软件技术基础

第七讲

上讲主要内容

- 串的定义和运算
- 串存储结构

◆定长顺序存储　◆堆分配存储　◆链式

上讲主要内容

- 串运算的实现
  - 连接
  - 求子串
  - 子串的定位
- 模式匹配算法

本讲主要内容

- 数组的定义和运算
- 数组的顺序存储结构
- 矩阵的压缩存储
- 广义表的定义

数组

例

- 一维数组 An= $[a_1 \quad a_2 \quad \cdots \quad a_n]$

二维数组 Amn=

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \cdots & \cdots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Amn=

$$[[a_{11} \quad a_{12} \quad \cdots \quad a_{1n}],[a_{21} \quad a_{22} \quad \cdots \quad a_{2n}],\cdots,[a_{m1} \quad a_{m2} \quad \cdots \quad a_{mn}]]$$

Amn=

$$[[a_{11} \quad a_{21} \quad \cdots \quad a_{m1}],[a_{12} \quad a_{22} \quad \cdots \quad a_{m2}],\cdots,[a_{1n} \quad a_{2n} \quad \cdots \quad a_{mn}]]$$

- n维数组 $A_{b1*b2*bn}$

- **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（前件）和两个后继（后件）。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，多维数组和广义表是对线性表的扩展：线性表中的数据元素本身又是一个多层次的线性表。

数组

定义

数组是由值与下标构成的有序对，结构中的每一个数据元素都与其下标有关。

性质

- 数据元素数目固定，即一旦说明了一个数组结构，其元素数目不再有增减变化；
- 数据元素具有相同的类型；
- 数据元素的下标关系具有上下界的约束并且下标有序。

数组的定义

ADT Array{

数据对象：j_i=0,…,b_i-1,i=1,2,…,n,

D={$a_{j1j2…jn}$| 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，$a_{j1j2…jn}$∈ElemSet}

数据关系：R={R_1, R_2, …, Rn}

Ri={<$a_{j1…ji…jn}$, $a_{j1…ji+1…jn}$>| $0 \leq j_k \leq b_k-1$, $1 \leq k \leq n$ 且$k \neq i$, $0 \leq j_i \leq b_i-2$, $a_{j1…ji…jn}$, $a_{j1…ji+1…jn}$∈D, i=2, …n}

数组的定义

基本操作：

- InitArray(&A,n,bound1,…,boundn)
- DestroyArray(&A)
- Value(A,&e,index1,…,indexn)
- Assign(&A,e,index1,…,indexn)

}ADT Array

数组

运算

- 给定一组下标，存取相应的数据元素；
- 给定一组下标，修改相应数据元素中的某个数据项的值。

数组的顺序表示和实现

由于计算机的内存结构是一维的，因此用一维内存来表示多维数组，就必须按某种次序将数组元素排成一列序列，然后将这个线性序列存放在存储器中。

又由于对数组一般不做插入和删除操作，也就是说，数组一旦建立，结构中的元素个数和元素间的关系就不再发生变化。因此，一般都是采用顺序存储的方法来表示数组。

数组的顺序存储结构

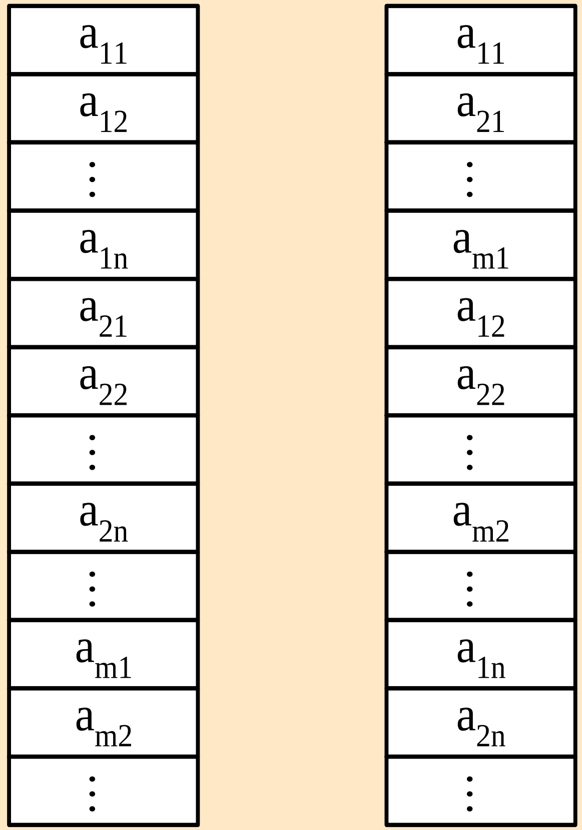
二维数组A[m][n]的存储

- 以行为主序的优先

$$Loc\ (a_{ij})=Loc\ (a_{11})+[(i-1)*n+(j-1)]*d$$

- 以列为主序的优先

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



(*a*) 以行为主序　(*b*) 以列为主序

数组的顺序存储结构

三维数组A[m][n][p]的存储

下标顺序

$$Loc(a_{ijk})=$$
$$Loc(a_{111})+[(i-1)*n*p+(j-1)*p+(k-1)]*d$$

逆下标顺序

$$Loc(a_{ijk})=$$
$$Loc(a_{111})+[(k-1)*m*n+(j-1)*m+(i-1)]*d$$

（1）下标顺序	（2）逆下标顺序
a_{111}	a_{111}
…	…
a_{11p}	a_{m11}
…	…
a_{1n1}	a_{1n1}
…	…
a_{1np}	a_{mn1}
……	……
a_{m11}	a_{11p}
…	…
a_{m1p}	a_{m1p}
…	…
a_{mn1}	a_{1np}
…	…
a_{mnp}	a_{mnp}

C语言中存储地址的计算

二维数组A[m][n]的存储

- 以行为主序的优先

$Loc(a_{ij})=Loc(a_{00})+[i*n+j]*d$

- 以列为主序的优先

$Loc(a_{ij})=Loc(a_{00})+[j*m+i]*d$

三维数组A[m][n][p]的存储

- 下标顺序

$Loc(a_{ijk})=$

$Loc(a_{000})+[i*n*p+j*p+k]*d$

- 逆下标顺序

$Loc(a_{ijk})=$

$Loc(a_{000})+[k*m*n+j*m+i]*d$

n维数组

n维数组：教材p93

$$Loc(j1, j2, \ldots, jn) = Loc(0, 0, \ldots, 0) + \sum c_i j_i$$

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

类型定义

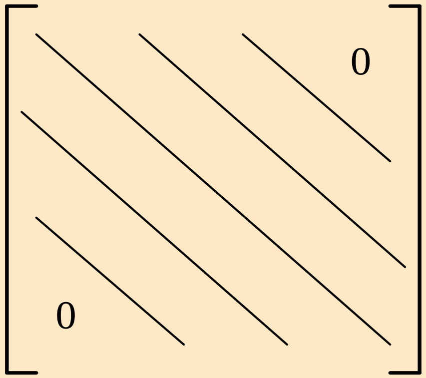
```
#include <stdarg.h>
#define MAX_ARRAY_DIM  8
typedef struct{
    ElemType *base;   //数据元素基址
    int dim;          //数组维数
    int *bounds;      //数组维界的地址
    int *constants;   //数组映像函数常量基址
}Array;
```

矩阵的压缩存储

对于值相同的元素或者零元素在矩阵中的分布具有一定规律的矩阵，我们称其为特殊矩阵

对角阵

所有的非零元素都集中在以主对角线为中心的带状区域中，即除了主对角线上和主对角线邻近的上、下方以外，其余元素均为零



（a）一般矩阵

$$\begin{bmatrix} a_{00} & a_{01} & & & \\ a_{10} & a_{11} & a_{12} & & \\ & \ddots & \ddots & \ddots & \\ & & & & a_{n-2,n-1} \\ & & & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix}$$

（b）三对角矩阵

即只在主对角线上含有非零元素的对角矩阵

$$\begin{bmatrix} a_{00} & & & 0 \\ & a_{11} & & \\ & & \ddots & \\ 0 & & & a_{n-1,n-1} \end{bmatrix}$$

n个非零元素

一维数组A[0,…,n-1];

A[k]=a[k][k]

k=i

三角阵

上三角矩阵是指矩阵的下三角（不包含对角线）中的元素均为常数（或零）的**n**阶矩阵，下三角矩阵则与之相反。

$$\begin{bmatrix} a_{00} & & & 0 \\ a_{10} & & & \\ \vdots & & \ddots & \\ a_{n-1,0} & & \cdots & a_{n-1,n-1} \end{bmatrix}$$

(*a*) 下三角矩阵

$$\begin{bmatrix} a_{00} & a_{01} & \cdots & a_{0,n-1} \\ & a_{11} & & \\ & & \ddots & \vdots \\ 0 & & & a_{n-1,n-1} \end{bmatrix}$$

(*b*) 上三角矩阵

下三角阵

n*（n+1）/2个非零元素

一维数组

A[0,…, n*（n+1）/2];

$$k = \begin{cases} \dfrac{i*(i+1)}{2} + j & i \geq j \\ \dfrac{n*(n+1)}{2} & i < j \end{cases}$$

对称阵

n阶方阵 A 元素满足$a_{ij}=a_{ji}(0\leqslant i, j\leqslant \tilde{n}1)$

下三角阵

n*（n+1）/2个非零元素

一维数组

A[0,…, n*（n+1）/2];

k=i∗(i+1)/2+j

其中，i=max(i, j)，

j=min(i, j)

矩阵的压缩存储

含有非零元素及**较多的零元素**，但非零元素的分布**没有任何规律**，这就是**稀疏矩阵稀疏矩阵**

A_{mn}中有s个非零元素，t个零元素，若s<<t，则称A为**稀疏矩阵**。

三元组表

三元组=（行i，列j，非零元素值）——行优先

三元组=（列j，行i，非零元素值）——列优先

三元组表

[定义]

将表示稀疏矩阵的非零元素的三元组按行优先（或列优先）的顺序排列（跳过零元素），则得到一个其结点均是三元组的线性表。我们将该线性表的顺序存储结构称为三元组表。

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];  //非零元三元组表
   int mu,nu,tu;  //行数、列数、非零元个数
}TSMatrix;
TSMatrix  a,b;
```

稀疏矩阵的三元组表示实例

$$A_{4\times5}=\begin{bmatrix} 0 & 5 & 0 & 0 & 8 \\ 1 & 0 & 3 & 0 & 0 \\ 0 & -2 & 0 & 0 & 0 \\ 6 & 0 & 0 & 0 & 0 \end{bmatrix}$$

稀疏矩阵A

	i	j	v
0	0	1	5
1	0	4	8
⋮	1	0	1
	1	2	3
⋮	2	1	-2
a→ t-1	3	0	6
⋮			
smax-1			

稀疏矩阵A的三元组表

稀疏矩阵三元组表示的转置运算

$$A_{4\times5}=\begin{bmatrix}0 & 5 & 0 & 0 & 8\\ 1 & 0 & 3 & 0 & 0\\ 0 & -2 & 0 & 0 & 0\\ 6 & 0 & 0 & 0 & 0\end{bmatrix} \xrightarrow{\text{转置为}} B_{5\times4}=\begin{bmatrix}0 & 1 & 0 & 6\\ 5 & 0 & -2 & 0\\ 0 & 3 & 0 & 0\\ 0 & 0 & 0 & 0\\ 8 & 0 & 0 & 0\end{bmatrix}$$

	i	j	v
0	0	1	5
1	0	4	8
⋮	1	0	1
	1	2	3
⋮	2	1	-2
a → t-1	3	0	6
⋮			
smax-1			

	i	j	v
0	0	1	1
1	0	3	6
⋮	1	0	5
	1	2	-2
⋮	2	1	3
b → t-1	4	0	8
⋮			
smax-1			

三元组表

- 如果简单地交换a→data中i和j中的内容，那么得到的b→data是一个按列优先顺序存储的稀疏矩阵B 。

	i	j	v
0	0	1	5
1	0	4	8
⋮	1	0	1
	1	2	3
⋮	2	1	-2
a→ t-1	3	0	6
⋮			
smax-1			

转置为

	i	j	v
0	0	1	1
1	0	3	6
⋮	1	0	5
	1	2	-2
⋮	2	1	3
b→ t-1	4	0	8
⋮			
smax-1			

	i	j	v
0	1	0	5
1	4	0	8
⋮	0	1	1
	2	1	3
⋮	2	1	-2
a→ t-1	0	3	6
⋮			
smax-1			

如果简单地交换a→data中的i和j

稀疏矩阵三元组表示的转置的算法

```
Void transmatrix(tripletable a,tripletable b)
  {
    int p,q, col;    /* p为a表指针；q为b表指针；col指示*a的列号(即*b的行号) */
    b.m=a.n;
    b.n=a.m;
    b.t=a.t;
    if(b.t<=0)
        printf("A=0\n");
     q=0;
```

```
for(col=1;col<=a.n;col++)
    for(p=0;p<=a.t;p++) /* 扫描整个三元组表 */
     if(a.data[p].j==col){/* 列号为col则进行置换 */
        b.data[q].i=a.data[p].j;
        b.data[q].j=a.data[p].i;
        b.data[q].v=a.data[p].v;
        q++; /* b结点序号加1 */
       }
 }
```

三元组表

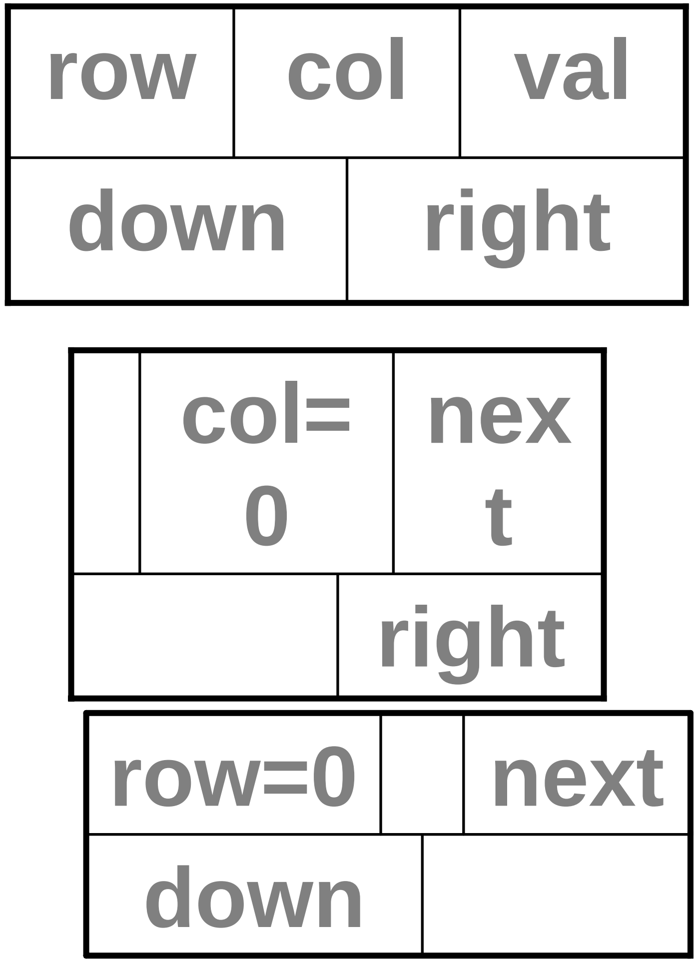
三元组顺序表虽然节省了存储空间，但时间复杂度比一般矩阵转置的算法还要复杂，同时还有可能增加算是法的难度。因此，此算法仅适用于t<=m*n的情况。

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。

十字链表

- **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



十字链表

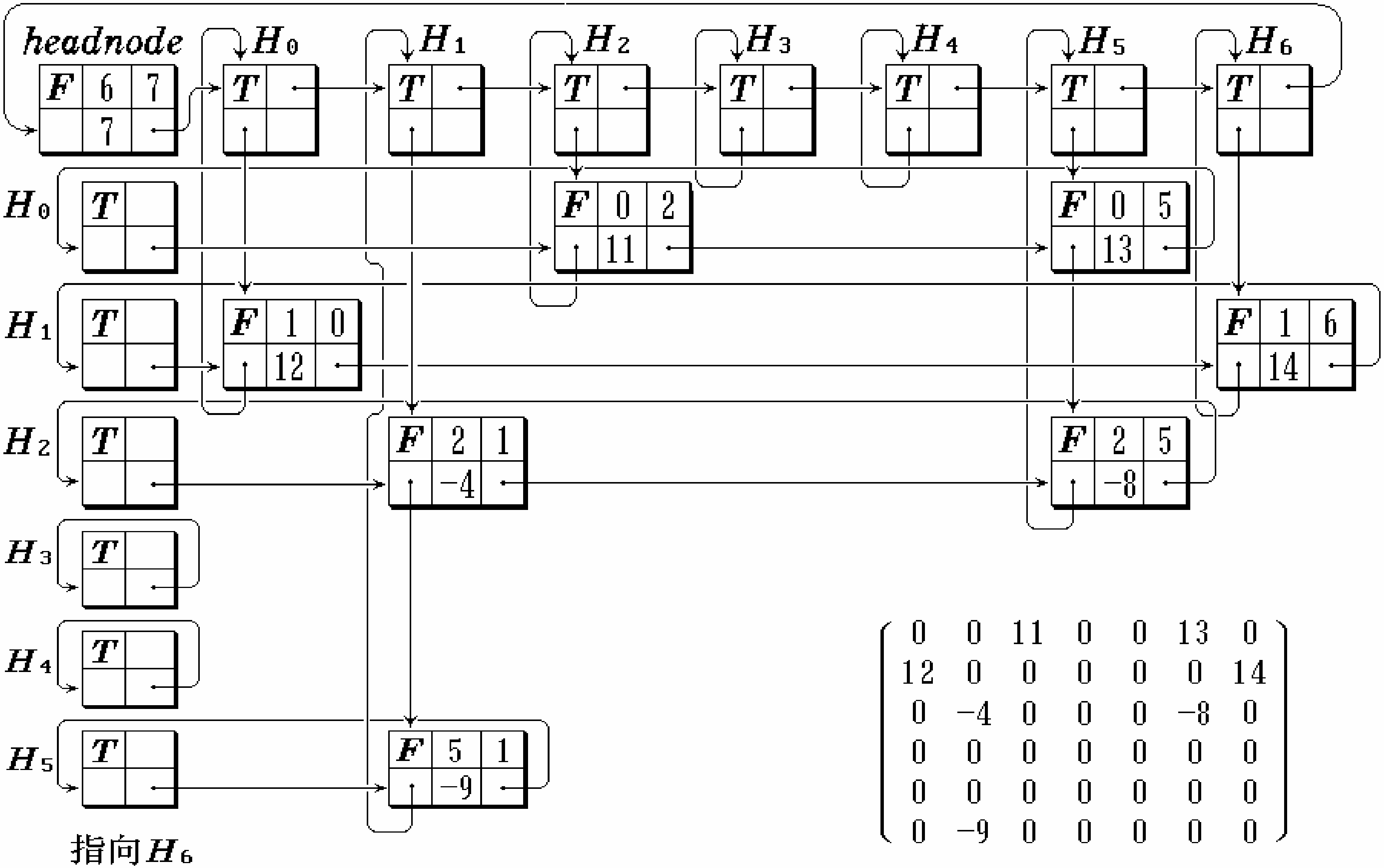
- 由于行、列表头结点互相不冲突，所以可以合并起来：

<table>
<tr><td colspan="2">row=0</td><td colspan="2">col=0</td><td colspan="2">next</td></tr>
<tr><td colspan="3">down</td><td colspan="3">right</td></tr>
</table>

- 总表头结点：

<table>
<tr><td colspan="2">row</td><td colspan="2">col</td><td colspan="2">next</td></tr>
<tr><td colspan="3"></td><td colspan="3"></td></tr>
</table>

稀疏矩阵的十字链表表示的示例



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

十字链表

类型定义：

```
typedef struct {
   int i,j;  //非零元的行下标和列下标
   ElemType e;
}Triple;
typedef struct OLNode{   //元素结点
   union { Triple  triple; OLNode *next};
   struct OLNode *right,*down;
         //该非零元所在行表和列表的后继链域
}OLNode, *OLink;
OLink  M;
```

广义表的定义

广义表（Lists，又称列表）是线性表的推广。

若放松对表元素的这种限制，容许它们具有其自身结构，这样就产生了广义表的概念。

[定义]

广义表是n (n>=0) 个元素$a_1, a_2, a_3, \ldots, a_n$的有限序列，其中$a_i$或者是原子项，或者是一个广义表。通常记作LS=（$a_1, a_2, a_3, \ldots, a_n$）。LS是广义表的名字，n为它的长度。若$a_i$是广义表，则称它为LS的子表。

小结

数组的基本概念

数组的顺序存储结构

特殊矩阵的存储方式
- 对角阵
- 三角阵
- 对称阵
- 稀疏矩阵

广义表的定义

作业

P143

<u>**23**</u>； <u>**25**</u>