Problem 1960: Maximum Product of the Length of Two Palindromic Substrings

Problem Information

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

Problem Description

You are given a

0-indexed

string

s

and are tasked with finding two

non-intersecting palindromic

substrings of

odd

length such that the product of their lengths is maximized.

More formally, you want to choose four integers

i

,

j

,

k

,

l

such that

0 <= i <= j < k <= l < s.length

and both the substrings

s[i...j]

and

s[k...l]

are palindromes and have odd lengths.

s[i...j]

denotes a substring from index

i

to index

j

inclusive

.

Return

the

maximum

possible product of the lengths of the two non-intersecting palindromic substrings.

A

palindrome

is a string that is the same forward and backward. A

substring

is a contiguous sequence of characters in a string.

Example 1:

Input:

s = "ababbb"

Output:

9

Explanation:

Substrings "aba" and "bbb" are palindromes with odd length. product = 3 * 3 = 9.

Example 2:

Input:

s = "zaaaxbbby"

Output:

9

Explanation:

Substrings "aaa" and "bbb" are palindromes with odd length. product = 3 * 3 = 9.

Constraints:

2 <= s.length <= 10

5

s

consists of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

C:

```
long long maxProduct(char* s) {

}
```

Go:

```
func maxProduct(s string) int64 {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

```
impl Solution {
pub fn max_product(s: String) -> i64 {

}
}
```

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): Long = {

}
}
```

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 Substrings
* Difficulty: Hard
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
long long maxProduct(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Maximum Product of the Length of Two Palindromic Substrings
* Difficulty: Hard
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public long maxProduct(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Maximum Product of the Length of Two Palindromic Substrings
Difficulty: Hard
Tags: string, tree, hash
```

```
Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

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 Substrings
* Difficulty: Hard
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Maximum Product of the Length of Two Palindromic Substrings
* Difficulty: Hard
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function maxProduct(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Maximum Product of the Length of Two Palindromic Substrings
* Difficulty: Hard
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public long MaxProduct(string s) {

}
}
```

C Solution:

```
/*
* Problem: Maximum Product of the Length of Two Palindromic Substrings
* Difficulty: Hard
* Tags: string, tree, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
```

```
*/

long long maxProduct(char* s) {

}
```

Go Solution:

```
// Problem: Maximum Product of the Length of Two Palindromic Substrings
// Difficulty: Hard
// Tags: string, tree, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func maxProduct(s string) int64 {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Maximum Product of the Length of Two Palindromic Substrings
// Difficulty: Hard
// Tags: string, tree, hash
```

```
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

impl Solution {
pub fn max_product(s: String) -> i64 {

}
}
```

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): Long = {

}
}
```

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?)
)
```