LEETCODE | 994 |

DATE

Rotting Oranges

Input: grid = $\begin{bmatrix} 2 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ Output = -1

Approach We need to find minimum time so that all oranges may turn rotten
So, we can approach it with BFS

Step1 To capture coordinates & time we can create a class

```
class Pair {
        int row;
        int col;
        int time;
}
```

Step2

```
int timeMin = 0;
int m = grid.length; //rows
int n = grid[0].length; //cols
int fresh = 0; int converted=0
Queue<Pair> queue. = new LinkedList<Pair>();
```

Step3 Mark visited[][] array elements = 1 where grid[i][j] = 2 & rest as 0

Step4

```
if(grid[i][j]==1){
        ↳ fresh++;
}
else if (grid[i][j]==2){
        visited[i][j]= 1;
        queue.offer(new Pair(i, j, 0));
}
```

classmate

PAGE

Step 5 Run a BFS while loop

```
while (! queue.isEmpty()) {
    Pair current = queue.poll();
    int row = current.row;
    int col = current.col;
    int time = current.time;
    if (time > minTime) {
        minTime = time;
    }
```

Step 6 Use directions (4 direction delta Row & col) and run for loop and secure column & rows

```
for (int i=0; i < directions; i++) {
    int effR = row + deltaRow[i];
    int effC = col + deltaCol[i];

// secure row & col
    if (effR >= 0 && effR < m &&
        effC >= 0 && effC < n &&
        visited[effR][effC] == 0 &&
        grid[effR][effC] == 1) {

        visited[effR][effC] = 1;
        grid[effR][effC] = 2;
        queue.offer(new Pair(effR, effC,
                             time + 1));
        converted++;
    }
}
```

oranges rotten here →

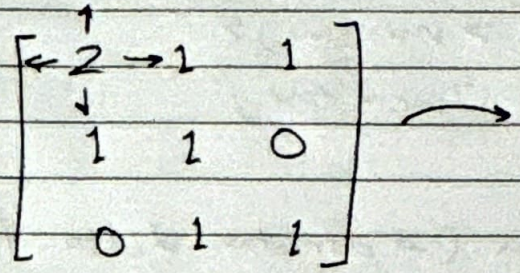
no. of converted rotten oranges ←

DATE

Step 7

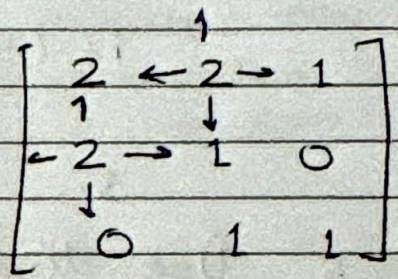
Return converted == fresh ? timeMin : -1;

Input →

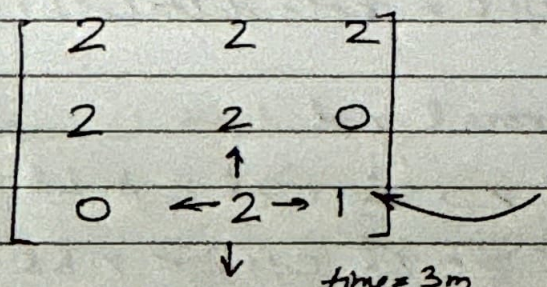


time = 0 m
fresh = 6
converted = 0

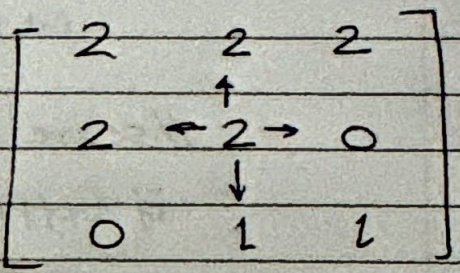
Output → 4 minutes



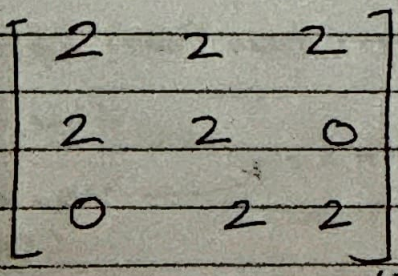
time = 1 m
fresh = 6
converted = 2



time = 2m
fresh = 6
converted = 4



time = 3m
fresh = 6
converted = 5



time = 4m
fresh = 6
converted = 6

converted = fresh
return 4 minutes