Problem 1446: Consecutive Characters

Problem Information

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

Problem Description

The

power

of the string is the maximum length of a non-empty substring that contains only one unique character.

Given a string

s

, return

the

power

of

s

.

Example 1:

Input:

s = "leetcode"

Output:

2

Explanation:

The substring "ee" is of length 2 with the character 'e' only.

Example 2:

Input:

s = "abbcccddddeeeeedcba"

Output:

5

Explanation:

The substring "eeeee" is of length 5 with the character 'e' only.

Constraints:

1 <= s.length <= 500

s

consists of only lowercase English letters.

Code Snippets

C++:

```
class Solution {
public:
```

```
int maxPower(string s) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

}

C#:

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

}
}
```

C:

```
int maxPower(char* s) {

}
```

Go:

```
func maxPower(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

```
end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Consecutive Characters
* Difficulty: Easy
* Tags: string, tree
*
* 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:
int maxPower(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Consecutive Characters
* Difficulty: Easy
* Tags: string, tree
*
* 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 int maxPower(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Consecutive Characters
Difficulty: Easy
Tags: string, tree

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Consecutive Characters
* Difficulty: Easy
* Tags: string, tree
*
* 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 maxPower = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Consecutive Characters
* Difficulty: Easy
* Tags: string, tree
*
* 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 maxPower(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Consecutive Characters
* Difficulty: Easy
* Tags: string, tree
*
* 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 int MaxPower(string s) {

}
}
```

C Solution:

```
/*
* Problem: Consecutive Characters
* Difficulty: Easy
* Tags: string, tree
*
* 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
*/

int maxPower(char* s) {

}
```

Go Solution:

```
// Problem: Consecutive Characters
// Difficulty: Easy
// Tags: string, tree
//
// 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 maxPower(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Consecutive Characters
// Difficulty: Easy
// Tags: string, tree
//
// 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_power(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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