

Longest Palindrome

Given a string which consists of lowercase or uppercase letters, find the length of the longest palindromes that can be built with those letters.

This is case sensitive, for example `"Aa"` is not considered a palindrome here.

Note:

Assume the length of given string will not exceed 1,010.

Example:

Input:
`"abcccd"`

Output:
7

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

Solution 1

```
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;  
}
```

written by [charliebobo604gmail.com](mailto:charliebobo604@gmail.com) original link [here](#)

Solution 2

I count how many letters appear an odd number of times. Because we can use **all** letters, except for each odd-count letter we must leave one, except one of them we can use.

Python:

```
def longestPalindrome(self, s):  
    odds = sum(v & 1 for v in collections.Counter(s).values())  
    return len(s) - odds + bool(odds)
```

C++:

```
int longestPalindrome(string s) {  
    int odds = 0;  
    for (char c='A'; c<='Z'; c++)  
        odds += count(s.begin(), s.end(), c) & 1;  
    return s.size() - odds + (odds > 0);  
}
```

Similar solutions (I actually like the **use** solutions better than the above, but I'm just so fond of my topic title :-)

```

def longestPalindrome(self, s):
    use = sum(v & ~1 for v in collections.Counter(s).values())
    return use + (use < len(s))

def longestPalindrome(self, s):
    counts = collections.Counter(s).values()
    return sum(v & ~1 for v in counts) + any(v & 1 for v in counts)

int longestPalindrome(string s) {
    int use = 0;
    for (char c='A'; c<='Z'; c++)
        use += count(s.begin(), s.end(), c) & ~1;
    return use + (use < s.size());
}

int longestPalindrome(string s) {
    vector<int> count(256);
    for (char c : s)
        ++count[c];
    int odds = 0;
    for (int c : count)
        odds += c & 1;
    return s.size() - odds + (odds > 0);
}

int longestPalindrome(string s) {
    vector<int> count(256);
    int odds = 0;
    for (char c : s)
        odds += ++count[c] & 1 ? 1 : -1;
    return s.size() - odds + (odds > 0);
}

```

written by [StefanPochmann](#) original link [here](#)

Solution 3

```
class Solution {
public:
    int longestPalindrome(string s) {
        vector<int> m(256, 0);
        for (auto& c : s) m[c - '\0']++;
        int result = 0;
        for (auto& i : m) result += i%2 ? (result%2 ? i-1 : i) : i;
        return result;
    }
};
```

written by [vesion](#) original link [here](#)

From [LeetCoder](#).