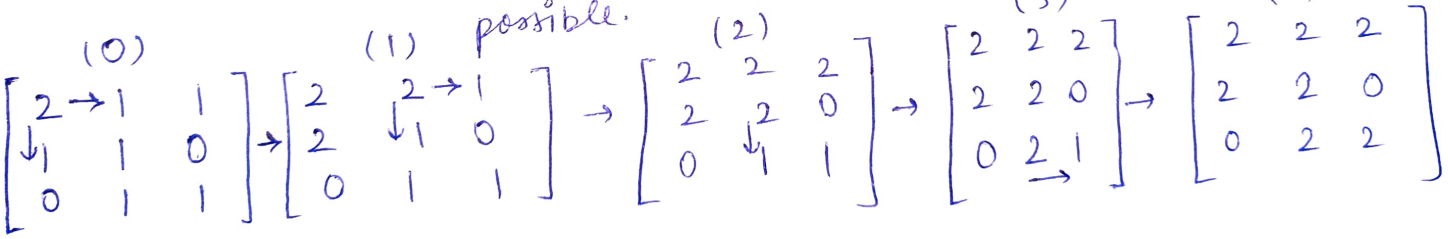
(7) Rotting Oranges

0 → Empty cell
1 → Fresh Orange
2 → Rotten Orange

Any fresh tomato that is 4-directionally adjacent to rotten orange becomes rotten.

After converting all fresh to rotten orange, if there are some fresh orange, just -1 as the operation is not possible.



Just return 4 here, as this was the minimum no. of minutes will taken to complete the operatn & also atlast there is no fresh orange.

Approach: BFS Traversal will be used here, as we traversing it breadth wise and while traversing we will increase time after each operatn.

Code:

```
bool isValid(grid,i,j,n,m) {
    if (i<n && i>=0 && j<m && j>=0 && grid[i][j]==1)
        return true;
    return false;
}
int bfs (q, grid, n, m) {
    int time=0;
    while (!q.empty()) {
        int size = q.size(), temp=0;
        while (size!=0) {
            int x1 = q.front().first;
            int y1 = q.front().second;
            q.pop();
            int ax[4] = {1,-1,0,0};
            int ay[4] = {0,0,1,-1};
            for (i→4) {
                int x = x1 + ax[i];
                int y = y1 + ay[i];
                if (isValid(grid,x,y,n,m)) {
                    temp++;
                    grid[x][y]=2;
                    q.push({x,y});
                }
            }
            size--;
        }
        if (temp!=0)
            time++;
    }
    return time;
}
int orangeRotting(grid) {
    int n = grid.size(), m = grid[0].size();
    int time=0, fresh=0;
```

→ Taking 2 helping arrays to covers all 4-directions

→ In this loop, this will be imply all 4 coordinates

(1,0)	(-1,0)	(0,1)	(0,-1)

```
    queue<pair<int,int>> q;
    for(i→n)
        for(i→m)
            if(grid[i][j]==2)
                q.push({i,j});
            else if(grid[i][j]==1)
                fresh++;
    if(fresh==0)
        return 0;
    time = bfs(q, grid, n, m);
    if(isFreesh(grid, n, m))
        return -1;
    return time
}
bool isFresh(grid, n, m){
    for(i→n)
        for(i→m)
            if(grid[i][j]==1)
                return 1;
    return 0;
}
```

→ Checking if there is no fresh oranges, we don't need to do operations

→ Checking if still, there is some fresh tomato.