



## Experiment 9

**Student Name:** Arfan Mohd

**UID:**22BCS12329

**Branch:** CSE

**Section/Group:** NTPP 603/B

**Semester:** 06

**Date of Performance:** 03/04/2025

**Subject Name:** AP Lab 2

**Subject Code:** 22CSP-351

### 1. Aim:

- a. Number of Islands.
- b. Word Ladder
- c. Surrounded Regions

### 2. Source Code:

#### a.

```
class Solution {
public:
    int numIslands(vector<vector<char>>& grid) {
        constexpr int kDirs[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
        const int m = grid.size();
        const int n = grid[0].size();
        int ans = 0;

        auto bfs = [&](int r, int c) {
            queue<pair<int, int>> q{{{r, c}}};
            grid[r][c] = '2'; // Mark '2' as visited.
            while (!q.empty()) {
                const auto [i, j] = q.front();
                q.pop();
                for (const auto& [dx, dy] : kDirs) {
                    const int x = i + dx;
                    const int y = j + dy;
                    if (x < 0 || x == m || y < 0 || y == n)
                        continue;
                }
            }
        };

        for (int r = 0; r < m; ++r)
            for (int c = 0; c < n; ++c)
                if (grid[r][c] == '1')
                    bfs(r, c);
        return ans;
    }
};
```

```
        if (grid[x][y] != '1')
            continue;
        q.emplace(x, y);
        grid[x][y] = '2'; // Mark '2' as visited.
    }
}
};

for (int i = 0; i < m; ++i)
    for (int j = 0; j < n; ++j)
        if (grid[i][j] == '1') {
            bfs(i, j);
            ++ans;
        }

return ans;
}
};
```

**b.**

```
class Solution {
public:
    void solve(vector<vector<char>>& board) {
        if (board.empty())
            return;

        constexpr int kDirs[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
        const int m = board.size();
        const int n = board[0].size();

        queue<pair<int, int>> q;

        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
                if (i * j == 0 || i == m - 1 || j == n - 1)
                    if (board[i][j] == '0') {
                        q.emplace(i, j);
                        board[i][j] = '*';
                    }

        // Mark the grids that stretch from the four sides with '*'.
        while (!q.empty()) {
```

```
const auto [i, j] = q.front();
q.pop();
for (const auto& [dx, dy] : kDirs) {
    const int x = i + dx;
    const int y = j + dy;
    if (x < 0 || x == m || y < 0 || y == n)
        continue;
    if (board[x][y] != 'O')
        continue;
    q.emplace(x, y);
    board[x][y] = '*';
}
}

for (vector<char>& row : board)
    for (char& c : row)
        if (c == '*')
            c = 'O';
        else if (c == 'O')
            c = 'X';
}
};
```

**C.**

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

        queue<string> q{{beginWord}};

        for (int step = 1; !q.empty(); ++step)
            for (int sz = q.size(); sz > 0; --sz) {
                string word = q.front();
                q.pop();
                for (int i = 0; i < word.length(); ++i) {
                    const char cache = word[i];
                    for (char c = 'a'; c <= 'z'; ++c) {
                        word[i] = c;
                        if (word == endWord)
                            return step + 1;
                        if (wordSet.contains(word)) {
```

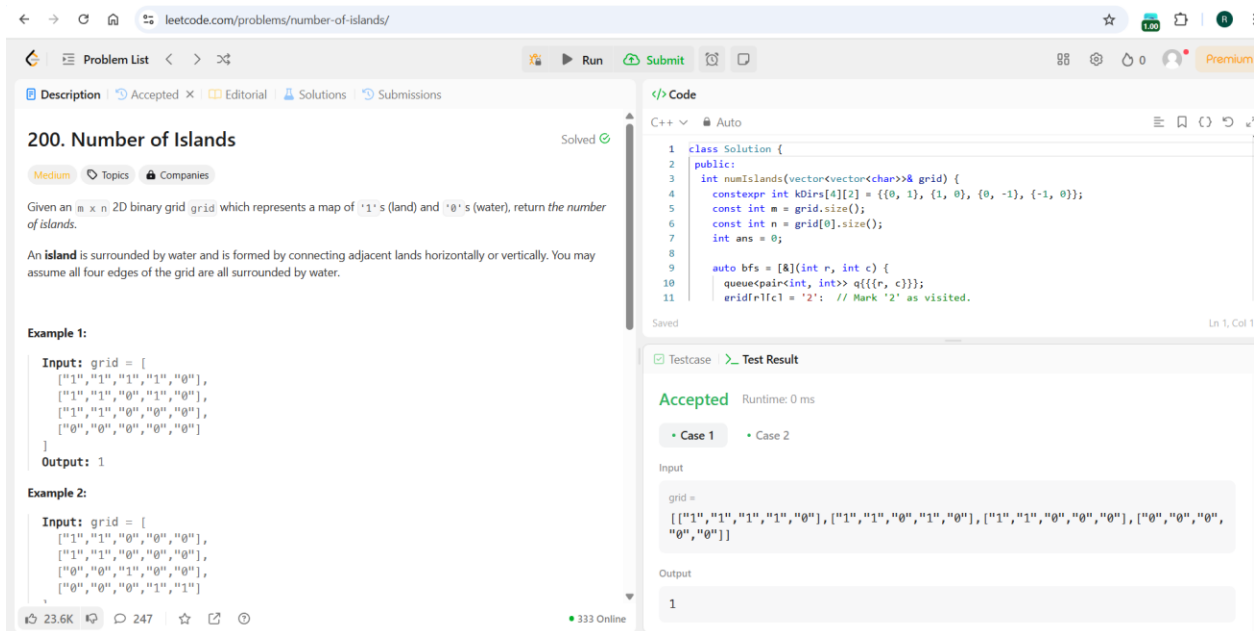
```

        q.push(word);
        wordSet.erase(word);
    }
}
word[i] = cache;
}
}
return 0;
}
};

```

### 3. Screenshot of Outputs:

a.



200. Number of Islands

Medium

Given an  $m \times n$  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

Example 2:

Input: `grid = [ [1,1,0,0,0], [1,1,0,0,0], [0,0,1,0,0], [0,0,0,1,1] ]`

Output: 2

Accepted Runtime: 0 ms

Case 1 Case 2

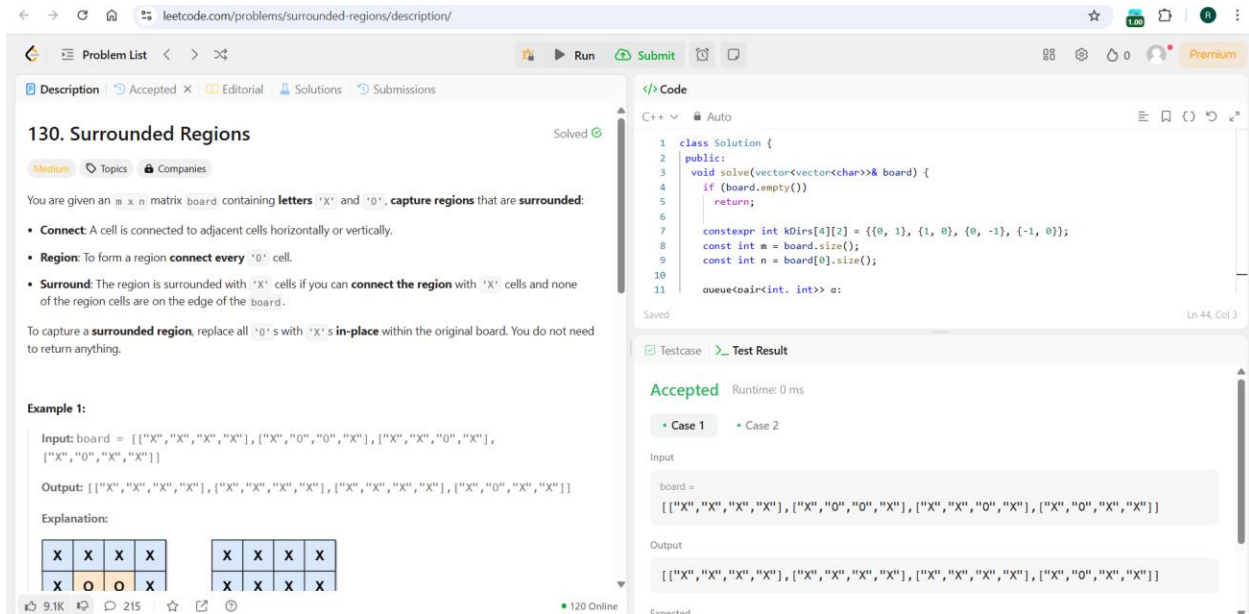
Input

grid = [ [1,1,1,1,0], [1,1,0,1,0], [1,1,0,0,0], [0,0,0,0,0] ]

Output

1

b.



**130. Surrounded Regions** Solved

Medium Topics Companies

You are given an  $m \times n$  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:

X	X	X	X	X	X	X	X
X	O	O	X	X	X	X	X

9.1K 215 120 Online

```

1 class Solution {
2 public:
3 void solve(vector<vector<char>>& board) {
4     if (board.empty())
5         return;
6
7     constexpr int kDirs[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};
8     const int m = board.size();
9     const int n = board[0].size();
10
11     queue<pair<int, int>> q;
  
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

board =

```

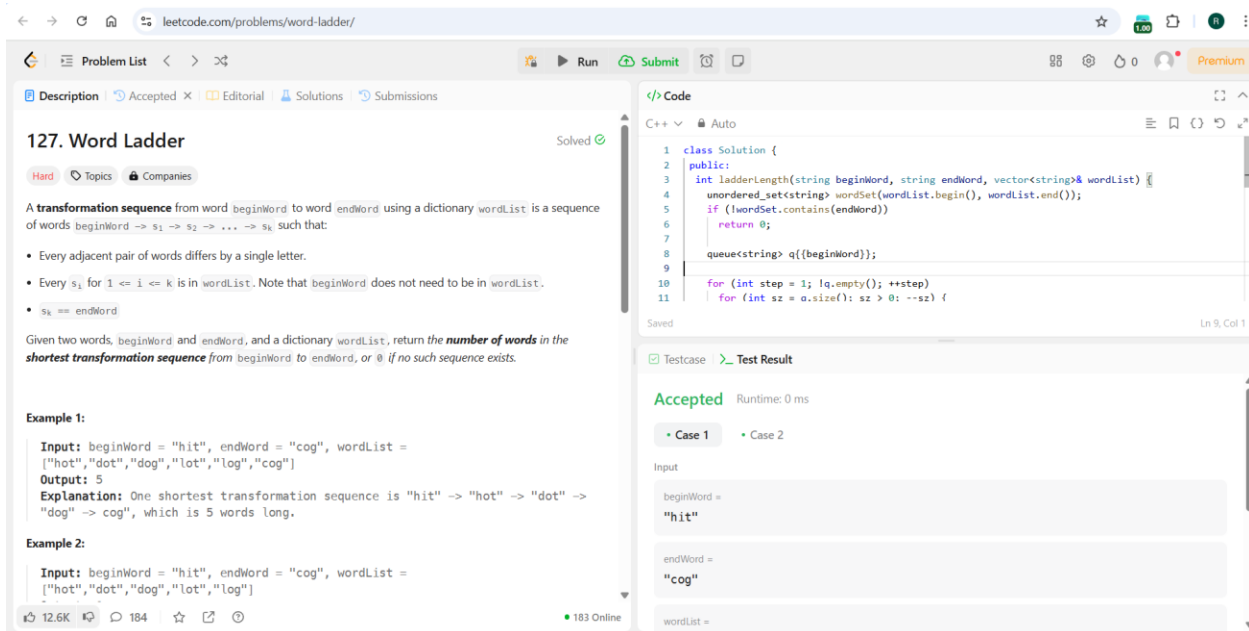
[["X","X","X","X"],
 ["X","O","O","X"],
 ["X","X","O","X"],
 ["X","O","X","X"]]
  
```

Output

```

[["X","X","X","X"],
 ["X","X","X","X"],
 ["X","X","X","X"],
 ["X","O","X","X"]]
  
```

C.



**127. Word Ladder** Solved

Hard Topics Companies

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  $1 \leq i \leq k$  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.

**Example 2:**

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

Output: 0

12.6K 184 183 Online

```

1 class Solution {
2 public:
3 int ladderLength(string beginWord, string endWord, vector<string>& wordList) {
4     unordered_set<string> wordSet(wordList.begin(), wordList.end());
5     if (!wordSet.contains(endWord))
6         return 0;
7
8     queue<string> q({beginWord});
9
10    for (int step = 1; !q.empty(); ++step)
11        for (int sz = q.size(); sz > 0; --sz) {
  
```

Accepted Runtime: 0 ms

Case 1 Case 2

Input

beginWord =

```
"hit"
```

endWord =

```
"cog"
```

wordList =

## 4. Learning Outcomes

(i) Learned about graph data structures.