Problem 1668: Maximum Repeating Substring

Problem Information

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

Problem Description

For a string

sequence

, a string

word

is

k

-repeating

if

word

concatenated

k

times is a substring of

sequence

. The

word

's

maximum

k

-repeating value

is the highest value

k

where

word

is

k

-repeating in

sequence

. If

word

is not a substring of

sequence

,

word

's maximum

k

-repeating value is

0

.

Given strings

sequence

and

word

, return

the

maximum

k

-repeating value

of

word

in

sequence

.

Example 1:

Input:

sequence = "ababc", word = "ab"

Output:

2

Explanation:

"abab" is a substring in "

abab

c".

Example 2:

Input:

sequence = "ababc", word = "ba"

Output:

1

Explanation:

"ba" is a substring in "a

ba

bc". "baba" is not a substring in "ababc".

Example 3:

Input:

sequence = "ababc", word = "ac"

Output:

0

Explanation:

"ac" is not a substring in "ababc".

Constraints:

1 <= sequence.length <= 100

1 <= word.length <= 100

sequence

and

word

contains only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int maxRepeating(string sequence, string word) {

}
};
```

Java:

```
class Solution {
public int maxRepeating(String sequence, String word) {
```

```
}
}
```

Python3:

```
class Solution:
def maxRepeating(self, sequence: str, word: str) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function maxRepeating(sequence: string, word: string): number {

};
```

C#:

```
public class Solution {
public int MaxRepeating(string sequence, string word) {

}
```

```
}
```

C:

```
int maxRepeating(char* sequence, char* word) {

}
```

Go:

```
func maxRepeating(sequence string, word string) int {

}
```

Kotlin:

```
class Solution {
fun maxRepeating(sequence: String, word: String): Int {

}
}
```

Swift:

```
class Solution {
func maxRepeating(_ sequence: String, _ word: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_repeating(sequence: String, word: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} sequence
# @param {String} word
```

```
# @return {Integer}
def max_repeating(sequence, word)

end
```

PHP:

```
class Solution {

/**
* @param String $sequence
* @param String $word
* @return Integer
*/
function maxRepeating($sequence, $word) {

}
}
```

Dart:

```
class Solution {
int maxRepeating(String sequence, String word) {

}
}
```

Scala:

```
object Solution {
def maxRepeating(sequence: String, word: String): Int = {

}
}
```

Elixir:

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

end
```

```
end
```

Erlang:

```
-spec max_repeating(Sequence :: unicode:unicode_binary(), Word ::
unicode:unicode_binary()) -> integer().
max_repeating(Sequence, Word) ->
.
```

Racket:

```
(define/contract (max-repeating sequence word)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Repeating Substring
* Difficulty: Easy
* Tags: string, tree, dp
*
* 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 maxRepeating(string sequence, string word) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Repeating Substring
```

```
* Difficulty: Easy
* Tags: string, tree, dp
*
* 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 maxRepeating(String sequence, String word) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Repeating Substring
Difficulty: Easy
Tags: string, tree, dp

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 maxRepeating(self, sequence: str, word: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Repeating Substring
* Difficulty: Easy
* Tags: string, tree, dp
*
* 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} sequence
* @param {string} word
* @return {number}
*/
var maxRepeating = function(sequence, word) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Repeating Substring
* Difficulty: Easy
* Tags: string, tree, dp
*
* 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 maxRepeating(sequence: string, word: string): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Repeating Substring
* Difficulty: Easy
* Tags: string, tree, dp
```

```
*
* 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 MaxRepeating(string sequence, string word) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Repeating Substring
* Difficulty: Easy
* Tags: string, tree, dp
*
* 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 maxRepeating(char* sequence, char* word) {

}
```

Go Solution:

```
// Problem: Maximum Repeating Substring
// Difficulty: Easy
// Tags: string, tree, dp
//
// 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 maxRepeating(sequence string, word string) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxRepeating(sequence: String, word: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxRepeating(_ sequence: String, _ word: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximum Repeating Substring
// Difficulty: Easy
// Tags: string, tree, dp
//
// 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 max_repeating(sequence: String, word: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} sequence
# @param {String} word
# @return {Integer}
def max_repeating(sequence, word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $sequence
* @param String $word
* @return Integer
*/
function maxRepeating($sequence, $word) {

}
}
```

Dart Solution:

```
class Solution {
int maxRepeating(String sequence, String word) {

}
}
```

Scala Solution:

```
object Solution {
def maxRepeating(sequence: String, word: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec max_repeating(Sequence :: unicode:unicode_binary(), Word ::
unicode:unicode_binary()) -> integer().
max_repeating(Sequence, Word) ->
.
```

Racket Solution:

```
(define/contract (max-repeating sequence word)
(-> string? string? exact-integer?)
)
```