DSA SERIES

- Learn Coding

Topic to be Covered today

Binary Tree Problems

LETS START TODAY'S LECTURE

199. Binary Tree Right Side View

```
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        if(root == NULL) return {};


        vector<int> result;


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


        while(!q.empty()){
            int n = q.size();
            TreeNode* node = NULL;


            while(n--){
                node = q.front();
                q.pop();
```

```
                if(node->left)
                q.push(node->left);


                if(node->right)
                q.push(node->right);
            }
            result.push_back(node->val);
        }


        return result;
    }
};
```

102. Binary Tree Level Order Traversal

```
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>> result;


        if (root == NULL)
            return result;


        queue<TreeNode*> que;
        que.push(root);


        while (!que.empty()) {
            int n = que.size();
            vector<int> ans;
            while (n--) {
                TreeNode* node = que.front();
                que.pop();
```

```
                ans.push_back(node->val);


                if (node->left)
                    que.push(node->left);
                if (node->right)
                    que.push(node->right);
            }
            result.push_back(ans);
        }


        return result;
    }
};
```

107. Binary Tree Level Order Traversal II

```
class Solution {
public:
    vector<vector<int>> levelOrderBottom(TreeNode* root) {


        vector<vector<int>> result;


        if (root == NULL)
            return result;


        queue<TreeNode*> que;
        que.push(root);


        while (!que.empty()) {
            int n = que.size();
            vector<int> ans;
            while (n--) {
```

```
                TreeNode* node = que.front();
                que.pop();


                ans.push_back(node->val);


                if (node->left)
                    que.push(node->left);
                if (node->right)
                    que.push(node->right);
            }
            result.push_back(ans);
        }


        reverse(result.begin(), result.end());


        return result;
    }
};
```

993. Cousins in Binary Tree

```
class Solution {
public:
    bool isCousins(TreeNode* root, int x, int y) {
        if (root == NULL)
            return false;


        queue<pair<TreeNode*, TreeNode*>> que;
        que.push({root, NULL});
        int depth = 0;


        TreeNode* parentX = NULL;
        TreeNode* parentY = NULL;


        int depthX = -1;
        int depthY = -1;
```

```
while (!que.empty()) {
    int n = que.size();


    while (n--) {
        auto [node, parent] = que.front();
        que.pop();


        if (node->val == x) {
            parentX = parent;
            depthX = depth;
        }


        if (node->val == y) {
            parentY = parent;
            depthY = depth;
        }
```

```
                if (node->left)
                    que.push({node->left, node});
                if (node->right)
                    que.push({node->right, node});
            }
            depth++;
        }


        return (parentX != parentY) && (depthX == depthY);
    }
};
```

637. Average of Levels in Binary Tree

```
class Solution {
public:
    vector<double> averageOfLevels(TreeNode* root) {
        if (!root)
            return {};


        vector<double> ans;


        queue<TreeNode*> que;
        que.push(root);


        while (!que.empty()) {
            int n = que.size();
            int temp = n;


            double val = 0.0;
```

```
        while (n--) {
                        TreeNode* node = que.front();
                        que.pop();


                        val += node->val;


                        if (node->left) {
                            que.push(node->left);
                        }


                        if (node->right) {
                            que.push(node->right);
                        }
                    }
                    ans.push_back(val / temp);
                }
                return ans;
            }
};
```

257. Binary Tree Paths

```
class Solution {
public:
    vector<string> result;


    void dfs(TreeNode* root, string path) {


        if (!root)
            return;


        if (!path.empty()) {
            path += "->";
        }
        path += to_string(root->val);


        if (!root->left && !root->right) {
            result.push_back(path);
            return;
        }
```

```
        dfs(root->left, path);
        dfs(root->right, path);
    }
    vector<string> binaryTreePaths(TreeNode* root) {
        dfs(root, "");
        return result;
    }
};
```

112. Path Sum

```
class Solution {
public:
    bool dfs(TreeNode* root, int sum, int& targetSum) {
        if (root == NULL)
            return false;


        sum += root->val;


        if (root->left == NULL && root->right == NULL) {
            if (sum == targetSum)
                return true;


            else
                return false;
        }
```

```
  bool leftSide = dfs(root->left, sum, targetSum);
        bool rightSide = dfs(root->right, sum, targetSum);


        return leftSide || rightSide;
    }
    bool hasPathSum(TreeNode* root, int targetSum) {
        int sum = 0;
        return dfs(root, sum, targetSum);
    }
};
```

113. Path Sum II

```
class Solution {
public:
    vector<vector<int>> result;


    void dfs(TreeNode* root, int sum, vector<int> temp, int& targetSum) {
        if (root == NULL)
            return;


        sum += root->val;
        temp.push_back(root->val);


        // if we reached the leaf node
        if (root->left == NULL && root->right == NULL) {
            if (sum == targetSum)
                result.push_back(temp);
```

```
            else
                return;
        }


        dfs(root->left, sum, temp, targetSum);
        dfs(root->right, sum, temp, targetSum);
    }
    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        vector<int> temp;
        int sum = 0;
        dfs(root, sum, temp, targetSum);


        return result;
    }
};
```

129. Sum Root to Leaf Numbers

```
class Solution {
public:
    int dfs(TreeNode* root, int sum) {
        if (root == NULL)
            return 0;


        sum = sum * 10 + root->val;


        if (!root->left && !root->right) {
            return sum;
        }


        return dfs(root->left, sum) + dfs(root->right, sum);
    }
    int sumNumbers(TreeNode* root) { return dfs(root, 0); }
};
```

2236. Root Equals Sum of Children

```
class Solution {
public:
    bool checkTree(TreeNode* root) {
        return root->val == root->left->val + root->right->val;
    }
};
```

965. Univalued Binary Tree

```
class Solution {
public:
    bool isUnivalTree(TreeNode* root) {
        queue<TreeNode*> que;
        que.push(root);
        unordered_set<int> st;

        while (!que.empty()) {
            TreeNode* node = que.front();
            que.pop();

            st.insert(node->val);

            if (node->left)
                que.push(node->left);
            if (node->right)
                que.push(node->right);
        }

        return st.size() == 1;
    }
};
```

Learn coding

THANK YOU