Problem 2696: Minimum String Length After Removing Substrings

Problem Information

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

Problem Description

You are given a string

s

consisting only of

uppercase

English letters.

You can apply some operations to this string where, in one operation, you can remove

any

occurrence of one of the substrings

"AB"

or

"CD"

from

s

.

Return

the

minimum

possible length of the resulting string that you can obtain

.

Note

that the string concatenates after removing the substring and could produce new

"AB"

or

"CD"

substrings.

Example 1:

Input:

s = "ABFCACDB"

Output:

2

Explanation:

We can do the following operations: - Remove the substring "

AB

FCACDB", so s = "FCACDB". - Remove the substring "FCA

CD

B", so s = "FCAB". - Remove the substring "FC

AB

", so s = "FC". So the resulting length of the string is 2. It can be shown that it is the minimum length that we can obtain.

Example 2:

Input:

s = "ACBBD"

Output:

5

Explanation:

We cannot do any operations on the string so the length remains the same.

Constraints:

1 <= s.length <= 100

s

consists only of uppercase English letters.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

C#:

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

}
}
```

C:

```
int minLength(char* s) {

}
```

Go:

```
func minLength(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum String Length After Removing Substrings
* Difficulty: Easy
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public:
int minLength(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum String Length After Removing Substrings
* Difficulty: Easy
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

class Solution {
public int minLength(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum String Length After Removing Substrings
Difficulty: Easy
Tags: string, tree, stack

Approach: String manipulation with hash map or two pointers
Time Complexity: O(n) or O(n log n)
Space Complexity: O(h) for recursion stack where h is height
"""

class Solution:
def minLength(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Minimum String Length After Removing Substrings
* Difficulty: Easy
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Minimum String Length After Removing Substrings
* Difficulty: Easy
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

function minLength(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum String Length After Removing Substrings
* Difficulty: Easy
* Tags: string, tree, stack
*
```

```
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

public class Solution {
public int MinLength(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum String Length After Removing Substrings
* Difficulty: Easy
* Tags: string, tree, stack
*
* Approach: String manipulation with hash map or two pointers
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(h) for recursion stack where h is height
*/

int minLength(char* s) {

}
```

Go Solution:

```
// Problem: Minimum String Length After Removing Substrings
// Difficulty: Easy
// Tags: string, tree, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

func minLength(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum String Length After Removing Substrings
// Difficulty: Easy
// Tags: string, tree, stack
//
// Approach: String manipulation with hash map or two pointers
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(h) for recursion stack where h is height

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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