链滴

重建二叉树, 二叉树中和为某一值的路径

作者：ReyRen
原文链接：https://ld246.com/article/1570791164702
来源网站：链滴

题目一:

输入某二叉树的前序遍历和中序遍历的结果, 请重建出该二叉树. 假设输入的前序遍历和中序遍历的结中都不含重复的数字. 例如输入前序遍历序列{1,2,4,7,3,5,6,8}和中序遍历序列{4,7,2,1,5,3,8,6}, 则重建叉树并返回.

时间限制:

1秒

空间限制:

32768K

解题思路:

首先看到树的遍历, 重构, 排序等, 一般就是经典的递归的使用(左右子树, 一直让递归下去. 也比较好理解.

再看到前序遍历了的数组, 首先能想到的是数组的第一个元素就是根节点.

题目中提到不包含重复的节点值. 所以可以从中序入手找到root节点就能知道, 在中序数组中左面的中序左子树, 右面的是中序右子树.

这样根据左右子树个树, 在前序数组中也能得到前序左子树和前序右子树的数组.

最后每个子树中依然上一开始的思路构建个字的tree就行了.

```
/**
 * Definition for binary tree
 * struct TreeNode {
```

```
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode(int x) : val(x), left(NULL), right(NULL) {}
 * };
 */
class Solution {
public:
    TreeNode *reConstructBinaryTree(vector<int> pre,vector<int> vin) {
        int preLen = pre.size();
        int vinLen = vin.size();
        if (preLen == 0 || vinLen == 0) {
            return NULL;
        }

        TreeNode *root = new TreeNode(pre[0]);//root节点

        /*通过中序数组算出左右子树个数,这样才能让前序也能得到前序的左树*/
        int genPos = 0;
        while (vin[genPos] != pre[0]) {
            genPos++;
        }
        //此时的genPos是带根的

        vector<int> preLeft, preRight, vinLeft, vinRight;

        /*左子树走一波*/
        for (int i = 0; i < genPos; i++) {
            vinLeft.push_back(vin[i]);
            preLeft.push_back(pre[i + 1]);//跳过根
        }
        /*右子树走一波*/
        for (int i = genPos + 1; i < vinLen; i++) {
            vinRight.push_back(vin[i]);
            preRight.push_back(pre[i]);
        }

        /*root的左右子节点就是左右子树的root*/
        root->left = reConstructBinaryTree(vinLeft, preLeft);
        root->right = reConstructBinaryTree(vinRight, preRight);

        return root;
    }
}
```

题目二:

输入一颗二叉树的根节点和一个整数, 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, 数组长度大的组靠前)

时间限制:

1秒

空间限制:

32768K

解题思路:

DFS的题. 但是需要注意的是当深度结束后, 要不断的回退到上一个结点.

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
    vector<vector<int>> res;
    vector<int> list;
public:
    void dfs(TreeNode* node, int left_number) {
        list.push_back(node->val);
        left_number = left_number - node->val;
        if (left_number == 0 && node->left == NULL && node->right == NULL) {
            res.push_back(list);
        }
        else {
            if (node->left) {
                dfs(node->left, left_number);
            }
            if (node->right) {
                dfs(node->right, left_number);
            }
        }
        list.pop_back();//这里很有意思
    }
    vector<vector<int> > FindPath(TreeNode* root,int expectNumber) {
        if(root) {
            (void)dfs(root, expectNumber);
        }
        return res;
    }
```

小知识点□+1:

中序遍历

中序遍历首先遍历左子树, 然后访问根结点, 最后遍历右子树

前序遍历

前序遍历首先访问根结点然后遍历左子树, 最后遍历右子树. 在遍历左、右子树时，仍然先访问根结点, 然后遍历左子树, 最后遍历右子树.

后续遍历

后序遍历首先遍历左子树, 然后遍历右子树, 最后访问根结点, 在遍历左、右子树时，仍然先遍历左子树, 然后遍历右子树, 最后遍历根结点.