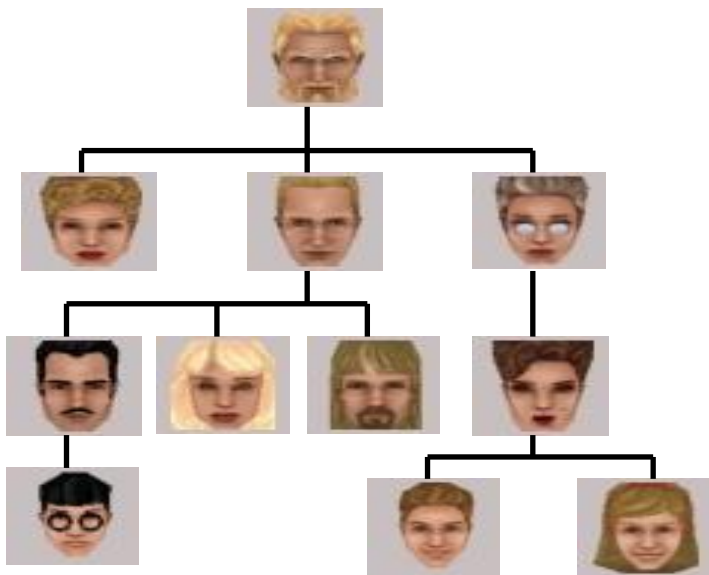


# 3.1 树与树的表示

# 什么是树

客观世界中许多事物存在层次关系

- 人类社会家谱
- 社会组织结构
- 图书信息管理



# 什么是树

分层次组织在管理上具有更高的效率!

数据管理的基本操作之一：查找

如何实现有效率的查找？

# 查找 (Searching)

**查找：**根据某个给定关键字 $K$ ，从集合 $R$ 中找出关键字与 $K$ 相同的记录

**静态查找：**集合中记录是固定的

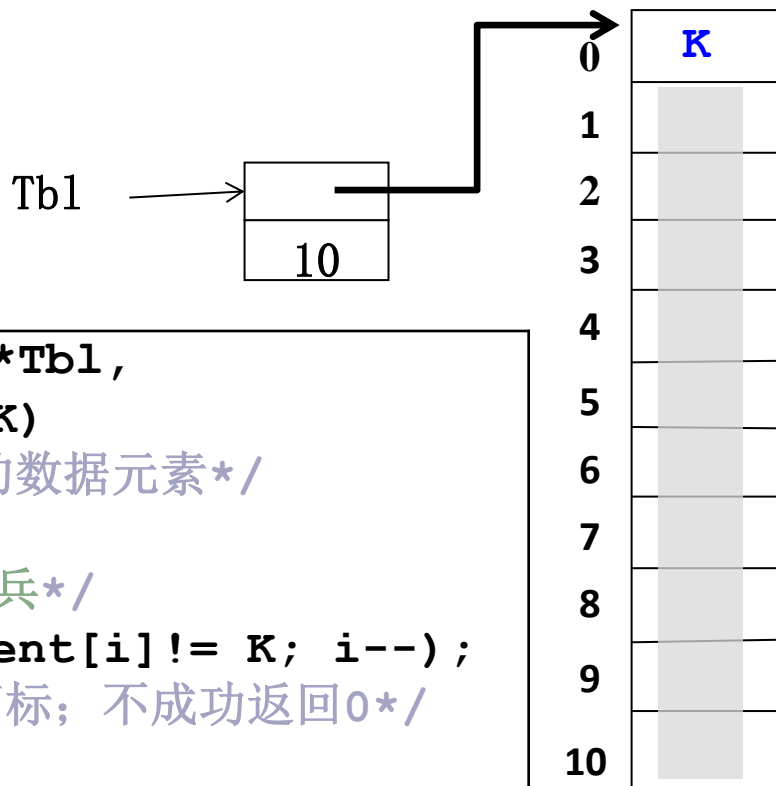
- 没有插入和删除操作，只有查找

**动态查找：**集合中记录是动态变化的

- 除查找，还可能发生插入和删除

# 静态查找

## 方法1：顺序查找



```
int SequentialSearch (StaticTable *Tbl,
                     ElementType K)
{ /*在表Tbl[1]~Tbl[n]中查找关键字为K的数据元素*/
    int i;
    Tbl->Element[0] = K;    /*建立哨兵*/
    for(i = Tbl->Length; Tbl->Element[i] != K; i--);
    return i; /*查找成功返回所在单元下标；不成功返回0*/
}
```

顺序查找算法的时间复杂度为  $O(n)$ 。




## 方法2：二分查找 (Binary Search)

- ❖ 假设n个数据元素的关键字满足有序（比如：小到大）

$$k_1 < k_2 < \dots < k_n$$

并且是连续存放（数组），那么可以进行二分查找。

**[例]** 假设有**13**个数据元素，按关键字由小到大顺序存放。  
二分查找关键字为**444**的数据元素过程如下：

|   |    |    |    |    |    |   |     |     |     |     |   |     |
|---|----|----|----|----|----|---|-----|-----|-----|-----|---|-----|
| 5   | 16 | 39 | 45 | 51 | 98 | 100   | 202 | 226 | 321 | 368 | 444   | 501 |
| 1   | 2  | 3  | 4  | 5  | 6  | 7   | 8   | 9   | 10  | 11  | 12  | 13  |
|  |    |    |    |    |    |  |     |     |     |     |  |     |
| left  |    |    |    |    |    | mid   |     |     |     |     | right   |     |

- 1、  $\text{left} = 1, \text{right} = 13; \text{mid} = (1+13)/2 = 7:$   **$100 < 444;$**
- 2、  $\text{left} = \text{mid}+1=8, \text{right} = 13; \text{mid} = (8+13)/2 = 10:$   **$321 < 444;$**
- 3、  $\text{left} = \text{mid}+1=11, \text{right} = 13; \text{mid} = (11+13)/2 = 12:$  **查找结束;**

**[例]** 仍然以上面13个数据元素构成的有序线性表为例  
二分查找关键字为 **43** 的数据元素如下：

|   |    |    |    |    |    |     |     |     |     |     |     |     |
|---|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|
| 5 | 16 | 39 | 45 | 51 | 98 | 100 | 202 | 226 | 321 | 368 | 444 | 501 |
| 1 | 2  | 3  | 4  | 5  | 6  | 7   | 8   | 9   | 10  | 11  | 12  | 13  |



left



mid



right

- 1、  $\text{left} = 1, \text{right} = 13; \text{mid} = (1+13)/2 = 7$ :  **$100 > 43$** ;
- 2、  $\text{left} = 1, \text{right} = \text{mid}-1 = 6; \text{mid} = (1+6)/2 = 3$ :  **$39 < 43$** ;
- 3、  $\text{left} = \text{mid}+1 = 4, \text{right} = 6; \text{mid} = (4+6)/2 = 5$ :  **$51 > 43$** ;
- 4、  $\text{left} = 4, \text{right} = \text{mid}-1 = 4; \text{mid} = (4+4)/2 = 4$ :  **$45 > 43$** ;
- 5、  $\text{left} = 4, \text{right} = \text{mid}-1 = 3$ ;  **$\text{left} > \text{right} ?$**  查找失败，结束;



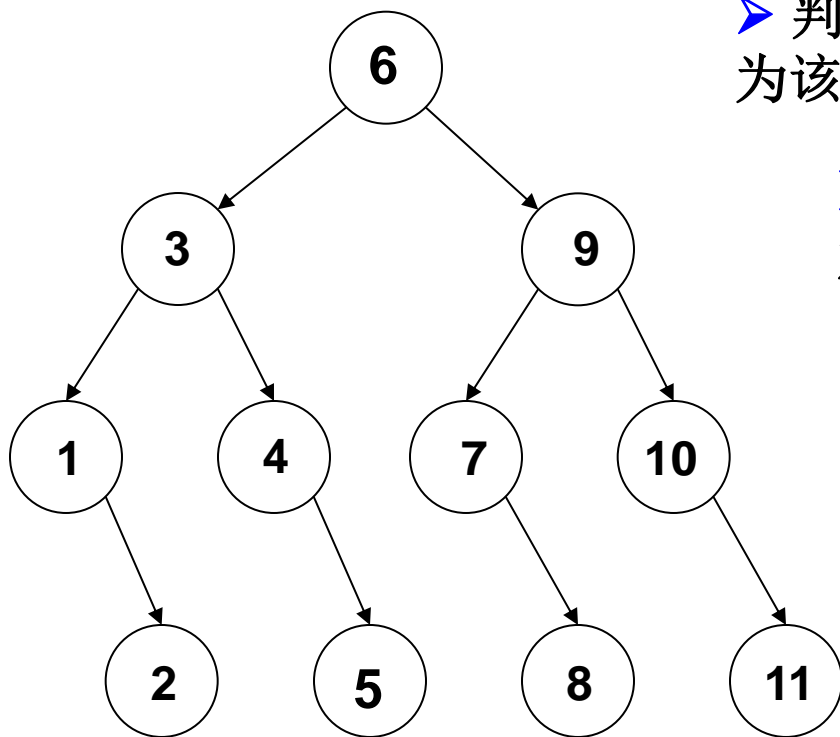
## 二分查找算法

```
int BinarySearch ( StaticTable * Tbl, ElementType K)
{ /*在表Tbl中查找关键字为K的数据元素*/
    int left, right, mid, NotFound=-1;

    left = 1;                /*初始左边界*/
    right = Tbl->Length;     /*初始右边界*/
    while ( left <= right )
    {
        mid = (left+right)/2; /*计算中间元素坐标*/
        if( K < Tbl->Element[mid])    right = mid-1; /*调整右边界*/
        else if( K > Tbl->Element[mid]) left = mid+1; /*调整左边界*/
        else return mid; /*查找成功，返回数据元素的下标*/
    }
    return NotFound; /*查找不成功，返回-1*/
}
```

□ 二分查找算法具有对数的时间复杂度 $O(\log N)$

## ❖ 11个元素的二分查找判定树



➤ 判定树上每个结点需要的查找次数刚好为该结点所在的层数;

➤ 查找成功时查找次数不会超过判定树的深度

➤  $n$ 个结点的判定树的深度为 $\lceil \log_2 n \rceil + 1$ .

➤  $ASL = (4*4 + 4*3 + 2*2 + 1)/11 = 3$

二分查找的启示?

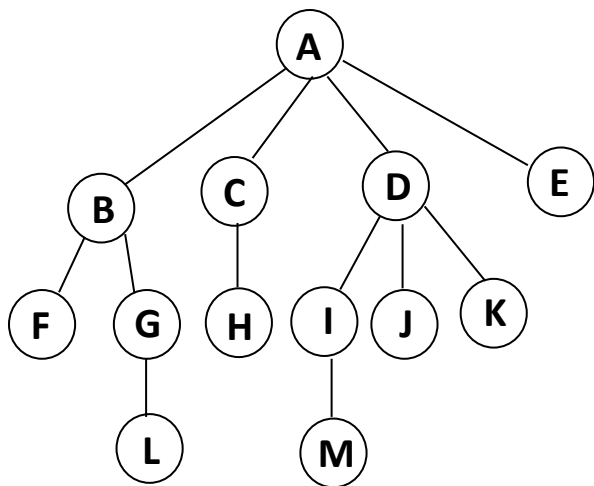
# 树的定义

**树 (Tree)** :  $n$  ( $n \geq 0$ ) 个结点构成的有限集合。

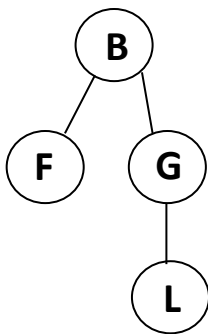
当  $n=0$  时, 称为**空树**;

对于任一棵**非空树** ( $n > 0$ ), 它具备以下性质:

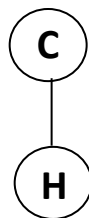
- 树中有一个称为“**根 (Root)**”的特殊结点, 用  $r$  表示;
- 其余结点可分为  $m$  ( $m > 0$ ) 个**互不相交的**有限集  $T_1, T_2, \dots, T_m$ , 其中每个集合本身又是一棵树, 称为原来树的“**子树 (SubTree)**”



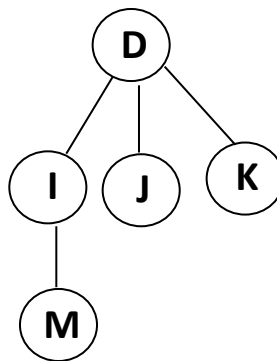
(a) 树  $T$



(b) 子树  $T_{A1}$



(c) 子树  $T_{A2}$

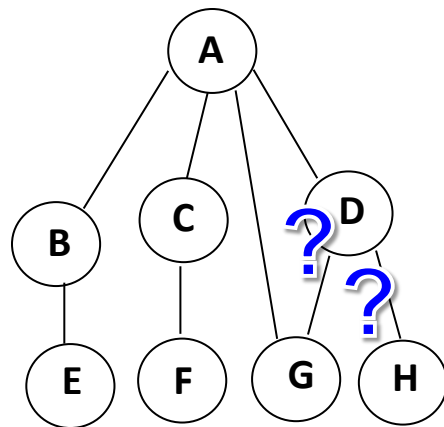
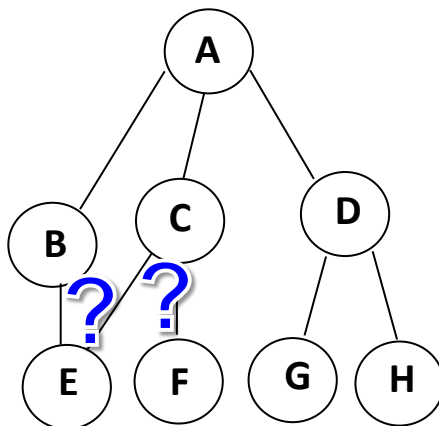
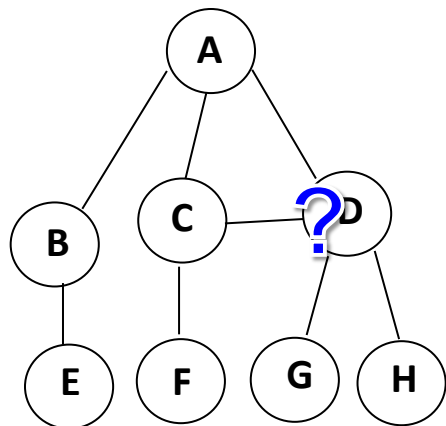


(d) 子树  $T_{A3}$

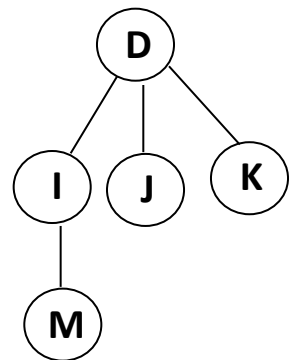


(e) 子树  $T_{A4}$

## ❖ 树与非树？

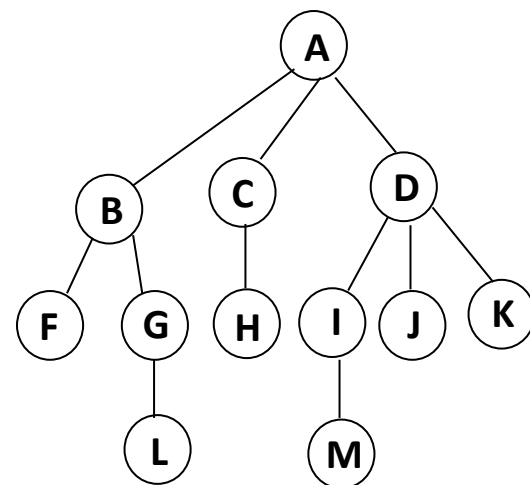


- 子树是**不相交**的；
- 除了根结点外，**每个结点有且仅有一个父结点**；
- 一棵**N**个结点的树有**N-1**条边。



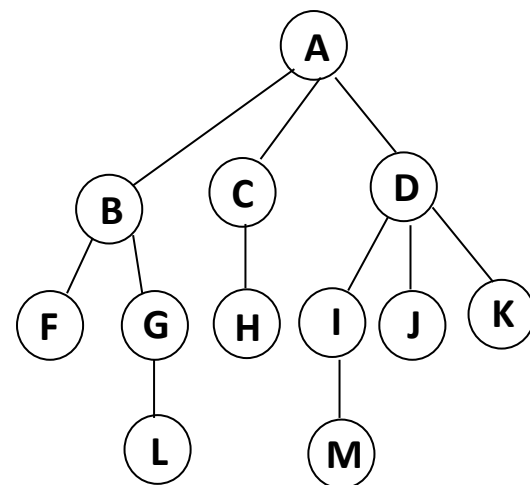
## ❖ 树的一些基本术语

1. 结点的度（Degree）：结点的子树个数
2. 树的度：树的所有结点中最大的度数
3. 叶结点（Leaf）：度为0的结点
4. 父结点（Parent）：有子树的结点是其子树的根结点的父结点
5. 子结点（Child）：若A结点是B结点的父结点，则称B结点是A结点的子结点；子结点也称孩子结点。
6. 兄弟结点（Sibling）：具有同一父结点的各结点彼此是兄弟结点。

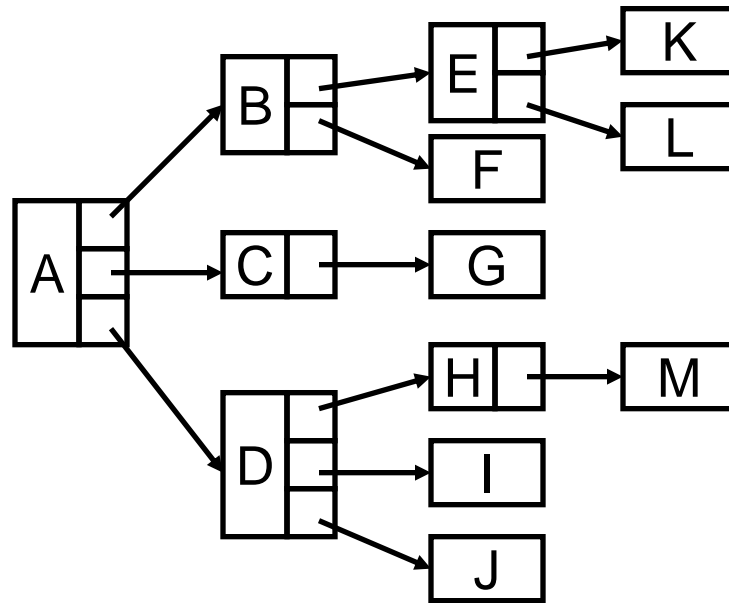
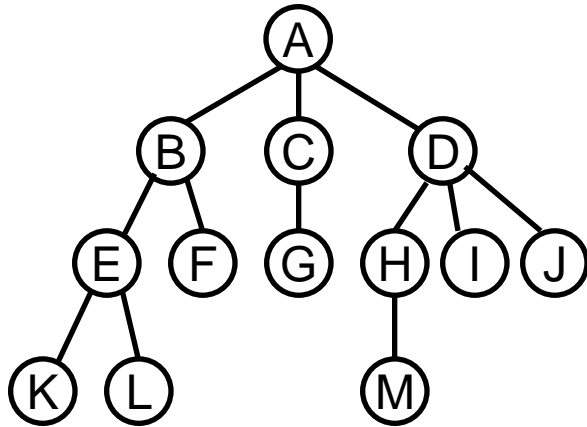


## ❖ 树的一些基本术语

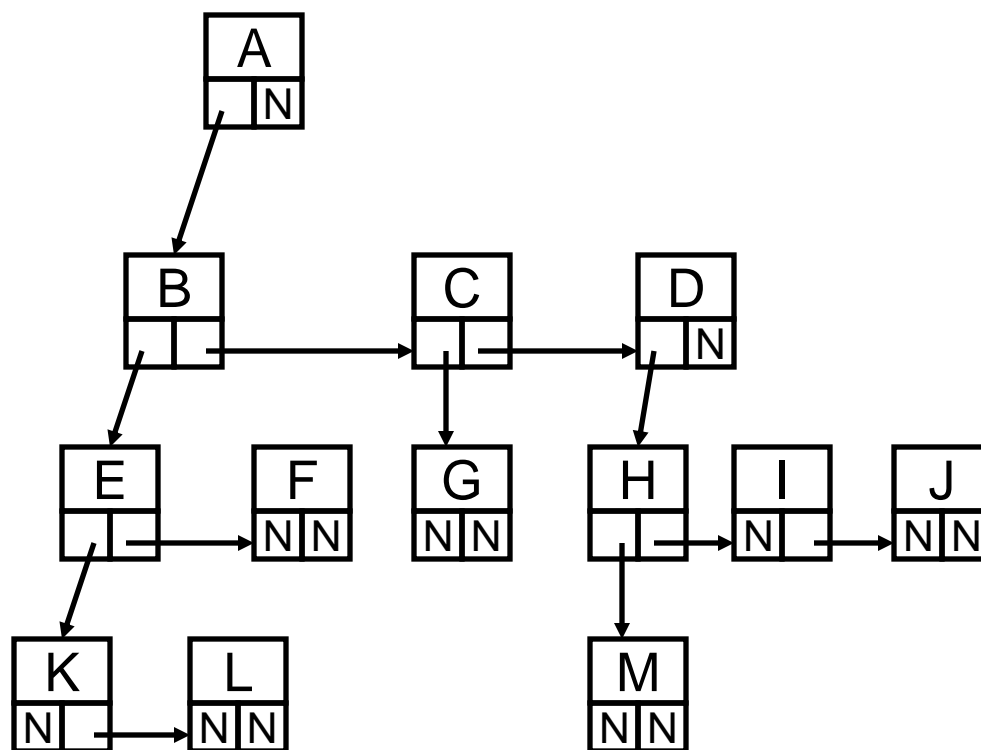
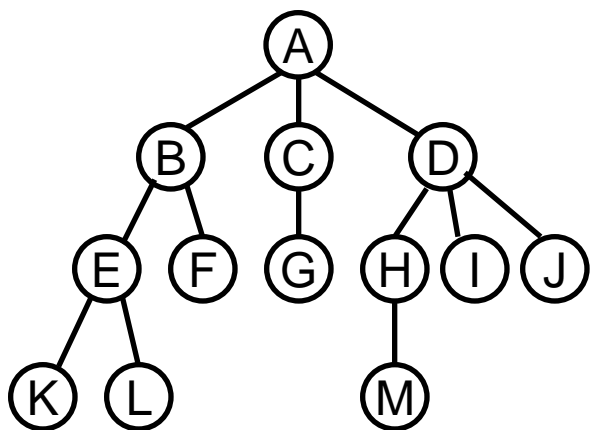
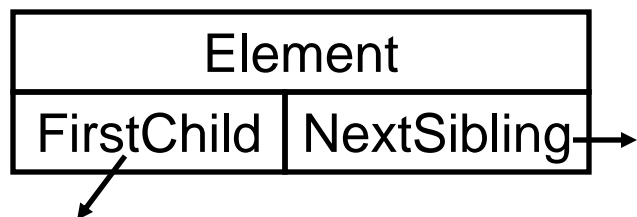
- 7. **路径和路径长度**: 从结点 $n_1$ 到 $n_k$ 的**路径**为一个结点序列 $n_1, n_2, \dots, n_k$ ,  $n_i$ 是 $n_{i+1}$ 的父结点。路径所包含边的个数为**路径的长度**。
- 9. **祖先结点(Ancessor)**: 沿**树根到某一结点路径**上的所有结点都是这个结点的祖先结点。
- 10. **子孙结点(Descendant)**: 某一结点的**子树**中的所有**结点**是这个结点的子孙。
- 11. **结点的层次 (Level)**: 规定**根结点在1层**, 其它任一结点的层数是其父结点的层数加1。
- 12. **树的深度 (Depth)**: 树中所有结点中的**最大层次**是这棵树的深度。



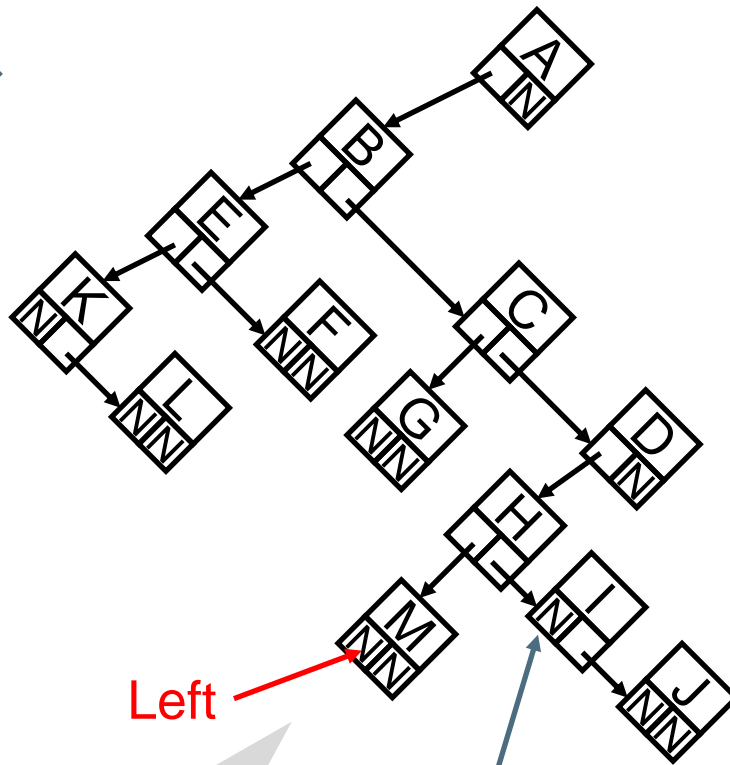
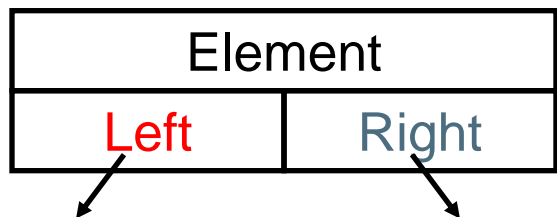
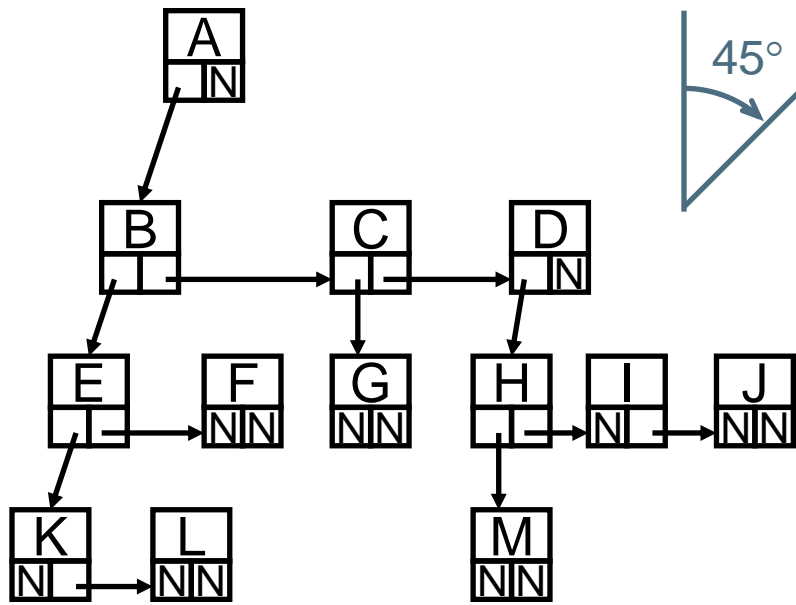
# 树的表示



## ❖ 儿子-兄弟表示法







Left

Right

二叉树