第5章 数组和稀疏矩阵

数组是由相同性质的数据元素组成的，可以看成是线性表的推广。本章介绍数组的概念、几种特殊矩阵的压缩存储和稀疏矩阵的相关算法。

5.1 数 组

5.1.1 数组的定义

扫一扫

视频讲解

一维数组是 $n(n>1)$ 个相同性质的数据元素 $a_1,a_2,\cdots,a_n$ 构成的有限序列，它本身就是一个线性表。**二维数组**可以看成是这样的一个线性表，它的每个数据元素也是一个线性表。例如，如图 5.1 所示的二维数组 $\boldsymbol{A}$ 以 m 行 n 列的矩阵形式表示，它可以看成是一个线性表：

$$\boldsymbol{A}=(A_1,A_2,\cdots,A_i,\cdots,A_m)$$

其中每个数据元素 A_i 也是一个线性表：

$$A_i=(a_{i,1},a_{i,2},\cdots,a_{i,n})$$

$$\boldsymbol{A}_{m\times n}=\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix}$$

图 5.1 二维数组的逻辑结构表示

对于 $d(d>1)$ 维数组，数据元素之间有 d 个关系（如二维数组中数据元素之间有行和列两个关系），但每个关系仍是线性关系。例如，如图 5.1 所示的二维数组 $\boldsymbol{A}$ 可以采用二元组形式化描述如下。

$$\begin{aligned}
&\boldsymbol{A}=(D,R)\\
&D=\{a_{1,1},a_{1,2},\cdots,a_{1,n},\cdots,a_{m,1},\cdots,a_{m,n}\}\\
&R=\{\mathrm{Row},\mathrm{Col}\}\\
&\mathrm{Row}=\{<a_{1,1},a_{1,2}>,<a_{1,2},a_{1,3}>,\cdots,<a_{1,n-1},a_{1,n}>,<a_{2,1},a_{2,2}>,<a_{2,2},a_{2,3}>,\cdots,\\
&\qquad <a_{2,n-1},a_{2,n}>,\cdots,<a_{m,1},a_{m,2}>,<a_{m,2},a_{m,3}>,\cdots,<a_{m,n-1},a_{m,n}>\}\\
&\mathrm{Col}=\{<a_{1,1},a_{2,1}>,<a_{2,1},a_{3,1}>,\cdots,<a_{m-1,1},a_{m-2,1}>,<a_{1,2},a_{2,2}>,<a_{2,2},a_{3,2}>,\cdots,\\
&\qquad <a_{m-1,2},a_{m,2}>,\cdots,<a_{1,n},a_{2,n}>,<a_{2,n},a_{3,n}>,\cdots,<a_{m-1,n},a_{m,n}>\}
\end{aligned}$$

其中，Row 是行关系，Col 是列关系。这种规则可以推广到三维及以上的数组。

通常，数组的基本运算主要有存取元素值。

几乎所有的高级程序设计语言都实现了数组数据结构，称为数组类型。在 C/C++语言中，数组类型具有如下特点。

（1）一个数组中所有元素具有相同的数据类型。

（2）数组一旦被定义，它的维数和每维大小就不再改变。

（3）数组中每个元素对应唯一的下标，可以通过该下标对元素进行存取操作。

5.1.2 数组的存储结构

扫一扫

视频讲解

由于数组一般不做插入或删除操作，也就是说，一旦建立了数组，其数据元素个数和元素之间的关系就不再发生变动，所以数组通常采用顺序存储结构。

由于内存的存储单元是一维结构，而数组是多维的，则用一组连续存储单元存放数组的元素就有个次序约定问题。

以二维数组为例，在C/C++语言中，由于数组下标从0开始，所以除特别指出外，后面的数组表示均统一为下标从0开始。

二维数组的存储次序有按行优先和按列优先两种方式，按行优先存储的形式如图5.2所示，假设每个元素占 k 个存储单元，$\mathrm{LOC}(a_{0,0})$ 表示 $a_{0,0}$ 元素的存储地址，对于元素 $a_{i,j}$，其存储地址为：

$$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{0,0})+(i\times n+j)\times k$$

k：每个元素占 k 个存储单元

j：在第 i 行中，$a_{i,j}$ 前面 j 个元素

$i\times n$：$a_{i,j}$ 前面有 $0\sim i-1$ 共 i 行，每行 n 个元素，共有 $i\times n$ 个元素

$$\boldsymbol{A}_{m\times n}=\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix} \Rightarrow \boxed{a_{0,0}a_{0,1}\cdots a_{0,n-1}}\ \boxed{a_{1,0}a_{1,1}\cdots a_{1,n-1}}\ \cdots\ \boxed{a_{m-1,0}\ a_{m-1,1}\cdots a_{m-1,n-1}}$$

第0行　第1行　第m−1行

图5.2　按行优先存储

按列优先存储的形式如图5.3所示，对于元素 $a_{i,j}$，其存储地址为：

$$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{0,0})+(j\times m+i)\times k$$

k：每个元素占 k 个存储单元

i：在第 j 列中，$a_{i,j}$ 前面 i 个元素

$j\times m$：$a_{i,j}$ 前面有 $0\sim j-1$ 共 j 列，每列 m 个元素，共有 $j\times m$ 个元素

$$\boldsymbol{A}_{m\times n}=\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m-1,0} & a_{m-1,1} & \cdots & a_{m-1,n-1} \end{bmatrix} \Rightarrow \boxed{a_{0,0}a_{1,0}\cdots a_{m-1,0}}\ \boxed{a_{0,1}a_{1,1}\cdots a_{m-1,0}}\ \cdots\ \boxed{a_{0,n-1}\ a_{1,n-1}\cdots a_{m-1,n-1}}$$

第0列　第1列　第n−1列

图5.3　按列优先存储

【例5.1】 对于二维数组 $\boldsymbol{A}[0..2][0..5]$，当按行优先存储时，元素 $A[2][3]$ 是第几个元素？当按列优先存储时，元素 $A[2][4]$ 是第几个元素？

解 这里 $m=3,n=6$，当按行优先存储时，每行6个元素，元素 $A[2][3]$ 的前面有两行计12个元素，在第2行中前面有三个元素，所以元素 $A[2][3]$ 是 $12+3+1=16$ 个元素。

当按列优先存储时，每列3个元素，元素 $A[2][4]$ 的前面有4列计12个元素，在第4列中前面有两个元素，所以元素 $A[2][4]$ 是 $12+2+1=15$ 个元素。

扫一扫

视频讲解

5.1.3　数组的算法设计示例

本节通过两个示例说明数组算法设计。

【例5.2】 设计一个算法，实现一个 $m\times n$ 的整型数组 $\boldsymbol{A}$ 的转置运算。

解 对于一个 $m\times n$ 的数组 $\boldsymbol{A}_{m\times n}$，其转置矩阵是一个 $n\times m$ 的矩阵 $\boldsymbol{B}_{n\times m}$，且 $B[i][j]=A[j][i]$，$0\leqslant i<m$，$0\leqslant j<n$，对应的算法如下。

```
void TransMat(int A[ ][MAX], int B[ ][MAX],int m,int n)
```

```
{   int i,j;
    for (i=0;i<m;i++)
    {   for (j=0;j<n;j++)
            B[i][j]=A[j][i];
    }
}
```

其中，MAX 为常量，表示二维数组的最大行、列数。

【例 5.3】 设计一个算法，实现一个 $m\times n$ 的整型数组 A 和一个 $n\times k$ 的整型数组 B 的相乘运算。

解 设 $\boldsymbol{C}=\boldsymbol{A}\times\boldsymbol{B}$，其中，$\boldsymbol{A}$ 是 $m\times n$ 矩阵，$\boldsymbol{B}$ 是 $n\times k$ 矩阵，则 $\boldsymbol{C}$ 为 $m\times k$ 矩阵，且 $C[i][j]=\sum_{l=0}^{n-1}A[i][l]\times B[l][j]$。对应的算法如下。

```
void MutMat(int A[ ][MAX],int B[ ][MAX],int C[ ][MAX],int m,int n,int k)
{   int i,j,l;
    for (i=0;i<m;i++)
    {   for (j=0;j<k;j++)
        {   C[i][j]=0;
            for (l=0;l<n;l++)
                C[i][j]+=A[i][l]*B[l][j];
        }
    }
}
```

5.2 特殊矩阵的压缩存储

特殊矩阵是指非零元素或零元素的分布有一定规律的矩阵，为了节省存储空间，特别是在高阶矩阵的情况下，可以利用特殊矩阵的规律，对它们进行压缩存储，也就是说，使多个相同的非零元素共享同一个存储单元，对零元素不分配存储空间。特殊矩阵的主要形式有对称矩阵、对角矩阵等，它们都是方阵，即行数和列数相同。

扫一扫

视频讲解

1. 对称矩阵的压缩存储

若一个 n 阶方阵 $\boldsymbol{A}=\{a_{i,j}\}$ 中的元素满足 $a_{i,j}=a_{j,i}(0\leqslant i,j\leqslant n-1)$，则称其为 n 阶**对称矩阵**。

一个 n 阶方阵的所有元素 $a_{i,j}$ 可以根据行、列下标划分为三部分，即上三角部分（$i<j$）、主对角线部分（$i=j$）和下三角部分（$i>j$），如图 5.4 所示。

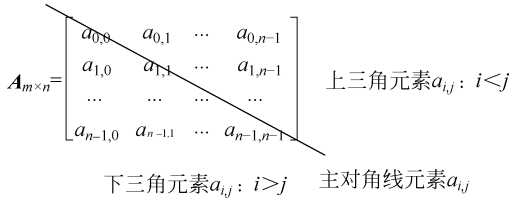


图 5.4 一个 n 阶方阵

如果直接采用二维数组存储对称矩阵，占用的内存空间为 n^2 个元素大小。由于对称矩阵的元素关于主对角线对称，因此在存储时可只存储其上三角和主对角线部分元素，或者下三角和主对角线部分的元素，使得对称的元素共享同一存储空间。这样，就可以将 n^2 个元素压缩存储到 $n(n+1)/2$ 个元素的空间中。不失一般性，以行序为主序存储其下三角和主对角线部分的元素，如图 5.5 所示。

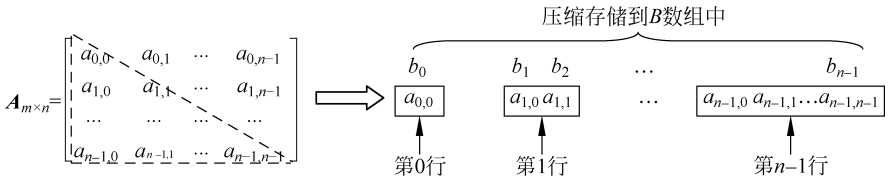


图 5.5　对称矩阵的压缩存储

假设以一维数组 $B=\{b_k\}$ 作为 n 阶对称矩阵 $\boldsymbol{A}$ 的压缩存储结构，在 B 中只存储对称矩阵 $\boldsymbol{A}$ 的下三角和主对角线部分的元素 $a_{i,j}(i\geqslant j)$，这样 B 中的元素个数为 $n(n+1)/2$。

(1) 将 $\boldsymbol{A}$ 中下三角和主对角线部分的元素 $a_{i,j}(i\geqslant j)$ 存储在 B 数组的 b_k 元素中。那么 k 和 i、j 之间是什么关系呢？

对于这样的元素 $a_{i,j}$，求出它前面共存储的元素个数。不包括第 i 行，它前面共有 i 行(行下标为 0～$i-1$，第 0 行有 1 个元素，第 1 行有 2 个元素，…，第 $i-1$ 行有 i 个元素)，则这 i 行共有 $1+2+\cdots+i=i(i+1)/2$ 个元素；在第 i 行中，元素 $a_{i,j}$ 的前面也有 j 个元素，则元素 $a_{i,j}$ 之前共有 $i(i+1)/2+j$ 个元素，所以有 $k=i(i+1)/2+j$。

(2) 对于 $\boldsymbol{A}$ 中上三角部分元素 $a_{i,j}(i<j)$，它的值等于 $a_{j,i}$，而 $a_{j,i}$ 元素在 B 中的存储位置 $k=j(j+1)/2+i$。

归纳起来，对于 $\boldsymbol{A}$ 中任一元素 $a_{i,j}$ 和 B 中元素 b_k 之间存在着如下对应关系：

$$k=\begin{cases} i(i+1)/2+j, & \text{当 } i\geqslant j \text{ 时} \\ j(j+1)/2+i, & \text{当 } i<j \text{ 时} \end{cases}$$

2. 三角矩阵的压缩存储

有些非对称的矩阵也可借用上述方法存储，如 n 阶下(上)三角矩阵。所谓 n 阶**下(上)三角矩阵**，是指矩阵的上(下)三角部分(不包括主对角线)中的元素均为常数 c 的 n 阶方阵。

设以一维数组 $B=\{b_k\}$ 作为 n 阶三角矩阵 $\boldsymbol{A}$ 的存储结构，B 中的元素个数为 $n(n+1)/2+1$(其中常数 c 占用一个元素空间)，则 $\boldsymbol{A}$ 中任一元素 $a_{i,j}$ 和 B 中元素 b_k 之间存在着如下对应关系。

上三角矩阵：

$$k=\begin{cases} i(2n-i+1)/2+j-i, & \text{当 } i\leqslant j \text{ 时} \\ n(n+1)/2, & \text{当 } i>j \text{ 时} \end{cases}$$

下三角矩阵：

$$k=\begin{cases} i(i+1)/2+j, & \text{当 } i\leqslant j \text{ 时} \\ n(n+1)/2, & \text{当 } i>j \text{ 时} \end{cases}$$

其中，B 的最后元素 $b_{n(n+1)/2}$ 中存放常数 c。

3. 对角矩阵的压缩存储

若一个 n 阶方阵 $\boldsymbol{A}$ 满足其所有非零元素都集中在以主对角线为中心的带状区域中，则称其为 n 阶**对角矩阵**。其主对角线上下方各有 b 条次对角线，称 b 为矩阵半带宽，$(2b+1)$为矩阵的带宽。对于半带宽为 $b(0\leqslant b\leqslant (n-1)/2)$的对角矩阵，其 $|i-j|\leqslant b$ 的元素 $a_{i,j}$ 不为零，其余元素为零。如图 5.6 所示是半带宽为 b 的对角矩阵示意图。

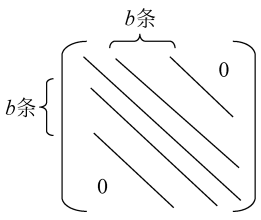


图 5.6 半带宽为 b 的对角矩阵

对于 $b=1$ 的三对角矩阵 $\boldsymbol{A}$，只存储其非零元素，并按行优先存储到一维数组 B 中，将 $\boldsymbol{A}$ 的非零元素 $a_{i,j}$ 存储到 B 的元素 b_k 中。$\boldsymbol{A}$ 中第 0 行和第 $n-1$ 行都只有两个非零元素，其余各行有三个非零元素。对于不在第 0 行的非零元素 $a_{i,j}$ 来说，在它前面存储了矩阵的前 i 行元素，这些元素的总数 k 为 $2+3(i-1)$。

(1) 若 $a_{i,j}$ 是第 i 行中需要存储的第 1 个元素，则 $k=2+3(i-1)=3i-1$，此时，$j=i-1$，即 $k=2i+i-1=2i+j$。

(2) 若 $a_{i,j}$ 是第 i 行中需要存储的第 2 个元素，则 $k=2+3(i-1)+1=3i$，此时，$i=j$，即$k=2i+i=2i+j$。

(3) 若 $a_{i,j}$ 是第 i 行中需要存储的第 3 个元素，则 $k=2+3(i-1)+2=3i+1$，此时，$j=i+1$，即 $k=2i+i+1=2i+j$。

归纳起来有：$k=2i+j$。

以上讨论的对称矩阵、三角矩阵、对角矩阵的压缩存储方法，是把有一定分布规律的值相同的元素(包括 0)压缩存储到一个存储空间中。这样的压缩存储只需在算法中按相应公式做一映射即可实现矩阵元素的随机存取。

5.3 稀疏矩阵

一个阶数较大的矩阵中的非零元素个数 s 相对于矩阵元素的总个数 t 非常小时，即 $s\ll t$ 时，称该矩阵为**稀疏矩阵**。例如一个 100×100 的矩阵，若其中只有 100 个非零元素，就可称其为稀疏矩阵。

稀疏矩阵的压缩存储方法是只存储非零元素，主要有三元组和十字链表两种方法。

扫一扫

视频讲解

5.3.1 稀疏矩阵的三元组表示

由于稀疏矩阵中非零元素的分布没有任何规律，所以在存储非零元素时还必须同时存储该非零元素所对应的行、列下标。这样稀疏矩阵中的每一个非零元素需由一个三元组$(i,j,a_{i,j})$唯一确定，稀疏矩阵中的所有非零元素构成一个三元组线性表，将其采用顺序存储结构存储，称为稀疏矩阵的**三元组表示**。

如图 5.7 所示是一个 6×7 阶稀疏矩阵 $\boldsymbol{A}$(为图示方便，所取的行列数都很小)及其对应的三元组表示。

$$A_{6\times7}=\begin{bmatrix}0&0&1&0&0&0&0\\0&2&0&0&0&0&0\\3&0&0&0&0&0&0\\0&0&0&5&0&0&0\\0&0&0&0&6&0&0\\0&0&0&0&0&7&4\end{bmatrix}$$

i	j	$a_{i,j}$
0	2	1
1	1	2
2	0	3
3	3	5
4	4	6
5	5	7
5	6	4

图 5.7　一个稀疏矩阵 **A** 及其对应的三元组表示

完整的稀疏矩阵三元组表示的类型声明如下。

```
#define MaxSize 100                          //矩阵中非零元素的最多个数
typedef struct
{    int r;                                  //行号
     int c;                                  //列号
     ElemType d;                             //元素值为 ElemType 类型
} TupNode;                                   //三元组定义
typedef struct
{    int rows;                               //行数
     int cols;                               //列数
     int nums;                               //非零元素个数
     TupNode data[MaxSize];
} TSMatrix;                                  //三元组顺序表定义
```

其中，data 域中表示的非零元素通常以行序为主序顺序排列，它是一种下标按行有序的存储结构。这种有序存储结构可简化大多数矩阵运算算法。下面的讨论均假设 data 域按行有序存储。

稀疏矩阵运算通常包括矩阵转置、矩阵加、矩阵减、矩阵乘等。这里仅讨论基本运算算法。

1. 从一个二维稀疏矩阵创建其三元组表示

算法的思路是，以行序方式扫描二维稀疏矩阵 **A**，将其非零的元素插入三元组 t 中。对应的算法如下。

```
void CreatMat(TSMatrix &t,ElemType A[M][N])
{    int i,j;
     t.rows=M;t.cols=N;t.nums=0;
     for (i=0;i<M;i++)
     {    for (j=0;j<N;j++)
          {    if (A[i][j]!=0)                 //只存储非零元素
               {    t.data[t.nums].r=i;t.data[t.nums].c=j;
                    t.data[t.nums].d=A[i][j];t.nums++;
               }
          }
     }
}
```

2. 三元组元素赋值

该运算的功能是对于稀疏矩阵 **A** 执行 $A[i][j]=x$ 操作(通常 x 是一个非零值)。由

于 **A** 不是直接采用二维数组存储而是三元组 t 存储的，需要在 t 中实现该赋值操作。

算法思路是，根据 i、j 下标先在三元组 t 中找到适当的位置 k，如果该元素原来就是非零元素(在 t 中找到了该元素)，直接将元素值修改为 x；否则(在 t 中找不到该元素)将 k～t.nums 个元素后移一个位置，将指定元素 x 插入 t.data[k]处。对应的算法如下。

```
int Value(TSMatrix &t,ElemType x,int i,int j)
{   int k=0,k1;
    if (i>=t.rows || j>=t.cols)
        return 0;                                          //参数错误时返回 0
    while (k<t.nums && i>t.data[k].r) k++;                 //查找行
    while (k<t.nums && i==t.data[k].r && j>t.data[k].c) k++;      //查找列
    if (k<t.nums && t.data[k].r==i && t.data[k].c==j)              //存在这样的元素
        t.data[k].d=x;
    else                                                   //不存在这样的元素时插入一个元素
    {   for (k1=t.nums-1;k1>=k;k1--)
        {   t.data[k1+1].r=t.data[k1].r;
            t.data[k1+1].c=t.data[k1].c;
            t.data[k1+1].d=t.data[k1].d;
        }
        t.data[k].r=i;t.data[k].c=j;t.data[k].d=x;
        t.nums++;
    }
    return 1;                                              //成功时返回 1
}
```

3. 将指定位置的元素值赋给变量

该运算的功能是对于稀疏矩阵 **A** 执行 $x=A[i][j]$操作。由于 **A** 不是直接采用二维数组存储而是三元组 t 存储的，需要在 t 中实现该取值操作。

算法思路是，根据 i、j 下标先在三元组 t 中找到指定的位置 k，如果该元素原来是非零元素(在 t 中找到了该元素)，直接将对应的元素值赋给 x；否则(在 t 中找不到该元素)说明该元素是一个零元素，置 $x=0$。对应的算法如下。

```
int Assign(TSMatrix t,ElemType &x,int i,int j)
{   int k=0;
    if (i>=t.rows || j>=t.cols)
        return 0;                                          //参数错误时返回 0
    while (k<t.nums && i>t.data[k].r) k++;                 //查找行
    while (k<t.nums && i==t.data[k].r && j>t.data[k].c) k++;      //查找列
    if (k<t.nums && t.data[k].r==i && t.data[k].c==j)
        x=t.data[k].d;
    else
        x=0;                                               //在三元组中没有找到表示是零元素
    return 1;                                              //成功时返回 1
}
```

4. 输出三元组运算算法

该运算的功能是输出稀疏矩阵对应的三元组表示。直接从头到尾扫描三元组 t，依次输出元素值。对应的算法如下。

```
void DispMat(TSMatrix t)
```

```
{   int i;
    if (t.nums<=0)                                    //没有非零元素时返回
        return;
    printf("\t%d\t%d\t%d\n",t.rows,t.cols,t.nums);
    printf("\t--------------------\n");
    for (i=0;i<t.nums;i++)
        printf("\t%d\t%d\t%d\n",t.data[i].r,t.data[i].c,t.data[i].d);
}
```

提示：将稀疏矩阵三元组表示的类型声明及其基本运算函数存放在 TSMatrix.cpp 文件中。

当稀疏矩阵三元组表示的基本运算设计好后，给出以下主函数调用这些基本运算函数，读者可以对照程序执行结果进行分析，进一步体会稀疏矩阵三元组表示的各种操作的实现过程。

```
#define M 6
#define N 7
#include "TSMatrix.cpp"                               //包括稀疏矩阵三元组表示的基本运算算法
int main()
{   TSMatrix t;
    ElemType x;
    ElemType a[M][N]={{0,0,1,0,0,0,0},{0,2,0,0,0,0,0},{3,0,0,0,0,0,0},
                      {0,0,0,5,0,0,0},{0,0,0,0,6,0,0},{0,0,0,0,0,7,4}};
    CreatMat(t,a);
    printf("三元组 t 表示:\n"); DispMat(t);
    printf("执行 A[4][1]=8\n");
    Value(t,8,4,1);
    printf("三元组 t 表示:\n"); DispMat(t);
    printf("求 x=A[4][1]\n");
    Assign(t,x,4,1);
    printf("x=%d\n",x);
}
```

上述程序的执行结果如下。

```
三元组 t 表示:
    6    7    7
    ------------
    0    2    1
    1    1    2
    2    0    3
    3    3    5
    4    4    6
    5    5    7
    5    6    4
执行 A[4][1]=8
三元组 t 表示:
    6    7    8
    ------------
    0    2    1
    1    1    2
```

```
    2    0    3
    3    3    5
    4    1    8
    4    4    6
    5    5    7
    5    6    4
求 x=A[4][1]
x=8
```

【例 5.4】 一个稀疏矩阵采用三元组表示压缩存储后，和直接采用二维数组存储相比会失去_____特性。

A. 顺序存储　　B. 随机存取　　C. 输入输出　　D. 以上都不对

解 当稀疏矩阵直接采用二维数组存储时，它具有随机存取特性。而采用三元组表示（或十字链表表示）时，对于给定稀疏矩阵中某个元素的下标(i,j)，在其三元组表示中查找对应元素值的时间不再是$O(1)$，所以尽管三元组表示占用的空间会减少，但不再具有随机存取特性。本题答案为 B。

扫一扫

视频讲解

5.3.2 稀疏矩阵的十字链表表示

十字链表是稀疏矩阵的一种链式存储结构。

对于一个$m\times n$的稀疏矩阵，每个非零元素用一个结点表示，该结点中存放该零元素的行号、列号和元素值。同一行中的所有非零元素结点链接成一个带行头结点的行循环单链表，将同一列的所有非零元素结点链接成一个带列头结点的列循环单链表。之所以采用循环单链表，是因为矩阵运算中常常是一行（列）操作完后进行下一行（列）操作，最后一行（列）操作完后进行第一行（列）操作。

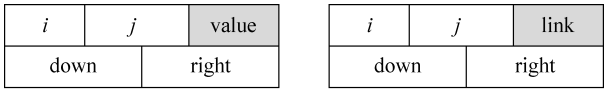
这样对稀疏矩阵的每个非零元素结点来说，它既是某个行链表中的一个结点，同时又是某个列链表中的一个结点，每个非零元素就好比在一个十字路口，由此称作**十字链表**。

每个非零元素结点的类型设计成如图 5.8(a)所示的结构，其中，i、j、value 分别代表非零元素所在的行号、列号和相应的元素值；down 和 right 分别称为向下指针和向右指针，分别用来链接同列中和同行中的下一个非零元素结点。

这样行循环单链表个数为m（每一行对应一个行循环单链表），列循环单链表个数为n（每一列对应一个列循环单链表），那么行列头结点的个数就是$m+n$。实际上，行头结点与列头结点是共享的，即$h[i]$表示第i行循环单链表的头结点，同时也是第i列循环单链表的头结点，这里$0\leqslant i<\mathrm{MAX}\{m,n\}$，即行列头结点的个数是$\mathrm{MAX}\{m,n\}$，所有行列头结点的类型与非零元素结点类型相同。

另外，将所有行列头结点再链接起来构成一个带头结点的循环单链表，这个头结点称为总头结点即 hm，通过 hm 来标识整个十字链表。

总头结点的类型设计成如图 5.8(b)所示的结构（之所以这样设计，是为了与非零元素结点类型一致，这样在整个十字链表中采用指针扫描所有结点时更方便），它的 link 域指向第一个行列头结点，其i、j域分别存放稀疏矩阵的行数m和列数n，而 down 和 right 域没有作用。



(a) 结点结构　　(b) 头结点结构

图 5.8　十字链表结点结构

从中看出，在 $m \times n$ 的稀疏矩阵的十字链表存储结构中，有 m 个行循环单链表，n 个列循环单链表，另加一个行列头结点构成的循环单链表，总的循环单链表个数是 $m+n+1$，总的头结点个数是 MAX$\{m,n\}+1$。

设稀疏矩阵如下：

$$\boldsymbol{B}_{3\times4}=\begin{bmatrix}1&0&0&2\\0&0&3&0\\0&0&0&4\end{bmatrix}$$

对应的十字链表如图 5.9 所示。为图示清楚，把每个行列头结点分别画成两个，实际上行列值相同的头结点只有一个。

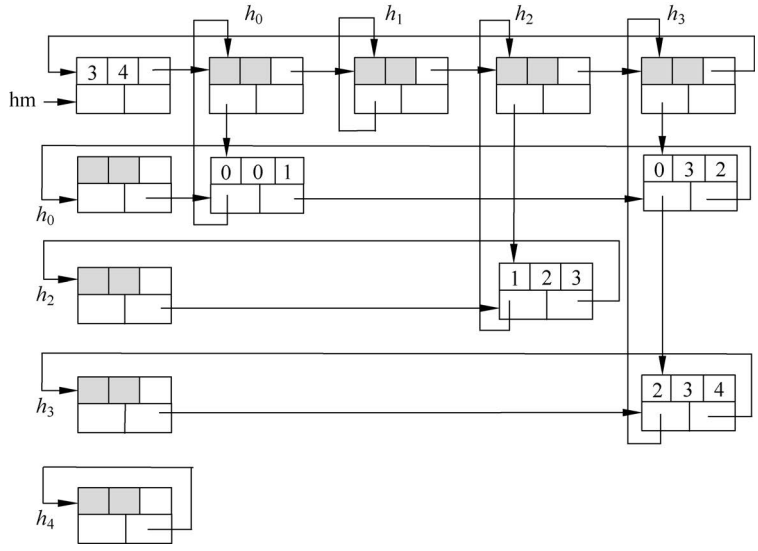


图 5.9　一个稀疏矩阵的十字链表

十字链表结点结构和头结点合起来声明的结点类型如下。

```
#define M 3                                    //矩阵行数
#define N 4                                    //矩阵列数
#define Max ((M)>(N)?(M):(N))                  //矩阵行列中较大者
typedef struct mtxn
{    int i;                                    //行号
     int j;                                    //列号
     struct mtxn *right, *down;                //向右和向下的指针
     union
     {    ElemType value;                      //存放非零元素值
          struct mtxn *link;
```

```
    } tag;
} MatNode;                                    //十字链表类型定义
```

有关稀疏矩阵采用十字链表表示时的相关运算算法与三元组表示类似，但更复杂，这里不再讨论。

【例 5.5】 一个 m 行 n 列的稀疏矩阵采用十字链表表示时，其中循环单链表的个数为_____。

A. $m+1$　　B. $n+1$　　C. $m+n+1$　　D. MAX$\{m,n\}+1$

解 稀疏矩阵采用十字链表表示时，每个非零元素对应一个结点，每行的所有结点构成一个带行头结点的循环单链表，每列的所有结点构成一个带列头结点的循环单链表，这些行列循环单链表的头结点是共享的，即第 i 行和第 i 列的头结点是共享的，所以行列头结点个数为 MAX$\{m,n\}$。最后将所有头结点合起来构成一个带总头结点的循环单链表。因此其中循环单链表的个数为 $m+n+1$，而头结点的个数为 MAX$\{m,n\}+1$。本题答案为 C。

小　结

(1) 数组是线性表的推广，$d(d\geqslant 1)$维数组中存在 d 个线性关系。

(2) 数组通常采用顺序存储方法，分为以行优先和以列优先两种存储方式。

(3) 特殊矩阵不是指具有特殊用途的矩阵，是指一类元素值分布具有某种规律的矩阵，可以采用压缩存储方法。

(4) 对称矩阵、三角矩阵和对角矩阵采用压缩存储的目的是节省内存空间。

(5) 数组通常采用顺序存储结构，具有随机存取特性。

(6) 稀疏矩阵的压缩存储方式主要有三元组和十字链表表示，前者属顺序存储结构，后者属链式存储结构。

(7) 稀疏矩阵采用三元组或十字链表压缩存储方式后，不再具有随机存取特性。

练　习　题

扫一扫

练习题

扫一扫

自测题

上机实验题

扫一扫

上机实验题

不需要学习计算机基础课程?

关于学校里面开设的课程,大家可能会觉得不够时髦,不够酷,净是一些计算机组成原理、数据结构等老掉牙的课程,远没有什么 SPRING 框架来得过瘾。呵呵,不过根据我的经验,工作几年以后,大家可能会觉得,最值钱的,恰恰是这些最土气的课程。用框架,永远不算本事,也没有什么核心竞争力,哪天框架死了,你就死了。而另一方面,框架也是人做的,大家以为,做框架需要哪些知识?是不是上述 old 的知识?

——摘自著名的 IT 达人肖舸等著《IT 学生解惑真经》