Rotten Oranges

Given a grid of dimension **nxm** where each cell in the grid can have values 0, 1 or 2 which has the following meaning:
0 : Empty cell
1 : Cells have fresh oranges
2 : Cells have rotten oranges

We have to determine what is the earliest time after which all the oranges are rotten. A rotten orange at index [i,j] can rot other fresh orange at indexes [i-1,j], [i+1,j], [i,j-1], [i,j+1] (**up**, **down**, **left** and **right**) in unit time.

Example 1:

Input: grid = {{0,1,2},{0,1,2},{2,1,1}}

Output: 1

Explanation: The grid is-

0 1 2

0 1 2

2 1 1

Oranges at positions (0,2), (1,2), (2,0)

will rot oranges at (0,1), (1,1), (2,2) and

(2,1) in unit time.

Example 2:

Input: grid = {{2,2,0,1}}

Output: -1

Explanation: The grid is-

2 2 0 1

Oranges at (0,0) and (0,1) can't rot orange at

(0,3).

Your Task:
You don't need to read or print anything, Your task is to complete the function **orangesRotting()** which takes grid as input parameter and returns the minimum time to rot all the fresh oranges. If not possible returns -1.

Expected Time Complexity: O(n*m)
Expected Auxiliary Space: O(n*m)

```
class Solution
{
    public:
    //Function to find minimum time required to rot all
oranges.
    int orangesRotting(vector<vector<int>>& grid) {
        // Code here
        int n = grid.size();
    int m = grid[0].size();

    queue<pair<pair<int, int>, int>> q;
    vector<vector<int>> vis(n, vector<int>(m, 0));  //
Initialize vis with proper dimensions

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 2) {
                q.push({{i, j}, 0});
                vis[i][j] = 2;  // Mark as visited
            }
        }
    }

    int tm = 0;
    int drow[] = {-1, 0, 1, 0};
    int dcol[] = {0, 1, 0, -1};

    while (!q.empty()) {
        int r = q.front().first.first;
        int c = q.front().first.second;
        int t = q.front().second;
        q.pop();
        tm = max(tm, t);

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

```
            int nrow = r + drow[i];
            int ncol = c + dcol[i];

            if (nrow >= 0 && nrow < n && ncol >= 0 && ncol < m
&& vis[nrow][ncol] != 2 && grid[nrow][ncol] == 1) {
                q.push({{nrow, ncol}, t + 1});
                vis[nrow][ncol] = 2;
                grid[nrow][ncol] = 2;  // Update grid to
reflect the change
            }
        }
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (grid[i][j] == 1)  // If any fresh orange
remains
                return -1;
        }
    }
    return tm;
    }
};
```