Problem 434: Number of Segments in a String

Problem Information

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

Problem Description

Given a string

s

, return

the number of segments in the string

.

A

segment

is defined to be a contiguous sequence of

non-space characters

.

Example 1:

Input:

s = "Hello, my name is John"

Output:

5

Explanation:

The five segments are ["Hello,", "my", "name", "is", "John"]

Example 2:

Input:

s = "Hello"

Output:

1

Constraints:

0 <= s.length <= 300

s

consists of lowercase and uppercase English letters, digits, or one of the following characters

"!@#$%^&*()_+-=',.:"

.

The only space character in

s

is

' '

.

Code Snippets

C++:

```
class Solution {
public:
int countSegments(string s) {

}
};
```

Java:

```
class Solution {
public int countSegments(String s) {

}
}
```

Python3:

```
class Solution:
def countSegments(self, s: str) -> int:
```

Python:

```
class Solution(object):
def countSegments(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {number}
*/
var countSegments = function(s) {
```

```
};
```

TypeScript:

```
function countSegments(s: string): number {

};
```

C#:

```
public class Solution {
public int CountSegments(string s) {

}
}
```

C:

```
int countSegments(char* s) {

}
```

Go:

```
func countSegments(s string) int {

}
```

Kotlin:

```
class Solution {
fun countSegments(s: String): Int {

}
}
```

Swift:

```
class Solution {
func countSegments(_ s: String) -> Int {

}
```

```
}
```

Rust:

```
impl Solution {
pub fn count_segments(s: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @return {Integer}
def count_segments(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function countSegments($s) {

}
}
```

Dart:

```
class Solution {
int countSegments(String s) {

}
}
```

Scala:

```
object Solution {
def countSegments(s: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_segments(s :: String.t) :: integer
def count_segments(s) do

end
end
```

Erlang:

```
-spec count_segments(S :: unicode:unicode_binary()) -> integer().
count_segments(S) ->
.
```

Racket:

```
(define/contract (count-segments s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Segments in a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int countSegments(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Segments in a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countSegments(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Segments in a String
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def countSegments(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countSegments(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Segments in a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} s
* @return {number}
*/
var countSegments = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Segments in a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countSegments(s: string): number {
```

```
};
```

C# Solution:

```
/*
* Problem: Number of Segments in a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int CountSegments(string s) {

}
}
```

C Solution:

```
/*
* Problem: Number of Segments in a String
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int countSegments(char* s) {

}
```

Go Solution:

```
// Problem: Number of Segments in a String
// Difficulty: Easy
```

```
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countSegments(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countSegments(s: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func countSegments(_ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Segments in a String
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn count_segments(s: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer}
def count_segments(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function countSegments($s) {

}
}
```

Dart Solution:

```
class Solution {
int countSegments(String s) {

}
}
```

Scala Solution:

```
object Solution {
def countSegments(s: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_segments(s :: String.t) :: integer
def count_segments(s) do
```

```
end
end
```

Erlang Solution:

```
-spec count_segments(S :: unicode:unicode_binary()) -> integer().
count_segments(S) ->
.
```

Racket Solution:

```
(define/contract (count-segments s)
(-> string? exact-integer?)
)
```