Problem 2315: Count Asterisks

Problem Information

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

Problem Description

You are given a string

s

, where every

two

consecutive vertical bars

'|'

are grouped into a

pair

. In other words, the 1

st

and 2

nd

'|'

make a pair, the 3

rd

and 4

th

'|'

make a pair, and so forth.

Return

the number of

'*'

in

s

,

excluding

the

'*'

between each pair of

'|'

.

Note

that each

'|'

will belong to

exactly

one pair.

Example 1:

Input:

s = "l|*e*et|c**o|*de|"

Output:

2

Explanation:

The considered characters are underlined: "

l

|*e*et|

c**o

|*de|". The characters between the first and second '|' are excluded from the answer. Also, the characters between the third and fourth '|' are excluded from the answer. There are 2 asterisks considered. Therefore, we return 2.

Example 2:

Input:

s = "iamprogrammer"

Output:

0

Explanation:

In this example, there are no asterisks in s. Therefore, we return 0.

Example 3:

Input:

s = "yo|uar|e**|b|e***au|tifu|l"

Output:

5

Explanation:

The considered characters are underlined: "

yo

|uar|

e**

|b|

e***au

|tifu|

l

". There are 5 asterisks considered. Therefore, we return 5.

Constraints:

1 <= s.length <= 1000

s

consists of lowercase English letters, vertical bars

'|'

, and asterisks

'*'

.

s

contains an

even

number of vertical bars

'|'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int countAsterisks(char* s) {

}
```

Go:

```
func countAsterisks(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (count-asterisks s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Asterisks
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Count Asterisks
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int countAsterisks(String s) {
}
}
```

Python3 Solution:

```
"""
Problem: Count Asterisks
Difficulty: Easy
Tags: string

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Asterisks
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

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

};
```

TypeScript Solution:

```
/**
* Problem: Count Asterisks
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function countAsterisks(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Count Asterisks
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}

}
```

C Solution:

```
/*
* Problem: Count Asterisks
* Difficulty: Easy
* Tags: string
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int countAsterisks(char* s) {

}
```

Go Solution:

```
// Problem: Count Asterisks
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func countAsterisks(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Asterisks
// Difficulty: Easy
// Tags: string
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (count-asterisks s)
(-> string? exact-integer?)
)
```