<https://leetcode.ca/all/286.html>

You are given a *m x n* 2D grid initialized with these three possible values.

* -1 - A wall or an obstacle.
* 0 - A gate.
* INF - Infinity means an empty room. We use the value 231 - 1 = 2147483647 to represent INF as you may assume that the distance to a gate is less than 2147483647.

Fill each empty room with the distance to its *nearest* gate. If it is impossible to reach a gate, it should be filled with INF.

Example:

Given the 2D grid:

INF -1 0 INF

INF INF INF -1

INF -1 INF -1

0 -1 INF INF

After running your function, the 2D grid should be:

3 -1 0 1

2 2 1 -1

1 -1 2 -1

0 -1 3 4

**Attempt 1: 2023-10-8**

**Solution 1: BFS + Level order traversal (10min, no extra space visited 2D array needed, just need + 1 based on current )**

class Solution {

public void wallsAndGates(int[][] rooms) {

Queue<int[]> q = new LinkedList<>();

int m = rooms.length;

int n = rooms[0].length;

for(int i = 0; i < m; i++) {

for(int j = 0; j < n; j++) {

if(rooms[i][j] == 0) {

q.offer(new int[] {i, j});

}

}

}

int[] dx = new int[] {0, 0, 1, -1};

int[] dy = new int[] {1, -1, 0, 0};

int level = 0;

while(!q.isEmpty()) {

int size = q.size();

for(int i = 0; i < size; i++) {

int[] cur = q.poll();

for(int k = 0; k < 4; k++) {

int new\_x = cur[0] + dx[k];

int new\_y = cur[1] + dy[k];

if(new\_x >= 0 && new\_x < m && new\_y >= 0 && new\_y < n && rooms[new\_x][new\_y] == Integer.MAX\_VALUE) {

rooms[new\_x][new\_y] = rooms[cur[0]][cur[1]] + 1;

q.offer(new int[] {new\_x, new\_y});

}

}

}

}

}

public static void main(String[] args) {

Solution so = new Solution();

int INF = Integer.MAX\_VALUE;

int[][] grid = new int[][] {{INF, -1, 0, INF}, {INF, INF, INF, -1}, {INF, -1, INF, -1}, {0, -1, INF, INF}};

so.wallsAndGates(grid);

System.out.println("done");

}

}

Time Complexity: O(M\*N)

Space Complexity: O(M\*N)