第五章 数组和广义表

数组

稀疏矩阵

广义表

数　组

一维数组

定义

相同类型的数据元素的集合。

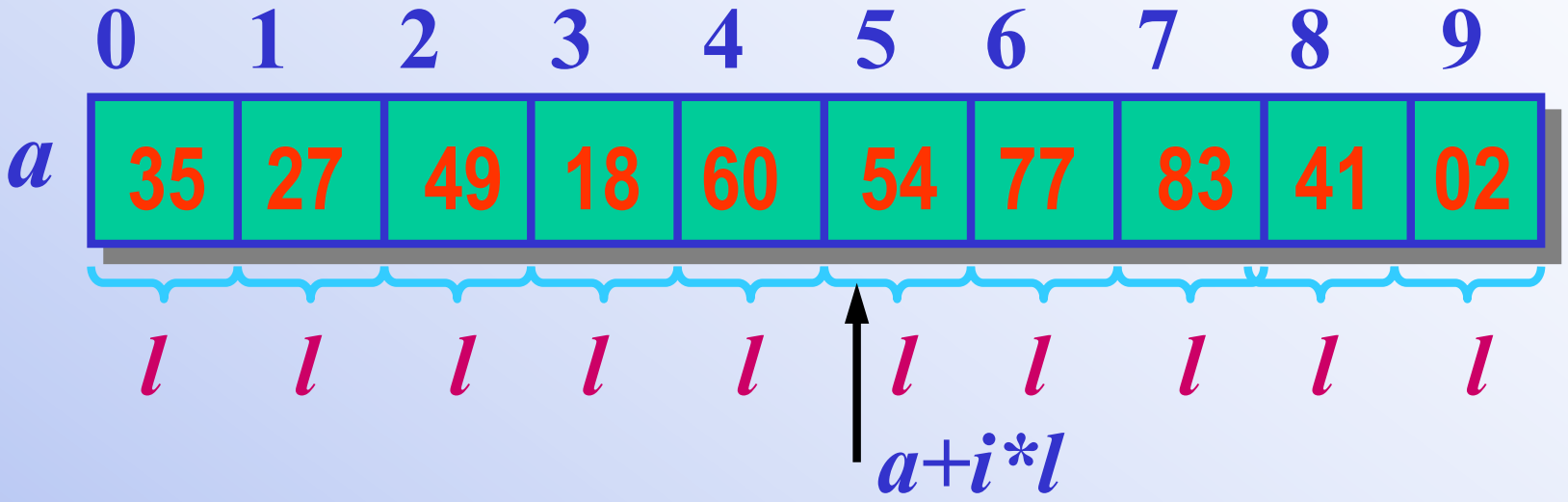
一维数组的示例

数组的定义和初始化

```
main ( ) {
    int a1[3] = { 3, 5, 7 }, *elem;
    for ( int i = 0; i < 3; i++ )
        printf ( “%d”, a1[i], “\n” );     //静态数组
    elem = a1;
    for ( int i = 0; i < 3; i++ ) {
        printf ( “%d”, *elem, “\n” );  //动态数组
        elem++;
    }
}
```

一维数组存储方式

$$LOC(i) = \begin{cases} a, & i = 0 \\ LOC(i-1)+l = a+i*l, & i > 0 \end{cases}$$



$$LOC(i) = LOC(i-1)+l = a+i*l$$

二维数组

类似于线性表，一个二维数组的逻辑结构可形式地表示为：2_Array=(D,R)

其中D={a_{ij}(i=0,1,…,m-1,j=0,1,…,n-1)}, a_{ij}是同类型数据元素的集合。

R={ROW,COL}是数据元素上关系的集合。

ROW={<a_{ij}, $a_{i(j+1)}$>|0<=i<=m-1,0<=j<=n-2}每一行上的列关系。

COL={<a_{ij}, $a_{(i+1)j}$>|0<=i<=m-2,0<=j<=n-1}每一列上的行关系。

$$a=\begin{pmatrix} a[0][0] & a[0][1] & \cdots & a[0][m-1] \\ a[1][0] & a[1][1] & \cdots & a[1][m-1] \\ a[2][0] & a[2][1] & \cdots & a[2][m-1] \\ \vdots & \vdots & \ddots & \vdots \\ a[n-1][0] & a[n-1][1] & \cdots & a[n-1][m-1] \end{pmatrix}$$

行优先存放：

设数组开始存放位置 LOC(0, 0) = a, 每个元素占用 *l* 个存储单元

$$LOC(i,j) = a + (i * m + j) * l$$

三维数组

各维元素个数为 m_1, m_2, m_3

下标为 i_1, i_2, i_3的数组元素的存储地址：

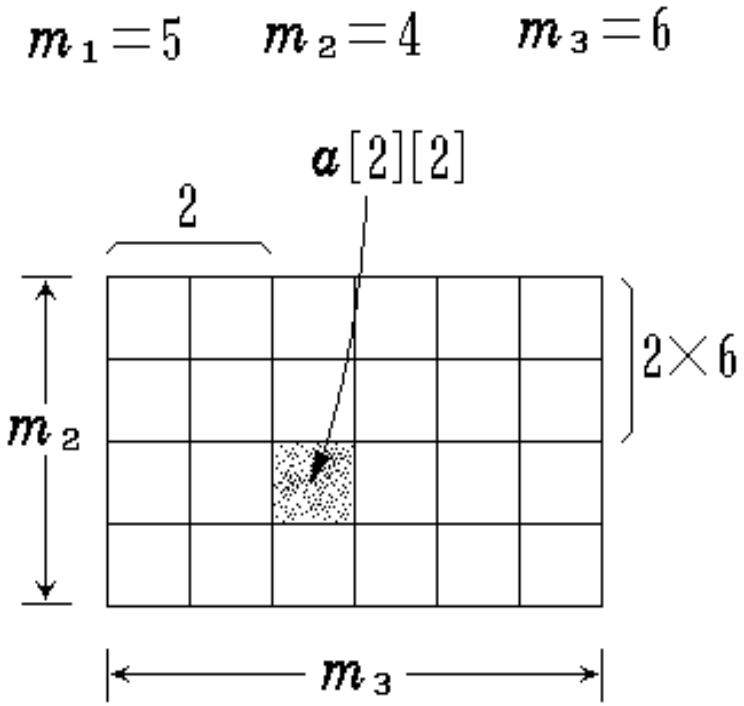
（按页/行/列存放）

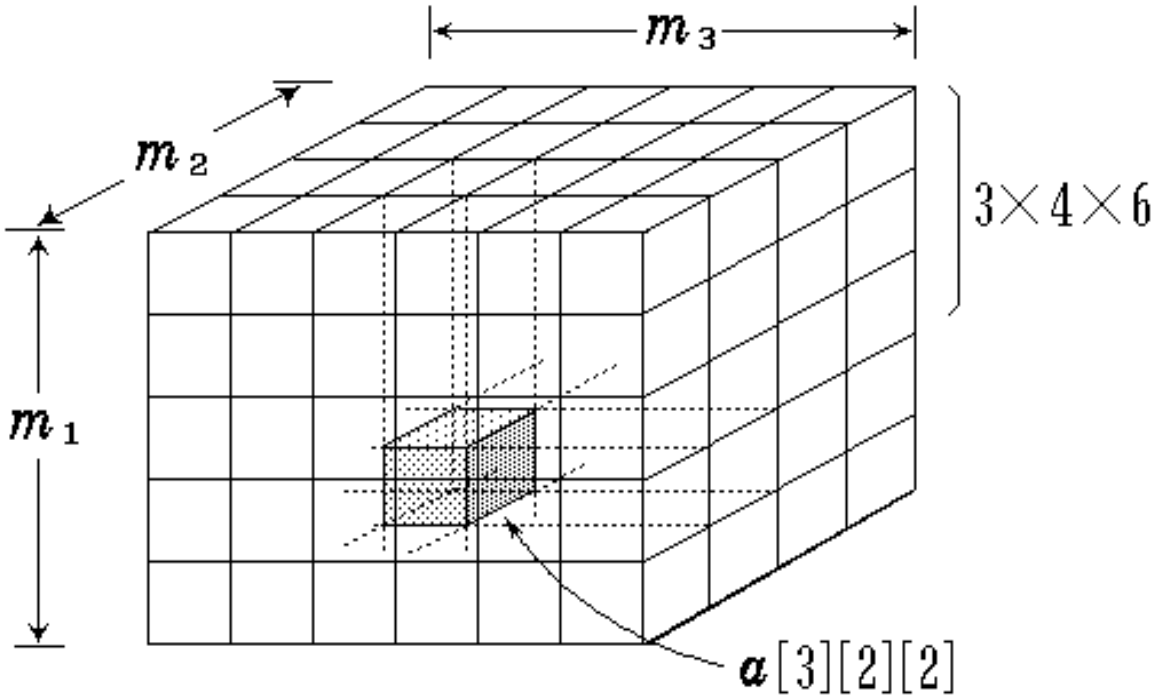
$$LOC(i_1, i_2, i_3) = a + (i_1 * m_2 * m_3 + i_2 * m_3 + i_3) * l$$

$i_1 * m_2 * m_3$：前i_1页总元素个数

$i_2 * m_3$：第i_1页的前i_2行总元素个数

二维数组　　三维数组





行向量　下标 i
列向量　下标 j

页向量　下标 i
行向量　下标 j
列向量　下标 k

n 维数组

各维元素个数为 $m_1, m_2, m_3, \ldots, m_n$

下标为 $i_1, i_2, i_3, \ldots, i_n$ 的数组元素的存储地址：

$$LOC(i_1, i_2, \ldots, i_n) = a + (i_1*m_2*m_3*\ldots*m_n + i_2*m_3*m_4*\ldots*m_n + \ldots\ldots + i_{n-1}*m_n + i_n) * l$$

$$= a + \left(\sum_{j=1}^{n-1} i_j * \prod_{k=j+1}^{n} m_k + i_n \right) * l$$

特殊矩阵的压缩存储

特殊矩阵是指非零元素或零元素的分布有一定规律的矩阵。

特殊矩阵的压缩存储主要是针对阶数很高的特殊矩阵。为节省存储空间，对可以不存储的元素，如零元素或对称元素，不再存储。

对称矩阵

三对角矩阵

对称矩阵的压缩存储

设有一个 n×n 的对称矩阵 A。

$$A=\begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1n-1} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-10} & a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} \end{bmatrix}$$

在矩阵中，$a_{ij} = a_{ji}$

为节约存储空间，只存对角线及对角线以上的元素，或者只存对角线及对角线以下的元素。前者称为上三角矩阵，后者称为下三角矩阵。

把它们按行存放于一个一维数组 B 中，称之为对称矩阵 A 的压缩存储方式。

数组 B 共有 $n + (n - 1) + \cdots + 1 = n*(n+1)/2$ 个元素。

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$

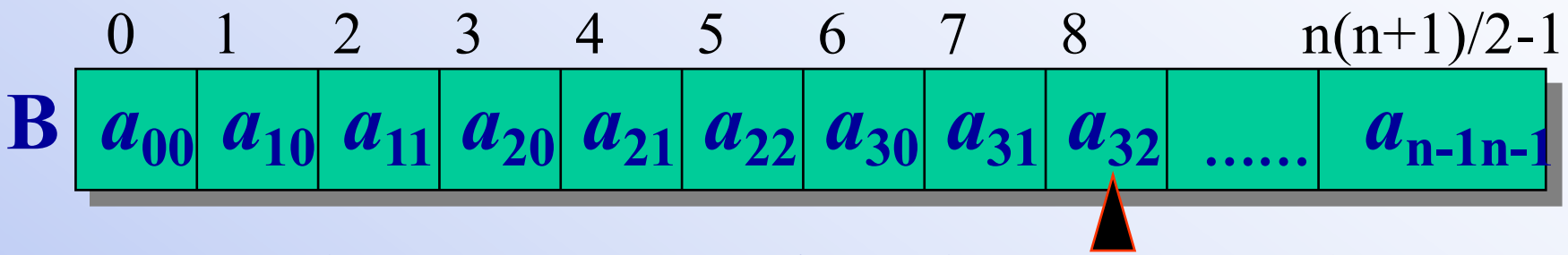
上三角矩阵

$$\begin{bmatrix} a_{03} & a_{02} & a_{01} & a_{00} \\ a_{13} & a_{12} & a_{11} & a_{10} \\ a_{23} & a_{22} & a_{21} & a_{20} \\ a_{33} & a_{32} & a_{31} & a_{30} \end{bmatrix}$$

下三角矩阵

$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1n-1} \\ a_{20} & a_{21} & a_{22} & \cdots & a_{2n-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-10} & a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} \end{bmatrix}$$

下三角矩阵



若 i ≥ j, 数组元素A[i][j]在数组B中的存放位置为 1 + 2 + … + i + j = (i + 1)* i / 2 + j

前i行元素总数　第i行第j个元素前元素个数

若 i < j，数组元素 A[i][j] 在矩阵的上三角部分，在数组 B 中没有存放，可以找它的对称元素A[j][i]：= j *(j +1) / 2 + i

若已知某矩阵元素位于数组B的第k个位置，可寻找满足

$$i\,(i+1)/2 \le k < (i+1)*(i+2)/2$$

的 i，此即为该元素的行号。

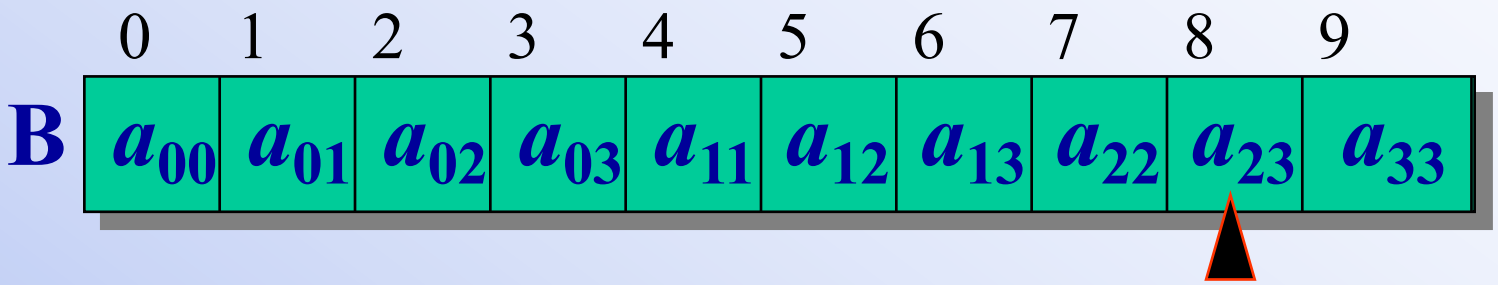
$$j = k - i*(i+1)/2$$

此即为该元素的列号。

例，当 k = 8，$3*4/2 = 6 \le k < 4*5/2 = 10$，取 i = 3。则 j = 8 - 3*4 / 2 = 2。

上三角矩阵

$$n = 4 \qquad \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix}$$



若$i \le j$，数组元素A[i][j]在数组B中的存放位置为 $n + (n-1) + (n-2) + \cdots + (n-i+1) + j-i$

前i行元素总数　第i行第j个元素前元素个数

若 $i \le j$，数组元素A[i][j]在数组B中的存放位置为

$$n + (n\text{-}1) + (n\text{-}2) + \cdots + (n\text{-}i\text{+}1) + j\text{-}i =$$

$$= (2*n\text{-}i\text{+}1) * i / 2 + j\text{-}i =$$

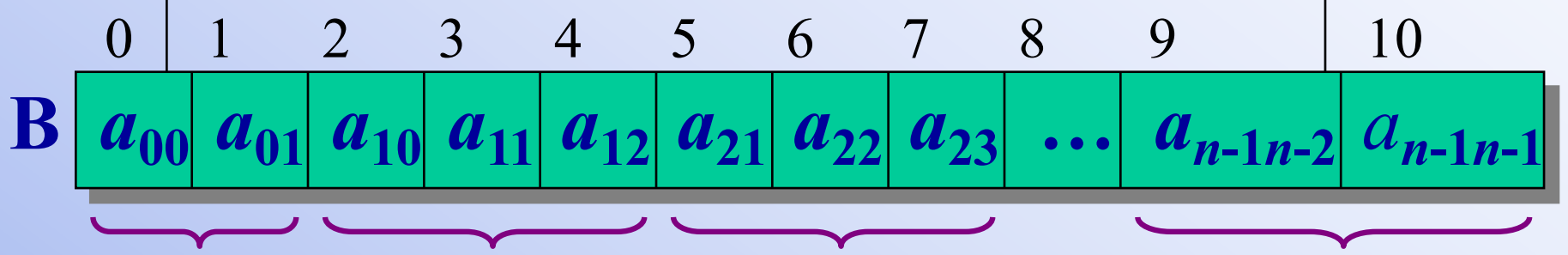
$$= (2*n\text{-}i\text{-}1) * i / 2 + j$$

若$i > j$，数组元素A[i][j]在矩阵的下三角部分，在数组 B 中没有存放。因此，找它的对称元素A[j][i]。

A[j][i]在数组 B 的第 $(2*n\text{-}j\text{-}1) * j / 2 + i$ 的位置中找到。

三对角矩阵的压缩存储

$$A=\begin{bmatrix} a_{00} & a_{01} & 0 & 0 & 0 & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 & 0 \\ 0 & a_{21} & a_{22} & a_{23} & 0 & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & a_{n-2n-3} & a_{n-2n-2} & a_{n-2n-1} \\ 0 & 0 & 0 & 0 & a_{n-1n-2} & a_{n-1n-1} \end{bmatrix}$$



三对角矩阵中除主对角线及在主对角线上下最临近的两条对角线上的元素外，所有其它元素均为0。总共有3n-2个非零元素。

将三对角矩阵A中三条对角线上的元素按行存放在一维数组 B 中，且a_{00}存放于B[0]。

在三条对角线上的元素a_{ij}满足

$$0 \le i \le n-1,\ i-1 \le j \le i+1$$

在一维数组 B 中 A[i][j] 在第 i 行，它前面有 3*i-1 个非零元素, 在本行中第 j 列前面有 j-i+1 个，所以元素 A[i][j] 在 B 中位置为 k = 2*i + j。

若已知三对角矩阵中某元素 A[i][j] 在数组 B[ ] 存放于第 k 个位置，则有

$$i = \lfloor (k+1)/3 \rfloor$$

$$j = k - 2 * i$$

例如，当 $k = 8$ 时，

$$i = \lfloor (8+1)/3 \rfloor = 3,\ j = 8 - 2*3 = 2$$

当 k = 10 时，

$$i = \lfloor (10+1)/3 \rfloor = 3,\ j = 10 - 2*3 = 4$$

稀疏矩阵 (Sparse Matrix)

$$A_{6\times7}=\begin{pmatrix}0&0&0&22&0&0&15\\0&11&0&0&0&17&0\\0&0&0&-6&0&0&0\\0&0&0&0&0&39&0\\91&0&0&0&0&0&0\\0&0&28&0&0&0&0\end{pmatrix}$$

非零元素个数远远少于矩阵元素个数

在上图中,矩阵A是6*7的矩阵,它有42个元素,但只有8个非零元素,且分布无规律可循,所以可以称之为稀疏矩阵。

稀疏矩阵的抽象数据类型(三元组顺序表）

```
#define MAXSIZE 12500
typedef struct{
        int i,j;            //非零元素行号/列号
        ElemType e;         //非零元素的值
}Triple; //三元组
typedef struct{
        Triple data[MAXSIZE+1];
        int mu,nu,tu; //矩阵行数、列数、非零元个数
}TSMatrix;//稀疏矩阵类定义
```

稀疏矩阵

$$\begin{pmatrix} 0 & 0 & 0 & 22 & 0 & 0 & 15 \\ 0 & 11 & 0 & 0 & 0 & 17 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 39 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	行 (row)	列 (col)	值 (value)
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	-6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28

转置矩阵

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 91 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 28 \\ 22 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 39 & 0 & 0 \\ 15 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	行 (row)	列 (col)	值 (value)
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	−6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

用三元组表表示的稀疏矩阵及其转置

	行 (row)	列 (col)	值 (value)
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	−6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28

	行 (row)	列 (col)	值 (value)
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	−6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

❖稀疏矩阵转置算法思想

显然，一个稀疏矩阵的转置仍然是一个稀疏矩阵，方法是：设将矩阵M转置为矩阵T

（1）将矩阵的行列值交换

（2）将每一个三元组的i和j相互调换

（3）重排三元组之间的次序

可以有两种处理方法：

方法一：按照M(m*n)的列序来进行转置

设矩阵列数为nu，对矩阵三元组表扫描nu次。第k次检测列号为k的项。

第k次扫描找寻所有列号为k的项，将其行号变列号、列号变行号，顺次存于转置矩阵三元组表。

❖稀疏矩阵的转置

```
Status TransposeSMatrix(TSMatrix M, TSMatrix &T){
        T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;
        //转置矩阵的列数,行数和非零元素个数
        if (T.tu){
                q=1; //矩阵T的指针
                for(col=1;col<=M.nu;++col)
                        for(p=1;p<=M.tu;++p)//矩阵M的指针
                                if(M.data[p].j==col){
                                        T.data[q].i=M.data[p].j;
                                        T.data[q].j=M.data[p].i;
                                        T.data[q].e=M.data[p].e;
                                        ++q;
                                }
        }
        return OK;
}//TransposeSMatrix
```

该算法主要工作是在p*col的两重循环中做的，所以时间复杂度是O(nu*tu)。而一般矩阵的转置算法是在nu*mu的两重循环中做的，时间复杂度是O(nu*mu)。当稀疏矩阵的非零元个数tu=nu*mu时，其时间复杂度

$O(nu*tu)=O(nu*nu*mu)=O(nu^2*mu)$大大高于一般矩阵的时间复杂度，所以该算法仅适用于tu<<nu*mu的稀疏矩阵。

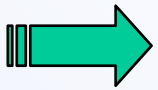
方法二：快速转置运算

在对M矩阵转置时，M矩阵的三元组中元素按行序排列，T矩阵中的元素按M矩阵的列序排列，前面的转置算法的特点是以T矩阵的三元组为中心，在M矩阵的三元组中通盘查找合适的结点置入T中。如果能预先确定M的每一列第一个非零元在T中应有的位置，则在转置时就可直接放到T中去，所以在转置前，应先求得M的**每一列中非零元的个数**和每一列的**第一个非零元在T中的位置**。

为此，需要两个辅助数组num和cpot,num表示M中第col列非零元素的个数。cpot表示M中第col列第一个非零元素在T中的位置。显然有：

$$\begin{cases} cpot[1]=1 \\ cpot[col]=cpot[col-1]+num[col-1] \end{cases}$$

	行 (row)	列 (col)	值 (value)
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	−6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28



	行 (row)	列 (col)	值 (value)
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	−6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

转置矩阵

矩阵M的辅助数组的值

Col	0	1	2	3	5	6
num[col]	1	1	1	2	2	1
cpot[col]	1	2	3	4	6	8

```
Status FastTransposeSMatrix(TSMatrix M, TSMatrix &T){
    T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;
    if(T.tu){
        for(col=1;col<=M.nu;++col) num[col]=0;//初始化num
        for(t=1;t<=M.tu;++t) ++num[M.data[t].j];//求M中每列非零元个数
        cpot[1]=1;
        for(col=2;col<=M.nu;++col) cpot[col]=cpot[col-1]+num[col-1];
    //求第col列中第一个非零元在T中的序号
        for(p=1;p<=M.tu;++p){
            col=M.data[p].j; q=cpot[col];
            T.data[q].i=M.data[p].j;
            T.data[q].j=M.data[p].i;
            T.data[q].e=M.data[p].e; ++cpot[col];//该列下一元素位置
        }//for
    }//if
    return OK;
}//FastTransposeSMatrix
```

行逻辑链接的顺序表

为便于随机存取任意一行的非零元，将快速转置矩阵的算法中的辅助数组cpot固定在稀疏矩阵的存储结构中。

```
typedef struct{
    Triple data[MAXSIZE+1];
    int rpos[MAXRC+1];
    int mu,nu,tu;
}RLSMatrix;
```

该存储方法便于某些运算如稀疏矩阵的相乘。

十字链表

以链式存储结构表示三元组的线性表。

广义表 (General Lists)

- **广义表的概念** n (≥ 0)个表元素组成的有限序列，记作

$$LS = (a_0, a_1, a_2, \ldots, a_{n-1})$$

LS是表名，a_i是表元素，它可以是表 (称为子表)，可以是数据元素(称为原子)。

- n为表的长度。 n = 0 的广义表为空表。
- n > 0时，表的第一个表元素称为广义表的表头(head)，除此之外，其它表元素组成的表称为广义表的表尾(tail)。

例如

A=( );//A是一个空表

B=(e);　　//表B有一个原子

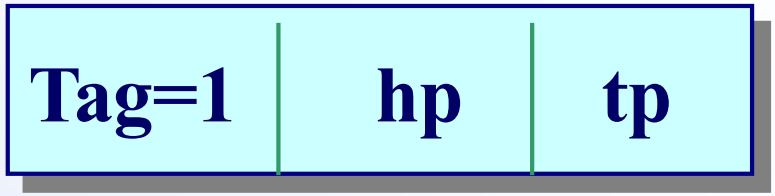
C=(a,(b,c,d));　　//两个元素为原子a和子表(b,c,d)

D=(A,B,C); //有三个元素均为列表

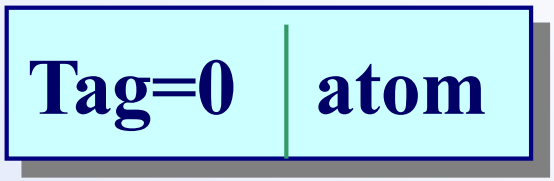
E=(a,E);　　//递归的列表,包含两个元素,一个是单元素a,另一个是子表,但该子表是其自身.所以,E相当于一个无限的广义表(a,(a,(a,…))).

广义表存储结构

方法一　表结点

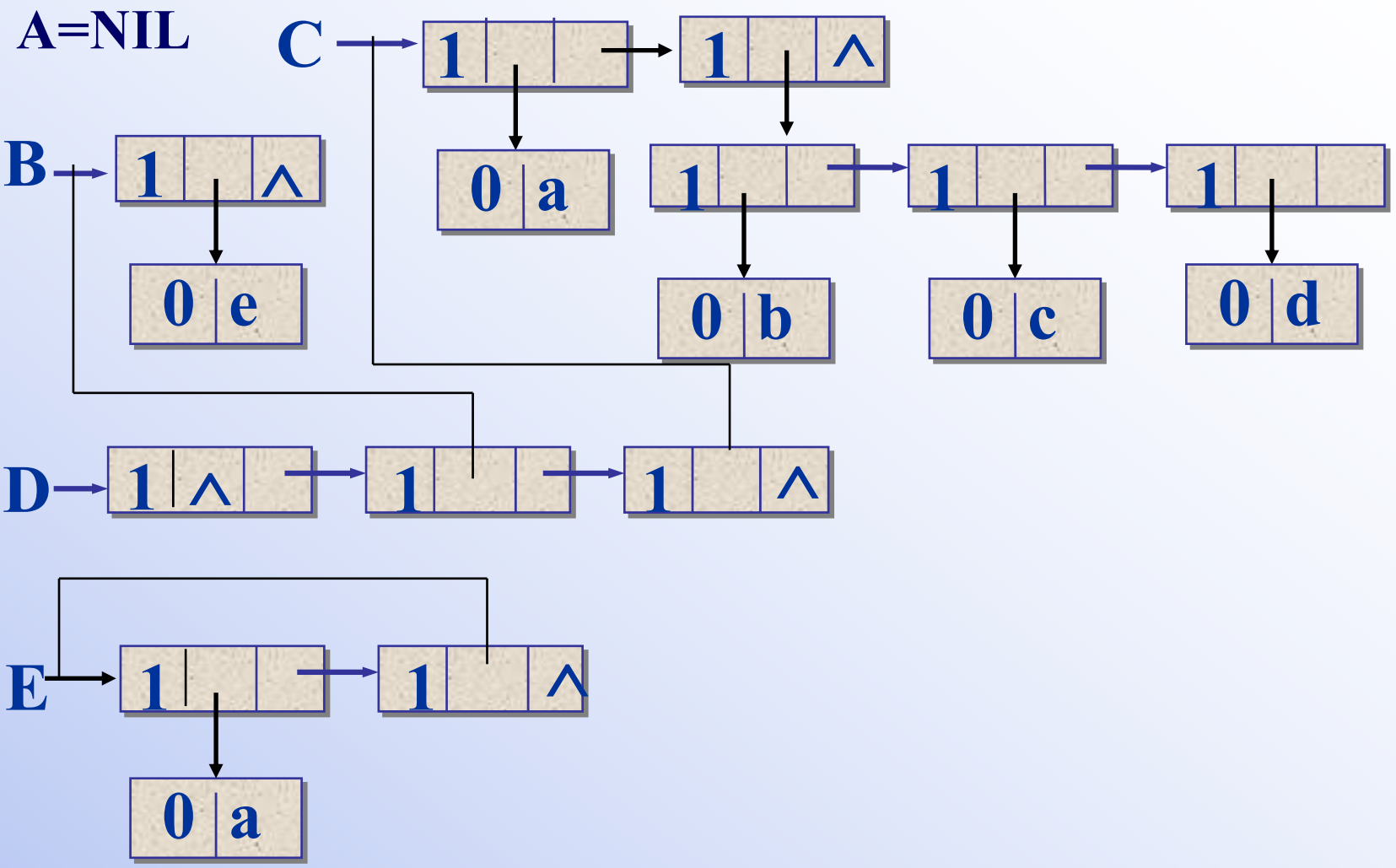


原子结点




```
typedef struct GLNode{
        int tag;
        union{
                char atom;
                struct {structGLNode *hp,*tp;}ptr;
        };
}*GList;
```

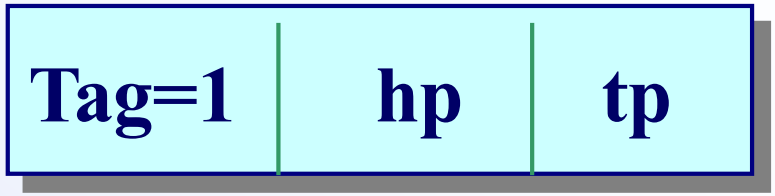
方法一



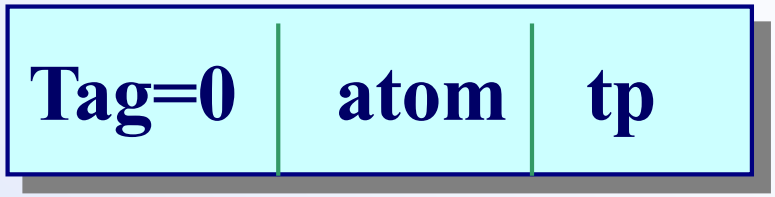
广义表存储结构

方法二

表结点

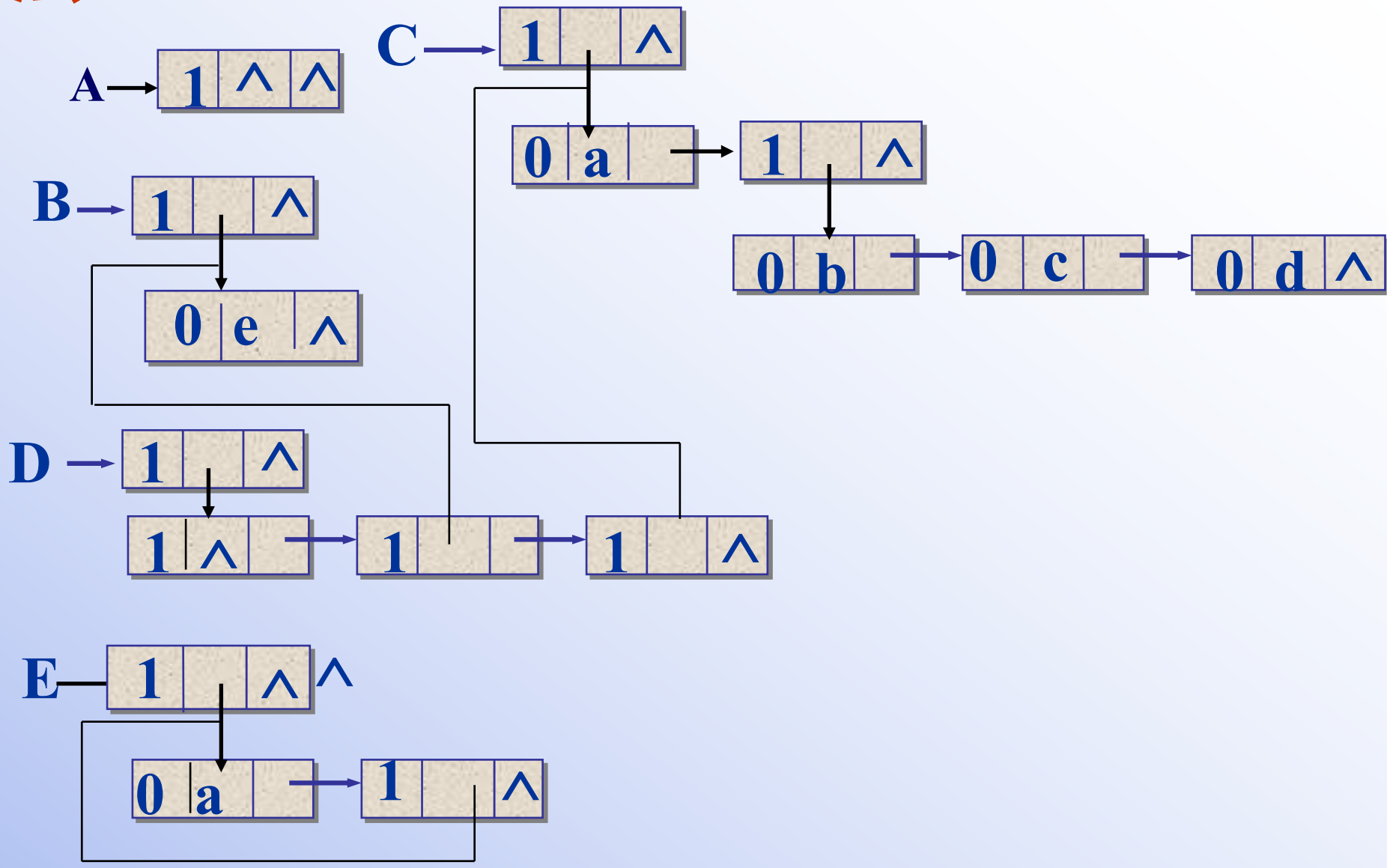


原子结点




```
typedef struct GLNode{
        int tag;
        union{
                char atom;
                struct GLNode *hp;
        };
        struct GLNode *tp;
}*GList;
```

方法二



广义表的递归算法

1.广义表的深度（广义表中括号的重数）

设非空广义表为LS=(a1,a2,…,an) 其中ai(i=1,2,…,n)或为原子或为LS的子表。原子的深度为零，空表的深度为1，其它情况下表长为各子表深度最大值加1。

```
int GListDepth(GList L){
    if(!L) return 1;
    if(L->tag==ATOM) return 0;
    for(max=0,pp=L;pp;pp=pp->ptr.tp){
        dep=GListDepth(pp->ptr.hp)
        if(dep>max) max=dep;
    }
    return max+1;
} //GListDepth
```

2.复制广义表

如果原表为空表，则直接将新表置空，否则分别复制原表的表头和表尾。

```
Status CopyGList(GList &T, GList L){
    if(!L) T=NULL;
    else{
        if(!(T=(GList)malloc(sizeof(GLNode)))) exit(OVERFLOW);
        T->tag=L->tag;
        if(L->tag==ATOM) T->atom=L->atom;
        else {CopyGList(T->ptr.hp,L->ptr.hp);
              CopyGList(T->ptu.tp,L->ptr.tp);
        }//else
    }//else
    return OK;
}//CopyGList
```