带你种一棵二叉树

今日目标：

1：能够说出树的定义，树的高度，深度，层等概念

2：能够说出二叉树的定义及特点

3：能够说出二叉树中满二叉树，完全二叉树的定义，及各自的特点；如何存储

4：能够说出二叉树的前序，中序，后序遍历形式

5：完成相关实战题目

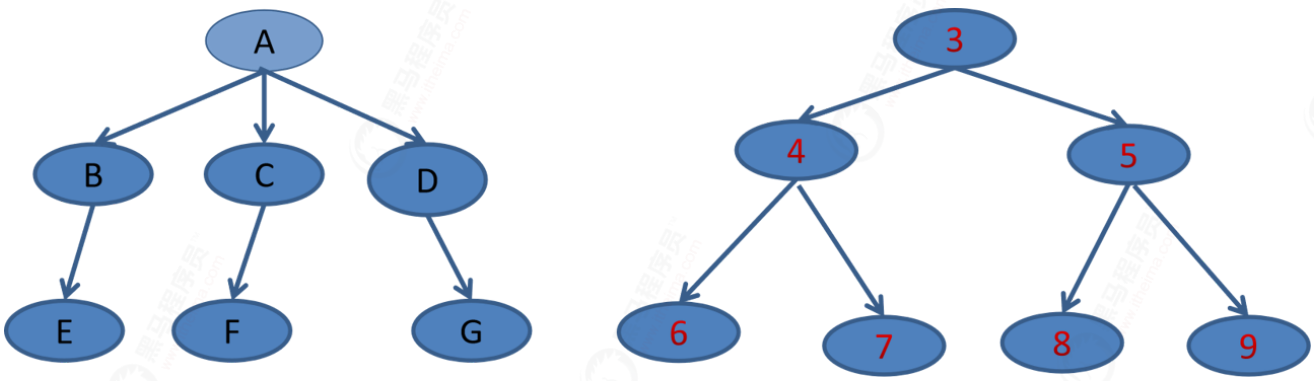
1、树

前面学习的：数组，链表，栈，队列，散列表，集合等均属于线性数据结构。而树及后面要学习的一些数据结构属于非线性数据结构！

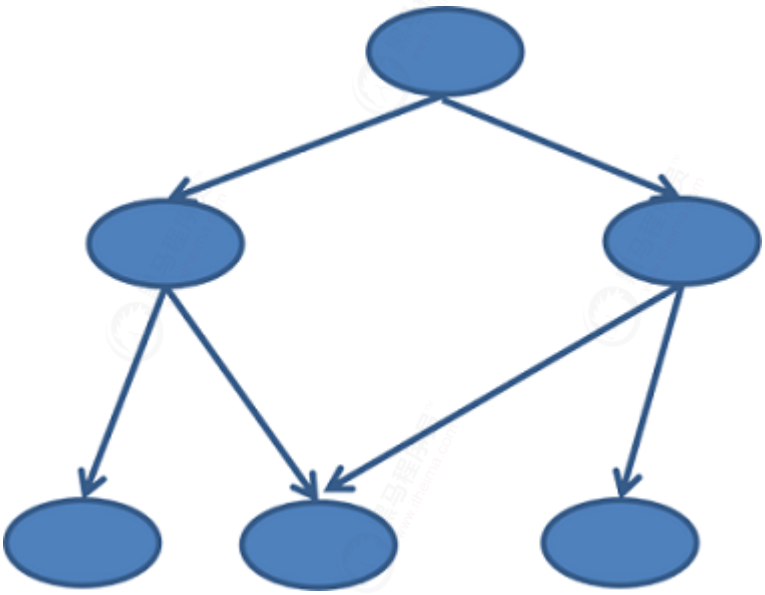
1.1、定义

> 树在维基百科中的定义为：，**树**（英语：Tree）是一种无向图（undirected graph），其中任意两个顶点间存在唯一一条路径。或者说，只要没有回路的连通图就是树。在计算机科学中，**树**（英语：tree）是一种抽象数据类型（ADT）或是实现这种抽象数据类型的数据结构，用来模拟具有树状结构性质的数据集合。它是由n（n>0）个有限节点组成一个具有层次关系的集合。

这个定义不是特别好懂，我们借助图形来理解就会非常的清晰



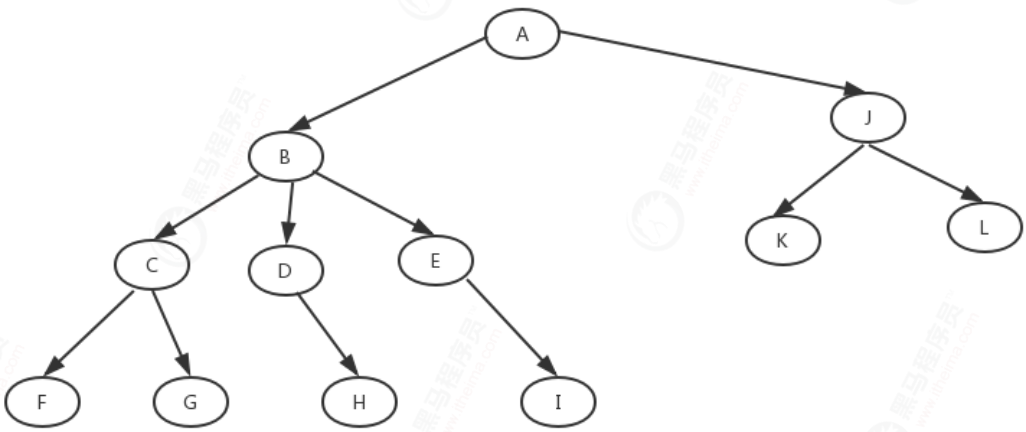
以上这些都是树，下面我们再看几个**不是树**的情况



那**树**具备什么样**的特点**呢?

- 每个节点都只有有限个子节点或无子节点;
- 没有父节点的节点称为根节点;
- 每一个非根节点有且只有一个父节点;
- 除了根节点外，每个子节点可以分为多个不相交的**子树**;
- 树里面没有环路(cycle)

比如在下方这副图中:



> 其中：节点B是节点C D E的**父节点**，C D E就是B的**子节点**，C D E之间称为**兄弟节点**，我们把没有父节点的A节点叫做**根节点**，我们把没有子节点的节点称为**叶子节点**如：F G H I K L均是叶子节点。
>
> 从节点B开始也是一个树，我们把它叫做根节点A的**子树**，子树也是一棵树，同样满足树的定义，其他同理可得。

1.2、相关概念

理解了树的定义之后我们来学习几个跟树相关的概念：**高度(Heigh)，深度(Depth)，层(Level)**，我们依次来看这几个概念：

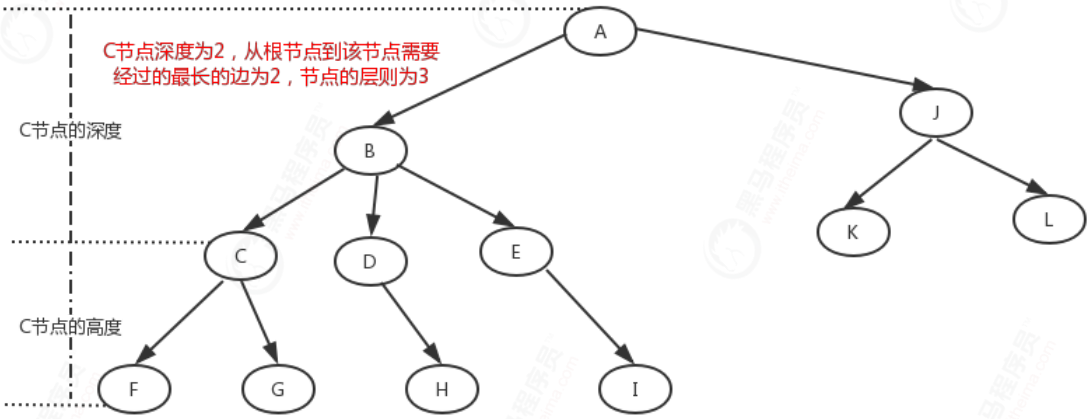
节点的高度：节点到叶子节点的最长路径(边数)，所有叶子节点的高度为0。

节点的深度：根节点到这个节点所经历的边的个数，根的深度为0。二叉树的深度为根节点到最远叶子节点的最长路径上的节点数。

节点的层数：节点的深度+1

树的高度：根节点的高度

我们用一幅图来继续说明如下：



2、二叉树

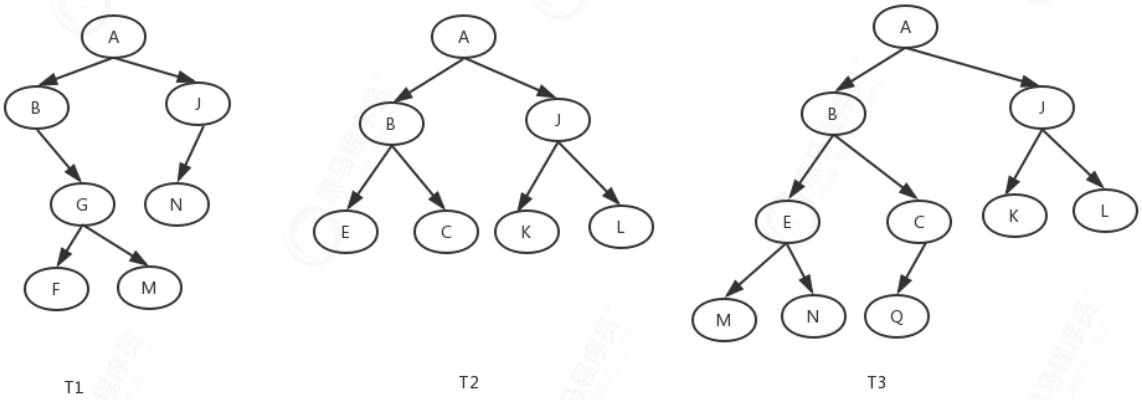
树这种数据结构形式结构是多种多样的，但是在实际企业开发中用的最多的还是二叉树

2.1、二叉树概述

2.1.1、定义及特点

二叉树，顾名思义，每个节点最多有两个“叉”，也就是两个子节点，分别是**左子节点**和**右子节点**。不过，二叉树并不要求每个节点都有两个子节点，有的节点只有左子节点，有的节点只有右子节点。

并且，**同理可得**，二叉树每个节点的**左子树和右子树也分别满足二叉树的定义。**



有了二叉树的概念定义之后，我们同理可得：三叉树，四叉树，五叉树，.....，N叉树。

> **思考：**从二叉树的定义来看，它符合什么特征？--recursion

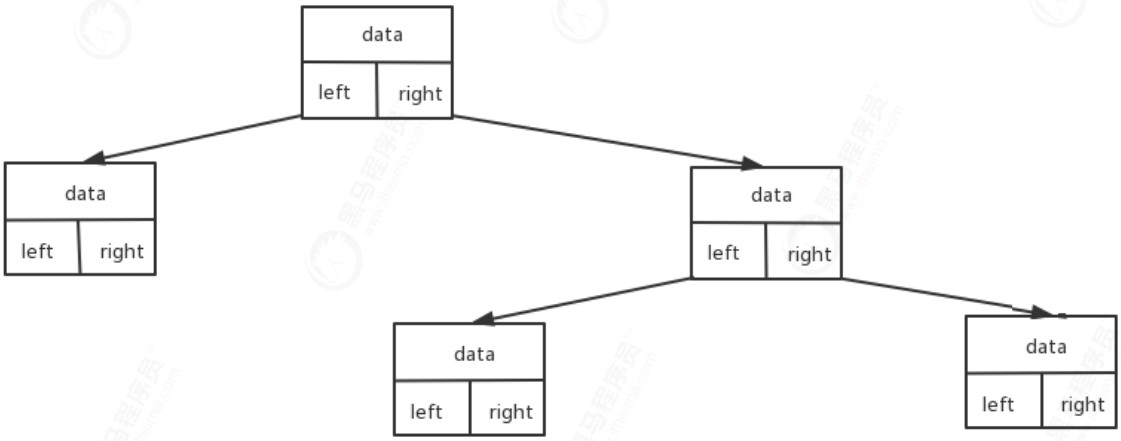
二叉树的存储特点

很多其他高级数据结构都是基于二叉树，他们的操作特点各有不同，但从存储上来说底层无外乎就是两种：数组存储，链式存储。

基于链式存储的树的节点可定义如下：

```java
public class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;

    TreeNode() {}
    TreeNode(int val) { this.val = val; }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;
        this.left = left;
        this.right = right;
    }
}
```

用图表示为：



依次同理，基于链式存储的N叉树的节点可定义为如下：

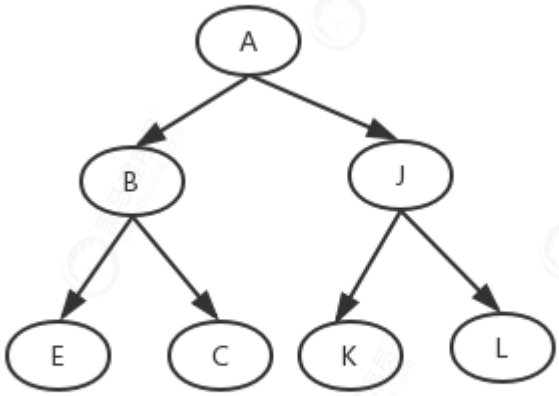
```
public class TreeNode {
    int val;
    List<TreeNode> children

    TreeNode() {}
    ...
}
```

2.1.2、分类

在二叉树中，有几种特殊的情况，分别叫做：**满二叉树，完全二叉树**

1、满二叉树

叶子节点全都在最底层，除了叶子节点之外，每个节点都有左右两个子节点，这种二叉树就叫作**满二叉树。**



> 树的高度为：

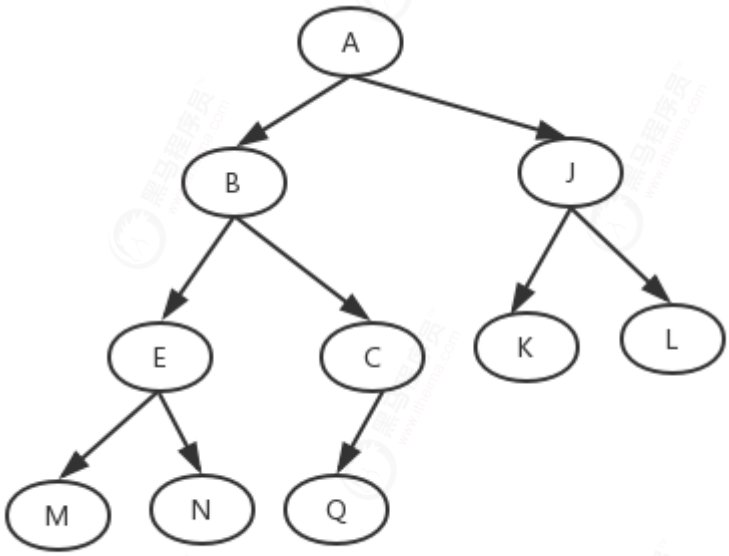
$$log_2 n$$

2、完全二叉树

1：叶子节点都在最底下两层，

2：最后一层的叶子节点都靠左排列(某个节点只有一个叶子节点的情况下)，

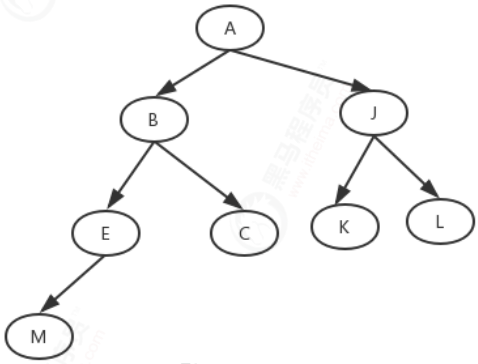
3：并且除了最后一层，其他层的节点个数都要达到最大，这种二叉树叫作**完全二叉树。**



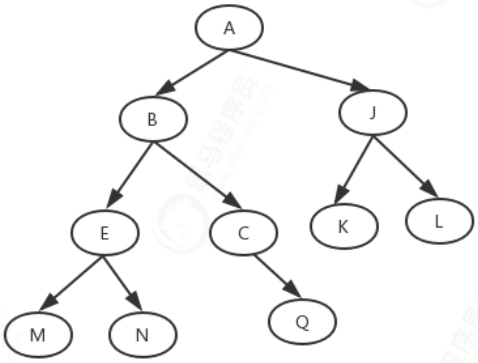
> 树的高度：<=

$$log_2 n$$

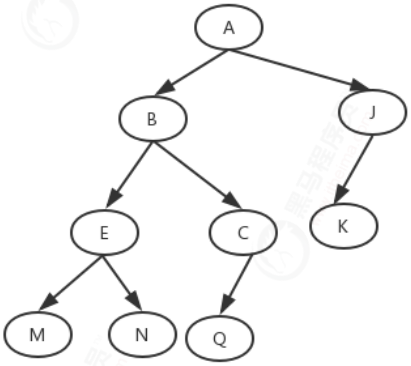
满二叉树我们特别容易理解，完全二叉树我们可能就不是特别能够分清楚，下面分析一下看下方图中哪些是完全二叉树?



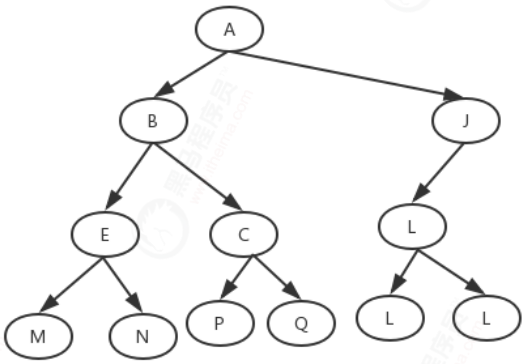
T1



T2



T3



T4

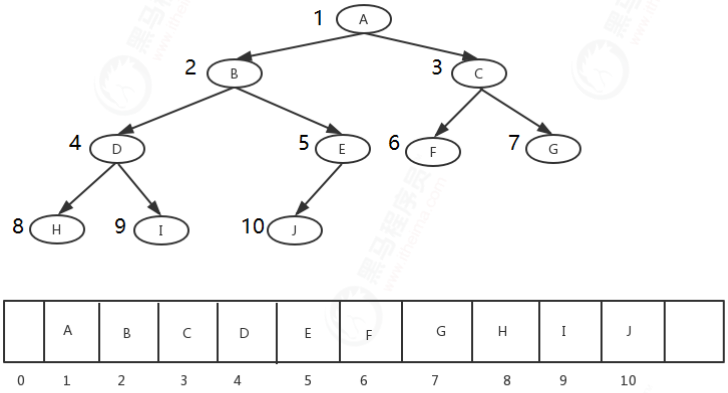
> 答案是：T1是完全二叉树，T2,T3,T4均不是完全二叉树，

问题：为什么偏偏把最后一层的叶子节点靠左排列的叫完全二叉树？如果靠右排列就不能叫完全二叉树了吗？原因是什么？

这个主要是基于**完全二叉树的存储**方式，前面说到想要存储一棵二叉树，我们有两种方法，一种是基于指针或者引用的二叉链式存储法，一种是基于数组的顺序存储法。

而对于**完全二叉树**我们**更倾向采用基于数组的顺序存储方式。**具体做法如下：

> 把根节点存储在下标 i = 1 的位置，那左子节点存储在下标 2 * i = 2 的位置，右子节点存储在 2 * i + 1 = 3 的位置。
>
> 以此类推，B 节点的左子节点存储在 2i = 2 * 2 = 4 的位置，右子节点存储在 2 * i + 1 = 2 * 2 + 1 = 5 的位置。如下图所示：



存储特征：

对于任意一个节点假设它存储在数组下标为k的位置则

1：下标 2 * k 的位置存储的是它的左子节点

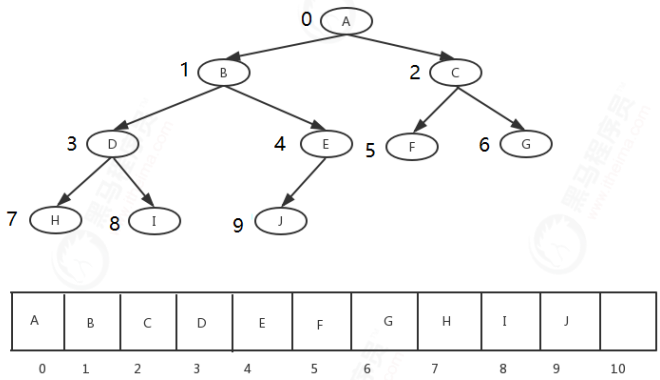
2：下标 2 * k +1 的位置存储它的右子节点

对于任意一个节点假设它存储在数组下标为k的位置则：

1：下标 k/2 的位置存储的是它的父节点

从存储结果来看，用**数组存储完全二叉树能够有效利用存储空间**，我们只是浪费了下标为0的一个存储位置。

当然我们也可以从数组下标为0的位置开始存储完全二叉树的根节点，这样对于任意一个节点存储在下标为k的位置，下标 2 * k +1 的位置存储它的左子节点，下标 2 * k +2 的位置存储它的右子节点，但是为了便于计算我们还是选择从下标为1的位置开始存储。



存储特征：

对于任意一个节点假设它存储在数组下标为k的位置则

1：下标 2 * k +1 的位置存储的是它的左子节点

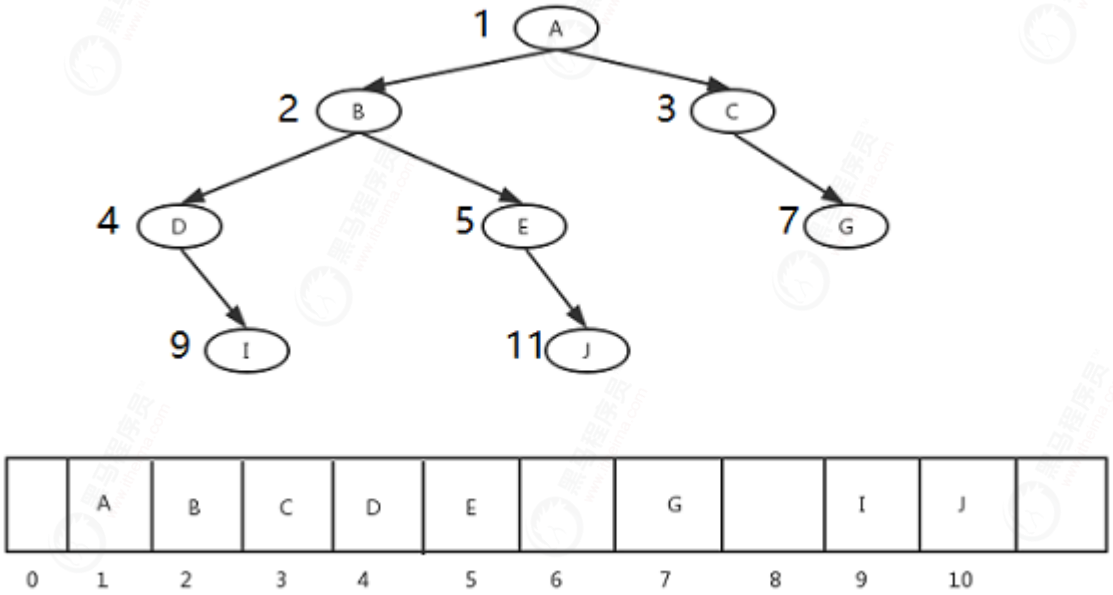
2：下标 2 * k +2 的位置存储它的右子节点

对于任意一个节点假设它存储在数组下标为k的位置则：

1：下标 (k-1) /2 的位置存储的是它的父节点

如此一来就可以通过下标计算，把整棵树都串起来。

但如果不是完全二叉树采用数组来进行顺序存储的话，如下图，浪费的存储空间就比较多了



总之：

如果某棵二叉树是一棵完全二叉树，那用数组存储无疑是最节省内存的一种方式。因为数组的存储方式并不需要像链式存储法那样，要存储额外的左右子节点的指针。这也是为什么完全二叉树要求最后一层的子节点都靠左并且除了最后一层，其他层的节点个数都要达到最大的原因。

重要：二叉树的性质：

性质1：在二叉树的第i层上至多有`2^(i-1)`个结点（i≥1）。

性质2：深度为k的二叉树最多有`2^k -1`个结点（k≥1）。

性质3：一棵二叉树的度为0结点数为n0，度为2的结点数为n2，则n0 = n2 + 1。

节点的度：节点的子节点个数（指出去的指针数），所以二叉树节点的度分三种0,1,2，

假设一个二叉树有n个节点：

- 度为0（叶子）的节点个数是n0
- 度为1的节点个数是n1
- 度为2的节点个数是n2

则有如下公式成立：

- n0 = n2 + 1

  如果该二叉树是完全二叉树，则有n1=0或者n1=1，因此

  当n1=0时：n0 = (n +1) / 2 ，此时n为奇数

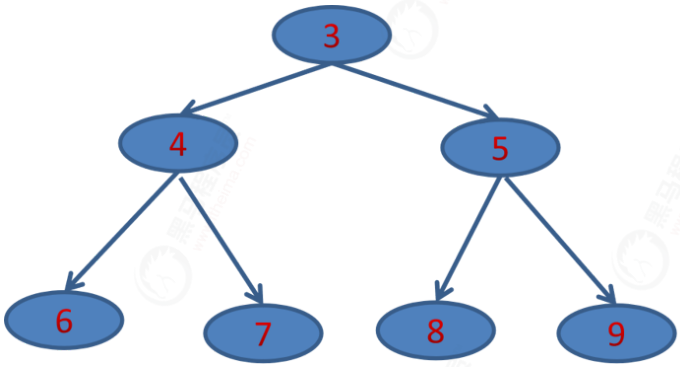
  当n1=1时，n0= n/2，此时n为偶数

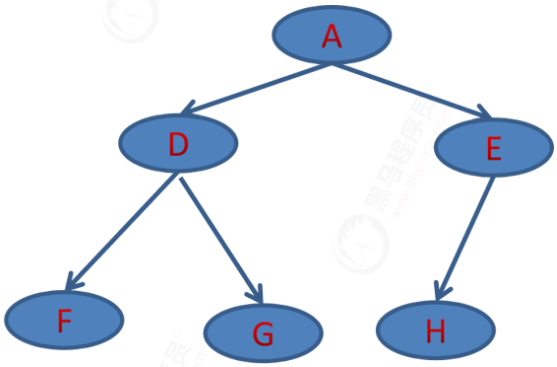
性质4：具有n个结点的完全二叉树的深度为floor(log2 n) + 1 。

2.2、二叉树遍历

前面讲述了二叉树的存储结构，接下来学习二叉树的遍历方式，经典的三种遍历方式：**前序遍历，中序遍历，后序遍历**，

> 数据结构和算法动态可视化： https://visualgo.net/zh/





前序遍历：对于树中的任意节点来说，先打印这个节点，然后再打印它的左子树，最后打印它的右子树。

> 对于图中第一棵树前序遍历结果：3467589

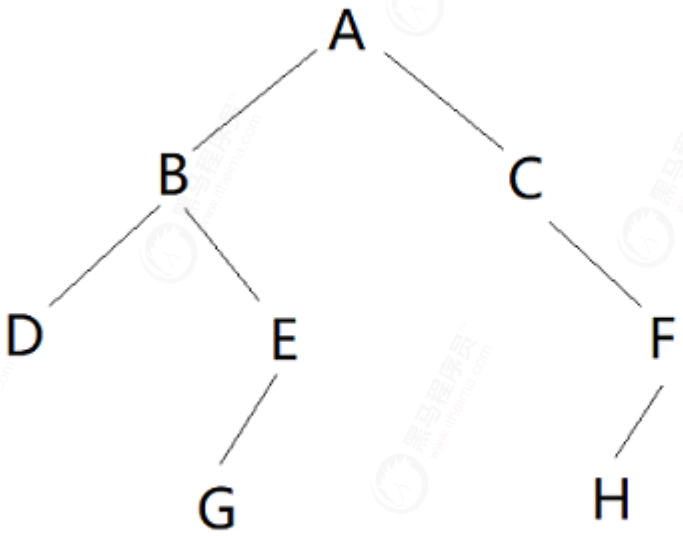
中序遍历：对于树中的任意节点来说，先打印它的左子树，然后再打印它本身，最后打印它的右子树。

> 对于图中第一棵树中序遍历结果：6473859

后序遍历：对于树中的任意节点来说，先打印它的左子树，然后再打印它的右子树，最后打印这个节点本身。

> 对于图中第一棵树后序遍历结果：6748953

> 课堂练习：请说出如下二叉树的前中后序遍历结果



> **思考：**从二叉树的遍历形式来看，适合用什么样的思想来完成呢？------recursion

二叉树遍历的时间复杂度是多少呢?

通过我们分析的二叉树的遍历流程我们可以发现，遍历二叉树的时间复杂度跟二叉树节点的个数n成正比，因此，**二叉树遍历的时间复杂度是O(n)。**

2.3、实战题目

144. 二叉树的前序遍历

腾讯，美团点评最近面试题，144：二叉树的前序遍历

```
class Solution {
    public List<Integer> preorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList();
        recur(root,res);
        return res;
    }

    public void recur(TreeNode node,List<Integer> res) {
        //递归终止条件
        if (node == null) {
            return ;

        }
```

```
        //处理当前逻辑
        res.add(node.val);
        //下探到下一层
        recur(node.left,res);
        recur(node.right,res);
    }
}
```

> **时间复杂度是O(n)**，
>
> **空间复杂度：O(height)**，其中height 表示二叉树的高度。递归函数需要栈空间，而栈空间取决于递归的深度，因此空间复杂度等价于二叉树的高度。

与该题几乎一样的是如下几道题目：

腾讯，小米最近面试题，94. 二叉树的中序遍历

字节，facebook最近面试题，145. 二叉树的后序遍历

> **进阶：**能否用迭代算法完成二叉树遍历？

589. N叉树的前序遍历

微软半年内面试题，589. N叉树的前序遍历

N叉树前序：先根，然后依次子节点

递归算法实现起来很简单

```
class Solution {
    public List<Integer> preorder(Node root) {
        List<Integer> res = new ArrayList();
        recur(root,res);
        return res;
    }

    public void recur(Node node,List<Integer> res) {
        //终止条件
        if (node == null) {
            return;
        }
        res.add(node.val);
        List<Node> children = node.children;
        if (children!= null && children.size() > 0) {
            for (Node n:children) {
                recur(n,res);
            }
        }
    }
}
```

与该题一样的还有一道：

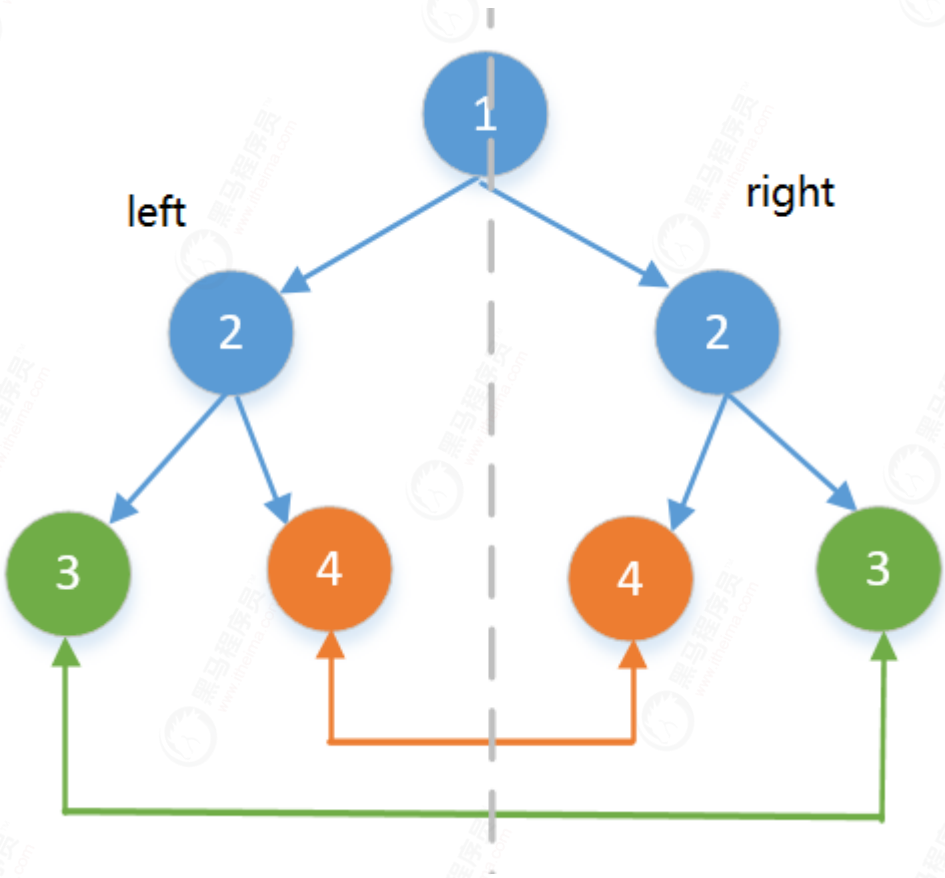
亚马逊半年到1年内考过，590. N叉树的后序遍历

N叉树后续：子节点（依次）然后根节点，

使用递归算法代码非常简单跟前序遍历几乎一样。

101. 对称二叉树

字节，google最近面试题，101. 对称二叉树



镜像对称，就是左右两边相等，也就是左子树和右子树是相等的，也就是说要递归的比较左子树和右子树。

我们将根节点的左子树记做 `left`，右子树记做 `right`。比较 `left` 是否等于 `right`，不等的话直接返回就可以了。如果相等，接着比较 `left` 的左节点和 `right` 的右节点，再比较 `left` 的右节点和 `right` 的左节点。

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        //特殊判断
        if (root == null) {
            return true;
        }

        return recur(root.left,root.right);
```

```
    }

    //开始递归
    public boolean recur(TreeNode left,TreeNode right) {
        if (left == null && right == null) {
            return true;
        }
        if (left == null && right != null) {
            return false;
        }
        if ( left != null && right == null) {
            return false;
        }
        if (left.val != right.val) {
            return false;
        }

        boolean one = recur(left.left,right.right);
        boolean two = recur(left.right,right.left);
        return one && two;
    }
}
```

时间复杂度是O(n)，因为要遍历n个节点

空间复杂度取决与递归的深度（跟递归使用的系统栈空间有关），在这里也就是跟该二叉树的高度有关，最坏情况下当二叉树退化成链表时就是O(n)

进阶：谷歌，字节，苹果最近面试题：226. 翻转二叉树，与剑指 Offer 27. 二叉树的镜像相同

翻转二叉树：交换左右子节点指针即可，然后一直递归到最底层

```
class Solution {
    public TreeNode invertTree(TreeNode root) {
        recur(root);
        return root;
    }

    public void recur (TreeNode node) {
        //递归终止条件
        if ( node == null) {
            return;
        }
        //交换左右子节点即可
        TreeNode temp = node.right;
        node.right = node.left;
        node.left = temp;
        //下探到下一层,分别翻转左子树和右子树
        recur(node.left);
        recur(node.right);
    }
}
```

106. 从中序与后序遍历序列构造二叉树

正式讲解这个实战题目前，先来看一道经典的笔试主观题目

题目1、小米2020秋招

> 已知一棵二叉树前序遍历和中序遍历分别为ABDEGCFH和DBGEACHF，则该二叉树的后序遍历为

- [ ] A: GEDHFBCA
- [ ] B: DGEBHFCA
- [ ] C: ABCDEFGH
- [ ] D: ACBFEDHG

> 题解：
>
> 1：记住一个特点，二叉树的前序遍历结果中第一个一定是根节点，后续遍历结果中根节点一定在最后
>
> 2：二叉树的遍历过程其实是一个递归过程，整个二叉树，它的子树，子子树等的遍历形式一样，所以一定要记住四个字：**同理可得**！
>
> 答案为：B

同理，如果知道一棵二叉树的后序遍历结果和中序遍历结果，我们同样能还原整棵树并得到它的前序遍历结果。

> 随堂练习：一棵二叉树后序遍历结果为：GDBHEFCA，中序遍历结果为：GDBAHECF 则前序遍历结果为：

字节跳动，亚马逊最近面试题，106. 从中序与后序遍历序列构造二叉树

结合中序遍历和后序遍历的特点及递归的思想解决

题解：https://leetcode-cn.com/problems/construct-binary-tree-from-inorder-and-postorder-traversal/solution/tu-jie-gou-zao-er-cha-shu-wei-wan-dai-xu-by-user72/

```
1  class Solution {
2
3      Map<Integer,Integer> map = new HashMap();
4
5      int[] post;
6
```

```
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        //缓存中序遍历结果的值和对应的数组下标
        for (int i=0;i< inorder.length;i++) {
            map.put(inorder[i] , i);
        }
        post = postorder;
        TreeNode root = recurBuild(0,inorder.length-1,0,postorder.length-1);
        return root;
    }
    /*
    is和ie分别是某子树中序遍历结果的数组下标开始和结束
    ps,pe分别是某子树后序遍历结果的数组下标开始和结束,并且pe位置的值就是这棵子树的根节点
    */
    public TreeNode recurBuild(int is,int ie,int ps,int pe) {
        if ( is > ie || ps > pe ) {
            return null;
        }
        //先找到子树的根节点
        TreeNode root = new TreeNode(post[pe]);
        //从中序遍历结果中拿到根节点的下标
        int ri = map.get(post[pe]);

        //分别递归的去构造该根节点的左子树和右子树
        root.left = recurBuild(is,ri-1,ps,ps+ri-is-1);
        root.right = recurBuild(ri+1,ie,ps+ri-is,pe-1);
        return root;
    }
}
```

扩展：还有一道同类型题目，[105. 从前序与中序遍历序列构造二叉树]

```
class Solution {
    Map<Integer,Integer> map = new HashMap();

    int[] pre;

    public TreeNode buildTree(int[] preorder, int[] inorder) {
        //先将中序遍历结果和对应的下标缓存
        for (int i=0;i<inorder.length;i++) {
            map.put(inorder[i],i);
        }
        pre = preorder;
        TreeNode root = recurBulid(0,pre.length-1,0,inorder.length-1);
        return root;
    }

    public TreeNode recurBulid(int ps,int pe,int is,int ie) {
        if (ps > pe || is > ie) {
            return null;
        }
        //先序遍历的开始位置是根
```

```
        TreeNode root = new TreeNode(pre[ps]);
        //找到根节点在中序遍历中的下标，这样可以分割出其左子树和右子树
        int ri = map.get(pre[ps]);

        //构建其左子树和右子树
        root.left = recurBulid(ps+1,ps+ri-is,is,ri-1);
        root.right = recurBulid(ps+ri-is+1,pe,ri+1,ie);
        return root;
    }
}
```

2.4、扩展题目

题目1、小米2020秋招

> 一棵有15个节点的完全二叉树和一棵同样有15个节点的普通二叉树，叶子节点的个数最多会差多少个?

- [ ] A: 3
- [ ] B: 5
- [ ] C: 7
- [ ] D: 9

> 题解：完全二叉树，除最后一层外，其他层节点数都达到了最大，而15个节点刚好是4层，树的高度是3，叶子节点为8个，要使两棵树叶子节点相差最多，那有15个节点的普通二叉树只能是退化成一个链表，只有一个叶子节点，所以相差最多为7。

题目2、小米2020秋招

> 现有一个包含m个节点的三叉树，即每个节点都有3个指向孩子节点的指针，请问：在这3m个指针中有（）个空指针

- [ ] A: 2m
- [ ] B: 2m+1
- [ ] C: 2m-1
- [ ] D: 3m

> 题解：对一个一棵三叉树，每个节点都有三个指针，所以m个节点的三叉树总共有3m个指针，但是对于每个节点来说都只需要一个指针指向自己，除了根节点以外，所以m个节点的三叉树需要耗费m-1个指针，因此有 3m-（m-1）= 2m +1 个指针是空指针

题目3、快手2020秋招

> 如果一棵二叉树的中序遍历是 BAC，那么它的先序遍历不可能是

- [ ] A: ABC
- [ ] B: CBA
- [ ] C: ACB
- [ ] D: BAC

> 题解：该题可根据中序和前序遍历特点逐个推导
>
> 答案为C

题目4、京东2020秋招

> 一颗二叉树的叶子节点有5个，出度为1的结点有3个，该二叉树的结点总个数是?

- [ ] A: 11
- [ ] B: 12
- [ ] C: 13
- [ ] D: 14

> 题解：
>
> > 根据二叉树的性质分析答案为12

题目5、京东2020秋招

> 完全二叉树699个节点，则叶子节点有多少个?

- [ ] A: 256
- [ ] B: 350
- [ ] C: 352
- [ ] D: 512

> 题解：根据上一题题解可直接得出答案为350

题目6、小米2020秋招

> 一个二叉树的先序遍历结果和中序遍历结果相同，则其所有非叶子节点必须满足的条件是?

- [ ] A: 只有左子树
- [ ] B: 只有右子树
- [ ] C: 节点的度为1
- [ ] D: 节点的度为2

题解：还是回归到先序遍历和中序遍历的特点

答案选B