Number of Islands

Difficulty	Medium
Category	Graph
Question	https://leetcode.com/problems/number-of-islands/
Solution	https://youtu.be/pV2kpPD66nE
Status	Done

Question

> 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

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"]
]
Output: 3
```

Idea

DFS on each cell with value 1, if there’s connection of 1s, the DFS will not terminate; If there’s a 1 that is surrounded with 0, current DFS will terminate. Use visited array to keep track of visited cells. Number of independent DFS tracks that starts from an unvisited 1 cell is an independent island

Solution

```
class Solution:

    def numIslands(self, grid: List[List[str]]) -> int:
        if not grid or not grid[0]:
            return 0

        islands = 0  # Initialize the count of islands to 0
        visit = set()  # Create a set to keep track of visited cells
        rows, cols = len(grid), len(grid[0])

        def dfs(r, c):
            if (
                r not in range(rows)
                or c not in range(cols)
                or grid[r][c] == "0"
                or (r, c) in visit
            ):
                return

            visit.add((r, c))  # Mark the cell as visited
            directions = [[0, 1], [0, -1], [1, 0], [-1, 0]]

            # Explore neighboring cells in all four directions
```

```
            for dr, dc in directions:
                dfs(r + dr, c + dc)

        for r in range(rows):
            for c in range(cols):
                if grid[r][c] == "1" and (r, c) not in visit:
                    islands += 1  # Found a new island, increment the count
                    dfs(r, c)  # Start DFS to explore the island

        return islands
```

Explanation

1. The `numIslands` function takes a 2D grid as input and counts the number of islands within it. An island is defined as a group of adjacent "1"s in the grid.
2. First, it checks if the grid is empty or has no columns, in which case there are no islands, and it returns 0.
3. It initializes `islands` to keep track of the number of islands found and a `visit` set to track visited cells. The `rows` and `cols` variables store the dimensions of the grid.
4. The `dfs` function is a recursive depth-first search function that explores an island starting from a given cell `(r, c)`.
5. It checks if `(r, c)` is outside the grid boundaries, if the cell contains a "0", or if it has already been visited. If any of these conditions is met, the function returns, effectively ending the recursive exploration.
6. If `(r, c)` is a valid cell to explore, it is added to the `visit` set, and the code defines four possible directions to move: up,

down, left, and right.

7. The `for` loop in the `dfs` function explores each direction and recursively calls `dfs` on neighboring cells.
8. The main loop in the `numIslands` function iterates through all cells in the grid. If a cell contains "1" and has not been visited, it means the start of a new island. The `islands` count is incremented, and the DFS process is initiated to explore and mark the entire island.
9. Finally, the code returns the count of islands found in the grid.

The given code is an efficient way to count the number of islands in a grid using depth-first search (DFS). It has a time complexity of O(M * N), where M is the number of rows, and N is the number of columns in the grid, and it uses a set to keep track of visited cells to avoid reprocessing them.