Problem 828: Count Unique Characters of All Substrings of a Given String

Problem Information

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

Problem Description

Let's define a function

countUniqueChars(s)

that returns the number of unique characters in

s

.

For example, calling

countUniqueChars(s)

if

s = "LEETCODE"

then

"L"

,

"T"

,

"C"

,

"O"

,

"D"

are the unique characters since they appear only once in

s

, therefore

countUniqueChars(s) = 5

.

Given a string

s

, return the sum of

countUniqueChars(t)

where

t

is a substring of

s

. The test cases are generated such that the answer fits in a 32-bit integer.

Notice that some substrings can be repeated so in this case you have to count the repeated ones too.

Example 1:

Input:

s = "ABC"

Output:

10

Explanation:

All possible substrings are: "A","B","C","AB","BC" and "ABC". Every substring is composed with only unique letters. Sum of lengths of all substring is 1 + 1 + 1 + 2 + 2 + 3 = 10

Example 2:

Input:

s = "ABA"

Output:

8

Explanation:

The same as example 1, except

countUniqueChars

("ABA") = 1.

Example 3:

Input:

s = "LEETCODE"

Output:

92

Constraints:

1 <= s.length <= 10

5

s

consists of uppercase English letters only.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int uniqueLetterString(char* s) {

}
```

Go:

```
func uniqueLetterString(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

/**
* @param String $s
* @return Integer
```

```
*/
function uniqueLetterString($s) {

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (unique-letter-string s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Count Unique Characters of All Substrings of a Given String
* Difficulty: Hard
* Tags: string, tree, dp, hash
*
* 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 uniqueLetterString(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Count Unique Characters of All Substrings of a Given String
* Difficulty: Hard
* Tags: string, tree, dp, hash
*
* 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 uniqueLetterString(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Count Unique Characters of All Substrings of a Given String
Difficulty: Hard
Tags: string, tree, dp, hash

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Count Unique Characters of All Substrings of a Given String
* Difficulty: Hard
* Tags: string, tree, dp, hash
*
* 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 uniqueLetterString = function(s) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Count Unique Characters of All Substrings of a Given String
* Difficulty: Hard
* Tags: string, tree, dp, hash
*
* 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 uniqueLetterString(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Count Unique Characters of All Substrings of a Given String
* Difficulty: Hard
* Tags: string, tree, dp, hash
*
* 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 UniqueLetterString(string s) {

}
}
```

C Solution:

```
/*
* Problem: Count Unique Characters of All Substrings of a Given String
* Difficulty: Hard
```

```
* Tags: string, tree, dp, hash
*
* 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 uniqueLetterString(char* s) {

}
```

Go Solution:

```
// Problem: Count Unique Characters of All Substrings of a Given String
// Difficulty: Hard
// Tags: string, tree, dp, hash
//
// 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 uniqueLetterString(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Count Unique Characters of All Substrings of a Given String
// Difficulty: Hard
// Tags: string, tree, dp, hash
//
// 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 unique_letter_string(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

```
}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (unique-letter-string s)
(-> string? exact-integer?)
)
```