Permutations of A in B

Problem Description

You are given two strings, **A** and **B,** of size **N** and **M,** respectively.

You have to find the **count of all permutations of A present in B as a substring**. You can assume a string will have only lowercase letters.

Problem Constraints

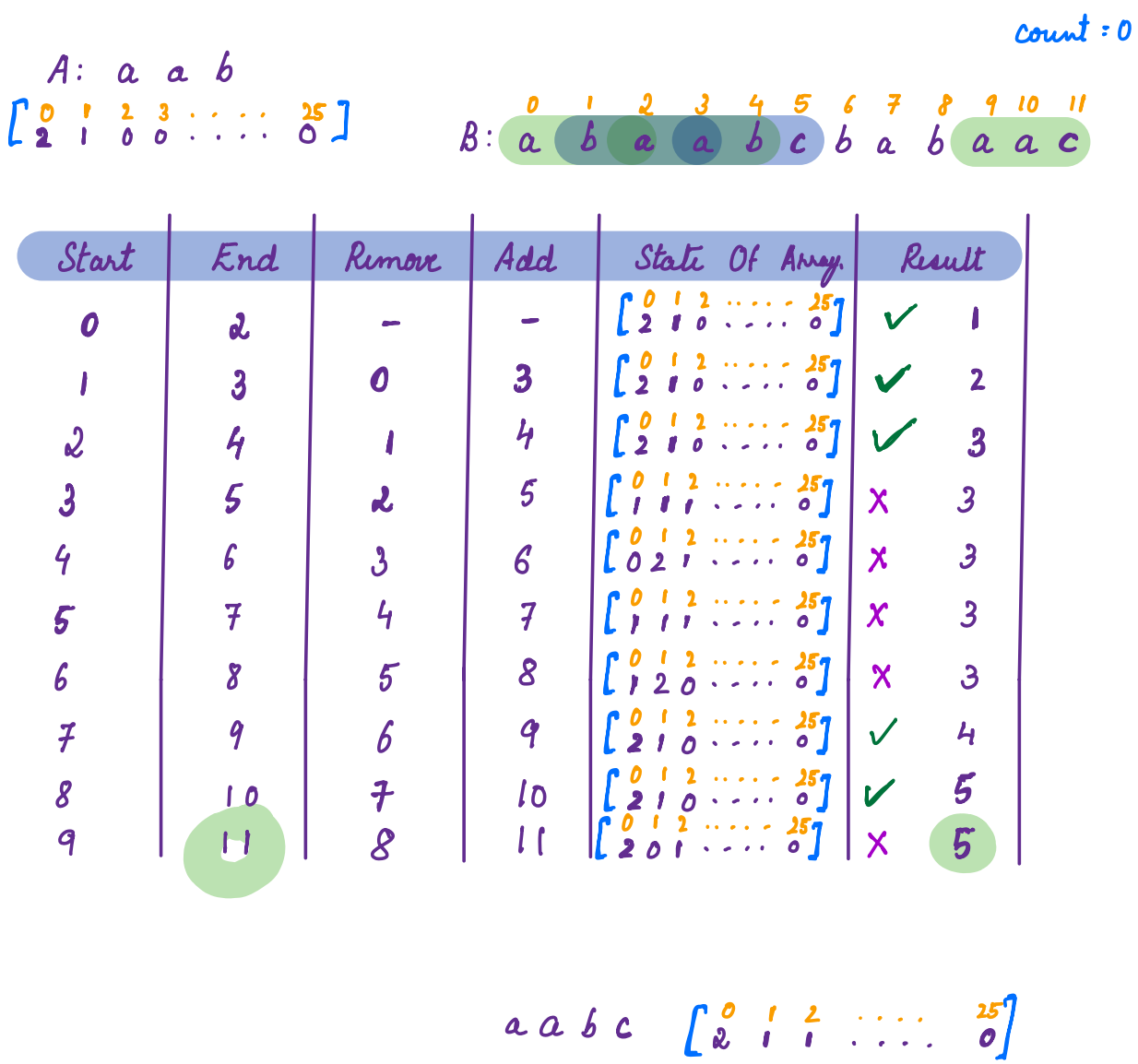
$1 <= N < M <= 10^5$

Input Format

Given two arguments, A and B of type String.

Output Format

Return a single integer, i.e., number of permutations of A present in B as a substring.



Code:

b a a [0 1 2 .... 25] = [0 0 0 .... 0]

'a'

```
int isAnagram(String A, String B)
{ int[] fA = new int[26];
  int[] fB = new int[26];
  for(i:0; i< A.length(); i++)
  { fA[A.charAt(i) - 'a']++;}
  for(i:0; i< B.length(); i++)
  { fB[B.charAt(i) - 'a']++;}
}
```

a → 0
b → 1
c → 2
:
z → 25

```
int count(String A, String B)
{
```

Longest Palindromic Substring

Problem Description

Given a string A of size N, find and return the **longest palindromic substring** in A.

Substring of string A is `A[i...j]` where 0 <= i <= j < len(A)

Palindrome string:
A string which reads the same backwards. More formally, A is palindrome if reverse(A) = A.

Incase of conflict, return the substring which occurs first (with the least starting index).

Problem Constraints

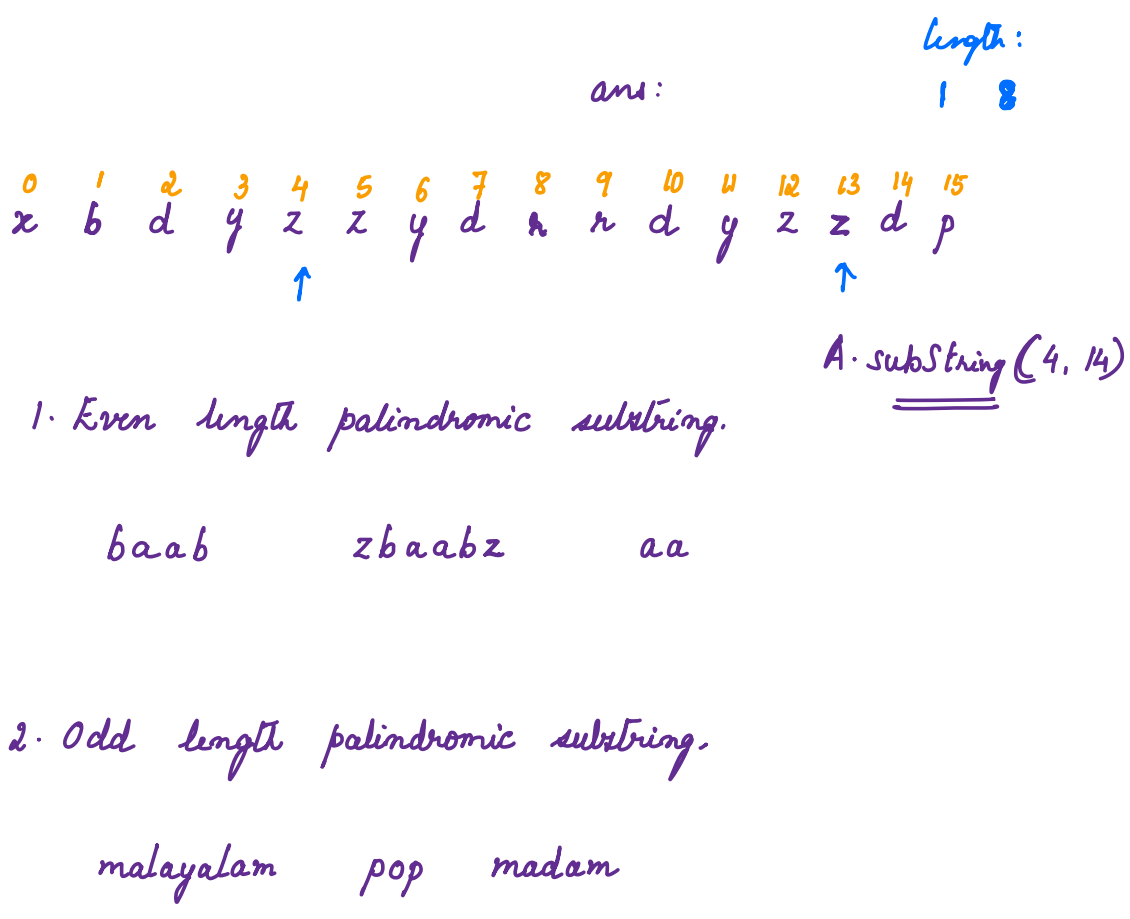
1 <= N <= 6000

Input Format

First and only argument is a string A.

Output Format

Return a string denoting the longest palindromic substring of string A.



3. Expansion from the middle.