Group Members : Kushagra Singh
Rahul Kumar
Shubham Jhawar

Question : Given an m*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 to '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"]]

Basically All of us have the same underlying algorithm i.e by using the reverse approach.

Firstly I solved the question using the boolean visited array(same size as the input array) initialized with ‘false’. So the visited array will keep records of Os which can not be captured . So we marked the same True. Next time we iterate through the input array and we capture all the remaining Os for which the visited array’s value is False. But this approach takes extra space of O(row* column)

Optimed Approach :
Firstly iterating through the boundary points and changing it to some other value say ‘#’ and then the remaining Os will be captured and flipped to X . In the end, we revert back the changed value to O.

Kushgra’s and I have used the same approach.

However there are some minute changes in the Rahul and Shubham’s code which is documented below:

Shubham's Approach :

During implementing the DFS algorithm we have to traverse in 4-direction: Up, down , left and right.
There are two ways we can do this . First one is just to call the DFS function and in the argument itself change the direction which is implemented by Kushagra and me.

```
Code :   dfs(board,x+1,y)     -> down
         dfs(board,x-1,y)     ->  up
         dfs(board,x,y+1)     -> right
         dfs(board,x,y-1)     -> left
```

Shubham has done it by storing the direction in  a list of lists and he iterates  through  the list to change the direction.

```
Code : steps  = [[-1,0],[1,0],[0,-1],[0,1]]
       For nbr in steps:
           X = x + nbr[0]
           Y = y + nbr[1]
           dfs(matrix,X,Y)
```

Remaining implementation is the same.
Time and space complexity remains the same.

Rahul's Approach :

During implementing the DFS algorithm we have to traverse in 4-direction: Up, down , left and right.

There are two ways we can do this . First one is just to call the DFS function and in the argument itself change the direction which is implemented by Kushagra and me.

```
Code :   dfs(board,x+1,y)     -> down
         dfs(board,x-1,y)     ->  up
         dfs(board,x,y+1)     -> right
         dfs(board,x,y-1)     -> left
```

Rahul has done it using storing the direction in two lists X and Y
He iterates through  the list to change the direction.

```
Code :  X = [0,1,-1,0]
        Y = [-1,0,0,1]

        For k in range(0,4):
            dfs(matrix,x+X[k],y+Y[k])
```

Remaining implementation is the same.
Time and space complexity remains the same.