Problem 2405: Optimal Partition of String

Problem Information

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

Problem Description

Given a string

s

, partition the string into one or more

substrings

such that the characters in each substring are

unique

. That is, no letter appears in a single substring more than

once

.

Return

the

minimum

number of substrings in such a partition.

Note that each character should belong to exactly one substring in a partition.

Example 1:

Input:

s = "abacaba"

Output:

4

Explanation:

Two possible partitions are ("a","ba","cab","a") and ("ab","a","ca","ba"). It can be shown that 4 is the minimum number of substrings needed.

Example 2:

Input:

s = "ssssss"

Output:

6

Explanation:

The only valid partition is ("s","s","s","s","s","s").

Constraints:

1 <= s.length <= 10

5

s

consists of only English lowercase letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

```
var partitionString = function(s) {

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int partitionString(char* s) {

}
```

Go:

```
func partitionString(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Optimal Partition of String
* Difficulty: Medium
* Tags: string, tree, greedy, 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 partitionString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Optimal Partition of String
* Difficulty: Medium
* Tags: string, tree, greedy, 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 partitionString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Optimal Partition of String
Difficulty: Medium
Tags: string, tree, greedy, 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 partitionString(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Optimal Partition of String
* Difficulty: Medium
* Tags: string, tree, greedy, 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 partitionString = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Optimal Partition of String
* Difficulty: Medium
* Tags: string, tree, greedy, 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 partitionString(s: string): number {
};
```

C# Solution:

```
/*
* Problem: Optimal Partition of String
* Difficulty: Medium
* Tags: string, tree, greedy, 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 PartitionString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Optimal Partition of String
* Difficulty: Medium
* Tags: string, tree, greedy, 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 partitionString(char* s) {

}
```

Go Solution:

```
// Problem: Optimal Partition of String
// Difficulty: Medium
// Tags: string, tree, greedy, 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 partitionString(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Optimal Partition of String
// Difficulty: Medium
// Tags: string, tree, greedy, 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 partition_string(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

```
end
end
```

Erlang Solution:

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

Racket Solution:

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