链滴

树的子结构, 二叉树的镜像, 树的遍历, 二叉搜索树的后序遍历序列

作者：ReyRen

原文链接：https://ld246.com/article/1569329965517

来源网站：链滴

题目一:

输入两棵二叉树A, B, 判断B是不是A的子结构(ps:我们约定空树不是任意一个树的子结构)

时间限制:

1秒

空间限制:

32768K

解题思路:

B是不是A的子结构, 也就是说以A中的某一个结点(包括根节点)为B的根节点. 并且直到遍历比较完B的个结点前, A都不可能遍历结束.

之前也提到过, 树类型的题目, 使用递归更好理解和实现. 这道题也是不例外的. 注意这里有个很有意思坑. 见代码注解:

```
/*
 * struct TreeNode {
 *     int val;
 *     struct TreeNode* left;
 *     struct TreeNode* right;
 *     TreeNode (int x) :
 *       val(x), left(NULL), right(NULL);
 * }
 */
class Solution {
public:
    /*找头部配*/
    bool HasSubtree(TreeNode* pRoot1, TreeNode* pRoot2) {
        bool result = false;
            if (pRoot1 != NULL && pRoot2 != NULL) {
            if (pRoot1->val == pRoot2->val) {
                /*说明头对上了, 接下来进行结点遍历对比喽*/
                result = doesTree1HasTree2(pRoot1, pRoot2);
            }
            /*只能重新找头和pRoot2进行头部匹配喽*/
            if (!result) {
                result = HasSubtree(pRoot1->left, pRoot2);
            }
            if(!result) {
                result = HasSubtree(pRoot1->right, pRoot2);
            }
        }
        return false;
    }
    /*找结点配*/
    bool doesTree1HasTree2(TreeNode* pNode1, TreeNode* pNode2) {
        /*坑!!: 底下这俩if如果先判断pNode1, 测试用例通不过. 是因为有可能p1和
```

```
         *p2同时结束, 这种情况是true的. 所以应该先去判断p2.
         */
        if (pNode2 == NULL) {
            return true;
        }
        if (pNode1 == NULL) {
            return false;
        }
        if (pNode1->val != pNode2->val) {
            return false;
        }
        return doesTree1HasTree2(pNode1->left, pNode2->left) && doesTree1HasTree2 (pNod
1->right, pNode2->right);
        /*让左右子树都进行笔记吧!*/
    }
}
```

题目二:

操作给定的二叉树, 将其变换为源二叉树的镜像.

```
二叉树的镜像定义：源二叉树
            8
           / \
          6   10
         / \  / \
        5  7 9 11
        镜像二叉树
            8
           / \
          10   6
         / \  / \
        11 9 7  5
```

时间限制:

1秒

空间限制:

32768K

解题思路:

递归:依然是树, 使用递归的思想是最为简单易懂的. 在进行镜像的时候, 从上到下观察会发现显示root结点的左子树和右子树互换位置, 然后再以root->left为根节点的左子树和右子树互换, root->right根节点的左子树和右子树互换.

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
```

```
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};
*/
class Solution {
public:
    void Mirror(TreeNode *pRoot) {
        if (pRoot == NULL) {
            return;
        }
        TreeNode* pTmp = pRoot->left;
        pRoot->left = pRoot->right;
        pRoot->right = pTmp;
        if (pRoot->left) {
            Mirror(pRoot->left);
        }
        if (pRoot->right) {
            Mirror(pRoot->right);
        }
    }
}
```

非递归:和递归的思想一致, root的左子树全部换完之后到root的右子树(反之亦然), 这里的root是时变换的. 只要保持这个宗旨深度遍历替换就行.

要想完成这样的过程, 我们需要一个能存取的容器, 将右子树(或左子树)的临时根节点(也就是遍历到哪的那个节点, 之所以称之为临时根节点就是要对这个节点的左右节点进行镜像了)全部都存下来, 这样便全部左子树都完成了镜像后, 返回来再进行右的镜像. 能达到这个需求的显然就是栈了.

```
class Solution {
public:
    void Mirror(TreeNode *pRoot) {
        if (pRoot == NULL) {
            return;
        }
        stack<TreeNode *> pStack;
        pStack.push(pRoot);
        while(pStack.size()) {
            TreeNode *tree = pStack.top();
             pStack.pop();
            TreeNode* pTmp = tree->left;
            tree->left = tree->right;
            tree->right = pTmp;
            if(tree->left) {
                pStack.push(tree->left);
            }
            if(tree->right) {
                pStack.push(tree->right);
            }
        }
    }
}
```

题目三:

从上往下打印出二叉树的每个节点, 同层节点从左至右打印.

时间限制:

1秒

空间限制:

32768K

解题思路:

这个可以借用一个队列来将每层的根节点的左右子节点都打入队列, 这样就可以了.

```
/*
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
    TreeNode(int x) :
            val(x), left(NULL), right(NULL) {
    }
};*/
class Solution {
public:
    vector<int> PrintFromTopToBottom(TreeNode* root) {
        vector<int> res;
        if(root == NULL) {
            return res;
        }
        queue<TreeNode *> q;
        q.push(root);
        while(!q.empty()) {
            res.push_back(q.front()->val);
            if (q.front()->left != NULL) {
                q.push(q.front()->left);
            }
            if (q.front()->right != NULL) {
                q.push(q.front()->right);
            }
            q.pop();
        }
        return res;
    }
}
```

题目四:

输入一个整数数组, 判断该数组是不是某二叉搜索树的后序遍历的结果. 如果是则输出Yes, 否则输出No.

假设输入的数组的任意两个数字都互不相同.

时间限制:

1秒

空间限制:

32768K

解题思路:

首先后续遍历可知数组的最后一个值是root, 那么在数组剩余的子数组中应该要有一个值x, x的左面的全部都小于root(因为后续遍历, 所以左面肯定是左子树范畴), x右面的值全部都大于root(右子树范畴). 完美的递归条件.

```
class Solution {
    bool judge(vector<int> &sequence, int begin, int end) {
        //当begin不断移动的和end甚至比end大(没找到pivot), 都是true的
        if (begin >= end) {
            return true;
        }
        ////找比根大的第一点
        int pivoit = begin;
        while (sequence[pivoit] < sequence[end]) {
            pivoit++;
        }
        //如果左面有一个大于end的就是false
        for (int i = begin; i < pivoit; i++) {
            if (sequence[i] > sequence[end]) {
                return false;
            }
        }
        //同理
        for (int i = pivoit; i < end; i++) {
            if (sequence[i] < sequence[end]) {
                return false;
            }
        }
        return judge (sequence, begin, pivoit - 1) && judge (sequence, pivoit, end - 1);
    }
public:
    bool VerifySquenceOfBST(vector<int> sequence) {
        if (!sequence.size())
            return false;
        }
        return judge(sequence, 0, sequence.size() - 1);
    }
}
```

小知识点:thumbsup:

什么是二叉搜索树

二叉查找树(Binary Search Tree)(又: 二叉搜索树, 二叉排序树)它或者是一棵空树, 或者是具有下列性的二叉树. 若它的左子树不空, 则左子树上所有结点的值均小于它的根结点的值; 若它的右子树不空, 则子树上所有结点的值均大于它的根结点的值; 它的左右子树也分别为二叉排序树

中序遍历

左右中