Problem 1160: Find Words That Can Be Formed by Characters

Problem Information

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

Problem Description

You are given an array of strings

words

and a string

chars

.

A string is

good

if it can be formed by characters from

chars

(each character can only be used once for

each

word in

words

).

Return

the sum of lengths of all good strings in words

.

Example 1:

Input:

words = ["cat","bt","hat","tree"], chars = "atach"

Output:

6

Explanation:

The strings that can be formed are "cat" and "hat" so the answer is 3 + 3 = 6.

Example 2:

Input:

words = ["hello","world","leetcode"], chars = "welldonehoneyr"

Output:

10

Explanation:

The strings that can be formed are "hello" and "world" so the answer is 5 + 5 = 10.

Constraints:

1 <= words.length <= 1000

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

words[i]

and

chars

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int countCharacters(vector<string>& words, string chars) {

}
};
```

Java:

```
class Solution {
public int countCharacters(String[] words, String chars) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def countCharacters(self, words, chars):
```

```
"""
:type words: List[str]
:type chars: str
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function countCharacters(words: string[], chars: string): number {

};
```

C#:

```
public class Solution {
public int CountCharacters(string[] words, string chars) {

}
}
```

C:

```
int countCharacters(char** words, int wordsSize, char* chars) {

}
```

Go:

```
func countCharacters(words []string, chars string) int {

}
```

Kotlin:

```
class Solution {
fun countCharacters(words: Array<String>, chars: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String[]} words
# @param {String} chars
# @return {Integer}
def count_characters(words, chars)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @param String $chars
* @return Integer
*/
function countCharacters($words, $chars) {



}
}
```

Dart:

```
class Solution {
int countCharacters(List<String> words, String chars) {

}
}
```

Scala:

```
object Solution {
def countCharacters(words: Array[String], chars: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec count_characters(Words :: [unicode:unicode_binary()], Chars ::
unicode:unicode_binary()) -> integer().
count_characters(Words, Chars) ->
.
```

Racket:

```
(define/contract (count-characters words chars)
(-> (listof string?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Find Words That Can Be Formed by Characters
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int countCharacters(vector<string>& words, string chars) {

}
};
```

Java Solution:

```
/**
* Problem: Find Words That Can Be Formed by Characters
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int countCharacters(String[] words, String chars) {

}
}
```

Python3 Solution:

```
"""
Problem: Find Words That Can Be Formed by Characters
Difficulty: Easy
Tags: array, string, tree, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def countCharacters(self, words: List[str], chars: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Find Words That Can Be Formed by Characters
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

/**
* @param {string[]} words
* @param {string} chars
* @return {number}
*/
```

```
var countCharacters = function(words, chars) {

};
```

TypeScript Solution:

```
/**
* Problem: Find Words That Can Be Formed by Characters
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function countCharacters(words: string[], chars: string): number {

};
```

C# Solution:

```
/*
* Problem: Find Words That Can Be Formed by Characters
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int CountCharacters(string[] words, string chars) {

}
}
```

C Solution:

```
/*
* Problem: Find Words That Can Be Formed by Characters
* Difficulty: Easy
* Tags: array, string, tree, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int countCharacters(char** words, int wordsSize, char* chars) {

}
```

Go Solution:

```
// Problem: Find Words That Can Be Formed by Characters
// Difficulty: Easy
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func countCharacters(words []string, chars string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countCharacters(words: Array<String>, chars: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```rust
// Problem: Find Words That Can Be Formed by Characters
// Difficulty: Easy
// Tags: array, string, tree, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn count_characters(words: Vec<String>, chars: String) -> i32 {

}
}
```

Ruby Solution:

```ruby
# @param {String[]} words
# @param {String} chars
# @return {Integer}
def count_characters(words, chars)

end
```

PHP Solution:

```php
class Solution {

/**
* @param String[] $words
* @param String $chars
* @return Integer
*/
function countCharacters($words, $chars) {

}
}
```

Dart Solution:

```
class Solution {
int countCharacters(List<String> words, String chars) {

}
}
```

Scala Solution:

```
object Solution {
def countCharacters(words: Array[String], chars: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec count_characters(Words :: [unicode:unicode_binary()], Chars ::
unicode:unicode_binary()) -> integer().
count_characters(Words, Chars) ->
.
```

Racket Solution:

```
(define/contract (count-characters words chars)
(-> (listof string?) string? exact-integer?)
)
```