RITIK PATIDAR
TAS203

GitHub link : https://github.com/ritikpatidarrp/PythonAssignment/blob/main/code.py

**Description:**

The **solve** method is designed to modify a given 2D board in-place by marking the connected components of 'O' characters that are completely surrounded by 'X' characters. The algorithm employs depth-first search (DFS) to identify and mark these components.

**CODE:**

```
class Solution:
    def solve(self, board: List[List[str]]) -> None:
        """
        Do not return anything, modify board in-place instead.
        """

        def checkIfSurroundedByX(i,j,board,vis):
            # Base case: if the position is out of bounds or
already visited, return False
            if i < 0 or j < 0 or i == len(board) or j ==
len(board[0]):
                return False
            # If the current position contains 'X', it is
surrounded
            if board[i][j] == 'X':
                return True
            # If the position has already been visited, return
True to avoid redundant checks
            if vis[i][j] == True:
                return True
            # Mark the position as visited
            vis[i][j] = True
            # Recursively check the neighbors in all four
directions
            ans = checkIfSurroundedByX(i-1, j, board, vis)
            ans = ans and checkIfSurroundedByX(i+1, j, board, vis)
            ans = ans and checkIfSurroundedByX(i, j-1, board, vis)
            ans = ans and checkIfSurroundedByX(i, j+1, board, vis)
            return ans

        def markXtheComponent(i,j,board):
            # Base case: if the current position contains 'X', do
nothing
            if board[i][j] == 'X':
                return
            # Mark the current position as 'X'
            board[i][j] = 'X'
            # Recursively mark the neighbors in all four
directions
            markXtheComponent(i-1, j, board)
```

```
            markXtheComponent(i+1, j, board)
            markXtheComponent(i, j-1, board)
            markXtheComponent(i, j+1, board)

        # Iterate through each cell in the given board.
        for i in range(0,len(board)):
            for j in range(0,len(board[0])):
                # For each 'O' cell, check if the connected
component starting from that cell is surrounded by 'X' using the
checkIfSurroundedByX function.
                if board[i][j]=='O':
                    vis = [[False]*len(board[0]) for _ in
range(0,len(board))]
                    # If the component is surrounded, mark the
entire component to 'X' using the markXtheComponent function.
                    if checkIfSurroundedByX(i,j,board,vis):
                        markXtheComponent(i,j,board)
```

**Algorithm:**

- Iterate through each cell in the given **board** using nested loops.
- For each 'O' cell encountered:
  Create a **vis** array to track visited positions and avoid redundant computations during DFS.
  Use the **checkIfSurroundedByX** function to determine if the connected component starting from the current cell is surrounded by 'X'.
  If surrounded, mark the entire connected component to 'X' using the **markXtheComponent** function.
- The algorithm thus ensures that all connected components of 'O' characters surrounded by 'X' are marked.

**Time Complexity:**
The time complexity is O(M * N), where M is the number of rows and N is the number of columns in the board. This is because, in the worst case, each cell is visited once during the DFS traversal.

**Space Complexity:**
The space complexity is O(M * N) due to the **vis** array used to track visited positions, where M is the number of rows and N is the number of columns in the board.