Problem 1525: Number of Good Ways to Split a String

Problem Information

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

Problem Description

You are given a string

s

.

A split is called

good

if you can split

s

into two non-empty strings

s

left

and

s

right

where their concatenation is equal to

s

(i.e.,

s

left

+ s

right

= s

) and the number of distinct letters in

s

left

and

s

right

is the same.

Return

the number of

good splits

you can make in

s

.

Example 1:

Input:

s = "aacaba"

Output:

2

Explanation:

There are 5 ways to split

"aacaba"

and 2 of them are good. ("a", "acaba") Left string and right string contains 1 and 3 different letters respectively. ("aa", "caba") Left string and right string contains 1 and 3 different letters respectively. ("aac", "aba") Left string and right string contains 2 and 2 different letters respectively (good split). ("aaca", "ba") Left string and right string contains 2 and 2 different letters respectively (good split). ("aacab", "a") Left string and right string contains 3 and 1 different letters respectively.

Example 2:

Input:

s = "abcd"

Output:

1

Explanation:

Split the string as follows ("ab", "cd").

Constraints:

1 <= s.length <= 10

5

s

consists of only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def numSplits(self, s):
```

```
"""
:type s: str
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int numSplits(char* s) {

}
```

Go:

```
func numSplits(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (num-splits s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Good Ways to Split a String
* Difficulty: Medium
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int numSplits(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Good Ways to Split a String
* Difficulty: Medium
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int numSplits(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Good Ways to Split a String
Difficulty: Medium
Tags: string, dp, hash
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Good Ways to Split a String
* Difficulty: Medium
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Number of Good Ways to Split a String
* Difficulty: Medium
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function numSplits(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Number of Good Ways to Split a String
* Difficulty: Medium
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* 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 NumSplits(string s) {

}
}
```

C Solution:

```
/*
* Problem: Number of Good Ways to Split a String
* Difficulty: Medium
* Tags: string, dp, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
```

```
*/

int numSplits(char* s) {

}
```

Go Solution:

```
// Problem: Number of Good Ways to Split a String
// Difficulty: Medium
// Tags: string, dp, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func numSplits(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Good Ways to Split a String
// Difficulty: Medium
// Tags: string, dp, hash
```

```
//
// Approach: String manipulation with hash map or two pointers
// 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_splits(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (num-splits s)
(-> string? exact-integer?)
)
```