# NAME- Abir Chakraborty| UID- 22BCS14321 | SECTION- 601/A 1 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));

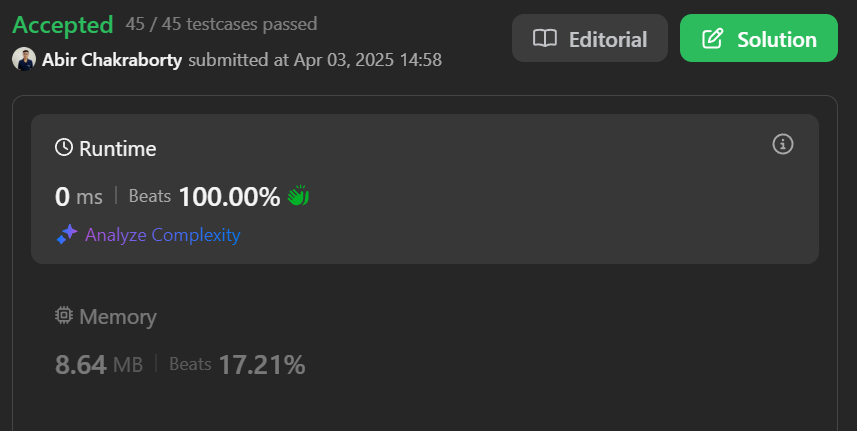
                }

            }

        }

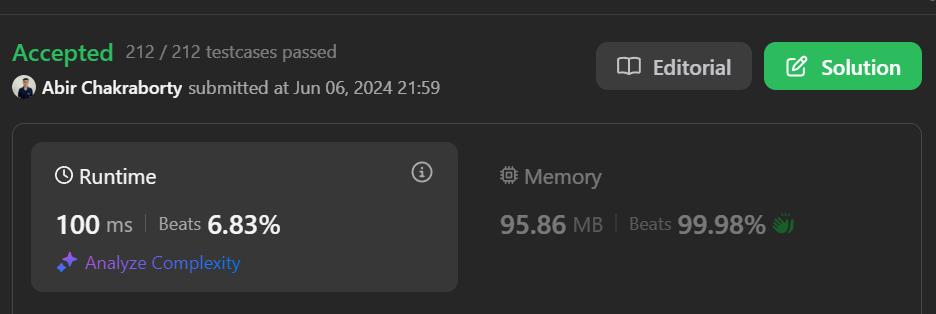
    }

};



# 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++) {

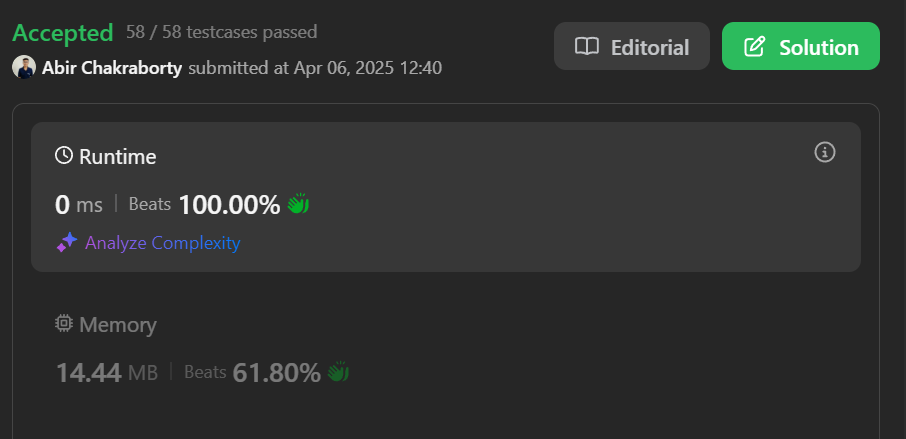
                if (!vis[i][j] && board[i][j] == 'O') board[i][j] = 'X';

            }

        }

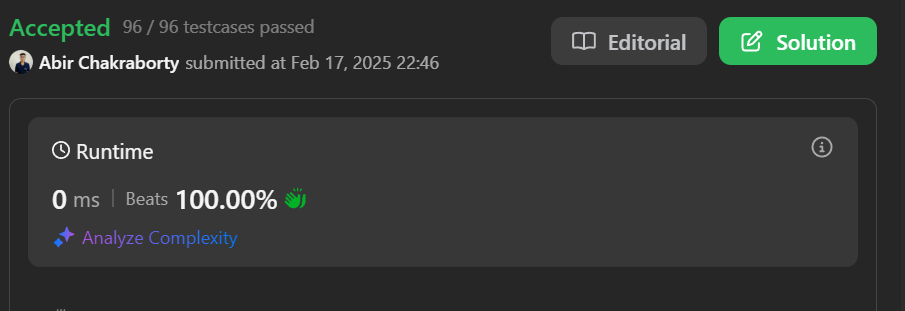
    }

};



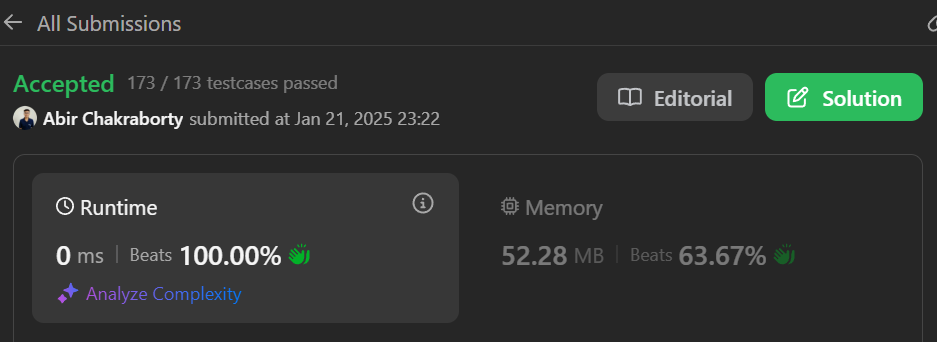
# Binary Tree Maximum Path Sum

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; }};



# Jump Game Friend Circles

# class Solution {private: void dfs(int node, vector<vector<int>>& adj, vector<int>& vis) { vis[node] = 1; for (int i = 0; i < adj[node].size(); i++) { if (adj[node][i] == 1 && !vis[i]) { dfs(i, adj, vis); } } }public: int findCircleNum(vector<vector<int>>& isConnected) { int v = isConnected.size(); vector<int> vis(v, 0); int cnt = 0; for (int i = 0; i < v; i++) { if (!vis[i]) { cnt++; dfs(i, isConnected, vis); } } return cnt; }};



## 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) {

## return right;

## }

## else if(right == NULL) {

## return left;

## }

## else {

## return root;

## }

## }

## };

## 

## Course Schedule

## class Solution {

## public:

## bool canFinish(int n, vector<vector<int>>& prerequisites) {

## vector<int> adj[n];

## vector<int> indegree(n, 0);

## vector<int> ans;

## for(auto x: prerequisites){

## adj[x[0]].push\_back(x[1]);

## indegree[x[1]]++;

## }

## queue<int> q;

## for(int i = 0; i < n; i++){

## if(indegree[i] == 0){

## q.push(i);

## }

## }

## while(!q.empty()){

## auto t = q.front();

## ans.push\_back(t);

## q.pop();

## for(auto x: adj[t]){

## indegree[x]--;

## if(indegree[x] == 0){

## q.push(x);

## }

## }

## }

## return ans.size() == n;

## }

## };

|  |  |
| --- | --- |
|  |  |
|  |  |

## Longest Increasing Path in a Matrix

class Solution {

public:

int m, n;

int memo[201][201];

int longestIncreasingPath(vector<vector<int>>& matrix) {

m = matrix.size();

n = matrix[0].size();

int ans = 0;

for (int r = 0; r < m; ++r)

for (int c = 0; c < n; ++c)

ans = max(ans, dfs(matrix, r, c));

return ans;

}

int DIR[5] = {0, 1, 0, -1, 0};

int dfs(vector<vector<int>>& matrix, int r, int c) {

if (memo[r][c] != 0) return memo[r][c];

int ans = 1;

for (int i = 0; i < 4; ++i) {

int nr = r + DIR[i], nc = c + DIR[i+1];

if (nr < 0 || nr == m || nc < 0 || nc == n || matrix[nr][nc] <= matrix[r][c]) continue;

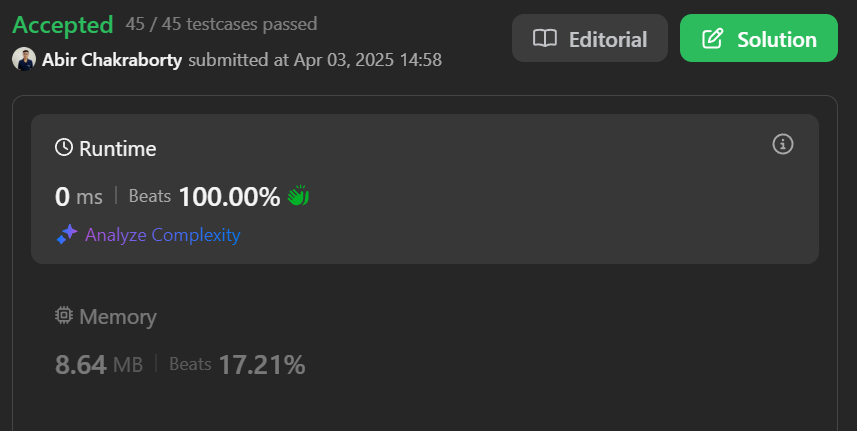
ans = max(ans, dfs(matrix, nr, nc) + 1);

}

return memo[r][c] = ans;

}

};



1. **Course Schedule II**

class Solution {public: vector<int> findOrder(int numCourses, vector<vector<int>>& prerequisites) { vector<vector<int>> adj(numCourses); vector<int> indegree(numCourses, 0); vector<int> ans; for (auto& pre : prerequisites) { int a = pre[0]; int b = pre[1]; adj[b].push\_back(a); indegree[a]++; } queue<int> q; for (int i = 0; i < numCourses; i++) { if (indegree[i] == 0) q.push(i); } while (!q.empty()) { int node = q.front(); q.pop(); ans.push\_back(node); for (int neighbor : adj[node]) { indegree[neighbor]--; if (indegree[neighbor] == 0) q.push(neighbor); } } if(ans.size() != numCourses) return {}; return ans; s }};

