# NAME- Akshit Boparai| UID- 22BCS14939 | SECTION- 601/A

# Number of Islands

class Solution {

public:

    int numIslands(vector<vector<char>>& grid) {

        int islands = 0;

        int rows = grid.size();

        int cols = grid[0].size();

        unordered\_set<string> visited;

        vector<pair<int, int>> directions = {{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

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

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

                if (grid[r][c] == '1' && visited.find(to\_string(r) + "," + to\_string(c)) == visited.end()) {

                    islands++;

                    bfs(grid, r, c, visited, directions, rows, cols);

                }

            }

        }

        return islands;

    }

private:

    void bfs(vector<vector<char>>& grid, int r, int c, unordered\_set<string>& visited, vector<pair<int, int>>& directions, int rows, int cols) {

        queue<pair<int, int>> q;

        visited.insert(to\_string(r) + "," + to\_string(c));

        q.push({r, c});

        while (!q.empty()) {

            auto [row, col] = q.front();

            q.pop();

            for (auto [dr, dc] : directions) {

                int nr = row + dr;

                int nc = col + dc;

                if (nr >= 0 && nr < rows && nc >= 0 && nc < cols && grid[nr][nc] == '1' && visited.find(to\_string(nr) + "," + to\_string(nc)) == visited.end()) {

                    q.push({nr, nc});

                    visited.insert(to\_string(nr) + "," + to\_string(nc));

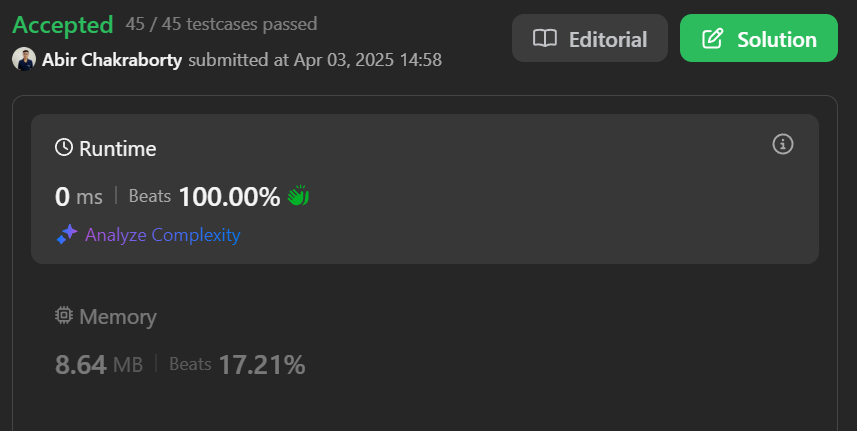
                }

            }

        }

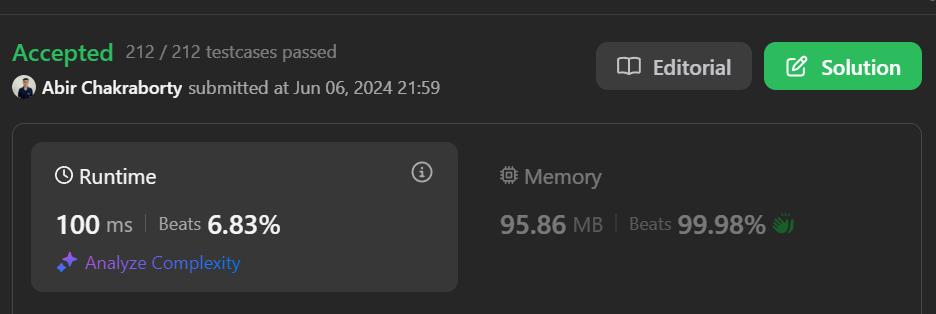
    }

};



# Word Ladder

class Solution {public: int ladderLength(string beginWord, string endWord, vector<string>& wordList) { unordered\_set<string> dict(wordList.begin(), wordList.end()); queue<string> todo; todo.push(beginWord); int ladder = 1; while (!todo.empty()) { int n = todo.size(); for (int i = 0; i < n; i++) { string word = todo.front(); todo.pop(); if (word == endWord) { return ladder; } dict.erase(word); for (int j = 0; j < word.size(); j++) { char c = word[j]; for (int k = 0; k < 26; k++) { word[j] = 'a' + k; if (dict.find(word) != dict.end()) { todo.push(word); } } word[j] = c; } } ladder++; } return 0; }};



# Surrounded Regions

class Solution {

private:

    void dfs(int row, int col, vector<vector<int>> &vis, vector<vector<char>> &board, int delrow[], int delcol[]) {

        vis[row][col] = 1;

        int n = board.size();

        int m = board[0].size();

        for (int i = 0; i < 4; i++) {

            int nrow = row + delrow[i];

            int ncol = col + delcol[i];

            if (nrow >= 0 && nrow < n && ncol >= 0 && ncol < m && !vis[nrow][ncol] && board[nrow][ncol] == 'O') {

                dfs(nrow, ncol, vis, board, delrow, delcol);

            }

        }

    }

public:

    void solve(vector<vector<char>>& board) {

        int n = board.size();

        if (n == 0) return;

        int m = board[0].size();

        int delrow[] = {-1, 0, 1, 0};

        int delcol[] = {0, 1, 0, -1};

        vector<vector<int>> vis(n, vector<int>(m, 0));

        for (int j = 0; j < m; j++) {

            if (!vis[0][j] && board[0][j] == 'O') dfs(0, j, vis, board, delrow, delcol);

            if (!vis[n - 1][j] && board[n - 1][j] == 'O') dfs(n - 1, j, vis, board, delrow, delcol);

        }

        for (int i = 0; i < n; i++) {

            if (!vis[i][0] && board[i][0] == 'O') dfs(i, 0, vis, board, delrow, delcol);

            if (!vis[i][m - 1] && board[i][m - 1] == 'O') dfs(i, m - 1, vis, board, delrow, delcol);

        }

        for (int i = 0; i < n; i++) {

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

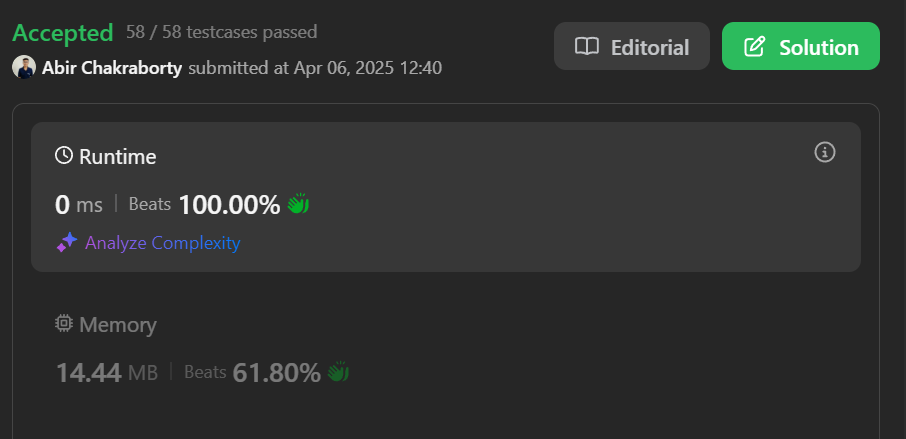
                if (!vis[i][j] && board[i][j] == 'O') board[i][j] = 'X';

            }

        }

    }

};



# Binary Tree Maximum Path Sum

class Solution {public: int findMaxPathSum(TreeNode\* root, int &maxi) { if (root == nullptr) { return 0; } int leftMaxPath = max(0, findMaxPathSum(root->left, maxi)); int rightMaxPath = max(0, findMaxPathSum(root->right, maxi)); maxi = max(maxi, leftMaxPath + rightMaxPath + root->val); return max(leftMaxPath, rightMaxPath) + root->val; } int maxPathSum(TreeNode\* root) { int maxi = INT\_MIN; findMaxPathSum(root, maxi); return maxi; }};

