Problem 466: Count The Repetitions

Problem Information

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

Problem Description

We define

str = [s, n]

as the string

str

which consists of the string

s

concatenated

n

times.

For example,

str == ["abc", 3] =="abcabcabc"

.

We define that string

s1

can be obtained from string

s2

if we can remove some characters from

s2

such that it becomes

s1

.

For example,

s1 = "abc"

can be obtained from

s2 = "ab

dbe

c"

based on our definition by removing the bolded underlined characters.

You are given two strings

s1

and

s2

and two integers

n1

and

n2

. You have the two strings

str1 = [s1, n1]

and

str2 = [s2, n2]

.

Return

the maximum integer

m

such that

str = [str2, m]

can be obtained from

str1

.

Example 1:

Input:

s1 = "acb", n1 = 4, s2 = "ab", n2 = 2

Output:

2

Example 2:

Input:

s1 = "acb", n1 = 1, s2 = "acb", n2 = 1

Output:

1

Constraints:

1 <= s1.length, s2.length <= 100

s1

and

s2

consist of lowercase English letters.

1 <= n1, n2 <= 10

6

Code Snippets

C++:

```
class Solution {
public:
int getMaxRepetitions(string s1, int n1, string s2, int n2) {
}
};
```

Java:

```
class Solution {
public int getMaxRepetitions(String s1, int n1, String s2, int n2) {

}
}
```

Python3:

```
class Solution:
def getMaxRepetitions(self, s1: str, n1: int, s2: str, n2: int) -> int:
```

Python:

```
class Solution(object):
def getMaxRepetitions(self, s1, n1, s2, n2):
"""
:type s1: str
:type n1: int
:type s2: str
:type n2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s1
* @param {number} n1
* @param {string} s2
* @param {number} n2
* @return {number}
*/
var getMaxRepetitions = function(s1, n1, s2, n2) {

};
```

TypeScript:

```
function getMaxRepetitions(s1: string, n1: number, s2: string, n2: number):
number {

};
```

C#:

```
public class Solution {
public int GetMaxRepetitions(string s1, int n1, string s2, int n2) {

}
}
```

C:

```
int getMaxRepetitions(char* s1, int n1, char* s2, int n2) {

}
```

Go:

```
func getMaxRepetitions(s1 string, n1 int, s2 string, n2 int) int {

}
```

Kotlin:

```
class Solution {
fun getMaxRepetitions(s1: String, n1: Int, s2: String, n2: Int): Int {

}
}
```

Swift:

```
class Solution {
func getMaxRepetitions(_ s1: String, _ n1: Int, _ s2: String, _ n2: Int) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn get_max_repetitions(s1: String, n1: i32, s2: String, n2: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s1
# @param {Integer} n1
# @param {String} s2
# @param {Integer} n2
# @return {Integer}
def get_max_repetitions(s1, n1, s2, n2)

end
```

PHP:

```
class Solution {

/**
* @param String $s1
* @param Integer $n1
* @param String $s2
* @param Integer $n2
* @return Integer
*/
function getMaxRepetitions($s1, $n1, $s2, $n2) {

}
}
```

Dart:

```
class Solution {
int getMaxRepetitions(String s1, int n1, String s2, int n2) {

}
}
```

Scala:

```
object Solution {
def getMaxRepetitions(s1: String, n1: Int, s2: String, n2: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec get_max_repetitions(s1 :: String.t, n1 :: integer, s2 :: String.t, n2
:: integer) :: integer
def get_max_repetitions(s1, n1, s2, n2) do

end
end
```

Erlang:

```
-spec get_max_repetitions(S1 :: unicode:unicode_binary(), N1 :: integer(), S2
:: unicode:unicode_binary(), N2 :: integer()) -> integer().
get_max_repetitions(S1, N1, S2, N2) ->
.
```

Racket:

```
(define/contract (get-max-repetitions s1 n1 s2 n2)
(-> string? exact-integer? string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count The Repetitions
* Difficulty: Hard
* Tags: string, 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 getMaxRepetitions(string s1, int n1, string s2, int n2) {

}
};
```

Java Solution:

```
/**
* Problem: Count The Repetitions
* Difficulty: Hard
* Tags: string, 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 getMaxRepetitions(String s1, int n1, String s2, int n2) {

}
}
```

Python3 Solution:

```
"""
Problem: Count The Repetitions
Difficulty: Hard
Tags: string, 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 getMaxRepetitions(self, s1: str, n1: int, s2: str, n2: int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def getMaxRepetitions(self, s1, n1, s2, n2):
"""
:type s1: str
:type n1: int
:type s2: str
:type n2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count The Repetitions
* Difficulty: Hard
* Tags: string, 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} s1
* @param {number} n1
* @param {string} s2
* @param {number} n2
* @return {number}
*/
var getMaxRepetitions = function(s1, n1, s2, n2) {

};
```

TypeScript Solution:

```
/**
* Problem: Count The Repetitions
* Difficulty: Hard
* Tags: string, 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 getMaxRepetitions(s1: string, n1: number, s2: string, n2: number):
number {

};
```

C# Solution:

```
/*
* Problem: Count The Repetitions
* Difficulty: Hard
* Tags: string, 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 GetMaxRepetitions(string s1, int n1, string s2, int n2) {

}
}
```

C Solution:

```
/*
* Problem: Count The Repetitions
* Difficulty: Hard
* Tags: string, 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 getMaxRepetitions(char* s1, int n1, char* s2, int n2) {

}
```

Go Solution:

```
// Problem: Count The Repetitions
// Difficulty: Hard
// Tags: string, 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 getMaxRepetitions(s1 string, n1 int, s2 string, n2 int) int {

}
```

Kotlin Solution:

```
class Solution {
fun getMaxRepetitions(s1: String, n1: Int, s2: String, n2: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func getMaxRepetitions(_ s1: String, _ n1: Int, _ s2: String, _ n2: Int) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Count The Repetitions
// Difficulty: Hard
// Tags: string, 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 get_max_repetitions(s1: String, n1: i32, s2: String, n2: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s1
# @param {Integer} n1
# @param {String} s2
# @param {Integer} n2
# @return {Integer}
def get_max_repetitions(s1, n1, s2, n2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s1
* @param Integer $n1
* @param String $s2
* @param Integer $n2
* @return Integer
*/
function getMaxRepetitions($s1, $n1, $s2, $n2) {

}
}
```

Dart Solution:

```
class Solution {
int getMaxRepetitions(String s1, int n1, String s2, int n2) {

}
}
```

Scala Solution:

```
object Solution {
def getMaxRepetitions(s1: String, n1: Int, s2: String, n2: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec get_max_repetitions(s1 :: String.t, n1 :: integer, s2 :: String.t, n2
:: integer) :: integer
def get_max_repetitions(s1, n1, s2, n2) do

end
end
```

Erlang Solution:

```
-spec get_max_repetitions(S1 :: unicode:unicode_binary(), N1 :: integer(), S2
:: unicode:unicode_binary(), N2 :: integer()) -> integer().
get_max_repetitions(S1, N1, S2, N2) ->
.
```

Racket Solution:

```
(define/contract (get-max-repetitions s1 n1 s2 n2)
(-> string? exact-integer? string? exact-integer? exact-integer?)
)
```