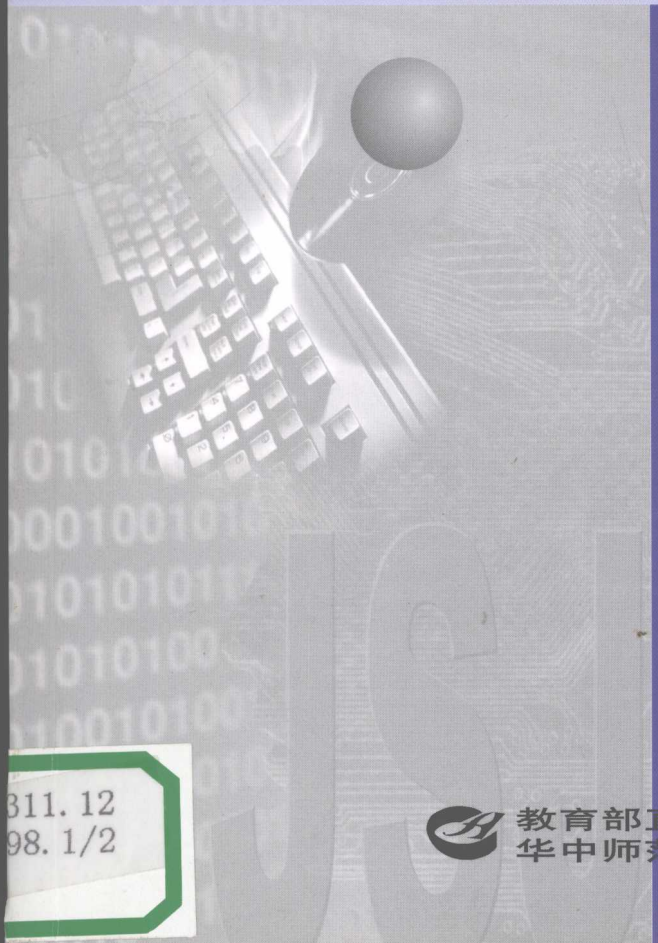
计算机科学与技术系列丛书

数据结构（下）——实训教程

SHUJU JIEGOU (XIA) — SHIXUN JIAOCHENG

李晓燕 主编



教育部直属师范大学
华中师范大学出版社

计算机科学与技术系列丛书

数据结构(下)

——实训教程

主　编：李晓燕
副主编：余　亮　史永莉
编　委：(按姓氏笔画排序)
万　臣　史永莉
汪怀杰　余　亮
李晓燕　姜艳艳
曾秀莲　韩丽娟

华中师范大学出版社

内容简介

本书是《数据结构》(上)的姊妹篇,为读者提供了《数据结构》(上)相应章节丰富的实例和练习,其中练习附有详尽的解答。这些实例和练习可以帮助读者提高对各种数据结构以及查找、分类算法的理解和应用。

本书可作为普通高等学校、二级学院本科和大专以及独立学院大专计算机专业的教材。

新出图证(鄂)字 10 号

图书在版编目(CIP)数据

数据结构(下)——实训教程/李晓燕主编. —武汉:华中师范大学出版社,2005.8
(计算机科学与技术系列丛书)
ISBN 7-5622-3228-8/TP·44
Ⅰ.数… Ⅱ.李… Ⅲ.数据结构—高等学校:技术学校—教学参考资料
Ⅳ.TP311.12
中国版本图书馆 CIP 数据核字(2005)第 058055 号

书　　名:数据结构(下)——实训教程
主　　编:李晓燕
选题策划:华中师范大学出版社第二编辑室　电话:027—67867362
出版发行:华中师范大学出版社©
地　　址:武汉市武昌珞瑜路 100 号　邮编:430079
销售电话:027—67867076　67863040　67867371　67861549
邮购电话:027—67861321　传真:027—67863291
网址:http://www.ccnup.com.cn　电子信箱:hscbs@public.wh.hb.cn
经　　销:新华书店湖北发行所　　督　　印:姜勇华
印　　刷:湖北地矿印业有限公司
责任编辑:杨　鹏　　责任校对:罗　艺　　封面设计:罗明波
开本/规格:787mm×960mm　1/16　　印　　张:10.5　　字　　数:200 千字
版次/印次:2005 年 8 月第 1 版,2005 年 8 月第 1 次印刷
印　　数:1-3500
定　　价:16.80 元

前　言

《数据结构》(下)——实训教程，是《数据结构》(上)的姊妹篇。《数据结构》上册，主要介绍了线性表、栈、队列、树、二叉树和图等各种基本数据结构，以及查找、分类算法。而《数据结构》下册，则为读者提供了丰富的实训实例和练习。这些实例和练习可以帮助读者提高对各种数据结构以及查找、分类算法的理解和应用。

《数据结构》下册分为10章：第1章《数据结构》描述语言实训；第2章线性表数据结构实训；第3章栈与队列数据结构实训；第4章数组数据结构实训；第5章串数据结构实训；第6章树与二叉树数据结构实训；第7章图数据结构实训；第8章查找实训；第9章分类实训；第10章文件数据结构实训。

其中，第1章由姜艳艳编写，第2章由史永莉编写，第3章由万臣编写，第4章和第5章由余亮编写，第6章和第7章由韩丽娟编写，第8章和第10章由曾秀莲编写，第9章由汪怀杰编写。全书由主编李晓燕教授统稿审定。

本书在编写过程中得到了作者所在高校和华中师大出版社领导和同志的支持和帮助，在此深表谢意。

书中可能出现的错误和不妥之处，请各位批评指正。

作者

于2005年5月

目　录

第1章 《数据结构》描述语言实训

1.1 算法描述工具——C语言

如何选择描述数据结构和算法的语言是十分重要的问题。近年来，在计算机教学和科学研究、以及系统程序和应用程序开发中，C语言的使用越来越广泛。因此，本教材采用C语言进行算法描述。

(1) 问题的规模尺寸可在宏定义中预定义，例如：

```
#define MAXSIZE 100
```

(2) 数据元素的类型可以写成ELEMTP，可以认为在宏定义中定义过，例如：

```
#define ELEMTP int
```

根据需要，可临时用具体的类型标识符来代替之。

(3) 一个算法往往以函数形式给出：

```
类型标识符          函数名(带类型说明的形参表)
{语句组}
```

例如：

```
int sum (int a, int b)
    {
      c=a+b;
      return (c);
    }
```

(4) 结构类型定义以及全局变量的说明等建议放在算法之前。

例 将n个整数按由大到小的顺序排列，并且输出排列结果。

分析 n个数据的逻辑结构是线性表($a_1, a_2, a_3, \cdots, a_n$)；选用一维数组作存储结构。每个数组元素是一个结构，它有两个域：一个是数据的序号域，一个是数据的值域：

```
struct ar
{
  int num; /*序号域*/
  int data; /*数据域*/
};
```

算法描述如下：

```
void simsort (struct ar a[MAXSIZE], int n)    /*n, 数组 a 的数据由主调
                                               函数提供*/
{
  int i, j,m;
  for(i=1; i<n; i++)
    for (j=i;j<n;j++)
      if(a[i]. data<a[j]. data)
       {  m=a[i];a[i]=a[j];a[j]=m;}
  for(i=0;i<n;i++)
    printf("%8d %8d %10d\n", i,a[i]. num,a[i]. data);
}
```

1.2 算法分析

算法分析是指在算法正确的情况下，对其优劣所进行的分析。一个好的算法通常是指：

(1) 算法对应的程序所耗时间少；

(2) 算法对应的程序所耗存储空间少；

(3) 算法结构性好、易读、易移植和易调试等。

但实际上“时间”和“空间”是矛盾的。要想算法执行得快，就得忍让一些存储空间；反之，要想节省存储空间，算法就会复杂一些，所耗时间也会长些。

本书中算法一般较短，对其分析主要是考虑算法的时间效率。算法对应程序的时耗，由下面诸多因素决定：

(1) 程度中输入数据的时间；

(2) 对源程序编译所需的时间；

(3) 执行每条语句的时间；

(4) 程序中可执行语句的执行次数。

对(1)，若采取脱机输入技术，则输入数据时间可与程序运行时间相覆盖；对(2)、(3)，依赖编译系统和计算机本身的速度；所以在编程中，主要考虑第 4 点。为此，引入下面几个概念：

1. 语句的频度(Frequency Count)

语句的频度指在算法(或程序)中可执行语句重复执行的次数。若某语句执行一次的时间为 t，执行次数为 f，则该语句所耗的时间估计为 t·f。

【实例 1】 求两个 n 阶方阵的乘积

$$C=A*B=\begin{bmatrix} C[0][0] & C[0][1] & \cdots\cdots & C[0][j] & \cdots\cdots & C[0][n-1] \\ C[1][0] & C[1][1] & \cdots\cdots & C[1][j] & \cdots\cdots & C[1][n-1] \\ \cdots\cdots & & & & & \\ \cdots\cdots & & & & & \\ C[i][0] & C[i][1] & \cdots\cdots & C[i][j] & \cdots\cdots & C[i][n-1] \\ \cdots\cdots & & & & & \\ C[n-1][0] & C[n-1][1] & \cdots\cdots & C[n-1][j] & \cdots\cdots & C[n-1][n-1] \end{bmatrix}$$

其中：

$$C[i][j]=\sum_{k=0}^{n-1} A[i][k]\times B[k][j]$$

算法描述：

```
#define    n    10
MATRIXM(A,B,C)
float A[n][n],B[n][n],C[n][n];
{
   int i, j, k;
   for(i=0;i<n;i++)
     for(j=0;j<n;j++)
     {
       C[i][j]=0;
       for(k=0;k<n;k++)
         C[i][j]=C[i][j]+A[i][k]*B[k][j];
     }
}
```

for 循环语句的执行次数实际上是循环变量的变化次数，所以上例中第一个 for 语句的执行次数（即频度）应为 n+1，其他语句的频度依次为 n(n+1)、n^2、n^2(n+1)和 n^3。

2. 算法的时间复杂度(Time Complexity)

算法的时间复杂度定义为算法中可执行语句的频度之和，记为 T(n)。T(n)是算法所需时间的一种估计，其中 n 为问题的规模（或大小、体积），它有时为算法的输入量，有时为算法的计算量。如上例中，问题的规模 n 为矩阵的阶，该算法的时间复杂度为：

$$\begin{aligned} T(n)&=(n+1)+n(n+1)+n^2+n^2(n+1)+n^3 \\ &=2n^3+3n^2+2n+1 \end{aligned}$$

当 $n\to\infty$ 时，$\lim_{n\to\infty}(T(n)/n^3)=2$，故 T(n)与 n^3 为同阶无穷大，或说 T(n)与 n^3 成

正比、T(n)的量级为 n^3，记为：

$$T(n)=O(n^3)$$

对不同的场合，有时以 n 取不同值时算法平均所耗时间作为 T(n)(如在查找场合)，有时又以算法最长所需时间作为 T(n)(如在排序场合)。当然，算法设计时，应使 T(n)的量级越小越好，以提高算法的执行速度。另外，在算法分析中关键要抓住一些循环语句的执行次数，而对一些循环之外语句的执行次数有时可忽略不计。

【实例 2】 在数组(A[0],A[1],A[2],…,A[n−1])中查找第一个与给定值 k 相等的元素的序号。

算法描述如下：

```
int search (datatype A[n],k)   /*在 A[0], A[1], A[2], …, A[n−1]中
                                 查找 k 的算法*/
{
   int i;
   i=0;
   while(i<n&&A[i]!=k)
     i++;
   if(i<n)
     return i;
   else
     return (−1);
}
```

本例应以平均查找次数(即查找时 k 与 A 中元素比较次数的平均值)作为算法的 T(n)。设查找每个元素的概率 $p_i(0\leqslant i\leqslant n-1)$均等，即 $p_i=1/n$；查找元素 k 时，k 与 A[i]的比较次数(即执行 while 循环语句的次数)$c_i=i+1$，则查找次数的平均值(或期望值)：

$$T(n)=\sum_{i=0}^{n-1}p_i(i+1)=\frac{1}{n}\sum_{i=0}^{n-1}(i+1)=\frac{1}{2}(n+1)$$

因为$\lim\limits_{n\to\infty}(T(n)/n)=\frac{1}{2}$，所以 T(n)=O(n)。此时，又称 T(n)为算法的渐进时间复杂度。当然，本例中若 k 不在数组 A 中，则 while 语句最多执行 n+1 次，量级同样为 O(n)。

【实例 3】 设某算法执行部分语句如下：

```
x=1;
for(i=1;i<=n;i++)
```

```
for(j=1;j<=i;j++)
  for(k=1;k<=j;k++)
    x++;
```

此例中，循环变量 i 从 1 变化到 n；对每个 i，循环变量 j 从 1 变化到 i；而对每个 j，循环变量 k 从 1 变化到 j。显然，最内循环体内的语句 x++执行频度是最高的，故以 x++的执行次数来刻画 T(n)，即：

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

$$=\frac{1}{6}n(n+1)(n+2)$$

因为$\lim_{n\to\infty}(T(n)/n^3)=\frac{1}{6}$，所以 $T(n)=O(n^3)$。

有时为加快算法的分析速度，而不纠缠一些细节，可对算法的 T(n)作大致的估计。

【实例 4】 设某算法的执行部分如下：

```
x=n;
y=1;
while(x>=(y+1)*(y+1))
  y++;
```

此例关键看 while 语句的执行次数(或 y++的执行次数)，y 从 1 开始，每次加 1，当其接近$\sqrt{n}$时，结束循环，故 T(n)估计为 $O(\sqrt{n})$。

T(n)的量级通常有：

O(c)——常数级，算法中不论问题规模多大，T(n)一致，是最理想的 T(n)量级；

O(n)——线性级；

$O(n^2)$，$O(n^3)$——平方、立方级；

$O(\log_2 n)$，$O(n*\log_2 n)$——对数、线性对数级；

$O(2^n)$——指数级，时间复杂度最差。

对较为复杂的算法，可分段分析其时间复杂度。如某算法可分为两部分，其时间复杂度分别为：

$$T_1(n)=O(f(n)),\ T_2(n)=O(g(n))$$

此时两部分问题的体积一致，则总的 $T(n)=T_1(n)+T_2(n)=O(\max(f(n),g(n)))$，$\max(f(n),g(n))$表示取 f(n)、g(n)中最大者。但若 $T_1(m)=O(f(m))$，$T_2(n)=O(g(n))$，两部分的体积不一致，则总的 $T(m,n)=T_1(m)+T_2(n)=O(f(m)+g(n))$。

另外,算法的空间复杂度也有类似的定义:设算法对应问题的体积(或规模)为 n,执行算法所占存储空间的量级为 D(n),则 D(n)为算法的空间复杂度(Space Complexity)。

1.3 算法设计实例

【实例 1】 试写一算法,从大至小依次输出顺序读入的三个整数 X,Y 和 Z 的值。

【解】

```
void print_descending(int x, int y, int z) /*按从大到小顺序输出三个数*/
{
   scanf("%d,%d,%d",&x,&y,&z);
   if(x<y) swap(x,y); /*swap 为表示交换的函数,以下同*/
   if(y<z) swap(y,z);
   if(x<y) swap(x,y); /*冒泡排序*/
   printf("%d,%d,%d",x,y,z);
}/*print_descending*/
```

【实例 2】 已知 k 阶裴波那契序列的定义为

$$f_0=0,\ f_1=0,\cdots,f_{k-2}=0;\ f_{k-1}=1;$$

$$f_n=f_{n-1}+f_{n-2}+\cdots+f_{n-k}(n=k,\ k+1,\cdots)$$

试编写求 k 阶裴波那契序列的第 m 项值的算法,k 和 m 均以参值的形式在参量表中出现。

【解】

```
status fib(int k, int m, int &f)
/*求 k 阶裴波那契序列的第 m 项的值 f*/
{
   int tempd,i,sum,j;
   if(k<2||m<0)
     return ERROR;
   if(m<k-1)
     f=0;
   else if(m=k-1||m==k)
    f=1;
   else
   {
     for(i=0;i<k-2;i++) tempd[i]=0;
```

```
            tempd[k-1]=1;
            tempd[k]=1; /*初始化*/
            sum=1;
            j=0;
            for(i=k+1;i<=m;i++;j++){
            /*求出序列第 k 至第 m 个元素的值*/
              tempd[i]=2*sum-tempd[j];
              sum=tempd[i];}
            f=tempd[m];
          }
          return OK;
        }/*fib*/
```

【分析】 k 阶裴波那契序列的第 m 项的值 f[m]=f[m-1]+f[m-2]+…+f[m-k]=f[m-1]+f[m-2]+…+f[m-k]+f[m-k-1]-f[m-k-1]=2*f[m-1]-f[m-k-1]

所以上述算法的时间复杂度仅为 O(m)。

【实例 3】 假设有 A、B、C、D、E5 个高等院校进行田径对抗赛，各院校的单项成绩均已存入计算机并构成一张表，表中每一行的形式为

项目名称	性别	校名	成绩	得分

编写算法，处理上述表格，以统计各院校的男、女总分和团体总分，并输出结果。

【解】

```
typedef struct
{
   char *sport;
   enum {male, female}gender;
   char schoolname;          /*校名为 'A'、'B'、'C'、'D' 或 'E'*/
   char *result;
   int score;
}resulttype;
typedef struct
{
  int malescore;
  int femalescore;
  int totalscore;
```

```
} scoretype;
void summary (resulttype result[])
/*求各校的男女总分和团体总分,假设结果已经储存在 result[]数组中*/
{
  scoretype score [MAXSIZE];
  int i=0;
  while (result[i]. sport)
  {
    switch(result[i]. schoolname)
    {
      case 'A'
        score[0]. totalscore+=result[i]. score;
        if(result[i]. gender==0)
          score[0]. malescore+=result[i]. score;
        else score[0]. femalescore+=result[i]. score;
        break;
      case 'B'
        score[1]. totalscore+=result[i]. score;
        if(result[i]. gender==0)
          score[1]. malescore+=result[i]. score;
        else score[1]. femalescore+=result[i]. score;
        break;
      case 'C'
        score[2]. totalscore+=result[i]. score;
        if (result[i]. gender==0)
          score[2]. malescore+=result[i]. score;
        else score[2]. femalescore+=result[i]. score;
        break;
      case 'D'
        score[3]. totalscore+=result[i]. score;
        if (result[i]. gender==0)
          score[3]. malescore+=result[i]. score;
        else score[3]. femalescore+=result[i]. score;
        break;
      case 'E'
```

```
            score[4].totalscore+=result[i].score;
            if(result[i].gender==0)
              score[4].malescore+=result[i].score;
            else score[4].femalescore+=result[i].score;
            break;
          }
          i++;
      }
      for(i=0;i<5;i++)
      {
        printf ("school:%c\n", result[i].schoolname);
        printf ("totalscore of male:%d\n",score[i].malescore);
        printf ("totalscore of female:%d\n",score[i].femalescore);
        printf ("totalscore of all:%d\n", score[i].totalscore);
      }
    }/*summary*/
```

【实例 4】 试编写算法，计算 $i! \cdot 2^i$ 的值并存入数组 a[1..arrsize]的第 i 个分量中(i=1,2,…,n)。假设计算机中允许的整数最大值为 maxint，则当 n>arrsize 或对某个 k(1≤k≤n)使 $k! \cdot 2^k$>maxint 时，应按出错处理。注意选择你认为较好的出错处理方法。

【解】

```
status algo119(int a[ARRSIZE])
/*求 i!*2^i 序列的值且不超过 maxint*/
{
  int last, i;
  last=1;
  for (i=1; i<=ARRSIZE;i++)
  {
    a[i-1]=last*2*i;
    if((a[i-1]/last)!=(2*i))
      return OVERFLOW;
    last=a[i-1];
  }
  return OK;
}/*algo 119*/
```

第2章　线性表数据结构实训

2.1　上机实例

【实例1】　求单链表的长度。

【解】

```
#include<stdio.h>
#include <stdlib.h>
#define ElemType int
typedef struct LNode
{
    ElemType data;
    struct LNode *next;
} LNode, *LinkList;

int Length (LinkList L)
/* L为单链表的头指针,此函数返回L所指单链表的长度 */
{
    int k;
    LNode *p;
    p=L->next;  /* 初始时,p指向单链表的第一个结点 */
    k=0;  /* 初始时,计数器k为0 */
    while (p!=NULL)
    {
        k++;  /* 计数器k增1 */
        p=p->next;  /* 指针p顺链向后移动 */
    }
    return (k);
} /* Length */
int main()
{
```

```
    LinkList L;
    LNode *p;
    L=(LinkList)malloc(sizeof(LNode));
    L->next=NULL;
    printf("请输入单链表元素,0 结束(以回车或者空格作为间隔;只能输入
                                        整数,其他字符视为结束):\n");
    while(1)
    {
        p=(LinkList)malloc(sizeof(LNode));
        p->next=NULL;
        scanf("%d",&p->data);
        if (p->data==0)
          break;
        p->next=L->next;
        L->next=p;
    }
    printf("单链表长度为:%d ",Length (L)) ;
    printf("OK!");
}
```

【实例 2】 创建一个单链表,并且输出所创建的单链表。

【解】

```
#include <stdio. h>
#include <stdlib. h>
typedef struct node
{
   int data;
   struct node *next;
}lnode;
void main ( )
{
   int i,n;
   lnode *p, *s, *q, *head;
   printf ("\n Please input the number of the node:");
   scanf ("%d",&n);
   if (n<0)
```