**Normal BST to Balanced BST: -**

**Easy** Accuracy: **66.08%** Submissions: **39K+** Points: **2**

Given a Binary Search Tree, modify the given BST such that it is balanced and has minimum possible height. Return the balanced BST.

**Example1:**

**Input:**

30

/

20

/

10  
**Output:**

20

/ \

10 30

**Example2:**

**Input:**

4

/

3

/

2

/

1

**Output:**

3 3 2

/ \ / \ / \

1 4 OR 2 4 OR 1 3

\ / \   
 2 1 4

**Your Task:**  
The task is to complete the function **buildBalancedTree()** which takes root as the input argument and returns the root of tree after converting the given BST into a balanced BST with minimum possible height. The driver code will print the height of the updated tree in output itself.  
   
**Expected Time Complexity:**O(N)  
**Expected Auxiliary Space:**O(N)  
Here N denotes total number of nodes in given BST.

**Constraints:**  
1 <= N <= 105  
1 <= Node data <= 109

**Code: -**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node \*left;

struct Node \*right;

Node(int x){

data = x;

left = NULL;

right = NULL;

}

};

// } Driver Code Ends

/\*Structure of the Node of the BST is as

struct Node

{

int data;

Node\* left, \*right;

};

\*/

class Solution{

public:

void fill(Node \*root, vector<Node \*> &arr){

if(!root) return;

fill(root->left, arr);

arr.push\_back(root);

fill(root->right, arr);

}

Node \*build(int start, int end, vector<Node \*> &arr){

if(start > end) return NULL;

int mid = start + (end - start) / 2;

arr[mid]->left = build(start, mid-1, arr);

arr[mid]->right = build(mid+1, end, arr);

return arr[mid];

}

// Your are required to complete this function

// function should return root of the modified BST

Node\* buildBalancedTree(Node\* root){

vector<Node \*> inorder;

fill(root, inorder);

return build(0, inorder.size()-1, inorder);

}

};

//{ Driver Code Starts.

Node\* insert(struct Node\* node, int key){

if (node == NULL) return new Node(key);

if (key < node->data)

node->left = insert(node->left, key);

else if (key > node->data)

node->right = insert(node->right, key);

return node;

}

void preOrder(Node\* node)

{

if (node == NULL)return;

printf("%d ", node->data);

preOrder(node->left);

preOrder(node->right);

}

int height(struct Node\* node)

{

if (node==NULL)

return 0;

int lDepth = height(node->left);

int rDepth = height(node->right);

if (lDepth > rDepth)

return(lDepth+1);

else

return(rDepth+1);

}

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 = new Node(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 = new Node(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 = new Node(stoi(currVal));

// Push it to the queue

queue.push(currNode->right);

}

i++;

}

return root;

}

Node\* buildBalancedTree(Node\* root);

int main()

{

int t;

cin>>t;

getchar();

while(t--)

{

struct Node \*root = NULL;

int n, temp;

string tree;

getline(cin,tree);

root = buildTree(tree);

// cout<<height(root)<<endl;

Solution obj;

root = obj.buildBalancedTree(root);

cout<<height(root)<<endl;

}

return 0;

}

// } Driver Code Ends

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

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