<https://leetcode.com/problems/rotting-oranges/>

You are given an m x n grid where each cell can have one of three values:

0representing an empty cell,

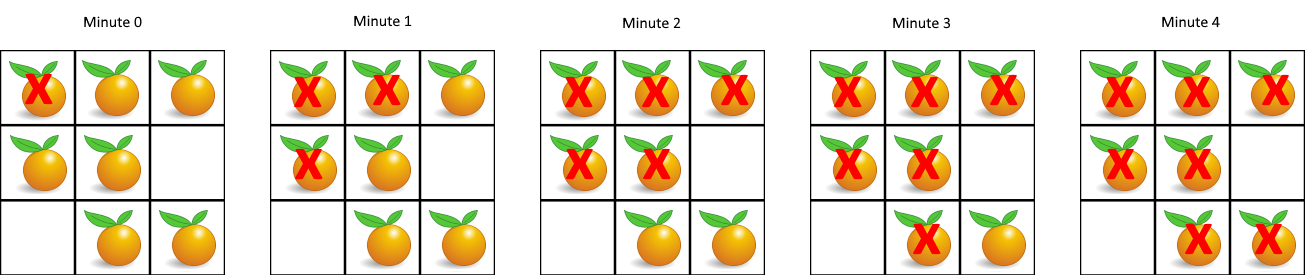
1representing a fresh orange, or

2representing a rotten orange.

Every minute, any fresh orange that is **4-directionally adjacent** to a rotten orange becomes rotten.

Return *the minimum number of minutes that must elapse until no cell has a fresh orange*. If *this is impossible, return*-1.

**Example 1:**



Input: grid = [[2,1,1],[1,1,0],[0,1,1]]

Output: 4

**Example 2:**

Input: grid = [[2,1,1],[0,1,1],[1,0,1]]

Output: -1

Explanation: The orange in the bottom left corner (row 2, column 0) is never rotten, because rotting only happens 4-directionally.

**Example 3:**

Input: grid = [[0,2]]

Output: 0

Explanation: Since there are already no fresh oranges at minute 0, the answer is just 0.

**Constraints:**

m == grid.length

n == grid[i].length

1 <= m, n <= 10

grid[i][j]is 0, 1, or 2.

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

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

class Solution {

    public int orangesRotting(int[][] grid) {

        int count = 0;

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

        int m = grid.length;

        int n = grid[0].length;

        for(int i = 0; i < m; i++) {

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

                if(grid[i][j] == 2) {

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

                }

                if(grid[i][j] == 1) {

                    count++;

                }

            }

        }

        // Corner case: Test out by input [[0]] or [[0, 2]], expect 0

        // Since there are already no fresh oranges at minute 0,

        // the answer is just 0.

        if(count == 0) {

            return 0;

        }

        // BFS level traversal, the 'level' required to empty queue

        // equal to minimum minutes required, initialize with -1 means

        // the 1st rotten orange pulled out of queue will be minute 0

        // since level = -1 will add 1 to 0

        int level = -1;

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

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

        while(!q.isEmpty()) {

            int size = q.size();

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

                int[] cur = q.poll();

                // Set to -1 means already visited

                grid[cur[0]][cur[1]] = -1;

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

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

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

                    && grid[new\_x][new\_y] == 1) {

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

                        // Set to -1 means already visited and must have this

                        // change to mark as visited before adding {new\_x, new\_y}

                        // into queue, otherwise if not mark visited and continue

                        // search in for loop on 4 directions will have chance to

                        // duplicate find same new cell equal to 1

                        grid[new\_x][new\_y] = -1;

                        count--;

                    }

                }

            }

            level++;

        }

        return count == 0 ? level : -1;

    }

}

**Refer to**

**The below way have extra space requirement for boolean visited 2D array**

// Refer to

// https://github.com/lampardchelsea/hello-world/blob/master/leetcode/BFS/WallsAndGates.java

class Solution {

    public int orangesRotting(int[][] grid) {

        int fresh\_orange = 0;

        boolean[][] visited = new boolean[grid.length][grid[0].length];

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

        for(int i = 0; i < grid.length; i++) {

            for(int j = 0; j < grid[0].length; j++) {

                if(grid[i][j] == 1) {

                    fresh\_orange++;

                } else if(grid[i][j] == 2) {

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

                    visited[i][j] = true;

                }

            }

        }

        if(fresh\_orange == 0) {

            return 0;

        }

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

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

        int time = 0;

        while(!q.isEmpty()) {

            if(fresh\_orange == 0) {

                return time;

            }

            time++;

            int size = q.size();

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

                int[] cur = q.poll();

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

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

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

                        visited[new\_x][new\_y] = true;

                        if(grid[new\_x][new\_y] == 1) {

                            grid[new\_x][new\_y] = 2;

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

                            fresh\_orange--;

                        }

                    }

                }

            }

        }

        return -1;

    }

}

**Refer to**

[L286.Walls and Gates (Ref.L994)](note://66821E6DBC3A440685B340EEF09645E5)

[L2101.Detonate the Maximum Bombs](note://WEB5cedd2cfffa728b8984907f37905ea3c)