Problem 730: Count Different Palindromic Subsequences

Problem Information

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

Problem Description

Given a string s, return

the number of different non-empty palindromic subsequences in

s

. Since the answer may be very large, return it

modulo

10

9

+ 7

.

A subsequence of a string is obtained by deleting zero or more characters from the string.

A sequence is palindromic if it is equal to the sequence reversed.

Two sequences

a

1

, a

2

, ...

and

b

1

, b

2

, ...

are different if there is some

i

for which

a

i

!= b

i

.

Example 1:

Input:

s = "bccb"

Output:

6

Explanation:

The 6 different non-empty palindromic subsequences are 'b', 'c', 'bb', 'cc', 'bcb', 'bccb'. Note that 'bcb' is counted only once, even though it occurs twice.

Example 2:

Input:

s = "abcdabcdabcdabcdabcdabcdabcdabcddcbadcbadcbadcbadcbadcbadcbadcba"

Output:

104860361

Explanation:

There are 3104860382 different non-empty palindromic subsequences, which is 104860361 modulo 10

9

+ 7.

Constraints:

1 <= s.length <= 1000

s[i]

is either

'a'

,

'b'

,

'c'

, or

'd'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int countPalindromicSubsequences(char* s) {

}
```

Go:

```
func countPalindromicSubsequences(s string) int {
}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Count Different 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 countPalindromicSubsequences(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count Different 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 countPalindromicSubsequences(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Different 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 countPalindromicSubsequences(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Different 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 countPalindromicSubsequences = function(s) {
};
```

TypeScript Solution:

```
/**
* Problem: Count Different 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 countPalindromicSubsequences(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Count Different 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 CountPalindromicSubsequences(string s) {

}
}
```

C Solution:

```
/*
* Problem: Count Different 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 countPalindromicSubsequences(char* s) {

}
```

Go Solution:

```
// Problem: Count Different 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 countPalindromicSubsequences(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Different 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_palindromic_subsequences(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

```
class Solution {
int countPalindromicSubsequences(String s) {
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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