Problem 409: Longest Palindrome

Problem Information

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

Problem Description

Given a string

s

which consists of lowercase or uppercase letters, return the length of the

longest

palindrome

that can be built with those letters.

Letters are

case sensitive

, for example,

"Aa"

is not considered a palindrome.

Example 1:

Input:

s = "abccccdd"

Output:

7

Explanation:

One longest palindrome that can be built is "dccaccd", whose length is 7.

Example 2:

Input:

s = "a"

Output:

1

Explanation:

The longest palindrome that can be built is "a", whose length is 1.

Constraints:

1 <= s.length <= 2000

s

consists of lowercase

and/or

uppercase English letters only.

Code Snippets

C++:

```
class Solution {
public:
int longestPalindrome(string s) {

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int longestPalindrome(char* s) {

}
```

Go:

```
func longestPalindrome(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Longest Palindrome
* Difficulty: Easy
* Tags: string, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public:
int longestPalindrome(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Longest Palindrome
* Difficulty: Easy
* Tags: string, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

class Solution {
public int longestPalindrome(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Longest Palindrome
Difficulty: Easy
Tags: string, greedy, hash

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) for hash map
"""

class Solution:
def longestPalindrome(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Longest Palindrome
* Difficulty: Easy
* Tags: string, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Longest Palindrome
* Difficulty: Easy
* Tags: string, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

function longestPalindrome(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Longest Palindrome
* Difficulty: Easy
* Tags: string, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

public class Solution {
public int LongestPalindrome(string s) {

}
}
```

C Solution:

```
/*
* Problem: Longest Palindrome
* Difficulty: Easy
* Tags: string, greedy, hash
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) for hash map
*/

int longestPalindrome(char* s) {

}
```

Go Solution:

```
// Problem: Longest Palindrome
// Difficulty: Easy
// Tags: string, greedy, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map
```

```
func longestPalindrome(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Longest Palindrome
// Difficulty: Easy
// Tags: string, greedy, hash
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) for hash map

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

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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