# Problem 1278: Palindrome Partitioning III

## Problem Information

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

## Problem Description

You are given a string

s

containing lowercase letters and an integer

k

. You need to :

First, change some characters of

s

to other lowercase English letters.

Then divide

s

into

k

non-empty disjoint substrings such that each substring is a palindrome.

Return

the minimal number of characters that you need to change to divide the string

.

Example 1:

Input:

s = "abc", k = 2

Output:

1

Explanation:

You can split the string into "ab" and "c", and change 1 character in "ab" to make it palindrome.

Example 2:

Input:

s = "aabbc", k = 3

Output:

0

Explanation:

You can split the string into "aa", "bb" and "c", all of them are palindrome.

Example 3:

Input:

s = "leetcode", k = 8

Output:

0

Constraints:

1 <= k <= s.length <= 100

.

s

only contains lowercase English letters.

## Code Snippets

**C++:**

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

}
};
```

**Java:**

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

}
}
```

**Python3:**

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

**Python:**

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

**JavaScript:**

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

};
```

**TypeScript:**

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

};
```

**C#:**

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

}
}
```

**C:**

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

}
```

**Go:**

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

}
```

**Kotlin:**

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

}
}
```

**Swift:**

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

}
}
```

**Rust:**

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

}
}
```

**Ruby:**

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

end
```

**PHP:**

```
class Solution {

/**
* @param String $s
```

```
* @param Integer $k
* @return Integer
*/
function palindromePartition($s, $k) {

}
}
```

**Dart:**

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

}
}
```

**Scala:**

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

}
}
```

**Elixir:**

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

end
end
```

**Erlang:**

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

**Racket:**

```
(define/contract (palindrome-partition s k)
(-> string? exact-integer? exact-integer?)
)
```

## Solutions

### C++ Solution:

```
/*
* Problem: Palindrome Partitioning III
* Difficulty: Hard
* Tags: string, tree, 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 palindromePartition(string s, int k) {

}
};
```

### Java Solution:

```
/**
* Problem: Palindrome Partitioning III
* Difficulty: Hard
* Tags: string, tree, 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 palindromePartition(String s, int k) {

}
```

```
}
```

### Python3 Solution:

```
"""
Problem: Palindrome Partitioning III
Difficulty: Hard
Tags: string, tree, 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 palindromePartition(self, s: str, k: int) -> int:
# TODO: Implement optimized solution
pass
```

### Python Solution:

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

### JavaScript Solution:

```
/**
* Problem: Palindrome Partitioning III
* Difficulty: Hard
* Tags: string, tree, 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
* @param {number} k
* @return {number}
*/
var palindromePartition = function(s, k) {

};
```

### TypeScript Solution:

```
/**
* Problem: Palindrome Partitioning III
* Difficulty: Hard
* Tags: string, tree, 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 palindromePartition(s: string, k: number): number {

};
```

### C# Solution:

```
/*
* Problem: Palindrome Partitioning III
* Difficulty: Hard
* Tags: string, tree, 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 PalindromePartition(string s, int k) {

}
```

```
}
```

### C Solution:

```
/*
* Problem: Palindrome Partitioning III
* Difficulty: Hard
* Tags: string, tree, 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 palindromePartition(char* s, int k) {

}
```

### Go Solution:

```
// Problem: Palindrome Partitioning III
// Difficulty: Hard
// Tags: string, tree, 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 palindromePartition(s string, k int) int {

}
```

### Kotlin Solution:

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

}
}
```

### Swift Solution:

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

}
}
```

### Rust Solution:

```
// Problem: Palindrome Partitioning III
// Difficulty: Hard
// Tags: string, tree, 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 palindrome_partition(s: String, k: i32) -> i32 {

}
}
```

### Ruby Solution:

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

end
```

### PHP Solution:

```
class Solution {

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

```
}
}
```

**Dart Solution:**

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

}
}
```

**Scala Solution:**

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

}
}
```

**Elixir Solution:**

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

end
end
```

**Erlang Solution:**

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

**Racket Solution:**

```
(define/contract (palindrome-partition s k)
(-> string? exact-integer? exact-integer?)
)
```