The question indicates that it is a question of Graph, we can solve this either by BFS or DFS, but to choose the best algorithm out of the two we will have to go step by step.

Imagine you have a rectangular board, like a chessboard, with some cells having 'X' and 'O'. Now, we need to find and mark those 'O' regions surrounded by 'X' and ensure we don't touch those 'O' cells on the borders.

Steps for the DFS approach:

- Start from the border of the board and check each cell. If the cell contains 'O', perform a Depth-First Search (DFS) to mark all connected 'O' cells as safe.
- After completing the DFS, mark all the safe 'O' cells as a special character, and say '#'.
- Traverse the entire board and convert the remaining 'O' cells to 'X' since they are surrounded.
- Finally, revert the special character '#' back to 'O' for those 'O' cells on the borders.

Steps for BFS Approach:

- While the queue is not empty, dequeue a cell and check its neighbours.
- If a neighbour is an 'O' and hasn't been visited, enqueue it and mark it as visited.
- After BFS, mark all the visited cells as safe, with a special character as we have used in the DFS approach that is '#'.
- Traverse the entire board and convert the remaining 'O' cells to 'X' since they are surrounded.
  Finally, revert the special character '#' back to 'O' for those 'O' cells on the borders.

Here the DFS traversal is more intuitive and logical as we require recursive traversing all the connected cells although we can use BFS also but here the issue of queue memory might also come. Although stack memory during recursive approach is also a concern while performing DFS but because of the intuitive nature of the question the DFS approach seems more logical than BFS.

Code:-

```
class Solution:
    def dfs(self, r, c, board, vis):
        vis[r][c] = 1
        dr = [-1, 0, 1, 0]
```

```
        dc = [0, 1, 0, -1]
        for i in range(4):
            nr = r + dr[i]
            nc = c + dc[i]
            if 0 <= nr < len(board) and 0 <= nc < len(board[0]) and board[nr][nc] == 'O' and vis[nr][nc] == 0:
                self.dfs(nr, nc, board, vis)

    def solve(self, board):
        m = len(board)
        n = len(board[0])
        vis = [[0] * n for _ in range(m)]

        for i in range(n):
            if board[0][i] == 'O' and vis[0][i] == 0:
                self.dfs(0, i, board, vis)
            if board[m - 1][i] == 'O' and vis[m - 1][i] == 0:
                self.dfs(m - 1, i, board, vis)

        for j in range(m):
            if board[j][0] == 'O' and vis[j][0] == 0:
                self.dfs(j, 0, board, vis)
            if board[j][n - 1] == 'O' and vis[j][n - 1] == 0:
                self.dfs(j, n - 1, board, vis)

        for i in range(m):
            for j in range(n):
                if vis[i][j] == 0 and board[i][j] == 'O':
                    board[i][j] = 'X'
```