Problem 686: Repeated String Match

Problem Information

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

Problem Description

Given two strings

a

and

b

, return

the minimum number of times you should repeat string

a

so that string

b

is a substring of it

. If it is impossible for

b

to be a substring of

a

after repeating it, return

-1

.

Notice:

string

"abc"

repeated 0 times is

""

, repeated 1 time is

"abc"

and repeated 2 times is

"abcabc"

.

Example 1:

Input:

a = "abcd", b = "cdabcdab"

Output:

3

Explanation:

We return 3 because by repeating a three times "ab

cdabcdab

cd", b is a substring of it.

Example 2:

Input:

a = "a", b = "aa"

Output:

2

Constraints:

1 <= a.length, b.length <= 10

4

a

and

b

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int repeatedStringMatch(string a, string b) {
```

```
}
};
```

Java:

```
class Solution {
public int repeatedStringMatch(String a, String b) {

}
}
```

Python3:

```
class Solution:
def repeatedStringMatch(self, a: str, b: str) -> int:
```

Python:

```
class Solution(object):
def repeatedStringMatch(self, a, b):
"""
:type a: str
:type b: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} a
* @param {string} b
* @return {number}
*/
var repeatedStringMatch = function(a, b) {

};
```

TypeScript:

```
function repeatedStringMatch(a: string, b: string): number {

};
```

C#:

```
public class Solution {
public int RepeatedStringMatch(string a, string b) {

}
}
```

C:

```
int repeatedStringMatch(char* a, char* b) {

}
```

Go:

```
func repeatedStringMatch(a string, b string) int {

}
```

Kotlin:

```
class Solution {
fun repeatedStringMatch(a: String, b: String): Int {

}
}
```

Swift:

```
class Solution {
func repeatedStringMatch(_ a: String, _ b: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn repeated_string_match(a: String, b: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} a
# @param {String} b
# @return {Integer}
def repeated_string_match(a, b)

end
```

PHP:

```
class Solution {

/**
* @param String $a
* @param String $b
* @return Integer
*/
function repeatedStringMatch($a, $b) {

}
}
```

Dart:

```
class Solution {
int repeatedStringMatch(String a, String b) {

}
}
```

Scala:

```
object Solution {
def repeatedStringMatch(a: String, b: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec repeated_string_match(a :: String.t, b :: String.t) :: integer
```

```
def repeated_string_match(a, b) do

end
end
```

Erlang:

```
-spec repeated_string_match(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary()) -> integer().
repeated_string_match(A, B) ->
.
```

Racket:

```
(define/contract (repeated-string-match a b)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Repeated String Match
* Difficulty: Medium
* Tags: string, tree
*
* 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 repeatedStringMatch(string a, string b) {

}
};
```

Java Solution:

```
/**
* Problem: Repeated String Match
* Difficulty: Medium
* Tags: string, tree
*
* 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 repeatedStringMatch(String a, String b) {

}
}
```

Python3 Solution:

```
"""
Problem: Repeated String Match
Difficulty: Medium
Tags: string, tree

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 repeatedStringMatch(self, a: str, b: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def repeatedStringMatch(self, a, b):
"""
:type a: str
:type b: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Repeated String Match
* Difficulty: Medium
* Tags: string, tree
*
* 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} a
* @param {string} b
* @return {number}
*/
var repeatedStringMatch = function(a, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Repeated String Match
* Difficulty: Medium
* Tags: string, tree
*
* 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 repeatedStringMatch(a: string, b: string): number {

};
```

C# Solution:

```
/*
* Problem: Repeated String Match
* Difficulty: Medium
```

```
* Tags: string, tree
*
* 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 RepeatedStringMatch(string a, string b) {

}
}
```

C Solution:

```
/*
* Problem: Repeated String Match
* Difficulty: Medium
* Tags: string, tree
*
* 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 repeatedStringMatch(char* a, char* b) {

}
```

Go Solution:

```
// Problem: Repeated String Match
// Difficulty: Medium
// Tags: string, tree
//
// 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 repeatedStringMatch(a string, b string) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun repeatedStringMatch(a: String, b: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func repeatedStringMatch(_ a: String, _ b: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Repeated String Match
// Difficulty: Medium
// Tags: string, tree
//
// 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 repeated_string_match(a: String, b: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} a
# @param {String} b
# @return {Integer}
def repeated_string_match(a, b)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $a
* @param String $b
* @return Integer
*/
function repeatedStringMatch($a, $b) {

}
}
```

Dart Solution:

```
class Solution {
int repeatedStringMatch(String a, String b) {

}
}
```

Scala Solution:

```
object Solution {
def repeatedStringMatch(a: String, b: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec repeated_string_match(a :: String.t, b :: String.t) :: integer
def repeated_string_match(a, b) do

end
end
```

Erlang Solution:

```
-spec repeated_string_match(A :: unicode:unicode_binary(), B ::
unicode:unicode_binary()) -> integer().
repeated_string_match(A, B) ->
.
```

Racket Solution:

```
(define/contract (repeated-string-match a b)
(-> string? string? exact-integer?)
)
```