第五章 数组和广义表

- 一维数组具有线性表的结构，但操作简单，一般不进行插入和删除操作，只定义给定下标读取元素和修改元素的操作
- 二维数组中，每个数据元素对应一对数组下标，在行方向上和列方向上都存在一个线性关系，即存在两个前驱和两个后继。也可看作是以线性表为数据元素的线性表。
- n维数组中，每个数据元素对应n个下标，受n个关系的制约，其中任一个关系都是线性关系。可看作是数据元素为n-1维数组的一维数组。
- 因此，多维数组是对线性表的扩展：线性表中的数据元素本身又是一个多层次的线性表。

5.1 数组的定义

- ADT Array{

  数据对象：$j_i=0,\cdots,b_i-1,\ i=1,2,\cdots,n,$

  $D=\{a_{j1j2\cdots jn} \mid n(>0)$称为数组的维数，$b_i$是数组第i维的长度，$j_i$是数组元素的第i维下标，

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

  数据关系：$R=\{R_1,R_2,\cdots,Rn\}$

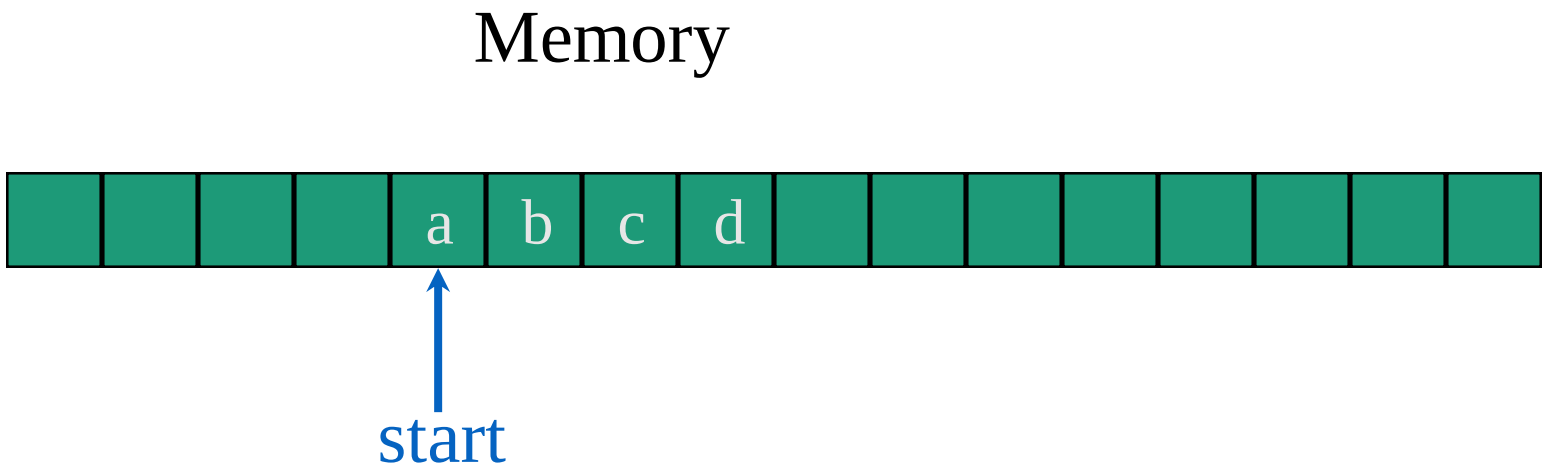
  $Ri=\{<a_{j1\cdots ji\cdots jn},a_{j1\cdots ji+1\cdots jn}> \mid 0\leqslant j_k\leqslant b_k-1,\ 1\leqslant k\leqslant n$ 且$k\neq i, 0\leqslant j_i\leqslant b_i-2,\ a_{j1\cdots ji\cdots jn},a_{j1\cdots ji+1\cdots jn} \in D, i=2,\cdots n\}$

基本操作

- InitArray(&A,n,$bound_1$,…,$bound_n$)
- DestroyArray(&A)
- Value(A,&e,$index_1$,…,$index_n$)
- Assign(&A,e,$index_1$,…,$index_n$)

}ADT Array

1D Array Representation In C



- `1-dimensional array x = [a, b, c, d]`
- `map into contiguous memory locations`

- location(x[i]) = start + i

2D Arrays

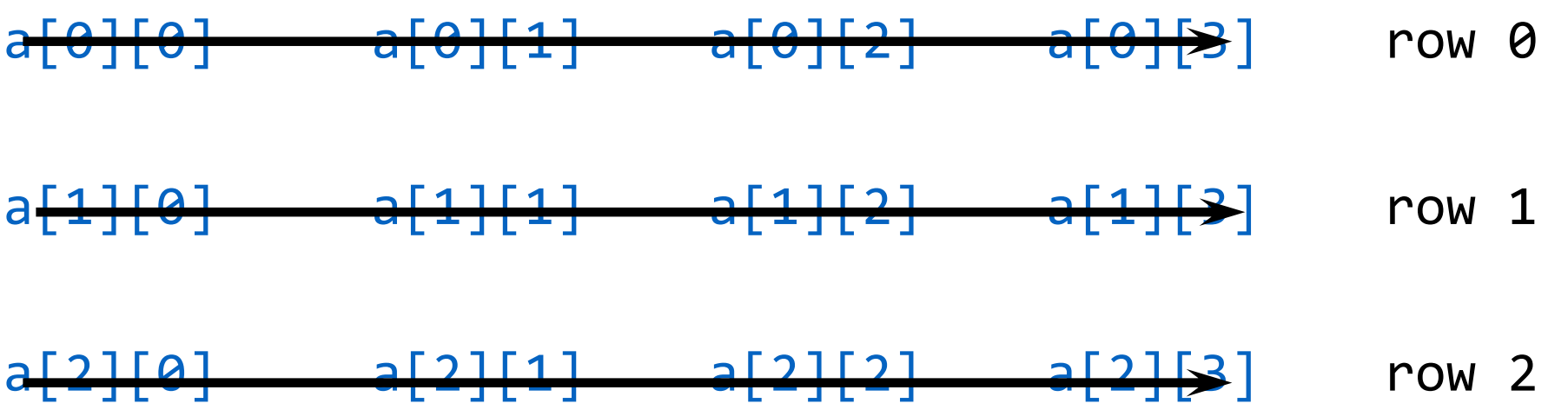
The elements of a 2-dimensional array `a` declared as:

```
int [][]a = new int[3][4];
```

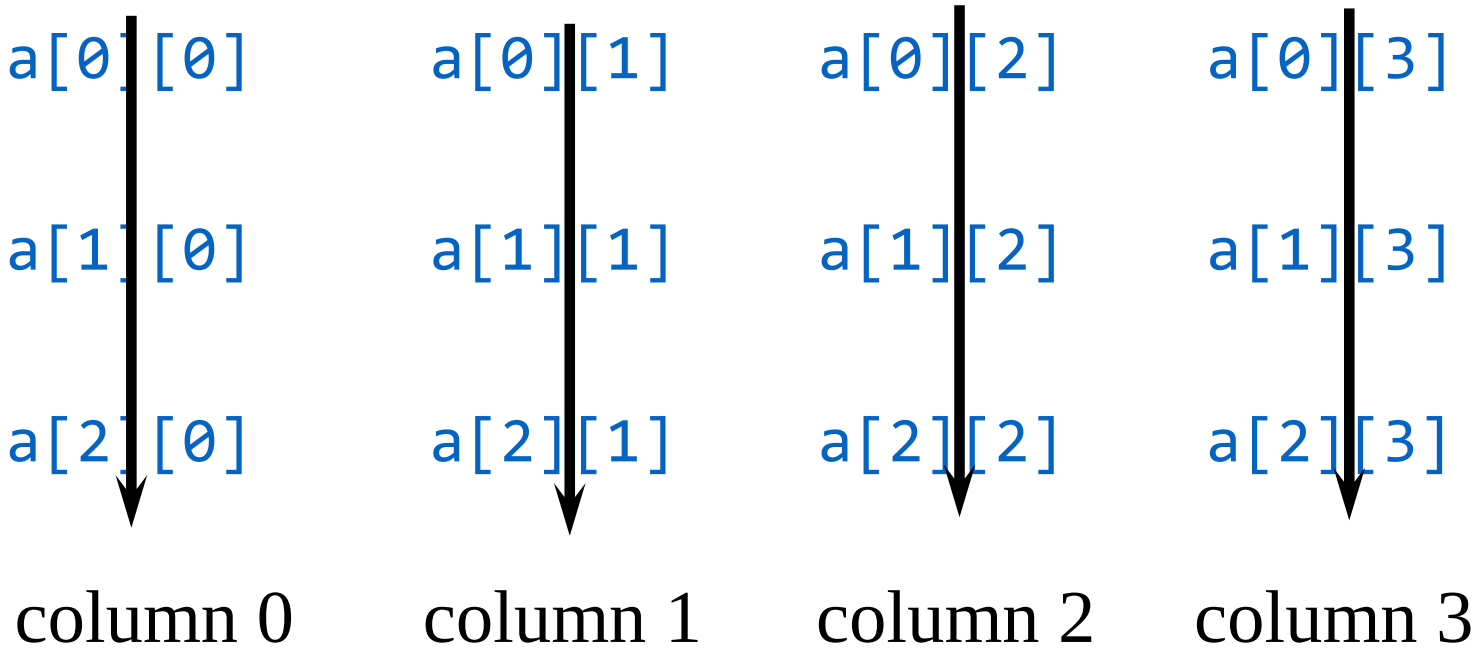
may be shown as a table

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

Rows Of A 2D Array



Columns Of A 2D Array



2D Array Representation In C++

2-dimensional array x

a, b, c, d

e, f, g, h

i, j, k, l

```
view 2D array as a 1D array of rows
    x = [row0, row1, row 2]
    row 0 = [a,b, c, d]
    row 1 = [e, f, g, h]
    row 2 = [i, j, k, l]
and store as 4 1D arrays
```

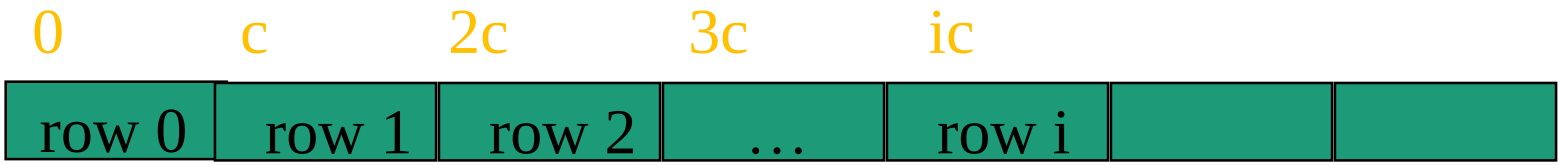
Row-Major Mapping

- Example 3 x 4 array:

```
a b c d
e f g h
i j k l
```

- Convert into 1D array y by collecting elements by rows.
- Within a row elements are collected from left to right.
- Rows are collected from top to bottom.
- We get {a, b, c, d, e, f, g, h, i, j, k, l}

Locating Element x[i][j]



- assume x has r rows and c columns
- each row has c elements
- i rows to the left of row i
- so ic elements to the left of x[i][0]
- so x[i][j] is mapped to position
  ic + j of the 1D array

5.2 数组的顺序表示

- 多维数组用一维的存储单元存放需约定次序。PASCAL和C语言是以行序为主序，FORTRAN以列序为主序。
- 给定维数和各维长度，数组的存储空间确定。
- 二维数组中任一元素a_{ij}的存储地址:

  $Loc(i,j)=Loc(0,0)+(b_2*i+j)*L$

- n维数组:

  $Loc(j_1,j_2,\cdots,j_n)=Loc(0,0,\cdots,0)+\sum c_i j_i$

  其中 $c_n=L$, $c_{i-1}=b_i*c_i$, $1<i\leqslant n$

类型定义

```
#include <stdarg.h>
#define MAX_ARRAY_DIM  8
typedef struct{
    ElemType *base;
    int dim;
    int *bounds;
    int *constants;
}Array;
```

5.3 矩阵的压缩存储

- 矩阵一般可用二维数组实现，特殊矩阵采用压缩存储。

- 压缩存储：
  为多个值相同的元素只分配一个存储空间，对零元不分配空间。

- 特殊矩阵：
  值相同的元素或者零元素在矩阵中的分布有一定规律

- 稀疏矩阵：
  非零元较零元少，且分布没有一定规律的矩阵

5.3.1. 特殊矩阵

- **对称矩阵：** $a_{ij}=a_{ji}$ $1\leqslant i,j\leqslant n$
  - 压缩存储方法：为每一对对称元分配一个存储空间
    - 将下三角的元素，按行存储到一维数组sa中
    - 共有n(n+1)/2个存储单元，
    - a_{ij}在一维数组中的位置k为：i(i-1)/2+j-1 当i>=j 否则
      j(j-1)/2+i-1
- **三角矩阵：上（下）三角中的元素均为常数c或0**
  - 压缩存储方法：同上，只存储上（下）三角元素。
  - 下三角：k=i*(i-1)/2+j-1
  - 上三角：k=(2n-i)(i-1)/2+j-1（按行）
    k=j(j-1)/2+i-1 （按列）
  - 注意：k从零开始，i,j从1开始
- **对角矩阵：所有非零元都集中在以主对角线为中心的带状区域中。**
  - 压缩方法：压缩存储到一维数组sa[]中，三对角矩阵有3n-2个元素。
  - k=2*i+j-3

5.3.2. 稀疏矩阵

1. 三元组的表示

- 稀疏矩阵可由表示非零元的三元组及其行列数唯一确定。
- t个非零元，t/(m*n)称为稀疏因子 <0.05
- 用三元组(i,j,a_{ij})存储行、列的位置，以及非零元素值。

（1）稀疏矩阵 (Sparse Matrix)

$$\mathbf{A}_{6\times7}=\begin{pmatrix}0&0&0&22&0&0&15\\0&11&0&0&0&17&0\\0&0&0&-6&0&0&0\\0&0&0&0&0&39&0\\91&0&0&0&0&0&0\\0&0&28&0&0&0&0\end{pmatrix}$$

行数$m=6$，列数$n=7$，非零元素个数$t=8$

A的转置矩阵

$$\mathbf{B}_{7\times6}=\begin{pmatrix}0&0&0&0&91&0\\0&11&0&0&0&0\\0&0&0&0&0&28\\22&0&-6&0&0&0\\0&0&0&0&0&0\\0&17&0&39&0&0\\15&0&0&0&0&0\end{pmatrix}$$

行数$m=7$，列数$n=6$，非零元素个数$t=8$

稀疏矩阵的三元组表表示

$$A_{6\times7}=\begin{pmatrix}0&0&0&22&0&0&15\\0&11&0&0&0&17&0\\0&0&0&-6&0&0&0\\0&0&0&0&0&39&0\\91&0&0&0&0&0&0\\0&0&28&0&0&0&0\end{pmatrix}$$

	行 (row)	列 (col)	值 (value)
[0]	1	4	22
[1]	1	7	15
[2]	2	2	11
[3]	2	6	17
[4]	3	4	−6
[5]	4	6	39
[6]	5	1	91
[7]	6	3	28

转置矩阵

$$B_{7\times6}=\begin{pmatrix}0&0&0&0&91&0\\0&11&0&0&0&0\\0&0&0&0&0&28\\22&0&-6&0&0&0\\0&0&0&0&0&0\\0&17&0&39&0&0\\15&0&0&0&0&0\end{pmatrix}$$

	行 (row)	列 (col)	值 (value)
[0]	1	5	91
[1]	2	2	11
[2]	3	6	28
[3]	4	1	22
[4]	4	3	−6
[5]	6	2	17
[6]	6	4	39
[7]	7	1	16

(2) 三元组顺序表

```
#define MAXSIZE 12500  //非零元个数最大值
typedef struct {
   int i,j;  //行下标和列下标
   ElemType e;
} Triple;
typedef struct{
 Triple data[MAXSIZE+1];//存非零元,data[0]未用
   int mu,nu,tu;  //行数、列数、非零元个数
}TSMatrix;
TSMatrix  a,b;
```

所需空间：3*tu+3

注：若tu与 mu * nu为同一数量级时，用三元组表示不能节省空间

(3)三元组表示稀疏矩阵的转置运算

- 算法1：按照b.data中的三元组的次序，即M的列序，依次在a.data中找到相应的三元组进行转置。
- 步骤：从k=1开始依次扫描找寻所有列号为k的项，将其行号变列号、列号变行号，顺次存于转置矩阵三元组表。

- 其时间复杂度为 O（a.nu* a.tu）。
- 例：若矩阵有200行，200列，10,000个非零元素，总共有2,000,000次处理。

i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

i	j	v
1	**3**	**-3**
1	**6**	**15**
2	**1**	**12**
2	**5**	**18**
3	**1**	**9**
3	**4**	**24**
4	**6**	**-7**
6	**3**	**14**

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

时间复杂度为：

O(a.nu* a.tu)

稀疏矩阵的转置（算法5.1)

```
Status TransposeSMatrix(TSMatrix M,TSMatrix &T)
{ int q,col,p;
  T.mu=M.nu; T.nu=M.mu; T.tu=M.tu;
  if (T.tu)
    { q=1;
        for (col=1;col<=T.mu;++col)
            for(p=1;p<=M.tu;++p)
                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; }
        }
   return OK; }
```

快速转置算法

- 方法：按a.data中三元组的次序进行转置，并将转置后的三元组置入b中恰当的位置。
- 建立辅助数组num和cpot，num[col]表示矩阵第col列中非零元的个数，cpot[col]指示第col列的第一个非零元素在b.data中的恰当位置。
- 按行扫描矩阵三元组表，根据某项的列号，确定它转置后的行号，查cpot表，按查到的位置直接将该项存入转置三元组表中。
- 转置时间复杂度为 O(nu+tu+nu+tu)=O(tu)。若矩阵有200列，10000个非零元素，总共需10000次处理。

i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

i	j	v
1	**3**	**-3**
1	**6**	**15**
2	**1**	**12**
2	**5**	**18**
3	**1**	**9**
3	**4**	**24**
4	**6**	**-7**
6	**3**	**14**

col	num	cpot
1	2	
2	2	
3	2	
4	1	
5	0	
6	1	
7	0	

时间复杂度为：
O(a. tu+)

cpot[1]=1

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

i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

i	j	v
1	**3**	**-3**
1	**6**	**15**
2	**1**	**12**
2	**5**	**18**
3	**1**	**9**
3	**4**	**24**
4	**6**	**-7**
6	**3**	**14**

col	num	cpot
1	**2**	1
2	2	3
3	2	5
4	1	7
5	0	8
6	1	8
7	0	9

cpot[1]=1

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

时间复杂度为：

O(a.tu+a.nu+)

i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

	i	j	v
1	**1**	**3**	**-3**
2	**1**	**6**	**15**
3	**2**	**1**	**12**
4	**2**	**5**	**18**
5	**3**	**1**	**9**
6	**3**	**4**	**24**
7	**4**	**6**	**-7**
8	**6**	**3**	**14**

col	num	cpot
1	**2**	1
2	2	3
3	2	5
4	1	7
5	0	8
6	1	8
7	0	9

cpot[1]=1

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

时间复杂度为：

O(a.tu+a.nu+a.tu)

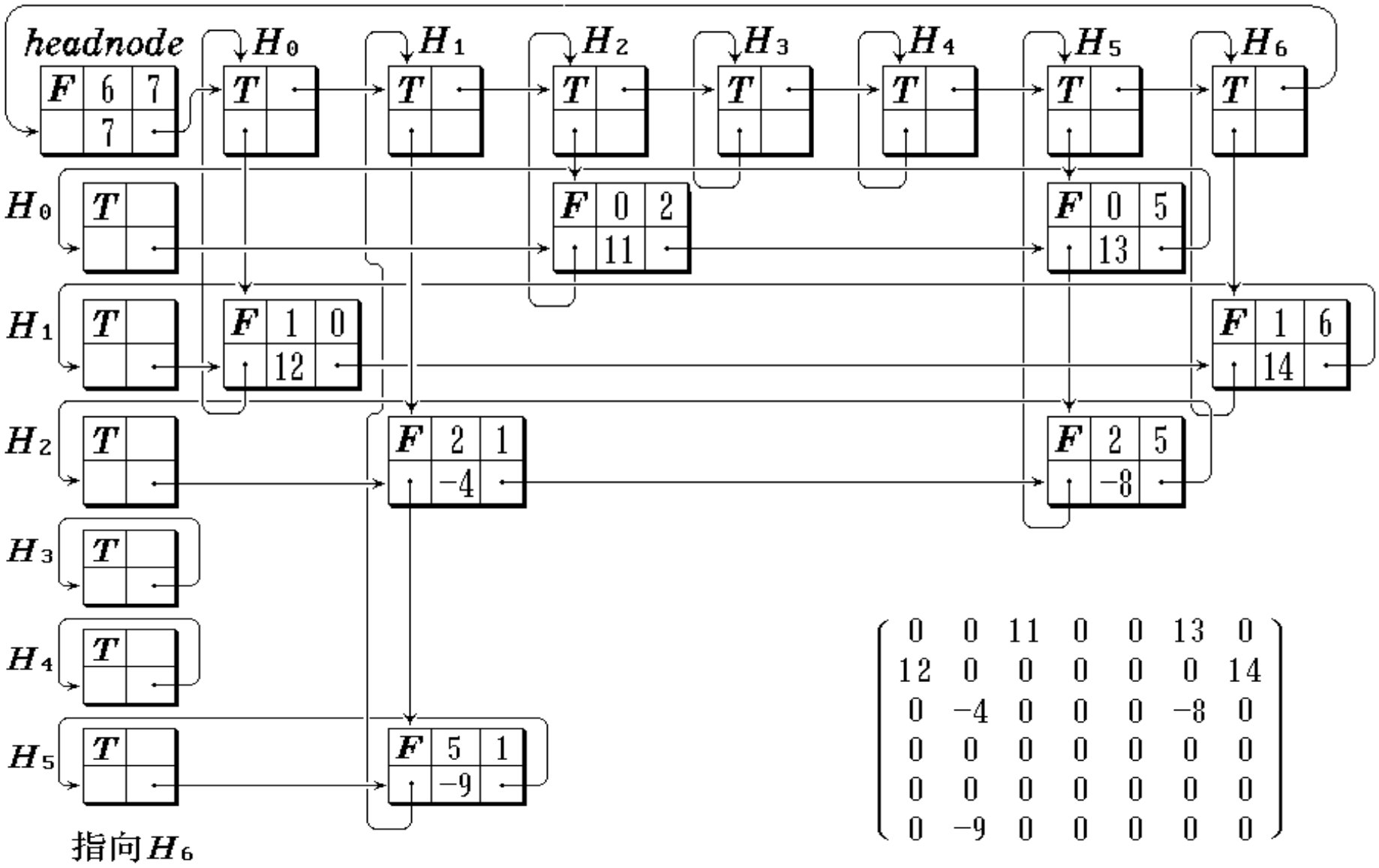
稀疏矩阵的快速转置(算法5.2)

```
Status FastTransposeSMatrix(TSMatrix M,TSMatrix &T)
 { T.mu=M.nu;  T.nu=M.mu;  T.tu=M.tu;
  if (T.tu)
     { for (col=1;col<=M.nu;++col) num[col]=0;
           for (t=1;t<=M.tu;++t) ++num[M.data[t].j];
           cpot[1]=1;
           for (col=2;col<=M.nu;++col)
                     cpot[col]=cpot[col-1]+num[col-1];
       for (p=1;p<=M.tu;++p)
           { col=M.data[p].j; q=cpot[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;
            ++cpot[col]; }
         }
  return OK;
 }
```

2.十字链表

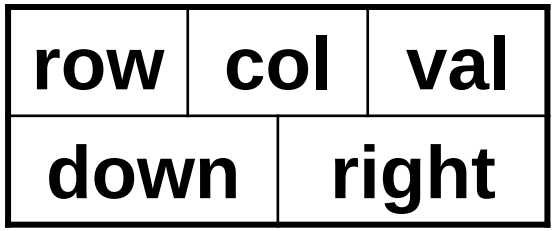
- 当矩阵中非零元素的个数和位置经过运算后变化较大时，就不宜采用顺序存储结构，而应采用链式存储结构来表示三元组。
- 稀疏矩阵的链接表示采用十字链表：行链表与列链表十字交叉。
- 行链表与列链表都是带表头结点的循环链表。用表头结点表征是第几行，第几列。

稀疏矩阵的十字链表表示的示例

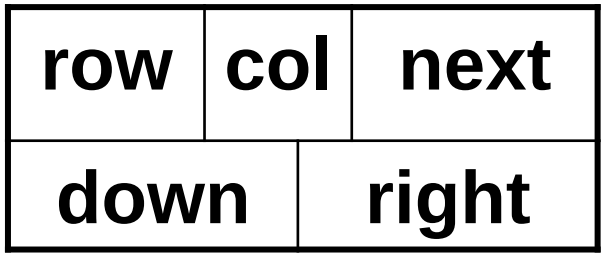


$$\begin{pmatrix} 0 & 0 & 11 & 0 & 0 & 13 & 0 \\ 12 & 0 & 0 & 0 & 0 & 0 & 14 \\ 0 & -4 & 0 & 0 & 0 & -8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -9 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

- **元素结点**
  - right——指向同一行中下一个非零元素的指针（向右域）
  - down——指向同一列中下一个非零元素的指针（向下域）



- **表头结点**
  - 行表头结点
  - 列表头结点
  - next用于表示头结点的链接



- 空间代价：需要辅助结点作链表的表头，同时每个结点要增加两个指针域，所以只有在矩阵较大和较稀疏时才能起到节省空间的效果。

十字链表的类型定义

```
typedef struct OLNode{   //元素结点
  int i,j;   //非零元的行和列下标
  ElemType e;
  struct OLNode *right,*down;
        //该非零元所在行表和列表的后继链域
 }  OLNode, *OLink;
typedef struct {
  OLink *rhead,*chead; //行和列链表头指针数组
  int mu,nu,tu;
 } CrossList;
```

注：分别用两个一维数组存储行链表的头指针和列链表的头指针，可加快访问速度。

十字链表的建立（算法5.4）