Problem 792: Number of Matching Subsequences

Problem Information

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

Problem Description

Given a string

s

and an array of strings

words

, return

the number of

words[i]

that is a subsequence of

s

.

A

subsequence

of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

For example,

"ace"

is a subsequence of

"abcde"

.

Example 1:

Input:

s = "abcde", words = ["a","bb","acd","ace"]

Output:

3

Explanation:

There are three strings in words that are a subsequence of s: "a", "acd", "ace".

Example 2:

Input:

s = "dsahjpjauf", words = ["ahjpjau","ja","ahbwzgqnuk","tnmlanowax"]

Output:

2

Constraints:

1 <= s.length <= 5 * 10

4

1 <= words.length <= 5000

1 <= words[i].length <= 50

s

and

words[i]

consist of only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

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

}
```

Go:

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

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String $s
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-matching-subseq s words)
(-> string? (listof string?) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Matching Subsequences
* Difficulty: Medium
* Tags: array, string, dp, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Number of Matching Subsequences
* Difficulty: Medium
* Tags: array, string, dp, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Number of Matching Subsequences
Difficulty: Medium
Tags: array, string, dp, hash, sort, search

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Matching Subsequences
* Difficulty: Medium
* Tags: array, string, dp, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Number of Matching Subsequences
* Difficulty: Medium
* Tags: array, string, dp, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

C# Solution:

```
/*
* Problem: Number of Matching Subsequences
* Difficulty: Medium
* Tags: array, string, dp, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

}
```

```
}
```

C Solution:

```
/*
* Problem: Number of Matching Subsequences
* Difficulty: Medium
* Tags: array, string, dp, hash, sort, search
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int numMatchingSubseq(char* s, char** words, int wordsSize) {

}
```

Go Solution:

```
// Problem: Number of Matching Subsequences
// Difficulty: Medium
// Tags: array, string, dp, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numMatchingSubseq(s string, words []string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Matching Subsequences
// Difficulty: Medium
// Tags: array, string, dp, hash, sort, search
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

```
}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-matching-subseq s words)
(-> string? (listof string?) exact-integer?)
)
```