Problem 3579: Minimum Steps to Convert String with Operations

Problem Information

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

Problem Description

You are given two strings,

word1

and

word2

, of equal length. You need to transform

word1

into

word2

.

For this, divide

word1

into one or more

contiguous

substrings

. For each substring

substr

you can perform the following operations:

Replace:

Replace the character at any one index of

substr

with another lowercase English letter.

Swap:

Swap any two characters in

substr

.

Reverse Substring:

Reverse

substr

.

Each of these counts as

one

operation and each character of each substring can be used in each type of operation at most once (i.e. no single index may be involved in more than one replace, one swap, or one reverse).

Return the

minimum number of operations

required to transform

word1

into

word2

.

Example 1:

Input:

word1 = "abcdf", word2 = "dacbe"

Output:

4

Explanation:

Divide

word1

into

"ab"

,

"c"

, and

"df"

. The operations are:

For the substring

"ab"

,

Perform operation of type 3 on

"ab" -> "ba"

.

Perform operation of type 1 on

"ba" -> "da"

.

For the substring

"c"

do no operations.

For the substring

"df"

,

Perform operation of type 1 on

"df" -> "bf"

.

Perform operation of type 1 on

"bf" -> "be"

.

Example 2:

Input:

word1 = "abceded", word2 = "baecfef"

Output:

4

Explanation:

Divide

word1

into

"ab"

,

"ce"

, and

"ded"

. The operations are:

For the substring

"ab"

,

Perform operation of type 2 on

"ab" -> "ba"

.

For the substring

"ce"

,

Perform operation of type 2 on

"ce" -> "ec"

.

For the substring

"ded"

,

Perform operation of type 1 on

"ded" -> "fed"

.

Perform operation of type 1 on

"fed" -> "fef"

.

Example 3:

Input:

word1 = "abcdef", word2 = "fedabc"

Output:

2

Explanation:

Divide

word1

into

"abcdef"

. The operations are:

For the substring

"abcdef"

,

Perform operation of type 3 on

"abcdef" -> "fedcba"

.

Perform operation of type 2 on

"fedcba" -> "fedabc"

.

Constraints:

1 <= word1.length == word2.length <= 100

word1

and

word2

consist only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int minOperations(string word1, string word2) {

}
};
```

Java:

```
class Solution {
public int minOperations(String word1, String word2) {

}
}
```

Python3:

```
class Solution:
def minOperations(self, word1: str, word2: str) -> int:
```

Python:

```
class Solution(object):
def minOperations(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} word1
* @param {string} word2
* @return {number}
*/
var minOperations = function(word1, word2) {

};
```

TypeScript:

```
function minOperations(word1: string, word2: string): number {

};
```

C#:

```
public class Solution {
public int MinOperations(string word1, string word2) {

}
}
```

C:

```
int minOperations(char* word1, char* word2) {

}
```

Go:

```
func minOperations(word1 string, word2 string) int {

}
```

Kotlin:

```
class Solution {
fun minOperations(word1: String, word2: String): Int {

}
}
```

Swift:

```
class Solution {
func minOperations(_ word1: String, _ word2: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn min_operations(word1: String, word2: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} word1
# @param {String} word2
# @return {Integer}
def min_operations(word1, word2)

end
```

PHP:

```
class Solution {

/**
* @param String $word1
```

```
* @param String $word2
* @return Integer
*/
function minOperations($word1, $word2) {

}
}
```

Dart:

```
class Solution {
int minOperations(String word1, String word2) {

}
}
```

Scala:

```
object Solution {
def minOperations(word1: String, word2: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec min_operations(word1 :: String.t, word2 :: String.t) :: integer
def min_operations(word1, word2) do

end
end
```

Erlang:

```
-spec min_operations(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
min_operations(Word1, Word2) ->
.
```

Racket:

```
(define/contract (min-operations word1 word2)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Steps to Convert String with Operations
* Difficulty: Hard
* Tags: string, tree, dp, greedy
*
* 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 minOperations(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Steps to Convert String with Operations
* Difficulty: Hard
* Tags: string, tree, dp, greedy
*
* 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 minOperations(String word1, String word2) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Minimum Steps to Convert String with Operations
Difficulty: Hard
Tags: string, tree, dp, greedy

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 minOperations(self, word1: str, word2: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def minOperations(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Minimum Steps to Convert String with Operations
* Difficulty: Hard
* Tags: string, tree, dp, greedy
*
* 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} word1
* @param {string} word2
* @return {number}
*/
var minOperations = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Steps to Convert String with Operations
* Difficulty: Hard
* Tags: string, tree, dp, greedy
*
* 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 minOperations(word1: string, word2: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Steps to Convert String with Operations
* Difficulty: Hard
* Tags: string, tree, dp, greedy
*
* 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 MinOperations(string word1, string word2) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Steps to Convert String with Operations
* Difficulty: Hard
* Tags: string, tree, dp, greedy
*
* 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 minOperations(char* word1, char* word2) {

}
```

Go Solution:

```
// Problem: Minimum Steps to Convert String with Operations
// Difficulty: Hard
// Tags: string, tree, dp, greedy
//
// 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 minOperations(word1 string, word2 string) int {

}
```

Kotlin Solution:

```
class Solution {
fun minOperations(word1: String, word2: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func minOperations(_ word1: String, _ word2: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Minimum Steps to Convert String with Operations
// Difficulty: Hard
// Tags: string, tree, dp, greedy
//
// 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 min_operations(word1: String, word2: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} word1
# @param {String} word2
# @return {Integer}
def min_operations(word1, word2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Integer
*/
function minOperations($word1, $word2) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int minOperations(String word1, String word2) {

}
}
```

Scala Solution:

```
object Solution {
def minOperations(word1: String, word2: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec min_operations(word1 :: String.t, word2 :: String.t) :: integer
def min_operations(word1, word2) do

end
end
```

Erlang Solution:

```
-spec min_operations(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
min_operations(Word1, Word2) ->
.
```

Racket Solution:

```
(define/contract (min-operations word1 word2)
(-> string? string? exact-integer?)
)
```