

数据结构与算法

主讲教师：刘峤

第5章 数组及广义表

第5章 内容提要

5.1 数组的定义

5.2 数组的顺序存储

5.3 矩阵的压缩存储

5.4 广义表 (**自学**)

数组

数组的定义和特点

- 定义：数组是一种特殊的线性表
 - 即：线性表中的数据元素本身也是一个线性表

数组特点

- 数组结构固定
- 数据元素同构

$$A_{m \times n} =$$

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}

以行序为主序

数组运算

- 给定一组下标，存取相应的数据元素

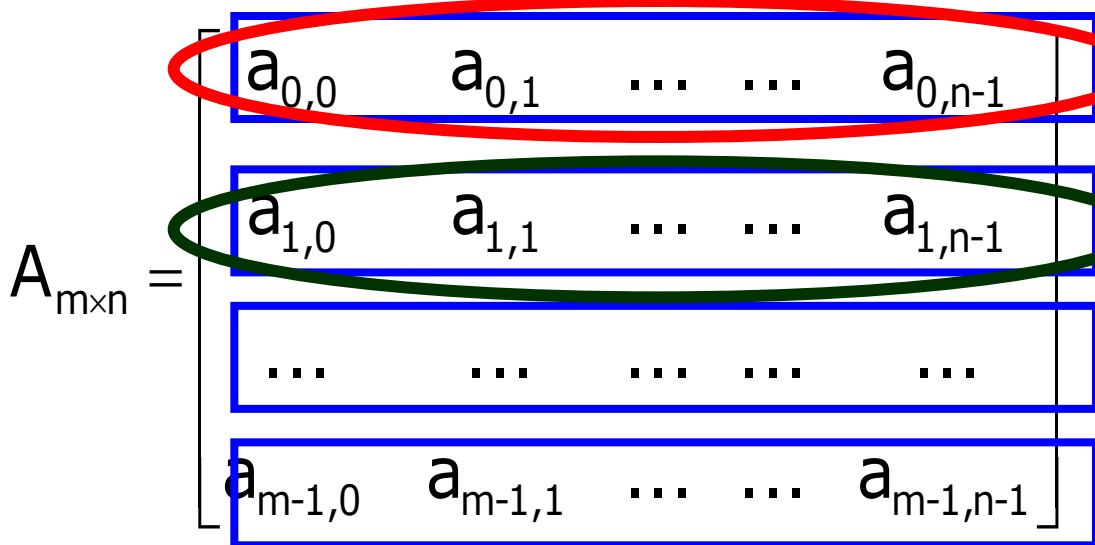
- 给定一组下标，修改数据元素的值..($a_{m1} a_{m2} \dots a_{mn}$)

\dots	\dots	\dots	\dots	\dots
a_{m1}	a_{m2}	\dots	\dots	a_{mn}

以列序为主序

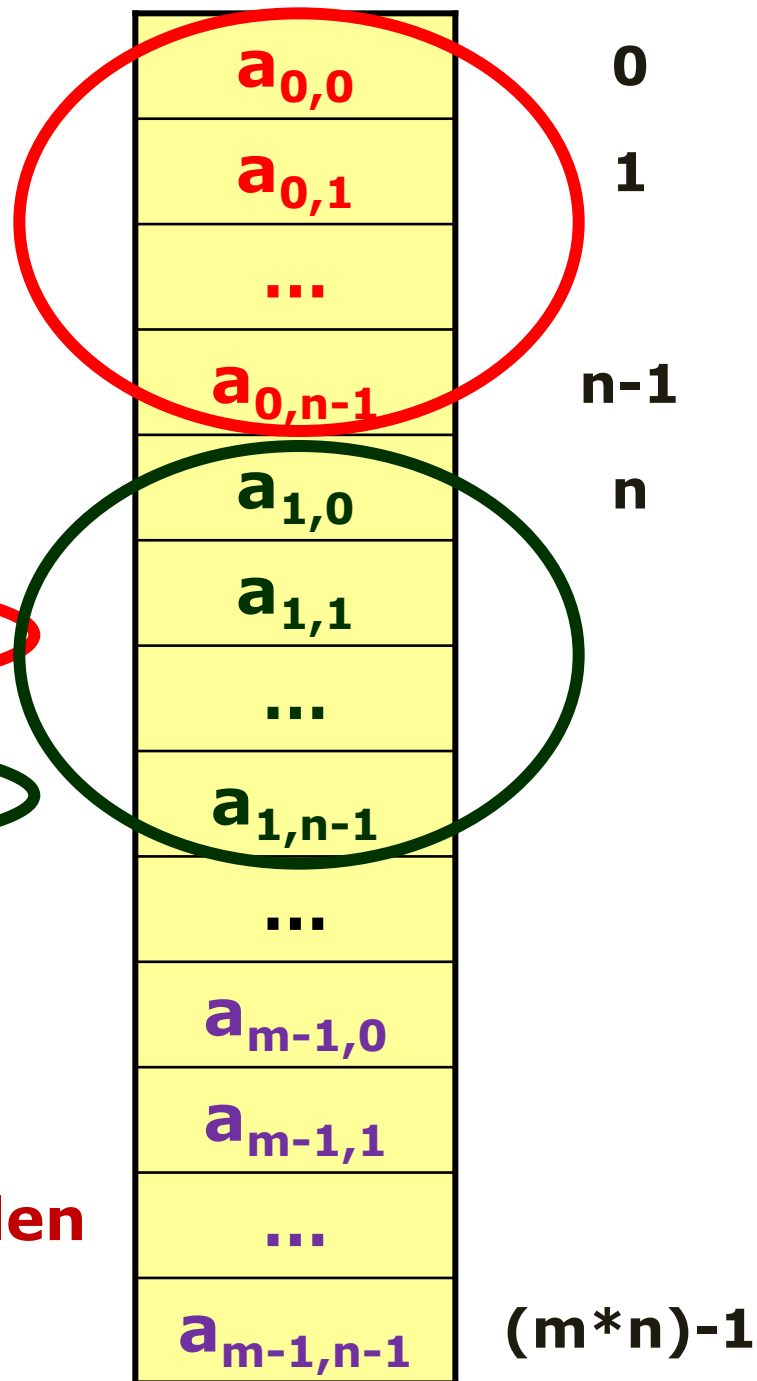
☞ 数组的顺序存储结构

- 次序约定
 - + 以行序为主序进行存储
 - + 以列序为主序进行存储



$$\text{Loc}(i, j) = \text{Loc}(0, 0) + (i \times n + j) \times \text{len}$$

$$\text{len} = \text{sizeof}(A[0][0])$$



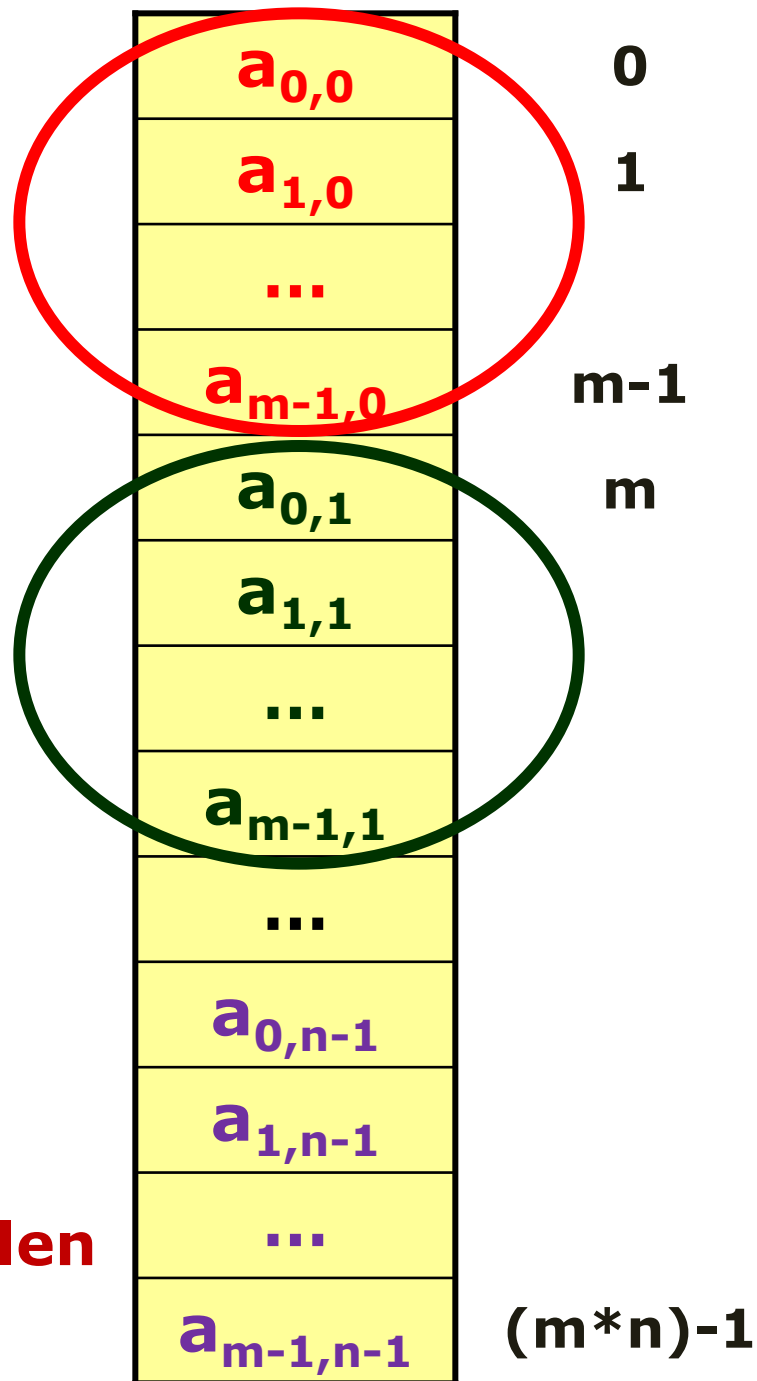
☞ 数组的顺序存储结构

- 次序约定
 - + 以行序为主序进行存储
 - + 以列序为主序进行存储



$$\text{Loc}(i, j) = \text{Loc}(0, 0) + (j \times m + i) \times \text{len}$$

$$\text{len} = \text{sizeof}(A[0][0])$$



数组的顺序存储结构

☞ 示例：设数组A[20][30]的基地址为1000

- 设：每个元素占2个存储单元
- 若以行为主序顺序存储，计算元素A[5, 6]的存储地址

$$\text{Loc}(i, j) = \text{Loc}(0, 0) + (i \times n + j) \times \text{len}$$

$$= 1000 + (5 \times 30 + 6) \times 2 = 1312$$

- 若以列为主序顺序存储，计算元素A[7, 8]的存储地址

$$\text{Loc}(i, j) = \text{Loc}(0, 0) + (j \times m + i) \times \text{len}$$

$$= 1000 + (8 \times 20 + 7) \times 2 = 1334$$

数组的顺序存储结构

∞ 推广：对于n维数组（按行主序存储），有：

$$\begin{aligned} & \text{Loc}(a_1, a_2, \dots, a_n) \\ &= \text{Loc}(0, \dots, 0) + (b_2 \times \dots \times b_n \times a_1 + \\ & \quad b_3 \times \dots \times b_n \times a_2 + \dots + b_n \times a_{n-1} + a_n) \times \text{len} \\ &= \text{Loc}(0, \dots, 0) + \left(\sum_{i=1}^{n-1} a_i \prod_{k=i+1}^n b_k + a_n \right) \times \text{len} \end{aligned}$$

其中： b_i 为数组第 i 维的下标

矩阵的压缩存储

对称矩阵: $a_{ik} = a_{ki}$

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

稀疏矩阵: 除个别外多数元素为零

$$\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

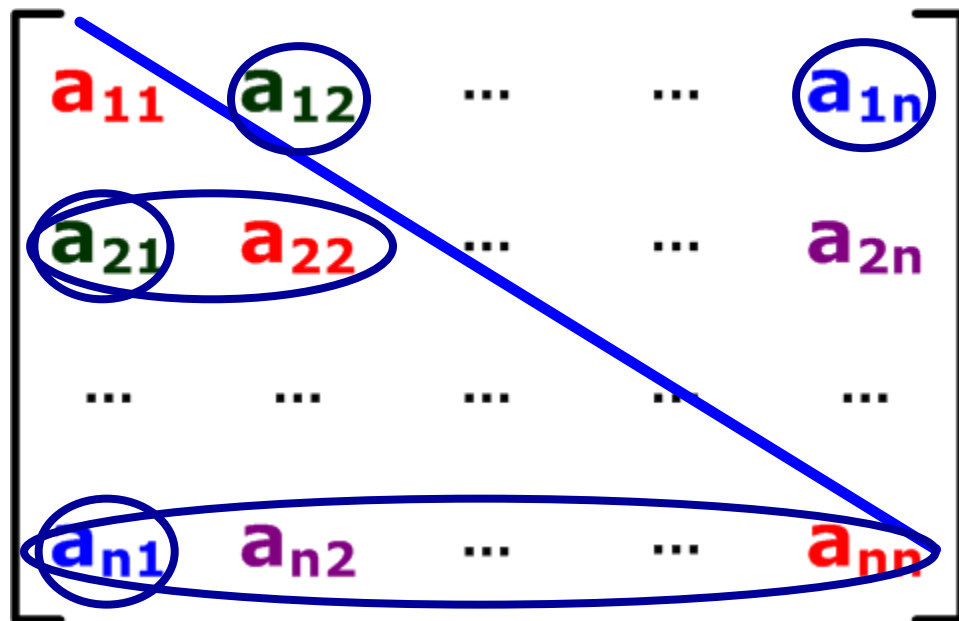
三角矩阵: 右上角或左下角元素为零

$$\begin{bmatrix} \mathbf{a_{11}} & 0 & 0 & \cdots & 0 \\ \mathbf{a_{21}} & \mathbf{a_{22}} & 0 & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & 0 \\ \mathbf{a_{n1}} & \mathbf{a_{n2}} & \mathbf{a_{n3}} & \cdots & \mathbf{a_{nn}} \end{bmatrix}$$

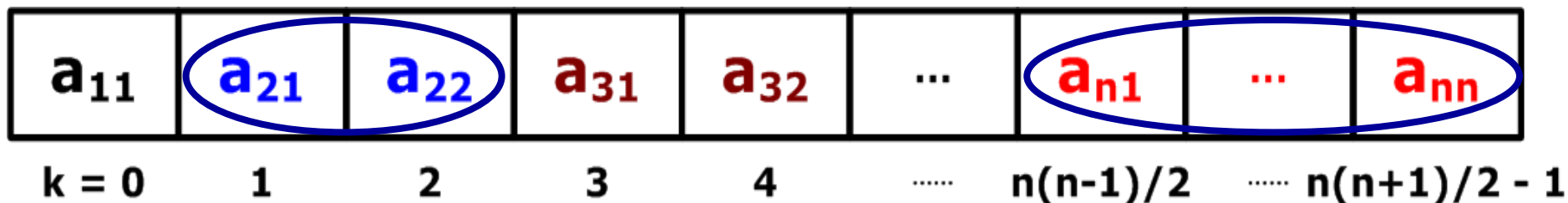
对角矩阵: 除对角元素外其余元素为零

$$\begin{bmatrix} \mathbf{a_{11}} & \mathbf{a_{12}} & 0 & 0 & 0 & \cdots & \cdots & 0 \\ \mathbf{a_{21}} & \mathbf{a_{22}} & \mathbf{a_{23}} & 0 & 0 & \cdots & \cdots & 0 \\ 0 & \mathbf{a_{32}} & \mathbf{a_{33}} & \mathbf{a_{34}} & 0 & \cdots & \cdots & 0 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ 0 & 0 & 0 & \cdots & \mathbf{a_{n-1,n-2}} & \mathbf{a_{n-1,n-1}} & \mathbf{a_{n-1,n}} \\ 0 & 0 & 0 & 0 & \cdots & \mathbf{a_{n,n-1}} & \mathbf{a_{nn}} \end{bmatrix}$$

矩阵的压缩存储：对称矩阵



按行主序存储：只需要存储下三角矩阵的元素值即可



怎样确定下标？ $k = i(i-1)/2 + j - 1$

矩阵的压缩存储：三角矩阵

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

按行主序存储：与对称矩阵类似

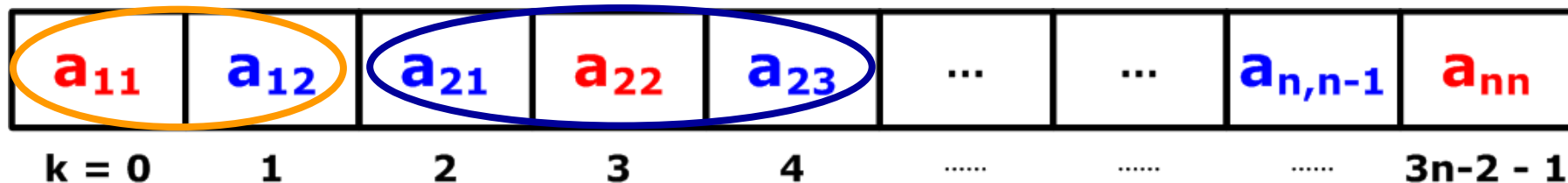
$\mathbf{a_{11}}$	$\mathbf{a_{21}}$	$\mathbf{a_{22}}$	$\mathbf{a_{31}}$	$\mathbf{a_{32}}$	\dots	$\mathbf{a_{n1}}$	\dots	$\mathbf{a_{nn}}$
$k = 0$	1	2	3	4	$n(n-1)/2$	$n(n+1)/2 - 1$

怎样确定下标？ $k = i(i-1)/2 + j - 1$

矩阵的压缩存储：对角矩阵

$$a_{10} \begin{bmatrix} a_{11} & a_{12} & 0 & 0 & 0 & \dots & \dots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & \dots & \dots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & \dots & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & 0 & 0 & \dots & a_{n,n-1} & a_{nn} \end{bmatrix}$$

按行主序存储



怎样确定下标？ $k = 2 \times i + j - 3$

矩阵的压缩存储：稀疏矩阵

$$\begin{bmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{bmatrix}$$

- 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：只存矩阵的行列维数以及非零元的行列下标和值
- 稀疏矩阵特点：矩阵由非零元素集合和矩阵维数唯一确定
 - 非零元素集合： $\{(1,2,12), (1,3,9), (3,1,-3), (3,6,14), (4,3,24), (5,2,18), (6,1,15), (6,4,-7)\}$
 - 矩阵维数： $(6, 7)$

矩阵的压缩存储：稀疏矩阵

采用顺序存储结构：三元组表

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

行列下标 **nrow** **ncol** **val** 非零元值

0	6	7	8
1	矩阵维数		元素个数
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

ma

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

$3(n+1)$

其中n为矩阵中的非零元个数

矩阵的压缩存储：稀疏矩阵

采用顺序存储结构：三元组表

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

行列下标 nrow ncol val 非零元值

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

```
#define MAXSIZE 20
typedef struct {
    int nrow, ncol;
    ElemType val;
}Triplet;
typedef struct{
    Triplet * T;
    int m, n, ne;
}TMatrix;
```

矩阵的压缩存储示例：求稀疏矩阵的转置

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

- 问题：已知稀疏矩阵的三元组表，求转置矩阵的三元组表
- 分析：首先考虑一般情况 **思考：如果是三元组表怎么办？**
 - 已知维度为 $m \times n$ 的矩阵A，求其转置矩阵B

```
for(row=0; row<m; ++row)           算法复杂度?  
    for(col=0; col<n; ++col)         T(n)=O(m×n)  
        B[col][row]=A[row][col];
```


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

$$B = A'$$

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

怎样实现?

	i	j	d
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

ma

	i	j	d
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

mb

矩阵的压缩存储示例：求稀疏矩阵的转置

编程思路

- 将矩阵行、列维数互换
 - 处理ma的第一行
- 将每个三元组中的 i 和 j 相互调换
- 对三元组重新排序
 - 使mb中元素按行主序方式排列

	i	j	d
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

ma

矩阵的压缩存储示例：求稀疏矩阵的转置

∞ 方法一：按ma的列序实现矩阵转置

- 根据mb的行主序：依次在ma中查找相应的三元组
 - 为找出ma每一列所有非零元素，需对其进行一次全局扫描
 - 由于ma按行主序存储，由此得到的mb也是按行主序存储
- 算法复杂度：设 $\mathbf{ma}_{m \times n}$ 中的非零元素个数为 \mathbf{e}

$$\mathbf{T(n)=O(n \times e)}$$

- 若 \mathbf{e} 与 $\mathbf{m \times n}$ 同数量级，则

$$T(n) = O(m \times n^2)$$

求稀疏矩阵的转置：常规方法

```
void transpose(TMatrix *ma, TMatrix *mb){
    mb->m = ma->n; mb->n = ma->m; mb->e = ma->e;
    int s, i, k = 0;
    for( s = 0; s < ma->n; ++s ){
        for( i = 0; i < (ma->e); ++i ){
            if(ma->T[i].ncol == s){
                mb->T[k].nrow = ma->T[i].ncol;
                mb->T[k].ncol = ma->T[i].nrow;
                mb->T[k].val = ma->T[i].val ;
                k++;
            }
        }
    }
}
```

思考：如何提高转置效率？

矩阵的压缩存储示例：求稀疏矩阵的转置

方法二：快速转置

- 从ma中顺序读取元素，转置后放入mb中恰当位置
- 关键：预先确定A中每一列第一个非零元在mb中位置
 - 转置前应首先求得矩阵A的每一列中非零元个数
- 实现：设置两个辅助数组
 - num[c]：存放矩阵A的第c列中的非零元个数
 - offset[c]：A中第c列第一个非零元在mb中的位置

$$\begin{cases} \text{offset}[1] = 1; \\ \text{offset}[c] = \text{offset}[c-1] + \text{num}[c-1]; \end{cases}$$

$$(2 \leq c \leq \text{ma}[0].n)$$

求稀疏矩阵的转置：快速转置

$A =$

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

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

从ma中顺序读取元素，转置后放入mb中恰当位置

- num[col]: 存放矩阵A的第col列中的非零元个数
- offset[col]: 指示A中第col列第一个非零元在mb中的位置

算法分析

- $T(n) = O(ma.n + ma.e)$
- 若 $ma.e$ 与 $m \times n$ 同数量级: $T(n) = O(m \times n)$

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

