5.3.2 稀疏矩阵(Sparse Matrix)

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

- 设矩阵 A 中有 s 个非零元素，若 s 远远小于矩阵元素的总数（即$s << m\times n$），则称 A 为稀疏矩阵。

- 设矩阵 A 中有 s 个非零元素。令 e = s/(m*n)，称 e 为矩阵的稀疏因子。
  - 通常认为 e≤0.05 时称之为稀疏矩阵。
- 在存储稀疏矩阵时，为节省存储空间，应只存储非零元素。但通常非零元素的分布没有规律，故在存储非零元素时，必须记下它所在的行和列的位置 (i, j)。
- 每一个三元组 (i, j, a_{ij}) 唯一确定了矩阵A的一个非零元素。因此，稀疏矩阵可由表示非零元素的一系列三元组及其行列数唯一确定。

1. 三元组顺序表

- 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。
- 在三元组顺序表中，行为主序，所有非零元素的三元组按行号递增的顺序排列；行号相等的按列号递增的顺序排序。
- 三元组顺序表中三元组的个数记忆在变量**tu** 中，此即矩阵中的非零元素个数。稀疏矩阵的行数和列数分别记忆在**mu** 和 **nu**中。

稀疏矩阵

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

```
((0,3,22),(0,6,15),(1,1,11),(1,5,17),
(2,3,-6),(3,5,39),(4,0,91),(5,2,28))
```

三元组顺序表

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

mu=6, nu=7, tu=8

// -----稀疏矩阵的三元组顺序表存储表示-------

```
#define MaxSize  100     //矩阵中非零元素最多个数
typedef int ElemType;    //矩阵元素数据类型
typedef struct {         //三元组定义
     int i, j;           //非零元素行号、列号
     ElemType  e;        //非零元素的值
} Triple;

typedef struct {         //稀疏矩阵结构定义
    int mu, nu, tu;      //矩阵行、列、非零元素
    Triple data[MaxSize];     //三元组表
} TSMatrix;
```

稀疏矩阵的转置

- 一个 $m \times n$ 的矩阵 A，它的转置矩阵 B 是一个 $n \times m$ 的矩阵，且 $A[i][j] = B[j][i]$。即
  - 矩阵 A 的行成为矩阵 B 的列
  - 矩阵 A 的列成为矩阵 B 的行。
- 在稀疏矩阵的三元组表中，非零矩阵元素按行存放。当行号相同时，按列号递增的顺序存放。
- 如果稀疏矩阵的转置运算基于三元组表，则矩阵的转置要直接对相应三元组表进行转置。

稀疏矩阵

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

对应三元组表

	行	列	值
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	-6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28

转置矩阵

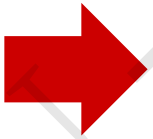
$$\begin{pmatrix} 0 & 0 & 0 & 0 & 91 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 28 \\ 22 & 0 & -6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 17 & 0 & 39 & 0 & 0 \\ 15 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

对应三元组表

	行	列	值
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	-6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

原矩阵三元组表

	行	列	值
[0]	0	3	22
[1]	0	6	15
[2]	1	1	11
[3]	1	5	17
[4]	2	3	-6
[5]	3	5	39
[6]	4	0	91
[7]	5	2	28



转置矩阵三元组表

	行	列	值
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	-6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

稀疏矩阵转置算法思想

- 设矩阵列数为 nu，对矩阵三元组表扫描 nu 次。第 k 次检测列号为 k 的项。
- 第 k 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，连同该元素的值，顺次存于转置矩阵三元组表。
- 若设矩阵非零元素有 tu 个，则上述二重循环执行的时间复杂性为O(nu×tu)。
- 若矩阵有 200 行，200 列，10,000 个非零元素,总共有 2,000,000 次处理。

稀疏矩阵的转置算法

```
void TransposeSMatrix(TSMatrix M, TSMatrix &T)
{ // 求稀疏矩阵M的转置矩阵T。算法5.1
    int p, q, col;
    T.mu = M.nu;
    T.nu = M.mu;
    T.tu = M.tu;
    if(T.tu) {
        q = 1;
        for(col = 1; col <= M.nu; ++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;
                }
    }
}
```

- 算法分析：
  - 此算法慢就慢在二重嵌套循环。若能一趟扫描过去就实现转置，运算速度将大大提高。为此，需要事先做点功课。这就是快速转置的想法。

快速转置算法

- 快速转置的想法是：对原矩阵 a 扫描一遍，按 a 中每一元素的列号，立即确定在转置矩阵 b 三元组表中的位置，并装入它。
- 为加速转置速度，建立两个辅助数组 **num[col]** 和**cpot[col]**：
  - **num[col]** 记录矩阵转置前各列非零元素个数，转置后就是各行非零元素个数；
  - **cpot[col]** 记录转置后各行第一个非零元素在转置三元组表中开始存放位置。

转置矩阵三元组表

	行	列	值
[0]	0	4	91
[1]	1	1	11
[2]	2	5	28
[3]	3	0	22
[4]	3	2	-6
[5]	5	1	17
[6]	5	3	39
[7]	6	0	16

num[col]

0	1	2	3	4	5	6
1	1	1	2	0	2	1

cpot[col]

0	1	2	3	4	5	6
0	1	2	3	5	5	7

- 扫描一遍三元组表，根据某项列号j，查 cpot 表，按 cpot[col] 所给位置直接将该项存入转置矩阵的三元组表中。

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	
num	1	1	1	2	0	2	1	矩阵 A 各列非零元素个数
cpot	0	1	2	3	5	5	7	矩阵 B 各行开始存放位置

A三元组	(0)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
行row	0	0	1	1	2	3	4	5
列col	3	6	1	5	3	5	0	2
值value	22	15	11	17	-6	39	91	28

稀疏矩阵的快速转置算法

```
void FastTransposeSMatrix(TSMatrix M, TSMatrix &T)
{
    // 快速求稀疏矩阵M的转置矩阵T。算法5.2
    int p, q, t, col, num[MaxSize], cpot[MaxSize];
    T.mu = M.nu;            // 给T的行、列数与非零元素个数赋值
    T.nu = M.mu;
    T.tu = M.tu;
    if(T.tu) {              // 是非零矩阵
        for(col = 0; col < M.nu; ++col)
            num[col] = 0;                   // 计数器初值设为0
        // 求M中每一列含非零元素个数
        for(t = 0; t < M.tu; ++t)
            ++num[M.data[t].j];
        // T的第0行的第0个非零元在T.data中的下标为0
        cpot[0] = 0;
        for(col = 1; col < M.nu; ++col)
            cpot[col] = cpot[col - 1] + num[col - 1];
```

```
        for(p = 0; p < M.tu; ++p) {
            col = M.data[p].j;          // 求得在M中的列数
            // q指示M当前的元素在T中的序号
            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;
            // T第col行的下一个非零元在T.data中的序号
            ++cpot[col];
        }
    }
}
```

- 该算法有 4 个并列单重循环，各自的时间复杂度为$O(nu)$, $O(tu)$, $O(nu)$, $O(tu)$。总的时间复杂度为$O(nu+tu)$。
- 若矩阵有 200 行，200 列，10,000 个非零元素,总共有 10,000 次处理。

2. 行逻辑链接的顺序表

- 为了便于随机存取任意一行的非零元，则需知道每一行的第一个非零元在三元组表中的位置。为此，可将指示“行”信息的辅助数组rpos固定在稀疏矩阵的存储结构中。称这种“带行链接信息”的三元组表为行逻辑链接的顺序表。

```
// 稀疏矩阵的三元组行逻辑链接的顺序表存储表示
#define MAXRC  20 // 最大行列数
typedef struct {
    Triple data[MaxSize];      // 非零元三元组表
    int rpos[MAXRC];  //各行第一个非零元素的位置表
    int mu, nu, tu;       // 行数、列数和非零元个数
} RLSMatrix;
```

	[0]	[1]	[2]	[3]	[4]	[5]
rpos	0	2	4	5	6	7

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

三元组顺序表

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

mu=6, nu=7, tu=8

3. 十字链表

- 在执行稀疏矩阵（+、-、*、/）操作时，稀疏矩阵的非零元素会发生动态变化，这时，使用三元组表有双重缺陷：

  (1) 不能直接访问矩阵元素；

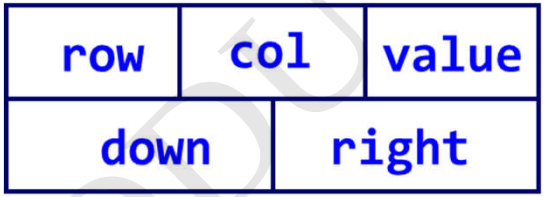
  (2) 插入或删除时可能发生大量元素移动；

- 用稀疏矩阵的链接表示可以避免这些情况。

十字链表的存储表示

一个结点除了数据域(i,j,e)之外，还应该用两个方向的指针（right，down)，分别指向行和列。

- right：用于链接同一行中的下一个元素；
- down：用于链接同一列中的下一个元素。

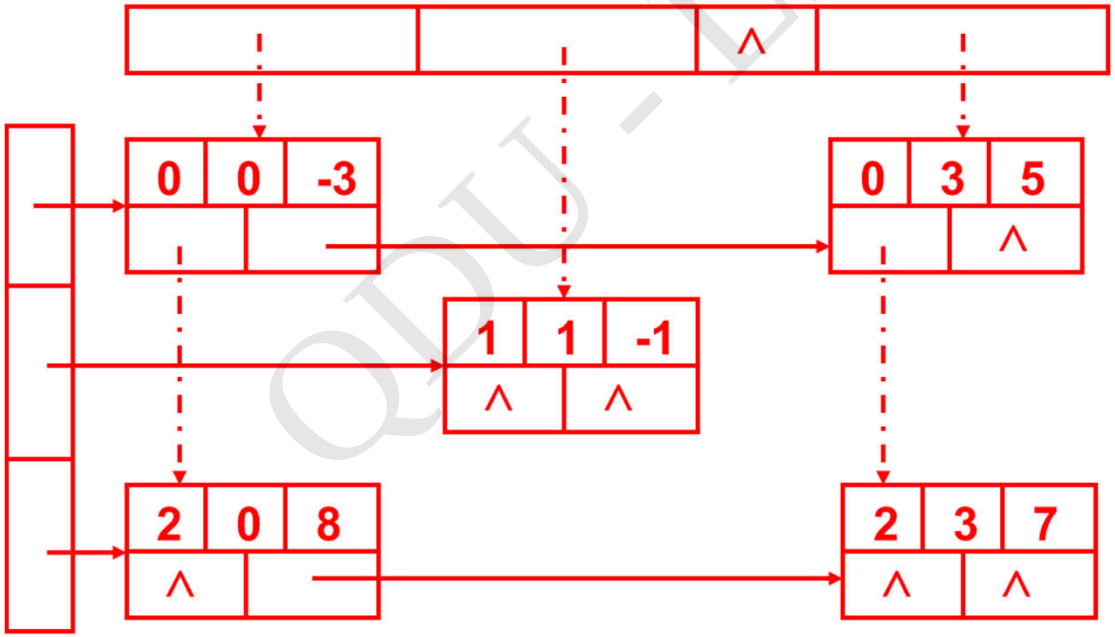


- 整个矩阵构成了一个十字交叉的链表，因此称十字链表。
- 每一行和每一列的头指针，用两个一维指针数组来存放。

十字链表的举例

row	col	value
down		right

$$M=\begin{bmatrix} -3 & 0 & 0 & -5 \\ 0 & -1 & 0 & 0 \\ 8 & 0 & 0 & 7 \end{bmatrix}$$



// 稀疏矩阵的十字链表存储表示

```
typedef struct OLNode {
    int i, j;       // 该非零元的行和列下标
    ElemType e;     // 非零元素值
    // 该非零元所在行表和列表的后继链域
    OLNode *right, *down;
} OLNode *OLink;

struct CrossList {
    OLink *rhead, *chead;
    int mu, nu, tu;       //行数、列数和非零元个数
};
```

END