Problem 1332: Remove Palindromic Subsequences

Problem Information

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

Problem Description

You are given a string

s

consisting

only

of letters

'a'

and

'b'

. In a single step you can remove one

palindromic subsequence

from

s

.

Return

the

minimum

number of steps to make the given string empty

.

A string is a

subsequence

of a given string if it is generated by deleting some characters of a given string without changing its order. Note that a subsequence does

not

necessarily need to be contiguous.

A string is called

palindrome

if is one that reads the same backward as well as forward.

Example 1:

Input:

s = "ababa"

Output:

1

Explanation:

s is already a palindrome, so its entirety can be removed in a single step.

Example 2:

Input:

s = "abb"

Output:

2

Explanation:

"

a

bb" -> "

bb

" -> "". Remove palindromic subsequence "a" then "bb".

Example 3:

Input:

s = "baabb"

Output:

2

Explanation:

"

baa

b

b

" -> "

b

" -> "". Remove palindromic subsequence "baab" then "b".

Constraints:

1 <= s.length <= 1000

s[i]

is either

'a'

or

'b'

.

Code Snippets

C++:

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

}
};
```

Java:

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

}
}
```

Python3:

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

Python:

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

JavaScript:

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

};
```

TypeScript:

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

};
```

C#:

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

```
}
}
```

C:

```
int removePalindromeSub(char* s) {

}
```

Go:

```
func removePalindromeSub(s string) int {

}
```

Kotlin:

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

}
}
```

Swift:

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

}
}
```

Rust:

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

}
}
```

Ruby:

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

end
```

PHP:

```
class Solution {

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

}
}
```

Dart:

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

}
}
```

Scala:

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

}
}
```

Elixir:

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

end
end
```

Erlang:

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

Racket:

```
(define/contract (remove-palindrome-sub s)
(-> string? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Remove Palindromic Subsequences
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

}
};
```

Java Solution:

```
/**
* Problem: Remove Palindromic Subsequences
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
```

```
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int removePalindromeSub(String s) {

}
}
```

Python3 Solution:

```
"""
Problem: Remove Palindromic Subsequences
Difficulty: Easy
Tags: array, string

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

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

Python Solution:

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

JavaScript Solution:

```
/**
* Problem: Remove Palindromic Subsequences
* Difficulty: Easy
```

```
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

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

};
```

TypeScript Solution:

```
/**
* Problem: Remove Palindromic Subsequences
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function removePalindromeSub(s: string): number {

};
```

C# Solution:

```
/*
* Problem: Remove Palindromic Subsequences
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
```

```
*/

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

}
}
```

C Solution:

```
/*
* Problem: Remove Palindromic Subsequences
* Difficulty: Easy
* Tags: array, string
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

int removePalindromeSub(char* s) {

}
```

Go Solution:

```
// Problem: Remove Palindromic Subsequences
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func removePalindromeSub(s string) int {

}
```

Kotlin Solution:

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

}
}
```

Swift Solution:

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

}
}
```

Rust Solution:

```
// Problem: Remove Palindromic Subsequences
// Difficulty: Easy
// Tags: array, string
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

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

}
}
```

Ruby Solution:

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

end
```

PHP Solution:

```
class Solution {
```

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

}
}
```

Dart Solution:

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

}
}
```

Scala Solution:

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

}
}
```

Elixir Solution:

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

end
end
```

Erlang Solution:

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

Racket Solution:

```
(define/contract (remove-palindrome-sub s)
(-> string? exact-integer?)
)
```