Problem 2533: Number of Good Binary Strings

Problem Information

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

Problem Description

You are given four integers

minLength

,

maxLength

,

oneGroup

and

zeroGroup

.

A binary string is

good

if it satisfies the following conditions:

The length of the string is in the range

[minLength, maxLength]

.

The size of each block of consecutive

1

's is a multiple of

oneGroup

.

For example in a binary string

00

11

0

1111

00

sizes of each block of consecutive ones are

[2,4]

.

The size of each block of consecutive

0

's is a multiple of

zeroGroup

.

For example, in a binary string

00

11

0

1111

00

sizes of each block of consecutive zeros are

[2,1,2]

.

Return

the number of

good

binary strings

. Since the answer may be too large, return it

modulo

10

9

+ 7

.

Note

that

0

is considered a multiple of all the numbers.

Example 1:

Input:

minLength = 2, maxLength = 3, oneGroup = 1, zeroGroup = 2

Output:

5

Explanation:

There are 5 good binary strings in this example: "00", "11", "001", "100", and "111". It can be proven that there are only 5 good strings satisfying all conditions.

Example 2:

Input:

minLength = 4, maxLength = 4, oneGroup = 4, zeroGroup = 3

Output:

1

Explanation:

There is only 1 good binary string in this example: "1111". It can be proven that there is only 1 good string satisfying all conditions.

Constraints:

1 <= minLength <= maxLength <= 10

5

1 <= oneGroup, zeroGroup <= maxLength

Code Snippets

C++:

```
class Solution {
public:
int goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
};
```

Java:

```
class Solution {
public int goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
}
```

Python3:

```
class Solution:
def goodBinaryStrings(self, minLength: int, maxLength: int, oneGroup: int,
zeroGroup: int) -> int:
```

Python:

```
class Solution(object):
def goodBinaryStrings(self, minLength, maxLength, oneGroup, zeroGroup):
"""
```

```
:type minLength: int
:type maxLength: int
:type oneGroup: int
:type zeroGroup: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} minLength
* @param {number} maxLength
* @param {number} oneGroup
* @param {number} zeroGroup
* @return {number}
*/
var goodBinaryStrings = function(minLength, maxLength, oneGroup, zeroGroup) {

};
```

TypeScript:

```
function goodBinaryStrings(minLength: number, maxLength: number, oneGroup:
number, zeroGroup: number): number {

};
```

C#:

```
public class Solution {
public int GoodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
}
```

C:

```
int goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
```

Go:

```
func goodBinaryStrings(minLength int, maxLength int, oneGroup int, zeroGroup
int) int {

}
```

Kotlin:

```
class Solution {
fun goodBinaryStrings(minLength: Int, maxLength: Int, oneGroup: Int,
zeroGroup: Int): Int {

}
}
```

Swift:

```
class Solution {
func goodBinaryStrings(_ minLength: Int, _ maxLength: Int, _ oneGroup: Int, _
zeroGroup: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn good_binary_strings(min_length: i32, max_length: i32, one_group: i32,
zero_group: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} min_length
# @param {Integer} max_length
# @param {Integer} one_group
# @param {Integer} zero_group
# @return {Integer}
def good_binary_strings(min_length, max_length, one_group, zero_group)
```

```
end
```

PHP:

```
class Solution {

/**
* @param Integer $minLength
* @param Integer $maxLength
* @param Integer $oneGroup
* @param Integer $zeroGroup
* @return Integer
*/
function goodBinaryStrings($minLength, $maxLength, $oneGroup, $zeroGroup) {

}
}
```

Dart:

```
class Solution {
int goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
}
```

Scala:

```
object Solution {
def goodBinaryStrings(minLength: Int, maxLength: Int, oneGroup: Int,
zeroGroup: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec good_binary_strings(min_length :: integer, max_length :: integer,
one_group :: integer, zero_group :: integer) :: integer
def good_binary_strings(min_length, max_length, one_group, zero_group) do
```

```
end
end
```

Erlang:

```
-spec good_binary_strings(MinLength :: integer(), MaxLength :: integer(),
OneGroup :: integer(), ZeroGroup :: integer()) -> integer().
good_binary_strings(MinLength, MaxLength, OneGroup, ZeroGroup) ->
.
```

Racket:

```
(define/contract (good-binary-strings minLength maxLength oneGroup zeroGroup)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Good Binary 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 goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Good Binary 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 goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Good Binary 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 goodBinaryStrings(self, minLength: int, maxLength: int, oneGroup: int,
zeroGroup: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def goodBinaryStrings(self, minLength, maxLength, oneGroup, zeroGroup):
"""
:type minLength: int
```

```
:type maxLength: int
:type oneGroup: int
:type zeroGroup: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Good Binary 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} minLength
* @param {number} maxLength
* @param {number} oneGroup
* @param {number} zeroGroup
* @return {number}
*/
var goodBinaryStrings = function(minLength, maxLength, oneGroup, zeroGroup) {

};
```

TypeScript Solution:

```
/**
* Problem: Number of Good Binary 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 goodBinaryStrings(minLength: number, maxLength: number, oneGroup:
number, zeroGroup: number): number {

};
```

C# Solution:

```
/*
* Problem: Number of Good Binary 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 GoodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
}
```

C Solution:

```
/*
* Problem: Number of Good Binary 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 goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {

}
```

Go Solution:

```
// Problem: Number of Good Binary 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 goodBinaryStrings(minLength int, maxLength int, oneGroup int, zeroGroup
int) int {

}
```

Kotlin Solution:

```
class Solution {
fun goodBinaryStrings(minLength: Int, maxLength: Int, oneGroup: Int,
zeroGroup: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func goodBinaryStrings(_ minLength: Int, _ maxLength: Int, _ oneGroup: Int, _
zeroGroup: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Good Binary 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 good_binary_strings(min_length: i32, max_length: i32, one_group: i32,
zero_group: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} min_length
# @param {Integer} max_length
# @param {Integer} one_group
# @param {Integer} zero_group
# @return {Integer}
def good_binary_strings(min_length, max_length, one_group, zero_group)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $minLength
* @param Integer $maxLength
* @param Integer $oneGroup
* @param Integer $zeroGroup
* @return Integer
*/
function goodBinaryStrings($minLength, $maxLength, $oneGroup, $zeroGroup) {

}
}
```

Dart Solution:

```
class Solution {
int goodBinaryStrings(int minLength, int maxLength, int oneGroup, int
zeroGroup) {
```

```
}
}
```

Scala Solution:

```
object Solution {
def goodBinaryStrings(minLength: Int, maxLength: Int, oneGroup: Int,
zeroGroup: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec good_binary_strings(min_length :: integer, max_length :: integer,
one_group :: integer, zero_group :: integer) :: integer
def good_binary_strings(min_length, max_length, one_group, zero_group) do

end
end
```

Erlang Solution:

```
-spec good_binary_strings(MinLength :: integer(), MaxLength :: integer(),
OneGroup :: integer(), ZeroGroup :: integer()) -> integer().
good_binary_strings(MinLength, MaxLength, OneGroup, ZeroGroup) ->
.
```

Racket Solution:

```
(define/contract (good-binary-strings minLength maxLength oneGroup zeroGroup)
(-> exact-integer? exact-integer? exact-integer? exact-integer?
exact-integer?)
)
```