Problem 583: Delete Operation for Two Strings

Problem Information

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

Problem Description

Given two strings

word1

and

word2

, return

the minimum number of

steps

required to make

word1

and

word2

the same

.

In one

step

, you can delete exactly one character in either string.

Example 1:

Input:

word1 = "sea", word2 = "eat"

Output:

2

Explanation:

You need one step to make "sea" to "ea" and another step to make "eat" to "ea".

Example 2:

Input:

word1 = "leetcode", word2 = "etco"

Output:

4

Constraints:

1 <= word1.length, word2.length <= 500

word1

and

word2

consist of only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} word1
* @param {string} word2
```

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
```

```
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Delete Operation for Two Strings
* Difficulty: Medium
* 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 minDistance(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Delete Operation for Two Strings
* Difficulty: Medium
* 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 minDistance(String word1, String word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Delete Operation for Two Strings
Difficulty: Medium
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 minDistance(self, word1: str, word2: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Delete Operation for Two Strings
* Difficulty: Medium
* 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} word1
* @param {string} word2
* @return {number}
*/
var minDistance = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Delete Operation for Two Strings
* Difficulty: Medium
* 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 minDistance(word1: string, word2: string): number {

};
```

C# Solution:

```
/*
* Problem: Delete Operation for Two Strings
* Difficulty: Medium
* 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 MinDistance(string word1, string word2) {

}
}
```

C Solution:

```
/*
* Problem: Delete Operation for Two Strings
* Difficulty: Medium
* 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 minDistance(char* word1, char* word2) {

}
```

Go Solution:

```
// Problem: Delete Operation for Two Strings
// Difficulty: Medium
// 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 minDistance(word1 string, word2 string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Delete Operation for Two Strings
// Difficulty: Medium
// 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 min_distance(word1: String, word2: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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