The Problem:

"Given a `grid` of `'1'`s (land) and `'0'`s (water), count the number of distinct islands (connected groups of `'1'`s). Connected means horizontally or vertically adjacent."

Input Example:

css
Copier le code

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

Output: `3`

Q1: Why is this an interview favorite?

1. Teaches **graph traversal** → BFS, DFS, or Union-Find.
2. Tests edge case handling (isolated cells, edge cells, etc.).
3. Prepares you for **connected component problems**.

Q2: Why is this a graph problem in disguise?

1. The grid = graph.
2. `'1'`s are **nodes**, connections = edges.
3. Counting islands = counting connected components in an implicit graph.

Q3: What are the key approaches to solve it?

1. **DFS:** Treat each `'1'` as the root and mark all connected nodes.
2. **BFS:** Similar to DFS but uses a queue.
3. **Union-Find:** Merge connected `'1'`s and count components.

Q4: Walk me through the DFS solution.

Here’s the process:

1. Iterate through every cell.
2. When a `'1'` is found → Start a DFS and mark all connected `'1'`s as `'0'` (visited).
3. Increment the island count.

DFS Code

python

Copier le code

```python
def num_islands(grid):
    if not grid or not grid[0]:
        return 0

    def dfs(grid, r, c):
        # Base case: Out of bounds or water
        if r < 0 or c < 0 or r >= len(grid) or c >= len(grid[0]) or grid[r][c] == '0':
            return
        # Mark as visited
        grid[r][c] = '0'
        # Visit neighbors
        dfs(grid, r + 1, c)  # Down
        dfs(grid, r - 1, c)  # Up
        dfs(grid, r, c + 1)  # Right
        dfs(grid, r, c - 1)  # Left

    count = 0
    for r in range(len(grid)):
        for c in range(len(grid[0])):
            if grid[r][c] == '1':  # New island found
                count += 1
                dfs(grid, r, c)

    return count

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

```
]
print(num_islands(grid))  # Output: 3
```

Q5: BFS—How does it work?

- Similar logic to DFS, but use a **queue** for traversal.
- Explore neighbors level by level.

BFS Code

python

Copier le code

```
from collections import deque

def num_islands_bfs(grid):
    if not grid or not grid[0]:
        return 0

    def bfs(grid, r, c):
        queue = deque([(r, c)])
        while queue:
            x, y = queue.popleft()
            for dx, dy in [(1, 0), (-1, 0), (0, 1), (0, -1)]:
                nx, ny = x + dx, y + dy
                if 0 <= nx < len(grid) and 0 <= ny < len(grid[0])
and grid[nx][ny] == '1':
                    grid[nx][ny] = '0'  # Mark visited
                    queue.append((nx, ny))

    count = 0
    for r in range(len(grid)):
        for c in range(len(grid[0])):
            if grid[r][c] == '1':
                count += 1
                bfs(grid, r, c)

    return count

# Example
print(num_islands_bfs(grid))  # Output: 3
```

Q6: Union-Find Method

- Treat each cell as a node in a **disjoint set.**
- Union connected land cells.
- Count the number of distinct sets at the end.

Union-Find has better performance when the grid is dense, but it's a bit more complex to implement.

Q7: Edge Cases to Keep in Mind

1. Empty grid: `grid = []` or `grid = [[]]`. Return `0`.
2. No land: `grid = [["0", "0"]]`. Return `0`.
3. All land: `grid = [["1", "1"], ["1", "1"]]`. Return `1`.
4. Land on edges and corners.

Q8: Time and Space Complexity

DFS/BFS

- **Time Complexity: O(R * C)** → Every cell is visited once.
- **Space Complexity: O(R * C)** → Stack/Queue size (worst case all land).

Union-Find

- **Time Complexity:** Close to **O(R * C)** due to path compression.
- **Space Complexity: O(R * C)** → Parent array.

Facts/Tweets to Drop Knowledge

1. "Every grid problem is secretly a graph problem—recognize the structure, and BFS/DFS will save the day." 🧠✨
2. "DFS vs. BFS? Choose what you're comfortable with. Both are beastly at exploring connected components." 🦾
3. "Union-Find feels fancy but shines for **dense grids**. BFS/DFS dominate sparse grids." 🎯
4. "Grid traversal = adjacency + marking visited. Nail this pattern for graph-related interviews."