Problem 2038: Remove Colored Pieces if Both Neighbors are the Same Color

Problem Information

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

Problem Description

There are

n

pieces arranged in a line, and each piece is colored either by

'A'

or by

'B'

. You are given a string

colors

of length

n

where

colors[i]

is the color of the

i

th

piece.

Alice and Bob are playing a game where they take

alternating turns

removing pieces from the line. In this game, Alice moves

first

.

Alice is only allowed to remove a piece colored

'A'

if

both its neighbors

are also colored

'A'

. She is

not allowed

to remove pieces that are colored

'B'

.

Bob is only allowed to remove a piece colored

'B'

if

both its neighbors

are also colored

'B'

. He is

not allowed

to remove pieces that are colored

'A'

.

Alice and Bob

cannot

remove pieces from the edge of the line.

If a player cannot make a move on their turn, that player

loses

and the other player

wins

.

Assuming Alice and Bob play optimally, return true

if Alice wins, or return

false

if Bob wins

.

Example 1:

Input:

colors = "AAABABB"

Output:

true

Explanation:

A

A

ABABB -> AABABB Alice moves first. She removes the second 'A' from the left since that is the only 'A' whose neighbors are both 'A'.

Now it's Bob's turn. Bob cannot make a move on his turn since there are no 'B's whose neighbors are both 'B'. Thus, Alice wins, so return true.

Example 2:

Input:

colors = "AA"

Output:

false

Explanation:

Alice has her turn first. There are only two 'A's and both are on the edge of the line, so she cannot move on her turn. Thus, Bob wins, so return false.

Example 3:

Input:

colors = "ABBBBBBBAAA"

Output:

false

Explanation:

ABBBBBBBA

A

A -> ABBBBBBBAA Alice moves first. Her only option is to remove the second to last 'A' from the right.

ABBBB

B

BBAA -> ABBBBBBAA Next is Bob's turn. He has many options for which 'B' piece to remove. He can pick any.

On Alice's second turn, she has no more pieces that she can remove. Thus, Bob wins, so return false.

Constraints:

1 <= colors.length <= 10

5

colors

consists of only the letters

'A'

and

'B'

Code Snippets

C++:

```
class Solution {
public:
bool winnerOfGame(string colors) {

}
};
```

Java:

```
class Solution {
public boolean winnerOfGame(String colors) {

}
}
```

Python3:

```
class Solution:
def winnerOfGame(self, colors: str) -> bool:
```

Python:

```
class Solution(object):
def winnerOfGame(self, colors):
```

```
"""
:type colors: str
:rtype: bool
"""
```

JavaScript:

```
/**
* @param {string} colors
* @return {boolean}
*/
var winnerOfGame = function(colors) {

};
```

TypeScript:

```
function winnerOfGame(colors: string): boolean {

};
```

C#:

```
public class Solution {
public bool WinnerOfGame(string colors) {

}
}
```

C:

```
bool winnerOfGame(char* colors) {

}
```

Go:

```
func winnerOfGame(colors string) bool {

}
```

Kotlin:

```
class Solution {
fun winnerOfGame(colors: String): Boolean {

}
}
```

Swift:

```
class Solution {
func winnerOfGame(_ colors: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn winner_of_game(colors: String) -> bool {

}
}
```

Ruby:

```
# @param {String} colors
# @return {Boolean}
def winner_of_game(colors)

end
```

PHP:

```
class Solution {

/**
* @param String $colors
* @return Boolean
*/
function winnerOfGame($colors) {

}
}
```

Dart:

```
class Solution {
bool winnerOfGame(String colors) {

}
}
```

Scala:

```
object Solution {
def winnerOfGame(colors: String): Boolean = {

}
}
```

Elixir:

```
defmodule Solution do
@spec winner_of_game(colors :: String.t) :: boolean
def winner_of_game(colors) do

end
end
```

Erlang:

```
-spec winner_of_game(Colors :: unicode:unicode_binary()) -> boolean().
winner_of_game(Colors) ->
.
```

Racket:

```
(define/contract (winner-of-game colors)
(-> string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Colored Pieces if Both Neighbors are the Same Color
* Difficulty: Medium
* Tags: string, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
bool winnerOfGame(string colors) {

}
};
```

Java Solution:

```
/**
* Problem: Remove Colored Pieces if Both Neighbors are the Same Color
* Difficulty: Medium
* Tags: string, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public boolean winnerOfGame(String colors) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Colored Pieces if Both Neighbors are the Same Color
Difficulty: Medium
Tags: string, greedy, math
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def winnerOfGame(self, colors: str) -> bool:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def winnerOfGame(self, colors):
"""
:type colors: str
:rtype: bool
"""
```

JavaScript Solution:

```
/**
* Problem: Remove Colored Pieces if Both Neighbors are the Same Color
* Difficulty: Medium
* Tags: string, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} colors
* @return {boolean}
*/
var winnerOfGame = function(colors) {

};
```

TypeScript Solution:

```
/**
* Problem: Remove Colored Pieces if Both Neighbors are the Same Color
* Difficulty: Medium
* Tags: string, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function winnerOfGame(colors: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Remove Colored Pieces if Both Neighbors are the Same Color
* Difficulty: Medium
* Tags: string, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public bool WinnerOfGame(string colors) {

}
}
```

C Solution:

```
/*
* Problem: Remove Colored Pieces if Both Neighbors are the Same Color
* Difficulty: Medium
* Tags: string, greedy, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

bool winnerOfGame(char* colors) {

}
```

Go Solution:

```
// Problem: Remove Colored Pieces if Both Neighbors are the Same Color
// Difficulty: Medium
// Tags: string, greedy, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func winnerOfGame(colors string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun winnerOfGame(colors: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func winnerOfGame(_ colors: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Remove Colored Pieces if Both Neighbors are the Same Color
// Difficulty: Medium
// Tags: string, greedy, math
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn winner_of_game(colors: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} colors
# @return {Boolean}
def winner_of_game(colors)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $colors
* @return Boolean
*/
function winnerOfGame($colors) {

}
}
```

Dart Solution:

```
class Solution {
bool winnerOfGame(String colors) {

}
}
```

Scala Solution:

```
object Solution {
def winnerOfGame(colors: String): Boolean = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec winner_of_game(colors :: String.t) :: boolean
def winner_of_game(colors) do

end
end
```

Erlang Solution:

```
-spec winner_of_game(Colors :: unicode:unicode_binary()) -> boolean().
winner_of_game(Colors) ->
.
```

Racket Solution:

```
(define/contract (winner-of-game colors)
(-> string? boolean?)
)
```