二叉树的剑指Offer一些题目

# 二叉树的下一个节点getNextNode

## 题目

给定一个二叉树和其中的一个结点，请找出**中序遍历**顺序的下一个结点并且返回。

**注意**，树中的结点不仅包含左右子结点，同时包含指向父结点的指针。

二叉树的数据结构：

public class TreeLinkNode {

int val;

TreeLinkNode left = null;

TreeLinkNode right = null;

TreeLinkNode next = null;//指向父节点

TreeLinkNode(int val) {

this.val = val;

}

}

## 思路分析

分成三个大类：

### 当前节点为null，则返回null；

### 当前节点有右子树，则返回右子树的最左子节点；

### 当前节点无右子树，分成两种情况：①当前节点为左子节点，则直接返回当前节点的父节点；②当前节点不是左子节点(也就是右子节点)，则不断向上寻找父节点，直到父节点为左子节点为止，返回该父节点的父节点，若寻找过程中父节点为null，则直接返回null。

## Java代码

/\*\*

\* 获取二叉树的下一个节点

\*/

public TreeLinkNode getNext(TreeLinkNode pNode){

//分成三种情况：当前节点**①为null;②有右子树;③无右子树，再根据左子节点和右子节点分成两类**。

**if(pNode == null) return null;**

if(pNode.right != null){//有右子树

pNode = pNode.right;

while(pNode.left != null){//寻找右子树的最左节点

pNode = pNode.left;

}

return pNode;

}else{//没有右子树

TreeLinkNode parent = pNode.next;//父节点

**if(parent == null) return null;//特殊情况:只有一个根节点**

if(pNode == parent.left){//当前节点为左子节点

return parent;

}else{//当前节点为右子节点

//中序遍历的最后一个元素，会出现parent.next 为null，需要返回null

**while(parent.next != null && parent != parent.next.left){**

**parent = parent.next;**

**}**

return parent.next;

}

}

}

# 二叉树中和为某一值的路径

## 题目介绍

题目描述：输入一颗二叉树的跟节点和一个整数，**打印出二叉树中结点值的和为输入整数的所有路径**。路径定义为从树的根结点开始往下一直到叶结点所经过的结点形成一条路径。(注意: 在返回值的list中，数组长度大的数组靠前)

## 思路介绍

\* 题目：寻找二叉树中和为某一值的所有路径

\* 思路：首先确定是一条路径的条件：叶子节点且当前值等于target

\* 即(root.left==null&&root.right==null)&&(currentSum==target)

\* 然后根据这个条件的其他情况分析：

\* 1. 叶子节点但当前值不等于target;只需要tempList删除当前节点值即可；

\* 2. 左节点存在，递归左节点；

\* 3. 右节点存在，递归右节点。

\* **注意：由于currentSum是int传递，因此，在递归过程跳出时，不需要减去当前节点值；但是对于tempList是个对象，需要在退出函数之前删除当前节点的值。**

## Java代码

/\*\*

\* 题目：寻找二叉树中和为某一值的所有路径

\* 思路：首先确定是一条路径的条件：叶子节点且当前值等于target

\* 即(root.left==null&&root.right==null)&&(currentSum==target)

\* 然后根据这个条件的其他情况分析：

\* 1. 叶子节点但当前值不等于target;只需要tempList删除当前节点值即可；

\* 2. 左节点存在，递归左节点；

\* 3. 右节点存在，递归右节点。

\* 注意：由于currentSum是int传递，因此，在递归过程跳出时，不需要减去当前节点值；

\* 但是对于tempList是个对象，需要在退出函数之前删除当前节点的值。

\*/

public ArrayList<ArrayList<Integer>> FindPath(TreeNode root, int target) {

ArrayList<ArrayList<Integer>> lists = new ArrayList<ArrayList<Integer>>();

if(root == null) return lists;

ArrayList<Integer> tempList = new ArrayList<Integer>();

findPathHandler(root,tempList,lists,target,0);

// System.out.println(tempList.toString());//最后结果tempList一定是空的[]

return lists;

}

/\*\*

\* 递归方法

\*/

public void findPathHandler(TreeNode root,ArrayList<Integer> tempList, ArrayList<ArrayList<Integer>> lists,int target,int currentSum){

currentSum += root.val;//当前访问路径的和

tempList.add(root.val);//将该值存入list集合中

//找到一条路径的条件：当前节点为叶子节点且路径之和为目标值

//如果是叶子节点但不等于目标值，只需要tempList删除当前节点值即可

if((root.left==null&&root.right==null)&&(currentSum==target)){

ArrayList<Integer> newList = new ArrayList<Integer>();

for(int i = 0;i < tempList.size();i++) newList.add(0);//初始化

Collections.copy(newList,tempList);//复制tempList至newList

lists.add(newList);//添加一条路径

}

if(root.left != null)

findPathHandler(root.left,tempList,lists,target,currentSum);

if(root.right != null)

findPathHandler(root.right,tempList,lists,target,currentSum);

**tempList.remove(tempList.size()-1);//很重要**

}