

数据结构与算法

第五章 数组和广义表

数组的定义

数组的顺序存储结构

矩阵的压缩存储

广义表的定义

广义表的存储结构

第一节数组的定义

数组可以看成是一种特殊的线性表，即线性表中数据元素本身也是一个线性表

■ 定义

$$A_{m \times n} = \begin{bmatrix} (a_{11}) & (a_{12}) & \dots & \dots & (a_{1n}) \\ (a_{21}) & a_{22} & \dots & \dots & a_{2n} \\ (\dots & \dots & \dots & \dots & \dots) \\ (a_{m1}) & (a_{m2}) & \dots & \dots & (a_{mn}) \end{bmatrix}$$

■ 数组特点

- ✧ 数组结构固定
- ✧ 数据元素同构

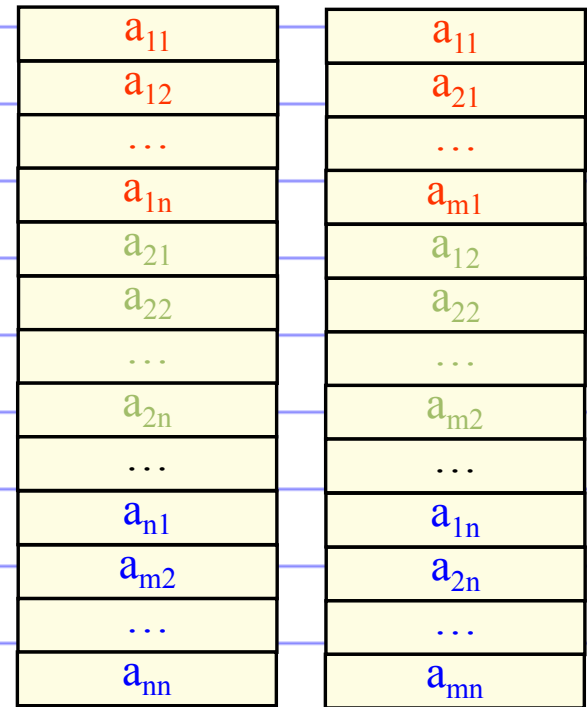
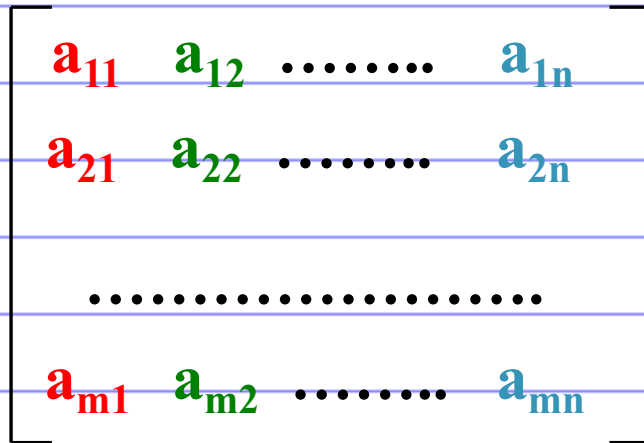
■ 数组运算

- ✧ 给定一组下标，存取相应的数据元素
- ✧ 给定一组下标，修改数据元素的值

第二节数组的顺序存储结构

□ 次序约定

- 以行序为主序
- 以列序为主序



列序: $Loc(a_{ij}) = Loc(a_{11}) + [(j-1)m + (i-1)] * l$

第三节 矩阵的压缩存储

- ☐ 对称矩阵
- ☐ 三角矩阵
- ☐ 对角矩阵
- ☐ 稀疏矩阵

稀疏矩阵的压缩存储方法

对称矩阵

$$\begin{bmatrix} \mathbf{a_{11}} & \mathbf{a_{12}} & \dots & \dots & \dots & \mathbf{a_{1n}} \\ \mathbf{a_{21}} & \mathbf{a_{22}} & \dots & \dots & \dots & \mathbf{a_{2n}} \\ & & \dots & & & \\ & & & & & \\ \mathbf{a_{n1}} & \mathbf{a_{n2}} & \dots & \dots & \dots & \mathbf{a_{nn}} \end{bmatrix}$$

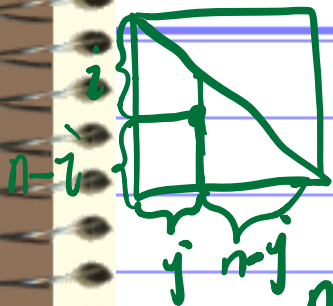
按行序为主序:

a ₁₁	a ₂₁	a ₂₂	a ₃₁	a ₃₂	a _{n1}	a _{nn}
k=0	1	2	3	4		n(n-1)/2		n(n+1)/2-1

$$k = \begin{cases} i(i-1)/2 + j - 1, & i \geq j \\ j(j-1)/2 + i - 1, & i < j \end{cases}$$

a_{11}
 a_{21} a_{22}
 a_{31} a_{32} a_{33}
 a_{41} a_{42} a_{43} a_{44}

若以列序为主序： 三角矩阵



$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + \frac{n(n-1)}{2} + (i-1)(n-j+1) + (j-1)$$

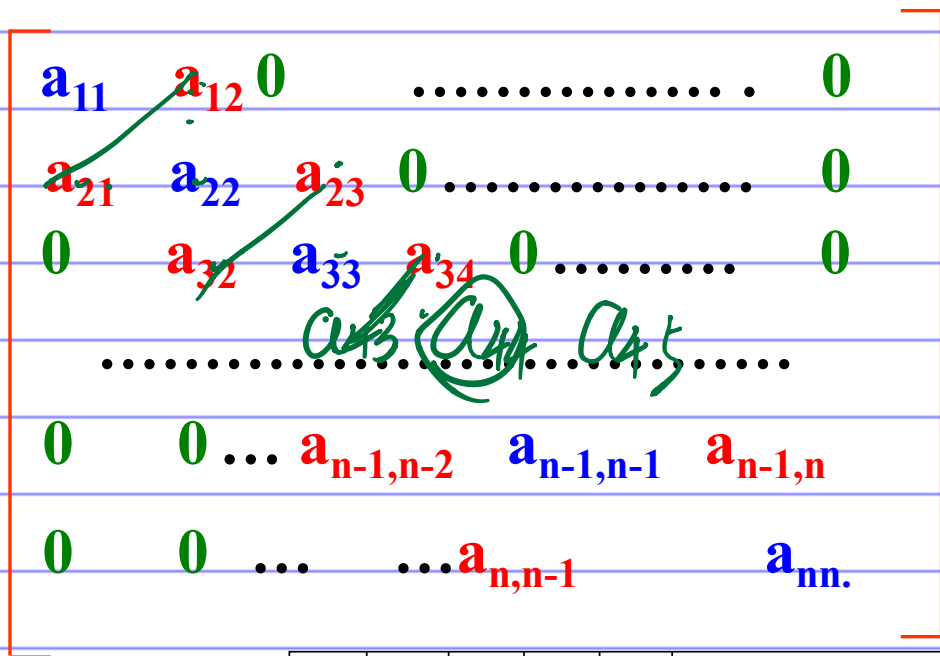
$$\begin{bmatrix} a_{11} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix}$$

按行序为主序：

a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{n1}	a_{nn}
$k=0$	1	2	3	4		$n(n-1)/2$		$n(n+1)/2-1$

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + [i(i-1)/2 + (j-1)] * 1$$

对角矩阵



按行序为主序：

a ₁₁	a ₁₂	a ₂₁	a ₂₂	a ₂₃	a _{n(n-1)}	a _{nn}
k=0	1	2	3	4		n(n-1)/2		n(n+1)/2-1

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + 2(i-1) + (j-1)$$

数据结构

前(i-1)行斜对角线 前1列相加

稀疏矩阵

- ✧ 定义：非零元较零元少，且分布没有一定规律的矩阵
- ✧ 压缩存储原则：只存矩阵的行列维数和每个非零元的行列下标及其值

$$M = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6 \times 7}$$

M由{(1,2,12), (1,3,9), (3,1,-3), (3,6,14), (4,3,24),
(5,2,18), (6,1,15), (6,4,-7)} 和矩阵维数 (6,7) 唯一确定

☆ 稀疏矩阵的压缩存储方法

☆ 三元组表

行列下标

非零元值

	i	j	v
0	6	7	8
1	1	2	12
2	1	3	9
3	1	3	9
4	3	1	-3
5	3	6	14
6	4	3	24
7	5	2	18
8	6	1	15
	6	4	-7

ma

$$M = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6 \times 7}$$

ma[0].i,ma[0].j,ma[0].v分别存放矩阵
行列维数和非零元个数

三元组表所需存储单元个数为 $3(t+1)$
其中t为非零元个数

三元组顺序表

```
#define MAXSIZE 12500
typedef struct {
    int i, j;
    ElemType e;
} Triple;
typedef struct {
    Triple data[MAXSIZE+1]; // data[0] 未用
    int mu, nu, tu; // 矩阵的行数列数和非零元个数
}
```

带辅助行向量的二元组表

增加一个辅助数组NRA[m+1]，其物理意义是第i行第一个非零元在二元组表中的起始地址（m为行数）

显然有：

$$\begin{cases} \text{NRA}[1]=1 \\ \text{NRA}[i]=\text{NRA}[i-1]+\text{第}i-1\text{行非零元个数}(i\geq 2) \end{cases}$$

列下标和
非零元值

NRA[0]不用或
存矩阵行数

NRA		j		v	
0	6	7	8	0	1
1	1	2	12	1	2
2	3	3	9	2	3
3	3	1	-3	3	4
4	5	6	14	4	5
5	6	3	24	5	6
6	7	2	18	6	7
		1	15	7	8
		4	-7	8	

矩阵列数和
非零元个数

二元组表需存储单元
个数为 $2(t+1)+m+1$

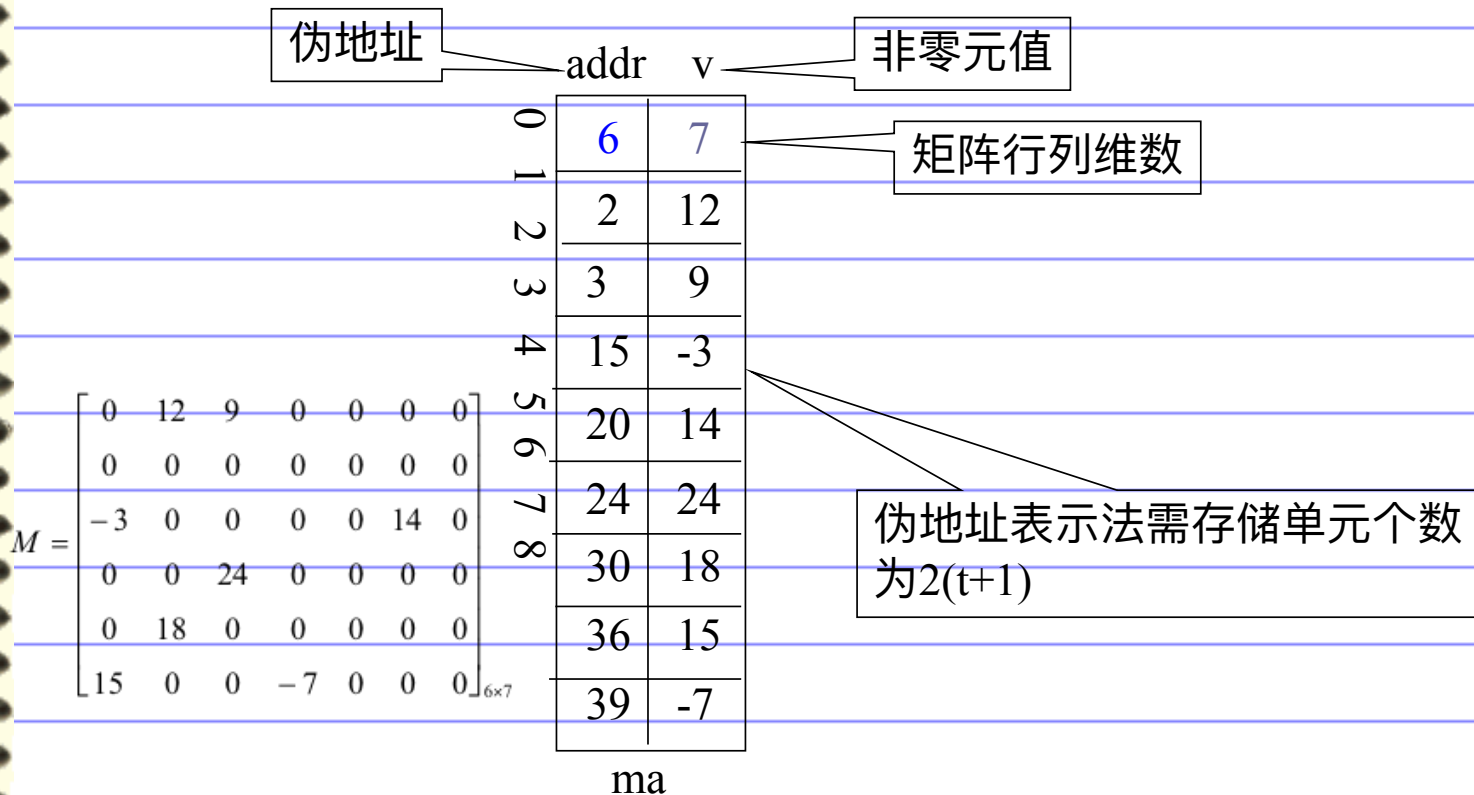
数据结构

ma

$$M = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6 \times 7}$$

伪地址表示法

伪地址：本元素在矩阵中（包括零元素在内）
按行优先顺序的相对位置



求转置矩阵

❓ 问题描述：已知一个稀疏矩阵的三元组表，求该矩阵转置矩阵的三元组表

❓ 问题分析

一般矩阵转置算法：

```
for(col=0;col<n;col++)  
    for(row=0;row<m;row++)  
        n[col][row]=m[row][col];  
T(n)=O(m×n)
```

$$M = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6 \times 7}$$

$$N = \begin{bmatrix} 0 & 0 & -3 & 0 & 0 & 15 \\ 12 & 0 & 0 & 0 & 18 & 0 \\ 9 & 0 & 0 & 24 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -7 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}_{7 \times 6}$$

	i	j	v
0	6	7	8
1	1	2	12
2	1	3	9
3	3	1	-3
4	3	6	14
5	4	3	24
6	5	2	18
7	6	1	15
8	6	4	-7

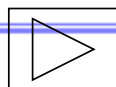
0	7	6	8
1	1	3	-3
2	1	6	15
3	2	1	12
4	2	5	18
5	3	1	9
6	3	4	24
7	4	6	-7
8	6	3	14

ma

mb

?

数据结构



❓解决思路：只要做到

①将矩阵行、列维数互换

②将每个三元组中的i和j相互调换

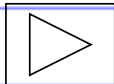
③重排三元组次序，使mb中元素以N的行(M的列)为主序

方法一：按M的列序转置

即按mb中三元组次序依次在ma中找到相应的三元组进行转置。

为找到M中每一列所有非零元素，需对其三元组表ma从第一行起扫描一遍。由于ma中以M行序为主序，所以由此得到的恰是mb中应有的顺序。

❓算法描述：



❓算法分析： $T(n) = O(M \text{的列数} n \times \text{非零元个数} t)$

❓若 t 与 $m \times n$ 同数量级，则 $T(n) = O(m \times n^2)$

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


	i	j	v	
0	6	7	8	
p → 1	1	2	12	← p
p → 2				
p → 3	1	3	9	← p
p → 4	3	1	-3	← p
p → 5	3	6	14	← p
p → 6				
p → 7	4	3	24	← p
p → 8	5	2	18	← p
p →	6	1	15	← p
p →	6	4	-7	← p

ma

col=1

	i	j	v	
0	7	6	8	
k → 1	1	3	-3	
k → 2				
k → 3	1	6	15	
k → 4	2	1	12	
k → 5	2	5	18	
k → 6				
k → 7	3	1	9	
k → 8	3	4	24	
	4	6	-7	
	6	3	14	

mb

col=2



方法二：快速转置

即按ma中三元组次序转置，转置结果放入b中恰当位置。

此法关键是要预先确定M中每一列第一个非零元在mb中位置，为确定这些位置，转置前应先求得M的每一列中非零元个数

实现：设两个数组

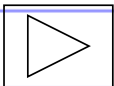
num[col]：表示矩阵M中第col列中非零元个数

cpot[col]：指示M中第col列第一个非零元在mb中位置

显然有：
$$\begin{cases} \text{cpot}[1]=1; \\ \text{cpot}[\text{col}]=\text{cpot}[\text{col}-1]+\text{num}[\text{col}-1]; \end{cases} \quad (2 \leq \text{col} \leq \text{ma}[0].j)$$

$$M = \begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}_{6 \times 7}$$

col	1	2	3	4	5	6	7
num[col]	2	2	2	1	0	1	0
cpot[col]	1	3	5	7	8	8	9

? 算法描述: 

? 算法分析: $T(n) = O(M \text{的列数} n + \text{非零元个数} t)$
若 t 与 $m \times n$ 同数量级, 则 $T(n) = O(m \times n)$

```

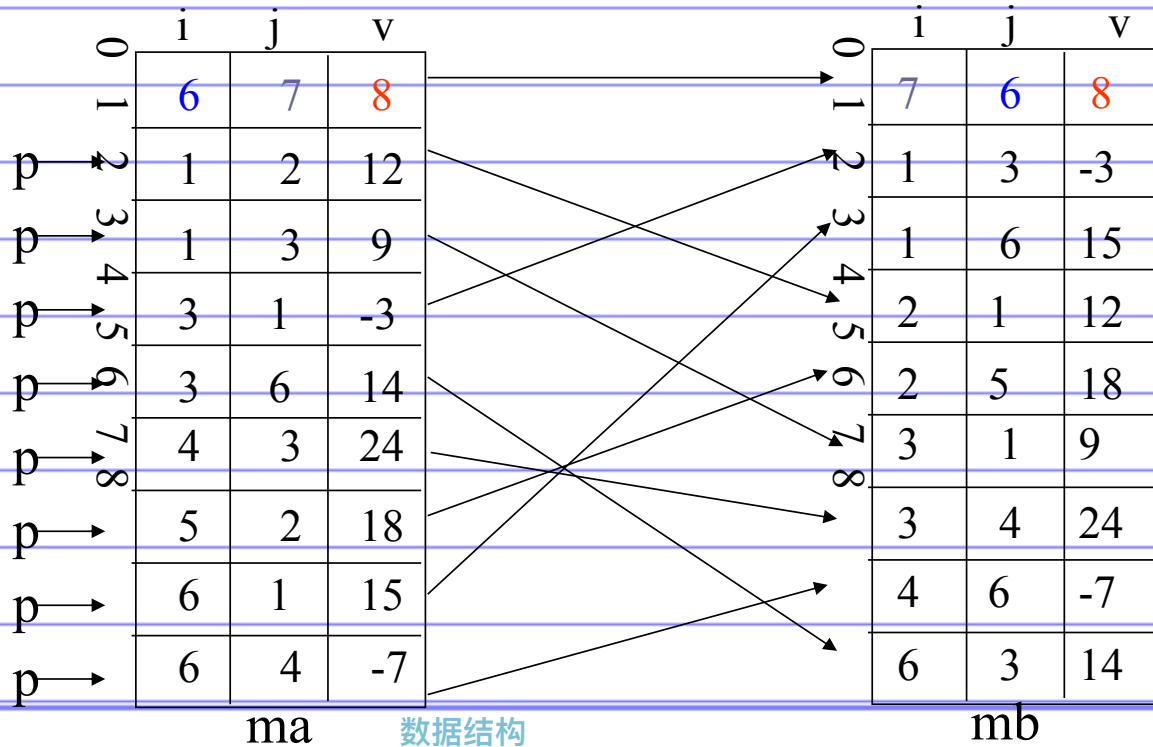
Status FastTransposeSMatrix(TSMatrix M, TSMatrix &T) {
    T.mu = M.nu; T.nu = M.mu; T.tu = M.tu;
    if(T.tu) {
        for(col=1; col<=M.nu; ++col) num[col]=0;
        for(t=1; t<=M.tu, ++t) ++num[M.data[t].j];
        cpot[1] = 1;
        for(col=2; col<=M.nu; ++col) cpot[col] = cpot[col-1] + num[col-1];
        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;
}

```

col	1	2	3	4	5	6	7
num[col]	2	2	2	1	0	1	0
cpot[col]	1	3	5	7	8	8	9

2 4 6 9

3 5 7



链式存储结构

□ 带行指针向量的单链表表示

❓ 每行的非零元用一个单链表存放

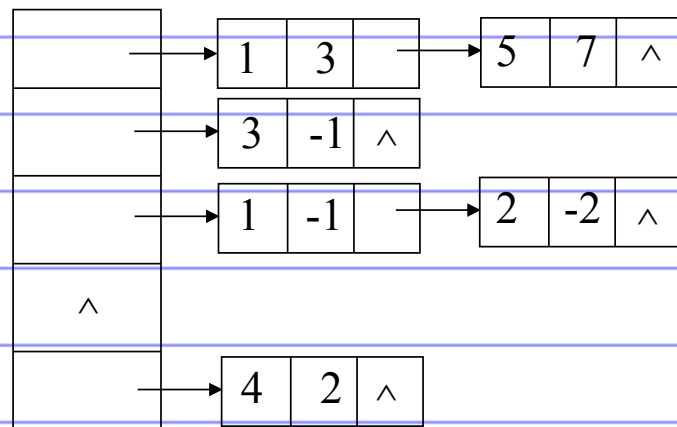
❓ 设置一个行指针数组，指向本行第一个非零元结点；若本行无非零元，则指针为空

```
typedef struct node
```

```
{   int col;
    int val;
    struct node *link;
}JD;
```

```
typedef struct node *TD;
```

$$A = \begin{bmatrix} 3 & 0 & 0 & 0 & 7 \\ 0 & 0 & -1 & 0 & 0 \\ -1 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$



需存储单元个数为 $3t+m$

十字链表

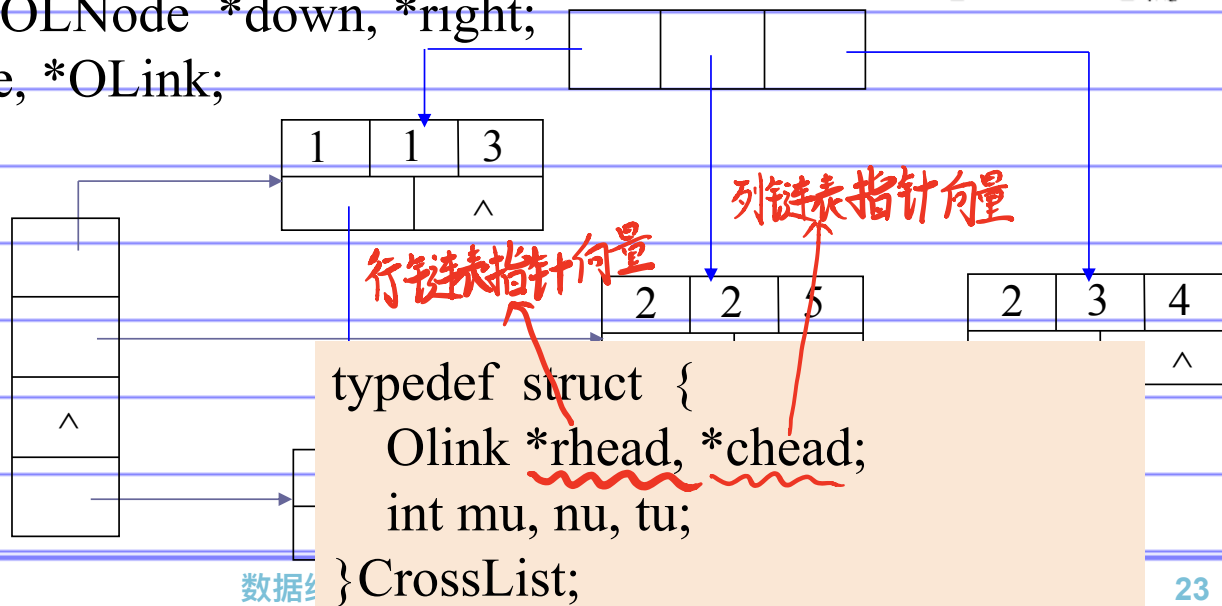
❓ 设行指针数组和列指针数组，分别指向每行、列第一个非零元

❓ 结点定义

```
typedef struct OLNode {
    int i, j;
    ElemType e;
    struct OLNode *down, *right;
} OLNode, *OLink;
```

row	col	val
	down	right

$$A = \begin{bmatrix} 3 & 0 & 0 \\ 0 & 5 & 4 \\ 0 & 0 & 0 \\ 8 & 0 & 0 \end{bmatrix}_{4 \times 3}$$



```
typedef struct {
    Olink *rhead, *chead;
    int mu, nu, tu;
} CrossList;
```

? 从键盘接收信息建立十字链表算法

见书P104：算法5.4

任意的非零元输入先后次序

注意：行表和列表的插入操作

? 算法分析： $T(n) = O(t \times s)$

其中：t—非零元个数

$$s = \max(m, n)$$

$m=4, n=3$

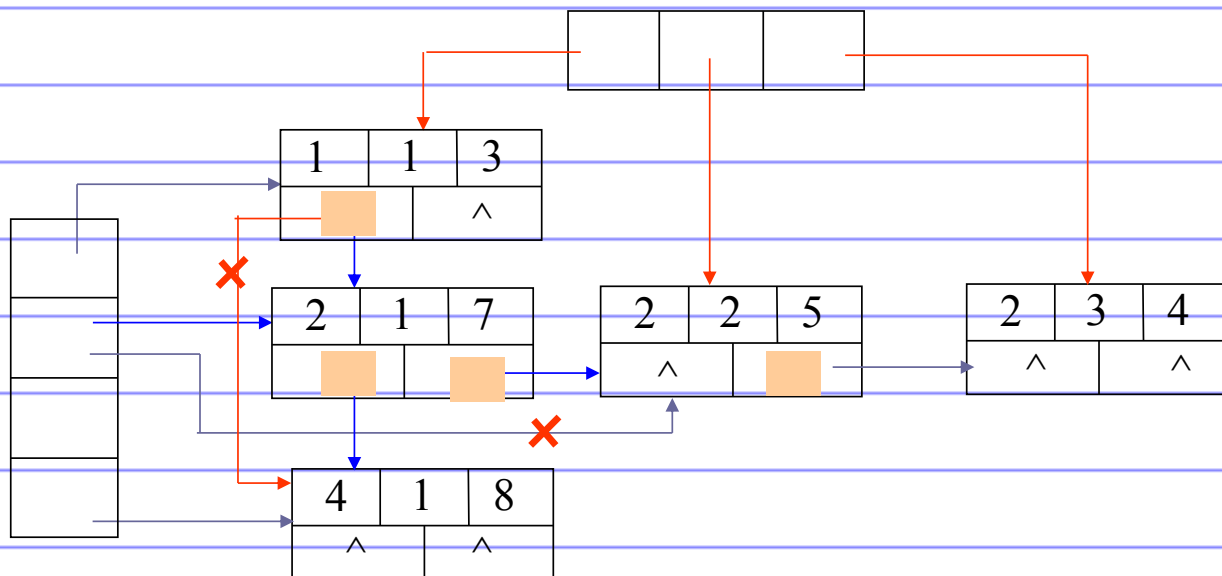
1,1,3

2,2,5

2,3,4

4,1,8

2,1,7



第四节 广义表的定义

- 顾名思义，广义表是线性表的推广，也有人称之为列表（Lists用复数形式以示与统称的表list的区别）。
- LISP语言，把广义表示为基本的数据结构，就连程序也表示为一系列的广义表。

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

其中,LS是广义表 (a_1, a_2, \dots, a_n) 的名称, n 是它的长度。

- 在线性表的定义中, a_i ($1 \leq i \leq n$)只限于是单个元素。而在广义表的定义中, a_i 可以是单个元素,也可以是广义表,分别称为广义表LS的原子和子表。
- 习惯上,用大写字母表示广义表的名称,用小写字母表示原子。当广义表LS非空时,称第一个元素 a_1 为LS的表头(Head),称其余元素组成的表为LS的表尾(Tail)。

广义表的定义

□ 广义表的定义是一个递归的定义，因为在描述广义表时又用到了广义表的概念。

(1) $A = ()$ —— A 是一个空表，它的长度为零。

(2) $B = (e)$ ——列表 B 只有一个原子 e ， B 的长度为 1。

(3) $C = (a, (b, c, d))$ ——列表 C 的长度为 2，两个元素分别为原子 a 和子表 (b, c, d) 。

(4) $D = (A, B, C)$ ——列表 D 的长度为 3，三个元素都是列表。显然，将子表的值代入后，则有
 $D = ((), (e), (a, (b, c, d)))$ 。

(5) $E = (a, E)$ ——这是一个递归的表，它的长度为 2。 E 相当于一个无限的列表 $E = (a, (a, (a, \dots)))$ 。

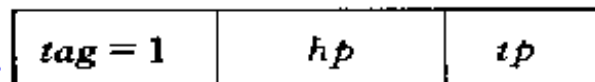
广义表的特征

- (1) 列表的元素可以是子表, 而子表的元素还可以是子表, ...。由此, 列表是一个多层次的结构, 可以用图形象地表示。
 - (2) 列表可为其它列表所共享。
 - (3) 列表可以是一个递归的表, 即列表也可以是其本身的一个子表。
- 根据前述对表头、表尾的定义可知: 任何一个非空列表其表头可能是原子, 也可能是列表. 而其表尾必定为列表。
 - 值得提醒的是列表()和(())不同。前者为空表. 长度 $n=0$; 后者长度 $n=1$, 可分解得到其表头、表尾均为空表()。

第五节 广义表的存储结构

□ 通常采用链式存储结构

□ 每个数据元素可用一个结点表示



表结点



原子结点

// --- 广义表的头尾链表存储表示 ---

```
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
```

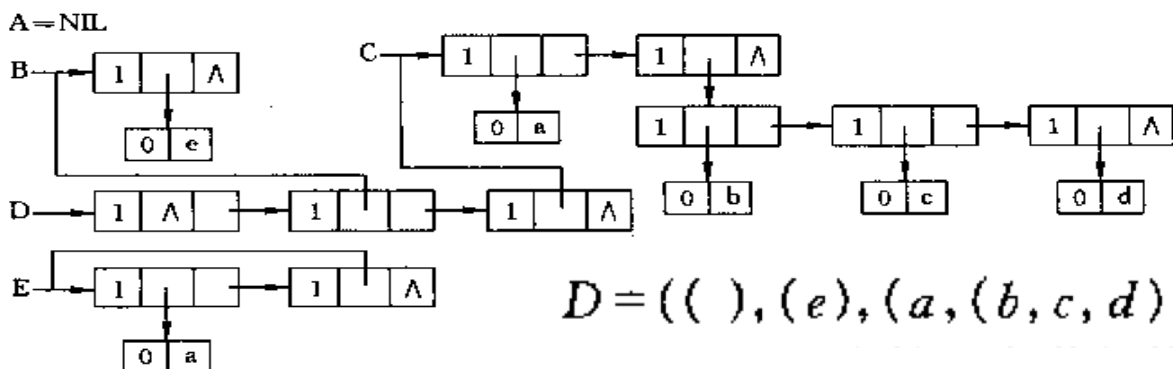
```
    };
```

```
}; // *GList; // 广义表类型
```

广义表的存储例1

□ 在这种存储结构中有几种情况：

- (1)除空表的表头指针为空外，对任何非空列表，其表头指针均指向一个表结点，且该结点中的hp域指示列表表头(或为原子结点，或为表结点)，tp域指向列表表尾(除非表尾为空，则指针为空，否则必为表结点)；
- (2)容易分清列表中原子和子表所在层次。
- (3)最高层的表结点个数即为列表的长度。



广义表的链式存储2

□ 扩展的线性表表示

tag = 1	hp	tp
---------	----	----

表结点

tag = 0	atom	tp
---------	------	----

原子结点

// --- 广义表的扩展线性链表存储表示 ---

```
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;  // 原子结点的值域
```

```
        struct GLNode * hp; // 表结点的表头指针
```

```
    };
```

```
    struct GLNode * tp; // 相当于线性链表的 next, 指向下一个元素结点
```

```
} * GList; // 广义表类型 GList 是一种扩展的线性链表
```

广义表的存储例2

