Problem 2522: Partition String Into Substrings With Values at Most K

Problem Information

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

Problem Description

You are given a string

s

consisting of digits from

1

to

9

and an integer

k

.

A partition of a string

s

is called

good

if:

Each digit of

s

is part of

exactly

one substring.

The value of each substring is less than or equal to

k

.

Return

the

minimum

number of substrings in a

good

partition of

s

. If no

good

partition of

s

exists, return

-1

.

Note

that:

The

value

of a string is its result when interpreted as an integer. For example, the value of

"123"

is

123

and the value of

"1"

is

1

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

s = "165462", k = 60

Output:

4

Explanation:

We can partition the string into substrings "16", "54", "6", and "2". Each substring has a value less than or equal to k = 60. It can be shown that we cannot partition the string into less than 4 substrings.

Example 2:

Input:

s = "238182", k = 5

Output:

-1

Explanation:

There is no good partition for this string.

Constraints:

1 <= s.length <= 10

5

s[i]

is a digit from

'1'

to

'9'

.

1 <= k <= 10

9

Code Snippets

C++:

```
class Solution {
public:
int minimumPartition(string s, int k) {

}
};
```

Java:

```
class Solution {
public int minimumPartition(String s, int k) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def minimumPartition(self, s, k):
```

```
"""
:type s: str
:type k: int
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function minimumPartition(s: string, k: number): number {

};
```

C#:

```
public class Solution {
public int MinimumPartition(string s, int k) {

}
}
```

C:

```
int minimumPartition(char* s, int k) {

}
```

Go:

```
func minimumPartition(s string, k int) int {

}
```

Kotlin:

```
class Solution {
fun minimumPartition(s: String, k: Int): Int {

}
}
```

Swift:

```
class Solution {
func minimumPartition(_ s: String, _ k: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn minimum_partition(s: String, k: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def minimum_partition(s, k)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function minimumPartition($s, $k) {
```

```
}
}
```

Dart:

```
class Solution {
int minimumPartition(String s, int k) {

}
}
```

Scala:

```
object Solution {
def minimumPartition(s: String, k: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec minimum_partition(s :: String.t, k :: integer) :: integer
def minimum_partition(s, k) do

end
end
```

Erlang:

```
-spec minimum_partition(S :: unicode:unicode_binary(), K :: integer()) ->
integer().
minimum_partition(S, K) ->
.
```

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Partition String Into Substrings With Values at Most K
* Difficulty: Medium
* Tags: string, tree, dp, greedy
*
* 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 minimumPartition(string s, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Partition String Into Substrings With Values at Most K
* Difficulty: Medium
* Tags: string, tree, dp, greedy
*
* 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 minimumPartition(String s, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Partition String Into Substrings With Values at Most K
Difficulty: Medium
Tags: string, tree, dp, greedy

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Partition String Into Substrings With Values at Most K
* Difficulty: Medium
* Tags: string, tree, dp, greedy
*
* 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
* @param {number} k
* @return {number}
*/
```

```
var minimumPartition = function(s, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Partition String Into Substrings With Values at Most K
* Difficulty: Medium
* Tags: string, tree, dp, greedy
*
* 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 minimumPartition(s: string, k: number): number {

};
```

C# Solution:

```
/*
* Problem: Partition String Into Substrings With Values at Most K
* Difficulty: Medium
* Tags: string, tree, dp, greedy
*
* 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 MinimumPartition(string s, int k) {

}
}
```

C Solution:

```
/*
* Problem: Partition String Into Substrings With Values at Most K
* Difficulty: Medium
* Tags: string, tree, dp, greedy
*
* 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 minimumPartition(char* s, int k) {

}
```

Go Solution:

```
// Problem: Partition String Into Substrings With Values at Most K
// Difficulty: Medium
// Tags: string, tree, dp, greedy
//
// 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 minimumPartition(s string, k int) int {

}
```

Kotlin Solution:

```
class Solution {
fun minimumPartition(s: String, k: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func minimumPartition(_ s: String, _ k: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Partition String Into Substrings With Values at Most K
// Difficulty: Medium
// Tags: string, tree, dp, greedy
//
// 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_partition(s: String, k: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def minimum_partition(s, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function minimumPartition($s, $k) {

}
}
```

Dart Solution:

```
class Solution {
int minimumPartition(String s, int k) {

}
}
```

Scala Solution:

```
object Solution {
def minimumPartition(s: String, k: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec minimum_partition(s :: String.t, k :: integer) :: integer
def minimum_partition(s, k) do

end
end
```

Erlang Solution:

```
-spec minimum_partition(S :: unicode:unicode_binary(), K :: integer()) ->
integer().
minimum_partition(S, K) ->
.
```

Racket Solution:

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