Problem 2506: Count Pairs Of Similar Strings

Problem Information

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

Problem Description

You are given a

0-indexed

string array

words

.

Two strings are

similar

if they consist of the same characters.

For example,

"abca"

and

"cba"

are similar since both consist of characters

'a'

,

'b'

, and

'c'

.

However,

"abacba"

and

"bcfd"

are not similar since they do not consist of the same characters.

Return

the number of pairs

(i, j)

such that

0 <= i < j <= word.length - 1

and the two strings

words[i]

and

words[j]

are similar

.

Example 1:

Input:

words = ["aba","aabb","abcd","bac","aabc"]

Output:

2

Explanation:

There are 2 pairs that satisfy the conditions: - i = 0 and j = 1 : both words[0] and words[1] only consist of characters 'a' and 'b'. - i = 3 and j = 4 : both words[3] and words[4] only consist of characters 'a', 'b', and 'c'.

Example 2:

Input:

words = ["aabb","ab","ba"]

Output:

3

Explanation:

There are 3 pairs that satisfy the conditions: - i = 0 and j = 1 : both words[0] and words[1] only consist of characters 'a' and 'b'. - i = 0 and j = 2 : both words[0] and words[2] only consist of characters 'a' and 'b'. - i = 1 and j = 2 : both words[1] and words[2] only consist of characters 'a' and 'b'.

Example 3:

Input:

words = ["nba","cba","dba"]

Output:

0

Explanation:

Since there does not exist any pair that satisfies the conditions, we return 0.

Constraints:

1 <= words.length <= 100

1 <= words[i].length <= 100

words[i]

consist of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int similarPairs(vector<string>& words) {

}
};
```

Java:

```
class Solution {
public int similarPairs(String[] words) {

}
```

```
}
```

Python3:

```
class Solution:
def similarPairs(self, words: List[str]) -> int:
```

Python:

```
class Solution(object):
def similarPairs(self, words):
"""
:type words: List[str]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} words
* @return {number}
*/
var similarPairs = function(words) {

};
```

TypeScript:

```
function similarPairs(words: string[]): number {

};
```

C#:

```
public class Solution {
public int SimilarPairs(string[] words) {

}
}
```

C:

```
int similarPairs(char** words, int wordsSize) {

}
```

Go:

```
func similarPairs(words []string) int {

}
```

Kotlin:

```
class Solution {
fun similarPairs(words: Array<String>): Int {

}
}
```

Swift:

```
class Solution {
func similarPairs(_ words: [String]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn similar_pairs(words: Vec<String>) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} words
# @return {Integer}
def similar_pairs(words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @return Integer
*/
function similarPairs($words) {

}
}
```

Dart:

```
class Solution {
int similarPairs(List<String> words) {

}
}
```

Scala:

```
object Solution {
def similarPairs(words: Array[String]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec similar_pairs(words :: [String.t]) :: integer
def similar_pairs(words) do

end
end
```

Erlang:

```
-spec similar_pairs(Words :: [unicode:unicode_binary()]) -> integer().
similar_pairs(Words) ->
.
```

Racket:

```
(define/contract (similar-pairs words)
(-> (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Pairs Of Similar Strings
* 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 similarPairs(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Count Pairs Of Similar Strings
* 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 similarPairs(String[] words) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Count Pairs Of Similar Strings
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 similarPairs(self, words: List[str]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def similarPairs(self, words):
"""
:type words: List[str]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Pairs Of Similar Strings
* 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[]} words
* @return {number}
*/
var similarPairs = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Pairs Of Similar Strings
* 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 similarPairs(words: string[]): number {

};
```

C# Solution:

```
/*
* Problem: Count Pairs Of Similar Strings
* 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 SimilarPairs(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Count Pairs Of Similar Strings
* 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 similarPairs(char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Count Pairs Of Similar Strings
// 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 similarPairs(words []string) int {

}
```

Kotlin Solution:

```
class Solution {
fun similarPairs(words: Array<String>): Int {

}
}
```

Swift Solution:

```
class Solution {
func similarPairs(_ words: [String]) -> Int {
```

```
}
}
```

Rust Solution:

```
// Problem: Count Pairs Of Similar Strings
// 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 similar_pairs(words: Vec<String>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {Integer}
def similar_pairs(words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return Integer
*/
function similarPairs($words) {

}
}
```

Dart Solution:

```
class Solution {
int similarPairs(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def similarPairs(words: Array[String]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec similar_pairs(words :: [String.t]) :: integer
def similar_pairs(words) do

end
end
```

Erlang Solution:

```
-spec similar_pairs(Words :: [unicode:unicode_binary()]) -> integer().
similar_pairs(Words) ->
.
```

Racket Solution:

```
(define/contract (similar-pairs words)
(-> (listof string?) exact-integer?)
)
```