**Predecessor and Successor :-**

Medium Accuracy: 47.36% Submissions: 68K+ Points: 4

There is BST given with the root node with the key part as an integer only. You need to find the in-order **successor** and **predecessor** of a given key. If either predecessor or successor is not found, then set it to **NULL**.

**Note**:- In an inorder traversal the number just **smaller** than the target is the predecessor and the number just **greater** than the target is the successor.

**Example 1:**

**Input:**

10

  / \

  2 11

  / \

  1 5

  / \

  3 6

  \

  4

key = 8

**Output:**6 10

**Explanation:**In the given BST the inorder predecessor of 8 is 6 and inorder successor of 8 is 10.

**Example 2:**

**Input:**

8

  / \

  1 9

  \ \

  4 10

  /

  3

key = 11

**Output:**10 -1

**Explanation:**In given BST, the inorder predecessor of 11 is 10 whereas it does not have any inorder successor.

**Your Task:**You don't need to print anything. You need to update **pre**with the predecessor of the key or **NULL**if the predecessor doesn't exist and **succ**to the successor of the key or **NULL**if the successor doesn't exist. pre and succ are passed as an argument to the function **findPreSuc()**.

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

**Constraints:**  
1 <= Number of nodes <= 104  
1 <= key of node <= 107  
1 <= key <= 107

**Code :-**

//{ Driver Code Starts

// C++ program to find predecessor and successor in a BST

#include "bits/stdc++.h"

using namespace std;

// BST Node

struct Node

{

int key;

struct Node \*left;

struct Node \*right;

Node(int x){

key = x;

left = NULL;

right = NULL;

}

};

// } Driver Code Ends

// This function finds predecessor and successor of key in BST.

// It sets pre and suc as predecessor and successor respectively

class Solution

{

public:

void findPreSuc(Node\* root, Node\*& pre, Node\*& suc, int key)

{

pre=NULL, suc=NULL;

Node \*ptr=root;

while(ptr){

if(ptr->key < key){

pre = ptr;

ptr = ptr->right;

}

else if(ptr->key > key){

suc = ptr;

ptr = ptr->left;

}

else if(ptr->key == key){

Node \* p = ptr->left;

Node \* q = ptr->right;

while(p){

pre = p;

p = p->right;

}

while(q!=NULL && q->key > key){

suc = q;

q = q->left;

}

return;

}

}

return;

}

};

//{ 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;

}

// Driver program to test above functions

int main() {

int t;

string tc;

getline(cin, tc);

t=stoi(tc);

while(t--)

{

string s;

getline(cin, s);

Node\* root = buildTree(s);

getline(cin, s);

int k = stoi(s);

Node \*pre=NULL,\*succ=NULL;

Solution ob;

ob.findPreSuc(root,pre,succ,k);

(pre!=NULL)?cout<<pre->key:cout<<-1;

cout<<" ";

(succ!=NULL)?cout<<succ->key:cout<<-1;

cout<<endl;

// inOrderTraversal(root);

}

return 0;

}

// } Driver Code Ends

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

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