Problem 2185: Counting Words With a Given Prefix

Problem Information

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

Problem Description

You are given an array of strings

words

and a string

pref

.

Return

the number of strings in

words

that contain

pref

as a

prefix

.

A

prefix

of a string

s

is any leading contiguous substring of

s

.

Example 1:

Input:

words = ["pay","

at

tention","practice","

at

tend"],

pref

= "at"

Output:

2

Explanation:

The 2 strings that contain "at" as a prefix are: "

at

tention" and "

at

tend".

Example 2:

Input:

words = ["leetcode","win","loops","success"],

pref

= "code"

Output:

0

Explanation:

There are no strings that contain "code" as a prefix.

Constraints:

1 <= words.length <= 100

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

words[i]

and

pref

consist of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int prefixCount(String[] words, String pref) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function prefixCount(words: string[], pref: string): number {

};
```

C#:

```
public class Solution {
public int PrefixCount(string[] words, string pref) {

}
}
```

C:

```
int prefixCount(char** words, int wordsSize, char* pref) {

}
```

Go:

```
func prefixCount(words []string, pref string) int {

}
```

Kotlin:

```
class Solution {
fun prefixCount(words: Array<String>, pref: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int prefixCount(List<String> words, String pref) {

}

}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec prefix_count(Words :: [unicode:unicode_binary()], Pref ::
unicode:unicode_binary()) -> integer().
prefix_count(Words, Pref) ->
.
```

Racket:

```
(define/contract (prefix-count words pref)
(-> (listof string?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Counting Words With a Given Prefix
* Difficulty: Easy
* Tags: array, string, tree
```

```
*
* 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 prefixCount(vector<string>& words, string pref) {

}
};
```

Java Solution:

```
/**
* Problem: Counting Words With a Given Prefix
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 prefixCount(String[] words, String pref) {

}
}
```

Python3 Solution:

```
"""
Problem: Counting Words With a Given Prefix
Difficulty: Easy
Tags: array, string, tree

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 prefixCount(self, words: List[str], pref: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Counting Words With a Given Prefix
* Difficulty: Easy
* Tags: array, string, tree
*
* 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} pref
* @return {number}
*/
var prefixCount = function(words, pref) {

};
```

TypeScript Solution:

```
/**
* Problem: Counting Words With a Given Prefix
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 prefixCount(words: string[], pref: string): number {

};
```

C# Solution:

```
/*
* Problem: Counting Words With a Given Prefix
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 PrefixCount(string[] words, string pref) {

}
}
```

C Solution:

```
/*
* Problem: Counting Words With a Given Prefix
* Difficulty: Easy
* Tags: array, string, tree
*
* 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 prefixCount(char** words, int wordsSize, char* pref) {

}
```

Go Solution:

```
// Problem: Counting Words With a Given Prefix
// Difficulty: Easy
// Tags: array, string, tree
//
// 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 prefixCount(words []string, pref string) int {

}
```

Kotlin Solution:

```
class Solution {
fun prefixCount(words: Array<String>, pref: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Counting Words With a Given Prefix
// Difficulty: Easy
// Tags: array, string, tree
```

```
//
// 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 prefix_count(words: Vec<String>, pref: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int prefixCount(List<String> words, String pref) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec prefix_count(Words :: [unicode:unicode_binary()], Pref ::
unicode:unicode_binary()) -> integer().
prefix_count(Words, Pref) ->
.
```

Racket Solution:

```
(define/contract (prefix-count words pref)
(-> (listof string?) string? exact-integer?)
)
```