Experiment-5(A)

Student Name: Ayush Pathania **UID:**22BCS16023
Branch: CSE **Section/Group:**NTPP_602-A
Semester:6 **Date of Performance:**17-02-25
Subject Name: Advanced Programming Lab-2 **Subject Code:**22CSH-359

1. **Title:**Trees (Maximum Depth of Binary Tree)
   https://leetcode.com/problems/maximum-depth-of-binary-tree/description/

2. **Objective:**To find the maximum depth of a binary tree, which is the number of nodes along the longest path from the root node down to the farthest leaf node.

3. **Algorithm:**

   - **Base Case**:

     If the root is `null`, return `0`. This represents that an empty tree or a leaf node's child node does not contribute to depth.

   - **Recursive Case**:

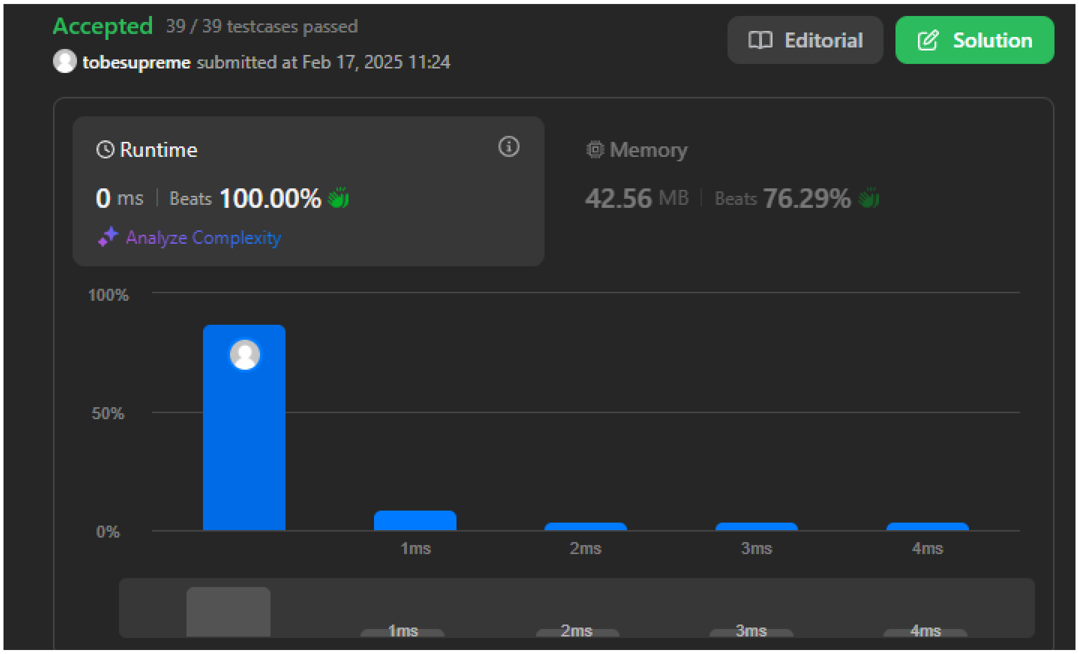
     If the root is not `null`, recursively find the depth of the left subtree and the depth of the right subtree.

     The depth of the current node is `1 + the maximum of the depths of the left and right subtrees`.

4. **Implementation/Code:**

```
class Solution {
   public int maxDepth(TreeNode root) {
      if (root == null) {
         return 0;
      }
      return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));
   }
}
```

5. **Output:**



6. **TimeComplexity:**O(N)

7. **SpaceComplexity:**O(H)

8. **LearningOutcomes:**

- **Recursion:**Understandinghowtosolvetreeproblemsusingrecursivefunctions.
- **TreeTraversal:**Gainingknowledgeofdepth-firstsearch(DFS)traversal.
- **DivideandConquer:**Applyingthedivideandconquerstrategytobreakdowntheproblem.
- **ComplexityAnalysis:**Learningthetimeandspacecomplexityofrecursivetreealgorithms.

Experiment5(B)

1. **Title:** SymmetricTree(https://leetcode.com/problems/symmetric-tree/description/

2. **Objective:** Todetermineifabinarytreeis**symmetric**arounditscenter,i.e.,whethertheleft subtree is a mirror reflection of the right subtree.

3. **Algorithm:**

Approach1:Recursive Solution

1. Ifthetreeisempty(rootisNone),itissymmetric.
2. Comparetheleftandrightsubtrees:
   - For a tree to be symmetric, the left subtree of the left child should be a mirror of the right subtree of the right child, and vice versa.
3. Recursivelycheckthesymmetryofthetwosubtrees:
   - Checkifthevaluesofthecurrentnodesareequal.
   - Recursivelycheck ifthe leftchildof theleftsubtreeissymmetricwith the right child of the right subtree, and the right child of the left subtree is symmetric with the left childof the right subtree.

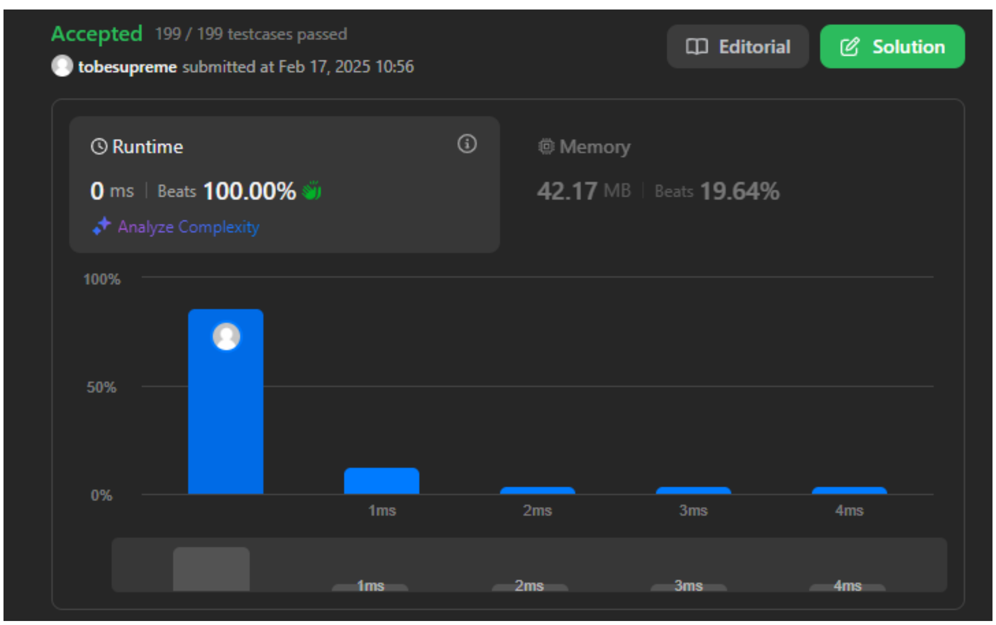
Approach2:IterativeSolution(UsingaQueue)

1. Initializeaqueueandstartbyaddingtheleftandrightchildrenoftheroot.
2. Whilethequeueisnotempty:
   - Dequeuetwonodesatatime.
   - Checkiftheirvaluesareequal.
   - Add the children in reverse order (left child of the left node, right child of the right node, and so on) to the queue.
3. Continueuntilallnodesarechecked.

4. Implementation/Code:

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) {
            return true;
        }
        return isMirror(root.left, root.right);
    }

    private boolean isMirror(TreeNode node1, TreeNode node2) {
        if (node1 == null && node2 == null) {
            return true;
        }
        if (node1 == null || node2 == null) {
            return false;
        }
        return node1.val == node2.val && isMirror(node1.left, node2.right) &&
isMirror(node1.right, node2.left);
    }
}
```

6. Output:



8.<u>TimeComplexity:</u>O(N)

9.<u>SpaceComplexity:</u>O(N)

Experiment5(C)

1. **Title:** Binary Tree Level Order Traversal (https://leetcode.com/problems/binary-tree-level-order-traversal/description/)

2. **Objective:** To perform a **Level Order Traversal** of a binary tree, where nodes are visited level by level from left to right.

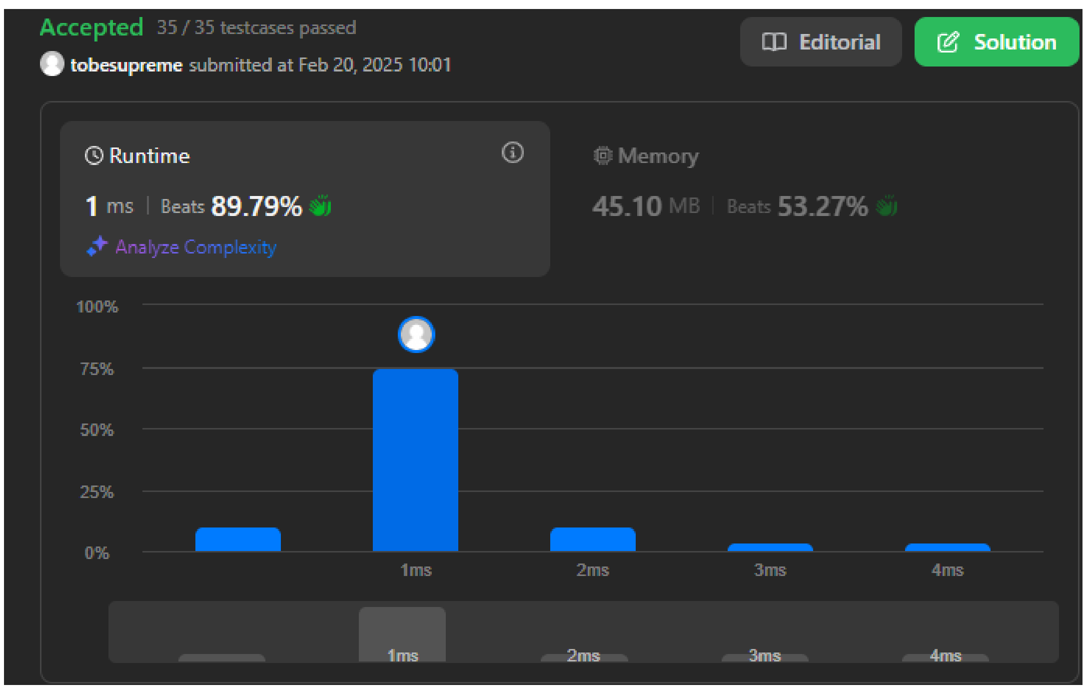
3. **Algorithm:**
   - **BaseCase:** IftherootisNone,returnanemptylist.
   - **InitializeQueue:**
     - Useaqueue(FIFO)tostorenodesforlevelordertraversal.
     - Startbyenqueuingtherootnode.
   - **Level-wiseTraversal:**
     - Foreachlevel:
     - Determinethenumberofnodesatthecurrentlevel.
     - Dequeueeachnode,collectitsvalue,andenqueueitschildren(leftandright).
     - Storethevaluesofthecurrentlevelinalist.
   - **ContinueuntilQueueisEmpty:**
     - Repeattheprocessforalllevels.
   - **ReturnResult:**
     - Returnthelistoflistscontainingnodevalueslevelbylevel.

5. **Implementation/Code:**

```
from class Solution {

   public List<List<Integer>> levelOrder(TreeNode root) {

      List<List<Integer>> ans = new ArrayList<>();

      if (root == null) return ans;

      Queue<TreeNode> queue = new LinkedList<>();

      queue.add(root);

      while (!queue.isEmpty()) {

         int levelSize = queue.size();

         List<Integer> level = new ArrayList<>();

         for (int i = 0; i < levelSize; ++i) {

            TreeNode node = queue.poll();

            level.add(node.val);

            if (node.left != null) queue.add(node.left);
```

```
                if (node.right != null) queue.add(node.right);
            }
            ans.add(level);
        }
        return ans;
    }
}
```

6. Output:



8. **TimeComplexity:** O(N)

9. **SpaceComplexity:** O(N)

10. LearningOutcomes:

- **Breadth-FirstSearch(BFS):** UsingaqueueforBFStraversal.
- **QueueDataStructure:** Efficientlymanagingnodeslevelbylevel.
- **EdgeCaseHandling:** Properlyhandlingemptytrees.
- **TimeandSpaceComplexityAnalysis:** Understandingcomplexitiesintreetraversal.