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
  - 5.3.1 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

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储

$$A=\begin{pmatrix} 15 & 0 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

如何只存储非零元素？

注意：稀疏矩阵中的非零元素的分布没有规律。

一元多项式的表示

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储

将稀疏矩阵中的每个非零元素表示为：

（行号，列号，非零元素值）——三元组(3-tuple, triple)

定义三元组：

row（行）	col（列）	item（值）

```
typedef struct {
    int row, col;            //行号，列号
    ElemType item            //非零元素值
}Triple;
```

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储

三元组表：将稀疏矩阵的非零元素对应的三元组所构成的集合，按行优先的顺序排列成一个线性表。

$$A=\begin{pmatrix} 15 & 0 & 0 & 0 & 0 & 0 \\ 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 9 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

三元组表＝((0,0,15), (1,1,11), (2,3,6), (4,0,9))

如何存储三元组表？

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储——三元组顺序表

采用顺序存储结构存储三元组表

$$A=\begin{pmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

三元组顺序表是否需要预留存储空间？

稀疏矩阵的修改操作

三元组顺序表的插入/删除操作

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储——三元组顺序表

采用顺序存储结构存储

三元组表

$$A=\begin{pmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储——三元组顺序表

存储结构定义：

```
#define  Max 100;
typedef struct
{
   Triple data[Max];   //存储非零元素
   int mu, nu, tu;          //行数，列数，非零元个数
}TSMatrix;

typedef struct {
    int row, col;                          //行号，列号
    ElemType item              //非零元素值
}Triple;
```

其中, data域中的三元组是以行序为主序进行排列的.

5.3 矩阵的压缩存储——稀疏矩阵

三元组顺序表操作——转置操作

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
Max	空闲	空闲	空闲

5（矩阵的行数）
6（矩阵的列数）
7（非零元个数）

$$A=\begin{pmatrix} 15 & 0 & 0 & 22 & 0 & -15 \\ 0 & 11 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 91 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$B=\begin{pmatrix} 15 & 0 & 0 & 0 & 91 \\ 0 & 11 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 \\ 22 & 0 & 6 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ -15 & 0 & 0 & 0 & 0 \end{pmatrix}$$

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3	2	1	3
4	3	0	22
5	3	2	6
6	5	0	-15
Max	空闲	空闲	空闲

6（矩阵行数）
5（矩阵列数）
7（非零元个数）

5.3 矩阵的压缩存储——稀疏矩阵

三元组顺序表转置算法——算法 I

基本思想：直接取，顺序存。即在*A*的三元组顺序表中依次找第0列、第1列、…直到最后一列的三元组，并将找到的每个三元组的行、列交换后顺序存储到*B*的三元组顺序表中。

5.3 矩阵的压缩存储——稀疏矩阵

设置矩阵B的行数、列数、非零元个数

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	-15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max-1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0			
1			
2			
3			
4			
5			
6			
Max-1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

在矩阵A中查找第0列非零元，顺序存储到矩阵B中

	row	col	item
→ 0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
→ 6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0	0	0	15
1	0	4	91
2			
3			
4			
5			
6			
Max−1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

在矩阵A中查找第1列非零元，顺序存储到矩阵B中

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
→ 3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3			
4			
5			
6			
Max−1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

在矩阵A中查找第2列非零元，顺序存储到矩阵B中

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
→ 4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3	2	1	3
4			
5			
6			
Max−1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

在矩阵A中查找第3列非零元，顺序存储到矩阵B中

	row	col	item
0	0	0	15
→ 1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
→ 5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3	2	1	3
4	3	0	22
5	3	2	6
6			
Max−1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

在矩阵A中查找第4列非零元，顺序存储到矩阵B中

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3	2	1	3
4	3	0	22
5	3	2	6
6			
Max−1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

在矩阵A中查找第5列非零元，顺序存储到矩阵B中

	row	col	item
0	0	0	15
1	0	3	22
→ 2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3	2	1	3
4	3	0	22
5	3	2	6
6	5	0	-15
Max−1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

在矩阵A中查找第6列非零元，顺序存储到矩阵B中

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3	2	1	3
4	3	0	22
5	3	2	6
6	5	0	-15
Max−1			
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

5.3 矩阵的压缩存储——稀疏矩阵

三元组顺序表转置算法 I ——伪代码

1. 设置转置后矩阵B的行数、列数和非零元个数；
2. 在B中设置初始存储位置pb；
3. for (col=最小列号; col<=最大列号; col++)
   3.1 在A中查找列号为col的三元组；
   3.2 交换其行号和列号，存入B中pb位置；
   3.3 pb++；

5.3 矩阵的压缩存储——稀疏矩阵

```
Status TransposeMatrix( TSMatrix  A,  TSMatrix  &B )
{   B.mu=A.nu;    B.nu=A.mu;     B.tu=A.tu;
    if( B.tu ) {
        pb=0;
        for( col=0; col<XXXX; col++ )
          for( pa=0; pa<XXXX; pa++ )
            if( A.data[pa].col == col )
            {   B.data[pb].col    = A.data[pa].row;
                B.data[pb].row  = A.data[pa].col;
                B.data[pb].item = A.data[pa].item;
                pb++;
            }
    }
    return OK;
}
```

5.3 矩阵的压缩存储——稀疏矩阵

```
Status TransposeMatrix( TSMatrix  A,  TSMatrix  &B )
{   B.mu=A.nu;    B.nu=A.mu;     B.tu=A.tu;
    if( B.tu ) {
        pb=0;
        for( col=0; col<A.nu; col++ )
          for( pa=0; pa<A.tu; pa++ )
            if( A.data[pa].col == col )
            {   B.data[pb].col    = A.data[pa].row;
                B.data[pb].row  = A.data[pa].col;
                B.data[pb].item = A.data[pa].item;
                pb++;
            }
    }
    return OK;
}
```

5.3 矩阵的压缩存储——稀疏矩阵

三元组顺序表转置算法——算法II

A

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

maxsize−1

B

	row	col	item
0	0	0	15
1	0	4	91
2	1	1	11
3	2	1	3
4	3	0	22
5	3	2	6
6	5	0	-15
	6（矩阵的行数）		
	5（矩阵的列数）		
	7（非零元个数）		

← 第0列第1个非零元素

第0列有2个非零元素

← 第1列第1个非零元素

5.3 矩阵的压缩存储——稀疏矩阵

三元组顺序表转置算法——算法II

基本思想：顺序取，直接存。即在*A*中依次取三元组，交换其行号和列号放到*B* 中适当位置。

如何确定当前从*A*中取出的三元组在*B*中的位置？

分析：*A*中第0列的第一个非零元素一定存储在*B*中下标为0的位置上，该列中其它非零元素应存放在*B*中后面连续的位置上，那么第1列的第一个非零元素在*B*中的位置便等于第0列的第一个非零元素在*B*中的位置加上第0列的非零元素的个数，以此类推。

5.3 矩阵的压缩存储——稀疏矩阵

三元组顺序表转置算法——算法II

数据结构设计：

引入两个数组作为辅助数据结构：

num[nu]：存储矩阵*A*中某列的非零元素的个数；

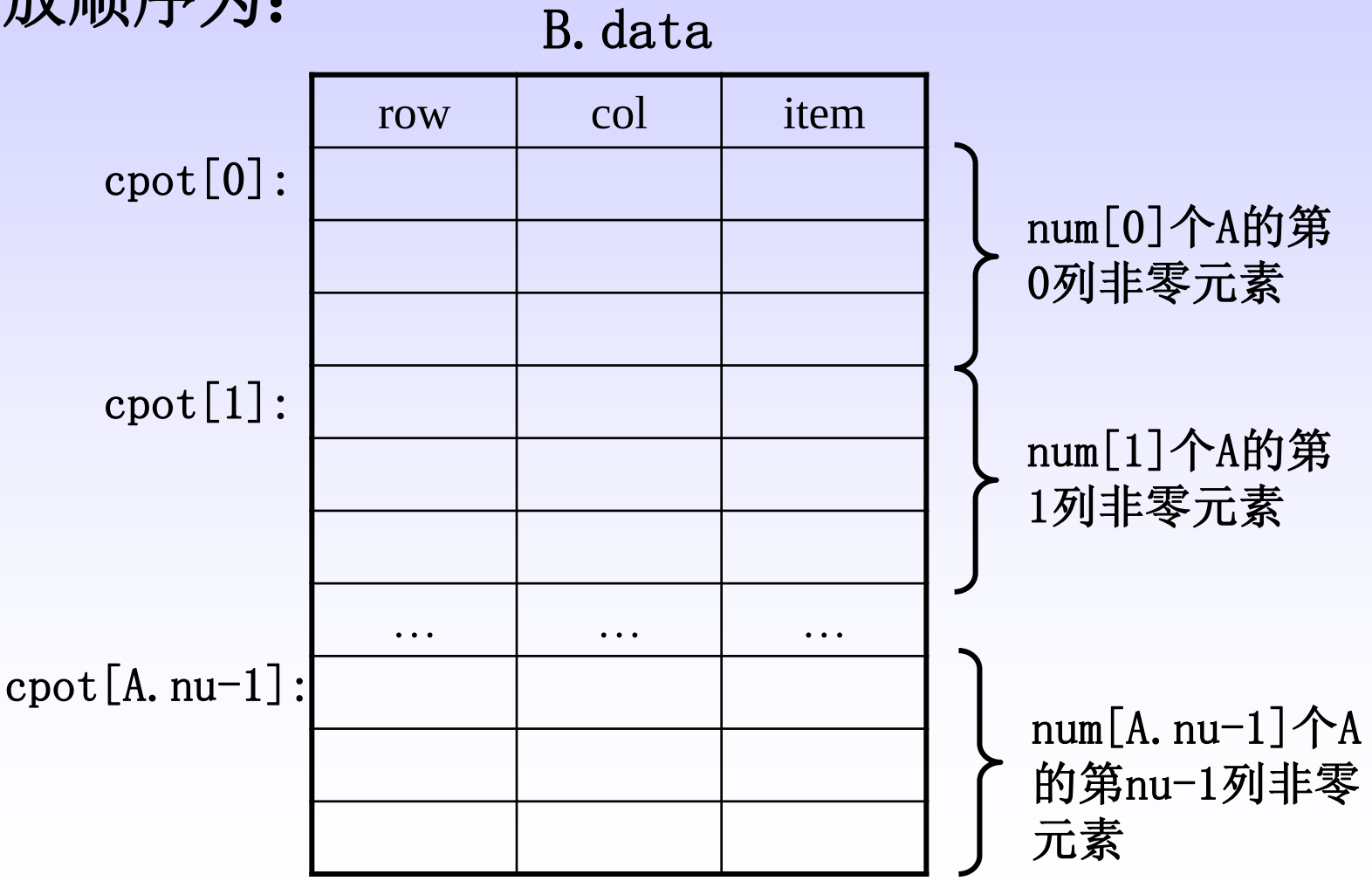
cpot[nu]：表示矩阵*A*中某列的第一个非零元素在B中的位置。

num与cpot存在如下递推关系：

$$\begin{cases} \text{cpot}[0]=0; \\ \text{cpot}[col]=\text{cpot}[col-1]+\text{num}[col-1]; \quad 1\le col<nu \end{cases}$$

5.3 矩阵的压缩存储——稀疏矩阵

矩阵A的非零元素在转置矩阵B的三元组顺序表中的存放顺序为：



5.3 矩阵的压缩存储——稀疏矩阵

根据矩阵*A*计算num和cpot

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
Max−1	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

col	0	1	2	3	4	5
num[col]	2	1	1	2	0	1
cpot[col]	0	2	3	4	6	6

5.3 矩阵的压缩存储——稀疏矩阵

将矩阵A中col列元素存放在*B*中下标为cpot[col]的位置

	row	col	item
→ 0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空	空	空
	闲	闲	闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item	
0	0	0	15	←cpot[0]
1				←cpot[0]
2				←cpot[1]
3				←cpot[2]
4				←cpot[3]
5				
6				←cpot[4] cpot[5]
	6（矩阵的行数）			
	5（矩阵的列数）			
	7（非零元个数）			

5.3 矩阵的压缩存储——稀疏矩阵

将矩阵A中col列元素存放在*B*中下标为cpot[col]的位置

	row	col	item
0	0	0	15
→ 1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空 闲	空 闲	空 闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item	
0	0	0	15	
1				←cpot[0]
2				←cpot[1]
3				←cpot[2]
4	3	0	22	←cpot[3]
5				←cpot[3]
6				←cpot[4] cpot[5]
	6（矩阵的行数）			
	5（矩阵的列数）			
	7（非零元个数）			

5.3 矩阵的压缩存储——稀疏矩阵

将矩阵A中col列元素存放在*B*中下标为cpot[col]的位置

	row	col	item
0	0	0	15
1	0	3	22
→ 2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空 闲	空 闲	空 闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item	
0	0	0	15	
1				←cpot[0]
2				←cpot[1]
3				←cpot[2]
4	3	0	22	
5				←cpot[3]
6	5	0	-15	←cpot[4] cpot[5]
				←cpot[5]
	6（矩阵的行数）			
	5（矩阵的列数）			
	7（非零元个数）			

5.3 矩阵的压缩存储——稀疏矩阵

将矩阵A中col列元素存放在*B*中下标为cpot[col]的位置

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
→ 3	1	1	11
4	1	2	3
5	2	3	6
6	4	0	91
	空 闲	空 闲	空 闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item	
0	0	0	15	
1				←cpot[0]
2	1	1	11	←cpot[1]
3				←cpot[2] cpot[1]
4	3	0	22	
5				←cpot[3]
6	5	0	-15	←cpot[4]
				←cpot[5]
	6（矩阵的行数）			
	5（矩阵的列数）			
	7（非零元个数）			

5.3 矩阵的压缩存储——稀疏矩阵

将矩阵A中col列元素存放在*B*中下标为cpot[col]的位置

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
→ 4	1	2	3
5	2	3	6
6	4	0	91
	空 闲	空 闲	空 闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item	
0	0	0	15	
1				←cpot[0]
2	1	1	11	
3	2	1	3	←cpot[2] cpot[1]
4	3	0	22	←cpot[3]
5				←cpot[4]
6	5	0	-15	←cpot[5]
	6（矩阵的行数）			
	5（矩阵的列数）			
	7（非零元个数）			

5.3 矩阵的压缩存储——稀疏矩阵

将矩阵A中col列元素存放在*B*中下标为cpot[col]的位置

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
→ 5	2	3	6
6	4	0	91
	空 闲	空 闲	空 闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item	
0	0	0	15	
1				←cpot[0]
2	1	1	11	
3	2	1	3	←cpot[1]
4	3	0	22	←cpot[2]
5	3	2	6	←cpot[3]
6	5	0	-15	←cpot[4] cpot[3]
				←cpot[5]
	6（矩阵的行数）			
	5（矩阵的列数）			
	7（非零元个数）			

5.3 矩阵的压缩存储——稀疏矩阵

将矩阵A中col列元素存放在*B*中下标为cpot[col]的位置

	row	col	item
0	0	0	15
1	0	3	22
2	0	5	−15
3	1	1	11
4	1	2	3
5	2	3	6
→ 6	4	0	91
	空 闲	空 闲	空 闲
	5（矩阵的行数）		
	6（矩阵的列数）		
	7（非零元个数）		

	row	col	item	
0	0	0	15	
1	0	4	91	←cpot[0]
2	1	1	11	←cpot[0]
3	2	1	3	←cpot[1]
4	3	0	22	←cpot[2]
5	3	2	6	
6	5	0	-15	←cpot[4] cpot[3]
				←cpot[5]
	6（矩阵的行数）			
	5（矩阵的列数）			
	7（非零元个数）			

5.3 矩阵的压缩存储——稀疏矩阵

三元组顺序表转置算法II——伪代码

1. 设置转置后矩阵B的行数、列数和非零元素的个数；
2. 计算A中每一列的非零元素个数；
3. 计算A中每一列的第一个非零元素在B中的下标；
4. 依次取A中的每一个非零元素对应的三元组；
   - 4.1 确定该元素在B中的下标pb；
   - 4.2 将该元素的行号列号交换后存入B中pb的位置；
   - 4.3 预置该元素所在列的下一个元素的存放位置；

5.3 矩阵的压缩存储——稀疏矩阵

```
void fasttranstri( TSMatrix  A, TSMatrix &B )
{   B.mu=A.nu; B.nu=A.mu; B.tu=A.tu;
    if(B.tu){
      for(col=0; col<A.nu; ++col)   num[col]=0;    //initiate num[]
      for(k=0; k<A.tu; ++k)          ++num[A.data[k].col];
      for(cpot[0]=0,col=1; col<A.nu; ++col)
          cpot[col]=cpot[col-1]+num[col-1];
      for(pa=0; pa<A.tu; ++pa)
      {     col=A.data[pa].col;
            pb=cpot[col];
            B.data[pb].row=A.data[pa].col;
            B.data[pb].col  =A.data[pa].row;
            B.data[pb].item=A.data[pa].item;
            ++cpot[col];
      }
    }
}
```

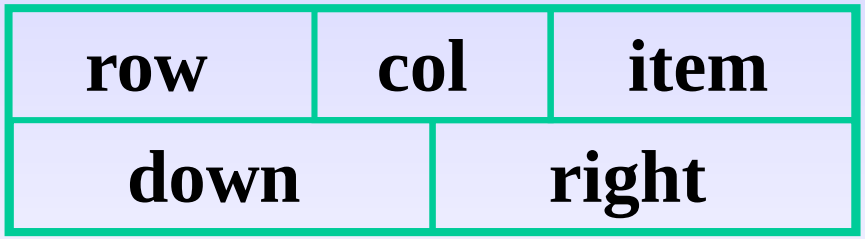
5.3 矩阵的压缩存储——稀疏矩阵

```
void fasttranstri( TSMatrix  A, TSMatrix &B )
{   B.mu=A.nu; B.nu=A.mu; B.tu=A.tu;
    if(B.tu){
      for(col=0; col<A.nu; ++col)   num[col]=0;    //initiate num[]
      for(k=0; k<A.tu; ++k)          ++num[A.data[k].col];   // calculate num[]
      for(cpot[0]=0,col=1; col<A.nu; ++col)
          cpot[col]=cpot[col-1]+num[col-1];          // calculate cpot[]
      for(pa=0; pa<A.tu; ++pa)                       // 依次处理A的每个非零元素
      {     col=A.data[pa].col;
            pb=cpot[col];
            B.data[pb].row=A.data[pa].col;
            B.data[pb].col  =A.data[pa].row;
            B.data[pb].item=A.data[pa].item;
            ++cpot[col];
      }
    }
}
```

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储——十字链表

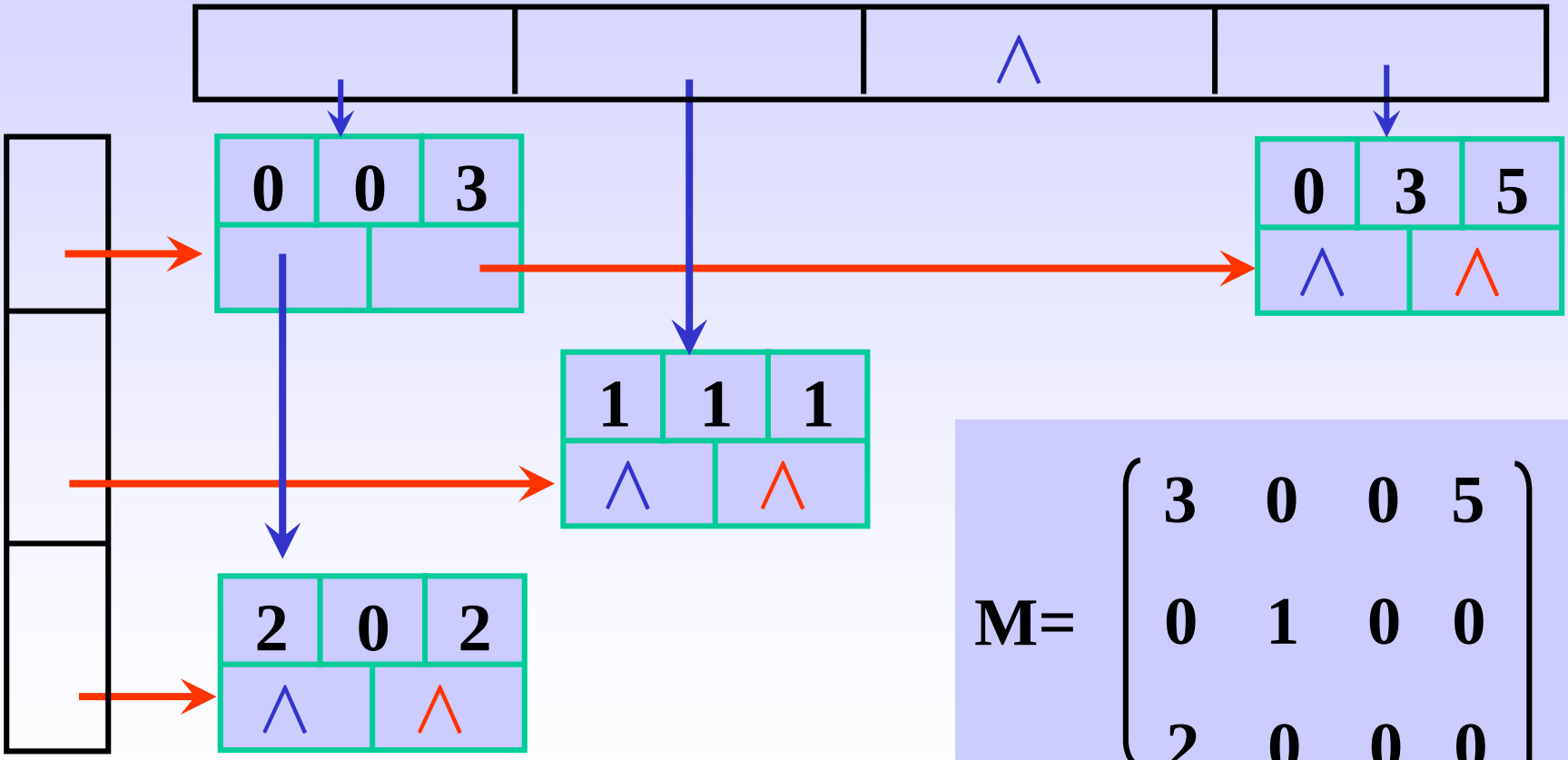
采用链接存储结构存储三元组表，每个非零元素对应的三元组存储为一个链表结点，结构为：



row：存储非零元素的行号
col：存储非零元素的列号
item：存储非零元素的值
right：指针域，指向同一行中的下一个三元组
down：指针域，指向同一列中的下一个三元组

5.3 矩阵的压缩存储——稀疏矩阵

稀疏矩阵的压缩存储——十字链表



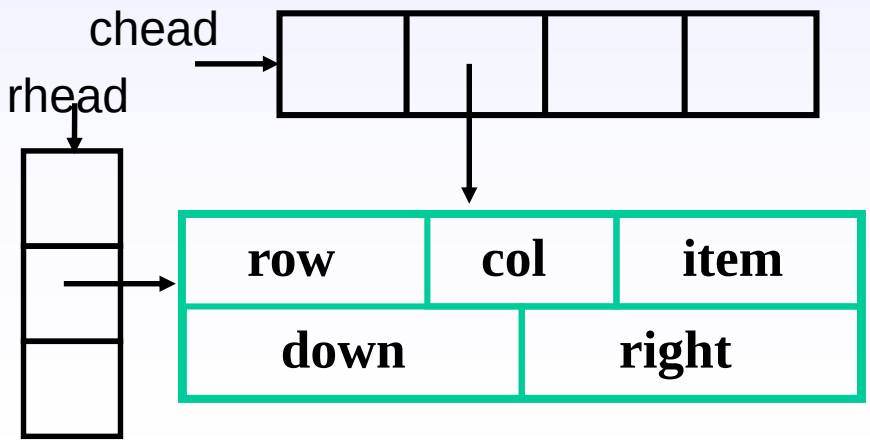
$$M=\begin{pmatrix} 3 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{pmatrix}$$

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
  - 5.3.1 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

5.3 矩阵的压缩存储——稀疏矩阵

```
typedef struct OLNode {
        int                 i, j;
        ElemType            e;
        Struct OLNode *right, *down;
    }OLNode, *OLink;
typedef struct {
        OLink *rhead, *chead;
        int       mu, nu, tu;
    }CrossList
```



5.3 矩阵的压缩存储——稀疏矩阵

```
Status CreateSMatrix_OL(CrossList &M)
{
  scanf(&m, &n, &t);
  M.Mu = m; M.nu = n; M.tu = t;
  if (!(M.rhead = (OLink *)malloc((m+1)*size(Olink)))) exit(Overflow);
  if (!(M.chead = (OLink *)malloc((n+1)*size(Olink)))) exit(Overflow);
  M.rhead[] = M.chead[] = NULL; //行指针列指针初始化
  for (scanf (&i, &j, &e); i != 0; scanf (&i, &j, &e))
   {  if (!(p = (OLNode *) malloc (size(OLNode))))   exit(Overflow);
      p->i = i; p->j = j; p->e = e;
      if (M.rhead[i] = = NULL ||M.rhead[i]->j > j)  {insert p into the row link;}
      else {search for the valid space and insert p;}
      if (M.chead[j] = = NULL ||M.rhead[j]->i > i)  {insert p into the column link;}
      else {search for the valid space and insert p;}
    }
  return OK;
}
```

5.4 广义表的定义

广义表的基本概念

广义表：n（$n\geq 0$）个数据元素的有限序列，记作：

$$LS=(a_1, a_2, \ldots\ldots, a_n)$$

其中：LS是广义表的名称，a_i（$1\leq i\leq n$）是LS的成员（或直接元素），它可以是单个的数据元素，也可以是一个广义表，分别称为LS的单元素（或原子）和子表。

通常用大写字母表示广义表，用小写字母表示单元素。

广义表是线性表的推广。

广义表的逻辑结构：直接元素之间是线性关系。

5.4 广义表的定义

广义表的示例

$A = ( )$

$B = (e)$

$C = (a, (b, c, d))$

$D = (A, B, C)$

$E = (a, E)$

$F = (( ))$

5.4 广义表的定义

广义表的基本概念

长度：广义表*LS*中的直接元素的个数；

深度：广义表*LS*中括号的最大嵌套层数。

表头：广义表*LS*非空时，称第一个元素为*LS*的表头；

表尾：广义表*LS*中除表头外其余元素组成的广义表。

广义表的表头为原子或列表，表尾为列表。

广义表( )和广义表(( ))不同？

5.4 广义表的定义

广义表的示例

$A = ( )$

$B = (e)$

$C = (a, (b, c, d))$

$D = (A, B, C)$

$E = (a, E)$

$F = (( ))$

5.4 广义表的定义

广义表的图形表示

$A = ( )$

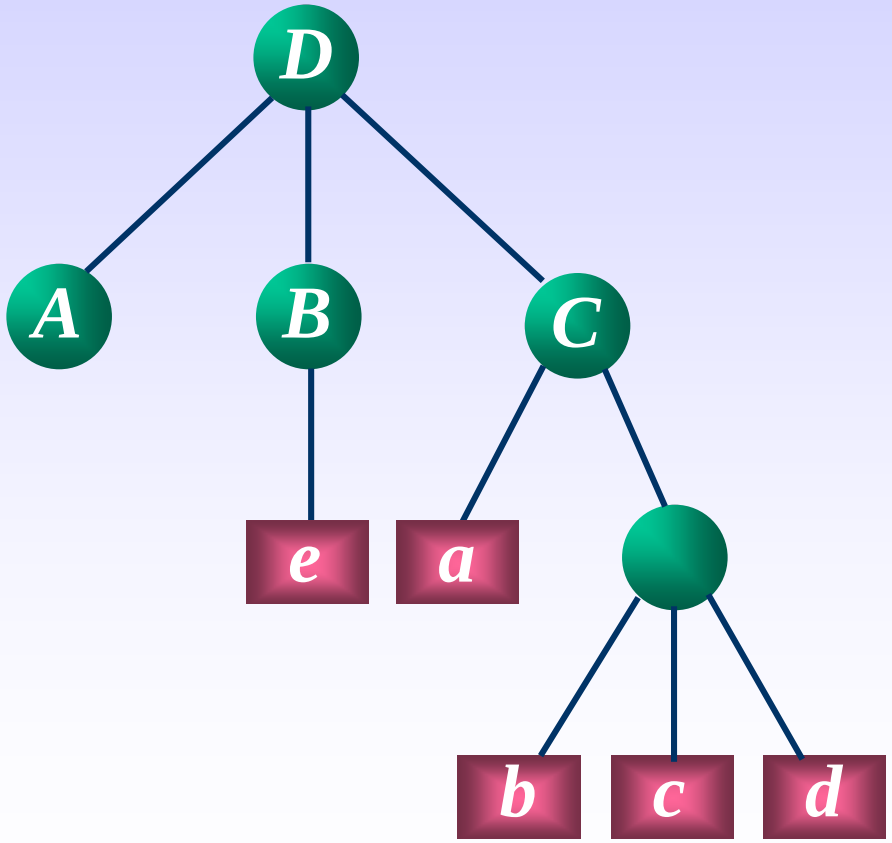
$B = (e)$

$C = (a, (b, c, d))$

$D = (A, B, C)$

$E = (a, E)$

$F = (( ))$



5.4 广义表的定义

ADT GList {

数据对象：D＝{ e_i | e_i ∈AtomSet 或 e_i ∈Glist, i=1,2,...,n, n≥0 }

数据关系：R1＝{ <e_{i-1} ,e_i >| e_{i-1}, e_i ∈D, i=2,...,n }

基本操作：

InitGList(&L);

操作结果：创建空的广义表L.

CreateGlist(&L, S);

初始条件：S是广义表的书写形式串；

操作结果：由S创建广义表L.

DestroyGlist(&L);

操作结果：销毁广义表L.

GetHead(L);

操作结果：取广义表L的头.

GetTail(L);

操作结果：取广义表L的尾. ……

}

5.5 广义表的存储结构

广义表可以采用顺序存储结构吗？

由于广义表中的数据元素的类型不统一，因此难以采用顺序存储结构来存储。

如何采用链接存储结构存储广义表？

若广义表不空，则可分解为表头和表尾；反之，一对确定的表头和表尾可唯一地确定一个广义表。

采用头尾表示法存储广义表

5.5 广义表的存储结构

广义表的存储结构——头尾表示法

广义表中的数据元素既可以是广义表也可以是单元素

表结点——存储广义表；元素结点——存储单元素

头尾表示法中的结点结构？

5.5 广义表的存储结构

广义表的存储结构——头尾表示法

结点结构

表结点　　元素结点

tag：区分表结点和元素结点的标志；

hp：指向表头结点的指针；

tp：指向表尾结点的指针；

data：数据域，存放单元素。

5.5 广义表的存储结构

广义表的存储结构——头尾表示法

定义结点结构

```
struct GLNode
{
   int tag;
   union
   {
      ElemType data;
      struct
      {
         GLNode *hp, *tp;
       } ptr;
    };
};
```

tag=1	hp	tp
tag=0	data	

5.5 广义表的存储结构

若广义表不空，则可分解为表头和表尾：

这个广义表一定是一个表节点

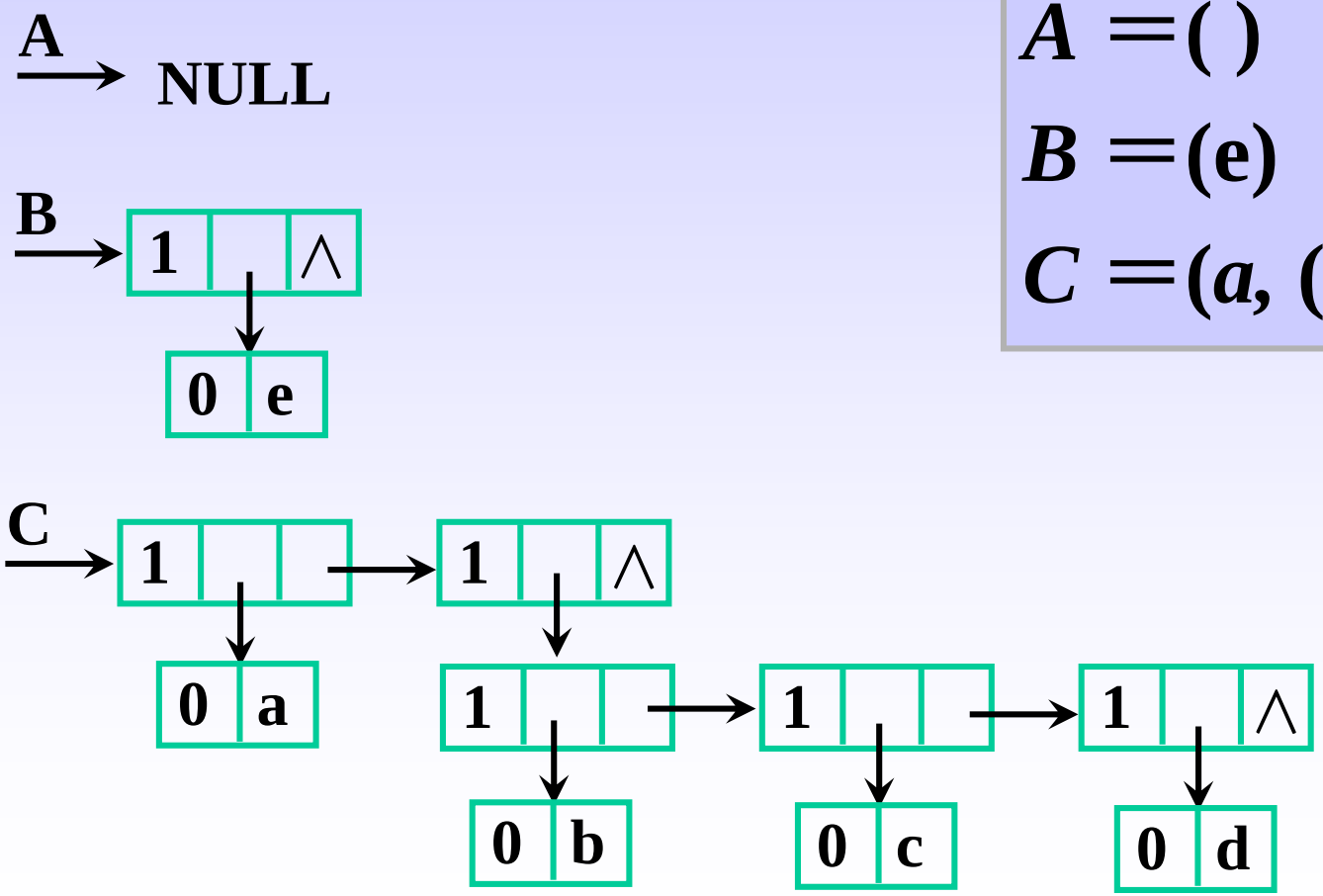
5.5 广义表的存储结构

广义表的存储结构——头尾表示法

$A = ()$

$B = (e)$

$C = (a, (b, c, d))$



5.5 广义表的存储结构

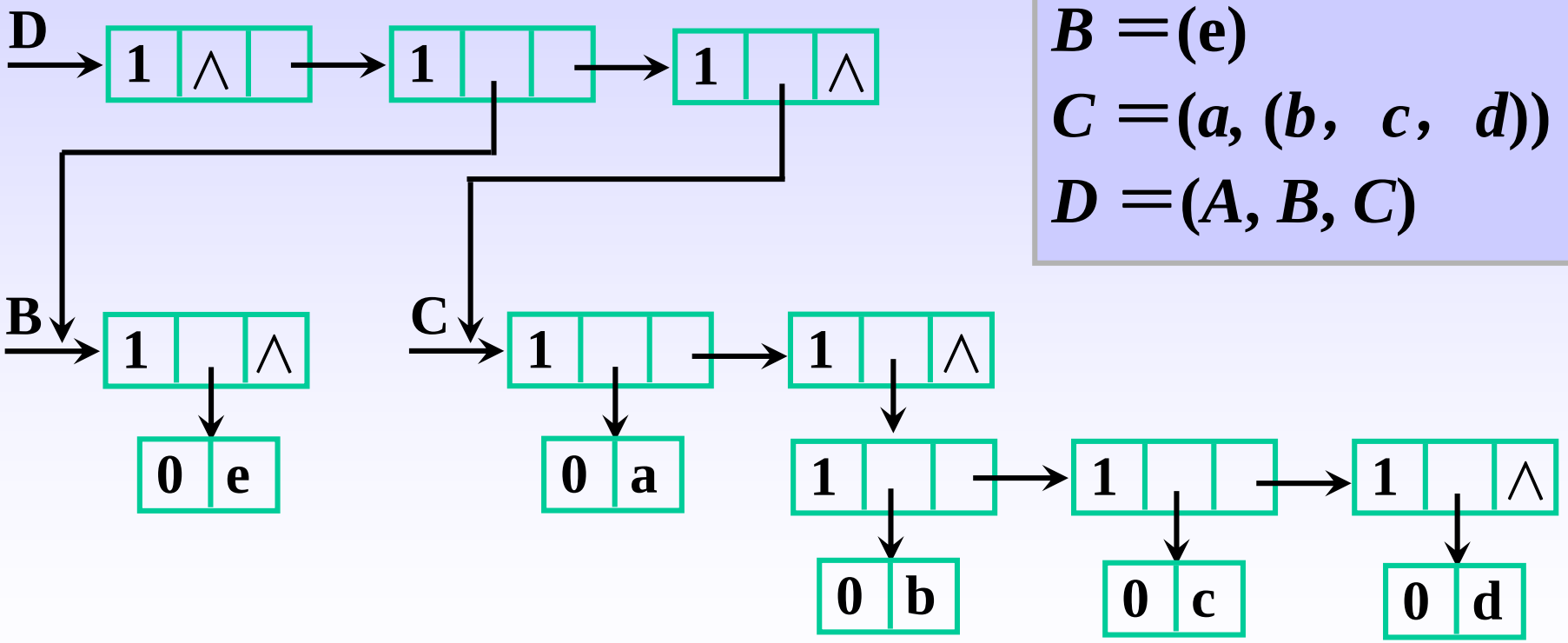
广义表的存储结构——头尾表示法

$A = ( )$

$B = (e)$

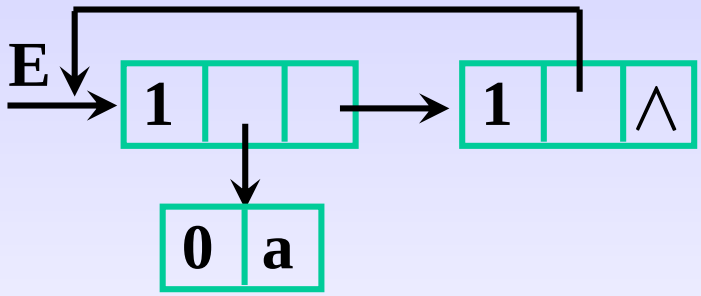
$C = (a, (b, c, d))$

$D = (A, B, C)$



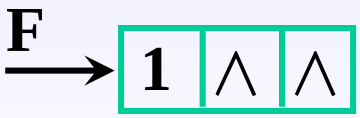
5.5 广义表的存储结构

广义表的存储结构——头尾表示法



$E=(a, E)$

$F=(())$



$A=()$

5.5 广义表的存储结构

广义表的存储结构——头尾表示法

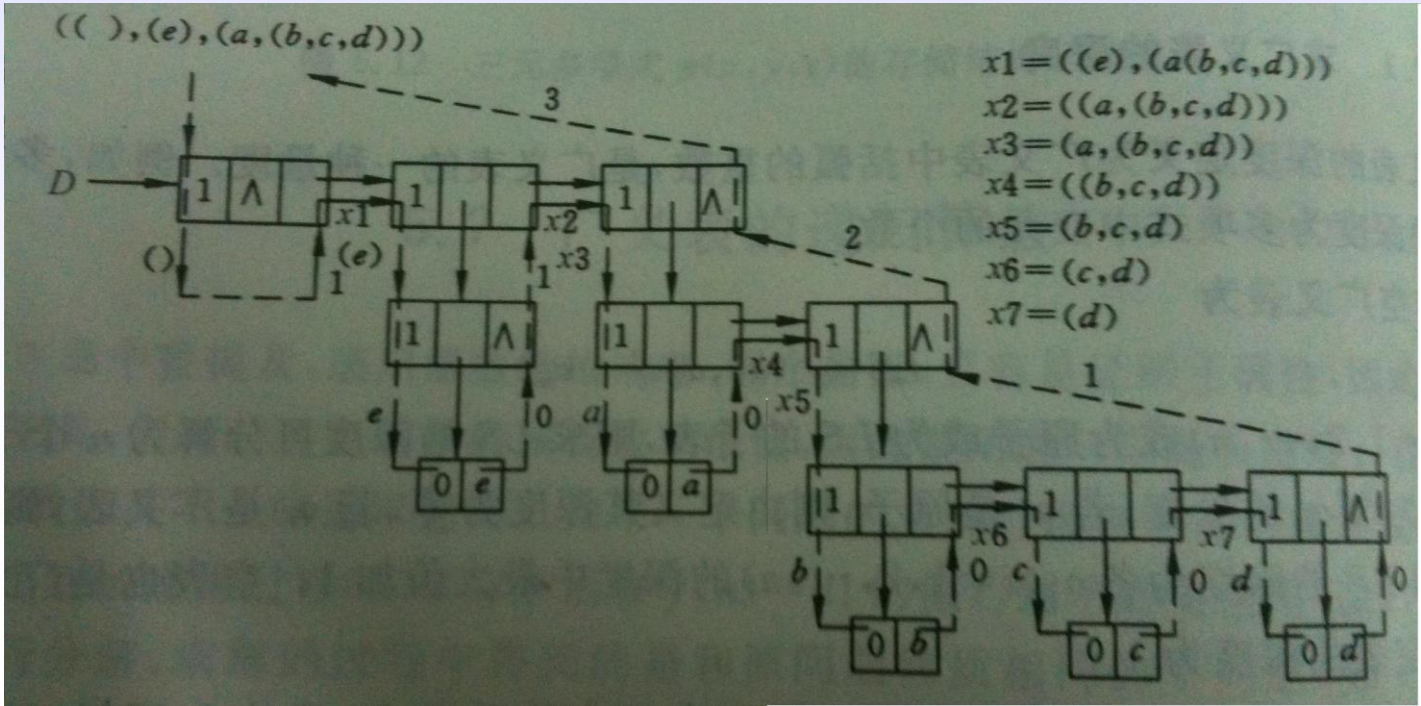
定义结点结构

```
struct GLNode
{
   int tag;
   union
   {
      ElemType data;
      struct
      {
         GLNode *hp, *tp;
       } ptr;
    };
};
```

tag=1	hp	tp
tag=0	data	

求广义表的深度

```
int GListDepth(GList L)
{  if(!L) return 1;
   if(L->tag == ATOM) return 0;
   for (max = 0, pp = L; pp; pp = pp->ptr.tp)
    {   dep = GListDepth(pp->ptr.hp);
        if (dep > max)
             max = dep;
    }
    return max + 1;
}
```



5.6 m元多项式的表示

设三元多项式P(x,y,z)为：

$$P(x,y,z)=x^{10}y^3z^2+2x^6y^3z^2+3x^5y^2z^2+x^4y^4z+6x^3y^4z+2yz+15$$

5.6 m元多项式的表示

设三元多项式P(x,y,z)为：

$$P(x,y,z)=x^{10}y^3z^2+2x^6y^3z^2+3x^5y^2z^2+x^4y^4z+6x^3y^4z+2yz+15$$

将P(x,y,z)改写为：

$$P(x,y,z)=((x^{10}+2x^6)y^3+3x^5y^2)z^2+((x^4+6x^3)y^4+2y)z+15$$

$$=Az^2+Bz+15$$

则P(x,y,z)可用广义表表示为：P=z((A,2), (B,1), (15,0))

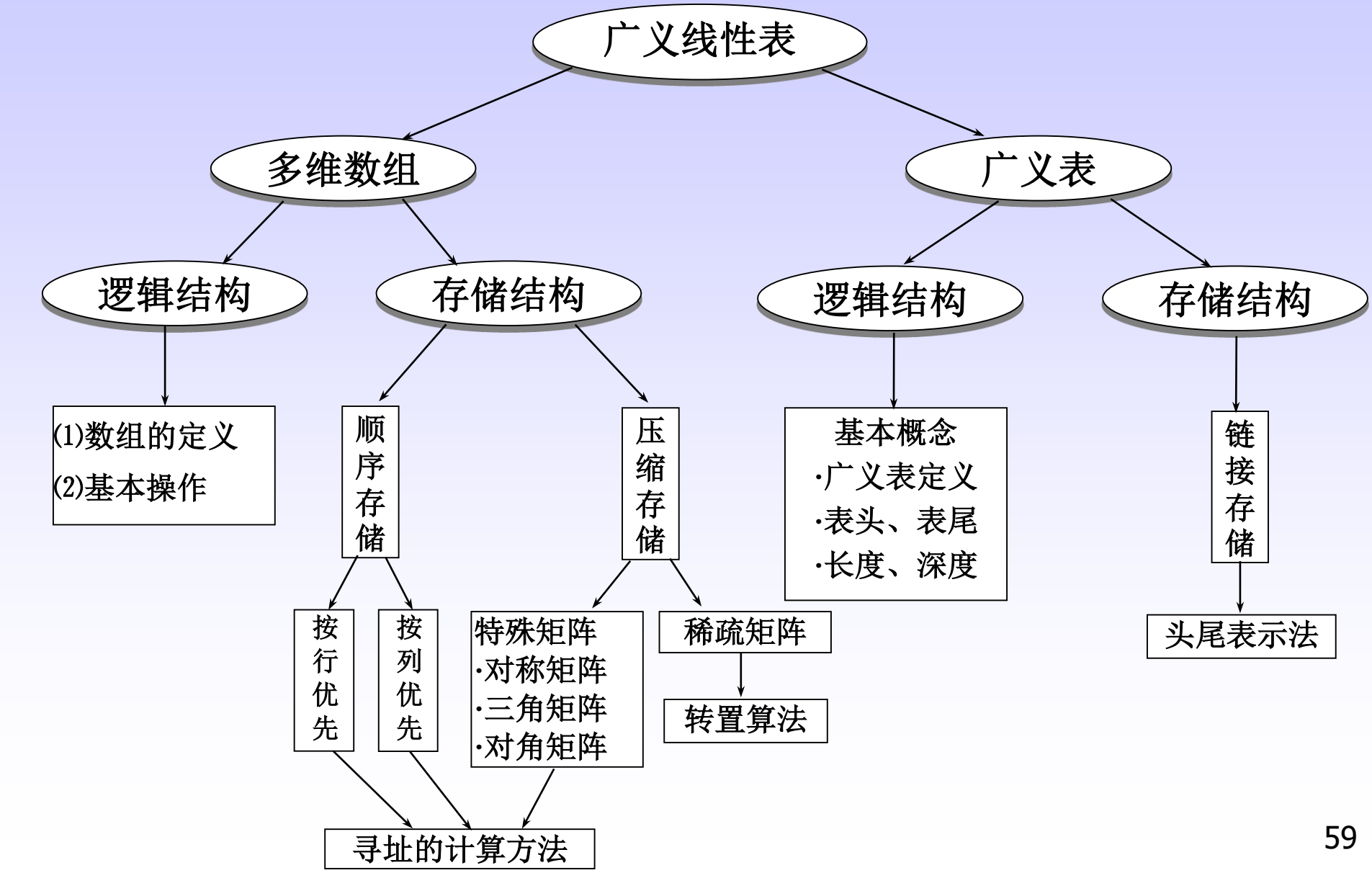
其中 A=y((C,3), (D,2))

C=x((1,10), (2,6)) D=x((3,5))

B=y((E,4), (F,1))

E=x((1,4), (6,3)) F=x((2,0))

本章小结


广义线性表
多维数组
广义表
逻辑结构
存储结构
逻辑结构
存储结构
(1)数组的定义
(2)基本操作
顺序存储
压缩存储
基本概念
·广义表定义
·表头、表尾
·长度、深度
链接存储
按行优先
按列优先
特殊矩阵
·对称矩阵
·三角矩阵
·对角矩阵
稀疏矩阵
头尾表示法
转置算法
寻址的计算方法

线性表小结

线性表——具有相同类型的数据元素的有限序列。

↓ 限制插入、删除位置

特殊线性表

- 栈——仅在表尾进行插入和删除操作的线性表。
- 队列——在一端进行插入操作，而另一端进行删除操作的线性表。
- 串——零个或多个字符组成的有限序列 。

↑ 限制元素类型为字符，操作以子串为对象

线性表——具有相同类型的数据元素的有限序列。

线性表小结

线性表——具有相同类型的数据元素的有限序列。

将元素的类型进行扩充

广义线性表

（多维）数组——线性表中的数据元素可以是线性表，但所有元素的类型相同。

广义表——线性表中的数据元素可以是线性表，且元素的类型可以不相同。

作业

- 第三章 栈和队列
  - 1.说明如何用两个栈来实现一个队列，用C写出你的算法实现，并分析出队、入队操作的时间复杂度。
  - 2.说明如何用两个队列来实现一个栈，用C写出你的算法实现，并分析出栈、入栈操作的时间复杂度。
- 第四章串
  - 1.用C编写函数strdel，参量是字符串string1和字符串string2，并且sizeof(string1)>=sizeof(string2)。，删除string1中第一次出现的string2，最后返回string1.

作业

- 《数据结构题集》
  - P.22、23：3.4(1)、3.6
  - P.28：4.6、4.8

作业

- 补充题
  - 矩阵A由m行n列元素构成，请用C编写算法，分别利用数组和广义表实现矩阵的转置运算。并分析两种算法的时间和空间复杂度。
- 《数据结构题集》
  - P.32：5.9
  - P.34： 5.17(1)(2)(3)
- 下下周二（11月1日）上课时交