Problem 3144: Minimum Substring Partition of Equal Character Frequency

Problem Information

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

Problem Description

Given a string

s

, you need to partition it into one or more

balanced

substrings

. For example, if

s == "ababcc"

then

("abab", "c", "c")

,

("ab", "abc", "c")

, and

("ababcc")

are all valid partitions, but

("a",

"bab"

, "cc")

,

(

"aba"

, "bc", "c")

, and

("ab",

"abcc"

)

are not. The unbalanced substrings are bolded.

Return the

minimum

number of substrings that you can partition

s

into.

Note:

A

balanced

string is a string where each character in the string occurs the same number of times.

Example 1:

Input:

s = "fabccddg"

Output:

3

Explanation:

We can partition the string

s

into 3 substrings in one of the following ways:

("fab, "ccdd", "g")

, or

("fabc", "cd", "dg")

.

Example 2:

Input:

s = "abababaccddb"

Output:

2

Explanation:

We can partition the string

s

into 2 substrings like so:

("abab", "abaccddb")

.

Constraints:

1 <= s.length <= 1000

s

consists only of English lowercase letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minimumSubstringsInPartition(char* s) {

}
```

Go:

```
func minimumSubstringsInPartition(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Substring Partition of Equal Character Frequency
* Difficulty: Medium
* Tags: string, tree, 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 minimumSubstringsInPartition(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Substring Partition of Equal Character Frequency
* Difficulty: Medium
* Tags: string, tree, 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 minimumSubstringsInPartition(String s) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Minimum Substring Partition of Equal Character Frequency
Difficulty: Medium
Tags: string, tree, 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 minimumSubstringsInPartition(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Substring Partition of Equal Character Frequency
* Difficulty: Medium
* Tags: string, tree, 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 minimumSubstringsInPartition = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Substring Partition of Equal Character Frequency
* Difficulty: Medium
* Tags: string, tree, 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 minimumSubstringsInPartition(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Substring Partition of Equal Character Frequency
* Difficulty: Medium
* Tags: string, tree, 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 MinimumSubstringsInPartition(string s) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Minimum Substring Partition of Equal Character Frequency
* Difficulty: Medium
* Tags: string, tree, 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 minimumSubstringsInPartition(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Substring Partition of Equal Character Frequency
// Difficulty: Medium
// Tags: string, tree, 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 minimumSubstringsInPartition(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Substring Partition of Equal Character Frequency
// Difficulty: Medium
// Tags: string, tree, 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 minimum_substrings_in_partition(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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