Problem 2085: Count Common Words With One Occurrence

Problem Information

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

Problem Description

Given two string arrays

words1

and

words2

, return

the number of strings that appear

exactly once

in

each

of the two arrays.

Example 1:

Input:

words1 = ["leetcode","is","amazing","as","is"], words2 = ["amazing","leetcode","is"]

Output:

2

Explanation:

- "leetcode" appears exactly once in each of the two arrays. We count this string. - "amazing" appears exactly once in each of the two arrays. We count this string. - "is" appears in each of the two arrays, but there are 2 occurrences of it in words1. We do not count this string. - "as" appears once in words1, but does not appear in words2. We do not count this string. Thus, there are 2 strings that appear exactly once in each of the two arrays.

Example 2:

Input:

words1 = ["b","bb","bbb"], words2 = ["a","aa","aaa"]

Output:

0

Explanation:

There are no strings that appear in each of the two arrays.

Example 3:

Input:

words1 = ["a","ab"], words2 = ["a","a","a","ab"]

Output:

1

Explanation:

The only string that appears exactly once in each of the two arrays is "ab".

Constraints:

1 <= words1.length, words2.length <= 1000

1 <= words1[i].length, words2[j].length <= 30

words1[i]

and

words2[j]

consists only of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int countWords(vector<string>& words1, vector<string>& words2) {

}
};
```

Java:

```
class Solution {
public int countWords(String[] words1, String[] words2) {

}
}
```

Python3:

```
class Solution:
def countWords(self, words1: List[str], words2: List[str]) -> int:
```

Python:

```
class Solution(object):
def countWords(self, words1, words2):
"""
:type words1: List[str]
:type words2: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} words1
* @param {string[]} words2
* @return {number}
*/
var countWords = function(words1, words2) {

};
```

TypeScript:

```
function countWords(words1: string[], words2: string[]): number {

};
```

C#:

```
public class Solution {
public int CountWords(string[] words1, string[] words2) {

}
}
```

C:

```
int countWords(char** words1, int words1Size, char** words2, int words2Size)
{

}
```

Go:

```
func countWords(words1 []string, words2 []string) int {

}
```

Kotlin:

```
class Solution {
fun countWords(words1: Array<String>, words2: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func countWords(_ words1: [String], _ words2: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_words(words1: Vec<String>, words2: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} words1
# @param {String[]} words2
# @return {Integer}
def count_words(words1, words2)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words1
```

```
* @param String[] $words2
* @return Integer
*/
function countWords($words1, $words2) {

}
}
```

Dart:

```
class Solution {
int countWords(List<String> words1, List<String> words2) {

}
}
```

Scala:

```
object Solution {
def countWords(words1: Array[String], words2: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_words(words1 :: [String.t], words2 :: [String.t]) :: integer
def count_words(words1, words2) do

end
end
```

Erlang:

```
-spec count_words(Words1 :: [unicode:unicode_binary()], Words2 ::
[unicode:unicode_binary()]) -> integer().
count_words(Words1, Words2) ->
.
```

Racket:

```
(define/contract (count-words words1 words2)
(-> (listof string?) (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Common Words With One Occurrence
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int countWords(vector<string>& words1, vector<string>& words2) {

}
};
```

Java Solution:

```
/**
* Problem: Count Common Words With One Occurrence
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int countWords(String[] words1, String[] words2) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count Common Words With One Occurrence
Difficulty: Easy
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def countWords(self, words1: List[str], words2: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countWords(self, words1, words2):
"""
:type words1: List[str]
:type words2: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Common Words With One Occurrence
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/
```

```
/**
* @param {string[]} words1
* @param {string[]} words2
* @return {number}
*/
var countWords = function(words1, words2) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Common Words With One Occurrence
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function countWords(words1: string[], words2: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Common Words With One Occurrence
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int CountWords(string[] words1, string[] words2) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Count Common Words With One Occurrence
* Difficulty: Easy
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int countWords(char** words1, int words1Size, char** words2, int words2Size)
{

}
```

Go Solution:

```
// Problem: Count Common Words With One Occurrence
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countWords(words1 []string, words2 []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countWords(words1: Array<String>, words2: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func countWords(_ words1: [String], _ words2: [String]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Common Words With One Occurrence
// Difficulty: Easy
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

impl Solution {
pub fn count_words(words1: Vec<String>, words2: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words1
# @param {String[]} words2
# @return {Integer}
def count_words(words1, words2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words1
* @param String[] $words2
* @return Integer
*/
```

```
function countWords($words1, $words2) {

}
}
```

Dart Solution:

```
class Solution {
int countWords(List<String> words1, List<String> words2) {

}
}
```

Scala Solution:

```
object Solution {
def countWords(words1: Array[String], words2: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_words(words1 :: [String.t], words2 :: [String.t]) :: integer
def count_words(words1, words2) do

end
end
```

Erlang Solution:

```
-spec count_words(Words1 :: [unicode:unicode_binary()], Words2 ::
[unicode:unicode_binary()]) -> integer().
count_words(Words1, Words2) ->
.
```

Racket Solution:

```
(define/contract (count-words words1 words2)
(-> (listof string?) (listof string?) exact-integer?)
```

```
)
```