Problem 2911: Minimum Changes to Make K Semi-palindromes

Problem Information

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

Problem Description

Given a string

s

and an integer

k

, partition

s

into

k

substrings

such that the letter changes needed to make each substring a

semi-palindrome

are minimized.

Return the

minimum

number of letter changes

required

.

A

semi-palindrome

is a special type of string that can be divided into

palindromes

based on a repeating pattern. To check if a string is a semi-palindrome:

Choose a positive divisor

d

of the string's length.

d

can range from

1

up to, but not including, the string's length. For a string of length

1

, it does not have a valid divisor as per this definition, since the only divisor is its length, which is not allowed.

For a given divisor

d

, divide the string into groups where each group contains characters from the string that follow a repeating pattern of length

d

. Specifically, the first group consists of characters at positions

1

,

1 + d

,

1 + 2d

, and so on; the second group includes characters at positions

2

,

2 + d

,

2 + 2d

, etc.

The string is considered a semi-palindrome if each of these groups forms a palindrome.

Consider the string

"abcabc"

:

The length of

"abcabc"

is

6

. Valid divisors are

1

,

2

, and

3

.

For

d = 1

: The entire string

"abcabc"

forms one group. Not a palindrome.

For

d = 2

:

Group 1 (positions

1, 3, 5

):

"acb"

Group 2 (positions

2, 4, 6

):

"bac"

Neither group forms a palindrome.

For

d = 3

:

Group 1 (positions

1, 4

):

"aa"

Group 2 (positions

2, 5

):

"bb"

Group 3 (positions

3, 6

):

"cc"

All groups form palindromes. Therefore,

"abcabc"

is a semi-palindrome.

Example 1:

Input:

s = "abcac", k = 2

Output:

1

Explanation:

Divide

s

into

"ab"

and

"cac"

.

"cac"

is already semi-palindrome. Change

"ab"

to

"aa"

, it becomes semi-palindrome with

d = 1

.

Example 2:

Input:

s = "abcdef", k = 2

Output:

2

Explanation:

Divide

s

into substrings

"abc"

and

"def"

. Each needs one change to become semi-palindrome.

Example 3:

Input:

s = "aabbaa", k = 3

Output:

0

Explanation:

Divide

s

into substrings

"aa"

,

"bb"

and

"aa"

. All are already semi-palindromes.

Constraints:

2 <= s.length <= 200

1 <= k <= s.length / 2

s

contains only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int minimumChanges(string s, int k) {

}
};
```

Java:

```
class Solution {
public int minimumChanges(String s, int k) {

}
}
```

Python3:

```
class Solution:
def minimumChanges(self, s: str, k: int) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function minimumChanges(s: string, k: number): number {

};
```

C#:

```
public class Solution {
public int MinimumChanges(string s, int k) {

}
}
```

C:

```
int minimumChanges(char* s, int k) {

}
```

Go:

```
func minimumChanges(s string, k int) int {

}
```

Kotlin:

```
class Solution {
fun minimumChanges(s: String, k: Int): Int {

}
```

```
}
```

Swift:

```
class Solution {
func minimumChanges(_ s: String, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_changes(s: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def minimum_changes(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function minimumChanges($s, $k) {

}
}
```

Dart:

```
class Solution {
int minimumChanges(String s, int k) {

}
}
```

Scala:

```
object Solution {
def minimumChanges(s: String, k: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec minimum_changes(S :: unicode:unicode_binary(), K :: integer()) ->
integer().
minimum_changes(S, K) ->
.
```

Racket:

```
(define/contract (minimum-changes s k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Changes to Make K Semi-palindromes
* Difficulty: Hard
* Tags: array, string, tree, 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 minimumChanges(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Changes to Make K Semi-palindromes
* Difficulty: Hard
* Tags: array, string, tree, 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 minimumChanges(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Changes to Make K Semi-palindromes
Difficulty: Hard
Tags: array, string, tree, 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 minimumChanges(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Changes to Make K Semi-palindromes
* Difficulty: Hard
* Tags: array, string, tree, 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} s
* @param {number} k
* @return {number}
*/
var minimumChanges = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Changes to Make K Semi-palindromes
* Difficulty: Hard
* Tags: array, string, tree, 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 minimumChanges(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Changes to Make K Semi-palindromes
* Difficulty: Hard
* Tags: array, string, tree, 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 MinimumChanges(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Changes to Make K Semi-palindromes
* Difficulty: Hard
* Tags: array, string, tree, 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 minimumChanges(char* s, int k) {

}
```

Go Solution:

```
// Problem: Minimum Changes to Make K Semi-palindromes
// Difficulty: Hard
// Tags: array, string, tree, 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 minimumChanges(s string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumChanges(s: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumChanges(_ s: String, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Changes to Make K Semi-palindromes
// Difficulty: Hard
```

```
// Tags: array, string, tree, 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_changes(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def minimum_changes(s, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function minimumChanges($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minimumChanges(String s, int k) {

}
```

```
}
```

Scala Solution:

```
object Solution {
def minimumChanges(s: String, k: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec minimum_changes(S :: unicode:unicode_binary(), K :: integer()) ->
integer().
minimum_changes(S, K) ->
.
```

Racket Solution:

```
(define/contract (minimum-changes s k)
(-> string? exact-integer? exact-integer?)
)
```