Problem 2466: Count Ways To Build Good Strings

Problem Information

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

Problem Description

Given the integers

zero

,

one

,

low

, and

high

, we can construct a string by starting with an empty string, and then at each step perform either of the following:

Append the character

'0'

zero

times.

Append the character

'1'

one

times.

This can be performed any number of times.

A

good

string is a string constructed by the above process having a

length

between

low

and

high

(

inclusive

).

Return

the number of

different

good strings that can be constructed satisfying these properties.

Since the answer can be large, return it

modulo

10

9

+ 7

.

Example 1:

Input:

low = 3, high = 3, zero = 1, one = 1

Output:

8

Explanation:

One possible valid good string is "011". It can be constructed as follows: "" -> "0" -> "01" -> "011". All binary strings from "000" to "111" are good strings in this example.

Example 2:

Input:

low = 2, high = 3, zero = 1, one = 2

Output:

5

Explanation:

The good strings are "00", "11", "000", "110", and "011".

Constraints:

1 <= low <= high <= 10

5

1 <= zero, one <= low

Code Snippets

C++:

```
class Solution {
public:
int countGoodStrings(int low, int high, int zero, int one) {

}
};
```

Java:

```
class Solution {
public int countGoodStrings(int low, int high, int zero, int one) {

}
}
```

Python3:

```
class Solution:
def countGoodStrings(self, low: int, high: int, zero: int, one: int) -> int:
```

Python:

```
class Solution(object):
def countGoodStrings(self, low, high, zero, one):
```

```
"""
:type low: int
:type high: int
:type zero: int
:type one: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} low
* @param {number} high
* @param {number} zero
* @param {number} one
* @return {number}
*/
var countGoodStrings = function(low, high, zero, one) {

};
```

TypeScript:

```
function countGoodStrings(low: number, high: number, zero: number, one:
number): number {

};
```

C#:

```
public class Solution {
public int CountGoodStrings(int low, int high, int zero, int one) {

}
}
```

C:

```
int countGoodStrings(int low, int high, int zero, int one) {

}
```

Go:

```
func countGoodStrings(low int, high int, zero int, one int) int {

}
```

Kotlin:

```
class Solution {
fun countGoodStrings(low: Int, high: Int, zero: Int, one: Int): Int {

}
}
```

Swift:

```
class Solution {
func countGoodStrings(_ low: Int, _ high: Int, _ zero: Int, _ one: Int) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_good_strings(low: i32, high: i32, zero: i32, one: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} low
# @param {Integer} high
# @param {Integer} zero
# @param {Integer} one
# @return {Integer}
def count_good_strings(low, high, zero, one)

end
```

PHP:

```
class Solution {

/**
* @param Integer $low
* @param Integer $high
* @param Integer $zero
* @param Integer $one
* @return Integer
*/
function countGoodStrings($low, $high, $zero, $one) {

}
}
```

Dart:

```
class Solution {
int countGoodStrings(int low, int high, int zero, int one) {

}
}
```

Scala:

```
object Solution {
def countGoodStrings(low: Int, high: Int, zero: Int, one: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_good_strings(low :: integer, high :: integer, zero :: integer,
one :: integer) :: integer
def count_good_strings(low, high, zero, one) do

end
end
```

Erlang:

```
-spec count_good_strings(Low :: integer(), High :: integer(), Zero ::
integer(), One :: integer()) -> integer().
count_good_strings(Low, High, Zero, One) ->
.
```

Racket:

```
(define/contract (count-good-strings low high zero one)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Ways To Build Good Strings
* Difficulty: Medium
* Tags: string, dp
*
* 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 countGoodStrings(int low, int high, int zero, int one) {

}
};
```

Java Solution:

```
/**
* Problem: Count Ways To Build Good Strings
* Difficulty: Medium
* Tags: string, dp
*
* 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 countGoodStrings(int low, int high, int zero, int one) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Ways To Build Good Strings
Difficulty: Medium
Tags: string, dp

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 countGoodStrings(self, low: int, high: int, zero: int, one: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countGoodStrings(self, low, high, zero, one):
"""
:type low: int
:type high: int
:type zero: int
:type one: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Ways To Build Good Strings
* Difficulty: Medium
* Tags: string, dp
*
* 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 {number} low
* @param {number} high
* @param {number} zero
* @param {number} one
* @return {number}
*/
var countGoodStrings = function(low, high, zero, one) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Ways To Build Good Strings
* Difficulty: Medium
* Tags: string, dp
*
* 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 countGoodStrings(low: number, high: number, zero: number, one:
number): number {

};
```

C# Solution:

```
/*
* Problem: Count Ways To Build Good Strings
```

```
* Difficulty: Medium
* Tags: string, dp
*
* 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 CountGoodStrings(int low, int high, int zero, int one) {

}
}
```

C Solution:

```
/*
* Problem: Count Ways To Build Good Strings
* Difficulty: Medium
* Tags: string, dp
*
* 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 countGoodStrings(int low, int high, int zero, int one) {

}
```

Go Solution:

```
// Problem: Count Ways To Build Good Strings
// Difficulty: Medium
// Tags: string, dp
//
// 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 countGoodStrings(low int, high int, zero int, one int) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun countGoodStrings(low: Int, high: Int, zero: Int, one: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func countGoodStrings(_ low: Int, _ high: Int, _ zero: Int, _ one: Int) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Count Ways To Build Good Strings
// Difficulty: Medium
// Tags: string, dp
//
// 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_good_strings(low: i32, high: i32, zero: i32, one: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} low
# @param {Integer} high
# @param {Integer} zero
```

```
# @param {Integer} one
# @return {Integer}
def count_good_strings(low, high, zero, one)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $low
* @param Integer $high
* @param Integer $zero
* @param Integer $one
* @return Integer
*/
function countGoodStrings($low, $high, $zero, $one) {

}
}
```

Dart Solution:

```
class Solution {
int countGoodStrings(int low, int high, int zero, int one) {

}
}
```

Scala Solution:

```
object Solution {
def countGoodStrings(low: Int, high: Int, zero: Int, one: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_good_strings(low :: integer, high :: integer, zero :: integer,
one :: integer) :: integer
def count_good_strings(low, high, zero, one) do

end
end
```

Erlang Solution:

```
-spec count_good_strings(Low :: integer(), High :: integer(), Zero ::
integer(), One :: integer()) -> integer().
count_good_strings(Low, High, Zero, One) ->
.
```

Racket Solution:

```
(define/contract (count-good-strings low high zero one)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```