第五章 数组和广义表

本质上是非线性结构。扩展的线性表：表中的数据元素本身也是一个数据结构。

5.1 数组和线性表的关系以及数组的运算

5.2 数组的顺序存储结构

5.3 矩阵的压缩存储

5.4 广义表的定义和表示方法

5.5 广义表的存储结构

5.6 广义表的递归算法

5.1 数组和线性表的关系以及数组的运算

任何数组A都可以看作一个线性表

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

二维数组m×n时，

a_i是数组中第i行所有元素，0≤i≤m-1，表中每一个元素是一个一维数组；

$$A_{m\times n}=\begin{pmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{pmatrix}$$

三维数组时，

表中每一个元素是一个二维数组；

n维数组时，

表中每一个元素是一个(n-1)维数组

数组与线性表之间的关系

数组是**线性表**的扩展，其数据元素是**某类型的一个数据**或者本身也是**一个线性表**

数组的特点

- 数组中**各元素都具有统一的类型**
- 可以认为，d维数组的非边界元素具有d个直接前趋和d个直接后继
- 数组维数确定后，数据元素个数和元素之间的关系不再发生改变，适合于**顺序存储（RAM的基本模型）**
- 每组有定义的下标都存在一个与其相对应的值

在数组上的基本操作

- 给定一组下标，取得相应的数据元素值
- 给定一组下标，修改相应的数据元素值

数组的基本操作定义

(1)构造n维数组　　A= InitArray(n, $bound_1$, …, $bound_n$)

(2)销毁数组A　　DestroyArray(A)

(3) 取得指定下标的数组元素值

e = Value(A, $index_1$, …, $index_n$)

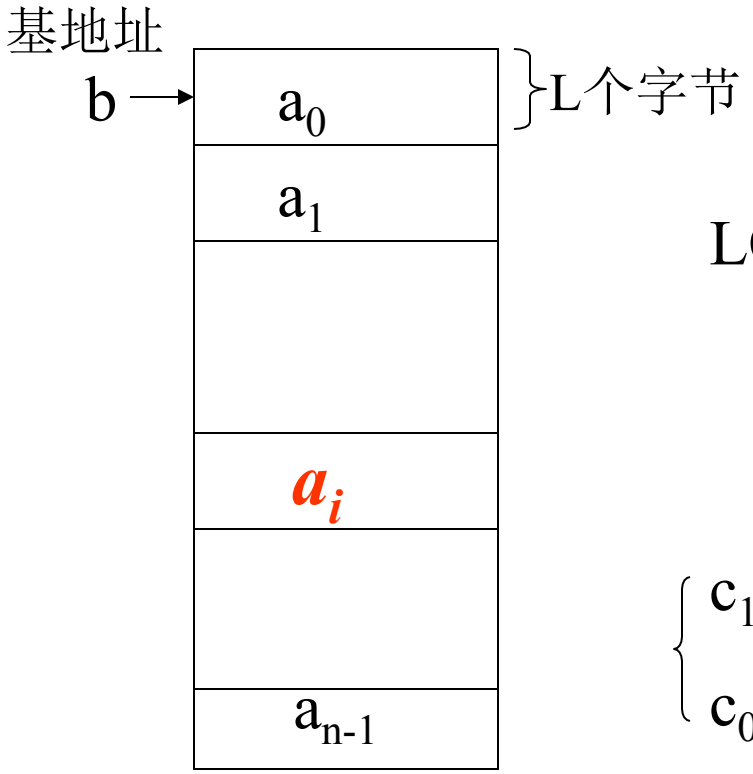
(4)为指定下标的数组元素重新赋值

Assign(A, e, $index_1$, …, $index_n$)

5.2 数组的顺序存储结构

一维数组

```
ElemType a[n];
```



$$LOC[i]=LOC[0]+i\times L$$

$$=b+ i \times L$$

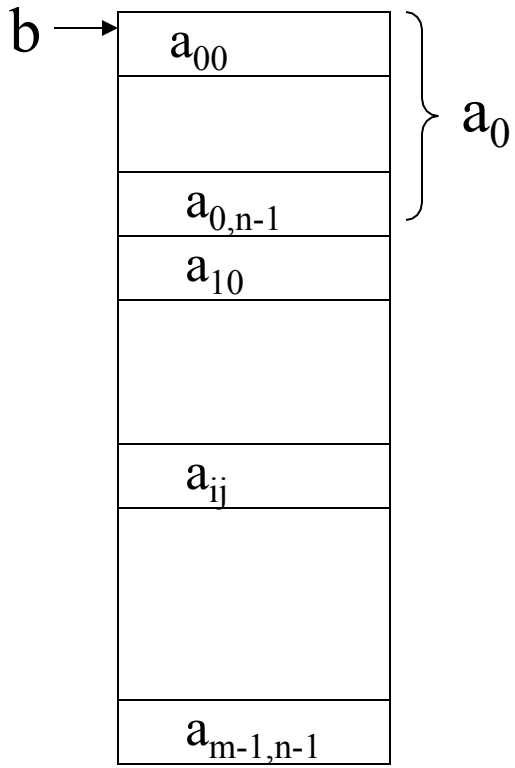
$$=c_0+c_1\times i$$

$$\begin{cases} c_1 = L \\ c_0 = b=LOC[0] \end{cases}$$

二维数组

```
ElemType a[m][n];
```

$$a=\begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \vdots & & \vdots\ a_{ij} \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix}_{m\times n}$$



$$LOC[i,j]=LOC[0,0]+(i\times n+j)\times L$$

$$=c_0+c_1\times i+c_2\times j$$

$$\begin{cases} c_2 = L \\ c_1 = n\times c_2 = \mathbf{b_2}\times c_2 \\ c_0 = b = LOC[0,0] \end{cases}$$

注：Pascal、C语言以行序为主序

“行序为主序”即“低下标优先”

“列序为主序”即“高下标优先

n维数组　　ElemType a[b_1][b_2] ... [b_n] ;

$$LOC[j_1, j_2, ..., j_n]=c_0+c_1\times j_1+c_2\times j_2+...+c_n\times j_n=c_0+\sum_{i=1}^{n}c_i\times j_i$$

$$\begin{cases} c_n=L; \\ c_{i-1}=c_i\times b_i \quad (1<i\le n) \\ c_0=b=LOC[0,0,...,0] \end{cases}$$

数组是一种随机存取结构:对任一元素定位时间相等.

思考：　ElemType　A[5, 10, 6, 8]

设每数组元素占用8个存储单元，起始地址为1000,求按低下标优先（类似行优先）顺序存储时，

LOC[3, 1, 3, 2]=?

参考解答　　维数：b1=5，b2=10，b3=6，b4=8

$$LOC[3,1,3,2]=LOC[0,0,0,0]+c_1j_1+c_2j_2+c_3j_3+c_4j_4$$

$$=1000+c_1j_1+c_2j_2+c_3j_3+8\times2$$

$$=1000+c_1j_1+c_2j_2+8\times8\times3+8\times2$$

$$=1000+c_1j_1+64\times6\times1+8\times8\times3+8\times2$$

$$=1000+384\times10\times3+64\times6\times1+8\times8\times3+8\times2$$

$$=13112$$

5.3 矩阵的压缩存储

目的是节省空间。

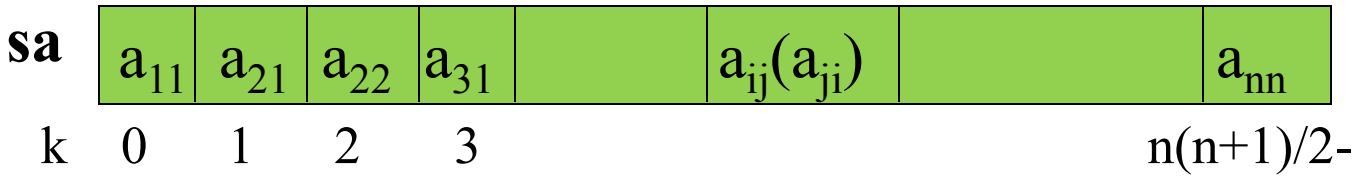
$$A=\begin{pmatrix} a_{11} & \cdots & a_{1j} & a_{1N} \\ \cdots & & & \cdots \\ a_{i1} & & a_{ij} & a_{iN} \\ \cdots & & & \cdots \\ a_{M1} & \cdots & a_{Mj} & a_{MN} \end{pmatrix}$$

对称矩阵

[特点] 在n×n的矩阵a中，满足如下：

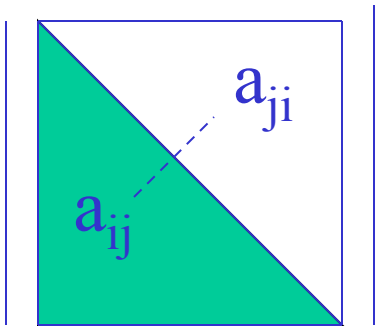
$$a_{ij}=a_{ji} \quad (1 \le i, j \le n)$$

[存储方法] 只存储下(或者上)三角(包括主对角线)的数据元素。共占用n(n+1)/2个元素空间: sa[0.. n(n+1)/2-1] 。



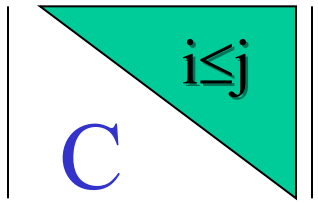
推导出k= $\begin{cases} i(i-1)/2+j-1 & \text{当} i \ge j \\ j(j-1)/2+i-1 & \text{当} i < j \end{cases}$

[根据(i,j)推导出索引k]

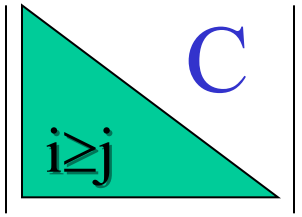


三角矩阵

[特点] 对角线以下(或者以上)的数据元素(不包括对角线)全部为常数c。



上三角矩阵



下三角矩阵

[存储方法] 重复元素C共享一个元素存储空间，共占用n(n+1)/2+1个元素空间: sa[0.. n(n+1)/2]。sa[0]=C

上三角矩阵

$$k=\begin{cases}(i-1)\times(2n-i+2)/2+j-i+1 & i\le j\\ 0 & i>j\end{cases}$$

下三角矩阵

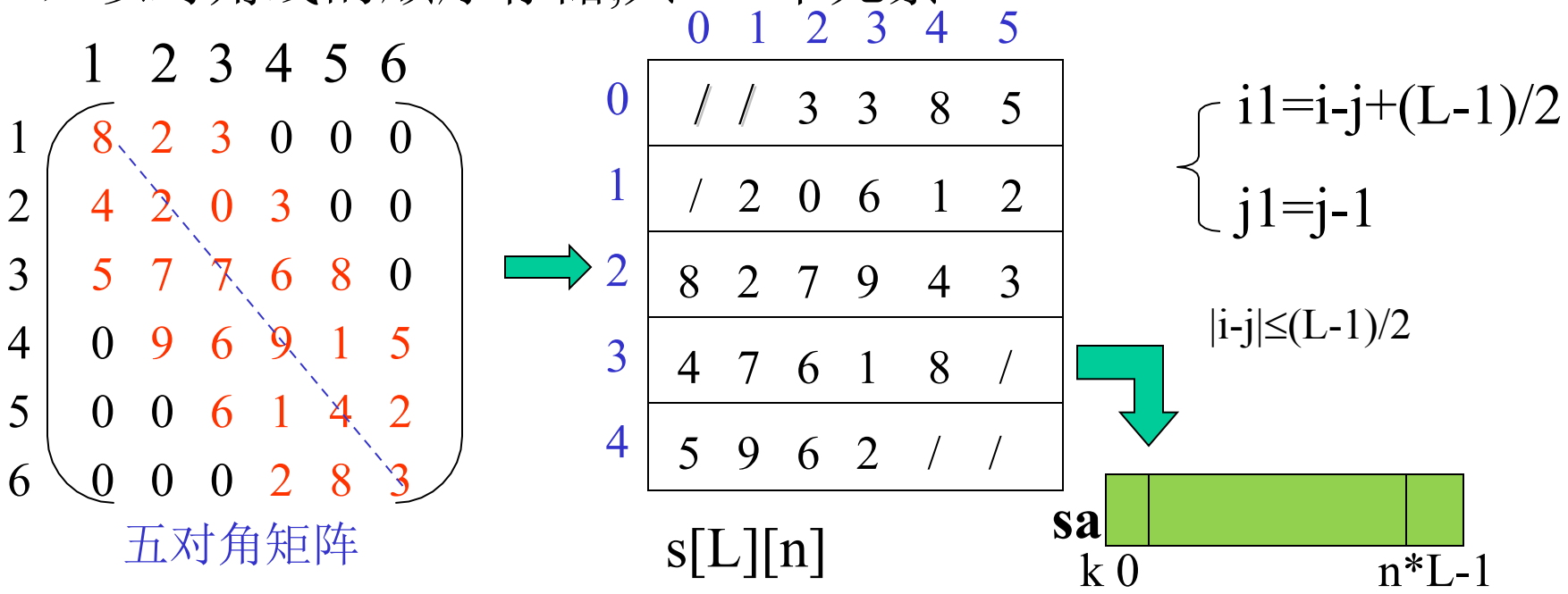
$$k=\begin{cases}i\times(i-1)/2+j & i\ge j\\ 0 & i<j\end{cases}$$

带状矩阵（对角矩阵）

[特点] 在n×n的方阵中，非零元素集中在主对角线及其两侧共L(奇数)条对角线的带状区域内 — L对角矩阵。

[存储方法] 只存储带状区内的元素。

1）以对角线的顺序存储,共n*L个元素。

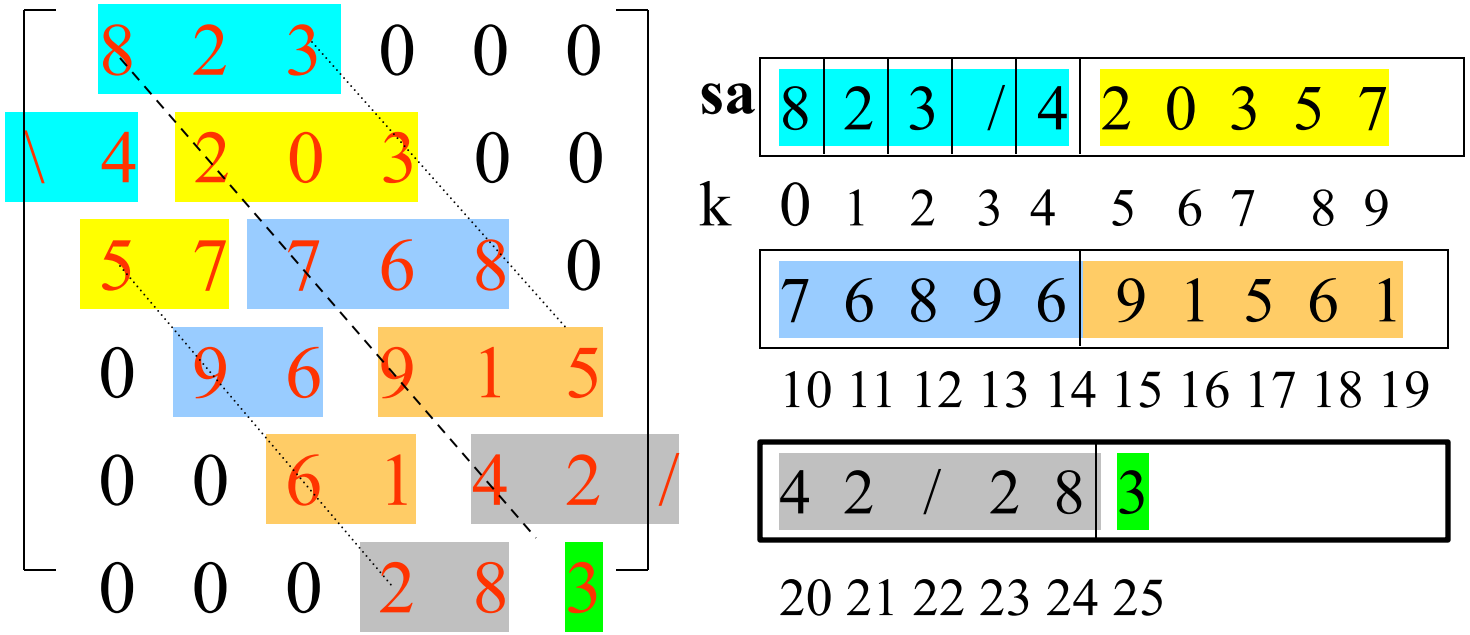


2）只存储带状区内的元素。从上一行的主对角线元素a_{i-1i-1}到本行的主对角线元素a_{ii}这一段最多有L个元素，共（n−1）L+1 个元素。

sa[0..(n-1)L]

存储地址计算： k=(i-1)L+(j-i)

$1 \le i, j \le n$　　$|i-j| \le (L-1)/2$



随机稀疏矩阵

[特点] 大多数元素为零。$\delta = t/(m \times n)$ ≤ 0.05

[常用存储方法] 记录每一非零元素(i,j,a_{ij})

节省空间，但丧失随机存取功能

- 顺序存储：三元组表
- 链式存储：十字(正交)链表

$$\begin{bmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & -6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 \end{bmatrix}$$

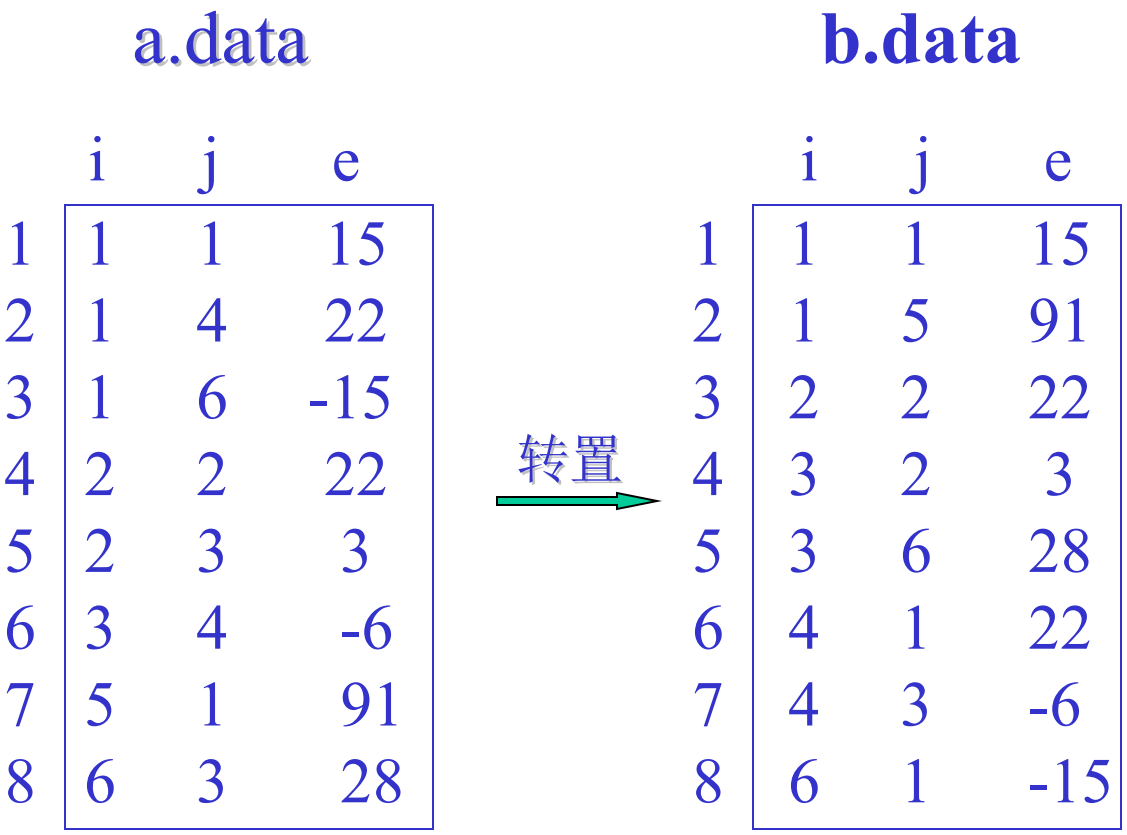
三元组表

[类型定义]

```
#define MAXSIZE   1000
    //设定非零元素最大值
struct Triple {
      int i, j;
      ElemType e;
} Triple;
struct TSMatrix {
      Triple data[MAXSIZE+1];//三元组表，data[0]未用
      int    mu, nu, tu;      //行数、列数、非零元个数
} TSMatrix;
```

	i	j	e
1	1	1	15
2	1	4	22
3	1	6	-15
4	2	2	22
5	2	3	3
6	3	4	-6
7	5	1	91
8	6	3	28
m			

[应用例] 求转置矩阵


a.data
i j e
1 1 1 15
2 1 4 22
3 1 6 -15
4 2 2 22
5 2 3 3
6 3 4 -6
7 5 1 91
8 6 3 28
转置
b.data
i j e
1 1 1 15
2 1 5 91
3 2 2 22
4 3 2 3
5 3 6 28
6 4 1 22
7 4 3 -6
8 6 1 -15

算法1 转置矩阵 { O(nu×tu) }

```
void TransMatrix(struct TSMatrix *a, struct TSMatrix *b) //求a的转置矩阵b
{
    b->mu = a->nu;  b->nu = a->mu;  b->tu = a->tu;
    if (b->tu == 0)
        return;

    q = 1;     //指示b中存放数据的位置，初值为1
    for (col = 1; col <= a->nu; col++) {   //对a的每一列
        for (p = 1; p <= a->tu; p++) {      //对a的每个三元组检查
            if (a->data[p].j == col) {     //找列号为col的三元组
                b->data[q].i = col;
                b->data[q].j =a->data[p].i;
                b->data[q].e =a->data[p].e;
                q++;                              //修正q值
            }
        }
    }
}
```

算法2 快速转置法 { O(nu+tu) }

引入两个辅助向量：

- num[1 ~a.nu]：a中每列的非零元素个数
- cpot[1 ~ a.nu]：a中每列的首个非零元素在b中的位置

a中的列号

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

a	i	j	e
1	1	1	15
2	1	4	22
3	1	6	-15
4	2	2	22
5	2	3	3
6	3	4	-6
7	5	1	91
8	6	3	28

转置

b	i	j	e
1	1	1	15
2	1	5	91
3	2	2	22
4	3	2	3
5	3	6	28
6	4	1	22
7	4	3	-6
8	6	1	-15

cpot[1]=1

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

(2≤col≤n)

```
void FastTransMatrix(struct TSMatrix *a, struct TSMatrix *b)
{
    b->mu = a->nu;  b->nu = a->mu;  b->tu = a->tu;

    for (col = 1; col <= a->nu; col++) // num清零
        num[col] = 0;
    for (t = 1; t <= a->tu; t++) //对a->tu个非零元素按列号计数
        num[a->data[t].j]++;

    cpot[1] = 1;                    //生成cpot
    for (col = 2; col <= a->nu; col++)
        cpot[col] = cpot[col - 1] + num[col - 1];

    //对tu个非零元素转置
    for (p = 1; p <= a->tu; p++)  {
        col = a->data[p].j;
        q = cpot[col];
        b->data[q].i = col;
        b->data[q].j = a->data[p].i;
        b->data[q].e = a->data[p].e;
        cpot[col]++;
    }
}
```