



第五章数组和广义表



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

5.4 广义表的定义

5.5 广义表的存储结构



5.1 数组的定义

$D = \{a_{j_1, j_2, \dots, j_n} \mid n > 0\}$ 称为数组的维数, b_i 是数组第 i 维的长度, j_i 是数组元素第 i 维的下标, a_{j_1, j_2, \dots, j_n} 属于 **ElemSet**

- $j_i = 0, \dots, b_i - 1, i = 1, 2, \dots, n$



二维数组: $b_1=m, b_2=n$

$$A_{m \times n} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \\ a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & & a_{m-1,n-1} \end{bmatrix}$$

看成 n 个数据元素的线性表，每个数据元素代表数组中的一列

$$A_{m \times n} = (R_0, R_1, \dots, R_{n-1})$$

$$R_0 = (a_{00}, a_{10}, \dots, a_{m-1,0})$$

$$R_i = (a_{0,i}, a_{1,i}, \dots, a_{m-1,i})$$

看成 m 个数据元素的线性表，每个数据元素代表数组中的一行

$$A_{m \times n} = (S_0, S_1, \dots, S_{m-1})$$

$$S_0 = (a_{00}, a_{01}, \dots, a_{0,n-1})$$

$$S_i = (a_{i,0}, a_{i,1}, \dots, a_{i,n-1})$$



基本操作

- 初始化一个数组
- 取数组元素的值
- 给数组元素赋值

特点:

- 数组一般不做**插入**、**删除**操作-----**顺序存储结构**
- 数组是多维的结构，而存储空间是一个一维的结构。



5.2 数组的顺序表示和实现

二维数组有两种顺序映象的方式：

- 以行序为主序
- 以列序为主序

以“行序为主序”的存储映象

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$



二维数组A中任一元素 $a_{i,j}$ 的存储位置

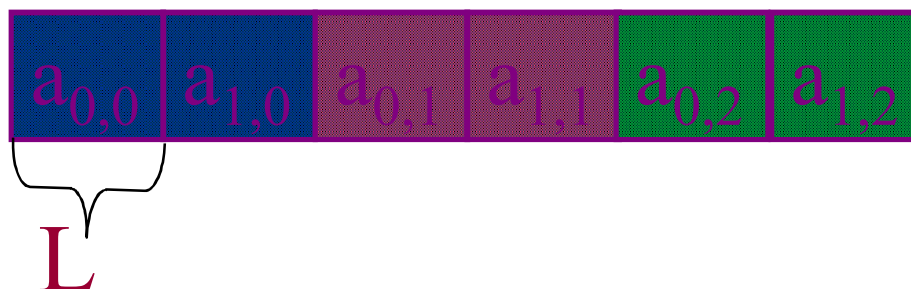
$$\text{LOC}(i,j) = \text{LOC}(0,0) + (b_2 \times i + j) \times L$$

称为基地址或基址

- 从数组的第一行开始依次存放每一行的数组元素;
- 存放第*i*行时, 从第一列开始顺次存放

以“列序为主序”的存储映象

$a_{0,0}$	$a_{0,1}$	$a_{0,2}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$



二维数组A中任一元素 $a_{i,j}$ 的存储位置

$$\text{LOC}(i,j) = \text{LOC}(0,0) + (b_1 \times j + i) \times L$$

↑ 称为基地址或基址

- 从数组的第一列开始依次存放每一列的数组元素；
- 存放第*i*列时，从第一行开始顺次存放



5.3 矩阵的压缩存储

为值相同的元素只分配一个空间，对零元不分配

□研究二类矩阵的压缩存储：

- 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、对称矩阵、对角矩阵

- 稀疏阵：零元多，分布无规律

□设计的压缩存储方式要方便访问操作，最好仍能“随机访问”

1、上三角矩阵 $(a_{ij})_{n \times n}$, $1 \leq i, j \leq n$

特点: $i > j$ 时, $a_{ij} = 0$ 或常量

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ 0 & a_{22} & a_{23} & \cdots & a_{2n} \\ 0 & 0 & a_{33} & \cdots & a_{3n} \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ 0 & 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ c & a_{22} & a_{23} & \cdots & a_{2n} \\ c & c & a_{33} & \cdots & a_{3n} \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ c & c & c & \cdots & a_{nn} \end{bmatrix}$$

1、上三角矩阵—列为主序压缩存储—数组sa[M]

特点： $i > j$ 时， $a_{ij} = 0$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{3n} \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ 0 & 0 & 0 & \dots & a_{nn} \end{bmatrix}$$

$$\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + ((j-1)j/2 + i - 1) * L$$

$i \leq j$

$$\text{Loc}(a_{ij}) = 0 + ((j-1)j/2 + i - 1) * 1$$

$i \leq j$

$k = (j-1)j/2 + i - 1$, $a_{ij} (i \leq j)$ 存于下标为 k 的数组元素中

$$M = n(n+1)/2$$

sa[0] sa[1]

a_{11}	a_{12}	a_{22}	a_{13}	a_{23}	...	a_{1n}	...	a_{nn}
----------	----------	----------	----------	----------	-----	----------	-----	----------

$$M = ?$$

列为主序压缩存储：从第一列开始依次存放每一列的“非0元”

1、上三角矩阵—列为主序压缩存储—数组sa[M]

特点： $i > j$ 时， $a_{ij} = \text{常量} C$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ C & a_{22} & a_{23} & \dots & a_{2n} \\ C & C & a_{33} & \dots & a_{3n} \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ & \cdot & & & \cdot \\ C & C & C & \dots & a_{nn} \end{bmatrix}$$

M=?

a_{11}	a_{12}	a_{22}	a_{13}	a_{23}	\dots	$a_{1,n}$	\dots	$a_{n,n}$	C
----------	----------	----------	----------	----------	---------	-----------	---------	-----------	-----

sa[0] sa[1]

sa[n(n+1)/2-1]

列为主序压缩存储：从第一列开始依次存放每一列的“非C元”

2、下三角矩阵 $(a_{ij})_{n \times n}$, $1 \leq i, j \leq n$

特点: $i < j$ 时, $a_{ij} = 0$ 或常量

$$A_{nn} = \begin{bmatrix} a_{11} & 0 & 0 & \cdots & 0 \\ a_{21} & a_{22} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & & a_{nn} \end{bmatrix}$$

- 压缩存储在 $n(n+1)/2$ 的空间中
- 若用数组 $sa[n(n+1)/2]$ 存储,
- $k = i(i-1)/2 + j - 1$ 当 $i \geq j$

a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	\cdots	a_{n1}	\cdots	a_{nn}
----------	----------	----------	----------	----------	----------	----------	----------	----------

$$A_{nn} = \begin{bmatrix} a_{11} & c & c & \cdots & c \\ a_{21} & a_{22} & c & \cdots & c \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & & a_{nn} \end{bmatrix}$$

行为主序压缩存储: 从第一行开始依次存放每一行的“非0 (C) 元”

3、对称矩阵

特点: $a_{ij} = a_{ji}$

只存上三角阵或只存下三角阵都可以

4、对角矩阵 -- $2d+1$ 对角阵：主对角线和主对角线上面 d 条对角线、主对角线下面 d 条对角线上的数据元素分布不规律，非0 (C)

$$\begin{bmatrix} a_{00} & a_{01} & 0 & 0 & 0 & \dots & 0 \\ a_{10} & a_{11} & a_{12} & 0 & 0 & \dots & 0 \\ 0 & a_{21} & a_{22} & a_{23} & 0 & \dots & 0 \\ 0 & 0 & a_{32} & a_{33} & a_{34} & \dots & 0 \\ 0 & 0 & 0 & a_{43} & a_{44} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & a_{n-1,n-1} \end{bmatrix}$$

3-对角阵

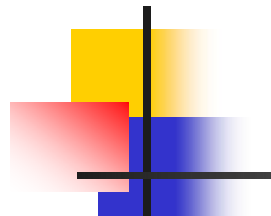
$$\begin{bmatrix} a_{00} & a_{01} & a_{02} & 0 & 0 & \dots & 0 \\ a_{10} & a_{11} & a_{12} & a_{13} & 0 & \dots & 0 \\ a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & \dots & 0 \\ 0 & a_{31} & a_{32} & a_{33} & a_{34} & \dots & 0 \\ 0 & 0 & a_{42} & a_{43} & a_{44} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & 0 & \dots & a_{n-1,n-1} \end{bmatrix}$$

5-对角阵

$2d+1$ 对角阵特点：第一行和最后一行每行有 $d+1$ 个数据元素，余下每行最多 $2d+1$ 个数据元素

压缩存储方法：第一行和最后一行各存 $d+1$ 个数据元素，余下每行存 $2d+1$ 个数据元素

压缩存储方法：第一行和最后一行各存 $d+1$ 个数据元素，余下每行存 $2d+1$ 个数据元素

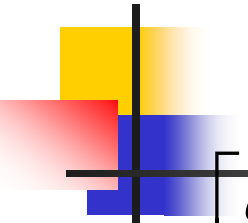


$$\begin{bmatrix}
 a_{00} & a_{01} & 0 & 0 & 0 & \dots & 0 \\
 a_{10} & a_{11} & a_{12} & 0 & 0 & \dots & 0 \\
 0 & a_{21} & a_{22} & a_{23} & 0 & \dots & 0 \\
 0 & 0 & a_{32} & a_{33} & a_{34} & \dots & 0 \\
 0 & 0 & 0 & a_{43} & a_{44} & \dots & 0 \\
 \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 0 & 0 & 0 & 0 & 0 & \dots & a_{n-1,n-1}
 \end{bmatrix}$$

a_{00}	a_{01}	a_{10}	a_{11}	a_{12}	a_{21}	a_{22}	a_{23}								
----------	----------	----------	----------	----------	----------	----------	----------	--	--	--	--	--	--	--	--

3-对角阵行为主序压缩存储

压缩存储方法：第一行和最后一行各存d+1个数据元素，余下
每行存2d+1个数据元素



$$\begin{bmatrix}
 a_{00} & a_{01} & a_{02} & 0 & 0 & \dots & 0 \\
 a_{10} & a_{11} & a_{12} & a_{13} & 0 & \dots & 0 \\
 a_{20} & a_{21} & a_{22} & a_{23} & a_{24} & \dots & 0 \\
 0 & a_{31} & a_{32} & a_{33} & a_{34} & \dots & 0 \\
 0 & 0 & a_{42} & a_{43} & a_{44} & \dots & 0 \\
 \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\
 0 & 0 & 0 & 0 & 0 & \dots & a_{n-1,n-1}
 \end{bmatrix}$$

a_{00}	a_{01}	a_{02}		a_{10}	a_{11}	a_{12}	a_{13}	a_{20}	a_{21}	a_{22}	a_{23}	a_{23}			
----------	----------	----------	--	----------	----------	----------	----------	----------	----------	----------	----------	----------	--	--	--

5-对角阵行为主序压缩存储



2d+1-对角阵行为主序压缩存储地址计算公式

- 矩阵元素下表从0开始的地址计算公式:
 - $\text{Loc}(a_{ij}) = \text{Loc}(a_{00}) + (2d+1)*i - d + j - i + d$
 - $0 \leq i, j \leq n-1, |i-j| \leq d$
- 矩阵元素下表从1开始的地址计算公式:
 - $\text{Loc}(a_{ij}) = \text{Loc}(a_{11}) + (2d+1)*(i-1) - d + j - i + d$
$$= \text{Loc}(a_{11}) + (2d+1)*(i-1) + j - I$$
 - $1 \leq i, j \leq n, |i-j| \leq d$

4. 稀疏矩阵: 非零元多, 在矩阵中随机出现

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。

常规存储方法缺点:

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

稀疏矩阵的压缩存储方法:

一、三元组顺序表

二、行逻辑联接的顺序表

三、十字链表

一、三元组顺序表

- 
- 采用一维数组以行为主序存放每一非零元;
 - 每一非零元只存行号、列号、非零元的值
-

```
#define MAXSIZE 12500

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 struct {
    Triple data[MAXSIZE + 1];
    int    mu, nu, tu;
} TSMatrix; // 稀疏矩阵类型
```

非零元以行为主序顺序存放

TSMatrix M;

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

1	5	-5
1	2	14
2	2	-7
3	1	36
3	4	28

1	2	14
1	5	-5
3	1	36
2	2	-7
3	4	28

i	j	e
1	2	14
1	5	-5
2	2	-7
3	1	36
3	4	28

M.data[0]

M.data[1]

M.data[2]

M.data[3]

M.data[4]

M.data[5]

如何求转置矩阵？

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

\mathbf{M}

i	j	e
1	2	14
1	5	-5
2	2	-7
3	1	36
3	4	28

T

1

2

2

4

5

3

1

2

3

1

36

14

-7

28

-5

i

j

e

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

方法1:

- 根据三元组顺序表的特点，首先扫描一遍三元素，将扫描到的列号为1的非0元行列交换存放于转置后的新阵，生成新阵第一行的非0元；
- 再扫描一遍三元素，将扫描到的列号为2的非0元行列交换存放于转置后的新阵，生成新阵第二行的非0元；

➤

方法1: 将矩阵M转置成矩阵T

```
Status TransposeSMatrix(TSMatrix M, TSMatrix &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++;}
```

```
    }
```

时间复杂度为: $O(M.nu * M.tu)$

```
    return OK; 缺点: 时间效率低
```

```
} // TransposeSMatrix
```


方法2: 减少原阵的扫描次数, 提高时间效率

Num[col]:存放矩阵T中每一行非零元的个数

Cpot[col]:存放矩阵T中每一行非零元的当前存放的位置

$t \leq M.tu$

初始时为每一行第一个非零元存放的位置

所谓“位置”, 即在三元组中存放的数组元素的下标

i	j	e
1	2	14
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

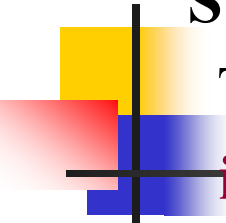
$cpot[1] = 1;$

for (col=2; col≤M.nu; ++col)

$cpot[col] = cpot[col-1] + num[col-1];$

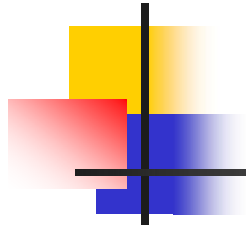
← $t=6$

方法2：减少原阵的扫描次数，提高时间效率



```
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];  
        cpot[1] = 1;  
        for (col=2; col<=M.nu; ++col)  
            cpot[col] = cpot[col-1] + num[col-1];  
        for (p=1; p<=M.tu; ++p) {      .....      }  
    } // if  
    return OK;  
} // FastTransposeSMatrix
```

方法2：减少原阵的扫描次数，提高时间效率



```
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) ... ..
```

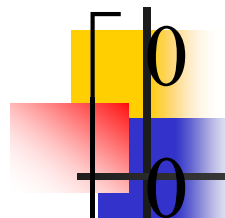
时间复杂度为: $O(M.nu+M.tu)$

二、行逻辑联接的顺序表

三元组顺序表又称**有序的双下标法**，它的特点是，非零元在表中按行序有序存储，因此**便于进行依行顺序处理的矩阵运算**。然而，若需随机存取某一行中的非零元，则需从头开始进行查找。

行逻辑联接的顺序表：随机存取某一行中的非零元，

```
#define MAXRC 500
typedef struct {
    Triple data[MAXSIZE + 1];
    int rpos[MAXRC + 1];
    int mu, nu, tu;
} RLSMatrix; // 行逻辑链接顺序表类型
```



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

<i>i</i>	<i>j</i>	<i>e</i>	
			T.data[0]
1	2	14	T.data[1]
1	5	-5	T.data[2]
2	2	-7	T.data[3]
3	1	36	T.data[4]
3	4	28	T.data[5]

rpos[];

1
3
4

例如：给定一组下标，求矩阵的元素值



```
ElemType value(RLSMatrix M, int r, int c)
```

```
{
```

```
    p = M.rpos[r];
```

```
    while (M.data[p].i==r && M.data[p].j < c)
```

```
        p++;
```

```
    if (M.data[p].i==r && M.data[p].j==c)
```

```
        return M.data[p].e;
```

```
    else return 0;
```

```
} // value
```

三、十字链表

采用链表存放稀疏矩阵的非0元

- 将稀疏矩阵每行的非0元按照列升序的顺序放在一个单链表中
- 将稀疏矩阵每列的非0元按照行升序的顺序放在一个单链表中
- 稀疏矩阵的每个非0元即位于一个行单链表，也同时位于一个列单链表
- 用一维数组保存每行非0元的单链表的头指针
- 用一维数组保存每列非0元的单链表的头指针
- 每个结点非0元的结点结构：
 - row, col, val 分别代表非0元的行号，列号和值
 - down 为指针，指向该非0元同一列的下一个非0元
 - right 为指针，指向该非0元同一行的下一个非0元

row	col	val
down		right

三、十字链表

