MEDIUM

Surrounded Regions | Replace O's with X's

Intuition

X X X X
X O X X
X O O X
X O X X
X X O O

O is surrounded with X if there are X at location just below, just above, just left and just right
Eg.

  X
X O X
  X

eg.

X X X X
X O X X
X O O X
X O X X
X X O O

The O's surrounded by X can be converted to X while last 2 aren't covered at bottom

Inference :

- If some 'O' is at the boundary then it cannot be converted
- They are bound to be surrounded by X only if they are not connected to a boundary wall

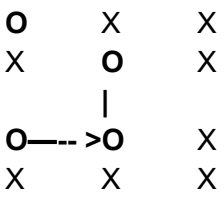
Eg.

X X X X
X O X X
X O O X
X O X X
X X O O

A set of O connected to the boundary can not be converted

We will be using the DFS algorithm for traversing the elements at the boundary

Eg.



Start from boundary and mark the O's that can be traversed as they can not be converted

Creating a result and visited matrix

Eg.

Original Matrix

X	X	X	X	X
X	O	O	X	**O**
X	X	O	X	**O**
X	**O**	X	O	X
O	**O**	X	X	X

Visited Matrix

0	0	0	0	0
0	0	0	0	1
0	0	0	0	1
0	1	0	0	0
1	1	0	0	0

dfs(4,0) -> dfs(4,1) | dfs(3,0) [Already X]

dfs(4,1) -> dfs(3,1) | dfs(4,2) [Already X]
dfs(3,1) -> dfs(3,0) [Already X] | dfs(2,1) [Already X] | dfs(3,2) [Already X] | visited (goes back)

Now traverse the visited matrix and where you have one convert the O's in matrix to the original matrix except those who are 1 in the visited matrix

Approach :

Fill

- Declare :
  - Visited matrix with all elements as 0 of size n*m
- Traverse for the first row and the last row and all columns :
  - If the element is unvisited and 'O' then :
    - Call dfs as dfs(i,j,visited,mat)
- Traverse for the first column and the last column and all rows :
  - If the element is unvisited and 'O' then :
    - Call dfs as dfs(i,j,visited,mat)
- Traverse for n*m :
  - If mat[i][j] is 'O' and unvisited (ie. it has not been touched by any element at boundary) then :
    - Convert it to 'X'
- Return mat

DFS()

- Compute the dimensions of the matrix
- Mark the source node from fill as visited
- Traverse in all 4 directions of immediate up,down,left,right :
  - Check for validity of new rows and cols :
    - If its unvisited and its and 'O':
      - Dfs call it as dfs(nrow,ncol,visited,mat)

Program Code :

```
void dfs(int row,int col,vector<vector<int>> &visited,vector<vector<char>>
&mat)
    {
        // computing the dimensions of the matrix
        int n = mat.size();
        int m = mat[0].size();

        // Assigning the source as visited
        visited[row][col] = 1;
```

```
        //traversing through the 4 direction adjacent elements
        int delRow[] = {-1,0,1,0};
        int delCol[] = {0,1,0,-1};

        for(int i=0;i<4;i++)
        {
            // calculating the neighbor rows and columns
            int nrow = row+delRow[i];
            int ncol = col+delCol[i];

            //checking for validity

            if(nrow<n && nrow>=0 && ncol<m && ncol>=0)
            {
                // checking if the element is unvisited and is and 'O'
connected to a boundary
                if(!visited[nrow][ncol] && mat[nrow][ncol]=='O')
                {
                    // recursive calling all the connected components that
are an 'O'
                    dfs(nrow,ncol,visited,mat);
                }
            }

        }

    }
    vector<vector<char>> fill(int n, int m, vector<vector<char>> mat)
    {
        // Declaring a visited array

        vector<vector<int>> visited(n,vector<int>(m,0));
        // traversing first row and last row and all columns
        for(int j=0;j<m;j++)
        {
            //Row - 0
            if(!visited[0][j] && mat[0][j]=='O')
            {
                dfs(0,j,visited,mat);
            }
            //Row - n-1
            if(!visited[n-1][j]&& mat[n-1][j]=='O')
            {
```

```
                dfs(n-1,j,visited,mat);
            }
        }

        //traversing for the first column and the last column and all rows
        for(int j=0;j<n;j++)
        {

            //Column 0
            if(!visited[j][0] && mat[j][0]=='O')
            {
                dfs(j,0,visited,mat);
            }
            //Column m-1
            if(!visited[j][m-1]&& mat[j][m-1]=='O')
            {
                 dfs(j,m-1,visited,mat);
            }
        }

        // Traversing for the complete matrix

        for(int i=0;i<n;i++)
        {
            for(int j=0;j<m;j++)
            {
                // checking if the element is not visited and it is O ie.
not touched by boundary
                if(!visited[i][j] && mat[i][j]=='O')
                {
                    // convert it to X
                    mat[i][j] = 'X';
                }
            }
        }
        return mat;
}
```

Time Complexity :

O(n*m)