Problem 794: Valid Tic-Tac-Toe State

Problem Information

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

Problem Description

Given a Tic-Tac-Toe board as a string array

board

, return

true

if and only if it is possible to reach this board position during the course of a valid tic-tac-toe game.

The board is a

3 x 3

array that consists of characters

' '

,

'X'

, and

'O'

. The

' '

character represents an empty square.

Here are the rules of Tic-Tac-Toe:

Players take turns placing characters into empty squares

' '

.

The first player always places

'X'

characters, while the second player always places

'O'

characters.

'X'

and

'O'

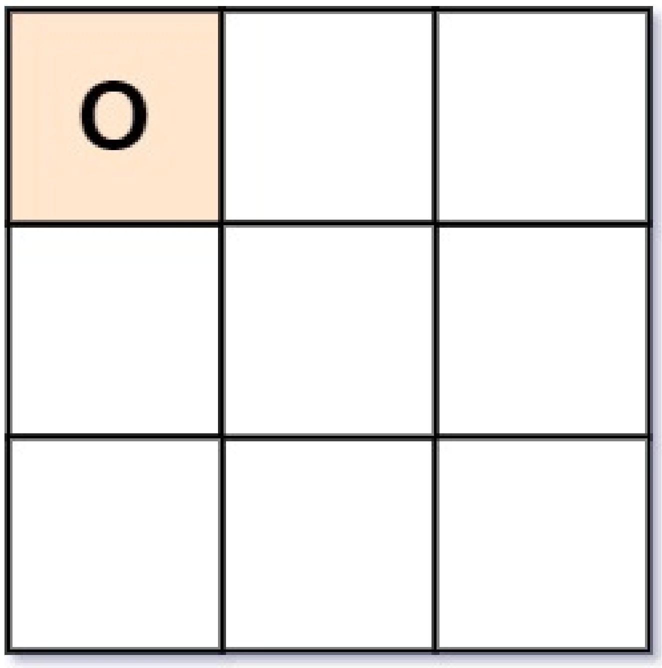
characters are always placed into empty squares, never filled ones.

The game ends when there are three of the same (non-empty) character filling any row, column, or diagonal.

The game also ends if all squares are non-empty.

No more moves can be played if the game is over.

Example 1:



Input:

board = ["O "," "," "]

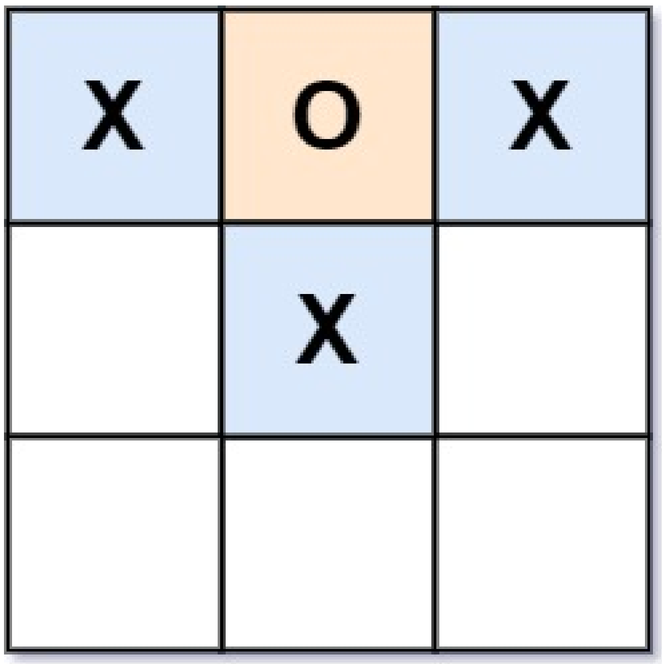
Output:

false

Explanation:

The first player always plays "X".

Example 2:



Input:

board = ["XOX"," X "," "]

Output:

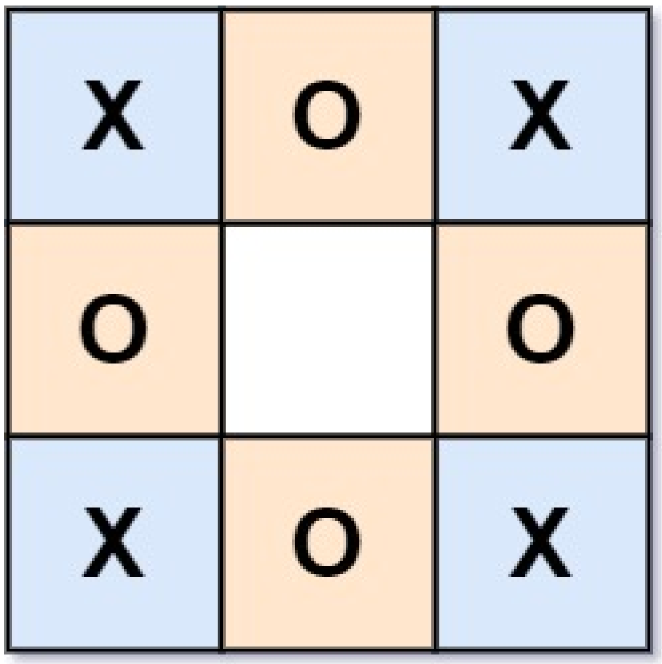
false

Explanation:

Players take turns making moves.

Example 3:

'



Input:

board = ["XOX","O O","XOX"]

Output:

true

Constraints:

board.length == 3

board[i].length == 3

board[i][j]

is either

'X'

,

'O'

, or

' '

.

Code Snippets

C++:

```
class Solution {
public:
bool validTicTacToe(vector<string>& board) {

}
};
```

Java:

```
class Solution {
public boolean validTicTacToe(String[] board) {

}
}
```

Python3:

```
class Solution:
def validTicTacToe(self, board: List[str]) -> bool:
```

Python:

```
class Solution(object):
def validTicTacToe(self, board):
"""
:type board: List[str]
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string[]} board
* @return {boolean}
*/
var validTicTacToe = function(board) {

};
```

TypeScript:

```
function validTicTacToe(board: string[]): boolean {

};
```

C#:

```
public class Solution {
public bool ValidTicTacToe(string[] board) {

}
}
```

C:

```
bool validTicTacToe(char** board, int boardSize) {

}
```

Go:

```
func validTicTacToe(board []string) bool {

}
```

Kotlin:

```
class Solution {
fun validTicTacToe(board: Array<String>): Boolean {

}
}
```

Swift:

```
class Solution {
func validTicTacToe(_ board: [String]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn valid_tic_tac_toe(board: Vec<String>) -> bool {

}
}
```

Ruby:

```
# @param {String[]} board
# @return {Boolean}
def valid_tic_tac_toe(board)

end
```

PHP:

```
class Solution {

/**
* @param String[] $board
* @return Boolean
*/
function validTicTacToe($board) {

}
}
```

Dart:

```
class Solution {
bool validTicTacToe(List<String> board) {

}
}
```

Scala:

```
object Solution {
def validTicTacToe(board: Array[String]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec valid_tic_tac_toe(board :: [String.t]) :: boolean
def valid_tic_tac_toe(board) do

end
end
```

Erlang:

```
-spec valid_tic_tac_toe(Board :: [unicode:unicode_binary()]) -> boolean().
valid_tic_tac_toe(Board) ->
.
```

Racket:

```
(define/contract (valid-tic-tac-toe board)
(-> (listof string?) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Valid Tic-Tac-Toe State
* Difficulty: Medium
* Tags: array, string
*
* 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:
bool validTicTacToe(vector<string>& board) {

}
};
```

Java Solution:

```
/**
* Problem: Valid Tic-Tac-Toe State
* Difficulty: Medium
* Tags: array, string
*
* 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 boolean validTicTacToe(String[] board) {

}
}
```

Python3 Solution:

```
"""
Problem: Valid Tic-Tac-Toe State
Difficulty: Medium
Tags: array, string

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 validTicTacToe(self, board: List[str]) -> bool:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def validTicTacToe(self, board):
"""
:type board: List[str]
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Valid Tic-Tac-Toe State
* Difficulty: Medium
* Tags: array, string
*
* 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 {string[]} board
* @return {boolean}
*/
var validTicTacToe = function(board) {

};
```

TypeScript Solution:

```
/**
* Problem: Valid Tic-Tac-Toe State
* Difficulty: Medium
* Tags: array, string
*
* 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
```

```
*/

function validTicTacToe(board: string[]): boolean {

};
```

C# Solution:

```
/*
* Problem: Valid Tic-Tac-Toe State
* Difficulty: Medium
* Tags: array, string
*
* 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 bool ValidTicTacToe(string[] board) {

}
}
```

C Solution:

```
/*
* Problem: Valid Tic-Tac-Toe State
* Difficulty: Medium
* Tags: array, string
*
* 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
*/

bool validTicTacToe(char** board, int boardSize) {

}
```

Go Solution:

```
// Problem: Valid Tic-Tac-Toe State
// Difficulty: Medium
// Tags: array, string
//
// 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 validTicTacToe(board []string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun validTicTacToe(board: Array<String>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func validTicTacToe(_ board: [String]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Valid Tic-Tac-Toe State
// Difficulty: Medium
// Tags: array, string
//
// 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 valid_tic_tac_toe(board: Vec<String>) -> bool {

}
}
```

Ruby Solution:

```
# @param {String[]} board
# @return {Boolean}
def valid_tic_tac_toe(board)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $board
* @return Boolean
*/
function validTicTacToe($board) {

}
}
```

Dart Solution:

```
class Solution {
bool validTicTacToe(List<String> board) {

}
}
```

Scala Solution:

```
object Solution {
def validTicTacToe(board: Array[String]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec valid_tic_tac_toe(board :: [String.t]) :: boolean
def valid_tic_tac_toe(board) do

end
end
```

Erlang Solution:

```
-spec valid_tic_tac_toe(Board :: [unicode:unicode_binary()]) -> boolean().
valid_tic_tac_toe(Board) ->
.
```

Racket Solution:

```
(define/contract (valid-tic-tac-toe board)
(-> (listof string?) boolean?)
)
```