Problem 1657: Determine if Two Strings Are Close

Problem Information

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

Problem Description

Two strings are considered

close

if you can attain one from the other using the following operations:

Operation 1: Swap any two

existing

characters.

For example,

a

b

cd

e

-> a

e

cd

b

Operation 2: Transform

every

occurrence of one

existing

character into another

existing

character, and do the same with the other character.

For example,

aa

c

abb

->

bb

c

baa

(all

a

's turn into

b

's, and all

b

's turn into

a

's)

You can use the operations on either string as many times as necessary.

Given two strings,

word1

and

word2

, return

true

if

word1

and

word2

are

close

, and

false

otherwise.

Example 1:

Input:

word1 = "abc", word2 = "bca"

Output:

true

Explanation:

You can attain word2 from word1 in 2 operations. Apply Operation 1: "a

bc

" -> "a

cb

" Apply Operation 1: "

a

c

b

" -> "

b

c

a

"

Example 2:

Input:

word1 = "a", word2 = "aa"

Output:

false

Explanation:

It is impossible to attain word2 from word1, or vice versa, in any number of operations.

Example 3:

Input:

word1 = "cabbba", word2 = "abbccc"

Output:

true

Explanation:

You can attain word2 from word1 in 3 operations. Apply Operation 1: "ca

b

bb

a

" -> "ca

a

bb

b

" Apply Operation 2: "

c

aa

bbb

" -> "

b

aa

ccc

" Apply Operation 2: "

baa

ccc" -> "

abb

ccc"

Constraints:

1 <= word1.length, word2.length <= 10

5

word1

and

word2

contain only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
bool closeStrings(string word1, string word2) {

}
};
```

Java:

```
class Solution {
public boolean closeStrings(String word1, String word2) {

}
}
```

Python3:

```
class Solution:
def closeStrings(self, word1: str, word2: str) -> bool:
```

Python:

```
class Solution(object):
def closeStrings(self, word1, word2):
"""
:type word1: str
```

```
:type word2: str
:rtype: bool
"""
```

JavaScript:

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

};
```

TypeScript:

```
function closeStrings(word1: string, word2: string): boolean {

};
```

C#:

```
public class Solution {
public bool CloseStrings(string word1, string word2) {

}
}
```

C:

```
bool closeStrings(char* word1, char* word2) {

}
```

Go:

```
func closeStrings(word1 string, word2 string) bool {

}
```

Kotlin:

```
class Solution {
fun closeStrings(word1: String, word2: String): Boolean {

}
}
```

Swift:

```
class Solution {
func closeStrings(_ word1: String, _ word2: String) -> Bool {

}
}
```

Rust:

```
impl Solution {
pub fn close_strings(word1: String, word2: String) -> bool {

}
}
```

Ruby:

```
# @param {String} word1
# @param {String} word2
# @return {Boolean}
def close_strings(word1, word2)

end
```

PHP:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Boolean
*/
function closeStrings($word1, $word2) {

}
```

```
}
```

Dart:

```
class Solution {
bool closeStrings(String word1, String word2) {

}
}
```

Scala:

```
object Solution {
def closeStrings(word1: String, word2: String): Boolean = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec close_strings(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> boolean().
close_strings(Word1, Word2) ->
.
```

Racket:

```
(define/contract (close-strings word1 word2)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Determine if Two Strings Are Close
* Difficulty: Medium
* Tags: string, hash, sort
*
* 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:
bool closeStrings(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Determine if Two Strings Are Close
* Difficulty: Medium
* Tags: string, hash, sort
*
* 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 boolean closeStrings(String word1, String word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Determine if Two Strings Are Close
```

```
Difficulty: Medium
Tags: string, hash, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Determine if Two Strings Are Close
* Difficulty: Medium
* Tags: string, hash, sort
*
* 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} word1
* @param {string} word2
* @return {boolean}
*/
var closeStrings = function(word1, word2) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Determine if Two Strings Are Close
* Difficulty: Medium
* Tags: string, hash, sort
*
* 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 closeStrings(word1: string, word2: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Determine if Two Strings Are Close
* Difficulty: Medium
* Tags: string, hash, sort
*
* 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 bool CloseStrings(string word1, string word2) {

}
}
```

C Solution:

```
/*
* Problem: Determine if Two Strings Are Close
```

```
* Difficulty: Medium
* Tags: string, hash, sort
*
* 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
*/

bool closeStrings(char* word1, char* word2) {

}
```

Go Solution:

```
// Problem: Determine if Two Strings Are Close
// Difficulty: Medium
// Tags: string, hash, sort
//
// 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 closeStrings(word1 string, word2 string) bool {

}
```

Kotlin Solution:

```
class Solution {
fun closeStrings(word1: String, word2: String): Boolean {

}
}
```

Swift Solution:

```
class Solution {
func closeStrings(_ word1: String, _ word2: String) -> Bool {

}
}
```

Rust Solution:

```
// Problem: Determine if Two Strings Are Close
// Difficulty: Medium
// Tags: string, hash, sort
//
// 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 close_strings(word1: String, word2: String) -> bool {

}
}
```

Ruby Solution:

```
# @param {String} word1
# @param {String} word2
# @return {Boolean}
def close_strings(word1, word2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Boolean
*/
function closeStrings($word1, $word2) {

}
}
```

Dart Solution:

```
class Solution {
bool closeStrings(String word1, String word2) {

}
}
```

Scala Solution:

```
object Solution {
def closeStrings(word1: String, word2: String): Boolean = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec close_strings(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> boolean().
close_strings(Word1, Word2) ->
.
```

Racket Solution:

```
(define/contract (close-strings word1 word2)
(-> string? string? boolean?)
)
```