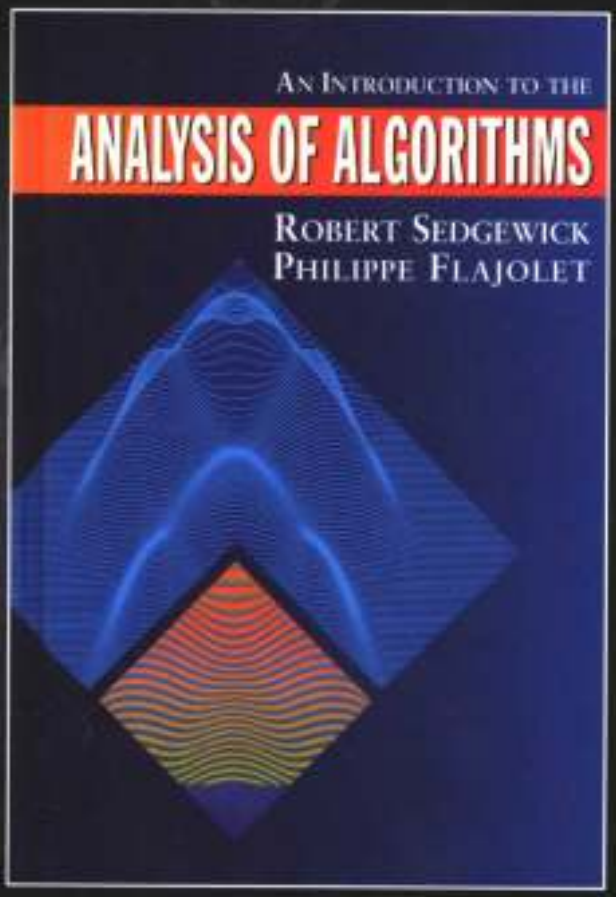
计 算 机 科 学 丛 书

算法分析导论

(美) Robert Sedgewick (法) Philippe Flajolet 著 冯舜玺 李学武 裴伟东 等译



An Introduction to the Analysis of Algorithms

机械工业出版社
China Machine Press

分析算法的人享有双重的幸福。首先，他们能够体验到优雅数学模式纯粹的美，这种模式存在于优美的计算过程之中；其次，当他们的理论使得其他工作能够做得更快、更经济时，他们能够得到实际的褒奖。本书作者是算法分析领域的世界级领袖，这部著作我们也是期盼已久了。

——Donald E. Knuth

本书全面介绍了算法的数学分析中使用的基本方法，所涉及的内容来自经典的数学课题（包括离散数学、初等实分析、组合数学），以及经典的计算机科学课题（包括算法和数据结构）。虽然书中论述了“最坏情形”和“复杂性”分析所需的基本数学工具，但是重点还是讨论“平均情形”或“概率”分析。论题涉及递归、生成函数、渐近性、树、串、映射等内容，以及对排序、树查找、串查找和散列诸算法的分析。

尽管人们极为关注算法的数学分析，但是关于普遍使用的方法和模型方面的基本信息尚不能为该领域的学生和研究人员直接使用。作者在本书中处理这种需求，把该领域出现的挑战以及为迎接这些挑战所必需的背景资料完美地结合在一起。

作者简介

Robert Sedgewick 斯坦福大学博士（导师为Donald E. Knuth），普林斯顿大学计算机科学系教授，Adobe Systems公司董事，曾是Xerox PARC的研究人员，还曾就职于美国国防部防御分析研究所以及INRIA。

Philippe Flajolet INRIA的高级研究主任，在Ecole Polytechnique和普林斯顿大学任教，并在斯坦福大学、智利大学和弗吉尼亚技术大学拥有访问席位。他还是法国科学院的通信会员。

ISBN 7-111-16441-5

9 787111 164418

HZ Books

华章图书

上架指导：计算机/算法

华章网站 http://www.hzbook.com

网上购书：www.china-pub.com

投稿热线：(010) 88379604

购书热线：(010) 68995259, 68995264

读者信箱：hzjsj@hzbook.com

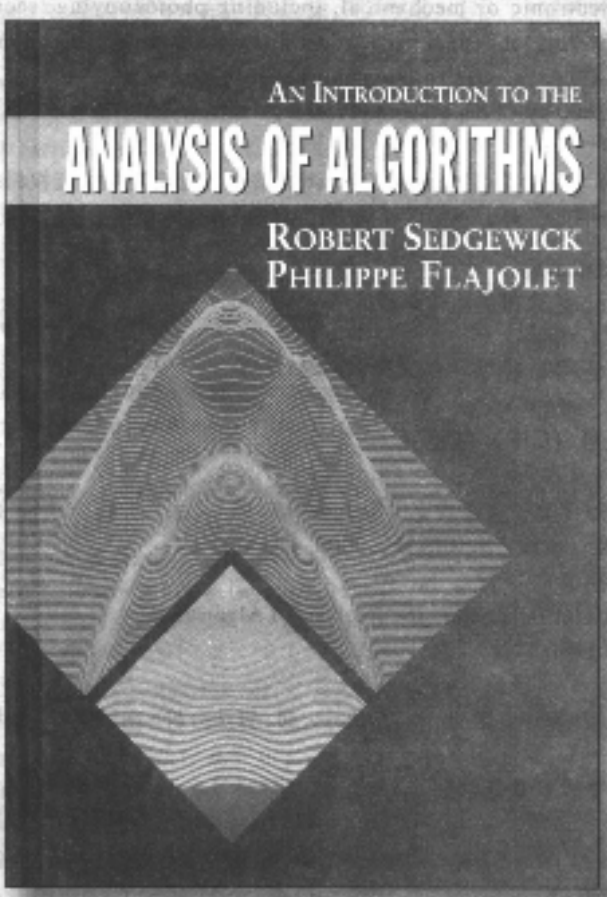
ISBN 7-111-16441-5/TP・4273

定价：38.00 元

计 算 机 科 学 丛 书

算法分析导论

(美) Robert Sedgewick (法) Philippe Flajolet 著 冯舜玺 李学武 裴伟东 等译



An Introduction to the Analysis of Algorithms

机械工业出版社
China Machine Press

本书阐述了用于算法数学分析的主要方法，所涉及的材料来自经典数学课题，包括离散数学、初等实分析、组合数学，以及来自经典的计算机科学课题，包括算法和数据结构。本书内容集中覆盖基础、重要和有趣的算法，前面侧重数学，后面集中讨论算法分析的应用，重点是算法分析的数学方法。每章包含大量习题以及参考文献，使读者可以更深入地理解书中的内容。

本书适合作为高等院校数学、计算机科学以及相关专业的本科生和研究生的教材，也可供相关技术人员参考。

Authorized translation from the English language edition entitled An introduction to the analysis of algorithms by Robert Sedgewick, Philippe Flajolet, published by Pearson Education, Inc, publishing as Addson-Wesley (ISBN0-201-40009-X), Copyright © 1999 by Addison-Wesley Publishing Company, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage retrieval system, without permission of Pearson Education, Inc.

Chinese simplified language edition published by China Machine Press.

Copyright © 2000 by China Machine Press.

本书中文简体字版由美国Pearson Education培生教育出版集团授权机械工业出版社独家出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书法律顾问　北京市展达律师事务所

本书版权登记号：图字：01-2003-4919

图书在版编目（CIP）数据

算法分析导论/（美）塞奇威克（Sedgewick, R.）（法）弗拉吉莱特（Flajolet, P）著；冯舜玺等译．-北京：机械工业出版社，2006.4

（计算机科学丛书）

书名原文：An Introduction to the Analysis of Algorithms

ISBN 7-111-16441-5

Ⅰ.算…　Ⅱ.①塞…　②弗…　③冯…　Ⅲ.算法分析-高等学校-教材　Ⅳ.TP311

中国版本图书馆CIP数据核字（2005）第030721号

机械工业出版社（北京市西城区百万庄大街22号　邮政编码　100037）

责任编辑：朱起飞

北京中兴印刷有限公司印刷·新华书店北京发行所发行

2006年4月第1版第1次印刷

787mm × 1092mm 1/16 · 20.5印张

定价：38.00元

凡购本书，如有倒页、脱页、缺页，由本社发行部调换

本社购书热线：（010）68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专程为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下:

电子邮件: hzjsj@hzbook.com
联系电话: (010) 68995264
联系地址: 北京市西城区百万庄南街1号
邮政编码: 100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周克定	周傲英	孟小峰	岳丽华	范　明
郑国梁	施伯乐	钟玉琢	唐世渭	袁崇义
高传善	梅　宏	程　旭	程时端	谢希仁
裘宗燕	戴　葵			

译 者 序

算法分析一般包括两种不同的方法。第一种方法研究确定最坏情形的性能，有时称之为计算复杂性。当面对一种新的算法时，算法分析能粗略地分析：新算法能够有多好？它比其他算法性能大致好多少？就算法分析来说，这种复杂性分析由于抛弃一些次要因素而丧失不少的精度，它提供不了算法具体实现的性能特征以及与其他算法具体比较所需要的足够信息。不过，我们可以把它看成是形成更精练、更准确分析过程的第一步。第二种方法通过确定最佳情形、最坏情形以及平均情形的性能来精确地刻画算法的性能，在需要的时候，能够通过可以精化的方法准确地预测特定算法的性能特征，其中最常见的是对时间和空间这样一些基本资源的消耗，尤其是时间。算法分析表示整个分析过程，它提供任意精度的解决方案。

本书是对算法的数学分析中主要方法的综述，涉及离散数学、数学分析、组合数学以及数据结构和算法。由于重点在于平均情形的分析，因此必然要涉及模型、分布、矩阵以及概率论和数理统计中的其他内容。

经过近几十年的发展，算法分析已经十分成熟，能够在标准的计算机课程中占有一席之地，本书为那些徘徊在该领域之外却苦于不得其门而入的新手指出通往研究之门的捷径。为本科生开设这门课程可以有少数内容现用现补，研究生在先修有关数学和计算机的相应知识的基础上，可以将本书用于算法分析的导论性课程。数学和计算机领域中不同专业的学生，选修本书可以各有侧重，作者在前言中提出了指导性的建议。

本书语言简炼，推导紧凑，能够很好地锻炼我们的独立阅读能力。在验证书中的结论和求解练习的结果时，若能使用像Matlab、Mathematica或MAPLE这样的计算机软件，则可帮助我们摆脱繁琐的演算，节省宝贵的时间。

本书的另一大特点是课文中以及各章末尾所列出的参考文献，对于有心深造的读者，这是十分宝贵的财富。

全书共有8章内容。第3章由李学武翻译，第4章、第6章、第7章和第8章由裴伟东翻译，其余工作由冯舜玺完成。因时间和水平所限，译文难免存在一定的问题，恳切期望广大读者批评指正。

译 者

2006年3月

序

分析算法的人享有双重的幸福。首先，他们能够体验到优雅数学模式纯粹的美，这种模式存在于优美的计算过程之中。其次，当他们的理论使得其他工作能够做得更快、更经济时，他们能够得到实际的褒奖。

数学模型始终是所有科学活动至关重要的灵感，尽管它们只是真实世界现象的近似理想化。在计算机内部，这种模型比以前更优美，因为计算机程序创建了人造世界。在这样的世界中，数学模型常常得以精确地使用。我想，这就是为什么当我还是研究生时就沉迷于算法分析，以及为什么这个课题迄今为止始终是我一生主要工作的原因。

然而，直到最近，算法分析在很大程度上依然是研究生和研究人员的禁区。它的概念确实不深奥也不困难，但是它们相对新颖，因此，挑选出学习它们和使用它们的最佳方法需要时间。

如今，经过30多年的发展，算法分析已经成熟到这样一种程度，即它已经为其在标准的计算机课程中占有一席之地做好了准备。因此，我们期盼已久的这部Sedgewick和Flajolet的著作极受欢迎。本书作者不仅是这个领域在世界范围内的领袖，而且还是理论阐述的大师。我敢肯定，每一位严肃的计算机科学家都将发现本书在许多方面颇富启迪性。

D. E. Knuth

前　言

本书是对算法数学分析中主要方法的综述。所涉及的材料来自经典的数学课题，包括离散数学、初等实分析、组合数学，以及来自经典的计算机科学课题，包括算法和数据结构。重点在于“平均情形”或“概率”分析，不过，也包括“最坏情形”和“复杂性”分析所需要的基本数学工具。

我们假设读者熟悉计算机科学和实分析中的基本概念。概括地说，读者应该既能编写程序，又能够证明定理；在其他方面，本书是自包容的。书中还在文献里提供了丰富的预备性材料。我们计划中的本书姐妹篇将讨论更为先进的方法。这两部著作总共涵盖那些主要的方法和技巧，并提供通向算法分析日益增长的文献的捷径。

本书计划作为算法数学分析初级和高级课程的教科书。它也可以在计算机科学的离散数学课程中使用，因为它包含了离散数学和组合数学中的基本方法以及计算机科学学生所熟悉的重要离散结构的基本性质。传统上，在这些课程中知识范围要广一些，但是许多教师可能发现这里的做法是使学生从事实质性部分学习的有用方式。本书还可以作为数学和应用数学的学生了解与算法和数据结构相关的计算机科学原理的教材。

辅以文献中论文的补充，本书可作为研究生算法分析导论性课程的基础来使用，或作为那些想要接近该领域文献的数学或计算机科学研究人员的参考书或自学基础材料。它还可以作为应用和方法的资源为字符组合数学和离散数学的学生和研究人员使用。

尽管算法数学分析方面的文献很多，但是关于普遍使用的方法和模型方面的基本信息尚不能很容易地为该领域的学生和研究人员直接使用。本书的目的就是处理这种情况，把想要提供给读者该领域问题的正确评价和学习正在发展以满足这些挑战的高级工具的需求背景的大量材料汇聚在一起。

背景要求。假设数学程度为在大学学习一年或两年的水平。组合数学和离散数学的基本课程可以提供有用的背景知识（可能与本书的某些内容重叠），实分析、数值方法或初等数论中的课程也是如此。我们将涉及所有这些领域，不过在这里只综述必需的材料，那些想要知道更多信息的读者可以参考标准的教科书。

假设编程经验为在大学学习过一个或两个学期的水平，包括初等数据结构。我们不在编程和算法实现问题上过多地耗费精力，算法和数据结构则是我们研究的中心课题。从某种意义上讲，我们的处理是完善的：对于基本的信息，我们给出概要的总结，必要时可以参考标准的教科书和一些基本的原始资料。

强烈推荐读者在数学计算时使用诸如MAPLE或Mathematica这样的计算机软件。当验证本书中的材料或求解习题结果时，这些软件系统可以帮助摆脱烦琐的计算。

相关的书籍。相关的教科书包括Knuth的*The Art of Computer Programming*；Gonnet和Baeza-Yates的*Handbook of Algorithms and Data Structure*；Sedgewick的*Algorithms*；Graham、Knuth和Patashnik的*Concrete Mathematics*；Cormen、Leiserson和Rivest的*Introduction to Algorithms*。本书可以看作是对这些著作的增补，下面依次评述。

实质上，本书最接近于Knuth的开创性著作。不过，我们的重点是在分析的数学方法上，而Knuth著作的范围广而全，算法的性质起着主要的作用，而分析的方法则是第二位的。本书可以作为Knuth的著作中所论及的先进结果的基础准备。

我们还讨论了自从Knuth的书出版以来始终在发展的算法分析中的方法和结果。Gonnet和Baeza-Yates的著作是这些结果的透彻综述，包括全面的参考文献；它主要介绍源自文献的一些相关结果。本书提供了通向该文献的基本预备内容。

我们也努力把重点放在基础、重要和有趣的算法上，像Sedgewick的*Algorithms*中所描述的一些算法，而Graham、Knuth和Patashnik的*Concrete Mathematics*则几乎全部侧重在数学方法方面。我们想要使本书成为Graham、Knuth和Patashnik的书中所讨论的基本数学方法和Sedgewick的书所涉及的基本算法的桥梁。

Cormen、Leiserson和Rivest的*Introduction to Algorithms*是那些提供通向算法的“设计和分析”研究文献著作的代表，一般是基于方法性能的粗略的最坏情形估计的。当需要更精确的结果时（对于大多数重要的方法），更复杂的模型和数学工具是必不可少的。Cormen、Leiserson和Rivest把重点放在算法的设计（通常带有为最坏情形性能定界的目的）上，而解析结果则用于帮助指导算法的设计。另一方面，我们的书是对他们的书以及类似的书的补充，因为我们侧重于算法的分析，特别是对那些能够用于获取可以预报算法性能的详细结果的方法。在这个过程中，我们也考虑与各种经典数学工具的关系，第1章全部用来描述这种环境。

本书还为其姐妹篇*Analytic Combinatorics*奠定了基础，那本书以更广的视野对本书涉及的材料进行一般的处理，并开发了高级方法和模型，这些方法和模型不仅在算法平均情形的分析中，而且在组合数学中，均可作为新的研究基础。对于阅读那本姐妹篇，需要更高水平的数学基础，相当于高年级本科生或低年级研究生所应具备的水平。当然，认真学习本书也足以具备这些基础。毫无疑问，使得本书充分有趣始终是我们的目的，读者将会惊喜地发现本书涉及更先进的材料。

如何使用本书。本书的读者在离散数学和计算机科学方面的基础很可能有相当大的差异。因此，了解书中的基本结构很有必要：全书总共八章，先是导论性的一章，接着的三章侧重于数学方法，最后四章集中讨论算法分析的应用：

引论

1. 算法分析概述

离散数学方法

2. 递归关系
3. 生成函数
4. 渐近逼近

算法与组合结构

5. 树
6. 排列
7. 串与trie树
8. 字与映射

第1章将书中材料恰当地进行概括，并帮助所有读者理解本书的基本目标以及其余各章在满足这些目标方面所担当的角色。第2章至第4章更侧重于数学，涉及离散数学的一些方法，

主要讨论一些基本概念和方法。第5章到第8章侧重于计算机科学，包含组合结构的一些性质，与基本算法的关系，以及若干解析结果。

虽然本书是自完备的，但是根据教师经验和学生的基础，可以有所侧重。一种方法是更偏重于数学，重点讲解本书前半部分的定理和证明，并介绍第5章到第8章的应用。另一种方法是更偏重于计算机科学，扼要地讨论第2章到第4章中的主要数学工具，而着重讨论本书第二部分的算法题材。不过，我们主要的意图是大部分的学生通过细心地通读本书，能够同时学到数学和计算机科学两个领域中新的东西。

具有较强计算机科学基础的学生可能已经见过本书后半部分的许多算法和数据结构，但在开始的时候他们的数学知识并不多；具有较强数学背景的学生很可能发现数学材料很熟悉，但算法和数据结构或许有些生疏。包括本书所有内容的课程可以帮助这两类学生根据他们已有的知识来填补知识缺口。

在每章末尾列出的参考文献以及散布在文中的数百道习题是对教材的补充，它们将鼓励读者更深刻地考虑书中的内容并考查原始的资料。

我们教授本书的经验已经证明，教师能够经常通过基于计算的实验课和家庭作业来补充讲义和阅读材料。这里所涵盖的内容对于学生使用诸如Mathematica或MAPLE这样的符号处理系统来深入研习专门知识是一个理想的构架。通过与经验研究比较而使数学研究得以确认的经验对于许多学生而言可能是非常有价值的。

致谢。我们非常感谢INRIA、普林斯顿大学和国家科学基金会，他们对本书提供了主要的支持。其他的支持由布朗大学、European Community（Alcom Project）、Institute for Defence Analyses、Ministère de la Recherche et de la Technologie、斯坦福大学、Université Libre de Bruxelles以及Xerox Palo Alto Research Center提供。本书耗费多年时间撰著，因此提供支持的个人和组织的名单很长，在此不能一一列出，深为抱歉。

Don Knuth对我们工作的影响极为重要，这从教材中明显可以看出。

普林斯顿、巴黎以及普罗维登斯大学的学生们这些年来对利用这些材料的早期版本授课的课程回馈了有益的信息。我们还要感谢Philippe Dumas、Mordecai Golin和他的学生San Kuen Chan、Ka Po Lam、Ngok Hing Leung、Derek Ka-Cheong Lueng和King Shan Lui，以及三位不具名评论家对手稿的详细评论。

R. S.

Ph. F.

1995年9月，Corfu

记号解释

$\lfloor x \rfloor$ **下取整函数**

小于或等于x的最大整数

$\lceil x \rceil$ **上取整函数**

大于或等于x的最小整数

$\{x\}$ **小数部分**

$x - \lfloor x \rfloor$

$\lg N$ **以2为底的对数**

$\log_2 N$

$\ln N$ **自然对数**

$\log_e N$

$\binom{n}{k}$ **二项式系数**

从n项中选择k项的方法数

$\begin{bmatrix} n \\ k \end{bmatrix}$ **第一类Stirling数**

n个元素的具有k个圈的排列数

$\begin{Bmatrix} n \\ k \end{Bmatrix}$ **第二类Stirling数**

将n个元素划分成k个非空子集的方法数

ϕ **黄金分割**

$(1+\sqrt{5})/2 = 1.618\ 03\cdots$

γ **欧拉常数**

σ **Stirling函数**

$\sqrt{2\pi} = 2.506\ 62\cdots$

目　录

出版者的话
专家指导委员会
译者序
序
前言
记号解释

第1章　算法分析概述 ……1
1.1　为什么要对算法进行分析……1
1.2　计算复杂性……2
1.3　算法分析的过程……6
1.4　平均情形分析……7
1.5　例：快速排序的分析……8
1.6　渐近逼近 ……13
1.7　分布 ……14
1.8　概率算法 ……16
参考文献 ……18
第2章　递归关系 ……21
2.1　基本性质 ……22
2.2　一阶递归 ……24
2.3　非线性一阶递归 ……26
2.4　高阶递归 ……28
2.5　求解递归的方法 ……32
2.6　二分分治递归和二进制数 ……37
2.7　一般的分治递归 ……43
参考文献 ……48
第3章　生成函数 ……51
3.1　常规生成函数 ……51
3.2　指数生成函数 ……55
3.3　利用生成函数求解递归 ……57
3.4　生成函数的展开 ……63
3.5　利用生成函数进行变换 ……65
3.6　关于生成函数的函数方程 ……67
3.7　利用OGF求解三数中值Quicksort递归 …69
3.8　利用生成函数的计数 ……71
3.9　符号方法 ……74
3.10　拉格朗日反演 ……80
3.11　概率生成函数……82
3.12　二元生成函数 ……84
3.13　特殊函数 ……91
参考文献 ……94
第4章　渐近逼近 ……97
4.1　有关渐近逼近的记号 ……98
4.2　渐近展开式 ……102
4.3　渐近展开式的操作 ……107
4.4　有限和的渐近逼近……112
4.5　欧拉-麦克劳林求和 ……114
4.6　二元渐近性……119
4.7　拉普拉斯方法 ……128
4.8　算法分析中的“正态”例 ……131
4.9　算法分析中的“泊松”例 ……135
4.10　生成函数的渐近性 ……136
参考文献……140
第5章　树 ……141
5.1　二叉树 ……141
5.2　树和森林 ……143
5.3　树的性质 ……145
5.4　树的算法 ……147
5.5　二叉查找树 ……150
5.6　Catalan树中的平均路径长 ……153
5.7　二叉查找树中的路径长 ……156
5.8　随机树的可加参数 ……159
5.9　高 ……162
5.10　树性质平均情形结果的小结 ……167
5.11　树和二叉树的表示 ……168
5.12　无序树 ……172
5.13　标号树 ……178
5.14　其他类型的树 ……180
参考文献……186

第6章 排列 ……189
6.1 排列的基本性质 ……190
6.2 排列的算法 ……194
6.3 排列的表示法 ……196
6.4 计数问题 ……200
6.5 利用CGF分析排列的性质……204
6.6 逆序与插入排序……211
6.7 左向右最小值与选择排序 ……216
6.8 圈与原位排列 ……220
6.9 极值参数 ……223
参考文献……226
第7章 串与trie树 ……229
7.1 串查找 ……230
7.2 位串的组合性质 ……232
7.3 规则表达式 ……239
7.4 有限状态自动机与Knuth-Morris-Pratt算法……242
7.5 上下文无关语法 ……244
7.6 trie树 ……248
7.7 trie算法 ……251
7.8 trie树的组合性质 ……254
7.9 更大的字母表 ……258
参考文献……259
第8章 字与映射 ……263
8.1 使用分离链接的散列 ……263
8.2 字的基本性质 ……265
8.3 生日悖论与赠券收藏家问题 ……269
8.4 占有约束与极值参数 ……274
8.5 占有分布 ……278
8.6 开放定址散列法 ……284
8.7 映射 ……289
8.8 整数因子分解与映射 ……297
参考文献……301
索引 ……303

第1章 算法分析概述

计算机算法性质的数学研究已经扩展到一个颇为广大的范畴，从一般复杂性研究直到特定解析结果的获得。在这一章，我们将讨论研究算法的各种手段中具有代表性的例子。同时，这些例子使我们得以介绍来自基本的和具有代表性领域即排序算法研究的各种经典的结果。

首先，我们将要考虑算法分析的一般动机，以及算法性能特征的各种研究方法之间的关系。其次，讨论计算复杂性，并把一个“最优的”排序算法，即归并排序（Mergesort）作为例子来考虑。之后，再对具有重要使用价值的排序算法的全面分析的主要方面进行考查，这个算法就是快速排序（Quicksort）。它包括研究对基本快速排序算法的各种改进，同时通过一些例子阐明，我们的分析是如何帮助人们调整参数以改进算法性能的。

本章将把我们的研究领域纳入到一些相关领域之中，并为本书其余部分的研究建立一个平台。在第2章到第4章，我们计划介绍基本的数学概念，这些概念是进行基本算法分析所必需的。第5章到第8章则讨论基本算法和数据结构的基本组合学性质。由于在计算机科学中使用的基本方法和经典数学分析之间存在着紧密的联系，因此我们在本书中同时讨论来自这两个方面的某些介绍性的材料。

1.1 为什么要对算法进行分析

关于这个基本的问题，有几种不同的答案，它们依赖于算法的环境：算法的预期使用；从实用性和理论的立场上看，一个算法与其他算法相比的价值；分析的困难程度和所要求的答案的精确性。

分析算法的最直接的原因是找出它的特性，以便估计它对各种应用的适用程度，或在相同的应用中与其他算法进行比较。我们注意的特征最常见的是时间和空间这样一些基本的资源，尤其是时间。简单地说，我们想要知道在一台特定的计算机上实现一个特定算法要运行多长时间，以及需要多少空间。一般说来，分析是相对独立于特定实现方法的——我们侧重于得到算法基本性能的相关结果，这些算法能够用来算出在具体的机器上对于实际资源需求的精确估计。

实践中，在算法和它的实现方法的特性之间达到完全独立做起来恐怕不容易。算法实现的质量、编译器的特性、机器的结构以及编程环境的其他主要方面，都将对性能产生剧烈的影响。我们必须认识到这些影响，这样才能保证分析的结果是有用的。另一方面，在某些情况下，算法分析可以帮助识别那些能够充分利用编程环境的方法。

偶尔有些性质很重要，但它们并不是时间和空间，因此我们分析的重点也要相应地改变。例如，驱动电路布线装置的算法可能用以确定所用线的总量，数值问题的算法可能要确定它提供的答案的准确程度。再有，有时候在分析中使用成倍的资源是可取的。例如，使用大量内存的算法可能比使用很少内存也能得到的算法所耗费的时间少得多。实际上，进行仔细分析的一个主要的动机就是提供准确的信息，以帮助在这样一些情况下作出正确的权衡决策。

术语*算法分析*（analysis of algorithms）一直用于描述两种相当不同的方法，这两种方法把计算机程序的研究置于科学的基础之上，现在我们对它们依次进行讨论。

第一种方法由Aho、Hopcroft和Ullman[2]所提倡，重点研究确定算法最坏情形性能的增长（“上界”）。这种分析的主要目标就是确定在下述意义下哪些算法是最优的：它们对同一问题的任何算法最坏情形的性能可以证明一个匹配的“下界”。有时称这种类型的分析为*计算复杂性*（computational complexity），不过，这个术语留给对问题、算法、语言和机器之间关系的一般研究或许更妥当。

2

处理算法分析的第二种方法由Knuth[13][14][15][17]提出，主要通过确定最佳情形、最坏情形和平均情形的性能来精确地刻画算法的性能，使用的是一套可以精化的方法，在需要的时候能够得出更加精确的答案。这种分析的主要目标是当运行在特定的计算机上的时候，能够准确地预测特定算法的性能特征。

我们可以把这两种方法看作是有效算法的设计和分析的必需阶段。当面对求解一个新问题的新算法的时候，我们的兴趣在于形成这样一种粗略的认识：新算法预期能有多好，对于同一问题它比其他的算法如何。计算复杂性研究能够提供这种认识。然而，在这样一种粗略的复杂性分析中丧失如此多的精度，以致提供不了使我们预测算法具体实现的性能的特定信息，也提供不了将一个算法与另外的算法进行正确比较所需要的特定信息。为了能够进行这些工作，我们需要关于算法实现、所使用的计算机，以及本书我们将要看到的，由算法使用的一些结构的数学性质的全部细节。计算复杂性可以看成是形成更精练、更准确分析的正在进行的过程的第一步；我们更愿意使用术语*算法分析*来表示整个的过程，其目标是提供所需任意精度的解决方案。

对算法的分析能够帮助我们对算法更深入地理解，并能够提出有根有据的改进。算法越复杂，分析越困难。在分析过程中，算法趋向于变得更短小、更简单和更讲究。尤为重要的是，正确分析所需要的仔细全面的审查常常使得算法更好和更有效地实现。分析要求对算法全面的理解，这种理解比只是简单得出行得通的实现方法远为全面和深刻。实际上，当解析的结果与经验研究一致的时候，我们将会确信算法的合理性以及分析过程的正确性。

有些算法是值得分析的，因为它们的分析能够增强现有的数学工具集。这样一些算法也许实用价值有限，但可能具有类似于实用价值的算法的性质，使得理解它们能够有助于理解未来更重要的方法。

3

另一方面，许多算法（有些有很强的实用价值，有些则很少甚至没有实用价值）具有复杂的性能结构，这种结构具有独立的数学意义。其算法分析产生的组合学问题的活跃成分导致一些具有挑战性的重要的数学问题，这些问题扩展了经典组合学的范畴，有助于揭示计算机程序的性质。

为了清楚地表达这些思想，下面首先从计算复杂性的角度，然后再以算法分析的观点，详细考虑某些经典的结果。作为阐述不同观点的实际例子，我们考查一些*排序算法*（sorting algorithms），它们以数值顺序、字母顺序、或其他顺序重新排列一组数据。这是一个重要的实际问题，该问题仍然是广泛研究的目标，因为排序程序在许多应用中起着核心的作用。

1.2 计算复杂性

计算复杂性的主要目标是把算法按照它们的性能特征进行分类。第一步是建立表达结果的相应数学概念：

定义 *给定函数$f(N)$，则*

$O(f(N))$表示所有$g(N)$的集合，其中$g(N)$满足当$N\to\infty$时，$|g(N)/f(N)|$上有界。

$\Omega(f(N))$ 表示所有$g(N)$的集合，其中$g(N)$满足当$N\to\infty$时，$|g(N)/f(N)|$以一个（严格的）正数为其下界。

$\Theta(f(N))$ 表示所有$g(N)$的集合，其中$g(N)$满足当$N\to\infty$时，$|g(N)/f(N)|$既上有界又下有界。

这些概念取自经典数论，由Knuth在1976年的一篇论文[16]的算法分析中提倡使用。在对算法性能的界的数学表述方面，这些记号得到了广泛的使用。

大O记法提供一种表达上界的方法；大Ω记法提供一种表达下界的方法；而大Θ记法则提供一种表达同时有上界又有下界的方法。大O记法在数学中最普通的用法是表示渐近近似性质，我们将在第4章对它进行详细的考察。这个记号在算法分析中也比其余两个记号使用得普遍，在描述算法运行时间的表达式中，它一般表示相对小“错误”的项。记号Ω和Θ直接与计算复杂性相关，不过类似的记号在其他的应用领域也在使用（见[16]）。 4

对于复杂性研究，这些记号的重要性在于，它们实现的细节被忽略的常数因子所隐藏。由于常数因子被忽略，因此使用这些记法的副作用在于，使用它们得到的数学结果要比寻找更精确的答案简单。例如，“自然”对数 $\ln N \equiv \log_e N$和“二进制”对数 $\lg N \equiv \log_2 N$经常出现，但是，它们之间的关系是相差一个常数因子，因此，在复杂性分析中我们可以把二者都看成是$O(\log N)$。

习题1.1 证明$f(N) = N\lg N + O(N)$意味着$f(N) = \Theta(N\log N)$。

作为使用这些记号研究算法性能特征的一个例子，我们考虑将数组`a[1..N]`中的数集排序的方法。输入是数组中的数，顺序是任意的并且是未知的；输出为该数组中同样这些数，但以递升的顺序排列。这是一个得到深入研究的基本问题：我们将考虑求解该问题的一个算法，然后证明，从精确的技术意义上看，算法是“最优的”。

首先，我们要证明，使用著名的递归算法，即归并排序（Mergesort）能够有效地解决这个排序问题。归并排序以及本书中处理的几乎所有的算法均在Sedgewick[19]中有详细的讨论，因此我们这里只给出简要的描述。对于各种算法、实现和应用的进一步细节有兴趣的读者，可以参考Cormen、Leiserson和Rivest[5]，Gonnet和Baeza-Yates[8]、Knuth[13][14][15]、Sedgewick[19]等人的著作以及其他一些资料。

归并排序将数组从中间分开，（递归地）将两部分排序，然后将所得到的已排序的子文件合并到一起从而得到排序结果。为此，该算法使用了两个辅助数组b和c，如程序1.1所示。通过`mergesort(1, N)`调用该过程将数组`a[1..N]`排序。在递归调用之后，排序的前一半`a[1..m]`被拷贝到辅助数组`b[1..m-1+1]`，而排序的后一半`a[m+1..r]`被拷贝到第2个辅助数组`c[1..r-m]`。然后，通过把元素`b[i]`和`c[j]`中的较小者移到`a[k]`，相应地使i或j增1来完成合并。程序用到“警戒标记”max以帮助完成当一个辅助数组已经没有元素时将另一个辅助数组剩下的部分移回到a，此处假设max比所有的元素都大。 5

习题1.2 实现一种归并排序，该排序将数组分成3个相等的部分，将这3部分排序，并进行3-路合并。凭经验比较该算法与标准的归并排序的运行时间。

归并排序是著名的*分治算法*（divide-and-conquer）设计范例的原型，其中，一个问题的解决是通过（递归地）求解更小的一些子问题并利用这些子问题的解来求解原问题而得到的。在本书中，我们将分析许多这样的算法。更一般地，我们将看到象归并排序这样的算法的递归结构如何直接导致它们性能特征的数学描述。

在当前环境下，归并排序是重要的方法，因为它能够保证不亚于任何排序算法的效率。为了更精确地表达这个结论，我们开始分析归并排序运行时间的控制因素，即它所用到的比

较次数。

程序1.1 归并排序（Mergesort）

```
procedure mergesort(l, r:  integer);
  var i, j, k, m:  integer;
  begin
  if r-l > 0 then
    begin
      m := (r+l) div 2;
      mergesort(l, m); mergesort(m+1, r);
      for i := 1 to m-l+1 do b[i] := a[l+i-1];
      for j := m+1 to r do c[j-m] := a[j];
      i := 1; j := 1; b[m-l+2]:= max; c[r-m+1] := max;
      for k := l to r do
        if b[i] < c[j]
          then begin a[k] := b[i]; i := i+1 end
          else begin a[k] := c[j]; j := j+1 end;
    end;
  end;
```

定理1.1(*归并排序*) *为将N个元素的数组排序，归并排序使用$N\lg N+O(N)$次比较。*

证明 若C_N为上面程序将N个元素排序所用的比较次数，则排序前一半元素的比较次数就是$C_{\lfloor N/2\rfloor}$，而排序后一半元素的比较次数则是$C_{\lceil N/2\rceil}$，合并的比较次数是N（对于下标k的每个值有一次比较）。换句话说，归并排序的比较次数由下面的递推关系精确描述

$$C_N = C_{\lfloor N/2\rfloor} + C_{\lceil N/2\rceil} + N \quad (N \geqslant 2, C_1 = 0) \tag{1-1}$$

为得到这个递推关系的解的性质的启示，我们考虑当N是2的幂时的情况：

$$C_{2^n} = 2C_{2^{n-1}} + 2^n \qquad (n \geqslant 1, C_1 = 0)$$

用2^n除方程的两边，我们发现

$$\frac{C_{2^n}}{2^n} = \frac{C_{2^{n-1}}}{2^{n-1}} + 1 = \frac{C_{2^{n-2}}}{2^{n-2}} + 2 = \frac{C_{2^{n-3}}}{2^{n-3}} + 3 = \cdots = \frac{C_{2^0}}{2^0} + n = n$$

这就证明了，当$N = 2^n$时，$C_N = N\lg N$；对于一般的N，定理可以通过从式（1-1）归纳证明。实际上，准确解是相当复杂的，它依赖于数N的二进制表示的性质。在第2章我们将详细讨论如何求解这样的递推关系。 ■

习题1.3 求出描述量$C_{N+1}-C_N$的递推关系，并利用所得到的关系证明

$$C_N = \sum_{1<k<N}\left(\lfloor \lg k\rfloor + 2\right)$$

6~7

习题1.4 证明$C_N = N\lceil \lg N\rceil + N - 2^{\lceil \lg N\rceil}$。

习题1.5 分析在习题1.2中提出的3路归并排序所使用的比较的次数。

对于大多数计算机，在上面程序中所用到的基本操作的相对开销将相差一个常数因子，因为它们均为一个基本指令周期的整数倍。再有，程序总的运行时间将在比较次数的常数倍范围之内，因此归并排序的运行时间将在$N\log N$的常数倍范围之内。

从计算复杂性的观点来看，归并排序证实$N\lg N$是排序问题固有困难性的一个“上界”：

存在一种算法，能够以与$N\log N$成比例的时间将任意

N-元素文件排序

这个结论的全部证明需要根据相关的操作和花费的时间，审慎地构造所用的计算机模型，而结果却是在相当宽松的假设下成立的。我们说排序的时间复杂度为$O(N\log N)$。

习题1.6 假设归并排序的运行时间是$cN\lg N + dN$，其中c和d是与机器相关的常数。证明，如果我们在特定的计算机上实现它的程序并对某个值N观察运行时间t_N，那么对于$2N$我们可以准确地估计运行时间为$2t_N(1+1/\lg N)$，而与机器无关。

习题1.7 在一台或更多的计算机上实现归并排序，观察$N = 50\ 000$时的运行时间，并用前面的习题预报$N = 100\ 000$时的运行时间。然后，观察$N = 100\ 000$时的运行时间并计算预报的百分比精度。

如上所实现的归并排序的运行时间只依赖于被排序的数组中的元素个数，而不依赖于它们排列的方式。对于许多其他的排序方法，其运行时间实际上可以作为输入数据初始排序状态的函数在变化。一般地，在复杂性研究中，我们最感兴趣的是最坏情形的性能，因为这可以对算法的性能特征提供一种保证，而不管输入的是什么；在对一些特定算法的分析中，我们最感兴趣的是平均情形的性能，因为它可以提供一种方式来预报算法对“典型”输入的性能。

我们总是在寻找更好的算法，自然产生一个问题：是否存在一种排序算法，它比归并排序有更好的渐进性能？下述来自计算复杂性的经典结果表明，本质上不存在这样的排序算法。 8

定理1.2 (排序的复杂度) 任何基于比较的排序程序必然至少用到$\lceil \lg N! \rceil > N\lg N - N/(\ln 2)$次比较。

证明 这个事实的全部证明可以在[2]或[15]中找到。直观地看，由每次比较能够将所考虑元素的可能排列数最多减少一半的事实可以推出这个结果。由于在排序前存在$N!$个可能的排列，而我们的目标就是在排序后得到其中的一个排列，因此比较的次数必然至少为使得$N!$在减到小于1之前始终被2整除的除法次权，即$\lceil \lg N! \rceil$。由Stirling关于阶乘函数的近似公式（见定理4.3的第2个推论）立即得到该定理的结论。 ■

从复杂度的观点来看，这个结果证实$N\log N$是排序问题固有困难性的一个“下界”：

- 所有基于比较的排序算法都需要以与$N\log N$成比例的时间将一个N-元素的输入文件排序

这是关于一整类算法的一般的综述，我们说排序方法的时间复杂度是$\Omega(N\log N)$。这个结论很重要，因为它与定理1.1的上界吻合，从而证明归并排序在没有算法能够具有更好的渐近运行时间的意义上是最优的，我们说排序的时间复杂度是$\Theta(N\log N)$。从计算复杂性的观点看，这已经完成了排序“问题”的“解”：匹配的上界和下界已经被证明。

再次指出，这些结果在相当一般的假设下是成立的，虽然它们或许不像看起来那样一般。例如，对于那些并不使用比较的排序算法，这个结论等于什么也没说。事实上，存在一些排序方法是以平均线性时间运行的，这些算法是基于一些寻址计算技巧的（如在第8章讨论的那些算法）。

习题1.8 设已知数组a[1··N]中每一项均从两个不同的值取其中的一个值。对这样的数组给出一种排序方法，其花费的时间与N成比例。

习题1.9 当从3个不同的值中取值时，给出上一道习题的答案。

在定理1.1和定理1.2的证明中,我们忽略了关于计算机和程序的相应模型的细节。计算复杂性的本质就是研究完善的模型，在这样的模型范围内可以评估重要问题的固有困难，并得以进一步研究“有效的”算法，这些算法体现了匹配那些下界的上界。对于许多重要问题领域，在渐近最坏情形性能的下界和上界之间仍然存在显著的差距。计算复杂性提供在深入研究这些问题的新算法方面的指导。我们需要那些能够降低已知上界的算法，然而寻找性能优于已知下界的算法却很难取得成效（除非寻找那种破坏模型的条件的算法或许可以一试，可 9

是模型条件又是已知下界赖以存在的基础！）。

因此计算复杂性提供一种方法，能够按照算法的渐近性能将算法分类。然而正是这种近似（“抛弃常数因子”）分析的过程常常限制了我们准确预报特定算法性能特征的能力，虽然近似分析扩展了复杂性成果的适用性。更重要的是，复杂性通常是基于最坏情形的分析，这可能过于悲观，在预报算法具体性能上不如平均情形分析实用。对于诸如归并排序这样一些最优算法这么做不是很贴切，但是我们将会看到，平均情形分析却能够帮助我们发现，有时候非最优算法在实践中更快。复杂性可以帮助我们鉴别优秀的算法，而此时我们更感兴趣的是精化分析以便能够更聪明地比较并改进它们。为此，我们需要对所用特定计算机性能特征的准确认识，以及精确确定期望的指令执行频率的数学技巧。在本书中，我们专注于这样的技巧。

1.3 算法分析的过程

虽然上面对排序和归并排序的分析论证了排序问题的固有“困难”，但是，仍然存在与排序（和归并排序）相关的许多重要问题还根本没有涉及。归并排序的实现方法在特定的计算机上运行预计可能花费多长时间？它的运行时间如何与其他$O(N \log N)$方法相比？（这样的方法有很多。）怎样将它与那些平均很快但在最坏情形或许没那么快的排序方法进行比较？如何将它与那些不是基于元素间比较的排序方法进行比较？要回答这样一些问题，就需要更详细的分析。本节我们简要描述如何进行这样的分析，然后给出对另一种排序算法，即快速排序的分析的应用实例。

10

为了分析算法，首先我们必须识别那些根本性的重要资源，使得详细的分析可以正常地、有的放矢地进行。由于运行时间在这里是最相关的资源，因此，我们用研究运行时间的术语来描述算法的过程。一个算法的运行时间的完整分析包括下述几个步骤：

- 将算法完全实现。
- 确定对于每个基本操作所需要的时间。
- 识别那些能够用来描述基本操作执行频率的未知的量。
- 研究出一种对于输入到程序的数据的实际模型。
- 假设为输入建立了模型，分析其中一些未知的量。
- 将每个操作的频率乘以操作的时间，然后把所有的乘积相加，计算出总的运行时间。

分析的第1步是要在特定的计算机上审慎地实现算法。我们保留用术语*程序*（program）来描述这样一种实现，于是一种算法对应许多的程序。这种实现不仅提供了具体的研究目标，而且还能够给出有用的经验数据以帮助检验我们的分析。算法的实现应该设计得可以有效地利用资源，而在过程中过早过分地强调效率却是不对的。事实上，分析的主要应用是对更好的实现提供有见识的指导。

下一步，是估计构成程序的每一种指令所需要的时间。通常这可以非常精确地完成，不过它紧密地依赖所使用计算机的特性。另一种方法是对于小量的输入直接运行程序来“估计”一些常量的值，或者总体上间接地进行，正如习题1.6中所描述的那样。在本书中，我们不详细考虑这个过程，而是把精力集中在分析中“与机器无关”的部分。

为了确定程序总的运行时间，有必要研究程序的分支结构，用未知的数学量表示程序构成指令的执行频率。如果这些量的值已知，那么我们可以直接把每种构成指令的时间需求乘以频率，然后将这些乘积相加，从而得到整个程序的运行时间。作为大多数编程环境一部分的靠模器（profilers）可以简化这项工作。在分析的第1阶段，我们专注于具有大频率值的量或对应大开销的量，原则上分析能够被精化以得到足够详细的解答。当上下文允许时，我们

11

常常把算法的“开销”作为“所讨论的量的值”的简称。

下一步是为程序的输入建立模型，以形成对指令频率的数学分析的基础。那些未知的频率值依赖于对算法的输入：输入的大小（我们通常把它写成N），在正常情况下，是用来表示结果的主要参数，不过那些输入数据项的顺序或值通常也影响着运行时间。这里的“建立模型”指的是对向算法典型的输入的精确描述。例如对于排序算法，通常方便的做法是假设输入是随机排序的而且是互异的，虽然即使当输入数据不是互异时程序通常也是正常运行的。对于排序算法，另一种可能是假设输入是取自相对大的范围的一些随机数，可以证明这两种模型几乎是等价的。最常见的，是我们使用最简单的“随机”输入的现有模型，这种模型常常是现实的。对于同一个算法可以使用几种不同的模型：一种模型的选用可能是使得分析尽可能的简单；而另一种模型可能更好地反映程序将被使用的具体的情况。

最后一步是分析那些未知的量，假设输入的模型已经建立。对于平均情形，这些量可以逐个地进行分析，然后用指令的次数乘以相应的平均值并相加，最后得到整个程序的运行时间。对于最坏情形的分析，得到整个程序的准确结果通常很困难，因此，常常用指令的次数乘以各个量的最坏情形的值，然后相加而得到算法的一个上界。

1.4 平均情形分析

在本书中，我们主要讨论适用于得到算法平均情形性能结果的一些方法。当然，数学方法一般说来适用于解决与算法性能有关的各种问题，不过我们最感兴趣的还是对于一组“典型”输入的资源使用情况能够给出精确的表述。

12

初等概率论有许多不同的方法计算量的平均值。当这些方法紧密相关时，对我们来说清楚地识别两种计算平均值的不同方法将是方便的。

分布的方法。令$\prod_N$是大小为N的各种可能的输入（数据）的组数，$\prod_{Nk}$是使算法产生开销k的、大小为N的输入组数，于是$\prod_N = \sum_k \prod_{Nk}$。此时，开销是$k$的概率为$\prod_{Nk}/\prod_N$，开销的数学期望为

$$\frac{1}{\Pi_N}\sum_k k\Pi_{Nk}$$

算法的分析依赖于“计数”。存在多少大小为N的输入以及有多少大小为N的输入使得算法开销是k？这些就是计算开销为k的概率的具体步骤，因此这种方法恐怕是初等概率论中最直接的方法。

累积的方法。令Σ_N为算法对所有大小是N的输入的总（或累积的）开销（即，$\Sigma_N = \Sigma_k k\Pi_{Nk}$，但问题在于不必用这种方法计算$\Sigma_N$）。此时的平均开销就是$\Sigma_N / \prod_N$。算法的分析依赖于较小的特定计数问题：对于所有的输入，算法总的开销是多少？我们将使用一般的工具，这些工具使得这种方法颇具魅力。

分布的方法给出完备的信息，它可以用来直接计算标准差和概率论中其他的矩。我们将看到，当使用其他方法时，间接（常常是更简单）的方法也能够计算这些矩。在本书中，虽然我们倾向于使用累积的方法，它最终将使我们得以用基本数据结构的组合学性质考虑算法的分析，但是我们还是两种方法都考虑使用。

许多算法解决问题是通过递归地求解一些小的子问题来完成的，因此它们应该服从平均开销或总开销必须满足的递推关系的结论。从算法直接得到递推关系常常是一种自然的处理方式，下一节的例子解释了这个过程。

不管平均情形的结果是怎么得到的，我们还是对平均情形的结果更关注，因为在随机输入是合理模型的大多数情形下，精确的分析可以帮助我们:

13

- 对于相同的作业比较不同的算法。
- 对一些特殊的应用，预报时间和空间需求。
- 比较将要运行同一算法的不同的计算机。

可以把平均情形的结果与经验数据比较来检验算法的实现、模型和分析。最终的目标是得到足够的信心使得用平均情形的结果来预报在一些特殊应用的各种环境下算法将如何表现。例如，我们可以算出一种新的机器结构对某个重要算法性能的可能的影响，通过分析这往往是能够做到的，也许在新的结构出现之前就能做到。

另一个重要的例子是当算法本身含有可以调整的参数的情形：分析能够指出参数取什么值最优。

1.5 例：快速排序的分析

为了叙述这套方法，我们在这里描述一个重要的特殊算法——Quicksort排序方法的一些结果。该方法在1962年由C.A.R.Hoare发现，他的论文[12]是算法分析中一篇较早的和深具洞察力的例子。方法的分析在Sedgewick[20]中也有非常详尽的论述（还可见于[22]），我们在这里仅作重点阐释。研究这种分析是值得的，这不仅因为排序方法被广泛使用并且其解析结果直接与实际相关，而且还因为分析本身说明许多问题，这些问题我们在本书稍后将会遇到。特别是，同样的分析事实上对树结构基本性质的研究也是适用的，而树结构具有广泛的重要性和适用性。更一般地，对Quicksort的分析指出我们如何着手分析一类广泛的递归程序。

程序1.2是Quicksort排序算法的Pascal语言实现。这是一个递归程序，它通过把数组a[l:r]划分成两个独立（更小）的部分，然后将这两部分分别排序，从而达到将数组a[l:r]排序的目的。当遇到r<l的空的子数组时递归终止。实际上，大小为1（即r = l）的子数组也算是已经“排序”了，因此在这种情形下也就没有什么需要做的了。我们可以通过把程序if语句中的r ≥ l改为r > l体现这一点：下面考查这种改进的推广。这种类型的改变初看起来似乎是微不足道的，但是，我们将看到，递归的本质却是保证程序将被用于大量的小文件，而使用这种类型的简单改进可以达到实质上的性能提高。

程序1.2 快速排序（Quicksort）

```
procedure quicksort(l, r:  integer);
  var v, t, i, j:  integer;
  begin
  if r >= l then
    begin
      v := a[r]; i := l-1; j := r;
      repeat
        repeat i := i+1 until a[i] >= v;
        repeat j := j-1 until a[j] <= v;
        t := a[i]; a[i] := a[j]; a[j] := t;
      until j <= i;
      a[j] := a[i]; a[i] := a[r]; a[r] := t;
      quicksort(l, i-1);
      quicksort(i+1, r)
    end
  end;
```

划分数组的过程把数组中最后位置上的元素（分割元（partitioning element））放到它应该占据的正确位置上，使得所有较小的元素都在它的前面，而所有较大的元素都在它的后面。

我们使用两个指针完成这种划分，其中一个指针从左扫描数组，另一个则从右进行。左指针逐次增1直到发现比分割元大的元素为止，右指针逐次减1直到发现比分割元小的元素为止。然后将这两个元素交换，过程继续进行直到两个指针相遇为止，此时的位置就是分割元被置入的位置。quicksort(1, N)调用将完成数组的排序，我们假设a[0]的值小于数组中任何其他元素的值（以免第一个分割元碰巧是最小的元素）。

存在几种方法实现刚刚描述的一般的递归策略、上面描述的实现方法取自Sedgewick[19]（也见于[20]）。为了进行讨论，假设数组a包含的是随机排列的互异的数。这是最便于分析的模型，在允许元素相等的或许是更现实的模型下研究这个程序也是可能的[21]。当输入的数组包含互异的数时，指针总是以j=i-1交叉，但就在这种现象被检测到之前进行了一次“额外”的交换。在外层repeat之后的三条赋值语句等价于撤销这次交换，然后a[i]和a[r]交换，将分割元安排就位。 14 ~ 15

实现算法之后，分析的第一步是估计该程序单个指令的资源需求。这对于任何特定的计算机来说都很简单，我们将忽略其细节。例如，“内层循环”指令

```
repeat i:=i+1 until a[i]>=v
```

在典型的计算机上可能被译成如下的汇编语言指令

```
LOOP    INC   I, 1          # increment  i
        CMP   V, A[I]       # compare V with A[i]
        BL    LOOP          # branch  if  less
```

该循环的一次迭代可能需要4个时间单位（每次内存引用一个）。

分析的下一步是给程序中指令的执行频率指定变量名。正常情况下，只涉及少数的真变量。所有指令的执行频率可以用这些少数变量表示。再有，我们希望把这些变量和算法本身而不是和任何特定的程序联系上。对于Quicksort，有三个自然的量被涉及：

A – 划分阶段数

B – 交换次数

C – 比较次数

在典型的计算机上，总的运行时间大约为

$$4C + 11B + 35A \tag{1-2}$$

这里的3个系数的精确值依赖于由编译器产生的汇编语言程序以及所用计算机的特性，上面给出的值是典型的情况。对在同一台计算机上实现的不同算法进行比较时，这样的表达式是相当有用的。事实上，即使Mergesort是“最优”的，Quicksort仍具有重要实际价值的原因在于，其每次比较的开销（C的系数）很可能比Mergesort明显地低，这就导致在一般的实际应用中Quicksort的运行时间显著地降低。 16

定理1.3 (Quicksort) *为将N个随机顺序的互异元素的数组排序，Quicksort平均使用*

$$N \quad \text{个划分阶段}$$

$$2(N+1)(H_{N+1}-1) \approx 2N\ln N - 0.846N \quad \text{次比较}$$

$$(N+1)(H_{N+1}-5/2)/3 + 1/2 \approx 0.333N\ln N - 1.346N \quad \text{次交换}$$

证明 首先，我们注意这里精确的答案由调和数

$$H_N = \sum_{1 \le k \le N} 1/k$$

表示，它是我们在算法分析中很可能要遇到的许多著名的“特殊”数列中的第一个。

和Mergesort一样，Quicksort的分析涉及确定和求解递推关系，这些关系直接反映上面给出的程序的递归特性。不过此时这些递归必须基于关于输入的概率表述。如果C_N是将N个元素排序的平均比较次数，那么我们有$C_0 = 0$以及

$$C_N = N+1+\frac{1}{N}\sum_{1\le j\le N}(C_{j-1}+C_{N-j}) \qquad (N>0) \tag{1-3}$$

为了得到总的平均比较次数，我们把第一次划分阶段的比较次数$(N+1)$加到用于划分后的子文件的比较次数中。当分割元为第j个最大元素（对于每个$1\le j\le N$，它以概率$1/N$出现）时，则划分后这些子文件大小为$j-1$和$N-j$。

现在，分析被简化成数学问题式（1-3），它不依赖于程序或算法的性质。这个递推关系多少要比式（1-1）复杂，因为右边直接依赖于所有前面的值，而不只是一部分的值。尽管如此，式（1-3）的求解并不困难：首先在和的第二部分中把j变成$N-j+1$，得到

17

$$C_N = (N+1)+\frac{2}{N}\sum_{1\le j\le N}C_{j-1} \qquad (N>0)$$

然后乘以N并减去对于$N-1$时的同一个公式，消掉求和项后得到：

$$NC_N-(N-1)C_{N-1} = 2N+2C_{N-1} \qquad (N>1)$$

再重新排列各项，得到一个简单的递归

$$NC_N = (N+1)C_{N-1}+2N \qquad (N>1)$$

两边除以$N(N+1)$，得到

$$\frac{C_N}{N+1} = \frac{C_{N-1}}{N}+\frac{2}{N+1} \qquad (N>1)$$

重复代入，最后剩下

$$\frac{C_N}{N+1} = \frac{C_1}{2}+2\sum_{3\le k\le N+1}1/k$$

这就完成了证明，注意$C_1 = 2$。

如上所述，每个元素恰好用于一次划分，因此阶段数总是N。通过对第一次划分阶段的平均交换次数的计算，平均的交换次数可以从这些结果中找出。

定理所叙述的近似结果由著名的调和数近似式$H_N \approx \ln N + 0.57721\cdots$得到。我们将在下面考虑这些近似并在第4章进行详细的讨论。 ■

18

习题1.10 给出Quicksort对N个元素所有$N!$种排列进行排序所使用的总的比较次数的递归。

习题1.11 证明，在划分一个随机排列后得到的两个子文件本身还都是随机排列。然后证明，如果右指针初始化在`j:=r+1`来进行划分，则情况会有不同。

习题1.12 按照上面的步骤求解递推关系

$$A_N = 1+\frac{2}{N}\sum_{1\le j\le N}A_{j-1} \qquad (N>0)$$

习题1.13 证明在第一次划分阶段（在指针交叉之前）平均交换次数为$(N-2)/6$。（因此，根据递归的线性性质，$B_N = \frac{1}{6}C_N-\frac{1}{2}A_N$。）

图1-1指出，定理1.3的解析结果如何与生成随机输入数据到程序并计算所用比较次数后算

出的经验结果进行比较。这些经验结果（对所示N的每个值1000次试验）概括为在中位数上的一点和跨越顶和底四分位点的竖线段（见[23]）；解析结果为平滑曲线，它以对应标准差的灰色区域拟合定理1.3给出的公式。当我们在后面考虑分布时再对它进行讨论。正如所料，图中拟合得非常好。

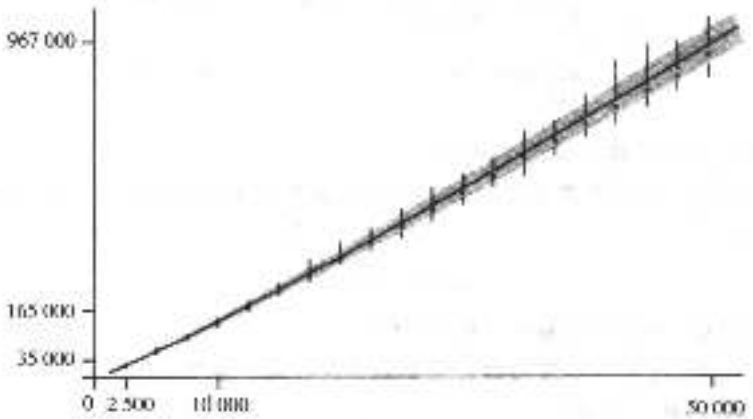


图1-1 Quicksort比较计数：经验结果和解析结果

定理1.3和式（1-2）意味着，对于上面描述的特定机器，Quicksort应该需要大约$11.667N\ln N - 16.810N$步将一个N元素的随机排列排序，而对于其他机器也可以像在式（1-2）和定理1.3之前的讨论那样，通过考查机器的特性得到类似的公式。这样的公式可以用来预报在一台特定机器上Quicksort的运行时间（具有很高的精度），更重要的是，它们可以用来计算和比较算法的各个变量，并提供效率的量化证据。

由于适当注意细节能够可靠处理机器依赖性，因此在本书中通常我们将致力于分析像“比较”和“交换”这样的一般算法相关的量，这不仅使我们把焦点集中在主要的分析技巧上，而且还能够扩展结果的适用性。例如，排序问题一个稍微广泛的应用是考虑把一些记录（record）作为要排序的项，这些记录除排序关键字（key）外还包含一些其他信息，因此访问一个记录（依赖于记录的大小）可能要比进行一次比较（依赖于记录和关键字的相对大小）昂贵得多。此时，从定理1.3可知，Quicksort大约比较关键字$2N\ln N$次，移动记录约$0.667N\ln N$次，而且我们能够计算开销的更精确的估计值或在适当的时候与其他算法进行比较。

Quicksort程序可以以几种方法改进，使它在许多计算环境下成为有力的排序方法。甚至对于一些复杂得多的改进版的快速排序都可以进行完善的分析，而且能够得到平均运行时间的表达式，并紧密地匹配所观察到的经验数据[22]。当然，提出的改进方法越复杂，分析就越复杂，有些改进可以通过扩展上面给出的论证来处理，不过，也有些改进则需要更强大的解析工具。

小的子文件。Quicksort最简单的变种基于下面的观察结果：对于非常小的文件（例如，大小为2的文件可以用一次比较和可能还有一次交换完成排序），该方法排序不是非常有效，因此，对于更小的子文件应该使用更简单的排序方法。下面的习题指出如何扩展上面的分析来研究一种混合算法，其中“插入排序”（见6.6节）用于大小小于M的文件。此时，分析方法可以用来帮助选择参数M的最优值。

习题1.14 当用Quicksort将一个大小为N的随机文件排序时，平均遇到多少大小为2或更小的子文件？

习题1.15 如果把前面Quicksort实现程序的第1行变为

```
if r - 1<M then insertionsort(l, r) else
```

（见6.6节）那么排序N个元素的总比较次数由下面的递归描述

$$C_N=\begin{cases}N+1+\dfrac{1}{N}\sum\limits_{1\leqslant j\leqslant N}(C_{j-1}+C_{N-j}) & (N>M)\\ \dfrac{1}{4}N(N-1) & (N\leqslant M)\end{cases}$$

照定理1.3证明中的做法准确地解出该递归。

习题1.16 在前面习题的答案中忽略小项（那些明显小于N的项），求出函数$f(M)$，使得比较次数近似地为

$$2N\ln N+f(M)N$$

画出函数$f(M)$，并求出使该函数最小化的M的值。

习题1.17 当M变大时，比较次数又从刚刚得到的最小值增加。M必须多大才能使比较次数超过原来（在$M=0$时）的数？

三数中值Quicksort。对Quicksort的自然改进就是使用如下取样：通过选取一个小样本，然后使用它们的中值作为分割元，估计分割元更可能接近文件的中值。例如，如果我们就用3个元素作为样本，那么这种“三数中值”Quicksort所需要的比较次数由下面的递归描述

$$C_N=N+1+\sum_{1\leqslant k\leqslant N}\frac{(N-k)(k-1)}{\binom{N}{3}}(C_{k-1}+C_{N-k})\qquad(N>3)\tag{1-4}$$

其中，$\binom{N}{3}$是二项式系数，即从N项中选取3项的方式的数目。因为第k个最小元是分割元的概率为$(N-k)(k-1)\Big/\binom{N}{3}$（与正常Quicksort的$1/N$相对），所以上式成立。我们希望能够求解这种递归以便能够确定使用多大的样本以及何时切换到插入排序。不过，这样的递归需要比迄今所使用的方法更复杂的技巧。在第2章和第3章中，我们将会看到求解这样一些递归更精确的解的方法，这些方法能够确定像样本大小以及对于小的子文件实施截止操作这样一些参数的最优取值。沿着这些方面广泛的研究已经得到结论：对于一般的实现方法而言，三数中值Quicksort使用从10到20范围的截止点将接近达到最优的性能。

20 ~ 21

基数-交换排序。Quicksort的另一种变化是利用下面的事实：关键字可以看成是二进制串。我们进行划分不是比较文件中的关键字，而是通过把前导位是比特0的关键字置于前导位是比特1的关键字的前面来划分文件的。然后，所得到的子文件以同样的方式使用第2个比特位独立地再行划分，如此等等。这种变化叫做“基数-交换排序”或“基数Quicksort排序”。这种变化怎么与基本算法比较呢？为了回答这个问题，首先我们必须注意，由于随机的比特位组成的关键字基本上不同于随机的排列，因此需要不同的数学模型。这种随机“比特串”模型或许更现实，因为它反映了具体的表示法，但是这两个模型可以被证明大致等价。（这个问题将在第7章更详细地讨论。）使用类似于上面给出的论证可以证明，由这种方法所需要的比特位比较的平均次数由下面的递归描述

$$C_N = N + \frac{1}{2^N}\sum_k \binom{N}{k}(C_k + C_{N-k}) \qquad (N > 1, C_0 = C_1 = 0)$$

实际上这是比上面给出的更难求解的递推关系——我们在第3章将会看到生成函数如何用于把这个关系转变成C_N的显式公式，而在第4章和第7章我们将看到如何得到一个近似解。

这类分析方法适用性的一种局限在于上面所有的递推关系均依赖算法的“保持随机性”特性：如果原始文件是随机顺序的，那么可以证明，划分后的子文件还是随机顺序的。算法的实现者可以不受这样的限制，该算法广泛使用的许多变种也确实不具有这个性质。不过，这些变种分析起来似乎特别困难。幸运的是（从分析学家的观点看），经验研究指出，它们的性能也不好。因此，虽然不能解析地量化，但随机性保持的要求似乎能够产生更精致更有效的Quicksort实现方法。尤为重要的是，保持随机性的算法确实得到性能的改进，这些改进可以完全从数学上量化，正如上面所描述的。 22

在Quicksort的实用变种的开发方面，数学上的分析起着重要的作用。我们将看到，有许多其他要考虑的问题。在那里，详细的数学分析成为算法设计过程的重要组成部分。

1.6 渐近逼近

上面给出的Quicksort平均运行时间的结论产生一个准确结果，但是我们还给出一个以著名函数表示的更简洁的近似表达式，这些著名函数还可以用来计算非常精确的数值结果。我们将要看到，常常出现这样的情况：准确的结果不是现成可用的，或至少近似结果的获得和解释要容易得多。算法分析的目标在理想情况下应该是获得准确的结果。从注重实效的观点来看，它或许更依赖于我们的一般目标，即能够使有用的性能结论尽量得到简明但却是精确的近似答案。

为此，我们需要使用处理这些逼近的经典技巧。在第4章，我们将考查欧拉-麦克劳林（Euler-Maclaurin）求和公式，它提供一种利用积分估计求和的方法。因此，我们可以通过计算

$$H_N = \sum_{1 \le k \le N} \frac{1}{k} \approx \int_1^N \frac{1}{x}\mathrm{d}x = \ln N$$

来逼近调和数。不过，关于≈的含义还可以精确得多，（例如）我们可以断言$H_N = \ln N + \gamma + 1/(2N) + O(1/N^2)$，其中$\gamma = 0.57721\cdots$是一个常数，在分析中叫做欧拉常数（Euler's constant）。虽然隐含在大O记号中的那些常数没有指定，但是这个公式还是提供了一种方法估计H_N的值。随着N的增加，公式的精度愈加精确。此外，如果我们需要更好的精度，那么我们可以得到H_N的一个精确到$O(N^{-3})$甚至$O(N^{-k})$的公式，其中k是任意常数。这样的近似叫做*渐近展开*（asymptotic expansion），它们是算法分析的核心，是第4章要讨论的内容。 23

渐近展开的使用可以看作是提供精确结果的理想目标和提供简洁近似结果的实际需要之间的一种折中。事实上，一方面如果需要的话，我们常常有能力得到一个更准确的表达式，但另一方面，我们又没有这种需要，因为只含有少数几项的表达式（如上面H_N的公式）已能够使我们将答案计算到多位小数的精度。一般我们还是使用≈记号将结果求和而不用命名一些无理常数，例如，定理1.3中的结论就是如此。

此外，准确结果和渐近近似值都服从概率模型中固有的不准确性（这种模型通常都是现实的理想化）并随机波动。表1-1显示对各种大小随机文件进行Quicksort的运行时间的准确、

近似和经验的值。准确值和近似值是从定理1.3给出的公式算出的；“经验”值是由小于10^6的随机正整数组成的100个文件测出的平均值。这不仅测试我们已经讨论的渐近逼近，而且也测试我们使用的随机排列模型中固有的“近似”（忽略相等关键字）。当出现相等关键字时，对Quicksort的分析在Sedgewick[21]中处理。

表1-1 由Quicksort使用的平均比较次数

文件大小	准确解	近似值	经验值
10 000	175 771	175 746	176 354
20 000	379 250	379 219	374 746
30 000	593 188	593 157	583 473
40 000	813 921	813 890	794 560
50 000	1 039 713	1 039 677	1 010 657
60 000	1 269 564	1 269 492	1 231 246
70 000	1 502 729	1 502 655	1 451 576
80 000	1 738 777	1 738 685	1 672 616
90 000	1 977 300	1 977 221	1 901 726
100 000	2 218 033	2 217 985	2 126 160

24

习题1.18 在由10^4个小于10^6的随机整数组成的文件中有多少关键字可能等于该文件中某个另外的关键字？模拟运行或进行数学分析（借助于数学计算系统的帮助），或二者都做。

习题1.19 用由小于M的随机正整数组成的一些文件试验，其中M = 10 000、1000、100或其他一些值。将Quicksort施于这些文件的性能与作用在相同大小的随机排列上的性能进行比较。描述使随机排列模型不准确的情形的特征。

习题1.20 讨论用类似于表1-1的表显示Mergesort的想法。

在计算复杂性方面，大O记号用来隐蔽所有排序的细节：语句“Mergesort需要$O(N\log N)$次比较”除算法、实现和计算机最基本的特征外隐藏了所有的细节。在算法分析中，渐近展开在保留了最重要的信息的同时，提供给我们一种可控的方式隐蔽不相关的细节，特别是所涉及的常数因子。最强大和一般的解析工具直接产生渐近展开，因此常常提供那些描述算法特性的、简洁而准确的表达式的简单直接结论。可是，有时在其他方法可用的情况下，我们却可能使用渐近估计来提供对程序性能的更精确的描述。

1.7 分布

一般说来，概率论告诉我们，关于开销分布$\prod_{Nk}$的其他事实也和我们对算法性能特征的理解相关。幸运的是，对事实上我们在算法分析中研究的所有例子，实际上知道平均值的渐近估计就足以作出可靠的预报。我们这里回顾一些基本的想法。那些根本不熟悉概率论的读者可以参考任何一本标准的教材，例如[6]。

由Quicksort对小的N所使用的比较次数的完整的分布在图1-2中表出。对于N的每一个值画出点$C_{Nk}/N!$：使Quicksort用到k次比较的那些输入的比例。每条曲线由于是完整的概率分布从而面积为1。由于平均$2N\ln N + O(N)$随着N增加而增加，因此这些曲线向右移动。对同样这组数据的稍微不同的观察在图1-3中描述，其中，每条曲线的水平轴适当地伸缩以便把平均值近似地置于中央，并且稍微平移以便分开这些曲线。可以看出，该分布收敛到一个“极限分布”。对于我们在本书中研究的许多问题，不仅像这样的极限分布确实存在，而且我们还能够准确

25

地刻画它们的特征。对于许多其他的问题，包括Quicksort，这是一个重要的挑战。但是，显而易见，这种分布是群集在平均值附近的。通常遇到的正是这种情况，事实上我们能够对这种结果做精确的表述，而不必知道该分布的更多细节。

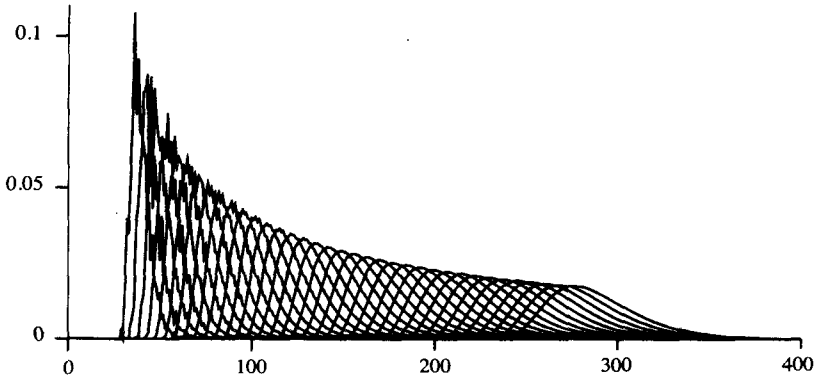


图1-2 在Quicksort中比较的分布（$10 < N \leqslant 50$）

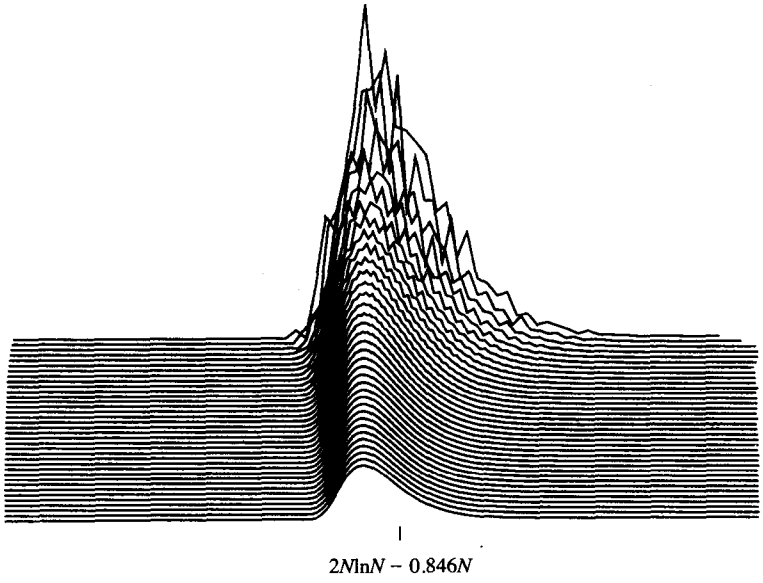


图1-3 在Quicksort中比较的分布（$10 < N \leqslant 50$）

（缩放并平移到中心且分离曲线）

上面已经讨论，如果$\prod_N$是大小为N的输入组数，Π_{Nk}是使得算法开销为k的大小为N的输入组数，则平均开销由下式给出

$$\mu = \sum_k k\Pi_{Nk} / \Pi_N$$

方差（variance）定义为

$$\sigma^2 = \sum_k (k-\mu)^2 \Pi_{Nk} / \Pi_N = \sum_k k^2 \Pi_{Nk} / \Pi_N - \mu^2$$

标准差（standard deviation）σ是方差的平方根。知道平均值和标准差通常就能可靠地预报性

能。完成这项工作的经典解析工具是切比雪夫不等式（Chebyshev inequality）：一次观察将大于标准差到平均值距离的c倍的概率小于$1/c^2$。如果标准差明显小于平均值，那么当N变大时观察到的值非常可能相当接近平均值。在算法分析中常常是这种情形。

习题1.21 在本章早给出的Mergesort实现中的比较次数的标准差是多少？

由Quicksort使用的比较次数的标准差为$\sqrt{(21-2\pi^2)/3N}=0.6482776N$（见3.12节）。因此，比如参考表1-1并取切比雪夫不等式中$c=\sqrt{10}$，则我们断言：存在多于90%的机会，当$N=100\ 000$时比较的次数在2 218 033的205 004（9.2%）之内。显然，这种精度对预报性能是足够的。

随着N的增加，相对精度也在增加。例如，当N增加时分布变得更集中在图1-3的峰值附近。实际上，切比雪夫不等式低估了在这种情形下的精度，如图1-4所示。这个图画出一个柱状图，显示由Quicksort作用到10 000个不同的随机文件时所用到的比较次数，其中每个文件有1000个元素。图中的阴影部分表示超过94%的试验落入这个实验的平均值的一个标准差之内。

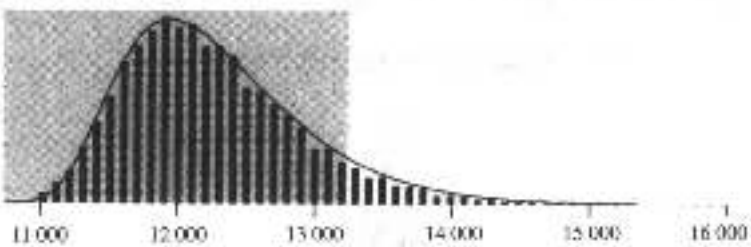


图1-4 Quicksort比较计数的经验柱状图
（$N=1000$的10 000次试验）

对于总的运行时间，我们可以将各个量的平均值求和（乘以开销），不过，计算总的运行时间的方差是一项复杂的计算，因为总的方差渐近地和最大的方差相同，所以我们不在这里进行繁琐的计算。对于大的N，标准差与平均值相比要小，这就解释了表1-1和图1-1中所观察到的精度。算法分析中不发生这种情况的情形很少见。如果我们对于平均开销有一个精确的渐进估计，并且知道标准差渐近地更小，那么我们通常认为算法是"被全面分析"的。

1.8 概率算法

Quicksort平均性能的分析依赖于随机顺序的输入。在许多实际场合下，这个假设不可能是严格合理的。一般说来，这反映算法分析中最严重的挑战之一：需要适当地将可能出现在实际中的输入模型公式化。

幸运的是，往往存在克服这个困难的方法：在使用算法之前先将输入"随机化"。对于排序算法，我们只要随机地排列输入文件然后就可进行排序。（见第6章中算法对此的特定实现。）如果照此进行，那么像上面所作的关于性能的概率论述就是完全合理的，并且能够在实际当中精确地预报性能，而不管输入如何排列。

26 ~ 28

通过进行随机选择（与任意的特定选择相反），只要算法能够从几个行动中选取其一，则付出较少的工作常常也能够得到同样的结果。对于Quicksort，这个原则就是随机地选取元素用作分割元，而不是每次都使用数组末尾的元素。如果这种方法仔细地实现（保留了文件中的随机性），那么就能够保证上面的概率分析是有效的。（对于小的子文件还应该使用截止的做法，因为它使得生成的随机数的个数减少大约M的大小。）随机化算法的其他一些例子可以

在[18]中找到。这样的算法在实践中是很有意义的，因为它们利用随机性得到了效率，并且以高概率避免了最坏情形的性能。此外，还可以作出关于性能的精确的概率表述。这进一步激发了为得到这样的结果而进行的高级技巧的研究。

我们一直在考虑的对Quicksort的分析的例子或许阐释了一套理想化的方法：不是所有的算法都能够像这种方法一样平稳地被处理。像上面这样全面的分析需要付出大量的努力，这种努力应该只留给那些最重要的算法。幸运的是，我们将看到存在许多基本的方法，确实享有使分析值得如此深入进行的基本要素：

- 可以指定现实的输入模型。
- 可以得到性能的数学描述。
- 可以解出简洁、准确的解。
- 结果可以用来比较一些变种，可与其他的算法进行比较，以及有助于调整算法的一些参数的值。

在本书中，我们考虑大量的这种方法，并着重考虑那些使第2点和第3点成立的数学方法。

通常，上面概述的这套方法中程序特定（依赖于实现）的部分被跳过，其目的或者是为了集中于算法设计，此时运行时间的粗略估计可能已经足够了，或者是为了集中在数学分析上，此时所涉及的数学问题的公式化以及求解方案是最感兴趣的问题。这就是那些涉及最重要的智力性挑战的领域，应该引起我们足够的注意。 29

上面已经提到，当前对计算机一般使用中，算法分析的一个重要挑战是将那些实际上代表输入的模型以及导致容易处理的分析问题的模型公式化。在这个问题上我们不犹豫，因为存在一大类组合学算法，对于这类算法这些模型是自然的。本书中我们稍详细地考虑这类算法的一些例子以及与其相关的基本结构。我们研究排列、树、串、trie树、字和映射，因为它们既是广泛研究的组合学结构又是广泛使用的数据结构，还因为“随机”结构既直接又现实。

在第2章～第4章，我们将着重讨论适用于算法性能研究的数学分析方法。这些材料在超越算法分析的许多应用中都是重要的。但是，我们讨论的范围由于为本书后面的应用做准备而将进一步展开。然后，在第5章～第8章，我们将应用这些方法分析某些基本的组合学算法，包括若干具有实际意义的问题。这些算法有许多在各种各样的计算机应用中具有基本的重要性，因此有必要对详细的分析进行相关的研究。在某些情况下，有些看似相当简单的算法可能导致非常复杂的数学分析；而在另外一些情况下，某些明显复杂的算法可能以直接简单的方式就能处理。在这两种情形下，分析都能够揭示在实践中正在使用的具有直接影响的算法间的重要区别。

本书囊括的基本方法当然适用于比我们能够在这个导论性的处理中讨论的方法广泛得多的一类算法和结构。我们在本书中将不涉及图论算法、数值算法或几何算法，不过许多这样的算法一直在被深入地研究。我们还简要地提到诸如摊还分析和概率方法这样一些方法，它们被成功地应用到许多重要算法的分析中。我们希望，掌握本书中引论性的材料是为了鉴赏在算法分析的研究文献中的那些材料所做的充足准备。除较早引用的Knuth、Sedgewick以及Aho、Hopcroft和Ullman的书之外，关于算法分析和计算复杂性信息的其他来源为Cormen、Leiserson和Rivest的书[5]以及Gonnet和Baeza-Yates的书[8]。 30

同等重要的是，我们讨论组合学性质的解析问题，它使我们开发一般的方法，这些方法可以帮助我们分析那些未来的、迄今尚未发现的算法。我们使用的方法产生于组合数学和渐进分析的经典领域，我们能够应用来自这些领域的经典方法，以统一的方式处理大量的算法。

在姐妹篇[7]中，我们考虑组合学架构，这种架构能够直接引出算法性能的解析描述，并从这些描述中得出渐进估计的方法。

在本书中，我们讨论重要的基本概念，而同时又为[7]更先进的处理奠定了基础。Graham、Knuth和Patashnik[9]是涉及我们用到的更多数学材料的良好来源；像Comtet[4]（组合学）和Herici[11]（分析）这样的标准参考文献也有相关的材料。一般说来，我们在本书中使用初等的组合学和实分析，而[7]则是从组合学的观点涉及更彻底的处理，并依赖于对渐近性的复分析。

我们的起点是研究那些广泛使用中的基本算法的特征，但是我们在本书中的基本目的是对所遇到的组合学和解析方法提供连贯的处理。在适当的时候，我们详细地考虑那些自然发生但可能（就当前所知）并不适用的任何算法的数学问题。采取这样一种处理方法，我们必然涉及范围和多样性的问题。此外，在全书的例子中可以看到，我们求解的问题是与许多重要应用直接相关的。

参考文献

1. M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*, Dover, New York, 1970.
2. A. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Algorithms*, Addison-Wesley, Reading, MA, 1975.
3. B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *Maple V Library Reference Manual*, Springer-Verlag, New York, 1991.
4. L. Comtet. *Advanced Combinatorics*, Reidel, Dordrecht, 1974.
5. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, MIT Press, New York, 1990.
6. W. Feller. *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 1957.
7. P. Flajolet and R. Sedgewick. *Analytic Combinatorics*, in preparation.
8. G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*, 2nd edition, Addison-Wesley, Reading, MA, 1991.
9. R. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.
10. D. H. Greene and D. E. Knuth. *Mathematics for the Analysis of Algorithms*, Birkhäuser, Boston, 1981.
11. P. Henrici. *Applied and Computational Complex Analysis*, 3 volumes, John Wiley, New York, 1977.
12. C. A. R. Hoare. "Quicksort," *Computer Journal* **5**, 1962, 10–15.
13. D. E. Knuth. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
14. D. E. Knuth. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.
15. D. E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

31

16. D. E. KNUTH. "Big Omicron and big Omega and big Theta," *SIGACT News*, April-June 1976, 18–24.
17. D. E. KNUTH. "Mathematical Analysis of Algorithms," *Information Processing 71*, Proceedings of the IFIP Congress, Ljubljana, 1971, 19–27.
18. M. O. RABIN. "Probabilistic algorithms," in *Algorithms and Complexity*, J. F. Traub, ed., Academic Press, New York, 1976, 21–39.
19. R. SEDGEWICK. *Algorithms*, 2nd edition, Addison-Wesley, Reading, MA, 1988.
20. R. SEDGEWICK. *Quicksort*, Garland Publishing, New York, 1980.
21. R. SEDGEWICK. "Quicksort with equal keys," *SIAM Journal on Computing* **6**, 1977, 240–267.
22. R. SEDGEWICK. "Implementing Quicksort programs," *Communications of the ACM* **21**, 1978, 847–856.
23. E. TUFTE. *The Visual Display of Quantitative Information*, Graphics Press, Chesire, CT, 1987. 32
24. J. S. VITTER AND P. FLAJOLET, "Analysis of algorithms and data structures," in *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, 431–524. 33

第2章 递归关系

我们考虑对其进行分析的算法通常都可以表示成递归或迭代的过程，这意味着，一般我们能够把求解特定问题的开销用求解一些更小的问题的开销来表示。正如在上一章Quicksort和Mergesort分析中所看到的，对于这种情况，数学上最基本的处理是使用递推关系。它代表着一种从程序的递归表示到描述其特性的函数的递归表示间直接映射的实现方式。虽然同样的递归分解仍然是问题的核心，但是还是有一些其他的方法也能够做到这一点。在第3章我们将看到，这也是算法分析中生成函数方法的应用基础。

由于递归本身携带了大量的信息，因此描述算法性能的递推关系的获得则是分析中向前迈出的重大一步。算法涉及的输入模型的特定性质都被封装在相对简单的数学表示之中。许多算法可能不服从这样一种简单的描述，幸运的是，许多最重要的算法都可以由递推公式相当简单地表示出来，而它们的分析则导致递归，或者描述平均情形，或者确定最坏情形的性能的界。这一点在第1章以及在第5章～第8章的许多例子中都做了解释。在本章，我们着重讨论各种递归的数学性质而不考虑它们的起因和来源。我们将遇到许多类型的递归，他们出现在本章研究一些特定算法的环境中，而且我们还要再涉及第1章讨论过的一些递归，不过现在关心的是递归本身。

首先，我们考查递归的某些基本性质以及将它们分类的方法。然后，考查“一阶”递归的准确解，这里，n的函数被表示成$n-1$的函数。我们还要讨论高阶线性常系数递归的准确解。然后，查看其他各种类型的递归并讨论求解某些非线性递归以及非常数系数递归的近似解的方法。接着，考查在算法分析中具有特殊重要性的一类递归的解：“分治”类的递归。它包括Mergesort递归的推导和准确解的获得，涉及整数的二进制表示。最后，通过考查适用于一大类分治算法的分析的一般结果来结束本章的讨论。 35

迄今为止我们考虑过的所有递归都有准确解。这样的递归在算法分析中常常出现，特别是使用递归对离散量进行精确计数的时候。但是，准确答案可能要涉及一些不相关的细节：例如，与近似解$2^n/3$相比，使用像 $(2^n-(-1)^n)/3$这样的准确解来工作大概就不值得这么费事了。在这种情况下，$(-1)^n$项的作用是使得答案为整数，与2^n相比是可以忽略的；另一方面，在像准确答案$2^n(1+(-1)^n)$ 中的项 $(-1)^n$ 是不能忽略的。有必要避免在用精度换取简单时过分粗心，以及用简单换取精度时过分热心的毛病。我们有兴趣于得到既简单又精确的近似表达式（即使准确解是可以得到的）。此外，我们常常遇到这样的递归，它的准确解是得不到的，但是我们可以估计解的增长率，并且在许多的情况下得到精确的渐近估计。

递推关系通常也叫做*差分方程*（difference equations），因为它们可以表示成离散的差分算子$\nabla f_n \equiv f_n - f_{n-1}$。它们是常微分方程的离散模拟。求解微分方程的技巧在这里也有相应的方法，因为类似的技巧常常能够用来求解相似的递归。我们在下一章将会看到，在某些情况下存在明显的对应，这种对应使得我们能够从微分方程的解得到递归的解。

有一大批文献讨论递归的性质，因为它们也直接产生于许多应用数学领域。例如，像直接导致递归的牛顿方法这样的数值迭代算法，可在像Bender和Orszag[3]的书中找到详细的描述。 36

本章的目的是概括论述通常产生于算法分析中的递归类型和某些基本的求解方法。我们可以使用*生成函数*（generating functions）以严格、系统的方式处理许多这样的递推关系，其

中生成函数将在下一章详细讨论。我们还将在第4章比较详细地考虑获取渐进逼近的一些工具。在第5章到第8章我们将遇到递归的许多不同的例子，它们描述了基本算法的性质。

一旦开始详细讨论先进的工具，我们就会看到，递归在算法分析中常常不是要使用的最自然的数学工具。递归可能会使分析复杂化，而这是可以通过在更高层次上的工作来避免的，即使用符号的方法得到生成函数间的关系，然后再对生成函数进行直接的分析。这种想法将在后面各章引入，并在[12]中详细处理。在许多情况下，实际上通向最终解决的最简单和最直接的途径是避免递归。指出这一点并不是阻挠对递归的研究，它的确对于许多的应用来说是相当富有成效的方法，不过我们要使读者确信，先进的工具可能对那些导致过分复杂的递归问题提供简单的解法。

简而言之，递归直接产生于算法分析的自然处理过程之中，并且对于许多重要的问题能够提供容易的解法。由于我们后面的重点在于生成函数技术，因此这里只对在求解递归的文献中已经得到的方法进行简要的介绍。关于求解递归的更多信息可以在包括[3]、[4]、[6]、[14]、[15]、[16]、[21]和[22]等标准的参考文献中找到。

2.1 基本性质

在第1章分析Quicksort和Mergesort时我们遇到下列3个递归:

$$C_N = \left(1+\frac{1}{N}\right)C_{N-1}+2 \qquad (N>1,\ C_1=2) \tag{2-1}$$

$$C_N = C_{\lfloor N/2\rfloor}+C_{\lceil N/2\rceil}+N \qquad (N>1,\ C_1=0) \tag{2-2}$$

$$C_N = N+1+\frac{1}{N}\sum_{1\le j\le N}(C_{j-1}+C_{N-j}) \quad (N>0,\ C_0=0) \tag{2-3}$$

37 每个方程都表示一些特定的问题。通过将两边乘以适当的因子我们求解式 (2-1)；得到式（2-2）的近似解是通过求解$N = 2^n$的特殊情况，然后由归纳法证明对于一般N的解来进行的；至于式（2-3），我们是通过从它对$N-1$情形下的同一方程减去它而将式（2-3）变换成式（2-1）的。

这种特别的技巧或许就是求解递归常常需要的“智囊”的代表。不过，刚刚提到的这种不多的“智囊”技巧不适用于通常发生的许多递归问题，甚至包括著名的线性递归

$$F_n = F_{n-1}+F_{n-2} \qquad (n>1, F_0=0,\ F_1=1)$$

它定义了斐波那契（Fibonacci）数列{0, 1, 1, 2, 3, 5, 8, 13, 21, 34,…}。斐波那契数已经过透彻的研究并实际上确实发生在许多重要算法的设计和分析之中。本章我们考虑求解它们以及其他递归的若干方法，而且在下一章和其后各章我们还将考虑其他适用的系统性方法。

表2-1 递归的分类

递归类型	典型例
1阶	
线性	$a_n = na_{n-1}-1$
非线性	$a_n = 1/(1+a_{n-1})$
2阶	
线性	$a_n = a_{n-1}+2a_{n-2}$
非线性	$a_n = a_{n-1}a_{n-2}+\sqrt{a_{n-2}}$
变系数	$a_n = na_{n-1}+(n-1)a_{n-2}+1$

（续）

递归类型	典型例
t阶	$a_n = f(a_{n-1}, a_{n-2}, \cdots, a_{n-t})$
n阶	$a_n = n + a_{n-1} + a_{n-2} \cdots + a_1$
分治情形	$a_n = a_{\lfloor n/2 \rfloor} + a_{\lceil n/2 \rceil} + n$

递归的分类是通过其各项组合的方式、所涉及系数的类型，以及所用到前面的项的项数和性质来进行的。表2-1列出我们将要考虑的递归以及代表性的例子。

38

值的计算。正常情况下，递归提供一种有效的方法计算所考虑的量的值。特别是，求解任意的递归的最初一步是用它计算一些小的值来获得递归如何增长的一种感觉。这可以通过手算一些小的值来完成，或者不难编制程序来计算一些更大的值来完成。例如，对应递归式(2-3)，程序2.1将计算对于所有小于等于Nmax的N的Quicksort的平均比较次数的准确值（见表1-1）。这个程序使用一个大小为Nmax的数组存储前面算出的值。应该避免使用直接基于递归的纯递归程序：通过计算所有的值$C_{N-1}, C_{N-2}, \cdots, C_1$来计算$C_N$将会导致特别低的低效，因为有许多的值将被毫无必要地重复计算。

如果有些情形像程序1.2那样已经足够，那么就要避免太深地追究数学。我们认为简洁的数学解更是我们所期望的——的确，我们把分析本身看作是使得程序2.1更为有效的一道工序！总之，这样的“解”可以用来证实我们的分析。另一个极端的做法是通过对所有可能的输入运行一个程序来计算该程序的平均运行时间的蛮干（通常是不实际的）方法。

习题2.1 写出计算斐波那契递归的递归程序和非递归程序，并用每个程序来计算F_{20}。解释这种情形下每个程序的性能。

习题2.2 计算C_{Nmax}的程序2.1使用了多少次算术运算？

程序2.1 计算值（Quicksort递归）

```
C[0] := 0;
for N := 1 to Nmax do
  begin
    C[N] := N+1;
    for i := 1 to N do C[N] := C[N]+(C[i-1]+C[N-i])/N;
  end;
```

39

习题2.3 编写一个直接使用递归式（2-1）的程序计算其值。这个程序所使用的算术运算次数比程序2.1（见前面的习题）如何？

习题2.4 估计使用递归式（2-2）和式（2-3）计算其值的递归程序和非递归程序需要多少次运算。

习题2.5 编写一个程序比较Quicksort、它的三数中值变体、基数交换排序，从第1章给出的那些递归求值。对于Quicksort，检验已知解的值；对于其他算法，作出解的性质的猜测。

缩放和平移。递归的一个基本性质是依赖于初始值：在线性递归

$$a_n = f(a_{n-1}) \quad (n>0,\ a_0 = 1)$$

中将初始条件$a_0 = 1$改变成$a_0 = 2$将会改变a_n对所有n的值（若$f(0) = 0$，则a_n的值都将增加一倍）。“初始值”可以出现在任何地方：若有

$$b_n = f(b_{n-1}) \quad (n>t,\ b_t = 1)$$

则必然有$b_n = a_{n-t}$。改变初始值叫做*缩放*（scaling）递归；移动初始值叫做*平移*（shifting）

递归。初始值最常见的是从问题直接得出，不过我们常常使用缩放和平移简化通向解的途径。我们不打算叙述递归的解的最一般形式，而是求解一种自然形式，并假设在适当的时候解可以缩放或平移。

线性性。具有多于一个初始值的线性递归可以通过独立地改变初始值并将解组合而得到“缩放”。如果$f(x,y)$是一个线性函数且$f(0,0)=0$，那么

$$a_n=f(a_{n-1},a_{n-2}) \qquad (n>1)$$

的解（初始值a_0和a_1的函数）为a_0乘以

$$u_n=f(u_{n-1},u_{n-2}) \qquad (n>1 \text{ 且} u_0=1,\ u_1=0)$$

的解加上a_1乘以

40

$$v_n=f(v_{n-1},v_{n-2}) \qquad (n>1,\ v_0=0,\ v_1=1)$$

的解。条件$f(0,0)=0$使得递归成为齐次递归：如果在f中有一个常数项，那么它以及初始值都必须要考虑。这以一种直接的方式将任意t阶齐次线性递归（对于任意一组初始值）的一般解作为t个特定的解的线性组合而加以推广。我们在第1章用到过这个过程求解描述由Quicksort所采取的交换次数的递归，结果表示成描述比较次数和阶段数的递归的形式。

习题2.6 求解递归

$$a_n=a_{n-1}+a_{n-2} \qquad (n>1,\ a_0=p,a_1=q)$$

用斐波那契数表示你的答案。

习题2.7 求解非齐次递归

$$a_n=a_{n-1}+a_{n-2}+r \qquad (n>1 \quad a_0=p,a_1=q)$$

用斐波那契数表示你的答案。

习题2.8 对于线性函数f，将递归

$$a_n=f(a_{n-1},a_{n-2}) \qquad n>1$$

的解表示成$a_0,a_1,f(0,0)$和$a_n=f(a_{n-1},a_{n-2})-f(0,0)$对于$a_1=1,a_0=0$和$a_0=1,a_1=0$的解。

2.2 一阶递归

或许最简单形式的递归可以直接化成一个乘积。递归

$$a_n=x_na_{n-1} \qquad (n>0,\ a_0=1)$$

等价于

$$a_n=\prod_{1\le k\le n}x_k$$

于是，如果$x_n=n$则$a_n=n!$，而如果$x_n=2$则$a_n=2^n$等等。

这种变换是迭代（iteration）的一个简单例子：将递归应用到它自己直至只剩下常数和初始值为止，然后再简化。迭代也直接适用于下一个最简单类型的递归，通常遇到的要多得多，它是立即简化为和：

41

$$a_n=a_{n-1}+y_n \qquad (n>0,\ a_0=0)$$

等价于

$$a_n = \sum_{1 \le k \le n} y_k$$

于是，如果$y_n = 1$那么$a_n = n$，而如果$y_n = n - 1$则$a_n = n(n-1)/2$等等。

表2-2给出一些经常遇到的离散和。更全面的表可以在像Graham、Knuth和Patashnik[14]或Riordan[23]这样的标准文献中找到。

习题2.9 求解递归

$$a_n = \frac{n}{n+2} a_{n-1} \qquad (n > 0,\ a_0 = 1)$$

习题2.10 求解递归

$$a_n = a_{n-1} + (-1)^n n \qquad (n > 0,\ a_0 = 1)$$

表2-2 初等离散和

几何级数	$\sum_{0 \le k < n} x^k = \frac{1-x^n}{1-x}$
等差级数	$\sum_{0 \le k < n} k = \frac{n(n-1)}{2} = \binom{n}{2}$
二项式系数	$\sum_{0 \le k \le n} \binom{k}{m} = \binom{n+1}{m+1}$
二项式定理	$\sum_{0 \le k \le n} \binom{n}{k} x^k y^{n-k} = (x+y)^n$
调和级数	$\sum_{1 \le k \le n} \frac{1}{k} = H_n$
调和级数的和	$\sum_{1 \le k \le n} H_k = nH_n - n$
范德蒙德卷积	$\sum_{0 \le k \le n} \binom{n}{k}\binom{m}{t-k} = \binom{n+m}{t}$

42

如果我们有一个不是这么简单的递归，那么我们常常可以通过将递归的两边同乘一个适当的因子而把它简化。在第1章我们已经见过一些这样的例子。例如将式（1-1）的两边除以$N+1$来求解式（1-1），得到一个$C_N/(N+1)$形式的简单递归，经过迭代可以直接变换成一个求和式。

习题2.11 求解递归

$$na_n = (n-2)a_{n-1} + 2 \qquad (n > 1,\ a_1 = 1)$$

（提示：用$n-1$乘以两边）。

习题2.12 求解递归

$$a_n = 2a_{n-1} + 1 \qquad n > 1 \text{且} a_1 = 1$$

（提示：用2^n除两边）。

用这种方法求解递推关系（差分方程）是对利用积分因子相乘然后再积分来求解微分方程的一种模拟。用于递推关系的因子有时候叫做*求和因子*（summation factor）。适当选择求和因子可以使求解许多实践中产生的递归成为可能。例如，描述用于三数中值Quicksort平均比较次数的递归的准确解就是由Knuth利用这样的技巧[18]得到的（也见[24]）。

定理2.1（一阶线性递归） 递归

$$a_n = x_n a_{n-1} + y_n \qquad (n > 0,\ a_0 = 0)$$

的显式解为

$$a_n = y_n + \sum_{1 \le j < n} y_j x_{j+1} x_{j+2} \cdots x_n$$

证明：用$x_n x_{n-1} \cdots x_1$除两边并进行迭代，我们有

$$a_n = x_n x_{n-1} \cdots x_1 \sum_{1 \le j < n} \frac{y_j}{x_j x_{j-1} \cdots x_1}$$

$$= y_n + \sum_{1 \le j < n} y_j x_{j+1} x_{j+2} \cdots x_n$$

43 同样的结果还可以通过用$x_{n+1} x_{n+2} \cdots$（假设它是收敛的）乘以两边并迭代而得到。 ■

例如，定理2.1的证明说的是，我们应该用

$$\frac{N+1}{N} \frac{N}{N-1} \frac{N-1}{N-2} \cdots \frac{3}{2} \frac{2}{1} = N+1$$

除两边来求解递归

$$C_N = \left(1 + \frac{1}{N}\right) C_{N-1} + 2 \quad N > 1 \text{ 且} C_1 = 2$$

这正是我们在1.5节所做的。再有，解

$$2(N+1)(H_{N+1} - 1)$$

也可以直接从定理叙述中所给出的解的显式形式得到。

定理2.1是将具有常或变系数一阶线性递归变换到和式的变换的完全特征化，求解递归的问题化成和式求值的问题。

习题2.13 求解递归

$$a_n = \frac{n}{n+1} a_{n-1} + 1 \qquad (n > 0,\ a_0 = 1)$$

习题2.14 以x、y和初始值a_t的形式写出

$$a_n = x_n a_{n-1} + y_n \qquad (n > t)$$

的解。

习题2.15 求解递归

$$n a_n = (n+1) a_{n-1} + 2n \qquad (n > 0,\ a_0 = 0)$$

习题2.16 求解递归

$$n a_n = (n-4) a_{n-1} + 12 n H_n \qquad (n > 4,\ \text{对} n \leqslant 4 \text{有} a_n = 0)$$

习题2.17 [Yao]（“2-3树的边缘分析”）求解递归

$$A_N = A_{N-1} - \frac{2A_{N-1}}{N} + 2\left(1 - \frac{2A_{N-1}}{N}\right) \quad (N > 0, A_0 = 0)$$

这个递归描述了下面的随机过程：将一组N个元素分成“2-结点”和“3-结点”。在每一步，每个2-结点以概率$2/N$转变成一个3-结点，而每个3-结点以概率$3/N$转变成两个2-结点。经过N步之后，2-结点的平均个数是多少？ 44

2.3 非线性一阶递归

当递归由关于a_n和a_{n-1}的非线性函数组成的时候，会出现很多的情况，我们不能期望像定

理2.1那样有一个封闭形式的解。本节考虑一些确实能够得到解的有趣的情形。

简单收敛。计算初始的一些值的一个令人信服的理由是，许多表面上复杂的递归实际上就收敛到一个常数。例如，考虑方程

$$a_n = 1/(1 + a_{n-1}) \qquad (n>0,\ a_0 = 1)$$

这是所谓的连分式方程，将在2.5节讨论。通过计算一些初始的值，我们可以猜测递归收敛到一个常数:

n	a_n	$\lvert a_n - (\sqrt{5}-1)/2 \rvert$
1	0.500 000 000 000	0.118 033 988 750
2	0.666 666 666 667	0.048 632 677 917
3	0.600 000 000 000	0.018 033 988 750
4	0.625 000 000 000	0.006 966 011 250
5	0.615 384 615 385	0.002 649 373 365
6	0.619 047 619 048	0.001 013 630 298
7	0.617 647 058 824	0.000 386 929 926
8	0.618 181 818 182	0.000 147 829 432
9	0.617 977 528 090	0.000 056 460 660

每次迭代增加常数位（约半位）的有效数字的位数，这叫做**简单收敛**。如果假设递归收敛到一个常数，那么我们知道，这个常数必然满足$\alpha = 1/(1 + \alpha)$，或$1 - \alpha - \alpha^2 = 0$，它导致解$\alpha = \left(\sqrt{5}-1\right)/2 \approx 0.6180334$。

习题2.18 定义$b_n = a_n - \alpha$，其中a_n和α定义如上，求出当n很大时b_n的近似公式。

习题2.19 证明$a_n = \cos(a_{n-1})$收敛并计算$\lim_{n\to\infty} a_n$到5位十进制小数位。

二次收敛和牛顿方法。这个计算函数根的著名迭代方法可以看作是计算一阶递归的近似解的过程（比如，可见[3]）。例如，计算正数β平方根的牛顿方法是对下面的公式进行迭代 45

$$a_n = \frac{1}{2}\left(a_{n-1} + \frac{\beta}{a_{n-1}}\right) \qquad (n > 0,\ a_0 = 1)$$

在该递归中变换变量，我们可以看到这个方法为什么这么有效。令$b_n = a_n - \alpha$，通过简单的代数整理发现

$$b_n = \frac{1}{2}\frac{b_{n-1}^2 + \beta - \alpha^2}{b_{n-1} + \alpha}$$

于是，若$\alpha = \sqrt{\beta}$，则大致有$b_n \approx b^2_{n-1}$。例如计算2的平方根，这个迭代给出下列序列:

n	a_n	$a_n - \sqrt{2}$
1	1.500 000 000 000	0.085 786 437 627
2	1.416 666 666 667	0.002 453 104 294
3	1.414 215 686 275	0.000 002 123 901
4	1.414 213 562 375	0.000 000 000 002
5	1.414 213 562 373	0.000 000 000 000

每一次迭代大约将现有的有效数字位数扩大1倍，这就是二次收敛的情形。

习题2.20 讨论当试图使用牛顿方法计算$\sqrt{-1}$时会发生什么情况:

$$a_n = \frac{1}{2}\left(a_{n-1} - \frac{1}{a_{n-1}}\right) \qquad (n>0,\ a_0 \neq 0)$$

缓慢收敛。考虑递归

$$a_n = a_{n-1}(1 - a_{n-1}) \qquad (n>0,\ a_0 = 1/2)$$

我们将在第5章看到类似的递归在“随机二叉树”高度的分析中的作用。由于递归中的项是递减的并且是正的，因此不难看出$\lim_{n\to\infty} a_n = 0$。为找出收敛速度，自然要考虑$1/a_n$。代入得到

46

$$\begin{aligned}\frac{1}{a_n} &= \frac{1}{a_{n-1}}\left(\frac{1}{1-a_{n-1}}\right)\\ &= \frac{1}{a_{n-1}}(1 + a_{n-1} + a_{n-1}^2 + \cdots)\\ &> \frac{1}{a_{n-1}} + 1\end{aligned}$$

反复套用该不等式则给出$1/a_n > n$或$a_n < 1/n$。因此我们发现$a_n = O(1/n)$。

习题2.21 证明$a_n = \Theta(1/n)$。计算开始的几项并猜测用c/n近似a_n的常数c，然后严格证明na_n趋向于一个常数。

习题2.22 [De Bruijn]证明递归

$$a_n = \sin(a_{n-1}) \quad (n>0,\ a_0 = 1)$$

的解满足$\lim_{n\to\infty} a_n = 0$及$a_n = O(1/\sqrt{n})$。（提示：考虑变量代换$b_n = 1/a_n$。）

刚刚考虑过的这3种情形是形式

$$a_n = f(a_{n-1})$$

的特殊情形，其中f为连续函数。如果a_n收敛到极限α，那么α必然是该函数的一个不动点，且$\alpha = f(\alpha)$。上面的3种情形是下面一般情形的代表：如果$0<|f'(\alpha)|<1$，则收敛是简单的；如果$f'(\alpha)=0$，则收敛是二次的；而若$|f'(\alpha)|=1$，则收敛是“慢”的。

习题2.23 当$f'(\alpha)>1$时会发生什么情况？

习题2.24 叙述对应上述三种情形局部收敛性（即当a_0足够接近α时）的充分性准则，并用$f'(\alpha)$和$f''(\alpha)$表示出收敛速度。

2.4 高阶递归

下面我们考虑关于a_n的方程右边是a_{n-2}、a_{n-3}等以及a_{n-1}的线性组合而系数则是常数的情形的递归。看一个简单例子，考虑递归

47

$$a_n = 3a_{n-1} - 2a_{n-2} \qquad (n>1,\ a_0 = 0,\ a_1 = 1)$$

首先观察$a_n - a_{n-1} = 2(a_{n-1} - a_{n-2})$，这是关于量$a_n - a_{n-1}$的初等递归，容易求解。迭代该过程得到$a_n - a_{n-1} = 2^{n-1}$，将这个初等递归重复并求和得到解$a_n = 2^n - 1$。我们还可以通过观察$a_n - 2a_{n-1} = a_{n-1} - 2a_{n-2}$来求解这个递归。这些方法恰好对应分解二次三项式$1 - 3x + 2x^2 = (1-2x)(1-x)$。

类似地，我们可以发现通过求解关于$a_n - 3a_{n-1}$或$a_n - 2a_{n-1}$的初等递归而得到

$$a_n = 5a_{n-1} - 6a_{n-2} \qquad (n>1,\ a_0 = 0,\ a_1 = 1)$$

的解为$a_n = 3^n - 2^n$。

习题2.25 给出以$a_n = 4^n - 3^n + 2^n$为解的递归。

这些例子阐明了解的一般形式，而且这种类型的递归可以显式解出。

定理2.2（常系数线性递归） 递归

$$a_n = x_1a_{n-1} + x_2a_{n-2} + \cdots + x_ta_{n-t} \qquad n \geqslant t$$

的所有解可以表示成形如$n^j\beta^n$的项的线性组合（组合系数依赖于初始条件$a_0, a_1, \cdots, a_{t-1}$），其中$\beta$是特征多项式

$$q(z) \equiv z^t - x_1z^{t-1} - x_2z^{t-2} - \cdots - x_t$$

的根，而若β的重数为v，则j满足$0 \leqslant j < v$。

证明：我们自然要找形如$a_n = \beta^n$形式的解。代入任意这样的解必然满足

$$\beta^n = x_1\beta^{n-1} + x_2\beta^{n-2} + \cdots + x_t\beta^{n-t} \qquad (n \geqslant t)$$

或等价地，

$$\beta^{n-t}q(\beta) = 0。$$

即对于特征多项式的任意的根β，β^n是递归的解。 48

其次，设β为$q(z)$的二重根。我们想要证明$n\beta^n$以及β^n都是递归的解。再次代入，必然有

$$n\beta^n = x_1(n-1)\beta^{n-1} + x_2(n-2)\beta^{n-2} + \cdots + x_t(n-t)\beta^{n-t} \qquad (n \geqslant t)$$

或等价地，

$$\beta^{n-t}((n-t)q(\beta) + \beta q'(\beta)) = 0$$

正如所料这是正确的，因为当β是二重根时$q(\beta) = q'(\beta) = 0$，对于更高的重数可以用类似的方式处理。

特征多项式有多少个根（包括重根），这个过程就给出递归的多少个解。它和递归的阶数t相同。不仅如此，这些解还是线性无关的（它们在∞有不同的增长阶）。由于t阶递归的解形成一个t维向量空间，因此递归的每一个解必然能够表示成形如$n^j\beta^n$这种特解的线性组合。 ■

求线性组合的系数。任意线性递归的准确解都可以从定理2.2通过使用初始值$a_0, a_1, \cdots, a_{t-1}$建立一个方程组，并解出含有解的项的线性组合中的常数系数而得到。例如，考虑递归

$$a_n = 5a_{n-1} - 6a_{n-2} \qquad n \geqslant 2 \text{ 且} a_0 = 0 \text{和} a_1 = 1$$

特征方程为$z^2 - 5z + 6 = (z-3)(z-2)$，于是

$$a_n = c_03^n + c_12^n$$

将$n = 0$和$n = 1$代入到公式中则有

$$a_0 = 0 = c_0 + c_1$$
$$a_1 = 1 = 3c_0 + 2c_1$$

这个方程组的解为$c_0 = 1$和$c_1 = -1$，因此$a_n = 3^n - 2^n$。 49

退化情形。我们已经给出了一种方法求解任意线性递归的准确解，求解过程使得通过初始条件确定全部解的方法明白清晰。当系数为0与/或某些根有相同的模时，结果虽然容易理解，但多少有些违反直觉。例如，考虑递归

$$a_n = 2a_{n-1} - a_{n-2} \qquad (n \geqslant 2,\ a_0 = 1 \text{和} a_1 = 2)$$

由于特征方程是$z^2-2z+1=(z-1)^2$（根为1，重数是2），因此解为

$$a_n=c_0 1^n+c_1 n 1^n$$

应用初始条件

$$a_0=1=c_0$$
$$a_1=2=c_0+c_1$$

得到$c_0=c_1=1$，因此$a_n=n+1$。但是如果初始条件是$a_0=a_1=1$，那么解就会是$a_n=1$，即为常数而不是线性增长。下面的例子更具有戏剧性，考虑递归

$$a_n=2a_{n-1}-a_{n-2}+2a_{n-3} \qquad (n>3)$$

此处的解为

$$a_n=c_0 1^n+c_1(-1)^n+c_2 2^n$$

而初始条件的各种选择可使得解的增长率是常数、指数或符号变动。这个例子指出，当处理递归时注意细节（初始条件）是相当重要的。

斐波那契数。我们已经提到过熟悉的序列{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, …}，它由原始的二阶递归

50

$$F_n=F_{n-1}+F_{n-2} \qquad (n>1,\ F_0=0,\ F_1=1)$$

定义。由于u^2-u-1的根是$\phi=(1+\sqrt{5})/2=1.61803\cdots$和$\hat{\phi}=(1-\sqrt{5})/2=-0.61803\cdots$，因此根据定理2.2，解为

$$F_N=c_0\phi^N+c_1\hat{\phi}^N$$

其中c_0和c_1为常数。应用初始条件

$$F_0=0=c_0+c_1$$
$$F_1=1=c_0\phi+c_1\hat{\phi}$$

得到解为

$$F_N=\frac{1}{\sqrt{5}}(\phi^N-\hat{\phi}^N)$$

由于按照绝对值ϕ比1大而$\hat{\phi}$小于1，因此在上述F_N的表达式中的项$\hat{\phi}^N$可以忽略，而事实上F_N总是最接近$\phi^N/\sqrt{5}$的整数。随着N的增大，比率F_{N+1}/F_N趋向于ϕ，它是在数学、艺术、建筑和自然中著名的黄金分割。

定理2.2提供了一种方法以获得高阶线性递归的全部准确解，不过我们在第3章和第4章还要讨论这个论题，因为那里先进的工具提供了方便的方法来获取在实践中有用的结果。定理3.3给出一种容易的方法计算系数，特别是识别那些变成零的项。此外推广了刚刚对斐波那契数观察到的现象：由于项$n^j\beta^n$都是指数的，因此对于大的n，（在具有非零系数的那些项中间）具有最大β的那些项将控制所有其他的项，而在这些项中具有最大的j的项将起决定的作用。生成函数（定理3.3）和渐近分析（定理4.1）给我们提供了方便的方法对于任意的线性递归明确地识别首项并计算其系数的值。在某些情形下，这可以提供一条捷径来获取好的近似解，特别是当t很大的时候。对于小的t，这里描述的获取准确解的方法是相当有效的。

习题2.26 解释如何求解形如

$$a_n=x_1a_{n-1}+x_2a_{n-2}+\cdots+x_ta_{n-t}+r \qquad (n\geqslant t)$$

的非齐次递归。

习题2.27 给出初始条件a_0和a_1使得

$$a_n = 5a_{n-1} - 6a_{n-2} \qquad (n>1)$$

的解为$a_n = 2^n$。存在使得解为$a_n = 2^n - 1$的初始条件吗？ 51

习题2.28 给出初始条件a_0, a_1和a_2使得

$$a_n = 2a_{n-1} - a_{n-2} + 2a_{n-3} \qquad (n>2)$$

的解的增长率为(*i*)常数，(*ii*)指数，(*iii*)符号波动。

习题2.29 求解递归

$$a_n = 2a_{n-1} + 4a_{n-2} \qquad (n>1,\ a_1 = 2,\ a_0 = 1)$$

习题2.30 求解递归

$$a_n = 2a_{n-1} - a_{n-2} \qquad (n>1,\ a_0 = 0,\ a_1 = 1)$$

将初始条件改为$a_0 = a_1 = 1$，再求解这个递归。

习题2.31 求解递归

$$a_n = a_{n-1} - a_{n-2} \qquad (n>1,\ a_0 = 0,\ a_1 = 1)$$

习题2.32 求解递归

$$2a_n = 3a_{n-1} - 3a_{n-2} + a_{n-3} \qquad (n>2,\ a_0 = 0,\ a_1 = 1,\ a_2 = 2)$$

习题2.33 找出描述一个序列的递归，这个序列的奇数项的增长阶呈指数递减，但偶数项呈指数递增。

习题2.34 给出“3阶”斐波那契递归

$$F_N^{(3)} = F_{N-1}^{(3)} + F_{N-2}^{(3)} + F_{N-3}^{(3)} \qquad (N>2,\ F_0^{(3)} = F_1^{(3)} = 0,\ F_2^{(3)} = 1)$$

的一个近似解。对于$F_{20}^{(3)}$将你的近似结果和准确的值进行比较。

非常数系数。如果系数不是常数，那么就需要更高级的技巧，因为定理2.2不再适用。典型情况下，生成函数（见第3章）或逼近方法（见下面）是需要的，但是某些更高阶的问题可以用求和因子来解决。例如，递归

$$a_n = na_{n-1} + n(n-1)a_{n-2} \qquad (n>1,\ a_1 = 1,\ a_0 = 0)$$

可以通过用$n!$直接除两边而解出，此时得到的是以$a_n/n!$表示的斐波那契递归，它指出$a_n = n!F_n$。 52

习题2.35 求解递归

$$n(n-1)a_n = (n-1)a_{n-1} + a_{n-2} \qquad (n>1,\ a_1 = 1,\ a_0 = 1)$$

符号解。虽然对于高阶递归还没有像定理2.2这样的闭型，但是对一般形式

$$a_n = s_{n-1}a_{n-1} + t_{n-2}a_{n-2} \qquad (n>1,\ a_1 = 1,\ a_0 = 0)$$

的迭代结果已经有了相当程度的研究。对于充分大的n，我们有

$$\begin{aligned}
a_2 &= s_1 \\
a_3 &= s_2s_1 + t_1 \\
a_4 &= s_3s_2s_1 + s_3t_1 + t_2s_1 \\
a_5 &= s_4s_3s_2s_1 + s_4s_3t_1 + s_4t_2s_1 + t_3s_2s_1 + t_3t_1 \\
a_6 &= s_5s_4s_3s_2s_1 + s_5s_4s_3t_1 + s_5s_4t_2s_1 + s_5t_3s_2s_1 + s_5t_3t_1 \\
&\quad + t_4s_3s_2s_1 + t_4s_3t_1 + t_4t_2s_1
\end{aligned}$$

等等。在a_n的展开式中单项的项数恰好是F_n，且展开式有许多其他的性质：它们与所谓的接续多项式（continuant polynomials）相关，后者则与连分数紧密相关（见下面）。具体细节可在

Graham、Knuth和Patashnik [14]中找到。

习题2.36 给出一个简单算法来确定是否给定的单项式$s_{i_1}s_{i_2}\cdots s_{i_p}t_{j_1}t_{j_2}\cdots t_{j_q}$出现在$a_n$的展开式中。这样的单项式有多少?

我们上面强调，对于常系数的情况最重要的是获得首项的渐近性能，因为虽然准确解可用但用起来是冗长的。对于非常数系数，准确解一般不是现成的，因此对许多的应用我们必须满足于近似解。现在就来讨论获取这种近似的方法。

2.5 求解递归的方法

非线性递归或带有变系数的递归通常可以从各种不同处理方法中选择一种方法求解或逼近。本节我们将考虑这样的一些方法和例子。

53

我们一直在处理至少（原则上）对某些值可以得到准确解的递归。这样的问题在算法分析中的确经常产生，可是我们自然要估计到会遇到没有已知方法求其准确解的那种递归。现在开始讨论处理这种递归有效的先进方法为时尚早，不过现在可以对如何获取精确近似解给出一些指导方针并考虑若干实例。这些方法对学习关于递归性能的基本事实是有用的，尽管更一般或先进的方法是适用的。

我们考虑四种一般的方法：*变量代换*（change of variable），它通过用另外的变量重新计算递归而将递归简化；*取值表法*（repertoire），它从给定的递归反过来求出解空间；*自精化方法*（bootstrapping），该法首先求出一个近似解，然后利用递归本身再求出更精确的解，如此继续直到得到足够精确的答案或者不太可能有进一步的改进为止；*扰动*（perturbation）*法*，该方法研究将递归变换为一个类似的、更简单的、具有已知解的递归的效果。前两种方法常常得到递归的准确解，后两种方法更经常用于获得近似解。

变量代换。定理2.1实际上描述的是变量代换："如果我们把变量代换成$b_n = a_n/(x_nx_{n-1}\cdots x_1)$，则$b_n$满足一个简单递归，当进行迭代时它可以化成一个和式。"在前一节以及本章中较早的一些地方也用到变量代换。更复杂的变量代换可以用于得到看起来很难的一些递归的准确解。例如考虑非线性二阶递归

$$a_n = \sqrt{a_{n-1}a_{n-2}} \qquad (n>1,\ a_0=1,\ a_1=2)$$

如果我们对这个方程的两边取对数并做变量代换$b_n = \lg a_n$，那么我们发现b_n满足

$$b_n = \frac{1}{2}(b_{n-1}+b_{n-2}) \qquad (n>1,\ b_0=0,\ b_1=1)$$

这是一个常系数线性递归。

54

习题2.37 给出b_n和a_n的准确公式。

习题2.38 求解递归

$$a_n = \sqrt{1+a_{n-1}^2} \qquad (n>0, a_0=0)$$

我们的下一个例子来自寄存器分配算法[11]的研究：

$$a_n = a_{n-1}^2 - 2 \qquad (n>0)$$

对于$a_0 = 0$或$a_0 = 2$，解为$a_n = 2$ $(n>1)$；而对于$a_0 = 1$，则解为$a_n = -1$ $(n>1)$，但是对于更大的a_0，则该递归对初始值a_0的依赖性要比我们见过的其他一阶递归更为复杂。

这是所谓的二次递归，它是少数可以通过变量代换被显式解出的二次递归之一。置$a_n = b_n + 1/b_n$，我们得到递归

$$b_n + \frac{1}{b_n} = b_{n-1}^2 + \frac{1}{b_{n-1}^2} \qquad (n>0,\ b_0+1/b_0=a_0)$$

但是这意味着可以让$b_n = b_{n-1}^2$来求解，迭代立即得到解

$$b_n = b_0^{2^n}$$

根据二次方程，b_0很容易从a_0计算:

$$b_0 = \frac{1}{2}\left(a_0 \pm \sqrt{a_0^2 - 4}\right)$$

因此

$$a_n = \left(\frac{1}{2}\left(a_0 + \sqrt{a_0^2 - 4}\right)\right)^{2^n} + \left(\frac{1}{2}\left(a_0 - \sqrt{a_0^2 - 4}\right)\right)^{2^n}$$

对于$a_0 > 2$，两个根中只有较大的根，即带有加号的那个根，在这个表达式中起支配作用。

习题2.39 从上面的讨论，对$a_0 = 3, 4$解出寄存器分配递归。讨论当$a_0 = 3/2$时会发生什么情况。

习题2.40 对$a_0 = 2 + \varepsilon$求解寄存器分配递归，其中ε是一个任意固定的正常数。给出一个精确的近似答案。

55

习题2.41 找出通过线性变换 $(b_n = f(\alpha, \beta, \gamma)a_n + g(\alpha, \beta, \gamma))$ 使得递归$a_n = \alpha a^2_{n-1} + \beta a_{n-1} + \gamma$化成$b_n = b^2_{n-1} - 2$的参数$\alpha, \beta$和$\gamma$的所有的值。特别地，证明$a_n = a^2_{n-1} + 1$不能化成这种形式。

习题2.42 [Melzak] 求解递归

$$a_n = 2a_{n-1}\sqrt{1 - a_{n-1}^2} \qquad \left(n > 0,\ a_0 = \frac{1}{2}\right)$$

其中$a_0 = 1/3$。画出a_6作为a_0的函数的图像并解释所观察到的现象。

一方面，基础线性性可能很难认出，而找出求解非线性递归的变量代换并不比找出计算定积分的变量代换容易。事实上，可以使用更高级的分析（迭代理论）来证明，大多数非线性递归不能以这种方式化简。另一方面，简化产生于实践中的递归的变量代换可能不难找到，有些变换可能导致线性的形式。正如寄存器分配的例子所表述的，这样的递归确实在算法分析中发生。

来看另外一个例子。考虑使用变量代换获得与连分式相关的递归的准确解。

$$a_n = 1/(1 + a_{n-1}) \qquad (n > 0,\ a_0 = 1)$$

反复计算这个递归得到序列

$$a_0 = 1$$

$$a_1 = \frac{1}{1+1} = \frac{1}{2}$$

$$a_2 = \frac{1}{1+\frac{1}{1+1}} = \frac{1}{1+\frac{1}{2}} = \frac{2}{3}$$

$$a_3 = \frac{1}{1+\frac{1}{1+\frac{1}{1+1}}} = \frac{1}{1+\frac{2}{3}} = \frac{3}{5}$$

$$a_4 = \frac{1}{1+\frac{1}{1+\frac{1}{1+\frac{1}{1+1}}}} = \frac{1}{1+\frac{3}{5}} = \frac{5}{8}$$

56

等等。从这些初始的值可以认出斐波那契数，我们当然想到形式$a_n = b_{n-1}/b_n$，将其代入到递归中则给出

$$\frac{b_{n-1}}{b_n} = 1\Big/\left(1+\frac{b_{n-2}}{b_{n-1}}\right) \qquad (n>1,\ b_0 = b_1 = 1)$$

两边除以b_{n-1}得到

$$\frac{1}{b_n} = \frac{1}{b_{n-1}+b_{n-2}} \qquad (n>1,\ b_0 = b_1 = 1)$$

它意味着$b_n = F_{n+1}$，即斐波那契序列。将这种做法推广，得到一种把一般的“连分式”类表示成递归的解的方法。

习题2.43 求解递归

$$a_n = \frac{\alpha a_{n-1}+\beta}{\gamma a_{n-1}+\delta} \qquad (n>0,\ a_0 = 1)$$

习题2.44 考虑递归

$$a_n = 1/(s_n + t_n a_{n-1}) \qquad (n>0,\ a_0 = 1)$$

其中$\{s_n\}$和$\{t_n\}$是任意的序列。把a_n表示成由一个线性递归定义的序列中的两个相邻项的比。

取值表法。在某些情况下通向准确解的另外一种方法是所谓的取值表法，此时我们使用一些已知的函数找出类似所要寻找的解的一族解，将它们组合以给出答案。这种方法主要适用于线性递归，包括下列几步：

- 通过添加一个附加的函数项将递归松弛；
- 将一些已知的函数代入到递归中，得到类似于该递归的一些等式；
- 将这些等式线性组合，得到恒等于该递归的方程。

例如，考虑递归

57

$$a_n = (n-1)a_{n-1} - na_{n-2} + n - 1 \qquad (n>1,\ a_0 = a_1 = 1)$$

在等号右边引入一个量$f(n)$，于是我们想要求解

$$a_n = (n-1)a_{n-1} - na_{n-2} + f(n)$$

其中$n>1$且$a_0 = a_1 = 1$以及$f(n) = n-1$。为此，我们为a_n列出各种可能并观察$f(n)$应取得的能够求解（暂时不管初始条件）递归的取值表。对于这个例子，我们得到下面的表：

a_n	$a_n - (n-1)a_{n-1} + na_{n-2}$
1	2
n	$n-1$
n^2	$n+1$

表的第1行说的是$a_n = 1$是$f(n) = 2$（以及初始条件$a_0 = 1$和$a_1 = 1$）时的一个解；第2行说的是$a_n = n$是$f(n) = n-1$（以及初始条件$a_0 = 0$和$a_1 = 1$）时的一个解；而第3行则是说$a_n = n^2$是$f(n) = n+1$（以及初始条件$a_0 = 0$和$a_1 = 1$）时的一个解。现在，它们的线性组合也是解。从第3行减去第1行得到的结果意味着$a_n = n^2 - 1$是$f(n) = n-1$（以及初始条件$a_0 = -1$和$a_1 = 0$）时的一个解。现在对于$f(n) = n-1$，我们有两个（线性无关的）解，将它们组合而得到右边的初始值，产生结果$a_n = n^2 - n + 1$。

该法的成功依赖于能够找到一组线性无关的解以及对初始条件的恰当处理。对解的形式的直觉和知识对于确定取值表是有用的。使用该法的经典例子在Knuth和Schönhage[20]对一个等价算法的分析中使用。

对于Quicksort递归，我们从

$$a_n = f(n) + \frac{2}{n}\sum_{1 \le j \le n} a_{j-1}$$

开始，其中$n>0$且$a_0=0$。这导致下面的取值表 58

a_n	$a_n - \left(2\sum_{0 \le j < n} a_j\right)/n$
1	-1
H_n	$-H_n+2$
n	1
$\lambda(n+1)$	0
nH_n	$\frac{1}{2}(n-1)+H_n$
$n(n-1)$	$\frac{1}{3}(n^2-1)+n-1$

于是，$2nH_n + 2H_n + \lambda(n+1) - 2$是$f(n) = n+1$时的一个解；以$\lambda = 2$分解初始值得到解

$$2(n+1)H_n - 2n$$

正如所料（见定理1.3）。这个解依赖于表的第5行，应该试一试它，因为由于其他的原因我们也许认为解可能是$O(n \log n)$。注意，取值表也可以方便地给出对于其他的$f(n)$的解，不过可能需要更详细的算法分析。

习题2.45 求解Quicksort递归，其中$f(n) = n^3$。

习题2.46 [Greene and Knuth] 求解Quicksort三数中值递归（见第1章中的方程（1-4）），使用取值表法（对于该递归使用差分和求和因子的直接解，见[18]或[24]，而对于使用生成函数得到的解，见第3章）。

自精化方法。我们常常能够猜测一个递归的解的近似值、然后递归本身可以用来对估计施加限制，以便得到更精确的估计。这种方法非正式地分为如下几步：

- 使用递归计算一些数值；
- 猜测解的近似形式；
- 把近似解代回到递归；
- 根据所猜测的解和代入的情况，证明解的更接近的界。

为了叙述清楚，设将这种方法应用到斐波那契递归：

$$a_n = a_{n-1} + a_{n-2} \qquad (n>1\text{且}a_0=0\text{和}a_1=1)$$ 59

首先我们注意a_n是递增的，因此$a_{n-1} > a_{n-2}$和$a_n > 2a_{n-2}$。重复这个不等式则有$a_n > 2^{n/2}$，于是我们知道a_n至少是指数的增长率。另一方面，$a_{n-2} < a_{n-1}$意味着$a_n < 2a_{n-1}$，或（重复使用而有）$a_n < 2^n$。这样，我们就证明了a_n的指数增长的上界和下界，而且我们感觉猜测解的形式$a_n \sim c_0\alpha^n$正确，其中$\sqrt{2} < \alpha < 2$。从递归得知α必然满足$\alpha^2 - \alpha - 1 = 0$，由此得到$\phi$和$\hat{\phi}$。确定$\alpha$的值以后，我们可以进行自精化并回到递归和初始值以找出适当的系数。

习题2.47 求解递归

$$a_n = 2/(n + a_{n-1}) \qquad (n>0,\ a_0 = 1)$$

习题2.48 使用自精化方法证明三数中值Quicksort用到$\alpha N\ln N + O(N)$次比较，确定α的值。

习题2.49 [Greene和 Knuth] 使用自精化方法证明

$$a_n = \frac{1}{n}\sum_{0\le k<n}\frac{a_k}{n-k} \qquad n>0,\ a_0 = 1$$

的解满足$n^2a_n = O(1)$。

扰动法。另外一种求递归近似解的途径是求解一个更简单的相关递归。这是求解递归的一般处理方法，它首先经过研究抽取占支配地位的主要部分而得到一个简化的递归，求解简化的递归，然后比较原始递归和简化递归的解。这种方法类似数值分析中一类熟知的方法，即*扰动方法*（perturbation methods）。这种方法可以非正式地分为如下几步进行：

- 将递归进行微小的修改，使变成一个已知的递归；
- 变换变量使获得已知的界并变换成一个关于解的（更小的）未知部分的递归；
- 定界或求解未知的“误差”项。

例如，考虑递归

60

$$a_{n+1} = 2a_n + \frac{a_{n-1}}{n^2} \qquad (n>1,\ a_0 = 1,\ a_1 = 2)$$

看来有理由假设其中的最后一项只对递归产生微小的影响，因为它的系数是$1/n^2$，于是

$$a_{n+1} \approx 2a_n$$

可以预料，粗略的形式为$a_n \approx 2^n$。为了更精确，我们考虑更简单的序列

$$b_{n+1} = 2b_n \qquad (n>0,\ b_0 = 1)$$

（从而$b_n = 2^n$）并通过形成比值来比较这两个递归

$$\rho_n = \frac{a_n}{b_n} = \frac{a_n}{2^n}$$

从这两个递归我们得到

$$\rho_{n+1} = \rho_n + \frac{1}{4n^2}\rho_{n-1} \qquad (n>0,\ \rho_0 = 1)$$

显然，ρ_n是递增的。为了证明它们趋向于一个常数，注意

$$\rho_{n+1} < \rho_n\left(1+\frac{1}{4n^2}\right) \qquad (n>1) \qquad \text{从而} \quad \rho_{n+1} < \prod_{k=1}^{n}\left(1+\frac{1}{4k^2}\right)$$

但是对应右边的无穷乘积单调收敛到

$$\alpha_0 = \prod_{k=1}^{\infty}\left(1+\frac{1}{4k^2}\right) = 1.46505\cdots$$

因此ρ_n以α_0为上界，并且随着它的增长，它必然收敛到一个常数。这样，我们就证明了

$$a_n \sim \alpha\cdot 2^n$$

其中常数$\alpha<1.46505\cdots$（此外，这个界并不太粗糙，例如$\rho_{100} = 1.44130\cdots$）。

上面的例子只是一个简单的例子，目的就是阐述这个方法。一般说来，情况可能要更复杂，可能需要将方法迭代若干步，很可能要引入几个中间的递归。它涉及自精化方法，这种

方法我们刚刚讨论过。如果简化的递归只有非闭型的表达式，那么也可能发生困难。虽然如此，扰动方法仍是计算递归渐近解的重要方法。 61

习题2.50 求出“扰动的”斐波那契递归

$$a_{n+1}=\left(1+\frac{1}{n}\right)a_n+\left(1-\frac{1}{n}\right)a_{n-1} \quad (n>1,\ a_0=0,\ a_1=1)$$

的解的渐近增长。

习题2.51 求解递归

$$a_n=na_{n-1}+n^2a_{n-2} \qquad (n>1,\ a_1=1,\ a_0=0)$$

习题2.52 [Aho and Sloane] 递归

$$a_n=a_{n-1}^2+1 \qquad (n>0,\ a_0=1)$$

满足$a_n \sim \lambda\alpha^{2^n}$，其中$\lambda$和$\alpha$均为常量。求对于$\alpha$的收敛级数并确定$\alpha$到50位十进制小数。（提示：考虑$b_n=\lg a_n$。）

习题2.53 求解下列斐波那契递归的扰动

$$a_n=\left(1-\frac{1}{n}\right)(a_{n-1}+a_{n-2}) \quad (n>1,\ a_0=a_1=1)$$

尝试形如$n^\alpha\ \phi^n$的解并确定α。

2.6 二分分治递归和二进制数

通过应用下面基本算法设计范例我们可以得到各种大量问题的好算法：把问题分成两个相等大小的子问题，递归地将它们解出，然后应用所得到的解再来求解原问题。Mergesort就是这种算法的一个原型，例如（见1.2节中定理1.2的证明），Mergesort所使用的比较次数由递归

$$C_N=C_{\lfloor N/2\rfloor}+C_{\lceil N/2\rceil}+N \qquad (N>1,\ C_1=0) \tag{2-4}$$

的解给出。这个递归和其他跟它类似的递归产生于在与Mergesort具有相同基本结构的各种算法的分析之中。通常能够确定满足这种递归的函数的渐近增长，但是需要在获取准确结果方面特别仔细，主要是因为若N是奇数则大小为N的问题不能被分成两个相等大小的子问题，能够做到最好的情形是使问题的大小相差为1。对于大的N这是可以忽略的，不过对于小的N需要注意，通常这种递归结构都要涉及许多小的子问题。 62

我们很快就要看到，这意味着准确解趋向于有周期，有时甚至是严重不连续，且常常不能够用光滑函数来描述的情形。例如，图2-1显示Mergesort递归式2-4和类似的递归

$$C_N=2C_{\lceil N/2\rceil}+N \qquad (N>1,\ C_1=0)$$

的解。前者相对平滑，后者的解以缺乏连续的基于不规则碎段的性能为其特征，这种性能常出现在分治递归中。

在图2-1中描述的两个函数是 $\sim N\lg N$并且当N是2的幂时精确地等于$N\lg N$的。图2-2还是这两个函数，只不过是减掉$N\lg N$后的图，它解释了这两个函数的线性项的周期行为。与Mergesort相关的周期函数的值相当小且连续，但其导数在2的幂处不连续；另一个函数的值相对要大而且基本上是不连续的。这样的行为使得我们在试图对比较程序进行精确甚至是渐近估计的时候都会遇到问题。然而幸运的是，当递归用数字表示法的方式理解的时候，通常我们可以相当容易地看出解的本性。为了解释这一点，我们开始考虑另外一个重要的算法，它

是一般问题求解策略的特殊实例，可以追溯到古代。

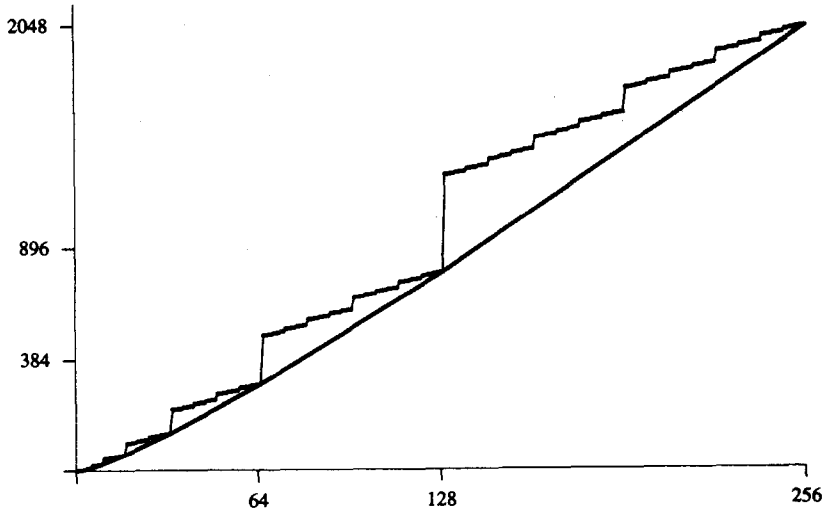


图2-1 二分分治递归的解

$C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ （下面的曲线）

$C_N = C_{\lceil N/2 \rceil} + C_{\lceil N/2 \rceil} + N$ （上面的曲线）

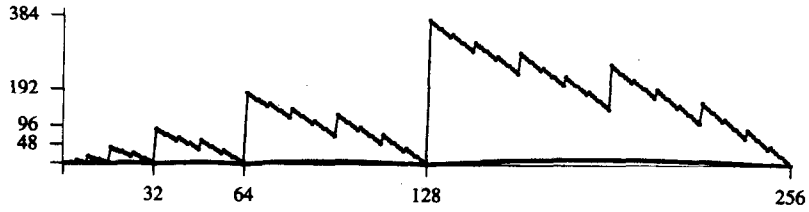


图2-2 二分分治递归的周期项

$C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N$ （下面的曲线）

$C_N = C_{\lceil N/2 \rceil} + C_{\lceil N/2 \rceil} + N$ （上面的曲线）

折半查找。最简单和最著名的二分分治算法之一是所谓的折半查找（binary search）。给定一个固定的数集，我们希望能够迅速确定一个所要查找的数是否在这个集合中。为此，首先将数集中的数排序，然后对于任意被查找的数，我们可以使用程序2.2所示的过程，查看一下中间的数，如果被查找的数较小，那么再（递归地）使用相同的方法查看表的左半部分。

定理2.3 (折半查找) *在对大小为N的表进行折半查找的不成功搜索中所用到的比较次数在最坏情形下等于N的二进制表示的位数。二者均可由递归*

$$B_N = B_{\lfloor N/2 \rfloor} + 1 \qquad (N \geqslant 2,\ B_1 = 1)$$

描述，它的准确解 $B_N = \lfloor \lg N \rfloor + 1$。

63 ~ 64

证明 在查看“中间”之后，一个元素被删除，文件的两部分大小分别为 $\lfloor (N-1)/2 \rfloor$ 和 $\lceil (N-1)/2 \rceil$。通过分别检查奇数N和偶数N来建立递归，两个数中的大者总是 $\lfloor N/2 \rfloor$。例如在大小为83的表中，在第1次比较之后两个子文件的大小都是41，可是在大小为82的表中，一个

子文件的大小为40，而另一个的大小为41。

因为 $\lfloor N/2 \rfloor$ 的计算正好等价于将N的二进制表示右移一个比特位，所以它等于N的二进制表示中的比特位数（忽略那些前导0）。反复进行递归相当于计算比特位的个数，当遇到最高位的1时计算结束。

若$2^n \leqslant N < 2^{n+1}$或取对数$n \leqslant \lg N < n+1$，则$N$的二进制表示中的比特数为$n+1$。这就是说，根据定义，$n = \lfloor \lg N \rfloor$。 ■

程序2.2 折半查找

```
function search(l, r, v:  integer):  integer;
  var x:  integer;
  begin
    x := (l+r) div 2;
    if l > r
      then << search unsuccessful >> else
    if v = a[x].key
      then search := x else
    if v < a[x].key
      then search := search(l, x-1, v)
      else search := search(x+1, r, v)
  end;
```

函数$\lg N$和 $\lfloor \lg N \rfloor$ 在图2-3中与小数部分即 $\{\lg N\} = \lg N - \lfloor \lg N \rfloor$ 一起画出。

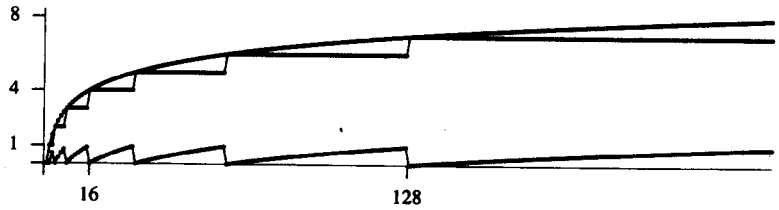


图2-3 $\lg N$（上）；$\lfloor \lg N \rfloor$（中）；$\{\lg N\}$（下）

习题2.54 在大小为N的表中用折半查找进行的不成功查找中所使用的比较次数在最佳的情况下是多少？

习题2.55 考虑“三分查找”算法，其中一个文件被分成三份，确定关键字在哪部分需要进行两次比较，该算法递归地使用。写出由该算法所使用的比较次数在最坏情况下的特征，并与折半查找进行比较。

Mergesort递归的准确解。 Mergesort递归2-2容易通过差分求解：如果D_N定义为$C_{N+1} - C_N$，那么D_N满足递归

$$D_N = D_{\lfloor N/2 \rfloor} + 1 \qquad (N \geqslant 2,\ D_1 = 0)$$

由此得到

$$D_N = \lfloor \lg N \rfloor + 2$$

于是，

$$C_N = N - 1 + \sum_{1 \leqslant k < N} \left(\lfloor \lg k \rfloor + 1 \right)$$

有许多方法对该和进行计算并求得C_N的准确公式。如上所述，采用记录整数二进制表示的关系的方法是有用的。特别地我们刚看到，$\left(\lfloor \lg k \rfloor + 1\right)$是$k$的二进制表示中的比特数（忽略那些前导0），因此$C_N$正好是小于$N$的所有正数的二进制表示中的比特数再加上$N-1$。

定理2.4 (Mergesort) Mergesort所使用的比较次数等于$N-1$加上所有小于N的数的二进制表示中的比特数。这两个量均由递归

$$C_N = C_{\lfloor N/2 \rfloor} + C_{\lceil N/2 \rceil} + N \qquad (N \geq 2,\ C_1 = 0)$$

描述，它的准确解为 $C_N = N\lfloor \lg N \rfloor + 2N - 2^{\lfloor \lg N \rfloor + 1}$。

证明 定理的第1部分通过上面的讨论建立。现在，所有小于N的$N-1$个数都有最右边的一个比特位；它们中的$N-2$个（即除去1后所有的）都有一个次最右边的比特位；它们中的$N-4$个（即除去1、2和3后所有的）都有一个从最右边数第3位置上的比特位；它们中的$N-8$个都有一个从最右边数第4位置上的比特位等等，因此我们必然有

65~66

$$\begin{aligned} C_N &= (N-1)+(N-1)+(N-2)+(N-4)+\cdots+(N-2^{\lfloor \lg N \rfloor}) \\ &= (N-1)+N(\lfloor \lg N \rfloor+1)-(1+2+4+\cdots+2^{\lfloor \lg N \rfloor}) \\ &= N\lfloor \lg N \rfloor + 2N - 2^{\lfloor \lg N \rfloor + 1} \end{aligned}$$

正如上面所注释过的，$\lfloor \lg N \rfloor$是具有周期特性的不连续函数。不过我们还提到，C_N本身是连续的，于是在涉及$\lfloor \lg N \rfloor$的这两个函数中的不连续性（但不是周期性）抵消了。这种现象在图2-4中做了解释，并被下面的推论中的计算所支持。 ■

推论 $C_N = N\lg N + N\theta(1-\{\lg N\})$，其中$\theta(x) = 1 + x - 2^x$是满足$\theta(0) = \theta(1) = 0$和$0 < \theta(x) < 0.086$的一个正函数，其中$0 < x < 1$。

证明 把分解式$\lfloor \lg N \rfloor = \lg N - \{\lg N\}$代入直接得到。值$0.086 \approx 1 - \lg e + \lg \lg e$通过置$\theta(x)$的导数为0可算得。 ■

67

习题2.56 通过考虑最右边的那些比特位，给出直接的证明：所有小于N的数的二进制表示中的比特位数满足式2-4，但是添加的项是$N-1$而不是N。

习题2.57 证明对所有的正N，有 $N\lfloor \lg N \rfloor + 2N - 2^{\lfloor \lg N \rfloor + 1} = N\lceil \lg N \rceil + N - 2^{\lceil \lg N \rceil}$。（见习题1.4）

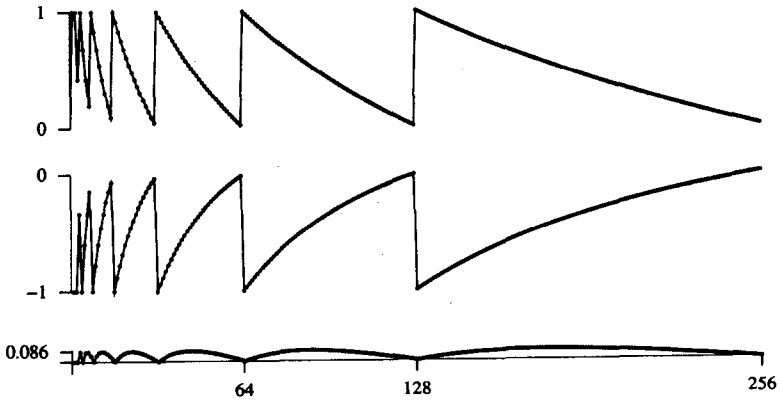


图2-4 周期函数$\theta(1-\{\lg N\})$的合成

$1-\{\lg N\}$（上）

$1-2^{1-\{\lg N\}}$（中）

$2-\{\lg N\}-2^{1-\{\lg N\}}$（下）

二进制数的其他性质。因为二进制整数在许多基本算法中自然地模拟（二分）决策过程，

所以我们来研究二进制整数的一些性质。上面遇到的量可能发生在任何通过递归地将其分为两部分的问题求解算法的分析中，如折半查找和Mergesort所用的方法，而类似的量显然将会在其他的分治算法中产生。为了完成我们对二分分治递归的研究，再考虑两个经常发生在算法分析中的数的二进制表示的性质。

定义 给定一个整数N，定义总体计数函数（population count function）ν_N为N的二进制表示中1的个数并定义累积总体计数函数（cumulated population count function）P_N为所有小于N的数的二进制表示中1的个数。

定义 给定一个整数N，定义标尺函数（ruler function）ψ_N为N的二进制表示中位于最后那些1的个数，并定义累积标尺函数（cumulated ruler function）R_N为所有小于N的数的二进制表示中位于最后的1的个数。

表2-3给出$N < 16$的这些函数的值，读者可能发现尝试对于大的N计算这些函数的值是有启发意义的。例如，83的二进制表示为1010011，因此$\nu_{83} = 4$以及$\psi_{83} = 2$。累积的值

$$P_N = \sum_{0 \le j < N} \nu_j \quad 和 \quad R_N = \sum_{0 \le j < N} \psi_j$$

不那么容易计算。例如$P_{84} = 215$而$R_{84} = 78$。

表2-3 标尺及总体计数函数

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
ν_N	1	1	2	1	2	2	3	1	2	2	3	2	3	3	4
P_N	0	1	2	4	5	7	9	12	13	15	17	20	22	25	28
ψ_N	1	0	2	0	1	0	3	0	1	0	2	0	1	0	4
R_N	0	1	1	3	3	4	4	7	7	8	8	10	10	11	11

68

不难看出，例如，

$$P_{2^n} = n2^{n-1} 和 R_{2^n} = 2^n - 1$$

和

$$P_N = \frac{1}{2} N \lg N + O(N) \quad 和 \quad R_N = N + O(\lg N)$$

不过准确的表示和更精确的渐近估计是很难得到的，如图2-5中$P_N - (N\lg N)/2$的图所示。

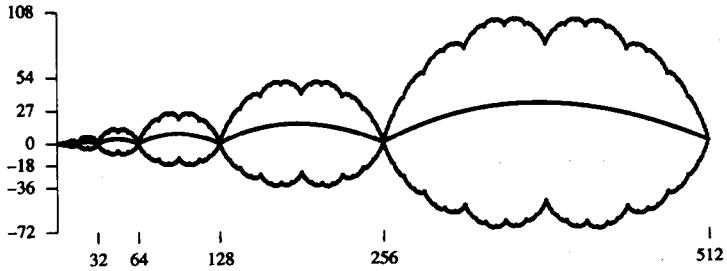


图2-5 比特计算中的周期项和小数项

{小于N的那些数中的比特1的个数} − $(N \lg N)/2$（上）
{小于N的那些数中的比特0的个数} − $(N \lg N)/2$（下）
{小于N的那些数中的比特数} − $N \lg N$（中）

上面提到，这种类型的函数满足那些能够直接得到但是可能明显不同的递归，甚至是在描述相同的量的时候。例如考虑最左边的比特，我们看到小于N的那$2^{\lfloor \lg N \rfloor}$个数以0开始而其余的则以1开始，因此我们有

69

$$P_N = P_{2^{\lfloor \lg N \rfloor}} + (N - 2^{\lfloor \lg N \rfloor}) + P_{N-2^{\lfloor \lg N \rfloor}}$$

不过考虑最右边的比特，显然

$$P_N = P_{\lfloor N/2 \rfloor} + P_{\lceil N/2 \rceil} + \lfloor N/2 \rfloor \qquad (N > 1,\ P_1 = 0)$$

这类似于Mergesort递归，它与对所有小于N的比特的计数有关（应该有大约一半的比特1那么多），但这个函数却显著不同。（比较图2-5和图2-2）对比特0计数的函数是类似的：二者在性能上都是碎段形，但是它们相互抵消而形成周期的连续函数，我们前面已经见到过（见图2-2）。Delange[7]详细地研究了函数P_N，并用无处可微的函数来表示。

其他递归。折半查找和Mergesort是典型的“二分分治”算法，它们阐述这样一些算法的实施是多么的自然，以及可能的解的特征。我们给出的例子和对已经看到的数的二进制表示的性质的关系，使得沿着我们一直在讨论的方向获取其他类似递归的更精确的解成为可能，记住这样的事实：因为所涉及的、甚至是主项中的函数的碎段式的性能，精确解可能是难以琢磨（或没有我们想像的那么有用）的。表2-4显示通常产生的一些递归以及从下一节给出的几个一般定理中得到的近似解。在这个表中，$a_{N/2}$意味着“$a_{\lfloor N/2 \rfloor}$或$a_{\lceil N/2 \rceil}$”，$2a_{N/2}$意味着“$a_{N/2} + a_{N/2}$”等等——在下一节我们讨论这类小的变动不影响表2-4中给出的渐近结果（不过它们妨碍我们给出更一般的估计）。

正常情况下，涉及这种递归的应用包括最坏情形的结果，如定理2.5和定理2.6，但是如果子问题是独立的并且在分割后仍然是“随机”的，那么对于某些问题，这些结果也可以产生期望的开销。

表2-4 二分分治递归及解

$a_N = a_{N/2} + 1$	$\lg N + O(1)$
$a_N = a_{N/2} + N$	$2N + O(\lg N)$
$a_N = a_{N/2} + N\lg N$	$\Theta(N\lg N)$
$a_N = 2a_{N/2} + 1$	$\Theta(N)$
$a_N = 2a_{N/2} + \lg N$	$\Theta(N)$
$a_N = 2a_{N/2} + N$	$N\log N + O(N)$
$a_N = 2a_{N/2} + N\lg N$	$\frac{1}{2}N\lg N^2 + O(N\log N)$
$a_N = 2a_{N/2} + N\lg^{\delta-1}N$	$\delta^{-1}N\lg^{\delta}N + O(N\lg^{\delta-1}N)$
$a_N = 2a_{N/2} + N^2$	$2N^2 + O(N)$
$a_N = 3a_{N/2} + N$	$\Theta(N^{\lg 3})$
$a_N = 4a_{N/2} + N$	$\Theta(N^2)$

从数的这些性质到位串的更具组合学特性的性质实际上只是一小步，认识这一点很重要。例如，假设我们想要知道随机位串中连续的0的最长串的平均长度，比方说是为了对一种运算器的设计进行某种优化。这是数的性质还是表示它的那些比特位的性质？这样一些问题直接导致更一般应用组合学的研究，我们将在第7章考虑。

在算法分析中常常遇到这种类型的函数，认识到它们与二进制数的简单性质之间的关系是很有意义的。当然在分治算法之外，这些函数还出现在以二进制表示的数的算术算法的直

接分析中。一个著名的例子就是加法器中进位传送链的期望长度的分析，该问题可以追溯到冯·诺依曼而由Knuth[19]完全解决。这些结果直接与串的基本算法分析相关，我们将在第7章看到。二分分治递归与其解的性质更详细的研究见于Allouche和Shallit[2]以及Flajolet和Golin[9]。

习题2.58 给出图2-5中所画出的几个函数的递归。

70 ~ 71

习题2.59 类似于对上面给出的P_N的做法，导出对R_N的递归。

习题2.60 画出递归

$$A_N = A_{\lfloor N/2 \rfloor} + A_{\lceil N/2 \rceil} + \lfloor \lg N \rfloor \qquad (N \geq 2,\ A_1 = 0)$$

的解，其中$1 \leq N \leq 512$。

习题2.61 画出递归

$$B_N = 3B_{\lceil N/2 \rceil} + N \qquad (N \geq 2,\ B_1 = 0)$$

的解，其中$1 \leq N \leq 512$。

习题2.62 画出

$$D_N = D_{\lceil N/2 \rceil} + D_{\lceil N/2 \rceil} + C_N \qquad (N > 1,\ D_1 = 0)$$

的解，其中C_N为

$$C_N = C_{\lceil N/2 \rceil} + C_{\lceil N/2 \rceil} + N \qquad (N > 1,\ C_1 = 0)$$

的解。考虑在每项中将$\lceil N/2 \rceil$改成$\lfloor N/2 \rfloor$所得到的该问题的变体。

习题2.63 取N的二进制表示，将其倒转，并解释它作为一个整数$\rho(N)$的结果。证明$\rho(N)$满足一个分治递归。画出它当$1 \leq N \leq 512$时的图，并解释你看到的现象。

习题2.64 在小于N的数的二进制表示中，那些1的初始串的平均长度是多少？假设所有这样的数都是等可能的。

习题2.65 长为N的随机比特串中，那些1的初始串的平均长度是多少？假设所有这样的串都是等可能的。

习题2.66 在随机比特（可能是无穷的）序列中，那些1的初始串的长度的平均值和方差是多少？

习题2.67 当二进制计数器从0到N增加N次时，所执行的进位的总数是多少？

2.7 一般的分治递归

更一般地，有效的算法和在复杂性研究中的上界常常通过扩展分治算法设计范例按照下列做法得到："把问题分成一些更小（或许有重叠）的子问题，递归地求解这些子问题，然后用所得到的解去求解原问题"。大量各种"分治"递归出现，它们依赖于子问题的个数和相对大小、这些子问题重叠的部分以及为了求解原问题而将它们重新组合起来的开销。确定满足这些递归的函数的渐近增长通常是可能的，不过如上所述，所涉及的函数的周期性和碎段性使得有必要仔细地说明细节。

72

为了寻求一般解，我们从递推公式

$$a(x) = \alpha a(x/\beta) + f(x) \qquad (x > 1, a(x) = 0(x \leq 1))$$

开始，确定一个定义在正实数上的函数。这本质上对应一个分治算法，该算法将一个大小为x的问题分成大小为x/β的α个子问题，并将它们以$f(x)$的代价重新组合。这里，$a(x)$是一个由正实数x定义的函数，从而$a(x/\beta)$是严格定义的。在大部分的应用中，α和β将是整数，虽然我们

在获取解的过程中并不使用这个事实。当然，我们坚持$\beta>1$。

例如，考虑$f(x)=x$的情形，而且限定整数$N=\beta^n$。在这种情形下，我们有

$$a_{\beta^n}=\alpha a_{\beta^{n-1}}+\beta^n \qquad (n>0,\ a_1=0)$$

用α^n除两边，并重复进行（即应用定理2.1），我们得到解

$$a_{\beta^n}=\alpha^n\sum_{1\leqslant j\leqslant n}\left(\frac{\beta}{\alpha}\right)^j$$

现在存在三种情形：如果$\alpha>\beta$，则该和收敛到一个常数；如果$\alpha=\beta$，则它的值是n；如果$\alpha<\beta$，则该和由最后的一项控制，从而是$O(\beta/\alpha)^n$。由于$\alpha^n=(\beta^{\log_\beta\alpha})^n=(\beta^n)^{\log_\beta\alpha}$，这就是说该递归的解当$\alpha>\beta$时为$O(N^{\log_\beta\alpha})$，当$\alpha=\beta$时为$O(N\log N)$，而当$\alpha<\beta$时为$O(N)$。虽然这个解只对$N=\beta^n$成立，但是它解释了在一般情形下所遇到的整体结构。

定理2.5（分治函数） 如果函数$a(x)$满足递归

$$a(x)=\alpha a(x/\beta)+x \quad (x>1,\ a(x)=0(x\leqslant 1))$$

则

$$\text{若 } \alpha<\beta \qquad a(x)\sim\frac{\beta}{\beta-\alpha}x$$

$$\text{若 } \alpha=\beta \qquad a(x)\sim x\log_\beta x$$

$$\text{若 } \alpha>\beta \qquad a(x)\sim\frac{\alpha}{\alpha-\beta}\left(\frac{\beta}{\alpha}\right)^{\{\log_\beta\alpha\}}x^{\log_\beta\alpha}$$

73

证明 基本想法是重复进行递归直到子问题能够使用初始条件时为止，该想法适用所有的分治递归。此处，我们有

$$\begin{aligned}a(x)&=x+\alpha a(x/\beta)\\&=x+\alpha\frac{x}{\beta}+\alpha a(x/\beta^2)\\&=x+\alpha\frac{x}{\beta}+\alpha^2\frac{x}{\beta^2}+\alpha a(x/\beta^3)\end{aligned}$$

等等。在$t=\lfloor\log_\beta x\rfloor$次迭代后，所出现的项$a(x/\beta^t)$可以用0来代替，迭代过程终止。这就得出解的准确表示：

$$a(x)=x\left(1+\frac{\alpha}{\beta}+\cdots+\frac{\alpha^t}{\beta^t}\right)$$

现在，可以考虑三种不同的情况。首先，如果$\alpha<\beta$，则和收敛且

$$a(x)\sim x\sum_{j\geqslant 0}\left(\frac{\alpha}{\beta}\right)^j=\frac{\beta}{\beta-\alpha}x$$

其次，如果$\alpha=\beta$，则和中的每一项都是1，解就是

$$a(x)=x(\lfloor\log_\beta x\rfloor+1)\sim x\log_\beta x$$

第三，如果$\alpha>\beta$，那么和中的最后一项占支配地位，从而

$$a(x)=x\left(\frac{\alpha}{\beta}\right)^{t}\left(1+\frac{\beta}{\alpha}+\cdots+\frac{\beta^{t}}{\alpha^{t}}\right)$$
$$\sim x\frac{\alpha}{\alpha-\beta}\left(\frac{\alpha}{\beta}\right)^{t}$$

如上所述，第三种情形中的表达式的周期特性可以通过把$\log_\beta x$的整数部分和小数部分分开而得以分离，并且写成$t=\lfloor\log_\beta x\rfloor=\log_\beta x-\{\log_\beta x\}$。由此得到

$$x\left(\frac{\alpha}{\beta}\right)^{t}=x\left(\frac{\alpha}{\beta}\right)^{\log_\beta x}\left(\frac{\alpha}{\beta}\right)^{-\{\log_\beta x\}}=x^{\log_\beta\alpha}\left(\frac{\beta}{\alpha}\right)^{\{\log_\beta x\}}$$

74

这是因为$\alpha^{\log_\beta x}=x^{\log_\beta\alpha}$，证明完成。

对于$\alpha<\beta$，在首项中没有周期性能，但是对于$\alpha>\beta$，项$x^{\log_\beta\alpha}$的系数是$\log_\beta x$的周期函数，它是有界的并在$\alpha/(\alpha-\beta)$和$\beta/(\alpha-\beta)$之间振荡。 ■

图2-6阐述了α和β相关的值如何影响函数的渐近增长。图中的方格对应分治算法的问题的大小。最上面的图指出性能如何是线性的，因为问题的大小以指数速度趋向于0，其中，一个问题分裂成两个子问题，而每个子问题是原问题的三分之一大小。中间的图指出整个问题大小是如何平衡的，因此需要一个“log”乘法因子，这里一个问题分裂成三个子问题，而每个子问题是原问题的三分之一大小。最后的图指出整个问题的大小是如何指数增长的，因此最后的项控制着总数，这里一个问题分裂成四个子问题，而每个子问题是原问题的三分之一大小。它显示了渐近增长，反映在一般情形下所发生情况的特征。

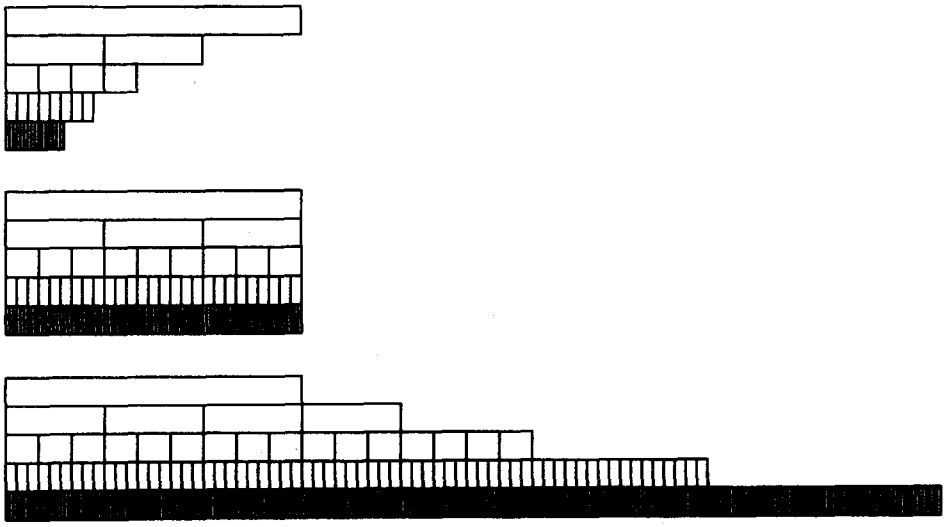


图2-6 $\beta=3$和$\alpha=2,3,4$时分治的情况

为了推广到能够应用到实际情况的地步，我们需要考虑其他的$f(x)$和比精确等于子问题大小的限制要宽松的细分策略（它将使我们回到对整数的递归上来）。对于其他的$f(x)$，我们完全按照上述进行：在最高一层上我们有一个开销$f(x)$的问题，接着有α个开销$f(x/\beta)$的问题，然后有α^2个开销$f(x/\beta^2)$的问题等等，因此总的开销为

75

$$f(x)+\alpha f(x/\beta)+\alpha^2 f(x/\beta^2)+\cdots$$

如上所述，存在三种情况：如果$\alpha>\beta$，那么和中后面的项起主要作用；如果$\alpha=\beta$，则这些项大致相等；而如果$\alpha<\beta$，则前面的项起主要作用。函数f的某些“光滑”约束对于得到精确答案是需要的。例如，如果我们限制f是形如$x^\gamma(\log x)^\delta$的形式——它实际上代表复杂性研究中出现的那些函数的主要部分——则一个类似于上面给出的结论的论断可以用来证明

$$\begin{array}{ll} 若\gamma<\log_\beta\alpha & a(x)\sim c_1x^\gamma(\log x)^\delta \\ 若\gamma=\log_\beta\alpha & a(x)\sim c_2x^\gamma(\log x)^{\delta+1} \\ 若\gamma>\log_\beta\alpha & a(x)=\Theta(x^{\log_\beta\alpha}) \end{array}$$

其中c_1和c_2为相应的常数，它们依赖于α、β和γ。

习题2.68 给出c_1和c_2的显式公式。通过从处理$\delta=0$的情形开始。

直观地看，我们期望甚至当这些子问题几乎（但不必完全）有相同的大小时能够得到同类的结果。事实上，因为“问题的大小”必然是整数，所以我们必须考虑这种情况：当然把一个大小为奇数的文件分成两部分得到几乎而不是完全相同的大小的子问题。而且为了估计$a(x)$的增长，我们不期望必须有$f(x)$的准确的值。当然，我们也对那些只定义在整数上的函数有兴趣。综合起来我们得到一个对于各种算法的分析都是有用的结果。

定理2.6（分治序列） *如果通过把一个大小为n的问题分成β部分，每部分大小为$n/\alpha+O(1)$，并在独立地求解这些子问题时因分割和组合附加开销$f(n)$，而使一个分治算法成功进行，那么当$f(n)=\Theta(n^\gamma(\log n)^\delta)$时，总的开销由*

$$\begin{array}{ll} 若\gamma<\log_\beta\alpha & a_n=\Theta(n^\gamma(\log n)^\delta) \\ 若\gamma=\log_\beta\alpha & a_n=\Theta(n^\gamma(\log n)^{\delta+1}) \\ 若\gamma>\log_\beta\alpha & a_n=\Theta(n^{\log_\beta\alpha}) \end{array}$$

76 *给出。*

证明 一般的策略与上面相同：重复递归直到初始条件满足为止，然后将项集中起来。所涉及的计算是相当复杂的，这里略去。细节可以在Flajolet和Sedgewick[13]中找到。 ■

在复杂性研究中，常常使用更一般的公式表示，因为可以使用的$f(n)$的信息不那么具体。在对$f(n)$光滑性的适当条件下，可以证明

$$\begin{array}{ll} 若f(n)=O(n^{\log_\beta\alpha-\varepsilon}) & a_n=\Theta(n^{\log_\beta\alpha}) \\ 若f(n)=\Theta(n^{\log_\beta\alpha}) & a_n=\Theta(n^{\log_\beta\alpha}\log n) \\ 若f(n)=\Omega(n^{\log_\beta\alpha+\varepsilon}) & a_n=\Theta(f(n)) \end{array}$$

这个结果主要归因于Bentley、Haken和Saxe[4]，类似结果的完善证明也见于[5]。这类结果通常用来证明算法渐近性能的上界和下界，方法是选择$f(n)$给开销适当定界。本书我们的兴趣通常在于对具体的$f(n)$获取更为精确的结果。

习题2.69 画出递归

$$a_N=3a_{\lfloor N/3\rfloor}+N \qquad (N>2,\ a_1=a_2=a_3=1)$$

的解的周期部分，其中$1\leqslant N\leqslant 512$。

习题2.70 对于将一个大小为N的问题分成3部分，且每部分大小或者$\lfloor N/3\rfloor$或者$\lceil N/3\rceil$的其他可能的方法回答前面的问题。

习题2.71 给出递归

$$a(x) = \alpha a_{x/\beta} + 2^x \quad (x > 1, a(x) = 0(x \leq 1))$$

的渐近解。

习题2.72 给出递归

$$a_N = a_{3N/4} + a_{N/4} + N \qquad (N > 2, a_1 = a_2 = a_3 = 1)$$

的渐近解。

习题2.73 给出递归

$$a_N = a_{N/2} + a_{N/4} + N \qquad (N > 2, a_1 = a_2 = a_3 = 1)$$

的渐近解。

习题2.74 考虑递归

$$a_n = a_{f(n)} + a_{g(n)} + a_{h(n)} + 1 \qquad (n > t, a_n = 1(n < t))$$

其中约束为$f(n) + g(n) + h(n) = n$。证明$a_n = \Theta(n)$。 77

习题2.75 考虑递归

$$a_n = a_{f(n)} + a_{g(n)} + 1 \qquad (N > t, a_n = 1(n < t))$$

其中$f(n) + g(n) = n - h(n)$。给出使得能够证明当$n \to \infty$时$a_n/n \to 0$的$h(n)$的最小值。

递推关系自然地对应迭代和递归程序，而它们在算法分析的各种应用中能够很好地为我们服务，因此本章我们综述了能够发生的递推关系的类型和使用它们进行模拟的某些方法。为了能够获取描述重要性能特征的递推关系而足够准确地理解一个算法常常是分析该算法的重要的第一步。给定一个递推关系，即使解析解太难而得不到，我们还是常常可以计算或估计实际应用中所需要的参数的。

存在大量的文献论述“差分方程”和递归，我们从这些文献中尽量选择一些有用的和相关的工具、方法和实例。有些一般的和基本的数学工具处理递归，但是找出解决一个特定递归的适当的途径常常具有挑战性。然而，仔细的分析可以有助于对发生于实践中的大量递归的基本性质的理解。我们可以计算递归的值以得到它的增长率的某些想法；尝试叠缩（重复）递归以获得解的渐近形式的想法；或是寻找一个求和因子、变量变换或一组能够导致准确解的取值表；或者应用诸如自精化方法或扰动方法这样的逼近技术以得到对解的估计。

计算复杂性研究常常依赖于求解为算法性能特征进行估计和定界的递归。特别地，本章末尾的“分治”递归在计算复杂性文献中出现得特别多。大部分这种递归都有类似的结构，它们反映在算法设计中平衡的程度。它们还与数系的性质紧密相关，从而当仔细分析时趋向于展现碎段式的性能。像我们已经看到的这种近似的界对在复杂性证明中上界的获得是适宜（并被广泛使用）的，不过对分析算法的性能则不是必须的，因为它们不是总能 78
够提供足够精确的信息以使我们预报性能。虽然我们必须认识到在解中将涉及周期性和可能的碎段性，但是在拥有关于$f(n)$和分治方法更精确的信息的情况下，我们还是常常能够得到更精确的估计。

我们的讨论一直限制在一个指标N的递归的情况。对于多元的和其他类型的递归，我们打算在介绍求解这些递归的更先进的工具之后再进行讨论。

在算法性能特征的研究中递归以一种自然的方式产生。当我们对复杂算法进行详细分析的时候，会遇到求解相当复杂的递归。在下一章，我们介绍生成函数，它是算法分析的基础。

生成函数不仅能够帮助我们求解递归，而且它们和算法在一个更高的层次上有着直接的联系，使我们得到由深入到许多应用内部的递归所描述的详细的结构。

参考文献

1. A. V. AHO AND N. J. A. SLOANE. "Some doubly exponential sequences," *Fibonacci Quarterly* **11**, 1973, 429–437.
2. J.-P. ALLOUCHE AND J. SHALLIT. "The ring of k-regular sequences," *Theoretical Computer Science* **98**, 1992, 163–197.
3. C. M. BENDER AND S. A. ORSZAG. *Advanced Mathematical Methods for Scientists and Engineers*, McGraw-Hill, New York, 1978.
4. J. L. BENTLEY, D. HAKEN, AND J. B. SAXE. "A general method for solving divide-and-conquer recurrences," *SIGACT News*, Fall 1980, 36–44.
5. T. H. CORMEN, C. E. LEISERSON, AND R. L. RIVEST. *Introduction to Algorithms*, MIT Press, New York, 1990.
6. N. G. DE BRUIJN. *Asymptotic Methods in Analysis*, Dover Publications, New York, 1981.
7. H. DELANGE. "Sur la fonction sommatoire de la fonction somme des chiffres," *L'enseignement Mathématique* **XXI**, 1975, 31–47.
8. P. FLAJOLET AND M. GOLIN. "Exact asymptotics of divide–and–conquer recurrences," in *Automata, Languages, and Programming*, A. Lingas, R. Karlsson, and S. Carlsson, ed., Lecture Notes in Computer Science #700, Springer Verlag, Berlin, 1993, 137–149.
9. P. FLAJOLET AND M. GOLIN. "Mellin transforms and asymptotics: the mergesort recurrence," *Acta Informatica* **31**, 1994, 673–696.
10. P. FLAJOLET, P. GRABNER, AND P. KIRSCHENHOFER. "Mellin Transforms and asymptotics: digital sums," *Theoretical Computer Science* **123**, 1994, 291–314.
11. P. FLAJOLET, J.-C. RAOULT, AND J. VUILLEMIN. "The number of registers required to evaluate arithmetic expressions," *Theoretical Computer Science* **9**, 1979, 99–125.
12. P. FLAJOLET AND R. SEDGEWICK. *Analytic Combinatorics*, in preparation.
13. P. FLAJOLET AND R. SEDGEWICK. "Asymptotic behavior of divide-and-conquer recurrences," in preparation.
14. R. L. GRAHAM, D. E. KNUTH, AND O. PATASHNIK. *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1988.
15. D. H. GREENE AND D. E. KNUTH. *Mathematics for the Analysis of Algorithms*, Birkhäuser, Boston, 1981.
16. P. HENRICI. *Applied and Computational Complex Analysis*, 3 volumes, John Wiley, New York, 1977.
17. D. E. KNUTH. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.
18. D. E. KNUTH. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
19. D. E. KNUTH. "The average time for carry propagation," *Indagationes*

79

Mathematicae **40**, 1978, 238–242.

20. D. E. KNUTH AND A. SCHÖNHAGE. "The expected linearity of a simple equivalence algorithm," *Theoretical Computer Science* **6**, 1978, 281–315.
21. G. LUEKER. "Some techniques for solving recurrences," *Computing Surveys* **12**, 1980, 419–436.
22. Z. A. MELZAK. *Companion to concrete mathematics*, John Wiley, New York, 1968.
23. J. RIORDAN. *Combinatorial Identities*, John Wiley, New York, 1968.
24. R. SEDGEWICK. "The analysis of quicksort programs," *Acta Informatica* **7**, 1977, 327–355.
25. A. YAO. "On random 2–3 trees," *Acta Informatica* **9**, 1978, 159–170.

80

第3章 生成函数

在这一章，我们将介绍在进行算法平均情形分析时所使用的主要概念：生成函数。这些数学材料是本书其他部分的基础，因此我们集中于核心方法的描述，而不专注于具体的应用，不过有时也会从算法性质中涉及一些例子。

在定义了基本术语“常规”生成函数与“指数型”生成函数之后，我们将叙述生成函数在解递推关系式方面的应用，其中包括对必要的数学工具的讨论。对于常规与指数型两类生成函数，我们将考查许多在实际应用中产生的基本函数，讨论它们的性质和基本操作方式。我们将讨论大量的例子，其中包括在第一章介绍的采用三数中值快速排序递归的实现细节。

然后我们将详细讨论生成函数在组合结构计数中可能发挥的作用，其中最重要的是关于“符号方法”的介绍，这种方法能使我们探讨所分析的对象和相应的生成函数之间的关系。为了进一步阐述相关的概念，从第5章到第8章，我们考查若干例子，包括熟知的计数二叉树的例子。在这前后，我们还将考查拉格朗日反演定理，这是对符号方法在解决具有递归结构问题方面的一个很好的补充。

一般地，我们所关心的不仅是组合结构的计数问题，而且还要分析它们的性质。为此，我们还要考查如何使用“双变量”生成函数以及如何与“概率”生成函数的使用相联系。

在本章的最后，将讨论各种特殊类型的生成函数，它们出现在算法分析的各种应用之中。

先前各章的目的是让读者充分认识到递归在算法分析中所发挥的关键性的作用。本章的目标之一是让读者确信：不仅许多基本的递归问题可以很容易地用生成函数求解，而且在一些试图回避递归的问题中，生成函数仍能发挥重要的作用。生成函数不仅是求解递归和计算矩的必需的技巧性很强的工具，而且它在以下两者之间建立了一种自然的联系：我们所研究的算法对象和探讨其性质所必需的解析方法。生成函数可提供两方面的服务：既可作为组合学工具进行计数的研究，又可作为解析工具对所感兴趣的量做精确的估计。

在本书中我们通过使用生成函数直接得到组合学性质的一些例子进行阐述，这些组合学性质过于复杂有时不能方便地用递归描述，因此本书所介绍的方法只是一个起点，关于更一般的组合技巧的研究请参看文献[3]。

鉴于生成函数对于全书内容的核心作用，我们给出了在对生成函数进行操作时所涉及的基本性质和常用技巧，并对本章大部分最重要的生成函数提供了一个清单供读者参考。我们介绍了来自组合学和算法分析领域的大量的内容充实的材料，尽管对每个特定的论题我们只作了较为简明的处理。关于这些论题的更完全的讨论可以在第5章到第8章的各种应用中找到，在本章最后给出了其他的参考文献目录，其中主要是[1]、[4]、[5]和[17]。

3.1 常规生成函数

在上一章中我们看到，算法分析的目的是要导出序列a_0，a_1，a_2，…中诸项的值的具体表达式，这个序列是度量某种性能的参数。本章我们将看到用一个单一的数学对象表示整个序列的好处。

定义 给定序列a_0，a_1，a_2，…，a_k，…，我们称下面的函数

$$A(z)=\sum_{k\geq 0}a_k z^k$$

为该序列的常规生成函数(OGF)，用记号$[z^k]A(z)$表示系数a_k。

表3-1 基本的常规生成函数

$1,1,1,1,\cdots,1,\cdots$	$\frac{1}{1-z}=\sum_{N\geq 0}z^N$
$0,1,2,3,4,\cdots,N,\cdots$	$\frac{z}{(1-z)^2}=\sum_{N\geq 1}Nz^N$
$0,0,1,3,6,10\cdots,\binom{N}{2},\cdots$	$\frac{z^2}{(1-z)^3}=\sum_{N\geq 2}\binom{N}{2}z^N$
$0,\cdots,0,1,M+1,\cdots,\binom{N}{M},\cdots$	$\frac{z^M}{(1-z)^{M+1}}=\sum_{N\geq M}\binom{N}{M}z^N$
$1,M,\binom{M}{2}\cdots,\binom{M}{N},\cdots,M,1$	$(1+z)^M=\sum_{N\geq 0}\binom{M}{N}z^N$
$1,M+1,\binom{M+2}{2},\binom{M+3}{3},\cdots$	$\frac{1}{(1-z)^{M+1}}=\sum_{N\geq 0}\binom{N+M}{N}z^N$
$1,0,1,0,\cdots,1,0,\cdots$	$\frac{1}{1-z^2}=\sum_{N\geq 0}z^{2N}$
$1,c,c^2,c^3,\cdots,c^N,\cdots$	$\frac{1}{1-cz}=\sum_{N\geq 0}c^N z^N$
$1,1,\frac{1}{2!},\frac{1}{3!},\frac{1}{4!},\cdots,\frac{1}{N!},\cdots$	$e^z=\sum_{N\geq 0}\frac{z^N}{N!}$
$0,1,\frac{1}{2},\frac{1}{3},\frac{1}{4},\cdots,\frac{1}{N},\cdots$	$\ln\frac{1}{1-z}=\sum_{N\geq 1}\frac{z^N}{N}$
$0,1,1+\frac{1}{2},1+\frac{1}{2}+\frac{1}{3},\cdots,H_N,\cdots$	$\frac{1}{1-z}\ln\frac{1}{1-z}=\sum_{N\geq 1}H_N z^N$
$0,0,1,3\left(\frac{1}{2}+\frac{1}{3}\right),4\left(\frac{1}{2}+\frac{1}{3}+\frac{1}{4}\right),\cdots$	$\frac{z}{(1-z)^2}\ln\frac{1}{1-z}=\sum_{N\geq 0}N(H_N-1)z^N$

82~83

在表3-1中，给出了一些基本生成函数及所对应的序列。下面将介绍如何推导这些生成函数，以及对这些函数的各种处理方式。表3-1中给出的常规生成函数是最基本的一些函数，它们在算法分析中经常会出现。每个序列都有多种表述方式（例如用简单的递推关系），但下面将看到，直接用生成函数表示它们会具有显著的优势。

定义中的和可能收敛，也可能不收敛，我们暂且忽略收敛性的问题，这里有两个理由：首先，我们在这些生成函数上所做的操作都是关于幂级数形式上的规范操作，即使是在收敛性不存在的情况下。其次，在我们的分析中所产生的序列通常收敛性都是有保证的，即至少对某个充分小的z，级数是收敛的。在算法分析的大量应用中，在典型分析的第一部分，我们通过细致地对照可以探讨幂级数与算法之间的形式上的关系，从而导出生成函数的显式公式。在典型分析的第二部分，我们可以得到生成函数详细的解析性质（这时收敛性将发挥重要的作用），从而导出描述算法基本性质的显式公式。

给定生成函数$A(z)=\sum_{k\geq 0}a_k z^k$和$B(z)=\sum_{k\geq 0}b_k z^k$，它们分别表示序列$\{a_0, a_1, a_2, \cdots, a_k, \cdots, \}$和$\{b_0, b_1, b_2, \cdots, b_k, \cdots, \}$，我们可以通过一些简单的变换得到其他一些序列的生成函数，在表3-2中给出了几种这样的一些操作。关于应用这些操作的例子可通过表3-1

中所列出各项之间的关系找到。

定理3.1(OGF操作) 如果两个序列$a_0, a_1, a_2, \cdots, a_k, \cdots$和$b_0, b_1, b_2, \cdots, b_k, \cdots$分别由常规生成函数$A(z) = \sum_{k\geq 0} a_k z^k$和$B(z) = \sum_{k\geq 0} b_k z^k$表示，那么表3-2给出的各种运算所产生的常规生成函数分别表示指定的序列。特别地，

$A(z) + B(z)$ 是表示$a_0 + b_0, a_1 + b_1, a_2 + b_2, \cdots$的OGF

$zA(z)$ 是表示$0, a_0, a_1, a_2, \cdots$的OGF

$A'(z)$ 是表示$a_1, 2a_2, 3a_3, \cdots$的OGF

$A(z)B(z)$ 是表示$a_0b_0, a_0b_1 + a_1b_0, a_0b_2 + a_1b_1 + a_2b_0, \cdots$的OGF

证明 上述大多数公式都是初等的表述，直接观察便可以验证。对于卷积（convolution）运算（以及作为特例的部分和）容易通过按同次幂求和导出

$$\begin{aligned} A(z)B(z) &= \sum_{i\geq 0} a_i z^i \sum_{j\geq 0} b_j z^j \\ &= \sum_{i,j\geq 0} a_i b_j z^{i+j} \\ &= \sum_{n\geq 0}\left(\sum_{0\leq k\leq n} a_k b_{n-k}\right) z^n \end{aligned}$$

在这个公式中，取$B(z) = 1/(1 - z)$，就得到部分和运算。以后我们将看到，卷积运算在生成函数的运算中起着特殊的作用。 ■

推论 表示调和数的常规生成函数为

$$\sum_{N\geq 1} H_N z^N = \frac{1}{1-z}\ln\frac{1}{1-z}$$

证明 从$1/(1 - z)$出发（它是表示1, 1, 1,…的OGF）,积分（得到表示0, 1, 1/2, 1/3, … 1/k, …的OGF）然后乘以$1/(1 - z)$即可。类似的例子可在表3-1所列出各项之间的关系中找到。 ■

表3-2 关于常规生成函数的运算

$A(z) = \sum_{n\geq 0} a_n z^n$	$a_0, a_1, a_2, \cdots, a_n, \cdots,$
$B(z) = \sum_{n\geq 0} b_n z^n$	$b_0, b_1, b_2, \cdots, b_n, \cdots,$
右移	
$zA(z) = \sum_{n\geq 1} a_{n-1} z^n$	$0, a_0, a_1, a_2, \cdots, a_{n-1}, \cdots,$
左移	
$\frac{A(z) - a_0}{z} = \sum_{n\geq 0} a_{n+1} z^n$	$a_1, a_2, a_3, \cdots, a_{n+1}, \cdots,$
下标乘（微分）	
$A'(z) = \sum_{n\geq 0} (n+1)a_{n+1} z^n$	$a_1, 2a_2, \cdots, (n+1)a_{n+1}, \cdots,$
下标除（积分）	
$\int_0^z A(t)dt = \sum_{n\geq 1} \frac{a_{n-1}}{n} z^n$	$0, a_0, \frac{a_1}{2}, \frac{a_2}{3} \cdots, \frac{a_{n-1}}{n}, \cdots,$

（续）

比例因子	
$A(\lambda z)=\sum_{n\geq 0}\lambda^n a_n z^n$	$a_0,\lambda a_1,\lambda^2 a_2,\cdots,\lambda^n a_n,\cdots,$
相加	
$A(z)+B(z)=\sum_{n\geq 0}(a_n+b_n)z^n$	$a_0+b_0,\cdots,a_n+b_n,\cdots,$
差分	
$(1-z)A(z)=a_0+\sum_{n\geq 1}(a_n-a_{n-1})z^n$	$a_0,a_1-a_0,\cdots,a_n-a_{n-1},\cdots,$
卷积	
$A(z)B(z)=\sum_{n\geq 0}\left(\sum_{0\leq k\leq n}a_k b_{n-k}\right)z^n$	$a_0b_0,a_1b_0+a_0b_1,\cdots,\sum_{0\leq k\leq n}a_k b_{n-k},\cdots,$
部分和	
$\frac{A(z)}{1-z}=\sum_{n\geq 0}\left(\sum_{0\leq k\leq n}a_k\right)z^n$	$a_1,a_1+a_2,\cdots,\sum_{0\leq k\leq n}a_k,\cdots,$

对于还不熟悉生成函数的读者，我们建议做以下习题，以便掌握应用这些变换的基本能力。

习题3.1 求以下每个序列的OGF

$$\{2^{k+1}\}_{k\geq 0}\qquad \{k2^{k+1}\}_{k\geq 0}\qquad \{kH_k\}_{k\geq 1}\qquad \{k^3\}_{k\geq 2}$$

习题3.2 求以下每个OGF对应的$[z^N]$

$$\frac{1}{(1-3z)^4},\qquad (1-z)^2\ln\frac{1}{1-z},\qquad \frac{1}{(1-2z^2)^2}$$

习题3.3 将表示调和数的OGF微分，以验证表3-1的最后一行。

习题3.4 证明

$$\sum_{1\leq k\leq N}H_k=(N+1)(H_{N+1}-1)$$

习题3.5 通过对下式用两种不同的方式分解因子（然后做相应的卷积运算），

$$\frac{z^M}{(1-z)^{M+1}}\ln\frac{1}{1-z}$$

84 ~ 86

证明调和数与二项式系数所满足的一般的恒等式。

习题3.6 求对应于以下序列的OGF

$$\left\{\frac{1}{k(n-k)}\right\}_{k\geq 1}$$

（提示：使用部分分式）

习题3.7 求对应于以下序列的OGF

$$\{H_k/k\}_{k\geq 1}$$

习题3.8 求以下每个OGF对应的 $[z^N]$

$$\frac{1}{1-z}\left(\ln\frac{1}{1-z}\right)^2\quad 与\quad \left(\ln\frac{1}{1-z}\right)^3$$

我们用记号

$$H_N^{(2)} \equiv 1+\frac{1}{2^2}+\frac{1}{3^2}+\cdots+\frac{1}{N^2}$$

表示上述展开式生成的“广义调和数”。

对于我们在算法分析中遇到的许多序列，用上述的初等运算就足够了，尽管在许多算法中还需要更先进的工具。由此出发可以看出，显然算法分析是围绕着两个问题展开的：一方面要确定一个显式公式作为给定的序列的生成函数；反过来还要利用生成函数的表达式确定表示序列中元素的精确公式。在后面以及第5章～第8章中，我们将看到许多有关的例子。

从形式上讲，我们可以用任何一组核函数$w_k(z)$来定义“生成函数”

$$A(z)=\sum_{k\geq 0} a_k w_k(z)$$

它封装了序列$a_0, a_1, \cdots, a_k, \cdots$。尽管本书几乎只考虑核函数$z^k$与$z^k/k!$（详见下节），但在算法分析中其他类型的核函数确实也会偶尔使用。我们将在本章最后简短地对它们进行讨论。 87

3.2 指数生成函数

对于某些序列，使用带有规范化因子的生成函数将更方便一些。

定义 给定序列a_0，a_1，a_2，…，a_k，…，我们称下面的函数

$$A(z)=\sum_{k\geq 0} a_k \frac{z^k}{k!}$$

为该序列的指数生成函数(EGF)。我们用记号$k!\,[z^k]A(z)$表示系数a_k。

表3-3 基本的指数生成函数

$1,1,1,1,\cdots,1,\cdots$	$e^z=\sum_{N\geq 0}\frac{z^N}{N!}$
$0,1,2,3,4,\cdots,N,\cdots$	$ze^z=\sum_{N\geq 1}\frac{z^N}{(N-1)!}$
$0,0,1,3,6,10,\cdots,\binom{N}{2},\cdots$	$\frac{1}{2}z^2e^z=\frac{1}{2}\sum_{N\geq 2}\frac{z^N}{(N-2)!}$
$0,\cdots,0,1,M+1,\cdots,\binom{N}{M},\cdots$	$\frac{1}{M!}z^Me^z=\frac{1}{M!}\sum_{N\geq M}\frac{z^N}{(N-M)!}$
$1,0,1,0,\cdots,1,0,\cdots$	$\frac{1}{2}(e^z+e^{-z})=\sum_{N\geq 0}\frac{1+(-1)^N}{2}\frac{z^N}{N!}$
$1,c,c^2,c^3,\cdots,c^N,\cdots$	$e^{cz}=\sum_{N\geq 0}\frac{c^Nz^N}{N!}$
$0,1,\frac{1}{2},\frac{1}{3},\cdots,\frac{1}{N+1},\cdots$	$\frac{e^z-1}{z}=\sum_{N\geq 0}\frac{z^N}{(N+1)!}$
$1,2,6,24,\cdots,N!,\cdots$	$\frac{1}{1-z}=\sum_{N\geq 0}\frac{N!z^N}{N!}$

88

关于$\{a_k\}$的指数型生成函数(EGF)与关于$\{a_k/k!\}$的常规生成函数相比，并没有新的内容，但它在组合学和算法分析中基于一些特定的简单原因而产生。假定系数a_k表示与一种具有k项结构相关的计数，进一步假定k项已被“标记”，使得每一项都有一个不同的标记。在某些场合下，这些标记是有关联的（这时用EGF比较合适），而在另外一些情况下各标记互不关联

（这时用OGF比较合适）。因子$k!$表示这些被标记项的排列的总数，如果他们没有被标记就不能相互区分。我们在3.9节将详细地讨论这个问题，届时我们将看到用于相关的生成函数和组合学对象的“符号方法”。这里，我们给出这种解释就是用以验证我们将要详细介绍的关于EGF的性质，这些性质都进行过深入的研究，因为它们有广泛的应用背景。

表3-3给出了若干个本书后面将要用到的基本指数型生成函数，表3-4给出了关于EGF的一些基本的运算。注意对于EGF的左移（或右移）运算与关于OGF的下标乘（或除）运算是相同的（参看表3-2），反之亦然。如同OGF，在表3-4列出了对于表3-3中的基本函数所进行的基本运算的应用，这些运算产生了大量在实际应用中涉及到的EGF，读者通过后面的习题将会熟悉这些函数。与OGF一样，我们很容易确立这些基本运算的有效性。

定理3.2（对EGF的操作） 如果两个序列$a_0, a_1, a_2, \cdots, a_k, \cdots$和$b_0, b_1, b_2, \cdots, b_k, \cdots$分别由指数生成函数$A(z) = \sum_{k\geq 0} a_k z^k/k!$和$B(z) = \sum_{k\geq 0} b_k z^k/k!$表示，那么表3-4给出的各种运算所产生的指数生成函数分别表示指定的序列。特别地，

$A(z) + B(z)$ 是表示$a_0 + b_0, a_1 + b_1, a_2 + b_2, \cdots$的EGF

$A'(z)$ 是表示$a_1, a_2, a_3, \cdots$的EGF

$zA(z)$ 是表示$0, a_1, 2a_2, 3a_3, \cdots$的EGF

$A(z)B(z)$ 是表示$a_0b_0, a_0b_1 + a_1b_0, a_0b_2 + 2a_1b_1 + a_2b_0, \cdots$的EGF

89

证明 如同定理3.1，这些运算都是初等的，直接观察便可以验证，而对于二项卷积，也很容易用OGF的卷积来验证：

$$A(z)B(z) = \sum_{n\geq 0}\sum_{0\leq k\leq n} \frac{a_k}{k!}\frac{b_{n-k}}{(n-k)!} z^n$$

$$= \sum_{n\geq 0}\sum_{0\leq k\leq n} \binom{n}{k} a_k b_{n-k} \frac{z^n}{n!}$$

■

表3-4 关于指数型生成函数的运算

$A(z) = \sum_{n\geq 0} a_n \frac{z^n}{n!}$	$a_0, a_1, a_2, \cdots, a_n, \cdots,$
$B(z) = \sum_{n\geq 0} b_n \frac{z^n}{n!}$	$b_0, b_1, b_2, \cdots, b_n, \cdots,$
右移（积分）	
$\int_0^z A(t)\mathrm{d}t = \sum_{n\geq 1} a_{n-1} \frac{z^n}{n!}$	$0, a_0, a_1, \cdots a_{n-1}, \cdots,$
左移（微分）	
$A'(z) = \sum_{n\geq 0} a_{n+1} \frac{z^n}{n!}$	$a_1, a_2, a_3, \cdots, a_{n+1}, \cdots,$
下标乘	
$zA(z) = \sum_{n\geq 0} na_{n-1} \frac{z^n}{n!}$	$0, a_0, 2a_1, 3a, \cdots, na_{n-1}, \cdots,$
下标除	
$(A(z) - A(0))/z = \sum_{n\geq 1} \frac{a_{n+1}}{n+1} \frac{z^n}{n!}$	$a_1, \frac{a_2}{2}, \frac{a_3}{3} \cdots, \frac{a_{n+1}}{n+1}, \cdots,$
相加	
$A(z) + B(z) = \sum_{n\geq 0} (a_n + b_n) \frac{z^n}{n!}$	$a_0 + b_0, \cdots, a_n + b_n, \cdots,$

（续）

差分 $A'(z)-A(z)=\sum_{n\geq 0}(a_{n+1}-a_n)\frac{z^n}{n!}$	$a_1-a_0, a_2-a_1, \cdots, a_{n+1}-a_n, \cdots,$
二项卷积 $A(z)B(z)=\sum_{n\geq 0}\left(\sum_{0\leq k\leq n}\binom{n}{k}a_k b_{n-k}\right)\frac{z^n}{n!}$	$a_0b_0, a_1b_0+a_0b_1, \cdots, \sum_{0\leq k\leq n}\binom{n}{k}a_k b_{n-k}, \cdots,$
二项和 $e^z A(z)=\sum_{n\geq 0}\left(\sum_{0\leq k\leq n}\binom{n}{k}a_k\right)\frac{z^n}{n!}$	$a_0, a_0+a_1, \cdots, \sum_{0\leq k\leq n}\binom{n}{k}a_k, \cdots,$

习题3.9 求以下每个序列的EGF

$$\{2^{k+1}\}_{k\geq 0} \qquad \{k2^{k+1}\}_{k\geq 0} \qquad \{k^3\}_{k\geq 2}$$

习题3.10 分别求关于序列1，3，5，7，…，与0，2，4，6，…，的EGF。

习题3.11 求以下每个EGF对应的$N![z^N]A(z)$

$$A(z)=\frac{1}{1-z}\ln\frac{1}{1-z}, A(z)=\left(\ln\frac{1}{1-z}\right)^2, \qquad A(z)=e^{z+z^2}$$

习题3.12 证明

$$N![z^N]e^z\int_0^z\frac{1-e^{-t}}{t}\mathrm{d}t=H_N$$

（提示：对于EGF $H(z)=\sum_{N\geq 0}H_N z^N/N!$构造一个一阶常微分方程）。

究竟是使用OGF还是使用EGF能够得到一个问题最方便的解，并不总是很明显的。有时一个方式可能导致简单的求解而另一个将需要使用很困难的技巧，有时两个都很好用。而对于今后我们所遇到的许多组合学和算法分析中的问题，使用OGF还是使用EGF，根据问题的结构往往还是容易选择的。此外，从解析函数的角度，人们往往更感兴趣的是：是否可以将表示一个序列的 OGF自动地转换成表示同一序列的EGF，或者反过来？（答案是肯定的，只要通过拉普拉斯变换，见习题3.14。）本书中，将讨论许多涉及OGF和EGF两类应用的例子。

90 ~ 91

习题3.13 给定序列$\{a_k\}$的EGF $A(z)$，求以下序列的EGF:

$$\left\{\sum_{0\leq k\leq N}N!\frac{a_k}{k!}\right\}$$

习题3.14 给定关于序列$\{a_k\}$的EGF $A(z)$，证明如果以下积分存在，那么该序列的OGF就可以由以下积分给出

$$\int_0^\infty A(zt)e^{-t}\mathrm{d}t$$

对表3-1和表3-3中的序列验证这个结论。

3.3 利用生成函数求解递归

下面，我们考查生成函数在求解递归中所能发挥的作用。当描述算法的某些基本性质的递推关系导出以后，作为算法分析的经典处理的第二步，使用生成函数求解这个递归。一些读者可能已经熟悉这些处理方法，因为它已被广泛地使用并且具有一些最基本的特征。不过，

我们后面会看到常常能够避免递归而直接用生成函数进行处理。

生成函数提供了用于求解许多递推关系的很机械的方法。给定一个表示某序列$\{a_n\}_{n\geq 0}$的递归，通常按以下步骤进行求解:

- 在递推式的两边乘以z^n，然后关于n求和。
- 处理所得的各个和，导出一个关于OGF的函数方程。
- 解这个函数方程，导出OGF的显式公式。
- 将OGF展开为一个幂级数，从而得到系数表达式（这些系数就是原来的序列中的元素）。

同样的方法也适用于EGF，只是两边乘以$z^n/n!$，然后在第一步关于n求和。究竟使用OGF还是EGF更方便，依赖于递归。

这种方法的一个最直接的例子是求解常系数线性递归（参见第2章）。

平凡线性递归。解下述递归

92

$$a_n = a_{n-1} + 1 \text{（对所有的}n \geq 1, a_0 = 0\text{）}$$

首先，两边乘以z^n，然后关于n求和，得到:

$$\sum_{n\geq 1} a_n z^n = \sum_{n\geq 1} a_{n-1} z^n + \frac{z}{1-z}$$

由于生成函数$A(z) = \sum_{n\geq 0} a_n z^n$，上述方程可表示为

$$A(z) = zA(z) + \frac{z}{1-z}$$

或$A(z) = z/(1-z)^2$，于是$a_n = n$就是所要求的。

简单指数型递归。为了解下述递归:

$$a_n = 2a_{n-1} + 1 \text{（对所有的}n \geq 1, a_0 = 1\text{）}$$

我们按照上述方法处理，发现相应的生成函数$A(z) = \sum_{n\geq 0} a_n z^n$满足

$$A(z) - 1 = 2zA(z) + \frac{z}{1-z}$$

由此可导出

$$A(z) = \frac{1}{(1-z)(1-2z)}$$

由表3-1可知，$1/(1-2z)$是关于序列$\{2^n\}$的生成函数，再由表3-2可知，乘以$1/(1-z)$后所对应的部分和为

$$a_n = \sum_{0\leq k\leq n} 2^k = 2^{n+1} - 1$$

部分分式。求解上述问题的系数还有一种方法，使用$A(z)$的部分分式展开式，这也是为了解决更困难的问题做准备。利用分母中的因子，生成函数可表示为两个分式之和:

$$\frac{1}{(1-z)(1-2z)} = \frac{c_0}{1-2z} + \frac{c_1}{1-z}$$

其中c_0，c_1是两个待定常数，交叉相乘可以看出，这两个常数满足方程组

$$c_0 + c_1 = 1$$

93

$$-c_0 - 2c_1 = 0$$

由此可得$c_0 = 2, c_1 = -1$。于是，

$$[z^n]\frac{1}{(1-z)(1-2z)}=[z^n]\left(\frac{2}{1-2z}-\frac{1}{1-z}\right)=2^{n+1}-1$$

这一技巧可用于任何分母为多项式的情形，从而可作为下面要讨论的解高阶线性递归的一般方法。

斐波那契数列。关于斐波那契数列

$$F_n=F_{n-1}+F_{n-2}\qquad(n>1,F_0=0,F_1=1)$$

的生成函数$F(z)=\sum_{k\geq 0}F_kz^k$满足

$$F(z)=zF(z)+z^2F(z)+z$$

由此可导出

$$F(z)=\frac{z}{1-z-z^2}=\frac{1}{\sqrt{5}}\left(\frac{1}{1-\phi z}-\frac{1}{1-\hat{\phi}z}\right)$$

利用部分分式，以及$1-z-z^2$可分解因子为$(1-z\phi)(1-z\hat{\phi})$，其中，

$$\phi=\frac{1+\sqrt{5}}{2},\quad \hat{\phi}=\frac{1-\sqrt{5}}{2}$$

它们是$1-z-z^2$的根的相反数。于是由表3-4可以直接得到

$$F_n=\frac{1}{\sqrt{5}}\left(\phi^n-\hat{\phi}^n\right)$$

当然，这与第2章已经给出的推导紧密相关，我们下面以一般项的形式考查这种关系。

习题3.15 求斐波那契数列的EGF。

高阶线性递归。生成函数可以使第2章所介绍的“因子分解”的过程更容易实现，这一过程用于求解常系数高阶线性递归。递归的因子分解对应于在生成函数的分母中出现的多项式的因子分解，这可导致一个部分分式从而得到一个显式的解。例如，递归

$$a_n=5a_{n-1}-6a_{n-2}\ (\text{对所有的}n>1,a_0=0,a_1=1)$$

94

意味着生成函数$a(z)=\sum_{n\geq 0}a_nz^n$为

$$a(z)=\frac{z}{1-5z+6z^2}=\frac{z}{(1-3z)(1-2z)}=\frac{1}{1-3z}-\frac{1}{1-2z}$$

于是必有$a_n=3^n-2^n$。

习题3.16 利用生成函数解以下递归

$a_n=-a_{n-1}+6a_{n-2}$，其中$n>1$且$a_0=0,a_1=1$；

$a_n=11a_{n-2}-6a_{n-3}$，其中$n>2$且$a_0=0,a_1=a_2=1$；

$a_n=3a_{n-1}-4a_{n-2}$，其中$n>1$且$a_0=0,a_1=1$；

$a_n=a_{n-1}-a_{n-2}$，其中$n>1$且$a_0=0,a_1=1$。

一般地，生成函数的显式表达式是两个多项式之比，按照分母多项式的根所做的部分分式展开式将导致关于这些根的幂的一个表达式。按这种思路，做更严格的推导，就可以得到定理2.2的证明。

定理3.3 (*表示线性递归的OGF*) *如果a_n满足递归*：

$$a_n=x_1a_{n-1}+x_2a_{n-2}+\cdots+x_ta_{n-t}$$

其中$n \geqslant t$，则生成函数$a(z) = \sum_{n \geqslant 0} a_n z^n$是有理函数$a(z) = f(z)/g(z)$，其中分母多项式是$g(z) = 1 - x_1 z - x_2 z^2 - \cdots - x_t z^t$，而分子多项式由初值$a_0, a_1, \cdots, a_{t-1}$确定。

证明 证明按照本节开始所介绍的解递归的一般范例，在递推式的两边乘以z^n，然后关于$n \geqslant t$求和得到

$$\sum_{n \geqslant t} a_n z^n = x_1 \sum_{n \geqslant t} a_{n-1} z^n + \cdots + x_t \sum_{n \geqslant t} a_{n-t} z^n$$

左边可表示为用$a(z)$减去一个初值的生成多项式，右边第一项可表示为用$za(z)$减去一个多项式等等。于是$a(z)$满足

95

$$a(z) - u_0(z) = (x_1 z a(z) - u_1(z)) + \cdots + (x_t z^t a(z) - u_t(z))$$

其中，多项式$u_0(z), u_1(z), \cdots, u_t(z)$至多是$t-1$次的，其系数仅依赖于初值$a_0, a_1, \cdots, a_{t-1}$，这个函数方程是线性的。

求解关于$a(z)$的函数方程，可得显式形式$a(z) = f(z)/g(z)$，其中$g(z)$具有前面已提到的形式，而

$$f(z) \equiv u_0(z) - u_1(z) - \cdots - u_t(z)$$

仅依赖于递归的初值，且次数小于t。 ■

由上述的一般形式可直接导出$f(z)$关于对初始条件依赖关系的公式，即由$f(z) = a(z)g(z)$及f的次数小于t，因此一定有

$$f(z) = g(z) \sum_{0 \leqslant n < t} a_n z^n \qquad (\bmod z^t)$$

这就给出了计算$f(z)$的系数的快捷方式，许多递归利用它可以快速准确地求解。

简单的例子。 解下面的递归

$$a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3} \qquad (n > 2, a_0 = 0, a_1 = a_2 = 1)$$

先计算:

$$g(z) = 1 - 2z - z^2 + 2z^3 = (1-z)(1+z)(1-2z)$$

然后利用初始条件，可得

$$\begin{aligned} f(z) &= (z + z^2)(1 - 2z - z^2 + 2z^3) \qquad (\bmod z^3) \\ &= (z - z^2) = z(1-z) \end{aligned}$$

由此可得

$$a(z) = \frac{f(z)}{g(z)} = \frac{z}{(1+z)(1-2z)} = \frac{1}{3}\left(\frac{1}{1-2z} - \frac{1}{1+z}\right)$$

96 于是，有 $a_n = \frac{1}{3}(2^n - (-1)^n)$。

约简。 在上面的递归中，因子$1 - z$被约去了，于是在解中没有出现常数项。考虑同样的递归，但我们采用不同的初始条件

$$a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3} \qquad (n > 2,\ a_0 = a_1 = a_2 = 1)$$

函数$g(z)$与上面相同，但是

$$\begin{aligned} f(z) &= (1 + z + z^2)(1 - 2z - z^2 + 2z^3) \quad (\bmod z^3) \\ &= (1 - z - 2z^2) = (1-2z)(1+z) \end{aligned}$$

在这种情况下，我们约去了一个平凡解$a(z)=f(z)/g(z)=1/(1-z)$，因而对所有$n\geqslant 0$,都有$a_n=1$。通过导出约掉因子的方式，初始条件可能对解的增长率产生显著的影响。

我们约定，$g(z)$分解因子后具有以下的形式

$$g(z)=(1-\beta_1 z)\cdot(1-\beta_2 z)\cdots(1-\beta_n z)$$

这种约定显得更自然一些。如果多项式$g(z)$满足$g(0)=1$（当$g(z)$像上面那样由递归导出时，通常是这样的），则它的所有根的积是1，而上式中的β_1，β_2，$\cdots\beta_n$恰好是各根的倒数。如果$q(z)$是定理2.2中的“特征多项式”，便可得到$g(z)=z^t q(1/z)$，于是各个β_i恰好是特征多项式的根。

复根。上面的所有处理对复根都是有效的。我们通过下面的递归给以说明

$$a_n=2a_{n-1}-a_{n-2}+2a_{n-3}\qquad(n>2,a_0=1,a_1=0,a_2=-1)$$

由此可得

$$g(z)=1-2z+z^2-2z^3=(1+z^2)(1-2z)$$

于是

$$f(z)=(1-z^4)(1-2z)\qquad(\bmod z^4)=(1-2z)$$

因此有

$$a(z)=\frac{f(z)}{g(z)}=\frac{1}{1+z^2}=\frac{1}{2}\left(\frac{1}{1-iz}+\frac{1}{1+iz}\right)$$

97

最后可得$a_n=\frac{1}{2}(i^n+(-i)^n)$。由此容易看到，当$n$是奇数时，$a_n$是0。当$n$是4的倍数时，$a_n$是1。当$n$是偶数但不是4的倍数时，$a_n$是$-1$（这个结果容易从$a(z)=1/(1+z^2)$直接导出）。对于初始条件$a_0=1$, $a_1=2$, $a_2=3$，可得$f(z)=1$，于是解按2^n的速率增长，而周期变化的项是由复根引起的。

多重根。当涉及多重根时，我们用表3-1第2、3行给出的展式完成相应的推导。例如，给定递归

$$a_n=5a_{n-1}-8a_{n-2}+4a_{n-3}\qquad(n>2,\ a_0=0,\ a_1=1,\ a_2=4)$$

由此可得

$$g(z)=1-5z+8z^2-4z^3=(1-z)(1-2z)^2$$

于是

$$f(z)=(z+4z^2)(1-5z+8z^2-4z^3)\qquad(\bmod z^3)=z(1-z)$$

因此有$a(z)=z/(1-2z)^2$，从而由表3-1可得$a_n=n2^{n-1}$。

由这些例子可归纳出精确求解线性递归的一般的直接方法:

- 由递归导出$g(z)$。
- 由$g(z)$和初始条件计算$f(z)$。
- 削去$f(z)/g(z)$中的公共因子。
- 利用部分分式将$f(z)/g(z)$表示为形如$(1-\beta z)^{-j}$的项的线性组合。
- 将每个部分分式按以下公式展开

$$[z^n](1-\beta z)^{-j}=\binom{n+j-1}{j-1}\beta^n$$

实际上，这个处理过程可以给出定理2.2的构造性的证明。

习题3.17 解以下递归

$$a_n = 5a_{n-1} - 8a_{n-2} + 4a_{n-3} \qquad (n > 2,\ a_0 = 1,\ a_1 = 2,\ a_2 = 4)$$

习题3.18 解以下递归

$$a_n = 2a_{n-2} - a_{n-4} \qquad (n > 4,\ a_0 = a_1 = 0,\ a_2 = a_3 = 1)$$

习题3.19 解以下递归

98

$$a_n = 6a_{n-1} - 12a_{n-2} + 18a_{n-3} - 27a_{n-4} \qquad (n > 4,\ a_0 = 0,\ a_1 = a_2 = a_3 = 1)$$

习题3.20 解以下递归

$$a_n = 3a_{n-1} - 3a_{n-2} + a_{n-3} \qquad (n > 2,\ a_0 = a_1 = 0,\ a_2 = 1)$$

然后再将初始条件a_1改为$a_1 = 1$，求解同一个递归。

习题3.21 解以下递归

$$a_n = \sum_{1 \le k \le t} \binom{t}{k} (-1)^{k-1} a_{n-k} \qquad (n \geqslant t,\ a_0 = \cdots = a_{t-2} = 0,\ a_{t-1} = 1)$$

用OGF求解Quicksort算法的递归。当递归的系数是下标n的多项式时，则约束生成函数的隐含关系是一个微分方程。例如，我们回忆第1章介绍的描述快速排序算法所使用的比较次数的基本递归

$$NC_N = N(N+1) + 2\sum_{1 \le k \le N} C_{k-1} \qquad (N \geqslant 1,\ C_0 = 0) \tag{3-1}$$

定义生成函数

$$C(z) = \sum_{N \geqslant 0} C_N z^N \tag{3-2}$$

利用前面介绍的方法可以建立$C(z)$必须满足的函数方程。先用z^N乘以式（3-1）的两边，然后关于N求和，得到

$$\sum_{N \geqslant 1} NC_N z^N = \sum_{N \geqslant 1} N(N+1) z^N + 2\sum_{N \geqslant 1} \sum_{1 \le k \le N} C_{k-1} z^N$$

现在，我们可以直接求出每一项。上式的左边是$zC'(z)$（对式（3-2）两边求导，再乘以z），右边的第一项是$2z/(1-z)^3$（见表3-1），剩下的一项是个二重求和，就是部分和卷积（见表3-2），结果为$zC(z)/(1-z)$，因此递归对应于一个关于生成函数的微分方程

99

$$C'(z) = \frac{2}{(1-z)^3} + 2\frac{C(z)}{1-z} \tag{3-3}$$

为了得到这个方程的解，先解对应的齐次方程$\rho'(z) = 2\rho(z)/(1-z)$，得到一个“积分因子”$\rho(z) = 1/(1-z)^2$，于是有

$$\begin{aligned}((1-z)^2 C(z))' &= (1-z)^2 C'(z) - 2(1-z)C(z) \\ &= (1-z)^2\left(C'(z) - 2\frac{C(z)}{1-z}\right) = \frac{2}{1-z}\end{aligned}$$

积分后就得到了所需要的结果

$$C(z) = \frac{2}{(1-z)^2} \ln\frac{1}{1-z} \tag{3-4}$$

定理3.4 (Quicksort的OGF) 一个随机排列的Quicksort算法所用平均比较次数为

$$C_N = [z^N]\frac{2}{(1-z)^2}\ln\frac{1}{1-z} = 2(N+1)(H_{N+1}-1)$$

证明 前面的讨论已经给出了生成函数的显式表达式，按照3.2节开始部分给出的用OGF求解递归的一般过程，这已经完成了第3步。为了进一步得到所需要的系数，只须将表示调和数的生成函数求导即可。 ■

利用OGF求解递归的方法，尽管相当一般化，但肯定不能依靠这种方法给出所有递归的解，在第2章最后给出的各种例子可以证明这一点。对于某些问题，可能导不出一个具有简单形式的和；而对另一些问题，要想导出生成函数的显式公式是相当困难的；还有一些问题，将表达式展开成幂级数会成为主要的障碍。但是，确实有许多递归，初看起来难于处理，但事实上是可以用生成函数求解的。

习题3.22 利用生成函数解以下递归

$$na_n = (n-2)a_{n-1} + 2 \qquad (n \geqslant 2, a_1 = 1)$$

习题3.23 [Greene与Knuth]解以下递归

$$(n+1)a_{n+1} = (n+t)a_n \qquad (n \geqslant 0, a_0 = 1)$$

习题3.24 解以下递归

$$a_n = n+1+\frac{t}{n}\sum_{1\leqslant k\leqslant n} a_{k-1} \qquad (n \geqslant 1,\ a_0 = 0)$$

其中，$t = 2-\varepsilon$与$t = 2+\varepsilon$，ε是一个小的正常数。 100

3.4 生成函数的展开

给定某生成函数的显式函数形式，我们希望导出求得相应序列的一般的方法。这个过程称为生成函数的“展开”，也就是将一个紧凑的函数形式转换为一个无穷级数。前面已经看到，利用表3-1到3-4所给出的基本恒等式和变换，通过代数运算可以处理许多函数。那么，表3-1和表3-3中的最基本的展开式又是怎样得到的呢？

当给定0点的各阶导数时，利用泰勒定理可以给出函数$f(z)$的展开式

$$f(z) = f(0) + f'(0)z + \frac{f''(0)}{2!}z^2 + \frac{f'''(0)}{3!}z^3 + \frac{f''''(0)}{4!}z^4 + \cdots$$

因此从原则上讲，通过计算各阶导数，可以求得给定的生成函数所对应的序列。

指数序列。由于e^z的各阶导数仍然是e^z，泰勒定理的最简单的应用就是建立下面的基本展开式

$$e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!} + \cdots$$

几何序列。由表3-1可知，关于序列$\{1, c, c^2, c^3, \cdots\}$的生成函数是$(1-cz)^{-1}$。由于$(1-cz)^{-1}$的$k$阶导数是$k!c^k(1-cz)^{-k-1}$，当$z=0$时就是$k!c^k$，于是泰勒定理验证了由表3-1给出的函数的下述展开式是正确的：

$$\frac{1}{1-cz} = \sum_{k\geqslant 0} c^k z^k$$

二项式定理。函数 $(1+z)^x$ 的 k 阶导数是

$$x(x-1)(x-2)\cdots(x-k+1)(1+z)^{x-k}$$

于是由泰勒定理，可得二项式定理的广义版本，即著名的牛顿公式

101

$$(1+z)^x=\sum_{k\geq 0}\binom{x}{k}z^k$$

其中二项式系数定义为

$$\binom{x}{k}\equiv x(x-1)(x-2)\cdots(x-k+1)/k!$$

这里一个重要的特例是

$$\frac{1}{\sqrt{1-4z}}=\sum_{k\geq 0}\binom{2k}{k}z^k$$

它可由以下恒等式导出

$$\begin{aligned}\binom{-1/2}{k}&=\frac{-\frac{1}{2}\left(-\frac{1}{2}-1\right)\left(-\frac{1}{2}-2\right)\cdots\left(-\frac{1}{2}-k+1\right)}{k!}\\&=\frac{(-1)^k}{2^k}\frac{1\cdot3\cdot5\cdots(2k-1)}{k!}\frac{2\cdot4\cdot6\cdots2K}{2^k k!}\\&=\frac{(-1)^k}{4^k}\binom{2k}{k}\end{aligned}$$

与此紧密相关的一个展式在算法分析中发挥核心的作用，我们将在3.8节和第5章中看到。

习题3.25 用泰勒定理写出下列函数展开式：

$$\sin(z),\qquad 2^z,\qquad ze^z$$

习题3.26 利用泰勒定理验证，表达式 $(1-az-bz^2)^{-1}$ 所对应的级数的展开式系数满足具有常系数的二阶线性递归。

习题3.27 利用泰勒定理直接验证

$$H(z)=\frac{1}{1-z}\ln\frac{1}{1-z}$$

是调和数列的生成函数。

习题3.28 求

$$[z^n]\frac{1}{\sqrt{1-z}}\ln\frac{1}{1-z}$$

102 的表达式（提示：展开 $(1-z)^{-\alpha}$，然后关于 α 求导）。

习题3.29 求

$$[z^n]\left(\frac{1}{1-z}\right)^t\ln\frac{1}{1-z}\quad（所有的整数 t>0）$$

的表达式。

原则上，通过直接应用泰勒定理，总是可以计算生成函数的系数的，但具体计算过程可

能过于复杂而无法实现。在大多数情况下，是把要展开的生成函数分解成若干个较简单的部分，而每个小部分的展开式是熟知的。前面的一些例子正是这样做的，其中包括使用卷积来展开二项系数和调和数的生成函数，以及使用分解成部分分式的方法来展开斐波那契数的生成函数。的确，这些都是精选的方法，在本书中，我们将广泛地使用这些方法。对于特定的一类问题，还可以选用其他工具帮助进行处理，例如，在本章的后面将详细地讨论的拉格朗日反演定理就是其中的一种。

此外，对于某些问题，试图通过分解得到相应的展开式似乎是做不到的。对这些问题，与其寻求展开生成函数的一般性工具，不如寻求一种能够直接得到关于系数的更简明的表示的手段，即一种可以直接导出关于系数的渐近估计的工具，它使我们忽略一些不相干的细节。尽管这种一般的方法涉及复分析，其内容超出了本书的范围，但我们所使用的求解线性递归的部分分式展开的方法正是出自同样的直觉。例如，由对应于斐波那契数列的部分分式展式，可直接导出当$z = 1/\phi$或$z = 1/\hat{\phi}$时，生成函数$F(z)$不收敛。但实际上这种“奇异性”又完全决定了系数F_N的渐近增长。对于这种情况，我们可以直接利用展开式验证系数按照ϕ^N的速率增长（在相差一个常数因子的前提下）。于是就有可能给出系数按这种方式增长的一般性的条件，并给出一般性的技巧，以确定其他的增长率。通过分析生成函数的奇异性，我们常常可以导出关于所讨论的量的更精确估计，而不是不得不借助于更详尽的展开式。该课题的讨论可参看4.10节，详细论述参看文献[3]。

然而依然存在着大量的序列，它们所对应的生成函数是已知的，对这些生成函数做简单的代数运算，就可以得到我们所关心的那些量的简单的表达式。对那些表示经典组合序列的基本生成函数，在本章还要做进一步的讨论。从第5章到第8章，将对组合算法分析中出现的那些熟知的函数给出全面的分析，必要时我们将讨论对这些函数的处理细节，我们拥有确定这些系数的强有力的工具。 103

3.5 利用生成函数进行变换

生成函数可以简明地表示一个无穷数列，其重要性依据这样的事实：对包含生成函数的等式做简单的运算，往往可以得到关于一些重要序列的令人惊讶的关系式，而这些关系式用其他方式是很难导出的。下面看几个基本的例子：

范德蒙卷积。这是关于二项式系数的恒等式（见第2章），

$$\sum_k \binom{r}{k}\binom{s}{N-k} = \binom{r+s}{N}$$

上式的推导十分简单，因为它就是下述的函数关系式中相应的系数的卷积

$$(1+z)^r(1+z)^s = (1+z)^{r+s}$$

利用更复杂的卷积，可以导出大量类似的恒等式。

Quicksort递归。用$(1-z)$乘以OGF，对应于对系数的差分，表3-2已经指明了这一点。第1章中，关于Quicksort的递归，就是这样处理的（当时并没有明确地标明）。为了得到相应的解，还要做其他的变换。这里我们要指出，对生成函数的表示方式所进行各种操作，比对序列形式的操作要容易得多。对此，本章的后面还要做详细的讨论。

斐波那契数。关于斐波那契数的生成函数可表示为

$$F(z) = \frac{z}{1-y} \qquad (y = z + z^2)$$ 104

关于y展开这个式子可得

$$F(z)=z\sum_{N\geq 0}y^N=z\sum_{N\geq 0}(z+z^2)^N$$
$$=\sum_{N\geq 0}\sum_k\binom{N}{k}z^{N+k+1}$$

但F_N就是z^N的系数，于是必然有

$$F_N=\sum_k\binom{N-k-1}{k}$$

这就是斐波那契数与Pascal三角形的对角线之间熟知的关系。

二项式变换。如果$a^n=(1-b)^n$对所有的n成立，则显然有$b^n=(1-a)^n$，令人惊讶的是，这一点可推广到任意序列：给定两个序列$\{a_n\}$和$\{b_n\}$，满足以下的等式

$$a_n=\sum_k\binom{n}{k}(-1)^k b_k$$

由表3-4可知，相应的生成函数满足$e^zB(-z)=A(z)$。当然，此时也有$A(-z)=e^zB(z)$，这意味着

$$b_n=\sum_k\binom{n}{k}(-1)^k a_k$$

在随后的各章，将看到更多类似处理的例子。

习题3.30 证明：

$$\sum_k\binom{2k}{k}\binom{2N-2k}{N-k}=4^N$$

习题3.31 用$(1-z)^2$乘以Quicksort的生成函数所满足的微分方程(3-3)的两边，所对应的关于序列$\{C_N\}$的递归是什么？

习题3.32 假定某OGF满足微分方程：

$$A'(z)=-A(z)+\frac{A(z)}{1-z}$$

它所对应的递归是什么？用$(1-z)$乘以上式的两边，然后令系数相等，可导出不同的递归，然后解这个递归。试将这种求解方式与直接求得OGF然后通过展开求解的方式进行比较。

105

习题3.33 通过下述卷积可导出关于二项式系数的什么样的恒等式？

$$(1+z)^r(1-z)^s=(1-z^2)^s(1+z)^{r-s}$$

其中$r>s$。

习题3.34 证明：

$$\sum_{0\leq k\leq t}\binom{t-k}{r}\binom{k}{s}=\binom{t+1}{r+s+1}$$

习题3.35 利用生成函数求和$\sum_{0\leq k\leq N}F_k$。

习题3.36 利用生成函数求关于$[z^n]\dfrac{z}{1-e^z}$的和式。

习题3.37 利用生成函数求关于 $[z^n]\frac{1}{2-e^z}$ 的和式。

习题3.38 [Dobinski，参看Comtet]证明：

$$n![z^n]e^{e^z-1}=e^{-1}\sum_{k\geq 0}\frac{k^n}{n!}$$

习题3.39 利用OGF证明二项变换恒等式。设$A(z)$与$B(z)$满足以下关系式：

$$B(z)=\frac{1}{1-z}A\left(\frac{z}{z-1}\right)$$

然后使用变量代换：$z=y/(y-1)$。

习题3.40 不使用生成函数，直接证明二项变换恒等式。

习题3.41 [Faà di Bruno's公式，参看Comtet] 设

$$f(z)=\sum_n f_n z^n \text{ 以及 } g(z)=\sum_n g_n z^n$$

用多项式定理导出$[z^n]f(g(z))$的表达式。

3.6 关于生成函数的函数方程

在算法分析中，算法的递归（或算法分析中的递推关系）往往要得到关于所对应生成函数的函数方程。我们已经看到了一些例子，在这些例子中，可以求出函数方程的显式解，然后展开以得到所需要的系数。而对于另外的一些情况，我们可以利用函数方程确定解的渐进性态而不用求出生成函数的显式形式。或者将问题进行变换，得到一个更容易求解的类似的形式。本节，我们将对不同类型的函数方程通过习题和例题，提供一些说明。 106

线性。关于斐波那契数的生成函数是一个典型的例子

$$f(z)=zf(z)+z^2f(z)+z$$

该线性方程将导出一个关于生成函数的显式公式，或许可以进一步展开。但这里的线性指的仅仅是出现在线性组合中的函数本身，而系数和随后导出的公式可以是任意复杂的。

非线性。对于更一般的情形，生成函数可能等于关于它本身的任意的一个函数，而不一定是线性函数。这方面的一个著名的例子是关于Catalan数的生成函数，它由函数方程定义如下

$$f(z)=zf(z)^2+1$$

以及关于树的生成函数，它满足以下的函数方程

$$f(z)=ze^{f(z)}$$

前者即将在后面讨论某些细节，而对后者的讨论放在第5章。依据非线性函数的性质，可以用代数方法导出生成函数的显式公式。在本章的后面，将讨论一个一般性的工具，叫做拉格朗日反演定理，利用它可在许多类似情况下导出所需要的系数。

微分。函数方程可能包含生成函数的导数，例如前面已见过的Quicksort的例子

$$f'(z)=\frac{2}{(1-z)^3}+2\frac{f(z)}{1-z}$$

下面将看到更详细的例子。当然，求解生成函数显式公式的能力与求解微分方程的能力直接相关。

复合函数。还有一些情况，函数方程可能含有生成函数的变量的线性或非线性的函数。例如下面一些来自算法分析的例子

107

$$f(z)=e^{z/2}f(z/2)$$
$$f(z)=z+f(z^2+z^3)$$

第一个式子与二叉树和基数排序相关（参看第7章），而第二个式子来自2-3树（参看第5章）。显然，如果任意编制一个复杂的方程，则不能确保解容易得到。求解这一类方程的一般工具，可参看文献[3]。

这些例子告诉我们，可以预期，在进行算法分析时可能会遇到生成函数的使用。本书我们将考查这些例子和关于生成函数的其他函数方程。通常，这些方程是一个分界线，它标志着详细的算法研究的终止和分析工具详细应用的开始。无论求解函数方程多么困难，重要的是要记住我们能够利用这些函数方程得到基础序列的性质。

与递归类似，迭代（iteration）技巧，即简单地将方程逐次代入方程本身，对于确定由函数方程决定的生成函数的性质往往是很有用的。例如，考虑满足以下方程的EGF;

$$f(z)=e^z f(z/2)$$

给定$f(0)=1$，就必然有:

$$\begin{aligned}f(z)&=e^z e^{z/2}f(z/4)\\&=e^z e^{z/2}e^{z/4}f(z/8)\\&\vdots\\&=e^{z+z/2+z/4+z/8+\cdots}\\&=e^{2z}\end{aligned}$$

这就证明2^n是以下递推关系的解:

$$f_n=\sum_k\binom{n}{k}\frac{f_k}{2^k}\qquad(n>0,\ f_0=1)$$

108 从技术上讲，迭代需要进行无穷多次，但由原始递归容易给出解的验证。

习题3.42 证明下面的展开式中的系数f_n

$$e^{z+z^2/2}=\sum_{n\geq 0}f_n\frac{z^n}{n!}$$

满足二阶线性递归$f_n=f_{n-1}+(n-1)f_{n-2}$（提示：求函数$f(z)=e^{z+z^2/2}$所满足的微分方程）。

习题3.43 解函数方程:

$$f(z)=e^{-z}f\left(\frac{z}{2}\right)+e^{2z}-1$$

假定$f(z)$是某个序列的EGF，导出相应的序列和解。

习题3.44 求OGF的显式公式，该OGF所表示的序列满足分治递归

$$f_{2n}=f_{2n-1}+f_n\qquad(n>1,\ f_0=0)$$
$$f_{2n+1}=f_{2n}\qquad(n>0,\ f_1=1)$$

习题3.45 迭代以下函数方程，以求得$f(z)$的显式公式，

$$f(z)=1+zf\left(\frac{z}{1+z}\right)$$

习题3.46 [Polya]给定由下述函数方程定义的$f(z)$:

$$f(z)=\frac{z}{1-f(z^2)}$$

求$a(z)$和$b(z)$的显式表达式，其中$f(z)=a(z)/b(z)$。

习题3.47 证明：仅存在一个形如$f(z)=\sum_{n>1}f_nz^n$的幂级数，满足$f(z)=\sin(f(z))$。

习题3.48 利用关于2-3树的函数方程导出一个基础递归，再用这个递归确定共有多少个具有100个结点的2-3树。

3.7 利用OGF求解三数中值Quicksort递归

作为处理关于生成函数的函数方程的一个详尽的例子，我们回顾一下在1.5 节给出的一个递推关系，该关系描述了三数中值Quicksort的平均比较次数，如果不使用生成函数，这个递推关系是很难处理的

$$C_N=N+1+\sum_{1<k<N}\frac{(N-k)(k-1)}{\binom{N}{3}}(C_{k-1}+C_{N-k})\qquad(N>2)$$

109

其中，$C_0=C_1=C_2=0$。为便于分析，我们用$N+1$表示为划分N个元素所需要的比较次数，实际的开销依赖于如何计算中位数和具体实现的其他性质，但可以限定在$N+1$的附近的一个小的可加常数的范围内。另外，初始条件$C_2=0$（由它可导出$C_3=4$）的选取也是为了便于分析，尽管具体实现一般是不同的值。如同在1.5节所做的，我们可以通过取这个递归的解和其他类似的递归的解的线性组合来解释其细节，诸如划分阶段数的计数这样的递归（指用1替代$N+1$得到的同样的递归）。

下面是按照利用生成函数求解递归的标准步骤处理的。用$N(N-1)(N-2)$乘以两边，然后消去和式中的对称部分，可得

$$N(N-1)(N-2)C_N=(N+1)N(N-1)(N-2)+12\sum_{1<k<N}(N-k)(k-1)C_{k-1}$$

然后，用z^{N-3}乘以两边，再对N求和，最后得到微分方程

$$C'''(z)=\frac{24}{(1-z)^5}+12\frac{C'(z)}{(1-z)^2}\tag{3-5}$$

对于高阶微分方程，不能总期望可以得到显式的解。但这个例子还是属于一个可以显式求解的类型。首先，用$(1-z)^3$乘以两边，得到

$$(1-z)^3C'''(z)=12(1-z)C'(z)+\frac{24}{(1-z)^2}\tag{3-6}$$

在这个方程中，每个系数中$(1-z)$的次数等于该项求导的阶数，这是一种在常微分方程理论中熟知的可解类型，叫做*欧拉方程*。用一个算子乘以两边，然后微分，可以将该式写成可以分解的形式。定义算子如下:

$$\psi C(z)=(1-z)\frac{\mathrm{d}}{\mathrm{d}z}C(z)$$

于是可以将式（3-6）重写为:

110

$$\psi(\psi+1)(\psi+2)C(z)=12\psi C(z)+\frac{24}{(1-z)^2}$$

将以上包含ψ的各式合并成一个多项式，再分解因子，可得:

$$\psi(\psi+5)(\psi-2)C(z)=\frac{24}{(1-z)^2}$$

这就意味着，求解该方程可转换为依次求解三个关于$C(z)$的一阶微分方程。

$$\psi U(z)=\frac{24}{(1-z)^2} \qquad 或 \quad U'(z)=\frac{24}{(1-z)^3}$$

$$(\psi+5)T(z)=U(z) \qquad 或 \quad T'(z)=-5\frac{T(z)}{1-z}+\frac{U(z)}{1-z}$$

$$(\psi-2)C(z)=T(z) \qquad 或 \quad C'(z)=2\frac{C(z)}{1-z}+\frac{T(z)}{1-z}$$

解这些一阶微分方程比求解原来的方程要简单得多，立即可以得到解。

定理3.5（三数中值Quicksort） 对于一个随机排列的三数中值Quicksort算法所使用的平均比较次数为:

$$C_N=\frac{12}{7}(N+1)\left(H_{N+1}-\frac{23}{14}\right) \qquad (N\geqslant 6)$$

证明 延续上面的讨论，解微分方程的结果为:

$$U(z)=\frac{12}{(1-z)^2}-12$$

$$T(z)=\frac{12}{7}\frac{1}{(1-z)^2}-\frac{12}{5}+\frac{24}{35}(1-z)^5$$

$$C(z)=\frac{12}{7}\frac{1}{(1-z)^2}\ln\frac{1}{1-z}-\frac{54}{49}\frac{1}{(1-z)^2}+\frac{6}{5}-\frac{24}{245}(1-z)^5$$

展开关于$C(z)$的表达式（忽略最后一项），就可得到所要的结果（见3.1节中的习题）。这个OGF的首项与标准的Quicksort的OGF相比，只差一个常数因子。 ■

111

我们可以将这个分解转换到$U(z)$而将$T(z)$转换到对应的序列的递归。考虑生成函数$U(z)=\sum U_N z^N$以及$T(z)=\sum T_N z^N$，这时关于生成函数所做的处理对应于关于递归所做的处理，但对照直接对递归求解，前者所使用的工具具有更广泛的可用性，而且多少也会更容易发现和使用。不仅如此，利用生成函数求解的方式还可以在求解大样本的问题中使用，进一步的细节可以在文献[8]或[12]中找到。

除了作为生成函数使用的实际例子之外，这个颇为详细的例子还表明，对于我们所关心的一些重要的性能特征多么精确的数学表述能够用来帮助我们选择算法的控制参数合适的值（此时是样本的大小）。例如上面的分析指出，利用三数中值Quicksort算法，我们可以节省大约14%的用于比较的开销。更细致的分析还要考虑一些额外的开销（主要的是额外的交换，因为分割元更接近中间）。分析表明，更大的样本将会导致进一步的改进。

习题3.49 证明$(1-z)^t C^{(t)}(z)=\psi(\psi+1)\cdots(\psi+t+1)C(z)$。

习题3.50 求在三数中值的Quicksort算法中所使用的交换的平均次数。

习题3.51 求在Quicksort算法中，在平均的意义下所使用的比较和交换的次数。算法修改为在分划时，使用5个元素的中位数。

习题3.52 [Euler] 讨论下述微分方程的解：

$$\sum_{0\le j\le r}(1-z)^{r-j}\frac{\mathrm{d}^j}{\mathrm{d}z^j}f(z)=0$$

并讨论非齐次情形的解，其中右边的项形如$(1-z)^{\alpha}$。

习题3.53 [van Emden,参看Knuth]证明：当具有$(2t+1)$个元素的样本的中位数用于分划时，Quicksort所使用的比较次数为

$$\frac{1}{H_{2t+2}-H_{t+1}}N\ln N+O(N)$$

3.8 利用生成函数的计数

上面的讨论集中于如何将生成函数作为一种分析工具，用于求解递推关系。这只是其功能的一部分，生成函数还用于系统地求解组合对象的计数问题。“组合对象”可以是正在被算法操作的数据结构，因此这种计数处理在算法分析中也具有同样重要的作用。

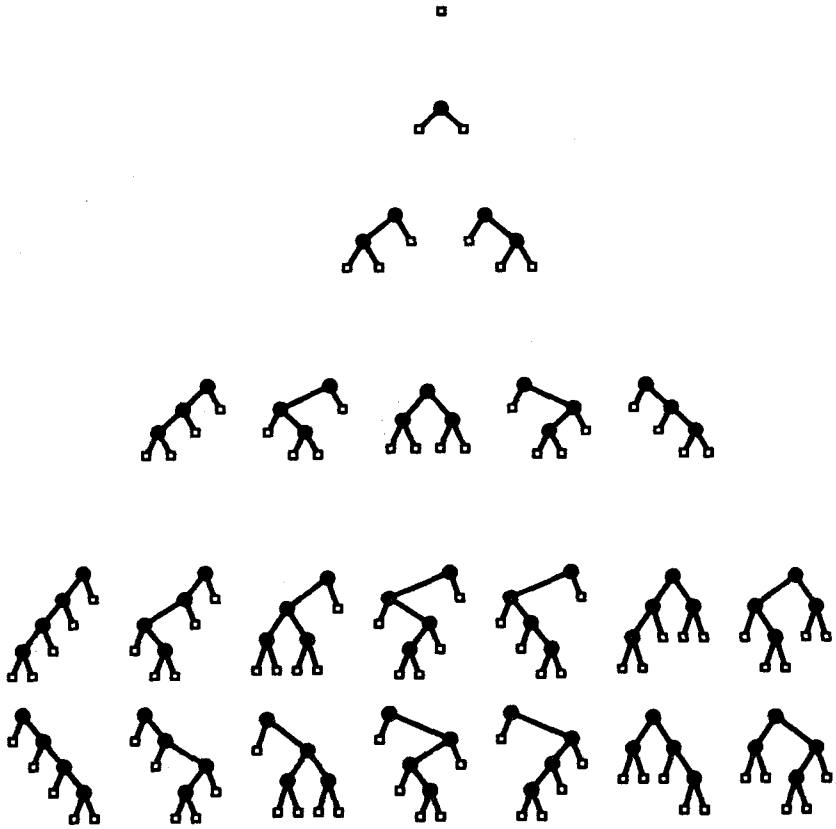


图3-1 含1、2、3、4和5的外部结点的所有二叉树

第一个例子是一个经典的组合学问题，该问题所对应的基本数据结构在第5章和本书其他

许多地方都会讨论到。二叉树（binary tree）是递归定义的一种结构，它或者是一个单个的外部结点（external node），或者是一个连接两棵二叉树的内部结点（internal node），这两棵二叉树分别称为左子树和右子树。图3-1给出了至多为5个结点的全部二叉树，二叉树出现在组合学和算法分析的许多问题中。例如，如果内部结点对应于两个参数的算术运算，外部结点对应变量，那么二叉树就对应算术表达式。现在要处理的问题是：有多少棵具有N个外部结点的二叉树？

112 ~ 113

二叉树的计数。处理方式之一是定义一个递归。设T_N是具有$N+1$个外部结点的二叉树的个数，由图3-1可知$T_0=1, T_1=1, T_2=2, T_3=5, T_4=14$。利用递归定义可以导出一个递推关系：一个具有$N+1$个外部结点的二叉树，如果左子树具有$k$个外部结点（这样的左子树有$T_{k-1}$个），则右子树一定有$N-k+1$个外部结点（存在$T_{N-k}$种可能）。于是，$T_N$必满足：

$$T_N=\sum_{1\le k\le N}T_{k-1}T_{N-k}\qquad (N>0,\ T_0=1)$$

这是一个简单的卷积。两边乘以z^N，再关于N求和，我们发现对应的OGF一定满足非线性函数方程

$$T(z)=zT(z)^2+1$$

利用二次方程，容易求出$T(z)$的表达式：

$$zT(z)=\frac{1}{2}\left(1\pm\sqrt{1-4z}\right)$$

为了保证$z=0$时两边相等，右边应取减号。

定理3.6 (二叉树的OGF) 具有$N+1$个外部结点的二叉树的棵数由Catalan数给出：

$$T_N=[z^{N+1}]\frac{1-\sqrt{1-4z}}{2}=\frac{1}{N+1}\binom{2N}{N}$$

证明 OGF的显式表达式已在上面给出。为了得到系数，对指数$\frac{1}{2}$用二项式定理展开（牛顿公式），得：

$$zT(z)=-\frac{1}{2}\sum_{N\ge 1}\binom{\frac{1}{2}}{N}(-4z)^N$$

114

令两边系数相等，便得到

$$\begin{aligned}T_N&=-\frac{1}{2}\binom{\frac{1}{2}}{N+1}(-4)^{N+1}\\&=-\frac{1}{2}\frac{\frac{1}{2}\left(\frac{1}{2}-1\right)\left(\frac{1}{2}-2\right)\cdots\left(\frac{1}{2}-N\right)(-4)^{N+1}}{(N+1)!}\\&=\frac{1\cdot3\cdot5\cdots(2N-1)\cdot2^N}{(N+1)!}\\&=\frac{1}{N+1}\frac{1\cdot3\cdot5\cdots(2N-1)}{N!}\frac{2\cdot4\cdot6\cdots2N}{1\cdot2\cdot3\cdots N}\\&=\frac{1}{N+1}\binom{2N}{N}\end{aligned}$$

■

我们将在第5章看到，二叉树的外部结点的个数恰好比内部结点的个数多1个。于是，Catalan数T_N也给出了具有N个内部结点的二叉树的棵数。在下一章我们将看到，其近似值是$T_N \sim 4^N / N\sqrt{\pi N}$。

二叉树的（直接）计数。有一种更简单的方式可用于确定上述生成函数的显式表达式，这种方式使我们进一步看到生成函数在计数问题中实质性的应用。定义$\mathcal{T}$为全部二叉树的集合，记号$|t|$表示二叉树t的内部结点的个数，其中$t \in \mathcal{T}$。于是，可以得到

$$\begin{aligned} T(z) &= \sum_{t \in \mathcal{T}} z^{|t|} \\ &= 1 + \sum_{t_L \in \mathcal{T}} \sum_{t_R \in \mathcal{T}} z^{|t_L|+|t_R|+1} \\ &= 1 + zT(z)^2 \end{aligned}$$

上式第一行是由定义得到的$T(z)$的另一种表达式。一棵恰好具有k个外部结点的树对z^k的系数的贡献恰好是1。于是和中的z^k的系数就是对具有k个内部结点的树的计数。第二行是由二叉树的递归定义得到的：一棵二叉树或者没有内部结点（对应于1），或者它可以分解为两棵独立的二叉树，这两棵子树的内部结点构成了原树的内部结点，再加上一个根结点。第三行的导出是由于下标变量t_L、t_R是各自独立的。建议读者仔细地研究这个基本的例子，在本书中将看到很多其他类似的例子。 115

习题3.54 修正上面的推导，直接导出具有N个外部结点的二叉树的棵数的生成函数。

兑换美元（波利亚(Polya)）。一个经典的利用生成函数计数的例子是波利亚提出的，问题描述如下："用1分、5分、1角、25分、50分的硬币兑换1美元，共有多少种兑换方法？"如果使用二叉树的直接计数的方法，其生成函数可表为

$$D(z) = \sum_{p,n,d,q,f \geq 0} z^{p+5n+10d+25q+50f}$$

其中求和所使用的下标p, n, d等等，分别表示所使用的1分、5分、1角等硬币的个数。每个总和为k分的一个选定的硬币的组合对z^k的系数的贡献恰好是1，于是这就是所需要的生成函数。由于在$D(z)$的表达式中，每个求和下标都是独立的，因此可得

$$\begin{aligned} D(z) &= \sum_p z^p \sum_n z^{5n} \sum_d z^{10d} \sum_q z^{25q} \sum_f z^{50f} \\ &= \frac{1}{(1-z)(1-z^5)(1-z^{10})(1-z^{25})(1-z^{50})} \end{aligned}$$

通过建立对应的递归，或利用计算机代数系统，可以求得$[z^{100}]D(z) = 292$。

习题3.55 讨论$[z^N]D(z)$的表达式的形式。

习题3.56 写出用于计算$[z^N]D(z)$的有效计算机程序，其中N由现场输入。

习题3.57 证明将N表示成2的幂的线性组合（具有整数系数）的方法总数的生成函数是

$$\prod_{k \geq 1} \frac{1}{1-z^{2^k}}$$

习题3.58 [Euler]证明：

$$\frac{1}{1-z} = (1+z)(1+z^2)(1+z^4)(1+z^8)\cdots$$

给出前t个因子的乘积的封闭形式，该式有时也被称为"计算机科学恒等式"，为什么？ 116

习题3.59 将上面的习题推广到以3为底的情形。

习题3.60 以N的二进制表示表出$[z^N](1-z)(1-z^2)(1-z^4)(1-z^8)\cdots$。

二项分布。恰好有k位为1的长度为N的二进制序列（显然有$N-k$位为0）共有多少个？设$\mathcal{B}_N$表示全部长度为N的二进制序列的集合，$\mathcal{B}_{Nk}$表示全部长度为N且恰好有k位为1的二进制序列的集合，相应的生成函数应满足

$$B_N(z)=\sum_k|\mathcal{B}_{Nk}|z^k$$

但注意到在$\mathcal{B}_N$中，每个恰好有k位为1的二进制字符串b对z^k的系数的贡献恰好是1。重写生成函数，使它按每个串“计数”：

$$B_N(z)=\sum_{b\in\mathcal{B}_N}z^{\{b\text{中1的位数}\}}\quad=\sum_{b\in\mathcal{B}_{Nk}}z^k\left(=\sum_k|B_{Nk}|z^k\right)$$

现在，我们把所有具有k个1的N位二进制串的集合表示为所有具有k个1的$N-1$位二进制串的集合（在每个串的前面加一位0）与所有具有$k-1$个1的$N-1$位二进制串的集合（在每个串的前面加一位1）的并。于是

$$\begin{aligned}B_N(z)&=\sum_{b\in\mathcal{B}_{(N-1)k}}z^k+\sum_{b\in\mathcal{B}_{(N-1)(k-1)}}z^k\\&=B_{N-1}(z)+zB_{N-1}(z)\end{aligned}$$

于是$B_N(z)=(1+z)^N$。利用二项式定理展开这个函数，即得到所需要的答案$|\mathcal{B}_{Nk}|=\binom{N}{k}$。

下面给出用生成函数求解计数问题的一般方法：

- 首先，对于要计数的组合学对象，写出带有关于下标求和的生成函数的一般的表达式。
- 然后，按照对应关系，将求和式分解为若干个处理对象的结构，以导出生成函数的显式公式。
- 最后，将生成函数表示为幂级数的形式，以求得系数的表达式。

117

在前面介绍二叉树计数问题的生成函数时，我们曾介绍了一种不同的处理方法：使用要处理的对象的结构，导出一个递归，然后利用生成函数解这个递归。对于简单的问题，使用这种还是那种方法，讲不出太多的理由，但对于较复杂的问题，直接方法可以避免有时候是和递归一起产生的单调冗长的计算，在本书的后面将看到许多这方面的例子。更重要的一点是，直接方法可以导致具有较强功能的一般化的处理。在下一节，我们将介绍一个计算框架，在这个框架内过程的第一步几乎是自动实现的，最后一步，即从生成函数的显式表达式中求出系数，则与求解递归是相同的。

3.9 符号方法

二叉树的函数方程是相当简单的，那么是否可以使用更直接的推导方法呢？答案是肯定的。树的递归定义与OGF的二次方程之间的相似性是十分明显的。本节我们将指出，这种相似性并不是一种巧合，而是本质的。我们可以将表达式

$$T(z)=1+zT(z)^2$$

解释为：“一棵二叉树或者是一棵空树或者由一个结点和两棵二叉树组成”。

这样处理有两个明显的特色：首先它是符号的，可以只使用少量的代数法则对符号信息

进行操作。有些作者强调，这事实上是在使用符号图求解，而不是对所得到的公式中诸如z之类的变量求解。其次它直接反映一种定义结构的方式，我们生成这些结构，而不是把它们分解开来进行分析。

无标号对象。像在前一节那样，当使用生成函数计数时，我们是在考虑那些组合对象的类，对每个对象定义一个记号“大小”。对于类$\mathcal{A}$，我们用a_n来表示类$\mathcal{A}$中大小为n的元素的个数。此时我们关注的是OGF

$$A(z)=\sum_{n\geq 0}a_n z^n=\sum_{\alpha\in\mathcal{A}}z^{|\alpha|}$$

给定两个组合对象的类$\mathcal{A}$和$\mathcal{B}$，可以用“不相交并”运算$\mathcal{A}+\mathcal{B}$将它们结合在一起，得到的类由$\mathcal{A}$、$\mathcal{B}$中不重复的元素组成。而运算“笛卡儿乘积”$\mathcal{A}\times\mathcal{B}$则给出一组有序对的对象的类，其元素一个来自$\mathcal{A}$，一个来自$\mathcal{B}$。下面的定理在关于组合对象的类的运算与相应的生成函数之间，给出一个简单的对应关系。这一结果可以使我们直接利用对象的构造性定义导出生成函数之间的函数关系。 118

定理3.7 (OGF符号方法) 设$\mathcal{A}$和$\mathcal{B}$是两个组合对象的类，如果$A(z)$是用于计算$\mathcal{A}$的OGF，而$B(z)$是用于计算$\mathcal{B}$的OGF，那么，

$A(z)+B(z)$ 是用于计算$\mathcal{A}+\mathcal{B}$的OGF，

$A(z)B(z)$ 是用于计算$\mathcal{A}\times\mathcal{B}$的OGF，

$\dfrac{1}{1-A(z)}$是用于计算$\mathcal{A}$中的对象序列的OGF。

证明 要证明的第一部分是显然成立的：如果a_n是$\mathcal{A}$中大小为n的对象的个数，而b_n是$\mathcal{B}$中大小为n的对象的个数，那么a_n+b_n就是$\mathcal{A}+\mathcal{B}$中大小为n的对象的个数。

为证明第二部分，注意到对于从0到n中的任何一个k，都可以在$\mathcal{A}$中找到大小为k的a_k个对象，再与从$\mathcal{B}$中找到的大小为$n-k$的b_{n-k}个对象组合在一起，构成$\mathcal{A}\times\mathcal{B}$中大小为$n$的对象，于是，$\mathcal{A}\times\mathcal{B}$中大小为$n$的对象共有

$$\sum_{0\leq k\leq n}a_k b_{n-k}$$

个，这个简单的卷积就蕴涵着所需要的结果。另外，

$$\sum_{\gamma\in\mathcal{A}\times\mathcal{B}}z^{|\gamma|}=\sum_{\alpha\in\mathcal{A}}\sum_{\beta\in\mathcal{B}}z^{|\alpha|+|\beta|}=A(z)B(z)$$

关于第三部分序列的结果可以由以下表达式导出：

$$\varepsilon+\mathcal{A}+\mathcal{A}^2+\mathcal{A}^3+\mathcal{A}^4+\cdots$$

这是关于$\mathcal{A}$中的对象序列的类的表达式。其中ε表示空序列。利用定理的前两部分，关于这个类计数的生成函数是：

$$1+A(z)+A(z)^2+A(z)^3+A(z)^4+\cdots=\frac{1}{1-A(z)}$$

■ 119

习题3.61 给出定理3.7的第2部分另一种证明，照前面关于二叉树的“直接”推导，利用生成函数的组合的形式给出这个证明。

下面看一个简单的例子。设$\mathcal{A}$是由大小为0的空对象ε组成的类，$\mathcal{B}$是由大小为1的两个对象$\{0,1\}$组成的类，则$A(z)=1$，$B(z)=2z$，于是组合类$\{\varepsilon,0,1\}$的OGF是$1+2z$。这里要注意，

空对象ε的OGF是1，它与空类ϕ完全不同，后者的OGF是0。设$\mathcal{B}$是所有二进制串的集合，一个二进制串或者为空，或者严格对应于由一个0或一个1后跟一个二进制串组成的有序对。符号表示为

$$\mathcal{B} = \varepsilon + \{0, 1\} \times \mathcal{B}$$

利用定理3.7，这个符号形式可直接转换为生成函数所满足的函数方程，即

$$B(z) = 1 + 2zB(z)$$

于是，$B(z) = 1/(1 - 2z)$，$B_N = 2^N$，这正是所期望的。此外，我们还可以把$\mathcal{B}$看作是由二进制字符表{0, 1}上的序列形成，其OGF就是$2z$，于是由定理3.7的序列法则，又得到了$B(z) = 1/(1 - 2z)$。

更有趣的类似的例子，考虑没有两个0相连的二进制串，这样的串或者是ε，或者是单个的0，或者1或01后跟一个没有两个0相连的二进制串。符号表示为

$$\mathcal{G} = \varepsilon + \{0\} + \{1, 01\} \times \mathcal{G}$$

由定理3.7，可以立即转换为计算这种数字串的生成函数的$G(z)$的分式:

$$G(z) = 1 + z + (z + z^2)G(z)$$

于是可得$G(z) = (1 + z)/(1 - z - z^2)$。由此直接导出没有两个0相连的长度为$N$的二进制串的个数为$F_N + F_{N+1} = F_{N+2}$，恰好是斐波那契数。

还有另外一种考查二进制串的方式：用1的位数作为大小的参数，于是$1 + z$可作为类{0, 1}的生成函数，$(1 + z)^2$可作为类{00, 01, 10, 11}的生成函数等等，而$(1 + z)^N$可作为表示1在长度为N的二进制串中的总数的生成函数。

120

再回到二叉树，其符号公式为

$$\mathcal{T} = \{\square\} + \{\bullet\} \times \mathcal{T} \times \mathcal{T}$$

为了计算内部结点的个数，将{□}转换为1，将{•}转换为z，可得

$$T(z) = 1 + zT(z)^2$$

我们在前面已经导出，这个函数方程定义了Catalan数。在第5章将看到，在研究各种类型的树的性质时，符号处理方式的简单性是令人信服的，特别是在与用递归进行分析的方法比较之后。

定理3.7也叫做“变换定理”，因为它给出了从定义结构的符号公式到对结构计数的生成函数的一个直接的变换。除了在定理3.7中介绍的并、笛卡儿积和序列等操作外，还有许多其他用于建立组合结构的操作，这些例子包括关于它们的集合或多重集等等。在第5章将看到其中大部分例子，相关内容全部含于文献[3]。

习题3.62 设$\mathcal{B}$定义为$\mathcal{A}$的所有有限个子集的集合，如果$A(z)$、$B(z)$分别是$\mathcal{A}$和$\mathcal{B}$的OGF，证明

$$B(z) = \prod_{n \geq 1} (1 + z^n)^{A_n} = \exp\left(A(z) - \frac{1}{2}A(z^2) + \frac{1}{3}A(z^3) - \cdots\right)$$

习题3.63 设$\mathcal{B}$定义为$\mathcal{A}$的所有有限个多重集的集合（允许重复的子集），如果$A(z)$、$B(z)$分别是$\mathcal{A}$和$\mathcal{B}$的OGF，证明

$$B(z)=\prod_{n\geq 1}\frac{1}{(1-z^n)^{A_n}}=\exp\left(A(z)+\frac{1}{2}A(z^2)+\frac{1}{3}A(z^3)+\cdots\right)$$

带有标号的对象。前面讨论的一个主要特点是对于构成组合对象的那些单个的项，是没有区别的。还有一种模式，是假定这些单项并不相同，需要区分。每个个体都是有标号的，因此把组合对象组装在一起时，每一个单项出现的顺序是有意义的。带有标号的对象一般是利用指数生成函数计数的。下面先用一个简单的例子说明基本的原理，然后在第5章和第6章再详细研究带有标号的对象的相关问题。 121

为具体起见，假定某个组合结构是由N个最基本的项（有时也称为“原子”）组成。将每个最基本的项用整数1到N标记，如果两个对象在结构中标号次序不相同，就要看作是两个不同的对象。

上面我们在讨论组装无标号的结构时，介绍了诸如“和”、“笛卡儿积”之类的操作，在此我们将定义一些类似的适用于有标号对象的结构。其主要差别是：当两个对象联合在一起时，必须重新标号，使得在大小为N的结果对象中，只能出现标号1到N。特别在做组合类的乘积运算时，操作会复杂一些。但实际上可以利用相应的“变换定理”，它与定理3.7非常类似。

圈与排列。最简单的标号结构是排列。我们在分析排序算法时已提到了这一点。一个排列就是一组有标号的项的一次有顺序的摆放。例如对于2 1 3，表示这样的顺序安排：标号为2的项排在第一位，标号为1的项排在第二位，标号为3的项排在第三位。N个具有标号的项共有$N!$种不同的排列顺序，于是，关于排列的EGF是

$$\sum_{p\in\mathcal{P}}\frac{z^{|p|}}{|p|!}=\sum_{N\geq 0}N!\frac{z^N}{N!}=\frac{1}{1-z}$$

作为一个更有启发性的例子，下面介绍一种由有标记项的“圈集”构成的组合类。这个例子似乎比上面的位串例子抽象一些，但实际上只不过是从另一个角度考查排列问题而已。在第6章将看到，圈和排列与大量的基本算法的分析直接相关。这里所使用的分析方法，可用于解决那些用其他方法难以解决的问题。

一个标号圈是若干有标号的项按照循环顺序的排列。例如，(1 2 3) 表示一个循环顺序，其顺序规定为标号为2的项紧跟在标号为1的项之后，标号为3的项紧跟在标号为2的项之后，而标号为1的项应跟在标号为3的项之后。于是，(1 2 3) 所表示的排列对象与(2 3 1) 相同，但与(2 1 3)不同。 122

习题3.64 证明：n个项的所有不同标号圈的总数是$(n-1)!$。

圈的集合可以简单地看作这样的对象的集合：在这些对象中间，顺序是不重要的。我们希望知道，具有N项的圈集合共有多少个？（其他有关的性质可参看第6章）。表3-5列举了$N=2,3,4$时全部的圈集合。

与无标号对象（例如树）一样，我们需要一些方式将标号对象类联合在一起。在这里“不相交并”运算本质上是相同的，但“乘积”运算要包括重新标号的过程。下面借助一个例子说明这一点。假设我们要从两个圈 (1) 和 (1 3 2) 出发，构造一个2-圈对象，最简单的结构是一种顺序排列：用具有1项的圈，后跟一个具有3项的圈，这样就出现了4项，于是标号就应该从1到4。有4种不同的方式可以作出保持一致性的处理，即可以保持每个圈成分的原有的标号顺序：

(1)(2 4 3)　(2)(1 4 3)　(3)(1 4 2}　(4)(1 3 2)。

其中单个元素组成的圈可以使用4个可能标号中的任何一个进行标号，而相应的3-圈只有一种标记方法用剩余标号进行标记。

表3-5　全部的有标号圈的集合（n=2,3,4）

(1)　(2)	(1)(2)(3)	(1)(2)(3)　(4)	(1)(2)　(3)(4)
(1　2)	(1)　(2 3)	(1)　(2 3 4)	(1)(3)　(2 4)
	(2)　(1 3)	(2)　(1 3 4)	(1)(4)　(2 3)
	(3)　(1 2)	(3)　(1 2 4)	(2)(3)　(1 4)
	(1 2 3)	(4)　(1 2 3)	(2)(4)　(1 3)
	(1 3 2)	(1)　(2 4 3)	(3)(4)　(1 2)
		(2)　(1 4 3)	(1 2 3 4)
		(3)　(1 4 2)	(1 2 4 3)
		(4)　(1 3 2)	(1 3 2 4)
		(1 2)　(3 4)	(1 3 4 2)
		(1 3)　(2 4)	(1 4 2 3)
		(1 4)　(2 3)	(1 4 3 2)

123

类似地，从两个相同的圈(1　2)出发，有6种不同的方式可以做出关于两个有标号圈的有序排列:

(1 2)(3 4)　(1 3)(2 4)　(1 4)(2 3)
(2 3)(1 4)　(2 4)(1 3)　(3 4)(1 2)

将这种处理推广到两个标号对象的类$\mathcal{A}$和$\mathcal{B}$的乘积运算$\mathcal{A}\star\mathcal{B}$，得到对象序偶的类，其中一项来自$\mathcal{A}$，而另一项来自$\mathcal{B}$，以所有一致的顺序重新标记。

下面，我们讨论通过构造对象集合来建立结构的对象。在上面的例子中，当把两个2-圈联合成一个2-圈有序排列时，在计数时每个标号圈集合出现了两次。这是因为，当把它们联合成集合时，圈的次序是不重要的。从两个相同的圈(1 2)出发，只有3种不同的方式可以构造两个标号圈的集合:

(1 2)(3 4)　(1 3)(2 4)　(1 4)(2 3)

一般地在一个有序排列中，如果有k个恒等的分量，在我们用所有可能的方式重新标记它们时，每个分量的集合要出现$k!$次，因为我们“忘记”了各分量之间的顺序。等价地，集合的个数可以通过有序排列的总数除以$k!$得到。

标号对象的符号方法。上面的讨论显然可以又得到一个变换定理:

定理3.8 (EGF的符号方法)　设$\mathcal{A}$和$\mathcal{B}$是两个标号的组合对象的类，如果$A(z)$是用于对$\mathcal{A}$计数的EGF，而$B(z)$是用于对$\mathcal{B}$计数的EGF，那么，

$A(z)+B(z)$ 是用于对$\mathcal{A}+\mathcal{B}$计数的EGF，

$A(z)B(z)$ 是用于对$\mathcal{A}\star\mathcal{B}$计数的EGF，

$\dfrac{1}{1-A(z)}$是用于对$\mathcal{A}$中的对象序列计数的EGF，

$e^{A(z)}$是用于对$\mathcal{A}$中的对象的集合计数的EGF。

证明　第一部分的证明与定理3.7中的证明相同。

为证明第二部分，注意到对于从0到n中的每一个k，都可以在$\mathcal{A}$中找到大小为k的a_k个对象，再与从$\mathcal{B}$中找到的大小为$n-k$的b_{n-k}个对象配对，构成$\mathcal{A}\star\mathcal{B}$中大小为$n$的对象。重新标号可以有$\binom{n}{k}$种不同的方式（直接选定标号，次序就被确定了）。于是，在$\mathcal{A}\star\mathcal{B}$中标号为$n$的对象共有 124

$$\sum_{0\leq k\leq n}\binom{n}{k}a_k b_{n-k}$$

个，仍然是一个简单的卷积。

关于第三部分序列的结果也与定理3.7相同。而关于集合的结果，可直接由下面的符号公式导出:

$$\varepsilon+\mathcal{A}+\mathcal{A}^2/2!+\mathcal{A}^3/3!+\mathcal{A}^4/4!+\cdots$$

其中$\mathcal{A}^2=\mathcal{A}\star\mathcal{A}$，$\mathcal{A}^3=\mathcal{A}\star\mathcal{A}\star\mathcal{A}$，等等。 ■

现在回到原先的例子。由于大小为n的有标号圈的总数是$(n-1)!$（见习题3.63），则相应的EGF为

$$\sum_{n\geq 1}\frac{z^n}{n}=\ln\frac{1}{1-z}$$

此时定理3.8表明，用于对有标号的圈集合计数的EGF是

$$\exp\left(\ln\frac{1}{1-z}\right)=\frac{1}{1-z}$$

这就是说，n项的圈的具有标号的集合的总数恰好是$n!$，即排列总数，这是一个基本的结果。在第6章，我们将看到，存在一些更容易的方法证明上述结论，但这里的证明结构，可以使我们更容易得到一些更加困难的问题的求解方法。

上面简短的介绍只给出了符号方法的一个表面轮廓，这一方法是现代组合分析的核心内容之一。更详细的内容可参看Goulden与Jackson[5],或Stanley[14]。在[3]中，我们给出了在算法分析中所涉及的关于这一方法的较为深入透彻的处理（还可以参看[16]）。该方法在理论上是相当完备的，并且已经建立了相应的计算机程序，这些程序可以从一个简单的递归定义出发，自动地确定具有某种结构的生成函数，详情可参看Flajolet、Salvy与Zimmerman[2]。在本书中，我们介绍了基本结构及其性质，当可能应用符号方法时，给出了通向[3]的一些线索。 125

由定理3.7和定理3.8所概括的符号方法，适用于结构不断扩充的集合。虽然它不能很自然地解决所有问题：某些组合对象由于有过多的内部交叉结构而难于使用这种处理方法，但对于具有很好的分解形式的组合结构，这是一种值得提倡的方法，例如各种树结构（第5章），一种典型的例子；排列（第6章）；串（第7章）；以及字或映射（第8章）等等。当确实可以使用这种方法时，往往会获得惊人的成功，特别地，它可以很快地分析基本结构的各种变体。对于第5章到第8章的许多基本结构，我们将给出不止一种推导方法，如同我们对二叉树和位串所做的工作。为了更好地促进基本工具的使用，我们尽力维持文字上的连贯性，同时要包含多种情形，包括那些适合使用符号方法但还没做出有效工作的地方。在后面我们将看到大量的例子，这些例子帮助我们得到对生成函数之间一些简单关系的基本组合学起因的正确评价。

3.10 拉格朗日反演

我们常常面临着从生成函数中提取系数的任务，而生成函数的定义往往隐含在函数方程之中，符号方法使得这项工作变得非常简单。下面的一般性工具可用来解决这个问题，对于与树的计数问题相关联的问题，这些工具具有特殊的重要性。

定理3.9 (拉格朗日反演定理) 如果生成函数$A(z)=\sum_{n\geq 0}a_nz^n$满足函数方程$z=f(A(z))$，其中$f(z)$满足$f(0)=0$，且$f'(0)\neq 0$，则

$$a_n\equiv[z^n]A(z)=\frac{1}{n}[u^{n-1}]\left(\frac{u}{f(u)}\right)^n$$

以及

$$[z^n](A(z))^m=\frac{m}{n}[u^{n-m}]\left(\frac{u}{f(u)}\right)^n$$

126 并且

$$[z^n]g(A(z))=\frac{1}{n}[u^{n-1}]g'(u)\left(\frac{u}{f(u)}\right)^n$$

证明 略。可参看[1]。其中包括大量的关于这一公式的文献，日期可以追溯到18世纪。■

函数求逆。函数f的逆函数是函数f^{-1}，它满足$f^{-1}(f(z))=f(f^{-1}(z))=z$，将$f^{-1}$作用到方程$z=f(A(z))$的两边，可以看到函数$A(z)$正好是函数$f(z)$的逆函数。在这个意义上，拉格朗日反演定理对于幂级数的转换，是一个广泛应用的工具。一个突出的特点是它在函数的逆函数的系数和函数的幂之间，提供了一个直接的对应关系。在本书的相关章节中，对于提取隐函数中的系数，它是一个很有用的工具。通过下面给出的大量例子，我们强调一种形式上的操作，但只给出一些应用框架（各种树的计数问题）。关于应用的更多的严格的表述是第5章到第8章的主要研究内容。

二叉树。设$T^{[2]}(z)=zT(z)$是对应于二叉树的关于外部结点计数的OGF，将函数方程

$$T^{[2]}(z)=z+T^{[2]}(z)^2$$

改写为

$$z=T^{[2]}(z)-T^{[2]}(z)^2$$

应用拉格朗日反演定理，取$f(u)=u-u^2$,则有

$$[z^n]T^{[2]}(z)=\frac{1}{n}[u^{n-1}]\left(\frac{u}{u-u^2}\right)^n=\frac{1}{n}[u^{n-1}]\left(\frac{1}{1-u}\right)^n$$

然后，利用表3-1，可得

$$\frac{u^{n-1}}{(1-u)^n}=\sum_{k\geq n-1}\binom{k}{n-1}u^k$$

于是，考虑到项$k=2n-2$，有

$$[u^{n-1}]\left(\frac{1}{1-u}\right)^n=\binom{2n-2}{n-1}\qquad 因此\qquad [z^n]T^{[2]}(z)=\frac{1}{n}\binom{2n-2}{n-1}$$

这正是所需要的。

三叉树。推广二叉树的一种途径是让树的每个结点具有多于两个的子结点。例如，三叉
127 树是这样一种树，其每个结点或者是外部结点，或者有三棵子树（左子树、中子树、右子树）。

对于$n=1,2,3,4,5,6,7,\cdots$，具有n个外部结点的三叉树的计数序列是1，0，1，0，3，0，3，…。利用符号方法，可以立即导出函数方程:

$$z=T^{[3]}(z)-T^{[3]}(z)^3$$

利用3.8节关于二叉树所做的处理，并不容易成功，因为现在是三次方程，而不是二次方程。但是应用拉格朗日反演定理，取$f(u)=u-u^3$，则可立即得到结果

$$[z^n]T^{[3]}(z)=\frac{1}{n}[u^{n-1}]\left(\frac{1}{1-u^2}\right)^n$$

然后利用与前面同样的处理，由表3-1可得

$$\frac{u^{2n-2}}{(1-u^2)^n}=\sum_{k\geq n-1}\binom{k}{n-1}u^{2k}$$

于是，考虑到项$2k=3n-3$（只有当n是奇数时，该项存在），有

$$[z^n]T^{[3]}(z)=\frac{1}{n}\binom{(3n-3)/2}{n-1}$$

对所有的奇数n成立。当n是偶数时，其值为0。

二叉树的森林。推广二叉树的另一种途径是考虑二叉树的集合，即所谓的森林。二叉树的k-森林是k棵二叉树的一个有序的序列。由定理3.7，关于k-森林的OGF正好是$(zT(z))^k$，其中$T(z)$是关于二叉树的OGF。由拉格朗日反演定理（使用定理3.9中第2种情况），具有n个外部结点的二叉树的k-森林的总数为

$$[z^n]\left(\frac{1-\sqrt{1-4z}}{2}\right)^k=\frac{k}{n}\binom{2n-k-1}{n-1}$$

这些数也叫做选票数（ballot numbers），在第7章将给出明确的解释。 128

标号树。考虑函数方程

$$C(z)=ze^{C(z)}$$

这是熟知的一般的（标号无序）树的指数型生成函数所满足的函数方程（见第5章）。应用拉格朗日反演定理，取$f(u)=ue^{-u}$，则可立即得到

$$[z^n]C(z)=\frac{1}{n}[u^{n-1}]e^{un}=\frac{1}{n}\frac{n^{n-1}}{(n-1)!}=\frac{n^{n-1}}{n!}$$

函数$C(z)$叫做Cayley函数。

习题3.65 求$[z^n]A(z)$，其中$A(z)$由以下函数方程定义

$$z=\frac{A(z)}{1-A(z)}$$

习题3.66 求$[z^n]e^{\alpha C(z)}$，其中$C(z)$是Cayley函数。

习题3.67 （阿贝尔二项式定理）利用前面习题的结果和恒等式$e^{(\alpha+\beta)C(z)}=e^{\alpha C(z)}e^{\beta C(z)}$证明

$$(\alpha+\beta)(n+\alpha+\beta)^{n-1}=\alpha\beta\sum_k\binom{n}{k}(k+\alpha)^{k-1}(n-k+\beta)^{n-k-1}$$

习题3.68 e^z-1的逆函数是什么？通过应用拉格朗日反演定理，将得到以幂级数表示的

什么结果？

习题3.69 求四叉树的总数。四叉树是这样的树，其每个结点或者是外部结点，或者有四棵有序的子树。

习题3.70 求具有n个结点的由三叉树组成的不同的3-森林的总数。

3.11 概率生成函数

在算法分析中使用概率的处理方式可以简化对平均值和方差的计算，概率生成函数与这种应用直接相关。

定义 给定随机变量X，该变量只取非负整数值，且$p_k = Pr(X = k)$，函数$P(u) = \sum_{k\geq 0} p_k u^k$称为关于随机变量$X$的概率生成函数（PGF）。

我们假定读者基本熟悉在1.7节以及在算法的平均情况的分析中讨论过的计算随机变量的平均值和标准差。但在这里还是先复习一下有关的定义，因为在本节和下一节要做一些相关的计算。

129

定义 X的期望值，或$E(X)$（也叫做X的平均值），定义为$\sum_{k\geq 0} kp_k$。根据$r_k \equiv Pr(X \leq k)$，它等价于$E(X) = \sum_{k\geq 0}(1 - r_k)$。$X$的方差，或$var(X)$定义为$\sum_{k\geq 0}(k - E(X))^2 p_k$。$X$的标准差定义为$\sqrt{\mathrm{var}(X)}$。

概率生成函数是很重要的，因为它提供了一种计算平均值和方差的计算方式，而不需要进行求离散和之类的冗长计算。

定理3.10（利用PGF求平均值和方差） 给定关于随机变量X的PGF $P(z)$，则X的期望值由$P'(1)$给出，且方差由$P''(1) + P'(1) - P'(1)^2$给出。

证明 如果$p_k \equiv \Pr(X = k)$，则

$$P'(1) = \sum_{k\geq 0} kp_k u^{k-1}\big|_{u=1} = \sum_{k\geq 0} kp_k$$

由定义，这就是所要的期望值。类似地，注意到$P(1) = 1$，则所述关于方差的结果由定义可以直接导出：

$$\begin{aligned}\sum_{k\geq 0}(k - P'(1))^2 p_k &= \sum_{k\geq 0} k^2 p_k - 2\sum_{k\geq 0} kP'(1)p_k + \sum_{k\geq 0} P'(1)^2 p_k \\ &= \sum_{k\geq 0} k^2 p_k - P'(1)^2 \\ &= P''(1) + P'(1) - P'(1)^2\end{aligned}$$

■

量$E(X^r) = \sum_k k^r p_k$叫做X的r阶矩，期望值是一阶矩，方差是二阶矩与一阶矩平方之差。

在3.9节的两个定理通过符号方法实现计数的合成法则，可转换为关于独立随机变量的联合PGF的相应表述。例如，如果$P(u)$、$Q(u)$是关于独立的随机变量X和Y的概率生成函数，则$P(u)Q(u)$是关于随机变量$X + Y$的概率生成函数，而且两个概率生成函数的积所表示的分布的平均值和方差，分别是各自平均值和方差的和。

130

习题3.71 给出按照$r_k = \Pr(X \leq k)$所表示的var(X)的简单表达式。

习题3.72 定义$\mathrm{mean}(P) \equiv P'(1)$, $\mathrm{var}(P) \equiv P''(1) + P'(1) - P'(1)^2$，证明

$\mathrm{mean}(PQ) = \mathrm{mean}(P) + \mathrm{mean}(Q)$, $\mathrm{var}(PQ) = \mathrm{var}(P) + \mathrm{var}(Q)$,对任何满足$P(1) = Q(1)$的可微函数$P$和$Q$，而不仅仅是对PGF成立。

均匀离散分布。给定整数$n>0$,假定X_n是随机变量，等可能地取整数值0, 1, 2, …, $n-1$的每一个值。则关于X_n的概率生成函数是

$$P_n(u)=\frac{1}{n}+\frac{1}{n}u+\frac{1}{n}u^2+\cdots+\frac{1}{n}u^{n-1}$$

期望值是

$$P_n'(1)=\frac{1}{n}(1+2+\cdots+(n-1))=\frac{n-1}{2}$$

而且由于

$$P_n''(1)=\frac{1}{n}(1\cdot 2+2\cdot 3+\cdots+(n-2)(n-1))=\frac{1}{6}(n-2)(n-1)$$

因此方差是

$$P_n''(1)+P_n'(1)-P_n'(1)^2=\frac{n^2-1}{12}$$

习题3.73 从闭形式:

$$P_n(u)=\frac{1-u^n}{n(1-n)}$$

验证上述结果，用洛必达法则计算1处的导数。

习题3.74 求关于随机变量的PGF，其中随机变量用于计算随机二进制串中前导0的个数，并用该PGF计算平均值和标准差。

二项分布。考虑一个具有N个独立位的随机串,每一位为0的概率是p,为1的概率是$q=1-p$。可以证明N位中恰好有k位是0的概率是

$$\binom{N}{k}p^kq^{N-k}$$

131

于是，对应的PGF是

$$P_N(u)=\sum_{0\le k\le N}\binom{N}{k}p^kq^{N-k}u^k=(pu+q)^N$$

另外可以看出，对应于单个位的那些0的PGF是$(pu+q)$，而且N位是互相独立的。于是对应于N位中的0的个数的PGF是$(pu+q)^N$, 0的平均个数是$P'(1)=pN$, 方差是$P''(1)+P'(1)-P'(1)^2=pqN$等等。很容易做这些计算，而不需要显式地确定单个的概率。

不能期待总是那么幸运，不能指望经常会遇到可以完全分解成独立的刻划前面例子的特征的PGF的情况。在这种情况下，结构数2^N的计数容易分解成N个简单因子。由于在计算平均值时，这些量出现在分母中，因此分子的因子分解也就成为可能。反过来，如果不能按这种方式进行因子分解，例如对Catalan数的情形，就不能期望会很容易地找到独立的参数，这就是我们强调在算法分析中，使用累加的及二元生成函数，而不是PGF的主要原因（参看下一节）。

Quicksort分布。设$Q_N(u)$是Quicksort所用到的比较次数的PGF，应用关于PGF的合成法则可以证明$Q_N(u)$满足函数方程

$$Q_N(u)=\frac{1}{N}\sum_{1\le k\le N}u^{N+1}Q_{k-1}(u)Q_{N-k}(u)$$

尽管依靠这个方程求得$QN(u)$的显式表达式相当困难，但它确实提供了一个进行矩的计算的基础。例如微分并计算在$u=1$点的值，将会直接导出我们在3.3节给出的标准的Quicksort的递推关系。注意，这个PGF对应于一个用比较次数作为下标的序列；我们在3.3节求解式（3-1）时使用的OGF是用文件中元素的个数作为下标，在下一节将看到如何将两者统一处理成一个二重生成函数。

尽管概率生成函数对于进行算法的平均情形的分析似乎是很自然的工具，但一般地说在处理组合结构的参数分析时，我们不强调使用PGF，下一节将给出一些明显的理由。在处理离散结构时，两种处理方式在形式上是相关的，但不是等价的。而计数方法更自然一些，并且允许做更灵活的处理。

132

3.12 二元生成函数

在算法分析中，通常我们不仅对给定大小的结构的计数有兴趣，而且还对与结构相关的各种参数的值有兴趣，符号方法也拓展到这种情况。

为了达到这个目的，我们使用二元生成函数。它们是两个变量的函数，表示双下标的序列：一个下标表示问题的大小，一个下标表示正在分析的参数的值。二元生成函数使我们能够只用一个二变量的生成函数获得两个下标。

定义 给定一个双下标序列$\{a_{nk}\}$，函数

$$A(u,z)=\sum_{n\ge 0}\sum_{k\ge 0}a_{nk}u^kz^n$$

叫做该序列的二元生成函数（BGF）。我们用记号$[u^kz^n]A(u,z)$表示a_{nk}；用$[z^n]A(u,z)$表示$\sum_{k\ge 0}a_{nk}u^k$；而用$[u^k]A(u,z)$表示$\sum_{n\ge 0}a_{nk}z^n$。

有时候需要把BGF通过除以$n!$变成指数的形式。于是，$\{a_{nk}\}$的指数BGF为

$$A(u,z)=\sum_{n\ge 0}\sum_{k\ge 0}a_{nk}u^k\frac{z^n}{n!}$$

我们经常使用BGF对组合结构中的参数值如下计数。对$p\in\mathcal{P}$,其中$\mathcal{P}$是一类组合结构，令$\mathrm{cost}(p)$为给出对每个结构定义的某个参数值的函数。此时，我们的兴趣是在BGF

$$P(u,z)=\sum_{p\in\mathcal{P}}u^{\{\mathrm{cost}(p)\}}z^{|p|}=\sum_{n\ge 0}\sum_{k\ge 0}p_{nk}u^kz^n$$

上，其中p_{nk}为大小为n和值为k的结构的个数。为了把所有大小为n的结构的值分开，我们也写成

$$P(u,z)=\sum_{n\ge 0}p_n(u)z^n \quad \text{其中} \quad p_n(u)=[z^n]A(u,z)=\sum_{k\ge 0}p_{nk}u^k$$

133

而为把所有值为k的结构分开，则写成

$$P(u,z)=\sum_{k\ge 0}q_k(z)u^k \quad \text{其中} \quad q_k(z)=[u^k]A(u,z)=\sum_{n\ge 0}p_{nk}z^n$$

再有，注意

$$P(1,z)=\sum_{p\in\mathcal{P}}z^{|p|}=\sum_{n\ge 0}p_n(1)z^n=\sum_{k\ge 0}q_k(z)$$

为对$\mathcal{P}$计数的常规生成函数。

我们的主要兴趣在于，如果所有大小为n的结构都等可能地取用，那么$p_n(u)/p_n(1)$就是代表开销的随机变量的PGF。因此，已知$p_n(u)$和$p_n(1)$就使我们能够计算平均开销以及其他的矩，这在前面一节已经描述。BGF基于组合结构开销参数的计数和分析对这类计算提供了方便的架构。

二项分布。令$\mathcal{B}$为所有二进制串的集合，并把二进制串的“开销”函数当作是比特1的个数。在这种情况下，$\{a_{nk}\}$则是具有k个1的n-比特二进制串的个数，于是相关的BGF是

$$P(u,z)=\sum_{n\geq 0}\sum_{k\geq 0}\binom{n}{k}u^k z^n=\sum_{n\geq 0}(1+u)^n z^n=\frac{1}{1-(1+u)z}$$

如3.8节那样，考虑另一种推导是有益的，这种推导与符号方法更紧密相关，它阐述了用于全书许多类似推导的基本处理手法。我们从定义

$$P(u,z)\equiv\sum_{b\in\mathcal{B}}u^{\{\#\text{ of 1-bits in }b\}}z^{|b|}$$

开始。每一个二进制串b或者为空，或者由一个比特（或0或1）后跟一个二进制串b'组成。这导致下面的分解

$$\begin{aligned}P(u,z)&=1+\sum_{b'\in\mathcal{B}}u^{\{\#\text{ of 1bits in }b'\}}z^{1+|b'|}+\sum_{b'\in\mathcal{B}}u^{1+\{\#\text{ of 1 bits in }b'\}}z^{1+|b'|}\\&=1+zP(u,z)+uzP(u,z)\end{aligned}$$

134

关于$P(u,z)$求解则给出与上面相同的闭型表达式。通常，在生成函数间像这样的简单关系可以用符号方法解释：它相当于把一个二进制串定义为空或一个比特后跟一个二进制串：对于比特0相应BGF为z，而对应比特1则BGF为uz。这些结论与像前一节关于BGF的结论的那些转换定理联合将直接得到这个结果。正如在3.8节提到的，第三种办法是利用“序列”结构论证在比特序列中比特1的个数的BGF为

$$P(u,z)=\sum_{N\geq 0}(z+uz)^N=\frac{1}{1-(1+u)z}$$

将其展开，我们立刻看到

$$p_n(u)=(1+u)^n\quad\text{和}\quad q_k(z)=z^k/(1-z)^{k+1}$$

当然，

$$[u^k]p_n(u)=[z^n]q_k(z)=\binom{n}{k}$$

正如所料。

我们当然不需要为这些平凡的工作使用这里所讨论的一般方法计算二进制串中1的个数，但是使用像这样一个容易的问题作为运行的例子帮助读者检验定义和理解概念是有益的。在本节后面以及第5章到第8章的多处地方我们将使用这些方法来帮助求解许多更难的问题。

BGF展开。把大小为n的结构作为$[z^n]P(u,z)=p_n(u)$分离开常常叫做BGF的“水平”展开。这来自于全BGF展开作为一个二维表的自然表示，u的幂沿水平方向增加，而z的幂沿垂直方向增加。例如，二项分布的BGF可以写成如下：

$$
\begin{aligned}
&z^0(u^0)+\\
&z^1(u^0+u^1)+\\
&z^2(u^0+2u^1+u^2)+\\
&z^3(u^0+3u^1+3u^2+u^3)+\\
&z^4(u^0+4u^1+6u^2+4u^3+u^4)+\\
&z^5(u^0+5u^1+10u^2+10u^3+5u^4+u^5)+\cdots
\end{aligned}
$$

135

或者可以垂直地进行这样的表，我们整理$[u^k]P(u,z)=q_k(z)$。对于我们的二项分布则有

$$
\begin{aligned}
&u^0(z^0+z^1+z^2+z^3+z^4+z^5+\cdots)+\\
&u^1(z^1+2z^2+3z^3+4z^4+5z^5+\cdots)+\\
&u^2(z^2+3z^3+6z^4+10z^5+\cdots)+\\
&u^3(z^3+4z^4+10z^5+\cdots)+\\
&u^4(z^4+5z^5+\cdots)+\\
&u^5(z^5+\cdots)+\cdots
\end{aligned}
$$

此即所谓BGF的垂直展开。我们将看到，这些表示方法在算法分析中是重要的，特别是当全BGF的显式表达式不是现成可用的时候。

矩的“水平”计算。使用这些记法，概率及矩的计算就简单了。对u微分并求在$u=1$的值，我们发现

$$p_n'(1)=\sum_{k>0}kp_{nk}$$

$P(u,z)$在$u=1$处对u的偏导数是该量的生成函数。现在$p_n(1)$是大小为n的$\mathcal{P}$的成员数，如果我们认为大小为n的$\mathcal{P}$的所有成员都是等可能的，那么大小为n的结构具有开销k的概率是$p_{nk}/p_n(1)$并且大小为n的结构的平均开销是$p'_n(1)/p_n(1)$。

定义 令$\mathcal{P}$为具有BGF $P(u,z)$的组合结构的一个类，则函数

$$\left.\frac{\partial P(u,z)}{\partial u}\right|_{u=1}=\sum_{p\in\mathcal{P}}\mathrm{cost}(p)z^{|p|}$$

被定义为该类的累加生成函数（CGF）。再有，令$\mathcal{P}_n$表示$\mathcal{P}$中大小为n的所有这种结构的类，则和

$$\sum_{p\in\mathcal{P}_n}\mathrm{cost}(p)$$

136

被定义为大小为n的这种结构的累加开销。

由于累加开销恰好是CGF中z^n的系数，因此这个术语是合理的。有时候累加开销也叫做非规范化平均值（unnormalized mean），因为真正的平均值是通过“规范化”得到的，或者是用大小为n的结构的数目去除而得到的。

定理3.11 (BGF和平均开销) 给定一类组合结构的BGF $P(u,z)$，则给定大小的所有结构的平均开销由被结构数目所除的累加开销给出，或

$$\frac{[z^n]\left.\dfrac{\partial P(u,z)}{\partial u}\right|_{u=1}}{[z^n]P(1,z)}$$

证明 计算是简单的，直接从$p_n(u)/p_n(1)$是相关PGF的结论推导，然后应用定理3.10即可。■

BGF的使用和定理3.11的重要性在于，平均开销可以通过从

$$\left.\frac{\partial P(u,z)}{\partial u}\right|_{u=1} \qquad \text{和} \qquad p(1,z)$$

独立地抽取系数并进行除法而算出。更紧凑的记法是常常把偏导数写成$P_u(1, z)$，标准差可以用类似的方法算出。这些记法和计算在表3-6中给出了总结。

对于上面给出的涉及二项分布的例子，长为n的二进制串的个数为

$$[z^n]\left.\frac{1}{1-(1+u)z}\right|_{u=1} = [z^n]\frac{1}{(1-2z)} = 2^n$$

且累加开销（所有n比特二进制串中比特1的个数）是

$$[z^n]\frac{\partial}{\partial u}\left.\frac{1}{1-(1+u)z}\right|_{u=1} = [z^n]\frac{z}{(1-2z)^2} = n2^{n-1}$$

因此比特1的平均个数是$n/2$。或者从$p_n(u) = (1 + u)^n$开始，结构的数目是$p_n(1) = 2^n$且累加开销为$p'_n(1) = n2^{n-1}$。或者通过直接论证来计算平均值：长为n的二进制串的个数是2^n，在长为n的所有二进制串中比特1的个数是$n2^{n-1}$，因为共有总数$n2^n$个比特，它们的一半是比特1。 137

表3-6 从二元生成函数计算矩

大小为n的结构的开销生成函数	$[z^n]P(u,z) = p_n(u)$
计数开销为k的结构的生成函数	$[u^k]P(u,z) = q_k(z)$
累加生成函数	$\left.\frac{\partial P(u,z)}{\partial u}\right\|_{u=1} = q(z) = \sum_{k>0} kq_k(z)$
大小为n的结构的个数	$[z^n]P(1,z) = p_n(1)$
累加开销	$[z^n]\left.\frac{\partial P(u,z)}{\partial u}\right\|_{u=1} = \sum_{k>0} kp_{nk} = p_n'(1) = [z^n]q(z)$
平均开销	$\frac{[z^n]\left.\frac{\partial P(u,z)}{\partial u}\right\|_{u=1}}{[z^n]P(1,z)} = \frac{p_n'(1)}{p_n(1)} = \frac{[z^n]q(z)}{p_n(1)}$
方差	$\frac{p_n''(1)}{p_n(1)} + \frac{P_n'(1)}{p_n(1)} - \left(\frac{p_n'(1)}{p_n(1)}\right)^2$

$$P(u,z) = \sum_{p\in\mathcal{P}} u^{\{\mathrm{cost}(p)\}} z^{|p|} = \sum_{n>0}\sum_{k>0} p_{nk}u^k z^n = \sum_{n>0} p_n(u)z^n = \sum_{k>0} q_k(z)u^k$$ 138

习题3.75 如上所示，利用表3-6和$p_n(u) = (1 + u)^n$，计算长为n的随机二进制串中比特1的个数的方差。

矩的“垂直”计算。另外，累加开销还可以用垂直展开来计算：

$$[z^n]\sum_{k>0} kq_k(z) = \sum_{k>0} kp_{nk}$$

推论 累加开销还等于

$$[z^n]\sum_{k\geq 0}(P(1,z)-r_k(z)) \quad 其中 \quad r_k(z)=\sum_{0\leq j\leq k}q_j(z)$$

证明 函数$r_k(z)$是开销不大于k的所有结构的生成函数。由于$r_k(z)-r_{k-1}(z)=q_k(z)$，因此累加开销为

$$[z^n]\sum_{k\geq 0}k(r_k(z)-r_{k-1}(z))$$

将其叠缩则得到所述结果。 ■

随着k在该和中的增加，那些初始的项抵消（所有小结构的开销都不大于k），因此该表达式本身就导致渐近逼近。到第5章我们再返回到这个课题上来，到那时我们首先要遇到一些适合垂直公式的问题。

习题3.76 从垂直展开通过首先计算上面描述的$r_k(z)$来检验二项分布的中值是$n/2$。

二叉树中的树叶。现在看一个更有趣的使用BGF的例子，我们来讨论确定大小为n的二叉树中其两个子结点均为外部结点的内部结点的平均个数问题。这样的外部结点叫做树叶。从图3-1观察（见3.8节）我们看到，对$n=0$、1、2、3和4，这种结点的总数是0、1、2、6和20。用Catalan数去除，则相关的平均值为0、1、1、6/5和10/7。用BGF

139

$$T(u,z)=\sum_{t\in\mathcal{T}}u^{\{\text{leaves}(t)\}}z^{|t|}$$

表示，图3-1中的树对应下面的项

$$\begin{aligned}
&(u^0)z^0+\\
&(u^1)z^1+\\
&(u^1+u^1)z^2+\\
&(u^1+u^1+u^2+u^1+u^1)z^3+\\
&(u^1+u^1+u^2+u^1+u^2+u^2+u^2+u^1+u^1+u^2+u^1+u^1+u^2+u^1)z^4
\end{aligned}$$

这种情况的二维情形强调BGF对于分析的功效。我们用一个变量z记录一维（大小），而另一个变量u记录另外一维（开销）。将这些项加起来得到

$$T(u,z)=1+uz^1+2uz^2+(4u+u^2)z^3+(8u+6u^2)z^4+\cdots$$

检查小的值发现

$$T(1,z)=1+z^1+2z^2+5z^3+14z^4+\cdots$$

和

$$T_u(1,z)=z^1+2z^2+6z^3+20z^4+\cdots$$

正如所料。函数方程

$$T(u,z)=1+uz+zT(u,z)^2-z$$

从符号方法得出（将变量z减去是为了避免大小为1的树被计算两次）。严格地说，我们需要一个变换定理来证明写这个方程是正确的（对于这样的定理可见[3]），但是，我们还可以验证方程或者像上面为推导Catalan数所做的那样使用生成函数通过直接论证而得到它。当然，置$u=1$则得到一个熟悉的函数方程，因为$T(1,z)$是Catalan数的OGF。现在对u微分并求在$u=1$的值，

得到

$$\begin{aligned} T_u(1,z) &= z + 2zT(1,z)T_u(1,z) \\ &= \frac{z}{1-2zT(1,z)} \\ &= \frac{z}{\sqrt{1-4z}} \end{aligned}$$

140

因此我们证明了，在大小为n的二叉树中两个结点都是外部结点的内部结点的平均个数为

$$\frac{[z^n]\dfrac{z}{\sqrt{1-4z}}}{\dfrac{1}{n+1}\dbinom{2n}{n}} = \frac{\dbinom{2n-2}{n-1}}{\dfrac{1}{n+1}\dbinom{2n}{n}} = \frac{(n+1)n}{2(2n-1)}$$

（见3.4节和3.8节），它趋向于$n/4$。二叉树中大约1/4的内部结点是树叶。

习题3.77 求大小为n的二叉树中其两个子结点都是内部结点的内部结点的平均个数。

习题3.78 求大小为n的二叉树中一个子结点是内部结点而另一个子结点是外部结点的内部结点的平均个数。

习题3.79 求$T(u,z)$的显式公式并计算二叉树中树叶数目的方差。

Quicksort分布。在1.5节和3.3节，我们已经比较详细地研究了Quicksort运行时间的平均情形分析，因此从BGF的角度考查所进行的分析是有益的，包括方差的计算。我们通过考虑指数BGF

$$Q(u,z) = \sum_{N\geq 0}\sum_{k\geq 0} q_{Nk}u^k\frac{z^N}{N!}$$

开始，其中q_{Nk}是由Quicksort对N个元素的所有排列所使用的比较次数的累积计数。现在因为N个元素中有$N!$种排列，所以这实际上是“概率”BGF: $[z^N]Q(u,z)$不是别的，而是在上节末尾引入的PGF $Q_N(u)$。我们将在第6章的几个例子中看到，只要我们研究排列的性质，这种在指数BGF和PGF间的关系就自然地成立。因此，通过用z^N乘以3.10节中递归的两边

$$Q_N(u) = \frac{1}{N}\sum_{1\leq k\leq N} u^{N+1}Q_{k-1}(u)Q_{N-k}(u)$$

并对N求和，我们可以得到函数方程

$$\frac{\partial}{\partial z}Q(u,z) = u^2Q^2(u,zu) \qquad (Q(u,0)=1)$$

该方程必然被BGF所满足，它携带了足够的信息使我们能够计算这种分布的各种矩。 141

定理3.12 (Quicksort方差) 由Quicksort所使用的比较次数的方差为

$$7N^2 - 4(N+1)^2H_N^{(2)} - 2(N+1)H_N + 13N \sim N^2\left(7 - \frac{2\pi^2}{3}\right)$$

证明 在上面的讨论和下面的习题中已经对本定理证明中的计算做了概括，有些计算最好是在计算机代数系统的帮助下进行。从近似$H_N \sim \ln N$（见定理4.3的第一个推论）和$H_N^{(2)} \sim \pi^2/6$（见习题4.58）可以推出渐近估计。该结果归于Knuth[8]。 ■

在1.7节讨论过，标准差（$\approx 0.65N$）渐近地小于平均值（$\approx 2N\ln N - 0.846N$）。这意味着当使用Quicksort对随机排列排序时（或者当分割元是随机选取时），所观察到的比较次数应该

以高概率接近平均值，随着N的增加情况更是如此。

习题3.80 确认

$$q^{[1]}(z)=\frac{\partial}{\partial u}Q(u,z)\Big|_{u=1}=\frac{1}{(1-z)^2}\ln\frac{1}{1-z}$$

并证明

$$q^{[2]}(z)=\frac{\partial^2}{\partial u^2}Q(u,z)\Big|_{u=1}=\frac{6}{(1-z)^3}+\frac{8}{(1-z)^3}\ln\frac{1}{1-z}+\frac{8}{(1-z)^3}\ln^2\frac{1}{1-z}$$
$$-\frac{6}{(1-z)^2}-\frac{12}{(1-z)^2}\ln\frac{1}{1-z}-\frac{4}{(1-z)^2}\ln^2\frac{1}{1-z}$$

习题3.81 提取$q^{[2]}(z)+q^{[1]}(z)$中z^N的系数并验证定理3.12中给出的方差的准确表达式（见习题3.8）。

二叉树中树叶的分析和Quicksort所使用的比较次数的分析，在算法分析中使用二元生成函数的许多其他例子中具有代表性，我们将在第5章～第8章看到。正如我们这里的例子所阐述的，一个原因是我们使用符号论证封装算法性质以及它们的生成函数之间关系中的数据结构的性质的能力。我们的例子还表明，另一个原因是由BGF为计算矩特别是平均值所提供的方便架构。

142

表3-7 一些经典的“特殊”生成函数

二项式系数	$\frac{1}{1-z-uz}=\sum_{n,k\geq 0}\binom{n}{k}u^kz^n$
	$\frac{z^k}{(1-z)^{k+1}}=\sum_{n\geq k}\binom{n}{k}z^n$
	$(1+u)^n=\sum_{k\geq 0}\binom{n}{k}u^k$
第一类Stirling数	$\frac{1}{(1-z)^u}=\sum_{n,k\geq 0}\left[{n \atop k}\right]u^k\frac{z^n}{n!}$
	$\frac{1}{k!}\left(\ln\frac{1}{1-z}\right)^k=\sum_{n\geq 0}\left[{n \atop k}\right]\frac{z^n}{n!}$
	$u(u+1)\cdots(u+n-1)=\sum_{k\geq 0}\left[{n \atop k}\right]u^k$
第二类Stirling数	$e^{u(e^z-1)}=\sum_{n,k\geq 0}\left\{{n \atop k}\right\}u^k\frac{z^n}{n!}$
	$\frac{1}{k!}(e^z-1)^k=\sum_{n\geq 0}\left\{{n \atop k}\right\}\frac{z^n}{n!}$
	$\frac{z^k}{(1-z)(1-2z)\cdots(1-kz)}=\sum_{n\geq k}\left\{{n \atop k}\right\}z^n$
伯努利数	$\frac{z}{(e^z-1)}=\sum_{n\geq 0}B_n\frac{z^n}{n!}$
Catalan数	$\frac{1-\sqrt{1-4z}}{2z}=\sum_{n\geq 0}\frac{1}{n+1}\binom{2n}{n}z^n$
调和数	$\frac{1}{1-z}\ln\frac{1}{1-z}=\sum_{n\geq 1}H_nz^n$
阶乘	$\frac{1}{1-z}=\sum_{n\geq 0}n!\frac{z^n}{n!}$
斐波那契数	$\frac{z}{1-z-z^2}=\sum_{n\geq 0}F_nz^n$

143

3.13 特殊函数

我们已经遇到一些特殊的数——诸如调和数、Fibonacci数、二项式系数以及$N!$——的序列，这些序列对所考查的问题是内在性的，并且它们出现在如此多不同的应用中，使得它们凭借它们本身的价值是值得研究的。这一节，再扼要考虑几个这样的序列。

我们定义表3-7中的这些序列为所给生成函数中的系数。另外，还存在组合学的解释也可以用来定义这些序列，不过我们更愿意使用生成函数作为定义，这样可以避免偏向任何特定的应用。我们可以把这些生成函数看作是添加到我们“已知”函数工具箱中的工具——这些特殊的函数如此频繁地出现，以至于我们对它们的性质有了相当透彻的理解。

这些序列主要源自组合数学：它们每一个都是对某个基本组合对象的“计数”，其中有一些将在下面进行扼要的描述。例如，$N!$是N个对象的排列的个数，H_N是当从左到右通过一个随机排列的时候我们遇到的比所有此前遇到的都大的值的平均次数（见第6章）。我们将避免全面讨论特殊数的组合数学，而是重点放在第5章到第8章讨论的基本算法和基本结构中起作用的那些特殊的数。关于特殊数的更多的信息可以在像Comtet[1]、Graham、Knuth和Patashnik[4]以及Goulden和Jackson[5]等的著作中找到，这些序列也出现在分析中。例如，我们可以利用它们从表示多项式的一种方式转化成另一种方式。下面我们介绍一些例子但是避免考虑全面的细节。

算法分析或许为特殊序列的研究添加一个新的趋势：我们反对用基础算法的基本性能特性定义特殊序列的做法，虽然如第5章到第8章所讨论的那样对每个序列都能够这么做。同时，熟悉这些序列是必要的，因为它们出现得太频繁了——或者是直接出现，如在研究实际上是处理基本组合学对象的算法的时候；或者是间接出现，如在得出下面生成函数之一的时候。不管是否我们意识到特殊组合学上的联系，深刻理解这些生成函数的性质在算法分析中常常是开拓性的。第5章到第8章将讨论这些与特殊算法密切相关的序列的更多细节。 144

二项式系数。我们已经假设读者熟悉这些特殊数的性质：数$\binom{n}{k}$表示从n个对象选择k个对象（不考虑置换）的方法数；它们是多项式$(1+x)^n$依x的幂展开时所产生的系数。我们已经看到，二项式系数在算法分析中经常出现，例如当Quicksort通过首先选择一个样本而被改进时就是如此。

Stirling数。存在两类Stirling数，它们可以用来在多项式的标准表示和使用所谓的递降阶乘幂$x^{\underline{k}} = x(x-1)(x-2)\cdots(x-k+1)$的表示之间来回转换。

$$x^{\underline{n}} = \sum_k \left[{n \atop k}\right](-1)^{n-k}x^k \qquad \text{和} \qquad x^n = \sum_k \left\{{n \atop k}\right\}x^{\underline{k}}$$

Stirling数有类似于二项式系数的组合学解释：$\left\{{n \atop k}\right\}$为将$n$个对象的集合分成$k$个非空子集的方法数；而$\left[{n \atop k}\right]$则是将$n$个对象分成$k$个非空圈的方法数。我们在3.9节已经接触过$\left[{n \atop k}\right]$Stirling分布并将在第6章详细地讨论它。$\left\{{n \atop k}\right\}$Stirling分布将在第8章讨论赠券收藏家问题时出现。

伯努利（Bernoulli）数。具有EGF $z/(e^z-1)$的序列在许多组合应用中出现。例如，如果我们要想写出小于N的整数的t次幂的和的显式表达式作为N的标准多项式，那么我们就需要这

些数。通过在

$$z=(B_0+B_1z+B_2/2z^2+B_3/6z^3+\cdots)\left(z+\frac{z^2}{2}+\frac{z^3}{6}+\cdots\right)$$

的等式中置z的系数相等，我们可以推出序列中的前几项。由此可知$B_0=1$，然后$B_1+B_0/2=0$，则$B_1=-1/2$，然后$B_2/2+B_1/2+B_0/6=0$，因此$B_2=1/6$等等。若令

$$S_{Nt}=\sum_{0\le k<N}k^t$$

145

则EGF由

$$S_N(z)=\sum_{t\ge 0}\sum_{0\le k<N}k^t\frac{z^t}{t!}=\sum_{0\le k<N}e^{kz}=\frac{e^{Nz}-1}{e^z-1}$$

给出。现在这是“已知”生成函数的一个卷积，由此得到显式公式

$$S_{Nt}=\frac{1}{t+1}\sum_{0\le k\le t}\binom{t+1}{k}B_kN^{t+1-k}$$

我们有

$$\sum_{1\le k\le N}k=\frac{N^2}{2}+\frac{N}{2}=\frac{N(N+1)}{2}$$

$$\sum_{1\le k\le N}k^2=\frac{N^3}{3}+\frac{N^2}{2}+\frac{N}{6}=\frac{N(N+1)(2N+1)}{6}$$

$$\sum_{1\le k\le N}k^3=\frac{N^4}{4}+\frac{N^3}{2}+\frac{N^2}{4}=\frac{N^2(N-1)^2}{4}$$

而一般地

$$\sum_{1\le k\le N}k^t\sim\frac{N^{t+1}}{t+1}$$

除了这种基本的应用之外，伯努利数在欧拉-麦克劳林求和公式中起着实质性的作用（将在4.5节中讨论），并且它们还自然地出现在算法分析的其他应用中。例如，它们出现在与第7章讨论的数字树相关的一族算法的生成函数中。

伯努利多项式。上面出现的多项式

$$B_m(x)=\sum_k\binom{m}{k}B_kx^{m-k}$$

具有EGF

$$\sum_{m\ge 0}B_m(x)\frac{z^m}{m!}=\frac{z}{e^z-1}e^{xz}$$

146

许多有趣的性质由此可以得到证明。例如，微分该EGF得到恒等式

$$B_m'(x)=mB_{m-1}(x)\qquad(m>1)$$

我们对伯努利多项式的主要兴趣是近似积分——欧拉-麦克劳林求和公式——的解析应用，下一章将对它进行详细的考查。

习题3.82 给出下列各式的闭型表达式

$$\sum_{n,k\geq 0}\binom{n}{k}u^k\frac{z^n}{n!}\qquad \sum_{n,k\geq 0}k!\left[{n \atop k}\right]u^k\frac{z^n}{n!}\qquad \sum_{n,k\geq 0}k!\left\{{n \atop k}\right\}u^k\frac{z^n}{n!}$$

习题3.83 从生成函数证明

$$\binom{n}{k}=\binom{n-1}{k}+\binom{n-1}{k-1}$$

习题3.84 从生成函数证明

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

习题3.85 从生成函数证明

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

习题3.86 证明对所有的$m>1$有$B_m(0)=B_m(1)=B_m$。

习题3.87 从生成函数证明对于奇数$k\geq 3$，B_k为零。

其他类型的生成函数。在本章开始我们提到，那些不同于z^k和$z^k/k!$的核函数可以导出其他类型的生成函数。例如，利用k^{-z}作为核函数给出狄利克莱生成函数（DGF），这种函数在数论中和在几种算法的分析中起着重要的作用。它们最好是理解为复数z的函数，这样它们的解析性质就超出了本书的范围。然而，我们提到它们是为了激发其他核函数的效用以及阐述可能发生的形式处理的种类。对于1, 1, 1,…，其DGF为

$$\zeta(z)=\sum_{k\geq 1}\frac{1}{k^z}$$

147

即黎曼ζ函数，这个函数在解析数论中起着核心的作用。

在算法分析中，DGF通常表示序列的数论性质并且用ζ函数表示。例如，

$$\zeta(z)^2=\sum_{k\geq 1}\frac{1}{k^z}\sum_{j\geq 1}\frac{1}{j^z}=\sum_{k\geq 1}\sum_{j\geq 1}\frac{1}{(kj)^z}=\sum_{N\geq 1}\sum_{j \text{ divides } N}\frac{1}{N^z}=\sum_{N\geq 1}\frac{d_N}{N^z}$$

其中d_N是N的因子的个数。换句话说，$\zeta(z)^2$是$\{d_N\}$的DGF。

实际上，当我们需要用到数的二进制表示法的时候，DGF是特别有用的。例如，我们可以容易地求出偶数的特征序列的DGF:

$$\sum_{\substack{N\geq 1\\ N \text{ even}}}\frac{1}{N^z}=\sum_{N\geq 1}\frac{1}{(2N)^z}=\frac{1}{2^z}\zeta(z)$$

再有，虽然这些形式处理是有趣的，但是这些函数在复平面上的解析性质是它们在算法分析中使用的重要方面。详细的细节可以在[3]或[8]中找到。

通过使用其他的核——诸如$z^k/(1-z^k)$(Lambert)，$\binom{z}{k}$（牛顿），或$z^k/(1-z)(1-z^2)\cdots(1-z^k)$（欧拉）——我们得到其他类型的生成函数，它们经过多个世纪已经证明在分析中具有有用的性质。这些函数偶尔出现在算法分析中，对它们性质的探讨令人着迷。我们顺便提到它们但是并不对它们做详细地考虑，因为它们起不到OGF和EGF那样的核心作用。这方面更多的信息可以在[4]、[6]、[9]和[15]中找到。

习题3.88 证明对于任意的$k\geq 0$，其二进制表示法中以k个0结束的数的特征序列的DGF为$\zeta(z)/2^{kz}$。

习题3.89 求出函数ψ_N的DGF，ψ_N为N的二进制表示法中尾部0的个数。

习题3.90 求出$\{N^2\}$的特征函数的DGF。

习题3.91 证明

$$\sum_k \frac{z^k}{1-z^k} = \sum_N d_N z^N$$

148 其中d_N为N的因子的个数。

在组合数学、概率论、以及解析数论中长期使用生成函数，因此一系列大量的数学工具得以发展，它们实际上与算法分析有着密切的关系。我们使用生成函数一方面作为组合工具帮助处理一些重要的量的精确计数过程，同时又作为解析工具以得出问题的解。由此看来，它们在算法分析中起着核心的作用。

我们引进生成函数作为一种求解算法分析中所产生的递归的工具，目的是强调它们对所研究的量（运行时间或其他特征参数作为问题大小的函数）的直接关系。但是我们也注意到，递归其实就是序列的一个特征；而对应的生成函数本身则是另一个特征。对于许多问题，情况正是这样，直接论证能够产生生成函数的显示表达式，而递归则可以被完全避免。这是符号方法的基础，该论题在后面各章的例子中将得到展开，并在[3]中被形式化。

想要理解我们正在充分有效地分析的结构，可以应用符号方法推出那些对应结构定义的生成函数的函数方程。此时，我们能够得到关于结构的信息的所有方式。更为重要的是，我们可以分析并比较以类似方式定义的那些结构的无数变种。

在本书中，我们常常给出经典的推导，一方面是为了保持和文献的一致性，同时也是为了使读者获得在算法分析中产生的各种生成函数的使用经验。但是我们要强调，这些函数间简单的函数关系在大多数情形下都能以系统和直接的方式通过符号方法得到解释。

GF重要性的第二个主要原因是它们的解析作用。利用生成函数表示法，我们常常能够将一个问题进行变换，以观察重要的序列如何用经典的特殊数列来表示。如果准确的表示法不能得到，那么生成函数表示法使我们能够使用基于复变函数性质的强大数学方法以便获得算法的性质。对于出现在算法分析中各种大量的函数，我们可以通过用经典函数表示的展开式找出系数的渐近行为的精确估计。如果不行，那么我们可以放心，复变渐近方法对于抽取系
149 数渐近值的估计量是随时可用的，这在第4章有简要的讨论，并在[3]中详加论述。

常规、指数和二元生成函数提供一个基本的架构，它是我们能够开发一条通向分析在算法设计中起核心作用的大量基础结构的系统途径。在下一章将要深入讨论的渐近方法的帮助之下，我们能够使用这些工具获取可以预报各种重要的和有用的算法性能特征的结果。这个主题将在第5章到第8章进行详细的讨论。

参考文献

1. L. Comtet. *Advanced Combinatorics*, Reidel, Dordrecht, 1974.
2. P. Flajolet, B. Salvy, and P. Zimmerman. "Automatic average–case analysis of algorithms," *Theoretical Computer Science* **79**, 1991, 37–109.
3. P. Flajolet and R. Sedgewick. *Analytic Combinatorics*, in preparation.
4. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989

5. I. GOULDEN AND D. JACKSON. *Combinatorial Enumeration*, John Wiley, New York, 1983.
6. G. H. HARDY. *Divergent Series*, Oxford University Press, 1947.
7. D. E. KNUTH. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
8. D. E. KNUTH. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
9. N. E. NÖRLUND. *Vorlesungen über Differenzenrechnung*, Chelsea Publishing Company, New York, 1954.
10. G. POLYA, R. E. TARJAN, AND D. R. WOODS. *Notes on Introductory Combinatorics*, Birkhäuser, Boston, 1983.
11. J. RIORDAN. *Introduction to Combinatorial Analysis*, Princeton University Press, Princeton, NJ, 1980. 150
12. R. SEDGEWICK. "The analysis of quicksort programs," *Acta Informatica* **7**, 1977, 327–355.
13. R. SEDGEWICK. *Quicksort*, Garland Publishing, New York, 1980.
14. R. P. STANLEY. *Enumerative Combinatorics*, Wadsworth & Brooks/Cole, 1986.
15. R. P. STANLEY. "Generating functions," in *Studies in Combinatorics*, (M.A.A. Studies in Mathematics, **17**, G. C. Rota, ed.), The Mathematical Association of America, 1978, 100–141.
16. J. S. VITTER AND P. FLAJOLET. "Analysis of algorithms and data structures," in *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, 431–524.
17. H. WILF. *Generatingfunctionology*, Academic Press, San Diego, 1990. 151

第4章 渐近逼近

在算法分析中，我们最初始的想法一般都倾向于导出准确的数学结果。然而，这种准确的解并不总是能够得到的，或者如果能够得到准确解，它们也可能太复杂，而没有多大用处。本节，我们将考查导出问题的近似解或逼近准确解的一些方法。因此，我们可能要把最初的倾向改成导出对所关注的量的简洁而准确的估计。

本章的主要目的是用类似于第3章的方式，对渐近展开式的基本性质给出一个综述，给出操作这些展开式的方法以及那些我们在算法分析中最常遇到的展开式。有时，这似乎使我们远离了算法分析，不过，我们将继续从第1章引入的与特定算法有关的问题中直接提取一些例子和习题，并为第5章到第8章中将要进行的多种算法的研究做一些准备工作。正如我们一直在做的那样，本章我们将把精力集中在来自于实际分析的方法上。用于复分析的渐近方法是[11]的一个主要论题，我们将在4.10节中对这些方法所基于的原理给出简要的讨论。

我们已经看到计算机算法分析要涉及离散数学中的工具，得出对答案的最容易表达的方式是离散函数（如调和数或二项式系数）而不是来自于分析中更常见的函数（如对数或指数）。然而，这两种类型的函数在一般情况下都有着紧密的联系，因此进行渐近分析的原因之一就是在两者之间进行“翻译”。

通常，一个问题具有“规模”的概念，我们感兴趣的是那些当规模变大时越来越准确的逼近。就数学的本质和我们所求解的问题而言，如果我们的答案要用一个单一的参数N来表示，那么这些答案通常就会（基本上）按照N和$\log N$的渐近级数来表示。这些级数并不一定要收敛（事实上它们通常是发散的），但它们初始的一些项能对算法分析中引出的许多的量给出非常准确的估计。我们的一般方法就是把这些量转换成这样的级数，然后用严格定义的方法对它们进行操作。

考虑渐近方法的一个动机就是要找到一种方便的方法，用来对我们所关注的量的特定值计算出好的逼近。我们的另一个动机是要对所有的量找到一个规范型，以便更容易地对这些量进行比较和结合。例如，我们在第1章中看到，与最优值$\lg N!$相比，知道Quicksort法所进行的比较次数接近$2NH_N$对我们是很有帮助的，但更有用的是知道二者都与$N\lg N$成正比，前者的比值是1.4421...，后者的比值是1，我们甚至可以得到更为准确的估计。

另外一个例子是我们在6.6节中将要遇到的一个简单排序算法，它的平均运行时间取决于量 $N4^{N-1}\Big/\binom{2N}{N}$。这是一个简洁而准确的结果，但要估计出该算法的运行时间，我们或许想知道，比如说，$N = 1000$时，这个量的值是多少。对较大的N而言，利用上述公式求值并不是一件简单的任务，因为公式中涉及两个很大的数相除，或是涉及重新安排计算以避免大数相除。本章，我们将看到如何证明这个量是非常接近于 $N\sqrt{\pi N}/4$的，当$N = 1000$时，用 $N\sqrt{\pi N}/4$求得的值为14 012，而准确值约为14 014，所以用 $N\sqrt{\pi N}/4$计算出来的值偏离准确值的程度大约只有万分之一。更重要的是与以前一样，近似结果把这个量的值随N增加而增加的方式很清晰地表现了出来（例如，通过检验我们知道，相应于$100N$的值大约是相应于N的值的1000倍），这就使我们把这个近似结果与其他算法或相同算法的其他形式所得到的类似结果进行比较变得更容易了。

使用近似值的另一个重要原因是：近似值能大大简化在许多问题的分析中可能要涉及到的符号运算，从而导出用其他方法可能得不到的简洁答案。在第2章中，当我们讨论递归的解时，就曾涉及到这一点，我们是通过求解类似的更简单的递归，然后再估计误差的办法来求解的。渐近分析提供了一个系统的方法来辅助这样的论证。

本章的一个主要论题就是我们计算和式的近似值的处理方法，因为准确的计算可能是困难的或是不可能的。特别地，我们将考虑如何利用欧拉-麦克劳林求和公式、通过使用积分逼近和式的方法来计算和式的值。我们还将考虑计算和式的拉普拉斯方法，通过调整求和范围，做出适用于不同范围的不同逼近。

154

我们还将考虑应用这些概念的几个例子，以求得第3章中所引入的某些特殊数列的近似值以及可能会出现在算法分析中的其他一些量的近似值。特别地，我们将较详细地考察*Ramanujan-Knuth*的Q-函数及其相关的分布，这些分布在分析中经常会出现。然后，我们将考虑在各种情形下二项分布的极限。正态逼近（normal approximation）和泊松逼近（Poisson approximation）是算法分析中非常有用的经典结果，它们也为本章所开发的工具提供了极好的应用范例。

关于这些论题的标准参考文献是由De Bruijn[6]所写的著作，任何对渐近分析真正感兴趣的人都应该读一读这本书。由Odlyzko[18]所给出的一个最新资料也提供了大量的信息和丰富的例子。关于正态和泊松逼近的具体资料可以在Feller[8]中找到。我们所考虑的这些论题中的许多细节问题也可以在[2]、[12]、[13]、[15]、[19]以及本章末尾所列出的其他参考文献中找到。[11]中详细讨论了基于复分析的各种方法。

4.1 有关渐近逼近的记号

下面的记号被广泛地用来对函数的近似值进行精确的说明，这些记号至少要追溯到上世纪初。

定义 给定一个函数$f(N)$，我们记

$g(N)=O(f(N))$，当且仅当$N\to\infty$时$|g(N)/f(N)|$上有界，

$g(N)=o(f(N))$，当且仅当$N\to\infty$时$g(N)/f(N)\to 0$，

$g(N)\sim f(N)$，当且仅当$N\to\infty$时$g(N)/f(N)\to 1$。

大O-记号和小o-记号提供了表示上界的方法（小o用于表示更强的结论），~-记号提供了表示渐近相等的方法。这里的大O-记号与第1章中我们用于计算复杂性的讨论时所给出的定义是一致的。人们已经提出了大量类似的记号和定义，对这些记号的含义感兴趣的读者可以阅读[6]或[12]中所给出的论述。

155

习题4.1 证明：$N/(N+1)=O(1)$，$2^N=o(N!)$和$\sqrt[N]{e}\sim 1$。

习题4.2 证明：$\dfrac{N}{N+1}=1+O\left(\dfrac{1}{N}\right)$及$\dfrac{N}{N+1}\sim 1-\dfrac{1}{N}$。

习题4.3 证明：如果$\alpha<\beta$，则$N^\alpha=o(N^\beta)$。

习题4.4 证明：对固定的r，$\dbinom{N}{r}=\dfrac{N^r}{r!}+O(N^{r-1})$以及$\dbinom{N+r}{r}=\dfrac{N^r}{r!}+O(N^{r-1})$。

习题4.5 证明：对所有的$\varepsilon>0$，有$\log N=o(N^\varepsilon)$。

习题4.6 证明：$\dfrac{1}{2+\ln N}=o(1)$和$\dfrac{1}{2+\cos N}=O(1)$但不等于$o(1)$。

正如我们在下面将要看到的，通常情况下我们并不需要直接应用定义来确定所关注的量的渐近值，因为大O-记号使我们有可能利用一组较少的基本代数操作得到逼近。

当在任意给定的点附近逼近实变量或复变量的函数时，我们将使用相同的记号。例如，我们称

$$当x\to\infty时，\quad \frac{1}{1+x}=\frac{1}{x}-\frac{1}{x^2}+\frac{1}{x^3}+O\left(\frac{1}{x^4}\right)$$

以及

$$当x\to 0时，\quad \frac{1}{1+x}=1-x+x^2-x^3+O(x^4)$$

涉及这种应用的更一般的大O-记号定义可以通过在上面的定义中将$N\to\infty$ 换成$x\to x_0$而得到，其中可以对x作任何限制（例如x必须是整数、实数或复数等）。极限值x_0通常是0或∞，但它也可以是任何其他值。一般从上下文中我们可以明显地看出所涉及的是一组什么样的数以及所涉及的极限是什么，因此一般情况下我们就写成“$x\to x_0$”或“$N\to\infty$”。当然，上述说明同样也适用于小o-记号和~-记号。 156

在算法分析中，我们避免诸如“此量的平均值为$O(f(N))$”这样的直接用法，因为这种用法给出的信息不足以予报性能。实际上，我们力求用大O-记号对那些其值远远小于主项或“首项”值的“误差”项来定界。非正式地讲，我们期望所涉及的项对于较大的N而言，其值小到可以忽略的程度。

大O-逼近。我们说$g(N)=f(N)+O(h(N))$，是指我们可以通过计算$f(N)$来逼近$g(N)$，且误差将在$h(N)$的一个常数因子范围内。像以往一样，用大O-记号时，所涉及的常数可以不指定，但通常假定其值不是很大。正如下面我们要讨论的那样，我们经常把此记号与$h(N)=o(f(N))$一起使用。

小o-逼近。一个更强的表述是：我们说$g(N)=f(N)+o(h(N))$，是指我们可以通过计算$f(N)$来逼近$g(N)$，且随着N的增大，误差与$h(N)$相比变得越来越小。在减小的速率中涉及了一个没有指定的函数，但通常假定此函数在数值上永远不会太大（即使是对较小的N而言）。

~-逼近。记号$g(N)\sim f(N)$ 用于表示最弱的非平凡小o-逼近$g(N)=f(N)+o(f(N))$。

以上这些记号是很有用的，因为它们使我们能够隐匿不重要的细节而不会失去数学上的严密性和结果的精确性。如果我们需要一个更精确的结果，我们就可以得到它，否则大部分细节性的计算都可以被隐匿掉。我们最感兴趣的是能够保持这种“潜在准确”的方法，因为只要我们愿意，就能够算出任意细致精度的答案。

指数级小项。当涉及到对数和指数时，我们值得花费一些精力去认识“指数级差别”，以避免那些对我们所关注的量的最终结果没有实质性影响的计算。例如，如果我们知道一个量的值为$2N+O(\log N)$，那么我们就有理由确信当N为一千或一百万的时候，$2N$是在真值的百分之几或十万分之几的范围之内，因此我们没有必要去找出$\log N$的系数或是将其展开式精确到 157
$O(1)$的范围之内。类似地，一个渐近估计$2^N+O(N^2)$ 也是非常准确的。但另一方面，知道一个量为$2N\ln N+O(N)$时，可能就不足以确信其估计值的相对误差是在一个因子2的范围之内，即使当N是一百万的时候也是一样。为了强调指数级差别，我们有时非正式地把一个量称为指数级小量，如果该量小于N的任意负数幂的话，即：对任意正数M，该量小于$O(1/N^M)$。典型的指数级小量有e^{-N}、$e^{-\log^2 N}$以及$(\log N)^{-\log N}$等。

习题4.7 证明：对任意正的常数ε，e^{-N^ε}是指数级小量。（即：对给定的ε，证明对任意固定的$M>0$，$e^{-N^\varepsilon}=O(N^{-M})$。）

习题4.8 证明：$e^{-\log^2 N}$和$(\log N)^{-\log N}$都是指数级小量。

习题4.9 如果$\alpha<\beta$，证明α^N相对于β^N而言是指数级小量。对$\beta=1.2$和$\alpha=1.1$，当用β^N逼近$\alpha^N+\beta^N$时，分别对$N=10$和$N=100$求逼近值的绝对误差和相对误差。

习题4.10 证明：一个指数级小量与N的任意一个多项式的乘积是一个指数级小量。

习题4.11 找出下列各递推关系中所隐含的关于a_n的最精确的表达式：

$$a_n = 2a_{n/2} + O(n)$$
$$a_n = 2a_{n/2} + o(n)$$
$$a_n \sim 2a_{n/2} + n$$

在每种情况下，都假定$a_{n/2}$是关于$a_{\lfloor n/2\rfloor}+O(1)$的简写。

习题4.12 利用第1章中的定义，求下列各递推关系中所隐含的关于a_n的最精确的表达式：

$$a_n = 2a_{n/2} + O(n)$$
$$a_n = 2a_{n/2} + \Theta(n)$$
$$a_n = 2a_{n/2} + \Omega(n)$$

在每种情况下，都假定$a_{n/2}$是$a_{\lfloor n/2\rfloor}+O(1)$的简写。

习题4.13 设$\beta>1$，并取$f(x)=x^\alpha$，其中$\alpha>0$。如果$a(x)$满足递归

158

$$\text{当}x\geqslant 1\text{时，}a(x)=a(x/\beta)+f(x)\text{；当}x<1\text{时，}a(x)=0$$

及$b(x)$满足递归

$$\text{当}x\geqslant 1\text{时，}b(x)=b(x/\beta+c)+f(x)\text{；当}x<1\text{时，}b(x)=0$$

证明 当$x\to\infty$时，$a(x)\sim b(x)$。将你的证明进行推广，使其能应用到更广的一类函数$f(x)$。

线性递归的渐近性。线性递归提供了渐近表达式可以导致实质简化的方式的解释。我们在2.4节和3.3节中曾经看到：任何线性递推序列$\{a_n\}$都有一个有理的OGF和一个形如$\beta^n n^j$的一些项的线性组合。从渐近的角度而言，很明显其中只有几项需要考虑，因为那些β较大的项相对于那些β较小的项而言处于指数级的支配地位（见习题4.9）。例如，在2.3节中我们知道

$$a_n = 5a_{n-1} - 6a_{n-2} \quad (n>1;\ a_0=0,\ a_1=1)$$

的准确解为3^n-2^n，但近似解3^n在$n>25$时能准确到十万分之几的范围内。总之，我们只需要追踪那些与最大绝对值或最大模数有关的项。

定理4.1（线性递归的渐近性） 假设一个有理生成函数$f(z)/g(z)$具有唯一一个最小模数的极点$1/\beta$（即$g(1/\alpha)=0$，且$\alpha\neq\beta$意味着$|1/\alpha|>|1/\beta|$或$|\alpha|<|\beta|$），其中$f(z)$与$g(z)$互素且$g(0)\neq 0$，那么，若$1/\beta$的重数是ν，则我们有

$$[z^n]\frac{f(z)}{g(z)} \sim C\beta^n n^{\upsilon-1}\text{，其中}\quad C=\upsilon\frac{(-\beta)^\upsilon f(1/\beta)}{g^{(\upsilon)}(1/\beta)}$$

证明 根据3.3节中的讨论，$[z^n]f(z)/g(z)$可以表示成一些项的和，它与$g(z)$的每个根$1/\alpha$相关联，即每一项都具有$[z^n]\,c_0(1-\alpha z)^{-\upsilon_\alpha}$的形式，其中$\nu_\alpha$是$\alpha$的重数。对所有$|\alpha|<|\beta|$的$\alpha$，这样的项相对于对应于$\beta$的项而言是指数级小量，因为

$$[z^n]\frac{1}{(1-\alpha z)^{\upsilon_\alpha}} = \binom{n+\upsilon_\alpha-1}{\upsilon_\alpha-1}\alpha^n$$

且对任意非负的M，$\alpha^n n^M = o(\beta^n)$（见习题4.10）。 159

因此，我们只需考虑与β相关的项:

$$[z^n]\frac{f(z)}{g(z)} \sim [z^n]\frac{c_0}{(1-\beta z)^v} \sim c_0\binom{n+v-1}{v-1}\beta^n \sim \frac{c_0}{(v-1)!}n^{v-1}\beta^n$$

（见习题4.4），所以下面剩下的问题就是要确定c_0，因为$(1-\beta z)$不是$f(z)$的因子，于是，由洛必达法则立即得出:

$$c_0 = \lim_{z\to 1/\beta}(1-\beta z)^v\frac{f(z)}{g(z)} = f(1/\beta)\frac{\lim_{z\to 1/\beta}(1-\beta z)^v}{\lim_{z\to 1/\beta}g(z)} = f(1/\beta)\frac{v!(-\beta)^v}{g^{(v)}(1/\beta)}$$

■

对于导致$g(z)$具有唯一最小模数极点的递归，该定理给出了确定解的渐近增长的方法，包括首项系数的计算方法。如果$g(z)$的最小模数的极点不止一个，那么在对应于这些极点的项中，具有最高重数的那些项将处于支配的地位（但不是指数级的）。这就导致确定线性递归的解的渐近增长的一般方法，它是对3.3节末尾所给出的求准确解的方法的一个改进。

- 从递归中导出$g(z)$;
- 根据$g(z)$和初始条件计算$f(z)$;
- 消去$f(z)/g(z)$的公因子。这可以通过分解$f(z)$和$g(z)$，并消去公因子来实现，但这并不需要对函数进行完全多项式分解，只需计算最大公因子;
- 在那些最小模数的项中，确定对应于最高重数极点的那些项;
- 利用定理4.1来确定系数。正如前面所指出的那样，对于较大的n，这种方法将会给出非常精确的答案，因为所忽略的那些项相对于所保留的那些项而言是指数级小量。

这个过程将立即导出对线性递归的解的简洁而准确的逼近。例如考虑递归

$$a_n = 2a_{n-1} + a_{n-2} - 2a_{n-3} \qquad (n>2;\ a_0=0,\ a_1=a_2=1)$$

在3.3节中我们已得出，其解的生成函数是

$$a(z) = \frac{f(z)}{g(z)} = \frac{z}{(1+z)(1-2z)}$$

这里$\beta=2$，$v=1$，$g'(1/2)=-3$和$f(1/2)=1/2$，所以由定理4.1知：$a_n \sim 2^n/3$，这和以前的结果是一致的。 160

习题4.14 利用定理4.1求递归

$$a_n = 5a_{n-1} - 8a_{n-2} + 4a_{n-3} \qquad (n>2;\ a_0=1,\ a_1=2,\ a_2=4)$$

的一个渐近解。当关于a_0和a_1的初始条件变为$a_0=1$和$a_1=2$时，求解同一递归。

习题4.15 利用定理4.1求递归

$$a_n = 2a_{n-2} - a_{n-4} \qquad (n>4;\ a_0=a_1=0,\ a_2=a_3=1)$$

的一个渐近解。

习题4.16 利用定理4.1求递归

$$a_n = 3a_{n-1} - 3a_{n-2} + a_{n-3} \qquad (n>2;\ a_0=a_1=0,\ a_2=1)$$

的一个渐近解。

习题4.17 [Miles，参看Knoth]证明多项式$z^t - z^{t-1} - \cdots - z - 1$具有$t$个不同的根，且对所

有的$t>1$，恰好只有一个根的模数大于1。

习题4.18 关于“第t阶斐波那契”递归

$$F_N^{[t]} = F_{N-1}^{[t]} + F_{N-2}^{[t]} + \cdots + F_{N-t}^{[t]}$$

其中$N \geqslant t$，$F_0^{[t]} = F_1^{[t]} = \cdots = F_{t-2}^{[t]} = 0$及$F_{t-1}^{[t]} = 1$，给出它的一个近似解。

习题4.19 [Schur] 证明：利用硬币面值$d_1, d_2, \cdots, d_t$(且$d_1 = 1$)兑换一张N元面值纸币的方式数渐近于

$$\frac{N^{t-1}}{d_1 d_2 \cdots d_t (t-1)!}$$

(见习题3.55)。

4.2 渐近展开式

正如上面所提到的，与方程$f(N) = O(g_0(N))$相比，我们更喜欢方程$f(N) = c_0 g_0(N) + O(g_1(N))$，其中$g_1(N) = o(g_0(N))$，因为该方程给出了常数$c_0$，所以当$N$增大时，它使我们在对$f(N)$的特定估计中，能够改进估计的精度。如果$g_0(N)$和$g_1(N)$比较接近，我们可能希望找到一个对应于$g_1$的常数，从而导出一个“更精确的”表达式：如果$g_2(N) = o(g_1(N))$，则我们可写成$f(N) = c_0 g_0(N) + c_1 g_1(N) + O(g_2(N))$。例如对于实际的$N$值，用展开式$2N\ln N + (2\gamma - 2)N + O(\log N)$来估计Quicksort所需要的平均比较次数时，比用表达式$2N\ln N + O(N)$来估计要精确得多，添加项$O(\log N)$和$O(1)$后，估计结果将更加准确，如表4-1所示。

161

表4-1 关于Quicksort比较次数的渐近估计

N	$2(N+1)(H_{N+1}-1)$	$2N\ln N$	$+(2\gamma-2)N$	$+2(\ln N+\gamma)+1$
10	44.43	46.05	37.59	44.35
100	847.85	921.03	836.47	847.84
1000	12 985.91	13 815.51	12 969.94	12 985.91
10 000	175 771.70	184 206.81	175 751.12	175 771.70

渐近展开（asymptotic expansion）的概念由Poincaré（参看[6]）研究并提出，它推广了这个想法：

定义 给定一个函数序列$\{g_k(N)\}_{k>0}$，当$k \geqslant 0$时，满足$g_{k+1}(N) = o(g_k(N))$，则称

$$f(N) \sim c_0 g_0(N) + c_1 g_1(N) + c_2 g_2(N) + \cdots$$

为关于f的一个渐近级数，或f的一个渐近展开式。渐近级数代表如下一系列公式

$$\begin{aligned}
f(N) &= O(g_0(N)) \\
f(N) &= c_0 g_0(N) + O(g_1(N)) \\
f(N) &= c_0 g_0(N) + c_1 g_1(N) + O(g_2(N)) \\
f(N) &= c_0 g_0(N) + c_1 g_1(N) + c_2 g_2(N) + O(g_3(N)) \\
&\vdots
\end{aligned}$$

并称$g_k(N)$为一个渐近级。

我们从渐近级数中每取一个项，都会给出一个更准确的渐近估计。在算法分析中，许多

常见函数都存在完全渐近级数，原则上我们主要考虑能够展开，以提供描述相关量的渐近展开式的方法。我们用~-记号简单表示去掉误差项中的信息，或者我们也可用大O-记号或小o-记号来给出更具体的信息。 162

这是对4.1节开始部分所给出的~-记号定义的一个扩充。其原来的用法仅涉及到右端的一个项，而当前的定义要求一系列的（递减的）项。

实际上，我们主要还是处理有限的展开式，而不是（无限的）渐近级数，例如我们使用记号

$$f(N) \sim c_0 g_0(N) + c_1 g_1(N) + c_2 g_2(N)$$

来表示一个有限展开式，其隐含的误差项为$o(g_2(N))$。我们最常使用的形式是：通过简单地截断渐近级数而得到形如

$$f(N) = c_0 g_0(N) + c_1 g_1(N) + c_2 g_2(N) + O(g_3(N))$$

的形式。在实际中我们对一个逼近通常只使用几项（可能三项或四项），因为在一般情况下，我们要得到的只是一个渐近级，对于较大的N而言，它可以使后面的项与前面的项相比要小得多。在表4-1所示的Quicksort的例子中，“更精确的”公式$2N\ln N + (2\gamma - 2)N + 2\ln N + 2\gamma + 1$对$N = 10$就已经给出了绝对误差小于0.1的结果。

习题4.20 扩充表4-1，使其包含$N = 10^5$和10^6时的情况。

Poincaré法的全面推广允许渐近展开式用递减的任意无穷函数级数来表示（在o-记号的意义下）。然而，我们最常关注的却是一组非常有限的函数：事实上，当随着N的增大逼近函数时，我们经常可以按照N的递减方幂来表示逼近。偶尔我们也可能用到其他函数，但通常我们满足于由N的幂、$\log N$，重复对数（如$\log\log N$）以及指数等乘积的递减级数的项所构成的渐近级。

当寻找一个渐近估计时，为得到结果所需要的精度，我们并不需要确定展开式中应该带有多少项。例如，我们经常需要减去或除以一些量，而对于这些量我们仅有它们的渐近估计，因此也可能会发生删除现象，它需要更多的项。一般情况下，在展开式中带有三项或四项，我们也许会重新推导，以使展开式更加精简，或一旦我们得知结果的本性后，对展开式增加更多的项。 163

泰勒展开式。泰勒级数是许多渐近展开的源泉：每个（无限的）泰勒展开式都能产生一个当$x \to 0$时的渐近级数。

表4-2给出了某些基本函数的渐近展开式，它们通过截断泰勒级数而导出。这些展开式都是经典的，并可根据泰勒定理立即推出。在后面的各节中，我们将利用这些展开式来描述处理渐近级数的方法，其他类似的展开可根据前一章中所给出的生成函数立即得到。表中前四个展开式可用作我们将要进行的许多渐近计算的基础（事实上前三个就足够了，因为几何展开是二项展开的特殊情况）。

作为应用表4-2的一个典型例子，考虑当$N \to \infty$时，求$\ln(N - 2)$的一个渐近展开的问题。我们可以通过抽取展开式中的第一项而求之，写成

$$\ln(N-2) = \ln N + \ln\left(1 - \frac{2}{N}\right) = \ln N - \frac{2}{N} + O\left(\frac{1}{N^2}\right)$$

也就是说，为了能应用表4-2，我们求出一个满足$x \to 0$时的替换（$x = -2/N$）。

表4-2 由泰勒级数（x→0）导出的渐近展开式

指数	$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + O(x^4)$
对数	$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} + O(x^4)$
二项式	$(1+x)^k = 1 + kx + \binom{k}{2}x^2 + \binom{k}{3}x^3 + O(x^4)$
几何	$\frac{1}{1-x} = 1 + x + x^2 + x^3 + O(x^4)$
三角函数	$\sin(x) = x - \frac{x^3}{6} + \frac{x^5}{120} + O(x^7)$
	$\cos(x) = 1 - \frac{x^2}{2} + \frac{x^4}{24} + O(x^6)$

164

或者，我们可以利用泰勒展开式中更多的项，以获得一个更一般的近似结果。例如，展开式

$$\ln(N+\sqrt{N}) = \ln N + \frac{1}{\sqrt{N}} - \frac{1}{2N} + O\left(\frac{1}{N^{3/2}}\right)$$

可通过先将$\ln N$分解出来，然后在$\ln(1+x)$的展式中取 $x = 1/\sqrt{N}$ 而得到。这种处理手段是典型的，我们将在后面看到许多例子。

习题4.21 当$x \to 0$时，将$\ln(1-x+x^2)$展开到$O(x^4)$。

习题4.22 给出关于$\ln(N^\alpha + N^\beta)$的一个渐近展开式，其中α和β为正常数，且$\alpha > \beta$。

习题4.23 给出关于 $\frac{N}{N-1}\ln\frac{N}{N-1}$ 的一个渐近展开式。

习题4.24 不用计算器，估计$e^{0.1} + \cos(0.1) - \ln(0.9)$的值，准确到$10^{-4}$。

习题4.25 证明：

$$\frac{1}{9801} = 0.000\ 102\ 030\ 405\ 060\ 708\ 091\ 011\cdots 474\ 849\ 50\cdots$$

准确到10^{-100}。你还能再多预测多少位数？给出一般性的结论。

非收敛渐近级数。任何收敛级数都能导出一个完全的渐近逼近，但反之却不一定成立——一个渐近级数完全可以是发散的，注意到这一点很重要。例如，我们可能有一个函数

$$f(N) \sim \sum_{k \geq 0} \frac{k!}{N^k}$$

这意味着（比如说）

$$f(N) = 1 + \frac{1}{N} + \frac{2}{N^2} + \frac{6}{N^3} + O\left(\frac{1}{N^4}\right)$$

165

尽管该函数的无限和是不收敛的。为什么允许这么做呢？如果我们从展开式中取固定个数的项，那么当$N \to \infty$ 时，定义中所隐含的相等是有意义的。也就是说，我们有一系列越来越好的逼近，但它们开始给出有用信息的点却变得越来越大。

Stirling公式。在发散的渐近级数中，一个最著名的例子就是Stirling公式，该公式是这样开始的

$$N! = \sqrt{2\pi N}\left(\frac{N}{e}\right)^N \left(1 + \frac{1}{12N} + \frac{1}{288N^2} + O\left(\frac{1}{N^3}\right)\right)$$

在4.6节中，我们将利用一个方法来指出该公式是如何导出来的，该方法能给出一个按N的幂次递减的完全（但却是发散的）级数。在实际中，该级数发散的这一事实并不重要，因为它的前几项能够给出极为准确的估计，正如表4-3所阐明的，我们在后面将进一步详细讨论。目前，严格地说，大O-记号中所暗示的常数意味着这样的一个公式并不能就N的具体值给出完整的信息，因为该常数是任意的（或未指定的）。但原则上，我们总能借助于渐近级数的资源，并通过证明关于这个常数的特定的界来克服这一缺点。例如我们能够证明：对所有的$N>1$，

$$N!=\sqrt{2\pi N}\left(\frac{N}{e}\right)^N\left(1+\frac{\theta_N}{12N}\right)$$

其中$0<\theta_N<1$（例如参见[1]）。和在这个例子中的情况一样，在一般情况下，假定大O-记号中所隐含的常数较小是安全的、而不必寻找精确的误差界。一般地，如果所需要的精度更高一些，那么对于足够大的N，渐近级数中的下一项将最终能提供这个精度。

表4-3 关于N!的Stirling公式的精度

N	$N!$	$\sqrt{2\pi N}\left(\frac{N}{e}\right)^N\left(1+\frac{1}{12N}+\frac{1}{288N^2}\right)$	绝对误差	相对误差
1	1	1.002 183 625	0.002 2	10^{-2}
2	2	2.000 628 669	0.000 6	10^{-3}
3	6	6.000 578 155	0.000 6	10^{-4}
4	24	24.000 988 29	0.001	10^{-4}
5	120	120.002 545 7	0.002	10^{-4}
6	720	720.008 870 1	0.009	10^{-4}
7	5 040	5 040.039 185	0.039	10^{-5}
8	40 320	40 320.210 31	0.210	10^{-5}
9	362 880	362 881.330 7	1.33	10^{-5}
10	3 628 800	3 628 809.711	9.71	10^{-5}

习题4.26 利用非渐近形式的Stirling公式，给出用 $N\sqrt{\pi N}/4$来估计 $N4^{N-1}/\binom{2N}{N}$ 时的一个误差界。

绝对误差。正如上面所定义的，一个有限的渐近展开式只有一个大O-项，我们下面将要讨论的是：如何进行各种标准操作而保持这一性质不变。如果可能，我们将力求以$f(N)=g(N)+O(h(N))$的形式来表示最终的答案，从而使由大O-记号所表示的未知误差，当N增加时，在绝对意义下变得可以忽略不计（这也意味着$h(N)=o(1)$）。在一个渐近级数中，我们可以通过在$g(N)$中包含更多的项，并取较小的$h(N)$来获得较准确的估计。例如，表4-4指出如何向调和数的渐近级数中加入项，以给出更准确的估计。我们在后面将指出这个级数是如何导出的。和Stirling公式一样，该级数是一个发散的渐近级数。

相对误差。我们总可以用另外一种形式$f(N)=g(N)(1+O(h(N))$来表示估计，其中$h(N)=o(1)$。在某些情况下，一个绝对误差可能随着N的增大而增大，对此我们也只能表示认可。相对误差将随着N的增大而减小，但当我们试图计算$f(N)$时，绝对误差未必是“可忽略不计的”。当$f(N)$是指数增长的情况时，我们会经常遇到这种类型的估计。例如，表4-3给出了Stirling公式中的绝对误差和相对误差。Stirling展开式的对数给出了一个关于$\ln N!$的渐近级数，其绝对误差非常小，如表4-5所示。

表4-4 调和数的渐近估计

N	H_N	$\ln N$	$+\gamma$	$+\frac{1}{2N}$	$+\frac{1}{12N^2}$
10	2.928 968 3	2.302 585 1	2.879 800 8	2.929 800 8	2.928 967 4
100	5.187 377 5	4.605 170 2	5.182 385 9	5.187 385 9	5.187 377 5
1 000	7.485 470 9	6.907 755 3	7.484 970 9	7.485 470 9	7.485 470 9
10 000	9.787 606 0	9.210 340 4	9.787 556 0	9.787 606 0	9.787 606 0
100 000	12.090 146 1	11.512 925 5	12.090 141 1	12.090 146 1	12.090 146 1
1 000 000	14.392 726 7	13.815 510 6	14.392 726 2	14.392 726 7	14.392 726 7

表4-5 Stirling公式中关于lnN!的绝对误差

N	$\ln N!$	$\left(N+\frac{1}{2}\right)\ln N-N+\ln\sqrt{2\pi}+\frac{1}{12N}$	误 差
10	15.104 413	15.104 415	10^{-6}
100	363.739 375 556	363.739 375 558	10^{-11}
1000	5 912.128 178 488 163	5 912.128 178 488 166	10^{-15}
1000	82 108.927 836 814 353 345 5	82 108.927 836 814 353 345 8	10^{-19}

我们通常只是在处理关于N的指数级大量时才使用“相对误差”，如$N!$ 或Catalan数。在算法分析中，这样的量通常出现在计算过程中的中间阶段；以后，诸如将这样的两个量相除或者取对数的运算等，就把我们又带回到绝对误差的范畴，对于应用中我们所关注的大多数的量而言都是如此。

当我们利用累积计数法计算平均值的时候，这种情况是很正常的。例如在第3章中，为了求出二叉树中的叶子数，我们计算出了所有树中的树叶总数，然后再将其除以Catalan数。在这种情况下，我们能够计算出一个准确的结果，但对许多其他问题而言，通常是将两个渐近估计相除。事实上，这个例子阐明了我们利用渐近性质的一个主要原因。在一棵树中，比如说1000个结点的树中，满足某个性质的结点的平均个数无疑将小于1000，因此我们就有可能利用生成函数导出以Catalan数和二项式系数表示的这个数的一个准确公式。但要计算出这个数（可能会涉及乘以和除以像2^{1000}或1000! 这样的数），如果不利用渐近性质，可能会是一件相当复杂的事情。在下一节中，我们将给出操作渐近展开式的基本技巧，这些技巧将使我们能导出这些情况下的精确渐近估计。

表4-6给出了组合数学及算法分析中经常遇到的一些特殊数列的渐近级数。这些逼近中的大多数都作为操作和导出渐近级数的例子在后面进行了推导。在本书中后面我们将经常引用这些展开式，因为在研究算法的性质时，数列本身的出现是件自然而然的事情，因此，渐近展开式为我们精确地量化算法的性能特征和恰当地进行算法比较提供了一种方便的方式。

166 ~ 168

表4-6 一些特殊权的渐近展开式（N→∞）

阶乘（Stirling公式）	$N!=\sqrt{2\pi N}\left(\frac{N}{e}\right)^N\left(1+\frac{1}{12N}+\frac{1}{288N^2}+O\left(\frac{1}{N^3}\right)\right)$
	$\ln N!=\left(N+\frac{1}{2}\right)\ln N-N+\ln\sqrt{2\pi}+\frac{1}{12N}+O\left(\frac{1}{N^3}\right)$
调和数	$H_N=\ln N+\gamma+\frac{1}{2N}-\frac{1}{12N^2}+O\left(\frac{1}{N^4}\right)$

（续）

二项式系数	$\binom{N}{k}=\frac{N^k}{k!}\left(1+O\left(\frac{1}{N}\right)\right)$，$k=O(1)$
	$=\frac{2^{N/2}}{\sqrt{\pi N}}\left(1+O\left(\frac{1}{N}\right)\right)$，$k=\frac{N}{2}+O(1)$
二项式分布的正态逼近	$\binom{2N}{N-k}\frac{1}{2^{2N}}=\frac{e^{-k^2/N}}{\sqrt{\pi N}}+O\left(\frac{1}{N^{3/2}}\right)$
二项式分布的泊松逼近	$\binom{N}{k}p^k(1-p)^{N-k}=\frac{\lambda^k e^{-\lambda}}{k!}+o(1)$，$p=\lambda/N$
第一类Stirling数	$\left[{N \atop k}\right]\sim\frac{(N-1)!}{(k-1)!}(\ln N)^{k-1}$，$k=O(1)$
第二类Stirling数	$\left\{{N \atop k}\right\}\sim\frac{k^N}{k!}$，$k=O(1)$
伯努利数	$B_{2N}=(-1)^N\frac{(2N)!}{(2\pi)^{2N}}(-2+O(4^{-N}))$
Catalan数	$T_N=\frac{1}{N+1}\binom{2N}{N}=\frac{4^N}{\sqrt{\pi N^3}}\left(1+O\left(\frac{1}{N}\right)\right)$
斐波那契数	$F_N=\frac{\phi^N}{\sqrt{5}}+O(\phi^{-N})$　　其中　$\phi=\frac{1+\sqrt{5}}{2}$

习题4.27 假设在大O-记号中所隐含的常数C的绝对值小于10。给出由绝对公式$H_N=\ln N+\gamma+O(1/N)$和相对公式$H_N=\ln N(1+O(1/\log N))$所隐含的H_{1000}的特定的界。

习题4.28 假设在大O-记号中所隐含的常数C的绝对值小于10。给出由相对公式

$$\frac{1}{N+1}\binom{2N}{N}=\frac{4^N}{\sqrt{\pi N^3}}\left(1+O\left(\frac{1}{N}\right)\right)$$

所隐含的第10个Catalan数的特定的界。

习题4.29 假设对于$N\geqslant N_0$，确定了$f(N)$的一个收敛的表示式

$$f(N)=\sum_{k\geqslant 0}a_kN^{-k}$$

其中N_0是一个固定的常数。证明：对任意的$M>0$，都有

$$f(N)=\sum_{0\leqslant k<M}a_kN^{-k}+O(N^{-M})$$

习题4.30 构造一个函数$f(N)$，满足$f(N)\sim\sum_{k\geqslant 0}\frac{k!}{N^k}$。

4.3 渐近展开式的操作

我们使用渐近级数，尤其是有限展开式，不仅仅是因为它们提供了一种简洁的方法，在对精度的某种控制下表示逼近的结果，而且还因为，它们相对比较容易操作，它们能使我们进行复杂的操作的同时却仍然使用简单的表达式。这是因为我们很少坚持使用正在研究的量的完全渐近级数，而是只对展开式中的前几项进行操作，这就使我们在进行每一次计算的时候，都可以抛弃那些不太重要的项。因此在实际中，我们可用只包含几项的规范形式，来精确地描述大量的函数。

大O-符号的基本性质。根据定义很容易证明大量的基本恒等式，这些关系使我们能够更

169 ~ 170

容易地去处理涉及大O-符号的表达式。这些关系都是一些直观的规则，其中的一些我们已经在默默地使用了。在这些恒等式中，我们用箭头来表示：任何在箭头左端的表达式都可以用箭头右端的表达式来简化：

$$f(N) \to O(f(N))$$
$$cO(f(N)) \to O(f(N))$$
$$O(cf(N)) \to O(f(N))$$
$$f(N) - g(N) = O(h(N)) \to f(N) = g(N) + O(h(N))$$
$$O(f(N))O(g(N)) \to O(f(N)g(N))$$
$$O(f(N)) + O(g(N)) \to O(g(N)) \qquad 若 \quad f(N) = O(g(N))$$

严格地说，使用方程左端的大O-符号是不严格的。我们的确经常把式子写成诸如$N^2 + N + O(1) = N^2 + O(N) = O(N^2)$的形式，从而避免使用像上述箭头符号那样形式的操作所带来的不便，但是我们绝不能使用方程$N = O(N^2)$和$N^2 = O(N^2)$，来得到一个荒谬的结论$N = N^2$。

对任何一个特定的函数，大O-记号实际上给出了多种可能的描述方法，但习惯的做法是利用上述规则，写出一个不带常数的简单的规范式。我们可以写成$O(N^2)$，而从不去写成$NO(N)$ 或$2O(N^2)$ 或$O(2N^2)$ 等，尽管这些式子都是等价的。习惯上我们把一个未指定的常数写成$O(1)$，而不是其他形式，如$O(3)$等。而且我们写$O(\log N)$时也不指定对数的底数（当底数为常数时），由于大O-符号的原因，指定底数相当于给出了一个不相关的常数。

通常我们能用一种简易的方式，通过一种或几种基本操作来减少对渐近展开式的操作，我们将依次进行考虑。在我们所讨论的例子中，通常考虑带有一项、两项或三项的级数（大O-项除外）。当然，我们所用的方法也适用于较长的级数。

习题4.31 当$N \to \infty$时，证明下列式子成立或者不成立：

171

$$e^N = O(N^2), \quad e^N = O(2^N), \quad 2^{-N} = O\left(\frac{1}{N^{10}}\right), \qquad N^{\ln N} = O(e^{(\ln N)^2})$$

化简。在进行渐近分析时，我们必须意识到一个主要原理：一个渐近级数实际上只精确到它的O-项，因此任何较小的项（在渐近的意义上）都完全可以被舍弃。例如，表达式$\ln N + O(1)$ 与表达式$\ln N + \gamma + O(1)$在数学上是等价的，但前者更为简单。

替换。最简单和最常见的渐近级数是通过适当地选择变量的值，并将其代入泰勒级数的展开式（如表4-2中的泰勒展开式）中、或代入其他渐近级数中而导出来的。例如，通过在几何级数

$$\frac{1}{1-x} = 1 + x + x^2 + O(x^3) \qquad (x \to 0)$$

中取$x = -1/N$，我们可求得

$$\frac{1}{N+1} = \frac{1}{N} - \frac{1}{N^2} + O\left(\frac{1}{N^3}\right) \qquad (N \to \infty)$$

类似地，

$$e^{1/N} = 1 + \frac{1}{N} + \frac{1}{2N^2} + \frac{1}{6N^3} + \cdots + \frac{1}{k!N^k} + O\left(\frac{1}{N^{k+1}}\right)$$

习题4.32 给出关于$e^{1/(N+1)}$的一个渐近展开式，准确到$O(N^{-3})$。

分解。在许多情况下，函数的“逼近”值显然有待检验，因此我们值得花费一定的精力来重写函数，使之能按照相对误差或绝对误差的形式来明确地反映逼近程度。例如对于较大的N，函数$1/(N^2+N)$显然非常接近于$1/N^2$，我们可以很容易地通过重写函数将这一事实明确地表示出来

$$\begin{aligned}\frac{1}{N^2+N}&=\frac{1}{N^2}\frac{1}{1+1/N}\\&=\frac{1}{N^2}\left(1+\frac{1}{N}+O\left(\frac{1}{N^2}\right)\right)\\&=\frac{1}{N^2}+\frac{1}{N^3}+O\left(\frac{1}{N^4}\right)\end{aligned}$$

如果我们遇到的是一个复杂的函数，其逼近值不能立即看出来的话，那么我们可能需要一个简短的试算误差的过程。 172

乘法。将两个渐近级数相乘，只不过是将它们逐项相乘，然后再合并各个项。例如

$$\begin{aligned}(H_N)^2&=\left(\ln N+\gamma+O\left(\frac{1}{N}\right)\right)\left(\ln N+\gamma+O\left(\frac{1}{N}\right)\right)\\&=\left((\ln N)^2+\gamma\ln N+O\left(\frac{\log N}{N}\right)\right)\\&\quad+\left(\gamma\ln N+\gamma^2+O\left(\frac{1}{N}\right)\right)\\&\quad+\left(O\left(\frac{\log N}{N}\right)+O\left(\frac{1}{N}\right)+O\left(\frac{1}{N^2}\right)\right)\\&=(\ln N)^2+2\gamma\ln N+\gamma^2+O\left(\frac{\log N}{N}\right)\end{aligned}$$

在这个例子中，乘积的绝对渐近“精度”要比两个因子的绝对渐近精度低——其结果只准确到$O(\log N/N)$。这种情况是正常的，我们通常在开始推导时，所取的渐近展开式中的项数应该比结果中所需要的项数多一些。我们常常采取一个两步的过程：先进行计算，如果结果没有达到所需要的精度时，则需精确地表示原来的两个因子，然后再重新计算。

习题4.33 计算$(H_N)^2$，准确到$O(1/N)$，然后再准确到$o(1/N)$。

作为另一个例子，我们来估计N的阶乘的平方：

$$\begin{aligned}N!N!&=\left(\sqrt{2\pi N}\left(\frac{N}{e}\right)^N\left(1+O\left(\frac{1}{N}\right)\right)\right)^2\\&=2\pi N\left(\frac{N}{e}\right)^{2N}\left(1+O\left(\frac{1}{N}\right)\right)\end{aligned}$$

因为

$$\left(1+O\left(\frac{1}{N}\right)\right)^2=1+2O\left(\frac{1}{N}\right)+O\left(\frac{1}{N^2}\right)=1+O\left(\frac{1}{N}\right)$$

除法。为了计算两个渐近级数的商，我们通常是对分母进行分解，并将其重新写成$1/(1-x)$的形式，其中x是某种符号的表达式且x趋于0，然后将其展开成一个几何级数，再与分子相

173 乘。例如，为了计算tan$x$的一个渐近展开式，我们可以将关于sin$x$的级数除以关于cos$x$的级数，过程如下：

$$\begin{aligned}\tan x &= \frac{\sin x}{\cos x} = \frac{x - x^3/6 + O(x^5)}{1 - x^2/2 + O(x^4)} \\ &= (x - x^3/6 + O(x^5))\frac{1}{1 - x^2/2 + O(x^4)} \\ &= (x - x^3/6 + O(x^5))(1 + x^2/2 + O(x^4)) \\ &= x + x^3/3 + O(x^5)\end{aligned}$$

习题4.34 导出一个关于cot$x$的渐近展开式，准确到$O(x^4)$。

习题4.35 导出一个关于$x/(e^x - 1)$的渐近展开式，准确到$O(x^5)$。

作为另外一个例子，我们考虑关于中央二项式系数$\binom{2N}{N}$的逼近。将级数（根据上面的结果）

$$N!N! = 2\pi N\left(\frac{N}{e}\right)^{2N}\left(1 + O\left(\frac{1}{N}\right)\right)$$

除以级数

$$(2N)! = 2\sqrt{\pi N}\left(\frac{2N}{e}\right)^{2N}\left(1 + O\left(\frac{1}{N}\right)\right)$$

其结果为

$$\binom{2N}{N} = \frac{2^{2N}}{\sqrt{\pi N}}\left(1 + O\left(\frac{1}{N}\right)\right)$$

将此结果乘以$1/(N+1) = 1/N - 1/N^2 + O(1/N^3)$，将得到表4-6中关于Catalan数的逼近。

指数/对数。在涉及幂或乘积的渐近性时，把$f(x)$写成$\exp\{\ln(f(x))\}$常常是简便地进行分析的开始。例如，利用Stirling近似推导Catalan数的渐近性的另一个方法可以写成

$$\begin{aligned}\frac{1}{N+1}\binom{2N}{N} &= \exp\{\ln((2N)!) - 2\ln N! - \ln(N+1)\} \\ &= \exp\left\{\left(2N + \frac{1}{2}\right)\ln(2N) - 2N + \ln\sqrt{2\pi} + O\left(\frac{1}{N}\right)\right. \\ &\quad \left. - 2\left(N + \frac{1}{2}\right)\ln N + 2N - 2\ln\sqrt{2\pi} + O\left(\frac{1}{N}\right) - \ln N + O\left(\frac{1}{N}\right)\right\} \\ &= \exp\left\{\left(2N + \frac{1}{2}\right)\ln 2 - \frac{3}{2}\ln N - \ln\sqrt{2\pi} + O\left(\frac{1}{N}\right)\right\}\end{aligned}$$

174

它仍然等价于表4-6中关于Catalan数的逼近。

习题4.36 继续对Catalan数进行展开，使其准确到$O(N^{-4})$。

习题4.37 求一个关于$\binom{3N}{N}/(N+1)$的渐近展开式。

习题4.38 求一个关于$(3N)!/(N!)^3$的渐近展开式。

另一个关于指数/对数操作的标准例子是如下关于e的逼近：

$$\begin{aligned}\left(1+\frac{1}{N}\right)^N &= \exp\left\{N\ln\left(1+\frac{1}{N}\right)\right\}\\ &= \exp\left\{N\left(\frac{1}{N}+O\left(\frac{1}{N^2}\right)\right)\right\}\\ &= \exp\left\{1+O\left(\frac{1}{N}\right)\right\}\\ &= e+O\left(\frac{1}{N}\right)\end{aligned}$$

推导过程中的最后一步将在后面给出证明。此外，通过考虑当N（比如说）为一百万或十亿时，如何计算这个展开式的值，我们就能意识到渐近分析的实用性了。

习题4.39 $\left(1-\frac{\lambda}{N}\right)^N$ 的近似值是多少？

习题4.40 给出 $\left(1-\frac{\ln N}{N}\right)^N$ 的一个三项渐近展开式。

习题4.41 假设银行存单的利息是“按天增加的”，也就是说，对365天而言，每天都将利息的1/365加入到存单上。对于一个\$10 000的存单，如果按天增加的利率为10%的话，那么与按年付息所应付的\$1000相比，一年后银行付的利息是多少？

合成。通过代入指数展开式，显然有

$$e^{1/N}=1+\frac{1}{N}+O\left(\frac{1}{N^2}\right)$$

但此式与

$$e^{O(1/N)}=1+O\left(\frac{1}{N}\right)$$

175

稍微有些不同，后者是由刚刚给出的两个推导所采用的。在这个例子中，将$O(1/N)$代入到指数展开式中仍然是有效的：

$$\begin{aligned}e^{O(1/N)} &= 1+O\left(\frac{1}{N}\right)+O\left(\left(O\left(\frac{1}{N}\right)\right)^2\right)\\ &= 1+O\left(\frac{1}{N}\right)\end{aligned}$$

由于我们经常处理一些相对较短的展开式，所以也经常对那些看起来相当复杂的函数设计一些简单的渐近估计，设计方法只是简单地进行幂级数代入。[7]中给出了控制这种操作的具体条件。

习题4.42 在不丢失渐近精度的条件下，化简渐近展开式$\exp\{1+1/N+O(1/N^2)\}$。

习题4.43 求一个关于$\ln(\sin((N!)^{-1}))$的渐近估计，准确到$O(1/N^2)$。

习题4.44 证明：$\sin(\tan(1/N))\sim 1/N$和$\tan(\sin(1/N))\sim 1/N$，然后求$\sin(\tan(1/N))-\tan(\sin(1/N))$增长的阶数。

习题4.45 求一个关于H_{T_N}的渐近估计，准确到$O(1/N)$，其中T_N是第N个Catalan数。

逆。假设有渐近展开式

$$y = x + c_2x^2 + c_3x^3 + O(x^4)$$

我们省略了常数项和一次项的系数以便于简化计算。该展开式可以通过一个自举的过程变成一个用y来表示x的方程，该过程类似于第2章中用于估计递归逼近解的过程。首先，我们必然有

$$x = O(y)$$

因为当$x \to 0$时，$x/y = x/(x + c_2x^2 + O(x^3))$是有界的。将其代入原始的展开式中，就意味着

$$y = x + O(y^2) \quad 或 \quad x = y + O(y^2)$$

再将此式代入原始展开式中，我们有

176

$$y = x + c_2(y + O(y^2))^2 + O(y^3) \quad 或 \quad x = y - c_2y^2 + O(y^3)$$

我们每一次将x回代到原始展开式中时，都能得到另外一项。继续回代，我们有

$$y = x + c_2(y - c_2y^2 + O(y^3))^2 + c_3(y - c_2y^2 + O(y^3))^3 + O(y^4) \quad 或$$

$$x = y - c_2y^2 + (2c_2^2 - c_3)y^3 + O(y^4)$$

习题4.46 设a_n定义为方程

$$n = a_ne^{a_n} \quad (n > 1)$$

的唯一正根。求关于a_n的一个渐近估计，准确到$O(1/(\log n)^3)$。

习题4.47 求幂级数

$$y = c_0 + c_1x + c_2x^2 + c_3x^3 + O(x^4)$$

的逆（提示：取$z = (y - c_0)/c_1$）。

4.4 有限和的渐近逼近

常常能够把一个量表示成有限和，因此，我们需要能够准确地估计这个有限和的值。在第2章已经看到，某些和可以被准确地求出值来，不过在更多的情况下准确值却不是容易得到的。再有，可能出现这种情况：我们所知道的只是那些被求和的量本身的估计值。

在[6]中，De Bruijn考虑了这种问题的一些细节。他概括了经常发生的一些不同的情形，主要考虑的常常是和中的项在值上变化剧烈的情况。在这一节我们扼要讨论一些初等的例子，不过重点侧重于欧拉-麦克劳林公式，它是用积分估计求和的基本工具。我们指出欧拉-麦克劳林公式如何给出调和数以及阶乘（Stirling公式）的渐近展开。

在本章的其余部分，我们考虑欧拉-麦克劳林求和的一些应用，特别是重点放在那些涉及以二项式系数为例的经典“二元”函数的被加项上。我们将会看到，这些应用在求和范围的不同部分所用的求和估计是不同的，但是，它们最终还是依赖于通过欧拉-麦克劳林求和的方法使用积分来对一个和式进行估计。在这方面更多的细节和相关的课题可以在[2]、[3]、[6]、[12]和[19]中找到。

177

确定尾部的界限。当有限和中的项急速递减的时候，通过利用无限和逼近该有限和并算出其无限尾部的大小的界，可以得到这个有限和的渐近估计。下面的例子是求“错位”排列(见第6章)的个数的经典例子，它解释了这样一种观点

$$N!\sum_{0\le k\le N}\frac{(-1)^k}{k!}=N!e^{-1}-R_N \qquad 其中 \qquad R_N=N!\sum_{k>N}\frac{(-1)^k}{k!}$$

现在我们可以通过利用尾部各项的界来确定尾部R_N的界

$$|R_N|<\frac{1}{N+1}+\frac{1}{(N+1)^2}+\frac{1}{(N+1)^3}+\cdots=\frac{1}{N}$$

因此这个和为$N!e^{-1}+O(1/N)$。在这种情况下，收敛是如此的快，以至于能够证明其值总是等于$N!e^{-1}$经舍入得到的最近的整数。

所涉及的无限和收敛到一个常数，但是对于这个常数可能不存在显式表达式。不过，快速收敛通常意味着以比较大的准确度计算它的值是容易的。下面的例子与研究trie树（见第7章）过程中出现的和有关:

$$\sum_{1\le k\le N}\frac{1}{2^k-1}=\sum_{k\ge 1}\frac{1}{2^k-1}-R_N \quad 其中 \quad R_N=\sum_{k>N}\frac{1}{2^k-1}$$

此时我们有

$$0<R_N<\sum_{k>N}\frac{1}{2^{k-1}}=\frac{1}{2^{N-1}}$$

因此，常数$1+1/3+1/7+1/15+\cdots=1.6066\ldots$是该有限和的极好的近似。将这个常数的值计算到任意合理的准确度并不困难。

尾部的使用。当有限和中的项快速递增的时候，其最后一项常常足以给出整个和的一个好的渐近估计。例如，

$$\sum_{0\le k\le N}k!=N!\left(1+\frac{1}{N}+\sum_{0\le k\le N-2}\frac{k!}{N!}\right)=N!\left(1+O\left(\frac{1}{N}\right)\right)$$

后面的等式的得出是因为在这个和中存在$N-1$项，其中的每一项都小于$1/(N(N-1))$。 178

习题4.48 给出$\sum_{1\le k\le N}1/(k^2H_k)$的一个渐近估计。

习题4.49 给出$\sum_{0\le k\le N}1/F_k$的一个渐近估计。

习题4.50 给出$\sum_{0\le k\le N}2^k/(2^k+1)$的一个渐近估计。

习题4.51 给出$\sum_{0\le k\le N}2^{k^2}$的一个渐近估计。

用积分逼近和。更一般地，我们期望应该能够用积分来估计一个和的值，从而能够利用已知积分的广泛知识。

当我们使用

$$\int_a^b f(x)\mathrm{d}x \text{ 来估计 } \sum_{a\le k<b}f(k)$$

的时候，误差的幅度有多大？这个问题的答案依赖于函数$f(x)$有多么光滑。本质上，在a和b之间$b-a$的每个单位区间中我们都用$f(k)$来估计$f(x)$。令

$$\delta_k=\max_{k\le x<k+1}|f(x)-f(k)|$$

表示每个区间中的最大误差，我们可以得出对总误差的一个粗略的近似:

$$\sum_{a\le k<b}f(k)=\int_a^b f(x)\mathrm{d}x+\Delta, \qquad \left(|\Delta|<\sum_{a\le k<b}\delta_k\right)$$

如果这个函数在整个区间$[a, b]$是单调递增或递减的，那么，将误差项叠缩就得到$|\Delta| \leqslant |f(a) - f(b)|$。例如，对于调和数，该方法给出估计

$$H_N = \sum_{1 \leqslant k \leqslant N} \frac{1}{k} = \int_1^N \frac{1}{x}\mathrm{d}x + \Delta = \ln N + \Delta$$

其中$|\Delta| \leqslant 1 - 1/N$，这是$H_N \sim \ln N$的一个简单的证明；而对于$N!$，这种方法则得到估计

$$\ln N! = \sum_{1 \leqslant k \leqslant N} \ln k = \int_1^N \ln x\,\mathrm{d}x + \Delta = N\ln N - N + 1 + \Delta$$

179

其中，对于$\ln N!$有$|\Delta| \leqslant \ln N$，这是$\ln N! \sim N\ln N - N$的一个简单的证明。这些估计的准确度依赖于在逼近误差时的仔细程度。

误差项的更精确的估计取决于函数$f(x)$的导数。由此导致使用欧拉-麦克劳林求和公式所得到的渐近级数，它是渐近分析中最为强大的工具之一。

4.5 欧拉-麦克劳林求和

在算法分析中，用积分来逼近和时有两种不同的方法。在第一种情况下，我们有一个定义在固定区间上的函数，并且我们是在沿该区间一组递增的点上抽取函数值来求和的，随着步长越来越小，和与积分的差值将收敛到零，这类似于古典的黎曼积分。在第二种情况下，我们有一个固定的函数和一个固定的离散步长，随着积分区间变得越来越大，和与积分的差值将收敛到一个常数。我们将分别考虑这两种情况，尽管它们都体现了同一个基本方法：一种要追溯到18世纪的方法。

欧拉-麦克劳林求和公式的一般形式。这个方法是以分部积分为基础的，并且要涉及到伯努利数（和伯努利多项式）——这在3.13节中已描述过了。我们从下面的公式开始

$$\int_0^1 g(x)\mathrm{d}x = \left(x - \frac{1}{2}\right)g(x)\Big|_0^1 - \int_0^1 \left(x - \frac{1}{2}\right)g'(x)\mathrm{d}x$$

这个式子是通过对$g(x)$分部积分，并“聪明地”选择积分常数$x - \frac{1}{2} = B_1(x)$而得到的。在这个式子中取$g(x) = f(x + k)$，我们将得到

$$\int_k^{k+1} f(x)\mathrm{d}x = \frac{f(k+1) + f(k)}{2} - \int_k^{k+1}\left(\{x\} - \frac{1}{2}\right)f'(x)\mathrm{d}x$$

与以往一样，其中$\{x\} \equiv x - \lfloor x \rfloor$表示$x$的小数部分。取所有大于等于$a$且小于$b$的$k$，并将这些式子相加得

$$\int_a^b f(x)\mathrm{d}x = \sum_{a \leqslant k < b} f(k) - \frac{f(a) + f(b)}{2} - \int_a^b \left(\{x\} - \frac{1}{2}\right)f'(x)\mathrm{d}x$$

180

这是因为除了a和b以外，$f(k)$对于k的每个值都出现在了两个式子中。于是重新整理这些项，我们将得到和与其对应的积分之间的一个精确的关系：

$$\sum_{a \leqslant k < b} f(k) = \int_a^b f(x)\mathrm{d}x + \frac{f(a) + f(b)}{2} + \int_a^b \left(\{x\} - \frac{1}{2}\right)f'(x)\mathrm{d}x$$

为了得知这个逼近到底有多好，需要能够对尾部那个积分找到一个界。我们本来也可以像前一节末尾所做的那样，通过找到一个绝对的界来达此目的，但结果表明，我们可以迭代这个

过程，通常只剩下一个非常小的误差项，这是因为$f(x)$的导数会变得越来越小（作为N的函数），和/或因为同样也包含在积分中的$\{x\}$的多项式变得越来越小。

定理4.2（欧拉-麦克劳林求和公式，第一种形式） 设$f(x)$是定义在区间$[a, b]$上的函数，其中a和b为整数，并假定当$1 \le i \le 2m$时，导数$f^{(i)}(x)$存在且连续，其中m是一个固定的常数，则

$$\sum_{a \le k < b} f(k) = \int_a^b f(x)\mathrm{d}x + \frac{f(a)+f(b)}{2} + \sum_{1 \le i \le m} \frac{B_{2i}}{(2i)!} f^{(2i-1)}(x)\Big|_a^b + R_m$$

其中B_{2i}是伯努利数，R_m是余项，且满足

$$|R_m| \le \frac{|B_{2m}|}{(2m)!}\int_a^b |f^{(2m)}(x)|\,\mathrm{d}x < \frac{4}{(2\pi)^{2m}}\int_a^b |f^{(2m)}(x)|\,\mathrm{d}x$$

证明 我们利用分部积分和伯努利多项式的基本性质来继续进行上面的讨论。对任意一个在$[0, 1]$中可微的函数$g(x)$以及任意的$i > 0$，我们可以对$g(x)B'_{i+1}(x)$进行分部积分得

$$\int_0^1 g(x)B'_{i+1}(x)\mathrm{d}x = B_{i+1}(x)g(x)\Big|_0^1 - \int_0^1 g'(x)B_{i+1}(x)\mathrm{d}x$$

现在由3.13节我们知道$B'_{i+1}(x) = (i+1)B_i(x)$，所以两边除以$(i+1)!$后，我们得到一个递推关系：

$$\int_0^1 g(x)\frac{B_i(x)}{i!}\mathrm{d}x = \frac{B_{i+1}(x)}{(i+1)!}g(x)\Big|_0^1 - \int_0^1 g'(x)\frac{B_{i+1}(x)}{(i+1)!}\mathrm{d}x$$

181

再从$i = 0$起开始迭代，将形式地给出

$$\int_0^1 g(x)\mathrm{d}x = \frac{B_1(x)}{1!}g(x)\Big|_0^1 - \frac{B_2(x)}{2!}g'(x)\Big|_0^1 + \frac{B_3(x)}{3!}g''(x)\Big|_0^1 - \cdots$$

其中对于无限可导的函数而言，该展开式可以展开到任意多个项。更确切地说经过m步后，我们停止迭代，并注意$B_1(x) = \left(x - \frac{1}{2}\right)$，以及对$i > 1$有$B_i(0) = B_i(1) = B_i$，且当$i$为大于1的奇数时，$B_i = 0$（见习题3.86和3.87），这样我们就得到公式

$$\int_0^1 g(x)\mathrm{d}x = \frac{g(0)+g(1)}{2} - \sum_{1 \le i \le m}\frac{B_{2i}}{(2i)!}g^{2i-1}(x)\Big|_0^1 - \int_0^1 g^{2m}(x)\frac{B_{2m}(x)}{(2m)!}\mathrm{d}x$$

将$g(x) = f(x+k)$代入，并对$a \le k < b$求和，就可简化成定理所述的结果，用同样的方法可以得到余项为

$$|R_m| = \int_a^b \left|\frac{B_{2m}(\{x\})}{(2m)!}f^{(2m)}(x)\right|\mathrm{d}x$$

再根据伯努利数的渐近性质，就可以得出定理中所述的关于余项的界。（更详细的细节可参见De Bruijn[6]或Graham、Knuth、Patashnik[12]。） ■

例如取$f(x) = e^x$，则公式的左端为$(e^b - e^a)/(e-1)$，右端所有导数都相同，因此令两端同时除以$e^b - e^a$并使m增加，我们就可以证明$1/(e-1) = \sum_k B_k/k!$。

因为伯努利数可以增长到相当大的数，所以定理中的式子常常是一个发散的渐近级数，因而该式子经常用于较小的m值。在定理4.2的一般性应用中，取前几个伯努利数$B_0 = 1$、$B_1 = -1/2$、$B_2 = 1/6$、$B_3 = 0$和$B_4 = -1/30$通常就足够了。我们可以把公式写成更简单的形式

$$\sum_{a<k<b} f(k)=\int_a^b f(x)\mathrm{d}x+\frac{1}{2}f(a)+\frac{1}{2}f(b)+\frac{1}{12}f'(x)\Big|_a^b-\frac{1}{720}f'''(x)\Big|_a^b+\cdots$$

182 但我们要清楚必须检查定理的条件以及误差的界，以保证应用是合法的。

取$f(x)=x^t$，则对足够大的m，导数项和余项将都会消失，从而可以证实：在表示整数幂的和中，伯努利数可以用来作为系数。我们有

$$\sum_{1<k<N} k=\frac{N^2}{2}+\frac{N}{2}=\frac{N(N+1)}{2}$$

$$\sum_{1<k<N} k^2=\frac{N^3}{3}+\frac{N^2}{2}+\frac{N}{6}=\frac{N(N+1)(2N+1)}{6}$$

等等，这与3.13节中的结果是完全一样的。

习题4.52 当把$\sum_{1<k<N}k^t$表示成N的幂的和时，用欧拉-麦克劳林求和公式确定其系数（见3.13节）。

推论 如果h为一无限可微函数，则有

$$\sum_{0<k<N} h(k/N)\sim N\int_0^1 h(x)\mathrm{d}x+\frac{1}{2}(h(0)+h(1))+\sum_{i\geq 1}\frac{B_{2i}}{(2i)!}\frac{1}{N^{2i-1}}h^{(2i-1)}(x)\Big|_0^1$$

证明 对$f(x)=h(x/N)$应用定理4.2。 ■

把上式两端同时除以N，将得出一个相应于h的黎曼和。换句话说，该推论是

$$\lim_{N\to\infty}\frac{1}{N}\sum_{0<k<N} h(k/N)=\int_0^1 h(x)\mathrm{d}x$$

的一个改进。

欧拉-麦克劳林求和对于求与这种类型相关的和的渐近展开将是非常有效的，例如

$$\sum_{0<k<N} h(k^2/N)$$

我们很快就会看到这样的应用。

习题4.53 求出一个关于

$$\sum_{0<k<N}\frac{1}{1+k/N}$$

183 的渐近展开式。

习题4.54 证明：

$$\sum_{0<k<N}\frac{1}{1+k^2/N^2}=\frac{\pi N}{4}+\frac{3}{4}-\frac{1}{24N}+O\left(\frac{1}{N^2}\right)$$

正如我们所说过的，当求和/积分的区间增大且步长固定时，定理4.2将不能给出足够准确的估计。例如，如果我们试图

$$\text{用}\int_a^b f(x)\mathrm{d}x\text{ 来估计 }H_k=\sum_{1<k<N}\frac{1}{k}$$

时，将遇到一个困难，这是因为当$N\to\infty$时，和与积分之间的差将趋向于一个未知的常数。下面我们将转向处理这一问题的欧拉-麦克劳林求和的形式。

欧拉-麦克劳林求和的离散形式。在定理4.2之前的讨论中，取$a=1$和$b=N$，我们将得到

$$\int_1^N f(x)\mathrm{d}x=\sum_{1\leqslant k\leqslant N} f(k)-\frac{1}{2}(f(1)+f(N))-\int_1^N\left(\{x\}-\frac{1}{2}\right)f'(x)\mathrm{d}x$$

如果当$N\to\infty$时，$f'(x)$趋于0的速度足够快，那么这个公式将把和与积分关联到一个常数因子上。特别地，如果量

$$C_f=\frac{1}{2}f(1)+\int_1^\infty\left(\{x\}-\frac{1}{2}\right)f'(x)\mathrm{d}x$$

存在，则它就定义了f的欧拉-麦克劳林常数，并且我们已经证明过

$$\lim_{N\to\infty}\left(\sum_{1\leqslant k\leqslant N} f(k)-\int_1^N f(x)\mathrm{d}x-\frac{1}{2}f(N)\right)=C_f$$

取$f(x)=1/x$，将给出关于调和数的一个逼近。在这种情况下，欧拉-麦克劳林常数就简称为*欧拉常数*（Euler's constant）：

$$\gamma=\frac{1}{2}-\int_1^\infty\left(\{x\}-\frac{1}{2}\right)\frac{\mathrm{d}x}{x^2}$$

因此，

$$H_N=\ln N+\gamma+o(1)$$

184

常数γ大约为0.57721…，且人们尚不知它是哪些其他基本常数的一个简单函数。

取$f(x)=\ln x$，将给出关于$\ln N!$的Stirling逼近。此时，欧拉-麦克劳林常数为

$$\int_1^\infty\left(\{x\}-\frac{1}{2}\right)\frac{\mathrm{d}x}{x}$$

实际上，这个常数确实是其他基本常数的一个简单函数：它等于$\ln\sqrt{2\pi}-1$。在4.7节中，我们将看到证明此结果的一种方法。值$\sigma=\sqrt{2\pi}$称为Stirling常数。该常数在算法分析和许多其他应用中经常出现。于是

$$\ln N!=N\ln N-N+\frac{1}{2}\ln N+\ln\sqrt{2\pi}+o(1)$$

当欧拉-麦克劳林常数已经严格地定义之后，继续进行分析以获得渐近级数中更多的项就变得相对容易了。总结上面的讨论，我们证明了

$$\sum_{1\leqslant k\leqslant N} f(k)=\int_1^N f(x)\mathrm{d}x+\frac{1}{2}f(N)+C_f-\int_N^\infty\left(\{x\}-\frac{1}{2}\right)f'(x)\mathrm{d}x$$

现在用与前面相同的方式，对剩余的积分反复进行分部积分，我们将得到一个包含伯努利数和高阶导数的展开式。这通常能导出一个完全的渐近展开式，因为一个最常见的事实是：具有光滑性质的函数的高阶导数在∞处将变得越来越小。

定理4.3（*欧拉-麦克劳林求和公式，第二种形式*）*设$f(x)$是定义在区间$[1,\infty)$上的函数，并假设导数$f^{(i)}(x)$存在且对$1\leqslant i\leqslant 2m$绝对可积，其中m是一个固定的常数。则有*

$$\sum_{1\leqslant k\leqslant N} f(k)=\int_1^N f(x)\mathrm{d}x+\frac{1}{2}f(N)+C_f+\sum_{1\leqslant k\leqslant m}\frac{B_{2k}}{(2k)!}f^{(2k-1)}(N)+R_m$$

其中C_f是与函数有关的一个常数，R_m是余项且满足

$$|R_m| = O\left(\int_N^{\infty} |f^{(2m)}(x)|\,\mathrm{d}x\right)$$

185 证明 用归纳法，对上面的论证进行扩展即可。细节可在[6]或[10]中找到。 ■

推论 调和数可以按N的降幂形式进行完全渐近展开：

$$H_N \sim \ln N + \gamma + \frac{1}{2N} - \frac{1}{12N^2} + \frac{1}{120N^4} - \cdots$$

证明 在定理4.3中取$f(x) = 1/x$，利用上面所讨论的常数γ，并注意余项与和中的最后一项是同阶的，对任意固定的m证明

$$H_N = \ln N + \gamma + \frac{1}{2N} - \sum_{1<k<M} \frac{B_{2k}}{2kN^{2k}} + O\left(\frac{1}{N^{2m}}\right)$$

由此式可得出推论所述的结果。 ■

推论 （Stirling公式）函数$\ln N!$和$N!$可以按N的降幂形式进行完全渐近展开：

$$\ln N! \sim \left(N + \frac{1}{2}\right)\ln N - N + \ln\sqrt{2\pi} + \frac{1}{12N} - \frac{1}{360N^3} + \cdots$$

和

$$N! \sim \sqrt{2\pi N}\left(\frac{N}{e}\right)^N \left(1 + \frac{1}{12N} + \frac{1}{288N^2} - \frac{139}{5140N^3} + \cdots\right)$$

证明 在定理4.3中取$f(x) = \ln x$，用和上面一样的方法进行论证以找到关于$\ln N!$的展开式。正如上面所提到过的，一阶导数不是绝对可积的，但欧拉-麦克劳林常数存在，所以定理4.3显然成立。关于$N!$的展开式可以通过取幂以及4.3节中所讨论过的基本操作而得到。 ■

推论 Catalan数可以按N的降幂形式进行完全渐近展开：

$$\frac{1}{N+1}\binom{2N}{N} \sim \frac{4^N}{N\sqrt{\pi N}}\left(1 - \frac{9}{8N} + \frac{145}{128N^2} - \cdots\right)$$

证明 通过对$(2N)!$和$N!$的渐近级数进行基本操作即可得出该结论。在4.3节所给出的例子
186 中可找到许多有关细节。 ■

欧拉-麦克劳林求和是一个一般性的工具，当满足函数必须是“光滑的”（在所期望的渐近级数中，有多少项，就有多少阶导数）限制、并且我们能够计算所涉及的积分时，这个工具是很有用的。我们刚才给出的关于阶乘、调和数以及Catalan数的渐近展开，在许多基本算法分析中都起着核心的作用，这种方法也出现在许多其他的应用中。

习题4.55 将γ计算到10位小数位。

习题4.56 证明：推广的（二阶）调和数可以渐近展开为

$$H_N^{(2)} = \sum_{1<k<N} \frac{1}{k^2} \sim \frac{\pi^2}{6} - \frac{1}{N} + \frac{1}{2N^2} - \frac{1}{6N^3} + \cdots$$

习题4.57 导出关于

$$H_N^{(3)} = \sum_{1 \le k \le N} \frac{1}{k^3}$$

的一个渐近展开式，准确到$O(1/N^3)$。

习题4.58 利用欧拉-麦克劳林求和，估计

$$\sum_{1 \le k \le N} \sqrt{k}, \qquad \sum_{1 \le k \le N} \frac{1}{\sqrt{k}}, \qquad \sum_{1 \le k \le N} \frac{1}{\sqrt[3]{k}}$$

准确到$O(1/N^2)$。

习题4.59 求出关于

$$\sum_{1 \le k \le N} \frac{(-1)^k}{k}, \qquad \sum_{1 \le k \le N} \frac{(-1)^k}{\sqrt{k}}$$

的完全渐近展开式。

4.6 二元渐近性

在逼近和的时候我们所面对的许多挑战性问题都与所谓的二元渐近性有联系，其中的被加数既依赖于求和的下标又依赖于描述渐近增长的“大小”参数。和我们以前多次的做法一样，假设这两个变元分别叫做k和N。现在，k和N的相应值以及它们的增长速度无疑对渐近估计起着重要的作用。关于这一点的一个简单例子是：注意对于固定的k而言，当$N \to \infty$ 时，函数$k^N/k!$以指数的量级增大，而对固定的N而言，当$k \to \infty$ 时，它却以指数的量级减小。 187

在求和的场合下，我们通常需要考虑所有小于N（或N的某个函数）的k值，因此，我们关注的是找到关于k的范围尽可能大的准确的渐近估计。求和之后将会消去变元k，这就使我们又回到了一元的情形。本节我们将详细考查几个在算法分析中起着核心作用的二元函数：Ramanujan函数和二项式系数。在接下来的几节中，我们将看到如何利用本节所产生的估计对涉及这些函数的和进行渐近逼近，以及这些估计是如何与算法分析中的应用相联系的。

Ramanujan分布。我们的第一个例子是关于一个分布的，该分布由Ramanujan（见[4]）首先进行了研究，后来由于这个分布在算法分析的许多应用中经常出现，Knuth[16]又对它进行了研究。我们将在第8章中看到，许多算法的性能特征都依赖于函数

$$Q(N) = \sum_{1 \le k \le N} \frac{N!}{(N-k)!N^k}$$

在概率论中，此函数也是为人们所熟知的生日函数：要找到具有相同生日的两个人（当一年中有N天时），$Q(N) + 1$则是我们所需要的期望试验次数。表4-7对较小的N和k给出了被加数的列表，而图4-1画出曲线图。在图中，分离的曲线是对每个N值给出来的，k的相继值由一些直线所连接。每条曲线的k轴的标度使得曲线能够布满整个图形。为了能估计出和的值，我们首先必须能够在当N增加时，对所有的k值精确地估计出被加数的值。

定理4.4 (Ramanujan Q-分布) 当$N \to \infty$ 时，下面的（相对）逼近对于$k = o(N^{2/3})$成立：

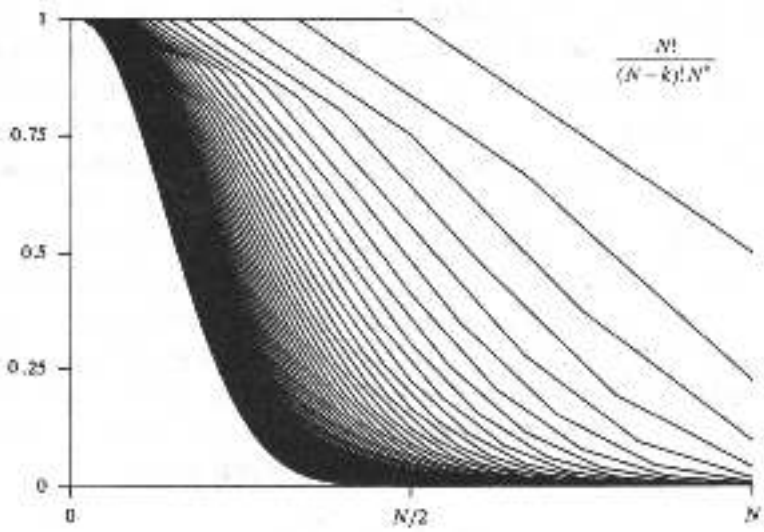
$$\frac{N!}{(N-k)!N^k} = e^{-k^2/(2N)}\left(1 + O\left(\frac{k}{N}\right) + O\left(\frac{k^3}{N^2}\right)\right)$$

此外对所有的k，下面的（绝对）逼近成立：

$$\frac{N!}{(N-k)!N^k}=e^{-k^2/(2N)}+O\left(\frac{1}{\sqrt{N}}\right)$$

表4-7 Ramanujan Q-分布 $\frac{N!}{(N-k)!N^k}$

$N\downarrow$ $k\rightarrow$	1	2	3	4	5	6	7	8	9	10
2	1	0.5000								
3	1	0.6667	0.2222							
4	1	0.7500	0.3750	0.0938						
5	1	0.8000	0.4800	0.1920	0.0384					
6	1	0.8333	0.5556	0.2778	0.0926	0.0154				
7	1	0.8571	0.6122	0.3499	0.1499	0.0428	0.0061			
8	1	0.8750	0.6563	0.4102	0.2051	0.0769	0.0192	0.0024		
9	1	0.8889	0.6914	0.4609	0.2561	0.1138	0.0379	0.0084	0.0009	
10	1	0.9000	0.7200	0.5040	0.3024	0.1512	0.0605	0.0181	0.0036	0.0004



188 ~ 189

图4-1 Ramanujan Q-分布，（k轴按N的比例标度）

证明 在4.3节中，我们曾利用“指数/对数”技巧证明过相对误差界，因此有

$$\begin{aligned}\frac{N!}{(N-k)!N^k}&=\frac{N(N-1)(N-2)\cdots(N-k+1)}{N^k}\\&=1\cdot\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\cdots\left(1-\frac{1}{N}\right)\\&=\exp\left\{\ln\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\cdots\left(1-\frac{1}{N}\right)\right\}\\&=\exp\left\{\sum_{1\le j<k}\ln\left(1-\frac{j}{N}\right)\right\}\end{aligned}$$

现在对$k = o(N)$，我们应用表4-2中的逼近

$$\ln(1+x) = x + O(x^2) \quad 取 \quad x = -j/N$$

并求和:

$$\frac{N!}{(N-k)!N^k} = \exp\left\{\sum_{1\le j<k}\left(-\frac{j}{N} + O\left(\frac{j^2}{N^2}\right)\right)\right\}$$

$$= \exp\left\{-\frac{k(k-1)}{2N} + O\left(\frac{k^3}{N^2}\right)\right\}$$

最后对于$k = o(N^{2/3})$，利用表4-2中的逼近$e^x = 1 + O(x)$，我们将立即得到定理所述的相对逼近。

结果中的两个O-项都应该保留，以便覆盖k值的范围。$O(k^3/N^2)$项本身是不充分的，因为例如当我们对结果的要求是$O(1/N)$时，对$k = O(1)$，它等于$O(1/N^2)$。$O(k/N)$项本身也是不充分的，例如当我们对结果的要求是$O(1/N^{1/5})$时，对$k = O(N^{3/5})$，它等于$O(1/N^{2/5})$。这说明对于二元渐近性，我们必须细心对待。

为找出绝对误差界，我们首先考虑k比较“小”时的情况，比如说$k < k_0$时，其中k_0是最靠近于$N^{3/5}$的整数。由于相对逼近肯定是成立的，因此我们有

$$\frac{N!}{(N-k)!N^k} = e^{-k^2/(2N)} + e^{-k^2/(2N)}O\left(\frac{k}{N}\right) + e^{-k^2/(2N)}O\left(\frac{k^3}{N^2}\right)$$

此时，第二项等于$O(1/\sqrt{N})$，因为我们可以将其重新写成：对所有的$x \ge 0$，$xe^{-x^2}O(1/\sqrt{N})$及 190
$xe^{-x^2} = O(1)$的形式。类似地，第三项具有$x^3e^{-x^2}O(1/\sqrt{N})$的形式，因此它等于$O(1/\sqrt{N})$，这是因为对所有的$x \ge 0$，$x^3e^{-x^2} = O(1)$。

下面我们考虑k比较“大”时的情况，或$k \ge k_0$时的情况。刚才给出的论证说明了

$$\frac{N!}{(N-k_0)!N^{k_0}} = e^{-\sqrt[3]{N}/2} + O\left(\frac{1}{\sqrt{N}}\right)$$

第一项是指数级小量，且其系数随着k的增加而减小，这就隐含了对于$k \ge k_0$，有

$$\frac{N!}{(N-k)!N^k} = O\left(\frac{1}{\sqrt{N}}\right)$$

但对于$k \ge k_0$，$\exp\{-k^2/(2N)\}$也是指数级小量，所以对$k \ge k_0$，定理中所述的绝对误差界成立。

上面的两段内容建立了对于所有的$k \ge 0$时的绝对误差界。在这个例子中，截断点$N^{3/5}$并不是特别重要：它只需要足够小，以使相对误差界对较小的k成立（稍小于$N^{2/3}$），且只需要足够大，以使各项对较大的k（稍大于$\sqrt{N}$）成为指数级小量。 ■

推论 对所有k和N，有 $\dfrac{N!}{(N-k)!N^k} \le e^{-k(k-1)/(2N)}$

证明 在上面的推导过程中，用不等式$\ln(1-x) \le -x$取代渐近估计即可。 ■

Ramanujan研究了应用到另一个函数的一组几乎完全相同的论证方法，即所谓的R-分布。表4-8给出了该函数关于较小的N和k值时的列表，其分布图由图4-2所示。关于这个函数，我们将对其渐近结果给出详细的阐述，后面将会清楚这样做的原因。

表4-8 Ramanujan R-分布 $\frac{N!N^k}{(N+k)!}$

$N\downarrow$ $k\rightarrow$	1	2	3	4	5	6	7	8	9
2	0.6667	0.3333							
3	0.7500	0.4500	0.2250						
4	0.8000	0.5333	0.3048	0.1524					
5	0.8333	0.5952	0.3720	0.2067	0.1033				
6	0.8571	0.6429	0.4286	0.2571	0.1403	0.0701			
7	0.8750	0.6806	0.4764	0.3032	0.1768	0.0952	0.0476		
8	0.8889	0.7111	0.5172	0.3448	0.2122	0.1212	0.0647	0.0323	
9	0.9000	0.7364	0.5523	0.3823	0.2458	0.1475	0.0830	0.0439	0.0220

定理4.5 (Ramanujan R-分布) 当$N\to\infty$时，下面的（相对）逼近对$k=o(N^{2/3})$成立：

$$\frac{N!N^k}{(N+k)!}=e^{-k^2/(2N)}\left(1+O\left(\frac{k}{N}\right)+O\left(\frac{k^3}{N^2}\right)\right)$$

此外对所有的k，下面的（绝对）逼近成立：

$$\frac{N!N^k}{(N+k)!}=e^{-k^2/(2N)}+O\left(\frac{1}{\sqrt{N}}\right)$$

证明 经过下面的第一步证明之后，其余部分与Q-分布的证明几乎完全一样：

$$\begin{aligned}\frac{N!N^k}{(N-k)!}&=\frac{N^k}{(N+k)(N+k-1)\cdots(N+1)}\\&=\frac{1}{\left(1+\frac{k}{N}\right)\left(1+\frac{k-1}{N}\right)\cdots\left(1+\frac{1}{N}\right)}\\&=\exp\left\{-\sum_{1\le j\le k}\ln\left(1+\frac{j}{N}\right)\right\}\\&=\exp\left\{\sum_{1\le j\le k}\left(-\frac{j}{N}+O\left(\frac{j^2}{N^2}\right)\right)\right\}\\&=\exp\left\{-\frac{k(k+1)}{2N}+O\left(\frac{k^3}{N^2}\right)\right\}\\&=e^{-k^2/(2N)}\left(1+O\left(\frac{k}{N}\right)+O\left(\frac{k^3}{N^2}\right)\right)\end{aligned}$$

■

191~192

与Q-分布的证明方法一样，绝对误差界成立。

推论 对所有的k和N（$k\le N$），都有 $\frac{N!N^k}{(N+k)!}\le e^{-k(k+1)/(4N)}$。

证明 用不等式$-\ln(1+x)\le -x/2$（对$0\le x\le 1$成立）取代上述推导过程中的渐近估计即可。■

习题4.60 证明：对所有的k和N，都有 $\frac{N!N^k}{(N+k)!}\ge e^{-k(k+1)/(2N)}$。

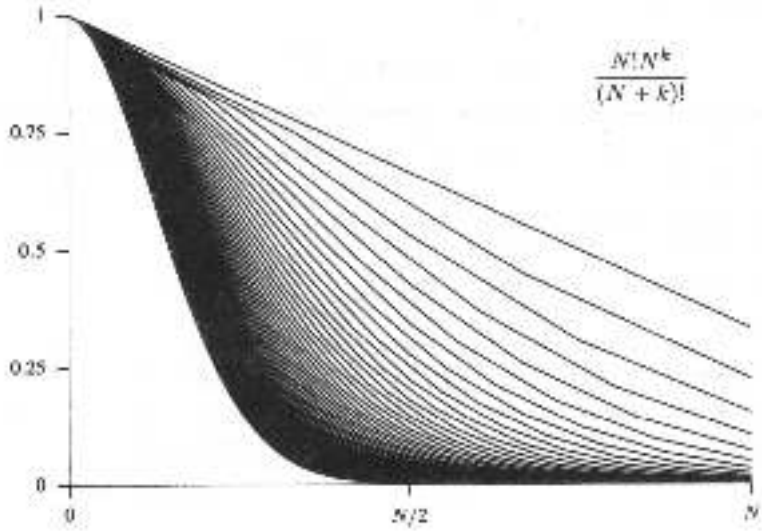


图4-2 Ramanujan R-分布，$2 \le N \le 60$（k轴按N的比例标度）

在本章的后一阶段，我们还将返回到Ramanujan分布上来，以处理几个应用问题。然而，在这样做之前，我们把注意力先转到一个二元分布上，即大家所熟知的二项分布（binomial distribution），该分布在算法分析中起着更加重要的作用。事实上，上面所给出的关于Ramanujan分布的渐近逼近的推导过程概括了逼近二项分布的基本方面。

习题4.61 利用关于$\ln N!$的Stirling公式，证明上面两个定理所给出的Ramanujan Q-分布和Ramanujan R-分布的相对误差界。

193

二项分布。给定N个随机的比特位，其中恰好有k个比特是0的概率就是我们所熟知的二项分布，也叫伯努利分布：

$$\frac{1}{2^N}\binom{N}{k} = \frac{1}{2^N}\frac{N!}{k!(N-k)!}$$

有兴趣的读者可以查阅Feller的经典教材[8]或任何标准的参考文献关于这一分布的性质及其在概率论中应用的有关信息。由于它在算法分析中经常出现，我们在这里将对它的许多重要性质加以概括。

表4-9给出了二项分布关于较小N值时的准确值和较大N值时的近似值。和以往一样，我们对此函数进行渐近分析的一个动机就是计算这样的一些值。$\binom{10000}{5000}/2^{9999}$ 的值大约为0.007 979，但人们不会这样去做：首先计算10 000！，然后将其除以5000！等等。事实上，人们对这个分布已经研究3个世纪了，寻找容易计算的逼近的动机在计算机出现之前就已经存在了。

习题4.62 编写一个程序，计算二项分布的精确值，准确到单精度浮点精度。

我们已经计算过中央二项式系数的近似值：

$$\binom{2N}{N} = \frac{2^{2N}}{\sqrt{\pi N}}\left(1+O\left(\frac{1}{N}\right)\right)$$

也就是说：在表4-9中，中间的那些项像$1/\sqrt{N}$那样递减。对于其他的k值，分布是如何表现的呢？图4-3给出了一个按比例绘制的分布图形，这个图形能给出我们一些暗示，下面我们将

给出一个精确的渐近分析。

表4-9 二项分布 $\binom{N}{k}/2^N$

$N\downarrow$ $k\rightarrow$	0	1	2	3	4	5	6	7	8	9
1	0.5000	0.5000								
2	0.2500	0.5000	0.2500							
3	0.1250	0.3750	0.3750	0.1250						
4	0.0625	0.2500	0.3750	0.2500	0.0625					
5	0.0312	0.1562	0.3125	0.3125	0.1562	0.0312				
6	0.0156	0.0938	0.2344	0.3125	0.2344	0.0938	0.0156			
7	0.0078	0.0547	0.1641	0.2734	0.2734	0.1641	0.0547	0.0078		
8	0.0039	0.0312	0.1094	0.2188	0.2734	0.2188	0.1094	0.0312	0.0039	
9	0.0020	0.0176	0.0703	0.1641	0.2461	0.2461	0.1641	0.0703	0.0176	0.0020

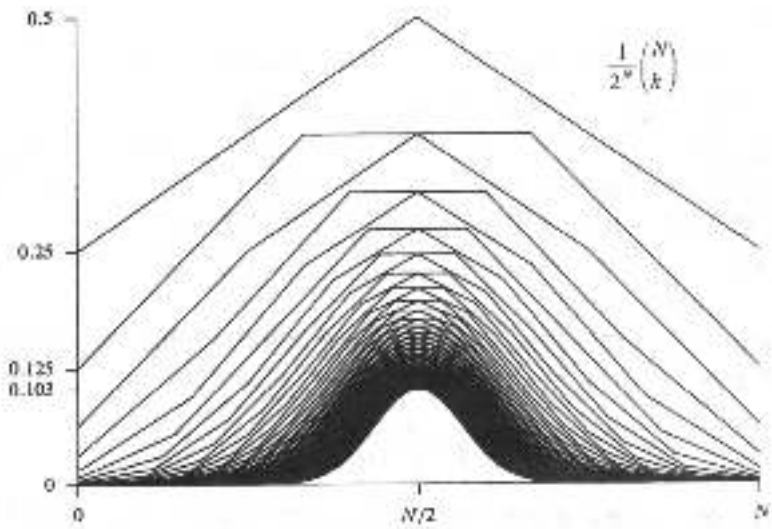


图4-3 二项分布，$2 \leqslant N \leqslant 60$（k轴按N的比例标度）

图中的极限曲线是我们所熟知的由正态概率密度函数 $e^{-x^2/2}/\sqrt{2\pi}$ 所描述的"钟形曲线"。曲线的顶部在 $\binom{N}{\lfloor N/2 \rfloor}/2^N \sim 1/\sqrt{\pi N/2}$ 之间，对 $N=60$，它大约为0.103。

我们这里的目的是利用我们已经得到的渐近工具来分析这个钟形曲线的性质。我们所给出的结果将是经典的，这些结果在概率与统计中起着非常重要的作用。我们对这些结果的兴趣不仅仅是因为它们在许多算法分析中的直接应用，还在于研究二项分布的渐近估计时所使用的技巧，这些技巧可以直接用在算法分析中出现的大量类似的问题之中。关于在概率统计背景下的正态逼近的论述，可参考，比如说，Feller[8]。

194 / 195

图4-3使得如下事实变得一目了然：曲线最重要的部分在中央附近——随着N值的增大，边缘附近的值变得可以忽略不计。这一点从我们开始时提出的概率模型中可以很直观地看出来：在一个随机比特流中，我们期望0和1的个数大致相等，随着比特流长度的增加，所有的比特位几乎全是0或全是1的概率将变得小到接近于零。下面，我们将对这种叙述给出更为精

确的量化。

正态逼近。由于二项分布中重要的值在中央附近，因此为方便起见，我们将重写这个分布，并考虑计算$\binom{2N}{N-k}/2^{2N}$。这个式子关于$k=0$是对称的，并随着$|k|$从0到N的递增而递减。这是处理任何分布时的重要一步：将那些最大的项放在开头部分，较小的项放在尾部，这更便于我们对尾部的定界和把注意力集中在主要的项上，尤其是在利用逼近来求和时更是如此，我们下面将要看到。

正如结果所表明的，我们已经看到证明经典正态逼近到二项分布所需要的基本方法。

定理4.6（正态逼近） 当$N\to\infty$时，下面的（相对）逼近对$k=o(N^{3/4})$成立：

$$\frac{1}{2^{2N}}\binom{2N}{N-k}=\frac{e^{-k^2/N}}{\sqrt{\pi N}}\left(1+O\left(\frac{1}{N}\right)+O\left(\frac{k^4}{N^3}\right)\right)$$

此外，下面的（绝对）逼近对所有的k成立：

$$\frac{1}{2^{2N}}\binom{2N}{N-k}=\frac{e^{-k^2/N}}{\sqrt{\pi N}}+O\left(\frac{1}{N^{3/2}}\right)$$

证明 如果把$\frac{1}{2^{2N}}\binom{2N}{N-k}$写成

$$\frac{1}{2^{2N}}\binom{2N}{N-k}=\frac{1}{2^{2N}}\frac{(2N)!}{N!N!}\frac{N!}{(N-k)!N^k}\frac{N!N^k}{(N+k)!}$$

我们可以看到：二项分布恰好是Ramanujan Q-分布和Ramanujan R-分布的乘积

$$\frac{1}{2^{2N}}\binom{2N}{N}=\frac{1}{\sqrt{\pi N}}\left(1+O\left(\frac{1}{N}\right)\right)$$

196

因此，我们可以通过简单地把这些量的渐近估计（由定理4.4、定理4.5以及定理4.3的第三个推论所给出）相乘，就能获得一个准确到$O(k^3/N^2)$的相对逼近，从而得到定理所述的结论。然而，在推导过程中多取一项，将会产生消项，从而给出更高的精度。与定理4.4和定理4.5的证明方法一样，我们有

$$\begin{aligned}
\frac{N!}{(N-k)!N^k}\frac{N!N^k}{(N+k)!}&=\exp\left\{\sum_{1\le j<k}\ln\left(1-\frac{j}{N}\right)-\sum_{1\le j\le k}\ln\left(1+\frac{j}{N}\right)\right\}\\
&=\exp\left\{\sum_{1\le j<k}\left(-\frac{j}{N}-\frac{j^2}{2N^2}+O\left(\frac{j^3}{N^3}\right)\right)-\sum_{1\le j\le k}\left(\frac{j}{N}-\frac{j^2}{2N^2}+O\left(\frac{j^3}{N^3}\right)\right)\right\}\\
&=\exp\left\{-\frac{k(k-1)}{2N}-\frac{k(k+1)}{2N}+O\left(\frac{k^2}{N^2}\right)+O\left(\frac{k^4}{N^3}\right)\right\}\\
&=\exp\left\{-\frac{k^2}{N}+O\left(\frac{k^2}{N^2}\right)+O\left(\frac{k^4}{N^3}\right)\right\}\\
&=e^{-k^2/N}\left(1+O\left(\frac{k^2}{N^2}\right)+O\left(\frac{k^4}{N^3}\right)\right)
\end{aligned}$$

改进的精度是由于和中消除j^2/N^2项得到的。$O(k^2/N^2)$项可以用$O(1/N)$来代替，因为如果$k<\sqrt{N}$，则k^2/N^2等于$O(1/N)$，如果$k>\sqrt{N}$，则k^2/N^2等于$O(k^4/N^3)$。

利用和定理4.4相同的证明过程，我们可以先对所有的$k>0$时建立起绝对误差界，再根据二项式系数的对称性可知，绝对误差界对所有的k都成立。 ■

这是对二项分布的正态逼近：函数$e^{-k^2/N}/\sqrt{\pi N}$是图4-3底部处著名的“钟形曲线”。大多数曲线将落在一个小的正负常数乘以$\sqrt{N}$的均值的范围之内，且在此范围之外尾部以指数的量级衰减。通常当我们操作正态逼近时，我们需要利用这两个事实。

推论 对所有的k和N，有$\frac{1}{2^{2N}}\binom{2N}{N-k}<e^{-k^2/(2N)}$。

197 **证明** 注意到$\binom{2N}{N}/2^{2N}<1$，并对定理4.4和定理4.5的推论中的界相乘即可证得。 ■

这种结果经常用来以下面的方式对分布的尾部进行定界。给定$\varepsilon>0$和$k>\sqrt{2N^{1+\varepsilon}}$，我们有

$$\frac{1}{2^{2N}}\binom{2N}{N-k}<e^{-(2N)^{\varepsilon}}$$

就是说，当k的增长稍稍快于$\sqrt{N}$的增长时，分布的尾部是指数级小量。

习题4.63 对$k=\sqrt{N}+O(1)$时的情形进行正态逼近，准确到$O(1/N^2)$。

习题4.64 作为N的一个函数，画出使二项概率大于0.001的最小的k。

泊松逼近。以稍微更一般的形式，二项分布给出了N次独立试验中成功k次的概率，其中每一次成功的概率为p：

$$\binom{N}{k}p^k(1-p)^{N-k}$$

我们将要在第8章中详细讨论，这个公式经常用“占有分布”的措辞来研究N个球在M个瓮中的分布。取$p=1/M$，则恰好有k个球落入某个特定的瓮（例如第一个瓮）中的概率是

$$\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k}$$

“球-瓮”模型是一个经典的模型（例如，参见[8]），并且事实上这个模型可以直接用于大量被广泛应用的基本算法中，我们将在第7和第8章中看到。

只要M是一个常数，则该分布就仍然可以用一个以N_p为中心的正态分布来逼近，正如习题4.67所证实、以及图4-4中（对$p=1/5$时）所阐明的那样。

M随着N变化时的情况特别值得关注。换句话说，我们取$p=1/M=\lambda/N$，其中λ是一个常数。这相当于进行N次试验，每次试验都有一个小的（λ/N）成功概率。于是我们期望成功的平均次数为λ。

198 相应于这种结论（在渐近极限中）的概率定律叫做*泊松律*。当我们要描述大量“主体”的整体行为时，这个定律是适用的，每个个体是“起作用的”、“成功的”、或是其他什么与众不同特征的概率是很小的。泊松定律的最早的一个应用（归功于19世纪的Von Bortkiewicz）就是描述普鲁士军队中被马匹踢死的骑兵的数量特征。从那以后，该定律就被广泛用来描述

各种各样的情形，从放射性衰减到对伦敦的空袭（参考[8]）等。在当前的环境下，泊松定律则用来描述球-盒模型中任意一个给定的盒（例如第一个盒）的情况。

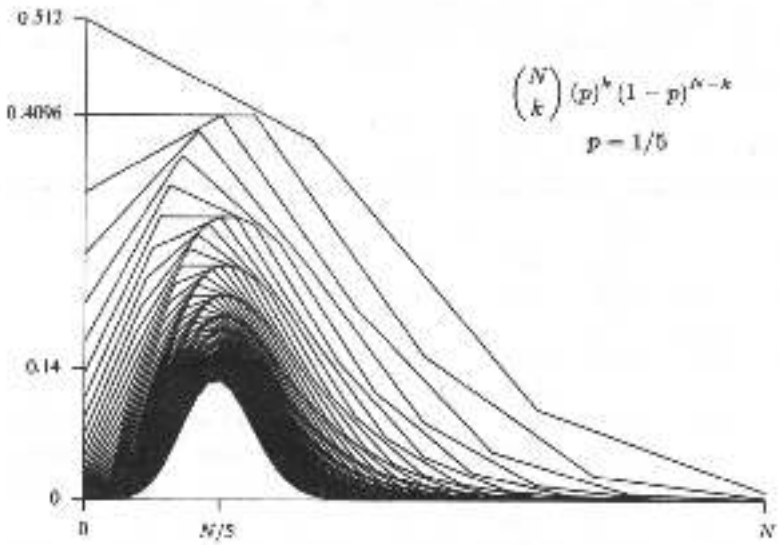


图4-4 二项分布，$3\leqslant N\leqslant 60$（k轴按N的比例标度）

定理4.7（泊松近似） 对固定的λ和所有的k，当$N\to\infty$时，有

$$\binom{N}{k}\left(\frac{\lambda}{N}\right)^k\left(1-\frac{\lambda}{N}\right)^{N-k}=\frac{\lambda^k e^{-\lambda}}{k!}+o(1)$$

特别地，当$k=O(N^{1/2})$时有

$$\binom{N}{k}\left(\frac{\lambda}{N}\right)^k\left(1-\frac{\lambda}{N}\right)^{N-k}=\frac{\lambda^k e^{-\lambda}}{k!}\left(1+O\left(\frac{1}{N}\right)+O\left(\frac{k}{N}\right)\right)$$

199

证明 把二项式系数写成另外一种形式：

$$\binom{N}{k}\left(\frac{\lambda}{N}\right)^k\left(1-\frac{\lambda}{N}\right)^{N-k}=\frac{\lambda^k}{k!}\frac{N!}{(N-k)!N^k}\left(1-\frac{\lambda}{N}\right)^{N-k}$$

我们看到，Ramanujan Q-分布在这里又一次出现了，把定理4.4的结果与

$$\begin{aligned}\left(1-\frac{\lambda}{N}\right)^{N-k}&=\exp\left\{(N-k)\ln\left(1-\frac{\lambda}{N}\right)\right\}\\&=\exp\left\{(N-k)\left(-\frac{\lambda}{N}+O\left(\frac{1}{N^2}\right)\right)\right\}\\&=e^{-\lambda}\left(1+O\left(\frac{1}{N}\right)+O\left(\frac{k}{N}\right)\right)\end{aligned}$$

结合，我们即可得到定理所述的相对误差界。

为了证明定理的第一部分（绝对误差界），我们通过对所有的$k>\sqrt{N}$（λ固定）的观察可知，泊松项与二项式的项均是指数级的小量，因而是$o(1)$。 ■

随着k的增大，$\lambda^k/k!$项也随之增大，直到$k=\lfloor\lambda\rfloor$时为止，然后$\lambda^k/k!$项迅速减小。和正态逼近一样，利用不等式而不是大O-记号进行上面的推导，则很容易导出尾部的界。图4-5显示了对较小的固定λ，当N增大时，该分布是如何演变的。表4-10给出了关于$\lambda=3$时的二项分布和泊松逼近。无论何时，只要概率相对较小，即使是对较小的N而言，泊松逼近对二项分布都是一个相当精确的估计。

表4-10 二项分布$\binom{N}{k}\left(\frac{3}{N}\right)^k\left(1-\frac{3}{N}\right)^{N-k}$，趋于泊松分布$3^ke^{-3}/k!$

$N\downarrow$ $k\rightarrow$	0	1	2	3	4	5	6	7	8	9
4	0.0039	0.0469	0.2109	0.4219	0.3164					
5	0.0102	0.0768	0.2304	0.3456	0.2592	0.0778				
6	0.0156	0.0938	0.2344	0.3125	0.2344	0.0938	0.0156			
7	0.0199	0.1044	0.2350	0.2938	0.2203	0.0991	0.0248	0.0027		
8	0.0233	0.1118	0.2347	0.2816	0.2112	0.1014	0.0304	0.0052	0.0004	
9	0.0260	0.1171	0.2341	0.2731	0.2048	0.1024	0.0341	0.0073	0.0009	
10	0.0282	0.1211	0.2335	0.2668	0.2001	0.1029	0.0368	0.0090	0.0014	0.0001
11	0.0301	0.1242	.02329	0.2620	0.1965	0.1031	0.0387	0.0104	0.0019	0.0002
12	0.0317	0.1267	0.2323	0.2581	0.1936	0.1032	0.0401	0.0115	0.0024	0.0004
13	0.0330	0.1288	0.2318	0.2550	0.1912	0.1033	0.0413	0.0124	0.0028	0.0005
14	0.0342	0.1305	0.2313	0.2523	0.1893	0.1032	0.0422	0.0132	0.0031	0.0006
100	0.0476	0.1471	0.2252	0.2275	0.1706	0.1013	0.0496	0.0206	0.0074	0.0023
∞	0.0498	0.1494	0.2240	0.2240	0.1680	0.1008	0.0504	0.0216	0.0027	0.0009

函数$\lambda^ke^{-\lambda}/k!$称为*泊松分布*，正如前面所提到的，该分布能够模拟许多随机过程。该分布的PGF为$e^{\lambda(z-1)}$，因此其均值和方差都是λ。更多的细节可参看Feller[8]。

在算法分析中，经常用泊松逼近来近似二项分布，例如像第8章中所描述的散列算法和涉及随机映射的其他问题的分析就是如此。

习题4.65 给出$\binom{N}{pN}p^{pN}(1-p)^{N-pN}$的渐近逼近。

习题4.66 对固定的p，给出$\binom{N}{k}p^k(1-p)^{N-k}$的渐近逼近。（*提示*：变换，使分布中最大的项在$k=0$处。）

习题4.67 对$p=\lambda/\sqrt{N}$的情形，给出二项分布的渐近逼近。

习题4.68 对$p=\lambda/\ln N$的情形，给出二项分布的渐近逼近。

4.7 拉普拉斯方法

在上一节的定理中，我们看到对于二元分布而言，不同的界适用于不同的范围。当在整个范围内估计和时，我们想利用我们的能力在不同的范围内得到被加数的精确估计。但另一方面，如果我们能在我们所关心的整个范围内固守单一一个函数的话，这无疑会给我们带来更多的方便。

本节将讨论一般的方法——*拉普拉斯方法*，它既能使我们估计和的值，也能使我们估计积分。在算法分析中，我们经常会遇到用这种方法来进行估计的和。通常情况下，我们也会利用我们的能力，用积分的方法来逼近这样的和。有关这种方法详尽的讨论和许多例子，可以

在Bender和Orszag[2]或De Bruijn[6]中找到。

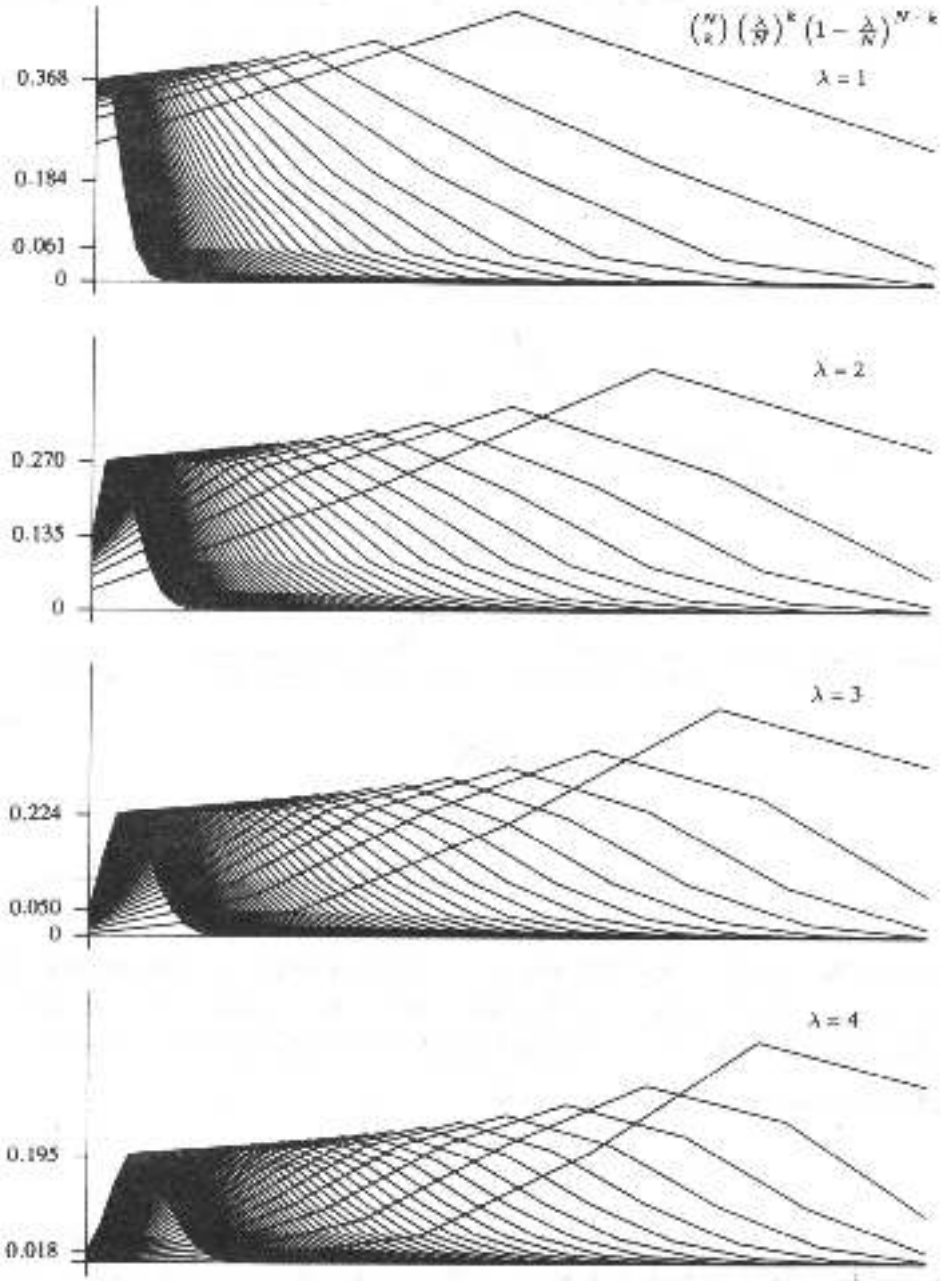


图4-5 二项分布，$3 \leqslant N \leqslant 60$及$p = \lambda/N$，趋于泊松分布$\lambda^k e^{-\lambda}/k!$
（k轴按N的比例标度）

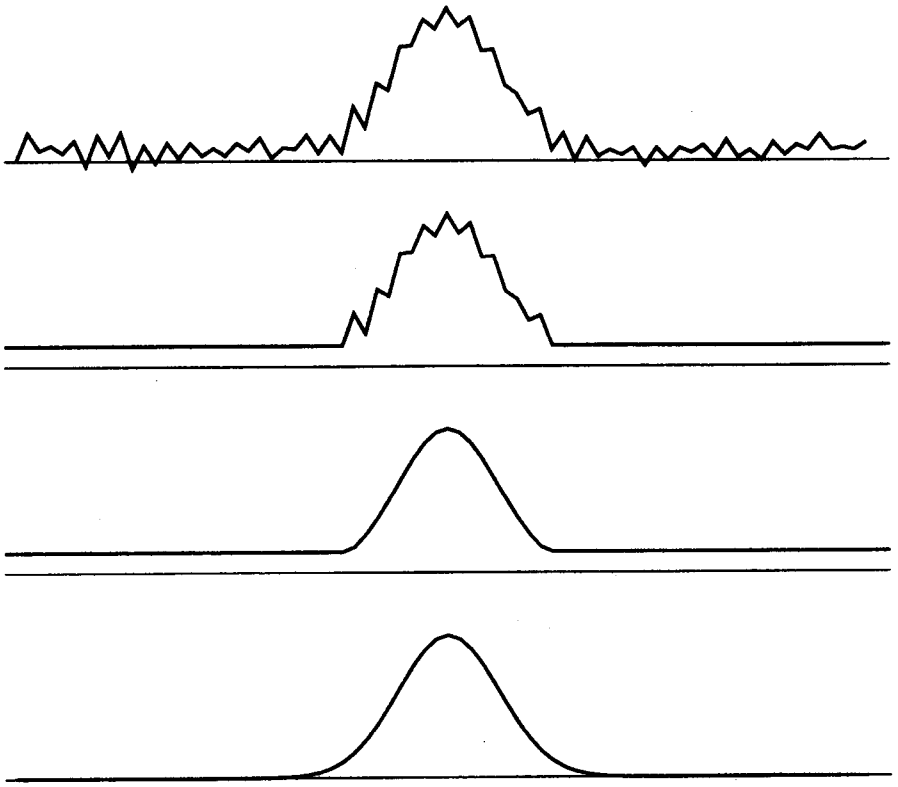
200 ~ 202

总之，该方法以下面三个步骤为中心对和来进行估计：

- 限制包含那些最大被加项的区域的范围；
- 逼近被加项，并对尾部定界；

• 扩展范围并对新的尾部定界，以得到一个更简单的和。

图4-6以一种图释的方式阐明这个方法。实际上当逼近和时，所涉及的函数都是阶梯函数；通常在逼近结束时，才以欧拉-麦克劳林公式应用的形式，呈现出一个“光滑”的函数。



203

图4-6 拉普拉斯法：尾部定界、逼近及扩展

应用拉普拉斯方法的一个典型的例子是对上一节开始部分我们引入的Ramanujan Q-函数值的计算。正如前面所提到过的，该函数在算法分析中是相当重要的，因为它出现在许多应用中，包括散列算法、随机映射、等价性算法以及内存缓存性能的分析等（见第8章）。

定理4.8 (Ramanujan Q-函数) 当$N\to\infty$时，

$$Q(N)=\sum_{1\le k\le N}\frac{N!}{(N-k)!N^k}=\sqrt{\pi N/2}+O(1)$$

证明 由于定理4.4中给出的两个估计都不适用于被加项的整个变化范围，所以我们必须把它们限定在它们所能适用的部分范围内。更确切地说，我们定义k_0为一整数，该整数是$o(N^{2/3})$级的量，并把和分成两部分：

$$\sum_{1\le k\le N}\frac{N!}{(N-k)!N^k}=\sum_{1\le k\le k_0}\frac{N!}{(N-k)!N^k}+\sum_{k_0<k\le N}\frac{N!}{(N-k)!N^k}$$

我们将充分利用这两部分对k的不同限制的条件，对这两部分分别进行逼近。对第一项（主项），我们利用定理4.4中的相对逼近。对第二项（尾部），$k>k_0$的限制以及它的各项是递减的事实

意味着这些项都是指数级的小量，这就和我们在定理4.4的证明中所讨论过的一样。把这两个结论合在一起，就证明了

$$Q(N)=\sum_{1<k<k_0}e^{-k^2/(2N)}\left(1+O\left(\frac{k}{N}\right)+O\left(\frac{k^3}{N^2}\right)\right)+\Delta$$

这里我们用Δ作为一个符号来表示一个项是指数级小量，否则就不特别指出这个项。此外对$k>k_0$，$\exp(-k^2/2N)$也是指数级小量，因而我们可以把$k>k_0$的那些项再加入进来，所以我们有

$$Q(N)=\sum_{k\geq 1}e^{-k^2/(2N)}+O(1)$$

实质上，我们已经用逼近的尾部替换了原来和中的尾部，这是合理的，因为二者都是指数级小量。我们把该式的误差项所贡献的一个绝对误差为$O(1)$这一结果的证明留做下面的一道习题，因为该证明只是关于主项证明的一个小小的改动，下一段中我们将对此给出讨论。当然，$O(1)$也吸收指数级小量。 204

剩下来的和是函数$e^{x^2/2}$以步长$1/\sqrt{N}$在有规律的间隔点上所取的函数值的和。因此，欧拉-麦克劳林定理提供了这个和的逼近

$$\sum_{k\geq 1}e^{-k^2/(2N)}=\sqrt{N}\int_0^\infty e^{-x^2/2}\mathrm{d}x+O(1)$$

该积分的值就是众所周知的$\sqrt{\pi/2}$，将其代入上面$Q(N)$表达式，就得出了定理所述的结果。■

注意，在这个例子中，既然大项是关于小的k值而出现的，因此我们只需处理一个尾部。一般情况下，正如图4-6所描述的，起决定作用的项都出现在范围内的中间某个地方，因此左、右尾部都需要处理。

习题4.69 把欧拉-麦克劳林求和应用到函数$xe^{-x^2/2}$和$x^3e^{-x^3/2}$上，证明

$$\sum_{1<k<k_0}e^{-k^2/(2N)}O\left(\frac{k}{N}\right)\qquad 和\qquad \sum_{1<k<k_0}e^{-k^2/(2N)}O\left(\frac{k^3}{N^2}\right)$$

都是$O(1)$。

Q-函数有几种形式；Knuth也定义了两种相关的函数——P-函数和R-函数（这是一个我们上面考察过的R-分布的和），表4-11列出了这些函数以及由Knuth所给出的渐近估计。注意根据Stirling逼近，我们有

$$Q(N)+R(N)=\sum_k\frac{N!}{k!}\frac{N^k}{N^N}=\frac{N!}{N^N}e^N=\sqrt{2\pi N}+\frac{1}{6}\sqrt{\frac{\pi}{2N}}+O\left(\frac{1}{N}\right)$$

习题4.70 证明：

$$P(N)=\sum_{k\geq 0}\frac{(N-k)^k(N-k)!}{N!}=\sqrt{\pi N/2}+O(1)$$

习题4.71 找出一个直接的论证方法，证明$R(N)-Q(N)\sim 2/3$。

4.8 算法分析中的“正态”例

有几个算法的分析要依赖类似于二项分布：

$$\sum_k F(k)\binom{2N}{N-k}\Big/\binom{2N}{N}$$

的求和。只要$F(k)$性能良好，这样的一个和就可以用拉普拉斯方法和欧拉-麦克劳林公式（定理4.2）精确地估计。我们将对这个和给出较详细的考查，因为它代表了算法分析中出现的许多类似的问题，还因为它提供了进一步阐明拉普拉斯方法的一个有力工具。

表4-11 Ramanujan P-函数、Q-函数、R-函数

$$
\begin{aligned}
Q(N) &= \sum_{1\le k\le N}\frac{N!}{(N-k)!N^k}\\
&= \sum_{1\le k\le N}\prod_{1\le j<k}\left(1-\frac{j}{N}\right) = \sum_{1\le k\le N}\left(1-\frac{1}{N}\right)\left(1-\frac{2}{N}\right)\cdots\left(1-\frac{k-1}{N}\right)\\
&= \sum_{0\le k\le N}\frac{N!}{k!}\frac{N^k}{N^N}\\
&= \sum_{k}\binom{N}{k}\frac{k!}{N^k}-1\\
&= \sqrt{\frac{\pi N}{2}}-\frac{1}{3}+\frac{1}{12}\sqrt{\frac{\pi}{2N}}+O\left(\frac{1}{N}\right)\\
P(N) &= \sum_{0\le k\le N}\frac{(N-k)^k(N-k)!}{N!}\\
&= \sum_{1\le k\le N}\prod_{1\le j<k}\left(\frac{N-k}{N-j}\right)\\
&= \sum_{0\le k\le N}\frac{k!}{N!}\frac{k^N}{k^k}\\
&= \sum_{k}\frac{(N-k)^k}{k!\binom{N}{k}}\\
&= \sqrt{\frac{\pi N}{2}-\frac{2}{3}+\frac{11}{24}\sqrt{\frac{\pi}{2N}}+O\left(\frac{1}{N}\right)}\\
R(N) &= \sum_{k\ge 0}\frac{N!N^k}{(N+k)!}\\
&= \sum_{1\le k\le N}\prod_{1\le j<k}\left(\frac{N}{N+j}\right)\\
&= \sum_{k\ge N}\frac{N!}{k!}\frac{N^k}{N^N}\\
&= \sqrt{\frac{\pi N}{2}}+\frac{1}{3}+\frac{1}{12}\sqrt{\frac{\pi}{2N}}+O\left(\frac{1}{N}\right)
\end{aligned}
$$

本节我们只简要概述涉及各种$F(k)$的应用的本质，这是因为这些应用完全是用引证的资料来描述的，我们将在几章中涉及一些相关的基本概念。在本节中，我们的目的是给出一个具体的例子，来说明渐近方法如何能对实际中出现的复杂表达式给出精确估计。上面这种类型的和有时叫做Catalan和，因为它们的产生与树的计数以及Catalan数有联系，正如我们在第5章将要看到的那样。它们也和路径的计数与归并算法有联系，我们在第6章中也将遇到这种问题。

2-级序列。如果数列中奇数位置上的数是递减的，而偶数位置上的数是递增的，这个数列叫做2-级序列。取两个有序的数列并将它们“混合”在一起（交替地从每个数列中取一个数），将得到一个2-级序列。在第6和第7章中，我们将考查这种序列的组合性质。许多归并和排序算法的分析都将得出对2-级序列的性质的研究。在Catalan和中取$F(k) = k^2$，将给出一

个2-级序列中逆序的平均数，这个数与用简单排序法对一个序列进行排序所花费的平均运行时间成正比。

Batcher奇-偶归并。另一个涉及排序方法的例子要归于Batcher（见[16]与[20]），这个算法适于硬件实现。取$F(k)= k\log k$将得出这种排序方法运行时间的首项，要得到更准确的分析也是可能的，不过要涉及复杂得多的$F(k)$。 205~207

对$F(k) = 1$，将立即得到结果 $4^N/\binom{2N}{N}$，我们已经证明过该结果渐近于 $\sqrt{\pi N}$。类似地对$F(k) = k$和$F(k) = k^2$，我们也能对Catalan和导出一个准确的值，该值是几个二项式系数的线性组合，然后用Stirling逼近找到一些渐近估计。与Batcher法中的分析一样，当$F(k)$更加复杂的时候，本节的方法就会发挥作用。我们对$F(k)$的主要假定就是：它可以由一个多项式来限定。

和定理4.6证明中的讨论一样，我们将对Ramanujan Q-分布和R-分布的乘积进行操作：

$$\frac{\binom{2N}{N-k}}{\binom{2N}{N}}=\frac{\dfrac{(2N)!}{(N-k)!(N+k)!}}{\dfrac{(2N)!}{N!N!}}$$

$$=\frac{N!N!}{(N-k)!(N+k)!}=\frac{N!}{(N-k)!N^k}\frac{N!N^k}{(N+k)!}$$

于是，我们就能利用定理4.6证明过程中所推导的结果以及该定理的推论在拉普拉斯方法中的应用来求解这个问题了。对于一个小常数$\varepsilon>0$，选取截断点 $k_0=\sqrt{2N^{1+\varepsilon}}$，我们得到如下逼近

$$\frac{N!N!}{(N-k)!(N+k)!}=\begin{cases}e^{-k^2/N}\left(1+O\left(\dfrac{1}{N^{1-2\varepsilon}}\right)\right) & \text{当}k<k_0\text{时}\\ O\left(e^{-(2N)^\varepsilon}\right) & \text{当}k\geqslant k_0\text{时}\end{cases}$$

这是我们逼近被加项，然后应用拉普拉斯方法所需要的基本信息。如上所述，我们利用该式的第一部分来逼近和的主要部分，用第二部分来对尾部定界：

$$\sum_k\frac{\binom{2N}{N-k}}{\binom{2N}{N}}F(k)=\sum_{|k|<k_0}F(k)e^{-k^2/N}\left(1+O\left(\frac{1}{N^{1-2\varepsilon}}\right)\right)+\Delta$$

其中，Δ仍代表指数级小项。和以前一样，我们仍可以把尾部重新加入进来，因为对$k>k_0$，$\exp(-k^2/N)$也是个指数级小量，这就得到： 208

定理4.9 (Catalan和) *如果$F(k)$能由一个多项式所定界，则*

$$\sum_k F(k)\binom{2N}{N-k}\Big/\binom{2N}{N}=\sum_k e^{-k^2/N}F(k)\left(1+O\left(\frac{1}{N^{1-2\varepsilon}}\right)\right)+\Delta$$

其中Δ表示指数级小的误差项。

证明 见前面的讨论。对于F的限制是保证误差项为指数级小量的需要的。 ■

如果序列$\{F(k)\}$是一个充分光滑的实函数$F(x)$的特定序列，那么该和就很容易用欧拉-麦克劳林公式来逼近。实际上，指数和$F(x)$的导数在∞处很快就会消失，因此所有误差项也都将在那里消失，并且在F的性能的适当条件下，我们期望有

$$\sum_{-\infty<k<\infty} F(k)\binom{2N}{N-k}\Big/\binom{2N}{N}=\int_{-\infty}^{\infty} e^{-x^2/N}F(x)\mathrm{d}x\left(1+O\left(\frac{1}{N^{1-2\varepsilon}}\right)\right)$$

对单侧求和也是一个类似的过程，其中对k的限制，比方说，是非负的，也将导致受到类似限制的积分。

Stirling常数。取$F(x)=1$则给出一个我们所熟知的积分，其预期的解为$\sqrt{\pi N}$。事实上，正如我们在4.5节中所承诺的那样，这个解相当于Stirling常数值的一个导数：我们知道，上面的和等于$4^N/\binom{2N}{N}$，通过与我们在4.3节中所做过的一样的基本处理可知它渐近于$\sigma\sqrt{N/2}$，但σ却作为一个未知量留在了Stirling公式中。取$N\to\infty$，我们将得到结果$\sigma=\sqrt{2\pi}$。

其他例子。在一个2-级文件中，平均逆序数可以由单侧形式的（即$k\geqslant 0$）Catalan和通过取$F(x)=x^2$来得到。这个积分很容易计算（分部积分法），其渐近结果为$N\sqrt{\pi N}/4$。对Batcher的归并方法，我们利用单侧和，取$F(x)=x\lg x+O(x)$，则得到如下估计

$$\int_0^{\infty} e^{-x^2/N}x\lg x\,\mathrm{d}x+O(N)$$

做替换$t=x^2/N$，将把积分变换成另一个众所周知的积分——“指数积分函数”，其渐近结果为

$$\frac{1}{4}N\lg N+O(N)$$

209

要获得关于$F(x)$的一个更好的逼近并不容易，正如[20]中所描述的，要获得更准确的逼近，必须利用复分析。表4-12总结了这些结果。

表4-12 Catalan和

$F(k)$	$\sum_{k\geqslant 0}F(k)\binom{2N}{N-k}\Big/\binom{2N}{N}$
1	$\sim\frac{\sqrt{\pi N}}{2}$
k	$\frac{N}{2}$
$k\lg k$	$\sim\frac{N\lg N}{4}$
k^2	$N4^{N-1}\Big/\binom{2N}{N}\sim\frac{\sqrt{\pi N^3}}{4}$

习题4.72 求关于$\sum_{k\geqslant 0}\binom{N}{k}^3$的渐近估计。

习题4.73 求关于$\sum_{k\geqslant 0}\frac{N!}{(N-2k)!k!2^k}$的渐近估计。

习题4.74 求关于$\sum_{k\geqslant 0}\binom{N-k}{k}^2$的渐近估计。

4.9 算法分析中的"泊松"例

另外几个基本算法可以导出对最大项在开头部分，且后面的项最终为指数级递减的和的求值。这种和更像是$\lambda<1$的泊松分布。这种和的一个例子是

$$\sum_k \binom{N}{k} \frac{f(k)}{2^k}$$

其中比如说，$f(k)$是一个快速递减的函数。

作为一个例子，我们考查第1章中所提到的基数交换排序法，该方法与第7章中的"trie树"数据结构有着紧密的联系。下面我们说明基数交换排序中描述位检查次数的递归的解将涉及"trie和" 210

$$S_N = \sum_{j\geq 0}\left(1-\left(1-\frac{1}{2^j}\right)^N\right)$$

展开这个二项式，并取$f(k)=(-1)^k,(-1/2)^k,(-1/4)^k,(-1/8)^k\cdots$等等，将得到上面形状的项的和。利用复分析可以精确地计算这个和，但利用逼近

$$1-\left(1-\frac{1}{2^j}\right)^N \sim 1-e^{-N/2^j}$$

可以很容易地得到一个非常好的估计。我们可以证明，当$j<\lg N$时，这个逼近可以精确到$O(1/N)$，且当$j\gg\lg N$时，两侧都非常小。所以，证明

$$S_N = \sum_{j\geq 0}(1-e^{-N/2^j})+o(1)$$

是一件初等的事情，通过将求和范围分成三个部分，不难得到该和的一个好的估计。如图4-7所示，对较小的j，被加项在1附近，对较大的j，被加项在0附近，当j在$\lg N$附近时，被加数从1到0之间过渡。更确切地说，当$j<(1-\varepsilon)\lg N$时，被加项非常接近于1；当$j>(1+\varepsilon)\lg N$时，它又非常接近于0；当j在这两个界之间时，该和无疑是在0和1之间。该论证证明了：对任何ε，一直到较小阶的项为止，该和在$(1-\varepsilon)\lg N$与$(1+\varepsilon)\lg N$之间；更精确地选择界，将证明该和$\sim\lg N$。

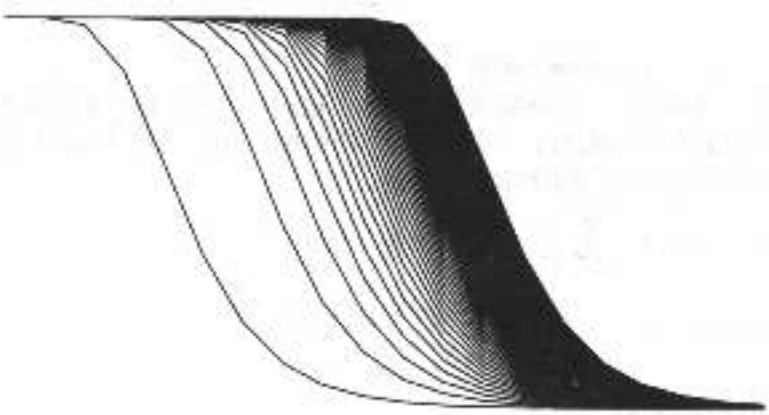


图4-7 $\sum_{j\geq 0}(1-e^{-N/2^j})$中的项的渐近行为

定理4.10 (Trie和) 当$N\to\infty$时，

$$S_N=\sum_{j\geq 0}\left(1-\left(1-\frac{1}{2^j}\right)^N\right)=\lg N+O(\log\log N)$$

证明 在上面的论证中，取界为$\lg N+\ln\ln nN$即可。更详细的论证可用来证明$S_N=\lg N+O(1)$。我们将在第7章中较详细地分析函数$S_N-\lg N$。 ■

推论 基数交换排序中所进行的位检验的平均次数是$N\lg N+O(N)$。

证明 如1.5节中所讨论过的，这个量满足递归

$$C_N=N+\frac{2}{2^N}\sum_k\binom{N}{k}C_k \qquad (N>1,\ C_0=0\ ,\ C_1=0)$$

两端乘以z^N并关于N求和，将得到一个简单易算的卷积，该卷积可以简化下面这个结论的证明：EGF$\sum_{N\geq 0}C_Nz^N/N!$必定满足函数方程

$$C(z)=ze^z-z+2e^{z/2}C(z/2)\ 。$$

正如我们可能会期望的那样，这个方程也能通过符号方法[10]得到。对该方程进行迭代，我们得到

$$\begin{aligned}C(z)&=ze^z-z+2e^{z/2}C(z/2)\\&=ze^z-z+2e^{z/2}\left(\frac{z}{2}e^{z/2}-\frac{z}{2}+2e^{z/4}C(z/4)\right)\\&=z(e^z-1)+z(e^z-e^{z/2})+4e^{3z/4}C(z/4)\\&=z(e^z-1)+z(e^z-e^{z/2})+z(e^z-e^{3z/4})+8e^{7z/8}C(z/8)\\&\vdots\\&=z\sum_{j\geq 0}(e^z-e^{(1-2^{-j})z})\end{aligned}$$

211
~
212

因此，

$$C_N=N![z^N]C(z)=N\sum_{j\geq 0}\left(1-\left(1-\frac{1}{2^j}\right)^{N-1}\right)$$

于是我们有$C_N=NS_{N-1}$，从而所断言的结果成立。 ■

在第7章将进一步讨论，一次项的值是摆动的。这或许并不奇怪，因为算法处理的是位，且具有2.6节中所讨论的某些算法的“二元分治”的韵味。不过，这一问题提出了重要的解析难题，这些难题用复分析方法将得到最好的处理。

习题4.75 求一个关于$\sum_{0<j<N}\left(1-\frac{1}{2^j}\right)^N$的渐近估计。

习题4.76 对$t>1$，求一个关于$\sum_{j\geq 0}(1-e^{-N/j^t})$的渐近估计。

4.10 生成函数的渐近性

就生成函数而言，符号法使我们分析大量复杂的组合问题成为可能。正如在许多例子中

已经表明的，我们常常能够展开这些生成函数，并逼近那些表示系数的组合和。然而，我们无疑会碰到许多生成函数，对它们这样的直接展开是不可行的。本节我们将提出一个直接的方法，即使是对复杂的生成函数而言，这个方法也是非常有效的。

收敛半径界。首先我们注意：了解一个生成函数的收敛半径，可以提供我们其系数增长速度的有关信息，这一点自从欧拉和柯西以来就为人们所知晓。

定理4.11（收敛半径界） 设$f(z)$是一个收敛半径$R>0$的幂级数，则对任何正数$r<R$，有

$$[z^n]f(z)=O(r^{-n})$$

证明 任取r满足$0<r<R$，并令$f_n=[z^n]f(z)$。级数$\sum_n f_n r^n$是收敛的；因此它的一般项$f_n r^n$趋于0，尤其是它的上界被一个常数所限定。 ■ 213

例如，Catalan生成函数对于$|z|<1/4$收敛，因为它含有$(1-4z)^{1/2}$，且二项级数$(1+u)^{1/2}$对$|u|<1$收敛。这就给出了它的界为

$$[z^n]\frac{1-\sqrt{1-4z}}{2}=O((4+\varepsilon)^n)$$

其中ε是任意的，这是由Stirling公式所导出的一个较弱的形式。

就组合生成函数而言，定理4.11的界是可以被加强的。更一般地，设$f(z)$具有正系数，那么

$$[z^n]f(z)\leqslant\min_{x\in(0,R)}\frac{f(x)}{x^n}$$

该结果可以简单地从$f_n x^n\leqslant f(x)$且$f_n x^n$只是一个收敛的正项和中的一项而证得。特别地，我们将在第6章中讨论带有约束圈长的排列时，要利用这个非常一般的定界技巧。

习题4.77 证明存在一个常数C，满足

$$[z^n]\exp(z/(1-z))=O(\exp(C\sqrt{n}))$$

习题4.78 关于整数分拆的OGF

$$[z^n]\prod_{k\geq 1}(1-z^k)^{-1}$$

确定类似的界。

系数的渐近分析。通过对生成函数的直接考察，我们不仅常常能够导出上界，而且也能导出渐近等价性。我们在4.1节中讨论有理生成函数的内容时，已经见到过关于这个问题的一个具体例子。例如，如果$g(z)$是一个多项式，r是一个整数，那么部分分式分解产生

$$[z^n]\frac{g(z)}{(1-z)^r}\sim g(1)\binom{n+r-1}{n}\sim g(1)\frac{n^{r-1}}{(r-1)!}$$

当然要假定$g(1)\neq 0$。实际上，一个更一般的结果也成立： 214

定理4.12（收敛半径逼近） 设$g(z)$具有严格大于1的收敛半径且假定$g(1)\neq 0$。则对任意的实数$\alpha\notin\{0,-1,-2,\cdots\}$，下式成立

$$[z^n]\frac{g(z)}{(1-z)^\alpha}\sim g(1)\binom{n+\alpha-1}{n}$$

证明 设$g(z)$的收敛半径>r，其中$r>1$。根据第一个定理，我们知道$g_n = [z^n]g(z) = O(r^{-n})$，特别地，和$\Sigma_n g_n$以几何速度收敛到$g(1)$。

接下来就是分析如下卷积:

$$[z^n]\frac{g(z)}{(1-z)^\alpha} = g_0\binom{n+\alpha-1}{n} + g_1\binom{n+\alpha-2}{n-1} + \cdots + g_n\binom{\alpha-1}{0}$$

$$= \binom{n+\alpha-1}{n}\left(g_0 + g_1\left(\frac{n}{n+\alpha-1}\right) + g_2\frac{n(n-1)}{(n+\alpha-1)(n+\alpha-2)}\right.$$

$$\left. + g_3\frac{n(n-1)(n-2)}{(n+\alpha-1)(n+\alpha-2)(n+\alpha-3)} + \cdots\right)$$

该和中带有下标j的项就是

$$g_j\frac{n(n-1)\cdots(n-j+1)}{(n+\alpha-1)(n+\alpha-2)\cdots(n+\alpha-j)}$$

当$n\to\infty$，它趋向于g_j。由此，我们推得

$$[z^n]g(z)(1-z)^{-\alpha} \sim \binom{n+\alpha-1}{n}(g_0 + g_1 + \cdots g_n) \sim g(1)\binom{n+\alpha-1}{n}$$

因为部分和$g_0 + g_1 + \cdots + g_n$以几何速度收敛到$g(1)$。 ■

习题4.79 对上面的断言给出详细的证明。

对各种特定的值α，我们可以导出更详细的渐近界。根据欧拉-麦克劳林公式，二项式系数$\binom{n+\alpha-1}{n}$为$O(n^{\alpha-1})$，且常数也是已知的（见习题4.80）。例如，我们已经见到过（在3.4节）

$$\binom{n-1/2}{n} = (-1)^n\binom{-1/2}{n} = \binom{2n}{n}/4^n \sim \frac{1}{\sqrt{\pi n}}$$

215

应用这种技巧的一个著名的例子(参见Comtet[5])就是将其用于函数

$$f(z) = \frac{e^{z/2+z^2/4}}{\sqrt{1-z}}$$

即所谓的2-正则图的EGF。它含有分子$g(z) = \exp(z/2 + z^2/4)$，其收敛半径显然为∞。于是定理4.12将立即给出

$$[z^n]f(z) \sim \frac{e^{3/4}}{\sqrt{\pi n}}$$

利用诸如定理4.12的结果，系数的渐近形式可直接从形如$(1-z)^{-\alpha}$的元素（称为“奇异”元素）转换而来，这种元素所起的作用与有理函数分析中的部分分数元素所起的作用非常类似。系数的渐近形式能被如此简单的方式进行解释是相当出乎意料的。在第6章，我们还将看到更多的与排列有关的这种例子。

定理4.12能自然地应用于许多我们将要遇到的具有不同收敛半径的生成函数乘积的场合。该定理只是全部类似结果中最简单的一个，这些结果是由Darboux于上上个世纪开始提出（参见[5]和[22]），并由Pólya和Szegö、Bender以及其他一些学者等人[3][9]进一步发展而得来的。这些方法在[11]中有详细的讨论，不像我们在这里所能够做的，对这些方法完整的研究需要复

变函数的理论。这种研究渐近性的方法称为奇异性分析（singularity analysis）。

习题4.80 证明：对某个常数$\Gamma(\alpha)$，有

$$[z^n]\binom{n+\alpha-1}{n} \sim \frac{n^{\alpha-1}}{\Gamma(\alpha)}$$

（提示：利用欧拉-麦克劳林求和。）常数$\Gamma(\alpha)$实际上是众所周知的伽马函数，它把阶乘函数推广到了整数范围之外，参见[21]。

习题4.81 证明：在定理4.12的条件下，可以导出一个完全渐近展开式。

习题4.82 将该方法推广到下列式子的分析中

$$[z^n]g(z)\log\frac{1}{1-z} \qquad 和 \qquad [z^n]\frac{g(z)}{1-z}\log\frac{1}{1-z}$$

216

渐近方法在算法分析中起着重要的作用。如果没有渐近性，我们可能还停留在极为复杂的准确答案状态，或是难以计算的闭型解式状态。利用渐近性，我们可以把注意力集中在解的那些对答案影响最大的部分上面。这种来自于分析的特别的洞察能力在算法设计过程中也起着一定的作用：当我们试图改进一个算法性能的时候，我们将把注意力集中在算法的某些部分上，这些部分恰好就是我们在渐近分析中所关注的那些项。

理解在渐近分析中经常使用的各种记号的区别是至关重要的。在本章中，我们引入了许多基本的习题和例子，以助于阐明这些区别，我们鼓励读者对它们进行仔细的研究。正确地使用基本定义和基本操作将能够大大简化渐近公式。

对生成函数而言，渐近分析的许多基本资料都是伴随着与著名的经典展开式相结合而产生的，因为生成函数与著名的特殊数有着密切的联系，如Stirling数、调和数、几何级数，以及二项式系数等。代数操作和化简也起着重要的作用。事实上，在这样的计算中，控制细节的能力将会使渐近表达式变得很有魅力。

我们详细考查了这些方法的应用，以导出二项分布逼近中的两个最重要的逼近：正态逼近和泊松逼近。我们还考查了由Ramanujan和Knuth所研究过的、将会在许多算法分析中出现的一些相关的函数。这些逼近不仅在算法分析中极为重要，而且它们还是经常出现的一类操作的原型。

我们把精力集中在来自实分析的所谓的基本方法上面。在基本方法中，熟练程度是很重要的，尤其是化简复杂的渐近表达式、细化估计、用积分逼近和，以及尾部定界，包括拉普拉斯方法等等。对渐近分析更好的理解依赖于对复平面上函数性质的理解。令人惊讶的是，强大的方法只源自于几个基本的性质，尤其是生成函数的奇异性。我们对基本方法给出了一个一般性的概念，详细的考查在[11]中可以找到。由于答案中会出现复数值，因而在详尽的渐近分析中，往往避免不了先进的技巧。特别地，我们看到许多例子，随着我们所研究的函数的增长，出现了一种摆动现象。这似乎有些出人意料，但当我们想到这种现象也曾出现在我们最基本的算法中，包括分治方法如Mergesort或涉及整数的二进制表示法，这种现象也就不足为怪了。复分析提供了一个简单的方法来解释这种现象。 217

我们可以用类似于上一章的方式来结束本章：本章所给出的方法，可以使我们在对基本计算机算法重要性质的研究中走得更远。这些技巧对我们在第5到第8章处理与树、排列、串及映射等有关的算法中将发挥重要的作用。

参考文献

1. M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*, Dover, New York, 1970.
2. C. M. Bender and S. A. Orszag. *Advanced Mathematical Methods for Scientists and Engineers*, McGraw-Hill, New York, 1978.
3. E. A. Bender. "Asymptotic methods in enumeration," *SIAM Review* **16**, 1974, 485–515.
4. B. C. Berndt. *Ramanujan's Notebooks, Parts I and II*, Springer-Verlag, Berlin, 1985 and 1989.
5. L. Comtet. *Advanced Combinatorics*, Reidel, Dordrecht, 1974.
6. N. G. De Bruijn. *Asymptotic Methods in Analysis*, Dover, New York, 1981.
7. A. Erdélyi. *Asymptotic Expansions*, Dover, New York, 1956.
8. W. Feller. *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 1957.
9. P. Flajolet and A. Odlyzko. "Singularity analysis of generating functions," *SIAM Journal on Discrete Mathematics* **3**, 1990, 216–240.
10. P. Flajolet, M. Regnier, and D. Sotteau. "Algebraic methods for trie statistics," *Annals of Discrete Mathematics* **25**, 1985, 145–188.
11. P. Flajolet and R. Sedgewick. *Analytic Combinatorics*, in preparation.
12. R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.
13. D. H. Greene and D. E. Knuth. *Mathematics for the Analysis of Algorithms*, Birkhäuser, Boston, 1981.
14. Peter Henrici. *Applied and Computational Complex Analysis*, 3 volumes, John Wiley, New York, 1977.
15. D. E. Knuth. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
16. D. E. Knuth. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
17. D. E. Knuth. "Big Omicron and big Omega and big Theta," *SIGACT News*, April-June 1976, 18–24.
18. A. Odlyzko. "Asymptotic enumeration methods," in *Handbook of Combinatorics*, R. Graham, M. Grötschel, and L. Lovász, eds., North Holland, 1995.
19. F. W. J. Olver. *Asymptotics and Special Functions*, Academic Press, New York, 1974.
20. R. Sedgewick. "Data movement in odd-even merging," *SIAM Journal on Computing* **7**, 1978, 239–272.
21. E. T. Whittaker and G. N. Watson. *A Course of Modern Analysis*, 4th edition, Cambridge University Press, Cambridge (England), 1927 (reprinted 1973).
22. H. Wilf. *Generatingfunctionology*, Academic Press, San Diego, 1990.

218

219

第5章 树

树是在许多实际算法中直接和间接用到的基本结构，为了能够分析这些算法，理解树的性质是很重要的。许多算法直接构建树；在另外一些情形下，树作为程序特别是递归程序的模型发挥着重要的作用。事实上，树是重要的经典递归定义的对象：一棵树或者为空，或者是连接一系列树（或树的多重集）的一个根结点。我们将详细考察结构的递归性质如何直接得出基于生成函数的递归分析。

本章的讨论从二叉树开始，它是一种特殊类型的树，最初在第3章引入。二叉树有很多有用的应用，而且特别适合用于计算机实现。然后，我们一般地考虑树的性质，包括它们与二叉树的紧密关系。树和二叉树也直接与若干像格路径、三角剖分和破产序列等其他组合学结构相关。我们讨论多种不同的表示树的方法不只是因为它们常常出现在应用中，而且还因为更换表示方法时常常使得解析论述更容易理解。

我们从纯组合学的观点（在这里列举并考查所有可能不同的结构的性质）以及算法的观点（此时这些结构由算法建立和使用，每种结构出现的概率由输入产生）考虑二叉树。前者在递归结构和算法分析中很重要，后者最重要的实例是称为二叉树搜索（binary tree search）的基本算法。二叉树和二叉树搜索在实践中是如此重要，我们需要相当详细地研究它们的性质，在第3章开始处理的基础上进行更充分的讨论。

像通常一样，在考虑计数问题之后，我们转到参数分析。重点在于路径长（path length）分析，它是树的基本参数，学习它很自然，了解它也很有用。我们还要考虑高（height）和某些其他的参数。关于各种类型的树的路径长和高的基本知识对于我们能够理解各种基本的计算机算法是至关重要的。对这些问题的分析是以那些基础结构的经典组合学的研究和现代算法研究间的关系为原型的，是本书反复出现的论题。

我们将看到，一方面路径长分析自然地安排在我们已经开发的基本工具之后，并且这种分析经过推广以提供一种研究更广泛的树参数的方法。另一方面，高的分析却提出了重要的技术挑战，同时也需要我们在第2章到第4章涉及的大部分主要工具。虽然在减轻描述问题和降低求解问题的困难之间没有任何必然的关系，但是在分析难度缓解中的差距初看起来多少有些令人惊奇，因为两种参数都有简单的递归表述，这些表述非常类似。

第3章讨论过，经典组合数学中的“符号方法”能够使得树计数问题的研究和树参数的分析统一起来，导致许多其他难以处理的问题得到非常直接的解法。为获得最佳效果，这需要在某些基本的组合学工具方面适当的投入（见[13]），因此，我们对本章考虑的许多重要问题提供直接的解析推导，同时也提供对结果如何能够由符号论证进行解释的非正式描述。这一章的详细研究也可以作为一道用以理解符号方法意义的习题来刻画其特征。

然后我们考虑一些不同类型的树，和关于树的性质的某些经典组合学结果，从二叉树的具体细节转到树作为无圈连通图的一般概念。我们的目标是提供通向树的组合学分析广泛文献结果的途径，同时，也提供大量算法应用的基础性工作。

5.1 二叉树

在第3章我们遇到二叉树，或许它是最简单的一种树。二叉树由两种不同类型的结点组成，

它们按照简单的递归定义被连接在一起:

定义 一棵二叉树或者是一个外部结点，或者是一个连接两棵有序的二叉树的内部结点，这两棵有序的二叉树分别叫做该结点的左子树和右子树。

222

通常外部结点用于表示空二叉树。它们同样可以作为占位符使用。除非课文中同时在考虑两种结点，否则我们就把内部结点简单地叫做树的“结点”。正常情况下我们认为一个结点的两棵子树由两个链连接到该结点上，这两个链是左链和右链。

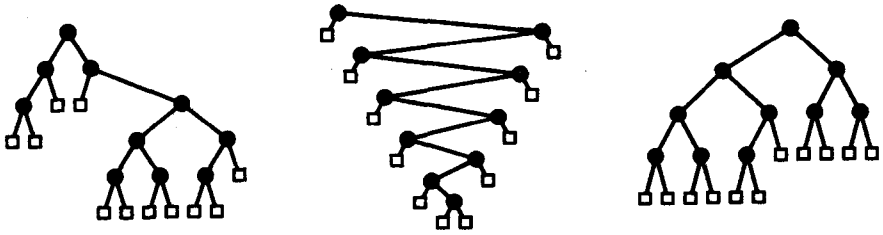
图5-1显示三棵二叉树，而图5-2中画出14棵二叉树，每棵都有四个内部结点（和五个外部结点）。根据定义，每个内部结点正好有两个链；按照习惯，我们把每个结点的子树画在该结点的下面，而把链用连接结点的线来表示。除去顶上的结点外，每个结点恰好有一个链通向它，顶上的特殊结点叫做根（root）。习惯上我们借用族谱中的术语：位于结点直接下方的结点叫做该结点的子结点；再往下面的那些结点叫做该结点的后裔结点；每个结点直接上方的结点叫做它的父结点；再往上面的结点叫做那个结点的祖先结点。外部结点没有子结点，而根结点没有父结点。

引理 在任意一棵二叉树中，外部结点的个数恰好比内部结点的个数多1。

证明 令e是外部结点的个数，i为内部结点的个数。我们以两种不同的方法对树的链进行计数。每个内部结点恰好有两个链从它发出，因此链的总数是$2i$。但是链的总数还等于$i+e-1$，因为除去根外每个结点恰好有1个链通向它。使这两个数相等，得到$2i=i+e-1$，或即$i=e-1$。 ■

习题5.1 利用归纳法写出上面结果的另外的证明。

我们已经考虑了二叉树的计数问题：下面的基本结果在第3章和第4章有过详细描述。



223

图5-1 三棵二叉树

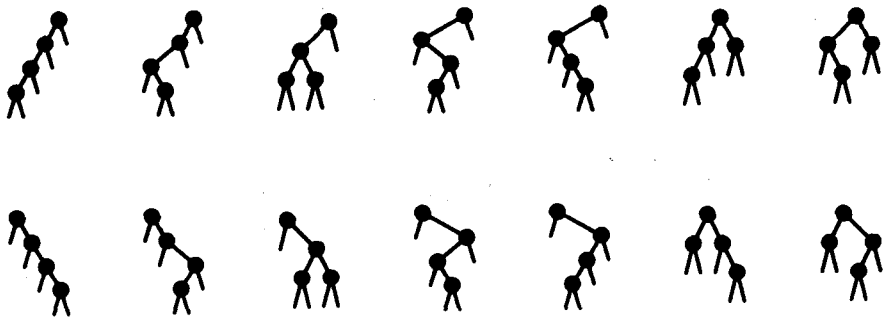


图5-2 具有四个内部结点的二叉树

定理5.1（二叉树的计数） 具有N个内部结点和$N+1$个外部结点的二叉树的棵数由Catalan数给出：

$$T_N = \frac{1}{N+1}\binom{2N}{N} = \frac{4^N}{\sqrt{\pi N^3}}\left(1+O\left(\frac{1}{N}\right)\right)$$

证明 我们简要叙述3.8节给出的证明。令$\mathcal{T}$表示所有二叉树的集合，相关的常规生成函数为

$$T(z) = \sum_{t\in\mathcal{T}} z^{|t|} = \sum_{N\geq 0} T_N z^N$$

就是说$T_N = [z^N]T(z)$ 是具有N个内部结点的二叉树的棵数。另外$zT(z)$还是由外部结点计数的树的OGF，因为$[z^N]zT(z) = T_{N-1}$是具有N个外部结点的二叉树的棵数。

一棵二叉树或者是一个外部结点，或者是一个连接两棵有序二叉树的内部结点，于是，符号方法告诉我们

$$\mathcal{T} = \{\square\} + (\bullet)\times\mathcal{T}\times\mathcal{T}$$

这直接变换成函数方程

$$T(z) = 1 + zT(z)^2$$

（注意，外部结点的计数给出等价的形式$zT(z) = z + (zT(z))^2$）利用二次公式求解并用二项式定理展开，得到

$$zT(z) = \frac{1}{2}\left(1-\sqrt{1-4z}\right) = -\frac{1}{2}\sum_{N\geq 1}\binom{\frac{1}{2}}{N}(-4z)^N$$

令系数相等则得到Catalan数

$$T_N = -\frac{1}{2}\binom{\frac{1}{2}}{N+1}(-4)^{N+1} = \frac{1}{N+1}\binom{2N}{N}$$

该近似值直接从Stirling近似（见定理4.3的推论）得到。 ■

在第3章，我们还看到一些相关的问题，像对二叉树的森林计数、对树叶计数等等。在这一章，我们将对更加复杂的问题使用类似的工具，以便发现二叉树和其他类型的树的有趣性质。

从现在起我们通过内部结点来对树进行计数。对于二叉树，由上面的引理可知这是严格等价的，有些其他类型的树没有外部结点。

习题5.2 在所有具有N个结点的二叉树当中占多大比例的二叉树其根的两棵子树均非空？对于$N=4$，答案是4/14（见图5-2）。

习题5.3 在所有具有$2N+1$个内部结点的二叉树当中占多大比例的二叉树其根的每棵子树都有N个内部结点？

5.2 树和森林

在二叉树中不存在有多于两个子结点的结点。这个特点使得在计算机的实现中如何表示和处理这样的树变得很明显，而这又自然地涉及把一个问题分成两个子问题的“分治算法”。然而，在许多应用中（以及更传统的数学应用中）我们需要考虑一般的树。

定义 一棵树（也叫做一般树）是一个连接到一系列不相交的树的结点（叫做根）。这样的一个树的序列叫做森林。

我们使用和二叉树相同的命名法则：一个结点的子树是它的子结点，根结点没有父结点，等等。对于某些计算而言，树是比二叉树更合适的模型。

所有5个结点的树如图5-3所示。这样的树的数目和具有4个内部结点的二叉树的数目相同的事实不是巧合。事实上如所周知，一般树在组合学上是与二叉树紧密相关的。下面我们考查建立这种关系的结构，不过首先我们考虑基于符号方法的解析证明。

224 ~ 225

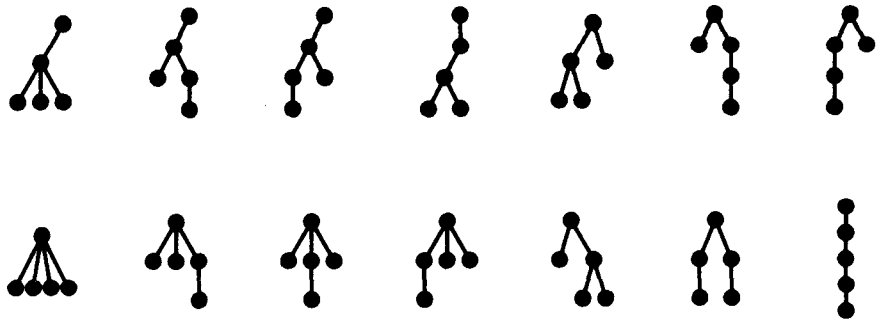


图5-3 具有5个结点的树

定理5.2（一般树的计数） 令G_N为具有N个结点的一般树的数目。则G_N恰好等于具有$N-1$个内部结点的二叉树的数目，并且该树由Catalan数给出：

$$G_N = T_{N-1} = \frac{1}{N}\binom{2N-2}{N-1}$$

证明 考虑生成函数

$$G(z) = \sum_{g \in \mathcal{G}} z^{|g|}$$

其中$\mathcal{G}$为所有树的集合。此时，由于一棵树可以有任意棵的子树，而这些子树也是树，因此我们有

$$\begin{aligned} G(z) &= \sum_{k \geq 0} \sum_{g_1 \in \mathcal{G}} \cdots \sum_{g_k \in \mathcal{G}} z^{|g_1|+\cdots+|g_k|+1} \\ &= z \sum_{k \geq 0} (G(z))^k \\ &= \frac{z}{1-G(z)} \end{aligned}$$

这样$G(z) - G(z)^2 = z$，从而我们得到$G(z) = zT(z)$，因为这两个函数都满足相同的函数方程。这就是说，具有N个结点的树的数目等于具有N个外部结点的二叉树的数目，也就是具有$N-1$个内部结点的二叉树的数目。 ■

226

这个结果也可以容易地通过符号方法得到。一个森林或者为空，或者为一系列的树：

$$\mathcal{F} = \varepsilon + \mathcal{G} + (\mathcal{G} \times \mathcal{G}) + (\mathcal{G} \times \mathcal{G} \times \mathcal{G}) + (\mathcal{G} \times \mathcal{G} \times \mathcal{G} \times \mathcal{G}) + \cdots$$

它可以直接变换（见定理3.7）成

$$F(z)=1+G(z)+G(z)^2+G(z)^3+G(z)^4+\cdots=\frac{1}{1-G(z)}$$

这种在森林和树（切掉根）之间明显的一一对应意味着$zF(z)=G(z)$，它又导出CatalanOGF的函数方程。

习题5.4 对于$t=1$、2和3，（渐近地）找出其根有t个子结点的那些树所占的比例。

旋转对应。在森林和二叉树之间存在一种容易构造的一一对应，使得二叉树能够用在计算机上表示一片森林。这种对应叫做旋转对应，或者叫做“第一个儿子，下一个兄弟”表示法，见图5-4中的解释。给定一片森林，它可以被一棵二叉树表示如下：二叉树的根为森林中第一棵树的根；它的右链指向森林其余部分的表示结果（不包括第一棵树）；它的左链指向由第一棵树的根的那些子树组成的森林的表示结果。换句话说，每一个结点有一个链接到它的第一个子结点左链和一个链接到它在森林中下一个兄弟结点的右链。对于图5-4，二叉树中的那些结点好像是一般树按照顺时针旋转方向放置的。

习题5.5 描述一棵给定的二叉树如何被表示成一片森林。

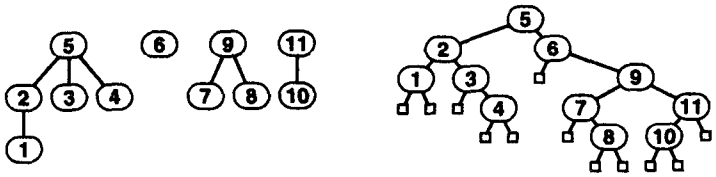


图5-4 在树和二叉树之间的旋转对应

227

5.3 树的性质

树自然地出现在计算机的各种应用中。树的大小的概念对于一般树通常指得是结点的个数，而对于二叉树，也可能是内部结点数或外部结点数，这要看上下文的情况而定。对于算法分析，我们主要的兴趣在于给定大小的树的两个基本性质：路径长和高。

我们常常谈到树中结点的层：根结点位于0层上，根的子结点在第1层，一般地，第k层的结点的子结点在第$k+1$层上。另外一种思考层的方法是从根到当前结点所必须经过的距离（链的数目）。我们特别对从每个结点到根的距离的和有特殊的兴趣。

定义 *给定一棵树t，树t中的每一个结点的层的和是路径长，该树所有结点中层的最大值为树的高。*

我们使用记号$|t|$表示树t的结点的个数，用记号$\pi(t)$表示路径长，而用$\eta(t)$表示树的高。这些定义对于森林也是成立的。此外，森林的路径长是组成该森林的树的路径长的和，而森林的高是组成该森林的树的高的最大值。

定义 *给定一棵二叉树t，其内部路径长为树t中每个内部结点的层的和，外部路径长是树t中每个外部结点的层的和，它的高为该树所有外部结点中的层的最大值。*

我们用记号$\pi(t)$表示二叉树的内部路径长，$\xi(t)$表示外部路径长，而用$\eta(t)$表示二叉树的高。除非在课文中特别声明用外部结点或所有结点进行计数更合适外，我们一般都用$|t|$表示二叉树的内部结点的个数。

定义 *在一般树中，树叶被定义为那些没有子结点的结点。在二叉树中，树叶被定义*

228 为其两个子结点都是外部结点的（内部）结点。

图5-4中右边的二叉树高为5，内部路径长为25，而外部路径长为47，有4片树叶；左边的森林高为2，路径长为8，有7片树叶。

递归定义。对于树的参数，使用递归定义常常很方便。在二叉树t中，如果t是一个外部结点，则我们刚刚定义的那些参数均为0；否则如果t的根是一个内部结点，且左、右子树分别用t_l和t_r表示，则有下列递归公式：

$$|t| = |t_l| + |t_r| + 1$$
$$\pi(t) = \pi(t_l) + \pi(t_r) + |t| - 1$$
$$\xi(t) = \xi(t_l) + \xi(t_r) + |t| + 1$$
$$\eta(t) = 1 + \max(\eta(t_l), \eta(t_r))$$

容易看出，这些公式与上面的定义是等价的。首先，二叉树的内部结点数是它的子树的结点个数的和加1（根）。其次，内部路径长是它的子树的内部路径长的和加$|t| - 1$，因为在这些子树中的$|t| - 1$个结点中的每一个结点当子树连接到原树上时都下移恰好1层。同样的论断对于外部路径长也是成立的，但应指出，具有$|t|$个内部结点的二叉树的两棵子树有$|t| + 1$个外部结点。关于高的结果由下面的事实得出：子树上所有结点的层恰好都比原来的层增1。

习题5.6 给出描述一般树的路径长和高的递归公式。

习题5.7 给出二叉树中以及一般树中树叶数目的递归公式。

当我们下面对参数进行分析的时候，上面这些定义将作为根据相关的生成函数推导出函数方程的基础。此外，它们还可用于对参数间的关系的归纳证明。

引理 任意二叉树t中的路径长满足$\xi(t) = \pi(t) + 2|t|$。

证明 从方程$\xi(t) = \xi(t_l) + \xi(t_r) + |t| + 1$减去方程$\pi(t) = \pi(t_l) + \pi(t_r) + |t| - 1$，该引理直接由
229 归纳法得到。 ■

路径长和高不是独立的参数：如果高很大，那么路径长必然也大，如下面的界所示，这个公式相对粗糙，不过有用。

引理 任意非空二叉树t的高和内部路径长满足不等式

$$\frac{\pi(t)}{|t|} \leqslant \eta(t) \leqslant \sqrt{2\pi(t)} + 1$$

证明 给定高度后，依据路径长直接的界得到。如果$\eta(t) = 0$，则$\pi(t) = 0$并且所述不等式成立。否则，我们必然有$\pi(t) < |t|\eta(t)$，因为每个内部结点的层必然严格小于树的高。不仅如此，在每个小于高的层上至少存在一个内部结点，因此我们必然有$\pi(t) \geqslant \sum_{0 \leqslant i < \eta(t)} i$。因此，$2\pi(t) \geqslant \eta(t)^2 - \eta(t) \geqslant (\eta(t) - 1)^2$（从右边减去量$\eta(t) - 1$，它是非负的），从而$\eta(t) \leqslant \sqrt{2\pi(t)} + 1$。将这两个不等式结合起来，最后得到引理的结果。 ■

在各种类型“随机”树的算法分析中，我们特别对于这些参数的平均值有兴趣。本章我们主要的讨论课题之一是这些量如何与基本算法相关的以及我们如何能够确定它们的期望值。图5-5和图5-6给出对于不同类型的树这些量如何不同的启示。图5-5是一棵随机二叉树，此时每棵二叉树被认为是等可能地出现的。图5-6是一片随机森林，其中每片森林都是等可能地出现的。每幅图还有在每层上的结点个数的标示，这些标示使得计算高（层数）和路径长（用i乘以在层i的结点数并对i求和）更容易。显然，随机二叉树的路径长和高的值要比随机森林的值大。本章主要目标之一是精确地确定这些量以及类似的量的值。

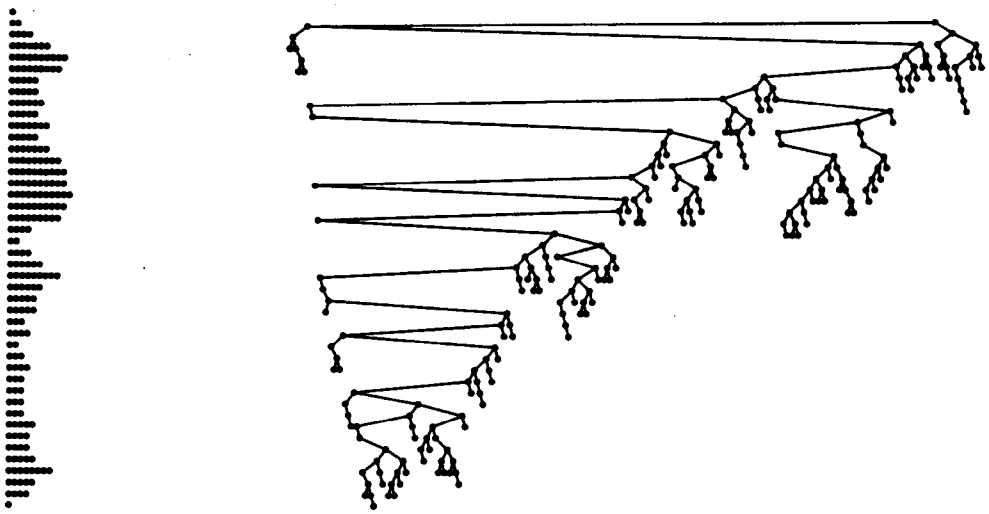


图5-5 具有256个内部结点的随机二叉树

习题5.8 证明具有N个结点的二叉树的高必然至少为$\lg N$。

习题5.9 [Kraft等式] 令k_j是二叉树的层j上的外部结点数。序列$\{k_0, k_1, \cdots, k_h\}$(其中$h$是树的高)描述了该树的全貌。证明，整数的向量描述二叉树的全貌当且仅当$\sum_j 2^{-k_j} = 1$。

习题5.10 给出具有N个结点的一般树的路径长的严格上界和下界。

习题5.11 给出具有N个内部结点的二叉树的内部路径长和外部路径长的严格上界和下界。

习题5.12 给出具有N个结点的二叉树的树叶数的严格上界和下界。

5.4 树的算法

树与算法分析的研究相关不仅因为树本身隐式地模拟了递归程序的行为，而且还因为树直接参与了广泛使用的许多基本算法。我们这里将简要描述一些最基本的这类算法。这种简要描述当然不能妥善处理算法设计中树结构的一般实用课题，不过我们可以指出，诸如路径长和高这类树参数的研究能够提供为大量重要算法的分析所需要的基本信息。

遍历。在计算机表示中，对树进行的基本操作之一就是遍历：系统地处理树的每一个结点。这种操作从组合学上看还是有趣的，因为它代表一种在（二维）树结构和各种（一维）线性表示之间建立一种对应的方式。

230 ~ 231

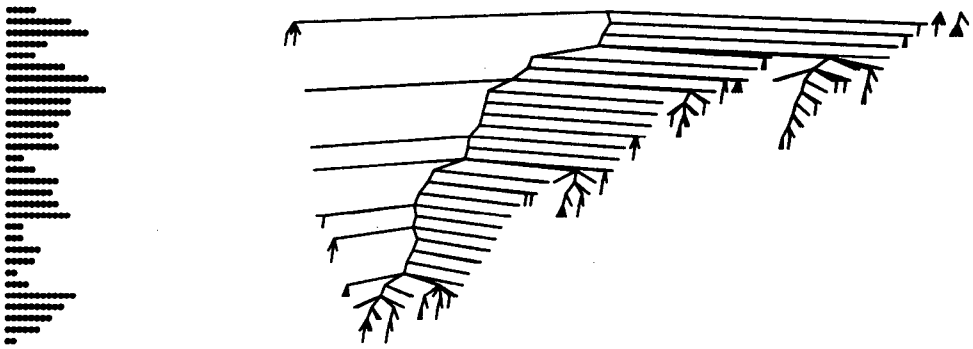


图5-6 具有256个结点的随机森林

树的递归性质导致遍历的一种简单的递归过程。我们“访问”树的根并递归地“访问”子树。根据是否我们在之前、之后或（对于二叉树）在两棵子树之间访问过根，可以得到三种不同类型的遍历之一：

以*先序*（preorder）访问树的所有结点：

- 访问根
- 访问子树（仍以先序）

以*后序*（postorder）访问树的所有结点：

- 访问子树（仍以后序）
- 访问根

以*中序*（inorder）访问树的所有结点：

- 访问左子树（仍以中序）
- 访问根
- 访问右子树（仍以中序）

程序5.1是二叉树先序遍历的一种实现。在这些方法中，对根的“访问”意味着应该系统地应用到整个树的结点上的任意过程。不同的过程可能适应不同类型的结点，例如二叉树的外部结点。

232

当一个递归过程调用被执行的时候，*下推栈*（pushdown stack）被用来存储当前的“环境”，以便在从该过程返回时会恢复环境。当遍历一棵树时由下推栈所使用的内存的最大量直接与树的高成比例。虽然在这种情况下内存的使用可能被程序设计人员隐蔽，但是它确实是一种重要的性能参数，我们有兴趣分析树的高。

另一种遍历树的方法叫做*层序*（level order）遍历：首先列出0层上所有的结点（根），然后从左到右列出1层上所有的结点，然后从左到右列出第2层上所有的结点，等等。这种方法不适合作为递归程序实现，但是它很容易非递归地实现，就像上面使用的是一个队列（先进先出数据结构）而不是栈。

程序5.1 二叉树的先序遍历

```
procedure traverse(x:  link);
  begin
  if x <> NIL then
    begin
    visit (x^.key);
    traverse (x^.left);
    traverse (x^.right)
    end
  end;
```

树的遍历算法是基本的和广泛适用的算法。关于这些算法的更多的细节以及递归实现和非递归实现之间的关系，可见Knuth[21]或Sedgewick[29]。

表达式求值。考虑由+、-、*、/等操作以及由数和字母表示的操作数组成的算术表达式。这样的表达式一般使用括号来指出运算之间的优先级。表达式可以被表示成二叉树，叫做*分析树*（parse trees）。例如，考虑只使用二目运算符的表达式的情况。这样的表达式对应一棵二叉树，操作符在内部结点上而操作数在外部结点上，如图5-7所示，它描绘了表达式

233

$$((x+y)*z)-(w+((a-(v+y))/((z+y)*x)))$$

的分析树。给定一棵分析树，我们可以使用一个简单的递归程序计算出对应表达式的值：（递归地）计算两棵子树的值，然后再应用运算符对所算得的值进行计算。外部结点的求值

给出相关变量的当前值。这个程序等价于诸如程序5.1这样的树遍历。如同树遍历，该程序的空间需求将正比于树的高。

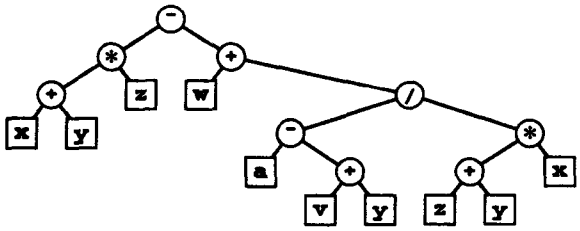


图5-7 一个算术表达式的树表示

一般说来，在解释程序中表达式树求值是直接如上面描述完成的，但是在由编译器所生成的代码中则是间接完成的。就是说，编译器可以先建立一个与某部分程序相联系的表达式树，然后将表达式树转换成一系列指令以便在程序执行时算出表达式的值。所使用的这些指令与机器指令很接近，例如，涉及用到机器寄存器的二进制算术运算。这种变换对应“代码生成”并通常导致寄存器分配问题。例如，下列代码可能由图5-7中的树生成:

```
r1←x+y
r2←r1*z
r3←v+y
r4←a-r3
r5←z+y
r6←r5*x
r7←r4/r6
r8←w+r7
r9←r2-r8
```

这种类型的代码能够容易地转换成机器码。在典型情况下，临时变量r1到r2可能对应机器的寄存器，一种容纳算术运算结果的（有限的）机器资源，而我们可能想要极小化所用寄存器的数目。例如，我们可以用指令

```
r7←w+r7
r7←r2-r7
```

代替上面的后两条指令而少使用两个寄存器。表达式的其他部分也可以进行类似的节省。计算一个表达式的值所需要的最少的寄存器的个数是一个具有直接实际意义的树参数。这个量由上所述可以用树的高来定界，但它是相当不同的。(例如，其所有结点除一个结点外都恰好只有一个空链的退化二叉树的高为N，但所对应的表达式却可以用一个寄存器来计算其值。)表达式求值本身就具有相当的重要性，而它在将计算机程序从高级语言变换成机器语言的过程中也体现出树的重要作用。编译器一般首先把程序“分析”成树的表示法，然后再处理树的表示。

习题5.13 为计算图5-7中表达式的值所需要的寄存器的最小数目是多少?

习题5.14 给出对应表达式$((a+b)*d)$和$((a+b)*(d-e)*(f+g))-h*i$的二叉树。再给出对这两棵树的先序、中序和后序遍历。

习题5.15 其操作符所要求的操作数个数变化的表达式对应一棵树，其中操作数在树叶

上，而操作符在非叶结点上。给出对应表达式 $((a^2+b+c)*(d^4-e^2)*(f+g+h))-i*j$ 的树的先序遍历和后序遍历，然后给出这棵树的二叉树表示，并给出所得到的二叉树的先序遍历、中序遍历和后序遍历。

234～235

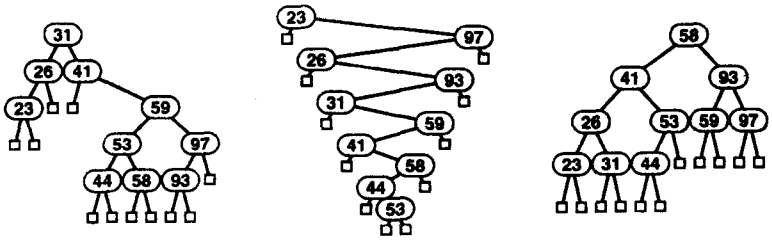
对于这些以及许多其他应用，我们关心树的路径长和高。当然为了考虑这些参数的平均值，我们需要指定一个模型，该模型将定义“随机”树的具体含义。对于树遍历和表达式处理，习惯上使用经验模型，其中所有的树都是等可能出现的。因此，对于这些应用，我们研究所谓的Catalan模型（Catalan models），其中大小为N的T_N棵二叉树或大小为$N+1$的一般的树中的每一棵都以等概率出现。这不是唯一的可能——在许多情况下，树是由外部数据引起，此时其他随机模型是合适的。下面我们考虑一个特别重要的例子。

5.5 二叉查找树

二叉树最重要的应用之一是二叉树搜索算法，一种使用二叉树对在许多应用中产生的基本问题提供有效解决方案的方法。二叉树搜索的分析阐述了所有的树都等可能地出现的模型和其基础分布是非均匀的模型之间的区别。这两个问题并列为本章基本概念之一。

词典、符号表或搜索问题是计算机科学中的基本问题：将一组关键字（或许还有相关的信息）组织起来以便对特定的某些关键字进行有效的查找。在2.6节中讨论的折半查找方法是求解这种问题的一种基本的方法，但是它需要预处理，使所有的关键字首先被排成有序的顺序。

二叉树结构可以用来对词典问题提供更加灵活的解决方案，树的构建是用每个结点上的关键字递增地进行的，构建的时候要保持每个结点的关键字不小于其左子树上的任何关键字同时不大于其右子树上的任何关键字。用这种方法可以给N-结点二叉树上的结点以统一的方式指定任意一组N个互异的关键字。例如，图5-8显示三棵不同的二叉查找树，它们包含一组整数如下：23 26 31 41 44 53 58 59 93 97。



236

图5-8 三棵二叉查找树

定义 二叉查找树就是一棵带有与其内部结点相联系的关键字的二叉树，满足约束：每个结点上的关键字大于或等于它的左子树上的所有关键字并小于或等于它的右子树上的所有关键字。

如果我们假设关键字已经被置于二叉查找树上，那么程序5.2给出词典问题的一个解决方案，它用到一个“搜索”过程的简单递归实现，该过程确定一个给定的关键字是否在这棵二叉查找树上。为了查找一个带有关键字v的结点，如果该树是空树则终止查找（查找不成功），如果根结点上的关键字是v，那么也终止查找（查找成功）。否则，如果v小于根结点上的关键字则查看左子树，如果v大于根结点上的关键字则查看右子树。给定一棵二叉查找树，

程序5.2返回一个包含关键字v的结点当且仅当该树存在这样的结点；否则程序返回NIL。验证这个结论很简单。一般说来，与一组特定的关键字相联系的二叉查找树有很多；对于任意这样的二叉查找树该算法都能够正常工作。

为了构造一棵二叉查找树，我们一个一个地把关键字添加到初始空树中，使用的是递归的策略。为把一个新的结点插入到一棵空树中，创建该结点，并使它的左、右指针为NIL，而且返回指向该结点的指针。（我们使用值NIL表示所有外部结点。）如果树不空，那么，若关键字小于根结点的关键字则把该关键字插入到左子树中；若关键字大于根结点的关键字则把该关键字插入到右子树中。这等价于对该关键字进行一次不成功查找，然后将一个包含该关键字的新结点插入到搜索终止处的外部结点上。程序5.3为该方法的实现。这种实现假设关键字不在树上；另一种实现是混合查找和插入的功能。

程序5.2 二叉树搜索

```
function search(v: integer, x: link): link;
  begin
  if x = NIL search := NIL else
  if v = x^.key then search := x else
  if v < x^.key then search(x^.left)
                 else search(x^.right)
  end;
```

237

由于在两个程序中每次调用search均导致最多一次递归调用，因此开发等价的非递归实现很简单，不需要使用栈。关于二叉查找树实现的更多细节可以在Knuth[22]或Sedgewick[29]或任何一本算法和数据结构的初级教材中都能找到。

例如，假设关键字30要被插入到图5-8左边的树中，那么我们就会创建一个新的结点作为26的右子结点。该树的形状和创建以及用它进行搜索所需要的步数依赖于关键字的插入顺序。例如，如果将关键字按下述顺序

31 26 23 41 59 53 44 58 97 93

插入到初始空树中，则图5-8左边的树就是所得的结果。如果将关键字以顺序

58 41 93 26 53 59 97 23 31 44

插入，那么所得到的树就是右边的结果。在根处的关键字必须首先出现；对于左子树的关键字和右子树的关键字相混但以相同的相对顺序出现的任何排序都将给出相同的树。如果关键字以

23 26 31 41 44 53 58 59 93 97

的顺序插入，那么结果将是一棵无叶的退化树。图5-8中间的树是退化树的另一个例子。

程序5.3 二叉树插入

```
function insert(v: integer, x: link): link
  begin
  if x = NIL then
    begin
    new(x); x^.left := NIL; x^.r := NIL; x^.key := v;
    end;
  else
    begin
    if v < x^.key
        then x^.left := insert(v, x^.left)
        else x^.right := insert(v, x^.right)
    end;
  return x;
  end;
```

忽略相等的关键字，我们可以考虑从其结点上的关键字的一个排列构造一棵树。一般地，许多不同的排列可以映射到同一棵树。图5-9显示在4个元素的排列和4个结点的树之间的映射。因此，例如如果关键字以随机顺序插入到初始空树中，那么每棵树是等可能出现的说法是不成立的。实际上幸运的是，对其进行搜索和构造开销低的更“平衡”的树结构比开销高的树结构更可能出现。在分析中我们将把它们量化。

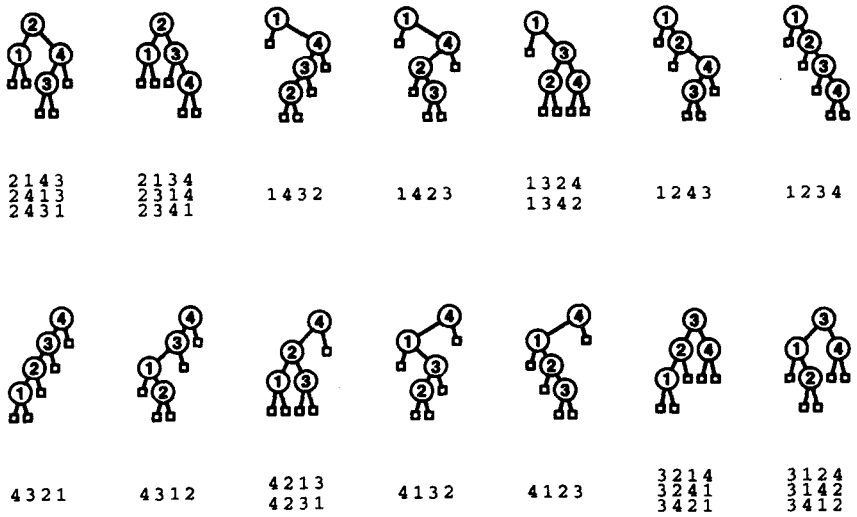


图5-9 具有4个结点的二叉查找树

238~239

有些树构造起来比其他的树要花费昂贵。在最坏的情况下，一棵退化的树，对于1到N之间的每一个i，插入第i个结点要考查$i-1$个内部结点，因此构造一棵树总共需要考查$N(N-1)/2$个结点。在最好的情况下，对于每一棵子树，中间的结点将在根处，在每棵子树中将有大约$N/2$个结点，因此标准的分治递归$T_N = 2T_{N/2} + N$成立，这意味着，构建这样的树总共需要大约$N\lg N$步（见2.6节）。

构造一棵特定的树的开销直接与它的内部路径长成比例，因为结点一旦被插入就不再移动，而结点的层恰好就是将它插入所需要的结点的个数。因此，构建一棵树的开销与能够导出其构建的每个插入序列是相同的。对于所有$N!$个排列，我们可以通过将结果树的内部路径长（累加的开销）相加然后除以$N!$而得到树的构建的平均开销。这个累加的开销也可以通过对所有的树将内部路径长和导出树的构建的排列的个数的乘积相加来计算。这是到一棵初始空树的N次随机插入之后我们期望的平均内部路径长，但是它不同于在所有树都是等可能的Catalan模型下一棵随机二叉树的平均内部路径长。这需要分别地分析，我们将在下一节考虑Catalan树的情形并在5.7节考虑二叉查找树的情形。

许多其他有用的操作在二叉树上很容易定义。例如，中序遍历能够得到排序顺序的关键字，这是在许多应用中附带的有用的收益。对于更详细的处理或实际问题的进一步讨论，可见Knuth[22]或Sedgewick[29]。

习题5.16 给出构建图5-8中每棵树的所有关键字插入序列。

习题5.17 证明，两个不同的关键字插入序列给不出相同的退化的树结构。如果所有$N!$个关键字插入序列是等可能出现的，那么产生退化树结构的概率是多少？

习题5.18 对于$N = 2^n - 1$，如果所有$N!$个关键字插入序列是等可能的，那么建立完美平

衡树结构（所有2^n外部结点均在第n层上）的概率是多少？

习题5.19 如果将5个互异关键字以随机顺序插入到初始空树中，那么哪些5-结点二叉查找树形状是最可能出现的？

习题5.20 证明二叉查找树的先序遍历可以是建立该树的输入关键字序列，就是说，以先序遍历一棵二叉查找树并将那些关键字插入到初始空树中则得到原树。对于后序与/或层序是否上述结论也成立？证明你的答案。

240

5.6 Catalan树中的平均路径长

为了开始对树参数的分析，我们考虑每棵树都是等可能出现的这种模型，为了避免与其他模型相混，我们添加修饰词Catalan表示在该假设下的随机树，因为一棵特定的树出现的概率是Catalan数的倒数。在5.4节提到，这种模型对于诸如表达式求值这样的应用是合理的，并且第3章开发的组合学工具可以直接应用到分析中。

二叉Catalan树。 如果考虑每棵N-结点树都是等可能的，那么具有N个内部结点的二叉树的平均（内部）路径长是多少？我们对这个重要问题的分析是在第3章引入的组合结构参数分析一般处理的原型：

- 定义一个二变量生成函数（BFG），一个变量表示树的大小，另一个变量则表示内部路径长。
- 得出一个由BFG满足的函数方程，或它相关的累积生成函数（CGF）。
- 抽取系数得到结果。

虽然我们从第3章知道直接基于生成函数的论断对于这样的问题通常是可用的，但是我们还是从第2步基于递归的论证开始。我们指出详细的论断（这里）和简单的论断（在下一小段）以进一步强调这个论点。

首先，观察在具有N个结点的随机二叉Catalan树中左子树有k个结点（而右子树有$N-k-1$个结点）的概率为T_kT_{N-k-1}/T_N（其中$T_N=\binom{2N}{N}\Big/(N+1)$是第$N$个Catalan数）。分母是可能的$N$-结点树的棵数而分子是使用左边有$k$个结点且右边有$N-k-1$个结点的任意树做成一棵$N$-结点树的方法数。我们把这个概率分布叫做Catalan分布。

图5-10指出随着N增长时的Catalan分布。关于分布的一个惊人的事实之一是，随着N的增

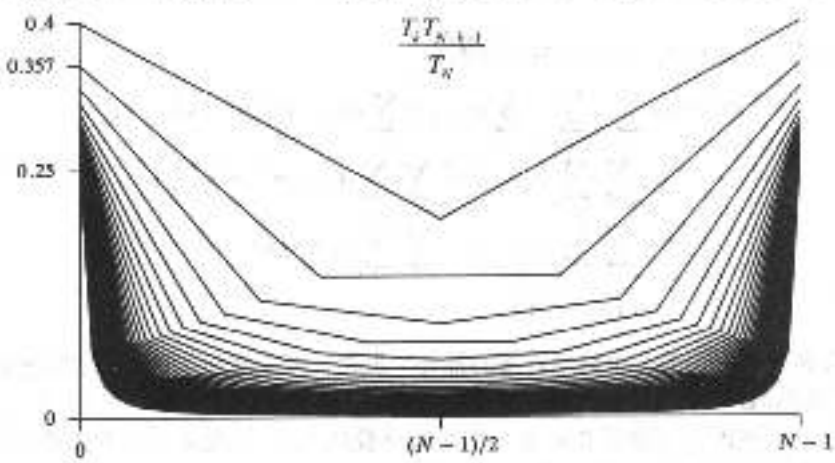


图5-10 Catalan分布（随机二叉树中子树的大小）
（k-轴按N的比例标度）

长，一棵子树是空树的概率趋向于一个常数：它是$2T_{N-1}/T_N \sim 1/2$。实际上，在一棵随机二叉树中大约有一半的结点可能会有一棵空子树，因此这样的树不是平衡得特别好的树。

利用Catalan分布，使用一个非常像我们在Quicksort时研究过的那样的递归，我们可以分析路径长：在随机二叉Catalan树中平均内部路径长由下面的递归描述

$$Q_N = N-1+\sum_{1\le k\le N}\frac{T_{k-1}T_{N-k}}{T_N}(Q_{k-1}+Q_{N-k}) \qquad (N>0)$$

其中$Q_0 = 0$。在这个递归下的论断是一般性的，通过用其他的分布来代替Catalan分布，它也可以在其他随机性模型下用来分析随机二叉树的结构。例如下面的讨论，二叉查找树的分析导致均匀分布（每棵子树大小均以概率$1/N$发生）而递归酷似第1章中的Quicksort递归。

定理5.3 (二叉树中的路径长) 一棵具有N个内部结点的随机二叉树中平均内部路径长为

$$\frac{(N+1)4^N}{\binom{2N}{N}}-3N-1 = N\sqrt{\pi N}-3N+O\left(\sqrt{N}\right)$$

241 ~ 242

证明 我们像在3.12节那样导出一个BGF：首先，概率生成函数$Q_N(u) = \sum_{k>0} q_{Nk}u^k$满足

$$Q_N(u) = u^{N-1}\sum_{1\le k\le N}\frac{T_{k-1}T_{N-k}}{T_N}Q_{k-1}(u)Q_{N-k}(u) \qquad (N>0)$$

其中，q_{Nk}为k是总的内部路径长的概率，$Q_0(u) = 1$。为了简化该递归，我们转到计数处理上，不用概率而使用$p_{Nk} = T_N q_{Nk}$（具有内部路径长k且大小为N的树的棵数）。从上面的递归可知，下式成立

$$\sum_{k\ge 0}p_{Nk}u^k = u^{N-1}\sum_{1\le k\le N}\sum_{r\ge 0}p_{(k-1)r}u^r\sum_{s\ge 0}p_{(N-k)s}u^s \qquad (N>0)$$

为了用二元生成函数

$$P(u,z) = \sum_{N\ge 0}\sum_{k\ge 0}p_{Nk}u^k z^N$$

表示，我们用z^N乘以前面的式子并对N求和得到

$$\begin{aligned}P(u,z) &= \sum_{N\ge 1}\sum_{1\le k\le N}\sum_{r\ge 0}p_{(k-1)r}u^r\sum_{s\ge 0}p_{(N-k)s}u^s z^N u^{N-1}+1\\ &= z\sum_{k\ge 0}\sum_{r\ge 0}p_{kr}u^r(zu)^k\sum_{N\ge k}\sum_{s\ge 0}p_{(N-k)s}u^s(zu)^{N-k}+1\\ &= z\sum_{k\ge 0}\sum_{r\ge 0}p_{kr}u^r(zu)^k\sum_{N\ge 0}\sum_{s\ge 0}p_{Ns}u^s(zu)^N+1\\ &= zP(u,zu)^2+1\end{aligned}$$

下面我们还将看到这个方程的一个简单直接的推导。现在，可以使用定理3.11得出想要的结果：置$u = 1$，得到Catalan数的生成函数的熟知的函数方程，因此$P(1,z) = T(z) = (1-\sqrt{1-4z})/(2z)$。如果我们把所有二叉树的内部路径长相加，那么偏导数$P_u(1,z)$就是累计总和的生成函数。由定理3.11，我们寻求的平均值为$[z^N]P_u(1,z)/[z^N]P(1,z)$。

对BGF函数方程的两边关于u微分（使用偏导数的链式法则）得到

$$P_u(u,z) = 2zP(u,zu)(P_u(u,zu) + zP_z(u,zu))$$ 243

计算该式在$u = 1$的值则给出CGF的函数方程:

$$P_u(1,z) = 2zT(z)(P_u(1,z) + zT'(z))$$

它产生解

$$P_u(1,z) = \frac{2z^2T(z)T'(z)}{1-2zT(z)}$$

现在$T(z) = (1 - \sqrt{1-4z})/(2z)$，因此，$1 - 2zT(z) = \sqrt{1-4z}$以及$zT'(z) = -T(z) + 1/\sqrt{1-4z}$。代入这些式子则得到显式表达式

$$zP_u(1,z) = \frac{z}{1-4z} - \frac{1-z}{\sqrt{1-4z}} + 1$$

将其展开则得到所述结果。 ■

这个结果通过图5-5中的巨大的随机二叉树得以解释：大树渐近地大致符合$\sqrt{N}$乘$\sqrt{N}$平方。

BGF的直接组合学推导。定理5.3的证明涉及二元生成函数

$$P(u,z) = \sum_{N>0}\sum_{k>0} p_{Nk}u^k z^N$$

其中p_{Nk}为具有N个结点及内部路径长为k的树的棵数。从第3章得知，这可以等价地表示成

$$P(u,z) = \sum_{t\in\mathcal{T}} u^{\pi(t)} z^{|t|}$$

此时，5.3节的递归定义直接导出

$$P(u,z) = \sum_{t_l\in\mathcal{T}}\sum_{t_r\in\mathcal{T}} u^{\pi(t_l)+\pi(t_r)+|t_l|+|t_r|} z^{|t_l|+|t_r|+1} + 1$$

结点的个数是1加上子树的结点个数，而内部路径长则是子树中每个结点的内部路径长加1的和。现在，很容易将双重求和重新安排成两个独立的和:

$$\begin{aligned} P(u,z) &= z\sum_{t_l\in\mathcal{T}}(zu)^{|t_l|}u^{\pi(t_l)}\sum_{t_r\in\mathcal{T}}(zu)^{|t_r|}u^{\pi(t_r)} + 1 \\ &= zP(u,zu)^2 + 1 \end{aligned}$$

读者可能希望仔细研究这个例子，体味其中的简明和精妙之处。通过符号方法也能够直接得出这种形式的这些方程（见[13]）。 244

累积生成函数。得到同样结果的另一条途径是直接导出CGF的函数方程，我们定义CGF

$$C_T(z) = P_u(1,z) = \sum_{t\in\mathcal{T}} \pi(t) z^{|t|}$$

其平均路径长为$[z^n]C_T(z)/[z^n]T(z)$。以如上完全相同的方式，二叉树的递归定义直接导致

$$\begin{aligned} C_T(z) &= \sum_{t_l\in\mathcal{T}}\sum_{t_r\in\mathcal{T}}(\pi(t_l) + \pi(t_r) + |t_l| + |t_r|)z^{|t_l|+|t_r|+1} \\ &= 2zC_T(z)T(z) + 2z^2T(z)T'(z) \end{aligned}$$

方程和在定理5.3得到的函数方程相同。

习题5.21 直接从路径长的递归式中导出该结果。

刚刚考虑的三种推导是基于二叉树的相同的组合分解，但是CGF隐藏了大部分的细节，而且它的确是求平均首选的方法。在定理5.3的证明中给出的复杂递归和这里给出的相同结果的“两行”推导之间的反差是典型的，我们将在本书中看到许多其他的问题，它们使用CGF显著地压缩了推导的细节。

一般Catalan树。继续以相同的方式，使用BGF能够找到一般随机树中期望的路径长。

定理5.4（一般树中的路径长） 具有N个内部结点的一般随机树中平均内部路径长为

$$\frac{N}{2}\left(\frac{4^{N-1}}{\binom{2N-2}{N-1}}-1\right)=\frac{N}{2}\left(\sqrt{\pi N}-1\right)+O\left(\sqrt{N}\right)$$

证明 如上进行

$$\begin{aligned}Q(u,z)&=\sum_{t\in\mathcal{G}}\pi(t)z^{|t|}\\&=\sum_{k>0}\sum_{t_1\in\mathcal{G}}\cdots\sum_{t_k\in\mathcal{G}}u^{\pi(t_1)+\cdots+\pi(t_k)+|t_1|+\cdots+|t_k|}z^{|t_1|+\cdots+|t_k|+1}\\&=z\sum_{k>0}Q(u,zu)^k\\&=\frac{z}{1-Q(u,zu)}\end{aligned}$$

245

置$u=1$，我们看到$Q(1,z)=G(z)=zT(z)=(1-\sqrt{1-4z})/2$是Catalan生成函数，我们在5.2节发现该函数对一般树的计数。将上面得到的BGF对u微分并求得$u=1$的值，由此给出CGF

$$C_G(z)=Q_u(1,z)=\frac{zC_G(z)+z^2G'(z)}{(1-G(z))^2}$$

经过简化得到

$$C_G(z)=\frac{1}{2}\frac{z}{1-4z}-\frac{1}{2}\frac{z}{\sqrt{1-4z}}$$

像上面一样，我们使用定理3.11并计算$[z^N]C_G(z)/[z^N]G(z)$，由此直接得出所述的结果。 ■

对于那些尚未完全确信BGF和CGF功效的读者，我们还是期望他们能够完成从递归导出上面结果的习题。

习题5.22 直接证明对于一般树的路径长，定理5.4的证明中对CGF给出的方程是正确的（正如我们对二叉树所做的）。

习题5.23 利用一般树和二叉树之间的旋转对应，从随机二叉树的相应结果中推导出随机一般树的平均路径长。

5.7 二叉查找树中的路径长

在二叉查找树中对路径长的分析实际上不是对树而是对排列的一个性质的研究，因为我们是从随机排列开始的。在第6章，我们将比较详细地讨论排列作为组合对象的性质。我们在

这里考虑BST中路径长的分析不仅因为将其与用刚刚给出的随机树的分析进行比较很有意义，而且还因为我们在第1章和第3章已经做完了所有的工作。

图5-9指出——并且分析证明——二叉查找树插入算法将更多的排列映射到具有小的内部路径长的平衡树要多于具有大的内部路径长的平衡树。因为二叉查找树以统一和灵活的方式提供混合查找、插入和其他操作，所以这种树被广泛地使用。由于查找本身的效率，因此二叉查找树具有重要使用价值。在任何查找算法的开销的分析中，存在两个重要的量：*构造开销*（construction cost）和*查找开销*（search cost），对于后者，通常将两种情形分开考虑比较合适：成功查找和不成功查找。在二叉查找树的情形，这些开销函数和路径长是紧密相关的。 246

构造开销。假设二叉查找树是通过逐步插入，从关键字的资源中随机提取（例如，在0和1之间独立和均匀地分布的随机数）而建成的。这意味着所有$N!$种关键字排序都是等可能的，因此与假设关键字是整数1到N的一个随机排列等价。现在观察到，这些树是通过一个分裂过程形成的：第一个被插入的关键字成为根处的结点，然后，左右子树独立地建立。N个关键字中第k个最小的关键字在根处的概率为$1/N$（与k无关），在这种情况下，大小为$k-1$和$N-k$的两棵子树分别建在左边和右边。建立这些子树总的开销对每个结点（共计$k-1+N-k=N-1$）而言比子树在根处大1，因此我们得到递归

$$C_N = N-1+\frac{1}{N}\sum_{1\le k\le N}(C_{k-1}+C_{N-k}) \qquad (N>0,\ C_0=0)$$

当然，在5.5节提到，这个递归还描述了二叉查找树的平均内部路径长。此外，这个递归是在第1章中由Quicksort算法使用的比较次数所解得的递归，不过这里用的是$N-1$而不是$N+1$。

这样，我们实际上已经在1.5节和3.12节做了构造二叉查找树的开销的分析。

定理5.5 (BST的构造开销) *在通过以随机顺序将N个不同的关键字插入到初始空树中的方法构造一棵二叉查找树的过程所涉及的平均比较次数（一棵随机二叉查找树的平均内部路径长）为*

$$2(N+1)(H_{N+1}-1)-2N \approx 1.386N\lg N-2.846N$$

其方差渐近接近于$(7-2\pi^2/3)N^2$。

证明 从上面的讨论可知，关于平均的解直接从定理1.2的证明和讨论中推出。方差的推导和定理3.12的证明相同。定义BGF 247

$$Q(u,z)=\sum_{p\in\mathcal{P}}\frac{u^{\xi(p)}z^{|p|}}{|p|!}$$

其中$\mathcal{P}$表示所有排列的集合，$\xi(p)$表示当p的那些元素用标准算法被插入到初始空树中所构造的二叉查找树的内部路径长。通过实际上与3.12节相同的计算，这棵BGF必然满足函数方程

$$\frac{\partial}{\partial z}Q(u,z)=Q^2(u,zu) \qquad (Q(u,0)=1)$$

它与对应的Quicksort的方程的区别仅在于它缺少一个u^2因子（起源于在递归中$N+1$和$N-1$的差别）。方差的计算正如3.12节中所做的，得到的结果完全相同（u^2因子不对方差构成影响）。 ■

这样，一棵“随机”二叉查找树只比完美平衡树多耗费40%的开销。图5-11显示一棵大的随机二叉查找树，通过与图5-5比较，该树平衡得相当好，它是在所有的树都是等可能的假设下的一棵“随机”树。

图5-11 从256个随机排序的关键字构建的二叉查找树

与Quicksort递归的关系突出了在算法分析中研究树的重要性的一个基本的原因：递归程序涉及树的固有结构。例如，图5-8左边的树也可以看作是使用程序1.2对关键字排序的过程的精确描述：我们把在根处的关键字看成是分隔元；左子树看作是对左子文件排序的表述；而右子树为右子文件排序的表述。二叉树也可以用来描述Mergesort的操作，其他类型的树隐含在其他递归程序的操作中。

248

习题5.24 对于图5-8中的每一棵树，给出那些引起程序1.2如树所描述的划分的排列。

查找开销。一次成功的查找就是找到以前插入的关键字的查找。我们假设，树中的每个关键字都是等可能被查找的。一次不成功查找是对前面尚未被插入的关键字的查找。就是说，所寻找的关键字不在树上，因此查找终止于外部结点。我们假设，每一个外部结点都是等可能地达到的，例如，在我们的模型中每次查找的情况就是这样，其中新的结点从随机资源中抽出。

我们想要分析在树中（除构造之外）查找的开销。这在应用中很重要，因为通常我们认为在大量的查找操作中将会涉及树，而对于许多的应用来说，构造的开销相比于查找开销就不那么重要了。为了进行分析，我们采纳概率模型，即树是由随机插入建成的，而在树中的查找是“随机”的，正如前面一段所描述的。这两种开销都直接与路径长相关。

定理5.6（在BST中的查找开销） *在一棵N个结点的随机二叉查找树中，成功查找的平均开销为$2H_N - 3 - 2H_N/N$，而不成功查找的平均开销为$2H_{N+1} - 2$。在这两种情况下，方差均为$2H_N + O(1)$。*

证明 在树中找到一个关键字所需要的比较次数恰好是插入该关键字所需要的次数加1，因为关键字在树中从来不移动。因此，成功查找的结果通过用N去除构建该树的开销（内部路径长，在定理5.5中给出）并加1而得到。

由于外部结点的层恰好就是在一次不成功查找期间达到它的开销，因此一次不成功查找的平均开销恰好是用$N + 1$除外部路径长，因此所述结果直接从5.3节的第一个引理和定理5.5得出。

249

方差需要不同类型的计算，我们将在下面讨论。 ■

利用PGF的分析。定理5.6的证明是前面得到的给出平均开销结果的方便的应用；不过，它并没有给出一种计算标准差的方法。

这是因为概率模型间的差别。对于内部路径长（构造开销），存在$N!$种不同的可能要考虑，而对于成功查找的开销，却存在$N \cdot N!$种可能。内部路径长是一个量，这个量大致在$N\lg N$和N^2之间变化，而成功查找开销在1和N之间变化。对于一棵特定的树，我们通过用N除内部路径长得到平均成功查找开销，但是刻画出查找开销的分布的特征却是另外一回事。例如，成功查找开销为1的概率是$1/N$，它根本与内部路径长为N的概率没有关系，当$N > 1$时这个概率是0。

习题5.25 成功查找开销为2的概率是多少？

习题5.26 通过插入1000个随机关键字到一棵初始空树来构造1000结点的随机二叉查找树，然后在这棵树中进行10 000次随机查找并画出查找开销得到的柱状图，与图1-4进行比较。

习题5.27 做前面的习题，但每次试验生成一棵新的树。

概率生成函数（PGF，见3.11节）提供另一种推导查找开销的方法，并且还能够对概率中的矩进行计算。例如，一次不成功查找的开销的PGF满足

$$p_N(u)=\left(\frac{N-1}{N+1}+\frac{2u}{N+1}\right)p_{N-1}(u)$$

因为如果查找终止在两个外部结点中的一个上，它以概率$2/(N+1)$发生，那么第N次插入使一次不成功查找的开销增1；否则增0。在1处微分并求值便得到求平均的一个简单递归，叠缩求和直接得到定理5.6的结果，而方差以类似的方式得到。这些计算可以总结为下面的习题。

习题5.28 [Lynch, cf. Knuth] 通过计算$p''_N(1)+p'_N(1)-p'(1)^2$，证明不成功查找开销的方差为$2H_{N+1}-4H^{(2)}_{N+1}+2$。

习题5.29 [Knott, cf. Knuth]利用PGF进行直接论证，求一次成功查找的开销的平均值和方差。

习题5.30 用不成功查找的PGF表示成功查找的PGF。利用这个结果，通过不成功查找的平均和方差表示成功查找的平均和方差。

250

5.8 随机树的可加参数

我们上面用来分析Catalan树和二叉查找树路径长的基于CGF的方法可以推广到包括一大类参数，它们可以对子树可加地定义。特别地，定义可加参数为任意一个其开销函数满足线性递归

$$c(t)=e(t)+\sum_s c(s)$$

的参数，其中的求和是对t的根的所有子树进行。这里的函数e叫做“费用”，它是与根相关的开销的一部分。下表给出开销函数和相关费用的几个例子:

费用函数$e(t)$	开销函数$c(t)$
1	大小$\|t\|$
$\|t\|-1$	内部路径长
$\delta_{\|t\|1}$	树叶数

通常我们将空二叉树的费用函数取为0。

对于Catalan树模型和BST模型的任意可加参数的平均情形分析能够得出一种全面的一般处理。事实上，它包含了关于树的性质的所有定理。

定理5.7（随机树中的可加参数） 令$C_T(z)$、$C_G(z)$或$C_B(z)$分别是二叉Catalan树、一般Catalan树和二叉查找树等模型的可加树参数$c(t)$的CGF，并令$E_T(z)$、$E_G(z)$和$E_B(z)$为相关费用函数$e(t)$的CGF。（对于二叉查找树的情形，是用指数的CGF）。这些函数通过下述方程相关

$$C_T(z)=\frac{E_T(z)}{\sqrt{1-4z}}\quad（二叉Catalan树）$$

$$C_G(z)=\frac{1}{2}E_G(z)\left(1+\frac{1}{\sqrt{1-4z}}\right)\quad（一般Catalan树）$$

$$C_B(z)=\frac{1}{(1-z)^2}\left(E_B(0)+\int_0^z(1-x)^2E_B'(x)\mathrm{d}x\right) \qquad (二叉查找树)$$

251

证明 证明直接从我们对路径长给出的论证得出。

首先，令$\mathcal{T}$为所有二叉Catalan树的集合。此时，正如5.6节所述，我们有

$$\begin{aligned}C_T(z)&=\sum_{t\in\mathcal{T}}c(t)z^{|t|}\\&=\sum_{t\in\mathcal{T}}e(t)z^{|t|}+\sum_{t_l\in\mathcal{T}}\sum_{t_r\in\mathcal{T}}(c(t_l)+c(t_r))z^{|t_l|+|t_r|+1}\\&=E_T(z)+2zT(z)C_T(z)\end{aligned}$$

其中 $T(z)=(1-\sqrt{1-4z})/(2z)$ 为Catalan数T_N的OGF。这直接导出所述的结果。

其次，对于一般Catalan树，令$\mathcal{G}$为树的集合。也如5.6节所述，我们有

$$\begin{aligned}C_G(z)&=\sum_{t\in\mathcal{G}}c(t)z^{|t|}\\&=\sum_{t\in\mathcal{G}}e(t)z^{|t|}+\sum_{k>0}\sum_{t_1\in\mathcal{G}}\cdots\sum_{t_k\in\mathcal{G}}(c(t_1)+\cdots+c(t_k))z^{|t_1|+\cdots+|t_k|+1}\\&=E_G(z)+z\sum_{k>0}kC_G(z)G^{k-1}(z)\\&=E_G(z)+\frac{zC_G(z)}{(1-G(z))^2}\end{aligned}$$

其中 $G(z)=zT(z)=(1-\sqrt{1-4z})/2$，Catalan数$T_{N-1}$的常规生成函数OGF，是对一般树的计数。将其代入并化简，同样导出所述结果。

对于二叉查找树，我们令c_N和e_N分别表示大小为N的随机BST的$c(t)$和$e(t)$的期望值。此时，指数CGF $C(z)$和$E(z)$与这些序列的OGF相同，我们按照3.3节的推导得出递归

$$c_N=e_N+\frac{2}{N}\sum_{1\le k\le N}c_{k-1} \qquad (N\ge 1,\ c_0=e_0)$$

它导出微分方程

$$C_B'(z)=E_B'(z)+2\frac{C_B(z)}{1-z} \qquad (C_B(0)=E_B(0))$$

252 我们完全像在3.3节那样求解该方程，得到定理所述的解。 ■

推论 上述可加参数的平均值由

$$[z^N]C_T(z)/T_N \qquad (二叉Catalan树)$$
$$[z^N]C_G(z)/T_{N-1} \qquad (一般Catalan树)$$
$$[z^N]C_B(z) \qquad (二叉查找树)$$

给出。

证明 这些结果直接从定义和定理3.11得出。 ■

这极大地推广了我们做过的计数和路径长分析，并使我们能够分析许多重要的参数。本章较早一些定理中得到的计数和路径长的结果都可以从这个定理的简单应用得到。例如，为了计算二叉Catalan树中的平均路径长，我们有

$$E_T(z)=1+\sum_{t\in T}(|t|-1)z^{|t|}=1+zT'(z)-T(z)$$

因此

$$C_T(z)=\frac{zT'(z)-T(z)+1}{\sqrt{1-4z}}$$

它等价于定理5.3的证明中得到的表达式。

树叶。作为定理5.7对于一个新问题使用的例子，我们考虑上述三种模型中每一种模型的树叶平均数的分析。这是与递归结构内存分配相关的一类重要问题的代表。例如，在一棵二叉树中，如果空间非常宝贵，那么我们不需要在树叶上保留指针：取代的方法是把它们标记作树叶（用一个比特位，而不是指针），使用所谓的变体记录。用这种方法节省多少空间？这个问题的答案依赖于树模型：确定树叶的平均数是定理5.7的一个直接应用，使用$e(t)=\delta_{|t|1}$，因此$E_T(z)=E_G(z)=E_B(z)=z$。

第一，对于二叉Catalan树，我们有$C_T(z)=z/\sqrt{1-4z}$。这正好匹配3.12节中得到的结果。

第二，对于一般Catalan树我们有

$$C_G(z)=\frac{z}{2}+\frac{z}{2\sqrt{1-4z}}$$

它导出对于$N>1$平均值恰好是$N/2$的结果。 253

第三，对于二叉查找树，我们得到

$$C_B(z)=\frac{1}{3}\frac{1}{(1-z)^2}+\frac{1}{3}(z-1)$$

因此对于$N>1$树叶的平均数为$(N+1)/3$。

推论 对于$N>1$，树叶的平均数由下面三个式子给出：

$\frac{N(N+1)}{2(2N-1)}\sim\frac{N}{4}$	在具有N个结点的随机二叉Catalan树中
$\frac{N}{2}$	在具有N个结点的随机一般Catalan树中
$\frac{N+1}{3}$	在由N个随机关键字建立的二叉查找树中

证明 参见上面的讨论。 ■

这些方法在分析涉及树的算法中显然是相当有用的，而且它们在某些其他情形下也是适用的。例如，在第6章中，通过与树的对应（见6.5节），我们还用到这些方法分析排列的性质。

习题5.31 求在具有N个结点的随机Catalan树中根的子结点的平均个数（从图5-3可知，对$N=5$时答案为2）。

习题5.32 在具有N个结点的随机Catalan树中，求具有一个子结点的结点的比率。

习题5.33 在具有N个结点的随机Catalan树中，对于$k=2$、3以及更高的值，求具有k个子结点的结点的比率。

习题5.34 二叉树中的内部结点呈现三种类型之一：它们或者有2个、或者有1个、或者有0个外部的子结点。在具有N个结点的随机二叉Catalan树中，每种类型的结点各占多大部分？

习题5.35 对于随机二叉查找树回答前面的问题。

习题5.36 建立关于树叶数目的BGF，并且估计这三种随机树模型中每个的方差。

254 **习题5.37** 证明这些关系类似于定理5.7中关于BGF的那些关系。

5.9 高

一棵树的平均高是多少？路径长分析（使用5.3节中的第2个引理）提出对于Catalan（二叉或一般）树$N^{1/2}$阶和$N^{3/4}$阶的下界和上界以及对于二叉查找树的$\log N$阶和$\sqrt{N\log N}$阶的下界和上界。得到平均高的更精确的估计实际上是一个更难回答的问题，尽管高的递归定义跟路径长的递归定义同样的简单。树的高等于1加上各子树的高的最大值，而树的路径长则是1加上各子树的路径长的和再加上各子树中的结点数。我们已经看到，后者的分解可以对应从子树“构建”树，而可加性是（通过开销GF方程的线性性）在分析中得到的反应。这样的处理均不适用对子树取最大值的操作。

二叉Catalan树的生成函数。我们从计算二叉Catalan树高的问题开始。为了按照对路径长的方式进行，我们开始考虑二元生成函数

$$P(u,z)=\sum_{N\geq 0}\sum_{h\geq 0}P_{Nh}u^h z^N=\sum_{t\in\mathcal{T}}u^{\eta(t)}z^{|t|}$$

现在，高的递归定义导出下面结果

$$P(u,z)=\sum_{t_l\in\mathcal{T}}\sum_{t_r\in\mathcal{T}}u^{\max(\eta(t_l),\eta(t_r))}z^{|t_l|+|t_r|+1}$$

对于路径长，本来是能够重新整理成一些独立的和的，但是“max”妨碍了这么做。

另一方面，使用在3.12节所描述的二元序列的“垂直”公式表示，我们可以得到一个简单的函数方程。令$\mathcal{T}_h$为树高不大于h的二叉Catalan树的集合，并有

$$T^{[h]}(z)=\sum_{t\in\mathcal{T}_h}z^{|t|}$$

按照与计数相同的方式进行，则可得到$T^{[h]}(z)$的一个简单函数方程:其高不大于$h+1$的任何树或者是空树，或者是一个根结点和两棵其高不大于h的子树，于是

$$\begin{aligned}T^{[h+1]}(z)&=1+\sum_{t_L\in\mathcal{T}_h}\sum_{t_R\in\mathcal{T}_h}z^{|t_L|+|t_R|+1}\\&=1+zT^{[h]}(z)^2\end{aligned}$$

255

这个结果通过符号方法也容易得到：它对应符号方程

$$\mathcal{T}_{h+1}=\{\square\}+\{\bullet\}\times\mathcal{T}_h\times\mathcal{T}_h$$

迭代该递归，得到

$$\begin{aligned}T^{[0]}(z)&=1\\T^{[1]}(z)&=1+z\\T^{[2]}(z)&=1+z+2z^2+z^3\\T^{[3]}(z)&=1+z+2z^2+5z^3+6z^4+6z^5+4z^6+z^7\\&\vdots\\T^{[\infty]}(z)&=1+z+2z^2+5z^3+14z^4+42z^5+132z^6+\cdots=T(z)\end{aligned}$$

读者可能发现对于图3-1中给出的那些小树的初始值验证这些公式是有助于理解的。现在，定理3.11的推论告诉我们，累计开销（N个结点的所有的树的高的和）由

$$[z^N]\sum_{h>0}(T(z)-T^{[h]}(z))$$

给出。不过我们现在的分析任务要困难的多，我们不是去估计那种已经有一个确定的函数方程的函数的展开式中的系数，我们需要的是估计由那些相关函数方程定义的一系列函数展开式的系数。这对于该特定问题实际上是一个非常艰巨的任务。

定理5.8（二叉树的高） 具有N个结点的随机二叉Catalan树平均高为$2\sqrt{\pi N}+O(N^{1/4+\varepsilon})$，其中$\varepsilon>0$是任意实数。

证明 忽略证明，不过我们可以看到上面的评述。细节可以在Flajolet和Odlyzko[10]中找到。■

Catalan树的平均高。对于一般Catalan树，确定平均高的问题仍然比分析路径长要困难得多，不过我们可以简述解法。（警告：该简述涉及从第2章到第4章许多先进方法的结合，应该提醒读者谨慎处理。） 256

首先，我们通过

$$\mathcal{G}_{h+1}=\{\bullet\}\times(\varepsilon+\mathcal{G}_h+(\mathcal{G}_h\times\mathcal{G}_h)+(\mathcal{G}_h\times\mathcal{G}_h\times\mathcal{G}_h)+(\mathcal{G}_h\times\mathcal{G}_h\times\mathcal{G}_h\times\mathcal{G}_h)+\cdots)$$

构造高$\leq h+1$的树的集合$\mathcal{G}_{h+1}$，上式由符号方法翻译成

$$G^{[h+1]}(z)=z(1+G^{[h]}(z)+G^{[h]}(z)^2+G^{[h]}(z)^3+G^{[h]}(z)^4+\cdots)=\frac{z}{1-G^{[h]}(z)}$$

迭代该递归，我们看到

$$\begin{aligned}
G^{[0]}(z)&=z\\
G^{[1]}(z)&=z\frac{1}{1-z}\\
G^{[2]}(z)&=\frac{z}{1-\frac{z}{1-z}}=z\frac{1-z}{1-2z}\\
G^{[3]}(z)&=\frac{z}{1-\frac{z}{1-\frac{z}{1-z}}}=z\frac{1-2z}{1-3z+z^2}\\
&\vdots\\
G^{[\infty]}(z)&=z+z^2+2z^3+5z^3+14z^5+42z^6+132z^7+\cdots=zT(z)
\end{aligned}$$

这些有理函数具有足够的代数结构，使得我们能够得到关于高的精确计数以及获取渐近估计。

定理5.9 (Catalan树高GF) 具有$N+1$个结点且其高大于或等于$h-1$的Catalan树的数目为

$$G_{N+1}-G_{N+1}^{[h-2]}=\sum_{k\geq 1}\left(\binom{2N}{N+1-kh}-2\binom{2N}{N-kh}+\binom{2N}{N-1-kh}\right)$$

证明 从基本递归和上面给出的初始值不难验证，$G^{[h]}(z)$可以表示成$G^{[h]}(z)=zF_{h+1}(z)/F_{h+2}(z)$ 257

的形式，其中$F_h(z)$为满足递归

$$F_{h+2}(z)=F_{h+1}(z)-zF_h(z) \qquad (h\geqslant 0, F_0(z)=0, F_1(z)=1)$$

的多项式

$$\begin{aligned}
F_0(z)&=0\\
F_1(z)&=1\\
F_2(z)&=1\\
F_3(z)&=1-z\\
F_4(z)&=1-2z\\
F_5(z)&=1-3z+z^2\\
F_6(z)&=1-4z+3z^2\\
F_7(z)&=1-5z+6z^2-z^3\\
&\vdots
\end{aligned}$$

族。这些函数有时叫做*斐波那契多项式*，因为它们推广了斐波那契数，后者即$z=-1$时斐波那契多项式的值。

当z固定时，上述斐波那契多项式递归是简单的常系数线性递归（见2.4节）。于是，它的解可以表示成特征方程$y^2-y+z=0$的解

$$\beta=\frac{1+\sqrt{1-4z}}{2} \quad 和 \quad \hat{\beta}=\frac{1-\sqrt{1-4z}}{2}$$

正如2.4节对斐波那契数所做的那样，精确求解则得到

$$F_h(z)=\frac{\beta^h-\hat{\beta}^h}{\beta-\hat{\beta}} \qquad 因此 \qquad G^{[h]}(z)=\frac{\beta^{h+1}-\hat{\beta}^{h+1}}{\beta^{h+2}-\hat{\beta}^{h+2}}$$

注意，这些根是与CatalanGF紧密相关的:

$$\hat{\beta}=G(z)=zT(z) \quad 和 \quad \beta=z/\hat{\beta}=z/G(z)=1/T(z)$$

此外，我们还有恒等式

258

$$z=\beta(1-\beta)=\hat{\beta}(1-\hat{\beta})$$

总而言之，具有有界高的树的GF满足公式

$$G^{[h]}(z)=2z\frac{(1+\sqrt{1-4z})^{h+1}-(1-\sqrt{1-4z})^{h+1}}{(1+\sqrt{1-4z})^{h+2}-(1-\sqrt{1-4z})^{h+2}}$$

且只需少许代数运算即可得到

$$G(z)-G^{[h]}(z)=\sqrt{1-4z}\,\frac{u^{h+2}}{1-u^{h+2}}$$

其中，$u=\hat{\beta}/\beta=G^2(z)/z$。这是$G(z)$的一个函数，它间接地由$z=G(z)(1-G(z))$定义，因此拉格朗日反演定理（3.10节中的定理3.9）适用，于是（经过一些计算）推出对于$[z^{N+1}](G(z)-G^{[h-2]}(z))$所述的结果。 ■

推论 *具有N个结点的随机Catalan树的平均高为$\sqrt{\pi N}+O(1)$。*

证明概要 根据定理3.11的推论，平均高由下式

$$\sum_{h\geq 1}\frac{[z^N](G(z)-G^{[h-1]}(z))}{G_N}$$

给出。由定理5.9，它可以化成三个和，它们很像Catalan和，并且可以用类似于定理4.9的证明方式处理。从二项式系数尾部的渐近结果(定理4.6的推论)可知，对于大的h这些项指数上是小的。应用定理5.9二项式和中每一项尾部的界，对于 $h>\sqrt{N}\log N$ 我们有

$$[z^N](G(z)-G^{[h-1]}(z))=O(N4^N e^{-(\log^2 N)})$$

这已经指出，期望的高本身为$O(N^{1/2}\log N)$。

对于h的更小的值，定理4.6的正态近似效果很好。正如我们在定理4.9的证明中所做的，使用逐项近似能够证明

$$\frac{[z^N](G(z)-G^{[h-1]}(z))}{G_N}\sim H(h/\sqrt{N})$$

259

其中，

$$H(x)\equiv\sum_{k\geq 1}(4k^2x^2-2)e^{-k^2x^2}$$

正像图4-7中所画的trie和，函数 $H(h/\sqrt{N})$ 当h小的时候接近于1而当h大的时候接近于0，当h接近 $\sqrt{N}$ 的时候将发生从1到0的转变。此时，通过欧拉-麦克劳林求和以及简单的积分求值，期望的高近似地表示为

$$\sum_{h\geq 1}H(h/\sqrt{N})\sim\sqrt{N}\int_0^\infty H(x)\mathrm{d}x\sim\sqrt{\pi N}$$

在最后的几步，我们忽略了误差项，这些项必须适当地保持一致。按照这种问题通常的做法，这样处理是不困难的，因为他们的尾部在指数上是小的，不过我们把细节留作下面的一些习题。对于证明该结果的相关但不同的处理的全部细节在De Bruijn, Knuth和Rice[7]中给出。 ■

这个分析是本书在处理中最困难的问题。它将求解线性递归和连分式（第2章）的技巧、生成函数展开，特别是通过拉格朗日反演定理（第3章）并与二项式逼近和欧拉-麦克劳林求和（第4章）结合起来。当我们认识到许多读者可能不期望不经过非常仔细的研究而深究这种范畴和复杂程度的证明的时候，我们还是给出了某些推导的细节，因为对于高的分析在理解树的性质方面是极其重要的。这种概述使我们认识到：（i）分析树的高不是一件容易的工作，不过（ii）使用我们在第2章到第4章讨论的基本方法是能够做到这种分析的。

习题5.38 证明 $F_{h+1}(z)=\sum_j\binom{h-j}{j}(-z)^j$ 。

习题5.39 指出使用拉格朗日反演定理展开$G(z)-G^{[h-2]}(z)$的展开细节。

习题5.40 提供书中推论的详细证明，包括适当考虑误差项。

习题5.41 画出函数$H(x)$的图。

二叉查找树的高。对于从随机排列建立的二叉查找树，求其平均高的问题也是相当困难的。由于平均路径长为$O(N\log N)$，因此我们期望二叉查找树的平均高是~$c\log N$，其中c是某个常数；实际上情况正是如此。 260

定理5.10（二叉查找树的高） 从N个随机关键字所建立的二叉查找树期望的高为$\sim c\log N$，其中$c \approx 4.31107\cdots$为$c\ln(2e/c)=1$当$c>2$的解。

证明 从略，可见Devroye[8]或Mahmoud[24]。 ■

虽然完全的证明至少像上面给出的Catalan树高的分析一样地令人望而生畏，但是推导生成函数间的函数关系还是容易做到的。令$q_N^{[h]}$为由N个随机关键字建立的其高不大于h的BST的概率。此时，使用通常的分裂论证并注意到子树的高不大于$h-1$，则有下列递归

$$q_N^{[h]}=\frac{1}{N}\sum_{1\le k\le N} q_{k-1}^{[h-1]}q_{N-1-k}^{[h-1]}$$

它立即导出下面的结果

$$\frac{\mathrm{d}}{\mathrm{d}z}q^{[h]}(z)=(q^{[h-1]}(z))^2$$

栈高。在算法分析中经常出现树的高。基本上，它不仅度量遍历树所需要的栈的大小，而且还度量当递归程序执行时所使用的空间。例如，在前面讨论过的表达式赋值算法中，树高$\eta(t)$度量当由t代表的表达式被求值时由递归栈所达到的最大深度。类似地，一棵二叉查找树的高度度量了为给关键字排序而由递归中序遍历所达到的最大栈深度，或使用递归Quicksort的实现时所用隐式栈的深度。

树的遍历和其他递归算法也可以不用递归而通过保留一个下推栈（后进先出数据结构）来实现。当存在多于一棵子树要访问的时候，除一棵外我们把所有的存储在栈上；当没有子树要访问的时候，我们弹出栈而得到一棵要访问的树。这样使用的栈项比支持递归实现的栈中所需要的项少，因为如果只有一棵子树要访问，则没有项需要进栈。（一种称为终端递归移除（end recursion removal）的技巧有时用于获得递归实现的等价性能。）当一棵树用这种方法遍历时所需要的最大栈大小是叫做栈高（stack height）的树参数，它类似于高。它可以通过递推公式定义：

261

$$s(t)=\begin{cases}0, & \text{如果}t\text{为一外部结点}\\ s(t_l), & \text{如果}t_r\text{为一外部结点}\\ s(t_r), & \text{如果}t_l\text{为一外部结点}\\ 1+\max(s(t_l),s(t_r)) & \text{其他情况}\end{cases}$$

由于旋转对应，事实上二叉Catalan树的栈高本质上像一般Catalan树高那样分布。因此，二叉Catalan树的平均栈高也在De Bruijn、Knuth和Rice[7]中讨论并证明为$\sim\sqrt{\pi N}$。

习题5.42 找出二叉树的栈高和对应森林的高之间的关系。

寄存器分配。当树表示一个算法表达式的时候，计算表达式值所需的寄存器的最小个数可以由下面的递归公式描述：

$$r(t)=\begin{cases}0, & \text{如果}t\text{为一外部结点}\\ r(t_l), & \text{如果}t_r\text{为一外部结点}\\ r(t_r), & \text{如果}t_l\text{为一外部结点}\\ 1+r(t_l), & \text{如果}r(t_l)=r(t_r);\\ \max(r(t_l),r(t_r)) & \text{其他情况}\end{cases}$$

这个量的平均值由Flajolet、Raoult和Vuillemin[12]以及Kemp[20]研究。虽然这个递归看起来

颇似高和栈高的对应递归，但是它的解却不是 $O(\sqrt{N})$，而是 ~ (lg N)/2。

5.10 树性质平均情形结果的小结

我们已经讨论了三种不同的树结构（二叉树、树、二叉查找树）和两个基本参数（路径长和高），总共给出六个定理描述在这些结构中的这两个参数的平均值。这些结果中的每一个都是基本的，并且值得相互联系地考虑它们。在5.8节指出，至于这些参数的基本解析方法可以扩展到包括树的许多的性质，我们可以通过考查这些参数和树模型之间的关系而把一些新的问题放到适当的地方。同时，我们简述这些结果的历史，它们被总结在表5-1和表5-2中。本节为了简洁，把二叉Catalan树直接叫做“二叉树”，Catalan树叫做“树”，二叉查找树叫做“BST”，我们认识到，在已经讨论的一系列分析中的主要目标一直是讨论术语上的区别和确定相关随机性模型上的量的差别。 262

图5-6、5-5和5-11分别显示一片随机森林（不包括根的随机树）、二叉树和二叉查找树。它们加强了表5-1和表5-2给出的分析信息：二叉树和树的高是类似的（与 $\sqrt{N}$ 成比例），树的高大约是二叉树高的一半；而二叉查找树中的路径要短得多（与log$N$成比例）。施加到二叉查找树结构上的概率分布对于具有短路径的树是有偏的。

也许最容易的问题是二叉查找树中的路径长分析，它容易用初等方法得到，这可以回溯到1960年Quicksort的发现[19]。树的构造开销的方差（与Quicksort的方差相同）显然由Knuth[22]首先发表；Knuth指出，描述方差和查找开销结果的递推关系是在1960年被发现的。通过对比，二叉查找树平均高的分析是相当困难的问题，并且是Devroye在1986[8][9]列出的要完成的表上的最后一个问题。

表5-1 树的期望路径长

	GF的函数方程	$[z^N]$的渐近估计
树	$Q(u,z)=\dfrac{z}{1-Q(z,zu)}$	$\dfrac{N}{2}\sqrt{\pi N}-\dfrac{N}{2}+O(\sqrt{N})$
二叉树	$Q(u,z)=zQ(u,zu)^2+1$	$N\sqrt{\pi N}-3N+O(\sqrt{N})$
BST	$\dfrac{\partial}{\partial z}Q(u,z)=Q(z,zu)^2$	$2N\ln N+(2\gamma-4)N+O(\log N)$

263

随机树和随机二叉树中的路径长分析起来也不困难，不过使用基于生成函数的或符号组合的工具是最佳的处理方法。使用这样一种处理，该参数（以及其他可加参数）的分析比计数难不了太多。

树高在基于树的计算机程序和递归程序的分析中的中心角色由于它们的广泛使用而是显而易见的，但是同样明显的是，像高这种树中的非可加参数的分析能够带来重大的技术挑战。树高（以及关于二叉树的栈高）的分析——由De Bruijn、Knuth和Rice[7]于1972年发表——证明了这样的挑战通过使用一些已知的分析技巧是能够被克服的，我们在5.9节已经概述过了。不过，企图沿着这些方向获取新的结果即使是对专家来说恐怕也是令人望而生畏的工作。例如，二叉树高的分析直到1982年才被Flajolet和Odlyzko[10]完成。

随机树中的路径长和高的分析是值得仔细研究的，因为它们阐述了生成函数的威力，以及那种适用于递归结构中“可加”和“非可加”参数的分析中的对比模式。我们在下一节将看到，树直接与概率和组合数学中许多经典问题相关，因此，我们考虑的某些问题具有非凡

的传统，这可以向前追溯到一或两个世纪。但是，为路径长和高研发精确的渐近结果的动机自然可以归结为树在算法分析中的重要性（见Knuth[7][21][22]）。

表5-2 树的期望高

	GF的函数方程	均值的渐近估计
树	$q^{[h+1]}(z)=\dfrac{z}{1-q^{[h]}(z)}$	$\sqrt{\pi N}+O(1)$
二叉树	$q^{[h+1]}(z)=z(q^{[h]}(z))^2+1$	$2\sqrt{\pi N}+O(N^{1/4+\epsilon})$
BST	$\dfrac{\mathrm{d}}{\mathrm{d}z}q^{[h+1]}(z)=(q^{[h]}(z))^2$	$(4.3110\cdots)\ln N+o(\log N)$

264

5.11 树和二叉树的表示

由于旋转对应，树和二叉树可以看作是表示相同组合学对象（它们可以用Catalan数计数）的两种特殊的方式。还出现一些表示它们的其他方法（例如，可见Read[28]），其中的一部分在这里摘述。

括号系统。每片具有N个结点的森林均对应一组N对括号：由定义可知，存在一系列的树，每棵树都由一个根和具有相同结构的一些子树组成。如果我们认为，每棵树都应该放入括号中，那么我们就立即得到只使用括号的一种表示法：对于森林中的每一棵树，写一个左括号，接着是包含这些（递归确定的）子树的森林的括号系统，后面跟着一个右括号。例如，图5-4左边的森林可以由

(((()) () ())) () (() ())) (())

表示。在这种表示和树结构之间的关系通过在对应这些括号括起的树的根结点的各层书写括号是容易看清的，做法如下：

```
(                    ) () (       ) (   )
   (   ) () ()            () ()     ()
      ()
```

将这种结构套叠在一起则得到括号系统表示法。

用树遍历方法的术语表述，我们可以发现通过递归地遍历一棵树则括号系统对应该树：当沿边下行时写一个“(”，而当沿边上行时写一个“)”。等价地，我们可以把“(”当作对应“开始一个递归调用”，而把“)”当作对应“终止一个递归调用”。

我们用这种方法只是描述树的形状，而不是结点中所包含的任何信息——下面我们考虑一些表示法，如果结点含有相关的关键字或其他附加信息，那么这些表示法是适用的。

*树形状的空间-有效表示法。*括号系统用$2N+O(1)$个比特的序列将大小为N的树编码，这比树使用指针的标准表示法明显地低。实际上，它接近信息理论最佳的编码长度，即Catalan数的对数：

265

$$\lg T_N = 2N - O(\log N)$$

树形状的这种表示法在非常大的树必须要存储（例如数据库的下标结构）或传输（例如哈夫曼树的表示法；见Sedgewick[29]）的应用中是非常有用的。

*树的先序和后序表示法。*将括号系统扩展到包含结点上的信息是简单的。如同树的遍历，根结点的信息可以在子树之前（先序）列出，也可以在子树之后（后序）列出。因此，图5-4

左边的森林的先序表示为

(5(2(1))(3)(4))(6)(9(7)(8))(11(10))

而后序表示则是

(((1)2)(3)(4)5)(6)((7)(8)9)((10)11)

当我们下面讨论细节的时候，将使用先序的术语，当然两种表示法实质上是等价的，这种精化的讨论也适用于后序。

先序度表示法。表示图5-4中的森林形状的另一种方法是整数串

3 1 0 0 0 0 2 0 0 1 0

即以先序列出各结点的子结点数。为了看清为什么这是唯一的表示法，让我们考虑同一个序列，但要从每一项减去1:

2 0 -1 -1 -1 -1 1 -1 -1 0 -1

从左到右进行可以分成若干子序列，具有如下性质：（i）子序列中的数的和为-1以及（ii）子序列中的数的任意前缀的和大于或等于-1。用括号给子序列定界，得到

(2 0 -1 -1 -1)(-1)(1 -1 -1)(0 -1)

266

这就给出一个到括号系统的对应：这些子序列中的每一个均对应森林中的一棵树。删除每个序列中的第一个数并递归地分解则得到括号系统。现在，每个被括起来的数列不仅和为-1，而且还有性质：任意前缀的和是非负的。通过归纳法直接证明，条件（i）和（ii）是在整数序列和树之间建立直接对应的充分必要条件。

二叉树遍历表示法。二叉树的表示法更简单，因为在外部结点（度为0）和内部结点（度为2）之间标记了区别。考虑图5-7中的表达式树，这种熟悉的表示方法就是用与在中间的根相应的关键字列出子树，再括起来：

((x + y)*z) - (w + ((a-(v + y))/((z + y)*x))))

当特别考虑算术表达式的时候，这叫做*中序*（inorder）或*中缀*（infix）。它对应中序树遍历，其中我们写一个“(”，然后遍历左子树，而后写出在根处的关键字，再遍历右子树，然后再写一个“)”。

对应先序遍历和后序遍历的表示方法可以用类似的方式定义。不过对于先序和后序，括号是不需要的：外部（操作数）和内部（操作符）结点是可识别的；因此先序结点度序列是隐藏在表示法中的，它确定树结构，与上面所讨论的方式相同。这样，上面序列的先序或前缀（prefix）列出为

- * + x y z + w / -a + v y * + z y x

而后序或后缀（postfix）列出则是

x y + z * w a v y + - z y +x * / + -

习题5.43 给定一棵（有序）树，使用旋转对应把它作为二叉树来考虑其表示法。讨论有序树的先序和后序表示法与对应二叉树的先序、中序和后序表示法之间的关系。 267

赌徒破产与格路径。对于二叉树，结点或者有两个子结点，或者没有子结点。如果我们对每个结点以先序列出比子结点数少1的数，那么或者得到+1，或者得到-1，我们简单地记成+或-。这样，一棵二叉树可以被唯一地表示成+和-的符号串。图5-7中的树变成

+ + + - - - + - + + - + - - + + - - -

这种编码是先序度表示法的特殊情形。+号和-号的哪些串对应二叉树？

赌徒破产序列。这些串恰好对应下面的情况。设一赌徒，从\$0元开始并投赌\$1元。如果他输了，那么他有\$ − 1元从而破产，但是如果他赢了，那么他就拥有\$1并再次投赌。他的财产图就是正-负号序列的部分和（正号的个数减去负号的个数）的图。任何不穿过\$0点的路径（除在最后一步外）都代表赌徒可能的破产路径，这样的路径也直接与二叉树对应。给定一棵二叉树，我们像刚刚描述的那样做出对应的路径：通过与我们上面使用的证明有序树的先序度表示法有效性相同的推理，这条路径是赌徒的破产路径。给定赌徒的破产路径，恰好有一种方法将该路径分成两条具有相同特征的子路径：删除第一步，并在它碰到\$0轴的第一位置将路径分裂。这种分裂（归纳地）导出对应的二叉树。

选票问题。考虑这种情况的第二种方式是考虑一次选举，其中胜者得到$N+1$张选票，而败者得到N张选票。此时正负序列对应选票的集合，而对应二叉树的那些正-负序列是那些在选票被计数时胜者从未落后的序列。

格子中的路径。思考这种情况的第三种方式是考虑在N行N列方格中的路径。这些路径从左上角到右下角行进，使用一些“右”步和“下”步。存在$\binom{2N}{N}$条这样的路径，但是每$N+1$条中只有一条开始“右”行并且不与对角线交叉，因为在这些路径和二叉树之间存在一个直接的对应，如图5-12所示。显然这是一个赌徒被旋转45°的财产图（或是在选举计票时获胜者的盈余票的图）。如果对应的树以先序遍历，那么这也是栈大小的图。

我们将研究这些*赌徒破产序列*或第6章和第7章中的*选票序列*的性质，它们实际上在排序算法和归并算法的分析中是相关的（见6.6节），并且它们可以用与串计数相关的一般工具来研究（见7.5节）。

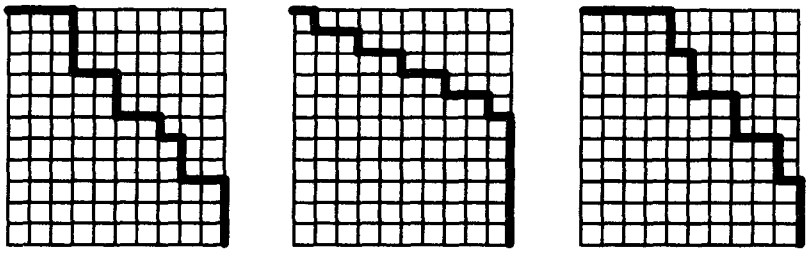


图5-12 图5-1中的各二叉树的格-路径表示法

习题5.44 找出并证明在N-步赌徒破产路径和$N-1$个结点的有序森林之间的对应的有效性。

习题5.45 有多少N位二进制串具有下述性质：对于所有的k，其前k位比特中1的个数不超过0的个数？

习题5.46 将一有序森林的括号表示法与相关的二叉树的加-减号表示法进行比较，解释比较的结果。

习题5.47 从加减号表示法证明，对一棵二叉树的广度优先搜索（即层序，将未访问的结点存储到一个队列中）和深度优先搜索（即先序，将未访问的结点存储到一个栈中）使用相同的空间量。

平面细分表示法。我们讨论另外一种经典的对应，因为它在组合数学中非常著名：“三角剖分N边形”表示法，在图5-13中对图5-1左边的那棵二叉树给出该表示法的显示。给定一个N边形，存在多少种方法把它分成具有连接顶点的非交叉“对角线”的三角形？答案是Catalan数，因为它和二叉树直接对应。这个标示着Catalan数的首次应用出现在欧拉和Segner1753年的工作中，大约是在Catalan本人一个世纪之前。从图5-13看这种对应很简单：给定一个三角

剖分的N边形，在每一条对角线上放一个内部结点并将一个结点（根结点）放到一条外面的边上，将其余每一条外面的边上放一个外部结点。然后把根与其所在三角形的另外两个结点连接起来，并且继续以这种方式向下连接到该树的底层。 268~269

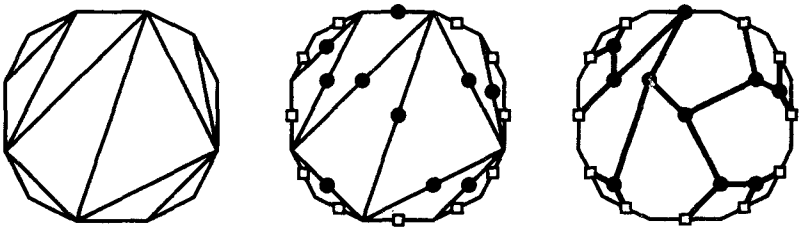


图5-13 二叉树对应三角剖分的N-边形

这种特殊的对应在组合数学中是经典的，而类似的平面细分方法最近已经得到研发，它在某些几何算法的设计和分析中很重要。例如，$2D$-树数据结构（见Sedgewick[29]）是基于用一些水平线划分平面上的一个矩形区域，然后再用垂直线进一步划分那些所得到的区域等等，继续划分直到理想的程度，水平线和垂直线交替进行。这种递归划分对应树的一种表示法。这种结构可以用于细分多维空间以确定点的定位及其他应用。

图5-14对于5-结点树总结了上面讨论的最著名的一些树表示法。我们将详细讨论这些表

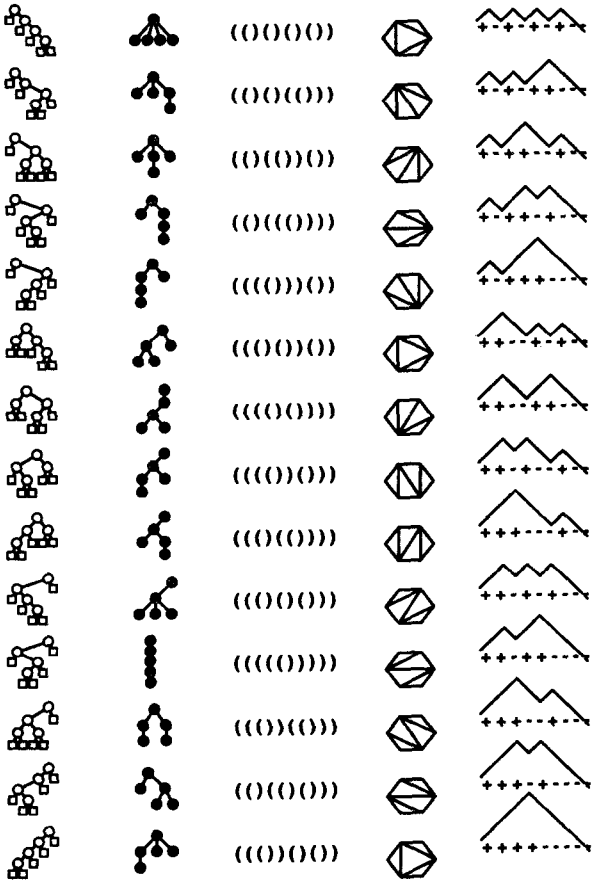


图5-14 二叉树、树、括号法、三角剖分、破产序列

示法，因为Catalan数经常出现在组合数学中，而且由于Catalan数可以对一种非常基本的数据结构进行计数，因此它们也经常出现在算法分析中。一种特定算法的性质常常用一种等价的表示法会比另外的表示法看得更透彻。

习题5.48 证明图5-13对应下列括号系统

(()()) ((()())()(()())(()()())) ((()())()())

270 ~ 271

习题5.49 当任一矩形的高对宽的比率在α和$1/\alpha$（$\alpha>1$为常数）之间时，给出用细分的矩形表示一棵树的方法，求出使α尽可能小的解。

习题5.50 有一种在三角剖分中左右对称的明显的对应反映树的左右对称。那么关于旋转又如何呢？是否在对应一种非对称三角剖分的N次旋转的N棵树之间存在某种关系？

习题5.51 考虑具有下面两个性质的N个整数的串：第一，如果$k>1$在该串中，那么$k-1$也在。第二，在任一整数的任意两次出现位置之间必然出现一个更大的整数。证明长为N的这种串的个数可由Catalan数描述，并找出与树或二叉树的直接对应。

5.12 无序树

上面给出的树和森林的定义的一个本质方面是树的序的概念：单棵树出现的顺序被认为是重要的。事实上，我们一直在研究的树也叫做有序树。当我们考虑各种计算机表示法，或者比如当我们在纸上画一棵树的时候，顺序的重要性是很自然的，因为我们必然要一棵一棵有先有后地画出——其树以不同顺序出现的那些森林看上去是不同的。然而，对有一些应用实际上顺序是无关的，树的内在图论结构占据核心重要的地位。当考虑基本定义时我们将看到这样一些算法的实例，然后再考虑无序树的计数问题。

定义 一棵无序树是一个添加到无序树的多重集中的结点（叫做根）。（这样的多重集称为无序森林。）

图5-15显示两片森林（其中的结点刚好是做了标记的），当将它们作为无序森林考虑时，二者是等价的。

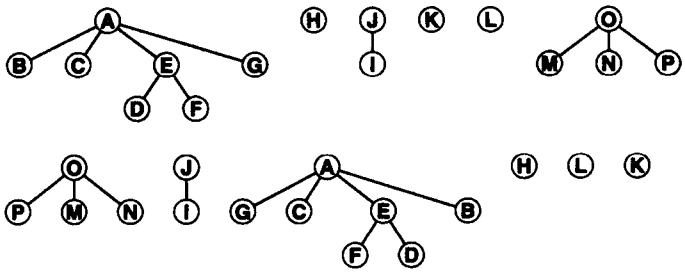


图5-15 同一有根无序森林的两种表示法

有根无序树。作为有根无序树自然出现的算法的例子，我们考虑求并问题：给出一个算法，该算法将处理N个互异项上的一系列等价关系，如果两项已经等价则返回0，如果其关系能够使等价类的个数减1则返回1。例如，给定标记为A到O的16项，则序列

$$A\equiv B \quad A\equiv C \quad B\equiv C \quad O\equiv M \quad J\equiv I \quad E\equiv D$$
$$O\equiv N \quad G\equiv A \quad E\equiv F \quad B\equiv D \quad O\equiv P \quad F\equiv A$$

应该得到返回值的序列

1 1 0 1 1 1 1 1 1 1 1 0

因为前两个关系使得A、B和C等价，然后第3式发现B和C已经等价，等等。

程序5.4给出这个问题的一个解法，它用到无序森林的显式“父链”表示法。将森林表示成一个数组：对应每个结点的项是它在包含它们的树中父结点的下标（那些根的下标为0）。算法认为根“表示”树中所有的结点。给定一个结点，它对应的根可以简单地通过跟踪父链直到发现0而被找到；给定一条边，可以找到对应两条组成边的根。如果两个结点对应相同的根，那么它们属于相同的等价类，这个新关系就是多余的；否则该关系连接至此尚不连通的两个成分。图5-15中的森林是由上面给出的例子构成的。森林的形状依赖于所见到的关系以及它们被表示的顺序。

程序5.4 求并

```
function find(x, y: integer; uf: boolean): boolean;
  var i, j: integer;
  begin
  i := x; while dad[i] > 0 do i := dad[i];
  j := y; while dad[j] > 0 do j := dad[j];
  if uf and (i<>j) then dad[j] := i;
  find := (i<>j)
  end;
```

该算法沿着树向上进行，从不考虑结点的子树（甚至不测试有多少子树）。从组合学上看，求并算法是从关系的排列到无序森林的映射。程序5.4是相当简单的，对于基本想法的各种改进已经提出并得到分析。需要注意的关键一点是，结点的子结点出现的顺序对算法是不重要的，或者说对于相关树的内部表示法不重要——这些算法提供无序树在计算中自然出现的一个例子。

无根（“自由”）树。更一般的概念是一种没有根结点之分的树。为了恰当定义“无根、无序树”（通常也叫做“自由树”，或者就叫做“树”），方便的方法是从一般到特殊，从图开始，图是基于结点的集合和它们之间连通关系的所有组合学对象的基本的基础结构。

定义 图是一组结点和一组连接一对不同结点（顶多连接任意一对结点）的边的集合。

我们可以想像从某个结点开始并“沿着”一条边到该边的构成结点，然后再沿一条边到另外的结点，等等。以这种方法从一个结点通向另外的结点的边的最短序列叫做简单路径。如果存在一条简单路径连接图的任意一对结点，那么这个图就是连通的。从一个结点开始又返回到它自己的简单路径叫做圈。

每棵树都是一个图；那么哪些图是树呢？众所周知，下面四个条件中的任意一个条件都是保证N个结点的图G是（无根无序）树的充分必要条件：

(i) G有$N-1$条边并且没有圈。

(ii) G有$N-1$条边并且是连通的。

(iii) 对于G的每一对顶点，恰好有一条简单路径将它们连接。

(iv) G是连通的，但是如果去掉其任意一条边均不再连通。

就是说，我们可以用这些条件中的任意一个来定义自由树。

272 ~ 274

作为以一种自然方式产生自由树的算法的例子，考虑我们能够提出的关于图的最基本问题：它是连通的，还是不连通？就是说，每一对顶点都存在一条路径连接它们吗？如果是这样，那么就存在组成这种路径的边的一个最小集，称之为图的生成树（spanning tree）。如果图不连通，则存在生成森林，其每个连通部分有一棵树。图5-16给出一个巨型图的一棵生成

树的实例。

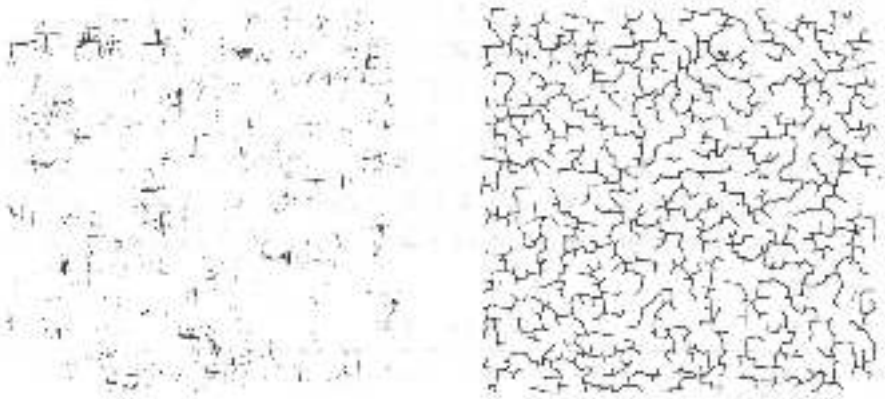


图5-16 一个巨型图和它的一棵生成树

定义 *N个顶点的图的生成树是图的形成一棵树的$N-1$条边的集合。*

按照树的基本性质，生成树必然包含所有的顶点，它的存在证明所有点对都通过某条路经而相连。生成树是无根无序树。

求生成树的一个著名算法是Kruskal算法：依次考虑每条边，检查向当前组成所建部分生成树的集合添加下一条边是否会有圈生成。如果没有，则把该边加入到生成树中并继续考虑下一条边。当集合含有$N-1$条边的时候，这些边则代表一棵无序无根树；实际上，它就是该图的生成树。实现Kruskal算法的一种方法是使用上面为测试圈而给出的求并算法。求解生成树的许多其他方法已经设计出来并进行了分析。不过，现在要注意的关键一点是，Kruskal算法是在计算中自然产生自由树的一个例子。

275

组合学文献包含大量的图论材料，包括许多教科书；而计算机科学文献包含大量图论算法的材料，也包括许多的教科书。当然，全面囊括这些材料将超出本书的范围，但是，理解要探讨的较简单的结构和算法，对于处理关于随机图的性质和图的算法分析的更困难问题是有效的准备。我们一直在考虑的那些方法直接适用的图论问题的例子可以在[13]和经典的参考文献Harary和Plamer[18]中找到。在第8章还将考虑某些特殊的图。不过现在让我们回到对各种类型的树的研究上来。

习题5.52 对N个被标记顶点的$2^{\binom{N}{2}}$个图中有多少是自由树？

习题5.53 对于上面列出的四个性质中的每一个，证明其他三个性质可以推出。（这实际上是12道习题！）

树的分级结构。我们已经定义的树的四种主要类型形成一种分级结构，如表5-3中的总结和描述。（i）自由树是最一般的简单无圈连通图；（ii）有根树有一个特殊的根结点；（iii）有序树是一种有根树，但是其结点的子树顺序很重要；（iv）二叉树是具有下面进一步限制的有序树：每一个结点的度为0或2。在我们使用的命名法中，形容词描述了将每一种类型的树与在分级结构中它上面的那种类型分开的特征。将每一种类型的树与在分级结构中它下面的那种类型分开的命名法也常常使用。因此，有时我们把自由树称为无根树，有根树称为无序树，而有序树称为一般Catalan树。

在文献中出现的多种称呼也是可用的。有序树常常叫做平面树，而无序树也叫做非平面树。术语平面的使用是因为这些结构通过平面上的连续操作可以互相变换，虽然这个术语被

广泛使用，但是我们还是更愿意使用有序，因为它关于计算机表示法具有自然的含义。表5-3中的术语有向指的是下面的事实：根是特殊的，因此存在边朝向根的方向；如果从上下文不能明显看出存在一个所提到的根，那么我们更愿意使用术语有根的。 276

表5-3 树的命名法小结

	其他名字	基本性质	相同的树	不同的树
自由树	无根树 树	连通的 无圈		
根树	外根树 有向树 无序树	给定根结点		
有序树	平面树 Catalan树	结点次序重要		
二叉树	二叉Catalan树	左，右子树		

随着定义越来越严格，被认为是不同树的数目越来越大，因此，给定树的大小，有根树比自由树多，有序树比有根树多，二叉树比有序树多。事实上，有根树的数目与自由树的数目之间的比率与N成正比；有序树对有根树的对应比率随着N指数地增长；二叉树对有序树的比率为常数。本节的其余部分专门推导解析结果，这些结果将量化上述特性。计数结果在表5-4中做了概括。

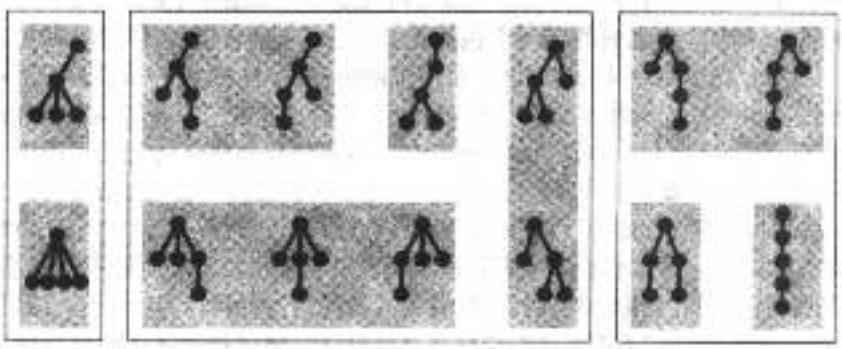


图5-17 具有5个结点的树（有序树、无序树以及无根树）

图5-17是对具有5个结点的树的分级结构的解释。图中画出14种不同的5-结点有序树，这些树用小阴影和较大的开矩形被进一步组织成等价类。存在9种不同的5-结点有根树（那些在阴影矩形中的树作为有根树是恒等的）和3种不同的5-结点自由树（被封在那些大矩形中的树作为自由树是恒等的）。有趣的是要注意，作为一种分级结构，这种表示法本身可以被表示成一棵树（见习题5.48）。注意，刚刚在图5-17中给出的计数对应表5-4中的第4列（$N=5$）。

从组合学的观点看，我们也许对自由树更感兴趣，因为它们在最本质的水平上分辨这些结构。从计算机应用的观点看，我们或许对二叉树和有序树更感兴趣，因为它们具有标准的计算机表示法能够将树唯一确定的性质，而我们对有根树的兴趣在于它们是典型的递归结构。在本书中，我们考虑所有这些类型的树不仅因为它们均出现在重要的计算机算法中，而且还

因为对它们的分析几乎阐释了我们介绍的所有解析方法。但是，我们保持对算法的偏好，将树这个词保留给有序树，后者恐怕是以最自然的方式出现在计算机应用之中的。组合数学教材更一般地将不加修饰的“树”用来描述无序树或自由树。

表5-4 非标号树的计数

	2	3	4	5	6	7	8	9	10	N
自由树	1	1	2	3	6	11	23	47	106	$\sim c_1\alpha^N/N^{5/2}$
根树	1	2	4	9	20	48	115	286	719	$\sim c_2\alpha^N/N^{3/2}$
有序树	1	2	5	14	42	132	429	1430	4862	$\sim c_3 4^N/N^{3/2}$
二叉树	2	5	14	42	132	429	1430	4862	16 796	$\sim(4c_3)4^N/N^{3/2}$

$\alpha \approx 2.9558, c_1 \approx 0.5350, c_2 \approx 0.4399, c_3 = 1/4\sqrt{\pi} \approx 0.1410$

习题5.54 在所有6个结点的有序树之间哪种6-结点自由树结构最常出现？（图5-17指出对5-结点的答案为图中间的那棵树。）

习题5.55 对于7、8以及更多的结点回答上一道习题的问题，能回答到多少结点就回答到多少结点。

在二叉树搜索和使用二叉树的其他算法中，我们直接使用连接到子树的链的序偶表示子树的序偶。类似地，表示一般Catalan树子树的典型方法是链接到子树的有序表。在树和它们的计算机表示之间存在着一一对应。实际上，我们在5.11节以一种明确的方式讨论过表示树和二叉树的一些不同方法。当考虑对有根树和自由树的表示时，情况是不同的，我们面对几种不同的方式以表示同一棵树。这在算法设计和分析中有许多的含义。典型的例子是树的同构问题：给定两个不同的树表示法，确定是否它们表示相同的有根树，或相同的自由树。当然，在算法分析中，每棵树的表示方法数可能对研究这样的算法产生影响。

在使用树的分析算法中遇到的第一个挑战是找出一个概率模型，该模型真实地近似即将发生的情况。这些树是随机的吗？树的分布是由外部随机性引起的吗？树的表示法如何影响算法和分析？许许多多的分析问题由此而产生。例如，上面提到的“求并”问题已经在使用一些不同的模型进行分析（见Knuth和Schönhage[23]）。我们可以假设，等价关系的序列是随机的结点对，或它们对应随机森林中的一些随机边，等等。我们已经在二叉查找树和二叉Catalan树中看到，基本的递归分解导致分析中的类似，而所得到的分布则导致分析中明显的差别。

277 ~ 279

对于各种各样的应用，我们的兴趣可能在于那些度量各种类型的树的基本特性的参数值，因此我们面对许多的分析问题要考虑。计数的结果是经典的（见[18]、[27]和[21]），并在表5-4中做了总结。这些结果的某些推导的讨论在下面给出。通过符号方法容易得到生成函数的函数方程，但是对系数的渐近估计在某些情况下稍微超出本书的范围。更多的细节可以在[13]中找到。

习题5.56 给出一个有效的算法，该算法以表示一棵树的一组边作为输入，并且产生该树的括号系统表示法作为输出。

习题5.57 给出一个有效的算法，该算法以表示一棵树的一组边作为输入，并且产生该树的二叉树表示法作为输出。

习题5.58 给出一个有效的算法，该算法以两棵二叉树作为输入，并且确定被当作无序树时它们是否不同。

习题5.59 [Aho, Hopcroft和Ullman] 给出一个有效的算法，该算法以两个括号系统作为输入，并确定它们是否表示相同的有根树。

有根无序树的计数。令$\mathcal{U}$为具有相关OGF

$$U(z)=\sum_{u\in\mathcal{U}} z^{|u|}=\sum_{N\geq 0} U_N z^N$$

的所有（无序）有根树的集合,其中U_N为具有N个结点的有根树的棵数。由于每棵有根树包含一个根和有根树的一个多重集，因此我们可以用两种方式把这个生成函数表示成无穷乘积:

$$U(z)=z\prod_{u\in\mathcal{U}}(1-z^{|u|})^{-1}=z\prod_{N}(1-z^N)^{-U_N}$$

第一个乘积是与符号方法相关的“多重集”结构的简单应用（见习题3.62）：对于每一棵树u，项$(1-z^{|u|})^{-1}$考虑到该集中u发生任意次数的可能。第二个乘积通过将对应具有N个结点的那些树的U_N项分组而得到。 280

定理5.11 (有根无序树的计数) 对有根无序树计数的OGF满足函数方程

$$U(z)=z\exp\left\{U(z)+\frac{1}{2}U(z^2)+\frac{1}{3}U(z^3)+\cdots\right\}$$

渐近地有

$$U_N\equiv[z^N]U(z)\sim c\alpha^N/N^{3/2}$$

其中$c\approx 0.4399237$和$\alpha\approx 2.9557649$。

证明 继续上面的讨论，两边取对数:

$$\begin{aligned}\ln\frac{U(z)}{z}&=-\sum_{N\geq 1}U_N\ln(1-z^N)\\&=\sum_{N\geq 1}U_N\left(z^N+\frac{1}{2}z^{2N}+\frac{1}{3}z^{3N}+\frac{1}{4}z^{4N}+\cdots\right)\\&=U(z)+\frac{1}{2}U(z^2)+\frac{1}{3}U(z^3)+\frac{1}{4}U(z^4)+\ldots.\end{aligned}$$

定理所述的函数方程由两边再取指数而得到。

渐近分析超出了本书的范围，它依赖于有关我们在第4章介绍的直接生成函数渐进性的复分析方法。细节可以在[13]、[18]和[27]中找到。 ■

这个结果告诉我们几个有趣的事实。首先，OGF不容许分析的初等函数的显式形式，可是它被该函数方程完全确定。事实上，同样的推理指出，高$\leq h$的树的OGF满足

$$U^{[0]}(z)=z;\quad U^{[h+1]}(z)=z\exp\left\{U^{[h]}(z)+\frac{1}{2}U^{[h]}(z^2)+\frac{1}{3}U^{[h]}(z^3)+\cdots\right\}$$

不仅如此，当$h\to\infty$时还有$U^{[h]}(z)\to U(z)$，而且两个级数到$h+1$项是一致的。这提供了一种计算任意个初始值的方法:

$$U(z)=z+z^2+2z^3+4z^4+9z^5+20z^6+48z^7+115z^8+286z^9+\cdots$$

还要注意的是，尽管OGF不容许闭的形式，但是精确的渐近分析还是可能受到影响。实际上，这个分析是所谓渐进计数Darboux-Polya方法的历史起源，我们在4.10节简要介绍过它。Polya在1937年用这种方法实现了各种树类型的渐近分析（见Polya和Read[27]，Polya经典论文的译文），特别是碳氢化合物、酒精等化学异构体的模型。 281

习题5.60 编写一个程序计算所有小于你的机器中可以表示的最大整数的U_N的值，使用

课文中提出的方法，使用该方法估计对于大的N需要多少次（无限精度）算术运算？

习题5.61 [Harary-Palmer] 证明

$$NU_{N+1} = \sum_{1 \le k \le N} \left(kU_k \sum_{k \le m \le N} T_{N+1-m} \right)$$

并推出：U_N能够以$O(N^2)$次算术运算确定。（提示：对函数方程微分。）

习题5.62 给出一个多项式时间算法以生成一棵大小为N的随机有根树。

自由树的计数。若不确定根，那么组合论证多少有些复杂，虽然这至少自1889年就已经知道（见Harary和Palmer[18]）。渐近估计可以通过生成函数论证，使用刚刚得到的有根树的渐近公式得出。我们把细节留作习题。

习题5.63 证明：N个结点的有根树的数目以N个结点的自由树的数目为下界，并以N个结点的自由树的数目的N倍为上界。（因此，这两个量增长的指数阶是相同的。）

习题5.64 令$F(z)$为自由树的OGF。证明

$$F(z) = U(z) - \frac{1}{2}U(z)^2 + \frac{1}{2}U(z^2)$$

习题5.65 导出表5-4中给出的自由树的渐近公式。使用前面的习题以及定理5.11中关于（无序）有根树给出的公式。

5.13 标号树

上面的计数结果假设树中的结点是不可区分的。相反，如果我们假设结点有不同的标识，那么将存在更多的方法使它们构成树。例如，当把不同的结点用作根的时候就得到不同的树。当我们考虑子树的阶以及指定根的时候，“不同”的树的数目将增加。我们把“标号”当作区分结点的组合学装置。当然，这和二叉查找树中的关键字毫无关系，它们是与结点相关的应用数据。

282

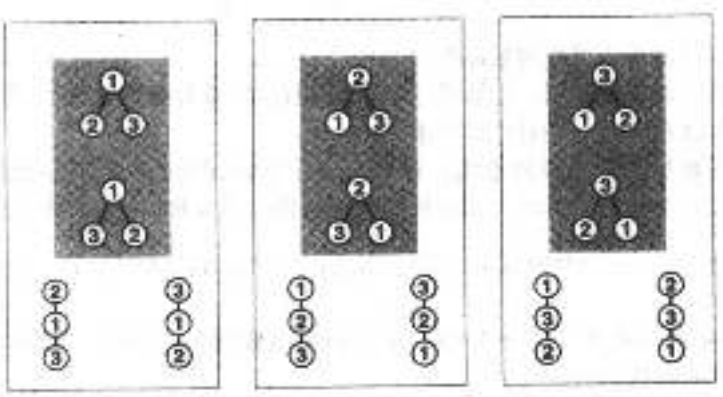


图5-18 具有3个结点的标号树（有序的、无序的、无根的）

不同类型的标号树在图5-18中解释，它们与图5-17相对应。所有的树都是不同的有根、有序和标号树；那些在阴影矩形中的树作为无序标号树是同构的，而那些在大矩形中的树作为无根、无序标号树也是同构的。像往常一样，我们的兴趣是想知道我们考虑过的每种类型中存在多少标号树。表5-5给出一些小的值以及对于各种标号树计数的渐近估计。表5-4中的第2列（$N=3$）对应图5-18中的树。

表5-5 标号树的计数

	2	3	4	5	6	7	N
有序树	2	12	120	1680	46 656	665 280	$\frac{(2N-2)!}{(N-1)!}$
有根树	2	9	64	625	7976	117 649	N^{N-1}
自由树	1	3	16	125	1296	16 807	N^{N-2}

正如第3章所讨论的，EGF是处理标号树计数的合适工具，这不仅因为存在多得多的可能性，而且还因为我们对标号结构使用的基本组合学处理方法通过EGF很自然地被理解。 283

习题5.66 4个结点的哪种树具有最多的不同标号方法？对于5、6以及更多结点回答这个问题，能回答到多少个结点就回答到多少。

对有序标号树的计数。令L为具有EGF

$$L(z)=\sum_{l\in\mathcal{L}}\frac{z^{|l|}}{l!}=\sum_{N\geq 0}L_N\frac{z^N}{N!}$$

的有序标号（有根）树的集合，其中L_N为具有N个结点的树的数目。有序标号森林或者为空，或者为有序标号树的无序序列，因此由符号方法我们有

$$\begin{aligned}L(z)&=z(1+L(z)+L(z)^2+L(z)^3+\cdots+L(z)^k+\cdots)\\&=\frac{z}{1-L(z)}\end{aligned}$$

这实质上和上面对有序（非标号）树所使用的是相同的论证，但是现在我们对标号对象用EGF（定理3.8）来计数，此前我们对非标号对象一直在使用OGF（定理3.7）。因此

$$L(z)=\frac{1-\sqrt{1-4z}}{2}$$

而具有N个结点的有序标号有根树的数目由

$$N![z^N]L(z)=N!\frac{1}{N}\binom{2N-2}{N-1}=\frac{(2N-2)!}{(N-1)!}$$

给出。

解释这个结果的另外方式是通过组合对应：非标号树由先序遍历确定，$N!$种排列中的任一排列均可与先序遍历一起使用，对具有N个结点的有序树指定标号，因此标号树的数目正是非标号树数目的$N!$倍。显然这种论证是一般的。对于有序树，标号和非标号是紧密相关的，而他们的计数仅差一个因子$N!$。 284

无序标号树的计数。无序（有根）标号树也叫做Cayley树，因为它们在19世纪由A. Cayley做过计数。令C是与EGF

$$C(z)=\sum_{c\in\mathcal{C}}\frac{z^{|c|}}{|c|!}=\sum_{N\geq 0}C_N\frac{z^N}{N!}$$

相关的所有Cayley树的集合，其中C_N为具有N个结点的Cayley树的数目。Cayley森林或者为空，或者是k棵具有不同标记的Cayley树的集合。因此，由符号方法得到

$$C(z)=z\left(1+C(z)+\frac{C(z)^2}{2!}+\frac{C(z)^3}{3!}+\cdots+\frac{C(z)^k}{k!}+\cdots\right)=ze^{C(z)}$$

这只在一点上不同于上面的论证：当序不重要时，$(C(z))^k$ 将每一森林计数$k!$次，因此我们用$k!$除之。

定理5.12（无序标号树的计数） 对无序标号树计数的OGF满足函数方程

$$C(z) = ze^{C(z)}$$

大小为N的这种树的数目为

$$C_N = N![z^N]C(z) = N^{N-1}$$

而这种树的无序k-森林的数目为

$$C_N^{[k]} = N![z^N]\frac{(C(z))^k}{k!} = \binom{N-1}{k-1}N^{N-k}$$

证明 该函数方程可以由拉格朗日反演（见3.10节）显式解出，由此立即产生定理所述结果。 ■

从组合学上看，对于无序标号树，N结点有根树的数目显然就是无根树的N倍：每一棵无根树恰好对应N棵有根树，只要把每个结点当作根即可看出。因此，标号自由树的数目为N^{N-2}。

285

对于非标号树和标号树的计数生成函数在表5-6中给出，以供参考。表5-5给出了标号树的系数的值。

表5-6 树计数生成函数

	非标号(OGF)	标号（EGF）
有序树	$G(z) = \frac{z}{1-G(z)}$	$L(z) = \frac{z}{1-L(z)}$
有根树	$U(z) = z\exp\left\{\sum_{i\geq 1} U(z^i)/i\right\}$	$C(z) = ze^{C(z)}$
自由树	$U(z) - U(z)^2/2 + U(z^2)/2$	$C(z) - C(z)^2/2$

习题5.67 找出用具有小于N的正整数的$(N-1)$元组表示标号有根树的方法。就是说，存在N^{N-1}个这样的N元组和同样数目的这种树：找出它们之间的一一对应。（提示：把结点和它们的父结点联系起来。）

习题5.68 证明，对标号自由树计数的EGF等于$C(z) - C(z)^2/2$。

5.14 其他类型的树

把各种局部和整体的限制施加到树上，例如满足特定应用的需求或者试图排除退化的情形，常常是很方便的。从组合学的观点看，任何限制都对应一种新种类的树，需要求解一组新的问题以对树进行计数并了解它们的统计性质。在这一节，我们列举许多著名的和广泛使用的特殊种类的树，以供参考。例子在图5-19中画出，而一些定义则在下面的讨论中给出。（注意记号：在这一节，我们使用$T(z)$表示CatalanOGF的各种推广的OGF，以强调分析中的相似性，而使读者解脱过多的符号负担。）

定义 一棵t-叉树或者是一个外部结点，或者是连接到t棵子树的有序序列上的一个内部结点，这些子树都是t-叉树。

286

这是二叉树的自然的推广，当我们在第3章考虑拉格朗日反演时，二叉树是作为一个例子

考虑的，我们坚持每个结点恰有t个分支，这些树通常被认为是有序的——这对应一种计算机表示法，在这种表示法中，每一个结点保留t个链以指向该结点的分支。在某些应用中，关键字可以和内部结点相联系；在另外一些情形下内部结点可能对应$t-1$个关键字的序列；在其他一些情形下，数据可以与外部结点相结合。这种类型的一种重要的树是四叉树，此时关于几何数据的信息是通过将一个区域分成四个象限来组织的，过程递归地进行。

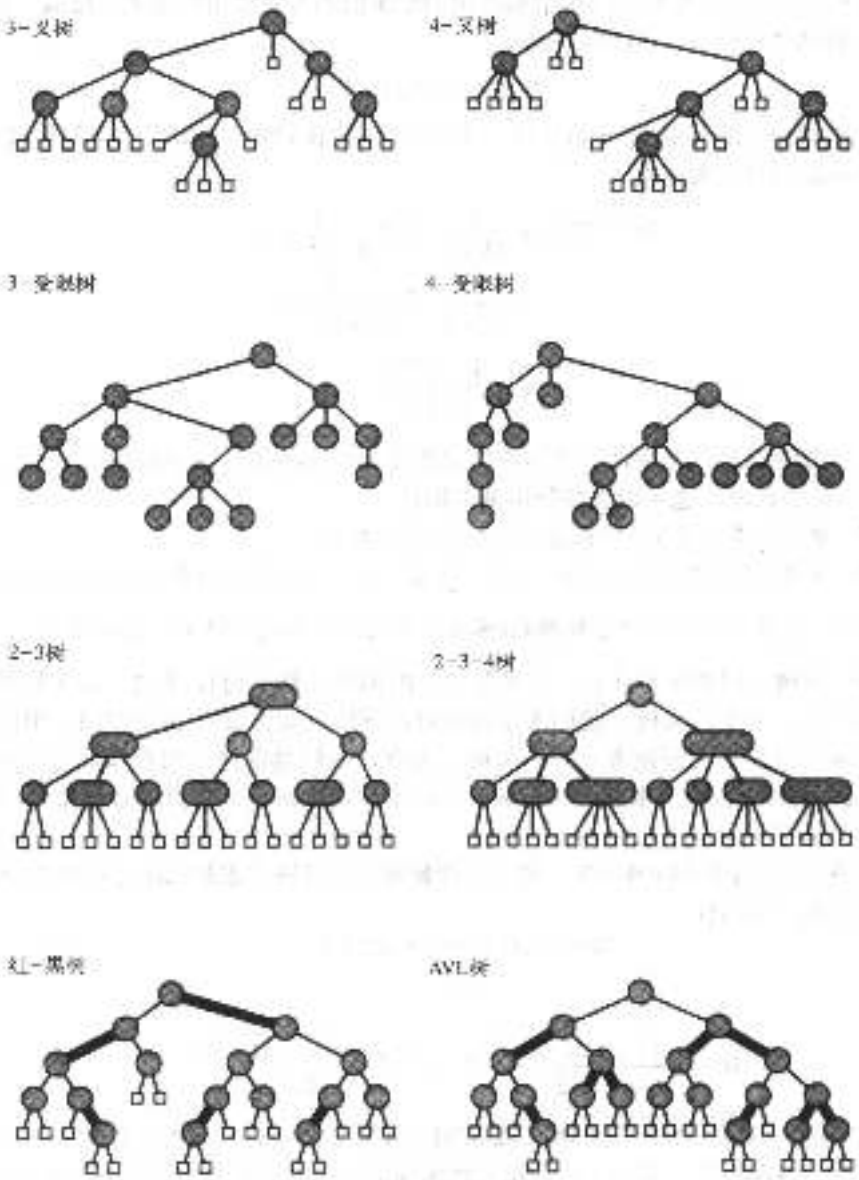


图5-19 各种其他类型的树的例子

定理5.13（t叉树的计数） 对t叉树（通过外部结点）进行计数的OGF满足函数方程

$$T(z)=z+(T(z))^t$$

具有N个内部结点和$(t-1)N+1$个外部结点的t-叉树的数目为

$$\frac{1}{(t-1)N+1}\binom{tN}{N} \sim c_t(\alpha_t)^N / N^{3/2}$$

其中$\alpha_t = t^t/(t-1)^{t-1}$且$c_t = 1/\sqrt{(2\pi)(t-1)^3/t}$。

证明　我们以与在3.11节对$t=3$的情形给出的解类似的方式使用拉格朗日反演。根据符号方法，具有由外部结点度量大小的OGF满足

$$T(z) = z + T(z)^3$$

这能够满足拉格朗日反演，由于$z = T(z)(1-T(z)^2)$，因此我们有具有$2N+1$个外部结点（N个内部结点）的树的数目的表达式:

$$\begin{aligned}[z^{2N+1}]T(z) &= \frac{1}{2N+1}[u^{2N}]\frac{1}{(1-u^2)^{2N+1}} \\ &= \frac{1}{2N+1}[u^{N}]\frac{1}{(1-u)^{2N+1}} \\ &= \frac{1}{2N+1}\binom{3N}{N}\end{aligned}$$

287 ~ 288

它等价于在3.11节中给出的表达式，将其推广立即得到所述结果。当我们使用与对Catalan数（见4.3节）相同的方法时，也可以直接得出渐近估计。 ■

习题5.69　求出具有总数N个内部结点的k-森林的数目。

习题5.70　对具有N个内部结点的t-叉树的数目，导出定理5.13中给出的渐近估计。

定义　一棵t-受限树是一个包含链接到t棵或更少的t-受限树的结点（称之为根）。

受限树和t-叉树之间的区别在于，不是每一个内部结点都必须有t个链。这在计算机表示法中直接产生结论：对于t-叉树，我们还要为所有内部结点均保留t个链的空间，但是t-受限树使用标准的对应可能被更好地表示为二叉树。再有，我们通常把它们看作是有序的，虽然我们也可能考虑无序的与/或无根的t-受限树。在t-受限树中每一个结点最多链接到$t+1$个其他的结点，如图5-19所示。

$t=2$的情形对应所谓的Motzkin数，对于这种数我们可以通过求解二次方程得到OGF $M(z)$的一个显式表达式。我们有

$$M(z) = z(1 + M(z) + M(z)^2)$$

从而

$$M(z) = \frac{1-z-\sqrt{1-2z-3z^2}}{2z} = \frac{1-z-\sqrt{(1+z)(1-3z)}}{2z}$$

现在，定理4.11提供了$[z^N]M(z)$为$O(3^N)$的直接证明，并且那些来自复渐近分析的方法产生更精确的渐近估计$3^N/\sqrt{3/4\pi N^3}$。实际上，使用大约相同的工作量我们能够得到更一般的结果。

定理5.14 (t-受限树的计数)　令$\theta(u) = 1 + u + u^2 + \cdots + u^t$。对$t$-受限树计数的OGF满足函数方程

$$T(z) = z\theta(T(z))$$

且t-受限树的棵树为

$$[z^N]T(z) = [u^{n-1}](\theta(u))^n \sim c_t\alpha_t^N / N^{3/2}$$

其中ρ是$\theta(\rho) - \rho\theta'(\rho) = 0$的最小正根，而常数$\alpha_t$和$c_t$由$\alpha_t = \theta'(\rho)$和$c_t = \sqrt{\theta(\rho)/2\pi\theta''(\rho)}$给出。

289

证明　定理的第1部分从符号法和拉格朗日反演直接得到。渐近结果需要奇性分析，用到定理4.12的一个拓展（见[13]）。

该结果从Meir和Moon在1978[25]证明的一个定理推得，实际上，它对一大类形如$1 + a_1u + a_2u^2 + \cdots$的多项式$\theta(u)$成立，其中需要满足约束：多项式的系数是正的，而且$a_1$和至少另外一个系数非零。 ■

对于小的t，t-受限树棵数的渐近估计在下表中给出：

	t	c_t	α_t
	2	0.4886025119	3.0
	3	0.2520904538	3.610718613
$c_t\alpha_t^N/N^{3/2}$	4	0.1932828341	3.834437249
	5	0.1691882413	3.925387252
	6	0.1571440515	3.965092635
	∞	0.1410473965	4.0

对于大的t的α_t值接近4，也许它是所期望的，因为此时的树像是一般Catalan树。

习题5.71　利用恒等式$1 + u + u^2 + \cdots + u^t = (1 - u^{t+1})/(1 - u)$找出具有$N$个结点的$t$-受限树的棵数的和式。

习题5.72　写出一个程序，给定t，对于N的所有使得其t-受限树的数目小于你的计算机中最大可表示整数的值，该程序将计算出这个数目。

习题5.73　求出“偶”t-受限树的数目，其中所有的结点均有偶数个小于t的子结点。

高-受限树。其他一些类型的树涉及对高的限制。这样的树之所以重要，是因为它们可以用作二叉树的替代对象，提供保证$O(\log N)$的搜索时间。Adel'son-Vel'skii和Landis[1]在1960年首先指出这一点，从此这样一些树就被广泛地研究（例如，见Bayer和McCreght[3]或Guibas和Sedgewick[17]）。平衡树具有重要的实际意义，因为它们将二叉树查找的简单性和灵活性与具有良好的最坏情形性能的插入操作结合了起来。它们常常用于大的数据库应用，因此其性能的渐近结果具有直接的实际意义。

290

定义　一棵高为0或-1的AVL树是一个外部结点；一棵高$h>0$的AVL树是链接到一棵左子树和一棵右子树的内部结点，两棵子树的高为$h-1$或$h-2$。

定义　一棵高为0的B-树是一个外部结点；一棵M阶且高$h>0$的B-树是一个被连接到一系列在$\lceil M/2 \rceil$和M棵之间的M阶和高为$h-1$的B树的内部结点。

3阶和4阶B-树通常分别叫做2-3树和2-3-4树。有几种方法是使用这些以及类似的结构设计出来的，称为平衡树算法，其一般想法是将排列映射到保证没有长路径的树结构。包括各种类型之间关系等更多的细节由Guibas和Sedgewick[17]给出，他们还指出许多这种结构（包括AVL树和B-树）可以映射到具有标记边的二叉树，如图5-19所示。

习题5.74　不用详细求解计数问题。对于大的N，试将下列各类树以它们基数的递增顺序

排列：3-叉树、3-受限树、2-3树、以及AVL树。

习题5.75 建立一个表，使该表能够给出少于15个结点的AVL树和2-3树的数目，注意，当作为无序树考虑的时候这些结点是不同的。

平衡树结构阐述了在应用中出现的树结构的变体。这些结构导致大量有趣的解析问题，并且它们分布在那些纯组合结构和部分“算法的”结构之间一系列不同程度的位置之上。没有哪种二叉树结构为了统计诸如路径长这样的量在随机插入下被精确地进行过分析，不管它们有多重要。对它们进行计数甚至变成一种挑战（例如，见Aho和Sloane[2]或Flajolet和Odlyzko[11]）。

对于这些类型结构中的每一种，我们有兴趣知道每种大小存在多少本质上不同的结构，以及各种重要参数的统计。对于某些结构，研究对它们计数的函数方程式是相对简单的，因为它们是递归地定义的。（有些平衡树结构甚至不能轻易地被递归定义和分析，但是却需要用将排列映射到这些结构的算法来定义。）如同树的高一样，函数方程是唯一的起点，而对这些结构的进一步分析事实上是相当困难的。对于我们已经讨论过的有些类型，生成函数的函数方程在表5-7中给出。

291

习题5.76 证明表5-7中给出的AVL树和2-3树的生成函数的函数方程。

表5-7 其他类型的树的生成函数

树类型（大小度量）	生成函数的函数方程
3-叉树（外部结点）	$T(z)=z+T(z)^3$
3-叉树（内部结点）	$T(z)=1+zT(z)^3$
3-受限树（结点）	$T(z)=z(1+T(z)+T(z)^2+T(z)^3)$
高为h的AVL树（内部结点）	$A_h(z)=\begin{cases}1 & h<0\\ 1 & h=0\\ zA_{h-1}(z)^2+2zA_{h-1}(z)A_{h-2}(z) & h>0\end{cases}$
高为h的2-3树（外部结点）	$B_h(z)=\begin{cases}z & h=0\\ B_{h-1}(z^2+z^3) & h>0\end{cases}$

更重要的是，就像对（均匀分布的）二叉树和二叉查找树（由算法从随机排列构造时形成分布的二叉树）进行的分析那样，我们常常需要知道对各种类型的树的统计信息，它是按照将某些其他的组合学对象变换成树结构的算法所引起的分布而得到的，这将导致更多的解析问题。就是说，我们已经定义的几种基本树结构适合许多的算法。我们使用术语二叉查找树把组合学对象（二叉树）与将排列映射到它的算法区分开；平衡树和其他一些算法需要使用类似的方式区分。

292

实际上，AVL树、B-树以及其他类型的查找树，当从随机排列构造获得分布时具有根本的意义。“每棵树等可能的”组合学对象之所以一直被研究，不仅因为相关的问题更适合组合学分析，而且还因为对它们的性质的认识可能会给出求解那些当作为数据结构进行分析时发生的问题的领悟。即使这样，仅对平衡树结构计数的基本问题还是相当的困难（例如，见[26]）。没有相关

的算法在随机排列模型下被分析，而平衡树算法的平均情形分析则是算法分析中的难题之一。

图5-20给出这种情形的复杂性的某种提示，它指出随机AVL树中子树大小的分布（所有的树都是等可能的）并可与图5-10，即对应Catalan树的图进行比较，对应BST的图是一系列的直线，高为$1/N$。对任意树的大小N，在Catalan树关于子树具有固定结点数有一个渐近常数概率的情况下，AVL树的平衡条件意味着对于大的N不能出现小的子树。实际上，对于大的N，我们可以期望树在子树的大小聚集在中间附近的意义下是“平衡”的。对某些N确实就是这种情况，而且对于其他某些N也是成立的，在分布中存在两个峰，这意味着大部分的树在一边或另一边具有明显少于一半的结点。事实上，分布展示一种摆动的特性，大致介于这两个极值之间。描述这种情形的解析表达式必须考虑这种摆动，因此不像我们想要的那么简练。据推测，在搜索应用中，当从排列构建平衡树时，也涉及类似的效果，不过这尚未被证明。

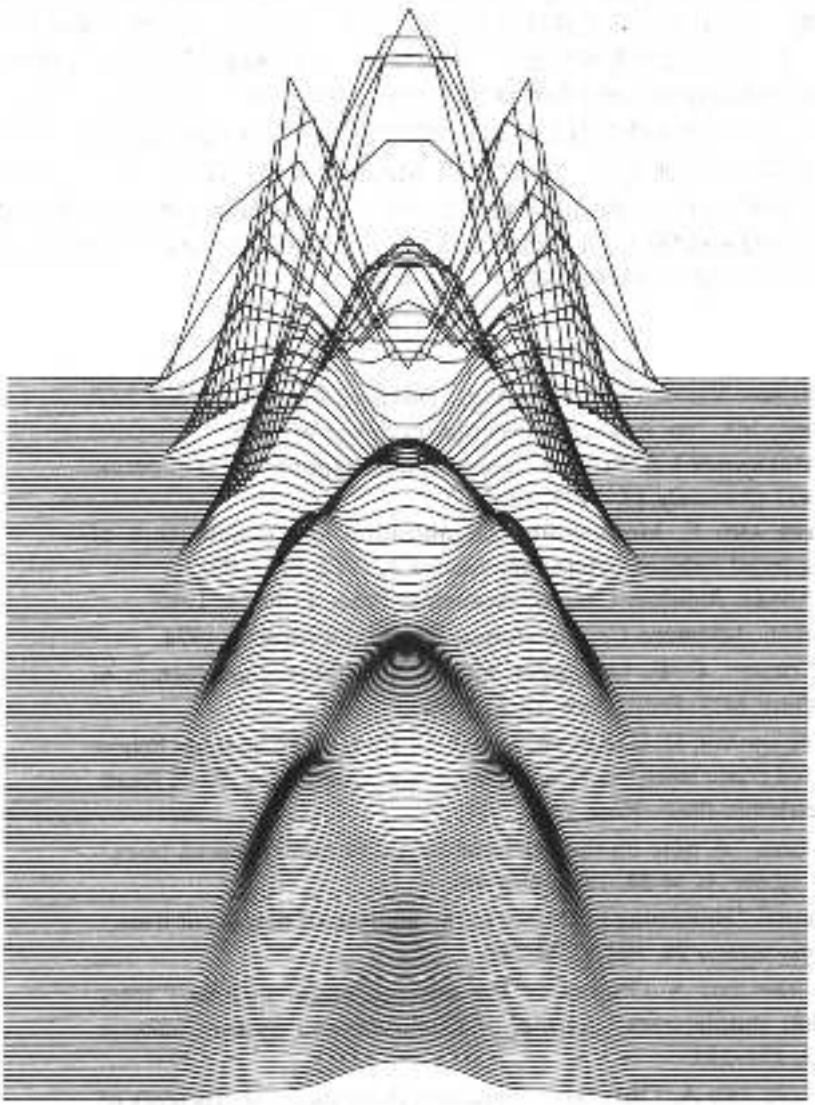


图5-20 AVL分布（随机AVL树中子树的大小）
（伸缩并平移成分离的曲线）

在我们考虑的算法中树作为显式的结构或者作为递归计算的模型是普遍遇到的。大部分关于最重要算法性质的知识都可以追溯到树的性质。

在后面各章我们将会遇到其他类型的树，不过，它们都享有一个内在的递归性质，该性质像上面那样使用生成函数能够使它们的分析更自然：这种递归结构直接导致产生生成函数的闭型表达式或递归公式的方程。分析的第2部分抽取所要的系数，这对于某些类型的树需要一些高级的技巧。

293 ~ 294

通过比较树路径长和树高而展现的分析上的差异是本质性的差异。一般地，我们可以递归地描述组合参数，但是，像路径长这样的“可加”参数要比像高这样的“非可加参数”处理起来容易得多，因为对应组合结构的生成函数结构可以在前一种情形下直接得到。

我们在本章的第一个论题一直是展示作为组合对象的树的广泛的分析历史。近年来发现一些一般的方法，这些方法有助于统一某些经典的结果，并使得认知任意复杂的新的树结构的特征成为可能。我们详细讨论它并给出文献[13]中的一些例子。当然许多重要的问题仍然有待解决。例如，关于树参数最基本问题之一的树高的一些精确结果只是最近才得到。

除了经典组合学和算法分析中的特殊应用之外，我们还尽力指出算法应用如何导致大量新的数学问题，这些问题本身具有有趣和复杂的结构。二叉查找树算法是许多问题的原型，对于这些问题我们知道如何求解：算法把某个输入组合对象（排列，在二叉查找树的情形）变换成某种形式的树。此时，我们的兴趣在于分析这些树的组合性质，但分析不是在统一的模型下，而是在由变换得到的分布下进行。认识所产生的组合结构的详细性质以及研究这些变换的效果是我们处理算法分析的基础。

参考文献

1. G. Adel'son-Vel'skii and E. Landis. *Doklady Akademii Nauk SSR* **146**, 1962, 263–266. English translation in *Soviet Math* **3**.
2. A. V. Aho and N J. A. Sloane. "Some doubly exponential sequences," *Fibonacci Quarterly* **11**, 1973, 429–437.
3. R. Bayer and E. McCreight. "Organization and maintenance of large ordered indexes," *Acta Informatica* **3**, 1972, 173–189.
4. B. Bollobás. *Random Graphs*, Academic Press, London, 1985.
5. L. Comtet. *Advanced Combinatorics*, Reidel, Dordrecht, 1974.
6. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, MIT Press, New York, 1990.
7. N. G. De Bruijn, D. E. Knuth, and S. O. Rice. "The average height of planted plane trees," in *Graph Theory and Computing*, R. C. Read, ed., Academic Press, New York, 1971.
8. L. Devroye. "A note on the expected height of binary search trees," *Journal of the ACM* **33**, 1986, 489–498.
9. L. Devroye. "Branching processes in the analysis of heights of trees," *Acta Informatica* **24**, 1987, 279–298.
10. P. Flajolet and A. Odlyzko. "The average height of binary trees and other simple trees," *Journal of Computer and System Sciences* **25**, 1982, 171–213.
11. P. Flajolet and A. Odlyzko. "Limit distributions for coefficients of iterates of polynomials with applications to combinatorial enumerations," *Mathematical Proceedings of the Cambridge Philosophical*

295

Society **96**, 1984, 237–253.

12. P. FLAJOLET, J.-C. RAOULT, AND J. VUILLEMIN. "The number of registers required to evaluate arithmetic expressions," *Theoretical Computer Science* **9**, 1979, 99–125.
13. P. FLAJOLET AND R. SEDGEWICK. *Analytic Combinatorics*, in preparation.
14. R. GRAHAM, D. E. KNUTH, AND O. PATASHNIK. *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.
15. G. H. GONNET AND R. BAEZA-YATES. *Handbook of Algorithms and Data Structures*, 2nd edition, Addison-Wesley, Reading, MA, 1991.
16. I. GOULDEN AND D. JACKSON. *Combinatorial Enumeration*, John Wiley, New York, 1983.
17. L. GUIBAS AND R. SEDGEWICK. "A dichromatic framework for balanced trees," in *Proceedings 19th Annual IEEE Symposium on Foundations of Computer Science*, 1978, 8–21.
18. F. HARARY AND E. M. PALMER. *Graphical Enumeration*, Academic Press, New York, 1973
19. C. A. R. HOARE. "Quicksort," *Computer Journal* **5**, 1962, 10–15.
20. R. KEMP. "The average number of registers needed to evaluate a binary tree optimally," *Acta Informatica* **11**, 1979, 363–372. 296
21. D. E. KNUTH. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
22. D. E. KNUTH. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
23. D. E. KNUTH AND A. SCHÖNHAGE. "The expected linearity of a simple equivalence algorithm," *Theoretical Computer Science* **6**, 1978, 281–315.
24. H. MAHMOUD. *Evolution of Random Search Trees*, John Wiley, New York, 1992.
25. A. MEIR AND J. W. MOON. "On the altitude of nodes in random trees," *Canadian Journal of Mathematics* **30**, 1978, 997–1015.
26. A. M. ODLYZKO. "Periodic oscillations of coefficients of power series that satisfy functional equations," *Advances in Mathematics* **44**, 1982, 180–205.
27. G. POLYA AND R. C. READ. *Combinatorial Enumeration of Groups, Graphs, and Chemical Compounds*, Springer-Verlag, New York, 1987. (English translation of original paper in *Acta Mathematica* **68**, 1937, 145–254.)
28. R. C. READ. "The coding of various kinds of unlabelled trees," in *Graph Theory and Computing*, R. C. Read, ed., Academic Press, New York, 1971.
29. R. SEDGEWICK. *Algorithms*, Addison-Wesley, Reading, MA, 1988.
30. J. S. VITTER AND P. FLAJOLET, "Analysis of algorithms and data structures," in *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, 431–524. 297

第6章 排　列

组合算法通常只处理N个元素的线性数组的相对顺序，因此我们可以把这些算法看成是按照某种顺序对数1到N的操作。这样的顺序安排叫做*排列*——具有大量有趣性质的常见组合对象。在第1章中，我们首次遇到了排列问题，并利用随机排列作为输入模型，对两个重要的基于比较的排序算法的分析进行了讨论。在本章，我们将全面考查排列的组合性质，指出它们是如何以一种自然的方式与基本排序算法相联系的，并利用概率函数、累积函数和二元生成函数对平均情形分析进行考察。

我们将涉及诸如插入排序、选择排序和冒泡排序等基本排序算法的分析，并将讨论大量在实践中极为重要的其他算法，如Shell排序算法、优先队列算法以及重排算法等。我们也许要涉及这些算法与排列的基本性质之间的对应关系，但我们将强调基本组合结构在算法分析中的重要性。

我们将通过定义排列的几个最重要的性质、考查几个例子、以及这些性质与例子之间的某些关系来开始本章内容。既要考查那些在基本排序算法分析中随时出现的性质，又要考查那些具有独立组合意义的性质。

接着，我们将考虑用不同的方式来表示排列，尤其是由逆序、圈和揭示排列及其逆之间关系的二维表示法所隐含的表示方式。这种表示法也有助于我们定义排列、二叉查找树以及“堆序树”之间明确的关系，并将对排列的某些性质的分析简化成对树的性质的研究。

然后，我们将考虑排列的计数问题，即：统计满足某些性质的排列数。这种问题可以利用生成函数（包括关于标号对象的符号法）以一种直接的方式来解决。特别地，我们将比较详细地考查与排列的“圈结构”有关的性质。

沿用和第5章一样的一般结构，我们接下来要对参数进行分析。对树而言，我们曾考虑过路径长度、树高、树叶的数目以及其他参数。对排列而言，我们将考虑诸如游程数和逆序数等性质，其中的许多性质都很容易分析。和以往一样，在所有排列等可能的假定下，我们也将关注与排列性质有关的各种方法下排列的期望“开销”。期望开销有时可以通过对长度为N、开销为k的排列数找到一个明确的表达式来计算，不过我们强调基于生成函数的便捷方法，如CGF的使用。

本文我们将对两个基本排序算法——插入排序和选择排序——的参数以及它们与排列的两个基本性质——逆序数和左向右最小值——的关系进行分析。我们将表明CGF是如何导出对这些算法进行比较直接的分析的。我们也将考虑把一个数组排列到位的问题以及该问题与排列的圈结构之间的关系。其中某些分析将导出有关第3章中特殊数的常见生成函数，例如，Stirling数与调和数的生成函数。

本章我们还将讨论模拟树高的问题，包括求一个随机排列中最短圈和最长圈的平均长度问题。至于树高，我们将建立关于配置了“垂直”下标的生成函数的函数方程，但渐近估计已经利用更先进的工具得到了最好的结果。

排列性质的研究阐明了在算法分析中平凡问题和难问题之间确实存在着一个令人满意的分界线。我们所考虑的某些问题可以通过初等论证方法很容易地加以解决；其他（类似的）问题虽然不是初等的，但可以通过我们一直都在关注的生成函数和渐近方法来研究；还有一

些其他（仍然是类似的）问题则需要用到高等复分析方法或概率方法来研究。

6.1 排列的基本性质

排列可以用许多方法来表示，最直接的方法就是对整数1到N的简单重新排列，正如下面的例子一样：

300

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

想像排列的一种方式就是我们可以把排列看成是对一个重排的具体规定："1到9、2到14、3到4"等等。显然，N个元素具有$N!$ 个不同的排列。排列有着许多基本性质，这些性质不仅从组合学观点来看有着内在的重要意义，而且它们在许多重要算法的研究中也起着重要的作用。本节我们将定义这些性质并给出一些基本结果；我们将在以后各节中讨论这些结果是如何导出来的，并把它们与算法分析联系起来。

我们将研究逆序、左向右最小值和最大值、圈、上升、游程、降落、峰、谷以及排列中的递增子序列；排列的逆排列；以及叫做对合与错位排列的特殊类型的排列。这些问题都将通过对整数1到N的一个排列$p_1p_2p_3\cdots p_N$来进行解释，在下面的定义和正文中，我们还要引用上面的样本排列。

定义 一个逆序是$i<j$且$p_i>p_j$的一个数对。如果q_j是$i<j$且$p_i>p_j$的个数，则称$q_1q_2\cdots q_N$是$p_1p_2\cdots p_N$的一个逆序表。我们用记号$\sigma(p)$来表示一个排列p中逆序的个数，即逆序表中各项的和。

上面给出的样本排列具有49个逆序，这可以通过在它的逆序表中填入元素而得以证实。

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
排列	9	14	4	1	12	2	10	13	5	6	11	3	8	15	7
逆序表	0	0	2	3	1	4	2	1	5	5	3	9	6	0	8

根据定义，一个排列的逆序表$q_1q_2\cdots q_N$中的项满足：对所有1到N的j，有$0\le q_j\le j$。正如我们将在6.3节中看到的那样，从满足这些约束的任意一个数的序列中可以构造出唯一一个排列。也就是说，在长度为N的逆序表和N个元素的排列之间存在着一一对应关系。这种对应将在本章后面基本排序算法的分析中加以利用，如插入排序和冒泡排序。

定义 左向右最大值是一个下标i，它满足对所有的$j<i$，都有$p_j<p_i$。我们用符号$\lambda(p)$来表示排列p中左向右最大值的个数。

301

所有排列中第一个元素都是一个左向右最大值；最大元素也是一个左向右最大值。如果最大元素是第一个元素，那么它是唯一的一个左向右最大值；否则至少有两个左向右最大值（第一个元素和最大元素）。一般说来，排列中可以存在多达N个左向右最大值（如在排列$12\cdots N$中）。在上面给出的样本排列中，有3个左向右最大值，它们的位置是1、2和14。注意：每个左向右最大值都对应于逆序表中的一个零（其左边不存在更大的元素），因而统计排列中左向右最大值的个数就是统计逆序表中零的个数。左向右最小值、右向左最小值以及右向左最大值的定义方法是类似的。在概率论中，左向右最大值也叫记录，这是因为当我们在排列中从左向右移动时，它们代表我们遇到了新"记录"的大数值。

习题6.1 解释如何根据逆序表计算左向右最小值、右向左最小值和右向左最大值。

定义　圈是一个下标序列$i_1 i_2 \cdots i_t$，其中$p_{i_1}=i_2, p_{i_2}=i_3, \cdots, p_{i_t}=i_1$。我们用符号$(i_1 i_2 \cdots i_t)$来表示一个圈。在长度为$N$的排列中，一个元素属于唯一的一个长度为1到$N$的圈；长度为$N$的排列是由一组长度为1到$N$的圈所构成的。错位排列是一个不存在圈长为1的排列。

我们的样本排列是由四个圈所构成的（其中的一个长度为1），因而它不是一个错位排列。

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
排列	9	14	4	1	12	2	10	13	5	6	11	3	8	15	7
圈	(1	9	5	12	3	4)	(2	14	15	7	10	6)	(8	13)	(11)

圈可以读成“1到9到5到12到3到4到1”等等。此排列中最长的圈长为6（它有两个这样的圈）；最短的圈长为1。对任何长度为t的圈，有t种等价的列出方式，构成一个排列的圈可以按任何顺序列出。也就是说，作为组合对象，排列等价于圈的集合，这一点我们曾在3.9节中检验过。

如果在每个圈中我们选择首先列出最大元素（圈头（cycle leaders））、并按圈头的递增顺序列出所有的圈，那么我们就得到了一个规范型，它有一个非常有趣的性质：小括号不是必需的，因为在规范型中，每个左向右最大值对应于一个新的圈。 302

圈	(1	9	5	12	3	4)	(2	14	15	7	10	6)	(8	13)	(11)
规范型	11	12	3	4	1	9	5	13	8	15	7	10	6	2	14

这就构成了一个组合证明：对随机排列而言，圈和左向右最大值具有相同的分布，我们在本章也将分析性地验证这一事实。在组合学中，这一事实叫做“Forta对应”或“基本对应”。

习题6.2　把我们的样本排列用圈的符号写出时，有多少种不同的方式？

习题6.3　元素为$2n$的排列中有多少个恰好有两个圈，且每个圈的长度为n？有多少个恰好有n个圈，且每个圈的长度为2？

习题6.4　在n个元素的排列中，当用圈来表示时，哪些排列具有最多的不同表示方法？

定义　排列$p_1 p_2 \cdots p_N$的逆排列是排列$q_1 q_2 \cdots q_N$满足$q_{p_i}=p_{q_i}=i$。排列的对合是这样的一个排列，其逆排列就是它自己：$p_{p_i}=i$。

对我们的样本排列而言，1在位置4、2在位置6、3在位置12、4在位置3，等等。

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

由定义可知，每个排列都有唯一的一个逆排列，且一个逆排列的逆排列就是原来的排列。下面的例子是一个对合，其圈形式的表示方法揭示了对合的重要性质。

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
对合	9	2	12	4	7	10	5	13	1	6	11	3	8	15	14
圈	(1	9)	(2)	(3	12)	(4)	(5	7)	(6	10)	(8	13)	(11)	(14	15)

显然，一个排列是一个对合，当且仅当其所有圈的长度是1或2。要对长度为N的对合个数确定出一个准确的估计，这将导致一个有趣的问题，该问题阐述了我们在本书中将要考查的大部分工具。 303

定义　上升是$p_i<p_{i+1}$的一次出现，降落是$p_{i-1}>p_i$的一次出现。游程是排列中一个最

大的、递增的连续子序列。峰是$p_{i-1}<p_i>p_{i+1}$的一次出现，谷是$p_{i-1}>p_i<p_{i+1}$的一次出现。双升是$p_{i-1}<p_i<p_{i+1}$的一次出现，双落是$p_{i-1}>p_i>p_{i+1}$的一次出现。我们用符号$\alpha(p)$来表示排列p中上升的个数。

在任何一个排列中，上升数与降落数之和都等于排列的长度减1。游程数比降落数多1，因为在一个排列中，除最后一个游程外，每个游程都必须以一个降落结束。如果我们考虑一个排列中对应于$N-1$个相邻两数之差的正、负号的符号表示法的话，上述事实以及其他事实都是明显的；降落对应+，上升对应−。

排列	9	14	4	1	12	2	10	13	5	6	11	3	8	15	7
上升/降落	−	+	+	−	+	−	−	+	−	−	+	−	−	+	

此表示法告诉我们：该排列中有8个上升和6个降落。除此之外，正号标志着游程的结束（最后一个除外），所以该排列有7个游程。双升、谷、峰和双落分别对应于− −、+−、− +、和+ +的出现。所以该排列有3个双升、5个谷、5个峰和1个双落。

定义 排列中的一个递增子序列是下标$i_1, i_2, \cdots i_k$的一个递增序列，且这些下标满足$p_{i_1}<p_{i_2}<\cdots<p_{i_k}$。

习惯上，空子序列被认为是“递增的”。因此，递增排列1 2 3 …N有2^N个递增子序列，其中每一个对应于下标的某个集合，递减排列$N\ N-1\ N-2\cdots 1$只有$N+1$个递增子序列。我们可以用类似于逆序的方法统计一个排列中的递增子序列：做一个关于$s_1s_2\cdots s_N$的表，用s_i表示起始于位置i的递增子序列的个数。我们的样本排列中有9个起始于位置1的递增子序列、2个起始于位置2的递增子序列，等等。

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
排列	9	14	4	1	12	2	10	13	5	6	11	3	8	15	7
子序列表	9	2	33	72	4	34	5	2	8	7	2	5	2	1	1

304

例如，子序列表中的第五项对应4个递增子序列：12，12 13，12 15，和12 13 15。累加该表中的各项（加上一个空子序列），就表明了我们的样本排列中递增子序列的个数是188。

习题6.5 编写一个程序，在多项式时间内计算一个给定的排列中递增子序列的个数。

表6-1对4个元素的所有排列列举了上述所有性质，表6-2对9个元素的几个随机排列给出了这些性质的值。仔细检查表6-1和表6-2就会揭示出排列的这些不同性质的特征以及本章我们将要证明的这些特征之间的关系。例如，我们已经提到过，左向右最大值个数的分布与圈数的分布是相同的。

表6-1 4个元素所有排列的基本性质

排列	子序列	逆序数	左-右最大值	圈	最大圈	游程	对合表	逆排列
1234	15	0	4	4	1	1	0000	1234
1243	13	1	3	3	2	2	0001	1243
1324	12	1	3	3	2	2	0010	1324
1342	9	2	3	2	3	2	0002	1423
1423	9	2	2	2	3	2	0011	1342
1432	7	3	2	3	3	3	0012	1432
2134	11	1	3	2	3	2	0100	2134
2143	9	2	2	2	2	3	0101	2143

（续）

排列	子序列	逆序数	左-右最大值	圈	最大圈	游程	对合表	逆排列
2314	9	2	3	2	3	2	0020	3124
2341	8	3	3	1	4	2	0003	4123
2413	7	4	2	1	4	2	0013	3142
2431	6	4	2	2	3	3	0013	4132
3124	8	1	2	2	3	2	0010	2314
3142	7	3	2	1	4	3	0002	2413
3214	7	3	2	3	2	3	0120	3214
3241	6	4	2	2	3	3	0003	4213
3412	6	4	2	2	2	2	0022	3412
3421	5	5	2	1	4	3	0023	4312
4123	8	3	1	1	4	2	0111	2341
4132	7	4	1	2	3	3	0112	2431
4213	6	4	1	2	3	3	0122	3142
4231	5	5	1	3	2	3	0113	4231
4312	5	5	1	1	4	3	0122	3421
4321	4	6	1	2	2	4	0123	4321

305

表6-2　9个元素的几个随机排列的基本性质

排列	逆序数	左-右最大值	圈	最大圈	游程	对合表	逆排列
961534872	21	1	2	6	7	012233127	395642871
412356798	4	5	5	3	4	011100001	234156798
732586941	19	3	4	6	3	012102058	932846157
236794815	15	5	2	3	7	000003174	812693475
162783954	13	5	4	5	4	001003045	136982457
259148736	16	3	2	4	4	000321253	418529763

凭直觉，我们期望上升和降落是等可能的，因此在一个长度为N的随机排列中，上升和降落都应该有大约$N/2$个。类似地，对每个元素而言，我们也将期望大约有一半比它大的元素位于其左方，因此，逆序数应该大约是$\sum_{1\le i\le N} i/2$，即：大约为$N^2/4$。我们将看到如何准确地量化这些量，从而计算出这些量的其他的矩，并用类似的技巧来研究左向右最小值和圈。

当然，如果要寻问更细节性的问题，那将要导出更为复杂的解析问题。例如，对合在排列中占多大比例？错位排列占多大比例？三个以上元素的排列中有多少个排列不含圈？少于三个元素的排列中有多少个不含圈？逆序表中最大元素的平均值是多少？一个排列中递增子序列的个数是多少？一个排列中最长圈的平均长度是多少？最长游程的平均长度是多少？这类问题将出现在我们将要进行的某些特定算法的研究中，同时我们也把它们列在了组合学文献中。

本章我们将回答这些问题中的大部分问题。表6-3归纳了我们将要得出的一些平均情形的结果。其中一部分结果的分析是相当直接的，但其他分析则需要更加先进的工具，正如我们将要看到的，当使用生成函数推导贯穿于本章内容的这些结果和其他结果时，我们还将在某种程度上考查它们与排序算法之间的关系。

表6-3 关于排列性质的累积总数及平均数

	2	3	4	5	6	7	准确平均	渐近估计
排列	2	6	24	120	720	5040	1	1
逆序	1	9	72	600	5400	52920	$N(N-1)/4$	$\sim N^2/4$
左-右最大值	3	11	50	274	1764	13068	H_N	~lnN
圈	3	11	50	274	1764	13068	H_N	$\sim \ln N$
上升	6	36	48	300	2160	17640	$(N-1)/2$	$\sim N/2$
递增子序列	5	27	169	1217	7939	72871	$\sum_{k>0}\binom{N}{k}/k!$	$O(e^{\sqrt{N}})$

6.2 排列的算法

就排列的本质而言，它们会直接或间接地出现在各种各样算法的分析当中。排列规定了数据对象排列顺序的方式，而许多算法需要按照某种规定的顺序处理数据。通常，一个复杂的算法往往要在某一阶段调用一个排序的过程，且排列与排序算法之间的直接关系也足以激发我们去详细地研究排列的性质。我们还将考查大量相关的例子。

排序。正如在第1章所见到的那样，我们经常假定对一个排序算法的输入是具有不同关键字的一系列随机顺序的记录。关键字的随机顺序特别要由任何一个能把关键字从连续分布中独立抽取出来的过程来产生。这就使得对排序算法的分析基本上等同于对排列性质的分析。在Knuth[10]从综合范畴开始的一书中，包含了有关这一课题的大量文献（例如，参见Gonnet and Baeza-Yates[5]）。适用于不同情形的排序算法有许许多多，而算法分析在我们理解它们的相对性能方面起到了至关重要作用。本章我们将学习排列的几个最基本的性质与几个主要的基本排序算法之间的直接联系。

习题6.6 设a_1、a_2、a_3是独立生成的0到1之间的“随机”数，并把它们看作满足连续分布$F(x)=Pr\{X\leq x\}$的随机变量X的值。证明事件$a_1<a_2<a_3$的概率是1/3!。把此结论推广到任意顺序的模式和任意多个关键字的情况。

306~307

重排。正如我们前面所提到过的，想像排列的一种方法就是可以把排列看成是对一个重新排列所做的具体规定，这就导致了排列与排序行为之间的直接联系。排序算法常常是通过间接地引用被排序的数组来实现的，而不是把元素移来移去、从而才使它们有序，我们是要计算出一个排列，这个排列能把元素排成顺序。实际上，任何排序算法都可以这样来实现：就我们所见过的算法而言，都是在维护一个存放排列的“下标”数组p[1]...p[N]。开始时，令p[i]=i，然后我们修改排序代码，对任何一个比较运算，我们引用a[p[i]]而不是a[i]，而在做任何数据移动时，我们引用p而不是a。这些改变确保在执行算法的任何时刻，a[p[1]], a[p[2]], …, a[p[N]]与原始算法中的a[1], a[2],…, a[N]是相同的。

例如，如果一个排序算法用这种方式把样本输入文件

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
关键字	29	41	77	26	58	59	97	82	12	44	63	31	53	23	93

排成递增序列，那么它将产生如下排列结果

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

解释这一结果的一种方法是：对原始输入可以通过首先输出第九个元素（12），然后第十四个元素（23），然后第四个元素（26），然后第一个元素（29），等等而以递增顺序完成输出

（或访问）。在此例中，注意我们所得到的排列是代表原始关键字顺序的那个排列的逆排列，即排列：

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

也就是说，排序相当于求一个排列的逆排列。

如果我们有一个输出数组b[1]...b[N]，那么实际完成一个排序的程序则是平凡的：

```
for i:=1 to N do b[i]:=a[p[i]]
```

对于数据移动开销太高的排序应用（例如，记录比关键字大得多时），这一改变将非常重要。如果没有空间可用于输出数组的话，那么重新排列使之“就位”仍然是可行的，我们将在本章稍后给出一个这样的算法。

308

有趣的是：注意排序的结果当然可以是一个对合。例如，排列

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

既可以代表输入文件的原始顺序

```
58  23  77  29  44  59  31  82  12  41  63  26  53  97  93
```

也可以代表规定如何重排文件并使之有序的排列。就是说，该排列可以用两种方式来解释：我们既可以说58是第九小的元素、23是第二小的元素、77是第十二小的元素等等，依此类推，也可以说文件中第九个元素（12）是最小的、第二个元素（26）是第二小的、第十二个元素（29）是第三小的等等，依此类推。

随机化。在不知道输入顺序是不是随机顺序的情况下，我们可以利用程序6.1所示的方法来建立一个随机顺序，然后再对数组进行排序。这一*随机化技术*采取一个能在给定范围内生成“随机”整数的过程（这样的程序已得到了很好的研究，参见Knuth[9]）。$N!$个输入顺序中的每一个都等可能地出现：第i次循环时，i个不同的重排中的任何一个都可能发生，其总数为$2\cdot3\cdot4\cdots\cdots N = N!$。正如在第1章中所提到过的，从算法分析的观点来看，用这种方法对输入顺序的随机化能把任何一个排序算法变成一个“概率算法”，其性能特征可由我们所研究的平均情形的结果来准确描述。事实上，这是最早的概率算法之一，因为它是由Hoare于1960年为Quicksort算法所提出来的（见第1章）。

程序6.1　随机排列一个数组

```
for i:=2 to N do
   begin
   t:=randominteger(1,i);
   v:=a[t]; a[t]:=a[i]; a[i]:=v;
end
```

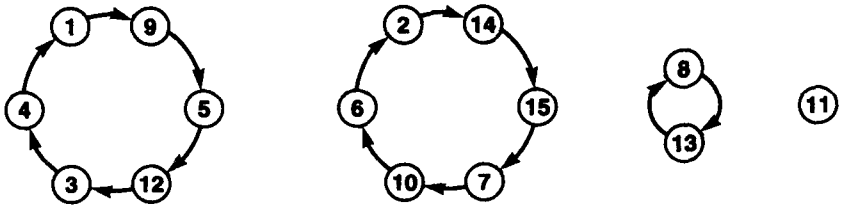
309

优先队列。在一个算法检验任何一个输入之前，并不总是必须把算法的输入重新排列成有序的顺序。另一个被广泛使用的方法是建立一个数据结构，它是由带有关键字的记录所构成的，它支持两种操作：把一个新项*插入*到数据结构中和*删除最小项*，即从数据结构中提取具有最小关键字的记录。这样的数据结构叫做*优先队列*。优先队列算法和排序算法是紧密联系的：例如，我们只需通过插入所有记录，然后再把它们删除的办法，就可以利用任何一个优先队列算法来实现排序算法。（记录将按递增的顺序输出。）然而优先队列算法更具有一般性，它们之所以能够得到广泛的应用，不仅仅因为插入和删除操作可以混合进行，还因为它们也支持其他几种操作。在算法分析的研究中，完善的优先队列算法性质的研究是最重要也是最具挑战性的领域之一。本章，我们将探讨一个重要的优先队列结构（堆序树）与二叉查

找树之间的关系，通过把它们都与排列相联系，这一关系将变得一目了然。

6.3 排列的表示法

尽管把排列表示成数1到N的重新排列通常是最方便的方法——我们一直都在这样做，然而许多其他的排列表示法却常常更为恰当。我们将看到，各种不同的表示法可以展示出排列的基本性质之间的关系，可以揭示关于某些特定算法分析的基本性质。由于N个元素有$N!$个不同的排列，因而具有$N!$个不同组合对象的任何集合都可能用来表示排列，本节我们将考查几个有用的表示法。



310

图6-1 排列的图表示法

圈结构。把一个排列$p_1p_2\cdots p_N$看成是一个把i映射到p_i的函数，将立即导出一个图的表示法，如图6-1所示。这种表示法的好处在于它使圈结构变得明显了。此外，一般函数的对应结构具有许多有趣的性质，我们将在第8章中研究它们。

*线性表示法。*排列与其圈结构表示法的规范型之间的基本对应定义了排列的一个“表示法”：正如上节所讨论过的，我们可以通过用一种规范型的线性形式写出排列的圈来表示一个排列，这种表示法是通过先识别每个圈的“圈头”（最大元素），然后按先列出圈头的顺序写出每一个圈，最后按圈头递增的顺序列出所有的圈的方法而得到：

排列	9	14	4	1	12	2	10	13	5	6	11	3	8	15	7
圈的形式	11	12	3	4	1	9	5	13	8	15	7	10	6	2	14

习题6.7 我们也可以按照先列出圈中最小元素的顺序写出每个圈，再按这些最小元素递增的顺序排列所有的圈。用这种表示法表示上述例子中的排列。

习题6.8 编写一个程序，对给定的一个排列，求其对应的规范型的圈表示式。

习题6.9 编写一个程序，对给定的一个规范型的圈表示式，求其对应的排列。

逆序表。排列与N个整数$q_1q_2\cdots q_N$，$0 \le q_i < i$的表列之间的一对一的关系很容易建立。给定一个排列，它的逆序表就是这样的一个表列；给定这样的一个表列，其对应的排列可以从右向左构造出来：对i从N递减到1，置p_i为未曾用过的数中第q_i大的数。考虑下面的例子：

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
逆序表	0	0	2	3	1	4	2	1	5	5	3	9	6	0	8
排列	9	14	4	1	12	2	10	13	5	6	11	3	8	15	7

排列可以通过在逆序表中从右向左移动构造出来：7是1到15中第八大的整数，15是剩余整数中最大的，8是剩余整数中第六大的，等等。因为对第i项输入有i种可能，所以共有$N!$个逆序表（和排列！）。

311

这种对应在分析中很重要，因为一个随机排列等价于一个“随机”逆序表，该逆序表是通过对其第j个输入项赋以0到$j-1$之间的一个随机整数而生成的。在建立关于逆序和以乘积的形式巧妙地分解左向右最大值的GF时，我们将利用到这一事实。

习题6.10　给出一个有效的算法，求对应于一个给定排列的逆序表；再给出一个算法，求对应于一个给定逆序表的排列。

习题6.11　定义逆序表的另外一种方法是令q_i等于排列中i的左方大于i的整数的个数。对这种逆序表证明一对一的对应关系。

格表示法。图6-2给出了一个二维表示法，它对研究排列的许多性质都是很有用的：排列$p_1p_2\cdots p_N$是通过对每个i，在第p_i行、第i列所对应的单元格中标上数p_i的方法来表示的。从右向左读这些数就会回到该排列。每行上和每列上都有一个标号，因此每一个单元格都对应唯一的一对标号：一个在它的行上，一个在它的列上。如果一对标号中的一个成员在单元格的下方而另一个在单元格的右方，那么这个标号对就是排列的一个逆序，对应的单元格在图6-2中都用圆点做上了记号。特别要注意的是：排列和其逆排列的图形是相互转置的，这也是下述结论的一个初等证明——每个排列都和它的逆排列有相同数量的逆序数。

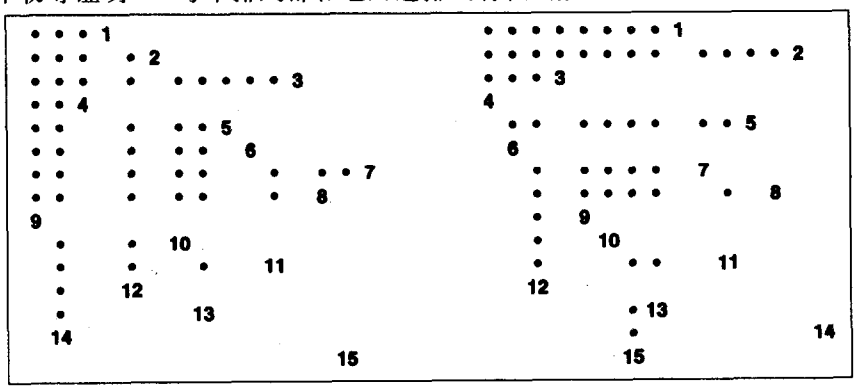


图6-2　排列和其逆的格表示法

312

习题6.12　证明在一个$N\times N$格的棋盘上放入k个相互非攻击型“车”的方式数为$\binom{N}{k}^2 k!$。

习题6.13　在格表示法中，假设我们对单元格的成员在其上方和其左方的单元格作标记，那么有多少个单元格被做了标记？对另外两种可能（“上和右”及“下和左”）回答相同的问题。

习题6.14　证明对合的格表示法关于主对角线是对称的。

二叉查找树。图6-3给出了排列与二叉查找树之间的一个直接关系。回忆5.5节，二叉查找树（BST）中每个结点都有一个关键字、一个指向由较小关键字构成的BST的左链（可能是空的）和一个指向由较大关键字构成的BST的右链（可能是空的）。在第5章中，我们分析了用程序5.3从一个随机排列向一个初始为空的树中相继插入关键字而生成的树的性质。图6-3阐明了排列的格表示法和BST树之间的一个直接对应关系：每个标号对应一个结点，标号的行号对应关键字的值，该标号上方和下方的部分标号分别生成结点的左右子树。特别地，二叉查找树对应$l, l+1,\cdots, r$行，且k行最左边（最小列号）的标号是由关键字k递归定义的，左子树对应行$l, l+1,\cdots, k-1$，右子树对应行$k+1, k+2,\cdots, r$。注意，可能有许多个排列对应同一棵二叉查找树：将对应于最上方结点的列号与对应于最下方结点的列号互换，将改变一个排列，但不改变树。事实上，在第3章中我们曾经看到过，不同二叉树结构的个数是由Catalan数来统计的，与$N!$相比，该数（大约为$4^N/N\sqrt{\pi N}$）还是比较小的，因此每一棵树肯定要有许多排列必然与之对应。上一章中关于BST树的解析结果或许就是以研究这种关系的本质为特征的。

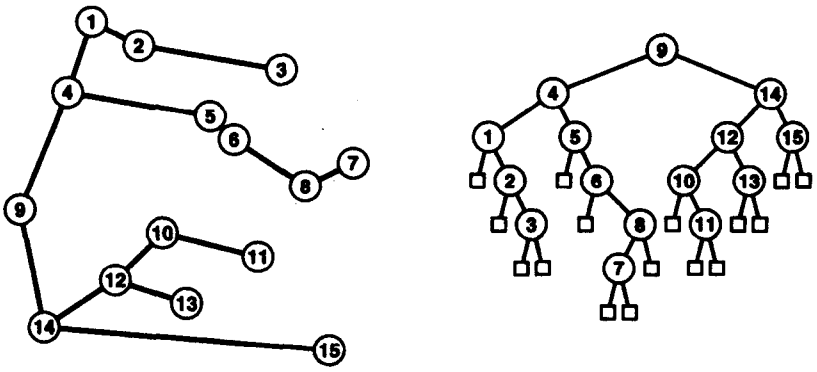


图6-3 对应于一个排列的二叉查找树

习题6.15 列出五个对应于图6-3中BST树的排列。

堆序树。正如图6-4所表明的那样，树也可以用类似的涉及列的方式、通过格表示法来建立。树中根结点关键字的值小于其子树中关键字的值，这样的树叫做*堆序树*（HOT）。堆序树对我们当前的学习内容非常重要，因为我们所关注的排列的性质可以很容易地看成是树的性质。例如，带有两个非空子树的结点对应于排列中的谷，而叶结点对应于排列中的峰。

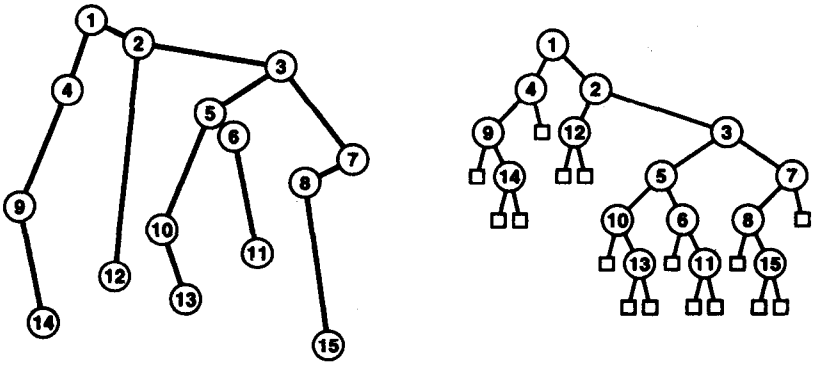


图6-4 对应于一个排列的堆序树

以这种树作为数据结构来实现优先队列也有很多好处。最小的关键字在根处，所以“删除最小的”操作可以这样来（递归地）实现：将该根结点用具有较小关键字的子树的根结点来取代。“插入”操作可以这样来（递归地）实现：在树中插入一个新结点，如果新结点的关键字不小于根结点的关键字，则将新结点插在根结点的右子树中；否则将新结点变成根结点，原来的根结点变成新结点的左子树，新结点的右子树为空。

313～314

从组合学的角度来看，注意到HOT树是唯一一个排列的完全编码是很重要的（这一点与BST树相反）。关于HOT树更详细的知识以及其他的应用，可参见Vuillemin[17]。

HOT树的直接计数。为了实现圈，考查HOT树的直接计数问题肯定会对我们有所启发（我们已经知道它们有$N!$个）。设$\mathcal{H}$是HOT树的集合，并考虑EGF

$$H(z)=\sum_{t\in\mathcal{H}}\frac{z^{|t|}}{|t|!}$$

现在，每个具有$|t_L|+|t_R|+1$个结点的HOT树都可以通过唯一的方法构造出来：将任意一棵位于左边大小为$|t_L|$的HOT树与位于右边的任意一棵大小为$|t_R|$的HOT树联合，将根赋以标号“1”，

通过将$|t_L| + |t_R|$个标号分成大小分别为$|t_L|$和$|t_R|$的两个集合来对子树进行标号：对每个集合排序，将集合元素以下标的顺序对两棵子树进行标号。该方法本质上是对第3章中所描述的标号乘积结构的一个重述。根据EGF，这就导致方程

$$H(z) = \sum_{t_L \in \mathcal{H}} \sum_{t_R \in \mathcal{H}} \binom{|t_L| + |t_R|}{|t_L|} \frac{z^{|t_L|+|t_R|+1}}{(|t_L| + |t_R| + 1)!}$$

对方程两边微分，立即得到

$$H'(z) = H^2(z) \qquad \text{或} \qquad H(z) = 1 + \int_0^z H^2(t)\mathrm{d}t$$

该公式也可以通过关于标号对象的符号法直接得出（见第3章和[4]）。两种基本操作对该公式的解释是：对根赋以标号1对应于积分，合并两棵子树对应于一个乘积。现在，微分方程的解是$H(z) = 1/(1 - z)$，这就检验了我们的知识：关于N个结点，有$N!$个HOT树。

类似的计算将导出以HOT树表示法为特征的排列的参数统计量（如峰、谷、上升和降落），正如我们下面的习题中所采用的、以及6.5节中要进一步讨论的计算一样。 315

习题6.16 描述HOT树中对应于排列中的上升、双升、降落和双落的结点的特征。

习题6.17 有多少个排列是严格交替的：对$1<i<N$，要么p_{i-1}和p_{i+1}都小于p_i，要么p_{i-1}和p_{i+1}都大于p_i？

习题6.18 列出对应于图6-4中的HOT树的5个排列。

习题6.19 设$K(z) = z/(1 - z)$是关于非空HOT树的EGF，给出一个直接论证，证明$K'(z) = 1 + 2K(z) + K^2(z)$。

习题6.20 用类似于以EGF对HOT树计数的论证方法导出二叉查找树中关于内部路径长度的指数CGF的微分方程（参见定理5.5的证明及6.5节）。

有多少个排列与给定的一个经由HOT或BST构造而成的二叉树相对应？这一问题的结果可以用一个简单的公式来表示。给定一棵树t，令$f(t)$是把t标记成一棵HOT树的方法数。根据刚才的论证，此函数满足下面的递归方程

$$f(t) = \binom{|t_L| + |t_R|}{|t_L|} f(t_L) f(t_R)$$

如果f是对应于一个BST的排列的个数的话，此公式同样成立，这是因为存在唯一一个对根（$|t_L| + 1$）的标号以及一个关于子树的剩余标号的自由划分。注意：t中结点的个数是$|t_L| + |t_R| + 1$，方程两端均除以此数，我们得到

$$\frac{f(|t_L| + |t_R| + 1)}{(|t_L| + |t_R| + 1)!} = \frac{1}{|t_L| + |t_R| + 1} \frac{f(t_L)}{t_L!} \frac{f(t_R)}{t_R!}$$

这就直接导出了下面的定理：

定理6.1 (HOT树和BST树的频数) 无论树t是HOT形的还是BST形的，其频数都是

$$f(t) = |t|!/(|u_1||u_2|\cdots|u_{|t|}|)$$

其中$u_1, \cdots, u_{|t|}$是以$|t|$个结点中的每一个结点为根的子树。换句话说，$f(t)$是用标号$1\cdots|t|$将t标记成一个HOT树的方式数，也是当使用标准算法时，导出建立一个t形BST树的$1\cdots|t|$的排列数。 316

证明 利用初始条件：$|t| = 1$时$f(t) = 1$，对上面给定的递归公式进行迭代即可。 ■

例如，标记图6-4中HOT树的方式数是15!/(15 · 3 · 2 · 11 · 9 · 5 · 2 · 2 · 3) = 1223040。

注意，HOT树和BST树频数一致的事实也是组合学中所期望的。由图6-3和图6-4所阐明的两种对应关系在结构上是相同的，只不过是两个轴互换罢了。换句话说，一个更强的性质也成立：对应于一个排列的BST与对应于其逆排列的HOT具有相同的形状（但不具有相同的标号）。

习题6.21 有多少个排列与图6-3中的BST树对应？

习题6.22 描述大小为N，频数最小及频数最大的二叉树的特征。

6.4 计数问题

本章引言部分所列举的许多问题都是计数问题。就是说，在我们所关注的各种性质中，我们想知道满足某个特定性质的排列数有多少。相应地，将其除以$N!$ 就可以得到满足某个特定性质的排列数的概率。我们配合对排列计数的EGF来进行讨论，这与讨论关于概率的OGF是等价的。

排列中大量有趣的性质可以通过对圈长的简单约束而表示出来，因此我们从详细考查这样的问题入手。我们很想知道：对一个给定的参数k，具有下列约束的排列的准确个数是多少（i）只有长度为k的圈；（ii）不存在长度大于k的圈；和（iii）不存在长度小于k的圈。本节，我们将对这些计数问题给出分析结果。表6-4给出了$N \leq 10$和$k \leq 4$时的统计数据。

表6-4 有圈长约束的排列的计数

圈长	1	2	3	4	5	6	7	8	9
全部 = 1	1	1	1	1	1	1	1	1	1
=2	0	1	0	3	0	15	0	105	0
= 3	0	0	2	0	0	40	0	0	2240
= 4	0	0	0	6	0	0	0	1260	0
<3（对合）	1	2	4	10	26	76	232	764	2620
<4	1	2	6	18	66	276	1212	5916	31068
<5	1	2	6	24	96	456	2472	14736	92304
>1（错位）	0	1	2	9	44	265	1854	14833	133496
>2	0	0	2	6	24	160	1140	8988	80864
>3	0	0	0	6	24	120	720	6300	58464
不受限	1	2	6	24	120	720	5040	40320	362880

长度相等的圈。只由长度为k的圈所构成的长度为N的排列一共有多少个？我们先对$k = 2$时给出一个简单的组合论证。如果N为奇数，则只含双元素圈构成的排列有零个，所以我们考虑N是偶数的情况。我们选出一半元素作为每个圈的第一个元素，然后指定第二个元素，则指定方式有（$N/2$）！种，因为不用考虑元素的顺序，所以每个排列统计了$2^{N/2}$次。由此可得出长度为2的圈所构成的排列总数为

$$\binom{N}{N/2}(N/2)!/2^{N/2} = \frac{N!}{(N/2)!2^{N/2}}$$

上式乘以z^N并除以$N!$后，我们就得到其指数生成函数

$$\sum_{N\text{为偶数}} \frac{z^N}{(N/2)!2^{N/2}} = e^{z^2/2}$$

正如我们在第3章中所见到的那样，符号法也能直接给出这个EGF，它利用的是“集合的”构造并观察长度为2的圈的EGF是$z^2/2$而得到的。对此结论进行推广，就可以证明仅由长度为k的圈构成的排列数的EGF是$e^{z^k/k}$。欢迎读者复习第3章中有关标号对象符号法的讨论，特别是对$k=1$和$k=2$来思考这个公式，从而了解“集合的”构造是如何工作的。

对合与圈长的上界。由于e^z是仅由单元素圈构成的排列数的EGF，且$e^{z^2/2}$是仅由双元素圈构成的排列数的EGF，所以由长度为1或长度为2的圈构成的排列数的EGF是$e^{z+z^2/2}$。类似地，仅由长度为1或2或3的圈构成的排列数的EGF是$e^{z+z^2/2+z^3/3}$，依此类推。和第3章一样，当我们完成这种重复的分析过程时，就得到了关于无圈长约束的排列数的EGF为

$$e^{z+z^2/2+z^3/3+z^4/4+\cdots}=\exp\left(\ln\frac{1}{1-z}\right)=\frac{1}{1-z}$$

317 ~ 318

这与我们所期望的结果是一致的。

习题6.23　求仅由偶数长度的圈所构成的排列数的EGF。对此结果进行推广，求仅由长度能被t整除的圈所构成的排列数的EGF。

正如上一节所简要提到的那样，对合可以被看成是一种圈长受到约束的排列，因此刚才的论证可以应用到计数上。在一个排列中，如果$p_i=j$，那么在其逆排列中，就有$p_j=i$：在一个对合中，对任意的$i\neq j$，上述两点同时成立，或者说，圈(i,j)必定出现。这一观察隐含了：对合是由长度为2（$p_{p_i}=i$）或1（$p_i=i$）的圈所构成的。因此，对合恰好就是那些仅由单元素圈和双元素圈所构成的排列，所以关于对合数的EGF是$e^{z+z^2/2}$。

定理6.2（最大圈长）　对不含圈长大于k的排列个数计数的EGF是

$$\exp(z+z^2/2+z^3/3+\cdots+z^k/k)$$

特别地，对合个数的EGF是

$$B(z)=e^{z+z^2/2}$$

且

$$N![z^N]B(z)=\sum_{N/2\geq k\geq 0}\frac{N!}{(N-2k)!2^k k!}\sim\frac{1}{\sqrt{2\sqrt{e}}}\left(\frac{N}{e}\right)^{N/2}e^{\sqrt{N}}$$

证明　参见上面关于EGF推导的讨论。对合的个数是通过展开

$$e^{z+z^2/2}=\sum_{j\geq 0}\frac{1}{j!}\left(z+\frac{z^2}{2}\right)^j=\sum_{j,k\geq 0}\frac{1}{j!}\binom{j}{k}z^{j-k}\left(\frac{z^2}{2}\right)^k$$

并累积$[z^N]$而得到的。

对合个数的渐近形式可以由第4章的拉普拉斯方法导出。取和式中相邻两项的比值，我们有

$$\frac{N!}{(N-2k)!2^k k!}\bigg/\frac{N!}{(N-2k-2)!2^{k+1}(k+1)!}=\frac{2(k+1)}{(N-2k)(N-2k-1)}$$

319

这表明和式中的项在k近似等于$(N-\sqrt{N})/2$之前是递增的，而后是递减的。利用Stirling逼近来估计峰值附近的主要贡献，并利用一个正态逼近来限定两个尾部，其结果的导出方式与第4章各个例子中结果的导出方式是一样的。Knuth[10]中给出了推导的细节。■

直接推导对合个数的EGF也具有启发性。设$B(z)$是关于对合数的指数生成函数，因而$b_N=N![z^N]B(z)$。根据定义，

$$B(z)=\sum_{\substack{p\in\mathcal{P}\\ p\text{个对合}}}\frac{z^{|p|}}{|p|!}$$

现在，每个长度为$|p|$的对合都对应于(i)一个长度为$|p|+1$的、通过在原对合中加入由$|p|+1$所构成的单元素的圈而形成的对合；及（ii）$|p|+1$个长度为$|p|+2$的对合，这些对合是通过在原对合中对每个1到$|p|+1$的k，对排列元素大于k的元素加1，然后添加一个由k和$|p|+2$所构成的双元素圈而形成的。上述结果意味着生成函数必须满足

$$B(z)=\sum_{\substack{p\in\mathcal{P}\\ p\text{个对合}}}\frac{z^{|p|+1}}{(|p|+1)!}+\sum_{\substack{p\in\mathcal{P}\\ p\text{个对合}}}(|p|+1)\frac{z^{|p|+2}}{(|p|+2)!}$$

对其两端微分，化简后得

$$B'(z)=(1+z)B(z)$$

该方程的解为

$$B(z)=e^{z+z^2/2}$$

习题6.24 证明长度为N的对合数满足如下递推关系

$$b_{N+1}=b_N+Nb_{N-1}，当N>0且b_0=b_1=1时$$

（例如，该递推关系可以用于计算表6-4中对应于对合那一行上的各项数据。）

320 **习题6.25** 导出一个递推关系，用来计算不存在圈长大于3的排列数。

习题6.26 利用4.10节中的方法，对不存在圈长大于k的排列数导出一个关于$N^{N(1-1/k)}$的界。

错位排列与圈长的下界。关于排列最著名的计数问题恐怕就是错位排列问题了。假设N个人在剧院寄存帽子，然后按随机顺序取走。任何一个人拿走的都不是他自己的帽子的概率是多少？例如，4个元素的错位排列是如下6个含有一个圈长为4的排列

2 3 4 1　2 4 1 3　3 4 2 1　3 1 4 2　4 3 1 2　4 1 2 3

和如下3个含有双元素圈的排列

2 1 4 3　3 4 1 2　4 3 2 1

四元素的所有其他排列都至少有一个单元素的圈，所以关于$N=4$的相应概率是$9/24=0.375$。

用我们已讨论过的方法求解这一问题非常直接。问题中的概率是不含单元素圈的排列数的EGF中z^N的系数。如上，我们可以通过圈长为2, 3, 4, …来构造错位排列，其EGF为

$$e^{z^2/2+z^3/3+\cdots}=\exp\left(\ln\frac{1}{1-z}-z\right)=\frac{e^{-z}}{1-z}$$

该结果的另一种推导方法如下所述。我们按下述方法来构造所有排列的类：取仅含单元素圈构成的排列和不含单元素圈的排列的“乘积”，用符号方程表示为

$$\mathcal{P}=\mathcal{C}^{[1]}\times\mathcal{D}^{[1]}$$

该方程可以翻译成一个隐式方程，它定义了对错位排列计数的EGF $D(z)$:

$$\frac{1}{1-z}=e^zD(z)$$

因为$1/(1-z)$计数所有排列，且e^z是对单元素的圈的计数。所以

$$D(z)=\frac{e^{-z}}{1-z}$$

321

这是生成函数的一个简单的卷积，它导出如下结果

$$[z^N]D(z)=\sum_{0\le k\le N}\frac{(-1)^k}{k!}$$

在4.4节中我们曾指出过这个和式渐近于$1/e$。

定理6.3（最小圈长）　不含圈长小于等于k的排列数的EGF为

$$D^{[k]}(z)=\frac{1}{1-z}e^{-z-z^2/2-z^3/3-\cdots-z^k/k}$$

渐近地，有

$$[z^N]D^{[k]}(z)\sim e^{-H_k}$$

特别地，对于错位排列有

$$[z^N]D(z)=\sum_{0\le k\le N}\frac{(-1)^k}{k!}\sim 1/e\approx 0.36787944$$

证明　我们可以对上述两个论证中的任意一个进行推广。例如，我们可以从下面的符号方程开始

$$\mathcal{P}=\mathcal{C}^{[1]}\times\mathcal{C}^{[2]}\times\cdots\times\mathcal{C}^{[k]}\times\mathcal{D}^{[k]}$$

将其翻译成生成函数的方程

$$\frac{1}{1-z}=e^z e^{z^2/2}\cdots e^{z^k/k}D^{[k]}(z)$$

从而得到

$$D^{[k]}(z)=\frac{e^{-z-z^2/2-\cdots-z^k/k}}{1-z}$$

定理中的渐近结果是定理4.12的一个直接推论。 ■

322

为参考起见，前面所讨论过的关于有约束圈长排列的计数问题的解在表6-5中已经列出。指数生成函数以及它们系数的渐近估计也都列了出来。

表6-5　带有圈长约束的排列的EGF

	EGF	$N![z^N]$的渐近估计
单元素圈	e^z	1
长为k的圈	$e^{z^k/k}$	—
所有排列	$\frac{1}{1-z}$	$N!\sim\sqrt{2\pi N}\left(\frac{N}{e}\right)^N$
错位排列	$\frac{e^{-z}}{1-z}$	$\sim N!e^{-1}$
所有$>k$的圈	$\frac{e^{-z-z^2/2-\cdots-z^k/k}}{1-z}$	$\sim N!e^{-H_k}$

（续）

	EGF	$N![z^N]$的渐近估计
对合	$e^{z+z^2/2}$	$\sim\frac{1}{\sqrt{2\sqrt{e}}}e^{\sqrt{N}}\left(\frac{N}{e}\right)^{N/2}$
所有$\leq k$的圈	$e^{z+z^2/2+\cdots+z^k/k}$	—

习题6.27 通过对关系式$(1-z)D(z)=e^z$微分并令两端系数相等，求出N个元素的错位排列数所满足的一个递推关系。

习题6.28 写一个程序，对给定的k，打印一个不含圈长小于k的N $(N<20)$ 元素排列个数的表。

习题6.29 一个N元素的排列是由这N个元素的一个子集所形成的序列。证明关于排列的EGF是$e^z/(1-z)$。将系数表示成一个简单的和式，并从组合学上解释这个和式。

6.5 利用CGF分析排列的性质

323 本节，我们利用累积生成函数（CGF）来概述一个基本方法，利用这个方法对本章提出的多种问题的排列性质进行分析。这个方法在第3章中已经介绍过，我们可以归纳如下：

- 定义一个形如 $B(z)=\sum_{p\in\mathcal{P}}\mathrm{cost}(p)z^{|p|}/|p|!$ 的指数CGF；
- 通过“分解”排列来划分和式，找出关于$B(Z)$的另一个表达式；
- 对$B(Z)$导出一个函数方程；
- 解方程或利用解析方法求出 $[z^N]B(z)$。

第二步所说的“划分”实际上是通过在排列中找出一个对应关系来实现的，最常见的划分是将长度为$|p|$的$|p|!$个排列划分成大小为$|p|$的$|p-1|!$个集合，其中每个集合都与其中的一个长度为$|p-1|$的排列相对应。特别地，如果$\mathcal{P}_q$是对应于一个给定排列q的长度为$|q|+1$的排列的集合，那么上述第二步相当于把CGF重写如下：

$$B(z)=\sum_{q\in\mathcal{P}}\sum_{p\in\mathcal{P}_q}\mathrm{cost}(p)\frac{z^{|q|+1}}{(|q|+1)!}$$

$\mathcal{P}_q$中的排列往往都有着紧密的关系，因而内层和式很容易计算，于是就导出了关于$B(Z)$的另一个表达式。通常是对该表达式微分，以便处理方程右端的 $z^{|q|}/|q|!$。

对排列而言，这个分析在某种程度上可以简化，因为关于指数CGF的阶乘也要计入排列总数之内，所以指数CGF也是所求平均值的一个常规GF。即，如果

$$B(z)=\sum_{p\in\mathcal{P}}\mathrm{cost}(p)z^{|p|}/|p|!$$

则有

$$[z^N]B(z)=\sum_k k\ \{\text{开销为}k\text{长度为}N\text{的排列数}\}/N!$$

这恰好就是平均开销。对其他组合结构而言，我们必须用由CGF得到的累积数除以总数才能
324 得到平均值，尽管也有其他情况：累积数具有非常简单的形式，除法可以并入生成函数之内。我们将在第7章看到另外一个例子。

涉及排列之间对应关系的组合论证也常常用来导出一个递推关系，甚至是一个完整的

BGF，于是它将会给出一个更强的结果，这是因为对BGF一个参数的明确了解，意味着对值的分布的了解。然而，当使用CGF时，我们用的是平均值，对平均值的处理用不着完全掌握分布，因而常常可能导出更简单的计算。

下述划分排列的简单方法作为这种处理方式的基础是最重要的一些方法。

“最大”或“最小”对应。给定一个长度为N的排列，考虑从这个排列中通过简单地去掉最大元素而形成的那个排列。这个长度为$N-1$的新排列恰好对应N个长度为N的不同排列，其中每个排列的最大元素取每一个可能的位置。例如:

5 4 1 2 3　4 5 1 2 3　4 1 5 2 3　4 1 2 5 3　4 1 2 3 5

都对应4 1 2 3。或进一步说，给定长度为$N-1$的$(N-1)!$个排列中的任意一个，我们都能通过把N放在每个可能的位置而确定出N个长度为N的不同排列。我们可以对任何其他元素做同样的事情，而不仅仅是最大元素，只要我们适当地给其余的元素编号。例如，使用最小元素要涉及对其他每个元素加1，然后将1放入每个可能的位置。

“第一”或“最后”对应。给定一个长度为N的排列，考虑从这个排列中通过简单地去掉第一个元素、然后对其余元素重新编号并保持原有顺序而形成的一个长度为$N-1$的特定的排列。该排列恰好对应N个长度为N的不同排列，其中每个排列的第一个元素取每一个可能的值。例如:

1 5 2 3 4　2 5 1 3 4　3 5 1 2 4　4 5 1 2 3　5 4 1 2 3

都对应4 1 2 3。或进一步说，给定长度为$N-1$的$(N-1)!$个排列中的任意一个排列，我们都能通过对每个从1到N的k，在该排列的第一个位置前插入k，然后对所有大于等于k的数加1而确定出N个长度为N的不同排列。这就定义了“第一”对应。我们可以对任何其他元素做同样的事情，而不仅仅是第一个元素。例如，如果使用最后一个元素作为对应依据，那么

325

5 2 3 4 1　5 1 3 4 2　5 1 2 4 3　5 1 2 3 4　4 1 2 3 5

都对应4 1 2 3。

二叉查找树分解。给定两个排列p_l和p_r，我们可以通过（i）对p_r的每个元素加$|p_l|+1$；（ii）用所有可能的方式将p_l和p_r混合；（iii）对混合后的每个排列加上前缀$|p_l|+1$来产生长度为$|p_l|+|p_r|+1$、总个数为

$$\binom{|p_l|+|p_r|}{|p_l|}$$

的排列。正如我们在6.3节中所见过的那样，用这种方式得到的所有排列将导出利用标准算法构造的相同的二叉查找树。因此，这种对应可用作BST及其相关算法分析的基础。（见习题6.20）

堆序树分解。一个排列可以通过取最小元素左边的那些元素和最小元素右边的那些元素并重新编号而唯一地分解成“左”“右”两个排列。例如，2 5 1 4 6 3可以分解成1 2和2 3 1，而6 2 4 3 1 5可以分解成4 1 3 2和1。递归地运用这种分解，就对应了HOT树的构造。关于参数用HOT树的结构给出一种自然的解释是很有用的。

注意，所有这些分解将导致CGF的微分方程，因为去除元素对应于改变计数序列，这可翻译成对生成函数的积分（见表3-4）。

游程和上升。作为使用CGF来分析排列性质的一个例子，我们来推导一个随机排列中游程的平均数。前面给出的初等论证表明了：在一个长度为N的排列中，游程的平均个数是$(N+1)/2$，

326 但是正如欧拉所发现的那样（更多细节见[10]和[11]），研究游程的整个分布才会更有意义。表6-6和图6-5对较小的N和k阐明了这种分布。

表6-6 排列中游程的分布（欧拉数）

$N\downarrow$	$k\rightarrow 1$	2	3	4	5	6	7	8	9	10
1	1									
2	1	1								
3	1	4	1							
4	1	11	11	1						
5	1	26	66	26	1					
6	1	57	302	302	57	1				
7	1	120	1191	2416	1191	120	1			
8	1	247	4293	15619	15619	4293	247	1		
9	1	502	14608	88234	156190	88234	14608	502	1	
10	1	1013	47840	455192	1310354	1310354	455192	47840	1013	1

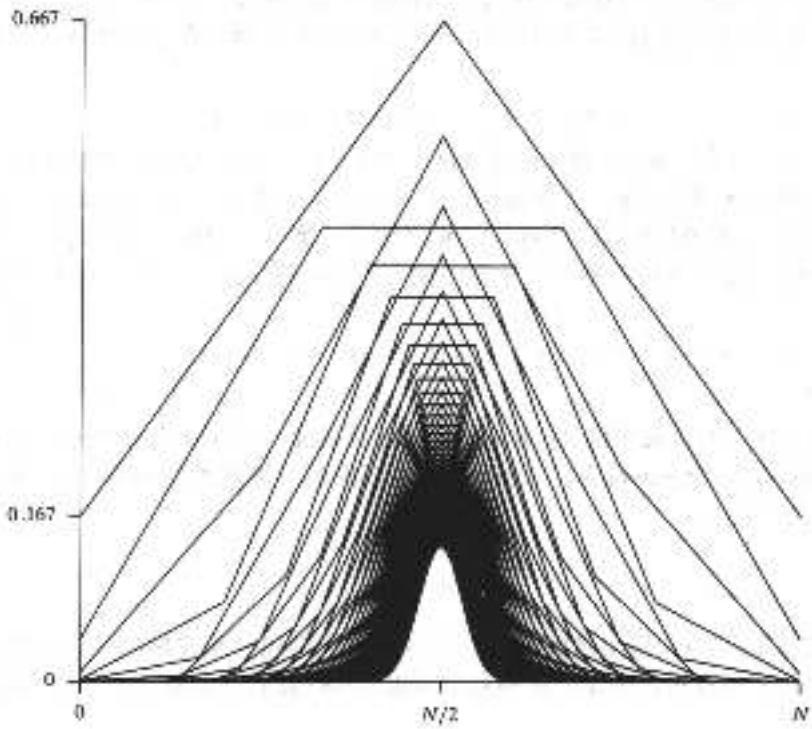


图6-5 $3\leqslant N\leqslant 60$时游程的分布（横轴按N的比例标度）（欧拉数）

我们从（指数的）CGF

$$A(z)=\sum_{p\in S}\alpha(p)\frac{z^{|p|}}{|p|!}$$

开始推导，利用“最大”对应：如果最大元素插入在p中一个游程的尾部的话，则游程数不变；否则，游程数增1。对应于给定排列p的排列中游程的总数是

$$(|p|+1)\alpha(p)+|p|+1-\alpha(p)=|p|\alpha(p)+|p|+1$$

这就导出了下面的另一个表达式

$$A(z)=\sum_{p\in\mathcal{P}}(|p|\alpha(p)+|p|+1)\frac{z^{|p|+1}}{(|p|+1)!}$$

如果对其微分，则上式将大大简化:

$$A'(z)=\sum_{p\in\mathcal{P}}(|p|\alpha(p)+|p|+1)\frac{z^{|p|}}{|p|!}$$

$$=zA'(z)+\frac{z}{(1-z)^2}+\frac{1}{1-z}$$

因此$A'(z)=1/(1-z)^3$，所以对给定的初始条件，可得

$$A(z)=\frac{1}{2(1-z)^2}-\frac{1}{2}$$

这样我们就得到了所预期的结果$[z^N]A(z)=(N+1)/2$。

本章我们还将对这个形式进行多次推导。在某些情况下（包括本次），通过对变量展开或利用符号法来找到平均游程数的二元生成函数（或一个递推关系）是有可能的，然而，CGF常常并不需要太多的计算就能给出平均值，必要时我们也能利用BGF或递推关系来找到它的标准差以及分布的其他的矩。

327
~
328

定理6.4 (欧拉数)　*具有k个游程的N元素排列可利用指数BGF*

$$A(u,z)\equiv\sum_{N\geq 0}\sum_{k\geq 0}A_{Nk}u^k\frac{z^N}{N!}=\frac{1-u}{1-ue^{z(1-u)}}$$

通过欧拉数A_{Nk}来计数。在长度$N>1$的排列中，平均游程数是$(N+1)/2$，其方差为$(N+1)/12$。

证明　利用“最大”对应，对给定的关于CGF的论证进行推广，将得到一个关于BGF的偏微分方程。我们把这个结论留作一个习题。现在我们考虑由同一个对应关系导出的基于递推关系的推导方法：要得到一个k个游程的排列，把最大元素插入到一个k个游程的排列中的一个现有游程的尾部有k种可能，使最大元素“破坏”一个$k-1$个游程的排列中的一个现有游程，从而使游程数增加到k、有$N-k+1$种可能。这就导致

$$A_{Nk}=(N-k+1)A_{(N-1)(k-1)}+kA_{(N-1)k}$$

此式和初始条件$A_{00}=1$及$A_{N0}=0$一起，对$N\geq 0$及$k>N$，就完全确定了A_{Nk}。

上式乘以z^Nu^k并对N和k求和则直接导出了下面的偏微分方程

$$A_z(u,z)=\frac{1}{1-uz}(uA(u,z)+u(1-u)A_u(u,z))$$

容易验证，定理中关于$A(u,z)$的表达式满足这个方程。

现在，我们像表3-6中那样来计算均值和方差，但正如前面所提到过的，需要注意：使用一个指数BGF时要自动地包括除以$N!$。这样，均值就可由下式给出:

$$[z^N]\frac{\partial A(u,z)}{\partial u}\bigg|_{u=1}=[z^N]\frac{1}{2(1-z)^2}=\frac{N+1}{2}$$

我们可以用类似的方法计算方差。 ■

正如前面所提到的，一个排列中除最后一个游程外，所有游程都以一个降落结束。因而定理6.4也隐含着这样的结论：一个排列中降落数的均值为$(N-1)/2$，方差为$(N+1)/12$，相同的结果也可以应用在上升数上。

329

习题6.30 给出一个简单的非计算性的证明：一个N元素的排列中上升数的均值是$(N-1)/2$。（提示：对每个排列$p_1p_2\cdots p_N$，考虑由$q_i=N+1-p_i$所构成的“补”$q_1q_2\cdots q_N$。）

习题6.31 推广上面的CGF变量，证明BGF $A(u,z)=\sum_{p\in\mathcal{P}}u^{\alpha(p)}z^{|p|}$ 满足定理6.4中给出的偏微分方程。

习题6.32 证明：

$$A_{Nk}=\sum_{0\le j\le k}(-1)^j\binom{N+1}{j}(k-j)^N$$

习题6.33 证明：对$N\geqslant 1$有

$$x^N=\sum_{1\le k\le N}A_{Nk}\binom{x+k-1}{N}\qquad(N\geqslant 1)$$

递增子序列。为CGF导出一个显式公式的另一个方法是为累积开销找出一个递推关系。例如，令

$$S(z)=\sum_{p\in\mathcal{P}}\{p\text{中递增子序列数}\}\frac{z^{|p|}}{|p|!}=\sum_{N\geqslant 0}S_N\frac{z^N}{N!}$$

那么S_N就代表所有长度为N的排列中递增子序列的总数。于是，我们发现

$$S_N=NS_{N-1}+\sum_{0\le k<N}\binom{N-1}{k}(N-1-k)!S_k\qquad\text{对}N>0\text{且}S_0=1$$

这是因为长度为$N-1$的排列有N个拷贝（所有递增子序列在那个排列中都出现N次）且在分开计数时，所有递增子序列都以最大元素结束。如果最大元素在第$k+1$位置，那么所有排列对前k个位置上元素的每一种选择出现$(N-1-k)!$次（较大元素的每种排列方式对应一次出现），每个最大元素对总数贡献S_k。该论证假定了空子序列是“递增的”，这与我们的定义是一致的。上式除以$N!$并对N求和式，我们就得到如下函数方程：

330

$$(1-z)S'(z)=(2-z)S(z)$$

其解为

$$S(z)=\frac{1}{1-z}\exp\left(\frac{z}{1-z}\right)$$

定理6.5（递增子序列） 在一个N元素的随机排列中，递增子序列数的均值为

$$\frac{S_N}{N!}=\sum_{0\le k\le N}\binom{N}{k}\frac{1}{k!}\sim\frac{1}{2\sqrt{\pi e}}e^{2\sqrt{N}}/N^{1/4}$$

证明 这个准确公式直接来自于上面的讨论，展开此显式公式就可以得到刚才给出的生成函数。

此外，我们可用拉普拉斯法导出该公式的渐近估计。取相邻两项的比值，我们有

$$\binom{N}{k}\frac{1}{k!}\Big/\binom{N}{k+1}\frac{1}{(k+1)!}=\frac{(k+1)^2}{N-k}$$

该比值表明，当k约等于$\sqrt{N}$时，就会出现一个峰。此外，和第4章中的几个例子一样，Stirling公式提供了局部逼近，而两个尾部可由一个正态逼近来限定。有关细节可在Lifschitz和Pittel[12]中找到。 ■

习题6.34　对S_N的准确公式给出一个直接的组合推导。(提示：考虑一个递增子序列所有可能出现的位置。)

习题6.35　在一个长度为N的随机排列中，找出长度为k的递增子序列数的EGF和一个渐近估计（其中k相对于N是固定的）。

习题6.36　在一个长度为N的随机排列中，找出长度至少为3的递增子序列数的EGF和一个渐近估计。

峰和谷。作为应用堆序树来分解排列的一个例子，我们现在要导出一些结果，以完善上升和游程的统计量。HOT树中的结点有三种类型：树叶（两个子结点均为外部子结点的结点）、单叉结点（一个内部子结点和一个外部子结点）和二叉结点（两个子结点均为内部子结点）。对不同类型结点的研究与对排列中峰和谷的研究有着直接的联系（见习题6.19）。此外，这些统计量还具有独立的意义，因为它们可用于HOT树和BST树的存储需求分析。 331

给定一棵堆序树，其相应的排列可以通过简单地按中缀（左到右）顺序列出结点的标号而得到。在这种对应下，很明显HOT树中的二叉结点对应排列中的一个峰：从左向右扫描，一个二叉结点前接其左子树上一个较小的元素，后接其右子树上另一个较小的元素。这样，对随机排列中峰的分析就简化成了对随机HOT树中二叉结点数的分析。

一个大小为N的随机HOT树是由一个大小为k的左子树和一个大小为$N-k-1$的右子树所构成的，其中，k值等可能地取自0到$N-1$，因此每种取值的概率是$1/N$。这个结果可以直接看出来（一个排列的最小值假定了每一个可能的位置是等可能的）或通过HOT-BST的等价关系看出来。因此，均值可用与第5章中为BST树而研制的相同的方法来计算。

例如，一个HOT树中二叉结点数的均值满足下面的递推关系：

$$V_N = \frac{1}{N}\sum_{0\le k\le N-1}(V_k + V_{N-k-1}) + \frac{N-2}{N} \qquad (N \geq 3)$$

这是因为除了最小元素是排列中的第一个或最后一个元素（该事件的概率是$2/N$）外，二叉结点数是左子树中的二叉结点数与右子树中的二叉结点数之和再加1。因此，从3.3节开始，我们已经在多种场合下见过这种形式的递推关系了。上式乘以z^{N-1}并求和，就导出了下面的微分方程

$$V'(z) = 2\frac{V(z)}{1-z} + \frac{z^2}{(1-z)^2}$$

它的解为

$$V(z) = \frac{1}{3}\frac{z^3}{(1-z)^2} \qquad \text{所以 } V_N = \frac{N-2}{3}$$

因此，一个随机排列中谷的平均数为$(N-2)/3$，其他一些相关量的类似结果也即随之而推出。

定理6.6（排列与树中结点的局部性质）　在一个N元素的随机排列中，谷、峰、双升和双落的平均个数分别是

$$\frac{N-2}{3}、\frac{N-2}{3}、\frac{N+1}{6}和\frac{N+1}{6}$$

332

在一个大小为N的随机HOT树或BST树中，二叉结点、树叶、左分枝结点和右分枝结点的平均个数分别是

$$\frac{N-2}{3}、\frac{N+1}{3}、\frac{N+1}{6}和\frac{N+1}{6}$$

证明　用类似于上面给出的论证方法（或仅用定理5.7）并利用各个量之间的简单关系，上述结果都能直接推导出来。也可参见习题6.16。

例如，排列中的一个降落要么是谷、要么是双落，所以双落数的平均值是

$$\frac{N-1}{2}-\frac{N-2}{3}=\frac{N+1}{6}$$

再例如，我们知道一个随机BST树中树叶的平均个数是$(N+1)/3$（或直接证明，就像前面对HOT树的证明一样），且由上面的论证知二叉结点的平均个数是$(N-2)/3$，因此，在左、右分枝结点等可能的情况下，单叉结点的平均个数是

$$N-\frac{N-2}{3}-\frac{N+1}{3}=\frac{N+1}{3}$$

■

习题6.37　证明：随机排列中谷和峰具有相同的分布。

习题6.38　假设叶子、单叉结点和二叉结点所需的空间分别与c_0、c_1、c_2成正比。证明随机HOT树和随机BST树所需的存储空间是$\sim(c_0+c_1+c_2)N/3$。

习题6.39　在上一题的假设下，证明随机二叉Catalan树所需的存储空间是$\sim(c_0+2c_1+c_2)N/4$。

习题6.40　证明：N个0到1之间的随机实数（均匀且独立生成的）的序列平均有$\sim N/6$个双升和$\sim N/6$个双落。导出一个直接的连续模型来证明这个渐近结果。

习题6.41　对习题6.19进行推广，证明：在HOT树中，右分枝结点和二叉结点的BGF满足

$$K_z(u,z)=1+(1+u)K(u,z)+K^2(u,z)$$

因此

$$K(u,z)=\frac{1-e^{(u-1)z}}{u-e^{(u-1)z}}$$

333 （注意：这个结果提供了关于欧拉数的BGF的另一种推导，因为$A(u,z)=1+uK(u,z)$。）

表6-7总结了前面导出的结果和一些我们将要在下面三节中推导的有关随机排列各种参数的平均值。排列是非常简单的组合对象，我们可以用多种方法导出这些结果，但前面的各种例子已充分表明，使用BGF和CGF的组合证明尤其直接和简单。接下来，我们将利用这些技巧来分析一些在几种基本算法分析中起着重要作用的量。

表6-7　排列性质的分析结果（平均情形）

	指数CGF	平均($[z^N]$)
左向右最小	$\frac{1}{1-z}\ln\frac{1}{1-z}$	H_N
圈	$\frac{1}{1-z}\ln\frac{1}{1-z}$	H_N
单圈	$\frac{z}{1-z}$	1

（续）

	指数CGF	平均([z^N])
圈 = k	$\frac{z^k}{k}\frac{z}{1-z}$	$\frac{1}{k}\ (N \geqslant k)$
圈 ⩽ k	$\frac{1}{1-z}\left(z+\frac{z^2}{2}+\cdots+\frac{z^k}{k}\right)$	$H_k(N \geqslant k)$
游程	$\frac{1}{2(1-z)^2}-\frac{1}{2}$	$\frac{N+1}{2}$
逆序	$\frac{z^2}{2(1-z)^3}$	$\frac{N(N-1)}{4}$
递增子序列	$\frac{1}{1-z}\exp\left(\frac{z}{1-z}\right)$	$\sim\frac{1}{2\sqrt{\pi e}}e^{2\sqrt{N}}/N^{1/4}$
峰、谷	$\frac{z^3}{3(1-z)^2}$	$\frac{N-2}{3}$

6.6 逆序与插入排序

程序6.2是*插入排序*的一个实现，是一个容易分析的简单排序方法。在该方法中，我们把元素“插入”到那些已经有序的元素中一个正确的位置，将所有较大元素都移动一个位置以腾出那个正确位置。图6-6给出了程序6.2对一个排列示例的操作过程，图中第i行阴影部分的元素就是在第i次插入时移动的元素。 334

插入排序的运行时间与$c_1N + c_2B + c_3$成正比，其中c_1、c_2、c_3是与实现有关的适当的常数，B是输入排列的一个函数，是a[j+1]:=a[j]的执行次数。插入每个元素时要移动的元素个数是被插入元素左方大于该元素的元素个数，因此我们可以直接考虑逆序表。图6-6右半部分是排序过程中排列的逆序表。当第i次插入结束后（见第i行），逆序表中前i个元素是零（因为排列中前i个元素已经有序），且逆序表中下一个元素指出了下一次插入时将有多少个元素要移动，因为它指出了第（$i + 1$）个元素左边大于该元素的个数。第i次插入对逆序表的影响只是将其第i项变成零。这意味着当插入排序运行于一个排列时，量B的值等于逆序表中各项数据之和——排列中逆序的总数。

程序6.2　插入排序

```
a[0] :=-infinity;
for i := 2 to N do
  begin
  v := a[i]; j := i-1;
  while a[j] > v  do
    begin a[j+1] := a[j]; j := j-1 end;
  a[j+1] := v;
  end
```

习题6.42　有多少个N元素的排列恰好有1个逆序？2个逆序？3个逆序？

习题6.43　说明如何修改插入排序，使之也能计算与元素的原始顺序相对应的排列的逆序表？

正如前面所提到过的，排列和逆序表之间存在着一对一的对应。在任何逆序表$q_1q_2\cdots q_N$中，每一个q_i必须在0到$i - 1$之间（特别地，q_1总是0）。对每一个q_i，有i个可能的取值，所以有$N!$个不同的逆序表。逆序表在用于分析时更加简单，因为逆序表中的数据是独立的：每一个q_i独 335

立于其他项的取值而取i个不同的值。

9	14	4	1	12	2	10	13	5	6	11	3	8	15	7		0	0	2	3	1	4	2	1	5	5	3	9	6	0	8
9	14	4	1	12	2	10	13	5	6	11	3	8	15	7		0	0	2	3	1	4	2	1	5	5	3	9	6	0	8
4	9	14	1	12	2	10	13	5	6	11	3	8	15	7		0	0	0	3	1	4	2	1	5	5	3	9	6	0	8
1	4	9	14	12	2	10	13	5	6	11	3	8	15	7		0	0	0	0	1	4	2	1	6	6	3	9	6	0	8
1	4	9	12	14	2	10	13	5	6	11	3	8	15	7		0	0	0	0	0	4	2	1	5	5	3	9	6	0	8
1	2	4	9	12	14	10	13	5	6	11	3	8	15	7		0	0	0	0	0	0	2	1	5	5	3	9	6	0	8
1	2	4	9	10	12	14	13	5	6	11	3	8	15	7		0	0	0	0	0	0	0	1	5	5	3	9	6	0	8
1	2	4	9	10	12	13	14	5	6	11	3	8	15	7		0	0	0	0	0	0	0	0	5	5	3	9	6	0	8
1	2	4	5	9	10	12	13	14	6	11	3	8	15	7		0	0	0	0	0	0	0	0	0	5	3	9	6	0	8
1	2	4	5	6	9	10	12	13	14	11	3	8	15	7		0	0	0	0	0	0	0	0	0	0	3	9	6	0	8
1	2	4	5	6	9	10	11	12	13	14	3	8	15	7		0	0	0	0	0	0	0	0	0	0	0	9	6	0	8
1	2	3	4	5	6	9	10	11	12	13	14	8	15	7		0	0	0	0	0	0	0	0	0	0	0	0	6	0	8
1	2	3	4	5	6	8	9	10	11	12	13	14	15	7		0	0	0	0	0	0	0	0	0	0	0	0	0	0	8
1	2	3	4	5	6	8	9	10	11	12	13	14	15	7		0	0	0	0	0	0	0	0	0	0	0	0	0	0	8
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图6-6 插入排序和逆序表

定理6.7（*逆序分布*） *具有k个逆序且大小为N的排列个数是*

$$[u^k]\prod_{1\le j\le N}\frac{1-u^j}{1-u}=[u^k](1+u)(1+u+u^2)\cdots(1+u+\cdots+u^{N-1})$$

一个N元素的随机排列平均有$N(N-1)/4$个逆序，其标准差为$N(N-1)(2N+5)/72$。

证明 我们给出利用PGF的推导方法；一个组合推导将遵循（几乎总是遵循）相同的方式。

在一个随机排列的逆序表中，第i项数据可以独立于其他数据项以概率$1/i$取0到$i-1$之间的每一个值。因此，涉及第N个元素的逆序个数的概率生成函数是$(1+u+u^2+\cdots+u^{N-1})/N$，这个结果与第$N$个元素前面的元素的排列方式没有关系。正如我们在第3章中所讨论过的那样，独立随机变量的和的PGF是各个随机变量PGF的乘积，所以在一个N元素的随机排列中，逆序总数的生成函数满足

336

$$b_N(u)=\frac{1+u+u^2+\cdots+u^{N-1}}{N}b_{N-1}(u)$$

也就是说，逆序数是j从1到N，具有独立的均匀分布的，具有OGF$(1+u+u^2+\cdots+u^{j-1})/j$的随机变量之和。定理中所述的计数GF等于$N!$乘以$b_N(u)$。平均值是单个平均值$(j-1)/2$的和，方差是单个方差$(j^2-1)/12$的和。 ■

图6-7给出了$[u^k]b_N(u)$的完整分布。曲线关于$N(N-1)/4$对称，且随着N的增长，它们向着中心收缩（不过是缓慢的）。这种曲线可以描绘成独立随机变量之和的分布。尽管这些分布是不同的，但根据概率论中的古典中心极限定理，可以证明它们的极限分布是正态的（如，见David和Barton[21]），这在算法分析中是非常典型的，但这里我们不做详细分析。

推论 *对一个有N条记录（记录中的关键字不同且顺序是随机的）的文件排序，插入排序平均执行$\sim N^2/4$次关键字比较，平均移动$\sim N^2/4$次记录。*

用CGF求解，使用累积生成函数求逆序的平均数也具有启发意义，因为它可以用作更为复杂问题的模型。考虑CGF

$$B(z)=\sum_{p\in\mathcal{P}}\sigma(p)\frac{z^{|p|}}{|p|!}$$

正如前面所提到过的，$B(z)$中$z^N/N!$的系数是所有长度为N的排列的逆序总数，因而$B(z)$是一个排列中逆序平均数的OGF。上面所给出的论证等价于从关于逆序的"水平"生成函数中计算累积开销；我们现在考虑另外一种直接使用CGF的推导方法。

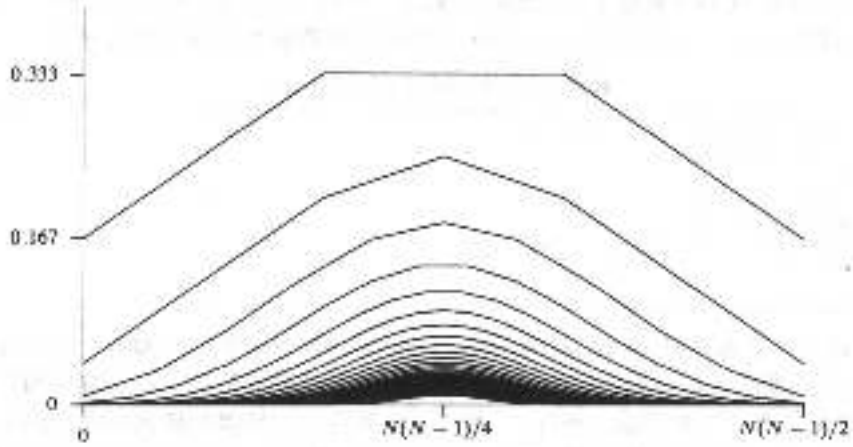


图6-7　逆序的分布，$3 \le N \le 60$（k轴按$\binom{N}{2}$的比例标度）

利用"最大"对应关系，所有长度为$|p|$的排列都对应于长度为$|p|+1$的通过在第k和第$k+1$个元素之间放入元素$|p|+1$而形成的$|p|+1$个排列（k在0到$|p|$之间）。这样的一个排列的逆序数比p多$|p|-k$个，这就导出了下面的表达式

$$B(z) = \sum_{p \in P} \sum_{0 \le k \le |p|} (\sigma(p) + |p| - k) \frac{z^{|p|+1}}{(|p|+1)!}$$

对k求和，得

$$B(z) = \sum_{p \in P} \sigma(p) \frac{z^{|p|+1}}{|p|!} + \sum_{p \in P} \binom{|p|+1}{2} \frac{z^{|p|+1}}{(|p|+1)!}$$

第一个和是$zB(z)$，第二个和很容易计算，因为它只依赖于排列的长度，所以长度为k的$k!$个排列可关于每个k累加起来，从而得到

$$B(z) = zB(z) + \frac{z}{2} \sum_{k \ge 0} kz^k = zB(z) + \frac{1}{2} \frac{z^2}{(1-z)^2}$$

所以，正如我们所期望的那样，关于$N(N-1)/4$的生成函数为$B(z) = z^2/(2(1-z)^2)$。

337
2
338

在本章的后面我们还将学习逆序表的其他性质，因为它们能够描述在某些其他算法分析中所产生的排列的其他性质。特别地，我们将考虑逆序表中零的个数，以及最大元素的值。

习题6.44　导出p_{Nk}所满足的一个递推关系，其中p_{Nk}是一个N元素随机排列恰有k个逆序的概率。

习题6.45　求关于所有长度为N的对合中逆序总数的CGF。

习题6.46　证明：当N充分大时，对任意固定的k，$N!p_{Nk}$是关于N的一个固定的多项式。

Shellsort。程序6.3给出了对插入排序一个实用的改进，叫做Shellsort（Shell排序）。它通过对文件进行多趟扫描，其中每一趟对间隔为h的元素所构成的h个独立的子文件（每一个大小约为N/h）进行排序，从而使运行时间减少到大大低于N^2。控制排序的"增量"h[t]，

h[t-1]，…，h[1]必须形成一个结束于1的递减序列。人们已做了很大努力去找一个增量的最佳序列，但几乎还没有得到什么分析结果：尽管Shellsort只是插入排序的一个简单的扩展，但这个排序已被证实是最难分析的。对Shellsort平均情况性能的完整描述还是一个没有解决的问题。Yao[19]曾利用类似于前面的技巧做过一个$(h, k, 1)$ Shellsort的分析，但结果和方法都变得复杂得多。对一般的Shellsort，即使函数形式也明显地依赖于增量序列。

程序6.3 Shellsort（Shell排序）

```
for h := h[t], h[t-1], ... , h[1] do
  for i := h-1 to N do
    begin
    v := a[i]; j := i-1;
    while a[j] > v  do
      begin a[j+1] := a[j]; j := j-1 end;
    a[j+1] := v;
    end
```

339

2-有序排列。考虑对h只取值2和1的情况时Shellsort的分析是很有趣的，因为这种分析与第5章树中路径长度的分析有着密切的联系。它与归并算法是等价的：对具有奇编号和偶编号位置上的文件独立排序（利用插入排序），然后用插入排序对所得的排列再进行排序。这样由两个相互间隔的已排序的排列组成的排列，叫做2-有序排列。2-有序排列的性质在其他归并算法的研究中也很重要。因为Shellsort的最后一趟（h = 1时）就是插入排序，其平均运行时间取决于2-有序排列中的平均逆序数。表6-8给出了3个2-有序排列的实例以及它们的逆序表。

表6-8 3个2-有序排列及其逆序表

4	1	5	2	6	3	9	7	10	8	13	11	15	12	16	14	19	17	20	18
0	1	0	2	0	3	0	1	0	2	0	1	0	2	0	2	0	1	0	2
1	4	2	5	3	6	8	7	9	12	10	13	11	14	17	15	18	16	19	20
0	0	1	0	2	0	0	1	0	0	1	0	2	0	0	1	0	2	0	0
4	1	5	2	6	3	7	8	12	9	13	10	14	11	15	17	16	18	20	19
0	1	0	2	0	3	0	0	0	1	0	2	0	3	0	0	1	0	0	1

设$S(z)$是对2-有序排列计数的OGF。显然，

$$S(z) = \sum_{N\geq 0}\binom{2N}{N}z^N = \frac{1}{\sqrt{1-4z}}$$

但我们将考虑另一个揭示结构的计数方法。图6-8说明了这样的事实：2-有序排列对应于一个$N \times N$网格中的路径，这些路径类似于第5章所描述的关于树的"赌徒破产"表示法中的那些路径。从左上角开始，如果i在一个奇编号位置，则向右移动；如果i在一个偶编号位置，则向下移动。由于向右有N步、向下有N步，因此移动将结束于右下角。

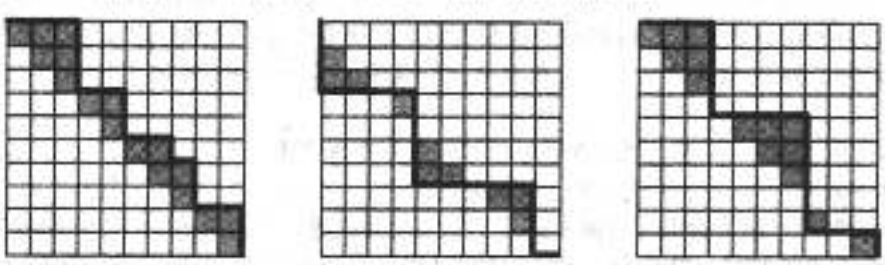


图6-8 关于表6-8中2-有序排列的格路径

这里，与第5章中所讨论的一样，没有碰到对角线的路径对应于树，并由如下生成函数来计数:

$$zT(z)=G(z)=(1-\sqrt{1-4z})/2$$

对2-有序排列而言，触及对角线的限制被去掉了。但是，任何穿越网格的路径必须在开始时触及对角线，这就导致对2-有序排列计数的OGF的符号方程:

$$S(z)=2G(z)S(z)+1$$

就是说，任何穿越网格的路径可以从一个除端点之外不触及对角线的初始部分后接一条一般的路径而唯一地构造出来。其中因子2说明初始部分既可能在对角线的上方，也可能在对角线的下方。正如我们所期望的那样，公式可化简为

$$S(z)=\frac{1}{1-2G(z)}=\frac{1}{1-(1-\sqrt{1-4z})}=\frac{1}{\sqrt{1-4z}}$$

Knuth[10]（再参看Vitter和Flajolet[16]）证明了这一相同的一般结构可用于书写关于逆序的BGF的显式表达式，其最终结果为：累积开销（长度为$2N$的所有2-有序排列中的逆序总数）就是$N4^{N-1}$。该论证基于如下观察：2-有序排列中的逆序数等于相应的格路径与“下-右-下-右……”对角线之间的方格数。

定理6.8 (2-有序排列中的逆序数)　*长度为$2N$的随机2-有序排列中的平均逆序数为*

$$N4^{N-1}\Big/\binom{2N}{N}\sim\sqrt{\pi/128}\,(2N)^{3/2}$$

证明　导出这一简单结果的计算是直接的但也是很复杂的，我们把它留作习题。在7.5节中，我们还将在更一般的场合下提及这个问题。■

340～341

习题6.47　证明：2-有序排列中的逆序数等于相应的格路径与“下-右-下-右……”对角线之间的方格数。

习题6.48　设$\mathcal{T}$是所有2-有序排列的集合，定义BGF为

$$P(u,z)=\sum_{p\in\mathcal{T}}u^{\{p\text{中的逆序数}\}}\frac{z^{|p|}}{|p|!}$$

用同样的方法定义$Q(u,z)$，但限制集合是由这样的2-有序排列所构成的：其对应的格路径除端点外不触及对角线。此外，类似地定义$S(u,z)$和$T(u,z)$，但限制2-有序排列对应的格路径除端点外整个位于对角线的上方。证明：$P(u,z)=1/(1-Q(u,z))$及$S(u,z)=1/(1-T(u,z))$。

习题6.49　证明$T(u,z)=uzS(u,uz)$和$Q(u,uz)=T(u,uz)+T(u,z)$。

习题6.50　利用前两个习题的结果证明

$$S(u,z)=uzS(u,z)S(u,uz)+1$$

和

$$P(u,z)=(uzS(u,uz)+zS(u,z))P(u,z)+1$$

习题6.51　利用前面习题的结果证明

$$P_u(1,z)=\frac{z}{(1-4z)^2}$$

推论　*对一个N元素的文件进行（2, 1）Shell sort所用比较的平均次数是* $N^2/8+\sqrt{\pi/128}\,N^{3/2}+O(N)$。

证明　假设N是偶数。第一趟是由两个含有$N/2$个元素的独立的排序所构成的，因而涉及$2(N/2)(N/2-1)/4)=N^2/8+O(N)$次比较，并留下一个随机2-有序文件。于是在第二趟中又用$\int\sqrt{\pi/128}N^{3/2}$次比较。

当N是奇数时有相同的渐近结果。因此，尽管（2,1）Shellsort需要对文件进行两趟排序，但它的比较次数是插入排序的一半。 ■

习题6.52　给出一个3-有序排列中平均逆序数的一个渐近公式，并分析当增量为3和1时Shellsort的情况。推广这一结果以估计（h, 1）Shellsort开销的主项以及当使用h的最佳值时的渐近开销（作为N的一个函数）。

342 **习题6.53**　分析如下排序算法：给定一个要排序的数组，对奇数位置和偶数位置上的元素进行逐步排序，然后用插入排序对所得的2-有序排列进行排序。当N为何值时，该算法比第1章中的简单的Quicksort所用的平均比较次数要少？

6.7 左向右最小值与选择排序

求一个数组中最小元素的平凡算法是从左向右扫描整个数组，跟踪当前所找到的最小值。通过连续地求最小值，导致另外一个叫做选择排序（selection sort）的简单排序方法，见程序6.4。图6-9给出了选择排序对我们的样本文件操作的图示。此外，左边的图表示排列，右边的图表示相应的逆序表。

程序6.4　选择排序

```
for j := 1 to N do
  begin
  v := infinity; k := j;
  for i := j to N do
    if a[i] < v then
      begin v := a[i]; k := i end;
  t := a[j]; a[j] := a[k]; a[k] := t;
  end;
```

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

0	0	2	3	1	4	2	1	5	6	8	0	6	0	8
0	0	2	3	1	4	2	1	5	5	3	0	6	7	0
0	1	2	3	0	4	1	0	5	5	2	0	5	0	0
0	1	2	3	0	4	1	3	5	5	1	0	0	0	0
0	1	2	3	3	4	0	2	4	4	0	0	0	0	0
0	1	2	3	3	4	0	2	4	4	0	0	0	0	0
0	1	2	3	3	4	2	1	4	0	0	0	0	0	0
0	0	2	3	3	4	1	0	0	0	0	0	0	0	0
0	0	2	3	3	4	1	0	0	0	0	0	0	0	0
0	0	2	3	3	4	0	0	0	0	0	0	0	0	0
0	1	1	3	2	0	0	0	0	0	0	0	0	0	0
0	1	0	3	0	0	0	0	0	0	0	0	0	0	0
0	1	2	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

图6-9　选择排序与左向右最小值

为了分析选择排序，我们首先需要分析在一个随机排列中“求最小值”的算法：程序6.4中外层循环的第一次迭代（$j=1$）。至于插入排序，此算法的运行时间可以用N和一个其值取决于特定排列的量来表示，在此程序中是“当前最小值”被修改的次数（变量v的值在程序6.4中被改变的次数），这恰好是排列中左向右最小值数。

这个量借助于逆序表很容易分析：正如上面所观察到的，在逆序表$q_1q_2\cdots q_N$中，每一个$q_i = i - 1$的项都对应一个左向右最小值，这是因为它左边的所有元素都比它大。因此，逆序表中每一项都是一个概率为$1/i$的左向右最小值，其取值独立于其他项，所以平均值为$\sum_{1<i<N} 1/i = H_N$。对此论证稍加推广，即可得到PGF。

定理6.9（左向右最小值的分布）　*具有k个左向右最小值的N个元素的排列是用第一类Stirling数来计数的：*

$$\begin{bmatrix} N \\ k \end{bmatrix} = [u^k]u(u+1)\cdots(u+N-1)$$

一个N元素的随机排列平均有H_N个左向右最小值，其方差是$H_N - H_N^{(2)}$。

证明　考虑一个N元素的随机排列中关于左向右最小值数的概率生成函数$P_N(u)$。与前面一样，我们可以将其分解成两个独立的随机变量：一个作为$N - 1$个元素的随机排列（其PGF为$P_{N-1}(u)$），另一个作为最后那个元素的贡献（其PGF为$(N - 1 + u)/N$，这是因为最后那个元素以概率$1/N$使左向右最小值的个数增1，否则增0）。因此我们有：

$$P_N(u) = \frac{N-1+u}{N} P_{N-1}(u)$$

并且，和前面一样，我们通过从简单概率生成函数$(z + k - 1)/k$中累加均值和方差来求得左向右最小值的均值和方差。计数GF等于$N!P_N(u)$。 ■

343 ~ 344

用CGF求解。和以往一样，我们引入指数CGF

$$B(z) = \sum_{p\in\mathcal{P}} \lambda(p)\frac{z^{|p|}}{|p|!}$$

因而$[z^N]B(z)$是一个N元素随机排列中左向右最小值的平均数。

像以前一样，我们可以根据组合定义直接导出一个函数方程，在这个实例中是从上方使用“最后”对应。在大小为$|p| + 1$、对应于一个给定排列p的$|p| + 1$个排列中，其中的一个以1结束（因此左向右最小值比p多一个），而其他$|p|$个不以1结束（因此左向右最小值数与p相同）。这就导致公式

$$\begin{aligned}
B(z) &= \sum_{p\in\mathcal{P}} (\lambda(p)+1)\frac{z^{|p|+1}}{(|p|+1)!} + \sum_{p\in\mathcal{P}} |p|\lambda(p)\frac{z^{|p|+1}}{(|p|+1)!} \\
&= \sum_{p\in\mathcal{P}} \lambda(p)\frac{z^{|p|+1}}{|p|!} + \sum_{p\in\mathcal{P}} \frac{z^{|p|+1}}{(|p|+1)!} \\
&= zB(z) + \sum_{k\geq 0} \frac{z^{k+1}}{(k+1)} \\
&= zB(z) + \ln\frac{1}{1-z}
\end{aligned}$$

因此解为

$$B(z) = \frac{1}{1-z}\ln\frac{1}{1-z}$$

这正是我们所期望的关于调和数的生成函数。

下面，我们将考虑如何扩展这一推论，这只需稍微再做一点工作，以给出一个描述整个

分布的指数BGF的显式表达式。

习题6.54 设P_{Nk}是一个具有k个左向右最小值的N元素随机排列的概率。给出P_{Nk}所满足的递推关系。

345

第一类Stirling数。继续上面的讨论，我们从

$$B(u,z)=\sum_{p\in\mathcal{P}}\frac{u^{\lambda(p)}z^{|p|}}{|p|!}=\sum_{N\geq 0}\sum_{k\geq 0}p_{Nk}u^k z^N$$

开始，其中p_{Nk}是一个具有k个左向右最小值的N元素随机排列的概率。用与上面相同的组合结构导出如下式子

$$B(u,z)=\sum_{p\in\mathcal{P}}\frac{u^{\lambda(p)+1}z^{|p|+1}}{(|p|+1)!}+\sum_{p\in\mathcal{P}}\frac{|p|u^{\lambda(p)}z^{|p|+1}}{(|p|+1)!}$$

关于z微分，我们有

$$\begin{aligned}B_z(u,z)&=\sum_{p\in\mathcal{P}}\frac{u^{\lambda(p)+1}z^{|p|}}{|p|!}+\sum_{p\in\mathcal{P}}\frac{u^{\lambda(p)}z^{|p|}}{(|p|-1)!}\\&=uB(u,z)+zB_z(u,z)\end{aligned}$$

解出$B_z(u,z)$，我们将得到一个简单的一阶微分方程

$$B_z(u,z)=\frac{u}{1-z}B(u,z)$$

其解为

$$B(u,z)=\frac{1}{(1-z)^u}$$

(因为$B(0,0)=1$)。对u求微分，然后在$u=1$处求解，即得到我们所期望的关于调和数的OGF。展开

$$B(u,z)=1+\frac{u}{1!}z+\frac{u(u+1)}{2!}z^2+\frac{u(u+1)(u+2)}{3!}z^3+\cdots$$

又得到定理6.9中所述的第一类Stirling数的表达式。

BGF $B(u,z)=(1-z)^{-u}$是我们在第3章中所见过的一种古典型。正如我们将要在下面所看到的，具有恰好k个左向右最小值的N元素的排列数等于恰好有k个圈的N元素的排列数。二者都由第一类Stirling数来计数，因此有时也把它们叫做Stirling“圈”数。其分布由图6-10和表6-9所示。

习题6.55 直接证明 $\sum_k k\begin{bmatrix}N\\k\end{bmatrix}=N!H_N$ 。

表6-9 左向右最小值及圈的分布（第一类Stirling数）

$N\downarrow$ $k\rightarrow$	1	2	3	4	5	6	7	8	9	10
1	1									
2	1	1								
3	2	3	1							
4	6	11	6	1						
5	24	50	35	10	1					
6	120	274	225	85	15	1				

（续）

N↓ k→	1	2	3	4	5	6	7	8	9	10
7	720	1764	1624	735	175	21	1			
8	5040	13068	13132	6769	1960	322	28	1		
9	40320	109584	118124	67284	22449	4536	546	36	1	
10	362880	1026576	1172700	723680	269325	63273	9450	870	45	1

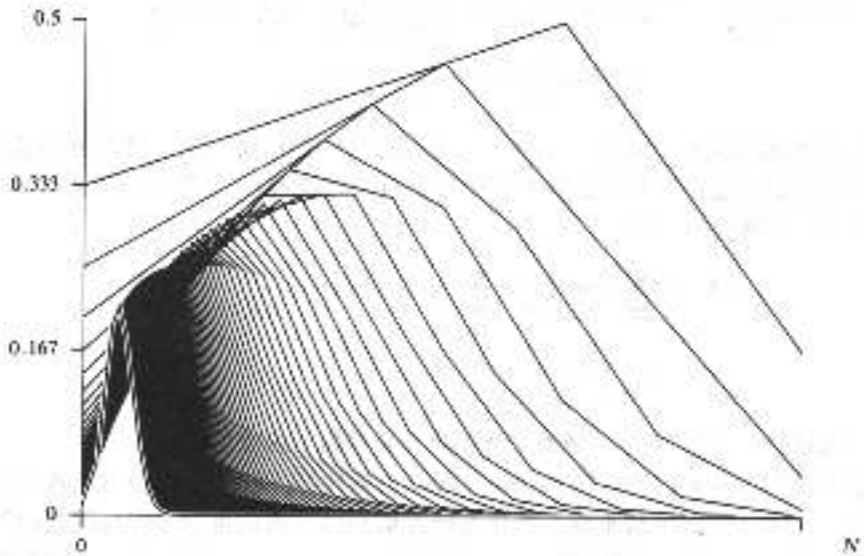


图6-10　左向右最小值及圈的分布（第一类Stirling数）

346
2
347

选择排序，在图6-9中，很明显第i步之后，逆序表中最右边的i个元素均为0，但对逆序表中其他元素的影响就比较难解释了。其原因是选择排序中的各趟是相互独立的：例如，如果在一次迭代中左向右最小值是个较小的数的话，那么我们可期望下一次迭代中左向右最小值也是个较小的数。运行时间中的"主"项是if语句的执行次数，很容易证明对每个输入的排列，该执行次数是$\binom{N+1}{2}$；或者说如果记录很大时，执行次数是两个记录交换的次数，很明显对每个输入的排列，该次数是$N-1$。

在记录非常大的应用中，交换记录的开销在运行时间上起支配地位。因为选择排序执行交换的可能次数最小，所以它是这类应用中的一个可选方法。

定理6.10（选择排序）　对随机顺序的互异关键字的N个记录的文件排序，选择排序平均执行$\sim N^2/2$次关键字比较，平均保存关键字值$\sim N\ln N$次，平均移动记录$\sim N$次。

证明　见上面的讨论。在程序6.4的运行时间中，唯一的变量（独立于排列的量）是整个排序过程中所碰到的右向左最大值的总个数：即if语句为真的次数。对于大记录，这段代码会被修改成保存一个关键字值和一个记录的下标，所以开销既不被确切地解释成关键字的比较，也不解释成一个记录的移动。

我们可以通过定义开销与排列之间的下述对应关系来导出这个总个数的平均值：给定$N-1$个元素的一个排列，通过对该排列的头部插入1、再将其他每个元素增1、最后将其他每个元

素与1交换位置而构造出N个N元素的排列。例如，

1 5 2 3 4　5 1 2 3 4　2 5 1 3 4　3 5 2 1 4　4 5 2 3 1

都与4 1 2 3对应。也就是说，如果上述排列中的任何一个是选择排序算法中的初始输入，那么当一次迭代结束后，其结果都将是1 5 2 3 4，对于随后的迭代，它与4 1 2 3是相同的。这种对应关系立即隐含了

348

$$B_N = B_{N-1} + H_N = (N+1)H_N - N$$

更具体地说，设cost(p)表示一个给定排列p的这种总个数的开销，并考虑CGF

$$B(z) = \sum_{N \geq 0} B_N z^N = \sum_{p \in \mathcal{P}} \mathrm{cost}(p) \frac{z^{|p|}}{|p|!}$$

上述对应关系表明该算法在下述意义下具有不变的行为：每个排列第一趟的开销为$\lambda(p)$，然后，如果我们考虑第一趟的结果（适用于所有$|p|!$种可能的输入），每个大小为$|p| - 1$的排列都出现相同的次数。这就导出了解

$$\begin{aligned} B(z) &= \sum_{p \in \mathcal{P}} \lambda(p) \frac{z^{|p|}}{|p|!} + \sum_{p \in \mathcal{P}} (|p|+1) \mathrm{cost}(p) \frac{z^{|p|+1}}{(|p|+1)!} \\ &= \frac{1}{1-z} \ln \frac{1}{1-z} + zB(z) \end{aligned}$$

因此，$B(z) = \ln(1/(1-z))/(1-z)^2$，是关于调和数部分和的生成函数。 ■

注意，这里我们不具备趟次间的独立性，所以上述技巧不能用来获得方差或这种分布的其他性质。这一点既微妙但又很重要。关于右向左最小值，我们可以很容易地把CGF的推导转换成关于BGF的推导（能够得到方差），但上述论证并没有这样扩展。缺少独立性似乎使这个问题几乎难以应付：直到1988年，当Yao[18]的一个巧妙分析证明了方差为$O(N^{3/2})$之后，这个问题的空白才被填上。

习题6.56 指定并分析这样一个算法，它从左向右扫描，并确定一个数组中的两个最小元素。

习题6.57 若存取记录的开销是存取关键字开销的100倍，且二者相对其他开销都很大的情况下，当N为何值时，选择排序比插入排序要好？

习题6.58 假定“交换”的开销是“存取记录”开销的两倍时，关于Quicksort和选择排序回答前面的问题。

习题6.59 考虑选择排序对链表的一个实现，每次迭代时，剩余的最小元素是通过扫描“输入”表而找到的，然后将其从该表中删掉并追加到一个“输出”表中。分析此算法。

349

习题6.60 假设要排序的N个项实际上由N个字的一些数组组成，其中每个数组的第一个字是排序的关键字。就我们目前所见到的4种基于比较的方法（Quicksort、Mergesort、插入排序和选择排序）中，哪一个最适合这种情况？就输入数据的量而言，该问题的复杂性是多少？

6.8 圈与原位排列

在某些情形下，可能需要把一个数组交换“到位”。正如6.2节中所描述的，可以设计一个排序算法间接地引用记录，求出一个能规定如何进行排列的排列，而不是真的重新排列这些记录。这里，我们从另一角度来考虑重新排列是如何被排列到位的。请看下面的一个例子：

下标	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
输入关键字	29	41	77	26	58	59	97	82	12	44	63	31	53	23	93
排列	9	14	4	1	12	2	10	13	5	6	11	3	8	15	7

这就是说，要将数组排成有序的，a[9]必须移到位置1、a[14]移到位置2、a[4]移到位置3，依此类推。完成这项任务的一种方法是：开始时先把a[1]保存到一个寄存器中，然后令a[1]等于a[p[1]]，置p[1]为j，依此类推，直到p[j]变成1为止，此时把保存的值放入a[j]。然后对每个还没有移动过的元素重复上述过程，但如果我们用同样的方法排列数组p，那么我们会很容易地识别那些需要移动的元素，如程序6.5所示。

程序6.5　原位排列

```
for i := 1 to N do
  if p[i] <> i then
    begin
    t := a[i]; k := i;
    repeat
      j := k; a[j] := a[p[j]];
      k := p[j]; p[j] := j;
    until k = i;
    a[j] := t
    end;
  end;
```

在上面的例子中，首先a[9]=12被移到位置1，然后a[5]=58被移到位置9，然后a[12]=31被移到位置5，然后a[3]=77被移到位置12，然后a[4]=26被移到位置3，然后原来在位置1上的29被放入位置4。与此同时，为反映排列中的这些移动，将p[1]置为1，p[9]置为9，依此类推。注意，当p[i]=i时，程序中的循环将什么也不做，所以该程序对任何元素的移动都不会超过一次。 350

该程序的内部循环迭代了多少次？显然，这个量依赖于排列的圈结构——程序本身实际上以一种相当简明方式展示了圈的概念。程序6.5的运行时间取决于圈的个数，因为一定的额外簿记工作量与每个圈都有联系。因此，这个量可以在1（例如，在排列2 3 4 …N1中）到N（例如在排列1 2 3 …N中）内变化，但结果证明这个量的平均值并不是很大。

定理6.11（圈分布）　*在一个随机排列中，圈数的分布与左向右最小值数的分布是相同的，并且可由第一类Stirling数给出。一个随机排列中的平均圈数是H_N，其标准差为$H_N - H_N^{(2)}$。*

证明　在6.1节中我们已经讨论过这样的事实：根据基本对应关系这两个分布是相同的。为了对此给出一个解析性的证实，我们给出一个直接基于二元生成函数

$$B(w,z)=\sum_{p\in P} w^{|p|} z^{\lambda(p)}$$

的分析，其中$\lambda(p)$是p中圈的个数。

和以前一样，可以根据组合定义直接导出一个函数方程。给定一个排列p，我们可以通过在每个圈（包括“空”圈）中的每个位置加入元素$|p|+1$来建立$|p|+1$个大小为$|p|+1$的排列。当然，在这些排列中，有一个排列比p多一个圈，有$|p|$个排列与p有相同个数的圈。这个对应关系从结构上来讲与我们为左向右最小值建立的对应关系是一样的。利用和前面所给的完全一样的论证方法，我们得到的生成函数当然是和关于左向右最小值个数完全一样的生成函数：

$$B(u,z)=\frac{1}{(1-z)^u}$$

351

因此，它们的分布是相同：平均值如前所述；恰好有k个圈的N元素的排列数由第一类Stirling数给出，等等。 ■

推论 *平均移动~$N+\ln N$次数据可以把一个文件排列到位。*

证明 见上面的讨论。 ■

圈长。了解一个排列中的圈长有助于我们更详细地分析程序6.5。例如，我们可能想知道是否值得修改此程序以使其在`p[i]=i`时不进行任何操作。从6.4节中关于带有圈长限制的排列的计数结果知道，对一个随机排列而言，这种情况至少发生一次的概率是$1-1/e$，但是我们期望它会发生多少次呢？最终结果表明，平均而言，一个随机排列中单元素圈的个数是1，因而我们可以分析有关圈长分布的其他事实。

单元素圈。为了和前面的其他推导相比较，我们利用前面的组合构造和CGF

$$B(z)=\sum_{p\in\mathcal{P}}\tau(p)\frac{z^{|p|}}{|p|!}$$

来求出单元素圈的平均个数，其中$\tau(p)$是一个排列p中单元素圈的个数，因而我们所要的答案就是$[z^N]B(z)$。根据前面的构造，长度为$|p|$的排列可以被分成大小为$|p|$的组，组中每一个排列对应一个长度为$|p|-1$的排列q。在对应于q的组中，其中的一个有$\tau(q)+1$个单元素圈，其中的$\tau(p)$个有$\tau(p)-1$个单元素圈，其余的有$\tau(p)$个单元素圈，因而对应于q的所有排列的单元素圈的总数为

$$\tau(q)+1+\tau(q)(\tau(q)-1)+(|p|-1-\tau(q))\tau(q)=1+|q|\tau(q)$$

所以

$$B(z)=\sum_{q\in\mathcal{P}}(1+|q|\tau(q))\frac{z^{|q|+1}}{(|q|+1)!}$$

352

对此公式微分，将得到如下简单形式

$$\begin{aligned}B'(z)&=\sum_{q\in\mathcal{P}}\frac{z^{|q|}}{|q|!}+\sum_{q\in\mathcal{P}}|q|\tau(q)\frac{z^{|q|}}{|q|!}\\&=\frac{1}{1-z}+zB'(z)\end{aligned}$$

所以正如我们所期望的那样，$B'(z)=1/(1-z)^2$及$B(z)=1/(1-z)$。

长度为k的圈。利用BGF，我们可以构造出一个更一般的论证，用来求一个随机排列中长度为k的圈的平均个数。通过改写6.4节中的论证，我们可以写出关于长度为k的圈数的（指数）BGF:

$$\exp\left(z+\frac{z^2}{2}+\frac{z^3}{3}+\cdots+\frac{z^{k-1}}{k-1}+\frac{z^k}{k}u+\frac{z^{k+1}}{k+1}+\cdots\right)$$

如果将其展开，那么每一项代表一个排列，其中u的指数记录了项z^k/k被使用的次数，或对应的排列中长度为k的圈的个数。现在，BGF可以改写为如下形式

$$\exp\left(z+\frac{z^2}{2}+\cdots+\frac{z^k}{k}+\cdots\right)\exp\left((u-1)\frac{z^k}{k}\right)=\frac{1}{1-z}\exp((u-1)z^k/k)$$

这种形式可以使我们计算我们所关注的量。

定理6.12（单元素圈的分布） *一个N元素的排列具有j个单元素圈的概率渐近于$e^{-1}/j!$。一个大小为$N\geq k$的随机排列中长度为k的圈的平均个数是$1/k$，其方差是$1/k$。*

证明　由定理6.3，所求概率由上面导出的BGF的系数所给定：

$$[u^j z^N]\frac{e^{(u-1)z}}{1-z}=\frac{1}{j!}[z^{N-j}]\frac{e^{-z}}{1-z}\sim\frac{e^{-1}}{j!}$$

平均值的计算是定理3.6的一个简单应用：关于u微分并在1处求解，得

$$[z^N]\frac{z^k}{k}\frac{1}{1-z}=\frac{1}{k}\qquad(N\geqslant k)$$

而方差可以用类似的计算来导出。 ■ 353

我们还可以归纳推广定理6.11并导出下面的结果：在一个N元素的随机排列中，长度$\leqslant k$的圈的平均个数是H_k。当然，此结论只在k不大于N时成立。

习题6.61　利用生成函数的渐近性（见4.10节）或用直接论证的方法证明：一个随机排列中具有j个长度为k的圈的概率渐近于泊松分布$e^{-\lambda}\lambda^j/j!$，其中$\lambda=1/k$。

习题6.62　对于长度为100的一个排列，在程序6.5的循环中，迭代次数不超过50的概率是多少？

习题6.63　[Knuth]考虑一种在排列数组不能被修改且没有额外内存可用的情形。一个要进行原位排列的算法可以按如下方法来设计：当遇到圈中最小的下标时，每个圈中的元素都要排列。对从`1`到`N`的`j`，通过从`k=j`开始、且当`k>j`时令`k=p[k]`的方法来检查每个下标，看它是不是圈中的最小下标。如果是，则像程序6.5那样对这个圈进行排列。证明：指令`k=p[k]`执行次数的BGF满足函数方程

$$B_u(u,z)=B(u,z)B(uz,z)$$

根据这个方程求随机排列的这个参数的均值和方差。（见[11]。）

6.9 极值参数

在第5章中我们已发现，树高要比路径长度难分析得多，这是因为计算树高要涉及取最大子树的值，而路径长度只涉及计数和加法，且后者的操作更自然地对应于生成函数的操作。本节，我们将考虑关于排列的类似的参数。一个排列中最长或最短的圈的平均长度是多少？最长游程的平均长度是多少？最长递增子序列又怎样？一个随机排列的逆序表中最大元素的平均值是多少？最后这个问题还引出了排序的另外一个基本算法，我们现在就开始讨论这个算法。

冒泡排序。这个方法很容易解释：要对一个数组排序，可重复扫描这个数组，如有必要，可把一个元素与其下一个元素交换以使它们有序。如果完成数组的某趟扫描时没有进行过任何交换（每个元素都不比它后面的元素大），排序就完成了。程序6.6中给出了一个这样的实现。为分析此算法，我们需要记录交换的次数和扫描的趟数。

程序6.6　冒泡排序

```
i := N+1;
repeat
  t := a[1]; i := i-1;
  for j := 2 to i do
    if a[j-1] > a[j] then
      begin t := a[j-1]; a[j-1] := a[j]; a[j] := t end
until (t = a[1]);
```

交换是很直接的：每个交换都是两个相邻元素的交换（和插入排序一样），所以总的交换次数恰好是一个排列中的逆序数。所用的趟数也和逆序表有直接的联系，如图6-11所示：每一趟结束后逆序表中的每个非零项的值实际上都减1，且当表中没有非零项时，程序结束。这就隐含了对一个排列的冒泡排序所需要的趟数恰好等于逆序表中的最大元素。这个量的分布由表6-10和图6-12所示。

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

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

图6-11 冒泡排序（排列及其相应的逆序表）

表6-10 逆序表中最大值的分布

$N\downarrow$ $k\rightarrow$	0	1	2	3	4	5	6	7	8	9	
1	1										
2	1	1									
3	1	3	2								
4	1	7	10	6							
5	1	15	38	42	24						
6	1	31	130	222	216	120					
7	1	63	422	1050	1464	1320	720				
8	1	127	1330	4686	8856	10920	9360	5040			
9	1	255	4118	20202	50424	80520	91440	75600	40320		
10	1	511	12610	85182	276696	558120	795600	851760	685440	362880	

定理6.13（逆序表中的最大值） 一个随机排列的逆序表中最大元素的均值为 $\sim N-\sqrt{\pi N/2}$ 。

证明 长度为N、所有项都小于k的逆序表的个数恰好是$k!k^{N-k}$，这是因为当$i\leqslant k$时，第i项可以是0到$i-1$之间的任何值，当$i>k$时，第i项可以是0到$k-1$之间的任何值。因此，最大项小于k的概率就是$k!k^{N-k}/N!$，因而我们所求的均值就为

$$\sum_{0\leqslant k\leqslant N}\left(1-\frac{k!k^{N-k}}{N!}\right)$$

和式中的第二项是“Ramanujan P-函数”，其渐近值已由表4-11给出。 ■

推论 对随机顺序，关键字互异的N个记录的文件排序，冒泡排序平均要做$\sim N^2/2$次关键字比较，并平均移动$\sim N^2/2$次记录（在$\sim N-\sqrt{\pi N/2}$趟中）。

证明 见上面的讨论。 ■

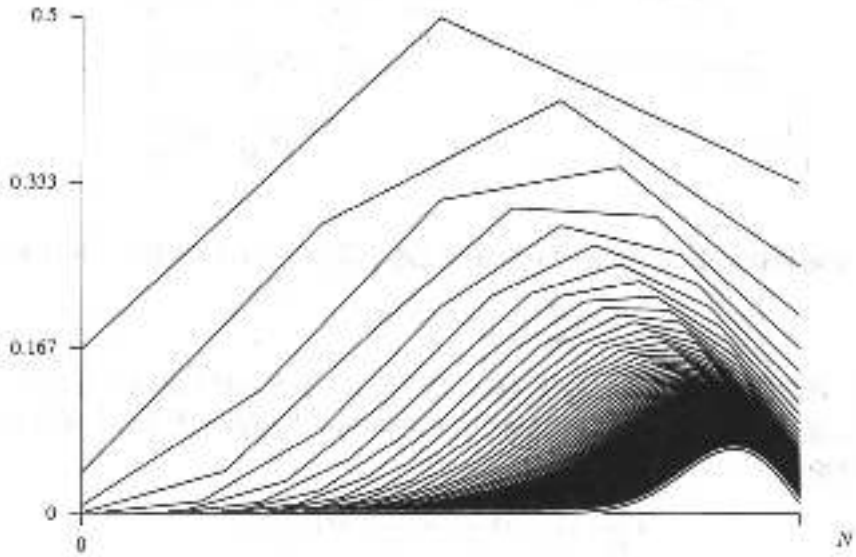


图6-12 最大逆序表项的分布

习题6.64 考虑对冒泡排序的一个修改：扫描数组对换方向（由右向左，然后由左向右），而始终保持后扫描之后对逆序数有什么影响？

最长圈和最短圈。 一个排列中最长圈的平均长度是多少？我们可以立即对此写出一个表达式。在本章开始阶段，曾导出过一个关于对不存在圈长大于k的排列计数的指数生成函数（见定理6.2）：

$$e^z = 1 + z + \frac{z^2}{2!} + \frac{z^3}{3!} + \frac{z^4}{4!} + \frac{z^5}{5!} + \frac{z^6}{6!} + \cdots$$

$$e^{z+z^2/2} = 1 + z + 2\frac{z^2}{2!} + 4\frac{z^3}{3!} + 10\frac{z^4}{4!} + 26\frac{z^5}{5!} + 76\frac{z^6}{6!} + \cdots$$

$$e^{z+z^2/2+z^3/3} = 1 + z + 2\frac{z^2}{2!} + 6\frac{z^3}{3!} + 18\frac{z^4}{4!} + 66\frac{z^5}{5!} + 276\frac{z^6}{6!} + \cdots$$

$$e^{z+z^2/2+z^3/3+z^4/4} = 1 + z + 2\frac{z^2}{2!} + 6\frac{z^3}{3!} + 24\frac{z^4}{4!} + 96\frac{z^5}{5!} + 456\frac{z^6}{6!} + \cdots$$

$$\vdots$$

$$e^{-\ln(1-z)} = \frac{1}{1-z} = 1 + z + 2\frac{z^2}{2!} + 6\frac{z^3}{3!} + 24\frac{z^4}{4!} + 120\frac{z^5}{5!} + 720\frac{z^6}{6!} + \cdots$$

354
1
357

根据这些方程，我们可以写出关于至少有一个圈长大于k的排列的生成函数，或等价地，关于最大圈长大于k的排列的生成函数：

$$\frac{1}{1-z} - e^z = z + 2\frac{z^2}{2!} + 6\frac{z^3}{3!} + 24\frac{z^4}{4!} + 120\frac{z^5}{5!} + 720\frac{z^6}{6!} + \cdots$$

$$\frac{1}{1-z} - e^z = \frac{z^2}{2!} + 5\frac{z^3}{3!} + 23\frac{z^4}{4!} + 119\frac{z^5}{5!} + 719\frac{z^6}{6!} + \cdots$$

$$\begin{aligned}
\frac{1}{1-z}-e^{z+z^2/2} &= 2\frac{z^3}{3!}+14\frac{z^4}{4!}+94\frac{z^5}{5!}+644\frac{z^6}{6!}+\cdots \\
\frac{1}{1-z}-e^{z+z^2/2+z^3/3} &= 6\frac{z^4}{4!}+54\frac{z^5}{5!}+444\frac{z^6}{6!}+\cdots \\
\frac{1}{1-z}-e^{z+z^2/2+z^3/3+z^4/4} &= 24\frac{z^5}{5!}+264\frac{z^6}{6!}+\cdots \\
&\vdots
\end{aligned}$$

由表3-6，一个随机排列中最大圈长的平均值可以通过累加这些方程来求出，并可表示如下:

$$[z^N]\sum_{k\geq 0}\left(\frac{1}{1-z}-e^{z+z^2/2+z^3/3+\cdots+z^k/k}\right)$$

一般情况下，对于极值参数而言，要从这些信息中导出一个渐近结果是相当复杂的。当N不大时，我们可以通过累加上述方程，计算这个量的准确值，从而得到关于最大圈长长度的指数CGF的一些初始项:

$$1\frac{z^1}{1!}+3\frac{z^2}{2!}+13\frac{z^3}{3!}+67\frac{z^4}{4!}+411\frac{z^5}{5!}+\cdots$$

实际上，一个随机排列中最大圈长为$\sim\lambda N$，其中$\lambda\approx 0.62433\ldots$。该结果由Golomb, Shepp和Lloyd[15]于1966年首次导出。

358

习题6.65 对所有的$N<10$，求长度为N的随机排列中最短圈的平均长度。(注：Shepp和Lloyd指出了这个量为$\sim e^{-\gamma}\ln N$，其中γ是欧拉常数。)

排列作为基本组合对象已得到了很好的研究，我们期望对它们性质的了解将有助于我们理解排序算法的性能特征。例如，圈和逆序等基本性质与插入排序、选择排序和冒泡排序等基本算法之间的直接对应关系就证实了我们预期的结果。

目前关于新排序算法的研究以及对它们性能的分析是相当活跃的。排序的变体，如优先队列、归并算法以及排序“网络”等，依然具有实际意义。新型计算机及新的应用需要新的方法和对老方法更好地理解，本章所概述的分析种类是设计和使用此类算法过程中的一个重要组成部分。

正如本章自始至终所提议的，目前已有可用的一般工具[4]能够回答本章提出的许多更为复杂的问题。我们强调了使用累积生成函数来分析排列的性质，因为它们对我们所关注的量的平均值提供了一个直接的“系统的”行动步骤。对排列性质的分析，累积方法比可用的递归法或BGF法常常能够以更简单、更直接的方式给出结果。和以往一样，虽然“垂直”GF可用来计算较小的值并以此开始分析，但分析极值参数（那些与可加性规则相对立的、由一个“最小”或“最大”规则所定义的参数）则要困难得多。

尽管排列作为一个组合对象而言比较简单，但对初学者而言，尚待处理的分析问题其数量之多常常是相当惊人的。然而，我们能够利用一套标准方法来回答有关排列性质的基本问题这一事实还是令人鼓舞的，这不仅仅是因为这些问题中的许多问题都来自重要的应用，而且还因为我们希望能够继续学习更加复杂的组合结构。

参考文献

1. L. Comtet. *Advanced Combinatorics*, Reidel, Dordrecht, 1974.
2. F. N. David and D. E. Barton. *Combinatorial Chance*, Charles Grif-

fin, London, 1962.

3. W. FELLER. *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 1957. 359
4. P. FLAJOLET AND R. SEDGEWICK. *Analytic Combinatorics*, in preparation.
5. G. H. GONNET AND R. BAEZA–YATES. *Handbook of Algorithms and Data Structures*, 2nd edition, Addison-Wesley, Reading, MA, 1991.
6. I. GOULDEN AND D. JACKSON. *Combinatorial Enumeration*, John Wiley, New York, 1983.
7. R. GRAHAM, D. E. KNUTH, AND O. PATASHNIK. *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.
8. D. E. KNUTH. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
9. D. E. KNUTH. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.
10. D. E. KNUTH. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
11. D. E. KNUTH. "Mathematical Analysis of Algorithms," *Information Processing 71*, Proceedings of the IFIP Congress, Ljubljana, 1971, 19–27.
12. V. LIFSCHITZ AND B. PITTEL. "The number of increasing subsequences of the random permutation," *Journal of Combinatorial Theory (Series A)* **31**, 1981, 1–20.
13. B. F. LOGAN AND L. A. SHEPP. "A variational problem from random Young tableaux," *Advances in Mathematics* **26**, 1977, 206–222.
14. R. SEDGEWICK. *Algorithms*, Addison-Wesley, Reading, MA, 1988.
15. L. SHEPP AND S. P. LLOYD. "Ordered cycle lengths in a random permutation," *Transactions of the American Mathematical Society* **121**, 1966, 340–357.
16. J. S. VITTER AND P. FLAJOLET. "Analysis of algorithms and data structures," in *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, 431–524.
17. J. VUILLEMIN. "A unifying look at data structures," *Communications of the ACM* **23**, 4, 1980, 229–239.
18. A. YAO. "Analysis of selection sort," to appear.
19. A. YAO. "An analysis of $(h, k, 1)$ Shellsort," *Journal of Algorithms* **1**, 1980, 14–50. 360

第7章 串与trie树

从一个固定的字母表中抽取的字符或字母的序列叫做*串*（string）。处理串的算法十分普遍，无论从计算理论核心中的基本方法还是到实践中大量重要应用的文本处理方法，都要涉及处理串的算法。本章我们将学习串的基本组合性质、一些有关查找串中模式的算法、以及相应的数据结构。

我们用*位串*（bitstring）这个词来表示仅由两个字符组成的串；如果字母表的大小$M>2$，我们就把串叫做*字节串*或*字*，或M-串（M-ary string）。在本章中，我们假定M是一个不大的固定常数，这个假定对于我们所关心的文本处理和位处理算法是合理的。如果M可以变大（例如，随着串的长度增加而增加），那么我们有一个稍微不同的组合对象——下一章将要提及的课题。这一课题与本章内容是有重要区别的，我们将在第8章中给出详细的讨论。本章我们主要关心的是从固定大小的字母表中产生的潜在的长串，以及它们作为序列时的性质。

从算法的观点看，将注意力集中在位串而不是字节串上并没有失去多少一般性：从一个较大的字母表中产生的字符串与通过对一些单个字符进行二进制编码所产生的位串是相对应的。相反地，从一个较大的字母表中产生的算法、数据结构或串的分析却以某种方式依赖于字母表的大小，位串中同样的依赖关系可以通过考虑块中的位反映出来。M-串与位串之间的这种特定的对应关系仅当M是2的幂时才是准确的。我们很容易把一个算法或分析从位串推广到M-串（本质上是将2全部变成M），所以在适当的时候我们就这样做。

随机位串恰好对应于独立的伯努利试验，后者在古典概率论中已得到了很好的研究；在算法分析中这样的结果是值得注意的，因为许多算法本质上明显地依赖于二进制串的性质。本章及下一章我们将回顾一些相关的经典结果。正如我们在前两章中对树和排列所做的事情一样，我们将从一种计算的观点来考虑问题，并利用生成函数作为组合分析的工具。这种方法对某些古典问题能产生简单的结果，并能给出一个非常通用的框架，在这个框架内可以考虑令人惊异的广大范围内的问题。

我们将考虑关于在一个给定的串中查找一个固定模式的算法，该算法用模式特定的有限状态自动机（FSA）的术语进行最好的描述。FSA不仅导致了统一、简洁、和有效的工具，而且结果证明自动机恰好与相应模式的生成函数相对应。本章我们将详细地研究一些这样的例子。

某些计算工作要涉及对串*集合*的明确的操作。串集合（通常是无限集）称作*语言*，是计算机科学中一个极为重要的外延理论的基础。语言是按照描述组成它们的串的难度来分类的。当前，我们将主要涉及*规则*的语言和*上下文无关*的语言，这些语言将描述许多有趣的组合结构。本章我们将给出许多例子，来阐明在分析语言的性质中生成函数的功用。值得注意的是，我们完全可以描述关于规则语言和上下文无关语言的生成函数的特征，并可以证明这些特征在本质上是根本不同的。

一个重要的叫做trie树的数据结构经常被用在处理串和有限串集合的算法中。trie树是类似于树的对象，其结构是由一组位串的位值所决定的。如果位串是独立的并且是随机选择的话，那么trie树就是一个具有大量有趣性质的组合对象。本章我们将着眼于基本的trie树算法、trie树的性质、以及相应的生成函数。trie树不仅在广泛的应用领域中有用，对它们的分析也能展现和引出许多重要的算法分析工具。

7.1 串查找

我们通过考虑一个关于“串查找”的基本算法来开始本章内容：给定一个长度为k的模式和某个长度为N的文本，并在文本中寻找该模式的出现。程序7.1给出了这个问题的直接解法。对文本中的每一个位置，程序从这个位置开始与模式（从头开始）一个字符一个字符地进行比较，以检查是否存在一个匹配。程序假定了两个不同的标记字符，一个用于模式的尾部（第$(k+1)$个模式字符），一个用于文本的尾部（第$(N+1)$个文本字符）。这样所有串的比较都将终止于字符的一个不匹配，并且我们可以通过简单地检查是不是标记引起的不匹配来判断模式是否在文本中。

362

程序7.1 串查找的基本方法

```
procedure stringsearch;
  var i, j:  integer;
  begin
  for i := 1 to N do
    begin
    j := 0;
    repeat j := j+1 until a[i+j-1] <> p[j];
    if j = k+1 then << match found >>
    end
  end;
```

针对不同的应用，基本算法的许多不同变化可能会值得我们关注：

- 当找到第一个匹配时结束；
- 打印出所有匹配的位置；
- 统计匹配的次数；
- 找出最长的匹配。

在程序7.1中，通过在指定的点，即找到匹配的地方增加适当的编码，上述变化就可以很容易地得以实现。在某些应用中，文本串尾部处的部分匹配可能也需要关注；如果不是这种情况，则当文本中所剩文本字符的个数少于k时，就可以通过结束循环来改进此程序。在其他各种具体情况下，类似的改进可能更适合实际情况。程序7.1所给出的基本实现对许多串查找应用来说是一个合理的通用方法，不过我们在本章还将看到更好的方法。

尽管对某些模式而言，现在已经有了一些对基本算法的小改进，但目前我们还是采用一个任意的但也是固定的模式。例如，我们可以只通过保留一个计数器、并扫描整个文本来实现查找k个连续的0字符的串（一个*游程*）：当遇到一个1时，将计数器清零，当遇到一个0时，将计数器增1，并当计数器达到k时停止扫描。我们也可反过来考虑这样的问题：当程序7.1查找k个连续0字符的串（假定k比较大）时，如果当它遇到一个小的游程，比方说五个0后跟一个1的游程时，程序7.1的执行过程是怎样的。它将检查所有五个0，当碰到1时确定出存在一个不匹配，然后将文本指针仅仅增1，因此，当它再检查了四个0和一个1后，又确定出一个不匹配，然后再检查三个0和一个1，依此类推。该算法对每个t个0的游程检验了$(t+1)(t+2)/2$个位之后才结束。在本章稍后，我们将探讨为避免这种重复性检查而对基本算法所进行的改进。目前，我们所关心的是探讨如何来分析这个基本算法。

363

“所有匹配”变体的分析。我们感兴趣的是在一个随机文本中找出程序7.1的平均运行时间。显然，运行时间和查找过程中要检查的字符个数成正比。由于每个串比较都在不匹配时结束，因此它的开销是1加上文本位置处那些与模式相匹配的字符的个数。表7-1对4位模式中的每种模式在一个样本文本串中的开销给出了结果。在文本串的每个位置上我们都关联了一个整数，该整数表示从文本位置起与模式相匹配的字符位置的个数。这些数的和加上N（不匹配的次数）

就是程序7.1中内层循环的迭代次数，显然它是程序运行时间的主项。利用累积法，通过一个简单的计数变量，我们就可以算出这个值。

表7-1 查找4位模式的开销（基本方法）

	0	1	1	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	总计
0000	1	0	0	0	1	0	3	2	1	0	4	4	3	2	1	0	2	1	0	4	4	3	2	1	0	0	39
0001	1	0	0	0	1	0	4	2	1	0	3	3	4	2	1	0	2	1	0	3	3	4	2	1	0	0	39
0010	1	0	0	0	1	0	2	4	1	0	2	2	2	4	1	0	4	1	0	2	2	2	3	1	0	0	35
0011	1	0	0	0	1	0	2	3	1	0	2	2	2	3	1	0	3	1	0	2	2	2	4	1	0	0	33
0100	2	0	0	0	4	0	1	1	4	0	1	1	1	1	4	0	1	4	0	1	1	1	1	2	0	0	31
0101	2	0	0	0	3	0	1	1	3	0	1	1	1	1	3	0	1	3	0	1	1	1	1	2	0	0	27
0110	3	0	0	0	2	0	1	1	2	0	1	1	1	1	2	0	1	2	0	1	1	1	1	3	0	0	22
0111	4	0	0	0	2	0	1	1	2	0	1	1	1	1	2	0	1	2	0	1	1	1	1	3	0	0	25
1000	0	1	1	2	0	4	0	0	0	4	0	0	0	0	0	3	0	0	4	0	0	0	0	0	1	1	21
1001	0	1	1	2	0	3	0	0	0	3	0	0	0	0	0	4	0	0	3	0	0	0	0	0	1	1	19
1010	0	1	1	2	0	2	0	0	0	2	0	0	0	0	0	2	0	0	2	0	0	0	0	0	1	1	14
1011	0	1	1	2	0	2	0	0	0	2	0	0	0	0	0	2	0	0	2	0	0	0	0	0	1	1	14
1100	0	2	3	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	2	1	13
1101	0	2	4	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	2	1	14
1110	0	4	2	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	2	1	14
1111	0	3	2	1	0	1	0	0	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	0	2	1	13

定理7.1（模式出现的计数） *一个长度为k的任意固定模式在一个长度为N的随机位串中出现的期望次数是$(N-k+1)/2^k$。*

证明 使用累积计数法，我们在所有2^N个N位位串中来统计模式出现的总次数。k位可以起始于$N-k+1$个位置中的任何一个位置，且对每个位置，有2^{N-k}个位串与模式同在那个位置。累积所有这些项即可得出所求的总数为$(N-k+1)2^{N-k}$，再除以2^N，即得到所述结果。■

推论 *在一个长度为N的文本串中，寻找一个长度为k的任意固定模式的所有出现时，由基本串查找算法所做的位比较的期望次数是$N(2-2^{-k})+O(1)$。*

证明 为找出算法在查找过程中所检查的位数，我们注意到可以用另外一种方式来解释表7-1中的数：它们记录了在文本中起始于相应位置的模式的前缀数。现在我们也可以利用上面的公式来统计前缀数。由此得出在所有2^N个N位位串中，模式的前缀出现的总次数的表达式

$$\sum_{1\le j\le k}(N-j+1)2^{N-j}$$

计算这个和式，将其除以2^N，再加上N（不匹配的次数），即可得到所述结果。■

推论 *在一个随机位串中要找到一个任意的无限长模式的最长匹配，基本串查找算法所检查的平均位数是~$2N$。*

以上这些结果都和模式无关。这似乎有悖于我们的直觉，因为我们知道，要查找一个由0构成的长串时，将要对文本中每个t个0的游程检查$O(t^2)$个位，而查找一个1后接一个由0构成的长串时却没有这种明显的“二次”行为。这种现象可以通过注意下面的事实来解释：后一种模式的前缀是在文本中的某个地方，但该前缀并没有像由0构成的串那样被捆绑在一起。因为我们使用的是累积总数，所以我们不必担心前缀不同的实例之间的独立性问题；我们对它

364 ~ 365

们统统计数。相反地，寻找第一个匹配所需的时间确实与模式有关，即使是对随机文本也是如此。这和我们要关注的另一个量——模式在文本中不出现的概率——有直接的联系。下一节，我们将探讨这些分析结果的细节。

为了重述这个区别，我们考虑使用OGF和关于这一问题的符号法。计数长度为k的一个任意固定模式的出现次数的OGF为

$$\frac{1}{1-2z}z^k\frac{1}{1-2z}$$

这是因为这种固定模式的出现是通过连接一个任意的字、该模式以及另一个任意的字而构成的。在计算中（模式的每个出现计数一次），某个特定的位串可能被计数过多次，这恰好就是我们所要寻找的。因此，和定理7.1中一样，在所有长度为N的位串中，一个长度为k的固定模式出现的次数是

$$[z^N]\frac{1}{1-2z}z^k\frac{1}{1-2z}=[z^{N-k}]\frac{1}{(1-2z)^2}=(N-k+1)2^{N-k}$$

此外，这个数与含有一个任意固定模式（一次或多次出现）的长度为N的位串个数是不同的。

本章稍后，我们将探讨对基本方法进行算法改进的问题。首先，我们将考查Knuth-Morris-Pratt算法，它是一个这样的算法：通过利用与模式长度成正比的预处理时间以获得一个“最佳”的查找时间，使得文本中的每个字符最多被检查一次。在本章的末尾我们还将看到，随着在预处理中更多的投入，文本可以被构建成一种数据结构，这种结构与一个叫做trie树的、允许模式查找在与模式长度成正比的时间内完成查找的一个通用结构相联系。trie树对大量其他有关位串的算法也提供了有效的支持，但在涉及基本的分析结果之前，我们先不考虑这些问题。

7.2 位串的组合性质

366 研究等可能的2^N个长度为N的由0和1所组成的随机字符串（位串）的性质是值得我们去关注的。位串是由OGF

$$S(z)=\sum_{s\in S}z^{|s|}=\sum_{N\geq 0}\{\text{长度为}N\text{的位串个数}\}z^N$$

来计数的，其中S表示所有位串的集合。位串要么为空串，要么第一个字符为0或1，因此

$$S(z)=1+2\sum_{s\in S}z^{|s|+1}=1+2zS(z)$$

所以$S(z)=(1-2z)^{-1}$，这正如我们所期望的一样，长度为N的位串数是2^N。同前两章中的排列和树一样，我们可以修改这个基本变量，以便研究位串更多有趣的性质。

例如，我们已经遇到过在一个随机位串中对1进行计数的问题，这是3.12节中的一个例子：其相应的二元生成函数涉及二项分布

$$P(u,z)=\sum_{s\in S}u^{v(s)}z^{|s|}=\frac{1}{1-z(1+u)}=\sum_N\sum_k\binom{N}{k}u^kz^N$$

我们曾用定理3.11计算过在一个N位随机串中位为1的平均个数是

$$[z^N]P_u(1,z)/2^N=N/2$$

等等。我们将在第8章中像刚才那样更详细地考查“全局”性质；在本章中，我们将主要把精

力更多地集中在涉及那些串中相互临近的位的“局部”性质上。

研究随机位串的性质等价于研究独立伯努利试验（或抛掷硬币）的性质，所以我们要用到概率论中的一些经典结果。本章我们不仅要把注意力特别集中在试验上，还要集中在事件的序列上，还有一些经典的结果与此有关。在概率论中，我们要考查随机试验的一个序列的性质并研究“等待时间”（比如，见Fller[6]）；在算法研究中，通过取序列中的位，我们将考查关于串中的模式查找这样的一个等价问题，这是用一种自然的方法来考虑相同的问题。正如我们将要看到的，这种考查的结果表明：形式语言理论和生成函数的结合，为与伯努利试验序列相联系的解析现象的研究提供了一个清晰的解释。 367

0的游程。我们从考查下面的基本问题开始：在一个随机位串中，我们期望在什么地方找到第一个有连续k个0的位串？表7-2给出了十个长随机位串以及第一次出现一个、两个、三个和四个0的位置。结果表明：生成函数对这个问题可导出一个简单的解，并且这个解可以代表许多类似的问题。

表7-2 样本位串中0的第一个游程的位置

	1	2	3	4
0101001110101000011001110001011101100011011111010	0	4	13	13
0111010101000101001001100101000010000101011010010 1	0	10	10	28
0101110101101111001000111000000101011001001111000 0	0	16	19	25
1010100001010011101010101111000011111000011100100 1	1	5	5	5
1111111100100000100100110010011000000010000011000 1	8	8	11	11
0010001000110010111001110000110010100000101100111 1	0	0	3	24
0101101111011001011000010010100101000010100111111 0	0	13	19	19
1001101000001001100101001010001100000111101001101 0	1	1	7	7
0110001001100110110001011111000100100011100111101 0	0	3	3	—
0101100100011000000100011001010101110011110010011 0	0	5	8	13
平　均	1.0	6.5	9.8	—

定理7.2 (0的游程) *不存在k个连续0的游程的位串个数的生成函数是*

$$S_k(z)=\frac{1-z^k}{1-2z+z^{k+1}}$$

证明 设S_k是不存在k个连续0的游程的位串数，则

$$S_k(z)=\sum_{s\in S_k} z^{|s|}=\sum_{N\geq 0}\{\text{长度为}N\text{的不存在}k\text{个0游程的位串个数}\}z^N$$ 368

于是，任何一个不含k个连续0的位串要么是（i）空串或由零到$k-1$个0构成的串；或是（ii）零到$k-1$个0，后接一个1，再后接任何一个不含k个连续的0的位串。根据位串中0的初始个数建立起这种对应关系，就可以得到一个函数方程，然后再得到关于OGF的一个显式表达式：

$$S_k(z)=1+z+\cdots+z^{k-1}+\sum_{s\in S_k 0}\sum_{0\leq j<k} z^{|s|+j+1}$$

$$=\frac{1-z^k}{1-z}(1+zS_k(z))$$

对此式求解，我们得到

$$S_k(z)=\frac{\dfrac{1-z^k}{1-z}}{1-z\dfrac{1-z^k}{1-z}}=\frac{1-z^k}{1-2z+z^{k+1}}$$

■

这些函数都是有理函数，因此如果必要的话可以将它们展开。考查较小的k值，对$k=1$、2、3，我们有如下展开式:

$$\frac{1-z}{1-2z+z^2}=1+z+z^2+z^3+z^4+z^5+z^6+z^7+\cdots$$

$$\frac{1-z^2}{1-2z+z^3}=1+2z+3z^2+5z^3+8z^4+13z^5+21z^6+34z^7+\cdots$$

$$\frac{1-z^3}{1-2z+z^4}=1+2z+4z^2+7z^3+13z^4+24z^5+44z^6+81z^7+\cdots$$

对于$k=1$，上式证实了长度为N的不含一个0的游程的串（全是1的串）有一个。对$k=2$，我们有

$$S_2(z)=\frac{1+z}{1-z-z^2}\qquad \text{所以 } [z^N]S_2(z)=F_{N+1}+F_N=F_{N+2}$$

这是斐波那契数（见2.4节）。事实上，表达式

$$S_k(z)=\frac{1+z+z^2+\cdots+z^{k-1}}{1-z-z^2-\cdots-z^k}$$

表明了$[z^N]S_k(z)$与习题4.18中所推广的斐波那契数满足相同的递推关系，但具有不同的初值。因此，定理4.1可用来求关于$[z^N]S_k(z)$的渐近估计。

369

推论 长度为N的不存在k个连续0的游程的位串数渐近于$c_k\beta_k^N$，其中β_k是多项式$z^k-z^{k-1}-\cdots-z-1=0$的最大模数的根而$c_k=(\beta^k+\beta^{k-1}+\cdots+\beta)/(\beta^{k-1}+2\beta^{k-2}+3\beta^{k-3}+\cdots+(k-1)\beta+k)$。

证明 由习题4.18和定理4.1可立即得证。对于较小的k值，表7-3给出了c_0和β的近似值。■

表7-3 不含k个0的游程的位串计数（$[z^N]S_k(z)\sim c_k\beta_k^N$）

k	$S_k(z)$	c_k	β_k
2	$\frac{1-z^2}{1-2z+z^3}$	1.17082…	1.61803…
3	$\frac{1-z^3}{1-2z+z^4}$	1.13745…	1.83929…
4	$\frac{1-z^4}{1-2z+z^5}$	1.09166…	1.92756…
5	$\frac{1-z^5}{1-2z+z^6}$	1.05753…	1.96595…
6	$\frac{1-z^6}{1-2z+z^7}$	1.03498…	1.98358…

习题7.1 给出$[z^N]S_k(z)$所满足的两个递推关系。

习题7.2 一个随机位串应该取多长时，才能有99%的把握保证至少有三个连续的0?

习题7.3 一个随机位串应该取多长时，才能有50%的把握保证至少有32个连续的0？

习题7.4 证明

$$[z^N]S_k(z)=\sum_i(-1)^i2^{N-(k+1)i}\left(\binom{N-ki}{i}-2^{-k}\binom{N-k(i+1)}{i}\right)$$

第一个0游程。计算第一个具有k个0的游程的平均位置的捷径是现成的，因为计算不含k个连续0的位串数的OGF与关于一个随机串中第一个具有k个连续0的游程中最后一位（结束）的位置的PGF有着紧密的联系，见如下运算： 370

$$\begin{aligned}S_k(z)&=\sum_{s\in S_k}z^{|s|}\\&=\sum_{N\geq 0}\{\text{长度为}N\text{的不含}k\text{个0的游程的位串数}\}z^N\\S_k(1/2)&=\sum_{N\geq 0}\{\text{长度为}N\text{的不含}k\text{个0的游程的位串数}\}/2^N\\S_k(1/2)&=\sum_{N\geq 0}\Pr\{\text{不含}k\text{个0的游程的随机位串的第一个}N\text{比特位}\}\\&=\sum_{N\geq 0}\Pr\{\text{具有}k\text{个0的第一个游程的结束位置}>N\}\end{aligned}$$

这个累积概率之和与我们的期望是相同的。

推论 *在一个随机位串中，第一个含有k个0的游程的平均结束位置是$S_k(1/2)=2^{k+1}-2$。*

生成函数可以大大简化期望的计算；欢迎任何怀疑这一事实的读者验证，比如说，通过关于第一个含有k个0的游程出现在位置j的概率为P_j来导出一个递推关系，然后计算$\sum jp_j$来验证推论7.4的结果。对于排列，我们发现N元素排列的总数$N!$使我们导出了EGF；对于位串，N元素位串的总数将使我们导出和上面一样的关于$z/2$的函数方程。

存在性。对定理7.2的第二个推论的证明也说明了求一个模式的第一个出现大约等价于计算不含该模式的串的个数，它也告诉了我们一个随机串中不含k个0的游程的概率是$[z^N]S_k(z/2)$。例如，对$k=1$，概率是$1/2^N$，因为只有全部是1的串才不含k个0的游程。对$k=2$，概率是$O((\phi/2)^N)$（其中$\phi=(1+\sqrt{5})/2=1.61803\cdots$），且以$N$的指数形式递减。对固定的$k$，这个结论总是成立的，因为在表7-3中，$\beta_k$保持严格小于2。稍微再进行一点详细的分析，就能揭示出一旦当N增加到超过2^k时，某个k位模式不出现的事件就变得越来越不可能。例如，根据表7-3进行一个快速计算就能说明：一个10-位串中不含六个0的游程的可能性为92%，一个100-位串不含六个0的游程的可能性为44%，一个1000-位串不含六个0的游程的可能性为0.02%。 371

最长游程。一个随机位串中最长的0的游程的平均长度是多少？图7-1给出了这个量的分布。正如我们在第5章中对树高和第6章中对排列中的圈长所做过的事情一样，我们可以通过累加上面所给出的“垂直”GF来得到一个关于在一个随机N-位串中最长0的串的平均长度的表达式：

$$\frac{1}{2^N}[z^N]\sum_{k\geq 0}\left(\frac{1}{1-2z}-\frac{1-z^k}{1-2z+z^{k+1}}\right)$$

Knuth[18]曾研究过一个非常类似的量，是关于在异步加法器中确定进位传送时间的应用中的一个量，并证明了这个量为$\ln N+O(1)$。其常数项有一个摆动行为；对图7-1仔细检查，将会使我们对其原因有所顿悟。实际上，描述摆动的函数与我们将要在本章末关于trie树分析的详

细研究是一致的。

习题7.5 求关于随机位串中前面的位全为1的串的个数的二元生成函数，并由此来计算这个量的平均值和标准差。

习题7.6 通过考虑不含两个连续的0的游程的位串，计算下面的关于斐波那契数的和：$\sum_{j\geq 0} F_j/2^j$。

习题7.7 求关于位串中最长的0的游程长度的BGF。

习题7.8 在一个随机位串中，标示k个0的游程的首次出现的随机变量的标准差是什么？

习题7.9 对$2<N<100$，用一个计算机代数系统做出N-位随机位串中最长的0游程的平均长度的曲线图。

习题7.10 当利用上节给出的基本算法求一个随机位串中第一个k个0的串时，需要检查多少个位？

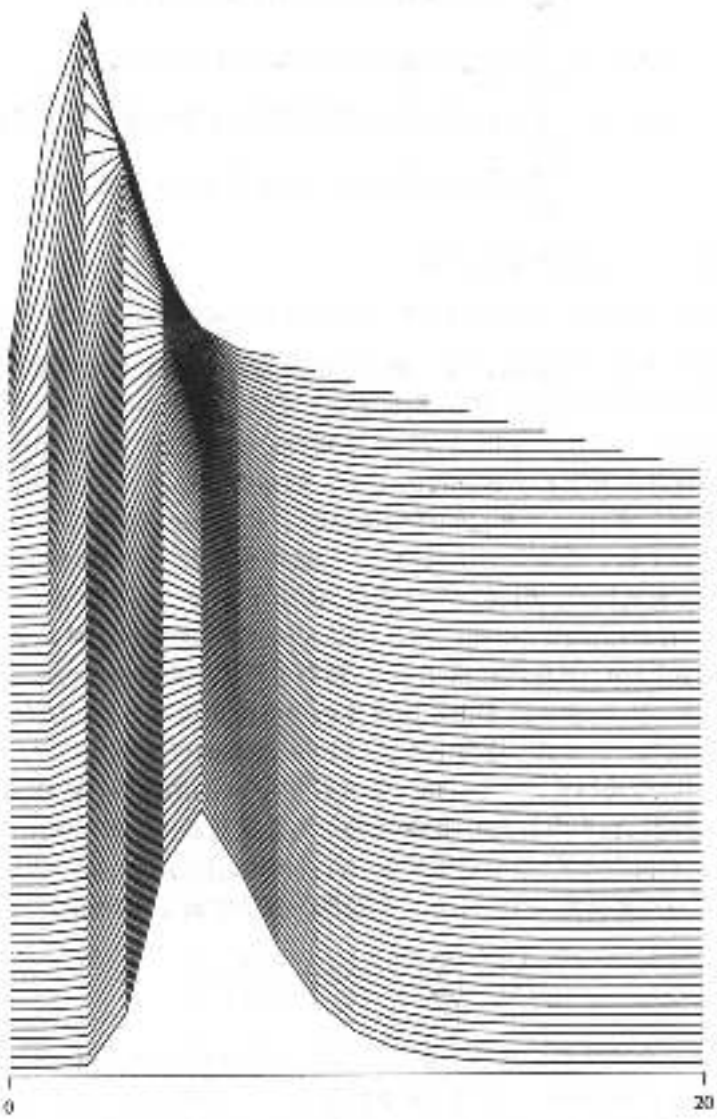


图7-1 一个随机位串中最长0游程的分布（水平轴解释为分离的曲线）

任意模式。开始时，我们可能会猜想上面的结果对任意固定的k位模式都成立，但这恰恰是错误的：在一个随机位串中，一个固定位模式的第一个出现的平均位置在很大程度上依赖于模式本身。例如通过下面的观察我们很容易就会发现：平均而言，像0001这样的模式往往要比0000这样的模式先出现，因为一旦000已经匹配，那么下一个字符对于两种模式出现的概率都是1/2，然而，对于0000的一个不匹配则意味着0001在文本中，并且再往后考虑四个字符也不可能存在任何的匹配，但是对于0001的不匹配则意味着0000在文本中，并且在文本中下一个位上就可能存在一个匹配。实际上，模式的依赖关系可以通过用一个函数来匹配该模式本身的方法很容易地表示出来。

372 ~ 373

定义 一个位串$b_0b_1\cdots b_{k-1}$的自相关是位串$c_0c_1\cdots c_{k-1}$，其中c_i的定义是：对$0 \leqslant j \leqslant k-1-i$，若$b_j = b_{i+j}$，则$c_i = 1$，否则$c_i = 0$。其相应的自相关多项式可以通过取位作为系数而得到：$c(z) = c_0 + c_1z + \cdots + c_{k-2}z^{k-2} + c_{k-1}z^{k-1}$。

自相关很容易计算，第i位可以这样来确定：向左移动i个位置，如果剩余的那些位与原始模式相匹配，则i位为1，否则为0。例如，表7-4说明了101001010的自相关是100001010，且其相应的自相关多项式为$1 + z^5 + z^7$。注意：c_0总是1。

表7-4 101001010的自相关

								1	0	1	0	0	1	0	1	0	
								1	0	1	0	0	1	0	1	0	1
							1	0	1	0	0	1	0	1	0		0
						1	0	1	0	0	1	0	1	0			0
					1	0	1	0	0	1	0	1	0				0
				1	0	1	0	0	1	0	1	0					0
			1	0	1	0	0	1	0	1	0						1
		1	0	1	0	0	1	0	1	0							0
	1	0	1	0	0	1	0	1	0								1
1	0	1	0	0	1	0	1	0									0

定理7.3（模式自相关） 不含模式$p_0p_1\cdots p_{k-1}$的位串个数的生成函数由下式给定

$$S(z) = \frac{c(z)}{z^k + (1-2z)c(z)}$$

其中$c(z)$是关于该模式的自相关多项式。

证明 可利用符号法对前面给出的关于k个连续的0的模式情况的证明进行推广。我们从$\mathcal{S}_p$的OGF开始，不含p的位串的集合是：

374

$$S(z) = \sum_{s \in \mathcal{S}_p} z^{|s|} = \sum_{N \geqslant 0} \{\text{不含}p\text{的长度为}N\text{的位串数}\} z^N$$

类似地，我们定义$\mathcal{T}_p$为以p结束但不含p的其他出现的长度为N的位串的类，并命名其相应的生成函数为$T(z)$。

现在，我们来考虑$\mathcal{S}_p$和$\mathcal{T}_p$两个符号之间的关系，这两个符号都能被翻译成关于$S(z)$和$T(z)$的联立方程。首先，$\mathcal{S}_p$和$\mathcal{T}_p$是不相交的，并且如果我们从属于$\mathcal{S}_p$或$\mathcal{T}_p$的一个位串中去掉最后一位的话，我们将得到$\mathcal{S}_p$中的一个位串（或空的位串）。用符号形式表示出来就是

$$\mathcal{S}_p + \mathcal{T}_p = \varepsilon + \mathcal{S}_p \times \{0,1\}$$

因为关于{0, 1}的OGF是$2z$，所以上述表示式可以翻译成

$$S(z) + T(z) = 1 + 2zS(z)$$

其次，考虑由$\mathcal{S}_p$中的一个串后接该模式所构成的串的集合。对于该模式自相关中的每个位置i，它都给出了$\mathcal{T}_p$中的一个串后接一个i-位的“尾部”。用符号形式表示出来就是

$$\mathcal{S}_p \times \langle\text{模式}\rangle = \mathcal{T}_p \times \sum_{c_i=0} \langle\text{尾巴}\rangle_i$$

因为<模式>的OGF是z^k，且<尾部>$_i$的OGF是z^i，所以上式可翻译成

$$S(z)z^k = T(z)\sum_{c_i=0} z^i = T(z)c(z)$$

将相应于OGF的$S(z)$和$T(z)$的两个方程联立求解，将立即得出定理所述的结果。 ■

对于由k个0（或k个1）构成的模式，其自相关多项式是$1 + z + z^2 + \cdots + z^{k-1} = (1 - z^k)/(1 - z)$，所以定理7.3和前面的定理7.2中的结果是一致的。

375

推论 具有自相关多项式$c(z)$的一个位串的首次出现的期望结尾位置由$2^k c(1/2)$给出。

表7-5给出了关于不含16个四位模式中的每一个模式的位串个数的生成函数。我们把这些模式分成了具有相同自相关的四个不同的模式组。对于每一组，该表还给出了OGF的分母中多项式的主根以及期望“等待时间”（模式的第一个出现位置），这个期望“等待时间”是由自相关多项式计算出来的。用与定理7.2的推论处理表7-3一样的方法，我们可以利用定理4.1来导出这些近似值，并应用它们来近似等待时间。也就是说：一个N-位串不含模式1000的概率大约是$(1.83929/2)^N$，等等。因此，比如说，一个10-位串不含1000的可能性大约是43%，与此相应，一个10-位串不含1111的可能性大约是69%。

表7-5 关于4-位模式的生成函数和等待时间

模 式	自 相 关	OGF	主 根	等 待
0000 1111	1111	$\frac{1-z^4}{1-2z+z^5}$	1.92756…	30
0001 0011 0111 1000 1100 1110	1000	$\frac{1}{1-2z+z^4}$	1.83929…	16
0010 0100 0110 1001 1011 1101	1001	$\frac{1+z^3}{1-2z+z^3-z^4}$	1.86676…	18
0101 1010	1010	$\frac{1+z^2}{1-2z+z^2-2z^3+z^4}$	1.88320…	20

值得注意的是，这样的结果通过生成函数竟然如此容易得到。尽管生成函数有此重要的本质和广泛的应用，但直到串-查找算法的系统性分析得到尝试之后，观察此类问题的这种简单方法才变得明显。这些结果以及更多的相关结果在Guibas和Odlyzko[14][15]的文章中都给出了详细的阐述。

习题7.11 计算下列模式中的每一个模式在一个随机位串中第一次出现的期望位置：(i) $k-1$个0后跟一个1；(ii) 一个1后跟$k-1$个0；(iii) 长度为$2k$的0 - 1交替的串；(iv) 长度为$2k+1$的0 - 1交替的串。

376

习题7.12 在一个随机位串中，长度为k的位模式中的哪一些可能最早出现？哪些模式可能最晚出现？

习题7.13 在一个随机位串中，标记长度为k的一个位模式的第一个位置的随机变量的标准差依赖于该模式吗？

较大的字母表。上述方法可直接应用到较大的字母表中去。例如，一个实质上和定理7.3相同的证明将指出：如果一个M-字符的字母表中不存在某个特定字符的k个连续出现的游程时，那么关于位串的生成函数是

$$\frac{1-z^k}{1-Mz+(M-1)z^{k+1}}$$

且在M-字符的字母表的一个随机串中，某个特定字符的k次出现的第一个游程的平均结束位置是$M(M^k-1)/(M-1)$。

习题7.14 假设一只猴子在一个32-键的键盘上随机击键。在它碰巧打出短语TO BE OR NOT TO BE之前，它已打出的字符数的期望值是多少？

7.3 规则表达式

使用上述生成函数的基本方法可以进行很多推广。为确定随机串的性质，我们最终导出这样的一些生成函数：它们可以对那些性质有着准确定义的串集合的基数进行计数。但是，研究串集合的具体描述方法已被归入了*形式语言*（formal languages）的论域，有关这一课题的文献很丰富。和任何一本标准教材所描述的一样，如Eilenberg[5]，我们仅使用形式语言的基本原理。

形式语言理论中最基本的概念就是*规则表达式*（regular expression），这是一种基于并、连接、和“星”操作的串集合的描述方法，这种方法可以描述如下。一个串集合（一个语言）是规则的，如果它能用规则表达式来描述。例如，描述不含四个连续的0的游程的所有位串的规则表达式是

$$S_4 = (1 + 01 + 001 + 0001)*(\varepsilon + 0 + 00 + 000)$$

在这个表达式中，+表示语言的并；两个语言的乘积将被解释为通过连接第一个语言中的一个串与第二个语言中的一个串所形成的串的语言；*是连接一个语言自身任意多次（包括0次）的简写。和以前一样，ε表示空字。我们在前面导出过相应该语言的OGF为 377

$$S_4(z) = \sum_{s\in S_4} z^{|s|} = \frac{1-z^4}{1-2z+z^5}$$

并通过对生成函数的操作，也推导过该语言的基本性质。我们将看到，在此意义下所考虑的其他语言也能很容易地用规则表达式来描述，因此也很容易用OGF来分析。

我们有一种相当简单的方法，能把串集合的形式描述（规则表达式）转换成对它们进行计数（OGF）的形式分析工具。这主要归功于Chomsky和Schützenberger[2]。唯一的要求是规则表达式要具有无二义性：语言中任意一个串都必须只有一种方法来导出。从形式语言理论中我们知道，任何一个规则语言都可以通过一个无二义性的规则表达式来指定。

定理7.4（关于规则表达式的OGF） *设$\mathcal{A}$和$\mathcal{B}$是无二义性的规则表达式，并假定$\mathcal{A}+\mathcal{B}$，$\mathcal{A}\times\mathcal{B}$和$\mathcal{A}^*$也是无二义性的。如果$A(z)$是对$\mathcal{A}$计数的OGF，$B(z)$是对$\mathcal{B}$计数的OGF，则*

$$A(z)+B(z)\text{是对}\mathcal{A}+\mathcal{B}\text{计数的OGF;}$$

$$A(z)B(z)\text{是对}\mathcal{AB}\text{计数的OGF;}$$

$$\frac{1}{1-A(z)}\text{是对}\mathcal{A}^*\text{的OGF。}$$

此外，对规则语言计数的OGF是有理函数。

证明　第一部分本质上与我们关于OGF的符号法的基本定理（定理3.3）是一致的，但在这里值得重述一下，因为这种应用非常重要。

如果a_N是$\mathcal{A}$中长度为N的串数，b_N是$\mathcal{B}$中长度为N的串数，那么$a_N + b_N$是$\mathcal{A} + \mathcal{B}$中长度为$N$的串数，这是因为语言无二义性的要求隐含了$\mathcal{A}\cap\mathcal{B}$为空。

378

类似地，和定理3.3的证明完全一样，我们可以使用一个简单的卷积来证明关于$\mathcal{AB}$的转换，并证明符号表示法

$$\mathcal{A}^* = \varepsilon + \mathcal{A} + \mathcal{A}^2 + \mathcal{A}^3 + \mathcal{A}^4 + \cdots$$

隐含了关于$\mathcal{A}^*$的规则。此外，语言无二义性的要求保证了语言中的串是作为不同的组合对象而生成的，因而是被唯一计数的。

定理的第二部分是由我们前面所说过的话——每个规则语言都可以通过无二义性的规则表达式来指定——而导出来的结果。对于具备形式语言知识的读者而言：如果一个语言是规则的，那么它可以被一个决定性的FSA所识别，Kleene定理的经典证明也把一个无二义性的规则表达式与一个决定性的自动机联系在了一起。我们在下面将探讨一些相应于FSA的算法含义。 ■

这样，我们就有了一个简单而直接的方法，把一个规则表达式转换成一个计数由该规则所描述的串的OGF，只要该规则表达式是无二义性的即可。此外，相继应用定理7.4时所得到的生成函数总是有理函数这一事实的一个重要意义就是：使用诸如定理4.1这样的一般工具，可以对OGF的系数进行渐近估计。

不含k个0的游程的串。前面，我们给出了关于S_k——不含k个连续0的出现的位串集合的一个规则表达式。例如，我们来考虑S_4。由定理7.4，我们可以立即找到关于

$$1 + 01 + 001 + 0001\text{的OGF是}z + z^2 + z^3 + z^4$$

且关于

$$\varepsilon + 0 + 00 + 000\text{的OGF是}1 + z + z^2 + z^3$$

所以，和以前一样，我们有

$$S_4(z) = \frac{1+z+z^2+z^3}{1-(z+z^2+z^3+z^4)} = \frac{\dfrac{1-z^4}{1-z}}{1-z\dfrac{1-z^4}{1-z}} = \frac{1-z^4}{1-2z+z^5}$$

379

三的倍数。规则表达式

$$(1(01^*0)^*10^*)^*$$

可以生成串集合11,110,1001,1100,1111,…，它们在二进制表示法中都是3的倍数。应用定理7.4，我们可以找到关于长度为N的这种串的个数的生成函数：

$$\frac{1}{1-\dfrac{z^2}{1-\dfrac{z^2}{1-z}}\left(\dfrac{1}{1-z}\right)} = \frac{1}{1-\dfrac{z^2}{1-z-z^2}} = \frac{1-z-z^2}{1-z-2z^2} = 1-\frac{z^2}{(1-2z)(1+z)}$$

这和3.3节中我们所遇到的第一批GF中的一个非常类似：对这个式子进行部分分式的展开，就可以得出结果$(2^{N-1}+(-1)^N)/3$。正如我们所期望的那样，在所有起始于1的位串中，大约有三分之一代表能够被3整除的数。

随机途径的高度。把0当作“上”，把1当作“下”，我们就可以引出位串与随机途径之间的一个对应关系。图7-2给出了三个随机途径的例子。如果我们限定当途径第一次到达起始高度时它们就结束（决不降到低于起始高度），那么我们所得到的途径就对应于5.11节中的赌徒破产序列。于是我们就能很容易地按照高度（所取得的最大值）、用规则表达式来统计这些途径：

	规则表达式	生成函数
高度<1	(10)*	$\frac{1}{1-z}$
高度<2	(1(10)*0)*	$\frac{1}{1-\frac{z}{1-z}}=\frac{1-z}{1-2z}$
高度<3	(1(1(10)*0)*0)*	$\frac{1}{1-\frac{z}{1-\frac{z}{1-z}}}=\frac{1-2z}{1-3z+z^2}$

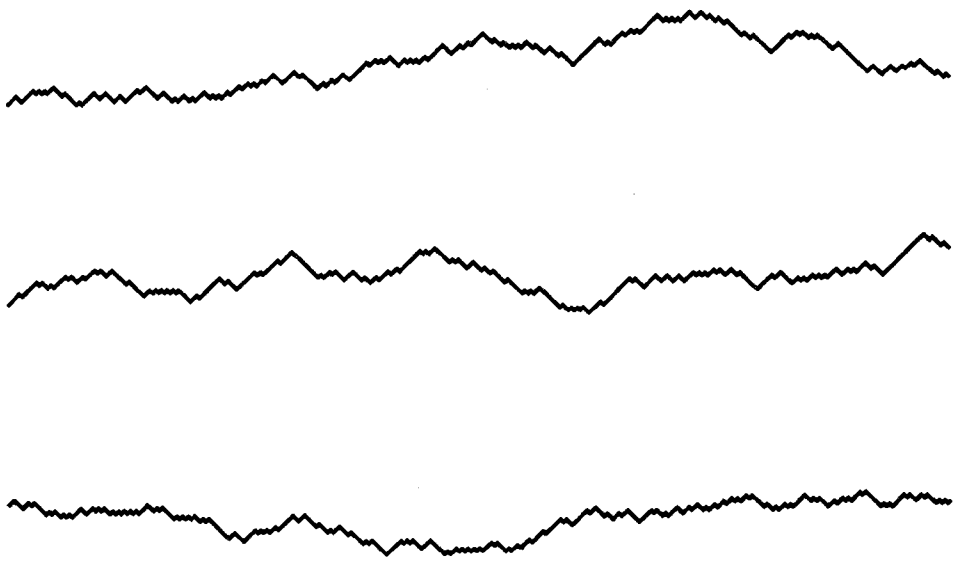


图7-2 三个随机途径

因为1和0是成对的，所以我们只需将它们中的一个翻译成z。这些GF相似于5.9节中导出的那些GF，且由定理5.9的推论，途径的平均高度是$\sim\sqrt{\pi N}$。这和5.11节中讨论过的赌徒破产的对应关系是一致的，并且这个高度等价于Catalan森林的高（见习题5.44）。

习题7.15 对所有不出现模式101101的串集合给出一个规则表达式。其对应的生成函数是什么？

习题7.16 在一个随机位串中，第二个不相交的k个0的串的平均位置是什么？

习题7.17 要导出具有规则表达式0*00*的N个0的每一个串，求不同导出方法的个数。

习题7.18 要导出具有规则表达式0*00*的N个0的每一个串，求不同导出方法的个数。

习题7.19 在一个随机位串中，求出现在第一个长度为4的每一个位模式之前的0的平均个数。

380 ~ 381

习题7.20 假设一只猴子在一个2-键键盘上随机击键。在它碰巧打出$2k$个0、1交替的一个串之前，它已击键的期望次数是多少？

7.4 有限状态自动机与Knuth-Morris-Pratt算法

串匹配的蛮力算法对许多应用来说是完全可以接受的，但是正如我们前面所看到的，该算法对高度自我重复的模式而言，运行速度很慢。解决这一问题将导致一个算法，该算法不仅具有实践价值，它也把串匹配和理论计算机科学的基本原理联系在了一起，并可由此导出更一般的算法。

例如，当查找一个k个0的串时，要克服7.1节中基本算法明显的低效问题是很容易做到的：当在文本位置i处遇到一个1时，将“模式”指针j重新置到起始位置，并在文本位置$i+1$处重新开始查找。这是利用了全-0模式的特殊性质，但这却引出了将此性质进行推广、以便对所有模式给出一个最佳算法的想法，这个想法由Knuth、Morris和Pratt[19]于1977年给出。

该想法就是要建立一个模式特定的有限状态自动机，它起始于一个初始状态，检查文本中的第一个字符；扫描文本字符；并根据扫描的值进行状态转换。当且仅当文本中的模式被找到时，一些状态才被指定为终态，自动机将在一个终态上结束。串查找的这种实现是对FSA的一个模拟，它建立在一个状态索引表的基础上。这使得串查找的实现变得极为简单，如程序7.2所示。该算法的关键问题是对转换表的计算，且转换表依赖于模式。

程序7.2 带有FSA的串查找（KMP算法）

```
procedure fsasearch;
  begin
  state := 0;
  for i := 1 to N do
    begin
    state := next(state, a[i]);
    if (state = k) then << match found >>
    end;
  end;
```

例如，表7-6的右端给出了关于模式10100110的一个相应的表，且图7-3给出了该FSA的一个图形表示。当此自动机在下面给出的样本文本段上运行的时候，它按指示取得状态转换。（下面的每个字符都给出了当其被检查时FSA所在的状态。）

382

```
01110101110001110010000011010011000001010111010001
00111234311200111201200001123456 78
```

表7-6 KMP状态转换表的例子

10100110			0 1
0	1	0	0 1
11	11	1	2 1
100	100	0	0 3
1011	1001	1	4 1
10101	10101	3	5 3
101000	100000	0	0 6
1010010	1000010	2	2 7
10100111	10000001	1	8 1
不匹配	自相关	第1次匹配	表

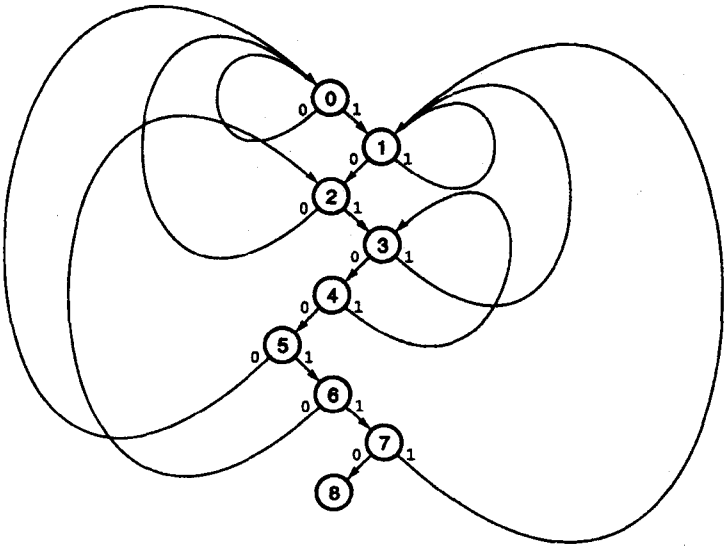


图7-3 关于10100110的Knuth-Morris-Pratt FSA

习题7.21 给出一个长度为25的文本串，它使图7-3的KMP自动机到达第二步的次数达到最大。

这是一个算法的例子，该算法十分复杂，且没有什么值得分析的价值，因为它只对文本中的每个字符检查一次。

定理7.5 (KMP串匹配) *在一个长度为N的二进制文本串中查找一个长度为k的模式时，Knuth-Morris-Pratt算法要做N次位的比较。*

证明 见上面的讨论。 ■

383～384

下一个表的构造要依赖于模式前缀的相关性，如表7-5所示。我们定义状态i对应于这样一种情形：模式中的前$i-1$个字符在文本中都已经匹配，但第i个字符不匹配。即状态i对应于文本中一个特定的i-位模式。例如对于表7-6中的模式，当且仅当文本中前4个位是1010时，FSA在状态4。如果下一位是0，我们将进入状态5；如果下一位是1，我们知道这不是一个能成功匹配的前缀，且文本中前5个位是10101。这时所要做的事情就是进入对应于第一点的状态，在这一点处当模式右移时与自身匹配，在本例中这一点是状态3。一般情况下，这一点恰好就是在这个位串的自相关中第二个1（或第二个0，如果第一个1是唯一的1时）所在的位置（从右边起按位来统计）。

习题7.22 给出关于模式110111011101的KMP状态转换表。

习题7.23 当使用KMP法确定文本011011100011101101111011001101110111011 01中是否包含上一题中的模式时，给出所做的状态转换。

习题7.24 描述关于具有$2k$个0、1交替的一个串的状态转换表。

我们一直在考虑的“串-查找”问题等价于确定文本串是不是包含在由（有二义性的）规则表达式

```
( 0 + 1)*<模式>(0 + 1)*
```

所描述的语言中。这是关于规则表达式的识别问题：给定一个规则表达式和一个文本串，确

定该串是否包含在由该规则表达式所描述的语言中。一般地，规则表达式的识别可以通过建立一个FSA来进行，且这种自动机的性质可以利用我们一直在使用的代数技巧来分析。尽管一般情况下我使通常是这么做的，然而，更专门的问题可以用更专门的技巧来有效地解决。KMP有限状态自动机就是使用这种技巧的一个最好的例子。

对较大的符号系统进行一般化推广要涉及对一个表的转换问题，尽管人们对如何改进这个表做过各种各样的研究，但这个表的大小仍然与模式的大小和字母表大小的乘积成正比。有关这种议题的细节以及文本查找的应用或许能在Gonnet和Baeza-Yates[12]中找到。

习题7.25 对$M = 4$，给出关于模式313131的KMP状态转换表。当利用KMP法确定是否文本1232032313230313131中包含该模式时，给出所做的状态转换。

习题7.26 直接证明：由一个决定性FSA所识别的语言有一个有理的OGF。

习题7.27 写一个计算机代数程序，计算OGF的标准有理型，其中OGF用来对由一个给定的决定性FSA所识别的语言进行计数。

7.5 上下文无关语法

规则表达式允许我们用一种形式方法简洁地描述语言的性质，实际上这是经得起分析检验的。语言层次中的下一个层次就是*上下文无关*的语言。例如，我们可能希望知道：

- 有多少个长度为$2N$的位串具有N个0和N个1？
- 给定一个长度为N的随机位串，平均而言，在它的前缀中有多少个前缀具有相同个数的0和1？
- 平均而言，在什么时刻，一个随机位串中0的个数首次超过1的个数？

所有这些问题都能用*上下文无关语法*来解决，它比规则表达式更富表达性。尽管上述第一个问题在组合学中是一个平凡的问题，但从语言理论中我们知道这样的集合是不能用规则表达式来描述的，必需使用上下文无关语言。正如我们将要看到的，我们也可以建立一些包含生成函数的自动机制，用来对无二义性的上下文无关语言进行计数，从而打开研究大量有趣问题的大门。

首先，我们简要概括一下形式语言理论中的一些基本定义。一个上下文无关语法是通过用并和连接乘积把*非终止符号*和字母（也叫*终止符号*）联系起来的一组*产生式*。其基本操作与规则表达式的操作类似（不需要“星”操作），然而由于非线性递归的可能性，非终止符号的引入建立了更强大的描述机制。如果一个语言可以用一个上下文无关语法来描述，那么该语言就是上下文无关的。事实上，我们一直都在使用与上下文无关语法等价的机制来定义某些我们一直分析着的组合结构。

385

这一事实的最重要的一个例子也许就是第5章中二叉树的定义，这个定义可以形式地改写成下面简单的无二义性语法：

```
<bin tree>:=<ext node>|<int node><bin tree><bin tree>
<int node>:=0
<ext node>:=1
```

非终止符号是用尖括号括起来的。我们可以把每个非终止符号都看成是对一个上下文无关语言的描述，这种描述是通过直接对非终止符号赋值一个字母表中的字符，或者是通过并或连接乘积运算而定义的。换句话说，我们也可以认为每个非终止符号都是一个*重写规则*，该规则指明了一个非终止符号是如何用表示交替重写的竖杠和表示连接的并置来进行重写的。按照第5章所介绍的一对一的对应关系，这个语法将产生相应于二叉树的位串：以先序访问树的

结点，对内部结点写0，对外部结点写1。现在，回忆对二叉树计数的生成函数，它满足

$$G(z) = z + G(z)G(z)$$

该方程——定义Catalna OGF的方程——实际上与二叉树的形式定义是完全一样的，只不过所用的符号不同而已。

注意，变量赋值对应于要进行计数的下标。为得到上述函数方程，我们曾把<external node>翻译成z，并忽略<internal node>，其中$[z^N]G(z)$是具有N个外部结点的二叉树的棵数。或者，我们也可以把<internal node>翻译成z，并忽略<external node>，以获得其关于OGF的函数方程

$$T(z) = 1 + zT(z)T(z)$$

386

其中$[z^N]T(z)$是具有N个内部结点的二叉树的棵数，或者我们可以把两个终止符号都翻译成z以得到

$$U(z) = z + zU(z)U(z)$$

这个式子是通过结点总数来对树计数的。

CFG和OGF之间的这种关系是本质性的，且在一般情况下都成立。我们把每个非终止符号都看成是对串集合的表示，这个串集合利用语法中的重写规则能够从非终止符号导出。于是，就像规则表达式一样，我们也有了一个把无二义性的上下文无关语法翻译成关于生成函数的函数方程的一般方法：

- 定义一个对应于每个非终止符号的OGF；
- 把终止符号的出现翻译成变量；
- 把语法中的连接翻译成OGF的乘法；
- 把语法中的并翻译成OGF的加法。

当执行这个过程的时候，将产生一个关于OGF的多项式方程系统。函数$f(z)$，即多项式方程$P(z, f(z)) = 0$的根，叫做代数函数。这样我们就证明了首先由Chomsky和Schützenberger[2]观察到的一个结果。

定理7.6（关于上下文无关语法的OGF） 设<A>和<B>是无二义性上下文无关语法中的非终止符号，并假定<A>|<B>和<A><B>也是无二义性的。如果$A(z)$是对能由<A>导出的串计数的OGF，$B(z)$是对<B>计数的OGF，那么

$$A(z) + B(z)\text{是对<A>|<B>计数的OGF}$$
$$A(z)\,B(z)\text{是对<A><B>计数的OGF}$$

此外，对无二义性上下文无关语言计数的OGF都是代数函数。

证明 见上面的讨论以及下面关于Groebner基础消元法的评论。 ■

该定理用到与定理7.4一样的方法，把关于语言的基本操作和使用符号法的OGF联系了起来，但与规则表达式相比，上下文无关语法的表达能力在两个重要方面所导致的结果是不同的。首先，它允许一个更一般的递归定义类型（递归可以是非线性的），因此它所产生的OGF具有一个更一般的形式——方程组一般是非线性的。其次，二义性起到了更为重要的作用。不是所有上下文无关语言都有一个无二义性的语法（甚至连是不是二义性的问题都决定不了），所以我们只能对具有无二义性语法的语言断定OGF是代数的。相反，正如前面所提到的，我们知道对所有规则语言都存在一个无二义性的规则表达式，因此，我们能够断定OGF关于所

387

有规则语言都是合理的。

定理7.6给出了对应于一个给定的无二义性上下文无关语法的多项式方程组。我们一旦有了这个方程组，就可以通过下面的最后两步得到一个“上下文无关”计数问题的解:

- 求解这个方程组，以得到一个关于OGF的代数方程;
- 求解、展开、以及/或者建立关于系数的渐近估计。

关于OGF的求解可以通过一个消元过程来实现，该过程可以把一个多项式系统减少到关于一个变量z的与所考虑的OGF相联系的一个单一的方程。例如，在某些计算机代数系统中实现的Groebner基（Groebner basis）算法可用来达此目的（见Geddes等的[11]）。特别地，定理7.6表明：如果$L(z)$是一个无二义性上下文无关语言的OGF，则对某个二元多项式$P(z, y)$，有$P(z, L(z)) = 0$。正如在下面的例子中我们将要看到的那样，在某些情况下，那个方程的解是一个能被展开的显式形式。在其他情况下，可能需要更先进的工具来对系数建立一个渐近形式（见参考文献[9]）。

2-有序排列。6.6节中关于对2-有序排列计数的讨论对应于建立串的下列无二义性上下文无关语法（串中具有相同个数的0和1）:

```
<S>:= <U>1<S> | <D>0<S> |ε
<U>:=<U><U>1 | 0
<D>:=<D><D>0 | 1
```

该语法中的非终止符号可以解释如下：<S>对应于所有具有相同个数的0和1的位串；<U>对应于0比1恰好多一个的所有位串，且在这些位串中不存在具有相同个数的0和1的前缀；<D>对应于1比0恰好多一个的所有位串，且在这些位串中不存在具有相同个数的0和1的前缀。

388

现在，根据定理7.5，把语法中的每个产生式都翻译成关于生成函数的一个方程:

$$S(z) = zU(z)S(z) + zD(z)S(z) + 1$$
$$U(z) = z + zU^2(z)$$
$$D(z) = z + zD^2(z)$$

当然，在这种情况下，$U(z)$和$D(z)$都是常见的来自于树的计数中的生成函数，所以我们可以直接地求解，从而得到

$$U(z) = D(z) = \frac{1}{2z}\left(1 - \sqrt{1 - 4z^2}\right)$$

将其代入第一个方程得

$$S(z) = \frac{1}{\sqrt{1 - 4z^2}} \qquad \text{所以} \qquad [z^{2N}]S(z) = \binom{2N}{N}$$

这和我们所期望的结果是一致的。

Groebner基消元法。一般说来，我们可能得不到显式的解，所以，我们在此简要叙述一下Groebner基消元过程是如何系统地求解这个方程组的。首先，我们要注意$U(z) = D(z)$，因为二者均满足相同的（不可约的）方程。因此，我们所需要做的事情就是从方程组中消去U

$$P_1 \equiv S - 2zUS - 1 = 0$$
$$P_2 \equiv U - zU^2 - z = 0$$

一般的策略是利用形如$AP - BQ$的重复合并，从系统中消去高次单项式，其中A、B是单项式，P、Q是有待消元的多项式。在这个例子中，是通过做$UP_1 - 2SP_2$，交错地消去U^2，从而得到

$$P_3 = -US - U + 2zS = 0$$

接下来，通过$2zP_3 - P_1$，消去US项，因此我们得到

$$P_4 = -2Uz + 4Sz^2 - S + 1 = 0$$

389

最后，将上式与$P_1 - SP_4$联合，完全消去U，得到

$$P_5 = S^2 - 1 - 4S^2z^2 = 0$$

因此，和前面的结果一样，$S(z) = 1/\sqrt{1-4z^2}$。

我们对该例子加入了一些细节以阐明一个基本观点：定理7.5给出了一个自动的方法对无二义性上下文无关语言计数。对于能够执行必要常规计算的计算机代数系统而言，这一点具有特别重要意义。

选票问题。上述最终的那个结果是初等的，但上下文无关语言自然应该是非常一般的，所以可用相同的技巧来解决一大类问题。这类问题中最突出的一个例子也许就是*选票问题*：假定在一次选举中，候选人0得到了$N + k$票，候选人1得到了N票，那么在整个计票过程中，候选人0总是领先的概率是多少？在目前情况下，该问题可以通过对$N + k$个0和N个1的位串计数来解决，这些位串应满足这样的性质：不存在相同个数的0和1的前缀。这个数目也是穿越一个$(N + k) \times N$的格子而不触到主对角线的路径数。对$k = 0$，答案是零，因为如果二候选人都有N票，那么在计票过程中，必有一个时刻他们打成平局，哪怕仅仅是在结束的时刻。对$k = 1$，由我们前面对2-有序排列的讨论可知，此数恰好是$[z^{2N+1}]U(z)$。对$k = 3$，我们有语法

```
<B>:=<U><U><U>
<U>:=<U><U>1 | 0
```

所以答案是$[z^{2N+3}](U(z))^3$。对上述结果进行推广，将立即得出关于所有k的结果。

定理7.7（*选票问题*） *一个随机位串中0比1多k个，且不存在相同个数的0和1的前缀的概率是$k/(2N + k)$。*

证明 根据上面的讨论，其结果可由下式给出

$$\frac{[z^{2N+k}]U(z)^k}{\binom{2N+k}{N}} = \frac{k}{2N+k}$$

390

这里，系数是通过对拉格朗日反演的直接应用而到的（见3.10节）。 ■

选票问题有着丰富的历史，这要追溯到1887年。更详细的讨论以及大量的相关问题，可参见由Feller[6]和Comtet[3]所写的著作。

这种类型的结果除了与树有直接的关系之外，还经常出现在算法分析与所谓的*历史*或*操作序列*的动态算法分析和数据结构的联系中。例如，选票问题等价于确定如下事件的概率：在一个初始为空的下推栈中进行“push”和“pop”操作时，在不许对空栈实行pop操作、也不许在栈中存放k个项的意义下，一个随机操作的序列是“合法的”的概率。其他应用可能要涉及更多的操作以及对合法序列的不同定义——下面的习题中给出了一些这样的例子。这种问题通常都能用上下文无关语法来解决。此种类型的许多应用在Pratt[21]早期的文章中都曾讨论过；也可参见Knuth[17]。

习题7.28 给定一个长度为N的随机位串，平均而言，它有多少个具有相同个数的0和1的前缀？

习题7.29 在一个随机位串中，0的个数从不超过1的个数的概率是多少？

习题7.30 给定一个长度为N的随机位串，平均而言，它有多少0比1多k个的前缀？在一个随机位串中，0的个数多于1的个数从不超过k的概率是多少？

习题7.31 假定一个栈的固定容量为M。对一个初始为空的下推栈进行N个“push”和“pop”操作的一个随机序列，当栈空时从不进行pop操作，或当栈满时从不进行push操作的概率是多少？

习题7.32 [Pratt]考虑一个数据结构，它有一个“插入”操作和两个不同类型的“删除”操作。在下列意义下：数据结构在随机操作序列之前和之后是空的，且删除操作总是对非空数据结构进行的，一个长度为N的随机操作序列是合法的概率是多少？

习题7.33 回答和上题相同的问题，但是把其中的一个“删除”操作改成一个“检查”操作，检查操作要对非空的数据结构进行，但它不删除任何项。

习题7.34 假定一只猴子在一个32-键的键盘上随机地击键，键盘上有26个字母A到Z，有符号+、*、(与)、空格和句点。在猴子碰巧打出一个合法的规则表达式之前，它已击出的期望字符数是多少？假定空格可以出现在规则表达式中的任何位置，且一个合法的规则表达式必须用圆括号括起来，而且表达式的末端恰好有一个句点。

391

7.6 trie树

任意N个不同位串（长度也可以不同）的集合唯一对应一棵trie树，也即一个带标号的二叉树结构。trie树可以用来表示串集合，包括文本串中所有模式的集合，所以它们提供了在文本中进行模式查找的一个快速方法，并且也可以代替传统的关键字-查找应用中的二叉树。trie树也出现在了计算机科学的许多其他应用中。

首先，我们可以把串集合和二叉树联系起来。给定一棵二叉树，将左链标为0，右链标为1，用从根到每个外部结点的路径上链的标号来确定每个外部结点。这就给出了从二叉树到位串集合的一个映射。例如，图7-4中左边的那棵树可以映射到位串集合

```
000 001 01 10 11000 11001 11010 11011 11100 11101 1111
```

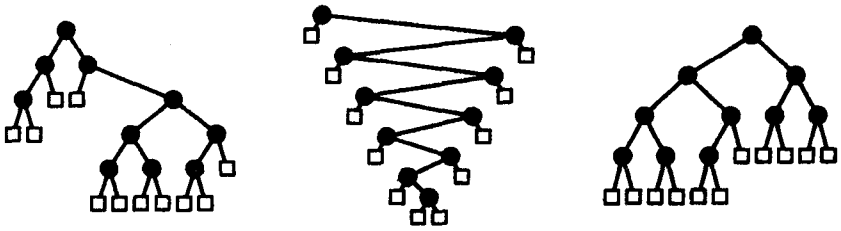
中间的那棵树映射到

```
0 100 10100 1010100 101010100 1010101010 1010101011 10101011 101011 1011 11
```

右边的那棵树映射到

```
0000 0001 0010 0011 0100 0101 011 100 101 110 111
```

用这种方法得到的所有位串的集合在结构上具有前缀无关的性质：任何一个串都不是另外的串的前缀。



392

图7-4 三棵trie树，每棵都表示11个位串

反之，给定一个位串集合，其相应的二叉树结构也可以通过把该集合按照串的前导位进行划分而递归地建立起来，正如下面的正式定义一样。假设没有一个位串是其他位串（严格）的前缀。这种假设是合理的，例如，当分析trie树的性质时，如果串无限长的话，这种假设也带来一定的方便。

定义 给定一个前缀-无关的位串集合$\mathcal{B}$，其相关的trie树是按如下方法递归定义的一棵二叉树。如果$\mathcal{B}$是空集，则其trie树为空且由一个空外部结点来表示。如果$|\mathcal{B}| = 1$，则trie树由对应于位串的一个外部结点组成。否则，定义$\mathcal{B}_0$（$\mathcal{B}_1$，分别地）是通过取$\mathcal{B}$的起始于0（1，分别地）的所有成员，并去掉每个成员中的初始位而得到的位串的集合。于是，关于$\mathcal{B}$的trie树是一个内部结点，其左方连接着关于$\mathcal{B}_0$的trie树，其右方连接着关于$\mathcal{B}_1$的trie树。

尽管上面的定义是trie树最常见的定义和使用方式，但通过对内部结点赋以一个相关的附加信息，就能用来处理前缀串了。

N个位串的trie树具有N个非空的外部结点，其中每一个结点对应一个位串，且该trie树可能有任意多个空的外部结点。和前面一样，把0当成“左”，1当成“右”，我们可以通过从根结点开始，沿着trie树向下查找，并根据串中的位从右向左读出的值来决定向左或是向右查找，以到达对应于位串的外部结点。一旦一个位串能够从trie树的所有其他位串中区分开来时，这个过程就在对应于该位串的外部结点处停止。

这个映射不是一对一的。许多位串集合都能映射到同一棵树结构，这里有两个原因：

- 不是所有外部结点都必须对应一个位串；
- 对位串加入更多的位并不改变结构。

我们将依次考虑这两个问题。

空外部结点对应于这样的情形：一些位串具有相同的位，以致于不能把它们从集合的其他位串中区分开来。例如，如果所有的位串都从0-位开始，那么相应于trie树的根的右子结点将不对应集合中任何一个位串。这样的结点可以在trie树中到处出现。 393

例如，图7-5给出了11个外部结点的3棵trie树，其中3、9和1分别是空的。在图中，空结点用小黑方块表示，非空结点（每一个都对应一个串）用稍大一点的空白方块表示。左边的那棵树代表串集合

```
000 001 11000 11001 11010 11011 11100 11101
```

中间的那棵树代表集合

```
1010101010 1010101011
```

右边的那棵树代表集合

```
0000 0001 0010 0011 0100 0101 100 101 110 111
```

我们可以对随机位串所需要的空外部结点的个数作出一个准确的分析。我们也可以这样来处理：使不需要的位不直接对应trie树结构中不需要的结点，而是用另外一种方式来表示。下面我们就比较详细地讨论这些事情。

在位串中加入更多的位并不改变结构这一事实是由定义中的“如果$|\mathcal{B}| = 1$”语句而引起的，因为在许多应用中，一旦位串被区分开，就停止对trie树继续分支是很方便的。对于有限位串而言，这个条件可以去掉，分支可以继续进行，直到每个位串都到达末端为止。我们把这样的trie树叫做关于位串集合的满trie树。反过来，我们把这样的位串集合称为关于trie树的

最小的位串集合。这些集合都只不过是对从根结点到每个非空外部结点的路径的编码。例如，我们对图7-4和图7-5所给出的位串集合也因而都是最小的。

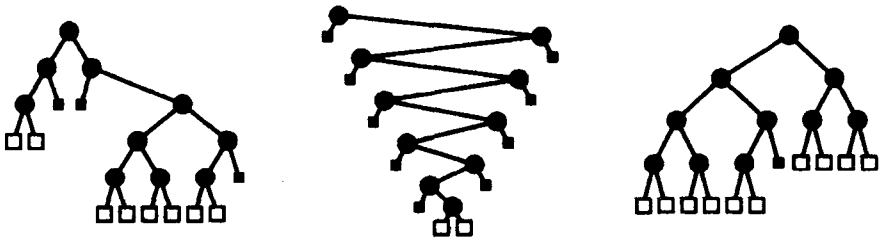


图7-5 3棵trie树，分别表示8、2、10个位串

我们所给出的递归定义导致了具有另外两个性质的二叉树结构：（i）外部结点可能是空结点；（ii）叶子的子结点必须是非空结点。就是说，我们决不会有两个空结点是兄弟结点，或者有一个空结点和一个非空结点是兄弟结点。图7-6给出了所有具有2个、3个和4个外部结点的不同trie树。

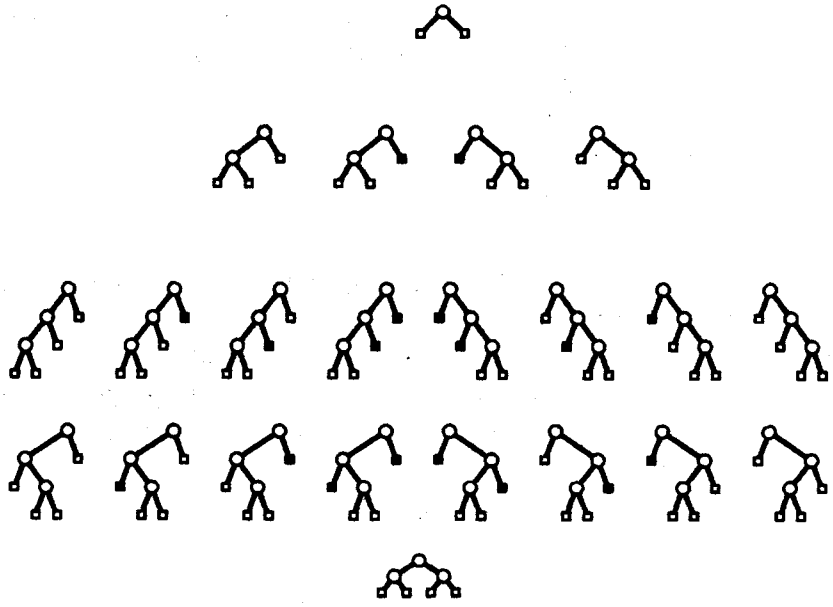


图7-6 有2个、3个和4个外部结点的trie树

图7-7给出了相应于具有5个外部结点的每种树形的最小位串集合（每个trie树都有5个外部结点，且所有5个外部结点都是非空的）。该图形也对相应于具有5个外部结点的任意trie树的最小位串集合作出了规定：在图形中，可以直接删去相应于树结构中任何与空外部结点相对应的位串（任何外部结点，如果它不是叶子的子结点，那么它就可能是空的）。

394～395

假定每种trie树结构都是等可能的，我们就能研究它们的性质（用类似于第5章中Catalan树的方法）；我们把这类问题留作了下面的习题。不过，在算法分析的场合下，我们还是把精力主要集中在位串和位串的算法上——最常应用trie树的地方——并集中精力把trie树看成是有效地识别一组串的机制、有效地表示串集合的结构，并能用它们来处理由随机串引出的概率分布。下面，我们将考虑trie树的几个算法的应用。

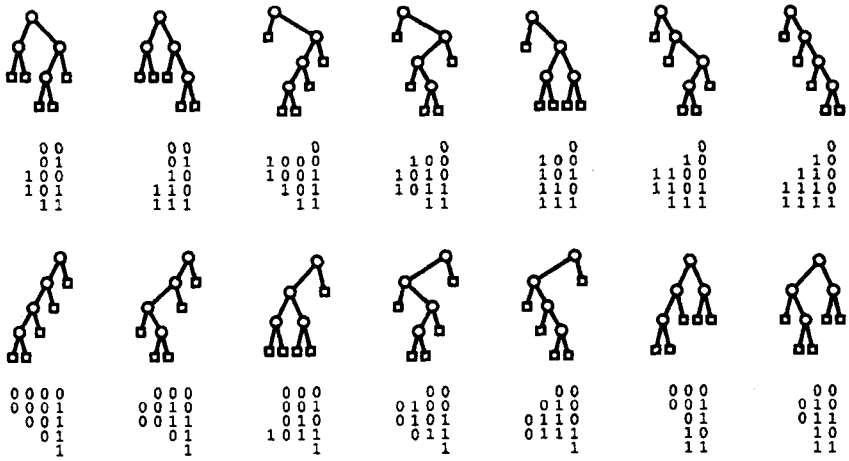


图7-7 关于5个外部结点（均为非空）trie树的位串集合

习题7.35 给出3棵trie树，使它们对应于图7-5中所给出的串集合，但每个串要按从右向左的顺序阅读。

习题7.36 5个3-位位串的不同集合共有$\binom{8}{5}=56$个，哪棵trie树与这些集合中的大多数集合相对应？哪棵trie树与这些集合中的极少数集合相对应？

习题7.37 有多少棵不同的trie树具有N个外部结点？

习题7.38 在一棵“随机”trie树中，外部结点是空结点的比例是多少（假定每个不同的trie树结构都等可能地出现）？

习题7.39 给定一个有限的串集合，设计一个简单的测试程序，确定是否有任何空外部结点在对应的trie树中。

396

7.7 trie算法

在数字计算中，二进制串几乎无处不在，而trie树结构又与二进制串的集合有着自然的对应关系，因此，有许多重要算法都要应用trie树也就毫不奇怪了。本节，我们将纵览几个这样的算法，以激发我们将要在下一节中对trie树的性质进行详细研究的兴趣。

trie树与数字查找。trie树可作为在一组二进制数据中用类似于二叉查找树的方式进行查找的算法基础，只不过是用位的比较来代替关键字的比较。

查找trie树。把位串当作关键字，我们可以把trie树当作诸如程序5.2那样的传统查找算法的基础：令x为根，b为0，然后沿着trie树向下查找，直到遇到一个外部结点时为止，将b增1，若关键字的第b位是0，则令x为x^.l，或者，若关键字的第b位是1，则令x为x^.r。如果使查找终止的外部结点是空结点，那么位串就不在trie树中；否则，就可以用相应的位串与关键字进行比较。在包含关键字的集合的适当条件下，这是一个非常有效的查找算法；细节可见Knuth[17]或Sedgewick[24]。下面的分析可用来确定trie树的性能是如何与一个给定的应用中二叉查找树的性能进行比较的。正如我们在第1章所讨论过的那样，要回答这样的问题，第一件要考虑的事情就是实现的性质。在这个特定的情形，这件事情尤其重要，因为在访问关键字的个别位时，如果处理不慎，对某些计算机而言开销可能会相当大。

习题7.40 解释如何修改一棵trie树，以把增加的一个新位串反映到它所代表的位串集合

上（对应一个二叉查找树的插入）。

Patricia trie树。我们将在下面看到，在一棵随机trie树中大约有44%的外部结点都是空的。这个比例之大可能令人难以接受。对这个问题，我们可以通过“瓦解”单向内部结点，并对每个结点都保留待检查的位的下标来加以避免。尽管对每个结点都必须附加相关的信息，但这种trie树的外部路径长度还是在某种程度上变小了。识别Patricia trie树的关键一点就是它不存在空的外部结点，或等价地说，它有$N-1$个内部结点。利用Patricia trie树来实现查找和插入有许多可用的技巧。细节问题在Knuth[17]或Sedgewick[24]中能够找到。

397

基数交换排序。正如在第1章所提到的，等长的一组位串可以通过下面的方式来进行排序：对它们进行分组，把那些所有0打头的位串放在那些所有1打头的位串前面（其所使用的过程类似于Quicksort的分组过程），然后递归地对这两部分进行排序。这种方法（称为基数排序）与trie树之间的关系和Quicksort与二叉查找树之间的关系是一样的。该排序所需时间基本上与所要检查的位数成正比。对于由随机的位组成的关键字来说，结果证明所要检查的位数和一棵随机trie树的“非空外部路径长”，即从根结点到每个非空外部结点的距离之和是一样的。

trie树编码。任何一棵带有标号的外部结点的trie树都对结点的标号定义了一个前缀码。例如，如果对图7-4中左边的那棵trie树中的外部结点从左到右用字母

C E F I L M N O P R S

进行标号，那么位串

011011000110011110011101001111111111011010

就是对词组

FILE COMPRESSION

的编码。

解码很简单：从trie树的根及位串的起始位置开始，按照位串所指示的方向（碰到0向左，碰到1向右）遍历trie树，每当遇到一个外部结点时，输出标号并从根处重新开始。如果把频繁使用的字母对路径较短的结点进行标号，那么，在这样的编码中所用的位数将大大少于标准编码中所用的位数。著名的哈夫曼编码法对给定的字符频率找到一个最佳的trie树结构（见Sedgewick[24]）。

trie树与模式匹配。trie树也可以用于文本文件中查找多个模式时的一个基本数据结构。例如，trie树在自然语言大词典的计算机化和其他类似的应用中已得到了成功的运用。正如下面我们将要描述的，根据不同的应用，trie树可以包含模式或者文本。

398

用后缀trie树进行串查找。在文本串固定（如一个词典），且有许多模式查找需要处理的应用中，查找时间可以通过对文本的预处理得到大大减少，方法如下：假设文本串是N个串的集合，每个串都从文本串中的每一个位置开始，向文本串的末端延伸（在末端k个字符处停止，其中k是要查找的最短模式的长度）。通过这组串建立一棵trie树（这样的trie树叫做关于文本串的后缀trie树）。为得知一个模式是否在文本中出现，从根开始沿着trie树向下查找，和以往一样，根据模式的位，碰到0时向左查找，碰到1时向右查找。如果到达了一个空的外部结点，则模式不在文本中；如果模式的位在一个内部结点处已用尽，那么模式就在文本中；如果到达了一个外部结点，那么把模式中剩余的位与外部结点中所表示的文本位进行比较，以确定是否存在一个匹配。

下面的分析表明，这个过程平均需要$O(\lg N)$次位的检查，比起基本算法的开销$O(N)$来说，这是一个巨大的改进，尽管建立树时有一个初始开销，但在同一个文本中查找多个模式时，

这个方法则是非常有用的。程序7.3给出了这一过程的实现。该程序调用了三个简单的过程：prefix，当其第一个参数是第二个参数的前缀时，它返回一个真值，否则返回假；firstbit，它返回参数的第一位；otherbits，它从参数中删去第一个位。在大多数计算机上，都有许多实现这些过程的非常有效的方式和方法。

程序7.3 用后缀trie树进行串查找

```
function search(v: integer, x: link): link;
  begin
  if x = NIL << no match >> else
  if prefix(v, x^.key) << match found >> else
  if firstbit(v) = 0 then search(otherbits(v), x^.left)
                     else search(otherbits(v), x^.right)
  end;
```

399

查找多个模式。trie树也可用来在文本的一趟扫描中查找多个模式，方法如下：首先，根据模式串建立一棵完全trie树。然后，对文本串中的每一个位置i，从trie树的顶部开始，沿着trie树向下查找，遇到文本中的0时向左查找，遇到文本中的1时向右查找，并在这个过程中进行字符匹配。这样的查找一定会在一个外部结点处停止。如果该外部结点是非空的，那么查找成功：trie树所表示的串之一就被找到了，它起始于文本中第i个位置。如果外部结点是空的，那么由trie树所表示的串都不会起始于i，因此文本指针可以增加到$i+1$，并从trie树的顶部重新开始上述过程。下面的分析意味着这种查找需要$O(N\lg M)$次位的检查，相对而言，若对基本算法应用M次时，其开销为$O(NM)$。

基于trie树的有限状态自动机。当刚才所述的查找过程在一个空外部结点上结束时，我们可以用和Knuth-Morris-Pratt算法完全一样的方法把事情做得更好一些，而不是回到trie树的顶部并备份文本指针。在空结点上的终止不仅告诉了我们所要查找的串不在数据库中，也告诉了我们在此不匹配之前的文本中的那些字符。那些字符恰好表明了下一次查找时我们所要检查的文本字符的位置，就像KMP算法一样，我们完全可以避免由于备份所引起的比较操作的浪费。事实上，程序7.2可以用在这种应用问题上而不必做任何修改；我们只需建立一个对应于一组串的FSA，而不是一个串的FSA。例如，自动机

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

对应于串集合000 011 1010。图7-8是该FSA的一个图形表示。例如，当此自动机在下面给出的样本文本段上运行时，它按指示取得状态转换。（下面的第二行给出了每个字符在进行检查时FSA所在的状态。）

```
1111001001001011101000001101001100000101011101000 1
02222534534534568
```

在这个例子中，FSA在状态8停止，并找到了一个模式011。Aho和Corasick[1]描述了对给定的一组模式建立一个自动机的过程。

400

以上这些算法和应用都具有代表性，它们展示了trie树数据结构在计算机应用中的重要价值。trie树的重要性不只是它们可以作为明确的数据结构，它们也能隐含地出现在基于位的算法中，或出现在要进行真正的“二元”决策的算法中。因此，描述随机trie树性质的解析结果具有非常广泛的应用。

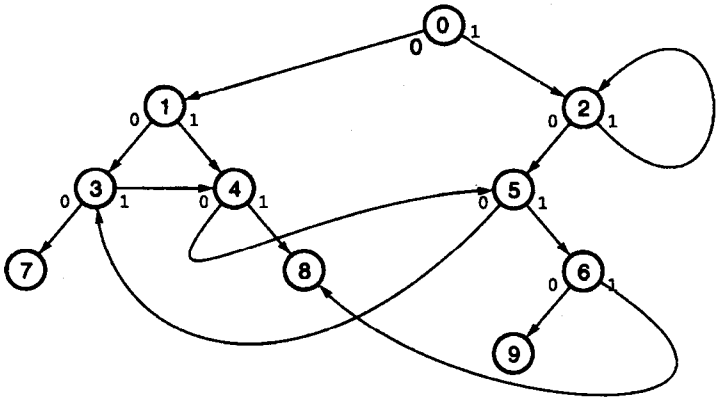


图7-8 关于000，011和1010的Aho-Corasick FSA

习题7.41 当使用图7-5中的中间那个trie树在文本串1001010011111001010100010101010010010中查找模式1010101010和1010101011中的一个模式时，所要检查的位是多少个比特？

习题7.42 给定一组模式串，描述一个方法来统计其中的一个模式在一个文本串中出现的次数。

习题7.43 根据文本串1001010011111001010100010101010010010建立一棵关于八位或更长的模式的后缀trie树。

习题7.44 给出对应于所有四位串的后缀trie树。

习题7.45 给出关于串集合01 100 1011010的Aho-Corasick FSA。

7.8 trie树的组合性质

作为组合对象，trie树只是在近期才得到了研究，尤其是与排列和树这样的经典组合对象相比而言更是如此。正如我们将要看到的，即使要完全理解trie树的最基本的性质，也需要利用我们在本书中所考虑的整套分析工具。

401

trie树的某些性质自然而然地呈现了它们有待于分析的特性。trie树中空结点的个数会是多少？平均外部路径长度是多少？或者说平均高度是多少？和二叉查找树一样，对这些基本性质的了解将会给出我们分析串查找以及应用树的其他算法时所需的必要信息。

一个需要考虑的更加基本的相关问题是，我们需要对用于分析的计算模型做出根本的改变：对二叉查找树而言，值得关注的只是关键字的相关顺序；而对于trie树而言，作为位串的关键字的二进制表示就必须发挥作用了。什么样的trie树才真正是随机的？尽管我们有几个可行的模型，但很自然的想法是，我们可以把一棵随机trie树看成是由N个随机有限位串所构成的。这样的模型对许多重要的trie树算法都是合适的，如数字查找算法等。这种模型和通过一个随机位串建立起来的后缀trie树是不一样的，尽管它确实也足以用来近似后缀trie树。

因此，我们将在假定：每个位串中的位以1/2的概率独立地取0或1下考虑trie树性质的分析。

定理7.8 (trie树的路径长和大小) 对应于N个随机位串的trie树平均外部路径长为$\sim N\lg N$。内部结点的平均个数渐近于$(1/\ln 2\pm 10^{-5})N$。

证明 我们从一个递推关系来开始证明：对$N>0$，N个位串中恰好有k个以0打头的概率是

伯努利概率 $\binom{N}{k}\Big/2^N$，所以，如果我们定义C_N是对应于N个随机位串的trie树中的平均外部路径长，则我们必有

$$C_N = N + \frac{1}{2^N}\sum_k \binom{N}{k}(C_k + C_{N-k}) \qquad (N>1,\ C_0 = C_1 = 0)$$

这恰好是我们在4.9节定理4.9中所考查过的关于基数交换排序时用于描述所检查的位数的递推关系。当时，我们曾证明

$$C_N = N![z^N]C(z) = N\sum_{j\geq 0}\left(1-\left(1-\frac{1}{2^j}\right)^{N-1}\right)$$

402

然后我们利用指数逼近导出

$$C_N \sim N\sum_{j\geq 0}(1-e^{-N/2^j}) \sim N\lg N$$

一个更准确的估计展示了该量值随着N的增加具有周期性的波动。正如我们在第4章中所见到的，和式中的项对较小的k以指数速度趋近于1，对较大的k以指数速度趋近于0，当k接近于$\lg N$时有一个过渡过程（见图4-5）。因此，我们把和式分成两部分：

$$\begin{aligned}
C_N/N &\sim \sum_{0\leq j<\lfloor\lg N\rfloor}(1-e^{-N/2^j}) + \sum_{j\geq\lfloor\lg N\rfloor}(1-e^{-N/2^j}) \\
&= \lfloor\lg N\rfloor - \sum_{0\leq j<\lfloor\lg N\rfloor} e^{-N/2^j} + \sum_{j\geq\lfloor\lg N\rfloor}(1-e^{-N/2^j}) \\
&= \lfloor\lg N\rfloor - \sum_{j<\lfloor\lg N\rfloor} e^{-N/2^j} + \sum_{j\geq\lfloor\lg N\rfloor}(1-e^{-N/2^j}) + O(e^{-N}) \\
&= \lfloor\lg N\rfloor - \sum_{j<0} e^{-N/2^{j+\lfloor\lg N\rfloor}} + \sum_{j\geq 0}(1-e^{-N/2^{j+\lfloor\lg N\rfloor}}) + O(e^{-N})
\end{aligned}$$

现在，就像我们在第2章中所做过的事情一样（见图2.3和2.4），将$\lg N$的小数部分分离出来，我们有

$$\lfloor\lg N\rfloor = \lg N - \{\lg N\} \quad 及 \quad N/2^{\lfloor\lg N\rfloor} = 2^{\lg N-\lfloor\lg N\rfloor} = 2^{\{\lg N\}}$$

这就导出了表达式

$$C_N/N \sim \lg N - \varepsilon(N)$$

其中

$$\varepsilon(N) = \{\lg N\} + \sum_{j<0} e^{-2^{\{\lg N\}-j}} - \sum_{j\geq 0}(1-e^{-2^{\{\lg N\}-j}})$$

403

现在，$\varepsilon(N)$显然是一个满足$\varepsilon(2N) = \varepsilon(N)$的周期函数，因为它只在项$\{\lg N\}$中依赖于$N$。

但这并不能立即排除$\varepsilon(N)$是常数的可能性，不过我们可以很容易检查$\varepsilon(N)$不是常数：$(C_N - N\lg N)/N$不趋向于一个极限，而是随着N的增加进行波动。图7-9给出了对$N<256$时此函数减去其平均值以后的图形，表7-7给出了$4\leq N\leq 16$时$\varepsilon(N)$的准确值（以及2的任意次幂乘以这些值）。函数$\varepsilon(N)$在数值上非常接近于1.332746；且其波动部分的振幅小于10^{-5}。

这个结果归功于Knuth[17]，他通过Mellin变换分析导出了关于$\varepsilon(N)$及平均值的显式表达式。Guibas、Ramshaw、Sedgewick[16]及Flajolet、Gourdon、Dumas[7]对此问题及其相关问题也都做过详细的研究（也可参见[26]和[9]）。

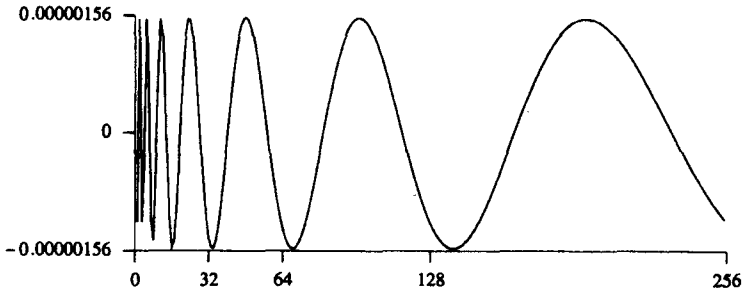


图7-9 trie树路径长中的周期性波动

表7-7 trie树路径长中的周期项

N	$\{\lg N\}$	$\Sigma_{j<0}e^{-2^{\{\lg N\}-j}}$	$\Sigma_{j\geq 0}(1-e^{-2^{\{\lg N\}-j}})$	$-\varepsilon(N)$
4	0.000 000 000	0.153 986 497	1.486 733 879	1.332 747 382
5	0.321 928 095	0.088 868 348	1.743 543 002	1.332 746 559
6	0.584 962 501	0.052 271 965	1.969 979 089	1.332 744 624
7	0.807 354 922	0.031 110 097	2.171 210 673	1.332 745 654
8	0.000 000 000	0.153 986 497	1.486 733 879	1.332 747 382
9	0.169 925 001	0.116 631 646	1.619 304 291	1.332 747 643
10	0.321 928 095	0.088 868 348	1.743 543 002	1.332 746 559
11	0.459 431 619	0.068 031 335	1.860 208 218	1.332 745 265
12	0.584 962 501	0.052 271 965	1.969 979 089	1.332 744 624
13	0.700 439 718	0.040 279 907	2.073 464 469	1.332 744 844
14	0.807 354 922	0.031 110 097	2.171 210 673	1.332 745 654
15	0.906 890 596	0.024 071 136	2.263 708 354	1.332 746 622
16	0.000 000 000	0.153 986 497	1.486 733 879	1.332 747 382
				$\gamma/\ln 2+1/2\approx 1.332746177$

我们可用类似的方法来分析N个位串的trie树中内部结点的个数问题，该数值可用下面的递归来描述：

$$A_N=1+\frac{1}{2^N}\sum_k\binom{N}{k}(A_k+A_{N-k})\qquad (N>1,\ A_0=A_1=0)$$

在此实例中，周期波动出现在主项中：实际上

$$A_N\sim\frac{N}{\ln 2}(1+\hat{\varepsilon}(N))$$

其中$\hat{\varepsilon}(N)$的绝对值小于10^{-5}。此分析的细节类似于前面的讨论，我们把它留作一道习题。 ■

因此，在一棵trie树的查找过程中，所要检查的平均位数是$\sim\lg N$，这是一个最佳值。Knuth还证明了此结果对Patricia和数字树查找也成立。我们还能得到准确的渐近值以及所涉及的常数及波动项的显式表达式。详尽细节可参考上面提到的参考文献。正如我们在证明所提到过的，此结果也适用于第1章中所讨论的Quicksort的变形——基数交换排序——的分析中。

推论 *对N个随机位串的数组排序，基数交换排序要进行$\sim N\lg N$次位的比较。*

证明 见上面的讨论。我们假定位串足够长（比方说位数远远大于$\lg N$），以至于我们可以把它们看成是“无限的”。为了与图1-3做出比较，图7-10给出了该排序的开销的分布。 ■

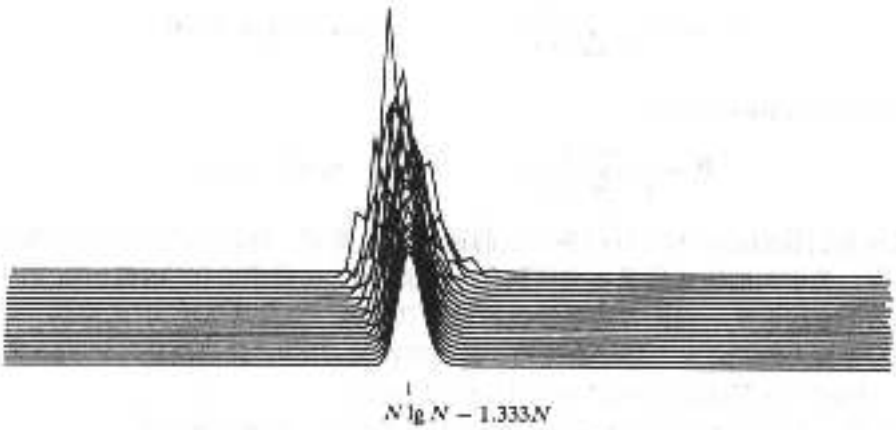


图7-10 trie树中路径长度的分布，$10 < N \leq 50$
（按比例绘制并平移到中心以对曲线进行分离）

习题7.46 证明A_N/N等于$1/\ln 2$加上一个波动项。

习题7.47 对于$N < 10^6$写一个程序计算A_N，准确到10^{-9}，并考察A_N/N的波动特性。

习题7.48 在关于$C(z)$的函数方程两端同时乘以e^{-z}，将其转换成关于$\hat{C}(z) = e^{-z}C(z)$的更简单的方程。用这个方程求出$\hat{C}_N = [z^N]\hat{C}(z)$，然后应用由$C(z) = e^z\hat{C}(z)$所隐含的卷积来证明

$$C_N = \sum_{2 \leq k \leq N} \binom{N}{k} \frac{k(-1)^k}{1 - 1/2^{k-1}}$$

习题7.49 直接证明：上题中所给出的和式等价于定理7.8的证明中所给出的关于C_N的表达式。

分布式领导人选举。 正如前面所指出的，随机trie树模型是非常普遍的。它对应一个一般的概率过程，其中“个体”（trie树查找中的记录）是通过投掷硬币的方式递归分离的。这个过程可作为各种资源分配策略的基础，尤其是在分布式场合下更是如此。作为一个例子，我们将考虑一个关于在分享一个分布式通路频道的N个个体中选出一个领导人的分布式算法。

此方法是按轮次来进行的，个体是按投掷硬币的方式被选上或被选掉的。给定一个个体集合：

- 如果集合是空的，则报告错误；
- 如果集合中只有一个个体，则宣布该个体为领导；
- 如果集合中多于一个个体，则对集合中所有成员独立地投掷0-1硬币，并对集合中得到1的个体递归地调用此过程。

404 ~ 406

如果我们从N个个体开始，那么在算法的执行过程中，我们期望N近似地减少到$N/2$，$N/4$，…。因此，我们期望这个过程大约在$\lg N$步以后结束，也许我们还需要更为准确的分析。此外，该算法也可能失败（如果每个人都投出0时，那么所有的人都将被选掉，从而选不出领导），因而了解算法成功的概率，也是值得我们关注的。

定理7.9（领导选举） 从N个候选者中选出一个领导，此随机化算法所用的平均轮次是$\lg N + O(1)$，其成功的概率渐近于$1/(2\ln 2) \pm 10^{-5} = 0.72135$。

证明 通过把算法的一次执行表示成一棵trie树，显然轮次的平均数是由N个随机位串所建立的trie树中右行分支的期望长度，因此它满足递归

$$R_N = 1 + \frac{1}{2^N}\sum_k \binom{N}{k} R_k \qquad (N > 1,\ R_0 = R_1 = 0)$$

类似地，成功的概率满足递归

$$P_N = \frac{1}{2^N}\sum_k \binom{N}{k} p_k \qquad (N > 1,\ p_0 = 0 \quad p_1 = 1)$$

这些递归与我们前面详细考虑过的路径长的递归相当类似。我们把对它们的求解留作习题。与前面一样，关于P_N的所述结果要准确到一个均值为0、振幅小于10^{-5}的波动项之内。 ■

如果这个方法失败，将此方法重复执行就可以得到一个成功概率为1的算法。平均而言，这个算法将需要大约$2\ln 2 \approx 1.3863$次迭代、最多轮次为$\sim 2\ln N$。此算法是Prodinger算法的一个简化形式，Prodinger算法平均在$\lg N + O(1)$轮后成功[22]。

习题7.50 求解定理7.9的证明中给出的关于R_N的递归，准确到波动项。

407 **习题7.51** 求解定理7.9的中给出的关于P_N的递归，准确到波动项。

习题7.52 分析重复轮次直到成功的领导人选举算法的变体形式。

7.9 更大的字母表

本章的结果是对由大小为M ($M > 2$) 的字母表的字符所构成的串所进行的推广。在我们所用过的组合技术中，没有哪个技术必须要依赖串是二进制的。在实践中，假定字母表的大小M是一个不大的常数是合理的：像26、2^8或甚至2^{16}这样的值可能是实践中所希望的。我们将不对关于更大的字母表的结果进行详细考查，但我们会给出几个一般性的评论。

在7.2节中我们曾注意到：在一个随机的M-字节串中，k个0的第一个游程在位置$M(M^k - 1)/(M - 1)$处结束。这似乎使得字母表的大小变成了一个重要的因素：例如，我们期望在一个随机位串的前几千个位内就能找到一个具有十个0的串，但是，在一个由8-位的字节组成的随机串中查找具有十个相同字符的一个串，要在前几千字节内找到将是极不可能的事情，因为第一个这样的串的期望位置大约在2^{80}字节处。然而，这只是用两种方法来看待分析结果而已，因为我们可以将一个位串用明显的方法转换成一个字节串（每次考虑8个位）。也就是说，字母表的大小并不像看起来的那么重要，因为它只通过它的对数起到干涉作用。

从实践和理论的观点来看，通常不用考虑字节串，只考虑位串就足够了，因为直接编码可以把任何一个关于字节串的有效算法转换成一个关于位串的有效算法，反之亦然。

把位作为组来考虑的一个主要优点是，我们可以把它们以下标的形式运用在表格中，或把它们当成整数来处理，这实质上是要利用计算机寻址或算术硬件来并行地匹配大量的位。它可对某些应用带来速度上的巨大提高。

多路trie树。一个M-叉trie树是由一组M-字节串所构成的，平均而言，它约有$N/\ln M$个内部结点，其外部路径长度渐近于$N\log_M N$。也就是说，使用大小为M的字母表可以节省$\ln M$倍的查找时间。但这要付出约$M/\ln M$倍存贮空间的代价，因为每个内部结点都必须有M个链，当M较大时，大部分链都是空的。

右-左串查找。考虑在一个（相对长的）文本中查找一个（相对短的）模式且二者均为字
408 节串时的问题。对前面的例子进行扩展，我们对文本中第M个字符与模式中最后一个字符进行检查。如果匹配，我们再对第（$M - 1$）个文本字符与模式中倒数第二个字符进行检查，依此类推，直到发生一个不匹配或证实已经全部匹配为止。实现这个过程的一种方法是：用与

KMP算法几乎相同的方法、基于不匹配建立一个“最佳移动”。但在这个例子中，一个更简单的方法可能会更加有效。如果一个文本字符不出现在模式中，那么这个字符就是我们可以多跳过M个字符的依据，因为在这样的字符处，位于模式右端任何位置上的每个字符都需要去匹配“当前”这个字符，而这个字符却不在模式中。对许多相关参数的合理值而言，这种事情是可能发生的，因此，这就把我们带到了这样的一个目标面前：在一个长度为N的文本串中查找一个长度为M的模式时，只检查N/M个文本字符。即使当文本字符确实出现在模式中时，我们也可以对字母表中的每一个字符，预先计算出它恰当的移动（从模式的右端到该字符的距离），以便安排模式文本中的“下一个可能的”匹配。Gonnet和Baeza-Yates[12]讨论了这种方法的细节，其思想起源于Boyer和Moore。

还有许多事情值得我们考虑。模式是应该进行位的移动还是字节移动？一个字节应该多大时才合适？一些右向左和左向右的混合型查找是可行的吗？另外一种方法是使用trie树串查找在所有第M个字节位置处检查所有可能的前缀和所有可能的后缀。许多类似的问题都是自然而然地出现的——这是一个非常活跃的研究领域，我们本章所讨论的各种分析在这个领域中将起到积极的作用。

习题7.53 一个算法要在一个文本串中查找M个0的游程（M不算太小），对于$k = 1, 2, 3$及更大的值，检查在kM处结束的t个位，如果该t位全为0，则检查该t位的两边，以确定该游程的长度，平均而言，这个算法要检查多少位？除了不假定该文本串中某个地方藏有M个0的游程之外，假定该文本串是随机的。

习题7.54 改写上一习题所给出的算法，在一个随机位串中查找最长的0串。

应该说，位串是关于计算机科学的基本组合对象，在考虑串和trie树的组合问题时，我们无疑已经接触到大量的基本范例。我们已经检查过的某些结果与概率论中的经典结果有联系，但许多其他结果既与计算机科学的基本概念有联系，也与重要的实践问题有联系。 409

我们考查了大量基本的串-查找算法。从算法设计和分析的观点看，这些算法是相当有意义的，不过它们还没有像排序和查找算法那样得到广泛的研究。有些算法，尤其是trie树查找算法，对许多更一般的情形都非常适用，串-查找算法的实际应用也是数不胜数的。

对串集合的考查导致了形式语言的研究和下面的有用结果：形式语言系统的基本理论性质与生成函数的分析性质之间存在着一种明确的关系。我们能用一种“自动的”方式分析一大类问题，尤其是在现代计算机代数系统的辅助之下。

在所有这些背景之下，trie树是极为重要的一种组合结构和一种数据结构。它是经典查找问题的一种实用的数据结构，并且它给出了排序过程的一个模型；它是固定的串集合的一种自然表示方法，并可应用于串查找问题；作为树的一种类型，它与基本的分治范例有着直接的联系；它与位的结合使它把自己与低层表示法以及低层过程直接联系在一起。

最后，trie树基本性质的分析提出了一个无法完全用初等方法来解决的重要课题。详细的分析需要对波动函数进行描述。Flajolet、Gourdon和Dumas[7]（还可见[9]）详细讨论了处理出现在此类问题中的各种函数的高级数学技巧（主要是Mellin变换）。因为许多算法都在或者是隐含地、或者是明确地处理二进制串，因而这种函数出现在大量的算法之中。

参考文献

1. A. V. Aho and M. J. Corasick. “Efficient string matching: An aid to bibliographic search,” *Communications of the ACM* **18**, 1975, 333–340.

2. N. CHOMSKY AND M. P. SCHÜTZENBERGER. "The algebraic theory of context-free languages," in *Computer Programming and Formal Languages*, P. Braffort and D. Hirschberg, eds., North Holland, 1963, 118–161.
3. L. COMTET. *Advanced Combinatorics*, Reidel, Dordrecht, 1974.
4. H. DELANGE. "Sur la fonction sommatoire de la fonction somme des chiffres," *L'enseignement Mathématique* **XXI**, 1975.
5. S. EILENBERG. *Automata, Languages, and Machines*, Volume A, Academic Press, New York, 1974.
6. W. FELLER. *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 1957.
7. P. FLAJOLET, X. GOURDON, AND P. DUMAS. "Mellin transforms and asymptotics: harmonic sums," *Theoretical Computer Science* **144**, 1995, 3–58.
8. P. FLAJOLET, M. REGNIER, AND D. SOTTEAU. "Algebraic methods for trie statistics," *Annals of Discrete Math.* **25**, 1985, 145–188.
9. P. FLAJOLET AND R. SEDGEWICK. *Analytic Combinatorics*, in preparation.
10. P. FLAJOLET AND R. SEDGEWICK. "Digital search trees revisited," *SIAM Journal on Computing* **15**, 1986, 748–767.
11. K. O. GEDDES, S. R. CZAPOR, AND G. LABAHN. *Algorithms for Computer Algebra*, Kluwer Academic Publishers, Boston, 1992.
12. G. H. GONNET AND R. BAEZA-YATES. *Handbook of Algorithms and Data Structures*, 2nd edition, Addison-Wesley, Reading, MA, 1991.
13. I. GOULDEN AND D. JACKSON. *Combinatorial Enumeration*, John Wiley, New York, 1983.
14. L. GUIBAS AND A. ODLYZKO. "Periods in strings," *Journal of Combinatorial Theory, Series A* **30**, 1981.
15. L. GUIBAS AND A. ODLYZKO. "String overlaps, pattern matching, and nontransitive games," *Journal of Combinatorial Theory, Series A* **30**, 1981, 19–42.
16. L. GUIBAS, L. RAMSHAW, AND R. SEDGEWICK. Unpublished work, 1979.
17. D. E. KNUTH. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
18. D. E. KNUTH. "The average time for carry propagation," *Indagationes Mathematicae* **40**, 1978, 238–242.
19. D. E. KNUTH, J. H. MORRIS, AND V. R. PRATT. "Fast pattern matching in strings," *SIAM Journal on Computing*, 1977, 323–350.
20. M. LOTHAIRE. *Combinatorics on Words*, Addison-Wesley, Reading, MA, 1983.
21. V. PRATT. "Counting permutations with double-ended queues, parallel stacks and parallel queues," in *Proceedings 5th Annual ACM Symposium on Theory of Computing*, 1973, 268–277.
22. H. PRODINGER. "How to select a loser," *Discrete Mathematics* **120**, 1993, 149–159.

23. A. SALOMAA AND M. SOITTOLA. *Automata-Theoretic Aspects of Formal Power Series*, Springer-Verlag, Berlin, 1978.
24. R. SEDGEWICK. *Algorithms*, 2nd edition, Addison-Wesley, Reading, MA, 1988.
25. L. TRABB-PARDO. Ph.D. thesis, Stanford University, 1977.
26. J. S. VITTER AND P. FLAJOLET, "Analysis of algorithms and data structures," in *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, 431–524.

412

第8章 字与映射

固定的字母表中的字符串或字，除了在上一章所考虑过的算法类型中的重要作用之外，它们在大量的应用中也是非常重要的。与第7章所研究的组合对象一样，本章我们将考虑相同的一组组合对象（当时我们称它们为“字节串”），但我们将从一种不同的观点去研究它们。

一个字可以被看作是把取自区间1到N的一个整数i（字符位置）映射到区间1到M的另一个整数j（字符的值，取自于M个字符的字母表）的映射。在上一章中，我们主要考虑了“局部”性质，包括与相继下标有关的值之间的关系（如相关性等）；本章，我们将考虑“全局”性质，包括全局范围内值的出现频率以及更加复杂的结构性质。在第7章中，我们一般认为字母表的大小M是一个较小的常数，且串的大小N是较大的数（甚至是无限大的）；在本章中，我们将考虑相应于这些参数的值的其他各种可能性。

作为基本组合对象，字经常出现在算法分析中。其中特别值得关注的算法是散列算法——一个重要的且被广泛用于信息检索的算法系列。我们将利用第3章中基本生成函数的计数技巧和第4章中的渐近结果来分析各种散列的变形。散列算法的使用率相当高，它们也有一个较长的历史。在这个历史中，算法分析扮演了中心角色。反过来，准确预测这类重要的实用算法性能的能力，一直都在刺激着算法分析技术的发展。事实上，Knuth曾说过，散列算法分析对他早期的系列著作的构架有着“强大的影响”[20][21][22]。

人们一直都在频繁地研究着字的基本组合性质，主要是因为这些性质可以作为独立Bernoulli试验序列的模型。这种分析包括二项分布的性质，其中的许多性质我们在第3和第4章中都详细地考查过。我们将要考虑的一些问题叫做占有问题，因为它们能用N个球随机分布在M个瓮中的模型来刻划，瓮中球的“占有分布”是我们将要研究的问题。这种经典问题中的许多问题都是初等的，然而，简单算法往往能够导出分析起来相当困难的算法的变形。 413

如果字母表的大小M较小，那么二项分布的正态逼近对占有问题的研究是比较合适的；如果M随着N增大，那么我们可以使用泊松逼近。在散列算法分析以及算法分析的其他应用中，这两种情况都将出现。

最后，我们引入映射（Map）的概念：一个函数映射把一个有限域映射到有限域自身。这是在算法分析中经常出现的另外一个重要的组合对象。映射与字之间的关系是这样一种关系：我们可以把映射看成是一个N-字母字母表中的N-字母字，但事实上映射与树、森林以及排列之间也有联系；对它们性质的分析揭示了一个有趣的组合结构，这个结构是对我们所研究过的几个结构的一般化推广。符号法在帮助我们研究映射的基本性质方面是一个有效的工具。最后我们将以对整数进行因子分解的一个算法作为随机映射原理的应用来结束本章内容。

8.1 使用分离链接的散列

程序8.1给出了一个关于信息检索的标准算法，该算法通过N个关键字的一个表把查找时间减少到原来的$1/M$。通过令M足够大，该算法通常能够胜过我们在5.5节和7.7节中所考查过的基于树和trie树的查找算法。

利用一个所谓的散列函数，我们可将每个关键字转换成1到M之间的一个整数，并假定对任何一个关键字，每一个整数值都等可能地通过散列函数来产生。当两个关键字散列到相同的值时（这种情况叫做冲突），我们就把它们存入单独的数据结构中。为得知一个给定的关键字是否在表中，我们用散列函数按相同的散列值在关键字中进行二次搜索。也许关于二次搜索的最简单的数据结构——我们在程序8.1中所使用的——就是一个无序的链表，这是一种非常容易建立和搜索的数据结构。该算法的实现使用了一个标记结点z，其初值为所要查找的值。一次不成功的查找就是找到z的查找。

散列算法的性能取决于散列函数的有效性，散列函数把任意的关键字等可能地转换成1到M之间的值。做这件事的一个典型的方法是：选用一个质数M，然后用一种自然的方法将关键字转换成大数，再用转换的数除以M取余数作为散列值。人们还设计出了一些更加复杂的方法。散列算法在一大类应用中得到了广泛的使用，经验证明，如果对散列过程加以注意，则不难保证散列函数将关键字转换成看起来是随机的散列值。这样，算法分析对于性能的预测就特别有效，因为一个直接的随机性假设总被认为是合理的，且这种假设已经溶入算法之中。

414

散列算法的性能也取决于二次搜索时所使用的数据结构。例如，考虑使用一个“有序的链表”，并用它按递增次序来存放元素。或者，考虑使用一个二叉查找树来存放具有相同散列值的关键字。当然，算法分析能够帮助我们在这些变化中进行选择。

如果N相对M而言较大、且散列函数产生的值又是随机的，那么我们可以期望每个链表中约有N/M个元素，因而搜索时间可以减少到原来的$1/M$。尽管我们不能使M任意的大，因为增大M意味着要使用额外的空间来维护二级数据结构，但我们也许能使$M = O(N)$，在这种情况下的搜索时间是常数。人们已经提出了能够达到这一标准的许多散列的变体，见程序8.1。

程序8.1 分离链接散列法

```
procedure insert(v: integer);
  var x: link;
  begin
  new(x); x^.next = table[hash(v)];
  table[hash(v)] = x;
  end;
function search(v: integer): link;
  var t: link;
  begin
  t := table[hash(v)]; z^.key = v;
  while t^.next^.key <> v do t := t^.next;
  if t = z then t = NIL;
  search := t;
  end;
```

415

在散列算法分析中，我们所关心的事情不仅是确定如何对各种散列算法进行比较，也要确定如何将散列算法与其他有关查找问题的算法进行比较，以及如何最好地设置参数（如散列表的长度）。为把注意力集中在数学运算上，我们将采用衡量散列算法性能的习惯方法：对成功与不成功的查找，统计所用的探测次数，或算出关键字与数据结构中元素的比较次数。和以往一样，更详细的分析包括：计算散列函数时的开销、访问和比较关键字时的开销等等，这些分析都需要对算法的相关性能做出决定性的说明。比如，尽管散列可能只包含常数次探测，但如果关键字很长，它也可能要比基于（比方说）tire树的搜索要慢，因为计算散列函数时要涉及检查整个关键字，而基于tire树的方法可能在检查比较少的位后就能确定出关键字。

长度为M的表中的N个散列值的序列只不过是由大小为M的字母表中的字母所组成的长度为N的一个字。因此，散列分析直接对应于字的组合性质的分析。接下来，我们将考虑字的基

本组合性质，包括分离链接法的散列分析，然后，我们再考查另外几个散列算法。

8.2 字的基本性质

定义 长度为N的M-字是一个函数f，它把整数从区间$[1...N]$映射到区间$[1...M]$。

与排列和树一样，我们通过写出字的函数表来说明它:

下标	1	2	3	4	5	6	7	8	9
字	1	8	2	8	4	5	9	0	4

我们不用下标，而是通过书写$f(1)f(2)\cdots f(N)$来说明函数，这种方法使得如下结果更为明显：一个长度为N的M-字等价于一个长度为N的字节串（字节大小为M）。当研究字的时候，我们通常把注意力集中在字中值的集合上面（例如，有多少个1？）。当我们在第7章研究字节串的时候，我们曾把注意力集中在字节串中整数序列的性质上（有多少个连续的1？）。 416

在离散概率中，这些组合对象经常用球-瓮模型的背景来研究。我们想像把N个球随机地分布在M个瓮中，则我们要寻问的问题是关于分布的结果。这与字是直接对应的：N个球对应于一个字中的N个字母，M个瓮对应于一个字母表中M个不同的字母，所以我们要指明每个球到底落入哪个瓮中。“字”和“球-瓮”这两个术语可以交换使用，并且我们将增加一个术语“散列表”，即，我们刚才所描述过的基本数据结构。图8-1给出了一个典型的球数相对较多、瓮数相对较少的球-瓮分布。

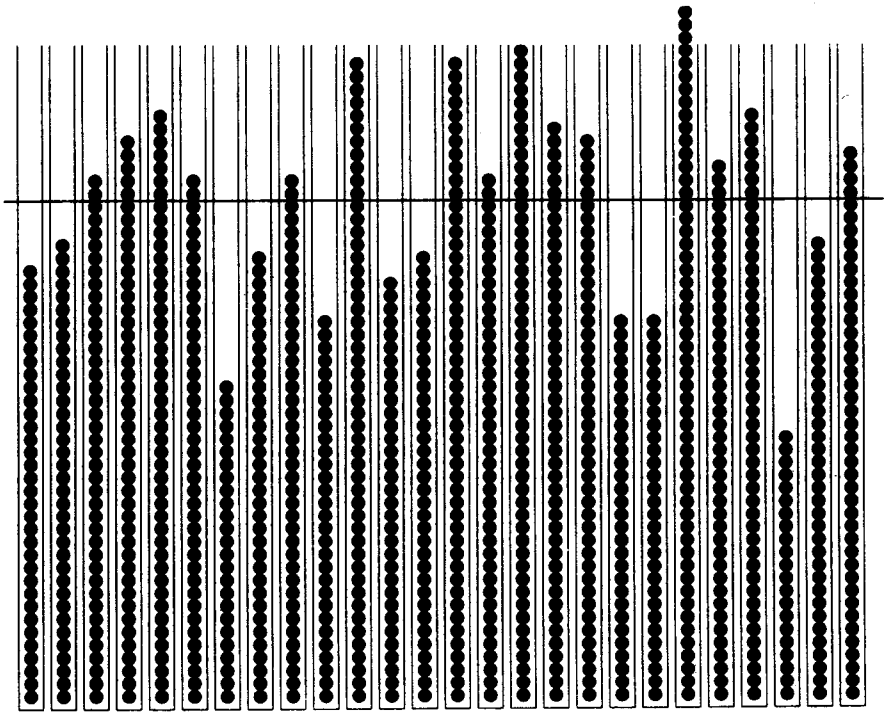


图8-1 将1024个球放入26个瓮中的随机分布

M和N的相对增长率问题是分析中的中心问题，实践中还会出现许多不同的情形。对于文本串或其他类型的“字”来说，我们通常把M看成是固定的，把N看成是我们所关心的某种变量。我们从一个固定的字母表的字母中产生不同长度的字，或者我们把不同个数的球扔入固 417

定个数的瓮中。对于其他应用，尤其是散列算法，我们则认为M是随N的增大而增大的，一般来说$M = \alpha N$，其中α在0和1之间。图8-2给出了$\alpha = 1$时的8个例子：很多瓮趋于空，某些瓮中只有一个球，只有很少的瓮中有多个球。正如我们将要看到的，我们的分析将导出图8-1和图8-2中所阐明的“占有分布”的准确特征。我们目前所讨论的这个问题的分布当然是二项分布，且我们所考查的结果也是经典的结果（几乎有两个世纪）；我们在第4章已经较详细地讨论过它们。

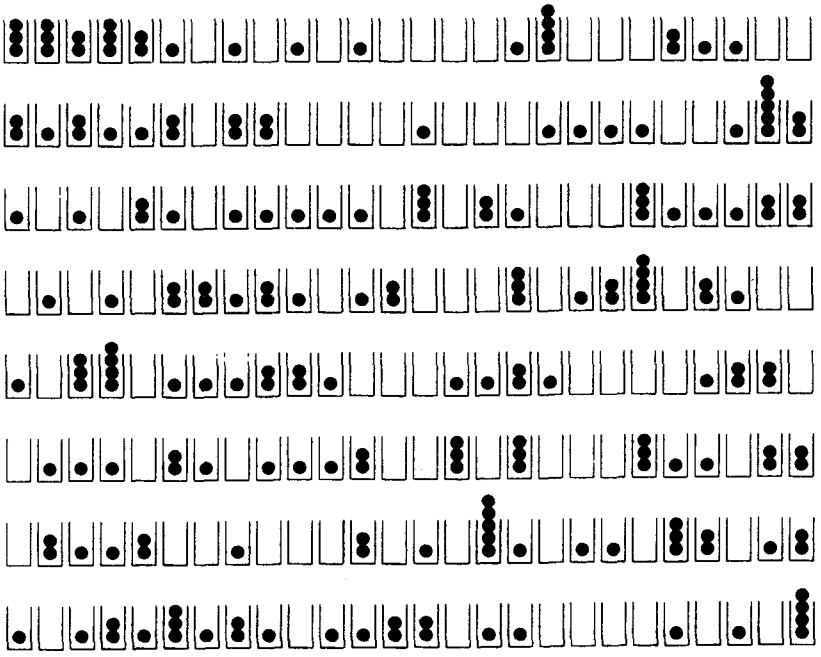


图8-2 26个球在26个瓮中的8个随机分布

表8-1给出了长度为4的所有的3-字（或把4个球分布到3个瓮中的所有方式），以及表示用过0、1、2、3和4次的字母数的占有总数（或含有0、1、2、3和4个球的瓮数）。该表也包含了我们所要考查的另外几个类型的统计量。

418

表8-1 占有分布及长度为4的3-字的性质，或4个球在3个瓮中的配置，或长度为4的散列序列（表长为3）

字	0	1	2	3	4	字	0	1	2	3	4	字	0	1	2	3	4
1111	2	0	0	0	1	2111	1	1	0	1	0	3111	1	1	0	1	0
1112	1	1	0	1	0	2112	1	0	2	0	0	3112	0	2	1	0	0
1113	1	1	0	1	0	2113	0	2	1	0	0	3113	1	0	2	0	0
1121	1	1	0	1	0	2121	1	0	2	0	0	3121	0	2	1	0	0
1122	1	0	2	0	0	2122	1	1	0	1	0	3122	0	2	1	0	0
1123	0	2	1	0	0	2123	0	2	1	0	0	3123	0	2	1	0	0
1131	1	1	0	1	0	2131	0	2	1	0	0	3131	1	0	2	0	0
1132	0	2	1	0	0	2132	0	2	1	0	0	3132	0	2	1	0	0
1133	1	0	2	0	0	2133	0	2	1	0	0	3133	1	1	0	1	0
1211	1	1	0	1	0	2211	1	0	2	0	0	3211	0	2	1	0	0

（续）

字	0	1	2	3	4	字	0	1	2	3	4	字	0	1	2	3	4
1212	1	0	2	0	0	2212	1	1	0	1	0	3212	0	2	1	0	0
1213	0	2	1	0	0	2213	0	2	1	0	0	3213	0	2	1	0	0
1221	1	0	2	0	0	2221	1	1	0	1	0	3221	0	2	1	0	0
1222	1	1	0	1	0	2222	2	0	0	0	1	3222	1	1	0	1	0
1223	0	2	1	0	0	2223	1	1	0	1	0	3223	1	0	2	0	0
1231	0	2	1	0	0	2231	0	2	1	0	0	3231	0	2	1	0	0
1232	0	2	1	0	0	2232	1	1	0	1	0	3232	1	0	2	0	0
1233	0	2	1	0	0	2233	1	0	2	0	0	3233	1	1	0	1	0
1311	1	1	0	1	0	2311	0	2	1	0	0	3311	1	0	2	0	0
1312	0	2	1	0	0	2312	0	2	1	0	0	3312	0	2	1	0	0
1313	1	0	2	0	0	2313	0	2	1	0	0	3313	1	1	0	1	0
1321	0	2	1	0	0	2321	0	2	1	0	0	3321	0	2	1	0	0
1322	0	2	1	0	0	2322	1	1	0	1	0	3322	1	0	2	0	0
1323	0	2	1	0	0	2323	1	0	2	0	0	3323	1	1	0	1	0
1331	1	0	2	0	0	2331	0	2	1	0	0	3331	1	1	0	1	0
1332	0	2	1	0	0	2332	1	0	2	0	0	3332	1	1	0	1	0
1333	1	1	0	1	0	2333	1	1	0	1	0	3333	2	0	0	0	1
												总共	48	96	72	24	3

3 × +24 × +18 × +36 ×

Pr{没有空瓮}	36/81 ≈ 0.444
Pr{占有小于3的瓮}	(18+36)/81 ≈ 0.667
Pr{占有小于4的瓮}	(24 + 18 + 36)/81 ≈ 0.963
空瓮平均数	(1.42 + 2.3)/81 ≈ 0.593
平均最大占有	(2.54+3.24+4.3)/81 ≈ 2.370
平均占有	(1.96+2.72+3.24+4.3)/3.81 ≈ 1.333
平均最小占有	(1.36)/81 ≈ 0.444

一般而言，当N个球随机地分布在M个瓮中时，我们将关注下列一类问题：

- 没有瓮是空瓮的概率是多少？
- 没有瓮中的球数多于1的概率是多少？
- 有多少个瓮是空的？
- 含有球数最多的瓮中的球是多少个？
- 含有球数最少的瓮中的球是多少个？

这些问题将立即关切到散列算法以及其他算法的实际实现问题：我们想知道，当使用带有分离链的散列时，我们所期望的链表长度是多少，我们期望有多少个空链表，等等。其中的一些问题是计数问题，它们类似于第6章中带有圈长约束的排列的计数问题，其他问题则需要我们更详细地分析字的性质。这些问题及其相关问题都依赖于球在瓮中的占有分布，我们下面将要对此进行详细的研究。

表8-2对3个瓮、球数的变化范围为1到10给出了上述几个问题中各个量的值，它们是利用下一节给出的结果计算出来的。其中第四列对应于表8-1，它表明这些值是如何计算出来的。随着球数的增加，我们将得到一种类似于图8-1所描绘的情形，即：球大约均匀地分布在各个

瓮中，每个瓮中约有N/M个球。该表也展示了我们在直觉上所期望的其他现象。例如，随着球数的增加，空瓮数将越来越少，且不存在空瓮的概率将越来越大。

表8-2 3个瓮中球的占有参数

球	1	2	3	4	5	6	7	8	9	10
概率										
占有小于2的瓮	1	0.667	0.222	0	0	0	0	0	0	0
占有小于3的瓮	1	1	0.889	0.667	0.370	0.123	0	0	0	0
占有小于4的瓮	1	1	1	0.963	0.864	0.700	0.480	0.256	0.085	0
占有大于0的瓮	0	0	0.222	0.444	0.617	0.741	0.826	0.883	0.922	0.948
占有大于1的瓮	0	0	0	0	0	0.123	0.288	0.448	0.585	0.693
占有大于2的瓮	0	0	0	0	0	0	0	0	0.085	0.213
平均										
空瓮数	2	1.33	0.889	0.593	0.395	0.263	0.176	0.117	0.0780	0.0520
最大占有	1	1.33	1.89	2.37	2.78	3.23	3.68	4.08	4.50	4.93
最小占有	0	0	0.222	0.444	0.617	0.864	1.11	1.33	1.59	1.85

419 ~ 420

M和N的相应值对上面提出的各种问题的答案起着多大的支配作用是值得关注的。如果球比瓮数多得多（$N>>M$），那么显然空瓮数将非常少。事实上，我们期望每个瓮中有N/M个球。这就是图8-1的情形。如果球比瓮数少得多（$N<<M$），那么大多数瓮都将是空的。一些最有趣也最重要的结果描述了当N和M相差一个常数因子时的情形。正如图8-2所表明的，即使当$M=N$时，瓮的占有也相当低。

表8-3给出了关于表8-2中对应的值，不过是对较大的值$M=8$时的对应值，和表8-2一样，N的变化范围是1到10。当球数较少时，我们有类似于图8-2所描绘的情形，即：有许多空瓮，且总体上说每个瓮中有较少的球。此外，根据下一节的分析结果，我们也能计算出准确的值。

表8-3 8个瓮中球的占有参数

球	1	2	3	4	5	6	7	8	9	10
概率										
占有小于2的瓮	1	0.875	0.656	0.410	0.205	0.077	0.019	0.002	0	0
占有小于3的瓮	1	1	0.984	0.943	0.872	0.769	0.642	0.501	0.361	0.237
占有小于4的瓮	1	1	1	0.998	0.991	0.976	0.950	0.910	0.855	0.784
占有大于0的瓮	0	0	0	0	0	0	0	0.002	0.011	0.028
占有大于1的瓮	0	0	0	0	0	0	0	0.000	0.000	0.000
平均										
空瓮数	7	6.13	5.36	4.69	4.10	3.59	3.14	2.75	2.41	2.10
最大占有	1	1.13	1.36	1.65	1.93	2.18	2.39	2.60	2.81	3.02
最小占有	0	0	0	0	0	0	0	0.002	0.011	0.028

与排列一样，我们可以在字之间建立组合对应关系，用来推导产生有关分析结果的CGF之间的函数关系。许多这样的对应都有潜在的用途，从中选出一个用于特定的应用是分析中的一门艺术。字比排列有更多的可能性，因为字有两个参数：关于长度为N的M-字之间的对应就好比是具有较小的N值和M值的字。

421

习题8.1 像表8-1一样，给出一个关于3个球在4个瓮中的表。

习题8.2 像表8-1与表8-3一样，给出一个关于2个瓮的表。

习题8.3 给出关于N和M的一个充分必要条件，使平均空瓮数等于平均最小的瓮占有。

“第一”对应或“最后”对应。给定一个长度为N的M-字，考虑通过简单地去掉第一个元素后而形成的那个字，它的长度为$N-1$。它恰好对应于M个长度为N的字，每个字的第一个元素具有每种可能的值。或进一步说，给定任何一个长度为$N-1$的M-字，我们都能通过对每个1到M的k，在其第一个位置前插入k，从而简单地确定出M个不同的M-字。例如，

1 2 1 3 3　2 2 1 3 3　3 2 1 3 3

都与2 1 3 3 对应。这就定义了“第一”对应；显然我们可以对任何其他元素做同样的事情，而不仅仅是第一个元素。这个对应说明，长度为N的M-字的个数是长度为$N-1$的M-字个数的M倍，重申一遍这个明显的事实就是：总数为M^N。

“最大”对应。给定一个长度为N的M-字，考虑通过简单地去掉所有的M的出现所形成的$(M-1)$-字。如果存在k个这样的出现（k可以从0变化到N），那么这个$(M-1)$-字的长度就是$N-k$，它恰好对应于长度为N的$\binom{N}{k}$个不同的字，其中的每一个字都对应于把k个元素加入到这个$(M-1)$-字中去的所有可能方式中一种。例如，

3 3 2 1　3 2 3 1　3 2 1 3　2 3 3 1　2 3 1 3　2 1 3 3

都对应2 1。这个对应导致下面的递推关系

$$M^N = \sum_{0 \le k \le N} \binom{N}{k} (M-1)^{N-k}$$

这是对二项式定理的重述。

8.3 生日悖论与赠券收藏家问题

我们知道球在瓮中的分布是二项分布，我们将在本章稍后详细讨论这个分布的性质。在这之前，我们先考虑关于球-瓮配置的两个典型问题，这两个问题都要涉及瓮中放入球时的动态过程。当N个球一个接一个地随机分布到M个瓮中时，我们想知道，平均而言，在 422

- 第一次有一个球落入一个非空的瓮中；
- 第一次不存在空瓮；

之前，有多少个球已经落入了瓮里。这两个问题分别叫做生日问题和赠券收藏家问题。和以前一样，这两个问题立即关切到散列和其他算法的实际实现的问题。例如，对生日问题的解答将告诉我们：在我们发现第一次冲突之前，我们期望已插入了多少个关键字；而对赠券收藏家问题的解答将告诉我们：在我们发现已没有空链表之前，我们期望已插入了多少个关键字。

生日问题。在此领域中，最著名的问题或许就是生日问题，该问题常常被描述如下：当有多少人聚在一起的时候，两个人有相同的生日比没有两个人有相同的生日更有可能？每次取一个人，第二个人与第一个人生日不同的概率是$(1-1/M)$；第三个人与前两人生日均不同的概率是$(1-2/M)$，依此类推，所以N个人的生日均不相同的概率是

$$\left(1-\frac{1}{M}\right)\left(1-\frac{2}{M}\right)\cdots\left(1-\frac{N-1}{M}\right) = \frac{N!}{M^N}\binom{M}{N}$$

图8-3给出了$M=365$时该分布的曲线图。

定理8.1（生日问题） 当N个球投入M个瓮中时，没有冲突的概率由下式给出

$$\left(1-\frac{1}{M}\right)\left(1-\frac{2}{M}\right)\cdots\left(1-\frac{N-1}{M}\right)$$

在第一个冲突发生之前，已投入的球的期望个数是

$$1+Q(M)=\sum_k\binom{M}{k}\frac{k!}{M^k}\sim\sqrt{\frac{\pi M}{2}}+\frac{2}{3}$$

其中$Q(M)$是Ramanujan Q-函数。

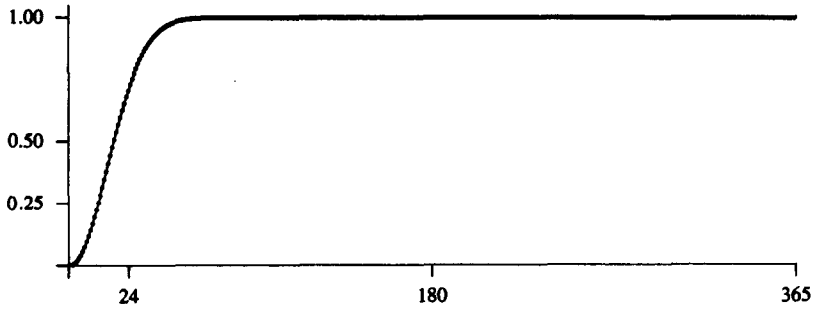


图8-3 两人具有相同生日的概率

证明 见前面关于概率分布的讨论。为求得期望值，令X表示第一次冲突出现之前球数的随机变量。于是，上面给出的概率恰好就是$\Pr(X>N)$。将它们相加，我们就得到了关于期望的表达式

$$\sum_{N\geq 0}\left(1-\frac{1}{M}\right)\left(1-\frac{2}{M}\right)\cdots\left(1-\frac{N-1}{M}\right)$$

由$Q(M)$的定义（见4.7节），该表达式恰好就是$1+Q(M)$。其渐近形式可根据定理4.8得到。 ■

$1+Q(365)$的值在24与25之间——这是我们要找到两个相同生日的期望人数。此问题也常常被称作“生日悖论”，因为人们所期望的人数可能要大的多。

找出中位数的值也是值得关注的：即关于上面给出的概率最接近1/2时N的值。这可以通过一个快速的计算机运算来求出，也可以使用渐近计算。用渐近计算，截止点由下式确定

$$\left(1-\frac{1}{M}\right)\left(1-\frac{2}{M}\right)\cdots\left(1-\frac{N-1}{M}\right)\sim\frac{1}{2}$$

$$\sum_{1\leq k\leq N}\ln\left(1-\frac{k}{M}\right)\sim\ln(1/2)$$

$$\sum_{1\leq k\leq N}\frac{k}{M}\sim\ln 2$$

$$\frac{N(N-1)}{2M}\sim\ln 2$$

$$N\sim\sqrt{2M\ln 2}$$

423~424

此值略低于答案：当$M=365$时，$N=23$。

在发生生日冲突之前，所聚集人数的平均值中$\sqrt{M}$的系数是$\sqrt{\pi/2}\approx 1.2533$，而关于中位数（要聚集的人数，该人数有50%的把握保证使一个生日冲突发生）中$\sqrt{M}$的系数是$\sqrt{2\ln 2}\approx 1.1774$。这些值对于$M=365$分别导出了近似值24.6112和22.4944，有趣的是我们注意到，在

这个例子中平均值和中位数并不渐近地相等。

习题8.4 对$M=365$，需要多少人时才能有99%的把握保证有两个人具有相同的生日？

习题8.5 估计定理8.1中关于生日分布的方差，并解释平均值和中位数的渐近值之间明显不一致的现象。

赠券收藏家问题。该领域中另一个著名的问题是经典的赠券收藏家问题：如果每个产品盒中都有一组M个赠券中的一个，平均而言，必须购买多少盒产品之后，才能得到所有的赠券？这个问题等价于在所有瓮中都至少有一个球之前，已经投入的球数的期望值是多少；或等价于在由定理8.1建立的散列表的所有链上都至少有一个关键字之前，已存入的关键字个数的期望值是多少。

为解决这个问题，我们先定义一个k-收藏，它是一个字，它由k个不同的字母组成，字中最后一个字母是该字母的唯一一次出现，且$\mathcal{W}_{Mk}$是M个可能赠券的k-收藏的集合。长度为N的k-收藏的个数除以M^N等于要得到k个不同的赠券所需收集的N个赠券数的概率。在球-瓮模型中，这是最后一个球落入一个空瓮的概率，使得非空的瓮数等于k。现在，该OGF

$$P_k(z)=\sum_{w\in\mathcal{W}_{Mk}} z^{|w|}$$

满足

$$P_k(z)=(k-1)\sum_{w\in\mathcal{W}_{Mk}} z^{|w|+1}+(M-(k-1))\sum_{w\in\mathcal{W}_{M(k-1)}} z^{|w|+1}$$

由“第一”对应：要么第一个赠券在w中的（$k-1$）个赠券（不包括最后那个）的集合之中，在这种情况下该字的其余部分是一个k-收藏，要么第一个赠券不在w中的（$k-1$）个赠券的集合之中，在这种情况下该字的其余部分是一个$(k-1)$-收藏。因此， 425

$$\begin{aligned}P_k(z)&=(k-1)zP_k(z)+(M-(k-1))zP_{k-1}(z)\\&=\frac{(M-(k-1))z}{1-(k-1)z}P_{k-1}(z)\end{aligned}$$

其中$P_0(z)=1$。注意$P_k(z/M)$是关于k-收藏的长度的PGF，所以我们有$P_k(1/M)=1$，且平均长度等于$P'_k(z/M)|_{z=1}$。对方程

$$P_k(z/M)=\frac{(M-(k-1))z}{M-(k-1)z}P_{k-1}(z/M)$$

的两边微分，在$z=1$处求解并化简，可得到下面的递推关系

$$\frac{\mathrm{d}}{\mathrm{d}z}P_k(z/M)\Big|_{z=1}=1+\frac{(k-1)}{M-(k-1)}+\frac{\mathrm{d}}{\mathrm{d}z}P_{k-1}(z/M)\Big|_{z=1}$$

套叠求解得

$$\frac{\mathrm{d}}{\mathrm{d}z}P_k(z/M)\Big|_{z=1}=\sum_{0\le j<k}\frac{M}{M-j}=M(H_M-H_{M-k})$$

定理8.2（赠券收藏家问题） *在所有M个瓮都非空之前，所投入的球的平均个数是*

$$MH_M=M\ln M+M\gamma+O(1)$$

其方差为

$$M^2H_M^{(2)}-MH_M\sim M^2\pi^2/6$$

第N个球落入最后一个空瓮的概率是

$$\frac{M!}{M^N}\left\{{N-1 \atop M-1}\right\}$$

证明 平均值已由上面的讨论导出过了。或者，通过套入关于k-收藏的OGF的递推关系，即可得出显式形式

$$P_k(z)=\frac{M(M-1)\cdots(M-k+1)}{(1-z)(1-2z)\cdots(1-(k-1)z)}z^k$$

426

从而可立即导出其PGF为

$$P_M(z/M)=\frac{M!z^M}{M(M-z)(M-2z)\cdots(M-(M-1)z)}$$

容易验证：$P_M(1/M)=1$，关于z微分并代入$z=1$处的值就可得到和前面所给出的同样的MH_M。再微分一次，并应用定理3.10，就可导出所要证明的方差的表达式。根据3.13节中的表3-7，利用等式

$$\sum_{N\geq M}\left\{{N \atop M}\right\}z^N=\frac{z^M}{(1-z)(1-2z)\cdots(1-Mz)}$$

通过在PGF中提取系数，将立即得出该问题的分布。 ■

习题8.6 在表8-1中找出所有的2-收藏和3-收藏，然后计算$P_2(z)$和$P_3(z)$，并检查z^4的系数。

第二类Stirling数。 正如在3.13节中所指出过的那样，Stirling“子集”的个数$\left\{{N \atop M}\right\}$也代表将一个$N$-元素集合划分成$M$个非空子集的方式数。我们将在本节稍后部分看到该事实的一个推导。从这个定义出发，可以导出赠券收藏家分布的另外一个推导方法，该方法如下：考虑$N-1$个球的集合和$M-1$个瓮。由于瓮的次序无关紧要，所以$N-1$个球落入$M-1$个不同的瓮中的方式数为$\left\{{N-1 \atop M-1}\right\}(M-1)!$。任何一个瓮都可以是最后那个装入球的瓮，所以$N$个球中的最后一个球落入$M$个瓮中最后那个瓮的方式数为$M\left\{{N-1 \atop M-1}\right\}(M-1)!$，将其除以$M^N$就可得出和定理8.2中所给出的完全一样的结果。

定理8.2中关于平均值的经典推导（比如说，参见Feller[9]）是这样进行的：一旦收集了k个赠券后，则需要j个或更多个额外的盒子来得到下一个赠券的概率是

$$\sum_{j\geq 0}\left(\frac{k}{M}\right)^j=\frac{1}{1-k/M}=\frac{M}{M-k}$$

对k求和就可得出和前面一样的结果MH_M。此推导所需的计算量比上面的推导所需的计算量要少，但生成函数能抓住问题的整体结构，使得计算方差时不用明显地担心所涉及的随机变量之间的相关性问题，并且还能给出完整的概率分布。

427

习题8.7 通过部分分式展开PGF，证明第N个球落入最后空瓮中的概率也可由交错和式$\sum_{0\leq j<M}\binom{M}{j}(-1)^j\left(1-\frac{j}{M}\right)^{N-1}$来表示。

习题8.8 给出一个至少收集N个盒子而得到M个赠券的一个完全收藏的概率表达式。

满射的直接计数。 不存在空瓮的球-瓮序列（每个字母至少出现一次的字）叫做满射

(surjections)。满射在许多场合下都存在，因为它们对应于把N个物品恰好分成M个不同的非空组。这里我们考虑一个方法来对它们进行计数，该方法类似于我们在第5章和第6章见过的几个论证。每个长度为$|f|$的M-满射都对应于长度为$|f|+1$的M个不同的M-满射，这M个不同的满射是通过向原来的满射中追加一个1到M间的整数而得到的；每个长度为$|f|$的$(M-1)$-满射都对应于长度为$|f|+1$的M个不同的M-满射，这M个不同的M-满射是通过在原来满射中的元素间M个可能的位置中的任一位置上插入M而得到的。每一个M-满射都是通过这种方法生成的（如果有一个单独的M，将其去掉就得到对应的（$M-1$）满射；否则去掉最后那个元素）。因此，其EGF

$$F_M(z)=\sum_{\substack{f\in\mathcal{F}_M\\ f\text{ 满射}}}\frac{z^{|f|}}{|f|!}$$

满足

$$F_M(z)=\sum_{\substack{f\in\mathcal{F}_M\\ f\text{ 满射}}}M\frac{z^{|f|+1}}{(|f|+1)!}+\sum_{\substack{f\in\mathcal{F}_{M-1}\\ f\text{满射}}}M\frac{z^{|f|+1}}{(|f|+1)!}$$

将其微分并化简后得

$$F'_M(z)=MF_M(z)+MF_{M-1}(z)$$

其解为

$$F_M(z)=(e^z-1)^M$$

428

这是一个关于第二类Stirling数的指数生成函数（见表3-7）。因此，我们就证明了长度为N的M-满射有$M!\left\{{N \atop M}\right\}$个。此外，从组合的定义出发，第二类Stirling数是对N个元素划分成M个子集的计数，这也能得到同一结果的一个直接证明，因为这些集合的$M!$个次序中的每一个都产生一个满射。

习题8.9 考虑满射间的“最大”对应：给定一个长度为N的M-满射，考虑通过去掉M的所有出现而形成的$(M-1)$-满射。求使用这种对应关系的满射的EGF。

习题8.10 写一个程序，打印出所有长度为N的M-满射，只要这样的满射个数小于1000时就要打印。

习题8.11 用二项式定理展开$(e^z-1)^M$，证明

$$N![z^N]F_M(z)=\sum_j\binom{M}{j}(-1)^{M-j}j^N=M!\left\{{N \atop M}\right\}$$

（见习题8.7）

习题8.12 证明：把N个元素分成非空子集的方式数为$N![z^N]e^{e^z-1}$（该结果定义了一个所谓的Bell数）。

习题8.13 证明：

$$N![z^N]e^{e^z-1}=\frac{1}{e}\sum_{k\geq 0}\frac{k^N}{k!}$$

习题8.14 证明关于第二类Stirling数的二元EGF为$\exp(u(e^z-1))$。

习题8.15 应用“最大”对应求长度为N的M-字的个数时可以导致递归

$$F_{NM} = \sum_{0<k<M} \binom{N}{k} F_{(N-k)(M-1)}$$

429 说明如何利用BGF来求解这个递归。

高速缓存算法。赠券收藏家的结果是经典的，它们在大量有用的算法分析中都有着直接的意义。例如，考虑一个“请求分页”系统，其中k-页高速缓存通过保存k个最新引用过的页来增强M-页内存的性能。如果“页引用”是随机的，那么赠券收藏家分析将会给出在高速缓存被填满之前的引用次数。

推论 *在一个M-页的内存系统中，一个大小为k的高速缓存被填满之前，页引用的平均次数是$M(H_M - H_{M-k})$。假定页引用是独立的和均匀分布的。*

证明 该结果可由定理8.2的证明中所给出的$P_k(z)$的计算直接得出。 ■

注意，尽管在所有的页都到达之前，高速缓存器已取得了~$M\ln M$次引用，但大小为αM的高速缓存器在大约$M\ln(1/(1-\alpha))$次引用后才被填满。例如，一个大小为$M/2$的高速缓存器在大约进行了$M\ln 2 \approx 0.69M$个页引用后将被填满，结果节省了19%的空间。在实际中，引用并不是随机的，而是相关的和不均匀的。例如，最近引用的页很可能被再次引用，结果导致高速缓存更高的节省量。因此，对实际情况的分析应给出关于高速缓存效率一些更低的界。此外，这样的分析也提供了在非均匀概率模型下有关更“现实的”分析的一个起点（见Flajolet、Gardy和Thimonier[10]）。

生日问题和赠券收藏家问题出现在用球装填容器过程中不同的两端。在生日问题中，我们加入球并观察第一次得到超过一个球的容器，平均而言，这要花大约$\sqrt{\pi M/2}$步。继续加入球，平均约$M\ln M$步后，我们最终将每个容器至少放入了一个球，达到了收藏家问题的结果。在这两个结果之间，当$M = N$时，我们将在下一节中看到大约有$1 - 1/e \approx 36\%$的容器是空的，而且其中的一个容器中应大约有$\ln N/\ln\ln N$个球。

这些结果对散列算法有着各种实际的应用。首先，生日问题隐含着的冲突往往会在早期
430 出现，所以必须设计解决冲突的策略。其次，填入过程往往很不均衡，结果是有相当多的空链表和几个长链表（几乎是对数长度）。第三，直到相当多次的插入之后，空链表才可能完全消失。

Feller[9]以及Flajolet、Gardy和Thimonier[10]就生日问题、赠券收藏家问题及其相关的应用问题给出了更多的细节。

8.4 占有约束与极值参数

解决生日问题要涉及对排列的个数计数的问题（没有一个字母出现两次的字数），解决赠券收藏家问题要涉及对满射的个数计数的问题（每个字母至少出现一次的字数）。本节，我们将描述这两个关于字的计数问题的一般化推广。

在第6章，我们曾讨论过带有圈长约束的排列计数问题；在第7章中，我们又讨论了带有连续位模式约束的位串计数问题。本节，我们将利用类似的技巧讨论带有字母出现频率约束的字的计数问题，或等价地说，带有占有约束的球-瓮配置问题，或带有冲突频率约束的散列序列，或带有变化范围频率约束的函数。

表8-4给出了3-字的情形。表中上方四行给出了任何一个字母的出现不多于1、2、3和4次的3-字数，下方四行给出了每个字母至少出现1、2、3和4次的3-字数（因此第五行对应于满射）。所给出的数是频数统计，因此，把每一项都除以M^N就可得到表8-2。第四列对应于表8-1。

这些数之间有着许多联系，利用非常类似于第6章中处理排列的方式，我们可以很容易地通过对符号法的系统应用，对这些联系予以揭示。

表8-4 带有字母频率约束的3-字的统计或带有占有限制的3个瓮中球的配置或带有冲突约束的散列序列（表大小为3）

频率	球 →	1	2	3	4	5	6	7	8	9	10	11	12
<2		3	6	6									
<3		3	9	24	54	90	90						
<4		3	9	27	78	210	510	1050	1680	1680			
<5		3	9	27	81	240	690	1890	4830	11130	22050	34650	34650
>0				6	36	150	540	1806	5796	18150	55980	171006	519156
>1							90	630	2940	11508	40950	125100	445896
>2										1680	12600	62370	256410
>3													34650

最大占有。由一个给定字母的最多k次出现所构成的字的EGF是$1+z+z^2/2!+\cdots+z^k/k!$。所以

$$(1+z+z^2/2!+\cdots+z^k/k!)^M$$

是M个不同字母中的每一个字母最多k次出现所构成的字的EGF。这是对标号对象符号法的一个直接应用（见定理3.8）。去掉关于k的约束就可得到EGF

$$(1+z+z^2/2!+\cdots+z^k/k!+\cdots)^M=e^{zM}$$

所以长度为N的M-字的总数就是$N![z^N]e^{zM}=M^N$，该结果和我们所期望的结果是一样的。

取$k=1$，得EGF为

$$(1+z)^M$$

它表明了每个字母（都不相同的字母）最多出现一次的字的个数是

$$N![z^N](1+z)^M=M(M-1)(M-2)\cdots(M-N+1)=N!\binom{M}{N}$$

这个量也叫做从M个元素中选出N个元素的所有可能的排列数，或有序的组合数。将排列数除以M^N就能得到定理8.1中关于生日问题的概率分布，因而符号法的使用提供了一个直接的一般化方法。

定理8.3（最大占有） *每个字母最多有k次出现的长度为N的字的个数是*

$$N![z^N]\left(1+\frac{z}{1!}+\frac{z^2}{2!}+\cdots+\frac{z^k}{k!}\right)^M$$

特别地，排列数（$k=1$）是$N!\binom{M}{N}$。

证明 见上面的讨论。 ■

431 ~ 432

表8-4中上半部分的数对应于$1\leqslant k\leqslant 4$时计算这些EGF的系数。例如，第二行对应于展开式

$$\left(1+z+\frac{z^2}{2!}\right)^3=1+3z+\frac{9}{2}z^2+4z^3+\frac{9}{4}z^4+\frac{3}{4}z^5+\frac{1}{8}z^6$$
$$=1+3z+9\frac{z^2}{2!}+24\frac{z^3}{3!}+54\frac{z^4}{4!}+90\frac{z^5}{5!}+90\frac{z^6}{6!}$$

习题8.16 求所有字母频数为偶数的M-字的EGF。

习题8.17 证明：对所有的$k\geqslant 0$，把$M(k+1)$个球分布到M个瓮中、所有瓮中的球数都大于k的方式数等于把$M(k+1)-1$个球分布到M个瓮中、所有瓮中的球数都小于$(k+2)$的方式数（见表8-4）。对这个量给出一个阶乘的商的形式的显式公式。

习题8.18 把球抛入N个瓮中，在第二个冲突发生以前，所抛入的期望球数是多少？这里假定“冲突”指的是事件“把球扔入一个非空的瓮中。”

习题8.19 把球抛入N个瓮中，在第二个冲突发生以前，所抛入的期望球数是多少？这里我们假定“冲突”指的是事件“把球抛入一个恰好只有一个球的瓮中”。

习题8.20 对不含相同字母三次出现的M-字的个数给出一个显式表达式。

习题8.21 像图8-3那样，对三人同生日的概率做出一个曲线图。

习题8.22 对$M=365$，需要有多少人时才能保证有50%的把握使三人具有相同的生日？四人又如何？

最小占有。通过与上面关于最大占有类似的论证可以得出，每个字母出现多于k次的字数的EGF为

$$(e^z-1-z-z^2/2!-\cdots-z^k/k!)^M$$

特别地，当$k=1$时，将给出每个字母至少出现一次的字数的EGF（没有空瓮的球-瓮序列数）：

$$(e^z-1)^M$$

433 这构成计算满射数的另一种直接的方法，我们将考虑对前一节赠券收藏家问题的推广。

定理8.4（最小占有） *每个字母最少出现k次的长度为N的字的数目是*

$$N![z^N]\left(e^z-1-\frac{z}{1!}-\frac{z^2}{2!}-\cdots-\frac{z^{k-1}}{(k-1)!}\right)^M$$

特别地，长度为N的M-满射的个数是

$$N![z^N](e^z-1)^M=M!\left\{{N \atop M}\right\}$$

证明 见上面的讨论。 ■

表8-4中下半部分的数对应于这些EGF的计算系数。例如，倒数第三行对应于展开式

$$(e^z-1-z)^3=\left(\frac{z^2}{2}+\frac{z^3}{6}+\frac{z^4}{24}+\frac{z^5}{120}+\cdots\right)^3$$
$$=\frac{1}{8}z^6+\frac{1}{8}z^7+\frac{7}{96}z^8+\frac{137}{4320}z^9+\frac{13}{1152}z^{10}+\cdots$$
$$=90\frac{z^6}{6!}+630\frac{z^7}{7!}+2940\frac{z^8}{8!}+11508\frac{z^9}{9!}+40958\frac{z^{10}}{10!}+\cdots$$

至于最大占有，生成函数简明地描述了这些值的计算。

表8-5根据定理8.3和8.4，对带有字母频数约束的字的计数问题的生成函数进行了总结。这两个定理与关于带有圈长约束的排列的定理6.2和6.3是相对应的。这4个定理，包括排列、满射、对合、错位排列的分析以及它们的推广，都是值得我们回顾的，因为它们对大量的组合结构和经典的计数问题给出了一个统一的描述方式。

描述定理8.3和8.4中函数的渐近值的特征，要涉及对多元渐近问题的讨论，其中每一个函数都根据所考虑的变化范围有不同的渐近体系。这也可以看成是我们对二项分布处理方法的一个推广（见第4章以及下面的讨论），其中，不同的近似用于参数值不同的变化范围。例如，对固定的M，当$N\to\infty$ 时，系数$[z^N](1+z+z^2/2)^M$最终将变为0。同时，对固定的M，系数之和为$(5/2)^M$，并在$5M/4$附近有一个峰，且在峰附近可以得到一个关于系数的正态逼近。因此，当M和N成比例时，存在着有趣的区域。类似地，当M固定时，我们可以考虑当$N\to\infty$ 时$[z^N](e^z-1-z)^M$的情况。这里，我们是在统计每个值至少假定两次的函数的个数。但从概率的角度看，这些函数中除了极少一部分外，绝大部分将对所有的值都至少假定两次（实际上大约N/M次）。因此，对固定的M，这个系数渐近于$[z^N](e^z)^M$。此外，当N增加并变到$O(M)$时，将存在一个有趣的转变。这样的渐近结果可用鞍点法得到很好的量化，正如Kolchim[23]所给出的详细讨论一样（也可参见[12]）。 434

表8-5 带有字母频数约束的字的EGF，或带有占有约束的球-瓮配置，或带有冲突频数约束的散列序列

1个瓮，占有k	$z^k/k!$
所有字	e^{zM}
所有占有>1（满射）	$(e^z-1)^M$
所有占有>k	$(e^z-1-z-z^2/2!\cdots-z^k/k!)^M$
无占有>1（排列）	$(1+z)^M$
无占有>k	$(1+z+z^2/2!+\cdots+z^k/k!)^M$

习题8.23 找出每个瓮中被抛入至少两个球之前已抛入的球的平均数。

期望最大占有。当N个球随机地分布在M个瓮中时，球在瓮中的平均最多个数是多少？这是一个极值参数，类似于我们已经遇到过的另外几个极值参数。正如我们对树高、排列中的最大圈长以及其他参数所做过的一样，我们可以利用生成函数来计算最大占有。

根据定理8.4，我们可以写出关于球-瓮配置中至少有一个瓮的占有大于k的生成函数，或等价地说，那些最大占有大于k的生成函数 435

$$
\begin{aligned}
e^{3z}-(1)^3 &= 3z+9\frac{z^2}{2!}+27\frac{z^3}{3!}+81\frac{z^4}{4!}+243\frac{z^5}{5!}+\cdots\\
e^{3z}-(1+z)^3 &= 3\frac{z^2}{2!}+21\frac{z^3}{3!}+81\frac{z^4}{4!}+243\frac{z^5}{5!}+\cdots\\
e^{3z}-\left(1+z+\frac{z^2}{2!}\right)^3 &= 6\frac{z^3}{3!}+27\frac{z^4}{4!}+153\frac{z^5}{5!}+\cdots\\
e^{3z}-\left(1+z+\frac{z^2}{2!}+\frac{z^3}{3!}\right)^3 &= 3\frac{z^4}{4!}+33\frac{z^5}{5!}+\cdots\\
e^{3z}-\left(1+z+\frac{z^2}{2!}+\frac{z^3}{3!}+\frac{z^4}{4!}\right)^3 &= 3\frac{z^5}{5!}+\cdots\\
&\vdots
\end{aligned}
$$

等等。将它们相加，我们就可得到当球分布在3个瓮中时，（累积）最大占有的指数CGF为

$$=3z+12\frac{z^2}{2!}+54\frac{z^3}{3!}+192\frac{z^4}{4!}+675\frac{z^5}{5!}+\cdots$$

将其除以3^N，即可得出表8-2中所给出的平均值。一般来说，平均最大占有可由下式给出：

$$\frac{N!}{M^N}[z^N]\sum_{k>0}\left(e^{Mz}-\left(\sum_{0<j<k}\frac{z^j}{j!}\right)^M\right)$$

Gonnet[14]证明了当N和M以$N/M=\alpha$（α为常数，上述结果中的主项与α无关）的方式随着N，$M\to\infty$时，上述结果为$\sim\ln N/\ln\ln N$。因此，比如说，当使用程序8.1时，平均而言，最长的链表的长度将是$\sim\ln N/\ln\ln N$。

习题8.24 当N个球分布到M个瓮中时，导出关于期望最小占有的指数CGF的表达式。对M和N小于20时的值列出表格。

习题8.25 在一个随机的字中，连续相同元素的块的平均个数是多少？

436 **习题8.26** 分析字中的“上升”和“游程”（参见6.1节）。

8.5 占有分布

一个M-字中恰好包含一个给定值的k个实例的概率是

$$\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k}$$

这个式子可以通过一个直接的计算建立起来：$\binom{N}{k}$（选取位置）乘以$(1/M)^k$（那些字母具有给定值的概率）乘以$(1-1/M)^k$（其他字母不具有给定值的概率）。我们在第4章曾详细研究过这个分布，也就是我们熟知的二项分布或伯努利分布，在本书中，我们已经在多种不同的场合下遇到过这个分布了。例如，表4-6给出了当$M=2$时的值。另一个例子在表8-6中给了出来，它给出了当$M=3$时对应的值；该表中第4行对应于表8-1。

表8-6 关于M = 3的占有分布$\binom{N}{k}(1/3)^k(2/3)^{N-k}$：

Pr{N个球分布到3个瓮中之后，一个瓮中有k个球}

$N\downarrow$ $k\rightarrow$	0	1	2	3	4	5	6
1	0.666 667	0.333 333					
2	0.444 444	0.444 444	0.111 111				
3	0.296 296	0.444 444	0.222 222	0.037 037			
4	0.197 531	0.395 062	0.296 296	0.098 765	0.012 346		
5	0.131 687	0.329 218	0.329 218	0.164 609	0.041 152	0.004 115	
6	0.087 791	0.263 375	0.329 218	0.219 479	0.082 305	0.016 461	0.001 372

涉及两个独立变量（球数和瓮数）的复杂情况以及对分布不同部分的关注，意味着我们必须对特定的应用小心准确地刻划出分布的特性。球-瓮模型所隐含的直觉知识对我们实现这一目的常常很有帮助。

本节，我们将考查多种参数值分布的准确公式和渐近估计。表8-7给出了几个样本值。例如，当100个球分布在100个瓮中时，我们期望大约18个瓮中有2个球，但任何一个翁中有多达

10个球的可能性是可以忽略的。另一方面，当100个球分布到10个瓮中时，1个或2个瓮中很可能有10个球（其他瓮中很可能有7到13个球），但很少可能有2个球的瓮。正如我们在第3章中所见到的，这些结果可以用正态逼近和泊松逼近来描述，就我们所关注的变化范围很大的值而言，这两种方法对于刻划这种分布都是准确的和有用的。 437

上述分布的推导很简单，但是用一种稍微不同的方式——累积计数的方式——来推导还是值得的。对于N个球和M个瓮来说，一个瓮中有k个球的球-瓮序列（长为N的那些M-字）的总数可由下式给定:

$$C_{Nk}^{[M]} = \sum_{u\text{瓮}} \{u\text{出现}k\text{次的长度为}N\text{的}M\text{-字的个数}\}$$
$$= \sum_{u\text{瓮}} \binom{N}{k}(M-1)^{N-k} = M\binom{N}{k}(M-1)^{N-k}$$

这就导出了BGF，和以往一样，我们可以利用BGF来计算矩。上式除以M^N就导出了分布的经典公式，我们在这里将重述这个公式，并根据第4章归纳出一些渐近结果。

表8-7 占有分布的例子

瓮 M	球 N	占有 k	有k个球的平均瓮数 $M\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k}$
2	2	2	0.500 000 000
2	10	2	0.087 890 625
2	10	10	0.001 953 125
10	2	2	0.100 000 000
10	10	2	1.937 102 445
10	10	10	0.000 000 001
10	100	2	0.016 231 966
10	100	10	1.318 653 468
100	2	2	0.010 000 000
100	10	2	0.415 235 112
100	10	10	0.000 000 000
100	100	2	18.486 481 882
100	100	10	0.000 007 006

438

定理8.5（占有分布） 当N个球随机地分布到M个瓮中时，具有k个球的平均瓮数是

$$M\binom{N}{k}\left(\frac{1}{M}\right)^k\left(1-\frac{1}{M}\right)^{N-k}$$

当M固定且$k = N/M + x\sqrt{N/M - N/M^2}$时（其中$x = O(1)$），上式为

$$M\frac{e^{-x^2}}{\sqrt{2\pi}} + O\left(\frac{1}{\sqrt{N}}\right) \quad \text{（正态逼近）}$$

当$N/M = \alpha > 0$固定、且$k = O(1)$时，上式为

$$M\frac{\alpha^k e^{-\alpha}}{k!} + o(M) \quad \text{（泊松逼近）}$$

证明 见前面的讨论。上面所述的二项分布逼近是由第4章得到的（习题4.66和定理4.7）。■

推论 当$N/M=\alpha$（常数）时，平均空瓮数渐近于$Me^{-\alpha}$。

推论 每个瓮中的平均球数是N/M，标准偏差为$\sqrt{N/M-N/M^2}$。

证明 用前面给出的累积开销乘以u^k和z^N，我们可得BGF

$$\begin{aligned}C^{[M]}(u,z)&=\sum_{N\geq 0}\sum_{k\geq 0}C_{Nk}^{[M]}u^kz^N=\sum_{N\geq 0}\sum_{k\geq 0}\binom{N}{k}(M-1)^{N-k}u^kz^N\\&=\sum_{N\geq 0}(M-1+u)^Nz^N\\&=\frac{1}{1-(M-1+u)z}\end{aligned}$$

上式除以M^N，或等价地用z/M代替z，就可将该累积BGF转换成稍微容易操作一些的PGF。对该PGF关于u微分，并在$u=1$处求解，即可得出和表3-6中一样的结果：

$$[z^N]\frac{\partial C^{[M]}(u,z/M)}{\partial u}\bigg|_{u=1}=[z^N]\frac{1}{M}\frac{z}{(1-z)^2}=\frac{N}{M}$$

439

及

$$[z^N]\frac{\partial^2 C^{[M]}(u,z/M)}{\partial u^2}\bigg|_{u=1}=[z^N]\frac{1}{M^2}\frac{z^2}{(1-z)^3}=\frac{N(N-1)}{M^2}$$

所以平均值为N/M，方差为$N(N-1)/M^2+N/M-(N/M)^2$，将其简化即可得到所述结果。■

我们曾使用熟知的经典方法给出过上述计算，然而符号法提供了一个快捷的推导方式。对一个特定的瓮而言，一个球未抛入该瓮的BGF是$(M-1)z$，而一个球抛入该壶的BGF是uz；因此和以前一样，关于球的一个序列的常规BGF是

$$\sum_{N\geq 0}((M-1+u)z)^N=\frac{1}{1-(M-1+u)z}$$

或者，和以前的处理方法一样，指数BGF

$$F(u,z)=\left(e^z+(u-1)\frac{z^k}{k!}\right)^M$$

将给出具有k个球的累积瓮数：

$$N![z^N]\frac{\partial F(u,z)}{\partial u}\bigg|_{u=1}=N![z^N]Me^{(M-1)z}\frac{z^k}{k!}=M\binom{N}{k}(M-1)^{N-k}$$

（类似的一个推导见6.4节）。

占有分布以及二项分布有着大量的应用，人们对这两个问题也进行过广泛地研究，所以导出上述这些结果的其他方法也有许许多多。事实上，重要的是要注意：每个瓮中的平均球数——这个看起来似乎是最重要的、最需要分析的量——是与分布完全无关的。无论球在翁中是如何分布的，累积开销都是N：统计每个瓮中的球数然后求出总数，就等价于直接统计球数。

440

不论球是不是随机分布的，每个瓮中的平均球数都是N/M。方差要告诉我们的是：一个给定的瓮中的球数是否可以期望在N/M的附近。

图8-4和8-5给出了在M的各种值下占有分布的图示。图8-4中最下方那组曲线恰好对应于图4-2，是中心在1/5处的二项分布。图8-5是对M较大时的描述，用泊松逼近比较合适。图8-5中下方两簇的极限曲线与图4-3中上方两簇的极限曲线是相同的，它们都是关于$N = 60$、$\lambda = 1$和$\lambda = 2$时的泊松分布。（图4-3中关于$N = 60$的其他极限曲线是关于$M = 15$和$M = 12$时的占有分布。）当M相对于N变小时，我们就进入了图8-4所阐述的范围内了，在这个范围内用正态逼近是比较合适的。

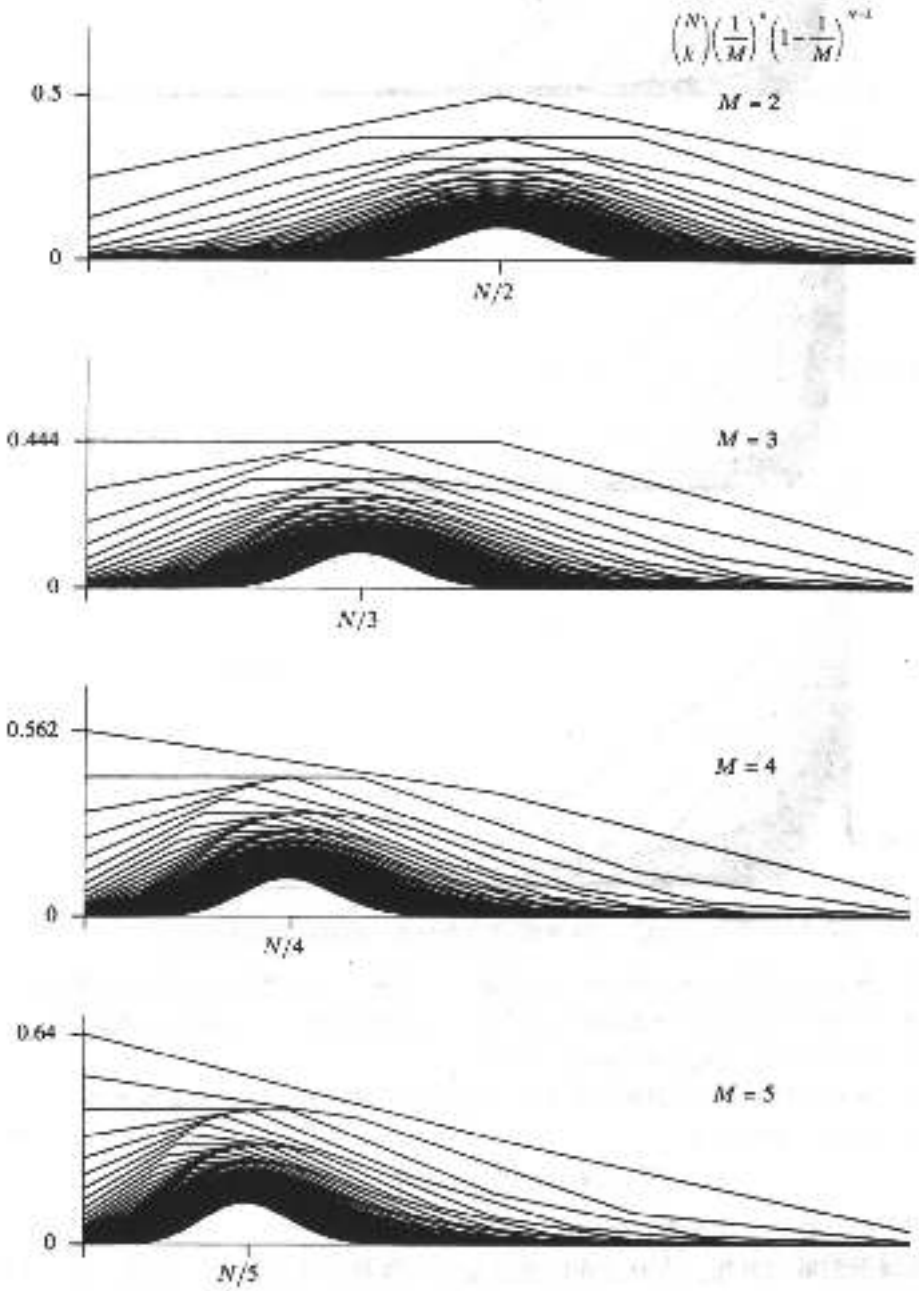


图8-4 关于M较小且$2 \leqslant N \leqslant 60$时的占有分布（$k$轴按$N$的比例标度）

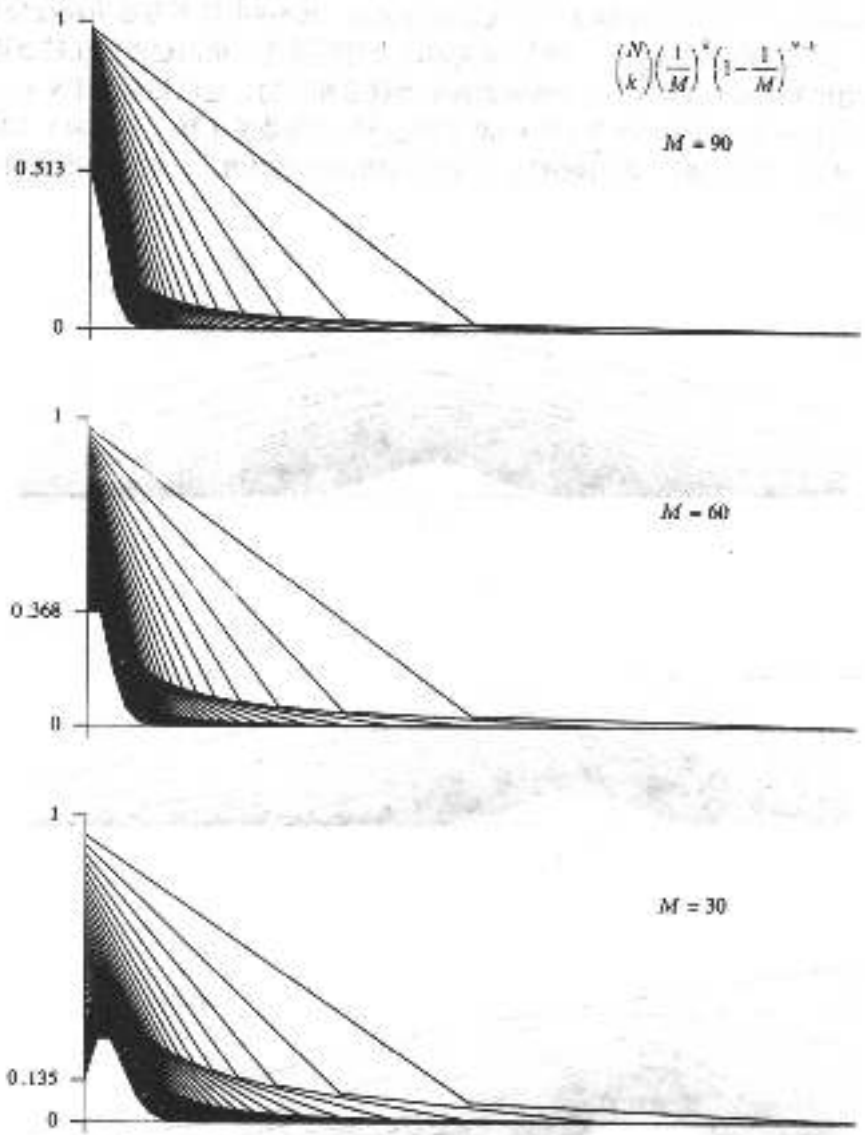


图8-5 关于M较大且$2 \leqslant N \leqslant 60$时的占有分布（$k$轴按$N$的比例标度）

习题8.27 当100个球随机分布到100个盒中时，其中的一个盒得到所有球的概率是多少？

习题8.28 当100个球随机分布到100个盒中时，每个盒得到一个球的概率是多少？

习题8.29 关于平均空盒数的标准差是多少？

习题8.30 当N个球随机分布到M个盒中时，每个盒中有偶数个球的概率是多少？

习题8.31 证明：对$N > 1$有

$$C_{Nk}^{(M)} = (M-1)C_{(N-1)k}^{(M)} + C_{(N-1)(k-1)}^{(M)}$$

并由此事实编写一个程序，对任意给定的M，打印出占有分布。

带有分离链表的散列分析。 占有分布的性质是散列算法分析的基础。例如，在一个使用分离链表的散列表中，一次不成功的搜索要涉及对一个随机链表的访问，并沿着该链表访问

到表尾。因此，这样的一次搜索的开销满足一个占有分布。

定理8.6（带有分离链表的散列） 对N个关键字使用一个长为M的表，平均而言，带有分离链表的散列对不成功的搜索需要进行N/M次探测，对成功的搜索需要进行$(N+1)/(2M)$次探测。

441～443

证明 关于不成功搜索的结果可以直接从前面的分析中得出。访问表中已存在的一个关键字的开销和把这个关键字存入表中的开销是相等的，所以，一次成功搜索的平均开销就是在建表过程中所有不成功搜索的平均开销，即：

$$\frac{1}{N}\sum_{1\le k\le N}\frac{k}{M}=\frac{N+1}{2M}$$

不成功搜索和成功搜索的开销之间的这种关系对许多查找算法都成立，包括二叉查找树在内。■

切比雪夫不等式表明：对1000个关键字，我们可能要使用100个链表，且每个链表中期望有10个数据项，并且至少有90%的把握保证一次搜索所检查的项数不会超过20个。对1000 000个关键字，我们可能要使用1000个链表，且切比雪夫不等式表明我们至少有99.9%的把握保证一次搜索所检查的项数不会超过2000个。尽管切比雪夫界在一般情况下都是适用的，但在我们所讨论的这种情况下，这个界还是相当粗糙的，我们可以通过直接的数值计算或通过关于1 000 000个关键字和表长为1000的泊松逼近公式证明：探测次数超过1300次的概率的量级为10^{-20}，探测次数超过2000次的概率的量级为10^{-170}。

此外，我们还能了解到有关散列结构的其他许多性质。例如，图8-6是函数$e^{-\alpha}$的图形，它告诉我们：空链表的百分比是关键字的个数与链表的个数之比的一个函数。这样的信息对于我们把一个算法调整到最佳性能有很大帮助。例如，在一个有大量关键字的应用中，正确地描述内存的需求，可能会涉及到要注意空间开销与每个链表的关系，以及所有的关键字最终都要存入相同的“表”中（计算机内存）等问题。

根据上面两点注意事项的启发，人们设计出了许多关于基本分离链接模式的变体。其中最引人注意的一个就是*接合散列法*，Vitter和Chen[29]给出了这个方法的详细分析（也可参见Knuth[22]）。这个方法在实际情况中，对一组性能参数值的分析进行应用是一个极好例子。

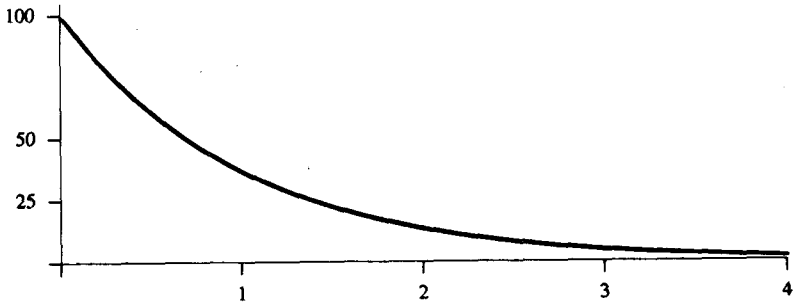


图8-6 空瓮的百分比是装填因子N/M的一个函数

习题8.32 对1000个关键字而言，当M为何值时使得带有分离链接的散列比二叉查找树访问的关键字要少？对1 000 000个关键字而言呢？

习题8.33 在带有分离链接的散列中，求成功查找所需的比较次数的标准差。

习题8.34 当散列表中的链表有序时（因此当发现一个关键字大于要查找的关键字时，可

以将一次查找截短），确定一次查找的比较次数的平均值和标准差。

习题8.35 [Broder和Karlin]分析程序8.1的下列变体：计算两个散列函数，将关键字存入两个链表中较短的链表中。

8.6 开放定址散列法

在分离链接散列法中，如果我们取$M = O(N)$，则搜索时间是$O(1)$，但为了维护二级数据结构，要使用大量指针形式的额外存储空间。所谓的开放定址法不使用指针，它在长为M的表内直接为N（$N \leq M$）个关键字指定地址。

生日悖论告诉我们：对具有相同散列值的某些关键字而言，它们并不需要一个太大的表，因而我们需要一个*冲突解决策略*来决定如何处理这样的冲突。

线性探测法。也许最简单的一个策略就是*线性探测法*：当把一个关键字插入到表中时，如果指定的位置（由散列值给定）已被占用，则检查前一个位置。如果前一个位置仍然已被占用，则检查再前面的一个位置，如此继续下去，直到找到一个空的位置。（如果到达了表的起始位置，则直接循环到尾部。）在球-瓮模型中，我们可以把线性探测想像成一种弹球盘（Pachinko）机：当一个瓮中已有一个球时，一个新球就会向左边跳，直到找到一个空瓮为止。

444 ~ 445

程序8.2给出了关于线性探测的搜索和插入的一个实现。程序中假定散列函数不返回零值，所以零可用来标记散列表中的空位置。

程序8.2 线性探测散列法

```
procedure insert(v: integer);
  var x: integer;
  begin
  x = hash(v);
  while table[x] <> 0 do x := (x-1) mod M;
  table[x] = v;
  end;
function search(v: integer): integer;
  var x: integer;
  begin
  x = hash(v);
  while (table[x] <> 0) and (table[x] <> v) do
    x := (x-1) mod M;
  if t = z then t = NIL;
  search := t;
  end;
```

我们将在下边看到：线性探测对一个几乎已满的表来说性能很不好，但对一个有着足够多空位置的表来说性能却相当不错。随着表越填越满，关键字容易“聚集”在一起，从而产生一个必须要进行搜索的长链以找到一个空位置。图8-7给出了一个利用线性探测填满表格的例子，其中最后两次插入时发生了聚集。避免聚集的一个简单办法就是：每当发现一个冲突时，不是查看前一个位置，而是查看前t个位置，其中t是由第二个散列函数计算出来的值。这种方法叫做*双散列*（double hashing）。

均匀散列。因为链表之间的相互依赖性，线性探测和双散列都很难进行分析。一个简单的近似模型就是假定长为M的表中N个关键字的每个占有配置都是等可能出现的。这等价于假定一个散列函数产生一个随机排列，且散列表中的每个位置是按随机次序检查的（关键字不同则次序也不同），直到找到一个空位置为止。

446

定理8.7（均匀散列） *对N个关键字使用长为M的表，平均而言，利用均匀散列时，用于成功查找和不成功的探测次数分别为*

$$\frac{M+1}{M-N+1} \quad 和 \quad \frac{M+1}{N}(H_{M+1}-H_{M-N+1})$$

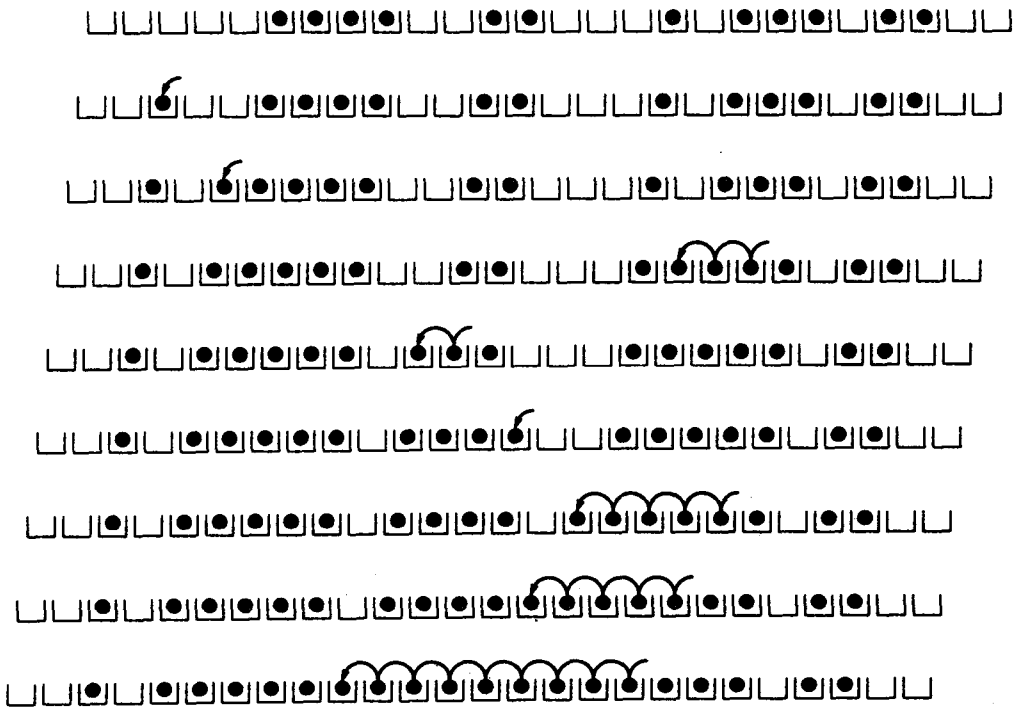


图8-7 线性探测散列法

证明 从散列位置开始，如果$k-1$个表位置都是满的，而第k个位置是空的，那么一次成功的查找需要k次探测。k个位置和$k-1$个关键字表明：满足这种情况的配置数和把其余$N-k+1$个关键字分布到其余$M-k$个位置的方式数是相同的。因此，在所有占有配置中，不成功查找的总开销是 447

$$\begin{aligned}
\sum_{1\le k\le M} k\binom{M-k}{N-k+1} &= \sum_{1\le k\le M} k\binom{M-k}{M-N-1} \\
&= \sum_{0\le k<M} (M-k)\binom{k}{M-N-1} \\
&= (M+1)\binom{M}{N} - \sum_{0\le k<M} (k+1)\binom{k}{M-N-1} \\
&= (M+1)\binom{M}{N} - (M-N)\sum_{0\le k<M}\binom{k+1}{M-N} \\
&= (M+1)\binom{M}{N} - (M-N)\binom{M+1}{N} \\
&= \frac{M+1}{M-N+1}\binom{M}{N}
\end{aligned}$$

将上式除以总的配置数 $\binom{M}{N}$，可得到不成功查找的平均开销。

成功查找的平均开销可以像定理8.6中的证明那样，通过求不成功查找开销的平均值而得到。 ■

因此，对$\alpha = N/M$，成功查找的平均代价渐近于$1/(1-\alpha)$。从直觉上看，对较小的α，我们期望第一个要检查的单元已满的概率为α，前两个要检查的单元已满的概率为α^2，依此类推，所以，平均开销应该渐近于

$$1+\alpha+\alpha^2+\alpha^3+\cdots$$

在均匀性假定下，这一分析证实了我们的直觉。和定理8.6中的证明一样，成功查找的平均开销可以通过求不成功查找平均开销的平均值计算出来。

均匀散列算法是不实用的，因为对每个关键字独立地生成一个随机排列的开销太大，然而，其相应的模型确实为其他冲突解决策略提供了一个性能目标。双散列就是逼近这样的一个“随机”冲突解决策略的尝试，且实际上其性能接近于上面给出的关于均匀散列的结果，不过导出这一结果曾经是多年以来一件非常困难的事情（见Guibas和Szemeredi[18]以及Lueker和Molodowitch[25]）。

448

线性探测法的分析。线性探测法是一个基本的搜索方法，对其聚集现象给出一个分析性的解释显然是值得关注的。Knuth第一个给出了该算法的分析，他声称这个算法的推导对他著作的构架方法有着重要的影响。他的著作在算法的数学分析中无疑对研究手段的建构方法有着重大影响。此算法的推导是一个原型范例，它表明一个简单的算法是如何导致有趣而又不平凡的数学问题的。

遵循Knuth的方法，我们定义三个量来对成功查找的累积开销产生一个准确的表达式：

f_{NM} = {0位为空的字数}，

g_{NMk} = {0位和$k+1$位为空，1到k位为满的字数}，

P_{NMj} = {插入第$(N+1)$个关键字需要$j+1$步的字数}。

和以往一样，上面出现的字指的是“散列值的序列。”

第一，通过注意位置0为空与任意其他表位置为空是等可能的，我们可以得到关于f_{NM}的一个显式表达式。这M^N个散列序列中的每一个都留下$M-N$个空的表位置，其总数为$(M-N)M^N$，除以M后得

$$f_{NM} = (M-N)M^{N-1}$$

第二，我们可以利用这个结果得到关于g_{NMk}的一个显式表达式。空位置把每个要计入总数的散列序列分成了独立的两个部分：一部分是含有k个元素的序列，它把k个元素散列到位置0到k中，且留下0位置为空；另一部分是含有$N-k$个元素的序列，它把$N-k$个元素散列到位置$k+1$到$M-1$，且留下$k+1$位置为空。因此

$$\begin{aligned} g_{NMk} &= \binom{N}{k} f_{k(k+1)} f_{(N-k)(M-k-1)} \\ &= \binom{N}{k}(k+1)^{k-1}(M-N-1)(M-k-1)^{N-k-1} \end{aligned}$$

第三，只要散列位置在k个连续非空单元所组成的块（这个块的两端均由空单元定界）的第k

个位置上时，则对于第（$N+1$）个关键字的插入，一个字将涉及$j+1$步（$j<k$）。再由循环的对称性，这样的字数是g_{NMk}，所以

$$p_{NMj}=\sum_{j<k<N} g_{NMk}$$

449

现在，就像前面一样，我们可以利用累积数P_{NMj}来计算平均查找的开销。因为关于不成功查找的累积开销是

$$\begin{aligned}\sum_{j>0}(j+1)p_{NMj}&=\sum_{j>0}(j+1)\sum_{j<k<N}g_{NMk}=\sum_{k>0}g_{NMk}\sum_{0<j<k}(j+1)\\&=\frac{1}{2}\sum_{k>0}(k+1)(k+2)g_{NMk}\\&=\frac{1}{2}\sum_{k>0}((k+1)+(k+1)^2)g_{NMk}\end{aligned}$$

因此，线性探测中不成功查找的平均开销是

$$\frac{1}{2}\left(S_{NM1}^{[1]}+S_{NM1}^{[2]}\right)$$

其中

$$S_{NMt}^{[i]}\equiv\frac{M-t-N}{M^N}\sum_k\binom{N}{k}(k+t)^{k-1+i}(M-k-t)^{N-k-i}$$

这个相当吓人的函数实际上用阿贝尔恒等式很容易计算（3.11节中习题3.66）。由此立即得出下面的结果

$$tS_{NMt}^{[0]}=1-\frac{N}{M}$$

对较大的i，容易证明（通过消去$(k+t)$的一个因子）

$$S_{NMt}^{[i]}=\frac{N}{M}S_{(N-1)M(t+1)}^{[i]}+tS_{NMt}^{[i-1]}$$

因此，

$$S_{NMt}^{[1]}=\frac{N}{M}S_{(N-1)M(t+1)}^{[1]}+1-\frac{N}{M}$$

其解为

$$S_{NMt}^{[1]}=1$$

这个结果正是我们所期望的，因为，比如说，$S_{NM1}^{[1]}=\sum_k p_{NMk}$就是概率的和。最后，当$i=2$时，我们有

$$S_{NMt}^{[2]}=\frac{N}{M}S_{(N-1)M(t+1)}^{[2]}+t$$

450

其解为

$$S_{NM1}^{[2]}=\sum_{0<i<N}i\frac{N!}{M^i(N-i)!}$$

定理8.8（线性探测散列法） 对N个关键字使用大小为M的表，平均而言，线性探测关于成功查找的探测次数为

$$\frac{1}{2}+\frac{1}{2}\sum_{0<i<N}\frac{(N-1)!}{M^i(N-i-1)!}=\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)+O\left(\frac{1}{N}\right)$$

关于不成功查找的探测次数为

$$\frac{1}{2}+\frac{1}{2}\sum_{0<i<N}i\frac{N!}{M^i(N-i)!}=\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)+O\left(\frac{1}{N}\right)$$

其渐近形式对$\alpha=N/M$及$\alpha<1$都成立。

证明 见上面的讨论。和前面一样，关于成功查找的表达式可以通过对不成功查找的结果求平均值而得到。

如果α严格小于1的话，则和式类似于定理4.8中的Ramanujan Q-函数，且这个和式由拉普拉斯方法不难估计。我们有

$$\sum_{0<i<N}\frac{N!}{M^i(N-i)!}=\sum_{0<i<N}\left(\frac{N}{M}\right)^i\frac{N!}{N^i(N-i)!}$$

将此和式分成两部分，我们可以利用该和式中的项当$i>\sqrt{N}$时开始变得小到可以忽略的事实，证明该和式为

$$\sum_{i>0}\left(\frac{N}{M}\right)^i\left(1+O\left(\frac{i^2}{N}\right)\right)=\frac{1}{1-\alpha}+O\left(\frac{1}{N}\right)$$

对上述式子加1再除以2，即可得到关于成功查找的所述结果。用类似的计算将得到关于不成功查找的所述估计式。 ■

451

推论 在对一个满表的一次成功的查找过程中，线性探测所检查的表中元素的平均个数$\sim\sqrt{\pi N/2}$。

证明 在上述表达式中取$M=N$恰好得到Ramanujan Q-函数，其近似值已在定理4.8中被证明。 ■

尽管解的形式比较简单，但是利用生成函数和符号法对线性探测平均开销进行推导的方法还没有找到。分析中出现的各种量之间有着许多有趣的性质。例如，如果我们用z^{N-1}乘以定理8.8中关于成功查找的表达式，再除以$(N-1)!$，然后对所有的$N>0$求和，将得到一个相当简洁的显式结果

$$\frac{1}{2}\left(e^z+\frac{e^M}{1-z/M}\right)$$

这个结果对于线性探测并没有什么直接的意义，因为所定义的量都只在$N<M$时有意义，但在组合解释意义上，此结果可能会是一个不错的选择。

表8-8总结了我们已讨论过的散列方法的渐近性能。该表包含了随着表长M和关键字个数N增加时，以装填因子$\alpha=N/M$的一个函数形式所表示的渐近开销；一个对较小的α进行开销估计的开销函数的展开式；关于α取特定值时函数的近似值等。该表还说明对于较小的α，所有方法的性能大体上是相同的；当表满的程度达到80%~90%时，线性探测的性能开始下降到难以令人接受的程度；除非表很满，否则双散列的性能相当接近于“最佳”的性能（与分离链表法相同）。这些结果以及相关结果在实际情况下散列的应用中都是非常有用的。

习题8.36 在平均搜索的开销超过之$\ln N$前，一个长为M的线性探测表中能插入多少个关键字？

习题8.37 对一个满表使用线性探测法时，计算一次不成功查找的准确开销。

习题8.38 对不成功查找开销的EGF给出一个显式表式法。

习题8.39 使用符号法导出在一次成功的查找中，对固定的M，线性探测所需要的探测次数的EGF。[⊖]

452

大小为M的表中关于N个关键字的准确开销。

表8-8 关于散列法的分析结果

	成功查找	不成功查找
分离链接法	$1+\frac{N}{2M}$	$1+\frac{N}{M}$
均匀散列法	$\frac{M+1}{M-N+1}$	$\frac{M+1}{N}(H_{M-1}-H_{M-N+1})$
线性探测法	$\frac{1}{2}\left(1+\sum_k \frac{k!}{M^k}\binom{N-1}{k}\right)$	$\frac{1}{2}\left(1+\sum_k k\frac{k!}{M^k}\binom{N}{k}\right)$

当$N, M\to\infty$, $\alpha \equiv N/M$时的渐近开销

	平均	0.5 0.9 0.95	小α
不成功查找			
分离链接法	$1+\alpha$	2 2 2	$1+\alpha$
均匀散列法	$\frac{1}{1-\alpha}$	2 10 20	$1+\alpha+\alpha^2+\cdots$
双散列法	$\frac{1}{1-\alpha}$	2 10 20	$1+\alpha+\alpha^2+\cdots$
线性探测法	$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$	3 51 201	$1+\alpha+\frac{3\alpha^2}{2}+\cdots$
成功查找			
分离链接法	$1+\frac{\alpha}{2}$	1 1 1	$1+\frac{\alpha}{2}$
均匀散列法	$\frac{1}{\alpha}\ln(1+\alpha)$	1 3 4	$1+\frac{\alpha}{2}+\frac{\alpha^2}{3}+\cdots$
双散列法	$\frac{1}{\alpha}\ln(1+\alpha)$	1 3 4	$1+\frac{\alpha}{2}+\frac{\alpha^2}{3}+\cdots$
线性探测法	$\frac{1}{2}\left(1+\frac{1}{1-\alpha}\right)$	2 6 11	$1+\frac{\alpha}{2}+\frac{\alpha^2}{2}+\cdots$

453

8.7 映射

散列到满表的研究将自然导致考查把一个1到N的整数集合映射到它自身的映射性质。对这些性质的研究又将导致一个定义简洁且重要的组合结构，然而这个结构却包含了本书我们所研究过的大部分内容。

定义 一个N-映射是一个函数f，它把区间$[1...N]$中的整数映射到区间$[1...N]$中。

和字、排列及树一样，我们还是通过写出映射的功能表来指定一个映射：

下标	1	2	3	4	5	6	7	8	9
映射	9	6	4	2	4	3	7	8	6

⊖ 我们不知道这道题目的答案！

和以往一样，我们去掉下标，并把一个映射简单地规定为区间1到N中N个整数的序列（映射的像）。显然，存在N^N个不同的N-映射。我们对排列（第6章）和树（第5章）曾使用过类似的表示方法——排列和树可以被认为是映射的两个特例。例如，排列是一个映射，只不过像中的整数都是不同的。

自然地，我们可以定义一个随机映射，它是由1到N之间变化的N个随机整数所构成的序列。我们将要关注对随机映射性质的研究。例如，一个随机映射是一个排列的概率为$N!/N^N \sim \sqrt{2\pi N}/e^N$。

像的基数。映射的某些性质可以从上一节得出的子的性质中推导出来。例如，由定理8.5我们知道，映射中出现k次的整数的平均个数为$\lambda = Ne^{-\lambda}/k!$，它是$\alpha = 1$时的泊松分布。我们关心的一个相关问题是所出现的不同整数的个数的分布，即像的基数。它等于N减去不出现的整数的个数，或等于一个占有模型中"空盒"的个数，所以根据定理8.5的推论，这个数的平均值是$(1-1/e)N$。通过一个简单的计数论证可以确定：像中有k个不同整数的映射个数是$\binom{N}{k}$（整数的选取）乘以$k!\left\{{N \atop k}\right\}$（基数为$k$的像对应的所有满射数）。因此，

$$C_{Nk} = k!\binom{N}{k}\left\{{N \atop k}\right\}$$

454 图8-8给出了这个分布的曲线图。

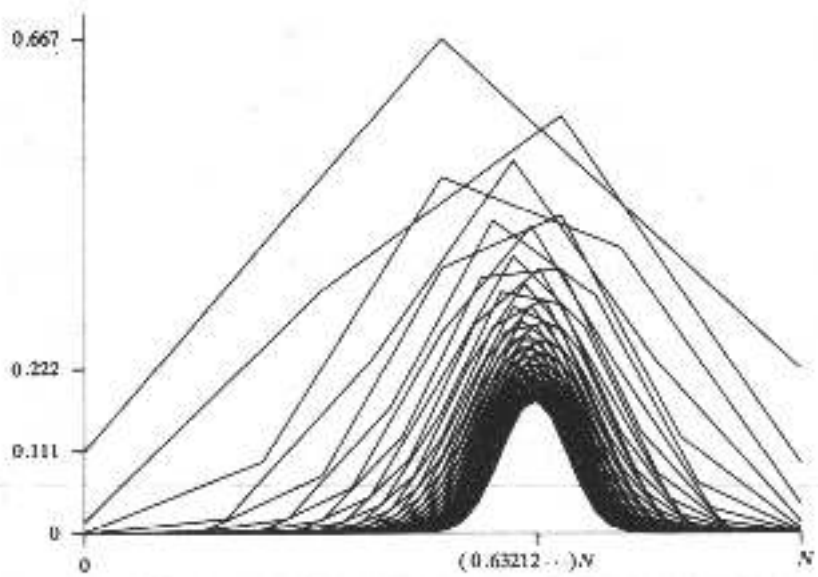


图8-8 $3 \le N \le 50$时随机映射的像基数（k轴按N的比例标度）

习题8.40 求像的基数分布的指数BGF。

习题8.41 利用组合论证的方法，求像的基数分布的指数BGF。

习题8.42 给出一个大小为N，且像中具有k个不同整数的映射的个数的递推关系，并利用此关系求得一个关于$N<20$时的值的表。

习题8.43 给出一个大小为N，且具有k个不同字母的M-字的个数的显式表达式。

随机数发生器。一个随机N-映射是任何一个皆以1到N的整数为定义域和值域的函数f，其

中所有N^N个这样的函数均等可能取到。例如，下面的映射由函数$f(i) = 1 + i^2 \bmod 9$所定义：

下标	1	2	3	4	5	6	7	8	9
映射	2	5	1	8	8	1	5	2	1

455

这种函数的一个应用就是用来模拟随机数发生器，也即这样的程序：它产生其性质尽可能类似于随机序列性质的数的序列。其想法是选择一个函数，该函数是一个N-映射，然后通过从一个叫做种子的初始值开始，迭代$f(x)$以产生一个（伪）随机序列。给定一个种子u_0，我们得到序列

$$
\begin{aligned}
&u_0\\
&u_1 = f(u_0)\\
&u_2 = f(u_1) = f(f(u_0))\\
&u_3 = f(u_2) = f(f(f(u_0)))\\
&\quad\vdots
\end{aligned}
$$

例如，线性同余随机数发生器基于函数

$$f(x) = (ax + b) \bmod N$$

二次随机数发生器基于函数

$$f(x) = (ax^2 + bx + c) \bmod N$$

二次随机数发生器与平方取中法有着紧密的联系，这个古老的想法要追溯到冯·诺依曼时代：从一个种子u_0开始，反复对前面生成的值进行平方，然后再抽取中间几位。例如，使用四位十进制数，从种子$u_0 = 1234$开始生成的序列是$u_1 = 5227$（因为$1234^2 = 01522756$），$u_2 = 3215$（因为$5227^2 = 27321529$），$u_3 = 3362$（因为$3215^2 = 10336225$），依此类推。

我们很容易设计一个线性同余随机数发生器来产生一个排列（即：它产生N个不同的值之后再重复）。读者可参考Knuth[21]，那里有一套完整的代数理论。

二次随机数发生器从数学上来讲比较难以分析。但Bach[2]已经证明：平均而言，迭代中的二次函数所具有的性质基本上与随机映射的性质相同。Bach使用了代数几何学中的深奥结果；正如我们将要看到的，就目前为止，在本书已给出的所有技巧的前提下，随机映射的性质相对而言还是较为容易分析一些的。在我们所关注的量的平均值渐近相等的意义下，模数N的二次三项式有N^3个，且我们断定它们代表N^N个随机映射。这种情况有些类似于前面我们所描述过的双散列的情况：在这两个例子中，实际方法（二次发生器、双散列）都要通过与随机模型的渐近等价关系来研究（随机映射、均匀散列）。 456

换句话说，二次发生器为所谓的随机数发生器的研究提供了一种动机，因为随机选择一个函数并对其进行迭代就是产生随机数的源泉。事实上，分析的结果却是否定的，因为分析结果表明线性同余生成器可能要比二次生成器更好一些（因为它们有较长的圈），而这些想法的一个有趣结果则是关于整数的因子分解的Pollard rho方法的设计与分析，我们将在本节末讨论这个问题。

习题8.44 证明：每个随机映射都必须至少有一个圈。

习题8.45 对$N = 100$、1000、$10\ 000$以及在这些值附近的质数，考查由$f(i) = 1 + (i^2 + 1) \bmod N$所定义的随机映射的性质。

路径长与连通分量。因为把映射应用于它自身的操作是有定义的，这就使我们想到，如果我们相继把映射作用于自身会发生什么情况。在一个映射中，对每个k，序列

$$f(k), f(f(k)), f(f(f(k))), f(f(f(f(k)))), \cdots$$

都是有确切定义的：该序列有什么性质呢？我们很容易就能发现：由于只有N个不同的值，所以该序列最终必定要重复一个值，并在这个值处变成一个圈。例如，如图8-9所示，在由$f(x) = (x^2 + 1)\bmod 99$所定义的映射中，从$x_0 = 3$处开始，我们最终将得到一个循环序列3, 10, 2, 5, 26, 83, 59, 17, 92, 50, 26, …。这样的序列总是要有一个圈的，且圈的前面有一个由某些值所构成的“尾巴”。在这个例子中，圈的长为6，尾部的长为4。了解随机映射中圈长和尾长的统计性质，是我们将要关注的事情。

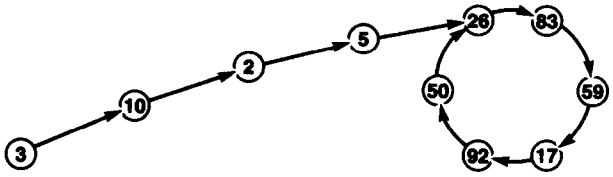


图8-9 从$x_0 = 3$处开始迭代$f(x) = (x^2 + 1)\bmod 99$时的尾部和圈

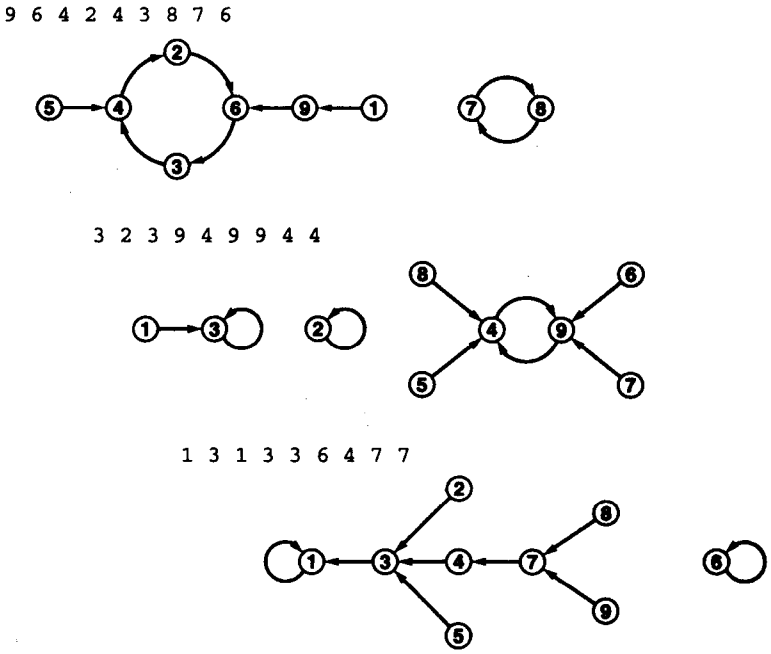


图8-10 3个随机映射的树-圈表示法

圈长和尾长取决于起始点。图8-10给出了一种图形表示，它对三个样本映射中的每一个i，给出了i连接到$f(i)$的图示。例如，在图中顶部的那个映射中，如果我们从7开始，我们马上就陷入了圈7，8，7，…中，但如果我们从1开始，我们就会遇到一个两-元素的尾巴，接着是一个四-元素的圈。这种表示法更清楚地揭示了映射的结构：每一个映射都能分解成一组连通的成分，也叫*连通映射*（connected maps）。每个组成部分都由一组环绕着同一个圈的所有点所构成，且圈上的每个点都附属于从那个点进入圈的所有点所构成的一棵树。从每一个单个点的观点来看，我们有一个如图8-9中那样的尾-圈，但整个结构无疑更能表达映射的信息。

457 ~ 458

正如前面我们所关注过的，映射是排列的推广，在排列中我们限定在一个变化范围内的每个元素必须出现一次，这就导致了一组圈。在对应于排列的映射中，所有尾部的长都是0。如果一个映射中所有的圈长都为1，那么该映射就是对应于森林的映射。这就自然导致我们考虑路径长的想法:

定义 映射f中关于下标k的路径长或ρ长是由迭代

$$f(k), f(f(k)), f(f(f(k))), f(f(f(f(k)))), \cdots$$

所得到的不同整数的个数。映射f中关于下标k的圈长是迭代达到的圈的长，映射f中关于下标k的尾长是ρ长减去圈长，或等价地说，是连接到圈时所需的步数。

一个下标的路径长叫做“ρ长”，这是由于尾部加上圈的形状使人联想到希腊字母ρ（见图8-9）。映射除了具有这些我们从个别的点所看到的性质外，我们也将关注映射中涉及所有点的总体度量。

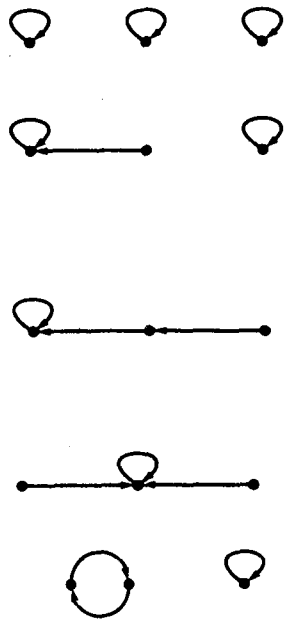
定义 映射f的ρ长是f中关于所有k的ρ长之和。映射f的树路径长是f中关于所有k的尾长之和。

因此，由图8-10，很容易检验9 6 4 2 4 3 8 7 6的ρ长度是36，树路径长为4；3 2 3 9 4 9 9 4 4的ρ长是20，树路径长为5；1 3 1 3 3 6 4 7 7的ρ长是27，树路径长为18。在这些定义中，树路径长不包括圈上任何结点的开销，而ρ长却包括结构中关于每个结点的整个圈长。两个定义都给出了那些是树的映射的路径长的标准概念。

459

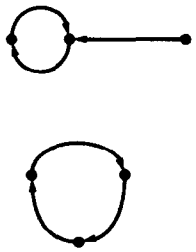
表8-9 3个元素的所有映射的基本性质

映射	圈	树	ρ长	最长圈	最长路径
123	3	0	3	1	1
113	2	1	4	1	2
121	2	1	4	1	2
122	2	1	4	1	2
133	2	1	4	1	2
223	2	1	4	1	2
323	2	1	4	1	2
112	1	1	6	1	3
131	1	1	6	1	3
221	1	1	6	1	3
322	1	1	6	1	3
233	1	1	6	1	3
313	1	1	6	1	3
111	1	2	5	1	2
222	1	2	5	1	2
333	1	2	5	1	2
213	2	0	5	2	2
321	2	0	5	2	2
132	2	0	5	2	2



（续）

映射	圈	树	ρ长	最长圈	最长路径
211	1	1	7	2	3
212	1	1	7	2	3
232	1	1	7	2	3
311	1	1	7	2	3
331	1	1	7	2	3
332	1	1	7	2	3
231	1	0	9	3	3
312	1	0	9	3	3



460

我们很想知道图8-10中所示的几种结构的一些基本性质：

- 存在多少个圈？
- 圈上有多少个点，树中有多少个点？
- 平均圈长是多少？
- 平均ρ长是多少？
- 最长圈的平均长是多少？
- 通向一个圈的最长路径的平均长是多少？
- 最长ρ路径的平均长是多少？

表8-9对所有的3-映射给出了上述基本数据一个详尽的列表，而表8-10给出了6个随机9-映射。我们在表8-9的右端画出了出现在3-映射中7个不同的树-圈结构，它提醒我们映射的树-圈表示法是有标号、有序的组合对象。

和我们在本章已经见过的其他几个问题一样，随机映射的某些性质可以用一种直接的概率论证方法来进行分析。例如，一个随机映射的平均ρ长度就能很容易地利用这种方法推导出来。

表8-10 一些随机9-映射的基本性质

映射	占有	分布	圈	树	ρ长度	最长圈	最长路径
323949944	012300003	5112	3	3	19	2	3
131336477	203101200	4221	2	1	27	1	5
517595744	100230201	4221	2	4	29	3	5
215681472	220111110	2520	1	2	42	2	8
213693481	212101011	2520	3	2	20	2	4
964243786	011212111	1620	2	1	34	4	6

定理8.9（ρ长） 平均而言，随机映射中一个随机点的ρ长是 $\sim\sqrt{\pi N/2}$，一个随机映射的ρ长 $\sim N\sqrt{\pi N/2}$。

证明 假定我们从x_0开始。显然$f(x_0)\neq x_0$的概率是$(N-1)/N$，ρ长大于等于1的概率也是这个值。类似地，ρ长大于等于2的概率等于头两个元素不同（$f(x_0)\neq x_0$）且第三个元素与头两个元素也都不同（$f(f(x_0))\neq x_0$且$f(f(x_0))\neq f(x_0)$）的概率，或等于$(N-1)/N$乘以$(N-2)/N$。依此类推，我们有

461

$$\Pr\{\rho\text{ 长}\geqslant k\}=\frac{N-1}{N}\frac{N-2}{N}\cdots\frac{N-k}{N}$$

因此，随机映射中一个随机点的平均ρ长是这些累积概率之和，它恰好是Ramanujan Q-函数，所以定理4.8的近似式给出了它的答案。映射中N个点中的每一个点都满足同样的论证方法，因此，映射的期望ρ长度可以通过乘以N来得到。 ■

这个问题等价于8.3节中的生日问题，尽管它们的随机模型在形式上并不相同。

习题8.46 证明：对随机映射中一个随机点的ρ长度的分析等价于对生日问题的分析。

生成函数。映射的许多其他性质更依赖于总体结构的相互作用。这些性质可以通过生成函数得到很好的分析。映射是树的圈的集合，所以它们的生成函数很容易用符号法推导出来。我们可用与第3章所介绍过的符号法完全一样的方法来统计“圈的集合”，只不过要以树作为基本对象。

根据第5章，我们从关于Cayley树的EGF:

$$C(z) = ze^{C(z)}$$

开始。和第3章一样，对树（连通映射）的圈计数的EGF为

$$\sum_{k\geq 1}\frac{C(z)^k}{k} = \ln\frac{1}{1-C(z)}$$

且对连通映射的集合计数的EGF为

$$\exp\left(\ln\frac{1}{1-C(z)}\right) = \frac{1}{1-C(z)}$$

表8-11 关于映射的指数生成函数

类	EGF	系数
树	$C(z) = ze^{C(z)}$	$(N-1)![u^{N-1}]e^{Nu} = N^{N-1}$
树圈	$\ln\frac{1}{1-C(z)}$	$(N-1)![u^{N-1}]\frac{1}{1-u}e^{Nu} \sim N^N/\sqrt{\pi N}$
映射	$\exp\left(\ln\frac{1}{1-C(z)}\right)$	$(N-1)![u^{N-1}]\frac{1}{(1-u)^2}e^{Nu} = N^N$

462

上面的函数方程包含了隐含定义的Cayley函数$C(z)$，而且拉格朗日反演定理可以直接应用。例如，对刚才导出的EGF应用该定理将导致下面的计算:

$$\begin{aligned}[z^N]\frac{1}{1-C(z)} &= \frac{1}{N}[u^{N-1}]\frac{1}{(1-u)^2}e^{Nu}\\ &= \sum_{0\leq k\leq N}(N-k)\frac{N^{k-1}}{k!} = \sum_{0\leq k\leq N}\frac{N^k}{k!} - \sum_{1\leq k\leq N}\frac{N^{k-1}}{(k-1)!}\\ &= \frac{N^N}{N!}\end{aligned}$$

这就验证了长为N的映射共有N^N个这一事实。

为方便起见，我们把涉及将拉格朗日反演定理应用到Cayley函数的计算归纳如下:

引理 对于Cayley函数$C(z)$，当 $g(z) \equiv \sum_{k\geq 0} g_k z^k$ 时，我们有

$$[z^N]g(C(z)) = \sum_{0\leq k\leq N}(N-k)g_{N-k}\frac{N^{k-1}}{k!}$$

证明 由定理3.9可立即得证。 ■

因此，N个结点的连通映射的个数是

$$N![z^N]\ln\frac{1}{1-C(z)}=N!\sum_{0<k<N}(N-k)\frac{1}{N-k}\frac{N^{k-1}}{k!}=N^{N-1}Q(N)$$

这里又一次出现了Ramanujan Q-函数。

上述结果意味着，一个随机映射是树的概率恰好为$1/N$，且一个随机映射是一个单连通分量的概率渐近于$\sqrt{\pi/(2N)}$。我们可以用类似的方式，利用BGF来分析随机映射的其他性质。

463

定理8.10（分量和圈） 一个随机N-映射平均有$\sim\frac{1}{2}\ln N$个分量和$\sim\sqrt{\pi N}$个圈上的结点。

证明 根据上面的讨论，关于分量个数分布的BGF为

$$\exp\left(u\ln\frac{1}{1-C(z)}\right)=\frac{1}{(1-C(z))^u}$$

于是，由上面给出的引理，得平均分量数为

$$\begin{aligned}\frac{1}{N^N}[z^N]\frac{\partial}{\partial u}\frac{1}{(1-C(z))^u}\Big|_{u=1}&=\frac{1}{N^N}[z^N]\frac{1}{1-C(z)}\ln\frac{1}{1-C(z)}\\&=\sum_{0<k<N}(N-k)H_{N-k}\frac{N^{k-1}}{k!}\end{aligned}$$

所述的渐近结果用定理4.8证明中的方法可以直接计算出来。

关于圈上结点个数的BGF为

$$\exp\left(\ln\frac{1}{1-uC(z)}\right)=\frac{1}{1-uC(z)}$$

从这个式子中提取的系数恰好就是上述结果。 ■

其他性质。各种其他性质可以用类似的方法来处理。并且，利用与第6章中关于排列以及本章前面关于字的同样的处理方法，我们可求得恰好有k个分量的映射数的EGF为$C(z)^k/k!$；最多有k个分量的映射数的EGF为$1+C(z)+C(z)^2/2!+\cdots+C(z)^k/k!$；等等。这些结果可用来得出关于极值参数的显式表达式，就像我们以前曾多次做过的那样。估计这些量的渐近方法在Flajolet和Odlyzko[11]以及Kolchin[23]中都有详细的讨论（也可参见[12]）。表8-12总结了[11]中所给出的结果。极值参数中的各种常量都可以显式地表示出来，不过这需要进行相当复杂的定积分。有趣的是我们注意到：对于平均值而言，圈长加上尾长等于ρ长，但对极值参数而言，上述结论并不成立。这意味着对大量的映射来说，最高的树并不附属于最长的圈。

464

表8-12 随机映射的渐近性质

随机点的平均值	
ρ长	$\sqrt{\pi N/2}$
尾长	$\sqrt{\pi N/8}$
圈长	$\sqrt{\pi N/8}$
树大小	$N/3$
分量大小	$2N/3$

（续）

平均数	
k-结点	$\frac{Ne^{-1}}{k!}$
k-圈	$\frac{1}{k}$
k-分量	$\frac{e^{-k}}{k!}$(# k - node connected maps)
k-树	$(\sqrt{\pi N/2})\frac{e^{-k}}{k!}$(# k - node trees)
极值参数（结点期望数）	
最长尾	$\sqrt{2\pi N}\ln 2 \approx 1.74\sqrt{N}$
最长圈	$\approx 0.78\sqrt{N}$
最长ρ-路径	$\approx 2.41\sqrt{N}$
最大树	$\approx 0.48N$
最大分量	$\approx 0.76N$

465

习题8.47 哪些N-映射具有最大和最小的ρ长？树路径长呢？

习题8.48 写一个程序，求出随机映射的ρ长和树路径长。对尽可能大的N，生成1000个随机映射，并计算平均圈长、ρ长和树路径长。

习题8.49 写一个程序，不能利用任何额外存储空间，求一个随机映射的ρ长和树路径长。

习题8.50 一个没有重复整数的映射是一个排列。给出一个有效的算法，确定一个映射是否是树。

习题8.51 对所有的$N<10$，计算一个随机N-映射中最大分量的平均大小。

习题8.52 用BGF证明定理8.9。

习题8.53 当一个随机映射迭代两次时，像中不同整数的平均个数是多少？

习题8.54 考虑对应于所有N-映射的N^N个树-圈结构。当$N<7$，且把这些结构看成是无标号、无顺序的对象时，这些结构中有多少个是不同的？（它们叫做**随机映射模式**（random mapping patterns）。）

习题8.55 描述偏映射（partial map）的图形结构，其中一个点的像可能无定义。建立相应的EGF方程，并验证长为N的偏映射的个数是$(N+1)^N$。

习题8.56 分析1到N范围内的$2N$个随机整数的序列中的“路径长”。

习题8.57 生成100个大小为10、100和1000的随机映射，并通过观察检验表8-12中所给出的统计量。

8.8 整数因子分解与映射

本节我们将考查Pollard ρ法，这是一个有效的分解整数的算法（在10到30个十进制位的“中等”范围内），这种方法要依赖映射的结构和概率性质，并基于对如下两个事实的应用。

- 某个圈上的点可以很快被找到（所用时间与一个起始点的ρ长成正比），所使用的内存为$O(1)$。
- 随机映射上一个随机点的平均ρ长为$O(\sqrt{N})$。

466

首先，我们考虑圈的检测问题，然后我们再考虑Pollard ρ法本身。

圈检测。要找出映射的一个圈上的一点，一个朴素的方法就是对映射进行迭代，存储查找结构中所有的函数值，观察每个新的值是不是结构中已经存在的某个值。该算法对大的映射并不实用，因为我们无法担负存储所有函数值的空间开销。

程序8.3给出了一个在任意映射中求圈上的点的方法，该方法只利用了常数空间而且不牺牲执行时间，它是由Floyd（参见Knuth[21]）给出的。在这个程序中，我们可以把点a看成正在以速度1沿着ρ-图运动（见图8-9），把点b看成正在以速度2沿着ρ-图运动。该算法依赖于这样的事实：一旦这两个点都在圈上时，它们必定会在某个点上相撞。例如，假定该算法用于图8-9中的映射且起始点为10时，则在7步之内，圈被检测出来，这可以通过跟踪a和b的取值而看出：

a	10	2	5	26	83	59	17
b	10	5	83	17	50	83	17

程序8.3 圈检测的Floyd方法

```
a := x; b := x; t := 0;
repeat
  a := f(a);
  b := f(f(b));
  t := t+1;
until (a = b);
```

定理8.11（圈检测） *给定一个映射和起始点x，Floyd算法使用常数空间，并在与x的ρ长成正比的时间内找到映射中的一个圈上的点。*

证明 设λ是x的尾长，μ为圈长，$\rho=\lambda+\mu$为ρ长。经λ步循环之后，点a到达了圈，而点b已经在圈上。现在我们在圈上有一个赛跑，其速度差为1。最多经过μ步，b将赶上a。设t为当算法结束时变量t的值，则我们也必有

467

$$t \leqslant \rho \leqslant 2t$$

左端的不等式成立，是因为t是点a的位置，且算法在a第二次开始绕圈运行之前就应结束。右端的不等式成立，是因为$2t$是点b的位置，且算法应在b至少绕行一圈之后才能结束。因此，该算法不仅给出了圈上的一个点（变量a和b结束时的值），也给出了起始点的ρ长度的一个估计，且所估计的值在一个因子2的范围之内。 ■

通过再多存储一些函数值，差不多可以消去算法所花费时间中的额外因子2（时间近似于ρ长），而所用的存储空间仍然比较合理。Sedgewick, Szymanksi和Yao[28]对这一问题给出了详细的研究。

习题8.58 利用Floyd方法在你的机器上对较短的圈检验随机数生成器。

习题8.59 利用Floyd方法检验平方取中法随机数生成器。

习题8.60 利用Floyd方法估计关于函数

$$f(x)=(x^2+c)\bmod N$$

中与各种起始值、c和N等有关的ρ长。

Pollard ρ法。ρ法是一个以高概率分解整数的随机化算法。程序8.4给出了它的一个实现，但要注意，必须假定算法能对很大的整数进行操作。该算法根据一个随机选择的c值，然后再

从随机选择的一个起始点开始迭代二次函数

$$f(x) = (x^2 + c) \bmod N$$

直到找到一个圈上的点为止。

为简单起见，假定$N = pq$，其中p和q是算法要找的素数。根据中国剩余定理，任何一个整数$y \bmod N$，都由y的值mod p和mod q所确定。特别地，函数f由一对函数

$$f_p(x) = x^2 + c \bmod p \text{和} f_q(x) = x^2 + c \bmod q$$

468

所确定。如果将圈检测的算法应用到f_p，并起始于一个初始值x，那么经过t_p步之后将会检测到一个圈，其中t_p的值最大是x（mod p）的ρ长度的二倍。类似地，经过t_q步之后，将会检测到关于t_q的一个圈（mod q）。因此，如果$t_p \neq t_q$（对大数而言，发生这种情况的概率非常大），那么经过$\min(t_p, t_q)$步之后，我们将发现算法中变量a、b的值a和b应满足

$$a \equiv b \pmod p \text{和} a \not\equiv b \pmod p \quad (t_p < t_q)$$

$$a \not\equiv b \pmod p \text{和} a \equiv b \pmod p \quad (t_p > t_q)$$

在上述任何一种情况之下，$a - b$和N的最大公因子都是N的一个非平凡因子。

程序8.4 分解因子的Pollard ρ法

```
function factor(N: integer): integer;
  var a, b, c, d: integer;
  begin
    a := randominteger(0, N-1);
    b := a;
    c := randominteger(0, N-1);
    repeat
      a := (a*a + c) mod N;
      b := (b*b + c)*(b*b + c) + c mod N;
      d := gcd(a-b mod N, N);
    until (d <> 1);
  factor := d;
  end;
```

与随机映射的联系。假设形如$f(x) = (x^2 + c) \bmod N$的二次函数的路径长性质渐近地等价于随机映射中的路径长。这种探索性的假设表明N个二次映射（c有N种可能的选择）的性质与N^N个随机映射的性质是相似的。换句话说，二次函数被假定成随机映射的一个“代表样本”。这个假设是很有道理的，并且通过模拟得到了广泛的证实，但它只是被部分地证明[2]。尽管如此，它还是导出了关于Pollard法的一个有用的近似分析。

469

上面的讨论表明：Pollard算法所花费的步数是$\min(t_p, t_q)$，其中t_p和t_q分别是f_p和f_q的ρ长。根据定理8.9，随机N-映射上一个随机点的ρ长为$O(\sqrt{N})$，所以在我们上面一段所讨论过的假设下，我们应期望算法在$O(\min(\sqrt{p}, \sqrt{q}))$步内结束，也即在$O(N^{1/4})$步内结束。此论证显然是对$N$有多于两个因子时的情形的推广。

定理8.12 (Pollard ρ法) *在二次函数中路径长渐近等于随机映射中路径长的探索性假设下，Pollard ρ法将一个复合整数N分解因子所需的期望步数是$O(\sqrt{p})$，其中p是N的最小素数因子。特别地，平均而言，N在$O(N^{1/4})$步内完成分解。*

证明 见上面的讨论。其整体界是根据$p < \sqrt{N}$的事实得出的。 ■

1980年，Brent[26]利用Pollard方法第一次分解了第8个费马数。Brent发现

$$F_8 = 2^{2^8} + 1 \approx 1.11579 \cdot 10^{77}$$

具有素数因子

1238926361552897

（也可参见Knuth[21]。）目前，虽然这个算法的近似分析是建立在一个还没有被完全证明的假设条件之下的，但这一事实并没有降低它的效用。事实上，对随机映射性质的了解给了我们这样的信心：这个方法是应该能够进行有效分解的，事实上它也确实如此。

表8-13给出了利用Pollard方法分解形如$N = pq$的数时所用的步数，其中p和q分别是所选取的接近于形如数1234…和4321…的质数。尽管c和起始点应该是随机选取的，但$c = 1$和起始点$a = b = 1$对该应用的效果也非常好（这样也容易重新产生结果）。从这个表中我们可以看到：当N以因子约为100的速度增加时，开销大约以因子为3的速度增加，这与定理8.12极为吻合。该方法在分解表中最后一个数（该数大约为1.35×10^{19}）时，所用的步数少于24 000，而穷举试验法将需要大约10^9次运算。

表8-13 Pollard算法的执行样例（c = 1）

N	步 数
13 · 23	4
127 · 331	10
1237 · 4327	21
12347 · 54323	132
123457 · 654323	243
1234577 · 7654337	1478
12345701 · 87654337	3939
123456791 · 987654323	11225
1234567907 · 10987654367	23932

字和映射与组合数学中的经典问题和算法分析中的“经典”问题有着直接的联系。我们讨论过的许多方法和结果在数学上都是众所周知的（伯努利试验、占有问题），并且它们广泛地用到算法分析领域之外。这些经典问题与现代应用也有着直接的联系，如预测散列算法的性能等。在这样的新领域中，对这些经典问题进行详细的研究将会导致具有独立意义的新问题。

散列算法是各种算法中第一批进行了数学分析的算法，且它们仍然具有最重要的实际意义。新的应用类型以及软、硬件基本特征的改变，使得散列算法与我们这里所讨论的以及文献中所介绍的有关散列算法的技术和结果仍然持续着这种紧密的联系。

随机映射的分析简明地总结了算法分析的一般方法。我们建立对应于基本组合结构的生成函数的函数方程，然后利用解析工具提取系数。在这种情况下符号法对前者尤其有效，而对后者而言，拉格朗日反演定理是一个重要的工具。

映射是以每个元素恰好映射到另外一个元素的事实为特征的。在图的表示中，这意味着图中恰好有N条边，且每个元素恰好有一条由该元素所指向的边，不过可能有多个元素指向某一个特定的元素。下一个推广就是图，图把上面的这种限制去掉了，即每个元素可以指向任意多个其他元素。与映射相比，或与我们在本书中所考查过的任何其他组合结构相比，图都更加复杂，因为把图分解成较简单的结构将更加困难，而这些简单结构通常都是我们通过求解递归或利用结构分解以产生生成函数之间的关系进行分析的基础。

470 ~ 471

随机图具有大量有趣的性质。关于这一课题人们已经写出了许多著作，这是一个十分活跃的研究领域。（如果要了解该领域的概观，可参见Bollobás[3]。）随机图的分析是以“概率方法”

为中心的，其焦点问题并不在于准确地列举出所有图的性质，而是要建立起复杂参数与易于处理的参数之间关系的适当的不等式。目前有许多处理图的重要基本算法，在有关这些算法的分析文献中也有许多例子。不同的随机模型适用于不同的应用，使得分析按着我们正在研究的适合于多种情况的路线来进行。学习随机图的性质在算法分析中是一个富有成效的研究领域。

我们以随机映射这个论题来结束本书内容比较合适，其中有多种原因：随机映射推广了基本而又普遍的结构（排列和树），本书中我们曾为这些结构付出过如此之多的关注；随机映射在随机数发生器和随机序列的使用中具有直接的实际意义；对随机映射的分析展示了符号计数法以及我们所使用的其他工具所具有的能力、简洁性和实用性；随机映射代表了我们朝着研究随机图的方向所迈出的第一步（例如，参见Janson、Knuth、Luczak和Pittel[19]），它们是基本的、能被广泛应用的结构。我们希望本书所涉及的基本工具和方法能为读者提供浓厚的兴趣和专门的技能，以解决这些及在将来出现的算法的分析中的其他问题。 472

参考文献

1. M. Abramowitz and I. Stegun. *Handbook of Mathematical Functions*, Dover, New York, 1970.
2. E. Bach. "Toward a theory of Pollard's rho method," *Information and Computation* **30**, 1989, 139–155.
3. B. Bollobás. *Random Graphs*, Academic Press, London, 1985.
4. R. P. Brent and J. M. Pollard. "Factorization of the eighth Fermat number," *Mathematics of Computation* **36**, 1981, 627–630.
5. B. Char, K. Geddes, G. Gonnet, B. Leong, M. Monagan, and S. Watt. *Maple V Library Reference Manual*, Springer-Verlag, New York, 1991.
6. L. Comtet. *Advanced Combinatorics*, Reidel, Dordrecht, 1974.
7. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*, MIT Press, New York, 1990.
8. F. N. David and D. E. Barton. *Combinatorial Chance*, Charles Griffin, London, 1962.
9. W. Feller. *An Introduction to Probability Theory and Its Applications*, John Wiley, New York, 1957.
10. P. Flajolet, D. Gardy, and L. Thimonier. "Birthday paradox, coupon collectors, caching algorithms and self-organizing search," *Discrete Applied Mathematics* **39**, 1992, 207–229.
11. P. Flajolet and A. M. Odlyzko. "Random mapping statistics," in *Advances in Cryptology*, J.-J. Quisquater and J. Vandewalle, eds., Lecture Notes in Computer Science No. 434, Springer-Verlag, New York, 1990, 329–354.
12. P. Flajolet and R. Sedgewick. *Analytic Combinatorics*, in preparation.
13. D. Foata and J. Riordan. "Mappings of acyclic and parking functions," *Aequationes Mathematicae* **10**, 1974, 10–22.
14. G. H. Gonnet. "Expected length of the longest probe sequence in hash code searching," *Journal of the ACM* **28**, 1981, 289–309.
15. G. H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures* (second edition), Addison-Wesley, Reading, MA, 1991.

16. I. GOULDEN AND D. JACKSON. *Combinatorial Enumeration*, John Wiley, New York, 1983.
17. R. GRAHAM, D. E. KNUTH, AND O. PATASHNIK. *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.
473
18. L. GUIBAS AND E. SZEMEREDI. "The analysis of double hashing," *Journal of Computer and Systems Sciences* **16**, 1978, 226–274.
19. S. JANSON, D. E. KNUTH, T. LUCZAK, AND B. PITTEL. "The birth of the giant component," *Random Structures and Algorithms* **4**, 1993, 233-358.
20. D. E. KNUTH. *The Art of Computer Programming. Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, MA, 1968.
21. D. E. KNUTH. *The Art of Computer Programming. Volume 2: Seminumerical Algorithms*, Addison-Wesley, Reading, MA, 1969.
22. D. E. KNUTH. *The Art of Computer Programming. Volume 3: Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.
23. V. F. KOLCHIN. *Random Mappings*, Optimization Software, New York, 1986.
24. V. F. KOLCHIN, B. A. SEVASTYANOV, AND V. P. CHISTYAKOV. *Random Allocations*, John Wiley, New York, 1978.
25. G. LUEKER AND M. MOLODOWITCH. "More analysis of double hashing," in *Proceedings 20th Annual ACM Symposium on Theory of Computing*, 1988, 354–359.
26. J. M. POLLARD. "A Monte Carlo method for factorization," *BIT* **15**, 1975, 331–334.
27. R. SEDGEWICK. *Algorithms*, 2nd edition, Addison-Wesley, Reading, MA, 1983.
28. R. SEDGEWICK, T. SZYMANSKI, AND A. YAO. "The complexity of finding cycles in periodic functions," *SIAM Journal on Computing* **11**, 1982, 376–390.
29. J. S. VITTER AND W. CHEN. *Design and analysis of coalesced hashing*, Oxford University Press, New York, 1987.
30. J. S. VITTER AND P. FLAJOLET, "Analysis of algorithms and data structures," in *Handbook of Theoretical Computer Science A: Algorithms and Complexity*, J. van Leeuwen, ed., Elsevier, Amsterdam, 1990, 431–524.
474

索　引

索引中的页码为英文原书页码，与书中边栏页码一致。

A

Abel's binomial theorem（阿贝尔二项式定理），129, 450
Absolute error (asymptotics)（绝对误差（渐近性）），167
Additive parameter (in tree)（（树中的）可加参数），251-254
Aho-Corasick algorithm (string)（Aho-Corasick算法（串））400-401
Alcohol（酒精），281
Algebraic geometry（代数几何），456
Alphabet（字母表）：参见String; Word
Ambiguity (formal language)（二义性形式语言）：参见Unambiguous
Analysis of algorithms（算法分析），1-4, 471-472
Ancestor node (in a tree)（（树中的）祖先结点），223
Arithmetic expression（算术表达式），233-236
Arrangement（排列），432
Assembly language（汇编语言），16
Asymptotic analysis（渐近分析）：参见Coefficient asymptotics; Euler-Maclaurin summation; Laplace method; Linear recurrence;Stirling's formula
Asymptotic approximation（渐近逼近），23, 153-155, 217-218
Asymptotic expansion（渐近展开），24, 153-159, 161-180
  finite（有限的），163
Asymptotic notations（渐近记法）
  o（小o-记法），155-159
  O（大O-记法），4-5, 155-159
  Ω（大Ω-记法），4-5
  Θ（大Θ-记法），4-5
  ~（~-记法），155-159
Asymptotic scale（渐近级），162
Asymptotic series（渐近级数），162
Autocorrelation (of a string)（（串）自相关），372-377
Automatic analysis of algorithms（算法自动分析），125-126, 410
Automaton（自动机）
  finite-state (FSA)（有限状态），361
  Kleene's theorem（Kleene定理），379
  and regular expression（与规则表达式），379
  and string searching（与串查找），382-385, 400-401
Average（平均）：参见Expected value
Average-case analysis（平均情形分析），12-14
AVL tree (Adel'son-Vel'skii and Landis（AVL树），290-292, 294

B

B-tree（B树），290-292
Bach's theorem (on quadratic maps)（（关于二次映射的）Bach定理），456, 469
Balanced tree（平衡树），239, 290
Ballot problem（选票问题），128, 268-272, 390-391
Balls（球），414, 416-421, 431-432, 437-445
Batcher's odd-even merge（奇偶合并）：参见Odd-even merge
Bell curve（Bell曲线）：参见Normal distribution
Bell number（Bell数），429
Bernoulli distribution (binomial distribution)（伯努利分布（二项分布）），194, 361, 413-414
Bernoulli number (B_N)，（伯努利数）
  asymptotics（渐近性），169
  definition（定义），143, 145-146
  in summations（在求和中），181-187
  values（值），182
Bernoulli polynomial ($B_m(x)$)（伯努利多项式）
  definition（定义），143, 146
  in summations（在求和中），181-187
Bernoulli trial:（伯努利试验）
BGF (Bivariate generating function)（二元生成函数）：参见Cumulative analysis;Generating function; Moments
Binary search（折半查找），64-65
Binary search tree（二叉查找树）
  additive parameter（可加参数），251-254
  average path length（平均路径长），246-248
  construction（构造），246-249

decomposition of permutations（排列的分解），326
definition（定义），237
frequency（频率），316-317
and heap-ordered tree（与堆序树），314-317
height（高），260-261
insertion program（插入程序），238
leaves（树叶），253-254, 331-334
node types（结点类型），331-334
and permutation（与排列），313-317
and Quicksort（与快速排序），248-249
search cost distribution（查找开销分布），249-250
search program（查找程序），237
and trie（与trie树），397
Binary tree（二叉树）
additive parameter（可加参数），251-254
average path length（平均路径长），241-245
counting（计数），113-116, 224
definition（定义），113, 222-223, 276
and general tree（与一般树），227
generating function（生成函数），114, 127-128, 224, 226, 386-387
height（高），228-230, 255-256
internal/external node（内部/外部结点），113, 222-223
internal/external path length（内部/外部路径长），228-229, 241-246
leaves（树叶），139-141, 228-229, 253- 254
rotation correspondence（旋转对应），227
and set of strings（与串集），392-396
table（表），113
and trie（与trie树），392-396
Binomial（二项式）
convolution（卷积），88-92
sum（和），88-92
transform（变换），105-106
Binomial coefficient（二项式系数），102, 143-145
asymptotics（渐近性），169
bivariate asymptotics（二元渐近性），194-202
definition（定义），102
normal approximation（正态逼近），196-198
Poisson approximation（泊松逼近），198-202
Binomial distribution（二项分布），117-118, 131-132, 134-135
asymptotics（渐近性），169, 194-207
and hashing（与散列），416-421, 437-445
normal approximation（正态逼近），196-198
occupancy problems（占有问题），416-421, 437-445
Poisson approximation（泊松逼近），198-202, 414
and random strings（与随机串），366-367
tails（尾部），197-198
and tries（与trie树），401-408
Binomial theorem (Newton's theorem)（二项式定理（牛顿定理）），101-102, 114, 117
Birthday problem（生日问题），188, 422-425, 432, 445, 462
Bitstring（位串），361；参见String
Bivariate asymptotics（二元渐近性），187-206
Bivariate generating function (BGF)（二元生成函数（BGF）：参见Cumulative analysis; Generating function; Moments
Bootstrapping（引导）：参见Iteration
Boyer-Moore algorithm(string searching)（Boyer-Moore算法（串查找）），408-409
BST：参见Binary search tree
Bubble sort（冒泡排序），355-356
Bytestring（字节串），361, 377, 413-414；参见String

C

Caching algorithm（高速缓存算法），430-431
Carry propagation（进位传送），71-72, 372
Cartesian product construction（笛卡儿乘积结构），118-119
Catalan distribution（Catalan分布），241-242
Catalan model (random trees)（Catalan模型（随机树），236, 240-241, 251-254, 293
Catalan number ($T_N = G_{N+1}$)（Catalan数），114, 143, 224, 226
asymptotics（渐近性），169, 174, 186
and ballot problem（与选票问题），128, 268-272, 390-391
and binary trees（与二叉树），113-116, 121
definition（定义），114
and general trees（与一般树），226-227
generating function（生成函数），114, 127-128, 224, 226, 386-387
history（历史），269-270
and symbolic method（与符号方法），121, 224, 226
table of combinatorial objects（组合学对象表），271
Catalan sums（Catalan和），207-210
Catalan tree（Catalan树）：参见Catalan model
Cayley function ($C(z) = ze^{C(z)}$)（Cayley函数），128-129
expansion of powers（幂展开），285
and labelled tree（与标号树），285
and Lagrange inversion（与拉格朗日反演），128-129
and maps（与映射），462-466

Cayley tree（Cayley树）：参见Labelled tree
Central limit theorem（中心极限定理），337
CFG（上下文无关语法）：参见Context-free grammar
CGF (Cumulative generating function)（累积生成函数）：参见Generating function; Cumulative analysis
Character (letter)（字符（字母））：参见String; Word
Characteristic polynomial (of a recurrence)（(递归的）特征多项式），48, 97
Chebyshev's inequality（切比雪夫不等式），26-27, 444-445
Child node (in a tree)（(树中）子结点），223
Chomsky and Schützenberger's theorem (formal language)（Chomsky与Schützenberger定理（形式语言）），378-379, 387-388
Clustering (hashing)（聚集（散列）），446, 449
Coalesced hashing（接合散列），445
Coefficient asymptotics (from GF)（(来自生成函数的）系数渐近性），103, 213-218
  and permutations（与排列），322
  and strings（与串），369-370
  and trees（与树），281, 289-290
Coefficient notation ($[z^n]f(z)$]）（系数记法），82, 88
Coin flipping（投掷硬币）：参见Binomial distribution
Collision (hashing, occupancy)（冲突（散列，占有）），414, 423-424, 445, 448
Combinatorial algorithm（组合算法），30-31, 272-276, 279
Comparison-based sorting（基于比较的排序），299-300
Complex analysis（复分析），103, 149, 213-218, 281, 289-290
Complexity of sorting（排序的复杂性），9
Computational complexity（计算复杂性），2-5, 8-10, 77
Computer algebra（计算机代数），388-390
Connected component (map)（连通分量（映射）），457-466
Connected graph（连通图），274
Constructions（构造）：参见Symbolic method
Context-free grammar（上下文无关语法），385-392
Continuant polynomial（接续多项式），53
Continued fraction（连分数），53, 56-57
  and height of trees（与树的高），256-260
Convergence（收敛性），45-47, 84, 153, 165- 166, 214-217
  asymptotic expansion（渐近展开），153, 165-166
  power series（幂级数），84
  quadratic（二次），45-46
  radius（半径），214-217
  simple（简单），45
  slow（慢），46-47
Convolution（卷积），84-86, 88-92, 114
  Vandermonde（范德蒙卷积），104
Cost function（开销函数），133, 136-139
Coupon collector problem（赠票收藏家问题），425-429
Cumulated cost（累积开销），13, 136
Cumulative analysis（累积分析），13, 139-140, 244, 323-324, 359
Cumulative analysis in applications（应用程序中的累积分析）
  permutations（排列），323-334, 338, 345, 352-353
  string searching（串查找），364-366
  trees（树），241-254
  words（字），421-422, 438
Cumulative generating function（累积生成函数）(CGF)：参见Cumulative analysis;Generating function
Cycle detection (map)（圈检测（映射）），467-468, 480
Cycle leader（圈首），302, 311
Cycles (in permutation)（(排列中的）圈）
  distribution（分布），350-355
  maximal/minimal（极大/极小），317-323
  and permutation（与排列），122-123, 302
  set of（圈的集合），122-126, 318-319, 321-323
Cycles (in map)（映射中的）圈，457-466

D

Darboux's method (asymptotics)（Darboux方法（渐近性）），216, 281, 290
Decomposable combinatorial structures（可分解组合结构），126
Decomposition of permutations（排列的分解），323-326
Derangement（错位排列），178, 302, 434
  asymptotics（渐近性），178, 322
  definition（定义），178, 302
  generating function（生成函数），321-323
Descendant node (in a tree)（(树中的）后裔结点），223
Dictionary problem（字典问题），64-65, 236, 398-401
Difference equation（差分方程），36, 211-213
Differential equation（微分方程），99-100, 109-112
Differential equations in applications（应用中的微分方程）
  BST（二叉查找树），251-254
  Eulerian numbers（欧拉数），329, 333
  HOT（堆序树），315
  increasing subsequences（递增子序列），330
  involutions（卷积），320
  maxima（最大值），346-347
  median-of-three Quicksort（三数中值快速排序Quicksort），109-112
  Quicksort（快速排序Quicksort），99-100

Digital searching（数字查找）：参见Trie
Dirichlet generating function (DGF)（狄利克莱生成函数），147
Discrete sums（离散和）
asymptotics（渐近性），177-180
table（表），42
Disjoint union（不相交并），118-119
Distributed algorithms（分布式算法），406-408
Distribution (probability)（分布（概率）），25-28
Distributional analysis（分布分析），13, 25-28
Divergent series (asymptotics)（发散级数（渐近性）），165-166
Divide-and-conquer（分治）
algorithm（算法），6
function（函数），73-75
periodicity（周期性），62-65, 74
recurrence（递归），62-77
theorem（定理），73, 76
Divisor function（除数函数），148
Dollar, changing a（美元，找零），116, 161
Double hashing（双散列），446, 448
Dynamic process（动态过程），422, 430

E

Elimination（消元法）：参见Groebner basis
Empty (list, urn in hashing)（空（表，散列中的瓮）），439, 454
Error term (in asymptotic expansion)（（在渐近展开式中的）误差项），162
Euler and Segner (on Catalan numbers)（欧拉和Segner（关于Catalan数）的工作），269-270
Euler's constant (γ)（欧拉常数），23, 184
Euler equation（欧拉方程），110-112
Euler-Maclaurin constants（欧拉-麦克劳林常数），184-187
Euler-Maclaurin summation（欧拉-麦克劳林求和），23, 154, 177, 180-187
and bivariate asymptotics（与二元渐近性），203-210
and Catalan sums（与Catalan和），203-210
discrete form（离散形式），184-187
and height of trees（与树的高），256-260
general form（一般形式），180-184
Eulerian number（欧拉数），328-329, 333
Exp-log transformation（指数-对数变换），174-175, 190
Expectation (of a discrete variable)（（离散变量的）数学期望），129
and PGF（与概率生成函数），129-132
Exponentially small term（指数级小项），157-158, 191, 204
Expression evaluation（表达式求值）：参见Register allocation
External node（外部结点）
binary tree（二叉树），113, 222-223
trie（trie树），392-396
External path length (binary tree)(外部路径长(二叉树))，113, 222-223
Extremal parameter (permutation)（极值参数（排列）），355-358

F

Faà di Bruno's formula（Faà di Bruno公式），106
Factorial powers（阶乘幂），145
Factorization (of integers)（（整数的）因子分解），414, 466-471
Fibonacci number (F_N)（斐波那契数），38, 50-51, 53, 56-57, 94, 120
asymptotics（渐近性），169
definition（定义），38, 143
generalized（广义的），161, 369-370
generating function（生成函数），94, 104-105, 143
golden ratio（黄金分割），51, 169, 371
and strings（与串），120, 369-370
Fibonacci polynomial（斐波那契多项式），258
Final state (automaton)（最后状态（自动机））：参见Automaton
Finite function（有限函数）：参见Map
Finite-state automaton (FSA)：（有限状态自动机）：参见Automaton
First correspondence（第一对应），325, 421-422
Floyd's cycle detection algorithm (map)（Floyd圈检测算法（映射）），467-468
Foata's correspondence (permutations)（Foata对应（排列）），303, 351
Footnote（脚注），452
Forest（森林），128, 225, 273, 285
Formal language（形式语言：）：参见Language
Formal power series（形式幂级数），84
Fractal（不规则碎段），63, 69-70, 78
Fractional part ({x})（小数部分），65, 180, 403-405
Free tree（自由树）：参见Unordered tree
Frequency (analysis of algorithm)（频率（算法分析）），11-12, 16
Frequency of letters（字母的频率）
table（表），435
in words（字中的），413-415, 431-432
Fringe analysis（边缘分析），44
FSA：参见Finite-state automaton

Full table (hashing)（满表（散列）），452
Function (finite)（(有限）函数）：参见Map
Functional equation（函数方程），106-109, 114, 149, 211-213, 385-391, 402-406
Functional equations in applications（应用中的函数方程）
binary search tree（二叉查找树），248, 261
context-free grammar（上下文无关语法），385-391
in situ permutation（在原位排列中），354
radix-exchange sort（基数-交换排序），212-213
trees（树），243-246, 248, 263-264, 284-286
tries（trie树），211-213, 402-406
2-ordered permutation（2-有序排列），343
unordered trees（无序树），281
Fundamental correspondence（基本对应）：参见Foata's correspondence

G

Gambler's ruin（赌徒破产），268-272, 380
Gamma function (of Euler)（(欧拉的）伽马函数），216
General tree（一般树）：参见Tree
Generalized harmonic number ($H_N^{(2)}$)（广义调和数），87, 141-142, 187, 250, 344
Generating function (GF)（生成函数），81-82, 148-150
bivariate (BGF)（二元（二元生成函数）），132-142
coefficient（系数），82-87
coefficient asymptotics（系数渐近性），213-218
cumulative (CGF)（累积的（累积生成函数）），13, 136-138
expansion（展开），101-104
exponential (EGF)（指数（指数生成函数）），88-92
exponential (operations)（指数（运算）），88
exponential (table)（指数（表）），88
linear recurrence（线性递归），92-99
ordinary (OGF)（常规（常规生成函数）），82-87
ordinary (operations)（常规（运算）），84-86
ordinary (table)（常规（表）），83
probability (PGF)（概率（概率生成函数）），129-132
and recurrences（与递归），92-100
singularities（奇异性），103, 216-218
symbolic method（符号方法），118-126
transformation（变换），104-106
GF：参见Generating function
Golden ratio ($\phi = (1+\sqrt{5})/2$)（黄金分割），51, 169, 371
Graphs（图），216, 272-286, 310, 471-472
Groebner basis (elimination)（Groeber基（消元法）），387-391

H

Harmonic number（调和数），17-18, 23, 143
asymptotics（渐近性），169, 184-187
generalized ($H_N^{(2)}$)（广义的），87, 141-142, 187, 250, 344
generating function（生成函数），86, 91
Hash function（散列函数），414
Hashing algorithms（散列算法），413-416
analytic results（解析结果），453
collision（冲突），423-424, 445
comparative table（比较表），453
dynamics（动力学），422, 430
empty list（空表），439
linear probing（线性探测），445-446, 449-453
longest list（最长表），436
open addressing（开放定址法），445-453
randomness assumption（随机性假设），416
separate chaining（分离链接法），414-416, 437-445
and trie（与trie树），416
uniform hashing（均匀散列），446-448
Heap-ordered tree (HOT)（堆序树），314-317
decomposition of permutations（排列的分解），326
equivalence with BST（与二叉查找树的等价性），316-317
node types（结点类型），331-334
Height (of a tree)（树的高）
definition（定义），228-229
in binary search trees（在二叉查找树中），260-261
in binary trees（在二叉树中），255-256
in general trees（在一般树中），256-260, 380-381
in random walk（在随机途径中），380-381
table (expectations)（表（期望）），264
Height-restricted tree（高受限树），290-293
Horizontal expansion (of a BGF)（(二元生成函数的）水平展开），135-138
Horse kicks (in the Prussian Army)（(在普鲁士军队中）马踢死人），199
HOT：参见Heap-ordered tree
Huffman tree（哈夫曼树），266, 398

I

Implicit function（隐函数），127
Increasing subsequence (permutation)（递增子序列（排列）），304, 330-331
Initial state (automaton)（初始状态（自动机））：参见Automaton
Infix representation（中缀表示法）：参见Inorder traversal
Information retrieval（信息检索），413
Information theory（信息论），266

Inorder traversal (of a tree) ((树的) 中序遍历), 231-233, 267, 332
Input, model (输入, 模型), 12, 29
Input, random (输入, 随机), 12-13, 19
Insertion sort (插入排序), 334-339
In situ permutation (rearrangement) (在原位排列中 (重新排列)), 307-308, 350-354
Integer factorization (整数分解), 466-471
Integer partitions (整数分拆), 214
Integrals (and sums) (积分 (与和)), 179-180
Integration factor (differential equation) (积分因子 (微分方程)), 100
Internal node (内部结点)
  binary tree (二叉树), 113, 222-223
  trie (trie树), 392-396
Internal path length (内部路径长)
  definition (binary tree) (定义 (二叉树)), 228-229
Inverse permutation (逆排列), 303, 312-313
Inversion (of a function) ((函数的) 逆), 127
  expansion (展开), 176
  Lagrange (拉格朗日): 参见Lagrange inversion
Inversion (in permutation) ((排列中的) 逆序)
  and bubble sort, 355-356 (与冒泡排序)
  definition (定义), 301
  distribution (分布), 336-339
  and insertion sorts (与插入排序), 334-343
  maximum (最大值), 355-356
  table (表), 301, 311-312, 334-339, 343-347, 355-356
Involution (permutation) (对合 (排列)), 303, 434
  enumeration (枚举), 317-320
Iteration (迭代), 41, 56, 73-74, 108, 213

K

Key (关键字), 20, 236, 414-416
Kleene's theorem (formal language) (Kleene定理 (形式语言)), 379
KMP: 参见Knuth-Morris-Pratt algorithm
Knuth (and analysis of algorithms) (Knuth (与算法分析)), 413, 449, 452
Knuth-Morris-Pratt algorithm (KMP) (Knuth-Morris-Pratt算法), 366, 382-385, 400-401, 408-409
Kraft equality (Kraft不等式), 230
Kruskal's algorithm (Kruskal算法), 275-276

L

Labelled cycle construction (标号圈结构), 462
Labelled object (标号对象), 89, 121-122
  symbolic method (符号方法), 121-126
Labelled product construction (标号乘积结构), 123-124
Labelled set construction (标号集合结构), 122-126
Labelled tree (Cayley tree) (标号树 (Cayley树)), 128-129
  enumeration (枚举), 282-286
  enumeration (table) (枚举 (表)), 283
  and map (与映射), 462-466
Lagrange inversion (拉格朗日反演), 126-129
Lagrange inversion in applications (应用中的拉格朗日反演)
  height of trees (树的高), 256-260
  labelled tree (标号树), 285-286
  maps (映射), 462-466
  t-ary tree (t-叉树), 288
  ternary tree (三叉树), 127-128
Lambert series (Lanbert级数), 148
Language (set of strings) (语言 (串的集合)), 362, 410-411
  context-free (上下文无关), 385-392
  regular (规则), 377-381
Laplace method (for sums) (拉普拉斯方法 (求和)), 155, 202-210
Laplace method in applications (应用程序中的拉普拉斯方法)
  height of trees (树的高), 256-260
  increasing subsequence (递增子序列), 330-331
  involutions (对合), 319
Laplace transform (拉普拉斯变换), 91-92
Largest correspondence (最大对应), 325, 421-422
Last correspondence (最后对应), 325, 345, 421-422, 428
Lattice path (格路径), 268-272, 339-343, 380, 390-391
Lattice representation (permutation) (格表示 (排列)), 312-313
Leader election (领导人选举), 406-408
Leaf (树叶)
  definition (binary tree) (定义 (二叉树)), 139, 228-229
  definition (general tree) (定义 (一般树)), 228
  expectation (trees) (期望 (树)), 139-140, 253-254
  expectation (binary search tree) (期望 (二叉查找树)), 253-254, 331-334
Left-to-right maximum/minimum (in permutation) ((在排列中) 左向右最大/最小: 参见Maximum
Letter (character) (字母 (字符): 参见String; Word
Level (of a node in a tree) ((树中结点的) 层次), 228
Level order traversal (层序遍历), 233

L'Hôpital's rule（洛必达法则），160
Limiting distribution（极限分布），26-27
Linear probing (hashing)（线性探测（散列）），445-446, 449-453
Linear recurrence（线性递归），40-41
  asymptotics（渐近性），159-161
  generating function（生成函数），92-99
Linear recurrence in applications（应用程序中的线性递归）
  fringe analysis（边缘分析），44
  height of tree（树的高），258
  run in string（串中的游程），369-370
Linked list (in hashing)（(在散列中）链表），414-416
List (in hashing)（(在散列中）表），414-416
Longest list (hashing)（最长表（散列）），436
Longest run (in string)（(在串中）最长游程），372-373
Lower bound (complexity)（下界（复杂性）），9

M

Map（映射），413-414
  connected component（连通分量），457-466
  cycles（圈），457-466
  image cardinality（象的基数），454-455
  main property table（主要性质表），465
  path length（路径长），457-466
  random（随机），454, 467-471
  and random number generators（与随机数发生器），455-457
  rho length（ρ长），459-461
  and tree（与树），459
  and word（与字），454
Maximum (left-to-right maximum in permutation)（最大值（排列中的左向右最大值））：参见Minimum
Mean（平均值）：参见Expectation
Median-of-three Quicksort（三数中值快速排序Quicksort），21, 59, 109-112
  recurrence（递归），21
  theorem（定理），109-112
Mellin transform（Mellin变换），404, 410
Memory management（内存管理），430-431, 445
Mergesort（归并排序），5-8
  program（程序），7
  recurrence（递归），7, 37, 62-64, 66-67
  theorem（定理），66-67
Middle square generator（平方取中发生器），456
Minimum (left-to-right minimum in permutation)（最小值（排列中的左向右最小值））
  definition（定义），301
  distribution（分布），343-347
  and selection sort（与选择排序），348-349
Model（模型）：参见Binary search tree model; Catalan model; Random map model; Random permutation model; Random string model; Random trie model
Moments (of a distribution)（(分布的)矩），13, 130
  and BGF (table)（与二元生成函数（表）），138
  horizontal computation（水平计算），136-138
  and PGF（与概率生成函数），130
  vertical computation（垂直计算），138, 139
Motzkin number（Motzkin数），289
Multiset construction（多重集结构），121, 280

N

Newton series（牛顿级数），149
Newton's algorithm（牛顿算法），45-46
Newton's theorem (binomial theorem)（牛顿定理（二项式定理）），101-102, 114
Nonplane tree（非平面树）：参见Ordered tree
Nonterminal symbol（非终止符号）：参见Context-free grammar
Normal approximation（正态逼近）
  and analysis of algorithms（与算法分析），207-210
  binomial distribution（二项分布），196-198, 414
  and hashing（与散列），437-445
  and height of trees（与树的高），196-198
Normal distribution（正态分布），155, 169, 195
Null object (ε)（空对象），120
Number representation（数表示法），64, 78-79

O

o-notation (o)（小o-记法），155-159
O-notation (O)（大O-记法），4-5, 155-159, 171-177
Occupancy problems（占有问题），413-414, 416-421, 431-432, 437-445：参见Hashing algorithms; Words
Occurrence (of a pattern)（(模式的）发生），362-366
Odd-even merge（奇偶归并），207-210
Omega-notation(Ω)（Ω记法），4-5
Open addressing hashing（开放定址散列法），445-453
Ordered combination（有序组合），432
Ordered tree（有序树）：参见Unordered tree
Oriented tree（有向树），276-277
Oscillation（震荡）：参见Periodicity

P

Pachinko machine（弹球盘机），445
Page reference (caching)（页引用（高速缓存）），430-431
Paradox（悖论）：参见Birthday problem

Parent node (in a tree) ((树中的) 父结点), 223, 273
Parse tree (of an expression) ((表达式的) 的分析树), 233
Partial fraction (部分分式), 93-94, 103, 216
Partial sum (部分和), 84-86
Path length (of a tree, of a map) ((树的, 图的) 路径长)
  average (BST model) (平均 (二叉查找树模型)), 246-248
  average (Catalan model) (平均 (Catalan模型)), 241-246
  average (map) (平均 (映射)), 457-466
  definition (map) (定义 (映射)), 457
  definition (tree) (定义 (树)), 228-229
  table (tree) (表 (树)), 263
Patricia trie (Patricia trie树), 397
Pattern (string) (模式 (串))
  autocorrelation (自相关), 372-377
  contained in strings (counting) (含于串中的 (计数)), 372-377
  multiple (多重), 400-401
  occurrence (发生), 362-366
Pattern-matching (模式匹配): 参见String searching
Peak (in permutation) ((在排列中的) 峰), 304, 314, 331-334
Periodicity (周期性), 62-65, 74, 97, 372, 402-405, 410
Permutation (排列), 122, 299-301
  algorithms (算法), 306-310
  and BST (与二叉查找树), 313-317
  basic properties (table) (基本性质 (表)), 305
  cumulative analysis (累积分析), 323-334, 338, 345, 352-353
  cycle length restriction (table) (圈长限制 (表)), 323
  cycles (圈), 122-126, 302, 311, 351-354
  decompositions (分解), 323-326
  extremal parameters (极值参数), 355-358
  and HOT (与堆序树), 313-317
  in situ (在原位), 307-308, 350-354
  increasing subsequence (递增子序列), 304, 330-331
  inverse (逆排列), 303, 312-313
  inversion (逆序), 301, 311-312, 334-339
  local properties (局部性质), 331-334
  longest cycle (最长圈), 317-320, 357-358
  maximum/minimum (最大值/最小值), 301, 343-349, 351
  maximum inversion (最大逆序), 356
  peak/valley (峰/谷), 304, 314, 331-334
  representation (表示法), 310-317
  rise (上升), 304, 326-330
  run (游程), 304, 326-330
  shortest cycle (最短圈), 321-323, 357-358
  singleton cycle (单圈), 321, 352-354
  table of properties (性质表), 334
Permutation model (random permutations) (排列模型 (随机排列)), 239
Perturbation method (扰动方法), 54, 60-62
PGF (Probability generating function) (概率生成函数): 参见Generating function;Moments
Planar subdivision (平面细分), 269-272
Plane tree (平面树), 276-277
Poincaré series (asymptotic expansion) (Poincaré级数 (渐近展开)), 163
Poisson approximation (泊松逼近)
  and analysis of algorithms (与算法分析), 210-213
  binomial distribution (二项分布), 198-202, 414
  and hashing (与散列), 437-445
Poisson distribution (泊松分布), 155, 169, 202, 353-354
Pole (of a rational function) ((有理函数的) 极), 159
Pollard's rho method (Pollard ρ方法), 466-471
Pólya (and Darboux's method) (Pólya (与达布方法)), 290
Polygon triangulation (多边形三角剖分), 269-272
Polynomial equation (多项式方程), 387-391
Population count function (人口计数函数), 68-70
Postfix representation (后缀表达式:): 参见Preorder traversal
Postorder traversal (of a tree) ((树的) 后序遍历), 231-233
Power series (幂级数), 84
Power sums (幂和), 146
Prefix (of a string) ((串的) 前缀), 365
Prefix representation (前缀表达式): 参见Preorder traversal
Prefix set (of strings) ((串的) 前缀集), 392-396
Preorder traversal (of a tree) ((树的) 先序遍历), 231-233
  program (程序), 233
  tree representation (树表示法), 265-269
Priority queue (优先队列), 310, 314
Probabilistic algorithm (概率算法), 28-29
Probability generating function (PGF) (概率生成函数), 129-132
  and BGF (与二元生成函数), 132-141
  and moments (与矩), 132-141
  and permutations (与排列), 336, 344
  and Quicksort (与快速排序Quicksort), 132
Probe (in hashing) ((在散列算法中的) 探测), 416, 447
Prodinger's algorithm (leader election) (Prodinger算法 (领导人选举)), 406-408
Product, Cartesian (笛卡儿乘积), 118-119
Product, labelled (标号乘积), 123-124
Program (程序), 11
Pushdown stack (下推栈): 参见Stack

Q

Q-function（Q函数）：参见Ramanujan Q-function
Quadratic map（二次映射），456, 467-471
Quadtree（四叉树），270, 288
Quicksort（快速排序），14-15
  analysis（分析），14-18
  asymptotics (table)（渐近性（表）），162
  and binary search trees（与二叉查找树），248-249
  distribution（分布），26-28, 132, 141-142
  empirical complexity（经验复杂度），19
  generating function（生成函数），99-100
  median-of-three（三数中值），21, 59, 109-112
  partitioning（划分），15
  PGF（概率生成函数），132
  program（程序），14
  recurrence（递归），17, 37, 39, 58-59
  small subfiles（小的子文件），20-21
  table（表），24
  variance（方差），27, 141-142

R

Radius of convergence（收敛半径），214-217
Radix-exchange sort（基数交换排序），22
  analysis（分析），210-213
  periodicity（周期性），402-405
  recurrence（递归），22
  and tries（与trie树），398, 402-406
Ramanujan distributions (P, Q, R)（Ramanujan分布），188-193, 299-300, 356
Ramanujan Q-function（Ramanujan Q-函数），188
  asymptotics（渐近性），203-206
  and birthday problem（与生日问题），424-425
  definition（定义），188
  and linear probing hashing（与线性探测散列法），424-425
  and maps（与映射），462-463
Random map model（随机映射模型），454-457, 465, 467-472：参见Map
Random number generator（随机数发生器），455-457, 467-468
Random permutation model:（随机排列模型），299-300, 446
Random string model（随机串模型），361, 365, 367：参见Binomial distribution; String; Word
Random tree model（随机树模型）：参见Binary search tree model; Catalan model
Random trie model（随机trie树模型），401-402
Random variable（随机变量），129-132
Random walk（随机途径），380-381：参见Gambler's ruin; Lattice path
Random word model（随机字模型），413-416：参见String; Word
Randomization（随机化），28-29, 309, 406-408, 415
Randomness assumption (hashing)（随机性假设（散列）），415
Randomness preservation（保持随机性），22
Rational function（有理函数），95, 159-161
  and patterns in strings（与串中模式），372-377
  and regular expression（与规则表达式），377-381
  and runs in strings（与串中游程），369-372
Rearrangement（重新排列）：参见In situ permutation
Record（记录），20
Record (in permutation)（（排列中的）记录）：参见Maximum
Recurrence（递归），7, 13, 17, 35-41, 78-79
  bootstrapping（自引导），54, 59-61
  change of variables（变量变换），54
  classification（分类），38-39
  first-order（一阶），41-44
  and generating functions（与生成函数），92-100, 149
  homogeneous（齐次），41
  iteration（迭代），41, 56, 73-74
  linear（线性），40-41
  linear constant coefficient（线性常系数），50-52, 92-99, 159-161
  nonlinear first-order（非线性一阶），45-47
  perturbation（扰动），54, 60-62
  quadratic（二次的），55
  repertoire（指令系统），54, 57-59
  scaling（标度），40
Recurrences in applications（应用程序中的递归）
  height of trees（树的高），255-257, 261
  other types of trees（其他类型的树），292
Recursive program（递归程序），15, 221
Register allocation（寄存器分配），55, 233-236, 262
Regular expression（规则表达式），377-381
  and automaton（与自动机），379, 384
  and generating function（与生成函数），378-379
Relative error (asymptotics)（相对误差（渐近性）），167
Repertoire method (recurrences)（取值表法（递归）），54, 57-59
Rewriting rule（重写规则）：参见Context-free grammar
Rho length（ρ长），459-461, 467-470
Rho method (of Pollard)（ρ方法（Pollard）），466-471

Riemann sum（黎曼和），180, 183
Riemann zeta function（黎曼θ函数），147-148
Rise (in permutation)（(在排列中）上升），304, 326-330
Root node (in a tree)（(树中的）根结点），223
Rooted tree（根树）：参见Tree; Unordered tree
Rotation correspondence (between binary trees and general trees)（旋转对应（在二叉树和一般树之间）），227, 262
Ruler function（标尺函数），68-70
Run (in permutation)（(排列中的）游程），304, 326-330
Run (in string)（(串中的）游程），366-373, 379
Running time（运行时间），11-12

S

Saddle point method (asymptotics)（鞍点方法（渐近性）），435
Search problem（搜索问题），236
Searching algorithms（查找算法）：参见Binary search; Binary search tree; Hashing algorithms; Preorder traversal; String searching
Selection sort（选择排序），343-350
Sentinel（标记），362, 363, 414
Separate chaining (hashing)（分离链接法（散列））：参见Hashing algorithms
Sequence construction（序列构造），119
Sequences（序列），84-86
Set construction（集合构造），121
Sets of cycles（圈的集合），122-126, 462
Sets of strings（串的集合），362, 392-396
Shellsort（希尔排序），339-343
Shift (sequences)（移位（序列）），84-86
Sim-notation (~)（Sim记法），155-159
Singleton cycle (in permutation)（(排列中的）单圈），321, 352-354
Singularity (of generating function)（(生成函数的）奇异性），103, 216-218
Singularity analysis（奇异性分析），216-218：参见Coefficient asymptotics; Darboux's method
  and trees（与树），290
Smallest correspondence (permutations)（最小对应（排列）），325
Sorting (complexity)（排序（复杂性）），9
Sorting algorithms（排序算法），4-12, 307：参见Bubble sort;Insertion sort; Mergesort; Quicksort; Radix-exchange sort; Selection sort; Shellsort
Special numbers（特殊数），142-147
  asymptotics (table)（渐近性（表）），169
  definitions (table)（定义（表）），143
Stack（栈），233, 261-262, 391-392
Standard deviation（标准差），13, 19, 26-27, 129-132, 142
Star operation on languages (*)（对语言的星操作），377
Stirling cycle number（Stirling圈数）：参见Stirling numbers of the first kind
Stirling numbers of the first kind ($\left[{n \atop k}\right]$)（第一类Stirling数），143, 145
  asymptotics（渐近性），169
  counting cycles（对圈的计数），351-352
  counting minima/maxima（对极大/极小的计数），344-349
Stirling numbers of the second kind, ($\left\{{n \atop k}\right\}$)（第二类Stirling数），143, 145
  asymptotics（渐近性），169
  counting subsets（对子集的计数），427, 429
  counting surjections（对满射的计数），427-429
  and coupon collector（与赠票收藏家），427-429
  and maps（与映射），454-455
Stirling subset number（Stirling子集数）：参见Stirling numbers of the second kind
Stirling's constant ($\sigma = \sqrt{2\pi}$)（Stirling常数），184- 187, 209
Stirling's formula ($N! \sim (N/e)^N \sqrt{2\pi N}$)（Stirling公式），166-167, 184-187
  table（表），168
  and trees（与树），169, 174, 186, 288
String（串），361-362
  autocorrelation（自相关），372-377
  bitstring（位串），361
  bytestring（字节串），361, 377, 408-409
  containing pattern (counting)（包含模式（计数）），372-377
  run（游程），368-373, 379
  and word（与字），413-414
String, set of（串的集）：参见Language; Trie
String searching（串查找），362, 409-410
  basic method（基本方法），362-366
  Boyer-Moore algorithm（Boyer-Moore算法），408-409
  KMP algorithm（Knuth-Morris-Pratt算法），382-385, 408- 409
  suffix trie（后缀trie树），399-401
  and tries（与trie树），362, 392, 398-401
Subtree（子树），113, 222-223
Successful search（成功查找），249, 414-416, 441-445, 446-448, 451-452
Suffix trie（后缀trie树），399-401, 402

Summation factor（求和因子），43, 52
Sums (and integrals)（和（与积分）），179
Surjection（满射），427-429, 433-435, 454- 455
Symbol table（符号表）：参见Dictionary problem
Symbolic method（符号方法），82, 118-126, 149
  labelled objects（标号对象），121-126
  theorems（定理），119, 124
  unlabelled objects（无标号对象），118-121
Symbolic method and constructions（符号方法与结构）
  Cartesian product（笛卡儿乘积），118-119
  context-free grammar（上下文无关语法），385-392
  labelled cycle（标号圈），462
  labelled product（标号乘积），123-124
  labelled sequence（标号序列），124-125
  labelled set（标号集），122-125
  min-rooting（最小值置根法），315
  multiset（多重集），121, 280
  regular expressions（规则表达式），378-379
  sequence（序列），119-120
  set（集），121

T

t-ary tree（t-叉树），286
t-restricted tree（t-受限树），287-290
Tail (in map)（(在映射中）尾部），457-466
Tails (of a sum, of a distribution)（(和的）尾部，（分布的）尾部），178, 197-198, 203
Taylor expansions（泰勒展开式），164-165
  table（表），164
Taylor theorem（泰勒定理），101
Telescoping (recurrence)（迭缩（递归）），78
Terminal symbol（终端符号）：参见Context-free grammar
Ternary tree（三叉树），127-128
Text searching（文本查找）：参见String searching
Theta notation (Θ)（Θ记法），4-5
Toll function（征税函数）：参见Additive parameter
Transition（转换）：参见Automaton
Translation theorem（转换定理），121, 135
Traversal（遍历）(of a tree（树的）), 231-233
Tree（树）：参见general tree
  additive parameter（可加参数），251-254
  algorithms（算法），231-240
  and arithmetic expression（与算术表达式），233-236
  average path length（平均路径长），245-246
  binary（二叉的）：参见Binary tree
  definition (general tree)（定义（一般树）），225
  enumeration（计数），226
  enumeration (table)（枚举（表）），278, 286
  height（高），228-230, 256-260, 380-381
  labelled（标号的）：参见Labelled tree
  and maps（与映射），459
  nomenclature (table)（命名法（表）），277
  ordered/unordered（有序/无序）：参见Unordered tree
  other types（其他类型），286-293
  representation（表示法），265-272
  rotation correspondence（旋转对应），227
  traversal（遍历），231-233, 265-272
Triangulation (of a polygon)（(多边形的）三角剖分），269-272
Trie（trie树），362, 392-396
  and binary search tree（与二叉查找树），397
  internal/external node（内部/外部结点），392-396, 402-406
  path length (analysis)（路径长（分析）），402-406
  and radix-exchange sort（与基数交换排序），211-213, 398, 402
  random（随机），401-402
  searching（查找），397-398
  size (analysis)（大小（分析）），402
  and string searching（与串查找），362, 392
  suffix（后缀），399-401
  sum（和），211-213
2-ordered permutations（2-有序排列），207-210, 339-343, 388-390
2-regular graph（2-正则图），216
2-3 tree（2-3树），291
  fringe analysis（边缘分析），44
  functional equation（函数方程），108
2D-tree（2D-树），270, 288

U

Unambiguous (formal language)（无二意的（形式语言））
  context-free grammar（上下文无关语法），385-388
  regular expression（规则表达式），378-379, 384
Uniform discrete distribution（均匀离散分布），131
Uniform hashing（均匀散列），446
Union-find problem（求并问题），272-273, 279
Unlabelled object（无标号对象），89, 118-119, 121-122
Unlabelled tree（无标号树）：参见Tree
Unnormalized mean（非规范化平均值）：参见Cumulated cost
Unordered tree（无序树），272-282, 285-286
Unsuccessful search（不成功查找），249, 414-416, 441-445, 446-448, 451-452

Urns（瓮），414, 416-421, 431-432, 437- 445

V

Valley (permutation)（谷（排列））：参见Peak Variance, 26-27, 129-132
and BGF（与二元生成函数），138
and PGF（与概率生成函数），129-132
Vertical expansion (of a BGF)（(二元生成函数的）垂直展开），136, 139
Void node (trie)（空结点（trie树）），392-396
Vuillemin's equivalence (BSTs and HOTs)（Vuillemin等价性（二叉查找树和堆序树）），316-317

W

Word（字），413-416
extremal statistics（极值统计），431-436
frequency restrictions (table)（频率限制（表）），435
minimal occupancy（极小占有），433-435
and map（与映射），454
maximal occupancy（极大占有），432-433, 435-436
occupancy distributions（占有分布），437-441
and occupancy problems（与占有问题），416- 421
and string（与串），413-414
Worst-case analysis（最坏情形分析），70

Z

Zeta function (of Riemann)（(黎曼的）ζ函数），147- 148