Problem 854: K-Similar Strings

Problem Information

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

Problem Description

Strings

s1

and

s2

are

k

-similar

(for some non-negative integer

k

) if we can swap the positions of two letters in

s1

exactly

k

times so that the resulting string equals

s2

.

Given two anagrams

s1

and

s2

, return the smallest

k

for which

s1

and

s2

are

k

-similar

.

Example 1:

Input:

s1 = "ab", s2 = "ba"

Output:

1

Explanation:

The two string are 1-similar because we can use one swap to change s1 to s2: "ab" --> "ba".

Example 2:

Input:

s1 = "abc", s2 = "bca"

Output:

2

Explanation:

The two strings are 2-similar because we can use two swaps to change s1 to s2: "abc" --> "bac" --> "bca".

Constraints:

1 <= s1.length <= 20

s2.length == s1.length

s1

and

s2

contain only lowercase letters from the set

{'a', 'b', 'c', 'd', 'e', 'f'}

.

s2

is an anagram of

s1

.

Code Snippets

C++:

```
class Solution {
public:
int kSimilarity(string s1, string s2) {

}
};
```

Java:

```
class Solution {
public int kSimilarity(String s1, String s2) {

}
}
```

Python3:

```
class Solution:
def kSimilarity(self, s1: str, s2: str) -> int:
```

Python:

```
class Solution(object):
def kSimilarity(self, s1, s2):
"""
:type s1: str
```

```
:type s2: str
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function kSimilarity(s1: string, s2: string): number {

};
```

C#:

```
public class Solution {
public int KSimilarity(string s1, string s2) {

}
}
```

C:

```
int kSimilarity(char* s1, char* s2) {

}
```

Go:

```
func kSimilarity(s1 string, s2 string) int {

}
```

Kotlin:

```
class Solution {
fun kSimilarity(s1: String, s2: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String} s1
# @param {String} s2
# @return {Integer}
def k_similarity(s1, s2)

end
```

PHP:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @return Integer
*/
function kSimilarity($s1, $s2) {

}
```

```
}
```

Dart:

```
class Solution {
int kSimilarity(String s1, String s2) {

}
}
```

Scala:

```
object Solution {
def kSimilarity(s1: String, s2: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec k_similarity(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> integer().
k_similarity(S1, S2) ->
.
```

Racket:

```
(define/contract (k-similarity s1 s2)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: K-Similar Strings
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int kSimilarity(string s1, string s2) {

}
};
```

Java Solution:

```
/**
* Problem: K-Similar Strings
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int kSimilarity(String s1, String s2) {

}
}
```

Python3 Solution:

```
"""
Problem: K-Similar Strings
```

```
Difficulty: Hard
Tags: string, hash, search

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def kSimilarity(self, s1: str, s2: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: K-Similar Strings
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: K-Similar Strings
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function kSimilarity(s1: string, s2: string): number {

};
```

C# Solution:

```
/*
* Problem: K-Similar Strings
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int KSimilarity(string s1, string s2) {

}
}
```

C Solution:

```
/*
* Problem: K-Similar Strings
```

```
* Difficulty: Hard
* Tags: string, hash, search
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int kSimilarity(char* s1, char* s2) {

}
```

Go Solution:

```
// Problem: K-Similar Strings
// Difficulty: Hard
// Tags: string, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func kSimilarity(s1 string, s2 string) int {

}
```

Kotlin Solution:

```
class Solution {
fun kSimilarity(s1: String, s2: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: K-Similar Strings
// Difficulty: Hard
// Tags: string, hash, search
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn k_similarity(s1: String, s2: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s1
# @param {String} s2
# @return {Integer}
def k_similarity(s1, s2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @return Integer
*/
function kSimilarity($s1, $s2) {

}
}
```

Dart Solution:

```
class Solution {
int kSimilarity(String s1, String s2) {

}
}
```

Scala Solution:

```
object Solution {
def kSimilarity(s1: String, s2: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec k_similarity(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary()) -> integer().
k_similarity(S1, S2) ->
.
```

Racket Solution:

```
(define/contract (k-similarity s1 s2)
(-> string? string? exact-integer?)
)
```