Problem 2209: Minimum White Tiles After Covering With Carpets

Problem Information

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

Problem Description

You are given a

0-indexed binary

string

floor

, which represents the colors of tiles on a floor:

floor[i] = '0'

denotes that the

i

th

tile of the floor is colored

black

.

On the other hand,

floor[i] = '1'

denotes that the

i

th

tile of the floor is colored

white

.

You are also given

numCarpets

and

carpetLen

. You have

numCarpets

black

carpets, each of length

carpetLen

tiles. Cover the tiles with the given carpets such that the number of

white

tiles still visible is

minimum

. Carpets may overlap one another.

Return

the

minimum

number of white tiles still visible.

Example 1:

Input:

floor = "10110101", numCarpets = 2, carpetLen = 2

Output:

2

Explanation:

The figure above shows one way of covering the tiles with the carpets such that only 2 white tiles are visible. No other way of covering the tiles with the carpets can leave less than 2 white tiles visible.

Example 2:

Input:

floor = "11111", numCarpets = 2, carpetLen = 3

Output:

0

Explanation:

The figure above shows one way of covering the tiles with the carpets such that no white tiles are visible. Note that the carpets are able to overlap one another.

Constraints:

1 <= carpetLen <= floor.length <= 1000

floor[i]

is either

'0'

or

'1'

.

1 <= numCarpets <= 1000

Code Snippets

C++:

```
class Solution {
public:
int minimumWhiteTiles(string floor, int numCarpets, int carpetLen) {

}
};
```

Java:

```
class Solution {
public int minimumWhiteTiles(String floor, int numCarpets, int carpetLen) {

}
}
```

Python3:

```
class Solution:
def minimumWhiteTiles(self, floor: str, numCarpets: int, carpetLen: int) ->
int:
```

Python:

```
class Solution(object):
def minimumWhiteTiles(self, floor, numCarpets, carpetLen):
"""
:type floor: str
:type numCarpets: int
:type carpetLen: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} floor
* @param {number} numCarpets
* @param {number} carpetLen
```

```
* @return {number}
*/
var minimumWhiteTiles = function(floor, numCarpets, carpetLen) {

};
```

TypeScript:

```
function minimumWhiteTiles(floor: string, numCarpets: number, carpetLen:
number): number {

};
```

C#:

```
public class Solution {
public int MinimumWhiteTiles(string floor, int numCarpets, int carpetLen) {

}
}
```

C:

```
int minimumWhiteTiles(char* floor, int numCarpets, int carpetLen) {

}
```

Go:

```
func minimumWhiteTiles(floor string, numCarpets int, carpetLen int) int {

}
```

Kotlin:

```
class Solution {
fun minimumWhiteTiles(floor: String, numCarpets: Int, carpetLen: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumWhiteTiles(_ floor: String, _ numCarpets: Int, _ carpetLen: Int)
-> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_white_tiles(floor: String, num_carpets: i32, carpet_len: i32)
-> i32 {

}
}
```

Ruby:

```
# @param {String} floor
# @param {Integer} num_carpets
# @param {Integer} carpet_len
# @return {Integer}
def minimum_white_tiles(floor, num_carpets, carpet_len)

end
```

PHP:

```
class Solution {

/**
* @param String $floor
* @param Integer $numCarpets
* @param Integer $carpetLen
* @return Integer
*/
function minimumWhiteTiles($floor, $numCarpets, $carpetLen) {

}
}
```

Dart:

```
class Solution {
int minimumWhiteTiles(String floor, int numCarpets, int carpetLen) {

}
}
```

Scala:

```
object Solution {
def minimumWhiteTiles(floor: String, numCarpets: Int, carpetLen: Int): Int =
{

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_white_tiles(floor :: String.t, num_carpets :: integer,
carpet_len :: integer) :: integer
def minimum_white_tiles(floor, num_carpets, carpet_len) do

end
end
```

Erlang:

```
-spec minimum_white_tiles(Floor :: unicode:unicode_binary(), NumCarpets ::
integer(), CarpetLen :: integer()) -> integer().
minimum_white_tiles(Floor, NumCarpets, CarpetLen) ->
.
```

Racket:

```
(define/contract (minimum-white-tiles floor numCarpets carpetLen)
(-> string? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum White Tiles After Covering With Carpets
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int minimumWhiteTiles(string floor, int numCarpets, int carpetLen) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum White Tiles After Covering With Carpets
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int minimumWhiteTiles(String floor, int numCarpets, int carpetLen) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum White Tiles After Covering With Carpets
Difficulty: Hard
Tags: array, string, dp
```

```
Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def minimumWhiteTiles(self, floor: str, numCarpets: int, carpetLen: int) ->
int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minimumWhiteTiles(self, floor, numCarpets, carpetLen):
"""
:type floor: str
:type numCarpets: int
:type carpetLen: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum White Tiles After Covering With Carpets
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {string} floor
* @param {number} numCarpets
* @param {number} carpetLen
* @return {number}
*/
var minimumWhiteTiles = function(floor, numCarpets, carpetLen) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Minimum White Tiles After Covering With Carpets
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function minimumWhiteTiles(floor: string, numCarpets: number, carpetLen:
number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum White Tiles After Covering With Carpets
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* 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 MinimumWhiteTiles(string floor, int numCarpets, int carpetLen) {

}
}
```

C Solution:

```
/*
* Problem: Minimum White Tiles After Covering With Carpets
* Difficulty: Hard
* Tags: array, string, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int minimumWhiteTiles(char* floor, int numCarpets, int carpetLen) {

}
```

Go Solution:

```
// Problem: Minimum White Tiles After Covering With Carpets
// Difficulty: Hard
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func minimumWhiteTiles(floor string, numCarpets int, carpetLen int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumWhiteTiles(floor: String, numCarpets: Int, carpetLen: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumWhiteTiles(_ floor: String, _ numCarpets: Int, _ carpetLen: Int)
-> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Minimum White Tiles After Covering With Carpets
// Difficulty: Hard
// Tags: array, string, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn minimum_white_tiles(floor: String, num_carpets: i32, carpet_len: i32)
-> i32 {

}
}
```

Ruby Solution:

```
# @param {String} floor
# @param {Integer} num_carpets
# @param {Integer} carpet_len
# @return {Integer}
def minimum_white_tiles(floor, num_carpets, carpet_len)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $floor
* @param Integer $numCarpets
* @param Integer $carpetLen
* @return Integer
*/
function minimumWhiteTiles($floor, $numCarpets, $carpetLen) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int minimumWhiteTiles(String floor, int numCarpets, int carpetLen) {

}
}
```

Scala Solution:

```
object Solution {
def minimumWhiteTiles(floor: String, numCarpets: Int, carpetLen: Int): Int =
{

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_white_tiles(floor :: String.t, num_carpets :: integer,
carpet_len :: integer) :: integer
def minimum_white_tiles(floor, num_carpets, carpet_len) do

end
end
```

Erlang Solution:

```
-spec minimum_white_tiles(Floor :: unicode:unicode_binary(), NumCarpets ::
integer(), CarpetLen :: integer()) -> integer().
minimum_white_tiles(Floor, NumCarpets, CarpetLen) ->
.
```

Racket Solution:

```
(define/contract (minimum-white-tiles floor numCarpets carpetLen)
(-> string? exact-integer? exact-integer? exact-integer?)
)
```