Problem 2767: Partition String Into Minimum Beautiful Substrings

Problem Information

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

Problem Description

Given a binary string

s

, partition the string into one or more

substrings

such that each substring is

beautiful

.

A string is

beautiful

if:

It doesn't contain leading zeros.

It's the

binary

representation of a number that is a power of

5

.

Return

the

minimum

number of substrings in such partition.

If it is impossible to partition the string

s

into beautiful substrings, return

-1

.

A

substring

is a contiguous sequence of characters in a string.

Example 1:

Input:

s = "1011"

Output:

2

Explanation:

We can paritition the given string into ["101", "1"]. - The string "101" does not contain leading zeros and is the binary representation of integer 5

1

= 5. - The string "1" does not contain leading zeros and is the binary representation of integer 5

0

= 1. It can be shown that 2 is the minimum number of beautiful substrings that s can be partitioned into.

Example 2:

Input:

s = "111"

Output:

3

Explanation:

We can paritition the given string into ["1", "1", "1"]. - The string "1" does not contain leading zeros and is the binary representation of integer 5

0

= 1. It can be shown that 3 is the minimum number of beautiful substrings that s can be partitioned into.

Example 3:

Input:

s = "0"

Output:

-1

Explanation:

We can not partition the given string into beautiful substrings.

Constraints:

1 <= s.length <= 15

s[i]

is either

'0'

or

'1'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

```
public class Solution {
public int MinimumBeautifulSubstrings(string s) {
}
}
```

C:

```
int minimumBeautifulSubstrings(char* s) {

}
```

Go:

```
func minimumBeautifulSubstrings(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Partition String Into Minimum Beautiful Substrings
* 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 minimumBeautifulSubstrings(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Partition String Into Minimum Beautiful Substrings
* 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 minimumBeautifulSubstrings(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition String Into Minimum Beautiful Substrings
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 minimumBeautifulSubstrings(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition String Into Minimum Beautiful Substrings
* 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 minimumBeautifulSubstrings = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition String Into Minimum Beautiful Substrings
* 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 minimumBeautifulSubstrings(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Partition String Into Minimum Beautiful Substrings
* 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 MinimumBeautifulSubstrings(string s) {

}
}
```

C Solution:

```
/*
* Problem: Partition String Into Minimum Beautiful Substrings
* 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 minimumBeautifulSubstrings(char* s) {

}
```

Go Solution:

```
// Problem: Partition String Into Minimum Beautiful Substrings
// 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 minimumBeautifulSubstrings(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Partition String Into Minimum Beautiful Substrings
// 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_beautiful_substrings(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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