第 5 章　数组、特殊矩阵

5.1　数组的逻辑结构及 ADT 描述

5.1.1　数组的概念

数组是数据类型相同的一组数据元素的有序集合,元素之间具有线性关系。元素在集合中的序是由一组称为“下标”的值确定的,一个数据元素称为一个数组元素。数组是计算机编程中的重要概念,运用数组可以方便地处理大规模的数据。

一个数据元素的位置由多个下标值确定,即参与多个线性关系,每个关系上都有前驱和后继。

数组的维数：确定一个数据元素在集合中的下标的个数。

5.1.2　数组的逻辑结构

1. 一维数组的数据结构

一维数组是由数字组成的以单纯的排序结构排列的结构单一的数组,是计算机程序中最基本的数组。当数组中每个元素都只带有一个下标时,称这样的数组为一维数组。其定义方式如下。

$$1-\mathrm{ARRAY}=(D,R)$$
$$D=\{a_i,i=c_1\cdots d_1\}$$
$$R=\{N\}$$
$$N=\{<a_{i-1},a_i>a_{i-1},a_i\in D_0,c_1+1\leqslant i\leqslant d_1\}$$

一维数组的元素个数为 d_1-c_1+1。

2. 二维数组的数据结构

二维数组也是一种基本的数组,其本质上还是一个一维数组,只是它的数据元素又是一个一维数组。二维数组是一个包含元素的表格结构,其中每个元素由两个索引(行索引和列索引,也称为行号和列号)指定。其定义方式如下。

$$2-\mathrm{ARRAY}=(D,R)$$
$$D=\{a_{ij},i=c_1\cdots d_1,j=c_2\cdots d_2,a_{ij}\in D_0\}$$
$$R=\{\mathrm{ROW},\mathrm{COL}\}$$
$$\mathrm{ROW}=\{<a_{ij},a_{ij+1}>a_{ij},a_{ij+1}\in D_0,c_1\leqslant i\leqslant d_1,c_2\leqslant j\leqslant d_2-1\}$$
$$\mathrm{COL}=\{<a_{ij},a_{i+1j}>a_{ij},a_{i+1j}\in D_0,c_1\leqslant i\leqslant d_1-1,c_2\leqslant j\leqslant d_2\}$$

二维数组的元素个数为$(d_1-c_1+1)\times(d_2-c_2+1)$。

3. 三维数组的数据结构

三维数组是一个维度为 3 的数组结构，它是最常见的多维数组，可以用来描述三维空间中的位置和状态。在三维数组中，每个元素可以由三个下标访问，这三个下标通常是三个不同的参量。更具体地说，可以把三维数组看作由多个二维数组堆叠而成的，而二维数组则可以看作由多个一维数组组成的。其定义方式如下。

$3-\mathrm{ARRAY}=(D,R)$

$D=\{a_{ijk},i=c_1\cdots d_1,j=c_2\cdots d_2,k=c_3\cdots d_3,a_{ijk}\in D_0\}$

$R=\{R_1,R_2,R_3\}$

$R_1=\{<a_{ijk},a_{ijk+1}>a_{ijk},a_{ijk+1}\in D_0,c_1\leqslant i\leqslant d_1,c_2\leqslant j\leqslant d_2,c_3\leqslant k\leqslant d_3-1\}$

$R_2=\{<a_{ijk},a_{ij+1k}>a_{ijk},a_{ij+1k}\in D_0,c_1\leqslant i\leqslant d_1,c_2\leqslant j\leqslant d_2-1,c_3\leqslant k\leqslant d_3\}$

$R_3=\{<a_{ijk},a_{i+1jk}>a_{ijk},a_{i+1jk}\in D_0,c_1\leqslant i\leqslant d_1-1,c_2\leqslant j\leqslant d_2,c_3\leqslant k\leqslant d_3\}$

三维数组的元素个数为$(d_1-c_1+1)\times(d_2-c_2+1)\times(d_3-c_3+1)$。

一维数组、二维数组和三维数组的具体形态如图 5.1 所示。

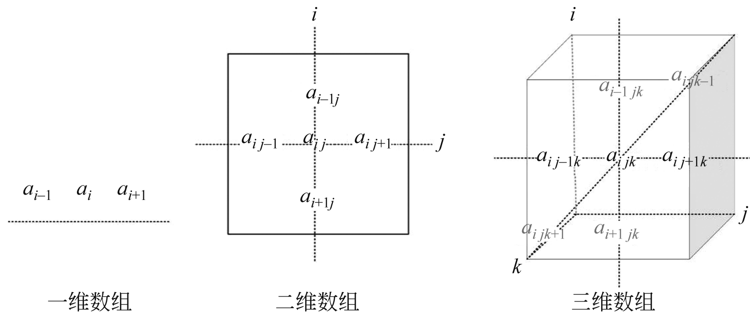


图 5.1　一维数组、二维数组和三维数组的具体形态

通过分析，可以得出以下结论。

(1) 数组中每个数据元素受多个线性关系制约，元素在每个线性关系上都有前驱和后继。

(2) 一个 n 维数组可以看作一个线性表，其每个数据元素是一个 $n-1$ 维的数组。

同理，可以推断出更多维度的数组的概念。

5.1.3　数组的操作

任何一个数组在使用之前，一定要做两件事：声明和初始化，然后才能对其做操作。操作包括数组数据元素的访问、修改，以及遍历数组中的所有数据元素等。本节重点介绍数组数据元素的访问。

1. 数组的声明和初始化

1）一维数组

(1) 声明一个一维数组，需要指定数组的类型，然后在类型后面加上方括号[]，最后是数组的名称。

例 5.1　声明一个一维数组。

```
int[]numbers;          //声明一个整型一维数组
String[]names;         //声明一个字符串型一维数组
```

(2) 一维数组静态初始化是指直接在声明一个一维数组时为该数组元素赋值。一个一维数组在创建后需要被初始化才能使用。我们使用{}进行初始化。初始化可以包括分配内存,并为数组元素赋予初始值。

例 5.2 一维数组静态初始化。

```
int[]numbers={1,2,3}d
String[]names={"Tom","Bob","Alice"};
```

(3) 一维数组动态初始化是指先使用运算符 new,此时只指定数组长度,由系统默认初始化,之后可以单独为每个元素赋值。这里需要注意的是,数组的长度是固定的,一旦创建,就不能更改。

例 5.3 一维数组动态初始化。

```
int[]numbers = new int[3];          //创建一个长度为 3 的一维整型数组
numbers[0] = 1;                     //1 赋给 numbers 数组的第一个值
numbers[1] = 2;                     //2 赋给 numbers 数组的第二个值
numbers[2] = 3;                     //3 赋给 numbers 数组的第三个值
```

2) 二维数组

(1) 声明一个二维数组,需要指定数组的类型,然后在类型后面加上两个方括号[],最后是数组的名称。

例 5.4 声明一个二维数组。

```
int[][]a;            //声明一个整型的二维数组
char[][]b;           //声明一个字符型的二维数组
```

(2) 二维数组静态初始化是指直接在声明一个二维数组时为该数组元素赋值。

例 5.5 二维数组静态初始化。

```
int[][]a={{1,2,3},{4,5,6},{7,8,9}};
```

(3) 二维数组动态初始化是指先使用运算符 new,只指定数组长度,由系统默认初始化,之后可以单独为每个元素赋值。

例 5.6 二维数组动态初始化。

```
int[][]a = new int[][]{{1,2,3},{4,5,6},{7,8,9}};
                                           //创建一个 3 行 3 列的二维整型数组,并为其赋值
char[][]b =  new char[2][3];               //创建一个 2 行 3 列的二维字符型数组
```

2. 访问数组数据元素

一般来说,数组建立后的主要操作就是访问数组数据元素,而访问数据元素的关键在于需要获取数据元素的位置。数组通过下标索引来访问数组中的数据元素。默认情况下,数组索引从 0 开始,最大索引是数组长度减 1。

例 5.7 访问数组元素。

```
System.out.println(names[2]);     //输出“Alice”
System.out.println(a[2][2]);      //输出 9
```

因此，数组的主要操作如下。

（1）数组初始化 create()。

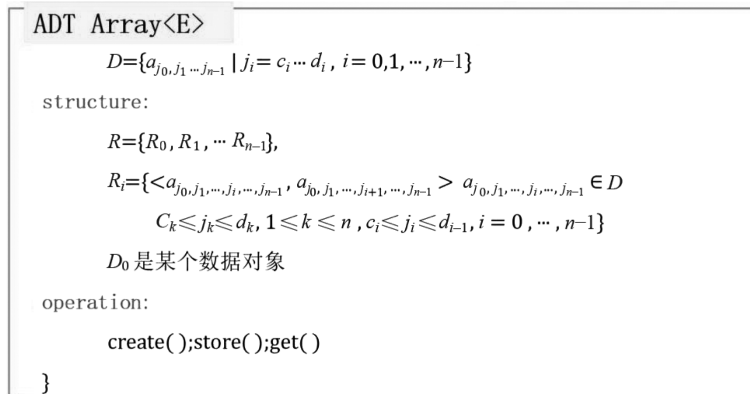
（2）给数组的数据元素赋值 store()。

（3）访问某个数组的数据元素 get()。

一个二维数组在创建后需要被初始化才能使用。我们使用{ }进行初始化。初始化可以包括分配内存，并为数组的数据元素赋予初始值。

5.1.4 数组的 ADT 描述

由前面的分析可以得出数组 ADT 的描述，如图 5.2 所示。



ADT Array<E>

$D=\{a_{j_0,j_1\cdots j_{n-1}} \mid j_i=c_i\cdots d_i,\ i=0,1,\cdots,n-1\}$

structure:

$R=\{R_0,R_1,\cdots R_{n-1}\}$,

$R_i=\{<a_{j_0,j_1,\cdots,j_i,\cdots,j_{n-1}},\ a_{j_0,j_1,\cdots,j_{i+1},\cdots,j_{n-1}}>\ a_{j_0,j_1,\cdots,j_i,\cdots,j_{n-1}}\in D$

$C_k\leqslant j_k\leqslant d_k,\ 1\leqslant k\leqslant n,\ c_i\leqslant j_i\leqslant d_{i-1},\ i=0,\cdots,n-1\}$

D_0 是某个数据对象

operation:

create();store();get()

}

图 5.2 数组的 ADT 描述

5.2 数组的顺序存储结构及运算

绝大多数语言中都是默认行优先存储，如 C、Java 等，但是 FORTAN 默认是列优先。

由于数组数据结构的操作比较少，且没有插入、取出运算，所以一般均采用顺序存储结构。因此，数组的存储结构同一般线性表的顺序存储结构完全相同。用地址连续的存储单元依次存储数组的各个元素。

无论是几维数组，在计算机中都按一维数组来存放，二维数组的顺序存储如图 5.3 所示。

5.2.1 二维数组的顺序存储

二维数组顺序存储通常采用行优先和列优先两种顺序进行存储。

1. 按行优先顺序

按行优先顺序存放是将数组看作若干个行向量。例如，二维数组 $A_{m\times n}$，可以看作 m 个行向量，每个行向量中有 n 个数据元素。二维数组中的每个数据元素由数据元素的两个下标表达式唯一地确定。

a_{00}	a_{01}	…	a_{0j}	…	a_{0n-1}
a_{10}	a_{11}	…		…	a_{1n-1}
…	…	…	…	…	…
a_{i0}	a_{i1}	…	a_{ij}	…	a_{in-1}
…	…	…	…	…	…
…	…	…	…	…	…

图 5.3　二维数组的顺序存储

访问数据元素的关键在于需要获取数据元素的位置。如果按行优先顺序，二维数组的数据元素存储位置应该是

$$\mathrm{LOC}(a_{ij}) = \mathrm{LOC}(a_{00}) + (i \times n + j) \times L$$

其中，L 是每个数据元素所占的存储单元，如图 5.4 所示。

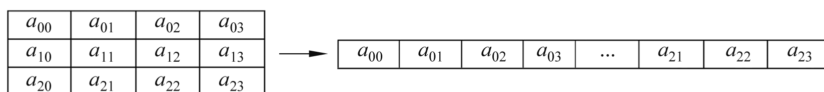


图 5.4　按行优先顺序存储二维数组

2. 按列优先顺序

按列优先顺序存放是将数组看作若干个列向量。例如，二维数组 $A_{m \times n}$，可以看作 n 个列向量，每个列向量中的 m 个数据元素。数组中的每个数据元素由数据元素的两个下标表达式唯一地确定。

如果按列优先顺序，数据元素的存储位置计算如下：

$$\mathrm{LOC}(a_{ij}) = \mathrm{LOC}(a_{00}) + (j \times m + i) \times L$$

其中，L 是每个数据元素所占的存储单元，如图 5.5 所示。

a_{00}	a_{01}	…	a_{0j}	…	a_{0n-1}
a_{10}	a_{11}	…	a_{1j}	…	a_{1n-1}
…	…	…	…	…	…
a_{i0}	a_{i1}	…	a_{ij}	…	a_{in-1}
…	…	…	…	…	…
a_{m-10}	a_{m-11}	…	a_{m-1j}	…	a_{m-in-1}

图 5.5　按列优先顺序存储二维数组

例 5.8　设数组 a[1…60,1…70]的基地址为 2048，每个数据元素占 2 个存储单元，若以列优先顺序存储，则数据元素 a[32,58]的存储地址是多少？

$$\begin{aligned}\mathrm{LOC}(a_{ij}) &= \mathrm{LOC}(a_{00}) + (j \times m + i) \times L \\ &= 2048 + [(58-1) \times 60 + (32-1)] \times 2 \\ &= 8950\end{aligned}$$

5.2.2 三维数组的顺序存储

三维数组的数据元素定位是按照页、行、列来描述，即先按照页来存储，每页中先按行存储，再按列存储。

三维数组 A_{ijk}，各维数据元素个数分别为 m、n、p。其中，i 表示页索引，j 表示行索引，k 表示列索引，如图 5.6 所示。

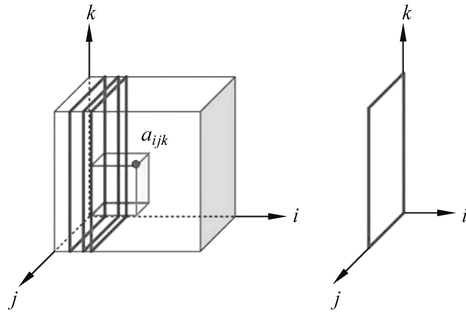


图 5.6 三维数组 A_{ijk}

如果按页优先顺序存储结构，则三维数组 A_{ijk} 各维数据元素个数分别为 m、n、p。按页优先顺序存储，强调先按页、再按行、最后按列存储，则三维数组的数据元素存储位置计算如下：

$$\mathrm{LOC}(a_{ijk})=\mathrm{LOC}(a_{000})+(i\times n\times p+j\times p+k)\times L$$

式中，$i\times n\times p$ 为前 i 页总数据元素的个数，$j\times p$ 为第 i 页的前 j 行总数据元素的个数，k 为第 i 页的第 j 行第 k 列前的数据元素的个数。

5.2.3 n 维数组的顺序存储

对于 n 维数组，各维数组个数为 $m_1,m_2,\cdots,m_n$，下标为 $i_1,i_2,\cdots,i_n$ 的数组元素的存储位置地址如下：

$$\begin{aligned}\mathrm{LOC}(a_{i1,i2,\cdots,in}) &= \mathrm{LOC}(a_{0,0,\cdots,0})+[(i_1\times m_2\times m_3\times\cdots\times m_n)\\ &\quad +i_2\times m_3\times m_4\times\cdots\times m_n+i_{n-1}\times m_n+i_n]\times L\\ &= \mathrm{LOC}(a_{0,0,\cdots,0})+\left\{\sum_{j=1}^{n-1}\left(i_j\times\prod_{k=j+1}^{n}m_k\right)+i_n\right\}\times L\end{aligned}$$

5.2.4 数组的顺序存储小结

从一维到 n 维数组的存储地址计算公式如下。

一维数组顺序存储：$\mathrm{LOC}(a_i)=\mathrm{LOC}(a_0)+i\times L$。

二维数组行优先存储：$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{0,0})+(i\times n+j)\times L$。

二维数组列优先存储：$\mathrm{LOC}(a_{ij})=\mathrm{LOC}(a_{00})+(j\times m+i)\times L$。

三维数组页优先存储：$\mathrm{LOC}(a_{ijk})=\mathrm{LOC}(a_{000})+(i\times n\times p+j\times p+k)\times L$。

n 维数组顺序存储：$\mathrm{LOC}(a_{i_1,i_2,\cdots,in})=\mathrm{LOC}(a_{0,0,\cdots,0})+\left\{\sum_{j=1}^{n-1}\left(i_j\times\prod_{k=j+1}^{n}m_k\right)+i_n\right\}\times L$。

数组存储的特点是顺序存储。顺序存储结构是一维的，但数组是多维的，因此，数组的存储要确定行和列的先后顺序。此外，根据数组的定义，在初始化时得到数组的数据元素个数，然后分配连续空间。因此，数组可以实现随机访问，即访问数组中任何数据元素花费的时间都是相同的。

5.3　特殊矩阵的压缩存储

矩阵也称为二维数组，实际上把每行或每列数据看成一个整体，作为一个数据元素，那么矩阵就是由一列或一行数据元素组成的。从这个角度看，矩阵完全符合线性表的特征，只是每个数据元素的类型又是一个线性表。因此，矩阵可以看成线性表的一种推广。

对于大小为 $n\times n$ 的矩阵 $\boldsymbol{A}_{n\times n}$，用二维数组顺序的存储结构存储时，所需的存储空间总是固定的，即占用空间维是 $n\times n$。但是在实际应用中，这些矩阵往往由许多相同的元素或含大量零元素，或者是有规律排列的元素。为了节省存储空间，可以对这类矩阵进行压缩存储。

压缩存储就是为相同值的多个元素占用一个存储空间，而零元素不分配存储空间。这种数据值相同的元素或者零元素的分布有一定规律的矩阵，称为特殊矩阵。假设每个数组的数据元素占 1 个存储单元。

特殊矩阵主要分为三类：对称矩阵、三角矩阵和稀疏矩阵。

5.3.1　对称矩阵的压缩存储

在一个 n 阶方阵 $\boldsymbol{A}$ 中，若元素满足下述性质：$a_{ij}=a_{ji}$，其中，$0\leqslant i,j\leqslant n-1$，则称 $\boldsymbol{A}$ 为对称矩阵，如式(5.1)所示。

$$\boldsymbol{A}_{n\times n}=\begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,n-2} & a_{0,n-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,n-2} & a_{1,n-1} \\ & & a_{i,j} & & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-2,0} & a_{n-2,1} & \cdots & a_{n-2,n-2} & a_{n-2,n-1} \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix} \tag{5.1}$$

对称矩阵中的元素关于主对角线对称，故只要存储对称矩阵中上三角或下三角中的元素，使得对称的数据元素共享一个存储空间。这样，能节约近一半的存储空间。

1. 按行优先顺序存放对称矩阵下三角

对称矩阵的下三角存储如式(5.2)所示。

$$\boldsymbol{A}_{n\times n}=\begin{bmatrix} a_{0,0} & c & \cdots & c & c \\ a_{1,0} & a_{1,1} & \cdots & c & c \\ & & a_{i,j} & & \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-2,0} & a_{n-2,1} & \cdots & a_{n-2,n-2} & c \\ a_{n-1,0} & a_{n-1,1} & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix} \tag{5.2}$$

如何找到按行优先存储上述这个对称矩阵下三角中任一个数组的数据元素 $a_{i,j}$ 的地址呢？如图 5.7 所示。

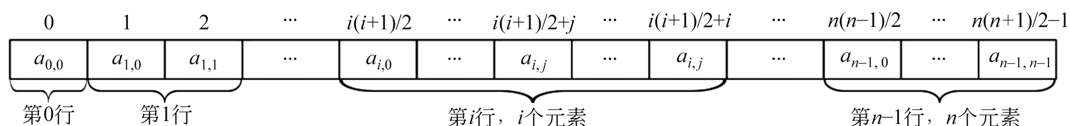


图 5.7　内存中按行优先存储对称矩阵的下三角

(1) 假设 $j \leqslant i$，则 $a_{i,j}$ 在下三角矩阵中，$a_{i,j}$ 之前各行（从 0 行到第 $i-1$ 行）一共有 $1+2+3+\cdots+i=\dfrac{i(i+1)}{2}$ 个数据元素，在第 i 行上，$a_{i,j}$ 之前恰好有 j 个数据元素。因此，$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{0,0})+\dfrac{i\times(i+1)}{2}+j\ (j \leqslant i)$。

(2) 假设 $i<j$，则 $a_{i,j}$ 在上三角矩阵中，因为有 $a_{ij}=a_{ji}$，所以只要交换 i 和 j 的位置即可得到：

$$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{0,0})+\frac{j\times(j+1)}{2}+i \quad (i<j)$$

因此，按行优先压缩存储对称矩阵中任一个数组的数据元素 $a_{i,j}$ 的地址为

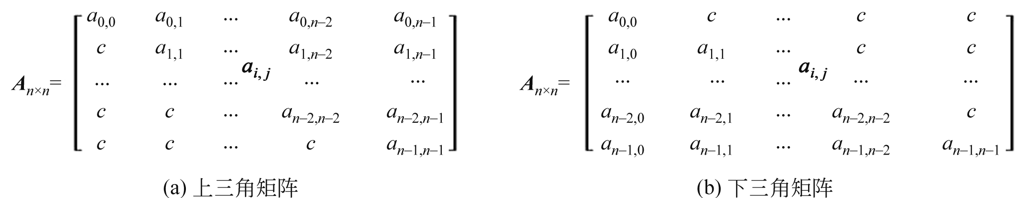
$$\mathrm{LOC}(a_{i,j})=\begin{cases}\mathrm{LOC}(a_{0,0})+\dfrac{i\times(i+1)}{2}+j, & 0\leqslant j\leqslant i<n\\[2ex] \mathrm{LOC}(a_{0,0})+\dfrac{j\times(j+1)}{2}+i, & 0\leqslant i\leqslant j<n\end{cases}$$

2. 按列优先顺序存放对称矩阵上三角

思考与讨论 5-1：根据上面的方法，试着写出如何找到按列优先存储图 5.6 所示的对称矩阵上三角中任一个数组的数据元素 $a_{i,j}$ 的地址呢？

5.3.2　三角矩阵的压缩存储

以主对角线划分，三角矩阵分为上三角矩阵和下三角矩阵两种。上三角矩阵是指矩阵的下三角（不包括主对角线）中的数据元素均为常数 c。下三角矩阵正好相反，它的主对角线上方均为常数 c，如图 5.8 所示。大多数情况下，常数 c 为 0。



(a) 上三角矩阵　　(b) 下三角矩阵

图 5.8　三角矩阵的两种形式

1. 按行优先顺序存放三角矩阵下三角

如何找到如图 5.8(b)所示的按行优先存储三角矩阵下三角中任一个数组的数据元素 $a_{i,j}$ 的地址呢？如图 5.9 所示。

(1) 假设 $j \leqslant i$，则 $a_{i,j}$ 在下三角矩阵中，$a_{i,j}$ 之前各行（从 0 行到第 $i-1$ 行）一共有 $1+2+3+\cdots+i=\dfrac{i\times(i+1)}{2}$ 个数据元素，在第 i 行上，$a_{i,j}$ 之前恰好有 j 个数据元素。因此，

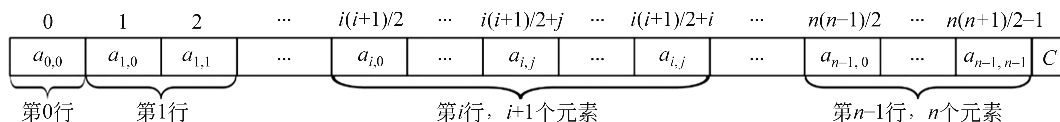


图 5.9　内存中按行优先存储三角矩阵的下三角

$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{0,0})+\dfrac{i\times(i+1)}{2}+j\ (j\leqslant i)$。

（2）假设 $i<j$，$a_{i,j}$ 在上三角矩阵中的值都为常数，此时仅需用数组的第 $\dfrac{n\times(n+1)}{2}$ 个位置存放这个常数值即可。

$$\mathrm{LOC}(a_{i,j})=\frac{n\times(n+1)}{2}\ (i<j)$$

因此，按行优先存储三角矩阵下三角中任一个数组的数据元素 $a_{i,j}$ 的地址为

$$\mathrm{LOC}(a_{i,j})=\begin{cases}\mathrm{LOC}(a_{0,0})+\dfrac{i\times(i+1)}{2}+j, & 0\leqslant j\leqslant i<n\\[2mm] \dfrac{n\times(n+1)}{2}, & 0\leqslant i<j<n\end{cases}$$

2. 按列优先顺序存放三角矩阵上三角

思考与讨论 5-2：根据上面的方法，试着写出如何在图 5.8(a)上找到按列优先存储三角矩阵上三角中任一个数组的数据元素 $a_{i,j}$ 的地址？

5.3.3　带状矩阵的压缩存储

对角矩阵中，所有的非零元素集中在以主对角线为中心的带状区域中，即除了主对角线和主对角线相邻两侧的若干条对角线上的元素之外，其余的数据元素皆为零。带状矩阵中最常见的是如图 5.10 所示的三对角带状矩阵。

$$\boldsymbol{A}_{n\times n}=\begin{bmatrix} a_{0,0} & a_{0,1} & 0 & \cdots & 0 & 0\\ a_{1,0} & a_{1,1} & a_{1,2} & \cdots & 0 & 0\\ 0 & a_{2,1} & a_{2,2} & \cdots & 0 & 0\\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots\\ 0 & 0 & 0 & \cdots & a_{n-2,n-2} & a_{n-2,n-1}\\ 0 & 0 & 0 & \cdots & a_{n-1,n-2} & a_{n-1,n-1} \end{bmatrix}$$

图 5.10　三对角带状矩阵

三对角带状矩阵具有以下特点。

在以下条件下 a_{ij} 非零，其他的数据元素均为零。

$$\begin{cases}\text{当 } i=1 \text{ 时}, j=1,2。\\ \text{当 } 1<i<n \text{ 时}, j=i-1,i,i+1。\\ \text{当 } i=n \text{ 时}, j=n-1,n。\end{cases}$$

1. 按行优先顺序存放带状矩阵

将带状区域上的非零数据元素按行序存储，如何找到图 5.10 所示带状矩阵中任一个非

零数据元素 $a_{i,j}$ 的地址呢？

（1）观察从第 0 行到第 $i-1$ 每行有多少个数据元素。在 $a_{i,j}$ 之前有 i 行，除了第 0 行有两个非零数据元素，其他行均有三个非零数据元素，因此共有 $3i-1$ 个非零数据元素。

（2）第 i 行时，可以分为 $j<i$，$j>i$ 和 $j=i$ 三种情况来考虑，最后整合成 $j-i+1$ 这种表达式。

根据上述分析，得出在 $a_{i,j}$ 之前有 i 行，共有 $3i-1$ 个非零数据元素，在第 i 行，有 $j-i+1$ 个非零数据元素。这样，对于三对角带状矩阵任意一个非零数据元素 $a_{i,j}$ 的地址为

$$\mathrm{LOC}(a_{i,j})=\mathrm{LOC}(a_{0,0})+3i-1+(j-i+1)=\mathrm{LOC}(a_{0,0})+2i+j$$

2. 按行优先顺序存放带状矩阵

思考与讨论 5-3：如何找到图 5.10 所示按列优先存储三对角带状矩阵中任一个非零数据元素 $a_{i,j}$ 的地址呢？

5.4 稀疏矩阵的压缩存储

5.4.1 稀疏矩阵的定义

矩阵 **A** 中非零数据元素的个数 s 远远小于矩阵数据元素的总数，并且非零数据元素的分布没有规律，通常认为矩阵中非零数据元素的总数比矩阵所有数据元素总数的值小于或等于 5%时，则称该矩阵为稀疏矩阵，该比值称为这个矩阵的稠密度，如图 5.11 所示。

$$\boldsymbol{A}=\begin{pmatrix} 0 & 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

图 5.11 稀疏矩阵

根据稀疏矩阵的特点，如果存储所有数据元素，显然存放了很多 0，空间利用率低。可以考虑只存放非零数据元素，但如果仅存放一个数据元素的值是不行的，因为不能确定它在矩阵中的位置，所以必须存储其下标。

5.4.2 稀疏矩阵的存储结构

稀疏矩阵的存储结构主要有两种：顺序存储和链式存储。

1. 顺序存储

顺序存储也称为三元组表，是稀疏矩阵一种常用的存储结构。在这种方法中，稀疏矩阵的非零数据元素可以用一个三元组 i,j,v 表示，其中，i 和 j 分别表示非零数据元素的行索引和列索引，v 表示该非零数据元素的值。一个稀疏矩阵中所有的非零数据元素可表示为这样一些三元组的集合，如图 5.12 所示。