TURING 图灵计算机科学丛书

PEARSON Addison Wesley

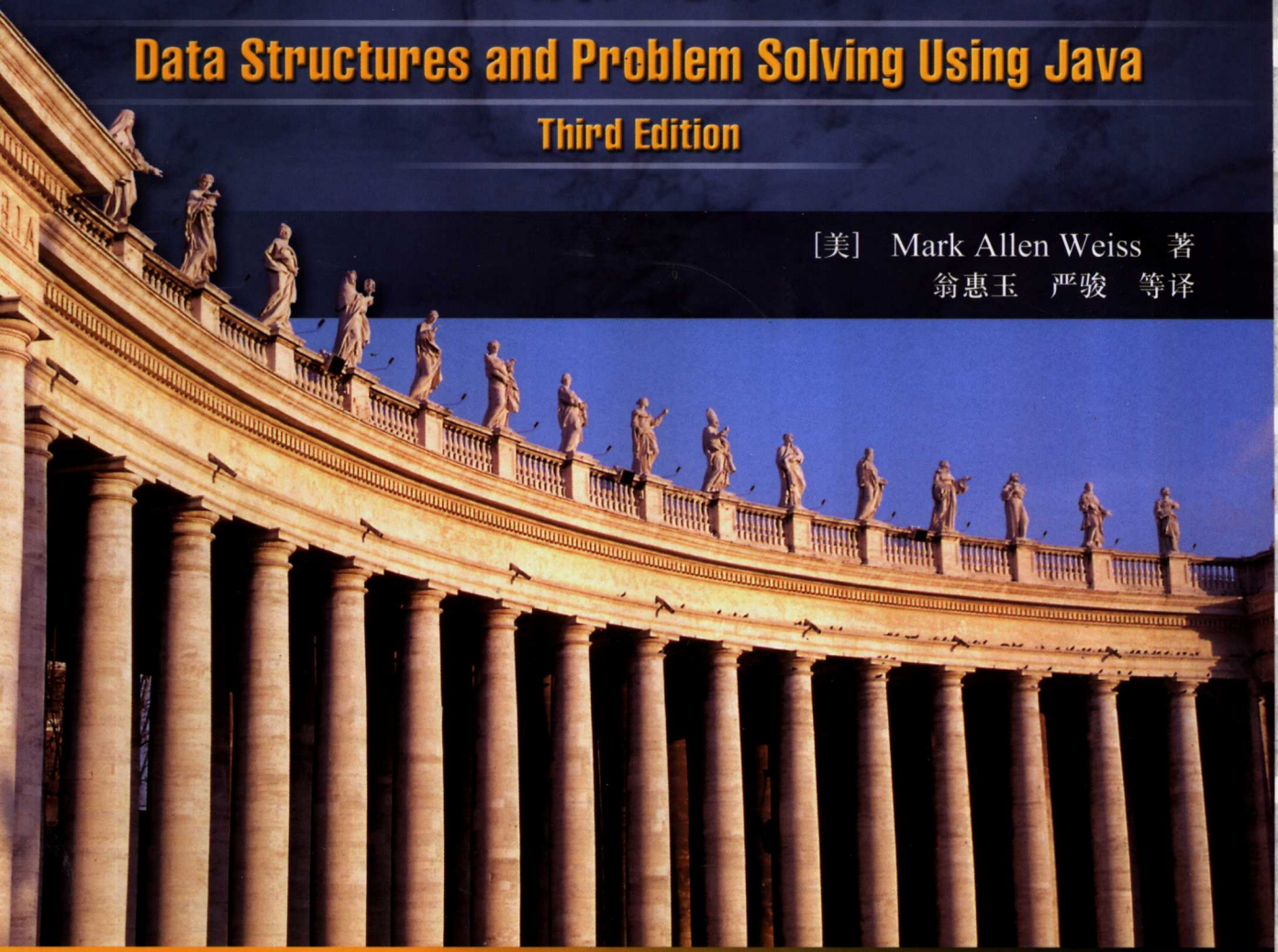
数据结构与问题求解

Java语言描述

（第3版）

Data Structures and Problem Solving Using Java

Third Edition



[美] Mark Allen Weiss 著

翁惠玉 严骏 等译

人民邮电出版社

POSTS & TELECOM PRESS

TURING 图灵计算机科学丛书

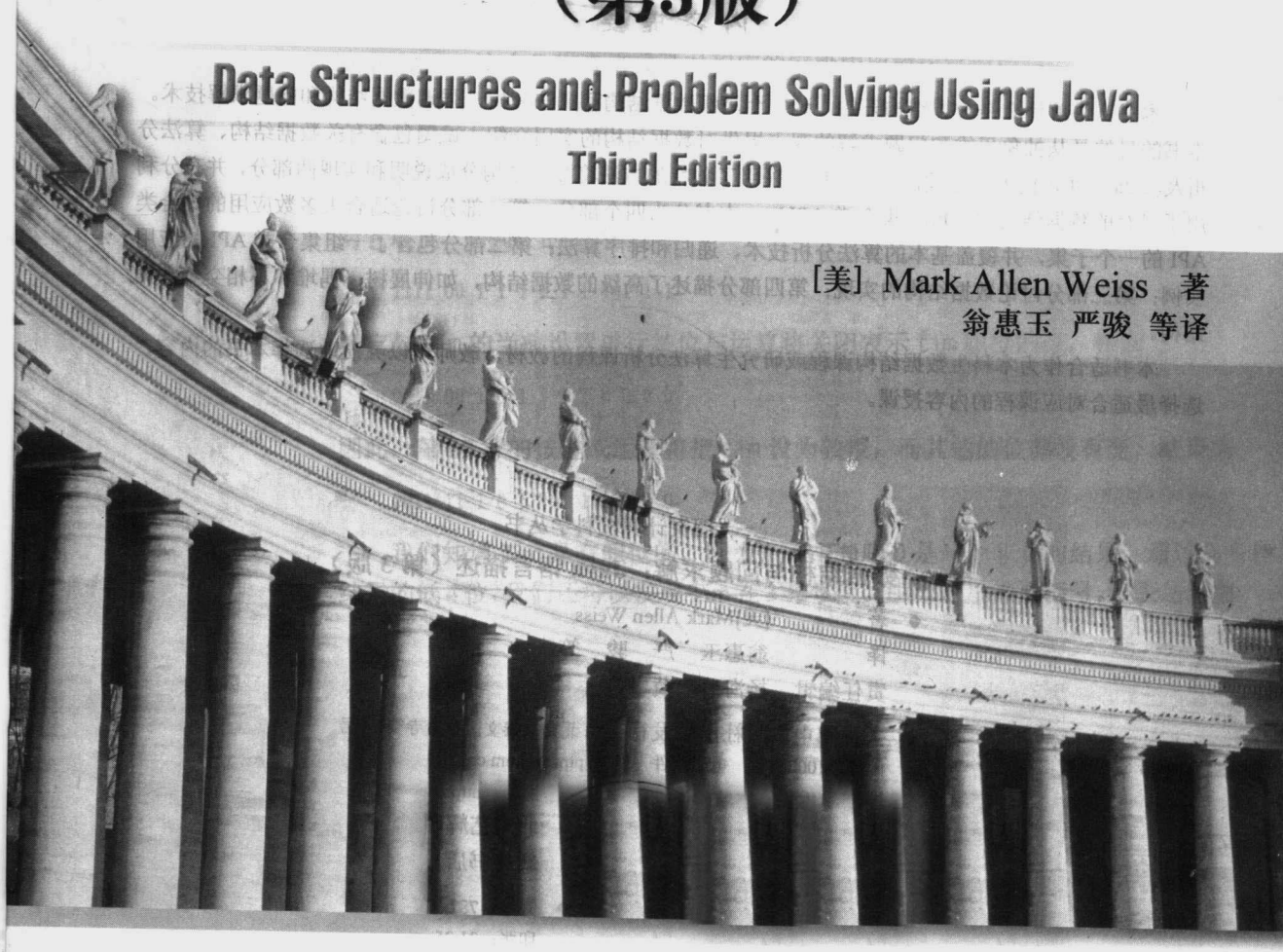
数据结构与问题求解

Java语言描述

（第3版）

Data Structures and Problem Solving Using Java

Third Edition



[美] Mark Allen Weiss 著

翁惠玉 严骏 等译

人民邮电出版社

POSTS & TELECOM PRESS

图书在版编目（CIP）数据

数据结构与问题求解：Java语言描述：第3版 /（美）维斯著；翁惠玉等译.
—北京：人民邮电出版社，2006.7
（图灵计算机科学丛书）
ISBN 7-115-14988-7

Ⅰ.数… Ⅱ.①维…②翁… Ⅲ.①数据结构—教材②JAVA语言—程序设计—教材
Ⅳ.①TP311.12②TP312

中国版本图书馆CIP数据核字（2006）第076821号

内 容 提 要

本书从讲解什么是数据结构开始，延伸至高级数据结构和算法分析，强调数据结构和问题求解技术。本书的目的是从抽象思维和问题求解的观点提供对数据结构的实用介绍，试图包含有关数据结构、算法分析及其Java实现的所有重要的细节。作者采用了独特的方法将数据结构分成说明和实现两部分，并充分利用了已有的数据结构库（Java集合类API）。本书分为四个部分：第一部分讨论适合大多数应用的集合类API的一个子集，并覆盖基本的算法分析技术、递归和排序算法；第二部分包含了一组集合类API的应用实例；第三部分讨论数据结构的实现；第四部分描述了高级的数据结构，如伸展树、偶堆和不相交集数据结构。

本书适合作为本科生数据结构课程或研究生算法分析课程的教材。教师可以灵活地选择本书的内容，选择最适合对应课程的内容授课。

图灵计算机科学丛书

数据结构与问题求解：Java语言描述（第3版）

◆ 著 [美]Mark Allen Weiss
译 翁惠玉 严 骏 等
责任编辑 杨海玲 黄 倩

◆ 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
北京艺辉印刷有限公司印刷
新华书店总店北京发行所经销

◆ 开本：787×1092 1/16
印张：31.25
字数：814千字 2006年7月第1版
印数：1－4 000册 2006年7月北京第1次印刷
著作权合同登记号 图字：01-2005-5229号
ISBN 7-115-14988-7/TP·5548

定价：49.00元

读者服务热线：(010)88593802 印装质量热线：(010)67129223

前　言

本书是为计算机科学专业的两学期课程而设计的，从讲述什么是数据结构开始，延伸至高级数据结构和算法分析。

数据结构课程的内容已经经过了若干年的演变，尽管对于它所覆盖的内容有一些共识，但在细节问题上还有大量的分歧。大家都接受的一个主题是软件开发的原理，最主要的是封装和信息隐藏的概念。从算法方面来看，所有的数据结构课程都趋向于包括运行时间分析、递归、基本排序算法和基本数据结构的介绍。许多大学还提供高级的课程，在更高的层次上讨论数据结构、算法和运行时间分析的问题。本书的内容是为这两个层次的课程设计的，这样就不必要购买第二本教材。

虽然在数据结构领域最激烈的争论都围绕着程序设计语言的选择，但还需要做出如下一些基本选择。

- 是否要尽早引入面向对象的设计或基于对象的设计。
- 数学上的严密程度。
- 在数据结构的实现和使用之间的适当平衡。
- 与所选语言相关的程序设计细节问题（如是否应该较早地使用 GUI）。

我写本书的目的是从抽象思维和问题求解的观点提供对数据结构的实用介绍。我试图覆盖有关数据结构、它们的分析以及它们的Java实现的所有重要细节，而不是停留在理论上很有趣但没有被广泛使用的数据结构。想在一个学期的课程中讲完本书中所有不同数据结构的使用与分析是不可能的，因此本书允许教师灵活地选择所教授的内容。教师需要确定在实践和理论之间的适当平衡，然后选择最适合课程的内容。正如前言后面讨论的那样，我按照尽量减少各章之间依赖性的原则来组织本书。

独特的方法

我最基本的前提是所有语言的软件开发工具来源于大的库，许多数据结构都是这些库的一部分。我预测数据结构课程的重点最终将会从实现转向使用。本书采用独特的方法将数据结构分成说明和实现两部分，并利用已有的数据结构库：Java 集合类 API。

第一部分的第 2 章中完整地讨论了集合类 API 中适合于大多数应用的一个子集。第一部分也覆盖了基本的分析技术、递归和排序。第二部分包含了集合类 API 中数据结构的一组应用。直到第三部分才讨论集合类 API 的实现，而这时这些数据结构早已用到了。因为集合类 API 是 Java 的一部分，因此学生可以尽早用已有的软件组件设计大的项目。

尽管本书中有许多集合类 API 的应用，但它既不是关于集合类 API 的书，也不是关于集合类 API 具体实现的初级教材，它仍然是一本强调数据结构和基本问题求解技术的书。当然，数据结构的设计中所用的通用技术在集合类 API 的实现中都可用，因此第三部分中有几章包括了集合类 API 的实现。不过教师可以在第三部分中选择较简单的实现，以避免讨论集合类 API 的协议。第 2 章介绍了集合类 API，它是理解第二部分中的代码的基础。本书试图只用集合类 API 的基本部分。

许多教师宁愿采用更传统的方法，即首先定义、实现每种数据结构，然后才是应用。因为第二部分和第三部分的材料之间没有依赖性，因此传统的课程也能采用本书。

预备知识

使用本书的学生应该具备面向对象或面向过程的程序设计语言的知识。基本的知识包括基本数据类型、运算符、控制结构、函数（方法）以及输入和输出（但不一定要知道数组和类）。

离散数学的知识也是非常有用的，但不是绝对的预备知识。书中给出了一些数学证明，但更复杂的证明要通过复习数学知识来实现。第 3 章和第 15 章~第 20 章需要一定程度的数学水平。教师可以简单地跳过数学证明部分，而只说明结果。本书的所有证明都清晰地标记出来，与正文分开。

第 3 版所做修改小结

(1) 本书的代码使用泛型（generics）进行了完全重写，泛型是 Java 5 引入的一种技术。代码还大量使用了增强的 for 循环和自动装箱。

(2) 在 Java 5 中，优先级队列是标准集合类 API 的一部分。这个修改反映在第 17 章的讨论和第二部分的某些代码中。

本书的结构

第一部分讨论大O表示法和算法范型，包括递归和随机化。我用一章介绍排序，另一章描述基本数据结构。用集合类API讲述数据结构的接口和运行时间。至此，教师可以有几种方法介绍剩余的内容，具体包括下面两种。

(1) 在第三部分中，当描述每个数据结构时讨论对应的实现（集合类 API 的版本或者较简单的版本）。教师可以要求学生以各种方式扩展类，如习题中所建议的。

(2) 先说明如何使用集合类 API 的每个类，在课程的稍后部分再介绍实现。第二部分中的实例研究可以用来支持这种方法。因为完整的实现在每个较新的 Java 编译器上都可使用，教师可以在程序设计项目中使用集合类 API。这种方式的详细介绍将在稍后讨论。

第四部分描述了高级的数据结构，如伸展树、偶堆和不相交集数据结构。如果时间允许的话，可以讲述这些内容，或者将其作为下一门课程的内容。

逐章内容的组织

第一部分集中讲基本的算法和构件块。第 1 章提供了时间复杂度和大 O 表示法的完整讨论，还讨论和分析了二分搜索。第 2 章非常重要，因为它覆盖了集合类 API，并直观地论证了每个数据结构所支持的操作的运行时间是多少（在第三部分讨论这些数据结构的实现，包括集合类 API 风格的版本和简化的版本）。这一章还讨论了迭代器模式以及嵌套、局部和匿名类。内部类推迟到第三部分，作为实现技术讨论。第 3 章讨论递归，首先介绍归纳法证明，然后讨论分治、动态规划和回溯。有一节专门描述一些递归的数值计算算法，这些算法可用来实现 RSA 密码系统。对许多学生来讲，第 3 章后半部分的材料更适合在后续课程中学习。第 4 章对一些基本的排序算法进行了描述、编码和分析，包括插入排序、谢尔排序、归并排序和快速排序，以及间接排序。它也证明了排序算法的标准下界，并讨论了与选择相关的问题。第 5 章讨论了随机数，包括它们

的生成和在随机化算法中的使用。

第二部分提供了一些实例研究，每一章围绕一个主题。第 6 章通过游戏介绍了一些重要技术。第 7 章通过平衡符号检查算法和标准运算符优先级解析算法讨论了计算机语言中栈的使用，而且提供了两个算法的完整实现代码。第 8 章讨论了文件压缩和交叉引用生成的基本实用程序。第 9 章泛泛地介绍了模拟的问题，首先介绍了一个可视为模拟的问题，然后介绍了更一般的事件驱动模拟。最后，第 10 章说明如何用数据结构来有效地实现图中的一些最短路径算法。

第三部分介绍数据结构的实现。第 11 章讨论了作为一种实现技术的内部类，并说明它们在ArrayList实现中的使用。在第三部分的剩余章节中提供了使用简单协议（各种insert、find和remove）的实现。在某些情况下，提供了使用更复杂Java语法的集合类API实现（复杂性还源自所需操作的数量较大）。在这一部分用到了一些数学知识，特别是第 15 章~第 17 章，教师可以根据教学情况跳过这些内容。第 12 章提供了栈和队列的实现：首先用扩展数组来实现，然后用链表来实现，最后讨论集合类API的版本。第 13 章讨论通用链表。单链表用一个简单的协议来说明，使用双向链表的更复杂的集合类API版本的实现在该章的最后讨论。第 14 章描述了树，并介绍了基本的遍历方式。第 15 章非常具体，它提供了二叉查找树的一些实现。一开始说明了基本的二叉查找树，然后延伸出支持顺序统计量的二叉查找树。该章讨论了AVL树但没有给出其实现，不过实现了更实用的红黑树和AA树。然后是集合类API中TreeSet和TreeMap的实现，最后研究了B树。第 16 章讨论了散列表，并在研究了一个较简单的可选方案后，介绍了作为HashSet和HashMap一部分的二次探测法的实现。第 17 章描述了二叉堆，并研究了堆排序和外排序。Java 5 的集合类API中有优先级队列，因此我们实现了标准的版本。

第四部分的内容适用于更高级的课程或作为一般的参考，其中的算法对一年级的学生也适用。但为了完备性，这一部分包含了超出一年级学生理解范围的复杂数学分析。第 18 章描述了伸展树，它是一种在实际应用中效果非常好的二叉查找树，在某些需要优先级队列的应用中，它是二叉堆的竞争者。第 19 章描述了支持归并操作的优先级队列，并提供了偶堆的实现。最后，第 20 章研究了不相交集的标准数据结构。

章之间的依赖性

一般来讲，大多数章是互相独立的，下面是一些需要注意的依赖关系。

- 第 1 章　这一章应该在第 2 章和第 4 章之前介绍。递归（第 3 章）可以在这一章前面讲，但是，这样教师就要忽略掉某些避免低效率递归的细节问题。
- 第 2 章　这一章可以在第二部分或第三部分之前讲，也可以和它们结合在一起讲。
- 第 3 章　3.1 节~3.3 节的内容应该在讨论递归排序算法、树、Tic-Tac-Toe 实例研究和最短路径算法之前讲，而 RSA 密码系统、动态规划和回溯（须先讨论 Tic-Tac-Toe）可以选讲。
- 第 4 章　这一章应该在第 1 章和第 3 章之后讲，不过没有第 1 章和第 3 章也能介绍谢尔排序。谢尔排序不是递归的（因此不需要第 3 章），其运行时间的严格分析太复杂了，超出了本书的范围（因此几乎不需要第 1 章）。
- 第 11 章　这部分材料应该在集合类 API 的实现之前讨论。
- 第 12 章和第 13 章　这两章可以交换次序讲授。不过我宁愿先讲第 12 章，因为我相信它们比链表简单。
- 第 14 章和第 15 章　这两章可以交换次序讲授或同时讲授。

独立部分

其余部分几乎没有或完全没有依赖关系。

- 第 5 章 有关随机数的部分可以在需要时介绍。
- 第二部分 第 6 章~第 10 章可以与集合类 API 一起讲，也可以在集合类 API 后面讲，这几章本身可以是任意次序。这一部分有一些对前面几章的引用，包括 6.2 节参考了 3.7 节的讨论，8.2 节参考了 7.1 节中类似的词法分析代码。
- 第 16 章和第 17 章 这两章可以在任何时候讲。
- 第四部分 第 18 章~第 20 章的内容是独立的，通常应包含在后续课程中。

数学

我试图为强调理论的数据结构课程和需要更多分析的后续课程提供数学上的严密性。这些内容以独立的定理的形式（在某些情况下，是以独立的节或小节的形式）与书的正文相区分。在不强调理论的课程中，教师可以跳过这些内容。

在所有的情况下，定理的理解并不需要定理的证明。这是接口（定理的陈述）和它的实现（证明）分开的另一个实例。某些数学内容（如 3.4 节）可以跳过，不会影响其他部分的理解。

课程的组织

教授这门课的一个关键问题是确定如何使用第一部分到第三部分的材料。第 1 章讨论大 O 表示法。有一个习题要求学生写一个短程序并将运行时间与分析结果相比较，它可以用来测试学生的理解程度。

第 2 章的关键的思想是以分类的方式说明不同的数据结构能够以不同的效率支持不同的访问模式。任何实例研究（除了使用递归的 Tic-Tac-Toe）都可以用来说明数据结构的应用。这样学生能了解数据结构以及它们是如何使用的，但不知道它们是如何有效地实现的。这就是本书与传统方法的区别所在。以这种方式观察事物将极大地提高学生抽象思维的能力。学生还可以提供某些集合类 API 组件的简单实现（第 2 章中的习题给出了一些建议），并看到已有的集合类 API 的高效数据结构实现和他们自己写的低效数据结构实现之间的区别。也可以要求学生扩充实例研究，但他们同样不需要知道数据结构的任何细节。

数据结构的有效实现可以在后面讨论，递归可以在教师认为合适的任何时候介绍，但是必须在二叉查找树以前。排序的细节可以在递归之后的任何时候讨论。这时，可以用同样的实例研究，根据对数据结构实现的修改进行实验，继续我们的课程。例如，学生可以用不同形式的平衡二叉查找树做实验。

选择更传统的方法的教师可以在讨论了第三部分中数据结构的实现后，再讨论第二部分中的实例研究。再重申一次，本书各章的安排尽可能设计成了互相独立的。

习题

习题有各种形式，我提供了四种。基本的简答题是一些简单的问题，或需要对本书描述的算法做手工的模拟。理论题考查一些需要数学分析或者理论上具有比较有趣的解的问题。实践题包含一些简单的程序设计问题，包括有关语法或某些有技巧的代码行的问题。最后，程序设计题包

含一些扩展性习题。

教学特色

- 书中的注释（加灰底的段落）用来突出要点。
- 关键概念列出重要的术语及它们的定义。
- 每章最后的常见错误列出了经常容易犯的错误。
- 在大部分章的最后提供了可供进一步阅读的参考文献。

补充材料

本书有各种可用的补充材料。对本书的所有读者，下面的资源可以在 http://www.aw-bc.com/cssupport 找到。

- 本书的源代码文件。（在每章后面的网上资源列出了该章代码的文件名。）[1]
- 本书中所有图的 PowerPoint 幻灯片。

此外，通过验证的教师可以使用以下的补充材料。访问 http://www.aw-bc.com/computing，并按英文书名 *Data Structures and Problem Solving Using Java* 进行搜索，一旦到达了本书的目录页，选择 Instructor Resources 链接即可。

- *Instructor's Guide* 给出了获得材料的几种途径。材料包括测试题的实例、作业、教学大纲以及部分习题的答案。

致谢

在完成本书的过程中，许许多多人都给予了我帮助。在以前版本和相关 C++版本中我已向他们表示了谢意。但还有很多人给我发过电子邮件指出错误或解释的不一致，这些问题我已经在这个版本中改正了，但无法一一向他们致谢。

对本版，我要感谢 Addison-Wesley 的朋友们，他们是本书的编辑 Michael Hirsch、项目编辑 Juliet Silveri 和 Gillian Hall，以及市场部的 Michelle Brown。同时也感谢文字编辑 Penelope Hull、校对 Holly McLean-Aldis，以及封面设计 Joyce Wells。

本书的某些材料是根据我的教材 *Efficient C Programming: A Practical Approach*（Prentice Hall, 1995）改编的，并得到了出版者的允许。我已经在章后面的适当位置包含了该参考文献。

我的主页 http://www.cs.fiu.edu/~weiss 将包含更新的源代码、勘误表以及用来接受错误报告的链接。

M.A.W

于佛罗里达州迈阿密市

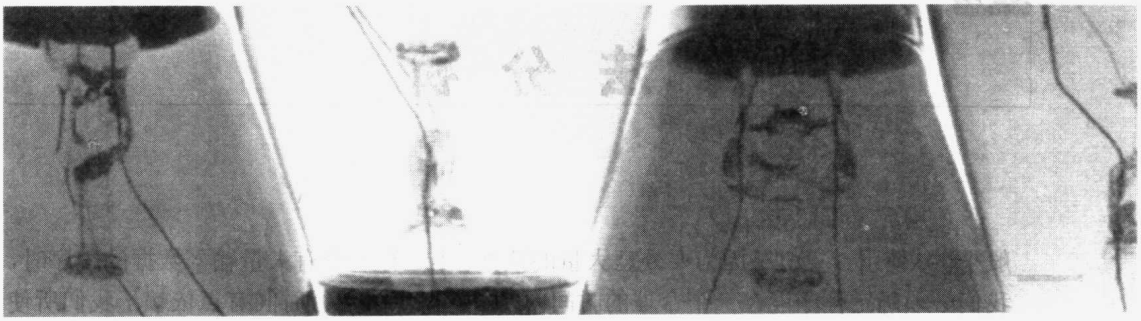
1. 本译本对英文原版进行了改编及删减，删掉了原版中的前 4 章讲述 Java 语言的内容，所以，读者下载源代码文件时要注意本书与原版各章的对应关系：第 1 章对应原版第 5 章，第 2 章对应原版第 6 章，依次类推。
——编者注

目　录

第一部分　算法和构件块

第四部分　高级数据结构

第一部分 算法和构件块



本部分内容

Part 1

第1章 算法分析

一般来讲，使用计算机是因为要处理大量的数据。当运行一个有大量输入数据的程序时，我们必须确定程序能在一个合理的时间内终止。虽然说运行时间有点依赖于我们所使用的程序设计语言，而且某种程度上还依赖于我们所使用的方法学（例如，面向过程的方法还是面向对象的方法），但通常这些因素在设计中都是不可改变的。即使这样，运行时间还是和算法的选择有很大的关系。

算法（algorithm）是一个明确指定的指令集合，计算机将按照这个指令集解决某个问题。一旦对某个问题给定了一个算法而且确定这个算法是正确的，下一步就是要确定该算法所需要的资源（比如时间和空间）量，这称为算法分析（algorithm analysis）。一个需要几吉字节内存的算法在大多数机器上都是不可行的，即使这个算法完全正确。

在本章中，我们要说明以下问题：

- 如何估计一个算法所需的时间。
- 如何使用能大幅度减少算法运行时间的技术。
- 如何使用可以更加严格地描述算法运行时间的数学框架。
- 如何写一个简单的二分搜索（binary search）程序。

1.1 什么是算法分析

任何算法运行所需的时间几乎总是取决于它必须处理的输入数据量。例如，我们预期排序10 000 个元素比排序 10 个元素需要更多的时间。因此，算法的运行时间是输入规模的函数。这个函数的准确值依赖于很多因素，如主机的速度、编译器的质量，在某些情况下还依赖于程序的质量。对于给定计算机上的给定程序，我们可以把运行时间函数画在一张图上。图 1-1 包含有 4 个程序，图中曲线代表算法分析中 4 种常见的函数：线性的、$O(N \log N)$的、平方的和立方的。输入规模 N 的范围是 1~100，运行时间的范围是 0~10 ms。快速看一下图 1-1 和图 1-2，就会发现，线性曲线、$O(N \log N)$曲线、平方曲线和立方曲线所代表的运行时间是优先级递减的。

数据越多意味着程序运行要花的时间越长。

实例之一是从因特网上下载文件的问题。假设开始有 2 s 的延迟（建立连接所需的时间），下载的速度是 1.6 KB/s。那么如果这个文件大小是 N KB，下载所需的时间可以用公式 $T(N) = N / 1.6 + 2$ 描述，这是一个线性函数（linear function）。下载一个 80KB 的文件大约需要 52 s，而下载一个 2 倍大（160 KB）的文件约需要 102 s，差不多 2 倍那么长。这种时间基本上直接正比于输入

数据量的特性，是线性算法（linear algorithm）的标志，线性算法是效率最高的算法。相反，如图 1-1 和图 1-2 所示的那样，一些非线性算法导致了大量的运行时间。比如，线性算法比立方算法的效率高得多。

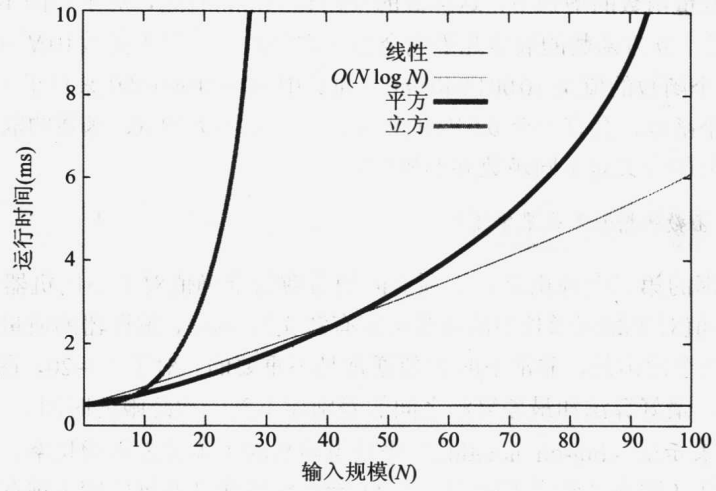


图 1-1　少量输入的运行时间

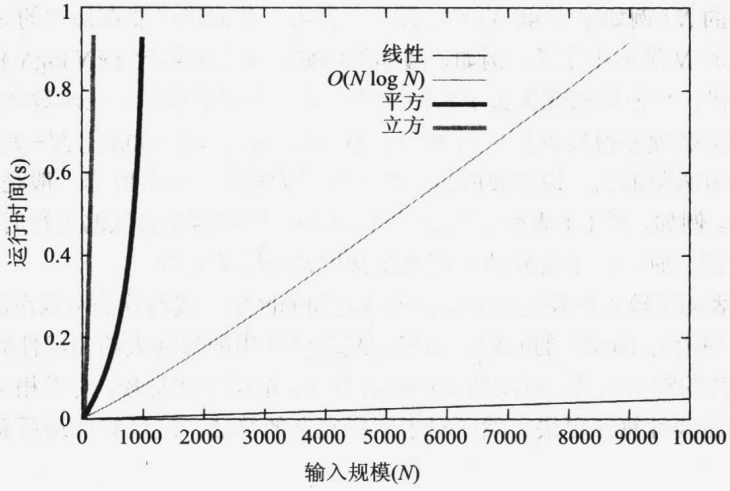


图 1-2　中等输入规模的运行时间

算法分析常见的函数中，线性算法效率最高。

本章将讲述如下几个重要问题。

- 使算法具有效率最高的曲线是不是总是很重要？
- 一条曲线比另一条曲线好多少？
- 如何判断一个特定算法的性能表示是哪条曲线？
- 如何设计算法以避免其性能表示为效率低的曲线？

立方函数（cubic function）是一个主项为 N^3 常数倍的函数。例如，$10N^3+N^2+40N+80$ 是一个立方函数。类似地，平方函数的主项是 N^2 的常数倍，线性函数的主项是 N 的常数倍。表达式 $O(N\log N)$代表了一个主项是 $N\log N$ 的函数。对数函数是一个缓慢递增的函数，例如 1 000 000

的对数（以 2 为底）仅为 20。对数比平方根或立方根等递增还要慢，我们将在 1.5 节中更深入地讨论对数。

在任意给定的点，两个函数中的任何一个都可能比另一个小，所以声称 $F(N) < G(N)$是没有意义的。我们改为度量函数的增长率，这样做的理由有三个。首先，对于如图 1-2 所示的立方函数，当 N 是 1000 时，立方函数的取值几乎完全由立方项决定。对于函数$10N^3 + N^2 + 40N + 80$，当 $N=1000$ 时，这个函数的值是 10 001 040 080，而其中 10 000 000 000 来自于 $10N^3$ 项。如果只用立方项来估算整个函数，会有一个 0.01%的误差。对于足够大的 N，函数的取值主要由它的主项决定（术语足够大的含义对不同函数并不相同）。

当 N 足够大时，函数的增长率是最重要的。

度量函数增长率的第二个理由是，主项前面的常数的准确值对于不同机器来说意义并不大（虽然该常数的相对值对于相同增长率的函数可能有意义）。例如，编译器的质量对这个常数可能有很大的影响。第三个理由是，非常小的 N 值通常是不重要的。对于 $N=20$，图 1-1 中所有的算法都在 5ms 内完成，最好算法和最差算法之间的差别还不到一次眨眼的时间。

我们使用大 O 表示法（big-oh notation）来获得函数的主项并表示增长率。例如，平方算法的运行时间为 $O(N^2)$（读为“N 平方阶”）。大 O 表示法也允许通过比较主项在函数之间建立一个相对次序，我们将在 1.4 节中更加正式地讨论大 O 表示法。

对于数值较小的 N（例如小于 40 的 N），图 1-1 表明一条曲线可能在最初的时候比另一条好，但这种情况对更大的 N 就不成立了。例如，最初的时候，平方曲线比 $O(N \log N)$曲线好，但是当 N 变得足够大的时候，平方算法就失去了它的优势。对于少量的输入，在函数之间做比较是困难的，因为这时主要常数项变得很重要。当 N 小于 50 时，函数 $N+2500$ 比 N^2 大。但最终线性函数的值总是小于平方函数的值。最重要的是，对于少量的输入，运行时间一般是无足轻重的，因此我们不需要担心。例如，图 1-1 表明了当 N 小于 25 时，所有四个算法的运行时间都少于 10ms。因此，当输入规模很小时，一个良好的原则就是使用最简单的算法。

图 1-2 清楚地表明了输入规模较大时不同曲线之间的区别。线性算法可以在远远小于 1s 的时间内解决一个输入规模为 10 000 的问题，$O(N \log N)$算法使用的时间大约是线性算法的 10 倍。注意，实际时间与包含的常数有关，可能会多些或者少些。根据这些常数，对于相当大的输入规模，$O(N \log N)$算法可能比线性算法更快。然而对于同样复杂的算法，线性算法最后总是比 $O(N \log N)$算法好。

对于平方和立方算法，这个关系就不成立了。当输入规模大于几千时，使用平方算法几乎是不可行的，而立方算法对于输入规模是几百时就不可行了。例如，对 100 000 个项使用简单的排序算法是不可行的，因为大多数简单排序算法（例如冒泡排序和选择排序）都是平方算法。第 4 章所讨论的排序算法是亚平方时间（subquadratic time）的（也就是说比 $O(N^2)$ 好），因而使得对大数组排序是可行的。

平方算法对输入规模超过几千是不可行的。

立方算法对输入规模是几百就不可行了。

这些曲线最显著的特征是，与其他算法相比，平方算法和立方算法对于合理的大输入规模是没有竞争力的。即使我们用效率很高的机器语言写一个平方算法的代码，再写一个比较差的线性