Problem 2068: Check Whether Two Strings are Almost Equivalent

Problem Information

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

Problem Description

Two strings

word1

and

word2

are considered

almost equivalent

if the differences between the frequencies of each letter from

'a'

to

'z'

between

word1

and

word2

is

at most

3

.

Given two strings

word1

and

word2

, each of length

n

, return

true

if

word1

and

word2

are

almost equivalent

, or

false

otherwise

.

The

frequency

of a letter

x

is the number of times it occurs in the string.

Example 1:

Input:

word1 = "aaaa", word2 = "bccb"

Output:

false

Explanation:

There are 4 'a's in "aaaa" but 0 'a's in "bccb". The difference is 4, which is more than the allowed 3.

Example 2:

Input:

word1 = "abcdeef", word2 = "abaaacc"

Output:

true

Explanation:

The differences between the frequencies of each letter in word1 and word2 are at most 3: - 'a' appears 1 time in word1 and 4 times in word2. The difference is 3. - 'b' appears 1 time in word1 and 1 time in word2. The difference is 0. - 'c' appears 1 time in word1 and 2 times in word2. The difference is 1. - 'd' appears 1 time in word1 and 0 times in word2. The difference is 1. - 'e' appears 2 times in word1 and 0 times in word2. The difference is 2. - 'f' appears 1 time in word1 and 0 times in word2. The difference is 1.

Example 3:

Input:

word1 = "cccddabba", word2 = "babababab"

Output:

true

Explanation:

The differences between the frequencies of each letter in word1 and word2 are at most 3: - 'a' appears 2 times in word1 and 4 times in word2. The difference is 2. - 'b' appears 2 times in word1 and 5 times in word2. The difference is 3. - 'c' appears 3 times in word1 and 0 times in word2. The difference is 3. - 'd' appears 2 times in word1 and 0 times in word2. The difference is 2.

Constraints:

n == word1.length == word2.length

1 <= n <= 100

word1

and

word2

consist only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
bool checkAlmostEquivalent(String word1, String word2) {
}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (check-almost-equivalent word1 word2)
(-> string? string? boolean?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Check Whether Two Strings are Almost Equivalent
* Difficulty: Easy
* Tags: string, hash
*
* 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 checkAlmostEquivalent(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Check Whether Two Strings are Almost Equivalent
* Difficulty: Easy
* Tags: string, hash
*
* 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 checkAlmostEquivalent(String word1, String word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Check Whether Two Strings are Almost Equivalent
Difficulty: Easy
Tags: string, hash
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Check Whether Two Strings are Almost Equivalent
* Difficulty: Easy
* Tags: string, hash
*
* 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 checkAlmostEquivalent = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Check Whether Two Strings are Almost Equivalent
* Difficulty: Easy
* Tags: string, hash
*
* 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 checkAlmostEquivalent(word1: string, word2: string): boolean {

};
```

C# Solution:

```
/*
* Problem: Check Whether Two Strings are Almost Equivalent
* Difficulty: Easy
* Tags: string, hash
*
* 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 CheckAlmostEquivalent(string word1, string word2) {

}
}
```

C Solution:

```
/*
* Problem: Check Whether Two Strings are Almost Equivalent
* Difficulty: Easy
* Tags: string, hash
*
* 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 checkAlmostEquivalent(char* word1, char* word2) {

}
```

Go Solution:

```
// Problem: Check Whether Two Strings are Almost Equivalent
// Difficulty: Easy
// Tags: string, hash
//
// 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 checkAlmostEquivalent(word1 string, word2 string) bool {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Check Whether Two Strings are Almost Equivalent
// Difficulty: Easy
// Tags: string, hash
//
// 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 check_almost_equivalent(word1: String, word2: String) -> bool {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}

}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (check-almost-equivalent word1 word2)
(-> string? string? boolean?)
)
```