第5章 数组和广义表

5.1 数组的定义
5.2 数组的顺序表示和实现
5.3 矩阵的压缩存储
5.3.1 特殊矩阵
5.3.2 稀疏矩阵
5.4 广义表的定义
5.5 广义表的存储结构

5.1 数组的定义

- 数组是一种特殊的数据结构：
  - 数组是存储结构，是语言内建的数据类型。它可以成为多种数据结构的存储表示。它的操作只有按下标“读/写”。
  - 数组又是逻辑结构，用于问题的解决。可以有查找、定位、插入、删除等操作。
- 一维数组的数组元素为不可再分割的单元素时，是线性结构；但它的数组元素是数组时，是多维数组，是非线性结构。

5.1.1 一维数组

- 定义 数组是相同类型的数据元素的集合，而一维数组的每个数组元素是一个序对，由下标（index）和值（value）组成。
  - 一维数组的示例

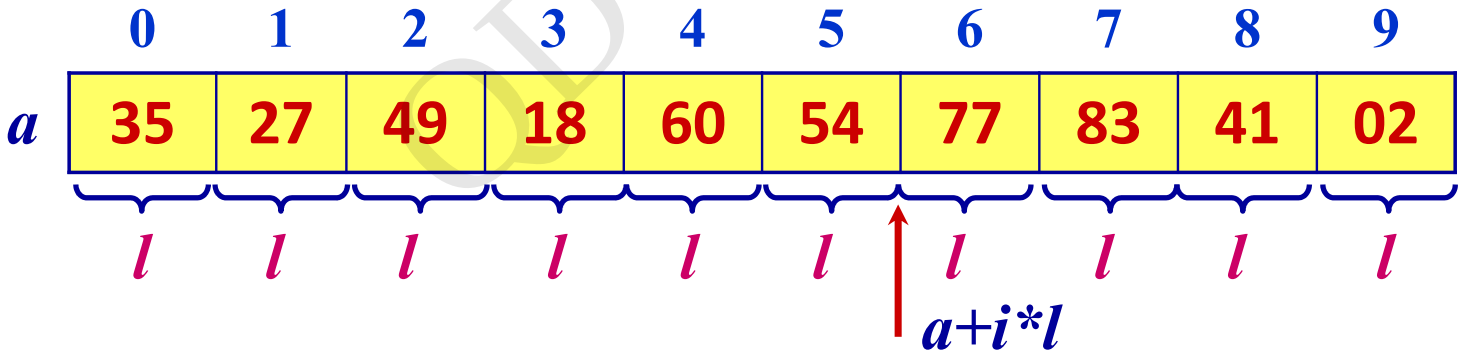
0	1	2	3	4	5	6	7	8	9
35	27	49	18	60	54	77	83	41	02

  - 在高级语言中的一维数组只能按元素的下标直接存取数组元素的值。

一维数组的顺序表示和实现

- 设每个数组元素占据相等的 l 个存储单元，第 0 号元素的存储地址为 a，则第 i 号数组元素的存储地址 LOC(i) 为：

$$\mathrm{LOC}(A[i]) = \begin{cases} a, & i = 0 \\ \mathrm{LOC}(A[i-1]) + l = a + i * l, & i > 0 \end{cases}$$



一维数组的定义和初始化

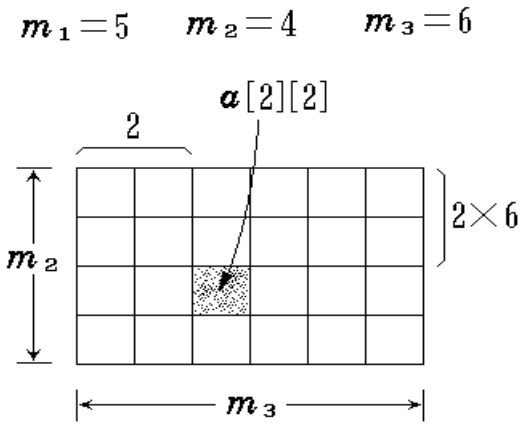
```
void main ( )
{
   int a[3] = { 3, 5, 7 }, *elem, i;          //静态数组
   for ( i = 0; i < 3; i++ )
         printf ( "%d", a[i] );

   elem = (int *) malloc (3*sizeof (int));  //动态数组
   for ( i = 0; i < 3; i++ )  scanf ("%d", &elem[i] );
   for ( i = 0; i < 3; i++ )
       printf ( "%d", *(elem + i ));
}
```

5.1.2 多维数组

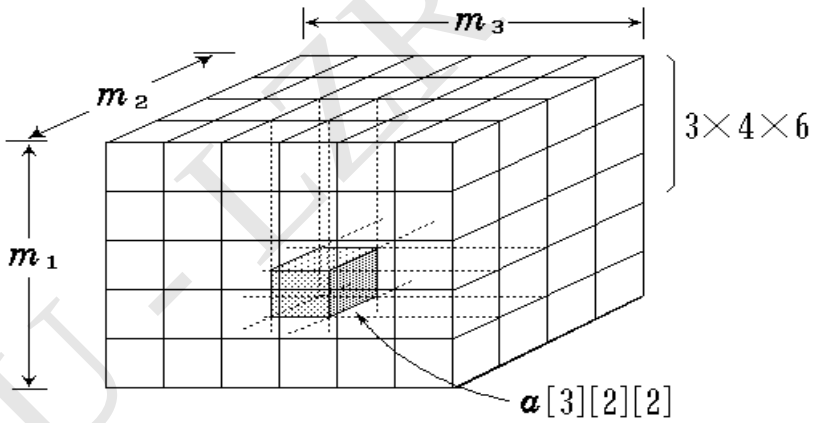
- 多维数组属于数组套数组，可以看做是一维数组的推广。多维数组的特点是每一个数据元素可以有多个直接前驱和多个直接后继。
  - 例如，二维数组可以视为其每一个数组元素为一维数组的一维数组，但从整体来看，每一个数组元素同时处于两个向量（行、列），它可能有两个直接前驱，有两个直接后继。必须有两个下标（行、列）以标识该元素的位置。
- 数组元素的下标一般具有固定的下界和上界，因此它比其他复杂的非线性结构简单。

二维数组



行向量 下标 *i*
列向量 下标 *j*

三维数组



页向量 下标 *i*
行向量 下标 *j*
列向量 下标 *k*

二维数组的顺序表示和实现

- 一维数组常被称为向量（vector）。
- 二维数组 A[m][n] 可看成是由 m 个行向量组成的向量，也可看成是由 n 个列向量组成的向量。
- 一个二维数组类型可以定义为其分量类型为一维数组类型的一维数组类型：

```
typedef ElemType array2[m][n];      //T为元素类型
```

等价于：

```
typedef  ElemType array1[n];          //行向量类型
typedef   array1   array2[m];   //二维数组类型
```

一个m行n列的二维数组A可以看作是每个数据元素都是相同类型的一维数组的一维数组。

$$A=\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & & & \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{bmatrix} \Rightarrow A=[A_1, A_2, \ldots, A_m]'$$

$$A_1=[a_{1,1}, a_{1,2}, \ldots, a_{1,n}]$$
$$A_2=[a_{1,1}, a_{2,2}, \ldots, a_{2,n}]$$
$$\cdots\cdots$$
$$A_m=[a_{m,1}, a_{m,2}, \ldots, a_{m,n}]$$

由此看出，多维数组是线性表的推广。

- 同理，一个三维数组类型可以定义为其数据元素为二维数组类型的一维数组类型。
  - 静态定义的数组，其维数和维界在数组定义时指定，在编译时静态分配存储空间。一旦数组空间用完则不能扩充。
  - 动态定义的数组，其维界不在说明语句中显式定义，而是在程序运行中创建数组时通过动态分配语句 **malloc** 分配存储空间和初始化，在撤销数组时通过 **free** 语句动态释放。
- 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列到一个序列中。

二维数组的动态定义和初始化

- 在程序中静态定义二维数组

```
#define m 30
int A[m][m], B[100][60];
```

- 在程序中用动态存储分配建立的二维数组。

```
int **A;   int m = 10,  n = 6,  i,  j;
*A = (int **) malloc (m*sizeof (int *));
for ( i = 0; i < m; i++ )
    A[i] = (int *) malloc (n*(int));
for ( i = 0; i < m; i++ )
   for ( j = 0; j < n; j++ )  scanf ("%d", &A[i][j] );
```

- 动态回收也需要分两步：

```
for ( i = 0; i < m; i++)
        free (A[i]);
free (A);
```

二维数组中数组元素的顺序存储

- 设有一个 n 行 m 列的二维数组，如图：

$$a=\begin{pmatrix} a[0][0] & a[0][1] & \cdots & a[0][m-1] \\ a[1][0] & a[1][1] & \cdots & a[1][m-1] \\ a[2][0] & a[2][1] & \cdots & a[2][m-1] \\ \vdots & \vdots & \ddots & \vdots \\ a[n-1][0] & a[n-1][1] & \cdots & a[n-1][m-1] \end{pmatrix}$$

- 用一维数组描述它的连续存放方式

$$a=\begin{pmatrix} a[0][0] & a[0][1] & \cdots & a[0][m-1] \\ a[1][0] & a[1][1] & \cdots & a[1][m-1] \\ a[2][0] & a[2][1] & \cdots & a[2][m-1] \\ \vdots & \vdots & \ddots & \vdots \\ a[n-1][0] & a[n-1][1] & \cdots & a[n-1][m-1] \end{pmatrix}$$

- 行优先存放（以行为主序）：

  设数组开始存放位置为 a，每个元素占用 l 个存储单元，则$a[i][j]$的存储位置为：

  $$\text{Loc}(a[i][j]) = a+(i*m+j)*l$$

  其中，m 是每行元素个数，即列数。

$$a=\begin{pmatrix} a[0][0] & a[0][1] & \cdots & a[0][m-1] \\ a[1][0] & a[1][1] & \cdots & a[1][m-1] \\ a[2][0] & a[2][1] & \cdots & a[2][m-1] \\ \vdots & \vdots & \ddots & \vdots \\ a[n-1][0] & a[n-1][1] & \cdots & a[n-1][m-1] \end{pmatrix}$$

- 列优先存放（以列为主序）：

  设数组开始存放位置为a，每个元素占用l个存储单元，则$a[i][j]$的存储位置为：

  $$\mathrm{Loc}(a[i][j]) = a+(j*n+i)*l$$

  其中，n是每列元素个数，即行数。

三维数组的顺序表示

- 各维元素个数为 m_1, m_2, m_3
- 下标为 i_1, i_2, i_3的数组元素的存储地址：

（按页 / 行 / 列存放）

$$\&(\,a[i_1][i_2][i_3]\,) = a + (\,i_1 * m_2 * m_3 + i_2 * m_3 + i_3\,) * l$$

前 i_1 页总元素个数

第 i_1 页前 i_2 行总元素个数

第 i_2 行前 i_3 列元素个数

n 维数组的顺序表示

- 各维元素个数为 $m_1, m_2, m_3, \ldots, m_n$
- 下标为 $i_1, i_2, i_3, \ldots, i_n$ 的数组元素的存储地址：

$$\&(a[i_1][i_2]\ldots[i_n]) = a + (i_1*m_2*m_3*\ldots*m_n + i_2*m_3*m_4*\ldots*m_n + \ldots\ldots + i_{n-1}*m_n + i_n)*l$$

$$= a + \left(\sum_{j=1}^{n-1} i_j * \prod_{k=j+1}^{n} m_k + i_n\right) * l$$

END