Problem 3127: Make a Square with the Same Color

Problem Information

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

Problem Description

You are given a 2D matrix

grid

of size

3 x 3

consisting only of characters

'B'

and

'W'

. Character

'W'

represents the white color

, and character

'B'

represents the black color

.

Your task is to change the color of

at most one

cell

so that the matrix has a

2 x 2

square where all cells are of the same color.

Return

true

if it is possible to create a

2 x 2

square of the same color, otherwise, return

false

.

Example 1:

Input:

grid = [["B","W","B"],["B","W","W"],["B","W","B"]]

Output:

true

Explanation:

It can be done by changing the color of the

grid[0][2]

.

Example 2:

Input:

grid = [["B","W","B"],["W","B","W"],["B","W","B"]]

Output:

false

Explanation:

It cannot be done by changing at most one cell.

Example 3:

Input:

grid = [["B","W","B"],["B","W","W"],["B","W","W"]]

Output:

true

Explanation:

The

grid

already contains a

2 x 2

square of the same color.

Constraints:

grid.length == 3

grid[i].length == 3

grid[i][j]

is either

'W'

or

'B'

.

Code Snippets

C++:

```
class Solution {
public:
bool canMakeSquare(vector<vector<char>>& grid) {

}
};
```

Java:

```
class Solution {
public boolean canMakeSquare(char[][] grid) {
}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {character[][]} grid
* @return {boolean}
*/
var canMakeSquare = function(grid) {

};
```

TypeScript:

```
function canMakeSquare(grid: string[][]): boolean {

};
```

C#:

```
public class Solution {
public bool CanMakeSquare(char[][] grid) {

}
}
```

C:

```
bool canMakeSquare(char** grid, int gridSize, int* gridColSize) {

}
```

Go:

```
func canMakeSquare(grid [][]byte) bool {

}
```

Kotlin:

```
class Solution {
fun canMakeSquare(grid: Array<CharArray>): Boolean {

}
}
```

Swift:

```
class Solution {
func canMakeSquare(_ grid: [[Character]]) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn can_make_square(grid: Vec<Vec<char>>) -> bool {

}
}
```

Ruby:

```
# @param {Character[][]} grid
# @return {Boolean}
def can_make_square(grid)

end
```

PHP:

```
class Solution {

/**
* @param String[][] $grid
* @return Boolean
*/
function canMakeSquare($grid) {

}
}
```

Dart:

```
class Solution {
bool canMakeSquare(List<List<String>> grid) {

}
}
```

Scala:

```
object Solution {
def canMakeSquare(grid: Array[Array[Char]]): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec can_make_square(grid :: [[char]]) :: boolean
def can_make_square(grid) do

end
end
```

Erlang:

```
-spec can_make_square(Grid :: [[char()]]) -> boolean().
can_make_square(Grid) ->
.
```

Racket:

```
(define/contract (can-make-square grid)
(-> (listof (listof char?)) boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make a Square with the Same Color
* Difficulty: Easy
* Tags: array
*
* 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 canMakeSquare(vector<vector<char>>& grid) {

}
};
```

Java Solution:

```
/**
* Problem: Make a Square with the Same Color
* Difficulty: Easy
* Tags: array
*
* 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 canMakeSquare(char[][] grid) {
}
}
```

Python3 Solution:

```
"""
Problem: Make a Square with the Same Color
Difficulty: Easy
Tags: array

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 canMakeSquare(self, grid: List[List[str]]) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Make a Square with the Same Color
* Difficulty: Easy
* Tags: array
*
* 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[][]} grid
* @return {boolean}
*/
var canMakeSquare = function(grid) {

};
```

TypeScript Solution:

```
/**
* Problem: Make a Square with the Same Color
* Difficulty: Easy
* Tags: array
*
* 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 canMakeSquare(grid: string[][]): boolean {

};
```

C# Solution:

```
/*
* Problem: Make a Square with the Same Color
* Difficulty: Easy
* Tags: array
*
* 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 CanMakeSquare(char[][] grid) {

}
}
```

C Solution:

```c
/*
* Problem: Make a Square with the Same Color
* Difficulty: Easy
* Tags: array
*
* 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 canMakeSquare(char** grid, int gridSize, int* gridColSize) {

}
```

Go Solution:

```go
// Problem: Make a Square with the Same Color
// Difficulty: Easy
// Tags: array
//
// 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 canMakeSquare(grid [][]byte) bool {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun canMakeSquare(grid: Array<CharArray>): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func canMakeSquare(_ grid: [[Character]]) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Make a Square with the Same Color
// Difficulty: Easy
// Tags: array
//
// 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 can_make_square(grid: Vec<Vec<char>>) -> bool {

}
}
```

Ruby Solution:

```
# @param {Character[][]} grid
# @return {Boolean}
def can_make_square(grid)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[][] $grid
* @return Boolean
*/
function canMakeSquare($grid) {

}
}
```

Dart Solution:

```
class Solution {
bool canMakeSquare(List<List<String>> grid) {

}
}
```

Scala Solution:

```
object Solution {
def canMakeSquare(grid: Array[Array[Char]]): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec can_make_square(grid :: [[char]]) :: boolean
def can_make_square(grid) do

end
end
```

Erlang Solution:

```
-spec can_make_square(Grid :: [[char()]]) -> boolean().
can_make_square(Grid) ->
.
```

Racket Solution:

```
(define/contract (can-make-square grid)
(-> (listof (listof char?)) boolean?)
)
```