二叉树解题思路小结：

- 在二叉树解题过程中，应尽量使用递归来进行处理，在递归的第一行代码必须为：
  `if(node == nullptr) return nullptr;`
  或者
  `if(node == nullptr) returan 0;`
  经过上述的判断后，才能继续下面的递归过程。

- 在二叉树解题过程中，使用递归的时候应尽量避免以二叉树结点为考虑单位，而应该以子二叉树来作为思路的考量单位，这样可以使自己对接下来的递归过程，在理解上更加容易掌握。

- 在遍历二叉树的过程中，如果出现return 1则表示找到了符合题意的情况，从而向上累积加1

二叉树类型：

- 平衡二叉树：任意结点的左右子树高度差的绝对值不超过1

- 完全二叉树：1）叶结点只可能出现在深度最大的两层上，对于最大层次中的叶结点，都一次排列在该层最左边的位置上 2）如果有度为1的结点，只可能有一个，且该结点只有左孩子结点，没有右孩子结点

  下列二叉树数为完全二叉树：

```
          1

     /        \
    2          3
   /  \       / \
  4    5     6   7
 / \  / \   /
8  9 10 11 12
```

常见二叉树算法：

1、求二叉树的最大深度：

```
int maxDeath(TreeNode node){
    if(node==null){
        return 0;
    }
    int left = maxDeath(node.left);
    int right = maxDeath(node.right);
    return Math.max(left,right) + 1;
}
```

2、求二叉树的最小深度：

```
int getMinDepth(TreeNode root){
        if(root == null){
            return 0;
        }
        return getMin(root);
    }
    int getMin(TreeNode root){
        if(root == null){
            return Integer.MAX_VALUE;
        }
        if(root.left == null&&root.right == null){
            return 1;
        }
        return Math.min(getMin(root.left),getMin(root.right)) + 1;
}
```

3、求二叉树中结点的数量：

```
int numOfTreeNode(TreeNode root){
        if(root == null){
            return 0;
        }
        int left = numOfTreeNode(root.left);
        int right = numOfTreeNode(root.right);
        return left + right + 1;
}
```

4、求二叉树中叶结点的数量：

```
int numsOfNoChildNode(TreeNode root){
        if(root == null){
            return 0;
        }
        if(root.left==null&&root.right==null){
            return 1;
        }
        return numsOfNodeTreeNode(root.left)+numsOfNodeTreeNode(root.right);
}
```

5、求二叉树中第k层结点的数量：

```
int numsOfkLevelTreeNode(TreeNode root,int k){
            if(root == null||k<1){
                return 0;
            }
            if(k==1){
                return 1;
            }
            int numsLeft = numsOfkLevelTreeNode(root.left,k-1);
            int numsRight = numsOfkLevelTreeNode(root.right,k-1);
            return numsLeft + numsRight;
}
```

6、判断二叉树是否是平衡二叉树：

```
 boolean isBalanced(TreeNode node){
        return maxDeath2(node)!=-1;
    }
    int maxDeath2(TreeNode node){
        if(node == null){
            return 0;
        }
        int left = maxDeath2(node.left);
        int right = maxDeath2(node.right);
        if(left==-1||right==-1||Math.abs(left-right)>1){
            return -1;
        }
        return Math.max(left, right) + 1;
}
```

7、判断二叉树是否是完全二叉树：

```
boolean isCompleteTreeNode(TreeNode root){
        if(root == null){
            return false;
        }
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.add(root);
        boolean result = true;
        boolean hasNoChild = false;
        while(!queue.isEmpty()){
            TreeNode current = queue.remove();
            if(hasNoChild){
                if(current.left!=null||current.right!=null){
                    result = false;
                    break;
                }
            }else{
                if(current.left!=null&&current.right!=null){
                    queue.add(current.left);
                    queue.add(current.right);
                }else if(current.left!=null&&current.right==null){
                    queue.add(current.left);
                    hasNoChild = true;
                }else if(current.left==null&&current.right!=null){
                    result = false;
                    break;
                }else{
                    hasNoChild = true;
                }
            }
        }
        return result;
}
```

8、两个二叉树是否完全相同：

```
boolean isSameTreeNode(TreeNode t1,TreeNode t2){
        if(t1==null&&t2==null){
            return true;
        }
        else if(t1==null||t2==null){
            return false;
        }
        if(t1.val != t2.val){
            return false;
        }
        boolean left = isSameTreeNode(t1.left,t2.left);
        boolean right = isSameTreeNode(t1.right,t2.right);
        return left&&right;
}
```

9、两个二叉树是互为镜像：

```
boolean isMirror(TreeNode t1,TreeNode t2){
        if(t1==null&&t2==null){
            return true;
        }
        if(t1==null||t2==null){
            return false;
        }
        if(t1.val != t2.val){
            return false;
        }
        return isMirror(t1.left,t2.right)&&isMirror(t1.right,t2.left);
}
```

10、翻转二叉树或者镜像二叉树：

```
TreeNode mirrorTreeNode(TreeNode root){
        if(root == null){
            return null;
        }
        TreeNode left = mirrorTreeNode(root.left);
        TreeNode right = mirrorTreeNode(root.right);
        root.left = right;
        root.right = left;
        return root;
}
```

11、求两个二叉树的最低公共祖先结点：

```
    TreeNode getLastCommonParent(TreeNode root,TreeNode t1,TreeNode t2){
        if(findNode(root.left,t1)){
            if(findNode(root.right,t2)){
                return root;
            }else{
                return getLastCommonParent(root.left,t1,t2);
            }
        }else{
            if(findNode(root.left,t2)){
                return root;
            }else{
                return getLastCommonParent(root.right,t1,t2)
            }
        }
    }
    // 查找节点node是否在当前 二叉树中
    boolean findNode(TreeNode root,TreeNode node){
        if(root == null || node == null){
            return false;
        }
        if(root == node){
            return true;
        }
        boolean found = findNode(root.left,node);
        if(!found){
            found = findNode(root.right,node);
        }
        return found;
    }
```

12、 二叉树的前序遍历：

```
// 迭代解法
  ArrayList<Integer> preOrder(TreeNode root){
        Stack<TreeNode> stack = new Stack<TreeNode>();
        ArrayList<Integer> list = new ArrayList<Integer>();
        if(root == null){
            return list;
        }
        stack.push(root);
        while(!stack.empty()){
            TreeNode node = stack.pop();
            list.add(node.val);
            if(node.right!=null){
                stack.push(node.right);
            }
            if(node.left != null){
                stack.push(node.left);
            }
        }
        return list;
    }
// 递归算法
  ArrayList<Integer> preOrderReverse(TreeNode root){
        ArrayList<Integer> result = new ArrayList<Integer>();
        preOrder2(root,result);
        return result;
    }
    void preOrder2(TreeNode root,ArrayList<Integer> result){
        if(root == null){
            return;
        }
        result.add(root.val);
        preOrder2(root.left,result);
        preOrder2(root.right,result);
    }
```

13、二叉树的中序遍历：

```
  ArrayList<Integer> inOrder(TreeNode root){
        ArrayList<Integer> list = new ArrayList<<Integer>();
        Stack<TreeNode> stack = new Stack<TreeNode>();
        TreeNode current = root;
        while(current != null|| !stack.empty()){
            while(current != null){
                stack.add(current);
                current = current.left;
            }
            current = stack.peek();
            stack.pop();
            list.add(current.val);
            current = current.right;
        }
        return list;
    }
```

14、二叉树的后序遍历：

```
  ArrayList<Integer> postOrder(TreeNode root){
        ArrayList<Integer> list = new ArrayList<Integer>();
        if(root == null){
            return list;
        }
        list.addAll(postOrder(root.left));
        list.addAll(postOrder(root.right));
        list.add(root.val);
        return list;
    }
```

15、二叉树的层次遍历：

```
  ArrayList<ArrayList<Integer>> levelOrder(TreeNode root){
        ArrayList<ArrayList<Integer>> result = new
     ArrayList<ArrayList<Integer>>();
        if(root == null){
            return result;
        }
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        queue.offer(root);
        while(!queue.isEmpty()){
            int size = queue.size();
            ArrayList<<Integer> level = new ArrayList<Integer>():
            for(int i = 0;i < size ;i++){
                TreeNode node = queue.poll();
                level.add(node.val);
                if(node.left != null){
                    queue.offer(node.left);
                }
                if(node.right != null){
                    queue.offer(node.right);
                }
            }
            result.add(Level);
        }
        return result;
    }
```

16、不同的二叉树：

```
// 给出 n，问由 1…n 为节点组成的不同的二叉查找树有多少种？
int numTrees(int n ){
        int[] counts = new int[n+2];
        counts[0] = 1;
        counts[1] = 1;
        for(int i = 2;i<=n;i++){
            for(int j = 0;j<i;j++){
                counts[i] += counts[j] * counts[i-j-1];
            }
        }
        return counts[n];
}
```

17、输入一个二叉树和一个整数，打印出二叉树中节点值的和等于输入整数所有的路径：

```
void findPath(TreeNode r,int i){
        if(root == null){
            return;
        }
        Stack<Integer> stack = new Stack<Integer>();
        int currentSum = 0;
        findPath(r, i, stack, currentSum);
}
void findPath(TreeNode r,int i,Stack<Integer> stack,int currentSum){
        currentSum+=r.val;
        stack.push(r.val);
        if(r.left==null&&r.right==null){
            if(currentSum==i){
                for(int path:stack){
                    System.out.println(path);
                }
            }
        }
        if(r.left!=null){
            findPath(r.left, i, stack, currentSum);
        }
        if(r.right!=null){
            findPath(r.right, i, stack, currentSum);
        }
        stack.pop();
}
```

18、前序遍历和后序遍历构造二叉树：

```
TreeNode buildTreeNode(int[] preorder,int[] inorder){
        if(preorder.length!=inorder.length){
            return null;
        }
        return
     myBuildTree(inorder,0,inorder.length-1,preorder,0,preorder.length-1);
}
TreeNode myBuildTree(int[] inorder,int instart,int inend,int[] preorder,int
     prestart,int preend){
        if(instart>inend){
            return null;
        }
        TreeNode root = new TreeNode(preorder[prestart]);
        int position = findPosition(inorder,instart,inend,preorder[start]);
        root.left =
     myBuildTree(inorder,instart,position-1,preorder,prestart+1,prestart+position
     -instart);
        root.right = myBuildTree(inorder,position+1,inend,preorder,position-
     inend+preend+1,preend);
        return root;
}
int findPosition(int[] arr,int start,int end,int key){
        int i;
        for(i = start;i<=end;i++){
            if(arr[i] == key){
                return i;
            }
        }
        return -1;
}
```

19、在二叉树中插入结点：

```
TreeNode insertNode(TreeNode root,TreeNode node){
        if(root == node){
            return node;
        }
        TreeNode tmp = new TreeNode();
        tmp = root;
        TreeNode last = null;
        while(tmp!=null){
            last = tmp;
            if(tmp.val>node.val){
                tmp = tmp.left;
            }else{
                tmp = tmp.right;
            }
        }
        if(last!=null){
            if(last.val>node.val){
                last.left = node;
            }else{
                last.right = node;
            }
        }
        return root;
}
```