第五章　数组

数组【学习目标】

1. 理解数组类型的特点及其在高级编程语言中的存储表示方法，并掌握数组在“以行为主”的存储表示中的地址计算方法。
2. 掌握特殊矩阵的存储压缩表示方法。
3. 掌握以三元组表示稀疏矩阵时进行矩阵运算所采用的处理方法。

第五章　数组

5.1　数组的定义和运算

5.2　数组的顺序存储结构

5.3　特殊矩阵的压缩存储

5.1 数组的定义和运算

数组是所有高级编程语言中都已实现的固有数据类型，因此学习过高级程序设计语言的读者对数组都不陌生，凡是遇到需要定义一大批同类型的数据时，都要定义一个数组来存放及表示这些数据。它和其它诸如整数、实数等原子类型不同，它是一种结构类型。换句话说，“数组”是一种数据结构。

5.1 数组的定义和运算

数组的准确定义：

- 数组是由值与下标构成的有序对，结构中的每一个数据元素都与其下标有关。
- 数组结构的性质：

(1) 数据元素数目固定：一旦声明了一个数组结构，其

元素数目不再有增减变化；

(2) 数据元素具有相同的类型；

(3) 数据元素的下标关系具有上下界的约束并且下标有序。

(4) 一维数组是线性结构。

数组的运算

数组一旦被定义，其维数（N）和每一维的上、下界均不能再变，数组中元素之间的关系也不再改变。因此数组的基本操作除初始化之外，只有通过给定的“一组下标”索引取得元素或修改元素的值这两种操作。

- 给定一组下标，查找、读取相应的数据元素；
- 给定一组下标，修改（写入）相应数据元素中的某个数据项的值。

数组举例

一维数组　　$An=[a_1,a_2,\ldots.a_n]$

二维数组　　$Amn=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \cdots & \cdots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$

或记作

$Amn=[a_{11}\ a_{12}\ \cdots\ a_{1n}],[a_{21}\ a_{22}\ \cdots\ a_{2n}],\cdots,[a_{m1}\ a_{m2}\ \cdots\ a_{mn}]$

或记作

$Amn=[a_{11}\ a_{21}\ \cdots\ a_{m1}],[a_{12}\ a_{22}\ \cdots\ a_{m2}],\cdots,[a_{1n}\ a_{2n}\ \cdots\ a_{mn}]$

由此可见，任何一个 n 维数组均可由 **n-1** 维数组来定义，当维数大于 1 时可以**看作是一维数组的推广**。

数组的逻辑结构

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \cdots & \cdots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

以二维数组 **Amn** 为例：

D={ $a_{i,j}$ **|1≤i≤m,1≤j≤n}**

R={ROW} 或 **{COL}**

其中：

COL ={$\langle a_{i-1,j}, a_{i,j}$ **>| i=2,…,m, 1≤j≤n}** （称作“列关系”）

ROW ={$\langle a_{i,j-1}, a_{i,j}$ **>| j=2,…,n, 1≤i≤m}** （称作“行关系”）

上述定义的二维数组中共有 **m ×n** 个元素，每个元素 $a_{i,j}$ 同时处于“行”和“列”的两个关系中，它既在“列关系 **COL**”中是 $a_{i,j}$ **(i>1)** 的后继，又在“行关系 **ROW**”中是 $a_{i,j}$ **(j>1)** 的后继。和线性表类似，数组中的每个元素都对应于一组下标（ ）。需要注意的是， C 语言的限定：各维下标的下限均约定为常量 **0** 。

数组的存储：

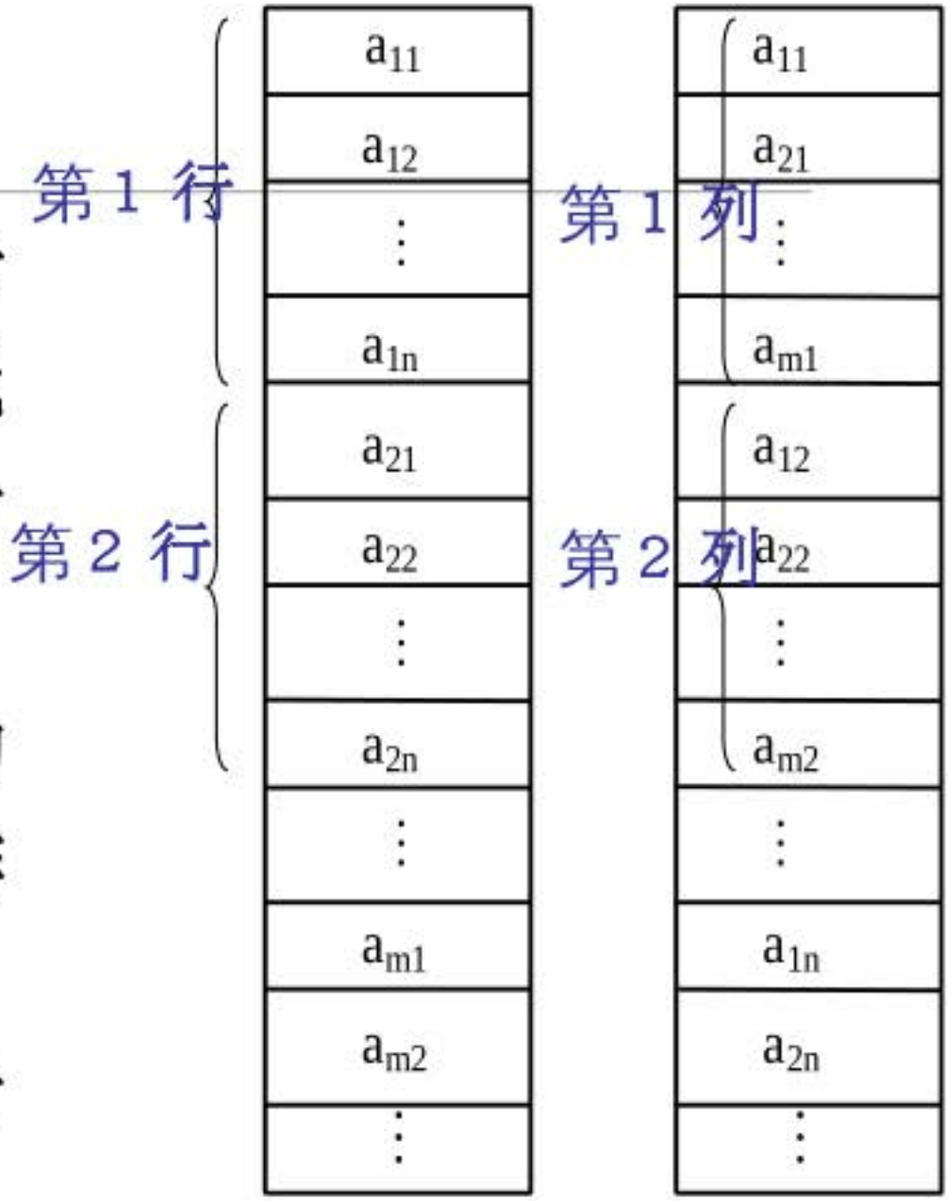
数组类型不作插入和删除的操作，因此只需要通过“顺序映象”得到它的存储结构，即借助数据元素在存储器中的相对位置来表示数据元素之间的逻辑关系。

由于数组中的数据元素之间是一个“多维”的关系，而存储地址是一维的，**因此高维数组的顺序存储表示要解决的是一个**“如何用一维的存储地址来表示多维的关系”的问题。

5.2 数组的顺序存储结构

二维数组的顺序存储分为：

- 以行为主序的优先存储：**将数组元素按行优先关系排列，第 i+1 行的数据元素紧跟在第 i 行数据元素的后面。同一行中元素以列下标次序排列。**
- 以列为主序的优先存储：**列为主序的优先存储是将数组元素按列优先关系排列，第 j+1 列的数据元素紧跟在第 j 列数据元素的后面，同一列中元素以行下标次序排列。**



(a) 以行为主序　(b) 以列为主序

现有高级语言中的大多数都是按＂以行为主＂实现数组类型的。

2. 数组的顺序存储结构

二维数组 Amn 的存储

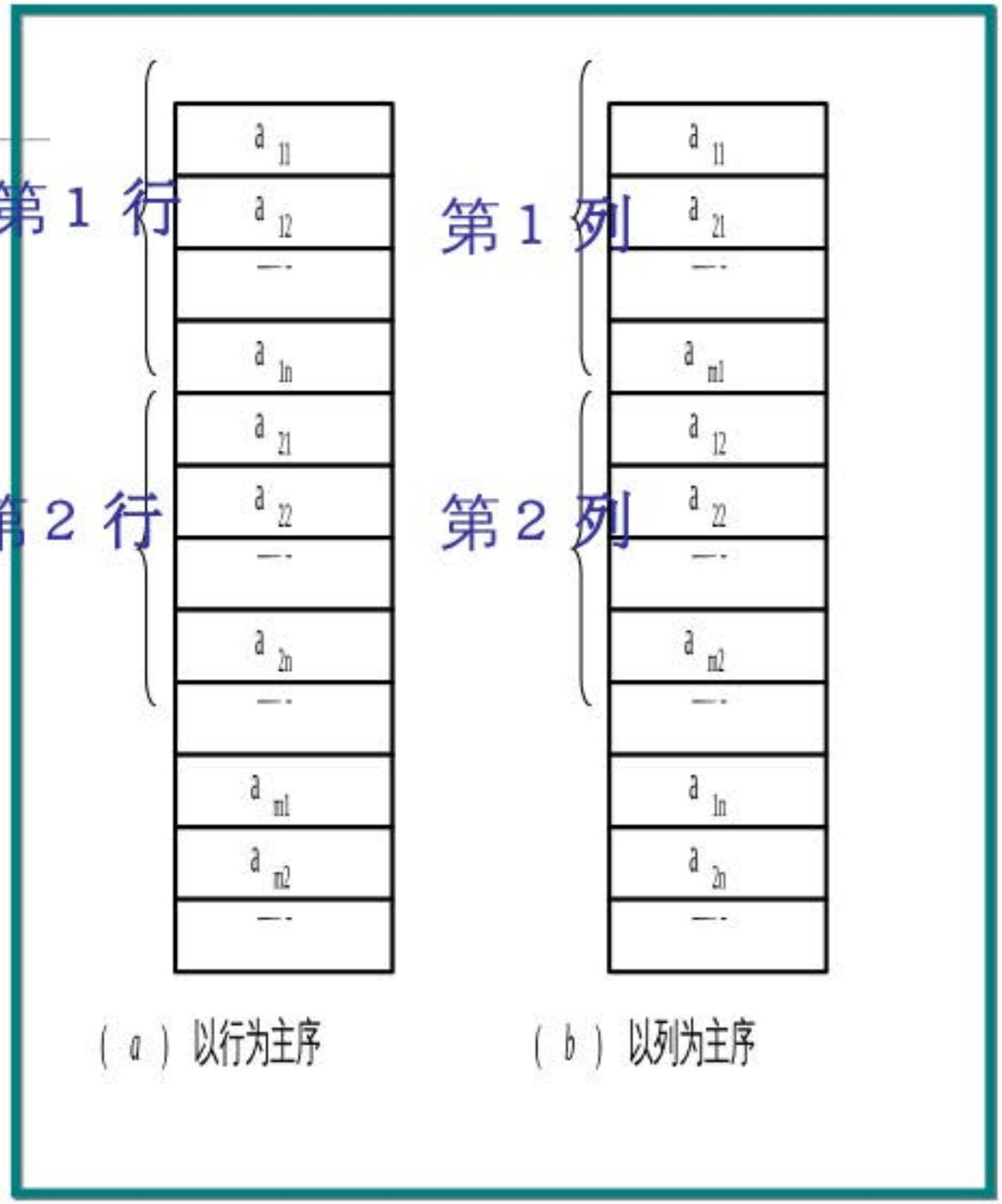
- 以行为主序的优先

$$Loc(a_{ij})=Loc(a_{11})+[(i-1)*n+(j-1)]*d$$

- 以列为主序的优先

$$Loc(a_{ij})=Loc(a_{11})+[(j-1)*m+(i-1)]*d$$

其中 **$Loc(a_{11})$** 是二维数组中第一个数据元素的存储地址，称为数组的 "基地址" 或 "基址"。



假如数组下标从非 0 、 1 开始，如二维数组 A 的行下标范围为

$c_1..d_1$，列下标范围为 $c_2..d_2$；

共计 d_1-c_1+1 个行，每行具有 d_2-c_2+1 个元素，即 d_2-c_2+1 个列向量。

则该二维数组以行优先存储的地址计算公式为：

- $Loc(a_{ij})=Loc(a_{c1c2})+[(i-c_1)*(d_2-c_2+1)+(j-c_2)]*d$

5.3 特殊矩阵的压缩存储

矩阵是

由 m 行和

在用高级语言

它使矩阵中的

储位置。然而

列特性的高阶

元素。为了节

，即不存零元

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

5.3 特殊矩阵的压缩存储

5.3.1 特殊矩阵

对于值相同的元素或者零元素在矩阵中的分布具有一定规律的矩阵，称之为特殊矩阵。

大致有这样三类特殊矩阵：

对角矩阵

上（下）三角矩阵

对称矩阵。

对角矩阵

所有的非零元素都集中在以主对角线为中心的（由 k 条对角线组成的）带状区域中，即除了主对角线上和主对角线邻近的上、下方以外，其余元素均为零，称为 K 对角阵。

$$\begin{bmatrix} a_{11} & a_{12} & & & & \\ a_{21} & a_{22} & a_{23} & & & \\ & \ddots & \ddots & \ddots & & \\ & & a_{n-1n-2} & a_{n-1n-1} & a_{n-1n} & \\ & & & a_{nn-1} & a_{nn} \end{bmatrix}$$

（b）三对角矩阵

只在主对角线上含有非零元素的对角矩阵是最简单最常用的对角矩阵。

$$\begin{bmatrix} a_{11} & & & 0 \\ & & & \\ & & \ddots & \\ 0 & & & a_{nn} \end{bmatrix}$$

n 个非零元素用一维数组 A[0,...,n-1] 存储；

$A[k]=a_{ii}$ 的对应关系是 k=i-1 。

三对角矩阵

当 |i-j|> 1 时 , a_{ij} = 0, (1<=i, j<=n)

$$An,n = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & \dots & 0 \\ a_{21} & a_{22} & a_{23} & 0 & \dots & 0 \\ 0 & a_{32} & a_{33} & a_{34} & \dots & 0 \\ & \dots\dots & & & & \\ 0 & 0 & 0 & \dots & a_{nn-1} & a_{nn} \end{pmatrix}$$

共计（3n-2）个非零元素

一维数组 SA[0,…,3*n-3] 作为数组 A 下三角元素的存储结构：

SA[k]=[a_{11}, a_{12}, a_{21}, a_{22}, a_{23}, a_{32}, a_{33}, a_{34},…, a_{nn-1}, a_{nn}]

k = 0 1 2 3 4 5 6 7 3n-2 3n-3

SA[k]和A[i，j]一一对应关系：

K 从 0 开始

$$A[i,j]=\begin{cases} SA[k] & \text{当 } |i-j|<=1 \\ 0 & \text{当 } |i-j|>1 \end{cases}$$

$$k = (i-1)*3-1+(j-i+1) = 2i+j-3 \quad \text{当 } |i-j|<=1$$

备注：如无特别指出，均认为一维数组从 0 下标开始存放元素，即 k 从 0 开始，而矩阵的行列下标则从 1 开始。

下三角矩阵中，一维数组 **A[k]** 与 $a_{i,j}$ 的对应关系是：**1=<i,j<=n**

行优先存储

$$k=\begin{cases}\dfrac{i*(i-1)}{2}+j-1 & i\geq j\\ \dfrac{n*(n+1)}{2} & i<j\end{cases}$$

K=0,1,2,3…..

列优先存储

$$k=\begin{cases}\dfrac{[n+(n-j+2)]*(j-1)}{2}+i-j & i\geq j\\ \dfrac{n*(n+1)}{2} & i<j\end{cases}$$

K=0,1,2,3…..

上三角矩阵中，一维数组 **A[k]** 与 $a_{i,j}$ 的对应关系是：

1=<i,j<=n

列优先存储

$$k=\begin{cases}\dfrac{j*(j-1)}{2}+i-1 & i\le j\\ \dfrac{n*(n+1)}{2} & i>j\end{cases}$$

K=0,1,2,3…..

行优先存储

$$k=\begin{cases}\dfrac{[n+(n-i+2)]*(i-1)}{2}+j-i & i\le j\\ \dfrac{n*(n+1)}{2} & i>j\end{cases}$$

K=0,1,2,3…..

例：下三角矩阵

$$A = \begin{pmatrix} \mathbf{4} & 0 & 0 & 0 & 0 \\ 5 & \mathbf{2} & 0 & 0 & 0 \\ 3 & 1 & \mathbf{3} & 0 & 0 \\ 2 & 5 & 2 & \mathbf{8} & 0 \\ 1 & 6 & 7 & 9 & \mathbf{5} \end{pmatrix}$$

$$k = \begin{cases} \dfrac{i*(i-1)}{2} + j-1 & i \geq j \\ \dfrac{n*(n+1)}{2} & i < j \end{cases}$$

如：**A[5,3] = 7**；

行优先存储时：**k = 5(5-1)/2 + 3-1 = 12**

故：**SA[12] = 7.**

对称矩阵

n 阶方阵 A 中元素满足 $a_{ij}=a_{ji}$ $(1\le i, j\le n)$，则称为对称阵。

存储元素总个数：$1+2+...+n = n(n+1)/2$ 。

若为对称矩阵中的对称的元素分配一个存储空间，则可将矩阵中n^2 个元素压缩存储到 $n(n+1)/2$ 个存储空间中。

假设以一维数组 A[n(n+1)/2] 压缩存储 n 阶对称方阵 A ，则一维数组中的数据元素和方阵中的元素 $a_{i,j}$ 之间存在着

下列一一对应的关系：

对称方阵中元素 $a_{i,j}$ 与一维数组 A[k] 的对应关系

n 阶对称方阵 A 中元素满足 $a_{ij}=a_{ji}$ (1≤i, j≤n)

假定按行优先关系存储，可得到 A[k] 与 $a_{i,j}$ 的对应关系如下：

i>=j 时 (下三角阵元素) :　k=i*(i-1)/2+j-1,　0=<k<=n(n+1)/2-1

因为 $a_{i,j}=a_{j,i}$ ，所以：

i<j 时（上三角阵中元素）：　k=j*(j-1)/2+i-1, 0=<k<=n(n+1)/2-1

统一起来:

k=i*(i-1)/2+j-1 ，其中 i=max(i,j), j =min(i,j)

5.3.2 稀疏矩阵

由于特殊矩阵中的非 0 元素在矩阵中的分布有着一定的规律，则可将这些非 0 元素连续存储在一个一维数组中。即矩阵中的任何一个元素 (下标 i,j) 和它们在一维数组中的位置（ 下标 k ）之间存在着某种“转换关系”，并且这种转换关系可以用数学公式表示。换言之，对于非 0 元素分布有规律的矩阵，我们总可以找到矩阵中的元素与一维数组下标之间的对应关系。

还有一类矩阵，其中也含有较多的 0 元素及少数的非 0 元素，但非 0 元素的分布没有规律， 这类矩阵称为稀疏矩阵。

如何存储这类稀疏矩阵中的非零元素?

首先应该分析一下，如果将稀疏矩阵中的所有元素都进行存储，那么它的弊病是什么?

一、浪费空间，二、浪费时间。

前者是指存放了大量没有用的零值元素，后者是指在进行矩阵

的运算中进行了很多与零元素的运算。

5.3.2 稀疏矩阵

含有非零元素及较多的零元素，但非零元素的分布没有任何规律，这样的矩阵称为稀疏矩阵。即稀疏矩阵 $A_{m\times n}$ 中有 t 个非零元素，若 t/m*n≤0.05，则称 A 为稀疏矩阵。

稀疏矩阵压缩存储方法：

- **三元组表**
- **十字链表**

三元组表

[定义]

将表示稀疏矩阵的非零元素的三元组按行优先（或列优先）的顺序排列（跳过零元素），则得到一个其结点均是三元组的线性表。将采用顺序存储结构的线性表的称为三元组顺序表，简称三元组表。

三元组：

三元组 = （行 i，列 j，非零元素值）——行优先

三元组 = （列 j，行 i，非零元素值）——列优先

例如表示矩阵

$$M=\begin{bmatrix}0&0&9&0&-7&0\\0&0&0&0&0&0\\0&0&0&8&0&0\\5&0&0&0&0&2\\0&0&0&0&16&0\end{bmatrix}$$

三元组线性表为：((1,3,9),(1,5,-7),(3,4,8),(4,1,5),(4,6,2),(5,5,16))

三元组表的结构类型定义：

```
#define smax 256
typedef int datatype;
typedef struct
{  int i, j;
    datatype v;
} node;
 typedef struct
{  int m, n, t;
    node data[smax];
} spmatrix;
spmatrix * a, *b;
```

```
/*  smax 为最大非零元素个数
   */

/*  行号，列号  */
/*  元素值  */
/* 三元组结点 */

/* 行数，  列数，  非零元素个数
   */
/*  三元组表  */
/*  稀疏矩阵类型  */
```

稀疏矩阵的三元组表示实例

$$A_{4\times5}=\begin{bmatrix}0 & 5 & 0 & 0 & 8\\ 1 & 0 & 3 & 0 & 0\\ 0 & -2 & 0 & 0 & 0\\ 6 & 0 & 0 & 0 & 0\end{bmatrix}$$

稀疏矩阵 **A**

spmatrix * a;

下标	i	j	v
0	1	2	5
1	1	5	8
⋮	2	1	1
	2	3	3
⋮	3	2	-2
a→ t-1	4	1	6
⋮			
smax-1			

行优先存储

稀疏矩阵 A 的三元组表

显然，在三元组顺序表中容易从给定的行列号（i,j）找到对应的矩阵元素：首先按行号 i 在顺序表中进行有序搜索，找到相同的 i 后再按列号进行有序搜索，若在三元组顺序表中找到行号和列号都和给定值相同的元素，则其非零元值即为所求，否则为矩阵中的 0 元素。

查找同行中的后继：同一行的下一个非零元即为顺序表中的后继。通常三元组表按行号为主序存储，因此，同一行号的下一个相邻结点即为后继；

查找同列中的后继：三元组顺序表以行号为主序有序，因此从上到下依次搜索，遇到的下一个列号相同的元素即为同一列的后继（下一个非 0 元）。

稀疏矩阵三元组表示的转置运算

$$A_{4\times5}=\begin{bmatrix}0 & 5 & 0 & 0 & 8\\ 1 & 0 & 3 & 0 & 0\\ 0 & -2 & 0 & 0 & 0\\ 6 & 0 & 0 & 0 & 0\end{bmatrix}$$

转置为

$$B_{5\times4}=\begin{bmatrix}0 & 1 & 0 & 6\\ 5 & 0 & -2 & 0\\ 0 & 3 & 0 & 0\\ 0 & 0 & 0 & 0\\ 8 & 0 & 0 & 0\end{bmatrix}$$

行优先存储

	i	j	v
0	1	2	5
1	1	5	8
⋮	2	1	1
	2	3	3
⋮	3	2	-2
a→ t-1	4	1	6
⋮			
smax-1			

如何得到？

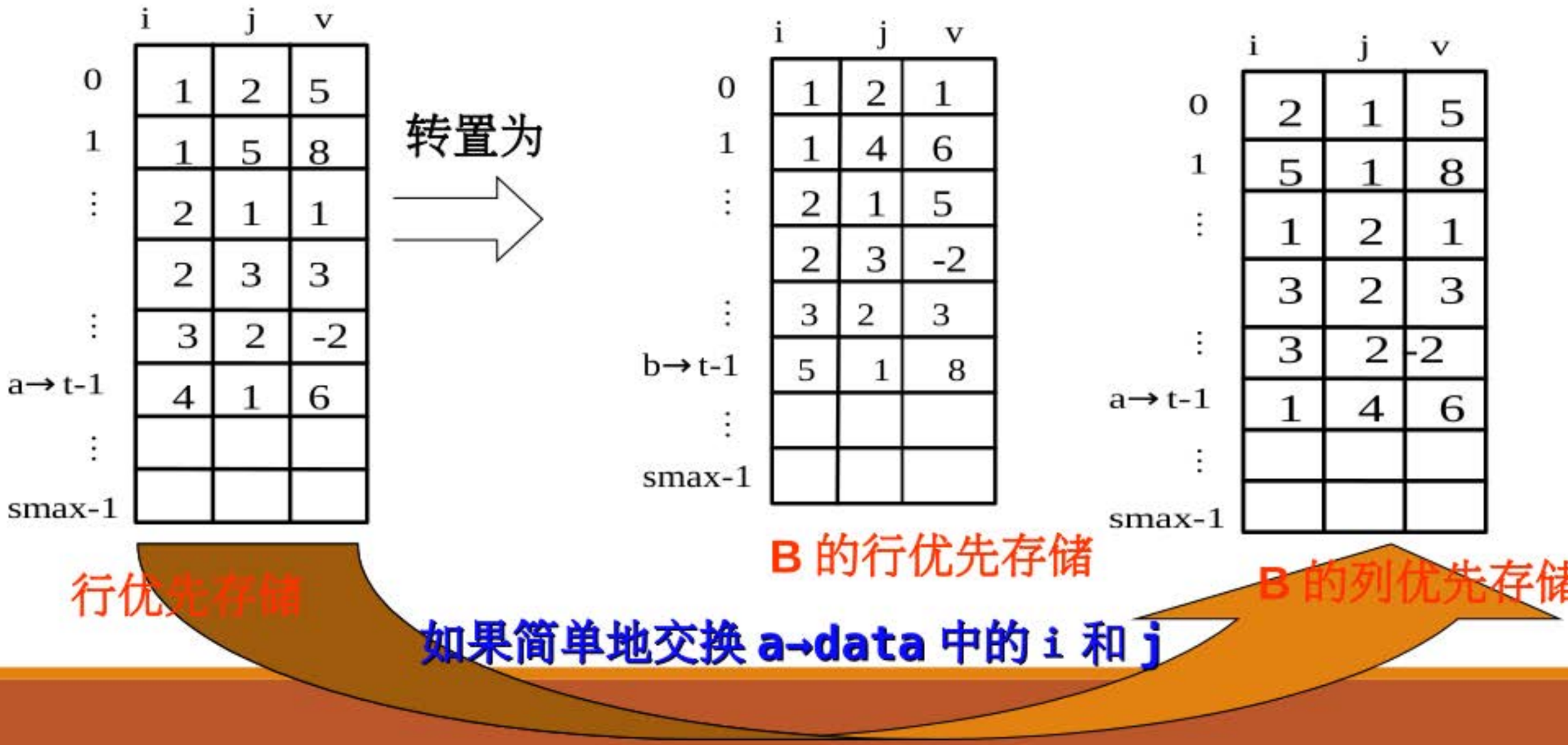
行优先存储

	i	j	v
0	1	2	1
1	1	4	6
⋮	2	1	5
	2	3	-2
⋮	3	2	3
b→ t-1	5	1	8
⋮			
smax-1			

三元组表 --- 转置运算

实现转置运算时需注意：

如果简单地交换 a→data 中 i 和 j 中的内容，那么得到的 b→data 是一个按列优先顺序存储的稀疏矩阵 B 。



ano	i	j	v
0	1	2	5
1	1	5	8
⋮	2	1	1
	2	3	3
⋮	3	2	-2
a→t-1	4	1	6
⋮			
smax-1			

- 方法：按照 a->data[] 的列序，依次在 a->data[] 中寻找相应的三元组并交换行列号填入 b 中，所得到的 b->data[] 是按行优先存放的，因此进行了转置。

bno	i	j	v
0	1	2	1
1	1	4	6
⋮	2	1	5
	2	3	-2
⋮	3	2	3
b→t-1	5	1	8
⋮			
smax-1			

- 步骤：从 a 矩阵下标 ano=0 开始扫描，遇到列号为 col(循环变量) 的项，将其行号变列号、列号变行号，顺次存于转置矩阵三元组表 b 中。

稀疏矩阵三元组表示的转置的算法

```
void transmat(spmatrix *a , spmatrix*b)
{
    int ano, bno, col;
    b→m=a→n;
    b→n=a→m;
    b→t=a→t;
    if (b→t>0)
      { bno=0;
        for (col=1;col<=a→n; col++)
        for (ano=0;ano<a→t; ano++)
        if (a→data[ano].j == col)

          { b→data[bno].i = a→data[ano].j;

            b→data[bno].j = a→data[ano].i;

}//O(t*n)
            b→data[bno].v = a→data[ano].v;
           bno++;
          }
      }
```

```
/*ano bno 表示 a 和 b 的数组下标变
    量 */
/* col 指示 *a 的列号 ( 即 *b 的行号 )
    */
/* *b 存放转置后的矩阵   */


/* 行列数交换 */
/*  有非零元素，则转置   */

/*  按 *a 的列序转置   */
/*  扫描整个三元组表   */
/*  列号为 col 则进行置换   */


/* b→data 结点序号加 1 */
/* if (b→t>0) */
/*  返回转置结果指针   */
```

算法复杂度分析：

- 矩阵 **A** 具有 **n** 列，对 **a->data** 按列扫描 **n** 趟， **a->data[]** 的长度为 **t** ，所以算法的时间复杂度为 **O(t*n),** 如果不采用压缩存储方式则时间复杂度为 **O （ m*n ）** 。

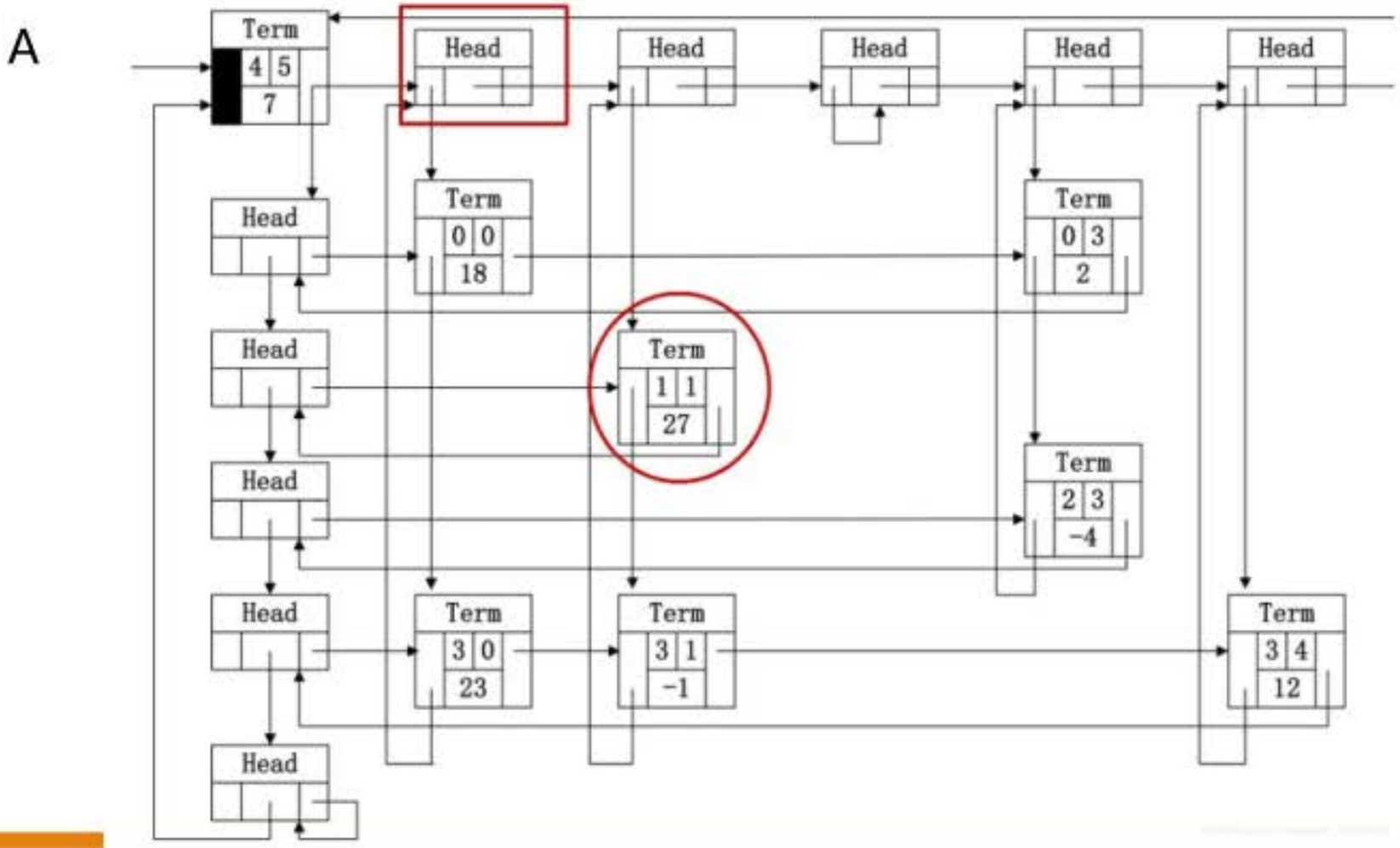
稀疏矩阵的十字链表存储方法:

稀疏矩阵的链接表示采用十字链表：行链表与列链表十字交叉。

$$A=\begin{bmatrix}18 & 0 & 0 & 2 & 0\\ 0 & 27 & 0 & 0 & 0\\ 0 & 0 & 0 & -4 & 0\\ 23 & -1 & 0 & 0 & 12\end{bmatrix}$$

行链表与列链表都是带（头结点）的（循环）链表。

❖ 矩阵A的多重链表图



Tag		
Down	*URegion*	Right

(a) 结点的总体结构

Term			
Down	Row	Col	Right
	Value		

(b) 矩阵非0元素结点

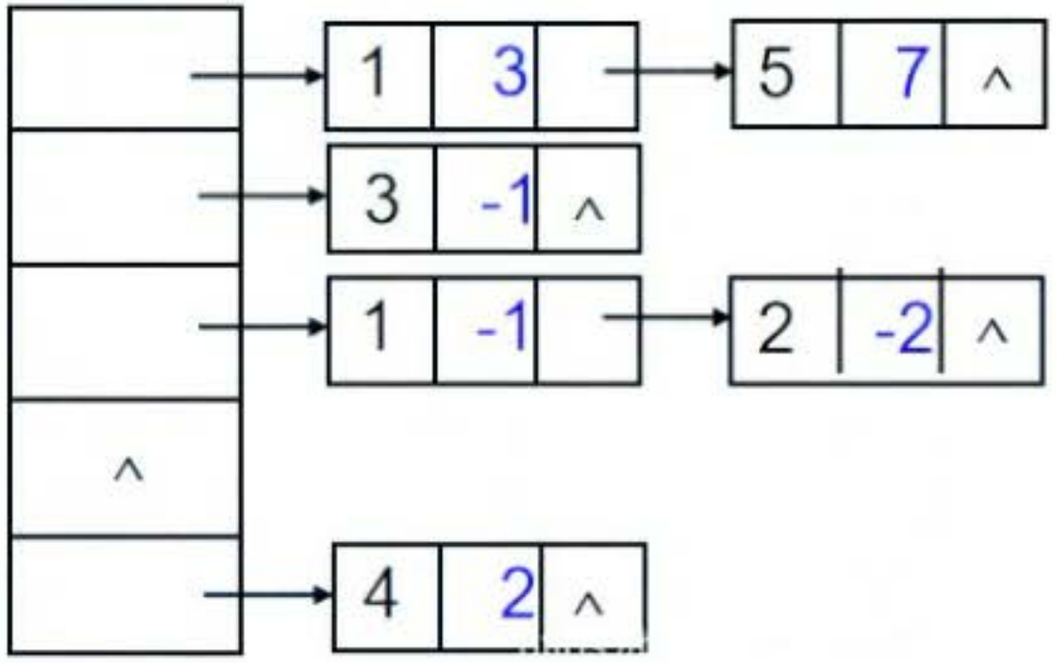
Head		
Down	Next	Right

(c) 头结点

稀疏矩阵的邻接表表示方法：

稀疏矩阵也可以表示成如右下图所示的数组和链表相结合的方式存储，这有点儿类似图的邻接链表表示方法

$$A=\begin{bmatrix} 3 & 0 & 0 & 0 & 7 \\ 0 & 0 & -1 & 0 & 0 \\ -1 & -2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$



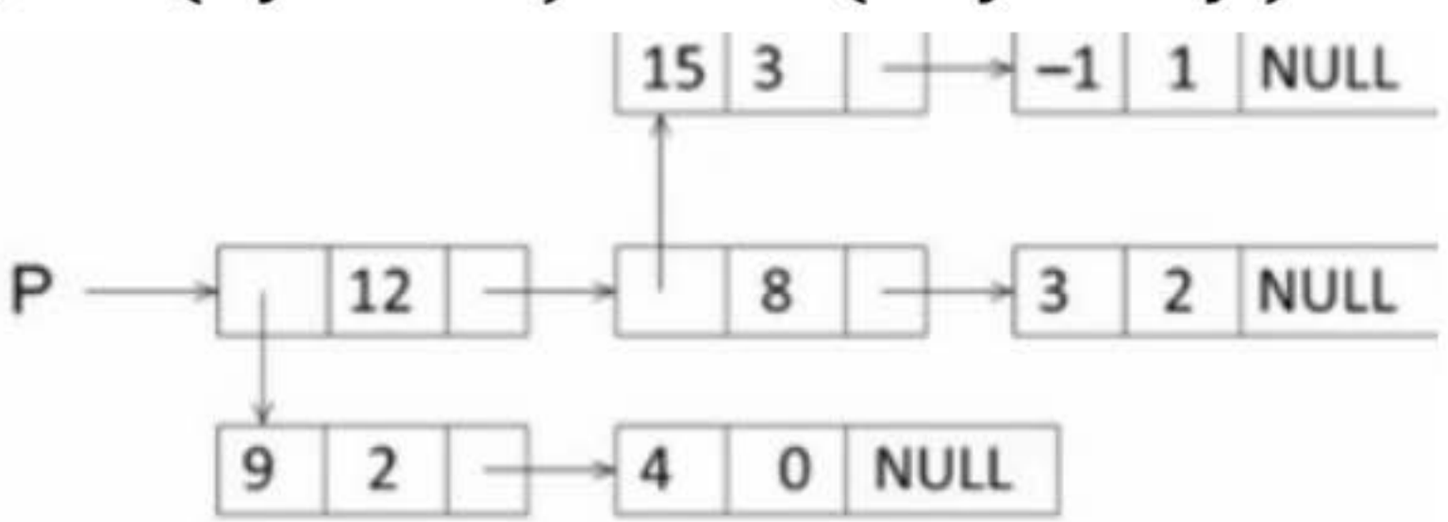
（了解）广义表

例如：前面我们学习了一元多项式的表示（循环链表或者链表），那么二元多项式如何表示呢？如下所示：

$$P(x,y)=9x^{12}y^2+4x^{12}+15x^8y^3-x^8y+3x^2$$

思路：看成一元多项式：ax^n1+bx^n2+cx^n3 的形式，其中的系数不是常数，是一元多项式。

$$P(x,y)=(9y^2+4)x^{12}+(15y^3-y)x^8+3x^2$$



多重链表：链表中的结点可能同时属于多个链，即链表中的结点的指针域会有多个。比如广义表是多重链表。但双向链表不是多重链表。

广义表的知识点

广义表一般记作： $LS=(a_1,a_2,\ldots,a_n)$ 。 LS 是广义表的名称，

在线性表中， $a_i(1\le i\le n)$ 只限于是单元素。

在广义表的定义中， a_i 可以是单元素，也可以是广义表，分别称为广义表 LS 的原子和子表。

小写字母

大写字母

表头（ Head ）：当广义表 LS 非空时，称第一个元素 a_1 为表头；

表尾（Tail）：其余元素组成的表 $(a_2,a_3,\ldots,a_n)$ 称为表尾。

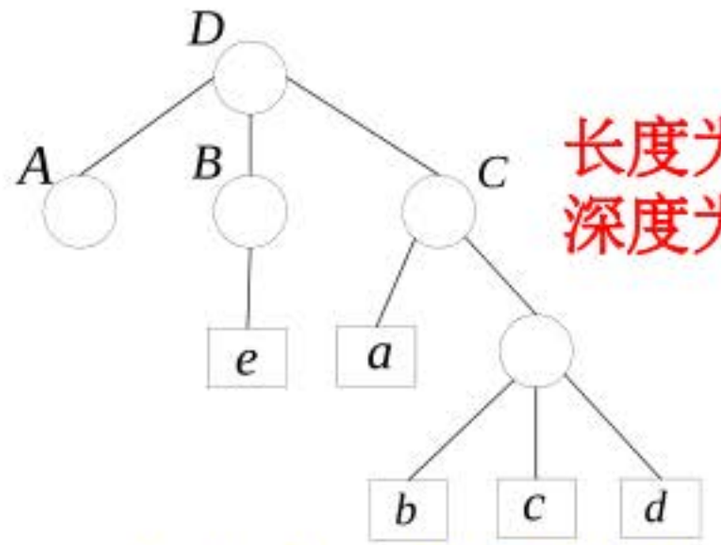
长度：所含元素的个数 n

深度：表展开后所含括号的最大层数（直接数的话，找到嵌套层次最多的元素，深度 = 这个元素的左括号个数 +1 ）

因此，任何一个非空广义表的表头可能是单元素也可能是广义表，但其表尾一定是广义表。

广义表举例:

- **A=( )**——A 是一个空表，它的长度为 0 。
- **B=(e)**—— 广义表 B 只有一个原子 e ，B 的长度为 1 。
- **C=(a,(b,c,d))**—— 广义表 C 的长度为 2 ，两个元素分别为原子 a 和子表 **(b,c,d)** 。
- **D=(A,B,C)**—— 广义表 D 的长度为 3 ，3 个元素都是子表。显然，将子表的值代入后，则有 **D=(( ),(e),(a,(b,c,d)))** 。
- **E=(a,E)**—— 这是一个递归的广义表，它的长度为 2 。 E 相当于一个无限的广义表 **E=(a,(a,(a,…)…))** 。
- **F=(A)**——F 是长度为 1 的广义表，其中的一个元素为子表。显然将子表带入后得到 **F=(( ))** 。



长度为 3
深度为 3

广义表 D 的图形表示

注意:

A=( ) 是一个无任何元素的空表，长度为 0 。

而 **F=(A)=(( ))** 不是空表， F 的长度为 **1** 。

广义表的常用基本操作

- 创建空的广义表 **InitGList （&L）**
- 求广义表 L 的长度 **GListLength （L）**
- 求广义表 L 的深度 **GListDepth （L）**
- 取广义表 L 的表头 **GetHead （L）**
- 取广义表 L 的表尾 **GetTail （L）**：即去掉表头元素后剩下的广义表部分
- 插入元素 e 作为广义表 L 的第一个元素 **InsertFirst_GL （&L，e）**
- 删除广义表 L 的第一个元素 **DeleteFirst_GL （&L，e）**

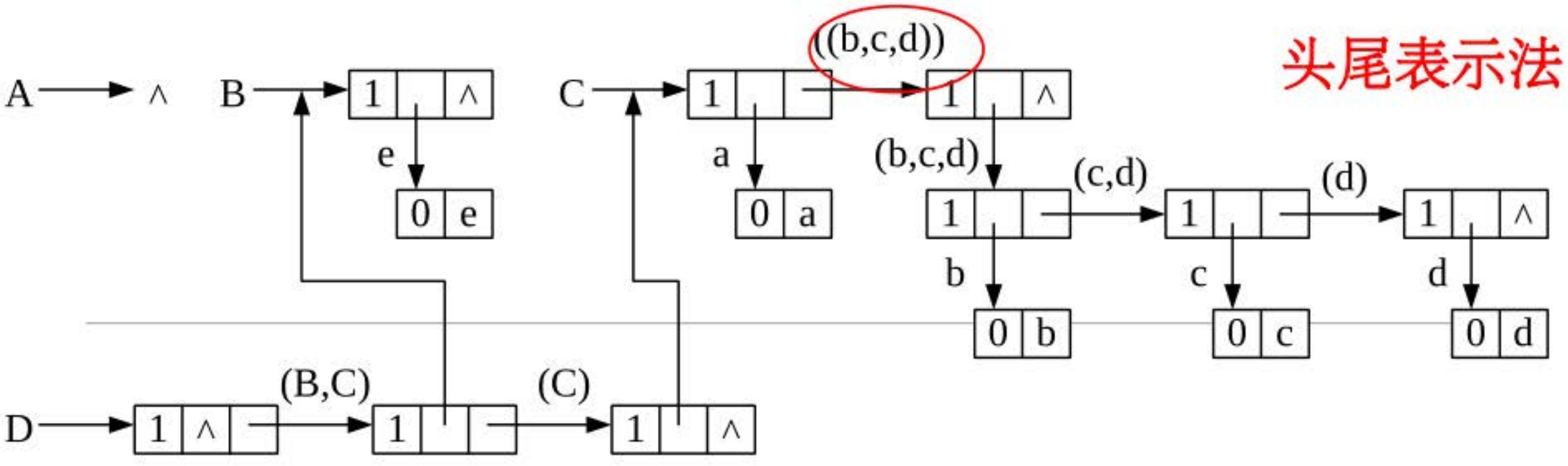
广义表的存储表示（1）：头尾表示法

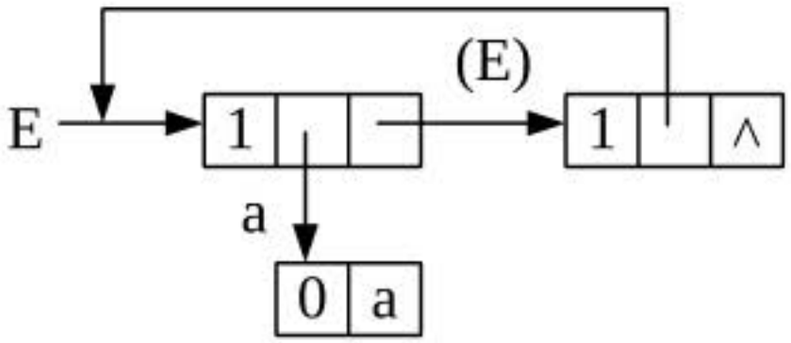
广义表头尾表示法的结点类型定义:

```
typedef struct node
{
    int tag; // 公共部分，用于区分原子结点和表结点标志域
    union{   // 原子结点和表结点共用内存
            datatype  data;   // 原子结点的数据域
            struct
            {  struct node *hp, *tp;
             }ptr; //ptr.hp 和 ptr.tp 分别指向广义表的表头和表尾
        };
}HTNode; // 广义表头尾表示法结点类型
```

(a) 表结点　　(b) 原子结点

图5.12 广义表头尾表示法的结点结构





(a) 表结点　　(b) 原子结点

图5.12 广义表头尾表示法的结点结构

图5.13 广义表头尾表示法存储结构示意图

```
D=(A,B,C)
 =(( ),(e),(a(b,c,d)))
E=(a,E)
```

广义表的存储表示（2）：孩子兄弟表示法

广义表孩子兄弟表示法的结点类型定义：

```
typedef struct node
{   int  tag; // 公共部分，用于区分原子结点和表结点
    union{     // 原子结点和表结点共用内存
            datatype  data;       // 原子结点的值域
            struct GLNode  *hp; // 表结点的表头指针
            };
   struct GLNode *tp; // 指向下一个兄弟结点的指针
}CBNode;                    // 广义表孩子兄弟表示法结点类型
```

（a） 表结点　　（b） 原子结点

图5.14 广义表孩子兄弟表示法的结点结构

孩子兄弟表示法

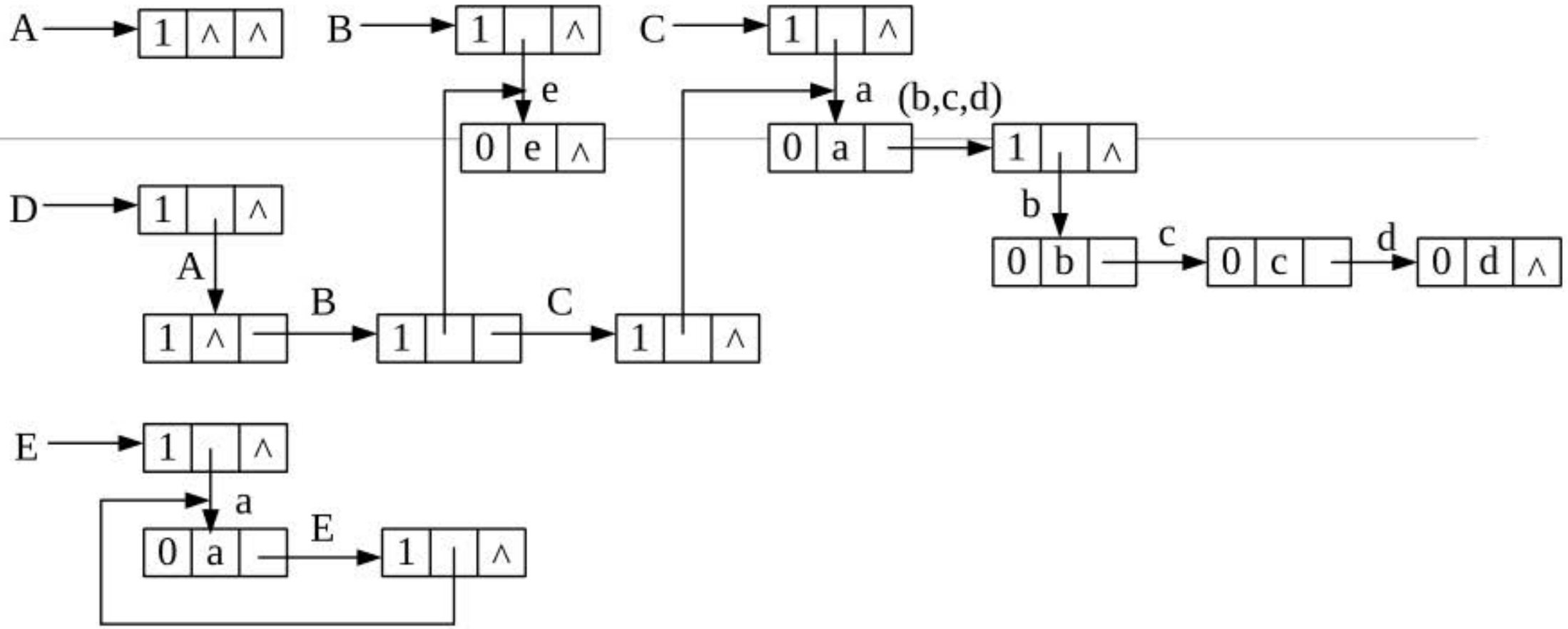
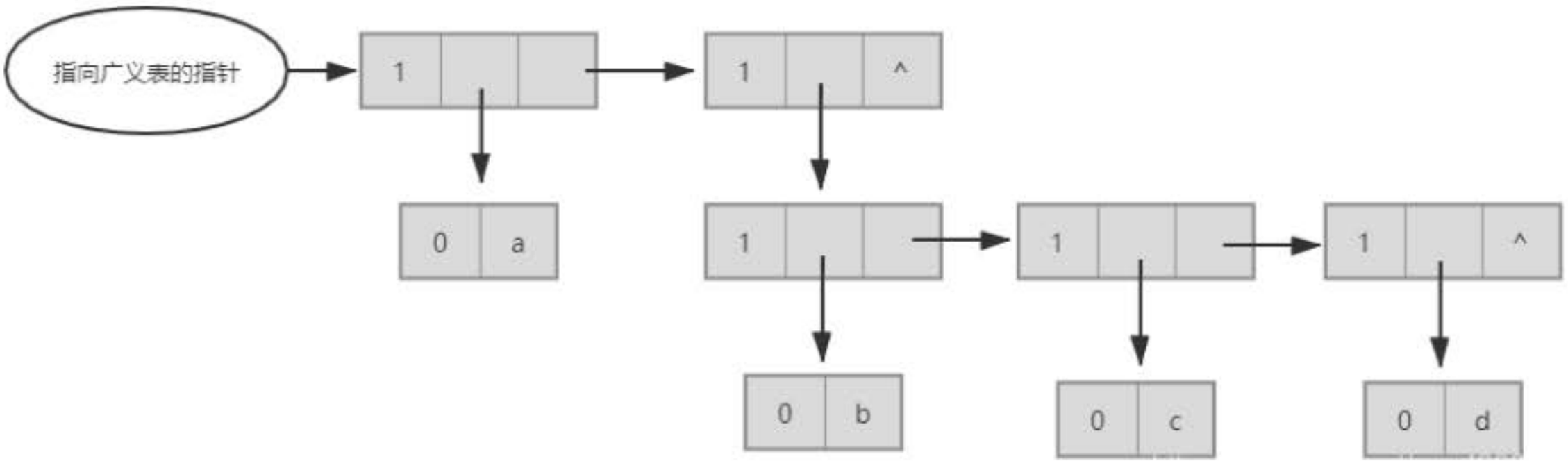


图5.15 广义表孩子兄弟表示法存储结构示意图

```
D=(A,B,C)
 =(( ),(e),(a,(b,c,d)))
E=(a,E)
```

图5.14 广义表孩子兄弟表示法的结点结构

广义表 (a,(b,c,d))



头尾表示法 还是 孩子兄弟表示法?

求广义表的表头与表尾举例:

例题 1 : 已知广义表 **L=((x,y,z),a,(u,t,w))** , 则 **head(head(tail(tail(L))))** 的结果是?

解 :(1) 用 **tail** 运算去掉表头 **(x,y,z)**, 得到

tail(L)=(a,(u,t,w))

(2) 用 **tail** 运算去掉表头 **a**, 得到

tail(tail(L)))=((u,t,w))

(3) 用 **head** 运算取出表头 **(u,t,w)**, 得到

head(tail(tail(L)))=（u,t,w）

(3) 用 **head** 运算取出表头 **u**, 得到

head(head(tail(tail(L))))=u

例题2：已知广义表LS=(a,(b,c,d))，则求出如下操作结果。

① head(LS)=? =a

② tail(LS)=? =((b,c,d))

③ head(tail(LS))=? =(b,c,d)

④ tail(tail(LS))=? =( )

⑤ head(head(tail(LS)))=? =b

⑥ tail(head(tail(LS)))=? =(c,d)

⑦ head(tail(head(tail(LS))))=? =c

⑧ tail(tail(head(tail(LS))))=? =(d)

⑨ head(tail(tail(head(tail(LS)))))=? =d

⑩ tail(tail(tail(head(tail(LS)))))=? =( )

例题3：求广义表E=(a,(b,c),E)的长度和深度?

解:

（1）由于广义表含3个元素，分别为原子a,子表(b,c),表E，递归定义。所以广义表的长度为3。

（2）由于E含有E是递归定义的，所以E是无限深度∞。