LEETCODE 994 Medium:

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.

C/C++

```cpp
class Solution {
public:
    bool isValidCase(vector<vector<int>>& grid,int x,int y){
        int n = grid.size();
        int m = grid[0].size();
        return (x<n && x>=0 && y<m && y>=0);
    }
    void fillQueueWithRotten(vector<vector<int>>&
grid,queue<pair<int,int>> &que){
        int n = grid.size();
        int m = grid[0].size();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j]==2) que.push({i,j});
            }
        }
    }
    bool isEdgeCase(vector<vector<int>>& grid){
        int n = grid.size();
        int m = grid[0].size();
        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(grid[i][j]==1) return true;
            }
        }
        return false;
    }
    int orangesRotting(vector<vector<int>>& grid) {
        queue<pair<int,int>> que;
        fillQueueWithRotten(grid,que);
        if(!isEdgeCase(grid)) return 0;
        int ans = 0;
        vector<int> dir = {-1,0,1,0,-1};
        while(!que.empty()){
            auto [x,y] = que.front();que.pop();
            ans = max(ans,grid[x][y]);
            for(int i=1;i<5;i++){
                int _x = x + dir[i];
                int _y = y + dir[i-1];
                if(isValidCase(grid,_x,_y) && grid[_x][_y]==1){
                    grid[_x][_y] = grid[x][y]+1;
                    que.push({_x,_y});
                }
            }
        }
        if(isEdgeCase(grid)) return -1;
        return ans-2;
    }
};
```