国内算法界著名学者、计算理论学组组长
朱洪 教授 推荐

算法设计、分析与实现
从入门到精通
C、C++和Java

徐子珊 编著

- 38 个经典范例，包括渐增型算法、分治算法、动态规划算法、贪婪算法、回溯算法、线性规划算法和计算几何等算法设计和实现技巧
- 26 个国际大学生程序设计竞赛真题的详细解析及算法的应用
- 3 种主流语言（C、C++和Java）实现算法范例程序

人民邮电出版社
POSTS & TELECOM PRESS

算法设计、分析与实现从入门到精通

C、C++和Java

人民邮电出版社
北京

图书在版编目（CIP）数据

算法设计、分析与实现从入门到精通 ：C、C++和Java / 徐子珊编著. -- 北京 ： 人民邮电出版社，2010.6
ISBN 978-7-115-22837-6

Ⅰ. ①算… Ⅱ. ①徐… Ⅲ. ①电子计算机－算法设计②电子计算机－算法分析③C语言－程序设计④JAVA语言－程序设计 Ⅳ. ①TP301.6②TP312

中国版本图书馆CIP数据核字(2010)第073307号

内 容 提 要

本书第 1 章～第 6 章按算法设计技巧分成渐增型算法、分治算法、动态规划算法、贪婪算法、回溯算法和图的搜索算法。每章针对一些经典问题给出解决问题的算法，并分析算法的时间复杂度。这样对于初学者来说，按照算法的设计方法划分，算法思想的阐述比较集中，有利于快速入门理解算法的精髓所在。一旦具备了算法设计的基本方法，按应用领域划分专题深入学习，读者可以结合已学的方法综合起来解决比较复杂的问题。本书第 7 章的线性规划和第 8 章的计算几何是综合算法部分，通过学习这些内容，读者将进一步地学习更前沿的随机算法、近似算法和并行算法等现代算法设计方法和实战技巧。

本书特色是按照算法之间逻辑关系编排学习顺序，并对每一个经典算法，都给出了完整的 C/C++/Java 三种主流编程语言的实现程序，是一本既能让读者清晰、轻松地理解算法思想，又能让读者编程实现算法的实用书籍。建议读者对照本书在计算机上自己创建项目、文件，进行录入、调试程序等操作，从中体会算法思想的精髓，体验编程成功带来的乐趣。

算法设计、分析与实现从入门到精通：C、C++和 Java

◆ 编　　著　徐子珊
　责任编辑　张　涛

◆ 人民邮电出版社出版发行　　北京市崇文区夕照寺街 14 号
　邮编　100061　　电子函件　315@ptpress.com.cn
　网址　http://www.ptpress.com.cn
　三河市海波印务有限公司印刷

◆ 开本：787×1092　1/16
　印张：26
　字数：693 千字　　2010 年 6 月第 1 版
　印数：1－3 000 册　　2010 年 6 月河北第 1 次印刷

ISBN 978-7-115-22837-6

定价：49.00 元

读者服务热线：(010)67132692　印装质量热线：(010)67129223
反盗版热线：(010)67171154

前 言

算法作为数学的一个分支已经存在几百年了。然而，算法真正焕发青春得到长足的发展，还是发生在20世纪电子计算机发明的时代。随着计算机技术的广泛应用，人们越来越清楚地认识到，作为计算机科学与工程最主要的技术——程序设计，其灵魂就是解决问题的算法。

能否提供一本既能让广大读者清晰、轻松地理解算法思想，又能为读者提供算法的程序实现各种关键技术建议的书是作者一个很长时间的思考。现在摆在读者面前的这本书，就是这一思考的结果。作者希望本书在读者研习算法与程序设计时不仅是书桌前或是床头边的参考书，更多的是计算机旁的参考书。

本书先按算法设计技巧分类为渐增型算法、递归分治算法、动态规划算法、贪婪算法、回溯算法和图的搜索算法等前6章。每章对2～3个经典问题针对一种算法设计技巧，给出解决问题的算法，并分析算法的时间复杂度。笔者认为，对于初学者来说，按算法的设计方法划分章节，算法思想的阐述比较集中，有利于入门。一旦具备了算法设计的基本方法，按应用领域划分专题深入学习，读者可以应用已学的方法综合起来解决比较复杂的问题。第7章的线性规划和第8章的计算几何可以算作这部分内容。在此基础上，读者可以更进一步探讨更前沿的随机算法、近似算法和并行算法等现代算法设计方法。本书之所以按照这样的8章编排，还有一个重要原因是它们之间有一定的前后逻辑关系：第1章渐增型算法和第2章分治算法是最基本的算法，并且在这两章内容展开的同时我们还介绍了后面各章所需要的数据结构，例如第2章介绍的优先队列就是第4章讨论贪婪算法时所需要的。建议读者以本书的章节顺序研读，特别是实现的程序中也有很多是前后呼应的代码段。

算法理论对于程序设计实践的指导意义已经被大家所接受，但不管是在校学生还是已踏上职业征程的程序员，很多人对算法学习都有一种“枯燥繁难”的先入之见。如何有效地学习算法的设计与分析，是本书试图与读者一起探讨的最重要的目标之一。其实，算法都是针对具体问题的。对于要解决的问题进行深入理解与分析，是设计出正确有效的算法的前提。然而，“深入理解与分析”似乎是个模糊概念，怎样的“度”才能认为是对问题深入理解了呢？计算学科提出了一个非常重要的“标准”：简单地说，能够把一个问题输入与输出准确地提炼并表述出来，就可以认为是理解了这个问题。这样对问题的描述称为“问题的形式化描述”。一旦形式化地描述出了问题的输入与输出，也就能准确地把握住算法解决问题所需要的条件和所要达到的目标。换句话说，设计算法是要在问题的输入与输出之间架起一座通达的桥梁。本书以此为目标，无论是对正文中讨论的问题还是“动手做”题目中的问题或者是作为应用的ICPC问题，我们都给出其形式化描述。有了问题的输入与输出的数据表示，运用人们的常识、数学知识和科学知识建立起从输入到输出之间的对应关系，这就是算法。

算法可以用自然语言或是图形等工具加以描述，但要将算法作为计算机程序的设计蓝图，则需要将其无歧义地表述出来。用伪代码表述算法是迄今为止人们所使用的最有效的方法。计算机程序设计语言当然也能做到无歧义地描述算法。事实上，市场上也有以某种具体的程序设计语言为描述工具的算法书籍，但这样的图书往往要面对一个两难问题：一方面，若要使具体语言描述的算法能直接运行，则必须做诸如变量声明、用该语言所提供的方式表示各种复杂的数学表达式以及其他操作等各种技术细节工作。这样就会大大降低算法描述的可读性，妨碍读者对算法思想本身的理解与接受。另一方面，如果为提高算法描述的可读性而省略上述的各种技术细节，则仍然回到伪代码描述的起点，让读者在实现程序时有隔靴抓痒的感觉。本书由Thomas H. Cormen等编著的《算法导

论》介绍的伪代码规范来描述书中所论述的每一个算法，使其清晰明了且易于转换成计算机程序。

有了作为程序设计蓝图的算法为代码描述，本书对每一个经典算法，都给出了完整的C/C++/Java 的实现代码及测试程序。怎样把实现代码的一般性方法明明白白地告诉读者，让他们能从中得到一些写代码的方法论方面的启迪是笔者的孜孜以求。按笔者体会，根据算法写程序，需要抓住 3 个重点：其一，根据算法所处理问题的输入输出以及语言本身的技术特点设计过程（函数或方法）的参数与返回值。包括参数个数、类型，返回值类型，等等。其二，需要考虑过程中所要用到的所有变量，包括局部量和全局量。其三，由于伪代码是在一个很高的抽象层面描述算法，所以在某些情形下要考虑充分利用语言的技术特性来实现关键的操作。

本书中的所有程序源代码都已在 Java SDK1.6.2、gcc3.4.4、g++3.4.4 上调试并正确运行。读者只要建立相应的程序项目，并在项目中建立相应的文件，在文件中逐行输入代码，编译后一定能够运行。应当说明的是，本书中所说的 C 代码，指的是纯粹的 C 代码，如 turbo C 2.0 这样的编译系统都能够编译的代码。建议读者自己创建项目、文件，录入代码，调试程序，体会算法思想，体验编程成功。

本书虽然用 3 种语言分别实现每个算法，读者可独立地阅读其中一种语言的程序。如果读者阅读两种或两种以上语言的算法实现代码，那么从本书中还能体会出这 3 种语言各自的特点，也能悟出一些程序设计共通的基本方法和技术。

本书收集了一些 ICPC 的题目（包括 2007 年中国赛区中几个分赛区——南京、北京、成都等的网赛题、场赛题和几个国际决赛题）来说明书中所介绍的经典算法的应用。有朋友建议我把这些题目翻译成中文，便于读者阅读理解。但是笔者非常赞同以下的观点：一个人接受科技教育得到的最大收获，是那些能够受用一生的一般性智能工具。而在这些工具中，最重要的是自然语言、数学和计算机科学。英语作为自然语言，已成为 Internet 语言。地球因为 Internet 已成为一个“村”。用此“村言”阅读理解村中的问题（这些问题都凝结了各方文化习俗及拟题高人的智慧，读起来妙趣横生），设计出解决这些问题的算法，并将算法实现为能在计算机上运行的程序，得到问题的解，岂不是一大乐趣？

著名的数学家华罗庚先生曾经说过，读数学书若不做习题似“入宝山而空返”。笔者以为先生的意思是说思想不经过实践检验，再好的理论和技能也难以掌握，难以应用。本书在每个经典问题算法的说明和实现后以及每个 ICPC 问题之后都留有“动手做”的题目，建议读者利用相关的算法及实现的方法来解决“动手做”中相似的问题。

中国人将学习的过程分成 4 个境界：比、从、北、化。笔者理解“比”就是接受先辈留下的智慧，“从”就是跟随先辈的经验认识世界，“北”就是借鉴先辈的智慧和经验解决现在自己的问题，而“化”就是融合先辈和自己的经验产生新的知识，解决新的问题。我们把这 4 个境界缩小到算法学习的过程中：学习前人的算法设计思想，根据算法思想设计程序，利用程序解决问题，投身建设创新进取。

源程序下载地址为 www.ptpress.com.cn。

在本书写作的掩卷之时，我的内心充满了感激。感谢我的导师朱洪教授，早在 5 年前我在复旦大学做国内访问学者时就为我提供了大量的资料，鼓励我写作，几年来给了我很多的指点和帮助，并不吝为本书撰写序言。感谢福建师范大学的陈志德博士、重庆工商大学的赖涵、韦一平、杨艺、罗芬等老师对书稿的审阅。能够顺利出版成书还必须感谢人民邮电出版社的编辑对我的帮助和高效率的编辑工作。最后，还要感谢我的夫人段泰然，没有她年复一年在生活中的关怀和容忍，不可能有这本书的顺利写作。笔者才疏学浅，算法书以这样的方式来写作是一种尝试，其中必有不当与错误，盼望学界同行与广大读者批评指正。

联系邮箱为 xu_zishan@163.com 或 zhangtao@ptpress.com.cn。

作者

2010 年 5 月

目　录

第1章　集腋成裘——渐增型算法 - 1

1.1　算法设计与分析---------- 1
1.2　插入排序算法---------- 4
1.2.1　算法描述与分析----------4
1.2.2　程序实现----------6
1.2.3　应用——赢得舞伴------- 30
1.3　两个有序序列的合并算法--32
1.3.1　算法描述与分析--------- 32
1.3.2　程序实现---------- 34
1.4　序列的划分 ----------45
1.4.1　算法描述与分析--------- 45
1.4.2　程序实现---------- 46
1.5　小结----------52

第2章　化整为零——分治算法 -- 53

2.1　Hanoi塔问题与递归算法--53
2.1.1　算法的描述与分析------ 53
2.1.2　程序实现---------- 56
2.1.3　应用——新Hanoi塔游戏 - 59
2.2　归并排序算法----------62
2.2.1　算法描述与分析--------- 62
2.2.2　程序实现---------- 63
2.2.3　应用——让舞伴更开心 -- 69
2.3　快速排序算法----------70
2.3.1　算法描述与分析--------- 70
2.3.2　程序实现---------- 72
2.4　堆的实现 ----------79
2.4.1　堆的概念及其创建------ 79
2.4.2　程序实现---------- 83
2.5　堆排序 ----------88
2.5.1　算法描述与分析--------- 88
2.5.2　程序实现---------- 89
2.6　基于二叉堆的优先队列 ---94
2.6.1　算法描述与分析--------- 94
2.6.2　程序实现---------- 95
2.7　关于排序算法---------- 105
2.7.1　比较型排序算法的时间复杂度----------105
2.7.2　C/C++/Java提供的排序函数（方法）---------- 107
2.7.3　应用——环法自行车赛- 108
2.8　小结----------109

第3章　记表备查——动态规划算法---------- 111

3.1　矩阵链乘法----------112
3.1.1　算法描述与分析------- 112
3.1.2　程序实现---------- 115
3.1.3　应用——牛牛玩牌----- 121
3.2　最长公共子序列----------123
3.2.1　算法描述与分析------- 123
3.2.2　程序实现---------- 126
3.2.3　算法的应用---------- 132
3.3　0-1背包问题----------136
3.3.1　算法描述与分析------- 136
3.3.2　程序实现---------- 138
3.3.3　算法的应用---------- 142
3.4　带权有向图中任意两点间的最短路径----------144
3.4.1　算法描述与分析------- 144
3.4.2　程序实现---------- 148
3.4.3　应用——牛牛聚会----- 153
3.5　小结----------155

第4章　高效的选择——贪婪算法　156

4.1　活动选择问题----------156
4.1.1　算法描述与分析------- 156
4.1.2　程序实现---------- 158
4.1.3　贪婪算法与动态规划 - 163
4.1.4　应用——海岸雷达----- 165
4.2　Huffman编码----------166
4.2.1　算法描述与分析------- 166
4.2.2　程序实现---------- 170
4.2.3　应用——Huffman树 --- 180
4.3　最小生成树----------183
4.3.1　算法描述与分析------- 183
4.3.2　程序实现---------- 187
4.3.3　应用——北方通信网-- 196
4.4　单源最短路径问题-------197

4.4.1 算法描述与分析--------197
4.4.2 程序实现----------200
4.4.3 应用——西气东送------207
4.5 小结 ---------- 210

第5章 艰苦卓绝——回溯算法 - 211

5.1 组合问题与回溯算法 ---- 211
5.1.1 3-着色问题 ----------211
5.1.2 *n*-皇后问题 ----------214
5.1.3 Hamilton回路问题------216
5.1.4 子集和问题----------218
5.2 解决组合问题的回溯算法框架 ---------- 219
5.2.1 算法框架----------219
5.2.2 程序实现----------223
5.3 排列树和子集树---------- 235
5.3.1 子集树问题----------236
5.3.2 排列树问题----------241
5.4 用回溯算法解决组合优化问题 ---------- 245
5.4.1 算法框架----------245
5.4.2 旅行商问题----------247
5.4.3 应用 ----------253
5.5 P，NP和NP-完全问题--- 260
5.6 小结 ---------- 262

第6章 图的搜索算法 -------- 264

6.1 广度优先搜索---------- 265
6.1.1 算法描述与分析--------265
6.1.2 程序实现----------268
6.1.3 应用——攻城略地------276
6.2 深度优先搜索---------- 278
6.2.1 算法描述与分析--------278
6.2.2 程序实现----------280
6.2.3 有向无圈图的拓扑排序 ----------283
6.2.4 应用——全排序---------290
6.3 有向图的强连通分支---- 292
6.3.1 算法描述与分析--------292
6.3.2 程序实现----------295
6.3.3 应用——亲情号---------300
6.4 无向图的双连通分支---- 303
6.4.1 算法描述与分析--------303
6.4.2 程序实现----------306
6.4.3 应用——雌雄大盗------308
6.5 流网络与最大流问题 ----310
6.5.1 算法描述与分析 ------- 310
6.5.2 程序实现---------- 319
6.5.3 应用---------- 321
6.6 小结----------324

第7章 集组合优化问题之大成——线性规划 ---------- 325

7.1 标准形式与松弛形式 ----328
7.1.1 线性规划的标准形式 - 328
7.1.2 线性规划的松弛形式 - 331
7.2 单纯形算法----------334
7.2.1 单纯形算法的例子 ---- 334
7.2.2 轴转操作---------- 337
7.2.3 正规的单纯形算法 ---- 340
7.3 初始基本可行解 ----------347
7.4 应用——将组合优化问题形式化为线性规划 -------355
7.5 小结----------359

第8章 图形学基础——计算几何 ---------- 360

8.1 线段的性质----------360
8.1.1 叉积及其应用 ---------- 361
8.1.2 程序实现---------- 364
8.2 判断是否存在线段相交 ---367
8.2.1 算法描述与分析 ------- 367
8.2.2 程序实现---------- 370
8.3 求凸壳----------374
8.3.1 Graham扫描 ---------- 375
8.3.2 Jarvis行进---------- 381
8.4 求最邻近点对 ----------384
8.4.1 算法描述与分析 ------- 385
8.4.2 程序实现---------- 387
8.5 应用----------389
8.5.1 光导管---------- 389
8.5.2 最小边界矩形 ---------- 391
8.5.3 得克萨斯一日游 ------- 392
8.6 小结----------394

附录 ---------- 395

参考文献 ---------- 410

第1章

集腋成裘——渐增型算法

1.1 算法设计与分析

1. 什么是算法

众所周知，算法是程序的灵魂。只有对需要解决的计算问题有一个正确的算法，才可能编写出解决此问题的程序。所谓算法就是解决一个计算问题的一系列计算步骤有序、合理的排列。对一个具体问题（有确定的输入数据）依次执行一个正确的算法中的各操作步骤，最终将得到该问题的解。算法研究有着悠久的历史，内容极其丰富。人们对各种典型的问题研究出了很多经典的算法设计方法。例如，本书详细讨论的渐增型算法、分治算法、动态规划、贪婪策略和回溯算法等都是具有代表性的经典算法设计方法。对这些方法的学习，可以为我们在解决各种具体问题时设计出正确、高效的算法提供有益的启示。

2. 算法分析基本概念

解决一个问题，算法不必是唯一的。对表示问题的数据的不同组织方式（数据结构），解决问题的不同策略（算法思想）将导致不同的算法。解决同一问题的不同算法，消耗的时间和空间资源量有所不同。算法运行所需要的计算机资源的量称为算法的复杂性。一般来说，解决同一问题的算法，需要的资源量越少，我们认为越优秀。计算算法运行所需资源量的过程称为算法复杂性分析，简称为算法分析。理论上，算法分析既要计算算法的时间复杂性，也要计算它的空间复杂性。然而，算法的运行时间都是消耗在数据处理上的，从这个意义上说，算法的空间复杂性不会超过时间复杂性。出于这个原因，人们多关注于算法的时间复杂性分析。本书中除非特别说明，所说的算法分析，仅局限于对算法的时间复杂性分析。

为客观、科学地评估算法的时间复杂性，我们设置一台抽象的计算机，它只用一个处理机，却有无限量的随机存储器。它的有限个基本操作——算术运算、逻辑运算和数据的移动（比如对变量的赋值）均在有限固定时间内完成，我们进一步假定所有这些基本操作都消耗一个时间单位。

称此抽象计算机为随机访问计算机，简记为 RAM。算法在 RAM 上运行时所需的时间，显然就是执行基本操作的次数。不难看出，一个算法的时间复杂性与输入的规模相关，一般来说，规模越大，需要执行的基本操作就越多，当然运行时间就越长。此外，即使问题输入的规模一定，不同的输入，也会导致运行时间的不同。很多文献对一个算法的运行时间，研究如下的 3 种情形：

- 对固定的输入规模，使运算时间最长的输入所消耗的运行时间称为算法的最坏情形时间。
- 对固定的输入规模，使运行时间最短的输入所消耗的时间，称为最好情形时间。
- 假定固定的输入规模为 n，所有不同输入构成的集合为 D_n，对问题的每一个输入为 $I \in D_n$，若已知该输入发生的概率为 $P(I)$，对应的运行时间为 $T(I)$，运行时间的数学期望值 $\sum_{I \in D_n} P(I)T(I)$ 称为算法的平均情形时间。

3. 实例

我们用一个简单的实例来说明这些概念：考虑在输入的线性表 $A[1...n]$中查找值为 x 的元素，若线性表中存在这样的元素，则输出下标最小者，否则报告不存在信息。我们用下面的算法来解决这个问题：从 $A[1]$起逐一扫描线性表中元素，若表中存在这样的元素，返回第一个遇到的值等于 x 的元素 $A[i]$的下标 i，否则，这个扫描过程将持续到表尾，报告无解信息，问题得以解决，如图 1-1 所示。

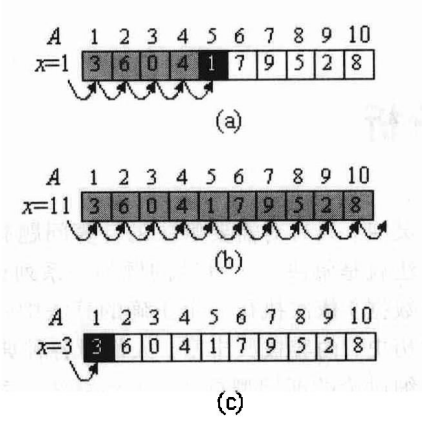

图 1-1　在线性表 A 中查找。(a) 查找值为 x=1 的元素，从 $A[1]$起依次要进行 5 次检测，第一次找到值为 1 的元素。(b) 查找值为 x=11 的元素，从 $A[1]$起依次检测完所有元素（进行 10 次检测），没有找到值为 11 的元素——最坏情形。(c) 查找值为 x=3 的元素，从 $A[1]$起仅进行一次检测就找到值为 3 的元素——最好情形。

这个算法对于无解输入（即输入的线性表 $A[1...n]$中不存在值等于 x 的元素），所消耗的时间最长，因为它需要重复检测 n 次（也就是要做 n 次逻辑运算），所以最坏情形时间为 n。

当输入的线性表中第一个元素 $A[1]$的值就等于 x，则算法仅进行一次检测就可返回答案。这是最好的情形，所以该算法的最好情形时间为 1。

假定第一个值等于 x 的元素等概地分布在 $A[1...n]$中，也就是说第一个等于 x 的元素 $A[i]$的下标 i 为 1, 2, ..., n 的概率均为 $1/n$。这样，我们的算法要进行 i 次检测的概率为 $1/n$。所以算法的平均情形时间为：

$$\sum_{i=1}^{n} P(\text{做}i\text{次检测}) \cdot i = \sum_{i=1}^{n} \frac{1}{n} \cdot i = \frac{1}{n} \sum_{i=1}^{n} i = \frac{1}{n} \cdot \frac{n(n+1)}{2} = \frac{n+1}{2}$$

显然，算法的最好情形时间是有“欺骗性”的，而平均情形时间的研究要用到概率统计的知

识。算法最坏情形时间可视为算法对固定输入规模 n 的运行时间的上界，用它来表示算法的时间复杂性是合理的。本书若无特殊说明，就将算法的最坏情形时间称为算法的运行时间。我们把算法的运行时间记为 T，输入的规模记为 n。则根据以上说明知 T 是 n 的递增函数，我们以 $T(n)$ 来表示算法的运行时间。

4. 算法的渐进运行时间

由于计算机技术不断地扩张其应用领域，所要解决的问题输入规模也越来越大，所以对固定的 n 来计算 $T(n)$ 的意义并不大，我们更倾向于评估当 $n\to\infty$ 时，$T(n)$ 趋于无穷大的快慢来分析算法的时间复杂性。我们往往用几个函数 $\tilde{Y}(n)$：幂函数 n^k（k 为正整数）、对数幂函数 $\lg^k n$（k 为正整数，底数为 2）和指数函数 a^n（a 为大于 1 的常数）作为“标准”，研究极限：

$$\lim_{n\to n}\frac{\tilde{Y}(n)}{T(n)}=\lambda$$

若 λ=非零常数，则称 $\tilde{Y}(n)$ 是 $T(n)$ 的渐近表达式，或称 $T(n)$ 渐近等于 $\tilde{Y}(n)$，记为 $T(n)=\Theta(\tilde{Y}(n))$，这个记号称为算法运行时间的渐近[1] Θ-记号，简称为 Θ-记号。例如，$T(n)=3n^2+2n+1$，由于

$$\lim_{n\to n}\frac{n^2}{T(n)}=\lim_{n\to n}\frac{n^2}{3n^2+2n+1}=1/3\neq 0$$

所以，有 $T(n)=\Theta(n^2)$，即此 $T(n)$ 渐近等于 n^2。其实，在一个算法的运行时间 $T(n)$ 中省略最高次项以外的所有项，且忽略最高次项的常数系数，就可得到它的渐近表达式 $\Theta(\tilde{Y}(n))$。用此方法也能得到 $3n^2+2n+1=\Theta(n^2)$。

5. 有效算法

如果两个算法运行时间的渐近表达式相同，则将它们视为具有相同时间复杂度的算法。显然，渐近时间为对数幂的算法优于渐近时间为幂函数的算法，而渐近时间为幂函数的算法则优于渐近时间为指数函数的算法。我们把渐近时间为幂函数的算法称为是具有多项式时间的算法，渐近时间不超过多项式的算法则称其为有效的算法。本书讨论的大多数问题都有解决它的有效算法，我们将在第 5 章讨论一些至今无法知道其是否有“有效的”算法的问题，并由此介绍算法研究的核心问题，也是计算机科学的核心问题——NP 难问题。

一旦选择了算法，就需要运用一种合适的程序设计技术（主要是程序设计语言）将算法实现为程序。利用计算机，运行这些程序帮助人们快速、正确地解决各种问题。本书旨在与读者一起分享从算法设计、分析到实现能运行的程序这一美妙的三部曲给我们带来的创造过程的快乐。

6. 渐增型算法

我们从讨论一类最简单的算法设计技术——渐增型算法开始。所谓渐增型算法（incremental algorithms），指的是算法使得表示问题的解从较小的部分渐渐扩张，最终成长为完整解。渐增型算法有一个共同的特征：构成算法的主体是一个循环结构，它逐步将部分解扩张成一个完整解。该循环将遵循一个始终不变的原则：每次重复之初，总维持着问题的一个部分解。我们将此特征

[1] 关于渐近记号的详细阐述请参见本书附录 B。

称为算法的循环不变量（loop invariant）。利用循环不变量来证明渐增型算法的正确性是软件正确性[1]证明的一种很好的方法。

本章介绍几种典型的渐增型算法，讨论它们的正确性，指出它们的效率，并用 C/C++/Java 加以实现。

1.2 插入排序算法

1.2.1 算法描述与分析

1. 问题的理解与描述

算法总是针对某个问题的。在正确理解问题的基础上，为便于我们设计出正确解决问题的算法，往往要将问题形式化地表示出来。任何计算问题，都有明确地反映问题中对象属性的数据，我们将其称为问题的输入。问题的解，也要以数据的形式表示出来，我们称其为问题的输出。本质上讲，一个正确的算法就是问题的输入与输出之间的一个对应。问题的形式化表示就是写出问题的输入与输出。在各种应用中，常要对数据进行排序。例如，玩扑克牌时，玩家对手中的牌通常要按点数从小到大进行排列。排序问题的形式化表示为：

输入：一组数$<a_1, a_2, ..., a_n>$。

输出：输入的一个排列（重排）$<a'_1, a'_2, ..., a'_n>$，满足 $a'_1 \leqslant a'_2 \leqslant, ..., \leqslant a'_n$。

我们把这样从小到大的顺序称为升序，如图 1-2 所示。反之，从大到小的排序称为降序。下面讨论的插入排序是解决升序排序问题的最简单的算法之一，它的想法与我们摸一手扑克牌相似。从左手为空开始，扑克牌背面朝上放于桌上，每次从桌上摸一张牌，并将其插入到左手正确的位置上，使得左手中的牌是有序的。为找到这张牌的正确插入位置，从右向左逐一比较左手中的牌。任何时候，左手中的牌都是排好序的，并且所有的牌都是桌上最前面的若干张。当摸完桌上的牌，则左手中的牌就排好序了。

图 1-2 按排点的升序排好序的扑克牌

2. 算法的伪代码描述

把牌抽象成由多个数值（牌面点数）组成的一个序列 $A=<a_1, a_2, ..., a_n>$，上述的插入排序思想可用伪代码[2]形式化地描述如下。

```
INSERTION-SORT (A)
1 for j ← 2 to length[A]
2    do key ← A[j]
3      ▷将 A[j]插入到排好序的序列 A[1… j - 1]中。
4      i ← j - 1
5      while i > 0 and A[i] > key
```

[1] 软件正确性指的是软件具有对所有合法输入能得到正确输出，对所有不合法输入能给出恰当的信息并作出恰当的善后处理的特征。对于大型软件而言，证明其正确性迄今为止无论是理论上还是实践上都还没有系统的方法，这里所介绍的用循环不变量证明软件中所包含的渐增型算法的正确性，在软件正确性证明研究中被认为是一种有效的局部方法。

[2] 本书所使用的伪代码规范在附录 A 中有详细介绍，初次接触伪代码的读者请先参阅附录 A 中的相关内容。

```
6        do A[i + 1] ← A[i]
7           i ← i - 1
8     A[i + 1] ← key
```

算法 1-1　解决排序问题的 INSERTION-SORT 算法

3. 算法的正确性

在过程 INSERTION-SORT 中，$A[1...j-1]$是部分解（对应于左手中排好序的牌），算法通过 1～8 行的 for 循环使得这个部分解（将 $A[j]$插入到 $A[1...j-1]$正确的位置上，使得 $A[1...j]$有序）逐渐成为全部解（$A[1...n]$有序）。按此认识，于是总结出 INSERTION-SORT （A）中第 1～8 行的 for 循环的循环不变量为：每次重复之初，子序列 $A[1...j-1]$由原来 $A[1...j-1]$中的元素组成，且已排好序。

对一个算法而言，如果任一合法的输入都能得到一个正确的输出，则说明该算法是正确的。利用循环不变量可以证明渐增型算法的正确性。以插入排序为例，证明分成以下 3 步：

初始状态：第 1～8 行的 for 循环第一次重复之初，$j=2$，$A[1...j-1]=A[1]$。这当然符合循环不变量：$A[1...j-1]$是由原来 $A[1...j-1]$中的元素组成，且已排好序。

维持状态：假定第 j（>2）次重复之初循环不变量为真，即本次重复之初 $A[1...j-1]$是由原来 $A[1...j-1]$中的元素组成，且已排好序，则在本次重复中将完成循环体内第 2～8 行的操作。循环体中所作的操作是将 $A[j]$插入到 $A[1...j-1]$合适的位置，使得 $A[1...j]$排好序。这样，本次重复的结果就成了下次重复之初（j 增值 1）时的状态：$A[1...j-1]$是由原来 $A[1...j-1]$中的元素组成，且已排好序。

终止状态：当 for 循环结束时，$j=n+1$。按照循环不变量的阐述，$A[1...n]$ 是由原来 $A[1...n]$ 中的元素组成，且已排好序。这正是我们所要的排序结果。因此，算法 INSERTION-SORT 是能正确解决排序问题的。

INSERTION-SORT 算法的一个运行实例如图 1-3 所示。

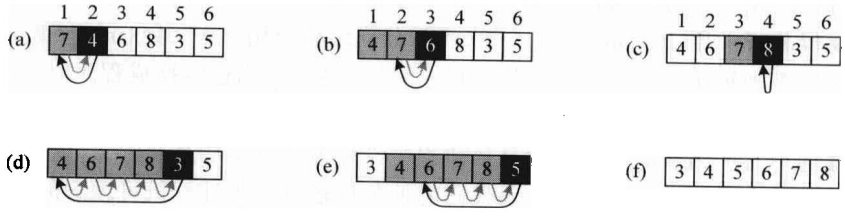

图 1-3　INSERTION-SORT 在 $A=\langle 7,4,6,8,3,5\rangle$上的操作。数组的下标表示在各方格的上方，存储在数组中的数值表示在各方格内。（a）～（e）第 1～8 行的 for 循环的各次重复。每次重复中，黑色方格放的是键 $A[j]$，它在第 5 行中逐一与其左边灰色方格内的元素比较。灰色的箭头指示处在第 6 行中被右移一格的元素，黑色箭头则指示出键在第 8 行移动到的位置。（f）最终排好序的数组。

4. 算法的运行时间

假定序列 A 含有 n 个元素，对 INSERTION-SORT 而言，最坏情形是序列 A 初始状态是降序排列。在此情形下，对第 1～8 行的 for 循环的 $n-1$ 次重复的每一次，内嵌的第 5～7 行的 while 循环将分别重复 1，2，…，$n-1$ 次。所以，第 6～7 行的操作将重复进行 $1+2+\cdots+n-1=n(n-1)/2$ 次。于是，该算法的最坏情形时间用 Θ-记号表示为 $\Theta(n^2)$。其实，经验告诉我们，对于嵌套循环，往往将每层循环的最多重复次数相乘就能得到最坏情形的运行时间。例如，在 INSERTION-SORT 中，外层的 for 循环重复 $n-1$ 次，内层的 while 循环最多重复 $n-1$ 次，所以可得最坏情形的运行时间为 $\Theta(n^2)$。

1.2.2 程序实现

1. 实现要点

我们已经看到，描述算法的伪代码过程与使用的计算机程序设计语言十分相像。然而，算法即使表示成了伪代码，它仍然不等同于程序。这主要出于如下几个原因：

（1）算法的伪代码描述着眼于算法思想的阐述，高度抽象是其最基本的特征之一。例如，在伪代码中可以用求和符号 $\sum_{i=1}^{n} x_i$ 简约地表示序列$<x_1, x_2, \ldots, x_n>$的累加，而在程序中，也许就要用一个循环结构来表示这个累加过程了。

（2）算法的伪代码描述并不关心数据的存储格式。仍然以上述的序列累加为例。算法无需说明序列$<x_1, x_2, \ldots, x_n>$是存储在一个数组中还是存储在一个链表中，这在程序中是不允许的。

（3）算法伪代码描述对变量无需事先声明，读者只要在上下文中能识别各变量及其用途就可以了。这对于我们选用的 C 语言，或是 C++语言，或是 Java 语言而言，都是行不通的。

可见，算法伪代码描述是写给人看的，它是人们用来设计程序的蓝图，而实现算法的程序是按设计蓝图施工而得到的成品，是写出来让机器运行的。

下面来看一看将算法的伪代码描述转换成计算机程序需要进行哪些基本的思考。首先，要将注意力放在算法伪代码过程对应于问题输入的参数以及对应于问题输出的向外部输出数据是什么。它们决定了实现的程序过程（独立函数或类中的方法）的参数和返回值。这包括要考虑有多少个参数和返回值，以及它们的意义和类型。

其次，还要考察算法过程中所需要访问的所有数据，包括变量和常量。要对每个变量考虑它的数据类型、存储类型、访问限制和初始值，等等。

我们说算法的伪代码过程与计算机程序很“像”，是因为伪代码规范的外部语法与程序设计语言的控制逻辑相近：用 if…then…else…来表示分支结构；用 for…或 while…表示循环结构；用语句的书写顺序表示执行顺序等。所以，在将算法的伪代码描述转换成程序时，可以借助伪代码的外部语法结构来决定程序的控制结构，因而节约我们在这方面的精力和劳动。然而，算法伪代码过程的描述中有些因素的抽象程度是目前程序设计语言所不能企及的，这就需要我们用程序设计语言提供的技术来为计算机将这些伪代码的抽象描述解释为计算机可理解的语句或表达式。以 1.2.1 小节讨论的插入排序算法 1-1 为例，按如下项目考虑程序实现。

【参数与返回值】过程 INSERTION-SORT 只有 1 个参数：欲排序的序列 A。由于排序的结果维持在序列 A 中，所以该过程无需返回任何值。

要转换成程序，当然需要向程序过程传递欲排序的序列 A，但需要考虑 A 中的元素是什么类型，A 按怎样的结构加以存储（数组还是链表）。

【数据设置】过程 INSERTION-SORT 中所访问的变量包括两重嵌套循环的控制变量 j 和 i，j 控制外层的 for 循环，而 i 控制内层的 while 循环。此外，还有序列元素 $A[j]$的值暂存变量 key。

在实现程序中，根据伪代码的上下文，可以确定 j 和 i 的类型为整型，因为它们要作为序列元素的下标。而对于变量 key，其类型必须与序列 A 中元素的类型一致。

【关键代码】过程 INSERTION-SORT 中的代码结构是很简单的，所以将其转换成程序并不困难。需要注意以下 4 个方面：

第 1，伪代码描述序列 A 的长度是用比较抽象的形式 $length[A]$表示，这对于抽象程度不高的语言而言（如 C 语言），就需要通过将序列 A 的长度 n 作为参数向程序过程传递的方法来实现。

第 2，伪代码中，序列元素下标是从 1 开始编号的，而在 C 类语言（C/C++/Java）中，数组等连续存储的序列的下标是从 0 开始编号的。

第 3，伪代码中变量的赋值符号“←”明确地表示出了赋值操作的方向性，而在 C 类语言中是使用运算符“=”表示赋值运算的。

第 4，在 C 类语言中有一套极具特色的运算符：自增/自减运算符。用 i++（或++i）表示 $i \leftarrow i+1$，i--（或--i）表示 $i \leftarrow i-1$。

本书中，采用 C、C++和 Java 3 种语言来实现算法。对每一个算法我们将为读者给出如上的“参数与返回值”、“数据设置”和“关键代码”3 个方面的讨论。而对每一种具体语言的实现，都将说明对应于该语言技术特点的这 3 个方面的补充信息。读者既可选择一种感兴趣的语言研读，也可兼读两种或三种语言的实现，以比较这 3 种语言的异同，从中获取更多乐趣。

2. C 语言实现

（1）整型数组版本。

如果数据序列连续存储，即表示为数组，这个算法用 C 语言来实现是很容易的。下面以整型数组为例。

```
1 #ifndef _INSERTIONSORT_H
2 #define _INSERTIONSORT_H
3 void insertionSort(int *a, int n){
4     int i,j,key;/*key 的类型与数组 a 的元素类型相同*/
5     for(j=1;j<n;j++){
6         key=a[j];/*key←a[i]*/
7         i=j-1;/*i← j-1*/
8         while((i>=0)&&(a[i]>key)){
9             a[i+1]=a[i];/* a[i+1] ←a[i]*/
10            i--;/*i←i-1*/
11        }
12        a[i+1]=key;/* a[i+1] ←key */
13    }
14 }
15 #endif    /* _INSERTIONSORT_H */
```

程序 1-1　实现算法 1-1 的 C 源代码文件 insertionsort.h

程序解析

如前所述，因为作为用指针传递给函数的数组 a 自身并不具有长度属性，要用另一个参数 n 才能表示数组长度，用它来表示算法中的 $length[A]$。

第 4 行中声明的整型变量 key 是对应于算法 1-1 中用来临时存放 $A[j]$的值的同名整型变量（因为参数 a 是整型数组）。

可以利用如下的简单程序来测试上述的函数。

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "insertionsort.h"
4 int main(int argc, char** argv){
5   int a[]={5,1,9,4,6,2,0,3,8,7},i;
6   insertionSort(a+1,7);/*对 a[0…9]中的 a[1…7]排序*/
7   for(i=0;i<10;i++)
8       printf("%d ", a[i]);
```

```
9    printf("\n");
10  return (EXIT_SUCCESS);
11 }
```

程序 1-2　测试程序 1-1 的 C 源代码文件 test.c

编写一个测试程序，需要抓住 3 个要点：测试的数据、过程的调用和程序的输出。本书中的每个测试程序，都将为读者说明这 3 个要点。对于程序 1-2，我们给出以下的解析。

程序解析

【测试数据】第 5 行声明了一个具有 10 个元素的整型数组 a[0...9]（初始化为{5,1,9,4,6,2,0,3,8,7}）。

【过程调用】第 6 行调用 insertionSort(a+1,7)，是对数组 a[0],a[1], ..., a[9]中 a[1],...a[7]排序。

【程序输出】第 7～9 行将数组 a（子数组 a[1...7]排好序）输出到屏幕。程序运行结果输出：

```
5 0 1 2 3 4 6 9 8 7
```

（2）任意类型数组版本。

注意，程序 1-1 中函数 insertionSort 的参数 a 是指向整数的指针。换句话说，它只能对整数数组排序，如果要对其他类型数据数组做插入排序，就需要修改程序的参数 a 类型和局部变量 key 的类型。这样，对不同的数据类型的数组要产生大量重复代码致使代码量剧增，并且还带来了函数的命名管理问题：要为每一个函数起一个唯一的名字。为了重用代码，需要做些技术改进。C 语言最重要的技术是指针，我们来看下面的代码。

```
1 #ifndef _INSERTIONSORT_H
2 #define _INSERTIONSORT_H
3 #include<string.h>
4 void insertionSort(void *a, int n,int size,int (*comp)(void *,void *)){
5     int i,j;
6     void *key=(void *)malloc(size);
7     for(j=1;j<n;j++){
8         memcpy(key,a+j*size,size);/*key←a[j]*/
9         i=j-1;
10        while((i>=0)&&(comp(a+i*size,key)>0)){/*a[i]>key*/
11            memcpy(a+(i+1)*size,a+i*size,size);/*a[i+1]←a[i]*/
12            i--;/*i←i-1*/
13        }
14        memcpy(a+(i+1)*size,key,size);/*a[i+1]←key*/
15    }
16 }
17 #endif    /* _INSERTIONSORT_H */
```

程序 1-3　对程序 1-1 进行了通用性改进的 C 源代码文件 insertionsort.h

程序解析

与程序 1-1 相比，新版的 insertionSort 函数的指针参数 a 的类型变成了 void*，这意味着它可以代表任何类型的数组指针。新增了两个参数：指出数组元素的存储长度的整型 size 和确定数组中元素大小比较规则的函数指针 comp，它们的意义下文解说。

第 6 行声明的局部变量 key 的类型也随之改变成 void*，用来存储 a[j]的值。

系统对 void*变量，是不能对其指向的单元作诸如赋值等操作的。因此，需要用底层内存块复制的函数来完成赋值操作。第 8、11 和 14 行调用 memcpy 函数完成这一操作。该函数定义于

头文件 string.h 中，其用户接口为：

```
void *memcpy(void *dest, const void *src, unsigned long size)
```

其功能是将以 src 指向的内存地址开始 size 个字节内的数据复制到以 dest 指向的 size 个字节内，如图 1-4 所示。这就说明了参数 size 的意义之一。

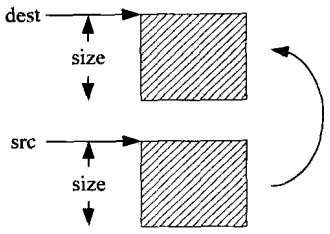

图 1-4　memcpy(void *dest, const void *src, unsigned long size)示意图

由于系统对 void*指向的单元长度默认为 1 个字节（相当于 char*指向的内存单元长度[1]）。这样，为访问数组 a 中的第 i 个元素简单地用 a[i]就不对了，而应该为 a[i*size]才对。这就使第 8、10、11 和 14 行中出现了诸如 a+i*size 的指针访问。这是参数 size 的第二个意义。

内嵌于 for 循环中的 while 循环的循环条件在两个版本中也有不同：程序 1-1 中是(i>=0)&&(a[i]>0)，而在新版本中变成(i>=0)&& (comp(a+i*size, key)>0)。原因是系统对 void *类型指针指向的内存数据不能执行比较运算，但事实上存储在 void *所指引的地址中的数据类型是应当（也是必须）能被比较的。这就需要有一个知道在 void *所指引的地址中的数据类型的函数来完成大小比较操作。这就是参数 comp 的意义。comp 是一个具有两个 void *类型参数，返回整数型数据的函数指针。下面的代码就是适合于不同数据类型的 comp 的实际例子。

```
1 #ifndef _COMPARE_H
2 #define _COMPARE_H
3 int intGreater(void *x,void *y){
4     return (*((int*)x))-(*((int*)y));
5 }
6 int intLess(void *x,void *y){
7     return intGreater(y,x);
8 }
9 int charGreater(void *x,void *y){
10    return (*((char*)x))-(*((char*)y));
11 }
12 int charLess(void *x,void *y){
13    return charGreater(y,x);
14 }
15 int strGreater(void *x,void *y){
16    return strcmp(*((char**)x),*((char**)y));
17 }
18 int strLess(void *x,void *y){
19    return strGreater(y,x);
20 }
21 int floatGreater(void *x,void *y){
22    if((*((float*)x)-*((float*)y))>0.0)
```

[1] 在 gcc 中这是默认的，但一些编译系统对于传递给诸如 memcpy 这样的函数的 viod*型实际参数时，需强制转换成（char*）才能正确编译，如在 TurboC2.0 和 MICROSOFT 的 visualC++9 等中。

```
23        return 1;
24    if((*((float*)x)-*((float*)y))<0.0)
25        return -1;
26    return 0;
27 }
28 int floatLess(void *x,void *y){
29    return floatGreater(y,x);
30 }
31 int doubleGreater(void *x,void *y){
32    if((*((double*)x)-*((double*)y))>0.0)
33        return 1;
34    if((*((double*)x)-*((double*)y))<0.0)
35        return -1;
36    return 0;
37 }
38 int doubleLess(void *x,void *y){
39    return doubleGreater(y,x);
40 }
41 #endif    /* _COMPARE_H */
```

程序 1-4　关于基本数据类型大小比较的 C 函数

这是一组关于比较有 void*指向的基本数据类型大小的 C 函数。其中，第 3～5 行和第 6～8 行定义的 intGreater 和 intLess 是用来检测 int 型数据大小的，第 31～37 行和第 38～40 行定义的 doublelGrater 和 doubleLess 是用来检测 double 型数据大小的。我们仅就第 21～27 行和第 28～30 行的 floatGreater 和 floatLess 函数加以解析，其余的留给读者阅读思考。

程序解析

第 21～27 行的 floatGreater 函数有两个参数 x 和 y，它们都是 void*型的指针，指向的地址内存储的是 float 型数据。根据 x 和 y 所指向的地址中的数据的大小，函数将返回 0（相等的情形），或返回大于 0 的整数 1（前者大于后者的情形），小于 0 的整数-1（前者小于后者的情形）。

由于比较的是 x 和 y 所指向的地址内的 float 型数据，而 x 和 y 是按 void*类型传递进来的，所以，需要先将 x 和 y 强制转换成指向 float 型数据的指针，然后用*运算符访问地址内的数据。这就是第 22 行和第 24 行中所检测的表达式的意义。

对于第 28～30 行定义的 floatLess 函数，其参数 x 和 y 的意义与 floatGreater 函数的一样。返回值类型一致但意义相反：x 指向的数据小于 y 指向的数据返回大于 0 的整数 1，x 指向的数据大于 y 指向的数据返回大于 0 的整数-1，两者相等则返回 0。依此意义，我们调用 floatGreater 传递的参数为 y 和 x，这样返回的值恰好符合上述的比较规则。

由于以后还会遇到对整数、字符串和浮点型数据的比较，所以将上述的这些函数定义写进一个 compare.h 的头文件中，并保存于一个名为 Utility[1]的文件夹中以备重用。

我们用下列的代码来测试程序 1-3。

```
1 #include <stdio.h>
2 #include<stdlib.h>
3 #include "insertionsort.h"
4 #include "…/Utility/compare.h"
5 int main(int argc, char** argv) {
6    int a[]={5,1,9,4,6,2,0,3,8,7},i;
```

[1] 关于本书正方代码的目录结构请参阅附录 E。