

Problem 1930: Unique Length-3 Palindromic Subsequences

Problem Information

Difficulty: Medium

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

Given a string

s

, return

the number of

unique palindromes of length three

that are a

subsequence

of

s

Note that even if there are multiple ways to obtain the same subsequence, it is still only counted

once

.

A

palindrome

is a string that reads the same forwards and backwards.

A

subsequence

of a string is a new string generated from the original string with some characters (can be none) deleted without changing the relative order of the remaining characters.

For example,

"ace"

is a subsequence of

"

a

b

c

d

e

"

.

Example 1:

Input:

s = "aabca"

Output:

3

Explanation:

The 3 palindromic subsequences of length 3 are: - "aba" (subsequence of "

a

a

b

c

a

") - "aaa" (subsequence of "

aa

bc

a

") - "aca" (subsequence of "

a

ab

ca

")

Example 2:

Input:

s = "adc"

Output:

0

Explanation:

There are no palindromic subsequences of length 3 in "adc".

Example 3:

Input:

s = "bbcbaba"

Output:

4

Explanation:

The 4 palindromic subsequences of length 3 are: - "bbb" (subsequence of "

bb

c

b

aba") - "bcb" (subsequence of "

b

b

cb

aba") - "bab" (subsequence of "

b

bcb

ab

a") - "aba" (subsequence of "bccb

aba

")

Constraints:

$3 \leq s.length \leq 10$

5

s

consists of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
    int countPalindromicSubsequence(string s) {
        }
};
```

Java:

```
class Solution {  
    public int countPalindromicSubsequence(String s) {  
  
    }  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function countPalindromicSubsequence(s: string): number {  
  
};
```

C#:

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

```
}
```

```
}
```

C:

```
int countPalindromicSubsequence(char* s) {  
  
}
```

Go:

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

Kotlin:

```
class Solution {  
    fun countPalindromicSubsequence(s: String): Int {  
  
    }  
}
```

Swift:

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

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

    }
}
```

Dart:

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

Scala:

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

Elixir:

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

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
 * Problem: Unique Length-3 Palindromic Subsequences
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

class Solution {
public:
    int countPalindromicSubsequence(string s) {

    }
};
```

Java Solution:

```
/**
 * Problem: Unique Length-3 Palindromic Subsequences
 * Difficulty: Medium
 * Tags: array, string, hash
 *
```

```

* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

```

```

class Solution {
    public int countPalindromicSubsequence(String s) {
        }
    }
}

```

Python3 Solution:

```

"""
Problem: Unique Length-3 Palindromic Subsequences
Difficulty: Medium
Tags: array, string, hash

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
    def countPalindromicSubsequence(self, s: str) -> int:
        # TODO: Implement optimized solution
        pass

```

Python Solution:

```

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

```

JavaScript Solution:

```

/**
 * Problem: Unique Length-3 Palindromic Subsequences

```

```

* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

```

```

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

```

TypeScript Solution:

```

/**
* Problem: Unique Length-3 Palindromic Subsequences
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

```

```

function countPalindromicSubsequence(s: string): number {
}

```

C# Solution:

```

/*
* Problem: Unique Length-3 Palindromic Subsequences
* Difficulty: Medium
* Tags: array, string, hash
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)

```

```

* Space Complexity: O(n) for hash map
*/
public class Solution {
    public int CountPalindromicSubsequence(string s) {
        }
    }
}

```

C Solution:

```

/*
 * Problem: Unique Length-3 Palindromic Subsequences
 * Difficulty: Medium
 * Tags: array, string, hash
 *
 * Approach: Use two pointers or sliding window technique
 * Time Complexity: O(n) or O(n log n)
 * Space Complexity: O(n) for hash map
 */

int countPalindromicSubsequence(char* s) {
}

```

Go Solution:

```

// Problem: Unique Length-3 Palindromic Subsequences
// Difficulty: Medium
// Tags: array, string, hash
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

func countPalindromicSubsequence(s string) int {
}

```

Kotlin Solution:

```
class Solution {  
    fun countPalindromicSubsequence(s: String): Int {  
        }  
        }  
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Unique Length-3 Palindromic Subsequences  
// Difficulty: Medium  
// Tags: array, string, hash  
//  
// Approach: Use two pointers or sliding window technique  
// Time Complexity: O(n) or O(n log n)  
// Space Complexity: O(n) for hash map  
  
impl Solution {  
    pub fn count_palindromic_subsequence(s: String) -> i32 {  
        }  
        }  
}
```

Ruby Solution:

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

PHP Solution:

```
class Solution {
```

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

}
```

Dart Solution:

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

}
```

Scala Solution:

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

}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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