Problem 3138: Minimum Length of Anagram Concatenation

Problem Information

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

Problem Description

You are given a string

s

, which is known to be a concatenation of

anagrams

of some string

t

.

Return the

minimum

possible length of the string

t

.

An

anagram

is formed by rearranging the letters of a string. For example, "aab", "aba", and, "baa" are anagrams of "aab".

Example 1:

Input:

s = "abba"

Output:

2

Explanation:

One possible string

t

could be

"ba"

.

Example 2:

Input:

s = "cdef"

Output:

4

Explanation:

One possible string

t

could be

"cdef"

, notice that

t

can be equal to

s

.

Example 2:

Input:

s = "abcbcacabbaccba"

Output:

3

Constraints:

1 <= s.length <= 10

5

s

consist only of lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minAnagramLength(char* s) {

}
```

Go:

```
func minAnagramLength(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (min-anagram-length s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Length of Anagram Concatenation
* Difficulty: Medium
* Tags: string, 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 minAnagramLength(string s) {
}
};
```

Java Solution:

```
/**
* Problem: Minimum Length of Anagram Concatenation
* Difficulty: Medium
* Tags: string, 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 minAnagramLength(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Length of Anagram Concatenation
Difficulty: Medium
Tags: string, 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 minAnagramLength(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Length of Anagram Concatenation
* Difficulty: Medium
* Tags: string, 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 minAnagramLength = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Length of Anagram Concatenation
* Difficulty: Medium
* Tags: string, 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 minAnagramLength(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Length of Anagram Concatenation
* Difficulty: Medium
* Tags: string, 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 MinAnagramLength(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Length of Anagram Concatenation
* Difficulty: Medium
* Tags: string, 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 minAnagramLength(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Length of Anagram Concatenation
// Difficulty: Medium
// Tags: string, 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 minAnagramLength(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Length of Anagram Concatenation
// Difficulty: Medium
// Tags: string, 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 min_anagram_length(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
```

```
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (min-anagram-length s)
(-> string? exact-integer?)
)
```