Problem 2379: Minimum Recolors to Get K Consecutive Black Blocks

Problem Information

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

Problem Description

You are given a

0-indexed

string

blocks

of length

n

, where

blocks[i]

is either

'W'

or

'B'

, representing the color of the

i

th

block. The characters

'W'

and

'B'

denote the colors white and black, respectively.

You are also given an integer

k

, which is the desired number of

consecutive

black blocks.

In one operation, you can

recolor

a white block such that it becomes a black block.

Return

the

minimum

number of operations needed such that there is at least

one

occurrence of

k

consecutive black blocks.

Example 1:

Input:

blocks = "WBBWWBBWBW", k = 7

Output:

3

Explanation:

One way to achieve 7 consecutive black blocks is to recolor the 0th, 3rd, and 4th blocks so that blocks = "BBBBBBBWBW". It can be shown that there is no way to achieve 7 consecutive black blocks in less than 3 operations. Therefore, we return 3.

Example 2:

Input:

blocks = "WBWBBBW", k = 2

Output:

0

Explanation:

No changes need to be made, since 2 consecutive black blocks already exist. Therefore, we return 0.

Constraints:

n == blocks.length

1 <= n <= 100

blocks[i]

is either

'W'

or

'B'

.

1 <= k <= n

Code Snippets

C++:

```
class Solution {
public:
int minimumRecolors(string blocks, int k) {

}
};
```

Java:

```
class Solution {
public int minimumRecolors(String blocks, int k) {

}
}
```

Python3:

```
class Solution:
def minimumRecolors(self, blocks: str, k: int) -> int:
```

Python:

```
class Solution(object):
def minimumRecolors(self, blocks, k):
"""
:type blocks: str
:type k: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} blocks
* @param {number} k
* @return {number}
*/
var minimumRecolors = function(blocks, k) {

};
```

TypeScript:

```
function minimumRecolors(blocks: string, k: number): number {

};
```

C#:

```
public class Solution {
public int MinimumRecolors(string blocks, int k) {

}
}
```

C:

```
int minimumRecolors(char* blocks, int k) {

}
```

Go:

```
func minimumRecolors(blocks string, k int) int {

}
```

Kotlin:

```
class Solution {
fun minimumRecolors(blocks: String, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumRecolors(_ blocks: String, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_recolors(blocks: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} blocks
# @param {Integer} k
# @return {Integer}
def minimum_recolors(blocks, k)

end
```

PHP:

```
class Solution {

/**
* @param String $blocks
* @param Integer $k
* @return Integer
*/
function minimumRecolors($blocks, $k) {

}
}
```

Dart:

```
class Solution {
int minimumRecolors(String blocks, int k) {

}
}
```

Scala:

```
object Solution {
def minimumRecolors(blocks: String, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_recolors(blocks :: String.t, k :: integer) :: integer
def minimum_recolors(blocks, k) do

end
end
```

Erlang:

```
-spec minimum_recolors(Blocks :: unicode:unicode_binary(), K :: integer()) ->
integer().
```

```
minimum_recolors(Blocks, K) ->
.
```

Racket:

```
(define/contract (minimum-recolors blocks k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Recolors to Get K Consecutive Black Blocks
* Difficulty: Easy
* 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:
int minimumRecolors(string blocks, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Recolors to Get K Consecutive Black Blocks
* Difficulty: Easy
* 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 int minimumRecolors(String blocks, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Recolors to Get K Consecutive Black Blocks
Difficulty: Easy
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 minimumRecolors(self, blocks: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumRecolors(self, blocks, k):
"""
:type blocks: str
:type k: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Recolors to Get K Consecutive Black Blocks
* Difficulty: Easy
* 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} blocks
* @param {number} k
* @return {number}
*/
var minimumRecolors = function(blocks, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Recolors to Get K Consecutive Black Blocks
* Difficulty: Easy
* 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 minimumRecolors(blocks: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Recolors to Get K Consecutive Black Blocks
* Difficulty: Easy
* 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 int MinimumRecolors(string blocks, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Recolors to Get K Consecutive Black Blocks
* Difficulty: Easy
* 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
*/

int minimumRecolors(char* blocks, int k) {

}
```

Go Solution:

```
// Problem: Minimum Recolors to Get K Consecutive Black Blocks
// Difficulty: Easy
// 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 minimumRecolors(blocks string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumRecolors(blocks: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumRecolors(_ blocks: String, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Recolors to Get K Consecutive Black Blocks
// Difficulty: Easy
// 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 minimum_recolors(blocks: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} blocks
# @param {Integer} k
# @return {Integer}
def minimum_recolors(blocks, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $blocks
* @param Integer $k
* @return Integer
*/
function minimumRecolors($blocks, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minimumRecolors(String blocks, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minimumRecolors(blocks: String, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_recolors(blocks :: String.t, k :: integer) :: integer
def minimum_recolors(blocks, k) do

end
end
```

Erlang Solution:

```
-spec minimum_recolors(Blocks :: unicode:unicode_binary(), K :: integer()) ->
integer().
minimum_recolors(Blocks, K) ->
```

```
.
```

Racket Solution:

```
(define/contract (minimum-recolors blocks k)
(-> string? exact-integer? exact-integer?)
)
```