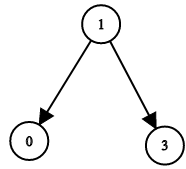
# 题目

给出两棵二叉搜索树，请你从两棵树中各找出一个节点，使得这两个节点的值之和等于目标值 Target。

如果可以找到返回 True，否则返回 False。

示例 1：

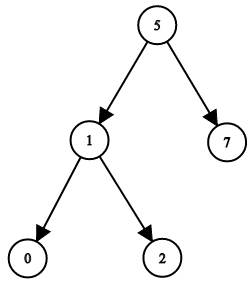


输入：root1 = [2,1,4], root2 = [1,0,3], target = 5

输出：true

解释：2 加 3 和为 5 。

示例 2：



输入：root1 = [0,-10,10], root2 = [5,1,7,0,2], target = 18

输出：false

提示：

每棵树上最多有 5000 个节点。

-10^9 <= target, node.val <= 10^9

# 分析

## 方法一：二叉树遍历

解题思路

步骤一：中序遍历二叉树

根据二叉搜索树性质可知，对于二叉搜索树中的每一个结点，左子树中的值 < 该结点值 < 右子树中的值，所以二叉搜索树的中序遍历的结果就是一个升序序列；

在本题中，使用递归算法遍历两个二叉搜索树，得到两个升序序列；

步骤二：双指针查找

建立两个指针，指针1从第一个序列的最小元素开始遍历，指针2从第二个序列的最大元素开始遍历；对两个指针指向元素进行求和，如果两元素之和与目标值相等，返回true；如果两元素之和小于目标值，指针1后移；如果两元素之和大于目标值，指针2前移；直到遍历完其中一个序列，如果最后没有找到，返回false。

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 {

private:

void inOrder(TreeNode\* root, vector<int> &ans) {

if (root == NULL) { return ; }

inOrder(root->left, ans);

ans.push\_back(root->val);

inOrder(root->right, ans);

}

public:

bool twoSumBSTs(TreeNode\* root1, TreeNode\* root2, int target) {

vector<int> tree1;

vector<int> tree2;

inOrder(root1, tree1);

inOrder(root2, tree2);

int l1 = tree1.size(), l2 = tree2.size();

int i = 0, j = l2 - 1;

while (i < l1 && j >= 0) {

int add = tree1[i] + tree2[j];

if (add > target) {

j--;

}

else if (add < target) {

i++;

}

else {

return true;

}

}

return false;

}

};

另一种方法：

用两个栈分别对两棵树进行中序遍历，左边的树从小到大遍历，右边的树从大到小遍历。

/\*\*

\* 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:

bool twoSumBSTs(TreeNode\* root1, TreeNode\* root2, int target) {

if(root1==NULL || root2==NULL)

return false;

stack<TreeNode\*> left,right;

getLeft(root1,left);

getRight(root2,right);

TreeNode \*l=left.top();//左指针

left.pop();

TreeNode \*r=right.top();//右指针

right.pop();

while(true){

int t=l->val+r->val;

if(t==target)//找到

return true;

if(t>target){

getRight(r->left,right);

if(right.empty())

break;

r=right.top();//右指针左移

right.pop();

}

else if(t<target){

getLeft(l->right,left);

if(left.empty())

break;

l=left.top();//左指针右移

left.pop();

}

}

return false;

}

void getLeft(TreeNode\* root, stack<TreeNode\*> &left){//将左边压入栈

while(root){

left.push(root);

root=root->left;

}

}

void getRight(TreeNode\* root, stack<TreeNode\*> &right){//将右边压入栈

while(root){

right.push(root);

root=root->right;

}

}

};