PYTHON ASSIGNMENT

PEER LEARNING DOCUMENT

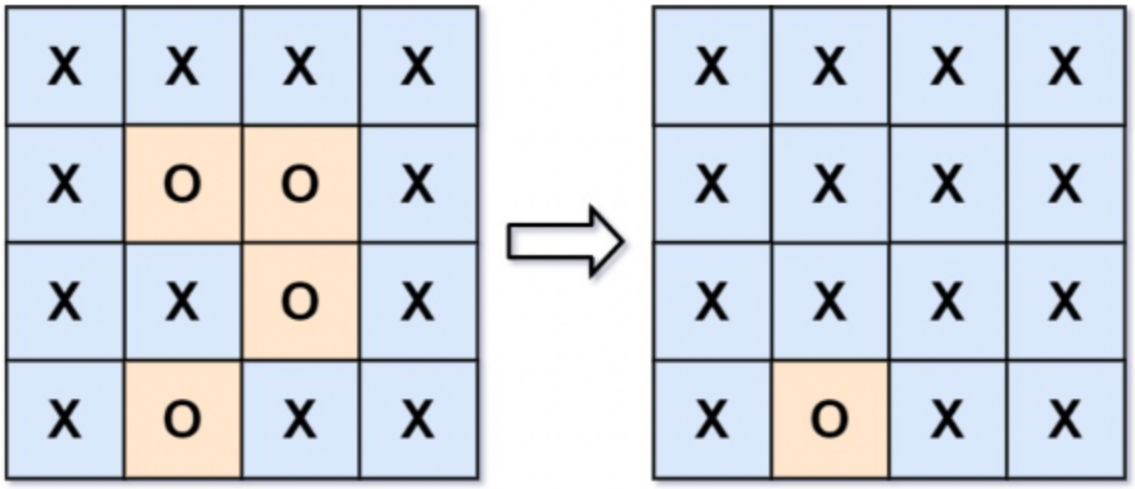
Problem Statement -

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 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.

Aswat Bisht

The Python code snippet above solves the problem of capturing a region in a board (2D array). The solution was well written by following object-oriented methods and by involving classes and objects. The class has three methods: isSafe, solve and init(Constructor).

Approach -

The method first adds all the cells on the board's boundaries to a queue. Then it starts a BFS from the cells in the queue, marking the cells that are connected to the boundaries and can be reached. Finally, all cells that are not marked are changed to 'X'.
Breadth-first search investigates every node in the graph starting with the root node. Then it chooses the closest node and investigates every new node. Any node in the graph can serve as the root node when employing BFS for traversal.
The process of searching every vertex in a tree or graph data structure is recursive. Every vertex in the graph is divided into two groups by BFS: visited and unvisited. A single node in a graph is chosen, and then all of the nodes next to that node are visited.

About Functions -

- __init__ -
  It starts by initializing the class Solution with two parameters, the number of rows and columns of the board.
- IsSafe -
  The isSafe method checks if a given cell (row, col) is within the boundaries of the board.
- Solve -
  The solve method implements the solution to the problem by using a breadth-first search (BFS) algorithm.
- Main -
  The main function takes the board's number of rows and columns as input and creates the board as a list of strings. Then, it creates an object of class Solution and calls its solve method to get the final result.

Performance and Complexity -

The code performs well, and it does explore boundaries only by not traversing every element in the grid, and performing BFS from that cell if it finds satisfying given constraints.

The time complexity of the above code is O(m * n) where m is the number of rows and n is the number of columns. This is because, in the worst-case scenario, all board elements are connected to boundary O and are visited exactly once by the BFS algorithm.

The space complexity of the code is O(m * n) as well. This is because the deque used in the BFS algorithm can hold up to m * n elements in the worst-case scenario where all elements are connected to the boundary O.

Ashish Chouhan

Approach -

The Python code snippet above solves the problem of capturing a region in a board (2D array). This code implements a DFS solution to solve the problem of changing the "O" connected to the border or connected to ‘O’ which is connected to the border to "$" and changing all other "O" to "X" and again by changing “$” to “O”.

DFS is a recursive method that searches every vertex in a graph or tree data structure. Beginning with the first node of graph G, the depth-first search (DFS) algorithm digs down until it reaches the target node, also known as the node with no children.

The DFS method can be implemented using a stack data structure due to its recursive nature. The DFS algorithm implementation procedure is comparable to that of the BFS algorithm.

About Functions -

- solve -
  The function takes the coordinates(i,j) as input arguments and does the DFS, and traverses through the given cell, marking all connected "O" to the border as "$".
- Main -
  The main code travels all borders and passes the elements to the solve function in order to perform DFS from them by taking it as root.

After marking the connected "O", the "$" is changed back to "O" and all other "O" is changed to "X". The code then outputs the modified board.

Performance and Complexity -

The code runs smoothly, and it does investigate boundaries by skipping over some elements of the grid and instead executing BFS from those cells when it finds that the provided constraints are satisfied.
The performance of the code is O(m * n) where m is the number of rows and n is the number of columns in the board. The code uses a DFS (depth-first search) algorithm to traverse the board, and each cell in the board is visited exactly once, which results in a time complexity of O(m * n).

The above code only uses a constant amount of extra space to store temporary variables, regardless of the size of the input matrix. The function "solve" performs a depth-first search (DFS) algorithm, and in DFS the extra space used is proportional to the maximum recursion depth, which is O(m * n) in this case, where m is the number of rows and n is the number of columns in the board.