# 题目

给你一棵二叉搜索树（BST）、它的根结点 root 以及目标值 V。

请将该树按要求拆分为两个子树：其中一个子树结点的值都必须小于等于给定的目标值 V；另一个子树结点的值都必须大于目标值 V；树中并非一定要存在值为 V 的结点。

除此之外，树中大部分结构都需要保留，也就是说原始树中父节点 P 的任意子节点 C，假如拆分后它们仍在同一个子树中，那么结点 P 应仍为 C 的父结点。

你需要返回拆分后两个子树的根结点 TreeNode，顺序随意。

示例：

输入：root = [4,2,6,1,3,5,7], V = 2

输出：[[2,1],[4,3,6,null,null,5,7]]

解释：

注意根结点 output[0] 和 output[1] 都是 TreeNode 对象，不是数组。

给定的树 [4,2,6,1,3,5,7] 可化为如下示意图：

4

/ \

2 6

/ \ / \

1 3 5 7

输出的示意图如下：

4

/ \

3 6 和 2

/ \ /

5 7 1

提示：

二叉搜索树节点个数不超过 50

二叉搜索树始终是有效的，并且每个节点的值都不相同

# 分析

解题思路

判断V在root还是左子树还是右子树，然后进行递归

如果root->val==V的话，记得root->right=NULL(切断)

思路参考见评论

代码

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode(int x) : val(x), left(NULL), right(NULL) {}

\* };

\*/

class Solution {

public:

vector<TreeNode\*> splitBST(TreeNode\* root, int V) {

if(root==NULL)

return {NULL,NULL};

else if(root->val==V){

TreeNode\* tmp=root->right;

root->right=NULL;

return {root,tmp};

}

//左子树

else if(root->val>V){

vector<TreeNode\*> t=splitBST(root->left,V);

root->left=t[1];

t[1]=root;

return t;

}

//右子树

else if(root->val<V){

vector<TreeNode\*> t=splitBST(root->right,V);

root->right=t[0];

t[0]=root;

return t;

}

return {NULL,NULL};

}

};

另一种写法：

class Solution1 {

public:

// a) index 0 存储 <= V 的子树

// b) index 1 存储 > V 的子树

vector<TreeNode\*> splitBST(TreeNode\* root, int V) {

vector<TreeNode\*> res = {nullptr, nullptr};

if (!root) {

return res;

}

if (root->val <= V) {

res = splitBST(root->right, V);

root->right = res[0];

return {root, res[1]}; // the smaller part of right child and root together makes the <= tree

} else {

res = splitBST(root->left, V);

root->left = res[1];

return {res[0], root};

}

}

};

class Solution {

public:

// a) index 0 存储 <= V 的子树

// b) index 1 存储 > V 的子树

vector<TreeNode\*> splitBST(TreeNode\* root, int V) {

vector<TreeNode\*> res = {nullptr, nullptr};

if (!root) {

return res;

}

if (root->val <= V) {

res = splitBST(root->right, V);

root->right = res[0];

res[0] = root; // the smaller part of right child and root together makes the <= tree

} else {

res = splitBST(root->left, V);

root->left = res[1];

res[1] = root; // the larger part of left child and root together makes the > tree

}

return res;

}

};