Problem 3490: Count Beautiful Numbers

Problem Information

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

Problem Description

You are given two positive integers,

l

and

r

. A positive integer is called

beautiful

if the product of its digits is divisible by the sum of its digits.

Return the count of

beautiful

numbers between

l

and

r

, inclusive.

Example 1:

Input:

l = 10, r = 20

Output:

2

Explanation:

The beautiful numbers in the range are 10 and 20.

Example 2:

Input:

l = 1, r = 15

Output:

10

Explanation:

The beautiful numbers in the range are 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10.

Constraints:

1 <= l <= r < 10

9

Code Snippets

C++:

```
class Solution {
public:
int beautifulNumbers(int l, int r) {

}
};
```

Java:

```
class Solution {
public int beautifulNumbers(int l, int r) {

}
}
```

Python3:

```
class Solution:
def beautifulNumbers(self, l: int, r: int) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function beautifulNumbers(l: number, r: number): number {

};
```

C#:

```
public class Solution {
public int BeautifulNumbers(int l, int r) {

}
}
```

C:

```
int beautifulNumbers(int l, int r) {

}
```

Go:

```
func beautifulNumbers(l int, r int) int {

}
```

Kotlin:

```
class Solution {
fun beautifulNumbers(l: Int, r: Int): Int {

}
}
```

Swift:

```
class Solution {
func beautifulNumbers(_ l: Int, _ r: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn beautiful_numbers(l: i32, r: i32) -> i32 {

}
}
```

Ruby:

```
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def beautiful_numbers(l, r)

end
```

PHP:

```
class Solution {

/**
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function beautifulNumbers($l, $r) {

}
}
```

Dart:

```
class Solution {
int beautifulNumbers(int l, int r) {

}
}
```

Scala:

```
object Solution {
def beautifulNumbers(l: Int, r: Int): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec beautiful_numbers(l :: integer, r :: integer) :: integer
def beautiful_numbers(l, r) do

end
end
```

Erlang:

```
-spec beautiful_numbers(L :: integer(), R :: integer()) -> integer().
beautiful_numbers(L, R) ->
.
```

Racket:

```
(define/contract (beautiful-numbers l r)
(-> exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Beautiful Numbers
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int beautifulNumbers(int l, int r) {
}
};
```

Java Solution:

```
/**
* Problem: Count Beautiful Numbers
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int beautifulNumbers(int l, int r) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Beautiful Numbers
Difficulty: Hard
Tags: dp

Approach: Dynamic programming with memoization or tabulation
Time Complexity: O(n * m) where n and m are problem dimensions
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def beautifulNumbers(self, l: int, r: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Beautiful Numbers
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

/**
* @param {number} l
* @param {number} r
* @return {number}
*/
var beautifulNumbers = function(l, r) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Beautiful Numbers
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

function beautifulNumbers(l: number, r: number): number {
};
```

C# Solution:

```
/*
* Problem: Count Beautiful Numbers
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int BeautifulNumbers(int l, int r) {

}
}
```

C Solution:

```
/*
* Problem: Count Beautiful Numbers
* Difficulty: Hard
* Tags: dp
*
* Approach: Dynamic programming with memoization or tabulation
* Time Complexity: O(n * m) where n and m are problem dimensions
* Space Complexity: O(n) or O(n * m) for DP table
*/

int beautifulNumbers(int l, int r) {

}
```

Go Solution:

```
// Problem: Count Beautiful Numbers
// Difficulty: Hard
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

func beautifulNumbers(l int, r int) int {

}
```

Kotlin Solution:

```
class Solution {
fun beautifulNumbers(l: Int, r: Int): Int {

}
}
```

Swift Solution:

```
class Solution {
func beautifulNumbers(_ l: Int, _ r: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Beautiful Numbers
// Difficulty: Hard
// Tags: dp
//
// Approach: Dynamic programming with memoization or tabulation
// Time Complexity: O(n * m) where n and m are problem dimensions
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn beautiful_numbers(l: i32, r: i32) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} l
# @param {Integer} r
# @return {Integer}
def beautiful_numbers(l, r)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $l
* @param Integer $r
* @return Integer
*/
function beautifulNumbers($l, $r) {

}
}
```

Dart Solution:

```
class Solution {
int beautifulNumbers(int l, int r) {

}
}
```

Scala Solution:

```
object Solution {
def beautifulNumbers(l: Int, r: Int): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec beautiful_numbers(l :: integer, r :: integer) :: integer
def beautiful_numbers(l, r) do

end
end
```

Erlang Solution:

```
-spec beautiful_numbers(L :: integer(), R :: integer()) -> integer().
beautiful_numbers(L, R) ->
.
```

Racket Solution:

```
(define/contract (beautiful-numbers l r)
(-> exact-integer? exact-integer? exact-integer?)
)
```