Rotten Oranges

Using BFS. Traverse the matrix once and push all elements index where the rotten element is present with an initial time of 0. For all adjacent 1 to these oranges pop from the queue and increment the time by 1 then push into the queue. Repeat until queue is empty. At last make a check whether 1 is left unvisited. Visited vector makes a track of unvisited oranges.

Code :

```
#include <bits/stdc++.h>
int minTimeToRot(vector<vector<int>>& grid, int n, int m)
{
    // Write your code here.
    vector<vector<int>> visited(n,vector<int>(m,0));
    queue<pair<pair<int,int>,int>> q;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
            if(grid[i][j]==2)
            {
                q.push({{i,j},0});
                visited[i][j]=1;
            }
        }
    }
    int mxtime=0;
    int drow[]={-1,1,0,0};
    int dcol[]={0,0,-1,1};
    while(!q.empty())
    {
        auto tp=q.front();
        int r=tp.first.first;
        int c=tp.first.second;
        int t=tp.second;
        q.pop();

        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 && grid[nrow][ncol]==1 && visited[nrow][ncol]==0)
            {
                q.push({{nrow,ncol},t+1});
                visited[nrow][ncol]=1;
                mxtime=max(mxtime,t+1);
            }
        }
    }
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<m;j++)
        {
```

```
            if(grid[i][j]==1 && visited[i][j]==0)
                return -1;
        }
    }
    return mxtime;
}
```

- Time Complexity : O(n^2)x4
- Space Complexity : O(n^2)