**130. Surrounded Regions**

<https://leetcode.com/problems/surrounded-regions/>

1. **Listen**

**Problem Statement:**

Given an m x 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:**

m x n matrix board filled with characters

cells can contain either 'X' and 'O'

**Goal:**

*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.

Note 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

**Return:**

the input matrix should be edited in-placed, and nothing is returned.

1. **Example**

**Example 1:**

A picture containing text, shoji, crossword puzzle, clipart

Description automatically generated

**Input:** board = [["X","X","X","X"],

["X","O","O","X"],

["X","X","O","X"],

["X","O","X","X"]]

**Output:** board = [["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.

**Example 2:**

**Input:** board = [["X"]]

**Output:** board = [["X"]]

**Constraints:**

* m == board.length
* n == board[i].length
* 1 <= m, n <= 200
* board[i][j] is 'X' or 'O'

**Test Cases:**

* 1x1 or 1x2 or 2x1 matrix (no elements should be flipped)
* All O’s are flipped
* No O’s are flipped

1. **Brute Force / Optimize / Walkthrough**

**Solution 1: Time = O(MN), Space = O(MN)**

We know a few things from the problem statements:

1. Regions cannot be connected diagonally

A picture containing text, shoji, crossword puzzle, clipart

Description automatically generated



1. Only regions that are connected horizontally or vertically on all four sides (4-directionally) surrounded by ‘X’s are flipped from ‘O’s to ‘X’s

A picture containing text, shoji, crossword puzzle, clipart

Description automatically generated



1. An ‘O’ should not be flipped if
   1. It is on the border, or
   2. It is adjacent to an ‘O’ that should not be flipped

A picture containing text, shoji, crossword puzzle, clipart

Description automatically generated



**First Idea**

* This sounds like we can run a DFS traversal on the matrix, with the constraint of avoiding any ‘O’s that are connected to the sides of the board.
* This is possible but is actually a complicated solution.
* There are many ways to solve this problem, but there is clever, more simple approach we can take.

**Reverse Thinking**

* This clever solution has to do with reversing the way you think about the problem, what some people call reverse thinking.
* Say you had a piece of paper divided into two regions. The problem statement you are given is to return only the ‘A’ region. This is the simple way to talk about it.

|  |  |
| --- | --- |
| **A** | **B** |

* The reverse way to think about it would be to say, ‘give me everything in this paper except for region ‘B’.
* While cumbersome, this alternative way of thinking can often be helpful when approaching a problem.

**Reverse Thinking Applied**

* We can apply this same way of thinking towards the problem at hand.
* The problem statement requires us to capture the surrounded regions.
* The opposite way of saying this would be to capture all everything except the unsurrounded regions.
* Specifically, capture all of the ‘O’ that are on or adjacent to an ‘O’ on the border.

A picture containing text, shoji, crossword puzzle, clipart

Description automatically generated

**DFS**

* But how do we know a region is not surrounded?
* We look for **any** ‘O’s that are directly connected to the border.

A picture containing text, shoji, crossword puzzle, clipart

Description automatically generated



* Then, if we can check if there are any ‘O’s adjacent/connected to these border ‘O’s by running DFS each time we encounter a border ‘O’.
* Therefore, were going to scan through every border cell. We know if we find any ‘O’s that it will be an unsurrounded region (which is our main goal to find).
* We know that we DON’T want to flip unsurrounded regions. So once we find these unsurrounded ‘O’ regions, how can we ensure they don’t get flipped?
* We reassign these unsurrounded regions to a temporary value. It can be any value, but we are going to change the ‘O’s to ‘T’s.
* If there are adjacent ‘O’s not on the border, they are part of the unsurrounded region. We therefore run a DFS each time we encounter an ‘O’ on the border, and chage the ‘O’, and all adjacent ‘O’s to ‘T’s.
* These ‘T’s essentially act as a marker for unsurrounded regions.

**Iterate Over The Matrix**

A picture containing text, shoji, crossword puzzle, clipart

Description automatically generated



* We then iterate over the matrix, changing any ‘O’s we see to ‘X’s.
* We can safely do this, because now we are sure any ‘O’s in unsurrounded regions are ‘T’s.
* Therefore, the only ‘O’s on the board will be those in surrounded regions.
* We also change any ‘T’s we see to ‘O’s to restore the original state of unsurrounded regions, because they are not supposed to be flipped.

**Runtime**

We will iterate over the entire matrix at most 3 times. Therefore, the time complexity will be O(3nm) = O(nm) where n and m are the dimensions of the board.

Since we use recursive DFS, the time complexity can be up to O(2N + 2M) = O(N + M)

1. **Implement**
2. **Test**