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。

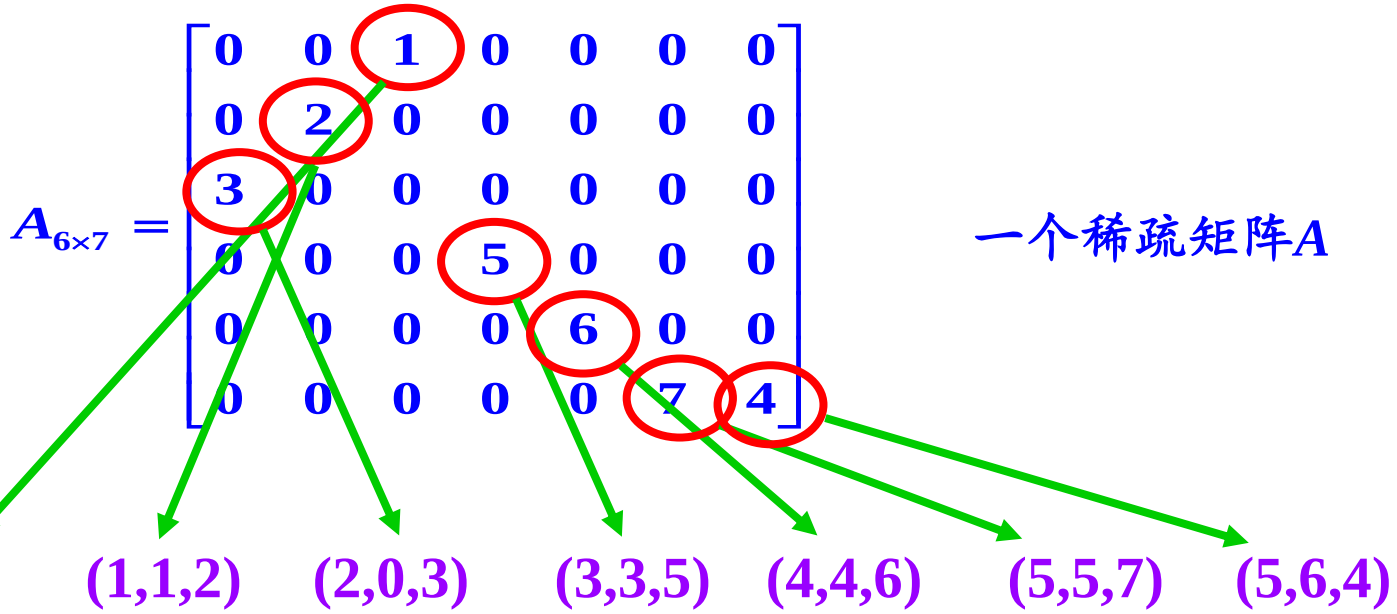
稀疏矩阵中的每一个非零元素需由一个三元组：

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

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



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

$A_{6\times7}=$

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

$A_{6\times7}=$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 7 & 8 \end{bmatrix}$$

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

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

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

插入

i	j	a_{ij}
0	2	1
1	1	2
2	0	3
3	3	5
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$0\leqslant i\leqslant m-1$，$0\leqslant j\leqslant n-1$。

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

转置

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

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++;
                    }
     }
}
```

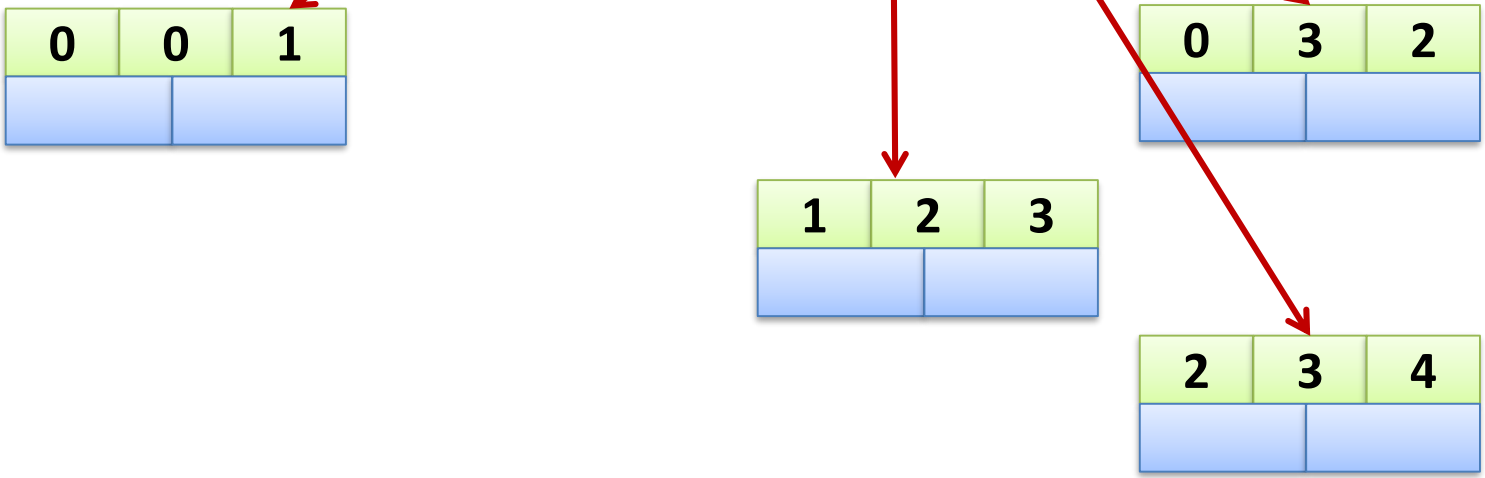
按第0、1、2、…、$n-1$ 列进行转换

若m行n列，t个非0元素，时间复杂度为$O(nt)$。

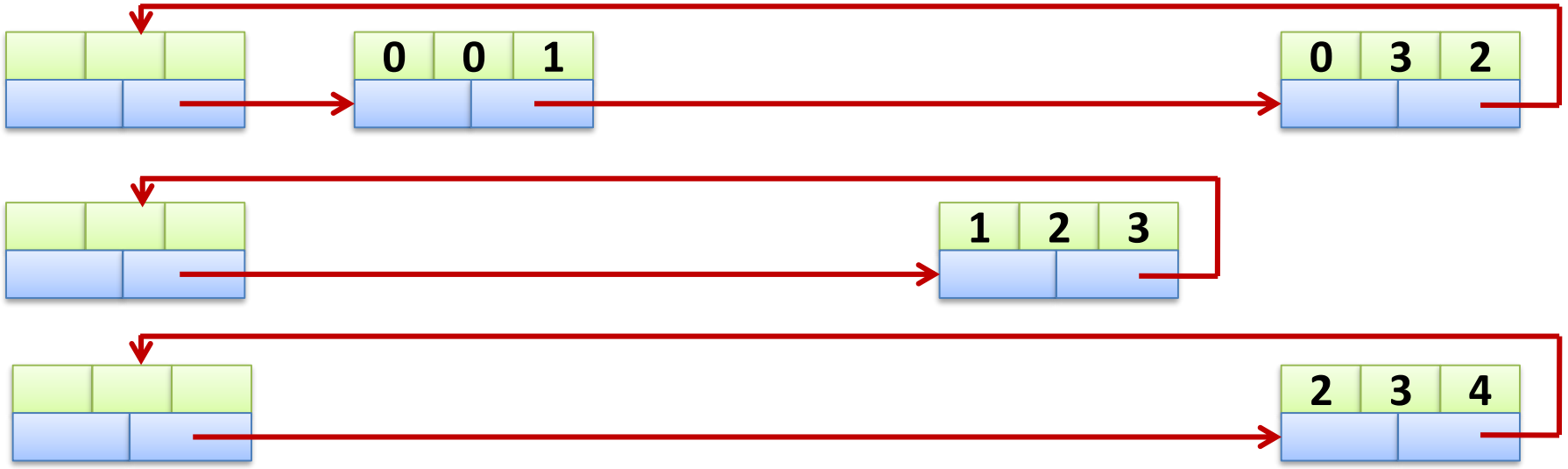
6.2.2 稀疏矩阵的十字链表表示

- 每个非零元素对应一个节点。

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

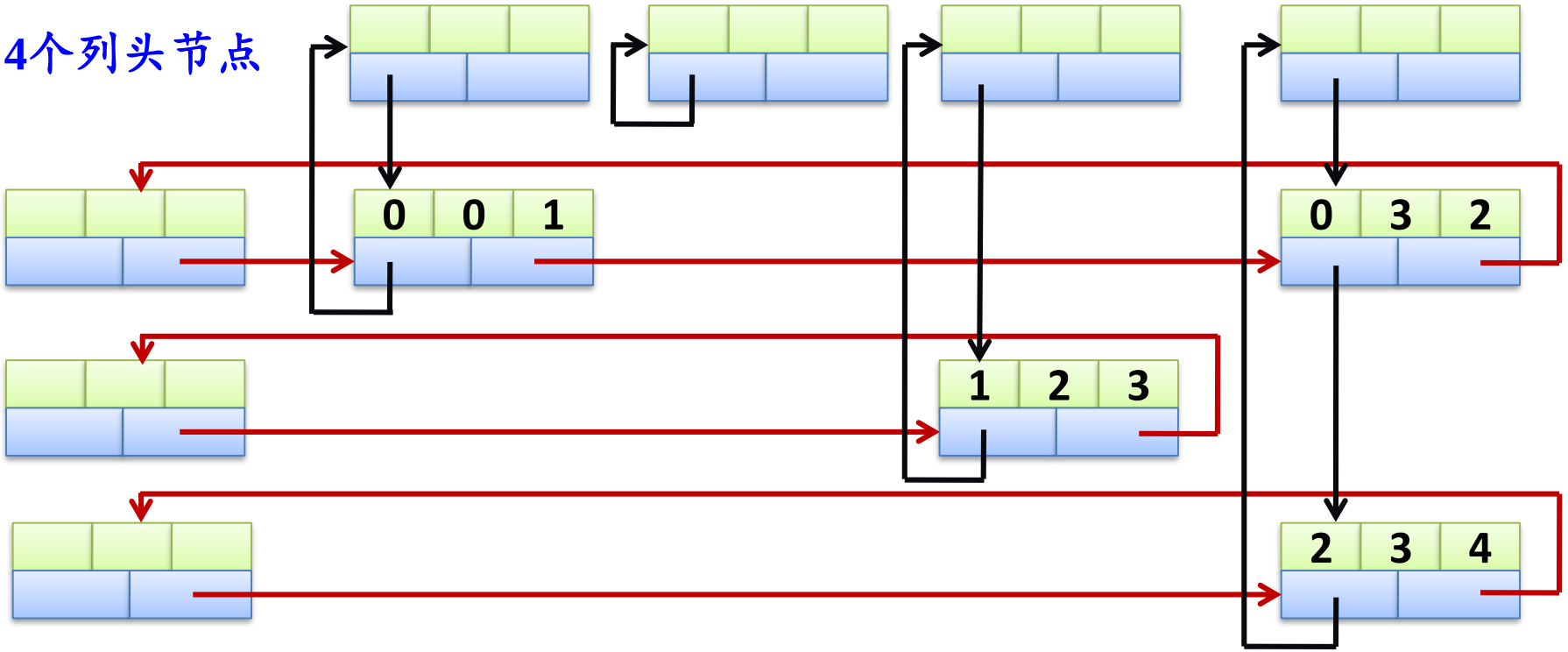


- 每行的所有节点链起来构成一个带行头节点的循环单链表。以$h[i]$（$0 \leqslant i \leqslant m-1$）作为第$i$行的头节点。

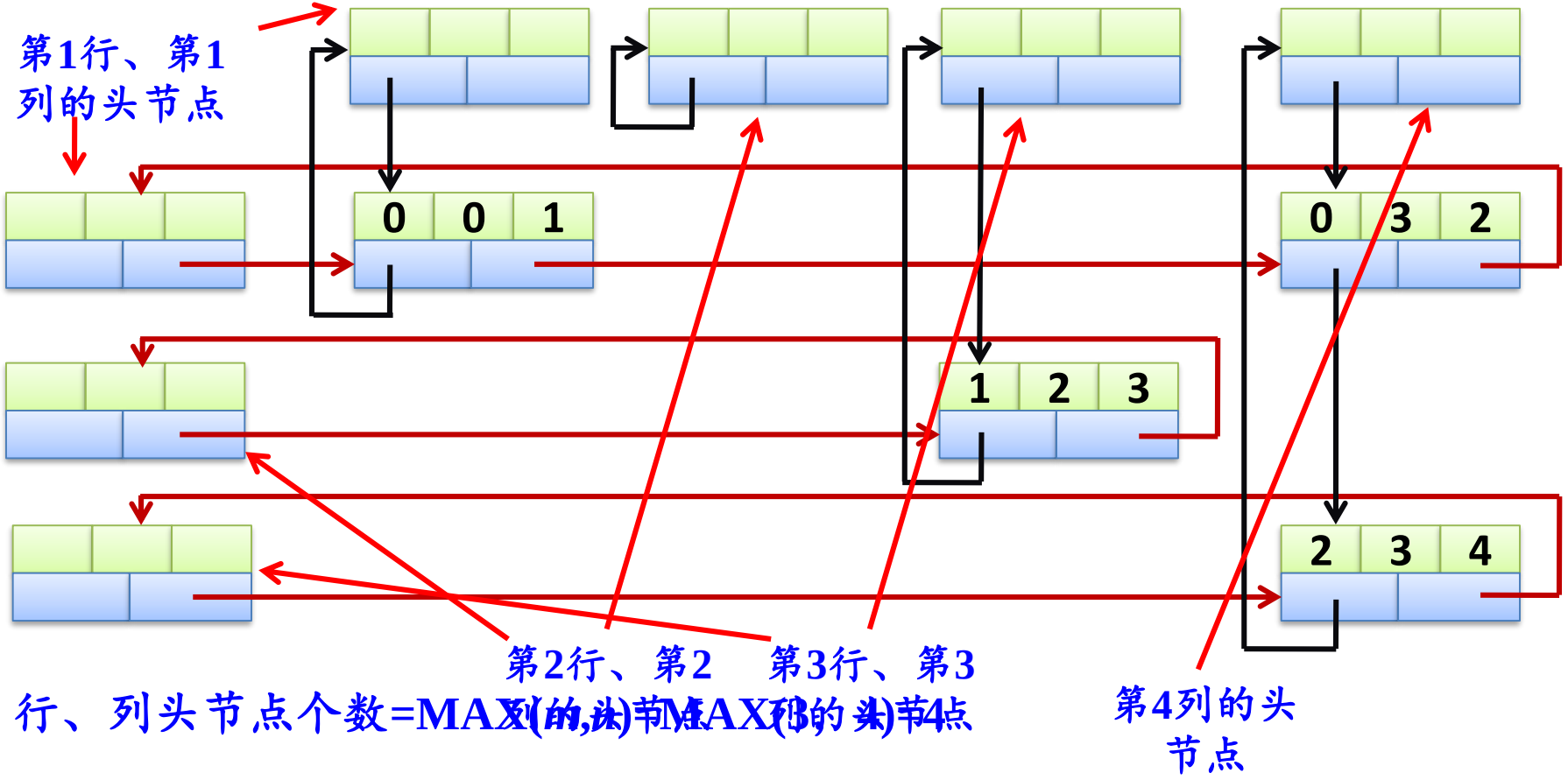


3个行头节点

- 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。

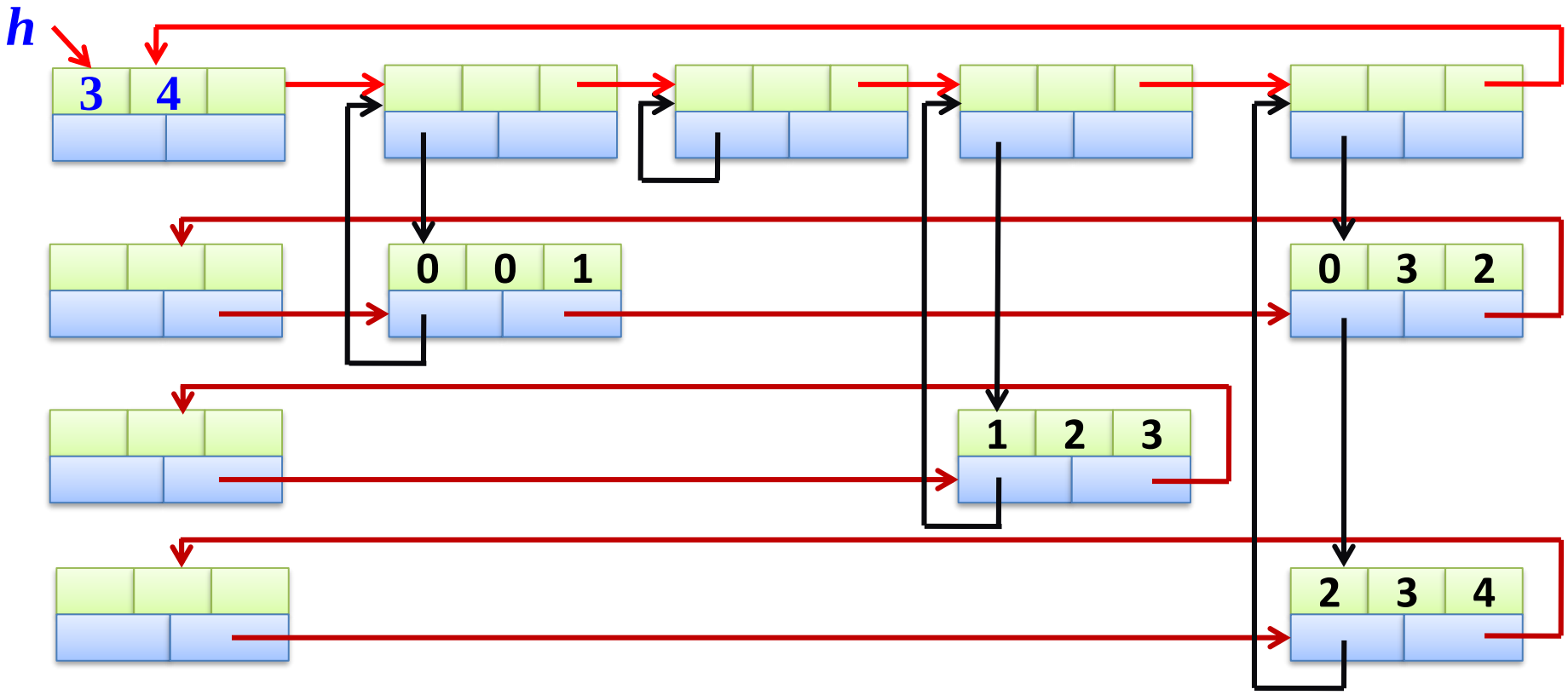

4个列头节点
0 0 1
0 3 2
1 2 3
2 3 4
3个行头节点

行、列头节点可以共享



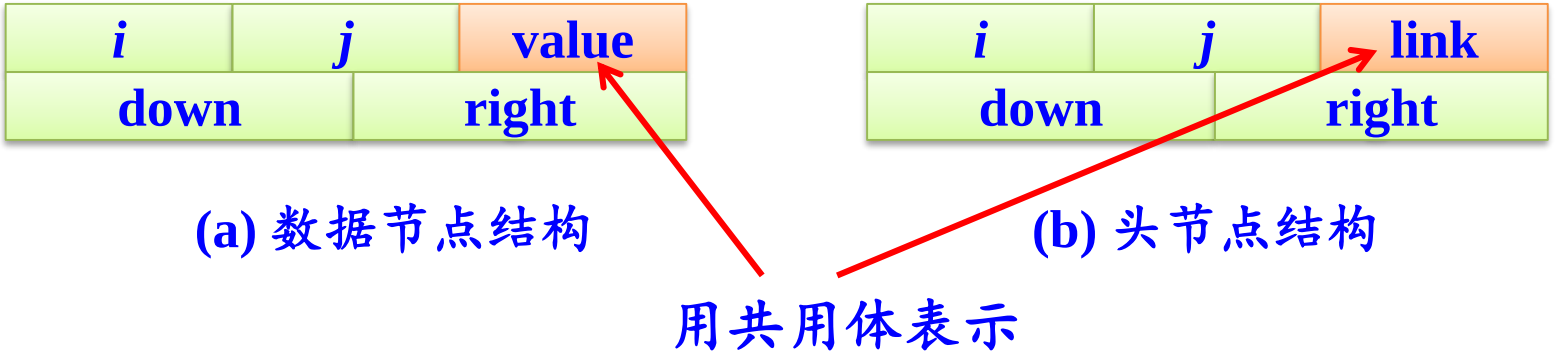
行、列头节点个数=MAX(M,N)+1

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：

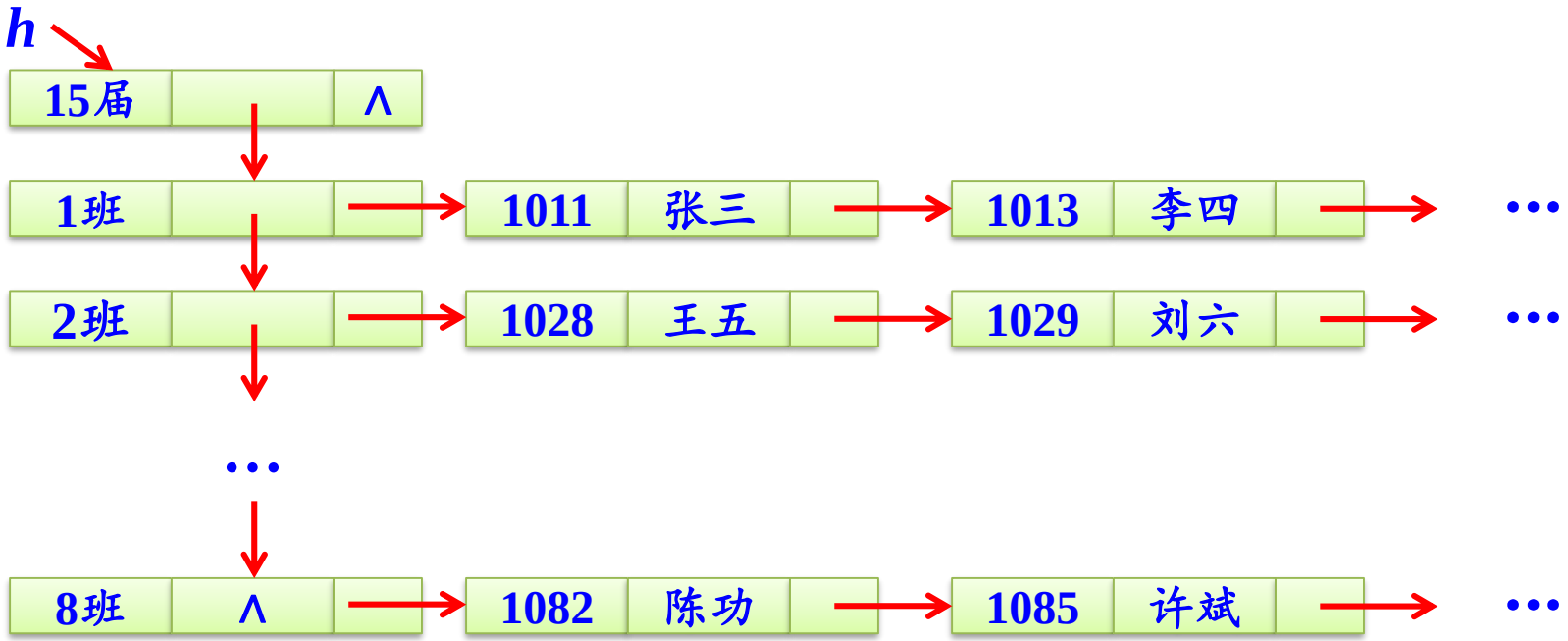


十字链表节点结构和头节点的数据结构可定义如下：

```
#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——