

# 二叉树

---

## 【本节目标】

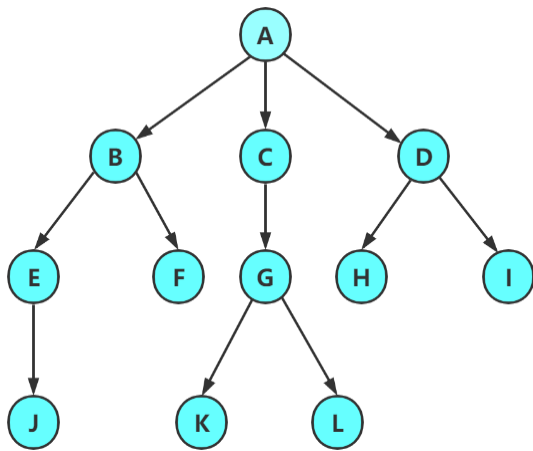
1. 掌握树的基本概念
2. 掌握二叉树概念及特性
3. 掌握二叉树的基本操作
4. 完成二叉树相关的面试题练习

## 1. 树型结构（了解）

### 1.1 概念

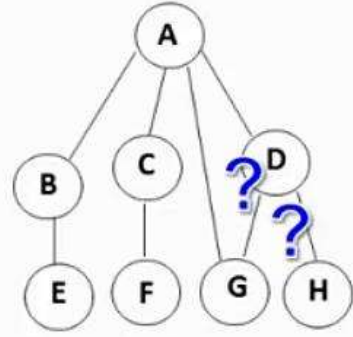
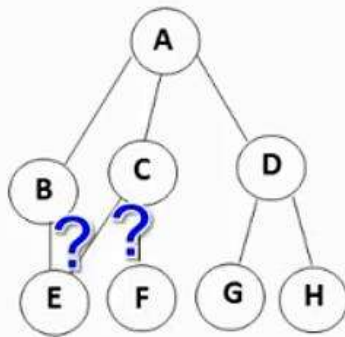
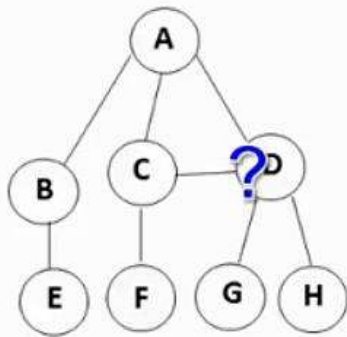
树是一种**非线性**的数据结构，它是由 $n$  ( $n \geq 0$ ) 个有限结点组成一个具有层次关系的集合。**把它叫做树是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。**它具有以下的特点：

- 有一个特殊的结点，称为根结点，根结点没有前驱结点
- 除根结点外，其余结点被分成 $M$  ( $M > 0$ ) 个互不相交的集合 $T_1$ 、 $T_2$ 、.....、 $T_m$ ，其中每一个集合 $T_i$  ( $1 \leq i \leq m$ ) 又是一棵与树类似的子树。每棵子树的根结点有且只有一个前驱，可以有0个或多个后继
- 树是递归定义的。

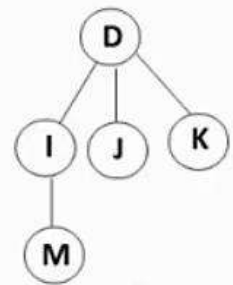


注意：树形结构中，子树之间不能有交集，否则就不是树形结构

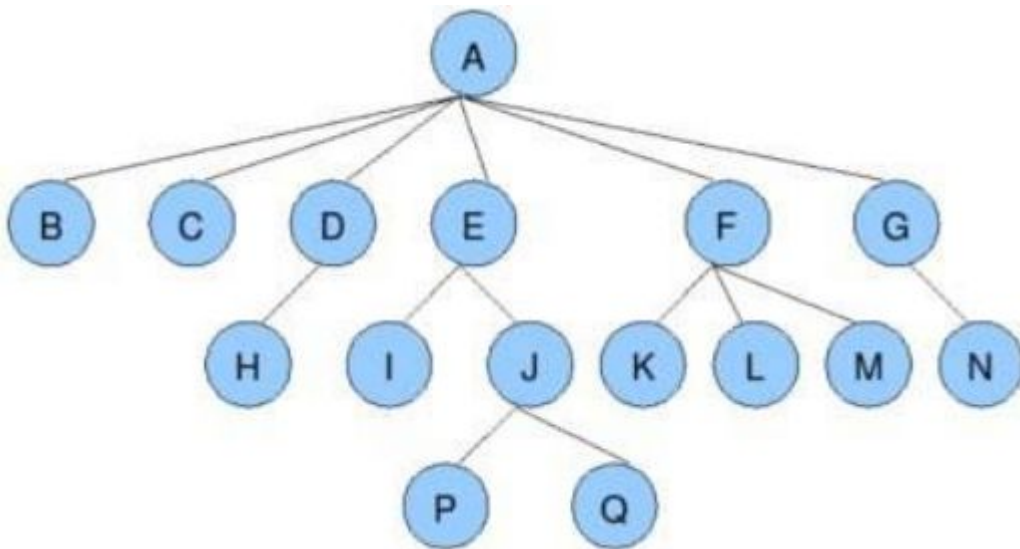
## ❖ 树与非树？



- 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。



## 1.2 概念 (重要)



**结点的度：**一个结点含有子树的个数称为该结点的度；如上图：A的度为6

**树的度：**一棵树中，所有结点度的最大值称为树的度；如上图：树的度为6

**叶子结点或终端结点：**度为0的结点称为叶结点；如上图：B、C、H、I...等节点为叶结点

**双亲结点或父结点：**若一个结点含有子结点，则这个结点称为其子结点的父结点；如上图：A是B的父结点

**孩子结点或子结点：**一个结点含有的子树的根结点称为该结点的子结点；如上图：B是A的孩子结点

**根结点：**一棵树中，没有双亲结点的结点；如上图：A

**结点的层次：**从根开始定义起，根为第1层，根的子结点为第2层，以此类推

**树的高度或深度**：树中结点的最大层次；如上图：树的高度为4

树的以下概念只需了解，在看书时只要知道是什么意思即可：

**非终端结点或分支结点**：度不为0的结点；如上图：D、E、F、G...等节点为分支结点

**兄弟结点**：具有相同父结点的结点互称为兄弟结点；如上图：B、C是兄弟结点

**堂兄弟结点**：双亲在同一层的结点互为堂兄弟；如上图：H、I互为兄弟结点

**结点的祖先**：从根到该结点所经分支上的所有结点；如上图：A是所有结点的祖先

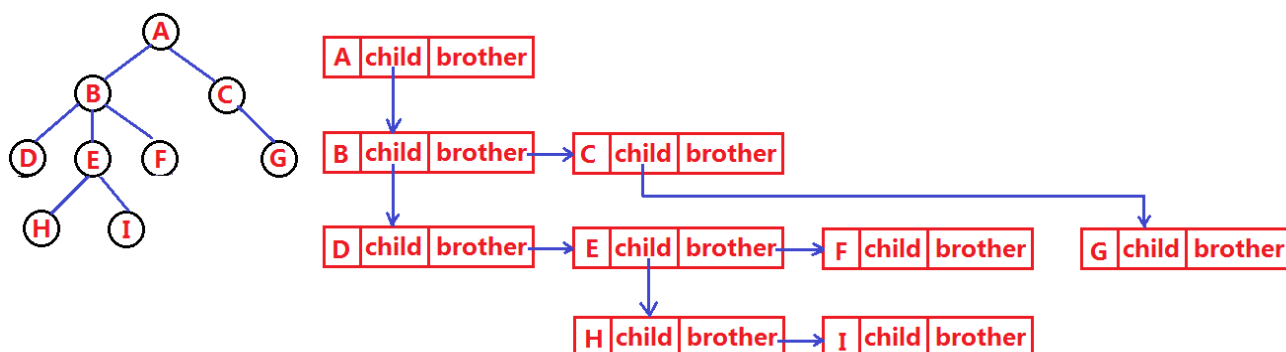
**子孙**：以某结点为根的子树中任一结点都称为该结点的子孙。如上图：所有结点都是A的子孙

**森林**：由m ( $m \geq 0$ ) 棵互不相交的树组成的集合称为森林

### 1.3 树的表示形式（了解）

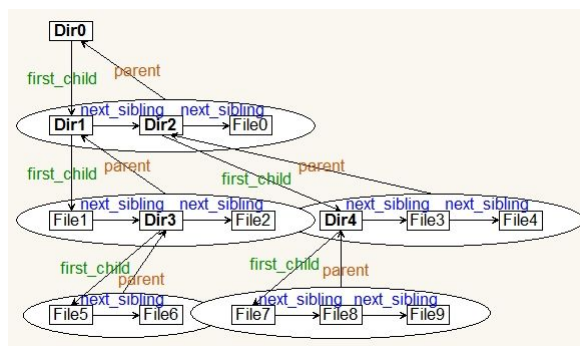
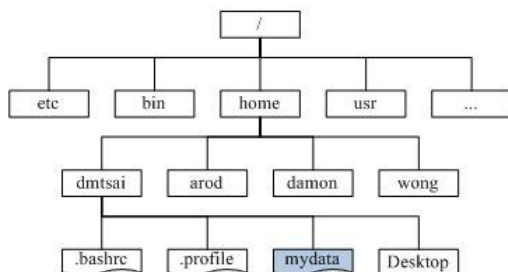
树结构相对线性表就比较复杂了，要存储表示起来就比较麻烦了，实际中树有很多种表示方式，如：**双亲表示法**，**孩子表示法**、**孩子双亲表示法**、**孩子兄弟表示法**等等。我们这里就简单的了解其中最常用的**孩子兄弟表示法**。

```
class Node {  
    int value;           // 树中存储的数据  
    Node firstChild;     // 第一个孩子引用  
    Node nextBrother;    // 下一个兄弟引用  
}
```



### 1.4 树的应用

文件系统管理（目录和文件）

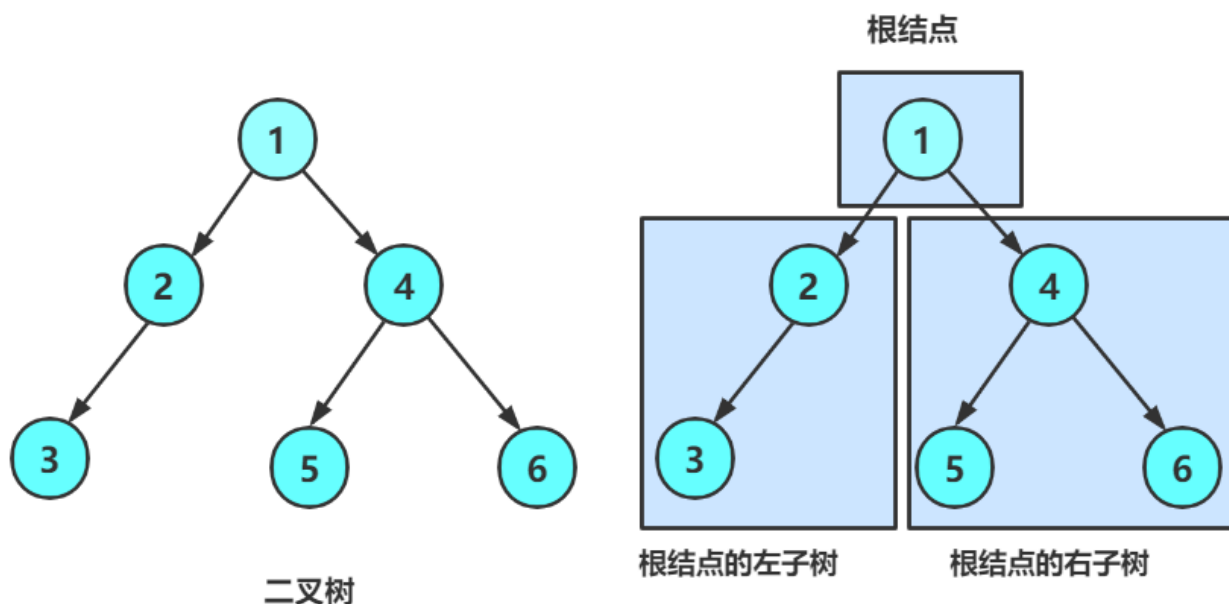


## 2. 二叉树（重点）

### 2.1 概念

一棵二叉树是结点的一个有限集合，该集合：

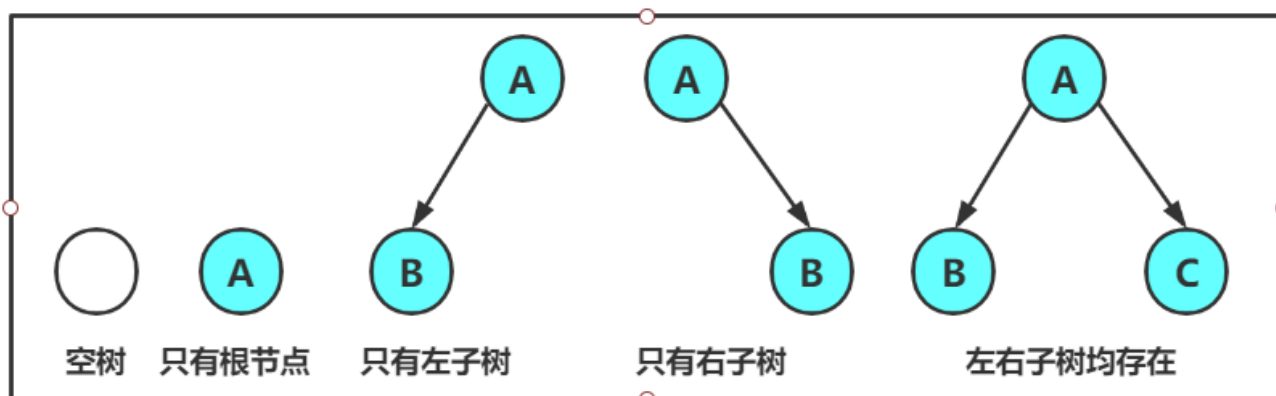
1. 或者为空
2. 或者是由一个根节点加上两棵别称为左子树和右子树的二叉树组成。



从上图可以看出：

1. 二叉树不存在度大于2的结点
2. 二叉树的子树有左右之分，次序不能颠倒，因此二叉树是有序树

注意：对于任意的二叉树都是由以下几种情况复合而成的：



大自然的奇观：

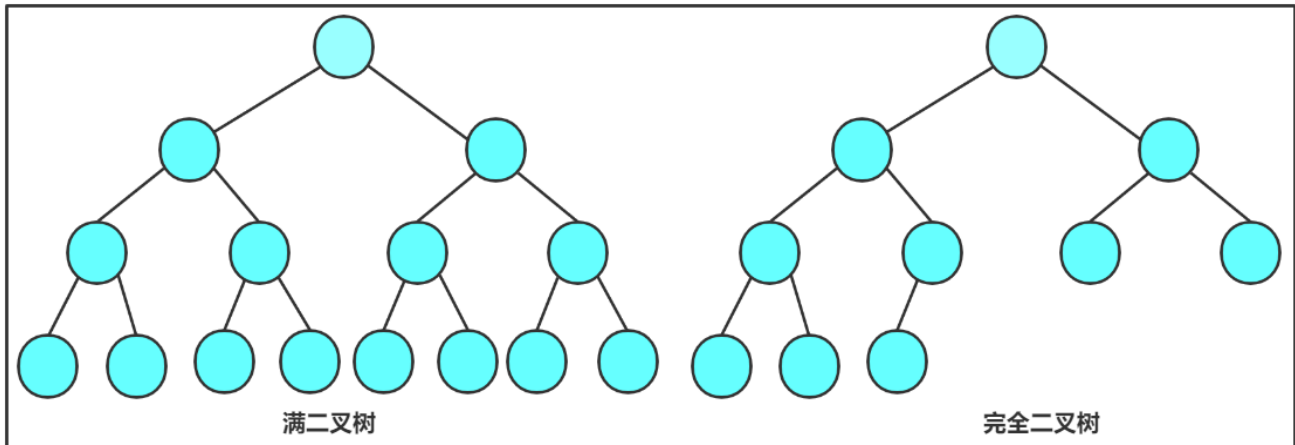




## 2.2 两种特殊的二叉树

1. **满二叉树**: 一棵二叉树，如果每层的结点数都达到最大值，则这棵二叉树就是满二叉树。也就是说，如果一棵二叉树的层数为 $K$ ，且结点总数是 $2^k - 1$ ，则它就是满二叉树。
2. **完全二叉树**: 完全二叉树是效率很高的数据结构，完全二叉树是由满二叉树而引出来的。对于深度为 $K$ 的，有 $n$ 个结点的二叉树，当且仅当其每一个结点都与深度为 $K$ 的满二叉树中编号从 $0$ 至 $n-1$ 的结点——对应时称之为完

全二叉树。要注意的是满二叉树是一种特殊的完全二叉树。



## 2.3 二叉树的性质

1. 若规定根结点的层数为1, 则一棵非空二叉树的第 $i$ 层上最多有 $2^{i-1}$  ( $i>0$ )个结点
2. 若规定只有根结点的二叉树的深度为1, 则深度为 $K$ 的二叉树的最大结点数是 $2^k - 1$  ( $k \geq 0$ )
3. 对任何一棵二叉树, 如果其叶结点个数为  $n_0$ , 度为2的非叶结点个数为  $n_2$ , 则有  $n_0 = n_2 + 1$
4. 具有 $n$ 个结点的完全二叉树的深度 $k$ 为 $\log_2(n+1)$ 上取整
5. 对于具有 $n$ 个结点的完全二叉树, 如果按照从上至下从左至右的顺序对所有节点从0开始编号, 则对于序号为 $i$ 的结点有:
  - 若 $i>0$ , 双亲序号:  $(i-1)/2$ ;  $i=0$ ,  $i$ 为根结点编号, 无双亲结点
  - 若 $2i+1 < n$ , 左孩子序号:  $2i+1$ , 否则无左孩子
  - 若 $2i+2 < n$ , 右孩子序号:  $2i+2$ , 否则无右孩子

1. 某二叉树共有 399 个结点, 其中有 199 个度为 2 的结点, 则该二叉树中的叶子结点数为 ( )

A 不存在这样的二叉树

B 200

C 198

D 199

2. 在具有  $2n$  个结点的完全二叉树中, 叶子结点个数为 ( )

A  $n$

B  $n+1$

C  $n-1$

D  $n/2$

3. 一个具有767个节点的完全二叉树, 其叶子节点个数为 ( )

A 383

B 384

C 385

D 386

4. 一棵完全二叉树的节点数为531个, 那么这棵树的高度为 ( )

A 11

B 10

C 8

D 12

答案:

- 1.B
- 2.A
- 3.B
- 4.B

## 2.4 二叉树的存储

二叉树的存储结构分为：顺序存储和类似于链表的链式存储。

顺序存储在下节介绍。

二叉树的链式存储是通过一个一个的节点引用起来的，常见的表示方式有二叉和三叉表示方式，具体如下：

```
// 孩子表示法
class Node {
    int val;    // 数据域
    Node left;  // 左孩子的引用，常常代表左孩子为根的整棵左子树
    Node right; // 右孩子的引用，常常代表右孩子为根的整棵右子树
}

// 孩子双亲表示法
class Node {
    int val;    // 数据域
    Node left;  // 左孩子的引用，常常代表左孩子为根的整棵左子树
    Node right; // 右孩子的引用，常常代表右孩子为根的整棵右子树
    Node parent; // 当前节点的根节点
}
```

孩子双亲表示法后序在平衡树位置介绍，本文采用孩子表示法来构建二叉树。

## 2.5 二叉树的基本操作

### 2.5.1 前置说明

在学习二叉树的基本操作前，需先要创建一棵二叉树，然后才能学习其相关的基本操作。由于现在大家对二叉树结构掌握还不够深入，为了降低大家学习成本，此处手动快速创建一棵简单的二叉树，快速进入二叉树操作学习，等二叉树结构了解的差不多时，我们反过头再来研究二叉树真正的创建方式。

```
public class BinaryTree{
    public static class BTreeNode{
        BTreeNode left;
        BTreeNode right;
        int value;

        BTreeNode(int value){
            this.value = value;
        }
    }

    private BTreeNode root;
```



```

public void createBinaryTree(){
    BTreeNode node1 = new BTreeNode(1);
    BTreeNode node2 = new BTreeNode(2);
    BTreeNode node3 = new BTreeNode(3);
    BTreeNode node4 = new BTreeNode(4);
    BTreeNode node5 = new BTreeNode(5);
    BTreeNode node6 = new BTreeNode(6);

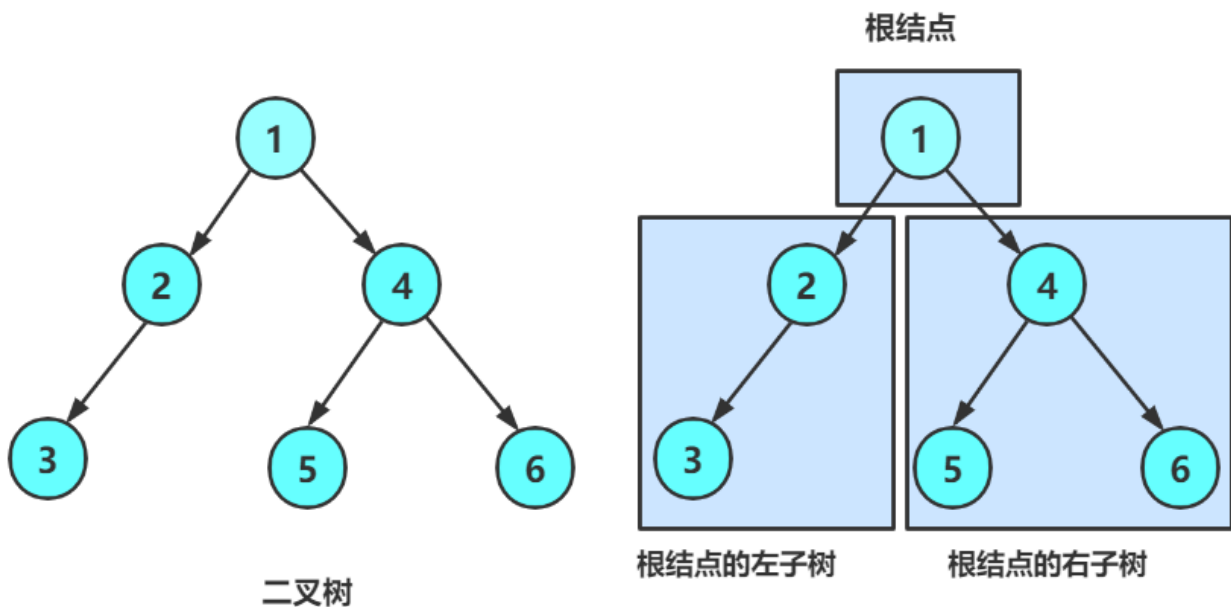
    root = node1;
    node1.left = node2;
    node2.left = node3;
    node1.right = node4;
    node4.left = node5;
    node5.right = node6;
}
}

```

注意：上述代码并不是创建二叉树的方式，真正创建二叉树方式后序详解重点讲解。

再看二叉树基本操作前，再回顾下二叉树的概念，**二叉树是**：

1. **空树**
2. **非空**：根节点，根节点的左子树、根节点的右子树组成的。

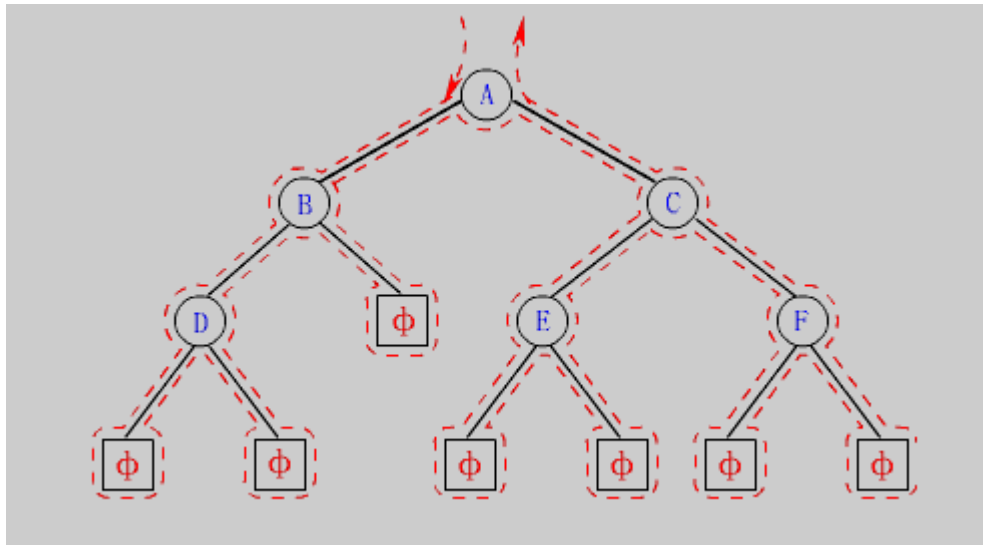


从概念中可以看出，二叉树定义是递归式的，因此后序基本操作中基本都是按照该概念实现的。

## 2.5.2 二叉树的遍历

### 1. 前中后序遍历

学习二叉树结构，最简单的方式就是遍历。所谓**遍历(Traversal)**是指沿着某条搜索路线，依次对树中每个结点均做一次且仅做一次访问。访问结点所做的操作依赖于具体的应用问题(比如：打印节点内容、节点内容加1)。遍历是二叉树上最重要的操作之一，是二叉树上进行其它运算之基础。



在遍历二叉树时，如果没有进行某种约定，每个人都按照自己的方式遍历，得出的结果就比较混乱，**如果按照某种规则进行约定，则每个人对于同一棵树的遍历结果肯定是相同的。**如果N代表根节点，L代表根节点的左子树，R代表根节点的右子树，则根据遍历根节点的先后次序有以下遍历方式：

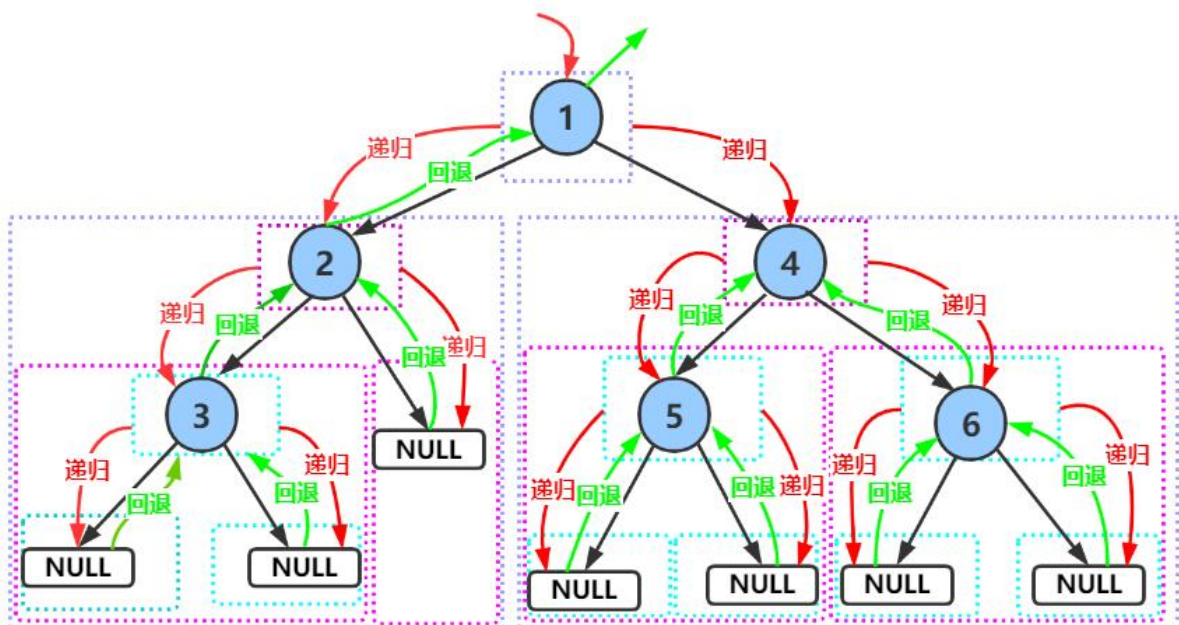
- NLR：前序遍历(Preorder Traversal 亦称先序遍历)——访问根结点--->根的左子树--->根的右子树。
- LNR：中序遍历(Inorder Traversal)——根的左子树--->根结点--->根的右子树。
- LRN：后序遍历(Postorder Traversal)——根的左子树--->根的右子树--->根结点。

```
// 前序遍历
void preOrder(Node root);

// 中序遍历
void inOrder(Node root);

// 后序遍历
void postOrder(Node root);
```

下面主要分析前序递归遍历，中序与后序图解类似，同学们可自己动手绘制。



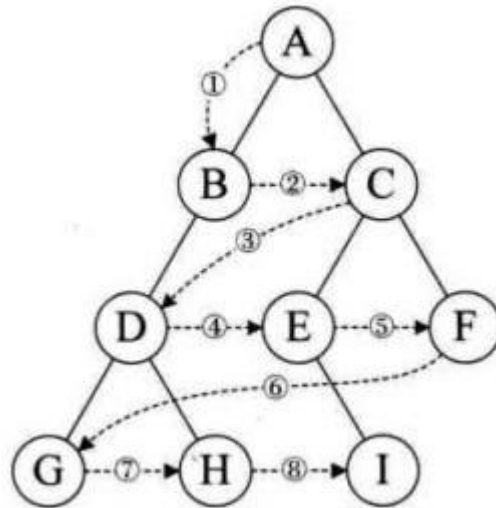
前序遍历结果：1 2 3 4 5 6

中序遍历结果：3 2 1 5 4 6

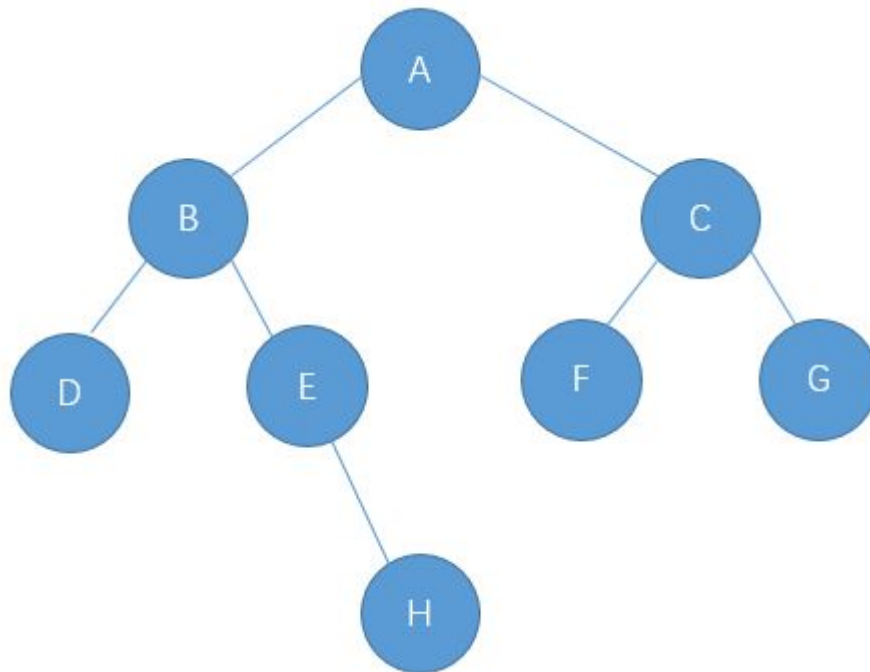
后序遍历结果：3 1 5 6 4 1

## 2. 层序遍历

**层序遍历：**除了先序遍历、中序遍历、后序遍历外，还可以对二叉树进行层序遍历。设二叉树的根节点所在层数为1，层序遍历就是从所在二叉树的根节点出发，首先访问第一层的树根节点，然后从左到右访问第2层上的节点，接着是第三层的节点，以此类推，自上而下，自左至右逐层访问树的结点的过程就是层序遍历。



【练习】请同学们根据以上二叉树的三种遍历方式，给出以下二叉树的：



选择题

- 1.某完全二叉树按层次输出（同一层从左到右）的序列为 ABCDEFGH 。该完全二叉树的前序序列为()  
A: ABDHECFG    B: ABCDEFGH    C: HDBEAFCH    D: HDEBFGCA
- 2.二叉树的先序遍历和中序遍历如下：先序遍历：EFHIGJK;中序遍历：HFIEJGK.则二叉树根结点为()  
A: E        B: F        C: G        D: H
- 3.设一棵二叉树的中序遍历序列：badce，后序遍历序列：bdeca，则二叉树前序遍历序列为()  
A: adbce    B: decab    C: debac    D: abcde
- 4.某二叉树的后序遍历序列与中序遍历序列相同，均为 ABCDEF，则按层次输出(同一层从左到右)的序列为()  
A: FEDCBA    B: CBAFED    C: DEFCBA    D: ABCDEF

【参考答案】 1.A 2.A 3.D 4.A

### 2.5.3 二叉树的基本操作

```
// 获取树中节点的个数
int size(Node root);

// 获取叶子节点的个数
int getLeafNodeCount(Node root);

// 子问题思路-求叶子结点个数

// 获取第K层节点的个数
int getKLevelNodeCount(Node root,int k);

// 获取二叉树的高度
int getHeight(Node root);

// 检测值为value的元素是否存在
Node find(Node root, int val);

//层序遍历
void levelOrder(Node root);

// 判断一棵树是不是完全二叉树
boolean isCompleteTree(Node root);
```

### 2.6 二叉树相关oj题

1. 检查两颗树是否相同。 [OJ链接](#)
2. 另一颗树的子树。 [OJ链接](#)
3. 翻转二叉树。 [OJ链接](#)
4. 判断一颗二叉树是否是平衡二叉树。 [OJ链接](#)
5. 对称二叉树。 [OJ链接](#)
6. 二叉树的构建及遍历。 [OJ链接](#)
7. 二叉树的分层遍历。 [OJ链接](#)
8. 给定一个二叉树, 找到该树中两个指定节点的最近公共祖先。 [OJ链接](#)
9. 根据一棵树的前序遍历与中序遍历构造二叉树。 [OJ链接](#)

10. 根据一棵树的中序遍历与后序遍历构造二叉树 ([课堂不讲解, 课后完成作业])。 [OI链接](#)
11. 二叉树创建字符串。 [OI链接](#)
12. 二叉树前序非递归遍历实现。 [OI链接](#)
13. 二叉树中序非递归遍历实现。 [OI链接](#)
14. 二叉树后序非递归遍历实现。 [OI链接](#)