第5章 数组和广义表

5.1 数组的定义和运算

5.2 数组的顺序存储和实现

5.3 特殊矩阵的压缩存储

5.4 广义表

1. 数组的定义和运算

- 数组是n(n＞1)个相同类型数据元素$a_0,a_1,\ldots,a_{n-1}$构成的有限序列,且该有限序列存储在一块地址连续的内存单元中。
- 数组的定义类似于采用顺序存储结构的线性表，是线性表在维数上的扩张，也就是线性表中的元素又是一个线性表。
- n维数组，b_i是第i维的长度，则n维数组共有$\prod_{i=1}^{n} b_i$个数据元素，每个元素受n个关系的制约，就单个关系而言，这n个关系仍然是线性的。

- 数组具有以下性质：

(1) 数组中的数据元素数目固定。一旦定义了一个数组，其数据元素数目不再有增减变化。

(2) 数组中的数据元素具有相同的数据类型。

(3) 数组中的每个数据元素都和一组惟一的下标值对应。

(4) 数组是一种随机存储结构。可随机存取数组中的任意数据元素。

- 数组的基本操作

(1)取值 Value(A,&e, index1,..., indexn)

(2)赋值Assign(&A, e, index1,…, indexn)

2. 数组的存储结构

- 一维数组中，$LOC(a_0)$确定，每个数据元素占用L个存储单元，则任一数据元素a_i的存储地址$LOC(a_i)$就可由以下公式求出：

$$LOC(a_i)=LOC(a_0)+i*L \quad (0 \leqslant i \leqslant n-1)$$

- 二维数组，由于计算机的存储结构是线性的，如何用线性的存储结构存放二维数组元素就有一个行／列次序排放问题。

二维数组通常可以描述为两种形式 ：

以行序为主序： PASCAL、C

可以看成 $A=(\lambda_0, \lambda_1, \cdots, \lambda_{m-1})$

其中λ_i是一个行向量形式的线性表，$0 \le i \le m-1$

$$\lambda_i = (a_{i0}, a_{i1}, \cdots, a_{i\,n-1})$$

$$A_{m\times n} = \begin{bmatrix} \begin{bmatrix} a_{00} & a_{01} & a_{02} & \cdots & a_{0,n-1} \end{bmatrix} \\ \begin{bmatrix} a_{10} & a_{11} & a_{12} & \cdots & a_{1,n-1} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} a_{m-1,0} & a_{m-1,1} & a_{m-1,2} & \cdots & a_{m-1,n-1} \end{bmatrix} \end{bmatrix}$$

以列序为主序： FORTRAN

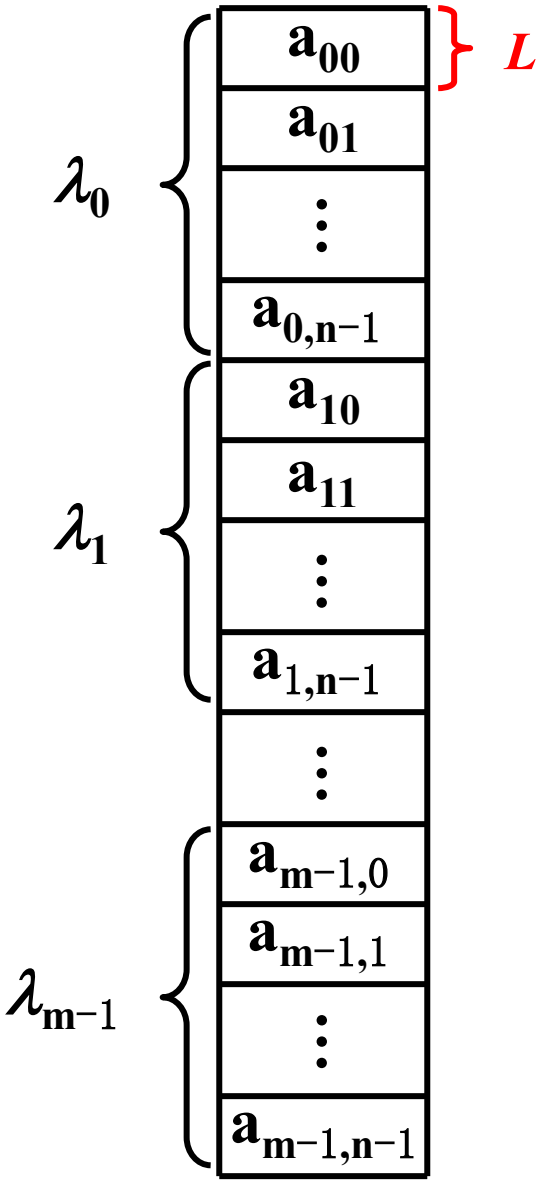
可以看成 $A=(\chi_0,\ \chi_1,\ \cdots,\ \chi_{n-1})$

其中 χ_j 是一个列向量形式的线性表，$0\le j\le n-1$

$$\chi_j=(a_{0j},\ a_{1j},\ \cdots,\ a_{m-1j})$$

$$A_{m\times n}=\begin{bmatrix} \begin{bmatrix} a_{00}\\ a_{10}\\ \vdots\\ a_{m-1,0}\end{bmatrix} & \begin{bmatrix} a_{01}\\ a_{11}\\ \vdots\\ a_{m-1,1}\end{bmatrix} & \begin{bmatrix} a_{02}\\ a_{12}\\ \vdots\\ a_{m-1,2}\end{bmatrix} & \cdots & \begin{bmatrix} a_{0,n-1}\\ a_{1,n-1}\\ \vdots\\ a_{m-1,n-1}\end{bmatrix}\end{bmatrix}$$

对于数组，一旦规定了维数和维界，如何计算数组元素的存储位置。



设数组以行序为主序。

设二维数组 A(m×n)

其数组元素 a_{ij} 的存储位置为

$$LOC(i, j) = LOC(0, 0) + (n \times i + j)\, L$$

其中，LOC(0，0)是a_{00}的存储位置；

*L*是每个数组元素占用的存储单元数；

例如，$LOC(1, 1) = LOC(0, 0) + (n \times 1 + 1)\, L$

- n维数组

每一元素对应下标$(j_1, j_2, \ldots, j_n)$，$0 \le j_i \le b_i - 1$，b_i为第i维的长度。以行序为主序。

$$LOC(j_1, j_2, \ldots, j_n) = LOC(0,0,\ldots,0) +$$

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

- 例: 对二维数组float a[5][4]计算：

(1) 数组a中的数组元素数目；

(2) 若数组a的起始地址为2000, 且每个数组元素长度为32位(即4个字节), 数组元素a[3][2]的内存地址。

元素数目共有5*4=20个。

C语言采用行序为主序的存储方式,则有：

$$LOC(a_{3,2})=LOC(a_{0,0})+(i*n+j)*k$$

$$=2000+(3*4+2)*4=2056。$$

3. 矩阵的压缩存储

- 特殊矩阵（对称矩阵、三角矩阵、对角矩阵）

特殊矩阵是指非零元素或零元素的分布有一定规律的矩阵,为了节省存储空间,特别是在高阶矩阵的情况下,可以利用特殊矩阵的规律,对它们进行压缩存储,也就是说,使多个相同的非零元素共享同一个存储单元,对零元素不分配存储空间。

●对称矩阵的压缩存储

若一个n阶方阵A中的元素满足$a_{i,j}=a_{j,i}(1\le i,j\le n)$, 则称其为n阶对称矩阵。

（1）只存储对称矩阵中上三角或下三角中的元素，
（2）将n^2个元素压缩存储到$n(n+1)/2$个元素的空间中，以一个一维数组作为A的存储空间。

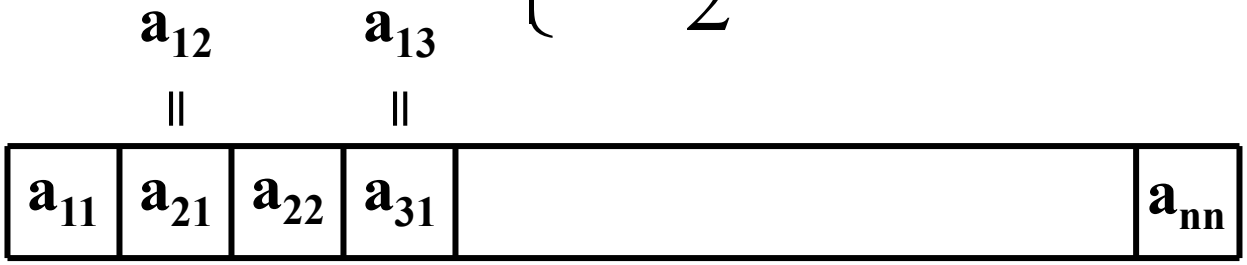
$$A_{n\times n}=\begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \cdots & & & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}$$

n^2个元素←→ n(n+1)/2个元素

$a_{ij}(1 \le i,j \le n)$←→ B[n(n+1)/2]

- 1+2+…+(i-1)+j-1；
- a_{ij} ←→ a_{ji}

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

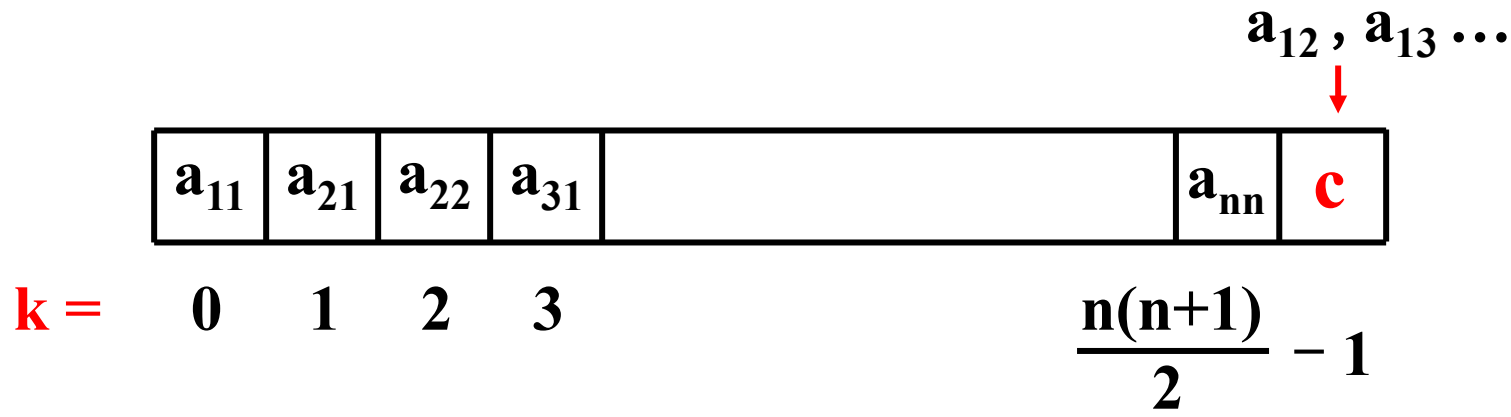


- 下三角矩阵的压缩存储　B[n(n+1)/2+1]

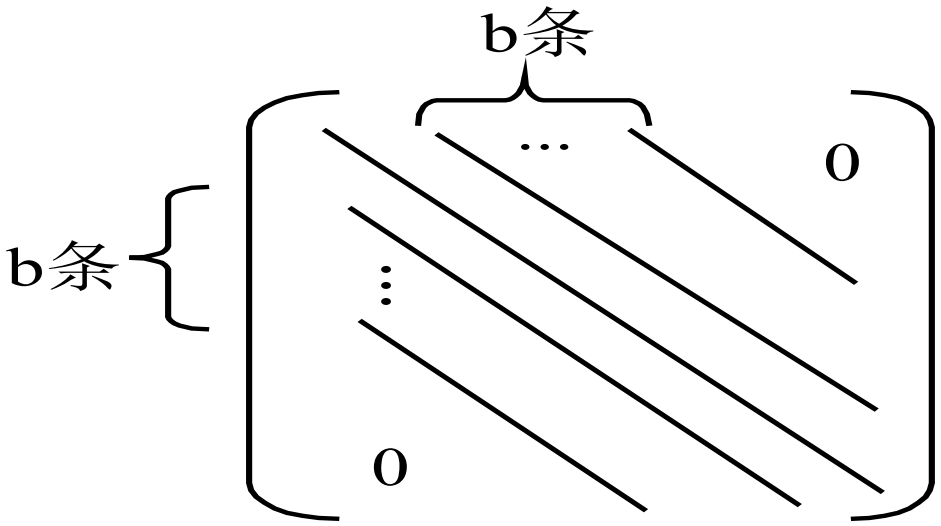
$$A_{n\times n}=\begin{bmatrix} a_{1,1} & & & \\ a_{2,1} & a_{2,2} & & \\ \vdots & \vdots & \ddots & \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}$$

C

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



- 对角矩阵：所有的非零元都集中在以主对角线为中心的带状区域内。



半带宽为b的对角矩阵

- 三对角矩阵

B[3n-2];

k=2(i-1)+j-1;

$$A_{n\times n}=\begin{bmatrix} a_{11} & a_{12} & & & & & \\ a_{21} & a_{22} & a_{23} & & & & \\ & a_{32} & a_{33} & a_{34} & & & \\ & & a_{43} & a_{44} & a_{45} & & \\ & & & \cdots & \cdots & \cdots & \\ & & & & & \cdots & \cdots \end{bmatrix}$$

①共有3n-2个非零元。

②主对角线左下方的元素下标有关系式：i=j+1

k = 3(i-1)-1+1-1 = 3(i-1)-1 = 2(i-1)+i-2 = 2(i-1)+j-1.

③主对角线上的元素下标有关系式：i=j

k = 3(i-1)-1+2-1 = 3(i-1) = 2(i-1)+i-1 = 2(i-1)+j-1.

④主对角线右上方的元素下标有关系式：i=j-1

k = 3(i-1)-1+3-1 = 3(i-1)+1 = 2(i-1)+i = 2(i-1)+j-1.

- 稀疏矩阵

  非零元较零元少，且没有一定规律。

  m×n的矩阵，有t个非零元，

  矩阵的稀疏因子：δ=t/(m*n)，当δ≤0.05时为稀疏矩阵。

压缩存储，只存储非零元

①三元组顺序表(i，j，$a_{i,j}$)

②行逻辑链接的顺序表

③十字链表

$$A_{6\times7}=\begin{bmatrix}0&12&9&0&0&0&0\\0&0&0&0&0&0&0\\-3&0&0&0&0&14&0\\0&0&24&0&0&0&0\\0&18&0&0&6&0&0\\15&0&0&-7&0&0&0\end{bmatrix}$$

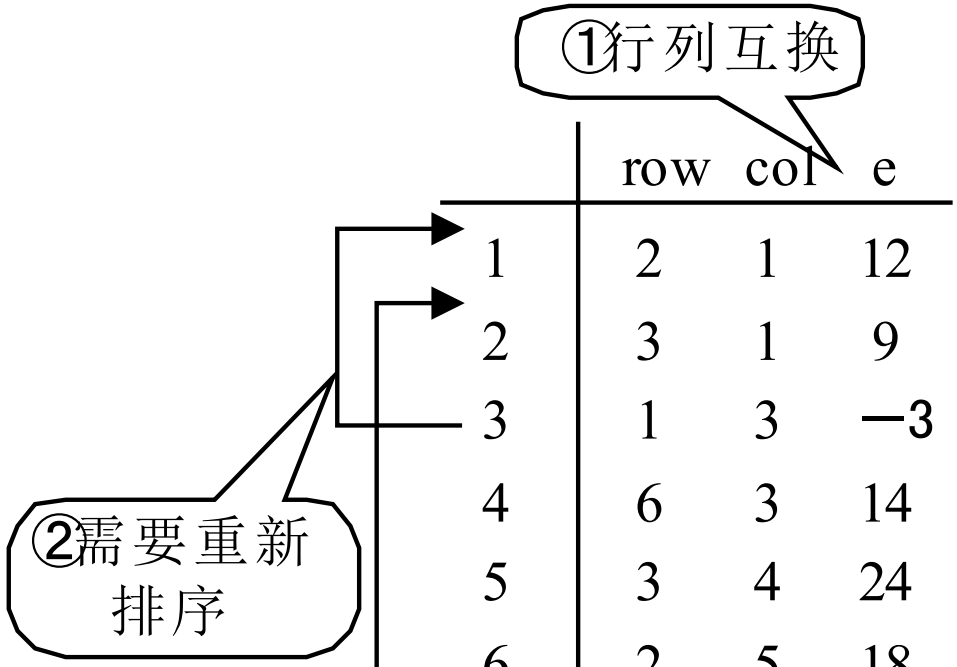
((1,2,12),(1,3,9),(3,1,-3),(3,6,14),
(4,3,24),(5,2,18),(6,1,15),(6,4,-7))

```
#define MaxSize  100  /*矩阵中非零元素最多个数*/
typedef struct
{    int    i;              /*行号*/
     int   j;               /*列号*/
     ElemType e;            /*元素值*/
} Triple;                   /*三元组定义*/
typedef struct
{    int     rows;                  /*行数*/
     int     cols;                  /*列数*/
     int     nums;                  /*非零元素个数*/
    Triple  data[MaxSize+1];  /*data[0]未用*/
} TSMatrix;          /*三元组顺序表定义*/
```

- 用三元组表实现稀疏矩阵的转置运算

一个6*7的矩阵A，以行序为主序顺序排列

	row	col	e
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



	row	col	e
1	2	1	12
2	3	1	9
3	1	3	−3
4	6	3	14
5	3	4	24
6	2	5	18
7	1	6	15
8	4	6	−7

- 矩阵的转置，方法一：

	row	col	e
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

(a) 三元组表 A

	row	col	e
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

(b) 三元组表 B

```
Status TransposeSMatrix(TSMatrix A, TSMatrix &B)
{
    B.rows=A.cols; B.cols=A.rows; B.nums=A.nums;
    if (B.nums!=0){
        q = 1;   /*q为B.data的下标*/
        for (col=1; col<=A.cols; col++)
            for (p=1; p<=A.nums; p++) /*p为A.data的下标*/
                if (A.data[p].j == col){
                    B.data[q].i = A.data[p].j;
                    B.data[q].j = A.data[p].i;
                    B.data[q].e = A.data[p].e;
                    q++;
                }
    }
    return OK;
}
```

方法1时间复杂度：

O(cols*nums)

当非零元个数nums和cols*rows同数量级时，

$O(rows*cols^2)$

仅适用于nums<<rows*cols.

常规存储方式时， 实现矩阵转置的经典算法如下：

```
for(i=0; i<m; i++)
    for (j=0; j< n; j++)
        T[i][ j] = M[j][i];
```

O(cols*rows)

- 方法二：

依次按三元组表A(6*7)的次序进行转置，转置后直接放到三元组表B的正确位置上。这种转置算法称为快速转置算法。

row	col	e
1	2	12
1	3	9
3	1	-3
3	6	14
4	3	24
5	2	18
6	1	15
6	4	-7

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

num[col]: A中第col列非零元的个数。

cpot[col]: B中第col行第一个非零元在B中的位置

- Status FastTranSMatrix(TSMatrix A, TSMatrix &B)

```
{
    B.rows=A.cols; B.cols=A.rows; B.nums=A.nums;
    if(B.nums){
        for(col=1; col<=A.cols; ++col)   num[col] = 0;
        for(t=1; t<=A.nums; ++t)  ++num[A.data[t].j];
        cpot[1] = 1;
        for(col=2; col<=A.cols; ++col)
            cpot[col] = cpot[col-1]+num[col-1];
        for(p=1; p<=A.nums; ++p){
            col = A.data[p].j;   q = cpot[col];
            B.data[q].i = A.data[p].j;  B.data[q].j = A.data[p].i;
            B.data[q].e = A.data[p].e;   ++cpot[col];
        }
    }
    return OK;
}
```

O(cols+nums)

当nums和cols*rows同数量级时

O(rows*cols)

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

	row	col	e
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

	row	col	e	
1	1	3	-3	← cpot[1]
2	1	6	15	
3	2	1	12	← cpot[2]
4	2	5	18	← cpot[2]
5	3	1	9	← cpot[3]
6	3	4	24	← cpot[3]
7	4	6	-7	← cpot[4]
8	6	3	14	← cpot[6]

小结

基本学习要点如下：

(1)理解数组和一般线性表之间的差异。

(2)重点掌握数组的顺序存储结构和元素地址计算方法。

(3)掌握各种特殊矩阵如对称矩阵、上、下三角矩阵和对角矩阵的压缩存储方法。

(4)掌握稀疏矩阵的各种存储结构以及基本运算实现算法。

(5)灵活运用数组这种数据结构解决一些综合应用问题。

- 练习：

  将一个A[1..100, 1..100]的三对角矩阵，按行优先存入一维数组B[1..298]，A中元素$a_{66,65}$在B数组中的位置k=?。

在主对角线左下方，65*3-1+1 = 195。

- 作业：

  5.2

  5.25

5.4 广义表

1. 广义表的定义（列表）

广义表是线性表的推广，是由零个或多个单元素或子表所组成的有限序列。

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

a_i：是单个数据元素,则a_i是广义表的原子；如果a_i是一个广义表,则a_i是广义表的子表。

规定用小写字母表示原子,用大写字母表示广义表的 表名。

- 广义表与线性表的区别：

线性表的成份都是结构上不可分的单个数据元素，而广义表的成份即可以是单元素，也可以是有结构的表，其定义是递归的定义。

- 广义表的长度：广义表中所含元素的个数 **n,n≥0**。
- 广义表的深度：广义表展开后所含的括号的最大层数。（广义表中括号嵌套的最大次数，广义表中括弧的重数）

- D=( )空表，长度为0，深度为1。
- A=(a, (b, c)) 长度为2，第一个元素为原子a，第二个元素为子表(b, c)，深度为2。
- B=(A, A, D) 长度为3， 其前两个元素为表A， 第三个元素为空表D，深度为3。
- C=(a, C)　长度为2递归定义的广义表，C相当于无穷表C=(a,(a, (a,…)))，深度无限。

  递归表的深度是无穷值,长度是有限值。
- F=((a, (a, b), ((a, b), c)))

  长度为1，深度为4。

- 广义表的表头(Head)和表尾(Tail)：

当广义表非空时，称第一个元素a_1为广义表的表头，其余元素组成的表$(a_2, a_3, \ldots, a_n)$称为广义表的表尾。

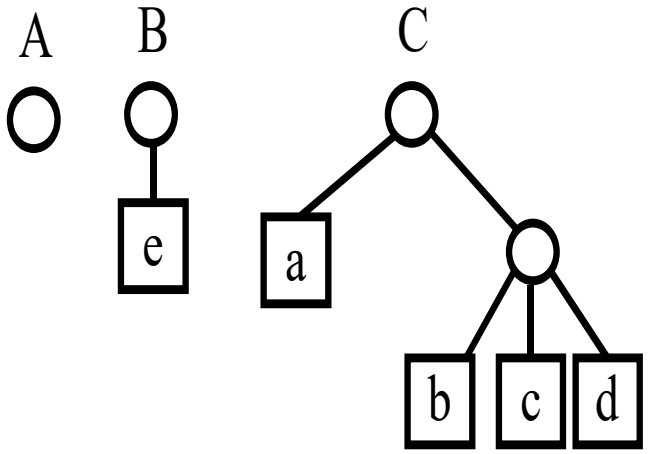
表头可能是原子，也可能是广义表，但表尾一定是广义表。

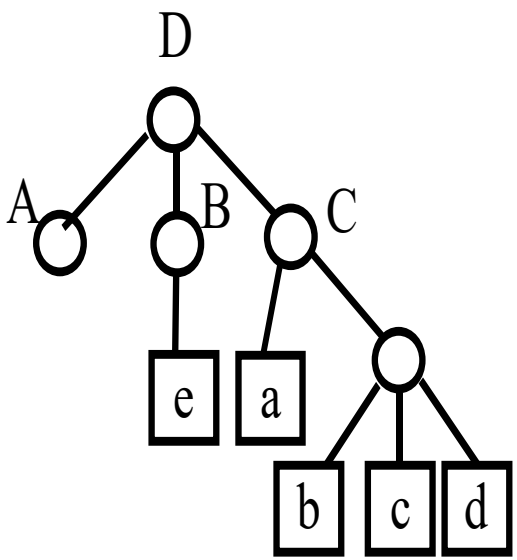
- 广义表的图形表示

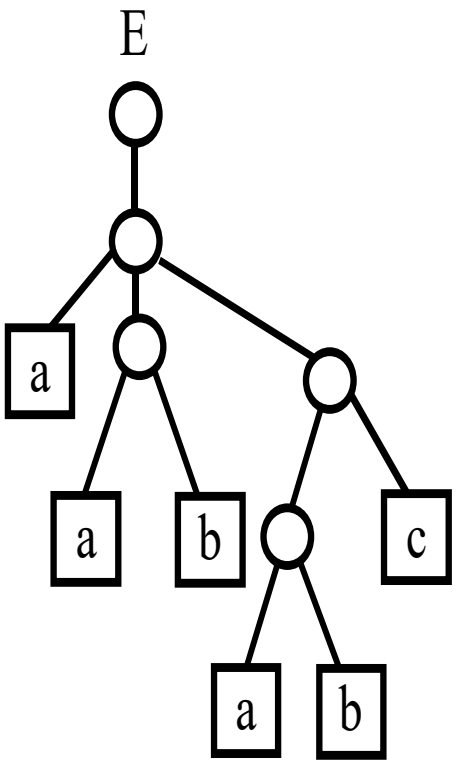
  用方框表示原子，用圆圈表示广义表。

A=( ); B=(e); C=(a, (b, c, d))

D=(A, B, C); E=((a, (a, b), ((a, b), c)))







2. 广义表的基本操作

- 取表头 GetHead(LS) = a_1。
- 取表尾 GetTail(LS) = $(a_2,a_3,\ldots,a_n)$。

①B=(e)　　GetHead(B) = e; GetTail(B) = ( ).

②A=(a, ((b, c), d, e))

GetHead(GetHead(GetTail(A))) =

GetHead(GetHead((((b, c), d, e)))) =

GetHead(((b, c), d, e)) = (b, c).

③A=( );　B = (( ))

A空表，长度0，深度1，无表头和表尾；

B长度1，深度2，表头( )，表尾( )。

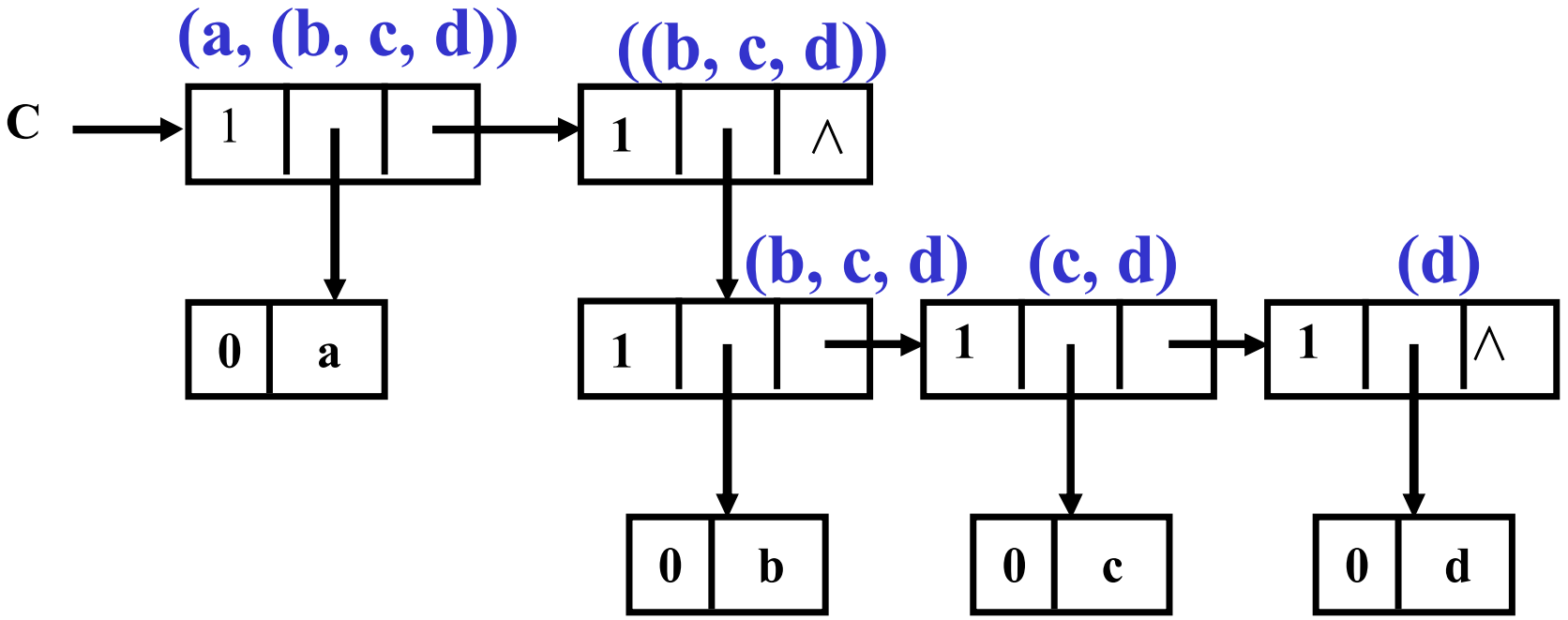
3. 广义表的存储结构

广义表是一种递归的数据结构,因此很难为每个广义表分配固定大小的存储空间,所以其存储结构只好采用动态链式结构。

- 广义表的头尾链表存储表示
- 广义表的扩展线性链表存储表示

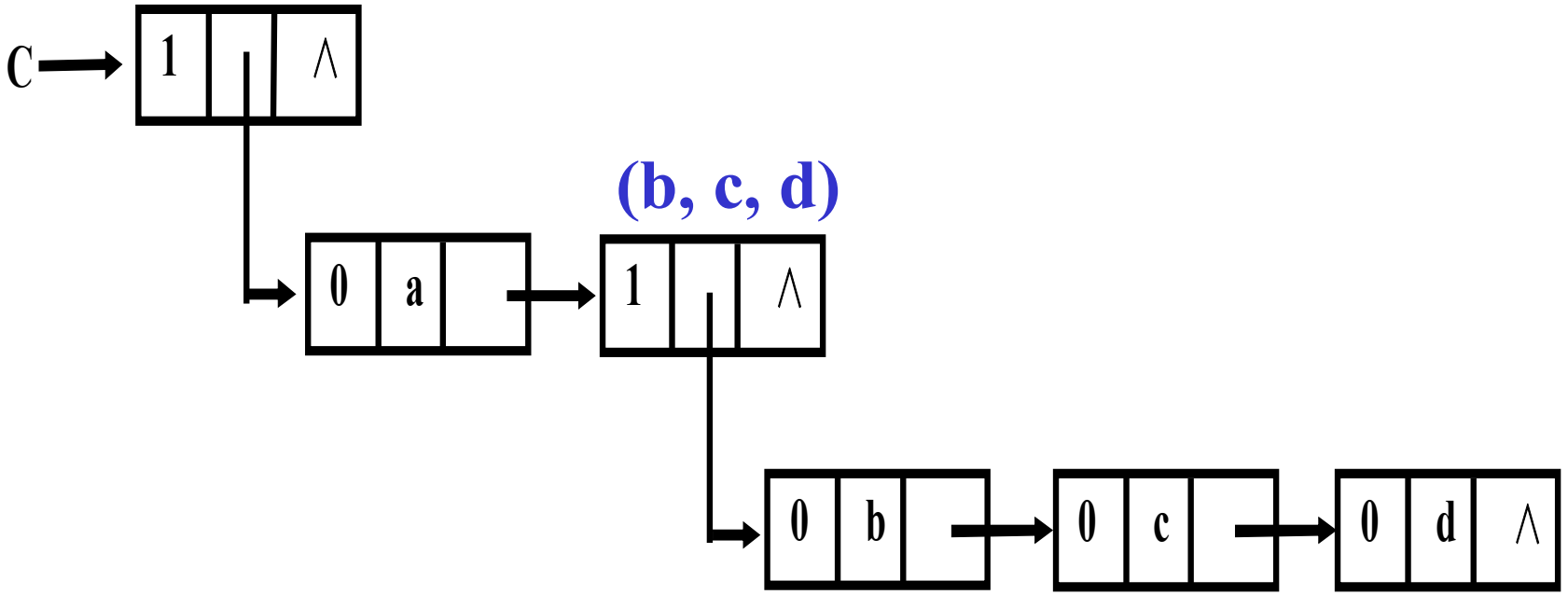
- 广义表的头尾链表存储表示

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



- 广义表的扩展线性链表存储表示（带表头结点）

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



- 作业：

5.10