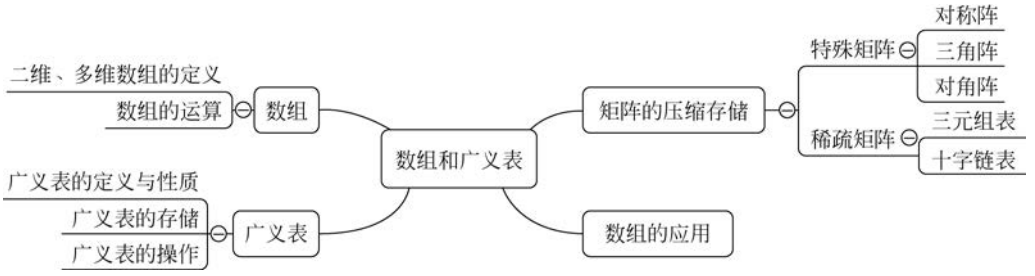
第5章 数组和广义表

5.1 基本知识提要

5.1.1 本章思维导图



5.1.2 常用术语解析

(1) 从逻辑结构上看,数组 A 是由 n(n>1)个相同类型数据元素 $a_1,a_2,\cdots,a_n$ 构成的有限序列,其逻辑表示为 $A=(a_1,a_2,\cdots,a_n)$,其中,$a_i(1\leqslant i\leqslant n)$表示数组 A 的第 i 个元素。

(2) 二维数组可以看作每个数据元素都是相同类型的一维数组。以此类推,任何多维数组都可以看作一个线性表,且表中的每个数据元素也是一个线性表。多维数组是线性表的推广。

(3) 特殊矩阵是指非零元素或者零元素的分布有一定规律的矩阵;反之,称为稀疏矩阵,即设矩阵 A 中有 s 个非零元素,若 s 远远小于矩阵元素的总数,则称 A 为稀疏矩阵。

(4) 广义表简称为表,它是线性表的推广。一个广义表是 n(n≥0)个元素的一个序列,若 n=0 时则称为空表。设 a_i 为广义表的第 i 个元素,则广义表 LS 的一般表示与线性表相同:$LS=(a_1,a_2,\cdots,a_i,\cdots,a_n)$。其中 n(n≥0)表示广义表的长度,即广义表中所含元素的个数。

(5) 如果广义表的元素 a_i 是单个数据元素,则 a_i 是广义表 LS 的原子;如果 a_i 是一个广义表,则 a_i 是广义表 LS 的子表。

5.1.3 重点知识整理

1. 数组的顺序存储结构

由于存储单元是一维的结构,而数组是个多维的结构,则用一组连续存储单元存放数组

的数据元素就有个次序约定问题。以一个 m 行 n 列的二维数组 A 为例，通常有两种顺序存储方式。

(1) 行优先顺序。将数组元素按行排列，即以行序为主序的存储方式，先存储第 1 行，然后存储第 2 行，最后存储第 m 行，则 A 的 m×n 个元素按行优先顺序存储的线性序列为：

$$a_{11},a_{12},\cdots,a_{1n},a_{21},a_{22},\cdots,a_{2n},\cdots,a_{m1},a_{m2},\cdots,a_{mn}$$

(2) 列优先顺序。采用以列为主序的存储方式，即先存储第一列数据元素，接着存储第二列数据元素，最后存储第 n 列数据元素，则 A 的 m×n 个元素按列优先顺序存储的线性序列为：

$$a_{11},a_{21},\cdots,a_{m1},a_{12},a_{22},\cdots,a_{m2},\cdots,a_{1n},a_{2n},\cdots,a_{mn}$$

对一个已知以行序为主序的计算机系统，假设每个数据元素占 L 个存储单元，则二维数组 A 中任一元素 a_{ij} 的存储位置可由下式确定：

$$LOC(a_{ij})=LOC(a_{00})+(i*n+j)*L$$

其中，$LOC(a_{ij})$是 a_{ij} 的存储位置，$LOC(a_{00})$是 a_{00} 的存储位置，即二维数组 A 的起始存储位置，也称为基地址。

同理，可推出在以列序为主序的计算机系统中有：

$$LOC(a_{ij})=LOC(a_{00})+(j*m+i)*L$$

2. 矩阵的压缩存储

(1) 特殊矩阵。

主要形式有对称矩阵、三角矩阵和对角矩阵。

① 若一个 n 阶方阵 A[n][n]中的元素满足 $a_{ij}=a_{ji}(0\leqslant i,j\leqslant n-1)$，则称其为 n 阶对称矩阵。由于对称矩阵中的元素关于主对角线对称，因此，在存储时可只存储对称矩阵中上三角或下三角中的元素，使得对称的元素共享一个存储空间。这样，就可以将 n^2 个元素压缩存储到 n(n+1)/2 个元素的空间中。

② 三角矩阵分为下三角矩阵和上三角矩阵两种。所谓下三角矩阵是指主对角线以上元素均为常数 c 或 0 的 n 阶矩阵；所谓上三角矩阵是指主对角线以下的元素均为常数 c 或 0 的 n 阶矩阵。对于这样的三角矩阵，同样也可采用对称矩阵的压缩存储方式将其上三角或下三角的元素存储在一维数组中，达到节约存储空间的目的。

③ 若一个 n 阶方阵 A[n][n]的所有非零元素都集中在以主对角线为中心的带状区域中，则称为 n 阶对角矩阵。即除了主对角线上和直接在对角线上、下方若干条对角线上的元素外，所有其他的元素为零。对这种矩阵，可按某个原则将其压缩存储到一维数组上。

(2) 稀疏矩阵。

① 稀疏矩阵的三元组顺序存储：按照压缩存储的概念，只存储稀疏矩阵的非零元素。除了存储非零元素的值外，还必须同时记下元素所在行和列的位置(i,j)，这样，一个三元组(i,j,a_{ij})唯一确定矩阵的一个非零元素。因此，稀疏矩阵又由表示非零元素的三元组及其行列数唯一确定。

② 稀疏矩阵的十字链表存储结构：用十字链表表示稀疏矩阵的基本思想是每个非零元素用一个结点存储，结点由 5 个域组成，如图 5.1 所示，其中，域 row 存储非零元素的行号；域 col 存储非零元素的列号；域 value

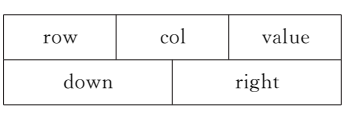


图 5.1　十字链表结点结构

存储元素的值；right 和 down 是两个指针域，right 指针指向同一行中下一个非零元素，down 指针指向同一列中下一个非零元素。

3. 广义表的存储

按结点形式的不同，广义表的链式存储结构又可以分为不同的两种存储方式：一种称为头尾表示法；另一种称为孩子兄弟表示法。

(1) 头尾表示法。

若广义表不空，则可分解成表头和表尾；反之，一对确定的表头和表尾可唯一地确定一个广义表。头尾表示法就是根据这一性质设计而成的一种存储方法。

由于广义表中的数据元素既可能是列表也可能是单元素，相应地在头尾表示法中结点的结构形式有两种：一种是表结点，用以表示列表；另一种是元素结点，用以表示单元素。在表结点中应该包括一个指向表头的指针和指向表尾的指针；而在元素结点中应该包括所表示单元素的元素值。为了区分这两类结点，在结点中还要设置一个标志域，如果标志为 1，则表示该结点为表结点；如果标志为 0，则表示该结点为元素结点。

(2) 孩子兄弟表示法。

在孩子兄弟表示法中，也有两种结点形式：一种是有孩子结点，用以表示列表；另一种是无孩子结点，用以表示单元素。在有孩子结点中包括一个指向第一个孩子(长子)的指针和一个指向兄弟的指针；而在无孩子结点中包括一个指向兄弟的指针和该元素的元素值。为了能区分这两类结点，在结点中还要设置一个标志域，如果标志为 1，则表示该结点为有孩子结点；如果标志为 0，则表示该结点为无孩子结点。

5.2 典型题解析


例 5.1

例 5.1 有一个 10 阶的对称矩阵 A，采用压缩存储方式以行序为主序存储，A[1][1]为第一个元素，其存储地址为 1，每个元素占一个地址空间，求 A[7][5]和 A[5][6]的地址。

例题解析：

按照以行序为主序的存储公式

$$Loc(i,j)=Loc(c1,c2)+[(i-c1)*(d2-c2+1)+(j-c2)]*L$$

则有：

Loc(A[7][5])=7(7−1)/2 + 5=26

Loc(A[5][6])=Loc(A[6][5])=6×(6−1)/2 + 5=20


例 5.2

例 5.2 二维数组 A[9][10]的元素都是 6 个字符组成的串，请回答下列问题：

(1) 存放 A 至少需要(　　)字节；

(2) A 的第 7 列和第 4 行共占(　　)字节；

(3) 若 A 按行存放，元素 A[7][4]的起始地址与 A 按列存放时(　　)元素的起始地址一致。

例题解析：

按照例题 5.1 给出的公式可知：

(1) 存放 A 需要 9×10×6=540(字节)。

(2) A 的第 7 列和第 4 行共占(9+10−1)×6=108(字节)。

(3) Loc (A[7][4])＝Loc (A[0][0]) ＋ [7×10＋4]×L　(按行序存储)。

Loc(A[i][j])＝Loc(A[0][0])＋[j×9＋i]×L　//(按列序存储,0≤i≤8,0≤j≤9)

所以,i＝2, j＝8。

即元素 A[7][4]的起始地址与 A 按列存放时 A[2] [8]的起始地址一致。

例 5.3　请创建一个稀疏矩阵 M 并利用十字链表存储表示。

例题解析:

```
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#typedef int Status;                /* Status 是函数的类型,其值是函数结果状态代码,如 OK 等 */
//存储结构定义
typedef struct OLNode
{
    int i, j;                                   /* 该非零元素的行和列下标 */
    ElemType e;                                 /* 非零元素值 */
    struct OLNode *right, *down;                /* 该非零元素所在行表和列表的后继链域 */
}OLNode, *OLink;
typedef struct
{
    OLink *rhead , *chead;          /* 行和列链表头指针向量基址,由 CreatSMatrix_OL()分配 */
    int mu, nu, tu;                             /* 稀疏矩阵的行数、列数和非零元素个数 */
}CrossList;
Status CreateSMatrix(CrossList *M)
{ /* 创建稀疏矩阵 M,采用十字链表存储表示 */
  int i, j, k, m, n, t;
    ElemType e;
    OLNode *p, *q;
    if((*M).rhead)
        DestroySMatrix(M);
    printf("请输入稀疏矩阵的行数、列数、非零元素个数: ");
    scanf("%d%d%d",&m,&n,&t);
    (*M).mu=m;
    (*M).nu=n;
    (*M).tu=t;
    (*M).rhead=(OLink*)malloc((m+1)*sizeof(OLink));
  if(!(*M).rhead)
     exit(OVERFLOW);
    (*M).chead=(OLink*)malloc((n+1)*sizeof(OLink));
    if(!(*M).chead)
        exit(OVERFLOW);
    for(k=1;k<=m;k++)                           /* 初始化行头指针向量; 各行链表为空链表 */
        (*M).rhead[k]=NULL;
    for(k=1;k<=n;k++)                           /* 初始化列头指针向量;各列链表为空链表 */
        (*M).chead[k]=NULL;
    printf("请按任意次序输入%d 个非零元素的行、列、元素值:\n",(*M).tu);
    for(k=0;k<t;k++)
    {
        scanf("%d%d%d",&i,&j,&e);
        p=(OLNode*)malloc(sizeof(OLNode));
        if(!p)
```

```
        exit(OVERFLOW);
    p->i=i;                                  /* 生成结点 */
    p->j=j;
    p->e=e;
     if((*M).rhead[i]==NULL||(*M).rhead[i]->j)  /* p插在该行的第一个结点处 */
     {
          p->right=(*M).rhead[i];
          (*M).rhead[i]=p;
     }
     else                                    /* 查询在行表中的插入位置 */
     {
       for(q=(*M).rhead[i];q->right&&q->right-<j;q=q->right)
            p->right=q->right;               /* 完成行插入 */
      q->right=p;
        }
     if((*M).chead[j]==NULL||(*M).chead[j]->i)  /* p插在该列的第一个结点处 */
      {
           p->down=(*M).chead[j];
           (*M).chead[j]=p;
      }
     else                                    /* 查询在列表中的插入位置 */
     {
        for(q=(*M).chead[j];q->down&&q->down-<i;q=q->down)
            p->down=q->down;                 /* 完成列插入 */
        q->down=p;
      }
  }
 return OK;   }
```

例 5.4 采用头尾链表存储结构，由广义表的书写形式串 S 创建广义表 L。

例题解析：

```
//Header.h
#ifndef HEADER_H
#define HEADER_H
//预定义常量和类型
//函数结果状态代码
#define TRUE 1
#define FALSE 0
#define OK 1
#define ERROR 0
#define INFEASIBLE -1
#define OVERFLOW -2
typedef int Status;
#endif

/**********************************************/
//GList.h
#ifndef GLIST_H
#define GLIST_H
#include <iostream>
#include "Header.h"

typedef char AtomType;                       //初始化
```

```
typedef enum {ATOM, LIST} ElemTag;        /* ATOM==0: 原子, LIST==1: 子表 */
typedef struct GLNode {
  ElemTag tag;                            /* 公共部分,用于区分原子结点和表结点 */
    union { /* 原子结点和表结点的联合部分 */
        AtomType atom;                    /* atom 是原子结点的值域,AtomType 由用户定义 */
        struct {
            struct GLNode *hp, *tp;
        } ptr;                            /* ptr 是表结点的指针域,ptr.hp 和 ptr.tp 分别指向表头和表尾 */
    };
} *GList, GLNode;                         /* 广义表类型 */
/* 创建空的广义表 L */
void InitGList(GList &L)
{
    L = NULL;
}
/* 采用头尾链表存储结构,由广义表的书写形式串 S 创建广义表 L.设 emp="()" */
void CreateGList(GList &L, string S)
{
    string sub, hsub, emp("()");
    GList p, q;
    if (S == emp)
        L = NULL;                         /* 创建空表 */
    else { /* S 不是空串 */
        L = (GList)malloc(sizeof(GLNode));
        if (!L)
            exit(OVERFLOW);
        if (S.length() == 1) { /* S 为单原子,只会出现在递归调用中 */
            L->tag = ATOM;
            L->atom = S[0];               /* 创建单原子广义表 */
            }
      else { /* S 为表 */
                L->tag = LIST;
                p = L;
                sub = S.substr(1, S.length() - 2);
                                  /* 脱外层括号(去掉第一个字符和最后一个字符)给串 sub */
              do { /* 重复建 n 个子表 */
                    Sever(sub, hsub);     /* 从 sub 中分离出表头串 hsub */
                    CreateGList(p->ptr.hp, hsub);
                    q = p;
                    if(!sub.empty()) { /* 表尾不空 */
                        p = (GLNode *)malloc(sizeof(GLNode));
                        if (!p)
                            exit(OVERFLOW);
                        p->tag = LIST;
                        q->ptr.tp = p;
                    }
                } while (!sub.empty());
                q->ptr.tp = NULL;
            }
        }
}
```

例 5.5　将一个 10×10 的对称矩阵 M 的上三角部分元素 m_{ij}(1≤i≤j≤10)按列优先存入 C 语言的一维数组 N 中,元素 $m_{7,2}$ 在 N 中的下标是(　　)。【2020 年研究生联考

题目】

A. 15　　B. 16　　C. 22　　D. 23

例题解析：由于对称矩阵 M 只存储上三角元素，而 $m_{7,2}$ 为下三角元素，因此存储的是它对应的上三角元素 $m_{2,7}$。

由于 M 是按列优先存储到一维数组中的，前 6 列共有(1+6)×6÷2=21 个元素，$m_{2,7}$ 是第 7 列的第 2 个元素，因此元素 $m_{2,7}$ 的序号为 21+2=23。C 语言的数组下标从 0 开始计数，因此元素 $m_{7,2}$ 在 N 中的下标是 22。故答案为 C。

5.3 知识拓展

数组和广义表是线性表吗？它们和第 2 章中介绍的线性表有什么区别？

一个二维数组可以看作每个数据元素都是相同类型的一维数组的一维数组。以此类推，任何多维数组都可以看作一个线性表，这时线性表中的每个数据元素也是一个线性表。因此，多维数组是线性表的推广。推广到 d(d≥3)维数组，不妨把它看作一个由 d−1 维数组作为数据元素的线性表；或者这样理解，它是一种较复杂的线性表结构，由简单的数据结构，即线性表——辗转合成而得。

广义表是一种非线性的数据结构，它也是线性表的一种推广。线性表被定义为一个有限的序列(a1,a2,a3,…,an)，其中 ai 被限定为是单个数据元素。广义表也是 n 个数据元素 d1,d2,d3,…,dn 的有限序列，但不同的是，广义表中的 di 则既可以是单个元素，又可以是一个广义表，通常记作：GL=(d1,d2,d3,…,dn)。广义表被广泛地应用于人工智能等领域的表处理语言 LISP 中，在 LISP 语言中，广义表是一种最基本的数据结构，就连 LISP 语言的程序也表示为一系列的广义表。

5.4 测试习题与参考答案

测试习题

一、填空题

1. 一维数组的逻辑结构是(　　)，存储结构是(　　)。

2. 对于二维数组或多维数组，分为按(　　)和按(　　)两种不同的存储方式存储。

3. 二维数组 A[c1..d1,c2..d2]共含有(　　)个元素。

4. 二维数组 A[10][20]采用列序为主方式存储，每个元素占一个存储单元，且 A[0][0]的地址是 200，则 A[6][12]的地址是(　　)。

5. 有一个 10 阶对称矩阵 A，采用以行为主序的压缩存储方式，A[0][0]的地址为 1，则 A[8][5]的地址是(　　)。

6. 广义表运算式 HEAD(TAIL((a,b,c),(x,y,z)))的结果为(　　)。

7. 所谓稀疏矩阵指的是(　　)。

8. 对矩阵压缩是为了(　　)。

9. 上三角矩阵压缩的下标对应关系为(　　)。

10. 当广义表中的每个元素都是原子时，广义表便成了(　　)。

11. 广义表的(　　)定义为广义表中括弧的重数。

12. 设广义表 L=((),()),则 head(L)是(　　),tail(L)是(　　),L 的长度是(　　),深度是(　　)。

13. 广义表 A=(((a,b),(c,d,e))),取出 A 中的原子 e 的操作是(　　)。

14. 广义表(a,(a,b),d,e,((i,j),k))的长度是(　　),深度是(　　)。

15. 已知广义表 LS=(a,(b,c,d),e),运用 head 和 tail 函数取出 LS 中原子 b 的运算是(　　)。

二、选择题

1. 数组 A 中，每个元素的长度为 3 字节，行下标 I 从 1 到 8，列下标 J 从 1 到 10，从首地址 SA 开始连续存放在存储器内，该数组占用的字节数为(　　)。

A. 80　　B. 100　　C. 240　　D. 270

2. 数组 A 中，每个元素的长度为 3 字节，行下标 I 从 1 到 8，列下标 J 从 1 到 10，从首地址 SA 开始连续存放在存储器内，该数组按行存放时，元素 A[8][5]的起始地址为(　　)。

A. SA+141　　B. SA+144　　C. SA+222　　D. SA+225

3. 一个 n×n 的对称矩阵，如果以行或列为主序放入内存，则其容量为(　　)。

A. n×n　　B. n×n/2

C. (n+1)×n/2　　D. (n+1)×(n+1)/2

4. 稀疏矩阵一般的压缩存储方法有两种，即(　　)。

A. 二维数组和三维数组　　B. 三元组和哈希

C. 三元组和十字链表　　D. 哈希和十字链表

5. 设有广义表 D=(a, b, D)，则其长度为(　　)，深度为(　　)。

A. 1　　B. 3　　C. ∞　　D. 5

6. 广义表运算式(Tail((a,B),(c,d)))的操作结果是(　　)。

A. (c,d)　　B. c,d　　C. ((c,d))　　D. d

7. 设有一个 10 阶的对称矩阵 A，采用压缩存储方式，以行序为主存储，a_{11} 为第一元素，其存储地址为 1，每个元素占一个地址空间，则 a_{85} 的地址为(　　)。

A. 13　　B. 33　　C. 18　　D. 40

8. 设有数组 A[i,j]，数组的每个元素长度为 3 字节，i 的值为 1 到 8，j 的值为 1 到 10，数组从内存首地址 BA 开始顺序存放，当用以列为主序存放时，元素 A[5,8]的存储首地址为(　　)。

A. BA+141　　B. BA+180　　C. BA+222　　D. BA+225

9. 假设以行序为主序存储二维数组 A=array[1..100,1..100]，设每个数据元素占 2 个存储单元，基地址为 10，则 LOC[5,5]=(　　)。

A. 808　　B. 818　　C. 1010　　D. 1020

10. 数组 A[0..5,0..6]的每个元素占 5 字节，将其按列优先次序存储在起始地址为 1000 的内存单元中，则元素 A[5,5]的地址是(　　)。

A. 1175　　B. 1180　　C. 1205　　D. 1210

11. 二维数组 A 的每个元素是由 6 个字符组成的串，其行下标 i=0,1,…,8，列下标 j=

1,2,…,10。若A按行先存储,元素A[8,5]的起始地址与当A按列先存储时的元素(　　)的起始地址相同。设每个字符占一个字节。

A. A[8,5]　　B. A[3,10]　　C. A[5,8]　　D. A[0,9]

12. 若对n阶对称矩阵A以行序为主序方式将其下三角形的元素(包括主对角线上的所有元素)依次存放于一维数组B[1..(n(n+1))/2]中,则在B中确定a_{ij}(i<j)的位置k的关系为(　　)。

A. i×(i−1)/2+j　　B. j×(j−1)/2+i

C. i×(i+1)/2+j　　D. j×(j+1)/2+i

13. 设A是n×n的对称矩阵,将A的对角线及对角线上方的元素以列为主序的次序存放在一维数组B[1..n(n+1)/2]中,对上述任一元素a_{ij}(1≤i,j≤n,且i≤j)在B中的位置为(　　)。

A. i*(i−1)/2+j　　B. j*(j−1)/2+i

C. j*(j−1)/2+i−1　　D. i*(i−1)/2+j−1

14. 用数组r存储静态链表,结点的next域指向后继,工作指针j指向链中结点,使j沿链移动的操作为(　　)。

A. j=r[j].next　　B. j=j+1

C. j=j−>next　　D. j=r[j]−>next

15. 已知广义表L=((x,y,z),a,(u,t,w)),从L表中取出原子项t的运算是(　　)。

A. head(tail(tail(L)))　　B. tail(head(head(tail(L))))

C. head(tail(head(tail(L))))　　D. head(tail(head(tail(tail(L)))))

16. 已知广义表LS=((a,b,c),(d,e,f)),运用head和tail函数取出LS中原子e的运算是(　　)。

A. head(tail(LS))　　B. tail(head(LS))

C. head(tail(head(tail(LS)))　　D. head(tail(tail(head(LS))))

17. 广义表运算式Tail(((a,b),(c,d)))的操作结果是(　　)。

A. (c,d)　　B. c,d　　C. ((c,d))　　D. d

18. 广义表L=(a,(b,c)),进行Tail(L)操作后的结果为(　　)。

A. c　　B. b,c　　C. (b,c)　　D. ((b,c))

19. 下面说法不正确的是(　　)。

A. 广义表的表头总是一个广义表　　B. 广义表的表尾总是一个广义表

C. 广义表难以用顺序存储结构　　D. 广义表可以是一个多层次的结构

20. 设广义表L=((a,b,c)),则L的长度和深度分别为(　　)。

A. 1和1　　B. 1和3　　C. 1和2　　D. 2和3

21. 已知二维数组A按行优先方式存储,每个元素占用1个存储单元,若元素A[0][0]的存储地址是100,A[3][3]的存储地址是220,则元素A[5][5]的存储地址是(　　)。【2021年研究生联考题目】

A. 295　　B. 300　　C. 301　　D. 306

22. 适用于压缩存储稀疏矩阵的两种存储结构是(　　)。【2017年研究生联考题目】

A. 三元组表和十字链表　　B. 三元组表和邻接矩阵

C. 十字链表和二叉链表　　D. 邻接矩阵和十字链表

三、判断题

1. 数组中存储的数可是任意类型的任何数据。（　　）

2. N×N 对称矩阵经过压缩存储后占用的存储单元是原先的 1/2。（　　）

3. 稀疏矩阵在用三元组表示法时，可节省空间，但对矩阵的操作会增加算法的难度及耗费更多的时间。（　　）

4. 广义表不是线性表。（　　）

5. Tail(a，b，c，d)得到的是(b，c，d)。（　　）

6. 数组不适合作为任何二叉树的存储结构。（　　）

7. 从逻辑结构上看，n 维数组的每个元素均属于 n 个向量。（　　）

8. 稀疏矩阵压缩存储后，必会失去随机存取功能。（　　）

9. 数组是同类型值的集合。（　　）

10. 数组可看成线性结构的一种推广，因此与线性表一样，可以对它进行插入、删除等操作。（　　）

11. 一个稀疏矩阵 $A_{m\times n}$ 采用三元组形式表示，若把三元组中有关行下标与列下标的值互换，并把 m 和 n 的值互换，则就完成了 $A_{m\times n}$ 的转置运算。（　　）

12. 广义表的取表尾运算，其结果通常是一个表，但有时也可是一个单元素值。（　　）

13. 若一个广义表的表头为空表，则此广义表也为空表。（　　）

14. 广义表中的元素或者是一个不可分割的原子，或者是一个非空的广义表。（　　）

15. 广义表的同级元素(直属于同一个表中的各元素)具有线性关系。（　　）

四、应用题

1. 数组 A[8][6][9]以行序为主序存储，设第一个元素的首地址为 54，每个元素长度为 5，求元素 A[2][4][5]的存储地址。

2. 数组 A 中，每个元素 A[i][j]的长度均为 32 个二进位，行下标从－1 到 9，列下标从 1 到 11，从首地址 s 开始连续存放主存储器中，主存储器字长为 16 位。求：

(1) 存放该数组需多少单元？

(2) 存放数组第 4 列所有元素至少需多少单元？

(3) 数组按行存放时，元素 A[7][4]的起始地址是多少？

(4) 数组按列存放时，元素 A[4][7]的起始地址是多少？

3. 将一个 A[1..100][1..100]的三对角矩阵，按行优先存入一维数组 B[1..m]中，试确定 m 的值，并求 A 中元素 A[77][78](即元素下标 i=77，j=78)在 B 数组中的位置 k。

4. 设 m×n 阶稀疏矩阵 A 有 t 个非零元素，其三元组表表示为 LTMA[1..(t+1)][1…3]，试问：非零元素的个数 t 达到什么程度时用 LTMA 表示 A 才有意义？

5. 设有上三角矩阵 $(a_{ij})_{n\times n}$，将其上三角中的元素按先行后列的顺序存于数组 B(1..m)中，使得 B[k]＝a_{ij} 且 k＝f1(i)＋f2(j)＋c，请推导出函数 f1、f2 和常数 c，要求 f1 和 f2 中不含常数项。

6. 设有广义表 K1(K2(K5(a,K3(c,d,e)),K6(b,k)),K3,K4(K3,f))，要求：

(1) 指出 K1 的各个元素的构成；

(2) 计算 K1、K2、K3、K4、K5、K6 的长度和深度。

7. 画出下列广义表的链接存储结构，并求其深度：(( ), a, ((b, c), ( ), d), (((e)))))。

8. 求下列广义表运算的结果。

(1) GetHead ((p,h,w))

(2) GetTail((b,k,p,h))

(3) GetHead(GetTail(((a,b),(c,d))))

(4) GetTail(GetHead(((a,b),(c,d))))

9. 用三元数组表示稀疏矩阵的转置矩阵，并简要写出解题步骤。

10. (1) 已知广义表 L=((((a))),((b)),(c),d)，试利用 head 和 tail 运算把原子项 c 从 L 中分离出来。

(2) 画出下列广义表的存储结构图，并利用取表头和取表尾的操作分离出原子 e。

(a,(),b),(((e))))

(3) 已知广义表 A=((a,b,c),(d,e,f))，试写出从表 A 中取出原子元素 e 的运算。

(4) 请将香蕉 banana 用工具 H( )—Head( )，T( )—Tail( )从 L 中取出。

L=(apple,(orange,(strawberry,(banana)),peach),pear)

(5) 试利用广义表取表头 head(ls)和取表尾 tail(ls)的基本运算，将原子 d 从下列表中分解出来，请写出每一步的运算结果。

L=((a,(b)),((c,d)),(e,f))

(6) 画出广义表 A=(a,(b,()),(((),c)))的第一种存储结构(表结点第二指针指向余表)图，并用取首元(head())和取尾元(tail())函数表示原子 c。

11. 请按行及按列优先顺序列出 4 维数组 $A_{2\times3\times2\times3}$ 的所有元素在内存中的存储次序，开始结点为 a0000。

12. 给出 C 语言的三维数组地址计算公式。

13. 设有三角矩阵 $A_{n\times n}$，将其三条对角线上的元素逐行地存储到向量 B[0..3n−3]中，使得 B[k]=a_{ij}，求：

(1) 用 i,j 表示 k 的下标变换公式。

(2) 用 k 表示 i,j 的下标变换公式。

14. 设二维数组 A[5,6]的每个元素占 4 字节，已知 Loc(a00)=1000，A 共占多少字节？A 的终端结点 a45 的起始地位为何？按行和按列优先存储时，a25 的起始地址分别为何？

15. 特殊矩阵和稀疏矩阵哪一种压缩存储后会失去随机存取的功能？为什么？

16. 画出下列广义表的图形表示：

(1) A(a,B(b,d),C(e,B(b,d),L(f,g)))　　(2) A(a,B(b,A))

17. 设广义表 L=((),())，试问 head(L)、tail(L)和 L 的长度、深度各为多少？

18. 求下列广义表运算的结果。

(1) head((p,h,w))　　(2) tail((b,k,p,h))

(3) head(((a,b),(c,d)))　　(4) tail(((a,b),(c,d)))

(5) head(tail(((a,b),(c,d))))　　(6) tailhead((((a,b),(c,d))))

五、算法设计题

1. 在数组 A[1..n]中有 n 个数据，试建立一个带有头结点的循环链表，头指针为 h，要

求链中数据从小到大排列，重复的数据在链中只保存一个。

2. 请编写递归算法，逆转广义表中的数据元素。例如：将广义表(a,((b, c),()),(((d),e),f))逆转为((f,(e,(d))),((),(c, b)),a)。

3. 编写一个过程，对一个 n×n 矩阵，通过行变换，使其每行元素的平均值按递增顺序排列。

4. 给定一个整数数组 b[0..N−1]，b 中连续的相等元素构成的子序列称为平台。试设计算法，求出 b 中最长平台的长度。

5. 给定 n×m 矩阵 A[a..b,c..d]，并设 A[i,j]≤A[i,j+1](a≤i≤b,c≤j≤d−1)和 A[i,j]≤A[i+1,j](a≤i≤b−1,c≤j≤d)。设计一算法判定 X 的值是否在 A 中，要求时间复杂度为 O(m+n)。

6. 对于二维整型数组 A[m,n]，分别编写相应函数实现如下功能。

(1) 求数组 A[5,7]边元素之和。

(2) 当 m=n 时分别求两条对角线上的元素之和，否则显示 m≠n 的信息。

7. 编写函数，将一维数组 A[n×n](n≤10)中的元素按蛇形方阵存放在二维数组 B[n][n]中，即 B[0][0]=A[0]，B[0][1]=A[1]，B[1][0]=A[2]，B[2][0]=A[3]，B[1][1]=A[4]，B[0][3]=A[6]，以此类推。

8. 编写一个函数将两个广义表合并成一个广义表。

9. 当三角矩阵采用转置矩阵压缩存储时，写一算法求三角矩阵在这种压缩存储表示下的转置矩阵。

10. 当稀疏矩阵 A 和 B 均以三元组表作为存储结构时，试写出矩阵相加的算法，其结果存放在三元组表 C 中。

参考答案

一、填空题

1. 线性结构　顺序结构

2. 以行为主序　以列为主序

3. (d1−c1+1)×(d2−c2+1)

4. 326

5. 42

6. (x,y,z)

7. 非零元素很少(t<<m×n)且分布没有规律

8. 节省存储空间

9. k=((i−1)×(2n−i+2))/2+(j−i+1)=(i−1)(2n−i)/2+j　(i≤j)

10. 线性表

11. 深度

12. ()　(())　2　2

13. head(tail(tail(head(tail(head(A))))))

14. 5　3

15. head(head(tail(LS)))

二、选择题

1. C　2. C　3. C　4. C　5. B　C　6. C　7. B　8. B　9. B

10. A　11. B　12. B　13. B　14. A　15. D　16. C　17. C

18. D　19. A　20. C　21. B　22. A

三、判断题

1. ×　2. ×　3. √　4. ×　5. √　6. ×　7. √　8. √　9. ×　10. ×

11. ×　12. ×　13. ×　14. ×　15. √

四、应用题

1. Loc(2,4,5)＝Loc(0,0,0)＋(2×6×9＋4×9＋5)＝54 ＋ 149×5＝799。

2. (1) 121×32/16＝242。

(2) 11×32/16＝22。

(3) LOC(A[0][0])＋(8×11＋3)×32/16＝LOC(A[0][0])＋182。

(4) LOC(A[0][0])＋182。

3. 三角矩阵共 3n－2 个元素，存入 B[1..3n－2]中，元素在一维数组 B 中的下标 k 和元素在矩阵中的下标 i 和 j 的对应关系为：

```
k=3(i-1)                    //主对角线左下角,即 i=j+1
k=3(i-1)+1                  //主对角线上,即 i=j
k=3(i-1)+2                  //主对角线上,即 i=j-1
```

由以上三式得

```
k=2(i-1)+j                  //1≤i, j≤n; 1≤k≤3n-2
```

故 A[77,78](即元素下标 i＝77,j＝78)在 B 数组中的位置为 k＝2×(77－1)＋78＝230。

4. 稀疏矩阵 A 有 t 个非零元素，加上行数 mu、列数 nu 和非零元素个数 tu(也算一个三元组)，共占用三元组表 LTMA 的 3(t＋1)个存储单元，用二维数组存储时占用 m×n 个单元，只有当 3(t＋1)＜m×n 时，用 LTMA 表示 A 才有意义。解不等式得 t＜m×n/3－1。

5. 上三角矩阵第一行有 n 个元素，第 i－1 行有 n－(i－1)＋1 个元素，第一行到第 i－1 行是等腰梯形，而第 i 行上第 j 个元素(即 a_{ij})是第 i 行上第 j－i＋1 个元素，故元素 a_{ij} 在一维数组中的存储位置(下标 k)为：

$$k=(n+(n-(i-1)+1))(i-1)/2+(j-i+1)=(2n-i+2)(i-1)/2+j-i+1$$

即

$$k=(n-1/2)i-n+j+1/2$$

则得

$$f_1(i)=(n+1/2)i-n+j+1/2,\quad f_2(j)=j,c=-n$$

6. (1) K1 由 K2,K3,K4 构成。

(2)

	K1	K2	K3	K4	K5	K6
长度：	3	2	3	2	2	2
深度：	4	3	1	2	2	1

7. 深度为 4 的广义链表存储结构如图 5.2 所示。

8. (1) GetHead ((p, h, w))＝p

(2) GetTail((b, k, p, h))＝(k, p, h)

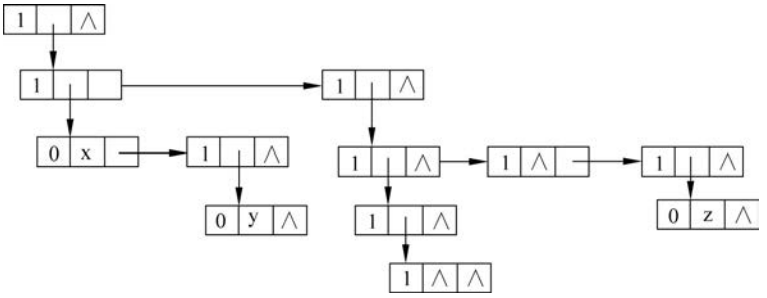


图 5.2　深度为 4 的广义链表存储结构

(3) GetHead(GetTail(((a,b),(c,d))))=GetHead(((c, d)))=(c, d)

(4) GetTail(GetHead(((a,b),(c,d))))=GetTail((a, b))=(b)

9. 设用 mu、nu 和 tu 表示稀疏矩阵行数、列数和非零元素个数，则转置矩阵的行数、列数和非零元素的个数分别是 nu、mu 和 tu。转置可按转置矩阵的三元组表中的元素顺序进行，即按稀疏矩阵的列序，从第 1 列到第 nu 列，每列中按行值递增顺序，找出非零元素，逐个放入转置矩阵的三元组表中，转置时行列值互换，元素值复制。按这种方法，第 1 列～第 1 个非零元素一定是转置后矩阵的三元组表中的第 1 个元素，第 1 列非零元素在第 2 列非零元素的前面。这种方法的时间复杂度是 O(n×p)，其中 p 是非零元素的个数，当 p 和 m×n 同量级时，时间复杂度为 $O(n^3)$。

另一种转置方法称作快速转置，使时间复杂度降为 O(m×n)。它是按稀疏矩阵三元组表中元素的顺序进行的。按顺序取出一个元素，放到转置矩阵三元组表的相应位置。这就要求出每列非零元素个数和每列第一个非零元素在转置矩阵三元组表中的位置，设置了两个附加向量。

10. (1) head(head(tail(tail(L)))),设 L=((a,(c),b),(((e))))

(2) head(head(head(head(tail(tail(L))))))(如图 5.3 所示)

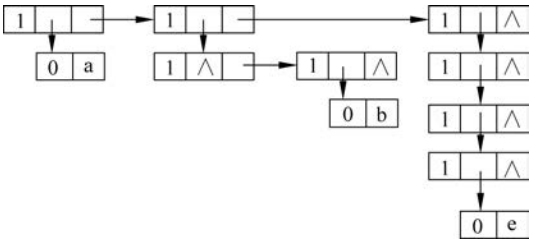


图 5.3　广义链表存储结构

(3) head(tail(head(tail(A))))

(4) H(H(T(H(T(H(T(L)))))))

(5) tail(L)=(((c,d)),(e,f))

head(tail(L))=((c,d))

head(head(tail(L)))=(c,d)

tail(head(head(tail(L))))=(d)

head(tail(head(head(tail(L)))))=d

(6) head(tail(head(head(tail(tail(A))))))

11. 按行优先的顺序排列时，先变化右边的下标，也就是右到左依次变化，这个 4 维数组的排列是这样的(将这个排列分行写出以便阅读，只要按从左到右的顺序存放就是在内存中的排列位置)：

a0000　a0001　a0002
a0010　a0011　a0012
a0100　a0101　a0102
a0110　a0111　a0112
a0200　a0201　a0202
a0210　a0211　a0212
a1000　a1001　a1002
a1010　a1011　a1012
a1100　a1101　a1102
a1110　a1111　a1112
a1200　a1201　a1202
a1210　a1211　a1212

按列优先的顺序排列恰恰相反，变化最快的是左边的下标，然后向右变化，所以这个 4 维数组的排列将是这样的(这里为了便于阅读，也将其书写为分行形式)：

a0000　a1000
a0100　a1100
a0200　a1200
a0010　a1010
a0110　a1110
a0210　a1210
a0001　a1001
a0101　a1101
a0201　a1201
a0011　a1011
a0111　a1111
a0211　a1211
a0002　a1002
a0102　a1102
a0202　a1202
a0012　a1012
a0112　a1112
a0212　a0212

12. 因为 C 语言的数组下标下界是 0，所以

$$Loc(Amnp)=Loc(A000)+((i\times n\times p)+k)\times d$$

其中 Amnp 表示三维数组，Loc(A000)表示数组起始位置，i、n、p、k 表示当前元素的下标，d 表示每个元素所占单元数。

13. (1) 要求 i、j 到 k 的下标变换公式，就是要知道在 k 之前已有几个非零元素，这些非零元素的个数就是 k 的值，一个元素所在行为 i，所在列为 j，则在其前面已有的非零元素个数为：

$$(i\times 3-1)+j-(i+1)$$

其中(i×3－1)是这个元素前面所有行的非零元素个数，j－(i＋1)是它所在列前面的非零元素个数。

化简可得：

```
k=2i+j                          // c 下标是从 0 开始的
```

(2) 因为 k 和 i，j 是一一对应的关系，因此不难算出：

```
i=(k+1)/3                  //k+1 表示当前元素前有几个非零元素，被 3 整除就得到行号
j=(k+1)%3+(k+1)/3-1        //k+1 除以 3 的余数就表示当前行中第几个非零元素
                           //加上前面的零元素所在列数就是当前列号
```

14. (1) 因含 5×6＝30 个元素，因此 A 共占 30×4＝120(字节)。

(2) a45 的起始地址为：Loc(a45)＝Loc(a00)＋(i×n＋j)×d＝1000＋(4×6＋5)×4＝1116。

(3) 按行优先顺序排列时，a25＝1000＋(2×6＋5)×4＝1068。

(4) 按列优先顺序排列时(二维数组可用行列下标互换来计算)：

$$a25=1000+(5\times 5+2)\times 4=1108$$

15. 稀疏矩阵在采用压缩存储后将会失去随机存储的功能。因为在这种矩阵中，非零元素的分布是没有规律的，为了压缩存储，就将每一个非零元素的值和它所在的行、列号作为一个结点存放在一起，这样的结点组成的线性表叫作三元组表，它不是简单的向量，所以无法用下标直接存取矩阵中的元素。

16. 图 5.4 是一个再入表。

图 5.5 则是一个递归表。

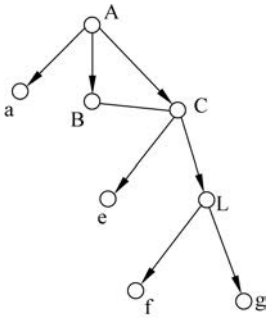


图 5.4　再入表

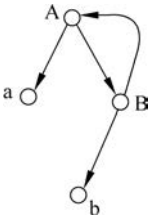


图 5.5　递归表

17. head(L)＝()。

tail(L)＝(())。

L 的长度为 2。

L 的深度为 2。

18.

(1) head ((p,h,w))＝p

(2) tail((b,k,p,h))＝(k,p,h)

(3) head (((a,b),(c,d)))=(a,b)

(4) tail(((a,b),(c,d)))=((c,d))

(5) head(tail(((a,b),(c,d))))=(c,d)

(6) tail(head(((a,b),(c,d))))=(b)

五、算法设计题

1.

```
LinkedList creat(ElemType A[ ], int n)
{h=(LinkedList)malloc(sizeof(LNode));
 h->next=h;
 for(i=0;i<n;i++)
  {pre=h;
   p=h->next;
   while(p!=h && p->data<A[i])
  {pre=p; p=p->next;}
   if(p==h || p->data!=A[i])
       {s=(LinkedList)malloc(sizeof(LNode));
      s->data=A[i]; pre->next=s; s->next=p;
      }
   }
  return(h);
   }
```

2.

```
Void GListInvert(GList p, GList t)
 {GList q=null;
          while(p)
              {if(p->tag!=0)
                    {m=p->val.hp;
                     GListInvert(m,n) ;
                      p->val.hp=n;
                      }
                  r=p->tp;
                  p->tp=q;
                  q=p;
                  p=r ;
              }
         t=q;
 }
```

3.

```
void Translation(float *matrix,int n)
{int i, j, k, l;
float sum,min;
float *p, *pi, *pk;
for(i=0; i<n; i++)
  { sum=0.0; pk=matrix+i*n;
    for (j=0; j<n; j++){sum+=*(pk); pk++;}
     *(p+i)=sum;
  }
for(i=0; i<n-1; i++)
```

```
    { min=*(p+i); k=i;
      for(j=i+1;j<n;j++) if(p[j]<min) {k=j; min=p[j];}
      if(i!=k)
     { pk=matrix+n*k;
       pi=matrix+n*i;
       for(j=0;j<n;j++)
         {sum=*(pk+j); *(pk+j)=*(pi+j); *(pi+j)=sum;}
       sum=p[i]; p[i]=p[k]; p[k]=sum;
     }
    }
  free(p);
}
```

4.

```
void Platform (int b[ ], int N)                    //求具有 N 个元素的整型数组 b 中最长平台的长度
{l=1;k=0;j=0;i=0;
 while(i<n-1)
{   while(i<n-1 && b[i]==b[i+1]) i++;
    if(i-j+1>l) {l=i-j+1;k=j;}                     //局部最长平台
    i++; j=i; }                                    //新平台起点
  printf("最长平台长度%d,在 b 数组中起始下标为%d", l, k);
}                                                  //Platform
```

5.

```
void search(datatype A[ ][ ], int a, int b, int c, int d, datatype x)
//n*m 矩阵 A,行下标从 a 到 b,列下标从 c 到 d,本算法查找 x 是否在矩阵 A 中
{i=a; j=d; flag=0;                                 //flag 是成功查到 x 的标志
  while(i<=b && j>=c)
    if(A[i][j]==x) {flag=1;break;}
    else if(A[i][j]>x) j--;
         else i++;
   if(flag) printf("A[%d][%d]=%d",i,j,x);          //假定 x 为整型
   else printf("矩阵 A 中无%d 元素", x);
  }算法 sear 结束
```

6.

(1)

```
#define M 5
#define N 7
long sum side (int a[M] [N])
{
long sum=0;
int i;
 for(i=0; i<M; i++)
       sum+=a[i][0];
 for(i=1; i<N; i++)
       sum+=a[O][i];
 for (i=1; i<M; i++)
       sum+=a[i] [N-1];
 for (i=l; i<N-1; i++)
       sum+=a[M-1] [i];
```

```
 return sum;
}
```

(2)

```
 long sum tilt (int a[M][N])
 {
 long sum=0;
 int i, j ;
if (M!=N)
 {
    printf ("\nM 不等于 N\n");
    return 0;
}
for (i=0; i<M; i++)
  sum+=a[i][i];
for (i=M-1; i>=0; i--)
  sum+=a[i][M-i-l];
if (M%2!=0)
  sum-= a[M/2] [M/2];
  return (sum);
}
```

7.

```
#define NMAX 20
int a[NMAX * NMAX],b[NMAX] [NMAX],cnt;
void MakeLine (int RowStart, int ColStart, int RowEnd)
{
     int i,j,step;
     if (RowStart <=RowEnd)
          step=l;
     else
          step=-l;
     for (i=RowStart, j=ColStart; (RowEnd-i) * step>=0 ; i+=.s tep, j -=step)
        b[i][j]=a[cnt++];
  }
  void MakeArray (int n)
  {
    int d:
    for (d=0;d<2 * n; d++)
      if(d<n)
           if (d%2= =0)
              MakeLine(d,0,0);
           else
              MakeLine(0,d,d);
      else
           if ( d%2= =0)
              Makeline (n-l, d-n+l, d-n+l);
          else
              MakeLine (d-n+l, n-l, n-l);
  }
  main()
  {
    int i, j, n;
    printf ("\nPlease input n: ");
    scanf("%d",&n);
```

```
    for (i=0;i<n*n; i++)
        a[i]=i+1;
    cnt=0;
    MakeArray (n);
    for(i=0;i<n;i++)
      {
        printf("\n");
        for (j=0; j<n; j++)
            printf("%5d",b[i][j]);
      }
    }
```

8.

```
int equal(GListNode *ha, GListNode *hb);
  void GListInsertTail(GListNode *h, GListNode *s);
GListNode *GListMerge (GListNode *ha, GListNode *hb)
{
GListNode p,q,r;
if (ha==NULL)
  {
     ha=hb;
     return ha;
  }
else
 {
    p=hb->val. sublist;
    while(p!=NULL)
     {
        q=ha->val.sublist;
        while (q!=NULL)
          {
              if (equal (p,q))
                  break;
              q=q->link;
          }
        if (q==NULL)
          {
              r=p->link;
             GListInsertTail (ha,p);
             p=r;
          }
      else
             p=p->link;
     }
    }
return ha;
  }
int equal (GListNode *ha, GlistNode *hb)
{
GListNode *p, *q ;
if (ha==NULL)
      if (hb==NULL)
          return 1;
      else
```

```
                return 0;
if (ha->tag! =hb->tag)
          return 0;
if(ha->tag==-0)
            if (ha->val. data==hb->val. data)
                    return l;
             else
                    return 0;
     else
     {
           p=ha->val. sublist;
           q=hb->val.sublist;
           while(p!-NULL&&ql =NULL&&equal (P,q))
               {
                  p=p->link;
                  q=q->link;
               }
          if (p=-NULL&&q==NUIL)
            return l;
    else
           return 0;
  }
}
void GlistInsertTail (GListNode *h, GListNode *s)
{
GListNode p,r;
p=h->val. sublist;
r=h;
while (p!=NULL)
    {
        r=p;
        p=p->link;
     }
 r->link=s;
 s->link=NULL;
}
```

9.

转置矩阵就是将矩阵元素的行号与列号互换，根据已知的三对角矩阵的特点，其转置矩阵对角线元素不变，非零的非对角线元素 a_{ij} 与 a_{ji} 互换位置。又知元素的下标和存放一维数组空间位置的关系为 k=2i+j。可以设计出这个矩阵的转置算法如下。

```
#define N 10                                    //矩阵行、列数
#define Length (3*N-2)                          //压缩矩阵的长度
typedef struct{
     int data[Length];
  }DiaMatrix;

void TransMatrix(DiaMatrix * C)
  { //压缩三对角矩阵转置
     int i, j;
     int t;
     for(i=0; i<N;i++)
       for(j=i; j<N; j++)
```

```
  if(i-j<=1&&i-j>=-1)
  { //将对应于行列号的压缩矩阵内的元素互换
            t=C-> data[2*i+j];
            C-> data[2*i+j]=C-> data[2*j+i];
            C-> data[2*j+i]=t;
          }                                          //endif
  }                                                  //end
```

10.

矩阵相加就是将两个矩阵中同一位置的元素值相加。由于两个稀疏矩阵的非零元素按三元组表形式存放，在建立新的三元组表 C 时，为了使三元组元素仍按行优先排列，所以每次插入的三元组不一定是 A 的，按照矩阵元素的行列去找 A 中的三元组，若有，则加入 C，同时，这个元素如果在 B 中也有，则加上 B 的这个元素值，否则这个值就不变；如果 A 中没有则找 B，若有则加入 C，若无则查找下一个矩阵元素。

```
#define MaxSize 10                                   //用户自定义
typedef int DataType;                                //用户自定义
typedef struct
  { //定义三元组
    int i,j;
    DataType v;
  }TriTupleNode;

typedef struct
  { //定义三元组表
    TriTupleNode data[MaxSize];
    int m,n,t;                                       //矩阵行、列及三元组表长度
  }TriTupleTable;

//以下为矩阵加算法
void AddTriTuple( TriTupleTable *A, TriTupleTable *B, TriTupleTable *C)
  { //将三元组表表示的稀疏矩阵 A,B 相加
    int k,l;
    DataType temp;
    C-> m=A-> m;                                     //矩阵行数
    C-> n=A-> n;                                     //矩阵列数
    C-> t=0;                                         //三元组表长度
    k=0; l=0;
    while (k<A-> t&&l<B-> t)
      {if((A-> data[k].i==B-> data[l].i)&&(A-> data[k].j==B-> data[l].j))
          {temp=A-> data[k].v+B-> data[l].v;
            if (!temp)                               //相加不为零,加入 C
              {C-> data[c-> t].i=A-> data[k].i;
                C-> data[c-> t].j=A-> data[k].j;
                C-> data[c-> t++].v=temp;
              }
            k++;l++;
          }
       if ((A-> data[k].i==B-> data[l].i)&&(A-> data[k].j<B-> data[l].j))
            ||(A-> data[k].i<B-> data[l].i)          //将 A 中三元组加入 C
         {C-> data[c-> t].i=A-> data[k].i;
          C-> data[c-> t].j=A-> data[k].j;
          C-> data[c-> t++].v=A-> data[k].v;
```

```
            k++;
        }
    if ((A->data[k].i==B->data[l].i)&&(A->data[k].j>B->data[l].j))
            ||(A->data[k].i>B->data[l].i)        //将B中三元组加入C
        {C->data[c->t].i=B->data[l].i;
         C->data[c->t].j=B->data[l].j;
         C->data[c->t++].v=B->data[l].v;
         l++;
        }
    }
  while (k<A->t)                                 //将A中剩余三元组加入C
    {C->data[c->t].i=A->data[k].i;
     C->data[c->t].j=A->data[k].j;
     C->data[c->t++].v=A->data[k].v;
     k++;
    }
  while (l<B->t)                                 //将B中剩余三元组加入C
    {C->data[c->t].i=B->data[l].i;
     C->data[c->t].j=B->data[l].j;
     C->data[c->t++].v=B->data[l].v;
     l++;
    }
}
```