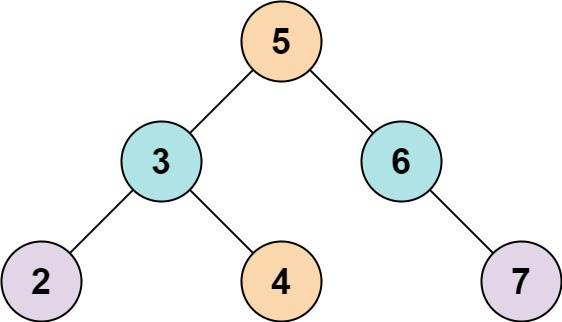
<https://leetcode.com/problems/two-sum-iv-input-is-a-bst/description/>

Given the root of a binary search tree and an integer k, return true **if there exist two elements in the BST such that their sum is equal to** k, **or** false **otherwise**.

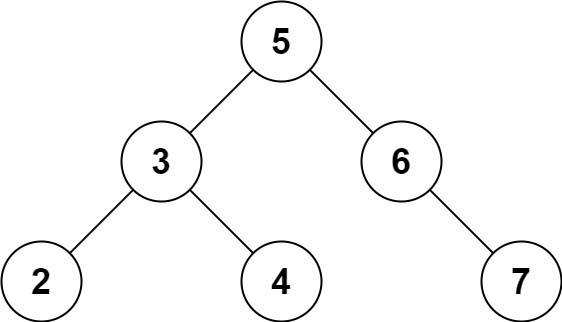
**Example 1:**



**Input:** root = [5,3,6,2,4,null,7], k = 9

**Output:** true

**Example 2:**



**Input:** root = [5,3,6,2,4,null,7], k = 28

**Output:** false

**Constraints:**

The number of nodes in the tree is in the range [1, 10^4].

-10^4 <= Node.val <= 10^4

root is guaranteed to be a valid binary search tree.

-10^5 <= k <= 10^5

**Attempt 1: 2024-01-011**

**Solution 1: Tree Traversal + Hash Table (10min)**

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public boolean findTarget(TreeNode root, int k) {

        Set<Integer> set = new HashSet<>();

        return helper(root, k, set);

    }

    private boolean helper(TreeNode root, int k, Set<Integer> set) {

        if(root == null) {

            return false;

        }

        if(set.contains(k - root.val)) {

            return true;

        }

        set.add(root.val);

        return helper(root.left, k, set) || helper(root.right, k, set);

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Solution 2: Inorder Traversal + Two Pointers (10min)**

**Since its BST, when do Inorder Traversal, we will get sorted result automatically**

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public boolean findTarget(TreeNode root, int k) {

        List<Integer> list = new ArrayList<>();

        helper(root, list);

        int i = 0;

        int j = list.size() - 1;

        while(i < j) {

            if(list.get(i) + list.get(j) == k) {

                return true;

            } else if(list.get(i) + list.get(j) > k) {

                j--;

            } else {

                i++;

            }

        }

        return false;

    }

    private void helper(TreeNode root, List<Integer> list) {

        if(root == null) {

            return;

        }

        helper(root.left, list);

        list.add(root.val);

        helper(root.right, list);

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Solution 3: Level order Traversal + Hash Table (10min)**

/\*\*

 \* Definition for a binary tree node.

 \* public class TreeNode {

 \*     int val;

 \*     TreeNode left;

 \*     TreeNode right;

 \*     TreeNode() {}

 \*     TreeNode(int val) { this.val = val; }

 \*     TreeNode(int val, TreeNode left, TreeNode right) {

 \*         this.val = val;

 \*         this.left = left;

 \*         this.right = right;

 \*     }

 \* }

 \*/

class Solution {

    public boolean findTarget(TreeNode root, int k) {

        Set<Integer> set = new HashSet<>();

        Queue<TreeNode> q = new LinkedList<>();

        q.offer(root);

        while(!q.isEmpty()) {

            int size = q.size();

            for(int i = 0; i < size; i++) {

                TreeNode node = q.poll();

                if(set.contains(k - node.val)) {

                    return true;

                }

                set.add(node.val);

                if(node.left != null) {

                    q.offer(node.left);

                }

                if(node.right != null) {

                    q.offer(node.right);

                }

            }

        }

        return false;

    }

}

Time Complexity: O(N)

Space Complexity: O(N)

**Refer to**

<https://grandyang.com/leetcode/653/>

这道题又是一道2sum的变种题，博主一直强调，平生不识TwoSum，刷尽LeetCode也枉然！只要是两数之和的题，一定要记得先尝试用HashSet来做，这道题只不过是把数组变成了一棵二叉树而已，换汤不换药，我们遍历二叉树就行，然后用一个HashSet，在递归函数函数中，如果node为空，返回false。如果k减去当前结点值在HashSet中存在，直接返回true；否则就将当前结点值加入HashSet，然后对左右子结点分别调用递归函数并且或起来返回即可，参见代码如下：

class Solution {

public:

bool findTarget(TreeNode\* root, int k) {

unordered\_set<int> st;

return helper(root, k, st);

}

bool helper(TreeNode\* node, int k, unordered\_set<int>& st) {

if (!node) return false;

if (st.count(k - node->val)) return true;

st.insert(node->val);

return helper(node->left, k, st) || helper(node->right, k, st);

}

};

我们也可以用层序遍历来做，这样就是迭代的写法了，但是利用HashSet的精髓还是没变的，参见代码如下：

class Solution {

public:

bool findTarget(TreeNode\* root, int k) {

if (!root) return false;

unordered\_set<int> st;

queue<TreeNode\*> q{{root}};

while (!q.empty()) {

auto t = q.front(); q.pop();

if (st.count(k - t->val)) return true;

st.insert(t->val);

if (t->left) q.push(t->left);

if (t->right) q.push(t->right);

}

return false;

}

};

由于输入是一棵二叉搜索树，那么我们可以先用中序遍历得到一个有序数组，然后在有序数组中找两数之和就很简单了，直接用双指针进行遍历即可，参见代码如下：

class Solution {

public:

bool findTarget(TreeNode\* root, int k) {

vector<int> nums;

inorder(root, nums);

for (int i = 0, j = (int)nums.size() - 1; i < j;) {

if (nums[i] + nums[j] == k) return true;

(nums[i] + nums[j] < k) ? ++i : --j;

}

return false;

}

void inorder(TreeNode\* node, vector<int>& nums) {

if (!node) return;

inorder(node->left, nums);

nums.push\_back(node->val);

inorder(node->right, nums);

}

};