算法设计与分析（实验班）

第1讲 概述

汪小林

北京大学 信息科学技术学院

课程主要内容

问题处理策略

计算复杂性理论

算法设计与优化

算法分析方法

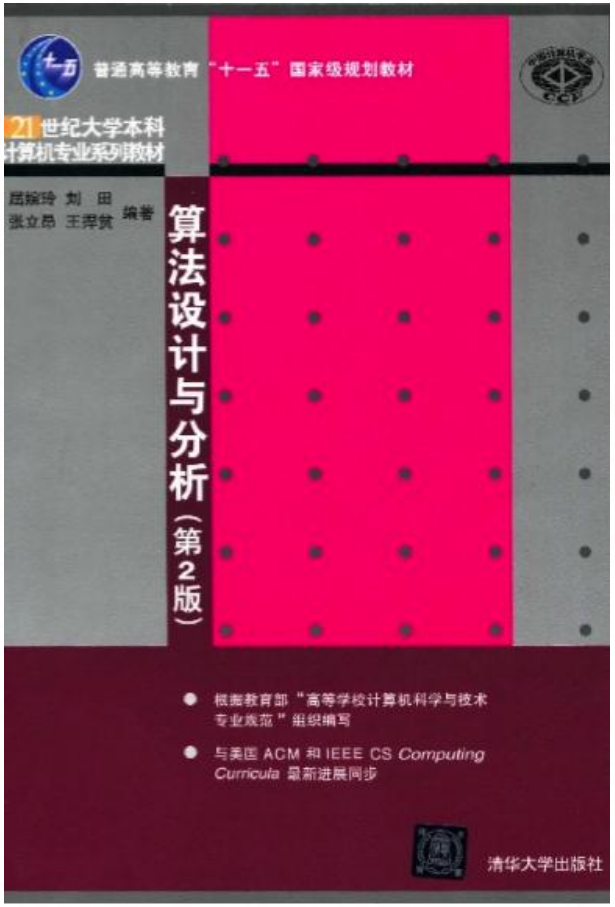
算法设计技术

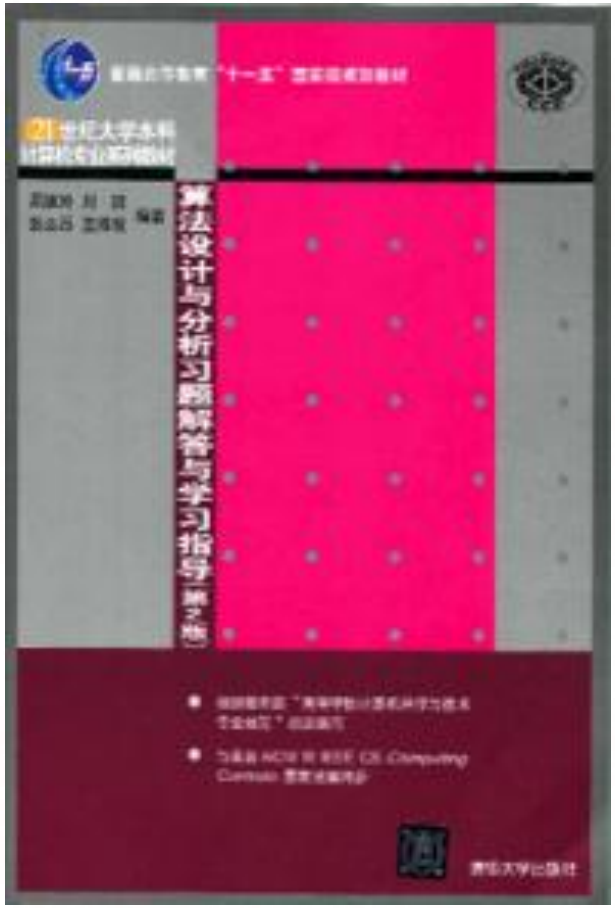
基础知识

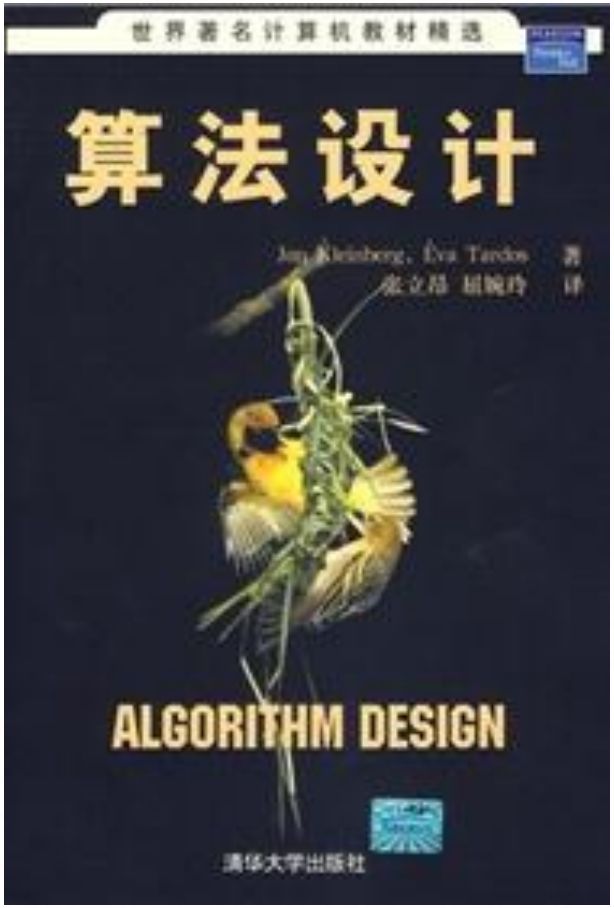
教材和网站

- 教材
  - 《算法设计与分析》 屈婉玲等 清华大学出版社
  - 《算法设计》Jon Kleinberg, Eva Tardos 清华大学出版社
- 参考书
  - 《算法导论》 Thomas H.C 机械工业出版社 (Introduction to Algorithms)第三版
- 教学网站
  - 学校教学网《算法设计与分析（实验班）》汪小林
  - http://programming.grids.cn http://poj.org

教材


普通高等教育"十一五"国家级规划教材
21世纪大学本科
计算机专业系列教材
算法设计与分析（第2版）
清华大学出版社


算法设计与分析习题解答与学习指导（第2版）


世界著名计算机教材精选
算法设计
ALGORITHM DESIGN
清华大学出版社

考核方法

- 期中笔试（10%）
- 期末笔试（40%）
- 作业、论文及小班课表现（50%）

本讲主要内容

- 算法的定义与基本概念
- 算法的分类
- 算法设计技术
- 算法的伪码描述

1.1、算法的基本概念

几个例子

例1：最少总等待时间调度问题

已知：任务集 $S = \{1, 2, \dots, n\}$，

第 j 项任务加工时间：$t_j \in Z^+, j = 1, 2, \dots, n$

一个可行调度方案：$1, 2, \dots, n$ 的排列 $i_1, i_2, \dots, i_n$

求：总等待时间最少的调度方案，

即求 S 的排列 $i_1, i_2, \dots, i_n$ 使得

$$\min\{\sum_{k=1}^{n}(n-k)t_{i_k}\}$$

求解方法

- 贪心策略：加工时间短的先做。
- 如何描述这个方法？这个方法是否对所有的实例都能得到最优解？如何证明？这个方法的效率如何？

例2 排序算法的评价

已有的排序算法：考察元素比较次数

- 插入排序、冒泡排序：最坏和平均状况下都为 $O(n^2)$
- 快速排序：最坏状况为 $O(n^2)$，平均状况下为 $O(n\log n)$
- 堆排序、二分归并排序：最坏和平均状况下都为 $O(n\log n)$
- ……

问题

- 那个排序算法效率最高？
- 是否可以找到更好的算法？排序问题的计算难度如何估计？

例3 货郎问题

货郎问题：

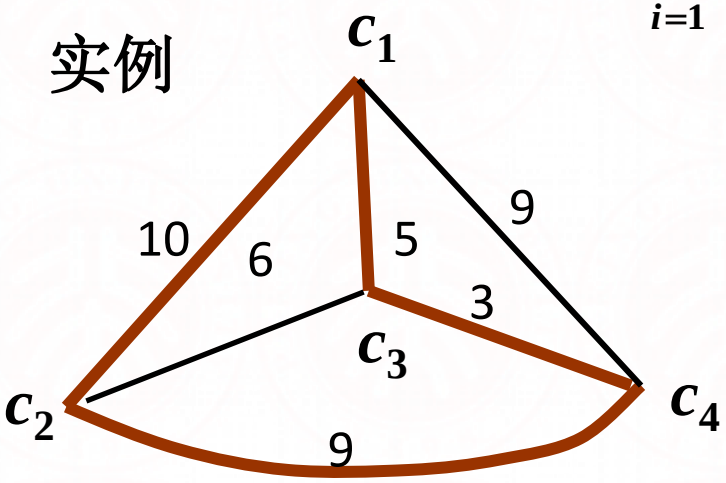
- 有穷个城市的集合$C = \{ c_1, c_2, \ldots, c_m\}$, 距离

$$d(c_i, c_j) = d(c_j, c_i) \in \mathbf{Z}^+, \quad 1 \le i < j \le m$$

- 求 $1, 2 \ldots, m$ 的排列 $k_1, k_2, \ldots, k_m$使得

$$\min\{\sum_{i=1}^{m-1} d(c_{k_i}, c_{k_{i+1}}) + d(c_{k_m}, c_{k_1})\}$$

实例



现状：至今没有找到有效的算法，
存在大量问题与它难度等价

问题：是否存在有效算法？

如何处理这类问题？

算法的定义

- 算法 是由定义明确的指令组成的有限序列表述的用于计算函数的有效方法。从初始状态和初始输入开始，通过指令的执行，经过有限数量的一系列定义明确的后续状态，最终产生输出并终止于结束状态。
  状态之间的转换不必是确定性的；例如对于随机算法，它涉及到计算过程中的随机输入。
  - 执行过程中的指令动作是定义明确的；
  - 初始化状态 -> 有限的状态转换 -> 结束状态；
  - 接收输入，产生输出。

- 严格形式化的算法定义还是一个未解决的问题。

算法的基本概念

- 问题
  - 需要回答的一般性提问，通常含有若干参数，对参数的一组赋值就得到问题的一个实例。
  - 区别于实际问题，往往是形式化的问题。
- 算法 A 解问题 P

  把问题 P 的任何实例作为算法 A 的输入，A 能够在有限步停机，并输出该实例的正确的解。

算法的基本概念

- 算法的时间复杂度

  针对问题指定基本运算，计数算法所做的基本运算次数

- 最坏情况下的时间复杂度

  算法求解输入规模为 n 的实例所需要的最长时间 $W(n)$

- 平均情况下的时间复杂度

  - 在指定输入的概率分布下，算法求解输入规模为 n 的实例所需要的平均时间 $A(n)$

- 其他时间复杂度：

  - 均摊时间（**Amortized**）、平滑分析（**Smoothed**）

检索问题的时间估计

检索问题

- 输入：非降顺序排列的数组 L，元素数为n； 数x
- 输出： j. 若x在L中，j 是 x 首次出现的序标；否则 $j = 0$
- 算法：顺序搜索
- 最坏情形下时间：$W(n) = n$
- 平均情形：实例集 S，实例 $I \in S$ 出现的概率是 p_I，算法对 I 的基本运算次数为 t_I，平均情况下的时间复杂度为

$$A(n) = \sum_{I \in S} t_I p_I$$

- 对于顺序搜索算法，假设 $x \in L$ 的概率为 p，x 在 L 不同位置等概分布，则

$$A(n) = \sum_{i=1}^{n} i \frac{p}{n} + (1-p)n = \frac{p(n+1)}{2} + (1-p)n$$

规范分析与实证分析

- 规范分析
  - 算法复杂性分析多是理论上的分析；
  - 算法用伪码描述，忽略编程语言、算法实现和运行环境对算法运行时间的影响；
  - 关注于当输入规模 n 趋近于无穷大时算法时间的渐进表示，例如：$O(n)$ 优于 $O(n^2)$
- 实证分析
  - 理论上更优的算法并非在实践上是最实用的算法；
  - 通过基准测试发现算法优化前后的性能改进；
  - 例：快速排序、线性规划单纯形法、斐波那契堆、SELECT算法

1.2、算法的分类

- 递归算法与迭代算法
  - 递归算法：通过反复对自身调用直到某种条件成立为止。例如函数式语言。
  - 迭代算法：通过循环或辅助以栈等数据结构来求解问题。与我们求解问题的常规思想类似。
- 递归算法和迭代算法是可以相互转换
  - 任何递归算法都可以利用栈转化为迭代算法；
  - 任何迭代算法也可以转化为没有循环的递归算法。

1.2、算法的分类

- 串行算法与并行算法
  - 串行算法：设计用于在串行计算机上执行的算法。
    - 算法的正确性可能依赖于算法指令按顺序一个接一个的执行。算法指令执行的乱序可能导致算法输出结果的不正确。
    - 串行算法的时间复杂性等于基本运算的执行次数。
  - 并行算法：设计可用于在多个处理器上同时协作执行的算法。很多迭代算法都可以并行化。
    - 并行算法的运行时间不仅与算法的并行加速比有关，还与计算节点间的通信开销相关。
    - **Amdahl**法则表明，大多数算法的并行加速比与处理器数之间并不能达到线性关系。

$$\frac{1}{f+\frac{1-f}{p}}$$

    - 多种并行计算模型：PRAM，格网，超立方体……

1.2、算法的分类

- 精确算法与近似算法
  - 精确算法：多数算法都是精确算法，算法将给出问题的明确的解。
  - 近似算法：对于NPC类问题，不太可能找到多项式时间的精确算法，只能在多项式时间内给出问题的近似解。
    - 近似比反应近似解与最优解之间的差距，是描述近似算法质量的重要指标。
    - 近似算法设计中常会采用贪心策略或随机策略。

1.2、算法的分类

- 离线算法和在线算法
  - 离线算法：基于在执行算法前输入数据已知的基本假设下，即具有问题完全信息前提下设计出来的算法。
    - 通常的算法都是离线算法：例如选择排序算法
  - 在线算法：通常顺序的一个一个的处理输入，并在得到每个输入时做出算法决策。
    - 在线算法例子：插入排序；页面淘汰；买冰鞋；
    - 加拿大旅行者问题：这个问题的目标是在一个有权图中以最小的代价到达一个目标节点，但这个有权图中有些边是不可靠的可能已经被剔除。然而一个旅行者只有到某个边的一个端点时才能确定该边是否已经被移除了。
    - 竞争比：在线算法的解与离线算法最优解之间的费用比。

1.3、算法设计技术

- 蛮力（穷举）——Brute-force or exhaustive search
- 分治策略——Divide and conquer
- 动态规划——Dynamic programming
- 贪心法——The greedy method
- 线性规划——Linear programming
- 问题规约——Reduction
- 搜索算法——Search and enumeration
  - 回溯(backtracking)、分支限界(branch and bound)
  - 随机算法(randomized algorithm)、启发式算法(heuristic algorithms)
- ……

1.4、算法的伪码描述

- 赋值语句：←
- 分支语句：if …then … [else…]
- 循环语句：while，for，repeat until
- 转向语句：goto
- 输出语句：return
- 调用：直接写过程的名字
- 注释：//…

例：求最大公约数

算法1.1 **Euclid(*m*,*n*)**

输入：非负整数 *m*, *n*，其中*m*与*n*不全为0

输出：*m* 与 *n* 的最大公约数

1. while *m* > 0 do

2. *r* ← *n* mod *m*

3. *n* ← *m*

4. *m* ← *r*

5. return *n*

例：改进的顺序检索

算法1.2 **Search(*L*, *x*)**

输入：数组*L*[1..*n*]，其元素按照从小到大排列，数 *x*.

输出：若 *x* 在*L*中，输出 *x* 的位置下标 *j*；否则输出0.

1. *j*←1

2. while *j*≤*n* and *x*>*L*[*j*] do

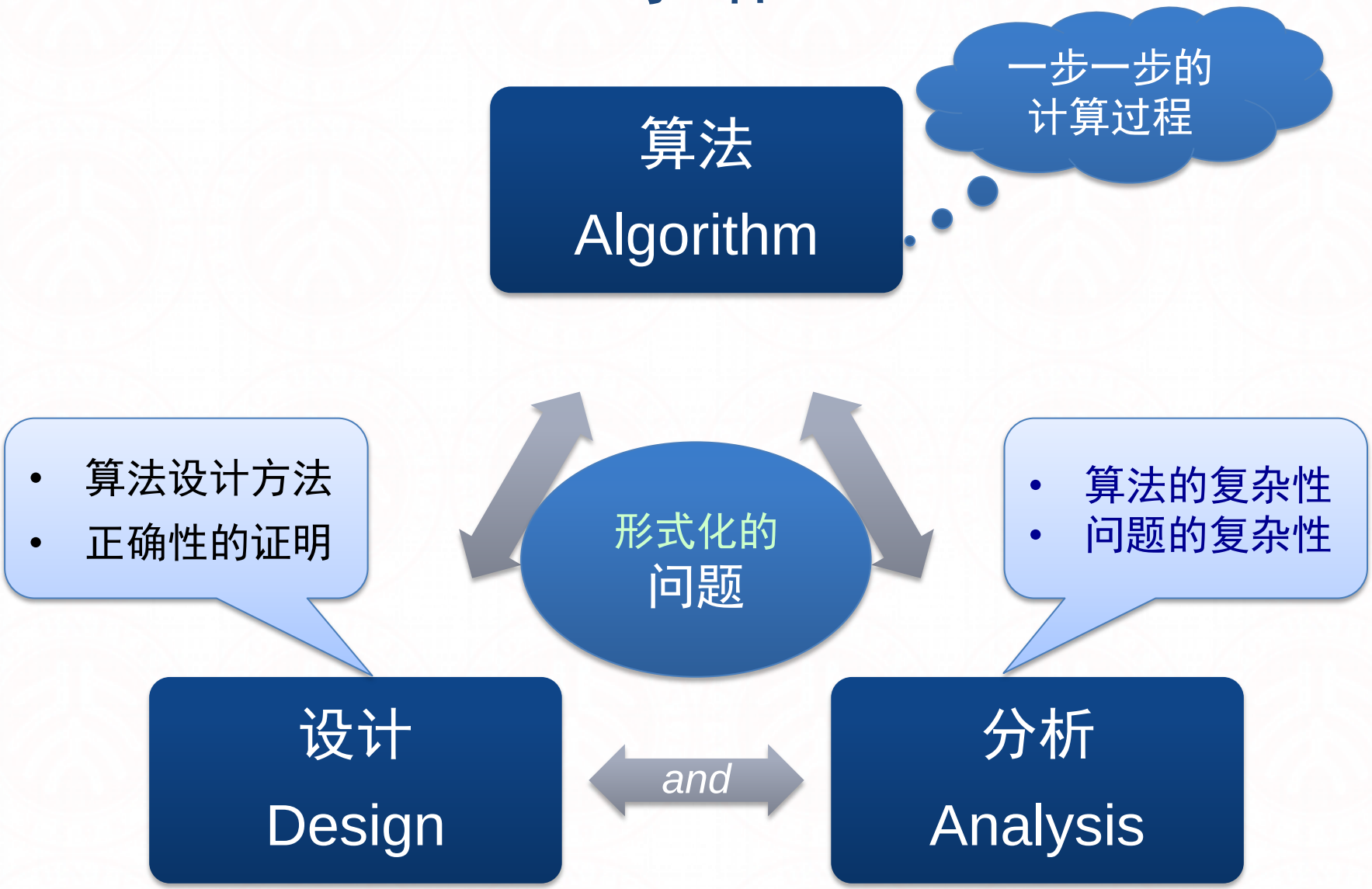
3. *j*←*j*+1

4. if *x*<*L*[*j*] or *j*>*n* then

5. *j*←0

6. return *j*

小结


算法
Algorithm
一步一步的
计算过程
形式化的
问题
• 算法设计方法
• 正确性的证明
• 算法的复杂性
• 问题的复杂性
设计
Design
and
分析
Analysis