

Problem D

Late Failure

Time limit: 1s

Memory limit: 256MB

A string is called a *palindrome* if it reads the same from left to right and from right to left.

To check whether a string is a palindrome, a programmer uses the following C++ function:

```
bool isPalindrome(string s) {  
    int low = 0;  
    int high = s.size() - 1;  
  
    while (low < high) {  
        if (s[low] != s[high])  
            return false;  
        ++low;  
        --high;  
    }  
    return true;  
}
```

You are given a string S consisting of uppercase English letters. You may rearrange the letters of S in any order you like (keeping the original order is also allowed) to obtain a new string T .

You want `isPalindrome(T)` to return `false`, but you also want the function to detect this as late as possible.

Find the largest number of iterations of the `while` loop that can be executed before the function returns `false`. The iteration in which the function returns `false` is counted, so this is the number of times the condition `s[low] != s[high]` is checked. If no rearrangement makes the function return `false`, print `-1` instead.

Input

- The only line contains the string S ($2 \leq |S| \leq 2 \cdot 10^5$). It consists only of uppercase English letters from A to Z.

Output

Print a single integer: the largest number of iterations described above, or `-1` if no rearrangement makes the function return `false`.

Sample Input	Sample Output
AABBCDEF	3
AAAAA	-1

Notes

In the first example, the letters can be arranged as `ABCDEFBA`. The first two comparisons both find two equal letters (A and A, then B and B). At the third comparison the function meets two different letters, C and F, and returns `false`. So the loop is executed 3 times.

In the second example, every rearrangement gives the same string `AAAAA`, which is a palindrome, so the function can never return `false`.