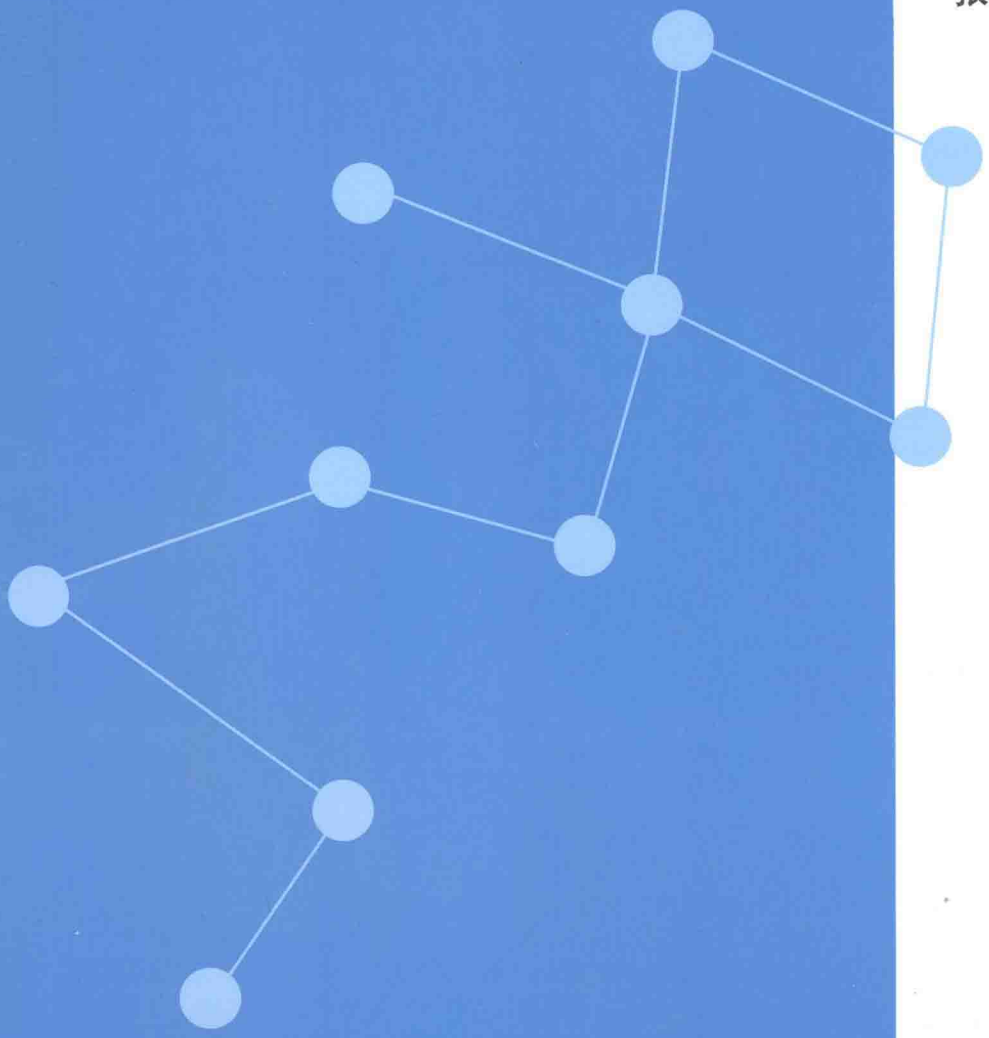
普通高等教育“十一五”国家级规划教材

数据结构习题解析与实训

（第2版）

张世和 徐继延 编著



清华大学出版社

普通高等教育“十一五”国家级规划教材

高职高专计算机教材精选

数据结构习题解析与实训（第2版）

张世和 徐继延 编著

清华大学出版社
北京

内 容 简 介

本书是普通高等教育“十一五”国家级规划教材《数据结构》(第2版)的配套教材,针对《数据结构》(第2版)的每一章习题,给出了习题的解析和参考答案,编程题的源程序汇集成光盘随书附上。本书还设计了一些没有题解的实训题供学生独立思考完成,能使学生加深对基础理论知识和概念的理解,得到实际应用的收获。

本书适合作为高职高专计算机及相关专业的教材,也适合计算机数据结构的爱好者自学。

图书在版编目(CIP)数据

数据结构习题解析与实训/张世和,徐继延编著.—2版.—北京:清华大学出版社,2008.8
(高职高专计算机教材精选)
ISBN 978-7-302-16904-8

Ⅰ.数… Ⅱ.①张… ②徐… 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(2008)第009081号

责任编辑:谢　琛　张为民
责任校对:梁　毅
责任印制:李红英

出版发行:清华大学出版社　　**地　　址**:北京清华大学学研大厦A座
http://www.tup.com.cn　　**邮　　编**:100084
社　总　机:010-62770175　　**邮　购**:010-62786544
投稿与读者服务:010-62776969,c-service@tup.tsinghua.edu.cn
质　量　反　馈:010-62772015,zhiliang@tup.tsinghua.edu.cn
印 装 者:山东新华印刷厂临沂厂
经　　销:全国新华书店
开　　本:185×260　　**印　张**:11.25　　**字　数**:258千字
附光盘1张
版　　次:2008年8月第2版　　**印　次**:2008年8月第1次印刷
印　　数:1～4000
定　　价:21.00元

前言

“数据结构”是计算机程序设计的重要理论基础，是计算机及其应用专业的一门重要基础课程和核心课程。

本书是普通高等教育“十一五”国家级规划教材《数据结构》的配套教材，内容上力求体现“以应用为主体”，强调理论知识的理解和运用，实现高职高专教学以实践体系为主及以技术应用能力培养为主的目标。

“数据结构”课程的基本原理和算法分析、设计有较强的抽象性和逻辑性，学生对课堂上讲授的基本知识还可以理解，但课后对独立编写算法和程序的习题，往往显得无从下手。而真正学好“数据结构”课程又必须掌握各类数据结构的应用练习，掌握算法及程序的设计思想、结构分析、编程方法和技巧。为了培养和提高学生对“数据结构”课程的兴趣，尽可能为后续学习打下一个良好的基础，我们编写了和《数据结构》(第 2 版)教材配套的习题解析与实训用书。

本书对应《数据结构》(第 2 版)中的每一章习题，给出习题的解析和参考答案，编程题的源程序汇集成光盘随书附上。希望在整个教学进程中，配合对习题题解的学习，提高学生对“数据结构”课程的兴趣，能使学生加深对基础理论知识和概念的理解，得到实际应用的收获。本习题集还设计了一些没有题解的实训练习题供学生独立思考完成。

习题集中所有的程序均用 C 语言编写。其中的算法和程序全部在 PC 机上用 Turbo C 调试通过。学生可以方便地在计算机上实现、验证这些程序。最后的附录汇总了教科书各章中介绍各类数据结构时用到的数据结构类型说明，和《数据结构》(第 2 版)的附录 A 相同。

本书程序中的输入、输出和注释均以中文描述和表达。程序可以在 Windows 98 操作系统、Visual C ++ 软件环境下编译运行(因为程序的源代码用的全是 Visual C ++ 中的 C 语言语句)。

由于习题较多，题解上可能存在不够完整和疏漏之处，本书在内容编排上也可能存在不够合理的地方，敬请读者批评指正。

编　者

2008 年 3 月

目录

第1章

绪　论

本书对应《数据结构》(第2版)教材上的章节，给出每一章的习题解答。习题中所有的算法和程序都用C语言编写并已上机调试通过，并在本书所配的光盘中提供了程序的源文件。考虑到函数调用的共享性，有的章节中还给出一些汇总性的编程习题及其源程序。算法和编程习题用到的数据结构类型说明定义都放在头文件datastru.h中。头文件datastru.h在光盘根目录下。第6章"树和二叉树"中建立链表结构二叉树的函数因为经常被调用所以作为函数文件"二叉树.c"放在光盘根目录下。

程序中的输入、输出和注释均以中文描述和表达。程序可以在Windows 98操作系统、Turbo C软件环境下编译运行；也可以在Windows 98操作系统、Visual C++ 6.0软件环境下编译运行。程序的源代码用的全是Visual C++中的C语言语句，所以源程序不做任何修改就可以在VC++下编译运行。

下面介绍在2种不同的运行环境下编译运行C语言源程序的过程，供上机练习时参考。

1.1　Windows 98和Visual C++ 6.0的编译环境

Visual C++ 6.0是Microsoft公司推出的、目前使用非常广泛的可视化编程环境，为使用者提供了强大的开发能力。本书中的每一个程序的源代码用的都是VC++中的C语言语句，所以程序可以不做任何修改就可在VC++下编译运行。只要使用中文版的Windows 98操作系统，程序可以在中文版或英文版Visual C++ 6.0环境下编译运行。

在运行程序前，应安装Microsoft Visual C++ 6.0的开发环境。在运行每个程序时，请先阅读每个程序的题目要求、结构说明及有关的分析和解释。下面以第6章的"二叉树中序遍历"习题为示例，说明程序运行的操作步骤。

1. 在硬盘上建立一个C++程序运行的目录如"c:\temp\数据结构"。

2. 把附带光盘"二叉树"子目录下的"二叉树中序遍历.c"源程序及有关文件包括"二叉树.c"源程序、根目录下的datatstru.h头文件复制到上面建立的"c:\temp\数据结构"目录之下。

3. 进入 Microsoft Visual C++ 6.0 开发环境，如图 1-1 所示。这是对应运行每一个数据结构习题最基本的 Visual C++ 6.0 开发环境界面。屏幕的最上端是标题栏，标题栏用于显示应用程序名和所打开的文件名。标题栏下面是菜单栏和工具栏。工具栏左下方是工作区窗口，右下方是编辑窗口，因为数据结构的习题是用 C 语言编写的，所以工具栏下方的工作区窗口没有用到，可将其关闭。最下方是状态栏。状态栏上面是输出窗口，用于显示程序编译、连接、运行过程中的有关信息。

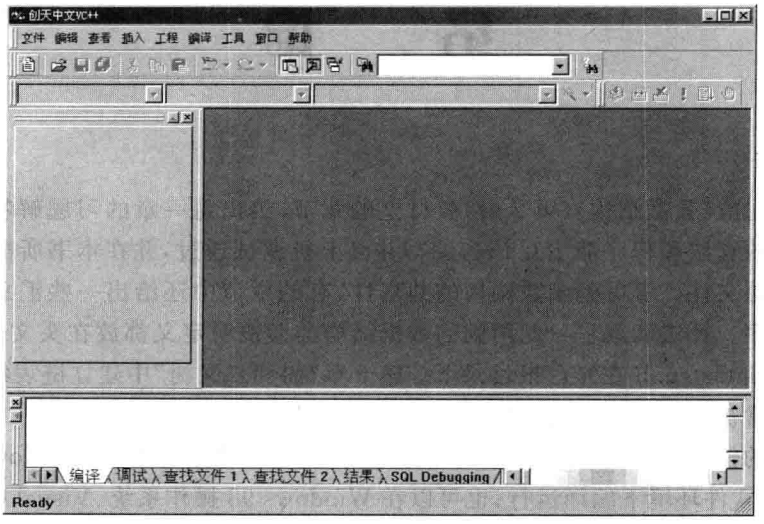


图 1-1　Visual C++ 6.0 开发环境界面(中文版)

4. 打开源程序：选择“文件”(File)->“打开”(Open)，选择“c:\temp\数据结构”目录下的“二叉树中序遍历.c”文件。在编辑窗口即可观察到这个文件的源代码，如图 1-2 所示。

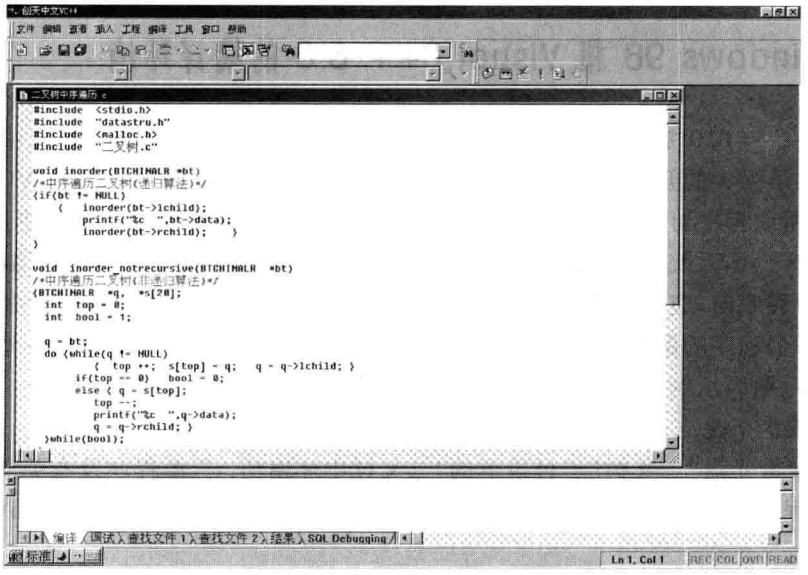


图 1-2　打开源程序

5. 对源程序进行编译操作：选择“编译”(Build)->“编译二叉树中序遍历.c”(Compile)执行编译。编译过程中出现的错误会显示在下面的输出窗口，根据错误信息提示，修改程序错误，直至错误和警告信息为0，如图1-3所示。

图1-3 对源程序进行编译操作

6. 程序运行：选择“编译”(Build)->“执行二叉树中序遍历.exe”(Execute)执行连接和运行操作。当程序运行时，将会弹出一个窗口，运行程序，显示信息，或等待用户输入数据。程序运行的结果也将显现在同一窗口内，如图1-4所示。

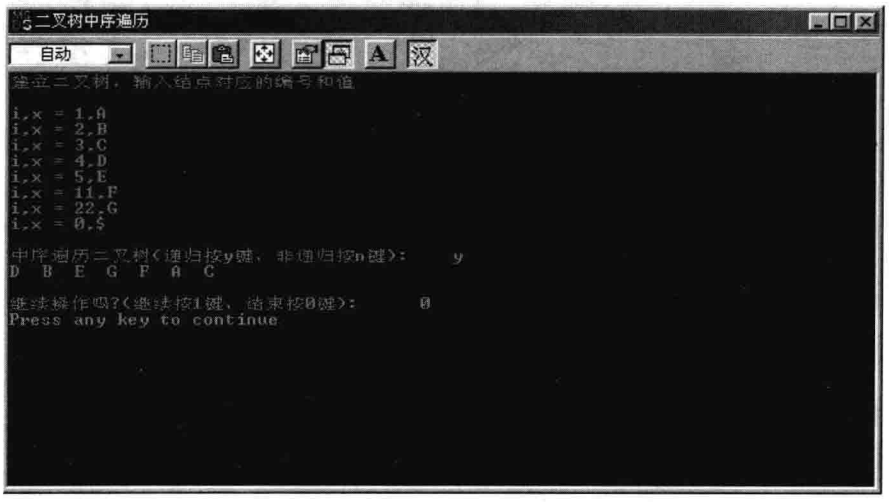


图1-4 程序运行显示

7. 本习题集中全部程序均在 Windows 98 操作系统、Microsoft Visual C ++ 6.0 软件环境下编译运行通过。

8. 有关 Microsoft Visual C ++ 6.0 软件本身的内容,请读者参考相关的书籍。

9. 最后需要说明的一点是:程序主要是在功能上和逻辑上实现了题目的要求,程序中一般没有对输入数据的合法性进行严格的判断和校验,输入数据的合法性由用户保证。因而如果发生用户输入数据不当时,程序可能会呈现出错的异常情况,遇到这种情况发生,只须重新运行源程序即可。

1.2 Windows 和 Turbo C 的编译环境

在运行程序前,应安装 Turbo C 软件。有关 Turbo C 软件本身的内容,请读者参考相关的书籍。在运行每个程序时,请先阅读每个程序的题目要求、结构说明及有关的分析和解释。下面以第 2 章的"顺序表并集"题目为示例,说明程序运行的操作步骤。

1. 在硬盘上建立一个 Turbo C 程序运行的目录如 c:\temp\sjjg。

2. 把附带光盘"线性表"子目录下的"顺序表并集.c"源程序及根目录下的 datatstru.h 头文件复制到 c:\temp\sjjg 所在的目录之下。

3. 进入 Microsoft Turbo C 开发环境,如图 1-5 所示。在这个开发环境下,程序中的中文输入和中文输出以及源程序中的中文注释均可正常显示。

图 1-5 Microsoft Turbo C 开发环境

4. 打开源程序:选择 File->Lode 打开"c:\temp\sjjg\顺序表并集.c"文件。在编辑窗口即可观察到这个文件的源代码,如图 1-6 所示。

5. 对源程序进行编译操作:选择 Compile->Compile to OBJ 执行编译。编译中出现的错误显示在下方 Message 窗口中,当编译通过,TC 窗口显示如图 1-7 所示。

图 1-6 打开源程序

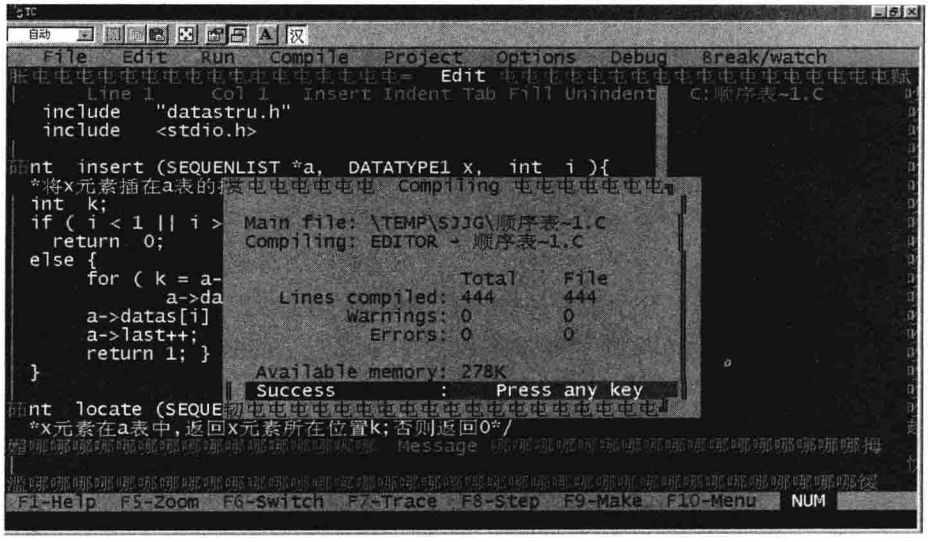


图 1-7 程序进行编译操作

6. 编译通过后选择 Run->Run 执行连接和运行操作。当程序运行时,将会弹出一个窗口,运行程序,显示信息,或等待用户输入数据。程序运行的结果数据将显现在另一窗口内,如图 1-8 所示。

7. 本习题集中全部程序均在 Windows 98 操作系统、Turbo C 软件环境下编译运行通过。要注意的是,在 Windows 98 操作系统、Turbo C 软件环境下编译运行时,程序中包含的头文件<malloc. h>均应改为<alloc. h>;包含的"二叉树. c"文件须改为包含完整路径的英文文件名方可。

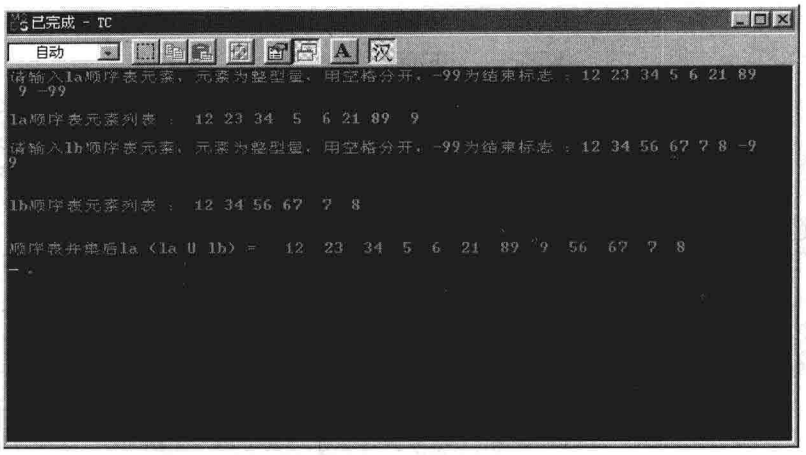


图 1-8 TC 程序运行窗口显示

1.3 习题解析

【习题 1】 简述下列术语：数据、数据元素、数据项、数据逻辑结构、数据存储结构、数据类型、算法。

【解答】

数据：是指所有能输入到计算机中并被计算机程序处理的符号的总称。

数据元素：是数据的基本单位，在计算机程序中通常作为一个整体进行考虑和处理。一个数据元素可以由若干个数据项组成，也可以只由一个数据项组成。数据元素又被称为元素、结点或记录。

数据项：是数据的不可分割的有独立含义的最小单位，数据项有时也称字段、域。数据、数据元素、数据项实际上反映了数据组织的三个层次：数据可由若干个数据元素构成，而数据元素又可以由一个或若干个数据项组成。

数据逻辑结构：数据结构主要就是研究数据元素之间的关联方式。数据元素之间存在的一种或多种特定的关系被称为数据的逻辑结构。

数据存储结构：研究数据在计算机中的存放方式，称为数据的物理结构，又称存储结构。数据的存储结构是数据在计算机存储器中的实现。数据元素在计算机中主要有两种不同的存储方法——顺序存储结构和链式存储结构。

数据类型：是和数据结构密切相关的一个概念。数据类型可分为两类，一类是非结构的原子类型，如 C 语言中的基本类型(整型、实型、字符型等)、指针类型和空类型。另一类是结构类型，它的成分可以是多个结构类型组成，是可以分解的。

算法：算法是指解决问题的一种方法或过程描述。

【习题 2】 分析下面语句段执行的时间复杂度。

(1)

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

```
for (j=1; j<=n ; j++)
    s++;
```

【解答】

双重 for 循环语句，其中外循环 n 次，对每一次外循环，内循环 s ++ ；语句执行 n 次。总的 s ++ ；语句共执行 n^2 次，时间复杂度为 $O(n^2)$。

(2)

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

【解答】

双重 for 循环语句，其中外循环 n 次，对每一次外循环，内循环 s ++ ；语句执行次数都在变化。

第一次外循环时，内循环 s ++ ；语句执行次数为 n 次；

第二次外循环时，内循环 s ++ ；语句执行次数为 n－1 次；

第三次外循环时，内循环 s ++ ；语句执行次数为 n－2 次；

……

第 n 次外循环时，内循环 s ++ ；语句执行次数为 1 次。

总之，“s ++ ；” 语句共执行 n＋(n－1)＋(n－2)＋…＋2＋1＝1/2n(n＋1)次，时间复杂度为 $O(n^2)$。

(3)

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

【解答】

双重 for 循环语句，其中外循环 n 次，对每一次外循环，内循环 s ++ ；语句执行次数都在变化。

第一次外循环时，内循环 s ++ ；语句执行次数为 1 次；

第二次外循环时，内循环 s ++ ；语句执行次数为 2 次；

第三次外循环时，内循环 s ++ ；语句执行次数为 3 次；

……

第 n 次外循环时，内循环 s ++ ；语句执行次数为 n 次。

总之，“s ++ ；”语句共执行 1＋2＋3＋…＋(n－2)＋(n－1)＋n＝1/2n(n＋1)次，时间复杂度为 $O(n^2)$。

(4)

```
i=1; k=0;
while (i<=n-1) {
  k+=10*i;
  i++;
```

```
}
```

【解答】

i=1；语句执行 1 次。

k=0；语句执行 1 次。

while 循环语句在(i≤n－1)条件满足时，执行 k＋＝10＊i；和 i＋＋；两条语句。

当 n=1 时；while 循环条件(i≤n－1)不满足，k＋＝10＊i；和 i＋＋；两条语句不执行。

当 n=2 时；while 循环条件(i≤n－1)满足一次，k＋＝10＊i；和 i＋＋；两条语句执行一次。

当 n=3 时；while 循环条件(i≤n－1)满足二次，k＋＝10＊i；和 i＋＋；两条语句执行两次。

……

以此类推，总的语句执行次数为 1＋1＋2＊(n－1)次，时间复杂度为 O(n)。

【习题 3】 试画出与下列程序段等价的流程图。

(1)

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

【解答】

流程图如图 1-9 所示。

(2)

```
i=0;
do {
    i++;
}while ((i !=n) && ( a[i] !=x ));
```

【解答】

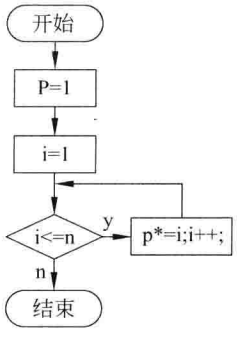


图 1-9 习题 3 中(1)的流程图

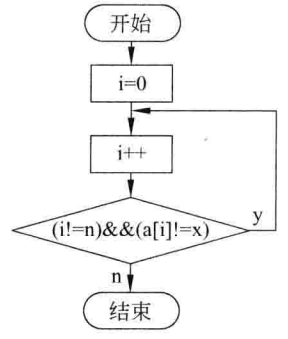


图 1-10 习题 3 中(2)的流程图

【习题 4】 按 n 的增长率由小至大顺序排列下列各函数。

$(2/3)^n$，$(3/2)^n$，n^2，n^n，$n!$，2^n，n，$\log_2 n$，n^3

【解答】

$\log_2 n < n < n^2 < n^3 < (2/3)^n < (3/2)^n < 2^n < n! < n^n$

【习题 5】 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。

【解答】

```
void order( int x, int y, int z)
{ int a;

if(x<y) {a=x; x=y; y=a;};
if(x<z) {a=x; x=z; z=a; };
if(y<z) {a=y; y=z; z=a; };
printf("从大至小依次输出顺序读入的三个整数 X、Y 和 Z:%d %d %d\n\n", x,y,z);
}
```

【习题 6】 编一个程序，输出所有小于等于 n（n 为一个大于 2 的正整数）的素数。

【解答】

```
#include "stdio.h"

main()
{    int n,i,j;
     int flag;      /* flag为素数判别标志变量 */

     scanf("% d",&n);
     for( i=2 ; i<=n ; i++)
     {    flag=1;
          for( j=2 ; j<i ; j++)
              if( i%j==0 ) flag=0;
          if(flag) printf("%d\n",i); }
}
```

1.4 实训练习题

【习题 7】 举出一个数据结构的例子，叙述其逻辑结构、存储结构及在其结构上的操作内容。

【习题 8】 判断以下叙述的正确性。

(1) 数据项是数据的最小单位。

(2) 数据的物理结构是指数据在计算机内实际的存储形式。

(3) 顺序存储方式只能用于存储线性结构。

(4) 逻辑结构相同的数据，数据类型也一定相同。

【习题 9】 编一个程序，计算任一输入的正整数的各位数字之和。

第2章

线 性 表

线性表是一种最简单也是最基本的数据结构，用向量存储结构实现的线性表称为顺序表。它的主要基本操作有插入、删除和查找。主教材例题中顺序表的数据结构如下：

```
#define DATATYPE1 int
#define MAXSIZE 100

typedef struct
{DATATYPE1 datas[MAXSIZE];
int last;
}SEQUENLIST;
```

说明：元素的数据类型设定为整数，表长不超过100。元素长度存放在last变量中，元素在数组datas中，从下标为1的单元开始存放。

2.1 习题解析

【习题1】 一顺序表元素值递增有序，编写算法，删除顺序表中值相同的多余元素。

题目分析：在有序表上，删除值相同的多余元素（重复元素只保留一个）。删除后表仍保持有序。

【解答】

```
void delete_sqelist1(SEQUENLIST * a )
{    int i, m;
     i=0;
     printf("a.last=%d\n",a->last);
     while (i<a->last) {
         while((i<a->last)&&(a->datas[i]==a->datas[i+1]))
         {    for(m=i; m<a->last; m++)
              a->datas[m]=a->datas[m+1];
              a->last--; }
```

```
            i++; }
};
```

【习题 2】 一顺序表元素值无序，编写算法，删除顺序表中值相同的多余元素。

题目分析：顺序表中的元素不要求有序，删除值相同的多余元素（重复元素只保留一个）。

【解答】

```
void delete_sqelist2(SEQUENLIST * a )
{    int i, j, k;
     for ( i=0; i<a->last; i++) {
         j=i+1;
         while(j<a->last) {
             while(a->datas[i]==a->datas[j])
               {for(k=j; k<a->last; k++)
               a->datas[k]=a->datas[k+1];
               a->last--;}
             j++; }
     }
}
```

【习题 3】 编写程序，将一顺序表 A 中的元素逆置，算法所用的辅助空间尽可能地少。

题目分析：按输入的数据建立一个顺序表，利用最少的辅助空间实现表中元素逆置存放。

例如：顺序表 A＝{100，909，8，70，160，50，44}，逆置后，A＝{44，50，160，70，8，909，100}。

下面是实现顺序表中元素逆置的两个源程序，体现了两种不同的算法。

【解答 1】

```
#include "datastru.h"
#include <stdio.h>

main()
{  SEQUENLIST a;
   int i, j, k, temp;

printf("请输入顺序表元素，元素为整型量，用空格分开，-99为结束标志:");
j=0; k=1; i=0; scanf("%d",&i);
while (i !=-99) { j++; a.datas[k]=i; k++; scanf("%d",&i); }  /* 输入顺序表元素 */
a.last=j;
printf("\n逆置前顺序表元素列表:");
for (i=1; i<=a.last; i++)                    /* 逆置前顺序表元素显示 */
    printf("%d",a.datas[i]);
printf("\n");
```

```
for(i=1; i<=a.last / 2; i++)                    /* 逆置顺序表元素 */
    {  temp=a.datas[i]; a.datas[i]=a.datas[a.last-i+1];
       a.datas[a.last-i+1]=temp;}
printf("\n逆置后顺序表元素列表 : ");            /* 逆置后顺序表元素显示 */
for (i=1; i<=a.last; i++)
    printf("%d",a.datas[i]);
printf("\n");
}
```

【解答 2】

```
#include "datastru.h"
#include <stdio.h>

main( )
{  SEQUENLIST a;
   int i, j, k, temp;

printf("请输入顺序表元素,元素为整型量,用空格分开,-99为结束标志:");
j=0; k=1; i=0; scanf("%d",&i);
while (i!=-99) { j++; a.datas[k]=i; k++; scanf("%d",&i);}/* 输入顺序表元素 */
a.last=j;
printf("\n逆置前顺序表元素列表:");
for (i=1; i<=a.last; i++)                       /* 逆置前顺序表元素显示 */
    printf("%d",a.datas[i]);
printf("\n");
for(i= 1; i<=a.last / 2; i++)                   /* 逆置顺序表元素 */
    {  temp=a.datas[i]; a.datas[i]=a.datas[a.last-i+1];
       a.datas[a.last-i+1]=temp;}
printf("\n逆置后顺序表元素列表:");              /* 逆置后顺序表元素显示 */
  for (i=1; i<=a.last; i++)
    printf("%d",a.datas[i]);
printf("\n");
}
```

【习题 4】 编写程序,输出已知顺序表 A 中元素的最大值和次最大值。

题目分析:按输入的数据建立一个顺序表,表中元素值不可以重复。求出表中最大值元素和次大值元素。

注意:程序要求元素个数在 2 个以上。

例如:顺序表 A={10, 23, 34, 5, 61, 72, 29, 20},运行结果:最大值元素是 72,次大值元素是 61。

【解答】

```
#include "datastru.h"
```

```
#include <stdio.h>

main( )
{  SEQUENLIST a;
   int i, j, k, max, submax;

printf("请输入顺序表元素,元素为整型量,用空格分开,-99为结束标志:");
j=0; k=1; i=0; scanf("%d",&i);
while (i !=-99) { j++; a.datas[k]=i; k++; scanf("%d",&i); }
a.last=j;
printf("\n顺序表元素列表:");
for (i=1; i<=a.last; i++)
    printf("% d",a.datas[i]);
printf("\n");
if (a.datas[1]>a.datas[2])
  { max=a.datas[1]; submax=a.datas[2]; }/* 初始最大值元素 max 和次大值元素 submax */
else {max=a.datas[2]; submax=a.datas[1]; }
for(i=3; i<=a.last; i++)
/* 求最大值元素 max 和次大值元素 submax */
    { if (a.datas[i]>max)
      { submax=max; max=a.datas[i]; }
    else if (a.datas[i]>submax) submax=a.datas[i];
  }
printf("\n最大值元素=%d, 次大值元素=%d",max,submax);
printf("\n");
}
```

【习题 5】 设有两个按元素值递增有序的顺序表 la 和 lb,编写程序将 la 表和 lb 表归并成一个新的递增有序的顺序表 lc(值相同的元素均保留在 lc 表中)。此题的核心算法见主教材中第 2 章的例 2-3。

题目分析:按输入的数据建立二个有序表 la 和 lb(元素值递增有序),合并成一个新的递增有序的顺序表 lc。值相同的元素均保留在 lc 中,即 lc 表长=la 表长+lb 表长。

测试数据:la={10, 23, 34, 5, 61, 72, 29, 20}, lb={1, 3, 34, 61, 56, 21, 11}。

运行结果:lc={1, 3, 5, 10, 11, 20, 21, 23, 29, 34, 34, 56, 61, 61, 72},输入表中的数据可以是无序的,但程序中对应建立的 la 和 lb 表是有序表,合并后新的顺序表 lc 中元素也递增有序。程序运行过程如图 2-1 所示。

【解答】

```
#include "datastru.h"
#include <stdio.h>

void merge_sqlist(SEQUENLIST la,SEQUENLIST lb,SEQUENLIST * lc){
/* 两有序表合并 */
```

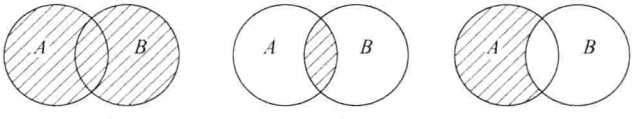
图 2-1　两个有序表合并运行过程

```
  int i , j , k ;

  i=j=k=1 ;
  while( i<=la.last && j<=lb.last )
      if( la.datas[i]<=lb.datas[j])
        {lc->datas[k]=la.datas[i] ;
        k++; i++;}
      else
        {lc->datas[k]=lb.datas[j] ;
        k++; j++;}
while( i<=la.last )
  {  lc->datas[k]=la.datas[i] ;
     k++; i++;}
while( j<=lb.last )
{  lc->datas[k]=lb.datas[j] ;
   k++; j++;}
  lc->last=k-1;
  return;
}

main( )
{  SEQUENLIST la, lb, lc;
   int i, k, m;

printf("请输入 la 顺序表元素,元素为整型量,用空格分开,-99 为结束标志:");
la.last=0; i=0; scanf("%d",&i);
while (i !=-99) {/*输入 la 顺序表元素,建立有序表*/
    k=la.last;
```

```
    while((k>=1) && ( i<la.datas[k])) k--;
    for(m=la.last; m>=k+1; m--) la.datas[m+1]=la.datas[m];
    la.datas[k+1]=i; la.last++;
    scanf("%d",&i);}
printf("\n\n 请输入 lb 顺序表元素,元素为整型量,用空格分开,-99 为结束标志:");
lb.last=0; i=0; scanf("%d",&i);
while (i !=-99) {/* 输入 lb 顺序表元素,建立有序表 */
    k=lb.last;
    while((k>=1) && ( i<lb.datas[k])) k--;
    for(m=lb.last; m>=k+1; m--) lb.datas[m+1]=lb.datas[m];
    lb.datas[k+1]=i; lb.last++;
    scanf("%d",&i); }
printf("\nla 有序表元素列表:");
for (i=1; i<=la.last; i++)
  printf("%d",la.datas[i]);
printf("\n");
printf("\nlb 有序表元素列表:");
for (i=1; i<=lb.last; i++)
  printf("%d",lb.datas[i]);
printf("\n");
merge_sqlist (la, lb, &lc);
printf("\n 合并后 lc 有序表元素列表:");
for (i=1; i<=lc.last; i++)
  printf(" %d",lc.datas[i]);
printf("\n");
}
```

【习题 6】 已知两个顺序表 la 和 lb,同一表中无重复元素,编写程序实现 la 和 lb 的并集运算,并集结果放在 la 表中。

集合上最基本的操作是求集合的并、交、差运算。例如,A, B 是两个集合,A={a, b, c},B={b, d},则并集 $A \cup B$={a, b, c, d},交集 $A \cap B$={b},差集 $A-B$={a, c},对应集合的文氏(Venn)图如图 2-2 所示。集合运算中,集合的存储形式可采用顺序表,集合中元素的类型为整型量。



(a) $A \cup B$(或 $A+B$)　(b) $A \cap B$(或 $A*B$)　(c) $A-B$

图 2-2　集合的文氏(Venn)图

题目分析:按输入的数据建立两个集合 la 和 lb,同一集合中无重复元素,用顺序表存放。对两集合进行并集运算 la∪lb,结果放在 la 表中。

算法思路:将 lb 表中的每一个元素作为给定值放在 la 表中查找,若给定值在 la 表中

存在,则不作任何运算处理;若给定值在 la 表中不存在,则将此元素加入 la 表中(追加在 la 表的最后即可)。

注意:输入的数据应确保同一集合中无重复元素,程序不作检查。

测试数据:la={10, 23, 34, 5, 61, 72, 29, 20}, lb={ 5, 4, 76, 61, 29}。

运行结果:la={10, 23, 34, 5, 61, 72, 29, 20, 4, 76},顺序表存放的两个集合 la 和 lb 进行并集运算 la∪lb 的过程如图 2-3 所示。

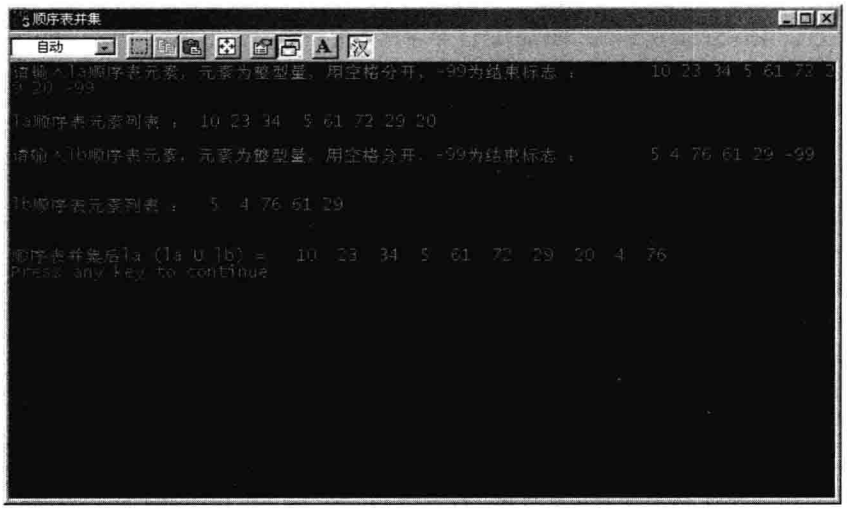


图 2-3　集合 la 和 lb 进行并集运算 la∪lb 的过程

【解答】

```
#include "datastru.h"
#include <stdio.h>

int insert (SEQUENLIST * a, DATATYPE1 x, int i ){
/* 将 x 元素插在 a 表中指定的 i 位置上 */
  int k;

  if ( i<1 || i>a->last+1 || a->last>=MAXSIZE)
    return 0;
  else {for ( k=a->last ; k>=i ; k--)
        a->datas[ k+1 ]=a->datas[k];
      a->datas[i]=x; a->last++;
      return 1; }
  }

int locate (SEQUENLIST * a, DATATYPE1 x){
/* x元素在 a 表中,返回 x 元素所在位置 k;否则返回 0) */
  int k;
```

```
  k=1;
  while (k<=a->last && a->datas[k] !=x )
    k++;
  if ( k<=a->last)
    return k ;
  else return 0;
}

void unite ( SEQUENLIST * la, SEQUENLIST lb)
{  int i;

   for ( i=1 ; i<=lb.last ; i++)
   if(!locate(la,lb.datas[i]))            /* 如果 lb 表中元素不在 la 表中, */
   insert ( la, lb.datas[i], la->last+1);  /* 则将 lb 表中的该元素插到 la 表的最后 */
}

main( )
{  SEQUENLIST a, b;
   int i, j, k;

printf("请输入 la 顺序表元素,元素为整型量,用空格分开,-99 为结束标志:");
j=0; k=1; i=0; scanf("%d",&i);
while(i !=-99){j++; a.datas[k]=i; k++; scanf("%d",&i);}/* 输入 la 顺序表元素 */
a.last=j;
printf("\nla 顺序表元素列表 : ");
for (i=1; i<=a.last; i++)
    printf("%d",a.datas[i]);
printf("\n");
printf("\n 请输入 lb 顺序表元素,元素为整型量,用空格分开,-99 为结束标志:");
j=0; k=1; i=0; scanf("%d",&i);
while(i !=-99){j++; b.datas[k]=i; k++; scanf("%d",&i);}/* 输入 lb 顺序表元素 */
b.last=j;
printf("\n\nlb 顺序表元素列表:");
for (i=1; i<=b.last; i++)
    printf("%d",b.datas[i]);
printf("\n");
unite(&a,b);                              /* a 顺序表和 b 顺序表并集 */
printf("\n\n 顺序表并集后 la (la ∪ lb)=");
    for (i=1; i<=a.last; i++)
      printf(" %d",a.datas[i]);
printf("\n");
}
```

2.2 实训练习题

【习题 7】 已知一顺序表 A,设计一算法删除顺序表中值为 item 的数据元素。

【习题 8】 已知一顺序表 A,表中都是不相等的整数。设计一算法,把表中所有的奇数移到所有的偶数前面去。

【习题 9】 已知两个顺序表 la 和 lb,同一表中无重复元素,编写程序实现 la 和 lb 的交集运算,交集结果放在 la 表中。

【习题 10】 已知两个顺序表 la 和 lb,同一表中无重复元素,编写程序实现 la 和 lb 的差集运算,差集结果放在 la 表中。

链式存储结构

链式存储结构是存放数据的另一种重要方式。通常将链式存储的线性表称为单链表。链式存储结构相对于顺序存储结构最大的不同，一是数据的逻辑结构和物理存储相互独立，逻辑关系上相邻的元素物理位置上不一定是相邻的。二是数据元素所需的存储空间可动态生成获得。

本章的习题重点是单链表的各种建立算法及在单链表上的常见操作的程序实现。单链表的数据结构如下：

```
#define DATATYPE2 char

typedef struct node
{ DATATYPE2 data;
struct node * next;
}LINKLIST;
```

说明：单链表中元素结点的数据域为 data，类型设定为字符，结点的指针域为 next。

3.1 习题解析

【习题 1】 若线性表的元素总数基本稳定，且很少进行插入和删除，但要求快速存取表中元素，应采用哪种存储结构？为什么？

【解答】 由于顺序存储结构一旦确定了起始位置，对线性表中的任何一个元素都可以进行随机存取，且存取速度较高。已知线性表的元素总数基本稳定，且很少进行插入和删除，恰好避开了顺序存储结构的缺点。因此，应采用顺序存储结构。

【习题 2】 对线性表而言，什么情况下采用链表比顺序表好？

【解答】 如在线性表上经常进行插入和删除的操作，采用链表可避免移动大量的元素；而且经常进行插入和删除的操作，线性表需要的空间量难以预估，采用链表可以动态分配空间。

【习题 3】 分析单链表、循环链表和双向链表的相同点和各自的特点。

【解答】 相同点在于都是链式存储结构，表中元素对应的结点中除自身的信息域，还

有关联信息的指针域。

单链表的特点是链表结构较简单,表的操作也较简单,但从单链表中某一结点不能直接找到其前驱结点。

而双向链表因结点中既有指向后继结点的信息又有指向前驱结点的信息,所以从双向链表中某一结点既可直接找到其后继结点,又能直接找到其前驱结点。但表的操作比单链表要复杂些。

循环链表的特点是可从表中任一结点出发遍历表中全部元素结点。

【习题 4】 已知 L 是无表头结点的单链表,且 P 结点既不是首结点,也不是尾结点,试从下列提供的语句中选出合适的语句序列。

(1) 在 P 结点后插入 S 结点:_____。

(2) 在 P 结点前插入 S 结点:_____。

(3) 在表首插入 S 结点:_____。

(4) 在表尾插入 S 结点:_____。

① P—>next=S;

② P—>next=P—>next—>next;

③ P—>next=S—>next;

④ S—>next=P—>next;

⑤ S—>next=L;

⑥ S—>next=P;

⑦ S—>next=NULL;

⑧ Q=P;

⑨ while (P—>next !=Q) P=P—>next;

⑩ while (Q—>next !=NULL) Q=Q—>next;

⑪ P=Q;

⑫ P=L;

⑬ L=S;

⑭ L=P;

【解答】 在 P 结点后插入 S 结点:_③、①_。

在 P 结点前插入 S 结点:_⑧、⑫、⑨、④、①_。

在表首插入 S 结点:_⑤、⑬_。

在表尾插入 S 结点:_⑦、⑪、①_。

【习题 5】 单项选择题。

(1) 在一个长度为 n 的顺序表中向第 i 个元素($0<i\leqslant n+1$)之前插入一个新元素时,需向后移动_____个元素。

A. $n-i$　　B. $n-i+1$　　C. $n-i-1$　　D. i

【解答】 在第 i 个元素($0<i\leqslant n+1$)之前插入一个新元素时,应将第 i 个元素及之后的全部元素后移,后移的元素个数为 $n-i+1$ 个,所以应选 B。注意题中 i 的范围 $0<i\leqslant n+1$ 是考虑了可在长度为 n 的顺序表中的 n 个元素之前插入新元素,并可在表中最后一

个元素之后插入新元素。

(2) 线性表采用链式存储结构时,其地址_____。

A. 必须是连续的　　B. 一定是不连续的

C. 部分地址必须是连续的　　D. 连续与否均可以

【解答】 链式存储结构是动态存储结构,不要求地址是连续的,这是与顺序存储结构最大的区别,所以选 D。

(3) 在一个单链表中,删除 * p 结点之后的一个结点的操作是_____。

A. p—>next=p;　　B. p—>next—>next=p—>next;

C. p—>next—>next=p;　　D. p—>next=p—>next—>next;

【解答】 D。

(4) 在一个双链表中,在 * p 结点之后插入结点 * s 的操作是_____。

A. s—>prior=p; p—>next=s; p—>next—>prior=s; s—>next=p—>next;

B. s—>next=p—>next; p—>next—>prior=s; p—>next=s; s—>prior=p;

C. p—>next=s; s—>prior=p; s—>next=p—>next; p—>next—>prior=s;

D. p—>next—>prior=s; s—>next=p—>next; s—>prior=p; p—>next=s;

【解答】 B。

(5) 在不带头结点 * head 的单循环链表中,尾结点 * p 的条件是_____。

A. head !=NULL　　B. head—>next !=head

C. p==NULL　　D. p—>next==head

【解答】 D。

【习题 6】 判断以下叙述的正确性。

(1) 分配给单链表的内存单元地址必须是连续的。

(2) 与顺序表相比,在链表上实现顺序访问,其算法的效率比较低。

(3) 向顺序表中插入一个元素,平均要移动约一半的元素。

(4) 在带头结点的循环单链表中,任何一个结点的指针都不可能为空。

(5) 在有 n 个元素的顺序表中,删除任意一个元素所需移动结点的平均次数为 $n-1$ 个。

【解答】

(1) 错误。分配给单链表的内存单元地址可以是不连续的。

(2) 错误。在顺序表和链表上实现顺序访问,其算法的时间复杂度都是 $O(n)$。

(3) 正确。

(4) 正确。

(5) 错误。删除第 1 个元素移动结点次数为 $n-1$ 个。删除最后一个元素没有结点移动。删除任意一个元素所需移动结点的平均次数约一半的元素为 $n/2$。

【习题 7】 设计一算法,在一个不带头结点的单链表上,用最少的辅助空间实现单链表元素的逆置。

【解答】

```
LINKLIST * invertlink(LINKLIST * head){
```

```
/*单链表元素逆置*/
  LINKLIST *p, *q, *r;

  q=NULL; p=head;
  while(p !=NULL)
      {r=q; q=p ; p=p->next; q->next=r;}
  return q;
}
```

【习题 8】 按下列要求建立单链表,编写程序实现:

(1) 按头插入法建立不带头结点的单链表。

题目分析:将输入的数据按头插入法建立一个不带头结点的单链表。输入数据时以输入一串字符的方式实现,“$”字符为输入结束字符。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_nohead(LINKLIST *head){
/*不带头结点的单链表:输出单链表元素值并计数*/
  int i=0;
  LINKLIST *p;

  p=head;
  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {printf(" %c",p->data); i++; p=p->next;}
  printf("\n");
  return i;
}

LINKLIST *creatlink_nohead_head(LINKLIST *head) {
/*用头插入法建立不带头结点的单链表*/
  LINKLIST *t;
  char ch;

  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 : ");
  while((ch=getchar())!='$ ')
    {  t=(LINKLIST *) malloc(sizeof(LINKLIST));
       t->data=ch; t->next=head; head=t;}
return(head);
}
```

```
main()
  { LINKLIST * head=NULL;
    int num;

    printf("\n 建立单链表\n\n");
    head=creatlink_nohead_head(head);
    fflush(stdin);
    num=count_nohead(head);
    printf("单链表元素个数=%d\n", num);
}
```

(2) 按头插入法建立一个带头结点的单链表。

题目分析：将输入的数据按头插入法建立一个带头结点的单链表。输入数据方式同上。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_head(LINKLIST * head){
/* 带头结点的单链表:输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;

  p= head->next;
  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {printf(" % c",p->data); i++; p=p->next;}
  printf("\n");
  return i;
}

LINKLIST * creatlink_head_head(LINKLIST * head) {
/* 用头插入法建立带头结点的单链表 */
  LINKLIST * t;
  char ch;

  t=(LINKLIST * )malloc(sizeof(LINKLIST));
  head=t; t->next=NULL;
  printf("单链表元素值为单个字符, 连续输入,$ 为结束字符 : ");
  while((ch=getchar())!='$ ')
    {t=(LINKLIST * ) malloc(sizeof(LINKLIST));
     t->data=ch; t->next=head->next; head->next=t;}
  return(head);
```

```
}

main()
  { LINKLIST * head=NULL;
    int num;

    printf("\n 建立单链表\n\n");
    head=creatlink_head_head(head);
    fflush(stdin);
    num=count_head(head);
    printf("单链表元素个数=%d\n", num);
}
```

(3) 按尾插入法建立不带头结点的单链表。

题目分析：将输入的数据按尾插入法建立一个不带头结点的单链表。输入数据方式同上。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_nohead(LINKLIST * head){
/* 不带头结点的单链表:输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;
  p=head;

  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {i++; printf(" % c",p->data); p=p->next;}
  printf("\n");
  return i;
}

LINKLIST * creatlink_nohead_rail(LINKLIST * head){
/* 用尾插入法建立不带头结点的单链表 */
  LINKLIST * last, * t;
  char ch;

  last= head;
  printf("单链表元素值为单个字符, 连续输入,$为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; t->next=NULL;
      if (head==NULL) {head=t; last=t;}
```

```
        else { last->next=t; last=t;}
    }
    return (head);
}

main()
{
    LINKLIST * head= NULL;
    int num;

    printf("\n 建立单链表\n\n");
    head= creatlink_nohead_rail(head);
    fflush(stdin);
    num= count_nohead(head);
    printf("单链表元素个数= % d\n", num);
}
```

(4) 按尾插入法建立一个带头结点的单链表。

题目分析：将输入的数据按尾插入法建立一个带头结点的单链表。输入数据方式同上。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int count_head(LINKLIST * head){
/* 带头结点的单链表：输出单链表元素值并计数 */
    int i=0;
    LINKLIST * p;

    p=head->next;
    printf("输出单链表元素值 : ");
    while(p !=NULL)
      {i++; printf(" % c",p->data); p=p->next;}
    printf("\n");
    return i;
}

LINKLIST * creatlink_head_rail(LINKLIST * head){
/* 用尾插入法建立带头结点的单链表 */
    LINKLIST * last, * t;
    char ch;

    t= (LINKLIST * )malloc(sizeof(LINKLIST));
```

```
    head=t; last=t; t->next=NULL;
    printf("单链表元素值为单个字符,连续输入,$为结束字符 : ");
    while ((ch=getchar()) !='$ ')
      { t=(LINKLIST * )malloc(sizeof(LINKLIST));
        t->data=ch; t->next=NULL;
        last->next=t; last=t;}
    return (head);
}

main()
{ LINKLIST * head=NULL;
  int num;

printf("\n 建立单链表\n\n");
head=creatlink_head_rail(head);
fflush(stdin);
num=count_head(head);
printf("单链表元素个数=%d\n", num);
}
```

【习题 9】 设 L 为带头结点的单链表,表中元素值递增有序,编写程序删除表中值相同的多余元素。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

void delete(LINKLIST * a){
/* 在有序链表中删除重复元素,保留一个 */
LINKLIST * la;

la=a->next;
while(la !=NULL && la->next !=NULL)
  if (la->data==la->next->data) la->next=la->next->next;
  else la=la->next;
}

int count_head(LINKLIST * head){
/* 带头结点的单链表:输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;

  p=head->next;
  printf("输出单链表元素值 : ");
```

```
  while(p !=NULL)
    {i++; printf(" % c",p->data); p=p->next;}
  printf("\n\n");
  return i;
}

LINKLIST * creatlink_order_head(LINKLIST * head)
/* 建立带头结点的有序单链表 */
{ LINKLIST * t, * p, * q;
  char ch;

  t= (LINKLIST * )malloc(sizeof(LINKLIST));
  head=t; t->next=NULL;
  printf("单链表元素值为单个字符，连续输入,$ 为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t= (LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; q=head; p=head->next;
      while( p !=NULL && p->data<=ch) {q=p; p=p->next;}
      q->next=t; t->next=p;
    }
  return(head);
}

main()
{
  LINKLIST * head=NULL;
  int num;

  printf("\n 建立单链表\n\n");
  head=creatlink_order_head(head);
  fflush(stdin);
  num=count_head(head);
  printf("\n 删除重复元素后\n\n");
  delete(head);
  num=count_head(head);
}
```

【习题 10】 设 L 为带头结点的单链表，表中元素值无序，编写程序删除表中值相同的多余元素。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>
```

```
void delete(LINKLIST * a){
/* 在无序单链表中删除重复元素,保留一个 */
LINKLIST * la, * p, * q;

la=a->next;
while(la !=NULL)
  { q=la; p=la->next;
    while(p !=NULL)
      { if (p->data==la->data) { p=p->next; q->next=p;}
        else { q=p; p=p->next;}
      }
    la=la->next;
  }
}

int count_head(LINKLIST * head){
/* 带头结点的单链表:输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;

  p=head->next;
  printf("输出单链表元素值 :");
  while(p !=NULL)
    {i++; printf(" % c",p->data); p=p->next;}
  printf("\n");
  return i;
}

LINKLIST * creatlink_head_rail(LINKLIST * head){
/* 用尾插入法建立带头结点的单链表 */
  LINKLIST * last, * t;
  char ch;

  t=(LINKLIST * )malloc(sizeof(LINKLIST));
  head=t; last=t; t->next=NULL;
  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 :");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; t->next=NULL;
      last->next=t; last=t;}
  return (head);
}

main()
```

```
{
  LINKLIST * head=NULL;
  int num;

printf("\n 建立单链表\n\n");
head=creatlink_head_rail(head);
fflush(stdin);
num=count_head(head);
printf("\n 删除重复元素后\n\n");
delete(head);
num=count_head(head);
}
```

【习题 11】 在一个带头结点的单链表上，表中元素值递增有序，编一程序，在单链表中插入一个元素，插入后表中元素值仍保持递增有序。

题目分析：建立一个带头结点的数据从小到大有序排列的单链表。将输入的给定值插入单链表中正确的位置上。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

inser_order(DATATYPE2 x, LINKLIST * head){
/*将给定值 x 插入有序表中，并保持有序*/
  LINKLIST * pr, * pn, * pp;

  pr=head; pn=head->next;
  while(pn !=NULL && pn->data<x)
    {pr=pn; pn=pn->next;}
  pp= (LINKLIST * )malloc(sizeof(LINKLIST));
  pp->data=x; pp->next=pr->next; pr->next=pp;
}

int count_head(LINKLIST * head){
/*带头结点的单链表:输出单链表元素值并计数*/
  int i=0;
  LINKLIST * p;

  p=head->next;
  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {printf(" % c",p->data); i++; p=p->next;}
  printf("\n");
```

```
    return i;
}

LINKLIST * creatlink_order_head(LINKLIST * head)
/* 建立带头结点的有序单链表 */
{ LINKLIST * t, * p, * q;
  char ch;

  t=(LINKLIST * )malloc(sizeof(LINKLIST));
  head=t; t->next=NULL;
  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; q=head; p=head->next;
      while( p !=NULL && p->data<=ch) {q=p; p=p->next;}
      q->next=t; t->next=p;
    }
  return(head);
}

main()
{ LINKLIST * head=NULL;
  int num;
  char ch;

  printf("\n 建立单链表\n\n");
  head=creatlink_order_head(head);
  fflush(stdin);
  num=count_head(head);
  printf("单链表元素个数=% d\n", num);
  printf("\n 输入插入元素值 : ");
  ch=getchar();
  inser_order(ch, head);
  printf("\n 元素插入后\n\n");
  num=count_head(head);
  printf("插入后单链表元素个数=% d\n", num);
}
```

【习题 12】 设有两个按元素值递增有序的带头结点的单链表 A 和 B,编写程序将 A 表和 B 表归并成一个新的递增有序的带头结点的单链表 C(值相同的元素均保留在 C 表中)。此题的核心算法见主教材中第 3 章的例 3-8。

【解答】

```
#include "datastru.h"
#include <stdio.h>
```

```
#include <malloc.h>

void unite(LINKLIST * a, LINKLIST * b, LINKLIST * c){
/* 两有序链表 a, b 合并到一新链表 c 中,c 表中元素值也有序 */
LINKLIST * la, * lb, * lc, * p;

la=a->next; lb=b->next; lc=c;
while(la !=NULL && lb !=NULL)
  { if (la->data<=lb->data)
    { p= (LINKLIST * ) malloc(sizeof(LINKLIST));
      p->data=la->data; p->next=NULL;
      lc->next=p; lc=lc->next; la=la->next;
    }
  else
    { p= (LINKLIST * ) malloc(sizeof(LINKLIST));
      p->data=lb->data; p->next=NULL;
      lc->next=p; lc=lc->next; lb=lb->next;
    }
  }
  while(la !=NULL)
  { p= (LINKLIST * ) malloc(sizeof(LINKLIST));
    p->data=la->data; p->next=NULL;
    lc->next=p; lc=lc->next; la=la->next;
  }
while(lb !=NULL)
  { p= (LINKLIST * ) malloc(sizeof(LINKLIST));
    p->data=lb->data; p->next=NULL;
    lc->next=p; lc=lc->next; lb=lb->next;
  }
}

int count_head(LINKLIST * head){
/* 带头结点的单链表：输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;
  p=head->next;
  printf("输出单链表元素值 : ");
  while(p !=NULL)
    {i++; printf(" % c",p->data); p=p->next;}
  printf("\n");
  return i;
}

LINKLIST * creatlink_order_head(LINKLIST * head)
```

```
/*建立带头结点的元素值有序的单链表*/
{ LINKLIST *t, *p, *q;
  char ch;

  t=(LINKLIST *)malloc(sizeof(LINKLIST));
  head=t; t->next=NULL;
  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 :");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST *)malloc(sizeof(LINKLIST));
      t->data=ch; q=head; p=head->next;
      while( p !=NULL && p->data<=ch) {q=p; p=p->next;}
      q->next=t; t->next=p;
    }
  return(head);
}

main()
{
  LINKLIST *a1=NULL, *a2=NULL, *c;
  int num;

  printf("\n 建立单链表 a1\n\n");
  a1=creatlink_order_head(a1);
  fflush(stdin);
  num=count_head(a1);
  printf("单链表 a1 元素个数=% d\n", num);
  printf("\n 建立单链表 a2\n\n");
  a2=creatlink_order_head(a2);
  fflush(stdin);
  num=count_head(a2);
  printf("单链表 a2 元素个数=% d\n", num);
  c=(LINKLIST *)malloc(sizeof(LINKLIST));
  c->next=NULL;
  unite(a1, a2, c);
  printf("\n\n 合并到单链表 c 中\n");
  num=count_head(c);
  printf("单链表 c 元素个数=% d\n", num);
}
```

【习题 13】 设有两个线性表 A 和 B,都是单链表存储结构。同一个表中的元素各不相同,且递增有序。编写程序将 A 和 B 的并集构成一新的线性表 C,且 C 中元素也递增有序。

题目分析:第 2 章中介绍了集合的并集运算,集合的存储形式是顺序表。集合的存储形式也可采用链表,集合中元素的类型为字符。此题即按输入的数据建立两个集合 a1

和 a2，同一集合中无重复元素，单链表存放。对两集合进行并集运算 a1∪a2，结果放在 a1 表中。算法的思路同顺序表并集。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int locate(LINKLIST * a,char x)
/* 检查元素 x 是否在 a 表中 */
{LINKLIST * la;

la=a->next;
while(la !=NULL)
    if(la->data==x) return 1;
    else la=la->next;
return 0;
}

insert(LINKLIST * a,char x)
/* 将 x 元素加入 a 表中 */
{LINKLIST * p;

p=(LINKLIST * ) malloc(sizeof(LINKLIST));
p->data=x;p->next=a->next;a->next=p;
}

void unionn(LINKLIST * a1, LINKLIST * b1){
/* 两链表并集 */
LINKLIST * lb;

lb=b1->next;
while(lb !=NULL)
  { if (! locate(a1,lb->data))        /* 如果 b1 表中的一个元素不在 a1 表中 */
    insert(a1,lb->data);              /* 则将 b1 表中的该元素加入 a1 表中 */
    lb=lb->next;}
}


int count_head(LINKLIST * head){
/* 带头结点的单链表:输出单链表元素值并计数 */
  int i=0;
  LINKLIST * p;
  p=head->next;
```

```
    printf("输出单链表元素值 : ");
    while(p !=NULL)
      {printf(" % c",p->data); i++; p=p->next;}
  printf("\n");
  return i;
  }

  LINKLIST * creatlink_head_rail(LINKLIST * head){
  /* 用尾插入法建立带头结点的单链表 */
    LINKLIST * last, * t;
    char ch;

    t=(LINKLIST * )malloc(sizeof(LINKLIST));
    head=t; last=t; t->next=NULL;
    printf("单链表元素值为单个字符, 连续输入,$为结束字符 : ");
    while ((ch=getchar()) !='$ ')
      { t=(LINKLIST * )malloc(sizeof(LINKLIST)); t->data=ch;
        t->next=NULL; last->next=t; last=t;}
    return (head);
  }

  main()
    { LINKLIST * a1, * a2;
      int num;

      printf("\n 建立单链表 a1 \n\n");
      a1=NULL;
      a1=creatlink_head_rail(a1);
      fflush(stdin);
      num=count_head(a1);
      printf("\n 建立单链表 a2 \n\n");
      a2=NULL;
      a2=creatlink_head_rail(a2);
      fflush(stdin);
      num=count_head(a2);
      printf("\n\n 两链表并集运算\n\n");
      unionn(a1,a2);
      num=count_head(a1);
  }
```

图 3-1 是以单链表存储的两集合进行并集运算 a1∪a2 的过程示意图。

注意：输入的数据应确保同一集合中无重复元素，程序不作检查。

【习题 14】 单链表的综合练习题：这是一个将单链表上的各个操作合并在一综合程序中的练习。包括建立单链表、逆置单链表、在有序链表上插入一元素、删除重复元素、两

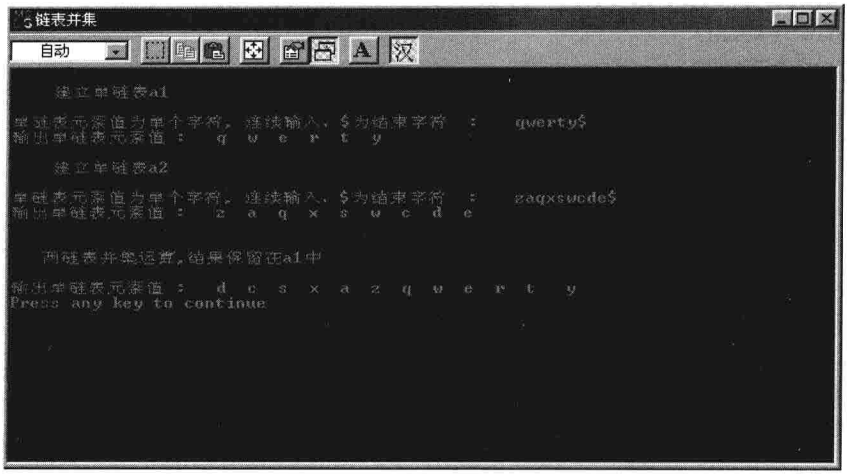


图 3-1 单链表存放的两集合进行并集运算 a1∪a2

链表合并及两链表并集等操作。通过菜单选择方式运行如图 3-2 所示。这样一些函数可共享被调用。

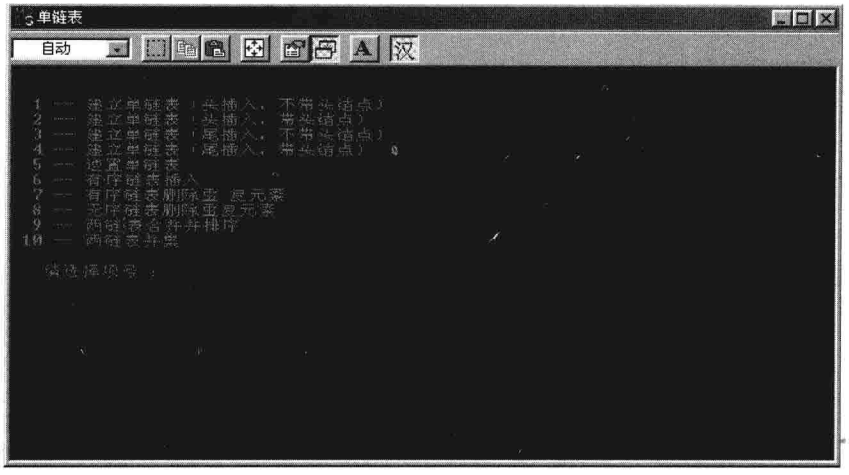


图 3-2 单链表综合习题菜单选择方式

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

int locate(LINKLIST * a,char x)
/* 检查元素 x 是否在 a 表中 */
{LINKLIST * la;
```

```
la=a->next;
while(la !=NULL)
    if(la->data==x) return 1;
    else la=la->next;
return 0;
}

insert(LINKLIST * a,char x)
/* 将 x元素加入 a表中 */
{LINKLIST * p;

p= (LINKLIST * ) malloc(sizeof(LINKLIST));p->data=x;
p->next=a->next;a->next=p;
}

void unionn(LINKLIST * a1, LINKLIST * b1){
/* 两链表并集 */
LINKLIST * lb;

lb=b1->next;
while(lb !=NULL)
  { if (!locate(a1,lb->data))      /* 如果 b1表中的一个元素不在 a1表中 */
     insert(a1,lb->data);          /* 则将 b1表中的该元素加入 a1表中 */
    lb=lb->next;}
}

void unite(LINKLIST * a, LINKLIST * b, LINKLIST * c){
/* a, b为两有序链表,合并到 c表,并保持有序 */
LINKLIST * la, * lb, * lc, * p;

la=a->next; lb=b->next; lc=c;
while(la !=NULL && lb !=NULL)
  { if (la->data<=lb->data)
      { p= (LINKLIST * ) malloc(sizeof(LINKLIST));
        p->data=la->data; p->next=NULL;
        lc->next=p; lc=lc->next; la=la->next;
      }
  else
    { p= (LINKLIST * ) malloc(sizeof(LINKLIST));
      p->data=lb->data; p->next=NULL;
      lc->next=p; lc=lc->next; lb=lb->next;
    }
  }
  while(la !=NULL)
```

```
  { p=(LINKLIST *) malloc(sizeof(LINKLIST));
    p->data=la->data; p->next=NULL;
    lc->next=p; lc=lc->next; la=la->next;
  }
while(lb !=NULL)
  { p=(LINKLIST *) malloc(sizeof(LINKLIST));
    p->data=lb->data; p->next=NULL;
    lc->next=p; lc=lc->next; lb=lb->next;
  }
}

void delete1(LINKLIST * a){
/* 无序链表中删除重复元素,重复元素保留一个 */
LINKLIST * la, * p, * q;

la=a->next;
while(la !=NULL)
  { q=la; p=la->next;
    while(p !=NULL)
      { if (p->data==la->data) { p=p->next; q->next=p;}
        else { q=p; p=p->next;}
      }
    la=la->next;
  }
}

void delete(LINKLIST * a){
/* 有序链表中删除重复元素,重复元素保留一个 */
LINKLIST * la;

la=a->next;
while(la !=NULL && la->next !=NULL)
  if (la->data==la->next->data)
    la->next=la->next->next;
  else la=la->next;
}

inser_order(DATATYPE2 x, LINKLIST * head){
/* 有序表中插入元素 x,并保持表有序 */
  LINKLIST * pr, * pn, * pp;

  pr=head; pn=head->next;
  while(pn !=NULL && pn->data<x)
    { pr=pn;
```

```
        pn=pn->next;}
    pp=(LINKLIST *)malloc(sizeof(LINKLIST));
    pp->data=x; pp->next=pr->next; pr->next=pp;
}

LINKLIST *invertlink(LINKLIST *head){
/*单链表元素逆置*/
    LINKLIST *p, *q, *r;

    q=NULL; p=head;
    while(p !=NULL)
        {r=q; q=p ; p=p->next; q->next=r;}
    return q;
}

int count_nohead(LINKLIST *head){
/*不带头结点的单链表:输出单链表元素值并计数*/
    int i=0;
    LINKLIST *p;
    p=head;
    printf("输出单链表元素值 :");
    while(p !=NULL)
      {printf("%c",p->data); i++; p=p->next;}
    printf("\n");
    return i;
}

int count_head(LINKLIST *head){
/*带头结点的单链表:输出单链表元素值并计数*/
    int i=0;
    LINKLIST *p;

    p=head->next;
    printf("输出单链表元素值 :");
    while(p !=NULL)
      {printf("%c",p->data); i++; p=p->next;}
    printf("\n");
    return i;
}

LINKLIST *creatlink_nohead_head(LINKLIST *head) {
/*用头插入法建立不带头结点的单链表*/
    LINKLIST *t;
    char ch;
```

```
  printf("单链表元素值为单个字符，连续输入，$ 为结束字符 : ");
  while((ch=getchar())!='$ ')
    {  t=(LINKLIST * ) malloc(sizeof(LINKLIST));
       t->data=ch; t->next=head; head=t;}
return(head);
}

LINKLIST * creatlink_head_head(LINKLIST * head) {
/* 用头插入法建立带头结点的单链表 */
  LINKLIST * t;
  char ch;

  t=(LINKLIST * )malloc(sizeof(LINKLIST));
  head=t;
  t->next=NULL;
  printf("单链表元素值为单个字符，连续输入，$为结束字符 : ");
  while((ch=getchar())!='$ ')
    { t=(LINKLIST * ) malloc(sizeof(LINKLIST));
      t->data=ch; t->next=head->next; head->next=t;}
  return(head);
}

LINKLIST * creatlink_nohead_rail(LINKLIST * head){
/* 用尾插入法建立不带头结点的单链表 */
  LINKLIST * last, * t;
  char ch;

  last=head;
  printf("单链表元素值为单个字符，连续输入，$ 为结束字符 : ");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST * )malloc(sizeof(LINKLIST));
      t->data=ch; t->next=NULL;
      if (head==NULL) {head=t; last=t;}
      else { last->next=t; last=t;}
  }
  return (head);
}

LINKLIST * creatlink_head_rail(LINKLIST * head){
/* 用尾插入法建立带头结点的单链表 */
  LINKLIST * last, * t;
  char ch;
```

```
    t=(LINKLIST *)malloc(sizeof(LINKLIST));
    head=t; last=t;
    t->next=NULL;
    printf("单链表元素值为单个字符,连续输入,$ 为结束字符 :");
    while ((ch=getchar()) !='$ ')
      { t=(LINKLIST *)malloc(sizeof(LINKLIST));
        t->data=ch; t->next=NULL; last->next=t; last=t;}
    return (head);
}

LINKLIST *creatlink_order_head(LINKLIST *head)
/*建立带头结点的有序单链表*/
{ LINKLIST *t, *p, *q;
  char ch;

  t=(LINKLIST *)malloc(sizeof(LINKLIST));
  head=t; t->next=NULL;
  printf("单链表元素值为单个字符,连续输入,$ 为结束字符 :");
  while ((ch=getchar()) !='$ ')
    { t=(LINKLIST *)malloc(sizeof(LINKLIST));
      t->data=ch;
      q=head; p=head->next;
      while(p !=NULL && p->data<=ch) {
      q=p; p=p->next;}
      q->next=t; t->next=p;
    }
  return(head);
}

main()
  { LINKLIST *head, *a1, *a2, *c;
    int num=0,loop,j;
    char ch;

    loop=1;
    while (loop) {
    printf("\n\n");
    printf(" 1——建立单链表(头插入,不带头结点)\n");
    printf(" 2——建立单链表(头插入,带头结点)\n");
    printf(" 3——建立单链表(尾插入,不带头结点)\n");
    printf(" 4——建立单链表(尾插入,带头结点)\n");
    printf(" 5——逆置单链表\n");
    printf(" 6——有序链表插入\n");
    printf(" 7——有序链表删除重复元素\n");
```

```
printf(" 8——无序链表删除重复元素\n");
printf(" 9——两链表合并并排序\n");
printf(" 10——两链表并集\n\n");
printf(" 请选择项号 : ");
scanf("% d",&j);
fflush(stdin);
printf("\n\n");
if(j>=1 && j<=10)
    switch(j) {
        case 1: printf("\n 建立单链表\n\n");
            head=NULL;
            head=creatlink_nohead_head(head);
            fflush(stdin);
            num=count_nohead(head);
            printf("单链表元素个数=% d\n", num);
            break;
        case 2: printf("\n 建立单链表\n\n");
            head=NULL;
            head=creatlink_head_head(head);
            fflush(stdin);
            num=count_head(head);
            printf("单链表元素个数=% d\n", num);
            break;
        case 3: printf("\n 建立单链表\n\n");
            head=NULL;
            head=creatlink_nohead_rail(head);
            fflush(stdin);
            num=count_nohead(head);
            printf("单链表元素个数=% d\n", num);
            break;
        case 4: printf("\n 建立单链表\n\n");
            head=NULL;
            head=creatlink_head_rail(head);
            fflush(stdin);
            num=count_head(head);
            printf("单链表元素个数=% d\n", num);
            break;
        case 5: printf("\n 建立单链表\n\n");
            head=NULL;
            head=creatlink_nohead_head(head);
            fflush(stdin);
            num=count_nohead(head);
            printf("单链表元素个数=% d\n", num);
            printf("\n 逆置单链表\n\n");
```

```
        head=invertlink(head);
        num=count_nohead(head);
        break;
    case 6: printf("\n 建立单链表\n\n");
        head=NULL;
        head=creatlink_order_head(head);
        fflush(stdin);
        num=count_head(head);
        printf("单链表元素个数=% d\n", num);
        printf("\n 输入插入元素值 : ");
        ch=getchar();
        inser_order(ch, head);
        printf("\n 元素插入后\n\n");
        num=count_head(head);
        printf("插入后单链表元素个数=% d\n", num);
        break;
    case 7: printf("\n 建立单链表\n\n");
        head=NULL;
        head=creatlink_order_head(head);
        fflush(stdin);
        num=count_head(head);
        printf("\n 删除重复元素后\n\n");
        delete(head);
        num=count_head(head);
        break;
    case 8: printf("\n 建立单链表\n\n");
        head=NULL;
        head=creatlink_head_rail(head);
        fflush(stdin);
        num=count_head(head);
        printf("\n 删除重复元素后\n\n");
        delete1(head);
        num=count_head(head);
        break;
    case 9: printf("\n 建立单链表 a1\n\n");
        a1=NULL;
        a1=creatlink_order_head(a1);
        fflush(stdin);
        num=count_head(a1);
        printf("单链表 a1 元素个数=% d\n", num);
        printf("\n 建立单链表 a2\n\n");
        a2=NULL;
        a2=creatlink_order_head(a2);
        fflush(stdin);
```

```
                num=count_head(a2);
                printf("单链表 a2 元素个数=% d\n", num);
                c=NULL;
                c=(LINKLIST * )malloc(sizeof(LINKLIST));
                c->next=NULL;
                unite(a1, a2, c);
                num=count_head(c);
                printf("合并到单链表 c 中,元素个数=% d\n", num);
                break;
            case 10:printf("\n 建立单链表 a1 \n\n");
                a1=NULL;
                a1=creatlink_head_rail(a1);
                fflush(stdin);
                num=count_head(a1);
                printf("\n 建立单链表 a2 \n\n");
                a2=NULL;
                a2=creatlink_head_rail(a2);
                fflush(stdin);
                num=count_head(a2);
                printf("\n\n 两链表并集运算\n\n");
                unionn(a1,a2);
                num=count_head(a1);
        }
    printf("结束此练习吗? (0——结束 1——继续) : ");
    scanf("% d",&loop);
    printf("\n");
    }
}
```

3.2 实训练习题

【习题 15】 叙述线性表的两种存储结构各自的特点。

【习题 16】 已知 P 结点是某双向链表的中间结点,试从下列提供的语句中选出合适的语句序列。

(1) 在 P 结点后插入 S 结点: _____。

(2) 在 P 结点前插入 S 结点: _____。

(3) 删除 P 结点的直接后继结点: _____。

(4) 删除 P 结点的直接前驱结点: _____。

(5) 删除 P 结点: _____。

① P—>next=P—>next—>next;

② P—>prior=P—>prior—>prior;

③ P—>next=S;
④ P—>prior=S;
⑤ S—>next=P;
⑥ S—>prior=P;
⑦ S—>next=P—>next;
⑧ S—>prior=P—>prior;
⑨ P—>prior—>next=P—>next;
⑩ P—>prior—>next=P;
⑪ P—>next—>prior=P;
⑫ P—>next—>prior=S;
⑬ P—>prior—>next=S;
⑭ P—>next—>prior=P—>prior;
⑮ Q=P—>next;
⑯ Q=P—>prior;
⑰ free(P);
⑱ free(Q);

【习题 17】 单项选择题。

1. 在线性表的下列存储结构中,读取元素花费时间最少的是_____。
   A. 单链表　　B. 双链表　　C. 循环链表　　D. 顺序表
2. 在单链表中,若 * p 结点不是末尾结点,在其后插入 * s 结点的操作是_____。
   A. s—>next=p; p—>next=s;　　B. s—>next=p—>next; p—>next=s;
   C. s—>next=p—>next; p=s;　　D. p—>next=s; s—>next=p;
3. 在带头结点 * head 的单循环链表中,至少有一个结点的条件是_____。
   A. head—>next !=NULL　　B. head—>next !=head
   C. p==NULL　　D. p—>next==head

【习题 18】 判断以下叙述的正确性。

(1) 顺序存储方式的优点是存储率高,且插入元素和删除元素效率高。
(2) 线性表的链式存储方式优于顺序存储方式。
(3) 顺序存储结构属于静态结构,链式存储结构属于动态结构。
(4) 对于单链表,只有从头结点(或第一个元素结点)开始才能扫描表中全部结点。
(5) 对于单循环链表,从表中任一结点出发都能扫描表中全部结点。
(6) 双链表的特点是找结点的前驱结点很容易,找结点的后继结点不容易。

【习题 19】 对应书中图 3-11 所示的循环链表的结构上,按下列要求设计对应算法。

(1) 在表尾的最后元素后插入一个元素 x。
(2) 在表的第一个元素前插入元素 x。

【习题 20】 设计一个算法判定一个带头结点的单链表的元素值是否是递增的。

【习题 21】 设有两个线性表 A 和 B,都是单链表存储结构。同一个表中的元素各不相同,且递增有序。编写一程序,构成一新的线性表 C,C 为 A 和 B 的交集,且 C 中元素

也递增有序。

题目分析：按输入的数据建立两个集合 a1 和 a2，同一集合中无重复元素，单链表存放。对两集合进行交集运算 a1∩a2，结果放在 a1 表中。算法的思路同单链表并集。

【习题 22】 设有两个线性表 A 和 B，都是单链表存储结构。同一个表中的元素各不相同，且递增有序。编写一程序，构成一新的线性表 C，C 为 A 和 B 的差集，且 C 中元素也递增有序。

题目分析：按用户输入的数据建立两个集合 a1 和 a2，同一集合中无重复元素，单链表存放。对两集合进行差集运算 a1－a2，结果放在 a1 表中。算法的思路同单链表并集。

第4章 栈和队列

栈和队列是两种操作受到限制的特殊类型的线性表。

对栈中数据元素的操作要遵循元素后进先出的原则。栈分顺序存储和链式存储两种实现方法。对栈中数据元素的操作要特别注意栈空条件、栈满条件和栈顶指针操作的正确。

队列的特点是对数据元素的操作要遵循元素先进先出的原则。队列分顺序存储和链式存储两种实现方法。改进后的顺序存储队列又称循环队列。对队列中数据元素的操作要特别注意队空条件、队满条件和队头指针、队尾指针操作的正确。

4.1 习题解析

【习题 1】 简述栈和队列这两种数据结构的相同点和不同点，以及与线性表的相同点和不同点。

【解答】 栈和队列是两种特殊的线性结构。从数据的逻辑结构角度看它们是线性表，从操作角度看它们是操作受限制的线性表。

栈是限定在表的一端进行插入或删除操作的线性表。插入元素叫进栈，删除元素叫出栈。进栈和出栈元素都只能在栈顶一端进行，所以每次出栈的元素总是当前栈中栈顶所在的元素，它是最后进栈的元素，而最先进栈的元素要到最后才能出栈。栈又称为后进先出的线性表。

队列也是一种特殊的线性表。它所有的插入操作均限定在表的一端进行，而所有的删除操作则限定在表的另一端进行。允许删除操作的一端称为队头，允许插入操作的一端称为队尾。上述规定决定了先进队列的元素先出队列。就如平时排队买东西一样。因此队列又称作先进先出的线性表。

【习题 2】 如果进栈的元素序列为 A、B、C、D，则可能得到的出栈序列有多少？写出全部的可能序列。

【解答】 以 A 开头的出栈序列有 ABCD、ABDC、ACBD、ACDB、ADCB，以 B 开头的出栈序列有 BACD、BADC、BCAD、BCDA、BDCA，以 C 开头的出栈序列有 CBAD、CBDA、

CDBA，以 D 开头的出栈序列有 DCBA。

【习题 3】 如果栈的元素序列为 123456，则能否得到 435612 和 135426 的出栈序列？并说明为什么不能得到或如何得到。

【解答】 能得到 135426 的出栈序列，因为 1 进 1 出 2 进 3 进 3 出 4 进 5 进 5 出 4 出 2 出 6 进 6 出符合在栈上后进先出的操作规则。

不能得到 435612 的出栈序列，因为 1 进 2 进 3 进 4 进 4 出 3 出 5 进 5 出 6 进 6 出符合在栈上后进先出的操作规则，其后只能 2 出 1 出得到 435621，得不到 435612 的出栈序列。

【习题 4】 写出下列程序段的运行结果(队列中的元素类型是 char)。

```
main()
{
  SEQQUEUE a, *q;
  char x, y;

  q=&a;
  x='e'; y='c';
  initqueue(q);
  enqueue(q,'h'); enqueue(q,'r'); enqueue(q,y);
  x=dequeue(q);
  enqueue(q,x);
  x=dequeue(q);
  enqueue(q,'a');
  while(!empty(q))
    {  y=dequeue(q);
       printf("%c",y);}
  printf("%c\n",x);
}
```

【解答】 运行结果是：

```
r c a h
```

【习题 5】 假设以 I 和 O 分别表示入栈和出栈操作，栈的初态和终态均为空，入栈和出栈的操作序列可表示为仅由 I 和 O 组成的序列，下面所示的序列中哪些是正确的？

(1) IOIIOIOO　　(2) IOOIOIIO　　(3) IIIOIOIO　　(4) IIIOOIOO

【解答】 (1) IOIIOIOO 正确。

(2) IOOIOIIO 错误。

(3) IIIOIOIO 错误。

(4) IIIOOIOO 正确。

【习题 6】 对于向量结构的循环队列，写出计算队列中元素个数的公式。

【解答】 对于向量结构的循环队列，队尾指针 rear 有时大于队头尾指针 front，有时小于队头尾指针 front，所以求循环队列中元素个数的公式是(rear－front＋MAXSIZE) %

MAXSIZE。

【习题7】 单项选择题。

（1）已知一个栈的进栈序列是1,2,3,…,n,其输出序列是$p_1,p_2,\cdots,p_n$,若$p_1=n$,则p_i的值_____。

A. i　　B. $n-i$　　C. $n-i+1$　　D. 不确定

【解答】 若$p_1=n$,则输出序列一定是n, $n-1$, $n-2$, …, 3, 2, 1,则$p_2=n-1$,$p_3=n-2,\cdots,p_n=1$,推断出p_i的值是$n-i+1$,答案是C。

（2）设n个元素进栈的序列是1,2,3,…,n,其输出序列是$p_1,p_2,\cdots,p_n$,若$p_1=3$,则p_2的值_____。

A. 可能是2　　B. 一定是2　　C. 可能是1　　D. 一定是1

【解答】 若$p_1=3$,说明1,2,3,已进栈,3出栈了,后面可能出栈的可能是2,可能是4或4后面的元素,但一定不可能是1。所以答案是A。

（3）设n个元素进栈的序列是$p_1,p_2,\cdots,p_n$,其输出序列是1,2,3,…,n,若$p_3=3$,则p_1的值_____。

A. 可能是2　　B. 一定是2　　C. 不可能是1　　D. 一定是1

【解答】 若$p_3=3$,进栈序列是$p_1,p_2,3,\cdots,p_n$,分析输出序列可知只有以下两种情况：一是$p_1=1,p_2=2$,p_1进栈后出栈,p_2进栈后出栈,输出序列是1,2,3,二是$p_1=2,p_2=1$,p_1进栈p_2进栈后p_2出栈p_1出栈,输出序列也是1,2,3,所以p_1可能是2的答案A正确。

【习题8】 判断以下叙述的正确性。

（1）栈底元素是不能删除的元素。

（2）顺序栈中元素值的大小必须是有序的。

（3）栈是一种对进栈、出栈操作总次数做了限制的线性表。

（4）对顺序栈进行进栈、出栈操作,不涉及元素的前、后移动问题。

（5）空栈没有栈顶指针。

【解答】 （1）错误。栈底元素是可以删除的。

（2）错误。顺序栈是指用顺序存储结构实现的栈,栈中元素值不要求必须是有序的。

（3）错误。对进栈、出栈操作总次数没有限制。

（4）正确。

（5）错误。空栈时,栈顶指针指向0。

【习题9】 编写一个程序实现以下顺序栈上的各种基本操作,用一个主程序串成。

（1）栈的初始化。

（2）栈不满的情况下元素进栈。

（3）栈不空的情况下元素出栈。

（4）输出栈中元素。

（5）计算栈中元素个数。

【解答】

```
#include "stdio.h"
#include"datastru.h"
```

```
void initstack(SEQSTACK * s)              /* 初始化空栈 */
{  s->top=0; }

int empty(SEQSTACK * s)                   /* 判栈空 */
{  if(s->top==0) return 1;
   else  return 0; }

int push(SEQSTACK * s, DATATYPE1 x)       /* 进栈 */
{  if(s->top==MAXSIZE- 1)
   {  printf("Overflow\n");
      return 0; }
   else { s->top++;
          s->data[s->top]=x;
          return 1; }
}

DATATYPE1 pop(SEQSTACK * s)               /* 出栈 */
{  DATATYPE1 x;

   if(empty(s)) {  printf("Underflow\n");
                   x=NULL; }
   else {  x=(s->data)[s->top];
           s->top--; }
   return x;
}

int stacklen(SEQSTACK * s)                /* 计算栈中元素个数 */
{  return(s->top ); }
void output_stack(SEQSTACK * s)           /* 输出栈中元素 */
{
   printf("buttom:");
   for(i=1;i<=s->top ;i++)
       printf("% d:",s->data[i]);
   printf("top\n");
}

main()
{  SEQSTACK s;
   DATATYPE1 a;
   int com,i;

   initstack(&s);                         /* 初始化空栈 */
   do  { printf("\n1:进栈 2: 出栈 3: 输出栈中元素 4: 栈中元素个数 0:退出\n");
```

```
        printf("enter a number(0-4):");
        scanf("% d",&com);
        switch(com) { case 1: printf(" enter a element:");
                              scanf("% d",&a);
                              push(&s,a);
                              output_stack(&s);
                              break;
                      case 2: a=pop(&s);
                              printf("pop :% d \n" ,a);
                              break;
                      case 3: output_stack
                              break;
                      case 4: printf("栈中有% d个元素 \n",stacklen(&s));
                              break; }
              } while(com!=0);
}
```

【习题 10】 编写一个程序实现以下循环队列上的各种基本操作,用一个主程序串成。

(1) 循环队列的初始化。

(2) 队列不满的情况下元素入队列。

(3) 队列不空的情况下元素出队列。

(4) 输出队列中元素。

(5) 计算队列中元素个数。

【解答】

```
#include "stdio.h"
#include "datastru.h"

int empty(SEQQUEUE * q)                    /* 判队空 */
{  if(q->rear==q->front) return 1;
   else return 0; }

void initqueue(SEQQUEUE * q)               /* 初始化队列 */
{  q->rear=q->front=-1; }

int enqueue(SEQQUEUE * q,DATATYPE1 x)      /* 进队 */
{  int r;
   if(q->front==(q->rear+1) %  MAXSIZE)
   { printf(" Queue is full. \n "); r=0;}
   else {q->rear= (q->rear+1) %  MAXSIZE;
         (q->data)[q->rear]=x;
         r=1;}
   return r;
```

```
}

DATATYPE1 dequeue(SEQQUEUE * q)          /* 出队 */
{  DATATYPE1 v;
   if(empty(q))
       {printf(" Queue is empty. \n "); v=NULL;}
   else {q->front=(q->front+1) %  MAXSIZE;
       v=(q->data)[q->front]; }
   return v;
}

int current_size(SEQQUEUE * q)           /* 队列中元素的个数 */
{  int n;
   if(q->rear>=q->front) n=(q->rear)-(q->front);
   else n=MAXSIZE+(q->rear)-(q->front);
   return n;
}

void output_queue(SEQQUEUE * q)          /* 输出队列中元素 */
{  int i;
   if(empty(q)) printf(" Queue is empty. \n ");
   else {  printf("front|");
           for(i=(q->front+1)%  MAXSIZE ;i!=q->rear ;i=(i+1)%  MAXSIZE)
           printf(" % d ",q->data[i]);
           printf(" % d |rear\n",q->data[i]);}
}

main()
{  SEQQUEUE s;
   DATATYPE1 a;
   int com, i ;

   initqueue(&s);                        /* 初始化空队列 */
   do  { printf("\n1:进队 2:出队 3: 输出队中元素 4: 队中元素个数 0:退出\n");
         printf("enter a number(0-4):");
         scanf("% d",&com);
         switch(com) {case 1: printf(" enter a element:");
                              scanf("% d",&a);
                              enqueue(&s,a);
                              output_queue(&s);
                              break;
                      case 2: a=dequeue(&s);
                              printf("del :% d \n" ,a);
                              output_queue(&s);
```

```
                    break;
                case 3: output_queue(&s);
                    break;
                case 4: printf("队列中有% d个元素\n",current_size(&s));
                    break; }
    }while(com!=0);
}
```

【习题 11】 编一个程序,将输入的非负十进制整数转换为八进制数输出。在顺序栈结构上实现。

题目分析:将输入的非负十进制整数转换为八进制数输出,可借用栈结构来实现,也可以用递归算法实现。

结构说明:栈结构用顺序存储方式实现,结构设定如下:

```
#define DATATYPE1 int
#define MAXSIZE 100

typedef struct
{ DATATYPE1 data[MAXSIZE];
  int top;
}SEQSTACK;
```

【解答 1】

```
#include "datastru.h"
#include <stdio.h>

void initstack(SEQSTACK * s)
/* 顺序栈初始化 */
  { s->top=0; }

DATATYPE1 gettop(SEQSTACK * s)
/* 返回栈顶元素 */
  { DATATYPE1 x;
    if(s->top==0)
      {printf("栈空\n"); x=0;}
    else x=(s->data)[s->top];
    return x;
  }

int push(SEQSTACK * s, DATATYPE1 x)
/* 元素 x 入栈 */
{  if(s->top==MAXSIZE-1)
   { printf("栈满\n"); return 0;}
  else
```

```
    { s->top++; (s->data)[s->top]=x; return 1;}
}

DATATYPE1 pop(SEQSTACK * s)
/*返回栈顶元素并删除栈顶元素*/
  { DATATYPE1 x;

    if(s->top==0) { printf("栈空\n"); x=0;}
      else { x=(s->data)[s->top]; s->top--;}
    return x;
}

main()
{  SEQSTACK stack, * s;
   int n;

   s=&stack; initstack(s); n=0;
   printf("输入一非负整数(十进制):");
   scanf("%d",&n); push(s,'#');
   while(n!=0)
       {push(s, n % 8);            /* % 为求余数运算符, 余数入栈*/
       n=n / 8;}                   /* /为求整数商运算符,商不为零,继续运行*/
   printf("\n\n对应的八进制数为 : ");
   while(gettop(s) != '#') printf("%d",pop(s));
   printf("\n");
}
```

【解答 2】 用递归算法实现非负十进制整数转换为八进制数的程序如下。递归算法中用到的栈由系统提供。一般将递归算法改写成非递归算法要用到栈结构处理。

```
#include <stdio.h>

void d_to_or(int x)
/*非负十进制整数转换为八进制数的递归算法*/
{  if(x / 8 !=0) d_to_or(x / 8);
   printf("%d",x % 8);
}

main()
{ int x;

printf("输入一非负整数(十进制):");
scanf("%d",&x);
printf("\n\n对应的八进制数为 : ");
d_to_or(x);
```

```
printf("\n\n");
}
```

【习题 12】 在一个表达式中含有括号“(”和“)”,编一程序,判断表达式中的括号是否正确配对。

题目分析:输入一个只有左括号“(”和右括号“)”的序列,程序对括号配对的正确性检查并给出结果,配对检查的算法中用到栈结构。例如,括号序列“(()())”、“()()(())”为正确配对,括号序列“()())”、“()(()”“)()(”为不正确配对等。

注意:输入序列中只能出现左括号“(”和右括号“)”,序列的语法正确性由用户保证。

【解答】

```
#include "datastru.h"
#include <stdio.h>

void initstack(SEQSTACK * s)
/* 顺序栈初始化 */
  { s->top=0; }

int push(SEQSTACK * s, DATATYPE1 x)
/* 元素 x 入栈 */
  {  if(s->top==MAXSIZE-1) {printf("栈满\n"); return 0;}
     else {s->top++; (s->data)[s->top]=x; return 1;}
  }

DATATYPE1 pop(SEQSTACK * s)
  /* 返回栈顶元素并删除栈顶元素 */
  {DATATYPE1 x;

   if(s->top==0) {printf("栈空\n"); x=0;}
   else {x=(s->data)[s->top]; s->top--;}
   return x;
  }

DATATYPE1 gettop(SEQSTACK * s)
/* 返回栈顶元素 */
  {DATATYPE1 x;

  if(s->top==0) {printf("栈空\n"); x=0;}
  else x=(s->data)[s->top];
  return x;
  }

check(SEQSTACK * s)
{
```

```
    int bool;
    char ch;

    push(s,'#'); printf("\n请输入一串括号,回车键结束 :");
    ch=getchar(); bool=1;
    while(ch !='\n' && bool)
      { if(ch=='(') push(s,ch);          /*遇到左括号,左括号入栈*/
        if(ch==')')                      /*遇到右括号:*/
          if(gettop(s)=='#') bool= 0;    /*栈顶元素为# 号,配对错误*/
          else pop(s);                   /*栈顶元素为左括号,消括号*/
        ch=getchar();}                   /*继续处理*/
    if(gettop(s) !='# ') bool=0;
    if(bool) printf("\n括号配对正确\n");
    else printf("\n括号配对错误\n");
}

main()
{
    SEQSTACK st, *s;

    s=&st;
    initstack(s);
    check(s);
}
```

【习题 13】 编写一个链队列管理的程序。这是一个加深理解在链队列结构上元素插入和删除的程序。在建立的带头结点的空链队列上，如果输入奇数，则奇数入队列；如果输入偶数，则队列中的第一个元素出队列；如果输入 0，则退出程序。

题目分析：这是一个加深理解在链队列结构上元素插入和删除的程序。首先建立带头结点的空链队列；如果输入奇数，则奇数入队列；如果输入偶数，则队列中的第一个元素出队列；如果输入 0，则退出程序。程序中可反复输入奇数和偶数，直至输入 0 退出程序。

结构说明：队列用链表结构实现，结构如下：

```
#define DATATYPE1 int

typedef struct qnode
        { DATATYPE1 data;
          struct qnode *next;
  }LINKQLIST;

typedef struct
{ LINKQLIST *front, *rear;
}LINKQUEUE;
```

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

DATATYPE1 dellinkqueue(LINKQUEUE * q)
/* 删除队头元素并返回 */
{  LINKQLIST * p;
   DATATYPE1 v;

   if(q->front==q->rear) {printf("队列空\n"); v=0;}
   else { p=(q->front)->next;
          (q->front)->next=p->next;
          if(p->next==NULL) q->rear=q->front;
          v=p->data;
          free(p);}
   return v;
}

void enlinkqueue(LINKQUEUE * q, DATATYPE1 x)
/* 元素 x 入队列 */
{  (q->rear)->next=(LINKQLIST * )malloc(sizeof(LINKQLIST));
   q->rear=(q->rear)->next; (q->rear)->data=x;
   (q->rear)->next=NULL;
}

void initlinkqueue(LINKQUEUE * q)
/* 链队列初始化 */
{  q->front=(LINKQLIST * )malloc(sizeof(LINKQLIST));
   (q->front)->next=NULL; q->rear=q->front;
}

void outlinkqueue(LINKQUEUE * q)
/* 链队列元素依次显示 */
{  LINKQLIST * p;

   p=q->front; printf("队列元素显示 : ");
   while(p !=q->rear)
     {p=p->next; printf("% d ",p->data);}
   printf("\n");
}

main()
{
```

```
    LINKQUEUE lq, *p;
    int j;

    p=&lq;
    initlinkqueue(p);
    printf("输入一整数(奇数——入队列、偶数——删除队头元素、0——退出):");
    scanf("%d", &j);
    while(j !=0)                          /*输入 0——退出*/
      { if(j % 2==1) enlinkqueue(p,j);     /*输入奇数——入队列*/
        else j=dellinkqueue(p);            /*输入偶数——删除队头元素*/
        outlinkqueue(p);
        printf("\n输入一整数(奇数——入队列、偶数——删除队头元素、0——退出):");
        scanf("%d", &j);}                  /*继续输入*/
}
```

【习题 14】 编写一个程序，打印扬辉三角系数。将二项式$(a+b)^i$展开，其系数构成杨辉三角形，按行将展开式系数的前 n 行显示出来。杨辉三角形系数如下：

```
          1    1
        1    2    1
      1    3    3    1
    1    4    6    4    1
  1    5    10   10   5    1
1    6    15   20   15   6    1
```

题目要求：将二项式$(a+b)^i$展开，其系数构成杨辉三角形，按行将展开式系数的前 n 行显示出来。杨辉三角形系数如下：

```
          1    1
        1    2    1
      1    3    3    1
    1    4    6    4    1
  1    5    10   10   5    1
1    6    15   20   15   6    1
```

从三角形的结构可知，除第一行以外，在显示第 i 行时，可利用上一行(第 $i-1$ 行)的数据。在显示第 $i+1$ 行时，又可用到第 i 行的数据。若在每行的两边各加上一个 0，如图 4-1 所示。

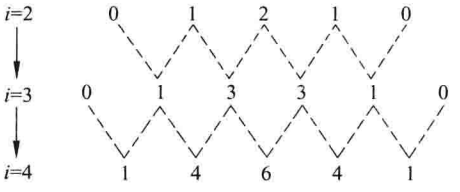


图 4-1 第 i 行元素与第 $i+1$ 行元素的关系

从图 4-1 中可以很清楚地看到下一行的系数可在上一行显示的同时计算得到。在程序中设置一个队列，在输出上一行系数时，同时计算

下一行的系数并预先存入队列中。在各行系数之间插入一个0,为计算方便用。

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

DATATYPE1 dellinkqueue(LINKQUEUE * q)
/*删除队头元素并返回*/
{  LINKQLIST * p;
   DATATYPE1 v;

   if(q->front==q->rear)
       {printf("队列空\n"); v=0;}
   else
       { p=(q->front)->next;
         (q->front)->next=p->next;
         if(p->next==NULL)
             q->rear=q->front;
         v=p->data;
         free(p);}
   return v;
}

DATATYPE1 getlinkqueue(LINKQUEUE * q)
{/*返回队头元素*/
    DATATYPE1 v;

    if(q->front==q->rear) {printf("队列空\n"); v=0;}
    else v=(q->front)->next->data;
    return v;
}

void enlinkqueue(LINKQUEUE * q, DATATYPE1 x)
{/*元素x入队列*/
    (q->rear)->next=(LINKQLIST * )malloc(sizeof(LINKQLIST));
    q->rear=(q->rear)->next;
    (q->rear)->data=x;
    (q->rear)->next=NULL;
}

void initlinkqueue(LINKQUEUE * q)
{/*链队列初始化*/
    q->front=(LINKQLIST * )malloc(sizeof(LINKQLIST));
```

```
    (q->front)->next=NULL;
    q->rear= q->front;
}

void main(){
    LINKQUEUE lq, *q;
    int n, s, t, k, i, m, j;

    printf("打印扬辉三角形,输入行数 : ");
    scanf("%d",&n); q=&lq;
    initlinkqueue(q);                          /*链队列初始化*/
    enlinkqueue(q, 1); enlinkqueue(q, 1);      /*预先放入第一行的两个系数 1*/
    s=0;                                       /*计算下一行系数时用到的工作单元*/
    for(i=1; i<=n; i++) {                      /*逐行处理*/
        printf("\n");                          /*换行*/
        for(m=i; m<=n; m++) printf(" ");
        enlinkqueue(q, 0);                     /*各行之间插入一个 0*/
        for(j= 1; j<=i+2; j++) {               /*处理第 i 行的 i+2 个系数(包括一个 0)*/
            t=getlinkqueue(q);                 /*取队头元素*/
            k=dellinkqueue(q);                 /*删除队头元素*/
            enlinkqueue(q, s+t);               /*计算下一行的系数,入队列*/
            s=t;
            if(j != (i+2)) printf("%d ",s);    /*打印一个系数,第 i+2 个是 0,不显示*/
            }
        printf("\n");
}
printf("\n\n");
}
```

4.2 实训练习题

【习题 15】 有 5 个元素,其入栈的次序为 A,B,C,D,E,在各种可能的出栈的次序中,以元素 C,D 最先出栈(即 C 为第一个出栈并且 D 为第二个出栈)的次序有哪几个?以元素 B,D 最先出栈(即 B 为第一个出栈并且 D 为第二个出栈)的次序有哪几个?

【习题 16】 写出下列程序段的运行结果(栈中的元素类型是 char)。

```
main()
{
  SEQSTACK s, *p;
  char x, y;

  p=&s;
  initstack(p);
```

```
    x='c'; y='k';
    push(p,x); push(p,'a'); push(p,y);
    x=pop(p);
    push(p,'t'); push(p,x);
    x=pop(p);
    push(p,'s');
    while(!empty(p))
        {  y=pop(p);
           printf("% c",y);}
    printf("% c\n",x);
}
```

【习题 17】 判断以下叙述的正确性。

(1) 在 n 个元素进栈后,它们的出栈顺序和进栈顺序一定正好相反。

(2) 栈顶元素和栈底元素不可能是同一元素。

(3) 若用 $s[1]\sim s[m]$ 表示顺序栈的存储空间,则对栈的进栈、出栈操作只能进行 m 次。

(4) n 个元素进队列的顺序和出队列的顺序总是一致的。

(5) 无论是顺序队列还是链接队列,插入和删除元素运算的时间复杂度都是 O(1)。

(6) 栈和队列都是限制存取端的线性表。

【习题 18】 写一算法依次打印一顺序栈中的元素值。

【习题 19】 写一算法依次打印一链队列中的元素值。

【习题 20】 设单链表中存放着 n 个字符,写一算法,判断该字符串是否有中心对称关系,例如,xyzzyx、xyzyx 都算是中心对称的字符串。

【习题 21】 编一程序,将输入的非负十进制整数转换为二进制数输出。在顺序栈结构上实现。

题目分析:将输入的非负十进制整数转换为二进制数输出显示。一个程序用栈结构实现;一个程序用递归算法实现。算法的思路可参考习题 11。

【习题 22】 编一程序,将输入的非负十进制整数逆向显示,如输入 1234,输出显示 4321。

题目分析:将输入的非负十进制整数逆向显示,如输入 1234,输出显示 4321。用递归算法实现。

第5章

其他线性数据结构

本章主要介绍串和多维数组，多维数组中重点是二维数组。

5.1 习题解析

【习题 1】 设

```
s="I AM A STUDENT",
t="GOOD",
q="WORKER",
```

求：STRLEN(s)，STRLEN(t)，SUBSTR(s,8,7)，SUBSTR(t,2,1)，INDEX(s,"A")，INDEX(s,t)，REPLACT(s,"STUDENT",q)，CONCAT(SUBSTR(s,6,2),CONCAT(t,SUBSTR(s,7,8)))。

【解答】

```
STRLEN(s)=14
STRLEN(t)=4
SUBSTR(s,8,7)="STUDENT"
SUBSTR(t,2,1)="O"
INDEX(s,"A")=3
INDEX(s,t)=0
REPLACT(s,"STUDENT",q)="I AM A WORKER"

SUBSTR(s,7,8)=" STUDENT"
CONCAT(t,SUBSTR(s,7,8))="GOOD STUDENT"
SUBSTR(s,6,2)="A"
CONCAT(SUBSTR(s,6,2),CONCAT(t,SUBSTR(s,7,8)))="A GOOD STUDENT"
```

【习题 2】 已知下列字符串：

```
a="THIS", f="A SAMPLE",c="GOOD", d="NE", b=" ",g="IS",
```

```
s=CONCAT(a,CONCAT(CONCAT(b, SUBSTR(a,3,2)),SUBSTR(f,2,7))),
t=REPLACE(f,SUBSTR(f,3,6),c),
u=CONCAT(SUBSTR(c,3,1),d),
v=CONCAT(s,CONCAT(b,CONCAT(t,CONCAT(b,u)))).
```

问：s,t,v,STRLEN(s),INDEX(v,g),INDEX(u,g) 各是什么?

【解答】

```
SUBSTR(f,2,7)=" SAMPLE"
CONCAT(b, SUBSTR(a,3,2))=" IS"
CONCAT(CONCAT(b, SUBSTR(a,3,2)),SUBSTR(f,2,7))=" IS SAMPLE"
s=CONCAT(a,CONCAT(CONCAT(b, SUBSTR(a,3,2)),SUBSTR(f,2,7)))="THIS IS SAMPLE"

t="A GOOD"
u=CONCAT(SUBSTR(c,3,1),d)="ONE"
CONCAT(t,CONCAT(b,u))="A GOOD ONE"
v=CONCAT(s,CONCAT(b,CONCAT(t,CONCAT(b,u))))="THIS IS SAMPLE A GOOD ONE"

STRLEN(s)=14
INDEX(v,g)=3
INDEX(u,g)=0
```

【习题 3】 已知：s="$(XYZ)+*$", t="$(X+Z)*Y$"。试利用联接、求子串等基本运算，将 s 转化为 t。

【解答】

```
t=CONCAT(SUBSTR(s,1,2), SUBSTR(s,6,1))
t=CONCAT(t, SUBSTR(s,4,2))
t=CONCAT(t, SUBSTR(s,7,1))
t=CONCAT(t, SUBSTR(s,3,1))
```

【习题 4】 简述下列每对术语的区别：

空串和空格串；串变量和串常量；主串和子串。

【解答】

空串和空格串：长度为 0 的串称为空串。空串是任意串的子串。

由一个或多个空格组成的串称为空格串，空格串的长度是不会为 0 的。

串变量和串常量：串是由零个或多个字符组成的有限序列。一般记为 $S='a_1, a_2, \cdots, a_n'$ $(n\geqslant 0)$其中，S 为串名，即串变量。用单引号括起来的字符序列是串变量的值，$a_i(1\leqslant i\leqslant n)$可以是字母、数字或其他字符。当串变量中的值确定时，引用此串时，此串为串常量，如串常量 C="data structure"。

主串和子串：串中任意一个连续的字符组成的子序列称为该串的子串。包含子串的串相应地称为主串。

【习题 5】 单项选择题。

(1) 数组 A[0..5, 0..6]的每个元素占 5 个单元，将其按列优先次序存储在起始地址

为1000的连续内存单元中，则元素a[5][5]的地址_____。

A. 1175　　B. 1180　　C. 1205　　D. 1210

【解答】 因按列优先次序存储元素，第a[5][5]前有5列＊6个元素/每列＋5个/第6列上＝35个元素，每个元素占5个单元，则元素a[5][5]的地址是1000＋35＊5＝1175，正确答案为A。

(2) 一个$n*n$的对称矩阵，如果以行或列为主序放入内存，则容量为_____。

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

【解答】 对称矩阵中的每一对对称元素，可以只分配一个元素的存储空间，从而将n^2个元素的存储空间压缩到$n(n+1)/2$个元素的存储空间。正确答案为C。

(3) 对矩阵的压缩存储是为了_____。

A. 方便运算　　B. 节省空间　　C. 方便存储　　D. 提高运算速度

【解答】 对矩阵的压缩存储是为了节省空间，正确答案为B。

(4) 串是一种特殊的线性表，其特殊性体现在_____。

A. 可以顺序存储　　B. 数据元素是一个字符

C. 可以链接存储　　D. 数据元素可以是多个字符

【解答】 串是一种特殊的线性表，它的特殊性在于：串中的每一个数据元素仅由一个字符组成。正确答案为B。

(5) 串的长度是_____。

A. 串中不同字母的个数　　B. 串中不同字符的个数

C. 串中所含字符的个数且大于0　　D. 串中所含字符的个数

【解答】 串的长度是串中所含字符的个数，正确答案为D。

【习题6】 判断以下叙述的正确性。

(1) 用一维数组表示矩阵，可以简化对矩阵的存取操作。

(2) 对称矩阵的特点是非零元素只出现在矩阵的两条对角线上。

(3) 稀疏矩阵的特点是矩阵中元素较少。

(4) 在表示稀疏矩阵的三元组顺序表中，各元素的排列顺序与矩阵元素值的大小有关。

【解答】

(1) 错误。一维数组只是存储矩阵的一种方法。

(2) 错误。n阶方阵A中元素满足下列条件：$a_{ij}==a_{ji}(i,j=0,1,\cdots,n-1)$的矩阵为对称矩阵。

(3) 错误。稀疏矩阵的特点是矩阵中非零元素较少。

(4) 错误。

【习题7】 设有二维数组A_{6*8}，每个元素占6个字节存储，顺序存放，A的起地址为1000，计算：

(1) 数组A的体积(即存储量)；

(2) 数组的最后一个元素$a_{5,7}$的起地址；

(3) 按行优先存放时,元素 $a_{1,4}$ 的起地址;

(4) 按列优先存放时,元素 $a_{4,7}$ 的起地址。

【解答】

(1) 二维数组 A_{6*8},即 6 行 8 列,下标为(0..5,0..7),数组 A 的体积(即存储量)=6 * 8 * 6=288 个字节。

(2) 数组的最后一个元素 $a_{5,7}$ 的起地址=(6 * 8−1) * 6+1000=1282。

(3) 因按行优先存放,元素 $a_{1,4}$ 前有 1 * 8+4=12 个元素,$a_{1,4}$ 的起地址=12 * 6+1000=1072。

(4) 因按列优先存放,元素 $a_{4,7}$ 前有 7 * 6+4=46 个元素,$a_{4,7}$ 的起地址=46 * 6+1000=1276。

【习题 8】 已知稀疏矩阵 A 如下所示,画出它的三元组表的示意图。

$$A=\begin{bmatrix}0 & 1 & 0 & 0 & -1\\ 4 & 0 & 0 & 0 & 0\\ 0 & 0 & 0 & 7 & 0\\ 0 & 6 & 0 & 0 & 0\\ 8 & 0 & 9 & 0 & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix}$$

【解答】 对应的三元组表 a 如图 5-1 所示。

	i	j	v	
a.data[1]	1	2	1	
[2]	1	5	−1	a.m=b(行)
[3]	2	1	4	a.n=5(列)
[4]	3	4	7	a.t=7(个非零元素)
[5]	4	2	6	
[6]	5	1	8	
[7]	5	3	9	

图 5-1 三元组表 a

【习题 9】 已知一矩阵 $A(m * n)$,按行优先原则存放在一维数组 B 中,下标从 1 开始。写一算法,将矩阵转置并按行优先原则存放在一维数组 C 中,下标从 1 开始。

【解答 1】

```
void rmat1(int * B ,int * C, int m,int n)
{  int k,i,j;
   k=1;
   for ( i=1; i<=n; i++)
        for ( j=0; j<m; j++)
        {  C[k]=B[ i+j * n];
           k++;}
```

```
}
```

【解答 2】

```
void rmat2(int * B ,int * C, int m,int n)
{  int i,j;
   for (i=1; i<=m; i++)
      for (j=1; j<=n; j++)
          C [( j-1) * m+i ]=B [(i-1) * n+j ];
}
```

【习题 10】 编写一程序，在串的顺序定长存储结构上实现子串的定位操作。

题目分析：子串的定位操作又称串的模式匹配操作。是各种串处理中最重要的操作之一。输入主串 S 和子串 T，若在主串 S 中存在和 T 相等的子串，则返回在 S 中出现的第一个和 T 相等的子串在 S 中的位置，否则返回 0。注意 T 不能是空串。

结构说明：在串的顺序定长存储结构上实现。结构如下：

```
#define MAXSIZE 100

typedef struct
{  char ch[MAXSIZE];      //字串最大长度限定为 100
   int len;
}SEQSTRING;
```

【解答】

```
#include "datastru.h"
#include <stdio.h>

int index(SEQSTRING s,SEQSTRING t)
{/* 模式匹配 */
int i=1, j=1;
while(i<=s.len && j<=t.len)
    if(s.ch[i]==t.ch[j]) {i++; j++;}
    else {i=i-j+2; j=1;}
if(j>t.len) return i-t.len;
else return 0;
}

main() {
SEQSTRING s, t;
int i;
char ch;

printf("\n 请输入主串,回车键结束:");
ch=getchar(); i=1;
```

```
while(ch != '\n')
   {s.ch[i]=ch; i++;                    /* 主串从 s.ch 的下标 1 开始放起 */
    ch=getchar();}
s.len=i-1;                              /* 主串长度放在 s.len 中 */
printf("\n 请输入子串,回车键结束:");
ch=getchar(); i=1;
while(ch != '\n')
   { t.ch[i]=ch; i++;                   /* 主串从 t.ch 的下标 1 开始放起 */
     ch=getchar();}
t.len=i-1;                              /* 子串长度放在 t.len 中 */
i=index(s,t);                           /* 匹配操作算法 */
if(i==0) printf("\n 子串不在主串中!\n\n");
else printf("\n 子串在主串中! 从位置% d开始.\n\n",i);
}
```

【习题 11】 编写一程序,将一个三元组存储的稀疏矩阵转置。转置后稀疏矩阵仍为三元组存储结构。

题目分析:对一个稀疏矩阵,按提示输入其行号、列号及每一个元素值,程序建立稀疏矩阵的三元组存储结构。将三元组存储结构的稀疏矩阵转置。转置后稀疏矩阵仍为三元组存储结构。程序显示转置前后稀疏矩阵的三元组结构。

结构说明:三元组存储结构的稀疏矩阵结构说明如下:

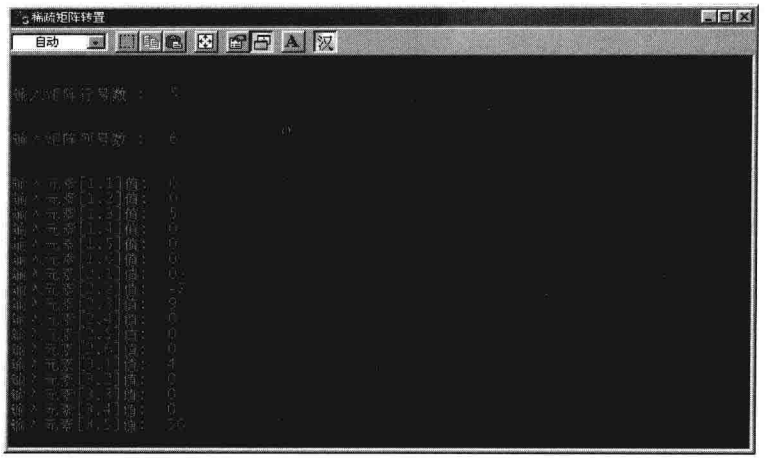
```
#define MAXLEN 40
#define DATATYPE1 int

typedef struct
{  int i, j;                  /* 一个三元组单元中放入非零元素的行号、列号 */
   DATATYPE1 v;               /* 及该非零元素的值 */
}NODE;

typedef struct
{  int m, n, t;               /* 稀疏矩阵的行号、列号、非零元素个数 */
   NODE data[MAXLEN];         /* 三元组向量 */
}SPMATRIX;
```

稀疏矩阵中非零元素值设定为整型量,非零元素个数设定最多 40 个。如图 5-2(a)~(d)所示是以一个 5 行 6 列 7 个非零元素的稀疏矩阵为例,显示程序输入矩阵数据及三元组结构的稀疏矩阵转置的过程。

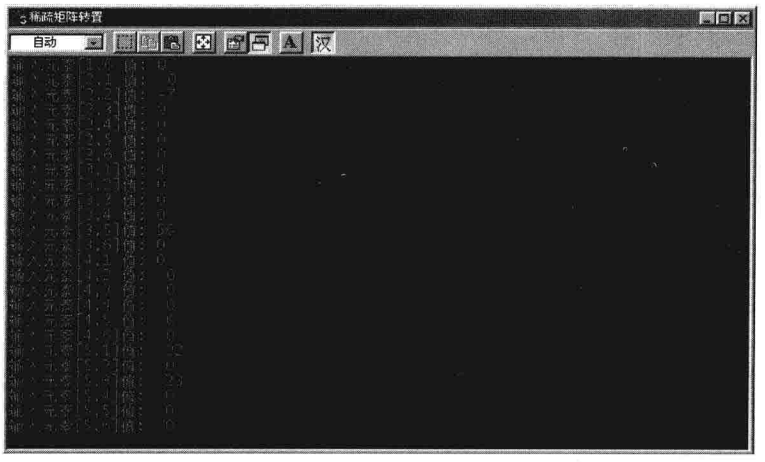
$$\begin{pmatrix} 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & -7 & 9 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 56 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 22 & 0 & 23 & 0 & 0 & 0 \end{pmatrix}$$

(a)

图 5-2 稀疏矩阵转置的过程



(b)



(c)

(d)

图 5-2（续）

【解答】

```
#include "datastru.h"
#include<malloc.h>
#include<stdio.h>

SPMATRIX transpose(SPMATRIX a)
{/*稀疏矩阵(三元组存储结构)转置算法*/
   int p, q, col;
   SPMATRIX b;

   b.m=a.n; b.n=a.m; b.t=a.t;
   if(a.t!=0)
      {q=1;
      for(col=1;col<=a.n;col++) /* a稀疏矩阵的列数即为转置后稀疏矩阵的行数*/
          for(p=1;p<=a.t;p++)
            if(a.data[p].j==col)
              {b.data[q].j=a.data[p].i;
              b.data[q].i=a.data[p].j;
              b.data[q].v=a.data[p].v;
              q++;}}
    return b;
}

void printmatrix(SPMATRIX c)
{/*稀疏矩阵(三元组存储结构)显示*/
 int n,i;

 n=c.t;
 for(i=1;i<=n;i++)
 printf("[%d]行号=%d列号=%d元素值=%d\n",i,c.data[i].i,c.data[i].j,c.data[i].
 v);
}

main()
{  SPMATRIX a;
   SPMATRIX b;
   int i,j,r,c,t,n;

   n=1;
   printf("\n\n输入矩阵行号数 : ");
   scanf("%d",&r);
   printf("\n\n输入矩阵列号数 : ");
   scanf("%d",&c);
   a.m=r; a.n=c;
```

```
    printf("\n\n");
    for(i=0;i<r;i++)                    /* 输入矩阵元素值 */
      for(j=0;j<c;j++)
        {printf("输入元素[% d,% d]值: ",i+1,j+1);
        scanf("% d",&t);
        if(t!=0) {a.data[n].i=i+1; /* 非零元素存入稀疏矩阵三元组存储结构中 */
          a.data[n].j=j+1; a.data[n].v=t; n=n+1;}
        }
    n=n-1; a.t=n;                       /* a.t 中为稀疏矩阵非零元素个数 */
    printf("\n\n稀疏矩阵三元组表示 : \n\n");
    printmatrix(a);                     /* 稀疏矩阵(三元组存储结构)转置 */
    b=transpose(a);
    printf("\n\n转置后稀疏矩阵三元组表示 : \n\n");
    printmatrix(b);
    printf("\n\n");
}
```

5.2 实训练习题

【习题 12】 有两个串 s1 和 s2，设计一算法，求一个这样的串，该串中的字符是 s1 和 s2 中的公共字符。

【习题 13】 编写程序，判断两顺序串是否相等。

题目分析：判断两顺序串是否相等的含义如下——两顺序串的长度必须相等；两顺序串各对应位置上的字符对应相等。

【习题 14】 两个具有相同行号数和列号数的稀疏矩阵采用三元组表存储结构，三元组表分别为 A 和 B，编写程序，实现两矩阵相加，结果矩阵存放在三元组表 C 中。

题目分析：矩阵相加必须是在两个具有相同行号和列号的矩阵上进行。对稀疏矩阵而言，可用三元组表存放稀疏矩阵。两稀疏矩阵相加的结果存放在另一新三元组表内。

算法思路提示：

(1) 稀疏矩阵用三元组表存放(以行优先存放)具有以下的特点：元组中的个数就是非零元素的个数；元组中的第一列按行号的顺序由小到大排列；元组中的第二列是列号，列号在行号相同时也是由小到大排列。

(2) 相加的两矩阵和相加结果存放的矩阵均以三元组表结构存放，均具有上述的特点。

(3) 矩阵相加：两矩阵同行同列的对应元素相加，结果不为零的元素以一个三元组元素放入新三元组表中。

(4) 以三个指针 i，j，k 分别指向两相加矩阵对应的三元组表 A，B 及结果存放的新三元组表 C。

树和二叉树

本章的重点是二叉树。

通过本章的习题可以在机器上运行程序验证二叉树的各种遍历算法和在二叉树上的各种操作，包括二叉树左右子树交换、求二叉树叶子结点个数、求二叉树的深度、求某结点的双亲结点等。

二叉树的二叉链表存储结构如下：

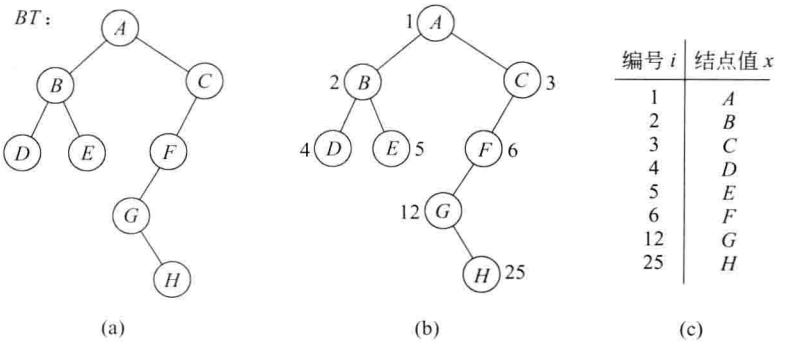
```
#define DATATYPE2 char

typedef struct node1
{ DATATYPE2 data;
 struct node1 * lchild, * rchild;
}BTCHINALR;
```

要实现在二叉树上的各种操作，一般将二叉树以二叉链表结构在内存中建立起来，而后在其上面进行各种操作。下面提供一个可实现的二叉树建立的函数，供各应用程序调用，函数源程序以“二叉树.c”为名放在盘上。“二叉树.c”源代码如下：

```
BTCHINALR * createbt( )
{   BTCHINALR * q;
    struct node1 * s[50];
    int j,i,x;

    printf("建立二叉树，输入结点对应的编号和值\n\n");
    printf("i,x=");
    scanf("% d,% c",&i,&x);
    while(i !=0 && x !='$ ')
      {q= (BTCHINALR * )malloc(sizeof(BTCHINALR));        /* 建立一个新结点 q* /
       q->data=x; q->lchild=NULL; q->rchild=NULL;
       s[i]=q;                    /* q新结点地址存入 s指针数组中 * /
       if(i !=1)                  /* i=1,q结点是根结点 * /
         {j=i / 2;                /* 否则求 q结点的双亲结点的编号 j* /
```

```
            if(i % 2==0) s[j]->lchild=q;  /*q结点编号为偶数则挂在双亲结点j的左边*/
            else s[j]->rchild=q;}   /*q结点编号为奇数则挂在双亲结点j的右边*/
        printf("i,x=");
        scanf("%d,%c",&i,&x);}
    return s[1];                    /*返回根结点地址*/
}
```

下面举例说明二叉树的建立。设有一棵二叉树如图 6-1(a)所示，将二叉树模拟为完全二叉树从根开始对结点进行编号，编号从 1 开始，结果如图 6-1(b)所示。在运行过程中要求输入结点对应的编号和值时，请按图 6-1(c)中的数据输入，最后以编号 $i=0$；结点值 $x=$'$'结束。



编号 i	结点值 x
1	A
2	B
3	C
4	D
5	E
6	F
12	G
25	H

图 6-1　二叉树

(a) 二叉树 BT；(b) 二叉树 BT 的结点编号；(c) 二叉树 BT 的序列表

6.1　习题解析

【习题 1】 写出图 6-2 中树的叶子结点、非终端结点、各结点的度和树的深度。

【解答】

叶子结点：F，G，L，C，I，M，K。

非终端结点：A，B，C，D，E，J。

各结点的度：A 结点度$=4$，B 结点度$=3$，C 结点度$=0$，D 结点度$=1$，E 结点度$=2$，F 结点度$=0$，G 结点度$=0$，L 结点度$=0$，I 结点度$=0$，J 结点度$=1$，K 结点度$=0$，M 结点度$=0$。

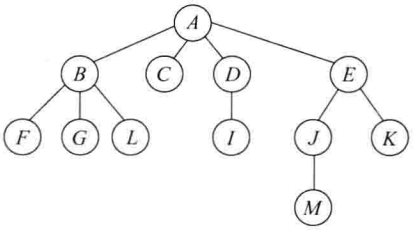


图 6-2　树

树的深度：4。

【习题 2】 分别画出含 3 个结点的无序树与含 3 个结点的二叉树的所有不同形态。

【解答】

含 3 个结点的无序树的所有形态如图 6-3 所示。

含 3 个结点的二叉树的所有形态如图 6-4 所示。

【习题 3】 分别画出图 6-5 中所示二叉树的二叉链表、三叉链表和顺序存储结构示意图。

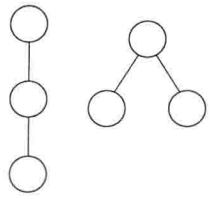


图 6-3　含 3 个结点的无序树的所有形态

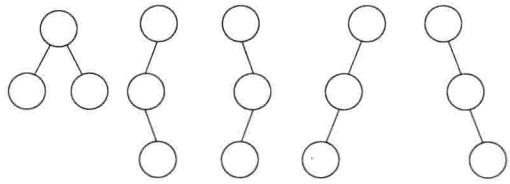


图 6-4　含 3 个结点的二叉树的所有形态

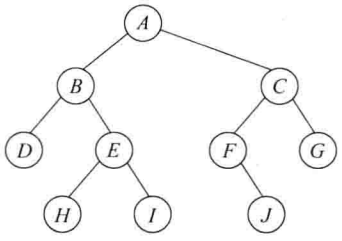


图 6-5　二叉树

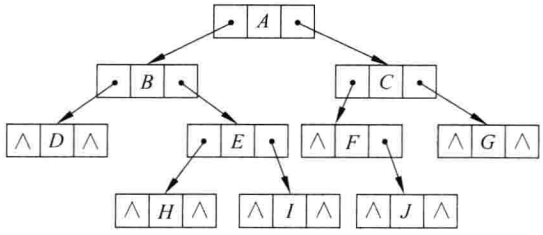


图 6-6　二叉链表存储结构

【解答】

二叉链表存储结构示意图如图 6-6 所示。

三叉链表存储结构示意图如图 6-7 所示。

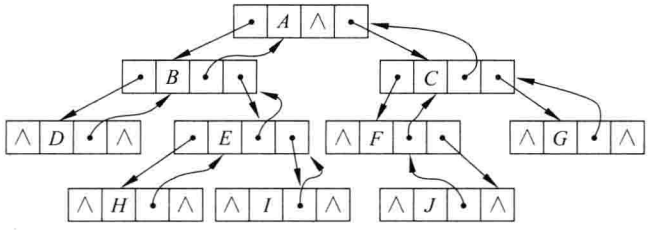


图 6-7　三叉链表存储结构

顺序存储结构示意图如图 6-8 所示。

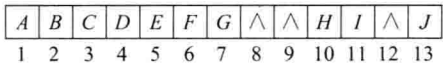


图 6-8　顺序存储结构

【习题 4】 分别写出图 6-5 中所示二叉树的先根遍历、中根遍历、后根遍历的结点访问序列。

【解答】

先根遍历访问序列：*ABDEHICFJG*。

中根遍历访问序列：*DBHEIAFJCG*。

后根遍历访问序列：*DHIEBJFGCA*。

【习题 5】 已知二叉树有 50 个叶子结点，该二叉树的总结点数至少有多少？

【解答】 已知 $n_0=50$，根据二叉树的特性得 $n_0=n_2+1$，$n_2=n_0-1=49$，二叉树的总结点数 $n=n_1+n_2+n_0=n_1+50+40=n_1+99$，如果度为 1 的结点数 n_1 为 0，则二叉树的总结点数至少有 99 个。

【习题 6】 已知完全二叉树的第 8 层有 8 个叶子结点，则完全二叉树的叶子结点数是多少？

【解答】 第 8 层有 8 个结点的完全二叉树共有 8 层，上面 7 层每一层结点都是满的，第 7 层的结点数是 $2^{i-1}=2^{7-1}=64$，其中 4 个结点有 8 个叶子结点，则完全二叉树的叶子结点数 $=64-4+8=68$ 个。

【习题 7】 一棵二叉树的先序序列、中序序列、后序序列分别如下，其中有一部分未显示出来，求出空格处的内容，并画出该二叉树。

先序序列：$-B-F-ICEH-G$。

中序序列：$D-KFIA-EJC-$。

后序序列：$-K-FBHJ-G-A$。

【解答】 由后序序列显示的部分可得知二叉树的根是 A，进一步分析得知：A 的左子树部分对应的中序序列为 $D-KFI$，5 个结点。对应的先序序列为 $B-F-I$，对应的后序序列为 $-K-FB$。A 的右子树部分对应的中序序列为 $-EJC-$，5 个结点，对应的先序序列为 $CEH-G$，对应的后序序列为 $HJ-G-$。再用递归的思路分析左右子树可推出二叉树如图 6-9 所示。

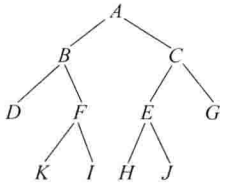


图 6-9 二叉树

先序序列：$\underline{A}B\underline{D}F\underline{K}ICEHJG$。

中序序列：$D\underline{B}KFIA\underline{H}EJC\underline{G}$。

后序序列：$\underline{D}K\underline{I}FBHJ\underline{E}G\underline{C}A$。

【习题 8】 如果一棵度为 m 的树有 n_1 个度为 1 的结点，n_2 个度为 2 的结点，……，n_m 个度为 m 的结点，则该树共有多少个叶子结点？

【解答】 设树共有叶子结点 n_0，该树总的结点数 $n=n_0+n_1+n_2+\cdots+n_m$，从该树分支的角度分析得 $n-1=0*n_0+1*n_1+2*n_2+\cdots+m*n_m$，即 $n_0=1*n_2+2*n_3+\cdots+(m-1)*n_m+1$。

$$n_0=\sum_{i=1}^{m}(i-1)_{n_i}+1$$

【习题 9】 已知在一棵含有 n 个结点的树中，只有度为 k 的分支结点和度为 0 的叶子结点，试求该树含有的叶子结点的数目。

【解答】 设此树度为 0 的叶子结点 n_0 个，度为 k 的分支结点 n_k 个。该树总的结点数 $n=n_0+n_k$；从该树分支的角度分析得 $n-1=k*n_k$，计算得 $n_0=n-(n-1)/k$ 个。

【习题 10】 将图 6-10 中所示的森林转换为二叉树。

【解答】 森林由三棵树组成，根据森林转换成二叉树的规则，第一棵树的根是二叉树的根 A，第一棵树根的全部子树组成二叉树的左子树部分，其余的二棵树组成二叉树的右子树部分。递归下去，组成的二叉树如图 6-11 所示。

【习题 11】 分别画出图 6-12 中所示二叉树对应的森林。

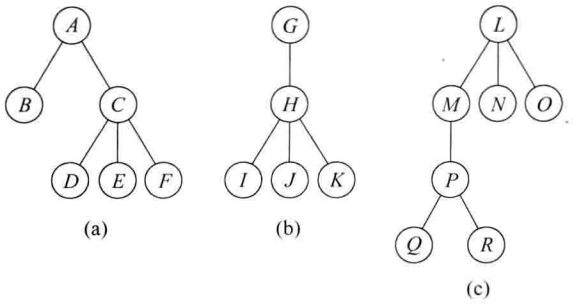


图 6-10　森林

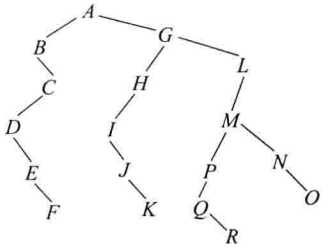


图 6-11　森林转换成的二叉树

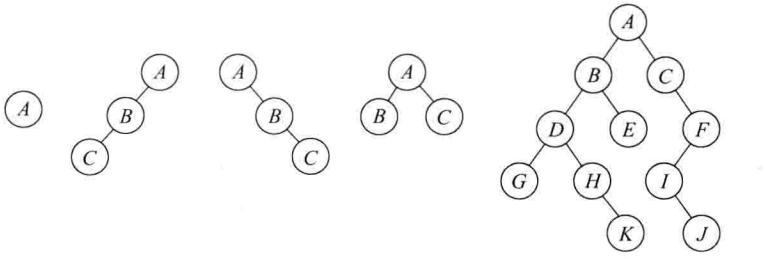


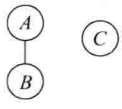
图 6-12　二叉树

【解答】 图 6-12 的分图所对应的森林如下。

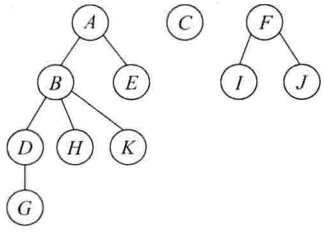
(a) 对应的森林

(b) 对应的森林

(c) 对应的森林



(d) 对应的森林



(e) 对应的森林

【习题 12】 图 6-14 给出了图 6-13 树的带双亲域的孩子链表结构示意图，写出其存储结构数据类型说明。

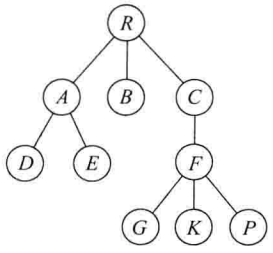


图 6-13 树

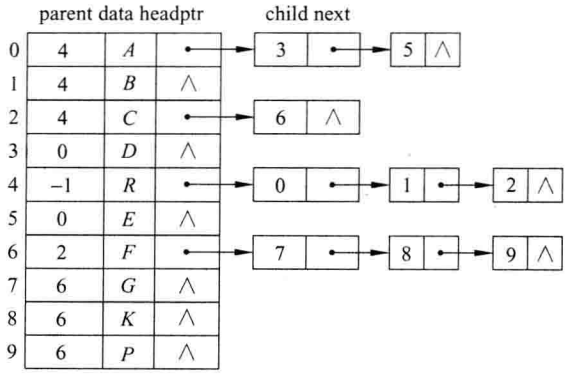


图 6-14 带双亲域的孩子链表

【解答】

树的孩子链表表示法存储结构类型说明如下：

```
typedef struct cnode
{ int child;
struct cnode * next;
}CHILDLINK;

typedef struct
{  DATATYPE2 data;
   CHILDLINK * headptr;
}CTNODE;

typedef struct
{CTNODE nodes[MAXSIZE];
int nodenum, rootset;
}CTTREE;
```

只要在结点 CTNODE 中加入双亲域信息即可，树的带双亲域的孩子链表表示法存储结构类型说明如下：

```
typedef struct cnode
{ int child;
struct cnode * next;
}CHILDLINK;

typedef struct
{  DATATYPE1 parent; /* 树中结点的双亲结点信息 */
   DATATYPE2 data;
   CHILDLINK * headptr;
}CTNODE;
```

```
typedef struct
{CTNODE nodes[MAXSIZE];
int nodenum, rootset;
}CT_PT_TREE;
```

【习题 13】 给定权值 $w=(7, 18, 3, 32, 5, 26, 12, 8)$，构造一棵关于 w 的相应的哈夫曼树，并求其路径长度 WPL。

【解答】

构造关于 w 的哈夫曼树过程如图 6-15 所示。

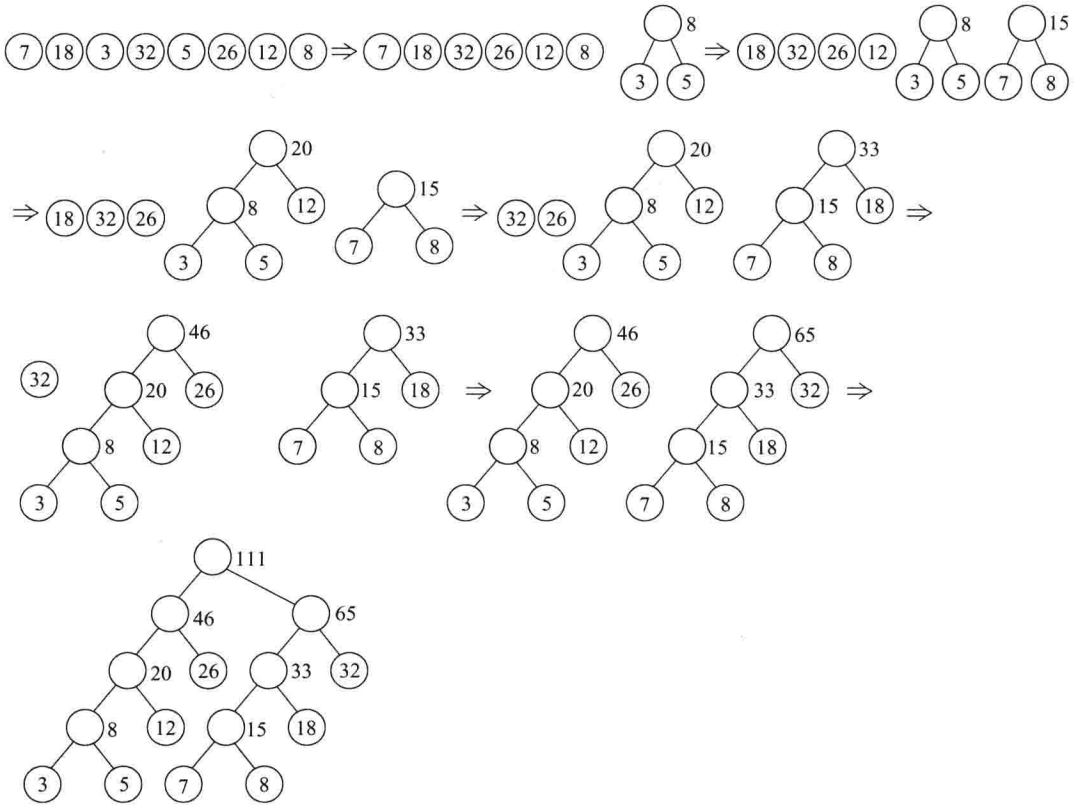


图 6-15　构造关于 w 的哈夫曼树过程

构成的哈夫曼树的形态由于权值叶结点所放的左右不同而不同；另一方面在有相同权值时可以任选一个权值也会产生不同形态的哈夫曼树，但是其 WPL 是唯一的。

$$WPL=3*4+5*4+7*4+8*4+12*3+18*3+26*2+32*2=298$$

【习题 14】 假设用于通信的电文仅由 8 个字母组成，字母在电文中出现的频率分别为 7，19，2，6，32，3，21，10，试为这 8 个字母设计哈夫曼编码。

【解答】 设 8 个字母为 Z1，Z2，Z3，Z4，Z5，Z6，Z7，Z8，在电文中出现的频率分别为 7，19，2，6，32，3，21，10，将 8 个字母作为叶子结点，出现的频率作为各个对应叶子结点的权值，设计一棵哈夫曼树，树中左子树分支表示 bit“0”，右子树分支表示 bit “1”，则得

到的哈夫曼树和二进制前缀编码如图 6-16 所示。

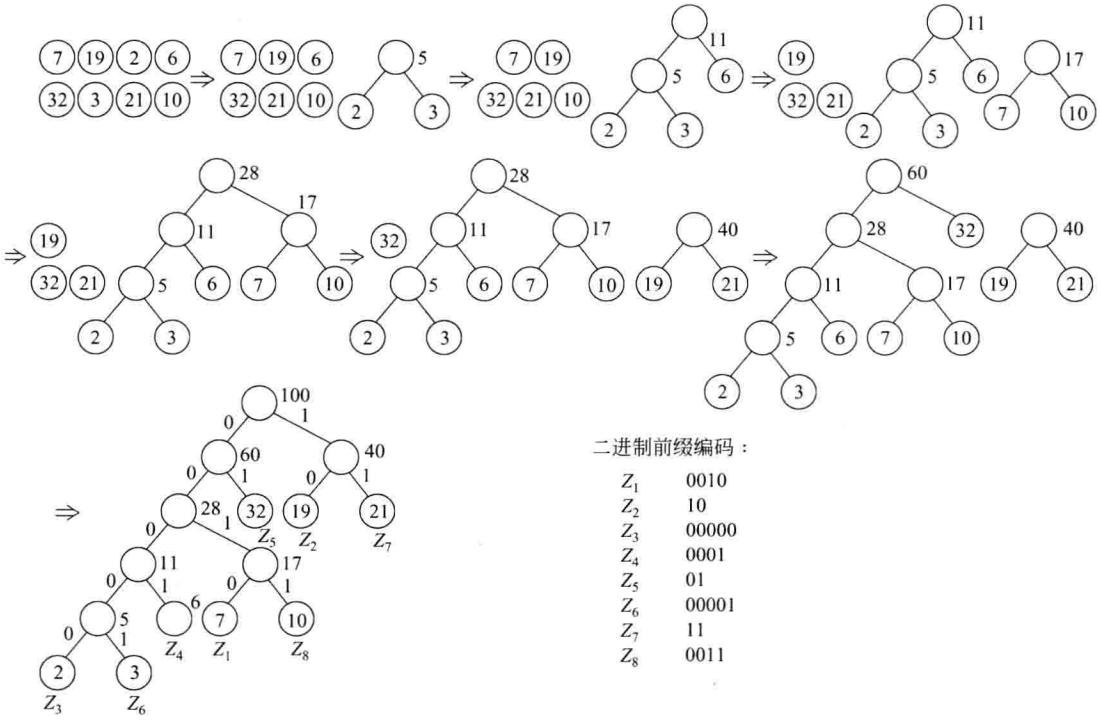


图 6-16　哈夫曼树和二进制前缀编码

【习题 15】 试编写一个将百分制转换成五分制的算法，要求其时间性能尽可能地高（即平均比较次数尽可能少）。假定学生成绩分布情况如下：

分数	0～59	60～69	70～79	80～89	90～100
比例	0.05	0.15	0.40	0.30	0.10

【解答】 百分制转换成五分制的原则如下：

百分制	0～59	60～69	70～79	80～89	90～100
五分制	E	D	C	B	A

将学生成绩分布的比例值作为权值构造一棵哈夫曼树，以此为判定树来设计算法。构造的哈夫曼树如图 6-17(a)所示，对应的判定树如图 6-17(b)所示，转换算法如图 6-17(c)所示。

【习题 16】 单项选择题。

(1) 用双亲存储结构表示树，其优点之一是比较方便_____。

A. 找指定结点的双亲结点　　B. 找指定结点的孩子结点

C. 找指定结点的兄弟结点　　D. 判断某结点是不是叶子结点

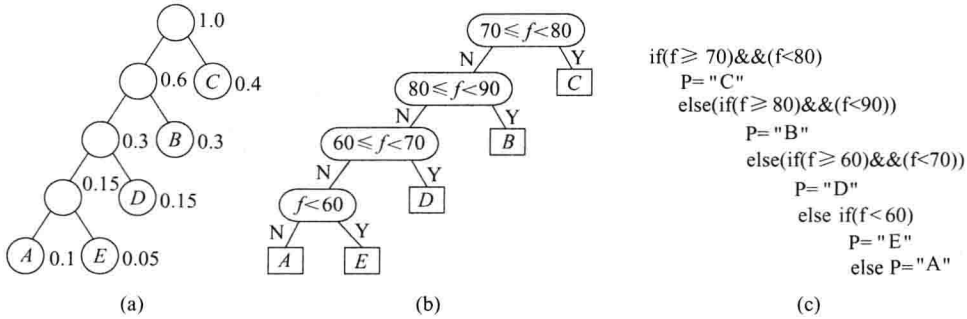


图 6-17　哈夫曼树及其对应的判定树和转换算法

【解答】 A。

(2) 在表示树的兄弟链表结构中有 6 个空的左指针域,7 个空的右指针域,则该树中树叶结点的个数_____。

A. 有 7 个　　B. 有 6 个　　C. 有 5 个　　D. 不能确定

【解答】 在树的孩子兄弟链表结构中,左指针指向结点的第一个孩子结点,右指针指向结点的兄弟结点,该树有 6 个空的左指针域,表示有 6 个结点无孩子结点,所以该树中树叶结点的个数为 6,正确答案为 B。

(3) 一棵具有 n ($n>1$)个结点的二叉树,存放在二叉链表结构中,空指针域个数是_____。

A. $n-1$　　B. $n+1$　　C. n　　D. $n-2$

【解答】 n ($n>1$)个结点的二叉树,用二叉链表存放有 n ($n>1$)个结点 $2n$ 个指针域。n ($n>1$)个结点有 $n-1$ 个分支,即 $n-1$ 个指针域不空,空指针域个数为 $2n-(n-1)=n+1$ 个,正确答案为 B。

(4) 一棵完全的二叉树上有 1001 个结点,其叶子结点的个数是_____。

A. 250　　B. 501　　C. 254　　D. 505

【解答】 二叉树的总结点数 $n=n_0+n_1+n_2$,又根据二叉树的性质 $n_0=n_2+1$,则 $n=2n_2+n_1+1$,完全的二叉树上 n_1 只能是 1 或 0,所以 $1001=2n_2+n_1+1$,$2n_2$ 一定是偶数,1001 是奇数,n_1 一定是 0,$n_2=500$,$n_0=501$,正确答案为 B。

(5) 在任何一棵二叉树中,如果结点 a 有左孩子 b、右孩子 c,则在结点的先序序列、中序序列、后序序列中,_____。

A. 结点 b 一定在结点 a 的前面　　B. 结点 a 一定在结点 c 的前面

C. 结点 b 一定在结点 c 的前面　　D. 结点 a 一定在结点 b 的前面

【解答】 无论哪种遍历方法,都是先遍历左子树,后先遍历右子树,所以结点 b 一定在结点 c 的前面,正确答案为 C 。

(6) 设有 13 个值,用它们组成一棵哈夫曼树共有_____个结点。

A. 13　　B. 12　　C. 26　　D. 25

【解答】 具有 n 个叶子结点的哈夫曼树共有 $2n-1$ 个结点,13 个值组成一棵哈夫曼树共有 25 个结点,正确答案为 D 。

【习题 17】 判断以下叙述的正确性。

(1) 树状结构中每一个结点都有一个前驱结点。

(2) 在树状结构中,处于同一层上的结点之间都存在兄弟关系。

(3) $n(n>2)$个结点的二叉树中,至少有一个度为 2 的结点。

(4) 完全二叉树中的每个结点或者没有孩子或者有 2 个孩子。

(5) 哈夫曼树中不存在度为 1 的结点。

(6) 在二叉树中,具有一个孩子的双亲结点,在中序遍列序列中,它没有后继孩子结点。

(7) 已知二叉树的先序序列和后序序列,并不能唯一确定这棵二叉树。

(8) 存在这样的二叉树,对它采用任何次序的遍历,结果相同。

【解答】

(1) 错误。根结点没有前驱结点。

(2) 错误。处于同一层上的结点但不是同一双亲的结点之间只是堂兄弟关系。

(3) 错误。$n(n>2)$个结点的二叉树中,可以没有度为 2 的结点。

(4) 错误。完全二叉树中可能出现一个结点有左孩子的情况。

(5) 正确。

(6) 错误。仅在只具有左孩子而无右孩子的双亲结点,在中序遍列序列中才没有后继孩子结点。

(7) 正确。从先序序列和后序序列,只能确定根结点,区分不出左子树和右子树,从而不能唯一确定这棵二叉树。

(8) 正确。当二叉树只有一个结点时,三种遍历结果相同。

【习题 18】 写一算法,将一棵以二叉链表存储结构存储的二叉树 t,按顺序方式存储在数组 A 中。

【解答】

```
DATATYPE2 * preorder_array(BTCHINALR * bt,DATATYPE2 * A)
{   DATATYPE2 * p;

    p=A;
    if(bt !=NULL)
    {   * p= bt->data;
        p++;
        p=preorder_array(bt->lchild,p);
        p=preorder_array(bt->rchild,p) ;}
    return p;
}
```

【习题 19】 一棵具有 n 个结点的完全二叉树以顺序方式存储在数组 A 中。写一算法,用二叉链表存储结构构造该二叉树。

【解答】

```
BTCHINALR * arr_bt(DATATYPE2 * p,int n)
{   BTCHINALR * q;
    BTCHINALR * s[20];
    int j,i,x;

    for(i=1;i<=n;i++)
    {  q=(BTCHINALR * )malloc(sizeof(BTCHINALR)); /* 生成一个结点 */
       q->data=p[i];
       q->lchild=NULL;
       q->rchild=NULL;
       s[i]=q;
       if(i !=1) {j=i / 2;                    //j 为 i 的双亲结点
       if(i %  2==0) s[j]->lchild=q;          //i 为 j 的左孩子
       else s[j]->rchild=q;                   //i 为 j 的右孩子
       }
    }
    return s[1];
}
```

【习题 20】 写出二叉树的先序遍历非递归算法,二叉树采用二叉链表存储方式。

【解答】

```
void preorder(BTCHINALR * b)
{  BTCHINALR * st[MAXSIZE], * p;
   int top=-1;

   if(b!=NULL)
   {  top++;
      st[top]=b;
      while(top!=- 1) {  p=st[top];
                         top--;
                         printf("% c ",p->data);
                         if(p->rchild!=NULL) {  top++;
                                                st[top]=p->rchild; }
                         if(p->lchild!=NULL){  top++;
                                               st[top]=p->lchild; }
                      }
      printf("\n"); }
}
```

【习题 21】 编一算法,判别给定的二叉树是否是满二叉树。二叉树采用二叉链表存储方式。

【解答】

```
int isfullbt(BTCHINALR * bt)
{  int flag=0;
```

```
    BTCHINALR * bq[MAXSIZE];
    BTCHINALR * p;
    int order[MAXSIZE];
    int f,r;                    /* 与 bq 组成队列 */
    int i=1;                    /* 二叉树节点顺序编号 */

    f=0; r=0;
    if(bt !=NULL)
    {  r++; bq[r]=bt; order[r]=i;
       while(f<r){  f++;
                    p=bq[f];
                    i++;
                    if(p->lchild!=NULL ) { r++;
                                           bq[r]=p->lchild;
                                           order[r]=i; }
                    i++;
                    if(p->rchild!=NULL ){  r++;
                                           bq[r]=p->rchild;
                                           order[r]=i; }
       }
    }
    if (order[f]==f)            /* 判别是否完全二叉树 */
    {  while(f% 2==1)
       f=f/2;                   /* 判别 f 是否等于 2 的幂次-1 */
       flag=!f; }
    return flag;
}
```

【习题 22】 在以双亲链表结构存储的树中,写出:

(1) 求树中结点双亲的算法;

(2) 求树中结点孩子的算法。

【解答】

(1) 求树中结点双亲的算法:

```
PTNODE parent(PTTREE t, PTNODE n )
{PTNODE p;

if (n.parent !=- 1 ) p=t.nodes[n.parent];
else {   printf("this is root! \n");
         p=n; }
return p;
}
```

(2) 求树中结点孩子的算法:

```
PTTREE child(PTTREE t,int n )
{   PTTREE c;
    int i;

    c.nodenum=0;
    for(i=0;i<t.nodenum ;i++)
        if(t.nodes[i].parent==n)
        {  c.nodes[c.nodenum]=t.nodes[i];
           c.nodenum++; }
    return c;
}
```

【习题 23】 已知一棵具有 n 个结点的二叉树,写一算法,用二叉链表存储方式构造建立。

题目分析:本章的很多编程题都是对二叉树的各种操作,如对二叉树的各种遍历,求二叉树叶子结点个数,求二叉树的深度,二叉树左右子树交换等,这些操作都基于在内存中建立的二叉树上,下面提供一个建立二叉树的函数,供各应用程序调用,函数源程序以"二叉树.c"为名存放在光盘上。

二叉树的存储结构采用二叉链表,结构如下:

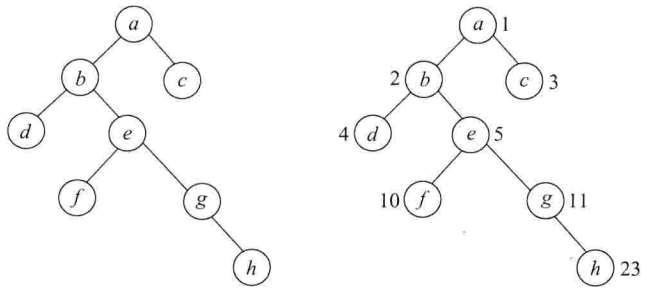
```
#define DATATYPE2 char

typedef struct node1
{ DATATYPE2 data;
struct node1 * lchild, * rchild;
}BTCHINALR;
```

【解答】 源代码如下:

```
BTCHINALR * createbt( )
{   BTCHINALR * q;
    struct node1 * s[50];
    int j,i,x;

    printf("建立二叉树,输入结点对应的编号和值\n\n");
    printf("i,x=");
    scanf("% d,% c",&i,&x);
    while(i !=0 && x !='$ ')
      {q= (BTCHINALR * )malloc(sizeof(BTCHINALR));    /* 建立一个新结点 q* /
      q->data=x; q->lchild=NULL; q->rchild=NULL;
      s[i]=q;                          /* q新结点地址存入 s指针数组中 * /
      if(i !=1)                        /* i=1,q结点是根结点 * /
        {j=i / 2;                      /* 否则求 q结点的双亲结点的编号 j* /
         if(i %  2==0) s[j]->lchild=q;  /* q结点编号为偶数则挂在双亲结点 j的左边 * /
         else s[j]->rchild=q;}         /* q结点编号为奇数则挂在双亲结点 j的右边 * /
```

```
        printf("i,x=");
        scanf("% d,% c",&i,&x);}
    return s[1];          /* 返回根结点地址 */
}
```

下面举例说明二叉树的建立。设有一棵二叉树如图 6-18(a)所示，二叉树结点的值设为整数，将二叉树模拟为完全二叉树从根开始对结点进行编号，编号从 1 开始，结果如图 6-18(b)所示。在运行过程中要求输入结点对应的编号和值时，请按图 6-18(c)中的数据输入，最后以编号 $i=0$；结点值 $x='\$'$ 结束。



(a)　　(b)

i	x
1	a
2	b
3	c
4	d
5	e
10	f
11	g
23	h
0	\$

(c)

图 6-18　二叉树的建立

【习题 24】 编写程序，实现对二叉树中序遍历，并输出遍历结果。算法可以用递归算法实现也可以用非递归算法实现。二叉树采用二叉链表存储方式。

【解答】

```
#include <stdio.h>
#include "datastru.h"
#include <malloc.h>
#include "二叉树.c"

void inorder(BTCHINALR * bt)
/* 中序遍历二叉树 (递归算法) */
{if(bt !=NULL)
    {   inorder(bt->lchild);
        printf("% c ",bt->data);
        inorder(bt->rchild); }
}

void inorder_notrecursive(BTCHINALR * bt)
/* 中序遍历二叉树 (非递归算法) */
{BTCHINALR * q, * s[20];
  int top=0;
  int bool=1;
```

```
    q=bt;
    do { while(q !=NULL)
            { top++; s[top]=q; q=q->lchild; }
         if(top==0) bool=0;
         else { q=s[top]; top--;
           printf("% c ",q->data); q=q->rchild; }
    }while(bool);
}

main( )
{ BTCHINALR * bt;
  char ch;
  int i;

bt=createbt(); i=1;
while(i) {
    printf("\n中序遍历二叉树(递归按 y 键,非递归按 n 键): ");
    fflush(stdin); scanf("% c",&ch);
    if(ch=='y') inorder(bt); else inorder_notrecursive(bt);
    printf("\n");
    printf("\n继续操作吗? (继续按 1 键,结束按 0 键): ");
    fflush(stdin); scanf("% d",&i);
  }
}
```

【习题 25】 编写程序,计算二叉树叶子结点数,二叉树采用二叉链表存储方式。

题目分析:计算二叉树叶子结点数的算法很多很多,可用递归算法实现,也可用非递归算法实现。本程序采用了递归算法实现。

【解答】

```
#include <stdio.h>
#include "datastru.h"
#include <malloc.h>
#include "二叉树.c"

int leaf(BTCHINALR * bt)
{  if(bt==NULL) return 0;
   else if(bt->lchild==NULL && bt->rchild==NULL) return 1;
        else return(leaf(bt->lchild)+leaf(bt->rchild));
}

main( )
{ BTCHINALR * bt;
  int leafnum;
```

```
 bt=createbt( );
 leafnum=leaf(bt);
 printf("\n 二叉树的叶子结点数=% d\n\n",leafnum);
}
```

【习题 26】 编写程序，计算二叉树的深度，树根为第一层。二叉树采用二叉链表存储方式。

【解答】

```
#include <stdio.h>
#include "datastru.h"
#include <malloc.h>
#include "二叉树.c"

int treehight(BTCHINALR * bt)
{   int lh, rh, h;

    if(bt==NULL) h=0;
    else
      { lh=treehight(bt->lchild);
        rh=treehight(bt->rchild);
        h=(lh>rh ? lh : rh)+1;}
    return h;
}

main( )
{ BTCHINALR * bt;
  int treeh;

 bt=createbt( ); treeh=treehight(bt);
 printf("\n 二叉树深度=% d\n\n",treeh);
}
```

【习题 27】 编一程序，查找给定结点是否在二叉树中存在。设定二叉树中结点值互不相同。二叉树采用二叉链表存储方式。

【解答】

```
#include "datastru.h"
#include<stdio.h>
#include<malloc.h>
#include "二叉树.c"

BTCHINALR * search_ch(BTCHINALR * cur, char x){
/* 先序查找 */
BTCHINALR * temp;
```

```
    if(cur==NULL) return NULL;
    if(x==cur->data) return cur;
    temp=search_ch(cur->lchild,x);
    if (temp !=NULL) return temp;
    else return search_ch(cur->rchild,x);
};

main()
{ BTCHINALR * bt, * p;
  char ch;
  int i;

bt=createbt(); fflush(stdin); i=1;
while(i){ printf("输入待查结点数据: "); scanf("% c",&ch);
          p=search_ch(bt,ch);
          if(p==NULL) printf("无此结点\n\n");
          else printf("结点存在\n\n");
          printf("\n继续操作吗? (继续按 1 键,结束按 0 键): \n");
          fflush(stdin); scanf("% d",&i); fflush(stdin);}
printf("\n");
}
```

【习题 28】 编一程序,查找给定结点的双亲结点是否在二叉树中存在。二叉树用二叉链表存储结构存储。

题目分析:实现此功能是基于上一算法的基础。当确定给定结点存在二叉树中,则输出该结点的双亲结点;如果给定结点是根结点,则给出相应的提示信息。

【解答】

```
#include "datastru.h"
#include<stdio.h>
#include<malloc.h>
#include "二叉树.c"

BTCHINALR * search_ch(BTCHINALR * cur, char x){
/*先序查找:在以 cur 为根的子树中查找值为 x 的结点是否存在*/
BTCHINALR * temp;

    if(cur==NULL) return NULL;
    if(x==cur->data) return cur;
    temp=search_ch(cur->lchild,x);
    if (temp !=NULL) return temp;
    else return search_ch(cur->rchild,x);
};
```

```
BTCHINALR * parent(BTCHINALR * start, BTCHINALR * current){
/* 从 start 所指结点起查找当前结点 current 的父亲结点 */
    BTCHINALR * p;

    if(start==NULL)return NULL;
    if(start->lchild==current||start->rchild==current)return start;
    p=parent(start->lchild,current);
    if(p!=NULL) return p;
    else return parent(start->rchild,current);
}

main()
{
  BTCHINALR * bt, * p, * temp;
  char ch;
  int i;

bt=createbt();    /* 建立二叉树,根为 bt */
fflush(stdin);
i=1;
while(i){ printf("\n 输入结点数据: "); scanf("% c",&ch);
          p=search_ch(bt,ch);
          if(p==NULL) printf("\n 无此结点\n\n");
          else {if(p==bt) printf("\n 无双亲结点,是根结点\n\n");
              else {temp= parent(bt,p);
                    printf("\n% c 的双亲结点是% c\n\n",ch,temp->data);}}
          printf("\n 继续操作吗? (继续按 1 键,结束按 0 键): \n");
          fflush(stdin);scanf("% d",&i);fflush(stdin);}
printf("\n");
}
```

【习题 29】 编一程序,实现二叉树左右子树交换。二叉树采用二叉链表存储方式。

题目分析:从二叉树的根结点开始,将每一个结点的左右子树交换。为验证算法的正确,特显示交换前及交换后的二叉树的中序遍历结果。二叉树左右子树交换算法用递归算法实现。前面图 6-18(a)所示的二叉树在运行本算法后的新二叉树如图 6-19 所示。运行过程如图 6-20 所示。

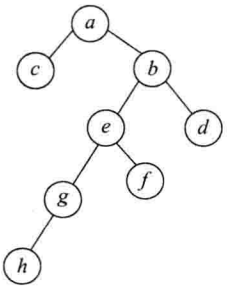


图 6-19 二叉树

【解答】

```
#include "datastru.h"
#include<stdio.h>
#include<malloc.h>
#include "二叉树.c"
```

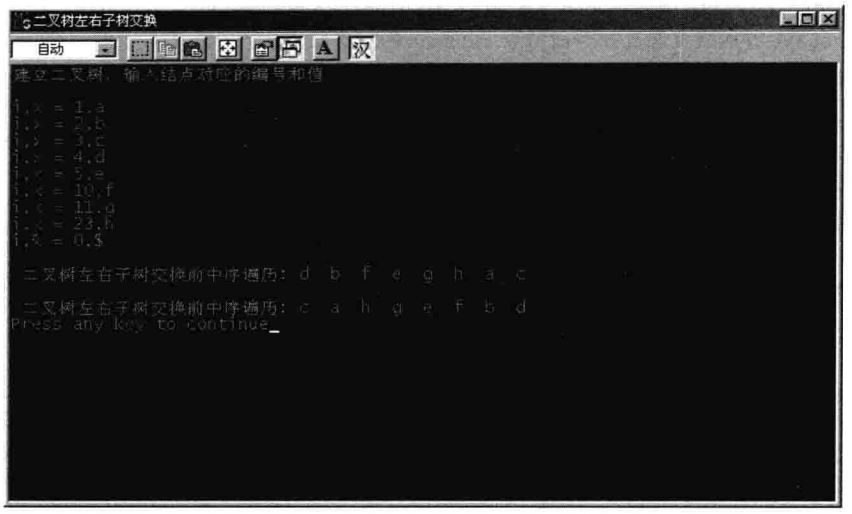


图 6-20　二叉树左右子树交换运行过程

```
BTCHINALR * change(BTCHINALR * bt)
/* 二叉树左右子树交换递归算法 */
{  BTCHINALR * p;

   if(bt!=NULL)
     if(bt->lchild!=NULL || bt->rchild!=NULL)
       {p= (BTCHINALR * )malloc(sizeof(BTCHINALR));
        p->data=bt->data; p->rchild=NULL;
        p->lchild=bt->lchild;
        bt->lchild=bt->rchild; bt->rchild=p->lchild;
        bt->lchild=change(bt->lchild);
        bt->rchild=change(bt->rchild);
       }
   return bt;
}

void inorder(BTCHINALR * bt)
/* 中序遍历二叉树(递归算法) */
{if(bt !=NULL)
    {   inorder(bt->lchild);
        printf("% c ",bt->data);
        inorder(bt->rchild); }
}

main()
{
  BTCHINALR * bt;
```

```
    bt=createbt();
    printf("\n 二叉树左右子树交换前中序遍历: ");
    inorder(bt);
    printf("\n");
    bt=change(bt);
    printf("\n 二叉树左右子树交换前中序遍历: ");
    inorder(bt);
    printf("\n");
}
```

【习题 30】 编写人机下棋游戏程序。

题目分析：这是一个井字游戏人机下棋的程序，游戏程序的逻辑思路采用树形结构分析法。具体说明如图 6-21～图 6-26 所示。

1. 游戏规则。

棋盘格式如图 6-21 所示。参加者是人和计算机。人和计算机在棋盘上交替走步，约定使用符号计算机用“x”，人用“o”，谁使棋盘上三点一直线为己方符号，则胜。

2. 程序要求。

- 按游戏树法则编程，使用求值函数计算。
- 要求下棋深度至少考虑一步，程序结果只能是计算机胜或平局。
- 开始由哪一方先走可以选择。

3. 算法辅导。

(1) 从初始状态开始，有三种不同的走法，如图 6-22 所示。

- 中间——m_1。
- 四角——m_2。
- 边中间——m_3。

1	2	3
4	5	6
7	8	9

图 6-21 棋盘格式

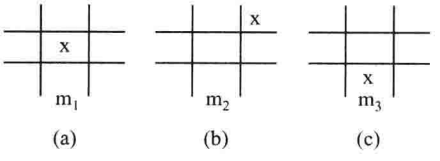


图 6-22 三种不同走法

(2) 计算机考虑的优先级次序。

① 已有一行三个对方的符号，即输；

② 已有一行三个自己的符号，即赢；

③ 建立一行包括三个自己的符号；

④ 阻止对方形成一行三个对方的符号(即在计算机下了一步以后，对手不应出现在一行上有两个对方符号和一个空格的情况)；

⑤ 建立一行包括两个自己的符号和一个空格；

⑥ 阻止对方形成一行两个对方符号和一个空格；

⑦ 建立一行包括一个自己的符号和两个空格。

(3) 变量规定。

① n_3存放一行三个“o”的行数；

② c_3存放一行三个“x”的行数；

③ n_2存放一行两个“o”的行数（另一个为空格）；

④ c_2存放一行两个“x”的行数（另一个为空格）；

⑤ n_1存放一行只有一个“o”的行数（另两个为空格）；

⑥ c_1存放一行只有一个“x”的行数（另一个为空格）。

在图 6-23 中，$c_1=2$，$n_1=3$，$c_2=n_2=c_3=n_3=0$。

在图 6-24 中，$c_1=3$，$n_1=1$，$n_2=1$，$c_2=c_3=n_3=0$。

虚线是由“x”符号组成的行，实线是由“o”符号组成的行。

4. 求值函数。

$$y=k_3*c_3-k'_2*n_2+k_2*c_2-k'_1*n_1+k_1*c_1$$

其中，k_3，k'_2，k_2，k'_1，k_1是常数，分别可取 128，63，31，15，7。k_3可以任取，其他值满足下列不等式：

$$k_3>2*k'_2,\quad k'_2>2*k_2,\quad k_2>2*k'_1,\quad k'_1>2*k_1$$

5. 最佳移动如何确定(观察深度为一步的移动)。

以计算机走第一步为例，图 6-25 是计算机可能走的九步棋面 $s_1\sim s_9$，s_1表示计算机第一步走 5 号位，s_2表示走 1 号位，s_3表示走 2 号位，走 3、7、9 号位的棋面和 s_2对称，走 4、6、8 号位的棋面和 s_3对称。

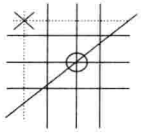


图 6-23 优先级次序之一

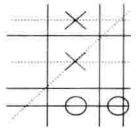


图 6-24 优先级次序之二

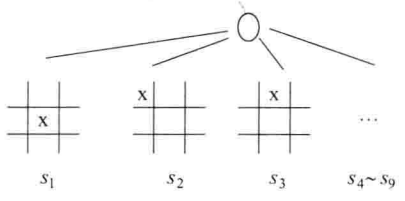


图 6-25 可能走的九步棋面

在计算机所有可能走步的棋面构成的树中，把求值函数用于终端结点，分别求出 s_1，s_2，s_3的值，取最大值的走步对计算机最有利，即可确定走这一步。

如果人先走，设人走在 6 号位，则计算机所有可能走的棋面构成的树也如图 6-26 所示，求值函数用于终端结点，求出终端结点对应的函数值，取其中最大值对应的走步为计算机走步。

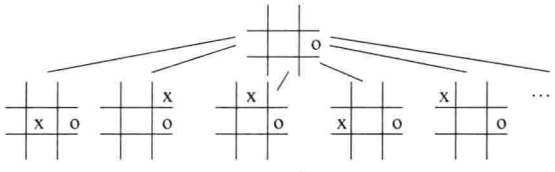


图 6-26 所有可能走的棋面构成的树

【解答】

```
#include<stdio.h>

typedef char chess [10];
typedef int temparr [10];

chess arr;
temparr brr;
int number, suc, c3, n2, c2, n1, c1;
char ch;

void inarrdata(chess a)
{ a[1]='1'; a[2]='2'; a[3]='3';
  a[4]='4'; a[5]='5'; a[6]='6';
  a[7]='7'; a[8]='8'; a[9]='9';
}

void display(chess a)
{ printf(" 棋盘显示\n");
  printf("\n"); printf("\n");
  printf(" %c | %c | %c\n",a[1],a[2],a[3]);
  printf("----------\n");
  printf(" %c | %c | %c\n",a[4],a[5],a[6]);
  printf("----------\n");
  printf(" %c | %c | %c\n",a[7],a[8],a[9]);
  printf("\n"); printf("\n");
}

int arrfull()
{ int i;
  int arrf=0;

  for(i=1; i<=9; i++)
      if(i==arr[i]-48) arrf=1;
  return arrf;
}

void cn(int line)
{  switch(line)
    {case 0: c3=c3+1; break;
     case 1: n2=n2+1; break;
     case 2: c2=c2+1; break;
     case 3: n1=n1+1; break;
     case 4: c1=c1+1; break;}
```

```
}

int linenum(char a,char b,char c)
{  int ln=5;

   if((a=='X') && (b=='X') && (c=='X'))
       ln=0;
   if(((a=='O') && (b=='O') && (c !='O'))
   || ((a=='O') && (b !='O') && (c=='O'))
   || ((a !='O') && (b=='O') && (c=='O')))
       ln=1;
   if(((a=='X') && (b=='X') && (c !='X'))
   || ((a=='X') && (b !='X') && (c=='X'))
   || ((a !='X') && (b=='X') && (c=='X')))
       ln=2;
   if(((a=='O') && (b !='O') && (c !='O'))
   || ((a !='O') && (b=='O') && (c !='O'))
   || ((a !='O') && (b !='O') && (c=='O')))
       ln=3;
   if(((a=='X') && (b !='X') && (c !='X'))
   || ((a !='X') && (b=='X') && (c !='X'))
   || ((a !='X') && (b !='X') && (c=='X')))
       ln=4;
   return ln;
}

int maxbrr(int * br)
    { int temp, i, mb;
      temp=-888;

      for(i=1; i<=9; i++)
          {if (temp<=br[i])
              {temp=br[i];
              mb=i;}}
      return mb;
    }

void manstep()
/*人走棋处理模块*/
{  int j;

   display(arr);
   if(arrfull())              /*如果棋盘上还有可下棋的位置,给人走一步棋*/
   {printf("输入下一步对应位置编号 : ");
```

```
    scanf("% d",&j);
    printf("\n");
    while((j<1)||(j>9) || (j !=arr[j]-48))
       { printf("输入错误,重新输入 : ");
         scanf("% d",&j);}
   arr[j]='O';}
}

void computerstep()
{/* 计算机走棋处理模块 */
   int i;

   if (arrfull()) /* 如果棋盘上还有可下棋的位置,则计算机走一步棋 */
      {for(i=1; i<=9; i++) /* 对每一步可走棋的位置进行计算 */
        {if(i==arr[i]-48)
          { c3=0; n2=0; c2=0;
            n1=0; c1=0;
            arr[i]='X';
            number=linenum(arr[1],arr[2],arr[3]); cn(number);
            number=linenum(arr[4],arr[5],arr[6]); cn(number);
            number=linenum(arr[7],arr[8],arr[9]); cn(number);
            number=linenum(arr[1],arr[4],arr[7]); cn(number);
            number=linenum(arr[2],arr[5],arr[8]); cn(number);
            number=linenum(arr[3],arr[6],arr[9]); cn(number);
            number=linenum(arr[1],arr[5],arr[9]); cn(number);
            number=linenum(arr[3],arr[5],arr[7]); cn(number);
            brr[i]=(128*c3-63*n2+31*c2-15*n1+7*c1); /* 计算此步权值 */
            arr[i]=i+48;}
         else brr[i]=-999;}
       arr[maxbrr(brr)]='X';        /* 确定计算机走哪一步:权值最大的一步 */
       c3=0; n2=0; c2=0; n1=0; c1=0;
       number=linenum(arr[1],arr[2],arr[3]); cn(number);
       number=linenum(arr[4],arr[5],arr[6]); cn(number);
       number=linenum(arr[7],arr[8],arr[9]); cn(number);
       number=linenum(arr[1],arr[4],arr[7]); cn(number);
       number=linenum(arr[2],arr[5],arr[8]); cn(number);
       number=linenum(arr[3],arr[6],arr[9]); cn(number);
       number=linenum(arr[1],arr[5],arr[9]); cn(number);
       number=linenum(arr[3],arr[5],arr[7]); cn(number);
       if(c3 !=0 )                  /* 计算机已胜 */
           { display(arr);
             printf("\n");
             printf("计算机胜!!! \n\n\n");
             fflush(stdin);
```

```
                suc=0;}
            }
        else suc=0;
    }

    main()
      {
       inarrdata(arr);                    /*棋盘坐标编号*/
       display(arr);                      /*显示初始坐标*/
       suc=1;
       printf("你先走吗? (y/n) ");
       scanf("%c",&ch);
       printf("\n");
       if ((ch=='y') || (ch=='Y'))   /*人先走棋*/
          { while (suc )
              { manstep(); computerstep();}
           display(arr);}
      else { while (suc )                 /*计算机先走棋*/
               { computerstep();
                 if(suc) manstep();}
           }
    }
```

6.2 实训练习题

【习题 31】 简述二叉树与度为 2 的树之间的差别。

【习题 32】 找出分别满足下列条件的所有二叉树：

(1) 先根序列和中根序列相同；

(2) 后根序列和中根序列相同；

(3) 先根序列和后根序列相同。

【习题 33】 已知一棵二叉树的中根序列和先根序列分别为 *ECBHFDJIGA* 和 *ABCEDFHGIJ*，试画出这棵二叉树。

【习题 34】 单项选择题。

(1) 用双亲存储结构表示树，其优点之一是_____比较方便。

A. 判断两个指定结点是不是兄弟结点

B. 找指定结点的双亲结点

C. 判断指定结点在第几层

D. 计算指定结点的度数

(2) 一棵有 124 个叶子结点的完全的二叉树，最多有_____个结点。

A. 247　　B. 248　　C. 249　　D. 250

(3) 在高度为 h 的完全的二叉树中，_____。

A. 度为 0 的结点都在第 h 层上

B. 第 i ($1 \leqslant i \leqslant h$)层上结点都是度为 2 的结点

C. 第 i ($1 \leqslant i \leqslant h-1$)层上有 2^{i-1} 个结点

D. 不存在度为 1 的结点

(4) 根据使用频率为 5 个字符设计的哈夫曼编码不可能是_____。

A. 111,110,10,01,00　　B. 000,001,010,011,1

C. 100,11,10,1,0　　D. 001,000,01,11,10

【习题 35】 判断以下叙述的正确性。

(1) 度为 m 的树中至少有一个度为 m 的结点。

(2) 只要知道完全二叉树中结点的先序序列,就可以唯一确定它的逻辑结构。

(3) 哈夫曼树中,权值较大的叶结点一般都离根结点较远。

(4) 在二叉树中,具有两个孩子的双亲结点,在中序遍列序列中,它的后继结点最多只能有一个双亲结点。

(5) 二叉树就是度为 2 的树。

(6) 将一棵树转换成二叉树后,根结点没有左子树。

【习题 36】 在以孩子链表结构存储的树中,写出:

(1) 求树中结点双亲的算法;

(2) 求树中结点孩子的算法。

【习题 37】 计算二叉树中结点的总数,你能写出多少种算法?二叉树用二叉链表存储结构存储。

【习题 38】 编一算法,判别给定的二叉树是否是完全二叉树。二叉树用二叉链表存储结构存储。

【习题 39】 编写程序,实现对建立的二叉树进行先序遍历,并输出遍历结果。二叉树用二叉链表存储结构存储。

【习题 40】 编写程序,实现对建立的二叉树进行后序遍历,并输出遍历结果。二叉树用二叉链表存储结构存储。

【习题 41】 计算二叉树叶子结点数的算法很多很多,粗算也可有六七种。编写程序,实现这一操作(两个以上)。

【习题 42】 二叉树用二叉链表存储结构存储,编写程序,输出给定结点到根结点的路径。

题目分析:求给定结点到根结点的路径可基于习题 28 查找给定结点的双亲结点的程序的思路。如果给定结点存在又是根结点,则该结点无祖先结点;如果给定结点存在又不是根结点,则查找给定结点的双亲结点,再查双亲结点的双亲结点,一直查到根结点为止,即求得给定结点到根结点的全部祖先结点。

第7章 图

图是另一种重要的非线性结构，它比树的结构更复杂，更灵活。习题中涉及图的两种常用的存储结构即图的邻接矩阵和邻接链表。

7.1 习题解析

【习题 1】 画出无向图 7-1 的邻接矩阵和邻接链表示意图，写出每个顶点的度。

【解答】

邻接矩阵示意图如图 7-2 所示。

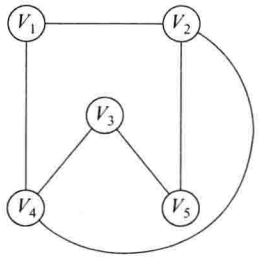


图 7-1 无向图

$$\begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

图 7-2 习题 1 的邻接矩阵

邻接链表示意图如图 7-3 所示。

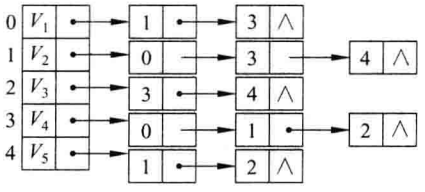


图 7-3 习题 1 的邻接链表

顶点 V_1 的度为 2，顶点 V_2 的度为 3，顶点 V_3 的度为 2，顶点 V_4 的度为 3，顶点 V_5 的度为 2。

【习题 2】 画出有向图 7-4 的邻接矩阵、邻接链表和逆邻接链表示意图，并写出每个顶点的入度、出度。

【解答】

邻接矩阵示意图如图 7-5 所示。

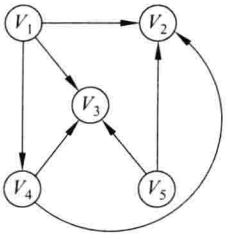


图 7-4 有向图

$$\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{bmatrix}$$

图 7-5 习题 2 的邻接矩阵

邻接链表示意图如图 7-6 所示。

逆邻接链表示意图如图 7-7 所示。

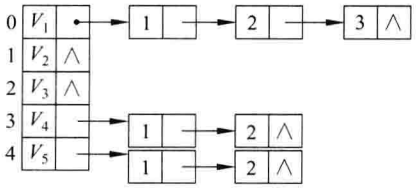


图 7-6 习题 2 的邻接链表

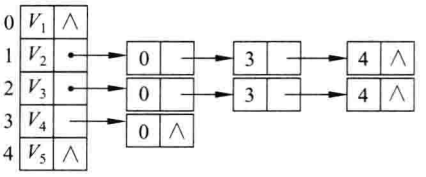


图 7-7 习题 2 的逆邻接链表

顶点 V_1 的入度为 0，出度为 3；顶点 V_2 的入度为 3，出度为 0；顶点 V_3 的入度为 3，出度为 0；顶点 V_4 的入度为 1，出度为 2；顶点 V_5 的入度为 0，出度为 2。

【习题 3】 对应图 7-8，写出从 V_1 出发深度优先搜索遍历结果和广度优先搜索遍历结果各 3 个。

【解答】

深度优先搜索遍历 1：$V_1V_2V_4V_3V_5$。深度优先搜索遍历 2：$V_1V_3V_5V_2V_4$。深度优先搜索遍历 3：$V_1V_3V_2V_4V_5$。

广度优先搜索遍历 1：$V_1V_2V_3V_4V_5$。广度优先搜索遍历 2：$V_1V_4V_3V_2V_5$。广度优先搜索遍历 3：$V_1V_3V_2V_4V_5$。

【习题 4】 求图 7-9 的连通分量。

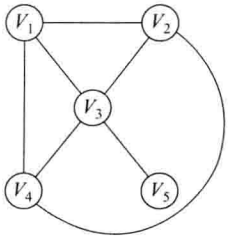


图 7-8 习题 3 的图

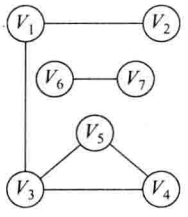


图 7-9 习题 4 的图

【解答】 有 2 个连通分量,如图 7-10 所示。

【习题 5】 画出图 7-11 生成最小生成树的示意过程图。

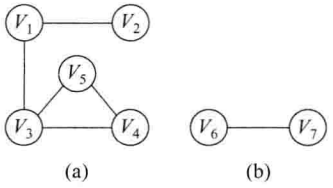


图 7-10 连通分量

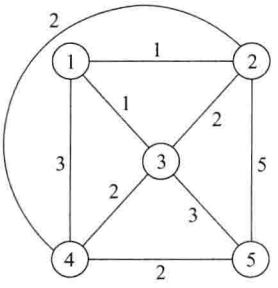


图 7-11 习题 5 的图

【解答】

生成最小生成树的示意过程如图 7-12 所示。

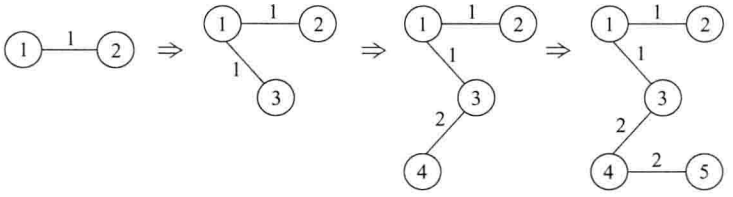


图 7-12 生成最小生成树的示意过程

【习题 6】 叙述对有向无环图求拓扑排序序列的步骤。并写出图 7-13 有向无环图的 4 个不同的拓扑有序序列。

【解答】 有向无环图上求拓扑排序序列的步骤如下:

(1) 在有向无环图中选择一个入度为 0 的顶点,输出该顶点。

(2) 从图中删除该顶点和所有以它为始点的弧。

(3) 重复执行(1)、(2)直到找不到入度为 0 的顶点时,完成拓扑排序(此时图中应无顶点存在)。

图中顶点 1 和顶点 3 是入度为 0 的顶点,拓扑有序序列可以从顶点 1 或顶点 3 开始,4 个拓扑有序序列:

1,2,3,4,5,6,7,8

1,4,2,7,8,3,5,6

3,5,1,2,4,6,7,8

3,1,5,2,4,6,7,8

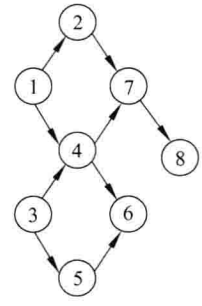


图 7-13 有向无环图

【习题 7】 写出有向无环图(见图 7-14)的所有的拓扑序列。试添加一条弧后,此图即只有唯一一个拓扑序列。

【解答】 所有的拓扑序列为:

1,3,2,4

1,2,3,4

2,1,3,4

添加一条弧以后如能使图中入度为0的顶点和出度为0的顶点都是唯一的，则一定仅有唯一的拓扑序列。如图7-15所示，拓扑序列唯一：1,3,2,4。

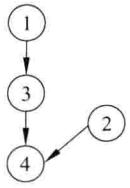


图7-14 有向无环图

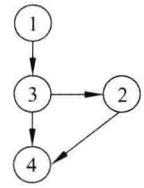


图7-15 习题7的拓扑序列

【习题8】 有 n 个顶点的有向强连通图最多有多少条边？最少有多少条边？

【解答】 有 n 个顶点的有向强连通图最多有 $n(n-1)$ 条边。即 n 个顶点的有向完全图；最少有 n 条边，即 n 个顶点依次首尾相接构成一个环状图。

【习题9】 单项选择题。

1. 所谓简单路径是指_____。

A. 任何一条边在这条路径上不重复出现

B. 任何一个顶点在这条路径上不重复出现

C. 这条路径由一个顶点序列构成，不包含边

D. 这条路径由一个边的序列构成，不包含顶点

【解答】 路径序列中顶点不重复出现的路径称为简单路径。本题答案为B。

2. 带权有向图 G 用邻接矩阵 A 存储，则 V_i 的入度等于A中_____。

A. 第 i 行非∞的元素之和　　B. 第 i 列非∞的元素之和

C. 第 i 行非∞且非0的元素个数　　D. 第 i 列非∞且非0的元素个数

【解答】 有向图的邻接矩阵中，0和∞表示的都不是有向边，而入度是由邻接矩阵的列中元素计算出来的。本题答案为D。

3. 无向图的邻接矩阵是一个_____。

A. 对称矩阵　　B. 零矩阵

C. 上三角矩阵　　D. 对角矩阵

【解答】 A。

4. 一个有 n 个顶点的无向图最多有_____条边。

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

【解答】 C。

5. 在一个具有 n 个顶点的无向图中，要连通全部顶点至少需要_____条边。

A. n　　B. $n+1$　　C. $n-1$　　D. $n/2$

【解答】 C。

6. 一个有向图 G 的邻接表存储如图7-16所示，现按深度优先搜索遍历，从 V_1 出发，所得到的顶点序列是_____。

A. 1,2,3,4,5　　B. 1,2,3,5,4

C. 1,2,4,5,3　　D. 1,2,5,3,4

【解答】 B。

7. 对图 7-17 所示的无向图，从顶点 V_1 开始进行深度优先遍历，可得到顶点访问序列是_____。

A. 1,2,4,3,5,7,6　　B. 1,2,4,3,5,6,7

C. 1,2,4,5,6,3,7　　D. 1,2,3,4,5,7,6

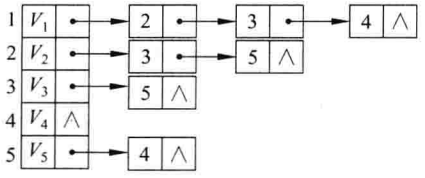


图 7-16　一个有向图 G 的邻接表存储

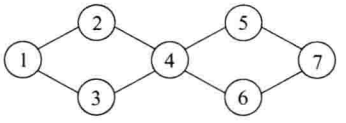


图 7-17　无向图

【解答】 选项 B 中从顶点 5 到顶点 6 不符合深度优先遍历规则；选项 C 中从顶点 5 到顶点 6 不符合深度优先遍历规则；选项 D 中从顶点 2 到顶点 3 不符合深度优先遍历规则。本题答案为 A。

8. 对如图 7-17 所示的无向图，从顶点 1 开始进行广度优先遍历，可得到顶点访问序列是_____。

A. 1,3,2,4,5,6,7　　B. 1,2,4,3,5,6,7

C. 1,2,3,4,5,7,6　　D. 2,5,1,4,7,3,6

【解答】 选项 B 中从顶点 2 到顶点 4 不符合广度优先遍历规则；选项 C 中从顶点 5 到顶点 7 不符合广度优先遍历规则；选项 D 中从顶点 2 开始就不符合广度优先遍历规则。本题答案为 A。

9. 一个无向连通图的生成树是含有该连通图的全部顶点的_____。

A. 极小连通子图　　B. 极小子图

C. 极大连通子图　　D. 极大子图

【解答】 A。

10. 若图的邻接矩阵中主对角线上的元素全是 0，其余元素全是 1，则可以断定该图一定是_____。

A. 无向图　　B. 不是带权图　　C. 有向图　　D. 完全图

【解答】 D 。

【习题 10】 判断以下叙述的正确性。

(1) n 个顶点的无向图至多有 $n(n-1)$ 条边。

(2) 在有向图中，各顶点的入度之和等于各顶点的出度之和。

(3) 如果表示有向图的邻接矩阵是对称矩阵，则该有向图一定是完全有向图。

(4) 连通图的生成树包含了图中所有顶点。

(5) 强连通图不能进行拓扑排序。

(6) 只要无向网络中没有权值相同的边,其最小生成树就是唯一的。

(7) 只要无向网络中有权值相同的边,其最小生成树就不可能是唯一的。

(8) 连通分量是无向图中的极小连通子图。

(9) 强连通分量是有向图中的极大强连通子图。

(10) 在一个有向图的拓扑序列中,若顶点 a 在顶点 b 之前,则图中必有一条弧$<a,b>$。

(11) 有向图的遍历不可采用广度优先搜索方法。

【解答】

(1) 错误。n 个顶点的无向图至多有 $n(n-1)/2$ 条边。

(2) 正确。在有向图中,每一条弧既是一个顶点的入度又是另一个顶点的出度。

(3) 错误。只能表示有向图中的如两个顶点 a,b 之间有 a 到 b 的弧,则一定有 b 到 a 的弧。不一定是完全有向图。

(4) 正确。

(5) 正确。拓扑排序的前提是无环有向图。强连通图中一定有环。

(6) 正确。

(7) 错误。其最小生成树有可能是唯一的,有可能不唯一。

(8) 错误。连通分量是无向图中的极大连通子图。

(9) 正确。强连通分图也称为极大强连通子图。

(10) 错误。拓扑序列中顶点 a 在顶点 b 之前,并不一定必有一条弧$<a,b>$。

(11) 错误。有向图的遍历可以采用广度优先搜索方法。

【习题 11】 已知有 m 个顶点的无向图,采用邻接矩阵结构存储,写出下列算法:

(1) 计算图中有多少条边?

(2) 判断任意两个顶点 i 和 j 之间是否有边相连?

(3) 计算任意一个顶点的度是多少?

【解答】

(1) 计算图中有多少条边的算法。

```
int edges(int * a,int m)
{   int i,j;
    int e=0;
    for (i=1;i<m;i++)
        for(j=0;j<i;j++)
            if(* (a+i * m+j)==1)
                e++;
    return e;
}
```

(2) 判断两个顶点 i 和 j 之间是否有边相连的算法。

```
int isadj(int i, int j, int * a ,int m)
{  return * (a+(i-1) * m+j-1); }
```

(3) 计算一个顶点的度数的算法。

```
int degree(int i,int * a ,int m)
{   int d=0;
    int j;

    for (j=0;j<m;j++)
          if(* (a+(i-1) * m+j)==1)
                d++;
    return d;
}
```

【习题 12】 参照建立有向网的邻接矩阵的算法,写出下列算法:

(1) 建立有向图邻接矩阵算法。

题目分析:参照主教材中建立有向网的邻接矩阵的算法,建立有向图邻接矩阵的算法如下:

【解答】

```
MGRAPH create_mgraph()
{
 int i, j, k;
 char b, t;
 MGRAPH mg;

 mg.kind=1;
 printf("请输入顶点数和边数:");
 scanf("% d,% d", &i, &j);
 mg.vexnum=i;
 mg.arcnum=j;
 for(i=0; i<mg.vexnum; i++)
    { printf(" 第 % d 个顶点信息:", i+1);
      getchar();
      scanf("% d", &mg.vexs[i]);}
for(i=0; i<mg.vexnum; i++)
    for(j=0; j<mg.vexnum; j++)
        mg.arcs[i][j]=0;
for(k=1; k<=mg.arcnum; k++)
    { printf("第 % d 条边的起始顶点编号和终止顶点编号: ",k);
      scanf("% d,% d",&i,&j);
      while(i<1 || i>mg.vexnum || j<1 || j>mg.vexnum)
        { printf(" 编号超出范围,重新输入:");
          scanf("% d,% d", &i, &j);}
      mg.arcs[i-1][j-1]=1;}
  return mg;
}
```

(2) 建立无向网邻接矩阵算法。

题目分析：参照主教材中建立有向网的邻接矩阵的算法，建立无向网邻接矩阵的算法如下：

【解答】

```
#define MAX 10000                    //设∞为 MAX

MGRAPH create_mgraph()
{
  int i, j, k,h;
  char b, t;
  MGRAPH mg;

  mg.kind=4;
  printf("请输入顶点数和边数:");
  scanf("% d,% d", &i, &j);
  mg.vexnum=i;
  mg.arcnum=j;
  for(i=0; i<mg.vexnum; i++)
      { printf(" 第 % d 个顶点信息:", i+1);
        getchar();
        scanf("% d", &mg.vexs[i]);}
  for(i=0; i<mg.vexnum; i++)
      for(j=0; j<mg.vexnum; j++)
          mg.arcs[i][j]=MAX;
  for(k=1; k<=mg.arcnum; k++)
      { printf("第 % d 条边的起始顶点编号和终止顶点编号: ",k);
        scanf("% d,% d",&i,&j);
        while(i<1 || i>mg.vexnum || j<1 || j>mg.vexnum)
          { printf(" 编号超出范围,重新输入:");
            scanf("% d,% d", &i, &j);}
        printf("此边的权值:");
        scanf("% d", &h);
        mg.arcs[i-1][j-1]=h;
        mg.arcs[j-1][i-1]=h;
        }
  return mg;
}
```

(3) 建立无向图邻接矩阵算法。

题目分析：参照主教材中建立有向网的邻接矩阵的算法，建立无向图邻接矩阵的算法如下：

【解答】

```
MGRAPH create_mgraph()
{
```

```
    int i, j, k;
    char b, t;
    MGRAPH mg;

    mg.kind=2;
    printf("请输入顶点数和边数:");
    scanf("% d,% d", &i, &j);
    mg.vexnum=i;
    mg.arcnum=j;
    for(i=0; i<mg.vexnum; i++)
        { printf(" 第 % d 个顶点信息:", i+1);
          getchar();
          scanf("% d", &mg.vexs[i]);}
    for(i=0; i<mg.vexnum; i++)
        for(j=0; j<mg.vexnum; j++)
            mg.arcs[i][j]=0;
    for(k=1; k<=mg.arcnum; k++)
        { printf("第 % d 条边的起始顶点编号和终止顶点编号: ",k);
          scanf("% d,% d",&i,&j);
          while(i<1 || i>mg.vexnum || j<1 || j>mg.vexnum)
            { printf(" 编号超出范围,重新输入:");
              scanf("% d,% d", &i, &j);}
          mg.arcs[i-1][j-1]=1;
          mg.arcs[j-1][i-1]=1;}
    return mg;
}
```

【习题 13】 已知一连通图,采用邻接矩阵结构存储,编写程序,生成连通图的邻接链表结构。

习题分析:图的邻接矩阵结构如下:

```
#define VEXTYPE int
#define ADJTYPE int
#define MAXLEN 40

typedef struct
{  otherdata …;   /* 图中边的信息,在下面的分析和讨论中忽略不考虑 */
   VEXTYPE vexs[MAXLEN];            /* 图中顶点的信息 */
   ADJTYPE arcs[MAXLEN][MAXLEN];    /* 邻接矩阵 */
   int vexnum, arcnum;              /* 顶点数和边数 */
   int kind;                        /* 图的类型 */
}MGRAPH;
```

连通图的邻接链表结构如下:

```
#define VEXTYPE int
```

```
#define MAXLEN 40

typedef struct node3          /*表结点结构*/
{  int adjvex;                /*存放与表头结点相邻接的顶点在数组中的序号*/
   struct node3 next;         /*指向与表头结点相邻接的下一个顶点的表结点*/
}EDGENODE;

typrdef struct
{ VEXTYPE vertex;             /*存放图中顶点的信息*/
  EDGENODE link;              /*指针指向对应的单链表中的表结点*/
} VEXNODE;

typedef struct
{ VEXNODE adjlist[MAXLEN]; /*邻接链表表头向量*/
  int vexnum, arcnum;         /*顶点数和边数*/
  int kind;                   /*图的类型*/
}ADJGRAPH;
```

【解答】

```
#include "datastru.h"
#include<stdio.h>
#include<malloc.h>

MGRAPH create_mgraph()
{/*建立图的邻接矩阵*/
    int i,j,k;
    MGRAPH mg;

    mg.kind=2;
    printf("\n\n输入顶点数和边数(用逗号隔开):");
    scanf("%d,%d",&i,&j);fflush(stdin);
    mg.vexnum=i;
    mg.arcnum=j;
    printf("\n\n");
    for(i=0;i<mg.vexnum;i++)
      {printf("输入顶点%d的值:",i+1);
       scanf("%d",&mg.vexs[i]); fflush(stdin);}
    for(i=0;i<mg.vexnum;i++)
      for(j=0;j<mg.vexnum;j++)
        mg.arcs[i][j]=0;
    for(k=1;k<=mg.arcnum;k++)
      {printf("输入第%d条边的起始顶点和终止顶点(用逗号隔开):",k);
       scanf("%d,%d",&i,&j);
       fflush(stdin);
```

```
        while(i<1||i>mg.vexnum||j<1||j>mg.vexnum)
          { printf("输入错,重新输入: ");
        scanf("% d,% d",&i,&j);}
         mg.arcs[i-1][j-1]=1;
         mg.arcs[j-1][i-1]=1;}
      return mg;
}

ADJGRAPH mg_to_adjg(MGRAPH mg)
{ int i,j,n;
  ADJGRAPH adjg;
  EDGENODE * p;
  n=mg.vexnum;
  adjg.vexnum=mg.vexnum;
  adjg.arcnum=mg.arcnum;
  for(i=0;i<n;i++)
    {  adjg.adjlist[i].vertex=mg.vexs[i];
       adjg.adjlist[i].link=NULL;
       for(j=0;j<n;j++)
    if(j!=i && mg.arcs[i][j]==1)
     {  p=malloc(sizeof(EDGENODE));
        p->adjvex=j;
        p->next=adjg.adjlist[i].link;
        adjg.adjlist[i].link=p;
     }
    }
  return adjg;
}

main()
{  MGRAPH mg;
   ADJGRAPH adjg;
   int i,j,n;
   EDGENODE * p;


   printf("\n建立图的邻接矩阵并转换为图的邻接链表\n");
   mg=create_mgraph();                    /* 建立图的邻接矩阵 */
   printf("\n\n图的邻接矩阵显示: \n"); /* 图的邻接矩阵显示 */
   n=mg.vexnum;
   for(i=0;i<n;i++)
   { for(j=0;j<n;j++)
       printf("<% d,% d>[% d] ",i+1,j+1,mg.arcs[i][j]);
       printf("\n");
```

```
    }
    adjg=mg_to_adjg(mg); /* 图的邻接矩阵结构转换为图的邻接链表 */
    printf("\n");
    printf("\n\n图的邻接链表显示：\n"); /* 图的邻接链表显示 */
    for(i=0;i<n;i++)
      { printf("<%d>%d",i,adjg.adjlist[i].vertex);
        p=adjg.adjlist[i].link;
        while(p!=NULL)
         { printf("->%d",p->adjvex);
      p=p->next;
        }
      printf("->NULL");
      printf("\n");
    }
}
```

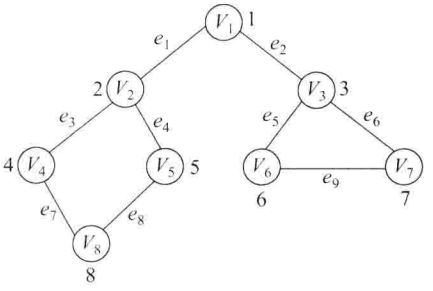


图 7-18　连通图编号

举例说明：以连通图 7-18 为例，顶点值设定为 $V_1 = 100$，$V_2 = 200$，$V_3 = 300$，$V_4 = 400$，$V_5 = 500$，$V_6 = 600$，$V_7 = 700$，$V_8 = 800$，8 个顶点和 9 条边的编号如图 7-18 所示。

输出结果如图 7-19 所示。

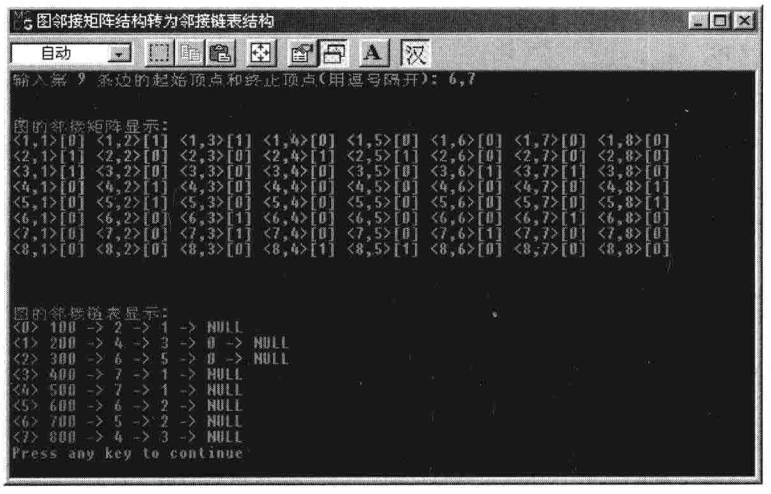


图 7-19　连通图邻接矩阵结构转换为图的邻接链表结构

【习题 14】 在以邻接链表为存储结构的连通图上，编写程序，实现连通图的广度优先遍历。

【解答】 图的邻接链表结构说明如下：

```
#define VEXTYPE int
#define MAXLEN 40
```

```
typedef struct node3            /* 表结点结构 */
{   int adjvex;                 /* 存放与表头结点相邻接的顶点在数组中的序号 */
    struct node3 next;          /* 指向与表头结点相邻接的下一个顶点的表结点 */
}EDGENODE;

typrdef struct
{ VEXTYPE vertex;               /* 存放图中顶点的信息 */
  EDGENODE link;                /* 指针指向对应的单链表中的表结点 */
} VEXNODE;

typedef struct
{ VEXNODE adjlist[MAXLEN];      /* 邻接链表表头向量 */
  int vexnum, arcnum;           /* 顶点数和边数 */
  int kind;                     /* 图的类型 */
}ADJGRAPH;
```

运行程序如下：

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

ADJGRAPH creat_adjgraph(){
EDGENODE * p;
int i, s, d;
ADJGRAPH adjg;

adjg.kind=2;
printf("\n\n输入顶点数和边数(用逗号隔开) : ");
scanf("% d,% d", &s, &d);fflush(stdin);
adjg.vexnum=s;                  /* 存放顶点数在 adjg.vexnum 中 */
adjg.arcnum=d;                  /* 存放边点数在 adjg.arcnum 中 */
printf("\n\n");
for(i=0; i<adjg.vexnum; i++)
  {printf("输入顶点 % d 的值 : ", i+1);
   scanf("% d", &adjg.adjlist[i].vertex);   /* 输入顶点的值 */
   fflush(stdin);
   adjg.adjlist[i].link=NULL;}
printf("\n\n");
for(i=0; i<adjg.arcnum; i++)
  {printf("输入第 % d 条边的起始顶点和终止顶点(用逗号隔开): ", i+1);
   scanf("% d,% d", &s, &d);    /* 输入边的起始顶点和终止顶点 */
   fflush(stdin);
   while(s<1 || s>adjg.vexnum || d<1 || d>adjg.vexnum)
       { printf("输入错,重新输入: ");
```

```
        scanf("%d,%d", &s, &d); }
    s--;
    d--;
    p=malloc(sizeof(EDGENODE));/*建立一个和边相关的结点*/
    p->adjvex=d;
    p->next=adjg.adjlist[s].link;/*挂到对应的单链表上*/
    adjg.adjlist[s].link=p;
    p=malloc(sizeof(EDGENODE));/*建立另一个和边相关的结点*/
    p->adjvex=s;
    p->next=adjg.adjlist[d].link;/*挂到对应的单链表上*/
    adjg.adjlist[d].link=p;}
return adjg;
}

void initlinkqueue(LINKQUEUE *q)
{/*初始化广度遍历中用到的链队列*/
    q->front=malloc(sizeof(LINKQLIST));
    (q->front)->next=NULL; q->rear=q->front;
}

int emptylinkqueue(LINKQUEUE *q)
{/*判队空? */
    int v;

    if(q->front==q->rear) v=1;
    else v=0;
    return v;
}


void enlinkqueue(LINKQUEUE *q, DATATYPE1 x)
{/*元素x入队列*/
    (q->rear)->next=malloc(sizeof(LINKQLIST));
    q->rear=(q->rear)->next; (q->rear)->data=x;
    (q->rear)->next=NULL;
}

DATATYPE1 dellinkqueue(LINKQUEUE *q)
{/*删除队头元素*/
    LINKQLIST *p;
    DATATYPE1 v;

    if(emptylinkqueue(q)) {printf("队列空.\n"); v=0;}
    else
```

```
        {p=(q->front)->next;
         (q->front)->next=p->next;
         if(p->next==NULL) q->rear=q->front;
         v=p->data;
         free(p);}
    return v;
}

void bfs(ADJGRAPH adjg, int vi)
{/*连通图的广度遍历算法:从 vi 顶点出发*/
int visited[MAXLEN];
int i, v;
EDGENODE *p;
LINKQUEUE que, *q;

q=&que;
initlinkqueue(q);
for(i=0; i<adjg.vexnum; i++)
    visited[i]=0;                    /*visited数组初始化,均置 0*/
visited[vi-1]=1;                     /*从编号为 vi 的顶点出发,visited数组对应置 1*/
printf("%d", adjg.adjlist[vi-1].vertex);  /*输出 vi 顶点*/
enlinkqueue(q,vi);                   /*vi 顶点入队列*/
while(!emptylinkqueue(q))            /*队列不空,继续进行遍历*/
  {v=dellinkqueue(q);                /*取队头元素放入 v 变量中*/
  v--;
  p=adjg.adjlist[v].link;
  while(p !=NULL)                    /*对应单链表不空,进行广度遍历*/
      {if(visited[p->adjvex]==0)
          {visited[p->adjvex]=1;
          printf("%d", adjg.adjlist[p->adjvex].vertex);
          enlinkqueue(q, (p->adjvex)+1);}   /*遍历到的结点编号入队列*/
      p=p->next;}
  }
}

main()
{  ADJGRAPH ag;
   int j;

  printf("\n连通图的广度遍历\n");
  ag=creat_adjgraph();               /*建立连通图的邻接链表结构*/
  printf("\n\n输入深度遍历起始顶点(1--%d) :",ag.vexnum);
  scanf("%d",&j);
  printf("\n\n广度遍历结果序列 :");
```

```
    bfs(ag,j);                          /* 连通图的广度遍历算法 */
    printf("\n\n");
}
```

举例说明：以连通图 7-18 为例。顶点值设定为 $V_1=100$，$V_2=200$，$V_3=300$，$V_4=400$，$V_5=500$，$V_6=600$，$V_7=700$，$V_8=800$，8 个顶点和 9 条边的编号如图 7-18 所示。

数据输入过程如图 7-20 所示。

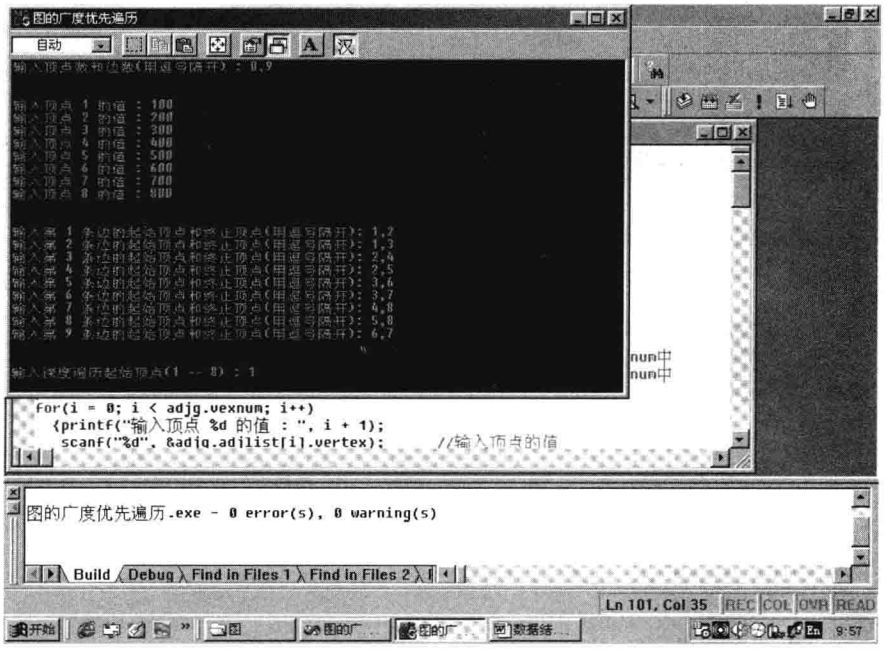


图 7-20　数据输入

输出结果如图 7-21 所示。

注意：此连通图邻接链表结构的建立过程是依赖于图 7-18 中 8 个顶点和 9 条边的编号。顶点和边的编号可以预先规定，编法不同，则数据输入的过程和输出的结果也会不同，但都是广度优先遍历算法的正确结果。

【习题 15】 编写程序，在以邻接链表为存储结构的无向图上，计算连通分量的个数及输出对每个连通分量的广度优先遍历结果。

题目分析：在习题 14 的基础上，稍加分析和设计，就可编写出计算无向图有几个连通分量及对每个连通分量的广度优先遍历结果的程序。也可编写判断无向图是否连通的程序。如果要遍历一个非连通的无向图，则需多次调用广度优先遍历算法 bfs，每一次都得到非连通图中的一个连通分量的广度优先遍历结果，调用 bfs 的次数就是非连通图的连通分量的个数。

【解答】

```
#include "stdio.h"
#include "malloc.h"
```

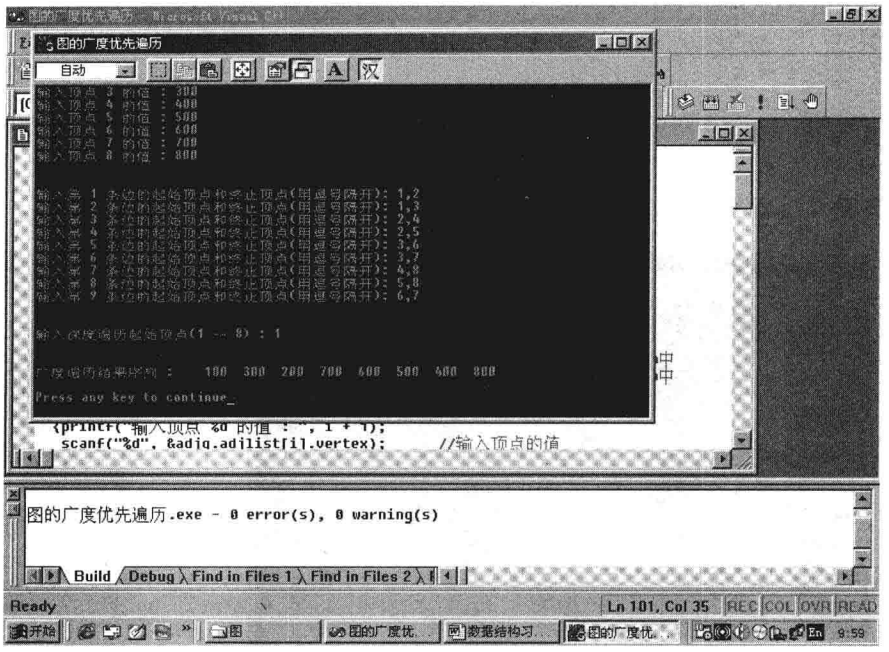


图 7-21　输出结果

```
#define VEXTYPE int
#define MAXLEN 40
#define DATATYPE1 int

typedef struct qnode
        {  DATATYPE1 data;
           struct qnode * next;
 }LINKQLIST;

typedef struct
{ LINKQLIST * front, * rear;
}LINKQUEUE;

typedef struct node3        /* 表结点结构 */
{  int adjvex;              /* 存放与表头结点相邻接的顶点在数组中的序号 */
   struct node3 * next;     /* 指向与表头结点相邻接的下一个顶点的表结点 */
}EDGENODE;

typedef struct
{ VEXTYPE vertex;           /* 存放图中顶点的信息 */
  EDGENODE * link;          /* 指针指向对应的单链表中的表结点 */
} VEXNODE;
```

```
typedef struct
{ VEXNODE adjlist[MAXLEN]; /* 邻接链表表头向量 */
  int vexnum, arcnum;      /* 顶点数和边数 */
  int kind;                /* 图的类型 */
}ADJGRAPH;

void enlinkqueue(LINKQUEUE * q,DATATYPE1 x)
{   (q->rear)->next=malloc(sizeof(LINKQLIST));
    q->rear=(q->rear)->next;
    (q->rear)->data=x;
    (q->rear)->next=NULL;
}

void initlinkqueue(LINKQUEUE * q)
{   q->front=malloc(sizeof(LINKQLIST));
    (q->front)->next=NULL;
    q->rear=q->front;
}

int emptylinkqueue(LINKQUEUE * q)
{   int v;
    if(q->front==q->rear)
        v=1;
    else v=0;
    return v;
}

DATATYPE1 dellinkqueue(LINKQUEUE * q)
{   LINKQLIST * p;
    DATATYPE1 v;

    if(emptylinkqueue(q))    { printf("Queue is empty.\n");
                               v=NULL;}
    else    { p=(q->front)->next;
              (q->front)->next=p->next;
              if(p->next==NULL)
              q->rear=q->front;
              v=p->data;
              free(p); }
    return v;
}

ADJGRAPH creat_adjgraph()
{ EDGENODE * p;
```

```
  int i, s, d;
  ADJGRAPH adjg;

adjg.kind= 1;
printf("请输入顶点数和边数:");
scanf("% d,% d", &s, &d);
adjg.vexnum=s;
adjg.arcnum=d;
for(i=0; i<adjg.vexnum; i++)
  {printf(" 第 % d 个顶点信息:", i+1);
   scanf("% d", &adjg.adjlist[i].vertex);
   adjg.adjlist[i].link=NULL;}
for(i=0; i<adjg.arcnum; i++)
  {printf("第 % d 条边的起始顶点编号和终止顶点编号:", i+1);
   scanf("% d,% d", &s, &d);
   while(s<1 || s>adjg.vexnum || d<1 || d>adjg.vexnum)
      { printf(" 编号超出范围,重新输入:");
        scanf("% d,% d", &s, &d); }
        s--;
        d--;
        p=malloc(sizeof(EDGENODE));
        p->adjvex=d;
        p->next=adjg.adjlist[s].link;
        adjg.adjlist[s].link=p;}
return adjg;
}
```

/* 在以邻接链表为存储结构的无向图上,计算连通分量的个数及输出对每个连通分量的广度优先遍历结果 */

```
void bfsg(ADJGRAPH adjg)
{   int visited[MAXLEN];
    int i, v,vi;
    int n=0;
    EDGENODE *p;
    LINKQUEUE que, *q;

    if(adjg.vexnum==0) return ;
    q=&que;
    initlinkqueue(q);
    for(i=0; i<adjg.vexnum; i++)
       visited[i]=0;
    vi=1;
    while(visited[vi-1]==0)
    {  visited[vi-1]=1;
```

```
        printf("第% d连通分量: % d",vi, adjg.adjlist[vi- 1].vertex);
        enlinkqueue(q, vi);
        while(!emptylinkqueue(q))
          {v=dellinkqueue(q);
           v--;
           p=adjg.adjlist[v].link;
           while(p !=NULL)
            {if(visited[p->adjvex]==0)
                {visited[p->adjvex]=1;
                 printf(" % d", adjg.adjlist[p->adjvex].vertex);
                 enlinkqueue(q,(p->adjvex)+1);}
             p=p->next;}}
        n++;
        printf("\n");
        i=1;
        while(visited[i-1]==1 && i<=adjg.vexnum )
            i++;
        if(i<=adjg.vexnum )
            vi=i;
    }
    printf("\n有% d个连通分量\n",n);
}

main()
{  ADJGRAPH G1;
   G1=creat_adjgraph();
   bfsg(G1);
}
```

举例说明：以非连通图 7-22 为例。顶点值设定为 $V_1=100$，$V_2=200$，$V_3=300$，$V_4=400$，$V_5=500$，$V_6=600$，$V_7=700$，$V_8=800$，$V_9=900$，9 个顶点和 9 条边的编号如图 7-22 所示。

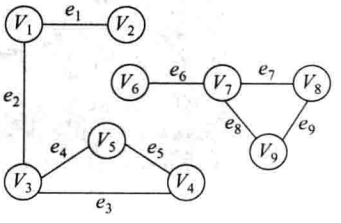


图 7-22　一个非连通图

数据输入过程如图 7-23 所示。

输出结果如图 7-24 所示。

【习题 16】 在以邻接链表为存储结构的连通图上，编写程序，实现连通图的深度优先遍历。

【解答】

```
#include <stdio.h>
#include "datastru.h"
#include <malloc.h>

int visited[MAXLEN];
```

图 7-23　数据输入

图 7-24　输出结果

```
ADJGRAPH creat_adjgraph()
{   EDGENODE * p;
    int i, s, d;
    ADJGRAPH adjg;

adjg.kind=2;
printf("\n\n 输入顶点数和边数(用逗号隔开):");
```

```
scanf("% d,% d", &s, &d);fflush(stdin);
adjg.vexnum=s;                          /* 存放顶点数在 adjg.vexnum 中 */
adjg.arcnum=d;                          /* 存放边点数在 adjg.arcnum 中 */
printf("\n\n");
for(i=0; i<adjg.vexnum; i++)            /* 输入顶点的值 */
  {printf("输入顶点 % d 的值 : ", i+1);
   scanf("% d", &adjg.adjlist[i].vertex);
   fflush(stdin);
   adjg.adjlist[i].link=NULL;}
   printf("\n");
for(i=0; i<adjg.arcnum; i++)
  {printf("输入第 % d 条边的起始顶点和终止顶点(用逗号隔开): ", i+1);
   scanf("% d,% d", &s, &d);            /* 输入边的起始顶点和终止顶点 */
   fflush(stdin);
   while(s<1 || s>adjg.vexnum || d<1 || d>adjg.vexnum)
      { printf("输入错,重新输入: ");
        scanf("% d,% d", &s, &d); }
   s--;
   d--;
   p=malloc(sizeof(EDGENODE));          /* 建立一个和边相关的结点 */
   p->adjvex=d;
   p->next=adjg.adjlist[s].link;        /* 挂到对应的单链表上 */
   adjg.adjlist[s].link=p;
   p=malloc(sizeof(EDGENODE));          /* 建立另一个和边相关的结点 */
   p->adjvex=s;
   p->next=adjg.adjlist[d].link;        /* 挂到对应的单链表上 */
   adjg.adjlist[d].link=p;}
return adjg;
}

void dfs(ADJGRAPH adjg, int v){
/* 连通图的深度遍历算法:从 v 顶点出发 */
EDGENODE *p;

visited[v-1]=1;                         /* 从编号为 v 的顶点出发,visited 数组对应置 1 */
v--;
printf(" % d", adjg.adjlist[v].vertex);   /* 输出 v 顶点 */
p=adjg.adjlist[v].link;                 /* 取单链表的第一个结点 */
while(p !=NULL)                         /* 对应单链表不空,进行遍历 */
   { if(visited[p->adjvex]==0)          /* 如果有未被访问过的结点 */
     dfs(adjg, (p->adjvex)+ 1);         /* 递归调用深度遍历算法 */
     p=p->next;}                        /* 取单链表的下一个结点 */
}
```

```
main()
{
   ADJGRAPH ag;
   int i;
   printf("\n 连通图的深度遍历\n");
   ag=creat_adjgraph();              /* 建立连通图的邻接链表结构 */
   for(i=0; i<ag.vexnum; i++)        /* visited 数组初始化,均置 0 */
     visited[i]=0;
   printf("\n\n 输入深度遍历起始顶点(1--% d) : ",ag.vexnum);
   scanf("% d",&i);
   printf("\n\n 深度遍结果序列 : ");
   dfs(ag,i);                        /* 连通图的深度遍历算法 */
   printf("\n\n");
}
```

举例说明：以连通图 7-18 为例。顶点值设定为 $V_1=100$，$V_2=200$，$V_3=300$，$V_4=400$，$V_5=500$，$V_6=600$，$V_7=700$，$V_8=800$，8 个顶点和 9 条边的编号如图 7-18 所示。

数据输入过程及输出结果如图 7-25 所示。

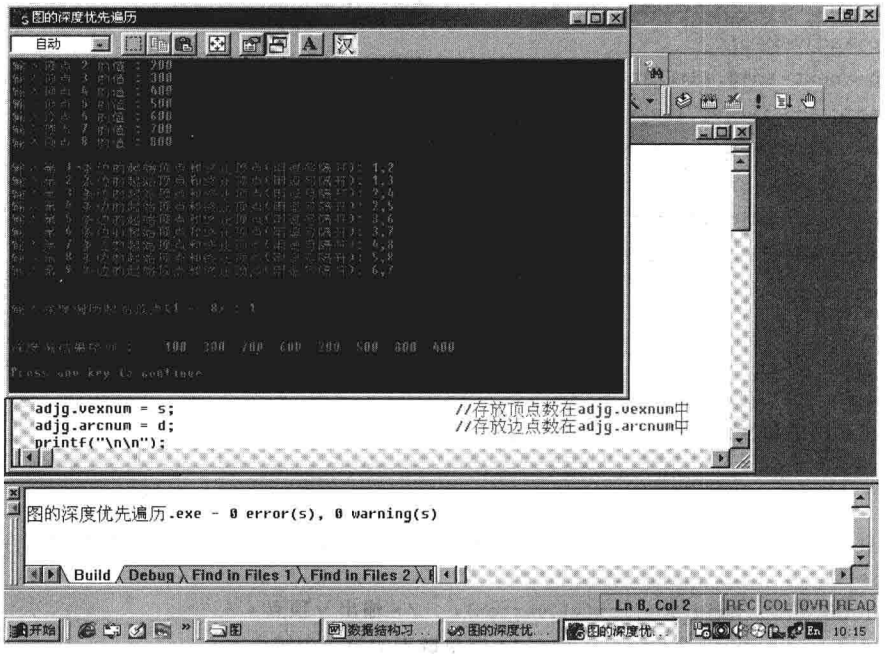


图 7-25　数据输入及输出结果

【习题 17】　编写程序，在以邻接链表为存储结构的无向图上，计算连通分量的个数及输出对每个连通分量的深度优先遍历结果。

题目分析：在习题 16 的基础上，稍加分析和设计，就可编写出计算无向图有几个连

通分量及对每个连通分量的深度优先遍历结果的程序。该程序也可判断无向图是否是连通的。如果要遍历一个非连通的无向图，则需多次调用广度优先遍历算法 dfs，每一次都得到非连通图中的一个连通分量的广度优先遍历结果，调用 dfs 的次数就是非连通图的连通分量的个数。

【解答】

```
#include "stdio.h"
#include "malloc.h"
#define VEXTYPE int
#define MAXLEN 40
#define DATATYPE1 int

typedef struct qnode
        {  DATATYPE1 data;
           struct qnode * next;
}LINKQLIST;

typedef struct
{ LINKQLIST * front, * rear;
}LINKQUEUE;

typedef struct node3                    /* 表结点结构 */
{  int adjvex;                          /* 存放与表头结点相邻接的顶点在数组中的序号 */
   struct node3 * next;                 /* 指向与表头结点相邻接的下一个顶点的表结点 */
}EDGENODE;

typedef struct
{ VEXTYPE vertex;                       /* 存放图中顶点的信息 */
  EDGENODE * link;                      /* 指针指向对应的单链表中的表结点 */
} VEXNODE;

typedef struct
{ VEXNODE adjlist[MAXLEN];              /* 邻接链表表头向量 */
  int vexnum, arcnum;                   /* 顶点数和边数 */
  int kind;                             /* 图的类型 */
}ADJGRAPH;

ADJGRAPH creat_adjgraph()
{ EDGENODE * p;
  int i, s, d;
  ADJGRAPH adjg;

adjg.kind=1;
printf("请输入顶点数和边数:");
```

```
scanf("% d,% d", &s, &d);
adjg.vexnum=s;
adjg.arcnum=d;
for(i=0; i<adjg.vexnum; i++)
  {printf(" 第 % d 个顶点信息:", i+1);
   scanf("% d", &adjg.adjlist[i].vertex);
   adjg.adjlist[i].link=NULL;}
for(i=0; i<adjg.arcnum; i++)
  {printf("第 % d 条边的起始顶点编号和终止顶点编号:", i+1);
   scanf("% d,% d", &s, &d);
   while(s<1 || s>adjg.vexnum || d<1 || d>adjg.vexnum)
      { printf(" 编号超出范围,重新输入:");
        scanf("% d,% d", &s, &d); }
        s--;
        d--;
        p=malloc(sizeof(EDGENODE));
        p->adjvex=d;
        p->next=adjg.adjlist[s].link;
        adjg.adjlist[s].link=p;}
return adjg;
}

void dfs(ADJGRAPH adjg,int v,int * visited)
{  EDGENODE * p;

   visited[v-1]=1;
   printf(" % d", adjg.adjlist[v-1].vertex);
   p=adjg.adjlist[v-1].link;
   while(p !=NULL)
     { if(visited[p->adjvex]==0)
             dfs(adjg, (p->adjvex)+1,visited);
       p=p->next;}
}

/* 在以邻接链表为存储结构的无向图上,
计算连通分量的个数及输出对每个连通分量的深度优先遍历结果 */
void dfsg(ADJGRAPH adjg)
{  int visited[MAXLEN];
   int i,vi,n=0;

   if(adjg.vexnum==0) return ;
   for(i=0; i<adjg.vexnum; i++) visited[i]=0;
   vi=1;
   while(visited[vi-1]==0)
```

```
    {  printf("第%d连通分量:",n+1);
       dfs(adjg,vi,visited);
       n++;
       printf("\n");
       i=1;
       while(visited[i-1]==1 && i<=adjg.vexnum ) i++;
       if(i<=adjg.vexnum ) vi=i;
    }
    printf("\n有%d个连通分量\n",n);
}

main()
{  ADJGRAPH G1;
   G1=creat_adjgraph();
   dfsg(G1);
}
```

举例说明：以非连通图 7-22 为例，输出结果如图 7-26 所示。

图 7-26　输出结果

【习题 18】 编写程序，在以邻接链表为存储结构的有向图上，实现有向图的拓扑排序。

【解答】

```
#include<stdio.h>
#include<malloc.h>
#include "datastru.h"

ADJGRAPH creat_adjgraph()
```

```
{/* 建立有向图的邻接链表结构 */
  EDGENODE *p;
  int i,s,d;
  ADJGRAPH adjg;

  adjg.kind=1;
  printf("\n\n输入顶点数和边数(用逗号隔开):");
  scanf("%d,%d",&s,&d);fflush(stdin);
  adjg.vexnum=s;                          /* 存放顶点数在 adjg.vexnum 中 */
  adjg.arcnum=d;                          /* 存放边点数在 adjg.arcnum 中 */
  printf("\n\n");
  for(i=0; i<adjg.vexnum; i++)            /* 邻接链表顶点初始化 */
    {printf("输入顶点 %d 的值:", i+1);
     scanf("%d",&adjg.adjlist[i].vertex); /* 输入顶点的值 */
     fflush(stdin);
     adjg.adjlist[i].link=NULL;
     adjg.adjlist[i].id=0;}
     printf("\n\n");
  for(i=0; i<adjg.arcnum; i++)
    {printf("输入第 %d 条有向弧的起始顶点和终止顶点(用逗号隔开):", i+1);
     scanf("%d,%d",&s,&d);                /* 输入弧的起始顶点和终止顶点 */
     fflush(stdin);
     while(s<1 || s>adjg.vexnum || d<1 || d>adjg.vexnum)
       {printf("输入错,重新输入:");
        scanf("%d,%d",&s,&d);}
     s--;
     d--;
     p=malloc(sizeof(EDGENODE));          /* 每条弧对应生成一个结点 */
     p->adjvex=d;
     p->next=adjg.adjlist[s].link;        /* 结点插入对应的单链表上 */
     adjg.adjlist[s].link=p;
     adjg.adjlist[d].id++;}               /* 弧的终端顶点入度加 1 */
return adjg;
}

void topsort(ADJGRAPH ag)
{  int i, j, k, m, n, top;
   EDGENODE *p;

   n=ag.vexnum;
   top=-1;                                /* 拓扑排序中用到的栈初始化 */
   for(i=0; i<n; i++)
     if(ag.adjlist[i].id==0) /* 入度为 0 的结点压入一链栈,top 指向栈顶结点 */
       { ag.adjlist[i].id=top;
```

```
        top=i;}
    m=0;
    printf("\n\n有向图拓扑排序结果 : ");
    while(top !=- 1)                    /*栈不空,进行拓扑排序*/
      {j=top;                           /*取栈顶元素*/
       top=ag.adjlist[top].id;          /*删除一个栈顶元素*/
       printf(" %d", ag.adjlist[j].vertex);  /*输出一个拓扑排序结点即栈顶元素*/
       m++;                             /*拓扑排序结点计数加 1*/
       p=ag.adjlist[j].link;
       while(p !=NULL)                  /*如果拓扑排序结点有出度*/
        {k=p->adjvex;
         ag.adjlist[k].id--;            /*删除相关的弧,即弧终点结点的入度减 1*/
         if(ag.adjlist[k].id==0)        /*出现新的入度为 0 的顶点,将其入栈*/
         {ag.adjlist[k].id=top;
          top=k;}
         p=p->next;}}
    if(m<n)
      printf("\n\n有向图中有环路 !\n\n");
                                  /*拓扑排序过程中输出的顶点数<有向图的顶点数*/
}

main()
{ ADJGRAPH ag;

  printf("\n有向图的拓扑排序\n");
  ag=creat_adjgraph();              /*建立有向图的邻接链表结构*/
  topsort(ag);                      /*进行拓扑排序*/
  printf("\n\n");
}
```

举例说明：设有向图，顶点值设定为 $V_1=100$，$V_2=200$，$V_3=300$，$V_4=400$，$V_5=500$，$V_6=600$，6 个顶点和 8 条边的编号为图 7-27 所示。

数据输入过程及输出结果如图 7-28 所示。

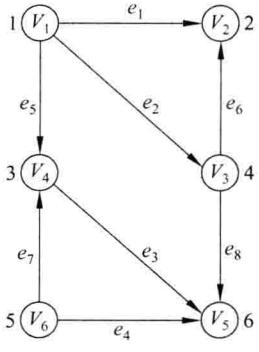


图 7-27 有向图

【习题 19】 在以邻接矩阵为存储结构的有向网上，求指定单源点到其他顶点的最短路径。

【解答】

```
#include "datastru.h"
#include<stdio.h>
#include<malloc.h>
#define MAX 10000

MGRAPH create_mgraph(){
```

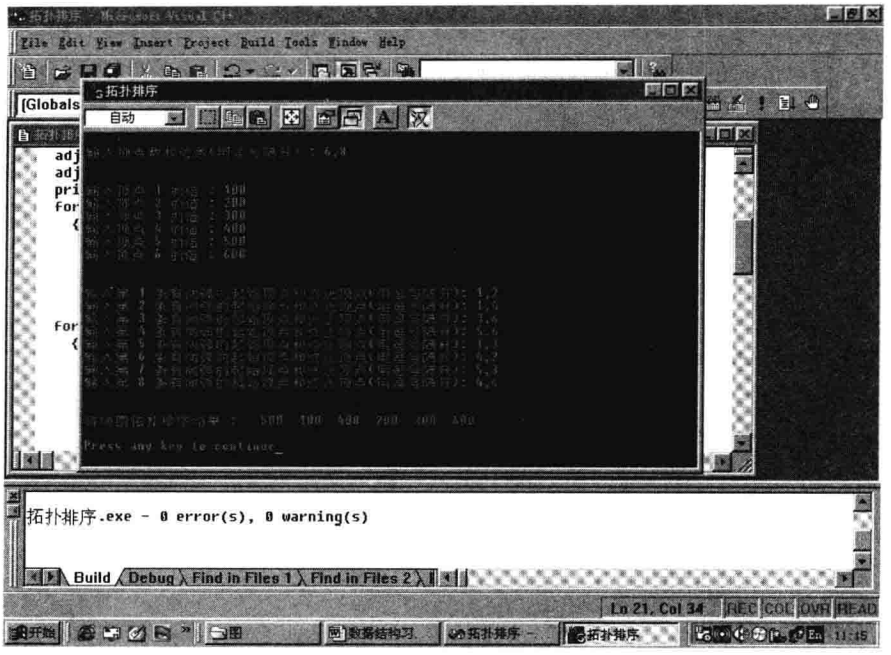


图 7-28　数据输入及输出结果

```
/*建立有向图的邻接矩阵结构*/
int i, j, k, h;
 MGRAPH mg;

  mg.kind=3;
  printf("\n\n输入顶点数和边数(用逗号隔开):");
  scanf("%d,%d", &i,&j);
  mg.vexnum=i;                              /*存放顶点数在mg.vexnum中 */
  mg.arcnum=j;                              /*存放边点数在mg.arcnum中*/
  fflush(stdin);
  for(i=0; i<mg.vexnum; i++)
    {printf("输入顶点%d的值:",i+1);  /*输入顶点的值*/
     scanf("%d", &mg.vexs[i]);
     fflush(stdin);}
  for(i=0; i<mg.vexnum; i++)                /*邻接矩阵初始化*/
    for(j=0; j<mg.vexnum; j++)
   mg.arcs[i][j]=MAX;
  for(k=1; k<=mg.arcnum; k++)
    {printf("输入第%d条边的起始顶点和终止顶点(用逗号隔开):",k);
     scanf("%d,%d",&i,&j);                  /*输入弧的起始顶点和终止顶点*/
     fflush(stdin);
     while(i<1 || i>mg.vexnum || j<1 || j>mg.vexnum)
    {printf("输入错,重新输入:");
```

```
        scanf("% d,% d", &i, &j);}
        printf("输入此边权值 : ");          /* 输入弧上之权值 */
        scanf("% d", &h);
        mg.arcs[i-1][j-1]=h;}
    return mg;
}

main()
{
    MGRAPH mg;
    int cost[MAXLEN][MAXLEN];
    int path[MAXLEN], s[MAXLEN];
    int dist[MAXLEN];
    int i, j, n, v0, min, u;

    printf("\n求有向图单源点最短路径\n");
    mg=create_mgraph();                  /* 建立有向图的邻接矩阵结构 */
    printf("\n\n起始顶点为 : ");          /* 有向图中顶点的编号从 1 编起 */
    scanf("% d", &v0);
    v0--;
    n=mg.vexnum;
    for(i=0; i<n; i++)                   /* cost 矩阵初始化 */
      {for(j=0; j<n; j++)
         cost[i][j]=mg.arcs[i][j];
         cost[i][i]=0;}
    for(i=0; i<n; i++)
        {dist[i]=cost[v0][i];            /* dist 数组初始化 */
        if(dist[i]<MAX && dist[i]>0)     /* path 数组初始化 */
          path[i]=v0;}
    for(i=0; i<n; i++)                   /* s 数组初始化 */
        s[i]=0;
    s[v0]=1;
    for(i=0; i<n; i++)                   /* 按最短路径递增算法计算 */
        {  min=MAX ;
        u=v0;
        for(j=0; j<n; j++)
            if(s[j]==0 && dist[j]<min)
              {min=dist[j];
               u=j;}
        s[u]=1;                          /* u 顶点是求得最短路径的顶点编号 */
        for(j=0; j<n; j++)
            if(s[j]==0 && dist[u]+cost[u][j]<dist[j])/* 调整 dist */
              {dist[j]=dist[u]+cost[u][j];
            path[j]=u;}                  /* path 记录了路径经过的顶点 */
```

```
        }
    for(i=0; i<n; i++)                    /* 打印结果 */
        if(s[i]==1)
            {u=i;
            while(u !=v0)
                {printf("% d<-", u+1);
                 u=path[u];}
            printf("% d ", u+1);
            printf(" d=% d\n", dist[i]); /* 有路径 */
           }
        else
            printf("% d<-% d d=MAX\n ", i+1, v0+1);/* 无路径 */
printf("\n\n");
}
```

举例说明：设有向网，顶点值设定为 $V_1=100$，$V_2=200$，$V_3=300$，$V_4=400$，$V_5=500$，$V_6=600$，$V_7=700$，7 个顶点和 11 条边的编号为图 7-29 所示。

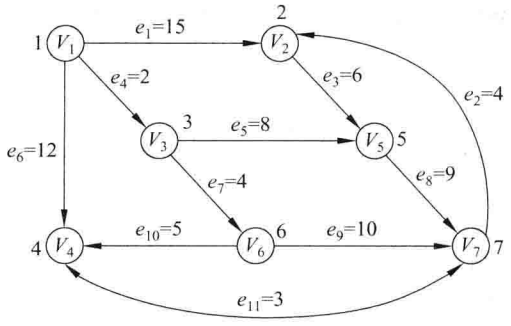


图 7-29　有向网

数据输入过程如图 7-30 所示，输出结果如图 7-31 所示。

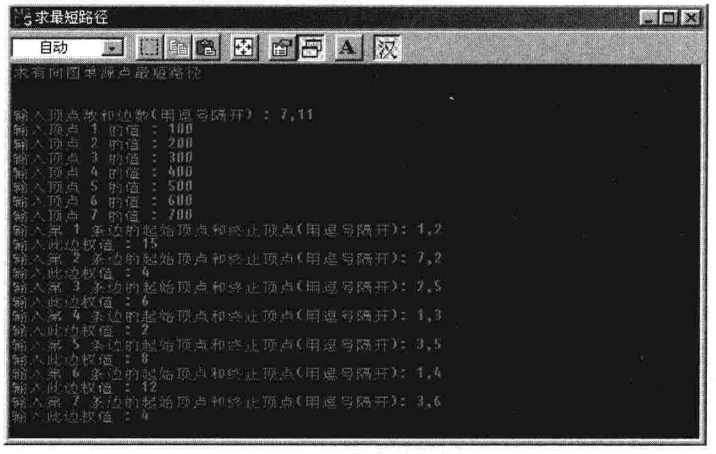


图 7-30　数据输入

图 7-31 输出结果

7.2 实训练习题

【习题 20】 有一个带权无向图如图 7-32 所示，其邻接矩阵的数组表示如下，试完成下列要求：

(1) 写出在该图上从顶点 1 出发进行深度优先遍历的顶点序列。

(2) 画出该图的带权邻接表。

(3) 画出按普里姆算法构造最小生成树(森林)的示意图。

1	0	3	5	∞	∞	9	∞
2	3	0	6	∞	∞	∞	10
3	5	6	0	∞	∞	∞	4
4	∞	∞	∞	0	3	∞	∞
5	∞	∞	∞	3	0	∞	∞
6	∞	∞	∞	6	5	∞	∞
7	9	∞	∞	∞	∞	0	7
8	∞	10	4	∞	∞	7	0

图 7-32 带权无向图的邻接矩阵

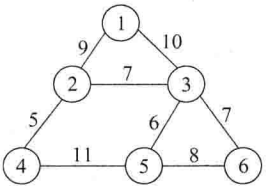


图 7-33 带权无向图

【习题 21】 给定如图 7-33 所示的带权无向图。

(1) 画出该图的邻接表存储结构。

(2) 根据该图的邻接表存储结构，从顶点 1 出发，调用 DFS 和 BFS 算法遍历该图，写出遍历序列。

(3) 画出该图的邻接矩阵。

【习题 22】 一个有向图 G 的邻接表存储如图 7-34 所示，要求：

(1) 画出其邻接矩阵存储。

(2) 写出图的所有强连通分量。

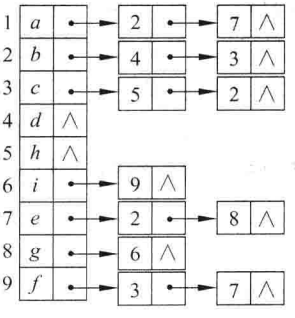


图 7-34 一个有向图 G 的邻接表存储

【习题 23】 单项选择题。

1. 在一个无向图中,所有顶点的度之和等于边数的倍。

A. 1/2　　B. 1　　C. 2　　D. 4

2. 具有 6 个顶点的无向图至少应有_____条边才能确保是一个连通图。

A. 5　　B. 6　　C. 7　　D. 8

3. 对于一个具有 n 个顶点的无向图,若采用邻接矩阵表示,则该矩阵大小是_____。

A. n　　B. $(n-1)^2$　　C. $n-1$　　D. n^2

4. 如果从无向图的任一顶点出发进行一次深度优先搜索即可访问所有顶点,则该图一定是_____。

A. 完全图　　B. 连通图　　C. 有回路　　D. 一棵树

5. 任何一个无向连通图_____最小生成树。

A. 只有一棵　　B. 有一棵或多棵

C. 一定有多棵　　D. 可能不存在

6. 对于含有 n 个顶点的带权连通图,它的最小生成树是指图中任意一个_____。

A. 由 $n-1$ 条权值最小的边构成的子图

B. 由 $n-1$ 条权值之和最小的边构成的子图

C. 由 $n-l$ 条权值之和最小的边构成的连通子图

D. 由 n 个顶点构成的边的权值之和最小的连通子图

【习题 24】 判断以下叙述的正确性。

(1) 邻接矩阵只存储了边的信息,没有存储顶点的信息。

(2) 如果表示图的邻接矩阵是对称矩阵,则该图一定是无向图。

(3) 对 n 个顶点的连通图 G 来说,如果其中的某个子图有 n 个顶点、$n-1$ 条边,则该子图一定是 G 的生成树。

(4) 从 n 个顶点的连通图中选取 $n-1$ 条权值最小的边,即可构成最小生成树。

(5) 如果表示某个图的邻接矩阵是不对称矩阵,则该图一定是有向图。

(6) 无向图中的极大连通子图称为连通分量。

(7) 连通图的广度优先搜索中一般要采用队列来暂存刚访问过的顶点。

(8) 图的深度优先搜索中一般要采用栈来暂存刚访问过的顶点。

【习题 25】 已知一个无向图,采用邻接链表结构存储,编写算法。

(1) 计算图中有多少条边?

(2) 判断任意两个顶点 i 和 j 之间是否有边相连?

(3) 计算任意一个顶点的度是多少?

【习题 26】 编写算法,在无向图的邻接链表结构上,生成无向图的邻接矩阵结构。

查　找

查找又称检索，是数据处理中使用频繁的一种重要操作。查找表上的基本操作有建表、查找、读取表中元素(记录)、插入和删除。本章给出在顺序表上查找、有序表上查找、散列表上查找的算法和程序。最后给出一个二叉排序树的建立、查找、插入、删除、显示的综合练习程序。

8.1　习题解析

【习题 1】 单项选择题。

1. 在二叉排序树中，凡是新插入的结点，都是没有_____的。

A. 孩子　　B. 关键字　　C. 左孩子　　D. 右孩子

【解答】 正确答案 A。在二叉排序树中插入的结点均为叶子结点，没有孩子。

2. 只有在顺序存储结构上才能实现的查找方法是_____法。

A. 顺序查找　　B. 二分查找　　C. 树形查找　　D. 散列查找

【解答】 B。

3. 有一个长度为 12 的有序表，按二分查找法对该表进行查找，在表内各元素等概率情况下，查找成功所需的平均比较次数为_____。

A. 35/12　　B. 37/12　　C. 39/12　　D. 43/12

【解答】 B。

4. 有一个有序表 $R[1\cdots13]=(1,3,9,12,32,41,45,62,75,77,82,95,100)$，当用二分查找法查找值为 82 的结点时，经_____次比较后查找成功。

A. 1　　B. 2　　C. 4　　D. 8

【解答】 $72=13$，$R[11]=82$，第 1 次与 $R[(1+13)/2=7]=45$ 比较，第 2 次与 $R[(8+13)/2=10]=77$ 比较，第 3 次只与 $[(11+13)/2=12]=95$ 比较，第 4 次只与 $[(10+12)/2=11]=85$ 比较，成功，总共比较 4 次。正确答案 C。

5. 如图 8-15(a)所示的一棵二叉排序树，其不成功的平均查找长度是_____。

A. 21/7　　B. 28/7　　C. 15/6　　D. 21/6

【解答】 如图 8-15(b)所示,不带数字的结点均为查找不成功的外部结点。在查找失败时,其比较过程是经历了一条从判定树根到某个外部结点的路径,所需的关键字比较次数是该路径上内部结点的总数。其平均查找长度为(2×2+3×3+4×2)/7=21/7。本题正确答案为 A。

6. 采用分块查找时,若线性表中共有 625 个元素,查找每个元素的概率相同,假设采用顺序查找来确定结点所在的块,则每块分为_____个结点最佳。

A. 9　　B. 25　　C. 6　　D. 625

【解答】 分块查找时最佳块数=$\sqrt{625}$=25。本题正确答案为 B。

7. 设散列表长 $m=14$,散列函数 H(key)=key mod 11。表中已有 4 个结点 addr(15)=4,addr(38)=5,addr(61)=6,addr(84)=7,其余地址为空。如用二次探测再散列法处理冲突,则关键字为 49 的结点的地址是_____。

A. 8　　B. 3　　C. 5　　D. 9

【解答】 addr(49)=49%11=5 冲突,hl=(5+1')%11=6 仍冲突,tlz=(5+2")%11=9 不冲突。所以本题正确答案为 D。

8. 散列表的平均查找长度_____。

A. 与处理冲突方法有关而与表的长度无关

B. 与处理冲突方法无关而与表的长度有关

C. 与处理冲突方法有关且与表的长度有关

D. 与处理冲突方法无关且与表的长度无关

【解答】 散列表的平均查找长度与处理冲突方法有关,与表的装填因子有关,但与表的长度无关。本题正确答案为 A。

【习题 2】 判断以下叙述的正确性。

(1) 用顺序表和单链表表示的有序表均可使用二分查找方法来提高查找速度。

(2) 有 n 个数存放在一维数组 $A[1\cdots n]$中,在进行顺序查找时,这 n 个数的排列有序或无序其平均查找长度不同。

(3) 二叉排序树的任意一棵子树中,关键字最小的结点必无左孩子,关键字最大的结点必无右孩子。

(4) 散列表的查找效率主要取决于散列表造表时选取的散列函数和处理冲突的方法。

(5) 二叉排序树上的查找都是从根结点开始的,查找失败一定落在叶子上。

【解答】

(1) 错误。链表表示的有序表不能用二分查找法查找。

(2) 错误。因顺序查找既适合于有序表也适合于无序表。对这两种表,若对每个元素的查找概率相等,则顺序查找的 ASL 相同,并且都是中;对于查找概率不同的情况,则按查找概率由大到小排列的无序表的 ASL 要比有序表的 ASL 小。

(3) 正确。

(4) 正确。

(5) 错误。若非叶子结点无左子树或无右子树时，也都可以是查找失败的外部结点。

【习题 3】 可以生成如图 8-1 所示二叉排序树的关键字初始排列有很多种，请写出其中的 5 种。

【解答】 可以生成如图二叉排序树的关键字初始排列有：

5 4 7 2 1 3 6　　5 7 6 4 2 3 1　　5 7 4 2 6 1 3

5 7 4 6 2 1 3　　5 4 2 7 6 1 3

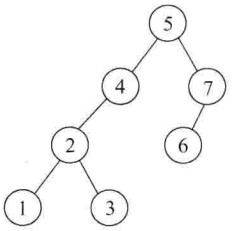


图 8-1　二叉排序树

【习题 4】 已知一任意关键字序列 (19, 14, 22, 01, 66, 21, 83, 27, 56, 13, 10, 50)。

(1) 按元素在序列中的次序建立一棵初始为空的二叉排序树，画出完成后的二叉排序树。

(2) 在(1)的基础上插入结点 24 后，画出对应的二叉排序树。

(3) 在(2)的基础上删除结点 66，画出对应的二叉排序树。

【解答】

(1)、(2)、(3)的二叉排序树分别如图 8-2(a)、(b)、(c)所示。

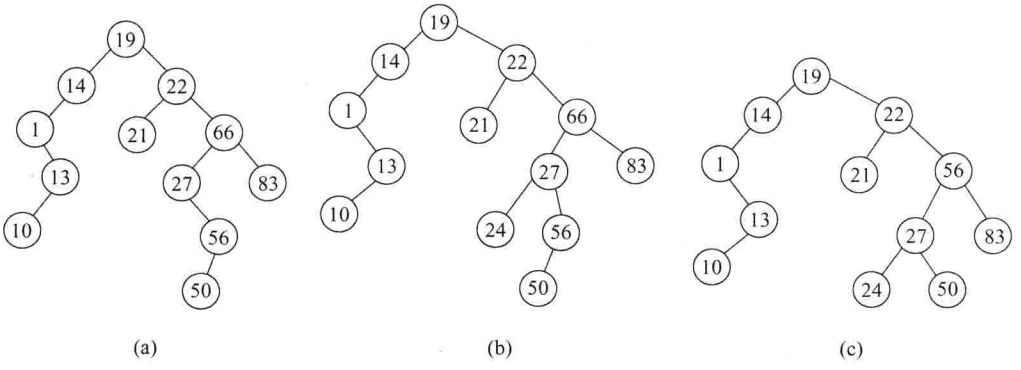


图 8-2　二叉排序树

【习题 5】 在地址空间为 0～16 的散列区域中，对下面关键字序列构造两个散列表：

(Jan, Feb, Mar, Apr, May, June, July, Aug, Sep, Oct, Nov, Dec)

(1) 用闭散列表线性探测法处理冲突；

(2) 用开散列表链地址法处理冲突。

设函数为 $H(x)=i/2$，其中 i 为关键字的第一个字母在字母表中的序号。画出两个散列表，并分别求这两个散列表在等概率情况下查找成功的平均查找长度及查找不成功的平均查找长度。

【解答】

(1) 用闭散列表线性探测法处理冲突，如图 8-3 所示。

查找成功的平均查找长度＝(1＋2＋1＋1＋1＋1＋2＋4＋5＋2＋5＋6)/12＝31/12

查找不成功的平均查找长度＝(5＋4＋3＋2＋1＋9＋8＋7＋6＋5＋4＋3＋2＋1＋1＋1＋1)/17＝63/17

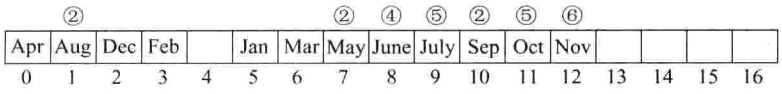


图 8-3　用闭散列表线性探测法处理冲突

（2）用开散列表链地址法处理冲突，如图 8-4 所示。

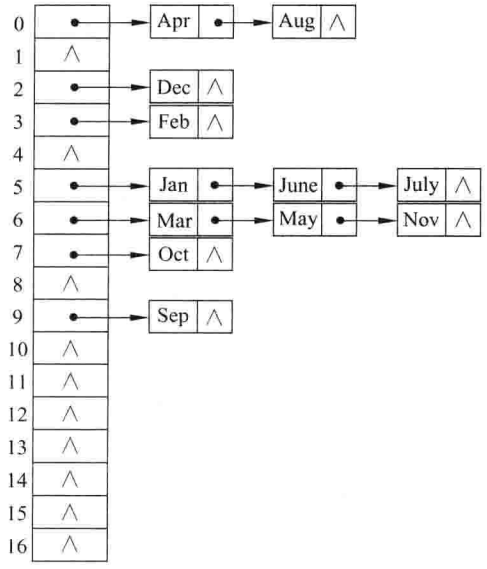


图 8-4　用开散列表链地址法处理冲突

查找成功的平均查找长度＝(1＋2＋1＋1＋1＋2＋3＋1＋2＋3＋1＋1)/12＝19/12

查找不成功的平均查找长度＝(3＋1＋2＋2＋1＋4＋4＋2＋1＋2＋1＋1＋
1＋1＋1＋1＋1)/17＝29/17

【习题 6】 编写程序，实现在顺序表上顺序查找元素。

题目分析：查找表的存储结构为顺序表，对表中记录存放的先后次序没有任何要求。建立一个顺序表，记录从下标为 1 的单元开始放入，下标为 0 的单元起监视哨作用。输入待查记录的关键字进行查找。为了简化算法，记录只含一个整型量关键字字段，记录的其余数据部分忽略。

结构说明：查找表的存储结构如下：

```
#define KEYTYPE int
#define MAXSIZE 100

typedef struct
{   KEYTYPE key;
}SSELEMENT;

typedef struct
```

```
{  SSELEMENT r[MAXSIZE];
int len;
}SSTABLE;
```

【解答】

```
#include "datastru.h"
#include <stdio.h>

int seq_search(KEYTYPE k, SSTABLE * st)
{/* 顺序表上查找元素 */
  int j;

  j=st->len;                          /* 顺序表元素个数 */
  st->r[0].key=k;                     /* st->r[0]单元作为监视哨 */
  while(st->r[j].key !=k) j--;        /* 顺序表从后向前查找 */
  return j;                           /* j=0, 找不到;j<>0 找到 */
}

main()
{  SSTABLE a;
   int i, j, k;

printf("请输入顺序表元素,元素为整型量,用空格分开,-99为结束标志:");
j=0; k=1; i=0; scanf("% d",&i);
while (i !=-99) { j++; a.r[k].key=i; k++; scanf("% d",&i); }/* 输入顺序表元素 */
a.len=j;
printf("\n顺序表元素列表显示:");
for (i=1; i<=a.len; i++)
    printf("% d ",a.r[i].key);
printf("\n");
printf("\n输入待查元素关键字 : ");
scanf("% d",&i);
k= seq_search(i, &a);
if (k==0) printf("表中待查元素不存在\n\n");
else printf("表中待查元素存在\n\n");
}
```

【习题 7】 编写程序,实现在有序表上二分法查找元素。

题目分析:查找表的存储结构为有序表,即表中记录按关键字大小排序存放。本例建立一个有序表,记录从下标为 1 的单元开始放入。输入待查记录的关键字进行查找。为了简化算法,记录只含一个整型量关键字字段,记录的其余数据部分忽略不考虑。此程序中要求对整型量关键字数据的输入按从小到大顺序输入。

结构说明:有序表的存储结构如下:

```
#define KEYTYPE int
```

```
#define MAXSIZE 100

typedef struct
{  KEYTYPE key;
}SSELEMENT;

typedef struct
{  SSELEMENT r[MAXSIZE];
   int len;
}SSTABLE;
```

【解答 1】

```
#include "datastru.h"
#include <stdio.h>

int search_bin(KEYTYPE k, SSTABLE * st)
{ /* 有序表上二分法查找 */
   int low, high, mid;

   low=1; high=st->len;      /* low=1 表示元素从下标为 1 的单元放起 */
                             /* high=st->len 表示最后一个元素所在下标 */
   while(low<=high)          /* low<=high 为继续查找的条件 */
      { mid=(low+high)/2;
        if(k==st->r[mid].key) return mid;   /* k==st->r[mid].key 表示查找成功 */
        else if(k<st->r[mid].key)           /* 否则继续二分查找 */
                high=mid-1;
             else low=mid+1; }
   return 0;                                /* 查找不成功,返回 0 */
}

main( )
{  SSTABLE a;
   int i, j, k;

printf("请输入有序表元素,元素为整型量(从小到大输入),用空格分开,-99 为结束标
志 :");
j=0; k=1; i=0; scanf("% d",&i);
while (i !=-99) { j++; a.r[k].key=i; k++; scanf("% d",&i); }/* 输入有序表元素 */
a.len=j;
printf("\n 有序表元素列表显示 :");
for (i=1; i<=a.len; i++)
    printf("% d ",a.r[i]);
printf("\n");
printf("\n 输入待查元素关键字 : ");
```

```
scanf("% d",&i);
k=search_bin(i, &a);
if (k==0) printf("表中待查元素不存在\n\n");
else printf("表中待查元素存在\n\n");
}
```

【解答 2】

```
#include "datastru.h"
#include <stdio.h>
SSTABLE a;

int search_bin(KEYTYPE k, int low, int high)
{  /* 有序表上二分法查找,递归算法 */
   int mid;

   mid=-1;
   if(low<=high)            /* low 表示当前表中第 1 个元素所在下标 */
                            /* high 表示当前表中最后一个元素所在下标 */
     {
     mid=(low+high)/2;      /* mid 表示当前表中中间一个元素所在下标 */
     if(a.r[mid].key<k) mid=search_bin(k, mid+1,high);  /* 二分法递归查找 */
     else if(a.r[mid].key>k) mid=search_bin(k,low,high-1);
     }
return mid;                 /* mid=-1 查找不成功;mid<>-1 查找成功 */
}

main( )
{ int i, j, k;

printf("请输入有序表元素,元素为整型量(从小到大输入),用空格分开,-99 为结束标志:");
j=0; k=1; i=0; scanf("% d",&i);
while (i !=-99) { j++; a.r[k].key=i; k++; scanf("% d",&i); }/* 输入有序表元素 */
a.len=j;
printf("\n 有序表元素列表显示:");
for (i=1; i<=a.len; i++)
    printf("% d ",a.r[i].key);
printf("\n");
printf("\n 输入待查元素关键字 : ");
scanf("% d",&i);
k=search_bin(i, 1, a.len);
if (k==-1) printf("表中待查元素不存在\n\n");
else printf("表中待查元素存在\n\n");
}
```

【习题 8】 编写程序,实现在开散列表上查找元素。

题目分析：建立一个开散列表，散列函数为 H(key)=key MOD 13。输入待查记录的关键字在开散列表上进行查找。为了简化算法，表中记录只含一个正整型量关键字字段，记录的其余数据部分忽略。

结构说明：

```
#define KEYTYPE int
#define MAXSIZE 100

typedef struct node5
{
  KEYTYPE key;
  struct node5 * next;
}CHAINHASH;

CHAINHASH * HTC[MAXSIZE];
```

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>

void creat_chain_hash(CHAINHASH * HTC[])
{/* 建立开散列表 */
  CHAINHASH * p;
  int i, j;

  i=0; scanf("% d",&i);      /* 输入开散列表元素关键字值 */
  while (i !=-99) {
    j=i % 13;                /* 散列函数：ADD(rec(key))=key MOD 13 */
    p= (CHAINHASH * ) malloc(sizeof(CHAINHASH)); /* 生成结点，挂入开散列表中 */
        p->next=HTC[j];
        p->key=i;
        HTC[j]=p;
        scanf("% d",&i); }  /* 输入开散列表元素关键字值 */
}

void print_chain_hash(CHAINHASH * HTC[])
{/* 显示开散列表 */
 int i;
 CHAINHASH * p;

for(i=0; i<13; i++) {
    if(HTC[i]==NULL) printf(" % 3d | ^ \n",i);
    else { p=HTC[i];
```

```
            printf(" % 3d |->",i);
            while(p !=NULL) {printf("% 5d->",p->key); p=p->next; }
            printf("\n");
        }
    }
    }

    int search_chain_hash(CHAINHASH * HTC[], int k)
    {/* 开散列表中查找元素 */
        CHAINHASH * p;
        int j;

        j=k % 13;          /* 散列函数: ADD(rec(key))=key MOD 13 */
        p=HTC[j];
        if(p !=NULL)       /* 开散列表中查找元素 */
            { while((p->key !=k)&&(p->next !=NULL)) p=p->next;
              if (p->key==k) return 1;       /* 查找成功,返回 1 */
              else return 0;                 /* 查找不成功,返回 0 */
            }
        else return 0;
    }

    main()
      { CHAINHASH * HTC[MAXSIZE];
        int i, k;

        printf("\n 建立开散列表\n\n");
        for (i=0; i<MAXSIZE; i++)
           HTC[i]=NULL;
        printf("请输入开散列表元素关键字值,关键字为正整型量,用空格分开,-99 为结束标
        志 : ");
        creat_chain_hash(HTC);
        printf("显示建立的开散列表 : \n\n");
        print_chain_hash(HTC);
        printf("\n 输入待查元素关键字 : ");
        scanf("% d",&i);
        k=search_chain_hash(HTC, i);
        if (k==0) printf("开散列表中待查元素不存在\n\n");
        else printf("开散列表中待查元素存在\n\n");
    }
```

【习题 9】 编写程序,实现在开散列表上插入元素。

题目分析:建立一个开散列表,散列函数为 H(key)=key MOD 13。输入插入记录的关键字,在开散列表上进行查找,查找不到,进行插入。为了简化算法,表中记录只含一

个正整型量关键字字段,记录的其余数据部分忽略。

结构说明:

```
#define KEYTYPE int
#define MAXSIZE 100

typedef struct node5
{
  KEYTYPE key;
  struct node5 * next;
}CHAINHASH;

CHAINHASH * HTC[MAXSIZE];
```

【解答】

```
#include "datastru.h"
#include <stdio.h>
#include <malloc.h>


void creat_chain_hash(CHAINHASH * HTC[])
{/* 建立开散列表 */
  CHAINHASH * p;
  int i, j;

  i=0; scanf("% d",&i);          /* 输入开散列表元素关键字值 */
  while (i !=-99) {
    j=i %  13;                   /* 散列函数: ADD(rec(key))=key MOD 13 */
    p=(CHAINHASH * ) malloc(sizeof(CHAINHASH));  /* 生成结点,挂入开散列表中 */
        p->next=HTC[j];
        p->key=i;
        HTC[j]=p;
        scanf("% d",&i); }       /* 输入开散列表元素关键字值 */
}

void print_chain_hash(CHAINHASH * HTC[])
{/* 显示开散列表 */
  int i;
  CHAINHASH * p;

for(i=0; i<13; i++) {
    if(HTC[i]==NULL) printf(" % 3d | ^ \n",i);
    else { p=HTC[i];
           printf(" % 3d |->",i);
```

```
            while(p !=NULL) {printf("% 5d->",p->key); p=p->next; }
            printf("\n");
    }
}
}

int search_chain_hash(CHAINHASH * HTC[], int k)
{/* 开散列表中查找元素 */
    CHAINHASH * p;
    int j;

    j=k % 13;                    /* 散列函数: ADD(rec(key))=key MOD 13 */
    p=HTC[j];
    if(p !=NULL)                 /* 开散列表中查找元素 */
        { while((p->key !=k)&&(p->next !=NULL)) p=p->next;
          if (p->key==k) return 1;   /* 查找成功,返回 1 */
          else return 0;         /* 查找不成功,返回 0 */
        }
    else return 0;
}

insert_chain_hash(CHAINHASH * HTC[], int i)
{/* 元素插入散列表中 */
  CHAINHASH * p;
  int j;

  j=i % 13;                      /* 散列函数: ADD(rec(key))=key MOD 13 */
  p= (CHAINHASH * ) malloc(sizeof(CHAINHASH)); /* 生成结点,挂入开散列表中 */
  p->next=HTC[j];
  p->key=i;
  HTC[j]=p;
}

main()
  { CHAINHASH * HTC[MAXSIZE];
    int i, k;

    printf("\n 建立开散列表\n\n");
    for (i=0; i<MAXSIZE; i++)
       HTC[i]=NULL;
    printf("请输入开散列表元素关键字值,关键字为正整型量,用空格分开,-99 为结束标
    志 : ");
    creat_chain_hash(HTC);
    printf("显示建立的开散列表 : \n\n");
```

```
    print_chain_hash(HTC);
    printf("\n输入待插入元素关键字 : ");
    scanf("% d",&i);
    k=search_chain_hash(HTC, i);
    if (k==0) { printf("表中待插入元素不存在,可插入散列表中\n\n");
                insert_chain_hash(HTC, i);
                printf("显示插入后的开散列表 : \n\n");
                print_chain_hash(HTC);}
    else printf("表中待插入元素存在,不可插入散列表中\n\n");
}
```

【习题 10】 编写程序,实现在闭散列表上查找元素。

题目分析:建立一个闭散列表,散列函数为 H(key)=key MOD 13。闭散列表长度为 16。输入待查记录的关键字,在闭散列表上进行查找。建立和查找过程中发生冲突时采用线性探测法处理。为了简化算法,表中记录只含一个正整型量关键字字段,记录的其余数据部分忽略。

结构说明:

```
#define KEYTYPE int
#define MAXSIZE 100

typedef struct
{
KEYTYPE key;
}HASHTABLE;

HASHTABLE * HTC[MAXSIZE];
```

【解答】

```
#include "datastru.h"
#include <stdio.h>

int seq_hash_search(KEYTYPE k, HASHTABLE * st)
{/* 闭散列表中查找元素 */
   int i, j;

   i=0;
   j=k %  13;            /* 散列函数: ADD(rec(key))=key MOD 13 */
   while(i<16 && st[j].key !=k && st[j].key !=-99)
     { i++;
       j=(j+1) %  16;}  /* 发生冲突,用线性探测法解决 */
   if(st[j].key !=k) return 0;      /* 查找不成功,返回 0 */
   else return 1;                   /* 查找成功,返回 1 */
}
```

```
void print_hashtable(HASHTABLE * st)
{/* 显示闭散列表 */
int i;

for(i=0; i<16; i++) printf("% 4d",i);
printf("\n\n");
for(i=0; i<16; i++) printf("% 4d",st[i].key);
printf("\n\n");
}

main( )
{ HASHTABLE a[16];                    /* 闭散列表长度为 16 */
  int i, j, k;

printf("\n 建立闭散列表\n\n");
printf("请输入闭散列表元素关键字值,关键字为正整型量,用空格分开,-99 为结束标志 : ");
for( j=0; j<16; j++)                  /* 闭散列表元素初始化 */
  a[j].key=-99;
scanf("% d",&i);                      /* 建立闭散列表:输入闭散列表元素关键字值 */
while (i !=- 99)
{ k=i % 13;                           /* 散列函数: ADD(rec(key))=key MOD 13 */
  if(a[k].key==-99) a[k].key=i;
  else { k=(k+1) % 16;                /* 发生冲突,用线性探测法解决 */
  while(a[k].key==-99) a[k].key=i;}
  scanf("% d",&i);                    /* 输入闭散列表元素关键字值 */
}
print_hashtable(a);
printf("\n");
printf("\n 输入待查元素关键字 : ");
scanf("% d",&i);
k=seq_hash_search(i, a);
if (k==0) printf("表中待查元素不存在\n\n");
else printf("表中待查元素存在\n\n");
}
```

【习题 11】 编写二叉排序树的综合练习程序。加深对二叉排序树的建立、插入、查找、删除、显示等算法的理解。

结构说明：二叉排序树中结点的结构采用二叉链表结构，描述如下：

```
#define KEYTYPE int

typedef struct node4
{KEYTYPE key;
  struct node4 * lchild, * rchild;
} BSTNODE;
```

程序运行说明：在运行此程序时，要求先选择建立二叉排序树而后再操作其他选择项。

【解答】

```
#include "datastru.h"
#include<string.h>
#include<malloc.h>
#include<stdio.h>

BSTNODE * searchnode(int w, BSTNODE * r)
/*按给定值在二叉排序树中查询*/
{  BSTNODE * p;

   if(r==NULL) p=NULL;                    /*空二叉排序树*/
   else { if(w==r->key) p=r;              /*给定值和根结点相同*/
          else if(w>r->key) p=searchnode(w, r->rchild);  /*递归继续查询*/
               else   p=searchnode(w, r->lchild);}
   return p;
}

BSTNODE * insert_bst(int w, BSTNODE * p)
/*将一个元素插入二叉排序树中*/
{  if(p==NULL )
   /*如果二叉排序树空,p作为新二叉排序树的根结点*/
       {p=malloc(sizeof(BSTNODE));
          p->lchild=NULL; p->rchild=NULL; p->key=w;}
   else if(w>p->key) p->rchild=insert_bst(w, p->rchild);
     /*如果二叉排序树不空,p作为叶子结点递归插入二叉排序树中*/
     else p->lchild=insert_bst(w, p->lchild);
   return p;                              /*返回根结点*/
}

BSTNODE * getfather(BSTNODE * p, BSTNODE * r)
/*在以 r 为根结点的二叉排序树中查询 p 结点的双亲结点并返回双亲结点的地址*/
{  BSTNODE * pf;

   if(r==NULL || p==r) pf=NULL;
   else { if(p==r->lchild || p==r->rchild) pf=r;
          else if(p->key>r->key) pf=getfather(p, r->rchild);
               else pf=getfather(p, r->lchild);}
   return pf;
}

BSTNODE * dele_bst(BSTNODE * p, BSTNODE * r){
```

```
/*p结点存在,删除p结点的算法*/
BSTNODE *temp, *tfather, *pf;

pf=getfather(p, r);   /*查找p结点的双亲结点,返回双亲结点的地址pf*/
if(p->lchild==NULL && p->rchild==NULL && pf !=NULL)
   /*被删结点是叶子结点,不是根结点*/
    if(pf->lchild==p) pf->lchild=NULL;
    else pf->rchild=NULL;
if(p->lchild==NULL && p->rchild==NULL && pf==NULL) r=NULL;
   /*被删结点是叶子结点,又是根结点*/
if(p->lchild==NULL && p->rchild !=NULL && pf !=NULL)
   /*被删结点有右孩子,无左孩子,不是根结点*/
   if(pf->lchild==p) pf->lchild=p->rchild;
   else pf->rchild=p->rchild;
if(p->lchild==NULL && p->rchild !=NULL && pf==NULL) r=p->rchild;
   /*被删结点有右孩子,无左孩子,又是根结点*/
if(p->lchild !=NULL && p->rchild==NULL && pf !=NULL)
   /*被删结点有左孩子,无右孩子,不是根结点*/
    if(pf->lchild==p) pf->lchild=p->lchild;
    else pf->rchild=p->lchild;
if(p->lchild !=NULL && p->rchild==NULL && pf==NULL) r=p->lchild;
   /*被删结点有左孩子,无右孩子,又是根结点*/
if(p->lchild !=NULL && p->rchild !=NULL)
   /*被删结点又有左孩子,又有右孩子*/
    {temp=p->lchild; tfather=p;
    while(temp->rchild !=NULL) {
       tfather=temp;
              temp=temp->rchild;}
    p->key=temp->key;
    if(tfather !=p) tfather->rchild=temp->lchild;
    else tfather->lchild=temp->lchild;
    }
printf("\n");
if(r !=NULL) printf("二叉排序树根结点是 %d\n\n", r->key);
else printf("二叉排序树空\n\n");
return r;
}

void print_bst(BSTNODE *p){
/*显示二叉排序树*/
if(p !=NULL )
      { print_bst(p->lchild);
        printf("%d ", p->key);
        if(p->lchild !=NULL) printf("%d的左结点是%d ", p->key, p->lchild->key);
```

```
        else printf("% d无左结点 ", p->key);
        if(p->rchild !=NULL) printf("% d的右结点是% d", p->key, p->rchild->key);
        else printf("% d无右结点", p->key);
        printf("\n");
        print_bst(p->rchild);
      }
}

BSTNODE * creat_bst() {
/* 建立二叉排序树 */
BSTNODE * root, * p;
int loop=0;
int s;

    root=NULL;
    do{ printf("\n"); printf("输入数据(整数,结束输入 0) : "); scanf("% d",&s);
        if(s==0) loop=1;
        else {p=searchnode(s, root);
              if(p==NULL) {root=insert_bst(s, root);
              /* 输入的数据不在二叉排序树中,方可插入二叉排序树 */
                        print_bst(root);} /* 边插入边显示二叉排序树中结点值 */
              else printf("输入的数据已存在,不能插入\n");
          }
        if(root !=NULL) printf("\n二叉排序树根结点为 % d\n\n", root->key);
    }while(!loop);
    return root;
}

BSTNODE * insert(BSTNODE * root) {
/* 将用户输入的结点插入二叉排序树中 */
int s;
BSTNODE * p;

    printf("\n");
    printf("输入插入结点数据(整数) : ");
    scanf("% d",&s);
    if(s !=0){p=searchnode(s,root);
            if(p==NULL){ root=insert_bst(s,root);
  /* 输入的数据不在二叉排序树中,方可插入二叉排序树 */
                        print_bst(root);   /* 边插入边显示二叉排序树中结点值 */
                        if(root !=NULL) printf("\n二叉排序树根结点为 % d\n\n",
                        root->key);}
                else printf("结点已存在,不再插入!!\n");}
    return root;
```

```
}

search_bst(BSTNODE * root) {
/* 在二叉排序树中查询用户输入的结点是否存在 */
int s;
BSTNODE * p;

printf("\n"); printf("输入待查结点值(整数): "); scanf("% d",&s);
if(s !=0)
    {p=searchnode(s, root);
     if(p==NULL) printf("结点不存在 !\n");
     else printf("结点存在 !\n");}
}

BSTNODE * delete(BSTNODE * root) {
/* 在二叉排序树中删除用户指定的结点 */
int s;
BSTNODE * p;
char ch;

printf("\n"); printf("输入待删除结点值(整数): "); scanf("% d", &s);
if(s !=0)
      { p=searchnode(s,root);                    /* 用户指定要删除的结点存在吗? */
        if(p==NULL) printf("结点不存在 !\n\n"); /* 结点不存在 */
        else { printf("结点存在,删除吗 ? (Y/N) ");
        fflush(stdin);
        scanf("% c",&ch);
                if((ch=='y')||(ch=='Y')) {root=dele_bst(p,root); print_bst(root);}
                                           /* 结点存在,确认删除 */
          }
      }
   return root;
}

main()
{
  BSTNODE * root;
  int loop, i;

  loop=1;
  while(loop)
  { printf("\n\n\n");
    printf(" 1: 二叉排序树--- 建立\n");
    printf(" 2: 二叉排序树--- 插入\n");
```

```
        printf(" 3: 二叉排序树--- 查找\n");
        printf(" 4: 二叉排序树--- 删除\n");
        printf(" 5: 二叉排序树--- 显示\n");
        printf(" 0: 退出\n");
        scanf("% d",&i);
        switch(i)
        { case 0: loop=0; break;                    /* 退出 */
          case 1: root=creat_bst(); break;          /* 建立 */
          case 2: root=insert(root); break;         /* 插入 */
          case 3: search_bst(root); break;          /* 查找 */
          case 4: root=delete(root); break;         /* 删除 */
          case 5: printf("\n");                     /* 显示 */
                  if(root !=NULL) printf("二叉排序树的根是% d\n",root->key);
                  print_bst(root); break;
        }
    }
}
```

8.2 实训练习题

【习题 12】 分别画出在线性表(a, c, e, f, g, j, k, m, n)中进行二分查找,以查关键字等于 b、k 的过程。

【习题 13】 将二分法查找算法改写成递归算法。

【习题 14】 写一个算法,实现顺序表上顺序查找,并将监视哨设在高端。

题目分析:参照习题 6,查找表的存储结构为顺序表,对表中记录存放的先后次序没有任何要求。建立一个顺序表,记录从下标为 1 的单元开始放入,监视哨设在高位。输入待查记录的关键字进行查找。

【习题 15】 编写程序,实现在开散列表上删除元素。

题目分析:参照习题 8,建立一个开散列表,散列函数为 H(key)=key MOD 13。为了简化算法,表中记录只含一个整型量关键字字段,记录的其余数据部分忽略。输入删除记录的关键字,在开散列表上进行查找,查找成功,进行删除。

【习题 16】 编写程序,实现在闭散列表上插入元素。

题目分析:参照习题 10,建立一个闭散列表,散列函数为 H(key)=key MOD 13。为了简化算法,表中记录只含一个整型量关键字字段,记录的其余数据部分忽略。输入插入记录的关键字,在闭散列表上进行查找,查找不到,进行插入。建立和查找过程中发生冲突时采用线性探测法处理。

内部排序

以下内部排序的各种算法中，排序对象的数据元素表采用顺序存储结构。为简单起见，元素最大容量为100，都只包含一个字段即关键字段，且关键字都是整型值。排序表的结构如下：

```
#define KEYTYPE int
#define MAXSIZE 100

typedef struct
{ KEYTYPE key;
}RECNODE;

RECNODE a[MAXSIZE];
```

9.1 习题解析

【习题1】 单项选择题。

1. 下列排序方法中，时间复杂度不受数据初始状态影响，恒为$O(n\log_2 n)$的是_____。

A. 堆排序　　B. 冒泡排序　　C. 直接选择排序　　D. 快速排序

【解答】 A。

2. 下列排序方法中，在待排序的数据已经为有序时，花费时间最多的是_____。

A. 快速排序　　B. 直接选择排序　　C. 冒泡排序　　D. 堆排序

【解答】 A。

3. 已知表A在排序前已按关键字值递增排序，则_____方法的比较次数最少。

A. 直接插入排序　　B. 快速排序　　C. 归并排序　　D. 选择排序

【解答】 A。

4. 快速排序方法在_____情况下最不利于发挥其长处。

A. 要排序的数据量太大　　B. 要排序的数据中含有多个相同值

C. 要排序的数据已基本有序　　D. 要排序的数据个数为奇数

【解答】 C。

5. 若一组记录的关键字为(46,79,56,38,40,84),则利用堆排序的方法建立的初始堆为_____。

A. 79,46,56,38,40,80　　B. 84,79,56,38,40,46

C. 84,79,56,46,40,38　　D. 84,56,79,40,46,38

【解答】 B。

6. 若一组记录的关键字为(46,79,56,38,40,84),则利用快速排序的方法,以第1个记录为基准得到的一次划分结果为_____。

A. 38,40,46,56,79,84　　B. 40,38,46,79,56,84

C. 40,38,46,56,79,84　　D. 40,38,46,84,56,79

【解答】 对于(46,79,56,38,40,84),取出46,将40复制到46处,得到(40,79,56,38,40,84),再将79复制到40处,得到(40,79,56,38,79,84),将38复制到79处,得到(40,38,56,38,79,84),将56复制到38处,得到(40,38,56,56,79,84),将46复制到56处,得到(40,38,46,56,79,84)。本题答案为C。

7. 已知10个数据元素为(54,28,16,34,73,62,95,60,26,43),对该数列按从小到大排序,经过一趟冒泡排序后的序列为_____。

A. 16,28,34,54,73,62,60,26,43,95

B. 28,16,34,54,62,73,60,26,43,95

C. 28,16,34,54,62,60,73,26,43,95

D. 16,28,34,54,62,60,73,26,43,95

【解答】 冒泡排序每趟经过比较交换从无序区中产生一个最大的元素,放在序列的最后,根据冒泡排序算法的比较交换规则答案B是正确的结果。

8. 以下排序方法中,最好时间复杂度为O(n)的依次是_____、_____。

A. 直接插入排序　　B. 直接选择排序

C. 冒泡排序　　D. 快速排序

【解答】 A、C。

9. 以下排序方法中,最坏时间复杂度为O(n^2)的依次是_____、_____。

A. 直接插入排序　　B. 直接选择排序

C. 堆排序　　D. 归并排序

【解答】 A、B。

【习题2】 判断以下叙述的正确性。

(1) 只有在记录的关键字的初始状态为逆序排列的情况下,冒泡排序过程中元素的移动次数才会达到最大值。

(2) 只有在记录的关键字的初始状态为逆序排列的情况下,直接选择排序过程中元素的移动次数才会达到最大值。

(3) 只有在记录的关键字的初始状态为逆序排列的情况下,直接插入排序过程中元素的移动次数才会达到最大值。

(4) 只有在记录的关键字的初始状态为顺序排列的情况下，快速排序过程中关键字的比较次数才会达到最大值。

(5) 对 n 个元素进行直接选择排序，关键字的比较次数总是 $n(n-1)/2$ 次。

(6) 对 n 个元素执行快速排序，在进行第一次分组时，关键字的比较次数总是 $n-1$ 次。

【解答】

(1) 正确。

(2) 错误。

(3) 正确。

(4) 错误。

(5) 正确。

(6) 正确。

【习题 3】 以关键字序列(265,301,751,129,937,863,742,694,076,438)为例，分别写出执行各种排序算法的各趟排序结束时，关键字序列的状态。

【解答】

(1) 直接插入排序各趟排序如下：

初始状态：265,301,751,129,937,863,742,694,076,438。

第 1 趟 (265,301),751,129,937,863,742,694,076,438。

第 2 趟 (265,301,751),129,937,863,742,694,076,438。

第 3 趟 (129,265,301,751),937,863,742,694,076,438。

第 4 趟 (129,265,301,751,937),863,742,694,076,438。

第 5 趟 (129,265,301,751,863,937),742,694,076,438。

第 6 趟 (129,265,301,742,751,863,937),694,076,438。

第 7 趟 (129,265,301,694,742,751,863,937),076,438。

第 8 趟 (076,129,265,301,694,742,751,863,937),438。

第 9 趟 (076,129,265,301,438,694,742,751,863,937)。

(2) 冒泡排序各趟排序如下：

初始状态：265,301,751,129,937,863,742,694,076,438。

第 1 趟：265,301,129,751,863,742,694,076,438,(937)。

第 2 趟：265,129,301,751,742,694,076,438,(863,937)。

第 3 趟：129,265,301,742,694,076,438,(751,863,937)。

第 4 趟：129,265,301,694,076,438,(742,751,863,937)。

第 5 趟：129,265,301,076,438,(694,742,751,863,937)。

第 6 趟：129,265,076,301,(438,694,742,751,863,937)。

第 7 趟：129,076,265,(301,438,694,742,751,863,937)。

第 8 趟：076,129,(265,301,438,694,742,751,863,937)。

第 9 趟：076,(129,265,301,438,694,742,751,863,937)。

(3) 直接选择排序各趟排序如下：

初始状态：265,301,751,129,937,863,742,694,076,438。
第 1 趟：(265,301),751,129,937,863,742,694,076,438。
第 2 趟：(265,301,751),129,937,863,742,694,076,438。
第 3 趟：(129,265,301,751),937,863,742,694,076,438。
第 4 趟：(129,265,301,751,937),863,742,694,076,438。
第 5 趟：(129,265,301,751,863,937),742,694,076,438。
第 6 趟：(129,265,301,742,751,863,937),694,076,438。
第 7 趟：(129,265,301,694,742,751,863,937),076,438。
第 8 趟：(076,129,265,301,694,742,751,863,937),438。
第 9 趟：(076,129,265,301,438,694,742,751,863,937)。
(4) 快速排序各趟排序如下：
初始状态：265,301,751,129,937,863,742,694,076,438。
第 1 趟：(076,129),265,(751,937,863,742,694,301,438)。
第 2 趟：076,129,265,(438,301,694,742),751,(863,937)。
第 3 趟：076,129,265,301,438,(694,742),751,(863,937)。
第 4 趟：076,129,265,301,438,694,742,751,(863,937)。
第 5 趟：076,129,265,301,438,694,742,751,863,937。
(5) 堆排序(大顶堆)各趟排序如下：
初始状态：265,301,751,129,937,863,742,694,076,438。
第 1 趟：(937,694,863,265,438,751,742,129,076,301)。
第 2 趟：(863,694,751,265,438,301,742,129,076),937。
第 3 趟：(751,694,742,265,438,301,076,129),863,937。
第 4 趟：(742,694,301,265,438,129,076),751,863,937。
第 5 趟：(694,438,301,265,076,129),742,751,863,937。
第 6 趟：(438,265,301,129,076),694,742,751,863,937。
第 7 趟：(301,265,076,129),438,694,742,751,863,937。
第 8 趟：(265,129,076),301,438,694,742,751,863,937。
第 9 趟：(129,076),265,301,438,694,742,751,863,937。
第 10 趟：076,129,265,301,438,694,742,751,863,937。
(6) 归并排序各趟排序如下：
初始状态：265,301,751,129,937,863,742,694,076,438。
第 1 趟：[265],[301],[751],[129],[937],[863],[742],[694],[076],[438]。
第 2 趟：[265,301],[129,751],[863,937],[694,742],[076,438]。
第 3 趟：[129,265,301,751],[694,742,863,937],[076,438]。
第 4 趟：[076,129,265,301,438,694,742,751,863,937]。
(7) 基数排序各趟排序如下：
初始状态：265,301,751,129,937,863,742,694,076,438。
第 1 趟(个位)：301,751,742,863,694,265,076,937,438,129。

第 2 趟(十位)：301,129,937,438,742,751,265,863,076,694。

第 3 趟(百位)：076,129,265,301,438,694,742,751,863,937。

【习题 4】 一个线性表中的元素为正整数和负整数。设计一算法，将正整数和负整数分开，使线性表的前一半为负整数，后一半为正整数。

【解答】

```
void sgroup(SEQUENLIST * q )
{
    int flag=1;
    int k=0,j,t;
    if (q->last<=0)
        return;
    while(flag)
    {
        while(q->datas[k]<0 && k<q->last)
            k++;

        for (j=k+1;q->datas[j]>=0 && j<q->last ; j++);
        if (j<q->last)
        {
            t=q->datas[k];
            q->datas[k]=q->datas[j];
            q->datas[j]=t;
        }
        else
            flag=0;
    }
}
```

【习题 5】 不难看出，对长度为 n 的记录序列进行快速排序时，所需进行的比较次数依赖于这 n 个元素的初始序列。

(1) $n=7$ 时在最好情况下需进行多少次比较？说明理由。

(2) 对 $n=7$ 给出一个最好情况的初始排列实例。

【解答】

(1) 因快速排序的最好情况是每次选定的枢轴记录都将待排序列分成两个独立的长度几乎相等的子序列，即第一趟快速排序的范围是 n 个记录，第二趟快速排序的范围是两个长度各为 $n/2$ 个记录的子序列，第三趟快速排序的范围是四个长度各为 $n/4$ 个记录的子序列，依此类推，整个算法时间复杂度为 $O(n\log_2 n)$。$n=7$ 时在最好情况下需进行 14 次比较。

对 $n=7$，第一趟比较 6 次，将待排序列分成两个长度为 3 的子序列，第二趟对两个长度为 3 的子序列各比较 2 次，共 4 次，待排序列分成四个长度为 1 的子序列，第三趟对四个长度为 1 的子序列各比较 1 次，共 4 次，完成了对长度为 7 的记录的快速排序，共

14 次。

(2) 对 $n=7$ 给出一个最好情况的初始排列实例：

[50 35 5 70 32 80 60]
[32 35 5] 50 [70 80 60]
[5] 32 [35] 50 [60] 70 [80]
5 32 35 50 60 70 80

【习题 6】 判别以下序列是否为堆(小顶堆或大顶堆)，如果不是，则按算法把它调整为堆。

(1) (100, 86, 48, 73, 35, 39, 42, 57, 66, 21)

(2) (12, 70, 33, 65, 24, 56, 48, 92, 86, 33)

(3) (103, 97, 56, 38, 66, 53, 42, 12, 30, 52, 06, 20)

(4) (05, 56, 20, 3, 23, 40, 38, 29, 61, 5, 76, 28, 100)

【解答】

(1)和(3)是大顶堆，(2)和(4)不是堆。

调整(2)为堆：70 和 24 交换后，该序列调整为小顶堆。

调整(4)为堆：(05, 56, 20, 3, 23, 40, 38, 29, 61, 5, 76, 28, 100)。

第一趟：(05, 56, 20, 3, 23, 28, 38, 29, 61, 5, 76, 40, 100)。

第二趟：(05, 56, 20, 3, 5, 28, 38, 29, 61, 23, 76, 40, 100)。

第三趟：(05, 3, 20, 29, 5, 28, 38, 56, 61, 23, 76, 40, 100)。

第四趟：(3, 05, 20, 29, 5, 28, 38, 56, 61, 23, 76, 40, 100)，该序列调整为小顶堆。

【习题 7】 编写程序，实现直接插入排序。

题目分析：将输入的若干个整数按直接插入排序算法从小到大排序。数据从数组的 1 单元放起。

【解答】

```
#include "datastru.h"
#include <stdio.h>

void insertsort(RECNODE * r, int n)
{/* 直接插入排序 */
    int i,j;

    for(i=2; i<=n; i++)
    { r[0]=r[i]; j=i-1;              /* r[0]是监视哨,j 表示当前已排好序列的长度 */
      while(r[0].key<r[j].key)       /* 确定插入位置 */
          {r[j+1]=r[j]; j--;}
      r[j+1]=r[0];                   /* 元素插入 */
    }
}
```

```
main( )
{  RECNODE a[MAXSIZE];
   int i, j, k, len;

   printf("\n\n输入待排序数据(整数,以空格隔开,0结束) : "); k=0; scanf("% d",&j);
   while(j !=0) { k++; a[k].key=j; scanf("% d",&j); }
   len=k;
   printf("\n待排序前 : ");
   for (i=0; i<len; i++) printf(" % d",a[i+1].key);
   printf("\n");
   insertsort (a,len);
   printf("\n\n待排序后 : ");
   for (i=0; i<len; i++) printf(" % d",a[i+1].key);
   printf("\n\n");
}
```

【习题 8】 编写程序,实现冒泡排序。

题目分析:将输入的若干个整数按冒泡排序算法从小到大排序。数据从数组的 1 单元放起。

【解答】

```
#include "datastru.h"
#include <stdio.h>

void bublesort(RECNODE * r, int n)
{/* 简单交换排序:冒泡排序 */
    int i, j;
    RECNODE temp;
    for(i=1; i<n; i++)
      for(j=n-1; j>=i; j--)
        if(r[j+1].key<r[j].key)
          {temp=r[j+1]; r[j+1]=r[j]; r[j]=temp;}
}

main( )
{ RECNODE a[MAXSIZE];
  int i, j, k, len;

  printf("\n\n输入待排序数据(整数,以空格隔开,0结束) : "); k=0; scanf("% d",&j);
  while(j !=0) { k++; a[k].key=j; scanf("% d",&j); }
  len=k;
  printf("\n待排序前 : ");
  for (i=0; i<len; i++) printf(" % d",a[i+1].key);
  printf("\n");
  bublesort (a,len);
```

```
    printf("\n\n待排序后 : ");
    for (i=0; i<len; i++) printf(" % d",a[i+1].key);
    printf("\n\n");
}
```

【习题 9】 编写程序,实现简单选择排序。

题目分析:将输入的若干个整数按简单选择排序算法从小到大排序。数据从数组的1单元放起。

【解答】

```
#include "datastru.h"
#include <stdio.h>

void selesort(RECNODE * r, int n)
{/* 简单选择排序 */
    int i,j,k;
    RECNODE temp;

    for(i=1; i<n; i++)
      { k=i;            /* k:最小关键字的初始位置 */
        for(j=i+1; j<=n; j++)
          if(r[j].key<r[k].key)
           k=j;         /* k:跟踪记录当前最小关键字的位置 */
        if(k !=i)       /* 最小关键字元素和待排序列的第一个元素交换 */
          {temp=r[i]; r[i]=r[k]; r[k]=temp;}
    }
}

main( )
{ RECNODE a[MAXSIZE];
  int i, j, k, len;

  printf("\n\n输入待排序数据(整数,以空格隔开,0结束) : "); k=0; scanf("% d",&j);
  while(j !=0) { k++; a[k].key=j; scanf("% d",&j); }
  len=k;
  printf("\n待排序前 : ");
  for (i=0; i<len; i++) printf(" % d",a[i+1].key);
  printf("\n");
  selesort (a,len);
  printf("\n\n待排序后 : ");
  for (i=0; i<len; i++) printf(" % d",a[i+1].key);
  printf("\n\n");
}
```

【习题 10】 编写程序,实现快速排序。

题目分析：将输入的若干个整数按快速排序算法从小到大排序。数据从数组的1单元放起。

【解答】

```
#include "datastru.h"
#include <stdio.h>

int partition(RECNODE * r, int * low, int * high)
{/* 一趟快速排序,返回 i,产生了两个独立的待排子序列 */
    int i, j;
    RECNODE temp;

    i= * low; j= * high;
    temp=r[i];                              /* 枢轴记录保存在 temp 变量中 */
    do{ while((r[j].key>=temp.key) && (i<j)) j--;  /* j 指针记录和枢轴记录比较 */
        if(i<j) { r[i]=r[j]; i++;}
        while((r[i].key<=temp.key) && (i<j)) i++;  /* i 指针记录和枢轴记录比较 */
        if(i<j) { r[j]=r[i]; j--;}
          }while(i !=j);
    r[i]=temp;                              /* 枢轴记录的排序位置确定在 i */
    return i;
}

void quicksort(RECNODE * r, int start, int end)
{/* 快速排序 */
    int i;
    if(start<end)
      {i= partition(r, &start, &end);      /* 一趟快速排序,返回 i,产生了两个独立的待排
                                              子序列 */
          quicksort(r, start, i-1);        /* 对两个独立的待排子序列分别递归调用快速
                                              排序算法 */
          quicksort(r, i+1,end);}
}

main( )
{ RECNODE a[MAXSIZE];
  int i, j, k, len, start;

  printf("\n\n输入待排序数据(整数,以空格隔开,0 结束) : "); k=0; scanf("% d",&j);
  while(j !=0) { k++; a[k].key=j; scanf("% d",&j); }
  len=k; start=1;
  printf("\n待排序前 : ");
  for (i=0; i<len; i++) printf(" % d",a[i+1].key);
  printf("\n");
```

```
    quicksort (a,start,len);
    printf("\n\n待排序后 : ");
    for (i=0; i<len; i++) printf(" % d",a[i+1].key);
    printf("\n\n");
}
```

【习题 11】 编写程序,实现堆排序。

题目分析:将输入的若干个整数按堆排序算法从小到大排序。数据从数组的 1 单元放起。

【解答】

```
#include "datastru.h"
#include <stdio.h>

void sift(RECNODE * r, int i, int m)
{/* i是根结点编号,m是以 i结点为根的子树中最后一个结点的编号 */
    int j;
    RECNODE temp;

    temp=r[i];
    j=2* i;                 /* j为 i根结点的左孩子 */
    while(j<=m)
      {if(j<m && (r[j].key<r[j+1].key))
         j++;               /* 当 i结点有左右孩子时,j取关键字大的孩子结点编号 */
       if(temp.key<r[j].key)
         { r[i]=r[j]; i=j; j=2* i;}/* 按堆定义调整,并向下一层筛选调整 */
       else break; /* 筛选调整完成,跳出循环 */
      }
    r[i]=temp;
}

void heapsort(RECNODE * r, int n)
{/* 堆排序:n为 r表中记录数,从 r[1]开始放起 */
    int i;
    RECNODE temp;

    for(i=n/2; i>=1; i--)
      sift(r, i, n);        /* 将无序序列建成大堆 */
    for(i=n; i>=2; i--)
      {temp=r[1];           /* 堆顶及堆尾元素交换 */
      r[1]=r[i];
      r[i]=temp;
      sift(r,1,i-1);        /* 交换后,从第一个元素开始调整为大堆,每次记录个数少一个 */
     }
}
```

```
main( )
{ RECNODE a[MAXSIZE];
  int i, j, k, len;

  printf("\n\n输入待排序数据(整数,以空格隔开,0结束) : "); k=0; scanf("% d",&j);
  while(j !=0) { k++; a[k].key=j; scanf("% d",&j); }
  len=k;
  printf("\n待排序前 : ");
  for (i=0; i<len; i++) printf(" % d",a[i+1].key);
  printf("\n");
  heapsort (a,len);
  printf("\n\n待排序后 : ");
  for (i=0; i<len; i++) printf(" % d",a[i+1].key);
  printf("\n\n");
}
```

【习题 12】 编写程序,实现二路归并排序。

题目分析:将输入的若干个整数按二路归并排序算法从小到大排序。数据从数组的0单元放起。

【解答】

```
#include "datastru.h"
#include <stdio.h>

void merge(RECNODE * r, int low, int m, int high)
{/* 两相邻的有序表(一个从 low 到 m;另一个从 m+1 到 high)合并为一个有序表(从 low 到 high) */
    RECNODE r1[MAXSIZE];              /* 合并时用的缓冲向量 */
    int i, j, k;

    i=low;
    j=m+ 1;
    k=0;
    while(i<=m && j<=high)            /* 两相邻的有序表合并 */
      if(r[i].key<=r[j].key)
        {r1[k]=r[i]; i++; k++;}
      else
        {r1[k]=r[j]; j++; k++;}
    while(i<=m)                       /* 有序表剩余部分处理 */
       {r1[k]=r[i]; i++; k++;}
    while(j<=high)                    /* 有序表剩余部分处理 */
       {r1[k]=r[j]; j++; k++;}
    for(i=low, k=0; i<=high; i++, k++)  /* 缓冲向量 r1 复制到原来的 r 中 */
       r[i]=r1[k];
```

```
}

void merge_one(RECNODE * r, int lenth, int n)
{/* 二路归并中的"一趟归并"算法 */
    int i=0;

    while(i+2 * lenth-1<n)
      {merge(r, i, i+lenth-1, i+2 * lenth-1);  /* 两子序列长度相等的 */
       i=i+2 * lenth;}                         /* 情况下调用 merge */
    if(i+lenth-1<n-1)
       merge(r, i, i+lenth-1, n-1);  /* 序列中的余留部分处理 */
}

void mergesort(RECNODE * r, int n)
{/* 二路归并排序算法 */
    int lenth=1;                     /* 有序子序列长度初始值=1 */

    while(lenth<n)
      {merge_one(r, lenth, n);       /* 调用"一趟归并"的算法 */
       lenth=2 * lenth;}             /* 有序子序列长度加倍 */
}

main( )
{ RECNODE a[MAXSIZE];
  int i, j, k, len;

  printf("\n\n输入待排序数据(整数,以空格隔开,0 结束) : "); k=0; scanf("% d",&j);
  while(j !=0) { k++; a[k-1].key=j; scanf("% d",&j); }
  len=k;
  printf("\n待排序前 : ");
  for (i=0; i<len; i++) printf(" % d",a[i].key);
  printf("\n");
  mergesort (a,len);
  printf("\n");
  printf("\n\n待排序后 : ");
  for (i=0; i<len; i++) printf(" % d",a[i].key);
  printf("\n");
}
```

【习题 13】 编写程序,将上面的各个排序算法合并在一个综合程序中,通过菜单选择方式对数据进行排序。

【解答】

```
#include <stdio.h>
#include "datastru.h"
```

```
int createList(RECNODE * r)
{ int j, k;

 printf("\n\n输入待排序数据(整数,以空格隔开,0结束) : "); k=0; scanf("% d",&j);
 while(j !=0) { k++; r[k].key=j; scanf("% d",&j); }
 return k;
}

frontdisplayList(RECNODE * r, int n)
{int i;

 printf("\n待排序前 : ");
 for (i=0; i<n; i++) printf(" % d",r[i+1].key);
 printf("\n\n");
}

reardisplayList(RECNODE * r, int n)
{int i;

 printf("\n\n待排序后 : ");
 for (i=0; i<n; i++) printf(" % d",r[i+1].key);
 printf("\n\n");
}

void insertsort(RECNODE * r, int n)
{/* 直接插入排序 */
    int i,j;

    for(i=2; i<=n; i++)
      { r[0]=r[i]; j=i-1;            /* r[0]是监视哨,j表示当前已排好序列的长度 */
        while(r[0].key< r[j].key)    /* 确定插入位置 */
           {r[j+1]=r[j]; j--;}
        r[j+1]=r[0];                 /* 元素插入 */
    }
}

void bublesort(RECNODE * r, int n)
{/* 简单交换排序:冒泡排序 */
    int i, j;
    RECNODE temp;
    for(i=1; i<n; i++)
      for(j=n- 1; j>=i; j--)
        if(r[j+1].key<r[j].key)
```

```
            {temp=r[j+1]; r[j+1]=r[j]; r[j]=temp;}
}

int partition(RECNODE * r, int * low, int * high)
{/* 一趟快速排序,返回 i,产生了两个独立的待排子序列 */
    int i, j;
    RECNODE temp;

    i= * low; j= * high;
    temp=r[i];                                    /* 枢轴记录保存在 temp 变量中 */
    do{ while((r[j].key>=temp.key) && (i<j)) j--;   /* j 指针记录和枢轴记录比较 */
        if(i<j) { r[i]=r[j]; i++;}
        while((r[i].key<=temp.key) && (i<j)) i++;   /* i 指针记录和枢轴记录比较 */
        if(i<j) { r[j]=r[i]; j--;}
          }while(i !=j);
    r[i]=temp;                                    /* 枢轴记录的排序位置确定在 i */
    return i;
}

void quicksort(RECNODE * r, int start, int end)
{/* 快速排序 */
    int i;
    if(start<end)
      {i= partition(r, &start, &end);  /* 一趟快速排序,返回 i,产生了两个独立的待排子
                                          序列 */
          quicksort(r, start, i-1);    /* 对两个独立的待排子序列分别递归调用快速排序
                                          算法 */
          quicksort(r, i+1,end);}
}

void selesort(RECNODE * r, int n)
{/* 简单选择排序 */
    int i,j,k;
    RECNODE temp;

    for(i=1; i<n; i++)
      { k=i;                    /* k:最小关键字的初始位置 */
        for(j=i+1; j<=n; j++)
          if(r[j].key<r[k].key)
        k=j;                    /* k:跟踪记录当前最小关键字的位置 */
        if(k !=i)               /* 最小关键字元素和待排序列的第一个元素交换 */
          {temp=r[i]; r[i]=r[k]; r[k]=temp;}
    }
}
```

```
void sift(RECNODE * r, int i, int m)
{/* i是根结点编号,m是以 i 结点为根的子树中最后一个结点的编号 */
    int j;
    RECNODE temp;

    temp=r[i];
    j=2*i;              /* j 为 i 根结点的左孩子 */
    while(j<=m)
      {if(j<m && (r[j].key<r[j+1].key))
        j++;            /* 当 i 结点有左右孩子时,j 取关键字大的孩子结点编号 */
       if(temp.key<r[j].key)
        { r[i]=r[j]; i=j; j=2*i;}/* 按堆定义调整,并向下一层筛选调整 */
       else break; /* 筛选调整完成,跳出循环 */
      }
    r[i]=temp;
}

void heapsort(RECNODE * r, int n)
{/* 堆排序: n为 r 表中记录数,从 r[1]开始放起 */
    int i;
    RECNODE temp;

    for(i=n/2; i>=1; i--)
      sift(r, i, n);      /* 将无序序列建成大堆 */
    for(i=n; i>=2; i--)
      {temp=r[1];         /* 堆顶及堆尾元素交换 */
       r[1]=r[i];
       r[i]=temp;
       sift(r,1,i-1);     /* 交换后,从第一个元素开始调整为大堆,每次记录个数少一个 */
    }
}

void merge(RECNODE * r, int low, int m, int high)
{/* 两相邻的有序表(一个从 low 到 m;另一个从 m+1 到 high)合并为一个有序表(从 low 到 high) */
    RECNODE r1[MAXSIZE];                    /* 合并时用的缓冲向量 */
    int i, j, k;

    i=low;
    j=m+1;
    k=0;
    while(i<=m && j<=high)                  /* 两相邻的有序表合并 */
      if(r[i].key<=r[j].key)
```

```
            {r1[k]=r[i]; i++; k++;}
        else
            {r1[k]=r[j]; j++; k++;}
    while(i<=m)                                    /*有序表剩余部分处理*/
        {r1[k]=r[i]; i++; k++;}
    while(j<=high)                                 /*有序表剩余部分处理*/
        {r1[k]=r[j]; j++; k++;}
    for(i=low, k=0; i<=high; i++, k++)             /*缓冲向量 r1 复制到原来的 r 中*/
        r[i]=r1[k];

}

void merge_one(RECNODE * r, int lenth, int n)
{/*二路归并中的"一趟归并"算法*/
    int i=0;

    while(i+2*lenth-1<n)
      {merge(r, i, i+lenth-1, i+2*lenth-1);        /*两子序列长度相等的*/
       i=i+2*lenth;}                               /*情况下调用 merge*/
    if(i+lenth-1<n-1)
       merge(r, i, i+lenth-1, n-1);                /*序列中的余留部分处理*/
}

void mergesort(RECNODE * r, int n)
{/*二路归并排序算法*/
    int lenth=1;                                   /*有序子序列长度初始值= 1*/

    while(lenth<n)
      {merge_one(r, lenth, n);                     /*调用"一趟归并"的算法*/
       lenth=2*lenth;}                             /*有序子序列长度加倍*/
}

void main() {
    RECNODE a[MAXSIZE];
    int len, b,j,k;
    int loop=1;

    while (loop) {
    printf("\n\n排序综合练习\n\n");
    printf(" 0--退出\n");
    printf(" 1--直接插入排序\n");
    printf(" 2--简单交换(冒泡)排序\n");
    printf(" 3--快速排序\n");
    printf(" 4--简单选择排序\n");
```

```
        printf(" 5--堆排序\n");
        printf(" 6--二路归并排序\n");
        printf("\n\n 请选择项号 : ");
        scanf("% d",&b);
        printf("\n\n");
        if(b>=0 && b<=6)
            switch(b) {
                case 0: loop=0;
                        break;
                case 1: len=createList(a);
                     frontdisplayList(a,len);
                     insertsort(a,len);
                     reardisplayList(a,len);
                     break;
                case 2: len=createList(a);
                     frontdisplayList(a,len);
                     bublesort(a, len);
                     reardisplayList(a,len);
                     break;
                case 3: len=createList(a);
                     frontdisplayList(a,len);
                     quicksort(a, 1, len);
                     reardisplayList(a,len);
                     break;
                case 4: len=createList(a);
                     frontdisplayList(a,len);
                     selesort(a, len);
                     reardisplayList(a,len);
                     break;
                case 5: len=createList(a);
                     frontdisplayList(a,len);
                     heapsort(a, len);
                     reardisplayList(a,len);
                     break;
case 6: printf("\n\n 输入待排序数据(整数,以空格隔开,0 结束) : "); k=0; scanf("% d",&j);
                     while(j !=0) { k++; a[k-1].key=j; scanf("% d",&j); }
                     len=k;
                     printf("\n 待排序前 : ");
                     for (j=0; j<len; j++) printf(" % d",a[j].key);
                     printf("\n\n");
                     mergesort(a, len);
                     printf("\n\n 待排序后 : ");
                     for (j=0; j<len; j++) printf(" % d",a[j].key);
                     printf("\n\n");
```

```
                break;
            }
        printf("\n\n结束此练习吗？(0——结束 1——继续)：");
        scanf("% d",&loop);
        printf("\n\n");
        }
}
```

9.2 实训练习题

【习题 14】 单项选择题。

1. 在下列排序方法中，某一趟结束后未必能选出一个元素放在其最终位置上的是_____。

A. 堆排序　　B. 冒泡排序　　C. 直接插入排序　　D. 快速排序

2. 依次将待排序序列中的元素和有序序列合并为一个新的有序子序列的排序方法是_____。

A. 快速排序　　B. 插入排序　　C. 冒泡排序　　D. 堆排序

3. 在下列排序方法中，关键字比较的次数与记录的初始排列次序无关的是_____。

A. 快速排序　　B. 冒泡排序　　C. 插入排序　　D. 选择排序

4. 数据表 A 中有 10000 个元素，如果仅要求选出其中最大的 10 个元素，则采用_____方法最节省时间。

A. 堆排序　　B. 希尔排序　　C. 快速排序　　D. 直接选择排序

5. 一组记录的关键字为(25,48,16,35,79,82,23,40,36,72)，其中，含有 5 个长度为 2 的有序表，按归并排序的方法对该序列进行一趟归并后的结果为_____。

A. 16,25,35,48,23,40,79,82,36,72

B. 16,25,35,48,79,82,23,36,40,72

C. 16,25,48,35,79,82,23,36,40,72

D. 16,25,35,48,79,23,36,40,72,82

6. 有一组记录的关键字为(48,36,68,99,75,24,28,52)进行快速排序，要求结果从小到大排序，则进行一次划分之后结果为_____。

A. (24 28 36) 48 (52 68 75 99)　　B. (28 36 24) 48 (75 99 68 52)

C. (36 88 99) 48 (75 24 28 52)　　D. (28 36 24) 48 (99 75 68 52)

7. 以下排序方法中，平均时间复杂度为 O(n^2)的依次是_____、_____。

A. 直接插入排序　　B. 冒泡排序

C. 快速排序　　D. 基数排序

【习题 15】 判断以下叙述的正确性。

(1) 内排序中的快速排序方法，在任何情况下均可得到最快的排序效果。

(2) 基数排序的设计思想是依照对关键字值的比较来实施的。

(3) 当待排序的元素很大时,为了交换元素的位置,移动元素要占用较多的时间,这是影响时间复杂度的主要因素。

(4) 对一个堆,按二叉树层次进行遍历可以得到一个有序序列。

【习题 16】 对于给定的一组关键字:83, 40, 63,13, 84, 35, 96, 57, 39, 79, 61, 15。分别画出用直接插入排序、冒泡排序、简单选择排序、快速排序、堆排序、归并排序对上述序列进行排序中的各趟结果。

【习题 17】 试比较直接插入排序、冒泡排序、简单选择排序、快速排序、堆排序、归并排序的特点和各自的适应性。分析它们的算法平均时间复杂度。

【习题 18】 编写程序,实现直接插入排序,将监视哨设在高位。

【习题 19】 编写程序,实现快速排序,用非递归算法实现。用栈和队列都可以。

题目分析:前面习题 11 快速排序算法是个递归算法,在使用非递归算法实现快速排序时,通常要利用一个栈来记忆待排序区间的两个端点位置。那么能否用队列来代替这个栈呢?答案是可以的。试在习题 11 快速排序算法的基础上,写出用栈实现快速排序的非递归程序和用队列实现快速排序的非递归程序。

数据存储类型说明

```
#define   DATATYPE1    int
#define   DATATYPE2    char
#define   KEYTYPE      int
#define   MAXSIZE      100
#define   MAXLEN       40
#define   VEXTYPE      int
#define   ADJTYPE      int
```

顺序表的存储结构的数据类型说明：

```
typedef struct
{ DATATYPE1 datas[MAXSIZE];
  int last;
}SEQUENLIST;
```

单链表的存储结构的数据类型说明：

```
typedef struct node
{
 DATATYPE2 data;
 struct node * next;
}LINKLIST;
```

双链表的存储结构的数据类型说明：

```
typedef struct dnode
{DATATYPE2 data;
 struct dnode * prior, * next;
} DLINKLIST;
```

顺序栈的存储结构的数据类型说明：

```
typedef struct
{ DATATYPE1 data[MAXSIZE];
```

```
  int top;
}SEQSTACK;
```

链栈的存储结构的数据类型说明：

```
typedef struct snode
   {  DATATYPE2 data;
      struct snode * next;
}LINKSTACK;
```

顺序队列的存储结构的数据类型说明：

```
typedef struct
   {  DATATYPE1 data[MAXSIZE];
      int front, rear;
}SEQQUEUE;
```

链队列的存储结构的数据类型说明：

```
typedef struct qnode
   {  DATATYPE1 data;
      struct qnode * next;
}LINKQLIST;

typedef struct
{ LINKQLIST * front, * rear;
}LINKQUEUE;
```

顺序串的存储结构的数据类型说明：

```
typedef struct
{  char ch[MAXSIZE];
   int len;
}SEQSTRING;
```

串的堆存储结构的数据类型说明：

```
typedef struct
{  char * ch;
   int len;
} HSTRING;
```

稀疏矩阵三元组顺序存放的数据类型说明：

```
typedef struct
{  int i, j;
   DATATYPE1 v;
}NODE;

typedef struct
```

```
{  int m, n, t;
   NODE data[MAXLEN];
}SPMATRIX;
```

二叉树的顺序存储结构类型说明：

```
typedef struct
{ DATATYPE2 bt[MAXSIZE];
  int btnum;
}BTSEQ;
```

二叉树的二叉链表存储结构类型说明：

```
typedef struct node1
{ DATATYPE2 data;
  struct node1 * lchild, * rchild;
}BTCHINALR;
```

二叉树的三叉链表(带双亲结点指针域)的存储结构类型说明：

```
typedef struct node2
{ DATATYPE2 data;
  struct node2 * lchild, * rchild, * parent;
}BTCHINALRP;
```

树的双亲表示法存储结构类型说明：

```
typedef struct
{  DATATYPE2 data;
   int parent;
}PTNODE;

typedef struct
{ PTNODE nodes[MAXSIZE];
  int nodenum;
}PTTREE;
```

树的孩子链表表示法存储结构类型说明：

```
typedef struct cnode
{ int child;
struct cnode * next;
}CHILDLINK;

typedef struct
{  DATATYPE2 data;
   CHILDLINK * headptr;
}CTNODE;
```

```
typedef struct
{CTNODE nodes[MAXSIZE];
 int nodenum, rootset;
}CTTREE;
```

树的孩子兄弟表示法二叉链表结构类型说明：

```
typedef struct csnode
{  DATATYPE2 data;
   struct csnode * firstchild, * nextsibling;
}CSNODE;
```

图的邻接矩阵存储结构说明：

```
typedef struct
{  otherdata …;   //图中边的信息,在下面的分析和讨论中忽略不考虑
   VEXTYPE vexs[MAXLEN];              //图中边的信息
   ADJTYPE arcs[MAXLEN][MAXLEN];      // 邻接矩阵
   int vexnum, arcnum;                //顶点数和边数
   int kind;                          //图的类型
                                      //有向图 1;无向图 2;有向网 3;无向网 4
}MGRAPH;
```

图的邻接链表存储结构说明：

```
typedef struct node3              //表结点结构
{  int adjvex;                    //存放与表头结点相邻接的顶点在数组中的序号
   struct node3 * next;           //指向与表头结点相邻接的下一个顶点的表结点
}EDGENODE;

typedef struct
{ VEXTYPE vertex;                 //存放图中顶的信息
  EDGENODE * link;                //指针指向对应的单链表中的表结点
} VEXNODE;

typedef struct
{ VEXNODE adjlist[MAXLEN];        //邻接链表表头向量
  int vexnum, arcnum;             //顶点数和边数
  int kind;                       //图的类型：
                                  //有向图 1;无向图 2;有向网 3;无向网 4
}ADJGRAPH;
```

顺序表上顺序查找数据类型说明：

```
typedef struct
{  KEYTYPE key;
   otherdata …;   //记录的其余数据部分,在下面的讨论和算法中忽略不考虑
}SSELEMENT;
```

```
typedef struct
{  SSELEMENT r[MAXSIZE];
   int len;
}SSTABLE;
```

采用二叉链表作为存储结构的二叉排序树,数据类型说明:

```
typedef struct node4
{KEYTYPE key;
 otherdata …;   //结点的其余数据部分,在下面的讨论和算法中忽略不考虑
struct node4 * lchild, * rchild;
} BSTNODE;
```

开散列表结构类型说明:

```
typedef struct node5
{
 KEYTYPE key;
 otherdata …;   //记录的其余数据部分,在下面的讨论和算法中忽略不考虑
 struct node5 * next;
}CHAINHASH;
```

闭散列表结构类型说明:

```
typedef struct
{
  KEYTYPE key;
  otherdata …;   //记录的其余数据部分,在下面的讨论和算法中忽略不考虑
}HASHTABLE;
```

待排记录的数据类型说明:

```
typedef struct
{ KEYTYPE key;
  otherdata …;   //记录的其余数据部分,在下面的讨论和算法中忽略不考虑
}RECNODE;
```

参考文献

[1] Horowitz E, Sahni S. Fundamentals of Data Structures, 1976

[2] 严蔚敏，吴伟民. 数据结构(第二版). 北京：清华大学出版社，1992

[3] 陈小平. 数据结构. 南京：南京大学出版社，1994

[4] 唐策善，李龙澍，黄刘生. 数据结构——用C语言描述. 北京：高等教育出版社，1995

[5] 杨秀金. 数据结构实用教程. 北京：科学出版社，1996

[6] 谭浩强. C程序设计. 北京：清华大学出版社，1998

[7] haffe C A 著. 数据结构及算法分析. 张铭，刘晓丹译. 北京：电子工业出版社，1998

[8] 李春葆. 数据结构(C语言篇)习题与解析. 北京：清华大学出版社，1999

[9] 张世和. 数据结构. 北京：清华大学出版社，2000

[10] 殷人昆. 数据结构. 北京：清华大学出版社，2001

[11] 李春葆. 数据结构教程学习指导. 北京：清华大学出版社，2005

本书特色

* 本书是普通高等教育“十一五”国家级规划教材“数据结构”的配套教材，内容上力求体现“以应用为主体”，强调理论知识的理解和运用。

* 对应“数据结构”教材中的每一章习题，给出习题的解析和参考答案，并且设计了一些没有题解的实训题供学生独立思考完成。

* 程序的源代码用的是 Visual C++ 中的C语言语句，可以在 Visual C++ 下编译运行，也可以在 DOS 操作系统及 Turbo C 软件环境下编译运行。

* 配书光盘中有教材课件和习题中编程题的源程序。

ISBN 978-7-302-16904-8
9 787302 169048 >
定价：21.00元（含光盘）