Assignment-9

Student Name: Anjali
UID: 22BCS10665
Branch: BE- CSE
Section/Group:22BCS_IOT-614/B
Semester: 6
Date of Performance:10-04-25
Subject Name: AP LAB-II
Subject Code: 22CSP-351

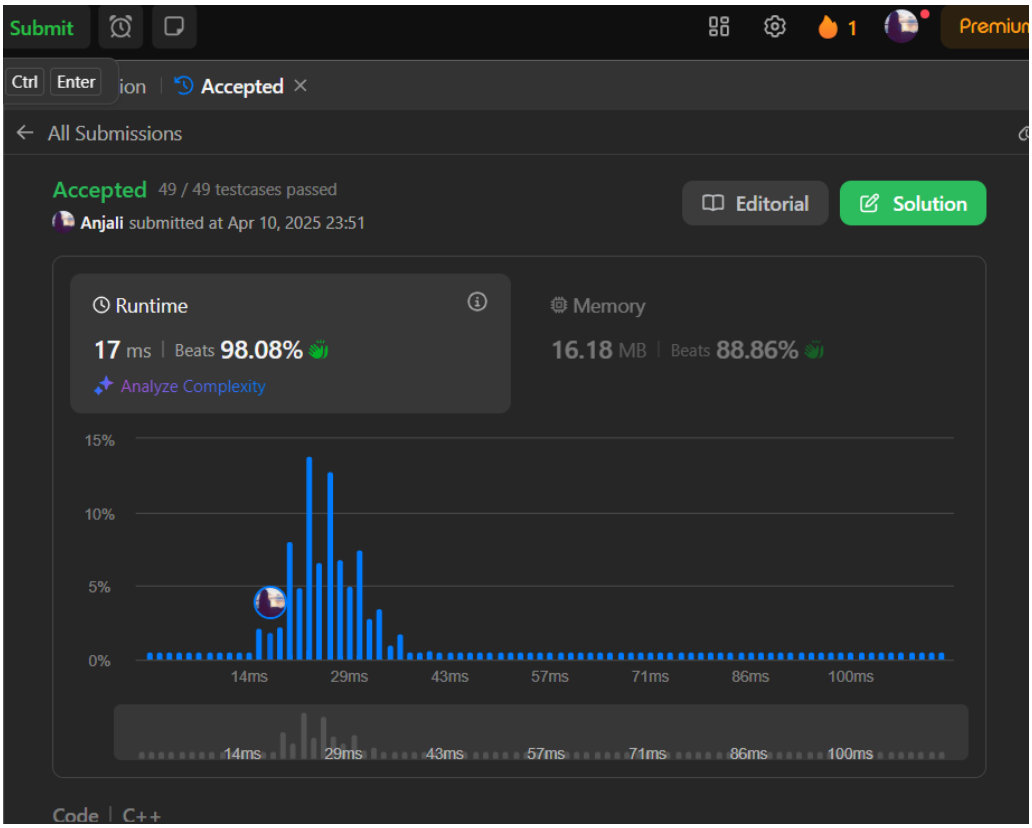
PROBLEM-1: Number of Islands

- **Objective:** Given an m x n 2D binary grid grid which represents a map of '1's (land) and '0's (water), return *the number of islands*.
  An island is surrounded by water and is formed by connecting adjacent lands horizontally or vertically. You may assume all four edges of the grid are all surrounded by water.

- **Implementation/Code:**

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

- **Output:**



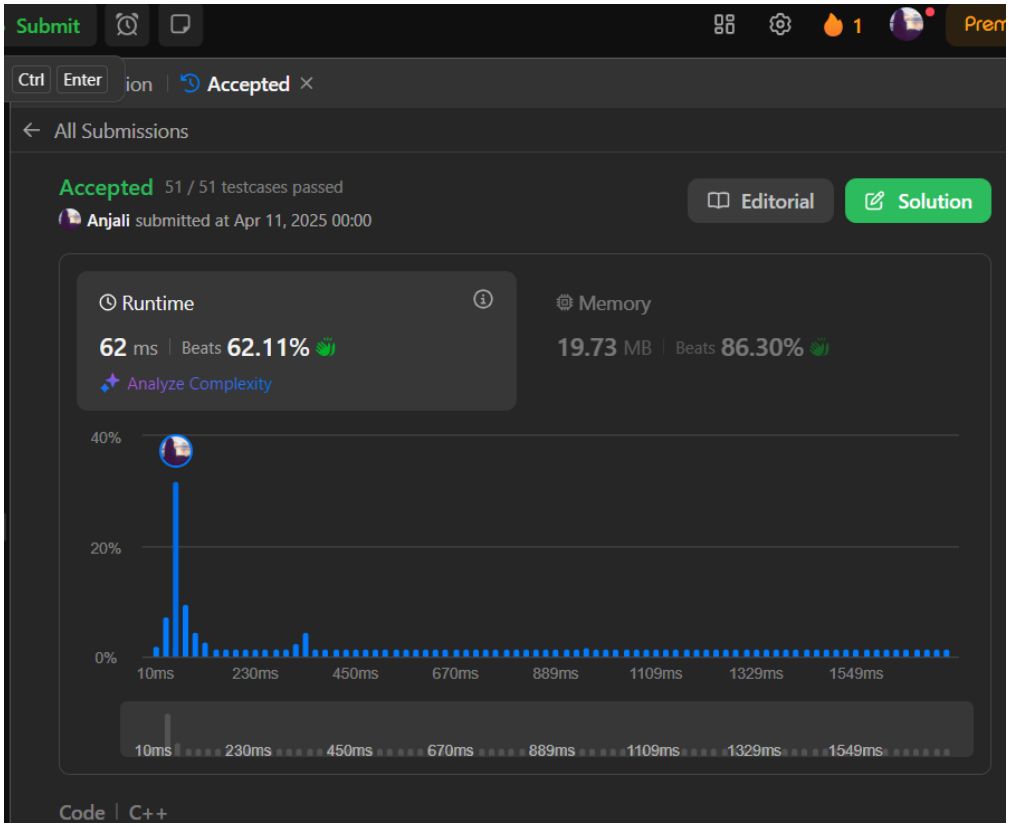
PROBLEM-2: Word Ladder

- **Objective:** A transformation sequence from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s_1 -> s_2 -> ... -> s_k such that:
  Every adjacent pair of words differs by a single letter.
  Every s_i for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.
  s_k == endWord
  Given two words, beginWord and endWord, and a dictionary wordList, return *the number of words in the shortest transformation sequence from* beginWord *to* endWord, *or* 0 *if no such sequence exists.*

- **Implementation/Code:**

```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
        unordered_set<string> dict(wordList.begin(), wordList.end());
        if (!dict.count(endWord)) return 0;

        queue<string> q;
        q.push(beginWord);
        int level = 1;

        while (!q.empty()) {
            int size = q.size();
            while (size--) {
                string word = q.front();
                q.pop();

                for (int i = 0; i < word.size(); ++i) {
                    char original = word[i];
                    for (char c = 'a'; c <= 'z'; ++c) {
                        word[i] = c;
                        if (word == endWord) return level + 1;
                        if (dict.count(word)) {
                            q.push(word);
                            dict.erase(word);
                        }
                    }
                    word[i] = original;
                }
            }
            level++;
        }

        return 0;
    }
     };
```

- **Output:**



PROBLEM-3: Surrounded Regions

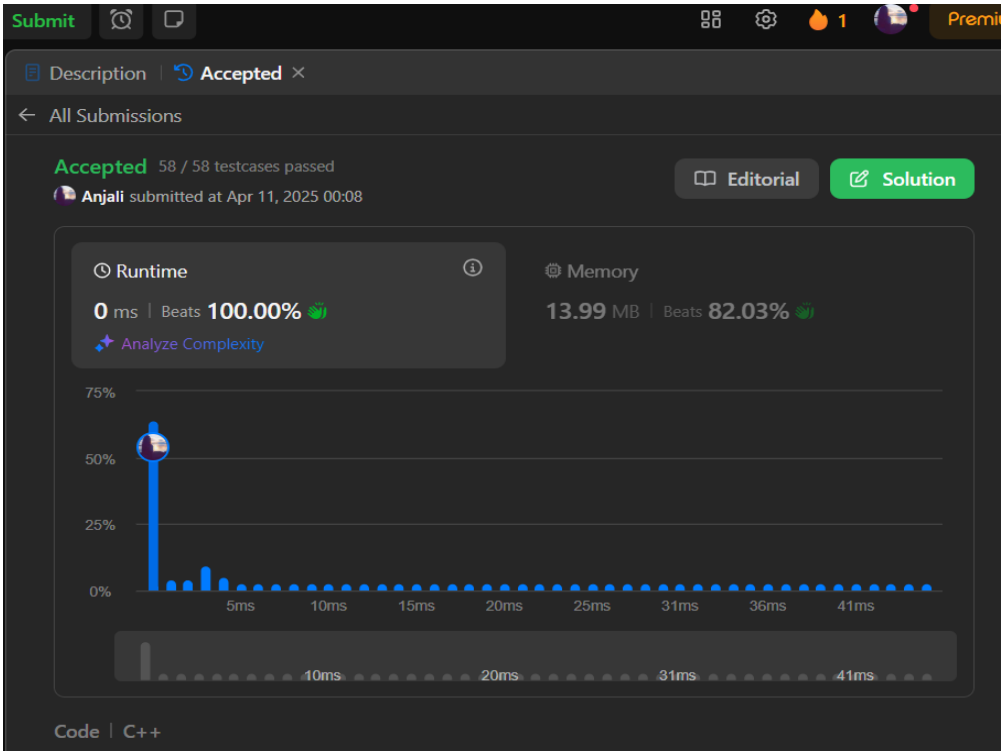
- **Objective:** You are given an m x n matrix board containing letters 'X' and 'O', capture regions that are surrounded:
  Connect: A cell is connected to adjacent cells horizontally or vertically.
  Region: To form a region connect every 'O' cell.
  Surround: The region is surrounded with 'X' cells if you can connect the region with 'X' cells and none of the region cells are on the edge of the board.
  To capture a surrounded region, replace all 'O's with 'X's in-place within the original board. You do not need to return anything.

- **Implementation/Code:**

```
class Solution {
public:
    void dfs(vector<vector<char>>& board, int i, int j) {
        int m = board.size(), n = board[0].size();
        if (i < 0 || i >= m || j < 0 || j >= n || board[i][j] != 'O') return;

        board[i][j] = '#';
        dfs(board, i + 1, j);
```

```
        dfs(board, i - 1, j);
        dfs(board, i, j + 1);
        dfs(board, i, j - 1);
    }

    void solve(vector<vector<char>>& board) {
        int m = board.size(), n = board[0].size();

        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) {
                if (board[i][j] == 'O') board[i][j] = 'X';
                else if (board[i][j] == '#') board[i][j] = 'O';
            }
        }
    }
};
```

- **Output:**



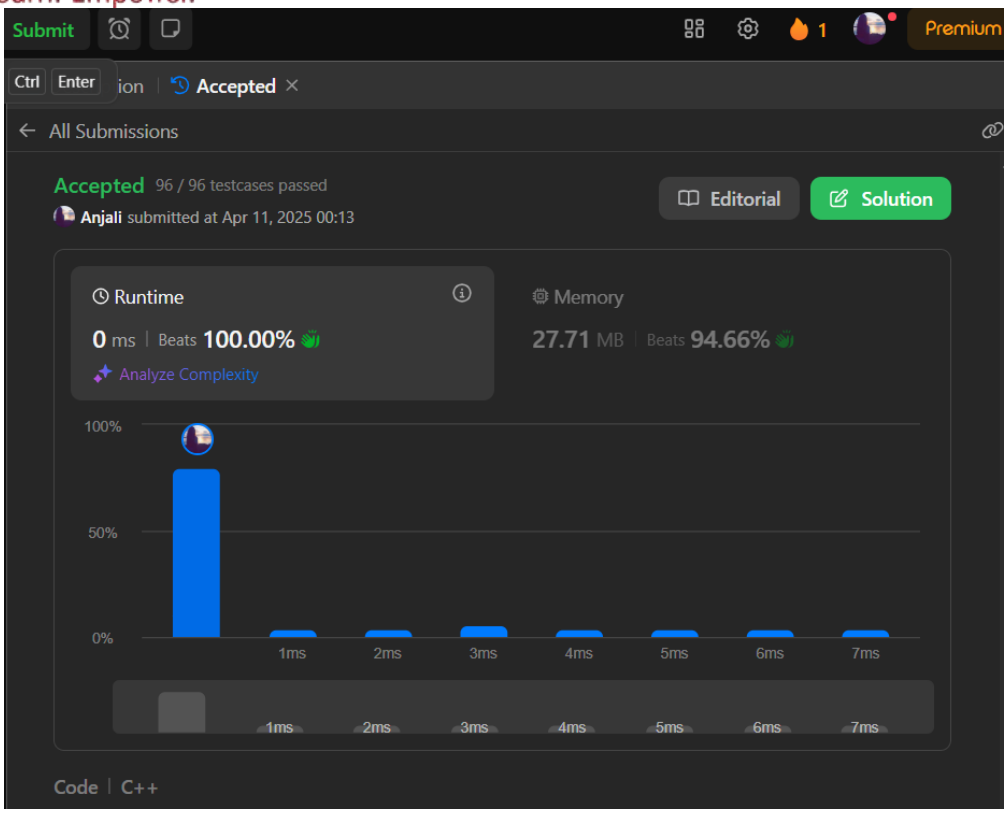
PROBLEM-4: Binary Tree Maximum Path Sum

- **Objective:** 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*.

- **Implementation/Code:**

```
class Solution {
public:
    int maxSum = INT_MIN;

    int maxPathSumHelper(TreeNode* root) {
        if (!root) return 0;
        int leftMax = max(0, maxPathSumHelper(root->left));
        int rightMax = max(0, maxPathSumHelper(root->right));
        maxSum = max(maxSum, root->val + leftMax + rightMax);

        return root->val + max(leftMax, rightMax);
    }

    int maxPathSum(TreeNode* root) {
        maxPathSumHelper(root);
        return maxSum;
    }
     };
```

- **Output:**


Submit
1
Premium
Ctrl
Enter
ion
Accepted
All Submissions
Accepted
96 / 96 testcases passed
Anjali submitted at Apr 11, 2025 00:13
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
27.71 MB | Beats 94.66%
100%
50%
0%
1ms
2ms
3ms
4ms
5ms
6ms
7ms
Code | C++