DOMAIN WINTER CAMP WORKSHEET

DAY-6 (25/12/2024)

Student Name :- Pratham Kapoor **University ID :- 22BCS10732**

Branch :- B.E. (C.S.E.) **Section/Group :- FL_ 603-A**

Problem-1 :- Given the root of a binary tree, you need to find the sum of all the node values in the binary tree. **(Very Easy)**

Source Code

```
#include<iostream>
using namespace std;

struct TreeNode {
   int Val;
   TreeNode *Left;
   TreeNode *Right;
   TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

int FindSum(TreeNode *Root) {
   if (Root == NULL) return 0;
   return Root->Val + FindSum(Root->Left) + FindSum(Root->Right);
}

int main() {
   int NumberOfNodes;
   cout << "Enter Number of Nodes in the Tree :- ";
   cin >> NumberOfNodes;
   TreeNode *Nodes[NumberOfNodes];
```

```
    for (int I = 0; I < NumberOfNodes; I++) {
        cout << "Enter Value for Node " << I + 1 << " :- ";
        int Value;
        cin >> Value;
        Nodes[I] = new TreeNode(Value);
    }
    for (int I = 0; I < NumberOfNodes; I++)
   {
        cout << "Enter Left and Right Children Indexes for Node " << I + 1 << "
(-1 for None) :- ";
        int Left, Right;
        cin >> Left >> Right;
        if (Left != -1) Nodes[I]->Left = Nodes[Left - 1];
        if (Right != -1) Nodes[I]->Right = Nodes[Right - 1];
    }
    int TotalSum = FindSum(Nodes[0]);
    cout << "\nThe Sum of All Node Values is " << TotalSum;
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 7
Enter Value for Node 1 :- 5
Enter Value for Node 2 :- 2
Enter Value for Node 3 :- 6
Enter Value for Node 4 :- 1
Enter Value for Node 5 :- 3
Enter Value for Node 6 :- 4
Enter Value for Node 7 :- 7
Enter Left and Right Children Indexes for Node 1 (-1 for None) :- 2 3
Enter Left and Right Children Indexes for Node 2 (-1 for None) :- 4 5
Enter Left and Right Children Indexes for Node 3 (-1 for None) :- 6 7
Enter Left and Right Children Indexes for Node 4 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 5 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 6 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 7 (-1 for None) :- -1 -1

The Sum of All Node Values is 28
```

Problem-2 **:-** Given the root of a binary tree, return the preorder traversal of its nodes' values. **(Very Easy)**

Source Code

```
#include<iostream>
#include<vector>
using namespace std;

struct TreeNode {
   int Val;
   TreeNode *Left;
   TreeNode *Right;
   TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

void PreorderTraversal(TreeNode *Root, vector<int> &Result) {
   if (Root == NULL) return;
   Result.push_back(Root->Val);
   PreorderTraversal(Root->Left, Result);
   PreorderTraversal(Root->Right, Result);
}

int main()
{
   int NumberOfNodes;
   cout << "Enter Number of Nodes in the Tree :- ";
   cin >> NumberOfNodes;
   TreeNode *Nodes[NumberOfNodes];
   for (int I = 0; I < NumberOfNodes; I++) {
      cout << "Enter Value for Node " << I + 1 << " :- ";
      int Value;
```

```
        cin >> Value;
        Nodes[I] = new TreeNode(Value);
    }

    for (int I = 0; I < NumberOfNodes; I++)
    {
        cout << "Enter Left and Right Children Indexes for Node " << I + 1 << "
(-1 for None) :- ";
        int Left, Right;
        cin >> Left >> Right;
        if (Left != -1) Nodes[I]->Left = Nodes[Left - 1];
        if (Right != -1) Nodes[I]->Right = Nodes[Right - 1];
    }

    vector<int> PreorderResult;
    PreorderTraversal(Nodes[0], PreorderResult);
    cout << "\nThe Preorder Traversal is :- ";
    for (int Val : PreorderResult)
    {
        cout << Val << " ";
    }

    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 3
Enter Value for Node 1 :- 1
Enter Value for Node 2 :- 2
Enter Value for Node 3 :- 3
Enter Left and Right Children Indexes for Node 1 (-1 for None) :- -1 2
Enter Left and Right Children Indexes for Node 2 (-1 for None) :- 3 -1
Enter Left and Right Children Indexes for Node 3 (-1 for None) :- -1 -1

The Preorder Traversal is :- 1 2 3
```

Problem-3 :- A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node. **(Very Easy)**

Source Code

```
#include<iostream>
using namespace std;

struct TreeNode {
   int Val;
   TreeNode *Left;
   TreeNode *Right;
   TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

int MaxDepth(TreeNode *Root) {
   if (Root == NULL) return 0;
   int LeftDepth = MaxDepth(Root->Left);
   int RightDepth = MaxDepth(Root->Right);
   return max(LeftDepth, RightDepth) + 1;
}

int main() {
   int NumberOfNodes;
   cout << "Enter Number of Nodes in the Tree :- ";
   cin >> NumberOfNodes;
   TreeNode *Nodes[NumberOfNodes];
   for (int I = 0; I < NumberOfNodes; I++) {
      cout << "Enter Value for Node " << I + 1 << " :- ";
      int Value;
      cin >> Value;
      Nodes[I] = new TreeNode(Value);
```

```
    }
    for (int I = 0; I < NumberOfNodes; I++) {
        cout << "Enter Left and Right Children Indexes for Node " << I + 1 << "
(-1 for None) :- ";
        int Left, Right;
        cin >> Left >> Right;
        if (Left != -1) Nodes[I]->Left = Nodes[Left - 1];
        if (Right != -1) Nodes[I]->Right = Nodes[Right - 1];
    }

    int Depth = MaxDepth(Nodes[0]);
    cout << "\nThe Maximum Depth of the Tree is " << Depth;
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 2
Enter Value for Node 1 :- 1
Enter Value for Node 2 :- 2
Enter Left and Right Children Indexes for Node 1 (-1 for None) :- -1 2
Enter Left and Right Children Indexes for Node 2 (-1 for None) :- -1 -1

The Maximum Depth of the Tree is 2
```

Problem-4 :- Two binary trees are considered the same if they are structurally identical, and the nodes have the same value. **(Easy)**

Source Code

```
#include<iostream>
using namespace std;

struct TreeNode
{
```

```
    int Val;
    TreeNode *Left;
    TreeNode *Right;
    TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

bool IsSameTree(TreeNode *P, TreeNode *Q)
{
    if (P == NULL && Q == NULL) return true;
    if (P == NULL || Q == NULL) return false;
    if (P->Val != Q->Val) return false;
    return IsSameTree(P->Left, Q->Left) && IsSameTree(P->Right, Q->Right);
}

int main()
{
    int NumberOfNodes;
    cout << "Enter Number of Nodes in the First Tree :- ";
    cin >> NumberOfNodes;
    TreeNode *NodesP[NumberOfNodes];
    for (int I = 0; I < NumberOfNodes; I++) {
        cout << "Enter Value for Node " << I + 1 << " in Tree P :- ";
        int Value;
        cin >> Value;
        NodesP[I] = new TreeNode(Value);
    }

    for (int I = 0; I < NumberOfNodes; I++)
    {
        cout << "Enter Left and Right Children Indexes for Node " << I + 1 << " in Tree P (-1 for None) :- ";
```

```
        int Left, Right;
        cin >> Left >> Right;
        if (Left != -1) NodesP[I]->Left = NodesP[Left - 1];
        if (Right != -1) NodesP[I]->Right = NodesP[Right - 1];
    }

    cout << "\nEnter Number of Nodes in the Second Tree :- ";
    cin >> NumberOfNodes;
    TreeNode *NodesQ[NumberOfNodes];
    for (int I = 0; I < NumberOfNodes; I++) {
        cout << "Enter Value for Node " << I + 1 << " in Tree Q :- ";
        int Value;
        cin >> Value;
        NodesQ[I] = new TreeNode(Value);
    }

    for (int I = 0; I < NumberOfNodes; I++)
   {
        cout << "Enter Left and Right Children Indexes for Node " << I + 1 << "
in Tree Q (-1 for None) :- ";
        int Left, Right;
        cin >> Left >> Right;
        if (Left != -1) NodesQ[I]->Left = NodesQ[Left - 1];
        if (Right != -1) NodesQ[I]->Right = NodesQ[Right - 1];
    }

    bool Same = IsSameTree(NodesP[0], NodesQ[0]);
    if (Same) cout << "\nTrue, the Trees are Identical.";
    else cout << "\nFalse, the Trees are Not Identical.";
    return 0;
}
```

Output

```
Enter Number of Nodes in the First Tree :- 3
Enter Value for Node 1 in Tree P :- 1
Enter Value for Node 2 in Tree P :- 2
Enter Value for Node 3 in Tree P :- 3
Enter Left and Right Children Indexes for Node 1 in Tree P (-1 for None) :- 2 3
Enter Left and Right Children Indexes for Node 2 in Tree P (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 3 in Tree P (-1 for None) :- -1 -1

Enter Number of Nodes in the Second Tree :- 3
Enter Value for Node 1 in Tree Q :- 1
Enter Value for Node 2 in Tree Q :- 2
Enter Value for Node 3 in Tree Q :- 3
Enter Left and Right Children Indexes for Node 1 in Tree Q (-1 for None) :- 2 3
Enter Left and Right Children Indexes for Node 2 in Tree Q (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 3 in Tree Q (-1 for None) :- -1 -1

True, the Trees are Identical.
```

Problem-5 :- Given a binary tree and a sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum. Return false if no such path can be found. **(Easy)**

Source Code

```
#include<iostream>
using namespace std;

struct TreeNode {
   int Val;
   TreeNode *Left;
   TreeNode *Right;
   TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

bool HasPathSum(TreeNode* Root, int TargetSum) {
   if (Root == NULL) return false;
   if (Root->Left == NULL && Root->Right == NULL) {
```

```
        return TargetSum == Root->Val;
    }
    return HasPathSum(Root->Left, TargetSum - Root->Val) ||
HasPathSum(Root->Right, TargetSum - Root->Val);
}

int main() {
    int NumberOfNodes, TargetSum;
    cout << "Enter Number of Nodes in the Tree :- ";
    cin >> NumberOfNodes;
    cout << "Enter Target Sum :- ";
    cin >> TargetSum;
    cout << endl ;
    TreeNode *Nodes[NumberOfNodes];
    for (int I = 0; I < NumberOfNodes; I++) {
        cout << "Enter Value for Node " << I + 1 << " :- ";
        int Value;
        cin >> Value;
        Nodes[I] = new TreeNode(Value);
    }
    cout << endl ;
    for (int I = 0; I < NumberOfNodes; I++) {
        cout << "Enter Left and Right Children Indexes for Node " << I + 1 << "
(-1 for None) :- ";
        int Left, Right;
        cin >> Left >> Right;
        if (Left != -1) Nodes[I]->Left = Nodes[Left - 1];
        if (Right != -1) Nodes[I]->Right = Nodes[Right - 1];
    }

    if (HasPathSum(Nodes[0], TargetSum)) {
```

```
        cout << "\nTrue, there is a Root-to-Leaf Path with the Given Sum" << endl;
    } else {
        cout << "\nFalse, No Root-to-Leaf Path with the Given Sum Exists" << endl;
    }
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 15
Enter Target Sum :- 22

Enter Value for Node 1 :- 5
Enter Value for Node 2 :- 4
Enter Value for Node 3 :- 8
Enter Value for Node 4 :- 11
Enter Value for Node 5 :- -1
Enter Value for Node 6 :- 13
Enter Value for Node 7 :- 4
Enter Value for Node 8 :- 7
Enter Value for Node 9 :- 2
Enter Value for Node 10 :- -1
Enter Value for Node 11 :- -1
Enter Value for Node 12 :- -1
Enter Value for Node 13 :- -1
Enter Value for Node 14 :- -1
Enter Value for Node 15 :- 1

Enter Left and Right Children Indexes for Node 1 (-1 for None) :- 2 3
Enter Left and Right Children Indexes for Node 2 (-1 for None) :- 4 5
Enter Left and Right Children Indexes for Node 3 (-1 for None) :- 6 7
Enter Left and Right Children Indexes for Node 4 (-1 for None) :- 8 9
Enter Left and Right Children Indexes for Node 5 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 6 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 7 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 8 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 9 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 10 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 11 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 12 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 13 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 14 (-1 for None) :- -1 -1
Enter Left and Right Children Indexes for Node 15 (-1 for None) :- -1 -1

True, there is a Root-to-Leaf Path with the Given Sum
```

Problem-6 :- Given two integer arrays preorder and inorder where preorder is the preorder traversal of a binary tree and inorder is the inorder traversal of the same tree, construct and return the binary tree. **(Medium)**

Source Code

```
#include<iostream>
#include<vector>
using namespace std;

struct TreeNode {
   int Val;
   TreeNode *Left;
   TreeNode *Right;
   TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

TreeNode* BuildTree(vector<int>& Preorder, vector<int>& Inorder, int& PreIndex, int InStart, int InEnd) {
   if (InStart > InEnd) return NULL;

   int RootVal = Preorder[PreIndex++];
   TreeNode* Root = new TreeNode(RootVal);

   int InIndex = InStart;
   while (Inorder[InIndex] != RootVal) {
      InIndex++;
   }

   Root->Left = BuildTree(Preorder, Inorder, PreIndex, InStart, InIndex - 1);
   Root->Right = BuildTree(Preorder, Inorder, PreIndex, InIndex + 1, InEnd);
```

```
    return Root;
}

void InorderTraversal(TreeNode* Root) {
    if (Root == NULL) return;
    InorderTraversal(Root->Left);
    cout << Root->Val << " ";
    InorderTraversal(Root->Right);
}

int main() {
    int N;
    cout << "Enter Number of Nodes in the Tree :- ";
    cin >> N;

    vector<int> Preorder(N), Inorder(N);

    cout << "Enter Preorder Traversal :- ";
    for (int I = 0; I < N; I++) {
        cin >> Preorder[I];
    }

    cout << "Enter Inorder Traversal :- ";
    for (int I = 0; I < N; I++) {
        cin >> Inorder[I];
    }

    int PreIndex = 0;
    TreeNode* Root = BuildTree(Preorder, Inorder, PreIndex, 0, N - 1);

    cout << "\nThe Constructed Tree is ";
    InorderTraversal(Root);
```

```
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 1
Enter Preorder Traversal :- -1
Enter Inorder Traversal :- -1

The Constructed Tree is -1
```

Problem-7 :- Given two integer arrays inorder and postorder where inorder is the inorder traversal of a binary tree and postorder is the postorder traversal of the same tree, construct and return the binary tree. **(Medium)**

Source Code

```
#include<iostream>
#include<vector>
using namespace std;

struct TreeNode {
    int Val;
    TreeNode *Left;
    TreeNode *Right;
    TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

TreeNode* BuildTree(vector<int>& Inorder, vector<int>& Postorder, int& PostIndex, int InStart, int InEnd) {
    if (InStart > InEnd) return NULL;

    int RootVal = Postorder[PostIndex--];
```

```
    TreeNode* Root = new TreeNode(RootVal);

    int InIndex = InStart;
    while (Inorder[InIndex] != RootVal) {
        InIndex++;
    }

    Root->Right = BuildTree(Inorder, Postorder, PostIndex, InIndex + 1, InEnd);
    Root->Left = BuildTree(Inorder, Postorder, PostIndex, InStart, InIndex - 1);

    return Root;
}

void InorderTraversal(TreeNode* Root) {
    if (Root == NULL) return;
    InorderTraversal(Root->Left);
    cout << Root->Val << " ";
    InorderTraversal(Root->Right);
}

int main() {
    int N;
    cout << "Enter Number of Nodes in the Tree :- ";
    cin >> N;

    vector<int> Inorder(N), Postorder(N);

    cout << "Enter Inorder Traversal :- ";
    for (int I = 0; I < N; I++) {
        cin >> Inorder[I];
    }
```

```
    cout << "Enter Postorder Traversal :- ";
    for (int I = 0; I < N; I++) {
        cin >> Postorder[I];
    }

    int PostIndex = N - 1;
    TreeNode* Root = BuildTree(Inorder, Postorder, PostIndex, 0, N - 1);

    cout << "\nThe Constructed Tree is ";
    InorderTraversal(Root);
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 1
Enter Inorder Traversal :- -1
Enter Postorder Traversal :- -1

The Constructed Tree is -1
```

Problem-8 :- Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree. The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow a node to be a descendant of itself). **(Medium)**

Source Code

```
#include<iostream>
using namespace std;

struct TreeNode {
    int Val;
```

```
    TreeNode *Left;
    TreeNode *Right;
    TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

TreeNode* LowestCommonAncestor(TreeNode* Root, TreeNode* P,
TreeNode* Q) {
    if (Root == NULL || Root == P || Root == Q) {
        return Root;
    }

    TreeNode* Left = LowestCommonAncestor(Root->Left, P, Q);
    TreeNode* Right = LowestCommonAncestor(Root->Right, P, Q);

    if (Left == NULL) {
        return Right;
    }
    if (Right == NULL) {
        return Left;
    }
    return Root;
}

TreeNode* Insert(TreeNode* Root, int Val) {
    if (Root == NULL) {
        return new TreeNode(Val);
    }
    if (Val < Root->Val) {
        Root->Left = Insert(Root->Left, Val);
    } else {
        Root->Right = Insert(Root->Right, Val);
    }
```

```
    return Root;
}

int main() {
    TreeNode* Root = NULL;
    int N, Val;

    cout << "Enter Number of Nodes in the Tree :- ";
    cin >> N;

    cout << "Enter Values for the Nodes of the Tree :- ";
    for (int i = 0; i < N; i++) {
        cin >> Val;
        Root = Insert(Root, Val);
    }

    int p_val, q_val;
    cout << "Enter the Value of Node P :- ";
    cin >> p_val;
    cout << "Enter the Value of Node Q :- ";
    cin >> q_val;

    TreeNode* p = Root;
    TreeNode* q = Root;

    while (p != NULL && p->Val != p_val) {
        if (p_val < p->Val) p = p->Left;
        else p = p->Right;
    }

    while (q != NULL && q->Val != q_val) {
        if (q_val < q->Val) q = q->Left;
```

```
        else q = q->Right;
    }

    TreeNode* LCA = LowestCommonAncestor(Root, p, q);
    if (LCA != NULL) {
        cout << "\nThe Lowest Common Ancestor is " << LCA->Val << endl;
    } else {
        cout << "\nNo Lowest Common Ancestor Found" << endl;
    }
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 2
Enter Values for the Nodes of the Tree :- 1 2
Enter the Value of Node P :- 1
Enter the Value of Node Q :- 2

The Lowest Common Ancestor is 1
```

Problem-9 :- Given the root of a binary tree, return the level order traversal of its nodes' values. (i.e., from left to right, level by level). **(Medium)**

Source Code

```
#include<iostream>
#include<vector>
#include<queue>
using namespace std;

struct TreeNode {
    int Val;
    TreeNode *Left;
```

```
    TreeNode *Right;
    TreeNode(int X) : Val(X), Left(NULL), Right(NULL) {}
};

vector<vector<int>> LevelOrder(TreeNode* Root) {
    vector<vector<int>> result;
    if (Root == NULL) return result;

    queue<TreeNode*> q;
    q.push(Root);

    while (!q.empty()) {
        int levelSize = q.size();
        vector<int> currentLevel;

        for (int i = 0; i < levelSize; i++) {
            TreeNode* node = q.front();
            q.pop();
            currentLevel.push_back(node->Val);

            if (node->Left != NULL) q.push(node->Left);
            if (node->Right != NULL) q.push(node->Right);
        }
        result.push_back(currentLevel);
    }
    return result;
}

TreeNode* Insert(TreeNode* Root, int Val)
{
    if (Root == NULL) {
        return new TreeNode(Val);
```

```
    }
    if (Val < Root->Val) {
        Root->Left = Insert(Root->Left, Val);
    } else {
        Root->Right = Insert(Root->Right, Val);
    }
    return Root;
}

int main()
{
    TreeNode* Root = NULL;
    int N, Val;
    cout << "Enter Number of Nodes in the Tree :- ";
    cin >> N;
    cout << "Enter the Values for the Nodes of the Tree :- ";
    for (int i = 0; i < N; i++) {
        cin >> Val;
        Root = Insert(Root, Val);
    }
    vector<vector<int>> result = LevelOrder(Root);
    cout << "\nThe Level Order Traversal of Tree is ";
    for (auto level : result)
   {
        for (int value : level) {
            cout << value << " ";
        }
        cout << endl;
    }
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 1
Enter the Values for the Nodes of the Tree :- 1

The Level Order Traversal of Tree is 1
```

Problem-10 :- Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between). **(Hard)**

Source Code

```
#include<iostream>
#include<queue>
#include<vector>
#include<algorithm>
using namespace std;

struct Node {
   int val;
   Node *left;
   Node *right;
   Node(int x) : val(x), left(NULL), right(NULL) {}
};

vector<vector<int>> ZigzagLevelOrder(Node* root) {
   vector<vector<int>> result;
   if (root == NULL) return result;

   queue<Node*> q;
   q.push(root);
```

```
    bool leftToRight = true;

    while (!q.empty()) {
        int levelSize = q.size();
        vector<int> currentLevel;

        for (int i = 0; i < levelSize; i++) {
            Node* node = q.front();
            q.pop();
            currentLevel.push_back(node->val);

            if (node->left != NULL) q.push(node->left);
            if (node->right != NULL) q.push(node->right);
        }

        if (!leftToRight) {
            reverse(currentLevel.begin(), currentLevel.end());
        }

        result.push_back(currentLevel);
        leftToRight = !leftToRight;
    }
    return result;
}

Node* Insert(Node* root, int val) {
    if (root == NULL) {
        return new Node(val);
    }
    if (val < root->val) {
        root->left = Insert(root->left, val);
    } else {
```

```
        root->right = Insert(root->right, val);
    }
    return root;
}

int main() {
    Node* root = NULL;
    int N, Val;
    cout << "Enter Number of Nodes in the Tree :- ";
    cin >> N;
    cout << "Enter the Values for the Nodes of the Tree :- ";
    for (int i = 0; i < N; i++) {
        cin >> Val;
        root = Insert(root, Val);
    }

    vector<vector<int>> zigzagOrder = ZigzagLevelOrder(root);

    cout << "\nThe Zigzag Level Order Traversal is [";
    for (int i = 0; i < zigzagOrder.size(); i++) {
        cout << "[";
        for (int j = 0; j < zigzagOrder[i].size(); j++) {
            cout << zigzagOrder[i][j];
            if (j < zigzagOrder[i].size() - 1) cout << ",";
        }
        cout << "]";
        if (i < zigzagOrder.size() - 1) cout << ",";
    }
    cout << "]" << endl;

    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 1
Enter the Values for the Nodes of the Tree :- 1

The Zigzag Level Order Traversal is [[1]]
```

Problem-11 :- A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root. The path sum of a path is the sum of the node's values in the path. Given the root of a binary tree, return the maximum path sum of any non-empty path. **(Hard)**

Source Code

```
#include<iostream>
#include<algorithm>
using namespace std;

struct Node {
   int val;
   Node *left;
   Node *right;
   Node(int x) : val(x), left(NULL), right(NULL) {}
};

class Solution {
public:
   int maxPathSum(Node* root) {
      int maxSum = INT_MIN;
      maxGain(root, maxSum);
```

```
        return maxSum;
    }

private:
    int maxGain(Node* node, int &maxSum) {
        if (node == NULL) return 0;

        int leftGain = max(maxGain(node->left, maxSum), 0);
        int rightGain = max(maxGain(node->right, maxSum), 0);

        int priceNewPath = node->val + leftGain + rightGain;

        maxSum = max(maxSum, priceNewPath);

        return node->val + max(leftGain, rightGain);
    }
};

Node* Insert(Node* root, int val) {
    if (root == NULL) {
        return new Node(val);
    }
    if (val < root->val) {
        root->left = Insert(root->left, val);
    } else {
        root->right = Insert(root->right, val);
    }
    return root;
}

int main() {
    Node* root = NULL;
```

```
    int N, Val;
    cout << "Enter Number of Nodes in the Tree :- ";
    cin >> N;
    cout << "Enter the Values for the Nodes of the Tree :- ";
    for (int i = 0; i < N; i++) {
        cin >> Val;
        root = Insert(root, Val);
    }
    Solution solution;
    int maxPath = solution.maxPathSum(root);
    cout << "\nThe Maximum Path Sum is " << maxPath << endl;
    return 0;
}
```

Output

```
Enter Number of Nodes in the Tree :- 3
Enter the Values for the Nodes of the Tree :- 1 2 3

The Maximum Path Sum is 6
```

Problem-12 :- Given a binary search tree (BST), write a function to find the kth smallest element in the tree. **(Hard)**

Source Code

```
#include<iostream>
#include<queue>
#include<sstream>
using namespace std;

struct Node{
    int Value;
```

```
    Node* Left;
    Node* Right;
    Node(int V): Value(V), Left(NULL), Right(NULL) {}
};

Node* BuildTreeFromInput(int NumberOfNodes){
    string Input;
    cout<<"Enter the Values of the Tree in Level Order - ";
    cin.ignore();
    getline(cin, Input);
    stringstream Ss(Input);
    string Token;

    if(!(Ss>>Token)||Token=="#"){
        return NULL;
    }
    Node* Root=new Node(stoi(Token));
    queue<Node*> Q;
    Q.push(Root);
    NumberOfNodes--;

    while(!Q.empty()&&NumberOfNodes>0){
        Node* Current=Q.front();
        Q.pop();

        if(Ss>>Token){
            if(Token=="#"){
                Current->Left=NULL;
            } else {
                Current->Left=new Node(stoi(Token));
                Q.push(Current->Left);
            }
```

```
            NumberOfNodes--;
        }

        if(NumberOfNodes>0&&Ss>>Token){
            if(Token=="#"){
                Current->Right=NULL;
            } else {
                Current->Right=new Node(stoi(Token));
                Q.push(Current->Right);
            }
            NumberOfNodes--;
        }
    }
    return Root;
}

void InOrder(Node* Root,int& K,int& Result){
    if(Root==NULL||K<=0){
        return;
    }
    InOrder(Root->Left,K,Result);
    K--;
    if(K==0){
        Result=Root->Value;
        return;
    }
    InOrder(Root->Right,K,Result);
}

int FindKthSmallest(Node* Root,int K){
    int Result=-1;
    InOrder(Root,K,Result);
```

```
    return Result;
}

int main(){
    int NumberOfNodes,K;
    cout<<"Enter the Number of Nodes in the Tree :- ";
    cin>>NumberOfNodes;
    Node* Root=BuildTreeFromInput(NumberOfNodes);
    cout<<"\nEnter the Value of K :- ";
    cin>>K;
    int KthSmallest=FindKthSmallest(Root,K);
    if(KthSmallest!=-1){
        cout<<"\nThe Kth Smallest Element is "<<KthSmallest<<endl;
    }
    else{
        cout<<"\nInvalid K Value"<<endl;
    }
    return 0;
}
```

Output

```
Enter the Number of Nodes in the Tree :- 5
Enter the Values of the Tree in Level Order - 3 1 4 # 2

Enter the Value of K :- 1

The Kth Smallest Element is 1
```

Problem-13 :- Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom. **(Hard)**

Source Code

```
#include<iostream>
#include<queue>
#include<vector>
#include<sstream>
using namespace std;

struct Node{
   int Value;
   Node* Left;
   Node* Right;
   Node(int V): Value(V), Left(NULL), Right(NULL) {}
};

Node* BuildTreeFromInput(int NumberOfNodes){
   string Input;
   cout<<"Enter the Tree Values in Level Order - ";
   cin.ignore();
   getline(cin, Input);
   stringstream Ss(Input);
   string Token;

   if(!(Ss>>Token)||Token=="#"){
      return NULL;
   }
   Node* Root=new Node(stoi(Token));
   queue<Node*> Q;
   Q.push(Root);
   NumberOfNodes--;

   while(!Q.empty()&&NumberOfNodes>0){
```

```
        Node* Current=Q.front();
        Q.pop();

        if(Ss>>Token){
            if(Token=="#"){
                Current->Left=NULL;
            } else {
                Current->Left=new Node(stoi(Token));
                Q.push(Current->Left);
            }
            NumberOfNodes--;
        }

        if(NumberOfNodes>0&&Ss>>Token){
            if(Token=="#"){
                Current->Right=NULL;
            } else {
                Current->Right=new Node(stoi(Token));
                Q.push(Current->Right);
            }
            NumberOfNodes--;
        }
    }
    return Root;
}

vector<int> RightSideView(Node* Root){
    vector<int> Result;
    if(Root==NULL){
        return Result;
    }
    queue<Node*> Q;
```

```
    Q.push(Root);
    while(!Q.empty()){
        int Size=Q.size();
        for(int I=0;I<Size;I++){
            Node* Current=Q.front();
            Q.pop();
            if(I==Size-1){
                Result.push_back(Current->Value);
            }
            if(Current->Left){
                Q.push(Current->Left);
            }
            if(Current->Right){
                Q.push(Current->Right);
            }
        }
    }
    return Result;
}

int main(){
    int NumberOfNodes;
    cout<<"Enter the Number of Nodes in the Tree :- ";
    cin>>NumberOfNodes;
    Node* Root=BuildTreeFromInput(NumberOfNodes);
    vector<int> RightView=RightSideView(Root);
    cout<<"\nThe Right Side View of the Tree is [";
    for(size_t I=0;I<RightView.size();I++){
        cout<<RightView[I];
        if(I<RightView.size()-1){
            cout<<",";
        }
```

```
    }
    cout<<"]"<<endl;
    return 0;
}
```

Output

```
Enter the Number of Nodes in the Tree :- 7
Enter the Tree Values in Level Order - 1 2 3 # 5 # 4

The Right Side View of the Tree is [1,3,4]
```

Problem-14 :- You are given a tree (i.e. a connected, undirected graph that has no cycles) rooted at node 0 consisting of n nodes numbered from 0 to n - 1. The tree is represented by a 0-indexed array parent of size n, where parent[i] is the parent of node i. Since node 0 is the root, parent[0] == -1. You are also given a string s of length n, where s[i] is the character assigned to node i. Return the length of the longest path in the tree such that no pair of adjacent nodes on the path have the same character assigned to them. **(Very Hard)**

Source Code

```
#include<iostream>
#include<vector>
#include<unordered_map>
#include<algorithm>
using namespace std;

int Dfs(int Node,const vector<vector<int>>&Tree,const string&S,int&MaxPath) {
    int Longest=0,SecondLongest=0;
    for(int Child:Tree[Node]) {
        int Length=Dfs(Child,Tree,S,MaxPath);
```

```
        if(S[Node]!=S[Child]) {
            if(Length>Longest) {
                SecondLongest=Longest;
                Longest=Length;
            } else if(Length>SecondLongest) {
                SecondLongest=Length;
            }
        }
    }
    MaxPath=max(MaxPath,Longest+SecondLongest+1);
    return Longest+1;
}

int LongestPath(vector<int>&Parent,string&S) {
    int N=Parent.size();
    vector<vector<int>>Tree(N);

    for(int I=1;I<N;I++)
    {
        Tree[Parent[I]].push_back(I);
    }

    int MaxPath=0;
    Dfs(0,Tree,S,MaxPath);
    return MaxPath;
}

int main()
{
    int N;
    cout<<"Enter the Number Of Nodes :- ";cin>>N;
    vector<int>Parent(N);
```

```
    cout<<"Enter the Parent Array :- ";
    for(int I=0;I<N;I++)
    {
        cin>>Parent[I];
    }
    string S;
    cout<<"Enter the String :- ";cin>>S;
    cout<<"\nThe Length Of the Longest Path is "<<LongestPath(Parent,S);
    return 0;
}
```

Output

```
Enter the Number Of Nodes :- 6
Enter the Parent Array :- -1 0 0 1 1 2
Enter the String :- abacbe

The Length Of the Longest Path is 3
```

Problem-15 :- Alice has an undirected tree with n nodes labeled from 0 to n - 1. The tree is represented as a 2D integer array edges of length n - 1 where edges[i] = [ai, bi] indicates that there is an edge between nodes ai and bi in the tree. Alice wants Bob to find the root of the tree. She allows Bob to make several guesses about her tree. In one guess, he does the following :- Chooses two distinct integers u and v such that there exists an edge [u, v] in the tree. He tells Alice that u is the parent of v in the tree. Bob's guesses are represented by a 2D integer array guesses where guesses[j] = [uj, vj] indicates Bob guessed uj to be the parent of vj. Alice being lazy, does not reply to each of Bob's guesses, but just says that at least k of his guesses are true. Given the 2D integer arrays edges, guesses and the integer k, return the number of possible nodes that can be the root of Alice's tree. If there is no such tree, return 0. **(Very Hard)**

Source Code

```
#include<iostream>
#include<vector>
#include<unordered_map>
#include<unordered_set>
using namespace std;

void Dfs(int Node,int Parent,unordered_map<int,vector<int>>&Tree,unordered_set<string>&GuessesSet,int&Count) {
   for(int Child:Tree[Node]) {
      if(Child==Parent)continue;
      if(GuessesSet.count(to_string(Node)+"-"+to_string(Child)))Count++;
      Dfs(Child,Node,Tree,GuessesSet,Count);
   }
}

void Recalculate(int Node,int Parent,unordered_map<int,vector<int>>&Tree,unordered_set<string>&GuessesSet,vector<int>&CountTrue,int&Result,int K) {
   for(int Child:Tree[Node]) {
      if(Child==Parent)continue;
      CountTrue[Child]=CountTrue[Node];
      if(GuessesSet.count(to_string(Node)+"-"+to_string(Child)))CountTrue[Child]--;
      if(GuessesSet.count(to_string(Child)+"-"+to_string(Node)))CountTrue[Child]++;
      if(CountTrue[Child]>=K)Result++;
      Recalculate(Child,Node,Tree,GuessesSet,CountTrue,Result,K);
   }
}\
```

```
int
CountPossibleRoots(vector<vector<int>>&Edges,vector<vector<int>>&Gue
sses,int K)
{
   int N=Edges.size()+1;
   unordered_map<int,vector<int>>Tree;
   for(auto&E:Edges) {
      Tree[E[0]].push_back(E[1]);
      Tree[E[1]].push_back(E[0]);
   }
   unordered_set<string>GuessesSet;
   for(auto&G:Guesses) {
      GuessesSet.insert(to_string(G[0])+"-"+to_string(G[1]));
   }
   vector<int>CountTrue(N,0);
   int Count=0;
   Dfs(0,-1,Tree,GuessesSet,Count);
   CountTrue[0]=Count;
   int Result=CountTrue[0]>=K?1:0;
   Recalculate(0,-1,Tree,GuessesSet,CountTrue,Result,K);
   return Result;
}

int main()
{
   int E,G,K;
   cout<<"Enter the Number Of Edges :- ";cin>>E;
   vector<vector<int>>Edges(E,vector<int>(2));
   for(int I=0;I<E;I++) {
      cout<<"Enter the Edge "<<I+1<<" :- ";
      cin>>Edges[I][0]>>Edges[I][1];
   }
```

```
    cout<<"\nEnter the Number Of Guesses :- ";cin>>G;
    vector<vector<int>>Guesses(G,vector<int>(2));
    for(int I=0;I<G;I++) {
        cout<<"Enter the Guess "<<I+1<<" :- ";
        cin>>Guesses[I][0]>>Guesses[I][1];
    }
    cout<<"\nEnter the Minimum Value of K :- ";cin>>K;
    cout<<"\nThe Number Of Possible Roots are "<<CountPossibleRoots(Edges,Guesses,K);
    return 0;
}
```

Output

```
Enter the Number Of Edges :- 4
Enter the Edge 1 :- 0 1
Enter the Edge 2 :- 1 2
Enter the Edge 3 :- 1 3
Enter the Edge 4 :- 4 2

Enter the Number Of Guesses :- 4
Enter the Guess 1 :- 1 3
Enter the Guess 2 :- 0 1
Enter the Guess 3 :- 1 0
Enter the Guess 4 :- 2 4

Enter the Minimum Value of K :- 3

The Number Of Possible Roots are 3
```