Problem 3746: Minimum String Length After Balanced Removals

Problem Information

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

Problem Description

You are given a string

s

consisting only of the characters

'a'

and

'b'

.

You are allowed to repeatedly remove

any

substring

where the number of

'a'

characters is equal to the number of

'b'

characters. After each removal, the remaining parts of the string are concatenated together without gaps.

Return an integer denoting the

minimum possible length

of the string after performing any number of such operations.

Example 1:

Input:

s =

"aabbab"

Output:

0

Explanation:

The substring

"aabbab"

has three

'a'

and three

'b'

. Since their counts are equal, we can remove the entire string directly. The minimum length is 0.

Example 2:

Input:

s =

"aaaa"

Output:

4

Explanation:

Every substring of

"aaaa"

contains only

'a'

characters. No substring can be removed as a result, so the minimum length remains 4.

Example 3:

Input:

s =

"aaabb"

Output:

1

Explanation:

First, remove the substring

"ab"

, leaving

"aab"

. Next, remove the new substring

"ab"

, leaving

"a"

. No further removals are possible, so the minimum length is 1.

Constraints:

1 <= s.length <= 10

5

s[i]

is either

'a'

or

'b'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

}
}
```

C:

```
int minLengthAfterRemovals(char* s) {

}
```

Go:

```
func minLengthAfterRemovals(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

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

Solutions

C++ Solution:

```
/*
* Problem: Minimum String Length After Balanced Removals
* Difficulty: Medium
* 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 minLengthAfterRemovals(string s) {

}
};
```

Java Solution:

```
/**
* Problem: Minimum String Length After Balanced Removals
* Difficulty: Medium
* 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 minLengthAfterRemovals(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Minimum String Length After Balanced Removals
Difficulty: Medium
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 minLengthAfterRemovals(self, s: str) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```javascript
/**
* Problem: Minimum String Length After Balanced Removals
* Difficulty: Medium
* 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 minLengthAfterRemovals = function(s) {

};
```

TypeScript Solution:

```typescript
/**
* Problem: Minimum String Length After Balanced Removals
* Difficulty: Medium
* 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 minLengthAfterRemovals(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Minimum String Length After Balanced Removals
* Difficulty: Medium
* 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 MinLengthAfterRemovals(string s) {

}
}
```

C Solution:

```
/*
* Problem: Minimum String Length After Balanced Removals
* Difficulty: Medium
* 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 minLengthAfterRemovals(char* s) {

}
```

Go Solution:

```
// Problem: Minimum String Length After Balanced Removals
// Difficulty: Medium
// 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 minLengthAfterRemovals(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Minimum String Length After Balanced Removals
// Difficulty: Medium
// 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_after_removals(s: String) -> i32 {

}
}
```

Ruby Solution:

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

```
end
```

PHP Solution:

```
class Solution {

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

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