**Inorder Successor in BST :-**

Easy Accuracy: 34.97% Submissions: 86K+ Points: 2

Given a BST, and a reference to a Node x in the BST. Find the Inorder Successor of the given node in the BST.

**Example 1:**

**Input:**

**2**

  / \

1 3

K(data of x) = 2

**Output:** 3

**Explanation:**

Inorder traversal : 1 2 3

Hence, inorder successor of 2 is 3.

**Example 2:**

**Input:**

20

  / \

  822

  / \

  4 12

  /\

  10 14

K(data of x) = 8

**Output:** 10

**Explanation:**

Inorder traversal: 4 8 10 12 14 20 22

Hence, successor of 8 is 10.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **inOrderSuccessor()**. This function takes the root node and the reference node as argument and returns the node that is inOrder successor of the reference node. If there is no successor, return null value.

**Expected Time Complexity:** O(Height of the BST).  
**Expected Auxiliary Space:** O(1).

**Constraints:**  
1 <= N <= 105, where N is number of nodes

**Code :-**

//{ Driver Code Starts

#include<bits/stdc++.h>

using namespace std;

// Tree Node

struct Node {

int data;

Node \*left;

Node \*right;

Node(int val) {

data = val;

left = right = NULL;

}

};

// 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 = 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\* search(Node\* root, int key)

{

// Base Cases: root is null or key is present at root

if (root == NULL || root->data == key)

return root;

// Key is greater than root's key

if (root->data < key)

return search(root->right, key);

// Key is smaller than root's key

return search(root->left, key);

}

// } Driver Code Ends

class Solution{

public:

// returns the inorder successor of the Node x in BST (rooted at 'root')

Node \* inOrderSuccessor(Node \*root, Node \*x){

if(!x) return NULL;

Node \*great=NULL, \*ptr=root;

bool find=false;

while(true){

if(!ptr) return great;

if(ptr->data > x->data) great = ptr;

if(find==true){

ptr=ptr->left;

continue;

}

if(ptr->data == x->data){

find = true;

ptr = ptr->right;

}

else if(x->data < ptr->data)

ptr = ptr->left;

else if(x->data > ptr->data)

ptr = ptr->right;

}

}

};

//{ Driver Code Starts.

int main()

{

int t;

cin>>t;

getchar();

while(t--)

{

string s;

getline(cin,s);

Node\* head = buildTree(s);

int k;

cin>>k;

getchar();

Node\* kNode = search( head, k );

Solution obj;

Node\* suc = obj.inOrderSuccessor(head, kNode);

if( suc == NULL ) cout<< "-1";

else cout<< suc->data;

cout << endl;

}

return 1;

}

// } Driver Code Ends

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

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