Problem 488: Zuma Game

Problem Information

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

Problem Description

You are playing a variation of the game Zuma.

In this variation of Zuma, there is a

single row

of colored balls on a board, where each ball can be colored red

'R'

, yellow

'Y'

, blue

'B'

, green

'G'

, or white

'W'

. You also have several colored balls in your hand.

Your goal is to

clear all

of the balls from the board. On each turn:

Pick

any

ball from your hand and insert it in between two balls in the row or on either end of the row.

If there is a group of

three or more consecutive balls

of the

same color

, remove the group of balls from the board.

If this removal causes more groups of three or more of the same color to form, then continue removing each group until there are none left.

If there are no more balls on the board, then you win the game.

Repeat this process until you either win or do not have any more balls in your hand.

Given a string

board

, representing the row of balls on the board, and a string

hand

, representing the balls in your hand, return

the

minimum

number of balls you have to insert to clear all the balls from the board. If you cannot clear all the balls from the board using the balls in your hand, return

-1

.

Example 1:

Input:

board = "WRRBBW", hand = "RB"

Output:

-1

Explanation:

It is impossible to clear all the balls. The best you can do is: - Insert 'R' so the board becomes WRR

R

BBW. W

RRR

BBW -> WBBW. - Insert 'B' so the board becomes WBB

B

W. W

BBB

W -> WW. There are still balls remaining on the board, and you are out of balls to insert.

Example 2:

Input:

board = "WWRRBBWW", hand = "WRBRW"

Output:

2

Explanation:

To make the board empty: - Insert 'R' so the board becomes WWRR

R

BBWW. WW

RRR

BBWW -> WWBBWW. - Insert 'B' so the board becomes WWBB

B

WW. WW

BBB

WW ->

WWWW

-> empty. 2 balls from your hand were needed to clear the board.

Example 3:

Input:

board = "G", hand = "GGGGG"

Output:

2

Explanation:

To make the board empty: - Insert 'G' so the board becomes G

G

. - Insert 'G' so the board becomes GG

G

.

GGG

-> empty. 2 balls from your hand were needed to clear the board.

Constraints:

1 <= board.length <= 16

1 <= hand.length <= 5

board

and

hand

consist of the characters

'R'

,

'Y'

,

'B'

,

'G'

, and

'W'

.

The initial row of balls on the board will

not

have any groups of three or more consecutive balls of the same color.

Code Snippets

C++:

```
class Solution {
public:
int findMinStep(string board, string hand) {

}
};
```

Java:

```
class Solution {
public int findMinStep(String board, String hand) {

}
}
```

Python3:

```
class Solution:
def findMinStep(self, board: str, hand: str) -> int:
```

Python:

```
class Solution(object):
def findMinStep(self, board, hand):
"""
:type board: str
:type hand: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} board
* @param {string} hand
* @return {number}
*/
var findMinStep = function(board, hand) {

};
```

TypeScript:

```
function findMinStep(board: string, hand: string): number {

};
```

C#:

```
public class Solution {
public int FindMinStep(string board, string hand) {
}
}
```

C:

```
int findMinStep(char* board, char* hand) {

}
```

Go:

```
func findMinStep(board string, hand string) int {

}
```

Kotlin:

```
class Solution {
fun findMinStep(board: String, hand: String): Int {

}
}
```

Swift:

```
class Solution {
func findMinStep(_ board: String, _ hand: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn find_min_step(board: String, hand: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} board
# @param {String} hand
# @return {Integer}
def find_min_step(board, hand)

end
```

PHP:

```
class Solution {

/**
* @param String $board
* @param String $hand
* @return Integer
*/
function findMinStep($board, $hand) {

}
}
```

Dart:

```
class Solution {
int findMinStep(String board, String hand) {

}
}
```

Scala:

```
object Solution {
def findMinStep(board: String, hand: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec find_min_step(board :: String.t, hand :: String.t) :: integer
def find_min_step(board, hand) do
```

```
end
end
```

Erlang:

```
-spec find_min_step(Board :: unicode:unicode_binary(), Hand ::
unicode:unicode_binary()) -> integer().
find_min_step(Board, Hand) ->
.
```

Racket:

```
(define/contract (find-min-step board hand)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Zuma Game
* Difficulty: Hard
* Tags: string, dp, search, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int findMinStep(string board, string hand) {

}
};
```

Java Solution:

```
/**
* Problem: Zuma Game
* Difficulty: Hard
* Tags: string, dp, search, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int findMinStep(String board, String hand) {

}
}
```

Python3 Solution:

```
"""
Problem: Zuma Game
Difficulty: Hard
Tags: string, dp, search, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def findMinStep(self, board: str, hand: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def findMinStep(self, board, hand):
"""
:type board: str
:type hand: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Zuma Game
* Difficulty: Hard
* Tags: string, dp, search, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} board
* @param {string} hand
* @return {number}
*/
var findMinStep = function(board, hand) {

};
```

TypeScript Solution:

```
/**
* Problem: Zuma Game
* Difficulty: Hard
* Tags: string, dp, search, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function findMinStep(board: string, hand: string): number {

};
```

C# Solution:

```
/*
* Problem: Zuma Game
* Difficulty: Hard
* Tags: string, dp, search, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int FindMinStep(string board, string hand) {

}
}
```

C Solution:

```
/*
* Problem: Zuma Game
* Difficulty: Hard
* Tags: string, dp, search, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int findMinStep(char* board, char* hand) {

}
```

Go Solution:

```
// Problem: Zuma Game
// Difficulty: Hard
// Tags: string, dp, search, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func findMinStep(board string, hand string) int {
}
```

Kotlin Solution:

```
class Solution {
fun findMinStep(board: String, hand: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func findMinStep(_ board: String, _ hand: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Zuma Game
// Difficulty: Hard
// Tags: string, dp, search, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn find_min_step(board: String, hand: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} board
# @param {String} hand
# @return {Integer}
def find_min_step(board, hand)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $board
* @param String $hand
* @return Integer
*/
function findMinStep($board, $hand) {

}
}
```

Dart Solution:

```
class Solution {
int findMinStep(String board, String hand) {

}
}
```

Scala Solution:

```
object Solution {
def findMinStep(board: String, hand: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_min_step(board :: String.t, hand :: String.t) :: integer
def find_min_step(board, hand) do

end
end
```

Erlang Solution:

```
-spec find_min_step(Board :: unicode:unicode_binary(), Hand ::
unicode:unicode_binary()) -> integer().
find_min_step(Board, Hand) ->
.
```

Racket Solution:

```
(define/contract (find-min-step board hand)
(-> string? string? exact-integer?)
)
```