Problem 2937: Make Three Strings Equal

Problem Information

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

Problem Description

You are given three strings:

s1

,

s2

, and

s3

. In one operation you can choose one of these strings and delete its

rightmost

character. Note that you

cannot

completely empty a string.

Return the

minimum number of operations

required to make the strings equal

.

If it is impossible to make them equal, return

-1

.

Example 1:

Input:

s1 = "abc", s2 = "abb", s3 = "ab"

Output:

2

Explanation:

Deleting the rightmost character from both

s1

and

s2

will result in three equal strings.

Example 2:

Input:

s1 = "dac", s2 = "bac", s3 = "cac"

Output:

-1

Explanation:

Since the first letters of

s1

and

s2

differ, they cannot be made equal.

Constraints:

1 <= s1.length, s2.length, s3.length <= 100

s1

,

s2

and

s3

consist only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int findMinimumOperations(string s1, string s2, string s3) {
}
};
```

Java:

```
class Solution {
public int findMinimumOperations(String s1, String s2, String s3) {

}
}
```

Python3:

```
class Solution:
def findMinimumOperations(self, s1: str, s2: str, s3: str) -> int:
```

Python:

```
class Solution(object):
def findMinimumOperations(self, s1, s2, s3):
"""
:type s1: str
:type s2: str
:type s3: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s1
* @param {string} s2
* @param {string} s3
* @return {number}
*/
var findMinimumOperations = function(s1, s2, s3) {

};
```

TypeScript:

```
function findMinimumOperations(s1: string, s2: string, s3: string): number {


};
```

C#:

```
public class Solution {
public int FindMinimumOperations(string s1, string s2, string s3) {


}
}
```

C:

```
int findMinimumOperations(char* s1, char* s2, char* s3) {


}
```

Go:

```
func findMinimumOperations(s1 string, s2 string, s3 string) int {


}
```

Kotlin:

```
class Solution {
fun findMinimumOperations(s1: String, s2: String, s3: String): Int {


}
}
```

Swift:

```
class Solution {
func findMinimumOperations(_ s1: String, _ s2: String, _ s3: String) -> Int {


}
}
```

Rust:

```
impl Solution {
pub fn find_minimum_operations(s1: String, s2: String, s3: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s1
# @param {String} s2
# @param {String} s3
# @return {Integer}
def find_minimum_operations(s1, s2, s3)

end
```

PHP:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @param String $s3
* @return Integer
*/
function findMinimumOperations($s1, $s2, $s3) {

}
}
```

Dart:

```
class Solution {
int findMinimumOperations(String s1, String s2, String s3) {

}
}
```

Scala:

```
object Solution {
def findMinimumOperations(s1: String, s2: String, s3: String): Int = {
}
}
```

Elixir:

```
defmodule Solution do
@spec find_minimum_operations(s1 :: String.t, s2 :: String.t, s3 :: String.t)
:: integer
def find_minimum_operations(s1, s2, s3) do

end
end
```

Erlang:

```
-spec find_minimum_operations(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary(), S3 :: unicode:unicode_binary()) -> integer().
find_minimum_operations(S1, S2, S3) ->
.
```

Racket:

```
(define/contract (find-minimum-operations s1 s2 s3)
(-> string? string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Make Three Strings Equal
* 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 findMinimumOperations(string s1, string s2, string s3) {

}
};
```

Java Solution:

```
/**
* Problem: Make Three Strings Equal
* 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 findMinimumOperations(String s1, String s2, String s3) {

}
}
```

Python3 Solution:

```
"""
Problem: Make Three Strings Equal
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 findMinimumOperations(self, s1: str, s2: str, s3: str) -> int:
# TODO: Implement optimized solution
```

```
pass
```

Python Solution:

```
class Solution(object):
def findMinimumOperations(self, s1, s2, s3):
"""
:type s1: str
:type s2: str
:type s3: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Make Three Strings Equal
* 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} s1
* @param {string} s2
* @param {string} s3
* @return {number}
*/
var findMinimumOperations = function(s1, s2, s3) {

};
```

TypeScript Solution:

```
/**
* Problem: Make Three Strings Equal
* 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 findMinimumOperations(s1: string, s2: string, s3: string): number {

};
```

C# Solution:

```
/*
* Problem: Make Three Strings Equal
* 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 FindMinimumOperations(string s1, string s2, string s3) {

}
}
```

C Solution:

```
/*
* Problem: Make Three Strings Equal
* 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 findMinimumOperations(char* s1, char* s2, char* s3) {
}
```

Go Solution:

```go
// Problem: Make Three Strings Equal
// 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 findMinimumOperations(s1 string, s2 string, s3 string) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun findMinimumOperations(s1: String, s2: String, s3: String): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func findMinimumOperations(_ s1: String, _ s2: String, _ s3: String) -> Int {

}
}
```

Rust Solution:

```rust
// Problem: Make Three Strings Equal
// 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_minimum_operations(s1: String, s2: String, s3: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s1
# @param {String} s2
# @param {String} s3
# @return {Integer}
def find_minimum_operations(s1, s2, s3)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s1
* @param String $s2
* @param String $s3
* @return Integer
*/
function findMinimumOperations($s1, $s2, $s3) {

}
}
```

Dart Solution:

```
class Solution {
int findMinimumOperations(String s1, String s2, String s3) {

}
}
```

Scala Solution:

```
object Solution {
def findMinimumOperations(s1: String, s2: String, s3: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec find_minimum_operations(s1 :: String.t, s2 :: String.t, s3 :: String.t)
:: integer
def find_minimum_operations(s1, s2, s3) do

end
end
```

Erlang Solution:

```
-spec find_minimum_operations(S1 :: unicode:unicode_binary(), S2 ::
unicode:unicode_binary(), S3 :: unicode:unicode_binary()) -> integer().
find_minimum_operations(S1, S2, S3) ->
.
```

Racket Solution:

```
(define/contract (find-minimum-operations s1 s2 s3)
(-> string? string? string? exact-integer?)
)
```