Problem 1647: Minimum Deletions to Make Character Frequencies Unique

Problem Information

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

Problem Description

A string

s

is called

good

if there are no two different characters in

s

that have the same

frequency

.

Given a string

s

, return

the

minimum

number of characters you need to delete to make

s

good

.

The

frequency

of a character in a string is the number of times it appears in the string. For example, in the string

"aab"

, the

frequency

of

'a'

is

2

, while the

frequency

of

'b'

is

1

.

Example 1:

Input:

s = "aab"

Output:

0

Explanation:

s

is already good.

Example 2:

Input:

s = "aaabbbcc"

Output:

2

Explanation:

You can delete two 'b's resulting in the good string "aaabcc". Another way it to delete one 'b' and one 'c' resulting in the good string "aaabbc".

Example 3:

Input:

s = "ceabaacb"

Output:

2

Explanation:

You can delete both 'c's resulting in the good string "eabaab". Note that we only care about characters that are still in the string at the end (i.e. frequency of 0 is ignored).

Constraints:

1 <= s.length <= 10

5

s

contains only lowercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

```
class Solution {
public int minDeletions(String s) {
}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minDeletions(char* s) {

}
```

Go:

```
func minDeletions(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum Deletions to Make Character Frequencies Unique
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* 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 minDeletions(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum Deletions to Make Character Frequencies Unique
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* 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 minDeletions(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum Deletions to Make Character Frequencies Unique
Difficulty: Medium
Tags: string, greedy, hash, sort

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum Deletions to Make Character Frequencies Unique
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* 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 minDeletions = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimum Deletions to Make Character Frequencies Unique
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* 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 minDeletions(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum Deletions to Make Character Frequencies Unique
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* 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 MinDeletions(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum Deletions to Make Character Frequencies Unique
* Difficulty: Medium
* Tags: string, greedy, hash, sort
*
* 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 minDeletions(char* s) {

}
```

Go Solution:

```
// Problem: Minimum Deletions to Make Character Frequencies Unique
// Difficulty: Medium
// Tags: string, greedy, hash, sort
//
// 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 minDeletions(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

Rust Solution:

```
// Problem: Minimum Deletions to Make Character Frequencies Unique
// Difficulty: Medium
// Tags: string, greedy, hash, sort
//
// 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_deletions(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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