第5章 数组和广义表

本章重点与难点

- **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。 广义表的基本概念。

- **难点：**

稀疏矩阵的压缩存储表示和实现算法。

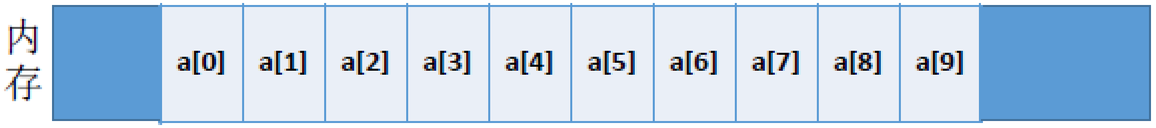
第五章 数组和广义表

第五章 数组和广义表

5.1 数组的类型定义

- 数组：
  - 是由下标（index）和值（value）组成的序对（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。

一维数组：$(a_1, a_2, a_3, \cdots, a_i, \cdots, a_n)$



5.1 数组的类型定义

- 数组：
  - 是由下标（index）和值（value）组成的序对（index，value）的集合。

二维数组：对二维数组可以理解成一维数组，其每个元素又是一个一维数组；

$$A=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix} \Rightarrow$$

$A=(A_1，A_2，\ldots\ldots，A_n)$

其中：

$A_i=(a_{1i}，a_{2i}，\ldots\ldots，a_{mi})$

$(1\le i\le n)$

5.1 数组的类型定义

- **数组：**
- **多维数组：每个元素受n(n≥1)个线性关系的约束，每个元素在n个线性关系中的序号i_1、i_2、…、i_n称为该元素的下标，并称该数组为n 维数组。**
- 数组仍是一种特殊形式的线性表，具有线性关系。
  - 多维数组包含多个关系，每个关系都是一个线性关系，因此仍然是线性结构。

5.1 数组的类型定义

- 基本操作：
  - 初始化： Create（）
    - 建立一个空数组；
    - int A[ ][ ]
  - 存取：Retrieve（array，index）
    - 给定一组下标，读出对应的数组元素；
    - A[i][j]
  - 修改：Store（array，index，value）：
    - 给定一组下标，存储或修改与其相对应的数组元素。
    - A[i][j]=8。
  - 无需插入和删除操作

第五章 数组和广义表

5.2 数组的顺序表示和实现

- **数组的存储结构：**
  - 数组没有插入和删除操作，所以，不用预留空间，适合采用顺序存储。
  - 数组是多维的结构，而存储空间是一个一维的结构。
  - 数组的顺序存储
    - 用一组连续的存储单元来实现（多维）数组的存储。
    - 高维数组可以看成是由多个低维数组组成的。

5.2 数组的顺序表示和实现

- 二维数组的存储与寻址
- 常用的映射（存储）方法有两种：
- 按行优先：先行后列，先存储行号较小的元素，行号相同者先存储列号较小的元素。（高级语言一般以行序为主序）
- 按列优先：先列后行，先存储列号较小的元素，列号相同者先存储行号较小的元素。（例如matlab）

例：**ElementType A[2][3]；**

$$A = \begin{bmatrix} A[0][0] & A[0][1] & A[0][2] \\ A[1][0] & A[1][1] & A[1][2] \end{bmatrix}$$

	内存	内存	
第一行	A[0][0]	A[0][0]	第一列
	A[0][1]	A[1][0]	
	A[0][2]	A[0][1]	第二列
第二行	A[1][0]	A[1][1]	
	A[1][1]	A[0][2]	第三列
	A[1][2]	A[1][2]	

5.2 数组的顺序表示和实现

- 数组元素的地址关系：

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

5.2 数组的顺序表示和实现

- 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:

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

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:

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

第五章 数组和广义表

5.3 矩阵的压缩存储

- 特殊矩阵的压缩存储：
  - 特殊矩阵：很多值相同的元素或者零元素在矩阵中的分布有一 定的规律。
  - 稀疏矩阵：矩阵中零元素的个数远远多于非零元素。
- 压缩存储的基本思想是：
  - 为多个值相同的元素只分配一个存储空间；
  - 对特定值的（如零）元素不分配存储空间。

5.3 矩阵的压缩存储

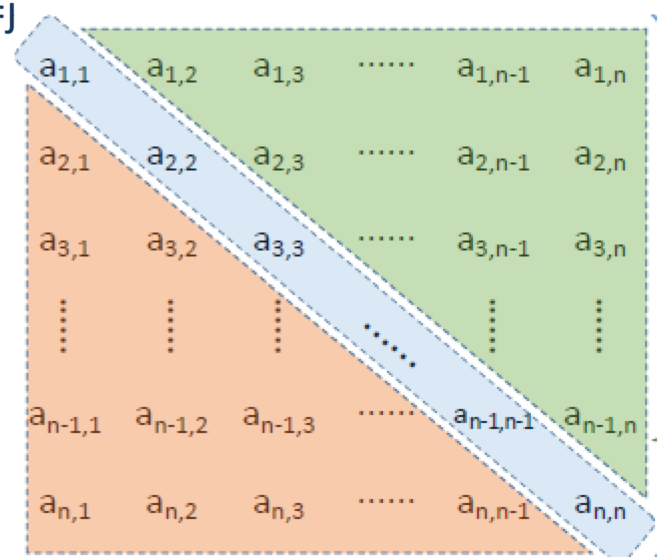
- **对称矩阵**

若n 阶矩阵 A 中的元素满足下述性质

$$a_{ij} = a_{ji} \qquad 1 \le i,\ j \le n$$

则称 n 阶对称阵。

主对角线：i=j



上三角区：i>j

下三角区：i>j

压缩存储策略：只存储主对角线+下三角区（或主对角线+上三角区）

5.3 矩阵的压缩存储

- 对称矩阵

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

压缩存储策略：只存储主对角线+下三角区（或主对角线+上三角区）

需要多大存储空间？

B[k]	a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	…	a_{ij}	…	a_{n1}	…	a_{nn}
k =	1	2	3	4	5	6		i(i-1)/2+j		n(n-1)/2+1		n(n+1)/2

5.3 矩阵的压缩存储

- 对称矩阵

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots\cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots\cdots & a_{2,n-1} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots\cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots\cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots\cdots & a_{n,n-1} & a_{n,n} \end{pmatrix}$$

压缩存储策略：只存储主对角线+下三角区（或主对角线+上三角区）

B[k]												
	a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	…	a_{ij}	…	a_{n1}	…	a_{nn}

如何使用？

B[k] 和 a_{ij} 的一一对应关系为:

$$k = \begin{cases} i(i-1)/2 + j & \text{当 } i \geq j \\ j(j-1)/2 + i & \text{当 } i < j \end{cases}$$

5.3 矩阵的压缩存储

- 对称矩阵

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \cdots\cdots & a_{1,n-1} & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots\cdots & a_{2,n-1} & a_{2,n} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots\cdots & a_{3,n-1} & a_{3,n} \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots\cdots & a_{n-1,n-1} & a_{n-1,n} \\ a_{n,1} & a_{n,2} & a_{n,3} & \cdots\cdots & a_{n,n-1} & a_{n,n} \end{bmatrix}$$

压缩存储策略：只存储主对角线+下三角区（或主对角线+上三角区）

B[k]	a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	…	a_{ij}	…	a_{n1}	…	a_{nn}

如何使用？

a[*i*][*j*]=A(*B*, *i*, *j*)

```
ElementType   A(ElementType B[ ] , int i, int j)
{
   int k ;
   if ( i >= j )
         k = i*(i-1)/2 + j ;
   else
         k =j ( j – 1 ) / 2 + i ;
         return( B[k] ) ;
}
```

5.3.1 特殊矩阵

- 下(上)三角矩阵

$$
\begin{pmatrix}
a_{1,1} & c & c & \cdots & c & c \\
a_{2,1} & a_{2,2} & c & \cdots & c & c \\
a_{3,1} & a_{3,2} & a_{3,3} & \cdots & c & c \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
a_{n-1,1} & a_{n-1,2} & a_{n-1,3} & \cdots & a_{n-1,n-1} & c \\
a_{n,1} & a_{n,2} & a_{n,3} & \cdots & a_{n,n-1} & a_{n,n}
\end{pmatrix}
\qquad
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\
c & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\
c & c & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c & c & c & \cdots & a_{n-1,n-1} & a_{n-1,n} \\
c & c & c & \cdots & c & a_{n,n}
\end{pmatrix}
$$

下三角矩阵：除了主对角线和下三角区，其余的元素都相同

上三角矩阵：除了主对角线和上三角区，其余的元素都相同

5.3.1 特殊矩阵

- 下(上)三角矩阵

压缩存储策略：只存储主对角线+上三角区

$$
\begin{pmatrix}
a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,n-1} & a_{1,n} \\
c & a_{2,2} & a_{2,3} & \cdots & a_{2,n-1} & a_{2,n} \\
c & c & a_{3,3} & \cdots & a_{3,n-1} & a_{3,n} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
c & c & c & \cdots & a_{n-1,n-1} & a_{n-1,n} \\
c & c & c & \cdots & c & a_{n,n}
\end{pmatrix}
$$

B[k]	a_{11}	a_{21}	a_{22}	a_{31}	a_{32}	a_{33}	…	a_{ij}	…	a_{n1}	…	a_{nn}	C

思考题：按行优先的原则，$a_{i,j}$是第几个元素？

5.3.2 稀疏矩阵

- **稀疏矩阵** 零元素的个数远远多于非零元素的个数

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

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 $\delta \le 0.05$ 的矩阵为稀疏矩阵。

以常规方法，即以二维数组表示高阶的稀疏矩阵时产生的问题：
1）零值元素占的空间很大；
2）计算中进行了很多和零值的运算；

5.3.2 稀疏矩阵

- 解决问题的原则：
  - ① 尽可能少存或者不存零值元素；
  - ② 尽可能减少没有实际意义的运算；
  - ③ 运算方便，即：
    - 能尽可能快地找到与下标值（i，j）对应的元素；
    - 能尽可能快地找到同一行或同一列的非零值元素。

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

5.3.2 稀疏矩阵

- 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

5.3.3 矩阵的压缩存储

- 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：

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

1	2	1
2	4	2
3	1	1

```
#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;  //行数，列数和非零元的个数
} TSMatrix;                                      // 稀疏矩阵类型
```

失去随机存取的特性

5.3.3 矩阵的压缩存储

- 三元组顺序表转置的实现：
  - 示例

(1)从矩阵到转置矩阵

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

矩阵M

转置矩阵T

5.3.3 矩阵的压缩存储

- 求转置矩阵的操作：
  - 用常规的二维数组表示时的算法

```
for (col=1; col<=nu;++col)
   for (row=1; row<=mu; ++row)
      T[col][row] = M[row][col];
```

其时间复杂度为：O(mu * nu)

5.3.3 矩阵的压缩存储

- 三元组顺序表转置传统算法:

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

矩阵M

转值矩阵T

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

5.3.3 矩阵的压缩存储

- 三元组顺序表转置传统算法：

```
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;
}
```

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

5.3.3 矩阵的压缩存储

- 三元组顺序表转置算法时间复杂性：

```
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)

5.3.3 矩阵的压缩存储

- 三元组顺序表快速转置的实现：

$$M = \begin{pmatrix} 0 & 12 & 9 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -3 & 0 & 0 & 0 & 0 & 14 & 0 \\ 0 & 0 & 24 & 0 & 0 & 0 & 0 \\ 0 & 18 & 0 & 0 & 0 & 0 & 0 \\ 15 & 0 & 0 & -7 & 0 & 0 & 0 \end{pmatrix} \qquad T = M' = \begin{pmatrix} 0 & 0 & -3 & 0 & 0 & 15 \\ 12 & 0 & 0 & 0 & 18 & 0 \\ 9 & 0 & 0 & 24 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -7 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 14 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

i	j	v
1	3	-3
1	6	15
2	1	12
2	5	18
3	1	9
3	4	24
4	6	-7
6	3	14

5.3.3 矩阵的压缩存储

算法改进:

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

设向量

num[col], 表示矩阵M中第 col 列中非零元素的个数,

cpot[col] 指示M中第 col 列的第一个非零元素在 三元组中的恰当位置。

矩阵M的向量 cpot 的值

col	1	2	3	4	5	6	7
Num[col]	2	2	2	1	0	1	0
Cpot[col]	1	3	5	7	8	8	9

Cpot [1] = 1 ;

Cpot [col] = cpot[col – 1] + num[col – 1]

$2 \le col \le a.nu$

5.3.3 矩阵的压缩存储

t	i	j	v
→	1	2	12
	1	3	9
	3	1	-3
	3	6	14
	4	3	24
	5	2	18
	6	1	15
	6	4	-7

```
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];
```

col	1	2	3	4	5	6	7
Num[col]	2	2	2	1	0	1	0
Cpot[col]	1	3	5	7	8	8	9

5.3.3 矩阵的压缩存储

```
Void  FastTransposMatrix( 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 )
             {   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 ] ;
             }
         }
}
```

i	j	v
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

i	j	v
1	3	-3
1	6	15
2	1	12
2	5	18
3	1	9
3	4	24
4	6	-7
6	3	14

col	1	2	3	4	5	6	7
Num[col]	2	2	2	1	0	1	0
Cpot[col]	1	3	5	7	8	8	9

转置的改进算法:

T(n) = O(nu + tu)

5.3.3 矩阵的压缩存储

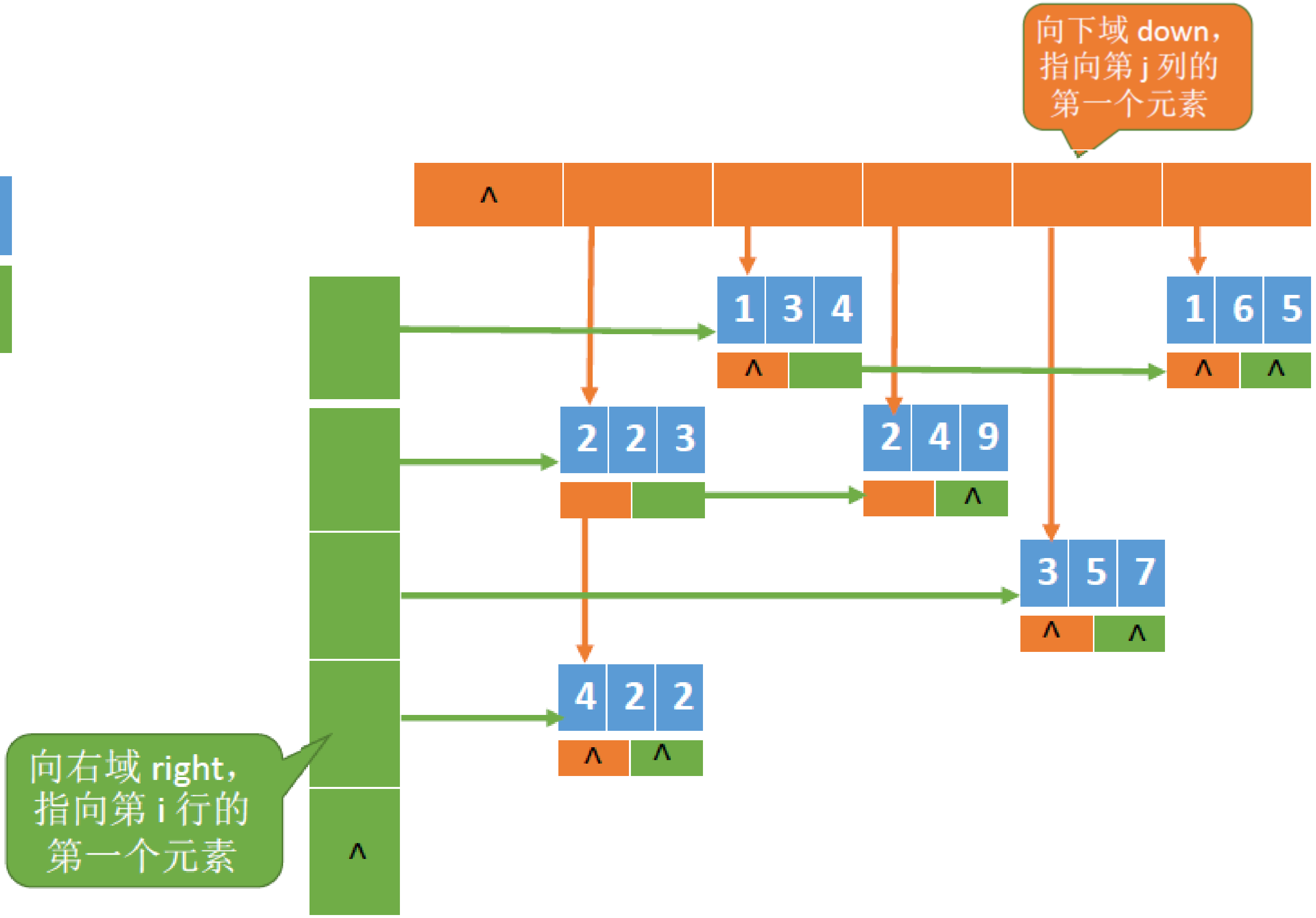
数组的链接式存储 多维数组,特别是稀疏矩阵可以采用链接式存储。

结点Node:

非零数据结点说明:

行	列	值
指向同列的下一个元素	指向同行的下一个元素	

$$\begin{pmatrix} 0 & 0 & 4 & 0 & 0 & 5 \\ 0 & 3 & 0 & 9 & 0 & 0 \\ 0 & 0 & 0 & 0 & 7 & 0 \\ 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$



第五章 数组和广义表

5.4 广义表的类型定义

- 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。

5.4 广义表的类型定义

- 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=（$a_1,a_2,\ldots\ldots a_n$)

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

子表：如果a_i是广义表，称为子表，用大写字母表示；
原子：如果a_i是单个元素，称为原子，用小写字母表示。

例如：$D = (E, F) = ((a, (b, c)),\ F)$

5.4 广义表的类型定义

广义表是递归定义的线性结构，

$$LS = (\alpha_1, \alpha_2, \cdots, \alpha_n)$$

其中：α_i 或为原子 或为广义表

例如：A = ( )

F = (d, (e))

D = ((a,(b,c)), F)

C = (A, D, F)

B = (a, B) = (a, (a, (a, ··· ,)))

5.4 广义表的类型定义

- 广义表 LS = (α_1, α_2, ..., α_n)的结构特点：　D = ((a,(b,c)), a)

1) 广义表中的数据元素有相对次序；

2) 广义表的长度定义为最外层包含元素个数；

3) 广义表的深度定义为所含括弧的重数；

注意：“原子”的深度为 0

“空表”的深度为 1

广义表的深度=Max {子表的深度} +1

4) 广义表可以共享（不必列出子表的值，而是通过子表的名称来引用）；

5) 广义表可以是一个递归的表。

递归表的深度是无穷值，长度是有限值。

5.4 广义表的类型定义

□广义表 LS = (α_1, α_2, ..., α_n)的结构特点:

6) 任何一个非空广义表 LS = (α_1, α_2, ..., α_n)均可分解为
表头 Head(LS) = α_1 和表尾 Tail(LS) = (α_2, ..., α_n) 两部分。

例如: D = (E, F) = ((a, (b, c)), F)

Head(D) = E　　Tail(D) = (F)

Head(E) = a　　Tail(E) = ((b, c))

Head(((b, c))) = (b, c)　　Tail(((b, c))) = ( )

Head((b, c)) = b　　Tail((b, c)) = (c)

Head((c)) = c　　Tail((c)) = ( )

5.4 广义表的类型定义

- 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,……）））） 。

5.4 广义表的类型定义

广义表是一个多层次的线性结构

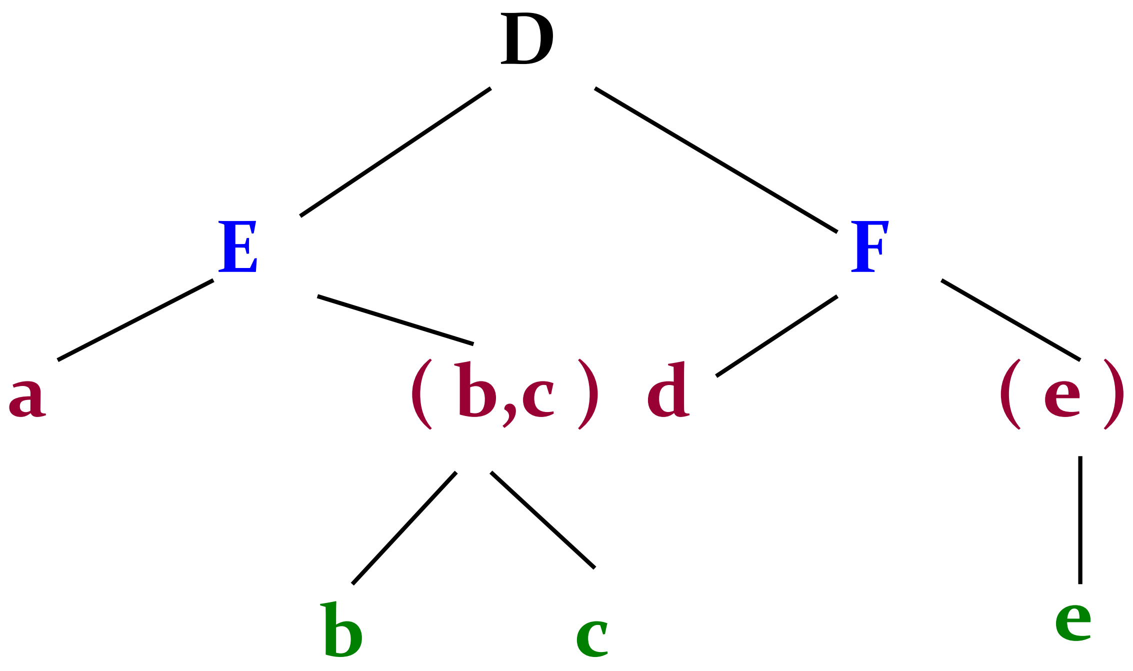
例如：

D=(E,F)

其中：

E=(a, (b, c))

F=(d, (e))



5.4 广义表的类型定义

- 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

5.4 广义表的类型定义

- 基本操作：

InitGlist(&L) //创建空表
DestroyGlist(&L) // 销毁
GListLength(L) //求表长
GListDepth(L) //求表的深度
GetHead(L) //取表头
GetTail(L) //取表尾

第五章 数组和广义表

5.5 广义表的存储结构

- 广义表表示方法

广义表从结构上可以分解成

广义表 ＝ 表头 ＋ 表尾 ⟹ 表头、表尾分析法

广义表 ＝ 子表1 + 子表2 + … + 子表n

⇓

子表分析法

5.5 广义表的存储结构

- 表头、表尾分析法

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指向表头，tp指向表尾

5.5 广义表的存储结构

- 广义表的头尾链表存储表示

```
typedef enum{ATOM,LIST} ElemTag;
typedef struct GLNode{
    ElemTag tag;
    union{
    AtomType atom;
    struct {struct GLNode *hp,*tp;}ptr;
    }
}*Glist;
```

tag=1	hp	tp

tag=0	atom

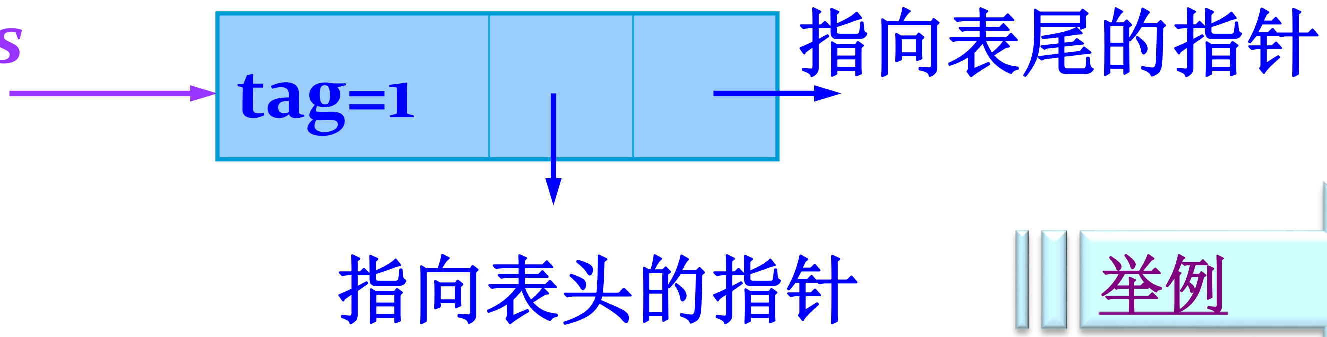
5.5 广义表的存储结构

- 表头、表尾分析法

1) 表头、表尾分析法：

空表　　*ls=Null*

非空表 *ls*

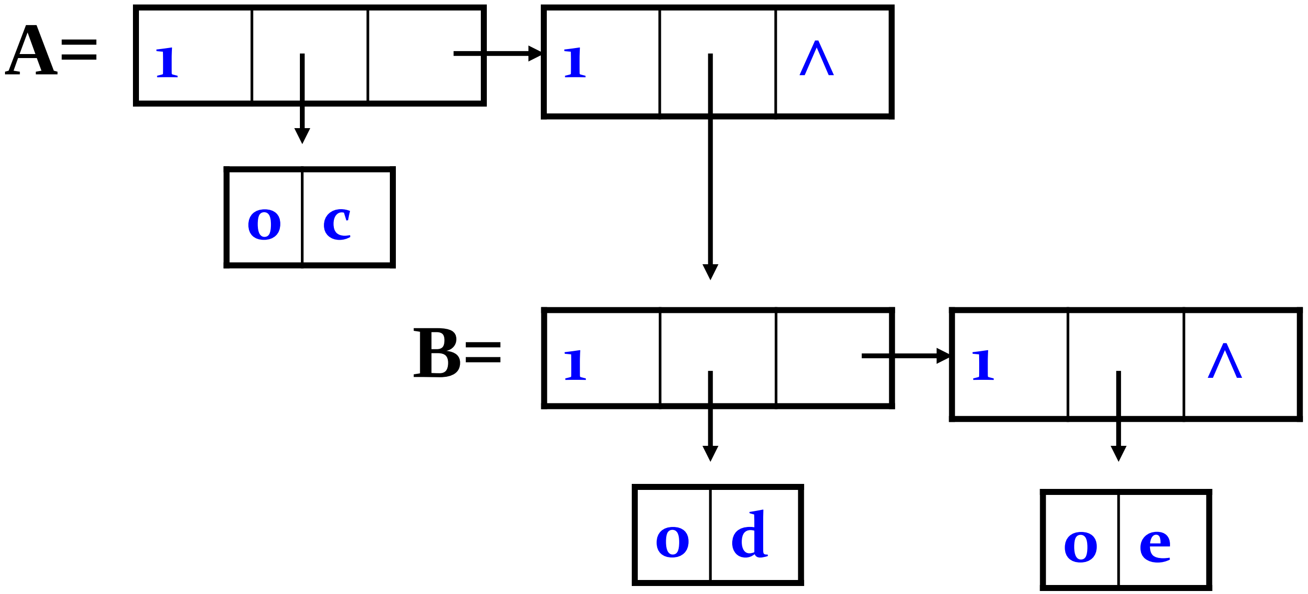


举例

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



5.5 广义表的存储结构

- 子表分析法

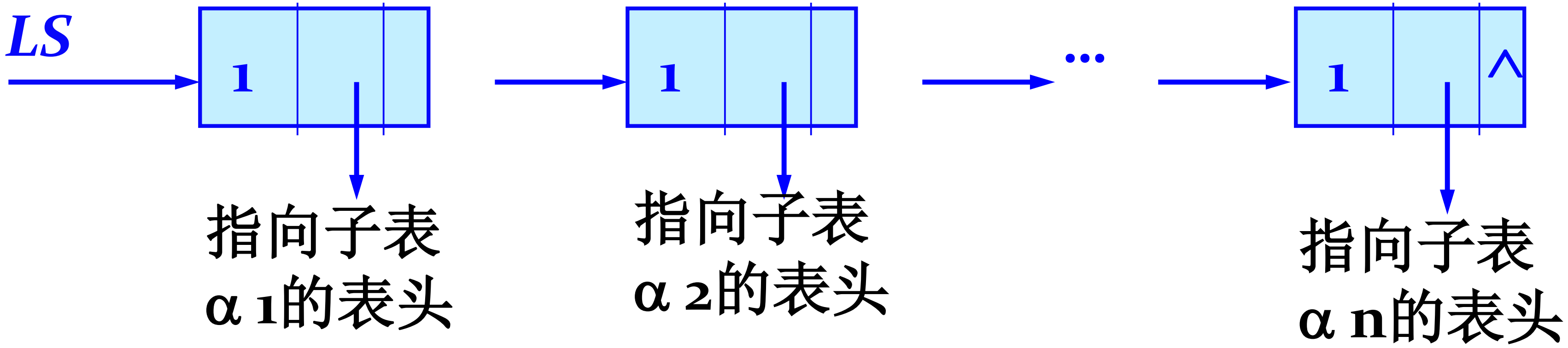
广义表 LS = (α1, α2, …, αn)包括n个子表，可以看成是线性链表

tag=1	hp	tp

tag=0	atom

空表　　ls=Null

非空表



5.5 广义表的存储结构

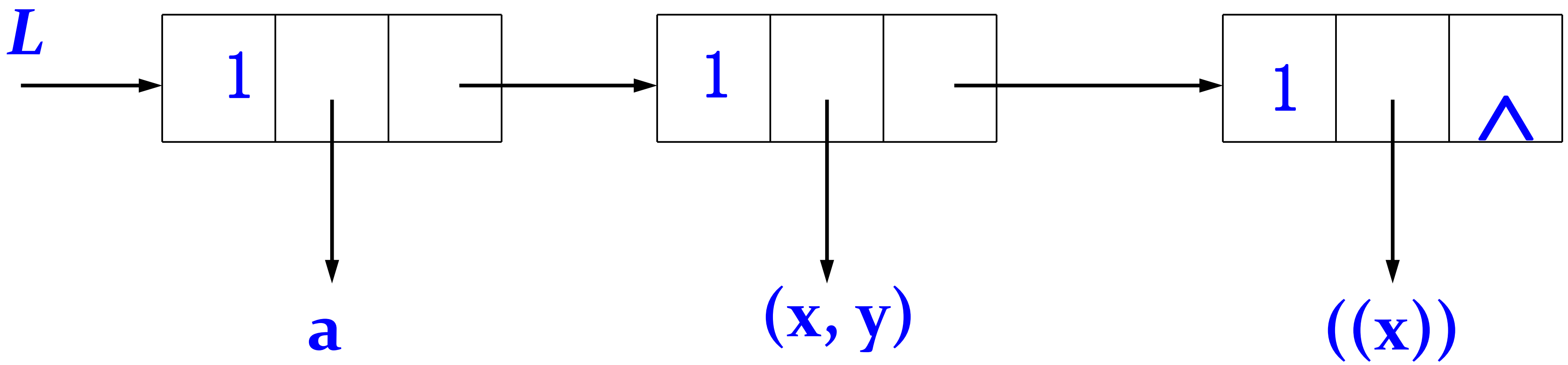
- 子表分析法

tag=1	hp	tp

tag=0	atom

tag=0	atom	tp

例如: L=(a, (x,y), ((x)))



可以看成是线性表

5.5 广义表的存储结构

- 扩展头尾指针结点结构

```
typedef enum{ATOM,LIST} ElemTag;
typedef struct GLNode{
    ElemTag tag;
    union{
    AtomType atom; //原子结点
    struct GLNode *hp; //定义子表的头指针
    };
    struct GLNode *tp;//相当于线性链表中的next，指向下一个元素的节点;
}
```

表结点：

tag=1	hp	tp

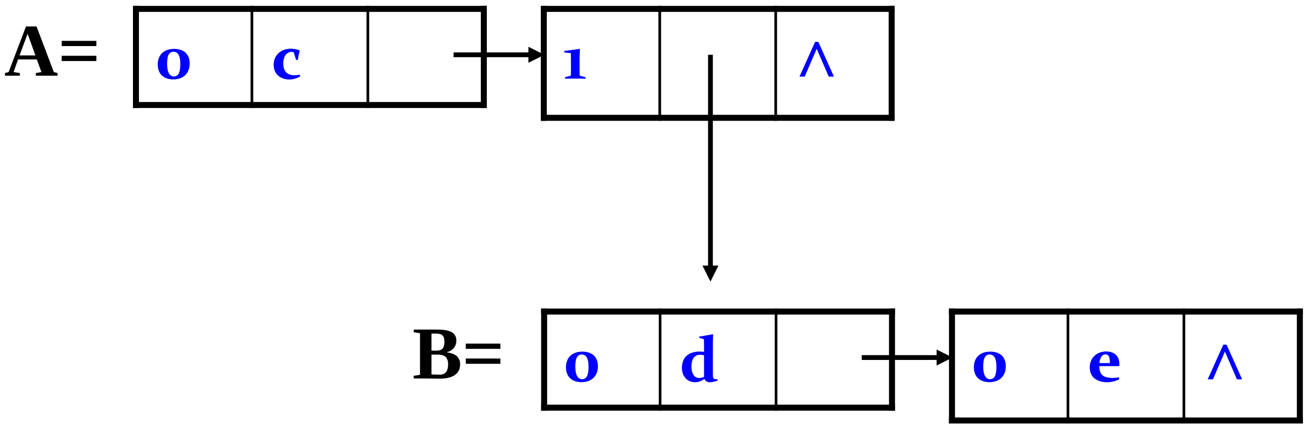
原子结点：

tag=0	atom	tp

5.5 广义表的存储结构

- 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



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；

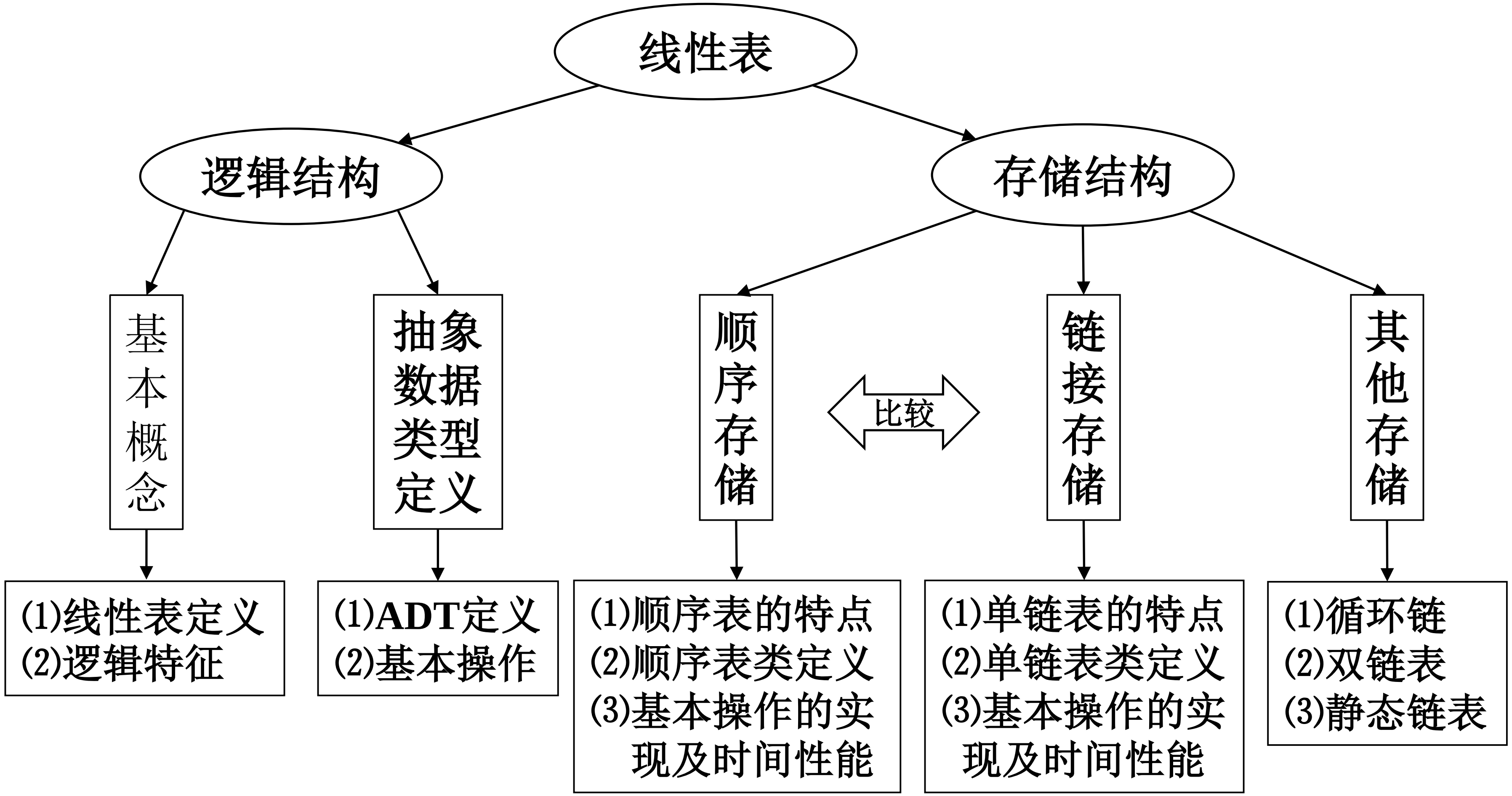
(3)特殊矩阵压缩存储时的下标变换公式；

(4)稀疏矩阵的压缩存储方法；

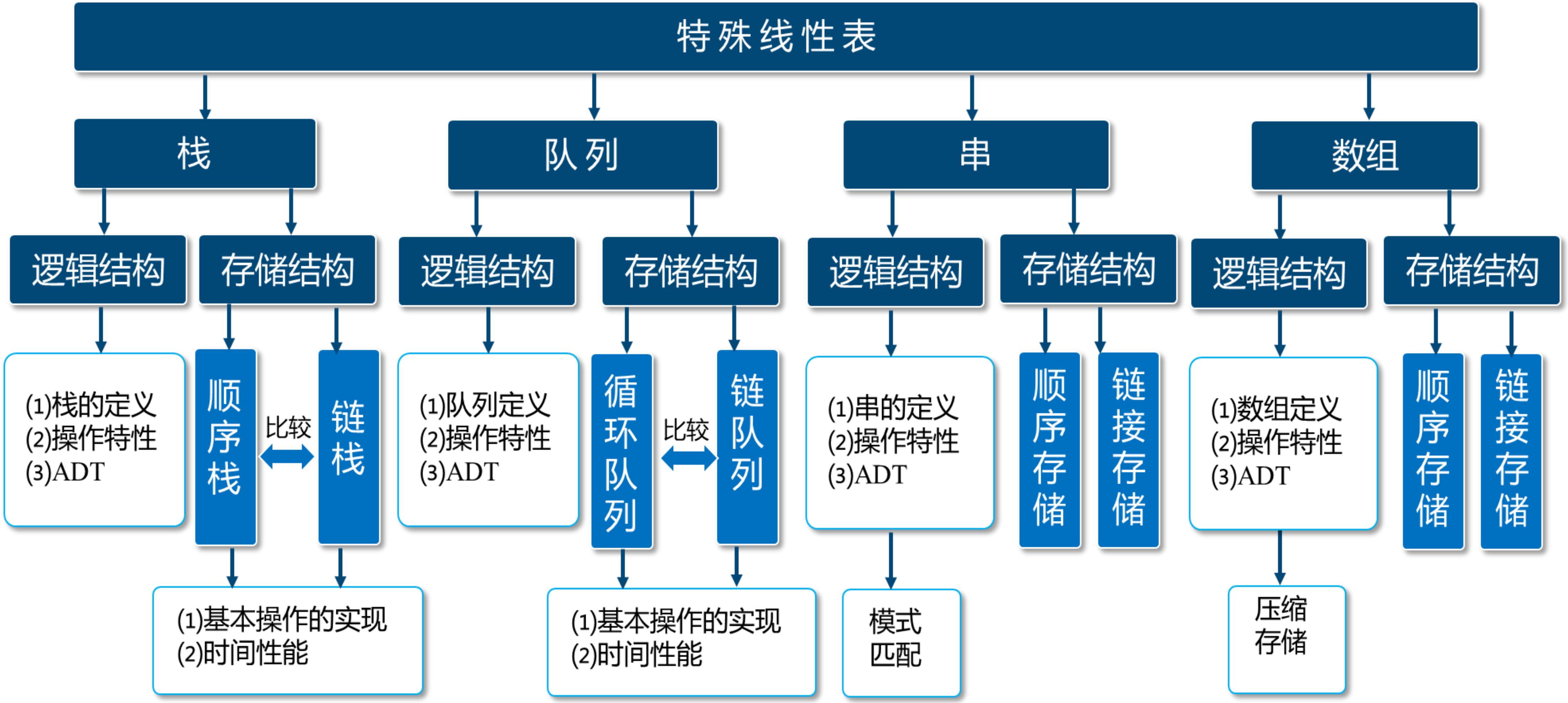
(5)三元组表示稀疏矩阵时进行矩阵运算采用的算法。

(6)广义表的定义、存储和性质。

知识点总结


线性表
逻辑结构
存储结构
基本概念
抽象数据类型定义
顺序存储
比较
链接存储
其他存储
(1)线性表定义
(2)逻辑特征
(1)ADT定义
(2)基本操作
(1)顺序表的特点
(2)顺序表类定义
(3)基本操作的实现及时间性能
(1)单链表的特点
(2)单链表类定义
(3)基本操作的实现及时间性能
(1)循环链
(2)双链表
(3)静态链表

线性表小结


特殊线性表
栈
队列
串
数组
逻辑结构
存储结构
(1)栈的定义
(2)操作特性
(3)ADT
顺序栈
比较
链栈
(1)基本操作的实现
(2)时间性能
(1)队列定义
(2)操作特性
(3)ADT
循环队列
比较
链队列
(1)基本操作的实现
(2)时间性能
(1)串的定义
(2)操作特性
(3)ADT
顺序存储
链接存储
模式匹配
(1)数组定义
(2)操作特性
(3)ADT
顺序存储
链接存储
压缩存储