Problem 2484: Count Palindromic Subsequences

Problem Information

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

Problem Description

Given a string of digits

s

, return

the number of

palindromic subsequences

of

s

having length

5

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

Note:

A string is

palindromic

if it reads the same forward and backward.

A

subsequence

is a string that can be derived from another string by deleting some or no characters without changing the order of the remaining characters.

Example 1:

Input:

s = "103301"

Output:

2

Explanation:

There are 6 possible subsequences of length 5:
"10330","10331","10301","10301","13301","03301". Two of them (both equal to "10301") are palindromic.

Example 2:

Input:

s = "0000000"

Output:

21

Explanation:

All 21 subsequences are "00000", which is palindromic.

Example 3:

Input:

s = "9999900000"

Output:

2

Explanation:

The only two palindromic subsequences are "99999" and "00000".

Constraints:

1 <= s.length <= 10

4

s

consists of digits.

Code Snippets

C++:

```
class Solution {
public:
int countPalindromes(string s) {

}
};
```

Java:

```
class Solution {
public int countPalindromes(String s) {

}
}
```

Python3:

```
class Solution:
def countPalindromes(self, s: str) -> int:
```

Python:

```
class Solution(object):
def countPalindromes(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} s
* @return {number}
*/
var countPalindromes = function(s) {

};
```

TypeScript:

```
function countPalindromes(s: string): number {
```

```
};
```

C#:

```
public class Solution {
public int CountPalindromes(string s) {

}
}
```

C:

```
int countPalindromes(char* s) {

}
```

Go:

```
func countPalindromes(s string) int {

}
```

Kotlin:

```
class Solution {
fun countPalindromes(s: String): Int {

}
}
```

Swift:

```
class Solution {
func countPalindromes(_ s: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn count_palindromes(s: String) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {String} s
# @return {Integer}
def count_palindromes(s)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function countPalindromes($s) {

}
}
```

Dart:

```
class Solution {
int countPalindromes(String s) {

}
}
```

Scala:

```
object Solution {
def countPalindromes(s: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec count_palindromes(s :: String.t) :: integer
def count_palindromes(s) do

end
end
```

Erlang:

```
-spec count_palindromes(S :: unicode:unicode_binary()) -> integer().
count_palindromes(S) ->
.
```

Racket:

```
(define/contract (count-palindromes s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Palindromic Subsequences
* Difficulty: Hard
* 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 countPalindromes(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count Palindromic Subsequences
* Difficulty: Hard
* 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 countPalindromes(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Palindromic Subsequences
Difficulty: Hard
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 countPalindromes(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def countPalindromes(self, s):
"""
:type s: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Count Palindromic Subsequences
* Difficulty: Hard
* 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 {string} s
* @return {number}
*/
var countPalindromes = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Count Palindromic Subsequences
* Difficulty: Hard
* 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 countPalindromes(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Count Palindromic Subsequences
* Difficulty: Hard
* 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 CountPalindromes(string s) {

}
}
```

C Solution:

```
/*
* Problem: Count Palindromic Subsequences
* Difficulty: Hard
* 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 countPalindromes(char* s) {

}
```

Go Solution:

```
// Problem: Count Palindromic Subsequences
// Difficulty: Hard
// 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 countPalindromes(s string) int {

}
```

Kotlin Solution:

```
class Solution {
fun countPalindromes(s: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func countPalindromes(_ s: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Count Palindromic Subsequences
// Difficulty: Hard
// 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_palindromes(s: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @return {Integer}
def count_palindromes(s)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @return Integer
*/
function countPalindromes($s) {

}
}
```

Dart Solution:

```
class Solution {
int countPalindromes(String s) {

}
}
```

Scala Solution:

```
object Solution {
def countPalindromes(s: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec count_palindromes(s :: String.t) :: integer
def count_palindromes(s) do

end
end
```

Erlang Solution:

```
-spec count_palindromes(S :: unicode:unicode_binary()) -> integer().
count_palindromes(S) ->
.
```

Racket Solution:

```
(define/contract (count-palindromes s)
(-> string? exact-integer?)
)
```