MICROSOFT PREPARATION

DAY : 08

Rotting Oranges:

Problem Link : https://leetcode.com/problems/rotting-oranges/

Test Cases Passed : 306 / 306

Time Used : 30.15

Difficulty Level : MEDIUM

Approach Used :

- Calculating the dimensions of the grid
- Creating a visited vector having initially all elements as 0
- Creating an empty queue storing pair of element index and time
- Inserting all the oranges into the queue that are already rotten with a time required as 0
- Creating a variable to store the max time and initializing it with -1
- Traversing until the queue becomes empty :
    - Extracting the first element from the queue
    - Updating max time with max(maxtime, timeofnode)
    - Traversing for all 4 direction adjacent elements :
        - Checking validity of indexes :
            - Checking if not visited and is not a rotten orange :
                - Marking orange as visited
                - Pushing the orange into the queue with time as time+1
- Traversing the complete visited vector to check if the element is not a orange and is left unvisited :
    - Return -1
- Return maxtime variable

Function Code :

```
class Solution {
public:
   int orangesRotting(vector<vector<int>>& grid) {
```

```
        int n = grid.size();
        int m = grid[0].size();
        vector<pair<int, int>> directions = {{-1, 0}, {0, 1}, {1, 0}, {0, -1}};

        queue<pair<pair<int, int>, int>> q;
        vector<vector<int>> visited(n, vector<int>(m, 0));
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == 2) {
                    q.push({{i, j}, 0});
                    visited[i][j] = 1;
                }
            }
        }

        int minutes = 0;

        while (!q.empty()) {
            int row = q.front().first.first;
            int col = q.front().first.second;
            int time = q.front().second;
            q.pop();

            minutes = max(minutes, time);

            for (auto dir : directions) {
                int newRow = row + dir.first;
                int newCol = col + dir.second;

                if (newRow >= 0 && newRow < n && newCol >= 0 && newCol < m &&
grid[newRow][newCol] == 1 && !visited[newRow][newCol]) {
                    visited[newRow][newCol] = 1;
                    q.push({{newRow, newCol}, time + 1});
                }
            }
        }

        for (int i = 0; i < n; i++) {
            for (int j = 0; j < m; j++) {
                if (grid[i][j] == 1 && !visited[i][j]) {
                    return -1;
                }
            }
        }

        return minutes;
    }
```

```
};
```