**Assignment 9**

**Name – Aditya Taragi Uid – 22BCS10422.**

**Problem 1: Number of Islands (**<https://leetcode.com/problems/number-of-islands/> **)**

**Code:**class Solution {

public:

    void dfs(vector<vector<char>>& grid, int i, int j) {

        if (i < 0 || j < 0 || i >= grid.size() || j >= grid[0].size() || 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 count = 0;

        for (int i = 0; i < grid.size(); ++i) {

            for (int j = 0; j < grid[0].size(); ++j) {

                if (grid[i][j] == '1') {

                    ++count;

                    dfs(grid, i, j);

                }

            }

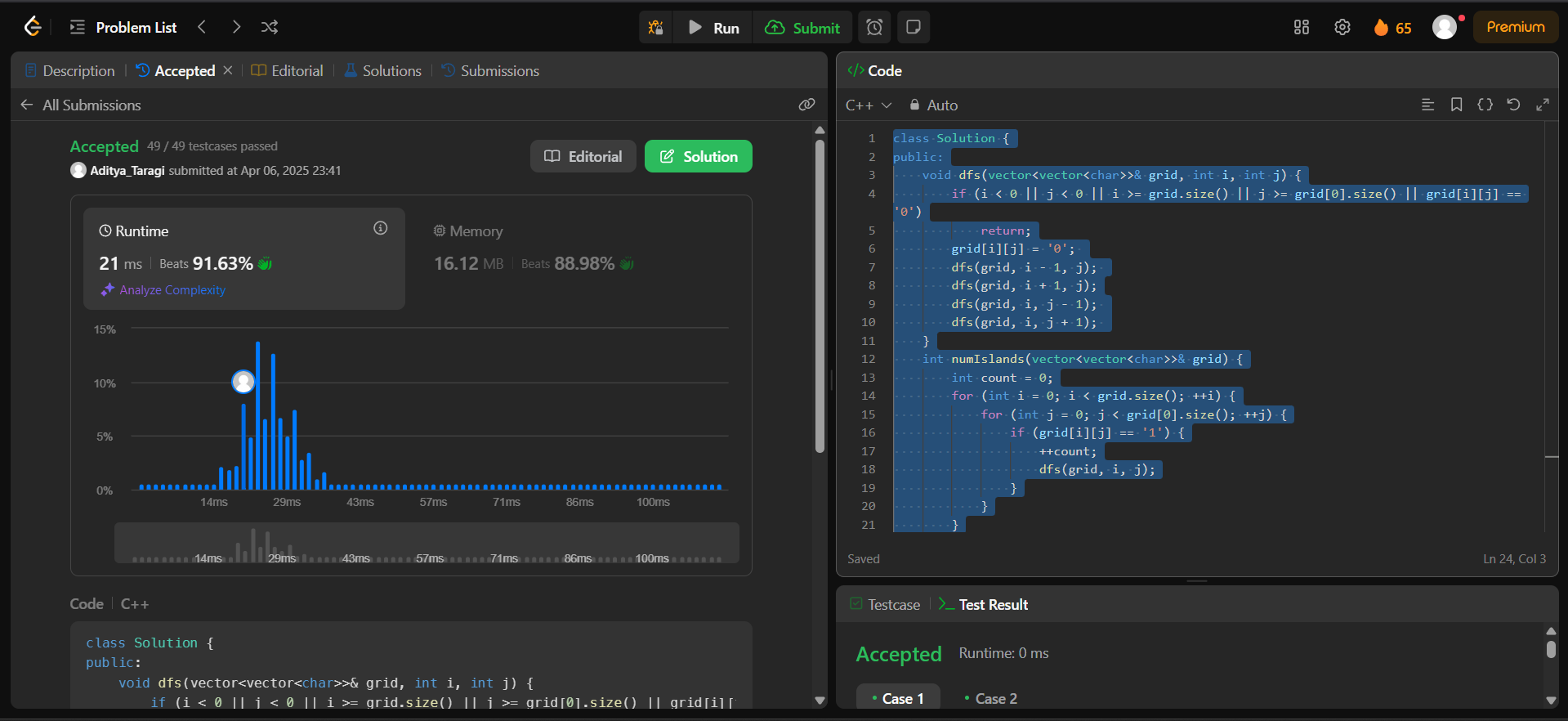
        }

        return count;

    }

};

**Screenshot:**



**Problem 2: Surrounded Regions (**<https://leetcode.com/problems/surrounded-regions/> **)**

**Code:**class Solution {

public:

    void dfs(vector<vector<char>>& board, int i, int j) {

        if (i < 0 || j < 0 || i >= board.size() || j >= board[0].size() || board[i][j] != 'O') {

            return;

        }

        board[i][j] = 'T';

        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) {

        if (board.empty()) return;

        int rows = board.size(), cols = board[0].size();

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

            if (board[i][0] == 'O') dfs(board, i, 0);

            if (board[i][cols - 1] == 'O') dfs(board, i, cols - 1);

        }

        for (int j = 0; j < cols; ++j) {

            if (board[0][j] == 'O') dfs(board, 0, j);

            if (board[rows - 1][j] == 'O') dfs(board, rows - 1, j);

        }

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

            for (int j = 0; j < cols; ++j) {

                if (board[i][j] == 'O') {

                    board[i][j] = 'X';

                } else if (board[i][j] == 'T') {

                    board[i][j] = 'O';

                }

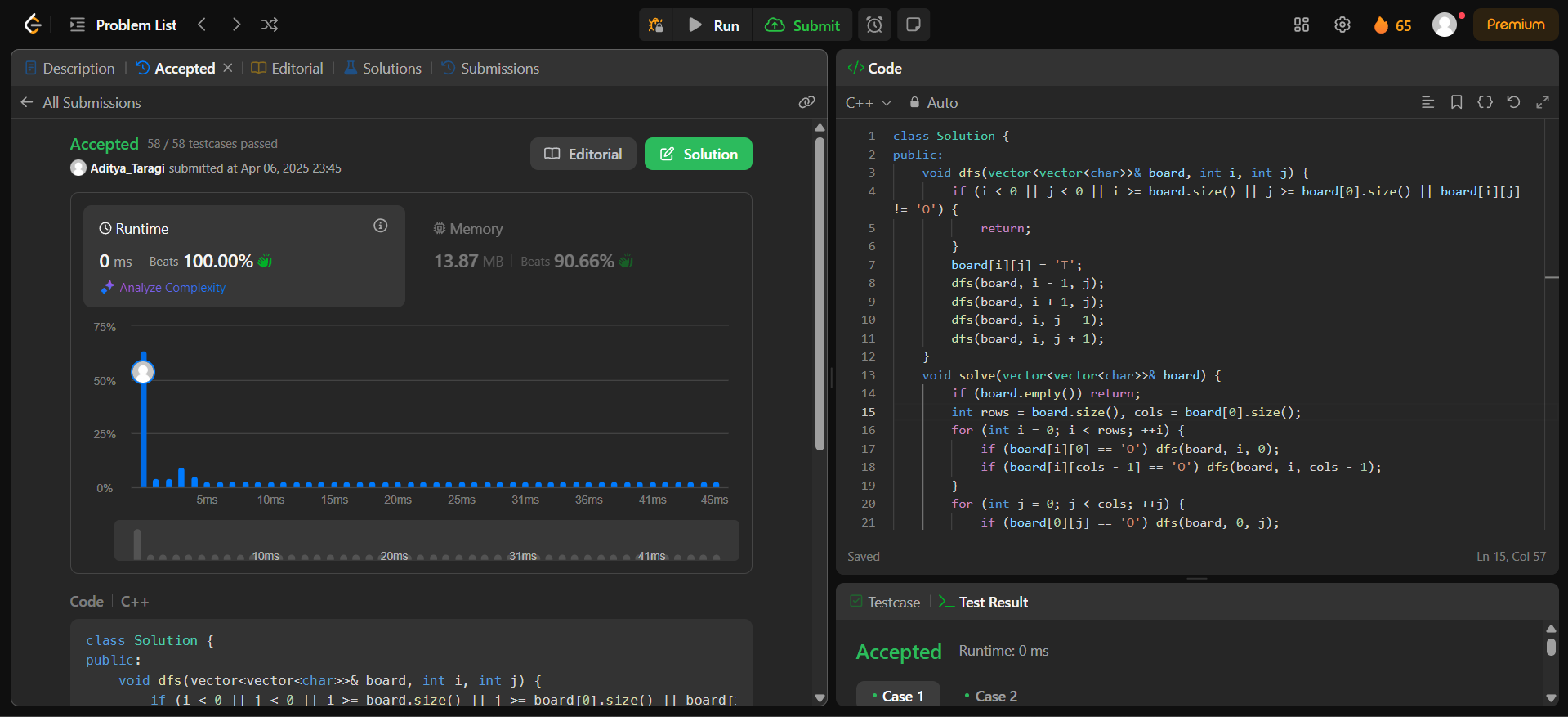
            }

        }

    }

};

**Screenshot:**



**Problem 3: Number of Provinces (**<https://leetcode.com/problems/number-of-provinces/> **)**

**Code:**class Solution {

public:

    void dfs(vector<vector<int>>& isConnected, vector<bool>& visited, int city) {

        visited[city] = true;

        for (int i = 0; i < isConnected.size(); ++i) {

            if (isConnected[city][i] == 1 && !visited[i]) {

                dfs(isConnected, visited, i);

            }

        }

    }

    int findCircleNum(vector<vector<int>>& isConnected) {

        int n = isConnected.size();

        vector<bool> visited(n, false);

        int provinces = 0;

        for (int i = 0; i < n; ++i) {

            if (!visited[i]) {

                dfs(isConnected, visited, i);

                ++provinces;

            }

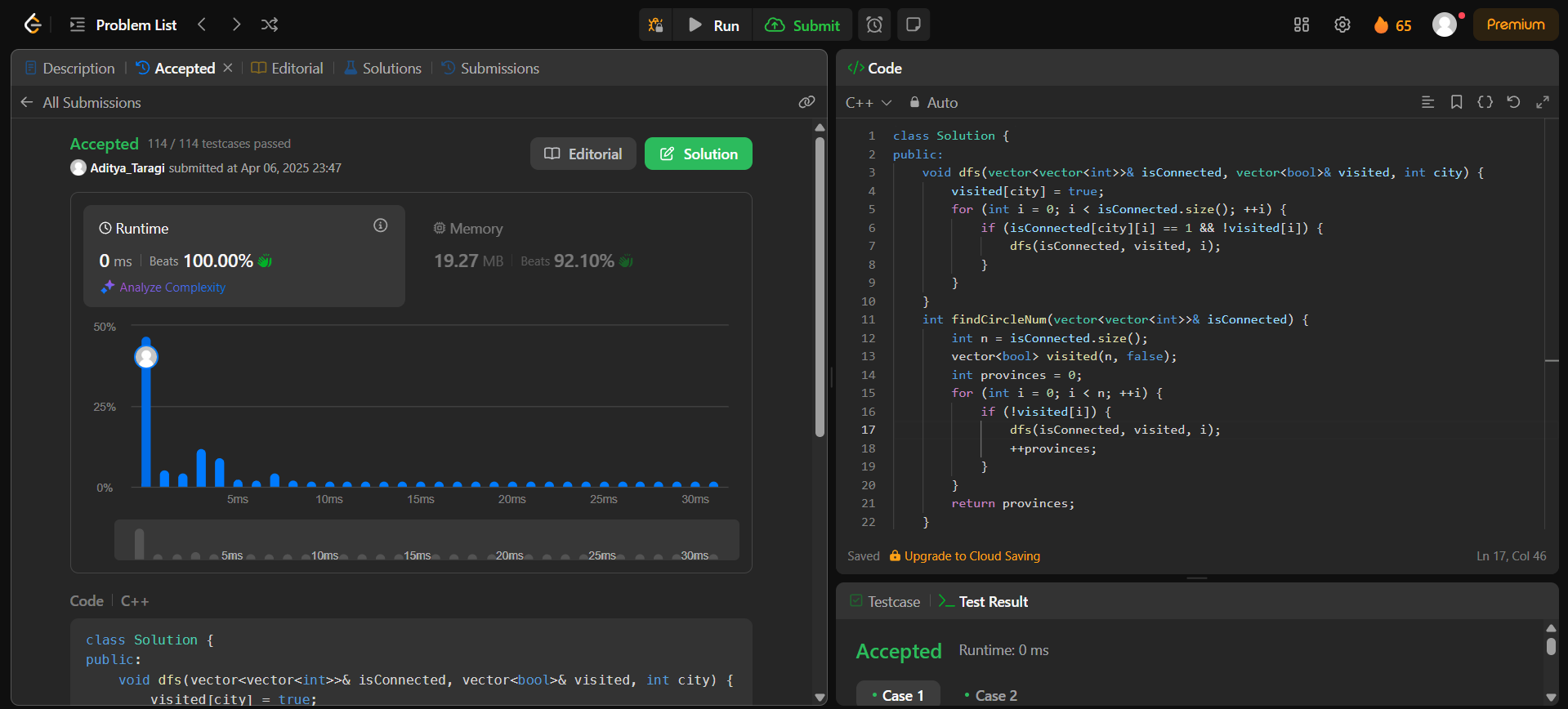
        }

        return provinces;

    }

};

**Screenshot:**



**Problem 4: Maximum Score from Removing Substrings (**<https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/> **)**

**Code:**class Solution {

public:

    TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

        if (!root || root == p || root == q) {

            return root;

        }

        TreeNode\* left = lowestCommonAncestor(root->left, p, q);

        TreeNode\* right = lowestCommonAncestor(root->right, p, q);

        if (left && right) {

            return root;

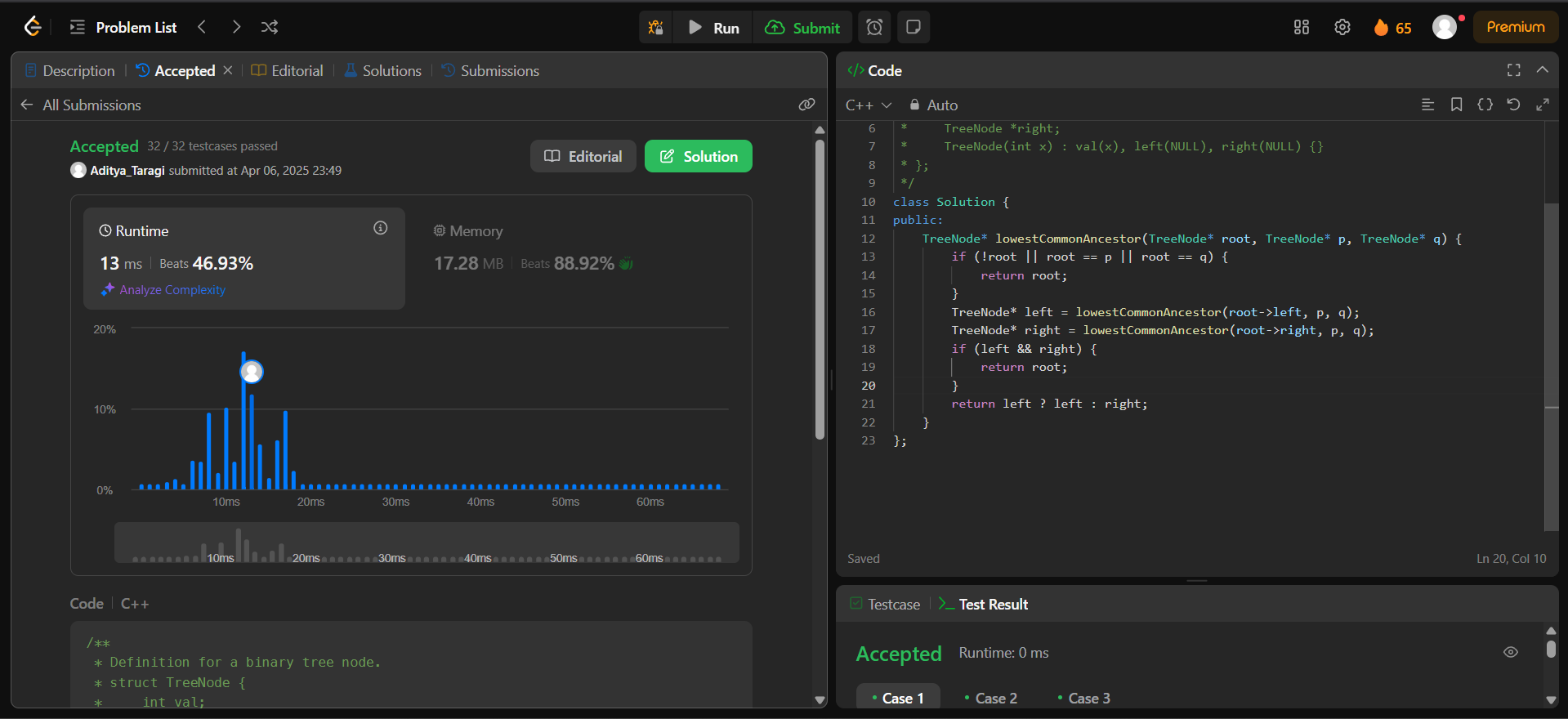
        }

        return left ? left : right;

    }

};

**Screenshot:**



**Problem 5: Word Ladder (**<https://leetcode.com/problems/word-ladder/> **)**

**Code:**class Solution {

public:

    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {

        unordered\_set<string> wordSet(wordList.begin(), wordList.end());

        if (wordSet.find(endWord) == wordSet.end()) {

            return 0;

        }

        queue<pair<string, int>> q;

        q.push({beginWord, 1});

        while (!q.empty()) {

            string word = q.front().first;

            int steps = q.front().second;

            q.pop();

            for (int i = 0; i < word.size(); ++i) {

                string temp = word;

                for (char c = 'a'; c <= 'z'; ++c) {

                    temp[i] = c;

                    if (temp == word) continue;

                    if (temp == endWord) return steps + 1;

                    if (wordSet.find(temp) != wordSet.end()) {

                        q.push({temp, steps + 1});

                        wordSet.erase(temp);

                    }

                }

            }

        }

        return 0;

    }

};

**Screenshot:**

