \min\{j\mid(j>k\text{ 且 }x_j=2)\text{ 或 }j=n\}, & \text{如果 } x_k=2\end{cases}$$

(这个对应是由作者同E. A本德一起得到的。)

9. 鸡尾混合的扫描次数是 $2\max(u_1, \cdots, u_n)-(0$ 或 $1)$，因为每对扫描（左一右一左）都使得每个非0的 u 减去1。

10. 从一个分布方法（快速排序或基数交换）开始，直到得到一些单卷文件为止。而且要耐心点。

5.4.9 节

1. $\frac{1}{4}-\left(x \bmod \frac{1}{2}\right)^2$ 圆转动。

2. 对于固定的 k，q，r 和 $i\neq i'$，$k=a_{iq}$ 和 $k+1=a_{i'r}$ 的概率是 $f(q, r, k)L!L!(PL-2L)!/(PL)!$，

其中

$$f(q, r, k)=\binom{k-1}{q-1}\binom{k-q}{r-1}\binom{PL-k-1}{L-q}\binom{PL-k-1-L+q}{L-r}$$
$$=\binom{k-1}{q+r-2}\binom{q+r-2}{q-1}\binom{PL-k-1}{2L-q-r}\binom{2L-q-r}{L-q}$$

而且

$$\sum_{\substack{1\leqslant k\leqslant PL\\ 1\leqslant q,r<L}}|q-r|f(q, r, k)$$
$$=\sum_{q,r}|q-r|\binom{PL-1}{2L-1}\binom{q+r-2}{q-1}\binom{2L-q-r}{L-q}=\binom{PL-1}{2L-1}A_{2L-1}$$

对于固定的 k，q 和 i，$k=a_{iq}$ 和 $k+1=a_{i,q+1}$ 的概率是

$$g(k, q)\Big/\binom{PL}{L}, \text{ 其中 } g(k, q)=\binom{k-1}{q-1}\binom{PL-k-1}{L-q-1}$$

以及

$$\sum_{\substack{1\leqslant k\leqslant PL\\ 1\leqslant q<L}}g(k, q)=\sum_{1\leqslant q<L}\binom{PL-1}{L-1}=(L-1)\binom{PL-1}{L-1}$$

〔*SIAM J. Computing* 1 (1972), 161-166.〕

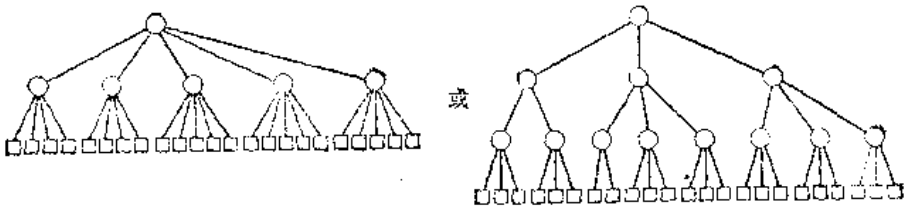
3. 在 $2\leqslant m\leqslant\min(9, n)$ 的范围内对（5）取极小值。

4. (a) $(0.000725(\sqrt{P}+1)^2+0.014)L$。(b) 把公式（5）中的"$\alpha mn+\beta n$"改为"$(0.000725)\sqrt{m}+1)^2+0.014)n$"。〔计算机的实验表明，通过这个新的递归式定义的最优树的本质，和对于 $\alpha=0.00145$，$\beta=0.01545$ 由定理K所定义的那些极其相同；

事实上，当 $30\leqslant n\leqslant 100$ 时，对于两个递归式都是最优的树是存在的。本题中提出的修改，如同正文中的例子那样，在 $n=64$ 或 100 时，节约大约 10% 的合并时间。这种风格的缓冲区分配已于 1954 年由 H·西沃德所考虑，他发现四路合并使寻找的时间极小化。〕

5. 设 $A_m(n)$，$B_m(n)$ 表示其所有的叶分别在（偶，奇）级上的一些 m 株树的集合。则 $A_1(1)=0$，$B_1(1)=\alpha+\beta$；当 $M\geqslant 2$ 时如同在（4）中那样定义 $A_m(n)$ 和 $B_m(n)$；$A_1(n)\min_{1\leqslant m\leqslant n}(\alpha mn+\beta n+B_m(n))$，$B_1(n)=\min_{1\leqslant m\leqslant n}(\alpha mn+\beta n+A_m(n))$。后边的等式是正确定义的，且则 $A_1(n)$ 和 $B_1(n)$ 是互相定义的！

6.



$A_1(23)=B_1(23)=268$。〔奇怪，$n=23$ 是仅有的 $\leqslant 50$ 的具下列性质的值，对于它，在奇偶性没有限制的情况下，任何具有 n 个叶的相等奇偶性的树都不是最优的。也许它是当 $\alpha=\beta$ 时仅有的这样的值。〕

7. 考虑在任何树中的诸量 $\alpha d_1+\beta e_1$，…，$\alpha d_n+\beta e_n$，其中 $(d_j,\ e_j)$ 是对于第 j 片叶的（叉数和，路径长度）。对于权 $w_1\leqslant\cdots\leqslant w_n$ 的一个最优树将有 $\alpha d_1+\beta e_1\geqslant\cdots\geqslant\alpha d_n+\beta e_n$。总是有可能重新安排它们使得 $\alpha d_1+\beta e_1=\cdots=\alpha d_1+\beta e_k$，其中头 k 片叶被合并在一起。

9. 设 d 极小化 $(\alpha m+\beta)/\ln m$。使用凸性的一个简单归纳法证明 $A_1(n)\geqslant(\alpha d+\beta)n\log_d n$，而且当 $n=d^t$ 时等式成立。一个适当的上界从完备的 d 叉树得到，因为对于 $n=d^t+(d-1)r$，$0\leqslant r\leqslant d^t$ 它们有 $D(\tau)=dE(\tau)$，$H(\tau)=tn+dr$。

10. 见 *Proc. ACM Symp. Theory of Computing* 6 (1974)，216-229。

11. 利用习题 1.2.4-38，当 $2.3^{q-1}\leqslant n/m\leqslant 3^q$ 时 $f_m(n)=3qn+2(n-3^q m)$；当 $3^q\leqslant n/m\leqslant 2.3^q$ 时为 $3qn+4(n-3^q m)$。于是 $f_2(n)+2n\geqslant f(n)$，而且当且仅当 $4.3^{q-1}\leqslant n\leqslant 2.3^q$ 时等式成立；$f_3(n)+3n=f(n)$；$f_4(n)+4n\geqslant f(n)$，而且当且仅当 $n=4.3^q$ 等式成立；对所有的 $m\geqslant 5$，$f_m(n)+mn>f(n)$。

12. 利用描述一，1:1，1:1:1，1:1:1:1 或 2:2，2:3，2:2:2，…，$\lfloor n/3\rfloor:\lfloor(n+1)/3\rfloor:\lfloor(n+2)/3\rfloor$，…；对于 $4.3^q\leqslant n\leqslant 4.3^{q+1}$ 这给出其所有叶在级 $q+2$ 上的树（当 $n=4.3^q$ 时，形成两株这样的树）。

14. 穷举并考察 n 的所有分划，对于 $n=1,2,3,\cdots$，可找到下列树的描述。一，1:1，1:1:1，1:1:1:1，1:1:1:1:1，1:1:1:1:1:1，1:1:1:1:3，1:1:3:3，3:3:3，1:3:3:3，3:4:4，3:3:3:3，3:3:3:4，3:3:4:4，3:4:4:4，4:4:4:4，…，5:6:6:6:12，6:6:6:6:12，6:6:6:6:13，…（这些叉数似乎总是 $\leqslant 6$，但这样一个结果看来十分难以证明）。

15. 如果 a 个人开始时上了电梯，则集中率在头一次停止时至多增加 $a+c$。当它下一次停止在最初一层时，比率至多增加 $b+c-a$。因此在 k 次停止之后，比率顶多增加 $(k-1)b+kc$。

16. 十一次停止：123456 至层 2，334466 至层 3，444666 至层 4，256666 至层 5，

466666 至层 6，123445 至层 4，112335 至层 5，222333 至层 3，122225至层 2，111555 至层 5，111111 至层 1〔这是极小的，因为由对称性，当电梯的容量任意时，停十次的一个解法，都可重新安排成依次停在层 2, 3, 4, 5, 6，p_2，p_3，p_4，p_5，1 上，其中 $p_2p_3p_4p_5$ 是{2，3，4，5}的一个排列；这样的调度只有当 $b \geqslant 8$ 时才可能。参考 Martin Gardner, *Sci. Amer.* 228(May，1973)，107〕。

17. 考虑向后进行步骤（g），分布诸记录到箱子 1 中，然后箱子 2，箱子 3。这个操作恰巧是步骤（d）正在键文件上模拟的操作。

18. 内部排序必须小心地根据分页的要求来选择；诸如谢尔的递减增量排序、地址计算、堆排序以及表排序方法都将是灾难性的，因为它们要求大量页的“工作集合”。快速排序、基数交换以及顺序分配合并或基数排序是适合于一个分页环境的更好得多的方法。

一个外部排序的设计者能做的某些事情，“实际上不可能”包括在一个自动分页的方法中。这些事情是：（a）预报下一次应该读的输入文件，使得当需要时有数据备用；（b）按照硬件和数据的特征选择缓冲区大小和合并的次序。

另一方面，如果程序员是仔细的，而且知道所使用的实际机器的性质，则一部虚拟机器相当容易编程序，且它能给出不坏的结果。见布朗（Brawn），古斯塔夫森(Gustavson)和曼金（Mankin）所作的研究〔*CACM* 13(1970) 483～494〕。

19. 至少有 $(nb)!/b!^{2n}$ 个配置，而且在 s 次停止之后由给定的一个停止可以得到的这个数至多是 $((n-1)\binom{b+c}{b})^s$，它比 $n^s2^{(b+c)s}$ 小。因此 $s>(\ln(nb)!-2n\ln b!)/(\ln n+(b+c)\ln 2)$ 并得出所述结果。

5.5 节

1. 在一个给定的状态下判定哪一个排序算法最好是困难的。

2. 对于小的N，表插入；对于中等的N，例如，$N=64$，表合并；对于大的N，基数表排序。

3.（由 V. 普拉特给出的解）给定有待合并的两个非减的路段 α，β，以一个直截了当的方式确定子路段 $\alpha_1\alpha_2\alpha_3\beta_1\beta_2\beta_3$，使得 α_2 和 β_2 正好包含 α 和 β 的键中具有整个文件中值的那些键。通过逐次的“颠倒”，首先形成 $\alpha_1\alpha_2\beta_1^R\alpha_3^R\beta_2\beta_3$，然后 $\alpha_1\beta_1\alpha_2^R\beta_2^R\alpha_3\beta_3$，然后 $\alpha_1\beta_1\alpha_2\beta_2\alpha_3\beta_3$，我们就能对于长度$\leqslant N/2$ 的子文件 $\alpha_1\beta_1$ 和 $\alpha_3\beta_3$ 递归地完成合并。

由 L. 特拉布·帕多（L. Trabb Pardo）提出的一个相当复杂的算法提供了对于这一问题最好的答案：有可能以$O(n)$ 时间进行稳定的合并和以$O(n\log_2 n)$ 时间进行稳定的排序，并且对于固定数量的下标变量仅仅使用$O(\log_2 n)$ 个二进位的辅助存储，而无须以任何方式变换诸记录。

6.1 节

1. $\sqrt{(N^2-1)/12}$。

2. **S1′**.〔初始化〕 置P←FIRST。

   S2′.〔比较〕 如果K＝KEY(P)，则算法成功地结束。

   S3′.〔前进〕 置P←LINK(P)。

S4′.〔文件结束?〕 如果P≠A，则转回S2′否则这个算法以失败告终。

3.

```
KEY       EQU     3:5
LINK      EQU     1:2
START     LDA     K              1
          LD1     FIRST          1
2H        CMPA    0,1(KEY)       C
          JE      SUCCESS        C
          LD1     0,1(LINK)      C - S
          J1NZ    2B             C - S
FAILURE   EQU     *              1 - S      (6C - 3S + 4)u
```

4．是，如果我们有方法置“KEY(A)”等于K的话〔但在程序Q′中使用的循环重复技术，在这种情况下无效〕。

5．否，程序Q总是至少做和程序Q′同样多的操作。

6．以

```
JE    *+4
CMPA  KEY+N+2, 1
JNE   3B
INC1  1
```

代替行08，而且把行03～04改变成为ENT1-2-N；3H　INC1　3。

7．注意$\overline{C}_N=\frac{1}{2}\overline{C}_{N-1}+1$。

8．欧拉求和公式给出

$$H_n^{(x)}\sim 常数+n^{1-x}/(1-x)+\frac{1}{2}n^{-x}-B_{2xn}{}^{-1-x}/2!+B_5x(x+1)n^{-2-x}/3!-\cdots$$

〔常数是$\zeta(x)$；复变理论告诉我们

$$\zeta(x)=2^x\pi^{x-1}\sin\left(\frac{1}{2}\pi x\right)\Gamma(1-x)\zeta(1-x)$$

这是当$x<0$时特别有用的一个公式。〕

9．是；事实上，$\overline{C}_N=N-N^{-\theta}H_{N-1}^{(-\theta)}=\theta N/(1+\theta)+\frac{1}{2}+O(N^{-\theta})$。

10．$p_1\leqslant\cdots\leqslant p_N$；极大值$\overline{C}_N=(N+1)-$(极小值$\overline{C}_N$)〔类似地，在不等长情况下，极大平均查找时间是$L_1(1+p_1)+\cdots+L_N(1+p_N)$减去极小平均查找时间〕。

11．(a) $f_{m-1}(x_{i_1},\ \cdots,\ x_{i_{m-1}})$ p_i的诸项分别表示在此之前的诸要求序列的概率，这些要求序列均使R_i位于位置m中。(b) 第二个恒等式从累计开头的$\binom{n}{m}$种情况得出，注意每个P_{nk}出现的次数，第三个恒等式从第二个通过反演得来〔或者，可以使用包含和排除原理〕。(c) $\sum_{m\geqslant 0}mP_{nm}=nQ_{nn}-Q_{n,n-1}$；因此

$$d_i=1+(N-1)-p_i\sum_{j\neq i}1/(p_i+p_j)$$

$$\sum p_id_i=N-\sum_{i<j}(p_i^2+p_j^2)/(p_i+p_j)=N-\sum_{i<j}(p_i+p_j-2p_ip_j/(p_i+p_j))=(17)$$

12. $\overline{C}_N=2^{1-N}+2\Sigma_{0\leqslant n\leqslant N-2}1/(2^n+1)$，它迅速地收敛到 $2\alpha'\approx 2.5290$；习题 5.2.4-13 给出 α' 到 40 位数字。

13. 在计算了相当麻烦的求和

$$\Sigma_{1\leqslant i\leqslant n}i^2H_{n+i}=\frac{1}{6}n(n+1)(2n+1)(2H_{2n}-H_n)-\frac{1}{36}n(10n^2+9n-1)$$

之后，我们得到答案

$$\overline{C}_N=\frac{4}{3}N-\frac{2}{3}(2N+1)(H_{2N}-H_N)+\frac{5}{6}-\frac{1}{3}(N+1)^{-1}\approx .409N$$

14. 我们可以假定 $x_1\leqslant x_2\leqslant\cdots\leqslant x_n$；则当 $y_{a1}\leqslant y_{a2}\leqslant\cdots\leqslant y_{an}$ 时，出现极大值，而当 $y_{a1}\geqslant\cdots\geqslant y_{an}$ 时出现极小值，这是根据类似于定理 S 中所作的论证得到的。

15. 如同在定理 S 中那样论证，排列 $R_1R_2\cdots R_N$ 是最优的充要条件是

$$P_1/L_1(1-P_1)\geqslant\cdots\geqslant P_N/L_N(1-P_N)$$

16. 当且仅当 $T_1/(1-p_1)\leqslant\cdots\leqslant T_N/(1-p_N)$ 时预期的时间 $T_1+p_1T_2+p_1p_2T_3+\cdots+p_1p_2\cdots p_{N-1}T_N$ 被极小化〔*BIT* 3 (1963)，255-256；詹姆斯·R. 斯莱格尔（James R. Slagle）得到了一些有趣的推广，*JACM* 11 (1964)，253-264〕。

17. 以递增的截止时间（它们分别同有关的时间 T_i 无关）的顺序做诸工作！〔当然，实际上某些作业比其它作业更重要，而且我们可能要使极大的加权延缓极小化。或者，我们可能希望使和数 $\Sigma_{1\leqslant i\leqslant n}\max(T_{a_1}+\cdots+T_{a_i}-D_{a_i},\ 0)$ 极小化。看来这些问题中没有一个是有一个简单的解的。〕

18. 如果 s 存在，则设 $h=1$，如果 s 不存在，则设 h 为 0。设 $A=\{i\mid q_i<r_i\}$，$B=\{i\mid q_i=r_i\}$，$C=\{i\mid q_i>r_i\}$，$D=\{j\mid t_j>0\}$；则对于 (q,r) 排列的和 $\Sigma_{1\leqslant i,j\leqslant N}p_ip_jd_{|i-j|}$ 减去对于 (q',r') 排列的对应的和等于

$$2\sum_{i\in A,\ j\in C}(q_i-r_i)(q_j-r_j)(d_{|i-j|}-d_{h+1+2k-i-j})$$
$$+2\sum_{i\in C,\ j\in D}(q_i-r_i)t_j(d_{h+2k-i+j}-d_{i-1+j})$$

这是正的，除非 $C=\phi$ 或 $A\cup D=\phi$。由于当 $m=0$，1 时风琴管的那些排法是仅有的，不可能被这个构造改进，也不可能被这个构造的左右对偶所改进，由此即得出所希望的结果。

〔这个结果基本上是 G. H. 哈迪（G. H. Hardy）、J. E. 利特尔伍德（J. E. Littlewood 以及 G. 波利亚（G. Po'lya）给出的（*proc. London Math. Soc.*(2), 25 (1926) 265-282)，他们证明，事实上，在诸 p 和诸 q 所有独立的重新排列中，当诸 p 和诸 q 都在一致的风琴管次序下时，$\Sigma p_iq_jd_{|i-j|}$ 达到极小值。关于进一步的评述和推广，见他们的书 "*INequalities*" (Combridqe University Press, 1934)，Chapter10.〕

19. 所有安排都是同样好的。假定 $d(i,\ i)=0$，我们有

$$\Sigma p_ip_jd(i,j)=\frac{1}{2}\Sigma p_ip_j(d(i,\ j)+d(j,\ i))=\frac{1}{2}c$$

〔对于 $i\neq j$ 的特殊情况 $d(i,\ j)=1+(j-i)\bmod N$ 是 K. E. 艾弗森在 "*A Programming Longuage*" (New York: Wiley, 1962)，138 中给出的，R. L. 巴伯（R. L. Baber）（见 *JACM* 10 (1963)，478-486）已经研究了一条带可以向前读，重绕或不读

而倒退 k 个块时，同带的查找有关的某些其它问题，W. D. 弗雷泽发现，如果允许我们复制文件中的某些信息，则有可能相当大量地减少查找时间；关于一个类似问题的经验解法参考E. B. 艾克尔伯格（E. B. Eichelberge）等，*IBM J. Research & Development* 12 (1968)，130-159。〕

20. 如同习题 18 中那样，以 $m=0$ 或 $m=h=1$ 从（q，r）进行到（q'，r'）给出了

$$\sum_{i\in A,\ j\in C}(q_i-r_i)(q_j-r_j)(d\,|\,i-j\,|-\min(d_{h+1+2k-i-j},\ d_{i+j-1}))$$

的纯变化，除 A 或 C 为 ϕ 外它都是正的。由循环对称性得出，仅有的最优安排是风琴管配置的循环移位〔对于具有相同答案的一类不同的问题，见 T. S. Motzkin 和 E. G. Straus, *Proc. Amer. Soc.* 7(1956)1014-1021〕。

21. 这个问题实际上是由 L. H. 哈珀（L. H. Harper）（见 *J. SIAM* 12(1964)，131-135）首先解决的，关于推广及其它工作的参考文献，见 *J. Applied Probability* 4 (1967)，397-401。

22. 大小为 1000 的一个优先队（比如说，表示为一个堆，见 5.2.3 节）。在这个队中记入前 1000个记录，并把具有最大 $d(K_i,\ K)$ 的元素放在前端。对于满足 $d(K_i,\ K)<d$（队的前端，K）的每一个后继 K_i 以 R_i 代替前端元素，并对队进行重新调整。

6.2.1 节

1. 归纳地证明，每当我们达到步骤 B2 时 $K_{l-1}<K<K_{u+1}$；而且每当达到 B3 时，$l\leqslant i\leqslant u$。

2. （a，c）否，如果 $l=u$ 和 $K>K_u$ 则它循环。（b）是，它是行的！但当不存在 K 时，将经常有对于 $l=u$ 和 $K<K_u$ 的循环。

3. 这是对于 $N=3$ 的算法 6.1T，在一次成功的查找中，该算法平均作 $(N+1)/2$ 次比较；在一次不成功的查找中，它作 $N/2+1-1/(N+1)$ 次。

4. 它必然是对于 $N=127$ 的一次不成功的查找；因此由定理 B，答案是 $138u$。

5. 程序 6.1Q′ 有 $1.75N+8.5-(N \bmod 2)/4N$ 的平均运行时间；这胜过程序 B 当且仅当 $N\leqslant 44$〔仅仅对于 $N\leqslant 11$ 它胜过程序 C〕。

7. （a）肯定不是。（b）算法 U 中带圆括弧的注释将始终成立，所以它是有效的，但当 N 为奇数时仅当 $K_0=-\infty$ 和 $K_{N+1}=+\infty$ 两者都出现时才是如此。

8. （a）N。通过归纳法证明这一点是有趣的，注意，如果我们以 $N+1$ 来代替 N，则清 DELTA 中恰有一个将增值〔关于一个推广，见 *AMM* 77 (1970)，884〕。（b）极大值 $=\Sigma$DELTA〔j〕$=N$；极小值 $=2$DELTA〔1〕$-\Sigma$DELTA〔j〕$=N \bmod 2$。

9. 考虑相应的树（例如图 6）：当 N 是奇数时，这个根的左子树是右子树的一个镜面映象，所以 $K<K_i$ 出现的次数同 $K>K_i$ 一样多；平均有，$C1=\frac{1}{2}(C+S)$，而且 $C2=\frac{1}{2}(C-S)$，$A=\frac{1}{2}(1-S)$。当 N 是偶数时，这株树相当于 $N+1$ 的一株树，其中所有标号减 1，除了◎变成冗余外，平均有：

$$C1=\frac{1}{2}(C+1)-\frac{1}{2}\lfloor\log_2 N\rfloor/N$$

$$C2=\frac{1}{2}(C-1)+\frac{1}{2}\lfloor\log_2 N\rfloor/N,\quad A=0\text{ 对于 }S=1$$

$$C1=\frac{1}{2}\frac{N}{N+1}(\lfloor\log_2 N\rfloor+1)$$

$$C2=\frac{1}{2}\frac{N+2}{N+1}(\lfloor\log_2 N\rfloor+1),\quad A=\frac{1}{2}\frac{N}{N+1}\text{ 对于 }S=0$$

（正文中指出了C的平均值。）

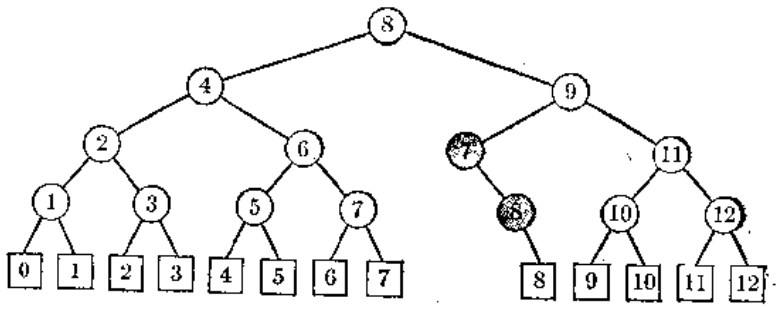
10. 当且仅当$N=2^k-1$。

11 使用“宏扩展”程序，其中包括诸 DELTA；例如，对于$N=10$：

```
START   ENT1    5
        LDA     K
        CMPA    KEY,1
        JL      C3A
C3A     JE      SUCCESS         C3A     EQU     *
        INC1    3                       DEC1    3
        CMPA    KEY,1                   CMPA    KEY,1
        JL      C3B                     JGE     C4B
C4B     JE      SUCCESS         C3B     EQU     *
        INC1    1                       DEC1    1
        CMPA    KEY,1                   CMPA    KEY,1
        JL      C3C                     JGE     C4C
C4C     JE      SUCCESS         C3C     EQU     *
        INC1    1                       DEC1    1
        CMPA    KEY,1                   CMPA    KEY,1
        JE      SUCCESS                 JE      SUCCESS
        JMP     FAILURE                 JMP     FAILURE
```

〔习题 23 表明大多数“JE”指令都可消去，同时产生大约长 $6\log_2 N$ 行的程序，它只花费大约 $4\log_2 N$ 个时间单位；但仅仅对于$N\geqslant 1000$（近似地）这个程序才是更快的。〕

12.



13

$N=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$C_N=$	1	$1\frac{1}{2}$	$1\frac{1}{3}$	$2\frac{1}{4}$	$2\frac{1}{5}$	$2\frac{2}{6}$	$2\frac{3}{7}$	$3\frac{1}{8}$	3	3	3	$3\frac{2}{12}$	$3\frac{3}{13}$	$3\frac{3}{14}$	$3\frac{4}{15}$	$4\frac{1}{16}$
$C'_N=$	1	$1\frac{2}{3}$	2	$2\frac{3}{5}$	$2\frac{4}{6}$	3	3	$3\frac{6}{9}$	$3\frac{6}{10}$	$3\frac{8}{11}$	$3\frac{8}{12}$	4	4	4	4	$4\frac{13}{17}$

14．一个思想是求使得$N+M$有形式$F_{k+1}-1$的最小的$M\geqslant 0$，然后在F1中从$i\leftarrow F_k-M$开始，而且在F2的开头处插入“如果$i\leqslant 0$，转向F4”。一个更好的解决方法是，对于斐波那契的情况采取沙尔的思想：如果第一次比较的结果是$K>K_{F_k}$，则置$i\leftarrow i-M$而且转到F4（从这以后正常进行）。这就避免了内循环中的额外时间。

15．在2.3.2节的“自然对应”之下，如果我们撤销线性图表最顶部的节点，则阶为k且左右颠倒的斐波那契树，是直到第k个月止对应于线性图表的二叉树。

16．外部节点出现于级$\lfloor k/2\rfloor$到$k-1$上；这些级之间的差别大于1，除非当$k=0$，1，2，3，4，时。

17．设路径长度是$k-A(n)$；则当$0<m<F_{j+1}$时，$A(F_j)=j$和$A(F_j+m)=1+A(m)$。

18．成功的查找：$A_k=0$，$C_k=(3kF_{k+1}+(k-4)F_k)/5(F_{k+1}-1)-1$，$C/k=C_{k-1}$。不成功的查找：$A'_k=F_k/F_{k+1}$，$C'_k=(3kF_{k+1}+(k-4)F_k)/5F_{k+1}$，$C1'_k=C'_{k-1}+F_{k-1}/F_{k+1}$。$C2=C-C1$（参考习题1.2.8-12）。

20．（a）$p^{-p}q^{-q}$。（b）至少有两个错误。第一个大错误是该因子不是一线性函数，所以它不能简单地“平均之”。实际上剩下pN个元素的概率为p，剩下qN个的概率为q，所以我们可预期剩下$(p^2+q^2)N$个；于是，平均的“减少因子”实际上是$1/(p^2+q^2)$。在k次迭代之后的减少因子是$1/(p^2-q^2)^k$，但我们不能作出结论说$b=1/(p^2+q^2)$，因为找出某些项所需要的迭代次数要比找出其它的项多得多，这是第二个失误。〔作出似是而非的概率论证是非常容易的，但我们必须防止这样的圈套！〕

21．它是不可能的，因为这个方法依赖于键的值。

23．当$\geqslant$时转向右边，当$<$时转向左边；当达到节点$\boxed{i}$时，由（1）得出$k_i\leqslant k<k_{i+1}$，所以关于相等性的最后一次测试将区别成功或失误（键$K_0=-\infty$将总是存在的）。

算法C将被改变成，如果在步骤C2中$K=K_i$，则转到C4。在C3中如果DELTA[j]$=0$，则置$i\leftarrow i-1$并转到C5，在C4中如果DELTA[j]$=0$，则直接地转到C5，增加一个新步骤C5：“如果$K=K_i$，则这算法成功地结束，否则它以失败告终。”〔除非N极其大，否则这不会加快程序C。〕

24．可以安排诸键使得我们首先置$i\leftarrow 1$，然后按$K<K_i$或$K>K_i$置$i\leftarrow 2i$或$2i+1$。当$i>N$时，这个查找是不成功的。例如当$N=12$时，键的安排必须是

$$K_8<K_4<K_9<K_2<K_{10}<K_5<K_{11}<K_1<K_{12}<K_6<K_3<K_7$$

对于在MIX上编的程序，这个方法大约将花费$6\log_2 N$个时间单位，所以它比程序C更快，唯一的缺点是首先建立表格时要有点技巧。

25．（a）由于$a_0=1-b_0$，$a_1=2a_0-b_1$，$a_2=2a_1-b_2$，等等，我们有$A(z)+B(z)=1+2zA(z)$。通过考虑$A(1)$，$B(1)$，$B(1/2)$，$A'(1)$，$B'(1)$，从这个关系立即得出在2.3.4.5节中导出的若干公式。如果我们使用两个变量来区别一条通路的向左走和向右走，则我们得到更一般的结果$A(x,\ y)+B(x,\ y)=1+(x+y)A(x,\ y)$，这是在$t$叉树中成立的一个公式的特殊情况〔参考R. M. Karp, IRE *Trans.* IT-7(1961), 27-38〕。（b）$\mathrm{Var}(g)=((N+1)/N)\mathrm{Var}(h)-((N+1)/N^2)\mathrm{mean}(h)^2+2$。

26．如果我们适当地排列左边和右边，则具有一个完全的k级分布的三条带多阶段合

并的合并树就是 $k+1$阶的斐波那契树（重画 5.4.4 节的多阶段$T=3$，$S=13$的图形，并颠倒A. C的左和右子树，即得图 8）。

27. 2^k 个结果中至多 $k+1$ 个会出现，因为我们可以把下标排序，使得$K_{i_1}<K_{i_2}<\cdots<K_{i_k}$。于是这个查找可以通过一株在每个节点处至多有（$k+1$）路分枝的树来描述，在第$m$步中可以找到的项目数至多是 $k(k+1)^{m-1}$；因此平均比较次数至少是 N^{-1}乘以多重集合$\{k\cdot 1, k(k+1)\cdot 2, k(k+1)^2\cdot 3, \cdots\}$的最小的$N$个元素之和，当$N\geqslant(k+1)^n-1$ 时，平均比较次数$\geqslant N^{-1}\Sigma_{1\leqslant m\leqslant n}k(k+1)^{m-1}m>n-1/k$。

6.2.2 节

1. 使用一个表头节点，并设 ROOT=RLINK(HEAD)；以 P←HEAD 在步骤 T 4 处开始这个算法，步骤 T 5 就如同当$K>$KEY(HEAD) 时那样执行〔于是改变程序 T 的行04和 05 为“ENT1 ROOT；CMPA K”〕。

2. 在步骤 T 5 中，置 RTAG(Q)←−，还有，当插入左边时，置 RLINK(Q)← P；当插入右边时，置 RLINK(Q)←RLINK(P)及 RTAG(P)←+。在步骤 T 4 中，把测试 RLINK(P)≠A 改变成为 RTAG(P)≠+〔如果诸节点被插入到递增的诸位置Q，而且如果所有的删去都是后进先出，则可删去 RTAG 场，因为当且仅当 RLINK(P)<P时，RTAG(P)为−。类似的注释对于同时的左和右穿线亦可应用〕。

3. 我们可以以一个正确的地址代替A，并在这个算法的开始处置 KEY(A)←K；然后可从内循环撤销对于 LLINK 或 RLINK=A 的判断。然而为了进行一个正确的插入，我们需要引进跟随 P 的另一个指针变量；这可以在不失去所述速度的优点下进行，并通过如程序 6.2.1 下中那样重复代码的方法来完成，于是 MIX 的时间将被减少成大约 5.5C。

4. 对于$N\geqslant 2^n-1$，$C_N=1+(0.1+1.2+\cdots+(n-1)2^{n-1}+C'_{2^n}+\cdots+C'_{N-1})/N=(1+1/N)C'_N-1$，对于$N\geqslant 2^n-1$，这些方程的解是 $C'_N=2(H_{N+1}-H_{2^n})+n$，它节省了 $2H_{2^n}-n-2\approx n(\ln 4+1)$次比较，对于 $n=1,2,3,4$实际的改进分别是 0，$\frac{1}{6}$，$\frac{61}{140}$，$\frac{274399}{360360}$；于是对于小的固定的 n 获利是相当小的〔关于一个等价的排序问题，有关的更详细的推导，见Frazer and Mckellar, *JACM* 17(1970), 502〕。

5. (a) 第一个元素必须是 CAPRICORN；然后我们把产生左子树的方式数乘以产生右子树的方式数再乘以$\binom{10}{3}$，即为把这两个序列搅混到一起的方式数，于是答案成为

$$\binom{10}{3}\binom{2}{0}\binom{1}{0}\binom{0}{0}\binom{6}{3}\binom{2}{0}\binom{1}{0}\binom{0}{0}\binom{2}{1}\binom{0}{0}\binom{0}{0}=4800$$

〔一般地说，这个答案是$\binom{l+r}{r}$对于所有节点的乘积，其中 l 和 r 代表这个节点的左和右子树的大小，这等于$N!$除以子树大小的乘积。它是和习题 5.1.4-20 中一样的公式；其实，如果我们以 k 代替查找树中的 a_k（用习题 6 的记号）〕，则在产生一株具体的查找树的诸排列和在该习题中计数的诸“拓扑”排列之间，有明显的一一对应。(b) $2^{N-1}=1024$；在除开最后一步之外的每一个步骤中，插入最小的或最大的剩余键。

6. (a) 对于其"得分"是 k 的 P_{nk} 个排列 $a_1\cdots a_{n-1}a_n$ 中的每一个，构造 $n+1$ 个排列 $a'_1\cdots a'_{n-1}ma'_n$。其中 $a'_j=a_j$ 或 a_j+1，根据 $a_j<m$ 或 $a_j\geqslant m$ 而定〔参考 1.2.5 节，方法 2〕。如果 $m=a_n$ 或 a_n+1，则这个排列"得分" $k+1$，否则它得分 k。(b) $G_n(z)=(2z+n-2)(2z+n-3)\cdots(2z)$，因此

$$P_{nk}=\left[\begin{matrix} n-1 \\ k \end{matrix}\right]2^k$$

实质上，这个母函数是由W. C. 林奇得到的 (*Comp. J.* 7(1965), 299-302)。(c) 诸概率的母函数是 $g_n(z)=G_n(z)/n!$。这是各概率的单个母函数的一个乘积，所以

$$\begin{aligned}\operatorname{var}(g_n)&=\sum_{0\leqslant k\leqslant n-2}\operatorname{var}((2z+k)/(2+k))\\&=\sum_{0\leqslant k\leqslant n-2}(2/(k+2)-4/(k+2)^2)=2H_n-4H_n^{(2)}+2\end{aligned}$$

〔由习题6.2.1-25(b)，我们可以利用 C'_n 的均值和方差计算 C_n 的方差，它是 $(2+10/n)H_n-4(1+1/n)(H_n^{(2)}/m+H_n^2)+2$；这个公式由G. D. 诺特给出（未发表）〕。

7. 建立类似于在习题 5.2.2-32 的答案中的递归式，但是 $C_{nm}=1+(A_{nm}+B_{nm})/n$。解是 $C_{nm}=H_m+H_{n+1-m}-1$〔*CACM* 12 (1969) 79-80；也见 L. Guibas *Acta Informatica* 4 (1975), 293-298〕。

8. (a) $g_n(z)=z^{n-1}\sum_{1\leqslant i\leqslant n}g_{i-1}(z)g_{n-i}(z)/n$，$g_0(z)=1$。(b) $7n^2-4(n+1)^2H_n^{(2)}-2(n+1)H_n+13n$〔P. F. Windley, *Comp. J.* 3(1960), 86 给出了一些递归关系，由这些递归关系，可以从数值上计算这个方差，但他未得到解。注意，这个解不是简单地相关于习题 6 的答案中所述的 C_n 的方差的〕。

10. 例如，键的每个字 x 都可为 $(ax)\bmod m$ 所代替，其中 m 是计算机字的大小而 a 是同 m 互质的一个随机乘数。可建议采用接近于 $(\phi-1)m$ 的一个值见 (6.4节)。树方法的灵活存储分配，使它们较之其它杂凑代码方案更有吸引力。

11. $N-2$，但是这仅仅是在删去①$N(N-1)\cdots2$ 时，以 $1/(N\cdot N!)$ 的概率出现。

12. 定理H的证明中 $\frac{1}{2}(n+1)(n+2)$ 个删去的属于情况 1，所以答案是 $(N+1)/2N$。

13. 是，事实上定理H的证明表明，如果我们对于任何固定的 i，删去第 i 个插入的元素，则结果是随机的 (G. D. Knott (Ph. D. thesis, Stanford, 1975) 已经证明，在随机插入一个任意序列，接着逐次删去插入的第 $(k_1, \cdots, k_d)$ 个元素（对于任何固定的序列 $k_1, \cdots, k_d$）之后，结果是随机的。)

14. 命NODE(T)在级 k 上，而且设LLINK(T)=Λ，RLINK(T)=R_1，LLINK(R_1)=R_2，…，LLINK(R_d)=Λ，其中$R_d\neq\Lambda$且 $d\geqslant1$。设对于 $1\leqslant i\leqslant d$，NODE(R_i)在它的右子树中有 n_i 个节点，通过步骤 $D1\frac{1}{2}$，内部路径长度减少 $k+d+n_1+\cdots+n_d$；如果没有这个步骤，则它减少 $k+d+n_d$。

15. 11, 13, 25, 11, 12。〔如果 a_j 是 $\{a_1, a_2, a_3\}$ 的（最小的，中间的，最大的），则在删去之后得到树↘(4, 2, 3)× 4 次。〕

16. 是；甚至在定理H的证明中所定义的对于排列的删去操作也是可交换的（如果我

们忽略重新编号这一点)。如果在X和Y之间有一个元素，则删去显然是可交换的，因为这个操作仅受X、Y相对位置，以及它们的后继者位置的影响；而且在X的删去和Y的删去之间没有交互作用。另一方面，如果Y是X的后继，而且Y是最大的元素，则删去的两种次序都有简单地撤销X和Y的作用，如果Y是X的后继者以及Z是Y的后继者，则两种删去的次序都有以Z代替X、Y或Z的第一次出现的作用，而且删去这些元素在这个排列内的第二次和第三次出现。

18. 使用习题1.2.7-14。

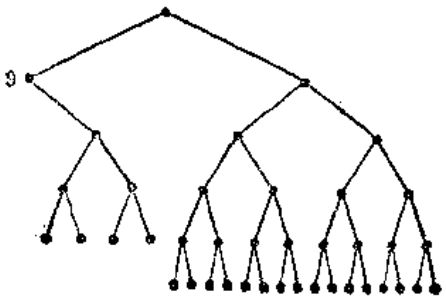
19. $2H_N-1-2\sum_{1\leqslant k\leqslant N}(k-1)^\theta/kN^\theta=2H_N-1-2/\theta+O(N^{-\theta})$。

20. 确实是。假定$K_1<\cdots<K_N$，使得由算法T构造的树是退化的；如果比如说$P_k=(1+((N+1)/2-K)\epsilon)/N$，则平均的比较次数是$(N+1)/2-(N^2-1)\epsilon/12$，而最优树只需少于$\lceil\log_2 N\rceil$次比较。

21. $\frac{1}{8}$，$\frac{3}{20}$，$\frac{9}{20}$，$\frac{3}{20}$，$\frac{1}{8}$(大部分角是30°、60°或90°)。

22. 当$d=2$时这是显然的，而对$d>2$我们有$r[i,j-1]\leqslant r[i+1,j-1]\leqslant r[i+1,j]$。

23.



〔增加第一个节点的权将使它最终移到根的位置；这提示，动态地维持一株完全最优的树是困难的。〕

24. 设c是通过删去一株最优树的第n个节点得到的一株树的代价，则$c(0,n-1)\leqslant c\leqslant c(0,n)-q_{n-1}$，这是因为删去操作总是把$\boxed{n-1}$上移一级。还有$c(0,n)\leqslant c(0,n-1)+q_{n-1}$，因为所述的替代产生具上述代价的一株树。由此得到$c(0,n-1)=c=c(0,n)=q_{n-1}$。

25. (a) 假设$A\leqslant B$和$B\leqslant C$，且$a\in A$，$b\in B$，$c\in C$，$c<a$。如果$c\leqslant b$，则$c\in B$；因此$c\in A$和$a\in B$；因此$a\in c$。如果$c>b$，则$a\in B$；因此$a\in C$和$c\in B$；因此$c\in A$。(b) 不难证明。

26. 对于某个实数$y\geqslant 0$和整数$l>0$，每株树的代价有$y+lx$的形式。有限个这样的函数的(对于所有树的)极小值总有所述的形式。

27. (a) 习题24的答案(特别是$c=c(0,n-1)$的事实)意味着$R(0,n-1)=R(0,n)\setminus\{n\}$。(b) 如果$l=l'$，则提示中的结果是显然的，否则设对于$\boxed{n}$的各通路是

$$\textcircled{r_1},\ \textcircled{r_2},\ \ldots,\ \boxed{r_l}\quad 和\quad \textcircled{s_1},\ \textcircled{s_2},\ \ldots,\ \boxed{s_{l'}}.$$

由于 $r=r_0>s_0=s$ 和 $r_{s'}<s_{l'}=n$，我们可以求出一个级 $k\geqslant 0$ 使得 $r_k>s_k$ 和 $r_{k+1}\leqslant s_{k+1}$。由归纳法我们有 $r_{k+1}\in R(r_k,n)$，$s_{k+1}\in R(s_k,n)$，以及 $R(s_k,n)\leqslant R(r_k,n)$，因此 $r_{k+1}\in R(s_k,n)$ 和 $s_{k+1}\in R(r_k,n)$；便得到提示中的结果。

现在证明 $R'_h\leqslant R_h$，设 $r\in R'_h$，$s\in R_h$，$s<r$，而且考虑当 $x=x_h$ 时出现的诸最优树；我们必须有 $l\geqslant l_n$ 且可以假定 $l'=l_h$。为了证明 $R_h\leqslant R'_{h+1}$，设 $r\in R_h$，$s\in R'_{h+1}$，$s<r$，而且考虑当 $x=x_{h+1}$ 时出现的诸最优树；我们必然有 $l'\leqslant l_h$ 因而可以假定 $l=l_h$。

29. 它是一株 YOU 在顶上，THE 在底下的退化树，平均需要 19.158 次比较。

道格拉斯·A. 哈密尔顿（Douglas A. Hamilton）已经证明，总有一株退化树是最坏的，因此存在求悲观的二分查找树的一个 $O(n^2)$ 算法。

30. 这已为拉塞尔·韦斯纳（Russell Wessnar）所证明（待出）。

31. （a）30；（b）31；于是有按照规则（1）构造成的树，它的代价比阶段 3 将成功的树要少。

32. 假设我们在一步中形成 q_i+q_j，而且在下一步有 $q_{i'}+q_{j'}<q_i+q_j$，则由于在 $q_{i'}$ 和 $q_{j'}$ 之间有外部节点 $\boxed{q_i}$ 或 $\boxed{q_j}$（或两者），组合 $q_{i'}+q_{j'}$ 必定已经在第一步被封锁了。如果有两个外部节点，则我们必定有 $q_{i'}>q_i$，$q_{j'}>q_j$；否则必定有 $q_{i'}>q_j$，$q_{j'}>q_i$。

35. 设 $i<j$；C_i 是 x_i 的前导 e_i+1 个二进位以及 C_j 是 $x_j\geqslant x_i+2^{-e_i-1}+2^{-e_j-1}$ 的前导 e_j+1 个二进位。因此如果 $e_i\leqslant e_j$，则 C_i 不可能是 C_j 的一个前缀；如果 $e_i>e_j$，则 C_j 不可能是 c_i 的一个前缀。

36. 见 *MAC Tech. Memo*, 69（M. I. T., November 1975），41pp。

38. 设 $a_r=\sum_{i\in S(r)}c_i$；则对于 $1\leqslant r<\lceil n/2\rceil$，$a_{r+1}-a_r\geqslant a_r-a_{r-1}$（参考习题 32）。因此对所有 r，$a_r\leqslant r/\lceil n/2\rceil$；由于关系式 $a_r>r/\lceil n/2\rceil$ 和 $a_{r-1}\leqslant(r-1)/\lceil n/2\rceil$ 意味着 $1\geqslant a_{\lceil n/2\rceil}=a_r+\sum_{r\leqslant k\leqslant\lceil n/2\rceil}(a_{k+1}-a_k)\geqslant a_r+(\lceil n/2\rceil-r)(a_r-a_{r-1})>r/\lceil n/2\rceil+(\lceil n/2\rceil-r)/\lceil n/2\rceil=1$，故得出这结果。使用关于给定的"最坏"概率的胡-塔克算法可完成这个证明：所得到的树在级 $q+1$ 上有 $n+1-2q$ 对外部节点，而且所有其它的外部节点都在级 q 上，所以 $f(n)=q(n+1-2q)/\lceil n/2\rceil+(q-1)(1-(n+1-2^q)/\lceil n/2\rceil)$。这至少同 T 的代价一样大，这个 T 在级 q 上的权为 $a_{n+1-2^q}\leqslant(n+1-2^q)/\lceil n/2\rceil$〔*Acta Informatica* 1(1972)，307-310。〕

6.2.3 节

1. 节点的对称次序必须通过变换加以保持，否则我们将失去一株二分查找树。

2. B(S)=0 仅当 S 指向树根时才能发生（在步骤 A3 或 A4 中它决不曾被改变），而且从 s 到插入点的所有节点是平衡的。

3. 设 ρ_h 是在高度为 h 的平衡树中，不平衡节点的最大可能比率，于是 $\rho_1=0$，$\rho_2=\frac{1}{2}$，$\rho_3=\frac{1}{2}$，我们将证明 $\rho_h=(F_{h+1}-1)/(F_{h+2}-1)$。设 T_h 是使 ρ_n 极大化的一株树；则我们可以假设它的左子树有高度 $h-1$ 且它的右子树有高度 $h-2$，因为如果两个子树的高度都是 $h-1$，则比率将小于 ρ_{h-1}，于是 T_h 的比率至多是 $(\rho_{h-1}N_l+\rho_{h-2}N_r+1)/(N_l+N_r+1)$，其中在（左、右）子树中有 (N_l,N_r) 个节点，当 (N_l,N_r) 取它们的极小值时，这个公式取它的极大值；因此我们可以假定 T_h 是一株斐波那契树。

4. 当 $n=7$ 时

有更大的路径长度。〔注：C. C. Foster, *Proc. ACM Nat, Conf.* 20(1965), 197-198, 给出了构造具有极大路径长度的N节点平衡树的一个不正确的过程；爱德华·洛格已经发现，福斯特的图 3 在 24 个步骤之后给出了一个非最优的结果（编号为 22 的点可被撤销以留下编号为 25 的节点。）〕

5. 这可以通过归纳法来证明；如果 T_N 表示被构造的树，则我们有

$$T_N=\begin{cases}\bigcirc\ (T_{2^{n-1}-1},\ T_{N-2^{n-1}}), & \text{如果}\quad 2^n\le N<2^n+2^{n-1};\\ \bigcirc\ (T_{2^n-1},\ T_{N-2^n}), & \text{如果}\quad 2^n+2^{n-1}\le N<2^{n+1}.\end{cases}$$

6. $zB_i(z)B_j(z)$ 中 z^n 的系数是 n 个节点的二叉树的个数，这株树的左子树是高度为 i 的一株平衡二叉树，而其右子树是高度为 j 的一株平衡二叉树。

7. $C_{n+1}=C_n^2+2B_{n-1}B_{n-2}$；因此如果我们设 $\alpha_0=\ln 2$，$\alpha_1=0$，以及$\alpha_{n+2}=\ln(1+2B_{n+1}\cdot B_n/C_{n+2}^2)=O(1/B_nC_{n+2})$，和 $\theta=\exp(\alpha_0/2+\alpha_1/4+\alpha_2/8+\cdots)$，则我们求得 $0<\theta^{2^n}-C_n=C_n(\exp(\alpha_n/2+\alpha_{n+1}/4+\cdots)-1)<1$，即 $C_n=\lfloor\theta^{2^n}\rfloor$。关于双重指数序列的一般结果，见 *Fibonacci Quarterly* 11(1973), 429-437。

8. 设 $b_n=B_h'(1)/B_h(1)+1$，而且设 ϵ_h 是非常小的量 $2B_hB_{h-1}(b_h-b_{h-1})/B_{h+1}$，则$b_h=2b_{h-1}-\epsilon_h$ 因此 $b_h=2^h(1-\epsilon_1/2-\epsilon_2/4-\cdots)+r_h$，其中 $r_h=\epsilon_{h+1}/2+\epsilon_{h+2}/4+\cdots$对于很大的 h 是极其小的〔*Zh. Vychisl. matem. i maten. fiziki* 6, 2 (1966), 384-394〕。

11. 当 $n\geqslant 2$ 时显然有和－－B 和＋－B 同样多的＋A，而且在＋和－之间有对称性。如果有M个＋A 和－A 类型的节点，当 $n\geqslant 1$ 时所有可能情况的考虑表明，下一个随机插入得到$M-1$ 个这样节点的概率为 $3M/(n+1)$，否则它恰好得到$M+1$ 个这样的节点。由此得出结果〔*SIAM J. Computing* 8 (1979), 33-41〕。

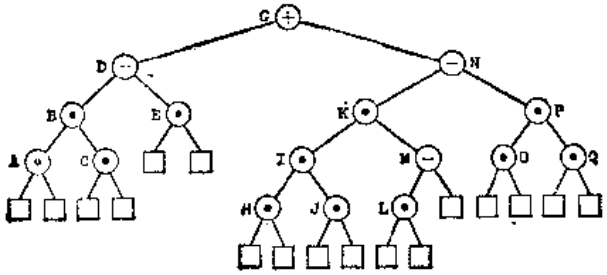
12. 当插入 (12) 的第二个外部节点时出现极大值；$C=4$，$C1=3$，$D=3$，$A=1$，$F=G=H=1$，$E=J=0$，加上行 67 中的 JMP，总计用 132 u 的时间。当插入 (13) 的第三个最后的外部节点时出现极小值；$C=2$，$C1=1$，$D=2$，$A=E=F=G=H=J=0$，总计用 61 u 的时间〔程序 6.2.2T 相应的数字是 74 u 和 26 u〕。

13. 当树改变时，仅仅需要更新$O(\log_2 N)$个 RANK 值，"简单的"系统可能需要非常广泛的变化。

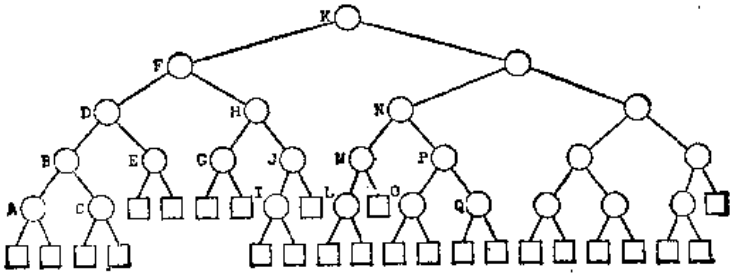
14. 是。（但是在表上的典型的操作是充分地非随机的，以致大概将出现退化树。）

15．使用算法 6.2.2T 且在步骤 T1 中置M为 0，而且每当在步骤 T2 中$K \geqslant$KEY(P)时，M←M+RANK(P)。

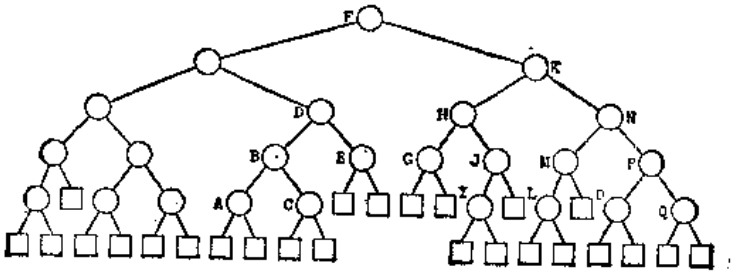
16．删去E；情况 3 在D处重新平衡。删去G；以G代F；情况 2 在H处重新平衡；在K处调整平衡因子。



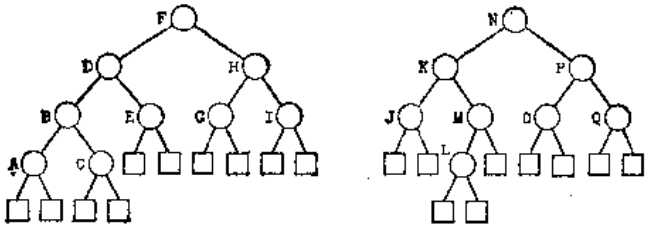
17．(c)



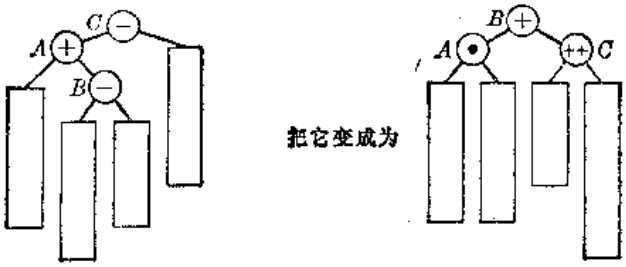
(b)



18．

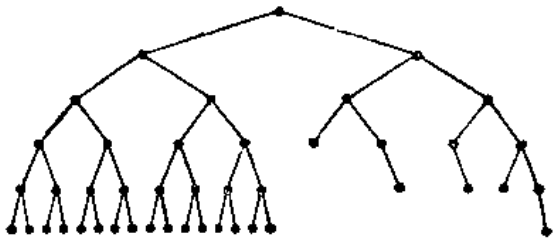


19．(由克拉克·克兰给出的解。）有一种情况，它不能由根处的单转动或双转动来处理，即是



而后通过应用在C处的单转动或双转动重新解决不平衡性。

20.



也许最困难的是在类似于这株的树的最左边插入一个新节点。D. S. 赫希伯格（D. S. Hirschberg）已经给出了一个至多花费$O(\log_2 n)^2$个步骤的插入算法，见*CACM* 19(1976)，471-473。

21. 算法A在阶为$N\log_2 N$的步骤中作此工作（见习题5）；下列算法使用一个递归方法的有趣的迭代方案，在$O(N)$步内建立同一些树，我们使用三个辅助表：

$D_1 \cdots D_l$ （实际上控制递归的一个二进计数器）

$J_1 \cdots J_l$ （指向“交接点”的指针表）

$T_1 \cdots T_l$ （指向树的一个指针表）

这里$l=\lceil \log_2(N+1)\rceil$，为了方便起见，这个算法也置$D_0 \leftarrow 1$，$J_0 \leftarrow J_{l+1} \leftarrow \Lambda$。

G1.〔初始化〕 置$l \leftarrow 0$，$J_0 \leftarrow J_1 \leftarrow \Lambda$，$D_0 \leftarrow 1$。

G2.〔得到下一项目〕 设P指向下一个输入节点（我们可以调用另一个联立子程序以便得到P），如果输入已尽，则转到G5。否则，置$k \leftarrow 1$，$Q \leftarrow \Lambda$，而且交换$P \leftarrow J_1$。

G3.〔进位〕 如果$k > l$（等价地，如果$P=\Lambda$），则置$l \leftarrow l+1$，$D_k \leftarrow 1$，$T_k \leftarrow Q$，$J_{k+1} \leftarrow \Lambda$，并返回G2，否则置$D_k \leftarrow 1-D_k$，交换$Q \leftrightarrow T_k$，$P \leftarrow J_{k+1}$，而且$k$增加1。如果现在$D_{k-1}=0$，则重复这一步骤。

G4.〔连接〕 置$\mathrm{LLINK}(P) \leftarrow T_k$，$\mathrm{RLINK}(P) \leftarrow Q$，$B(P) \leftarrow 0$，$T_k \leftarrow P$，并且返回G2。

G5.〔完成〕 对于$1 \leqslant k \leqslant l$，置$\mathrm{LLINK}(J_k) \leftarrow T_k$，$\mathrm{RLINK}(J_k) \leftarrow J_{k-1}$，$B(J_k) \leftarrow 1-D_{k-1}$，然后结束这个算法（$J_l$指向所求树的根）。

步骤G3被执行$2N-\nu(N)$次，其中$\nu(N)$是N的二进表示中1的个数。

22. 具有N个内部节点的一株加权平衡树的高度，总处于$\log_2(N+1)$和$2\log_2(N+1)$之间。为了达到这个上界，注意这个根的较大的子树至多有$(N+1)/\sqrt{2}$个外部节点。

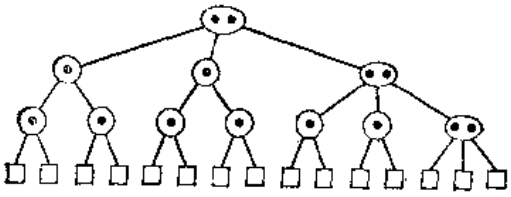
23.（a）构造一株树，其右子树是具有2^n-1个节点的完备的二叉树，其左子树是具有$F_{n+1}-1$个节点的一株斐波那契树。（b）其右子树大约是$2\log_2 N$级高，其左子树大约是$\log_2 N$级高，构造一株加权平衡树（参考习题22）。

24. 考虑满足这个条件但不是完全平衡的最小树。则，它的左和右子树是完全平衡的，所以它们分别有2^l和2^r个外部节点，其中$r \neq l$。但这同所述条件矛盾。

25. 在树的底部插入一个节点，我们从底往上地校验在查找通路上每个节点处权的平衡。假设在已经于右子树中插入一个新节点之后，在（1）中节点A处出现不平衡性，而B和它的子树是加权平衡的，则一次单转动将恢复平衡，除非$(|\alpha|+|\beta|)/|\gamma|>\sqrt{2}+1$，

其中 $|x|$ 表示在一个树 x 中的外部节点数。但在这种情况下，可以证明一个双转动已足够〔*SIAM J. Computing* 2(1973)，33-34〕。

27．有时需要在包含两个键的节点中作两次比较，在类似于下边那样的一株树中，出现最坏的情况，它有时需要进行 $2\log_2(N+2)-2$ 次比较。



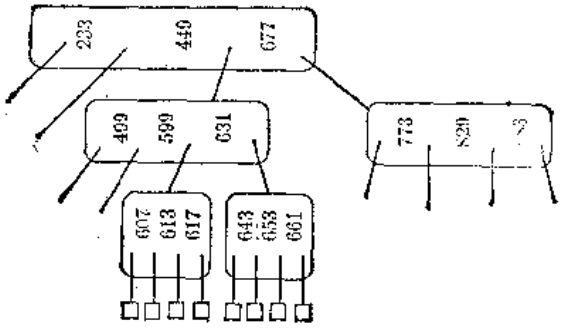
29．姚期智给出的部分解：对于 $N\geqslant 6$ 个键最低级将平均包含 $\frac{2}{7}(N+1)$ 个单键节点，$\frac{1}{7}(N+1)$ 个双键节点。对于很大的 N 平均的总节点数在 $0.70N$ 和 $0.79N$ 之间〔*Acta Informatica*，9 (1978)，171-181〕。

30．对于最好适合，按大小次序排列诸记录，以一个任意的规则来割断相等情况中的联系（参考习题2.5-9）。对于第一个适合，按位置次序安排诸记录，而且在每个节点中以一个额外的场指出在以该节点为根的子树中最大区域的大小。可以使插入和删去不影响这个额外的场（尽管运行时间是 $O(\log_2 n)$，但实际上大概仍然不能胜过习题 2.5-6 中的"ROVER"方法；但没有 ROVER 时存储分配可能更好些，因为在这种情况下通常遇紧急情况时可有一很大的区域供使用）。

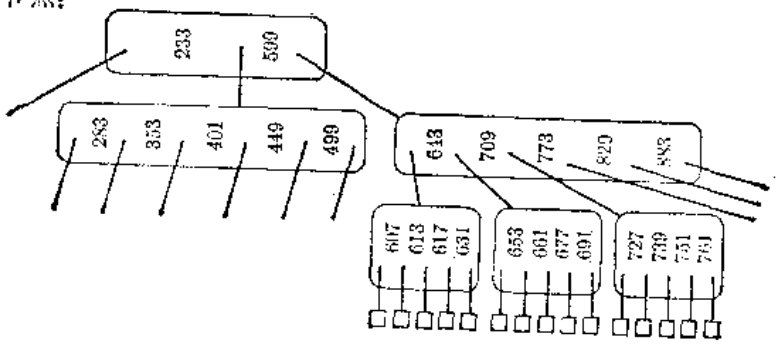
31．使用一个接近于平衡的树，并对于最左部分附加向上的链接，另有一个栈，栈中内容用于沿着这条通路作推迟的平衡因子校准（每个插入作有限次这样一些校准）。

6.2.4 节

1．双节点分开：



2．改变的节点：



(当然一株 B^* 树没有非根的 3 键节点，尽管图 30 有这种节点。)

3．(a) $1+2\cdot 50+2\cdot 51\cdot 50+2\cdot 51\cdot 51\cdot 50=2\cdot 51^3-1=265301$。(b) $1+2\cdot 50+(2\cdot 51\cdot 100-100)+((2\cdot 51\cdot 101-100)\cdot 100-100)=101^3=1030301$。(c) $1+2\cdot 66+(2\cdot 67\cdot 66+2)+(2\cdot 67\cdot 67\cdot 66+2\cdot 67)=601661$(小于(b)!)。

4．在分开一个非根的节点之前，先弄清楚：它确有两个已装满的兄弟，然后才把这三个节点分成为 4 个。若此节点是根，则仅当它有$3\lfloor(3m-3)/4\rfloor$个键以上时才能分开它。

5．解释1．试求所述极小的极大：450。如果我们有 1005 个字符，而且被传送给 FATHER（父亲）的键必须是 50 个字符长，则最坏的情况就出现了：445 字符＋指针＋50个字符的键＋指针＋50个字符的键＋指针＋445字符。

解释 2，试使分开之后键的个数相等，以便保持高的分枝因子：155（15个短的键后边跟有一些 16 个字符长的键）。

6．是，例如我们可以以 i 加上子树 P_0，$\cdots P_{i-1}$ 中的键来代替（1）中的每个K_i。查找、插入以及删去算法可适当地加以修改。

7．如果有待删去的键不在级 $l-1$ 上，则以它的后继者代替它，并删去这个后继者。为了删去在级 $l-1$ 上的一个键，我们简单地抹掉它；如果这使得节点太空，则就考察它的右（或左）兄弟，并且“下溢”，即，从这个兄弟移进一些键，使得这两个节点近似地有相同的数据量。这下溢的操作仅当该兄弟的装满程度为最低时才无效，但在该种情况下两个节点可以叠合成一个（回来自它们的父亲的一个键在一起）；这样一个叠合可能引起这个父亲进一步下溢，等等。

9．简述：推广分页方案，使得在每一个时刻仅有一个用户独占缓冲区的使用；必须小心地修改查找、插入以及删去算法，使得这样的独占使用仅当绝对需要时方在一段有限时间里被许可，并且还不致出现死锁。当一个具体的用户需要修改一个父亲的页时，他应查看从他最后一次观察该页以来，它是否被改变或移去了。

10．给定具有N个内部节点的一株树$\mathcal{T}$，设有需要 k 次存取的 $a_k^{(j)}$ 个外部节点，而且其父亲节点属于含有 j 个键的一个页；又设 $A^{(j)}(z)$是对应的母函数，于是 $A^{(1)}(1)+\cdots+A^{(M)}(1)=N+1$（对于 $1\leqslant j<M$，注意$a_k^{(j)}$是 $j+1$ 的一个倍数）。下一个随机插入产生$N+1$株同等概率的树，其母函数通过把某个系数 $a_k^{(j)}$减少 $j+1$ 而且把 $j+2$ 加到 $a_k^{(j+1)}$上而得到；或者（如果 $j=M$）通过使某个 $a_k^{(M)}$ 减 1，以及把 2 加到 $a_{k+1}^{(1)}$ 上而得到。现在 $B_N^{(j)}(z)$ 是 $A^{(j)}(z)$ 对所有$\mathcal{T}$求和，再乘上 $(N+1)^{-1}$，这里 $A^{(j)}(z)$是$\mathcal{T}$乘上$\mathcal{T}$出现的概率的母函数；由此可得出所述的诸递归关系。

递归式有形式

$$(B_N^{(1)}(z),\cdots,B_N^{(M)}(z)^T=(I+(N+1)^{-1}\mathbf{W}(z))(B_{N-1}^{(1)}(z),\cdots,B_{N-1}^{(M)}(z))^T$$
$$=\cdots=g_N(\mathbf{W}(z))(0,\cdots,0,1)^T$$

其中

$$g_n(x)=(1+x/(n+1))\cdots(1+x/2)=\binom{x+n+1}{n+1}/(x+1)$$

由此得出 $C_N'=(1,\ \cdots,\ 1)(B_N^{(1)\prime}(1),\ \cdots,\ B_N^{(M)\prime}(1))^T=2B_{N-1}^{(M)}(1)/(N+1)+C_{N-1}'=2f_N(\mathbf{W})_{MM}$，其中 $f_n(x)=g_{n-1}(x)/(n+1)+\cdots+g_1(x)/2=(g_n(x)-1)/x$，而且$\mathbf{W}=\mathbf{W}(1)$(下标$MM$表示矩阵的右下角元素)。现在$\mathbf{W}=\mathbf{S}^{-1}\mathrm{diag}(\lambda_1,\ \cdots,\ \lambda_M)\mathbf{S}$，其中$\mathbf{S}$是某个

矩阵，diag(λ_1，…，λ_n) 表示其元素是 $\chi(\lambda)=(\lambda+2)\cdots(\lambda+M+1)-(M+1)!$的根的对角矩阵（这些根是不同的，因为 $\chi(\lambda)=\chi'(\lambda)=0$ 意味着 $1/(\lambda+2)+\cdots+1/(\lambda+M+1)=0$；后者仅当 λ 是实数且 $-M-1<\lambda<-2$ 时才能成立，它意味着 $|\lambda+2|\cdots|\lambda+M+1|<(M+1)!$，矛盾)。如果 $p(x)$ 是任意多项式，则 $p(\mathbf{W})=p(\mathbf{S}^{-1}\mathrm{diag}(\lambda_1,\cdots,\lambda_M)\mathbf{S})=\mathbf{S}^{-1}\mathrm{diag}(p(\lambda_1),\cdots,p(\lambda_M))\mathbf{S}$；因此 $p(\mathbf{W})$的右下角元素有形式 $c_1p(\lambda_1)+\cdots+c_Mp(\lambda_M)$，其中 c_1，c_2，…，c_M 是某些同 p 无关的常数，这些常数可以通过置 $p(\lambda)=\chi(\lambda)/(\lambda-\lambda_i)$ 来求值；由于对于 $0\leqslant k\leqslant M-1$，$(\mathbf{W}^k)_{MM}=(-2)^k$，我们有 $p(\mathbf{W})_{MM}=p(-2)=(M+1)!/(\lambda_i+2)=c_ip(\lambda_i)=c_i\chi'(\lambda_i)=c_i(M+1)!(1/(\lambda_i+2)+\cdots+1/(\lambda_i+M+1))$；因此，$c_i=(\lambda_i+2)^{-1}((1/(\lambda_i+2)+\cdots+1/(\lambda_i+M+1)))^{-1}$。这得出了一个“显式”公式$C'_N=\sum_{1\leqslant i\leqslant M}2c_if_N(\lambda_i)$；剩下只是研究诸根 λ_i。注意对于所有 i，$|\lambda_i+M+1|\leqslant M+1$，否则我们有$|\lambda_i+2|\cdots|\lambda_i+M+1|>(M+1)!$矛盾。取 $\lambda_1=0$，这意味着对 $2\leqslant i\leqslant M$，$\Re(\lambda_i)<0$，由等式1.2.5－15，当 $n\to\infty$时 $g_n(x)\sim(n+1)^x/\Gamma(x+2)$；因此对于 $2\leqslant i\leqslant M$，$g_n(\lambda_i)\to 0$。结果$C'_N=2c_1f_N(0)+O(1)=H_N/(H_{M+1}-1)+O(1)$。

注意：上述分析也与在5.2.2节中简短地讨论的“抽样排序算法”有关。这些计算很容易推广以证明对于 $1\leqslant j<M$，$B_N^{(j)}(1)\sim(H_{M+1}-1)^{-1}/(j+2)$，且 $B_N^{(M)}(1)\sim(H_{M+1}-1)^{-1}/2$。因此在未充满的页上内部节点的总数近似地为$N(1/(3\times2)+2/(4\times3)+\cdots+(M-1)/((M+1)\times M))/(H_{M+1}-1)=N(1-M/(M+1)(H_{M+1}-1))$；而且所使用的页的总数近似为$N(1/(3\times2)+1/(4\times3)+\cdots+1/((M+1)\times M)+1/(M+1)/(H_{M+1}-1)\doteq N/2(H_{M+1}-1)$，存储使用渐近地为 $2(H_{M+1}-1)/M$。

6.3 节

1．叶（复数）。

2．利用新的键作为变元实施算法T；它将在步骤T3或T4中以失败告终。如果是在T3中，则只需置NODE(P)的表项 k 成为K并终止此插入算法，否则置这个表项成为一新节点Q←AVAIL的地址，该节点只包含空的链接，然后置P←Q。现在置 k 和 k' 分别成为K和X的下一个字符；如果 $k\neq k'$，则把K存入NODE(P)的位置 k 处，并把X存入位置 k' 中，但如果 $k=k'$，则再一次使 k 的位置指向一个新节点Q←AVAIL，置P←Q，并且重复这个过程直到最终 $k\neq k'$（我们必须假定没有一个键是另一个的前缀）。

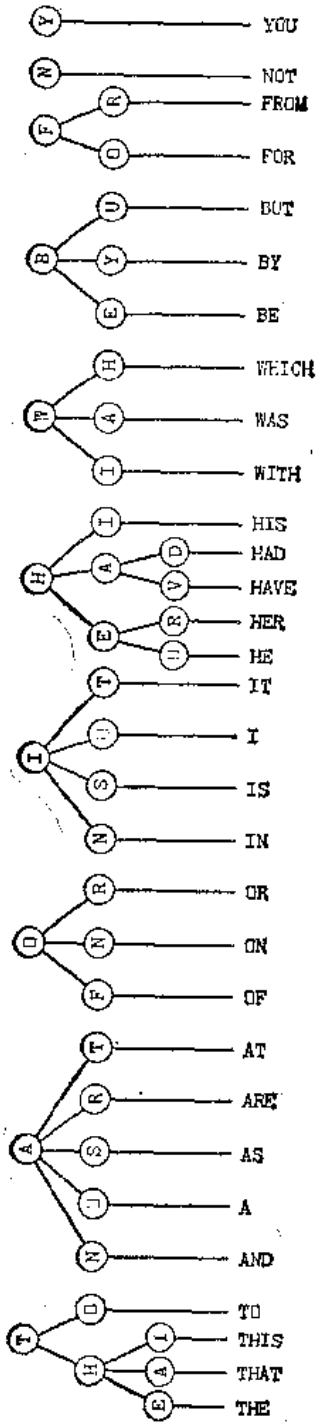
3．在键出现的节点处，以一个空的链接来代替此键。如果这个节点现在成了“无用的”，即，如果它除了有一个项是键X外，其它的所有项都是空的，则删去这个节点，并且以X代替它父亲中相应的指针。如果父亲节点现在已成为无用，则以同样的方式删去它。

4．在压缩了的表中，如果查找是成功的，则情况和完全的表是一样的，但如果查找是不成功的。则可能要多费若干次另外的迭代。例如，如象TRASH这样的输入变元将使程序T花费六次迭代（多于五次！）；这是最坏的情况。有必要验证，对空白的序列不可能有无穷的循环〔这有名的49位压缩法是由J．斯科特·菲什伯恩（J. Scot Fishburn）给出的，他证明了48个位置是不够的。一般来说，如果我们要来压缩分别含 x_1，…，x_n 个非零项的 n 个稀疏表，则用同以前放置的表不相冲突的极小数量 r_j 补偿第 j 个表的“最先适配”方法将有 $r_j\leqslant(x_1+\cdots+x_{j-1})x_j$，因为每个以前的非零项顶多能封锁 x_j 个补偿。对

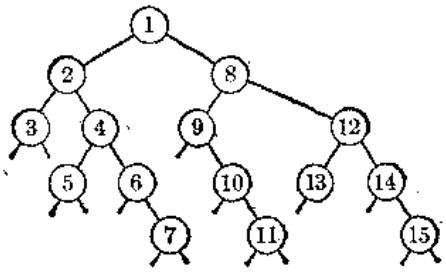
于表 1 中的数据这最坏情况估计给出 $r_i \leqslant 93$，并保证分别含有10，5，4，3，3，3，3，3，2，2，2，2 个非零项、长度为 30 的任何12个表可以压缩成 93＋30个连续的单元，而不论非零项的模式如何。R. E. 塔简和姚期智已经给出对于这个方法进一步的改进，*CACM* 22(1979)，606-611。〕

库尔特·马里（Kurt Maly）已经提出了可节省检索结构存储的一个更有用的方法，见*CACM* 19(1976)，409-415。

5．在每个族中，通过从左到右地以概率递减的次序排列字母，首先测试最可能的结果。这排列的最优性可以如同在定理 6.1 S 中那样证明（参考 *CACM* 12(1969)，72-76)。



6.



7. 例如，8，4，12，1，2，3，5，6，7，9，10，11，13，14，15（不管使用什么序列，左子树在级4上不能包含多于两个节点，右子树也不能）。甚至这株“最坏”的树也是在最好的四株树之内，所以我们看到，数字查找树对于插入次序不是非常敏感的。

8. 是。KEY场现在仅仅包含一个截断了的键；表示节点位置的诸前导二进位被砍掉了（算法T的一项类似的修改是可能的）。

9.

START	LDX	K	1	D1. 初始化 (rX←K)
	LD1	ROOT	1	P←ROOT(rI1←P)
	JMP	2F	1	
4H	LD2	0,1(RLINK)	C2	D4. 右移，Q←RLINK(P)
	J2Z	5F	C2	如果Q＝Λ，则移到D5
1H	ENT1	0,2	C－1	P←Q
2H	CMPX	1,1	C	D2. 比较
	JE	SUCCESS	C	如果K＝KEY(P)，则转出
	SLB	1	C－S	把K左移一个二进位
	JAO	4B	C－S	如果移掉的二进位是1，则转到D4
	LD2	0,1(LLINK)	C1	D3. 左移，Q←LLINK(P)
	J2NZ	1B	C1	如果Q≠Λ，则以P←Q转到D2
5H	如同在程序6.2.2T中那样继续，并交换rA、rX的作用。			

这个程序查找阶段的运行时间是 $(10C-3S+4)u$，其中 $C-S$ 是二进探查的数目。因此，对于随机数据，近似的平均运行时间是

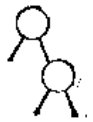
	成功的	不成功的
程序6.2.2T	$15\ln N-12.34$	$15\ln N-2.34$
这个程序	$14.41\ln N-6.17$	$14.41\ln N+1.26$

（因此，除非N非常大；程序6.2.2T要稍许快些。）

10. 设$\oplus$表示对于n个二进位数的“异或”操作，而且设$f(x)=n-\lceil\log_2(x+1)\rceil$，即$x$的前导0位的个数。一种解法：（b）如果算法T执行的一个查找在步骤T3处以失败告终，则k比至今所作的二进位探查数小1；否则如果查找在步骤T4处结束，则$k=f(K\oplus X)$。（a，c）做正规的查找，但是也要随时记住$K\oplus$KEY(P)的极小值x，其中的KEY(P)遍历在查找过程中与K作过比较的所有KEY(P)。然后$k=f(x)$)。（证明，没有其它的键能比与K比较过的键与K有更多的共同二进位。在情况（a）中，极大的k值或者对于$\leqslant K$的最大键，或者对于$>k$的最小键出现。）

11．否；消去只有一个空子树的一个节点会“忘掉”在非空子树键中的一个二进位。为删去一个节点，应该以它的终端后代之一代替它，比如说，只要可能时通过向右查找来实现。

12．把在0和1之间的三个随机数α、β、γ插入到开始时为空的一株树中；然后使用前面答案中所建议的算法，以概率p删去α，以概率q删去β，以概率r删去γ，则我们以概率$\frac{1}{4}p+\frac{1}{2}q+\frac{1}{2}r$得到树



而且仅当$p=0$时，这概率是$\frac{1}{2}$。

13．对每个节点加一个KEY场，而且在考察步骤T2的向量元素之前把K同这个键加以比较。表1将改变如下：节点（1），…，（12）将分别包含键THE，AND，BE，FOR，HIS，IN，OF，TO，WITH，HAVE，HE，THAT（如果我们按频率递减的次序插入它们），而且这些键将从他们以前的位置中被删去〔对应的程序在这种情况下将比程序T更慢和更复杂，算法D的一个更直接的M进推广将建立具有N个节点的一株树，每个节点有一个键和M个链接〕。

14．如果$j\leqslant n$，则仅有一个位置，即KEY(P)。但是如果$j>n$，则通过遍历节点P的子树找出所有出现的集合：如果有r个出现，则这株子树包含$r-1$个节点（包括节点P在内），而且因此它有r个链接场的TAG=1；这些链接场指向访问TEXT（即匹配K的位置的）所有节点（完全不必再检验TEXT了）。

15．为开始形成这株树，置KEY(HEAD)成为第一个TEXT访问，而且LLINK(HEAD)←HEAD，LTAG(HEAD)←1，用下列插入算法把其余的TEXT访问记入树中。

置K成为我们希望记入的新键（这是插入算法所作的对于TEXT阵列的第一次访问）实施算法P；它必然以失败告终，因为不允许一个键是另外键的一个前缀（步骤P6作对于TEXT的第二次访问；不再需要更多的访问了！）；现在假设在步骤P6中找到的键同变元K在前l个二进位中一致，但是在第$l+1$个位置却不同，在K中是数字b而在键中是$1-b$（尽管算法P中的查找可能要使j比l大得多，但可以证明，这里所确定的过程将在K和现存的任何键之间求得最大的匹配。于是，以K的前l位开始的正文的所有键，都有$1-b$作为它的第$l+1$位二进位）。现在重复算法P并且以这些l个前导二进位代替K（即$n\leftarrow l$）。这次查找将是成功的，所以我们不必实施步骤P6，现在置R⇐AVAIL，KEY(R)←TEXT中新键的位置。如果LLINK(Q)=P，则置LLINK(Q)←R，$t\leftarrow$LTAG(Q)，LTAG(Q)←0；否则置RLINK(Q)←R，$t\leftarrow$RTAG(Q)，RTAG(Q)←0。如果$b=0$，置LTAG(R)←1，LLINK(R)←R，RTAG(R)←t，RLINK(R)←P；否则置RTAG(R)←1，RLINK(R)←R，LTAG(R)←t，LLINK(R)←P。如果$t=1$，则置SKIP(R)←$1+l-j$；否则置SKIP(R)←$1+l-j+$SKIP(P)和SKIP(P)←$j-l-1$。

16．树的建立恰恰需要从下边的一个节点到该节点的一条虚线的链接；它来自树的一个部分，在这部分中此键第一次不同于所有其它键。如果树中没有这样的部分，则这些算法失败。我们可以简单地去掉作为其它键的前缀的键，但那样的话习题14的算法将没有足够的数据来找出这个变元的所有出现。

17．如果我们定义 $a_0=a_1=0$，则

$$x_n=a_n+\sum_{k\geqslant 2}\binom{N}{k}(-1)^k\hat{a}_k/(m^{k-1}-1)=\sum_{k\geqslant 2}\binom{N}{k}(-1)^k\hat{a}_k m^{k-1}/(m^{k-1}-1)$$

18．为了解（4），我们需要 $a_n=1-\delta_{n0}-\delta_{n1}$ 的变换，即 $\hat{a}_n=\delta_{n0}-1+n$；因此对于 $N\geqslant 2$，我们得到 $A_N=1-U_N+V_N$，其中 $U_N=K(N,\ O,\ M)$ 和 $V_N=K(1,\ N,\ M)$，这里用的是习题19的记号。类似地，为了解（5），取 $a_n=n-\delta_{n1}=\hat{a}_n$，而且对于 $N\geqslant 2$，得到 $C_N=N+V_N$。

19．对于 $s=1$，我们有 $V_n=K(n,1,m)=n(\ln n+\gamma)/(\ln m)-\frac{1}{2}-f_0(n-1)+O(1)$，对于 $s\geqslant 2$，我们有

$$K(n,\ s,\ m)=(-1)^s n(1/(\ln m)+f_{s-1}(n-s))/s(s-1)+O(1)$$

其中

$$f_s(n)=\frac{2}{\ln m}\sum_{k\geqslant 1}\Re(\Gamma(s-2\pi ik/(\ln m))\exp(2\pi ik\log_m n))$$

是 $\log_2 n$ 的一个周期函数〔在这个推导中我们有

$$K(n+s,\ s,\ m)/(-1)^s\binom{n+s}{s}=$$

$$(n^{-s+1}/2\pi i)\int_{1/2-i\infty}^{1/2+i\infty}\Gamma(z)n^{s-1-z}dz/(m^{s-1-z}-1)+O(n^{-s})$$

对于小的 m 和 s，诸 f 小得可以忽略；参考习题5.2.2-46。注意，对于固定的 a，$f_s(n-a)=f_s(n)+O(n^{-1})$。

20．对于（a）令 $a_n=1-\delta_{n0}-\delta_{n1}\cdots\delta_{ns}$；对于（b）令 $a_n=n-\delta_{n1}-2\delta_{n2},\ \cdots s\delta_{ns}$；对于（c）我们要解递归式

$$y_n=m^{1-n}\sum_k\binom{n}{k}(m-1)^{n-k}y_k\qquad 对于\ n>s$$

$$y_n=\binom{n+1}{2}\qquad 对于\ n\leqslant s$$

置 $x_n=y_n-n$ 得出习题17形式的递归式，且

$$a_n=(1-M^{-1})\sum_{0\leqslant k\leqslant s}\binom{k}{2}\delta_{nk}$$

因此，在前边习题的记号下，答案是（a）$1-K(N,\ O,\ M)+K(N,\ 1,\ M)-\cdots+(-1)^{s-1}K(N,\ s,M)=N/(s\ln M)-N(f_{-1}(N)+f_0(N-1)+f_1(N-2)/2\cdot 1+\cdots+f_{s-1}(N-s)/s(s-1))-O(1)$；（b）$N^{-1}(N+K(N,\ 1,\ M)-2K(N,\ 2,\ M)+\cdots$

$+(-1)^{s-1}sK(N,\ s,\ M))=(\ln N+\gamma-H_{s-1})/(\ln M)+\frac{1}{2}-(f_0(N-1)+f_1(N-2)/1+\cdots+f_{s-1}(N-s)/(s-1))+O(N^{-1})$；(c) $N^{-1}(N+(1-M^{-1})\sum_{2\leqslant k\leqslant s}(-1)^k\binom{k}{2}K(N,\ k,\ M))=1+\frac{1}{2}(1-M^{-1})((s-1)/(\ln M)+f_1(N-2)+\cdots+f_{s-1}(N-s))+O(N^{-1})$。

21．设共有A_N个节点，非空指针的个数是A_N-1，而且非指针的个数是N，因此空指针的总数是MA_N-A_N+1-N。除以m，即得它们在任何固定位置的平均值〔A_N的平均值出现于习题20(a)中〕。

22．对于M^l个前导二进位序列中的每一个，只要至少两个键有这个序列的二进位型式，即有一个节点与此相应。由于恰恰k个键有一个特定的二进位型式的概率是

$$\binom{N}{k}M^{-lk}(1-M^{-l})^{N-k}$$

所以在k级上检索结构的平均节点数是$M^l(1-(1-M^{-l})^N)-N(1-M^{-l})^{N-1}$。

23．更一般地说，考虑如同习题20中那样任意s的情况，如果有a_l个节点在级l上，则它们包含a_{l+1}个链接以及Ma_l-a_{l+1}个位置，在这些位置上查找可能是不成功的。数字探查的平均数因此将是$\sum_{l\geqslant 0}(l+1)M^{-l-1}(Ma_l-a_{l+1})=\sum_{l\geqslant 0}M^{-l}a_l$。把$a_l$的这一公式使用于一随机检索结构中，它等于$1+(N+1)^{-1}(K(N+1,\ 1,\ M)-2K(N+1,\ 2,\ M)+\cdots+(-1)^s(s+1)K(N+1,\ s+1,\ M))=(\ln N+\gamma+H_s)/(\ln M)+\frac{1}{2}-f_0(N)-f_1(N-1)/1-\cdots-f_s(N-s)/s+O(N^{-1})$。

24．我们必须解递归式$x_0=x_1=y_0=y_1=0$

$$x_n=m^{-n}\sum_{n_1+\cdots+n_m=n}\binom{n}{n_1,\ \cdots,\ n_m}\left((x_{n_1}+\cdots+x_{n_m}+\sum_{1\leqslant j\leqslant m}(1-\delta_{n_j0})\right)$$
$$=a_n+m^{1-n}\sum_k\binom{n}{k}x_k$$

$$y_n=m^{-n}\sum_{n_1+\cdots+n_m=n}\binom{n}{n_1,\ \cdots,\ n_m}\left((y_{n_1}+\cdots+y_{n_m}+\sum_{1\leqslant i\leqslant j\leqslant m}(1-\delta_{n_j0})n_i\right)$$
$$=b_n+m^{1-n}\sum_K\binom{n}{k}y_k$$

其中$n\geqslant 2$，$a_n=m(1-(1-1/m)^n)$和$b_n=\frac{1}{2}(m-1)n(1-(1-1/m)^{n-1})$。由习题17、18，答案是(a) $x_N=N+V_N-U_N-\delta_{N1}=A_N+N-1$〔一个本来也可直接得到的结果，因为森林中的节点数总是比在对应的检索结构中的节点数大$N-1$!〕；且(b) $y_N/N=\frac{1}{2}(M-1)V_N/N=\frac{1}{2}(M-1)((\ln N+\gamma)/(\ln M)-\frac{1}{2}-f_0(N-1))+O(N^{-1})$。

25．(a) 设$A_N=M(N-1)/(M-1)-E_N$；则对于$N\geqslant 2$，我们有$(1-M^{1-N})E_N=$

$M-1-M(1-1/M)^{N-1}+M^{1-N}\sum_{0<k<N}\binom{N}{k}(M-1)^{N-k}E_k$。由于$M-1-M(1-1/M)^{N-1}\geqslant 0$，由归纳法，我们有$E_N\geqslant 0$。(b) 由定理1.2.7A连同$x=1/(M-1)$，$n=N-1$，我们求得$D_N=a_N+M^{1-N}\sum_k\binom{N}{k}(M-1)^{N-k}D_k$，其中$a_1=0$，且对于$N\geqslant 2$，$0<a_N<M(1-1/M)^N/\ln M\leqslant(M-1)^2/M\ln M$。因此，$0\leqslant D_N\leqslant(M-1)^2A_N/M\ln M\leqslant(M-1)(N-1)/\ln M$。

26. 在习题5.1.1-16的第二个恒等式中取$q=-\frac{1}{2}$，$z=-\frac{1}{2}$，我们得到$1/3-1/(3\cdot 7)+1/(3\cdot 7\cdot 15)-\cdots=0.28879$；若使用$z=-\frac{1}{4}$和取这个结果的一半则要稍快些。或者，可以使用来自习题5.1.1-14的欧拉公式，它仅涉及2的负的次幂（约翰·伦奇已经计算了40D的值0.2887880950 86602 42127 88997 21929 23078 00889+）。

27. (为了开心，下列推导进行到$O(N^{-1})$。) 在习题5.2.2-38，48的记号下，我们有$\overline{C}_N=U_N+N-1+V_{N+1}/(N+1)-\alpha N-\beta+\sum_{n\geqslant 2}(-1)^n2^{-n(n+1)/2}(\prod_{1\leqslant r\leqslant n}(1-2^{-r})^{-1})\times(\sum_{m\geqslant 0}(2^{1-n})^m(1-2^{-m})^N)$，其中$\alpha=2/(1\cdot 1)-4/(3\cdot 3\cdot 1)+8/(7\cdot 7\cdot 3\cdot 1)-\cdots\approx 1.60669\ 51524\ 15291\ 76378\ 33015\ 23190\ 92458\ 04806$—和$\beta=2/(1\cdot 3\cdot 1)-4/(3\cdot 7\cdot 3\cdot 1)+8/(7\cdot 15\cdot 7\cdot 3\cdot 1)-\cdots\approx 0.60670$。这个数值计算提示了$\alpha=\beta+1$，这是一个不难证明的事实，由习题5.2.2-46，$\sum_{m\geqslant 0}(2^{1-n})^m(1-2^{-m})^N$的值是$O(N^{1-n})$；而且$V_{N+1}/(N+1)=U_{N+1}-U_N$。因此由习题5.2.2-50，$\overline{C}_N=U_{N+1}-(\alpha-1)N-\alpha+O(N^{-1})=(N+1)\log_2(N+1)+N((\gamma-1)/\ln 2+\frac{1}{2}-\alpha+f_{-1}(N))+\frac{1}{2}-1/\ln 4-\alpha-\frac{1}{2}f_1(N)+O(N^{-1})$。

28. 如果我们在显然的位置中以M替换2，则正文中和习题27中的推导可应用于一般的$M\geqslant 2$。因此在一次随机的成功查找中平均的数字探查次数是$\overline{C}_N/N=U_{N-1}-\alpha_M+1+O(N^{-1})=\log_M N+(\gamma-1)/\ln M+\frac{1}{2}-\alpha_M+f_{-1}(N)+(\log_M N)/N+O(N^{-1})$；而且对于不成功的情况它是$\overline{C}_{N+1}-\overline{C}_N=V_{N+2}/(N+2)-\alpha_M+1+O(N^{-1})=\log_M N+\gamma/\ln M+\frac{1}{2}-\alpha_M-f_0(N+1)+O(N^{-1})$。这里$f_s(m)$在题19中定义，而$\alpha_M=\sum_{j\geqslant 0}(-1)^jM^{j+1}/(M^{j+1}-1)^2(M^j-1)\cdots(M-1)$。

30. 通过迭代递归式，$h_n(z)$是形如

$$\binom{n}{p_1}(z/(2^{p_1}-1))\binom{p_1}{p_2}(z/(2^{p_2}-1))\cdots(z/(2^{p_m}-1))\binom{p_m}{1},\quad 对于\ n>p_1>\cdots>p_m>1$$

的所有可能的项之和。

31. $h_n'(1)=V_n$（参考习题5.2.2-36b）。

32. SKIP场之和是在对应的二进检索结构中的节点数，所以答案是A_N（参考习题20）。

33. (18) 是这样发现的：$A(2z)-2A(z)=e^{2z}-2e^z+1+A(z)(e^z-1)$可以被变换成为$A(2z)/(e^{2z}-1)=(e^z-1)/(e^z+1)+A(z)/(e^z-1)$。因此$A(z)=(e^z-1)\sum_{j\geqslant 1}\times(e^{z/2^j}-1)/(e^{z/2^j}+1)$。现在如果$f(z)=\sum c_nz^n$，则$\sum_{j\geqslant 1}f(z/2^j)=\sum c_nz^n/(2^n-1)$。在这

种情况下 $f(z)=(e^z-1)/(e^z+1)=\tanh(z/2)=1-2z^{-1}\times(z/(e^z-1)-2z/(e^{2z}-1))=\sum_{n\geq 1}B_{n+1}z^n(2^{n+1}-1)/(n+1)!$。从这个公式出发，下一步该怎么走就清楚了。

34. (a) 考虑 $\sum_{j\geq 1}\sum_{2\leq k<n}\binom{n}{k}B_k/2^{j(k-1)}$；由习题 1.2.11.2-4, $1^{n-1}+\cdots+(m-1)^{n-1}=(B_n(m)-B_n)/n$，(b) 在 $1/(e^x-1)$ 部分，考虑满足 $x\leq 2\ln n$ 的值就足够了，对于 $1\leq x\leq 2\ln n$，我们有 $\sum_{1<k<n/x}(1-kx/n)^{n-1}=\sum_{k\geq 1}e^{-kx}+O(x^2e^{-x}/n)$。对于 $x\leq 1$，我们有 $\sum_{1\leq k<n}\binom{n}{k}B_k(x/n)^k=\sum_{k\geq 0}B_kx^k/k!+O(x^2/n)$。(c) 当 $|x|<2\pi$ 时，如同在 5.2.2 节中那样来论证，然后使用解析展拓。(d) $-\frac{1}{2}\log_2(n/\pi)+\gamma/(2\ln 2)-\frac{3}{4}+f(n)+2/n$，其中 $f(n)=(2/\ln 2)\sum_{k\geq 1}\Re(\zeta(-2\pi ik/\ln 2)\Gamma(-2\pi ik/\ln 2)\exp(2\pi ik\log_2 n))=(1/\ln 2)\sum_{k\geq 1}\Re(\zeta(1+2\pi ik/\ln 2)\exp(2\pi ik\log_2(n/\pi))/\cos k(\pi^2k/\ln 2)$。

35. 诸键必须是 $\{\alpha 0\beta 0\omega_1,\ \alpha 0\beta 1\omega_2,\ \alpha 1\gamma 0\omega_3,\ \alpha 1\gamma 1\delta 0\omega_4,\ \alpha/\gamma 1\delta 1\omega_5\}$，其中 α，$\beta\cdots$ 是 0 和 1 的串且有 $|\alpha|=a-1$，$|\beta|=b-1$、等等。五个随机键有这个形式的概率是 $5!\,2^{a-1+b-1+c-1+d-1}/2^{a+b-a+b+a-c+a+c+d+a+c+d}=5!/2^{2a-b+2c+d+4}$。

36. 设 n 是内部节点的个数。(a) $(n!/2^I)\pi(1/s(x))=n!\pi(1/2^{I(x)-1}s(x)$，其中 I 是此树的内部路径长度。

(b) $((n+1)!/2^n)\pi(1/(2^{I(x)}-1))$(考虑对于所有的 a，b，c，$d\geq 1$ 来对习题 35的答案求和)。

37. 最小的修改了的外部路径长度实际上是 $2-1/2^{N-2}$，而且它出现在一株退化的树(其外部路径长度为极大者)中〔人们可以证明，最大的修改了的外部路径长度出现的条件是当且仅当外部节点出现在至多两个相邻的级上！但是如果说凡是其外部路径长度小于另一株树的外部路径长度者，一定有一个较大的修改了的外部路径长度，这一点就不见得总是正确的了〕。

38. 把求具有参数 $(\alpha,\ \beta)$，$\left(\alpha,\ \frac{1}{2}+\beta\right)$，…，$(\alpha,\ 2^{k-n}\beta)$ 的那些 k 节点的树看一些子问题。

39. 见宫川 (Miyakawa)，弓场 (Yuba)，杉藤 (Sugito) 及星守 (Hoshi Mamoru)，*SIAM J. Computing* 6(1977)，201-234。

40. 设 N/r 是这个序列的真的周期长度。构造一株Patricia类型的树，以 $a_0a_1\cdots$ 作为 TEXT，又以 N/r 个键从位置 0，1，…，$N/r-1$ 处开始(没有任何键是另一个的前缀，这是由于我们对 r 的选择所致)。在每个节点中还包括一个SIZE 场，该场包含在此节点下面的子树中带标志的链接场的数目。为做此操作，使用算法 P；如果这次查找是不成功的，则答案为 0，但如果它是成功的，而且 $j\leq n$，则答案为 r。最后，如果它是成功的，而且 $j>n$，则答案为 $r\cdot$SIZE(P)。

6.4 节

1. $-37\leq r11\leq 46$。因此在TABLE之前和之后的单元必须保证不包含匹配任何给定的变元的数据。例如，它们的第一个字节可以为 0，在这个范围内存储 K 肯定是坏的！〔在某种意义下，我们可以说，习题 6.3-4 中的方法使用较少的空间，因为该表的边界决不被超过。〕

2．TOW。〔读者能否找到≤5个字母的十个“常用的”字，填满－10和30之间所有剩下的间隔？〕

3．字符代码A＋T＝I＋N和B－E＝O－R，所以我们有$f(\mathrm{AT})=f(\mathrm{IN})$或$f(\mathrm{BE})=f(\mathrm{OR})$。注意表1的指令4和5相当好地解决了这个问题，同时保持rI1没有太宽的范围。

4．考虑具有k个对的情况，使

$$m^{-n}n!\sum_{0\leqslant k\leqslant n}\binom{m}{n-k}\binom{n-k}{k}2^{-k}<\frac{1}{2}$$

对于$m=365$为最小的n是88。如果你邀请88人，则生日试验的机会是.5111，但如果来了87人，则它就降低到.4995。

5．杂凑函数是坏的，因为它假定顶多有26个不同的值，而且它们中的某些要比其它的那些更经常地出现。甚至对于双重杂凑（比如说设$h_2(K)=1$加上K的第二个字节，而且$M=101$），查找所减慢的时间将超过因杂凑加快而节省的时间。还有$M=100$太小，因为FORTRAN程序通常有多于100个不同的变量。

6．在MIX上不行，因为算术溢出几乎总出现（被除数太大）〔最好能计算$(wK)\bmod M$，特别是如果线性探查以$c=1$被使用时，但不幸的是大多数计算机都不允许这一点，因为商溢出了〕。

7．如果$R(x)$是$P(x)$的一个倍式，则对于所有的$j\in s$，$R(\alpha^j)=0$在$GF(2^k)$中。设$R(x)=x^{a_1}+\cdots+x^{a_s}$，其中$a_1>\cdots>a_s\geqslant 0$且$s\leqslant t$，并且选择$t-s$个进一步的值$a_{s+1}$，…，$a_t$，使得$a_1$，…，$a_t$是小于$n$的不同的非负整数，范德蒙德矩阵

$$\begin{pmatrix}\alpha^{a_1}\cdots\alpha^{a_t}\\ \alpha^{2a_1}\cdots\alpha^{2a_t}\\ \vdots\quad\ \vdots\\ \alpha^{ta_1}\cdots\alpha^{ta_t}\end{pmatrix}$$

是奇异的，因为它的前s个列之和为0。但这同$\alpha^{a_1},\cdots,\alpha^{a_t}$是$GF(2^k)$的不同元素这一事实矛盾（见习题1.2.3-37）。

〔多项式杂凑的思想由M.哈南（M.Hanan）、S.默洛加（S.Muroga）、F.P.佩勒莫（F.P.Palermo）、N.雷弗（N.Raver）以及G.沙伊所首创；见*IBM J. Research & Development* 7(1963)121-129；*U.S.Patent* 3311888(March 28 1967)。〕

8．由归纳法，强归纳假设可通过对于$0\leqslant r\leqslant a_k$，$\{(-1)^k(rq_k+q_{k-1})\theta\}=(-1)^k(rq_k\theta-p_k)+(q_{k-1}\theta-p_{k-1})$这一事实而得到补充。对于$n=q_1$，$q_2+q_1$，$2q_2+q_1$，…，$a_2q_2+q_1=0q_4+q_3$，$q_4+q_3$，…，$a_4q_4+q_3=0q_6+q_5$，…出现$\{n\theta\}$的“创记录地低”的值；对于$n=q_0$，$q_1+q_0$，…，$a_1q_1+q_0=0q_3+q_2$，…出现“创记录地高”的值。这些是当形成一个具有新的长度编号为0的区间时的一些步骤。

9．我们有$\phi^{-1}=/1,1,1,\cdots/$和$\phi^{-2}=/2,1,1,\cdots/$。设在习题8的记号下$\theta=/a_1,a_2,\cdots/$和$\theta_k=/a_{k+1},a_{k+2},\cdots/$，以及$Q_k=q_k+q_{k-1}\theta_{k-2}$。如果$a_1>2$，则第一次分割是坏的，习题8中的三个区间大小分别是$(1-r\theta_{k-1})/Q_k$，θ_{k-1}/Q_k和$(1-(r-1)\theta_{k-1})/Q_k$，所以第一个长度对于第二个的比率是$(a_k-r)+\theta_k$，当$r=a_k$和$a_{k+1}\geqslant 2$时，这将小于$\frac{1}{2}$；

因此，如果我们不希望有坏的分割，则$\{a_2, a_3, \cdots\}$必须都等于1〔关于有关的定理，参考R. L. Graham和J. H. van Lint, Canadian J. Math. 20 (1968), 1020-1024，以及那里所引的参考文献。〕

10. 见F. M. 梁（F. M. Liang）在Discrete Math. 28 (1979), 325-326中漂亮的证明。

11. 如果$K=0$，则将是有问题的。如果要求键如同在程序L中那样是非0的，则这个变化将是值得的，而且我们也可以用0表示空位置。

12. 我们可以把K存于KEY(0)中，用下列诸行代替行14～19：

	STA	TABLE(KEY)	$A-S1$
	CMPA	TABLE,2(KEY)	$A-S1$
	JE	3F	$A-S1$
2H	ENT1	0,2	$C-1-S2$
	LD2	TABLE,1(LINK)	$C-1-S2$
	CMPA	TABLE,2(KEY)	$C-1-S2$
	JNE	2B	$C-1-S2$
3H	J2Z	5F	$A-S1$
	ENT1	0,2	$S2$
	JMP	SUCCESS	$S2$

时间的“节省”是$C-1-5A+S+4S1$，它实际上是一项纯粹的损失，因为C很少大于5。（一个内部循环并不总应该优化！）

13. 如同在算法C中那样，设表格的每个项中有一个附加的二进位TAG〔i〕场，以此来区别两种类型的项，这个解决办法使用循环表，并遵循艾伦·纽厄尔的建议，同时在每个表的第一个字中TAG〔i〕$=1$。

A1.〔初始化〕 置$i \leftarrow j \leftarrow h(K)+1$，$Q \leftarrow q(K)$。

A2.〔是否有一个表？〕 如果TABLE〔i〕是空的，则置TAG〔i〕$\leftarrow 1$而且转到A8，否则，如果TAG〔i〕$=0$，则转到A7。

A3.〔比较〕 如果$Q=$KEY〔i〕，则这个算法成功地结束。

A4.〔前进到下一个〕 如果LINK〔i〕$\neq j$，则置$i \leftarrow$LINK〔i〕，并转回A3。

A5.〔求空的节点〕 R减值一次或多次直到找出一个值，使得TABLE〔R〕是空的，如果$R=0$，则这个算法以溢出结束；否则置LINK〔i〕$\leftarrow R$。

A6.〔准备插入〕 置$i \leftarrow R$，TAG〔R〕$\leftarrow 0$，并转到A8。

A7.〔移动一个记录〕 重复地置$i \leftarrow$LINK〔i〕一次或多次，直到LINK〔i〕$=j$，然后作步骤A5，然后置TABLE〔R〕$\leftarrow$TABLE〔j〕，$i \leftarrow j$，TAG〔j〕$\leftarrow 1$。

A8.〔插入新的键〕 标记TABLE〔i〕为一个已占用的节点，且KEY〔i〕$\leftarrow Q$，LINK〔i〕$\leftarrow j$。〔注意，如果TABLE〔i〕是已占用的，则仅仅给定i的值，就有可能确定对应的完全的键。我们有$q(K)=$KEY〔i〕，然后如果置$i \leftarrow$LINK〔i〕0次或多次，直到TAG〔i〕$=1$，则我们将有$h(K)=i-1$。〕

14. J. S. 维特（J. S. Vitter）已经提出另一种方法，使用把表列结合在一起，而不是分开拉链，在一个很大的链接内存的“顶上”设置一个杂凑表〔Ph. D. thesis, Stanford

Univ.(1980), 72-73]。

插入一个新的键K：置Q⇐AVAIL，TAG(Q)←1，并存K于这个字中〔或者，如果所有键都是短的，则省略这一步而且在下列各步中以K代替Q〕。然后置R⇐AVAIL，TAG(R)←1，AUX(R)←Q，LINK(R)←A。置P←$h(K)$，然后：

如果TAG(P)＝0，则置TAG(P)←2，AUX(P)←R。

如果TAG(P)＝1，则置S⇐AUAIL，CONTENTS(S)←CONTENTS(P)，TAG(P)←2，AUX(P)←R，LINK(P)←S。

如果TAG(P)＝2，则置LINK(R)←AUX(P)，AUX(P)←R。

为检索一个键K：置P←$h(K)$，然后

如果TAG(P)≠2，则K是不存在的。

如果TAG(P)＝2，则置P←AUX(P)，然后置P←LINK(P)0次或多次直到TAG(P)＝1，而且AUX(P)指向（也许间接地通过TAG＝2的字指向）含有K的一个字（或者如果键是短的则AUX(P)＝K）或LINK(P)＝A为止。

(埃尔科克原先的方案，*Comp. J.* 8 (1965)，242-243，实际上使用TAG＝2和TAG＝3来区别长度为1的表列（当我们可以节省一个字的空间时）及更长的表列，这倒是一个值得的改进，因为我们大概有这样一个几乎所有的表列长度都为1的很大的散列表。)

15．如果知道总有一个空的节点，则会使得内部查找循环更快些，因为我们不需要维持一个计数器以确定步骤L2被实施多少次，较短的程序足以补偿这一个浪费的单元〔另一方面，有一种简洁方式省略变量N并允许这个表格在算法L中变成完全满的，而不显著地减慢这个方法，除非当这个表真正地溢出了：简单地校验$i<0$是否发生两次！这项技巧不可应用于算法D〕。

16．否；0总导致SUCCESS，无论它已经插入与否，而且SUCCESS在不同的时间以不同的i值出现。

17．然后第二个探查将总是对于位置0进行。

18．同（30）相对照，（31）中的代码花费大约$3(A-S1)$单位，它节省了$4u$乘(26)、(27)和(28)、(29)之间的差，对于一次成功的查找，仅当表是94%以上满时（31）才是有利的，而且它决不节省多于大约$\frac{1}{2}u$的时间。对于一次不成功的查找，当这个表是高于71%满时，（31）是有利的。

20．我们要证明

$$\binom{j}{2}\equiv\binom{k}{2}(\text{modulo}\,2^m)\ \text{和}\ 1\leqslant j\leqslant k\leqslant 2^m$$

意味着$j=k$。我们发现$j(j-1)\equiv k(k-1)(\text{modulo}\,2^{m+1})$意味着$(k-j)(k+j-1)\equiv 0$，如果$k-j$是奇数，则$k+j-1$必须是$2^{m+1}$的一个倍数，但这是不可能的，因为$2\leqslant k+j-1\leqslant 2^{m+1}-2$。因此$k-j$是偶数，所以$k+j-1$是奇数，所以$k-j$是$2^{m+1}$的一个倍数，所以$k=j$〔反之，如果$M$不是2的幂次，则这个探查序列无效〕。

探查序列有二次堆集，而且它增加程序D的运行时间（如同在（30）中修改的那样）

大约$\frac{1}{2}(C-1)-(A-S1)$ 单位，因为$B\approx\binom{C+1}{3}\Big/M$现在是可以忽略的。在这个表达到大约 60%满以前，这是一项小的改进。

21．如果N被减少，则算法D可能失误，因为它可能达到一种没有可用空间的状态，而且它将无限地循环。另一方面，如果N未被减少，则当仍然有空间时，算法D可以标志溢出。后一选择的弊病较少，因为重新杂凑可用来恢复被删去的单元（注意在这种情况下算法D将使N增值，而且仅当插入一项到一个以前的空位置时才作溢出判断，因为N表示非空的位置数）。我们也可以有两个计数器。

22．假设位置$j-1$，$j-2$，…，$j-k$被占用，而且$j-k-1$为空(modulo$M$)，则被插入之前先探查位置j并发现该位置已被占用的那些键，恰恰是在位置$j-1$到$j-k$之间的那些键，这些键的杂凑地址不处于$j-1$和$j-k$之间；这些“有问题的键”以插入的次序出现。算法R把第一个这样的键移动到位置j去，并在一个较小的有问题位置的范围内重复这个过程，直到不再剩下有问题的键为止。

23．J.S.维特〔Ph.D.thesis，Stanford Univ.（1980），61-68〕已经给出了进行结合拉链的一个删去法，它保持这个查找时间的分布。

24．$P(P-1)(P-2)P(P-1)P(P-1)/MP(MP-1)\cdots(MP-6)=M^{-7}\times(1-(5-21/M))P^{-1}+O(P^{-2})$)。一般地，一个杂凑序列$a_1\cdots a_N$的概率是$(\Pi_{0\leqslant j<M}P^{b_j})/MP^N=M^{-N}+O(P^{-1})$，其中$b_j$是等于$j$的$a_i$的个数。

25．设第（$n+1$）个键杂凑到位置a；P_k是M^{-N}乘使得k个位置a，$a-1$，…，$a-k+1$(modulo M)被占用和$a-k$个为空的杂凑序列数。由这个算法的循环对称性，具有$a+1$，…，$a+t$个被占用和$a+t+1$个空的这种序列的数目是$g(M,N,t+k)$。

26．$\frac{9!}{2!2!4!1!}f(3,2)f(3,2)f(5,4)f(2,1)=2^2\cdot3^5\cdot5^4\cdot7=4252500$

27．遵循提示

$$s(n,x,y)=\sum_{k\geqslant0}\binom{n}{k}x(x+k)^k(y-k)^{n-k-1}(y-n)$$
$$+n\sum_{k\geqslant0}\binom{n-1}{k-1}(x+k)^k(y-k)^{n-k-1}(y-n)$$

在第一个和中，以$n-k$代替k，而且应用阿贝尔公式；在第二个和中，以$k+1$代替k，现在

$$g(M,N,k)=\binom{N}{k}(k+1)^{k-1}(M-k-1)^{N-k-1}(M-N-1)$$

而且当$k=N=M-1$时有$0/0=1$，而且

$$M^N\Sigma(k+1)P_k=\sum_{k\geqslant0}\binom{k+2}{2}g(M,N,k)$$
$$=\frac{1}{2}\left(\sum_{k\geqslant0}(k+1)g(M,N,k)+\sum_{k\geqslant0}(k+1)^2g(M,N,k)\right)$$

第一个和是 $M^N\Sigma P_k=M^N$，而第二个是 $s(N,1,M-1)=M^N+N(2M^{N-1}+(N-1)\times(3M^{N-2}+\cdots))=M^NQ_2(M,N)$〔关于类似于 $s(n,x,y)$ 的和的进一步研究，见 J. Riordan, *Combinatorial Identities* (New York: Wiley, 1968), 18-23。〕

28. 设 $t(n,x,y)=\Sigma_{k\geqslant 0}\binom{n}{k}(x+k)^{k+2}(y-k)^{n-k-1}(y-n)$；然后如同在习题 27 中那样，我们求得 $t(n,x,y)=xs(n,x,y)+nt(n-1,x+1,y+1)$，$t(N,1,M-1)=M^N(3Q_3(M,M)-2Q_2(M,N))$。于是 $\Sigma(k+1)^2P_k=M^{-N}\Sigma\left(\frac{1}{3}(k+1)^3+\frac{1}{2}(k+1)^2+\frac{1}{6}(k+1)\right)g(M,N,k)-Q_3(M,N)-\frac{2}{3}Q_2(M,N)+\frac{1}{2}Q_1(M,N)+\frac{1}{6}$。减去 $(C'_N)^2$ 给出方差，它近似地等于 $\frac{3}{4}(1-\alpha)^{-4}-\frac{2}{3}(1-\alpha)^{-3}-\frac{1}{12}$。标准离差通常大于均值；例如，当 $\alpha=.9$ 时，均值是 50.5 而标准离差是 $\frac{1}{2}\sqrt{27333}\approx 83$。

29. 设 $M=m+1$，$N=n$；安全的停车序列恰是那样一些，其中当应用算法 L 于杂凑序列 $(M-a_1)\cdots(M-a_n)$ 时，位置 0 是空的，因此答案是 $f(m+1,n)=(m+1)^n-n(m+1)^{n-1}$〔这个停车问题是由 A. G. 康海姆和 B. 韦斯开始研究的，见 *SIAM J. Applied Math.* 14 (1966), 1266-1274，他们首先发表了对于算法 L 的一个正确的分析〕。

30. 显然，如果小汽车都停下了，则它们定义了这样一个序列。反之，如果存在 $p_1p_2\cdots p_n$，设 $q_1q_2\cdots q_n$ 是逆排列（$q_i=j$ 当且仅当 $p_j=i$），且 b_i 是等于 i 的 a_j 的个数，如果我们可以证明 $b_n\leqslant 1$，$b_{n-1}+b_n\leqslant 2$ 等等，或等价地 $b_1\geqslant 1$，$b_1+b_2\geqslant 2$ 等等，则每一辆小汽车都能停下；但这显然是真的，因为 k 个元素 a_{q_1}，…，a_{q_k} 全都 $\leqslant k$。

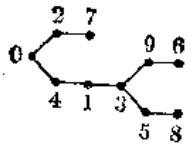
〔设 r_j 是 q_j 的"左影响"，即当且仅当 $q_{j-1}<q_j$，…，$q_{j-k+1}<q_j$ 且 $j=k$ 或 $q_{j-k}>q_j$ 时 $r_j=k$。在支配一个给定的醒来序列 $a_1\cdots a_n$ 的所有排列 $p_1\cdots p_n$ 当中，"立即停车"算法求得最小者（在字典编辑次序下）。康海姆和韦斯发现，导致一个给定排列 $p_1p_2\cdots p_n$ 的醒来序列数目是 $\Pi_{1\leqslant j\leqslant n}r_j$；值得注意的是，取遍所有排列 $q_1\cdots q_n$，这些乘积之和是 $(n+1)^{n-1}$。〕

31. 可能有许多有趣的联系，而且下列两个是作者特别喜爱的：

a）在上述答案的记号下，当且仅当 $(b_1, b_2, \cdots b_n, 0)$ 是在先序下树节点叉数的一个正确序列时，计数 b_1，b_2，…，b_n 对应于一个完全的停车序列（参考 2.3.3-9，它画出了后序）。每株这样的树都对应于 $\{0,\cdots,n\}$ 上 $n!/b_1!\cdots b_n!$ 株不同的带标号的自由树，因为我们能让 0 作为根的标号，而且对于 $k=1,2,\cdots,n$，我们可以逐次地由剩下未使用的标号中以 $(b_k+\cdots+b_n)!/b_k!(b_{k+1}+\cdots+b_n)!$ 种方式选择在先序下第 k 个节点的儿子的标号，以从左到右递增的次序附加诸标号。而且，每一个这样的计数序列，都对应于 $n!/b_1!\cdots b_n!$ 个醒来序列。

b）多米尼克·福塔已经给出下列别致的一一对应关系：设 $a_1\cdots a_n$ 是安全停车序列，它总在空位 j 处停放车 q_j。对于 $1\leqslant j\leqslant n$，当 $a_j=1$ 时从 j 到 0 画一条边，否则从 j 到 q_{a_j-1} 画一条边，就构造出了 $\{0,1,\cdots,n\}$ 上的一株带标号的自由树（把这株树的节点想成是汽车，汽车 j 被连接到另一辆车上，该车恰恰停在妻子 j 醒来处前面的位置上。于是，

习题 29 中所示的例子对应于自由树



由这株树出发，假定诸箭头都从根 0 发出，在每步选择最小的“源”，通过拓扑排序，可以得到停放的小汽车序列，从这个序列，$a_1\cdots a_n$ 可被重新构造出来。

实际上（a）和（b）中的两个构造是密切相关的。关于进一步的信息，见多米尼克·福塔和约翰·赖尔登“Mappings of acyclic and parking functions”（待出）。

32．设下标被循环处理，使得 $c_M=c_0$，$c_{M+1}=c_1$，等等。若对于所有的 j 有 $c_j=b_j+c_{j+1}-1$，则没有解，因为对于所有 j 之和将给出 $\Sigma c_j=\Sigma b_j+\Sigma c_j-M<\Sigma c_j$。因此，每个解有 $M-\Sigma b_j$ 个 j 的值，使得 $b_j=c_{j+1}=0$。如果 $(c'_0,\ \cdots,\ c'_{M-1})$ 是一个不同的解，则我们必然对所有的 j 有 $c'_{j+1}>0$；但这意味着 $c'_{j+2}>c_{j+2}$，$c'_{j+3}>c_{j+3}$，…，矛盾。这个解可以通过在 $c_0=0$ 的假定基础上定义 c_{M-1}，c_{M-2}，…而找到；然后，如果结果表明 c_0 大于 0，则只须重新定义 c_{M-1}，c_{M-2}，…，直到不再作进一步的改变为止〔用于从诸 b 中计算 Σc_j 的一个稍微更有效的算法，已经由 T. C. 洛（T. C. Lowe）和 D. C. 罗伯特（D. C. Robert）所给出（待出）〕。

33．诸概率不是独立的，因为条件 $b_0+b_1+\cdots+b_{M-1}=N$ 未考虑在内；这个推导允许 Σb_j 有任何给定的非负值的概率为非零，等式（46）不是严格地正确的；例如，它们意味着对于所有的 k，q_k 为正，这同 c_j 决不超过 $N-1$ 的事实矛盾。

34．（a）有 $\binom{n}{k}$ 种方式来选择 j 的集合使得 a_j 有一个特定的值，而且有 $(M-1)^{N-k}$ 种方式来赋值给其它的 a。因此

$$P_{Nk}=\binom{N}{k}(M-1)^{N-k}/M^N$$

（b）（50）中 $P_N(z)=B(z)$。（c）考虑为找出所有键的探查总数，不计算取图 38 中表列头表内指针的次数（如果使用这样的表的话）。长度为 k 的一个表列对总和数贡献了 $\binom{k+1}{2}$；因此

$$C_N=M\sum\binom{k+1}{2}P_{Nk}/N=(M/N)\left(\frac{1}{2}P''_N(1)+P'_N(1)\right)$$

（d）在情况（i）长度为 k 的一个表列要求 k 次探查（不计取表头），而在情况（ii），它要求 $k+\delta_{k0}$。于是在情况（ii），我们得到 $C'_N=\Sigma(k+\delta_{k0})P_{Nk}=P'_N(1)+P_N(0)=N/M+(1-1/M)^N\approx\alpha+e^{-\alpha}$，而情况（i）简单地有 $C'_N=N/M=\alpha$。公式 $MC'_N=M-N+NC_N$ 适用于情况（iii），因为 $M-N$ 个杂凑地址将发现一个空表位置，而 N 将引起去查找某个表列的结尾；这就得到（18）。

35．（i）$\Sigma\left(1+\frac{1}{2}k-(k+1)^{-1}\right)P_{Nk}=1+N/2M-M(1-(1-1/M)^{N+1})/(N$

$+1)\approx 1+\frac{1}{2}\alpha-(1-e^{-\alpha})/\alpha$。(ii) 在 (i) 的结果中加上 $\Sigma\delta_{k0}P_{Nk}=(1-1/M)^N\approx e^{-\alpha}$。(iii) 假定当一次不成功的查找在长度为 k 的一个表列的第 j 个元素处开始，给定的键相对于其它 k 个元素有随机次序，所以预期的查找长度是 $(j\cdot 1+2+\cdots+(k+1-j)+(k+1-j))/(k+1)$。现在对 j 求和给出 $MC'_N=M-N+M\Sigma(k^3+9k^2+2k)P_{Nk}/6(k+1)=M-N+M\left(\frac{1}{6}N(N-1)/M^2+\frac{3}{2}N/M-1+(M/(N+1))(1-(1-1/M)^{N-1})\right)$；因此 $C'_N\approx\frac{1}{2}\alpha+\frac{1}{6}\alpha^2+(1-e^{-\alpha})/\alpha$。

36. (i) $N/M-N/M^2$。(ii) $\Sigma(\delta_{k0}+k)^2P_{Nk}=\Sigma(\delta_{k0}+k^2)P_{Nk}=P_N(0)+P''_N(1)+P'_N(1)$。减去 C'^2_N 给出答案 $(M-1)N/M^2+(1-1/M)^N(1-2N/M-(1-1/M)^N)\approx\alpha+e^{-\alpha}(1-2\alpha-e^{-\alpha})\leqslant 1-e^{-1}-e^{-2}=0.4968$。〔对于数据结构 (iii)，将需要象在习题 37 中那样一个更复杂的分析。〕

37. 设 S_N 是 $(C_N-1)^2$ 的平均值；则

$$
\begin{aligned}
M^NNS_N &= \frac{1}{3}\sum\binom{N}{k_1,\ \cdots,\ k_M}\left(k_1\left(k_1-\frac{1}{2}\right)(k_1-1)+\cdots+k_M\left(k_M-\frac{1}{2}\right)(k_M-1)\right)\\
&= \frac{1}{3}M\sum\binom{N}{k}(M-1)^{N-k}k\left(k-\frac{1}{2}\right)(k-1)\\
&= \frac{1}{3}MN(N-1)(N-2)\sum\binom{N-3}{k-3}(M-1)^{N-k}\\
&\quad+\frac{1}{2}MN(N-1)\sum\binom{N-2}{k-2}(M-1)^{N-k}\\
&= \frac{1}{3}MN(N-1)(N-2)M^{N-2}+\frac{1}{2}MN(N-1)M^{N-2}
\end{aligned}
$$

方差是 $S_N-((N-1)/2M)^2=(N-1)(N+6M-5)/12M^2\approx\frac{1}{2}\alpha+\frac{1}{12}\alpha^2$。

38. 由等式 6.2.2-5,6，在不成功的情况下平均探查次数为 $\Sigma P_{Nk}(2H_{k+1}-2+\delta_{k0})$，在成功的情况下为 $(M/N)\Sigma P_{Nk}k(2(1+1/k)H_k-3)$。这些和分别等于 $2f(N)+2M(1-(1-1/M)^{N+1})/(N+1)+(1-1/M)^N-2$ 和 $2(M/N)f(N)+2f(N-1)+2M(1-(1-1/M)^N)/N-3$，其中 $f(N)=\Sigma P_{Nk}H_k$。习题 5.2.2-57 告诉我们当 $N=\alpha M, M\to\infty$ 时 $f(N)=\ln\alpha+\gamma+E_1(\alpha)+O(M^{-1})$。

〔树杂凑首先是由 P. F. 温德利提出的，见 *Comp. J.* 3 (1960)，84-88。上述分析表明树杂凑不比简单的拉链好到足以为额外的链接场辩护（表列无论如何总是短的）；而且当 M 很小时，它不比纯粹树查找好到足以为杂凑时间辩护。〕

39. (对算法 C 的分析的这一方法是由 J. S. 维特提出的) 当 $k\geqslant 2$ 时，我们有 $c_{N+1}(k)=(M-k)c_N(k)+(k-1)c_N(k-1)$，而且 $\Sigma kc_N(k)=NM^N$。因此 $S_{N+1}=\Sigma_{k\geqslant 2}\binom{k}{2}\times c_{N+1}(k)=\Sigma_{k\geqslant 2}\binom{k}{2}((M-k)c_N(k)+(k-1)c_N(k-1))=\Sigma_{k\geqslant 1}\left((M+2)\binom{k}{2}+k\right)\times c_N(k)=(M+2)S_N+NM^N$。

因此 $$S_N=(N-1)M^{N-1}+(N-2)M^{N-2}(M+2)+\cdots+M(M+2)^{N-2}$$
$$=\frac{1}{4}(M(M+2)^N-M^{N+1}-2NM^N)。$$

考虑在不成功的查找中探查的总数，对 $h(K)$ 的所有M个值求和，长度为 k 的每个表列对总和贡献 $k+\delta_{k0}+\binom{k}{2}$，因此 $M^{N+1}C'_N=M^{N+1}+S_N$。

40．象习题 39 中的 S_N 那样定义 U_N，但以 $\binom{k+1}{3}$ 来代替 $\binom{k}{2}$，我们求出 $U_{N+1}=(M+3)U_N+S_N+NM^N$，因此 $U_N=\frac{1}{36}(M^N(M-6N)-9M(M+2)^N+8M(M+3)^N)$。方差为 $2U_N/M^{N+1}+C'_N-C'^2_N$，它趋于 $\frac{35}{144}-\frac{1}{12}\alpha-\frac{1}{4}\alpha^2+\left(\frac{1}{4}\alpha-\frac{5}{8}\right)e^{2\alpha}+\frac{4}{9}e^{3\alpha}-\frac{1}{16}e^{4\alpha}$，对于 $N=\alpha M$，$M\to\infty$。当 $\alpha=1$ 时，这大约是 4.50，所以标准离差大约以 2.12 为界。

41．设 V_N 是在表格的“高”端被占用单元的块区的平均长度。这个块区的长度为 k 的概率是 $A_{Nk}(M-1-k)^{\underline{N-k}}/M^N$，其中 A_{Nk} 是具如下性质的杂凑序列（35）的个数，它使得在算法 C 作用下，前 $N-k$ 个和最后 k 个单元为已占用的，并且使得子序列 1，2，…，$N-k$ 以递增次序出现，因此

$$M^NV_N=\Sigma kA_{Nk}(M-1-k)^{\underline{N-k}}=M^{N+1}-\Sigma(M-k)A_{Nk}(M-1-k)^{\underline{N-k}}$$
$$=M^{N+1}-(M-N)\Sigma A_{Nk}(M-k)^{\underline{N-k}}=M^{N+1}-(M-N)(M+1)^N$$

现在 $T_N=(N/M)(1+V_N-T_0-\cdots-T_{N-1})$，因为 $T_0+\cdots+T_{N-1}$ 是在此之前 R 减值的平均次数，且 N/M 是这次它减值的概率。这个递归式的解是 $T_N=(N/M)(1+1/M)^N$（这样一个简单的公式应该有一个更简单的证明！）。

42．$S1_N$ 是以 $A=0$ 被插入的项数除以 N。

43．设 $N=\alpha M'$ 和 $M=\beta M'$，并设 $e^{-\lambda}+\lambda=1/\beta$，$\rho=\alpha/\beta$，则若 $\rho\leqslant\lambda$，就有 $C_N\approx 1+\frac{1}{2}\rho$ 和 $C'_N\approx\rho+e^{-\rho}$；若 $\rho\geqslant\lambda$，则 $C_N\approx\frac{1}{8\rho}(e^{2(\rho-\lambda)}-1-2(\rho-\lambda))(3-2/\beta+2\lambda)+\frac{1}{4}(\rho+\lambda)+\frac{1}{4}\lambda(1-\lambda/\rho)$ 和 $C'_N\approx 1/\beta+\frac{1}{4}(e^{2(\rho-\lambda)}-1)(3-2/\beta+2\lambda)-\frac{1}{2}(\rho-\lambda)$。对于 $\alpha=1$，当 $\beta\approx.853$ 时，我们得到最小的 $C_N\approx 1.69$；当 $\beta\approx.782$ 时出现最小的 $C'_N\approx 1.79$。所以尽管较小范围的杂凑地址引起更多的碰撞出现，但它得到了把最初的碰撞放到不和杂凑地址冲突的一个区域内的好处。这些结果是由 J. S. 维特给出的〔Ph. D. thesis, Stanford Univ. (1980)；*Proc. Symp. Foundations Comp. Sci.* 21 (1980), 238-247〕。

44．从左到右，从 1 到 m 给阵列位置编号，考虑同等可能地具有 k 个“p 步骤”和 $n-k$ 个“q 步骤”的所有 $\binom{n}{k}$ 个操作序列的集合，设 $g(m, n+1, k, r)$ 是 $\binom{n}{k}$ 乘上前 $r-1$ 个位置变成已占用的和第 r 个保持为空的概率，于是 $g(m, l, k, r)$ 是 $(m-1)^{-(l-1-k)}$ 乘以下述的概率取遍所有配置

$$1\leqslant a_1<\cdots<a_k<l;\ (c_1,\ \cdots,\ c_{l-1-k}),\ 2\leqslant c_i\leqslant m$$

之和：所述概率是当第 a_i 个操作是一个 p 步骤，而剩下的 $l-1-k$ 个操作是分别以选择位置 c_1，…，c_{l-1-k} 开始的诸 q 步骤时，第一个空位置是 r 的概率。通过附加进一步的条件，即第 a_i 个操作占用位置 b_j，对于确定的

$$1\leqslant b_1<\cdots<b_k<r$$

对所有的配置求和，我们得到递归式

$$g(m,\ l,\ k+1,\ r)=\sum_{\substack{a<l\\ b<r\\ 1\leqslant b\leqslant a}}\frac{(l-b-1)!}{(l-r)!}\cdot\frac{(m-r)!}{(m-b)!}(m-l+1)g(m,\ a,\ k,\ b)$$

$$g(m,\ l,\ 0,\ r)=\frac{(l-1)!}{(l-r)!}\cdot\frac{(m-r)!}{m!}(m-l+1)\left(P_l+(1-\delta_{r1})\frac{m}{l-1}(1-P_l)\right)$$

其中 $P_l=(m/(m-1))^{l-1}$。由此得出，如果 $G(m,\ l,\ k)=\Sigma_{1\leqslant r\leqslant l}(m+1-r)\times g(m,\ l,\ k,\ r)$，则我们有

$$G(m,\ l,k+1)=\frac{m-l+1}{m-l+2}\sum_{1\leqslant a<l}G(m,\ a,\ k);\quad G(m,\ l,\ 0)=\frac{m-l+1}{m-l+2}(m+P_l)$$

对于所述问题的解是 $m-\Sigma_{0\leqslant k\leqslant n}p^kq^{n-k}G(m,\ n+1,\ k)$。它（在实施了某些技巧之后）等于 $m-((m-n)/(m-n+1))\ (Q_n+mR+pSR)$，其中

$$Q_j=P_{j+1}q^j$$

$$R=\left(1-\frac{p}{m+1}\right)\left(1-\frac{p}{m}\right)\cdots\left(1-\frac{p}{m-n+2}\right)=\prod_{0\leqslant j<m}\left(1-\frac{p}{m+1-j}\right)$$

$$S=\frac{\left(1-\frac{1}{m+1}\right)Q_0}{\left(1-\frac{p}{m+1}\right)}+\frac{\left(1-\frac{1}{m}\right)Q_1}{\left(1-\frac{p}{m+1}\right)\left(1-\frac{p}{m}\right)}+\cdots+\frac{\left(1-\frac{1}{m-n+2}\right)Q_{n-1}}{R}$$

$$=\sum_{0\leqslant j<N}\frac{(1-1/(m+1-j))Q_j}{\Pi_{0\leqslant i\leqslant j}(1-p/(m+1-i))}$$

当 $p=1/m$ 时，对于所有的 j，$Q_j=1$。设 $w=m+1$，$n=\alpha w$，$w\to\infty$，我们求得 $\ln R=-p(H_w-H_{w(1-\alpha)})+O(p^2)$；因此 $R=1+w^{-1}\ln(1-\alpha)+O(w^{-2})$，而且类似地 $S=\alpha w+O(1)$。于是答案是 $(1-\alpha)^{-1}-1-\alpha-\ln(1-\alpha)+O(w^{-1})$。

注意：较简单的问题“以概率 p 占用最左边，否则占用任何随机选定的空位置”，通过在上边取 $P_j=1$ 即可获得解决，而且答案是 $m-(m+1)(m-n)/(m-n+1)$。为了得到具二次堆集的随机探查的 C'_N，置 $n=N-1$，$m=M$，并在上边的答案中加 1。

45．是的。见 L. 吉巴斯（待出）。

46．对于所有的 x 和非负整数 n 通过规则

$$\sum_k\binom{x+k}{k}\left[\!\left[\begin{matrix}n\\k\end{matrix}\right]\!\right]=(x+n+1)^n$$

对于 $k\geqslant 0$ 定义 $\left[\!\left[\begin{matrix}n\\k\end{matrix}\right]\!\right]$，置 $x=-1,-2,\cdots n-1$ 意味着

$$\left[\left[\begin{matrix} n \\ k \end{matrix}\right]\right]=\sum_j \binom{k}{j}(-1)^j(n-j)^n \quad \text{对于 } 0\leqslant k\leqslant n;$$

然后置 $x=0$ 意味着对于所有 $k>n$，我们可以取 $\left[\left[\begin{matrix} n \\ k \end{matrix}\right]\right]=0$，所以定义方程的两边是在 $n+1$ 个点上相等的 x 的 n 次多项式，由此得出数 $\left[\left[\begin{matrix} n \\ k \end{matrix}\right]\right]$ 有所述的性质。

设 $f(N, r)$ 是保持头 r 个为已占用的和下一个为空的杂凑序列 $a_1a_2\cdots a_N$ 的数目，有 $\binom{M-r-1}{N-r}$ 种占用单元的形式，而且每一种形式出现的次数就等于序列 $a'_1\cdots a'_N$，$1\leqslant a'_i\leqslant N$ 的数目，这种序列包含数 $r+1, r+2, \cdots, N$ 的每一个至少一次，按包含-排斥原理，有 $\left[\left[\begin{matrix} N \\ N-r \end{matrix}\right]\right]$ 个这样的序列；因此

$$f(N, r)=\binom{M-r-1}{N-r}\left[\left[\begin{matrix} N \\ N-r \end{matrix}\right]\right]$$

现在 $C'_N=1+M^{-N-1}\Sigma_{0\leqslant r\leqslant N} f(N, r)(\Sigma_{0\leqslant a<r} r+\Sigma_{r<a<M}((N-r)/(M-r-1))(r+1))=1+M^{-N-1}\Sigma_{0\leqslant r\leqslant N} f(N, r)(N+(N-1)r)$。设

$$S_n(x)=\sum_k k\binom{x+k}{k}\left[\left[\begin{matrix} n \\ k \end{matrix}\right]\right]$$

我们有

$$(x+1)^n S_n(x)+\sum_k \binom{x+k}{k}\left[\left[\begin{matrix} n \\ k \end{matrix}\right]\right]=\sum_k \binom{x+1+k}{k}\left[\left[\begin{matrix} n \\ k \end{matrix}\right]\right]$$

因此，$S_n(x)=(x+1)((x+n+2)^n-(x+n+1)^n)$。由此得出 $C'_N=N(1+1/M)-(N-1)(1-N/M)(1+1/M)^N\approx N(1-(1-\alpha)e^\alpha)$；而且 $C_N=(N-1)((1+1/M)/2+(1+1/M)^N)+(3M^2+6M+2)((1+1/M)^N-1)/N-(3M+2)(1+1/M)^N$，当 $N=M-1$ 时，它是 $(e-2.5)M+O(1)$。

关于数 $\left[\left[\begin{matrix} n \\ k \end{matrix}\right]\right]$ 的进一步性质，参见 John Riordan, *Combinatorial Identities* (New York: Wiley, 1968), 228。

47. 算法L的分析几乎可逐字应用！任何带有循环对称性的探查序列，只要它仅仅勘察同从前考察过了的那些位置相邻的位置，都将有相同的特性。

48. $C'_N=1+p+p^2+\cdots$，其中 $p=N/M$ 是一个随机位置被填满的概率；因此 $C'_N=M/(M-N)$，而且 $C_N=M^{-1}\Sigma_{0\leqslant k<N}$，$C'_k=N^{-1}M(H_M-H_{M-N})$，这些值近似地等于一致探查时的那些值，但稍微高些，因为在相同的位置有多次探查的机会。其实对于 $4=N<M\leqslant 16$，线性探查是更好的！

实际上，我们将不使用无穷多个杂凑函数，某些其它的方案，象线性探查，将最终地被用于最后的重新排序，这个方法比正文中描述的那些方法低劣，但是它有着历史上的重

要性，因为它提示了导致算法D的莫里斯方法，见 *CACM* 6 (1963)，101，其中 M. D. 麦基尔罗伊 (M. D. McIlroy) 把这个思想归功于 V. A. 维索特斯基 (V. A. Vyssotsky)；同样的技术也由 A. W. 霍尔特早在 1956 年就发现了，他成功地把它使用于 UNIVAC 的 GPX 系统中。

49. $C'_N-1=\Sigma_{k>b}(k-b)P_{nk}\approx\Sigma_{k>b}(k-b)e^{-\alpha b}(\alpha b)^k/k!=\alpha b t_b(\alpha)$〔注意：一般说来，如果 $P(z)$ 是任何概率的母函数，则

$$\sum_{b\geqslant 0}\left(\sum_{k>b}(k-b)P_k\right)z^b=P'(1)/(1-z)+z(P(z)-1)/(1-z)^2$$

$$\begin{aligned}C_{N-1}-1&=(M/N)\sum_{k>b}\binom{k-b+1}{2}P_{Nk}\\&=(M/2N)\sum_{k>b}(k(k-1)-2k(b-1)+b(b-1))P_{Nk}\\&\approx\frac{1}{2}e^{-b\alpha}(b\alpha)^b b!^{-1}(b+b\alpha-2b+2+(b\alpha^2-2\alpha(b-1)+b-1)R(\alpha,\ b))\end{aligned}$$

〔对于具有拉链的成功查找的分析，首先是由 W. P. 海亭于1957年进行的。(57)、(58)中的简单表达式是由 J. A. 范·德·普尔 (J. A. Van der Pool) 于1971年建立的；他也考虑了怎样把一个表示存储空间和存取次数的综合代价的函数极小化。因为 $\Sigma_{k>b}(k-b)^2\times P_{Nk}=(2N/M)(C_N-1)-(C'_N-1)$，我们可以确定 C'_N 和每个桶溢出次数的方差。溢出总数的方差可以通过把 M 乘以在一个桶中的方差来逼近，但这实际上太高了，因为总的记录数已被限制为 N。真正的方差可以象在习题37中那样来求得，也请参考 3.3.1 c 节中的 x 平方测试的推导。〕

50. 而且其次 $Q_0(M,\ N-1)=(M/N)(Q_0(M,\ N)-1)$。

51. $R(\alpha,\ n)=\alpha^{-1}(n!e^{\alpha n}(\alpha n)^{-n}-Q_0(\alpha n,\ n))$。

52. 参考等式1.2.11.3-9和习题3.1-14。

53. 由等式1.2.11.3-8，$\alpha(\alpha n)^n R(\alpha,\ n)=e^{\alpha n}\gamma(n+1,\ \alpha n)$；因此由提示的习题可知 $R(\alpha,\ n)=(1-\alpha)^{-1}-(1-\alpha)^{-3}n^{-1}+O(n^{-2})$（这个渐近公式可以更直接地通过(43)的方法得到，注意 $R(\alpha,\ n)$ 中 α^k 的系数是

$$1-\binom{k+2}{2}n^{-1}+O(k^4n^{-2})$$

事实上，由等式1.2.9-28，α^k 的系数是

$$\sum_{r\geqslant 0}(-1)^r n^{-r}\left\{{r+k+1 \atop k+1}\right\}$$

55. 如果 $B(z)C(z)=\Sigma s_i z^i$，则我们有 $c_0=s_0+\cdots+s_b$，$c_1=s_{b+1}$，$c_2=s_{b+2}$，…；因此 $B(z)C(z)=z^bC(z)+\alpha(z)$。现在 $P(z)=z^b$ 有 $b-1$ 个根 $|q_j|<1$，如同解 $e^{\alpha(q_j-1)}=\omega^{-j}q_j$，$\omega=e^{2\pi i/b}$ 那样确定之。为了求解 $e^{\alpha(q-1)}=\omega^{-1}q$，设 $t=\alpha q$ 和 $z=\alpha\omega e^{-\alpha}$，使得 $t=ze^t$。由拉格朗日公式我们得到

$$\frac{1}{1-q}=1+\sum_{r\geqslant 0}\gamma\sum_{n\geqslant r}\frac{n^{n-r-1}\omega^n\alpha^{n-\alpha}e^{-n\alpha}}{(n-r)!}$$

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

由阿贝尔极限定理，从单位圆内命$|\omega|\to 1$，这可以重新安排成

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

现在以ω^j代替ω，而且对于$1\leqslant j<b$求和，就把这式子简化成为

$$\frac{b-1}{2}+\alpha\frac{b-1}{2}+\sum_{m\geqslant 2}\alpha^m\left(-\frac{1}{2}+\frac{(-1)^m}{m!}b\sum_{n\geqslant 1}\binom{m-2}{nb-1}(-1)^{nb-1}(nb)^{m-1}\right)$$

采用了一些更多的技巧之后，便得了所希望的结果，因为

$$t_n(\alpha)=(-1)^{n-1}(n!\alpha)^{-1}\sum_{m>n}(-n\alpha)^m/m(m-1)(m-n-1)!$$

这项分析，可应用于广泛的问题中，它是由 N. T. J. Bailey，J. Roy. Stat. Soc. B16 (1954)，80-87；M. Tainiter, *JACM* 10(1963), 307-315；A. G. Konheim 和 B. Meister, *JACM* 19(1972)，92-108开始进行的。

58. 0 1 2 3 4和0 2 4 1 3，加1 1 1 1 1 mod 5的附加的移位，每个有$\frac{1}{10}$的概率[对于$M=6$，我们需要30个排例，而且存在一个从$\frac{1}{20}\times$ 0 1 2 3 4 5，$\frac{1}{60}\times$ 0 1 3 2 5 4，$\frac{1}{60}\times$ 0 2 4 3 1 5，$\frac{1}{20}\times$ 0 2 3 4 5 1，$\frac{1}{30}\times$ 0 3 4 1 2 5开始的解。对于$M=7$，我们需要49，而且通过$\frac{1}{35}\times$ 0 1 2 3 4 5 6，$\frac{2}{105}\times$ 0 1 5 3 2 4 6，$\frac{1}{35}\times$ 0 2 4 3 5 1 6，$\frac{2}{105}\times$ 0 2 6 3 1 4 5，$\frac{1}{35}\times$ 0 3 6 1 4 2 5，$\frac{1}{105}\times$ 0 3 2 6 4 1 5，$\frac{1}{105}\times$ 0 3 1 5 4 2 6生成一个解]。

59. 任何排列不能有大于

$$1\Big/\binom{M}{\lfloor M/2\rfloor}$$

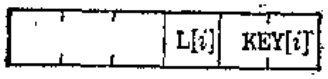
的概率，所以必然至少有

$$\binom{M}{\lfloor M/2\rfloor}=\exp(M\ln 2+O(\log M))$$

个排列有非0的概率。

63. 由习题3.3.2-8，MH_M 标准离差是$\pi M/\sqrt{6}$〔这就是简单地标记单元为“删去的”的删去方法的“失误的平均次数”〕。

65. 诸键可以被存储在一个顺序分配的独立的表格中（假定，如果有删去的话，它是按 LIFO方式进行的）。散列表的项指向这个“名字表”，例如 TABLE[i]可以有形式



其中L〔i〕是存储在单元KEY〔i〕，KEY〔i〕+1，…处的键中的字数。

散列表剩下的项可以以若干方式中的任何一种方式来使用：（a）作为算法C中的一个链接。（b）作为同键相关联的信息的一部分，或者（c）作为"第二个杂凑码"。后一个思想，是由罗伯特·莫里斯提出的，有时可加速一个查找〔对于某函数$h_2(K)$，仅当$h_2(K)$匹配它的第二个杂凑码时，我们仔细地考察KEY〔i〕中的键〕。

66．是；而且诸记录的安排是唯一的；每一次不成功的查找的平均探查数被减少成C_{N-1}，尽管当插入第N项时它保留C'_N。见Comp. J. 17(1974)，135-142。

67．设$q_k=p_k+p_{k+1}+\cdots$；然后$C_N=\sum_{k}k q_k$和$q_k\geqslant\max(0,\ 1-(k-1)(M-N)/M)$。

6.5 节

1．在从（$i-1$，j）运行到一个"新的最低记录"的值（i，$j-1$）的过程中，把每个往下的步骤改变成一个往上的步骤，这样来转化提示中描述的通路。如果做了c个这样的修改，则这条通路在（n，$n-2t+c$）处结束，其中$c\geqslant 0$和$c\geqslant 2t-n$；因此$n-2t+2c\geqslant n-2k$。在与被修改的通路相对应的排列中，表列B的最小的c个元素对应于被修改的诸向下步骤，而表列A包含$t-c$个对应于未修改的诸向下步骤。

当$t=k$时，不难看出这个构造是可逆的；因此恰恰构造了$\binom{n}{k}$个排列。注意，按照这个证明，表列A和C的内容可以以任意的次序出现。

附记：我们已经在习题2.2.1-4中以另一种方式计算了这些通路。当$k=\lfloor n/2\rfloor$时这个构造证明了斯佩纳（Sperner）引理，它指出，不可能有$\{1,\ 2\cdots,\ n\}$的$\binom{n}{\lfloor n/2\rfloor}$个以上的子集，使得这些子集中没有一个是包含于另一个当中的〔Emanule Sperner, *Math. Zeitschrift* 27(1928)，544-548〕。因为，如果我们有这样一个子集的集合，则$\binom{n}{k}$个排列中的每一个的初始位置上至多可以出现这些子集之一，但每个子集又应出现在某个排列中。这里所用的构造是一个更为一般的构造的改头换面的形式。N. G. 德·布鲁因等人〔*Nieuw Archief voor Wiskunde*(2)23(1951)，191-193〕通过这个构造，证明了斯佩纳的引理对多重集合的推广："设M是包含有n个元素（计算了多重性的）一个多重集合。M的所有$\lfloor n/2\rfloor$个元素的子多重集合的全体，是使得没有任何子多重集合包含在另一个当中的最大可能的这种集合。"例如，当$M=\{a,\ a,\ b,\ b,\ c,\ c\}$时，最大的这样的集合是由七个子多重集合$\{a,\ a,\ b\}$，$\{a,\ a,\ c\}$，$\{a,\ b,\ b\}$，$\{a,\ b,\ c\}$，$\{a,\ c,\ c\}$，$\{b,\ b,\ c\}$，$\{b,\ c,\ c\}$组成的。这对应于六个属性$A_1$，$B_1$，$A_2$，$B_2$，$A_3$，$B_3$的七种排列，在这些属性中，所有涉及$A_i$的询问也涉及$B_i$。

2．令a_{ijk}为对一些记录的所有访问的表列，这些记录分别以（i，j，k）为三个

属性的值，并假定 a_{011} 是三个表列 a_{011}，a_{101}，a_{110} 之最短者，则一个极小长度的表列是 $a_{001}a_{011}a_{111}a_{101}a_{100}a_{110}a_{111}a_{011}a_{010}$。然而，如果 a_{011} 是空的，而且 a_{001}，a_{010} 或 a_{100} 之一也是，则这个长度可以通过删去 a_{111} 的两个出现之一来缩短〔*CACM*15(1972)，802-808〕。

3.（a）茴香籽和/或蜂蜜，可能同豆蔻和/或香精组合。（b）无。

4. 设 p_t 是询问恰恰涉及 t 个二进位位置的概率，且 P_t 是在一个随机记录中给定的 t 个位置全部是 1 的概率，则答案是 $\Sigma_t p_t P_t$ 减去一个特定的记录是一个"真情报"的概率，即 $\binom{N-q}{r-z}\Big/\binom{N}{r}$，其中 $N=\binom{n}{k}$。由包含和排除原理

$$P_t=\sum_{j\geqslant 0}(-1)^j\binom{t}{j}f(n-j,\ k,\ r)/f(n,\ k,\ r)$$

其中 $f(n,\ k,\ r)$ 是在一个 n 位场中，选择 r 个不同的 k 位属性码的可能的数目，即

$$f(n,\ k,\ r)=\binom{\binom{n}{k}}{r}$$

而且（参考习题1.3.3-26）

$$\begin{aligned}p_t&=\sum_{i\geqslant 0}(-1)^i\binom{t+i}{i}\binom{n}{t+i}P_{t+i}|_{r=q}\\&=\binom{n}{t}\sum_{j\geqslant 0}(-1)^j\binom{t}{j}f(t-j,\ k,\ q)/f(n,\ k,\ q)\end{aligned}$$

注：上述计算的更一般的形式，首先是由 G. 奥罗斯兹（G. Orosz）和 L. 塔卡克斯（L. Takács）进行的，见 J. of Documentation12(1956)，231-234。均值 Σtp_t 容易证明是 $n((1-f(n-1,\ k,\ q)/f(n,\ k,\ q))$。另一个假定，即记录中和询问中的随机属性代码不必是不同的，如同在哈里森和布鲁姆的技术中那样，可以通过同样的方法来进行分析，并置 $f(n,\ k,\ r)=\binom{n}{k}^r$。当参数是在适当的范围中的，我们有 $P_t\approx(1-e^{-kt/n})^t$ 而且 $\Sigma p_tP_t\approx P_n(1-\exp(-kq/n))$。

6. $L(k)=\sum_j\binom{m_1}{j}\binom{m_2}{k-j}L_1(j)L_2(k-j)\Big/\binom{m_1+m_2}{k}$。〔因此如果 $L_1(k)\approx N_1\alpha^{-k}$ 而且 $L_2(k)\approx N_2\alpha^{-k}$，则 $L(k)\approx N_1N_2\alpha^{-k}$。〕

7.（a）$L(1)=3$，$L(2)=1\frac{3}{4}$。（b）$L(1)=3\frac{1}{4}$，$L(2)=2\frac{1}{3}$，$L(3)=1\frac{9}{16}$。〔注意：厄尔·萨喜多蒂（Earl Sacerdoti）已经提义映象

$$\begin{aligned}&0\ 0\ *\ *\to 1\\&0\ 1\ *\ *\to 2\\&1\ 0\ *\ *\to 3\\&1\ 1\ *\ *\to 4\end{aligned}$$

它有一个最坏的“最坏情况”，但是却有一个较好的平均情况：$L(1)=3$，$L(2)=2\frac{1}{6}$，$L(3)=1\frac{1}{2}$，当$m=2^n$时，存在一个类似的映象。〕

10．(a)必有$\frac{1}{6}v(v-1)$个三元组，而且x_v必然出现在它们的$\frac{1}{2}v$个之中。(b)因为v是奇数，对于每个i有唯一的三元组$\{x_i, y_i, z\}$，所以S'容易证明是一个斯坦纳三元组系统。在K'中不出现的对偶是$\{z, x_2\}$，$\{x_2, y_2\}$，$\{y_2, x_3\}$，$\{x_3, y_3\}$，…，$\{x_{v-1}, y_{v-1}\}$，$\{y_{v-1}, x_v\}$，$\{x_v, z\}$。(d)由情况$v=1$开始，应用操作$v\to 2v-2$，$v\to 2v+1$即得不具有$3k+2$形式的所有非负数，因为$6k+(1, 0, 1, 3)$诸情况分别地来自较小的情况$3k+(1, 0, 1, 3)$。

11．取$2v+1$个对象上的斯坦纳三元组系统，称诸对象之一为z，而且以这样一个方式来命名其余对象，即包含z的三元组是$\{z, x_i, \bar{x}_i\}$；删去这些三元组。

12．对于$0\leqslant k<14$，$\{k, (k+1)\bmod 14, (k+4)\bmod 14, (k+6)\bmod 14\}$，其中$(k+7)\bmod 14$是$k$的补〔补系统是所谓可除群区组设计的特殊情况；参看博斯(Bose)、施里克汗德(Shrikhande)和巴达查利亚(Bhattacharya)，*Ann. Math. Statistics*24(1953)，167-195〕。

16．设每一三元组对应于一个码字，其中每个码字恰有三个1的二进位，这些二进位标识对应的三元组的元素。如果u，v，w是不同的码字，v与w的叠加至多与u有两个公共的二进位1，因为它与单个的v或w至多有一位相同〔类似地，从阶为v的四元组系统我们可以构造$v(v-1)/12$个码字，它们中没有一个被包含在任何其余三个的叠加当中，等等〕。

附 录 A

数 值 量 表

表 1　在计算机程序分析和在标准子程序中经常使用的一些量（40个十进位）

$\sqrt{2}=1.$	41421	35623	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	−
$\sqrt[4]{2}=1.$	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	−
$\ln 10=2.$	30258	50929	94045	68401	79914	54684	36420	76011	+
$1/\ln 2=1.$	44269	50408	88963	40735	99246	81001	89213	74266	+
$1/\ln 10=0.$	43429	44819	03251	82765	11289	18916	60508	22944	−
$\pi=3.$	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	+
$\pi^2=9.$	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	40074	67764	41287	−
$\Gamma(2/3)=1.$	35411	79394	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	+
$\gamma=0.$	57721	56649	01532	86060	65120	90082	40243	10422	+
$\ln\pi=1.$	14472	98858	49400	17414	34273	51353	05871	16473	+
$\phi=1.$	61803	39887	49894	84820	45868	34365	63811	77203	+
$e^\gamma=1.$	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	−
$\ln\phi=0.$	48121	18250	59603	44749	77589	13424	36842	31352	−
$1/\ln\phi=2.$	07808	69212	35027	53760	13226	06117	79576	77422	−
$-\ln\ln 2=0.$	36651	29205	81664	32701	24391	58232	66946	94543	−

表 2　在八进制记法下，在标准子程序和在计算机程序分析中经常使用的一些量，在等号左边的每个量的名称，是以十进记号给出的

$0.1=$	0.06314	63146	31463	14631	46314	63146	31463	14631	4632
$0.01=$	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	56053	3442
$0.000001=$	0.00000	02061	57364	05536	66151	55323	07746	44470	2603
$0.0000001=$	0.00000	00153	27745	15274	53644	12741	72312	20254	0215
$0.00000001=$	0.00000	00012	57143	56106	04303	47374	77341	01512	6333
$0.000000001=$	0.00000	00001	04560	27640	46655	12262	71426	40121	2174
$0.0000000001=$	0.00000	00000	06676	33766	35367	55653	37265	34642	0163
$\sqrt{2}=$	1.32404	74631	77167	46220	42627	66115	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
$\sqrt[4]{2}=$	1.14067	74050	61556	12455	72152	64430	60271	02755	7314
$\ln 2=$	0.54271	02775	75071	73632	57117	07316	36007	71366	5364
$\ln 3=$	1.06237	24752	55006	05227	32440	63065	25012	35574	5534
$\ln 10=$	2.23273	06735	52524	25405	56512	66542	56026	46050	5071
$1/\ln 2=$	1.34252	16624	53405	77027	35750	37766	40644	35175	0435
$1/\ln 10=$	0.33626	75425	11562	41614	52325	33525	27655	14756	0622
$\pi=$	3.11037	55242	10264	30215	14230	53050	56006	70163	2112
$1^\circ=\pi/180=$	0.01073	72152	11224	72344	25603	54276	63351	22056	1155
$1/\pi=$	0.24276	30155	62344	20251	23760	47257	50765	15156	7007
$\pi^2=$	11.67517	14467	62135	71322	25561	15466	30021	40654	3410
$\sqrt{\pi}=\Gamma(1/2)=$	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	00471	72363	6166
$1/e=$	0.27426	53066	13167	46761	52726	75436	02440	52371	0336
$e^2=$	7.30714	45615	23355	33460	63507	35040	32664	25356	5022
$\gamma=$	0.44742	14770	67666	06172	23215	74376	01002	51313	2552
$\ln\pi=$	1.11206	40443	47503	36413	65374	52661	52410	37511	4606
$\phi=$	1.47433	57156	27751	23701	27634	71401	40271	66710	1501
$e^\gamma=$	1.61772	13452	61152	65761	22477	36553	53327	17554	2126
$e^{\pi/4}=$	2.14275	31512	16162	52370	35530	11342	53525	44307	0217
$\sin 1=$	0.65655	24436	04414	73402	03607	23644	11612	07474	1451
$\cos 1=$	0.42450	50037	32406	42711	07022	14666	27320	70675	1232
$s(3)=$	1.14735	00023	60014	20470	15613	42561	31715	10177	0662
$\ln\phi=$	0.36630	26256	61213	01145	13700	41004	52264	30700	4065
$1/\ln\phi=$	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 的若干值是由小约翰·W·伦奇（John W. Wrench, Jr.）计算出来的。

对于在这份表中未出现的高精度的常数值，见 J. 彼得斯（J. Peters），*"Ten Place Logarithms of Numbers from 1 to 100000"*, Appendix to Volume 1 (New York: F. Vngar Publ. Co., 1957)；以及 *"Handbook of Mathematical Functisns"*, ed. by. M.

Abramowitz and I. A. Stegun (Washington, D. C.: V. S. Govt. Printing Office, 1964), *Chapter* 1。

在同排序和查找算法的分析相联系时出现了一些没有通用名字的有趣的常数；这些常数的40位值出现在等式5.2.3-19和在习题5.2.3-27、5.2.4-13以及6.3-27的答案中。

表3　对于小n值的调和数、贝努利数以及斐波那契数

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$H_x=\sum_{n\geq 1}\left(\frac{1}{n}-\frac{1}{n+x}\right)$，则

$$H_{1/2}=2-2\ln 2$$

$$H_{1/3}=3-\frac{1}{2}\pi\Big/\sqrt{3}-\frac{3}{2}\ln 3$$

$$H_{2/3}=\frac{3}{2}+\frac{1}{2}\pi/\sqrt{3}-\frac{3}{2}\ln 3$$

$$H_{1/4}=4-\frac{1}{2}\pi-3\ln 2$$

$$H_{3/4}=\frac{4}{3}+\frac{1}{2}\pi-3\ln 2$$

$$H_{1/5}=5-\frac{1}{2}\pi\phi\sqrt{\frac{2+\phi}{5}}-\frac{1}{2}(3-\phi)\ln 5-\left(\phi-\frac{1}{2}\right)\ln(2+\phi)$$

$$H_{2/5}=\frac{5}{2}-\frac{1}{2}\pi/\phi\sqrt{2+\phi}-\frac{1}{2}(2+\phi)\ln 5+\left(\phi-\frac{1}{2}\right)\ln(2+\phi)$$

$$H_{3/5}=\frac{5}{3}+\frac{1}{2}\pi/\phi\sqrt{2+\phi}-\frac{1}{2}(2+\phi)\ln 5+\left(\phi-\frac{1}{2}\right)\ln(2+\phi)$$

$$H_{4/5}=\frac{5}{4}+\frac{1}{2}\pi\phi\sqrt{\frac{2+\phi}{5}}-\frac{1}{2}(3-\phi)\ln 5-\left(\phi-\frac{1}{2}\right)\ln(2+\phi)$$

$$H_{1/6}=6-\frac{1}{2}\pi\sqrt{3}-2\ln 2-\frac{3}{2}\ln 3$$

$$H_{5/6}=\frac{6}{5}+\frac{1}{2}\pi\sqrt{3}-2\ln 2-\frac{3}{2}\ln 3$$

且一般地，当 $0<p<q$ 时（参考习题1.2.9-19）

$$H_{p/q}=\frac{q}{p}-\frac{1}{2}\pi\cot\frac{q}{p}\pi-\ln 2q+2\sum_{1\leqslant n<p/2}\cos\frac{2\pi np}{q}\ln\sin\frac{n}{q}\pi$$

附 录 B

记号索引

在下列公式中，未进一步指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 实值函数

P 指针值的表达式，即，或者是Λ或者是某计算机内的一个地址。

α 符号串

形式符号	意　　义	参考节次
NODE(P)	节点（变量组，这些变量由它们的场名来区分），其地址为P，P≠Λ	2.1
F(P)	其场名为F的NODE(P)中的变量	2.1
CONTENTS(P)	其地址为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或$A[n]$	线性阵列A的第n个元素	1.1
A_{mn}或$A[m, n]$	矩形阵列m行n列处的元素	1.1
$V \leftarrow E$	赋给变量V以表达式E的值	1.1
$U \leftrightarrow V$	交换U和V的值	1.1
P ⇐AVAIL	置指针变量P的值成为一个新节点的地址，如果对于这个新节点已无空间可用则报告内存溢出	2.2.3
AVAIL⇐P	NODE(P)恢复成为可用存储；所有它的场亦不再存在	2.2.3
top(S)	一个非空栈S的顶部节点	2.2.1
$X \Leftarrow S$	从S退栈到X；置$X \leftarrow$top(S)；由非空栈S删去top(S)	2.2.1
$S \Leftarrow X$	X进栈：把由X表示的一个值或一组值插入到栈S的顶部作为一个新的项	2.2.1
$(B \Rightarrow E_1;\ E_2)$	条件表达式：如果B为真表示E_1，如果B为假则是E_2	8.1
δ_{jk}	克罗内克尔δ：$(j = k \Rightarrow 1;\ 0)$	1.2.6

（续）

形式符号	意　　义	参考节次
$\sum_{R(k)} f(k)$	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
$\min_{R(k)} f(k)$	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
$\max_{R(k)} f(k)$	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；$k \bmod j = 0$	1.2.4
U/V	集合差：$\{x \mid x$ 在 U 中但不在 V 中$\}$	1.2.4
$\gcd(j, k)$	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	矩形阵列 A 的转置：$A^T[j, k] = A[k, j]$	1.2.3
x^y	x 的 y 次方，x 为正	1.2.2
α^R	α 的左右反转	
$x^{\underline{k}}$	x 的 k 次方：$\left(k \geqslant 0 \Rightarrow \prod_{0 \leqslant j < k} x,\ 1/x^{-\underline{k}}\right)$	1.2.2
$x^{\overline{k}}$	x 上面 k：$(k \geqslant 0 \Rightarrow x(x+1)\cdots(x+k-1) = \prod_{0 \leqslant j < k}(x+j);\ 1/(x+k)^{-\overline{k}})$	1.2.6
$x^{\underline{k}}$	x 下面 k：$(k \geqslant 0 \Rightarrow x(x-1)\cdots(x-k+1) = \prod_{0 \leqslant j < k}(x-j);\ 1/(x-k)^{-\underline{k}}) = (-1)^k(-x)^{\overline{k}})$	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
$\left[{n \atop m}\right]$	第一类斯特林(Stirling)数：$\sum_{0 \leqslant k_1 \leqslant k_2 \leqslant \cdots \leqslant k_{n-m} \leqslant n} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\left\{{n \atop m}\right\}$	第二类斯特林数：$\sum_{0 \leqslant k_1 \leqslant k_2 \leqslant \cdots \leqslant k_{n-m} < m} k_1 k_2 \cdots k_{n-m}$	1.2.6
$\left\langle{n \atop k}\right\rangle$	欧拉数	5.1.3

（续）

形式符号	意　　义	参考节次
$\{a\|R(a)\}$	使得关系$R(a)$为真的所有a的集合	
$\{a_1,\cdots,a_n\}$	集合或多重集合$\{a_k\|1\leqslant k\leqslant n\}$	
$\{n_1\cdot a_1,\cdots,n_m\cdot a_m\}$	包含a_k的n_k次出现的多重集合	5.1.2
$\|\|M\|\|$	基数：多重集合M中元素的个数	
$\|x\|$	x的绝对值：$(x<0\Rightarrow -x;\ x)$	
$\|\alpha\|$	α的长度	
$\lfloor x\rfloor$	x的底限：最大整数函数：$\max\limits_{k\leqslant x} k$	1.2.4
$\lceil x\rceil$	x的顶限：最小整数函数：$\min\limits_{k\geqslant x} k$	1.2.4
$x\bmod y$	mod函数：$(y=0\Rightarrow x;\ x-y\lfloor x/y\rfloor)$	1.2.4
$x\equiv y(\text{modulo } z)$	同余关系：$x\bmod z=y\bmod z$	1.2.4
$\log_b x$	以b为底的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
$\exp x$	x的指数：e^x	1.2.2
$f'(x)$	f在x处的导数	1.2.9
$f''(x)$	f在x处的二阶导数	1.2.10
$f^{(n)}(x)$	第n阶导数：$(n=0\Rightarrow f(x);g'(x)$其中$g(x)=f^{(n-1)}(x))$	1.2.11.2
$f(x)\Big\|_y^z$	$f(z)-f(y)$	
$H_n^{(x)}$	$1+1/2^x+\cdots+1/n^x=\sum\limits_{1\leqslant k\leqslant n}1/k^x$	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
$\Re(w)$	复数w的实数部分	
$\Im(w)$	复数w的虚数部分	
$\zeta(x)$	黎塔函数：当$x>1$时即$H_\infty^{(x)}$	1.2.7
$\Gamma(x)$	伽玛函数：$\gamma(x,\infty)$；当x为一正整数时即$(x-1)!$	1.2.5
$\gamma(x,y)$	不完全的伽玛函数	1.2.11.3
γ	欧拉常数	1.2.7
e	自然对数的底：$\sum\limits_{k\geqslant 0}1/k!$	1.2.2

（续）

形式符号	意　　义	参考节次
ϵ	空串（长度为0的串）	
∞	无穷：大于任何数；或者一个任意高的键	5
$M \uplus N$	多重集合M和N之和；例如$\{a,a,b\} \uplus \{a,b,c\} = \{a,a,a,b,b,c\}$	4.6.3
Λ	空链接（指针不指向地址）	2.1
ϕ	空集合（没有元素的集合）	
ϕ	黄金比，$\frac{1}{2}(1+\sqrt{5})$	1.2.8
$\varphi(n)$	欧拉的求和函数：$\sum_{\substack{0\leqslant k<n \\ \gcd(k,n)=1}} 1$	1.2.4
$O(f(n))$	当$n\to\infty$时$f(n)$的大O；这是一个量，对于所有大的n，它的绝对值小于某个常数乘上$f(n)$	1.2.11.1
$O(f(x))$	$f(x)$的大O，对于小的x（或者对于在某个确定范围内的x）	1.2.11.1
$(\min x_1, \mathrm{ave}\, x_2, \max x_3, \mathrm{dev}\, x_4)$	一个有极小值x_1，（"预期的"）平均值x_2，极大值x_3，标准离差x_4的随机变量	1.2.10
$\mathrm{mean}(g)$	由母函数g表示的概率分布的均值：$g'(1)$	1.2.10
$\mathrm{var}(g)$	由母函数g表示的概率分布的方差：$g''(1)+g'(1)-g'(1)^2$	1.2.10
P $	在一个二叉树中NODE(P)的在对称次序下的后继者的地址	2.3.1
.	算法、程序或者证明的结束	1.1
⊔	一个空格	1.3.1
rA	MIX的寄存器A（累加器）	1.3.1
rX	MIX的寄存器X（扩充）	1.3.1
rI1, …, rI6	MIX的（变址）寄存器I1, …, I6	1.3.1
rJ	MIX的转移寄存器J	1.3.1
(L:R)	MIX字的部分字段，$0\leqslant L\leqslant R\leqslant 5$	1.3.1
OPADDRESS, 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"	7.3.1
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
$\approx$	近似相等	

名词和姓名中英对照表

如果你在这个索引中未找到它，那就请在整个目录册中仔细地寻找。
——Consumer's Guide, Sears, Roebuck and Co.(1897)

当一个索引项涉及有关的习题时，关于进一步的信息请看该习题的答案。

Abbreviated keys	缩写的键
Abel, Luther Charles	阿贝尔·卢瑟·查尔斯
Abel, Niels, Henrik, binomial formula	阿贝尔·尼尔斯·亨里克二项式公式
limit theorem	阿贝尔极限定理
Abraham, C. T.	亚伯拉罕C. T.
Abramowitz, Milton	阿布拉莫维茨·米尔顿
Addition of polynomials	多项式的加法
Addition to list, *see* Insertion	加到表列上，见插入
Address calculation sorting	地址计算排序
Address table sorting	地址表排序
Adelson-Velskii, Georgii Maksimovich	阿德尔森-维尔斯基，乔治·马克西莫维奇
Aho, Alfred Vaino	阿霍·阿尔弗雷德·维诺
Al-khowârizmî, Abu Ja'far Mohammed ibn Mûsâ	阿尔-科瓦里兹米·阿布·贾法·穆哈默德·穆萨
Алэкшеев Влабимир ЭВГЭНИВИЦ	阿历克谢耶夫·弗拉基米尔·叶甫根尼维奇
ALGOL	ALGOL语言
Algorithms, analysis of, *see* Analysis	算法的分析，见分析字头
comparison of, *see* Comparison, proof of	算法的比较，见比较字头，算法的证明
Allen, Charles Grant Blairfindie	艾伦·查尔斯·格兰特·布莱尔芬迪
Alphabetic code	字母代码
Alternating runs	交错路段
Amble, Ole	安贝尔·奥勒
Amdahl, Gene M.	阿姆达尔·吉恩·M
American Library Association	美国图书馆协会
AMM; American Mathematical Monthly	AMM；美国数学月刊
Amphisbaenic sort	两头蛇排序
Anagrams	变异字
Analysis of algorithms, *see also* complexity analysis	算法的分析，也见复杂性分析
AND (logical and)	AND 运算（逻辑与操作）
André, Antoine Desire	安德烈·安托万·德席里
Anuyogadvarā-sutra	安奴约加德瓦拉-苏特拉
Apollonius Sophista, son of Archibius	阿波洛尼厄斯·索菲斯塔，阿奇比尤斯的儿子
Archimedes	阿基米德
Argument	变元
Arisawa, Makota	艾里萨瓦·马科塔
Arithmetic progression	算术级数
Armerding, George W.	阿默丁·乔治·W
Armstrong, philip Nye	阿姆斯特朗·菲利普·奈
Arora, Sant Ram	阿罗拉·桑特·拉姆
Ashenhurst, Robert Lovett	阿申赫斯特·罗伯特·洛维特
Associative law	结合律
Associative memory	联想存储器

Asymptotic methods	渐近方法
Atia, Rabbi Mordecai	阿蒂亚·拉比·莫迪凯
Attributes, *See* Secondary keys	属性，见辅助键
binary	二进属性
compound	复合属性
Automatic programming	自动程序设计
Average cost, minimization of	平均代价的极小化
AVL trees, *see* Balanced trees	AVL 树，见平衡树
B-trees	B 树
Babbage, Charles	巴贝奇·查尔斯
Baber, Robert Laurence	巴伯·罗伯特·劳伦斯
Babylonian mathematics	巴比伦的数学
Backwards reading, *see* Read-backwards	向后读
Baer, Robert M.	巴伊尔·罗伯特·M
Bailey, Norman Thomas John	贝利·诺曼·托马斯·约翰
Balance factor	平衡因子
Balanced filing	平衡文件
Balanced merging	平衡合并
Balanced radix sort	平衡基数排序
Balanced trees	平衡树
Weight-balanced	加权平衡树
Ball, walter william Rouse	鲍尔·沃尔特·威廉·劳斯
Ballot problem	投票问题
Barnett, John Keith Ross	巴尼特·约翰·基思·罗斯
Barton, David Elliott	巴顿，戴维·埃里奥特
Basic query	基本查询
Batcher, Kenneth Edward	巴切尔·肯尼思·爱德华
sorting method, *see* Merge exchange	巴切尔排序方法，见合并交换
Batching	分批
Baudet Gerard	鲍德特·杰勒德
Bayer, paul Joseph	拜尔·保罗·约瑟夫
Bayer, Rudolf	拜尔·鲁道夫
Bayes, Anthony John	贝斯·安托尼·约翰
Bell, Colin J.	贝尔·科林·J
Bell, David Arthur	贝尔·戴维·阿瑟
Bell, James Richard	贝尔·詹姆斯·理查德
Bellman, Richard Ernest	贝尔曼·理查德·欧内斯特
Bencher, Dennis Leo	本彻·丹尼斯·利奥
Bender, Edward Anton	本德·爱德华·安东
Bennett, Brian Thomas	贝内特·布赖恩·托马斯
Berge, Claude	伯奇·克劳德
Berners-Lee, Conway Maurice	贝纳斯-李，康韦·莫里斯
Bernoulli numbers	贝努里数
Bertrand, Joseph Louis Fransois	伯特朗·约瑟夫·路易·弗朗索伊斯
Best-fit allocation	按“最好的适合”分配
Best Match, see Closest match	最好的匹配，见最接近的匹配
Best possible	最好的
Beta distribution	β（贝塔）分布
Betz, B. K	贝茨·B.K
Beus, H. Lynn	比尤斯·H·林恩
Bhâscara Âchârya	巴斯卡拉、阿查里亚
Bhattachrya, K. N	巴塔彻里亚，K.N

Bienayme, Irénee Jules	比奈米·艾琳·朱尔斯
Bierce, Ambrose	比尔奇·安布罗斯
BINAC	BINC（比纳克）计算机
Binary attributes	二进属性
Binary insertion sort	二叉插入排序
Binary merging	二叉合并
Binary Search	二分查找
uniform	均匀的二分查找
Binary search trees	二分查找树
optimum	最优二分查找树
pessimum	最劣二分查找树
Binary tree; Either empty or a root node and its left and right binary subtrees; see also complete binary tree, Extended binary tree	二叉树；或者空的，或者一个根节点及其左和右子树；也见完备二叉树，扩充二叉树
enumeration of	二叉树的枚举
triply-linked	三重链接的二叉树
Binomial coefficients	二项式系数
Binomial probability distribution	二项式概率分布
Binomial transform	二项式变换
Birthday paradox	生日诗论
Bisection, *see* Binary Search	二分法，见二分查找
Bitonic sort	双调排序
Bleier, Robert. E	布莱尔·罗伯特·E
Block (on tape)	(带上的)块区
Block designs	区组设计
Bloom, Burton H.	布卢姆·伯顿·H
Blum, Manuel	布勒姆·曼纽尔
Bochme McGraw, Elaine M.	贝姆·麦格劳·伊莱恩.M
Boerner, Hermann	博尔纳·赫尔曼
Boolean query	布尔查询
Booth, Andrew Donald	布恩·安德鲁·唐纳德
Boothroyd, John	布恩罗伊德·约翰
Bose, Raj Chandra	博斯·雷·钱德拉
Bottenbruch, Herman	博登布鲁赫·赫尔曼
Bouricius, Willard Gail	布里西尤斯·威拉德·盖尔
Bourne, Charles Percy	伯恩·查尔斯·珀西
Brawn, Barbara S.,	布朗·巴巴拉·S
Brent, Richard Peirce	布伦特·理查德·皮尔斯
Brock, Paul	布洛克·保罗
Brown, William Stanley	布朗·威廉·斯坦利
Brown, Mark Robbin	布朗·马克·罗宾
Brown, william stanley	布朗 威廉·斯坦利
Brute force	暴力，这里指硬算
Bryce, James W.	布赖斯·詹姆斯·W
Bubble sort	气泡排序
Buchholz, Werner	布赫霍尔兹·维纳
Buckets	桶
Buffering	缓冲
Size of buffers	缓冲区的大小
Bulk memory, see Direct-acess Storage	海量存储，见直接存取存储
Burge, William Herbert	伯奇·威廉·赫伯特
Burton, Robert	伯顿，罗伯特

CACM: Communication of the ACM	美国计算机协会通讯
Cancellation law	消去律
Card, edge-notched	卡片，边带槽口的
Card Sorters	卡片排序机
Carlitz, Leonard	卡里茨·伦纳德
Caron, Jacques	卡伦·雅克
Polyphase merge	卡伦多阶段合并
Carroll, Lewis (=Dodgson, Rev. Charles Lutwidge)	卡罗尔·刘易斯（道奇森·查尔斯，勒特威奇牧师）
Carter, William Caswell	卡特·威廉·卡斯韦尔
Cascade merge	级联合并
read-backward	向后读级联合并
rewind overlap	重绕重叠级联合并
Cascade numbers	级联数
Cascading pseudo-radix sort	级联伪基数排序
Catalan numbers	卡塔兰数
Catenated search	连接查找
Cawdrey (=Cawdry), Robert	考德雷伊（考德利）·罗伯特
Cayley, Arthur	凯莱·阿瑟
Census	人口调查，人口普查
Centered insertion, *see* Two-way insertion	中心插入，见两路插入
Césari, yves	塞扎里·伊维斯
Chaining	链接
Chakravarti, Gurugovinda	查克拉瓦蒂·格鲁戈文德
Chandra, Ashok Kumar	钱德拉·艾舒克·库马尔
Chang Shi-kuo	张系国
Channels	通道
Chartres, Bruce Aylwin	查特斯·布鲁斯·艾尔温
Chase, Stephen Martin	蔡斯·斯蒂芬·马丁
Chebyshev, pafnutil L'vovich	契比雪夫·帕夫纳蒂·勒沃维茨
polynomials	契比雪夫多项式
Chow, David Kuo-Kien	周·戴维·国权
Church, Randolph	丘奇·伦道夫
CI: MIX's comparison indicator	CI：MIX的比较指示符
Cliques	集团
Closest match, search for	查找最接近的匹配
Coalescing, lists	接合表
COBOL	COBOL语言
Cocktail Shaker sort	鸡尾混合排序
Coffman, E. G. Jr	小科夫曼·E. G
Coldrick, David Blair	科德里克，戴维·布莱尔
Colin, Andrew J. T.	科林·安德鲁·J. T
Collating, see Merging	整理，见合并
Collating sequence	整理序列
Collision resolution	冲突的解决
Column Sorting, see Distribution Sorting	列的排序，见分布排序
Comp. J.: The computer Journal	计算机杂志
Comparator modules	比较模块
Comparison counting sort	比较计数排序
Comparison of algorithms	算法比较
Comparison of keys	键比较
minimizing	键比较的极小化
multiprecision	多精度的键比较

parallel (i.e. simultaneous)	并行的（即同时的）键比较
searching by	通过键比较进行查找
sorting by	通过键比较进行排序
Comparison trees	比较树
Compiler techniques	编译程序技术
Complement notation	补码记号
Complements, finding all pairs	找出所有互补的对
Complete binary trees: A binary tree with nodes numbered 1 to n, where ⌊k/2⌋ is the father of node *k*	完备的二叉树：节点编号为1到n的一株二叉树，其中⌊k/2⌋为节点k的父亲
Complete P-ary tree	完备的P叉树
Complex partitions	复杂的分划
Complexity analysis of algorithm	算法复杂性分析
Component of a graph	一个图的分量
Compound attributes	复合属性
Compound leaf of a tree	一个树的复合叶
Compression of data	数据的压缩
Compromise merge	折衷合并
Computational complexity, *see* complexity	计算复杂性，见复杂性
Computed entry table, *see* Scatter table	算好的项的表，见散列表
Computer operator skilled	熟练的计算机操作员
Computer Sciences Corporation	计算机科学协会
Comrie, Leslie John	科姆里·莱斯利·约翰
Concatenation	连接
Convex function	凸函数
Cookies	家常小甜饼
Coroutine	联立子程序
Counting Sorting by	通过计数进行排序
Covering	覆盖
Coxeter, Harold Scott Macdonald	考克斯特·哈罗德·斯科特·麦克唐纳
Cramer, Gabriel	克拉默·加布里埃尔
Crane, Clark Allan	克兰·克拉克·阿伦
Creation, Book of	基督教《圣经》创世的书
Criss-Cross merge	交叉合并
Cross-indexing, *see* Secondary key retrieval	交叉索引，见辅助键检索
Cross-reference routine	交叉访问程序
Cube, n-dimensional linearized	线性化的n维立方体
Cundy, Henry Martyn	坎迪·亨利·马丁
Cycle of a permutation	一个排列的轮换
Cylinders	柱面
Daly, Lloyd William	戴利·洛伊德·威廉
Database	数据库
Data structure, Choice of	数据结构的选择
Dauer, Francis Watanabe	多尔·弗朗西斯·沃塔纳比
David, Florence Nightingale	戴维·弗洛伦斯·奈廷格尔
Davidson, Leon	戴维森·利昂
Davies, Donald Watts	戴维斯·唐纳德·瓦茨
de Balbine, Guy	德·巴尔宾·盖伊
de Bruijn, Nicolaas Gavert	德·布鲁因·尼古拉斯·戈维特
de La Briandais, Rene	德·拉·布朝恩戴斯，雷内

de Peyster, J·A 德·佩斯特·J. A
de Staël, Madame, see Staël-Holstein 德·斯塔耶尔·马达默，见斯塔耶尔-霍尔斯坦
Deadlines 截止时间
Deadlock 死锁
Dedekind, Richard 狄德金·理查德
sums 狄德金和数
Degenerate tree 退化树
Degenerative addresses 退化地址
Degree path length 叉数路径长度
Deletion; Removing an item 删去；撤销一个项
from B-tree 从B树删去
from balanced tree 从平衡树删去
from binary search tree 从二分查找树删去
from digital search tree 从数字查找树删去
from hash table 从杂凑表删去
from leftist tree 从最左边的树删去
from trie 从检索结构删去
Demons 精灵
Demuth, Howard 德穆思·霍华德
Dent, Warren Thomas 登特·沃伦·托马斯
Determinant 行列式
Diagram of partial order 偏序的图式
Digital searching 数字查找
Digital sorting 数字排序
Digital tree search 数字树查找
Diminishing increment sort, see Shellsort 减少增量排序，见谢尔排序
Dinsmore, Robert John 丁斯莫尔·罗伯特·约翰
Direct-access storage 直接存取存储
Directed graphs 有向图
Discrete entropy 离散的熵
Discrete logarithms 离散对数
Discriminant 判别式
Disks, *see* Direct-access storage 盘，见直接存取存储
Disorder, measures of 混乱的程度
Distribution counting sort 分布计数排序
Distribution function, *see* probability 分布函数，见概率
Distribution patterns 分布型式
Distribution sorting 分布排序
dual to merging 对偶于合并的分布排序
Dixon John Douglas 狄克逊·约翰·道格拉斯
Dodd, Marisue 多德·马里休
Dodgson, Rev. Charles Lutwidge, *see* Carroll 道奇森. 查尔斯·勒特威奇牧师，见卡罗尔
Doren, David G. 多琳. 戴维·G
Double-entry bookkeeping 双项簿记
Double hashing 双重杂凑
Doubly exponential sequence 双重指数序列
Douglas, Alexander Shafto 道格拉斯. 亚历山大·萨菲托
Drake, Paul 德雷克. 保罗
Drums, *see* Direct-access storage 鼓，见直接存取存储
Drysdale, Robert Lewis (Scot) Ⅰ 德赖斯代尔·罗伯特·刘易斯(斯科特)Ⅰ
Dudeney, Henry Ernest 杜德尼·亨利·欧内斯特
Dugundji, James 达冈德吉·詹姆斯
Dull, Brutus Cyclops 达尔·布鲁特斯·赛斯洛普斯

Dumey, Arnold Isaas	达米·阿诺德·伊萨克
Dwyer, Barry	德怀尔·巴里
Dummy runs	虚拟路段
Dynamic storage allocation	动态存储分配
Dynamic table searching, *see* symbol table algorithms	动态表查找，见符号表算法
Eckert, John Presper	埃克特·约翰·普雷斯佩
Edge-notched cards	边带槽口的卡片
Edighoffer, Judy Lynn Harkness	埃迪霍弗，朱迪·林·哈克尼斯
Editing routines	编辑子程序
Edmund, Norman Wilson	埃德蒙·诺曼·威尔逊
EDVAC	埃德瓦克计算机
Efficiency of a graph	一个图的效率
Eichelberger, Edward B.	艾克尔伯格·爱德华·B
Elcock, Edward Wray	埃尔科克·爱德华·雷
Elementary symmetric functions	初等对称函数
Elevators	电梯
Elias, Peter	伊莱亚斯·彼得
Ellery, Robert Lewis	埃勒里·罗伯特·刘易斯
Empirical data	经验数据
Encoding Surnames	对姓氏编码
English Language	英语
most common words of	英语中最常用的字
Entropy discrete	离散的熵
Enumeration of binary trees	二叉树的枚举
Enumeration of permutations	排列的枚举
Enumeration sorting	枚举排序
Equality of sets testing	判断集合是否相等
approrimate	集合的近似相等
Eratosthenes	伊拉托斯森斯
Erdélyi, Arthur	厄德尔莱·阿瑟
Erdös Paul	厄尔多斯·保罗
Эрдош Андрей Петрович	厄尔索夫·安德烈·彼得罗维奇
Espelid, Terje Oskar	埃斯皮里得，特里·奥斯卡
Euclid's algorithm	欧几里得算法
Euler, Leonhard	欧拉，伦哈德
numbers (secant numbers)	欧拉数(正割数)
summation formula	欧拉求和公式
Eulerian numbers	欧拉数
Eve, James	伊夫，詹姆斯
Even permutation	偶排列
Exchange Selection Sort, *see* Bubble Sort	交换选择排序，见气泡排序
Exchange sorting	交换排序
optimum	最优交换排序
Exclusive or	异或
Exercises, notes on	关于习题的说明
Exponential integral	幂积分
Extended binary tree: Either a single "external" node, or an "internal" root node plus its left and right extended binary subtrees	扩充的二叉树：或者是单个"外部"节点，或者是一个"内部"根节点加上它的左和右扩充二叉子树
External path length: sum of the level numbers of all external nodes, *see* Path length	外部路径长度：所有外部节点的级别数之和，见路径长度

External Searching 外部查找
External sorting 外部排序
  summary 外部排序综述

Factorials 阶乘
Factorization of permutations 排列的因子分解
Fallacies in probability 概率的谬误
Fast Fourier transform 快速傅里叶变换
Feature Cards 特性卡片
Feldman, Jerome Arthur 费尔德曼·杰罗姆·阿瑟
Feller, William 费勒·威廉
Feindler, Robert 范德勒·罗伯特
Feit, Walter 菲特·沃尔特
Ferguson, David Elton 弗格森·戴维·埃尔顿
Feurzeig wallace 福伊尔齐格·华莱士
Fibonacci, Leonardo 斐波那契·伦纳德
Fibonacci distributions 斐波那契分布
Fibonacci hashing 斐波那契杂凑
Fibonacci number system 斐波那契数系统
Fibonacci numbers 斐波那契数
  generalized, *see* Cascade number 广义的斐波那契数，见级联数
Fibonacci search 斐波那契查找
Fibonacci trees 斐波那契树
Fibonaccian Search 斐波那契查找
Fifo (first-in-first-out) tree Fifo（先进先出）树
File 文件
  partitioning 划分文件
  self-organizing 自组织的文件
Finite field 有限域
Finkel Raphael Ari 芬克尔·拉菲尔·阿里
First-fit allocation 按“头一个适合”进行分配
Fishburn, John Scot 菲什伯恩·约翰·斯科特
Fixed point of permutation 排列的不动点
Fletcher, William 弗莱彻·威廉
Floating buffers 浮动缓冲区
Floating point arithmetic 浮点算术
Flores, Ivan 弗洛里斯·伊凡
Floyd, Robert W 弗洛伊德·罗伯特·W
Foata, Dominique Cyprien 福塔·多米尼克·西普里安
Ford, Donald Floyd 福特·唐纳德·弗洛伊德
Ford, Lester Randolph 福特·莱斯特·伦道夫
Forecasting 预报
Forest 森林
FORTRAN FORTRAN语言
Foster, Caxton Croxford 福斯特·卡克斯顿·克劳福得
Fourier, Jean Baptiste Joseph, transform 傅里叶·琼·巴普蒂斯特·约瑟夫·傅里叶变换
Frame, James Sutherland 弗雷姆·詹姆斯·萨瑟兰
Frank, Robert Morris 弗兰克·罗伯特·莫里斯
Franklin, Fabian 富兰克林·费比恩
Frazer, William Donald 弗雷泽·威廉·唐纳德
Fredkin, Edward 弗雷德金·爱德华
Fredman, Michael Lawrence 弗雷德曼·迈克尔·劳伦斯
Free group 自由群

Frequency of access	访问频率
Friend, Edward Harry	弗兰德·爱德华·哈里
Frobenius, Ferdinand Georg	弗罗比尼厄斯·费迪南德·乔治
Front and rear compression	前和后的压缩
Gaines, Helen Fouche	盖恩斯·海伦·福奇
Gale, David	盖尔·戴维
Galen, claudius	盖伦·克劳迪亚斯
Galli, E. J.	加利·E. J.
Gamma function	伽玛（γ）函数
Gandz, Solomon	甘兹·所罗门
Gaps between prime numbers	质数之间的间隔
Gaps on tape	带上的间隔
Gardner, Erle Stanley	加德纳·厄尔·斯坦利
Gardner, Martin	加德纳·马丁
Gassner, Betty Jane	加斯纳·贝蒂·简
Gaudette, C. H	高德特·C. H
Gauss, Karl (=Carl) Friedrich	高斯·卡尔·弗里德里克
gcd; Greatest common divisor	gcd; 最大公因子
Generating function	生成（母）数
Genoa, Giovanni di	吉诺亚·乔瓦尼·迪
Ghosh, Sakti Pada	戈斯·萨克蒂·帕达
Gibson, Kim Dean	吉布森·金·迪安
Gilbert, Edgar Nelson	吉尔伯特·埃德加·纳尔逊
Gilstad, Russell L.	吉尔斯塔德·拉塞尔·L
Gleason, Andrew Mattei	格里森·安德鲁·马太
Goetz, Martin A.	戈茨·马丁·A
Golden ratio	黄金比
Goldenberg, Daniel	戈登堡·丹尼尔
Goldstein, Larry Joel	戈尔茨坦·拉里·乔尔
Gonnet Haas, Gaston Henry	康内特·哈斯 加斯顿·亨利
Goodwin, D. T.	古德温·D. T
Gotlieb, Calvin Carl	戈特利布·卡尔文·卡尔
Goto, Eiichi	戈托·埃里茨
Graham, Ronald Lewis	格雷厄姆·罗纳德·刘易斯印葛立恒
Grasselli, Antonio	格拉塞利·安东尼奥
Gray, Harry Joshua	格雷·哈里·乔舒亚
Greatest common divisor	最大公因子
Green, Milton Webster	格林·米尔顿·韦伯斯特
Greniewski, Marek	格里纽斯基·马雷克
Gross, Oliver Alfred	格罗斯·奥利弗·艾尔弗雷德
Group, free	自由群
Growth ratio	生长率
Gruenberger, Fred Joseph	格伦伯格·弗雷德·约瑟夫
Guibas, Leonidas	吉巴斯·利奥尼达斯
Gunji, Takao	冈吉·塔考
Gustafson, Richard Alexander	古斯塔夫森·理查德·亚历山大
Gustavson, Frances Goertzel	古斯塔夫森·弗朗西斯·戈茨尔
Gwehenberger, Gernot	格温贝格·杰尔诺特
Hadian, Abdolloh	哈迪安·阿布多拉
Hall, Marshall, Jr	小霍尔·马歇尔
Hall, philip	霍尔·菲利普

Halperin, John Harris	霍尔珀林·约翰·哈里斯
Halpern, Mark Irwin	哈尔彭·马克·欧文
Hamilton, Douglas	汉密尔顿·道格拉斯
Hanan, Maurice	哈南·莫里斯
Hardy, Godfrey Harold	哈迪·戈弗雷·哈罗德
Hardy, Norman	哈迪·诺曼
Harmonic numbers	调和数
Harper, Lawrence Hueston	哈珀·劳伦斯·休斯顿
Harrison, Malcolm Charles	哈里森·马尔科姆·查尔斯
Harrison, Michael Alexander	哈里森·迈克尔·亚历山大
Hash functions	杂凑函数
combinatorial	组合杂凑函数
Hash sequence	杂凑序列
Hashing	杂凑
HB[k] trees, Height-balanced trees, *see* Balanced trees	HB[k]树，高度平衡树，见平衡树
Heaps	堆
Heapsort	堆排序
Height of tree	树的高度
Heilbronn, Hans	海尔布伦·汉斯
Heising, William Paul	海辛·威廉·保罗
Heller, Robert Amdrew	赫勒·罗伯特·安德鲁
Hellerman, Herbert	赫勒曼·赫伯特
Hennie, Fredrick Clair	亨尼·弗里德里克·克莱尔
Hibbard, Thomas Nathaniel	希巴德·托马斯·纳撒尼尔
Hilbert, David	希尔伯特·戴维
Hildebrandt, Paul	希尔德布兰特·保罗
Hindenburg, Karl Friedrich	欣登伯格·卡尔·弗里德里克
Hinton, Charles Howard	欣顿·查尔斯·霍华德
Hirschberg, Daniel Syna Moses	赫西伯格·丹尼尔·西纳·摩西
Hoare, Charles Antony Richard	霍尔·查尔斯·安托尼·理查德
Hoey, Daniel J.	霍伊·丹尼尔·J
Holberton, Frances Elizabeth Snyder	霍尔伯顿·弗朗西斯·伊丽莎白·斯奈德
Hollerith, Herman	霍勒里特·赫尔曼
Holt Hopfenburg, Anatol Wolf	霍尔特·霍普芬伯格，阿纳托尔·沃尔夫
Homer	霍默
Homogeneous Comparisons	齐性比较
Hooker, William Weston	胡克·威廉·韦斯顿
Hooks	胡克斯
Hopcroft, John Edward	霍普克罗夫特·约翰·爱德华
Hopgood, Frank Robert Albert	霍普古德·弗兰克·罗伯特·艾伯特
h-ordered, *see* 2-ordered	h有序的，见2有序的
Hoshi, Mamoru	星守
Hosken, J.C	霍斯金·J.C.
h-sorting	h排序
Hu, Te Chiang	胡德强
Hu-Tucker algorithm	胡-塔克算法
Hubbard, George Underwood	哈巴德·乔治·安德伍德
Huffman, David Albert	赫夫曼·戴维·艾伯特
Hunt, Douglas H.	亨特·道格拉斯·H
Hurwitz, Henry	赫维茨·亨利
Hwang, Frank Kwangming	黄光明
Hyafil, Laurent Daniel	海亚菲尔·劳伦特·丹尼尔

IBM Corporation	IBM（国际商用机器）公司
IBM 701	IBM 701计算机
IBM 705	IBM 705计算机
Iff; If and only if	当且仅当
Inakibit-Anu of Urek	艾娜基比特-安奴
Inclusion-exclusion principle	包含-排除原理，或取舍原理
Inclusive query	包含查询
Incomplete gamma function	不完全的伽玛函数
Index for file partioning	用于文件划分的索引
Index modulo p	模P指数
Index of a permutation	一个排列的索引
Index-sequential file organization	按索引顺序的文件组织
Index to this book	本书索引
Infinity	无穷性，无限性
Information retrieval	情报检索
Information theory	信息论
Inner loop, *see* Main loop	内循环，见主循环
Insertion; Add a new item	插入；加上一个新的项
into B-tree	插入B树
into balanced tree	插入平衡树
into binary search tree	插入二分查找树
into digital search tree	插入数字查找树
into hash tables, *see* collision resolution	插入杂凑表，见冲突的解决
into leftist tree	插入左倾树
into trie	插入检索结构
Insertion sorting	插入排序
Interblock gaps	块间间隔
Intercalation product of permutations	排列的插入积
Interchanging blocks of data	交换数据块区
Internal (branch) node of binary tree, *see* Extended binary tree	二叉树的内部（分枝）节点，见扩充的二叉树
Internal path length; sum of the level numbers of all internal nodes, *see* path length	内部路径长度，所有内部节点的级别数之和，见路径长度
Internal searching	内部查找
summary	内部查找综述
Internal Sorting	内部排序
summary	内部排序综述
Interpolation Search	内插查找
Interpretive routine	解析子程序
Interval exchange sort	区间交换排序
Inverse of a permutation	一个排列的逆
Inversions of a permutation	一个排列的反序
table of	一个排列的反序表
Inverted files	反序文件
Involutions	卷积
Isaac, Earl J.	伊萨克·厄尔·J
Isbitz, Harold	伊斯比兹·哈罗德
Isidorus, St. of Seville	塞维利亚的圣伊西多拉斯
Isomorphisms	同构
Iverson, Kenneth Eugene	艾弗森·肯尼特·尤金
JACM, *Journal of the ACM*	美国计算机协会杂志
Jacobi, Carl Gustav Jacob	雅各比·卡尔·吉斯塔夫·雅各布

Jacobsen, William H. Jr	小雅各布森·威廉·H
JAE(jump A even)	JAE(A为偶则转移) 指令
Jainism	耆那教
JAO(jump A odd)	JAO(A为奇则转移) 指令
Johnsen, Robert Lawrence	约翰逊，罗伯特·劳伦斯
Johnsen, Thorstein Lunde	约翰逊·索尔斯坦·伦德
Johnson, Donald W.	约翰逊·唐纳德·W
Johnson, Lyle Robert	约翰逊·莱尔·罗伯特
Johnson, Selmer Martin	约翰逊·塞尔默·马丁
Johnson, Stephen Curtis	约翰逊·斯蒂芬·柯蒂斯
Josephus, Flavius Problem	约瑟夫斯·弗雷维尼斯问题
Jump operator of MIX	MIX的转移操作符
Kaman, Charles Henry	卡曼·查尔斯·亨利
Kant, Immanuel	坎特·伊曼纽尔
Karlton, philip Lewis	卡尔通·菲利普·刘易斯
Karp, Richard Manning	卡普·理查德·曼宁
Kaufman, Marc Thomas	考夫曼·马克·托马斯
Kautz, William Hall	考茨·威廉·霍尔
Kepoв, C.B.	克洛夫·C.B
Key	键
Key sorting	键排序
Key transformation, *see* Hashing	键转换，见杂凑
Khizder, Leonid Abramovich	凯泽德尔·利奥尼德·阿布拉莫维奇
Kipling, Joseph Rudyard	基普林·约瑟夫·拉迪亚德
Kircher, Athanasius	柯切尔·阿塔纳修斯
Kirchhoff, Gustav Robert, firstlaw	柯希霍夫·古斯塔夫·罗伯特第一定律
Kirkman, Rev. Thomas Penyngton	柯克曼·托马斯·彭宁顿
Kirkpatrick, David Galer	柯克帕特里克·戴维·盖勒
Kislitsyn, Sergei Sergeyevich	基斯里特辛·塞盖·塞盖耶维奇
Klarner, David Anthony	克拉尔纳·戴维·安托尼
Kleitman, Daniel J.	克莱特曼·丹尼尔·J
Klerer, Melvin	克莱尔·梅尔文
Knock-out tournament	淘汰赛
Knott, Gary Don	克诺特·加里·唐
Knuth, Donald Ervin	克努特·唐纳德·欧文
Koch, Gary Grove	科克·加里·格罗夫
Konheim, Alan Gustave	康海姆·艾伦·古斯塔夫
Korn, Granino Arthur	科恩·格拉尼诺·阿瑟
Kronmal, Richard Aaron	克朗马尔·理查德·艾伦
Kronrod, Mikhail Aleksandrovich	克朗罗德·米克海尔·阿列克山德罗维奇
Krutar, Rudolph Allen	克鲁塔·鲁道夫·艾伦
Kwan, Lun Cheung	关伦丛
KWIC index	上下文中的键索引
Ladd, George Trumbull	莱德·乔治·特朗布尔
Lagrange, Joseph Louis, Comte, inversion formula	拉格朗日·约瑟夫·路易斯·科姆特，反演公式
Lamb, Sidney	拉姆·西德尼
Lambert, Johann Heinrich	兰伯特·约翰·海因里希
Lampson, Butler Wright	兰普森·巴特勒·赖特
Landauer, Walter Isfried	兰多尔·沃尔特·艾斯弗里德
Lander, Leon Joseph	兰德·利昂·约瑟夫
Landis, Evgenii Mikhailovich	兰迪斯·尤金尼·米克海伊罗维奇

Largest-in-first out, *see* priority queue	最大者先出，见优先队
Latency time	等待时间
Latin squares	拉丁方
Lattice	格
Lattice paths	格点通路
Lazarus, Roger Ben	拉扎勒斯·罗杰·本
Leaf	叶
Least-recently-used page replacement	最近最少使用的页替换
Lee, Tsai-hwa	李再华
Lefkowitz, David	莱夫科维茨·戴维
Left-to-right maximum or minimum	自左至右极大或极小
Right-to-left	自右至左极大或极小
Leftist trees	左倾树
Lehman, Alfred Baker	莱曼·艾尔弗雷德·贝克
Lehmer, Derrich Henry	莱默·德里克·亨利
Leibholz, Stephen W	莱博霍尔兹·斯蒂芬·W
Level of a tree node: The distance to the root	一个树节点的级：到根的距离
Leveque, William Judan	勒维克·威廉·朱丹
Lexicographic order	字典编辑顺序
Liang, Franklin Mark	梁·富兰克林·马克
Library Card Sorting	图书馆卡片排序
Library tape allocation	库带的分配
Lifo(last-in-first-out) tree	Lifo（后进先出）树
Lin, Andrew Damon	林·安德鲁·达蒙
Lin Shen	林身
Linear algorithm for medium	求中值的线性算法
Linear algorithms(average time)for sorting	排序的线性算法（平均时间）
Linear list representation	线性表表示
Linear order	线性次序
Linear probing	线性探查
Linear quotient method, *see* Double hashing	线性商的方法，见双重杂凑
Ling, Huei	魏玲
Linked allocation	链接分配
Linn, John Charles	林·约翰·查尔斯
Lissajous, Jules Antoine	利萨卓斯·朱尔斯·安托尼
List insertion sort	表插入排序
List merge sort	表合并排序
List sorting, *see* List insertion, List merge, Multiple list insertion, Radix list sort, Tree insertion sort	表排序，见表插入，表合并，多重表插入，基数表排序，树插入排序
Littlewood, Dudley Ernest	利特尔伍德·达德利·欧内斯特
Littlewood, John Edensor	利特尔伍德·约翰·伊登索
Litwin, S.	利特文·S
Livius, Titus	利维亚斯·泰特斯
Lloyd, Stuort Phinney	洛伊德·斯图尔特·芬尼
Load factor	负载因子
Load point	装入点
Logan, Benjamin Franklin	洛根·本杰明·富兰克林
Logarithmic search, *see* Binary search	对数查找，见二分查找
Logarithms discrete	离散对数
Logg, George Edward	洛格·乔治·爱德华
Logical tape unit numbers	逻辑带设备号
Long runs	长路段

Loop optimization	循环优化
Lowe, Thomas C.	洛·托马斯·C
Лозинский, Эдигер Лзопиб Соломоновиц	洛津斯基·埃利格尔利奥尼德·所罗莫诺维奇
LSD: Least significant digit	LSD：最低位有效数字
Luhn, Hans Peter	卢恩·汉斯·彼得
tukasiewicz, Jan	卢卡谢维茨·简
Lum, Vincent Yu-sun	卢姆·文森特·尤-森
Lunnon, William Frederick	伦农·威廉·弗雷德里克
Lynch, William Charles	林奇·威廉·查尔斯
Machiavelli, Niccolò di Bernardo	麦希亚维利·尼科罗·迪·伯纳多
MacLaren, Malcolm Donald	麦克拉伦·马尔科姆·唐纳德
MacLeod, I. D. G	麦克劳德·I. D. G
MacMahon, Maj. Percy Alexander	麦克马洪·珀西·亚历山大少校
Macro assembler	宏汇编程序
Magnetic tape	磁带
Mahon, Maurice Hartland (=Magenta)	马洪·莫里斯·哈特兰（=马根塔）
Main loop: Part of a program whose instructions are performed much more often than the neighboring parts, *see* loop optimization	主循环，一个程序的这样一部分，其指令较相邻部分更经常地被执行，见循环优化
Mallach, Efrem Gershon	马拉兹·埃弗雷姆·格尔雄
Mallows, Colin Lingwood	马洛斯·科林·林伍德
Maly, Kurt	马里·库特
Maniac I	曼尼亚克I计算机
Mankin, Efrem S.	曼金·埃弗雷姆·S
Mann, Henry Berthold	曼·亨利·伯索德
Марков, Андреи Андреевиц Process	马尔科夫·安德烈·安德烈耶维奇，马尔科夫过程
Martin, Thomas Hughes	马丁·托马斯·休斯
Mason, Perry	梅森·佩里
Mathsort, *see* Distribution Counting	数学排序，见分布计数
Match, search for closest	查找最接近的匹配
Matrix transposition	矩阵转置
Matula, David William	马图拉·戴维·威廉
Mauchly, John William	莫茨利·约翰·威廉
Maximum, finding	找极大值
McAllester, Robert Linné	麦卡勒斯特·罗伯特·林尼
McAndrew, M. H.	麦卡安德鲁·M. H.
McCabe, John	麦凯布·约翰
McCall's Cookbook	麦考尔的食谱
McCarthy, John	麦卡锡·约翰
McCracken, Daniel Delbert	麦克拉肯·丹尼尔·戴尔伯特
McCreight, Edward Meyers	麦克赖特·爱德华·迈耶斯
McIlroy, Malcolm Douglas	麦基尔罗伊·马尔科姆·道格拉斯
McKellar, Archie Charles	麦凯勒·阿尔奇·查尔斯
McKenna, James	麦克纳·詹姆斯
McNamee, Carole Mattern	麦克纳米·卡罗里·马特恩
McNutt, Bruce	麦克纳特·布鲁斯
Measures of disorder	混乱的量度
Median	中间的
Median-of-three quicksort	三者取中的快速排序
Meister, Bernd	梅斯特·伯恩德
Merge exchange sort	合并交换排序
Merge insertion sort	合并插入排序

Merge patterns, *see* Balanced merge, Cascade merge, Oscillating sort, Polyphase Merge	合并型式，见平衡的合并，级联合并，交替排序，多阶段合并
dual to distribution patterns	对偶于分布型式的合并型式
for direct-access storage	直接存取存储的合并型式
optimum	最优的合并型式
summary	合并型式综述
tree representation of	合并型式的树表示
vector representation of	合并型式的向量表示
Merge sorting, *see* List merge Natural merge, Straight merge	合并排序，见表合并，自然合并，直接合并
external, *see* merge patterns	外部合并排序，见合并型式
internal	内部合并排序
k-way	k路合并排序
minimum comparisons	极小比较的合并排序
networks for	合并排序网络
Merging priority queues	合并优先队
Meyer, Curt	迈耶·柯特
Middle-square method	平方取中方法
Miles, Ernest Percy, Jr	小迈尔斯·欧内斯特·珀西
Minimization of average Cost	平均代价的极小化
Minimum-comparison algorithms	极少比较算法
merging	合并的极少比较算法
networks	网络的极少比较算法
searching	查找的极少比较算法
selection	选择的极少比较算法
sorting	排序的极少比较算法
Minimum memory	极小内存
Minimum time algorithms	极小时间的算法
Minker, Jack	明克尔·杰克
Misspelled names	拼错的名字
MIX Computer: Hypothetical machine defined in section 1.3	MIX计算机：在1.3节中定义的假想机器
MIXAL: The MIX assembly Language	MIXAL：MIX汇编语言
MIXT tape units	MIXT 带设备
MIXTEC disks and drums	MIXTEC盘和鼓
Miyakawa, Masahiro	宫川正弘
Möbius, August Ferdinand, function	莫比乌斯·奥古斯特·费迪南德，莫比乌斯函数
Modification, Program Self-	程序的自修改
Monotonic Subsequences	单调子序列
Mooers, Calvin Northrup	穆尔斯·卡尔文·诺思拉普
Moore, Edward Forrest	穆尔·爱德华·福雷斯特
Morris, Robert	莫里斯·罗伯特
Morrison, Donald Ross	莫里森·唐纳德·罗斯
Morse, Samuel Finley Breese Code	莫尔斯·塞缪尔·芬利·布鲁斯代码
Mortenson, John Albert	莫顿森·约翰·阿尔伯特
Moser, Leo	莫泽·利奥
Motzkin, Theodor Samuel	莫茨金·西奥多·塞缪尔
MOVE	MOVE（传送）指令
Muir, Thomas	米尔·托马斯
Multihead bubble sort	多头气泡排序
Multilist system	多表系统
Multinomial coefficients	多项式系数
Multiple attribute retrieval, *see* secondary keys	多属性的检索，见辅键

Multiple list insertion sort	多表插入排序
Multiplication of polynomials	多项式的乘法
Multiprecision comparison	多精度的比较
Multireel files	多卷文件
Multiset; Analogous to a set, but elements may appear more than once	多重集合：类似于一个集合，但是元素可以出现一次以上
Multiway trees	多路树
Muntz, Richard	芒茨·理查德
Munro, James Ian	芒罗·詹姆斯·伊恩
Muroga, Saburo	默罗加·萨伯罗
Nagler, Harry	纳格勒·哈里
Natural merge sort	自然合并排序
Natural selection	自然选择
Nelson, Raymond John	纳尔逊·雷蒙德·约翰
Netto, Otto Erwin Johannes Eugen	内特托·奥托·欧文·约翰尼斯·尤金
Networks, for merging	用于合并的网络
for permutation	用于排列的网络
for selection	用于选择的网络
for sorting	用于排序的网络
minimum delay	极小延迟的网络
Newcomb, Simon	纽科姆·西蒙
Newell, Alen	纽厄尔·阿伦
Newell, Allen	纽厄尔·艾伦
Newman, Donald Joseph	纽曼·唐纳德·约瑟夫
Nielsen, Jakob	尼尔森·雅各布
Nievergelt, Jurg	尼韦格尔特·朱尔格
Nikitin, Andrei Ivanovich	尼基廷·安德烈·伊凡诺维奇
Nitty-gritty	事情的真相，本质
Noshita, Kohei	野下·浩平
Norman, Robert Zane	诺曼·罗伯特·赞恩
Number-Crunching Computers	吞噬数字的计算机
Oblivious algorithm	漠视算法
O'Connor, Daniel J.	奥康纳尔·丹尼尔·J.
Octahedron, truncated	截八面体
Odd-even merge	奇-偶合并
Odd-even transposition Sort	奇-偶转置排序
Odell, Margaret K.	奥德尔·玛格丽特·K
Oderfeld, Jan	奥德尔菲尔德·简
Olson, Charles	奥尔森·查尔斯
On-line merge sort	联机合并排序
One-tape Sorting	单带排序
Ones' Complement notation	一的补码记号
Open addressing	开式寻址
Operating systems	操作系统
Optimization of program, *see* Loop optimization	程序的优化，见循环优化
Optimum exchange sorting	最优交换排序
Optimum open addressing	最优开式寻址
Optimum permutations	最优排列
Optimum polyphase merge	最优多阶段合并
Optimum Sorting	最优排序
Optimum trees, for merging	用于合并的最优树

for searching	用于查找的最优树
Oracles, *see* Adversaries	神谕，见对手
Order relation	次序关系
Order Statistics, *see* Selection of *t*th largest	次序统计，见第 *t* 个最大的选择
Ordered table, Searching in	有序表中的查找
Orosz, Gábor	奥罗斯泽·加博尔
OR (logical or)	OR（逻辑或）
Oscillating sort	交替排序
Overflow, arithmetic	算术溢出
in B-tree	B树中的溢出
in scatter table	散列表中的溢出
Own Coding	特种编码
P-operator Sort (Bose-Nelson method)	P-操作符排序（博斯-纳尔逊方法）
Packed data	压缩数据
Paging	分页
Painter, James Allan	佩因特·詹姆斯·阿伦
Palermo, Frank Pantaleone	佩勒莫·弗朗克·潘塔利昂
Papernov, A. A	佩珀诺夫·A. A
Parallel (simultaneous) computation	并行（同时）计算
Parker, Ernest Tilden	帕克·欧内斯特·蒂尔登
Parkin, Thomas Randall	帕金·托马斯·兰德尔
Parking problem	停车问题
Partial fraction	部分分式
Partial ordering	偏序
Partition exchange sort, *see* quicksort	分划交换排序，见快速排序
Partitioning a file	划分一个文件
Partitions of a number	一个数的分划
Pass; Part of the execution of an algorithm, traversing all the data once	一次扫描，或一趟；在执行一个算法的过程中遍历一次所有的数据
Patents	专利
Path length of tree	树的路径长度
Patricia	一种特殊的树结构
Patt, Yale Nance	帕特·耶尔·南斯
Patterson, George William	帕特森·乔治·威廉
Pentagonal number	五边形数
Perfect distribution, *see* Fibonacci distribution	完全的分布，见斐波那契分布
Perfect Shuffle	完全的洗牌
Permutation network	排列网络
Permutations	排列
cycle of	排列的轮换
enumeration of	排列的枚举
even	偶排列
factorization of	排列的因子分解
fixed points of	排列的不动点
index of	排列的下标
intercalation of	排列的插入
inverse of	排列的逆
Inversions of, *see* Inversions	排列的反序，见反序
multiset	多重集合的排列
optimum	最优排列
readings of	排列的读入
runs of	排列的路段

Peter, Laurence Johnston, principle 彼得·劳伦斯·约翰斯通原理
Peters, Johann(=Jean)Theodor 彼得斯·约翰（=琼）西奥多
Peterson, James Lyle 彼得森·詹姆斯·莱尔
Peterson, William Wesley 彼得森·威廉·韦斯利
Philco 2000 菲尔科2000计算机
Picard, Claude Francois 皮卡德·克劳德·弗朗索伊斯
Ping, czen 成平
Pinzka, Charles Frederick 平兹卡·查尔斯·弗雷德里克
Pipeline computer 流水线计算机
PL/1 PL/1语言
Playing cards 扑克牌
Pocket sort, *see* Distribution sorting 桶排序，见分布排序
Pohl, Ira Sheldon 波尔·艾拉·谢尔登
Poisson, Siméon Denis, distribution 泊松·西米恩·丹尼斯分布
Polist prefix notation 波兰前缀记号
Pollard, John Michael 波拉德·约翰·迈克尔
Pólya, György(=George) 波利亚·乔治
Polygon, regular 正规的多边形
Polynomial arithmetic 多项式算术
Polynomial hashing 多项式杂凑
Polyphase merge sorting 多阶段合并排序
  Caron variation 卡伦形式的多阶段合并排序
  optinum 多阶段优化
  read-backward 多阶段向后读
  tape-splitting 多阶段带的分开
Polyphase radix sort 多阶段基数排序
Post-office tree 邮局树
Posting, *see* Insertion 置入，见插入
Power of merge 合并的能力
Powers, James 鲍尔斯·詹姆斯
Pr: Probability Pr：概率
Pratt, Richard D. 普拉特·理查德·D
Pratt, Vaughan Ronald 普拉特·沃恩·罗纳德
  Sorting method 普拉特排序方法
Prediction 预言
Prefix search, *see* Trie Search 前缀查找，见检索结构查找
Preorder merge 前序合并
Prestet, Jean 普雷斯提特·琼
Prime numbers 质数
Primitive Sorting network 本原排序网络
Pring, Edward John 普林·爱德华·约翰
Priority dequeues 优先双队
Priority queues 优先队
Probability distributions 概率分布
Proof of algorithms 算法的证明
Propagation sort, *see* Bubble sort 传播排序，见气泡排序
Prusker, Francis 普鲁斯克尔·弗朗西斯
Prywes, Noah Shmarya 普赖韦斯·诺亚·施马赖亚

q-nomial coefficients q项式的系数
Quad trees 四叉树
Quadratic selection 二次选择
Queries 查询

Questionnaires	询问
Queues	队
Quicksort	快速排序
Radix exchange sort	基数交换排序
Radix insertion sort	基数插入排序
Radix list sort	基数表排序
Radix sorting, *see* distribution sorting	基数排序，见分布排序
Radke, Charles E.	拉德克·查尔斯·E
Railway switching	铁路转辙器
Random data for sorting	用于排序的随机数据
Raney, George Neal	拉尼·乔治·尼尔
Range queries	范围查询
Ranking, *see* sorting	排列，见排序
Raver, N.	雷弗·N
Ray Chaudhuri Dwijendra Kumar	雷·乔杜里·德维泽德拉·库马尔
Read-back check	向后读校验
Read-backwards	向后读
Readings of a permutation	一个排列的输入
Real-time applications	实时应用
Real-valued search tree	实值查找树
Rearrangeable network, *see* permutation network	可重新安排的网络，见排列网络
Reciprocals	倒数
Record	记录
Recurrence relations, techniques for solving	用于解决递归关系的技术
Recursion	递归
Recursion induction	递归归纳
Rehashing	重杂凑
Reingold, Edward Martin	莱因戈尔德·爱德华·马丁
Remington Rand Corporation	雷明顿·兰德公司
Removal, *see* Deletion	撤销，见删去
Replacement selection	替代选择
Reversal of data	数据的反排
Rewinding tape	重绕带
Rice, Stephan Oswald	赖斯·斯蒂芬·奥斯瓦德
Riesel, Hans	里塞尔·汉斯
Right-threaded tree	右穿线的树
Right-to-left (or left-to right) maxima or minima	从右到左（或从左到右）的极大或极小
Riordan, John	赖尔登·约翰
Rising, Hawley	赖辛·霍利
Rivest, Ronald Linn	里夫斯特·罗纳德·林
Roberts, David C.	罗伯特·戴维·C
Robinson, Gilbert de Beauregard	鲁宾逊·吉尔伯特·德·博罗加德
Rochester, Nathaniel	罗彻斯特·纳撒尼尔
Rocbuck, Alvah Curtis	罗巴克·阿尔瓦·柯蒂斯
Rollett, Arthur Percy	罗勒特·阿瑟·珀西
Roselle, David Paul	罗塞尔·戴维·保罗
Rotation of data	数据的转动
Rotem, Doron	罗登姆·多伦
Rothe, Heinrich August	罗泽·海因里希·奥古斯特
Rouché, Eugène	鲁彻·尤金
Royner, Paul David	罗夫纳·保罗·戴维
Royalties, use of	特许权的用法

Runs of a permutation 一个排列的路段
Russell, David Lewis 拉塞尔·戴维·刘易斯
Russell, Robert C. 拉塞尔·罗伯特·C
Rutherfold, Daniel Edwin 拉瑟福德·丹尼尔·埃德温

Sable, Jerome David 萨珀尔·杰罗姆·戴维
Sacerdoti, Earl David 萨西尔多蒂·厄尔·戴维
Sackman, Bertram Stanley 萨克曼·伯特伦·斯坦利
Samplesort 抽样排序
Samuel 塞缪尔
Samuel, Arthur Lee 塞缪尔·阿瑟·李
Sardelius, Martin 桑德里尼斯·马丁
Satellite information; Record minus key 附属信息；除键以外的记录内容
Scatter storage 散列存储
Schay, Geza. Jr 小贾伊·格扎
Schensted, craige Eugene 申斯迪德·克雷格·尤金
Schlegel, Stanislaus Ferdinand Victor 施莱格尔·斯坦尼斯劳斯·费迪南德·维克多
Schlumberger, Maurice Lorrain 施卢姆伯杰·莫里斯·洛兰
Schönhage Arnold 舍恩哈格·阿诺德
Schreier, Józef 施莱尔·乔泽夫
Schützenberger, Marcel Paul 舒普伯杰·马塞尔·保罗
Schwartz, Eugene Sidney 施瓦茨·尤金·西德尼
Schwartz, Jules 施瓦茨·朱尔斯
Scoins, H. Ian 斯科因斯·H. 伊恩
Scoville, Richard Arthur 斯科维尔·理查德·阿瑟
Scrambling function 爬树函数
Searching, *see* External Searching, Internal Searching, Comparison of keys, Digital Searching, Hashing, Secondary key retrieval; Static table Searching, Symbol table algorithms 查找，见外部查找，内部查找，键的比较，数字查找，杂凑，辅键的查找，静态查表，符号表算法
for Closest match 查找最接近的匹配
methods, *see* *B*-tree, Balanced trees, Binarg Search Chaining, Digital tree search, Fibonaccian Search, Interpolation Search Open addressing, Patricia, Sequential search, Tree Search, Trie search 查找方法，见 *B*-树，平衡的树，二分查找拉链，数字树查找，斐波那契查找，内插查找，开式寻址，Patricia，顺序查找，树查找，检索结构查找
related to sorting 同排序有关的查找
text 文本查找
Sears, Richard Warren 西尔斯·理查德·沃伦
Secondary clustering 二次堆集
Secondary key retrieval 用辅助键检索
Secondary Storage, *see* External Searching, External Sorting 二次存储，见外部查找，外部排序
Sedgewick, Robert 塞奇维克·罗伯特
Seek time 寻找时间
Sefer, yezirah 西费尔·耶齐拉
Selection of *t*th largest network for 第 *t* 个最大的选择用于第 *t* 个最大的选择的网络
Selection sorting 选择排序
Selection trees 选择树
Self-modifying program 自修改程序
Self organizing file 自组织文件
Selfridge John, Lewis 塞尔弗里奇·约翰·刘易斯
Senko, Michael Edward 森科·迈克尔·爱德华
Sentinel 哨兵，标志

Seperation sorting, *see* distribution	分开排序，见分布
Sequence search, *see* Digital tree search	顺序查找，见数字树查找
Sequential allocation	顺序分配
Sequential file processing	顺序文件处理
Sequential Searching	顺序查找
Sets, testing equality	判断集合是否相等
testing inclusion	判断集合的包含关系
Seward, Harold H.	西沃德·哈罗德·H
Sexagesimal number	十六进制数
Shackleton, P.	沙克尔顿·P
Shanks, Daniel	香克斯，丹尼尔
Shannon Claude Eldwood	香农·克劳德·埃尔德伍德
Shapiro, Gerald Norris	夏皮罗·杰拉尔德·诺里斯
Shapiro, Henry David	夏皮罗·亨利·戴维
Shar, Leonard Eric	沙尔·伦纳德·埃里克
Shell, Donald Lewis	谢尔·唐纳德·刘易斯
Shellsort	谢尔排序
Shepp, Lawrence Alan	谢波·劳伦斯·艾伦
Sherman, Philip Martin	谢尔曼·菲利普·马丁
Shockley, William Bradford	肖克利·威廉·布雷德福德
Sholmov, Leonid Ivanovich	肖尔莫夫·利奥尼德·伊凡诺维奇
Shrikhande, Sharadchandra Shankar	施里克汉德·沙拉德钱德拉·香克
Shuffling	洗牌
Siegel, Shelby	西格尔·谢尔比
Sift-up	筛选
Sifting, *see* Straight insertion	筛选，见直接插入
Signed-magnitude notation	带（正、负）号的量表示法
Silver, Ronald Lazarus	西尔弗·罗纳德·拉扎勒斯
Simulation	模拟
Simultaneous Comparisons	同时比较
Singer, Theodore	辛格·西奥多
Single hashing	单个杂凑
Singleton, Richard Collom	辛格尔顿·理查德·科洛姆
Sinking Sort, *see* straight insertion	陷入排序，见直接插入
SLB(Shift left Ax binary)	SLB（对AX寄存器作二进位左移）指令
Sloane, Neil James Alexander	斯隆·尼尔·詹姆斯·亚历山大
Stupecki, Jerzy	斯拉凯克基·杰齐
Smallest-in-first-out, *see* Priority queue	最小的先出，见优先队
Smith, Alan Jay	史密斯·艾伦·杰伊
Smith, Alfred Emanuel	史密斯·艾尔弗雷德·伊曼纽尔
Smith, Cyril Stanley	史密斯·西里尔·斯坦利
Smith, Wayne Earl	史密斯·韦恩·厄尔
Snow job	扫积雪的工作
Sobel, Milton	索贝尔·米尔顿
Sobel, Sheldon	索贝尔·谢尔登
Software	软件
Solitaire	单人纸牌戏
Sort generators	排序生成器
Sorting (into order), *see* External Sorting, Internal Sorting; Address Calculation Sorting, Distribution Sorting, Enumeration Sorting, Exchange Sorting, Insertion Sorting, Merge Sorting, Selection Sorting	排（成有序），见外部排序，内部排序；地址计算排序，分布排序，枚举排序，交换排序，插入排序，合并排序，选择排序

address table 地址表排序
key- 键排序
library Cards 图书馆卡片排序
linear average time algorithm for 用于排序的线性平均时间算法
list, *see* List sorting 表排序，见表排序
methods, *see* Binary insertion sort, Bitonic sort, Bubble sort, Cooktail shaker sort, Comparison counting sort, Distribution Counting sort, Heapsort, Interval exchange sort, List insertion sort, List merge sort, Median-of-three quicksort, Merge exchange sort, Merge insertion sort, Multiple list insertion sort, Natural merge sort, Odd-even transposition sort *P*-operator sort, Pratt sort, Quicksort, Radix exchange sort, Radix insertion sort, Radix list Sort, Samplesort, Shellsort, Straight insertion sort, Straight merge sort, Straight selection sort, Tree insertion sort, Tree selection sort, Two-way insertion sort; *see* also Merge pattern 排序方法，见二进插入排序，双调排序，气泡排序，鸡尾混合排序，比较计数排序，分布计数排序，堆排序，区间交换排序，表插入排序，表合并排序，三者取中快速排序，合并交换排序，合并插入排序，多重表插入排序，自然合并排序，奇偶转置排序，*P*-操作符排序，普拉特排序，快速排序，基数交换排序，基数插入排序，基数表排序，抽样排序，谢尔排序，直接插入排序，直接合并排序，直接选择排序，树插入排序，树选择排序，两路插入排序；也见合并型式
networks for 用于排序的网络
One tape 单带排序
related to searching 同查找有关的排序
stable 稳定的排序
topological 拓扑排序
two-tape 双带排序
Sós, Vera Turan 索斯·维拉·特兰
Soundex 探测法
Sparse array 稀疏数组
Speedup, *see* loop optimization 加速，见循环优化
Sperner, Emanuel 斯珀纳·伊曼纽尔
Speroni, Joseph 斯珀洛尼·约瑟夫
Splitting a list 分开一个表
Sprugnol, Renzo 斯普鲁格诺里·伦佐
Spruth, wilhelm Gustav Bernhard 斯普鲁思·威廉·古斯塔夫·伯恩哈德
SRB (Shift Right AX Binary) SRB（把 AX 寄存器作二进位右移）指令
Stable sorting 稳定的排序
Stacks 栈
Staël-Holstein, Anne Louise Germaine Necker, Baronne de 斯塔耶尔-霍尔斯坦安妮·路易斯·杰曼·尼科尔·巴伦尼·德
Standard sorting network 标准排序网络
Stanfel, Larry E. 斯坦菲尔·拉里·E
Stanley, Richard peter 斯坦利·理查德·彼得
Stasevich, G. V. 斯塔西维奇·G.V.
Static table searching 静态查表
Stearns, Richard Edwin 斯特恩斯·理查得·埃德温
Stegun, Irene Anne 斯蒂冈·艾琳·安妮
Steiner, Jacob, triple system 斯坦纳·雅各布·三元组系统
Steinhaus, Hugo Dynoizy 斯坦豪斯·雨果·戴诺伊齐
Step-downs 往下走
Stirling, James, approximation 斯特林·詹姆斯，近似
Stirling numbers 斯特林数
Stockmeyer, Paul kelly 斯托克迈耶·保罗·凯利
Stone, Harold Stuart 斯通·哈罗德·斯图尔特

Stop Start time	停止-开始时间
Straight insertion sort	直接插入排序
Straight merge sort	直接合并排序
Straight selection sort	直接选择排序
Stratum, *see* Cylinder	层次，见柱面
Strauss, Ernst Gabor	施特劳斯·厄恩·加博尔
String: A sequence of items	串：项的一个序列
String: An ordered block, *see* Run	串：一个有序的块区，见路段
Successful Search	成功的查找
Sue, Jeffrey yen	休·杰弗里·延
Sugito yoshio	杉藤芳雄
Summing factor	求和因子
Superimposed coding	叠加的编码
Surnames, encoding	对姓氏编码
Sussenguth, Edward Henry, Jr	小萨斯森古思·爱德华·亨利
Świerczkowski, Stanisław Slawomir	斯怀切柯夫斯基·斯坦尼斯洛·斯拉沃米尔
Swift, Jonathan	斯威夫特·乔纳森
Sylvester, James Joseph	斯尔威斯特·詹姆斯·约瑟夫
Symbol table algorithms	符号表算法
Symmetric functions	对称函数
Symmetric order: Left Subtree, then root, then right subtree	对称次序：左子树，然后根，然后右子树
Szamerédi Endre	森默勒迪·恩德雷
Szekeres, George	泽克勒斯·乔治
T-C algorithm, *see* Hu-Tucker algorithm	T-C算法，见胡-塔克算法
T-fifo tree	*T*-先进先出树
T lifo tree	*T*-后进先出树
Table	表格
Tableau	图表
Tag Sorting, *see* key sorting	标签排序，见键排序
Tainiter, Melvin	泰尼特·默尔文
Takács, Lajos	塔卡斯·拉乔斯
Tan, Kok Chye	陈国材
Tangent numbers	正切数
Tanny, Stephen	坦尼·斯蒂芬
Tape, *see* Magnetic tape	带，见磁带
Tape Controller	带控制器
Tape Searching	带的查找
Tape splitting	带的分开
Polyphase merge	多阶段合并时带的分开
Tape units reliability of	带设备的可靠性
Tardiness	缓慢，滞延
Tarjan, Robert Endre	塔简·罗伯特·恩德雷
Tartar, Michael E	塔塔尔·迈克尔·E
Tennis tournament	网球锦标赛
Terquem, Olry	特奎姆·奥里
Text Searching	稿文查找
Thiel, L. H	蒂尔·L. H
Thrall, Robert McDowell	思罗尔·罗伯特·麦克道尔
Threaded tree	穿线树
Three-distance theorem	三距离定理
Tomlinson, Robert, L. Jr	小汤姆林森·罗伯特·L

Topological Sorting	拓扑排序
Total order	全序
Touchard, Jacques	塔查德·雅克
Tournament	锦标赛
Transitive law	传递律
Transmission time	传输时间
Transposing a matrix	转置一个矩阵
Transposition Sorting, *see* Exchange Sorting	转置排序，见交换排序
Tree hashing	树杂凑
Tree insertion sort	树插入排序
Tree representation of distribution pattern	分布型式的树表示
Tree representation of merge pattern	合并型式的树表示
Tree selection sort	树选择排序
Tree search, *see also* digital tree search	树查找，也见数字树查找
Treesort, *see* Tree selection sort, Heapsort	树排序，见树选择排序，堆排序
Tree traversal	树遍历
Triblet, Charles Siegfried	特里博勒特·查尔斯·西格弗里德
Trichotomy law	三分律
Trie search	检索结构的查找
generalized	广义的检索结构的查找
Triple systems	三元组系统
Triply linked binary tree	三重链接的二叉树
Truesdell, Leon Edgar	特鲁斯德尔·利昂·埃德加
Truncated Octahedron	截八面体
Trybula, Stanisław	特赖布拉·斯塔尼斯洛
Tucker, Alan Curtiss	塔克·艾伦·柯蒂斯
Tumble instruction	滚进指令
Turán, Sós Vera	特兰·索斯·维拉
Turing, Alan Mathison, machine	图林·艾伦·马西森机器
Turski, Wladyslaw	塔斯基·乌拉迪斯洛
Twin heap	孪生堆
Twin primes	孪生质数
Two's complement notation	二的补码记号
Two-dimensional trees	二维树
Two-tape sorting	双带排序
Two-way insertion	两路插入
Typesetting	排字
Ullman, Jeffrey David	厄尔曼·杰弗里·戴维
Uniform binary search	均匀的二分查找
Uniform hashing	均匀杂凑
Uniform sorting	一致排序
UNIVAC I	尤尼瓦克 I 计算机
UNIVAC Larc	尤尼瓦克计算机上的 Larc 程序
Unsuccessful Search	不成功的查找
Updating a file	更新一个文件
Uzgalis, Robert	乌泽加里斯·罗伯特
Van der pool, Jan Albertus	范·德·普尔·简·艾伯塔斯
Van Emden, Maarten Herman	范·埃姆登·马尔登·赫尔曼
Van Lint, Jacobus Hendricus	范·林特·雅各布斯·亨德里沙斯
Van Valkenburg Mac Elwyn	范·瓦肯伯格·麦克·埃尔温
Van Voorhis, David Curtis	范·沃勒斯·戴维·柯蒂斯

Van Wijngaarden Adriaan 范·维因加尔登·艾德里安
Vandermonde, Alexamder Théophile 范德蒙德·亚历山大·西奥菲尔
Varga, Richard Steven 瓦尔加·理查德·史蒂文
Variable-length code 可变长的代码
Variable-length keys 可变长的键
Vector representation of merge pattern 合并型式的向量表示
Venn, John, L 维恩·约翰·L
Вершик, Анатолий Моисеич 维尔希克·阿纳托里·莫依谢维兹
Virtual memory 虚拟存储
Vitter, Jeffrey Scott 维特·杰弗里·斯科特
Von Mises, Richard, Edler 冯·迈西斯·理查德·埃德勒
Von Neumann, John 冯·诺伊曼·约翰
Vuillemin, Jean Etienne 武伊尔勒明·琼·埃逊恩尼
Vyssotsky, Victor Alexander 维索特斯基·维克多·亚历山大

Waks, David Jeffrey 威克斯·戴维·杰弗里
Waksman, Abraham 瓦克斯曼·亚伯拉罕
Walker, Ewing S. 沃克·尤因·S
Walker, Wayne A 沃克·韦恩·A
Wallis, John R, Jr 小沃里斯·约翰·R
Walters, John R., Jr 小沃尔特斯·约翰·R
Wang, Y. W 王亚威
Wang, Yihsiao 王义孝
Ward, Morgan 沃德·摩根
Waters, Samuel Joseph 沃特斯·塞缪尔·约瑟夫
Weidekind, Hartmut 韦得金德·哈特马特
Weighing 韦金
Weight-balanced frees 加权的平衡树
Weighted path length 加权的路径长度
Weisert, Conrad 韦塞特·康拉德
Weiss, Benjamin 韦斯·本杰明
Weiss, Harold 韦斯·哈罗德
Weissblum, Walter 韦斯布卢姆·沃尔特
Wells, Mark Brimhall 韦尔斯·马克·布林霍尔
Wessner, Russell 韦斯纳·拉塞尔
Wheeler David John 惠勒·戴维·约翰
Whirlwind 沃勒温德
Wiener, Norbert 维纳·诺伯
Williams, Francis A, Jr 小威廉斯·弗朗西斯·A
Williams, John William Joseph 威廉斯·约翰·威廉·约瑟夫
Windley, Peter, F 温德利·彼得·F
Wong, Chak-Kuen 黄泽权
Woodall, Arthur David 伍德尔·阿瑟·戴维
Woodrum, Luther Jay 伍德朗·卢瑟·杰伊
Worst binary Search trees 最劣的二叉查找树
Wrench, John William Jr 小伦奇·约翰·威廉
Wright, Edward Maitland 赖特·爱德华·梅特兰
Wyman, Max 怀曼·马克斯

Yamada Hiseo M 山田尚勇
Yao, Andrew Chi-Chih 姚期智
Yao, Foong France 姚储枫
Yap, Chee-keng 叶志坚
Yoash, Nahal Ben (pseud of Gidson yuval) 约亚斯·纳哈尔·本（吉迪·尤瓦尔）
Youden, William Wallace 尤登·威廉·华莱士
Young, Alfred (tableaux) 杨·阿尔弗雷德（杨氏图表）
Yuen, Pasteur Shih Teh 阮·帕斯迪尔·士德

Zalk, Martin, M 扎克，马丁·M
Zave, Derek Alan 扎夫·德里克·艾伦
Zeckendorf, Edouard 泽肯多夫·埃多瓦德
Zero-one principle 0-1原理
Zeta function 载塔（ζ）函数
Zipf, George Kingsley 泽弗·乔治·金斯利
Zipf's law 泽弗定律
Zoberbier, W 佐贝尔比尔·W
Zolnowsky, John Edward 佐诺斯基·约翰·爱德华

2D trees 2D树
2-ordered 2-有序的
2-3 trees 2-3树
80-20 law 80-20定律

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
	⊔	A	B	C	D	E	F	G	H	I	⊕	J	K	L	M	N	O	P	Q	R	Φ	Π	S	T	U

00	1	01	2	02	2	03	10
空操作 $NOP(0)$		$rA \leftarrow rA+V$ $ADD(0:5)$ $FADD(6)$		$rA \leftarrow rA-V$ $SUB(0:5)$ $FSUB(6)$		$rAX \leftarrow rAXV$ $MUL(0:5)$ $FMUL(6)$	
08	2	09	2	10	2	11	2
$rA \leftarrow V$ $LDA(0:5)$		$rI1 \leftarrow V$ $LD1(0:5)$		$rI2 \leftarrow V$ $LD2(0:5)$		$rI3 \leftarrow V$ $LD3(0:5)$	
16	2	17	2	18	2	19	2
$rA \leftarrow -V$ $LDAN(0:5)$		$rI1 \leftarrow -V$ $LDIN(0:5)$		$rI2 \leftarrow -V$ $LD2N(0:5)$		$rI3 \leftarrow -V$ $LD3N(0:5)$	
24	2	25	2	26	2	27	2
$F(M) \leftarrow rA$ $STA(0:5)$		$F(M) \leftarrow rI1$ $ST1(0:5)$		$F(M) \leftarrow rI2$ $ST2(0:5)$		$F(M) \leftarrow rI3$ $ST3(0:5)$	
32	2	33	2	34	1	35	1 + 7
$F(M) \leftarrow rJ$ $STJ(0:2)$		$F(M) \leftarrow 0$ $STZ(0:5)$		装置下忙碌吗？ $JBUS(0)$		控制，装置F $IOC(0)$	
40	1	41	1	42	1	43	1
rA:0，跳转 $JA[+]$		$rI1$:0，跳转 $J1[+]$		$rI2$:0，跳转 $J2[+]$		$rI3$:0，跳转 $J3[+]$	
48	1	49	1	50	1	51	1
$rA \leftarrow [rA]? \pm M$ $INCA(0)DECA(1)$ $ENTA(2)ENNA(3)$		$rI1 \leftarrow [rI1]? \pm M$ $INC1(0)DEC1(1)$ $ENT1(2)ENN1(3)$		$rI2 \leftarrow [rI2]? \pm M$ $INC2(0)DEC2(1)$ $ENT2(2)ENN2(3)$		$rI3 \leftarrow [rI3]? \pm M$ $INC3(0)DEC3(1)$ $ENT3(2)ENN3(3)$	
56	2	57	2	58	2	59	2
$rA(F):V \rightarrow CI$ $CMPA(0:5)$ $FCMP(6)$		$rI1(F):V \rightarrow CI$ $CMP1(0:5)$		$rI2(F):V \rightarrow CI$ $CMP2(0:5)$		$rI3(F):V \rightarrow CI$ $CMP3(0:5)$	

一般形式

C	t
说　明 $OP(F)$	

C = 操作码，指令的(5:5)字段

F = 操作码（OP）变形，指令的（4:4）字段

M = 变址后的指令地址

$V = F(M)$ = 单元M的F字段的内容

OP = 操作的符号名

(F) = 标准的F设置

t = 执行时间；T' = 互锁时间

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
V	W	X	Y	Z	0	1	2	3	4	5	6	7	8	9	.	,	( )	+	−	*	/	=	$	<	>	@	;	:	'

04	12	05	1	06	2	07	1 + 2 F
rA← rAX/V rX←余数 DIV(0:5) FDIV(6)		特殊的 NUM(0) CHAR(1) HLT(2)		移M位 SLA(0)SRA(1) SLAX(2)SRAX(3) SLC(4)SRC(5)		从M传送F个字到 rI1 MOVE(1)	
12	2	13	2	14	2	15	2
rI4←V LD4(0:5)		rI5←V LD5(0:5)		rI6←V LD6(0:5)		rX←V LDX(0:5)	
20	2	21	2	22	2	23	2
rI4←−V LD4N(0:5)		rI5←−V LD5N(0:5)		rI6←−V LD6N(0:5)		rX←−V LDXN(0:5)	
28	2	29	2	30	2	31	2
F(M)←rI4 ST4(0:5)		F(M)←rI5 ST5(0:5)		F(M)←rI6 ST6(0:5)		F(M)←rX STX(0:5)	
36	1 + T	37	1 + T	38	1	39	1
输入，装置F IN(0)		输出，装置F OUT(0)		装置F准备好了吗？ JRED(0)		跳 转 JMP(0)JSJ(1) JOV(2)JNOV(3) 还有下边的〔*〕	
44	1	45	1	46	1	47	1
rI4:0，跳转 J4〔+〕		rI5:0，跳转 J5〔+〕		rI6:0，跳转 J6〔+〕		rX:0，跳转 JX〔+〕	
52	1	53	1	54	1	55	1
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)	
60	2	61	2	62	2	63	2
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)	

rA = 寄存器A
rX = 寄存器X
rAX = 寄存器AX作为一个寄存器
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)

计算机程序设计技巧

第三卷 排序和查找

(美) D.E.克努特 著
曹胜文 高连霞 译
闫振群 等校

JISUANJI C GY