王争的算法训练营

习题课：二叉树

二叉树相关题型/套路

王争的算法训练营

重点中的重点，面试常考，是个硬骨头，一定要拿下

1）类似链表操作，二叉树上的操作一般都会涉及到树的遍历。

2）因为树跟子树结构、操作很相似，所以，绝大部分题目都使用递归实现。

二叉树的8类小题型：

题型1：二叉树前中后序遍历

题型2：二叉树按层遍历

题型3：二叉树上的递归

题型4：二叉查找树

题型5：LCA最近公共祖先

题型6：二叉树转单、双、循环链表

题型7：按照遍历结果反向构建二叉树

题型8：二叉树上的最长路径和（树形DP）

习题课：二叉树

王争的算法训练营

题型1：二叉树前中后序遍历

144. 二叉树的前序遍历（简单）
94. 二叉树的中序遍历（简单）
145. 二叉树的后序遍历（简单）
589. N 叉树的前序遍历（简单）
590. N 叉树的后序遍历（简单）

题型2：二叉树按层遍历

剑指 Offer 32 - I. 从上到下打印二叉树（中等）
102. 二叉树的层序遍历（中等）
剑指 Offer 32 - III. 从上到下打印二叉树 III（中等）
429. N 叉树的层序遍历（中等）
513. 找树左下角的值（中等）

题型3：二叉树上的递归

104. 二叉树的最大深度（简单）
559. N 叉树的最大深度（简单）
剑指 Offer 55 - II. 平衡二叉树（中等）
617. 合并二叉树（简单）
226. 翻转二叉树（简单）
101. 对称二叉树（中等）
98. 验证二叉搜索树（中等）

题型4：二叉查找树

剑指 Offer 54. 二叉搜索树的第k大节点（中等）
538. 把二叉搜索树转换为累加树（中等）
面试题 04.06. 后继者（中等）

题型5：LCA最近公共祖先

236. 二叉树的最近公共祖先（中等）
剑指 Offer 68 - I. 二叉搜索树的最近公共祖先（中等）

题型6：二叉树转单、双、循环链表

114. 二叉树展开为链表（中等）
面试题 17.12. BiNode（中等）
剑指 Offer 36. 二叉搜索树与双向链表（中等）
面试题 04.03. 特定深度节点链表（中等）

题型7：按照遍历结果反向构建二叉树

105. 从前序与中序遍历序列构造二叉树（中等）
889. 根据前序和后序遍历构造二叉树（中等）
106. 从中序与后序遍历序列构造二叉树（中等）
剑指 Offer 33. 二叉搜索树的后序遍历序列（中等）

题型8：二叉树上的最长路径和

543. 二叉树的直径（简单）
剑指 Offer 34. 二叉树中和为某一值的路径（中等）
124. 二叉树中的最大路径和（困难）
437. 路径总和 III（困难）

习题课：二叉树

题型1：二叉树前中后序遍历

144. 二叉树的前序遍历（简单）

94. 二叉树的中序遍历（简单）

145. 二叉树的后序遍历（简单）

589. N 叉树的前序遍历（简单）

590. N 叉树的后序遍历（简单）

```java
class Solution {
    List<Integer> result = new ArrayList<>();
    public List<Integer> preorderTraversal(TreeNode root) {
        preorder(root);
        return result;
    }

    public void preorder(TreeNode root) {
        if (root == null) return;
        result.add(root.val);
        preorder(root.left);
        preorder(root.right);
    }
}
```

```java
class Solution {
    private List<Integer> result = new ArrayList<>();

    public List<Integer> inorderTraversal(TreeNode root) {
      inorde(root);
      return result;
    }

    private void inorder(TreeNode root) {
      if (root == null) return;
      inorder(root.left);
      result.add(root.val);
      inorder(root.right);
    }
}
```

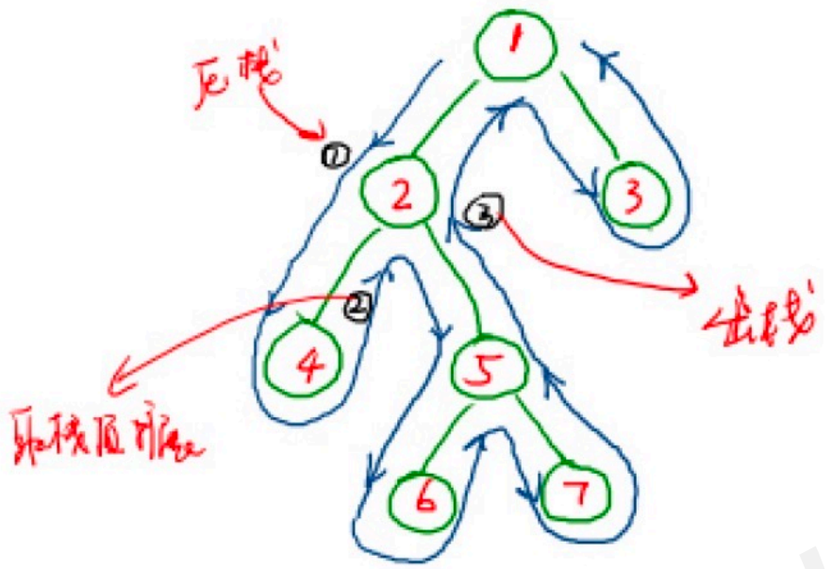
```java
class Solution {
    private List<Integer> result = new ArrayList<>();
    public List<Integer> postorderTraversal(TreeNode root) {
        postorder(root);
        return result;
    }

    private void postorder(TreeNode root) {
        if (root == null) return;
        postorder(root.left);
        postorder(root.right);
        result.add(root.val);
    }
}
```

栈顶元素

1）status=1 表示要扩展左子树，将左子节点入栈

2）status=2 表示左子树扩展完了，要扩展右子树，将右子节点入栈

3）status=3 表示左右子树都扩展完了，出栈




```
def dfs( root ) {
    ①
    dfs ( root . left );
    ②
    dfs (root . right);
    ③
}
```

```java
class Solution {
    private class SFrame {
        public int status = 1;
        public TreeNode node = null;
        public SFrame(int status, TreeNode node) {
            this.status = status;
            this.node = node;
        }
    }
    List<Integer> result = new ArrayList<>();
    public List<Integer> preorderTraversal(TreeNode root) {
        if (root == null) return result;
        Stack<SFrame> stack = new Stack<>();
        stack.push(new SFrame(1, root));
        while (!stack.isEmpty()) {
            if (stack.peek().status == 1) {
                result.add(stack.peek().node.val); // 前序(step=1)
                stack.peek().status = 2;
                if (stack.peek().node.left != null) {
                    stack.push(new SFrame(1, stack.peek().node.left));
                }
                continue;
            }
            if (stack.peek().status == 2) {
                stack.peek().status = 3;
                if (stack.peek().node.right != null) {
                    stack.push(new SFrame(1, stack.peek().node.right));
                }
                continue;
            }
            if (stack.peek().status == 3) {
                stack.pop();
            }
        }
        return result;
    }
}
```

习题课：二叉树

题型2：二叉树按层遍历

剑指 Offer 32 - I. 从上到下打印二叉树（中等）

102. 二叉树的层序遍历（中等）

剑指 Offer 32 - III. 从上到下打印二叉树 III （中等）

429. N 叉树的层序遍历（中等）

513. 找树左下角的值（中等）

二叉树相关题型/套路

王争的算法训练营

题型2：二叉树按层遍历

剑指 Offer 32 - I. 从上到下打印二叉树（中等） 标准的按层遍历

从上到下打印出二叉树的每个节点，同一层的节点按照从左到右的顺序打印。

按层遍历：树上的广度优先搜索
先/中/后序遍历：树上的深度优先搜索

图上的深度优先：一度脑的往下走，直到碰到南墙，然后折返到上一个路口再走

例如:
给定二叉树: `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

比较直接的解法：用二维数组
更优雅经典的解法：用队列

返回:

```
[3,9,20,15,7]
```

提示:

1. 节点总数 <= 1000

习题课：二叉树

```
class Solution {
    public int[] levelOrder(TreeNode root) {
        if (root == null) return new int[0];
        List<Integer> result = new ArrayList<>();
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            result.add(node.val);
            if (node.left != null) {
                queue.add(node.left);
            }
            if (node.right != null) {
                queue.add(node.right);
            }
        }
        int[] resultArray = new int[result.size()];
        for (int i = 0; i < result.size(); i++) {
            resultArray[i] = result.get(i);
        }
        return resultArray;
    }
}
```

习题课：二叉树

题型2：二叉树按层遍历

102. 二叉树的层序遍历（中等）

给你一个二叉树，请你返回其按 **层序遍历** 得到的节点值。 （即逐层地，从左到右访问所有节点）。

示例：
二叉树：`[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其层序遍历结果：

```
[
  [3],
  [9,20],
  [15,7]
]
```

1、记录每个节点的层号（递归、非递归）
2、由null来做每层的区隔
3、由size来做每层的区隔

由“一个一个的处理”转化成“一层一层的处理”

习题课：二叉树

1、记录每个节点的层号（递归）

```
class Solution {
    private List<List<Integer>> result = new ArrayList<List<Integer>>();
    public List<List<Integer>> levelOrder(TreeNode root) {
      dfs(root, 0);
      return result;
    }

    private void dfs(TreeNode root, int level) {
      if (root == null) return;
      if (level > result.size()-1) {
        result.add(new ArrayList<>());
      }
      result.get(level).add(root.val);
      dfs(root.left, level+1);
      dfs(root.right, level+1);
    }
}
```

习题课：二叉树

王争的算法训练营

1、记录每个节点的层号（非递归）

```
class Solution {
    public class TreeNodeWithLevel {
      TreeNode node;
      int level;
      public TreeNodeWithLevel(TreeNode node, int level) {
        this.node = node;
        this.level = level;
      }
    }

    public List<List<Integer>> levelOrder(TreeNode root) {
      List<List<Integer>> result = new ArrayList<List<Integer>>();
      if (root == null) return result;
      Queue<TreeNodeWithLevel> q = new LinkedList<>();
      q.add(new TreeNodeWithLevel(root, 0));
      while (!q.isEmpty()) {
        TreeNodeWithLevel tl = q.poll();
        if (tl.level > result.size()-1) {
          result.add(new ArrayList<>());
        }
        result.get(tl.level).add(tl.node.val);
        if (tl.node.left != null) {
          q.add(new TreeNodeWithLevel(tl.node.left, tl.level+1));
        }
        if (tl.node.right != null) {
          q.add(new TreeNodeWithLevel(tl.node.right, tl.level+1));
        }
      }
      return result;
    }
}
```

习题课：二叉树

王争的算法训练营

2、由null来做每层的区隔

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
      List<List<Integer>> result = new ArrayList<List<Integer>>();
      if (root == null) return result;
      Queue<TreeNode> q = new LinkedList<>();
      q.add(root);
      q.add(null);
      while (!q.isEmpty()) {
        List<Integer> curLevelNodes = new ArrayList<>();
        while (!q.isEmpty()) {
          TreeNode node = q.poll();
          if (node == null) {
            break;
          }
          curLevelNodes.add(node.val);
          if (node.left != null) {
            q.add(node.left);
          }
          if (node.right != null) {
            q.add(node.right);
          }
        }
        if (!curLevelNodes.isEmpty()) {
          result.add(curLevelNodes);
          q.add(null);
        }
      }
      return result;
    }
}
```

习题课：二叉树

3、由size来做每层的区隔

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
      List<List<Integer>> result = new ArrayList<List<Integer>>();
      if (root == null) return result;
      Queue<TreeNode> queue = new LinkedList<TreeNode>();
      queue.add(root);
      while (!queue.isEmpty()) {
        List<Integer> curLevelNodes = new ArrayList<>();
        int curLevelNum = queue.size(); //这一层有多少个节点
        for (int i = 0; i < curLevelNum; ++i) { //从队列中取出这一层的节点进行扩展
          TreeNode treeNode = queue.poll();
          curLevelNodes.add(treeNode.val);
          if (treeNode.left != null) {
            queue.add(treeNode.left);
          }
          if (treeNode.right != null) {
            queue.add(treeNode.right);
          }
        }
        result.add(curLevelNodes);
      }
      return result;
    }
}
```

习题课：二叉树

王争的算法训练营

题型2：二叉树按层遍历

剑指 Offer 32 - III. 从上到下打印二叉树 III （中等）

请实现一个函数按照之字形顺序打印二叉树，即第一行按照从左到右的顺序打印，第二层按照从右到左的顺序打印，第三行再按照从左到右的顺序打印，其他行以此类推。

例如:
给定二叉树: `[3,9,20,null,null,15,7]`,

```
    3
   / \
  9  20
    /  \
   15   7
```

返回其层次遍历结果：

```
[
  [3],
  [20,9],
  [15,7]
]
```

提示：

1. 节点总数 <= 1000

```java
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
      List<List<Integer>> result = new ArrayList<List<Integer>>();
      if (root == null) return result;
      Stack<TreeNode>[] stacks = new Stack[2];
      for (int i = 0; i < 2; ++i) {
        stacks[i] = new Stack<TreeNode>();
      }
      int turn = 0;
      stacks[turn].add(root);

      while (!stacks[turn].isEmpty()) {
        List<Integer> curLevelNodes = new ArrayList<>(); // 记录本层节点

        while(!stacks[turn].isEmpty()) {
          TreeNode treeNode = stacks[turn].pop();
          curLevelNodes.add(treeNode.val);
          if (turn==0) {
            if (treeNode.left != null) {
              stacks[1].push(treeNode.left);
            }
            if (treeNode.right != null) {
              stacks[1].push(treeNode.right);
            }
          } else {
            if (treeNode.right != null) {
              stacks[0].push(treeNode.right);
            }
            if (treeNode.left != null) {
              stacks[0].push(treeNode.left);
            }
          }
        }
        result.add(curLevelNodes);
        turn = (turn+1)%2;
      }
      return result;
    }
}
```

习题课：二叉树

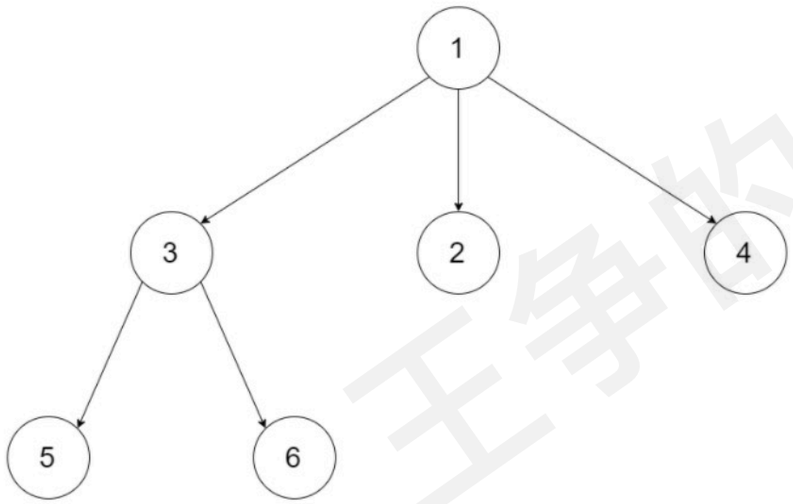
题型2：二叉树按层遍历

429. N 叉树的层序遍历（中等）

给定一个 N 叉树，返回其节点值的*层序遍历*。（即从左到右，逐层遍历）。

树的序列化输入是用层序遍历，每组子节点都由 null 值分隔（参见示例）。

示例 1：




```
输入：root = [1,null,3,2,4,null,5,6]
输出：[[1],[3,2,4],[5,6]]
```

习题课：二叉树

王争的算法训练营

题型2：二叉树按层遍历

429. N 叉树的层序遍历（中等）

```
class Solution {
    public List<List<Integer>> levelOrder(Node root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;
        Queue<Node> queue = new LinkedList<>();
        queue.add(root);
        while (!queue.isEmpty()) {
            int size = queue.size();
            List<Integer> level = new ArrayList<>();
            for (int i = 0; i < size; ++i) {
                Node node = queue.poll();
                level.add(node.val);
                for (int j = 0; j < node.children.size(); ++j) {
                    queue.add(node.children.get(j));
                }
            }
            result.add(level);
        }
        return result;
    }
}
```

习题课：二叉树

王争的算法训练营

题型2：二叉树按层遍历

513. 找树左下角的值（中等）

给定一个二叉树，在树的最后一行找到最左边的值。

示例 1:

输入：

```
    2
   / \
  1   3
```

输出：
1

示例 2:

输入：

```
        1
       / \
      2   3
     /   / \
    4   5   6
       /
      7
```

输出：
7

习题课：二叉树

王争的算法训练营

题型2：二叉树按层遍历

513. 找树左下角的值（中等）

```
class Solution {
    public int findBottomLeftValue(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        int result = -1;
        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            result = node.val;
            if (node.right != null) {
                queue.add(node.right);
            }
            if (node.left != null) {
                queue.add(node.left);
            }
        }
        return result;
    }
}
```

习题课：二叉树

题型3：二叉树上的递归

104. 二叉树的最大深度（简单） **（已讲）**

559. N 叉树的最大深度（简单）

剑指 Offer 55 - II. 平衡二叉树（中等） **（已讲）**

617. 合并二叉树（简单）

226. 翻转二叉树 （简单）

101. 对称二叉树（中等）

98. 验证二叉搜索树（中等）

习题课：二叉树

题型3：二叉树上的递归

104. 二叉树的最大深度（简单）

给定一个二叉树，找出其最大深度。

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。

说明: 叶子节点是指没有子节点的节点。

示例：

给定二叉树 `[3,9,20,null,null,15,7]`，

```
    3
   / \
  9  20
    /  \
   15   7
```

返回它的最大深度 3 。

习题课：二叉树

题型3：二叉树上的递归

104. 二叉树的最大深度（简单） **（已讲）**

```
class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        return Math.max(maxDepth(root.left), maxDepth(root.right)) + 1;
    }
}
```

习题课：二叉树

题型3：二叉树上的递归

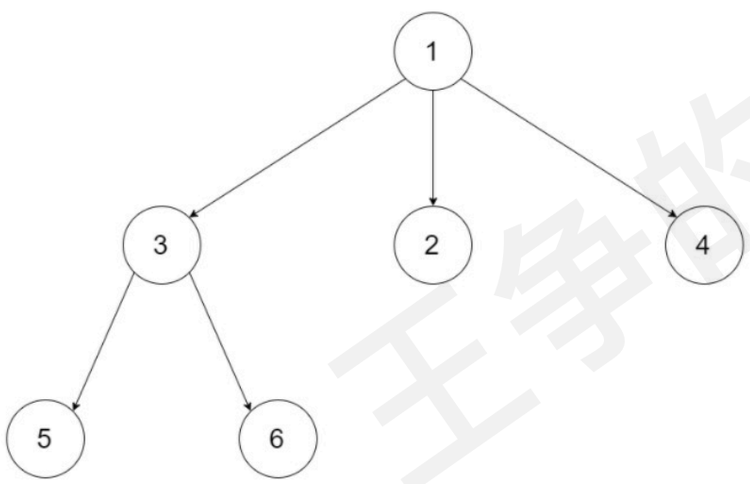
559. N 叉树的最大深度（简单）

给定一个 N 叉树，找到其最大深度。

最大深度是指从根节点到最远叶子节点的最长路径上的节点总数。

N 叉树输入按层序遍历序列化表示，每组子节点由空值分隔（请参见示例）。

示例 1：




```
输入：root = [1,null,3,2,4,null,5,6]
输出：3
```

习题课：二叉树

题型3：二叉树上的递归

559. N 叉树的最大深度（简单）

```
class Solution {
    public int maxDepth(Node root) {
        if (root == null) {
            return 0;
        }
        int childrenMaxDepth = 0;
        for (int i = 0; i < root.children.size(); ++i) {
            int depth = maxDepth(root.children.get(i));
            if (depth > childrenMaxDepth) {
                childrenMaxDepth = depth;
            }
        }
        return childrenMaxDepth+1;
    }
}
```

二叉树相关题型/套路

王争的算法训练营

题型3：二叉树上的递归 剑指 Offer 55 - II. 平衡二叉树（中等） 比较典型，求解的答案并非递归返回的值 （已讲）

输入一棵二叉树的根节点，判断该树是不是平衡二叉树。如果某二叉树中任意节点的左右子树的深度相差不超过1，那么它就是一棵平衡二叉树。

示例 1:

给定二叉树 `[3,9,20,null,null,15,7]`

```
    3
   / \
  9  20
    /  \
   15   7
```

返回 `true` 。

示例 2:

给定二叉树 `[1,2,2,3,3,null,null,4,4]`

```
       1
      / \
     2   2
    / \
   3   3
  / \
 4   4
```

返回 `false` 。

原问题：验证一个棵树是不是BT?

子问题1：左子树是或者不是平衡二叉树

子问题2：右子树是或者不是平衡二叉树

子问题是否能推导出原问题的答案?

转化成：求树的高度（或深度）

二叉树相关题型/套路

王争的算法训练营

题型3：二叉树上的递归 剑指 Offer 55 - II. 平衡二叉树（中等） 比较典型，求解的答案并非递归返回的值

- 返回值并非要求解的值
- 要求解的值放在全局变量中

```
class Solution {
    private boolean balanced = true;

    public boolean isBalanced(TreeNode root) {
        height(root);
        return balanced;
    }

    private int height(TreeNode root) {
        if (root == null) return 0;
        if (balanced == false) return 0; // 提前终止递归
        int leftHeight = height(root.left);
        int rightHeight = height(root.right);
        if (Math.abs(leftHeight-rightHeight) > 1) {
            balanced = false;
        }
        return Math.max(leftHeight, rightHeight)+1;
    }
}
```

二叉树相关题型/套路

王争的算法训练营

题型3：二叉树上的递归

617. 合并二叉树（简单） 二叉树的简单递归

类似合并两个链表

给定两个二叉树，想象当你将它们中的一个覆盖到另一个上时，两个二叉树的一些节点便会重叠。

你需要将他们合并为一个新的二叉树。合并的规则是如果两个节点重叠，那么将他们的值相加作为节点合并后的新值，否则**不为** NULL 的节点将直接作为新二叉树的节点。

示例 1:

输入:

```
        Tree 1                     Tree 2
          1                         2
         / \                       / \
        3   2                     1   3
       /                           \   \
      5                             4   7
```

输出:

合并后的树:

```
             3
            / \
           4   5
          / \   \
         5   4   7
```

注意: 合并必须从两个树的根节点开始。

王争的算法训练营

```
class Solution {
    public TreeNode mergeTrees(TreeNode t1, TreeNode t2) {
        if (t1 == null) return t2;
        if (t2 == null) return t1;

        TreeNode newNode = new TreeNode(t1.val+t2.val);
        // merge左子树
        TreeNode leftRoot = mergeTrees(t1.left, t2.left);
        // merge右子树
        TreeNode rightRoot = mergeTrees(t1.right, t2.right);
        // 拼接root、左子树、右子树
        newNode.left = leftRoot;
        newNode.right = rightRoot;
        return newNode;
    }
}
```

二叉树相关题型/套路

王争的算法训练营

题型3：二叉树上的递归

226. 翻转二叉树 （简单）

翻转一棵二叉树。

示例：

输入：

```
     4
   /   \
  2     7
 / \   / \
1   3 6   9
```

输出：

```
     4
   /   \
  7     2
 / \   / \
9   6 3   1
```

题型3：二叉树上的递归

226. 翻转二叉树 （简单）

```
class Solution {
    public TreeNode invertTree(TreeNode root) {
        if (root == null) return root;
        TreeNode leftNode = invertTree(root.left);
        TreeNode rightNode = invertTree(root.right);
        root.right = leftNode;
        root.left = rightNode;
        return root;
    }
}
```

二叉树相关题型/套路

题型3：二叉树上的递归

101. 对称二叉树（中等）

给定一个二叉树，检查它是否是镜像对称的。

例如，二叉树 `[1,2,2,3,4,4,3]` 是对称的。

```
    1
   / \
  2   2
 / \ / \
3  4 4  3
```

但是下面这个 `[1,2,2,null,3,null,3]` 则不是镜像对称的:

```
    1
   / \
  2   2
   \   \
   3    3
```

进阶：

你可以运用递归和迭代两种方法解决这个问题吗?

子问题：左子树镜像对称
子问题：右子树镜像对称
原问题：树是镜像对称的
子问题无法推导出原问题

判断一个二叉树是镜像对称的
等价于：
判断左右子树是镜像对称的

二叉树相关题型/套路

题型3：二叉树上的递归

101. 对称二叉树（中等）

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
      if (root == null) return true;
      return isSymmetric(root.left, root.right);
    }

    private boolean isSymmetric(TreeNode p, TreeNode q) {
      if (p == null && q == null) return true;
      if (p != null && q != null && p.val == q.val) {
        return isSymmetric(p.right, q.left) && isSymmetric(p.left, q.right);
      }
      return false;
    }
}
```

二叉树相关题型/套路

王争的算法训练营

题型3：二叉树上的递归

98. 验证二叉搜索树（中等）

给定一个二叉树，判断其是否是一个有效的二叉搜索树。

假设一个二叉搜索树具有如下特征：

- 节点的左子树只包含**小于**当前节点的数。
- 节点的右子树只包含**大于**当前节点的数。
- 所有左子树和右子树自身必须也是二叉搜索树。

示例 1:

```
输入:
    2
   / \
  1   3
输出: true
```

示例 2:

```
输入:
    5
   / \
  1   4
     / \
    3   6
输出: false
解释: 输入为: [5,1,4,null,null,3,6]。
     根节点的值为 5 ，但是其右子节点值为 4 。
```

子问题：左子树是BST?
子问题：右子树是BST?
原问题：二叉树是BST?

问题转化成：获取二叉树的最大值、最小值—》适合用递归来实现

```java
class Solution {
    private boolean isValid = true;
    public boolean isValidBST(TreeNode root) {
      if (root == null) return true;
      dfs(root);
      return isValid;
    }

    private int[] dfs(TreeNode root) {
      int min = root.val;
      int max = root.val;
      if (root.left != null) {
        int[] leftMinMax = dfs(root.left);
        if (isValid == false) return null;//提前退出条件
        if (leftMinMax[1] >= root.val) {
          isValid = false;
          return null;
        }
        min = leftMinMax[0];
      }
      if (root.right != null) {
        int[] rightMinMax = dfs(root.right);
        if (isValid == false) return null;//提前退出条件
        if (rightMinMax[0] <= root.val) {
          isValid = false;
          return null;
        }
        max = rightMinMax[1];
      }
      return new int[] {min, max};
    }
}
```

- 返回值并非要求解的值
- 要求解的值放在全局变量中

习题课：二叉树

题型4：二叉查找树

剑指 Offer 54. 二叉搜索树的第k大节点（中等）

538. 把二叉搜索树转换为累加树 （中等）

面试题 04.06. 后继者（中等）

二叉树相关题型/套路

王争的算法训练营

题型4：二叉查找树

剑指 Offer 54. 二叉搜索树的第k大节点（中等）**已讲**

给定一棵二叉搜索树，请找出其中第k大的节点。

借助二叉查找树中序遍历结果为有序序列的特点来解题

1. 左-根-右 从小到大排列
2. 右-根-左 从大到小排列

示例 1:

```
输入: root = [3,1,4,null,2], k = 1
   3
  / \
 1   4
  \
   2
输出: 4
```

示例 2:

```
输入: root = [5,3,6,2,4,null,null,1], k = 3
       5
      / \
     3   6
    / \
   2   4
  /
 1
输出: 4
```

二叉树相关题型/套路

题型4：二叉查找树 剑指 Offer 54. 二叉搜索树的第k大节点（中等）

```
class Solution {
    int count = 0;
    int result;
    public int kthLargest(TreeNode root, int k) {
        inorder(root, k);
        return result;
    }

    private void inorder(TreeNode root, int k) {
        if (root == null) return;
        inorder(root.right, k);//返回点一
        if (count >= k) return; // 提前终止递归，剪枝
        count++;
        if (count == k) {
            result = root.val;
            return; // 提前终止递归，剪枝
        }
        inorder(root.left, k);//返回点二
    }
}
```

找到之后，只归不递，拦截返回点之后的逻辑

习题课：二叉树

题型4：二叉查找树

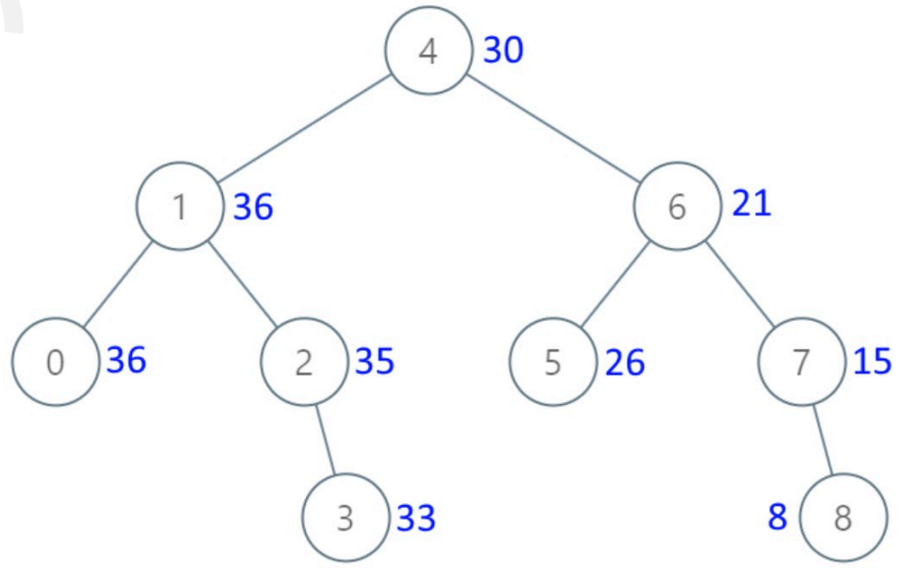
538. 把二叉搜索树转换为累加树 （中等）

给出二叉 **搜索** 树的根节点，该树的节点值各不相同，请你将其转换为累加树（Greater Sum Tree），使每个节点 `node` 的新值等于原树中大于或等于 `node.val` 的值之和。

提醒一下，二叉搜索树满足下列约束条件：

- 节点的左子树仅包含键 **小于** 节点键的节点。
- 节点的右子树仅包含键 **大于** 节点键的节点。
- 左右子树也必须是二叉搜索树。

示例 1：




```
输入：[4,1,6,0,2,5,7,null,null,null,3,null,null,null,8]
输出：[30,36,21,36,35,26,15,null,null,null,33,null,null,null,8]
```

题型4：二叉查找树

538. 把二叉搜索树转换为累加树 （中等）

```
class Solution {
    private int sum = 0;

    public TreeNode convertBST(TreeNode root) {
        inorder(root);
        return root;
    }

    public void inorder(TreeNode root) {
        if (root == null) return;
        inorder(root.right);
        sum += root.val;
        root.val = sum;
        inorder(root.left);
    }
}
```

习题课：二叉树

王争的算法训练营

题型4：二叉查找树

面试题 04.06. 后继者（中等）

设计一个算法，找出二叉搜索树中指定节点的“下一个”节点（也即中序后继）。

如果指定节点没有对应的“下一个”节点，则返回 `null`。

示例 1:

```
输入: root = [2,1,3], p = 1

  2
 / \
1   3

输出: 2
```

示例 2:

```
输入: root = [5,3,6,2,4,null,null,1], p = 6

      5
     / \
    3   6
   / \
  2   4
 /
1

输出: null
```

习题课：二叉树

题型4：二叉查找树

面试题 04.06. 后继者（中等）

```
class Solution {
    private boolean comming = false;
    private TreeNode successor = null;
    public TreeNode inorderSuccessor(TreeNode root, TreeNode p) {
        inorder(root, p);
        return successor;
    }

    private void inorder(TreeNode root, TreeNode p) {
        if (root == null) return;
        inorder(root.left, p);
        if (successor != nlll) return; // 提前退出条件
        if (comming == true) {
            successor = root;
            comming = false;
            return;
        }
        if (root == p) comming = true;
        inorder(root.right, p);
    }
}
```

提问环节

关注微信公众号“**小争哥**”，

后台回复“**PDF**”获取独家算法资料