Problem 3389: Minimum Operations to Make Character Frequencies Equal

Problem Information

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

Problem Description

You are given a string

s

.

A string

t

is called

good

if all characters of

t

occur the same number of times.

You can perform the following operations

any number of times

:

Delete a character from

s

.

Insert a character in

s

.

Change a character in

s

to its next letter in the alphabet.

Note

that you cannot change

'z'

to

'a'

using the third operation.

Return

the

minimum

number of operations required to make

s

good

.

Example 1:

Input:

s = "acab"

Output:

1

Explanation:

We can make

s

good by deleting one occurrence of character

'a'

.

Example 2:

Input:

s = "wddw"

Output:

0

Explanation:

We do not need to perform any operations since

s

is initially good.

Example 3:

Input:

s = "aaabc"

Output:

2

Explanation:

We can make

s

good by applying these operations:

Change one occurrence of

'a'

to

'b'

Insert one occurrence of

'c'

into

s

Constraints:

3 <= s.length <= 2 * 10

4

s

contains only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

```
class Solution(object):
def makeStringGood(self, s):
```

```
"""
:type s: str
:rtype: int
"""
```

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int makeStringGood(char* s) {

}
```

Go:

```
func makeStringGood(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (make-string-good s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Minimum Operations to Make Character Frequencies Equal
* Difficulty: Hard
* Tags: string, 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 makeStringGood(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Operations to Make Character Frequencies Equal
* Difficulty: Hard
* Tags: string, 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 makeStringGood(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Operations to Make Character Frequencies Equal
Difficulty: Hard
Tags: string, 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 makeStringGood(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Operations to Make Character Frequencies Equal
* Difficulty: Hard
* Tags: string, 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 makeStringGood = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Operations to Make Character Frequencies Equal
* Difficulty: Hard
* Tags: string, 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 makeStringGood(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Operations to Make Character Frequencies Equal
* Difficulty: Hard
* Tags: string, 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 MakeStringGood(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Operations to Make Character Frequencies Equal
* Difficulty: Hard
* Tags: string, 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 makeStringGood(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Operations to Make Character Frequencies Equal
// Difficulty: Hard
// Tags: string, 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 makeStringGood(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum Operations to Make Character Frequencies Equal
// Difficulty: Hard
// Tags: string, 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 make_string_good(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (make-string-good s)
(-> string? exact-integer?)
)
```