Problem 2416: Sum of Prefix Scores of Strings

Problem Information

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

Problem Description

You are given an array

words

of size

n

consisting of

non-empty

strings.

We define the

score

of a string

term

as the

number

of strings

words[i]

such that

term

is a

prefix

of

words[i]

.

For example, if

words = ["a", "ab", "abc", "cab"]

, then the score of

"ab"

is

2

, since

"ab"

is a prefix of both

"ab"

and

"abc"

.

Return

an array

answer

of size

n

where

answer[i]

is the

sum

of scores of every

non-empty

prefix of

words[i]

.

Note

that a string is considered as a prefix of itself.

Example 1:

Input:

words = ["abc","ab","bc","b"]

Output:

[5,4,3,2]

Explanation:

The answer for each string is the following: - "abc" has 3 prefixes: "a", "ab", and "abc". - There are 2 strings with the prefix "a", 2 strings with the prefix "ab", and 1 string with the prefix "abc". The total is answer[0] = 2 + 2 + 1 = 5. - "ab" has 2 prefixes: "a" and "ab". - There are 2 strings with the prefix "a", and 2 strings with the prefix "ab". The total is answer[1] = 2 + 2 = 4. - "bc" has 2 prefixes: "b" and "bc". - There are 2 strings with the prefix "b", and 1 string with the prefix "bc". The total is answer[2] = 2 + 1 = 3. - "b" has 1 prefix: "b". - There are 2 strings with the prefix "b". The total is answer[3] = 2.

Example 2:

Input:

words = ["abcd"]

Output:

[4]

Explanation:

"abcd" has 4 prefixes: "a", "ab", "abc", and "abcd". Each prefix has a score of one, so the total is answer[0] = 1 + 1 + 1 + 1 = 4.

Constraints:

1 <= words.length <= 1000

1 <= words[i].length <= 1000

words[i]

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
vector<int> sumPrefixScores(vector<string>& words) {

}
};
```

Java:

```
class Solution {
public int[] sumPrefixScores(String[] words) {

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function sumPrefixScores(words: string[]): number[] {

};
```

C#:

```
public class Solution {
public int[] SumPrefixScores(string[] words) {

}
}
```

C:

```
/**
* Note: The returned array must be malloced, assume caller calls free().
*/
int* sumPrefixScores(char** words, int wordsSize, int* returnSize) {

}
```

Go:

```
func sumPrefixScores(words []string) []int {

}
```

Kotlin:

```
class Solution {
fun sumPrefixScores(words: Array<String>): IntArray {

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

```
# @param {String[]} words
# @return {Integer[]}
def sum_prefix_scores(words)

end
```

PHP:

```
class Solution {

/**
* @param String[] $words
* @return Integer[]
*/
function sumPrefixScores($words) {

}
}
```

Dart:

```
class Solution {
List<int> sumPrefixScores(List<String> words) {

}
}
```

Scala:

```
object Solution {
def sumPrefixScores(words: Array[String]): Array[Int] = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec sum_prefix_scores(Words :: [unicode:unicode_binary()]) -> [integer()].
sum_prefix_scores(Words) ->
.
```

Racket:

```
(define/contract (sum-prefix-scores words)
(-> (listof string?) (listof exact-integer?))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sum of Prefix Scores of Strings
* Difficulty: Hard
* Tags: array, string
*
```

```
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
vector<int> sumPrefixScores(vector<string>& words) {

}
};
```

Java Solution:

```
/**
* Problem: Sum of Prefix Scores of Strings
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[] sumPrefixScores(String[] words) {

}
}
```

Python3 Solution:

```
"""
Problem: Sum of Prefix Scores of Strings
Difficulty: Hard
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""
```

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Sum of Prefix Scores of Strings
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string[]} words
* @return {number[]}
*/
var sumPrefixScores = function(words) {

};
```

TypeScript Solution:

```
/**
* Problem: Sum of Prefix Scores of Strings
* Difficulty: Hard
* Tags: array, string
```

```
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function sumPrefixScores(words: string[]): number[] {

};
```

C# Solution:

```
/*
* Problem: Sum of Prefix Scores of Strings
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[] SumPrefixScores(string[] words) {

}
}
```

C Solution:

```
/*
* Problem: Sum of Prefix Scores of Strings
* Difficulty: Hard
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
```

```
* Note: The returned array must be malloced, assume caller calls free().
*/
int* sumPrefixScores(char** words, int wordsSize, int* returnSize) {

}
```

Go Solution:

```
// Problem: Sum of Prefix Scores of Strings
// Difficulty: Hard
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func sumPrefixScores(words []string) []int {

}
```

Kotlin Solution:

```
class Solution {
fun sumPrefixScores(words: Array<String>): IntArray {

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Sum of Prefix Scores of Strings
// Difficulty: Hard
// Tags: array, string
```

```
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn sum_prefix_scores(words: Vec<String>) -> Vec<i32> {

}
}
```

Ruby Solution:

```
# @param {String[]} words
# @return {Integer[]}
def sum_prefix_scores(words)

end
```

PHP Solution:

```
class Solution {

/**
* @param String[] $words
* @return Integer[]
*/
function sumPrefixScores($words) {

}
}
```

Dart Solution:

```
class Solution {
List<int> sumPrefixScores(List<String> words) {

}
}
```

Scala Solution:

```
object Solution {
def sumPrefixScores(words: Array[String]): Array[Int] = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec sum_prefix_scores(Words :: [unicode:unicode_binary()]) -> [integer()].
sum_prefix_scores(Words) ->
.
```

Racket Solution:

```
(define/contract (sum-prefix-scores words)
(-> (listof string?) (listof exact-integer?))
)
```