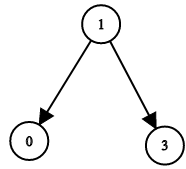
# 题目

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：

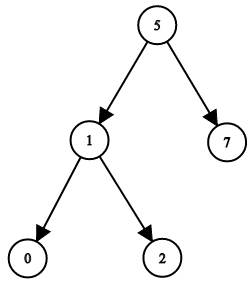


输入：root1 = [2,1,4], root2 = [1,0,3], target = 5

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

要解决这个问题，我们需要判断从两棵二叉搜索树中各取一个节点，是否存在两个节点的值之和等于目标值。我们可以利用二叉搜索树的特性（中序遍历为升序序列）来高效地解决这个问题。

解题思路

1、提取节点值：首先，分别对两棵二叉搜索树进行中序遍历，将遍历结果存储在两个数组中。由于二叉搜索树的中序遍历结果是升序的，因此这两个数组都是有序的。

2、双指针查找：利用双指针技术在两个有序数组中查找是否存在一对元素之和等于目标值。具体来说：

- 初始化两个指针，分别指向第一个数组的起始位置和第二个数组的结束位置。

- 计算两个指针所指元素的和，若等于目标值，则返回`true`。

- 若和小于目标值，将第一个指针右移（增大和）；若和大于目标值，将第二个指针左移（减小和）。

- 重复上述步骤，直到找到符合条件的元素对或指针越界。

代码：

/\*\*

\* Definition for a binary tree node.

\* struct TreeNode {

\* int val;

\* TreeNode \*left;

\* TreeNode \*right;

\* TreeNode() : val(0), left(nullptr), right(nullptr) {}

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

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

\* };

\*/

class Solution {

public:

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

// 存储两棵树的中序遍历结果（升序）

vector<int> vals1, vals2;

inorder(root1, vals1);

inorder(root2, vals2);

// 双指针查找目标和

int i = 0, j = vals2.size() - 1;

while (i < vals1.size() && j >= 0) {

int sum = vals1[i] + vals2[j];

if (sum == target) {

return true;

} else if (sum < target) {

// 和太小，增大vals1[i]

i++;

} else {

// 和太大，减小vals2[j]

j--;

}

}

return false;

}

private:

// 中序遍历二叉搜索树，将节点值存入数组（结果为升序）

void inorder(TreeNode\* root, vector<int>& vals) {

if (root == nullptr) return;

inorder(root->left, vals);

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

inorder(root->right, vals);

}

};

解释

1、中序遍历：通过中序遍历两棵二叉搜索树，我们得到两个升序数组vals1和vals2。这是因为二叉搜索树的特性保证了中序遍历结果的有序性。

2、双指针技术：利用两个指针分别从vals1的起始和vals2的末尾开始移动。通过比较当前指针所指元素的和与目标值的大小，调整指针位置，高效地查找是否存在符合条件的元素对。这种方法的时间复杂度为O(n + m)（其中n和m分别是两棵树的节点数），空间复杂度为O(n + m)（存储两个数组）。

这种方法充分利用了二叉搜索树的有序性和双指针的高效查找能力，能够在较短时间内判断是否存在满足条件的节点对。

## 方法二：中序遍历

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;

}

}

};