全国高等专科教育计算机类规划教材

数据结构
——习题、实验

朱建芳　主编

机械工业出版社
CHINA MACHINE PRESS

全国高等专科教育计算机类规划教材

数据结构——习题、实验

主　编　朱建芳
副主编　赵兰波
参　编　黄才进　傅宜宁　罗　彬

机械工业出版社

本书是与李新燕和靳敏编的《数据结构》一书配套的辅助教材，亦可单独使用。本书的编写目的是使学生通过实验、课程设计和大量的习题解答加深对数据结构基本知识的理解，掌握求解数据结构问题的思路和方法，提高应用数据结构解决实际问题的能力。

本书分为2大部分：第1部分是“典型例题解析与习题”，共分10章：绪论、线性表、栈和队列、其他线性数据结构、递归、树、图、查找、排序和文件，每章先给出一系列与本章知识相关的典型例题的分析与解答，然后给出大量的练习题，题型包括单选题、填空题和综合题或判断题，同时在每个习题后给出参考答案。第2部分是“实验与课程设计指导”，包括9个实验：顺序表的操作、单链表的操作、栈与队列的操作、多维数组与串、二叉树的操作、图的操作、散列表操作、排序操作、文件排序，以及课程设计参考。

本书可供高职高专院校计算机专业学生作为学习《数据结构》课程的辅助教材，也可供自学考试和计算机等级（三级或四级）考试的读者作为参考用书。

图书在版编目（CIP）数据

数据结构——习题、实验/朱建芳主编．—北京：机械工业出版社，2006.8

全国高等专科教育计算机类规划教材

ISBN 7-111-19666-X

Ⅰ.数… Ⅱ.朱… Ⅲ.数据结构—高等学校：技术学校—教材

Ⅳ.TP311.12

中国版本图书馆CIP数据核字（2006）第085155号

机械工业出版社（北京市百万庄大街22号 邮政编码100037）

策划编辑：王玉鑫、孔嘉峻 责任编辑：王玉鑫 周 莉

版式设计：张世琴 责任校对：李秋荣

封面设计：姚 毅 责任印制：杨 曦

北京机工印刷厂印刷

2006年8月第1版·第1次印刷

184mm×260mm·12.75印张·312字

0 001—4 000册

定价：19.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

本社购书热线电话（010）68326294

编辑热线电话（010）68354423

封面无防伪标均为盗版

前　言

数据结构是计算机及相关专业的一门重要专业基础课，研究对象是数据的各种逻辑结构、存储结构和相应的操作。由于数据结构的原理和算法比较抽象，所以数据结构是一门学生普遍反映难学的课程。本书提供了大量的习题解答，以帮助学生加深对数据结构基本概念和原理的理解，掌握求解数据结构问题的思路和方法；另外，本书还配合每一章内容设计了相应的实验，使学生通过上机实验，进一步加深理解和巩固所学知识，提高分析和解决问题的能力。

本书分为两大部分：第1部分是“典型例题解析与习题”，共分10章：绪论、线性表、栈和队列、其他线性数据结构、递归、树、图、查找、排序和文件，每章先给出一系列与本章知识相关的典型例题的分析与解答，然后给出大量的练习题，题型包括单选题、填空题和综合题或判断题，同时在每个习题后给出参考答案。第2部分是“实验与课程设计指导”，包括9个实验：顺序表的操作、单链表的操作、栈与队列的操作、多维数组与串、二叉树的操作、图的操作、散列表操作、排序操作、文件排序，以及课程设计参考。

本书由朱建芳担任主编，赵兰波任副主编。其中第1、2、3、6、7、8章和实验1、2、3、4、5、6、7和附录由朱建芳编写，第5、9、10章和实验8、9由赵兰波编写，第4章由黄才进编写，尊一宇和罗彬参加了部分内容的编写工作。全书由朱建芳负责统稿和定稿。

由于水平有限，加之时间仓促，错漏之处难免，敬请读者批评指正。

编　者

目　录

第 2 部分 实验与课程设计指导

第 1 部分　典型例题解析与习题

第 1 章　绪　　论

1.1　典型例题解析

例 1-1　什么是数据结构？数据结构的研究涉及哪三个方面？

解答：

(1) 数据结构是指数据及数据相互之间的关系。

(2) 数据结构的研究一般涉及以下三方面的内容：

1) 数据成员及它们相互之间的逻辑关系，也称为数据的逻辑结构，简称为数据结构。

2) 数据成员及其关系在计算机内的存储表示，也称为数据的物理结构，简称为存储结构。

3) 施加于该数据结构上的操作。

数据的逻辑结构是从逻辑关系上描述数据，是与计算机存储无关的。因此，数据的逻辑结构可以看作是从具体问题中抽象出来的数据模型，是数据的应用视图。数据的存储结构是数据的逻辑结构在计算机存储器中的实现（亦称为映像），它是依赖于计算机的，是数据的物理视图。数据的操作是定义于数据逻辑结构上的一组运算，每种数据结构都有一个运算的集合，如搜索、插入、删除、更新、排序等。

例 1-2　什么是算法？算法的 5 个特性是什么？如何评价一个算法？

解答：

(1) 算法是指解决某一特定问题的一种方法或一个指令序列。

(2) 一个算法应当具有以下特性：

1) 输入。一个算法必须有 0 个或多个输入。它们是算法开始运算前给予算法的量。这些输入取自于特定的对象的集合。它们可以使用输入语句由外部提供，也可以使用赋值语句在算法内给定。

2) 输出。一个算法应有一个或多个输出，输出的量是算法计算的结果。

3) 确定性。算法的每一步都必须确定，不能有二义性。对于每一种情况，需要执行的动作都应严格地、清晰地规定。

4) 有限性。一个算法必须由有限步组成，即算法必须可以终止，不能进入死循环。

5) 正确性。算法必须解决具体问题，完成所期望的功能，给出正确的输出。

(3) 评价一个算法一般从 4 个方面进行：正确性、运行时间、占用空间和简单性。其中最主要的是算法的运行时间和占用空间。

正确性是指算法是否正确；运行时间是指一个算法在计算机上运行所花费的时间，采用

时间复杂度来量度，所谓时间复杂度是指计算出相应的数量级，如 O（1）、O（$\log_2 n$）、O（n）或 O（n^2）等；占用空间指在计算机存储器上所占用的存储空间，主要考虑在算法运行过程中临时占用的存储空间的大小，称之为空间复杂度，一般以数量级形式给出；简单性是指算法的易读性等。

例 1-3 分析以下程序段的时间复杂度。

```
for (i = 1; i < n; i + + )
{
    a + + ;                                    ①
    for (j = 0; j < = (2 * n); j + + )
        b + + ;                                ②
}
```

解答： 分析一个算法的时间复杂度，一般重点考察算法中与问题规模 n 有关的循环体重复操作次数。这个算法总体上是一个双重循环结构，外循环重复 n - 1 次，内循环重复（2n + 1）次，所以语句①重复执行次数为 n - 1，语句②重复执行次数为（n - 1）*（2n + 1）= $2n^2 - n - 1$。则整个程序段的时间复杂度为 T（n）= $n - 1 + 2n^2 - n - 1 = O(n^2)$。

例 1-4 分析以下程序段的时间复杂度。

```
i = 1;                                         ①
while (i < = n)
    i = i * 2;                                 ②
```

解答： 语句①执行次数为 1；设语句②重复执行次数为 k，则从循环条件知 $2^k \leqslant n$，即 $k \leqslant \log_2 n$。则整个程序段的时间复杂度为 T（n）= $1 + \log_2 n = O(\log_2 n)$。

例 1-5 分析以下程序段的时间复杂度。

```
p = 1; i = 1;                                  ①
while (i < = n)
{
    p * = i;                                   ②
    i + + ;                                    ③
}
```

解答： 语句①执行次数为 1；语句②重复执行次数为 n；语句③重复执行次数为 n。则整个程序段的时间复杂度为 T（n）= 1 + n + n = O（n）。

例 1-6 分析以下算法的时间复杂度。

```
prime (int n)
{
    int i = 2;
    while ( (n%i)! = 0 && i * 1.0 < sqrt (n)) i + + ;
    if (i * 1.0 > sqrt (n))
        printf ("%d is a prime number. \ n", n);
    else
        printf ("%d is not a prime number. \ n");
```

解答：重点考察 while 循环语句，循环体 i + + ; 语句的重复执行次数不超过 sqrt (n)，所以该算法的时间复杂度为 O ($\sqrt{n}$)。

1.2 练习题

1.2.1 单选题

1-1　数据结构是一门研究非数值计算的程序设计问题中 ① 及它们之间的 ② 和运算等的学科。

①A. 数据元素　　B. 计算方法　　C. 逻辑存储　　D. 数据映像

②A. 结构　　B. 关系　　C. 运算　　D. 算法

答案：①A　　②B

1-2　在数据结构中，从逻辑上可以把数据结构分成_____。

A. 动态结构和静态结构　　B. 紧凑结构和非紧凑结构

C. 线性结构和非线性结构　　D. 内部结构和外部结构

答案：C

1-3　算法分析的目的是 ① ，算法分析的两个主要方面是 ② 。

①A. 找出数据结构的合理性　　B. 研究算法中的输入和输出的关系

C. 分析算法的效率以求改进　　D. 分析算法的易读性和文档性

②A. 空间复杂度和时间复杂度　　B. 正确性和简单性

C. 可读性和文档性　　D. 数据复杂性和程序复杂性

答案：①C　　②A + B

1-4　下面程序段的时间复杂度为_____。

```
for (i = 0; i < m; i + + )
    for (j = 0; j < n; j + + )
        a [i] [j] = i * j;
```

A. O (m^2)　　B. O (n^2)　　C. O (m * n)　　D. O (m + n)

答案：C

1-5　执行下面程序段时，执行 S 语句的次数为_____。

```
for (i = 1; i < = n; i + + )
    for ( j = 1; j < = i; j + + )
        S ;
```

A. n^2　　B. $n^2/2$　　C. n (n + 1)　　D. n (n + 1) /2

答案：D

1-6　下面算法的时间复杂度为_____。

```
int f (int n)
{
    if (n = = 0 || n = = 1) return 1;
```

```
    else return n * f (n-1);
}
```

A. O (1)　　B. O (n)　　C. O (n^2)　　D. O (n!)

答案：B

1.2.2 填空题

1-1 数据的逻辑结构被分为__①__、__②__、__③__和__④__四种。

答案：①集合结构 ②线性结构 ③树形结构 ④图形结构（次序不分先后）

1-2 数据的存储结构被分为__①__、__②__、__③__和__④__四种。

答案：①顺序结构 ②链接结构 ③索引结构 ④散列结构（次序不分先后）

1-3 在线性结构、树形结构和图形结构中，前驱和后继结点之间分别存在着__①__、__②__和__③__的联系。

答案：①1:1　②1:N　③M:N 或①1 对 1　②1 对多　③多对多

1-4 在下面程序段中，s = s + p; 语句的执行次数为__①__，p * = j; 语句的执行次数为__②__，该程序段的时间复杂度为__③__。

```
int i = 0, s = 0, p = 1, j = 1;
while ( + + i < = n)
 {
    for ( ; j < = i; j + + ) p * = j;
    s = s + p;
 }
```

答案：①n　②n (n + 1) /2　③O (n^2)

1-5 一个算法的时间复杂度为 ($2n^2 + 2n\log_2 n + 4n - 7$) / (5n)，其数量级表示为_____。

答案：O (n)

第2章 线 性 表

2.1 典型例题解析

2.1.1 顺序表

顺序表的数据结构类型定义如下：

```
# define ELEMTYPE int
# define MAXSIZE 100
typedef struct
{
    ELEMTYPE datas [MAXSIZE];
    int last;
} SEQUENLIST;
```

例 2-1 如何对顺序表进行初始化？分别写出在顺序表 a 中从开头插入和结尾插入新元素的算法，并分析时间复杂度。

解答：顺序表的初始化只需将表长度设为 0；从尾部插入一个新元素到顺序表的算法非常简单，只需将新元素插入到下标为 a->last 的位置上，然后表长度 a->last 加 1，算法时间复杂度为 O（1）；从头部插入新元素到顺序表中的算法则比较麻烦，每次必须先把顺序表中已有的所有元素后移一个位置，然后再将新元素插入到第 1 个位置上，然后表长度 a->last 加 1，算法时间复杂度为 O（n）。实现上述功能的函数如下：

```
/*顺序表初始化函数*/
void initiate (SEQUENLIST *a)
{
    a->last=0;
}
/*将新元素 x 从尾部插入顺序表 a 中的函数*/
int insertrear (SEQUENLIST *a, ELEMTYPE x)
{
    if (a->last==MAXSIZE)                    /*表已满则返回 0*/
        return 0;
    a->datas [a->last] =x;
    a->last++;
    return 1;
}
/*将新元素 x 从头部插入顺序表 a 中的函数*/
```

```
int insertfront (SEQUENLIST * a, ELEMTYPE x)
{
    int i;
    if (a - > last = = MAXSIZE)                    /* 表已满则返回 0 */
        return 0;
    for (i = a - > last; i > 0; i - - )             /* 将表中所有元素后移一个位置 */
        a - > datas [i] = a - > datas [i - 1];
    a - > datas [0] = x;                           /* 将元素 x 放入第 1 个位置 */
    a - > last + + ;                               /* 表长加 1 */
    return 1;
}
```

例 2-2 写一个算法，将一个值为 x 的元素插入到顺序表 A 中第 i 个位置前。并分析该算法的时间复杂度。

解答：插入元素后必须保持原来顺序表中的元素值及相互之间的前后关系不被破坏，因此，必须先把第 i 个位置及其后的所有元素往后移动一个位置，空出第 i 个位置，然后再将 x 放入该位置中，并令表长度加 1。

```
/* 将新元素 x 插入到顺序表 a 中第 i 个位置函数 */
int insert (SEQUENLIST * a, ELEMTYPE x, int i)
{
    int k;
    if (i < 1 || i > a - > last + 1 || a - > last > = MAXSIZE) /* i 超出范围则返回 0 */
        return 0;
    else
      {
        for (k = a - > last; k > = i; k - - )
            a - > datas [k] = a - > datas [k - 1];          ①
        a - > datas [i - 1] = x;
        a - > last = a - > last + 1;
        return 1;
      }
}
```

时间复杂度分析：该算法的基本操作主要是元素后移操作，所以重点考察 for 循环语句。设数据规模 n = a - > last，语句①的执行次数（即移动元素的次数）为 n - i + 1，所以移动次数不仅与顺序表长度有关，还与插入位置 i 有关。最好的情况是当 i = n + 1 时，移动次数为 0；最坏的情况是当 i = 1 时，移动次数为 n；平均的情况是，设在 1 ~ n + 1 之间任何位置插入元素的机会均等，即在任意位置 i 插入元素的概率 P = 1/ (n + 1)，则算法的平均移动次数为：

$$T(n) = \sum_{i=1}^{n+1}(n - i + 1)/n + 1 = \frac{n}{2}$$

即在顺序表上的插入操作，平均需要移动一半的元素，时间复杂度为 O (n)。

例 2-3 已知一个顺序表非递减有序，写一个算法插入一个值为 x 的元素并保持顺序表非递减有序。

解答： 先查找到适当的插入位置 i，然后将第 i 个位置及之后的所有元素后移一个位置，再将 x 插入到位置 i 中。实现算法的函数如下：

```
/*将新元素 x 插入到非递减有序的顺序表 a 中适当的位置的算法*/
void insert1 (SEQUENLIST *a, ELEMTYPE x)
{   int i;
    for (i=0; i<a->last; i++)
        {
            if (a->datas [i] >x) /*若第 i 个位置的元素值比 x 大则找到了插入位置*/
                break;
        }
    for (k=a->last; k>=i; k--) /*将第 i 个位置及之后的所有元素后移一个位置*/
        a->datas [k] =a->datas [k-1];
    a->datas [i-1] =x;              /*将 x 插入到 位置 i 中（下标为 i-1）*/
    a->last=a->last+1;              /*顺序表长度增 1*/
}
```

例 2-4 试用顺序表作为存储结构，写一个算法将线性表的元素逆置。例如顺序表 A = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)，逆置后为 A' = (10, 9, 8, 7, 6, 5, 4, 3, 2, 1)。要求算法空间复杂度为 O (1)。

解答： 因为要求算法的空间复杂度为 O (1)，所以只好在原顺序表存储空间中进行置换元素的操作，只需将前半部分的元素从前往后逐个与对应的后半部分的元素从后往前进行置换操作，即可实现整个线性表的逆置。

```
/*将顺序表 a 中的元素逆置——方法 1*/
void inverse1 (SEQUENLIST *a)
{
    ELEMTYPE t;
    int i, k;
    k=a->last;
    for (i=0; i<k/2; i++)
        {
            t=a->datas [k-1-i];
            a->datas [k-1-i] =a->datas [i];
            a->datas [i] =t;
        }
}
/*将顺序表 a 中的元素逆置——方法 2*/
void inverse2 (SEQUENLIST *a)
```

```
{
    int i, j; /* 用两个指针 i 和 j 分别指向数组的头和尾，同时向中间位置移动 */
    ELEMTYPE t;
    j = a -> last;
    i = 0;
    while (j > i)
        {
            t = a -> datas [j];
            a -> datas [j] = a -> datas [i];
            a -> datas [i] = t;
            i + +;
            j - -;
        }
}
```

例 2-5 按下面两种情况分别编写算法删除顺序表中多余的值相同的元素。

(1) 顺序表元素值递增有序。

(2) 顺序表元素值无序。

解答：

(1) 由于顺序表中的元素值递增有序排列，值相同的元素必为相邻的元素，因此，从第 1 个元素开始，将后面的元素逐个与之相比较，若相等，则记录重复元素个数的整型变量 n 增 1，直到后面的元素与之不相同即停止这一轮比较，若 n > 0，则表示有 n 个多余的值相同的元素，需将后面不相同的所有元素都往前移动 n 个位置以覆盖（删除）这些多余元素，同时要将顺序表长度减 n，并令 n = 0 即完成这一轮的比较、删除过程，然后转去进行下一个元素的新一轮比较、删除过程，直至所有多余的值相同的元素都被删除。

```
/* 删除递增有序顺序表中值相同多余元素的算法——方法 1 */
void delsame1_1 (SEQUENLIST * a)
{   int i, j, k, n = 0;                          /* n 记录重复元素个数 */
    for (i = 0; i < a -> last; i + +)
    {
    for (j = i + 1; j < a -> last; j + +)
    {   if (a -> datas [i] ! = a -> datas [j])    /* 元素值不等则结束这一趟比较 */
            break;
        n + +;                                    /* 重复元素个数增 1 */
    }
    if (n ! = 0)
    {   for (k = i + 1 + n; k < a -> last; k + +)  /* 将后面不同的所有元素向前移动 n
                                                     个位置 */
            a -> datas [k - n] = a -> datas [k];
        a -> last = a -> last - n;                /* 顺序表长度减 n */
```

```
            n=0;            /*记录重复元素个数的n复位为0*/
        }
    }
}
/*删除递增有序顺序表中值相同多余元素的算法——方法2，直接用j-i-1表示重复元素个数*/
void delsame1-2 (SEQUENLIST *a)
{   int i, j, k;
    for (i=0; i<a->last; i++)
    {   for (j=i+1; j<a->last; j++)
            if (a->datas [i]!=a->datas [j])
                break;
        if (j-i>1)
        {   for (k=i+1+(j-i-1); k<a->last; k++)
                a->datas [k-(j-i-1)]=a->datas [k];
            a->last=a->last-(j-i-1);
        }
    }
}
```

(2) 由于顺序表是无序的，值相同的元素不一定相邻，所以从第1个元素开始，将其后面的所有元素逐个与之相比较，若与之相等，则删除该元素，顺序表长度减1，然后转去下一个元素的循环比较、删除过程。

```
/*删除无序顺序表中值相同多余元素的算法——方法1*/
void delsame2_1 (SEQUENLIST *a)
{   int i, j, k;
    for (i=0; i<a->last; i++)
        for (j=i+1; j<a->last; j++)
            if (a->datas [i]==a->datas [j])      /*元素值相等则删除该元素*/
            {   for (k=j+1; k<a->last; k++)
                    a->datas [k-1]=a->datas [k];
                a->last--;                  /*顺序表长度减1*/
                j--; /*因为删除了一个元素，所以j不用后移，此处j--是为了抵消j++*/
            }
}
/*删除无序顺序表中值相同多余元素的算法——方法2，把内循环变量增1的语句j++放在循环体的判断语句的else分句中，即元素值不同才后移*/
void delsame2_2 (SEQUENLIST *a)
{   int i, j, k;
```

```
    for (i=0; i<a->last; i++)
        for (j=i+1; j<a->last; )
        if (a->datas[i] == a->datas[j])
        {   for (k=j+1; k<a->last; k++)
                a->datas[k-1] = a->datas[k];
            a->last--;
        }
        else j++;
}
```

可用下面的主函数测试例 2-5 算法的正确性。

```
/*检查例 2-5 算法正确性的主函数*/
void main ()
{
    int i, n;
    ELEMTYPE x;
    SEQUENLIST b;
    /*创建一个顺序表*/
    initiate (&b);
    printf ("Please input 10 number to test (1): \n");
    for (i=0; i<10; i++)
    {
        scanf ("%d", &x);
        insert (&b, x, b.last+1);        /*从尾部插入*/
    }
    delsame1_1 (&b);
    printf ("the list after delete: \n");
    traverse (&b);
    /*创建一个顺序表*/
    initiate (&b);
    printf ("Please input 10 number to test (2): \n");
    for (i=0; i<10; i++)
    {
        scanf ("%d", &x);
        insert (&b, x, b.last+1);        /*从尾部插入*/
    }
    delsame2_1 (&b);
    printf ("the list after delete: \n");
    traverse (&b);
    getch ();
```

```
}
```

2.1.2 单链表

单链表的结点结构类型定义如下：

```
# define ELEMTYPE char
typedef struct node
{
    ELEMTYPE data;
    struct node * next;
} LINKLIST;
```

例 2-6 输入若干个字符，以‘#’为结束标志，建立一个带头结点的单链表，要求单链表顺序与输入顺序保持一致。

解答： 首先申请一个结点的存储空间，生成表头结点，并初始化单链表；然后循环读入字符、申请结点存储空间、生成新结点、将新结点从尾部插入单链表中，直至读入字符‘#’，结束算法。实现该算法的函数如下：

```
void main ()
{
    LINKLIST * head, * last, * t;
    char ch;
    head = (LINKLIST * ) malloc (sizeof (LINKLIST));        /* 建立头结点 */
    last = head;                                            /* last 指针指向表尾结点 */
    head - > next = NULL;                                   /* 初始化单链表 */
    while ( (ch = getchar ())! = '#')
    {
        t = ( LINKLIST * ) malloc (sizeof (LINKLIST));      /* 生成新结点 */
        t - > data = ch;
        last - > next = t;                                  /* 插入至单链表表尾 */
        last = t;
    }
    last - > next = NULL;
}
```

例 2-7 写一个算法，求一个单链表（不带头结点，表头指针为 head）的长度，并分析算法时间复杂度。

解答： 从表头开始遍历单链表，每访问一个结点，令累计结点个数的变量 n 加 1。实现算法的函数如下：

```
/* 求单链表长度的函数 */
int length (LINKLIST * head)
{
    int n = 0;
```