Experiment 1

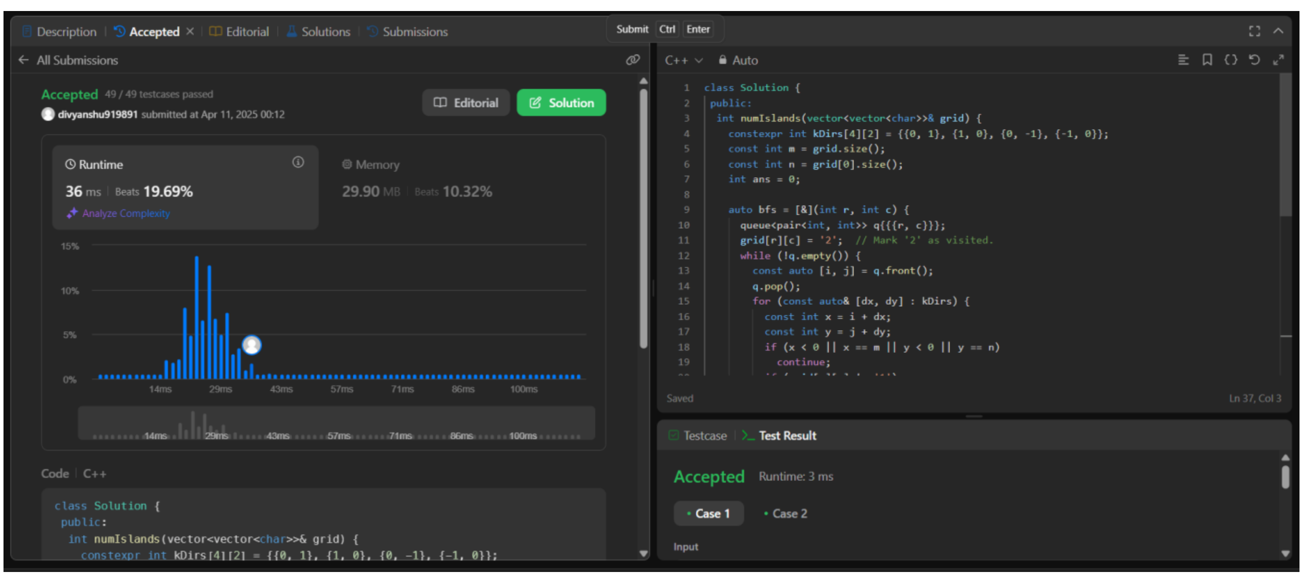
Student Name: Divyanshu **UID:22BCS14850**
Branch: CSE **Section/Group:614 B**
Semester: 6 **Date :09/04/2025**
Subject Name:AP Lab 2 **Subject Code: 22CSH-351**

1. Aim: Number of Islands

Problem Statement : 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.

2. Implementation/Code and Output:



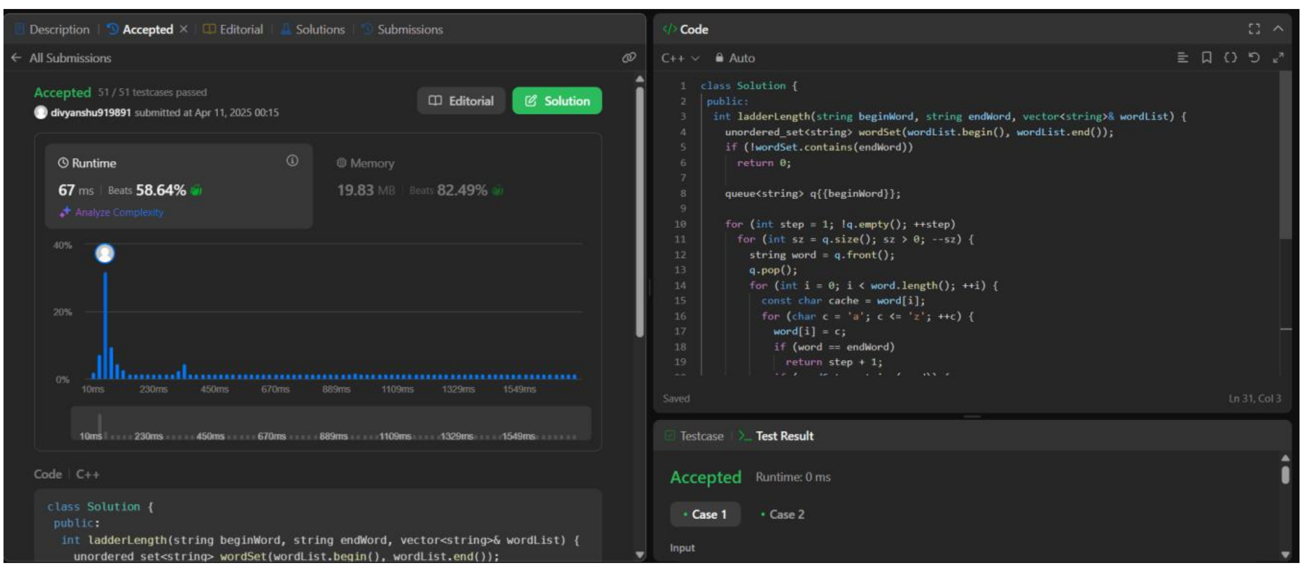
1.Aim: Word Ladder

Problem Statement: A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord` -> s_1 -> s_2 -> ... -> s_k such that:

- Every adjacent pair of words differs by a single letter.
- Every s_i for `1 <= i <= k` is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- s_k `== 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.*

2.Implementation/Code and Output:



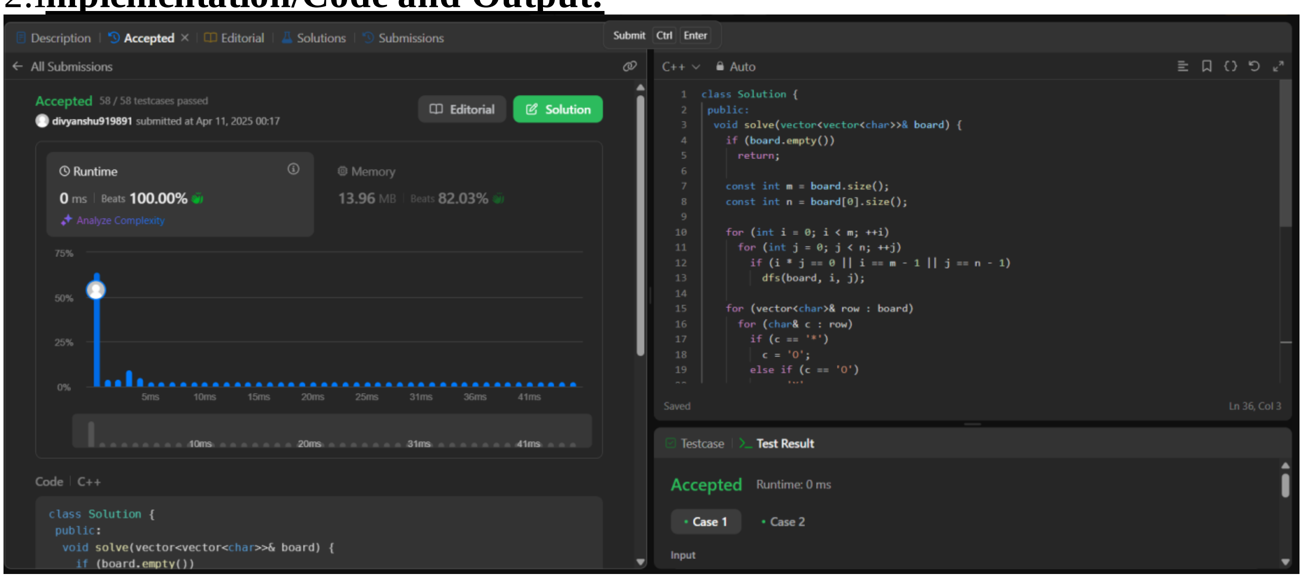
1.Aim:Surrounded Regions

Problem Statement: 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.

2.Implementation/Code and Output:



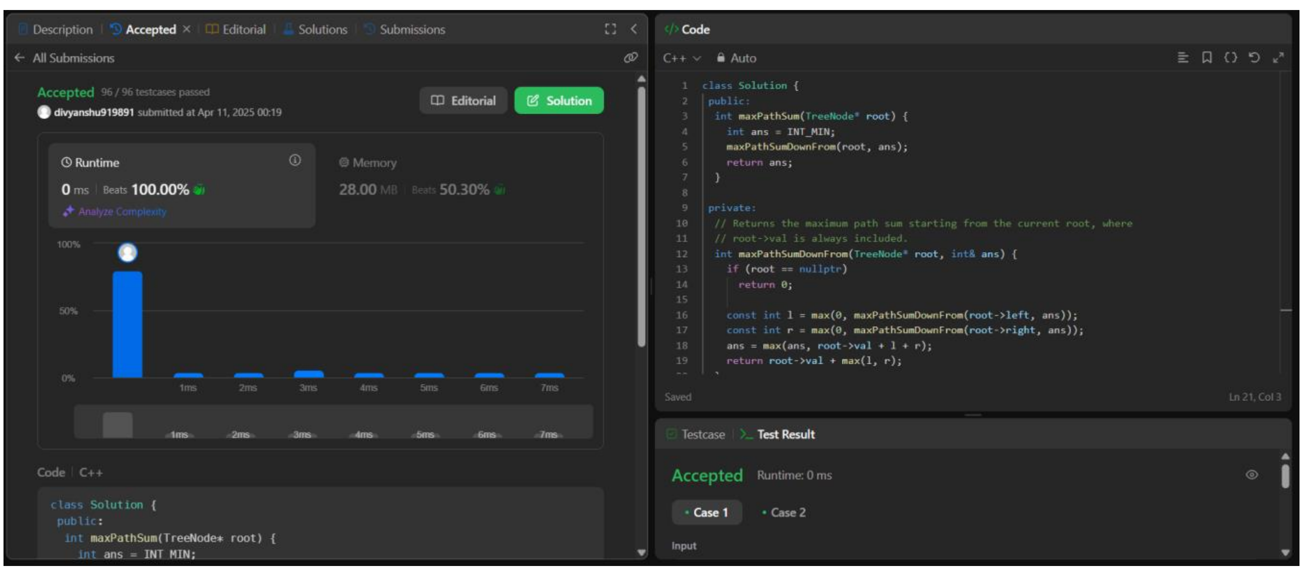
1.Aim: Binary Tree Maximum Path Sum

Problem Statements: 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*.

2.Implementation and output:



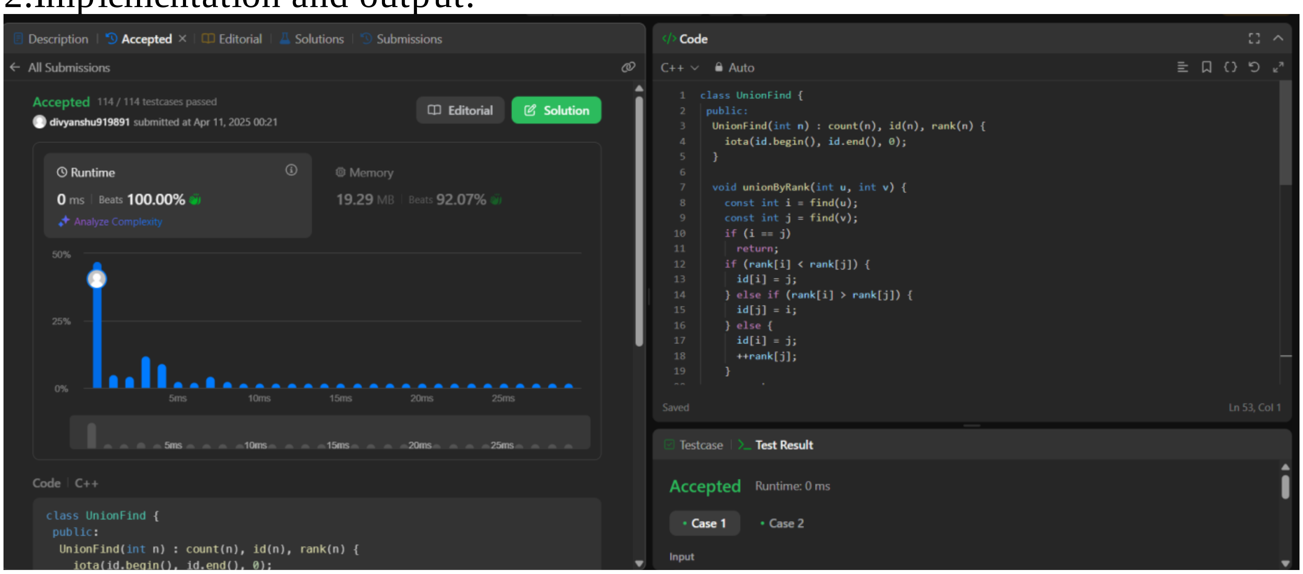
1.Aim: Number of Provinces

Problem Statement: 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.

2.Implementation and output:

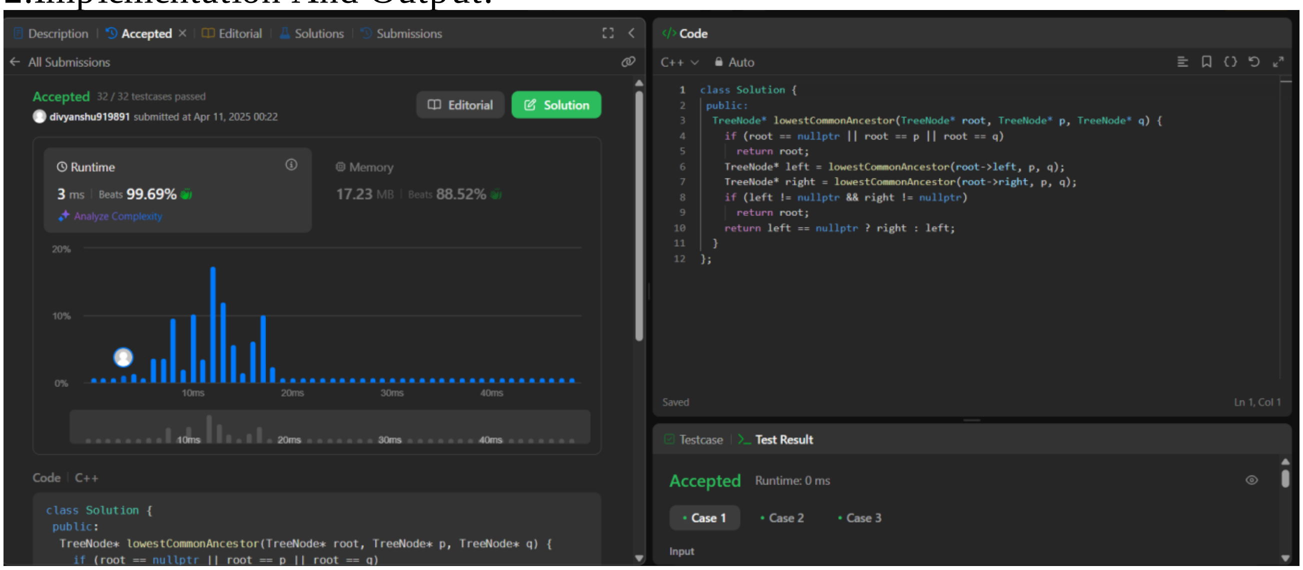


1.Aim: Lowest Common Ancestor of a Binary Tree

Problem Statement: Given a binary tree, find the lowest common ancestor (LCA) of two given nodes in the tree.

According to the definition of LCA on Wikipedia: “The lowest common ancestor is defined between two nodes p and q as the lowest node in T that has both p and q as descendants (where we allow **a node to be a descendant of itself**).”

2.Implementation And Output:



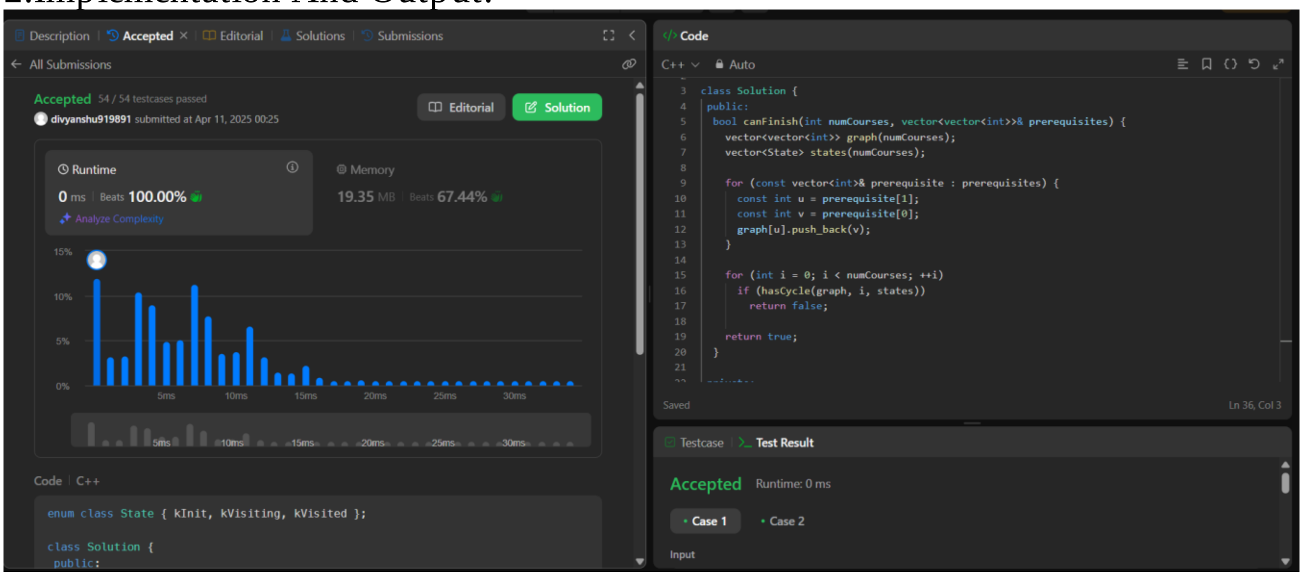
1.Aim: Course Schedule

Problem Statement: There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [a`$_i$`, b`$_i$`]` indicates that you **must** take course b_i first if you want to take course a_i.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

2.Implementation And Output:

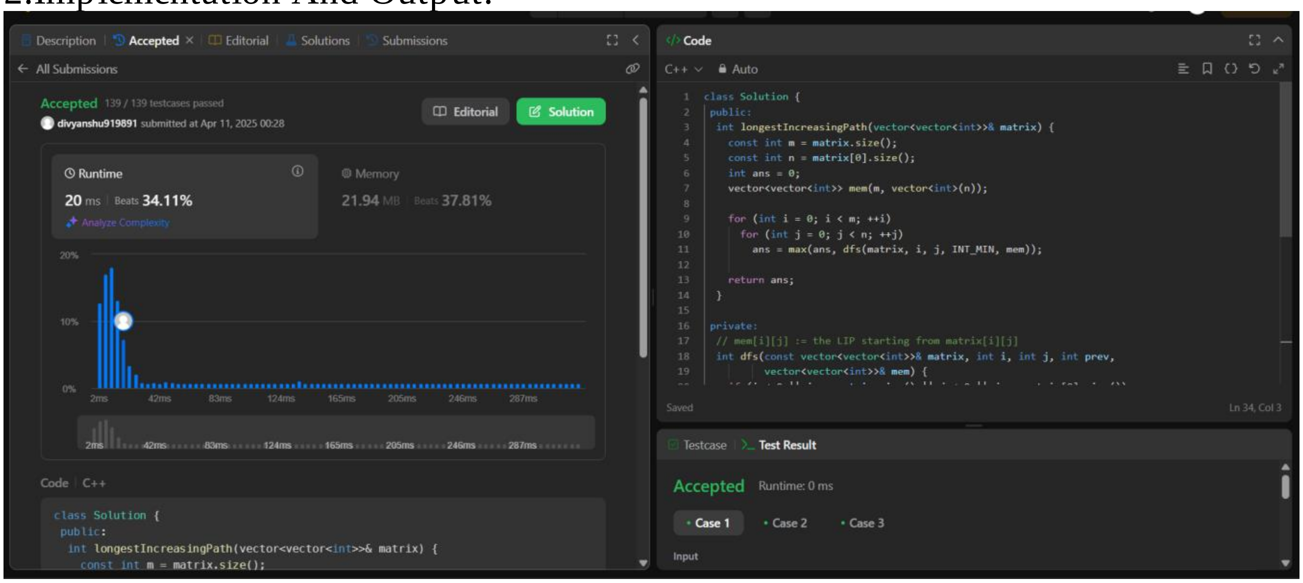


1.Aim: Longest Increasing Path in a Matrix

Problem Statement: Given an `m x n` integers `matrix`, return *the length of the longest increasing path in* `matrix`.

From each cell, you can either move in four directions: left, right, up, or down. You **may not** move **diagonally** or move **outside the boundary** (i.e., wrap-around is not allowed).

2.Implementation And Output:



1.Aim: **Course Schedule 2**

Problem Statement: There are a total of `numCourses` courses you have to take, labeled from `0` to `numCourses - 1`. You are given an array `prerequisites` where `prerequisites[i] = [`a_i`, `b_i`]` indicates that you **must** take course b_i first if you want to take course a_i.

- For example, the pair `[0, 1]`, indicates that to take course `0` you have to first take course `1`.

Return `true` if you can finish all courses. Otherwise, return `false`.

2.Implementation And Output: