



信息与软件工程学院

数据结构与算法

主讲教师：陈安龙

第5章 数组和广义表

- 5.1 数组的定义
- 5.2 矩阵的一般存储
- 5.3 三角矩阵的压缩存储
- 5.4 带状矩阵的压缩存储
- 5.5 稀疏矩阵的压缩存储
- 5.6 广义表（**自学**）

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：

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

其中， a_i ($1 \leq i \leq n$) 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。

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & & & \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix} \quad \longrightarrow \quad A = [A_1, A_2, \dots, A_m]$$

\downarrow

$$\begin{aligned} A_1 &= [a_{1,1}, a_{1,2}, \dots, a_{1,n}] \\ A_2 &= [a_{2,1}, a_{2,2}, \dots, a_{2,n}] \\ &\dots\dots\dots \\ A_m &= [a_{m,1}, a_{m,2}, \dots, a_{m,n}] \end{aligned}$$

由此看出，多维数组是线性表的推广。

数组的抽象数据类型定义:

数据对象: $D = \{ a_{j_1 j_2 \dots j_n} \mid n > 0, \text{ 称为数组的维数, } j_i \text{ 是数组的第 } i \text{ 维下标, } 1 \leq j_i \leq b_i, b_i \text{ 为数组第 } i \text{ 维的长度, } a_{j_1 j_2 \dots j_n} \in \text{ElementSet} \}$

数据关系: $R = \{ R_1, R_2, \dots, R_n \}$

$R_i = \{ \langle a_{j_1 \dots j_i \dots j_n}, a_{j_1 \dots j_{i+1} \dots j_n} \rangle \mid 1 \leq j_k \leq b_k, 1 \leq k \leq n \text{ 且 } k \neq i, 1 \leq j_i \leq b_i - 1, a_{j_1 \dots j_i \dots j_n}, a_{j_1 \dots j_{i+1} \dots j_n} \in D, i = 1, \dots, n \}$

基本操作：

- (1) **InitArray(A,n,bound₁,...,bound_n)**: 若维数n和各维的长度合法，则构造相应的数组A，并返回TRUE;
- (2) **DestroyArray (A)** : 销毁数组A;
- (3) **GetValue (A,e, index₁, ...,index_n)** : 若下标合法，用e返回数组A中由index₁, ...,index_n所指定的元素的值。
- (4) **SetValue (A,e,index₁, ...,index_n)** : 若下标合法，则将数组A中由index₁, ...,index_n所指定的元素的值置为e。

注意： 这里定义的数组下标是从1开始，与C语言的数组略有不同。

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 $\text{LOC}(a_1)$ 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 $\text{LOC}(a_i)$ 就可由以下公式求出：

$$\text{LOC}(a_i) = \text{LOC}(a_1) + (i-1) * k \quad (0 \leq i \leq n)$$

数组 a : a_1 a_2 a_3 ... a_{i-1} a_i ... a_n

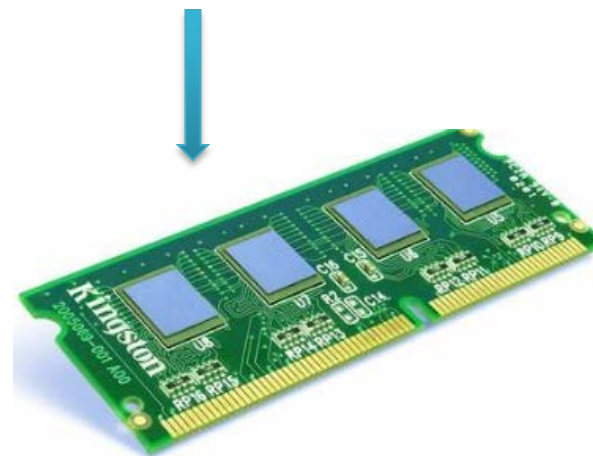
共 $i-1$ 个元素

一维数组具有**随机存储特性**：可以在 $O(1)$ 时间内找到序号为 i 的元素值。

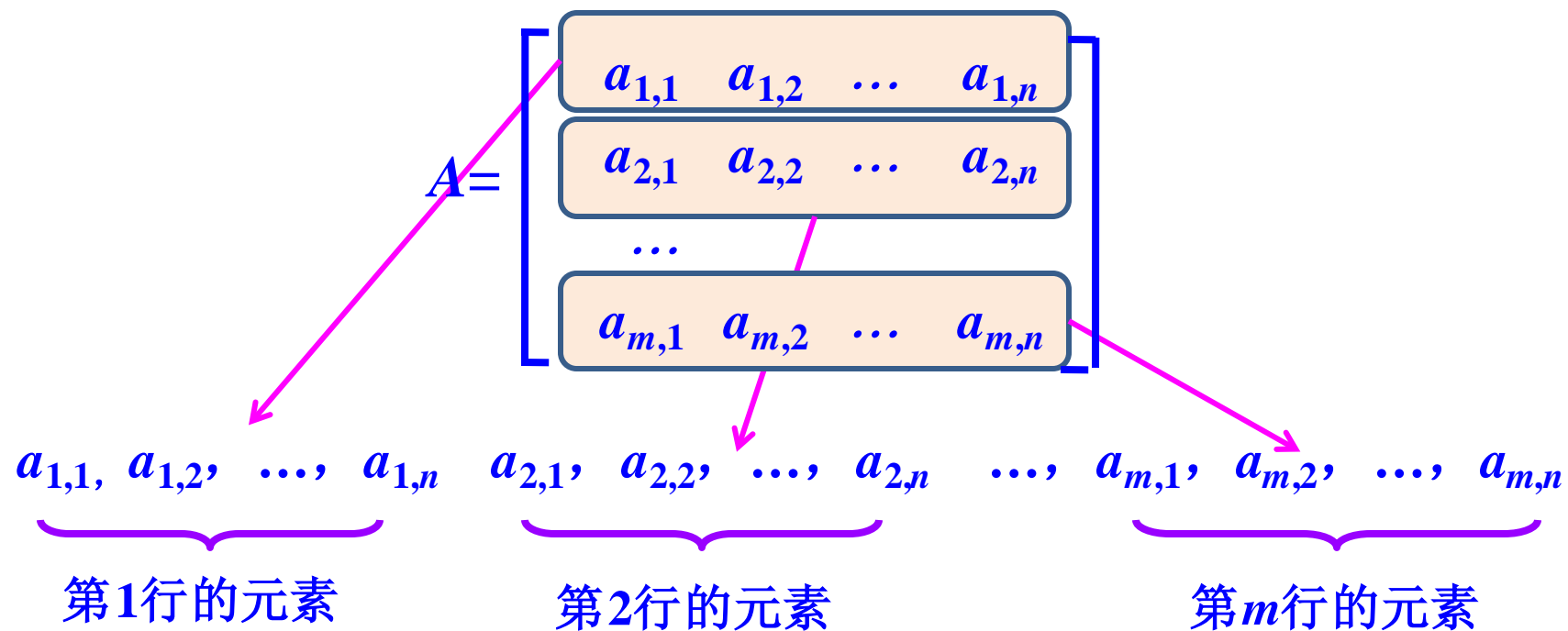
m 行 n 列的二维数组 $A_{m \times n}$ ，存储方式：

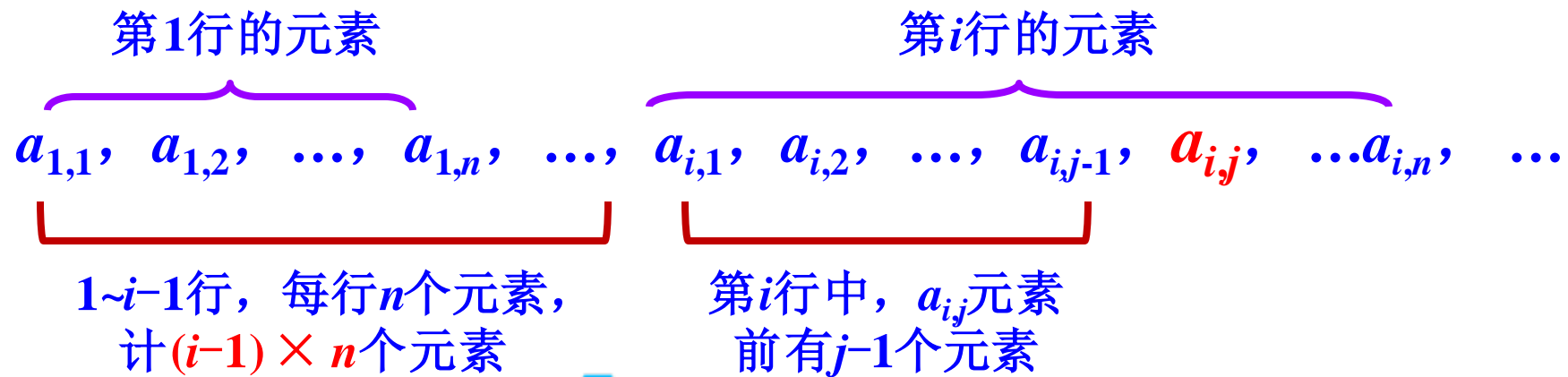
$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & & & \\ a_{m,1} & a_{m,2} & \dots & a_{m,n} \end{bmatrix}$$

- ① 以行序为主序的存储
- ② 以列序为主序的存储



① 以行序为主序的存储方式





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

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

② 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)：

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



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。



多维数组采用顺序存储时具有随机存储特性。

`int a[5][4][6]` 三维数组的行优先存储

$$m_1=5, m_2=4, m_3=6$$

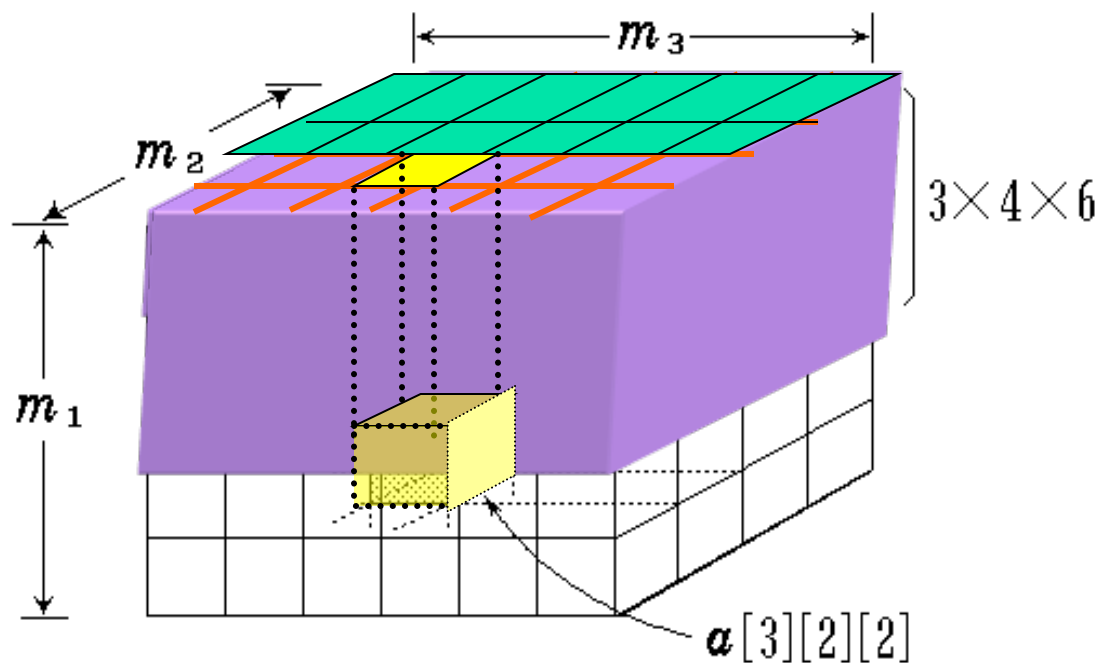
$a[i_1][i_2][i_3]$

页向量 下标 i_1

行向量 下标 i_2

列向量 下标 i_3

$a[3][2][2]$ 的存储地址:



$$\text{LOC}(3, 2, 2) = a_{111} + (2 * 4 * 6 + 1 * 6 + 1) * l$$

$$\text{LOC}(i_1, i_2, i_3) = a_{1,1,1} + \underbrace{((i_1-1) * m_2 * m_3)}_{\text{前 } i_1-1 \text{ 页总元素个数}} + \underbrace{((i_2-1) * m_3 + i_3 - 1)}_{\text{第 } i_1 \text{ 页的前 } i_2-1 \text{ 行总元素个数}} * l$$

前 i_1-1 页总
元素个数

第 i_1 页的前 i_2-1 行
总元素个数

四维数组的行优先存储

- ① 各维元素个数为 m_1, m_2, m_3, m_4
- ② 如果第1个元素为 $a_{1,1,1,1}$
- ③ 下标为 i_1, i_2, i_3, i_4 的数组元素的存储地址:

$$\text{LOC} (i_1, i_2, i_3, i_4) = \text{Loc}(a_{1,1,1,1}) + ((i_1-1)*m_2*m_3*m_4 \\ + (i_2 -1) *m_3*m_4 + (i_3 -1) *m_4 \\ + (i_4 -1)) * l$$

如果第1个元素为: a_{c_1, c_2, c_3, c_4}

$$\text{LOC} (i_1, i_2, i_3, i_4) = \text{Loc}(a_{c_1, c_2, c_3, c_4}) + ((i_1-c_1)*(m_2-c_2+1)*(m_3 -c_3+1) *(m_4 -c_4+1) \\ + (i_2 -c_2) * *(m_3 -c_3+1) *(m_4 -c_4+1) \\ + (i_3 -c_3) * (m_4 -c_4) \\ + (i_4 -c_4)) * l$$

n 维数组的行优先存储

- 各维元素个数为 $m_1, m_2, m_3, \dots, m_n$
- 如果第1个元素为 $a_{1,1,\dots,1}$
- 下标为 $i_1, i_2, i_3, \dots, i_n$ 的数组元素的存储地址:

$$\begin{aligned} \text{LOC} (i_1, i_2, \dots, i_n) &= \text{Loc}(a_{1\dots 1}) + (i_1 - 1) * m_2 * m_3 * \dots * m_n \\ &\quad + (i_2 - 1) * m_3 * m_4 * \dots * m_n + \dots + (i_{n-1} - 1) * m_n + (i_n - 1) * l \\ &= \text{Loc}(a_{1,\dots,1}) + \left(\sum_{j=2}^{n-1} \left((i_j - 1) * \prod_{k=j+1}^n m_k \right) + (i_n - 1) \right) * l \end{aligned}$$

n 维数组的行优先存储

- 各维元素个数为 $m_1, m_2, m_3, \dots, m_n$
- 如果第1个元素为 a_{c_1, c_2, \dots, c_n}
- 下标为 $i_1, i_2, i_3, \dots, i_n$ 的数组元素的存储地址:

$$\text{LOC}(i_1, i_2, \dots, i_n) = \text{Loc}(a_{c_1, \dots, c_n})$$

如果是列优先存储
则从 $i_n \rightarrow i_1$, 结构
不变, 顺序相反

$$\begin{aligned} &+ (i_1 - c_1) * (m_2 - c_2 + 1) * (m_3 - c_3 + 1) * \dots * (m_n - c_n + 1) \\ &+ (i_2 - c_2) * (m_3 - c_3 + 1) * (m_4 - c_4 + 1) * \dots * (m_n - c_n + 1) \\ &+ \dots + (i_{n-1} - c_{n-1}) * (m_n - c_n + 1) + i_n - c_n \end{aligned}$$

$$= \text{Loc}(a_{c_1, \dots, c_n}) + \left(\sum_{j=2}^{n-1} \left((i_j - c_j) * \prod_{k=j+1}^n (m_k - c_k + 1) \right) + (i_n - c_n) \right) * l$$

例1 【软考题】：一个二维数组A，行下标的范围是1到6，列下标的范围是0到7，每个数组元素用相邻的6个字节存储，存储器按字节编址。那么，这个数组的体积是288个字节。

答： $\text{Volume}=m*n*L=(6-1+1)*(7-0+1)*6=48*6=288$

例2：已知二维数组 $A_{m,m}$ 按行存储的元素地址公式是：

$$\text{Loc}(a_{ij})=\text{Loc}(a_{11})+[(i-1)*m+(j-1)]*L$$

按列存储的公式是？

$$\text{Loc}(a_{ij})=\text{Loc}(a_{11})+[(j-1)*m+(i-1)]*L \quad (\text{尽管是方阵，但公式仍不同})$$

例3：【00年计算机系考研题】设数组a[1...60, 1...70]的基地址为2048，每个元素占2个存储单元，若以列序为主序顺序存储，则元素a[32,58]的存储地址为8950。

答： 请注意审题！ 利用列优先通式：

$$LOC(a_{ij}) = LOC(a_{1,1}) + [(j-1)*m + (i-1)]*L$$

$$\text{得： } LOC(a_{32,58}) = \underline{2048} + [\underline{(58-1)*60} + \underline{32-1}] * 2 = 8950$$

6.1.3 特殊矩阵的压缩存储

特殊矩阵的主要形式有：

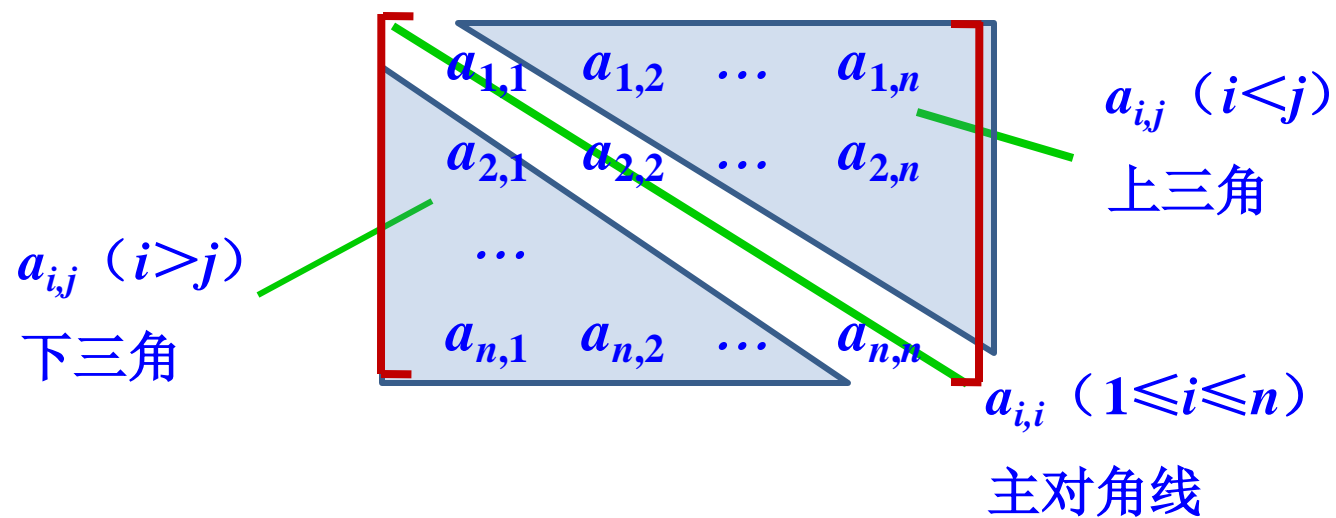
- (1) 对称矩阵
- (2) 上三角矩阵 / 下三角矩阵
- (3) 对角矩阵

它们都是方阵，即行数和列数相同。

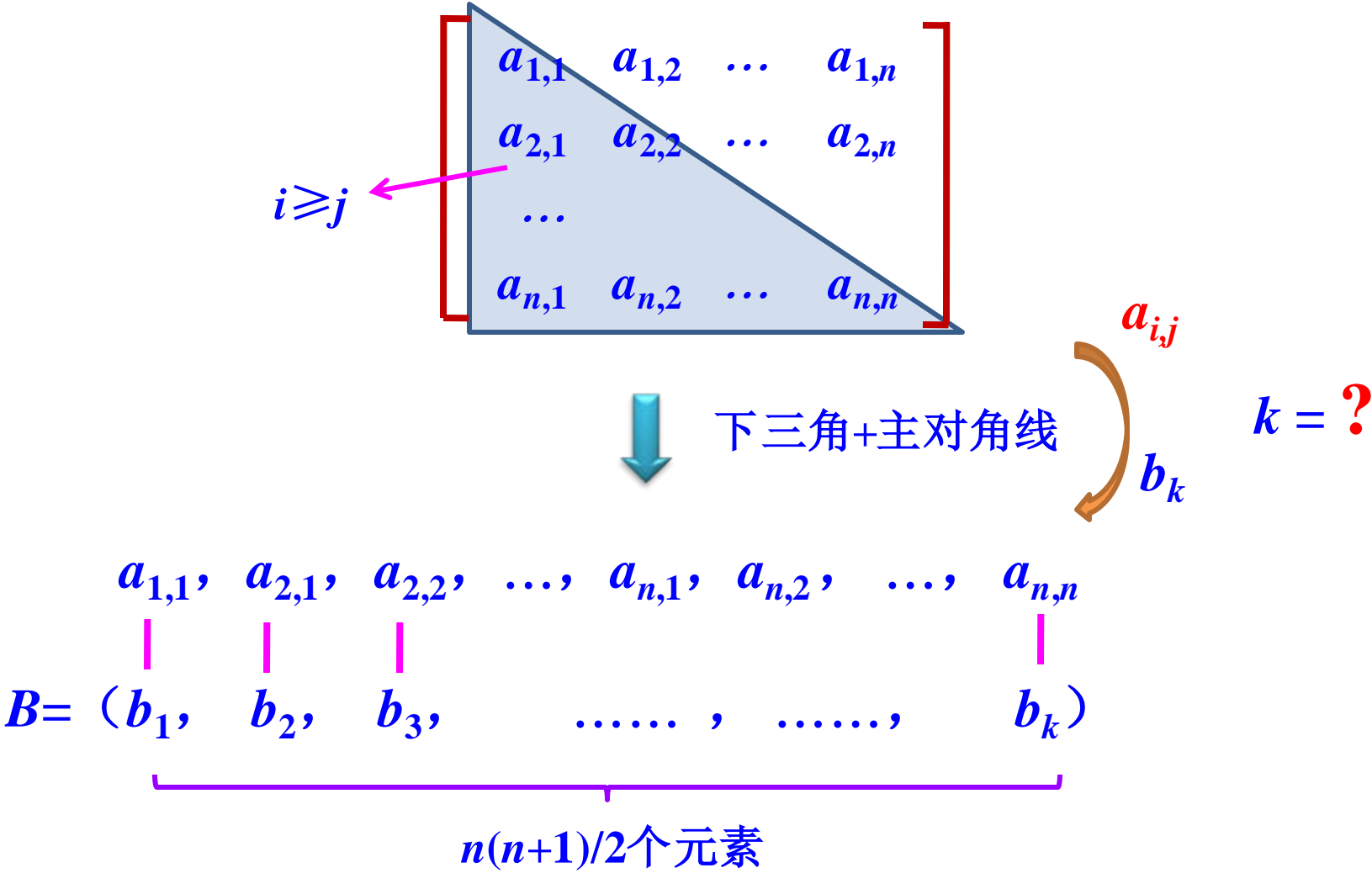
1

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 $a_{i,j}=a_{j,i}$ ($1 \leq i, j \leq n$)，则称其为 n 阶对称矩阵。



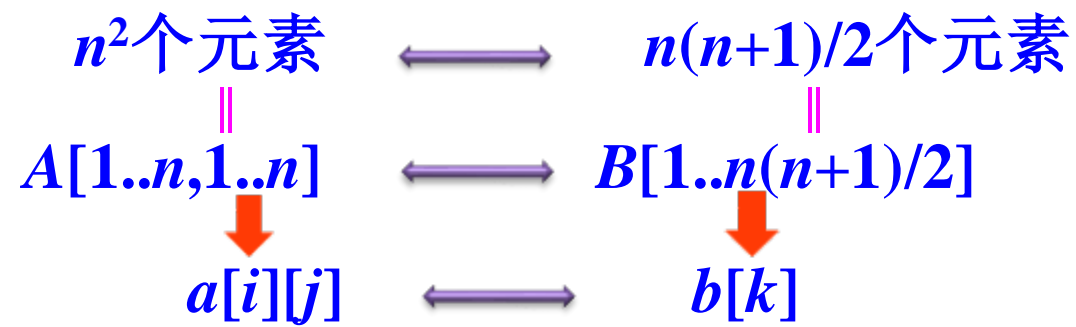
以行序为主序存储其下三角+主对角线的元素。



$$B=(\underbrace{a_{1,1}}_{\substack{1\text{个} \\ \text{元素}}}, \underbrace{a_{2,1}, a_{2,2}}_{\substack{2\text{个} \\ \text{元素}}}, \dots, \underbrace{a_{i-1,1}, \dots, a_{i-1,i-1}}_{i-1\text{个元素}}, \underbrace{a_{i,1}, \dots, a_{i,j-1}}_{j-1\text{个元素}}, \underbrace{a_{i,j}}_{\substack{\updownarrow \\ b_k}}, \dots, a_{n-1,n-1})$$

共计 $i(i-1)/2 + j - 1$ 个元素

$$k = \begin{cases} \frac{i(i-1)}{2} + j & \text{当 } i \geq j \text{ 时 (下三角+主对角线的元素)} \\ \frac{j(j-1)}{2} + i & \text{当 } i < j \text{ 时 } (a_{i,j} = a_{j,i}) \end{cases}$$



$$k = \begin{cases} \frac{i(i-1)}{2} + j & \text{当 } i \geq j \text{ 时} \\ \frac{j(j-1)}{2} + i & \text{当 } i < j \text{ 时 } (a_{ij} = a_{ji}) \end{cases}$$

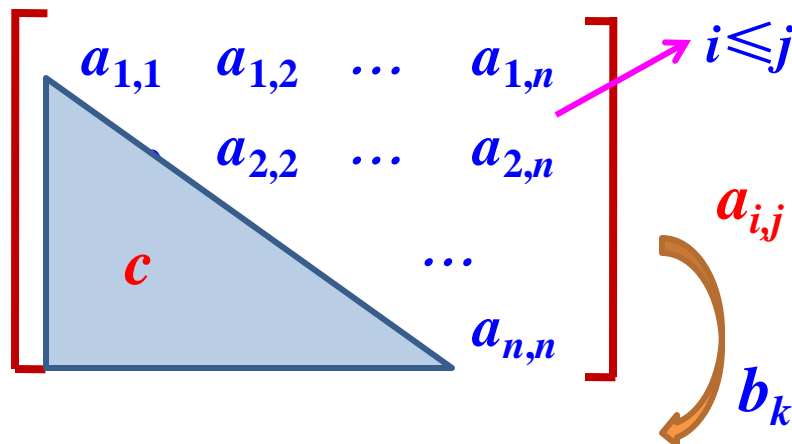
对于对称矩阵A，采用一维数组B存储，并提供A的所有运算。

$$Loc(i, j) = \begin{cases} Loc[1, 1] + l * [i(i-1)/2 + j - 1] & \text{当 } i \geq j \text{ 时 (下三角+主对角线的元素)} \\ Loc[1, 1] + l * [j(j-1) + i - 1] & \text{当 } i < j \text{ 时 } (a_{ij} = a_{ji}) \end{cases}$$

2

三角矩阵的压缩存储

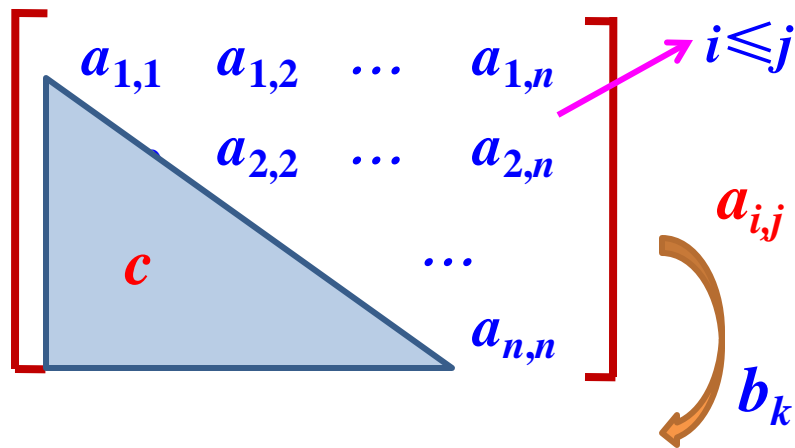
● 上三角矩阵:



按行存储

$$B = (a_{1,1}, a_{1,2}, \dots, a_{1,n}, a_{2,2}, \dots, a_{2,n}, \dots, a_{i-1,i-1}, \dots, a_{i-1,n}, a_{i,i}, \dots, a_{i,j-1}, a_{i,j}, \dots)$$

● 上三角矩阵:



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

共计 $(i-1)*(2n-i+1)/2 + j-i$ 个元素

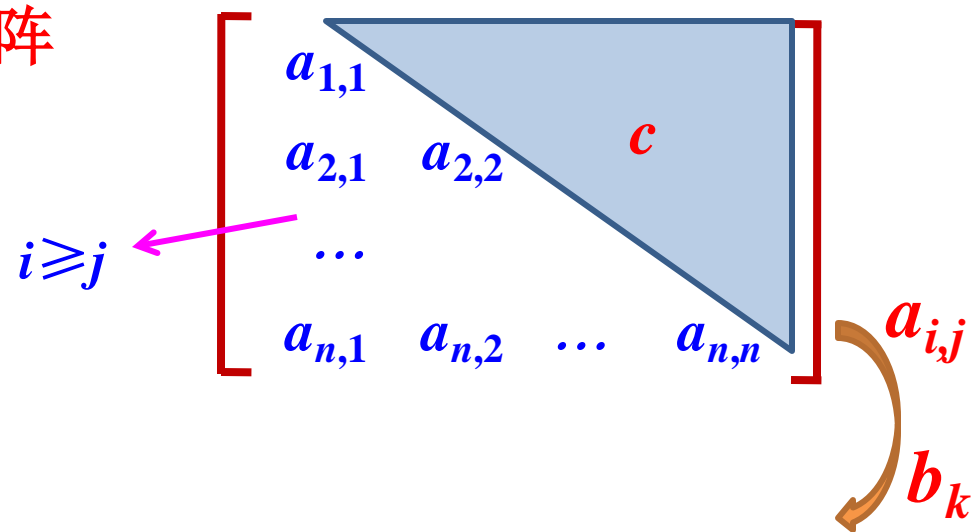
$$k = \begin{cases} \frac{(i-1)*(2n-i+1)}{2} + j - i & \text{当 } i \leq j \text{ 时} \\ \frac{n(n+1)}{2} + 1 & \text{当 } i > j \text{ 时} \end{cases}$$

前面的元素个数

存放常量 c

剩下的用一个位置存放, 它的前面固定有已存放元素的个数

● 下三角矩阵



$$B = (a_{1,1}, a_{2,1}, a_{2,2}, \dots, a_{n-1,1}, \dots, a_{n-1,n-1}, a_{n,1}, \dots, a_{n,n})$$



$$k = \begin{cases} \frac{i(i-1)}{2} + j & \text{当 } i \geq j \text{ 时} \\ \frac{n(n+1)}{2} + 1 & \text{当 } i < j \text{ 时} \end{cases}$$

存放一个常量 c

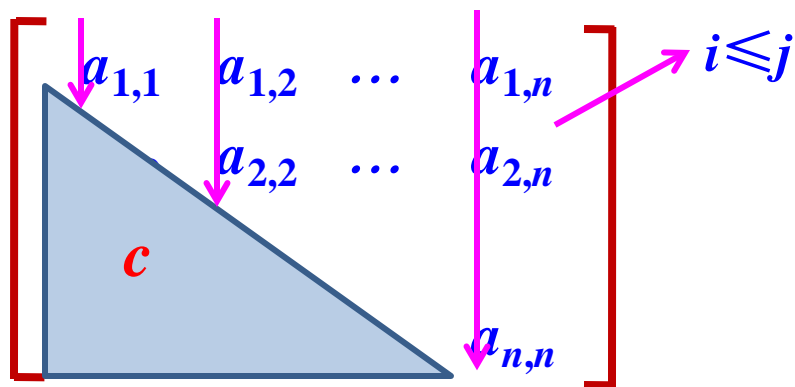
【例（补充）】若将 n 阶上三角矩阵 A 按列优先顺序压缩存放在一维数组 $B[1..n(n+1)/2]$ 中， A 中第一个非零元素 $a_{1,1}$ 存于 B 数组的 b_1 中，则应存放到 b_k 中的非零元素 a_{ij} ($i \leq j$) 的下标 i 、 j 与 k 的对应关系是_____。

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~ $j-1$ 列的元素个数: $j(j-1)/2$

第 j 列 a_{ij} 之前的元素个数: $i-1$

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

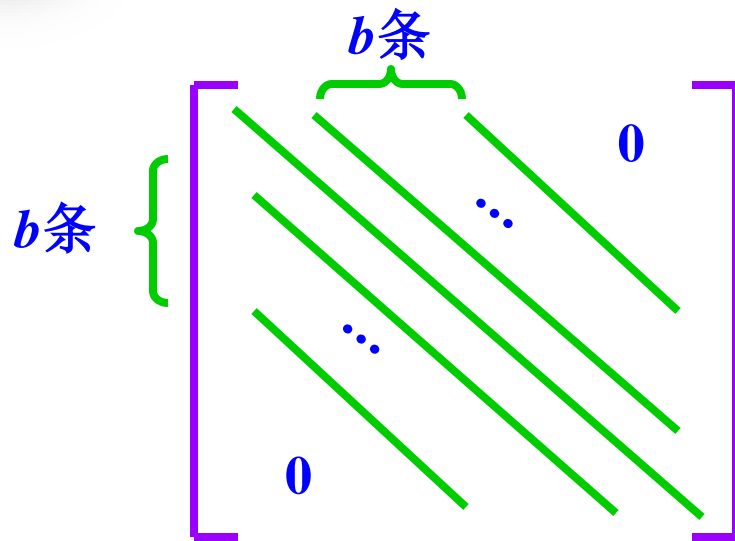
↓ 加上自身所在的位置

● 按行还是按列

● 初始下标从0还是从1开始

3

带状矩阵（对角矩阵）的压缩存储



半带宽为 b 的对角矩阵

带状矩阵，以三对角矩阵为例

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \dots\dots\dots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & \dots\dots\dots & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & \dots\dots\dots & 0 \\ \dots\dots\dots & \dots\dots\dots & \dots\dots\dots & \dots\dots\dots & \dots\dots\dots & \dots\dots\dots & \dots\dots\dots \\ 0 & 0 & \dots\dots\dots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \dots\dots\dots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

特点：

在 $\begin{cases} \text{当 } i=1 \text{ 时, } j=1,2; \\ \text{当 } 1<i<n \text{ 时, } j=i-1,i,i+1 \\ \text{当 } i=n \text{ 时, } j=n-1,n; \end{cases}$ 条件下, a_{ij} 非零, 其他元素均为零。

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

$$\begin{bmatrix} a_{1,1} & a_{1,2} & 0 & \dots & \dots & \dots & 0 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & \dots & \dots & 0 \\ 0 & a_{3,2} & a_{3,3} & a_{3,4} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & \dots & a_{n-1,n-2} & a_{n-1,n-1} & a_{n-1,n} \\ 0 & 0 & \dots & \dots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

其以行为主序的压缩后 k 计算公式如: $k = 2(i-1) + j$

三对角带状矩阵的压缩存储，以行序为主序进行存储，并且只存储非零元素。其方法为：

1. 确定存储该矩阵所需的一维向量空间的大小

从三对角带状矩阵中可看出：除第一行和最后一行只有两个元素外，其余各行均有3个非零元素。由此可得到一维向量所需的空间大小为： **$3n-2$** 。

2. 确定非零元素在一维数组空间中的位置

$$\text{LOC}[i, j] = \text{LOC}[1, 1] + (3 \times (i-1) - 1 + j - i + 1) * \text{size} = \text{LOC}[1, 1] + (2(i-1) + j - 1) * \text{size}$$



思考题：

特殊矩阵为什么采用压缩存储，需要解决什么问题？

5.2 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 $s \ll t$ 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。

定性的描述



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**。

5.2.1 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。

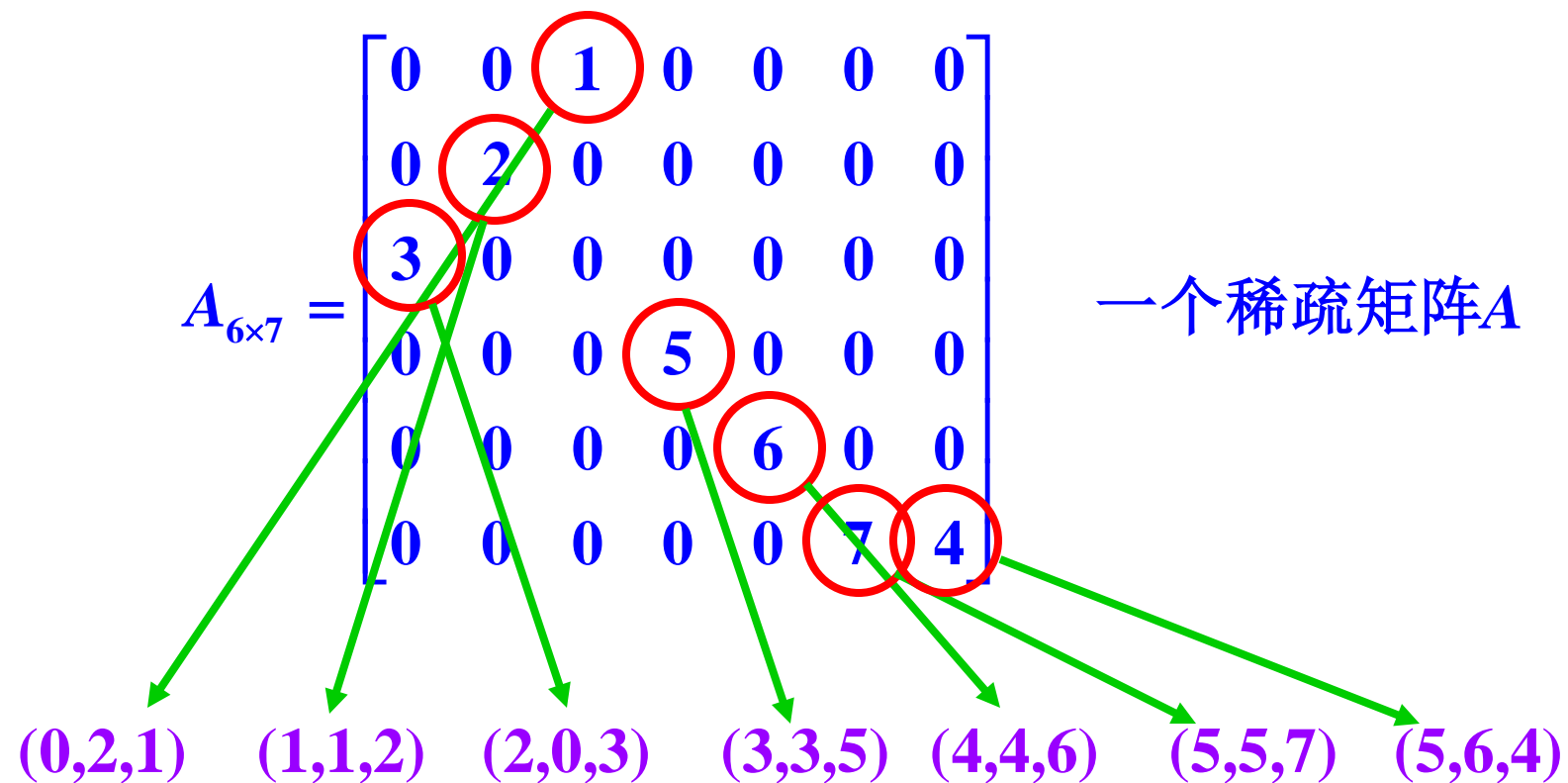
稀疏矩阵中的每一个非零元素需由一个三元组：

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

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 row;           //行号  
    int col;           //列号  
    ElemType e;        //元素值  
} Triple;              //三元组定义
```

存放一个非0元素

```
typedef struct  
{   int m;             //行数值  
    int n;             //列数值  
    int len;           //非零元素个数  
    Triple data[MaxSize+1];  
} TSMatrix;            //三元组顺序表定义
```

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。

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



t:

<i>i</i>	<i>j</i>	<i>a_{ij}</i>
1	3	1
2	2	2
3	1	3
4	4	5
5	5	6
6	6	7
6	7	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->m=M; t->n=N; t.len=0;
```

```
    for (i=1;i<=M;i++)
```

```
    {   for (j=1;j<=N;j++)
```

```
        if (A[i][j]!=0)
```

```
        {   t->data[t.len].r=i;
```

```
            t->data[t.len].c=j;
```

```
            t->data[t.len].d=A[i][j];
```

```
            t->len++;
```

```
        }
```

```
    }
```

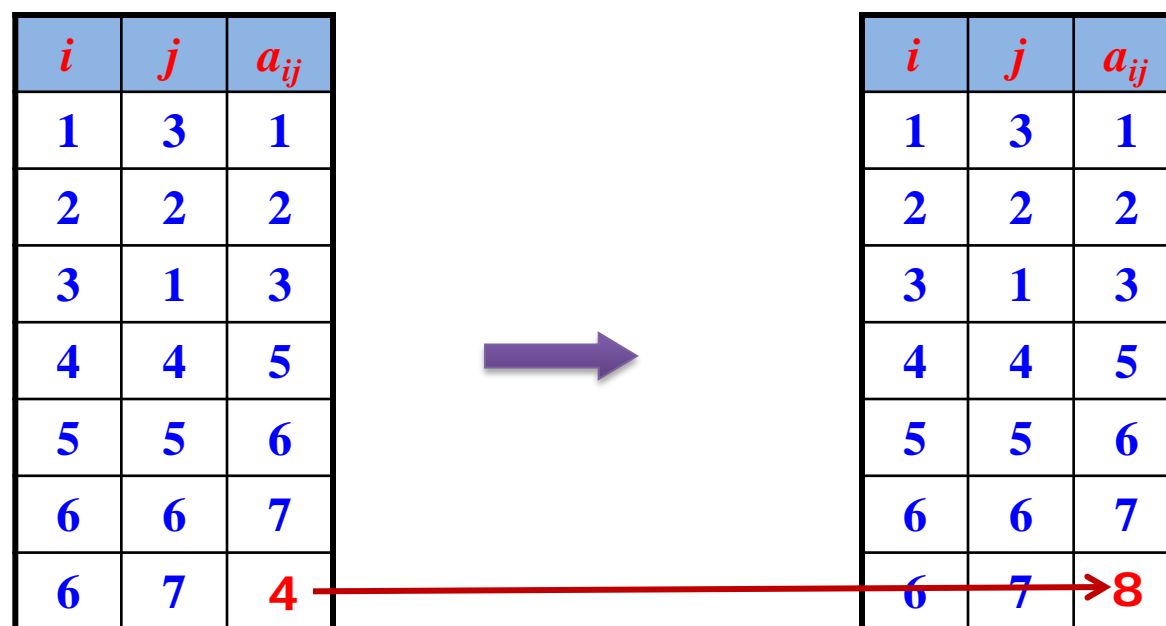
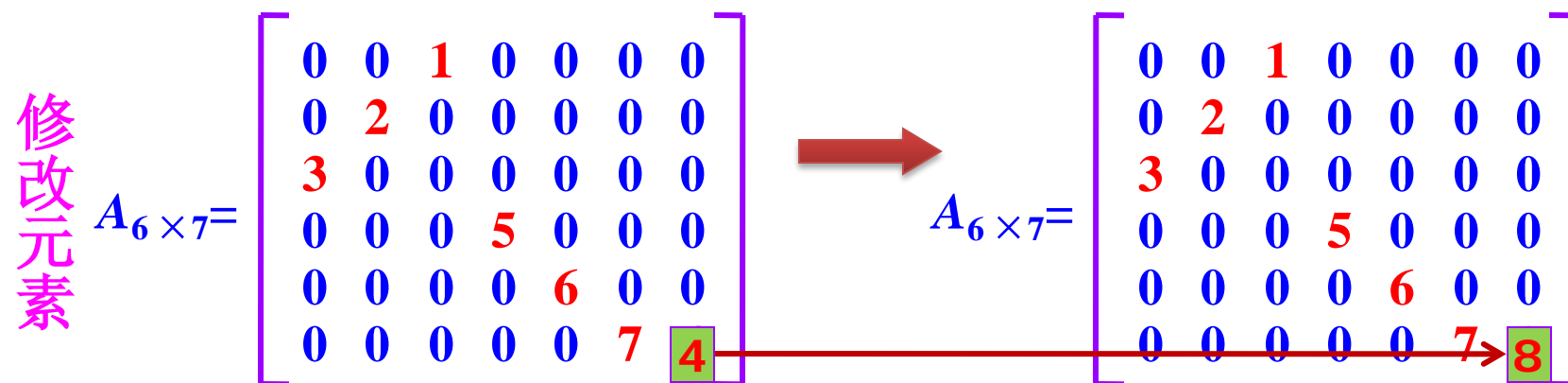
```
}
```

按行序方式扫描
所有元素

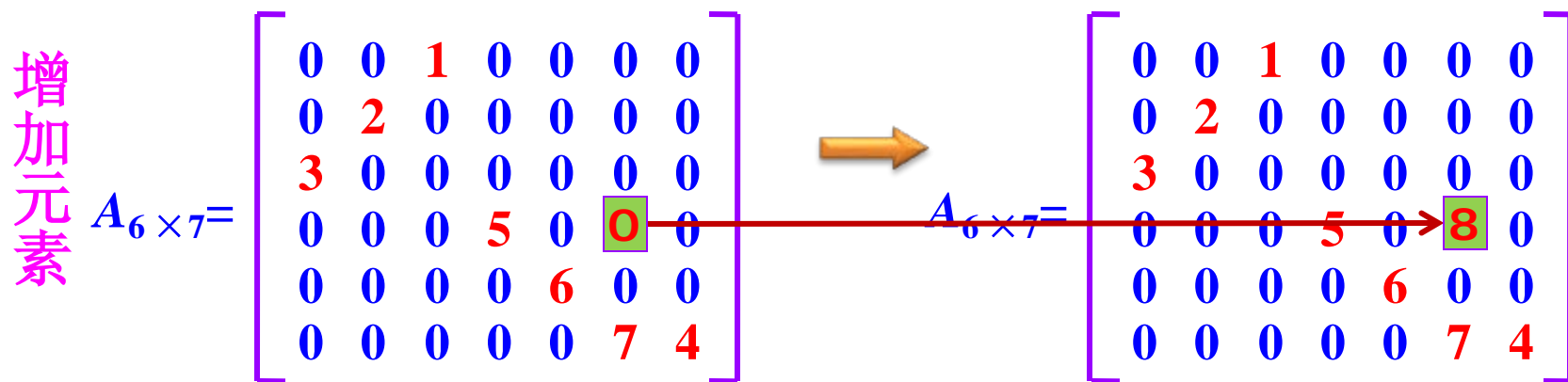
只存储非零元素

(2) 三元组元素赋值: $A[i][j]=x$

分为两种情况: ①将一个非0元素修改为另一个非0值, 如 $A[6][7]=8$ 。



②将一个0元素修改为非0值。如 $A[4][6]=8$



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



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

算法如下:

```
bool Value(TSMatrix *t, ElemType x, int i, int j)
```

```
{   int k=1, k1;
```

```
    if (i >= t->m || j >= t->n)
```

```
        return false;    //失败时返回false
```

```
    while (k <= t->len && i < t->data[k].row) k++;    //查找行
```

```
    while (k <= t->len && i == t->data[k].row && j < t->data[k].col) k++;    //查找列
```



在t中按行、列号查找

```
if (t->data[k].row==i && t->data[k].col==j) //存在这样的元素
    t->data[k].e=x;
```

修改元素

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

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=1;
```

```
    if (i>=t.m || j>=t.n)
```

```
        return false;
```

//失败时返回false

```
    while (k<=t.len && i<t.data[k].row) k++; //查找行
```

```
    while (k<=t.len && i==t.data[k].row
```

```
        && j<t.data[k].col) k++;
```

//查找列

在t中按行、
列号查找

```
    if (t.data[k].row==i && t.data[k].col==j)
```

```
        x=t.data[k].e;
```

找到了非
0的元素

```
    else
```

```
        x=0;
```

```
    return true;
```

//成功时返回true

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.len<=0) return;
    printf("\t%d\t%d\t%d\n",t.m,t.n,t.len);
    printf(" ----- \n");
    for (i=1;i<=t.len;i++)
        printf("\t%d\t%d\t%d\n", t.data[i].row,t.data[i].col, t.data[i].e);
}
```

(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 $b_{ij} = a_{ji}$ ，其中 $1 \leq i \leq m$ ， $0 \leq j \leq n$ 。


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

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

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

一种非高效的算法：按第1、2、 \dots 、 n 列进行转换

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



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


矩阵转置


```

void TransTSMatrix(TSMatrix t, TSMatrix *tb)
{
    int i, j=1, v;                                //j为tb.data的下标
    tb->m=t.n; tb->n=t.m; tb->len=t.len;
    if (t.len!=0)                                //当存在非零元素时执行转置
    {
        for (v=1; v<=t.n; v++)                  //tb->data[j]中记录以列序排列
        {                                       //i为t.data的下标
            for (i=1; i<=t.len; i++)
            {
                if (t.data[i].c==v)
                {
                    tb->data[j].row=t.data[i].col;
                    tb->data[j].col=t.data[i].row;
                    tb->data[j].e=t.data[i].e;
                    j++;
                }
            }
        }
    }
}

```

按第1、2、 \dots 、 n 列进行转换

m 行 n 列， t 个非0元素，时间复杂度为 $O(nt)$ 。

❖ 快速高效的转置算法

实现： 设两个数组

`num[col]`: 表示矩阵M中第col列中非零元个数

`position[col]`: 指示M中第col列第一个非零元在T中位置

显然有：

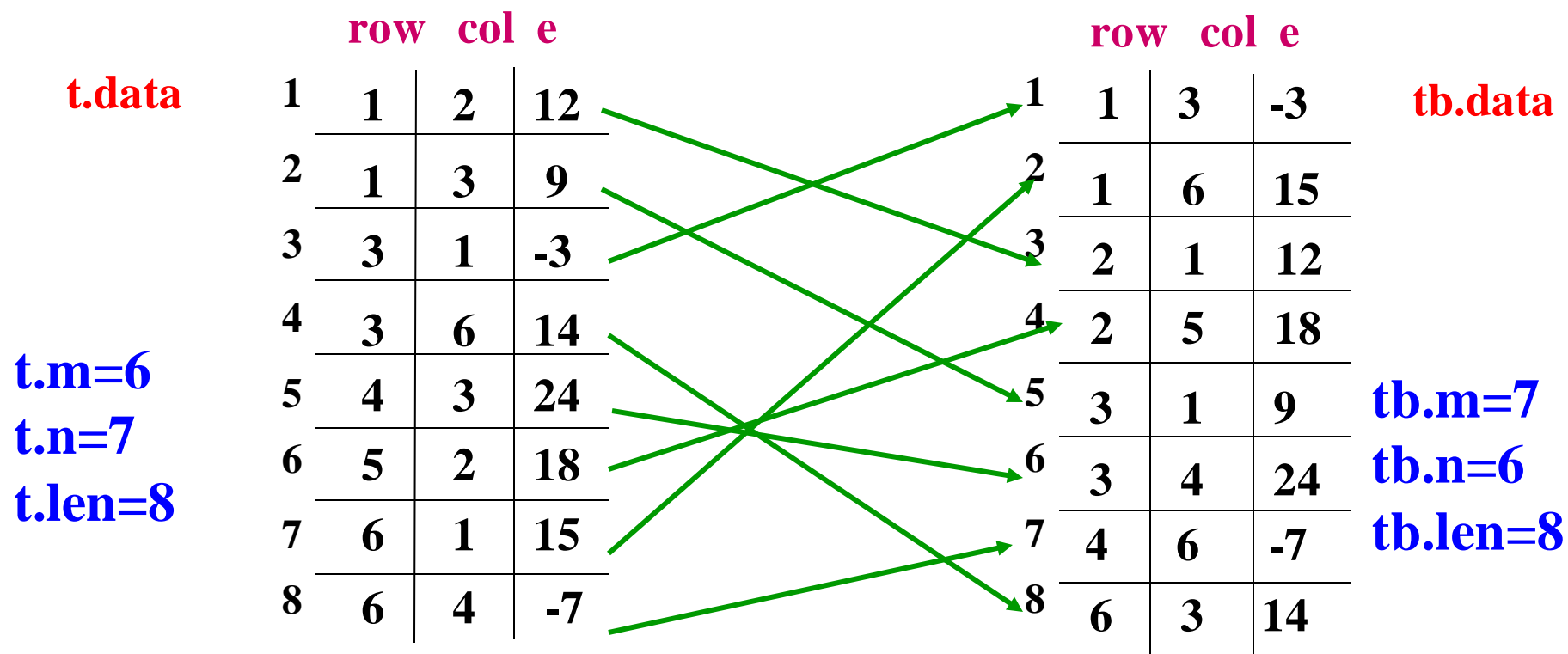
`position[0]=0;`

`position[col]= position[col-1]+num[col-1];` $(2 \leq col \leq t.n)$

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

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

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



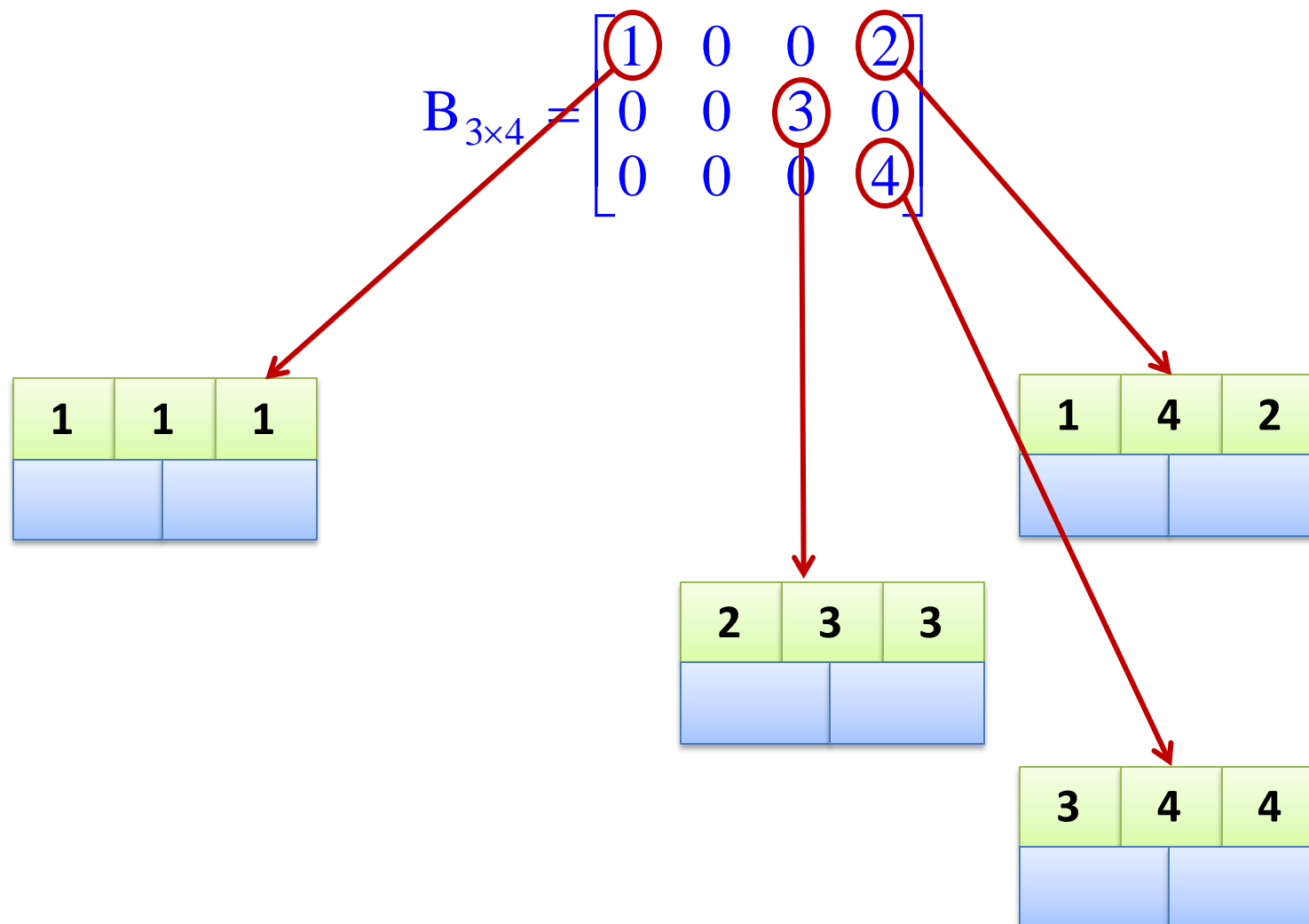
```

void FastTransTSMatrix(TSMatrix t, TSMatrix *tb)
{
    tb->m=t.n; tb->n=t.m; tb->len=t.len;
    if(tb->len)
    {
        for(int col=0;col<t.n;col++) num[col]=0;           //清零
        for(int k=1;k<=t.len;k++) ++num[t.data[k].c];      //求每列非零元素个数
        position[0]=0;
        for(int col = 2;col<=t.n;col++)
            position[col] = position[col-1]+num[col-1]; //每列第一个非零元素在tb中的序号
        for(int p = 0;p<t.tu;p++)
        {
            col=t.data[p].col;
            q=position[col]; //第col列在tb中的位置下标
            tb->data[q].row= t.data[p].col;
            tb->data[q].col= t.data[p].row;
            tb->data[q].e= t.data[p].e;
            ++position[col]; //下一个第col列非零元素的位置
        }
    }
}

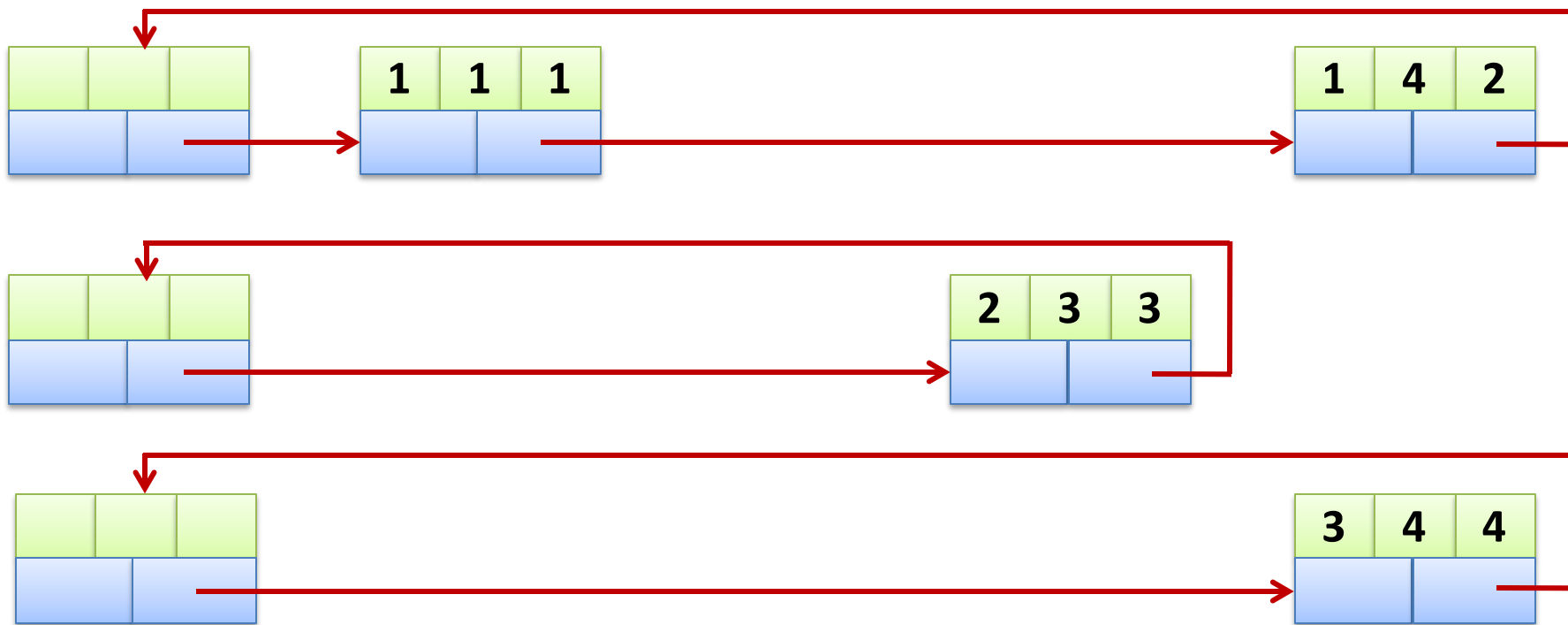
```

6.2.2 稀疏矩阵的十字链表表示

- 每个非零元素对应一个结点。

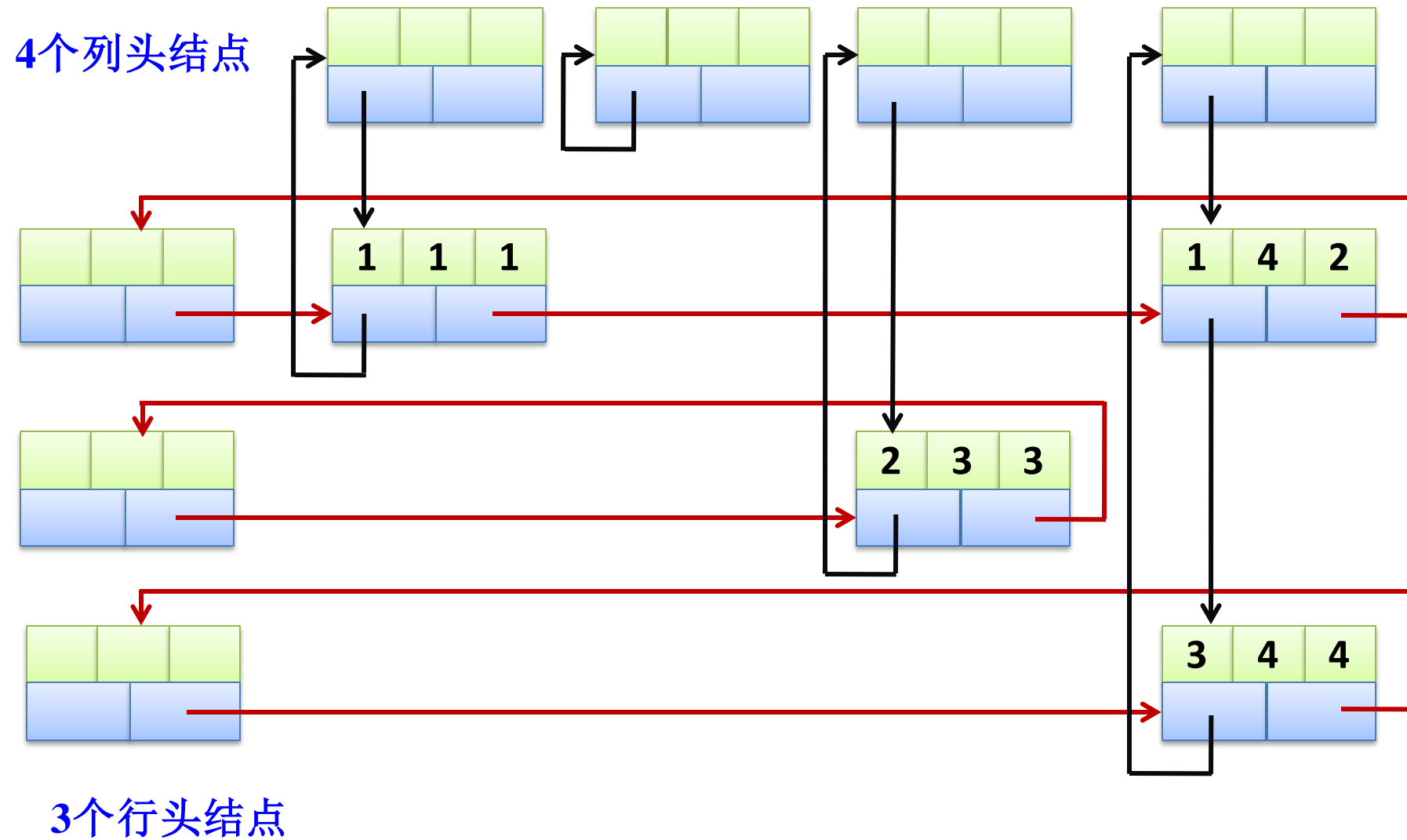


- 每行的所有结点链起来构成一个带行头结点的循环单链表。以 $h[i]$ ($0 \leq i \leq m-1$) 作为第 i 行的头结点。

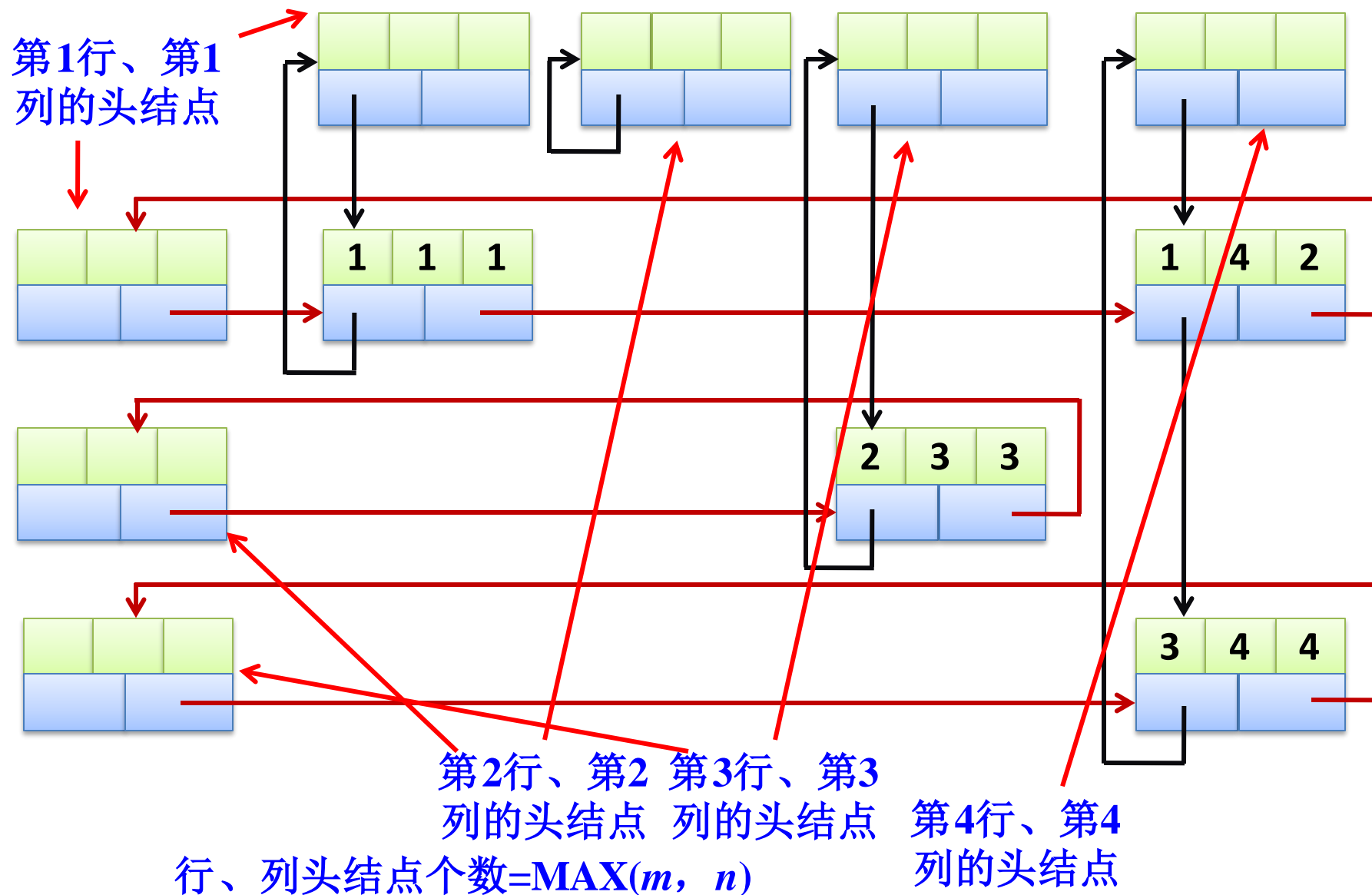


3个行头结点

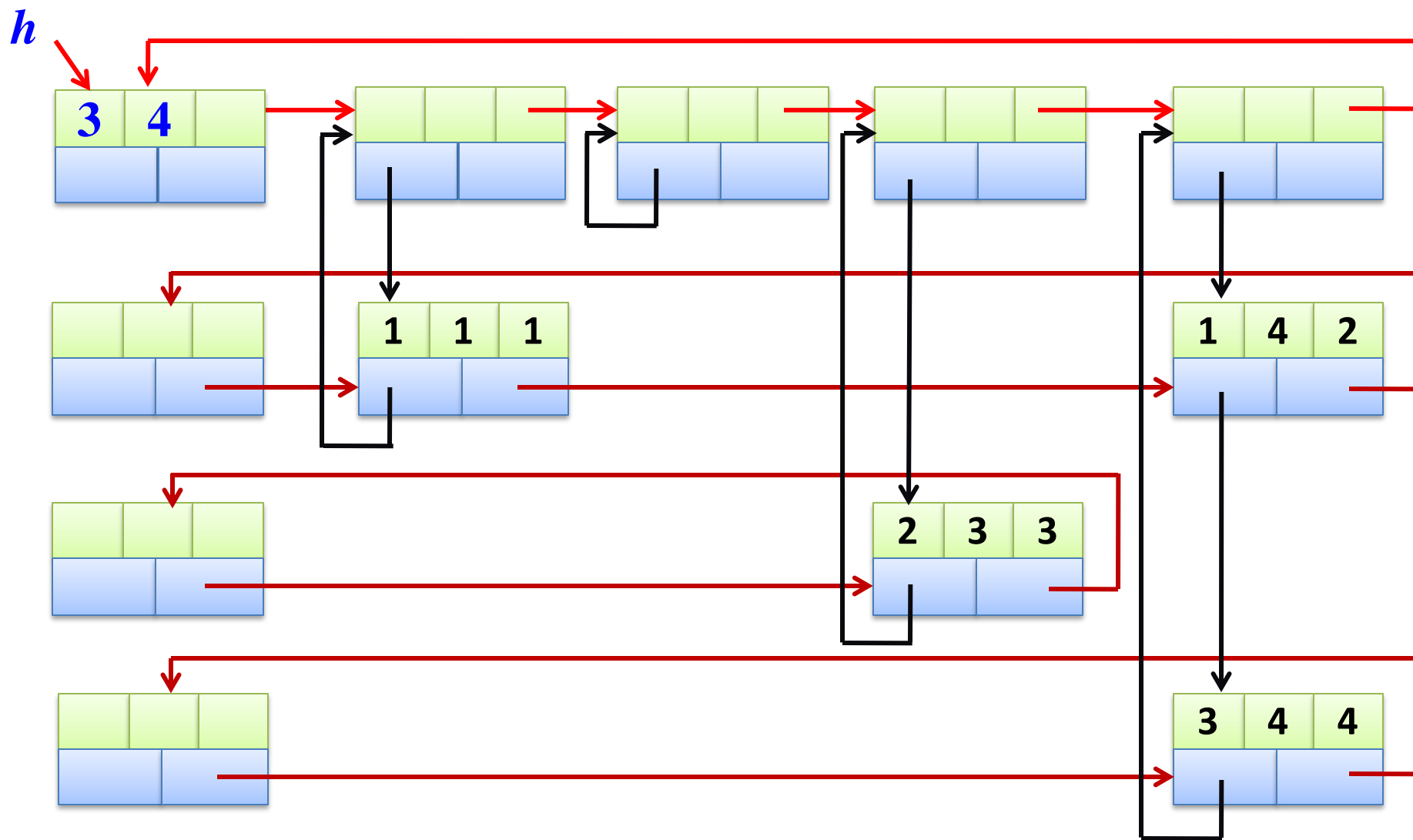
- 每列的所有结点链起来构成一个带列头结点的循环单链表。
以 $h[i]$ ($0 \leq i \leq m-1$) 作为第 i 列的头结点。



行、列头结点可以共享

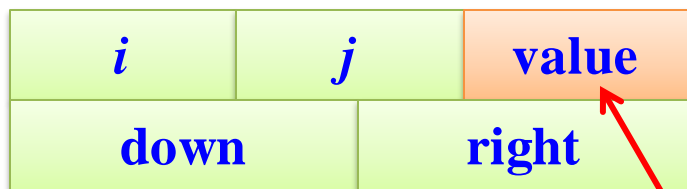


增加一个总头结点，并把所有行、列头结点链起来构成一个循环单链表

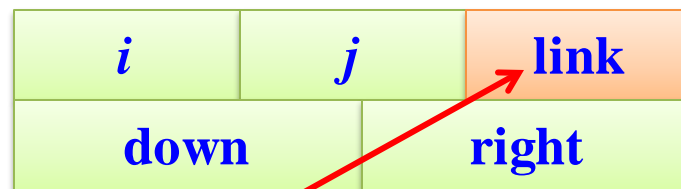


总的头结点个数= $\text{MAX}(m,n)+1$

为了统一，设计结点类型如下：



(a) 数据结点结构



(b) 头结点结构

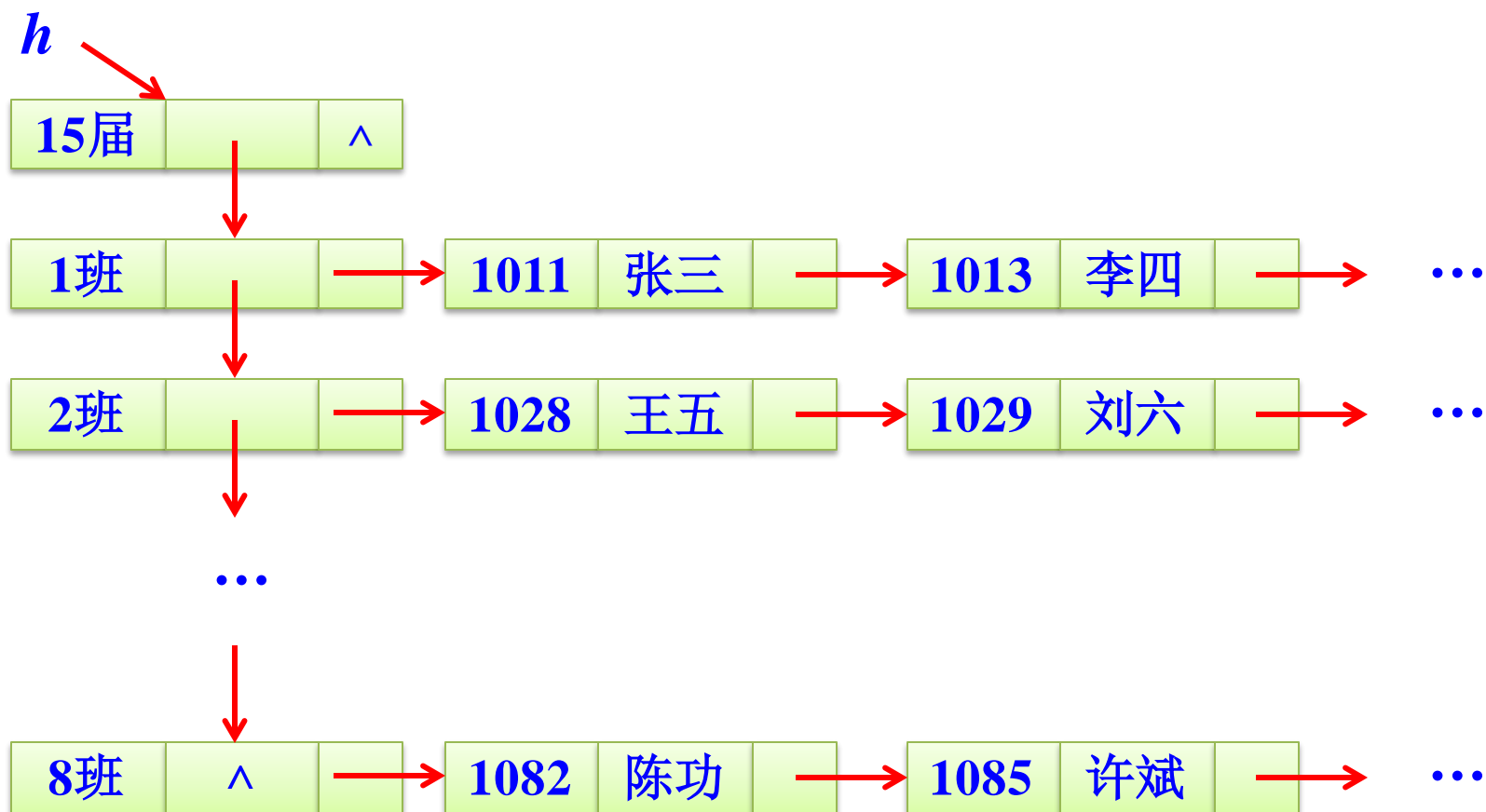
用共用体表示

十字链表结点结构和头结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; //十字链表结点类型声明
```

有关算法不做介绍。

【例（补充）】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

本章结束