Problem 2716: Minimize String Length

Problem Information

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

Problem Description

Given a string

s

, you have two types of operation:

Choose an index

i

in the string, and let

c

be the character in position

i

.

Delete

the

closest occurrence

of

c

to the

left

of

i

(if exists).

Choose an index

i

in the string, and let

c

be the character in position

i

.

Delete

the

closest occurrence

of

c

to the

right

of

i

(if exists).

Your task is to

minimize

the length of

s

by performing the above operations zero or more times.

Return an integer denoting the length of the

minimized

string.

Example 1:

Input:

s = "aaabc"

Output:

3

Explanation:

Operation 2: we choose

i = 1

so

c

is 'a', then we remove

s[2]

as it is closest 'a' character to the right of

s[1]

.

s

becomes "aabc" after this.

Operation 1: we choose

i = 1

so

c

is 'a', then we remove

s[0]

as it is closest 'a' character to the left of

s[1]

.

s

becomes "abc" after this.

Example 2:

Input:

s = "cbbd"

Output:

3

Explanation:

Operation 1: we choose

i = 2

so

c

is 'b', then we remove

s[1]

as it is closest 'b' character to the left of

s[1]

.

s

becomes "cbd" after this.

Example 3:

Input:

s = "baadccab"

Output:

4

Explanation:

Operation 1: we choose

i = 6

so

c

is 'a', then we remove

s[2]

as it is closest 'a' character to the left of

s[6]

.

s

becomes "badccab" after this.

Operation 2: we choose

i = 0

so

c

is 'b', then we remove

s[6]

as it is closest 'b' character to the right of

s[0]

.

s

becomes "badcca" fter this.

Operation 2: we choose

i = 3

so

c

is 'c', then we remove

s[4]

as it is closest 'c' character to the right of

s[3]

.

s

becomes "badca" after this.

Operation 1: we choose

i = 4

so

c

is 'a', then we remove

s[1]

as it is closest 'a' character to the left of

s[4]

.

s

becomes "bdca" after this.

Constraints:

1 <= s.length <= 100

s

contains only lowercase English letters

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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


}
}
```

C:

```
int minimizedStringLength(char* s) {

}
```

Go:

```
func minimizedStringLength(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimize String Length
* Difficulty: Easy
* 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 minimizedStringLength(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimize String Length
* Difficulty: Easy
* 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 minimizedStringLength(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimize String Length
Difficulty: Easy
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 minimizedStringLength(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimize String Length
* Difficulty: Easy
* 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 minimizedStringLength = function(s) {

};
```

TypeScript Solution:

```
/**
* Problem: Minimize String Length
* Difficulty: Easy
* 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 minimizedStringLength(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimize String Length
* Difficulty: Easy
* 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 MinimizedStringLength(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimize String Length
* Difficulty: Easy
* 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 minimizedStringLength(char* s) {

}
```

Go Solution:

```
// Problem: Minimize String Length
// Difficulty: Easy
// 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 minimizedStringLength(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimize String Length
// Difficulty: Easy
// 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 minimized_string_length(s: String) -> i32 {

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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