**Rotten Oranges**

**Medium**Accuracy: 50.57% Submissions: 22582 Points: 4

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 minimum time required to rot all oranges. 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)

**Constraints:**  
1 ≤ n, m ≤ 500

class Solution  {

    public:

    //Function to find minimum time required to rot all oranges.

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

        // Code here

        vector<pair<int, int>> aux;

        queue<pair<int, int>> q;

        int no1=0;

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

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

                if (grid[i][j]==2) q.push({i, j});

                else if (grid[i][j]==1) no1++;

            }

        }

        q.push({-1, -1});

        int tim=0;

        while (!q.empty()) {

            pair<int, int> p=q.front();

            q.pop();

            int i=p.first;

            int j=p.second;

            if (i==-1 and j==-1) {

                if (q.empty()) break;

                else {

                    tim++;

                    q.push({-1, -1});

                    i=q.front().first;

                    j=q.front().second;

                }

            }

            if (i-1>=0 and grid[i-1][j]==1) {

                grid[i-1][j]=2;

                no1--;

                q.push({i-1, j});

            }

            if (i+1<grid.size() and grid[i+1][j]==1) {

                grid[i+1][j]=2;

                no1--;

                q.push({i+1, j});

            }

            if (j+1<grid[0].size() and grid[i][j+1]==1) {

                grid[i][j+1]=2;

                no1--;

                q.push({i, j+1});

            }

            if (j-1>=0 and grid[i][j-1]==1) {

                grid[i][j-1]=2;

                no1--;

                q.push({i, j-1});

            }

        }

        if (no1) return -1;

        else return tim;

    }

};