**Find a pair with given target in BST**

**Medium**

Given a Binary Search Tree and a target sum. Check whether there's a pair of Nodes in the BST with value summing up to the target sum.

**Example 1:**

**Input:**

2

/ \

  1 3

sum = 5

**Output:** 1

**Explanation:**

Nodes with value 2 and 3 sum up to 5.

**Example 2:**

**Input:**

6

  /

  5

  /

  3

  / \

  1 4

sum = 2

**Output:** 0

**Explanation:**

There's no pair that sums up to 2.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(Height of the BST).

**Constraints:**  
1 ≤ Number of Nodes ≤ 105  
1 ≤ Sum ≤ 106

**Company Tags**

[**VMWare**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=VMWare) [**Flipkart**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Flipkart) [**Amazon**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Amazon) [**Microsoft**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Microsoft) [**OYO Rooms**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=OYO%20Rooms) [**Snapdeal**](https://practice.geeksforgeeks.org/explore/?company%5b%5d=Snapdeal)

//{ Driver Code Starts

//Initial Template for Java

import java.io.\*;

import java.util.\*;

import java.math.\*;

class Node

{

int data;

Node left, right;

public Node(int d)

{

data = d;

left = right = null;

}

}

class CodingMaxima

{

static Node buildTree(String str)

{

// Corner Case

if(str.length() == 0 || str.equals('N'))

return null;

String[] s = str.split(" ");

Node root = new Node(Integer.parseInt(s[0]));

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

q.add(root);

// Starting from the second element

int i = 1;

while(!q.isEmpty() && i < s.length)

{

// Get and remove the front of the queue

Node currNode = q.remove();

// Get the current node's value from the string

String currVal = s[i];

// If the left child is not null

if(!currVal.equals("N"))

{

// Create the left child for the current node

currNode.left = new Node(Integer.parseInt(currVal));

// Push it to the queue

q.add(currNode.left);

}

// For the right child

i++;

if(i >= s.length)

break;

currVal = s[i];

// If the right child is not null

if(!currVal.equals("N"))

{

// Create the right child for the current node

currNode.right = new Node(Integer.parseInt(currVal));

// Push it to the queue

q.add(currNode.right);

}

i++;

}

return root;

}

public static void main(String args[]) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int t = Integer.parseInt(br.readLine().trim());

while(t>0)

{

String s = br.readLine();

Node root = buildTree(s);

int target = Integer.parseInt(br.readLine().trim());

Solution T = new Solution();

System.out.println(T.isPairPresent(root,target));

t--;

}

}

}

// } Driver Code Ends

//User function Template for Java

class Solution

{

// root : the root Node of the given BST

// target : the target sum

public int isPairPresent(Node root, int target)

{

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

if(helper(root, target, set))

return 1;

return 0;

}

private boolean helper(Node root, int target, HashSet<Integer> set) {

if(root==null)

return false;

if(helper(root.left, target, set))

return true;

if(set.contains(target-root.data))

return true;

else{

set.add(root.data);

}

return helper(root.right, target, set);

}

}