算法与数据结构体系课程

liuyubobobo

线性查找

liuyubobobo

线性查找

一个非常简单的算法

适应更多的数据类型；

如何编写正确的程序；

性能测试

复杂度分析

什么是算法

liuyubobobo

什么是算法

Algorithm 的本意：解决问题的方法

一系列解决问题的，清晰，可执行的计算机指令

什么是算法

一系列解决问题的，清晰，可执行的计算机指令

生活中也有算法

问路：如何去天安门

如何求解一元二次方程?

菜谱

什么是算法

做法

1、 猪里脊肉切细丝，加腌肉调料腌制十几分钟；
2、 绿尖椒、胡萝卜、冬笋分别切细丝，黑木耳泡软洗净切细丝备用；
3、 调好鱼香汁备用，葱、姜、蒜切末备用，泡辣椒切末备用；
4、 锅中放足量油，油六七成热时放入肉丝大火快速滑散至变白，盛出备用；
5、 锅中放少许油，放入葱、姜、蒜末炒香，放入泡辣辣末炒出红油；
6、 放入胡萝卜、冬笋、木耳翻炒2分钟，放入尖椒翻炒均匀；
7、 放入炒好的肉丝迅速翻炒均匀；
8、 倒入鱼香汁快速翻炒均匀即可。

什么是算法

一系列解决问题的，清晰，可执行的计算机指令

1. 有限性
2. 确定性：不会产生二义性
3. 可行性
4. 输入
5. 输出

什么是算法

一系列解决问题的，清晰，可执行的计算机指令

线性查找法

liuyubobobo

线性查找法

在一沓试卷中，找到属于自己的那张试卷

第 1 张：不是

第 2 张：不是

第 3 张：不是

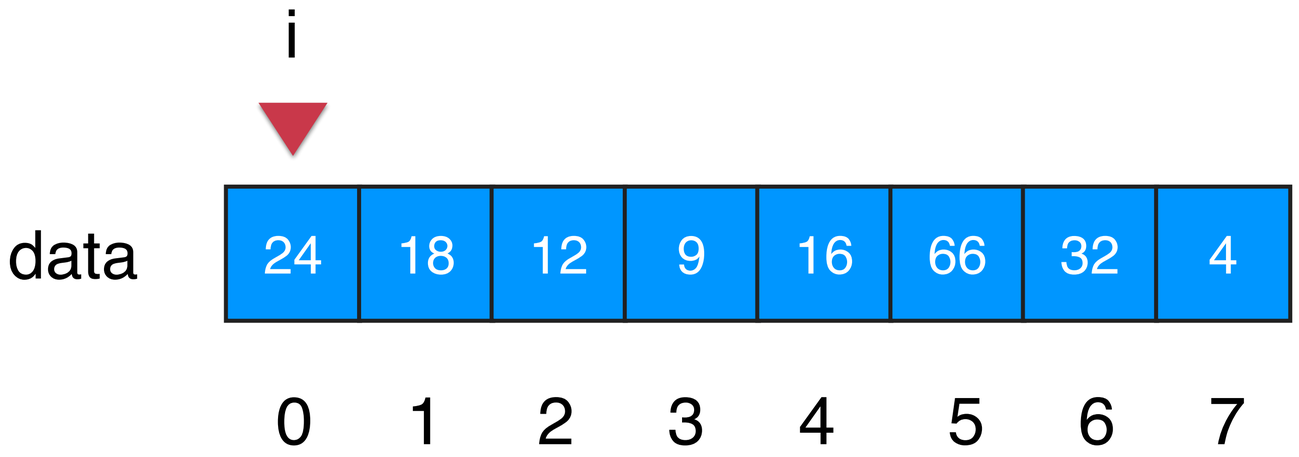
…

第 5 张：是！找到！

线性查找法

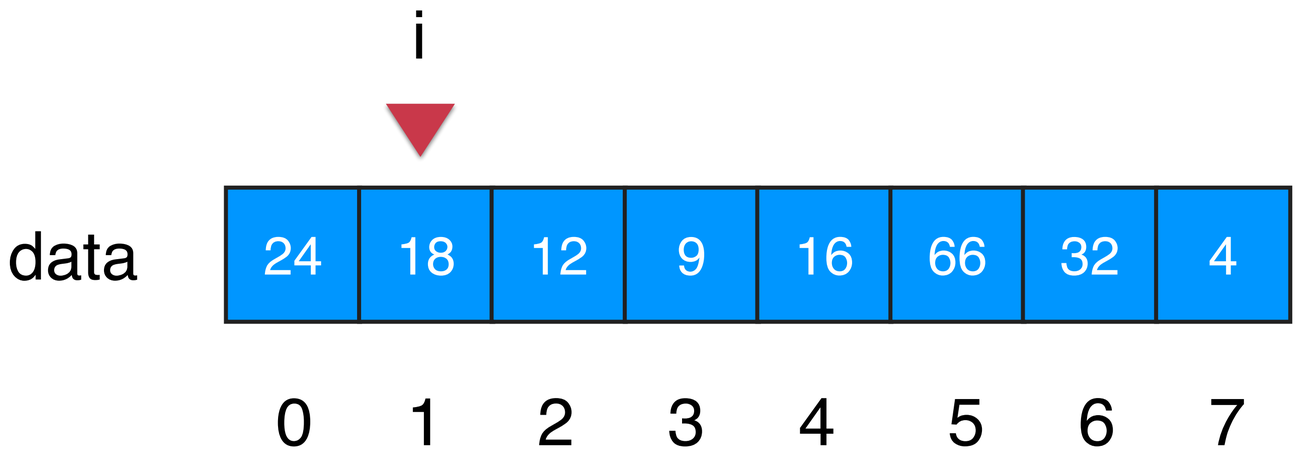
线性查找法

在 data 数组中查找 16



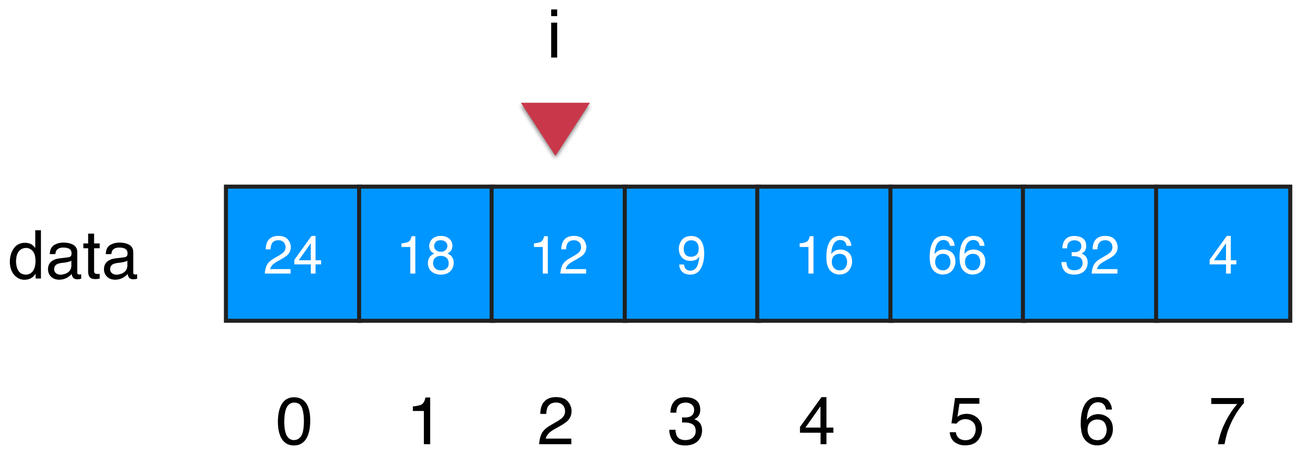
线性查找法

在 data 数组中查找 16



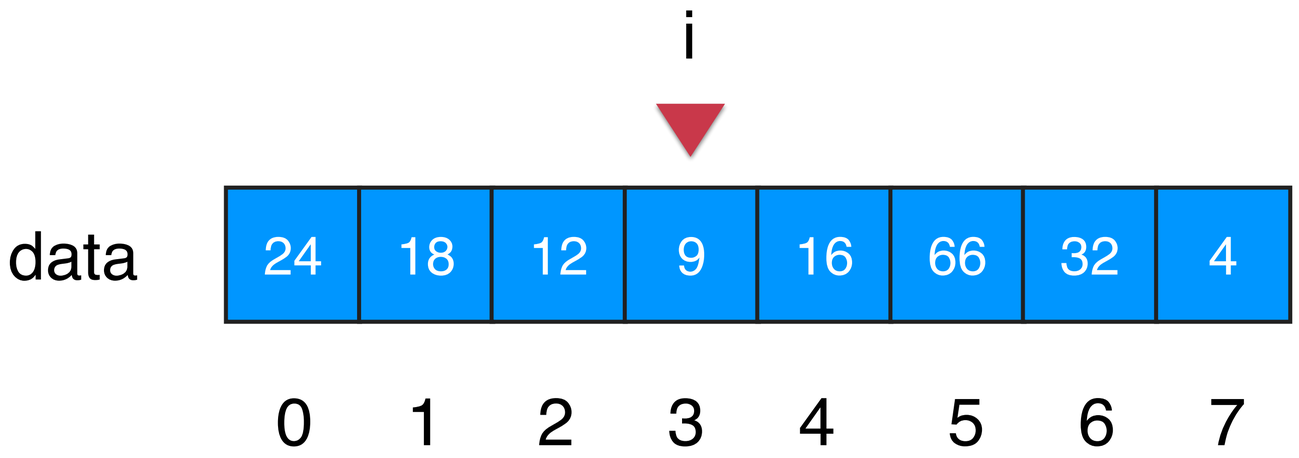
线性查找法

在 data 数组中查找 16



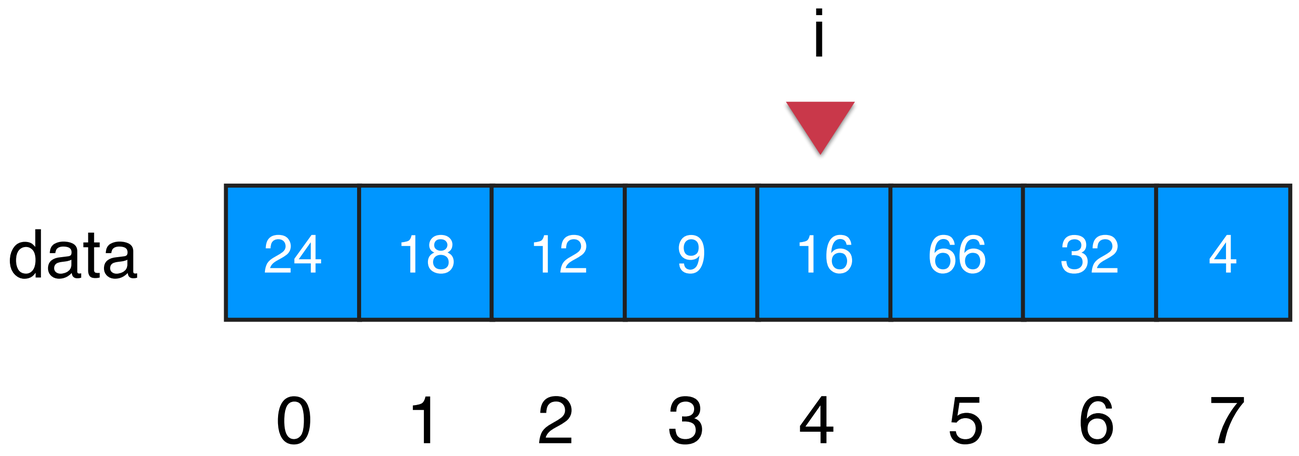
线性查找法

在 data 数组中查找 16



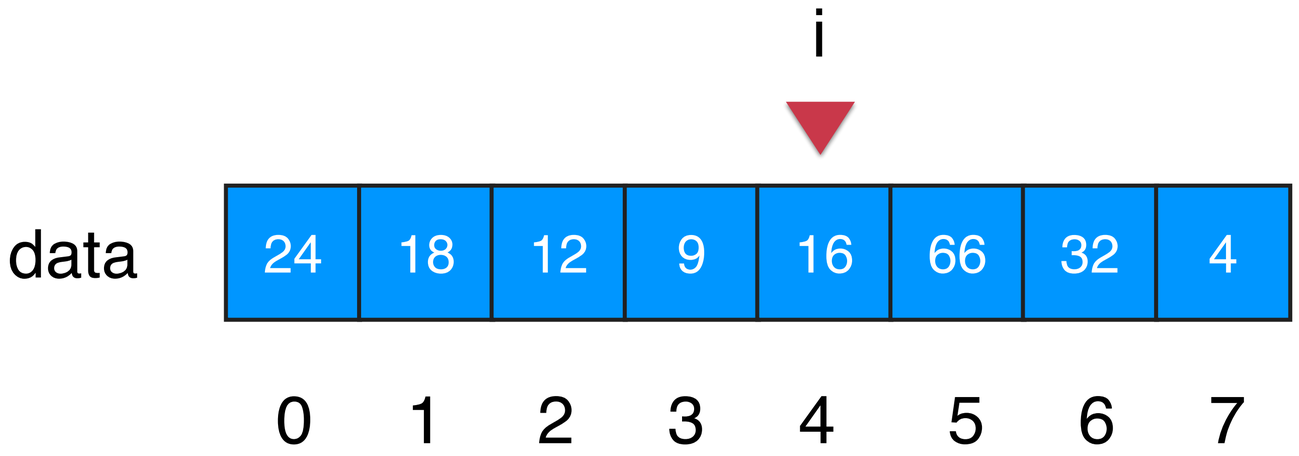
线性查找法

在 data 数组中查找 16



线性查找法

在 data 数组中查找 16



输入：数组，和目标元素

输出：目标元素所在的索引；若不存在，返回 -1

实现线性查找算法

liuyubobobo

实现线性查找算法

实现线性查找法

实现线性查找算法

使用 static

实现线性查找算法

修改构造函数为 private

使用泛型

liuyubobobo

使用泛型

使用泛型

使用泛型

- 不可以是基本数据类型，只能是类对象

  boolean , byte , char , short , int , long , float , double

- 每个基本数据类型都有对应的包装类

  Boolean , Byte , Character , Short , Integer , Long , Float , Double

使用泛型

作业：设计一个 Student 类?

在算法中使用自定义类

liuyubobobo

在算法中使用自定义类

设计 Student 类

循环不变量

liuyubobobo

循环不变量

```
public static <E> int search(E[] data, E target){

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

        if(data[i].equals(target))

            return i;

    return -1;

}
```

确认 data[i] 是否是目标

data[0…i - 1] 中没有找到目标

data[0…i) 中没有找到目标

循环不变量

确认 data[i] 不是目标

data[0…i] 中没有找到目标

循环不变量

```
public static <E> int search(E[] data, E target){

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

        if(data[i].equals(target))

            return i;

    return -1;

}
```

data[0…i - 1] 中没有找到目标

循环不变量

循环体：维持循环不变量

“证明”算法的正确性

写出正确的代码

循环不变量

写出正确的代码

定义清楚循环不变量

维护循环不变量

定义清楚函数的功能

LinearSearch

输入：数组，和目标元素

输出：目标元素所在的索引；若不存在，返回 -1

复杂度分析

liuyubobobo

复杂度分析

非常理论化的一个内容

计算复杂性理论

我们的课程：不强调这么理论化的内容

复杂度分析

复杂度分析：表示算法的性能

```
public static <E> int search(E[] data, E target){
    for(int i = 0; i < data.length; i ++)
        if(data[i].equals(target))
            return i;
    return -1;
}
```

通常看最差的情况

算法运行的上界

n = data.length

T = n？ T = 2n？ T = 3n？ T = 4n？

T = 5n？ T = 5n + 2？ 单位：ms？

O(n)

复杂度分析

复杂度分析：表示算法的性能

算法运行的上界

T = 5n + 2?　　T = c1 * n + c2

O(n)　　常数不重要

复杂度描述的是随着数据规模 n 的增大，

算法性能的变化趋势

复杂度分析

复杂度分析：表示算法的性能　　算法运行的上界

常数不重要　　复杂度描述的是随着数据规模 n 的增大，

算法性能的变化趋势

T1 = 10000n　　T2 =$2n^2$　　10000n < $2n^2$

O(n)　　<　　O(n^2)　　n > 5000

n_0 = 5000

存在某一临界点 n_0, 当 n >= n_0, T1 < T2

常见算法复杂度

liuyubobobo

常见算法复杂度

线性查找法　　O(n)

```
for(int i = 0; i < data.length; i ++)
    if(data[i].equals(target))
        return i;
```

常见算法复杂度

一个数组中的元素可以组成哪些数据对 $O(n^2)$

```
for(int i = 0; i < data.length; i ++)

    for(int j = i + 1; j < data.length; j ++)

        // 获得一个数据对 (data[i], data[j])
```

常见算法复杂度

遍历一个 n*n 的二维数组　$O(n^2)$

```
for(int i = 0; i < n; i ++)
    for(int j = 0; j < n; j ++)
        // 遍历到 A[i][j]
```

常见算法复杂度

遍历一个 n*n 的二维数组　$O(n^2)$

```
for(int i = 0; i < n; i ++)
    for(int j = 0; j < n; j ++)
        // 遍历到 A[i][j]
```

遍历一个 a*a 的二维数组　$O(n)$

a*a = n

```
for(int i = 0; i < a; i ++)
    for(int j = 0; j < a; j ++)
        // 遍历到 A[i][j]
```

明确 n 是谁。

常见算法复杂度

数字 n 的二进制位数 $O(logn)$ $O(log_2 n)$

```
while(n){
    n % 2 // n 的二进制中的一位
    n /= 2;
}
```

数字 n 的十进制位数? $O(log_{10} n)$

$$log_a b = \frac{log_c b}{log_c a}$$

$$log_2 n = \frac{log_{10} n}{log_{10} 2}$$

常见算法复杂度

数字 n 的二进制位数 $O(logn)$

```
while(n){
    n % 2 // n 的二进制中的一位
    n /= 2;
}
```

不能数循环个数

常见算法复杂度

数字 n 的所有约数

```
for(int i = 1; i <= n; i ++)
    if(n % i == 0)
        // i 是 n 的一个约数
```

O(n)

```
for(int i = 1; i * i <= n; i ++)
    if(n % i == 0)
        // i 和 n / i 是 n 的两个约数
```

$O(\sqrt{n})$

常见算法复杂度

长度为 n 的二进制数字 $O(2^n)$

长度为 n 的数组的所有排列 $O(n!)$

常见算法复杂度

判断数字 n 是否是偶数? O(1)

```
return n % 2 == 0
```

常见算法复杂度

$$O(1) < O(logn) < O(\sqrt{n}) < O(n) < O(nlogn) < O(n^2) < O(2^n) < O(n!)$$

常见算法复杂度

$$O(1) < O(logn) < O(\sqrt{n}) < O(n) < O(nlogn) < O(n^2) < O(2^n) < O(n!)$$

空间复杂度同理

```
public static <E> int search(E[] data, E target){
    for(int i = 0; i < data.length; i ++)
        if(data[i].equals(target))
            return i;
    return -1;
```

空间复杂度 O(1)

测试算法性能

liuyubobobo

测试算法性能

测试 LinearSearch 的性能

本章小结

liuyubobobo

本章小结

线性查找法

使用泛型：泛型方法

使用自定义的类

如何编写正确的程序：循环不变量

复杂度分析

测试算法性能

其他

欢迎大家关注我的个人公众号：是不是很酷

算法与数据结构体系课程

liuyubobobo