Problem 2002: Maximum Product of the Length of Two Palindromic Subsequences

Problem Information

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

Problem Description

Given a string

s

, find two

disjoint palindromic subsequences

of

s

such that the

product

of their lengths is

maximized

. The two subsequences are

disjoint

if they do not both pick a character at the same index.

Return

the

maximum

possible

product

of the lengths of the two palindromic subsequences

.

A

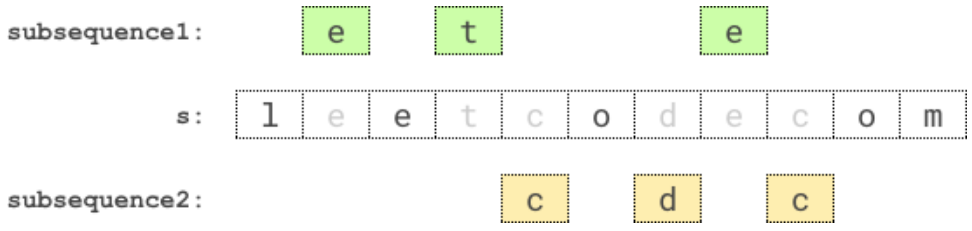
subsequence

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

palindromic

if it reads the same forward and backward.

Example 1:



Input:

s = "leetcodecom"

Output:

9

Explanation

: An optimal solution is to choose "ete" for the 1

st

subsequence and "cdc" for the 2

nd

subsequence. The product of their lengths is: 3 * 3 = 9.

Example 2:

Input:

s = "bb"

Output:

1

Explanation

: An optimal solution is to choose "b" (the first character) for the 1

st

subsequence and "b" (the second character) for the 2

nd

subsequence. The product of their lengths is: 1 * 1 = 1.

Example 3:

Input:

s = "accbcaxxcxx"

Output:

25

Explanation

: An optimal solution is to choose "accca" for the 1

st

subsequence and "xxcxx" for the 2

nd

subsequence. The product of their lengths is: 5 * 5 = 25.

Constraints:

2 <= s.length <= 12

s

consists of lowercase English letters only.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int maxProduct(char* s) {

}
```

Go:

```
func maxProduct(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (max-product s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Maximum Product of the Length of Two Palindromic Subsequences
* Difficulty: Medium
* 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 maxProduct(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Product of the Length of Two Palindromic Subsequences
* Difficulty: Medium
* 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 maxProduct(String s) {
```

```
}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product of the Length of Two Palindromic Subsequences
Difficulty: Medium
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 maxProduct(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Maximum Product of the Length of Two Palindromic Subsequences
* Difficulty: Medium
* 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 maxProduct = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product of the Length of Two Palindromic Subsequences
* Difficulty: Medium
* 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 maxProduct(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Product of the Length of Two Palindromic Subsequences
* Difficulty: Medium
* 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 MaxProduct(string s) {

}
```

```
}
```

C Solution:

```
/*
* Problem: Maximum Product of the Length of Two Palindromic Subsequences
* Difficulty: Medium
* 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 maxProduct(char* s) {

}
```

Go Solution:

```
// Problem: Maximum Product of the Length of Two Palindromic Subsequences
// Difficulty: Medium
// 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 maxProduct(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product of the Length of Two Palindromic Subsequences
// Difficulty: Medium
// 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 max_product(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (max-product s)
(-> string? exact-integer?)
)
```