数据结构

Data Structure

张先宜

手机：18056307221 13909696718

邮箱：zxianyi@163.com

QQ：702190939

QQ群：XC数据结构交流群 275437164

第4章 串、数组与广义表

【本章内容】

4.1 串

【本节内容】

4.1.1 串的定义

4.1.2 串的运算

4.2.3 串的存储

4.1.1 串（string）的定义

【定义】串（string），即字符串，是由n个字符$a_1, a_2, \ldots, a_n$组成的有限序列。或：元素是字符的线性表。

- 记作　S="$a_1, a_2, \ldots, a_n$"，
- 其中 n >=0 为串长度。　n=0 时为空串。

【注意】空串和空格串的区别。

- **空串——没有元素。**
- **空格串——元素是空格符。**

- 子串 —— 串S中若干个连续的字符组成的序列。

4.1.2 串的运算

① **赋值 -- =**

将一个串值S1传送给一个串名S。（如：S=S1）

② **求长度 -- length(S)**

返回串S的长度值。

③ **连接运算 -- +**

将S1和S2连接成一个新串。如： S1+S2

④ **求子串 -- substr (S, i, j)**

返回串S从第i个元素开始的j个元素所组成 的子串。

⑤ **串比较 -- strcmp(S1,S2)**

- 比较两个串的大小。左对齐按ASCII码逐字符进行比较。
- 结果可定为：
  - **-1 -- S1与S2第一个不同的字符的ASCII值S1<S2**
  - **0 -- S1与S2相同**
  - **1 -- 第一个不同字符ASCII值S1>S2**

■ **也可以定为其他取值方式**

- 以下运算可由上述基本运算完成>>>>

⑥ 插入 -- insert (S, i, S1)

将子串S1插入到串S的从第i个字符开始的位置上。

⑦ 删除 -- deleteStr (S, i, len)

删除串S中从第i个字符开始的len个字符。

⑧ 替换 -- replace(S, i, len, S1)

用字串S1替换串S中从第i个字符开始的len个字符。

⑨ 定位 -- index (S, S1)

模式匹配，查询S1在S中第一次出现的位置。

【例】S=“a1,a2,…,an”，在S的第i个位置插入S1

- 使用常用运算方法⑥：insert(S, i, S1)，此为常用运算。
- 可转换为以下基本操作：

① X1=substr(S, 1, i-1); //取出S的前i-1个字符到X1

② X2=substr(S, i, length(S) - (i - 1));
//取S后面字符到X2

③ S = X1 + S1 + X2; //为基本运算—合成新的S

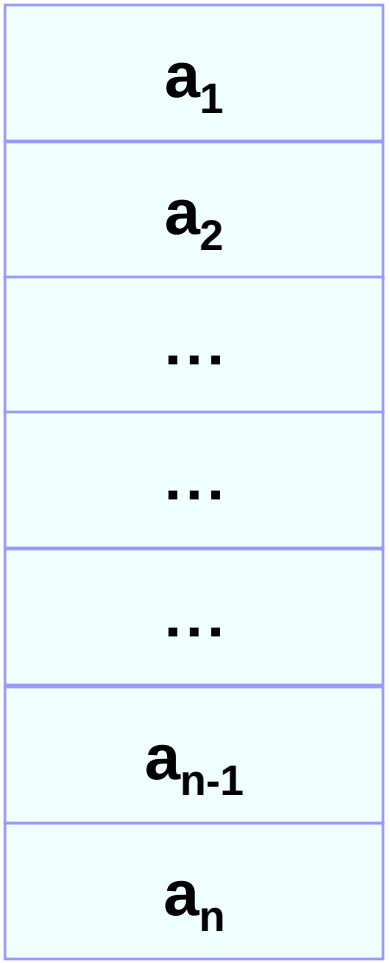
- 可见：后4种常用运算，可由前5种基本操作联合完成。

4.1.3 串的存储

1. 顺序串

（1）非紧凑格式：

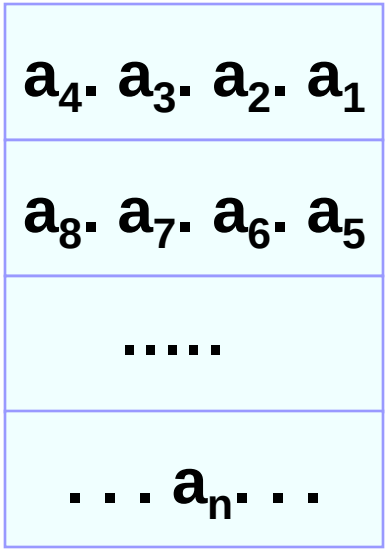
- 一个单元存储一个字符；
- 【优点】运算方便；
- 【缺点】浪费空间。



非紧凑格式

（2）紧凑格式：

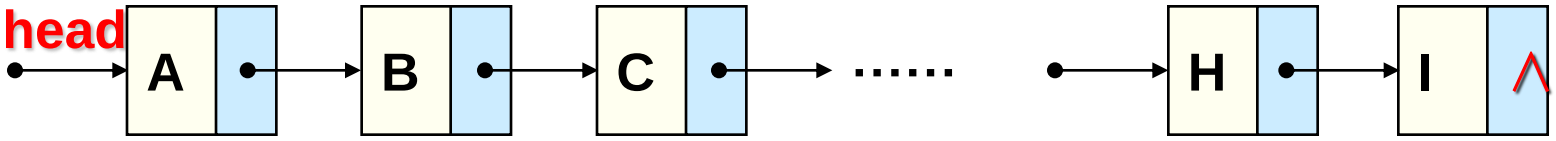
- 一个单元存储多个字符；
- 【优点】节省空间；
- 【缺点】运算不方便。



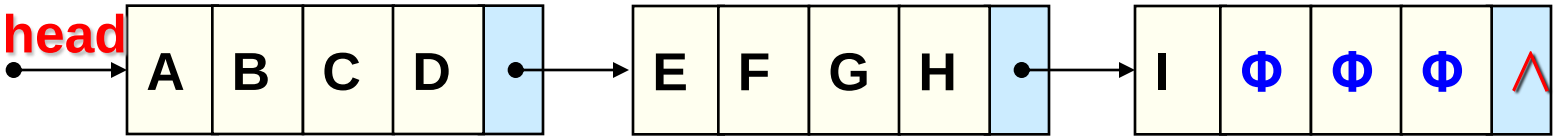
紧凑格式

2. 链串

- 采用块链存储结构。
- 一个结点可以存储一个字符，也可以存储多个字符。
- 一个结点存储多个字符时，最后结点可能有空位置。



(a) 结点大小为 1 的链串



(b) 结点大小为 4 的链串

【例3.1】结点容量为1的链串S1和S2实现运算strcmp(S1,S2)，根据情况返回-1、0和1。

【分析】

(1)可用带头结点的单链表实现；

(2) #typedef char elementType;

(3)逐结点比较（S1指针Pa、S2指针Pb）：

- 第一个字符不同，返回-1或1；
- 相同，S1和S2均未结束，取下一个字符，循环。
- 当一个串结束，另一个串没结束处理：
  - S1结束，S2未结束，S1<S2，return -1;
  - S1未结束，S2结束，S1>S2，return 1

【算法描述-1】

```
int strcmp( node* S1, node* S2 )
{   node *Pa=S1->next, *Pb=S2->next;
        //pa、pb初始指向两个串的第一个字符（结点）
    while( Pa!=NULL && Pb!=NULL )    //两个串皆有结点
    {
        if( Pa->data==Pb->data )    //S1=S2
        {
            Pa=Pa->next;    //对应字符相同，移到下一个字符
            Pb=Pb->next;
        }
        else if( Pa->data<Pb->data )
            return -1;    //S1<S2
        else
            return 1;     //S1>S2
    }
```

【算法描述-2】

```
        //有一个串结束或两个都结束处理
    if( Pa==NULL )   //S1结束
    {
        if( Pb!=NULL )
            return -1;   //S1结束，S2未结束
        else
            return 0;    //两个串同时结束，为相等的情况
    }
    else   //此为S1未结束，但S2已经结束，所以S1>S2
        return 1;
}
```

The Apple 1 which was sold as a do-it-yourself kit (1976, without the lovely case seen here)

iPhone5

4.2 数组

【本节内容】

4.2.1 数组的定义与运算

4.2.2 数组的存储

4.2.3 矩阵的压缩存储

- **数组是一种简单且非常重要的数据结构；**
- **高级编程语言几乎都支持数组结构；**
- **数组使用和应用非常广泛。**
- **本章讨论数组的实现原理。**

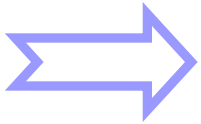
4.2.1 数组的定义和运算

1、数组（Array）的定义：

- 有限个相同类型的变量组成的序列。
  - **若每个变量是一维数组，则为二维数组；**
  - **若每个变量是n-1维数组，则为n维数组。**

$(a_1, a_2, a_3, \ldots, a_n)$

一维数组示例



$$\begin{pmatrix} a_{11},a_{12},\cdots,a_{1n} \\ a_{21},a_{22},\cdots,a_{2n} \\ a_{31},a_{32},\cdots,a_{3n} \\ \cdots \quad \cdots \quad \cdots \quad \cdots \\ a_{m1},a_{m2},\cdots,a_{mn} \end{pmatrix}_{m\times n}$$

二维数组示例

2、数组的运算：

① 给定一组下标，存/取数组元素的值；

② 计算元素的地址

4.2.2 数组的顺序存储

- 以二维数组为例讨论

$$\begin{pmatrix} a_{11},a_{12},\cdots,a_{1n} \\ a_{21},a_{22},\cdots,a_{2n} \\ a_{31},a_{32},\cdots,a_{3n} \\ \cdots\ \ \cdots\ \ \cdots\ \ \cdots \\ a_{m1},a_{m2},\cdots,a_{mn} \end{pmatrix}_{m\times n}$$

1、行优先存储：

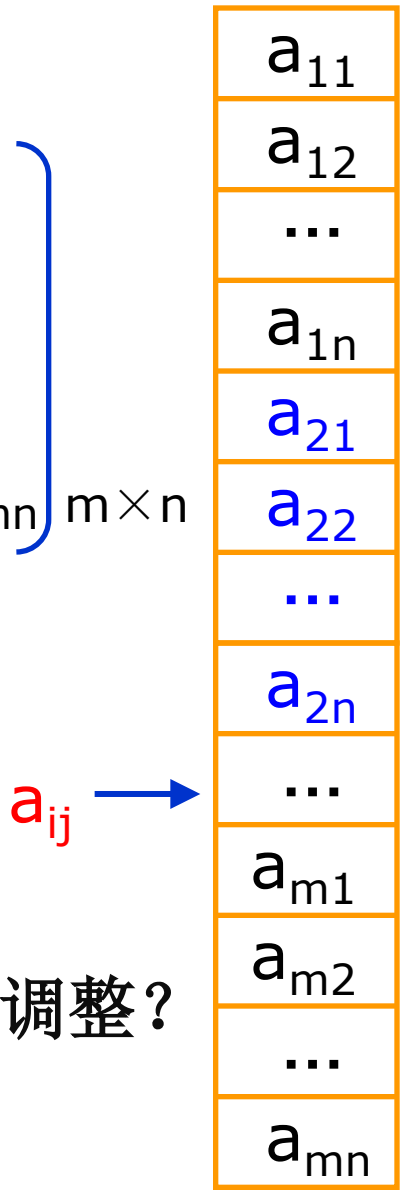
① 求元素a_{ij}的序号

Num=(i-1)*n+j

② 求元素a_{ij}的地址

Addr=Addr0+(Num-1)*C

- 其中，**Addr0为数组首地址；**
- **C为每个元素的长度（字节数）。**



【注意】这里数组下标从1开始。

【问题】如果数组下标从0开始，公式如何调整？

$$\begin{pmatrix} a_{11},a_{12},\cdots,a_{1n} \\ a_{21},a_{22},\cdots,a_{2n} \\ a_{31},a_{32},\cdots,a_{3n} \\ \cdots \quad \cdots \quad \cdots \quad \cdots \\ a_{m1},a_{m2},\cdots,a_{mn} \end{pmatrix}_{m\times n}$$

2、列优先存储：

① 求元素a_{ij}的序号

$$Num=(j-1)*m+i$$

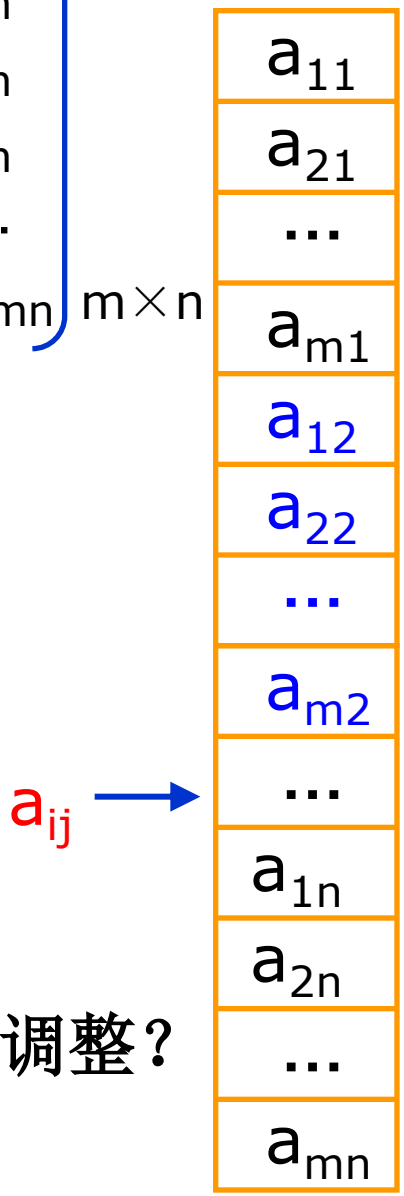
② 求元素a_{ij}的地址

$$Addr=Addr0+(Num-1)*C$$

- 其中，Addr0为数组首地址；
- C为每个元素的长度（字节数）。

【注意】这里数组下标从1开始。

【问题】如果数组下标从0开始，公式如何调整？



- n维数组如何实现行优先和列优先存储呢？
  - 从二维数组可以看出：
    - 行优先：行号变化慢，列号变化快；
    - 列优先：列号变化慢，行号变化快。
  - 对n维数组，要先求出n-1数组，再将n-1维数组的每个元素扩展为一维数组即可。
    - 假定最后一维有m个元素，下标变量为k。
    - n-1维数组元素下标用x（x可能由多个下标变量组成）表示。

- 行优先扩展方法：
  - 将最后一维下标k，作为n-1维数组每个元素a_x的列下标，即为：a_{xk}，即将原来a_x扩展为$a_{x1},a_{x2},\ldots,a_{xm}$。

- 列优先扩展方法：
  - 将最后一维下标k，作为n-1维数组每个元素a_x的行下标，即为：a_{kx}，即将原来a_x扩展为$a_{1x},a_{2x},\ldots,a_{mx}$。

- 下面以3 × 3 × 3的三维数组为例讨论存储顺序
  - 先求出一维数组，无论行优先，还是列优先，存储顺序都为： a_1, a_2, a_3 。

求二维数组顺序

- 行优先扩展为二维数组，按a_{xk}扩展，
- a_1扩展为：a_{11}, a_{12}, a_{13};
- a_2扩展为：a_{21}, a_{22}, a_{23};
- a_3扩展为：a_{31}, a_{32}, a_{33}。

- 列优先扩展为二维数组，按a_{kx}扩展，
- a_1扩展为：a_{11}, a_{21}, a_{31};
- a_2扩展为：a_{12}, a_{22}, a_{32};
- a_3扩展为：a_{13}, a_{23}, a_{33}。

$$\begin{pmatrix} a_{11}, a_{12}, a_{13} \\ a_{21}, a_{22}, a_{23} \\ a_{31}, a_{32}, a_{33} \end{pmatrix}_{3\times3}$$

求三维数组顺序

- 行优先扩展为三维数组，按a_{xk}扩展，
- a_{11}: a_{111}, a_{112}, a_{113}
- a_{12}: a_{121}, a_{122}, a_{123}
- a_{13}: a_{131}, a_{132}, a_{133}
- a_{21}: a_{211}, a_{212}, a_{213}
- a_{22}: a_{221}, a_{222}, a_{223}
- a_{23}: a_{231}, a_{232}, a_{233}
- a_{31}: a_{311}, a_{312}, a_{313}
- a_{32}: a_{321}, a_{322}, a_{323}
- a_{33}: a_{331}, a_{332}, a_{333}

$$\begin{pmatrix} a_{11}, a_{12}, a_{13} \\ a_{21}, a_{22}, a_{23} \\ a_{31}, a_{32}, a_{33} \end{pmatrix}_{3\times3}$$

- 列优先扩展为三维数组，按a_{kx}扩展，
- a_{11}: a_{111}, a_{211}, a_{311}
- a_{21}: a_{121}, a_{221}, a_{321}
- a_{31}: a_{131}, a_{231}, a_{331}
- a_{12}: a_{112}, a_{212}, a_{312}
- a_{22}: a_{122}, a_{222}, a_{322}
- a_{32}: a_{132}, a_{232}, a_{332}
- a_{13}: a_{113}, a_{213}, a_{313}
- a_{23}: a_{123}, a_{223}, a_{323}
- a_{33}: a_{133}, a_{233}, a_{333}

$$\begin{pmatrix} a_{11}, a_{12}, a_{13} \\ a_{21}, a_{22}, a_{23} \\ a_{31}, a_{32}, a_{33} \end{pmatrix}_{3\times 3}$$

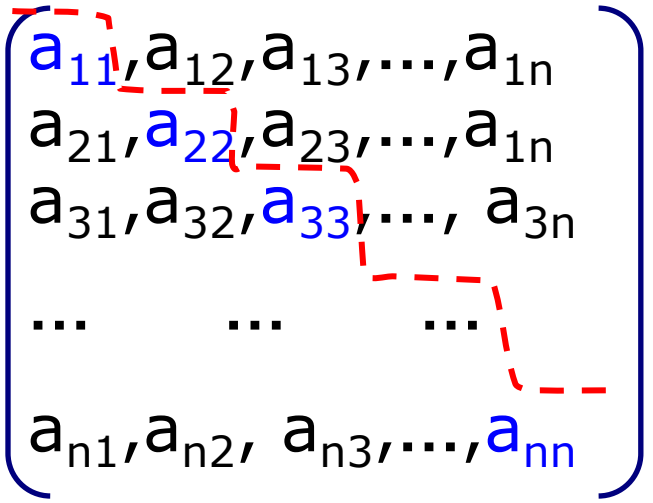
4.2.3 矩阵的压缩存储

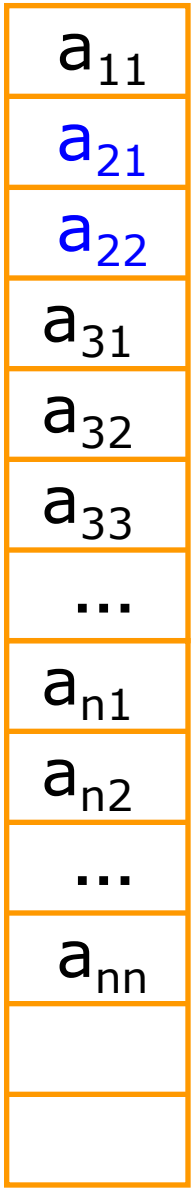
1、对称矩阵

- 特点：$a_{ij}=a_{ji}$
- 求元素a_{ij}的序号

  Num=1+2+…+(i-1)+j
- 求元素a_{ij}的地址

  Addr=Addr0+(Num-1)*C
  - **Addr0和C含义同前。**
- 存储下三角，**i>=j**

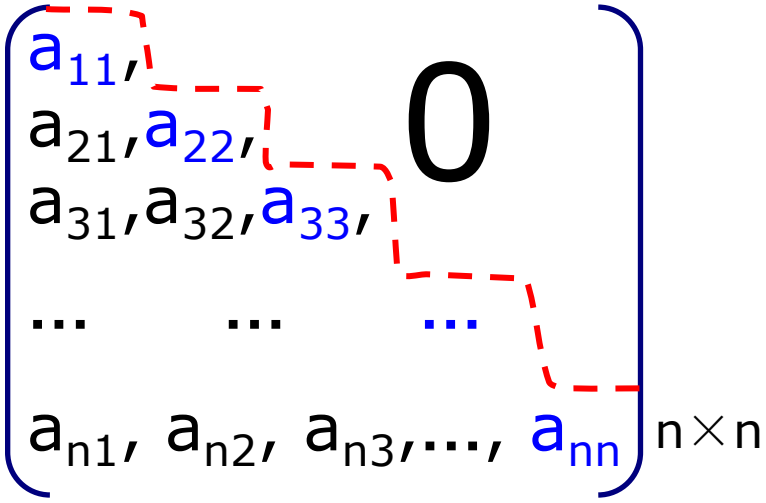


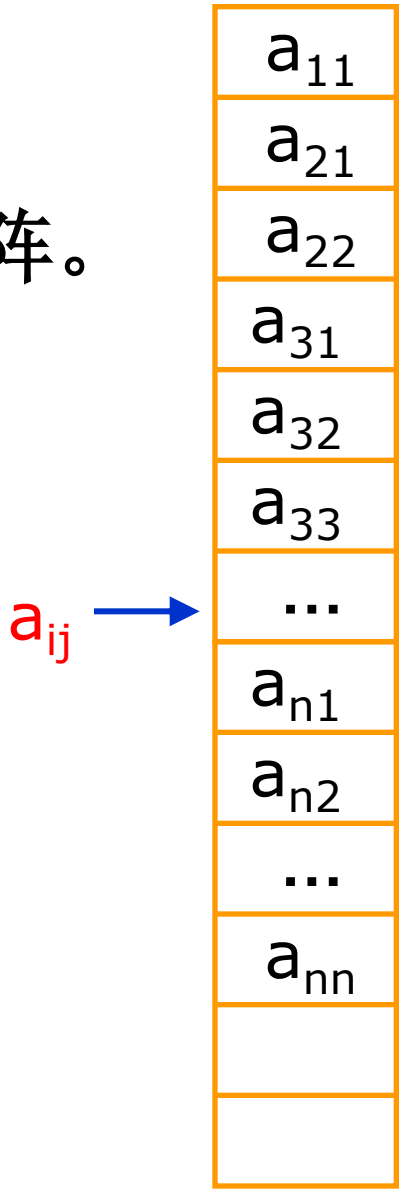


- a_{ij}下三角序号：i>=j
  - Num = 1+2+3+…+(i-1)+j = i*(i-1)/2+j
- a_{ij}上三角序号：i<=j
  - Num = 1+2+3+…+(j-1)+I = j*(j-1)/2+i
- a_{ij}的地址:addr=addr0+(Num-1)*c
  - 其中，Addr0为数组首地址；
  - C为每个元素的长度（字节数）。

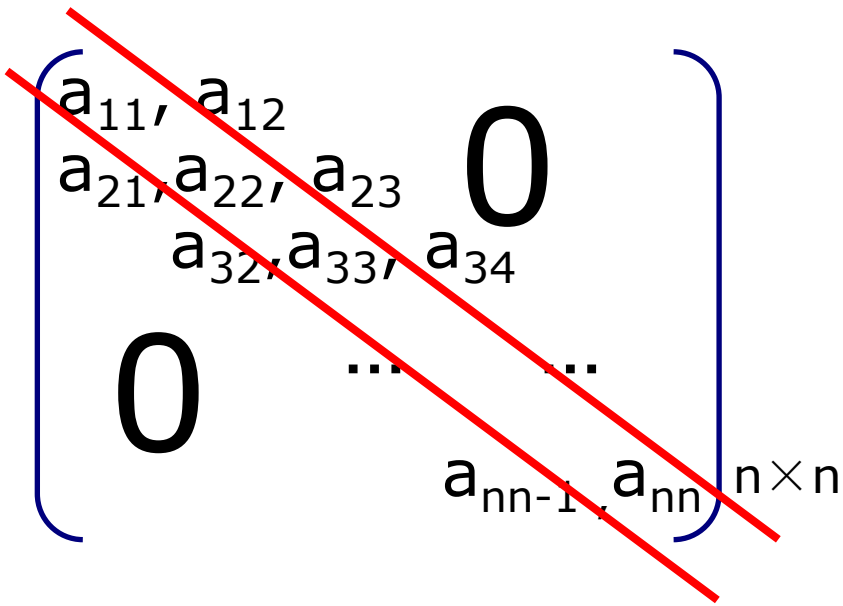
2、三角矩阵

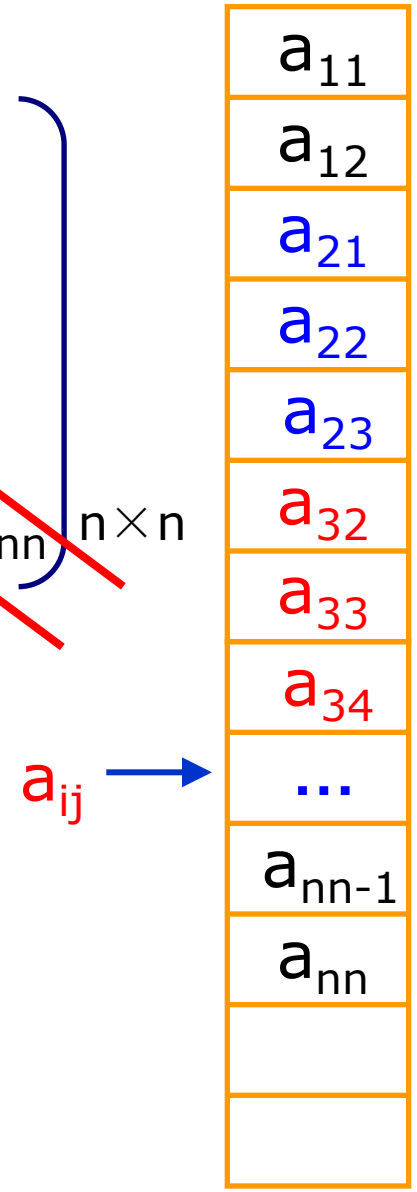
- 对称矩阵的求解方法适用于三角矩阵。





3、对角矩阵





aij →

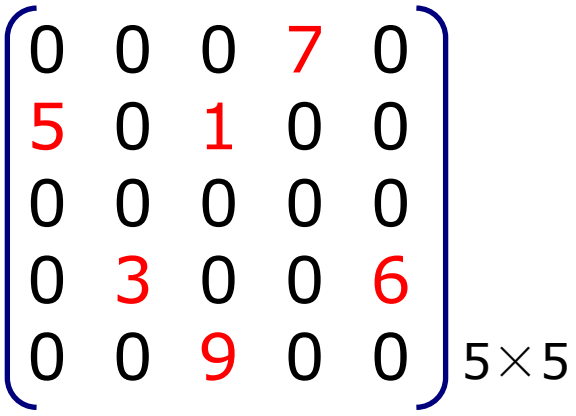
- 求元素a_{ij}的序号

  Num = (3(i-1)-1)+(j-i+2)=2i+j-2

  - **其中：|i-j|<=1**

4、稀疏矩阵的压缩存储

- 数组中非零元素非常少，称为稀疏矩阵。



行 列 值

(1,4,7)
(2,1,5)
(2,3,1)
(4,2,3)
(4,5,6)
(5,3,9)

三元组

行数 列数 非0元素个数

(5,5,6)

- 三元组存储描述

```
typedef struct {
    int i, j;                  //元素行、列下标
    elementType x;             //元素值
} tuple;
```

- 数组结构描述

```
typedef struct {
    int mu, nu, tu;
        //行数、列数、非0元素个数
    tuple data[MaxNum];
        //保存数组的三元组
} spMatrix;
```

- 数组的应用
  - 数组应用非常广泛，这里列举几个案例:
  - ① 矩阵运算等；
  - ② 图像处理，如OpenCV中的基本结构Mat就是一个二维数组；
  - ③ 顺序存储结构存放数据元素；
  - …

习惯比聪明和运气更重要。

4.3 广义表

【本节内容】

- **广义表（Generalized List）**

- **广义表是线性表的推广，在软件设计中有重要的作用。比如人工智能语言LISP中，数据结构和程序本身都表示为广义表。**

4.3.1 广义表的定义

【定义】L 是由 n 个元素 $a_1, a_2, \cdots, a_n$ 组成的有限序列，其中：a_i 是一个原子（不可分割的元素），或者是一个广义表（子表）；

记作：$L=(a_1, a_2 \cdots a_n)$，

称n(n >= 0)为表的长度。

n = 0 时L为空表，记作：L=( )。

- 约定：在书写时，一般用小写字母表示原子，用大写字母表示广义表。

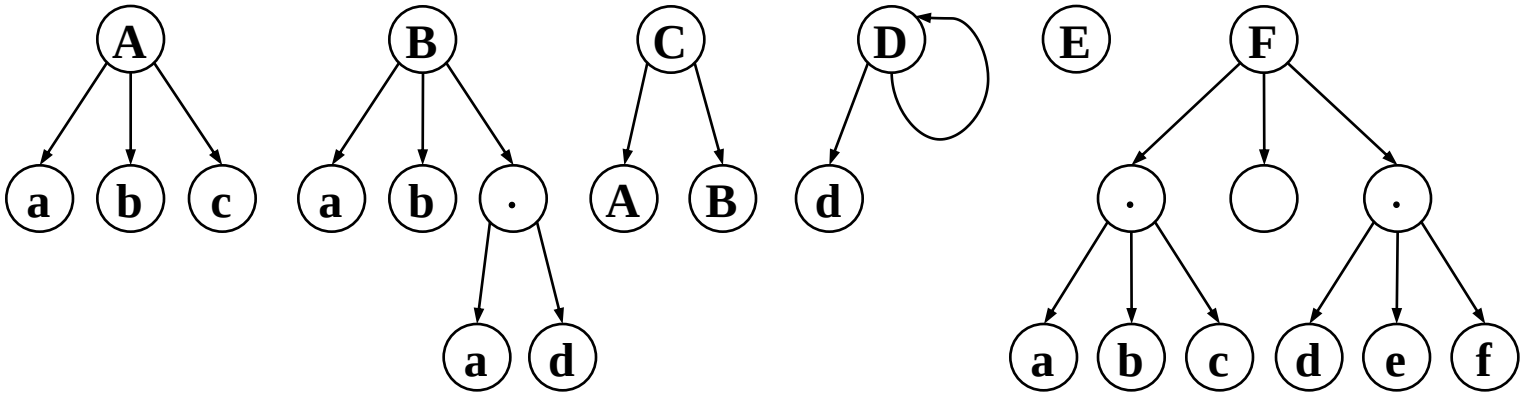
- 由定义可知，广义表是线性表的推广，然而，两者有明显的不同：
  - **线性表中每个元素的类型相同，而广义表中每个元素可以是原子又可以是广义表。**

- 广义表是线性表的推广，当广义表中每个元素都是原子时，即为线性表。

广义表的一些实例

广义表	说明
A=(a,b,c)	表A有3个元素，每个元素都是原子
B=(a,b,(a,d))	表B有3个元素，第1和第2个元素为原子，第3个元素是子表
C=(A,B)	表C有2个元素，都是广义表（子表）
D=(d,D)	表D有2个元素，分别是原子和子表，且子表为D自己
E=( )	表E为空表，没有元素
F=((a,b,c),( ),(d,e,f))	表F有3个元素，皆为子表，其中第二个元素为空表

- 广义表的图形表示，方法如下：
  - ① 用一个“点”代表一个元素，即广义表、子表或原子。
  - ② 用箭头指示一个表的所有元素，并在“点”附近标注元素信息。
  - 前面示例广义表的图形表示如下：



广义表图示法示例

4.3.2 广义表的基本运算

- 针对广义表可以定义多个运算，其中两个最基本的运算是取表头和取表尾运算。

① 取表头： head(L)

- 返回广义表L的第一个元素。
- 对非空广义表，取出的表头元素可能是原子，也可能是子表。

② 取表尾： tail(L)

- 返回广义表L中除去第一个元素，剩下元素构成的子表。
- 对非空广义表，取出的表尾一定是一个子表。

- 前面实例取表头、取表尾结果

取表头	取表尾
head(A)=a	**tail(A)=(b, c)**
head(B)=a	**tail(B)=(b, (a, d))**
head(C)=A	**tail(C)=(B)**
head(D)=d	**tail(D)=(D)**
head(E)无解	**tail(E)无解**
head(F)=(a, b, c)	**tail(F)=(( ),(d, e, f))**

- 特别提醒注意以下两点：

① 广义表中括号用来区分元素还是广义表，是数据的一部分，不能随意增减。

- 例如广义表(a, b)和((a, b))不是同一个广义表。

② 许多初学者对取表尾运算的理解有问题。

- 例如，单纯从字面意义上可能错误地将取表尾运算理解为“取广义表最后一个元素”，这是不对的，应该注意。
- 事实上，初学者可以这样来理解和记忆取表尾运算：取表尾运算就是“去表头”运算。

- 除了取表头和取表尾运算外，有时可能还需要其它一些运算，
  - 如“取出广义表中第3个元素”。
  - 如何实现这类运算？
  - 是否需要另外定义相应的运算呢？
  - 利用取表头和取表尾运算可以实现吗？

head(tail(tail(L)))

【例3.2】对广义表L=(a, (b, (c, d), e))，写出运算head(tail(head(tail(L))))的运算结果。

【解】由内往外逐层执行。

① 执行L1=tail(L)=((b, (c, d), e))；

② 执行L2=head(L1)=(b, (c, d), e)；

③ 执行L3=tail(L2) =((c, d), e)；

④ 执行L4=head(L3)=(c, d)。

☞ 最终结果为：(c, d)。

【思考问题】

① 如果要取出广义表L中第3个和第4个元素，应怎样构造复合函数？

② 对广义表L=(a, (b, (c, d), e))，写出运算head(tail(tail(head(tail(L)))))的运算结果。

4.3.3 广义表的存储

- 因为广义表中元素可以是原子，或是广义表（子表），难以用顺序表存储，通常采用链式存储结构。

1. 头尾链表存储结构

- 因为广义表元素可能是原子，也可能是广义表，需要2种类型的结点：表结点、原子结点。
- 非空广义表可以分解为表头和表尾，所以，一对确定的表头和表尾唯一地确定一个广义表。

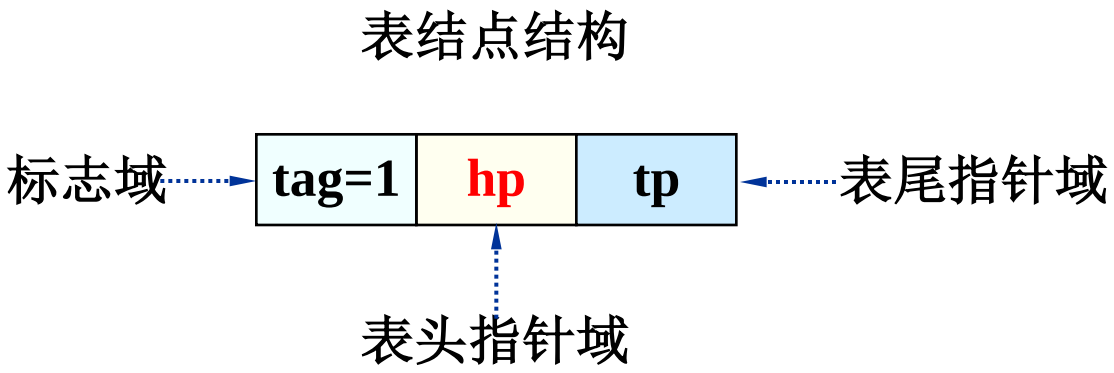
表结点结构

三个域构成：

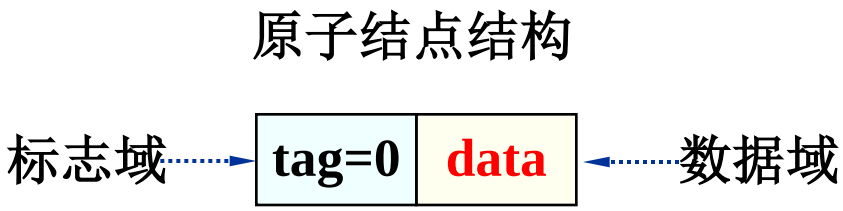
①标志域tag，tag=1标识为表结点；

②表头指针域hp，指示表头；

③表尾指针域tp，指示表尾。



- **原子结点结构**
  - ☞二个域构成：
  - ①标志域tag，tag=0标识为原子结点；
  - ②数据域data，存放原子的元素值；

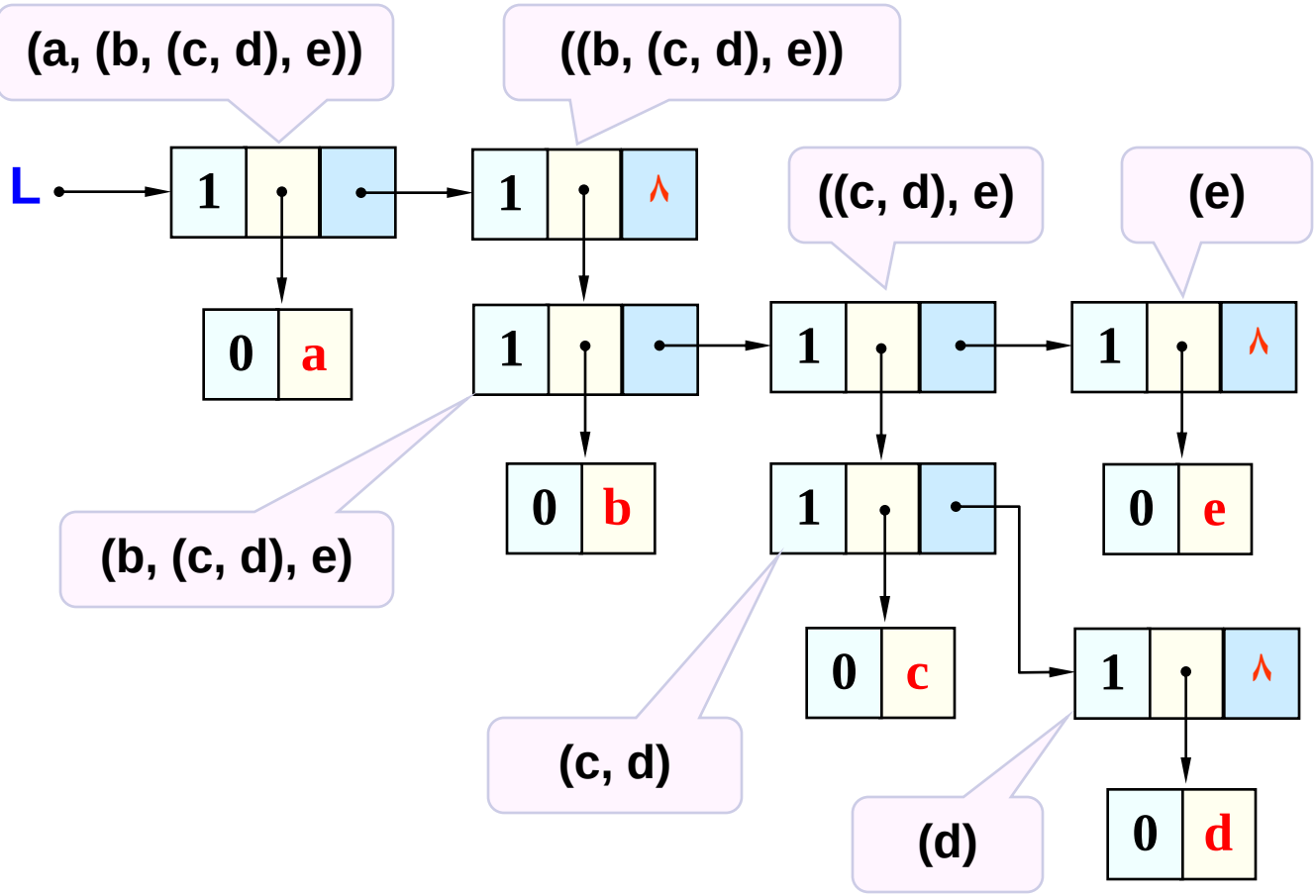


- 头尾链表结点结构描述

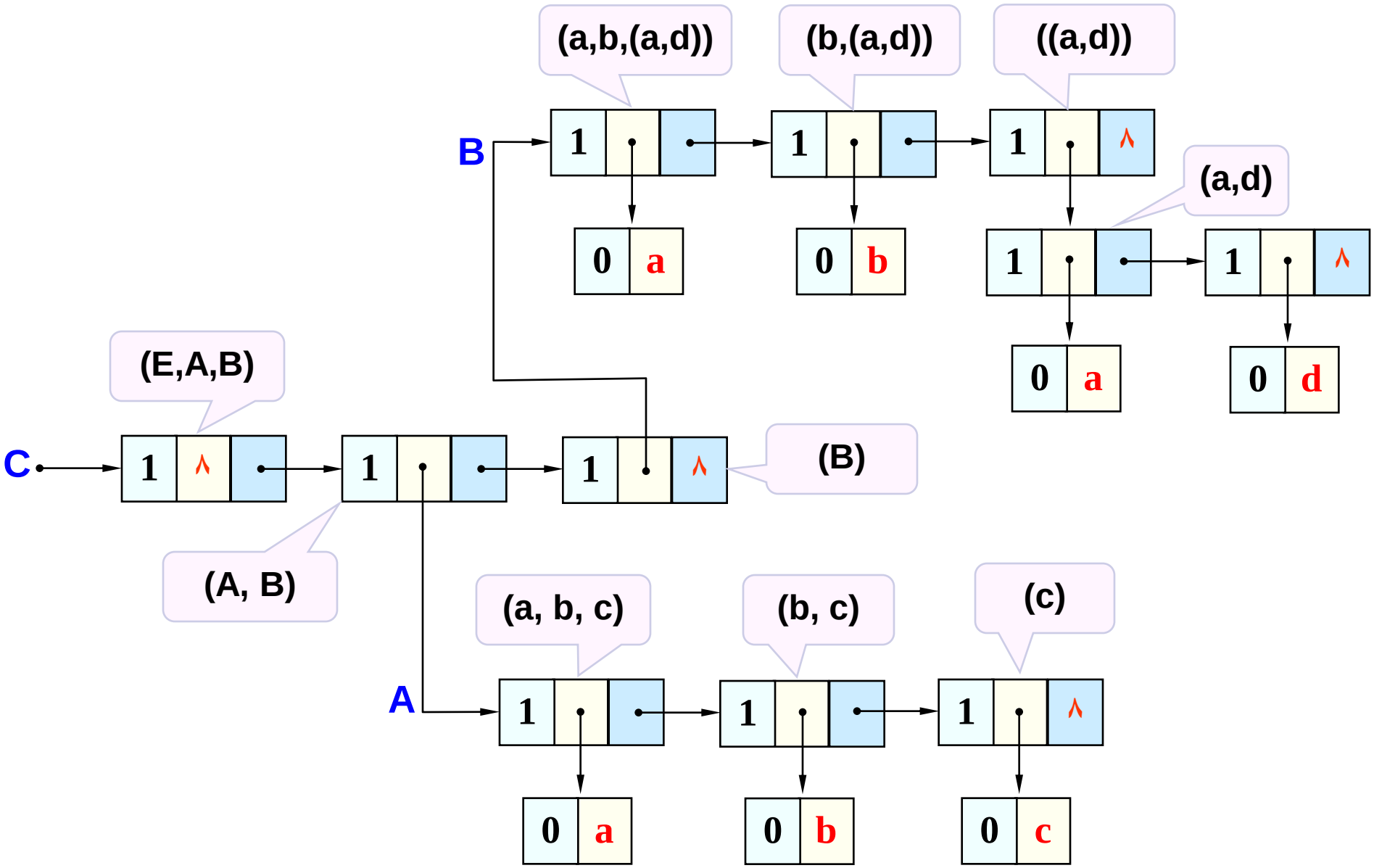
```
typedef char elementType;
typedef struct glNode
{
   int tag;   //tag==1为子表结点；tag==0为原子结点
   union     //原子结点和表结点的联合部分
   {
     elementType data;  //存放原子结点的数据元素
     struct
     {                              //hp为表头指针；tp为表尾指针
            struct glNode *hp, *tp;
     } ptr;     //ptr.hp为表头指针；ptr.tp为表尾指针
   };
}glNode;   //广义表结点类型
```

- 广义表头尾链表表示示意图
  - 对给定的广义表，画一表结点表示；
  - 对广义表执行取表头、取表尾运算。
  - 取表头时，
    - 如果取出的是原子，画一个原子结点，用hp指示；
    - 如果是广义表，画一个表结点，用hp指示；
    - 如果为空表，置hp为空。
  - 取表尾时，
    - 如果为空表，置tp为空；
    - 如果非空，画一个表结点，用tp指示。

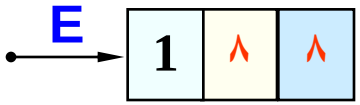
- L=(a, (b, (c, d), e))



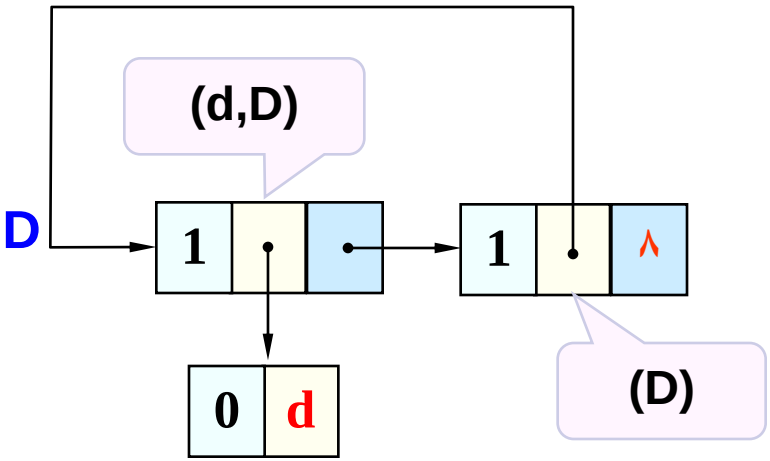
- A=(a,b,c)，B=(a,b,(a,d))，C=(E,A,B)



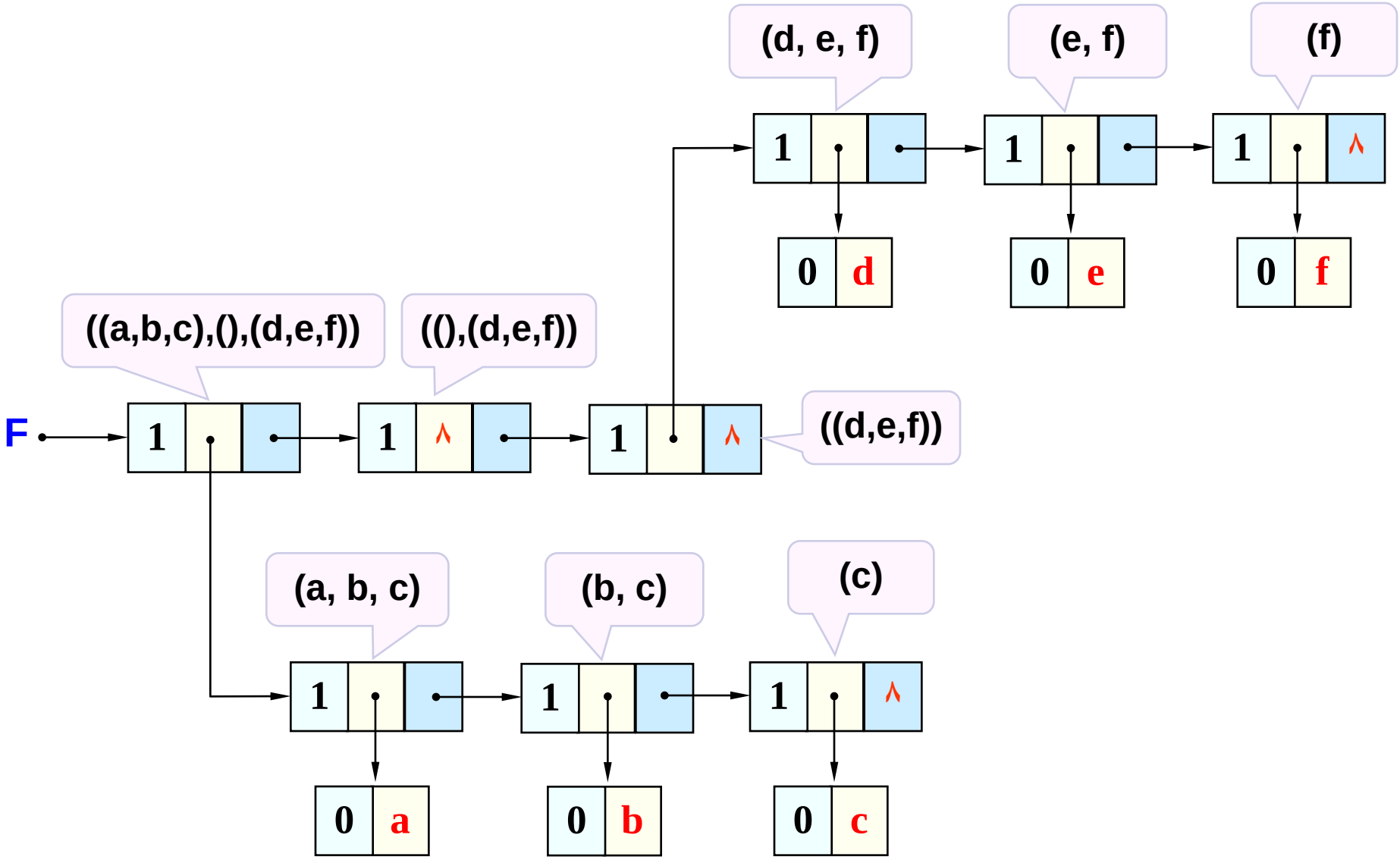
- E=()，空表



- D=(d, D)，递归广义表



- F=((a, b, c), (), (d, e, f))



- 广义表头尾链表表示小结
  - ① 任何广义表对应一个表结点；
  - ② 表结点的hp指示表头，取值可能为：空；原子指针，广义表指针；
  - ③ 表结点的tp指示表尾，取值可能为：空；广义表指针；
  - ④ 对每个广义表进行取表头、取表尾运算，直至空表。
  - ⑤ 从广义表第一个表结点开始，沿着tp指针对结点（表结点）进行计数，即执行tp=tp->tp，直至tp=NULL，计数值为广义表的元素个数，即表长度。

2. 扩展线性链表存储结构

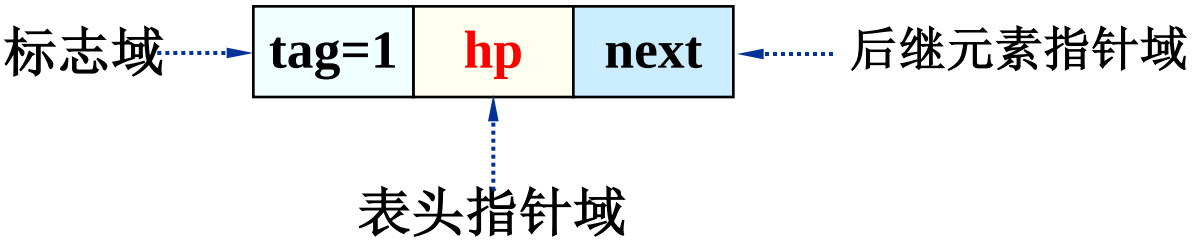
表结点和原子结点都有3个域组成：

① 表结点结构同头尾链表结构；

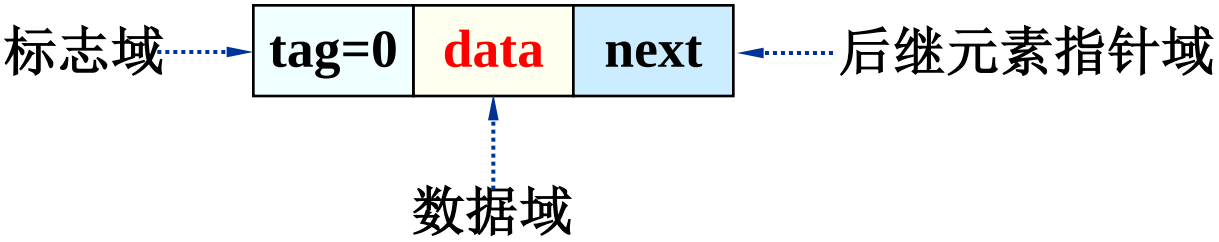
② 原子结点增加一个指针域，指示此原子的在表中的后继元素（原子或子表）。

- 表结点和原子结点结构如下图：

表结点结构

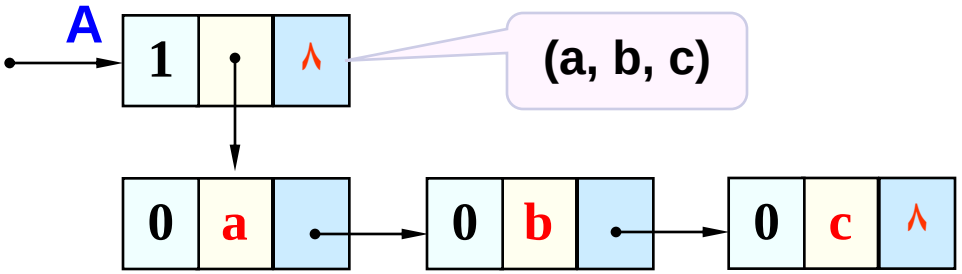


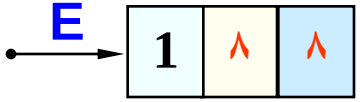
原子结点结构

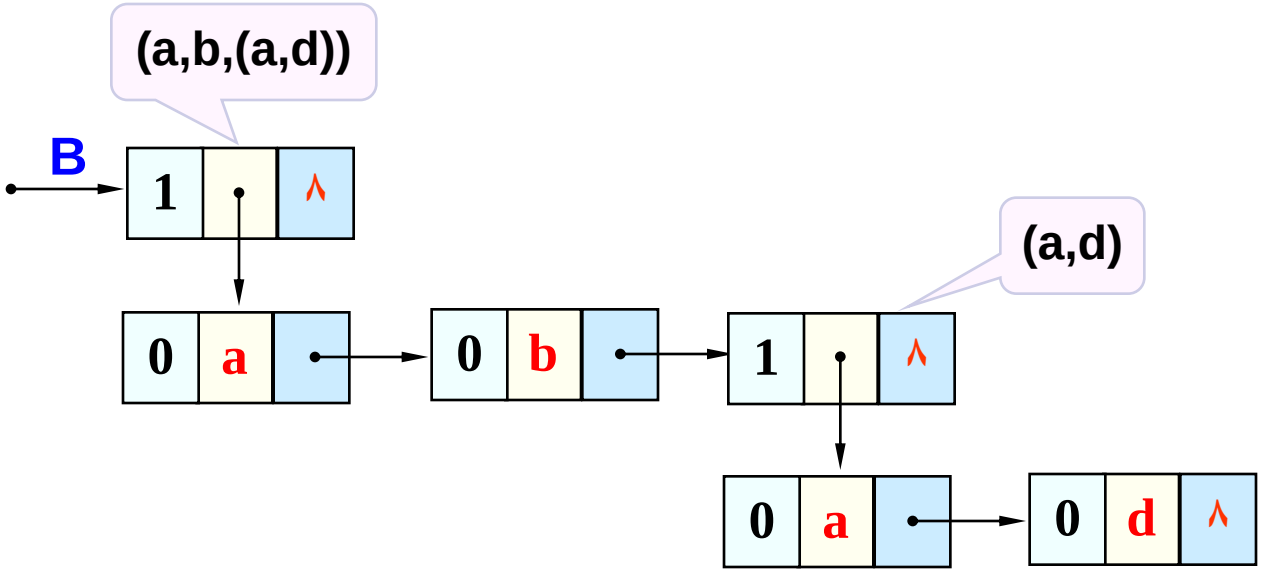


- 广义表扩展链表表示示意图
  - 每个广义表对应一个表结点；
  - 原始广义表结点的next为NULL，类似单链表的头结点；
  - 对广义表的每个元素，从表头元素开始：
    - 如果是原子，对应一个原子结点；
    - 如果是子表，对应一个表结点；
    - 元素结点（原子或子表）之间用next指针链接；
    - 最后一个元素的next为NULL。
  - 一个表中的元素之间用next指针链接，有些书上沿用头尾链表的tp指针名称不太合适，因为元素不一定就是表尾。

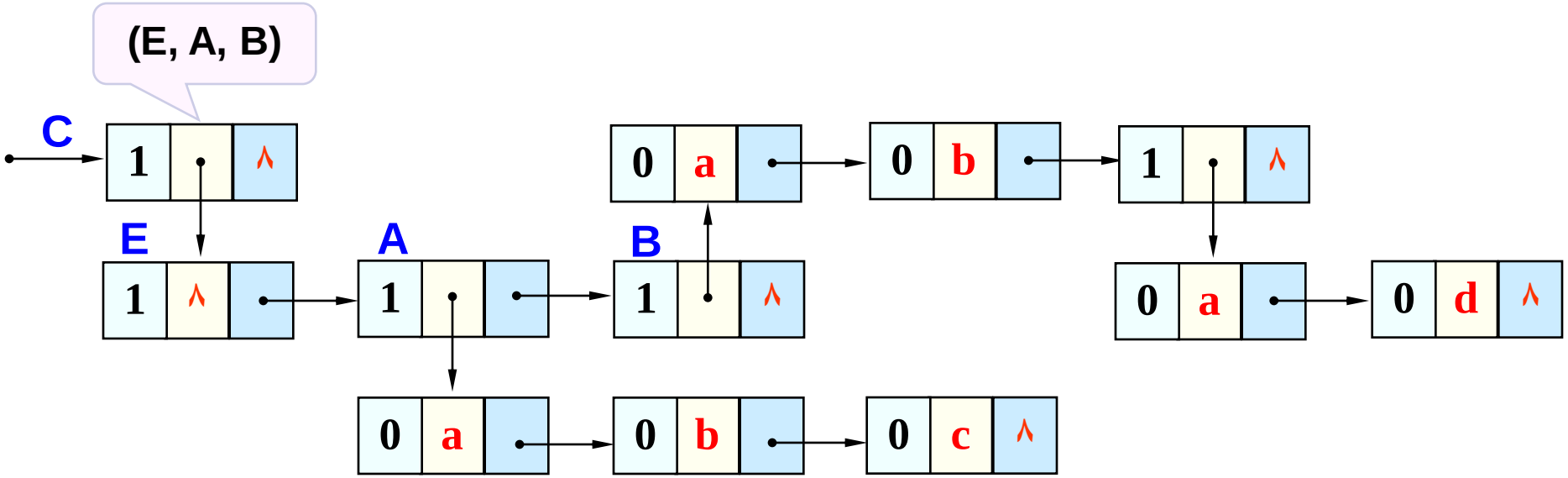
- A=(a,b,c)，B=(a,b,(a,d))，E=()



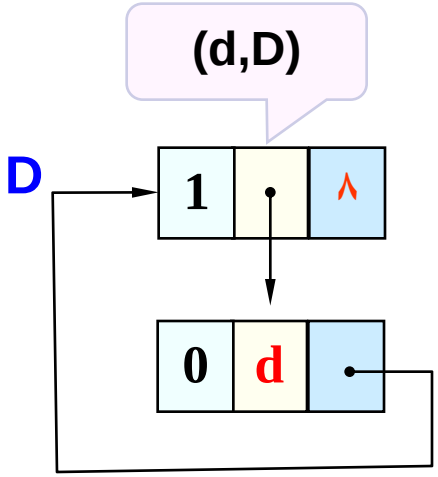




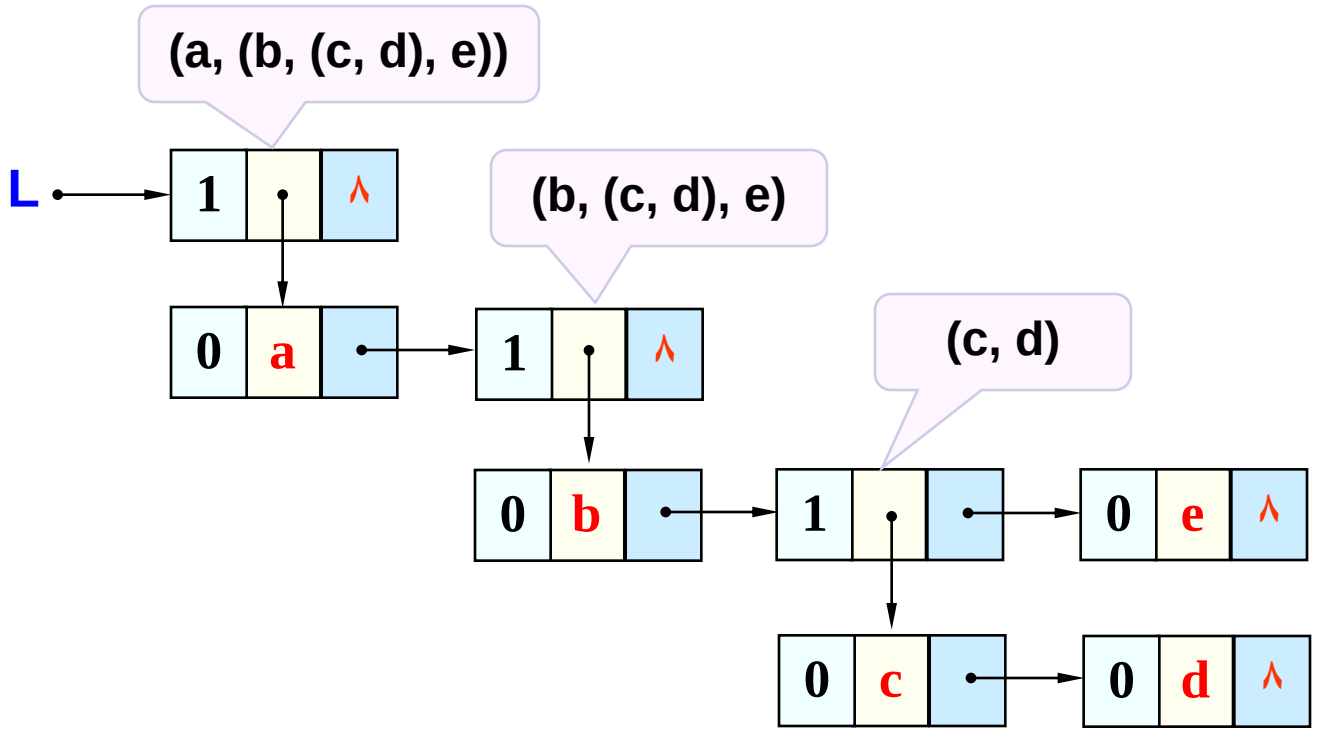
- C=(E,A,B)



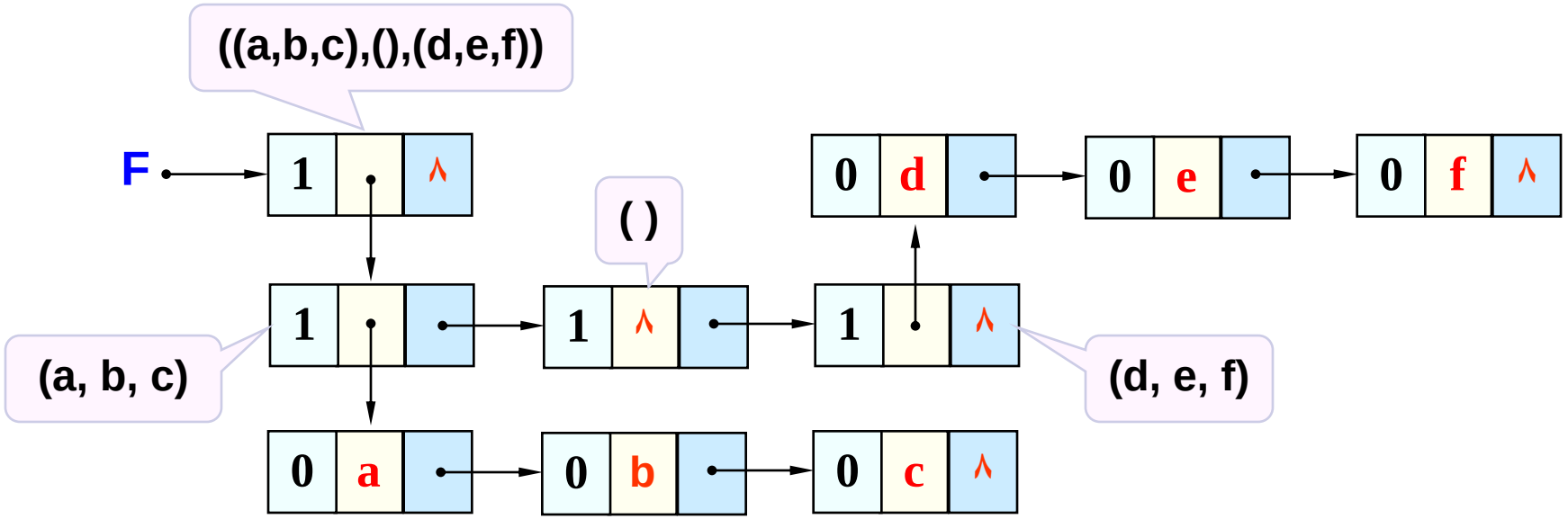
- D=(d, D)，递归广义表



- L=(a, (b, (c, d), e))



- F=((a, b, c), (), (d, e, f))



- 广义表扩展链表表示小结

① 任何广义表对应一个表结点；

② 表结点的hp指示表头元素，取值可能为：空；原子结点指针，广义表结点指针；

③ 结点的next指示下一个元素，取值可能为：空；原子结点指针；广义表结点指针；

④ 对每个广义表进行取表头元素，搜寻下一个元素，并用next链接。同一表中各元素通过next链接，有点像单链表结构。

⑤ 从广义表的hp找到表头元素，沿着表头元素结点的next指针对结点进行计数，即执行u=u->next，直至u=NULL，计数值为广义表的元素个数，即表长度。

- 扩展链表结点结构描述

```
typedef char elementType;
typedef struct glNode
{
   int tag;   //tag==1为表结点；tag==0为原子结点
   struct glNode *next;  //下一个元素结点指针
   union     //原子结点和表结点的联合部分
   {
      elementType data;  //存放原子结点的数据元素
      struct glNode *hp;  //hp为表头指针
   };
}glNode;   //广义表结点类型
```

Thank you !