Minimum time required to rot all oranges

Given a matrix of dimension m*n where each cell in the matrix 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
```

So we have to determine what is the minimum time required so that all the oranges become rotten. 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). If it is impossible to rot every orange then simply return -1.

Examples:

```
Input:  arr[][C] = { {2, 1, 0, 2, 1},
                     {1, 0, 1, 2, 1},
                     {1, 0, 0, 2, 1}};
Output:
All oranges can become rotten in 2 time frames.


Input:  arr[][C] = { {2, 1, 0, 2, 1},
                     {0, 0, 1, 2, 1},
                     {1, 0, 0, 2, 1}};
Output:
All oranges cannot be rotten.
```

Source: Microsoft Interview Question

We strongly recommend you to minimize your browser and try this yourself first.

The idea is to user Breadth First Search. Below is algorithm.

```
1) Create an empty Q.
2) Find all rotten oranges and enqueue them to Q.  Also enqueue
   a delimiter to indicate beginning of next time frame.
3) While Q is not empty do following
   3.a) While delimiter in Q is not reached
       (i) Dequeue an orange from queue, rot all adjacent oranges.
           While rotting the adjacents, make sure that time frame
           is incremented only once. And time frame is not icnremented
           if there are no adjacent oranges.
   3.b) Dequeue the old delimiter and enqueue a new delimiter. The
        oranges rotten in previous time frame lie between the two
        delimiters.
```

Below is C++ implementation of the above idea.

```
// C++ program to find minimum time required to make all
// oranges rotten
#include<bits/stdc++.h>
#define R 3
#define C 5
using namespace std;

// function to check whether a cell is valid / invalid
bool isvalid(int i, int j)
{
    return (i >= 0 && j >= 0 && i < R && j < C);
}

// structure for storing coordinates of the cell
```

```
struct ele {
    int x, y;
};

// Function to check whether the cell is delimiter
// which is (-1, -1)
bool isdelim(ele temp)
{
    return (temp.x == -1 && temp.y == -1);
}

// Function to check whether there is still a fresh
// orange remaining
bool checkall(int arr[][C])
{
    for (int i=0; i<R; i++)
       for (int j=0; j<C; j++)
          if (arr[i][j] == 1)
             return true;
    return false;
}

// This function finds if it is possible to rot all oranges or not.
// If possible, then it returns minimum time required to rot all,
// otherwise returns -1
int rotOranges(int arr[][C])
{
    // Create a queue of cells
    queue<ele> Q;
    ele temp;
    int ans = 0;

    // Store all the cells having rotten orange in first time frame
    for (int i=0; i<R; i++)
    {
       for (int j=0; j<C; j++)
       {
            if (arr[i][j] == 2)
            {
                temp.x = i;
                temp.y = j;
                Q.push(temp);
            }
        }
    }

    // Separate these rotten oranges from the oranges which will rotten
    // due the oranges in first time frame using delimiter which is (-1, -1)
    temp.x = -1;
    temp.y = -1;
    Q.push(temp);

    // Process the grid while there are rotten oranges in the Queue
    while (!Q.empty())
    {
        // This flag is used to determine whether even a single fresh
        // orange gets rotten due to rotten oranges in current time
        // frame so we can increase the count of the required time.
        bool flag = false;

        // Process all the rotten oranges in current time frame.
        while (!isdelim(Q.front()))
        {
            temp = Q.front();

            // Check right adjacent cell that if it can be rotten
            if (isvalid(temp.x+1, temp.y) && arr[temp.x+1][temp.y] == 1)
            {
                // if this is the first orange to get rotten, increase
                // count and set the flag.
                if (!flag) ans++, flag = true;

                // Make the orange rotten
```

```
                arr[temp.x+1][temp.y] = 2;

                // push the adjacent orange to Queue
                temp.x++;
                Q.push(temp);

                temp.x--; // Move back to current cell
            }

            // Check left adjacent cell that if it can be rotten
            if (isvalid(temp.x-1, temp.y) && arr[temp.x-1][temp.y] == 1) {
                if (!flag) ans++, flag = true;
                arr[temp.x-1][temp.y] = 2;
                temp.x--;
                Q.push(temp); // push this cell to Queue
                temp.x++;
            }

            // Check top adjacent cell that if it can be rotten
            if (isvalid(temp.x, temp.y+1) && arr[temp.x][temp.y+1] == 1) {
                if (!flag) ans++, flag = true;
                arr[temp.x][temp.y+1] = 2;
                temp.y++;
                Q.push(temp); // Push this cell to Queue
                temp.y--;
            }

            // Check bottom adjacent cell if it can be rotten
            if (isvalid(temp.x, temp.y-1) && arr[temp.x][temp.y-1] == 1) {
                if (!flag) ans++, flag = true;
                arr[temp.x][temp.y-1] = 2;
                temp.y--;
                Q.push(temp); // push this cell to Queue
            }

            Q.pop();
        }

        // Pop the delimiter
        Q.pop();

        // If oranges were rotten in current frame than separate the
        // rotten oranges using delimiter for the next frame for processing.
        if (!Q.empty()) {
            temp.x = -1;
            temp.y = -1;
            Q.push(temp);
        }

        // If Queue was empty than no rotten oranges left to process so exit
    }

    // Return -1 if all arranges could not rot, otherwise -1.
    return (checkall(arr))? -1: ans;
}

// Drive program
int main()
{
    int arr[][C] = { {2, 1, 0, 2, 1},
                     {1, 0, 1, 2, 1},
                     {1, 0, 0, 2, 1}};
    int ans = rotOranges(arr);
    if (ans == -1)
        cout << "All oranges cannot rot\n";
    else
         cout << "Time required for all oranges to rot => " << ans << endl;
    return 0;
}
```

Output:

```
Time required for all oranges to rot => 2
```