Problem 2255: Count Prefixes of a Given String

Problem Information

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

Problem Description

You are given a string array

words

and a string

s

, where

words[i]

and

s

comprise only of

lowercase English letters

.

Return

the

number of strings

in

words

that are a

prefix

of

s

.

A

prefix

of a string is a substring that occurs at the beginning of the string. A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

words = ["a","b","c","ab","bc","abc"], s = "abc"

Output:

3

Explanation:

The strings in words which are a prefix of s = "abc" are: "a", "ab", and "abc". Thus the number of strings in words which are a prefix of s is 3.

Example 2:

Input:

words = ["a","a"], s = "aa"

Output:

2

Explanation:

Both of the strings are a prefix of s. Note that the same string can occur multiple times in words, and it should be counted each time.

Constraints:

1 <= words.length <= 1000

1 <= words[i].length, s.length <= 10

words[i]

and

s

consist of lowercase English letters

only

.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int countPrefixes(String[] words, String s) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function countPrefixes(words: string[], s: string): number {

};
```

C#:

```
public class Solution {
public int CountPrefixes(string[] words, string s) {

}
}
```

C:

```
int countPrefixes(char** words, int wordsSize, char* s) {

}
```

Go:

```
func countPrefixes(words []string, s string) int {

}
```

Kotlin:

```
class Solution {
fun countPrefixes(words: Array<String>, s: String): Int {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int countPrefixes(List<String> words, String s) {

}
}
```

Scala:

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

}
```

```
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec count_prefixes(Words :: [unicode:unicode_binary()], S ::
unicode:unicode_binary()) -> integer().
count_prefixes(Words, S) ->
.
```

Racket:

```
(define/contract (count-prefixes words s)
(-> (listof string?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Prefixes of a Given String
* 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 countPrefixes(vector<string>& words, string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count Prefixes of a Given String
* 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 countPrefixes(String[] words, String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Prefixes of a Given String
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 countPrefixes(self, words: List[str], s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Prefixes of a Given String
* 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} s
* @return {number}
*/
var countPrefixes = function(words, s) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Prefixes of a Given String
* 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 countPrefixes(words: string[], s: string): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Count Prefixes of a Given String
* 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 CountPrefixes(string[] words, string s) {

}
}
```

C Solution:

```
/*
* Problem: Count Prefixes of a Given String
* 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 countPrefixes(char** words, int wordsSize, char* s) {

}
```

Go Solution:

```
// Problem: Count Prefixes of a Given String
// 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 countPrefixes(words []string, s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPrefixes(words: Array<String>, s: String): Int {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Prefixes of a Given String
// 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 count_prefixes(words: Vec<String>, s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int countPrefixes(List<String> words, String s) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec count_prefixes(Words :: [unicode:unicode_binary()], S ::
unicode:unicode_binary()) -> integer().
count_prefixes(Words, S) ->
.
```

Racket Solution:

```
(define/contract (count-prefixes words s)
(-> (listof string?) string? exact-integer?)
)
```