二叉树递归——返回值与不需返回值?

2022年9月15日 19:01

此题以leetcode113.路径总和2 、 leetcode 236.二叉树的最近公共祖先与leetcode.112路径总和作为引入
递归函数什么时候需要返回值什么时候不需要呢?
递归的三部曲:
一、确定递归函数的参数和返回类型
这引申出一个问题,函数需要返回值吗?(我们稍后再说)
二、确定终止条件
三、确定单层递归的逻辑
注意在这个步骤中,并不需要你去模拟每一次递归的实现,而是抽象出来,这层递归处理什么?需要给上一层递归返回什么?把握这两点就足够了

回到开始的问题,递归函数需要返回值吗?
如果需要搜索整颗二叉树而且不用处理递归返回值,那么递归函数就不要返回值。例如113.路径总和2
如果需要搜索整颗二叉树且需要处理递归返回值,那么递归函数就需要返回值。例如236.二叉树的最近公共祖先
如果要搜索其中一条符合条件的路径,那么递归一定需要返回值。例如112.路径总和
(因为遇到符合条件的路径就要及时返回!)

那么在二叉树以外,私以为函数是否需要返回值,关键在于单层递归是否需要处理上一层递归的返回值


113. 路径总和 II

难度 中等 841

给你二叉树的根节点 root 和一个整数目标和 targetSum ,找出所有 **从根节点到叶子节点** 路径总和等于给定目标和的路径。

叶子节点 是指没有子节点的节点。

首先,这题需要返回所有从根节点到叶子节点的路经总和满足条件的解
可以使用递归求解
一、确定函数的返回值,由于并不需要返回值,因此函数类型定义为void,且在每次递归中只需要传入节点指针与目标值

```
void traversal(TreeNode* node,int count)
```

二、终止条件为目前节点cur为叶子节点,并判断当前节点的值cur->val是否与目标值相等

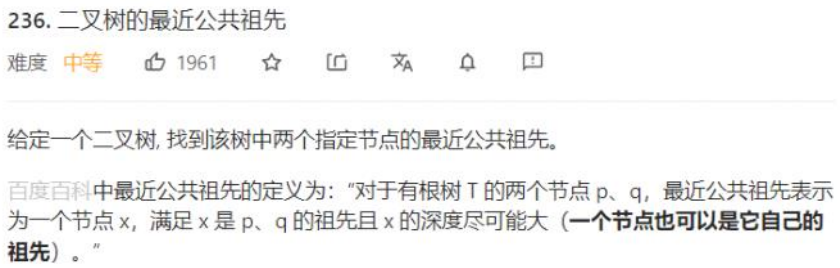
```
if(!node->left&&!node->right){
    if(!count){
        result.push_back(temp);
    }
    return;
}
```

三、单层递归的逻辑,如果左节点不为空,则对左子树进行搜索;如果右节点不为空则对右节点进行搜索。搜索完毕后直接返回 return;

```
if(node->left){
    temp.push_back(node->left->val);
    traversal(node->left,count-node->left->val);
    temp.pop_back();
}
if(node->right){
    temp.push_back(node->right->val);
    traversal(node->right,count-node->right->val);
    temp.pop_back();
}
return;
```

注意，在搜索中需要一个vector<int>来记录路径，注意在递归过程中的回溯处理！

若满足的条件的话，则将vector<int> 加入 vector<vector<int>> ans 中



此题看似需要找到符合条件的某一个节点即可，即只要这个节点的左（右）子树含有节点p、右（左）子树含有节点q那么就可以说该节点是两个指定节点的最近公共祖先

注意！还有一种情况是，节点p/q是所说的当前节点本身！说白了就是q在p的子树或者p在q的子树这种情况

因此这是需要返回值的！返回值应是节点指针即Treenode* 类型

但真的是这样吗？实际上我们需要搜索整颗二叉树！在三部曲结束后再说明

我们使用递归进行深度搜索

一、返回节点指针类型，传入参数为指定节点指针、根节点指针

```
TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q)
```

二、确定终止条件，若我们找到空节点说明没有搜索到指定节点，返回空节点，若我们找到了指定节点则返回指定节点！（为进行单层递归时，对上层递归做返回值处理做铺垫）

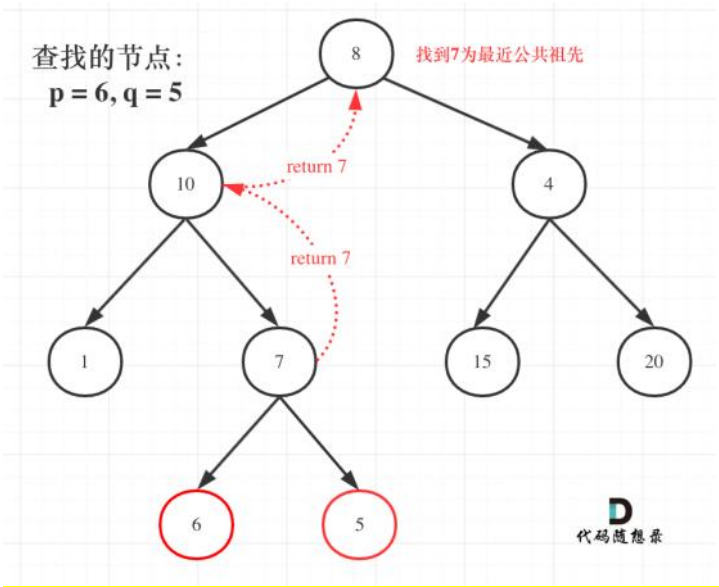
```
if(root==p||root==q||root==NULL) return root;
```

三、单层递归，该题应该从下往上搜索即回溯，而二叉树的后序遍历天然是回溯

因此我们用left接收当前节点cur左子树的返回值，right接收当前节点cur右子树的返回值，若left、right均不为空，则返回当前节点；若left、right均为空则返回空；若left为空则返回right；若right为空则返回left

```
if(left&&right){
    return root;
}
else if(left){
    return left;
}
else if(right){
    return right;
}
return nullptr;
```

现在来说明为什么是搜索整颗二叉树



如图所示，找到6和5的时候，直接返回7不可以吗？实际上要等待根节点右子树的搜索，即图中节点4，15，20

因为在通过left、right进行逻辑处理时，不能直接返回，要等到left、right逻辑处理完毕才能返回，因此实际上是搜索整颗二叉树！

---

112. 路径总和

难度 简单 996

给你二叉树的根节点 `root` 和一个表示目标和的整数 `targetSum` 。判断该树中是否存在 **根节点到叶子节点** 的路径，这条路径上所有节点值相加等于目标和 `targetSum` 。如果存在，返回 `true` ；否则，返回 `false` 。

叶子节点 是指没有子节点的节点。

这题只需要找到一条符合条件的路径即可，因此不需要搜索整颗二叉树，可以通过深度搜索的方式来查找，因此通过递归解题

一、由于符合条件即可停止，因此函数需要返回值，类型为bool类型，参数为节点指针与目标值

```
bool hasPathSum(TreeNode* root, int targetSum)
```

二、当找到叶子节点的时候，即终止搜索，并判断当前节点值cur->val 是否与目标值相等

```
if(!root->left&&!root->right){
    if(root->val==targetSum)
        return true;
    else
        return false;
}
```

若相等则返回true，反之则返回false

三、单层递归的逻辑，若左节点不为空，则对左子树进行搜索；若右节点不为空，则对右节点进行搜索。

注意需要对上层递归的返回值进行处理！若在上述两个搜索中找到了路径，则返回true，若都没找到则返回false

```
if(root->left){
    if(hasPathSum(root->left, targetSum-root->val))
        return true;
}
if(root->right){
    if(hasPathSum(root->right, targetSum-root->val)){
        return true;
    }
}
return false;
```

注意！此题不需要另外写一个递归函数，但是要防止root根节点为空的情况

因此在前面加上一个判断，若当前节点为空则直接return false；

```
bool hasPathSum(TreeNode* root, int targetSum) {
    if(!root)
        return false;
    if(!root->left&&!root->right){
        if(root->val==targetSum)
            return true;
        else
            return false;
    }
    if(root->left){
        if(hasPathSum(root->left, targetSum-root->val))
            return true;
    }
    if(root->right){
        if(hasPathSum(root->right, targetSum-root->val)){
            return true;
        }
    }
    return false;

}
```