**Lowest Common Ancestor in a BST**

Easy Accuracy: 65.2% Submissions: 129K+ Points: 2

Given a Binary Search Tree (with all values unique) and two node values n1 and n2 (n1!=n2). Find the Lowest Common Ancestors of the two nodes in the BST.

**Example 1:**

**Input:**

              5

           /  \

          4      6

    /       \

   3       7

                 \

                    8

n1 = 7, n2 = 8

**Output:** 7

**Example 2:**

**Input:**

2

  / \

  1 3

n1 = 1, n2 = 3

**Output:** 2

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **LCA()** which takes the root Node of the BST and two integer values n1 and n2 as inputs and returns the Lowest Common Ancestor of the Nodes with values n1 and n2 in the given BST.

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

**Constraints:**  
1 <= N <= 104

**CODE :-**

//{ Driver Code Starts

#include <bits/stdc++.h>

using namespace std;

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

// Function to Build Tree

// } Driver Code Ends

//Function to find the lowest common ancestor in a BST.

class Solution{

public:

Node\* LCA(Node \*root, int n1, int n2){

//prev = the root containing the same ancestor

//p1 = the root searching for n1

//p2 = the root searching for n2

Node \*prev=root, \*p1=root, \*p2=root;

//loop execution till both p1!=NULL and p2!=NULL

while(p1 && p2){

//if both p1, p2 value same, storing it as common ancestor

if(p1->data == p2->data) prev = p1;

if(p1->data==n1 && p2->data==n2) break;

//if p1 value = n1, don't change p1 node for next iteration

//else p1 node will change according to condition

if(n1 > p1->data) p1=p1->right;

else if(n1 < p1->data) p1=p1->left;

//if p2 value = n2, don't change p2 node for next iteration

//else p2 node will change according to condition

if(n2 > p2->data) p2=p2->right;

else if(n2 < p2->data) p2=p2->left;

}

//if anyone node pointer become NULL without finding n1 and n2

return prev;

}

};

//{ Driver Code Starts.

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;

}

int main()

{

int t;

scanf("%d ",&t);

while(t--)

{

string s;

int l , h;

getline(cin,s);

scanf("%d ",&l);

scanf("%d ",&h);

Node\* root = buildTree(s);

Solution sol;

int ans = sol.LCA(root, l, h)->data;

cout<<ans<<endl;

}

return 1;

}

// } Driver Code Ends

**T.C :- O(Height of BST)**

**S.C :- O(1)**