EXPERIMENT 9

Name: Vatsala Singh

UID: 22BCS10028

Section/Group: 605-A

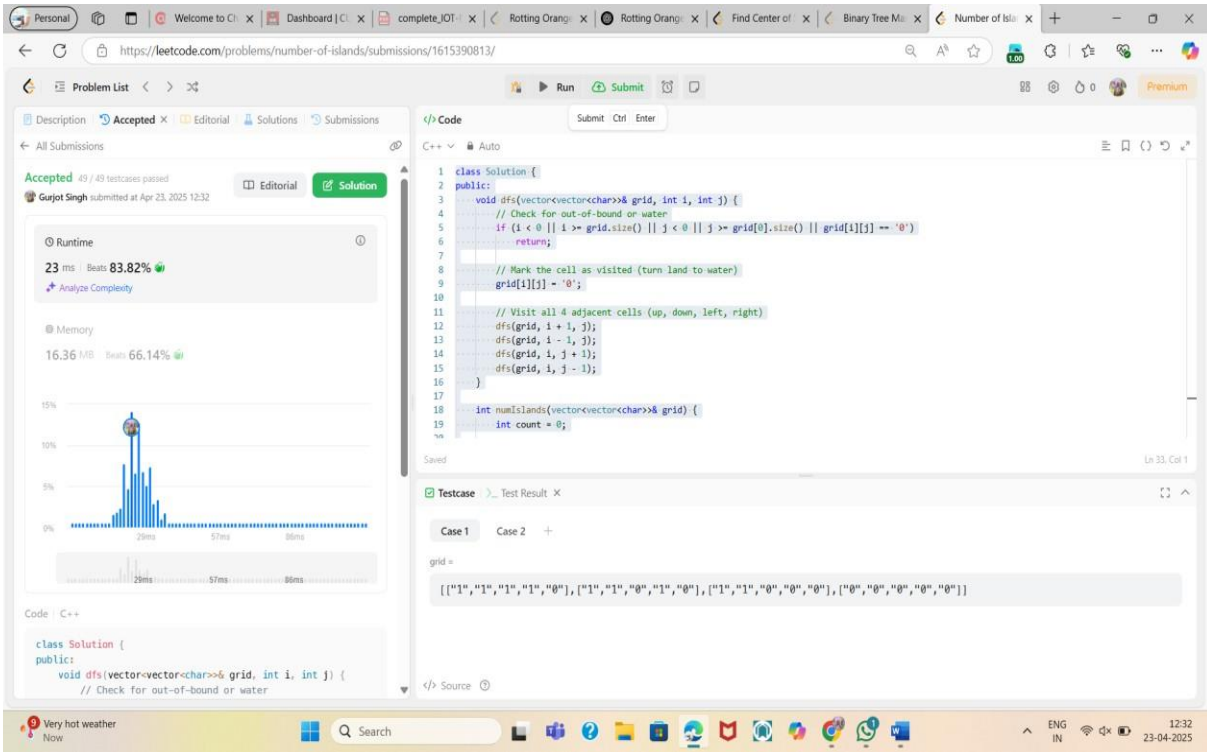
Ques- Number of Islands

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.

```
class Solution { public:

void dfs(vector<vector<char>>& grid, int i, int j) {

// Check for out-of-bound or water

if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || grid[i][j] == '0')

return;

// Mark the cell as visited (turn land to water)

grid[i][j] = '0';

// Visit all 4 adjacent cells (up, down, left, right)

dfs(grid, i + 1, j);        dfs(grid, i - 1, j);

dfs(grid, i, j + 1);        dfs(grid, i, j - 1);

}


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

int count = 0;


for (int i = 0; i < grid.size(); i++) {

for (int j = 0; j < grid[0].size(); j++) {

if (grid[i][j] == '1') {
```

```
count++;            // Found an island
dfs(grid, i, j);    // Mark all connected land
}
}
}

return count;
}
};
```



https://leetcode.com/problems/number-of-islands/submissions/1615390813/

Accepted 49 / 49 testcases passed

Gurjot Singh submitted at Apr 23, 2025 12:32

Runtime 23 ms | Beats 83.82%

Memory 16.36 MB | Beats 66.14%

Code | C++

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int i, int j) {
        // Check for out-of-bound or water
```

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int i, int j) {
        // Check for out-of-bound or water
        if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || grid[i][j] == '0')
            return;

        // Mark the cell as visited (turn land to water)
        grid[i][j] = '0';

        // Visit all 4 adjacent cells (up, down, left, right)
        dfs(grid, i + 1, j);
        dfs(grid, i - 1, j);
        dfs(grid, i, j + 1);
        dfs(grid, i, j - 1);
    }

    int numIslands(vector<vector<char>>& grid) {
        int count = 0;
```

grid =

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

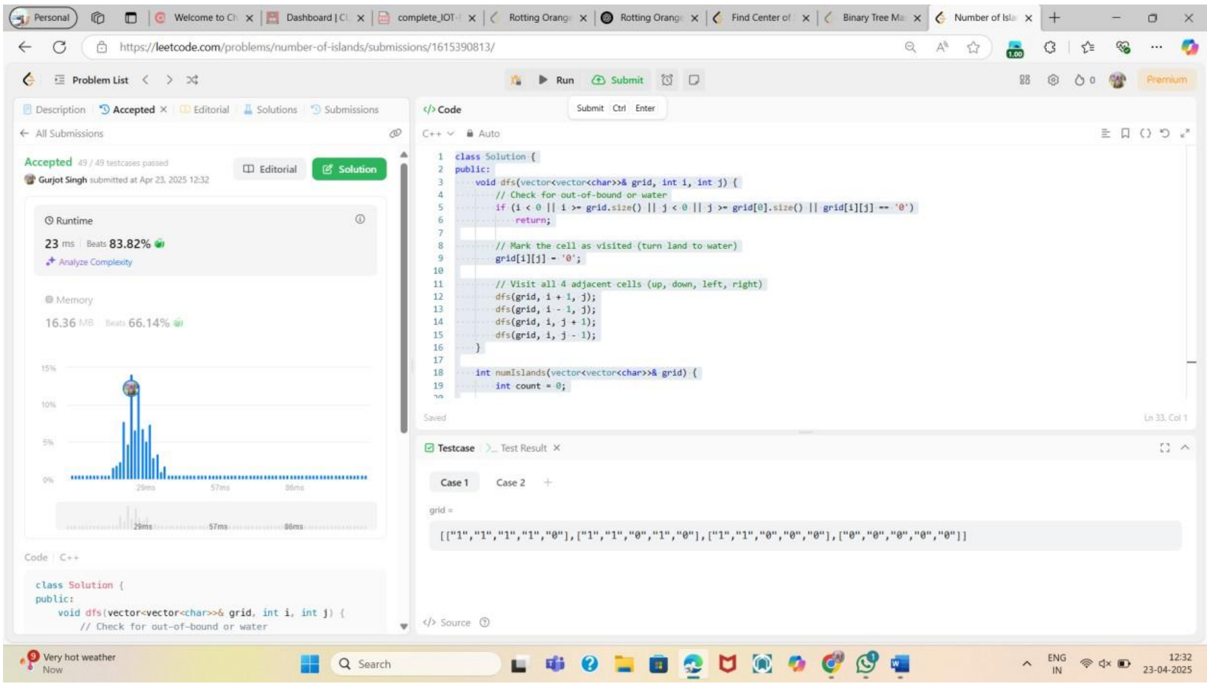
Ques- Course Schedule Problem:

There are a total of numCourses courses you have to take, labeled from 0 to numCourses - 1.

Some courses may have prerequisites, given as a list of prerequisites where prerequisites[i] = [a, b] means you must take course b before course a.

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

```
#include <iostream>

#include <vector>

#include <queue>

using namespace std;


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++) {        if

(indegree[i] == 0) q.push(i);

}


int count = 0;

while (!q.empty()) {

int curr = q.front();
q.pop();

count++;


for (int neighbor : adj[curr]) {

indegree[neighbor]--;             if

(indegree[neighbor] == 0)

q.push(neighbor);
}

}


return count == numCourses;

}


int main() {     int numCourses = 2;

vector<vector<int>> prerequisites = {{1, 0}};


cout << (canFinish(numCourses, prerequisites) ? "true" : "false") << endl;

return 0;

}
```



https://leetcode.com/problems/number-of-islands/submissions/1615390813/

Accepted 49 / 49 testcases passed

Gurjot Singh submitted at Apr 23, 2025 12:32

Runtime

23 ms | Beats 83.82%

Memory

16.36 MB | Beats 66.14%

Code | C++

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int i, int j) {
        // Check for out-of-bound or water
        if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || grid[i][j] == '0')
            return;

        // Mark the cell as visited (turn land to water)
        grid[i][j] = '0';

        // Visit all 4 adjacent cells (up, down, left, right)
        dfs(grid, i + 1, j);
        dfs(grid, i - 1, j);
        dfs(grid, i, j + 1);
        dfs(grid, i, j - 1);
    }

    int numIslands(vector<vector<char>>& grid) {
        int count = 0;
```

Testcase | Test Result

Case 1 | Case 2

grid =

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

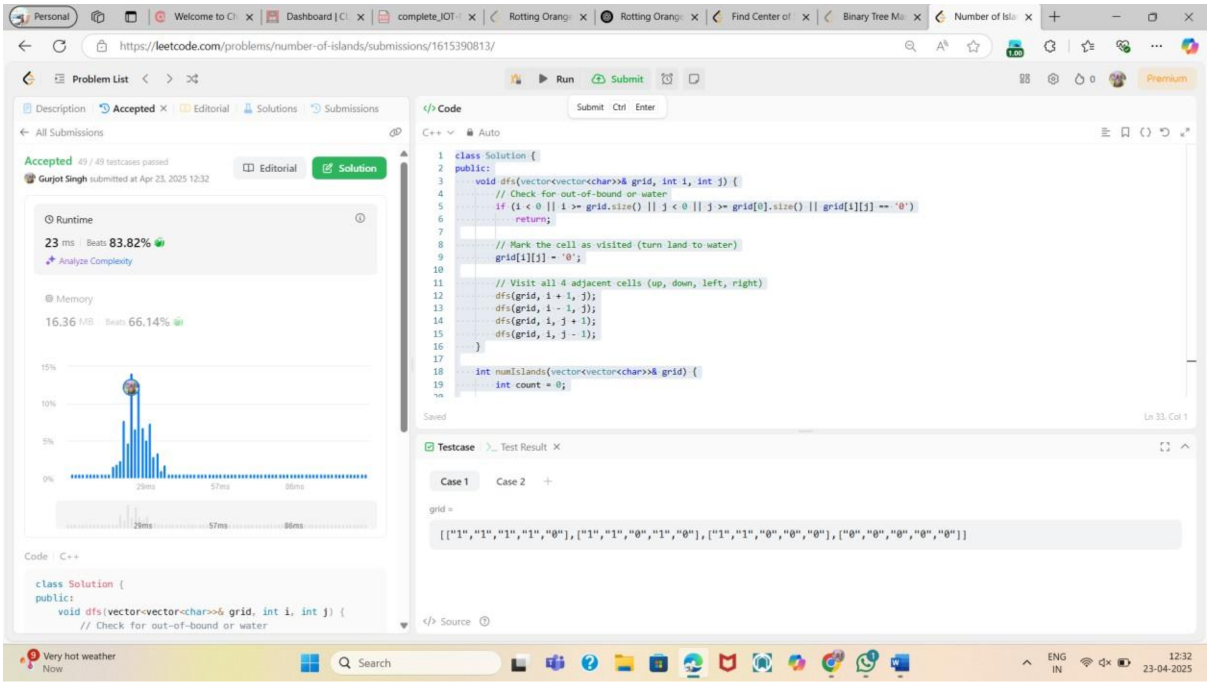
Ques- Longest Increasing Path in a Matrix Problem:

Given an m x n integers matrix, return the length of the longest increasing path in the matrix.

From each cell, you can move in 4 directions (up, down, left, right). You may not move diagonally or move outside the boundary.

```
cpp

CopyEdit

#include <iostream>

#include <vector>

using namespace

std;


class Solution { public:     int

longestIncreasingPath(vector<vector<int>>& matrix) {

int m = matrix.size();        int n = matrix[0].size();
```

```
vector<vector<int>> dp(m, vector<int>(n, 0));         int
maxLen = 0;

for (int i = 0; i < m; i++) {             for (int j = 0; j <
n; j++) {                 maxLen = max(maxLen,
dfs(matrix, dp, i, j));
}
}
return maxLen;
}

private:     vector<int> dir = {-1,
0, 1, 0, -1};

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

int maxPath = 1;         for
(int d = 0; d < 4; d++) {
int x = i + dir[d];             int
y = j + dir[d + 1];

if (x >= 0 && x < matrix.size() &&                 y >= 0 &&
y < matrix[0].size() &&                 matrix[x][y] >
matrix[i][j]) {                 maxPath = max(maxPath, 1 +
dfs(matrix, dp, x, y));
}
}

dp[i][j] = maxPath;

return maxPath;

}

};


int main() {     Solution sol;

vector<vector<int>> matrix =

{

{9, 9, 4},

{6, 6, 8},

{2, 1, 1}

};


cout << sol.longestIncreasingPath(matrix) << endl;

return 0;

}
```



Accepted 49 / 49 testcases passed

Gurjot Singh submitted at Apr 23, 2025 12:32

Runtime

23 ms | Beats 83.82%

Memory

16.36 MB | Beats 66.14%

Code | C++

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int i, int j) {
        // Check for out-of-bound or water
```

```
class Solution {
public:
    void dfs(vector<vector<char>>& grid, int i, int j) {
        // Check for out-of-bound or water
        if (i < 0 || i >= grid.size() || j < 0 || j >= grid[0].size() || grid[i][j] == '0')
            return;

        // Mark the cell as visited (turn land to water)
        grid[i][j] = '0';

        // Visit all 4 adjacent cells (up, down, left, right)
        dfs(grid, i + 1, j);
        dfs(grid, i - 1, j);
        dfs(grid, i, j + 1);
        dfs(grid, i, j - 1);
    }

    int numIslands(vector<vector<char>>& grid) {
        int count = 0;
```

grid =

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