Problem 3137: Minimum Number of Operations to Make Word K-Periodic

Problem Information

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

Problem Description

You are given a string

word

of size

n

, and an integer

k

such that

k

divides

n

.

In one operation, you can pick any two indices

i

and

j

, that are divisible by

k

, then replace the

substring

of length

k

starting at

i

with the substring of length

k

starting at

j

. That is, replace the substring

word[i..i + k - 1]

with the substring

word[j..j + k - 1]

.

Return

the

minimum

number of operations required to make

word

k-periodic

.

We say that

word

is

k-periodic

if there is some string

s

of length

k

such that

word

can be obtained by concatenating

s

an arbitrary number of times. For example, if word == “ababab”

, then

word

is 2-periodic for

s = "ab"

.

Example 1:

Input:

word = "leetcodeleet", k = 4

Output:

1

Explanation:

We can obtain a 4-periodic string by picking i = 4 and j = 0. After this operation, word becomes equal to "leetleetleet".

Example 2:

Input:

word = "

leetcoleet

", k = 2

Output:

3

Explanation:

We can obtain a 2-periodic string by applying the operations in the table below.

i

j

word

0

2

etetcoleet

4

0

etetetleet

6

0

etetetetet

Constraints:

1 <= n == word.length <= 10

5

1 <= k <= word.length

k

divides

word.length

.

word

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int minimumOperationsToMakeKPeriodic(string word, int k) {

}
};
```

Java:

```
class Solution {
public int minimumOperationsToMakeKPeriodic(String word, int k) {

}
}
```

Python3:

```
class Solution:
def minimumOperationsToMakeKPeriodic(self, word: str, k: int) -> int:
```

Python:

```
class Solution(object):
def minimumOperationsToMakeKPeriodic(self, word, k):
```

```
"""
:type word: str
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minimumOperationsToMakeKPeriodic(word: string, k: number): number {

};
```

C#:

```
public class Solution {
public int MinimumOperationsToMakeKPeriodic(string word, int k) {

}
}
```

C:

```
int minimumOperationsToMakeKPeriodic(char* word, int k) {

}
```

Go:

```
func minimumOperationsToMakeKPeriodic(word string, k int) int {

}
```

Kotlin:

```
class Solution {
fun minimumOperationsToMakeKPeriodic(word: String, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumOperationsToMakeKPeriodic(_ word: String, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_operations_to_make_k_periodic(word: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} word
# @param {Integer} k
# @return {Integer}
def minimum_operations_to_make_k_periodic(word, k)

end
```

PHP:

```
class Solution {

/**
* @param String $word
* @param Integer $k
* @return Integer
*/
function minimumOperationsToMakeKPeriodic($word, $k) {



}
}
```

Dart:

```
class Solution {
int minimumOperationsToMakeKPeriodic(String word, int k) {

}
}
```

Scala:

```
object Solution {
def minimumOperationsToMakeKPeriodic(word: String, k: Int): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec minimum_operations_to_make_k_periodic(Word :: unicode:unicode_binary(),
K :: integer()) -> integer().
minimum_operations_to_make_k_periodic(Word, K) ->
.
```

Racket:

```
(define/contract (minimum-operations-to-make-k-periodic word k)
(-> string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Number of Operations to Make Word K-Periodic
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minimumOperationsToMakeKPeriodic(string word, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Number of Operations to Make Word K-Periodic
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minimumOperationsToMakeKPeriodic(String word, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Number of Operations to Make Word K-Periodic
Difficulty: Medium
Tags: string, tree, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def minimumOperationsToMakeKPeriodic(self, word: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make Word K-Periodic
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string} word
* @param {number} k
* @return {number}
*/
```

```
var minimumOperationsToMakeKPeriodic = function(word, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Number of Operations to Make Word K-Periodic
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minimumOperationsToMakeKPeriodic(word: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Number of Operations to Make Word K-Periodic
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinimumOperationsToMakeKPeriodic(string word, int k) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Number of Operations to Make Word K-Periodic
* Difficulty: Medium
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minimumOperationsToMakeKPeriodic(char* word, int k) {

}
```

Go Solution:

```
// Problem: Minimum Number of Operations to Make Word K-Periodic
// Difficulty: Medium
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minimumOperationsToMakeKPeriodic(word string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumOperationsToMakeKPeriodic(word: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumOperationsToMakeKPeriodic(_ word: String, _ k: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Number of Operations to Make Word K-Periodic
// Difficulty: Medium
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn minimum_operations_to_make_k_periodic(word: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} word
# @param {Integer} k
# @return {Integer}
def minimum_operations_to_make_k_periodic(word, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word
* @param Integer $k
* @return Integer
*/
function minimumOperationsToMakeKPeriodic($word, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minimumOperationsToMakeKPeriodic(String word, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minimumOperationsToMakeKPeriodic(word: String, k: Int): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec minimum_operations_to_make_k_periodic(Word :: unicode:unicode_binary(),
K :: integer()) -> integer().
minimum_operations_to_make_k_periodic(Word, K) ->
.
```

Racket Solution:

```
(define/contract (minimum-operations-to-make-k-periodic word k)
(-> string? exact-integer? exact-integer?)
)
```