第5章 数组和广义表

5.1 数组的类型定义

5.2 数组的顺序表示和实现

5.3 矩阵的压缩存储

5.4 广义表的类型定义

5.5 广义表的存储结构

本章重点难点

重点：数组的存储表示方法；特殊矩阵压缩存储方法；稀疏矩阵的压缩存储方法。

难点：稀疏矩阵的压缩存储表示和实现算法。

第5章 数组和广义表

5.1 数组的类型定义

5.2 数组的顺序表示和实现

5.3 矩阵的压缩存储

5.4 广义表的类型定义

5.5 广义表的存储结构

5.1 数组的定义

□ 数组的定义

数组是由下标 (index) 和值(value)组成的有序对(index，value)的集合。

也可以定义为是由相同类型的数据元素组成有限序列。

□ 数组的特点

数组在内存中是采用一组连续的地质空间存储的，正是由于此种原因，才可以实现下标运算。

所有数组都是一个一维向量。

例

分析二维数组a[m][n]和三维数组a[m][n][p]在内存中的存放方式。

a[m][n]在内存中的存放方式是：

$(a_{00}, \cdots;a_{0n-1},a_{10}, \cdots;a_{1n-1}, \cdots a_{ij}, \cdots;a_{m-1\,0}, \cdots;a_{m-1n-1})$

$0 \le i \le m-1,\ 0 \le j \le n-1$;

a[m][n][p]在内存中的存放方式是：

$(a_{000}, \cdots;a_{00n-1},a_{010}, \cdots;a_{01n-1}, \cdots;a_{ijk}, \cdots;a_{m-1n-10}, \cdots;a_{m-1n-1p-1})$

$0 \le i \le m-1,\ 0 \le j \le n-1,\ 0 \le k \le p-1$

5.1 数组的定义

□ 数组的ADT定义

ADT Array {

数据对象： $j_i=0,\ldots,b_i-1,i=1,2,\ldots,n,$
$D=\{a_{j1,j2,\ldots,jn} \mid$ n称为数据元素的维数，b_i是数组第i维的长度，j_i 是数组元素的第i维下标， $a_{j1,j2,\ldots,jn} \in ElemSet\}$

数据关系： $R=\{R1, R2, ..., Rn\}$
$Ri=\{<a_{j1,\ldots ji,\ldots jn}, a_{j1,\ldots ji+1,\ldots jn}> \mid 0 \le j_k \le b_k -1,$
$1 \le k \le n$ 且$k \ne i$, $0 \le j_i \le b_i -2,$
$a_{j1,\ldots ji,\ldots jn}, a_{j1,\ldots ji+1,\ldots jn} \in D,\ i=2,\ldots,n \}$

基本操作：

} ADT Array

见下页

基本操作：

InitArray(&A, n, bound1, ..., boundn) //数组初始化

DestroyArray(&A) //销毁数组

Value(A, &e, index1, ..., indexn) //求数组的值

Assign(&A, e, index1, ..., indexn) //给数据元素赋值

第5章 数组和广义表

5.1 数组的类型定义

5.2 数组的顺序表示和实现

5.3 矩阵的压缩存储

5.4 广义表的类型定义

5.5 广义表的存储结构

5.2 数组的顺序表示和实现

□ 数组元素的地址关系

数组在内存中主要采用两种存储方式：

(1)以行序为主的存储方式；

(2)以列序为主的存储方式。

不同的存储方式有不同元素地址计算方法。

例

分别以行序为主和以列序为主求二维数组a[3][4]中元素a[1][2]地址，首地址用loc(a[0][0])表示，每个元素占用L个内存单位。

a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23

a00	a01	a02	a03
a10	a11	a12	a13
a20	a21	a22	a23

以行序为主：

loc(a[1][2])=loc(a[0][0])+[(1×4)+2]×L

以列序为主：

loc(a[1][2])=loc(a[0][0])+[(2×3)+1]×L

数组元素的地址关系 (行序为主)

设每个元素所占空间为L,A[0][0]的起始地址记为LOC[0,0]。

二维数组$A[b_1][b_2]$中元素A_{ij}的起始地址为：

$$LOC[i,j]=LOC[0,0]+(b_2\times i+j)L$$

三维数组$A[b_1][b_2][b_3]$中数据元素A[i][j][k]的起始地址为：

$$LOC[i,j,k]=LOC[0,0,0]+(b_2\times b_3\times i+b_3\times j+k)\times L$$

数组元素的地址关系 (行序为主)

设每个元素所占空间为L,A[0][0]的起始地址记为LOC[0,0]。

n维数组$A[b_1][b_2]...[b_n]$中数据元素$A[j_1,j_2,\ldots j_n]$的存储位置为：

$$LOC[j_1,j_2,\ldots j_n] = LOC[0,0,\ldots,0] + (b_2\times\ldots\times b_n\times j_1 + b_3\times\ldots\times b_n\times j_2 + \ldots + b_n\times j_{n-1} + j_n)\times L$$

5.2 数组的顺序表示和实现

数组的顺序存储类型实现

```
#include <stdarg.h>
                    //标准头文件，提供宏va_start、
                    //va_arg和va_end,用于存放变长参数表
#define MAX_ARRAY_DIM 8     //数组维数
typedef struct {
    Elemtype *base;                 //基址
    int dim;                        //维数
    int *bound;                     //数组维界基址
    int *constants;                 //数组映像函数常量基址
}Array;
```

第5章 数组和广义表

5.1 数组的类型定义

5.2 数组的顺序表示和实现

5.3 矩阵的压缩存储

5.4 广义表的类型定义

5.5 广义表的存储结构

5.3 矩阵的压缩存储

5.3.1 特殊矩阵

5.3.2 稀疏矩阵

5.3.3 矩阵的压缩存储

5.3.1 特殊矩阵

- 下(上)三角矩阵

对角线以上(下)元素都为0的矩阵称为(下)上三角矩阵与对称矩阵A

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 2 & 0 & 0 \\ 4 & 5 & 3 & 0 \\ 3 & 6 & 7 & 8 \end{pmatrix}$$

下三角矩阵

$$\begin{pmatrix} 1 & 5 & 6 & 3 \\ 0 & 2 & 1 & 3 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 8 \end{pmatrix}$$

上三角矩阵

5.3.1 特殊矩阵

- 下(上)三角矩阵的存储方式

下(上)三角矩阵对角线以上(下)元素都为零，根据这个特点可以定义一个长度为n*(n+1)/2的一维数组来存储。

- 下(上)三角矩阵压缩存储时地址对应关系

设下三角矩阵为a[n][n]，用一维数组sa[n*(n+1)/2]存储，则矩阵元素a[i][j]在数组sa中的位置为：

当i>=j时

a[i][j]对应存储在sa[i(i-1)/2+j-1]

5.3.2 稀疏矩阵

- 何谓稀疏矩阵

假设 m 行 n 列的矩阵含 t 个非零元素，

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。

5.3.2 稀疏矩阵

□ 稀疏矩阵的ADT定义

ADT SparseMatrix {

数据对象： $D=\{a_{ij} \mid i=1,2,\ldots,m;j=1,2,\ldots,n;$
$a_{ij} \in ElemSet,m,n$分别为行数与列数$\}$

数据关系： $R=\{Row, Col\}$

$Row=\{<a_{i,j}, a_{i,j+1}> \mid i=1,\ldots,m,j=1,\ldots,n-1\}$

$Col=\{<a_{i,j}, a_{i+1,j}> \mid i=1,\ldots,m-1,j=1,\ldots,n\}$

基本操作

见下页

} ADT Array

基本操作：

CreatSMatrix(&M)　　//创建稀疏矩阵M

DestroyArray(&M)　　//销毁稀疏矩阵M

……

TransposeSMatrix(M, &T)　//求稀疏矩阵M的转置矩阵T

MultSMatrix(M,N,&Q)　　//求稀疏矩阵M和N的乘积Q

稀疏矩阵的三元组顺序表存储

根据稀疏矩阵大部分元素的值都为零的特点，可以只存储稀疏矩阵的非零元素，三元组分别记录非零元素的行，列位置和元素值。

例 求矩阵M的三元组表示。

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{pmatrix}$$

矩阵M

M=((1，2，1)，(2，4，2)，(3，1，1))

5.3.3 矩阵的压缩存储

三元组顺序表的实现

```
#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];    //data[0]未用
     int     mu, nu, tu;
} TSMatrix;                       // 稀疏矩阵类型
```

5.3.3 矩阵的压缩存储

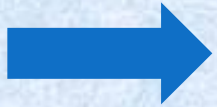
三元组顺序表转值的实现

示例分例

(1)从矩阵到转置矩阵

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 5 \\ 0 & 7 & 0 & 0 & 0 \\ 3 & 0 & 0 & 2 & 0 \end{pmatrix}$$

矩阵M



$$\begin{pmatrix} 0 & 0 & 3 \\ 1 & 7 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 2 \\ 5 & 0 & 0 \end{pmatrix}$$

转值矩阵T

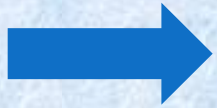
5.3.3 矩阵的压缩存储

三元组顺序表转值的实现

- 示例分例

(2)三元组的转值

1	2	1
1	5	5
2	2	7
3	1	3
3	4	2



1	3	3
2	1	1
2	2	7
4	3	2
5	1	5

三元组顺序表转值算法详解

```
Status TransposeSMatrix(TSMatrix M, TSMatrix &T)
{ int p, q, col;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;
}
```

- 三元组顺序表转值算法时间复杂性

```
for (col=1; col<=M.nu; ++col)
    for (p=1; p<=M.tu; ++p)
      ..............................
      }
```

时间复杂度为: O(M.nu*M.tu)

第5章 数组和广义表

广义表的引入

线性表要求数据元素的类型相同，在实际应用中线性表的数据类型往往不同。

例如：一个公司有董事长，总经理，秘书，人事部，分公司等等，董事长、总经理、秘书都是单个的人，而人事部、分公司又是一个组织。

如何在这种情况下应用线性表，就是广义表的范畴。

广义表的定义

广义表是线性表的推广，也称列表(Lists)。它是 n个元素的有限序列，记作A=（a_1,a_2,……a_n)

其中A是表名，n是广义表的长度，ai是广义表的元素，ai既可以是单个元素，也可以是广义表。

原子： 如果ai是单个元素，称为原子，一般用小写字母表示;

子表： 如果ai是广义表，称为子表，一般用大写字母表示。

5.4 广义表的类型定义

广义表的术语

表头(Head)：非空广义表的第一个元素a1;
表尾(Tail)：除了表头的其余元素组成的表;
深度：广义表中括号嵌套的最大层数。

特点

广义表的元素可以是子表，子表的元素还可以是子表，存储空间难以确定，常采用链式存储。

广义表的ADT

ADT Glist {

数据对象： $D=\{e_i \mid i=1,2,..,n;\ n\geq 0;$
$e_i \in AtomSet$ 或 $e_i \in GList,$
AtomSet为某个数据对象 **}**

数据关系： $LR=\{<e_{i-1}, e_i> \mid e_{i-1}, e_i \in D, 2\leq i\leq n\}$

基本操作： 见下页

} ADT Glist

基本操作

InitGlist(&L)　　//初始化
CreateGlist(&L,S)　　//销毁
GListLength(L)　　//求表长
GListDepth(L)　　//求表的深度
GetHead(L)　　//取表头
GetTail(L)　　//取表尾

基本操作举例

按例(1)的方式完成(2)(3)(4)填空

（1）B=（e）

只含一个原子，长度为1，深度为1。

（2）C=(a,(b，c，d))

有一个原子，一个子表，长度为2，深度为2。

（3）D=(B,C)

二个元素都是列表，长度为2，深度为3。

（4）E=(a,E)

是一个递归表，长度为2，深度无限，相当于E=（a,（a,（a,（a,……）））））。

基本操作举例

例

设广义表A=(a,b,c),B=(A,(c,d)),C=(a,(B,A),(e,f)),请写出下列各运算结果:

1.CAL(A)= a //计算表头

2.CDR(B)= ((c,d)) //计算表尾部

3.CAL(CAL(CAL(CDR(C))))= A

第5章 数组和广义表

5.5 广义表的存储结构

广义表的头尾指针结点结构

广义表通常采用头、尾指针的链表结构

表结点：

tag=1	hp	tp

原子结点：

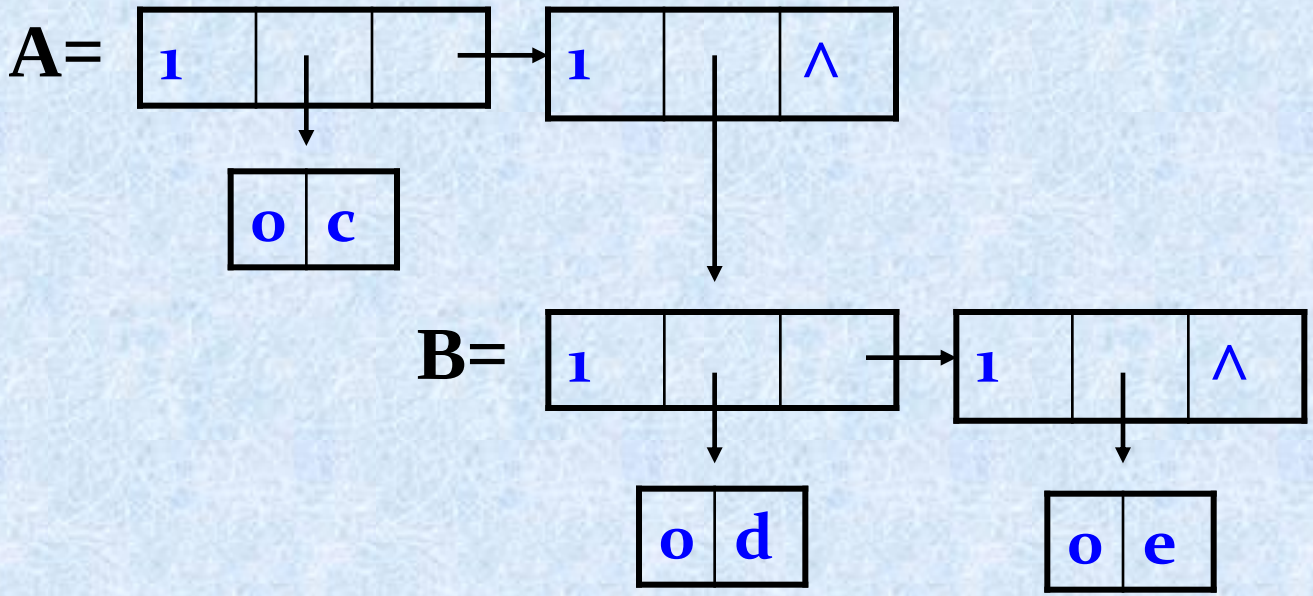
tag=0	atom

对于每一个结点，若tag=0表示这是一个原子结点，atom域存放该原子的值。若tag=1表示这是一个表结点，hp指向子表头，tp指向广义表的下一个

5.5 广义表的存储结构

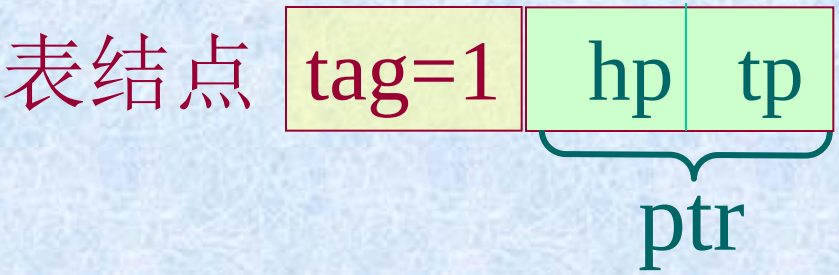
广义表的头尾指针结点结构

例 画出广义表A=(c,B),B=(d,e)的存储结构图



广义表的头尾链表存储表示

```
typedef enum{ATOM,LIST} ElemTag;
// ATOM==0:原子, LIST==1:子表
typedef struct GLNode{
    ElemTag tag;
    union{
    AtomType atom; // 原子结点的数据域
    struct {struct GLNode *hp,*tp;}ptr;
    };
}*Glist;
```



5.5 广义表的存储结构

扩展头尾指针结点结构

表结点：

tag=1	hp	tp

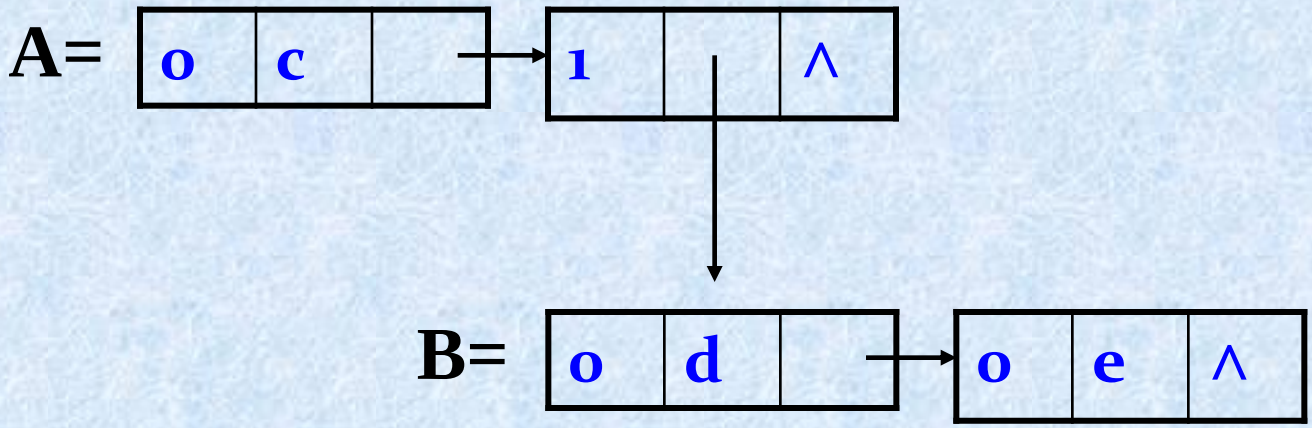
原子结点：

tag=0	atom	tp

对于每一个结点，若tag=0表示这是一个原子结点，atom域存放该原子的值。若tag=1表示这是一个表结点，hp指向子表头，tp指向广义表的下一个

扩展头尾指针结点结构

例 画出广义表A=(c,B),B=(d,e)的存储结构图



扩展头尾指针结点结构

```
typedef enum{ATOM,LIST} ElemTag;

typedef struct GLNode{
    ElemTag tag;
    union{
    AtomType atom;
    struct GLNode *hp;
    };
    struct GLNode *tp;
}
```

填空题 1分

设置

广义表((a,b),(()),(a,(b)))有 [填空1] 个元素

正常使用填空题需3.0以上版本雨课堂

作答

本章小结

熟练掌握：

(1)数组的存储表示方法；

(2)数组在存储结构中的地址计算方法；

(3)特殊矩阵压缩存储时的下标变换公式；

(4)稀疏矩阵的压缩存储方法；

(5)三元组表示稀疏矩阵时进行矩阵运算采用的算法。

(6)广义表的定义和性质。