<https://leetcode.com/problems/longest-palindrome/description/>

Given a string s which consists of lowercase or uppercase letters, return the length of the **longest palindrome**

that can be built with those letters.

Letters are **case sensitive**, for example, "Aa" is not considered a palindrome.

**Example 1:**

**Input:** s = "abccccdd"

**Output:** 7

**Explanation:** One longest palindrome that can be built is "dccaccd", whose length is 7.

**Example 2:**

**Input:** s = "a"

**Output:** 1

**Explanation:** The longest palindrome that can be built is "a", whose length is 1.

**Constraints:**

1 <= s.length <= 2000

s consists of lowercase **and/or** uppercase English letters only.

**Attempt 1: 2025-01-16**

**Solution 1: Hash Table (10 min)**

class Solution {

    public int longestPalindrome(String s) {

        int oddCount = 0;

        Map<Character, Integer> freq = new HashMap<>();

        for(char c : s.toCharArray()) {

            freq.put(c, freq.getOrDefault(c, 0) + 1);

            if(freq.get(c) % 2 == 1) {

                oddCount++;

            } else {

                oddCount--;

            }

        }

        if(oddCount > 1) {

            return s.length() - oddCount + 1;

        }

        return s.length();

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/longest-palindrome/solutions/3156147/c-easiest-beginner-friendly-sol-o-n-time-and-o-128-o-1-space/>

Initialize two variables, oddCount to store the number of characters with an odd count of occurrences and an unordered map ump to store the count of each character in the string.

Iterate through the string and for each character ch, increment the count of that character in the unordered map.

If the count of the current character ch is odd, increment oddCount. If the count is even, decrement oddCount.

If oddCount is greater than 1, return s.length() - oddCount + 1, which is the maximum length of a palindrome that can be formed by using all but one character with an odd count of occurrences.

If oddCount is not greater than 1, return s.length(), which is the length of the original string, as all characters can be used to form a palindrome.

class Solution {

public int longestPalindrome(String s) {

int oddCount = 0;

Map<Character, Integer> map = new HashMap<>();

for (char ch : s.toCharArray()) {

map.put(ch, map.getOrDefault(ch, 0) + 1);

if (map.get(ch) % 2 == 1)

oddCount++;

else

oddCount--;

}

if (oddCount > 1)

return s.length() - oddCount + 1;

return s.length();

}

}

<https://leetcode.com/problems/longest-palindrome/solutions/89604/simple-hashset-solution-java/>

public int longestPalindrome(String s) {

if(s==null || s.length()==0) return 0;

HashSet<Character> hs = new HashSet<Character>();

int count = 0;

for(int i=0; i<s.length(); i++){

if(hs.contains(s.charAt(i))){

hs.remove(s.charAt(i));

count++;

}else{

hs.add(s.charAt(i));

}

}

if(!hs.isEmpty()) return count\*2+1;

return count\*2;

}