算法设计与分析

Computer Algorithm Design & Analysis

2019.10

王多强

dqwang@mail.hust.edu.cn

群名称：2019-算法

群　号：835135560

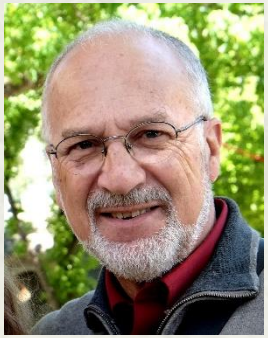
* 加入群：

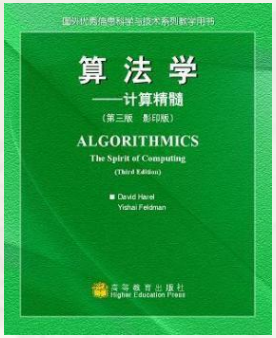
群名称：2019-算法

群　号：835135560

在信息技术高速发展的今天，算法被公认为是计算机科学的基石。

David Harel在他的《Algorithmics: The Spirit of Computing》（《算法学:计算精髓》）一书中说到：“算法不仅是计算机科学的一个分支，它更是计算机科学的核心”。





当今计算机技术的进步，如大数据、AI等，算法的进步是关键。

* 设计算法：从原始问题到建模求解的过程

分金币

圆桌旁坐着n个人，每人有一定数量的金币，金币总数能被n整除。每个人可以给他左右相邻的人一些金币，最终使得每个人的金币数目相等。怎么做？

你的任务是求出被转手的金币数量的最小值。

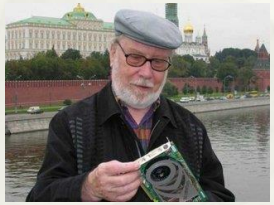
比如，n=4，且4个人的金币数量分别为1,2,5,4时，只需转移4枚金币（第3个人给第2个人两枚金币，第2个人和第4个人分别给第1个人1枚金币）即可实现每人手中的金币数目相等。

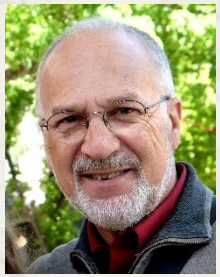
漫谈算法

- “高大上”的算法（Algorithm）
  - 算法是计算机软件的灵魂
    - 软件=数据结构+算法

    （Niklaus Wirth，瑞士，计算机科学家，
    图灵奖获得者，Pascal语言的发明者）
  - 算法是计算机科学的核心
    - the core of computer science

    （David Harel）
  - 国家科技综合实力的体现





算法不陌生

举个例子：排序（Sort）

The formally defined ***sorting problem***:

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

基本思想：小的数往前排，大的数往后排

怎么排？算法

- 冒泡排序
- 选择排序
- 归并排序
- 快速排序
- 堆排序
- Shell排序
- …

- 每种算法都是一组特定的规则，规定了一种处理数据的运算方法
- 不同的规则、不同的性质和性能
- 数据是死的，算法是活的

学习算法你能获得什么

- 积累丰富的解决问题的经验和技能
- 训练编程
  - 从简单技术的应用到技巧、技能的提高
- 训练思维
  - 严谨、科学的逻辑推理能力
  - 培养计算机系统观，培养计算机思维
  - 培养使用计算机问题求解能力

关于算法教学

课程核心：

介绍算法设计与分析的基本理论、方法和技术，训练程序思维，奠定算法设计的基础。

教学目的：

- 在理论学习上，掌握算法设计与分析的基本理论和方法，培养设计算法和算法分析的能力。
- 在实践教学上，掌握算法实现的技术、技巧，学习算法的正确性验证、效率分析、优化技术，以及算法在实际问题中的分析与应用。
- 培养独立研究和创新能力。

本课程需要的基础

- 数据结构 √
- 程序设计语言（C/C++）：结构化设计 √
- 一定的数学基础：高数、离散、概率 √
- 一些背景知识：操作系统、编译

授课形式:

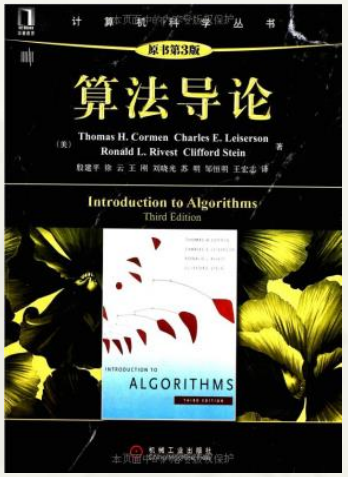
- 课堂教学：（√）
- 课堂讨论：
- 上机实践：需要提交实验报告

主要参考书

- **Introduction to Algorithms**

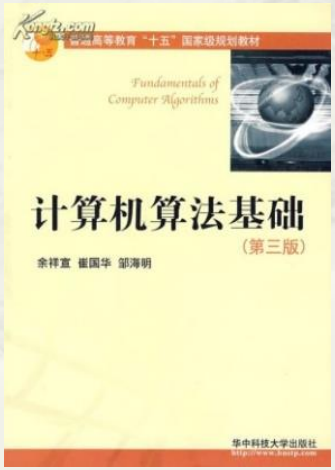
  Thomas H. Cormen,etc.,

  third edition, The MIT Press.



- **计算机算法基础**

  余祥宣等编著，华中科技大学出版社

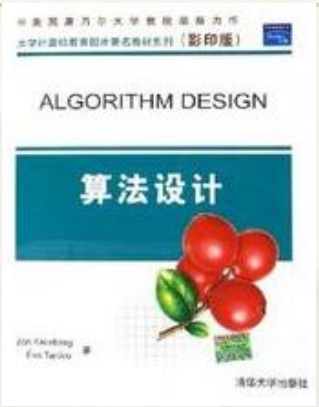


推荐阅读

- **Algorithm Design**

  Jon Kleinberg，Eva Tardos etc.

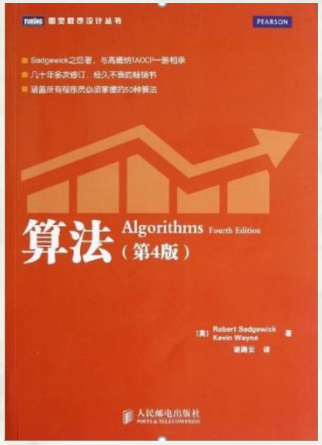
  Cornell University



- **Algorithms**

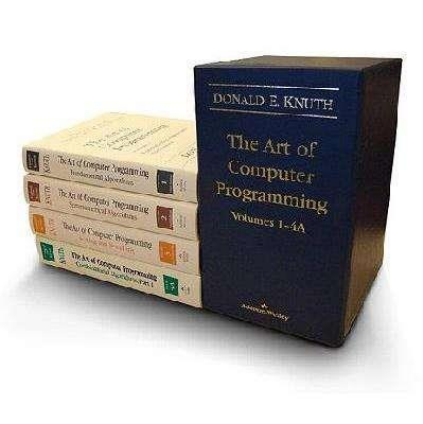
  Robert Sedgewick / Kevin

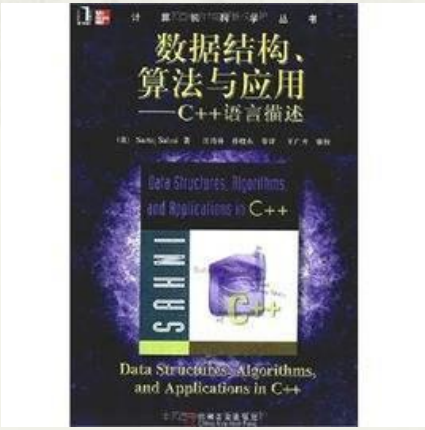
  Princeton University



其它参考书

- The Art of Computer Programming, Donald E.Knuth. Volume 1-3, Second Edition.
- Data Structures, Algorithms, and Applications in C++(Part 3) Sartaj Sahni, China Machine Press
- 算法设计与分析，王晓东，清华大学出版社
- etc.





Chapter 1

The Role of Algorithms in Computing

算法在计算中的作用

本章尝试回答以下问题：

- 什么是算法？
- 为什么算法值得研究？
- 算法的作用是什么？

1.1 What are algorithms?

非形式地说，算法就是任何良定义（well-defined）的计算过程，该过程取某个值或值的集合作为输入（input），并产生某个值或者值的集合作为输出（output）。

—— 算法就是把输入转换成输出的计算步骤。

—— 在计算机科学中，算法是使用计算机解一类问题的精确、有效方法的代名词；

算法是一组有穷的规则，它规定了解决某一特定类型问题的一系列运算。（选自《计算机算法基础》）

对算法概念的理解

- 算法由运算组成

  算术运算、逻辑运算、关系运算、赋值运算、过程调用

- 算法有其特殊性

  解决不同问题的算法是不相同的，没有万能的算法

- 算法是有穷的计算过程

  静态上：规则/运算/语句的数量有穷

  动态上：计算过程/计算时间有限

一般来说，问题陈述说明了期望的输入/输出关系，而算法描述了一个特定的计算过程来实现这种输入到输出的转换。

Example：排序问题

- **问题陈述：**排序问题（*sorting problem*）的形式定义如下

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

- **算　法：**冒泡排序、插入排序、归并排序等，将完成上述的排序过程。

问题实例(instance)：由计算一个问题所必需的输入组成。

For example,

Input sequence:　(31, 41, 59, 26, 41, 58)

Output sequence : (26, 31, 41, 41, 58, 59)

← instance

算法无处不在

- 从应用领域来看：
  - 人类基因工程（The Human Genome Project）：会用到很多算法设计思想，如：LCS in DNA comparison,SS（Shotgun Sequencing）.
  - 互联网应用（The Internet Applications）：管理和处理大数据（Big Data)、网络路由、搜索引擎的设计都离不开算法
  - 电子商务（Electronic commerce）：利用数值算法和数论进行个人信息的加密保护
  - 制造业和其他应用领域
- 从具体问题来看：
  - 图计算
  - 字符串处理
  - 计算几何问题
  - 最优化问题
  - Etc. See page from 3 to 5

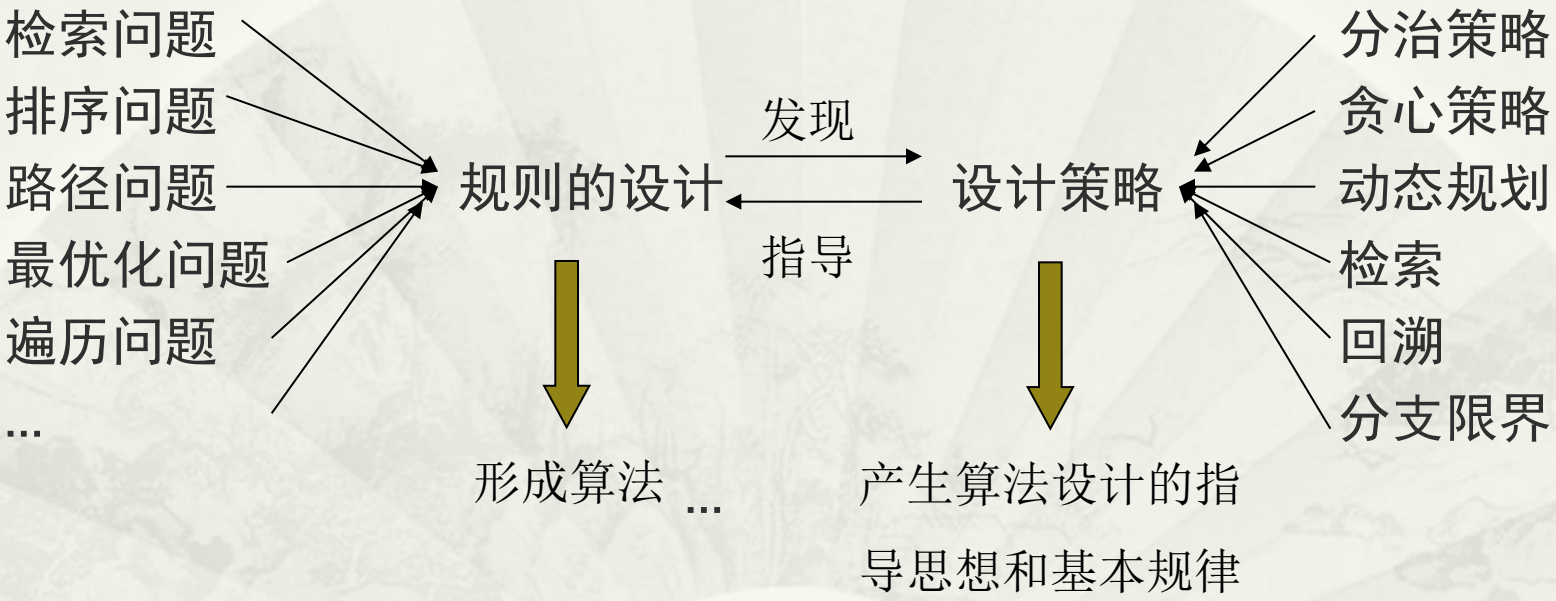
关于“算法正确”的概念

- 若对每个输入实例，算法都以正确的输出停机，则称该算法是正确的，并称正确的算法解决了给定的问题。
- 不正确的算法对某些输入实例可能根本不停机，也可能给出一个回答然后停机，但结果与期望不符甚至相反。
  - 但不正确的算法也并非绝对无用，在不正确的算法错误率可控时，有可能也是有用和有效的（如近似算法）。
  - 但通常，我们还只是关心正确的算法。

思考：

- 我们学会了解决一个个具体问题的算法，那么在这些算法的背后有没有一些共性的东西？有没有可以总结出来的规律和方法？这些规律和方法对于其它算法的设计有没有指导意义？
- 能不能找到一些算法设计的一般策略、技术和方法来指导未来新问题的求解？

算法：求解问题的一组规则



较高的算法设计能力不仅在于简单使用一些具体的算法，更在于对算法设计方法的掌握上。只有深入理解并掌握了算法设计的一般策略、规律和方法，才能在面对新问题时创造出新的算法。

算法学习要做到：

- 深入理解算法设计的一般规律、技术和方法
- 灵活运用现有的算法解决实际问题
- 在改造客观世界的过程中，运用学到的知识创造新的算法，解决新的问题——转变成“生产力”
- 实现知识到能力的转化

1.2 作为一种技术的算法

- 算法不仅强调其正确性
- 算法也有好坏之分

  效率（Efficiency）对于算法的有效性有非常重要的影响。

  - 现实应用中，时间和空间都是有限的资源，因此我们应选择时间和空间方面有效的算法来求解问题

为求解相同问题而设计的不同算法在效率方面常常具有显著的差别，这种差别可能比由于硬件和软件造成的差别还要重要的多。

For example：排序问题

insertion sort: time roughly equal to c_1n^2

merge sort: time roughly equal to c_2nlgn

Let $c_1 = 2$, and $c_2 = 50$,

假设对1000万个数进行排序

令 $c_1 = 2$, and $c_2 = 50$, 对1000万（10^7）个数进行排序。并设插入排序在每秒执行百亿（10^{10}）条指令的计算机上运行，归并排序在每秒仅执行1000万条指令的计算机上运行，计算机速度相差1000倍。程序的实际执行时间有什么差别呢？

* ***insertion sort:***

$$\frac{2 \cdot (10^7)^2 \text{ instructions}}{10^{10} \text{ instructions/second}} = 20{,}000 \text{ seconds (more than 5.5 hours)} .$$

* ***merge sort***:

$$\frac{50 \cdot 10^7 \lg 10^7 \text{ instructions}}{10^7 \text{ instructions/second}} \approx 1163 \text{ seconds (less than 20 minutes)} .$$

* 可见使用一个运行时间“增长”较慢，时间复杂度低、时间效率高的算法，即使采用较差的编译器、运行速度较慢的计算机，在数据规模足够大的时候，也比时间效率低的算法快。

上述的例子表明，我们应该像对待计算机硬件一样把算法看成是一种技术

——整个系统的性能不但依赖于选择快速的硬件，而且还依赖于选择有效的算法。

- 随着计算机能力的不断增强，我们使用计算机来求解的问题的规模会越来越大，算法之间效率的差异也变得越来越显著。
- 因此，是否具有算法知识与技术的坚实基础是区分真正熟练的程序员与初学者的一个基本特征。

使用现代计算技术，如果你对算法懂得不多，尽管你也可以完成一些任务，但如果有一个好的算法背景，你可以做更多的事情，也会做得更好一些。

End of Chpt 1

- 算法的基本概念
- 算法的重要性

Chapter 2

Getting Started

算法基础

- 本章将介绍一个贯穿整个课程、进行算法设计与分析的框架。
- 包括:
  - 问题的描述
  - 算法的伪代码描述
  - 算法正确性证明
  - 算法的分析基础等

伪代码（Pseudocode）

在本课程中，我们将用一种“伪代码”书写程序。

- 伪代码：非真实代码，类似于 C, C++, Java, Python, or Pascal，主要用于描述算法的计算步骤而非放在实际的计算机上运行，易于书写但没有固定的标准。本书的规范见**P11, “Pseudocode conventions”**。
- 伪代码和真实代码的区别在于:
  - 在伪代码中，关注的是使用最清晰、最简洁的表示方法来说明给定的算法。可以不用关心**软件工程**的问题。
  - 为了更简洁地表达算法的本质，用伪代码描述算法时，常常忽略数据抽象、模块性和错误处理等问题，从而使得算法设计的注意力放在**算法最核心**的部分。

2.1 插入排序（Insertion sort）

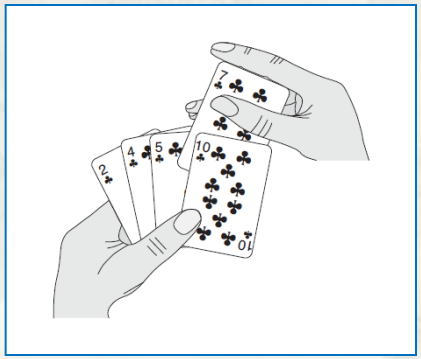
- 排序问题的描述：

Input: A sequence of n numbers $\langle a_1, a_2, \ldots, a_n \rangle$.

Output: A permutation (reordering) $\langle a'_1, a'_2, \ldots, a'_n \rangle$ of the input sequence such that $a'_1 \leq a'_2 \leq \cdots \leq a'_n$.

1. 插入排序的基本思想

- 插入排序的基本思想：



- 插入排序是一个对少量元素排序比较有效的算法

2. 插入排序的伪代码描述：INSERTION-SORT

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1 . . j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

说明：

- 设输入的原始数据在数组A中， INSERTION-SORT将“**原址排序**”输入的数：

  原址排序：算法在数组A空间中重排这些数，在任何时候，最多只有其中的常数个数字存储在数组之外。

- 过程结束时，输入数组A包含排序好的输出序列。

设A=<5,2,4,6,1,3>， INSERTION-SORT在A上的排序过程

如图所示：

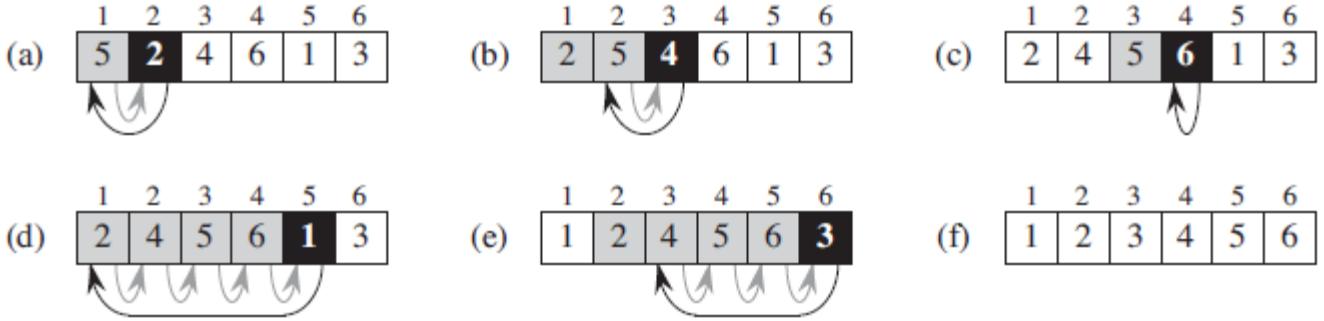


Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key moves to in line 8. **(f)** The final sorted array.

3. 循环不变式与插入算法的正确性

1）循环不变式（Loop invariants）

INSERTION-SORT 的for循环中，循环变量为j（j代表当前正要被插入到序列中的元素的下标）。循环过程具有以下性质：

子数组A[1~j-1]是已经被排好序的子序列。

这一性质，在j被赋予初值2，首次进入循环之前成立，而且每次循环之后（j加了1）、进入下一次循环之前也成立。

——把这种在第一次进入循环之前成立、每次循环之后还成立的关系称为“循环不变式”或“循环不变性质”。

- 插入排序for循环的循环不变式可以描述为：

  在第1~8行的for循环的每次迭代开始时，子数组A[1~j-1]由原来在A[1~j-1]中的元素组成，且已按序排列。

- 可以利用循环不变关系证明循环的正确性。

分三步走：

1）初始化：证明初始状态时循环不变式成立，即证明循环不变式在循环开始之前为真；

2）保持：证明每次循环之后、下一次循环开始之前循环不变式仍为真；

3）终止：证明循环可以在有限次循环之后终止。

证明循环正确性的思路：

- 第1）和2）步类似于数学归纳法的证明策略
- 第3）步保证算法可以终止
- 如果1）~3）都满足，则说明循环过程正确

（为什么？有理论保证，自学：程序的逻辑）

这里利用循环不变关系，证明插入排序的正确性。

1）初始化：证明循环不变式在循环开始之前为真

第一次循环之前，j=2，子数组A[1..j-1]实际上只有一个元素，即A[1]，且这个A[1]是A中原来的元素。所以表明第一次循环迭代之前循环不变式成立 —— 初始状态成立。

2）保持：证明每次循环之后循环不变式仍为真

观察for循环体，可以看到4~7行是将A[j-1]、A[j-2]、A[j-3]…依次向右移动一个位置（while循环，这里不另行证明），直到找到对当前A[j]适当的位置，之后在第8行将A[j]插入该位置。

这时子数组A[1..j]显然是由原来在A[1..j]中的元素组成，且已按序排列。再之后执行j+=1（下次循环开始之前的状态），此时原来的“A[1..j]”变成了新的A[1..j-1]。故循环不变式依然成立。

3）终止：循环可以有限次终止。

可以看到，每次循环后j都加1，而循环终止条件是j>n（即A.length），所以必有在n-1次循环后j=n+1，循环终止。

注：此时，循环不变式依然成立（即A[1..n]是由原A中的n个元素组成且已排序）。

故此，上述三条性质都成立，根据证明策略，插入排序正确 —— 确切地说是其中的for循环正确，但for循环是插入排序过程的主体，所以整个算法正确。

2.2 分析算法

- 分析算法主要是预测算法需要资源的程度。
- 资源包括：
  - 时间
  - 空间：内存
  - 其他：通信带宽、硬件资源等

但从算法的角度，我们主要关心算法的时间复杂度和空间复杂度。

1. 分析算法的目的

- **算法选择的需要**：对同一个问题可以设计不同的算法，不同的算法对时间和空间需求是不同的。

  因此，通过对算法的分析可以了解算法的性能，从而在解决同一问题的不同算法之间比较性能的好坏，选择好的算法，避免无益的人力和物力浪费。

- **算法优化的需要**：通过对算法分析，可以对算法作深入了解，发现其中可以改进的地方，从而可以进一步优化算法，让其更好地工作。

2. 重要的假设和约定

1）计算机模型的假设

- 计算机形式理论模型：有限自动机（FA）、Turing机
- RAM模型（random-access machine，随机访问机）
  - 在RAM模型中，指令一条一条被取出和执行，没有并发操作。
  - 关于RAM的讨论参考P13相关内容。
- 通用的顺序计算机模型：
  - 单CPU——串行算法（部分内容可能涉及多CPU（核）和并行处理的概念）
  - 有足够的“内存”，并能在固定的时间内存取任意数据单元

区分计算机的理论模型与实际的计算机

2）计算的约定

思考1：

一般，我们所说的算法/程序的执行时间是什么单位？

日常生活中，时间是以小时、分钟、秒、毫秒等时间单位为计量单位的。

但算法/程序的执行时间是什么？

$O(n^2)$、$O(n\log n)$...和执行时间有什么关系呢？

- 算法的执行时间是算法中所有运算执行时间的总和

可以表示为:

$$\text{算法的执行时间} = \sum f_i t_i$$

其中，f_i：是运算i的执行次数，称为该运算的频率计数

t_i：是运算i在实际的计算机上每执行一次所用的时间

- f_i仅与算法的控制流程有关，与实际使用的计算机硬件和编制程序的语言无关。
- t_i与程序设计语言和计算机硬件有关。

如何确定算法使用了哪些运算,每种运算的f_i和t_i又是多少?

运算的分类

依照运算的时间特性，将运算分为时间囿界于常数的运算和时间非囿界于常数的运算。

- 时间囿界于常数的运算：
  - 基本算术运算，如整数、浮点数的加、减、乘、除
  - 字符运算、赋值运算、过程调用等

特点：执行时间是固定量，与操作数无关。

例：　1+1 = 2　vs 10000+10000 = 20000

100*100 = 10000 vs 10000*10000 = 100000000

CALL INSERTIONSORT

更一般的情况

设有n种运算c_1，c_2，…，c_n，它们的实际执行时间分别是t_1，t_2，…，t_n。

令$t_0 = \max(t_1, t_2, \cdots, t_n)$，则每种运算执行一次的时间都可以用$t_0$进行抽象，即可**用$t_0$进行限界(上界)**。

进一步，视t_0为一个**单位时间**，则这些运算“理论”上都可视为仅花一个固定的单位时间t_0即可完成的运算

——称具有这种性质的运算为**时间囿界于常数的运算**。

- 时间非囿界于常数的运算

特点：运算的执行时间与操作数相关，每次执行的时间是一个不定的量。

如：

- 字符串操作：与字符串中字符的数量成正比，如字符串的比较运算strcmp。
- 记录操作：与记录的属性数、属性类型等有关

怎么分析时间非囿界于常数的运算？

- 这类运算通常是由更基本的运算组成的。而这些基本运算是时间囿界于常数的运算。 所以分析时间非囿界于常数的运算时，将其分解成若干时间囿界于常数的运算即可。

如：字符串比较时间t_{string}

$$t_{string} = Length\ (String) * t_{char}$$

思考：为什么要引入“时间囿界于常数的运算”这一概念?

引入“单位时间”的概念，将“单位时间”视为“1”，从而将算法的理论执行时间转换成“单位时间”意义下的执行次数，从而实现了“执行时间”与时间复杂度概念的统一。

$$\sum f_i t_i$$

3）工作数据集的选择

- 算法的执行情况与输入的数据有什么样的关系呢？
  - 算法的执行时间与输入数据的规模相关，一般规模越大，执行时间越长。

    如：插入算法，10个数的排序时间显然小于1000个数的排序时间
  - 在不同的数据配置上，同一算法有不同的执行情况，可分为最好、最坏和平均等情况讨论。

    思考：插入排序的最好、最坏和平均时间复杂度是多少？
- 编制不同的数据配置，分析算法的最好、最坏、平均工作情况是算法分析的一项重要工作

3. 如何进行算法分析?

对算法的全面分析将分两个阶段进行：事前分析和事后测试（理论分析和程序测试）。

- 事前分析：求算法时间/空间复杂度的限界函数。
  - 限界函数通常是关于问题规模n的特征函数，被表示成O、Ω或Θ的形式。

    如：归并排序的时间复杂度是Θ(nlogn)。

  - **特征函数**是通过分析算法的控制逻辑得来的，是对算法所用运算执行次数的统计结果。与使用的程序设计语言和计算机硬件无关。

- 事后测试：将算法编制成程序，放到实际的计算机上运行，收集程序的执行时间和空间占用等统计资料，然后进行分析判断。
  - 事后测试与物理实现直接有关。

这里，所谓的算法分析主要集中于与物理实现无关的**事前分析阶段** —— 获取算法的理论时间/空间复杂度——关于问题规模n的特征函数。

1）事前分析

目标：获取算法的时间/空间复杂度函数。

可以分为时间、空间两个方面：

- 时间分析
  - 了解算法中使用了哪些运算（具有单位执行时间的基本运算）。
  - 分析程序的控制流程，从而确定各类运算的执行次数。
  - 将对运算执行次数的分析结果表示成关于问题规模n的特征函数。
  - 算法性能有最好、平均、最坏等情况，与数据配置有关。分析典型的数据配置，了解算法在各种情况下的执行情况。

空间分析

- 分析算法中各类变量的定义情况
- 将空间占用量表示成为问题规模n的特征函数。
- 空间占用有最大、平均、最少等情况，与数据配置有关。分析典型的数据配置，了解算法在各种情况下的空间占用情况。

如何进行时间分析?

- 统计算法中各类运算的执行次数
  - 统计对象：运算
    1）基本运算：时间囿界于常数的运算
    2）复合运算：具有固定执行时间的程序块，如一条语句、一个过程，甚至一个函数，由基本运算组成。
  - 统计内容：频率计数，即算法中语句或运算的执行次数。
    - 顺序结构中的运算/语句执行次数计为1
    - 嵌套结构中的运算/语句执行次数等于被循环执行的次数

强调：语句或运算的执行次数是从算法的控制流程得来的。分析控制流程的关键是确定算法中使用的嵌套结构，包括循环语句、嵌套的调用等。

例：执行次数的统计

```
x←x+y
```

(a)

```
for i ←1 to n do
    x ← x + y
repeat
```

(b)

```
for i ←1 to n do
    for j ←1 to n do
        x ← x +y
    repeat
repeat
```

(c)

分析：

(a)： x←x+y执行了1次

(b)： x←x+y执行了n次

(c)： x←x+y执行了n^2次

思考：

```
for i ←1 to n do
    for j ←i to n do
        x ← x +y
    repeat
repeat
```

(d)

插入排序时间分析：

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

- cost列：给出了每一行语句执行一次的时间。这里假定第i行语句的执行的时间是c_i，且c_i 是一个常量。
- times列：给出了每一行语句被执行的总次数。其中，t_j表示对当前的j，第5行执行while循环测试的次数。

- 整个算法的执行时间是执行所有语句的时间之和。
  - 如果语句i需要执行n次，每次需要c_i的时间，则该语句的总执行时间是：$c_i n$.
  - 令 T (n)是输入n个值时INSERTION-SORT的运行时间，则有：

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1) .$$

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j - 1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

注：即使规模相同，一个算法的执行时间也可能依赖于给定的输入。

如：INSERTION-SORT

- 最好情况：初始数组已经排序好了。此时对每一次for循环，内部的while循环体都不会执行。

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

注：即使规模相同，一个

于给定的输入。

如：INSERTION-SORT

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

- 最好情况：初始数组已经排序好了。此时对每一次for循环，内部的while循环体都不会执行。

所以最好情况的运行时间为：

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)\,. \end{aligned}$$

- 第5行做了n-1次测试，但第6、7行没有被执行。
- 执行时间的表达式为n的**线性函数**，即具有an+b的形式。

- **最坏情况**：初始数组是反向排序的。

此时对每一次for循环，内部的while循环体都执行最多次数的测试（即j-1次）。

最坏情况的运行时间为

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j - 1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

- 执行时间的表达式为n的**二次函数**，即具有an²+bn+c的形式。

- **最坏情况**：初始数组是反向

此时对每一次for循环，

次数的测试（即j-1次）。

最坏情况的运行时间为：

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n}(t_j-1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n}(t_j-1)$
8 $A[i+1] = key$	c_8	$n-1$

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\
&\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\
&= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\
&\quad - (c_2 + c_4 + c_5 + c_8)\,.
\end{aligned}
$$

- 执行时间的表达式为n的**二次函数**，即具有an^2+bn+c的形式。

- 除了最坏情况、最好情况分析，还有平均情况分析。

  平均情况是规模为n的情况下，算法的平均执行时间。

  - 通常情况下，平均运行时间是算法在各种情况下执行时间之和与输入情况数的比值（算术平均）。

但一般，我们更关心算法的最坏情况执行时间。

为什么?

- 一个算法的最坏情况执行时间给出了任何输入的运行时间的一个上界。知道了这个界，就能确保算法绝不需要更长的时间，我们就不需要再对算法做更坏的打算。
- 对某些算法，最坏情况经常出现。
- 对很多算法，平均情况往往与最坏情况大致一样。
  - 如插入排序，就一般情况而言，while循环中为确定当前A[j]的插入位置，平均要检查一半的元素。那么导致的平均执行时间就数量级来说和最坏情况一样，依然是n的二次函数，只是常系数小了一些。

函数表达式的数量级（阶、增长量级）

- 函数表达式中的最高次项的次数称为函数表达式的数量级（阶）
  - 在频率计数函数表达式中，数量级是衡量频率计数的“大小”的一种测度。
  - 数量级从本质上反映了算法复杂度的高低

    数量级越小，算法的复杂度越低，同等规模下算法执行时间越短。

    数量级越大，算法的复杂度越高，同等规模下算法执行时间越长。

例：假如求解同一个问题的三个算法分别具有n， n^2 ， n^3 数量级。

若n=10，则可能的执行时间将分别是10，100，1000个单位时间。

实例：工资的级别 （定性的表示）

限界函数：用频率计数函数表达式中的**最高次项**表示限界函数，并忽略常系数，记为：g(n)。

- g(n)通常是关于n的形式简单的函数式，如：logn，nlogn等
- 不同算法的g(n)有不同的具体形式。
- g(n)通常是对算法中**最复杂**的计算部分分析而来的。
  - g(n)忽略了函数表达式中次数较低的“次要”项，体现了算法复杂性的最本质特征，且忽略常系数。（对于足够大的n，低阶项和常系数都不如最高次项中的因子重要，但对于较小的n却未必，一般还要分情况讨论）。
- g(n)通常取**单项**的形式。

空间特性分析（与时间特性的分析类似，略）

2）事后测试

- 用实际的程序设计语言把算法编制成程序，在实际的计算机硬件平台上运行程序，统计执行实际耗费的时间与空间，与事前分析的结论进行比较，验证先前的分析结论——包括正确性、执行性能等，比较并优化所设计的算法。
- 分析手段：作时、空性能分布图

2. 算法的五个重要特性

确定性、能行性、输入、输出、有穷性

1）确定性：算法使用的每种运算必须要有确切的定义，不能有二义性。

- 不符合确定性的运算如：5/0 ，将6或7与x相加

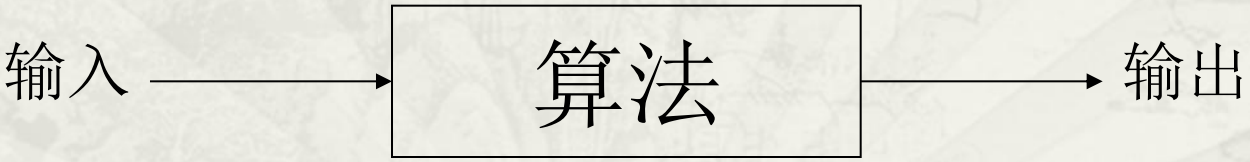
2）能行性：算法中有待实现的运算都是基本的运算，原理上每种运算都能由人用纸和笔在“有限”的时间内完成。

- 整数的算术运算是“能行”的，实数的算术运算可能是“不能行”的

3）输入：每个算法都有0个或多个输入。

- 这些输入是在算法开始之前给出的量，取自于特定的对象集合——定义域

4）输出：一个算法产生一个或多个输出，这些输出是同输入有某种特定关系的量。



5）有穷性

一个算法总是在执行了有穷步的运算之后终止。

- 计算过程：满足确定性、能行性、输入、输出，但**不一定满足有穷性**的一组规则称为计算过程。

- 操作系统是计算过程的典型代表：（不终止的运行过程）
- 算法是“可以终止的计算过程”。

- **时效性：**实际问题往往都有时间要求，只有在要求的时间内解决问题才是有意义的。

例：国际象棋（启发）、数值天气预报

- 基于算法的时效性，只有把在相当有穷步内终止的算法投入到计算机上运行才是实际可行的。

——这就要通过“算法分析”，了解算法性质，给出算法计算时间的一个精确的描述，以衡量算法的执行速度，选择合适的算法有效地解决问题。

2.3 分析结论的证明

当我们有了分析的结论，怎么证明结论是正确的？

常用的方法：

- 直接推导（略）
- 数学归纳法
- 反证法
- 反例法

1）数学归纳法

数学归纳法是用来证明某些与自然数有关的数学命题的一种推理方法，有广泛的应用。

已知最早的使用数学归纳法的证明出现于 Francesco Maurolico 的 Arithmeticorum libri duo (1575年)。Maurolico 证明了前 n 个奇数的总和是 n^2。

数学归纳法采用递推策略进行命题证明，分为标准的二部分完成：

第一步： 基准情形（递归基础、初始情形、平凡情形）

该部分证明定理对于某个（某些）小的值是正确的，一般证明命题在n=1（n_0）时命题成立。这是递推的基础。

第二步：归纳假设和推论的证明

该部分首先假设定理对于直到某个有限数k（或k以内）的所有情况都成立，然后使用这个假设证明定理对于下一个值（通常就是k+1）也成立。如果证明了k+1正确，则完成整个证明。

完成了上述两步，就可下结论：对任何自然数n（或$n \geq n_0$）结论都正确，证毕。

以上两步密切相关，缺一不可。而且证明的关键是n=k+1时命题成立的推证，这是无限递推下去的理论依据，它将判定命题的正确性由特殊推广到一般，使命题的正确性突破了有限而达到无限。

实例：证斐波那契数$F_i<(5/3)^i$

这里：$F_0 = 1$，$F_1 = 1$；

$F_i = F_{i-1} + F_{i-2}$，$i \geqslant 2$

证明：

1. 初始情形

容易验证$F_1 = 1 < 5/3$、$F_2 = 2 < (5/3)^2$，

所以，初始情形成立。

2. 归纳假设

设定理对于$i = 1, 2, \ldots\ldots k$都成立，现证明定理对于$i=k+1$时也成立，即$F_{k+1} < (5/3)^{k+1}$。

根据斐波那契数的定义有，

$$F_{k+1} = F_k + F_{k-1}$$

将归纳假设用于等号右边，有

$$F_{k+1} < (5/3)^k + (5/3)^{k-1}$$

$$< (3/5)(5/3)^{k+1} + (3/5)^2(5/3)^{k+1}$$

$$< (3/5+9/25)(5/3)^{k+1}$$

$$< (24/25)(5/3)^{k+1}$$

$$< (5/3)^{k+1}$$

∴n=k+1时结论成立。

由（1）、（2）知，定理得证。

2）反证法

反证法就是通过假设定理不成立，然后证明该假设将导致某个已知的性质不成立，从而证明假设是错误的而原始命题是正确的。

实例：证明存在无穷多个素数

反证法：假设定理不成立。则将存在最大的素数P_k。

令$P_1, P_2, \ldots\ldots P_k$是依次排列的所有素数

令 ： $N= P_1P_2.....P_k+1$

显然N是比P_k大的数。

思考：N是素数吗？

因为每个整数，或者是素数，或者是素数的乘积，而N均不能被$P_1, P_2, ...P_k$整除，因为用$P_1, P_2, ...P_k$中的任何一个除N总有余数1，所以N是素数。

而$N>P_k$，故，与P_k是最大的素数的假设相矛盾。

∴ 假设不成立，原定理得证。

3）反例法

反例法就是举一组具体的数据(算例)，带入定理给出的表达式，证明该数据计算出来的结果与预想结果不符，从而证明定理的结论有错。

特别的应用：证明定理不成立最直接的方法就是举出一个反例。

例：斐波那契数$F_k \leqslant k^2$是否成立？

举反例：令k=11

则：$F_{11} = 144 > 11^2$，所以原命题不成立。

课外阅读：

算法导论（3rd）：第1章、2.1插入排序、2.2分析算法

其他章节自行阅读

作业1：抄写　通过抄写，熟悉伪代码的书写方法

P10，INSERTIONSORT

P19，MERGESORT

P17，MERGE

作业2：

1.2-2

Suppose we are comparing implementations of insertion sort and merge sort on the same machine. For inputs of size n, insertion sort runs in $8n^2$ steps, while merge sort runs in $64n \lg n$ steps. For which values of n does insertion sort beat merge sort?

1.2-3

What is the smallest value of n such that an algorithm whose running time is $100n^2$ runs faster than an algorithm whose running time is 2^n on the same machine?

思考题：2.2-2 选择算法