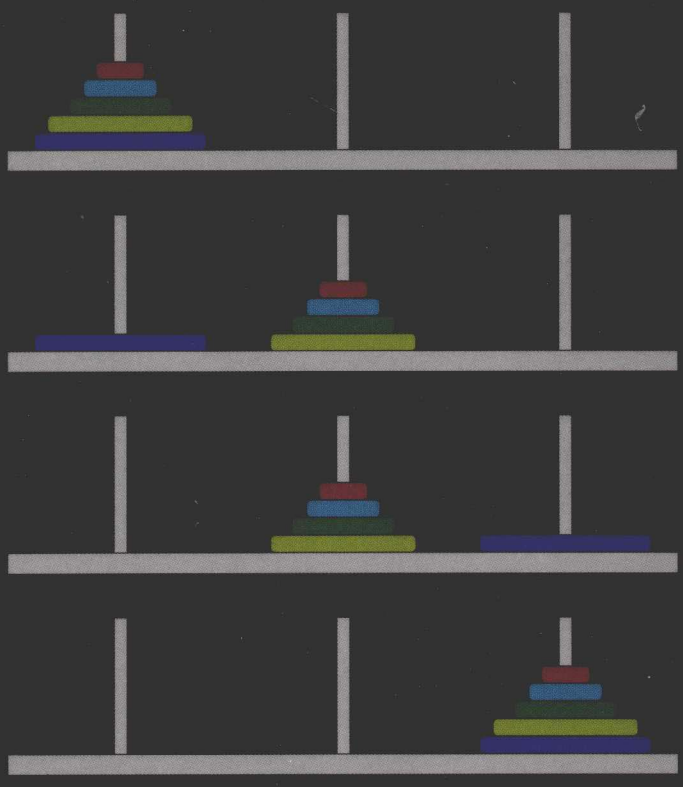
| 源程序 + PPT 教学资源 |

CD-ROM

算法设计、分析与实现：C、C++ 和 Java

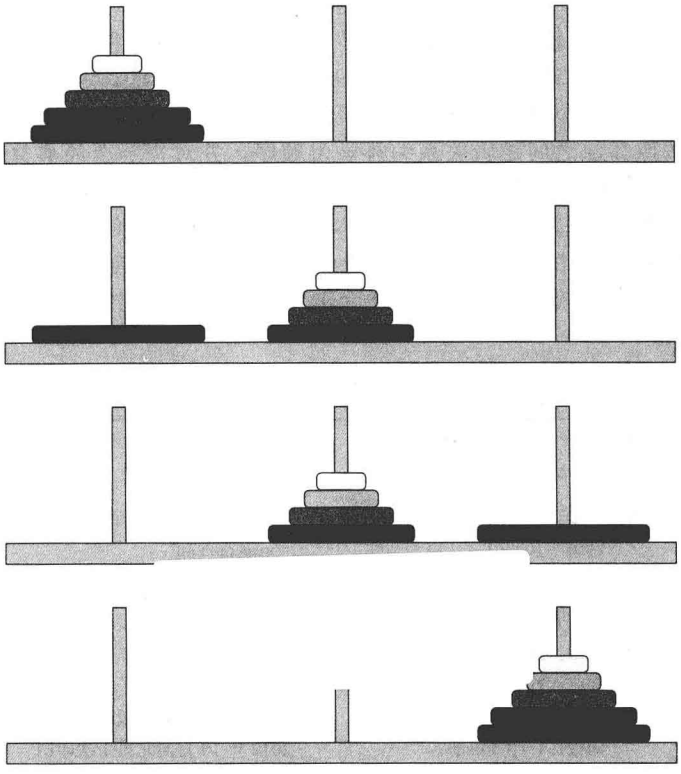
徐子珊 编著



人民邮电出版社
POSTS & TELECOM PRESS

算法设计、分析与实现：C、C++和Java

徐子珊 编著



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 : C、C++和Java / 徐子珊编著. -- 北京 : 人民邮电出版社, 2012.10
ISBN 978-7-115-28990-2

Ⅰ. ①算… Ⅱ. ①徐… Ⅲ. ①C语言－程序设计②JAVA语言－程序设计 Ⅳ. ①TP312

中国版本图书馆CIP数据核字(2012)第161351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，第 9 章是算法的实践部分，教给读者如何进行调试。

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。

本书适合各种学习算法的人员使用，也适合作为大中专院校的学习用书，及培训学校的教材。

算法设计、分析与实现：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　　电子邮件　315@ptpress.com.cn
网址　http://www.ptpress.com.cn
三河市潮河印业有限公司印刷

◆ 开本：800×1000　1/16
印张：29
字数：995 千字　　2012 年 10 月第 1 版
印数：1－3 500 册　　2012 年 10 月河北第 1 次印刷

ISBN 978-7-115-28990-2

定价：65.00 元（附光盘）

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

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。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，并用算法设计与分析的理论指导程序设计实践是本书试图与读者一起探索的最重要的目标之一。

本书前 6 章先按算法设计技巧分为渐增型算法、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。每章将 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。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，再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章节编排，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。

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；其二，需要考虑过程中所要用到的所有变量，包括局部变量和全局变量；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 、Visual C++ 2010 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，。

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。

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。

本书是笔者两年前编著的《算法设计、分析与实现从入门到精通 C、C++和 Java》一书的修订版。与原书相比，作了如下改进。

1．修补了原书中的错漏。

2．将第 5 章中较激进的 C 代码（把组合问题表示成结构体类型，将刻画问题行为的函数作为结构体中的指针属性）改写成比较传统的形式，以适应更多读者的思维习惯。

3．替换了个别应用题目。

4．将所有代码从 Linux 平台移植到 Windows 平台。编译系统从 gcc/g++转换成 Visual C++，Java 语言的开发平台换成流行的 Eclipse。这也是为了适应更多读者的习惯。

5．增加了说明本书代码使用及应用题实验方法的“第 9 章 实验指南”。

6．为希望将本书作为算法教材的老师编写了 PPT 课件资料。为偏爱 PDF 文件的老师提供了全套的课件 PDF 副本。全书的程序代码和 PPT、PDF 文件都刻录在随书光盘中。

诚恳期待读者对本书的写作提出意见与建议，联系邮箱为 xu_zishan@163.com，编辑联系邮箱为 zhangtao@ptpress.com.cn。

编者

目　录

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（正确地输出数据）。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 $A[1 \cdots n]$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。

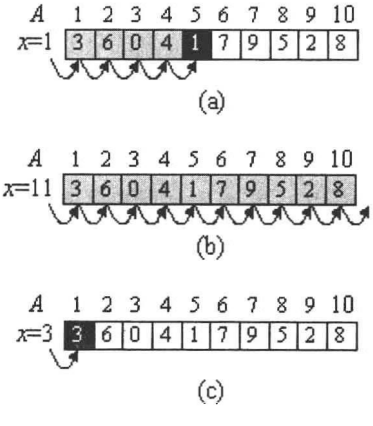


图 1-1　在线性表 A 中查找。（a）查找值为 x=1 的元素，从 $A[1]$起依次要进行 5 次检测，第一次找到值为 1 的元素。（b）查找值为 x=11 的元素，从 $A[1]$起依次检测完所有元素（进行 10 次检测），没有找到值为 11 的元素——最坏情形。（c）查找值为 x=3 的元素，从 $A[1]$起仅进行一次检测就找到值为 3 的元素——最好情形。

这个算法对于无解输入（即输入的线性表 $A[1\cdots n]$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 $A[1\cdots n]$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 来计算算法的运行时间 $T(n)$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：

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

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

[1] 关于渐近记号的详细阐述请参见本书附录 B。

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$<a_1, a_2, \cdots, a_n>$。

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

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

[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。

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 $A=<a_1, a_2, \cdots, a_n>$，上述的插入排序思想可用伪代码[1]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
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，$A[1\cdots j-1]$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 $A[1\cdots j-1]$正确的位置上，使得 $A[1\cdots j]$有序）逐渐成为全部解（$A[1\cdots n]$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 $A[1\cdots j-1]$由原来 $A[1\cdots j-1]$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，$j=2$，$A[1\cdots j-1]=A[1]$。这当然符合循环不变量：$A[1\cdots j-1]$是由原来 $A[1\cdots j-1]$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 $A[1\cdots j-1]$是由原来 $A[1\cdots j-1]$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 $A[1\cdots j-1]$合适的位置，使得 $A[1\cdots j]$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：$A[1\cdots j-1]$是由原来 $A[1\cdots j-1]$中的元素组成，且已排好序。

[1] 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。

终止状态：当 **for** 循环结束时，$j=n+1$。按照循环不变量的阐述，$A[1\cdots n]$ 是由原来 $A[1\cdots n]$ 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。

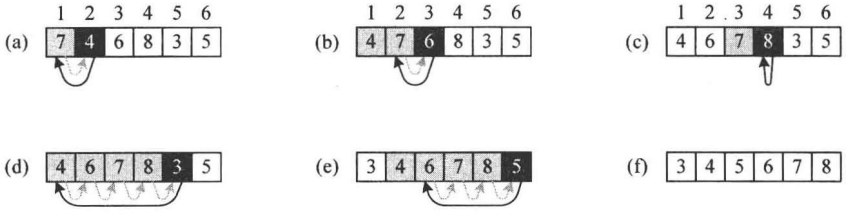


图 1-3　INSERTION-SORT 在 $A=\langle 7,4,6,8,3,5\rangle$ 上的操作。数组的下标表示在各方格的上方，存储在数组中的数值表示在各方格内。（a）~（e）第 1 ~ 8 行的 **for** 循环的各次重复。每次重复中，黑色方格放的是键 $A[j]$，它在第 5 行中逐一与其左边灰色方格内的元素比较。灰色的箭头指示处在第 6 行中被右移一格的元素，黑色箭头则指示出键在第 8 行移动到的位置。（f）最终排好序的数组。

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$\langle x_1, x_2, \cdots, x_n\rangle$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$\langle x_1, x_2, \cdots, x_n\rangle$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“$\leftarrow$”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 void insertionSort(int *a, int n){
2     int i,j,key;/*key 的类型与数组 a 的元素类型相同*/
3     for(j=1;j<n;j++){
4         key=a[j];/*key←a[i]*/
5         i=j-1;/*i← j-1*/
6         while((i>=0)&&(a[i]>key)){
7             a[i+1]=a[i];/* a[i+1] ←a[i]*/
8             i--;/*i←i-1*/
9         }
10        a[i+1]=key;/* a[i+1] ←key */
11    }
12 }
```

程序 1-1　实现算法 1-1 的 C 源代码

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

程序 1-3　对程序 1-1 进行了通用性改进的 C 源代码

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 4 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 6、9 和 12 行调用 memcpy 函数完成这一操作。该函数定义于头文件 string.h 中，其原型为：

```
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。