Problem 521: Longest Uncommon Subsequence I

Problem Information

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

Problem Description

Given two strings

a

and

b

, return

the length of the

longest uncommon subsequence

between

a

and

b

.

If no such uncommon subsequence exists, return

-1

.

An

uncommon subsequence

between two strings is a string that is a

subsequence

of exactly one of them

.

Example 1:

Input:

a = "aba", b = "cdc"

Output:

3

Explanation:

One longest uncommon subsequence is "aba" because "aba" is a subsequence of "aba" but not "cdc". Note that "cdc" is also a longest uncommon subsequence.

Example 2:

Input:

a = "aaa", b = "bbb"

Output:

3

Explanation:

The longest uncommon subsequences are "aaa" and "bbb".

Example 3:

Input:

a = "aaa", b = "aaa"

Output:

-1

Explanation:

Every subsequence of string a is also a subsequence of string b. Similarly, every subsequence of string b is also a subsequence of string a. So the answer would be

-1

.

Constraints:

1 <= a.length, b.length <= 100

a

and

b

consist of lower-case English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
```

```
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (find-lu-slength a b)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Uncommon Subsequence I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Longest Uncommon Subsequence I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int findLUSlength(String a, String b) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Uncommon Subsequence I
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def findLUSlength(self, a: str, b: str) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Uncommon Subsequence I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest Uncommon Subsequence I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function findLUSlength(a: string, b: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Uncommon Subsequence I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int FindLUSlength(string a, string b) {

}
}
```

C Solution:

```
/*
* Problem: Longest Uncommon Subsequence I
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int findLUSlength(char* a, char* b) {

}
```

Go Solution:

```
// Problem: Longest Uncommon Subsequence I
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func findLUSlength(a string, b string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Uncommon Subsequence I
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

```
}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
```

```
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (find-lu-slength a b)
(-> string? string? exact-integer?)
)
```