

# 第五章 数组和广义表

# 第五章 数组和广义表

## \* 内容提要:

- \* 本章主要内容是：数组的类型定义和存储结构；特殊矩阵和稀疏矩阵的压缩存储方法；广义表的逻辑结构和存储结构。
- \* 重点要求掌握数组转置等基本运算、稀疏矩阵的三元组形式的顺序表示和十字链表的表示方法、用三元组的形式来实现稀疏矩阵的转置运算。

前4章介绍的数据结构共同特点：

- （1）都属于线性数据结构；
- （2）每种数据结构中的数据元素，都作为原子数据，不再进行分解；

本章讨论的数据结构：**数组**，其特点是：

- 1) 从逻辑结构上看，**可看成是线性结构的一种扩展**；
- 2) 数据元素本身也是一个数据结构。

- \* **一维数组具有线性表的结构**，但操作简单，一般不进行插入和删除操作，只定义给定下标读取元素和修改元素的操作
- \* **二维数组**中，每个数据元素对应一对数组下标，在行方向上和列方向上都存在一个线性关系，即存在两个前驱（前件）和两个后继（后件）。也可看作是以线性表为数据元素的线性表。
- \* **n维数组**中，每个数据元素对应n个下标，受n个关系的制约，其中任一个关系都是线性关系。可看作是数据元素为n-1维数组的一维数组。
- \* 因此，多维数组和广义表是对**线性表的扩展**：线性表中的数据元素本身又是一个多层次的线性表。

# 第五章 数组和广义表

## 5.1 数组的定义

## 5.2 数组的顺序表示和实现

## 5.3 矩阵的压缩存储

## 5.4 广义表的定义

# 5.1 数组的定义

## 抽象数据类型

ADT Array{

数据对象：  $j_i=0, \dots, b_i-1, i=1, 2, \dots, n,$

$D=\{a_{j_1j_2\dots j_n} \mid n(>0) \text{ 称为数组的维数, } b_i \text{ 是数组第 } i \text{ 维的长度, } j_i \text{ 是数组元素的第 } i \text{ 维下标,}$

$a_{j_1j_2\dots j_n} \in \text{ElemSet}\}$

数据关系：  $R=\{R_1, R_2, \dots, R_n\}$

$R_i=\{ \langle a_{j_1\dots j_i\dots j_n}, a_{j_1\dots j_{i+1}\dots j_n} \rangle \mid 0 \leq j_k \leq b_k-1, 1 \leq k \leq n \text{ 且 } k \neq i, 0 \leq j_i \leq b_i-2,$

$a_{j_1\dots j_i\dots j_n}, a_{j_1\dots j_{i+1}\dots j_n} \in D, i=2, \dots, n\}$

## 基本操作：

- \* InitArray(&A, n, bound1,...,boundn)  
已知维数n和各维长度，构造数组A。
- \* DestroyArray(&A)
- \* Value(A, &e, index1,...,indexn)  
返回指定下标的A的元素值给e。
- \* Assign(&A, e, index1,...,indexn)  
将e的值赋给指定下标的A的元素。

}ADT Array

- \* 一个二维数组的逻辑结构可以定义成：

$$\text{Array\_2}=(D,R)$$

- \* 其中  $D=\{a_{ij} \mid i=c_1, c_1+1, \dots, d_1, j=c_2, c_2+1, \dots, d_2, a_{ij} \in D_0\}$

- \*  $R=\{\text{ROW}, \text{COL}\}$

- \*  $\text{ROW}=\{ \langle a_{ij}, a_{i',j+1} \rangle \mid c_1 \leq i \leq d_1, c_2 \leq j \leq d_2-1, a_{ij}, a_{i',j+1} \in D \}$

- \*  $\text{COL}=\{ \langle a_{ij}, a_{i+1,j} \rangle \mid c_1 \leq i \leq d_1-1, c_2 \leq j \leq d_2, a_{ij}, a_{i+1,j} \in D \}$



# 数组的概念

**数组**：由一组个数固定，类型相同的数据元素组成阵列。

以二维数组为例：二维数组中的每个元素都受两个线性关系的约束，即行关系和列关系，在每个关系中，每个元素 $a_{ij}$ 都有且仅有一个直接前趋，都有且仅有一个直接后继。

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

在行关系中

$a_{ij}$  直接前趋是  $a_{i \ j-1}$

$a_{ij}$  直接后继是  $a_{i \ j+1}$

在列关系中

$a_{ij}$  直接前趋是  $a_{i-1 \ j}$

$a_{ij}$  直接后继是  $a_{i+1 \ j}$

**二维数组：** 其每一个数据元素也是一个线性表的线性表

$$A = (\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \dots, \alpha_p)$$

其中每一个数据元素  $\alpha_j$  是一个列向量的线性表

$$\alpha_j = (a_{0j}, a_{1j}, a_{2j}, a_{3j}, \dots, a_{m-1j})$$

或  $\alpha_i$  是一个行向量的线性表

$$\alpha_i = (a_{i0}, a_{i1}, a_{i2}, a_{i3}, \dots, a_{in-1})$$

## 5.2 数组的顺序表示

- \* 数组的结构确定后（给定维数和各维长度），数组的存储空间确定。
- \* 一维数组在内存中的存放很简单，只要顺序存放在连续的内存单元即可。
- \* 多维数组用一维的存储单元存放需约定次序。PASCAL和C语言是以行序为主序，FORTRAN以列序为主序。

例：二维数组，如何用顺序结构表示？

内存地址是一维的，而数组是二维的，要将二维数组挤入一维的地址中，有两个策略：

以行为主序（C语言使用）

以列为主序（FORTRAN语言）

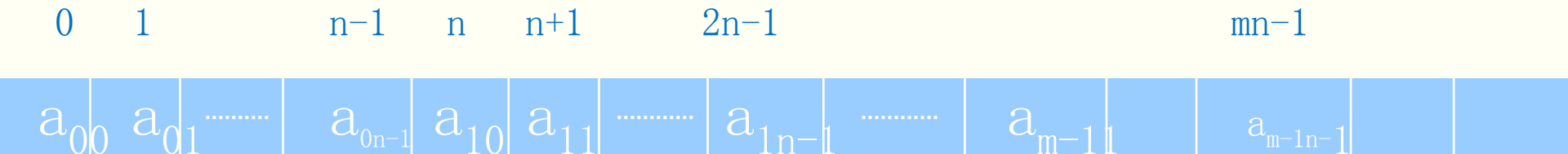
$$\begin{pmatrix} a_{00} & a_{01} & & a_{0n-1} \\ a_{10} & a_{11} & & a_{1n-1} \\ & & & \\ & & & \\ a_{m-10} & a_{m-11} & & a_{m-1n-1} \end{pmatrix}$$



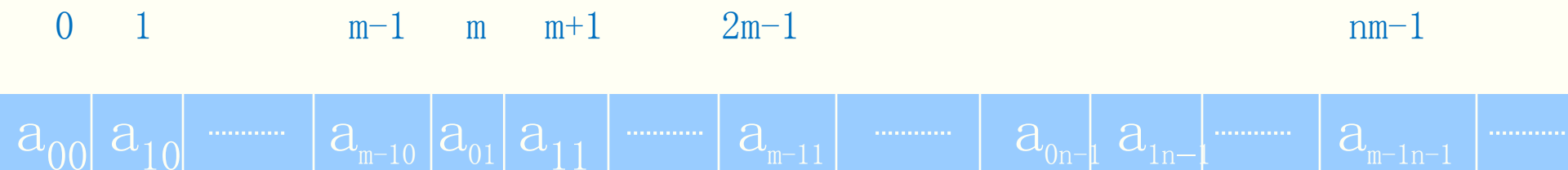
设A是一个具有m 行n列  
 的元素的二维数组：

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

以行为主序的方式：



以列为主序的方式：



- 二维数组中任一元素 $a_{ij}$ 的存储地址：
  - 按行存放
    - ✓  $\text{Loc}(i, j) = \text{Loc}(0, 0) + (b_2 * i + j) * L$
    - ✓  $L$ ：每个数据元素占据的存储单元元素
  - 按列存放
    - ✓  $\text{Loc}(i, j) = \text{Loc}(0, 0) + (b_1 * j + i) * L$
- $n$ 维数组：按行存放 p93
  - ✓  $\text{Loc}(j_1, j_2, \dots, j_n) = \text{Loc}(0, 0, \dots, 0) + \sum c_i j_i$
  - ✓ 其中  $c_n = L, c_{i-1} = b_i * c_i, 1 < i \leq n$

对于高维数组的情况：**行优先顺序**可规定为先排最右的下标，从右到左，最后排最左下标：**列优先顺序**与此相反，先排最左下标，从左向右，最后排最右下标。

按上述两种方式**顺序存储**的数组，只要知道开始节点的存放地址（即基地址），维数和每维的上、下界，以及每个数组元素所占用的单元数，就可以将数组元素的存放地址表示为其下标的线性函数。因此，数组中的任一元素可以在相同的时间内存取，即顺序存储的数组是一个**随机存取结构**。

例如：二维数组 $A_{mn}$ 按“**行优先顺序**”存储在内存中，假设每个元素占用 $d$ 个存储单元。元素 $a_{ij}$ 的存储地址应是数组的基地址加上排在 $a_{ij}$ 前面的元素所占用的单元数。因为 $a_{ij}$ 位于第 $i$ 行、第 $j$ 列，前面 $i$ 行一共有 $i \times n$ 个元素，第 $i$ 行上 $a_{ij}$ 前面又有 $j$ 个元素，故它前面一共有 $i \times n + j$ 个元素，因此， $a_{ij}$ 的地址计算函数为：

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

同样，三维数组 $A_{ijk}$ 按“**行优先顺序**”存储，其地址计算函数为：

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



## 5.3 矩阵的压缩存储

- 矩阵一般可用二维数组实现，特殊矩阵采用压缩存储。
- **压缩存储**：为多个值相同的元只分配一个存储空间，对零元不分配空间。
- **特殊矩阵**：值相同的元素或者零元素在矩阵中的分布有一定规律
- **稀疏矩阵**：非零元较零元少，且分布没有一定规律的矩阵

# 5.3.1. 特殊矩阵

## 1、对称矩阵的压缩存储

$$a_{ij} = a_{ji} \quad 1 \leq i, j \leq n$$

例：

$$\begin{pmatrix} 1 & 5 & 1 & 3 & 7 \\ 5 & 0 & 8 & 0 & 0 \\ 1 & 8 & 9 & 2 & 6 \\ 3 & 0 & 2 & 5 & 1 \\ 7 & 0 & 6 & 1 & 3 \end{pmatrix}$$

$a_{11}$

$a_{21} \quad a_{22}$

$a_{31} \quad a_{32} \quad a_{33}$

.....

$a_{n1} \quad a_{n2} \quad a_{n3} \dots a_{nn}$

- \* 压缩存储方法：为每一对对称元分配一个存储空间
  - \* 将下三角的元素（包括对角线），按行存储到一维数组  $sa \ [n(n+1)/2]$  中
  - \* 共有  $n(n+1)/2$  个存储单元，

为了便于访问对称矩阵A中的元素，必须在 $a_{ij}$ 和 $sa[k]$ 之间找一个对应关系。

若 $i \geq j$ ，则 $a_{ij}$ 在下三角形中。 $a_{ij}$ 之前的 $i-1$ 行一共有 $1+2+\cdots+i-1=i(i-1)/2$ 个元素，在第 $i$ 行上， $a_{ij}$ 之前恰有 $j-1$ 个元素，因此有：

$$k = i*(i-1)/2 + j-1$$
$$0 \leq k < n(n+1)/2$$

若 $i < j$ ，则 $a_{ij}$ 是在上三角矩阵中。因为 $a_{ij} = a_{ji}$ ，所以只要交换上述对应关系式中的 $i$ 和 $j$ 即可得到：

$$k = j * (j - 1) / 2 + i - 1 \quad 0 \leq k < n(n + 1) / 2$$

令  $I = \max(i, j)$ ，  $J = \min(i, j)$ ，则 $k$ 和  $i, j$ 的对应关系可统一为：

$$k = I * (I - 1) / 2 + J - 1 \quad 0 \leq k < n(n + 1) / 2$$

因此， $a_{ij}$ 的地址：

$$\text{LOC}(a_{ij}) = \text{LOC}(sa[k])$$

$$= \text{LOC}(sa[0]) + k * d = \text{LOC}(sa[0] + [I * (I - 1) / 2 + J - 1] * d)$$

有了上述的下标交换关系，对于任意给定一组下标  $(i, j)$ ，均可在  $sa[k]$  中找到矩阵元素  $a_{ij}$ ，反之，对所有的  $k = 0, 1, 2, \dots, n(n+1)/2 - 1$ ，都能确定  $sa[k]$  中的元素在矩阵中的位置  $(i, j)$ 。由此，称  $sa[n(n+1)/2]$  为阶对称矩阵  $A$  的压缩存储：

$a_{11}$	$a_{21}$	$a_{22}$	$a_{31}$	.....	$a_{n1}$	...	$a_{n,n}$
$k=0$	1	2	3		$n(n-1)/2$	...	$n(n+1)/2-1$

例如  $a_{21}$  和  $a_{12}$  均存储在  $sa[2]$  中，这是因为

$$k = I * (I - 1) / 2 + J - 1 = 2 * (2 - 1) / 2 + 1 - 1 = 1$$

## 2、三角矩阵的压缩存储

以主对角线划分，三角矩阵有上三角和下三角两种。

上三角矩阵的下三角（不包括主对角线）中的元素均为常数。下三角矩阵正好相反，它的主对角线上方均为常数。在大多数情况下，三角矩阵常数为零。

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ c & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ c & c & \dots & a_{nn} \end{pmatrix}$$

(a)上三角矩阵

$$\begin{pmatrix} a_{11} & c & \dots & c \\ a_{21} & a_{22} & \dots & c \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

(b)下三角矩阵

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到向量 $sa[0..n(n+1)/2]$ 中，其中 $c$ 存放在向量的最后一个分量中。

●下三角：  $k=i*(i-1)/2+j-1$

●上三角：  $k=(2n-i)(i-1)/2+j-1$ （按行）

$k=j(j-1)/2+i-1$ （按列）

●注意：  $k$ 从零开始， $i, j$ 从1开始

### 3、对角矩阵的压缩存储

●对角矩阵：所有非零元都集中在以主对角线为中心的带状区域中。

●压缩方法：压缩存储到一维数组sa[ ]中，三对角矩阵有 $3n-2$ 个元素。

● $k=2*i+j-3$

$$\begin{pmatrix} a_{11} & a_{12} & & & & \\ & a_{21} & a_{22} & a_{23} & & \\ & & a_{32} & a_{33} & a_{34} & \\ & & & & \dots & \dots & \dots \\ & & & & & & a_{n-1\ n-2} & a_{n-1\ n-1} & a_{n-1\ n} \\ & & & & & & & a_{n\ n-1} & a_{n\ n} \end{pmatrix}$$



## 5.3.2. 稀疏矩阵

什么是稀疏矩阵？

简单说，设在 $m \times n$ 矩阵 $A$ 中有 $t$ 个非零元素，若 $t$ 远远小于矩阵中零元素的个数 $m \times n - t$ ，则称 $A$ 为稀疏矩阵。

精确点，设在矩阵 $A$ 中，有 $t$ 个非零元素。令 $e = t / (m * n)$ ，称 $e$ 为矩阵的稀疏因子。通常认为 $e \leq 0.05$ 时称之为稀疏矩阵。

## 三元组的表示：

### 压缩存储

- 稀疏矩阵可由表示非零元的三元组及其行列数唯一确定。
- 用三元组( $i, j, a_{ij}$ )存储行和列的位置，及非零元数值。

例：稀疏矩阵M

$$M = \begin{bmatrix} 2 & 0 & 0 & 0 & 6 & 0 & 0 & 7 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -2 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 & 0 \\ 0 & 9 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

\* 矩阵M中非零元的三元组表示：

$(1, 1, 2)$  ,  $(1, 5, 6)$  ,  $(1, 8, 7)$  ,  $(2, 3, 1)$  ,  $(3, 3, -2)$  ,  $(3, 7, 3)$  ,  $(4, 6, 8)$  ,  $(5, 4, 5)$  ,  $(6, 2, 9)$

\* 以上是按照行号顺序，将三元组9个非零元素按顺序进行排列（当然也可以按照列号的顺序进行排列），如果再加上一个表示矩阵行数、列数和总的非零元素数目的特殊三元组  $(6, 8, 9)$  ，就可以唯一的确定一个矩阵。

# 1、三元组顺序表

- \*用顺序存储结构表示**三元组表**（Triple Table），来实现对稀疏矩阵的一种压缩存储形式，就称为**三元组顺序表**，简称三元组表。

## 稀疏矩阵的三元组顺序表存储表示:

```
#define MAXSIZE 12500 //非零元个数最大值  
typedef struct {
```

```
    int i, j; //行下标和列下标
```

```
    ElemType e;
```

```
    } Triple;
```

```
typedef struct {
```

```
    Triple data[MAXSIZE+1]; //非零元三元组表
```

```
    int mu, nu, tu; //行数、列数、非零元个数
```

```
} TSMatrix;
```

```
TSMatrix a, b;
```

所需空间:  $3*tu+3$       若  $> mu * nu$ , 则不必用三元组表示  $-mu, nu, tu$

# (1) 稀疏矩阵 (Sparse Matrix)

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

行数 $m = 6$ , 列数 $n = 7$ , 非零元素个数 $t = 8$

稀疏矩阵

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

转置矩阵

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



# 用三元组表表示的稀疏矩阵及其转置

行 ( <i>row</i> )	列 ( <i>col</i> )	值 ( <i>value</i> )
1	4	22
1	7	15
2	2	11
2	6	17
3	4	-6
4	6	39
5	1	91
6	3	28

行 ( <i>row</i> )	列 ( <i>col</i> )	值 ( <i>value</i> )
1	5	91
2	2	11
3	6	28
4	1	22
4	3	-6
6	2	17
6	4	39
7	1	15

# 三元组表稀疏矩阵的转置运算

- 方法：按照b.data中的三元组的次序，即M的列序，依次在a.data中找到相应的三元组进行转置。
- 步骤：从 $k=1$ 开始依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。
- 其时间复杂度为  $O(a.nu * a.tu)$ 。
- 例：若矩阵有200行，200列，10,000个非零元素，总共有2,000,000次处理。

## 稀疏矩阵的转置 （算法5.1）

```
Status TransposeSMatrix(TSMatrix M, TSMatrix &T)
{
    int q, col, p;
    T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;
    if (T.tu)
    {
        q=1;
        for (col=1; col<=T.mu; ++col)
            for (p=1; p<=M.tu; ++p)
                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;
}
```

# 快速转置算法

- 方法：按a.data中三元组的次序进行转置，并将转置后的三元组置入b中恰当的位置。
- 建立辅助数组num和cpot，num[col]表示矩阵第col列中非零元的个数，cpot[col]指示第col列的第一个非零元素在b.data中的恰当位置。
- 按行扫描矩阵三元组表，根据某项的列号，确定它转置后的行号，查cpot表，按查到的位置直接将该项存入转置三元组表中。
- 转置时间复杂度为  $O(nu+tu+nu+tu)=O(tu)$ 。若矩阵有200列，10000个非零元素，总共需10000次处理。

<i>col</i>	1	2	3	4	5	6	7	语 义
<i>num</i> <i>[col]</i>	1	1	1	2	0	2	1	矩阵 <i>A</i> 各列非 零元素个数
<i>cpot</i> <i>[col]</i>	1	2	3	4	6	6	8	矩阵 <i>B</i> 各行开 始存放位置

- $cpot[1]=1$
- $cpot[col]=cpot[col-1]+num[col-1]$

# 稀疏矩阵的快速转置(算法5.2)

```
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;
      for (t=1;t<=M.tu;++t) ++num[M.data[t].j];
      //求M中每一列含非零元个数
      cpot[1]=1;
      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]; }
  }
  return OK;
```

# 用三元组表表示的稀疏矩阵及其转置

行 ( <i>row</i> )	列 ( <i>col</i> )	值 ( <i>value</i> )
1	4	22
1	7	15
2	2	11
2	6	17
3	4	-6
4	6	39
5	1	91
6	3	28

行 ( <i>row</i> )	列 ( <i>col</i> )	值 ( <i>value</i> )
1	5	91
2	2	11
3	6	28
4	1	22
4	3	-6
6	2	17
6	4	39
7	1	15

## 2、 十字链表

- 当矩阵中非零元素的个数和位置经过运算后变化较大时，就不宜采用顺序存储结构，而应采用链式存储结构来表示三元组。
- 稀疏矩阵的链接表示采用十字链表：行链表与列链表十字交叉。
- 行链表与列链表都是带表头结点的循环链表。用表头结点表征是第几行，第几列。



## • 元素结点

- right——指向同一行中下一个非零元素的指针（向右域）
- down——指向同一列中下一个非零元素的指针（向下域）

<b>row</b>	<b>col</b>	<b>val</b>
<b>down</b>		<b>right</b>

	<b>col=0</b>	<b>next</b>
		<b>right</b>

## • 表头结点

- 行表头结点
- 列表头结点
- next用于表示头结点的链接

<b>row=0</b>		<b>next</b>
<b>down</b>		

- 由于行、列表头结点互相不冲突，所以可以合并起来：

<b>row=0</b>	<b>col=0</b>	<b>next</b>
<b>down</b>		<b>right</b>

- 总表头结点：

<b>row</b>	<b>col</b>	<b>next</b>

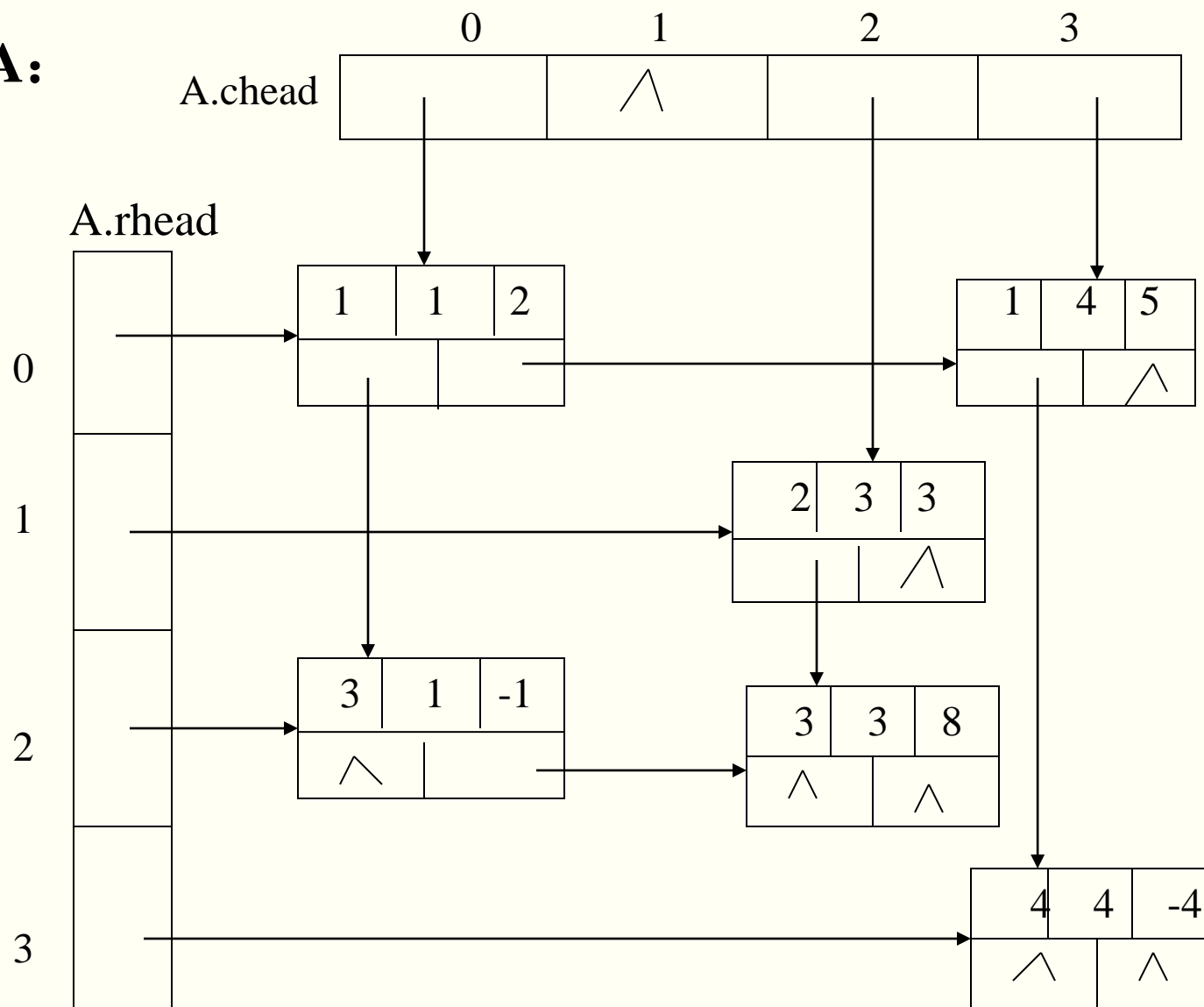
- 需要辅助结点作链表的表头，同时每个结点要增加两个指针域，所以只有在矩阵较大和较稀疏时才能起到节省空间的效果。
- 分别用两个一维数组存储行链表的头指针和列链表的头指针，可加快访问速度。

# 十字链表的类型定义

```
typedef struct OListNode{ //元素结点
    int i,j; //非零元的行和列下标
    ElemType e;
    struct OListNode *right,*down;
    //该非零元所在行表和列表的后继链域
} OListNode, *OLink;
typedef struct {
    OLink *rhead,*chead;
    //行和列链表头指针数组
    int mu,nu,tu;
} CrossList;
```

例：稀疏矩阵A：

$$\begin{pmatrix} 2 & 0 & 0 & 5 \\ 0 & 0 & 3 & 0 \\ -1 & 0 & 8 & 0 \\ 0 & 0 & 0 & -4 \end{pmatrix}$$



稀疏矩阵A的十字链表

# 十字链表的建立 (算法5.4)

\* 自学

## 小 结

- 1 矩阵压缩存储是指为多个值相同的元素分配一个存储空间，对零元素不分配存储空间；
- 2 特殊矩阵的压缩存储是根据元素的分布规律，确定元素的存储位置与元素在矩阵中的位置的对应关系；
- 3 稀疏矩阵的压缩存储除了要保存非零元素的值外，还要保存非零元素在矩阵中的位置。

## 5.4 广义表的定义

- \* **广义表**（Generalized Lists）是线性表的推广，又简称**表**（Lists）。广义表是 $n(n \geq 0)$ 个元素 $a_1, a_2, a_3, \dots, a_n$ 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

$$LS = (a_1, a_2, a_3, \dots, a_n)$$

LS是广义表的名字， $n$ 为它的长度。若 $a_i$ 是广义表，则称它为LS的子表。



- \* 在一个非空的广义表中，其元素 $a_i$ 可以是某一确定类型的单个元素，称为**原子**，也可以又是一个广义表，称为**子表**。因此广义表的定义是一种递归的定义，广义表是一种递归的数据结构。
- \* 当广义表非空的时候，称第一个元素 $a_1$ 为广义表A的**表头**（Head），称其余元素组成的**表**（ $a_2, a_3, \dots, a_n$ ）是A的**表尾**（Tail）。

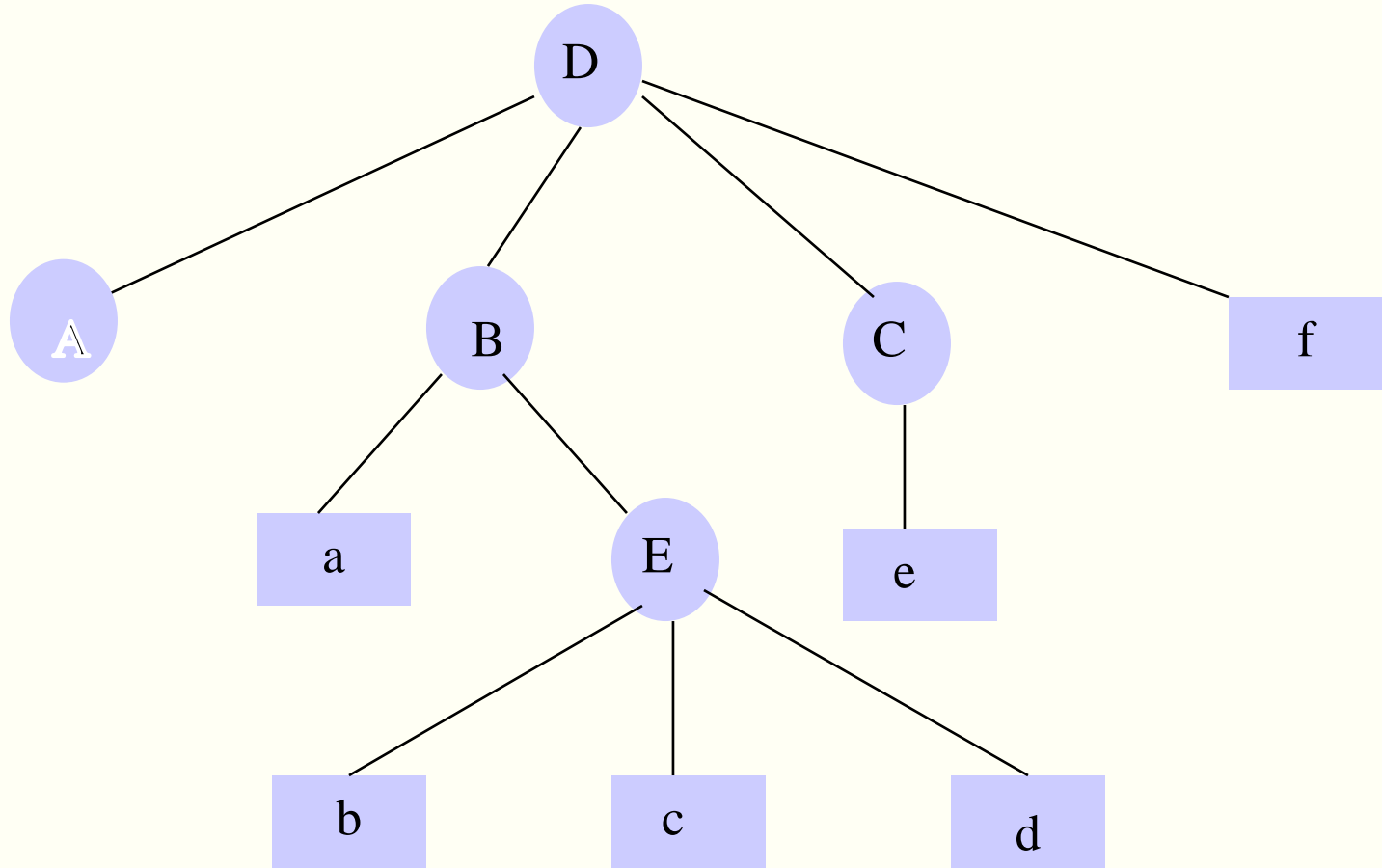
举例：

- (1)  $A = ()$  是一个空表，其长度为零。
- (2)  $B = (a, (b, c, d))$  的长度为2，两个元素分别是原子a和子表(b,c,d)，表头是元素a，表尾是子表((b,c,d))。
- (3)  $C = (e)$  只有一个原子e，C的长度为1,表头是元素e，表尾是空表  $()$ 。
- (4)  $D = (A, B, C, f)$  的长度为4，前三个元素都是广义表,第四个元素是原子f，表头是子表A，表尾是子表(B, C, f)。
- (5)  $E = (b, c, d)$  的长度为3，三个元素都是原子，表头是元素b，表尾是子表 (c, d)。
- (6)  $F = (a, F)$  递归表，长度为2，它是一个无限的广义表。

- \* 广义表的元素之间除了存在次序关系外还存在层次关系，广义表中元素最大的层数称为广义表的深度。相对于元素来说，它的层数就是包含该元素括号对的数目。例如广义表：

- \*  $G = (a, (b, (c, (d))))$

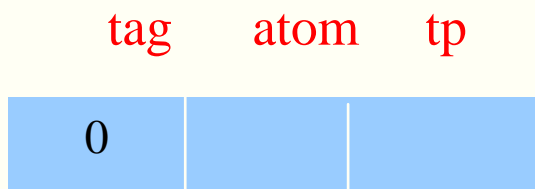
则数据元素a在第一层，数据元素b在第二层，数据元素c在第三层，数据元素d在第四层，广义表G的深度为4。



## 5.5 广义表的存储结构

由于广义表中数据元素可以具有不同结构，故难以用顺序结构表示广义表。通常采用链表存储方式。

如何设定链表结点？广义表中的数据元素可能为单元素（原子）或子表，由此需要两种结点：一种是表结点，用以表示广义表；一种是单元素结点，用以表示单元素（原子）。



单元素结点



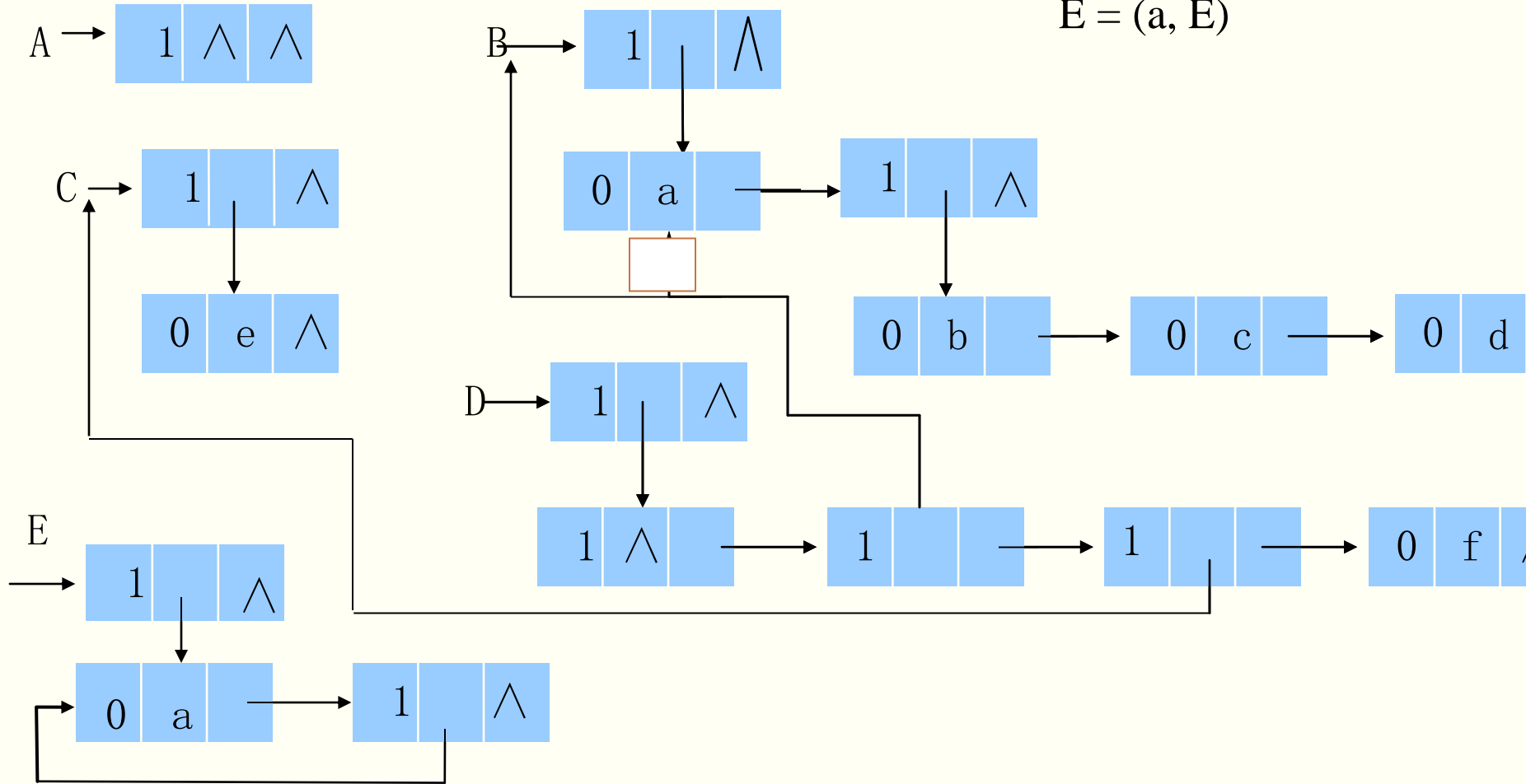
表结点

链表结点的类型定义如下：

```
struct GLNode  
{ int tag;          //标志域： 用于区分原子结点和表结点  
  union  
    { AtomType atom;          //原子结点的值域  
      struct GLNode *hp;    //表结点  
    };  
  struct GLNode *tp;    // 指向下一个  
} *GList
```

# 广义表的存储结构示例

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



# 本章学习要点

- 了解数组的两种表示方法，并掌握数组在以行为主的存储结构中的地址计算方法。
- 了解对特殊矩阵进行压缩存储时的下标变换公式。
- 了解稀疏矩阵的两种压缩存储方法的特点和适用范围，领会以三元组表示稀疏矩阵时进行矩阵运算采用的处理方法。
- 掌握广义表的定义及结构特点。