

Problem 2262: Total Appeal of A String

Problem Information

Difficulty: Hard

Acceptance Rate: 0.00%

Paid Only: No

Problem Description

The

appeal

of a string is the number of

distinct

characters found in the string.

For example, the appeal of

"abbca"

is

3

because it has

3

distinct characters:

'a'

,

'b'

, and

'c'

Given a string

s

, return

the

total appeal of all of its

substrings

.

A

substring

is a contiguous sequence of characters within a string.

Example 1:

Input:

s = "abbca"

Output:

Explanation:

The following are the substrings of "abbca": - Substrings of length 1: "a", "b", "b", "c", "a" have an appeal of 1, 1, 1, 1, and 1 respectively. The sum is 5. - Substrings of length 2: "ab", "bb", "bc", "ca" have an appeal of 2, 1, 2, and 2 respectively. The sum is 7. - Substrings of length 3: "abb", "bbc", "bca" have an appeal of 2, 2, and 3 respectively. The sum is 7. - Substrings of length 4: "abbc", "bbca" have an appeal of 3 and 3 respectively. The sum is 6. - Substrings of length 5: "abbca" has an appeal of 3. The sum is 3. The total sum is $5 + 7 + 7 + 6 + 3 = 28$.

Example 2:

Input:

s = "code"

Output:

20

Explanation:

The following are the substrings of "code": - Substrings of length 1: "c", "o", "d", "e" have an appeal of 1, 1, 1, and 1 respectively. The sum is 4. - Substrings of length 2: "co", "od", "de" have an appeal of 2, 2, and 2 respectively. The sum is 6. - Substrings of length 3: "cod", "ode" have an appeal of 3 and 3 respectively. The sum is 6. - Substrings of length 4: "code" has an appeal of 4. The sum is 4. The total sum is $4 + 6 + 6 + 4 = 20$.

Constraints:

$1 \leq s.length \leq 10$

5

s

consists of lowercase English letters.

Code Snippets

C++:

```
class Solution {  
public:  
    long long appealSum(string s) {  
  
    }  
};
```

Java:

```
class Solution {  
public long appealSum(String s) {  
  
}  
}
```

Python3:

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

Python:

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

JavaScript:

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

TypeScript:

```
function appealSum(s: string): number {  
}  
};
```

C#:

```
public class Solution {  
    public long AppealSum(string s) {  
  
    }  
}
```

C:

```
long long appealSum(char* s) {  
  
}
```

Go:

```
func appealSum(s string) int64 {  
  
}
```

Kotlin:

```
class Solution {  
    fun appealSum(s: String): Long {  
  
    }  
}
```

Swift:

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

Rust:

```
impl Solution {  
    pub fn appeal_sum(s: String) -> i64 {  
        }  
    }  
}
```

Ruby:

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

PHP:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return Integer  
     */  
    function appealSum($s) {  
  
    }  
}
```

Dart:

```
class Solution {  
    int appealSum(String s) {  
  
    }  
}
```

Scala:

```
object Solution {  
    def appealSum(s: String): Long = {  
  
    }  
}
```

Elixir:

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

Erlang:

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

Racket:

```
(define/contract (appeal-sum s)
  (-> string? exact-integer?))
```

Solutions

C++ Solution:

```
/*
 * Problem: Total Appeal of A 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:
  long long appealSum(string s) {
    }
};
```

Java Solution:

```
/**  
 * Problem: Total Appeal of A 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 long appealSum(String s) {  
        // Implementation  
    }  
}
```

Python3 Solution:

```
"""  
Problem: Total Appeal of A 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 appealSum(self, s: str) -> int:  
        # TODO: Implement optimized solution  
        pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**  
 * Problem: Total Appeal of A 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 appealSum = function(s) {  
  
};
```

TypeScript Solution:

```
/**  
 * Problem: Total Appeal of A 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 appealSum(s: string): number {  
  
};
```

C# Solution:

```

/*
 * Problem: Total Appeal of A 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 long AppealSum(string s) {
        return 0;
    }
}

```

C Solution:

```

/*
 * Problem: Total Appeal of A 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
 */

long long appealSum(char* s) {
    return 0;
}

```

Go Solution:

```

// Problem: Total Appeal of A 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 appealSum(s string) int64 {  
    }  
}
```

Kotlin Solution:

```
class Solution {  
    fun appealSum(s: String): Long {  
        }  
    }  
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Total Appeal of A 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 appeal_sum(s: String) -> i64 {  
        }  
    }  
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {  
  
    /**  
     * @param String $s  
     * @return Integer  
     */  
    function appealSum($s) {  
  
    }  
}
```

Dart Solution:

```
class Solution {  
int appealSum(String s) {  
  
}  
}
```

Scala Solution:

```
object Solution {  
def appealSum(s: String): Long = {  
  
}  
}
```

Elixir Solution:

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

Erlang Solution:

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

Racket Solution:

```
(define/contract (appeal-sum s)  
  (-> string? exact-integer?)  
  )
```