# **Leetcode 994 – Rotting Oranges**

## Problem Understanding

You are given a 2D grid where:

* 0 = empty cell
* 1 = fresh orange
* 2 = rotten orange

Each **minute**, any fresh orange 4-directionally adjacent to a rotten one becomes rotten.

Your goal is to return the **minimum number of minutes** that must elapse until **no fresh orange remains**. If it's not possible to rot all fresh oranges, return -1.

## Java Solution with Custom Pair Class

class Pair {

int x, y;

Pair(int x, int y) {

this.x = x;

this.y = y;

}

}

class Solution {

public boolean boundarycheck(int[][] grid, int x, int y) {

return x >= 0 && y >= 0 && x < grid.length && y < grid[0].length;

}

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

int min = 0, fresh = 0;

Queue<Pair> q = new LinkedList<>();

// Count fresh oranges and queue rotten ones

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

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

if (grid[i][j] == 2)

q.add(new Pair(i, j));

if (grid[i][j] == 1)

fresh++;

}

}

if (fresh == 0) return 0;

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

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

// BFS minute by minute

while (!q.isEmpty()) {

int size = q.size();

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

Pair p = q.poll();

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

int nx = p.x + dx[i];

int ny = p.y + dy[i];

if (boundarycheck(grid, nx, ny) && grid[nx][ny] == 1) {

q.add(new Pair(nx, ny));

grid[nx][ny] = 2;

fresh--;

}

}

}

min++;

}

return fresh != 0 ? -1 : min - 1;

}

}

## Dry Run with Table

### Input Grid:

[ [2,1,1],

[1,1,0],

[0,1,1] ]

|  |  |  |  |
| --- | --- | --- | --- |
| Minute | Fresh | Rotten Spread to | Updated Grid |
| 0 | 8 | (0,1), (1,0) | 2 2 12 1 00 1 1 |
| 1 | 6 | (0,2), (1,1) | 2 2 22 2 00 1 1 |
| 2 | 4 | (2,1), (1,2) ❌ (invalid) | 2 2 22 2 00 2 1 |
| 3 | 3 | (2,2) | 2 2 22 2 00 2 2 ✅ |

Final fresh = 0, so return min - 1 = 4.

## Time / Space Complexity

|  |  |
| --- | --- |
| Metric | Value |
| Time | O(m × n) |
| Space | O(m × n) (queue) |

## Alternate Approaches

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Time | Space | Notes |
| ✅ BFS (Optimal) | O(m×n) | O(m×n) | Layered spread per minute |
| ❌ DFS | O(m×n) | O(m×n) | Hard to track time properly |
| ❌ Brute Simulation | O((m×n)²) | O(1) | Inefficient |