Problem 1147: Longest Chunked Palindrome Decomposition

Problem Information

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

Problem Description

You are given a string

text

. You should split it to k substrings

(subtext

1

, subtext

2

, ..., subtext

k

)

such that:

subtext

i

is a

non-empty

string.

The concatenation of all the substrings is equal to

text

(i.e.,

subtext

1

+ subtext

2

+ ... + subtext

k

== text

).

subtext

i

== subtext

k - i + 1

for all valid values of

i

(i.e.,

1 <= i <= k

).

Return the largest possible value of

k

.

Example 1:

Input:

text = "ghiabcdefhelloadamhelloabcdefghi"

Output:

7

Explanation:

We can split the string on "(ghi)(abcdef)(hello)(adam)(hello)(abcdef)(ghi)".

Example 2:

Input:

text = "merchant"

Output:

1

Explanation:

We can split the string on "(merchant)".

Example 3:

Input:

text = "antaprezatepzapreanta"

Output:

11

Explanation:

We can split the string on "(a)(nt)(a)(pre)(za)(tep)(za)(pre)(a)(nt)(a)".

Constraints:

1 <= text.length <= 1000

text

consists only of lowercase English characters.

Code Snippets

C++:

```
class Solution {
public:
int longestDecomposition(string text) {

}
};
```

Java:

```
class Solution {
public int longestDecomposition(String text) {
}
}
```

Python3:

```
class Solution:
def longestDecomposition(self, text: str) -> int:
```

Python:

```
class Solution(object):
def longestDecomposition(self, text):
"""
:type text: str
:rtype: int
"""
```

JavaScript:

```
/**
* @param {string} text
* @return {number}
*/
var longestDecomposition = function(text) {

};
```

TypeScript:

```
function longestDecomposition(text: string): number {

};
```

C#:

```
public class Solution {
public int LongestDecomposition(string text) {

}
}
```

C:

```
int longestDecomposition(char* text) {

}
```

Go:

```
func longestDecomposition(text string) int {

}
```

Kotlin:

```
class Solution {
fun longestDecomposition(text: String): Int {

}
}
```

Swift:

```
class Solution {
func longestDecomposition(_ text: String) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn longest_decomposition(text: String) -> i32 {

}
}
```

Ruby:

```
# @param {String} text
# @return {Integer}
def longest_decomposition(text)

end
```

PHP:

```
class Solution {

/**
* @param String $text
* @return Integer
*/
function longestDecomposition($text) {

}
}
```

Dart:

```
class Solution {
int longestDecomposition(String text) {

}
}
```

Scala:

```
object Solution {
def longestDecomposition(text: String): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec longest_decomposition(text :: String.t) :: integer
def longest_decomposition(text) do

end
end
```

Erlang:

```
-spec longest_decomposition(Text :: unicode:unicode_binary()) -> integer().
longest_decomposition(Text) ->
.
```

Racket:

```
(define/contract (longest-decomposition text)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Longest Chunked Palindrome Decomposition
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy, hash
*
* 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 longestDecomposition(string text) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Chunked Palindrome Decomposition
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy, hash
*
* 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 longestDecomposition(String text) {
}
}
```

Python3 Solution:

```
"""
Problem: Longest Chunked Palindrome Decomposition
Difficulty: Hard
Tags: array, string, tree, dp, greedy, hash

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 longestDecomposition(self, text: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def longestDecomposition(self, text):
"""
:type text: str
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Chunked Palindrome Decomposition
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy, hash
*
* 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} text
* @return {number}
*/
var longestDecomposition = function(text) {

};
```

TypeScript Solution:

```
/**
* Problem: Longest Chunked Palindrome Decomposition
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy, hash
*
* 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 longestDecomposition(text: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Chunked Palindrome Decomposition
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy, hash
*
* 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 LongestDecomposition(string text) {

}
}
```

C Solution:

```
/*
* Problem: Longest Chunked Palindrome Decomposition
* Difficulty: Hard
* Tags: array, string, tree, dp, greedy, hash
*
* 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 longestDecomposition(char* text) {

}
```

Go Solution:

```
// Problem: Longest Chunked Palindrome Decomposition
// Difficulty: Hard
// Tags: array, string, tree, dp, greedy, hash
//
// 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 longestDecomposition(text string) int {

}
```

Kotlin Solution:

```
class Solution {
fun longestDecomposition(text: String): Int {

}
}
```

Swift Solution:

```
class Solution {
func longestDecomposition(_ text: String) -> Int {

}
}
```

Rust Solution:

```
// Problem: Longest Chunked Palindrome Decomposition
// Difficulty: Hard
// Tags: array, string, tree, dp, greedy, hash
//
// 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 longest_decomposition(text: String) -> i32 {

}
}
```

Ruby Solution:

```
# @param {String} text
# @return {Integer}
def longest_decomposition(text)

end
```

PHP Solution:

```
class Solution {

/**
* @param String $text
* @return Integer
*/
function longestDecomposition($text) {

}
}
```

Dart Solution:

```
class Solution {
int longestDecomposition(String text) {

}
}
```

Scala Solution:

```
object Solution {
def longestDecomposition(text: String): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec longest_decomposition(text :: String.t) :: integer
def longest_decomposition(text) do

end
end
```

Erlang Solution:

```
-spec longest_decomposition(Text :: unicode:unicode_binary()) -> integer().
longest_decomposition(Text) ->
.
```

Racket Solution:

```
(define/contract (longest-decomposition text)
(-> string? exact-integer?)
)
```