1332. Remove Palindromic Subsequences

08 June 2022 05:49 PM

1332. Remove Palindromic Subsequences

Easy 816 979 Add to List Share

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.

```
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: "abb" -> "bb" -> "".
Remove palindromic subsequence "a" then "bb".
```

Example 3:

```
Input: s = "baabb"
Output: 2
Explanation: "baabb" -> "b" -> "".
Remove palindromic subsequence "baab" then "b".
```

Constraints:

- `1 <= s.length <= 1000`
- `s[i]` is either `'a'` or `'b'`.

a a a a

(0)

a b a b a

(1)

b a a b b

b a a b b (2) b a a b b ⇒ So here we took 2 steps to convert into empty string

```java
class Solution {
    public int removePalindromeSub(String s) {
        if(s.isBlank()) return 0;

        if(isPalindrome(s)){
            return 1;
        }
        return 2;
    }

    boolean isPalindrome(String s){
        int left = 0, right = s.length() - 1;

        while(left < right){
            if(s.charAt(left) != s.charAt(right)){
                return false;
            }
            left++;
            right--;
        }
        return true;
    }
}
```

```java
class Solution {
    public int removePalindromeSub(String s) {
        if (s.equals(new StringBuilder(s).reverse().toString())) return 1;
        return 2;
    }
}
```

Logic

It's quite easy to trip over this problem since it looks like we have to do all sorts of stuff to the string to get our answer. Actually, once you identify the main trick behind this question, it's really quite a simple algorithm!

What is a Subsequence?

This is quite important to understand. Below are the main string sub-types (literally) that could be mentioned in a question.

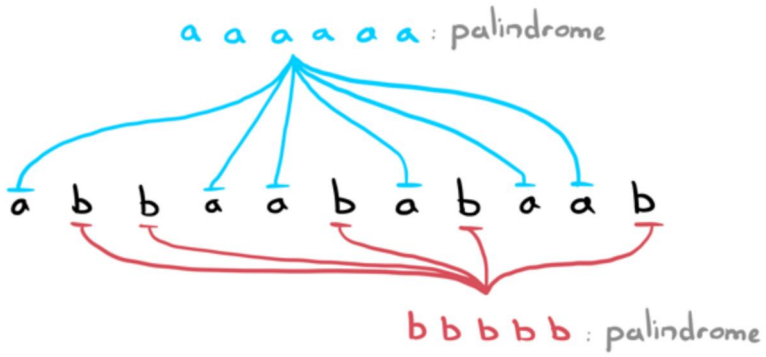
- **Substring:** A *contiguous* sequence of characters in a string.
- **Subsequence:** Any sequence of characters in a string with their *relative order is maintained*.

In particular, this question is asking to remove *subsequences* in the string. Therefore, we don't actually care where these subsequences are as long as they maintain their original relative ordering in the string. For example, a subsequence of "leetcode" could be "toe".

The Major Observation:

What makes a subsequence a palindrome? Well of course it's when the string is the same forwards and backwards. Is "a" a palindrome? Yes. Is "aa" a palindrome? Yes. What about "aaaaaaaa"? Yes!

> The main observation here is that any string consisting of the same letters is a palindrome. Since we're working with subsequences and the only characters in the string are 'a' and 'b', we know we can get rid of all palindromes in **at most 2 steps**.



What other cases are there? Well when could we ever remove a palindrome in just 1 step? When the input string itself is a palindrome of course. Awesome. Now we're ready to start coding!

Algorithm:

```
If string is a palindrome: return 1
Else return 2
```

> Solution with StringBuilder:

```
public int removePalindromeSub(String s) {
    if (s.equals(new StringBuilder(s).reverse().toString())) return 1;
    return 2;
}
```

Time Complexity: `O(n)` where `n` is the length of the string.
Space Complexity: `O(1)` (note that the StringBuilder solution takes `O(n)` time).