# Problem 2472: Maximum Number of Non-overlapping Palindrome Substrings

## Problem Information

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

## Problem Description

You are given a string

s

and a

positive

integer

k

.

Select a set of

non-overlapping

substrings from the string

s

that satisfy the following conditions:

The

length

of each substring is

at least

k

.

Each substring is a

palindrome

.

Return

the

maximum

number of substrings in an optimal selection

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

s = "abaccdbbd", k = 3

Output:

2

Explanation:

We can select the substrings underlined in s = "

aba

cc

dbbd

". Both "aba" and "dbbd" are palindromes and have a length of at least k = 3. It can be shown that we cannot find a selection with more than two valid substrings.

Example 2:

Input:

s = "adbcda", k = 2

Output:

0

Explanation:

There is no palindrome substring of length at least 2 in the string.

Constraints:

1 <= k <= s.length <= 2000

s

consists of lowercase English letters.

## Code Snippets

**C++:**

```
class Solution {
public:
int maxPalindromes(string s, int k) {

}
};
```

**Java:**

```
class Solution {
public int maxPalindromes(String s, int k) {

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

```
};
```

**TypeScript:**

```
function maxPalindromes(s: string, k: number): number {

};
```

**C#:**

```
public class Solution {
public int MaxPalindromes(string s, int k) {

}
}
```

**C:**

```
int maxPalindromes(char* s, int k) {

}
```

**Go:**

```
func maxPalindromes(s string, k int) int {

}
```

**Kotlin:**

```
class Solution {
fun maxPalindromes(s: String, k: Int): Int {

}
}
```

**Swift:**

```
class Solution {
func maxPalindromes(_ s: String, _ k: Int) -> Int {
```

```
}
}
```

**Rust:**

```
impl Solution {
pub fn max_palindromes(s: String, k: i32) -> i32 {

}
}
```

**Ruby:**

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def max_palindromes(s, k)

end
```

**PHP:**

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function maxPalindromes($s, $k) {

}
}
```

**Dart:**

```
class Solution {
int maxPalindromes(String s, int k) {

}
}
```

### Scala:

```
object Solution {
def maxPalindromes(s: String, k: Int): Int = {

}
}
```

### Elixir:

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

end
end
```

### Erlang:

```
-spec max_palindromes(S :: unicode:unicode_binary(), K :: integer()) ->
integer().
max_palindromes(S, K) ->
.
```

### Racket:

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

## Solutions

### C++ Solution:

```
/*
* Problem: Maximum Number of Non-overlapping Palindrome Substrings
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy
*
* 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 maxPalindromes(string s, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Maximum Number of Non-overlapping Palindrome Substrings
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy
*
* 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 maxPalindromes(String s, int k) {

}
}
```

### Python3 Solution:

```
"""
Problem: Maximum Number of Non-overlapping Palindrome Substrings
Difficulty: Hard
Tags: array, string, tree, dp, greedy

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 maxPalindromes(self, s: str, k: int) -> int:
```

```
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Maximum Number of Non-overlapping Palindrome Substrings
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy
*
* 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 {number} k
* @return {number}
*/
var maxPalindromes = function(s, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Maximum Number of Non-overlapping Palindrome Substrings
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy
*
```

```
* 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 maxPalindromes(s: string, k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Maximum Number of Non-overlapping Palindrome Substrings
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy
*
* 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 MaxPalindromes(string s, int k) {

}
}
```

### C Solution:

```
/*
* Problem: Maximum Number of Non-overlapping Palindrome Substrings
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy
*
* 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 maxPalindromes(char* s, int k) {
```

```
}
```

### Go Solution:

```
// Problem: Maximum Number of Non-overlapping Palindrome Substrings
// Difficulty: Hard
// Tags: array, string, tree, dp, greedy
//
// 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 maxPalindromes(s string, k int) int {

}
```

### Kotlin Solution:

```
class Solution {
fun maxPalindromes(s: String, k: Int): Int {

}
}
```

### Swift Solution:

```
class Solution {
func maxPalindromes(_ s: String, _ k: Int) -> Int {

}
}
```

### Rust Solution:

```
// Problem: Maximum Number of Non-overlapping Palindrome Substrings
// Difficulty: Hard
// Tags: array, string, tree, dp, greedy
//
// 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 max_palindromes(s: String, k: i32) -> i32 {

}
}
```

**Ruby Solution:**

```
# @param {String} s
# @param {Integer} k
# @return {Integer}
def max_palindromes(s, k)

end
```

**PHP Solution:**

```
class Solution {

/**
* @param String $s
* @param Integer $k
* @return Integer
*/
function maxPalindromes($s, $k) {

}
}
```

**Dart Solution:**

```
class Solution {
int maxPalindromes(String s, int k) {

}
}
```

**Scala Solution:**

```
object Solution {
def maxPalindromes(s: String, k: Int): Int = {

}
}
```

### Elixir Solution:

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

end
end
```

### Erlang Solution:

```
-spec max_palindromes(S :: unicode:unicode_binary(), K :: integer()) ->
integer().
max_palindromes(S, K) ->
.
```

### Racket Solution:

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