普通高等教育“十一五”国家级规划教材

Data Structures

博學

数据结构教程

主编　施伯乐

復旦大學出版社

普通高等教育“十一五”国家级规划教材

博學

数据结构教程

主编　施伯乐

编著　孙未未　汪　卫　张玥杰　陈彤兵　何震瀛

Data

復旦大學出版社

图书在版编目(CIP)数据

数据结构教程/施伯乐主编. —上海:复旦大学出版社,2011.6
ISBN 978-7-309-08164-0

Ⅰ. 数… Ⅱ. 施… Ⅲ. 数据结构-教材 Ⅳ. TP311.12

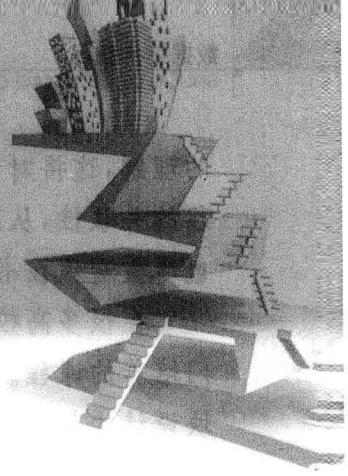
中国版本图书馆 CIP 数据核字(2011)第 103389 号

数据结构教程
施伯乐 主编
责任编辑/黄 乐

复旦大学出版社有限公司出版发行
上海市国权路 579 号 邮编:200433
网址:fupnet@ fudanpress. com http://www. fudanpress. com
门市零售:86-21-65642857 团体订购:86-21-65118853
外埠邮购:86-21-65109143
上海浦东北联印刷厂

开本 787×1092 1/16 印张 17.5 字数 404 千
2011 年 6 月第 1 版第 1 次印刷
印数 1—4 100

ISBN 978-7-309-08164-0/T·417
定价:35.00 元



前言
FOREWORD

数据结构是计算机科学和技术中的一门基础课程，是学习操作系统、数据库、编译原理、算法分析等课程的基础。数据结构课程的前导课程是程序设计和离散数学。本书选择用C++语言描述数据结构和算法实现，因此要求学生掌握面向对象的设计方法，以及用C++语言编写程序的能力。本书第1章中介绍了学习本书所必需的C++语言和面向对象方法的基础知识，如果学生在前导程序设计课程中学习的是C语言，则可以从1.3节开始学习，否则可以跳过这部分内容。在学习树和图有关的数据结构知识时，学生如果已经修学过离散数学课程会更容易理解，否则需要教师在上课时补充必要的基础知识。为减少与后续“算法分析”课程的重复，在算法性能分析时，本书原则上不进行形式化证明，不证明较难的结果。

随着技术的不断进步，数据结构的内容也在不断更新和扩展。作为教材，不可能详细介绍每种数据结构的最新研究进展，但我们在每章最后都有一节“进阶导读”，增加进阶和最新研究成果的介绍，并列出参考文献，用于引导学有余力的学生进一步学习和本章内容相关的数据结构知识。这些内容都是本书作者建议读者去了解的，我们鼓励勤于钻研的学生做较深入的自学。

虽然本书选择用C++语言描述数据结构和算法实现，但数据结构的思想和方法，并不和某种特定的语言绑定，书中介绍的数据结构和算法都可以方便地改用Java、C#、C语言等实现。

针对目前很多教材中程序代码偏多的问题，我们着重对数据结构核心思想和理论基础进行介绍。在教材中将着重对各种不同数据结构和方法优缺点的分析，以及对算法流程的介绍和分析，让学生能够对数据结构的基本理论有更深入

的理解，这将对学生提高理论水平、把握课程在整个学科中的地位以及与其他课程的关联性，从而对学科有一个大局观念发挥重要的作用。同时，我们在写作中也尽量避免在书中出现大段代码，更多地结合例子来讲算法思路。

在很多高校的教学体系中，数据结构是最后一门对学生的编程能力进行专门培养的课程。为了提高学生的动手能力，本教材一方面采用从问题分析到模块设计的整个过程进行论述的方法，以提高学生的分析能力，另一方面每章提供一批编程题目以备学生训练之用。考虑到各高校对上机实验课程设置和要求差异很大，本教材不包括对实验课程的具体编排。

本书内容的选取遵照教育部硕士研究生入学统考考试大纲和复旦大学计算机学院数据结构课程大纲进行，覆盖了线性表、树、图、堆、查找和索引、排序等经典内容，具体包括如下10章内容：

第1章基础。本章是开始学习数据结构的基础，内容包括数据结构和抽象数据类型的概念；程序性能分析的方法，通过对程序时间复杂度和空间复杂度分析来评价数据结构和算法的优劣，使用“大O”描述法；C++语言和面向对象方法基础（本内容适合没有系统学习过C++编程的读者学习）。

第2章线性表。线性表是最简单也是最常用的一种数据结构，包括顺序表示和链式表示两种方式。作为对基本数组和链表的扩展，还介绍了稀疏矩阵、循环链表和双向链表等数据结构。第3章和第4章介绍的串、栈和队列，在本质上都是线性表，由于它们有各自鲜明的特点，因此分别独立成章介绍。

第3章串。先介绍串基本概念及定义、表示和实现方法，然后介绍一些常用的串匹配算法，包括BF、KR、KMP、BM算法。

第4章栈和队列。栈是一种后进先出的顺序表，队列是一种先进先出的顺序表。本章介绍了栈和队列的基本操作、分别用数组和链表实现的方法、应用举例以及STL中的实现。

第5章递归和广义表。本章介绍了递归的思想、递归程序改造成非递归程序的几种方法、广义表基础。

第6章树、二叉树和森林。本章介绍了树、二叉树、森林的基本概念、表示和操作，包括二叉树的各种遍历操作、线索化二叉树，以及用树这种数据结构来实现堆和霍夫曼编码。

第7章查找与索引。本章首先介绍了基于顺序表的顺序查找、折半查找、插值查找和斐波那契查找，然后重点介绍了几种用于查找的树结构，包括二叉查找树、B—树和B+树，Tier树以及散列结构。

第8章图。本章在介绍了图的基本概念、存储结构之后，介绍了图的遍历、求连通分量、求生成树、求最短路径、拓扑排序和关键路径等算法。

第9章排序和第10章外部排序，分别介绍基于内存的内排序和基于磁盘等外存的外排序。在内排序中，除了简单的几种排序方法之外，介绍了归并排序、快速排序、堆排序、希尔排序、基数排序等5种不同的排序方法，并做了综合对比，最后介绍了STL中实际采用的排序技术。外排序因为存储介质特性不同，所采用的方法和内排序完全不同。第10章分磁盘文件外排序和磁带文件外排序两类，分别介绍了基本的排序方法。

作者所在的教学团队，以施伯乐教授为带头人，自20世纪80年代开始编写数据结构课程的教材，分别在1988年和1995年由复旦大学出版社出版，被复旦大学和其他兄弟高校选为计算机科学与技术等专业的教材，读者反馈良好，曾获上海市优秀教材一等奖和华东地区优秀教材一等奖等奖项。经过10余年的使用，教材的配套课件、实验和题库等教学手段得到了进一步的完善，主编及教材的主要编写者一直在从事该课程的教学工作，积累了相当多的教学心得和修订素材。在本版中，我们希望融入新的教学理念，补充部分新内容，编写出更适合目前计算机科学与技术专业本科教学需求的新版数据结构教材。

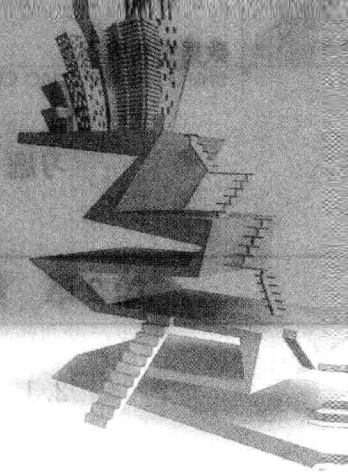
本书的编写工作由复旦大学首席教授施伯乐主持，确定了本书编写大纲和写作风格，并最后统稿；汪卫、孙未未、张玥杰、陈彤兵和何震瀛执笔编写。由于作者水平和时间所限，书中难免有错误和疏漏之处，希望读者不吝指正。在本书的编写和出版过程中，得到了复旦大学出版社的大力支持，尤其是本书责任编辑黄乐老师的高效工作给了我们很大帮助，在此表示衷心感谢！

编　者

2010年年底

目录

CONTENTS



第1章 基 础

数据结构是计算机科学和技术中的一门基础课程，是学习操作系统、数据库、编译原理、算法分析等课程的基础。数据结构课程的前导课程是程序设计和离散数学，后续相关课程有算法分析等。

1.1 什么是数据结构

本书是一本学习数据结构的教材，那么首先的一个问题是：什么是数据结构？

数据结构(data structures)指的是数据之间的相互关系，即计算机中存储和组织数据的形式。通常可以用一个二元组<D，R>来表示，或写成 DS=<D，R>，其中 D 是数据集合，R 是 D 中数据元素之间所存在的关系的集合。对于数据集合 D，如果 D 中的数据元素之间存在着不同的关系集合，如 R1 和 R2，那么 DS1=<D，R1>，DS2=<D，R2>是两个不同的数据结构。数据集合中的数据元素可以由若干数据项组成。数据结构就是根据各种不同的数据集合和数据元素之间的关系，研究如何表示、存储和操作(查找、插入、删除、修改)这些数据的技术。

通常情况下，精心选择的数据结构可以带来最优效率的算法。这也是评价数据结构优劣的标准。本章将随后介绍相关的评价标准和评价方法。

在将现实世界的问题转化为计算机世界的实现的过程中，为了帮助消除现实世界和计算机世界在问题表示上的巨大差别，先将问题转换为概念世界中的逻辑模型，然后再由此在计算机世界中实现。抽象数据类型就是这样一种逻辑模型，强调对该数据类型的操作接口，将数据结构的具体实现封装隐藏于受限接口后方。

最终，数据结构要以程序设计语言来实现。本书选择 C++语言作为描述工具，因为 C++中体现了抽象数据类型强调的抽象和封装等思想。本章将在 1.3 节和 1.4 节介绍抽象数据类型和 C++语言的基础知识。

1.2 程序性能分析

1.2.1 程序性能的衡量标准

在程序中算法和数据结构相互配合，而程序的性能是衡量一个程序好坏的重要指标，一般需要从时间复杂性和空间复杂性两个指标对程序的性能进行评价。所谓时间复杂性(time complexity)是一个算法执行时所需要消耗的时间，而空间复杂性(space complexity)是指程序在执行过程中相关的数据结构和变量所占据的内存和磁盘上的空间。影响一个程序性能的因素有很多，主要包括：

(1) 使用的算法和数据结构的合理性；

(2) 运行程序的计算机硬件环境；

(3) 编程语言和编程环境；

(4) 程序的质量和技巧的使用。

本书将着重从数据结构和算法的角度对程序的性能进行分析。程序性能的评估方法分为复杂性分析和事后测试两种途径，其中前者是通过对程序中语句的执行代价和次数进行分析，预测程序的执行代价。后者是通过在程序中加入相应的函数获取程序执行所需的时间和空间代价。

1.2.2 程序的事后测试

在程序的后期测试中可以在代码的相应位置添加时间或空间的获取函数来得到程序执行实际消耗的时间和空间代价。例如，我们可以通过增加 time()函数来获得某段程序执行的开始和终止时间。程序 1-1 给出了一个 a 和 b 两个矩阵相乘的程序，它将 $a[N][N]$和 $b[N][N]$两个 $N\times N$ 矩阵相乘，最终的结果放在 $c[N][N]$中返回给用户。

程序 1-1 矩阵乘法

```
void matrix_multi(int a[N][N], int b[N][N], int c[N][N])
{
    int i, j, k;
    for (i=0;i<N;i++)
        for(j=0;j<N; j++)
            c[i][j]=0;
    for (i=0;i<N;i++)
        for(j=0;j<N; j++)
            for (k=0;k<N; k++)
                c[i][j]=a[i][k]* b[k][j]+ c[i][j];
}
```

读者可以针对不同的矩阵大小进行测试。为了获取程序执行的时间，在每次程序开始

执行的地方记录程序执行的起始时间(利用 time 函数),在终止的时候记录其终止时间,这样就可以获得程序的具体执行时间。增加了 time 函数后的程序为:

程序 1-2 增加计时的矩阵乘法

```
void multi()
{
    int a[N][N], b[N][N], c[N][N];
    int i,j,k;
    k=0;
    for (i=0;i<N;i++)
        for (j=0;i<N;i++)
            a[i][j]=k++;
    for (i=0;i<N;i++)
        for (j=0;i<N;i++)
            b[i][j]=k++;
    long start,stop;
    time(&start);
    matix-multi(a, b, c);
    time(&stop);
    long runtime=stop-start;
    cout <<" time is " << runtime << endl;
}
```

通过设定不同的 N 的取值,我们可以获得测试的结果如表 1-1 所示。

表 1-1 矩阵乘法测试结果

N	10	20	50	100	200	500	1 000	1 500	2 000
运行时间(ms)	0.005	0.09	0.63	6	58	982	10 841	37 963	88 514

由于时间函数往往有精度的限制,同时机器也有快慢的差别。有时通过设置时间函数获得的时间值有可能低于时间函数的最小精度,例如假设 time()函数的最小精度是毫秒,而程序的运行时间是微秒级,从而无法获得程序的具体运行时间,这时,可以采用重复运行同一段时间的方法来获得它的运行时间。例如表 1-1 中 N 取 10 时的计算时间就是通过多次重复获得的。

在程序执行过程中消耗的空间代价包括:

(1) 程序代码的存储空间;

(2) 程序中常量和变量的存储空间;

(3) 程序动态申请的空间。

前两部分的空间代价是常数,在程序运行之前就已经可以计算出来了,例如在上例中三个矩阵变量占据的空间分别为 $4\times N\times N$、$4\times N\times N$、$4\times N\times N$。对于程序动态申请的空间,可以在程序每次向系统申请空间的时候,累计申请空间的大小,在释放空间时减去

相应的空间的方法来获得。通过对累计值的监控，就可以获得程序执行所需要的最大空间开销。

1.2.3 时间复杂性的计算方法

一个程序运行所需要的时间包括程序导入的时间和程序中每条语句执行的时间。其中程序导入的时间基本上是固定的，但是程序运行的时间则不是固定的，不同的数据规模对应的程序的执行时间是不同的，从上节的例子中就可以看到这一点。程序的时间复杂性分析是一种事前分析的方法，通过对程序语句的执行次数进行分析，从而从整体上分析程序的时间复杂性。即在程序执行之前对程序的每条原子语句和执行次数进行计算以获得整个程序的执行代价。所谓原子语句是指该语句的执行代价是一个常数，例如在矩阵相乘的算法中，每条赋值语句和 for 语句的执行代价都是常数。但是调用矩阵相乘的程序的执行代价就不是常数，所以不是原子语句。可以通过对原子语句的执行次数进行累加的方式获得整个程序的执行代价，我们假设每条原子语句的执行代价分别为：C_1、C_2、C_3、C_4、C_5、C_6、C_7、C_8、C_9、C_{10}，表 1-2 列出了程序每条语句的执行次数。

表 1-2 程序语句的执行代价及次数

程　　序	执行代价	执行次数
`int i, j, k;`	C_1	1
`for (i=0;i<N;i++)`	C_2	$N+1$
`  for(j=0;j<N; j++)`	C_3	$N(N+1)$
`  c[i][j]=0;`	C_4	$N\times N$
`for (i=0;i<N;i++)`	C_5	$N+1$
`  for(j=0;j<N; j++)`	C_6	$N(N+1)$
`  for (k=0;k<N; K++)`	C_7	$N\times N(N+1)$
`  c[i][j]=a[i][k]* b[k][j]+ c[i][j];`	C_8	$N\times N\times N$

从表 1-2 中可以计算出整个程序的复杂性是 $C_1+(N+1)C_2+N(N+1)\ C_3+(N\times N)C_4+(N+1)C_5+N(N+1)C_6+(N\times N(N+1))C_7+(N\times N\times N)C_8=(C_7+C_8)N^3+(C_3+C_4+C_6+C_7)N^2+(C_2+C_3+C_4+C_5)N+(C_1+C_5)$。其中 C_1、C_2、…、C_8 是常量，N 是表示数据量规模的变量。

1.2.4 空间复杂性的计算方法

一个程序运行所需要的空间包括程序代码本身的规模、初始的数据结构的大小和程序在执行过程中临时申请的空间三部分。其中前两部分基本上是固定的，所以空间复杂性分析主要是通过对第三部分程序执行过程中对临时申请的空间进行分析，这也是一种事前分析的方法，即在程序执行之前对程序每次申请和释放的空间进行统计。例如程序 1-3：