真实面试案例分析(上)
与面试评价标准

主讲人 令狐冲

最长回文子串
Longest Palindromic Substring

http://www.lintcode.com/problem/longest-palindromic-substring/

求一个字符串中最长的回文子串

回文串定义为：aba, abba 这样的正反都一样的字符串

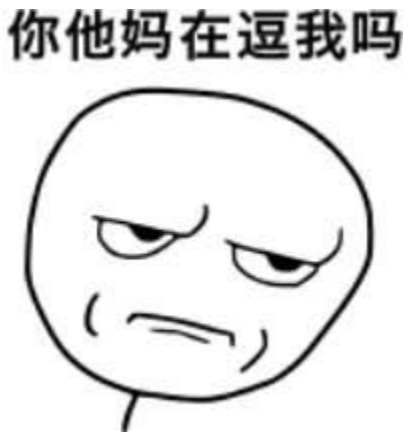
常见错误:我知道有个算法叫 Manacher's Algorithm

该算法可以在 O(n) 的时间内求得最长回文子串，n=字符串长度

这是该问题的最优算法，却绝对不是面试官想让实现的算法

为什么？

因为他自己也不会



大家来找茬 I


```python
class Solution:
    """
    @param s: input string
    @return: the longest palindromic substring
    """
    def longestPalindrome(self, s):
        for length in range(len(s), 0, -1):
            for i in range(len(s) - length + 1):
                l,r=i,i+length-1
                while l<r and s[l] == s[r]:
                    l+=1
                    r-=1
                if l>=r:
                    return s[i:i+length]
        return ""
```

```java
public class Solution {
    /**
     * @param s: input string
     * @return: the longest palindromic substring
     */
    public String longestPalindrome(String s) {
        for(int len=s.length();len >= 1;len--){
            for(int i=0;i+len<=s.length();i++){
                int l=i,r=i+len-1;
                while(l<r&&s.charAt(l)==s.charAt(r)){
                    l++;
                    r--;
                }
                if(l>=r){
                    return s.substring(i,i+len);
                }
            }
        }
        return "";
    }
}
```

O(n^3) 的实现方法下比较好的 Coding Quality 是什么样的


```python
class Solution:
    """
    @param s: input string
    @return: the longest palindromic substring
    """
    def longestPalindrome(self, s):
        if s is None:
            return None

        for length in range(len(s), 0, -1):
            for i in range(len(s) - length + 1):
                if self.is_palindrome(s, i, i + length - 1):
                    return s[i: i + length]

        return ""

    def is_palindrome(self, s, left, right):
        while left < right and s[left] == s[right]:
            left += 1
            right -= 1

        return left >= right
```

```java
public class Solution {
    /**
     * @param s: input string
     * @return: the longest palindromic substring
     */
    public String longestPalindrome(String s) {
        if (s == null) {
            return null;
        }

        for (int length = s.length(); length > 0; length--) {
            for (int start = 0; start + length <= s.length(); start++) {
                if (isPalindrome(s, start, start + length - 1)) {
                    return s.substring(start, start + length);
                }
            }
        }

        return "";
    }

    private boolean isPalindrome(String s, int left, int right) {
        while (left < right && s.charAt(left) == s.charAt(right)) {
            left++;
            right--;
        }
        return left >= right;
    }
}
```

大家来找茬 II


```python
class Solution:
    """
    @param s: input string
    @return: the longest palindromic substring
    """
    def longestPalindrome(self, s):
        if not s:
            return ""

        start, longest = 0, 0
        for middle in range(len(s)):
            # odd
            left, right = middle, middle
            while left >= 0 and right < len(s) and s[left] == s[right]:
                left -= 1
                right += 1
            if longest < right - left - 1:
                longest = right - left - 1
                start = left + 1

            # even
            left, right = middle, middle + 1
            while left >= 0 and right < len(s) and s[left] == s[right]:
                left -= 1
                right += 1
            if longest < right - left - 1:
                longest = right - left - 1
                start = left + 1

        return s[start:start + longest]
```

```java
public class Solution {
    /**
     * @param s: input string
     * @return: the longest palindromic substring
     */
    public String longestPalindrome(String s) {
        if (s == null) {
            return null;
        }

        int longest = 0, start = 0;
        for (int i = 0; i < s.length(); i++) {
            int left, right;
            // odd
            left = i; right = i;
            while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
                left--;
                right++;
            }
            if (longest < right - left - 1) {
                longest = right - left - 1;
                start = left + 1;
            }

            // even
            left = i; right = i + 1;
            while (left >= 0 && right < s.length() && s.charAt(left) == s.charAt(right)) {
                left--;
                right++;
            }
            if (longest < right - left - 1) {
                longest = right - left - 1;
                start = left + 1;
            }
        }

        return s.substring(start, start + longest);
    }
}
```

基于中心点枚举法 Enumeration 的最佳实践


```python
class Solution:
    """
    @param s: input string
    @return: the longest palindromic substring
    """
    def longestPalindrome(self, s):
        if not s:
            return s

        answer = (0, 0)
        for mid in range(len(s)):
            answer = max(answer, self.get_palindrome_from(s, mid, mid))
            answer = max(answer, self.get_palindrome_from(s, mid, mid + 1))

        return s[answer[1]: answer[0] + answer[1]]

    def get_palindrome_from(self, s, left, right):
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return right - left - 1, left + 1
```

```java
public class Solution {
    /**
     * @param s: input string
     * @return: the longest palindromic substring
     */
    public String longestPalindrome(String s) {
        if (s == null) {
            return null;
        }

        String longest = "";
        for (int i = 0; i < s.length(); i++) {
            String oddPalindrome = getPalindromeFrom(s, i, i);
            if (longest.length() < oddPalindrome.length()) {
                longest = oddPalindrome;
            }

            String evenPalindrome = getPalindromeFrom(s, i, i + 1);
            if (longest.length() < evenPalindrome.length()) {
                longest = evenPalindrome;
            }
        }

        return longest;
    }

    private String getPalindromeFrom(String s, int left, int right) {
        while (left >= 0 && right < s.length()) {
            if (s.charAt(left) != s.charAt(right)) {
                break;
            }
            left--;
            right++;
        }

        return s.substring(left + 1, right);
    }
}
```

Follow up: 不能枚举中心点

基于第一种方法，如何进行优化？

基于动态规划 Dynamic Programming 的最佳实践


```python
class Solution:
    """
    @param s: input string
    @return: the longest palindromic substring
    """
    def longestPalindrome(self, s):
        if not s:
            return ""

        n = len(s)
        is_palindrome = [[False] * n for _ in range(n)]
        for i in range(n):
            is_palindrome[i][i] = True
        for i in range(1, n):
            is_palindrome[i][i - 1] = True

        start, longest = 0, 1
        for length in range(2, n + 1):
            for i in range(n - length + 1):
                j = i + length - 1
                is_palindrome[i][j] = is_palindrome[i + 1][j - 1] and s[i] == s[j]
                if is_palindrome[i][j] and length > longest:
                    longest = length
                    start = i

        return s[start:start + longest]
```

```java
public class Solution {
    /**
     * @param s: input string
     * @return: the longest palindromic substring
     */
    public String longestPalindrome(String s) {
        if (s == null) {
            return null;
        }

        int n = s.length();

        // initialize a 2d array isPalindrome
        boolean[][] isPalindrome = new boolean[n][n];
        for (int i = 0; i < n; i++) {
            isPalindrome[i][i] = true;
            if (i > 0) {
                isPalindrome[i][i - 1] = true;
            }
        }

        // use dynamic programming to check every substring is palindrome or not
        int longest = 1, start = 0;
        for (int length = 2; length <= n; length++) {
            for (int i = 0; i + length <= n; i++) {
                int j = i + length - 1;
                isPalindrome[i][j] = isPalindrome[i + 1][j - 1] && s.charAt(i) == s.charAt(j);
                if (isPalindrome[i][j] && longest < j - i + 1) {
                    start = i;
                    longest = j - i + 1;
                }
            }
        }

        return s.substring(start, start + longest);
    }
}
```

从最长回文子串中我们可以得到的面试考点

- 面试不一定会要求你用最优复杂度的算法来解决问题
  - 因此单纯只刷LC之类的OJ，容易让你产生一定要用最优解来解决这样的误区
- 代码真的不是写出来就可以过
  - 代码质量（Coding Quality）很重要
  - 好的代码质量包括：
    - Bug Free
    - 好的代码风格（Coding Style），包括变量名命名规范有意义，合理的使用空格，善用空行
    - 容易让人读懂的逻辑。要把复杂的事情用简单的方式，别把简单的事情写复杂了。
    - 没有冗余代码
    - 有边界检测和异常处理

Longest Palindromic Substring 的全部算法及时间复杂度

Manacher's Algorithm - O(n) // **学有余力**可以阅读全文并背诵

后缀数组 Suffix Array - O(n) // 完全不用学

动态规划 Dynamic Programming - O(n^2) // 必须掌握

枚举法 Enumeration - O(n^2) // 必须掌握

参考代码:

http://www.jiuzhang.com/solution/longest-palindromic-substring/

今后所有课上讲的题目的参考答案都可以在这里查询到

从 Longest Palindromic Substring 看面试的评分体系

- **Strong Hire**
- 使用 O(n) 或者 O(nlogn) 的算法实现出来 (Manacher's Algorithm or Suffix Array)，并且代码质量合格，无 Bug 或者 有很小的bug但是能自己发现并解决，无需太多提示
- **Hire**
- 能够分别使用枚举法和动态规划实现时间复杂度 O(n^2) 的算法。并且代码质量优秀，无Bug，无重复代码，无需面试官给提示
- **Weak Hire**
- 只使用了其中一种 O(n^2) 的算法实现出来，代码质量还不错，可以有一些小 Bug，面试官可以给一些小提示
- **No Hire**
- 只能想出一种 O(n^2) 的算法，但是 Bug 太多，或者需要很多提示
- **Strong No Hire**
- 连一种 O(n^2) 的算法都想不到

面试评分和 Offer 的关系

有 >= 1 个 Strong No Hire => No offer

有 >= 2 个 No hire => No offer

有 1 个 No Hire + 1 个 Weak Hire => No Offer

有 1 个 No Hire，其他都是 Hire => Offer or 加面（取决于公司招人多不多，门槛高不高）

有 1 个 Weak Hire => Offer or 加面

特殊情况：

一个 Strong Hire + 一个 Strong No Hire => 开个会一起讨论一下，通常结果是加面或者No Offer。

独孤九剑 —— 总决式

想要做到 Bug Free 最重要的是优化你的 Coding Quality

工程师的代码长什么样比脸长什么样重要

快速提高 Coding Quality 的十二个技巧

- Coding Style 相关:
  - 二元运算符两边加空格，单元运算符不加空格
  - 花括号和 for, if 之间要加空格(Java)，圆括号和 if 之间要加空格
  - 用空行分隔开不同的逻辑块
  - 逗号后面加空格
- Readability 相关
  - 函数名和变量名用1-2个单词作为名称
  - 确保一个函数内部不超过 3 层缩进（indention）
  - 多用子函数来减少入口函数的代码量
  - 多用 continue 少用 if
- Bug Free 相关
  - 不管有没有可能出问题，都要对入口函数的参数进行异常检测
  - 访问一个下标的时候，一定要确保这个下标不会越界
  - 访问一个对象的属性或者方法时，一定要确保这个对象不是空
  - 不用全局变量