Experiment 8 A

Student Name: PARDEEP SINGH **UID: 22BCS16692**

Branch: CSE **Section/Group: Ntpp 602-A**

Semester: 6TH **Date of Performance:13/03/25**

Subject Name: AP Lab-2 **Subject Code: 22CSH-352**

1. TITLE:

Number of Islands

2. AIM:

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.

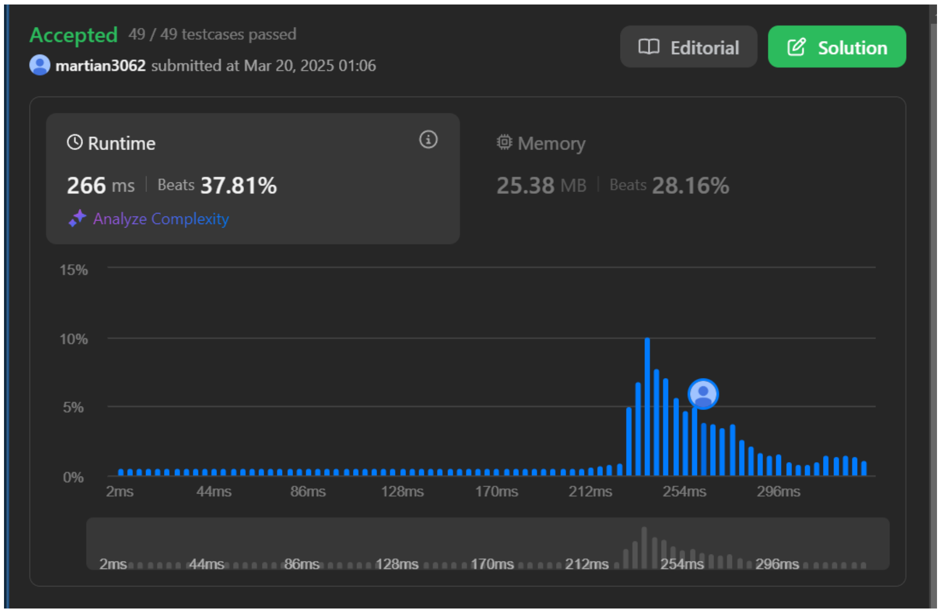
3. Algorithm

- Check if the grid is empty, return 0 if true.
- Define a BFS function to explore connected land ('1') and mark visited cells.
- Initialize a set visit to track visited cells and a variable count to count the number of islands.
- Iterate through each cell in the grid and start BFS if the cell is land and not visited.
- Mark all connected land during BFS traversal and increment the island count.
- Return the island count after checking all cells in the grid.

Implemetation/Code

```
class Solution:
    def numIslands(self, grid: List[List[str]]) -> int:
        if not grid:
            return 0
        def bfs(r, c):
            q = deque()
            visit.add((r, c))
            q.append((r, c))
            while q:
                row, col = q.popleft()
                directions = [[1, 0], [-1, 0], [0, 1], [0, -1]]
                for dr, dc in directions:
                    nr, nc = row + dr, col + dc
                    if 0 <= nr < rows and 0 <= nc < cols and grid[nr][nc] == '1' and (nr, nc) not in visit:
                        q.append((nr, nc))
                        visit.add((nr, nc))
        count = 0
        rows = len(grid)
        cols = len(grid[0])
        visit = set()
        for r in range(rows):
            for c in range(cols):
                if grid[r][c] == '1' and (r, c) not in visit:
                    bfs(r, c)
                    count += 1
        return count
```

Output


Accepted 49 / 49 testcases passed
martian3062 submitted at Mar 20, 2025 01:06
Runtime
266 ms | Beats 37.81%
Memory
25.38 MB | Beats 28.16%

Time Complexity : O(m*n)

Space Complexity : O(m*n)

Learning Outcomes:-

- Learn how to apply BFS for exploring connected components (like islands) in a grid.
- Understand how to use a set for efficiently tracking visited cells and prevent reprocessing during traversal.

Experiment 8 B

Student Name: PARDEEP SINGH	**UID: 22BCS16692**
Branch: CSE	**Section/Group: Ntpp 602-A**
Semester: 6TH	**Date of Performance:13/03/25**
Subject Name: AP Lab-2	**Subject Code: 22CSH-352**

1. TITLE:

Course Schedule

2. AIM:

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.

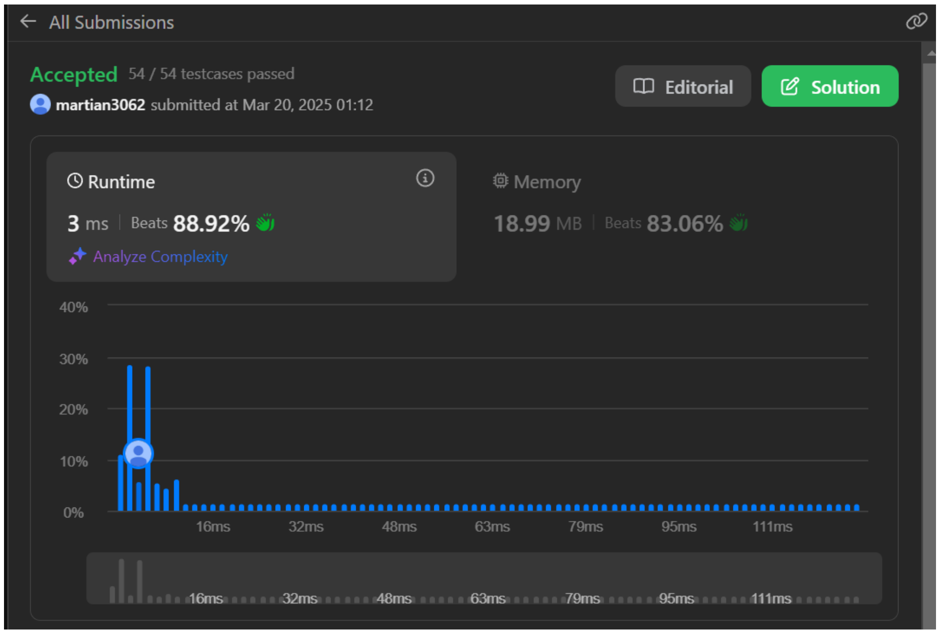
3. Algorithm

1. Initialize graph and inDegree to represent course dependencies and the number of prerequisites for each course.
2. Build the graph from the prerequisites list and update the in-degree of each course.
3. Add courses with no prerequisites (in-degree 0) to the queue.
4. Process each course in the queue, reducing the in-degree of dependent courses.
5. Add dependent courses to the queue when their in-degree becomes 0.
6. Return True if all courses are processed (i.e., no cycle); otherwise, return False.

Implemetation/Code:

```
class Solution:
from collections import deque
class Solution:
    def canFinish(self, numCourses: int, prerequisites: List[List[int]]) -> bool:
        graph = [[] for _ in range(numCourses)]
        inDegree = [0] * numCourses
        queue = deque([])
        processedCourses = 0
        for pre in prerequisites:
            [course, prereq] = pre
            graph[prereq].append(course)
            inDegree[course] += 1
        for i in range(numCourses):
            if inDegree[i] == 0:
                queue.append(i)
        while queue:
            course = queue.popleft()
            processedCourses +
            for nextCourse in graph[course]:
                inDegree[nextCourse] -= 1
                if (inDegree[nextCourse] == 0):
                    queue.append(nextCourse)
        return processedCourses == numCourses
```

Output



Accepted 54 / 54 testcases passed
martian3062 submitted at Mar 20, 2025 01:12

3 ms | Beats 88.92%

18.99 MB | Beats 83.06%

Time Complexity : O(V+E)

Space Complexity : O(V+E)

Learning Outcomes:-

- Understand how to detect cycles in a directed graph using topological sorting (Kahn's algorithm).
- Learn how to represent and traverse a graph using adjacency lists and in-degree arrays for efficient course scheduling problems.