# 994. Rotting Oranges

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

0 representing an empty cell,

1 representing a fresh orange, or

2 representing 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.

## SOLUTION IN C++

class Solution {

public:

int orangesRotting(vector<vector<int>>& grid) {

constexpr int dirs[4][2] = {{0, 1}, {1, 0}, {0, -1}, {-1, 0}};

const int m = grid.size();

const int n = grid[0].size();

auto isNeighborRotten = [&](int i, int j, const vector<vector<int>>& grid) {

for (const auto& [dx, dy] : dirs) {

const int r = i + dx;

const int c = j + dy;

if (r < 0 || r == m || c < 0 || c == n)

continue;

if (grid[r][c] == 2)

return true;

}

return false;

};

int ans = 0;

while (true) {

vector<vector<int>> nextGrid(m, vector<int>(n));

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

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

if (grid[i][j] == 1) { // fresh

if (isNeighborRotten(i, j, grid))

nextGrid[i][j] = 2;

else

nextGrid[i][j] = 1;

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

nextGrid[i][j] = 2; // Keep rotten.

}

if (nextGrid == grid)

break;

grid = nextGrid;

++ans;

}

return any\_of(grid.begin(), grid.end(),

[&](vector<int>& row) {

return ranges::any\_of(row, [&](int orange) { return orange == 1; });

})

? -1

: ans;

}

};