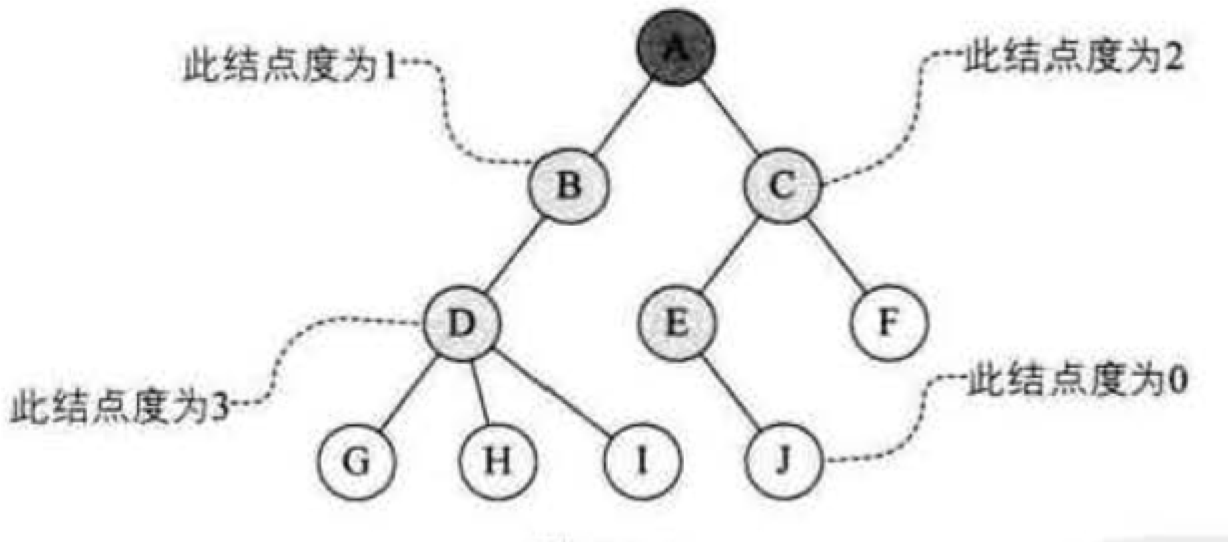
一、树

1. 树的相关概念

1-1. 结点

树的结点包含一个数据元素及若干指向其子树的分支。结点拥有的子树数称为结点的度。度为0的结点称为叶结点或终端结点；度不为0的结点称为非终端结点或分支结点。除根结点之外，分支结点也称为内部结点。树的度是树内各结点的度的最大值



2. 二叉树的定义

二叉树特点：

- 每个结点最多有两棵子树，所以二叉树中不存在度大于2的结点
- 左子树和右子树是有顺序的，次序不能任意颠倒
- 即使树中某结点之右一棵子树，也需要区分是左子树还是右子树

2-1. 特殊二叉树

2-1-1. 斜树

所有的结点都只有左子树的二叉树叫左斜树，所有结点都只有右子树的二叉树叫右斜树，这两者统称为斜树。斜树有个很明显的特点，就是每一层都只有一个结点，结点的个数与二叉树的深度相同。其实斜树相当于线性表，而线性表结构可以理解为树的的一种特殊形式。

2-1-2. 满二叉树

在一棵二叉树中，如果所有的分支结点都存在左子树和右子树，并且所有叶子都在同一层，这样的二叉树称为满二叉树。
满二叉树特点：

- 叶子只能出现在最下一层
- 非叶子结点的度一定是2
- 在同样深度的二叉树中，满二叉树的结点个数最多，叶子树最多

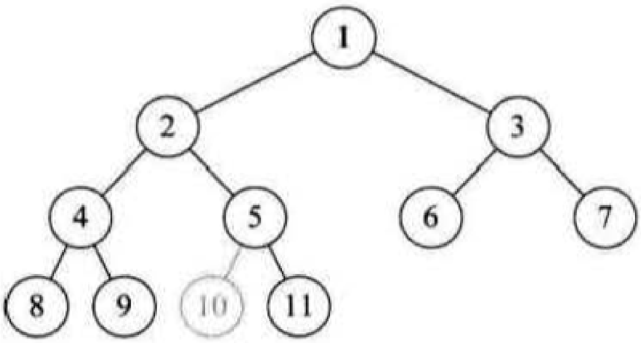
2-1-3. 完全二叉树

对一棵具有n个结点的二叉树按层序编号，如果编号为i的结点与同样深度的满二叉树编号为i的结点在二叉树中的位置相同，则这棵树称为完全二叉树。
完全二叉树特点：

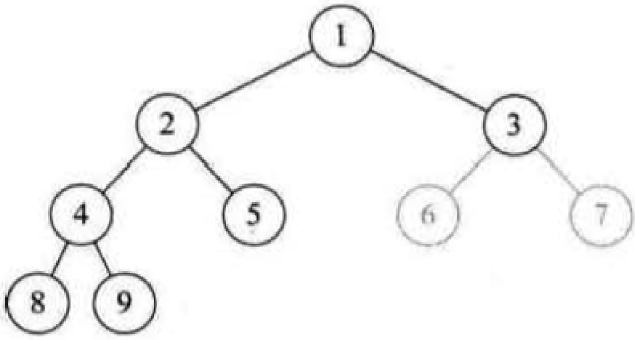
- 叶子结点只能出现在最下两层
- 最下层的叶子一定集中在左部连续位置
- 倒数二层，若有叶子结点，一定都在右部连续位置
- 如果结点度为1，则该结点只有左孩子，即不存在只有右子树的情况
- 同样结点数的二叉树，完全二叉树的深度最小

判别完全二叉树的方法:看着书的示意图，心中默默给每个结点按照满二叉树的结构逐层顺序编号，如果编号出现空档，就说明不是完全二叉树，否则就是。

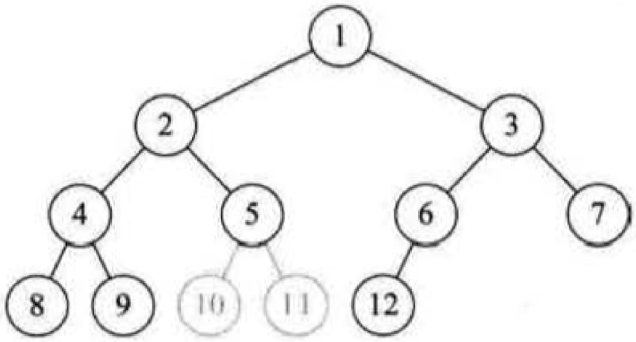
其次，完全二叉树的所有结点与同样深度的满二叉树，它们按层序编号相同的结点，是一一对应的。这里有个关键词是**按层序编号**，像图 6-5-7 中的树 1，因为 5 结点没有左子树，却有右子树，那就使得按层序编号的第 10 个编号空档了。同样道理，图 6-5-7 中的树 2，由于 3 结点没有子树，所以使得 6、7 编号的位置空档了。图 6-5-7 中的树 3 又是因为 5 编号下没有子树造成第 10 和第 11 位置空档。只有图 6-5-6 中的树，尽管它不是满二叉树，但是编号是连续的，所以它是完全二叉树。



树1



树2



树3

2-2. 二叉树性质

2-2-1. 性质1

在二叉树的第i层上至多有2^{i-1}个结点

2-2-2. 性质2

深度为k的二叉树至多有2^k-1个结点

2-2-3. 性质3

对任何一棵二叉树T，如果叶子结点数为n_0，度为2的结点数为n_2，则$n_0=n_2+1$(完全二叉树下$n_0=n_2+1$，N_1 = 1 或者 0)

2-2-4. 性质4

具有n个结点的完全二叉树的深度为$[\log_{2N}]+1$([x]表示不大于x的最大的整数)

2-2-5. 性质5

如果对一棵具有n个结点的完全二叉树(其深度为$[\log_{2N}+1]$)的结点按层序编号(从第1层到第$[\log_{2N}]+1$层，每层从左到右，对任一结点(1小于等于，1大于等于N))

1. 如果i = 1，则结点i是二叉树的根，无双亲；如果i > 1,则其双亲是结点[i/2]
2. 如果2*i > n,则结点i无左孩子(结点i为叶子结点)；否则其左孩子是节点2i
3. 如果2*i + 1> n,则结点i无右孩子(结点i为叶子结点)；否则其右孩子是节点2i+1

2-2-6. 性质6

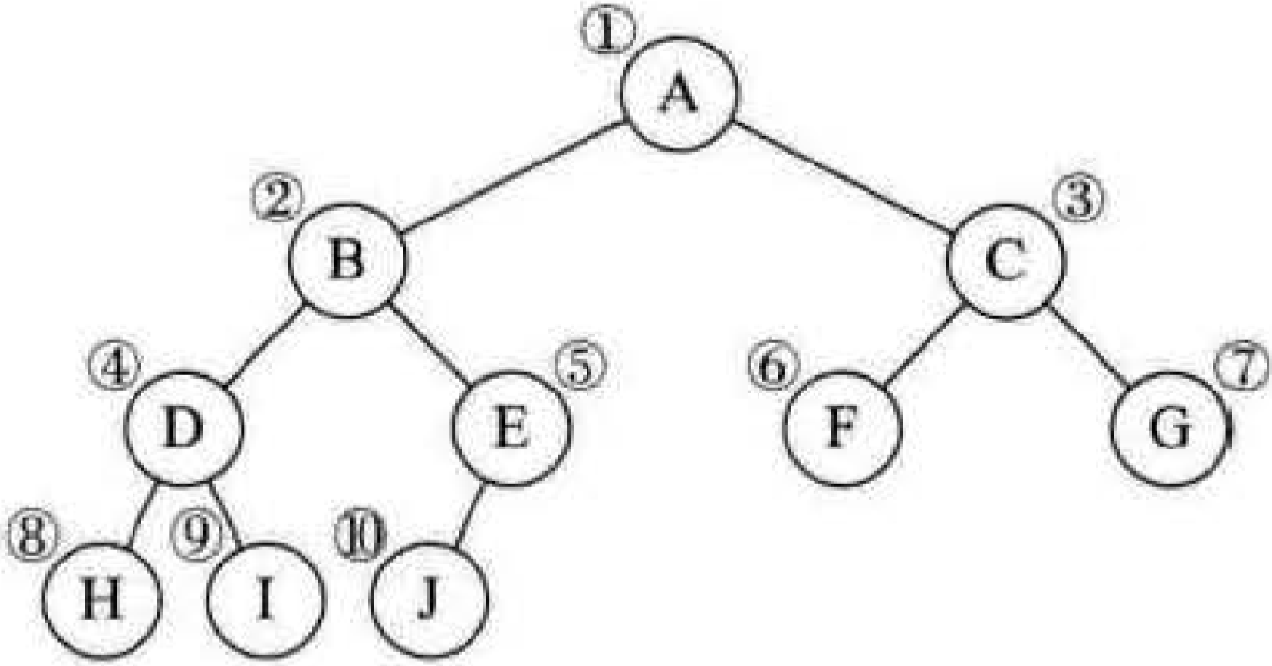
含有n个结点的不相似的二叉树有：卡特兰数--C(n)=(1/(n+1))*((2n)!/(n!n!))*
*比如含有三个结点的树：C(3) = (2*3)!/(3！*3！)/(3+1)=5

2-2-7. 性质7

m个结点有有m-1个非空指针，其余皆为空指针

2-3. 二叉树顺序存储结构

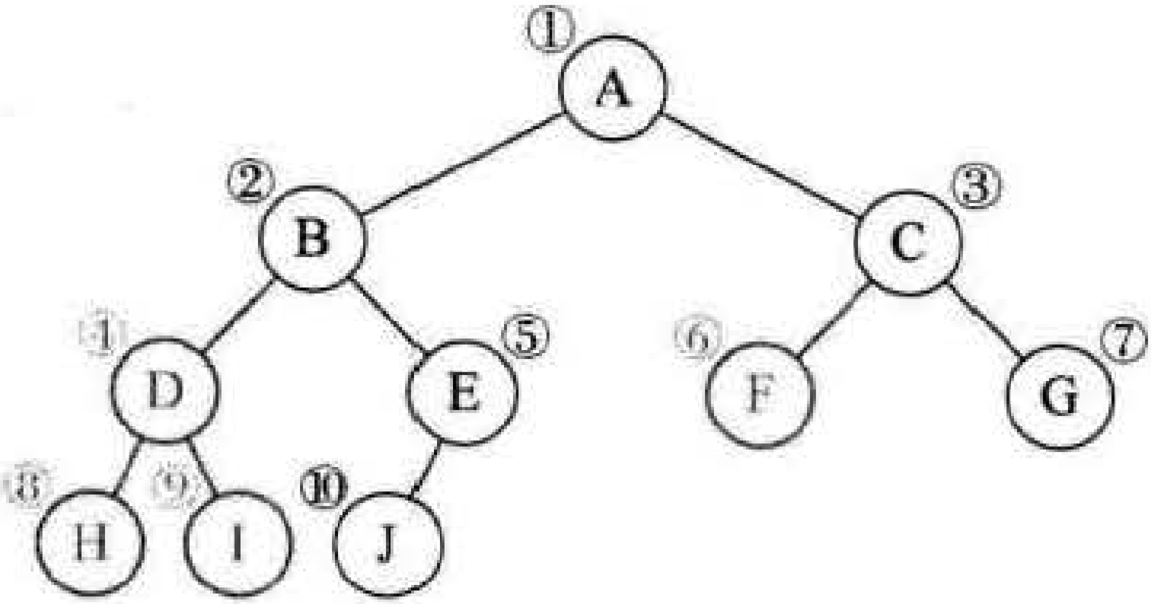
二叉树的顺序存储结构就是用一维数组存储二叉树中的结点，并且存储的位置，即数组下标要能体现结点之间的逻辑关系
完全二叉树:



对应数组下标：

下标:	1	2	3	4	5	6	7	8	9	10
	A	B	C	D	E	F	G	H	I	J

完全二叉树的优越性就体现在此，由于严格的定义，因此用顺序结构也可以表现出二叉树的结构。对于一般的二叉树，层序编号不能反映逻辑关系，但是可以将其按完全二叉树编号，将不存在的结点设置为“^”。



下标:	1	2	3	4	5	6	7	8	9	10
	A	B	C	∧	E	∧	G	∧	∧	J

(浅色代表不存在)

考虑一种极端的情况，一棵深度为k的右斜树，它只有k个结点，却需要分配2^{n-1}个存储单元空间，这显然是对存储空间的浪费。因此，顺序存储结构只用于完全二叉树

2-4. 二叉链表

//TODO

2-5. 遍历二叉树

2-5-1. 二叉树遍历原理

二叉树的遍历是指从根结点出发，按照某种次序依次访问二叉树中所有结点，使得每个结点被访问一次且仅被访问一次
访问是一种抽象操作，根据实际需要来确定做什么。在这里i我们可以简单地假定就是输出结点的数据信息。

广度优先(BFS)

- 层次遍历

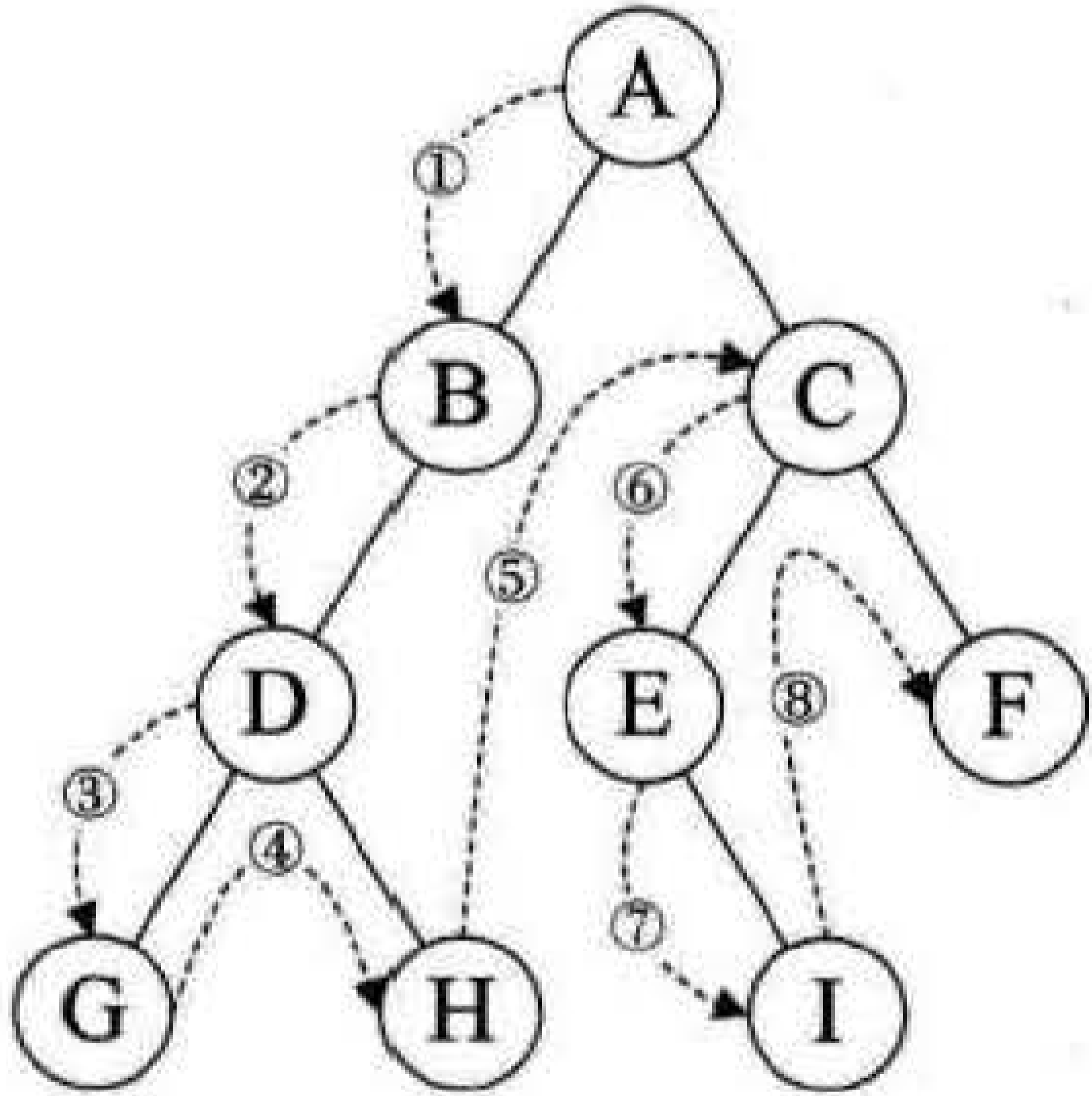
深度优先(DFS)

- 前序遍历
- 中序遍历
- 后序遍历

2-5-2. 二叉树遍历方法

2-5-2-1. 前序遍历

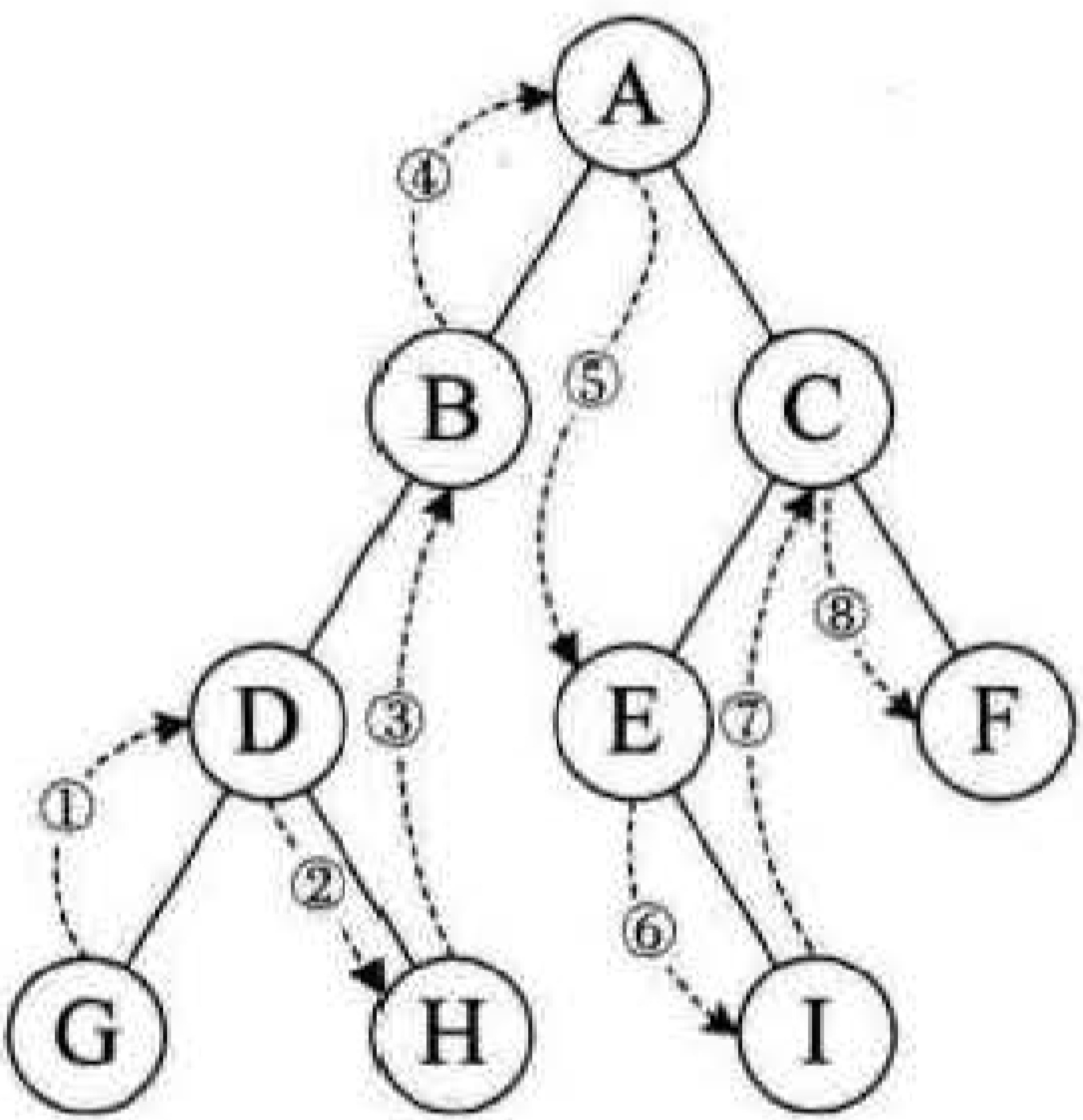
规则是若二叉树为空，则空操作返回，否则先访问根结点，然后前序遍历左子树，再前序遍历右子树




```
/**
     * 先序遍历
     * @param root
     */
    public static void preOrderTravle(TreeNode root){
        if(root == null){
            return;
        }
        root.display();
        preOrderTravle(root.leftChild);
        preOrderTravle(root.rightChild);
    }
```

2-5-2-2. 中序遍历

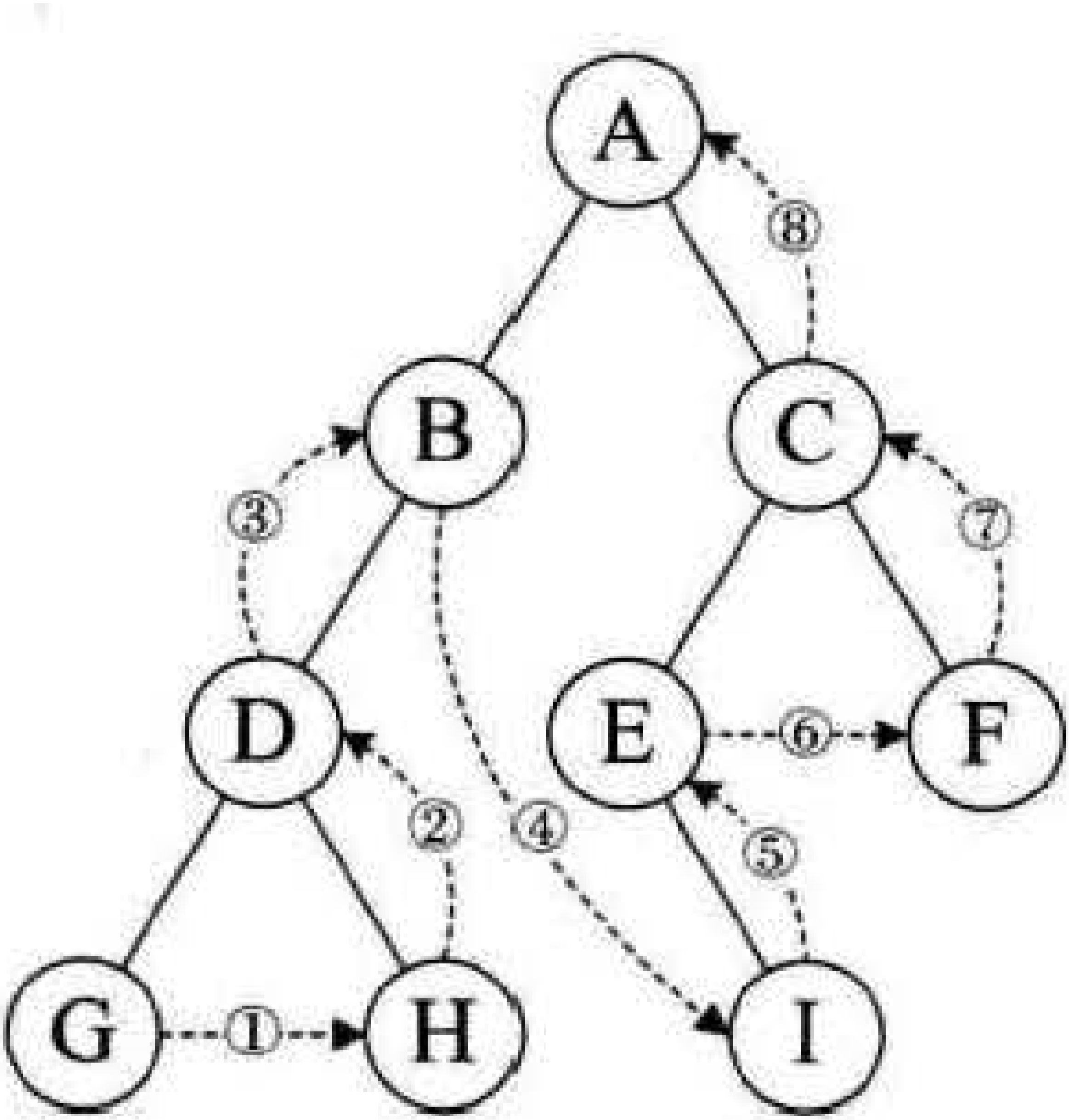
规则是若树为空，则空操作返回，否则从根结点开始(注意不是先访问根结点),中序遍历根结点的左子树，然后是访问根结点，最后中序遍历右子树




```
/**
     * 中序遍历
     */
    public static void inOrderTravle(TreeNode root){
        if(root == null){
            return;
        }
        inOrderTravle(root.leftChild);
        root.display();
        inOrderTravle(root.rightChild);
    }
```

2-5-2-3. 后序遍历

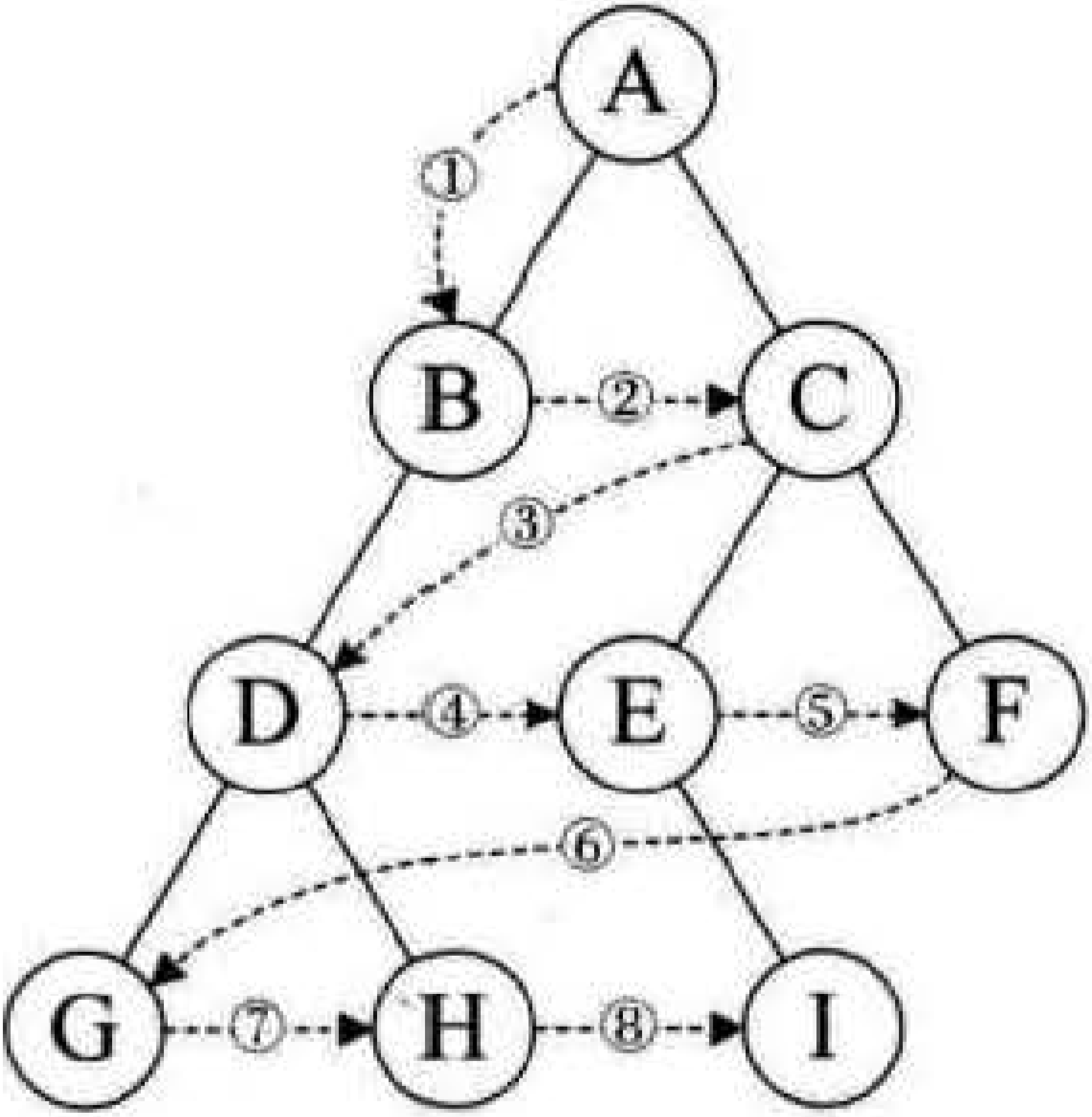
规则是若树为空，则空操作返回，否则从左到右先叶子后结点的方式访问左右子树，最后是访问根结点




```
/**
     * 后序遍历
     */
    public static void afterOrderTravle(TreeNode root){
        if(root == null){
            return;
        }
        afterOrderTravle(root.leftChild);
        afterOrderTravle(root.rightChild);
        root.display();
    }
```

2-5-2-4. 层序遍历

规则是若树为空，则空操作返回，否则从树的第一层，也就是根节点开始访问，从上而下逐层遍历，在同一层中，按从左到右的顺序对结点逐个访问

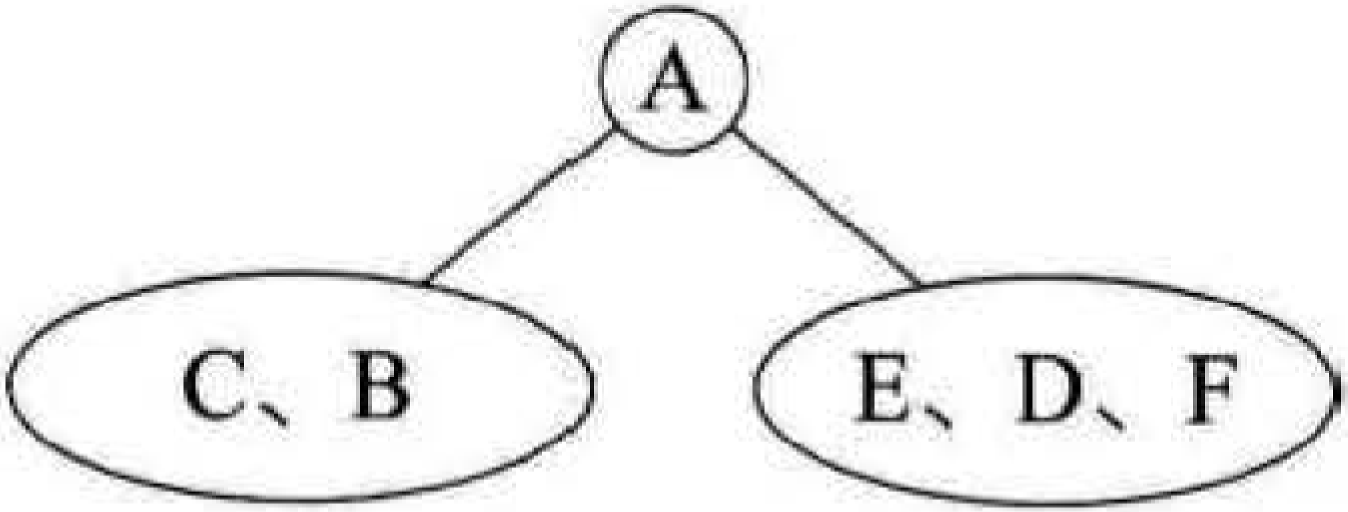


2-5-3. 推导遍历结果

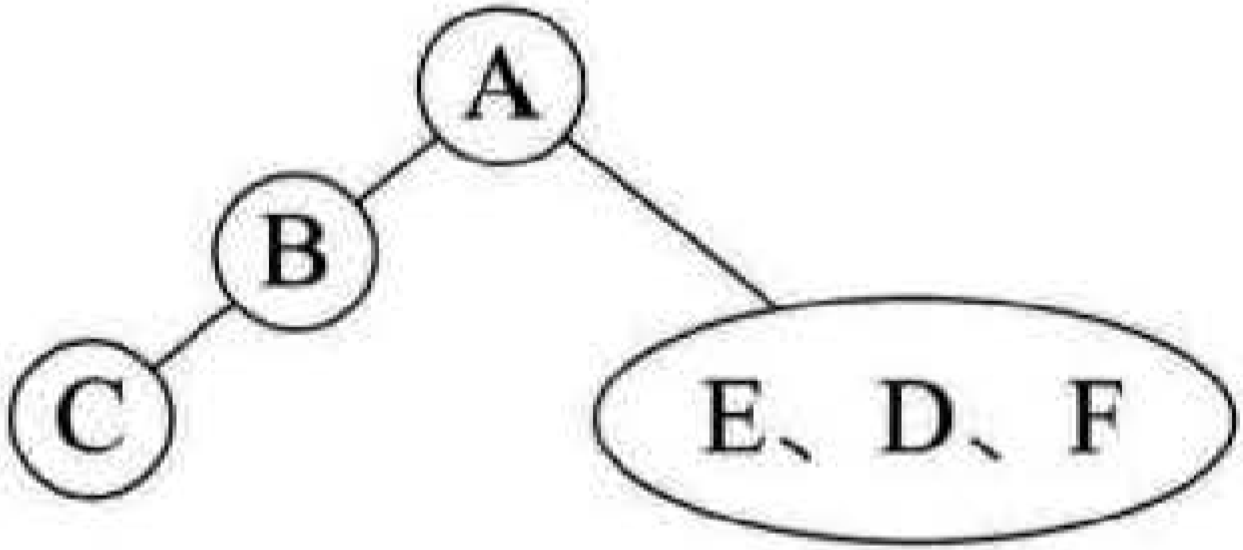
有时候我们会碰到这种题目，给定一棵二叉树的前序遍历和中序遍历，求其后序遍历。对于这样的题目，我们首先得了解前序遍历和中序遍历的原理，对于三种遍历而言，前序遍历是先打印根结点再递归调用左子树和右子树，中序是递归调用左子树到根再到右子树，后序是左子树到右子树再到根结点(遍历中把每一棵子树看作新的一棵树，这样就好理解很多)。

2-5-3-1. 题目解析

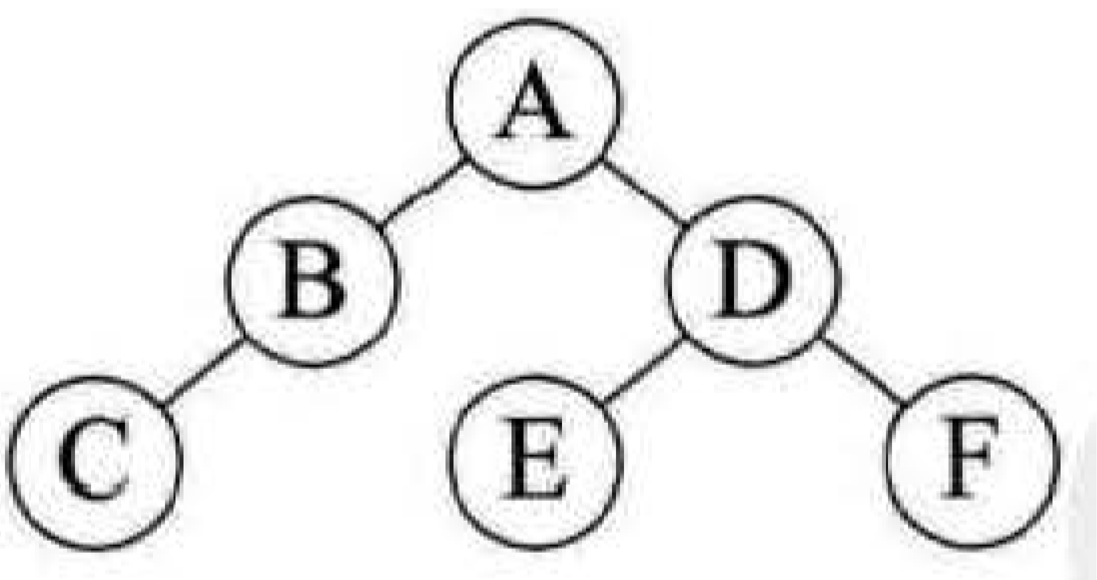
已知一棵二叉树的前序遍历序列为ABCDEF，中序遍历序列为CBAEDF，请问这棵树的后序遍历结果。
前序序列为ABCDEF，第一个字母是A被打印出来，说明A是根结点的数据。再由中序序列是CBAEDF，可以知道C和B是A的左子树结点，E、D、F是A的右子树的结点。



之后看前序中的C和B，它的顺序是ABCDEF，是先打印B后打印C，所以可以确定B是A的左孩子，而C就只能是B的孩子，究竟是左孩子还是右孩子此时还不确定。再看中序序列是CBAEDF，C是在B的前面打印的，这就说明C是B的左孩子



再看前序中的E、D、F，它的顺序是ABCDEF，那就意味着D是A的右孩子，E和F是D的子孙，注意，它们中有一个不一定是孩子的，还有可能是孙子。再看中序序列CBAEDF，由于E在D的左侧，而F在D的右侧，所以可以确定E是D的左孩子，F是D的右孩子，因此最终得到的二叉树是：



- 已知前序遍历序列和中序遍历序列，可以唯一确定一棵二叉树
- 已知后序遍历序列和中序遍历序列，可以唯一确定一棵二叉树

2-6. 森林与二叉树的转换

2-7. 赫尔曼树

赫尔曼树

2-8. 有向树

2-9. 线索树