01 Matrix - Documentation

📑 Table of Contents

1. Problem Statement

You are given an m x n binary matrix mat containing only 0s and 1s.Return a matrix where each cell containing a 1 is replaced with the distance to the nearest 0.

The distance between two adjacent cells (up, down, left, right) is 1.

➕ Constraints:

- 1 <= m, n <= 10^4
- 1 <= m * n <= 10^4
- mat[i][j] is either 0 or 1
- There is at least one 0 in mat

2. 💡 Intuition

If we start a BFS from every 1, it would be inefficient (Time: $O((m*n)^2)$).Instead, we start BFS from all 0s at once, and expand to their neighbors, marking the shortest distance to a 0.

3. 🔍 Key Observations

- All cells with 0 are already at distance 0.
- A 1 cell's minimum distance to 0 is the shortest path to any 0, traversing through valid 4-directional neighbors.
- BFS guarantees shortest path traversal in unweighted graphs, so it fits perfectly here.

4. 🚀 Approach

✅ Breadth-First Search (BFS) from all 0s:

- Initialize a queue with all 0 positions.
- Set all 1s to -1 to indicate they are unvisited.
- Run BFS from the queue:
  - For each cell popped, explore its 4 neighbors.
  - If a neighbor is -1, assign it current_distance + 1 and push it to the queue.

5. ⚠ Edge Cases

- Matrix already contains only 0s → output will be the same matrix.
- Matrix contains 1s but no 0 → not possible per constraints.
- Matrix is 1x1 → either 0 or 1 with a single cell.

6. 📊 Complexity Analysis

⏱ Time Complexity

- O(m * n) — each cell is visited at most once.

💾 Space Complexity

- O(m * n) — for the queue and visited updates (modifying the matrix in-place).

7. 🔁 **Alternative Approaches**

🔷 Dynamic Programming (2-pass):

- First pass: top-left to bottom-right.
- Second pass: bottom-right to top-left.
- For each 1, update using minimum distance from neighbors.

🔷 Time: O(m*n)

🔷 Space: O(1) extra if in-place

⚠ DP is harder to debug and less intuitive than BFS in this context.

8. □ **Test Cases**

✅ Test Case 1:

Input: [[0,0,0],[0,1,0],[0,0,0]]
Output: [[0,0,0],[0,1,0],[0,0,0]]

✅ Test Case 2:

Input: [[0,0,0],[0,1,0],[1,1,1]]
Output: [[0,0,0],[0,1,0],[1,2,1]]

✅ Test Case 3:

Input: [[1,1,1],[1,0,1],[1,1,1]]
Output: [[2,1,2],[1,0,1],[2,1,2]]

✅ Test Case 4:

Input: [[0]]
Output: [[0]]

9. □ Final Thoughts

- BFS from all 0s ensures a clean and optimal solution.
- Very scalable and handles large matrices efficiently.
- Alternative 2-pass DP can be explored if in-place and no queue usage is preferred.