Problem 3498: Reverse Degree of a String

Problem Information

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

Problem Description

Given a string

s

, calculate its

reverse degree

.

The

reverse degree

is calculated as follows:

For each character, multiply its position in the

reversed

alphabet (

'a'

= 26,

'b'

= 25, ...,

'z'

= 1) with its position in the string

(1-indexed)

.

Sum these products for all characters in the string.

Return the

reverse degree

of

s

.

Example 1:

Input:

s = "abc"

Output:

148

Explanation:

Letter

Index in Reversed Alphabet

Index in String

Product

'a'

26

1

26

'b'

25

2

50

'c'

24

3

72

The reversed degree is

26 + 50 + 72 = 148

.

Example 2:

Input:

s = "zaza"

Output:

160

Explanation:

Letter

Index in Reversed Alphabet

Index in String

Product

'z'

1

1

1

'a'

26

2

52

'z'

1

3

3

'a'

26

4

104

The reverse degree is

1 + 52 + 3 + 104 = 160

.

Constraints:

1 <= s.length <= 1000

s

contains only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
```

```
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int reverseDegree(char* s) {

}
```

Go:

```
func reverseDegree(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (reverse-degree s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reverse Degree of a String
* 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 reverseDegree(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Reverse Degree of a String
* 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 reverseDegree(String s) {

}
```

```
}
```

Python3 Solution:

```
"""
Problem: Reverse Degree of a String
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 reverseDegree(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Reverse Degree of a String
* 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 reverseDegree = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Reverse Degree of a String
* 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 reverseDegree(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Reverse Degree of a String
* 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 ReverseDegree(string s) {

}
}
```

C Solution:

```
/*
* Problem: Reverse Degree of a String
* 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 reverseDegree(char* s) {

}
```

Go Solution:

```
// Problem: Reverse Degree of a String
// 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 reverseDegree(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

```
}
}
```

Rust Solution:

```
// Problem: Reverse Degree of a String
// 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 reverse_degree(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (reverse-degree s)
(-> string? exact-integer?)
)
```