Problem 3504: Longest Palindrome After Substring Concatenation II

Problem Information

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

Problem Description

You are given two strings,

s

and

t

.

You can create a new string by selecting a

substring

from

s

(possibly empty) and a substring from

t

(possibly empty), then concatenating them

in order

.

Return the length of the

longest

palindrome

that can be formed this way.

Example 1:

Input:

s = "a", t = "a"

Output:

2

Explanation:

Concatenating

"a"

from

s

and

"a"

from

t

results in

"aa"

, which is a palindrome of length 2.

Example 2:

Input:

s = "abc", t = "def"

Output:

1

Explanation:

Since all characters are different, the longest palindrome is any single character, so the answer is 1.

Example 3:

Input:

s = "b", t = "aaaa"

Output:

4

Explanation:

Selecting "

aaaa

" from

t

is the longest palindrome, so the answer is 4.

Example 4:

Input:

s = "abcde", t = "ecdba"

Output:

5

Explanation:

Concatenating

"abc"

from

s

and

"ba"

from

t

results in

"abcba"

, which is a palindrome of length 5.

Constraints:

1 <= s.length, t.length <= 1000

s

and

t

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int longestPalindrome(string s, string t) {

}
};
```

Java:

```
class Solution {
public int longestPalindrome(String s, String t) {

}
}
```

Python3:

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

Python:

```
class Solution(object):
def longestPalindrome(self, s, t):
"""
:type s: str
```

```
:type t: str
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

```
function longestPalindrome(s: string, t: string): number {

};
```

C#:

```
public class Solution {
public int LongestPalindrome(string s, string t) {

}
}
```

C:

```
int longestPalindrome(char* s, char* t) {

}
```

Go:

```
func longestPalindrome(s string, t string) int {

}
```

Kotlin:

```
class Solution {
fun longestPalindrome(s: String, t: String): Int {

}
}
```

Swift:

```
class Solution {
func longestPalindrome(_ s: String, _ t: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_palindrome(s: String, t: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} s
# @param {String} t
# @return {Integer}
def longest_palindrome(s, t)

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Integer
*/
function longestPalindrome($s, $t) {

}
```

```
}
```

Dart:

```
class Solution {
int longestPalindrome(String s, String t) {

}
}
```

Scala:

```
object Solution {
def longestPalindrome(s: String, t: String): Int = {

}
}
```

Elixir:

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

end
end
```

Erlang:

```
-spec longest_palindrome(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> integer().
longest_palindrome(S, T) ->
.
```

Racket:

```
(define/contract (longest-palindrome s t)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Palindrome After Substring Concatenation II
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int longestPalindrome(string s, string t) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Palindrome After Substring Concatenation II
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int longestPalindrome(String s, String t) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Palindrome After Substring Concatenation II
```

```
Difficulty: Hard
Tags: array, string, tree, dp

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Longest Palindrome After Substring Concatenation II
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

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

```
};
```

TypeScript Solution:

```
/**
* Problem: Longest Palindrome After Substring Concatenation II
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function longestPalindrome(s: string, t: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Palindrome After Substring Concatenation II
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* 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 LongestPalindrome(string s, string t) {

}
}
```

C Solution:

```
/*
* Problem: Longest Palindrome After Substring Concatenation II
```

```
* Difficulty: Hard
* Tags: array, string, tree, dp
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int longestPalindrome(char* s, char* t) {

}
```

Go Solution:

```
// Problem: Longest Palindrome After Substring Concatenation II
// Difficulty: Hard
// Tags: array, string, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func longestPalindrome(s string, t string) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestPalindrome(s: String, t: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestPalindrome(_ s: String, _ t: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Palindrome After Substring Concatenation II
// Difficulty: Hard
// Tags: array, string, tree, dp
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn longest_palindrome(s: String, t: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} s
# @param {String} t
# @return {Integer}
def longest_palindrome(s, t)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $s
* @param String $t
* @return Integer
*/
function longestPalindrome($s, $t) {

}
}
```

Dart Solution:

```
class Solution {
int longestPalindrome(String s, String t) {

}
}
```

Scala Solution:

```
object Solution {
def longestPalindrome(s: String, t: String): Int = {

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

```
-spec longest_palindrome(S :: unicode:unicode_binary(), T ::
unicode:unicode_binary()) -> integer().
longest_palindrome(S, T) ->
.
```

Racket Solution:

```
(define/contract (longest-palindrome s t)
(-> string? string? exact-integer?)
)
```