Description | Solution | Discuss (156) | Submissions

1216. Valid Palindrome III

Hard 366 6 Add to List Share

Given a string `s` and an integer `k`, return `true` if `s` is a `k` **-palindrome**.

A string is `k` **-palindrome** if it can be transformed into a palindrome by removing at most `k` characters from it.

Example 1:

```
Input: s = "abcdeca", k = 2
Output: true
Explanation: Remove 'b' and 'e' characters.
```

Example 2:

```
Input: s = "abbababa", k = 1
Output: true
```

Constraints:

- `1 <= s.length <= 1000`
- `s` consists of only lowercase English letters.
- `1 <= k <= s.length`

Accepted 22,969 | Submissions 44,687

Seen this question in a real interview before? Yes No

Companies

0 ~ 6 months 6 months ~ 1 year 1 year ~ 2 years

Facebook | 9

Related Topics

String Dynamic Programming

Hide Hint 1

Can you reduce this problem to a classic problem?

Hide Hint 2

The problem is equivalent to finding any palindromic subsequence of length at least N-K where N is the length of the string.

Hide Hint 3

Try to find the longest palindromic subsequence.

Hide Hint 4

Use DP to do that.

Java | Autocomplete

```java
class Solution {
    Integer memo[][];
    int isValidPalindrome(String s, int i, int j) {

        // Base case, only 1 letter remaining.
        if (i == j)
            return 0;

        // Base case 2, only 2 letters remaining.
        if (i == j - 1)
            return s.charAt(i) != s.charAt(j) ? 1 : 0;

        //Return the precomputed value if exists.
        if (memo[i][j] != null)
            return memo[i][j];

        // Case 1: Character at `i` equals character at `j`
        if (s.charAt(i) == s.charAt(j))
            return memo[i][j] = isValidPalindrome(s, i + 1, j - 1);

        // Case 2: Character at `i` does not equal character at `j`.
        // Either delete character at `i` or delete character at `j`
        // and try to match the two pointers using recursion.
        // We need to take the minimum of the two results and add 1
        // representing the cost of deletion.
        return memo[i][j] = 1 + Math.min(isValidPalindrome(s, i + 1, j), isValidPalindrome(s, i, j - 1));
    }
public boolean isValidPalindrome(String s, int k) {
        memo = new Integer[s.length()][s.length()];

        // Return true if the minimum cost to create a palindrome by only deleting the letters
        // is less than or equal to `k`.
        return isValidPalindrome(s, 0, s.length() - 1) <= k;
    }
};
```