Form a palindrome

Tuesday, December 14, 2021 10:15 AM

Form a palindrome
Medium Accuracy: 53.0% Submissions: 12345 Points: 4

Given a string, find the minimum number of characters to be inserted to convert it to palindrome.

For Example:

ab: Number of insertions required is 1. **b**ab or aba

aa: Number of insertions required is 0. aa

abcd: Number of insertions required is 3. **dcb**abcd

Example 1:

Input: str = "abcd"
Output: 3
Explanation: Inserted character marked with bold characters in **dcb**abcd
Example 2:

Input: str = "aa"
Output: 0
Explanation:"aa" is already a palindrome.

Your Task:

You don't need to read input or print anything. Your task is to complete the function **countMin()** which takes the string **str** as inputs and returns the answer.

Expected Time Complexity: $O(N^2)$, N = |str|

Expected Auxiliary Space: $O(N^2)$

Constraints:

$1 \leq |str| \leq 10^3$

str contains only lower case alphabets

From <https://practice.geeksforgeeks.org/problems/form-a-palindrome1455/1>

Company Tags

- Airtel

- Amazon
- 

- Google


From <https://practice.geeksforgeeks.org/problems/form-a-palindrome1455/1>

Minimum characters to be added at front to make string palindrome

- Difficulty Level : Hard
- Last Updated : 13 Sep, 2021

Given a string str we need to tell minimum characters to be added at front of string to make string palindrome.

Examples:

```
Input  : str = "ABC"
Output : 2
We can make above string palindrome as "CBABC"
by adding 'B' and 'C' at front.
Input  : str = "AACECAAAA";
Output : 2
We can make above string palindrome as AAAACECAAAA
by adding two A's at front of string.
```


Recommended: Please try your approach on ***{IDE}*** first, before moving on to the solution.

Naive approach: Start checking the string each time if it is a palindrome and if not, then delete the last character and check again. When the string gets reduced to wither a palindrome or empty then the number of characters deleted from the end till now will be the answer as those characters could have been inserted at the beginning of the original string in the order which will will make the string a

palindrome.

Below is the implementation of the above approach:

- C++
- Java
- Python 3
- C#
- Javascript

```java
// Java program for getting minimum character to be
// added at front to make string palindrome

class GFG {

// function for checking string is palindrome or not
    static boolean ispalindrome(String s) {
        int l = s.length();

        for (int i = 0, j = l - 1; i <= j; i++, j--) {
            if (s.charAt(i) != s.charAt(j)) {
                return false;
            }
        }
        return true;
    }

// Driver code
    public static void main(String[] args) {
        String s = "BABABAA";
        int cnt = 0;
        int flag = 0;

        while (s.length() > 0) {
            // if string becomes palindrome then break
            if (ispalindrome(s)) {
                flag = 1;
                break;
            } else {
                cnt++;

                // erase the last element of the string
                s = s.substring(0, s.length() - 1);
                //s.erase(s.begin() + s.length() - 1);
            }
        }

        // print the number of insertion at front
        if (flag == 1) {
            System.out.println(cnt);
        }
    }
}

// This code is contributed by 29AjayKumar
```

Output:

2

Thank you **Sanny Kumar** for suggesting this approach.

Efficient approach: We can solve this problem **efficiently in O(n) time** using lps array of KMP algorithm.

First we concat string by concatenating given string, a special character and reverse of given string then we will get lps array for this concatenated string, recall that each index of lps array represent longest proper prefix which is also suffix. We can use this lps array for solving the problem.

```
For string = AACECAAAA
Concatenated String = AACECAAAA$AAAACECAA
LPS array will be {0, 1, 0, 0, 0, 1, 2, 2, 2,
                   0, 1, 2, 2, 2, 3, 4, 5, 6, 7}
```

Here we are only interested in the last value of this lps array because it shows us the largest suffix of the reversed string that matches the prefix of the original string i.e these many characters already satisfy the palindrome property. Finally minimum number of character needed to make the string a palindrome is length of the input string minus last entry of our lps array. Pease see below code for better understanding

- C++
- Java
- Python3
- C#

```java
// Java program for getting minimum character to be
// added at front to make string palindrome
import java.util.*;
class GFG
{

// returns vector lps for given string str
public static int[] computeLPSArray(String str)
{
    int n = str.length();
    int lps[] = new int[n];
    int i = 1, len = 0;
    lps[0] = 0; // lps[0] is always 0

    while (i < n)
    {
        if (str.charAt(i) == str.charAt(len))
        {
            len++;
            lps[i] = len;
            i++;
        }
        else
        {
            // This is tricky. Consider the example.
            // AAACAAAA and i = 7. The idea is similar
```

```
            // to search step.
            if (len != 0)
            {
                len = lps[len - 1];

                // Also, note that we do not increment
                // i here
            }
            else
            {
                lps[i] = 0;
                i++;
            }
        }
    }
    return lps;
}

// Method returns minimum character to be added at
// front to make string palindrome
static int getMinCharToAddedToMakeStringPalin(String str)
{
    StringBuilder s = new StringBuilder();
    s.append(str);

    // Get concatenation of string, special character
    // and reverse string
    String rev = s.reverse().toString();
    s.reverse().append("$").append(rev);

    // Get LPS array of this concatenated string
    int lps[] = computeLPSArray(s.toString());
    return str.length() - lps[s.length() - 1];
}

// Driver Code
public static void main(String args[])
{
    String str = "AACECAAAA";
    System.out.println(getMinCharToAddedToMakeStringPalin(str));
}
}

// This code is contributed by Sparsh Singhal
```

Output:

2

From <https://www.geeksforgeeks.org/minimum-characters-added-front-make-string-palindrome/>

Minimum insertions to form a palindrome | DP-28

- Difficulty Level : Hard
- Last Updated : 09 Nov, 2021

Given string **str**, the task is to find the minimum number of characters to be inserted to convert it to a palindrome.

Before we go further, let us understand with a few examples:

- **ab:** Number of insertions required is 1 i.e. **b**ab
- **aa:** Number of insertions required is 0 i.e. aa
- **abcd:** Number of insertions required is 3 i.e. **dcb**abcd
- **abcda:** Number of insertions required is 2 i.e. a**dc**bcda which is the same as the number of insertions in the substring bcd(Why?).
- **abcde:** Number of insertions required is 4 i.e. **edcb**abcde

Recommended: Please solve it on "***PRACTICE***" first, before moving on to the solution.

Let the input string be *str[l……h]*. The problem can be broken down into three parts:

1. Find the minimum number of insertions in the substring str[l+1,…….h].
2. Find the minimum number of insertions in the substring str[l…….h-1].
3. Find the minimum number of insertions in the substring str[l+1……h-1].

Recursive Approach: The minimum number of insertions in the string **str[l…..h]** can be given as:

- minInsertions(str[l+1…..h-1]) if str[l] is equal to str[h]
- min(minInsertions(str[l…..h-1]), minInsertions(str[l+1…..h])) + 1 otherwise

Below is the implementation of the above approach:

- C++
- C
- Java
- Python 3
- C#
- Javascript

```
// A Naive recursive Java program to find minimum
// number insertions needed to make a string
// palindrome
class GFG {

    // Recursive function to find minimum number
    // of insertions
    static int findMinInsertions(char str[], int l,
                                                int h)
    {
        // Base Cases
        if (l > h) return Integer.MAX_VALUE;
        if (l == h) return 0;
        if (l == h - 1) return (str[l] == str[h])? 0 : 1;

        // Check if the first and last characters
        // are same. On the basis of the  comparison
        // result, decide which subrpoblem(s) to call
        return (str[l] == str[h])?
            findMinInsertions(str, l + 1, h - 1):
            (Integer.min(findMinInsertions(str, l, h - 1),
            findMinInsertions(str, l + 1, h)) + 1);
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        String str= "geeks";
        System.out.println(findMinInsertions(str.toCharArray(),
                                            0, str.length()-1));
    }
}
// This code is contributed by Sumit Ghosh
```

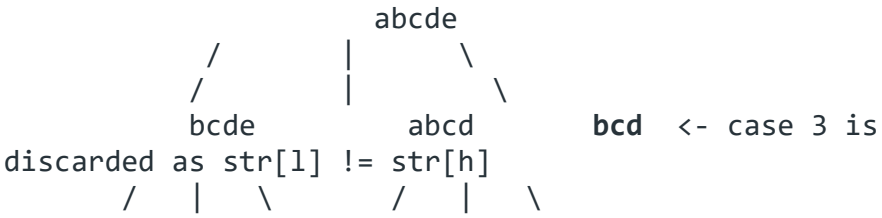
Output:

```
3
```

Dynamic Programming based Solution

If we observe the above approach carefully, we can find that it exhibits overlapping subproblems.

Suppose we want to find the minimum number of insertions in string “abcde”:



```
      /    |    \     /    |    \
     cde   bcd  cd   bcd abc bc
   / | \  / | \ /|\ / | \
de cd d cd bc c.....
```

The substrings in bold show that the recursion is to be terminated and the recursion tree cannot originate from there. Substring in the same color indicates overlapping subproblems.

How to re-use solutions of subproblems? The memorization technique is used to avoid similar subproblem recalls. We can create a table to store the results of subproblems so that they can be used directly if the same subproblem is encountered again.

The below table represents the stored values for the string abcde.

```
a b c d e
----------
0 1 2 3 4
0 0 1 2 3
0 0 0 1 2
0 0 0 0 1
0 0 0 0 0
```

How to fill the table?

The table should be filled in a diagonal fashion. For the string abcde, 0….4, the following should be ordered in which the table is filled:

```
Gap = 1: (0, 1) (1, 2) (2, 3) (3, 4)
Gap = 2: (0, 2) (1, 3) (2, 4)

Gap = 3: (0, 3) (1, 4)

Gap = 4: (0, 4)
```

Below is the implementation of the above approach:

- C++
- C
- Java
- Python3
- C#
- Javascript

```java
// A Java solution for Dynamic Programming
// based program to find minimum number
// insertions needed to make a string
// palindrome
import java.util.Arrays;

class GFG
{
    // A DP function to find minimum number
    // of insertions
    static int findMinInsertionsDP(char str[], int n)
    {
        // Create a table of size n*n. table[i][j]
        // will store minimum number of insertions
```

```
        // needed to convert str[i..j] to a palindrome.
        int table[][] = new int[n][n];
        int l, h, gap;

        // Fill the table
        for (gap = 1; gap < n; ++gap)
        for (l = 0, h = gap; h < n; ++l, ++h)
            table[l][h] = (str[l] == str[h])?
                           table[l+1][h-1] :
                          (Integer.min(table[l][h-1],
                                 table[l+1][h]) + 1);

        // Return minimum number of insertions
        // for str[0..n-1]
        return table[0][n-1];
    }

    // Driver program to test above function.
    public static void main(String args[])
    {
        String str = "geeks";
        System.out.println(
        findMinInsertionsDP(str.toCharArray(), str.length()));
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

3

Time complexity: O(N^2)

Auxiliary Space: O(N^2)

Another Dynamic Programming Solution (Variation of Longest Common Subsequence Problem)

The problem of finding minimum insertions can also be solved using Longest Common Subsequence (LCS) Problem. If we find out the LCS of string and its reverse, we know how many maximum characters can form a palindrome. We need to insert the remaining characters. Following are the steps.

1. Find the length of LCS of the input string and its reverse. Let the length be 'l'.
2. The minimum number of insertions needed is the length of the input string minus 'l'.
   Below is the implementation of the above approach:

- C++
- C
- Java
- Python3
- C#
- Javascript

```
// An LCS based Java program to find minimum
// number insertions needed to make a string
// palindrome
class GFG
```

```
{
    /* Returns length of LCS for X[0..m-1],
       Y[0..n-1]. See http://goo.gl/bHQVP for
       details of this function */
    static int lcs( String X, String Y, int m, int n )
    {
        int L[][] = new int[m+1][n+1];
        int i, j;

        /* Following steps build L[m+1][n+1] in
           bottom up fashion. Note that L[i][j]
           contains length of LCS of X[0..i-1]
           and Y[0..j-1] */
        for (i=0; i<=m; i++)
        {
            for (j=0; j<=n; j++)
            {
                if (i == 0 || j == 0)
                    L[i][j] = 0;

                else if (X.charAt(i-1) == Y.charAt(j-1))
                    L[i][j] = L[i-1][j-1] + 1;

                else
                    L[i][j] = Integer.max(L[i-1][j], L[i][j-1]);
            }
        }

        /* L[m][n] contains length of LCS for
           X[0..n-1] and Y[0..m-1] */
        return L[m][n];
    }

    // LCS based function to find minimum number
    // of insertions
    static int findMinInsertionsLCS(String str, int n)
    {
        // Using StringBuffer to reverse a String
        StringBuffer sb = new StringBuffer(str);
        sb.reverse();
        String revString = sb.toString();

        // The output is length of string minus
        // length of lcs of str and it reverse
        return (n - lcs(str, revString , n, n));
    }

    // Driver program to test above functions
    public static void main(String args[])
    {
        String str = "geeks";
        System.out.println(
        findMinInsertionsLCS(str, str.length()));
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

From <https://www.geeksforgeeks.org/minimum-insertions-to-form-a-palindrome-dp-28/>