Problem 1593: Split a String Into the Max Number of Unique Substrings

Problem Information

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

Problem Description

Given a string

s

,

return

the maximum number of unique substrings that the given string can be split into

.

You can split string

s

into any list of

non-empty substrings

, where the concatenation of the substrings forms the original string. However, you must split the substrings such that all of them are

unique

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

s = "ababccc"

Output:

5

Explanation

: One way to split maximally is ['a', 'b', 'ab', 'c', 'cc']. Splitting like ['a', 'b', 'a', 'b', 'c', 'cc'] is not valid as you have 'a' and 'b' multiple times.

Example 2:

Input:

s = "aba"

Output:

2

Explanation

: One way to split maximally is ['a', 'ba'].

Example 3:

Input:

s = "aa"

Output:

1

Explanation

: It is impossible to split the string any further.

Constraints:

1 <= s.length <= 16

s

contains only lower case English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maxUniqueSplit(char* s) {

}
```

Go:

```
func maxUniqueSplit(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
```

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Split a String Into the Max Number of Unique Substrings
* Difficulty: Medium
* Tags: string, tree, hash
*
* 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 maxUniqueSplit(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Split a String Into the Max Number of Unique Substrings
* Difficulty: Medium
* Tags: string, tree, hash
*
* 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 maxUniqueSplit(String s) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Split a String Into the Max Number of Unique Substrings
Difficulty: Medium
Tags: string, tree, hash

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 maxUniqueSplit(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Split a String Into the Max Number of Unique Substrings
* Difficulty: Medium
* Tags: string, tree, hash
*
* 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 maxUniqueSplit = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Split a String Into the Max Number of Unique Substrings
* Difficulty: Medium
* Tags: string, tree, hash
*
* 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 maxUniqueSplit(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Split a String Into the Max Number of Unique Substrings
* Difficulty: Medium
* Tags: string, tree, hash
*
* 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 MaxUniqueSplit(string s) {

}
}
```

C Solution:

```
/*
* Problem: Split a String Into the Max Number of Unique Substrings
* Difficulty: Medium
* Tags: string, tree, hash
*
* 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 maxUniqueSplit(char* s) {

}
```

Go Solution:

```
// Problem: Split a String Into the Max Number of Unique Substrings
// Difficulty: Medium
// Tags: string, tree, hash
//
// 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 maxUniqueSplit(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Split a String Into the Max Number of Unique Substrings
// Difficulty: Medium
// Tags: string, tree, hash
//
// 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 max_unique_split(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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