Problem 130: Surrounded Regions

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an

m x n

matrix

board

containing

letters

'X'

and

'O'

,

capture regions

that are

surrounded

:

Connect

: A cell is connected to adjacent cells horizontally or vertically.

Region

: To form a region

connect every

'O'

cell.

Surround

: The region is surrounded with

'X'

cells if you can

connect the region

with

'X'

cells and none of the region cells are on the edge of the

board

.

To capture a

surrounded region

, replace all

'O'

s with

'X'

s

in-place

within the original board. You do not need to return anything.

Example 1:

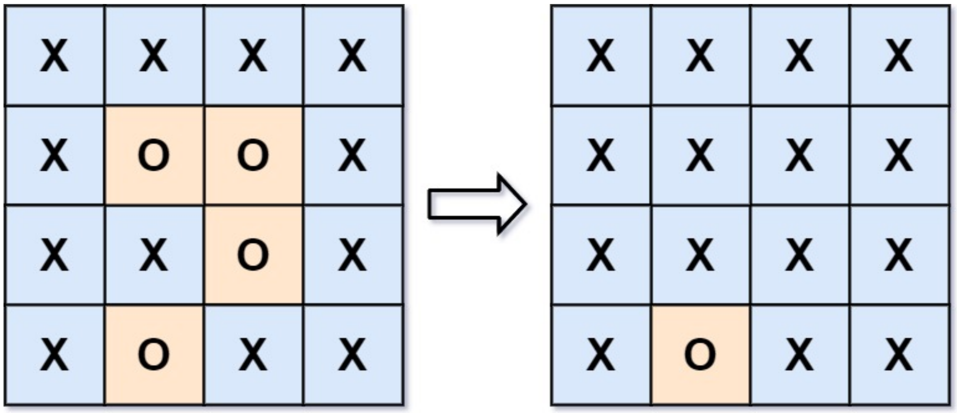
Input:

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:



X	X	X	X
X	O	O	X
X	X	O	X
X	O	X	X

X	X	X	X
X	X	X	X
X	X	X	X
X	O	X	X

In the above diagram, the bottom region is not captured because it is on the edge of the board and cannot be surrounded.

Example 2:

Input:

board = [["X"]]

Output:

[["X"]]

Constraints:

m == board.length

n == board[i].length

1 <= m, n <= 200

board[i][j]

is

'X'

or

'O'

.

Code Snippets

C++:

```
class Solution {
public:
void solve(vector<vector<char>>& board) {
}
};
```

Java:

```
class Solution {
public void solve(char[][] board) {

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {character[][]} board
* @return {void} Do not return anything, modify board in-place instead.
*/
var solve = function(board) {

};
```

TypeScript:

```
/**
Do not return anything, modify board in-place instead.
```

```
*/
function solve(board: string[][]): void {

};
```

C#:

```
public class Solution {
public void Solve(char[][] board) {

}
}
```

C:

```
void solve(char** board, int boardSize, int* boardColSize) {

}
```

Go:

```
func solve(board [][]byte) {

}
```

Kotlin:

```
class Solution {
fun solve(board: Array<CharArray>): Unit {

}
}
```

Swift:

```
class Solution {
func solve(_ board: inout [[Character]]) {

}
}
```

Rust:

```
impl Solution {
pub fn solve(board: &mut Vec<Vec<char>>) {

}
}
```

Ruby:

```
# @param {Character[][]} board
# @return {Void} Do not return anything, modify board in-place instead.
def solve(board)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $board
* @return NULL
*/
function solve(&$board) {

}
}
```

Dart:

```
class Solution {
void solve(List<List<String>> board) {

}
}
```

Scala:

```
object Solution {
def solve(board: Array[Array[Char]]): Unit = {

}
}
```

Solutions

C++ Solution:

```
/*
* Problem: Surrounded Regions
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
void solve(vector<vector<char>>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Surrounded Regions
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public void solve(char[][] board) {

}
}
```

Python3 Solution:

```
"""
Problem: Surrounded Regions
Difficulty: Medium
Tags: array, graph, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def solve(self, board: List[List[str]]) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Surrounded Regions
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {character[][]} board
* @return {void} Do not return anything, modify board in-place instead.
*/
var solve = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Surrounded Regions
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
Do not return anything, modify board in-place instead.
*/
function solve(board: string[][]): void {

};
```

C# Solution:

```
/*
* Problem: Surrounded Regions
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public void Solve(char[][] board) {

}
}
```

C Solution:

```
/*
* Problem: Surrounded Regions
* Difficulty: Medium
* Tags: array, graph, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

void solve(char** board, int boardSize, int* boardColSize) {

}
```

Go Solution:

```
// Problem: Surrounded Regions
// Difficulty: Medium
// Tags: array, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func solve(board [][]byte) {

}
```

Kotlin Solution:

```
class Solution {
fun solve(board: Array<CharArray>): Unit {

}
}
```

Swift Solution:

```
class Solution {
func solve(_ board: inout [[Character]]) {

}
}
```

Rust Solution:

```
// Problem: Surrounded Regions
// Difficulty: Medium
// Tags: array, graph, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn solve(board: &mut Vec<Vec<char>>) {

}
}
```

Ruby Solution:

```
# @param {Character[][]} board
# @return {Void} Do not return anything, modify board in-place instead.
def solve(board)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $board
* @return NULL
*/
function solve(&$board) {

}
}
```

Dart Solution:

```
class Solution {
void solve(List<List<String>> board) {

}
}
```

Scala Solution:

```
object Solution {
def solve(board: Array[Array[Char]]): Unit = {

}
}
```