**############################################################**

**1. Count Leaves in Binary Tree**

Given a Binary Tree of size **N** , You have to count leaves in it. For example, there are two leaves in following tree

        1  
     /      \  
   10      39  
  /  
5

**Example 1:**

**Input:**

Given Tree is

4

/ \

8 10

/ / \

7 5 1

/

3

**Output:**

3

**Explanation:**

Three leaves are 3 , 5 and 1.

**Your Task:**  
You don't have to take input. Complete the function **countLeaves()**that takes **root**node of given treeas parameter and **returns**the count of leaves in tree . The **printing**is done by the **driver**code.

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

**Note:**The **Input/Ouput** format and **Example** given below is used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console, and should not print anything on stdout/console. The task is to complete the function specified, and not to write the full code.

int countLeaves(Node\* root)

{

if(root == NULL)

return 0;

if(root->left == NULL && root->right == NULL)

return 1;

return (countLeaves(root->left) + countLeaves(root->right));

}

**############################################################**

**2. Right View of Binary Tree**

Given a Binary Tree, find **Right view** of it. Right view of a Binary Tree is set of nodes visible when tree is viewed from **right**side.

Right view of following tree is 1 3 7 8.

          1  
       /     \  
     2        3  
   /   \      /    \  
  4     5   6    7  
    \  
     8

**Example 1:**

**Input:**

       1

   /    \

  3      2

**Output:** 1 2

**Example 2:**

**Input:**

     10

   /   \

 20     30

/   \

40  60

**Output:** 10 30 60

**Your Task:**  
Just complete the **function rightView()**that takes **node**as **parameter**and returns the right view as a list.

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

**Constraints:**  
1 <= Number of nodes <= 105  
1 <= Data of a node <= 105.

vector<int> rightView(Node \*root)

{

if(root == NULL)

return {};

vector<int> ans;

queue<Node\*> q;

q.push(root);

while(!q.empty())

{

int n = q.size();

for(int i=0; i<n; i++)

{

Node \*cur = q.front();

q.pop();

if(i == n-1)

ans.push\_back(cur->data);

if(cur->left != NULL)

q.push(cur->left);

if(cur->right != NULL)

q.push(cur->right);

}

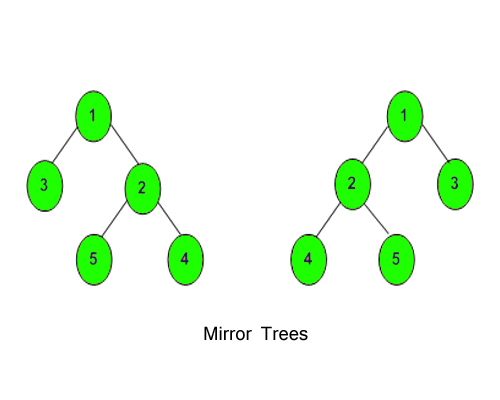
}

return ans;

}

**############################################################**

**3. Two Mirror Trees**

Given a Two Binary Trees, write a function that returns true if one is mirror of other, else returns false.  


**Example 1:**

**Input:**

**T1:** 1 **T2:** 1

  / \ / \

  2 3 3 2

**Output:** 1

**Example 2:**

**Input:**

**T1:** 10 **T2:** 10

  / \ / \

  20 30 20 30

  / \ / \

  40 60 40 60

**Output:** 0

**Your Task:**  
You don't need to take input. Just complete the function**areMirror()**that takes root **node of two tree**as parameter and returns **true,**if one is the mirror of other else returns **false**. (The driver's code print 1 if the returned value is **true,**otherwise 0)  
**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(Height of the Tree).  
  
**Constraints:**  
1 <= Number of nodes<= 10000  
-1000 <= Data of a node<= 1000

int areMirror(Node\* a, Node\* b)

{

if(a == NULL && b == NULL)

return 1;

queue<Node\*> q1, q2;

q1.push(a);

q2.push(b);

while(!q1.empty() && !q2.empty())

{

int n1 = q1.size();

int n2 = q2.size();

if(n1 != n2)

return 0;

for(int i=0, j=n1-1; i<n1 && j>=0; i++, j--)

{

Node \*t1 = q1.front();

Node \*t2 = q2.front();

q1.pop();

q2.pop();

if(t1->data == t2->data)

{

if(t1->left != NULL && t2->right != NULL)

{

q1.push(t1->left);

q2.push(t2->right);

}

if(t1->right != NULL && t2->left != NULL)

{

q1.push(t1->right);

q2.push(t2->left);

}

}

else

return 0;

}

}

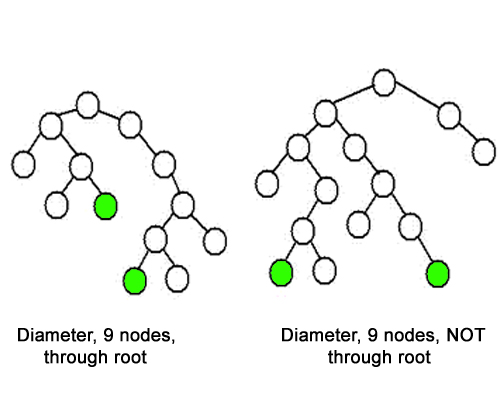
return 1;

}

**############################################################**

**4. Diameter of Binary Tree**

Given a Binary Tree,**find diameter of it**.  
The diameter of a tree is the number of nodes on the longest path between two end nodes in the tree. The diagram below shows two trees each with diameter nine, the leaves that form the ends of a longest path are shaded (note that there is more than one path in each tree of length nine, but no path longer than nine nodes).

[](http://geeksforgeeks.org/wp-content/uploads/tree_diameter.GIF)

**Example 1:**

**Input:**

     1

    /  \

   2    3

**Output:** 3

**Example 2:**

**Input:**

         10

       /   \

     20    30

   /   \

   40   60

**Output:** 4

**Your Task:**  
You need to **complete**the **function diameter()**that takes **root**as **parameter**and **returns**the **diameter**.  
  
**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(Height of the Tree).

**Constraints:**  
1 <= Number of nodes <= 10000  
1 <= Data of a node <= 1000

int diam(Node \*root, int &h)

{

if(root == NULL)

{

return 0;

}

int l = diam(root->left, h);

int r = diam(root->right, h);

int temp = max(l, r) + 1;

int ans = max(temp, l + r + 1);

h = max(h, ans);

return temp;

}

int diameter(Node\* root) {

if(root == NULL)

return 0;

int h = INT\_MIN;

diam(root, h);

return h;

}

**############################################################**

**5. Sum Tree**

Given a Binary Tree. Return **1** if, for every node **X** in the tree other than the leaves, its value is equal to the sum of its left subtree's value and its right subtree's value. Else return **0**.

An empty tree is also a Sum Tree as the sum of an empty tree can be considered to be 0. A leaf node is also considered a Sum Tree.

**Example 1:**

**Input:**

3

/ \

1 2

**Output:** 1

**Explanation:** The sum of left subtree and right subtree

is 1 + 2 = 3, which is the value of the root node.

Therefore,the given binary tree is a **sum tree**.

**Example 2:**

**Input:**

10

/ \

20 30

/ \

10 10

**Output:** 0

**Explanation:** The given tree is not a sum

tree. For the root node, sum of elements

in left subtree is 40 and sum of elements

in right subtree is 30. Root element = 10

which is not equal to 30+40?

**Your Task:**  
You dont need to read input or print anything. Complete the function **isSumTree()**which takes **root**node as input parameter and returns true if the tree is a SumTree else it returns false.

**Expected Time Complexity:**O(N)  
**Expected Auxiliary Space:** O(Height of the Tree)

**Constraints:**  
1 ≤ number of nodes ≤ 104

int check(Node \*root)

{

if(root == NULL)

return 0;

else if(root->left == NULL && root->right == NULL)

return root->data;

else if(root->data == (check(root->left) + check(root->right)))

return 2 \* root->data;

}

bool isSumTree(Node\* root)

{

if(root->data == check(root)/2)

return true;

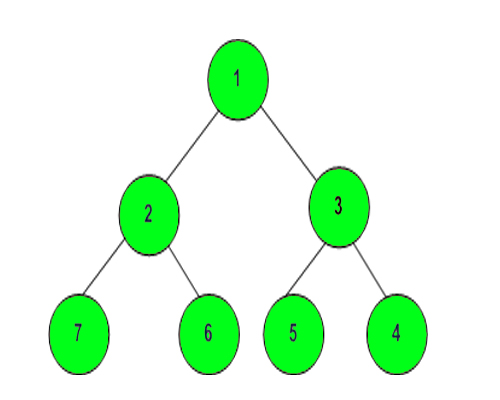
else

return false;

}

**############################################################**

**6. Level order traversal in spiral form**

Complete the function to find spiral order traversal of a tree. For below tree, function should return 1, 2, 3, 4, 5, 6, 7.  
  
  
****

**Example 1:**

**Input:**

      1

   /   \

  3     2

**Output:**1 3 2

**Example 2:**

**Input:**

           10

        /     \

       20     30

     /    \

   40     60

**Output:** 10 20 30 60 40

**Your Task:**  
The task is to complete the function **findSpiral**() which takes **root**node as input parameter and returns the elements in spiral form of level order traversal as a list. The newline is automatically appended by the driver code.  
**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(N).

**Constraints:**  
0 <= Number of nodes <= 105  
0 <= Data of a node <= 105

vector<int> findSpiral(Node \*root)

{

if(root == NULL)

return {};

stack<Node\*> s1, s2;

s1.push(root);

vector<int> ans;

while(!s1.empty() || !s2.empty())

{

while(!s1.empty())

{ Node \*t = s1.top();

s1.pop();

ans.push\_back(t->data);

if(t->right != NULL)

s2.push(t->right);

if(t->left != NULL)

s2.push(t->left);

}

while(!s2.empty())

{

Node \*t = s2.top();

s2.pop();

ans.push\_back(t->data);

if(t->left != NULL)

s1.push(t->left);

if(t->right != NULL)

s1.push(t->right);

}

}

return ans;

}

**############################################################**

**7. Lowest Common Ancestor in a Binary Tree**

Given a Binary Tree with all **unique** values and two nodes value **n1** and **n2**. The task is to find the**lowest common ancestor** of the given two nodes. We may assume that either both n1 and n2 are present in the tree or none of them is present.

**Example 1:**

**Input:**

n1 = 2 , n2 = 3

    1

  /  \

  2   3

**Output:** 1

**Explanation:**

LCA of 2 and 3 is 1.

**Example 2:**

**Input:**

n1 = 3 , n2 = 4

        5

       /

      2

    /  \

   3   4

**Output:** 2

**Explanation:**

LCA of 3 and 4 is 2.

**Your Task:**  
You don't have to read input or print anything. Your task is to complete the function **lca()**that takes nodes, **n1, and n2** as parameters and returns **LCA**node as output.

**Expected Time Complexity:**O(N).  
**Expected Auxiliary Space:**O(Height of Tree).

**Constraints:**  
1 ≤ Number of nodes ≤ 105  
1 ≤ Data of a node ≤ 105

Node\* lca(Node\* root ,int n1 ,int n2 )

{

if(root == NULL)

return NULL;

if(root->data == n1 || root->data == n2)

return root;

Node \*left = lca(root->left, n1, n2);

Node \*right = lca(root->right, n1, n2);

if(left != NULL && right != NULL)

return root;

if(left == NULL)

return right;

if(right == NULL)

return left;

}

**############################################################**

**8. Determine if Two Trees are Identical**

Given two binary trees, the task is to find if both of them are identical or not.

**Example 2:**

**Input:**

1 1

  / \ / \

  2 3 2 3

**Output:** Yes

**Explanation:** There are two trees both

having 3 nodes and 2 edges, both trees

are identical having the root as 1,

left child of 1 is 2 and right child

of 1 is 3.

**Example 2:**

**Input:**

1 1

  / \ / \

 2 3 3 2

**Output:** No

**Explanation:** There are two trees both

having 3 nodes and 2 edges, but both

trees are not identical.

**Your task:**  
Since this is a functional problem you don't have to worry about input, you just have to complete the function **isIdentical()** that takes two roots as parameters and returns true or false. The printing is done by the driver code.

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

**Constraints:**  
1 <= Number of nodes <= 105  
1 <=Data of a node <= 105

bool isIdentical(Node \*n1, Node \*n2) {

if(n1 == NULL && n2 == NULL)

return true;

if(n1 != NULL && n2 == NULL)

return false;

if(n1 == NULL && n2 != NULL)

return false;

if(n1->data != n2->data)

return false;

return isIdentical(n1->left, n2->left) && isIdentical(n1->right, n2->right);

}

**############################################################**

**9. Root to leaf path sum**

Given a binary tree and an integer S, check whether there is root to leaf path with its sum as S.

**Example 1:**

**Input:**

Tree =

1

/ \

2 3

S = 2

**Output:** 0

**Explanation:**

There is no root to leaf path with sum 2.

**Example 2:**

**Input:**

Tree =

1

/ \

2 3

S = 4

**Output:** 1

**Explanation:**

The sum of path from leaf node 3 to root 1 is 4.

**Your Task:**  
You dont need to read input or print anything. Complete the function**hasPathSum()** which takes **root**node and target sum **S** as input parameter and returns true if path exists otherwise it returns false.

**Expected Time Complexity:**O(N)  
**Expected Auxiliary Space:** O(height of tree)

**Constraints:**  
1 ≤ N ≤ 10^4  
1 ≤ S ≤ 10^6

bool hasPathSum(Node \*root, int sum) {

if(root == NULL || sum == 0)

return 0;

if(root->left == NULL && root->right == NULL)

return (sum == root->data);

return (hasPathSum(root->left, sum - root->data) || hasPathSum(root->right, sum - root->data));

}

**############################################################**

**10. Vertical sum**

Given a Binary Tree, find vertical sum of the nodes that are in same vertical line. Print all sums through different vertical lines starting from left-most vertical line to right-most vertical line.

**Example 1:**

**Input:**

1

/ \

2 3

/ \ / \

4 5 6 7

**Output:**

**Explanation:**

The tree has 5 vertical lines

Vertical-Line-1 has only one node

4 => vertical sum is 4

Vertical-Line-2: has only one node

2=> vertical sum is 2

Vertical-Line-3: has three nodes:

1,5,6 => vertical sum is 1+5+6 = 12

Vertical-Line-4: has only one node 3

=> vertical sum is 3

Vertical-Line-5: has only one node 7

=> vertical sum is 7

**Your Task:**  
You don't need to take input. Just complete the function**verticalSum()**that takes **root**node of the treeas parameter and returns an array containing the vertical sum of tree from left to right.

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

**Constraints:**  
1<=Number of nodes<=1000

void sum(Node \*root, map<int, int> &mp, int pos) {

if(root == NULL)

return;

sum(root->left, mp, pos - 1);

mp[pos] += root->data;

sum(root->right, mp, pos + 1);

}

vector <int> verticalSum(Node \*root) {

map<int, int> mp;

sum(root, mp, 0);

vector<int> ans;

for(auto i : mp)

ans.push\_back(i.second);

return ans;

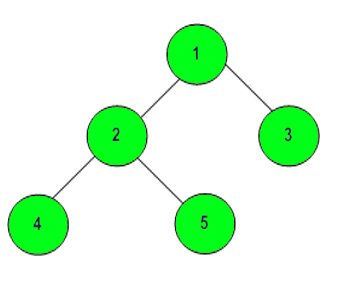
}

**############################################################**

**11. Kth Ancestor in a Tree**

Given a binary tree of size  **N**, a **node** and a positive integer **k**., Your task is to complete the function **kthAncestor()**, the function should return the **kth** ancestor of the given node in the binary tree. If there does not exist any such ancestor then return -1.

**Example 1:**



**Input:**

K = 2

Node = 4

**Output:** 1

**Explanation:**

Since, K is 2 and node is 4, so we

first need to locate the node and

look k times its ancestors.

Here in this Case node 4 has 1 as his

2nd Ancestor aka the Root of the tree.

**Example 2:**

**Input:**

k=1

node=3

1

/ \

2 3

**Output:**

1

**Explanation:**

K=1 and node=3 ,Kth ancestor of node 3 is 1.

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

**Constraints:**  
1<=N<=104  
1<= K <= 100

Node\* check(Node \*root, int &k, int node, int &val)

{

if(root == NULL)

return NULL;

if(root->data == node || check(root->left, k, node, val) || check(root->right, k, node, val)) {

if(k == 0) {

val = root->data;

return NULL;

}

else

k--;

return root;

}

}

int kthAncestor(Node \*root, int k, int node) {

if(root == NULL)

return -1;

int val = -1;

check(root, k, node, val);

return val;

}

**############################################################**

**13. Deletion in a Binary Tree**

Given a Binary Tree of size **N**, your task is to complete the function **deletionBT()**, that should delete a given node from the tree by making sure that tree shrinks from the bottom (the deleted node is replaced by bottommost and rightmost node).  
Example:



**Your Task:**  
You don't have to take input. Complete the function **deletionBT()**that takes **root** node of the tree and given node value (**Key**) as input parameter and return the root of the modified tree.  
  
**Example 1:**

**Input:**

Key=1

1

/ \

4 7

/ \

5 6

**Output:**

5 4 6 7

**Explanation:**

Modified Tree after deletion the

node with key = 1

7

/

4

/ \

5 6

The Inorder traversal of the modified

tree is 5 4 6 7

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

void deletend(Node \*root, Node \*last)

{

queue<Node\*> q;

q.push(root);

Node \*t;

while(!q.empty())

{

t = q.front();

q.pop();

if(t == last)

{

t = NULL;

delete(last);

return;

}

if(t->right)

{

if(t->right == last)

{

t->right = NULL;

delete(last);

return;

}

else

q.push(t->right);

}

if(t->left)

{

if(t->left == last)

{

t->left = NULL;

delete(last);

return;

}

else

q.push(t->left);

}

}

}

struct Node\* deletionBT(struct Node\* root, int key)

{

queue<Node\*> q;

q.push(root);

Node \*last = NULL, \*node = NULL;

while(!q.empty())

{

last = q.front();

q.pop();

if(last->data == key)

node = last;

if(last->left)

q.push(last->left);

if(last->right)

q.push(last->right);

}

if(node)

{

node->data = last->data;

deletend(root, last);

}

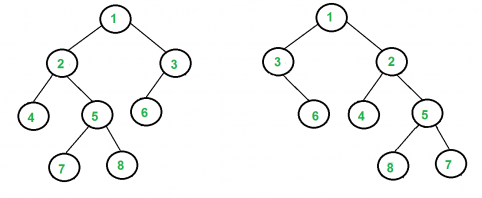
return root;

}

**############################################################**

**14. Check if Tree is Isomorphic**

Given two Binary Trees. Check whether they are Isomorphic or not.

**Note:**  
Two trees are called isomorphic if one can be obtained from another by a series of flips, i.e. by swapping left and right children of several nodes. Any number of nodes at any level can have their children swapped. Two empty trees are isomorphic.  
For example, the following two trees are isomorphic with the following sub-trees flipped: 2 and 3, NULL and 6, 7 and 8.  
[](https://media.geeksforgeeks.org/wp-content/cdn-uploads/ISomorphicTrees-e1368593305854.png)

**Example 1:**

**Input:**

**T1** 1 **T2:** 1

  / \ / \

  2 3 3 2

  / /

  44

**Output:** No

**Example 2:**

**Input:**

**T1** 1 **T2:** 1

  / \ / \

  2 3 3 2

  / \

  44

**Output:** Yes

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function**isomorphic()**that takesthe root nodes of both the Binary Trees as its input and returns True if the two trees are isomorphic. Else, it returns False. (The driver code will print Yes if the returned values are true, otherwise false.)

**Expected Time Complexity:**O(min(M, N)) where M and N are the sizes of the two trees.  
**Expected Auxiliary Space:**O(min(H1, H2)) where H1 and H2 are the heights of the two trees.

**Constraints:**  
1<=Number of nodes<=105

bool isIsomorphic(Node \*root1, Node \*root2)

{

if(root1 == NULL && root2 == NULL)

return true;

if(root1 == NULL || root2 == NULL)

return false;

if(root1->data != root2->data)

return false;

return (isIsomorphic(root1->left, root2->left) && isIsomorphic(root1->right, root2->right)) ||

(isIsomorphic(root1->left, root2->right) && isIsomorphic(root1->right, root2->left));

}

**############################################################**

**15. Children Sum Parent**

Given a Binary Tree. Check whether all of its nodes have the value equal to the sum of their child nodes.

**Example 1:**

**Input:**

10

/

10

**Output:** 1

**Explanation:** Here, every node is sum of

its left and right child.

**Example 2:**

**Input:**

1

/ \

  4 3

  / \

  5 N

**Output:** 0

**Explanation:** Here, 1 is the root node

and 4, 3 are its child nodes. 4 + 3 =

7 which is not equal to the value of

root node. Hence, this tree does not

satisfy the given conditions.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **isSumProperty**() that takes the root Node of the Binary Tree as input and returns 1 if all the nodes in the tree satisfy the following properties. Else, it returns 0.  
For every node, data value must be equal to the sum of data values in left and right children. Consider data value as 0 for NULL child.  Also, leaves are considered to follow the property.

**Expected Time Complexiy:**O(N).  
**Expected Auxiliary Space:**O(Height of the Tree).

**Constraints:**  
1 <= N <= 105  
1 <= Data on nodes <= 105

int isSumProperty(Node \*root)

{

if(root == NULL || root->left == NULL && root->right == NULL)

return 1;

queue<Node\*> q;

q.push(root);

while(!q.empty())

{

Node \*t = q.front();

q.pop();

if(t->left != NULL && t->right != NULL)

{ if(t->data == t->left->data + t->right->data)

{ q.push(t->left);

q.push(t->right);

}

else

return 0;

}

else if(t->left != NULL && t->right == NULL)

if(t->left->data == t->data)

q.push(t->left);

else

return 0;

else if(t->left == NULL && t->right != NULL)

if(t->right->data == t->data)

q.push(t->right);

else

return 0;

else

continue;

}

return 1;

}

**############################################################**

**16. Construct Binary Tree from Parent Array**

Given an array of size **N** that can be used to represents a tree. The array indexes are values in tree nodes and array values give the parent node of that particular index (or node). The value of the root node index would always be **-1** as there is no parent for root. Construct the standard linked representation of Binary Tree from this array representation.

**Example 1:**

**Input:**

N = 7

parent[] = {-1,0,0,1,1,3,5}

**Output:** 0 1 2 3 4 5 6

**Explanation:** the tree generated

will have a structure like

    0

       /   \

      1     2

     / \

    3   4

   /

  5

/

6

**Example 2:**

**Input:**

N = 3

parent[] = {2, 0, -1}

**Output:** 2 0 1

**Explanation:** the tree generated will

have a sturcture like

             2

             /

           0

          /

        1

**Your Task:**  
You don't need to read input or print anything. The task is to complete the function **createTree()**which takes 2 arguments **parent[]** and **N** and returns the root node of the constructed tree.

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

**Constraints:**  
1 <= N <= 103

Node \*createTree(int arr[], int n)

{

vector<Node\*> mem(n);

for(int i=0; i<n; i++)

mem[i] = new Node(i);

Node \*root;

for(int i=0; i<n; i++)

{

int data = arr[i];

if(data == -1)

root = mem[i];

else

{

if(mem[data]->left == NULL)

mem[data]->left = mem[i];

else

mem[data]->right = mem[i];

}

}

return root;

}

**############################################################**

**17. Min distance between two given nodes of a Binary Tree**

Given a binary tree and two node values your task is to find the minimum distance between them.

**Example 1:**

**Input:**

1

  / \

  2 3

a = 2, b = 3

**Output:** 2

**Explanation:** The tree formed is:

      1

     /   \

   2     3

We need the distance between 2 and 3.

Being at node 2, we need to take two

steps ahead in order to reach node 3.

The path followed will be:

2 -> 1 -> 3. Hence, the result is 2.

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **findDist()**which takes the **root**node of the Tree and the two node values **a** and **b** as input parameters and returns the minimum distance between the nodes represented by the two given node values.

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

**Constraints:**  
1 <= Number of nodes <= 104  
1 <= Data of a node <= 105  
  
  
  
**Note:**The **Input/Ouput** format and **Example** given are used for system's internal purpose, and should be used by a user for **Expected Output** only. As it is a function problem, hence a user should not read any input from stdin/console. The task is to complete the function specified, and not to write the full code.

Node\* LCA(Node \*root, int a, int b)

{

if(root == NULL)

return NULL;

if(root->data == a || root->data == b)

return root;

Node \*left = LCA(root->left, a, b);

Node \*right = LCA(root->right, a, b);

if(left != NULL && right != NULL)

return root;

if(left == NULL)

return right;

if(right == NULL)

return left;

}

int dist(Node \*root, int key)

{

if(root == NULL)

return -1;

int d = -1;

if((root->data == key) ||(d = dist(root->left, key)) >= 0 || (d = dist(root->right, key)) >= 0)

return d + 1;

return d;

}

int findDist(Node\* root, int a, int b)

{

Node \*lca = LCA(root, a, b);

int d1 = dist(lca, a);

int d2 = dist(lca, b);

return d1 + d2;

}

**############################################################**

**18. Duplicate subtree in Binary Tree**

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

**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 arguement and returns 1 if the binary tree contains a duplicate sub-tree of size two or more, else 0.

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

**Constraints:**  
1<=length of string<=100

unordered\_map<string, int> mp;

string getstr(Node \*root)

{

if(root == NULL)

return "#";

string str = "";

if(root->left == NULL && root->right == NULL)

{

str = root->data;

return str;

}

str += root->data;

str += getstr(root->left);

str += getstr(root->right);

mp[str]++;

return str;

}

bool dupSub(Node \*root)

{

mp.clear();

getstr(root);

for(auto i : mp)

{ if(i.second > 1)

return true;

}

return false;

}

**############################################################**

**19. Number of Turns in Binary Tree**

Given a binary tree and data value of two of its nodes. Find the number of turns needed to reach from one node to another in the given binary tree.

**Example 1:**

**Input :**

Tree =

1

/ \

2 3

/ \ / \

4 5 6 7

/ / \

8 9 10

first node = 5

second node = 10

**Output:** 4

**Explanation :**

Turns will be at 2, 1, 3, 6.

**Example 2:**

**Input :**

Tree =

1

/ \

2 3

/ \ / \

4 5 6 7

/ / \

8 9 10

first node = 1

second node = 4

**Output :** -1

**Explanation:** No turn is required since

they are in a straight line.

**Your Task:**  
You don't need to read input or print anything. Complete the function **NumberOFTurns()** which takes root node and data value of 2 nodes as input parameters and returns number of turns required to navigate between them. If the two nodes are in a straight line, ie- the path does not involve any turns, return -1.

**Expected Time Complexity:** O(N)  
**Expected Auxiliary Space:**O(Height of Tree)

**Constraints:**  
1 ≤ N ≤ 10^3

Node\* LCA(Node \*root, int a, int b)

{

if(root == NULL)

return NULL;

if(root->data == a || root->data == b)

return root;

Node \*left = LCA(root->left, a, b);

Node \*right = LCA(root->right, a, b);

if(left != NULL && right != NULL)

return root;

if(left != NULL && right == NULL)

return left;

if(left == NULL && right != NULL)

return right;

}

void turn(Node \*root, int data, int &ans, int rotate, int val)

{

if(root == NULL)

return;

if(root->data == data)

{ ans += val;

return;

}

if(rotate == -1)

{

turn(root->left, data, ans, -1, val);

turn(root->right, data, ans, 1, val + 1);

}

else

{

turn(root->left, data, ans, -1, val + 1);

turn(root->right, data, ans, 1, val);

}

}

int NumberOFTurns(struct Node\* root, int a, int b)

{

if(root == NULL)

return -1;

Node \*lca = LCA(root, a, b);

int ans = 0;

turn(lca->left, a, ans, -1, 0);

turn(lca->left, b, ans, -1, 0);

turn(lca->right, a, ans, 1, 0);

turn(lca->right, b, ans, 1, 0);

if(lca->data != a && lca->data != b)

ans++;

if(ans == 0)

return -1;

else

return ans;

}

**############################################################**