

## Experiment 1

**Student Name:** Divyanshu  
**Branch:** CSE  
**Semester:** 6  
**Subject Name:** AP Lab 2

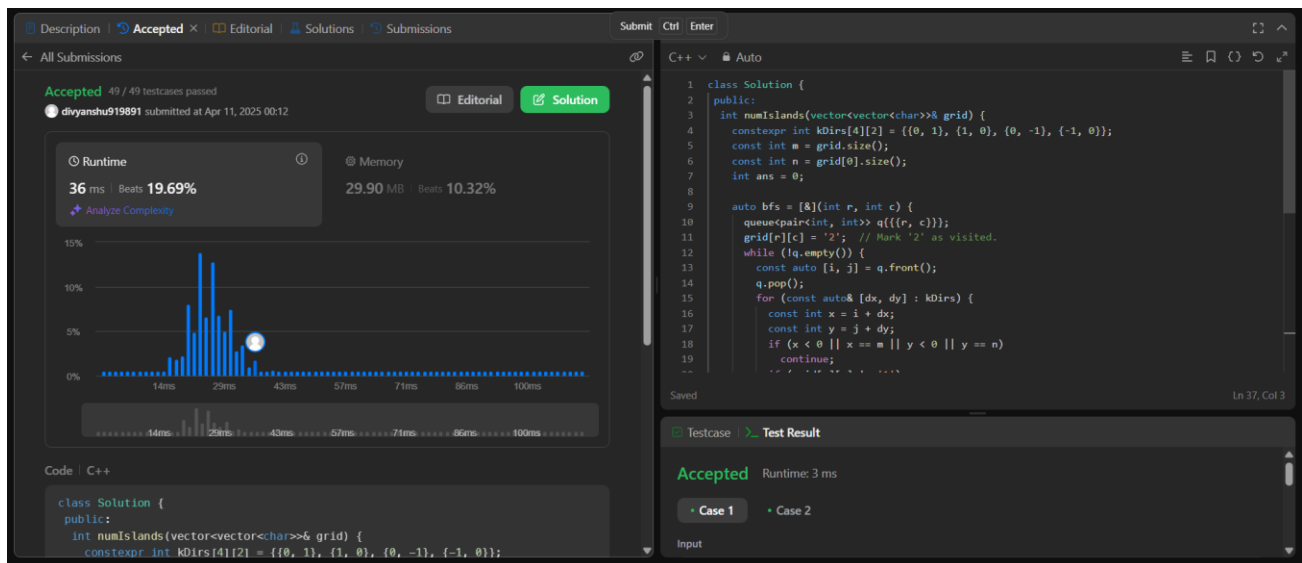
**UID:**22BCS14850  
**Section/Group:**614 B  
**Date :**09/04/2025  
**Subject Code:** 22CSH-351

### 1. Aim: **Number of Islands**

**Problem Statement :** 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.

### 2. Implementation/Code and Output:



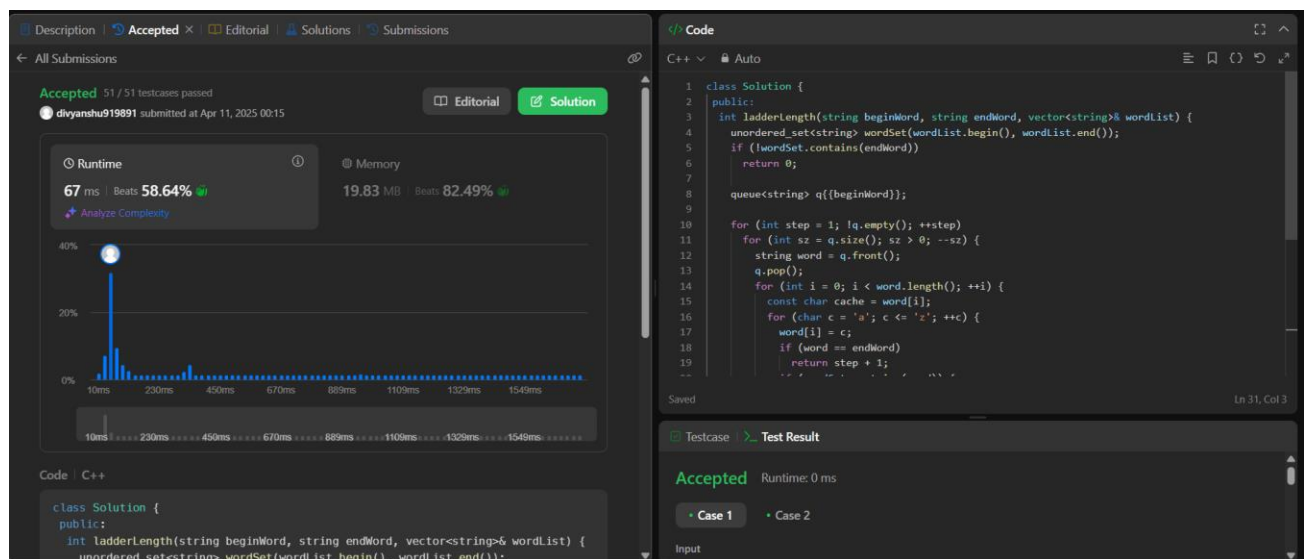
## 1.Aim: Word Ladder

**Problem Statement:** A **transformation sequence** from word `beginWord` to word `endWord` using a dictionary `wordList` is a sequence of words `beginWord`  $\rightarrow s_1 \rightarrow s_2 \rightarrow \dots \rightarrow s_k$  such that:

- Every adjacent pair of words differs by a single letter.
- Every  $s_i$  for  $1 \leq i \leq k$  is in `wordList`. Note that `beginWord` does not need to be in `wordList`.
- $s_k == \text{endWord}$

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:



**Accepted** 51 / 51 testcases passed  
divyanshu919891 submitted at Apr 11, 2025 00:15

**Runtime** 67 ms | Beats 58.64%  
**Memory** 19.83 MB | Beats 82.49%

**Code** C++

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

**Testcase** **Test Result**  
**Accepted** Runtime: 0 ms  
Case 1 Case 2

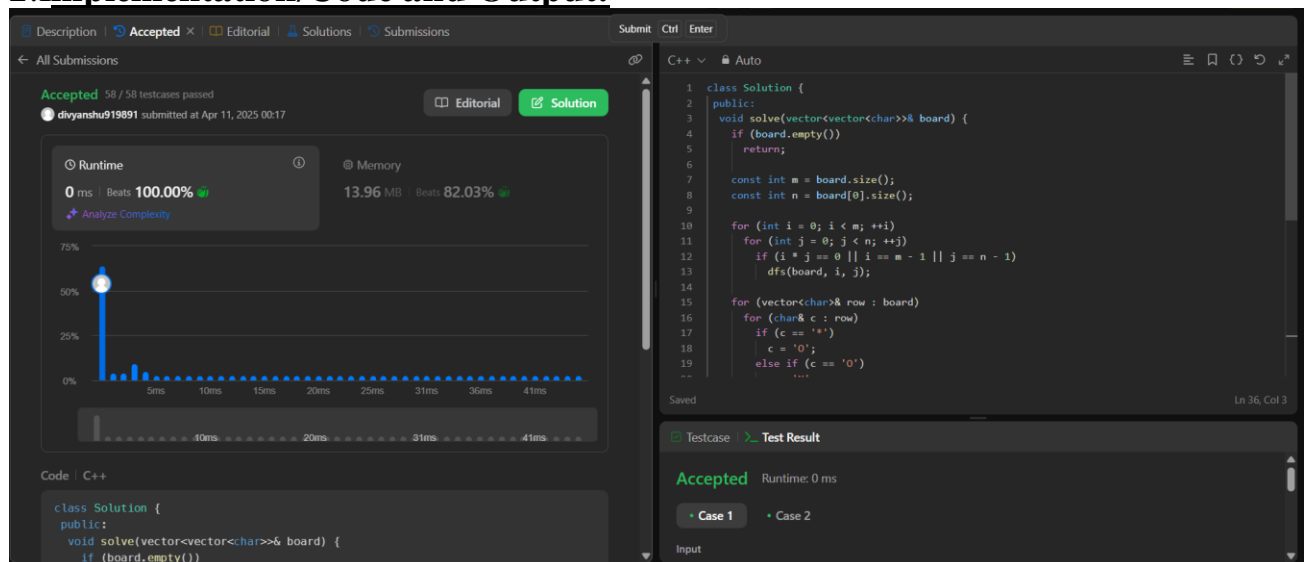
## 1. Aim: **Surrounded Regions**

**Problem Statement:** 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.

## 2. Implementation/Code and Output:



The screenshot displays a coding platform interface for the 'Surrounded Regions' problem. The top navigation bar includes 'Description', 'Accepted', 'Editorial', 'Solutions', and 'Submissions'. The 'Accepted' tab is active, showing a submission by 'divyanshu919891' submitted at Apr 11, 2025 00:17. The submission status is 'Accepted' with 58/58 testcases passed. The runtime is 0 ms (Beats 100.00%) and memory is 13.96 MB (Beats 82.03%). A runtime graph shows a single peak at 0 ms. The code is written in C++ and implements a DFS solution to capture surrounded regions.

```

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

        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
                if (i * j == 0 || i == m - 1 || j == n - 1)
                    dfs(board, i, j);

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

The bottom right section shows the 'Testcase' and 'Test Result' for 'Case 1', which is 'Accepted' with a runtime of 0 ms.

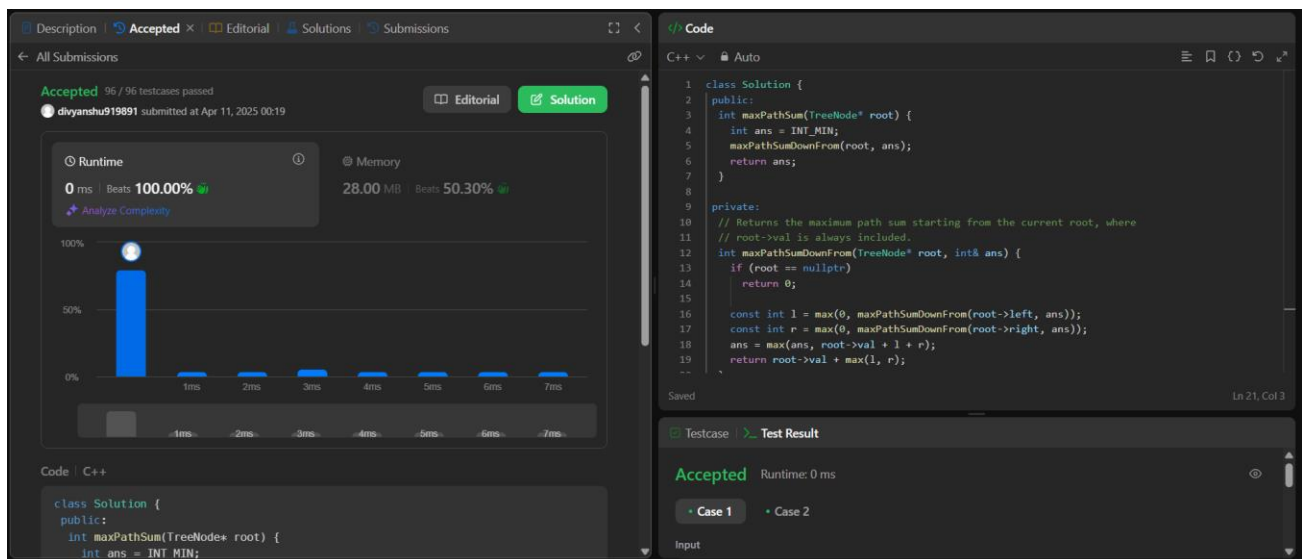
## 1. Aim: Binary Tree Maximum Path Sum

**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:



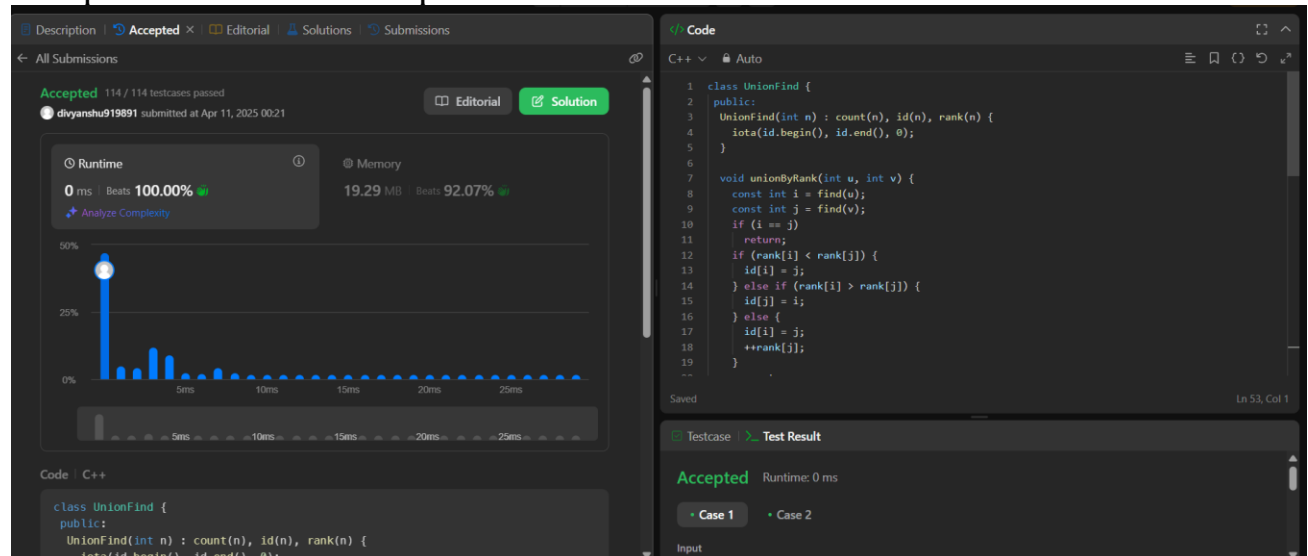
## 1. Aim: **Number of Provinces**

**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  $n \times n$  matrix `isConnected` where `isConnected[i][j] = 1` if the  $i^{\text{th}}$  city and the  $j^{\text{th}}$  city are directly connected, and `isConnected[i][j] = 0` otherwise.

## 2. Implementation and output:

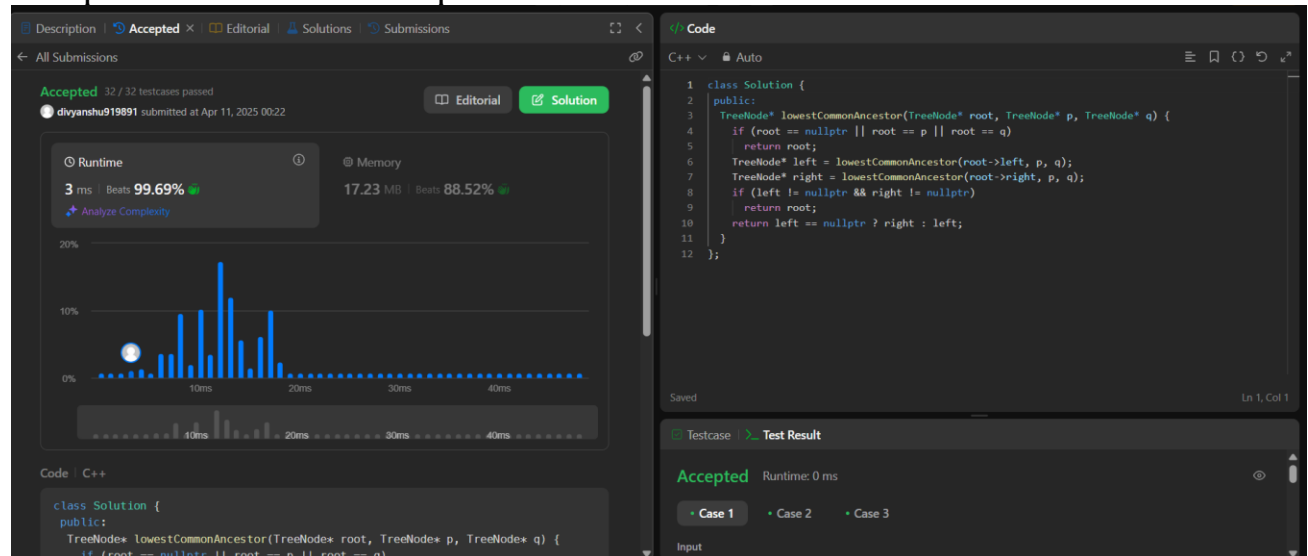


## 1.Aim: **Lowest Common Ancestor of a Binary Tree**

**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:



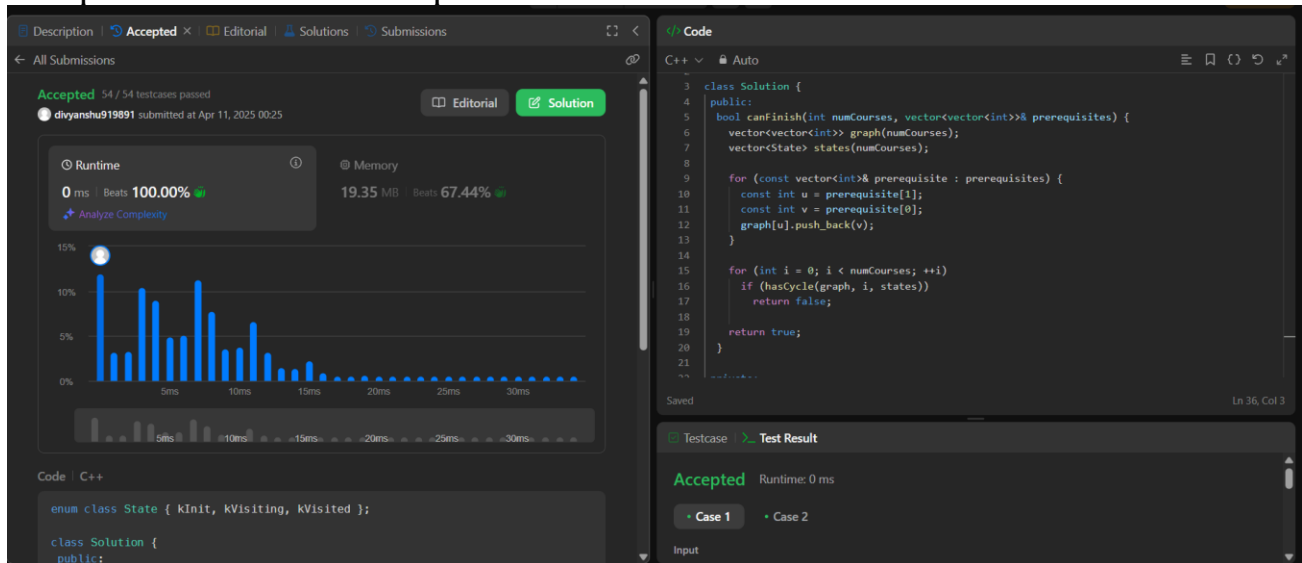
## 1.Aim: **Course Schedule**

**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  $\text{prerequisites}[i] = [a_i, b_i]$  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:

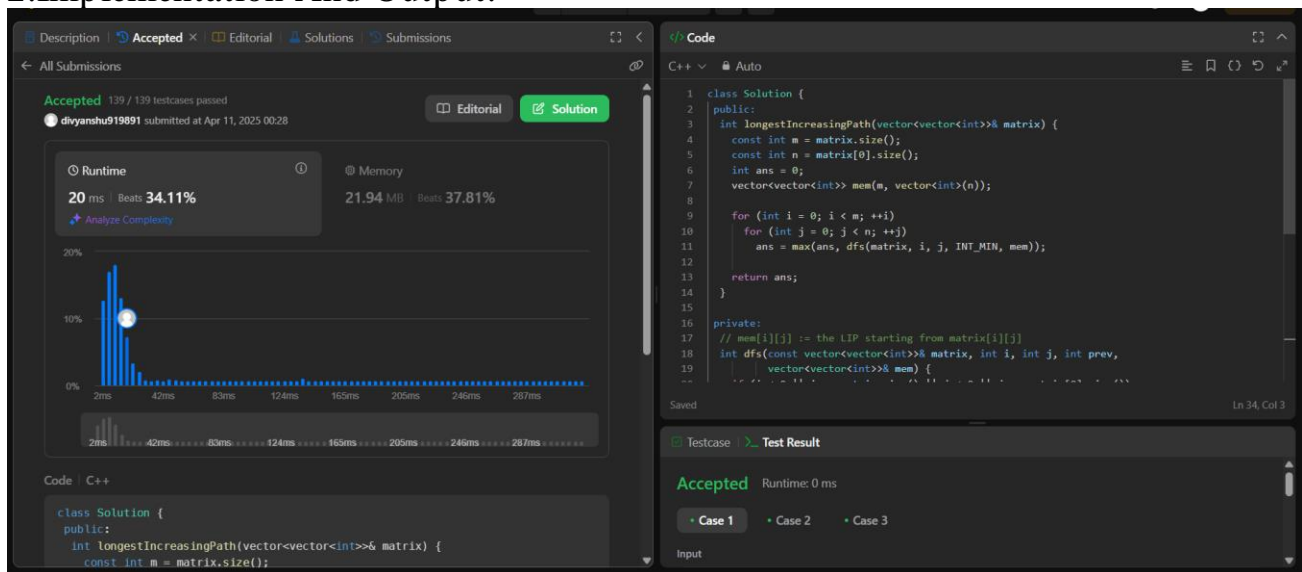


## 1. Aim: Longest Increasing Path in a Matrix

**Problem Statement:** Given an  $m \times n$  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:



The screenshot displays a code editor interface for a C++ solution. The code implements a function `longestIncreasingPath` that takes a matrix and returns the length of the longest increasing path. The implementation uses a depth-first search (DFS) with memoization to avoid redundant calculations. The code is as follows:

```
class Solution {
public:
    int longestIncreasingPath(vector<vector<int>>& matrix) {
        const int m = matrix.size();
        const int n = matrix[0].size();
        int ans = 0;
        vector<vector<int>> mem(m, vector<int>(n));
        for (int i = 0; i < m; ++i)
            for (int j = 0; j < n; ++j)
                ans = max(ans, dfs(matrix, i, j, INT_MIN, mem));
        return ans;
    }
private:
    // mem[i][j] := the LIP starting from matrix[i][j]
    int dfs(const vector<vector<int>>& matrix, int i, int j, int prev,
            vector<vector<int>>& mem) {
        if (i < 0 || i >= matrix.size() || j < 0 || j >= matrix[0].size() ||
            matrix[i][j] <= prev) return 0;
        if (mem[i][j] != 0) return mem[i][j];
        int ans = 1;
        if (i > 0 && matrix[i-1][j] < matrix[i][j]) ans = max(ans, 1 + dfs(matrix, i-1, j, matrix[i][j], mem));
        if (i < matrix.size()-1 && matrix[i+1][j] < matrix[i][j]) ans = max(ans, 1 + dfs(matrix, i+1, j, matrix[i][j], mem));
        if (j > 0 && matrix[i][j-1] < matrix[i][j]) ans = max(ans, 1 + dfs(matrix, i, j-1, matrix[i][j], mem));
        if (j < matrix[0].size()-1 && matrix[i][j+1] < matrix[i][j]) ans = max(ans, 1 + dfs(matrix, i, j+1, matrix[i][j], mem));
        mem[i][j] = ans;
        return ans;
    }
};
```

The output section shows the code is **Accepted** with a runtime of 0 ms. The test cases are Case 1, Case 2, and Case 3.

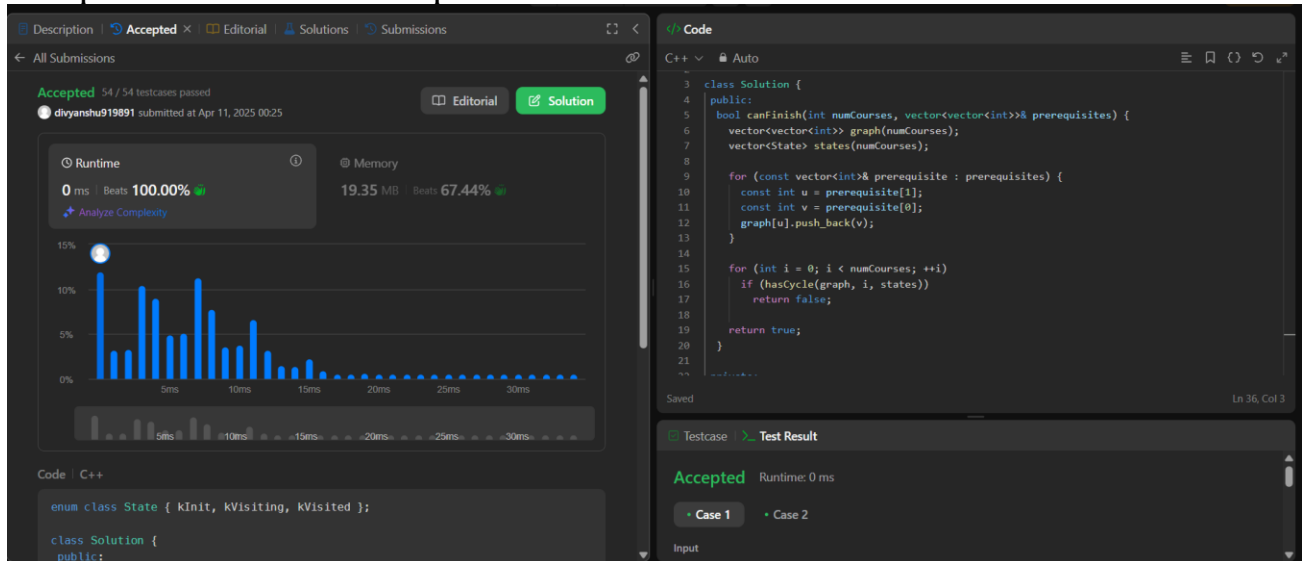


## 1.Aim: Course Schedule 2

**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] = [ai, bi]` indicates that you **must** take course `bi` first if you want to take course `ai`.

- 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:





# **DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

---

Discover. Learn. Empower.