PYTHON ASSIGNMENT

TEST CASE SCENARIOS DOCUMENT

Approach -

The code is an implementation of the problem to solve surrounding regions in a 2D board. The board is represented by a list of lists where the character 'X' represents an obstacle, 'O' represents a surrounded region, and 'Y' represents a region that is not surrounded. The code implements two different approaches to solve this problem, one using BFS (Breadth-First Search) and the other using DFS (Depth-First Search).

In the BFS approach, the code first checks if the board is empty. If it's not empty, the code calculates the dimensions of the board and then adds cells with the entry 'O' in a queue. The BFS implementation uses the queue and pop elements from the left end of the queue and explores the vertices. If an 'O' is encountered, it's marked as 'Y' and the program explores the vertices in its four directions (up, down, left, and right) and repeats the same process. Finally, the code replaces 'Y' with 'O' and all other elements with 'X'.

In the DFS approach, the code works similarly to the BFS approach. The code first checks if the board is empty, calculates the dimensions of the board, and implements a recursive function 'dfs' to explore the vertices. The base case of the 'dfs' function is if the vertices are outside the board or if the element is not 'O'. If the base case is not met, the code replaces the element with 'Y' and explores its neighbors in the four directions (up, down, left, and right). Finally, the code replaces 'Y' with 'O' and all other elements with 'X'.

Test Case - 1(test_empty_list)

- **Input & output -**
    - board = []
    - expected_output = []
- **Explanation -** This is the base case, and the same board is returned.

Test Case - 2(test_given_input)

- **Input & output -**
    - board = [["X","X","X","X"],["X","O","O","X"],["X","X","O","X"],["X","O","X","X"]]
    - expected_output = [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","O","X","X"]]
- **Explanation -** In both BFS and DFS, the root node is board[4][1] element, and we perform DFS or BFS from the root but no node in its four neighbors is ‘O’ and connected with the border 'O', only this 'O' gets converted to 'Y' and remaining are left as they are. Finally, the "O"s are changed to "X"s and the "Y"s are converted to "O"s, producing the desired results. The above input gives the expected output as above with the code.

Test Case - 3(test_random_input)

- **Input & output -**
    - board = [["X","O","X","O"],["O","X","O","X"],["X","O","X","O"],["O","X","O","X"]]
    - expected_output = [["X","O","X","O"],["O","X","X","X"],["X","X","X","O"],["O","X","O","X"]]
- **Explanation -** In both BFS and DFS, the root node is board[0][1] element, and we perform DFS or BFS from the root but no node in its four neighbors is ‘O’ and connected with the border 'O', only this 'O' gets converted to 'Y' and moves to next ‘O’ in the border which is board[0][3] and same as earlier no ‘O’ is connected to it so moves to next, the process continues like this as no border ‘O’ is connected with internal ‘O’s so all the ‘O’s in the border are converted to ‘Y and internal ‘O’s are remained as it is. Finally, the "O"s are changed to "X"s and the "Y"s are converted to "O"s, producing the desired results. The above input gives the expected output as above with the code.

Test Case - 4(test_no_border_Os)

- **Input & output -**
    - board = [["X","X","X","X"],["X","O","X","X"],["X","X","X","X"],["X","X","X","X"]]
    - expected_output = [["X","X","X","X"],["X","X","X","X"],["X","X","X","X"],["X","X","X","X"]]
- **Explanation -** We do not need to traverse the graph because there is no border 'O', and all internal 'O's are changed to 'X' since there is no boundary 'O'.

Test Case - 5(test_single_elementO)

- **Input & output -**
    - board = [["O"]]
    - expected_output = [["O"]]
- **Explanation -** The board is returned in its original state because there is just one element on it.

Test Case - 6(test_single_elementX)

- **Input & output -**
    - board = [["X"]]
    - expected_output = [["X"]]
- **Explanation -** The board is returned in its original state because there is just one element on it.

Test Case - 7(test_all_elementsX)

- **Input & output -**
    - board = [["X","X","X"],["X","X","X"],["X","X","X"]]
    - expected_output = [["X","X","X"],["X","X","X"],["X","X","X"]]

Explanation - The board is returned in its original state because there is no ‘O’s in it.

Test Case - 8(test_all_elementsO)

- **Input & output -**
    - board = [["O","O","O"],["O","O","O"],["O","O","O"]]
    - expected_output = [["O","O","O"],["O","O","O"],["O","O","O"]]

Explanation - The board is returned in its original state because there is no ‘X’s in it.

Test Case - 9(test_invalid_entries)

- **Input & output -**
    - board = [["C", "X"], ["X", "X"]]
    - expected_output - Error statement

Explanation - An error statement is returned as it has invalid entries(i.e characters other than ‘X’ and ‘O’).

Test Case - 10(test_invalid_shape)

- **Input & output -**
    - board = [["X","X","O"],["X","X"],["X"]]
    - expected_output - Error statement

Explanation - An error statement is returned as it has invalid shape(i.e all rows does not have equal no of elements).