Problem 2478: Number of Beautiful Partitions

Problem Information

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

Problem Description

You are given a string

s

that consists of the digits

'1'

to

'9'

and two integers

k

and

minLength

.

A partition of

s

is called

beautiful

if:

s

is partitioned into

k

non-intersecting substrings.

Each substring has a length of

at least

minLength

.

Each substring starts with a

prime

digit and ends with a

non-prime

digit. Prime digits are

'2'

,

'3'

,

'5'

, and

'7'

, and the rest of the digits are non-prime.

Return

the number of

beautiful

partitions of

s

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

s = "23542185131", k = 3, minLength = 2

Output:

3

Explanation:

There exists three ways to create a beautiful partition: "2354 | 218 | 5131" "2354 | 21851 | 31" "2354218 | 51 | 31"

Example 2:

Input:

s = "23542185131", k = 3, minLength = 3

Output:

1

Explanation:

There exists one way to create a beautiful partition: "2354 | 218 | 5131".

Example 3:

Input:

s = "3312958", k = 3, minLength = 1

Output:

1

Explanation:

There exists one way to create a beautiful partition: "331 | 29 | 58".

Constraints:

1 <= k, minLength <= s.length <= 1000

s

consists of the digits

'1'

to

'9'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function beautifulPartitions(s: string, k: number, minLength: number): number
{

};
```

C#:

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

}
}
```

C:

```
int beautifulPartitions(char* s, int k, int minLength) {

}
```

Go:

```
func beautifulPartitions(s string, k int, minLength int) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

```
class Solution {
int beautifulPartitions(String s, int k, int minLength) {

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (beautiful-partitions s k minLength)
(-> string? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Beautiful Partitions
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int beautifulPartitions(string s, int k, int minLength) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Beautiful Partitions
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int beautifulPartitions(String s, int k, int minLength) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Beautiful Partitions
Difficulty: Hard
Tags: array, string, tree, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def beautifulPartitions(self, s: str, k: int, minLength: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Number of Beautiful Partitions
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* 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
* @param {number} minLength
* @return {number}
*/
var beautifulPartitions = function(s, k, minLength) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Beautiful Partitions
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function beautifulPartitions(s: string, k: number, minLength: number): number
{

};
```

C# Solution:

```
/*
* Problem: Number of Beautiful Partitions
* Difficulty: Hard
* Tags: array, string, tree, dp
```

```
*
* Approach: Use two pointers or sliding window technique
* 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 BeautifulPartitions(string s, int k, int minLength) {

}
}
```

C Solution:

```
/*
* Problem: Number of Beautiful Partitions
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int beautifulPartitions(char* s, int k, int minLength) {

}
```

Go Solution:

```
// Problem: Number of Beautiful Partitions
// Difficulty: Hard
// Tags: array, string, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func beautifulPartitions(s string, k int, minLength int) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Number of Beautiful Partitions
// Difficulty: Hard
// Tags: array, string, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn beautiful_partitions(s: String, k: i32, min_length: i32) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int beautifulPartitions(String s, int k, int minLength) {

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (beautiful-partitions s k minLength)
(-> string? exact-integer? exact-integer? exact-integer?)
)
```