Problem 1967: Number of Strings That Appear as Substrings in Word

Problem Information

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

Problem Description

Given an array of strings

patterns

and a string

word

, return

the

number

of strings in

patterns

that exist as a

substring

in

word

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

patterns = ["a","abc","bc","d"], word = "abc"

Output:

3

Explanation:

- "a" appears as a substring in "

a

bc". - "abc" appears as a substring in "

abc

". - "bc" appears as a substring in "a

bc

". - "d" does not appear as a substring in "abc". 3 of the strings in patterns appear as a substring in word.

Example 2:

Input:

patterns = ["a","b","c"], word = "aaaaabbbbb"

Output:

2

Explanation:

- "a" appears as a substring in "a

a

aaabbbbb". - "b" appears as a substring in "aaaaabbbb

b

". - "c" does not appear as a substring in "aaaaabbbbb". 2 of the strings in patterns appear as a substring in word.

Example 3:

Input:

patterns = ["a","a","a"], word = "ab"

Output:

3

Explanation:

Each of the patterns appears as a substring in word "

a

b".

Constraints:

1 <= patterns.length <= 100

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

1 <= word.length <= 100

patterns[i]

and

word

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int numOfStrings(vector<string>& patterns, string word) {

}
};
```

Java:

```
class Solution {
public int numOfStrings(String[] patterns, String word) {

}
}
```

Python3:

```
class Solution:
def numOfStrings(self, patterns: List[str], word: str) -> int:
```

Python:

```
class Solution(object):
def numOfStrings(self, patterns, word):
"""
:type patterns: List[str]
:type word: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string[]} patterns
* @param {string} word
* @return {number}
*/
var numOfStrings = function(patterns, word) {

};
```

TypeScript:

```
function numOfStrings(patterns: string[], word: string): number {

};
```

C#:

```
public class Solution {
public int NumOfStrings(string[] patterns, string word) {

}
}
```

C:

```
int numOfStrings(char** patterns, int patternsSize, char* word) {

}
```

Go:

```
func numOfStrings(patterns []string, word string) int {
```

```
}
```

Kotlin:

```
class Solution {
fun numOfStrings(patterns: Array<String>, word: String): Int {

}
}
```

Swift:

```
class Solution {
func numOfStrings(_ patterns: [String], _ word: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn num_of_strings(patterns: Vec<String>, word: String) -> i32 {

}
}
```

Ruby:

```
# @param {String[]} patterns
# @param {String} word
# @return {Integer}
def num_of_strings(patterns, word)

end
```

PHP:

```
class Solution {

/**
* @param String[] $patterns
* @param String $word
```

```
* @return Integer
*/
function numOfStrings($patterns, $word) {

}
}
```

Dart:

```
class Solution {
int numOfStrings(List<String> patterns, String word) {

}
}
```

Scala:

```
object Solution {
def numOfStrings(patterns: Array[String], word: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec num_of_strings(patterns :: [String.t], word :: String.t) :: integer
def num_of_strings(patterns, word) do

end
end
```

Erlang:

```
-spec num_of_strings(Patterns :: [unicode:unicode_binary()], Word ::
unicode:unicode_binary()) -> integer().
num_of_strings(Patterns, Word) ->
.
```

Racket:

```
(define/contract (num-of-strings patterns word)
(-> (listof string?) string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Strings That Appear as Substrings in Word
* 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 numOfStrings(vector<string>& patterns, string word) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Strings That Appear as Substrings in Word
* 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 numOfStrings(String[] patterns, String word) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Number of Strings That Appear as Substrings in Word
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 numOfStrings(self, patterns: List[str], word: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numOfStrings(self, patterns, word):
"""
:type patterns: List[str]
:type word: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Strings That Appear as Substrings in Word
* 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[]} patterns
* @param {string} word
* @return {number}
*/
var numOfStrings = function(patterns, word) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Strings That Appear as Substrings in Word
* 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 numOfStrings(patterns: string[], word: string): number {

};
```

C# Solution:

```
/*
* Problem: Number of Strings That Appear as Substrings in Word
* 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 NumOfStrings(string[] patterns, string word) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Number of Strings That Appear as Substrings in Word
* 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 numOfStrings(char** patterns, int patternsSize, char* word) {

}
```

Go Solution:

```
// Problem: Number of Strings That Appear as Substrings in Word
// 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 numOfStrings(patterns []string, word string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numOfStrings(patterns: Array<String>, word: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func numOfStrings(_ patterns: [String], _ word: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Strings That Appear as Substrings in Word
// 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 num_of_strings(patterns: Vec<String>, word: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String[]} patterns
# @param {String} word
# @return {Integer}
def num_of_strings(patterns, word)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $patterns
* @param String $word
* @return Integer
*/
function numOfStrings($patterns, $word) {
```

```
}
}
```

Dart Solution:

```
class Solution {
int numOfStrings(List<String> patterns, String word) {

}
}
```

Scala Solution:

```
object Solution {
def numOfStrings(patterns: Array[String], word: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec num_of_strings(patterns :: [String.t], word :: String.t) :: integer
def num_of_strings(patterns, word) do

end
end
```

Erlang Solution:

```
-spec num_of_strings(Patterns :: [unicode:unicode_binary()], Word ::
unicode:unicode_binary()) -> integer().
num_of_strings(Patterns, Word) ->
.
```

Racket Solution:

```
(define/contract (num-of-strings patterns word)
(-> (listof string?) string? exact-integer?)
)
```