Problem 3519: Count Numbers with Non-Decreasing Digits

Problem Information

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

Problem Description

You are given two integers,

l

and

r

, represented as strings, and an integer

b

. Return the count of integers in the inclusive range

[l, r]

whose digits are in

non-decreasing

order when represented in base

b

.

An integer is considered to have

non-decreasing

digits if, when read from left to right (from the most significant digit to the least significant digit), each digit is greater than or equal to the previous one.

Since the answer may be too large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

l = "23", r = "28", b = 8

Output:

3

Explanation:

The numbers from 23 to 28 in base 8 are: 27, 30, 31, 32, 33, and 34.

Out of these, 27, 33, and 34 have non-decreasing digits. Hence, the output is 3.

Example 2:

Input:

l = "2", r = "7", b = 2

Output:

2

Explanation:

The numbers from 2 to 7 in base 2 are: 10, 11, 100, 101, 110, and 111.

Out of these, 11 and 111 have non-decreasing digits. Hence, the output is 2.

Constraints:

1 <= l.length <= r.length <= 100

2 <= b <= 10

l

and

r

consist only of digits.

The value represented by

l

is less than or equal to the value represented by

r

.

l

and

r

do not contain leading zeros.

Code Snippets

C++:

```
class Solution {
public:
int countNumbers(string l, string r, int b) {

}
};
```

Java:

```
class Solution {
public int countNumbers(String l, String r, int b) {

}
}
```

Python3:

```
class Solution:
def countNumbers(self, l: str, r: str, b: int) -> int:
```

Python:

```
class Solution(object):
def countNumbers(self, l, r, b):
"""
:type l: str
:type r: str
:type b: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} l
* @param {string} r
* @param {number} b
* @return {number}
*/
var countNumbers = function(l, r, b) {

};
```

TypeScript:

```
function countNumbers(l: string, r: string, b: number): number {

};
```

C#:

```
public class Solution {
public int CountNumbers(string l, string r, int b) {

}
}
```

C:

```
int countNumbers(char* l, char* r, int b) {

}
```

Go:

```
func countNumbers(l string, r string, b int) int {

}
```

Kotlin:

```
class Solution {
fun countNumbers(l: String, r: String, b: Int): Int {
```

```
}
}
```

Swift:

```
class Solution {
func countNumbers(_ l: String, _ r: String, _ b: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_numbers(l: String, r: String, b: i32) -> i32 {

}
}
```

Ruby:

```
# @param {String} l
# @param {String} r
# @param {Integer} b
# @return {Integer}
def count_numbers(l, r, b)

end
```

PHP:

```
class Solution {

/**
* @param String $l
* @param String $r
* @param Integer $b
* @return Integer
*/
function countNumbers($l, $r, $b) {

}
```

```
}
```

Dart:

```
class Solution {
int countNumbers(String l, String r, int b) {

}
}
```

Scala:

```
object Solution {
def countNumbers(l: String, r: String, b: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_numbers(l :: String.t, r :: String.t, b :: integer) :: integer
def count_numbers(l, r, b) do

end
end
```

Erlang:

```
-spec count_numbers(L :: unicode:unicode_binary(), R ::
unicode:unicode_binary(), B :: integer()) -> integer().
count_numbers(L, R, B) ->
.
```

Racket:

```
(define/contract (count-numbers l r b)
(-> string? string? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Numbers with Non-Decreasing Digits
* Difficulty: Hard
* Tags: string, dp, math
*
* 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 countNumbers(string l, string r, int b) {

}
};
```

Java Solution:

```
/**
* Problem: Count Numbers with Non-Decreasing Digits
* Difficulty: Hard
* Tags: string, dp, math
*
* 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 countNumbers(String l, String r, int b) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Numbers with Non-Decreasing Digits
```

```
Difficulty: Hard
Tags: string, dp, math

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 countNumbers(self, l: str, r: str, b: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countNumbers(self, l, r, b):
"""
:type l: str
:type r: str
:type b: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Numbers with Non-Decreasing Digits
* Difficulty: Hard
* Tags: string, dp, math
*
* 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} l
* @param {string} r
* @param {number} b
* @return {number}
```

```
*/
var countNumbers = function(l, r, b) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Numbers with Non-Decreasing Digits
* Difficulty: Hard
* Tags: string, dp, math
*
* 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 countNumbers(l: string, r: string, b: number): number {

};
```

C# Solution:

```
/*
* Problem: Count Numbers with Non-Decreasing Digits
* Difficulty: Hard
* Tags: string, dp, math
*
* 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 CountNumbers(string l, string r, int b) {

}
}
```

C Solution:

```c
/*
* Problem: Count Numbers with Non-Decreasing Digits
* Difficulty: Hard
* Tags: string, dp, math
*
* 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 countNumbers(char* l, char* r, int b) {

}
```

Go Solution:

```go
// Problem: Count Numbers with Non-Decreasing Digits
// Difficulty: Hard
// Tags: string, dp, math
//
// 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 countNumbers(l string, r string, b int) int {

}
```

Kotlin Solution:

```kotlin
class Solution {
fun countNumbers(l: String, r: String, b: Int): Int {

}
}
```

Swift Solution:

```swift
class Solution {
func countNumbers(_ l: String, _ r: String, _ b: Int) -> Int {

}
```

```
}
```

Rust Solution:

```
// Problem: Count Numbers with Non-Decreasing Digits
// Difficulty: Hard
// Tags: string, dp, math
//
// 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 count_numbers(l: String, r: String, b: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} l
# @param {String} r
# @param {Integer} b
# @return {Integer}
def count_numbers(l, r, b)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $l
* @param String $r
* @param Integer $b
* @return Integer
*/
function countNumbers($l, $r, $b) {

}
```

```
}
```

Dart Solution:

```
class Solution {
int countNumbers(String l, String r, int b) {

}
}
```

Scala Solution:

```
object Solution {
def countNumbers(l: String, r: String, b: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_numbers(l :: String.t, r :: String.t, b :: integer) :: integer
def count_numbers(l, r, b) do

end
end
```

Erlang Solution:

```
-spec count_numbers(L :: unicode:unicode_binary(), R ::
unicode:unicode_binary(), B :: integer()) -> integer().
count_numbers(L, R, B) ->
.
```

Racket Solution:

```
(define/contract (count-numbers l r b)
(-> string? string? exact-integer? exact-integer?)
)
```