Assignment 9 (Advance Programming)

Name – Paras Aggarwal

UID – 22BCS16963

200. Number of Islands

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.

Example 1:

Input: grid = [

["1","1","1","1","0"],

["1","1","0","1","0"],

["1","1","0","0","0"],

["0","0","0","0","0"]

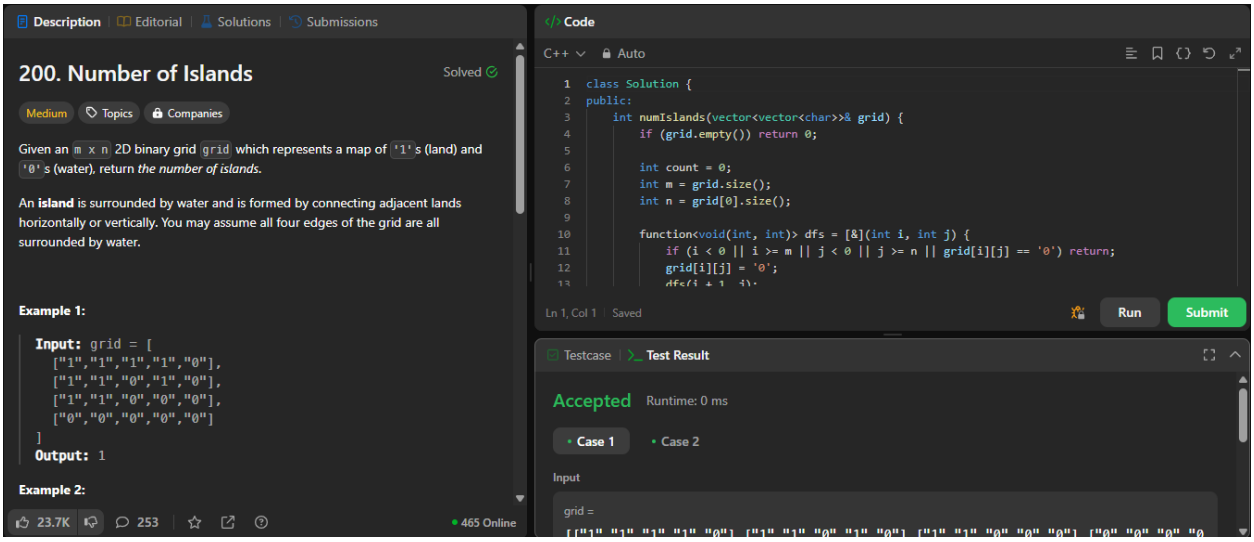
]

Output: 1

Solution:

```
class Solution {

public:

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

        if (grid.empty()) return 0;


        int count = 0;

        int m = grid.size()
```

```
        int n = grid[0].size();
        function<void(int, int)> dfs = [&](int i, int j) {
            if (i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == '0') return;
            grid[i][j] = '0';
            dfs(i + 1, j);
            dfs(i - 1, j);
            dfs(i, j + 1);
            dfs(i, j - 1);
        };
        for (int i = 0; i < m; ++i) {
            for (int j = 0; j < n; ++j) {
                if (grid[i][j] == '1') {
                    ++count;
                    dfs(i, j);
                }
            }
        }
        return count;
    }
};
```



127. Word Ladder

A transformation sequence from word beginWord to word endWord using a dictionary wordList is a sequence of words beginWord -> s1 -> s2 -> ... -> sk such that:

Every adjacent pair of words differs by a single letter.

Every si for 1 <= i <= k is in wordList. Note that beginWord does not need to be in wordList.

sk == 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.

Example 1:

Input: beginWord = "hit", endWord = "cog", wordList = ["hot","dot","dog","lot","log","cog"]

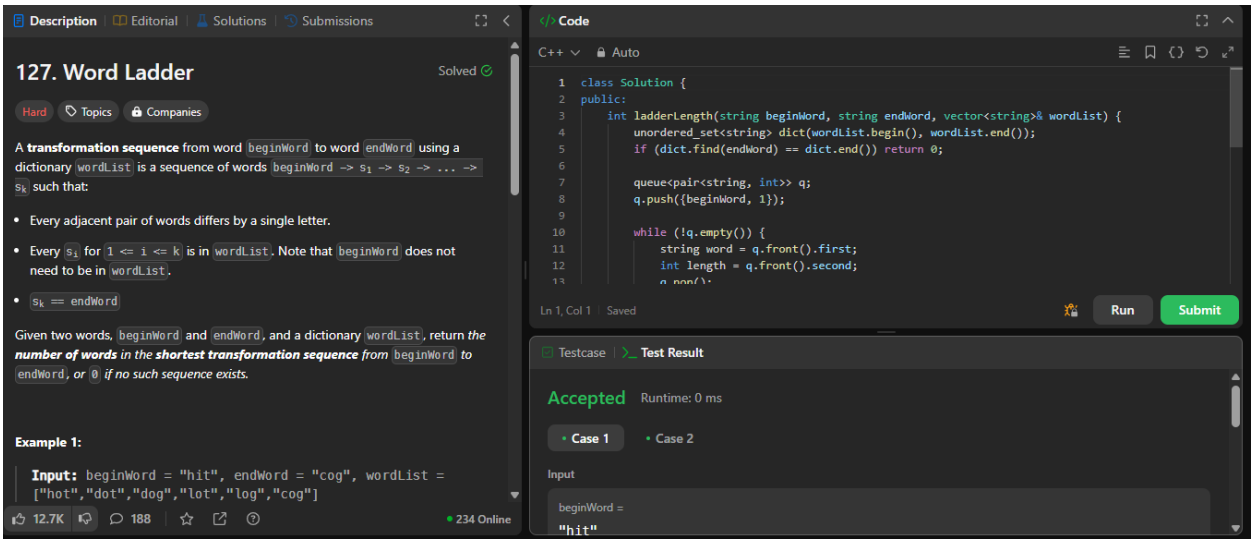
Output: 5

Explanation: One shortest transformation sequence is "hit" -> "hot" -> "dot" -> "dog" -> cog", which is 5 words long.

Solution:

```
class Solution {
public:
    int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
```

```
        unordered_set<string> dict(wordList.begin(), wordList.end());
        if (dict.find(endWord) == dict.end()) return 0;

        queue<pair<string, int>> q;
        q.push({beginWord, 1});

        while (!q.empty()) {
            string word = q.front().first;
            int length = 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 == endWord) return length + 1;
                    if (dict.find(temp) != dict.end()) {
                        q.push({temp, length + 1});
                        dict.erase(temp);
                    }
                }
            }
        }

        return 0;
    }
};
```



130. Surrounded Regions

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.

Example 1:

Input: board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]

Output: [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]

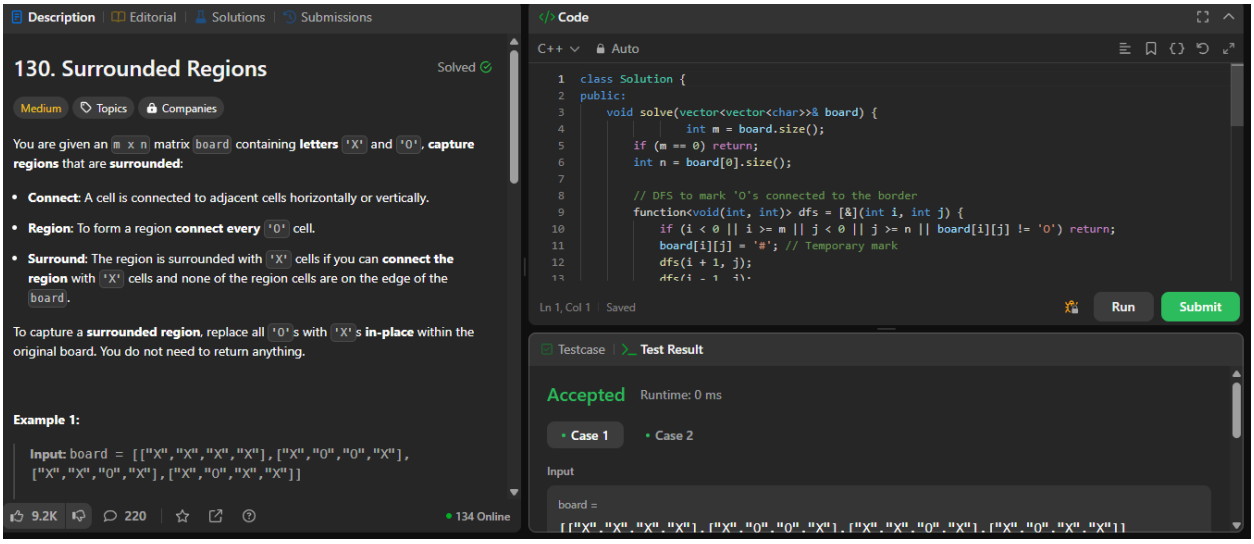
Explanation:

In the above diagram, the bottom region is not captured because it is on the edge of the board and cannot be surrounded.

Solution:

```
class Solution {
public:
    void solve(vector<vector<char>>& board) {
            int m = board.size();
        if (m == 0) return;
        int n = board[0].size();

        // DFS to mark 'O's connected to the border
        function<void(int, int)> dfs = [&](int i, int j) {
            if (i < 0 || i >= m || j < 0 || j >= n || board[i][j] != 'O') return;
            board[i][j] = '#'; // Temporary mark
            dfs(i + 1, j);
            dfs(i - 1, j);
            dfs(i, j + 1);
            dfs(i, j - 1);
        };

        // Start DFS from the borders
        for (int i = 0; i < m; ++i) {
            dfs(i, 0);
            dfs(i, n - 1);
        }
        for (int j = 0; j < n; ++j) {
            dfs(0, j);
            dfs(m - 1, j);
        }
```

```
        // Final pass to capture surrounded regions
        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';
                }
            }
        }
    }
};
```



124. Binary Tree Maximum Path Sum

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.

Example 1:

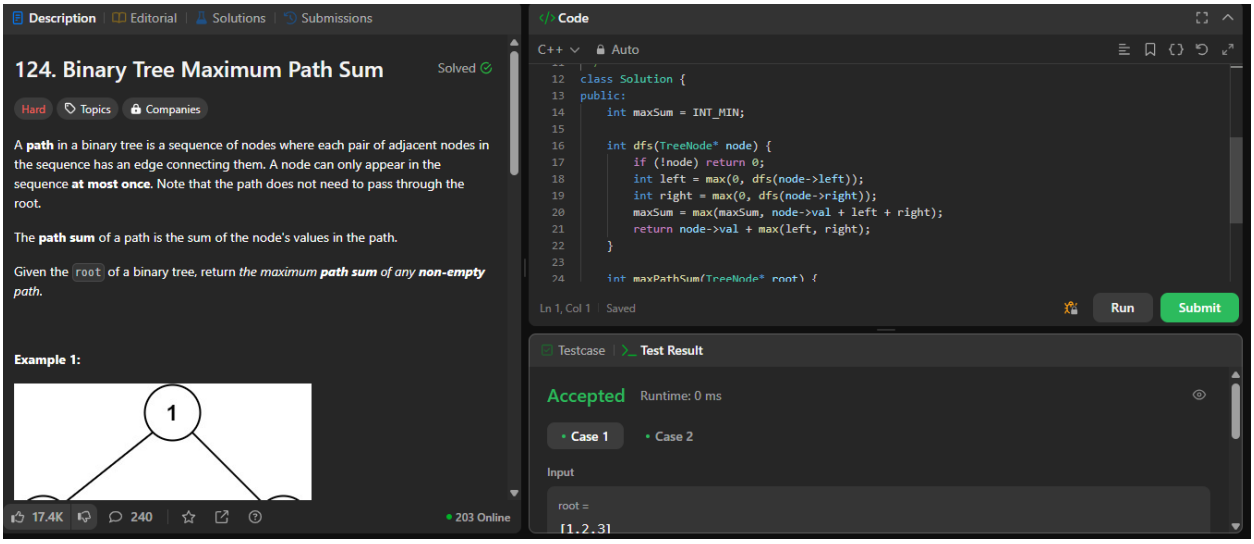
Input: root = [1,2,3]

Output: 6

Explanation: The optimal path is 2 -> 1 -> 3 with a path sum of 2 + 1 + 3 = 6.

Solution:

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

    int dfs(TreeNode* node) {
        if (!node) return 0;
        int left = max(0, dfs(node->left));
        int right = max(0, dfs(node->right));
        maxSum = max(maxSum, node->val + left + right);
        return node->val + max(left, right);
    }
    int maxPathSum(TreeNode* root) {
        dfs(root);
        return maxSum
};
```



547. Number of Provinces

There are n cities. Some of them are connected, while some are not. If city a is connected directly with city b, and city b is connected directly with city c, then city a is connected indirectly with city c.

A province is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an n x n matrix isConnected where isConnected[i][j] = 1 if the ith city and the jth city are directly connected, and isConnected[i][j] = 0 otherwise.

Return the total number of provinces.

Example 1:

Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]

Output: 2

Solution:

```
class Solution {
public:
    void dfs(vector<vector<int>>& isConnected, vector<bool>& visited, int city) {
        visited[city] = true;
```

```
        for (int j = 0; j < isConnected.size(); ++j) {
            if (isConnected[city][j] == 1 && !visited[j]) {
                dfs(isConnected, visited, j);
            }
        }
    }
    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;
    }
};
```

Description | Editorial | Solutions | Submissions

547. Number of Provinces

Medium | Topics | Companies

There are `n` cities. Some of them are connected, while some are not. If city `a` is connected directly with city `b`, and city `b` is connected directly with city `c`, then city `a` is connected indirectly with city `c`.

A **province** is a group of directly or indirectly connected cities and no other cities outside of the group.

You are given an `n x n` matrix `isConnected` where `isConnected[i][j] = 1` if the i^{th} city and the j^{th} city are directly connected, and `isConnected[i][j] = 0` otherwise.

Return *the total number of* ***provinces***.

Example 1:

10.3K | 134 | 160 Online

Code

C++ | Auto

```
        for (int j = 0; j < isConnected.size(); ++j) {
            if (isConnected[city][j] == 1 && !visited[j]) {
                dfs(isConnected, visited, j);
            }
        }
    }

    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) {
```

Ln 26, Col 3 | Saved

Run | Submit

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2

Input

isConnected =

[[1,1,0],[1,1,0],[0,0,1]]