Problem 1977: Number of Ways to Separate Numbers

Problem Information

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

Problem Description

You wrote down many

positive

integers in a string called

num

. However, you realized that you forgot to add commas to seperate the different numbers. You remember that the list of integers was

non-decreasing

and that

no

integer had leading zeros.

Return

the

number of possible lists of integers

that you could have written down to get the string

num

. Since the answer may be large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

num = "327"

Output:

2

Explanation:

You could have written down the numbers: 3, 27 327

Example 2:

Input:

num = "094"

Output:

0

Explanation:

No numbers can have leading zeros and all numbers must be positive.

Example 3:

Input:

num = "0"

Output:

0

Explanation:

No numbers can have leading zeros and all numbers must be positive.

Constraints:

1 <= num.length <= 3500

num

consists of digits

'0'

through

'9'

.

Code Snippets

C++:

```
class Solution {
public:
int numberOfCombinations(string num) {

}
};
```

Java:

```
class Solution {
public int numberOfCombinations(String num) {

}
}
```

Python3:

```
class Solution:
def numberOfCombinations(self, num: str) -> int:
```

Python:

```
class Solution(object):
def numberOfCombinations(self, num):
"""
:type num: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} num
* @return {number}
*/
var numberOfCombinations = function(num) {

};
```

TypeScript:

```
function numberOfCombinations(num: string): number {
```

```
};
```

C#:

```
public class Solution {
public int NumberOfCombinations(string num) {

}
}
```

C:

```
int numberOfCombinations(char* num) {

}
```

Go:

```
func numberOfCombinations(num string) int {

}
```

Kotlin:

```
class Solution {
fun numberOfCombinations(num: String): Int {

}
}
```

Swift:

```
class Solution {
func numberOfCombinations(_ num: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn number_of_combinations(num: String) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {String} num
# @return {Integer}
def number_of_combinations(num)

end
```

PHP:

```
class Solution {

/**
* @param String $num
* @return Integer
*/
function numberOfCombinations($num) {

}
}
```

Dart:

```
class Solution {
int numberOfCombinations(String num) {

}
}
```

Scala:

```
object Solution {
def numberOfCombinations(num: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec number_of_combinations(num :: String.t) :: integer
def number_of_combinations(num) do

end
end
```

Erlang:

```
-spec number_of_combinations(Num :: unicode:unicode_binary()) -> integer().
number_of_combinations(Num) ->
.
```

Racket:

```
(define/contract (number-of-combinations num)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Separate Numbers
* Difficulty: Hard
* Tags: array, string, 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 numberOfCombinations(string num) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Separate Numbers
* Difficulty: Hard
* Tags: array, string, 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 numberOfCombinations(String num) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Separate Numbers
Difficulty: Hard
Tags: array, string, 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 numberOfCombinations(self, num: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def numberOfCombinations(self, num):
"""
:type num: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Separate Numbers
* Difficulty: Hard
* Tags: array, string, 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} num
* @return {number}
*/
var numberOfCombinations = function(num) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Separate Numbers
* Difficulty: Hard
* Tags: array, string, 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 numberOfCombinations(num: string): number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Separate Numbers
* Difficulty: Hard
* Tags: array, string, 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 NumberOfCombinations(string num) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Separate Numbers
* Difficulty: Hard
* Tags: array, string, 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 numberOfCombinations(char* num) {

}
```

Go Solution:

```
// Problem: Number of Ways to Separate Numbers
// Difficulty: Hard
// Tags: array, string, 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 numberOfCombinations(num string) int {

}
```

Kotlin Solution:

```
class Solution {
fun numberOfCombinations(num: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func numberOfCombinations(_ num: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Separate Numbers
// Difficulty: Hard
// Tags: array, string, 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 number_of_combinations(num: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} num
# @return {Integer}
def number_of_combinations(num)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $num
* @return Integer
*/
function numberOfCombinations($num) {

}
}
```

Dart Solution:

```
class Solution {
int numberOfCombinations(String num) {

}
}
```

Scala Solution:

```
object Solution {
def numberOfCombinations(num: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec number_of_combinations(num :: String.t) :: integer
def number_of_combinations(num) do

end
end
```

Erlang Solution:

```
-spec number_of_combinations(Num :: unicode:unicode_binary()) -> integer().
number_of_combinations(Num) ->
.
```

Racket Solution:

```
(define/contract (number-of-combinations num)
(-> string? exact-integer?)
)
```