EXPERIMENT 9

STUDENT NAME: LAKSHIT MALHOTRA **UID:** 22BCS13047

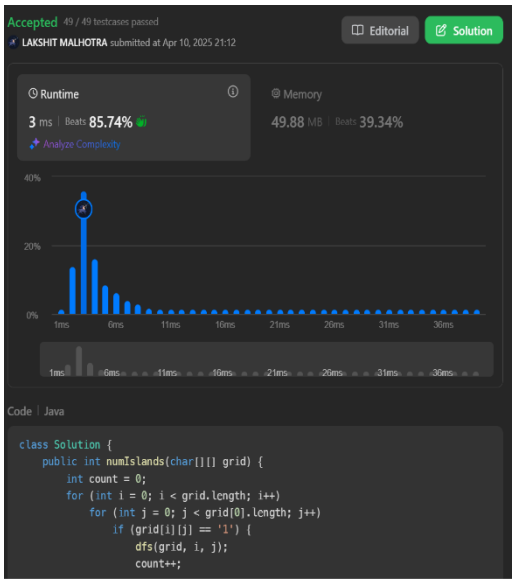
BRANCH: CSE **SECTION:** 22BCS_FL_IOT_601A

SEMESTER: 6 **DATE OF PERFORMANCE:** 9/4/25

SUBJECT NAME: AP LAB -2 **SUBJECT CODE:** 22CSP-351

200. NUMBER OF ISLANDS (MEDIUM)

https://leetcode.com/problems/number-of-islands/



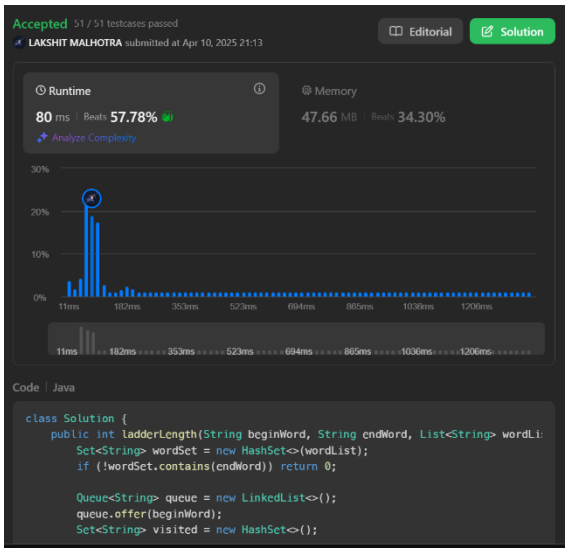

```
class Solution {
    public int numIslands(char[][] grid) {
        int count = 0;
        for (int i = 0; i < grid.length; i++)
            for (int j = 0; j < grid[0].length; j++)
                if (grid[i][j] == '1') {
                    dfs(grid, i, j);
                    count++;
                }
        return count;
    }
    void dfs(char[][] grid, int i, int j) {
        if (i < 0 || j < 0 || i >= grid.length || j >= grid[0].length || grid[i][j] == '0') return;
        grid[i][j] = '0';
        dfs(grid, i + 1, j);
        dfs(grid, i - 1, j);
        dfs(grid, i, j + 1);
        dfs(grid, i, j - 1);
    }
}
```

127. WORD LADDER (HARD)

https://leetcode.com/problems/word-ladder/

```
class Solution {
    public int ladderLength(String beginWord, String endWord, List<String> wordList) {
        Set<String> wordSet = new HashSet<>(wordList);
        if (!wordSet.contains(endWord)) return 0;
```

```
        Queue<String> queue = new LinkedList<>();
        queue.offer(beginWord);
        Set<String> visited = new HashSet<>();
        visited.add(beginWord);
        int level = 1;
        while (!queue.isEmpty()) {
            int size = queue.size();
            while (size-- > 0) {
                String word = queue.poll();
                if (word.equals(endWord)) return level;
                char[] chars = word.toCharArray();
                for (int i = 0; i < chars.length; i++) {
                    char original = chars[i];
                    for (char c = 'a'; c <= 'z'; c++) {
                        chars[i] = c;
                        String newWord = new String(chars);
                        if (wordSet.contains(newWord) && !visited.contains(newWord)) {
                            visited.add(newWord);
                            queue.offer(newWord);
                        }
                    }
                    chars[i] = original;
                }
            }
            level++;
        }
        return 0;
    }
}
```

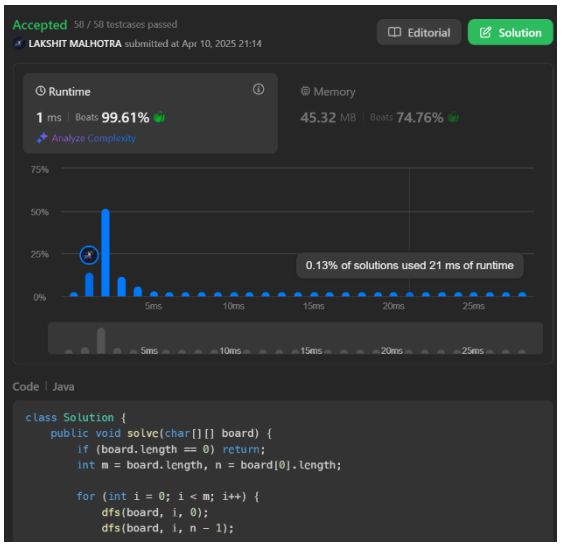


130. SURROUNDED REGIONS (MEDIUM)

🔗 https://leetcode.com/problems/surrounded-regions/

```
class Solution {
    public void solve(char[][] board) {
        if (board.length == 0) return;
        int m = board.length, n = board[0].length;
        for (int i = 0; i < m; i++) {
            dfs(board, i, 0);
            dfs(board, i, n - 1);
        }
```

```
        for (int j = 0; j < n; j++) {
            dfs(board, 0, j);
            dfs(board, m - 1, j);
        }
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                board[i][j] = board[i][j] == 'E' ? 'O' : 'X';
    }
    void dfs(char[][] board, int i, int j) {
        if (i < 0 || j < 0 || i >= board.length || j >=
board[0].length || board[i][j] != 'O') return;
        board[i][j] = 'E';
        dfs(board, i + 1, j);
        dfs(board, i - 1, j);
        dfs(board, i, j + 1);
        dfs(board, i, j - 1);
    }
}
```

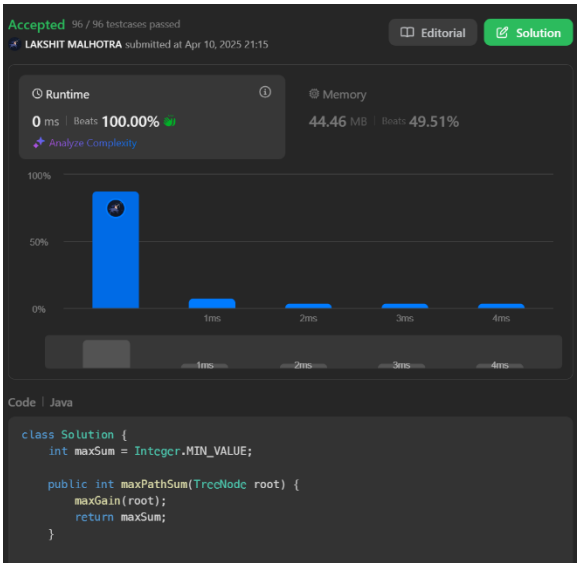


124. BINARY TREE MAXIMUM PATH SUM (HARD)

🔗 https://leetcode.com/problems/binary-tree-maximum-path-sum/

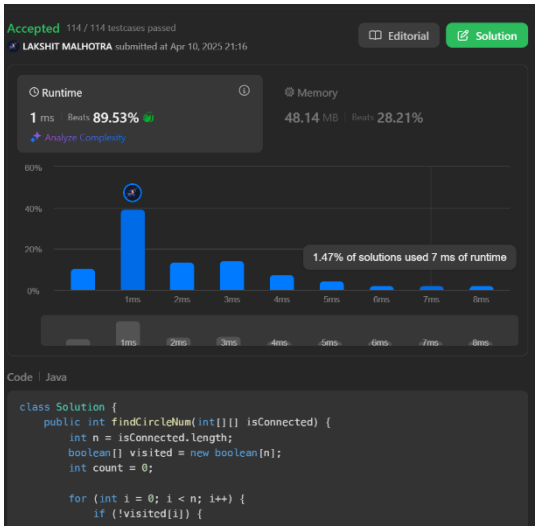
```
class Solution {
    int maxSum = Integer.MIN_VALUE;

    public int maxPathSum(TreeNode root) {
        maxGain(root);
        return maxSum;
    }
    private int maxGain(TreeNode node) {
        if (node == null) return 0;
        int leftGain = Math.max(maxGain(node.left),
0);
        int rightGain =
Math.max(maxGain(node.right), 0);
        int priceNewPath = node.val + leftGain +
rightGain;
        maxSum = Math.max(maxSum, priceNewPath);
        return node.val + Math.max(leftGain, rightGain);
    }
}
```



547. FRIEND CIRCLES / NUMBER OF PROVINCES (MEDIUM)

🔗 https://leetcode.com/problems/number-of-provinces/

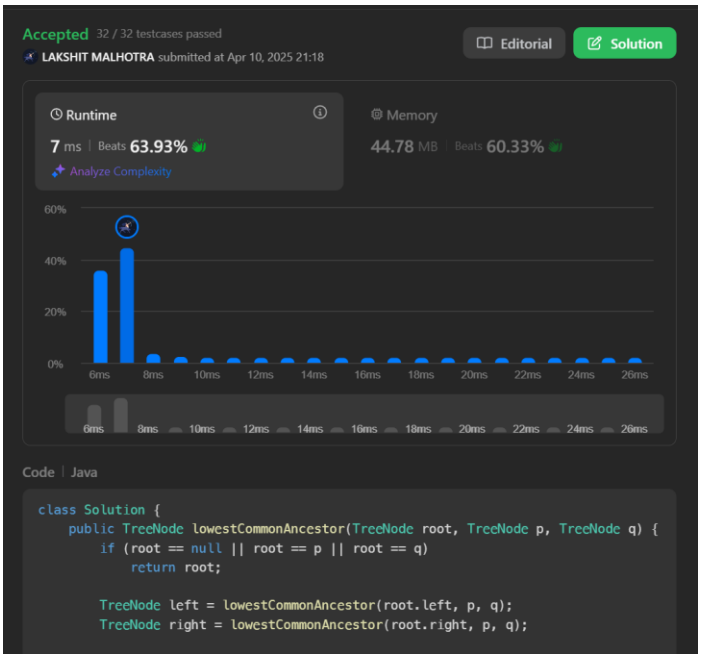



```
class Solution {
    public int findCircleNum(int[][] isConnected) {
        int n = isConnected.length;
        boolean[] visited = new boolean[n];
        int count = 0;
        for (int i = 0; i < n; i++) {
            if (!visited[i]) {
                dfs(isConnected, visited, i);
                count++;
            }
        }
        return count;
    }
    void dfs(int[][] isConnected, boolean[] visited, int i) {
        for (int j = 0; j < isConnected.length; j++) {
            if (isConnected[i][j] == 1 && !visited[j]) {
                visited[j] = true;
                dfs(isConnected, visited, j);
            }
        }
    }
}
```

236. LOWEST COMMON ANCESTOR OF A BINARY TREE (MEDIUM)

🔗 https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/

```
class Solution {
    public 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 && right != null)
            return root;
        return left != null ? left : right;
    }
}
```



207. COURSE SCHEDULE (MEDIUM)

🔗 https://leetcode.com/problems/course-schedule/

```
class Solution {
    public boolean canFinish(int numCourses, int[][] prerequisites) {
        List<List<Integer>> graph = new ArrayList<>();
        for (int i = 0; i < numCourses; i++)
            graph.add(new ArrayList<>());
        for (int[] pre : prerequisites)
            graph.get(pre[1]).add(pre[0]);
        boolean[] visited = new boolean[numCourses];
        boolean[] recStack = new boolean[numCourses];
        for (int i = 0; i < numCourses; i++)
            if (isCyclic(graph, visited, recStack, i))
                return false;
        return true;
    }
```

```
    private boolean isCyclic(List<List<Integer>> graph, boolean[] visited, boolean[]
recStack, int node) {
        if (recStack[node]) return true;
        if (visited[node]) return false;
        visited[node] = recStack[node] = true;
        for (int neighbor : graph.get(node))
            if (isCyclic(graph, visited, recStack, neighbor))
                return true;
        recStack[node] = false;
        return false;
    }
}
```