第6章 数组和稀疏矩阵

6.1 数　组

6.2 稀疏矩阵

6.1 数　组

6.1.1 数组的基本概念

从逻辑结构上看，一维数组A是n（$n>1$）个相同类型数据元素a_1、a_2、…、a_n构成的有限序列，其逻辑表示为：

$$A=(a_1, a_2, \ldots, a_n)$$

其中，a_i（$1 \leqslant i \leqslant n$）表示数组$A$的第$i$个元素。

一个m行n列的二维数组A可以看作是每个数据元素都是相同类型的一维数组的一维数组。

$$A=\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & & & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \Rightarrow A=[A_1, A_2, \ldots, A_m]$$

$$\Downarrow$$

$$A_1=[a_{1,1}, a_{1,2}, \ldots, a_{1,n}]$$

$$A_2=[a_{1,1}, a_{2,2}, \ldots, a_{2,n}]$$

$$\cdots \quad \cdots$$

$$A_m=[a_{m,1}, a_{m,2}, \ldots, a_{m,n}]$$

由此看出，多维数组是线性表的推广。

数组抽象数据类型＝逻辑结构＋基本运算（运算描述）

数组的基本运算如下：

❶ Value(A,index$_1$,index$_2$,…,index$_d$)：即A(index$_1$,index$_2$,…,index$_d$)=e，元素赋值。

❷ Assign(A,e,index$_1$,index$_2$,…,index$_d$)：即e= A(index$_1$,index$_2$,…,index$_d$)，取元素值。

❸ ADisp(A,b_1,b_2,…,b_d)：输出d维数组A的所有元素值。

6.1.2 数组的存储结构

将数组的所有元素存储在一块地址连续的内存单元中，这是一种顺序存储结构。

几乎所有的计算机语言都支持数组类型，以C/C++语言为例，其中数组数据类型具有以下性质：

- 数组中的数据元素数目固定。
- 数组中的所有数据元素具有相同的数据类型。
- 数组中的每个数据元素都有一组唯一的下标。
- 数组是一种随机存储结构。可随机存取数组中的任意数据元素。

一维数组：一旦a_1的存储地址LOC(a_1)确定，并假设每个数据元素占用k个存储单元，则任一数据元素a_i的存储地址LOC(a_i)就可由以下公式求出：

$$\mathrm{LOC}(a_i)=\mathrm{LOC}(a_1)+(i-1)*k \qquad (0\leqslant i\leqslant n)$$

数组a：$\underbrace{a_1 \quad a_2 \quad a_3 \quad \cdots \quad a_{i-1}}_{\text{共}i-1\text{个元素}} \quad a_i \quad \cdots \quad a_n$

表明一维数组具有随机存储特性。

对于一个m行n列的二维数组$A_{m\times n}$，存储方式：

- 以行序为主序的存储
- 以列序为主序的存储

❶ 以行序为主序的存储方式

$$A=\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & & & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

$$\underbrace{a_{1,1},\ a_{1,2},\ \ldots,\ a_{1,n}}_{\text{第1行的元素}}\quad \underbrace{a_{2,1},\ a_{2,2},\ \ldots,\ a_{2,n}}_{\text{第2行的元素}}\quad \ldots,\quad \underbrace{a_{m,1},\ a_{m,2},\ \ldots,\ a_{m,n}}_{\text{第}m\text{行的元素}}$$

第1行的元素　　第i行的元素

$a_{1,1}$， $a_{1,2}$， …， $a_{1,n}$， …， $a_{i,1}$， $a_{i,2}$， …， $a_{i,j-1}$， $a_{i,j}$， …$a_{i,n}$， …

1~i−1行，每行n个元素，计$(i-1)\times n$个元素

第i行中，$a_{i,j}$元素前有$j-1$个元素

则$a_{i,j}$元素前共有 $(i-1)\times n+j-1$个元素

$$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{1,1})+[(i-1)\times n+(j-1)]\times k$$

❷ 以列序为主序的存储方式

同理可推出在以列序为主序的计算机系统中有：

$$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{1,1})+[(j-1)\times m+(i-1)]\times k$$

其中m为行数。

所以，二维数组采用顺序存储结构时，也具有随机存取特性。

是指给定序号i（下标），可以在O(1)的时间内找到相应的元素值。

同样，多维数组采用顺序存储时具有随机存储特性。

6.1.3　特殊矩阵的压缩存储

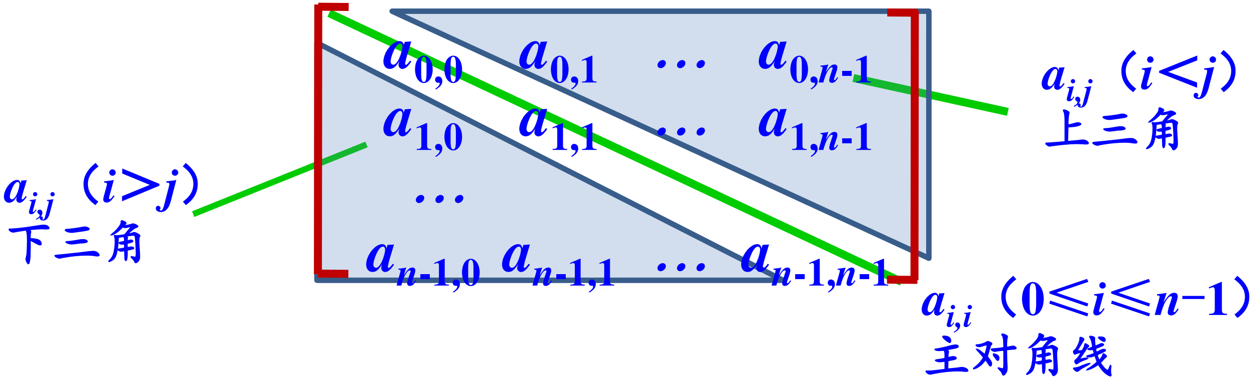
特殊矩阵的主要形式有：

- 对称矩阵
- 上三角矩阵 / 下三角矩阵
- 对角矩阵

它们都是方阵，即行数和列数相同。

1、对称矩阵的压缩存储

若一个n阶方阵$A[n][n]$中的元素满足$a_{i,j}=a_{j,i}$（$0 \leqslant i, j \leqslant n-1$），则称其为$n$阶**对称矩阵**。



以行序为主序存储其下三角+主对角线的元素。

$$i \geqslant j \qquad \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \cdots & & & \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix}$$

下三角+主对角线

$$a_{i,j} \longrightarrow b_k \qquad k = ?$$

$$a_{0,0},\quad a_{1,0},\quad a_{1,1},\quad \cdots,\quad a_{n-1,0},\quad a_{n-1,1},\quad \cdots,\quad a_{n-1,n-1}$$

$$B = \underbrace{(b_0,\quad b_1,\quad b_2,\quad \cdots\cdots,\quad \cdots\cdots,\quad b_s)}_{n(n+1)/2\text{个元素}}$$

$$b_k \updownarrow$$

$$B=(\underbrace{a_{0,0}}_{\text{1个元素}},\underbrace{a_{1,0},a_{1,1}}_{\text{2个元素}},\ldots,\underbrace{a_{i-1,0},\ \ldots\ ,a_{i-1,i-1}}_{i\text{个元素}},\ \underbrace{a_{i,0}\ ,\ \ldots,\ a_{i,j-1}}_{j\text{个元素}},\ a_{i,j},\ \ldots,\ a_{n-1,n-1})$$

共计$i(i+1)/2+j$个元素

$$k=\begin{cases}\dfrac{i(i+1)}{2}+j & \text{当 } i\geqslant j\text{时（下三角+主对角线的元素）}\\[2ex] \dfrac{j(j+1)}{2}+i & \text{当}i<j\text{时（}a_{i,j}=a_{j,i}\text{）}\end{cases}$$

n^2个元素 ⟷ $n(n+1)/2$个元素

‖ ‖

$A[0..n-1,0..n-1]$ ⟷ $B[0..n(n+1)/2-1]$

↓ ↓

$a[i][j]$ ⟷ $b[k]$

↓

$$k=\begin{cases}\dfrac{i(i+1)}{2}+j & \text{当 } i\geqslant j\text{时}\\ \dfrac{j(j+1)}{2}+i & \text{当}i<j\text{时}(a_{i,j}=a_{j,i})\end{cases}$$

对于对称矩阵A，采用一维数组B存储，并提供A的所有运算。

2、三角矩阵的压缩存储

● 上三角矩阵：

$$\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ & a_{1,1} & \cdots & a_{1,n-1} \\ c & & & \cdots \\ & & & a_{n-1,n-1} \end{bmatrix} \qquad i \leqslant j$$

$a_{i,j} \to b_k$

$$B=(\underbrace{a_{0,0},a_{0,1},\ldots,a_{0,n-1}}_{n\text{个元素}},\underbrace{a_{1,1},\ldots,a_{1,n-1}}_{n-1\text{个元素}},\ldots,\underbrace{a_{i-1,i-1},\ldots,a_{i-1,n-1}}_{n-i+1\text{个元素}},\underbrace{a_{i,i},\ldots,a_{i,j-1}}_{j-i\text{个元素}},a_{i,j},\ldots)$$

共计$i(2n-i+1)/2+j-i$个元素

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

存放常量c

- 下三角矩阵

$$i \geqslant j \quad \begin{bmatrix} a_{0,0} & & & c \\ a_{1,0} & a_{1,1} & & \\ \cdots & & & \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-1} \end{bmatrix} \quad a_{i,j} \rightarrow b_k$$

$$B = (a_{0,0},\ a_{1,0},\ a_{1,1},\ \ldots,\ a_{n-1,0},\ a_{n-1,1},\ \ldots,\ a_{n-1,n-1})$$

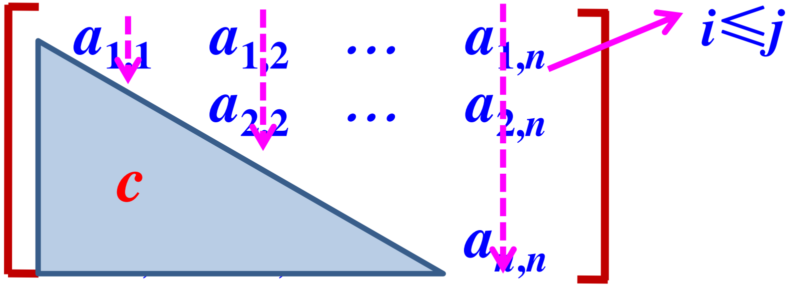
$$k = \begin{cases} \dfrac{i(i+1)}{2} + j & \text{当 } i \geqslant j \text{ 时} \\[2ex] \dfrac{n(n+1)}{2} & \text{当 } i < j \text{ 时} \end{cases}$$

存放一个常量c

【例6-1】若将n阶上三角矩阵A按列优先顺序压缩存放在一维数组$B[1..n(n+1)/2]$中，A中第一个非零元素$a_{1,1}$存于B数组的b_1中，则应存放到b_k中的非零元素$a_{i,j}$（$i \leqslant j$）的下标i、j与k的对应关系是________。

A. $i(i+1)/2+j$　　B. $i(i-1)/2+j$

C. $j(j+1)/2+i$　　D. $j(j-1)/2+i$



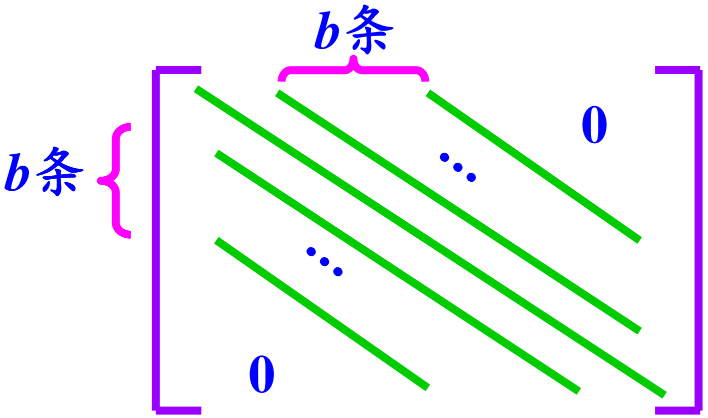
$1 \sim j-1$列的元素个数：$j(j-1)/2$

第j列a_{ij}之前的元素个数：$i-1$

$k=j(j-1)/2+i-1+1=j(j-1)/2+i$

- 按行还是按列
- 初始下标从0还是从1开始

3、对角矩阵的压缩存储



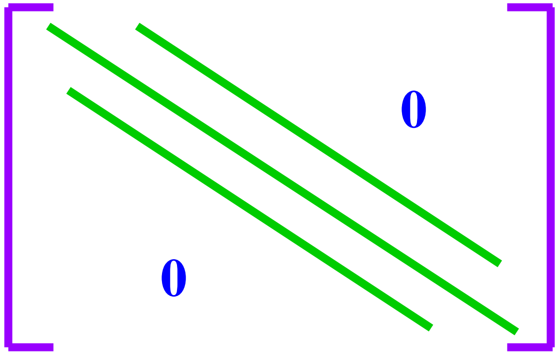
半带宽为b的对角矩阵

对角矩阵 压缩存储

$A \longleftrightarrow B$

$a[i][j] \longleftrightarrow b[k]$

当b=1时称为三对角矩阵 →



其压缩地址计算公式如下：

$$k = 2i + j$$

思考题：

特殊矩阵为什么采用压缩存储，需要解决什么问题？

——本讲完——

6.2 稀疏矩阵

稀疏矩阵的定义

一个阶数较大的矩阵中的非零元素个数s相对于矩阵元素的总个数t十分小时，即s<<t时，称该矩阵为**稀疏矩阵**。

例如一个100×100的矩阵，若其中只有100个非零元素，就可称其为稀疏矩阵。

稀疏矩阵和特殊矩阵的不同点：

- 特殊矩阵的特殊元素（值相同元素、常量元素）分布有规律。
- 稀疏矩阵的特殊元素（非0元素）分布没有规律。

6.2.1 稀疏矩阵的三元组表示

稀疏矩阵的压缩存储方法是只存储非零元素。

稀疏矩阵中的每一个非零元素需由一个三元组：

$$(i\ ,\ j\ ,\ a_{i,j})$$

唯一确定，稀疏矩阵中的所有非零元素构成三元组线性表。

稀疏矩阵三元组表示的演示

一个6×7阶稀疏矩阵*A*的三元组线性表表示

$$A_{6\times7}=\begin{bmatrix}0&0&1&0&0&0&0\\0&2&0&0&0&0&0\\3&0&0&0&0&0&0\\0&0&0&5&0&0&0\\0&0&0&0&6&0&0\\0&0&0&0&0&7&4\end{bmatrix}$$

一个稀疏矩阵*A*

(0,2,1)　(1,1,2)　(2,0,3)　(3,3,5)　(4,4,6)　(5,5,7)　(5,6,4)

三元组线性表：

((0,2,1)，(1,1,2)，(2,0,3)，(3,3,5)，(4,4,6)，(5,5,7)，(5,6,4))

把稀疏矩阵的三元组线性表按顺序存储结构存储，则称为稀疏矩阵的三元组顺序表。

```
#define MaxSize  100        //矩阵中非零元素最多个数
typedef struct
{    int r;                 //行号
     int c;                 //列号
     ElemType d;            //元素值
}  TupNode;                 //三元组定义
typedef struct
{    int rows;              //行数值
     int cols;              //列数值
     int nums;              //非零元素个数
     TupNode data[MaxSize];
} TSMatrix;                 //三元组顺序表定义
```

存放一个非0元素

存放整个稀疏矩阵

（1）从一个二维矩阵创建其三元组表示

以行序方式扫描二维矩阵A，将其非零的元素插入到三元组t的后面。

$$A_{6\times7}=\begin{bmatrix}0&0&1&0&0&0&0\\0&2&0&0&0&0&0\\3&0&0&0&0&0&0\\0&0&0&5&0&0&0\\0&0&0&0&6&0&0\\0&0&0&0&0&7&4\end{bmatrix}$$

t:

i	j	a_{ij}
0	2	1
1	1	2
2	0	3
3	3	5
4	4	6
5	5	7
5	6	4

约定：data域中表示的非零元素通常以行序为主序顺序排列，它是一种下标按行有序的存储结构。

这种有序存储结构可简化大多数矩阵运算算法。

```
void CreatMat(TSMatrix &t,ElemType A[M][N])
{     int i,j;  t.rows=M;  t.cols=N;  t.nums=0;
    for (i=0;i<M;i++)
    {     for (j=0;j<N;j++)

            if (A[i][j]!=0)
             {     t.data[t.nums].r=i;
                   t.data[t.nums].c=j;
                   t.data[t.nums].d=A[i][j];
                   t.nums++;
             }
    }
}
```

按行、列序方式扫描所有元素

只存储非零元素

（2）三元组元素赋值：$A[i][j]=x$，分为两种情况：

❶ 将一个非0元素修改为另一个非0值，如A[5][6]=8。

修改元素

$$A_{6\times 7}=\begin{bmatrix} 0&0&1&0&0&0&0\\ 0&2&0&0&0&0&0\\ 3&0&0&0&0&0&0\\ 0&0&0&5&0&0&0\\ 0&0&0&0&6&0&0\\ 0&0&0&0&0&7&\boxed{4} \end{bmatrix} \qquad A_{6\times 7}=\begin{bmatrix} 0&0&1&0&0&0&0\\ 0&2&0&0&0&0&0\\ 3&0&0&0&0&0&0\\ 0&0&0&5&0&0&0\\ 0&0&0&0&6&0&0\\ 0&0&0&0&0&7&\boxed{8} \end{bmatrix}$$

i	j	a_{ij}
0	2	1
1	1	2
2	0	3
3	3	5
4	4	6
5	5	7
5	6	4

i	j	a_{ij}
0	2	1
1	1	2
2	0	3
3	3	5
4	4	6
5	5	7
5	6	8

❷ 将一个0元素修改为非0值。如A[3][5]=8

增加元素

$$A_{6\times7}=\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & \boxed{0} & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 4 \end{bmatrix}$$

$$A_{6\times7}=\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & \boxed{8} & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 4 \end{bmatrix}$$

i	j	a_{ij}
0	2	1
1	1	2
2	0	3
3	3	5
4	4	6
5	5	7
5	6	4

	i	j	a_{ij}
	0	2	1
	1	1	2
	2	0	3
	3	3	5
插入 →	3	5	8
	4	4	6
	5	5	7
	5	6	8

算法如下：

```
bool Value(TSMatrix &t,ElemType x,int i,int j)
{      int k=0,  k1;
       if  (i>=t.rows || j>=t.cols)
              return false;                                   //失败时返回false

    while (k<t.nums && i>t.data[k].r) k++;          //查找行
    while (k<t.nums && i==t.data[k].r && j>t.data[k].c)
          k++;                                          //查找列
```

在t中按行、列号查找

```
    if (t.data[k].r==i && t.data[k].c==j)  //存在这样的元素
            t.data[k].d=x;

   else             //不存在这样的元素时插入一个元素
    {     for (k1=t.nums-1;k1>=k;k1--)
          {     t.data[k1+1].r=t.data[k1].r;
                t.data[k1+1].c=t.data[k1].c;
                t.data[k1+1].d=t.data[k1].d;
          }
          t.data[k].r=i;t.data[k].c=j;t.data[k].d=x;
          t.nums++;
    }
    return true;                        //成功时返回true
}
```

修改元素

增加元素

（3）将指定位置的元素值赋给变量　执行$x=A[i][j]$

先在三元组t中找到指定的位置，再将该处的元素值赋给x。

```
bool Assign(TSMatrix t,ElemType &x,int i,int j)
{    int k=0;
     if (i>=t.rows || j>=t.cols)
          return false;                               //失败时返回false

     while (k<t.nums && i>t.data[k].r) k++;           //查找行
     while (k<t.nums && i==t.data[k].r
          && j>t.data[k].c)    k++;                    //查找列

     if (t.data[k].r==i && t.data[k].c==j)
          x = t.data[k].d;
     else
          x = 0;
     return true;                                     //成功时返回true
}
```

在t中按行、列号查找

找到了非0的元素

没有找到，为0元素

（4）输出三元组

从头到尾扫描三元组t，依次输出元素值。

```
void DispMat(TSMatrix t)
{     int i;
      if  (t.nums<=0) return;
      printf(“\t%d\t%d\t%d\n",t.rows,t.cols,t.nums);
      printf("  ------------------\n");
      for (i=0;i<t.nums;i++)
            printf("\t%d\t%d\t%d\n", t.data[i].r,t.data[i].c, t.data[i].d);
}
```

（5）矩阵转置

对于一个$m \times n$的矩阵$A_{m \times n}$，其转置矩阵是一个$n \times m$的矩阵$B_{n \times m}$，满足$b_{i,j}=a_{j,i}$，其中$0 \leqslant i \leqslant m-1$，$0 \leqslant j \leqslant n-1$。

$$A_{6\times7}=\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 4 \end{bmatrix} \xrightarrow{\text{转置}} B_{7\times6}=\begin{bmatrix} 0 & 0 & 3 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 \\ 0 & 0 & 0 & 0 & 0 & 4 \end{bmatrix}$$

i	j	a_{ij}
0	2	1
1	1	2
2	0	3
3	3	5
4	4	6
5	5	7
5	6	4

i	j	b_{ij}
0	2	3
1	1	2
2	0	1
3	3	5
4	4	6
5	5	7
6	5	4

一种非高效的算法：按第0、1、2、⋯、n−1列进行转换

i	j	a_{ij}
0	2	1
1	1	2
2	0	3
3	3	5
4	4	6
5	5	7
5	6	4

i	j	b_{ij}
0	2	3
1	1	2
2	0	1
3	3	5
4	4	6
5	5	7
6	5	4

矩阵转置

```
void TranTat(TSMatrix t,TSMatrix &tb)
{     int p,q=0,v;                                    //q为tb.data的下标
      tb.rows=t.cols;  tb.cols=t.rows;  tb.nums=t.nums;
      if (t.nums!=0)                                  //当存在非零元素时执行转置
      {
          for (v=0;v<t.cols;v++)                      //tb.data[q]中记录以列序排列
              for (p=0;p<t.nums;p++)                  //p为t.data的下标
                  if (t.data[p].c==v)
                  {    tb.data[q].r=t.data[p].c;
                       tb.data[q].c=t.data[p].r;
                       tb.data[q].d=t.data[p].d;
                       q++;
                  }
      }
}
```

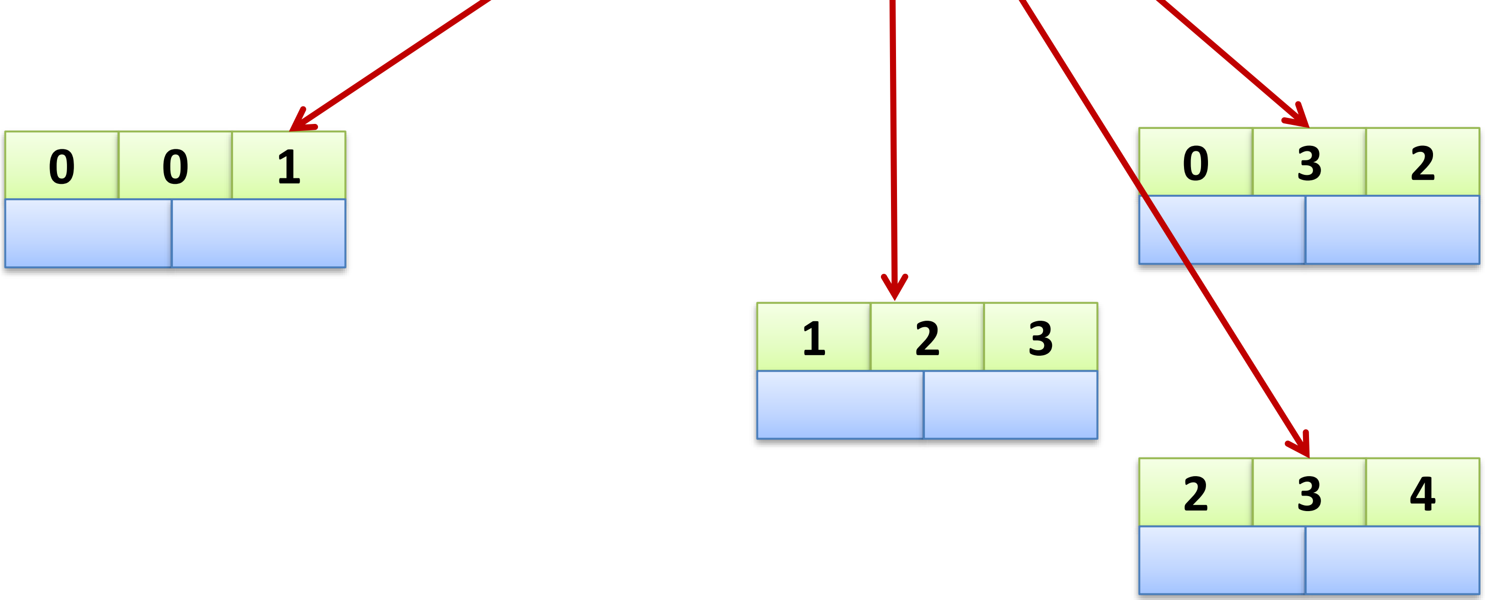
按第0、1、2、…、$n-1$ 列进行转换

若m行n列，t个非0元素，时间复杂度为$O(nt)$。

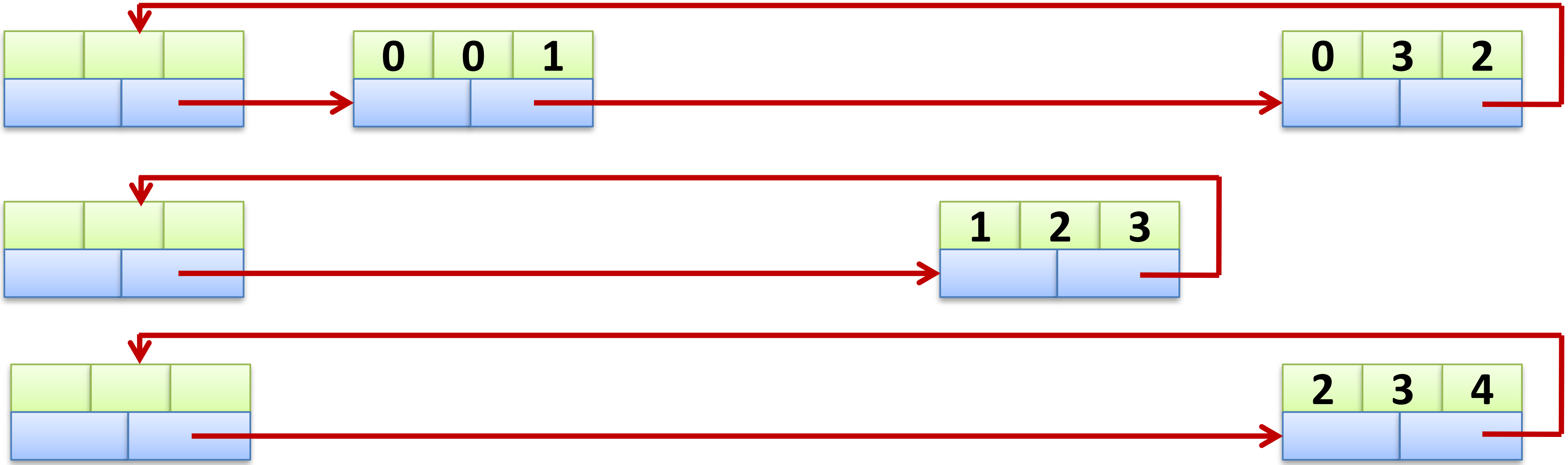
6.2.2 稀疏矩阵的十字链表表示

- 每个非零元素对应一个节点。

$$B_{3\times4}=\begin{bmatrix}1 & 0 & 0 & 2\\ 0 & 0 & 3 & 0\\ 0 & 0 & 0 & 4\end{bmatrix}$$

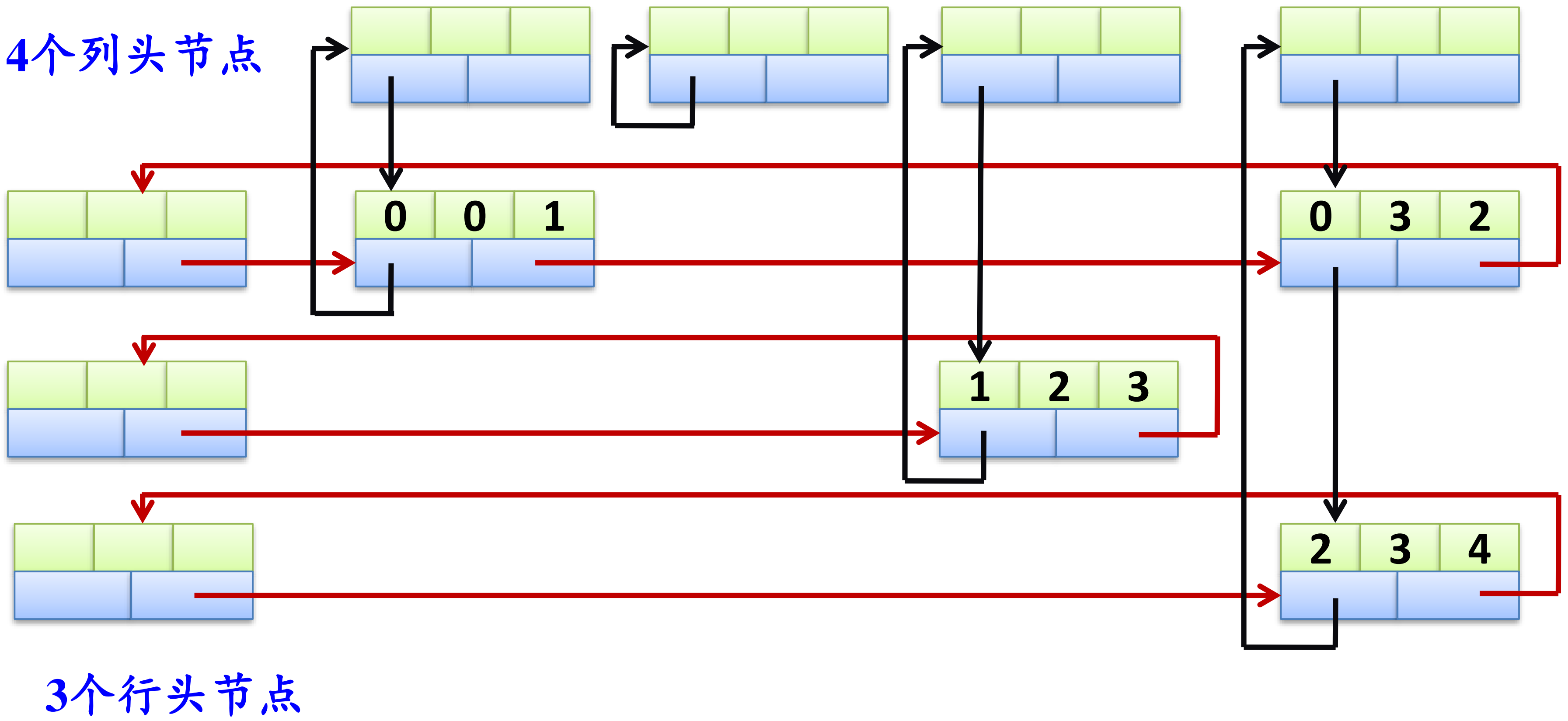


- 每行的所有节点链起来构成一个带行头节点的循环单链表。以 $h[i]$（$0 \leqslant i \leqslant m-1$）作为第$i$行的头节点。

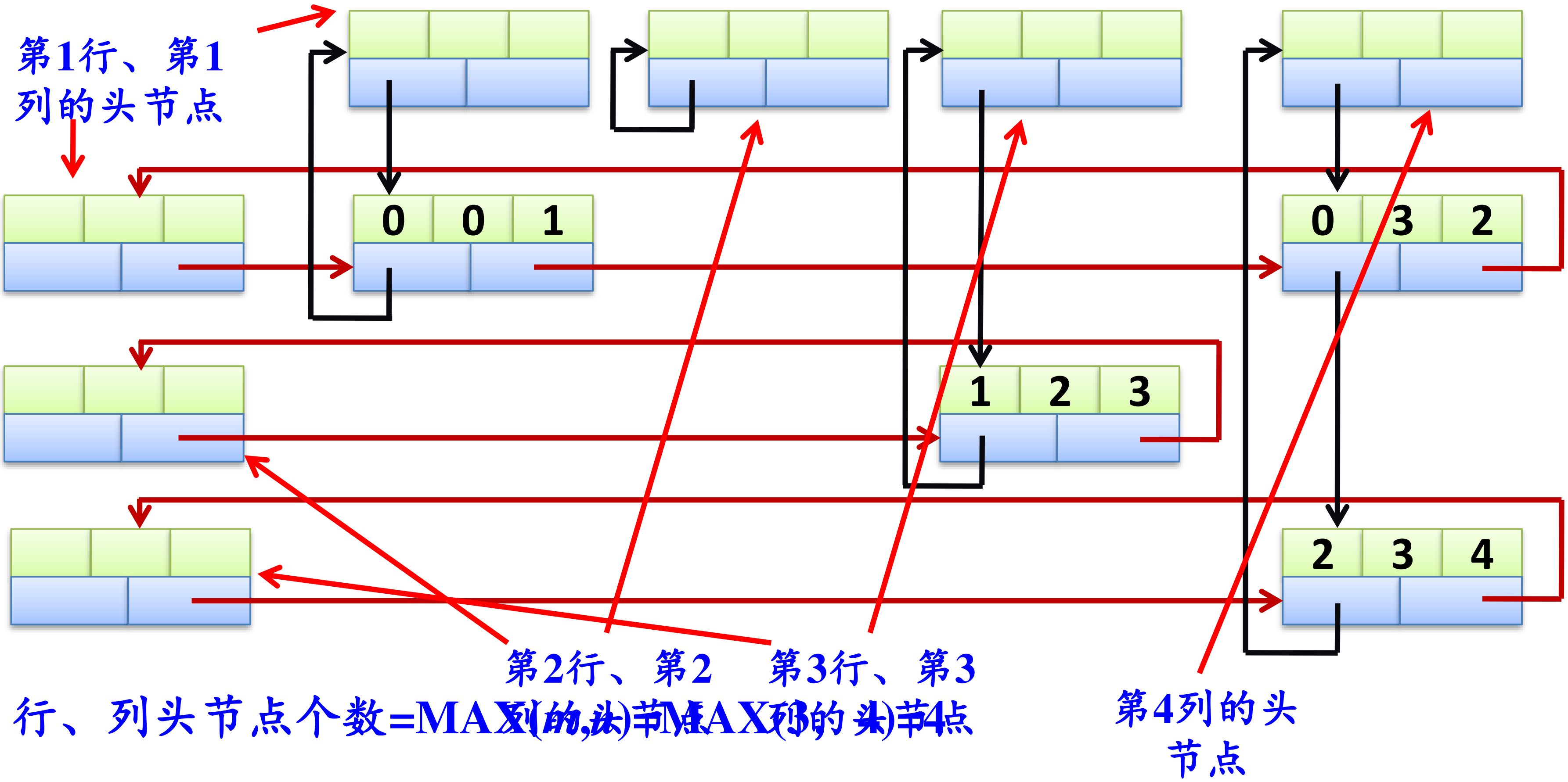


3个行头节点

- 每列的所有节点链起来构成一个带列头节点的循环单链表。以 $h[i]$（$0 \leqslant i \leqslant m-1$）作为第$i$列的头节点。

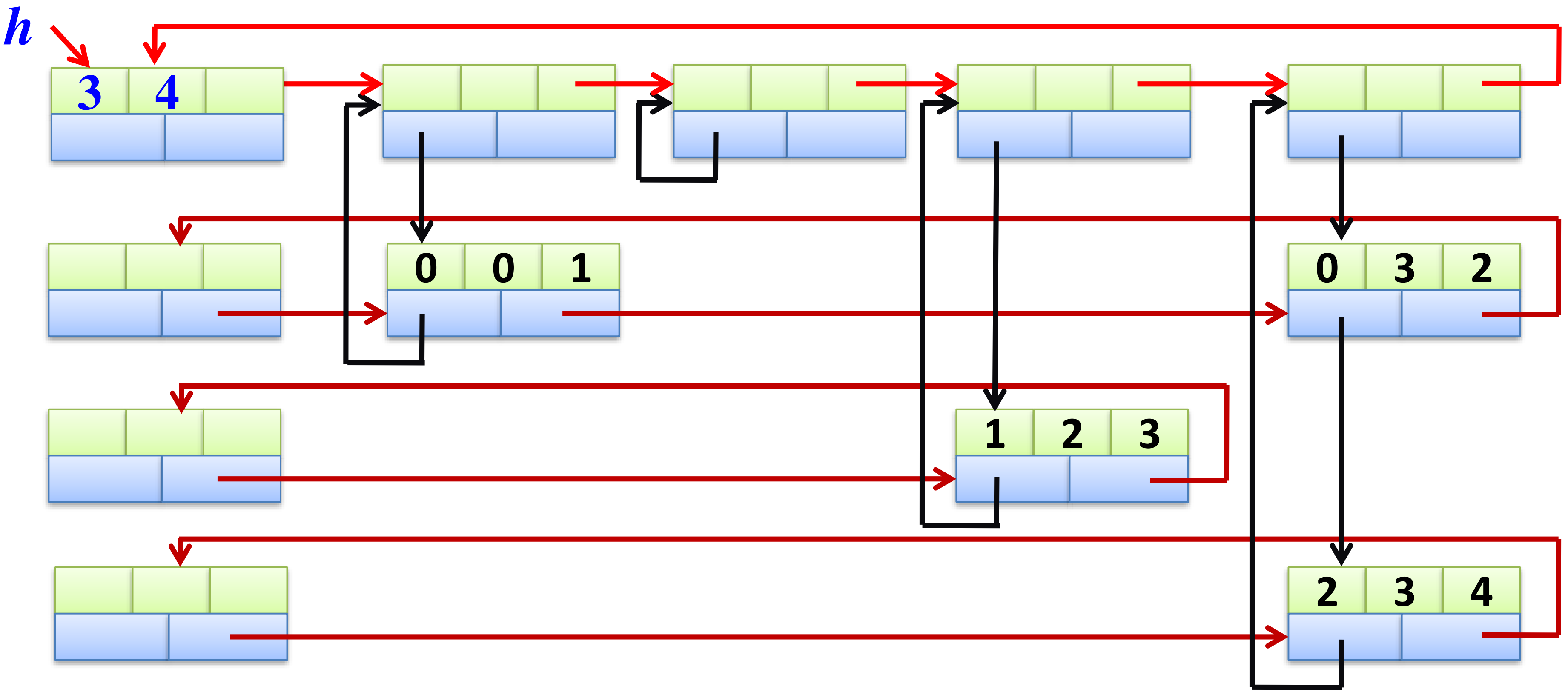

4个列头节点
0 0 1
0 3 2
1 2 3
2 3 4
3个行头节点

行、列头节点可以共享



行、列头节点个数=MAX(m,n)=MAX(3,4)=4

增加一个总头节点，并把所有行、列头节点链起来构成一个循环单链表



总的头节点个数=MAX(m,n)+1=4+1=5

为了统一，设计节点类型如下：

i	j	value
down	right	

(a) 数据节点结构

i	j	link
down	right	

(b) 头节点结构

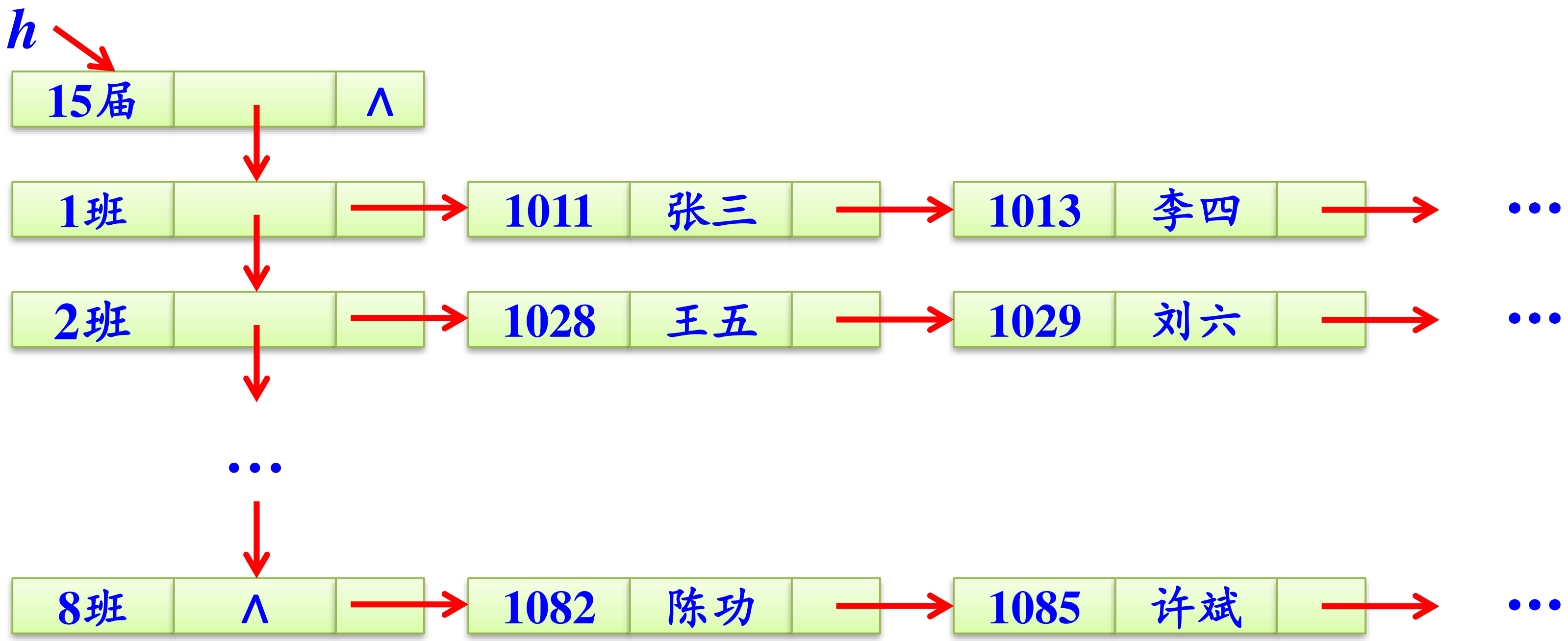
用共用体表示

十字链表节点结构和头节点的数据结构可定义如下：

```
#define M 3                         //矩阵行
#define N 4                         //矩阵列
#define Max ((M)>(N)?(M):(N))       //矩阵行列较大者
typedef struct mtxn
{    int row;                       //行号
     int col;                       //列号
     struct mtxn *right,*down;      //向右和向下的指针
     union                          //共用体类型
     {     int value;
           struct mtxn *link;
     } tag;
} MatNode;                          //十字链表节点类型声明
```

有关算法不做介绍。

【例6-2】十字链表的启示：设计存储某年级所有学生的存储结构：



通过h来唯一标识学生存储结构。

思考题

一个稀疏矩阵采用压缩后，和直接采用二维数组存储相比会失去_______特性。

A.顺序存储　　B.随机存取

C.输入输出　　D.以上都不对

——本章完——

第7周小结

1 数 组

❶ 数组的两个基本操作？

- 按照给定的下标，取（读）相应的元素值
- 按照给定的下标，存（写）相应的元素值

❷ 为什么说数组是线性表的推广或扩展，而不说数组就是一种线性表？

从逻辑结构的角度看，一维数组是一种线性表。

二维数组可以看成数组元素为一维数组的一维数组，所以二维数组是线性结构，可以看成是线性表。

但就二维数组的形状而言，它又是非线性结构，因此将二维数组看成是线性表的推广更准确。

三维及以上维的数组亦如此。

❸ 计算数组中给定元素的地址

- 数组的存储方式（按行或者按列优先存放）
- 计算给定元素的前面的元素个数s
- 每个元素的存储空间k
- 该元素地址=起始元素地址+$s \times k$

设二维数组a[10][20]，每个数组元素占用1个存储单元，若按列优先顺序存放数组元素，a[0][0]的存储地址为200，则a[6][2]的存储地址是多少？

解： a数组的行下标为0～9，列行下标为0～19。

元素a[6][2]前面有列下标为0～1两列，每列10个元素，计2×10=20。在下标为2的列中，元素a[6][2]前面有行下标为0～5的6个元素。

⇨ a[6][2]前面有s=20+6=26个元素。

LOC(a[6][2])=LOC(a[0][0])+26×1=200+26=226。

设某二维数组*a*[10][20]采用顺序存储方式，每个数组元素占用1个存储单元，*a*[0][0]的存储地址为200，*a*[6][2]的存储地址是322，则该数组（　）　。

A.只能按行优先存储

B.只能按列优先存储

C.按行优先存储或按列优先存储均可

D.以上都不对

解： 这里有m=10，n=20，k=1，一个m行n列的二维数组的顺序存储方式只能按行优先或列优先存放。

假设按行优先存放，有LOC($a_{i,\ j}$)=LOC($a_{0,\ 0}$)+(i×n+j)×k，对于a[6][2]元素，其地址 LOC(a[6][2])=LOC(a[0][0])+[6×20+2]×1=322。

假设按列优先存放，有LOC($a_{i,\ j}$)=LOC($a_{0,\ 0}$)+(j×m+i)×k，对于a[6][2]元素，其地址 LOC(a[6][2])=LOC(a[0][0])+[2×10+6]×1=226。

a[6][2]的存储地址是322 ⇨ 只能按行优先存储，**A**

2 特 殊 矩 阵

特殊矩阵
- 对称矩阵
- 上（下）三角矩阵
- 对角矩阵

- 都是方阵
- 元素下标（i，j）可以确定元素的位置

i=j：主对角

i<j：上三角

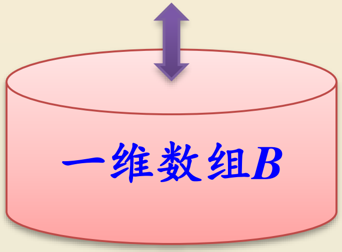
$$
\begin{bmatrix}
a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\
a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\
\cdots & \cdots & \cdots & \cdots \\
a_{n,1} & a_{n,2} & \cdots & a_{n,n}
\end{bmatrix}
$$

i>j：下三角

❶ 什么是特殊矩阵的压缩存储？为什么需要压缩存储？

压缩存储：

提供二维数组的逻辑操作：$A[i][j]$



特殊矩阵采用压缩存储的目的是节省存储空间.

❷ 特殊矩阵压缩存储后具有随机存取特性吗？

这里讨论的特殊矩阵A都是二维的方阵，采用一维数组B压缩存储：

$A[i][j] \Leftrightarrow B[k]$

$k = f(i,\ j)$

f函数的执行时间为O(1)

所以，压缩存储后具有随机存取特性。

③ 在计算对称矩阵的压缩存储时应注意什么问题？

在计算对称矩阵*A*的压缩存储时应注意以下几点：

- 对称矩阵是按上三角还是按下三角存放。
- 对称矩阵元素是按行还是按列优先存放。
- *B*数组的下标从1开始还是从0开始。

设$n \times n$的上三角矩阵A[0..n-1，0..n-1]已压缩到一维数组B[0..m]中，若按列为主序存储，则$A[i][j]$对应的B中存储位置k为多少，给出推导过程。

解： A、B的下标都从0开始。

对于上三角部分的$A[i][j]$（$i \leqslant j$）元素，按列为主序存储时：

前面有0～j-1共j列，第0列有1个元素，第1列有2个元素，…，第j-1列有j个元素，所以这j列的元素个数$=1+2+\cdots+j=j(j+1)/2$；

在第j列中，$A[i][j]$元素前有A[0..i-1，j]共i个元素。所以$A[i][j]$元素前有$j(j+1)/2+i$个元素，则在B中的位置$k=j(j+1)/2+i$。

3 稀 疏 矩 阵

❶ 从特殊元素分布看，稀疏矩阵和特殊矩阵相比有什么不同？

- 特殊矩阵中的特殊元素（相同元素、常数元素）分布有规律性
- 稀疏矩阵中的特殊元素（非0元素）分布没有规律性，即随机的

❷ 稀疏矩阵压缩存储后具有随机存取特性吗？

- 稀疏矩阵用十字链表作存储结构自然失去了随机存取的功能。
- 即便用三元组表的顺序存储结构，存取下标为*i*和*j*的元素时，要扫描三元组表，时间复杂度为O(t)，因此也失去了随机存取的功能。