Problem 664: Strange Printer

Problem Information

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

Problem Description

There is a strange printer with the following two special properties:

The printer can only print a sequence of

the same character

each time.

At each turn, the printer can print new characters starting from and ending at any place and will cover the original existing characters.

Given a string

s

, return

the minimum number of turns the printer needed to print it

.

Example 1:

Input:

s = "aaabbb"

Output:

2

Explanation:

Print "aaa" first and then print "bbb".

Example 2:

Input:

s = "aba"

Output:

2

Explanation:

Print "aaa" first and then print "b" from the second place of the string, which will cover the existing character 'a'.

Constraints:

1 <= s.length <= 100

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
int strangePrinter(string s) {
}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int strangePrinter(char* s) {

}
```

Go:

```
func strangePrinter(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

```
(define/contract (strange-printer s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Strange Printer
* Difficulty: Hard
* 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 strangePrinter(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Strange Printer
* Difficulty: Hard
* 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 strangePrinter(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Strange Printer
Difficulty: Hard
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 strangePrinter(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Strange Printer
* Difficulty: Hard
* 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 strangePrinter = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Strange Printer
* Difficulty: Hard
* 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 strangePrinter(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Strange Printer
* Difficulty: Hard
* 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 StrangePrinter(string s) {

}
}
```

C Solution:

```
/*
* Problem: Strange Printer
* Difficulty: Hard
* 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 strangePrinter(char* s) {

}
```

Go Solution:

```
// Problem: Strange Printer
// Difficulty: Hard
// 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 strangePrinter(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Strange Printer
// Difficulty: Hard
// 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 strange_printer(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (strange-printer s)
(-> string? exact-integer?)
)
```