Problem 1221: Split a String in Balanced Strings

Problem Information

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

Problem Description

Balanced

strings are those that have an equal quantity of

'L'

and

'R'

characters.

Given a

balanced

string

s

, split it into some number of substrings such that:

Each substring is balanced.

Return

the

maximum

number of balanced strings you can obtain.

Example 1:

Input:

s = "RLRRLLRLRL"

Output:

4

Explanation:

s can be split into "RL", "RRLL", "RL", "RL", each substring contains same number of 'L' and 'R'.

Example 2:

Input:

s = "RLRRRLLRLL"

Output:

2

Explanation:

s can be split into "RL", "RRRLLRLL", each substring contains same number of 'L' and 'R'. Note that s cannot be split into "RL", "RR", "RL", "LR", "LL", because the 2

nd

and 5

th

substrings are not balanced.

Example 3:

Input:

s = "LLLLRRRR"

Output:

1

Explanation:

s can be split into "LLLLRRRR".

Constraints:

2 <= s.length <= 1000

s[i]

is either

'L'

or

'R'

.

s

is a

balanced

string.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

```
/**
* @param {string} s
```

```
 * @return {number}
 */
var balancedStringSplit = function(s) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int balancedStringSplit(char* s) {

}
```

Go:

```
func balancedStringSplit(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (balanced-string-split s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Split a String in Balanced Strings
* Difficulty: Easy
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/
```

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

}
};
```

Java Solution:

```
/**
* Problem: Split a String in Balanced Strings
* Difficulty: Easy
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int balancedStringSplit(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Split a String in Balanced Strings
Difficulty: Easy
Tags: string, tree, greedy

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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

```
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Split a String in Balanced Strings
* Difficulty: Easy
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Split a String in Balanced Strings
* Difficulty: Easy
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

function balancedStringSplit(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Split a String in Balanced Strings
* Difficulty: Easy
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* 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 BalancedStringSplit(string s) {

}
}
```

C Solution:

```
/*
* Problem: Split a String in Balanced Strings
* Difficulty: Easy
* Tags: string, tree, greedy
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int balancedStringSplit(char* s) {

}
```

Go Solution:

```
// Problem: Split a String in Balanced Strings
// Difficulty: Easy
// Tags: string, tree, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func balancedStringSplit(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Split a String in Balanced Strings
// Difficulty: Easy
// Tags: string, tree, greedy
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
```

```
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (balanced-string-split s)
(-> string? exact-integer?)
)
```