ACM/ICPC国际大学生程序设计竞赛

程序设计中的组合数学

吴文虎 主编
孙 贺 编著

清华大学出版社

ACM/ICPC国际大学生程序设计竞赛

程序设计中的组合数学

吴文虎 主编
孙 贺 编著

清华大学出版社
北京

内 容 简 介

本书系统地介绍了与程序设计竞赛有关的组合数学的基本理论和算法设计与分析的常用方法。全书共分 8 章，分别为：算法基础、组合数学初探、排列与组合、容斥原理、母函数、拟阵、贪心算法和 Pólya 定理。本书突出组合数学算法的设计与优化，从而更便于参加程序设计竞赛的读者学习组合数学。

本书可作为 ACM/ICPC 国际大学生程序设计竞赛和国际信息学奥林匹克竞赛（IOI）的培训教材，也可供从事组合数学与算法研究的人员参考。

图书在版编目(CIP)数据

程序设计中的组合数学/吴文虎主编；孙贺编著. —北京：清华大学出版社，2005.5
ISBN 7-302-10800-5

I. 程… II. ①吴… ②孙… III.组合数学—应用—程序设计 IV. TP311.1

中国版本图书馆 CIP 数据核字（2005）第 031484 号

出 版 者：清华大学出版社　　**地　　址**：北京清华大学学研大厦
http://www.tup.com.cn　　**邮　　编**：100084
社 总 机：010-62770175　　**客户服务**：010-62776969
组稿编辑：薛　阳
文稿编辑：陶萃渊
印 刷 者：北京四季青印刷厂
装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：160×230　**印张**：10.75　**字数**：180 千字
版　　次：2005 年 5 月第 1 版　2005 年 5 月第 1 次印刷
书　　号：ISBN 7-302-10800-5/TP·7180
印　　数：1～3000
定　　价：19.00 元

孙贺 1984年1月生，现就读于复旦大学。高中时参加信息学奥林匹克竞赛活动，撰写了关于信息学奥赛方面的论文数篇，发表在《信息学奥林匹克》、《数字冲浪》上，并在大学期间参与了多个省市信息学奥林匹克竞赛的命题和培训工作。2002年作为全世界年龄最小的报告人应邀在第24届国际数学家大会（ICM2002)上做15分钟报告，在ICM2002-SCC上做20分钟报告，其论文摘要入选ICM2002摘要集。2002年获得复旦大学“光华自立奖”。2004年获得复旦大学在校师生及校友的最高荣誉——复旦大学校长奖，成为该校历史上获得这一荣誉的第一位本科生。2005年秋起在复旦大学计算机科学与工程系攻读博士学位，研究方向为计算理论。

序

ACM/ICPC 是美国计算机协会（ACM）组织的国际大学生程序设计竞赛（International Collegiate Programming Contest）的简称，每年一次的赛事已成为目前规模最大和最有影响力的全球性高校间计算机学科竞赛。

每年度的 ACM/ICPC 从头年 9 月份开始，先进行各大洲和地区的预选赛，再从中选出一定比例的优胜者参加年度总决赛。2003—2004 年度的 ACM/ICPC 是第 28 届赛事，共有来自 75 个国家 1411 所大学的 3150 支队伍参加。经过第一阶段的预选赛，有 73 支队伍于 2004 年春天参加了总决赛。

ACM/ICPC 参赛选手必须是大学本科生，由三人组成一队共用一台计算机。这项赛事与中学生的信息学奥林匹克竞赛既有联系又有较大区别，被称为大学生的信息学奥林匹克。

目前 ACM/ICPC 已被越来越多的国内高校所认识，吸引了越来越多的队伍参赛，水平提高很快。

ACM/ICPC 的试题具有如下特点：

1．题目有些是从信息技术的实际开发过程中演变而来的，趣味性和实用性较强。

2．试题所要考查的知识范围较为全面，不仅要求参赛队员具备非常强的程序设计基础，包括高级语言、数据结构、计算方法等基础知识，还需要熟练掌握人工智能、组合数学、计算几何等较为深层次的内容。特别是知识广博和数学功底强者占优。

3．题目新颖、灵活，绝大部分没有固定解法，给参赛者留下了广阔的思维空间，这对于创造性的培养十分有益。

参加 ACM/ICPC 活动是一个增长知识、培养能力的绝好机会，竞赛中所体现出来的团队精神也是当代大学生应当推崇的。

就我指导中学生和大学生参加信息学竞赛的体会，核心是计算思维能力的训练，每一道题目都须构建数学模型，想出优化算法。考虑空间和时间的复杂度是至关重要的。一般，对问题抽象的程度越高，算法的

时空复杂度就会越低，这也就是为什么数学功底强者占优的原因。

数学是智力探究的需要。组合数学是研究“如何安排”的一门学科，在计算机解题的过程中会遇到许多涉及有限个物体（或事件）按一定的规则（或约束条件）该如何安排的问题，包括符合规则的安排是否存在？有多少种？怎样安排？怎样安排才能取得最佳效果？等等。在编程的时候，有还是没有组合数学的知识，情况会大不一样。

本书的作者孙贺曾是信息学奥林匹克的优秀选手，他在高中阶段就对算法与程序设计产生了浓厚的兴趣。在大学二年级就写出了本书的初稿，又经过比较长时间的认真推敲与充实完善，现在正式出版了。孙贺所写的这本书，数学味道更浓，借助于数学手段把问题分析得更为透彻，所给出的程序更为简捷和高效。因此，我愿意将这本书推荐给参加ACM/ICPC 的大学生看，也愿意将这本书推荐给参加 IOI（国际信息学奥林匹克）的中学生看。在这里我要说算法的确是艺术。艺术与科学是相通的，都会给人以美的感受。数学的美已在算法中得到了充分的体现，一个颇具匠心的好算法会让你拍案叫绝。体味思维艺术之美，我以为这可能是更高层次的享受。

科学思维能力的提高是成就事业的一个最重要的因素，希望本书会对你的进步起到促进作用。

清华大学计算机系教授，博士生导师
国际信息学奥林匹克中国队总教练
吴文虎
2005 年 4 月

前　言

组合数学的基本原理是在17世纪到18世纪由Leibniz、Jacob和Euler等数学家建立起来的。早期组合数学的研究来源于概率论中的相关问题。进入20世纪后，组合理论与数学其他分支的联系越来越密切，众多学者将组合学与数学其他分支的有关理论相结合，得到了许多富有意义的结果：1928年，英国数学家Ramsey提出并证明了一个关于集合论的Ramsey定理；1935年Whitney同时推广了图和矩阵的概念，引入了拟阵；1937年，Pólya将群的理论引入组合数学中，给出了一种普遍适用的计数方法——Pólya定理；1964年，Rota把数论中的麦比乌斯函数及反演公式应用于定义在一般偏序集上的二元函数构成的“结合函数”之上，引进了广义麦比乌斯函数及反演公式。如今，组合数学已成为数学中发展最为迅速的分支之一。

本书内容简介　本书的目的是尽可能详细地介绍组合数学中必不可少的核心理论，而对于基础组合学中的许多结论（尤其是与程序设计竞赛关系很小的内容）则被省略了。希望通过这样的处理，使参加程序设计竞赛的读者能够从这些核心理论的学习中掌握组合数学研究问题的方法，而不是单纯地了解理论。

大多数组合数学的书籍将重点放在定理的证明上，而本书则用大量的篇幅介绍如何用组合数学的原理来设计算法，并通过各种方法来不断地优化算法，使其具有较低的时间复杂度和较小的空间开销，同时对于一般的组合数学教材中很少涉及的拟阵理论也给予了介绍。除此之外，还用一定的篇幅介绍与理论计算机科学密切相关的*NP*-问题，SAT问题等内容。对于部分例题，本书给出了多种求解算法，并对程序在具体运行时所需要的时间加以比较。

本书阅读指南　全书共分八章。

第1章介绍与算法有关的基本理论。内容涉及：在复杂度分析中使用的基本记号。时间复杂度与空间复杂度的概念、平摊分析、*P*类与*NP*类、规约、*NP*-完全问题等。

第 2 章介绍组合数学的研究范围。除了给出组合数学研究的若干经典问题外，还介绍鸽笼原理的知识。

第 3 章介绍排列与组合的有关理论，以及排列与组合的生成算法。

第 4 章介绍容斥原理，并涉及错位排列和欧拉函数的知识。

第 5 章是母函数。这一章从二项式定理引入普通型母函数，并引入指数型母函数，在 5.3~5.5 节中介绍三个程序设计竞赛中的题目。5.6 节总结组合计数的常见方法，5.7 节通过介绍 *NPC* 问题的母函数表示，向读者展示了母函数理论更为广泛的应用。

第 6 章介绍拟阵的有关理论。6.1 节引入了拟阵的定义；6.2 节着重介绍与算法设计相关的拟阵的性质；6.3 节使用拟阵的方法，分析了若干图的贪心算法。

第 7 章作为拟阵理论的推广，对贪心算法给予了介绍。7.1 节介绍贪心算法的定义与特点；7.2 节分析一道程序设计竞赛试题；7.3 节讨论用贪心算法求近似解的有关问题。

第 8 章介绍 Pólya 定理。8.1 节介绍关于群和置换群的简单知识；8.2 节和 8.3 节分别介绍 Burnside 引理和 Pólya 定理。

附录中介绍了本书所使用的若干基本记号和与本书内容密切相关的其他数学知识，供没有系统学习过离散数学、数论与级数的读者参考。建议首先阅读此附录，可提高学习效率。

本书用若干个完整的章节来介绍近几年来在国际和国内各类程序设计竞赛中运用组合数学方法求解的典型题目，而那些在程序设计竞赛中涉及较少的关于组合数学中的有关结论，则放到了相关章节的例题中，这使得读者可以用尽可能少的时间来掌握组合学中的基本原理。书中介绍的有关内容的扩展放到了习题中，给读者留下了思考的余地。

本书习题的说明

本书的习题分为四类：

1．组合数学基础知识的训练

这一类题目主要是帮助读者巩固在本书相应章节所介绍的基础知识。这类问题在题号后标以[B]。在本书的学习过程中，建议读者独立地完成这一类习题。

2．更难一些的题目和组合数学相关知识的扩展

这一类问题主要是与书中所介绍的理论相关内容的扩展以及更难一些的题目。由于本书并不是一本介绍组合数学理论的基础教材，所以在组合数学中与程序设计竞赛关系不大但却能很好地体现组合数学研究方法的相关问题和组合数学中的一些重要结论，在这类习题中给予了介绍。这类问题在题目的序号后标以[F]。

3．算法的优化与分析

本书中部分习题要求读者改进例题中的有关算法，或者是分析算法的时间复杂度和空间复杂度。这类问题在题号后标以[D]。

4．程序设计竞赛试题

这部分题目选自世界各地的大中学生程序设计竞赛。这类问题在题号后标以[P]，并且位于每章习题的后半部分。

第 1 章介绍在本书后续章节的学习中需要用到的算法知识。所以在这一章的习题中基础性的题目标以[B]，比较难的题目标以[F]。

书中个别的习题前会有两个英文字母。一个标有[PD]的习题既要求读者根据题目的意图编写程序，同时要求做出与题目有关的算法分析。这类题目是一些很典型的问题，读者应格外引起注意。作者着重推荐的题目在题号前加以▶标记。

本书致谢

本书是在清华大学计算机科学与技术系博士生导师、国际信息学奥林匹克中国队总教练吴文虎教授的指导下完成的，他不仅仔细地审阅了本书的初稿，而且提出了许多宝贵的修改意见。复旦大学 ACM 竞赛组吴永辉老师与作者就书稿的内容进行了多次讨论。清华大学出版社的编辑在本书的出版过程中做了大量工作，使得本书更加完善。在此表示衷心的感谢。

由于作者水平有限，书中难免有疏漏和错误之处，恳请广大读者批评指正。

作者电子邮件地址：sunhe@fudan.edu.cn

孙 贺

2005 年 4 月于上海

目　录

第1章 算法基础

程序设计的核心是算法设计。在进行算法设计的时候，首先要保证算法的正确性，其次要使所设计的算法尽可能高效，即需要尽可能少的时间和空间。本章介绍算法设计与分析的基础知识，其中 1.1 节介绍算法的定义与特点；1.2 节讨论算法的时间复杂性与空间复杂性；1.3 节讨论 P 与 NP 类复杂性以及 NP-完全问题。

本章的内容不属于组合数学的范畴，之所以将它列为第 1 章，是因为算法的基础知识包括许多人所熟悉的大 O 表示法，以及一些人不太熟悉的 P 类与 NP 类，这些都是在后续章节的讨论中所经常涉及的概念。

1.1 算法

从 1946 年世界上第一台电子数字计算机 ENIAC 诞生至今，计算机科学家们所研究的范围越来越广泛。从早期的科学计算、图形与图像的显示、有真实感图形的产生，到目前处于计算机前沿领域的网络技术、虚拟现实、人工生命等课题，要通晓这里所有的分支是不现实的。但是若要问计算机技术发展中最核心的问题是什么，答案是算法设计。正如程序设计技术的先驱者 Knuth D.E 所说的那样，计算机科学就是算法的科学。

所谓算法，最通俗的解释就是解决问题的方法与步骤。比如说，来自各省的中学生参加全国信息学奥林匹克竞赛，并要从这百余名选手中选拔出最优秀的 20 人进入国家集训队。那么怎么做呢？组委会总在是比赛前一段时间约定比赛的地点，然后指定比赛前各地选手报到的时间，全国各地的选手报到后组委会组织比赛的开幕式，选手的试机，接下来是第一天比赛，第二天旅游，第三天组织第二试……这就是一个算法。其实，算法在日常生活中是无处不在的。但是在计算机科学中所提到的算法的概念又与我们平时所说的算法不同。为什么呢？那就是因为在计算机科学中的算法必须是计算机可理解的。

在计算机科学中，**算法**是指由有限指令集合中的一系列指令所组成的过程，这个过程对于一个给定的取自输入集的输入，能够经过若干步后输出结果。在上面的叙述中，涉及输入集的概念。**输入集**是指一个特定算法所能够接受的输入数据的全体。比如说对于一个分解质因数的算法而言，数字 5 是在输入集中，而英文单词 China 则不是输入集中的元素。

算法应该具有下面五个特征。

（1）有穷性：算法必须是可终止的，也就是说一个算法必须保证执行有限步之后结束。

（2）确切性：算法是每一步都应确切地、无歧义地定义。对于每一种情况，需要执行的动作都应严格地、清晰地规定。

（3）输入：一个算法必须有零个或多个输入。它们是算法开始运算前给予的量。这些输入取自于特定的对象的集合。它们可以使用输入语句由外部提供，也可以使用赋值语句在算法内给定。

（4）输出：一个算法有一个或多个输出，以反映对输入数据加工后的结果。没有输出的算法是毫无意义的。

（5）可行性：算法原则上能够精确地运行，也就是说人们用笔和纸做有限次运算后即可完成。

1.2 时间复杂度与空间复杂度

一个算法在执行的时候需要一定的时间与空间。理论上讲，可以通过消耗更多空间的办法来换取比较少的运行时间，也可以在仅有的比较少的空间的情况下使算法运行更多的时间来实现同样的功能。但是人们发现，有些问题，比如我们所熟悉的排序问题，无论怎样地挥霍空间，算法运行时所需要的时间都不可能无限地降低，在本节中我们用$\Omega(n\log n)$来表示排序算法的时间下界。也有一些问题，对于一个看似比较小的输入数据规模，计算机要计算出结果其时间花费也要若干世纪。由此可以看出，如果求解一个问题需要过量的时间或存储量，那么即使它可判定从而在理论上用计算机可以解决，在实际中它还是不可解的。

一方面，人们希望找出一个问题所需要时间的下界是什么。显然，如果排序问题的时间复杂度的下界是$\Omega(n\log n)$，那么对于一个时间复杂度为 $O(n\log n)$的排序算法而言，即使花再大的精力也只能使它的时间复

杂度降低常数倍。另外一个方面，人们希望回答是什么使一些问题很难计算，比如用最好的计算机算也要若干世纪，而使另外一些问题容易计算（这样的例子太多了）。这个问题是复杂性理论的核心。在过去的 30 年时间里，人们对这个问题进行了深入细致的研究，但是仍然不知道它的答案。迄今为止，复杂性理论的一个重要成果是：发现了一个按照计算难度给问题分类的完美体系，这将在 1.3 节中给予介绍。

上面所说的时间和空间都是一种直观的概念。它可以通过程序运行时间的快慢、所需要的数组的个数及大小来确定。在本节中，我们将对算法所需要的时间与空间在理论上进行分析。

1.2.1　基本记号

对于一个规模为 n 的输入数据，我们希望知道一个算法需要运行多长时间后得到结果，也希望知道随着 n 的增长，算法所需要的时间的增长速度如何。因为对于一个数据规模很小的输入数据来说，算法总是能够比较快地得到答案。但是，当 n 增大时算法的效率怎样？进一步，人们需要用一种方式来表示算法所需时间的增长速度。为了这个目的，在算法分析中，需要经常用一些记号来表示算法运行时间的上下界。值得注意的是，这些记号是从数学的角度加以定义的，所以我们也可以用这些记号来表示空间复杂度。

定义 1.1　设 f 和 g 是两个函数，如果存在 $c>0$，$N>0$，使得当 $n>N$ 时，$0\leqslant f(n)\leqslant cg(n)$ 成立，则称 $g(n)$ 是 $f(n)$ 的**上界**，记为 $f(n)=O(g(n))$。此时 $g(n)$ 也称为 $f(n)$ 的**渐近上界**。

与渐进上界相对应的是渐近下界，渐近下界的定义如下：

定义 1.2　设 f 和 g 是两个函数，如果存在 $c>0$，$N>0$，使得当 $n>N$ 时，$0\leqslant cg(n)\leqslant f(n)$ 成立，则称 $g(n)$ 是 $f(n)$ 的**下界**，记为 $f(n)=\Omega(g(n))$。此时 $g(n)$ 也称为 $f(n)$ 的**渐近下界**。

比较渐近上界与渐近下界的定义可以看出：渐近上界的定义 $f(n)=O(g(n))$ 描述了 $f(n)$ 的增长速度不会超过 $g(n)$ 的增长速度的常数倍；而渐近下界的定义 $f(n)=\Omega(g(n))$ 描述了 $f(n)$ 的增长速度不会低于 $g(n)$ 的增长速度的常数倍。

需要注意的是：引入渐近上界与渐近下界的定义绝不是为了比较两个函数 $f(n)$ 与 $g(n)$ 的大小，比如 $f(n)=n^2$，$g(n)=2n^2$，虽然 $f(n)<g(n)$，但

我们既可以写成$f(n)=O(g(n))$，也可以写成$f(n)=\Omega(g(n))$。

例 1.1 $f(n)=3n^2+2n+4$。因为当 $n\geqslant 4$ 时，$f(n)\leqslant 4n^2$，因此有 $f(n)=O(n^2)$。当然，写成$f(n)=O(n^3)$也是正确的。

一般情况下，我们遇到的大部分函数 $f(n)$都有一个最高次项 h，在这种情况下写成$f(n)=\Omega(g(n))$，$g(n)$是不带系数的 h。

例 1.2 在算法分析时得到的函数 $f(n)$经常含有对数。比如 $f(n)=cn\log_2 n$，这里 c 为常数。因为 $\log_b n=\log n/\log b$，所以，写$f(n)=O(n\log n)$时无须再指明底数（反正也要忽略常数因子）。

与前面两个记号类似的有如下的定义：

定义 1.3 设 f 和 g 是两个函数，如果对于任意的正数 $c>0$，存在 $N>0$，使得当 $n>N$ 时，$0\leqslant f(n)<cg(n)$成立，则记为$f(n)=o(g(n))$。

定义 1.4 设 f 和 g 是两个函数，如果对于任意的正数 $c>0$，存在 $N>0$，使得当 $n>N$ 时，$0\leqslant cg(n)<f(n)$成立，则记为$f(n)=\omega(g(n))$。

从上面的两个定义可知，当$f(n)=o(g(n))$时，则

$$\lim_{n\to+\infty}\frac{f(n)}{g(n)}=0$$

当$f(n)=\omega(g(n))$时则

$$\lim_{n\to+\infty}\frac{f(n)}{g(n)}=+\infty$$

例 1.3 还是上例$f(n)=3n^2+2n+4$。此时可以写成$f(n)=o(n^3)$。但是

$$\lim_{n\to+\infty}\frac{3n^2+2n+4}{n^2}=3$$

所以写成$f(n)=o(n^2)$是错误的。

在算法分析的时候我们还经常遇到这样的情况，即两个算法的时间效率在“同一水平”上，为了表示这种情况，引入如下记号：

定义 1.5 设 f 和 g 是两个函数，如果存在 $c_1>0$，$c_2>0$，$N>0$，使得当 $n>N$ 时，$0\leqslant c_1g(n)\leqslant f(n)\leqslant c_2g(n)$成立，则记为$f(n)=\Theta(g(n))$。

例 1.4 $f(n)=3n^2+2n+4$，取 $c_1=3$，$c_2=4$，$N=4$，则当 $n>N$ 时为，

$$0\leqslant 3n^2\leqslant 3n^2+2n+4\leqslant 4n^2$$

所以可以写为

$$f(n)=\Theta(n^2)$$

定理 1.1 设f,g 为 $\mathbb{N}\to\mathbb{N}$ 的全函数，$\lim\limits_{n\to+\infty}f(n)=+\infty$，$\lim\limits_{n\to+\infty}g(n)=+\infty$

且

$$\lim_{n\to+\infty}\frac{f(n)}{g(n)}=\beta$$

则

（1）若$\beta\in\mathbb{R}^+$，则$f(n)$，$g(n)$有相同的增长速度；

（2）若$\beta=\infty$，则$f(n)$比$g(n)$增长快；

（3）若$\beta=0$，则$g(n)$比$f(n)$增长快。

证明　（1）若$\beta\in\mathbb{R}^+$，则2β，$\frac{1}{2}\beta\in\mathbb{R}^+$，当$n$充分大时则

$$\frac{f(n)}{g(n)}\leqslant 2\beta,\ f(n)\leqslant 2\beta\cdot g(n)$$

即$f(n)=O(g(n))$；又因为

$$\frac{f(n)}{g(n)}\geqslant\frac{1}{2}\beta,\ g(n)\leqslant\frac{2}{\beta}\cdot f(n)$$

即$g(n)=O(f(n))$，因此$f(n)$，$g(n)$有相同的增长速度。

（2）因为

$$\frac{f(n)}{g(n)}\to+\infty$$

于是

$$\frac{f(n)}{g(n)}\geqslant 1$$

对于充分大的n一致地成立，因此$g(n)\leqslant 1\cdot f(n)$对于充分大的$n$一致地成立，即$g(n)=O(f(n))$；

另一方面，若$f(n)=O(g(n))$，则存在$m\in\mathbb{N}^+$，使得$f(n)\leqslant m\cdot g(n)$对于充分大的$m$一致地成立，于是

$$\lim_{n\to+\infty}\frac{f(n)}{g(n)}\leqslant m\neq+\infty$$

矛盾。

（3）因为

$$\lim_{n\to+\infty}\frac{f(n)}{g(n)}=0$$

则

$$\lim_{n\to+\infty}\frac{g(n)}{f(n)}=+\infty$$

由（2）知 $g(n)$比 $f(n)$增长快。

定理 1.2 函数 $k^n(k>0)$比任何多项式的增长速度快。

1.2.2 时间复杂度

在算法分析中，根据一个算法在实际运行时所需要的时间与空间的消耗来评价算法的优劣。而在这两个方面中，我们更多的是考查算法的时间效率。

一个算法，对于不同的输入数据规模，算法执行时所需要的时间是不同的。对一组给定规模的输入数据，算法运行时在最坏情况下所需的时间称为**最坏情况时间复杂度**，记为 $W(n)$，即

$$W(n)=\max\{T(x)||x|=n\}$$

在分析排序算法的时候，经常用到最坏情况时间复杂度的概念。比如快速排序在最坏情况下的时间复杂度为 $O(n^2)$，而归并排序在最坏情况下的时间复杂度为 $O(n\log n)$。

有一些算法，比如刚刚提到的快速排序算法，它在最坏情况下的时间复杂度为 $O(n^2)$，与冒泡排序的时间复杂度相同，但是快速排序是一种平均性能很好的算法，我们评价一个算法的性能时，更多的时候是考虑这个算法在大多数情况下的效率怎样。因为对于一个算法而言，如果我们选择这样的输入数据使算法永远在最坏情况下运行，这种情况在现实中是很少出现的。为了表示一个算法在平均情况下的性能，在此引入平均情况时间复杂度的概念。在一个给定的输入数据规模的条件下，以输入集中该规模下的各种数据作为输入，算法运行所需的平均时间称为**平均情况时间复杂度**，记为 $A(n)$，即

$$A(n)=\sum_{|x|=n}T(x)p(x)$$

其中 $p(x)$表示输入数据 x 在所有规模为 n 的输入数据中出现的概率。

1.2.3 时间复杂度的分析

给出一个算法，通常总是希望知道这个算法的时间效率如何，这就是时间复杂度分析所要解决的问题。在此我们通过例题来介绍两种比较