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

数组一旦定义，只做存取和修改操作，一般不做插入删除操作，因此使用顺序存储表示

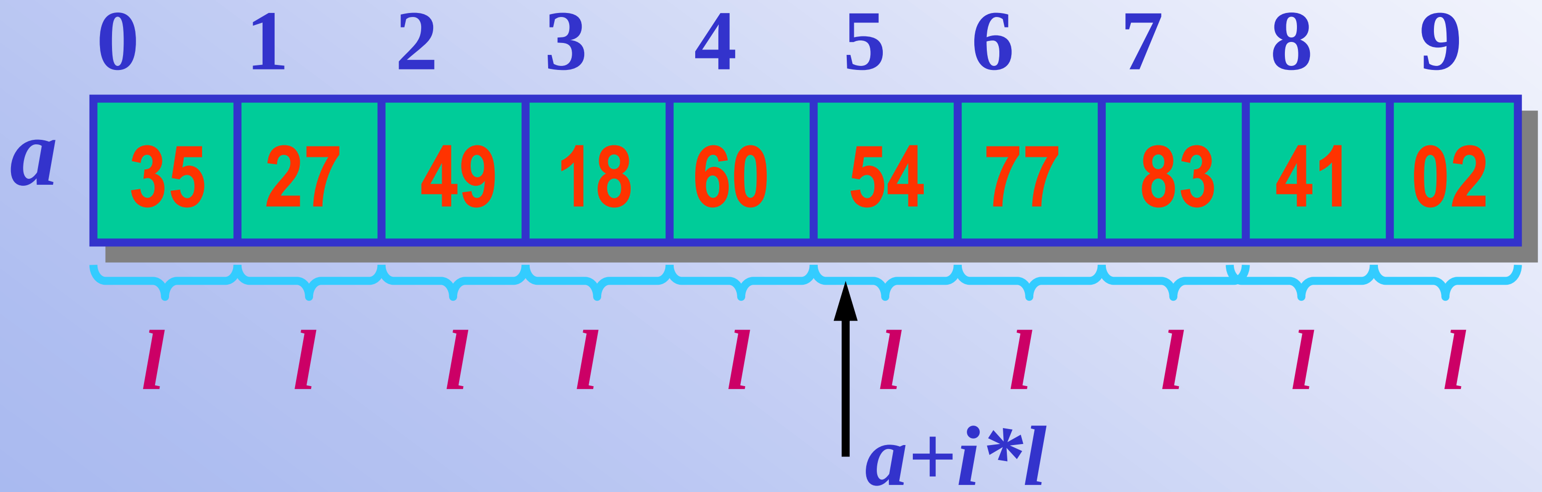
0	1	2	3	4	5	6	7	8	9
35	27	49	18	60	54	77	83	41	02

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

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



$$\mathrm{LOC}(i) = \mathrm{LOC}(i-1)+l = a+i*l$$

二维数组

n=1时的数组（一维数组）为定长线性表

n维数组为线性表的推广

一个二维数组的逻辑结构可形式地表示为：

2_Array=(D,R)

其中$D=\{a_{ij}(i=0,1,\ldots,m-1,j=0,1,\ldots,n-1)\}$, 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。

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

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

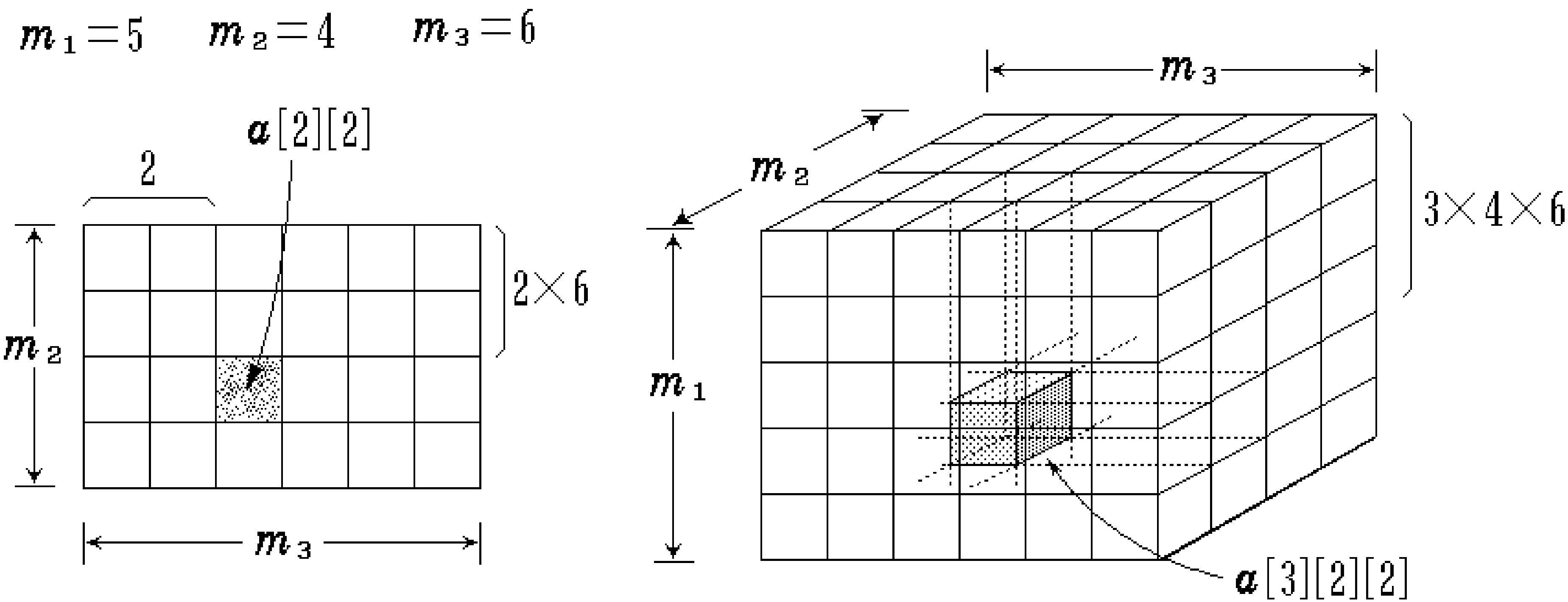
各维元素个数为 m_1, m_2, m_3

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)

$$LOC(i_1, i_2, i_3) = a + (\underbrace{i_1 * m_2 * m_3}_{\text{前}i_1\text{页总元素个数}} + \underbrace{i_2 * m_3}_{\text{第}i_1\text{页的前}i_2\text{行总元素个数}} + i_3) * l$$

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

三维数组以行为主序存储，其元素地址公式为：

LOC(Aijk)=LOC(Ac1c2c3)+[(i-c1)V2V3+(j-c2)V3+(k-c3)]*L

其中ci,di是各维的下界和上界

Vi=di-ci+1是各维元素个数

L是一个元素所占的存储单元数

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：

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

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

练习

在以行序为主序的存储结构中，下标从0开始计数，给出三维数组A2*3*4的地址的计算公式

练习

在以行序为主序的存储结构中，下标从0开始计数，给出三维数组A2*3*4的地址的计算公式

$LOC(A[i][j][k])$

$= LOC(A[0][0][0]) + i*12 + j*4 + k$

列序存储呢?

练习

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

(1)存放该数组所需多少单元?

(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?

练习

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

(1)存放该数组所需多少单元？ 11*11*32/16=242

(2)存放数组第4列所有元素至少需多少单元？

11*32/16=22

(3)数组按行存放时，元素A[7，4]的起始地址是多少？ s+(8*11+3)*32/16=s+182

(4)数组按列存放时，元素A[4，7]的起始地址是多少？ s+(6*11+5)*32/16=s+142

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、三角矩阵

对角矩阵、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，$a_{ij} = a_{ji}$
有近一半的数据冗余

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) + ··· + 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

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

下三角矩阵

需要存储 n + (n - 1) + ··· + 1 =n*(n+1)/2 个元素。

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

下三角矩阵

	0	1	2	3	4	5	6	7	8		n(n+1)/2-1
B	a_{00}	a_{10}	a_{11}	a_{20}	a_{21}	a_{22}	a_{30}	a_{31}	a_{32}	……	a_{n-1n-1}

若 i ≥ j, 数组元素A[i][j]在数组B中的存放位置为 $\underbrace{1 + 2 + \cdots + i}_{\text{前i行元素总数}} + \underbrace{j}_{\text{第i行第j个元素前元素个数}} = (i + 1) * i / 2 + j$

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$A[j][i]$：$= j *(j +1) / 2 + i$

反推呢？ $k \to (i, j)$

若已知某矩阵元素（i,j）,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, 为该元素的行号

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

j取值[0,i],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

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

	0	1	2	3	4	5	6	7	8	9
B	a_{00}	a_{01}	a_{02}	a_{03}	a_{11}	a_{12}	a_{13}	a_{22}	a_{23}	a_{33}

若$i \le j$，数组元素A[i][j]在数组B中的存放位置为 $\underbrace{n + (n-1) + (n-2) + \cdots + (n-i+1)}_{\text{前}i\text{行元素总数}} + \underbrace{j-i}_{\text{第}i\text{行第}j\text{个元素前元素个数}}$

若 $i \le j$，数组元素A[i][j]在数组B中的存放位置为

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

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

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

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 $(2*n-j-1) * j / 2 + i$ 的位置中找到。

三对角矩阵的压缩存储

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

	0	1	2	3	4	5	6	7	8	9	10
B	a_{00}	a_{01}	a_{10}	a_{11}	a_{12}	a_{21}	a_{22}	a_{23}	…	$a_{n\text{-}1n\text{-}2}$	$a_{n\text{-}1n\text{-}1}$

- 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。

反推呢?

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，

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

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。

稀疏矩阵 (Sparse Matrix)

$$A_{6\times 7} = \begin{pmatrix} 0 & 0 & 0 & 22 & 0 & 0 & 15 \\ 0 & 11 & 0 & 0 & 0 & 17 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 39 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 & 0 \end{pmatrix}$$

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]; //非零元素三元组表，data[0]未用
        int mu,nu,tu; //矩阵行数、列数、非零元素个数
}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)
[1]	1	4	22
[2]	1	7	15
[3]	2	2	11
[4]	2	6	17
[5]	3	4	−6
[6]	4	6	39
[7]	5	1	91
[8]	6	3	28

转置矩阵

	行 (row)	列 (col)	值 (value)
[1]	1	4	22
[2]	1	7	15
[3]	2	2	11
[4]	2	6	17
[5]	3	4	−6
[6]	4	6	39
[7]	5	1	91
[8]	6	3	28

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

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

	行 (row)	列 (col)	值 (value)
[1]	1	5	91
[2]	2	2	11
[3]	3	6	28
[4]	4	1	22
[5]	4	3	−6
[6]	6	2	17
[7]	6	4	39
[8]	7	1	16

用三元组表示的稀疏矩阵及其转置

	行 (row)	列 (col)	值 (value)
[1]	1	4	22
[2]	1	7	15
[3]	2	2	11
[4]	2	6	17
[5]	3	4	−6
[6]	4	6	39
[7]	5	1	91
[8]	6	3	28

	行 (row)	列 (col)	值 (value)
[1]	1	5	91
[2]	2	2	11
[3]	3	6	28
[4]	4	1	22
[5]	4	3	−6
[6]	6	2	17
[7]	6	4	39
[8]	7	1	16

稀疏矩阵转置算法思想

稀疏矩阵的转置仍然是稀疏矩阵，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
```

时间复杂度：nu*tu

适用于tu<<nu*mu

- 算法时间复杂度是O(nu*tu)
- 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^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

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**。

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

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

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

	行 (row)	列 (col)	值 (value)
[1]	1	4	22
[2]	1	7	15
[3]	2	2	11
[4]	2	6	17
[5]	3	4	−6
[6]	4	6	39
[7]	5	1	91
[8]	6	3	28

	行 (row)	列 (col)	值 (value)
[1]	1	5	91
[2]	2	2	11
[3]	3	6	28
[4]	4	1	22
[5]	4	3	−6
[6]	6	2	17
[7]	6	4	39
[8]	7	1	16

转置矩阵

矩阵M的辅助数组的值

Col	1	2	3	4	5	6	7
num[col]	1	1	1	2	0	2	1
cpot[col]	1	2	3	4	6	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;
        //求第col列中第一个非零元在T中的序号
        for(col=2;col<=M.nu;++col) cpot[col]=cpot[col-1]+num[col-1];
        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
```

时间复杂度O(nu+tu)

练习

特殊矩阵和稀疏矩阵哪一种压缩存储后失去随机存取的功能?

练习

特殊矩阵和稀疏矩阵哪一种压缩存储后失去随机存取的功能?

特殊矩阵是按某种原则压缩存储到一维数组上，有一一对应关系，可随机存取；稀疏矩阵的压缩存储方式为三元组顺序表，已不是简单的向量，不能随机存取。

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]; //各行第一个非零元位置表
    int mu,nu,tu;
}RLSMatrix;
```

该存储方法便于某些运算如稀疏矩阵的相乘

rpos为每行第一个非零元素的位置

稀疏矩阵的乘法操作

https://www.cnblogs.com/wangkundentisy/p/9267764.html

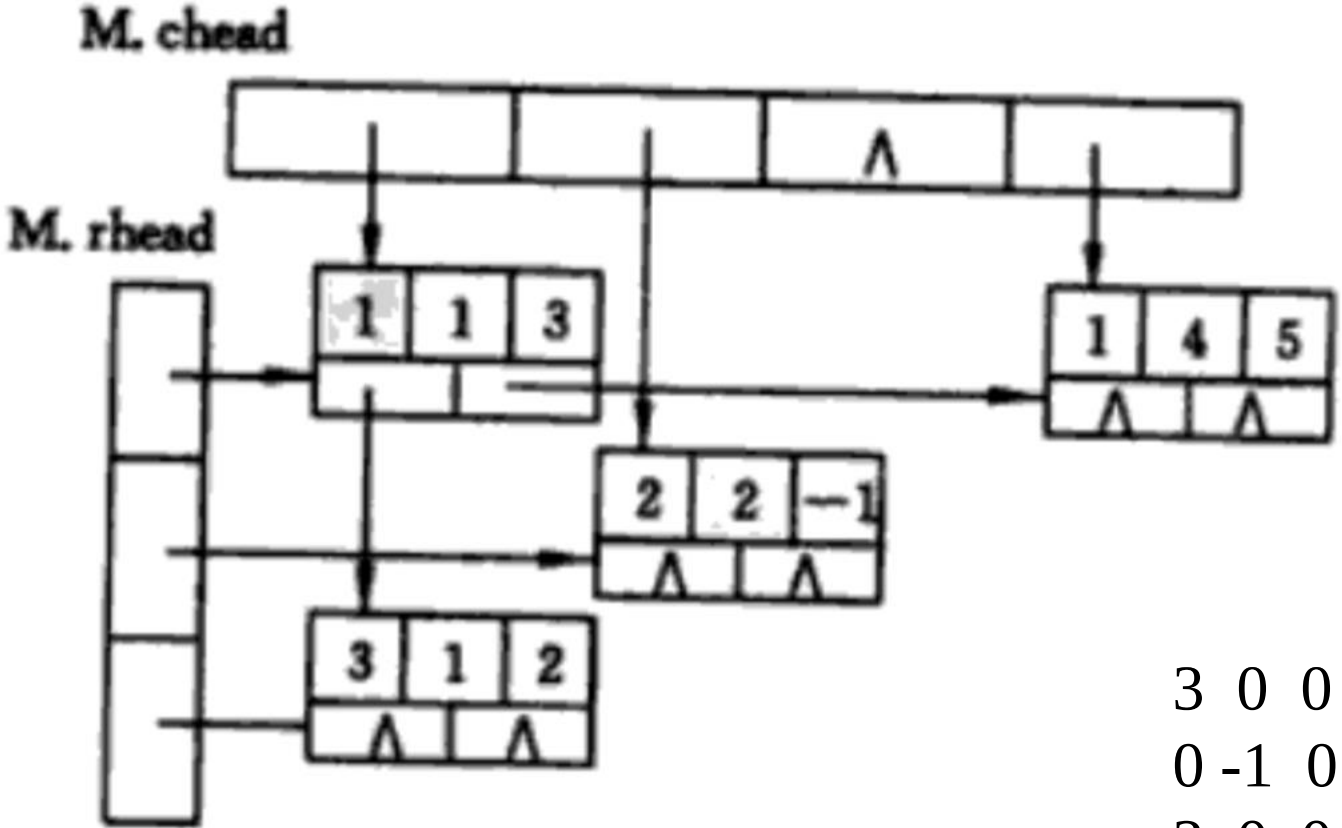
十字链表

当矩阵的非零元素个数和位置在操作中变化较大时（如矩阵加法或乘法），不宜采用顺序存储结构，以链式存储结构表示三元组的线性表。

i	j	e	right	down

right链接同一行中的下一个元素，down链接同一列中的而下一个元素

使用行链表头指针和列链表头指针



$$\begin{matrix} 3 & 0 & 0 & 5 \\ 0 & -1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{matrix}$$

图 5.6 稀疏矩阵 M 的十字链表

```
        // ----- 稀疏矩阵的十字链表存储表示 -----
typedef struct OLNode {
    int              i, j;                     // 该非零元的行和列下标
    ElemType         e;
    struct OLNode * right, * down;             // 该非零元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 CreateSMatrix 分配.
    int     mu, nu, tu;                        // 稀疏矩阵的行数、列数和非零元个数
}CrossList;
```

```
Status CreateSMatrix_OL (CrossList &M) {
  // 创建稀疏矩阵 M。采用十字链表存储表示。
  if (M) free(M);
  scanf(&m, &n, &t );      // 输入 M 的行数、列数和非零元个数
  M.mu := m;   M.nu := n;   M.tu := t;
  if (!(M.rhead = (OLink * )malloc((m + 1) * sizeof(OLink))))exit(OVERFLOW);
  if (!(M.chead = (OLink * )malloc((n + 1) * sizeof(OLink)))) exit(OVERFLOW);
  M.rhead [ ] = M.chead [ ] = NULL ;      // 初始化行列头指针向量;各行列链表为空链表
  for (scanf(&i, &j, &e); i!= 0; scanf(&i, &j, &e)) {// 按任意次序输入非零元
    if (!(p = (OLNode * )malloc(sizeof(OLNode)))) exit(OVERFLOW);
    p->i = i;   p->j = j;   p->e = e;                    // 生成结点
    if (M.rhead[i] == NULL || M.rhead [i]->j>j)   {p->right = M.rhead [i];M.rhead [i] = p;}
    else {        // 寻查在行表中的插入位置
      for ( q = M.rhead [i]; (q->right) && q->right->j < j;   q = q->right );
      p->right = q->right;   q ->right = p;          }  // 完成行插入
    if (M.chead [j] == NULL || M.chead[j]->i>i)   {p->down = M.chead [j]; M.chead [j] = p;}
    else {        // 寻查在列表中的插入位置
      for ( q = M.chead[j]; (q->down) && q->down->i < i;   q = q->down );
      p->down = q->down;   q->down = p;        }     //  完成列插入
  }
  return OK;
} // CreateSMatrix_OL
```

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)。

广义表可以看作是线性表的推广，线性表是广义表的特例

线性表是由**n**个数据元素组成的有限序列。其中每个组成元素被限定为单个元素

广义表的数据元素可以是一个称为广义表原子的元素，也可以是一个称为广义表子表的广义表

广义表很灵活，可以兼容线性表、数据、树、图等数据结构

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

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

- 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)。

A=( );//A是一个空表

B=(e);　　//表B有一个原子，表头为e，表尾为（）

C=(a,(b,c,d));　　//两个元素为原子a和子表 (b,c,d)，表头为a，表尾为（（bcd））

D=(A,B,C); //有三个元素均为列表，表头为A，表尾为（B，C）

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,…))).

F=(());　//含有一个表元素，该表元素为一个空表

练习

1 求广义表运算的结果

- HEAD[(a,b),(c,d)];
- TAIL[(a,b),(c,d)];

2 利用广义表的Head和Tail运算，把原子d分别从下列广义表中分离出来

L1 = ((((a),b),d),e))

练习

如果是HEAD[((a,b),(c,d))]呢？

1 求广义表运算的结果

- **HEAD[(a,b),(c,d)]; (a,b)**
- **TAIL[(a,b),(c,d)]; ((c,d))**

2 利用广义表的Head和Tail运算，把原子d分别从下列广义表中分离出来

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

Head（Tail（Head（Head（L））））

广义表存储结构

方法一

表结点

Tag=1	hp	tp

表示表头指针 表示表尾指针

原子结点

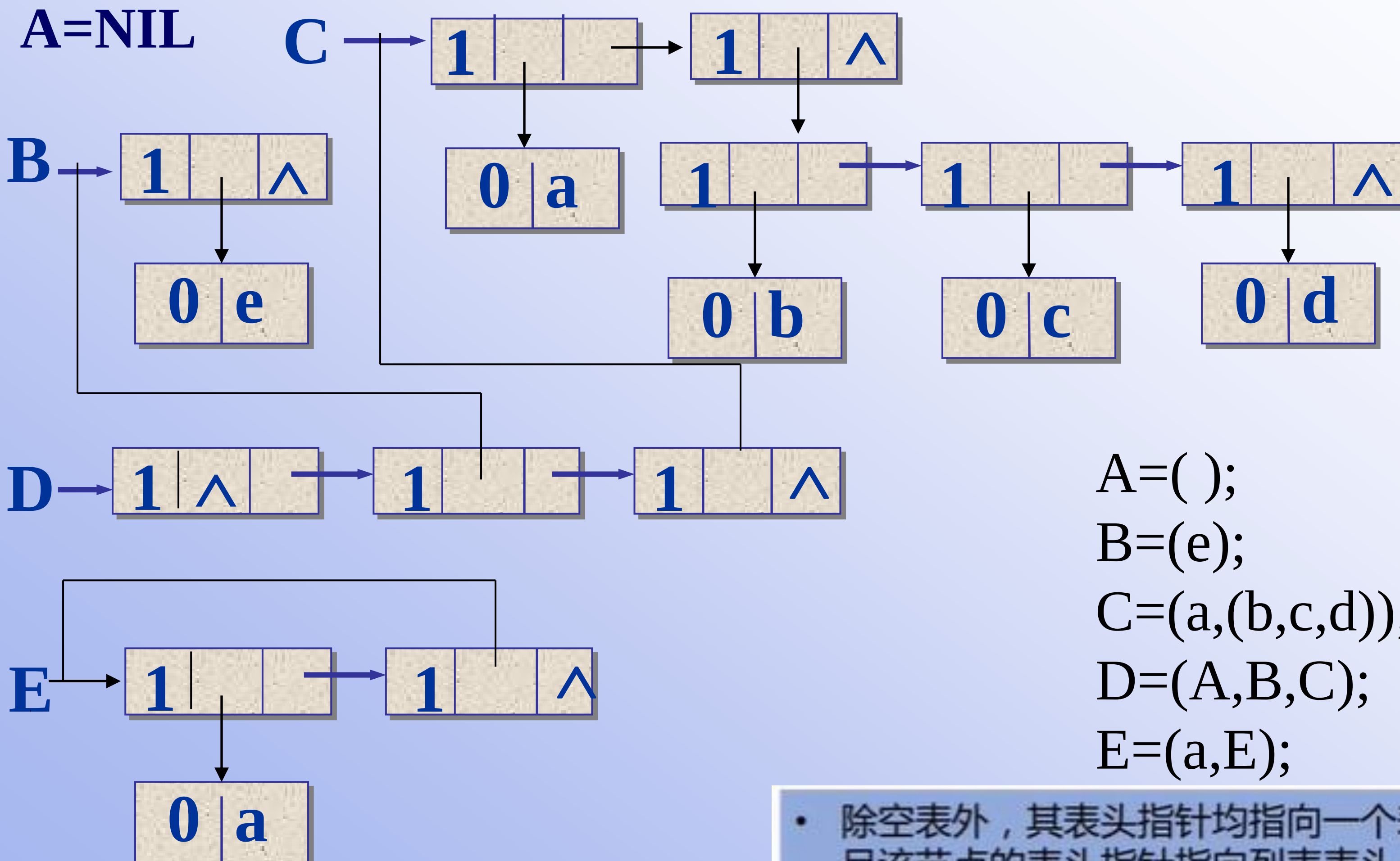
Tag=0	atom

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

方法一

表结点

原子结点



A=( );
B=(e);
C=(a,(b,c,d));
D=(A,B,C);
E=(a,E);

- 除空表外，其表头指针均指向一个表节点，且该节点的表头指针指向列表表头，表尾节点指向列表表尾；
- 容易分清列表中原子和子表的层次关系；
- 最高层表节点的个数即为表的长度。

广义表存储结构

方法二

表结点

Tag=1	hp	tp

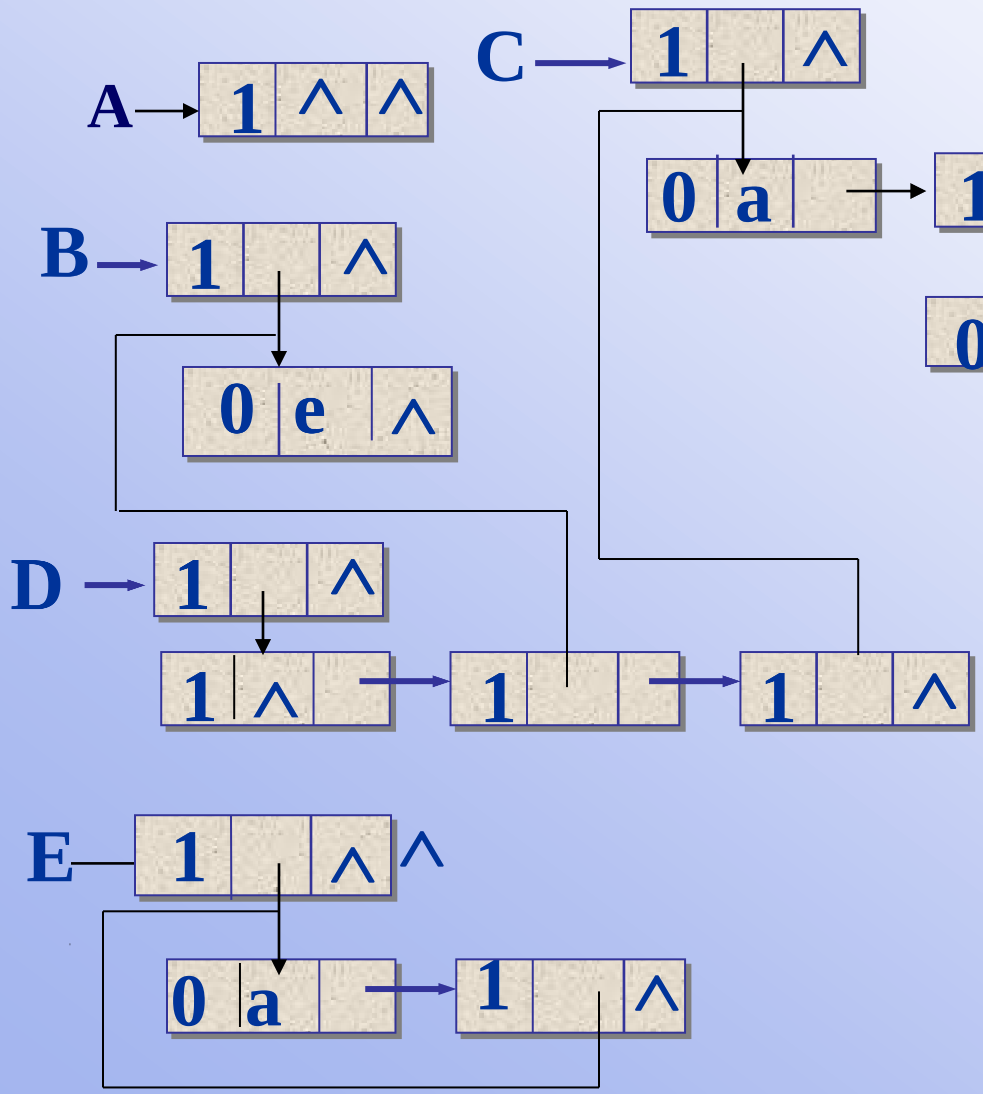
原子结点

Tag=0	atom	tp

```
typedef struct GLNode{
        int tag;
        union{
                char atom;
                struct GLNode *hp;//表结点的头指针
        };
        struct GLNode *tp;//指向下一个元素
}*GList;
```

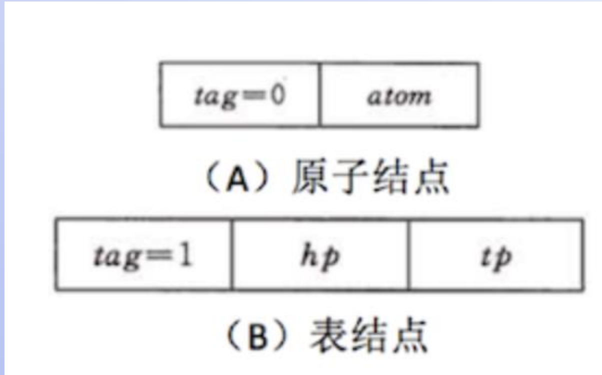
方法二

表结点	Tag=1	hp	tp
原子结点	Tag=0	atom	tp



A=( );
B=(e);
C=(a,(b,c,d));
D=(A,B,C);
E=(a,E);

总结



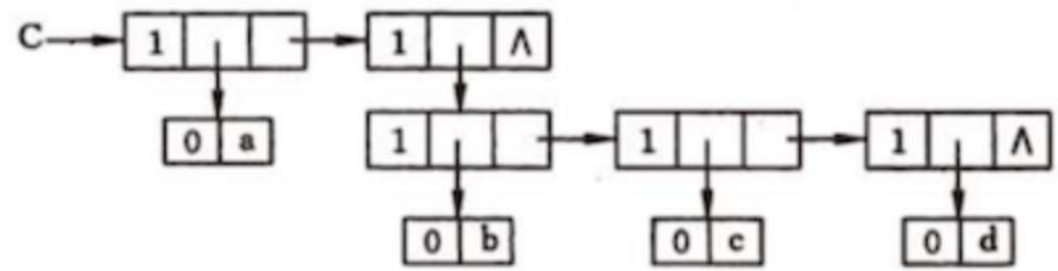
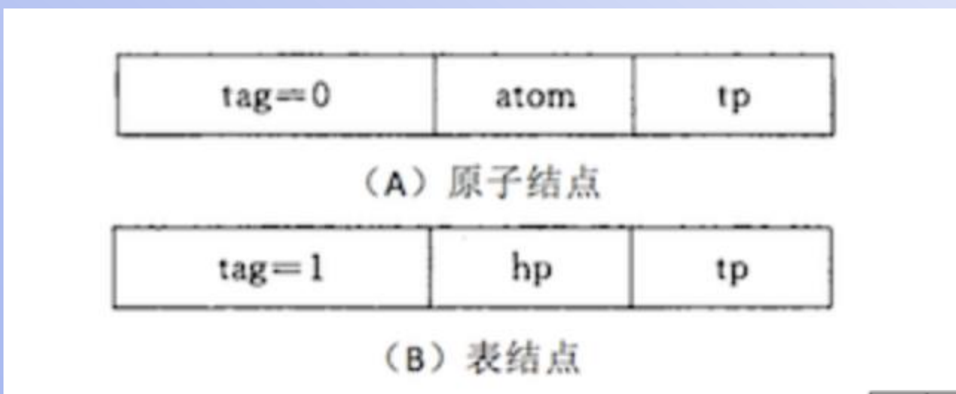


图 2 广义表 {a,{b,c,d}} 的结构示意图



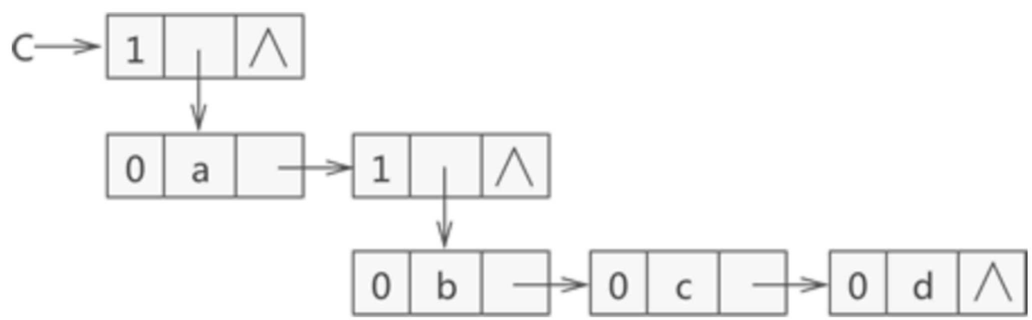
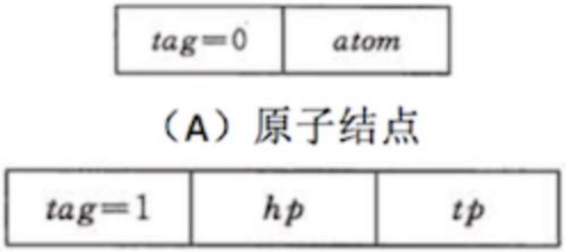


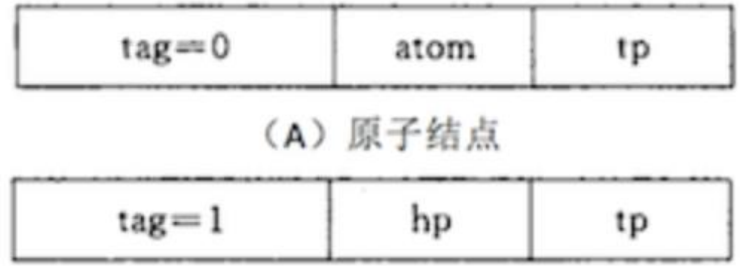
图 4 广义表 {a,{b,c,d}} 的存储结构示意图

无论采用以上哪一种节点结构存储广义表，都不要破坏广义表中各数据元素之间的并列关系

总结



(A) 原子结点
(B) 表结点



(A) 原子结点
(B) 表结点

```c
typedef struct GLNode{
    int tag;//标志域
    union{
        char atom;//原子结点的值域
        struct{
            struct GLNode * hp,*tp;
        }ptr;//子表结点的指针域，hp指向表头；tp指向表尾
    };
}*Glist;
```

```c
typedef struct GLNode{
    int tag;//标志域
    union{
        int atom;//原子结点的值域
        struct GLNode *hp;//子表结点的指针域，hp指向表头
    };
    struct GLNode * tp;//这里的tp相当于链表的next指针，用于指向下一个数据元素
}*Glist;
```

练习

画出下列广义表的存储结构图，并求它的深度

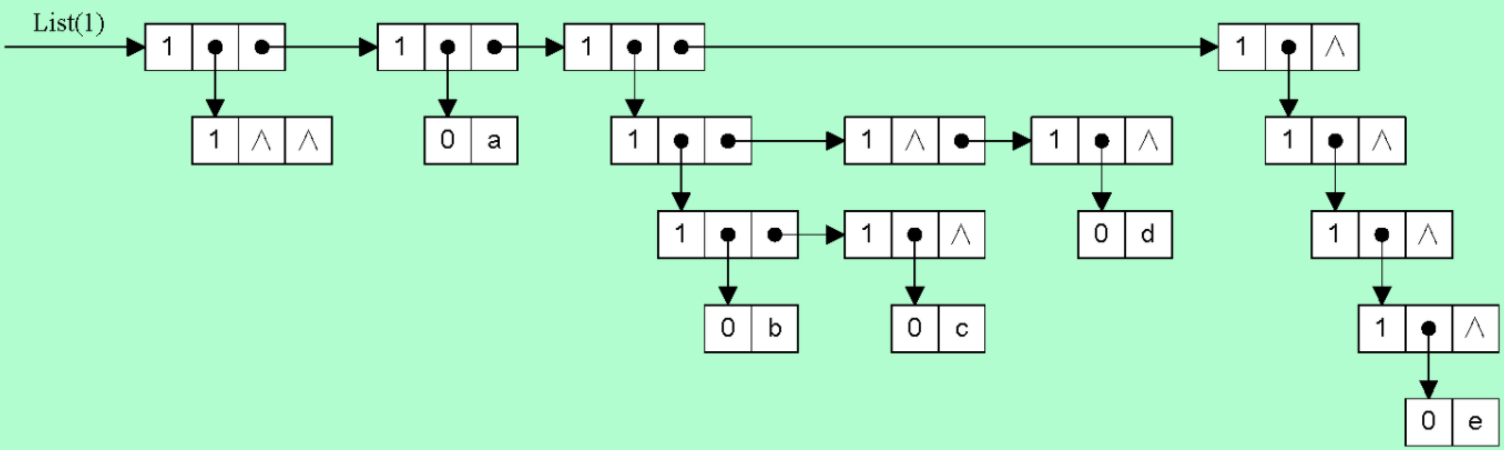
(1) (((())),a,((b,c),(),d),(((e))))

(2) ((((a),b)),(((),(d)),(e,f)))

画出下列广义表的存储结构图，并求它的深度

(1) (((()),a,((b,c),(),d),(((e))))

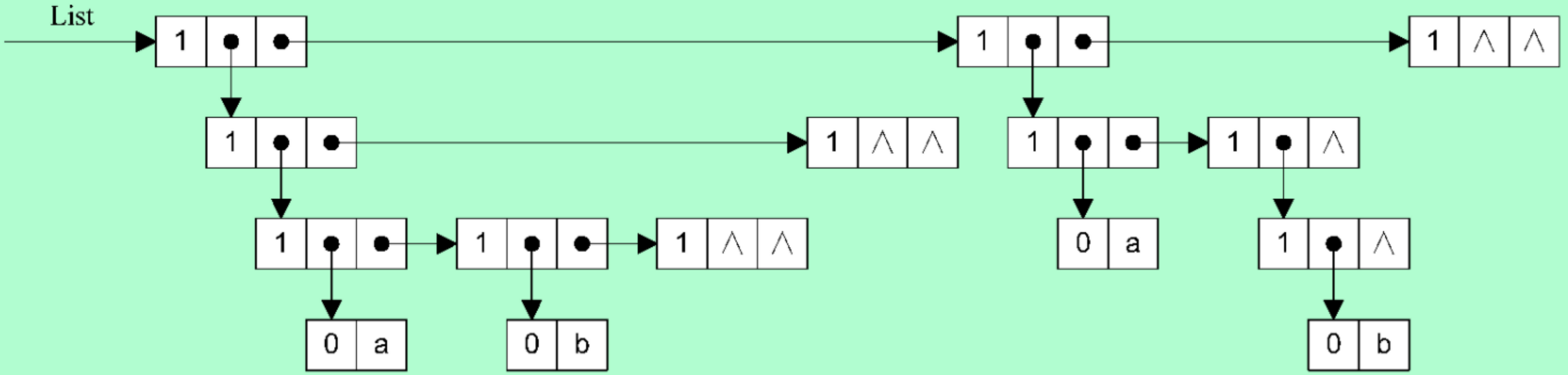
deep = 4.



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。

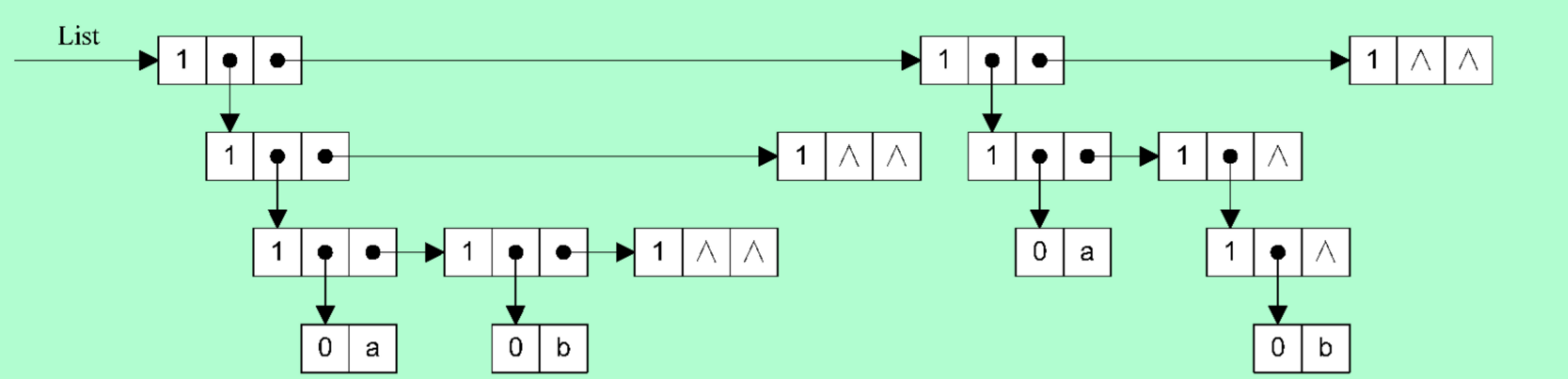
练习

已知下图为广义表的链接存储结构，写出该图表示的广义表。



练习

已知下图为广义表的链接存储结构，写出该图表示的广义表。



广义表：(((a,b,()),()),(a,(b)),())

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。

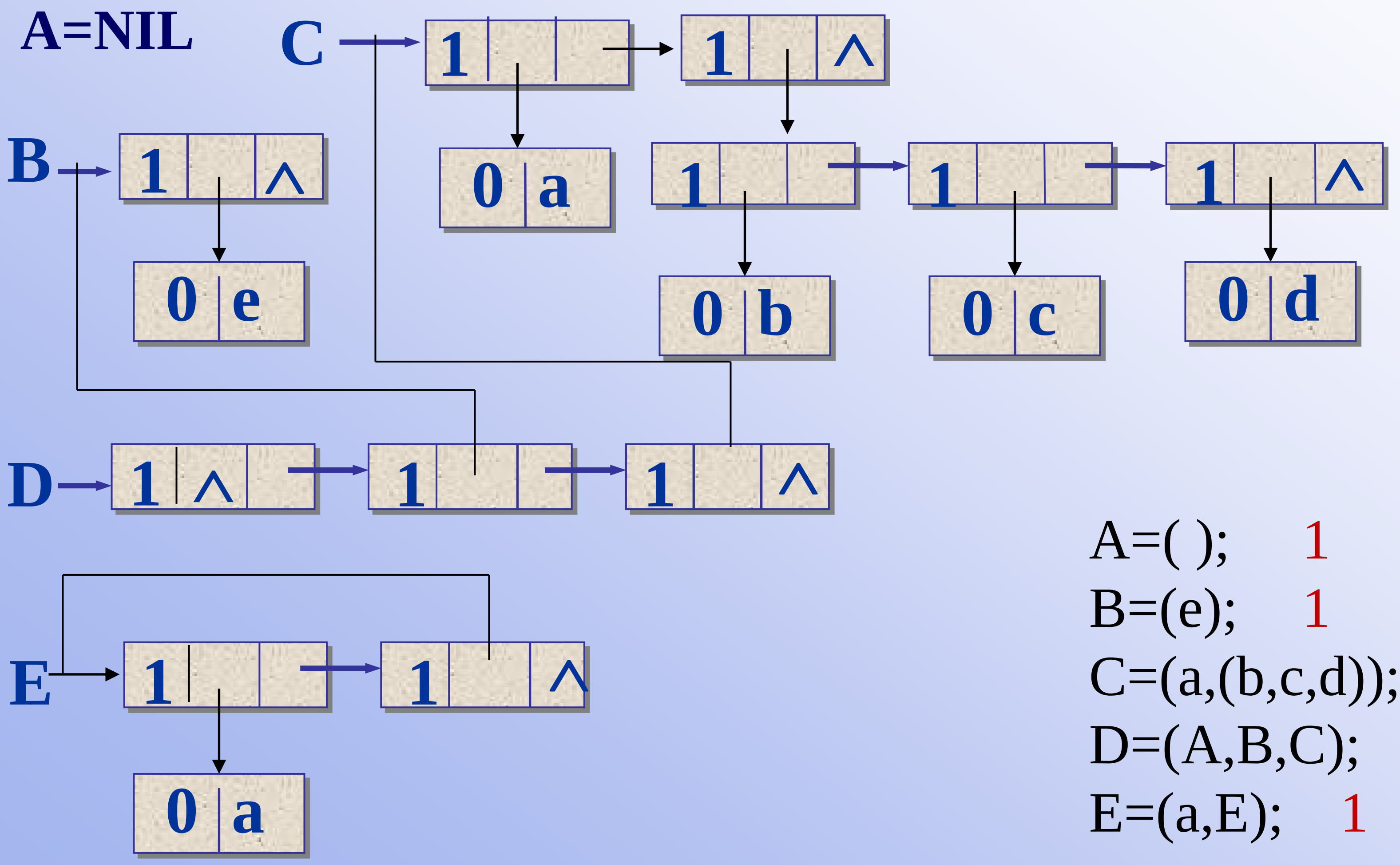
原子的深度为零，空表的深度为1，其它情况下表长为各子表深度最大值加1。

```
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
```

```
typedef struct GLNode{
    int tag;//标志域
    union{
        char atom;//原子结点的值域
        struct{
            struct GLNode * hp,*tp;
        }ptr;//子表结点的指针域，hp指向表头；tp指向表尾
    };
}*Glist;
```

通过pp获取到表中的每一个元素

```
for(max=0,pp=L;pp;pp=pp->ptr.tp){
    dep=GListDepth(pp->ptr.hp)
    if(dep>max) max=dep;
}
```



A=( ); 1

B=(e); 1

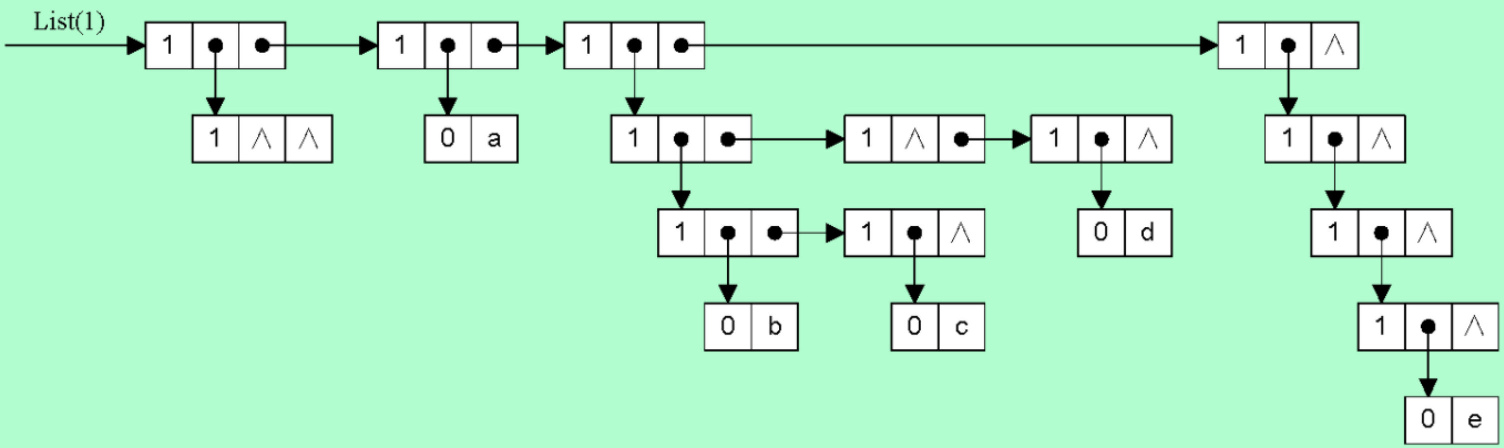
C=(a,(b,c,d)); 2

D=(A,B,C); 3

E=(a,E); 1

```
for(max=0,pp=L;pp;pp=pp->ptr.tp){
    dep=GListDepth(pp->ptr.hp)
    if(dep>max) max=dep;
}
```

deep = 4.



(((()),a,((b,c),(),d),(((e))))

2.复制广义表

如果原表为空表，则直接将新表置空，否则复制原子或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->ptr.tp,L->ptr.tp);
        }//else
    }//else
    return OK;
}//CopyGList
```

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

谢谢！