Multi-Source BFS Explanation

prepare DSA notes for this program. Mention pattern, add code with comments.

```
class Solution {
    public int[][] updateMatrix(int[][] mat) {
        int m = mat.length;
        int n = mat[0].length;
        int[][] dist = new int[m][n];
        Queue<int[]> q = new LinkedList<>();

        for(int i=0; i<m; i++){
            for(int j=0; j<n; j++){
                if(mat[i][j] == 0){
                    dist[i][j] = 0;
                    q.offer(new int[]{i, j});
                }
                else{
                    dist[i][j] = Integer.MAX_VALUE;
                }
            }
        }

        int[][] directions = {{-1,0}, {1,0}, {0,-1}, {0,1}};

        while(!q.isEmpty()){
            int[] cell = q.poll();
            int row = cell[0];
            int col = cell[1];

            for(int[] dir : directions){
                int newRow = row + dir[0];
                int newCol = col + dir[1];

                if(newRow >= 0 && newRow < m && newCol >= 0 && newCol < n){
                    if(dist[newRow][newCol] > dist[row][col] + 1){
                        dist[newRow][newCol] = dist[row][col] + 1;
                        q.offer(new int[]{newRow, newCol});
                    }
                }
            }
        }
        return dist;
    }
}
```

Here are the **DSA notes** for the given program, including the **pattern**, **explanation**, and **Java code with comments**.

✅ Problem: 01 Matrix (Leetcode 542)

🔍 Pattern:

Multi-Source Breadth-First Search (BFS)

- This is a **BFS on Grid** problem where **all cells with value 0** act as sources simultaneously.
- You expand from all zeros to compute the minimum distance to the nearest 0 for every cell with value 1.

📌 Intuition:

- Cells with value `0` are at distance 0 from themselves.
- Cells with value `1` need to find the shortest path to a `0`.
- We use a **BFS** starting from **all 0s**. This way, the first time we reach a 1, we know it's the shortest path from a 0.

🧠 Key Concepts:

- Multi-source BFS.
- Traverse grid in 4 directions.
- Keep a distance matrix updated.
- BFS ensures shortest path since it explores level by level.

📦 Space Complexity:

- `O(m * n)` for the distance matrix and queue.

⏱ Time Complexity:

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

✅ Java Code with Comments:

```java
class Solution {
    public int[][] updateMatrix(int[][] mat) {
        int m = mat.length;        // number of rows
        int n = mat[0].length;     // number of columns
        int[][] dist = new int[m][n]; // stores final distance values
        Queue<int[]> q = new LinkedList<>(); // BFS queue

        // Step 1: Initialize queue with all 0s, set 1s as infinity
        for(int i = 0; i < m; i++) {
            for(int j = 0; j < n; j++) {
                if(mat[i][j] == 0) {
                    dist[i][j] = 0;         // distance to itself is 0
                    q.offer(new int[]{i, j}); // enqueue all 0s
                } else {
                    dist[i][j] = Integer.MAX_VALUE; // initialize 1s as max
                }
            }
        }

        // Step 2: Directions array for moving up, down, left, right
        int[][] directions = {{-1,0}, {1,0}, {0,-1}, {0,1}};

        // Step 3: BFS traversal
        while(!q.isEmpty()) {
            int[] cell = q.poll();     // get current cell
            int row = cell[0];
            int col = cell[1];
            for(int[] dir : directions) {
                int newRow = row + dir[0];
                int newCol = col + dir[1];

                // check bounds
                if(newRow >= 0 && newRow < m && newCol >= 0 && newCol < n) {
                    // check if we found a shorter path
                    if(dist[newRow][newCol] > dist[row][col] + 1) {
                        dist[newRow][newCol] = dist[row][col] + 1;
                        q.offer(new int[]{newRow, newCol}); // enqueue updated cell
                    }
                }
            }
        }

        return dist;
    }
}
```

🧩 Summary:

Aspect	Details
Pattern	Multi-source BFS on grid
Input	Matrix with 0s and 1s
Output	Matrix of distances to nearest 0
Key Approach	Start BFS from all 0s
Direction	4-way (up, down, left, right)

Let me know if you want a dry run or want this written as a PDF/Markdown for revision.