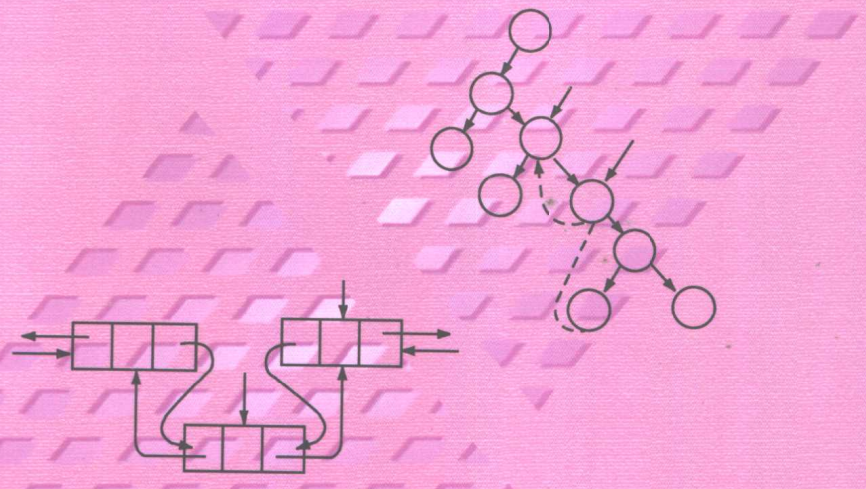
21世纪大学课程辅导丛书

数据结构
学习指导与典型题解

朱战立　张选平　编著



西安交通大学出版社

21世纪大学课程辅导丛书

数据结构

学习指导与典型题解

朱战立　张选平　编著

西安交通大学出版社

内 容 提 要

计算机是广泛使用的工具，数据结构课程是学习计算机软件设计的基础课程。本书是作者在长期教学经验积累的基础上精心编著的数据结构课程的学习参考书。全书共分 10 章，各章主要由学习指导、典型题解和上机实习题解三部分组成。学习指导部分综述该章的学习要点；典型题解部分的例题是作者精心编选的，具有典型意义；上机实习题解部分是专为解决学生感觉完成上机实习题比较困难，以及自学学生上机实习不能保证而设计的。另外，附录中收录了一些本科生、研究生和自学考试的试卷。书中的算法用 C 语言描述。

本书是计算机本科和专科学生、报考计算机专业硕士研究生的考生、参加国家高等教育自学考试的考生、参加高等学校专升本考试的考生、参加计算机等级三级和四级考试的考生的非常适宜的学习参考书。

图书在版编目(CIP)数据

数据结构：学习指导与典型题解释/朱战立，张选平编著 .—西安：西安交通大学出版社，2002.3
ISBN 7-5605-1501-0

Ⅰ. 数… Ⅱ. ①朱…②张… Ⅲ. 数据结构-高等学校-教学参考资料 Ⅳ. TP311.12

中国版本图书馆 CIP 数据核字(2002)第 005599 号

*

西安交通大学出版社出版发行
(西安市兴庆南路 25 号 邮政编码：710049 电话：(029)2668315)
西安向阳印刷厂印装
各地新华书店经销

*

开本：787 mm×1 092 mm 1/16 印张：19.625 字数：400 千字
2002 年 3 月第 1 版 2002 年 3 月第 1 次印刷
印数：0 001～5 000 定价：23.00 元

发行科电话：(029)2668357，2667874

前　言

计算机是广泛使用的工具，数据结构课程是学习计算机软件设计的基础课程。本书作者在高校教学已有15年之久，近10年来又一直讲授数据结构课程，所以，本书是作者在长期教学经验积累的基础上精心编著的数据结构课程的学习参考书。

全书共分10章，包括了表、堆栈、队列、数组、串、广义表、树、二叉树、图、递归程序设计、排序、查找等典型数据结构课程内容。每章主要由学习指导、典型题解和上机实习题解三部分组成。学习指导部分综述该章的学习要点，是该章学习的总结和考试复习的大纲；典型题解部分的例题是作者精心编选的，具有典型意义，作者在解题时注意结合基本概念进行分析，对读者理解学习内容和掌握解题方法很有帮助；数据结构课程学习要与上机实习过程相结合，不少学生在完成上机实习题时感觉比较困难，另外许多自学学生上机实习难以保证，解部分是专为解决上述学生的问题设计的。另外，本书的附录部分收录了一些高校本科生考卷、研究生考卷、国家高等教育自学考试考卷、高等学校专升本考试考卷，这些材料既可作为读者的学习练习，也可作为读者的考试前自测。

书中的算法用C语言描述，所有算法和上机实习题程序都是在计算机上调试通过的。

本书可作为计算机本科和专科学生、报考计算机专业硕士研究生的考生、参加国家成人自学考试的考生、参加计算机等级三级和四级考试的考生的非常适宜的学习参考书。

本书的第7,8,9章和附录由张选平编写，其余各章由朱战立编写，全书由朱战立修改定稿。尽管作者在写作过程中非常认真和努力，但由于水平有限，错误和不足之处在所难免，敬请读者批评指正。

编　著　者

2001.6

朱战立 副教授，1982年毕业于上海复旦大学计算机科学系。曾作为访问学者在英国曼彻斯特理工学院学习访问半年。长期从事计算机学科的教学和科研工作。已出版了3本教材和1本专著，发表论文 20多篇。已出版的数据结构教材有：《数据结构 —— 使用 C 语言》第1版和第2版，《数据结构 —— 使用 C++ 语言》。其中，前两本教材获部级优秀教材三等奖。

张选平 副教授，1984年毕业于西安交通大学计算机软件专业，1987年硕士毕业于中国科技大学，现任教于西安交通大学电信学院计算机系，从事数据结构教学多年。参与撰写教材及教学参考书3本，发表论文10余篇。

数据结构

学习指导与典型题解

目　录

第 7 章　树与二叉树

第 8 章　图

第 9 章　内部排序

第 10 章　查找

附录　典型试题和解答

第1章 概 述

本章讨论的数据结构的基本概念和方法将贯穿数据结构课程的整个学习过程。本章主要对数据结构课程学习中将遇到的基本概念和方法作概括性的讨论，其中1.1节概述了数据结构的基本概念，1.2节讨论了抽象数据类型的概念和具体表示方法，1.3节从数学定义角度出发对难于理解的算法的效率作了定义。算法设计一直是初学者最感头痛的内容，1.4节通过实际例子讨论了算法设计要考虑的问题。数据结构课程的任务之一是训练学生的软件设计能力，并要求软件设计过程按规范进行，1.5节给出的算法书写规范和1.6节给出的实习报告书写规范就是为此目的编写的。

1.1 数据结构的基本概念

定义1-1 数据是人们利用文字符号、数字符号以及其他规定的符号对现实世界的事物及其活动所作的抽象描述。

定义1-2 表示数据的基本单位称作数据元素。数据元素通常由若干个数据项组成。

例如要描述学生的情况，我们可设计姓名、性别、年龄等数据项来进行描述。描述一个具体学生的一行数据称作一个数据元素，描述的学生不同则数据元素的值不同。

当我们讨论抽象的数据元素时，我们用符号 $a_0, a_1, \cdots, a_{n-1}$ 表示有 n 个数据元素的抽象数据元素。当设计问题具体确定时，这些抽象数据元素 $a_0, a_1, \cdots, a_{n-1}$ 即可转变为相应具体的数据元素值。抽象数据元素值既可以是高级语言已定义的数据类型，也可以是自己新定义的抽象数据类型。当要具体登记某学校的学生情况时，抽象数据元素 $a_0, a_1, \cdots, a_{n-1}$ 即可转变为具体学生的数据元素值。

定义1-3 数据之间的相互联系称为数据的逻辑结构。

按照数据之间的相互联系方式，数据的逻辑结构可分为线性结构、树结构和图结构。线性结构除第一个和最后一个数据元素外每个数据元素只有一个前驱数据元素和一个后继数据元素，线性结构如图1.1(a)所示。由于线性结构的数据元素呈现为线性序列，所以有些教科书也称线性结构为线性序列，或简称序列。树结构除根结点外每个数据元素只有一个前驱数据元素和若干个后继数据元素，树结构如图1.1(b)所示。图结构的每个数据元素可有若干个前驱数据元素和若干个后继数据元素，图结构如图1.1(c)所示。

定义1-4 一种数据的逻辑结构在计算机中的存储方式称为数据的存储结构，或称数据的物理结构。

数据存储结构的基本形式有两种：一种是顺序存储结构；另一种是链式存储结构。顺序存储结构是把数据元素存储在一块连续地址空间的内存中，其特点是逻辑上相邻的数据元素在物理上也相邻，数据间的逻辑关系表现在数据元素的存储位置关系上。用高级程序设计语言实现顺序存储结构的基本方法是使用数组。图1.2(a)是线性结构数据元素 $a_0, a_1, \cdots, a_{n-1}$ 的

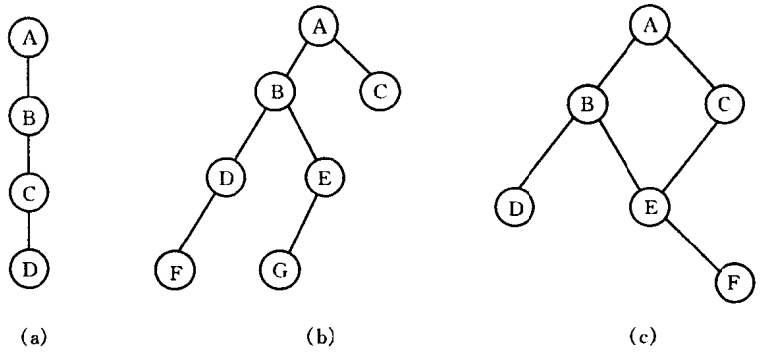


图 1.1　基本的数据逻辑结构形式
(a)线性结构；　(b)树结构；　(c)图结构

顺序存储结构形式。

链式存储结构是使用指针把发生联系的数据元素链接起来，其特点是逻辑上相邻的数据元素在物理上不一定相邻，数据间的逻辑关系表现在结点的链接关系上。图 1.2(b)是线性结构数据元素 $a_0, a_1, \cdots, a_{n-1}$ 的链式存储结构形式。

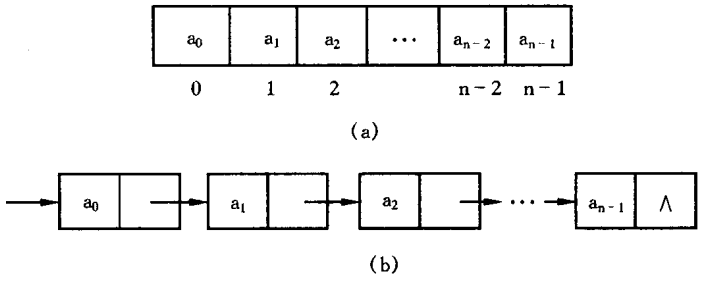


图 1.2　基本存储结构形式
(a) 顺序存储结构；　(b) 链式存储结构

顺序存储结构和链式存储结构是两种最基本、最常用的存储结构。除此而外，还有仿真指针(也称作静态数组)和间接地址存储结构。

定义 1-5　对每一种数据需要进行的处理称为数据的操作，对一个数据所有可能的操作称为数据的操作集合。

例如，对学生情况数据可能进行的处理有：插入一条数据元素，删除一条数据元素，列出所有数据元素的值，等等。

定义 1-6　类型是一组值的集合。

例如，整数类型就是具体计算机所允许表示的整数数值的集合，通常是 -32 767～32 767。又例如，布尔类型就是 true 和 false 组成的集合。

定义 1-7　数据类型是指一个类型和定义在这个类型上的操作集合。

例如，当我们说计算机中的整数数据类型时，我们就不仅指计算机所允许表示的整数数值的集合，而且指能对这个整数类型进行的加(+)、减(-)、乘(*)、除(/)和求模(%)操作。

在本课程中，通常把在已有的数据类型基础上设计新的数据类型的过程称作数据结构设

计，在这里，数据结构和数据类型含义相同。

1.2 抽象数据类型

定义 1-8 抽象数据类型(abstract data type，缩写为 ADT)是指一个逻辑概念上的类型和这个类型上的操作集合。

换句话说，抽象数据类型是在高级程序设计语言中已有的基本数据类型的基础上或在用户已定义的抽象数据类型的基础上，用户所定义的新的数据类型。利用抽象数据类型可以构建出软件设计更高一级的平台，简化软件设计的步骤。

在软件设计中，抽象数据类型定义属于最重要的软件设计的一个文档。抽象数据类型的定义在形式上可简可繁，最为规范的抽象数据类型定义形式如下：

```
ADT 抽象数据类型名 is
    数据：
        数据元素集合和数据元素间关系以及数据元素意义的描述
    操作：
    操作名 1：
        输入：该操作的调用参数描述
        前置条件：执行此操作前系统必须的状态
        功能：该操作实现的功能描述
        输出：操作名的返回值描述
        后置条件：执行此操作后系统的状态描述
    操作名 2：
        ⋮
        ⋮
end ADT 抽象数据类型名
```

下面，我们给出一个简单的抽象数据类型 Circular 的定义，并假设该抽象数据类型Circular允许的操作只包括求圆的面积。

```
ADT Circular is
    数据：
        radius：圆的半径，为非负整数
    操作：
        Area：
        输入：无
        前置条件：无
        功能：计算半径为 radius 的圆的面积
        输出：计算出的圆的面积
        后置条件：无
end ADT Circular
```

抽象数据类型的定义是设计者之间的设计接口。一方面，根据抽象数据类型的定义可以

完成抽象数据类型的软件实现;另一方面,软件设计中凡是需要使用抽象数据类型的地方,可以方便地根据抽象数据类型的定义来使用它们。

抽象数据类型设计是软件设计的基础。数据结构课程中讨论的各种基本数据结构,如表、堆栈、队列、串、树、二叉树、图等,都可看作是设计更复杂软件的抽象数据类型。数据结构课程安排有相当数量的上机实习,任何上机实习的设计都应首先设计出相应的抽象数据类型。

各种基本数据结构(如堆栈和队列)是各种应用程序编程的基本成分,当应用程序中所要用到的各种基本数据结构都已有设计好的可重复使用的抽象数据类型时,以后的程序设计和程序维护将会大大简化。此外,我们在分析和设计应用程序时,若能识别出各种可重复使用的部件时,我们可将这些部件设计成可重复使用的抽象数据类型,这样,以后的程序设计和程序维护将进一步更加简化。

1.3 算法和算法的时间复杂度

定义 1-9 算法是对特定问题求解步骤描述的计算机指令的有限序列。

算法满足以下性质:

(1) 输入性:具有零个或若干个输入量;

(2) 输出性:至少产生一个输出或执行一个有意义的操作;

(3) 有限性:计算机的指令执行序列是有限的;

(4) 确定性:每一条指令的含义明确,无二义性;

(5) 可执行性:每一条指令都应在有限的时间内完成。

算法设计方法和算法复杂度分析的学习是数据结构课程学习的一个重点,也是一个难点。

定义 1-10 算法的时间复杂度是问题规模的函数,算法的时间复杂度也称作算法的时间效率。

一个特定问题可以有多种算法来解决,具体选用哪种算法来解决需要做算法的时间复杂度分析和空间复杂度分析,在目前的计算机硬件条件下,决定哪种算法更合适的主要因素是算法的时间复杂度分析结果。

算法的时间复杂度分析通常采用 $O(f(n))$ 表示法($O(f(n))$ 读作大 O 的 $f(n)$)。

定义 1-11 $T(n)=O(f(n))$ 当且仅当存在正常数 c 和 n_0,对所有的 $n(n\geqslant n_0)$ 满足 $T(n)\leqslant c\times f(n)$。

换句话说,$O(f(n))$ 给出了函数 $f(n)$ 的上界。

由于上述定义中对所有的 $n(n\geqslant n_0)$,只要 n 比较大一般均成立,而我们考虑的算法的时间复杂度也主要是在数据个数 n 相当大时的情况,所以我们具体分析一个算法的时间复杂度 $T(n)$ 时一般不考虑 n 为一个较小的数时 $T(n)\leqslant c\times f(n)$ 不成立的情况。

令函数 $T(n)$ 为算法的时间复杂度,其中 n 为算法处理的数据个数。则 $T(n)=O(f(n))$ 表示算法的时间复杂度随数据个数 n 的增长率与函数 $f(n)$ 的增长率相同,或者说两者具有相同的数量级。

当算法的时间复杂度 $T(n)$ 与数据个数 n 无关系时,$T(n)\leqslant c\times 1$,所以此时算法的时间复杂度 $T(n)=O(1)$;当算法的时间复杂度 $T(n)$ 与数据个数 n 为线性关系时,$T(n)\leqslant c\times n$,所以此时算法的时间复杂度 $T(n)=O(n)$;当算法的时间复杂度 $T(n)$ 与数据个数 n 为

平方关系时，$T(n) \leqslant c \times n_2$，所以此时算法的时间复杂度 $T(n) = O(n_2)$；依此类推，还有 $O(n_3)$，$O(\text{lb}n)$，$O(2_n)$ 等等。可见，分析一个算法中基本语句的执行次数就可求出该算法的时间复杂度。

算法的时间复杂度分析是本门课程学习的一个难点，下面的四个例题是算法时间复杂度分析的四种典型情况。

例 1-1 设数组 a 和 b 在前边部分已赋值，求如下两个 n 阶矩阵相乘运算程序段的时间复杂度。

```
for(i=0; i<n; i++)
    for(j=0; j<n; j++)
    {
        c[i][j]=0;                                  /* 基本语句 1 */
        for(k=0; k<n; k++)
            c[i][j]=c[i][j]+a[i][k] * b[k][j];      /* 基本语句 2 */
    }
```

解 设基本语句的执行次数为 $f(n)$，有

$$f(n) = n^2 + n^3$$

因程序段的时间复杂度 $T(n) = f(n) = n^2 + n^3 \leqslant c \times n^3 = O(n^3)$，其中 c 为常数，所以该程序段的时间复杂度为 $O(n^3)$。

例 1-2 设 n 为如下程序段处理的数据个数，求下面程序段的时间复杂度。

```
for(i=1; i <= n; i=2 * i)
  printf("i=%d\n", i);      /* 基本语句 */
```

解 设基本语句的执行次数为 $f(n)$，有 $2f(n) \leqslant n$，即有 $f(n) \leqslant \text{lb}n$。

因程序段的时间复杂度 $T(n) = f(n) \leqslant \text{lb}n \leqslant c * \text{lb}n = O(\text{lb}n)$，其中 c 为常数，所以该程序段的时间复杂度为 $O(\text{lb}n)$。

在很多情况中，算法中数据元素的取值情况不同，则算法的时间复杂度也会不同。此时，算法的时间复杂度应是数据元素最坏情况下取值的时间复杂度或数据元素等概率取值情况下的平均(或称期望)时间复杂度。

例 1-3 下边的算法是用冒泡排序法对数组 a 中的 n 个整数类型的数据元素(a[0]~a[n-1])从小到大排序的算法，求该算法的时间复杂度。

```
void BubbleSort(int a[], int n)
{
int i, j, flag=1;
int temp;

for(i=1; i<n && flag==1; i++)
{
    flag=0;
    for(j=0; j<n-i; j++)
    {
```

```
            if(a[j]>a[j+1])
            {
                flag=1;
                temp=a[j];
                a[j]=a[j+1];
                a[j+1]=temp;
            }
        }
    }
```

解 这个算法的时间复杂度随待排序数据的不同而不同。当某次排序过程中没有任何两个数组元素交换位置，则表明数组元素已排序完毕，此时算法将因标记 flag=0 不满足循环条件而结束。但是，在最坏的情况下，每次排序过程中都至少有两个数组元素交换位置，因此，应按最坏情况计算该算法的时间复杂度。

设基本语句的执行次数为 $f(n)$，最坏情况下有

$$f(n) \approx n + 4 \times n^2/2$$

因算法的时间复杂度 $T(n) = f(n) \approx n + 2 \times n^2 \leqslant c \times n^2 = O(n^2)$，其中 c 为常数，所以该算法的时间复杂度为 $O(n^2)$。

例 1-4 下边算法描述在一个有 n 个数据元素的数组 a 中删除第 i 个位置的数组元素，要求当删除成功时数组元素个数减 1，求该算法的时间复杂度。其中，数组下标从 0 至 $n-1$。

```
int Delete(int a[], int *n, int i)
{
int j;
if(i<0 || i>*n) return 0;        /*删除位置错误,失败返回*/
for(j=i+1; j<*n; j++) a[j-1]=a[j];      /*顺次移位填补*/
(*n)--;                          /*数组元素个数减1*/
return 1;                        /*删除成功返回*/
}
```

解 这个算法的时间复杂度随删除数据的位置不同而不同。当删除最后一个位置的数组元素时有 $i=n-1$，$j=i+1=n$，此时因不需移位填补而循环次数为 0，当删除到最后一个位置的数组元素时有 $i=n-2$，$j=i+1=n-1$，此时因只需移位填补一次而循环次数为 1，依此类推，当删除第一个位置的数组元素时有 $i=0$，$j=i+1=1$，此时因需移位填补 $n-1$ 次而循环次数为 $n-1$。因此，算法的时间复杂度应是删除数据的位置等概率取值情况下的平均时间复杂度。

假设删除任何位置上的数据元素都是等概率的(一般情况下均可作等概率假设)，设 Pi 为删除第 i 个位置上数据元素的概率，则有 $P_i = 1/n$，设 E 为删除数组元素的平均次数，则有

$$E = \frac{1}{n}\sum_{i=0}^{n-1}(n-1-i) = \frac{1}{n}[(n-1)+(n-2)+\cdots+2+1+0]$$

$$= \frac{1}{n} \cdot \frac{n(n-1)}{2} = \frac{n-1}{2}$$

因该算法的时间复杂度 $T(n) = E \leqslant (n+1)/2 \leqslant c \times n = O(n)$，其中 c 为常数，所以该算法的时间复杂度为 $O(n)$。

算法的时间复杂度是衡量一个算法好坏的重要指标。一般来说，具有多项式时间复杂度的算法是可接受、可实际使用的算法；具有指数时间复杂度的算法，是只有当 n 足够小时才可使用的算法。表 1.1 给出了多项式增长与指数增长的比较。从表 1.1 可以看出，当 $n = 50$ 时，多项式函数 $n^3 = 125\,000$，而指数函数 $2^n = 1.0 \times 10^{15}$，$n! = 3.0 \times 10^{64}$，$n^n = 8.9 \times 10^{84}$。

表 1.1　多项式增长和指数增长的比较

大小	多项式			指数		
n	n	n^2	n^3	2^n	$n!$	n^n
1	1	1	1	2	1	1
2	2	4	8	4	2	4
3	3	9	27	8	6	27
4	4	16	64	16	24	256
5	5	25	125	32	120	3 125
6	6	36	216	64	720	46 656
7	7	49	343	128	5 040	823 543
8	8	64	512	256	40 320	16 777 216
9	9	81	729	512	362 800	3.9E8
10	10	100	1 000	1 024	3 628 800	1.9E10
⋮	⋮	⋮	⋮	⋮	⋮	⋮
20	20	400	8 000	1 048 376	2.4E18	1.0E25
30	30	900	27 000	1.0E9	2.7E32	2.1E44
40	40	1 600	64 000	1.0E12	8.2E47	1.2E64
50	50	2 500	125 000	1.0E15	3.0E64	8.9E84
⋮	⋮	⋮	⋮	⋮	⋮	⋮
100	100	10 000	1.0E6	1.3E30	9.3E157	1.0E200

通常，当基本语句的计算次数超过 1.0×10^{15} 时，该算法的计算机执行时间就无法接受。我们可以计算如下：计算机每秒可执行一亿（1.0×10^9）条基本语句，则执行一个需 1.0×10^{15} 次基本操作的算法需要的时间为：

$$\begin{aligned} T &= 1.0 \times 10^{15} / 1.0 \times 10^9 = 1.0 \times 10^6 (\text{s}) \\ &= 1.0 \times 10^6 / 3\,600 = 277.8\ (\text{h}) \\ &= 277.8/24 = 11.6\ (\text{d}) \end{aligned}$$