§5. 数组和广义表

5.1. 数组的定义

ADT Array {　　//P.90

数据对象：

$j_i=0,...,b_i-1,\ i=1,2,...,n,$

$D=\{a_{j1j2...jn}$ | n(>0)称为数组的维数，

b_i是数组第i维的长度

j_i是数组元素第i维的下标

$a_{j1j2...jn}\in ElemSet\}$

数据关系：

$R = \{R_1,\ R_2,\ ...,\ R_n\}$

$R_i = \{\langle a_{j1...ji...jn},\ a_{j1...ji+1...jn}\rangle$ |

$0\leqslant j_k\leqslant b_k-1,\ 1\leqslant k\leqslant n$ 且$k\neq i$,

$0\leqslant j_i\leqslant b_i-2,$

$a_{j1...ji...jn},\ a_{j1...ji+1...jn}\in D,\ i=2,...n\}$

基本操作：

...

}

具体到C/C++语言：　int a[10];

元素　　　：a[j1]
n=1　　　：维数
b1=10　　：第1维长度
j1∈[0..9]：第1维下标
a[j1]∈int

R = { R_1 }
R1={ <a[j1],a[j1+1]> |
0≤j1≤b1-2,　[0..8]
a[j1],a[j1+1]∈D
}

具体到C/C++语言：　int a[10][15];

元素　　　　　　　　　：a[j1][j2]
n=2　　　　　　　　　：维数
b1=10/b2=15　　　　　：第1/2维长度
j1∈[0..9]/j2∈[0..14]：第1/2维下标
a[j1][j2]∈int

R ={ R_1 , R_2 }
R1={ <a[j1][j2],a[j1+1][j2]> | 0≤j1≤b1-2,　[0..8]
0≤j2≤b2-1,　[0..14]
a[j1][j2],a[j1+1][j2]∈D }
R2={ <a[j1][j2],a[j1][j2+1]> | 0≤j1≤b1-1,　[0..9]
0≤j2≤b2-2,　[0..13]
a[j1][j2],a[j1][j2+1]∈D }

具体到C/C++语言：　int a[10][15][20];

元素　　　　　　　　　　　　：a[j1][j2][j3]
n=3　　　　　　　　　　　　：维数
b1=10/b2=15/b3=20　　　　　：第1/2/3维长度
j1∈[0..9]/j2∈[0..14]/j3∈[0..19]：
第1/2/3维下标
a[j1][j2][j3]∈int

R ={ R_1 , R_2 , R_3 }
R1={ <a[j1][j2][j3],a[j1+1][j2][j3]> | 0≤j1≤b1-2,　[0..8]
0≤j2≤b2-1,　[0..14]
0≤j3≤b3-1,　[0..19]
a[j1][j2][j3],a[j1+1][j2][j3]∈D }
R2={ <a[j1][j2][j3],a[j1][j2+1][j3]> | 0≤j1≤b1-1,　[0..9]
0≤j2≤b2-2,　[0..13]
0≤j3≤b3-1,　[0..19]
a[j1][j2][j3],a[j1][j2+1][j3]∈D }
R3={ <a[j1][j2][j3],a[j1][j2][j3+1]> | 0≤j1≤b1-1,　[0..9]
0≤j2≤b2-1,　[0..14]
0≤j3≤b3-2,　[0..18]
a[j1][j2][j3],a[j1][j2][j3+1]∈D }

§5. 数组和广义表

5.1.数组的定义

★ 称为n维数组，每一组的维数可各不相同

★ 结构确定后不能进行动态修改结构的操作(维数和维界不能改变)

★ 共有 b1*b2*...*bn个元素 ($\prod_{i=1}^{n} \boldsymbol{b_i}$)

★ 数组中的元素必须同构，元素类型可以是基本数据类型，也可以是构造数据类型

★ 受n个关系约束，每一个元素在n个关系上分别有直接前驱(第1个除外)和直接后继(最后一个元素除外),n个关系均为线性结构

★ n维数组也可以看做是元素是n-1维数组的一维数组

一维数组：A={a_0， a_1， ...，a_{n-1} }

二维数组：A=

行向量形式

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

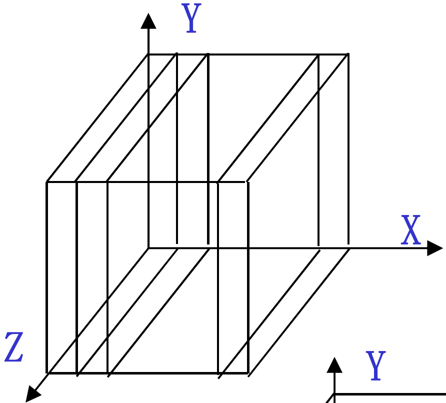
二维数组：A=

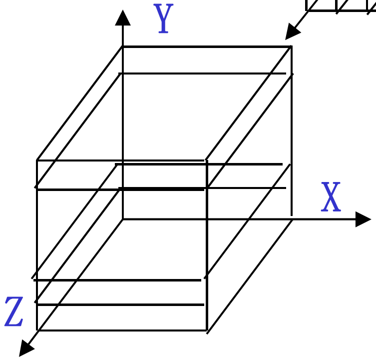
列向量形式

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

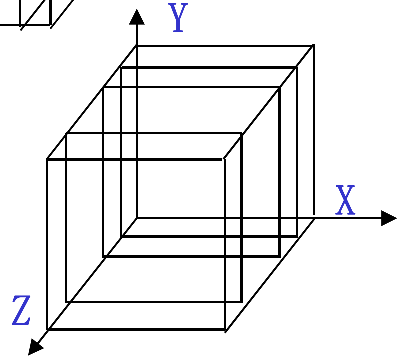
三维数组：

X轴向量形式





Y轴向量形式



Z轴向量形式

§5. 数组和广义表

5.2. 数组的顺序表示和实现

5.2.1. 一维数组的顺序存储

★ 用一组连续的存储单元来存放数据元素，逻辑/物理结构对应

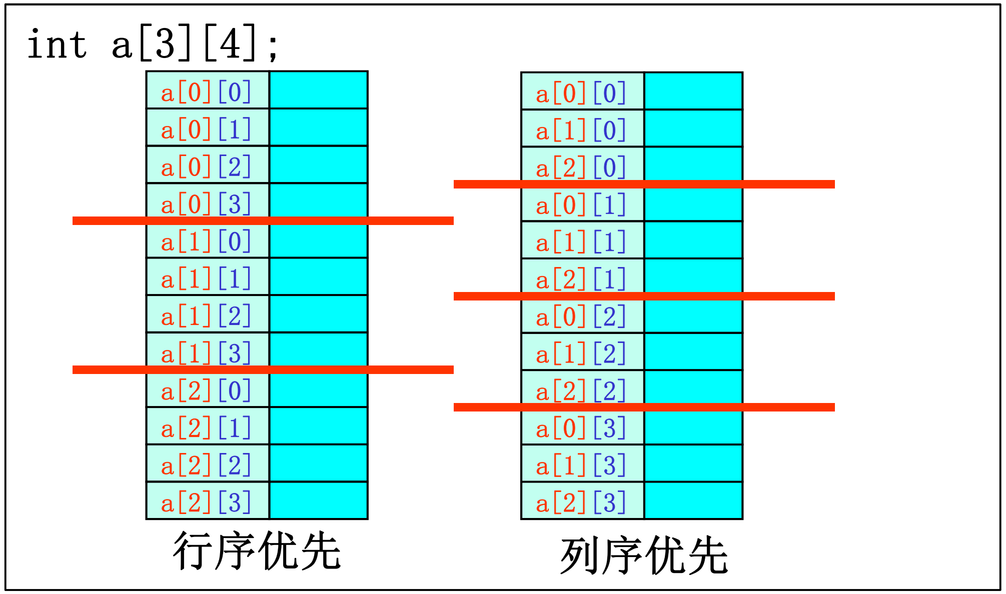
★ 一维数组第i个元素的存储位置

Loc(i) = Loc(0)+i*sizeof(数据元素类型)

5.2.2. 二维数组的顺序存储

★ 计算机的存储单元是一维结构，因此二维数组存放到存储单元中有两种方法

- 先行后列：以行序为主序，也称为行序优先
- 先列后行：以列序为主序，也称为列序优先



★ 二维数组第[i,j]个元素的存储位置(行序优先)

Loc(i,j)=Loc(0,0)+(i*b2+j)*sizeof(数据元素)

§5. 数组和广义表

5.2. 数组的顺序表示和实现

5.2.3. 多维数组的顺序存储

★ 两种存储方法

- 行序优先：先n维，次第n-1维，...，最后第1维
- 列序优先：先第1维，次第2维，...，最后第n维

```
int a[3][4][5];
```

```
a[0][0][0], a[0][0][1], ... , a[0][0][4],
a[0][1][0],             ...,  a[0][1][4],
...
a[0][3][0],             ...,  a[0][3][4],
a[1][0][0],             ...,  a[1][0][4],
...                     ...,  a[2][3][4]
```

行序优先

```
a[0][0][0], a[1][0][0], a[2][0][0],
a[0][1][0], a[1][1][0], a[2][1][0],
a[0][2][0], a[1][2][0], a[2][1][0],
...
a[0][3][4], a[1][3][4], a[2][3][4]
```

列序优先

★ 多维数组的元素存储位置

Loc(j1, j2, ..., jn) = Loc(0, 0, ... 0) +
(b2*b3*...*bn*j1 + b3*b4*...*bn*j2
+ ... + bn*jn-1+jn)*sizeof(数据元素)

= Loc(0, 0, ... 0) + $(\sum_{i=1}^{n-1} j_i \prod_{k=i+1}^{n} b_k$ + jn)*sizeof(数据元素)

设二维数组A[6][10],每个数组元素占用4个存储单元，若按行优先顺序存放,a[0][0]的地址是860，则a[3][5]的地址是___A___

A 1000　　860+(10*3+5)*4 = 1000

B 860

C 1140

D 1200

§5.数组和广义表

5.3.矩阵的压缩存储

5.3.1.特殊矩阵的压缩存储

★ 特殊矩阵的含义

值相同的元素/零元素在矩阵中的分布有一定规律

★ 压缩存储的含义

为多个值相同的元素只分配一个存储空间，对零元素不分配空间

§5. 数组和广义表

5.3. 矩阵的压缩存储

5.3.1. 特殊矩阵的压缩存储

★ 对称矩阵

若矩阵A的元素满足以下性质

$a_{ij} = a_{ji} \qquad 0 \leqslant i, j \leqslant n-1$

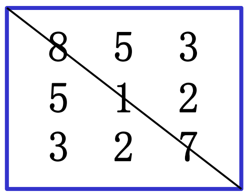
则称为n阶对称矩阵

- n*n的方阵
- 关于主对角线对称
- 压缩存储方法：每一对对称元素占一个空间

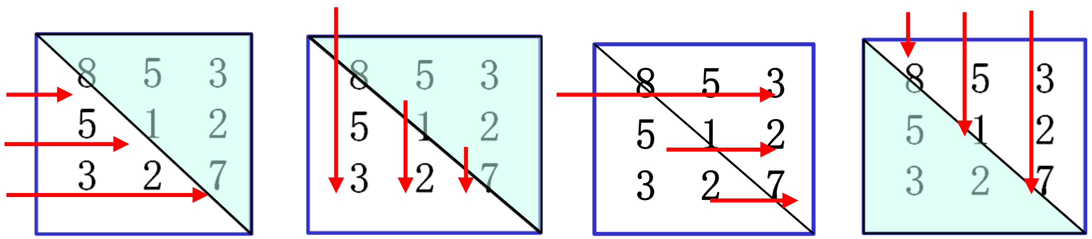
二维(n^2) => 一维($\frac{n(n+1)}{2}$)

a[n][n] => sa[$\frac{n(n+1)}{2}$]

[0..n-1][0..n-1] => [0..$\frac{n(n+1)}{2}$-1]

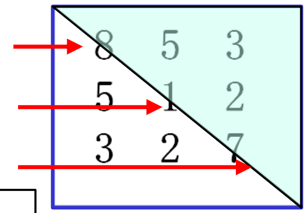


- 元素的排列顺序有多种方式



下三角 行序优先　下三角 列序优先　上三角 行序优先　上三角 列序优先

§5. 数组和广义表



下三角
行序优先

- 不失一般性，讨论下三角行序优先，则a[i][j] 和 sa[k]的对应关系为

[0..n-1][0..n-1] => [0.. $\frac{n(n+1)}{2}$-1]

一维/二维都从0开始，正常C/C++方式的数组表示方法

$$k=\begin{cases}\frac{i(i+1)}{2}+j & (i\geqslant j)\\ \frac{j(j+1)}{2}+i & (i<j)\end{cases}$$

a[0][0]		= sa[0]	= 8
a[1][0]	= a[0][1]	= sa[1]	= 5
a[1][1]	=	= sa[2]	= 1
a[2][0]	= a[0][2]	= sa[3]	= 3
a[2][1]	= a[1][2]	= sa[4]	= 2
a[2][2]		= sa[5]	= 7

[1..n][1..n] => [0.. $\frac{n(n+1)}{2}$-1]

二维从1开始，一维从0开始，书 P.95 公式5-3对应的是这种情况!!!

$$k=\begin{cases}\frac{i(i-1)}{2}+j-1 & (i\geqslant j)\\ \frac{j(j-1)}{2}+i-1 & (i<j)\end{cases}$$

a[1][1]		= sa[0]	= 8
a[2][1]	= a[1][2]	= sa[1]	= 5
a[2][2]	=	= sa[2]	= 1
a[3][1]	= a[1][3]	= sa[3]	= 3
a[3][2]	= a[2][3]	= sa[4]	= 2
a[3][3]		= sa[5]	= 7

[1..n][1..n] => [1.. $\frac{n(n+1)}{2}$]

一维/二维数组都从1开始（某些参考书）

$$k=\begin{cases}\frac{i(i-1)}{2}+j & (i\geqslant j)\\ \frac{j(j-1)}{2}+i & (i<j)\end{cases}$$

a[1][1]		= sa[1]	= 8
a[2][1]	= a[1][2]	= sa[2]	= 5
a[2][2]	=	= sa[3]	= 1
a[3][1]	= a[1][3]	= sa[4]	= 3
a[3][2]	= a[2][3]	= sa[5]	= 2
a[3][3]		= sa[6]	= 7

§5. 数组和广义表

5.3. 矩阵的压缩存储

5.3.1. 特殊矩阵的压缩存储

★ 三角矩阵

n阶矩阵，其上(下)三角(含对角线)存储任意数据，而下(上)三角部分为零或常数c

$$\begin{cases}\text{上三角矩阵：上三角为任意数据(含对角线)，下三角为c或0}\\ \text{下三角矩阵：下三角为任意数据(含对角线)，上三角为c或0}\end{cases}$$

$$\begin{bmatrix}1&2&3\\0&4&5\\0&0&6\end{bmatrix}\quad\begin{bmatrix}1&2&3\\7&4&5\\7&7&6\end{bmatrix}\qquad\begin{bmatrix}1&0&0\\2&3&0\\4&5&6\end{bmatrix}\quad\begin{bmatrix}1&7&7\\2&3&7\\4&5&6\end{bmatrix}$$

上三角矩阵　　　　下三角矩阵

- 压缩存储方法：只存储上(下)三角元素+常数0/c

  $[0..n-1][0..n-1] \Rightarrow [0..\frac{n(n+1)}{2}]$

  $[0]\ ..\ [\frac{n(n+1)}{2}-1]$：上/下三角

  $[\frac{n(n+1)}{2}]$：常数0/c

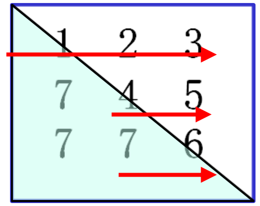
- 元素的排列顺序有行优先/列优先，不失一般性，选择行优先方式

§5. 数组和广义表

- 元素的排列顺序有行优先/列优先，不失一般性，选择行优先方式

[0..n-1][0..n-1] => [0..$\frac{n(n+1)}{2}$]

$$k=\begin{cases}\dfrac{i*(2n-i-1)}{2}+j & (i\leqslant j)\\ n(n+1)/2 & (i>j)\end{cases}$$

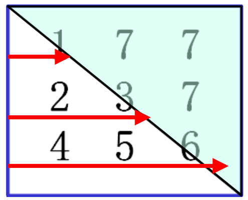


上三角矩阵
行优先

a[0][0] = sa[0] = 1
a[0][1] = sa[1] = 2
a[0][2] = sa[2] = 3
a[1][1] = sa[3] = 4
a[1][2] = sa[4] = 5
a[2][2] = sa[5] = 6
a[1][0] = a[2][0]=a[2][1] = sa[6] = 7

[0..n-1][0..n-1] => [0..$\frac{n(n+1)}{2}$]

$$k=\begin{cases}\dfrac{i(i+1)}{2}+j & (i\geqslant j)\\ n(n+1)/2 & (i<j)\end{cases}$$



下三角矩阵
行优先

a[0][0] = sa[0] = 1
a[1][0] = sa[1] = 2
a[1][1] = sa[2] = 3
a[2][0] = sa[3] = 4
a[2][1] = sa[4] = 5
a[2][2] = sa[5] = 6
a[0][1] = a[0][2]=a[1][2] = sa[6] = 7

§5. 数组和广义表

5.3.矩阵的压缩存储

5.3.1.特殊矩阵的压缩存储

★ 对角矩阵

n阶矩阵，非零元素集中在以主对角线为中心的带状区域内，其它位置的元素值为0或常数c

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 9 & 10 & 11 \\ 0 & 0 & 0 & 12 & 13 \end{bmatrix} \qquad \begin{bmatrix} 1 & 9 & 9 & 9 & 9 \\ 9 & 2 & 9 & 9 & 9 \\ 9 & 9 & 3 & 9 & 9 \\ 9 & 9 & 9 & 4 & 9 \\ 9 & 9 & 9 & 9 & 5 \end{bmatrix}$$

- 压缩存储方法：以一定规律从二维 => 一维
- 元素的排列顺序有行优先/列优先，不失一般性，选择行优先方式

§ 5. 数组和广义表

- 压缩存储方法：以一定规律从二维 => 一维

[0..n-1][0..n-1] => [0..3n-2]

n^2 => (3n-2) + 1 = 3n-1

[0]..[3n-3]：带状区域内的元素

[3n-2]：常数0/c

$$
\begin{cases} k=2i+j & (|i-j| \leqslant 1) \\ k=3n-2 & (|i-j|>1) \end{cases}
$$

$$
\begin{matrix} 1 & 2 & 0 & 0 & 0 \\ 3 & 4 & 5 & 0 & 0 \\ 0 & 6 & 7 & 8 & 0 \\ 0 & 0 & 9 & 10 & 11 \\ 0 & 0 & 0 & 12 & 13 \end{matrix}
$$

a[0][0] = sa[0] = 1
a[0][1] = sa[1] = 2
a[1][0] = sa[2] = 3
...
a[4][3] = sa[11] = 12
a[4][4] = sa[12] = 13
a[0][2]/a[4][2]等 = sa[13] = 0

[0..n-1][0..n-1] => [0..n]

n^2 => n+1

[0]..[n-1]：对角线的数字

[n]：常数0/c

$$
\begin{cases} k=i & (i=j) \\ k=n & (i \neq j) \end{cases}
$$

$$
\begin{matrix} 1 & 9 & 9 & 9 & 9 \\ 9 & 2 & 9 & 9 & 9 \\ 9 & 9 & 3 & 9 & 9 \\ 9 & 9 & 9 & 4 & 9 \\ 9 & 9 & 9 & 9 & 5 \end{matrix}
$$

a[0][0] = sa[0] = 1
a[1][1] = sa[1] = 2
a[2][2] = sa[2] = 3
a[3][3] = sa[3] = 4
a[4][4] = sa[4] = 5

a[0][1]/a[4][3]等 = sa[5] = 9

§5.数组和广义表

5.3.矩阵的压缩存储

5.3.1.特殊矩阵的压缩存储

5.3.2.稀疏矩阵

★ 含义：非零元素少，但分布无规律

- 矩阵可以是任意 m*n 形式，不要求方阵
- m*n的矩阵中有t个非零元素，若 $\delta=\frac{t}{m*n}\leqslant 0.05$ 则认为是稀疏矩阵， δ称为稀疏因子

★ 稀疏矩阵的形式定义

P.96 – 97

★ 稀疏矩阵的压缩存储

记录元素值的同时记录该元素的行、列值(行优先)

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

三元组表：从0开始	P.97 例 从1开始
(0, 1, 12)	(1, 2, 12)
(0, 2, 9)	(1, 3, 9)
(2, 0, -3)	(3, 1, -3)
(2, 5, 14)	(3, 6, 14)
(3, 2, 24)	(4, 3, 24)
(4, 1, 18)	(5, 2, 18)
(5, 0, 15)	(6, 1, 15)
(5, 3, -7)	(6, 4, - 7)

§5. 数组和广义表

5.3. 矩阵的压缩存储

5.3.2. 稀疏矩阵

★ 稀疏矩阵的压缩存储的顺序表示 - 三元组表

```
#define MAXSIZE   12500 //非零元素的最大数量
typedef struct {
    int       i, j;      //行、列下标
    ElemType  e;         //值
} Triple;
typedef struct {
    Tirple data[MAXSIZE];//非零元素
    int mu, nu, tu;        //行数、列数、非零元素数
} TSMatrix;
```

1、书 P.98 为 data[MAXSIZE+1] 且说明 date[0] 不用，是因为存储从1开始，而本定义是从[0]开始

2、若仿照顺序表、顺序栈等，定义为 Triple *data; 使用时再动态申请空间并保持可扩展性，应该更合理一些（矩阵的运算会使非零元素增减）

§5. 数组和广义表

5.3. 矩阵的压缩存储

5.3.2. 稀疏矩阵

★ 稀疏矩阵的压缩存储的顺序表示 – 三元组表

基于三元组表的矩阵转置算法：

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

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

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

行列互换
顺序调整

三元组表：
从[0]开始
(0, 2, -3)
(0, 5, 15)
(1, 0, 12)
(1, 4, 18)
(2, 0, 9)
(2, 3, 24)
(3, 5, -7)
(5, 2, 14)

规律？

列值相同的元素，
转置后的相对位置
不变

```
status TransposeSMatrix(TSMatrix M, TSMatrix &T)
{   T.mu = M.nu;  T.nu = M.mu;  T.tu = M.tu; //行列互换
    if (T.tu) {
      q=0;
      for(col=0; col<M.nu; col++)  //外循环为M的列
         for(p=0; p<M.tu; p++)     //扫描全部非零数据
            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++;
            }
    } //end of if
    return OK;
}
```

和P.99 算法5.1略有不同
书从1开始，本例从0开始

行列值互换

普通矩阵：for(i=0; i<mu; i++)
for(j=0; j<nu; i++)
T[j][i] = M[i][j];
时间复杂度O(mu*nu)

本算法：时间复杂度：O(nu*tu)
若普通矩阵，则 tu = mu*nu => O(mu*nu²)

本算法=> 只适用于 tu << mu*nu 的情况

col=0 时

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

三元组表：
从[0]开始
(0, 2, -3)
(0, 5, 15)

col=1 时

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

三元组表：
从[0]开始
(0, 2, -3)
(0, 5, 15)
(1, 0, 12)
(1, 4, 18)

col=2 时

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

三元组表：
从[0]开始
(0, 2, -3)
(0, 5, 15)
(1, 0, 12)
(1, 4, 18)
(2, 0, 9)
(2, 3, 24)

col=3 时

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

三元组表：
从[0]开始
(0, 2, -3)
(0, 5, 15)
(1, 0, 12)
(1, 4, 18)
(2, 0, 9)
(2, 3, 24)
(3, 5, -7)

col=4时

找不到符合
条件的元素

col=5 时

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

三元组表：
从[0]开始
(0, 2, -3)
(0, 5, 15)
(1, 0, 12)
(1, 4, 18)
(2, 0, 9)
(2, 3, 24)
(3, 5, -7)
(5, 2, 14)

§5. 数组和广义表

5.3. 矩阵的压缩存储

5.3.2. 稀疏矩阵

★ 稀疏矩阵的压缩存储的顺序表示 – 三元组表

基于三元组表的矩阵快速转置算法：

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

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

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

行列互换
顺序调整

三元组表：
从[0]开始
(0, 2, -3)
(0, 5, 15)
(1, 0, 12)
(1, 4, 18)
(2, 0, 9)
(2, 3, 24)
(3, 5, -7)
(5, 2, 14)

1、M中第[0]项(0,1,12)如果放入T中，应是哪个位置?
答:转置后行为1
统计列为0的项数，共2项
=> 应放入[2]中

2、M中第[1]项(0,2, 9)如果放入T中，应是哪个位置?
答:转置后行为2
统计列为0的项数，共2项
统计列为1的项数，共2项
=> 应放入[4]中

3、M中第[5]项(4,1,18)如果放入T中，应是哪个位置?
答:转置后行为1
统计列为0的项数，共2项
已放入1项列为1的项数
=> 应放入[3]中

准备工作：
1、统计M中每列的非零元素的数量，放入num中
2、计算M每列的第1个非零元素在T中的位置,放入cpot中

```
Status FastTransposeSMatrix(TSMatrix M,TSMatrix &T)
{   T.mu=M.nu;  T.nu=M.mu;  T.tu=M.tu;    //行列互换
    if (T.tu) {
        for (col=0; col<M.nu; col++)
            num[col]=0;                                  num数组清0
        for (t=0; t<M.tu; t++)                     M.data[t].j是列下标
            num[ M.data[t].j ]++;                       num对应下标++
        cpot[0]=0;                                 计算cpot数组各项的值
        for (col=1; col<M.nu; col++)
            cpot[col] = cpot[col-1] + num[col-1];
        for (p=0; p<M.tu; p++) {  //循环M的所有非零元素
            col = M.data[p].j; //取列下标
            q   = cpot[col];    //该列在T中的起始下标位置
                                //即该元素在T中的插入位置
            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]++; //起始位置已放了数据
            }            //++后是下一插入位置
        } //end of if
    return OK;
}
```

```
普通矩阵：for(i=0; i<mu; i++)
              for(j=0; j<nu; i++)
                  T[j][i] = M[i][j];
          时间复杂度O(mu*nu)
本算法：4个并列的循环 nu+tu+nu+tu
时间复杂度：O(nu+tu)
若普通矩阵，则 tu = mu*nu => O(mu*nu)
与普通矩阵转置的时间复杂度相同
```

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

cpot公式：

cpot[0] = 0

cpot[col]=cpot[col-1]+num[col-1]   0<col<M.nu

与P.99表5.1/公式5-4的差别：从1开始/从0开始

准备工作：

1、统计M中每列的非零元素的数量，放入num中

2、计算M每列的第1个非零元素在T中的位置，放入cpot中

```
Status FastTransposeSMatrix(TSMatrix M,TSMatrix &T)
{    T.mu=M.nu;  T.nu=M.mu;  T.tu=M.tu;    //行列互换
     if (T.tu) {
        for (col=0; col<M.nu; col++)
            num[col]=0;                              num数组清0
        for (t=0; t<M.tu; t++)              M.data[t].j是列下标
            num[ M.data[t].j ]++;                    num对应下标++
        cpot[0]=0;                           计算cpot数组各项的值
        for (col=1; col<M.nu; col++)
            cpot[col] = cpot[col-1] + num[col-1];
        for (p=0; p<M.tu; p++) {  //循环M的所有非零元素
            col = M.data[p].j; //取列下标
            q   = cpot[col];   //该列在T中的起始下标位置
                               //即该元素在T中的插入位置
            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]++; //起始位置已放了数据
            }            //++后是下一插入位置
        } //end of if
     return OK;
}
```

普通矩阵：for(i=0; i<mu; i++)
for(j=0; j<nu; i++)
T[j][i] = M[i][j];
时间复杂度O(mu*nu)
本算法：4个并列的循环 nu+tu+nu+tu
时间复杂度：O(nu+tu)
若普通矩阵，则 tu = mu*nu => O(mu*nu)
与普通矩阵转置的时间复杂度相同

循环执行第1次时的情况

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

cpot公式：
cpot[0] = 0
cpot[col]=cpot[col-1]+num[col-1]   0<col<M.nu

三元组表：
从[0]开始
(0, 1, 12)
(0, 2, 9)
(2, 0, -3)
(2, 5, 14)
(3, 2, 24)
(4, 1, 18)
(5, 0, 15)
(5, 3, -7)

行列互换
顺序调整

col=1
q=2
赋值
cpot[1]++

三元组表：
从[0]开始
[0]
[1]
[2]   (1, 0, 12)
[3]
[4]
[5]
[6]
[7]

§5. 数组和广义表

5.3. 矩阵的压缩存储

5.3.2. 稀疏矩阵

★ 稀疏矩阵的压缩存储的顺序表示 - 三元组表

- 带行连接信息的三元组表及稀疏矩阵相乘(略)

  P.100 - 103

- 顺序表示的缺点

  同线性表的顺序表示，插入/删除元素效率低

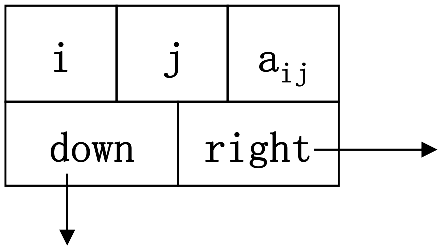
  例：矩阵相加，会导致元素的插入/删除

$$\begin{bmatrix} 0 & 5 & 0 \\ 0 & 0 & 0 \\ 3 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & -5 & 0 \\ 0 & 0 & 2 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 2 \\ 3 & 0 & 0 \end{bmatrix}$$

★ 稀疏矩阵的压缩存储的链式表示 - 十字链表

一个结点由5部分组成

[行标，列标，值，下指针，右指针]




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

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

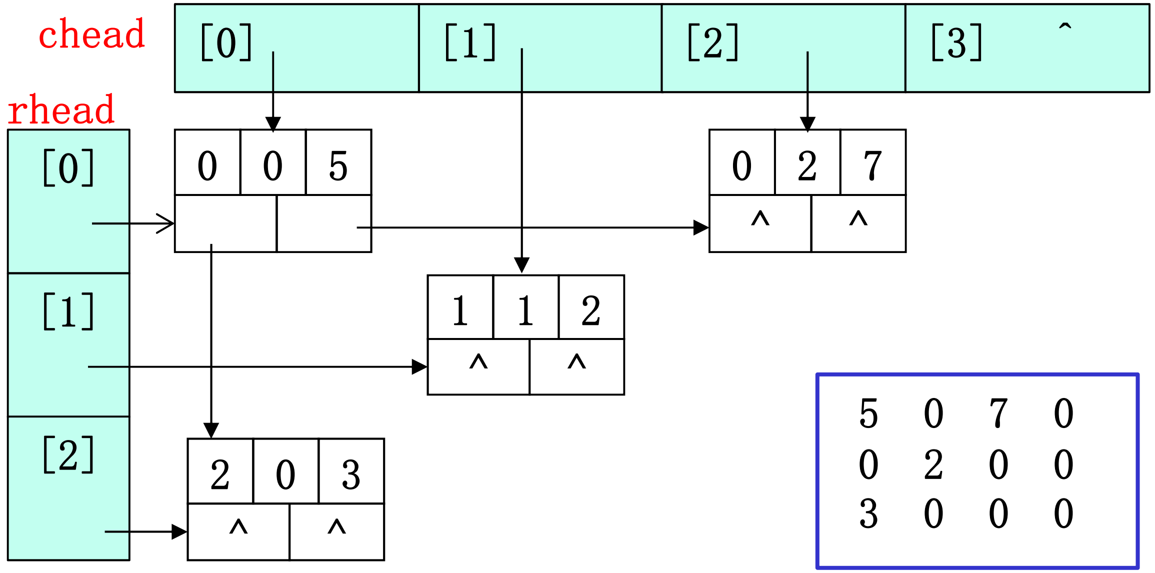
§5.数组和广义表

★ 稀疏矩阵的压缩存储的链式表示 – 十字链表

一个结点由5部分组成

[行标，列标，值，下指针，右指针]

★ 头指针分别存放在两个一维数组(同顺序表，动态申请当做一维数组使用)中，数组大小为m,n，每个数组元素中存放一个单链表的头指针



$$\begin{matrix} 5 & 0 & 7 & 0 \\ 0 & 2 & 0 & 0 \\ 3 & 0 & 0 & 0 \end{matrix}$$

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

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

若OLink rhead[m]，事先不知m值

```
CrossList L;
L.rhead = (OLink *)malloc( m*sizeof(OLink) ); //C
L.chead = new OLink[n];                        //C++
L.rhead[0] = NULL;
...
```

- 基于十字链表的矩阵建立(自学，需掌握)
- 基于十字链表的矩阵相加(自学，需掌握)

§5.数组和广义表

5.4.广义表的定义

5.4.1.广义表的含义

广义表是一种线性表，其中元素既可以是元素，也可以是另一个线性表

★ 是线性表的推广，也称为列表(lists　表-list)

5.4.2.广义表的形式定义

P.107 - 108

5.4.3.基本术语

★ 广义表的长度：表中元素的个数
★ 原子：广义表中元素为单个元素
★ 子表：广义表中元素为另一个线性表
★ 表头：非空线性表的第1个元素(可以是单元素/表)
★ 表尾：除表头外的其它元素组成的表(必然是表)
★ 广义表的深度：表中元素最大的层数

§5. 数组和广义表

例1：A=()

长度为零，无元素

GetHead(A) = ∅

GetTail(A) = () //不是∅

例2：B=(e)

长度为1，元素是原子e

GetHead(B) = e

GetTail(B) = () //不是∅

例3：C=(a, (b, c, d))

长度为2，两个元素分别原子a和子表(b, c, d)

GetHead(C) = a

GetTail(C) = ((b, c, d)) //两层括号

例4：D=(A，B，C)

长度为3，三个元素分别是广义表A/B/C

GetHead(D) = A

GetTail(D) = (B，C)

例5：E=(a, E)

长度为2，是递归表，元素分别是a和E

E是无限广义表，E=(a, (a, (a, (a, …))))

GetHead(E) = a;

GetTail(E) = (E) //有括号

§5. 数组和广义表

5.4. 广义表的定义

5.4.4. 广义表的基本特性

★ 多层次结构，子表可以嵌套

★ 允许在一个广义表中包含另一个表

★ 允许递归

★ 元素之间除次序关系外还存在层次关系(深度)

例：F＝（a，（b，（c，（d））））

a的位置：第1层

b的位置：第2层

c的位置：第3层

d的位置：第4层

广义表的深度：4

例1：A=()	深度：1
例2：B=(e)	深度：1
例3：C=(a,(b,c,d))	深度：2
例4：D=(A，B，C)	深度：3 //A/B为1，C为2
例5：E=(a,E)	深度：∞

§5.数组和广义表

5.5.广义表的存储结构(略)

5.6.m元多项式的表示(略)

5.7.广义表的递归算法(略)