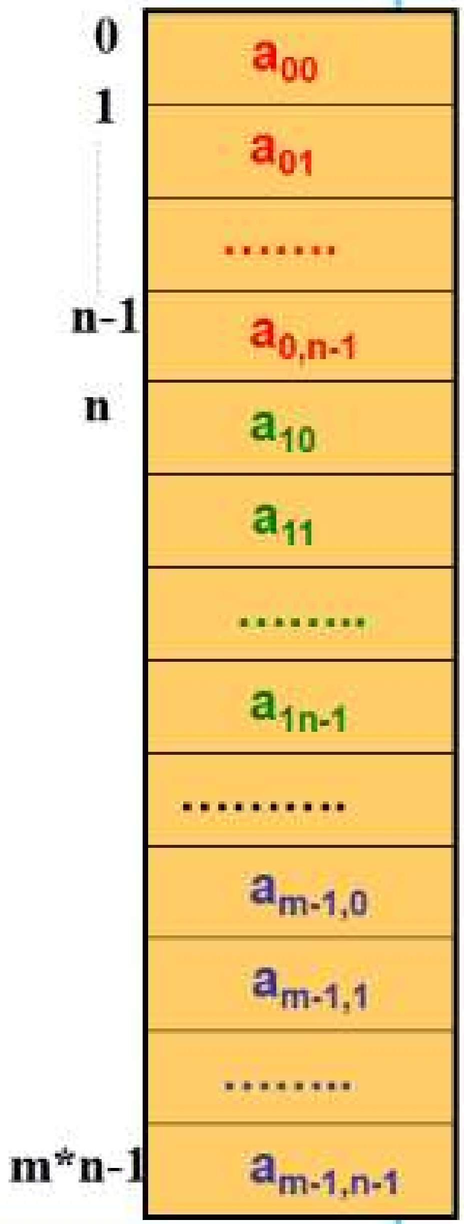
按行序为主序存放

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



按列序为主序存放

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

0	a_{00}
1	a_{10}
	……..
m-1	$a_{m-1,0}$
m	a_{01}
	a_{11}
	………
	$a_{m-1,1}$
	…………
	$a_{0,n-1}$
	$a_{1,n-1}$
	………
m*n-1	$a_{m-1,n-1}$

数组

如何建立二维数组行列下标与一维数组下标的关系？

17

数组

设一般的二维数组是A[0..b_1−1, 0..b_2−1]

$$A_{mn}=\begin{bmatrix} a_{00} & \cdots & \cdots & \cdots & a_{0,b2-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{i0} & \cdots & a_{ij} & \cdots & a_{i,b2-1} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{b1-1,0} & \cdots & \cdots & \cdots & a_{b1-1,b2-1} \end{bmatrix}$$

则行优先存储时的地址公式为:

$LOC(a_{ij})=LOC(a_{00})+(b_2*i+j)*L$

二维数组列优先存储的通式为:

$LOC(a_{ij})=LOC(a_{00})+(b_1*j+i)*L$

18

数组

在科学与工程计算问题中，矩阵是一种常用的数学对象，在高级语言编制程序时，将一个矩阵描述为一个二维数组。矩阵在这种存储表示之下，可以对其元素进行随机存取，各种矩阵运算也非常简单。

19

数组

什么是稀疏矩阵？

- 设矩阵A中有s个非零元素，若s远远小于矩阵元素的总数（即$s \ll m \times n$），则称A为稀疏矩阵。
- 有s个非零元素。令$e=s/(m \times n)$，称e为矩阵的稀疏因子。通常认为$e \leq 0.05$时为稀疏矩阵。

20

数组

稀疏矩阵的压缩存储方法

[三元组顺序表]

[行逻辑联接的顺序表]

[十字链表]

21

数组

三元组顺序表

- 存储稀疏矩阵时，为了节省存储单元，使用压缩存储方法。
- 非零元素的分布一般是没有规律的，因此在存储非零元素的同时，还必须同时记下它所在的行和列的位置（i, j）。
- 一个三元组(i, j, a_{ij})唯一确定了矩阵A的一个非零元。因此，稀疏矩阵可由表示非零元的三元组及其行列数唯一确定。

22

例如，下列稀疏矩阵

$$
M=\begin{array}{c} 1\\2\\3\\4\\5\\6\\ \\ \end{array}
\begin{array}{c}
\begin{bmatrix}
0 & 12 & 9 & 0 & 0 & 0 & 0\\
0 & 0 & 0 & 0 & 0 & 0 & 0\\
-3 & 0 & 0 & 0 & 0 & 14 & 0\\
0 & 0 & 24 & 0 & 0 & 0 & 0\\
0 & 18 & 0 & 0 & 0 & 0 & 0\\
15 & 0 & 0 & -7 & 0 & 0 & 0
\end{bmatrix}_{6\times 7}\\
\begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array}
\end{array}
$$

可由三元组表

((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

数组

```
#define  MAXSIZE  12500
typedef struct {
   int  i, j;      //该非零元的行下标和列下标
   ElemType  e;    // 该非零元的值
} Triple;  // 三元组类型
```

```
typedef struct {
   Triple  data[MAXSIZE+1];
   int  mu, nu, tu; (mu行数,nu列数,tu非零元个数)
} TSMatrix;  // 稀疏矩阵类型
```

24

数组

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

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

注意：三元组表中的元素按行（或列）排列。

稀疏矩阵压缩存储的缺点：将失去随机存取功能

25

求转置矩阵算法

$$M=\begin{bmatrix}0 & 14 & 0 & 0 & -5\\ 0 & -7 & 0 & 0 & 0\\ 36 & 0 & 0 & 28 & 0\end{bmatrix} \Rightarrow T=\begin{bmatrix}0 & 0 & 36\\ 14 & -7 & 0\\ 0 & 0 & 0\\ 0 & 0 & 28\\ -5 & 0 & 0\end{bmatrix}$$

用常规的二维数组表示时的算法

```
for (c=1; c<=nu; ++c)
    for (r=1; r<=mu; ++r)
        T[c][r] = M[r][c];
```

其时间复杂度为: O(mu×nu)

26

求转置矩阵算法

$$M=\begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 \end{pmatrix} \xrightarrow{\text{转置后}} T=\begin{pmatrix} 0 & 0 & -3 & 0 & 0 & 15 \\ 12 & 0 & 0 & 0 & 18 & 0 \\ 9 & 0 & 0 & 24 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -7 \\ 0 & 0 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

三元组表 M.data	?	三元组表 T.data
(1, 2, 12)		(1, 3, -3)
(1, 3, 9)		(1, 6, 15)
(3, 1, -3)		(2, 1, 12)
(3, 5, 14)		(2, 5, 18)
(4, 3, 24)		(3, 1, 9)
(5, 2, 18)		(3, 4, 24)
(6, 1, 15)		(4, 6, -7)
(6, 4, -7)		(5, 3, 14)

27

投票 最多可选1项

若采用三元组压缩技术存储稀疏矩阵，只要把每个元素的行下标和列下标互换，就完成了对该矩阵的转置运算，这种说法正确吗

A 正确

B 不正确

求转置矩阵算法

不正确！

（1）每个元素的行下标和列下标互换（即三元组中的i和j互换）；

（2）T的总行数mu和总列数nu与M值不同（互换）；

（3）重排三元组内元素顺序，使转置后的三元组也按行（或列）为主序有规律的排列。

上述（1）和（2）容易实现，难点在（3）。

有两种实现方法：
- 压缩转置
- (压缩)快速转置

方法1：压缩转置

求转置矩阵算法

思路：反复扫描M.data中的列序，从小到大依次进行转置。

M

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

T

	i	j	e
0	7	6	8
q→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

col=1

求转置矩阵算法

```
Status TransPoseSMatrix(TSMatrix M, TSMatrix &T)
{//用三元组表存放稀疏矩阵M，求M的转置矩阵T
 （1） T.mu=M.nu;  T.nu=M.mu;  T.tu=M.tu;
//nu是列数,mu是行数,tu是非零元素个数
 （1） if (T.tu) {
 （2）  q=1;     //q是转置矩阵T的结点编号
 （3） for(col=1; col<=M.nu; col++)
 （4）  {for(p=1; p<=M.tu; p++)   //p是M三元表中结点编号
 （5）     {if (M.data[p].j==col)
 （6）        {T.data[q].i=M.data[p].j;  T.data[q].j=M.data[p].i;
 （7）          T.data[q].e=M.data[p].e;  q++;
 （8）        }
 （9）       }
 （10）    }
  }
   return OK;
} //TranposeSMatrix
```

求转置矩阵算法

压缩转置算法的效率分析

1、主要时间消耗在查找M.data[p].j=col的元素，由两重循环完成：

```
for(col=1; col<=M.nu; col++)   循环次数＝nu
    for(p=1; p<=M.tu; p++)     循环次数＝tu
```

所以该算法的时间复杂度为O(nu*tu)

——即M的列数与M中非零元素的个数之积

最坏情况：M中全是非零元素，此时tu=mu*nu，

时间复杂度为 $O(nu^2*mu)$

注：若M中基本上是非零元素时，即使用非压缩传统转置算法的时间复杂度也不过是O(nu*mu)

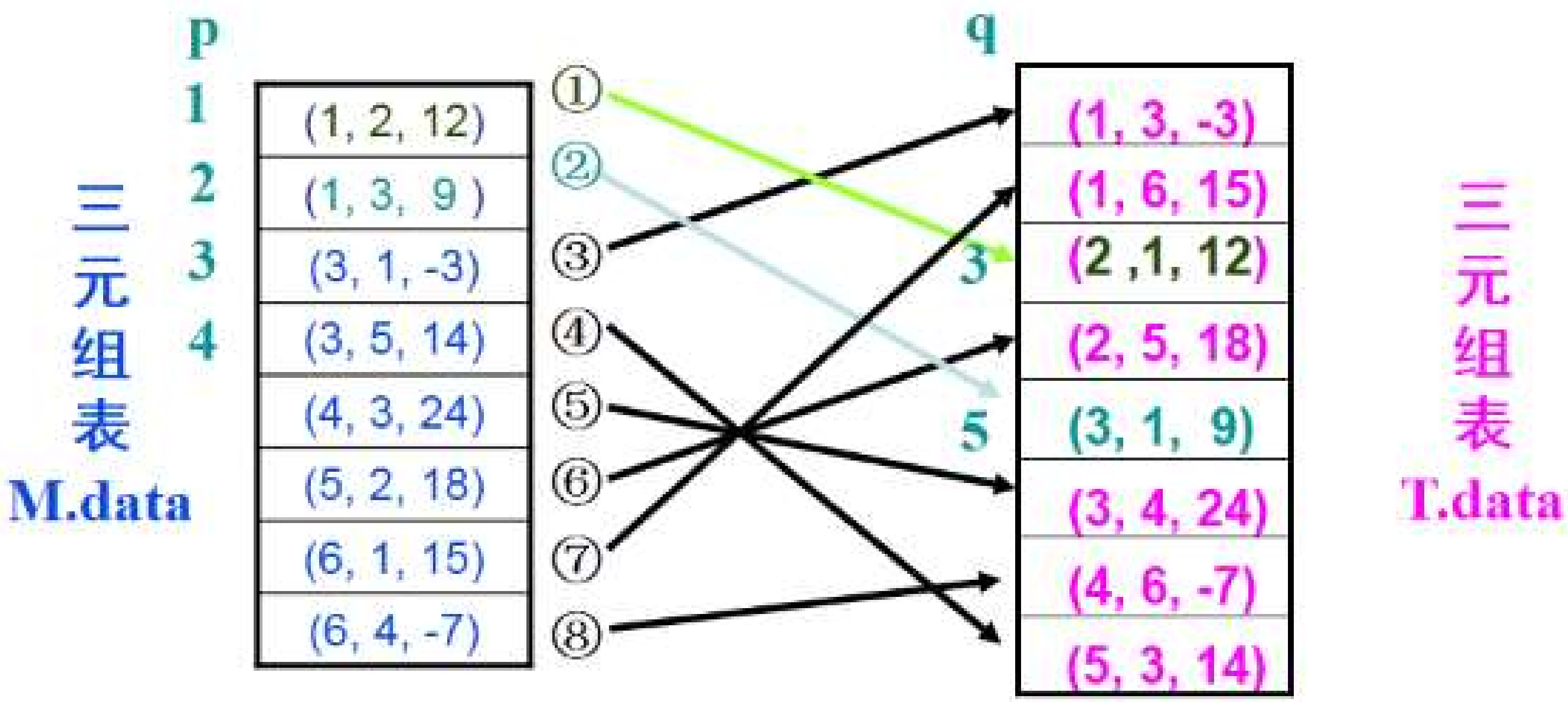
结论：压缩转置算法不能滥用。

前提：仅适用于非零元素个数很少（即tu<<mu*nu）的情况。

32

快速求转置矩阵算法

思路：依次把M. data中的元素直接送入T. data的恰当位置上（即M三元组的p指针不回溯）。



关键：怎样寻找T. data的“恰当”位置？

33

快速求转置矩阵算法

设计思路：

如果能预知M矩阵中每一列(即T的每一行)的非零元素个数，又能预知每一列第一个非零元素在T.data中的位置，则扫描M.data时便可以将每个元素准确定位。

快速求转置矩阵算法

令：M中的列变量用col表示；

num[col]： 存放M中第col 列中非0元素个数，

cpot[col]： 存放M中第col列的第一个非0元素的位置，

（即T.data中待计算的"恰当"位置所需参考点）

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

$$
\begin{array}{cc}
\text{col} & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
M= & \begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 \end{pmatrix}
\end{array}
$$

规律： $cpot(1)=1$

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

35

转置矩阵算法

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

M

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

T

	i	j	v
0	6	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	5	3	14

36

快速求转置矩阵算法

```
Status FastTransposeSMatrix（TSMatirx M, TSMatirx &T）
{
  （1）T.mu = M.nu ;T.nu = M.mu ; T.tu = M.tu ;
  （2）if (T.tu) {
  （3）for(col = 1; col <=M.nu; col++)  num[col] =0;
  （4） for( i = 1; i <=M.tu; i ++) {col =M.data[ i ] .j ; ++num [col] ;}
  （5） cpot[ 1 ] =1;
  （6）for(col = 2; col <=M.nu; col++)  cpot[col]=cpot[col-1]+num[col-1] ;
  （7） for( p =1; p <=M.tu ; p ++ )
  （8）{  col =M.data[ p ]. j ; q =cpot [ col ];
  （9）    T.data[q].i = M.data[p]. j;
  （10）    T.data[q].j = M.data[p]. i;
  （11）T.data[q]. e = M.data[p].e;
   （12） ++cpot[col] ;
   （13）  } //for
  （14）} //if
   （15）return OK;
} //FastTranposeSMatrix;
```