教学内容---第四章

DS/SS

1. 绪论
2. 线性表
3. 栈、队列和串
4. 数组
5. 广义表
6. 树和二叉树
7. 图
8. 动态存储管理
9. 查找
10. 内部排序
11. 外部排序
12. 文件

4.1数组的逻辑结构

数组的抽象数据类型定义

ADT Array {

数据对象：$j_i = 0,\ldots,b_i - 1,\ i = 1,2,\ldots,n$

$D=\{a_{j1\,j2\cdots\,jn} \mid a_{j1\ j2\cdots\ jn}$属于ElemSet, n(>0)为数组维数，

j_i为数组元素的第i维下标， b_i为第i维的长度 }

数据关系：$R = \{R1,R2,\ldots,Rn\}$

$Ri=\{< a_{j1\cdots ji\cdots\ jn} , a_{j1\cdots ji+1\cdots\ jn} >|$

$0<= j_k <= b_k -1,\quad 1<=k<=n,$ 且 $k != i,$

$0<= j_i <= b_i -2,$

$a_{j1\cdots ji\cdots\ jn} , a_{j1\cdots ji+1\cdots\ jn}$属于D, i =2,3, …, n}

基本操作：

InitArray(&A,n,bound1,…,boundn)

DestroyArray(&A)

Value(A,&e,index1,…, indexn) ---------- 查找

Assign(&A,e,index1 ,…, indexn)

} ADT Array

4.2数组的存储结构

DS/SS

数组的顺序存储

内涵：

数组的顺序存储指用一组地址连续的存储单元按照某种规则存放数组中的数据元素。

特点：

* 存储单元地址连续（需要一段连续空间）
* 存储规则（以行（列）为主序）决定元素实际存储位置
* 随机存取
* 存储密度最大（100%）

4.2数组的存储结构（续）

随机存取

BASIC、PL/1、COBOL、PASCAL、C语言都采用以行序为主序；而FORTRAN语言采用以列序为主序。（以后不做说明，均采用以行序为主序。）

对于二维数组A[b1,b2]，设每个元素占L个存储单元，则有：

A中任一元素$a_{i,j}$的存储位置：

$$LOC(i,j)=LOC(0,0)+(b2*i+j)*L$$

对于n维数组A[b1,b2,…,bn]，设每个元素占L个存储单元，则有：

A中任一元素$a_{j1\ j2\ \cdots\ jn}$的存储位置：

$$LOC(j_1, j_2, \ldots, j_n)=LOC(0,0,\ldots 0)+(b2*\ldots*bn*j_1+b3*\ldots*bn*j_2+\ldots+bn*j_{n-1}+j_n)*L$$

上式称为n维数组的映象函数。位置是下标的线性函数，所以随机存取。

4.2数组的存储结构（续）

数组顺序存储的数据类型描述

```
//---------数组的顺序存储表示----------
typedef  struct {
         ElemType        *base          //存储空间基址
         int        dim                 //数组维数
         int        *bounds             //数组维界基址
         int        *constants          //数组映象函数常量基址
} Array
```

4.3矩阵的压缩结构

DS/SS

基本概念和术语

- **压缩存储**：为多个值相同的元只分配一个存储空间；对零元不分配空间。
- **特殊矩阵**：值相同的元或零元在矩阵中的分布有一定的规律，则这类矩阵为特殊矩阵。
- **稀疏因子**：假设在m*n矩阵中非零元个数为t，则称t/(m*n)为稀疏因子。
- **稀疏矩阵**：值相同的元或零元在矩阵中的分布没有一定的规律且稀疏因子小于等于0.05时，称这类矩阵为稀疏矩阵。

特殊矩阵和稀疏矩阵都需要压缩存储

4.3矩阵的压缩结构（续）

DS/SS

特殊矩阵

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ & \cdots\cdots & & & \\ & & a_{i,i} & \cdots & a_{in} \\ & \cdots & & & \\ a_{n1} & \cdots & & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

n阶**对称矩阵**A[n,n]:

$a_{ij} = a_{ji}$ 1<=i,j<=n，用sa[n(n+1)/2]压缩存储。设sa[k]中存放着元素a_{ij}，则有：k=i(i-1)/2+j-1 （当i>=j）；

k=j(j-1)/2+i-1 （当i<j）。

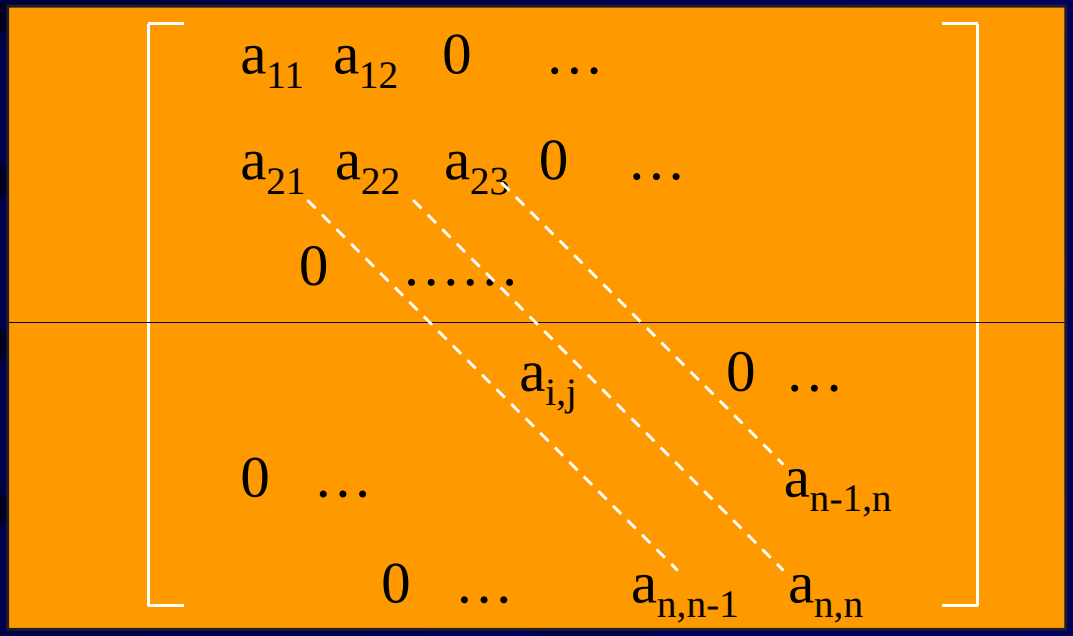
n阶**三角矩阵**A[n,n]:

*下三角矩阵：$a_{ij} = c$, i<j;

*上三角矩阵：$a_{ij} = c$, i>j;

用sa[n(n+1)/2+1]压缩存储。对下三角矩阵，设sa[k]中存放着元素a_{ij},则有：k=i(i-1)/2+j-1 (i>=j);k= n(n+1)/2+1(i<j）。

4.3矩阵的压缩结构（续）



n阶**对角矩阵**A[n,n]，如三对角矩阵：除主对角线和两条副对角线之外，其他元素都为零。用线性空间sa[0..3n-1]压缩存储。设sa[k]中存放着元素a_{ij}，则有：

k=3(i-1)-1+j-(i-1)=2i+j-3 (|i-j|<=1)

i = (k+1)/3 + 1 (0<=k<=3n-1); j = k-2i+3= k- 2((k+1)/3)+1

4.3矩阵的压缩结构（续）

稀疏矩阵的逻辑结构抽象---三元组表

矩阵中的元素一般可由一个三元组（i,j, a_{ij} ）唯一确定。

稀疏矩阵可由表示非零元的三元组及其行列数唯一确定。

例：矩阵M对应的数据结构为如下线性表（以行序为主序排列）：

((1,2,12),(1,3,9),(3,1,-3),(3,6,14),(4,3,24),(5,2,18),(6,1,15),(6,4,-7))

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

4.3矩阵的压缩结构（续）

三元组顺序表

```
//----------稀疏矩阵的三元组顺序表存储表示-----------
#define  MAXSIZE          12500    //非零元最大个数
typedef  struct {
             int          i,j;                    //非零元的行、列下标
             ElemType          e;
}Triple;
typedef  struct{
             Triple     data[MAXSIZE+1 ]          //data[0]未用,行序为主序
             int          mu,nu,tu;               //矩阵的行、列和非零元数
}TSMatrix;
```

4.3矩阵的压缩结构（续）

三元组顺序表上矩

三元组表描述的稀疏矩阵转置基本步骤：

*将矩阵的行列值相互交换；

*将每个三元组中的i和j相互调换；

*重排三元组之间的次序（保证转置后的矩阵的三元组顺序表仍然以行序为主序存储）

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

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

4.3矩阵的压缩结构（续）

三元组顺序表上矩阵转置运算

M.data

0			
1	1	2	12
2	1	3	9
3	3	1	-3
4	3	6	14
5	4	3	24
6	5	2	18
7	6	1	15
8	6	4	-7

T.data

0			
1	1	3	-3
2	1	6	15
3	2	1	12
4	2	5	18
5	3	1	9
6	3	4	24
7	4	6	-7
8	6	3	14

4.3矩阵的压缩结构（续）

三元组顺序表上矩阵转置运算

“重排三元组之间的次序”有两种处理手法，对应着两种处理算法。

第一种：

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。

按照矩阵a的列序来进行转置，为了找到a的每一列中所有非零元素，须对三元组表a.data从第一行起整个扫描一遍，由于a.data是以a的行序为主序来存放每个非零元，由此得到的恰是b.data应有的顺序。

4.3矩阵的压缩结构（续）

DS/SS

M.mu=6	M.nu=7	M.tu=8

M.data

0			
1	1	2	12
2	1	3	9
3	3	1	-3
4	3	6	14
5	4	3	24
6	5	2	18
7	6	1	15
8	6	4	-7

T.mu=7	T.nu=6	T.tu=8

T.data

0			
1	1	3	-3
2	1	6	15
3	2	1	12
4	2	5	18
5	3	1	9
6	3	4	24
7	4	6	-7
8	6	3	14

4.3矩阵的压缩结构（续）

```
Status TransposeSMatrix(TSMatrix M,TSMatrix &T) {
//采用三元组顺序表存储，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; }
    }
    return OK;  } // 算法时间复杂度：O(nu*tu)，适合tu<<mu*nu的情况
```

对M中每一列，扫描一遍所有非零元

4.3矩阵的压缩结构（续）

第二种：

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。

预先确定矩阵a中每一列的第一个非零元在b.data中应有的位置，然后在对a.data中的三元组依次做转置时，便可直接放到b.data中恰当的位置上。

转置前，先求得a的每一列中非零元的个数，进而求得每一列第一个非零元在b.data中应有的位置。

具体做法：num[col]放置矩阵a第col列中非零元的个数；cpot[col]指示a中第col列的第一个非零元在b.data中的恰当位置。有：

cpot[1] = 1;

cpot[col] = cpot[col-1] + num[col-1] 2<=col<=a.nu

	1	2	3	4	5	6	7
num	2	2	2	1	0	1	0
cpot	3	5	7	8	8	9	9

M.mu=6 M.nu=7 M.tu=8 T.mu=7 T.nu=6 T.tu=8

M.data

0			
1	1	2	12
2	1	3	9
3	3	1	-3
4	3	6	14
5	4	3	24
6	5	2	18
7	6	1	15
8	6	4	-7

T.data

0			
1	1	3	-3
2	1	6	15
3	2	1	12
4	2	5	18
5	3	1	9
6	3	4	24
7	4	6	-7
8	6	3	14

4.3矩阵的压缩结构（续）

```
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]=copt[col-1]+num[col-1];
      for (p =1;p<=M.tu; ++p) {
         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]; }
  } return OK;  } // 算法时间复杂度：O(nu+tu)
```

统计M中每一列的非零元个数

初始化M中每一列第一个非零元应该在T中的位置

M中col列上下一个非零元应该在T中的位置

4.3矩阵的压缩结构（续）

三元组十字链表

```
//----------稀疏矩阵的三元组十字链表存储表示-----------
typedef  struct  OLNode{
        int         i,j;                    //非零元的行、列下标
        ElemType            e;              //数组映象函数常量基址
        struct      OLNode    *right, *down  //行表与列表的后继指针
} OLNode            *OLink;
typedef  struct {
        OLink     *rhead, *chead     //行列链表头指针向量基址
        int         mu,nu,tu;               //矩阵的行、列和非零元数
}CrossList;
```

4.3矩阵的压缩结构（续）

三元组十字链表

$$M=\begin{bmatrix} 3 & 0 & 0 & 5 \\ 0 & -1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$$