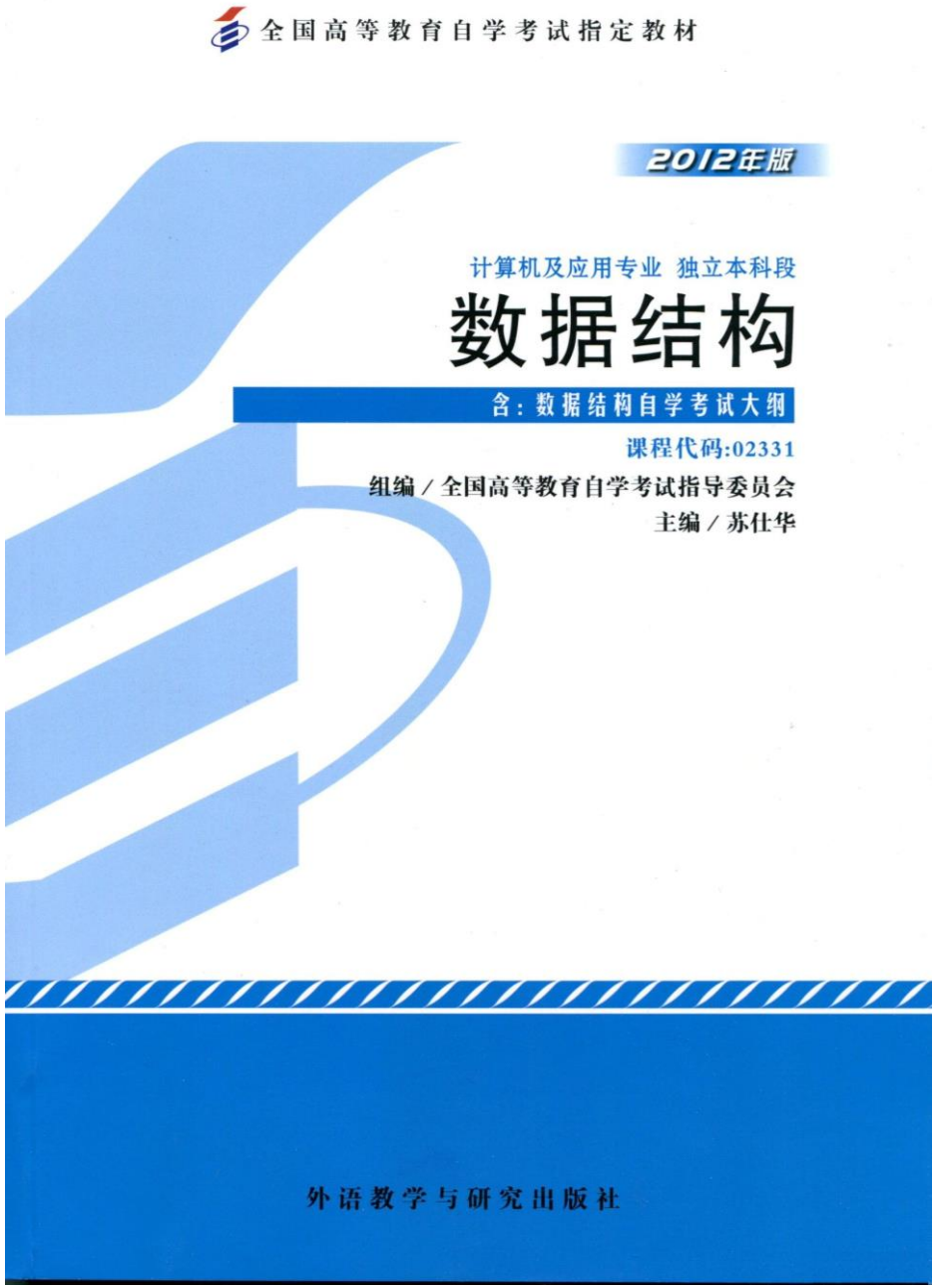
尚德机构

数据结构

主讲：王老师

学习是一种信仰！ IN LEARING WE TRUST

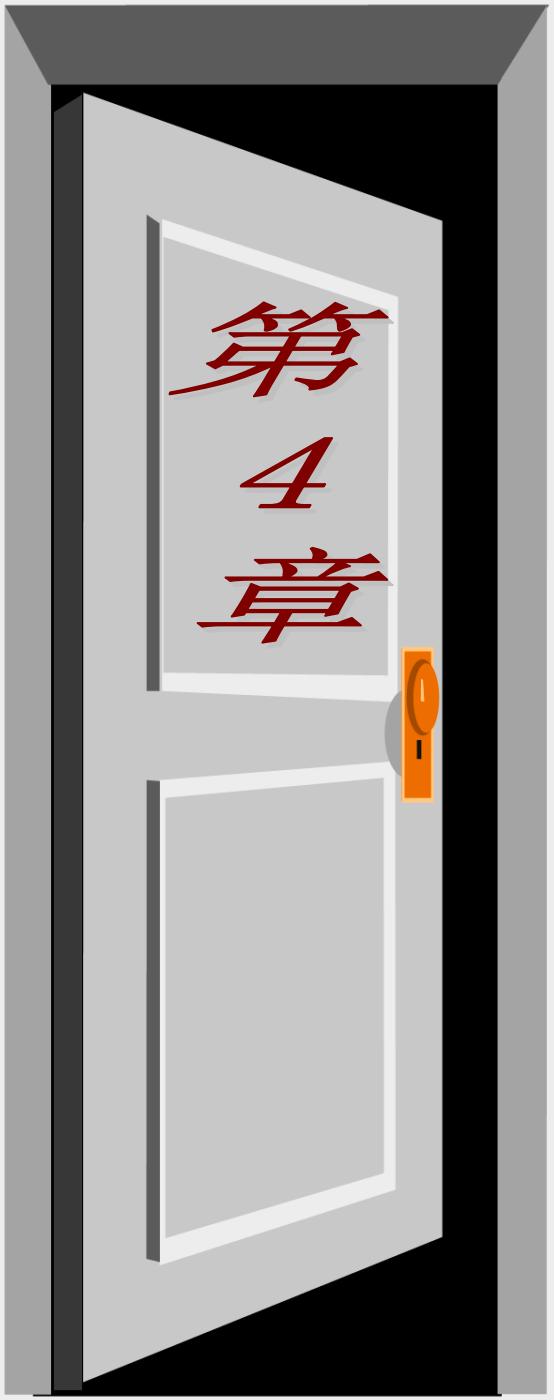


全国高等教育自学考试指定教材 2331数据结构

主　编　　苏仕华

出版社　　外语教学与研究出版社

出版时间　2012年3月



多维数组和广义表

$$A_{m\times n}=\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix}$$

二维数组又可以以行向量形式表示为：

$A_{m\times n}=[[a_{00}, a_{01}, \ldots, a_{0\ n-1}], [a_{10}, a_{11}, \ldots, a_{1\ n-1}], \ldots, [a_{m-1\ 0}, a_{m-1\ 1}, \ldots, a_{m-1\ n-1}]]$

或者以列向量形式表示为:

$A_{m\times n}=[[a_{00}, a_{10}, \ldots, a_{m-1\ 0}], [a_{01}, a_{11}, \ldots, a_{m-1\ 1}], \ldots, [a_{0\ n-1}, a_{1\ n-1}, \ldots, a_{m-1\ n-1}]]$

多维数组是一种复杂的数据结构。以二维数组为例，数组元素之间的关系除了边界元素外，每个元素a_{ij}都恰好有两个直接前趋和两个直接后继：行向量上的直接前趋是a_{ij-1}和a_{ij+1}，列向量上的直接前趋是a_{i-1j}和直接后继叫a_{i+1j}。并且二维数组也只有一个开始结点a_{00}，它没有前趋；仅有一个终端结点$a_{m-1\ n-1}$它没有后继。此外，边界上的结点（开始结点和终端结点除外）只有一个直接前趋或者只有一个直接后继。

4.1.1一、数组的顺序存储

数组在各种高级语言中通常也有两种不同的顺序存储方式，其中最典型的Pascal和C语言是按行优先顺序存储的，而Fortran语言则是按列优先顺序存储的。

（1）按行优先顺序存储，即将数组元素按行向量排列，第i+1个行向量紧接在第i个行向量后面。A的m×n个元素按行优先顺序存储的线性序列为：

$a_{00}, a_{01}, \ldots, a_{o\ n-1}, a_{10}, a_{11}, \ldots, a_{1\ n-1}, a_{m-1\ 0}, a_{m-1\ 1}, \ldots, a_{m-1\ n-1}$

（2）按列优先顺序存储，即将数组元素按列向量排列，第j+1个列向量紧接在第j个列向量之后，A的m×n个元素按列优先顺序存储的线性序列为：

$a_{00}, a_{10}, \ldots, a_{m-1\ 0}, a_{01}, a_{11}, \ldots, a_{m-1\ 1}, a_{0\ n-1}, a_{1\ n-1}, \ldots, a_{m-1\ n-1}$

4.1.1 数组的顺序存储

二维数组$A_{m\times n}$按行优先顺序存储在内存中，假设每个元素占d个存储单元，数组元素a_{ij}(i=0, 1, ..., m-1; j=0, 1, ..., n-1)位于第i行、第j列，前面i行共有i×n个元素，第i行上a_{ij}前面又有j个元素，因此它的前面一共有i×n+j个元素，所以在C语言中的数组元素a_{ij}的地址计算函数为：

$$LOC(a_{ij})=LOC(a_{00})+(i\times n+j)\times d$$

例如，有数组$A_{4\times 5}$, d=2，$LOC(a_{00})=100$，计算a_{23}的存储地址。

因为i=2，j=3，n=5，根据地址计算函数得：

$$LOC(a_{23})=100+(2\times 5+3)\times 2=126$$

同理，三维数组$A_{m\times n\times p}$按行优先顺序在内存中，计算数组元素a_{ijk}的地址计算函数为：

$$LOC(a_{ijk})=LOC(a_{000})+(i\times n\times p+j\times p+k)\times d$$

真题演练

设二维数组M有3行4列，按行优先的方式存储，每个元素占6个存储单元。第1个元素的存储地址为100，则M[2][2]的存储地址为（　）。

A: 135

B: 153

C: 160

D: 165

真题演练

设二维数组M有3行4列，按行优先的方式存储，每个元素占6个存储单元。第1个元素的存储地址为100，则M[2][2]的存储地址为（　）。

A: 135

B: 153

C: 160

D: 165

答案：C

真题演练

数组A[2][3]按行优先顺序存放，A的首地址为10。若A中每个元素占用一个存储单元，则元素A[1][2]的存储地址是（　）。

A: 10

B: 12

C: 14

D: 15

真题演练

数组A[2][3]按行优先顺序存放，A的首地址为10。若A中每个元素占用一个存储单元，则元素A[1][2]的存储地址是（　）。

A: 10

B: 12

C: 14

D: 15

答案：D

真题演练

二维数组A[10][6]采用行优先的存储方法,若每个元素占4个存储单元,已知元素A[3][4]的存储地址为1000,则元素A[4][3]的存储地址为(　　)。

A: 1020

B: 1024

C: 1036

D: 1240

真题演练

二维数组A[10][6]采用行优先的存储方法,若每个元素占4个存储单元,已知元素A[3][4]的存储地址为1000,则元素A[4][3]的存储地址为(　　)。

A: 1020

B: 1024

C: 1036

D: 1240

答案：A

真题演练

二维数组A[4][5]按行优先顺序存储,若每个元素占2个存储单元,且第一个元素A[0][0]的存储地址为1000,则数组元素A[3][2]的存储地址为(　　)。

A: 1012

B: 1017

C: 1034

D: 1036

真题演练

二维数组A[4][5]按行优先顺序存储,若每个元素占2个存储单元,且第一个元素A[0][0]的存储地址为1000,则数组元素A[3][2]的存储地址为(　　)。

A: 1012

B: 1017

C: 1034

D: 1036

答案：C

真题演练

二维数组M，行下标取值范围为0～8，列下标取值范围为1～10，若按行优先存储时，元素M[8][5]的存储地址为ar，则按列优先存储时，地址ar存储的数组元素应是（　）。

A: M[8][5]

B: M[5][8]

C: M[3][10]

D: M[0][9]

真题演练

二维数组M，行下标取值范围为0～8，列下标取值范围为1～10，若按行优先存储时，元素M[8][5]的存储地址为ar，则按列优先存储时，地址ar存储的数组元素应是（　）。

A: M[8][5]

B: M[5][8]

C: M[3][10]

D: M[0][9]

答案：C

真题演练

A是7×4的二维数组，按行优先方式顺序存储，元素A[0][0]的存储地址为1000，若每个元素占2个字节，则元素A[3][3]的存储地址为（　　）。

A: 1015

B: 1016

C: 1028

D: 1030

真题演练

A是7×4的二维数组，按行优先方式顺序存储，元素A[0][0]的存储地址为1000，若每个元素占2个字节，则元素A[3][3]的存储地址为（　　）。

A: 1015

B: 1016

C: 1028

D: 1030

答案：D

真题演练

已知10x12的二维数组A，按“行优先顺序”存储，每个元素占1个存储单元，已知A[1][1]的存储地址为420，则A[5][5]的存储地址为（　）。

A: 470

B: 471

C: 472

D: 473

真题演练

已知10x12的二维数组A，按“行优先顺序”存储，每个元素占1个存储单元，已知A[1][1]的存储地址为420，则A[5][5]的存储地址为（　）。

A: 470

B: 471

C: 472

D: 473

答案：C

真题演练

在二维数组a[8][10]中，每个数组元素a[i][j]占用3个存储空间，所有数组元素存放在一个连续的存储空间中，则该数组需要的存储空间个数是（　）。

A: 80

B: 100

C: 240

D: 270

真题演练

在二维数组a[8][10]中，每个数组元素a[i][j]占用3个存储空间，所有数组元素存放在一个连续的存储空间中，则该数组需要的存储空间个数是（　）。

A: 80

B: 100

C: 240

D: 270

答案：C

真题演练

二维数组a[10][20]按行优先顺序存放在连续的存储空间中，元素a[0][0]的存储地址为200，若每个元素占1个存储空间，则元素a[6][2]的存储地址是（　）。

A: 226

B: 322

C: 341

D: 342

真题演练

二维数组a[10][20]按行优先顺序存放在连续的存储空间中，元素a[0][0]的存储地址为200，若每个元素占1个存储空间，则元素a[6][2]的存储地址是（　）。

A: 226

B: 322

C: 341

D: 342

答案：B

真题演练

在二维数组a[9][10]中：每个数组元素占用3个存储空间，从首地址SA开始按行优先连续存放，则元素a[8][5]的起始地址是（　）。

A: SA+141

B: SA+144

C: SA+222

D: SA+255

真题演练

在二维数组a[9][10]中：每个数组元素占用3个存储空间，从首地址SA开始按行优先连续存放，则元素a[8][5]的起始地址是（　）。

A: SA+141

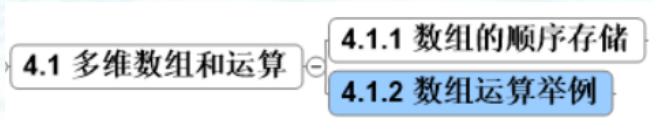
B: SA+144

C: SA+222

D: SA+255

答案：D

4.1.2二、数组运算举例



【例4.1】设计一个算法，实现矩阵A_{mn}的转置矩阵取B_{nm}。

分析：对于一个mxn的矩阵A，其转置矩阵是一个n×m的矩阵B，而且B[i][j]=A[j][i]，0≤i≤n-1，0≤j≤m-1。假设m=5，n=8，其实现算法如下：

```
void trsmat ( int a[][8], int b[][5], int m, int n)
{
   int i, j;
   for(j=0; j<m; j++)
      for(i=0; i<n; i++)
            b[i][j]=a[j][i];
}
```

4.2第二节 矩阵的压缩存储

在有些情况下，矩阵中含有许多值相同或者值为零的元素，如果还按前面的方法来存储这种矩阵，就会产生大量的空间浪费。**为了节省存储空间**，可以对这类矩阵采用压缩存储。

特殊矩阵，指的是相同值的元素或者零元素在矩阵中的分布有一定规律的矩阵。下面分别讨论几种特殊矩阵的压缩存储。

1. 对称矩阵

若n阶方阵A中的元素满足下述性质：

$a_{ij}=a_{ji}$　　$(0\le i, j\le n-1)$

则称A为n阶的对称矩阵。对称矩阵中的元素是关于主对角线对称的，所以只需要存储矩阵上三角或下三角的元素即可，让两个对称的元素共享一个存储空间。这样，就能够节省近一半的存储空间。按C语言的“按行优先”存储主对角线（包括主对角线）以下的元素，如图4.2所示。

$$\begin{bmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ a_{20} & a_{21} & a_{22} & \\ \cdots & \cdots & \cdots & \\ a_{n-10} & a_{n-11} & \cdots & a_{n-1n-1} \end{bmatrix}$$

$$\begin{bmatrix} a_{00} & & & \\ a_{10} & a_{11} & & \\ a_{20} & a_{21} & a_{22} & \\ \cdots & \cdots & \cdots & \\ a_{n-10} & a_{n-11} & \cdots & a_{n-1n-1} \end{bmatrix}$$

在以上的下三角矩阵中，第i行（0≤i≤n-1）恰好有i+1个元素，所以**元素总数**为：

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

4.2.1 特殊矩阵

现假设以一维数组sa[n(n+1)/2]作为n阶对称矩阵A的存储结构，那么，矩阵中元素a_{ij}和数组元素sa[k]之间存在着一一对应关系。这种对应关系分析如下：

若i≥j时，则a_{ij}在下三角矩阵中。a_{ij}之前i行（从0行到第i-1行）一共有1+2+3...+i=i×(i+1)/2个元素，在第i行上，a_{ij}之前恰有j个元素。因此有：

$$k=i\times(i+1)/2+j \qquad (0\le k<n(n+1)/2)$$

若i<j时，则a_{ij}在上三角矩阵中。因为有$a_{ij}=a_{ji}$，所以只要交换i和j即可得到：

$$k=j\times(j+1)/2+i \qquad (0\le k<n(n+1)/2)$$

因此，有：

$$k=\begin{cases}\dfrac{i\times(i+1)}{2}+j & i\geqslant j,\\ \dfrac{j\times(j+1)}{2}+i & i<j,\end{cases} \qquad 0\leqslant k\leqslant n(n+1)/2-1。$$

a_{ij}的存储地址可用下面的公式计算：

$LOC(a_{ij})=LOC(sa[k])=LOC(sa[0])+k\times d$

有了上述的计算公式，就能够立即找到矩阵元素a_{ij}在其压缩存储表示sa中的对应位置k。例如，a_{32}和a_{23}都存储在sa[8]中，这是因为

$k=i\times（i+1）/2+j=3x(3+1)/2+2=8$

【例4.3】已知d和B是两个阶的对称矩阵，因为是对称矩阵，所以仅需要输入下三角元素值存入一维数组。试写一算法，求对称矩阵A和B的乘积。

分析：如果是两个完整的矩阵相乘，其算法是比较简单的，但由于是对称矩阵，所以要清楚对称矩阵的第i行和第j列的元素数据在一维数组中的位置，其位置计算公式为:

1=i×（i+1）/2+j　　当i≥j时（A_{ij}，B_{ij}处于下三角中）

1=j×（j+1）/2+i　　当i<j时（A_{ij}，B_{ij}处于上三角中）

其中，1代表A_{ij}在其对称矩阵中的位置，而且0≤1≤n（n+1）/2。因此，实现本题功能的算法如下页:


```
void matrixmult ( int a[], int b[], int c[] [20], int n)
{   //n为A、B矩阵下三角元素个数，a,b分别为一维数组，
    //存放矩阵A和B的下三角元素值，c存放A和B的乘积
    for ( i=0; i<20; i++)
        for ( j=0; j<20; j++) {
            s=0;
            for ( k=0; k<n; k++) {
                if ( i>=k )                    //表示元素为下三角的元素，计算在a数组中的下标
                    11=i*(i+1)/2+k;
                else                           //表示元素为上三角的元素，计算下标
                    11=k*(k+1)/2+i;
                if ( k>=j )                    //表示元素为下三角的元素，计算在b数组中的下标
                    12=k*(k+1)/2+j;
                else
                    12=j*(j+1)/2+k;
                    s=s+a[11]*b[12];
                }
            c[i, j]=s;
            }
    }
```

2. 三角矩阵

以主对角线划分，三角矩阵有上三角和下三角两种。下三角矩阵正好相反，它的主对角线上方均为常数c或零，如图4.3(a)所示；上三角矩阵是指矩阵的下三角（不包括对角线）中的元素均为常数c或是零的n阶方阵，如图4.3(b)所示。一般情况下，三角矩阵的常数c均为零。

$$\begin{bmatrix} a_{00} & c & c & \cdots & c \\ a_{10} & a_{11} & c & \cdots & c \\ a_{20} & a_{21} & a_{22} & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & c \\ a_{n-1,0} & a_{n-1,1} & a_{n-1,2} & \cdots & a_{n-1,n-1} \end{bmatrix}$$

（a）下三角矩阵

$$\begin{bmatrix} a_{00} & a_{01} & a_{0,2} & \cdots & a_{0,n-1} \\ c & a_{11} & a_{1,2} & \cdots & a_{1,n-1} \\ c & c & a_{22} & \cdots & a_{2,n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ c & c & \cdots & c & a_{n-1,n-1} \end{bmatrix}$$

（b）上三角矩阵

4.2.1 特殊矩阵

三角矩阵中的重复元素C可共享一个存储空间，其余的元素正好有n(n+1)/2个，因此，三角矩阵可压缩存储在一维数组sa[n(n+1)/2+1]中，其中c存放在数组的最后一个元素中。

在上三角矩阵中，主对角线上第m(0≤m<n)行上恰好有n-m个元素，按行优先顺序存储上三角矩阵中元素a_{ij}时，a_{ij}之前有i行（0∽i-1），一共有元素个数：

$$\sum_{m=0}^{i-1}(n-m)=i\times(2\times n-i+1)/2$$

而在第i行，a_{ij}之前有j-i个元素，因此sa[k]和a_{ij}存储位置的对应关系为:

$$k=\begin{cases} i\times(2n-i+1)/2+j-i & \text{当} i\leqslant j \\ n\times(n+1)/2 & \text{当} i>j \end{cases}$$

而在下三角矩阵中，sa[k]和a_{ij}存储位置对应关系与前面介绍的对称矩阵压缩存储类似，只是在当i<j时，下三角矩阵中仅有一个存储位置。因此，该对应关系为：

$$k=\begin{cases} i\times(i+1)/2+j & \text{当} i\geqslant j \\ n\times(n+1)/2 & \text{当} i<j \end{cases}$$

真题演练

A是一个10×10的对称矩阵，若采用行优先的下三角压缩存储，第一个元素a0,0的存储地址为1，每个元素占一个存储单元，则a7,5的地址为（ ）。

A: 25

B: 26

C: 33

D: 34

真题演练

A是一个10×10的对称矩阵，若采用行优先的下三角压缩存储，第一个元素a0,0的存储地址为1，每个元素占一个存储单元，则a7,5的地址为（ ）。

A: 25

B: 26

C: 33

D: 34

答案：D

真题演练

设有一个10阶的对称矩阵A,采用行优先压缩存储方式,a11为第一个元素，其存储地址为1,每个元素占一个字节空间,则a85的地址为(　　)。

A: 13

B: 18

C: 33

D: 40

真题演练

设有一个10阶的对称矩阵A,采用行优先压缩存储方式,a11为第一个元素，其存储地址为1,每个元素占一个字节空间,则a85的地址为(　)。

A: 13

B: 18

C: 33

D: 40

答案：C

真题演练

设有一个10阶的下三角矩阵A，采用行优先压缩存储方式，a11为第一个元素，其存储地址为1000，每个元素占一个地址单元，则a85的地址为 （ ）。

A: 1012

B: 1017

C: 1032

D: 1039

真题演练

设有一个10阶的下三角矩阵A，采用行优先压缩存储方式，a11为第一个元素，其存储地址为1000，每个元素占一个地址单元，则a85的地址为 （ ）。

A: 1012

B: 1017

C: 1032

D: 1039

答案：C

4.2.2二、稀疏矩阵

由于特殊矩阵中非零元素的分布是有规律的，因此总可以找到矩阵元素与一维数组的下标的对应关系。但还有一种矩阵，其中有个非零元素，而远远小于矩阵元素的总数，通常把这种矩阵称为稀疏矩阵。为了节省存储单元，也可用压缩存储方法只存储非零元素。由于稀疏矩阵非零元素的分布一般是没有规律的，因此在存储非零元素时，除了存储非零元素的值之外，还必须同时存储该元素的行、列位置（即下标），所以可用一个称为三元组（i，j，a_{ij}）来唯一确定一个非零元素。

当用三元组来表示非零元素时，对稀疏矩阵进行压缩存储通常有两种方法：顺序存储和链式存储。链式存储一般采用的是十字链表法，结构比较复杂，在这里就不再介绍，只介绍**顺序存储方式**的压缩存储技术。

1. 三元组表

如果将表示稀疏矩阵非零元素的三元组按行优先的顺序排列，则可得到一个其结点均为三元组的线性表，将这种线性表的顺序存储结构称为三元组表。

为了操作方便，对稀疏矩阵的总行数、总列数以及非零元素个数均作为三元组表的辅助属性加以描述。三元组表的类型定义如下：

```
#define MaxSize 1000                          //假设非零元素个数的最大为1000个
typedef struct {
    int i, j;                                 //非零元素的行号、列号（下标）
    DataType v;                               //非零元素值
} TriTupleNode;
typedef struct {
    TriTupleNode data [ MaxSize ];            //存储三元组的数组
    int m, n, t;                              //矩阵的行数、列数和非零元素个数
} TSMatrix;                                   //稀疏矩阵类型
```

【例4.4】试写一个算法，建立顺序存储稀疏矩阵的三元组表。

```
void CreateTriTable ( TSMatrix * b, int a[][5], int m, int n )
{   //建立稀疏矩阵的三元组表
    int i, j, k=0 ;
    for ( i=0; i<m; i++)
        for ( j=0; j<n; j++)
            if ( a[i][j]!=0 ) {                  //找出非零元素
                b->data[k].i=i;                  //记录非零元素行下标
                b->data[k].j=j;                  //记录非零元素列下标
                b->data[k].v=a[i][j];            //保存非零值
                k++;                             //统计非零元素个数
            }
        b->m=m; b->n=n;                          //记录矩阵行列数
        b->t=k;                                  //保存非零个数
}
```

【例4.5】试写一个算法，实现以三元组表结构存储的稀疏矩阵的转置运算。

（1）一般的转置算法

假设a是TSMatrix型变量，表示稀疏矩阵M；b是指向稀疏矩阵T的TSMatrix型指针变量。从上面的分析可知，要将转置成就是将的三元组表a.data转置为T的三元组表b->data。要想得到如图4.5所示的按行优先顺序存储的b->data，就必须重新排列三元组表的顺序。由于M的行是T的列，所以按a.data的列转置，所得到的转置矩阵T的三元组表b->data一定是按行优先顺序存放的。实现这种方法的基本思想是：对M中的每一列col(0≤col≤a.n-1)从头至尾依次扫描三元组表，找出所有列号等于col的那些三元组，并将它们的行号和列号互换后再依次存入b->data中，这样就可得到T的按行优先的三元组表。

$$M_{4\times5}=\begin{bmatrix}0&3&0&5&0\\0&0&-2&0&0\\1&0&0&0&6\\0&0&8&0&0\end{bmatrix}\qquad T_{5\times4}=\begin{bmatrix}0&0&1&0\\3&0&0&0\\0&-2&0&8\\5&0&0&0\\0&0&6&0\end{bmatrix}$$

图 4.4　稀疏矩阵 M 和它的转置矩阵 T

	i	j	v
0	0	1	3
1	0	3	5
2	1	2	−2
3	2	0	1
4	2	4	6
5	3	2	8
⋮			

a.data

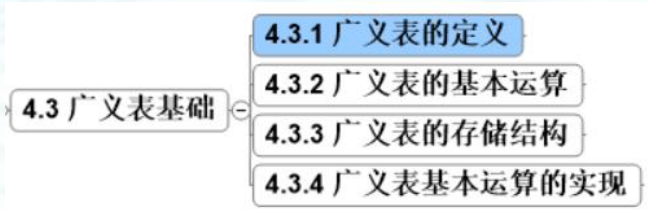
	i	j	v
0	0	2	1
1	1	0	3
2	2	1	−2
3	2	3	8
4	3	0	5
5	4	2	6
⋮			

b->data

图 4.5　稀疏矩阵的三元组表及其转置

广义表是n（n>0）个元素a_1, a_2, ..., a_n 的有限序列，其中a_i或者是原子项，或者是一个广义表，通常记作LS=(a_1, a_2, ..., a_n)。LS是广义表的名字，n为它的长度。若A又是广义表，则称它为LS的子表。为了区分原子和广义表，在书写时习惯上用大写字母表示广义表，用小写字母表示原子。通常用圆括号将广义表括起来，用逗号分隔其中的元素。当广义表LS非空时，称第一个元素a_1是LS的表头（head），其余元素组成的表（a_2, a_3, ..., a_n)称为LS的表尾（tail）。

4.3.1 广义表的定义



（1）A=( )——A是一个空表，其长度为零。

（2）B=(a)——B是一个只有一个原子的广义表，其长度为1。

（3）C=(a，（b，c))——C是一个长度为2的广义表，第一个元素是原子，第二个元素是子表。

（4）D= (A, B, C)=(( ), (a), (a , (b, c)))——D是一个长度为3的广义表，其中三个元素均为子表。

（5）E=(C, d)=((a, (b, c)), d)——E是一个长度为2的广义表，第一个元素是子表，第二个元素是原子。

（6）F=(e, F)=(e, (e, (e,…)))——F是一个递归的表，它的长度为2，第一个元素是原子，第二个元素是表自身，展开后它是一个无限的广义表。

一个表展开后所含括号的层数称为广义表的深度。例如，表A、B、C、D、E的深度分别为1、1、2、3、3，而表F的深度为无穷大∞。

4.3.1 广义表的定义

没有共享结点,没有递归结点的表是**纯表**。有共享结点的表是**再入表**,有递归结点的表是**递归表**。

广义表的几个重要性质:

（1）广义表的元素可以是子表，而子表又可以含有子表，因此广义表是一个多层次结构的表，它可以用图来形象地表示。如图4.6所示。

D= (A, B, C)=(( ), (a), (a , (b, c)))

E=(C, d)=((a, (b, c)), d)

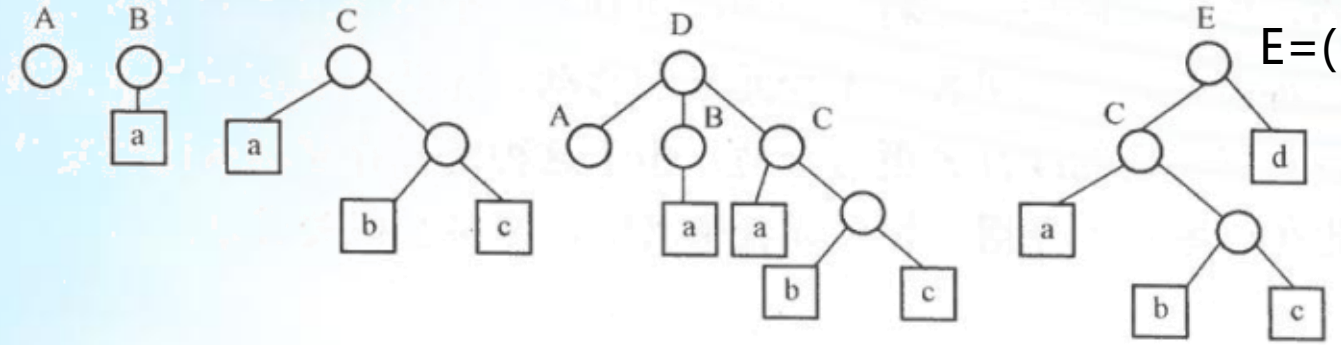


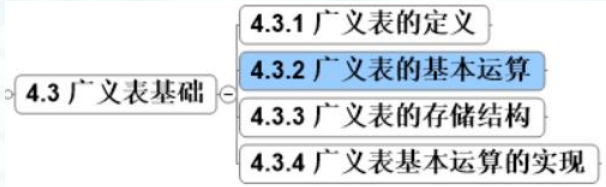
图 4.6　广义表的图形表示

（2）广义表具有递归和共享的性质，例如，表F就是一个递归的广义表，D表是共享的表，在表D中可以不必列出子表的值，而是通过子表的名字来引用。

D= (A, B, C)=(( ), (a), (a , (b, c)))

F=(e, F)=(e, (e, (e,…)))

4.3.2二、广义表的基本运算



【例4.6】取表头head(LS)、取表尾tail(LS)运算。由表头和表尾的定义可知，任何一个非空的广义表其表头可能是原子，也可能是子表，而其表尾一定是子表。以上一节中所给的广义表为例，给出下列运算的结果：

head(B)=a,　　tail(B)=( )

head(C)=a,　　tail(C)=((a, b))　　tail(tail(C))=( )

head(D)=A,　　tail(D)=(B,C),　　tail(tail(D))=(C)=((a, (b, c)))

head(E)=C,　　head(head(E))=a,　　tail(E)=(d)

B=(a)

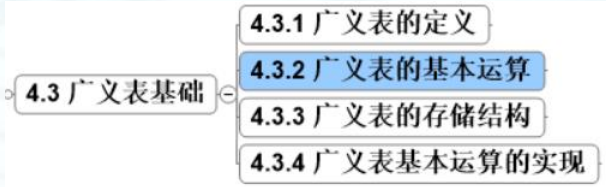
C=(a，　(b，c))

D= (A, B, C)=(( ), (a), (a , (b, c)))

E=(C, d)=((a, (b, c)), d)

F=(e, F)=(e, (e, (e,…)))

4.3.2二、广义表的基本运算



【例4.7】已知有下列的广义表，试求出下面广义表的表头head( )、表尾tail( )、表长length( )和深度depth( )。

（1）A=(a, (b, c, d), e, (f, g));　　（2）B=((a));

（3）C=(y, (z, w), (x, (z, w), a));　　（4）D=(x, ((y), B), D)。

解答：

（1）head(A)=a, tail(A)=((b, c, d), e, (f, g)),length(A)=4, depth(A)=2;

（2）head(B)=(a), tail(B)=( ), length(B)=1, depth(B)=2;

（3）head(C)=y, tail(C)=((z, w), (x, (z, w), a)),length(C)=3, depth(C)=3;

（4）head(D)=x, tail(D)=(((y), B), D),length(D)=3, depth(D)=∞

要特别注意的是，广义表( )和(( ))是不同的，前者为空表，长度为0；而后者是由空表作元素的广义表，其长度为1，它可以分解得到表头、表尾为空表() 。

真题演练

对广义表L=(a,())执行操作tail(L)的结果是(　　)。

A: ()

B: (())

C: a

D: (a)

真题演练

对广义表L=(a,())执行操作tail(L)的结果是(　　)。

A: ()

B: (())

C: a

D: (a)

答案：B

真题演练

已知广义表LS=(((a，b，c)，d)，e，(f，g，(h,i))，LS的深度是（　）。

A: 2

B: 3

C: 4

D: 5

真题演练

已知广义表LS=(((a，b，c)，d)，e，(f，g，(h,i))，LS的深度是（　）。

A: 2

B: 3

C: 4

D: 5

答案：B

真题演练

已知广义表G,head(G)与tail(G)的深度均为6,则G的深度是(　)。

A: 5

B: 6

C: 7

D: 8

真题演练

已知广义表G,head(G)与tail(G)的深度均为6,则G的深度是(　)。

A: 5

B: 6

C: 7

D: 8

答案：C

真题演练

广义表A=(a，B，(a，B，(a，B，...)))的长度为（ ）。

A: 1

B: 2

C: 3

D: 无限值

真题演练

广义表A=(a，B，(a，B，(a，B，...)))的长度为（ ）。

A: 1

B: 2

C: 3

D: 无限值

答案：C

真题演练

对于广义表A，若head(A)等于tail(A)，则表A为（　）。

A: ( )

B: (( ))

C: (( ),( ))

D: (( ),( ),( ))

真题演练

对于广义表A，若head(A)等于tail(A)，则表A为（　）。

A: ( )

B: (( ))

C: (( ),( ))

D: (( ),( ),( ))

答案：B

真题演练

广义表A=(a，(b，e，(e，f，g，h)))的表长是（ ）。

A: 2

B: 3

C: 4

D: 7

真题演练

广义表A=(a，(b，e，(e，f，g，h)))的表长是（ ）。

A: 2

B: 3

C: 4

D: 7

答案：A

真题演练

广义表A=(a(b，c，(e，f，g，h)))的深度是（　）。

A: 2

B: 3

C: 4

D: 7

真题演练

广义表A=(a(b，c，(e，f，g，h)))的深度是（ ）。

A: 2

B: 3

C: 4

D: 7

答案：B

真题演练

广义表A=(x，((y)，((a))，A))的深度是（　）。

A: 2

B: 3

C: 4

D: ∞

真题演练

广义表A=(x，((y)，((a))，A))的深度是（　）。

A: 2

B: 3

C: 4

D: ∞

答案：D

真题演练

广义表A=(a,(b,c,(e,f)))，函数head(head(tail(A)))的运算结果是（　）。

A: a

B: b

C: c

D: e

真题演练

广义表A=(a,(b,c,(e,f)))，函数head(head(tail(A)))的运算结果是（　）。

A: a

B: b

C: c

D: e

答案：B

真题演练

已知广义表LS=(((a))，((b，(c))，(d，(e，f)))，())，LS的长度是（　）。

A: 2

B: 3

C: 4

D: 5

真题演练

已知广义表LS=(((a))，((b，(c))，(d，(e，f)))，())，LS的长度是（　）。

A: 2

B: 3

C: 4

D: 5

答案：B

真题演练

广义表((a,b),(c,d))的表尾是（　）。

A: b

B: d

C: (c,d)

D: ((c,d))

真题演练

广义表((a,b),(c,d))的表尾是（　）。

A: b

B: d

C: (c,d)

D: ((c,d))

答案：D

真题演练

已知广义表LS =(((a,b)),((c,(d)),(e,(f))),(g,h))，LS的深度是（ ）。

A: 2

B: 3

C: 4

D: 5

真题演练

已知广义表LS =(((a,b)),((c,(d)),(e,(f))),(g,h))，LS的深度是（ ）。

A: 2

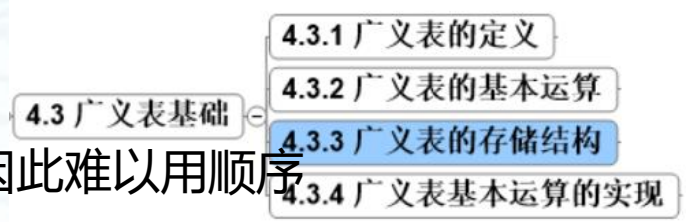
B: 3

C: 4

D: 5

答案：C

4.3.3三、广义表的存储结构



由于广义表中的元素本身又可以具有结构，是一种带有层次的非线性结构，因此难以用顺序存储结构表示，通常采用链式存储结构，每个元素可用一个结点表示，结点结构如下所示。

tag	data/slink	link

每个结点由三个域构成，其中tag是一个标志位，用来区分当前结点是原子还是子表，当tag为零值时，该结点是子表，第二个域为slink，用以存放子表的地址；当tag为1时，该结点是原子结点，第二个域为data，用以存放元素值。link域是用来存放与本元素同一层的下一个元素对应结点的地址，当该元素是所在层的最后一个元素时，link的值为NULL。其广义表及结点类型描述如下：

```
typedef enum { atom, list } NodeTag;    //atom=0，表示原子；list=1,表示子表
typedef struct GLNode {
    NodeTag tag;          //用以区分原子结点和表结点
    union {
        DataType data;    //用以存放原子结点值，其类型由用户自定义
        GLNode * slink;   //指向子表的指针
    };
    GLNode * next;        //指向下一个表结点
} * Glist;                //广义表类型
```

4.3.3 广义表的存储结构

(1) A=( )

(2) B=(a)

(3) C=(a, (b, c))

(4) D= (A, B, C)=(( ), (a), (a , (b, c)))

(5) E=(C, d)=((a, (b, c)), d)

(6) F=(e, F)=(e, (e, (e,…)))

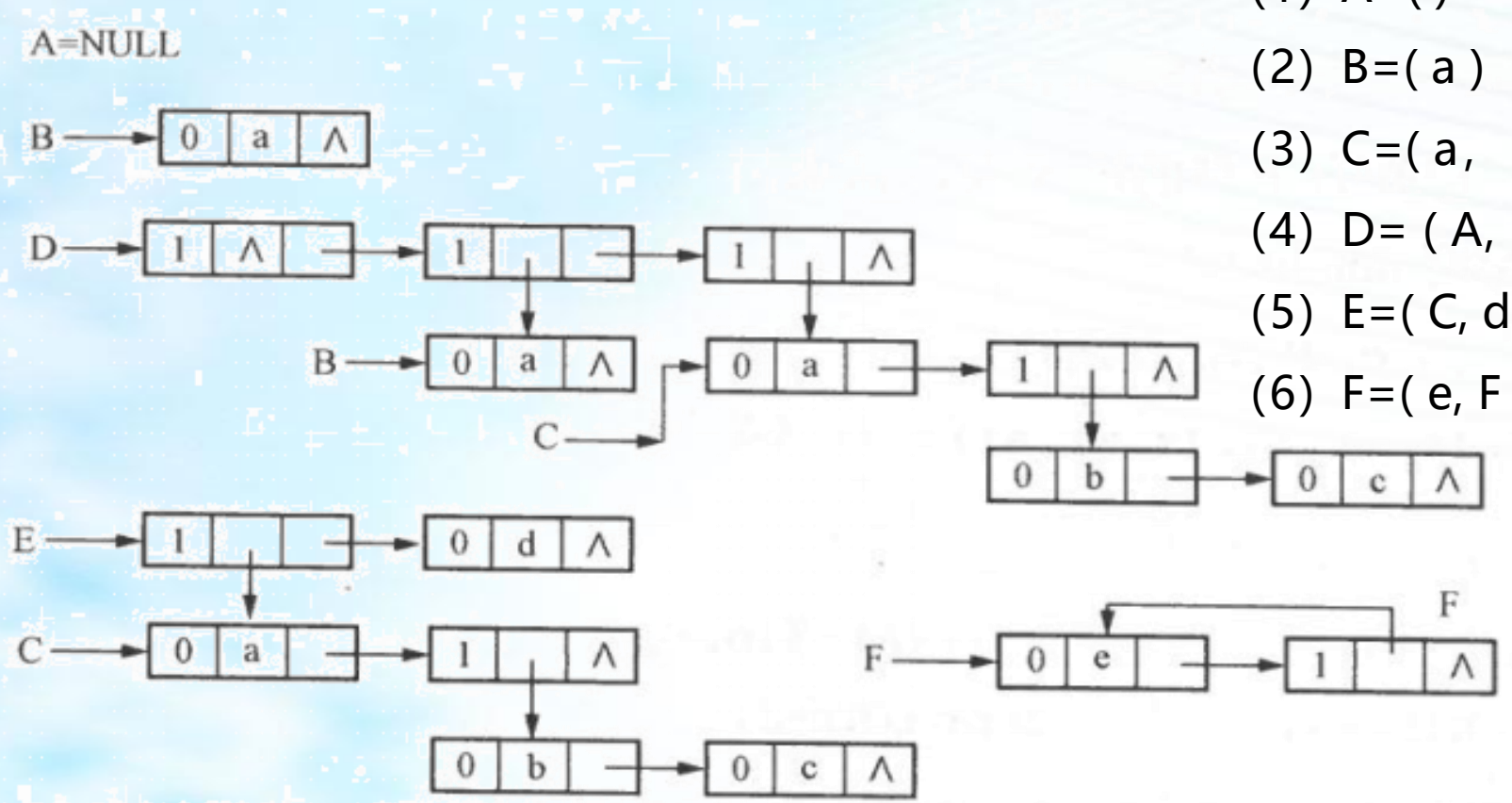
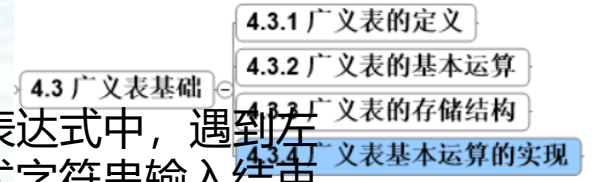


图 4.7 广义表的单链表示法

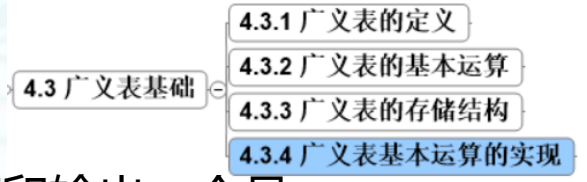
4.3.4四、广义表基本运算的实现



1. 建立广义表的存储结构CreatGList（Glist&GL）

通过用户输入的广义表表达式建立相应的广义表，并且边输入边建立。基本思想：在广义表表达式中，遇到左括号“（”时递归构造子表，否则构造原子结点；遇到逗号时递归构造后续广义表，直到表达式字符串输入结束（假设结束符为“；”）。因此，实现算法的C语言函数定义如下：

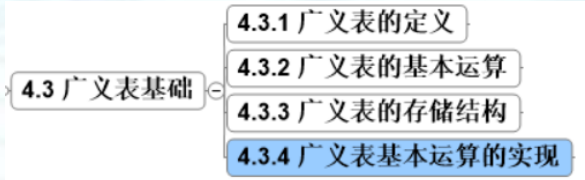
```
Glist CreatGList ( Glist GL )
{   char ch; scanf( “%cn”, &ch);
    if(ch!=' ') {
        GL=( GLNode * ) malloc ( sizeof ( GLNode ) );
        if(ch=='(') {
            GL->tag=list;
            GL->slink=CreatGList ( GL->slink);//递归调用构造子表
        }
        else {                                                    //构造原子结点
            GL->tag=atom;
            GL->data=ch;
        }
    }
    else GL=NULL;
    scanf("%c",&ch);
    if(GL!=NULL)
    if(ch==',')
        GL->link=CreatGList(GL->link);          //递归构造后续广义表
    else
        GL->link=NULL;                          //表示遇到’ ) ‘或结束符 ‘ ; ‘ 时，无后续表
    return GL;
}
```



2. 输出广义表PrintGList (Glist GL)

输出广义表所采用的算法思想：若遇到tag=1的结点，是一个子表的开始，则先打印输出一个号，如果该子表为空，则输出一个空格符，否则递归调用输出该子表，子表打印输出完后，再打印一个号；若遇到tag=0的结点，则直接输出其数据域的值。若还有后续元素，则递归调用打印后续每个元素，直到遇到link域为NULL。因此，算法可用C语言函数实现如下：

```
void PrintGList ( Glist GL )
{   if(GL!=NULL) {
      if(GL->tag==list) {
         printf(″ (″) ;
         if(GL->slink==NULL)printf(″ ″)
         else PrintGList (GL->slink);                    //递归调用输出子表
      }
      else printf(″%c″, GL->data);                       //输出结点数据域值
      if(GL->tag==list)
         printf (″) ″);
      if(GL->1ink!=NULL) {
         printf (″,″);
         PrintGList(GL->link);                           //递归调用输出下一个结点
      }
   }
}
```



3. 广义表的查找FindGlistX (Glist GL，DataType x，int & mark)

在给定的广义表中查找数据域为x的结点，采用的算法思想是：若遇到tag=0的原子结点，如果是要找的结点，则查找成功；否则，若还有后续元素，则递归调用本过程查找后续元素，直到遇到link域为NULL的元素。若遇到tag=1的结点，则递归调用本过程在该子表中查找，若还有后续元素，则递归调用本过程查找后续每个元素，直到遇到link域为NULL的元素。

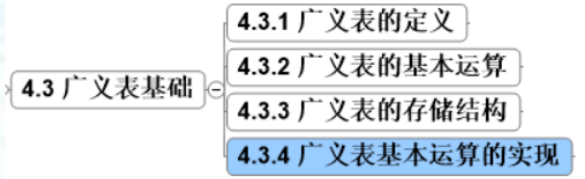
设f(p, x)为查找函数，当查找成功时返回true，否则返回false,则有如下递归模型：

f(p, x)=true　　若p->tag=0且p->data=x

f(p, x)=f(p->link, x)　　若p->tag=0且p->data^=x

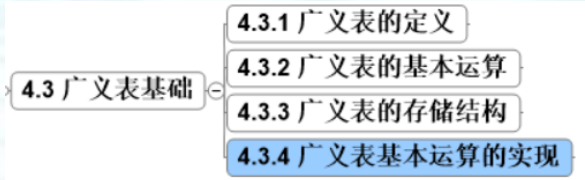
f(p, x)=f(p->slink, x) or f(p->link, x)　　若p->tag=1

4.3.4四、广义表基本运算的实现



可见，实现上述算法的过程是比较容易的，具体实现C函数如下:

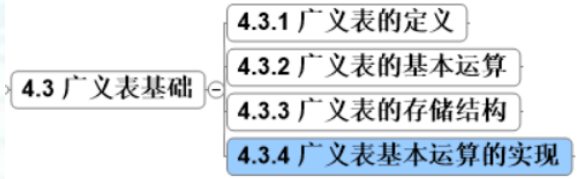
```
void FindGlistX ( Glist GL, DataType x, int *mark )
{  //调用广义表GL所指向的广义表，mark=false, x为待查找的元素值,
   //若查找成功，mark=true，p为指向数据域为x的结点
   if(GL!=NULL) {
      if(GL->tag==0 && GL->data==x ) {
         p=GL;                                          //p为全局量
         *mark=1;
      }
      else
         if(GL->tag==1) FindGlistX (GL->slink, x, mark);
         FindGlistX (GL->link, x, mark);
   }
}
```



4. 求广义表表头head(Glist GL)

一个广义表的表头指的是该广义表的第一个元素，因此求表头操作比较简单，实现算法描述如下：

```
Glist head (Glist GL)
{   Glist p;
    if(GL!=NULL && GL->tag!=0) {        //不为空表并且不只是原子
    p=GL->slink;
    p->link=NULL;
    return p;                           //返回GL中指向子表的指针;
    }
    else return NULL;
}
```



5. 求广义表表尾tail(GlistGL)

一个广义表的表尾指的是除去该广义表的第一个元素后的所有剩余部分，由此，实现上述算法描述如下：

```
Glist tail (Glist GL)
{   Glist p;
    if(GL!=NULL && GL->tag!=0) {                    //表不为空表并且有表尾
        p=GL->slink;
        p=p~>link;                                  //P指向第二个元素
        GL->slink=p;                                //删除广义表第一个元素；
    }
    return p;
}
```

4.3.4四、广义表基本运算的实现

7. 求广义表的深度depth(Glist GL，int & maxdh)

假设广义表是一个无共享子表的非递归表，求其表深度的算法思想是：扫描广义表的第一层的每个结点，对每个结点递归调用计算出其子表的深度，取最大的子表深度，然后加1即为广义表的最大深度。其递归模型如下：

maxdh(GL)=0　　GL为单个元素即GL->tag==0

maxdh(GL)=1　　GL为空表即GL->tag==1且GL->slink==NULL

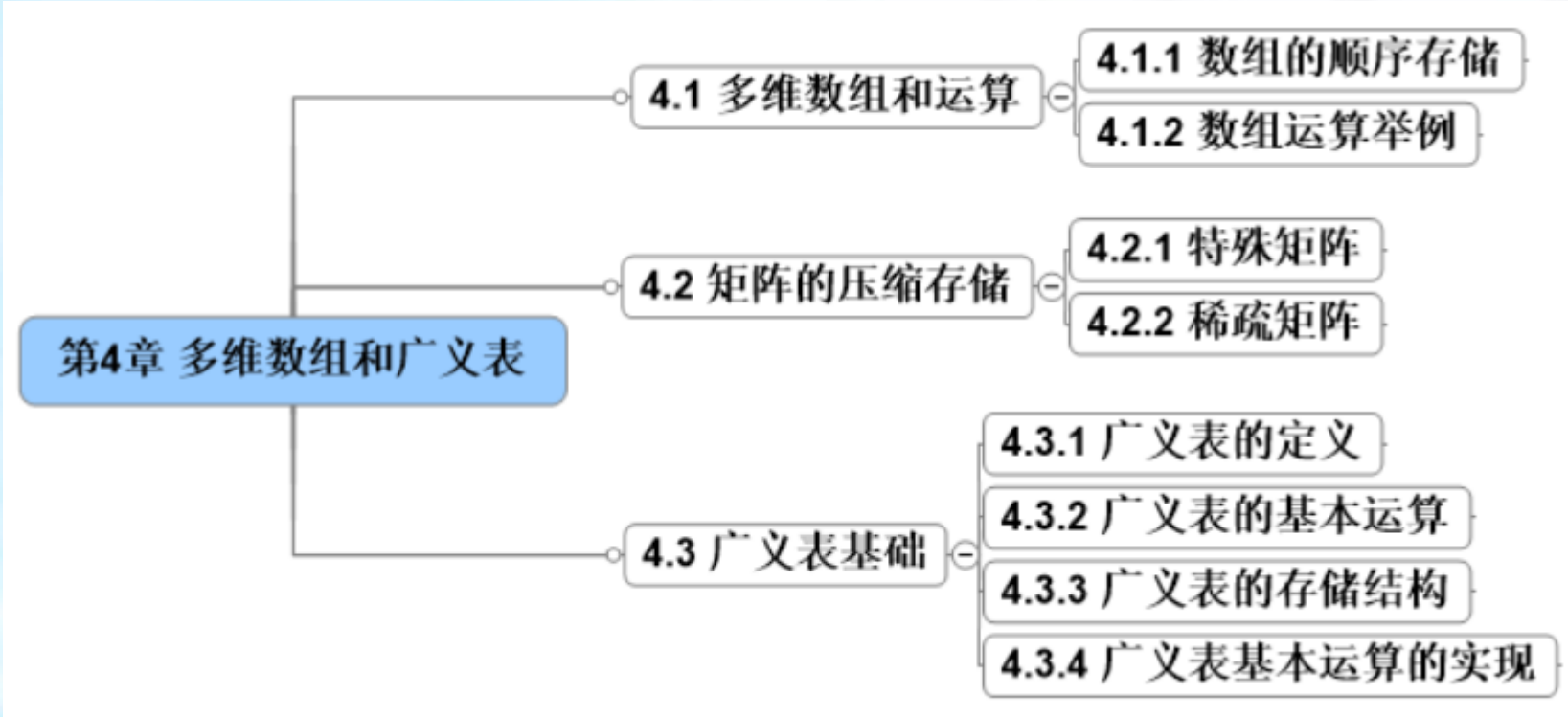
maxdh(GL)=max(maxdh(GL1), maxdh(GL2), …，maxdh(GLn))+1

其中，GL=(GL1, GL2, …，GLn)。

因此，实现该功能的算法描述如下：

```
void depth(Glist GL, int *maxdh)
{   int h;
    if(GL->tag==0) *maxdh=0;        //说明广义表为单个元素
    else
        if(GL->tag==1 && GL->slink==NULL)
            *maxdh=1;               //广义表为空表
        else{                       //进行递归求解
            GL=GL->slink;           //进入第一层
            *maxdh=0;
            do {                    //循环扫描表的第一层的每个结点，对每个结点求其子表深度,
                depth(GL, &h);
                if(h>*maxdh) *maxdh=h;  //取最大的子表深度
                    GL=GL->link;
            } while(GL!=NULL);
            *maxdh=*maxdh+1;        //子表最大深度加1
        }
}
```

本章总结


第4章 多维数组和广义表
4.1 多维数组和运算
4.1.1 数组的顺序存储
4.1.2 数组运算举例
4.2 矩阵的压缩存储
4.2.1 特殊矩阵
4.2.2 稀疏矩阵
4.3 广义表基础
4.3.1 广义表的定义
4.3.2 广义表的基本运算
4.3.3 广义表的存储结构
4.3.4 广义表基本运算的实现

祝大家顺利通过考试！