Palindrome Permutation

Given a string, determine if a permutation of the string could form a palindrome.

For example,
`"code"` -> False, `"aab"` -> True, `"carerac"` -> True.

1. Consider the palindromes of odd vs even length. What difference do you notice?
2. Count the frequency of each character.
3. If each character occurs even number of times, then it must be a palindrome. How about character which occurs odd number of times?

Solution 1

The idea is to iterate over string, adding current character to `set` if `set` doesn't contain that character, or removing current character from `set` if `set` contains it. When the iteration is finished, just return `set.size()==0 || set.size()==1`.

`set.size()==0` corresponds to the situation when there are even number of any character in the string, and `set.size()==1` corresponsds to the fact that there are even number of any character except one.

```java
public class Solution {
    public boolean canPermutePalindrome(String s) {
        Set<Character> set=new HashSet<Character>();
        for(int i=0; i<s.length(); ++i){
            if (!set.contains(s.charAt(i)))
                set.add(s.charAt(i));
            else
                set.remove(s.charAt(i));
        }
        return set.size()==0 || set.size()==1;
    }
}
```

written by ammv original link here

Solution 2

Just check that no more than one character appears an odd number of times. Because if there is one, then it must be in the middle of the palindrome. So we can't have two of them.

Python

First count all characters in a `Counter`, then count the odd ones.

```
def canPermutePalindrome(self, s):
    return sum(v % 2 for v in collections.Counter(s).values()) < 2
```

Ruby

Using an integer as a bitset (Ruby has arbitrarily large integers).

```
def can_permute_palindrome(s)
  x = s.chars.map { |c| 1 << c.ord }.reduce(0, :^)
  x & x-1 == 0
end
```

C++

Using a bitset.

```
bool canPermutePalindrome(string s) {
    bitset<256> b;
    for (char c : s)
        b.flip(c);
    return b.count() < 2;
}
```

C

Tricky one. Increase `odds` when the increased counter is odd, decrease it otherwise.

```
bool canPermutePalindrome(char* s) {
    int ctr[256] = {}, odds = 0;
    while (*s)
        odds += ++ctr[*s++] & 1 ? 1 : -1;
    return odds < 2;
}
```

Thanks to jianchao.li.fighter for pointing out a nicer way in the comments to which I switched now because it's clearer and faster. Some speed test results (see comments for details):

```
        odds += ++ctr[*s++] % 2 * 2 - 1;       // 1499 ms mean-of-five (my origin
al)
        odds += (ctr[*s++] ^= 1) * 2 - 1;      // 1196 ms mean-of-five
        odds += ++ctr[*s++] % 2 ? 1 : -1;      // 1108 ms mean-of-five
        odds += ((++ctr[*s++] & 1) << 1) - 1;  // 1217 ms mean-of-five
        odds += ++ctr[*s++] & 1 ? 1 : -1;      // 1132 ms mean-of-five
```

Java

Using a BitSet.

```java
public boolean canPermutePalindrome(String s) {
    BitSet bs = new BitSet();
    for (byte b : s.getBytes())
        bs.flip(b);
    return bs.cardinality() < 2;
}
```

written by StefanPochmann original link here

Solution 3

Explanation

The basic idea is using HashSet to find the number of single characters, which should be at most 1.

```
public boolean canPermutePalindrome(String s) {
    Set<Character>set = new HashSet<Character>();
    for (char c : s.toCharArray())
        if (set.contains(c)) set.remove(c);// If char already exists in set, then
remove it from set
        else set.add(c);// If char doesn't exists in set, then add it to set
    return set.size() <= 1;
}
```

written by Pixel_ original link here

From Leetcoder.