Problem: Number of Islands

You are given a 2D binary grid representing a map where:

- '1' represents land,
- '0' represents water.

The goal is to count the number of islands. An island is formed by connecting adjacent lands either horizontally or vertically. Diagonal connections do not count. The grid is surrounded by water, and all four edges are assumed to be water as well.

Input:

- ***A 2D binary grid grid of size m x n, where:***
  - m == grid.length: the number of rows,
  - n == grid[i].length: the number of columns.
- ***Each element of the grid is either:***
  - '0' (representing water), or
  - '1' (representing land).

Output:

- Return the number of islands present in the grid.

Approach:

1. Iterate Through the Grid:

- The basic idea is to go through each cell in the grid. If the cell contains '1' (land), it means that we have discovered a new island. At that point, we need to explore the entire island, marking all the connected land cells to ensure the entire island is counted only once.

2. Depth-First Search (DFS) or Breadth-First Search (BFS):

- To explore an island (i.e., connected land cells), we can use either DFS or BFS. Both methods can be applied to traverse all connected '1's:
  - ***DFS:*** Recursively visits all the adjacent land cells (up, down, left, right) from a given starting point.
  - ***BFS:*** Uses a queue to explore the adjacent cells layer by layer, ensuring that all connected land cells are processed.
- In this problem, DFS is commonly used because it can be easily implemented recursively.

3. Marking Cells as Visited:

- Once an island is discovered, all connected '1's are part of the same island. To prevent re-counting the same island multiple times, we need to mark the land cells as "visited." This can be done by converting '1' (land) to '0' (water), indicating that the cell has already been processed. This avoids the need for additional space to track visited cells.

4. Island Counting:

- Each time we encounter a new unvisited land cell ('1'), we increase the island count. After marking all connected cells of the current island as visited, the process continues for the rest of the grid.

Example 1:

Input: grid = [

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

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

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

["0","0","0","0","0"]

]

Step-by-Step Process:

1. Start at grid[0][0], which is '1'. This represents the start of an island.
2. Perform DFS (or BFS) from this point, visiting all connected '1's (land) and marking them as '0' (water).
3. After marking all connected cells, we continue traversing the grid.
4. Since all '1's have been marked, we only find 1 island.

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"]

]

Step-by-Step Process:

1. Start at grid[0][0], which is '1'. Perform DFS to mark all connected cells.
2. Continue traversing the grid and encounter grid[2][2], which is another unvisited '1'. This represents a new island.
3. Perform DFS from grid[2][2] and mark all connected '1's.
4. Similarly, encounter grid[3][3] which represents the start of another island.
5. Perform DFS from grid[3][3], marking grid[3][4] as part of the same island.

Output: 3

Constraints:

- m == grid.length: The number of rows in the grid.
- n == grid[i].length: The number of columns in the grid.
- 1 <= m, n <= 300: The dimensions of the grid can be as large as 300x300.
- grid[i][j] is '0' or '1': Each cell is either water ('0') or land ('1').

Key Concepts:

1. Grid Representation:

- The grid is a 2D list where each cell is either '0' (water) or '1' (land).
- Cells are connected horizontally or vertically to form an island.

2. DFS/BFS for Exploration:

- DFS (Depth-First Search) or BFS (Breadth-First Search) can be used to traverse all connected land cells starting from a given cell.
- DFS is implemented recursively, exploring all directions from the starting point until all connected cells are visited.

3. Marking Cells as Visited:

- To avoid revisiting the same island, once a cell is visited, it is marked as '0' (water).
- This prevents counting the same island multiple times.

4. Counting Islands:

- Each time we find a new land cell ('1'), we increment the island count.
- The DFS/BFS function is called to mark all cells in the current island as visited.

Time and Space Complexity:

Time Complexity: O(m * n)

- Every cell is visited once during the grid traversal, and the DFS/BFS function explores all connected land cells, ensuring that each cell is processed in constant time.

Space Complexity:

- In the worst case, where the grid is entirely filled with land, the recursive DFS call stack can reach O(m * n) in depth.
- For BFS, the queue will store the same number of cells in the worst case, leading to a similar space complexity.

Edge Cases:

1. **Empty Grid:** If the input grid is empty (grid = []), the function should immediately return 0 as there are no islands.
2. **Single Row/Column:** For grids that are a single row or a single column, the algorithm should handle these smaller cases efficiently.
3. **All Water:** A grid that contains only '0's (all water) should return 0, as no islands are present.
4. **All Land:** A grid that contains only '1's (all land) should return 1, as the entire grid is one large island.

Conclusion:

- This problem can be efficiently solved by treating the grid as a graph and using DFS/BFS to explore connected components (islands). By marking cells as visited and exploring in all four directions (up, down, left, right), we can count the number of distinct islands in the grid.