SQP12 Rotting Oranges

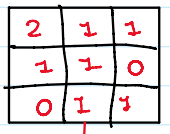
Monday, December 5, 2022 10:56 AM

The Problem says that you have a m×n Grid each cell have a same Value that is Represent the

0 → Represent an Empty cell
1 → Represent a Fresh Orange
2 → Represent a rotten Orange

→ And we moved four direction only ↔ with Contiguous Oranges

Our task is Return the Minimum Number of time to take all the Fresh Oranges get Rotten
→ and if any Oranges are remaining so Simply Return -1



Example 1:

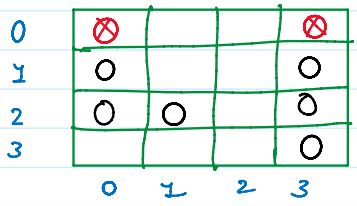
→ In Stage 4 My all the Oranges get Rotten

```
Input: grid = [[2,1,1],[1,1,0],[0,1,1]]
Output: 4
```

→ At a Single time only Moved 4 direction ↔ ↕ → other wise Not moved

→ that is the over problem say

→ Lets Understand with Example with 4×4 Matrix



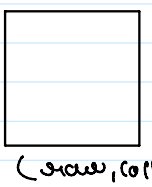
Thought Process

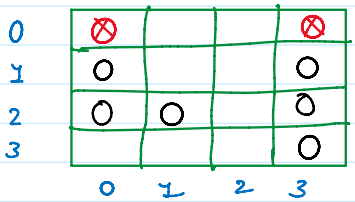
→ Here we used the concept of BFS
→ We requirised the Queue data Structure

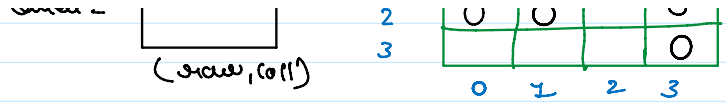
→ First we count the How many Oranges we have

Oranges = 8

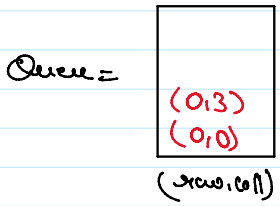
→ Maintain the Queue data Structure

Queue = 

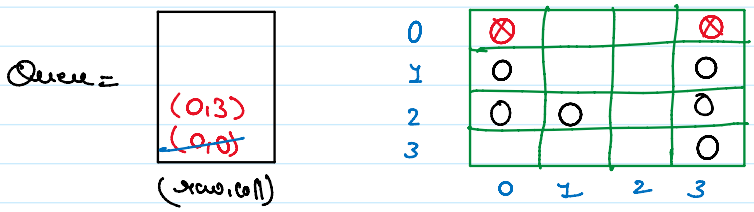




→ Now push the Rotten Oranges index into the Queue data structure



→ After pushing the Rotten Oranges start iteration on next Rotten Oranges that we push into the Queue data structure in ↑ ↓ ← → direction

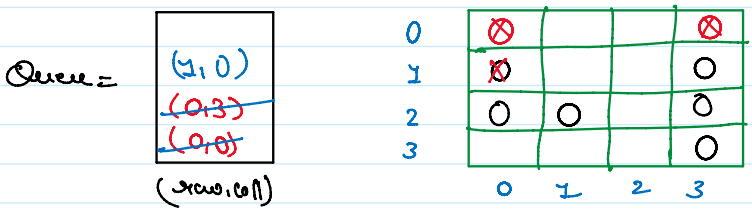


Not moved because out of bound

Iteration on = (0,0) → (0,0) ← Present Empty cell

out of bound

Yes we moved on the (1,0) and marked as a Rotten Oranges and put into the Queue (3)



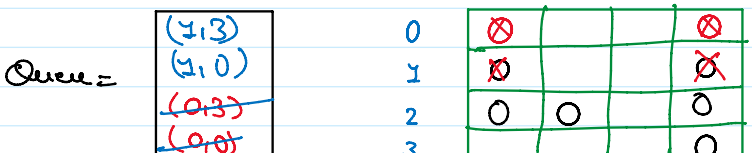
Iteration on = (0,3)

Same process follow Again

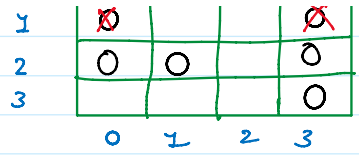
out of bound

Empty cell ← (0,3) → out of bound

Yes we moved on the (1,3) and marked as a Rotten Oranges and put into the Queue



Queue = (1,0) ~~(0,3)~~ ~~(0,0)~~ (row,col)



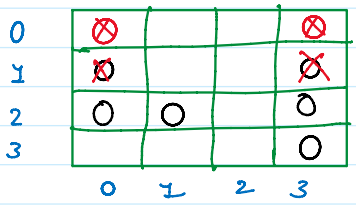
Important Note

→ After Complete the one full Iteration of all the existence Rotten Oranges into the Queue increase the Over Minute + 1

→ After Complete the Iteration we check into the Queue if any present into the Queue that Means we Rotten Some Oranges

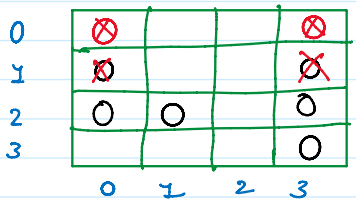
Minute = 1

Queue = (1,3) (1,0) ~~(0,3)~~ ~~(0,0)~~ (row,col)

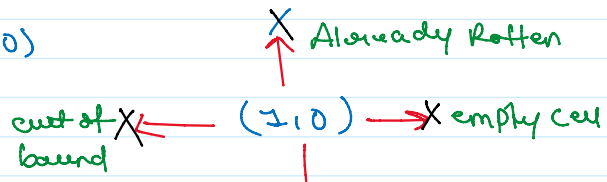


→ Again Same Approach follows take the Indices from the Queue and perform Operation

Queue = ~~(1,3)~~ ~~(1,0)~~ ~~(0,3)~~ ~~(0,0)~~ (row,col)

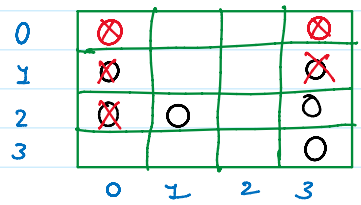


Iteration on = (1,0)

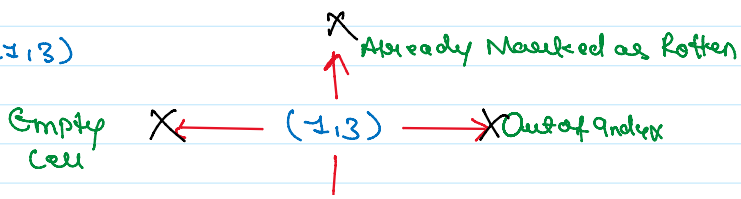


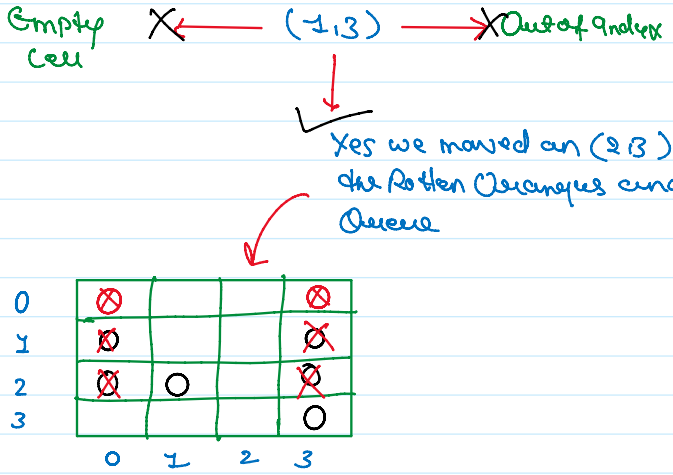
Yes we moved on the (1,0) and marked as the Rotten Orange and pushed into the Queue

Queue = (2,0) ~~(1,3)~~ ~~(1,0)~~ ~~(0,3)~~ ~~(0,0)~~ (row,col)



Iteration on = (1,3)



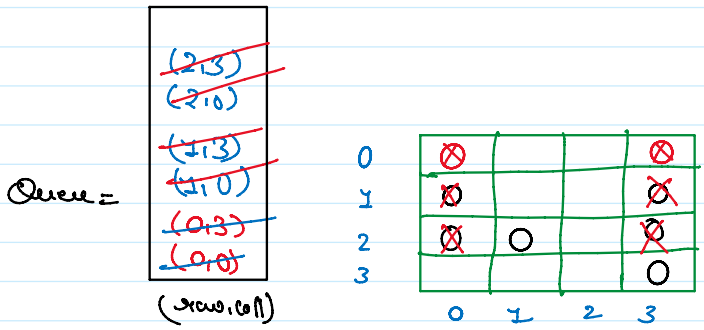


After Complete the one full Iteration of all the existence Rotten Orange into the Queue Increase the Over Minute + 1

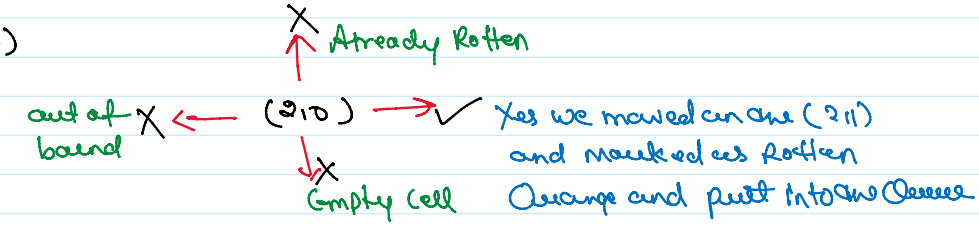
After Complete the Iteration we check into the Queue if any present into the Queue that Means we Rotten Some Oranges

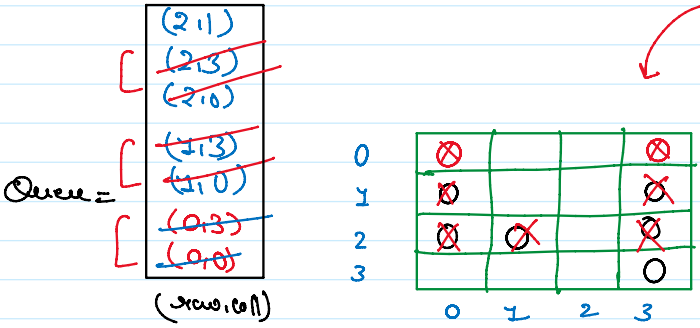
Minute = 2

→ Again Same Approach follows take the Indices from the Queue and perform Operation



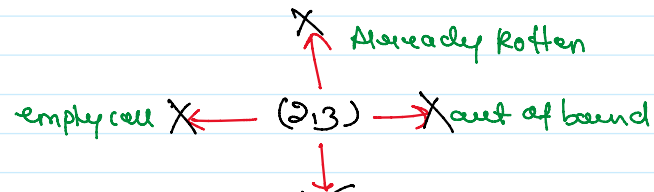
Iteration on = (2,0)



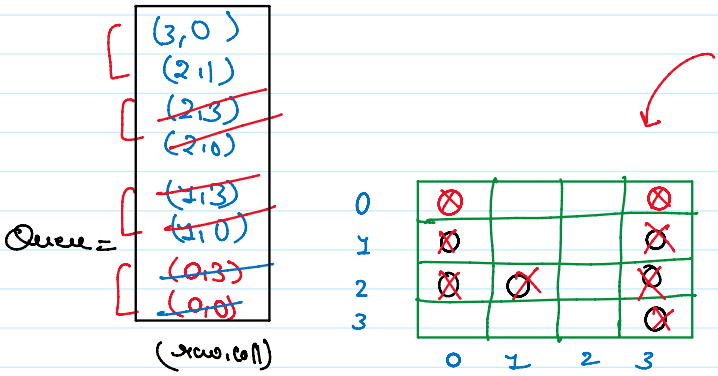


Iteration on = (2,3)

Iteration on = (2,3)



Yes we moved on the (3,1) and Marked as the Rotten Oranges and put into the Queue

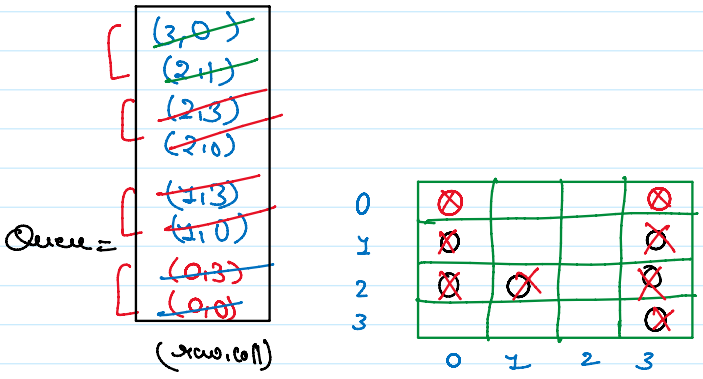


After Complete the one full Iteration of all the existence Rotten Oranges into the Queue Increase the Over Minute + 1

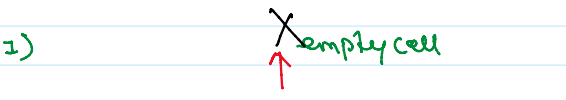
→ after complete the Iteration we check into the Queue if any present into the Queue that Means we Rotten Some Oranges

Minute = 3

→ Again Same Approach follows take the Indices from the Queue and perform Operation

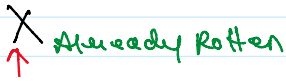


Iteration on = (2,1)



→ Hence we say on this Operation we not do anything No need to anything into the Queue

Iteration on = (3,0)



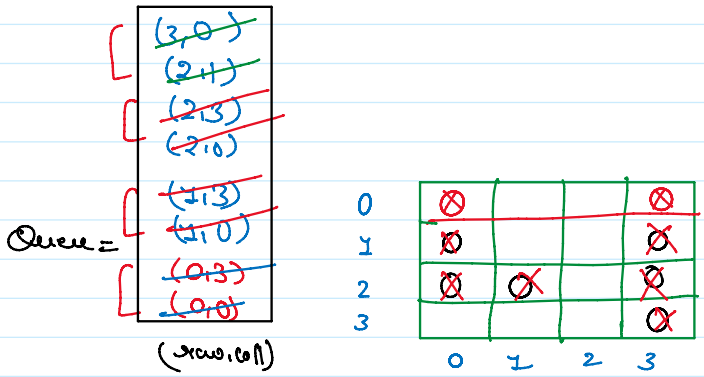
Iteration on = (3,0)

Already Rotten

Empty Cell ← (3,0) → out of bound

out of bound

→ Hence we say on this Operation we Not do anything No need to anything into the Queeee



After Complete the one full Iteration of all the existence Rotten Oranges into the Queeee

→ Check if Queue is Empty that Means we Cannot perform any Operation on previously Iteration

→ And that is the point where end the Iteration and between the time

→ Now My total Oranges into the grid is = 8

→ And My Queeee total pair is = 8

8 == 8 ✓

that Means 9 Make all the Oranges are get Rotten

Minute = 3 answer

Important Points —

if total oranges are ≠ to the total pair of the queeee that Means we can Not make all the Oranges are get rotten

→ So on this case we Return -1 and that is our answer

Time Complexity = O(N x N) X 4 → because we Iterate into the (4) direction

→ Beause we Iterate Nested loop and on Worst case we Iterate for every Oranges

SpaceComplexity = O(N x N)

→ because on worst case we push

Space Complexity = O(N×N)

→ because in worst case we push all one list of index into the deque and is RottenOrange

#Implementation Python

```
# lc: https://leetcode.com/problems/rotting-oranges/
# gfg:  https://practice.geeksforgeeks.org/problems/rotten-oranges2536/1

from collections import deque
class Solution:
    def orangesRotting(self, grid: List[List[int]]) -> int:

        if len(grid) == 0:
            return 0
        row = len(grid)
        coll = len(grid[0])

        total = 0
        days = 0
        count = 0

        queue = deque()

        for i in range(row):
            for j in range(coll):
                if grid[i][j] != 0:
                    total += 1
                if grid[i][j] == 2:
                    queue.append([i, j])

        print(total)
        print(queue)

        dx = [0, 0, 1, -1]
        dy = [1, -1, 0, 0]

        while len(queue) != 0:
            sizeOfQueue = len(queue)
            count += sizeOfQueue

            while sizeOfQueue > 0:
                data = queue.popleft()
                x = data[0]
                y = data[1]

                for i in range(4):
                    nx = x + dx[i]
                    ny = y + dy[i]

                    if nx < 0 or ny < 0 or nx >= row or ny >= coll or grid[nx][ny] != 1:
                        continue
                    grid[nx][ny] = 2
                    queue.append([nx, ny])

                sizeOfQueue -= 1

            if len(queue) != 0:
                days += 1
```

→ dx, dy → for ←↑→↓ → direction of moved four every indexes

→ direction of moved decided where we moved

#Implementation Java

```
class Solution {


    public int orangesRotting(int[][] grid) {

        if(grid == null || grid.length == 0) return 0;

        int rows = grid.length;
        int cols = grid[0].length;

        Queue<int[]> queue = new LinkedList<>();
        int count_fresh = 0;

        //Put the position of all rotten oranges in queue
        //count the number of fresh oranges
        for(int i = 0 ; i < rows ; i++) {
            for(int j = 0 ; j < cols ; j++) {
                if(grid[i][j] == 2) {
                    queue.offer(new int[]{i , j});
                }
                if(grid[i][j] != 0) {
                    count_fresh++;
                }
            }
        }

        if(count_fresh == 0) return 0;
        int countMin = 0, cnt = 0;
        int dx[] = {0, 0, 1, -1};
        int dy[] = {1, -1, 0, 0};

        //bfs starting from initially rotten oranges
        while(!queue.isEmpty()) {
            int size = queue.size();
            cnt += size;
            for(int i = 0 ; i < size ; i++) {
                int[] point = queue.poll();
                for(int j = 0;j<4;j++) {
                    int x = point[0] + dx[j];
                    int y = point[1] + dy[j];

                    if(x < 0 || y < 0 || x >= rows || y >= cols ||
grid[x][y] == 0 || grid[x][y] == 2){
                        continue;
                    }

                    grid[x][y] = 2;
                    queue.offer(new int[]{x , y});
                }
            }
            if(queue.size() != 0) {
                countMin++;
            }
        }
```

```
    if len(queue) != 0:
        days += 1

if total == count:
    return days
return -1
```

```
                countMin++;
            }
        }
        return count_fresh == cnt ? countMin : -1;

    }
}
```