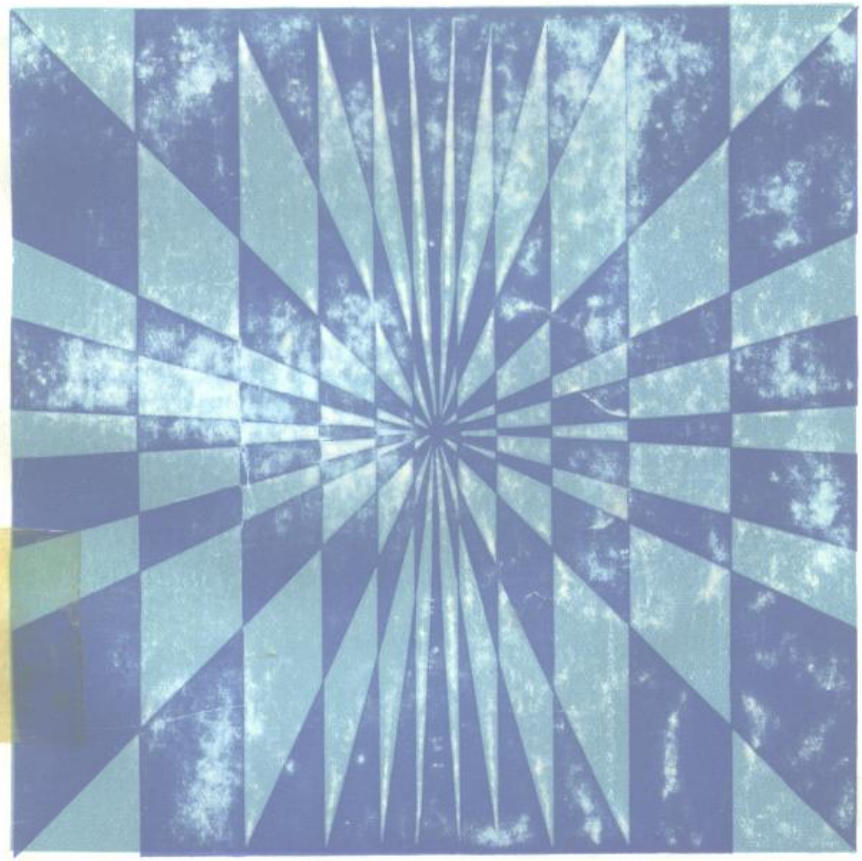
高等学校教学用书

电子计算机
算法设计与分析

陈增武　编



浙江大学出版社

高等学校教学用书

电子计算机
算法设计与分析

陈增武 编

浙江大学出版社

内 容 简 介

算法设计与分析是计算机科学的核心问题之一，随着计算机的应用日趋广泛及其性能的大幅度提高，算法的研究更显得具有重大的理论意义和实用价值。

本书共分八章。结合典型和实用的问题系统地介绍了算法设计的主要方法，讨论了算法分析的基本技术。对分类、集合运算、图论等重要领域进行了系统的研究和分析，对某些问题（如UNION—FIND树、NP完全问题等）作了较深入的讨论。

本书内容丰富，结构严谨，配有适量习题，宜用作高等院校计算机系及相关专业的大学生和研究生的教材，亦可供研究应用计算机的科研工作者和工程技术人员参考。

电子计算机算法设计与分析

陈增武　　编

责任编辑　应伯根

*　*　*

浙江大学出版社出版

浙江大学印刷厂印刷

浙江省新华书店发行

*　*　*

开本787×1092 1/32　印张11.5　字数267千字

1986年5月　第一版

1986年5月　第一次印刷

印刷1—6 000

统一书号：15337·012　　定价2.00元

前　言

算法研究是计算机科学的核心问题之一，算法的设计与分析是一个饶有趣味而又有重大实践意义的研究领域。在实用上，以快速付里叶变换为代表的快速算法的出现，大大促进了一些应用领域（如图象处理）的研究；在理论上，NP 完全问题的提出和研究，集中表现了这一领域的深度和难度。本书的目的是介绍这个领域的一些基本结果，讨论算法设计的一些常用方法，典型和常用问题的算法，较细致地探讨了各类算法的复杂程度，期望能对关心本领域的读者有所裨益。

本书共分八章。第一章介绍基本概念，扼要回顾了初等的数据结构，熟悉这方面内容的读者可以越过第二和第三节。

第二章介绍的分类算法不仅应用广泛，而且数学上的研究比较深入。为了避免和“数据结构”的内容重复，故重点放在分类算法的分析上。

第三、第四章介绍几种最常用的算法设计方法，并举了一些具有典型或实用价值的例子。

第五章至第七章介绍了几个重要领域的算法：集合运算、图算法和串匹配算法。

第八章介绍计算机难于处理的一些问题，主要是讨论 NP 完全问题。鉴于许多难处理的问题有广泛的应用背景，所以也介绍了一些近似算法。

本书的算法描述采用类似于PASCAL 语言的结构，为突出概念和方法，便读者容易阅读，在描述时允许混入易于求精的自然语句，同时一般都略去了程序的说明部分。只要知道一些

计算机高级语言和数据结构知识的读者，阅读起来是不会有什么困难的。

每一章都附有适量的习题，可供练习，其中有一些打上“*”号表示这些题目有一定的难度。

在编写过程中许多同事提供了宝贵的意见，周炳生同志为本书审稿并校订，王泽兵同志为本书准备了习题，对此作者表示衷心的感谢。

目　录

第一章　算法和初等数据结构

给出一个问题，如何为它设计一个算法？应当用什么测度去评判算法的优劣？这是计算机科学上具有理论价值和在程序设计中有重大实践意义的课题。本书将从算法设计和分析这两个侧面来讨论这些问题。

本章介绍一些基本概念，诸如什么是算法，算法的复杂度，基本分析技术等。为了以后各章的需要，还扼要地回顾了初等数据结构的基本内容。

1.1　算法及其复杂度（Algorithms and their Complexity）

1.1.1　什么是算法？

算法（Algorithm）一词是由Algorism衍生而来的。Algorism原作算术解释。它来源于一本著名的波斯数学教材：“波斯教科书”，作者是阿布·贾法·穆哈默德·伊本·穆萨·阿科瓦里茨米（Abu Ja far Mohammed ibn Mûsâ al-Khowârizmî）。较近出版的韦氏字典给它作了如下的解释，“解一确定类问题的任何一种特殊方法”。在计算机科学中，算法指的是用计算机解一个问题的精确而有效的方法。因为它的确切定义要建立在通用的计算模型的基础上（例如第八章的图灵机），在这里，我们只给出一个非形式化的描述。

简而言之，算法是能被机械地执行的动作（或称规则、指

令）的有穷集合，一个动作的一次执行称为一步，而能够用算法来解的函数或问题称为可计算函数或可计算问题。一个算法有五大特征：

1. 输入　一个算法有0个或多个输入量。这些输入量是算法所要求的初始信息，它们取自某一特定的集合；

2. 确定性　算法的每一步骤都必须有确定的意义，动作不能有二义性。例如，算法中不允许有“计算5/0”或“将6或7与x相加”这样的运算；

3. 有穷性　一个算法对任一合法输入必须在执行有穷步后终止；

4. 输出　一个算法有一个或多个输出信息，它们常是同输入信息有特定联系的量；

5. 能行性　这里指算法中的所有动作必须是相当基本的，也就是说，每一步至少在原理上能由人在有限的时间内用笔和纸来完成。进行整数的算术运算是能行运算的一个例子，而实数的算术运算则不是能行的，因为某些数值只能由无限长的十进制数展开来表示，这种数相加将与能行性这一性质违背（一般规定最基本的算术动作是二进制位运算或十进制位运算）。

算法与通常所说的计算过程有密切的联系，但并不是等同的概念。过程也要求有输入、输出、确定性和能行性，但不要求有穷性，所以不一定能终止。数字计算机的操作系统是计算过程的一个重要例子。这一过程能控制作业的执行，它使得当没有作业可用时，这一过程并不终止，而是处于等待状态，一直到一个新的作业进入为止。

算法研究的中心议题是，对于现实生活中所提出的特定问题，设计出一个求解的方法；并对方法的优劣作出评价。前者是算法设计的任务，后者则是算法分析的使命。

在本书中将学习各种被证明是有用的设计技术，用这些技术已得出了很好的算法。好的算法往往与一定的数据结构相联系，所以书中不可避免地要涉及到各种初等的和高等的数据结构。

人的思想要用各种自然语言来表达，算法也是一样，要用各种语言来表示。算法同程序的区别在于：一个计算程序往往是指用某一特定的高级或低级的计算机语言所书写的一个计算过程或算法，而一个特定的算法不必一定和一个程序有什么关系。一个算法可以有多种描述方式，如图解描述和语言（包括自然语言和计算机语言等）描述都是允许的。此外，一个计算机程序往往只能在某种计算机上执行，而一个算法，既可以编成程序在各计算机上执行，也可以用笔和纸手工地执行，甚至可以用算盘和其它工具来执行。

本书选择了一种类似于PASCAL的语言来表达算法，但为了掌握算法的核心思想和基本步骤，在前后文自明的情况下，省去了说明部分，同时允许使用自然语言表达的语句，仅要求这些语句易于逐步求精转变为PASCAL 的语句序列，以期达到能在保持PASCAL 语言的结构化程序设计风格的同时，更清晰地表达出算法的要领。

算法设计出来以后，必须证明它对所有可能的合法输入都能算出正确的答案，这一过程称之为算法确 认（algorithm validation）。确认的目的在于使我们确信这一算法将能正确无误地工作。在这一阶段，算法还不需要写成程序的形式。一旦算法得到确认以后，就可以将其写成程序并开始下一阶段的工作。这下一阶段称为程序证明（Program Proving），也叫程序验证（Program verification），目前这个领域的研究十分活跃，但还处于相当初级的阶段。另一种确定程序正确性的方法是对程序

进行测试，即在抽样数据集上执行程序，以确定是否产生错误的结果。如果有则进行修改。然而，正如 *E*.Dijkstra 所指出的那样，“调试只能指出有错误，而不能保证它们不存在错误”。从原理上说，程序的正确性证明比上千次调试有价值得多，因为它保证了程序对于各种可能的输入都能正确地工作。此外，在程序调试认为正确以后（或证明为正确以后），可以进一步测试程序执行的时间空间分布情况，为辨别算法的有效性和优化的方向提供依据。程序证明和程序调试超出了本书讨论的范围，读者可参阅有关的专门书籍和文章。

算法性能的讨论是算法分析的任务，也是算法研究的最主要的课题。用计算机执行算法时，要使用计算机的中央处理器（CPU）执行各种操作，要用存储器来存放程序和它的数据。算法分析是确定一个算法需多少计算时间和存贮空间的技术。这是一个饶有趣味而又有时是十分困难的领域，常需要运用各种数学技巧。算法分析的目的之一是对同一问题的各种实现的算法进行比较，对它们的性能作出定量的判断；另一个目的是确定算法是否存在什么性能上的限制，如难易程度，最好的下界、最坏情况和平均性态等等，给算法设计提供理论上的指导。

1.1.2 算法复杂度

对算法的评价可以有各种标准。如清晰性、可读性、易修改、易排错等，它们属于结构化程序设计的讨论范畴；在算法分析中，主要研究的则是算法在运行时所运行的时间和占用的空间。下面我们先引进几个概念。

问题的规模 n(Size of Problem)——是输入量大小的一个测度，也称为问题的大小或尺寸，它视不同的问题而有不同的含

义。如对矩阵来说是阶数，对图来说是顶点个数或边数，对多项式乘除法来说是多项式阶数，对集合的运算来说是集合中元素个数等等。

空间复杂度$S(n)$——依据其算法，编成程序后在计算机所占用的存储单元总数，其中包括程序本身的长度和它所用到的工作单元长度。显然$S(n)$是随着n的增长而增长的。我们称$S(n)$为空间复杂度（space complexity）。其极限情况称为渐近空间复杂度。

时间复杂度$T(n)$—依据其算法，编成程序后在计算机中耗费的时间。显然$T(n)$也是随着n的增长而增长的。我们称$T(n)$为时间复杂度（time complexity）。其极限情况称为渐近时间复杂度。

不同的计算机的存储单元有不同的容量，为了确定起见，我们规定算法中所讨论的基本类型的数据（如整数，实数，字符等）均能存于一个单元中（更确切地说，可以存在不超过k个单元中，k为某一常数）。这样在分析空间复杂度时，不必考虑到具体计算机的单元的大小。

程序的运行时间与下述因素有关：

1. 程序的输入量；
2. 编译的目标代码的质量；
3. 执行程序的各机器指令的性质和速度；
4. 构成程序的算法的时间复杂度。

第一个因素表明，运行时间应该是输入规模n的函数，表示为$T(n)$。对许多程序来说，运行时间实际上是特定输入量的函数，而不仅仅是n的函数，在这种情况下我们又把它区分为两种情况。

最坏情况运行时间（复杂度）$T(n)$——规模为n的所有输

入量程序运行时间的最大值；

期望（平均）运行时间（复杂度）$T_{avb}(n)$—规模为 n 的所有输入量程序运行时间的平均值。

期望复杂度比最坏情况复杂度更难确定，通常必须作出一些关于输入分布的假设，而比较符合实际的假设在数学上又常常不易处理。我们在以后的讨论中，将着重于最坏情况，因为它较易处理而且应用广泛，但是，有最佳的最坏情况复杂度的算法不一定有最佳的期望复杂度，反之，有些算法（如快速分类）虽然最坏情况复杂度很高，而平均性态良好，实践证明是一个有效的算法。

第二，三因素表明，程序的运行时间与用来编译程序的编译器以及执行的机器有关，因此$T(n)$不能确切地表达成函数，而只能用“阶”来表示，例如：“某某算法的运行时间和n^2成正比”，在这里我们不确定出比例常数，因为它和编译器、机器以及其它因素有关。下面引进几个符号来表达这种关系。

O—$f(n)=O(g(n))$（读作$f(n)$是$g(n)$的大O）当且仅当存在两个正常数C和n_0，使得对所有的$n\geqslant n_0$，有$|f(n)|\leqslant C|g(n)|$。

Ω—$f(n)=\Omega(g(n))$（读作$f(n)$是$g(n)$的大Ω）当且仅当存在正常数C和n_0，使得对于所有的$n\geqslant n_0$，有$|f(n)|\geqslant C|g(n)|$。

θ—$f(n)=\theta(g(n))$当且仅当存在正常数C_1、C_2和n_0，使得对于所有的$n\geqslant n_0$，有

$$C_1|g(n)|\leqq|f(n)|\leqq C_2|g(n)|$$

在算法分析中通常$f(n)$和$g(n)$都是取非负值的函数。

例如，$A(n)=a_mn^m+\cdots+a_1n+a_0$是$m$次多项式，不难证明$A(n)=Q(n^m)$。同样，易于看出和式$\sum_{i=1}^{n}i=n(n+1)/2=\theta(n^2)$

$= \Omega(n^2)$，$\sum_{i=1}^{n} i^k = O(n^{k+1}) = \theta(n^{k+1})$，最后一式是从$\sum_{i=1}^{n} i^k = \frac{n^{k+1}}{K+1}$

$+ \frac{n^k}{2}$ +低次项推出的。

设$T_1(n)$和$T_2(n)$是两个程序段P_1和P_2的运行时间，$T_1(n) = O(f(n))$，$T_2(n) = O(g(n))$，则执行P_1接着执行P_2的总运行时间是$O(max(f(n), g(n))$，此外，还有$T_1(n) \cdot T_2(n) = O(f(n) \cdot g(n))$。

因为我们用类似于PASCAL的语言描述算法，故必须对它的一些基本语句的运行时间作出估计。首先，各种算术运算和逻辑运算的时间为$O(1)$（即囿于常数的时间）。赋值（数据传递）的时间亦为$O(1)$，因此，赋值语句（以后用←代替：=）

变量←表达式

的时间复杂度是计算这个表达式且把它的值赋给变量所花的时间。

对于*if*语句：if〈条件〉then〈语句〉else〈语句〉，它的时间耗费是测试表达式所需的耗费加上跟在then后面的语句或跟在else后面的语句（看执行的是那一个）的耗费的和。

while语句：while〈条件〉do〈语句〉的耗费是计算条件的时间的耗费（循环多少次就累计多少次）加上执行语句所需耗费的时间（执行多少次就累加多少次）之和。repeat语句和for语句的耗费说明相仿。

在程序中经常要调用函数或过程，我们规定调用的耗费是执行这个过程说明的语句的耗费之和，分程序语句的耗费则等于分程序中执行的各个语句的耗费的和。

前已说明，在算法中允许出现一些自然语句，它们的复杂度要根据内容来分析。例如：语句“设a是集合s的最小元

素”，其复杂度为$O(|s|)$，（$|s|$表示s中元素的个数）。语句“把n个元素的集合分类”，根据第二章比较分类的结论，它的复杂度可视作$O(n\log n)$。

为了提高算法可读性，我们允许使用非数值标号，例如goto output是允许的，此外还引进了return（表达式）和exit语句，它们的复杂度都是$O(1)$。

也许有人揣度，由于现代计算机的飞速发展，计算速度的惊人增长，算法的复杂度高一些可能不那么重要了。但事实恰恰相反，当计算机的速度更快，处理的问题更大时，复杂度所产生的影响愈益显著，从而有效算法的设计也显得更加重要。

假定我们有下述六个具有多项式和指数复杂度的算法A_1—A_6。

算法	时间复杂度
A_1	n
A_2	n^2
A_3	n^3
A_4	n^5
A_5	2^n
A_6	3^n

其中时间复杂度定义为算法处理一个大小为n的输入时所需的微秒数。因此，用A_1处理一个大小为10的输入时需要运算时间为0.00001秒。图1·1示出了它们在运算时间增长上的巨大差别。

如果新一代计算机的速度较目前计算机的速度提高100倍至1000倍，它们对解决问题的大小的影响示于图1·2中。可以看到，速度提高1000倍时，在同等时间内（例如一小时，算法A_2能使大小增加31倍多，而A_5只仅能便解的问题的大小增加绝对量10。

算法	时间复杂度	问题大小 n					
		10	20	30	40	50	60
A_1	n	0.00001 秒	0.00002 秒	0.00003 秒	0.00004 秒	0.00005 秒	0.00006 秒
A_2	n^2	0.0001 秒	0.0004 秒	0.0009 秒	0.0016 秒	0.0025 秒	0.0036 秒
A_3	n^3	0.001 秒	0.008 秒	0.027 秒	0.064 秒	0.125 秒	0.216 秒
A_4	n^5	0.1 秒	3.2 秒	24.3 秒	1.7 分	5.2 分	13.0 分
A_5	2^n	0.001 秒	1.0 秒	17.9 分	12.7 天	35.7 年	366 世纪
A_6	3^n	0.059 秒	58 分	6.5 年	3855 世纪	2×10^8 世纪	1.3×10^{13} 世纪

图1.1 几个多项式和指数时间复杂度算法运行时间的比较

在不增加速度的情况下，考察便用更有效的算法的效果。再参看图1·1，以一分钟为基础来考察，$S_1=60\times10^6$，$S_2=\sqrt{60\times10^6}=7.746\times10^3$，$S_3=\sqrt[3]{60\times10^6}=3.9\times10^2$。因此，用算法$A_2$代替$A_3$可解大将近20倍的问题，用$A_1$代替$A_2$可以解大$7.74\times10^3$倍的问题。以一小时或一天作比较时，差别就更为显著。因此，算法的渐近复杂度是算法好坏的重要测度。可以预见，随着计算机速度的提高，算法的复杂度的改进也将更为重要。

应当注意的是，在阶为$f(x)$的复杂度的定义中，常数C可

时间复杂度	在提高速度前的最大问题大小	提高速度100倍后的最大问题大小	提高速度1000倍后的最大问题大小
n	S_1	$100S_1$	$1000S_1$
n^2	S_2	$10S_2$	$31.6S_2$
n^3	S_3	$4.64S_3$	$10S_3$
n^5	S_4	$2.5S_4$	$3.98S_4$
2^n	S_5	$S_5+6.64$	$S_5+9.97$
3^n	S_6	$S_6+4.19$	$S_6+6.29$

图1.2 提高计算机速度后的效果

能很小，也可能很大。因此对较小的问题，甚至在我们感兴趣的那种大小的范围内，也许渐近复杂度较劣的算法效果更好。例如，假定算法 B_1，B_2，B_3，B_4，B_5 的时间复杂度实际是 $1000n$，$100n\log n$，$10n^2$，n^3 和 2^n，则 B_5 对 $2\leqslant n\leqslant 9$ 大小的问题最好，B_3 对 $10\leqslant n\leqslant 58$ 最好，B_2 对于 $59\leqslant n\leqslant 1024$ 最好，B_1 仅当 $n>1024$ 时才是最好的。另一方面，有少数最坏情况复杂度为指数函数的算法在实践中证明是有效算法，线性规划中的单纯形算法就是一个突出的例子，背包问题的分枝限界法也是一个成功的例子，它们都有指数时间复杂度，可是实践表明运行时间相当快，其奥秘在于这些算法(可惜为数很少)的平均性态良好。因此，在具体选择算法时，应根据算法的时间和空间复杂度，结合具体因素（问题大小，机器容量，平均性态等），选择使用最好的算法。

1.1.3 几个例子

例1.1 求数组 A 的元素的最大值