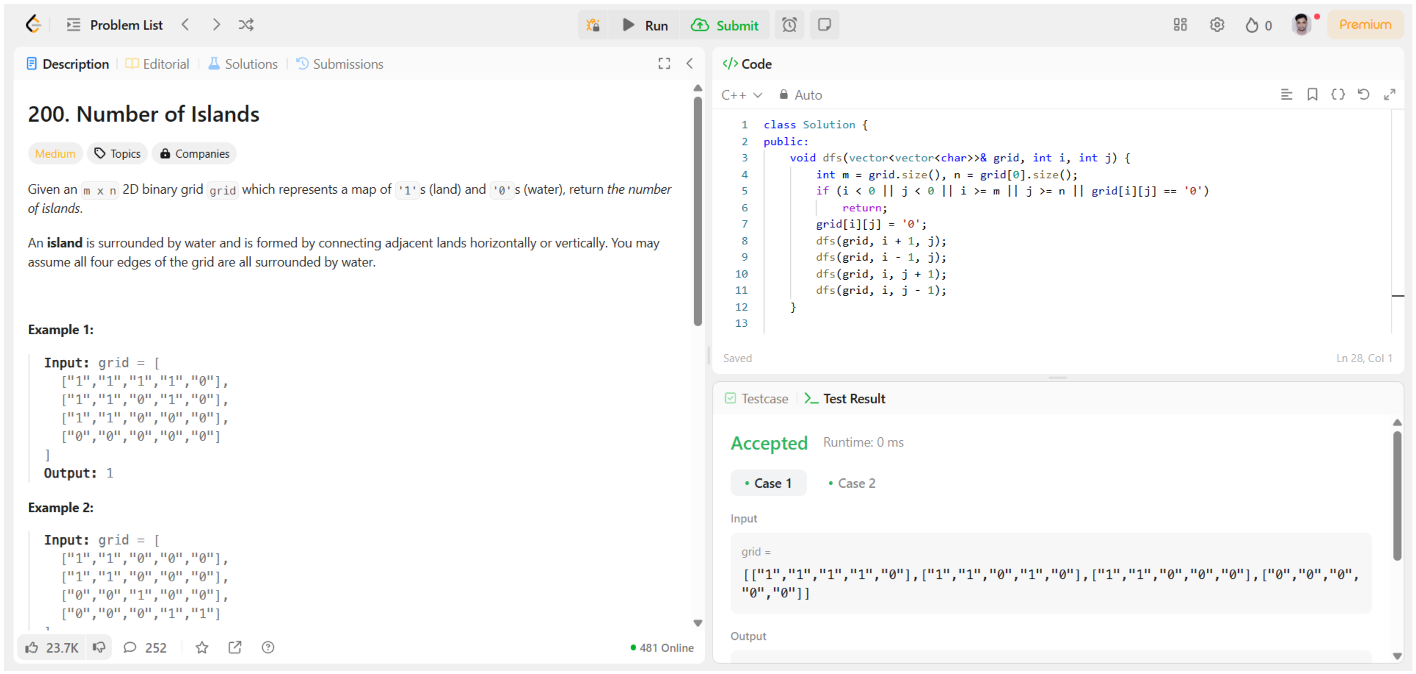
# Experiment 9

**Student Name: Saurav Ashiwal** **UID:22BCS13250**
**Branch: CSE** **Section/Group: 614/B**
**Semester: 6** **Date of Performance:10/04/2025**
**Subject Name: AP lab-2** **Subject Code: 22CSP-351**

## **Q 1.** Number of Islands



200. Number of Islands

Medium Topics Companies

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
```

**Example 2:**

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

23.7K 252 481 Online

```cpp
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int i, int j) {
        int m = grid.size(), n = grid[0].size();
        if (i < 0 || j < 0 || i >= m || j >= n || grid[i][j] == '0')
            return;
        grid[i][j] = '0';
        dfs(grid, i + 1, j);
        dfs(grid, i - 1, j);
        dfs(grid, i, j + 1);
        dfs(grid, i, j - 1);
    }

```

Saved Ln 28, Col 1

Accepted Runtime: 0 ms

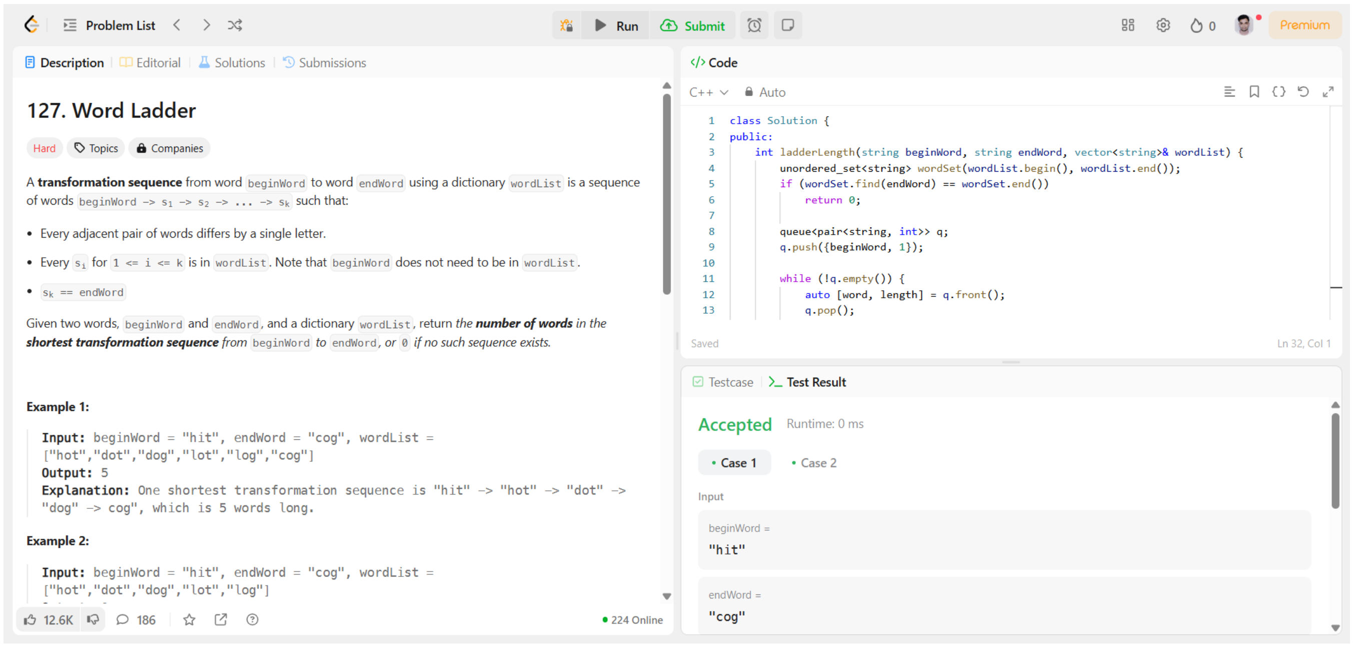
Case 1 Case 2

Input

```
grid =
[["1","1","1","1","0"],["1","1","0","1","0"],["1","1","0","0","0"],["0","0","0",
"0","0"]]
```

Output

## 2. Word Ladder



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.
```

**Example 2:**

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

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

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

        while (!q.empty()) {
            auto [word, length] = q.front();
            q.pop();
```

Saved    Ln 32, Col 1

Accepted    Runtime: 0 ms

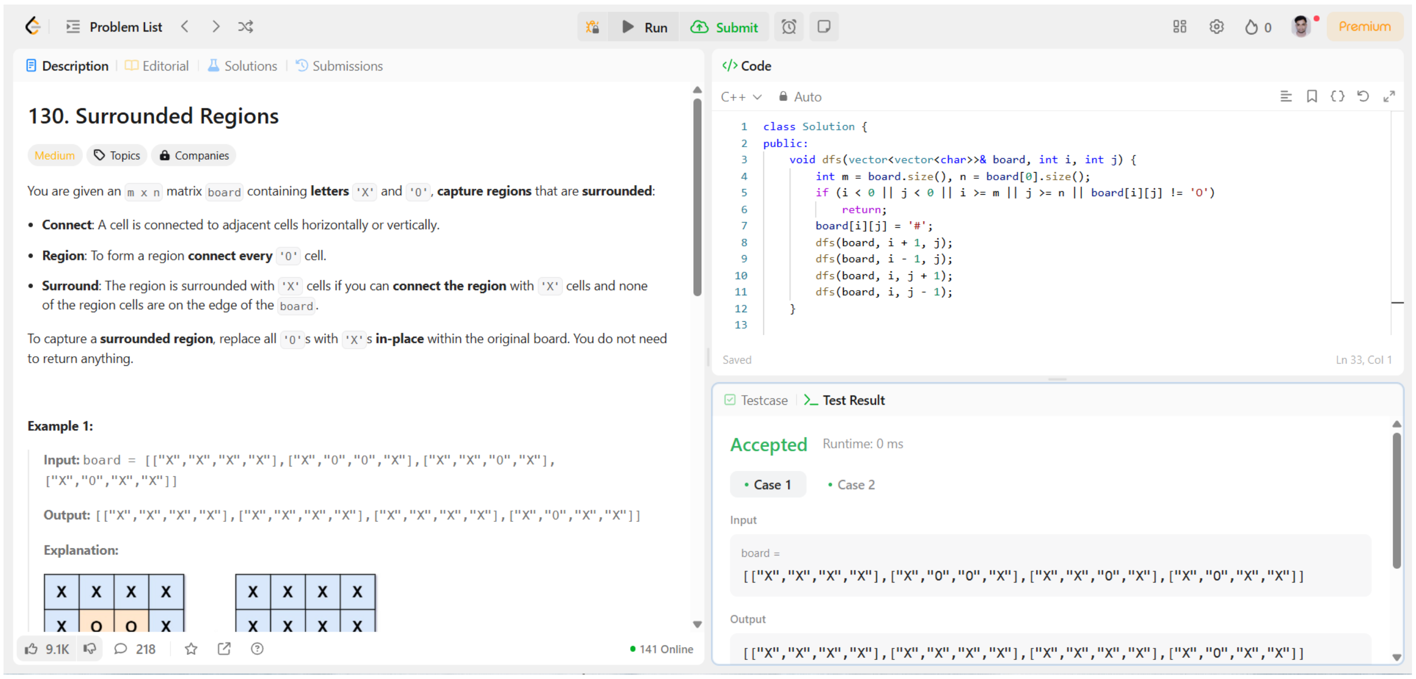
Case 1    Case 2

Input

beginWord =
"hit"

endWord =
"cog"

## 3. Surrounded Regions



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:**

| X | X | X | X |
|---|---|---|---|
| X | O | O | X |

| X | X | X | X |
|---|---|---|---|
| X | X | X | X |

```cpp
class Solution {
public:
    void dfs(vector<vector<char>>& board, int i, int j) {
        int m = board.size(), n = board[0].size();
        if (i < 0 || j < 0 || i >= m || j >= n || board[i][j] != 'O')
            return;
        board[i][j] = '#';
        dfs(board, i + 1, j);
        dfs(board, i - 1, j);
        dfs(board, i, j + 1);
        dfs(board, i, j - 1);
    }

```

Saved    Ln 33, Col 1

Accepted    Runtime: 0 ms

Case 1    Case 2

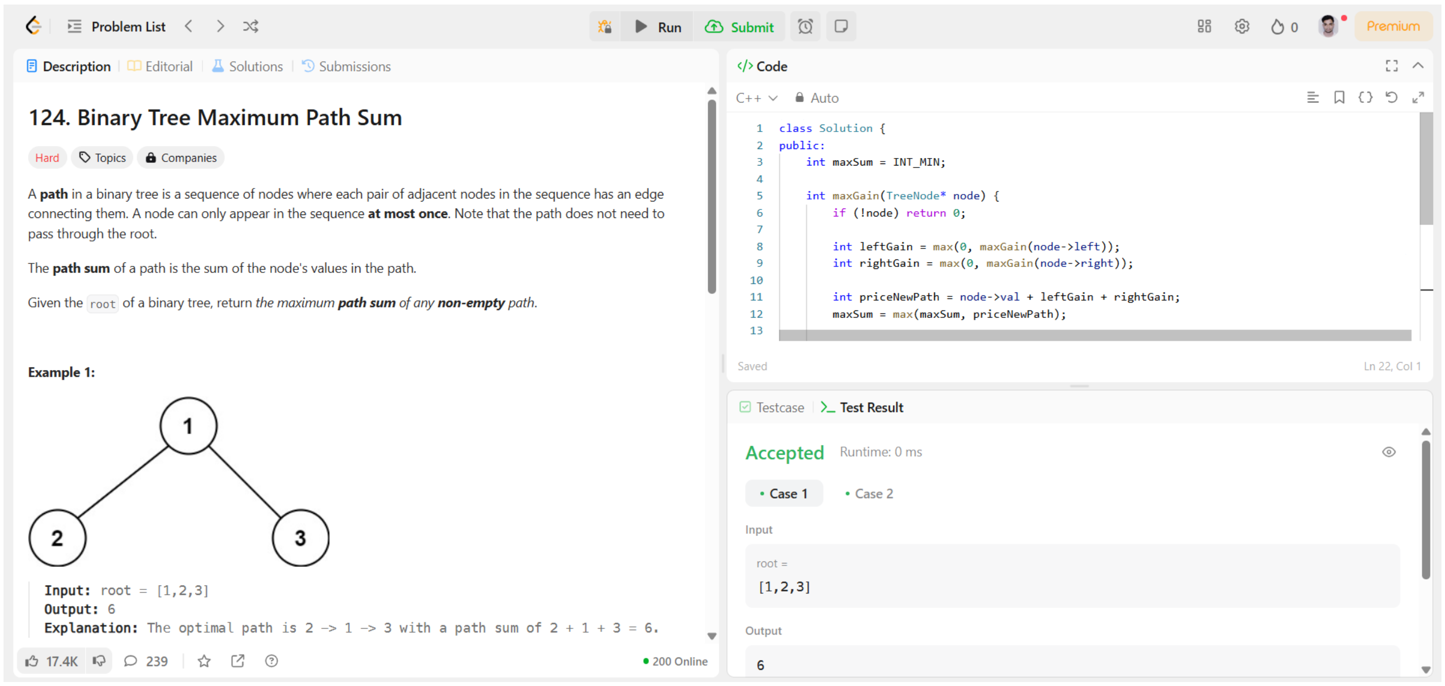
Input

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"]]

4.. Binary Tree Maximum Path Sum



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.
```

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

    int maxGain(TreeNode* node) {
        if (!node) return 0;

        int leftGain = max(0, maxGain(node->left));
        int rightGain = max(0, maxGain(node->right));

        int priceNewPath = node->val + leftGain + rightGain;
        maxSum = max(maxSum, priceNewPath);

```

Accepted Runtime: 0 ms

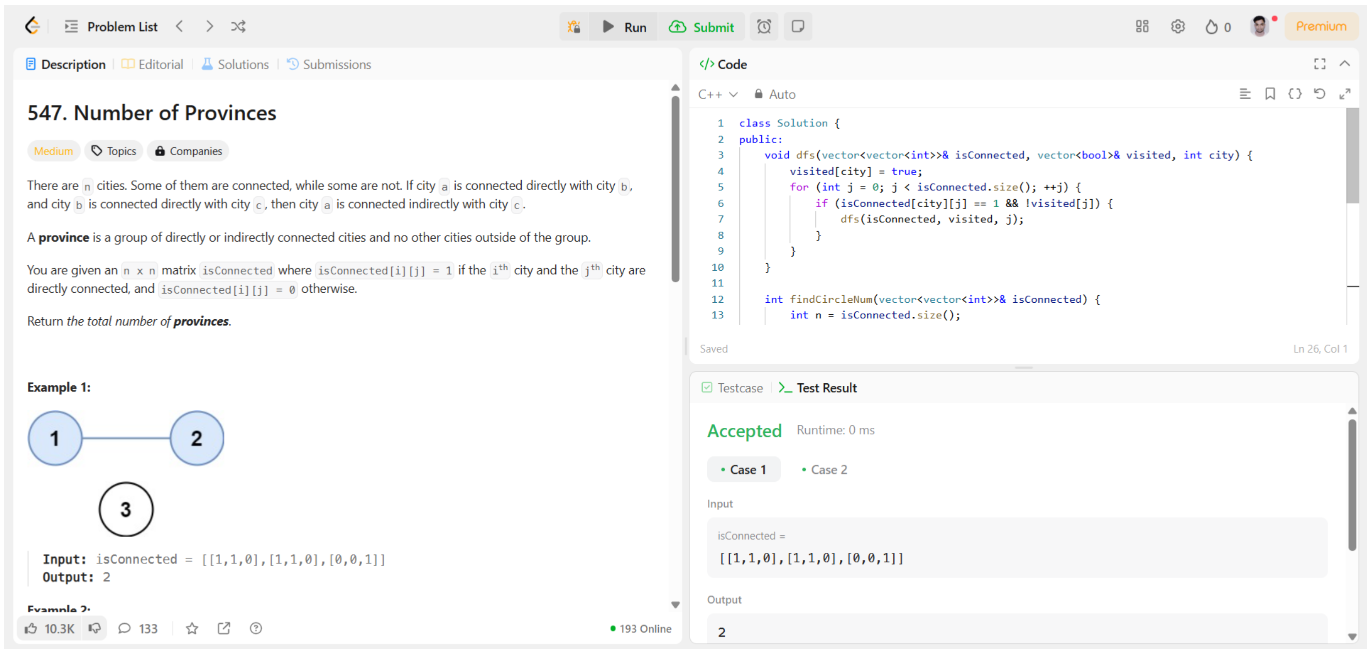
Input

root =
[1,2,3]

Output

6

## 5. Friend Circles



Problem List Run Submit Premium

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:**

```
Input: isConnected = [[1,1,0],[1,1,0],[0,0,1]]
Output: 2
```

10.3K 133 193 Online

Code

C++ Auto

```cpp
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();
```

Saved Ln 26, Col 1

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

isConnected =

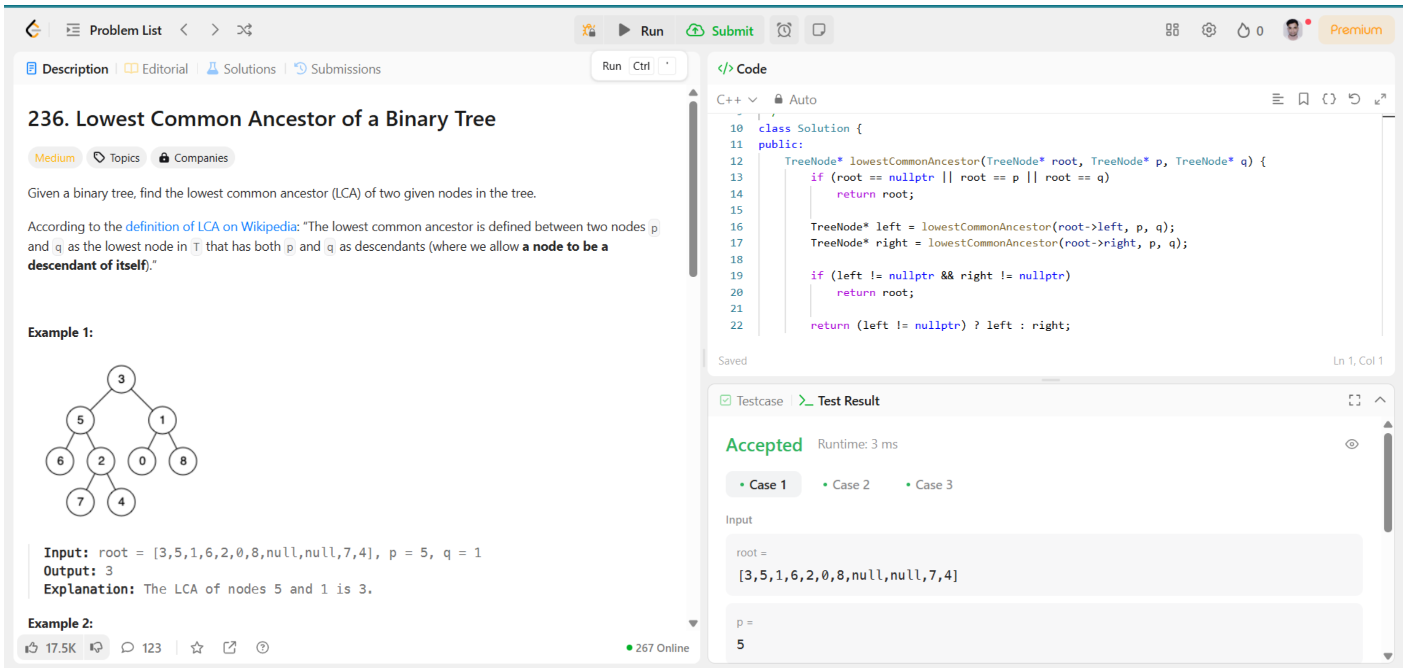
```
[[1,1,0],[1,1,0],[0,0,1]]
```

Output

```
2
```

`

6. Lowest Common Ancestor of a Binary Tree



236. Lowest Common Ancestor of a Binary Tree

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**)."

**Example 1:**

```
Input: root = [3,5,1,6,2,0,8,null,null,7,4], p = 5, q = 1
Output: 3
Explanation: The LCA of nodes 5 and 1 is 3.
```

**Example 2:**

17.5K 123 267 Online

```cpp
class Solution {
public:
    TreeNode* lowestCommonAncestor(TreeNode* root, TreeNode* p, TreeNode* q) {
        if (root == nullptr || root == p || root == q)
            return root;

        TreeNode* left = lowestCommonAncestor(root->left, p, q);
        TreeNode* right = lowestCommonAncestor(root->right, p, q);

        if (left != nullptr && right != nullptr)
            return root;

        return (left != nullptr) ? left : right;
```

Saved Ln 1, Col 1

Testcase | Test Result

Accepted Runtime: 3 ms

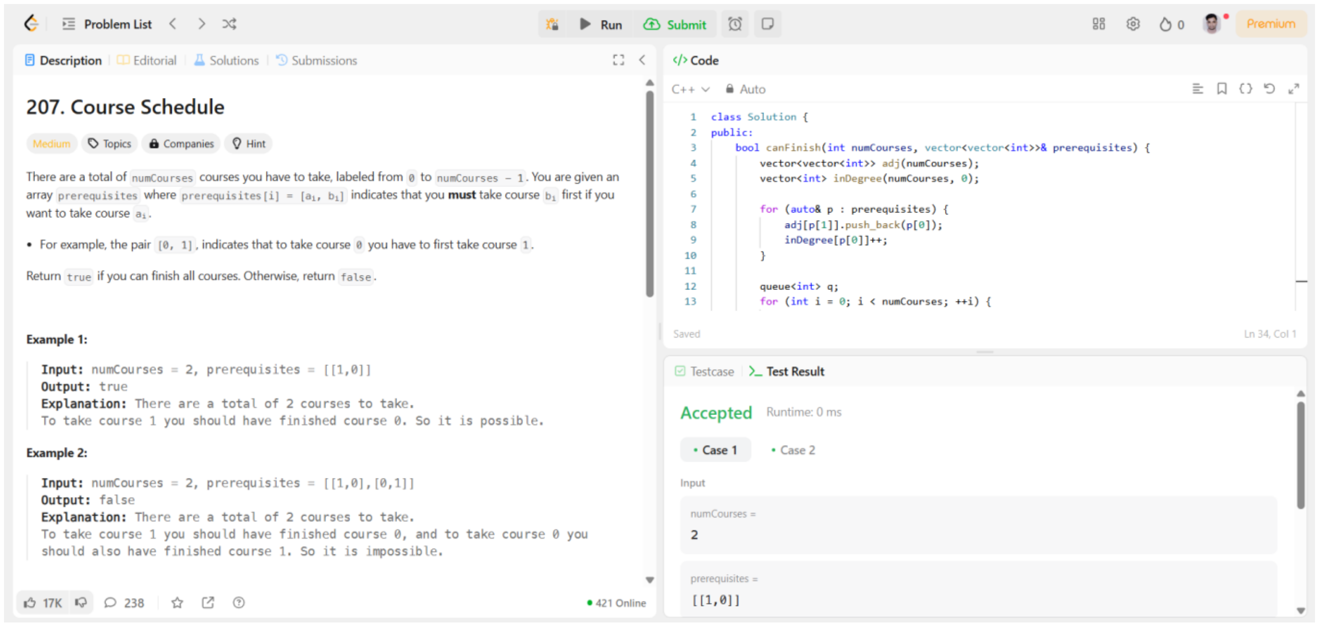
Case 1 Case 2 Case 3

Input

root =
[3,5,1,6,2,0,8,null,null,7,4]

p =
5

## 7. Course Schedule



### 207. Course Schedule

Medium · Topics · Companies · Hint

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`.

**Example 1:**

```
Input: numCourses = 2, prerequisites = [[1,0]]
Output: true
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0. So it is possible.
```

**Example 2:**

```
Input: numCourses = 2, prerequisites = [[1,0],[0,1]]
Output: false
Explanation: There are a total of 2 courses to take.
To take course 1 you should have finished course 0, and to take course 0 you
should also have finished course 1. So it is impossible.
```

```cpp
class Solution {
public:
    bool canFinish(int numCourses, vector<vector<int>>& prerequisites) {
        vector<vector<int>> adj(numCourses);
        vector<int> inDegree(numCourses, 0);

        for (auto& p : prerequisites) {
            adj[p[1]].push_back(p[0]);
            inDegree[p[0]]++;
        }

        queue<int> q;
        for (int i = 0; i < numCourses; ++i) {
```

Accepted Runtime: 0 ms

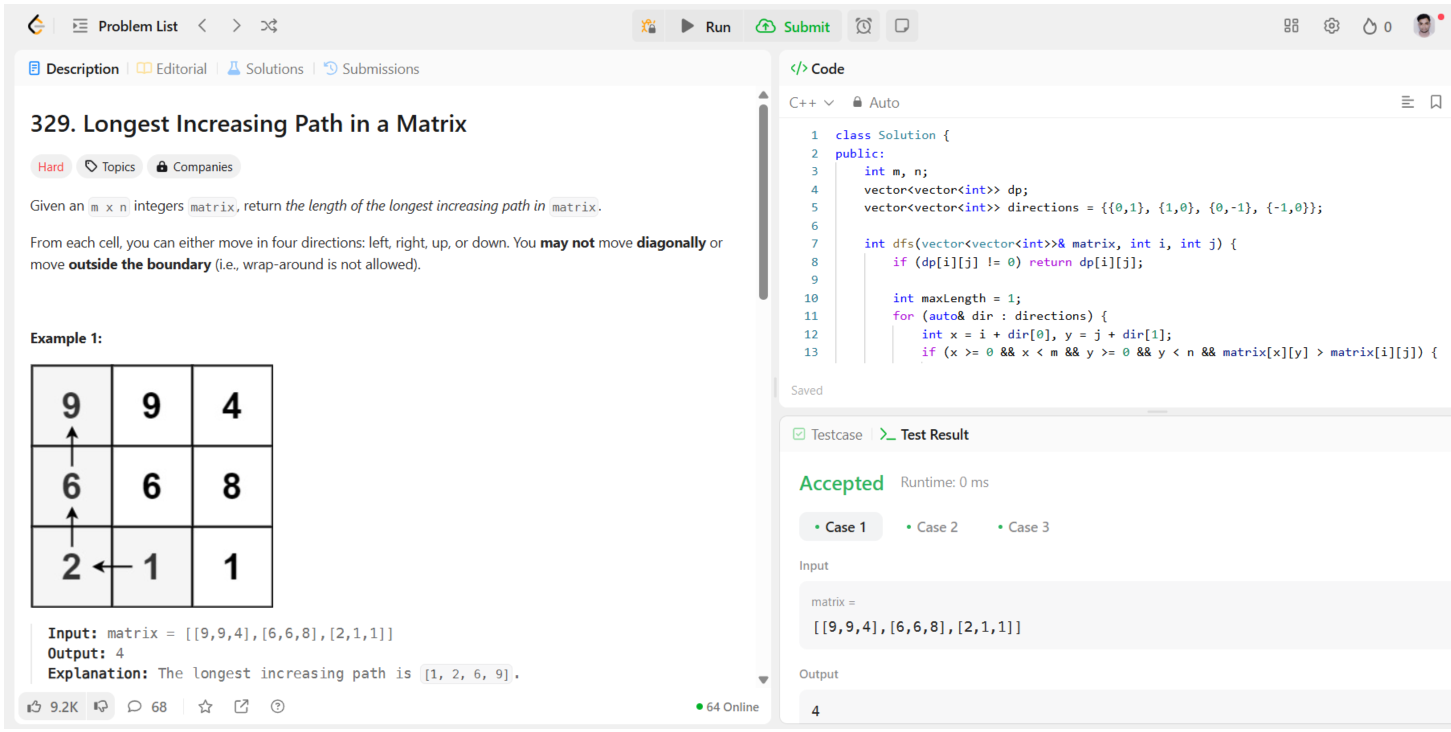
Case 1 · Case 2

Input

```
numCourses =
2
```

```
prerequisites =
[[1,0]]
```

## 8. Longest Increasing Path in a Matrix



**329. Longest Increasing Path in a Matrix**

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).

**Example 1:**

```
Input: matrix = [[9,9,4],[6,6,8],[2,1,1]]
Output: 4
Explanation: The longest increasing path is [1, 2, 6, 9].
```

```
class Solution {
public:
    int m, n;
    vector<vector<int>> dp;
    vector<vector<int>> directions = {{0,1}, {1,0}, {0,-1}, {-1,0}};

    int dfs(vector<vector<int>>& matrix, int i, int j) {
        if (dp[i][j] != 0) return dp[i][j];

        int maxLength = 1;
        for (auto& dir : directions) {
            int x = i + dir[0], y = j + dir[1];
            if (x >= 0 && x < m && y >= 0 && y < n && matrix[x][y] > matrix[i][j]) {
```

Accepted Runtime: 0 ms

Input

matrix =

```
[[9,9,4],[6,6,8],[2,1,1]]
```

Output

```
4
```