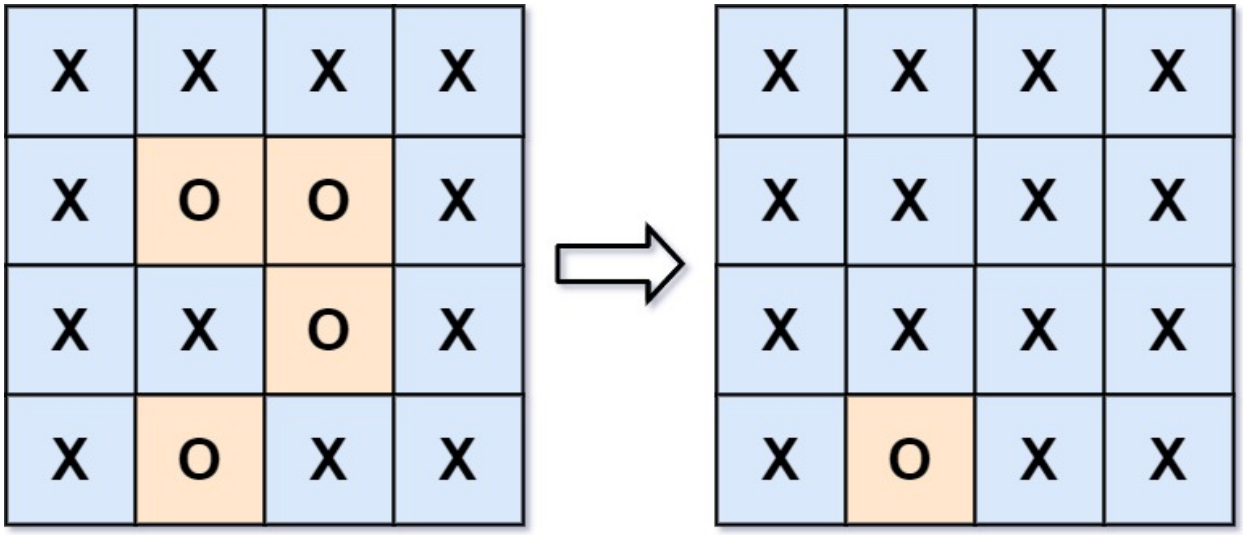
Given an m x n matrix board containing 'X' and 'O', capture all regions that are 4-directionally surrounded by 'X'. A region is captured by flipping all 'O's into 'X's in that surrounded region.



Input: board =
[["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]

Output: [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]

Explanation: Notice that an 'O' should not be flipped if:
- It is on the border, or
- It is adjacent to an 'O' that should not be flipped.
The bottom 'O' is on the border, so it is not flipped.
The other three 'O' form a surrounded region, so they are flipped.

Link - [Rohith Boodireddy]

My peers code link:

Deva Rithish Punna
Mohan Gundluri
Sonali Gudey

Observation: since we need to convert all the O's to 'X' which are surrounded by 'X's in all 4 directions.so first step is to find the O's which are not surrounded by X's in all 4 directions and it is possible only when
i) 'O' is on the border
ii)and all the connecting that 'O'

My approach:
Step one: find all the Os in the borders
Step two: Do a Depth first search or Breadth first search from the above Os and turn them to any different symbol($)
Step Three: Then Make the other O's Xs and the $'s Os

Review of my peers code:
DFS Comparison:

All 3 of my peers and my code have the same time and space complexity and the approach was similar.But i think they can still improve their code.

Here is the part where i think they can improve their code

My **Code:**

```
  for i in range(m):
        if board[i][0] == 'O':
            dfs(i,0)
        if board[i][n-1] == 'O':
            dfs(i, n-1)
   for j in range(n):
        if board[0][j] == 'O':
          dfs(0, j)
        if board[m-1][j] == 'O':
          dfs(m-1, j)
```

My peer's code:

```
for i in range (0,n):
  for j in range (0,m):
    if (board[i][j]=='O' and (i in [0,n-1] or j in [0,m-1])):
        dfs(board,i,j,n,m)
```

My program runs slightly faster because I am only traversing the boundaries and DFS/BFS the O's but my peers' code iterates through the whole 2d list instead of just iterating through the boundaries . with small m*n there won't be much difference but as the list size increases their code becomes slightly slower .

Let's say there is m*n matrix.
->my code traverses the boundaries (m+n) and dfs the found O,s and eventually the **average** case time complexity becomes m*n(100*100) and for updation m*n
****Time complexity - 2*(m*n) + m+n —-->O(m*n)

->whereas my peers code traverses the whole 2d list m*n(100*100) and dfs the found O's and eventually the **average** case time complexity becomes m*n(100*100) and for updation m*n(100*100)
****Time complexity - 3*(m*n) —-->O(m*n)

BFS Comparsion:
2 of my peers(mohan and sonali) have coded in both dfs and bfs.Dfs comparison was given above and bfs comparison follows
->my code and my peers code have same time and space complexity which is (m*n) and (m*n) respectively and approach followed was similar
->but there is a very slight things which my peers can improve

My code:

```
for i in range(len(dx)):
      new_row,new_col = row+dx[i],col+dy[i]
       if (0 <= new_row < m and 0 <= new_col < n and board[new_row][new_col] == 'O'):
         q.append((new_row,new_col))
```

My peers code:

```
while queue:
       r, c = queue.popleft()
       if 0 <= r < m and 0 <= c < n and board[r][c] == 'O':
           board[r][c] = 'N'

           queue.append((r-1, c))
           queue.append((r+1, c))
           queue.append((r, c-1))
           queue.append((r, c+1))
```

->i am pushing the values only if it is ‘O’ and then doing bfs .while my peers pushing every element surrounding in it and then checking .so in this they can improve their space complexity .In real scenarios this doesn't make difference but that is the only difference i found

****Time Complexity - O(m*n)
****space complexity - O(m*n)