

# 复杂度理论与双指针算法入门

主讲人 令狐冲

# 四个复杂度

时间复杂度 - 核心考察点

空间复杂度 - 次要考察点

编程复杂度 - 能看得懂

思维复杂度 - 能想得出

## P问题 Polynomial

$O(n)$ ,  $O(n^2)$ ,  $O(n^3)$

$O(n + m)$ ,  $O(\sqrt{n})$ ,  $O(1)$

$O(\log n)$ ,  $O(n \log n)$

## NP 问题 Nondeterministic Polynomial

$O(2^n)$ ,  $O(n^n)$ ,  $O(n!)$

# 只考虑最高项

$$O(2^N + N^2) = O(2^N)$$

$$O(N^3 + 1000N^2) = O(N^3)$$

# 不考虑常数项和系数

$$O(100N+1000) = O(N)$$

$$O(\log N) = O(\log(N^2)) = O(\log_4(N))$$

# 小练习

$O(n+m)$  和  $O(\max(n,m))$  谁更大？



# 时间复杂度为 $O(N)$ 的算法有哪些？

双指针算法、打擂台算法  
单调栈算法、单调队列算法  
等等

# 三种双指针算法

相向双指针(判断回文串)

背向双指针(最长回文串)

同向双指针



# 相向双指针的分类

## Reverse 型

- 翻转字符串
- 判断回文串

## Two Sum 型

- 两数之和
- 三数之和

## Partition 型

- 快速排序
- 颜色排序

# Valid Palindrome

<https://www.lintcode.com/problem/valid-palindrome/>

判断一个字符串忽略大小写和非法字符之后是否是一个回文串

race a car 不是回文

A man, a plan, a canal: Panama 是回文

```
1 class Solution:
2     """
3     @param s: A string
4     @return: Whether the string is a valid palindrome
5     """
6     def isPalindrome(self, s):
7         left, right = 0, len(s) - 1
8         while left < right:
9             while left < right and not self.is_valid(s[left]):
10                 left += 1
11             while left < right and not self.is_valid(s[right]):
12                 right -= 1
13             if left < right and s[left].lower() != s[right].lower():
14                 return False
15             left += 1
16             right -= 1
17         return True
18
19     def is_valid(self, char):
20         return char.isdigit() or char.isalpha()
```

## 熟练字符串的几个常用函数

- isdigit()
- isalpha()
- lower()
- upper()

```
public boolean isPalindrome(String s) {
    if (s == null) {
        return false;
    }

    int left = 0, right = s.length() - 1;
    while (left < right) {
        while (left < right && !isValid(s.charAt(left))) {
            left++;
        }
        while (left < right && !isValid(s.charAt(right))) {
            right--;
        }
        if (left < right && !isEqual(s.charAt(left), s.charAt(right))) {
            return false;
        }
        left++;
        right--;
    }

    return true;
}

private boolean isValid(char c) {
    return Character.isLetter(c) || Character.isDigit(c);
}

private boolean isEqual(char a, char b) {
    return Character.toLowerCase(a) == Character.toLowerCase(b);
}
```

## 熟练字符类的几个常用函数

- isLetter(c)
- isDigit(c)
- toLowerCase(c)
- toUpperCase(c)

# Valid Palindrome II

<https://www.lintcode.com/problem/valid-palindrome-ii/>

是否可以在去掉一个字符的情况下是一个回文串

abca 可以, 去掉b或者c

abc 无法做到

# 这份代码有什么问题？

```
public boolean validPalindrome(String s) {
    if (s == null) {
        return false;
    }

    int left = 0, right = s.length() - 1;
    while (left < right) {
        if (s.charAt(left) != s.charAt(right)) {
            break;
        }
        left++;
        right--;
    }

    if (left >= right) {
        return true;
    }

    return isPalindrome(s, left + 1, right) || isPalindrome(s, left, right - 1);
}

private boolean isPalindrome(String s, int left, int right) {
    while (left < right) {
        if (s.charAt(left) != s.charAt(right)) {
            return false;
        }
        left++;
        right--;
    }

    return true;
}
```

## 巧用子函数，避免重复代码 (Java)

Java 当需要返回多个返回值的时候  
需要新建一个类

```
class Pair {  
    int left, right;  
    public Pair(int left, int right) {  
        this.left = left;  
        this.right = right;  
    }  
}
```

```
public boolean validPalindrome(String s) {  
    if (s == null) {  
        return false;  
    }  
  
    Pair pair = findDifference(s, 0, s.length() - 1);  
    if (pair.left >= pair.right) {  
        return true;  
    }  
  
    return isPalindrome(s, pair.left + 1, pair.right)  
        || isPalindrome(s, pair.left, pair.right - 1);  
}  
  
private Pair findDifference(String s, int left, int right) {  
    while (left < right && s.charAt(left) == s.charAt(right)) {  
        left++;  
        right--;  
    }  
    return new Pair(left, right);  
}  
  
private boolean isPalindrome(String s, int left, int right) {  
    Pair pair = findDifference(s, left, right);  
    return pair.left >= pair.right;  
}
```



## 巧用子函数，避免重复代码 (Python)

Python 可以直接返回

多元组(tuple)作为函数的返回值

```
1 class Solution:
2     def validPalindrome(self, s):
3         if s is None:
4             return False
5
6         left, right = self.findDifference(s, 0, len(s) - 1)
7         if left >= right:
8             return True
9
10        return self.isPalindrome(s, left + 1, right) or\
11               self.isPalindrome(s, left, right - 1)
12
13    def isPalindrome(self, s, left, right):
14        left, right = self.findDifference(s, left, right)
15        return left >= right
16
17    def findDifference(self, s, left, right):
18        while left < right:
19            if s[left] != s[right]:
20                return left, right
21            left += 1
22            right -= 1
23        return left, right
```



# Two Sum

<https://www.lintcode.com/problem/two-sum/>

在未排序数组中，找到两个数之和等于给定的 target

nums = [6, 4, 2, 9], target = 10

你的程序应返回 4 和 6

# 哈希表的实现方法

时间复杂度  $O(n)$

空间复杂度  $O(n)$

# 排序 + 双指针

时间复杂度  $O(n \log n)$  


空间复杂度  $O(1)$

注意空间复杂度一般指额外空间复杂度  
即不包含输入和输出

Follow Up 1:

- 如果输入数据已经排序, 哪个算法更好? 

Follow Up 2:

- 如果需要返回所找的两个数在数组中的下标, 哪个算法更好? 

# 敬请期待

在后面的直播课程中  
我们还将讲解 Two Sum 的“十种”变形题  
让你彻底搞定这一类型的相向双指针