5.3 矩阵的压缩存储

在数值分析中经常出现一些阶数很高的矩阵，同时在矩阵中有许多值相同的元素或零元素。有时为节省空间，可以对这类矩阵进行**压缩存储**。

假设值相同的元素或零元素在矩阵中的分布有一定规律，则称此类矩阵为**特殊矩阵**；反之，称为**稀疏矩阵**。

以常规方法，即以二维数组表示高阶的稀疏矩阵时产生的问题：

1) 零值元素占了很大空间；

2) 计算中进行了很多和零值的运算，遇除法，还需判别除数是否为零。

解决问题的原则:

1) 尽可能少存或不存零值元素；

2) 尽可能减少没有实际意义的运算；

3) 操作方便。 即：能尽可能快地找到与下标值（i，j）对应的元素，能尽可能快地找到同一行或同一列的非零值元。

5.3.1 特殊矩阵

非零元在矩阵中的分布有一定规则。例如: 上（下）三角矩阵、对角矩阵、对称矩阵等。

若n阶矩阵A中的元素满足下述性质

$$a_{ij}=a_{ji} \quad 1<=i,j<=n$$

则称为n阶对称矩阵。

对于对称矩阵，可以为每一对对称元分配一个存储空间，则可将n^2个元素压缩存储到n（n+1）/2 个元的空间中。不失一般性，可以行序为主序存储其下三角（包括对角线）中的元素。

假设以一维数组sa[n(n+1)/2]作为n阶对称矩阵A存储结构，则sa[k] 和矩阵元素a_{ij}之间存在着一一对应的关系。

$$k=\begin{cases}\dfrac{i(i-1)}{2}+j-1 & \text{当}i>=j\ \text{对角线下方}\\ \dfrac{j(j-1)}{2}+i-1 & \text{当}i<j\ \ \text{对角线上方}\end{cases}$$

对于任意给定一组下标（i，j），均可在sa中找到矩阵元素a_{ij}，反之，对所有的k=0,1,2,...,（n（n+1））/2-1，都能确定sa[k] 中的元在矩阵中的位置（i，j）。由此称sa[n（n+1）/2] 为n阶对称矩阵A的压缩存储

这种压缩存储的方法同样也适应于三角矩阵和对角矩阵。

5.3.2 稀疏矩阵

何谓稀疏矩阵？

假设 m 行 n 列的矩阵含 t 个非零元素，则称

$$\delta = \frac{t}{m \times n}$$

为**稀疏因子。**

通常认为 $\delta \leq 0.05$ 的矩阵为稀疏矩阵。

一、三元组顺序表

```
// - - 稀疏矩阵的三元组顺序表存储表示 - -
#define  MAXSIZE  12500 //最大非零元个数
 typedef struct {
     int  i，j；      //该非零元的行下标和列下标
     ElemType  e；     // 该非零元的值
 } Triple；   // 三元组类型

  typedef union {
       Triple  data[MAXSIZE + 1]； //非零元三元组表
        int   mu，nu，tu； //矩阵的行数、列数和非零元个数
 } TSMatrix； // 稀疏矩阵类型
```

例如，稀疏矩阵

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

三元组

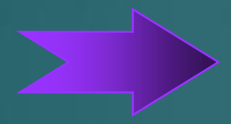
i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

如何求稀疏矩阵的转置矩阵？

例如，稀疏矩阵

$$\begin{bmatrix} 0 & 14 & 0 & 0 & -5 \\ 0 & -7 & 0 & 0 & 0 \\ 36 & 0 & 0 & 28 & 0 \end{bmatrix}$$

转置



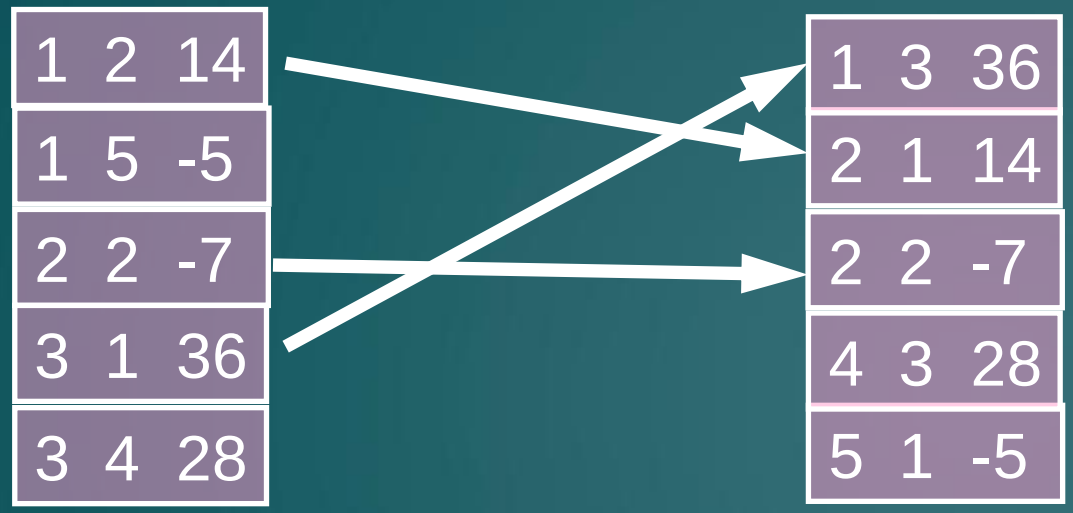
$$\begin{bmatrix} 0 & 0 & 36 \\ 14 & -7 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 28 \\ -5 & 0 & 0 \end{bmatrix}$$

用常规的二维数组表示时的算法

```
for (col=1； col<=nu； ++col)
    for (row=1； row<=mu； ++row)
      T[col][row] = M[row][col]；
```

其中，M为原稀疏矩阵，T为转置后的稀疏矩阵，其时间复杂度为： O(mu×nu)

用“三元组”表示时如何实现？



a.data

b.data

假设a和b是TSMatrix型的变量，分别表示矩阵M和T。可以有两种处理方法：

（1）按照b.data中三元组的次序依次在a.data中找到相应的三元组进行转置。换句话说，按照矩阵M的列序来进行转置。

为了找到M的每一列中所有的非零元素，需要对其三元组表a.data 从第一行起整个扫描一遍，由于a.data 是以M的行序为主序来存放每个非零元的，由此得到的恰是b.data应有顺序。

具体算法如5.1所示

```
Status TransposeSMatrix(TSMatrix M, TSMatrix &T){
  //采用三元组表存储表示，求稀疏矩阵M的转置矩阵T
  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；} // if
  }
  return OK；
} // TransposeSMatrix
```

算法 5.1

该算法主要工作是在p和col的两重循环中完成的，故算法的时间复杂度为**O(nu*tu)**。

当非零元的个数tu和mu*nu同数量级时，算法5.1的时间复杂度就为$\mathbf{O(nu*nu^2)}$了。

（2）按照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。

如果能预先确定矩阵M中每一列（即T中每一行）的第一个非零元在b.data中的准确位置，则对a.data中的三元组依次转置时，便可直接放到b.data中恰当的位置上。

需附设num和cpot两个向量。num[col]表示矩阵M中第col列中非零元的个数，cpot[col]指示M中第col列的第一个非零元在b.data中的恰当位置。

$$\begin{cases} \text{cpot}[1]=1; \\ \text{cpot}[\text{col}]=\text{cpot}[\text{col}-1]+\text{num}[\text{col}-1] \end{cases}$$

例如：下面稀疏矩阵的三元组表示对应各向量值如下：

1	2	15
1	5	-5
2	2	-7
3	1	36
3	4	28

col	1	2	3	4	5
Num[col]	1	2	0	1	1
Cpot[col]	**1**	**2**	**4**	**4**	**5**

```
cpot[1] = 1 ;
for (col=2 ; col<=M.nu ; ++col)
  cpot[col] = cpot[col-1] + num[col-1] ;
```

该转置方法称为快速转置，如算法5.2所示。

```
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];//求M中每一
    cpot[1] = 1;                                    //列非零元个数
    for (col=2; col<=M.nu; ++col)
       cpot[col] = cpot[col-1] + num[col-1];
    for (p=1; p<=M.tu; ++p) {     ...转置矩阵元素    }
  } // if
  return OK;
} // FastTransposeSMatrix
```

算法 5.2

转置矩阵元素：

```
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]
```

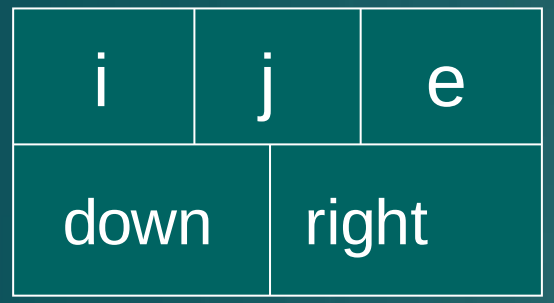
分析算法FastTransposeSMatrix的时间复杂度：

```
for (col=1; col<=M.nu; ++col) … …

for (t=1; t<=M.tu; ++t) … …

for (col=2; col<=M.nu; ++col) … …

for (p=1; p<=M.tu; ++p) … …
```

时间复杂度为: O(M.nu+M.tu)

三、 十字链表

对稀疏矩阵的每个非零元，建立一个结点。结点形式如下：

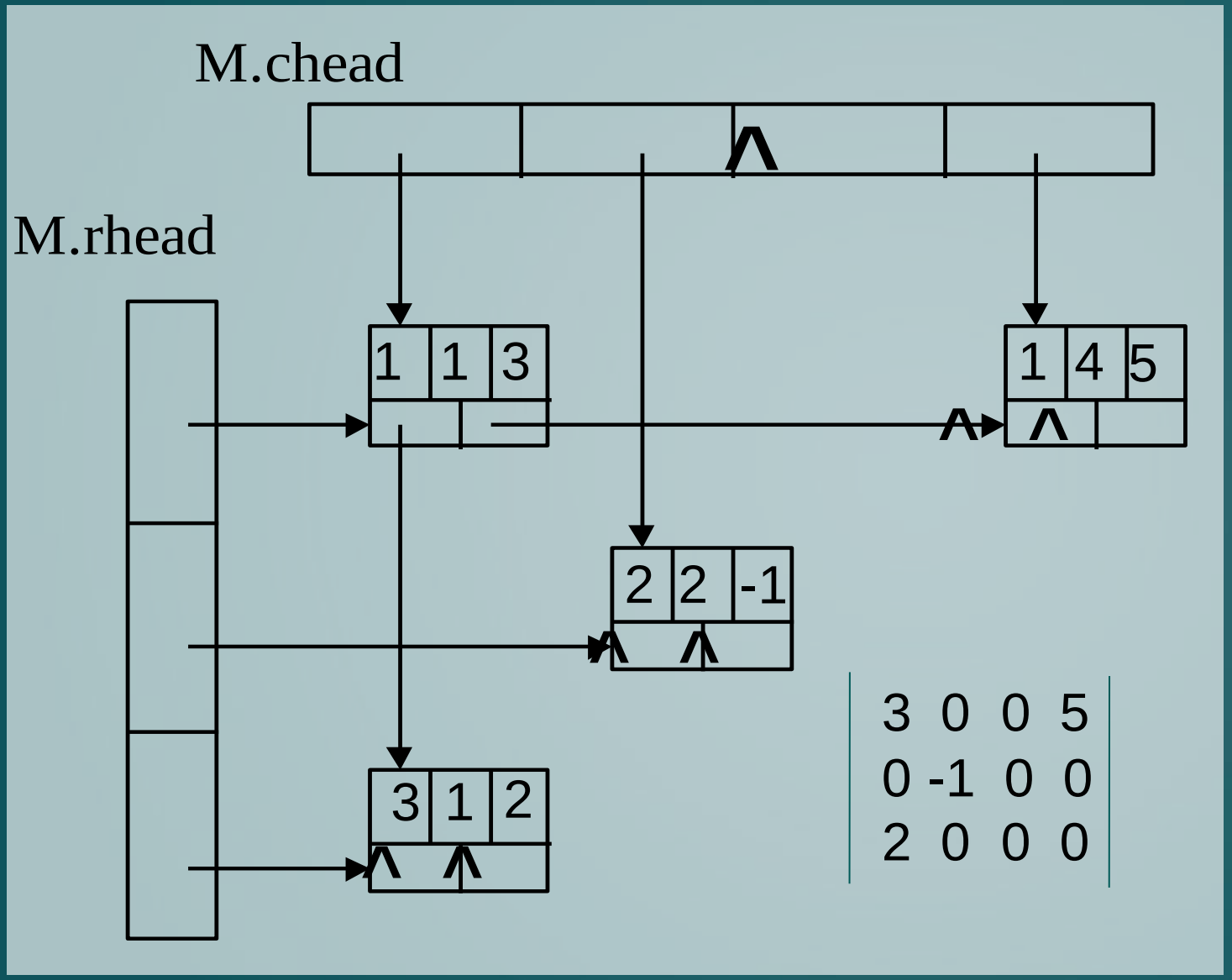


其中：i、j和e分别表示该非零元所在的行、列和非零元的值。

向右域right指向同一行中下一个非零元，向下域down指向同一列中下一个非零元。

每个非零元既是某个行链表中的一个结点，又是某个列链表中的一个结点，整个矩阵构成一个十字交叉的链表。

例如：



```
typedef struct OLNode{
    int  I，j；   //该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；  //行和列链表头指针向量基址由
   CreateSMatrix分配
    int   mu，nu，tu；  //稀疏矩阵的行数、列数和非零元个数
  } CrossList；
```