**Problem Statement**

In the “Minimum Characters to be Added at Front to Make String Palindrome” problem we have given a [string](https://www.tutorialcup.com/interview/string/) “s”. Write a program to find the minimum characters to be added at the front to make a string palindrome.

**Input Format**

The first and only one line containing a string**“s”**.

**Output Format**

The first and only one line containing an integer value **n**. Here n denotes the minimum characters to be added at the front to make a string palindrome.

**Constraints**

* 1<=|s|<=10^6
* s[i] must be a lower case English alphabet

**Example**

edef

1

**Explanation:**If we add “f” at the front of string “s” then our string satisfied the condition of the palindrome. So, here only 1 should be added in the front.

**Algorithm**

**1.** Concatenate the string with a special symbol and reverse of the given string ie, c = s + $ + rs

**2.** Build an LPS array in which each index represents the longest proper prefix which is also a suffix.

**3.** As the last value in the LPS array is the largest suffix that matches the prefix of the original string. So, there are those many characters that satisfy the palindrome property

**4.** Now, find the difference between the length of the string and the last value in the LPS array, which is the minimum number of characters needed to make a string palindrome.

**Working of above example**

s = “edef”

Example for proper prefixes and suffix(Knowledge required to form LPS)

Proper prefixes of “abc” are  ” “, “a”, “ab” and Suffixes of “abc” are “c”, “bc”, “abc”.

s (after concatenation)= “edef$fede”

LPS = {0, 0, 1, 0, 0, 0, 1, 2, 3 } //Here index is the longest length of largest prefix that matches suffix.

The last value of LPS = 3, so there are 3 characters that satisfy palindrome property.

Now, find the difference between the length of a given string(ie, 4) and the Last value(3)

Therefore, we need 1 character to make it a palindrome.

### [Time Complexity](https://en.wikipedia.org/wiki/Time_complexity)

**O(n)** where **n** is the size of the given string **“s”**. Here we find the “lps array of KMP algorithm” which takes linear time to compute.

### [Space Complexity](https://en.wikipedia.org/wiki/Space_complexity)

**O(n)** because we create an LSP array to compute our answer. And here the maximum size of the LSP array is **n**.