Problem 906: Super Palindromes

Problem Information

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

Problem Description

Let's say a positive integer is a

super-palindrome

if it is a palindrome, and it is also the square of a palindrome.

Given two positive integers

left

and

right

represented as strings, return

the number of

super-palindromes

integers in the inclusive range

[left, right]

.

Example 1:

Input:

left = "4", right = "1000"

Output:

4

Explanation

: 4, 9, 121, and 484 are superpalindromes. Note that 676 is not a superpalindrome: 26 * 26 = 676, but 26 is not a palindrome.

Example 2:

Input:

left = "1", right = "2"

Output:

1

Constraints:

1 <= left.length, right.length <= 18

left

and

right

consist of only digits.

left

and

right

cannot have leading zeros.

left

and

right

represent integers in the range

[1, 10

18

- 1]

.

left

is less than or equal to

right

.

Code Snippets

C++:

```
class Solution {
public:
int superpalindromesInRange(string left, string right) {
```

```
}
};
```

Java:

```
class Solution {
public int superpalindromesInRange(String left, String right) {

}
}
```

Python3:

```
class Solution:
def superpalindromesInRange(self, left: str, right: str) -> int:
```

Python:

```
class Solution(object):
def superpalindromesInRange(self, left, right):
"""
:type left: str
:type right: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} left
* @param {string} right
* @return {number}
*/
var superpalindromesInRange = function(left, right) {

};
```

TypeScript:

```
function superpalindromesInRange(left: string, right: string): number {

};
```

C#:

```
public class Solution {
public int SuperpalindromesInRange(string left, string right) {

}
}
```

C:

```
int superpalindromesInRange(char* left, char* right) {

}
```

Go:

```
func superpalindromesInRange(left string, right string) int {

}
```

Kotlin:

```
class Solution {
fun superpalindromesInRange(left: String, right: String): Int {

}
}
```

Swift:

```
class Solution {
func superpalindromesInRange(_ left: String, _ right: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn superpalindromes_in_range(left: String, right: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} left
# @param {String} right
# @return {Integer}
def superpalindromes_in_range(left, right)

end
```

PHP:

```
class Solution {

/**
* @param String $left
* @param String $right
* @return Integer
*/
function superpalindromesInRange($left, $right) {

}
}
```

Dart:

```
class Solution {
int superpalindromesInRange(String left, String right) {

}
}
```

Scala:

```
object Solution {
def superpalindromesInRange(left: String, right: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec superpalindromes_in_range(left :: String.t, right :: String.t) ::
```

```
integer
def superpalindromes_in_range(left, right) do

end
end
```

Erlang:

```
-spec superpalindromes_in_range(Left :: unicode:unicode_binary(), Right ::
unicode:unicode_binary()) -> integer().
superpalindromes_in_range(Left, Right) ->
.
```

Racket:

```
(define/contract (superpalindromes-in-range left right)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Super Palindromes
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
int superpalindromesInRange(string left, string right) {

}
};
```

Java Solution:

```
/**
* Problem: Super Palindromes
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int superpalindromesInRange(String left, String right) {

}
}
```

Python3 Solution:

```
"""
Problem: Super Palindromes
Difficulty: Hard
Tags: string, math

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def superpalindromesInRange(self, left: str, right: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def superpalindromesInRange(self, left, right):
"""
:type left: str
:type right: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Super Palindromes
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {string} left
* @param {string} right
* @return {number}
*/
var superpalindromesInRange = function(left, right) {

};
```

TypeScript Solution:

```
/**
* Problem: Super Palindromes
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function superpalindromesInRange(left: string, right: string): number {

};
```

C# Solution:

```
/*
* Problem: Super Palindromes
* Difficulty: Hard
```

```
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int SuperpalindromesInRange(string left, string right) {

}
}
```

C Solution:

```
/*
* Problem: Super Palindromes
* Difficulty: Hard
* Tags: string, math
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int superpalindromesInRange(char* left, char* right) {

}
```

Go Solution:

```
// Problem: Super Palindromes
// Difficulty: Hard
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func superpalindromesInRange(left string, right string) int {
```

```
}
```

Kotlin Solution:

```
class Solution {
fun superpalindromesInRange(left: String, right: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func superpalindromesInRange(_ left: String, _ right: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Super Palindromes
// Difficulty: Hard
// Tags: string, math
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn superpalindromes_in_range(left: String, right: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} left
# @param {String} right
# @return {Integer}
def superpalindromes_in_range(left, right)
```

```
end
```

PHP Solution:

```
class Solution {

/**
* @param String $left
* @param String $right
* @return Integer
*/
function superpalindromesInRange($left, $right) {

}
}
```

Dart Solution:

```
class Solution {
int superpalindromesInRange(String left, String right) {

}
}
```

Scala Solution:

```
object Solution {
def superpalindromesInRange(left: String, right: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec superpalindromes_in_range(left :: String.t, right :: String.t) ::
integer
def superpalindromes_in_range(left, right) do

end
end
```

Erlang Solution:

```
-spec superpalindromes_in_range(Left :: unicode:unicode_binary(), Right ::
unicode:unicode_binary()) -> integer().
superpalindromes_in_range(Left, Right) ->
.
```

Racket Solution:

```
(define/contract (superpalindromes-in-range left right)
(-> string? string? exact-integer?)
)
```