**Duplicate subtree in Binary Tree: -**

**Medium** Accuracy: **34.23%** Submissions: **79K+** Points: **4**

Given a binary tree, find out whether it contains a duplicate sub-tree of size two or more, or not.

**Note:** Two same leaf nodes are not considered as subtree as size of a leaf node is one.   
  
**Example 1 :**

**Input :**

1

/ \

2 3

/ \ \

4 5 2

/ \

4 5

**Output :** 1

**Explanation :**

2

/ \

4 5

is the duplicate sub-tree.

**Example 2 :**

**Input :**

1

/ \

2 3

**Output:** 0

**Explanation:** There is no duplicate sub-tree

in the given binary tree.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **dupSub()** which takes root of the tree as the only argument and returns 1 if the binary tree contains a duplicate sub-tree of size two or more, else 0.

**Expected Time Complexity:** O(N)  
**Expected Space Complexity:** O(N)

**Constraints:**  
0 ≤ Data of nodes ≤ 9  
1 ≤ Number of nodes ≤ 105

**Code: -**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

struct Node

{

char data;

struct Node \*left;

struct Node \*right;

Node(char x) {

data = x;

left = NULL;

right = NULL;

}

};

struct 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;

}

// } Driver Code Ends

/\*The structure of the Binary Tree Node is

struct Node

{

char data;

struct Node\* left;

struct Node\* right;

};\*/

class Solution {

public:

pair<string,int> helper(Node \*root, unordered\_map<string,bool> &mp){

// base case

if(!root)

return {"N",0};

// recursive case

pair<string,int> left = helper(root->left, mp);

string leftsubtree = left.first;

if(leftsubtree.size() == 0)

return {"", 1};

int leftcountnode = left.second;

pair<string,int> right = helper(root->right, mp);

string rightsubtree = right.first;

if(rightsubtree.size() == 0)

return {"", 1};

int rightcountnode = right.second;

string cursubtree = root->data + "-" + leftsubtree + "-" + rightsubtree;

int curcountnode = leftcountnode + 1 + rightcountnode;

// cursubtree already found && its size >= 2

if(curcountnode >= 2 and mp.find(cursubtree) != mp.end()){

cout << cursubtree;

return {"", 1};

}

mp[cursubtree] = true;

return {cursubtree, curcountnode};

}

/\*This function returns true if the tree contains

a duplicate subtree of size 2 or more else returns false\*/

int dupSub(Node \*root) {

unordered\_map<string,bool> mp;

return helper(root, mp).first.size() == 0;

}

};

//{ Driver Code Starts.

int main()

{

int t;

cin >> t;

//cout << t << "\n";

while (t--)

{

string treeString;

getline(cin >> ws, treeString);

struct Node\* root = buildTree(treeString);

Solution ob;

cout << ob.dupSub(root) << "\n";

}

return 0;

}

// } Driver Code Ends

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

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