**Check for Balanced Tree: -**

**Easy** Accuracy: **43.15%** Submissions: **260K+** Points: **2**

Given a binary tree, find if it is height balanced or not.   
A tree is height balanced if difference between heights of left and right subtrees is **not more than one** for all nodes of tree.

**A height balanced tree**  
        1  
     /     \  
   10      39  
  /  
5

**An unbalanced tree**  
        1  
     /      
   10     
  /  
5

**Example 1:**

**Input:**

      1

   /

   2

   \

    3

**Output:** 0

**Explanation:** The max difference in height

of left subtree and right subtree is 2,

which is greater than 1. Hence unbalanced

**Example 2:**

**Input:**

       10

    /   \

   20   30

  /   \

40   60

**Output:** 1

**Explanation:** The max difference in height

of left subtree and right subtree is 1.

Hence balanced.

**Your Task:**  
You don't need to take input. Just complete the function**isBalanced()**that takes root **node**as parameter and returns **true,**if the tree is balanced else returns **false**.

**Constraints:**  
1 <= Number of nodes <= 105  
1 <= Data of a node <= 109

**Expected time complexity:**O(N)  
**Expected auxiliary space:**O(h) , where h = height of tree

**Code:-**

//{ Driver Code Starts

//Initial Template for C++

#include <bits/stdc++.h>

using namespace std;

// Tree Node

struct Node {

int data;

Node\* left;

Node\* right;

};

// Utility function to create a new Tree Node

Node\* newNode(int val) {

Node\* temp = new Node;

temp->data = val;

temp->left = NULL;

temp->right = NULL;

return temp;

}

// Function to Build Tree

Node\* buildTree(string str) {

// Corner Case

if (str.length() == 0 || str[0] == 'N') return NULL;

// Creating vector of strings from input

// string after spliting by space

vector<string> ip;

istringstream iss(str);

for (string str; iss >> str;) ip.push\_back(str);

// Create the root of the tree

Node\* root = newNode(stoi(ip[0]));

// Push the root to the queue

queue<Node\*> queue;

queue.push(root);

// Starting from the second element

int i = 1;

while (!queue.empty() && i < ip.size()) {

// Get and remove the front of the queue

Node\* currNode = queue.front();

queue.pop();

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

string currVal = ip[i];

// If the left child is not null

if (currVal != "N") {

// Create the left child for the current node

currNode->left = newNode(stoi(currVal));

// Push it to the queue

queue.push(currNode->left);

}

// For the right child

i++;

if (i >= ip.size()) break;

currVal = ip[i];

// If the right child is not null

if (currVal != "N") {

// Create the right child for the current node

currNode->right = newNode(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

// } Driver Code Ends

/\* A binary tree node structure

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

Node(int x){

data = x;

left = right = NULL;

}

};

\*/

class Solution{

public:

int helper(Node \*root){

if(!root)

return 1e7;

int leftheight = helper(root->left);

if(leftheight == false)

return false;

int rightheight = helper(root->right);

if(rightheight == false)

return false;

int balancefactor = leftheight - rightheight;

if(-1 <= balancefactor and balancefactor <= 1)

return 1 + max(leftheight, rightheight);

return false;

}

//Function to check whether a binary tree is balanced or not.

bool isBalanced(Node \*root){

return helper(root);

}

};

//{ Driver Code Starts.

/\* Driver program to test size function\*/

int main() {

int t;

scanf("%d ", &t);

while (t--) {

string s, ch;

getline(cin, s);

Node\* root = buildTree(s);

Solution ob;

cout << ob.isBalanced(root) << endl;

}

return 0;

}

// } Driver Code Ends

**T.C: - O(N)**

**S.C: - O(h), h = height of tree**