**Binary Tree to CDLL:**

Given a **Binary Tree**of**N** edges. The task is to convert this to a **Circular Doubly Linked List (CDLL)** in-place. The **left and right pointers**in nodes are to be used as **previous and next pointers**respectively in **CDLL**. The order of nodes in **CDLL**must be same as **Inorder**of the given **Binary Tree**. The first node of **Inorder traversal**(left most node in **BT**) must be **head node**of the **CDLL**.

**Example 1:**

**Input:**

      1

   /   \

  3     2

**Output:**

3 1 2

2 1 3

**Explanation:** After converting tree to CDLL

when CDLL is is traversed from head to

tail and then tail to head, elements

are displayed as in the output.

**Example 2:**

**Input:**

     10

  /    \

 20    30

/  \

40  60

**Output:**

40 20 60 10 30

30 10 60 20 40

**Explanation:**After converting tree to CDLL,

when CDLL is is traversed from head to

tail and then tail to head, elements

are displayed as in the output.

**Your Task:**  
You don't have to take input or print anything. Complete the function **bTreeToCList()**that takes root as a parameter and **returns**the **head of the list**. The driver code prints the linked list twice, first time in the **right order**, and another time in **reverse order**.

**Constraints:**  
1 <= N <= 103  
1 <= Data of a node <= 104;

Approach 1:

1. We take tree node pointer with the help traversal technique the node the stabilize the connection between doubly pointer node.
2. Return node.

Task first is to create a tree if you have given value of array list:

// Online C++ compiler to run C++ program online

#include <iostream>

using namespace std;

#include<bits/stdc++.h>

class Node

{

public:

int data;

Node \*left,\*right;

Node(int data)

{

this->data=data;

this->left=NULL;

this->right=NULL;

}

};

Node\* insert(Node \*root , int val , queue<Node\*>&q)

{

Node \*node = new Node(val);

if(root==NULL)

{

root=node;

}

else if(q.front()->left==NULL)

{

q.front()->left=node;

}

else

{

q.front()->right=node;

q.pop();

}

q.push(node);

return root;

}

Node\* create(int num[],int n)

{

Node \*root=NULL;

queue<Node\*>q;

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

{

root=insert(root,num[i],q);

}

return root;

}

void printinorder(Node \* root)

{

if(root==NULL)return;

queue<Node\*> q;

q.push(root);

while(!q.empty())

{

cout<<q.front()->data<<" ";

if(q.front()->left!=NULL)

{

q.push(q.front()->left);

}

if(q.front()->right!=NULL)

{

q.push(q.front()->right);

}

q.pop();

}

}

int main() {

// Write C++ code here

// std::cout << "Hello world!";

int arr[] ={10,20,30,40,50,60};

Node \*root=create(arr,6);

printinorder(root);

return 0;

}

Task 2 is actual function code :

Node \*bTreeToCList(Node \*root)

{

//add code here.

stack<Node\*> s;

Node \*dummy= new Node;

Node \*prev;

prev=dummy;

while(1)

{

while(root!=NULL)

{

s.push(root);

root=root->left;

}

if(s.size()==0)

{

break;

}

root=s.top();

s.pop();

Node \*curr=root,\*currprev=prev;

prev->right=curr;

prev=curr;

prev->left=currprev;

root=root->right;

}

dummy->right->left=prev;

prev->right=dummy->right;

return dummy->right;

}

**Expected time complexity:**O(N)

**Expected auxiliary space:**O(h) , where h = height of tree

======================================================================

### K Sum Paths:

Given a binary tree and an integer **K**. Find the number of paths in the tree which have their sum equal to K.  
A path may start from any node and end at any node in the **downward**direction.  
Since the answer may be very large, compute it modulo **109+7**.

**Example 1:**

**Input:**

Tree =

1

/ \

2 3

K = 3

**Output:**   
2

**Explanation:**

Path 1 : 1 + 2 = 3

Path 2 : only leaf node 3

**Example 2:**

**Input:**

Tree =

1

/ \

3 -1

/ \ / \

2 1 4 5

/ / \ \

1 1 2 6

K = 5

**Output:**   
8

**Explanation:**

The following paths sum to K.

3 2

3 1 1

1 3 1

4 1

1 -1 4 1

-1 4 2

5

1 -1 5

**Your Task:**    
You don't need to read input or print anything. Complete the function **sumK()** which takes root node and integer K as input parameters and returns the number of paths that have sum K.

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

**Constraints:**  
1 ≤ N ≤ 2\*104  
-105 ≤ Node Value ≤ 105  
-109 ≤ K ≤ 109

### Approach 1:

### Code:

### Declear a variable answer that will update the answer.

### Map to consider the sum or substraction number is found and store into answer.

### void findk(Node \*root,unordered\_map<int,int>&m,int sum,int &res,int k)

### {

### if(root!=NULL)

### {

### if(root->data+sum==k)

### {

### res++;

### }

### res+=m[root->data+sum-k];

### m[root->data+sum]++;

### findk(root->left,m,sum+root->data,res,k);

### findk(root->right,m,sum+root->data,res,k);

### m[root->data+sum]--;

### }

### }

### 

### int sumK(Node \*root,int k)

### {

### // code here

### int res=0;

### unordered\_map<int,int>m;

### findk(root,m,0,res,k);

### return res;

### }

|  |
| --- |
| **int** main()  {      Node\* root = **new** Node(1);      root->left = **new** Node(2);      root->left->left = **new** Node(1);      root->left->right = **new** Node(2);      root->right = **new** Node(-1);      root->right->left = **new** Node(3);      root->right->left->left = **new** Node(2);      root->right->left->right = **new** Node(5);    **int** k = 3;      cout << "No of paths with sum equals to " <<  k           << " are : "  << sumK(root, k) << "\n";    **return** 0;  } |

### Determine if Two Trees are Identical

Given two binary trees, the task is to find if both of them are identical or not.  
**Note:** You need to return true or false, the printing is done by the driver code.

**Example 1:**

**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 <= 109

Code:  
bool isIdentical(Node \*r1, Node \*r2)

{

//Your Code here

if(r1==NULL||r2==NULL)

{

return r1==r2;

}

return (r1->data==r2->data)&&isIdentical(r1->left,r->left)&&isIdentical(r1->right,r2->right);

}

### ========================================================

### 

### AVL Tree Insertion:

Given an AVL tree and N values to be inserted in the tree. Write a function to insert elements into the given **AVL tree**.

**Note:**  
The tree will be checked after each insertion.   
If it violates the properties of balanced BST, an error message will be printed followed by the inorder traversal of the tree at that moment.  
If instead all insertions are successful, inorder traversal of the tree will be printed.

**Example 1:**

**Input:**N = 3Values to be inserted = {5,1,4}

**Output:**1 4 5  
**Explanation:**Value to be inserted = 5 5

Value to be inserted = 1

5

/

1

Value to be inserted = 4

5 4

/ **LR rotation** / \

1 -----------> 1 5

 \

 4  
Therefore the inorder of the final tree will be 1, 4, 5.

**Example 2:**

**Input:**  
N = 7Values to be inserted = {21,26,30,9,4,14,28}

**Output:**4 9 14 21 26 28 30  
**Explanation:**  
Value to be inserted = 21 21

Value to be inserted = 26

21

\

26

Value to be inserted = 30

21 26

\ **LL rotation** / \

26 -----------> 21 30

\

30  
Value to be inserted = 9  
 26  
 / \  
 21 30  
 /  
9  
Value to be inserted = 4  
 26 26  
 / \ / \  
 21 30 9 30  
 / **RR rotation** / \  
 9 -----------> 4 21  
 /  
4  
Value to be inserted = 14  
 26 21  
 / \ / \  
 9 30 9 26  
 / \ **LR rotation** / \ \  
 4 21 -----------> 4 14 30  
     /  
 14  
Value to be inserted = 28  
 21 21  
 / \ / \  
 9 26 9 28  
 / \ \ **RL rotation** / \ / \  
 4 14 30 -----------> 4 14 26 30  
 /  
 28  
Therefore the inorder of the final tree will be 4, 9, 14, 21, 26, 28, 30.

**Your Task:**  
You don't need to read input or print anything. Complete the function**insertToAVL()** which takes the root of the tree and the value of the node to be inserted as input parameters and returns the root of the modified tree.

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

**Constraints:**  
1 ≤ N ≤ 2000

### Code and Explanation:

### // importnt function of height when node is null then its not return 0.

### int height(Node \*node)

### {

### if(node==NULL)

### {

### return 0;

### }

### return node->height;

### }

### 

### 

### //compute blance factor

### int balenceFactor(Node \*node)

### {

### if(node==NULL)

### {

### return 0;

### }

### return height(node->left)-height(node->right);

### }

### //Right roatation function

### Node\* rightRotation(Node \*x)

### {

### Node \*y=x->left;

### Node \*t=y->right;

### y->right=x;

### x->left=t;

### //updata the height;

### x->height=1+max(height(x->left),height(x->right));

### y->height=1+max(height(y->left),height(y->right));

### return y;

### }

### Node\* leftRotation(Node \*x)

### {

### Node \*y=x->right;

### Node \*t=y->left;

### y->left=x;

### x->right=t;

### //updata the height;

### x->height=1+max(height(x->left),height(x->right));

### y->height=1+max(height(y->left),height(y->right));

### return y;

### }

### Node\* insertToAVL(Node\* node, int data)

### {

### //Your code here

### // step 1: insert the node into bst.

### if(node==NULL)return new Node(data);

### if(node->data>data)

### {

### node->left=insertToAVL(node->left,data);

### }

### else if(node->data<data)

### {

### node->right=insertToAVL(node->right,data);

### }

### else

### return node;

### //step 2: now compute the height of node

### node->height=1+max(height(node->left),height(node->right));

### //step 3: check the balence factor.

### int balance=balenceFactor(node);

### //step 4: check the balance factor\

### //1. L L (Right roatation)

### if(balance>1&&node->left->data>data)

### {

### return rightRotation(node);

### }

### //2. R R (Left roatation)

### if(balance<-1&&node->right->data<data)

### {

### return leftRotation(node);

### }

### //3. R L (Left and right roatation)

### if(balance<-1&&node->right->data>data)

### {

### node->right=rightRotation(node->right);

### return leftRotation(node);

### }

### //4. L R (Left and right roatation)

### if(balance>1&&node->left->data<data)

### {

### node->left=leftRotation(node->left);

### return rightRotation(node);

### }

### return node;

### }

### Q1(GFG) Check whether BST contains Dead End

### Link: <https://www.geeksforgeeks.org/problems/check-whether-bst-contains-dead-end/1>

Given a [Binary Search Tree](https://www.geeksforgeeks.org/binary-search-tree-data-structure/) that contains **unique positive integer values greater than 0**. The task is to complete the function **isDeadEnd** which returns **true** if the BST contains a **dead end**else returns **false**. Here **Dead End**means a **leaf** node, at which no other node can be inserted.

**Example 1:**

**Input :**

  8

/ \

5 9

/ \

2 7

/

1

**Output :**   
Yes

**Explanation :**   
Node 1 is a Dead End in the given BST.

**Example 2:**

**Input :**

  8

/ \

7 10

/ / \

2 9 13

**Output :**   
Yes

**Explanation :**   
Node 9 is a Dead End in the given BST.

**Your Task:** You don't have to input or print anything. Complete the function **isDeadEnd()**which takes **BST** as input and returns a boolean value.

**Expected Time Complexity:** **O(N),** where **N** is the number of nodes in the **BST.  
Expected Space Complexity:** **O(N)**

**Constraints:**  
1 <= N <= 10011 <= Value of Nodes <= 10001

### Approach1: to use any traversal algorithm and store all the node value into set as well as store all the leaf node into another vector.

### Apply the condition while traversal the leaf vector and check the both value+1 and value-1 are found into set return true.

### And one edge case is to if value is 1 return true (not possible to store any value (negative value are not allowed))

### Approach2: Improve the space complexity is using map if leaf node is encounter the frequency map is 2 else 1 (problem statement say that all element occur once.)