计算机算法与程序设计实践

董 东　周丙寅　编著

清华大学出版社

计算机算法与程序设计实践

Computer Algorithm and Practical Programming

清华大学出版社
北京

内 容 简 介

本书专注于综合应用各种算法思想进行程序设计以实现问题求解，其特色在于：面向实践、面向过程、面向实用.在风格上追求简单明快，并力图展示问题求解过程，而不仅仅是给出结果.书中不仅分析题目、设计算法，还按照统一的程序设计风格编程实现算法.

全书共分 13 章.第 1 章介绍算法与程序、算法复杂性分析及 ACM/ICPC 题目特点、解题原则等内容；第 2 章至第 13 章分别介绍数据结构、字符串、模拟、高精度计算、递归与分治、递推、贪心、动态规划、搜索、图论、数学和计算几何的基本知识，针对若干相应问题分析和设计算法并编程求解.

图书在版编目（CIP）数据

计算机算法与程序设计实践/董东，周丙寅编著. —北京：清华大学出版社，2010.5
ISBN 978-7-302-20807-5

Ⅰ. ①计…　Ⅱ. ①董… ②周…　Ⅲ. ①电子计算机—算法理论 ②程序设计
Ⅳ. ①TP301.6 ②TP311.1

中国版本图书馆 CIP 数据核字(2009)第 156665 号

责任编辑：汪汉友
责任校对：时翠兰
责任印制：杨　艳

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 装 者：北京鑫海金澳胶印有限公司
经　　销：全国新华书店
开　　本：185×260　　**印　张**：29.25　　**字　数**：709 千字
版　　次：2010 年 5 月第 1 版　　**印　次**：2010 年 5 月第 1 次印刷
印　　数：1～3000
定　　价：45.00 元

产品编号：032269-01

前　言

教育部计算机科学与技术教学指导委员会在《高等学校计算机科学与技术专业发展战略研究报告暨专业规范(试行)》提出"加强学生实践和动手能力的培养".接着,在《高等学校计算机科学与技术专业实践教学体系与规范》中提出计算机专业高级人才的基本学科能力包括:计算思维能力、算法设计与分析能力、程序设计与实现能力和系统分析、开发和应用能力.其中,计算思维能力主要包括形式化、模型化、抽象化能力和逻辑推演能力.

给定一个问题,能够将其进行抽象,建立模型;然后,综合运用各种知识,设计出求解问题的有效算法;最后,通过程序设计实现算法,以求解问题.这在当前计算学科教学环节中,越来越受到重视.为了强化学生对学科基本知识、基本方法、问题求解基本思想的掌握,需要通过实践教学,让学生亲身体验,在实践中理解并运用.通过实践,不仅可以培养运用各种知识解决实际问题的能力,而且还可以引导学生对未知进行探索,培养创新能力.在我们的教学实践中,通过设置"问题求解与程序设计"、"信息学竞赛"、"算法分析与设计"等课程,通过鼓励和组织学生参加 ACM 国际大学生程序设计竞赛(ACM/ICPC),以图提高学生基本学科能力和动手实践能力.

ACM/ICPC 是世界上公认的规模最大、水平最高的国际大学生程序设计竞赛.竞赛充分展示了大学生程序设计与问题求解能力,是一种行之有效的实践教学形式.作为一种全新的发现和培养计算学科顶尖学生的方式,ACM/ICPC 得到全世界各大学的积极响应.在 2008 年第 33 届 ACM/ICPC 亚洲区预选赛中,168 所高校的 1190 支队伍参加了杭州赛区的网络预选赛;137 所高校的 803 支队伍参加了合肥赛区的网络预选赛;138 所高校的 905 支队伍参加了北京赛区的网络预选赛;151 所高校的 838 支队伍参加了成都赛区的网络预选赛;152 所高校的 1145 支队伍参加了哈尔滨赛区的网络预选赛.

本书主要目的是服务于计算学科相关专业的实践教学环节,专注于问题求解.其特色在于:面向实践、面向过程、面向实用.沿着"问题→建立模型→设计算法→程序设计→测试与调试"的线索,综合应用各门课程知识.

全书共分 13 章.第 1 章介绍算法与程序、算法复杂性分析及 ACM/ICPC 题目特点、解题原则等内容;第 2 章至第 13 章分别介绍数据结构、字符串、模拟、高精度计算、递归与分治、递推、贪心、动态规划、搜索、图论、数学和计算几何的基本知识,针对若干相应问题分析和设计算法并编程求解,给出相关训练题集.

本书是在历年指导学生参加竞赛的基础上形成的.在风格上追求简单明快,并力图展示问题求解过程,而不仅仅是给出结果.书中不仅分析题目、设计算法,还按照统一的程序设计风格编程实现算法.所提供程序全部运行成功,并在 Online Judge 上 Accept.

感谢清华大学吴文虎教授审阅了写作大纲并提出宝贵意见;感谢北京大学李文新教授的支持;感谢河北师范大学孙兆豪教授为本书提出了建议,使我们受益匪浅;感谢河北师范大学王志巍副教授和郭瑞强副教授所给予的一些建议和帮助;感谢河北师范大学、河北师范大学数学与信息科学学院、河北师范大学学生处和团委对我们的实践给予了热情支持;感谢

河北师范大学 ACM/ICPC 代表队的宋彪和吴志强，他们为本书提供了部分例题和求解程序，并和作者一起讨论分析；感谢河北师范大学 ACM/ICPC 代表队的杨晓光、杨瑞龙和张斌等为本书提供了部分训练题集，并提出了一些修改建议；感谢所有为本书编写提供支持和帮助的人们.

本书适用范围较广，可以用于高等院校计算机科学与技术、信息与计算科学、数学与应用数学和物理等专业的实践教学课程，或用于相关专业研究生一年级的实践教学课程；也可以用于 ACM/ICPC 等相关程序设计竞赛的基础训练；还可以作为程序设计和问题求解兴趣爱好者的参考书籍.

本书配套电子资源可从清华大学出版社网站下载.

尽管我们竭尽所能，但水平所限，错误和不足之处在所难免. 恳请专家和读者批评指正. 可通过电子信箱 donald. ddong@gmail. com 和 b. y. zhou@163. com 联系.

董　东　周丙寅

2010 年元月

符 号 表

$\mathbf{N}$	全体自然数
$\mathbf{Z}$	全体整数
$\mathbf{Z}^+$	全体正整数
$\mathbf{R}$	全体实数
$\neg$	逻辑非
$\wedge$	逻辑与
$\vee$	逻辑或
&	按位与
|	按位或
$<<$	左移位
$>>$	右移位
true	逻辑真
false	逻辑假
NULL	空
$\llcorner\!\lrcorner$	空格符
↵	换行符
'a'	字符常量 a
"abcxyz"	字符串常量 abcxyz
$\lfloor x \rfloor$	下整数，对 $\forall x\in\mathbf{R}, n\in\mathbf{Z}, \lfloor x \rfloor=n$，如果 $n\leqslant x<n+1$.
$\lceil x \rceil$	上整数，对 $\forall x\in\mathbf{R}, n\in\mathbf{Z}, \lceil x \rceil=n$，如果 $n-1<x\leqslant n$.
$[x]_n$	下阶乘，对 $\forall x\in\mathbf{R}, n\in\mathbf{Z}^+, [x]_n=x(x-1)\cdots(x-n+1)$.
$[x]^n$	上阶乘，对 $\forall x\in\mathbf{R}, n\in\mathbf{Z}^+, [x]^n=x(x+1)\cdots(x+n-1)$.
Length(s)	实例 s 的长度，如字符串等.

目　录

第1章 基本知识

使用计算机求解问题，算法与程序是极其重要的两个方面. 其中，算法是关键，其性能从根本上决定着问题求解的效率；程序同样重要，因为一个算法再好，如果没能正确编程实现，轻则算法效率得不到体现，重则无法求解问题.

作为计算学科人才，应具备算法设计与分析能力和程序设计与实现能力. ACM 国际大学生程序设计竞赛(ACM/ICPC)[①]和相关活动，提供了一个极好的实践平台. 本章简单介绍算法与程序和 ACM/ICPC 问题求解等内容.

1.1 算法与程序

程序设计的核心是算法设计. 在进行算法设计时，首先要保证算法的正确性，其次要尽可能提高算法的效率. 本节简要介绍算法与程序的概念和特性.

1. 算法

算法(Algorithm)就是一系列的计算步骤，用来将输入数据转换成输出结果. 可以把算法理解为问题解决的方法. 一个算法也是一个有穷规则的集合，其中的规则定义了一个解决某一特定类型问题的运算序列. 算法有如下 5 个重要特性.

(1) 输入. 一个算法有零个或多个输入，即在算法开始之前，对算法最初给出的量. 这些输入取自于特定的对象集合.

(2) 输出. 一个算法有一个或多个输出，即同输入有某个特定关系的量.

(3) 确定性. 算法的每一个步骤，必须是确切定义的. 对于每种情况，有待执行的动作必须严格的和清晰的规定之.

(4) 有穷性. 一个算法必须总是在执行有穷步之后结束.

(5) 可行性. 一般来说，还期望一个算法是可行的. 这意味着算法中所有有待实现的运算必须都是基本的，即是说，它们原则上都是能够精确运行的，而且人们用笔和纸做有穷次即可完成.

“算法”的中文名称出自西汉末年(公元前 1 世纪)编纂的天文学著作《周髀算经》，其中提出了测太阳高和远的陈子测日法. 公元前 400 年至公元前 300 年间，古希腊数学家欧几里得(Euclid)提出了求两个正整数最大公约数的算法，称为欧几里得算法，它被人们认为是史上第一个算法. 20 世纪的数学家图灵(Alan Turing)和冯·诺依曼(John von Neumann)等人的思想对算法的发展起到了重要的作用.

例如，求两个正整数 x 和 y 最大公约数的算法步骤如下：

(1) 如果 $x<y$，则交换 x 和 y 的值.

(2) $r \leftarrow x \bmod y$.

---

① 简介参考附录 A.

(3) 如果 $r=0$，则算法结束，y 即为所求；否则，转(4).

(4) $x \leftarrow y$，$y \leftarrow r$，转(2).

2. 程序

程序(Program)与算法不同，它是算法采用某种程序设计语言的具体实现. 程序可以不满足算法的有穷性. 例如，操作系统是一个在无限循环中执行的程序，因而不是一个算法. 不过可以把操作系统的各种任务看成一些单独的问题，每一个问题由操作系统的一个子程序通过特定算法实现，该子程序得到输出结果后便终止.

例如，上面求两个正整数 x 和 y 最大公约数的算法可用C++ 语言实现如下.

```
//求正整数 x 和 y 最大公约数. 输入为正整数 x 和 y. 输出为正整数 x 和 y 的最大公约数.
int gcd(int x, int y)
{
    if(x<y)                                    //交换 x 和 y 的值.
    {
        int t=x; x=y; y=t;
    }

    //运算直至 x 除以 y 的余数为 0, 求得 x 和 y 的最大公约数.
    while(y!=0)
    {
        int r=x%y; x=y; y=r;
    }

    return x;                                  //返回 x 和 y 的最大公约数.
}
```

1.2 算法复杂性分析

求解同一个问题，可以设计不同算法，它们在问题规模上的性能表现一般并不相同. 算法复杂性分析的目的之一便是比较不同算法的优劣，预测算法性能随问题规模的变化情况.

算法复杂性的高低体现在运行算法所需计算机资源的多少上，所需越多，算法复杂性就越高；所需越少，算法复杂性就越低. 最重要的计算机资源是时间和空间资源，因此，算法复杂性有时间复杂性和空间复杂性两个方面.

在进行算法分析时，为了避繁就简而又不失一般性，引入了渐进复杂性分析，即从极限角度，研究算法运行时间和所需空间是如何随输入规模的无限增长而增长的. 这里，只介绍记号 O 和 Θ.

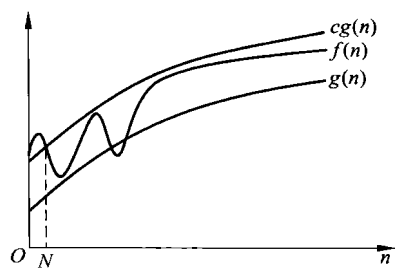

图 1-1

定义 1：设 f 和 g 是两个函数，如果存在常数 $c>0$，$N>0$，使得当 $n>N$ 时 $0 \leqslant f(n) \leqslant cg(n)$ 成立，则称 $g(n)$ 是 $f(n)$ 的渐进上界，记为 $f(n)=O(g(n))$.

O 记号使得可以只关注表达式中的最高阶项，而忽略其系数和低阶项. 在对算法性能进行精确近似的同时，使得对算法性能的表达简明，图 1-1 给出了一个示例.

定义 2：设 f 和 g 是两个函数，如果存在常数 $c_1>0, c_2>0, N>0$，使得当 $n>N$ 时 $0\leqslant c_1 g(n)\leqslant f(n)\leqslant c_2 g(n)$ 成立，则记为 $f(n)=\Theta(g(n))$.

输入规模为 n 时，算法运行所需的最长时间，称为最坏时间复杂度，即

$$T_{\max}(n) = \max\{T(x) \mid\mid x \mid = n\};$$

算法运行所需的最短时间，称为最好时间复杂度，即

$$T_{\min}(n) = \min\{T(x) \mid\mid x \mid = n\};$$

算法运行所需的平均时间，称为平均时间复杂度，即

$$T_{\text{ave}}(n) = \sum_{|x|=n} T(x)p(x).$$

其中，$|x|$ 为输入 x 的规模；$p(x)$ 为 x 在所有规模为 n 的输入中出现的概率.

在分析一个算法时，其所需空间一般分为如下两个部分.

(1) 固定部分. 其大小与输入和输出无关，主要包括存放程序代码、常量和简单变量等所用的空间.

(2) 可变部分. 主要包括大小与实例特性有关的变量和递归栈等所用的空间.

时间与空间是一个权衡关系. 很多情况下，一个好算法可以同时在时间和空间上达到最优；但更多情况下，时间与空间是矛盾的. 用时间换空间的常用方法是重复计算，用空间换时间的常用方法是预处理. 理论上，可以通过消耗更多空间，换取比较少的运行时间；也可以通过运行更长时间，换取比较少的空间消耗. 但是，有些问题，例如熟悉的排序问题，无论怎样消耗空间，算法运行时间都不可能无限降低.

1.3 ACM /ICPC 问题求解

要想顺利解决 ACM/ICPC 问题，不仅要掌握相关知识，更要有清晰灵活的头脑，并熟知一些基本技巧. 在学习积累知识的同时，还要勤于思考和实践，以培养和提高自己分析解决问题的能力，不断丰富自己的知识和经验.

1.3.1 竞赛的特点

ACM/ICPC 主要考察综合运用数学、算法和数据结构等知识解决具体问题的能力，它有如下特点.

(1) 问题难度大，强调算法的高效性，不仅要解决给定问题，而且要以最佳方式解决.

(2) 问题涉及知识面广，有很强的应用背景. 除程序设计语言、数据结构、算法分析与设计等经典计算机学科内容外，数学、人工智能、计算几何、计算机图形学和生物信息学等各学科内容也常见于问题中，尤其对数学要求非常高.

(3) 许多问题并无固定解法，需要创造性的设计求解算法. ACM/ICPC 不仅强调学科基础，还强调全面素质和能力.

(4) 问题使用英文表述，对英语要求较高.

(5) 要求 3 人共用 1 台计算机合作编程求解问题，强调团队协作精神.

概括来说就是强调算法的高效性、知识面广、数学和英语要求高、团队协作和创新精神.

ACM/ICPC 要求编程者以某种高级语言为工具，通过设计算法并编程实现来解决生活中的问题，这些问题非一般工具软件所能解决. ACM/ICPC 要求参赛者具备：

(1) 扎实的数学基础和算法知识，能够深刻认识和理解要解决的问题.

(2) 娴熟的编程技术，能够使用某种程序设计语言实现设计的算法.

(3) 较强的实践和创造能力，能够独立思考、提出质疑、拓延思路、洞悉规律，创造性的运用知识于不同的问题.

(4) 过硬的心理素质和意志，落后时不急不躁，领先时不盲目乐观.

(5) 良好的团队协作精神，3 个人能够齐心合作共用 1 台计算机进行比赛.

1.3.2 常见问题类型

求解问题用到的知识和算法往往并不单一，而是某些知识和算法的综合运用. 因此，对问题精确分类几乎是不可能的. 这里，只是按照问题考查的主要知识点和算法，将它们大致分为如下几类.

1. 基本题目

指简单易懂、算法设计容易、编程实现简单的题目.

2. 数据结构

指需要使用特定数据结构解决的题目，包括栈、队列、树、图、堆、哈希、并查集、线段树、Trie 结构和排序查找算法等.

3. 字符串操作

指以基本字符串操作(如格式化输出)为背景的题目. 通常情况下，这类题目对算法要求不是很高，而是考查基本的编程能力和技巧.

4. 模拟题

通常情况下，题目给出某个系统运作规律的描述，只要按照题目描述直接对系统运作进行模拟即可. 但对一些比较复杂的题目，却需要分析问题本质，设计效率更高的算法求解.

5. 高精度计算

指以整数和实数的高精度计算为背景的题目. 一般情况下，使用字符串数组模拟手工计算即可. 允许时，可以使用程序语言中提供的 API，例如 Java 中的大整数类.

6. 递归与分治

指主要使用递归与分治思想设计算法求解的题目. 如最邻近点对问题等.

7. 贪心

指主要使用贪心思想设计算法求解的题目. 如活动时间安排问题等.

8. 动态规划

指主要使用动态规划思想设计算法求解的题目. 如矩阵连乘问题、0－1 背包问题、最长公共子序列问题、最长不降子序列问题等.

9. 搜索

指主要使用搜索技术设计算法求解的题目. 包括枚举、广度优先搜索、深度优先搜索和启发式搜索等. 由于搜索灵活性较强，因此不易掌握.

10. 图论

指主要以图论知识和算法为背景的题目. 包括最小生成树、最短路径、拓扑排序、独立集、最大团、匹配问题、网络流和平面图等.

11. 数学

指主要使用数学知识设计算法求解的题目. 包括数论、组合数学、离散数学、几何、代数、微积分、概率论、博弈论等. 近几年，微积分、概率论和博弈论的题目逐渐增多.

12. 计算几何

指主要使用计算几何相关知识设计算法求解的题目. 包括位置关系相关问题(如点与多边形的位置关系、线段与线段的位置关系和多边形与多边形的位置关系等)、周长与面积问题(如多边形并或交的周长与面积等)、凸包问题(如平面点集的凸包)、多边形的可见核问题和三角剖分问题等.

13. 特殊问题

指需要创造新算法进行求解的题目. 这类题目一般难度较大.

1.3.3 解题的几点原则

制定一些合理原则,并遵照这些原则求解问题,可以大大提高解题效率. 下面,介绍几点基本原则.

1. 评估题目难易程度

给定题目,应养成首先对题目难易程度进行评估的习惯,切忌看到一个感觉顺手的题目就马上开始编程求解. 这个习惯在竞赛中尤为重要,因为竞赛的总时间是一定的,根据竞赛排名规则,最好的策略就是先做最简单的题目,并用最短的时间、最少的提交次数正确求解. 这样,才能有利于良好心态的保持,才能争取更多有效时间求解其余题目.

2. 警惕思维定势

作一些思维方面的准备是非常有必要的,但务必尽量保持清醒灵活的头脑,切忌思维定势. 对于一个题目,要严格按照步骤分析设计算法,使其在给定时空限制下求解问题,并考虑是否有更简捷的算法. 切忌想到一个貌似可行的方法就急着编程求解,这在大多数情况下将导致错误的结果.

避免思维定势看似简单,好像只要心里想着就行,其实不然. 例如,在某程序设计比赛中,有一道简单题目[①],概述如下:给定 $n(n_1+n_2<n\leqslant 500000000)$ 个正整数 $a_i(1\leqslant a_i\leqslant 10^8$, $i=1,2,\cdots,n)$,去掉前 $n_1(1\leqslant n_1\leqslant 10)$ 个最大的数和后 $n_2(1\leqslant n_2\leqslant 10)$ 个最小的数,求剩下各数的平均数. 比赛时,没有对内存作限制. 大多数参赛同学使用的方法是,首先存储全部 n 个正整数,然后对它们进行快速排序,最后求出中间那些数的平均数. 这可以说是平时处理类似问题时的一种思维定势. 且不说快速排序算法的时间复杂度为 $O(n\log n)$,单就其编程实现来说,对不知道使用标准库函数的低年级同学来说,也是有一点复杂度的,至少要花时间输入代码. 其实,可以先从这 n 个数中删除前 n_1 个最大的数和后 n_2 个最小的数,然后求剩余各数的平均数即可. 而删除一个数的时间复杂度为 $O(n)$,故总的时间复杂度为 $O((n_1+n_2)n)$. 这在 n 规模增大时是优于快速排序算法的,更重要的是编程实现的复杂度低,用很短时间即可实现,且更容易保证正确性. 当然,这个题目在保证时间复杂度基本为 $O((n_1+n_2)n)$ 的前提下,还可以使空间复杂度降低为 $O(n_1+n_2)$,算法描述如下.

建立两个表 L_1 和 L_2,对它们进行动态维护,使得它们最终保存的数据分别是给定的 n 个数排序后的前 n_1 个数和后 n_2 个数,同时计算出其余各数的和 S,最后计算平均值即可.

首先,读入 n_1+n_2 个数初始化表 L_1 和 L_2,使得 L_1 中的数都不小于 L_2 中的数;初始化 S 为 0. 对以后输入的每一个数 a,依次执行:

- 若 L_1 中的最小数小于 a,则将其与 a 交换;

---

① 题目来源:http://acm.pku.edu.cn/Judeg0nline/problem?id=2833.

- 若 L_2 中的最大数大于 a，则将其与 a 交换；
- 将 a 加到 S 中.

这样即可得到排序后位于中间的 $n-(n_1+n_2)$ 个数的和 S，再求平均就是答案.

3. 确定算法和数据结构

读懂题目并进行分析后，就要进一步确定求解题目所用的算法和数据结构. 这就需要具备分析算法时空复杂度的能力，熟知各种常用数据结构和算法的特性. 对一个求解算法，能够判断它是否可以在题目时空限制下求解问题. 几种常见的时间复杂度数量级排序如下，其中 c 为常数时间复杂度.

$$c < \log n < n < n\log n < n^2 < n^3 < 2^n < n!.$$

表 1-1 列出了给定问题规模 n 为 1000 时，使用不同时间复杂度数量级算法求解它所用的时间；表 1-2 列出了给定运行时间为 60ms 时，使用不同时间复杂度数量级算法求解同一问题，能够求解的问题最大规模. 通过这两个表，可以直观的观察上面各个时间复杂度数量级的性能差别.

表 1-1 不同时间复杂度数量级算法求解同一规模问题所用时间

时间复杂度数量级	n	时间(ms)
$O(1)$	1000	1
$O(\log n)$	1000	9.96
$O(n)$	1000	1000
$O(n\log n)$	1000	9960
$O(n^2)$	1000	1000000
$O(n^3)$	1000	10^9
$O(2^n)$	1000	2^{1000}
$O(n!)$	1000	uncountable

表 1-2 不同时间复杂度数量级算法用时 60ms 求解问题的最大规模

时间复杂度数量级	时间(ms)	最大的 n
$O(1)$	60	Virtually infinite
$O(\log n)$	60	6^{18000}
$O(n)$	60	60000
$O(n\log n)$	60	5000
$O(n^2)$	60	244
$O(n^3)$	60	39
$O(2^n)$	60	16
$O(n!)$	60	8

时间复杂度分析的几种基本情况及相关知识列举如下.

(1) 如果算法有 k 层循环，且第 $i(i=1,2,\cdots,k)$ 层的循环次数为 a_in+b_i（a_i 和 b_i 为常量，$a_i\neq 0$），则其时间复杂度为 $O(n^k)$.