Problem 1771: Maximize Palindrome Length From Subsequences

Problem Information

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

Problem Description

You are given two strings,

word1

and

word2

. You want to construct a string in the following manner:

Choose some

non-empty

subsequence

subsequence1

from

word1

.

Choose some

non-empty

subsequence

subsequence2

from

word2

.

Concatenate the subsequences:

subsequence1 + subsequence2

, to make the string.

Return

the

length

of the longest

palindrome

that can be constructed in the described manner.

If no palindromes can be constructed, return

0

.

A

subsequence

of a string

s

is a string that can be made by deleting some (possibly none) characters from

s

without changing the order of the remaining characters.

A

palindrome

is a string that reads the same forward as well as backward.

Example 1:

Input:

word1 = "cacb", word2 = "cbba"

Output:

5

Explanation:

Choose "ab" from word1 and "cba" from word2 to make "abcba", which is a palindrome.

Example 2:

Input:

word1 = "ab", word2 = "ab"

Output:

3

Explanation:

Choose "ab" from word1 and "a" from word2 to make "aba", which is a palindrome.

Example 3:

Input:

word1 = "aa", word2 = "bb"

Output:

0

Explanation:

You cannot construct a palindrome from the described method, so return 0.

Constraints:

1 <= word1.length, word2.length <= 1000

word1

and

word2

consist of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
```

```
int longestPalindrome(string word1, string word2) {

}
};
```

Java:

```
class Solution {
public int longestPalindrome(String word1, String word2) {

}
}
```

Python3:

```
class Solution:
def longestPalindrome(self, word1: str, word2: str) -> int:
```

Python:

```
class Solution(object):
def longestPalindrome(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} word1
* @param {string} word2
* @return {number}
*/
var longestPalindrome = function(word1, word2) {

};
```

TypeScript:

```
function longestPalindrome(word1: string, word2: string): number {

};
```

C#:

```
public class Solution {
public int LongestPalindrome(string word1, string word2) {

}
}
```

C:

```
int longestPalindrome(char* word1, char* word2) {

}
```

Go:

```
func longestPalindrome(word1 string, word2 string) int {

}
```

Kotlin:

```
class Solution {
fun longestPalindrome(word1: String, word2: String): Int {

}
}
```

Swift:

```
class Solution {
func longestPalindrome(_ word1: String, _ word2: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_palindrome(word1: String, word2: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} word1
# @param {String} word2
# @return {Integer}
def longest_palindrome(word1, word2)

end
```

PHP:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Integer
*/
function longestPalindrome($word1, $word2) {

}
}
```

Dart:

```
class Solution {
int longestPalindrome(String word1, String word2) {

}
}
```

Scala:

```
object Solution {
def longestPalindrome(word1: String, word2: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_palindrome(word1 :: String.t, word2 :: String.t) :: integer
def longest_palindrome(word1, word2) do

end
end
```

Erlang:

```
-spec longest_palindrome(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
longest_palindrome(Word1, Word2) ->
.
```

Racket:

```
(define/contract (longest-palindrome word1 word2)
(-> string? string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximize Palindrome Length From 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 longestPalindrome(string word1, string word2) {

}
};
```

Java Solution:

```
/**
* Problem: Maximize Palindrome Length From 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 longestPalindrome(String word1, String word2) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximize Palindrome Length From 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 longestPalindrome(self, word1: str, word2: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestPalindrome(self, word1, word2):
"""
:type word1: str
:type word2: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Maximize Palindrome Length From 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} word1
* @param {string} word2
* @return {number}
*/
var longestPalindrome = function(word1, word2) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximize Palindrome Length From 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 longestPalindrome(word1: string, word2: string): number {
};
```

C# Solution:

```
/*
* Problem: Maximize Palindrome Length From 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 LongestPalindrome(string word1, string word2) {

}
}
```

C Solution:

```
/*
* Problem: Maximize Palindrome Length From 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 longestPalindrome(char* word1, char* word2) {

}
```

Go Solution:

```
// Problem: Maximize Palindrome Length From 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 longestPalindrome(word1 string, word2 string) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestPalindrome(word1: String, word2: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestPalindrome(_ word1: String, _ word2: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Maximize Palindrome Length From 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 longest_palindrome(word1: String, word2: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} word1
# @param {String} word2
# @return {Integer}
def longest_palindrome(word1, word2)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $word1
* @param String $word2
* @return Integer
*/
function longestPalindrome($word1, $word2) {

}
}
```

Dart Solution:

```
class Solution {
int longestPalindrome(String word1, String word2) {

}
}
```

Scala Solution:

```
object Solution {
def longestPalindrome(word1: String, word2: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_palindrome(word1 :: String.t, word2 :: String.t) :: integer
def longest_palindrome(word1, word2) do

end
end
```

Erlang Solution:

```
-spec longest_palindrome(Word1 :: unicode:unicode_binary(), Word2 ::
unicode:unicode_binary()) -> integer().
longest_palindrome(Word1, Word2) ->
.
```

Racket Solution:

```
(define/contract (longest-palindrome word1 word2)
(-> string? string? exact-integer?)
)
```