算法概论

第一讲：基础知识

薛健

工程科学学院

Last Modified: 2019.11.10

主要内容

1 引言

2 背景知识

3 基本数据结构

4 算法分析基础

主要内容

1 引言
- 算法简介
- 关于本课程

2 背景知识

3 基本数据结构

4 算法分析基础

主要内容

什么是算法?

- 词源
  - 古代中国的数学著作《周髀算经》、《九章算术》⇒ 计算的方法 **(算法)**
  - 古代阿拉伯的数学著作 “Al-Kitāb al-mukhtaṣar fī hīsāb al-ğabr wa’l-muqābala”（大致意思 “The Compendious Book on Calculation by Completion and Balancing”）
  - 作者：Abū ‘Abdallāh Muḥammad ibn Mūsā al-Khwārizmī (花拉子米，约 780-850)
  - al-ğabr ⇒ algebra
  - al-Khwārizmī ⇒ Algorism 和 **Algorithm**

什么是算法?

词源

- 古代中国的数学著作《周髀算经》、《九章算术》⇒ 计算的方法 (**算法**)
- 古代阿拉伯的数学著作 “Al-Kitāb al-mukhtaṣar fī hīsāb al-ğabr wa’l-muqābala” (大致意思 “The Compendious Book on Calculation by Completion and Balancing”)
- 作者：Abū ‘Abdallāh Muḥammad ibn Mūsā al-Khwārizmī (花拉子米，约 780-850)
- al-ğabr ⇒ algebra
- al-Khwārizmī ⇒ Algorism 和 **Algorithm**

定义

- 算法是解某一特定问题的一组有穷规则的序列 (一个计算的具体步骤)。

Euclidean Algorithm

- 欧几里德算法 (Euclidean Algorithm, Euclid's Algorithm)：最早由古希腊数学家欧几里德在其《几何原本》第 VII 和第 X 卷中描述的求最大公约数 (Greatest Common Divisor) 的过程，从 20 世纪 50 年代开始，欧几里德所描述的这一过程被称为 "Euclidean Algorithm"，Algorithm 这个术语在学术上便具有了现在的含义。

Euclidean Algorithm

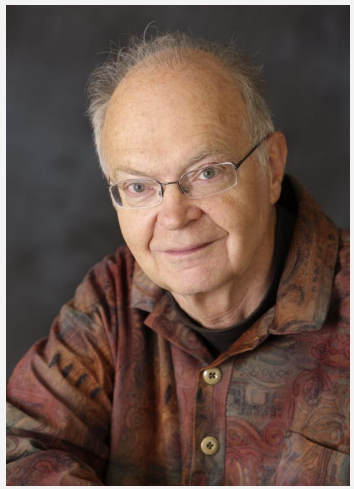
- 欧几里德算法 (Euclidean Algorithm, Euclid's Algorithm)：最早由古希腊数学家欧几里德在其《几何原本》第 VII 和第 X 卷中描述的求最大公约数 (Greatest Common Divisor) 的过程，从 20 世纪 50 年代开始，欧几里德所描述的这一过程被称为 "Euclidean Algorithm"，Algorithm 这个术语在学术上便具有了现在的含义。

Algorithm GCD(a, b)

```
1 while b ≠ 0 do
2 |   t ← b;
3 |   b ← a mod b;
4 |   a ← t;
5 end
6 return a;
```

算法的基本特征

Donald E. Knuth:



- 有限性 (Finiteness)：算法在执行有限步之后必须终止
- 确定性 (Definiteness)：算法的每个步骤都有精确的定义，即要执行的每一个动作都是清晰的、无歧义的
- 输入 (Input)：一个算法有 0 个或多个输入，作为算法开始执行前的初始值或初始状态
- 输出 (Output)：一个算法有 1 个或多个输出，这些输出与输入存在特定关系
- 能行性 (Effectiveness)：算法中的待实现的运算都是基本运算，原则上可由人用纸和笔在有限时间内精确地完成

一个例子

问题 假定 E 是包含 n 个元素的序列，给定键值 K，若 K 在序列 E 中，则返回其索引位置，否则返回 -1。

输入 E, n, K：E 为包含 n 个元素的序列 (索引从 0 到 $n-1$)，K 为某给定键值

输出 K 在序列 E 中的位置；或 -1 若未找到

方案 将 K 与序列 E 中的元素逐个比较，直到找到某一匹配或比较完所有序列元素

算法描述

Algorithm SeqSearch($E[], n, K$)

```
1 for i ← 0 to n − 1 do
2 |   if E[i] = K then
3 |   |   return i;
4 |   end
5 end
6 return −1;
```

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)
  - 元素 (键值) 的比较

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)
  - 元素 (键值) 的比较
- 最坏情况 (Worst-case) 分析

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)
  - 元素 (键值) 的比较
- 最坏情况 (Worst-case) 分析
  - 对于特定算法，其执行时间的长短与输入数据的多少 (或问题规模) 有关;
  - 最坏情况下的执行时间是输入数据量 n 的函数 $W(n)$;
  - 在上一例子中，显然 $W(n) = n$

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)
  - 元素 (键值) 的比较
- 最坏情况 (Worst-case) 分析
  - 对于特定算法，其执行时间的长短与输入数据的多少 (或问题规模) 有关;
  - 最坏情况下的执行时间是输入数据量 n 的函数 $W(n)$;
  - 在上一例子中，显然 $W(n) = n$
- 平均情况 (Average-case) 分析

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)
  - 元素 (键值) 的比较
- 最坏情况 (Worst-case) 分析
  - 对于特定算法，其执行时间的长短与输入数据的多少 (或问题规模) 有关;
  - 最坏情况下的执行时间是输入数据量 n 的函数 $W(n)$;
  - 在上一例子中，显然 $W(n) = n$
- 平均情况 (Average-case) 分析
  - 假设 K 出现在序列 E 中的概率为 q
  - 在平均情况下执行时间 $A(n) = n(1 - \frac{1}{2}q) + \frac{1}{2}q$ (*Why?*)

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)
  - 元素 (键值) 的比较
- 最坏情况 (Worst-case) 分析
  - 对于特定算法，其执行时间的长短与输入数据的多少 (或问题规模) 有关;
  - 最坏情况下的执行时间是输入数据量 n 的函数 $W(n)$;
  - 在上一例子中，显然 $W(n) = n$
- 平均情况 (Average-case) 分析
  - 假设 K 出现在序列 E 中的概率为 q
  - 在平均情况下执行时间 $A(n) = n(1 - \frac{1}{2}q) + \frac{1}{2}q$ (*Why?*)
- 最好情况 (Best-case) 分析

算法分析

如何衡量一个算法的效率 (工作量、执行时间……)?

- 基本操作 (Basic Operation)
  - 元素 (键值) 的比较
- 最坏情况 (Worst-case) 分析
  - 对于特定算法，其执行时间的长短与输入数据的多少 (或问题规模) 有关;
  - 最坏情况下的执行时间是输入数据量 n 的函数 $W(n)$;
  - 在上一例子中，显然 $W(n) = n$
- 平均情况 (Average-case) 分析
  - 假设 K 出现在序列 E 中的概率为 q
  - 在平均情况下执行时间 $A(n) = n(1 - \frac{1}{2}q) + \frac{1}{2}q$ (*Why?*)
- 最好情况 (Best-case) 分析
  - 在某些特定输入条件下执行时间最短
  - 多数情况下无实际意义

算法分析

还有优化的可能吗？

- 是否存在更快的算法？
- 如果序列 E 是有序序列：
  - 使用折半查找 (Binary Search) 会更快
  - $W(n) = \lceil \lg(n+1) \rceil$
  - 可以证明对于有序序列，折半查找是最快的

算法分析

还有优化的可能吗？

- 是否存在更快的算法？
- 如果序列 E 是有序序列：
  - 使用折半查找 (Binary Search) 会更快
  - $W(n) = \lceil \lg(n+1) \rceil$
  - 可以证明对于有序序列，折半查找是最快的

如何证明算法的正确性 (Correctness)？

几点结论

- 算法 $\neq$ 程序；算法设计 $\neq$ 程序设计
- 数据结构 + 算法 $\Rightarrow$ 程序
- 算法是程序中抽象的、一般化的、本质的部分
- 算法分析一般包括对算法执行时间和占用空间的分析，而时间复杂度是我们首要考虑的问题
  - **时间复杂度**：对算法执行时间的度量
  - **空间复杂度**：对算法执行过程中所占用空间的度量
- 算法复杂度一般表示成输入数据规模的函数，与具体计算环境无关，一般只关心其复杂度的**阶**
- 对于某一类特定问题来说，解决问题的算法其效率存在某一上限，而我们无论如何改进算法都无法逾越这一上限
  - 问题的复杂度存在下界 (Lower Bounds)
  - 如何找到这个下界?

主要内容

关于本课程

- 主要内容：计算机算法设计与分析的一般性理论和方法

关于本课程

- **主要内容**：计算机算法设计与分析的一般性理论和方法
  - **基础知识**：背景知识；基本数据结构；算法分析基础
  - **递归与分治法**：递归与数学归纳法；分治法原理及实例分析
  - **排序算法**：各种排序算法设计及其复杂度分析
  - **选择和检索**：选择算法及对手论证法；动态集合搜索及分摊时间分析
  - **高级设计与分析技术**：贪心算法；动态规划；字符串匹配算法
  - **图算法**：图的表示及其基本算法设计和分析
  - **$\mathcal{NP}$-完全问题介绍**：问题分类及多项式时间复杂度；$\mathcal{NP}$-完全性及其证明；$\mathcal{NP}$-完全问题

关于本课程

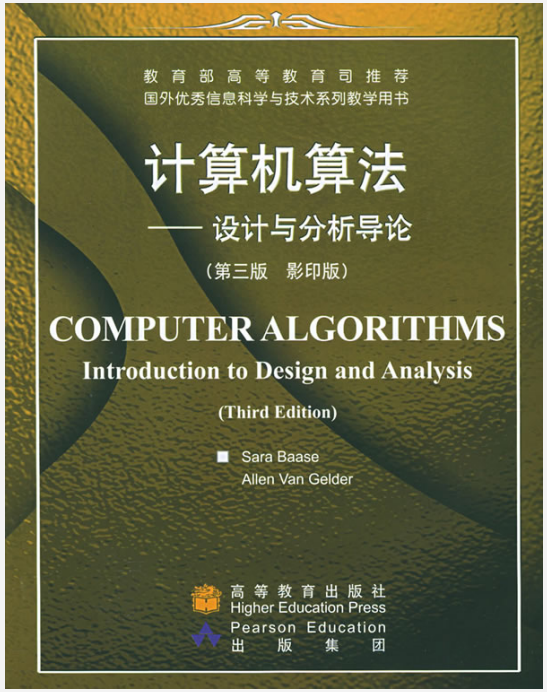
- **主要内容**：计算机算法设计与分析的一般性理论和方法
  - **基础知识**：背景知识；基本数据结构；算法分析基础
  - **递归与分治法**：递归与数学归纳法；分治法原理及实例分析
  - **排序算法**：各种排序算法设计及其复杂度分析
  - **选择和检索**：选择算法及对手论证法；动态集合搜索及分摊时间分析
  - **高级设计与分析技术**：贪心算法；动态规划；字符串匹配算法
  - **图算法**：图的表示及其基本算法设计和分析
  - **$\mathcal{NP}$-完全问题介绍**：问题分类及多项式时间复杂度；$\mathcal{NP}$-完全性及其证明；$\mathcal{NP}$-完全问题
- **目的**：建立良好的算法理论概念模型，培养使用计算机解决实际问题的能力

关于本课程

- **主要内容**：计算机算法设计与分析的一般性理论和方法
  - **基础知识**：背景知识；基本数据结构；算法分析基础
  - **递归与分治法**：递归与数学归纳法；分治法原理及实例分析
  - **排序算法**：各种排序算法设计及其复杂度分析
  - **选择和检索**：选择算法及对手论证法；动态集合搜索及分摊时间分析
  - **高级设计与分析技术**：贪心算法；动态规划；字符串匹配算法
  - **图算法**：图的表示及其基本算法设计和分析
  - **$\mathcal{NP}$-完全问题介绍**：问题分类及多项式时间复杂度；$\mathcal{NP}$-完全性及其证明；$\mathcal{NP}$-完全问题
- **目的**：建立良好的算法理论概念模型，培养使用计算机解决实际问题的能力
- **要求**：
  - 掌握算法复杂度分析的基本理论和方法，能够分析各类算法的时间和空间复杂度
  - 掌握算法设计的几类常用策略，能够根据具体问题及时间复杂度要求设计相应的算法

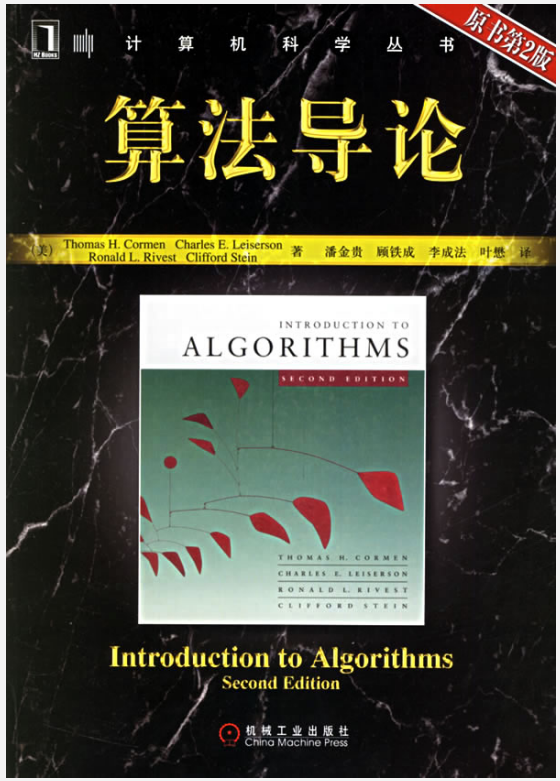
教材

Sara Baase and Allen Van Gelder.
《计算机算法——设计与分析导论》(第三版影印版) (***Computer Algorithms: Introduction to Design and Analysis (3rd Edition)***)
Pearson Education, 2000. 高等教育出版社, 2001.7.



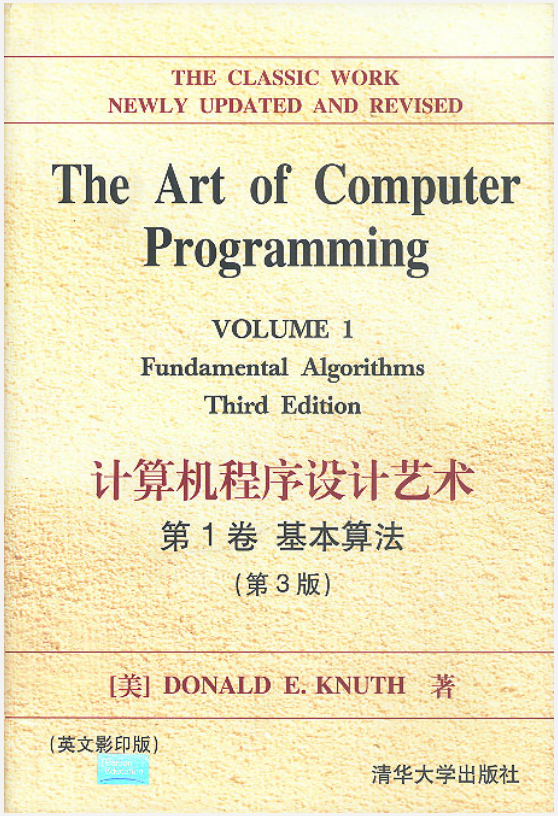
主要参考书目

- Thomas H. Cormen, Charles E. Leiserson, et al. 著, 潘金贵等译
  《算法导论》(*Introduction to Algorithms (2nd Edition)*)
  机械工业出版社, 2006.9



主要参考书目 (cont.)

- Donald E. Knuth
  The Art of Computer Programming - Volume 1: Fundamental Algorithms (3rd Edition)
  Addison-Wesley, 1997
  影印版：清华大学出版社, 2002
  中译本：《计算机程序设计艺术 第一卷 基本算法》. 苏运霖译, 国防工业出版社, 2002

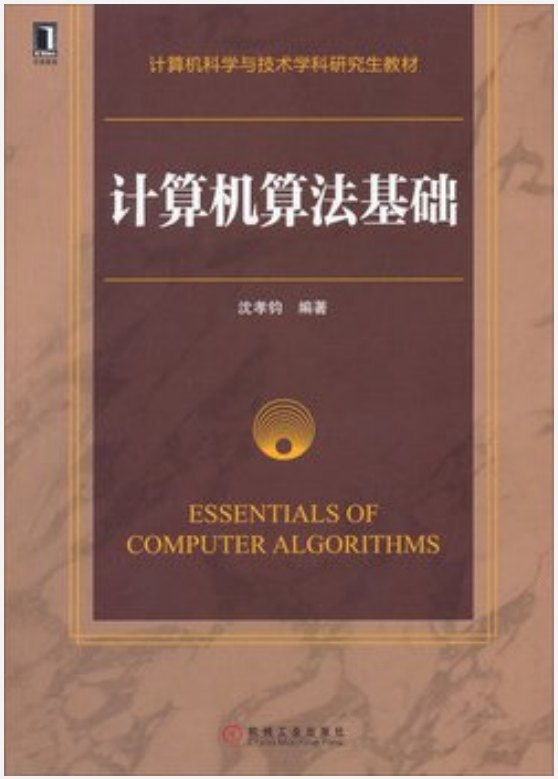


主要参考书目 (cont.)

沈孝钧

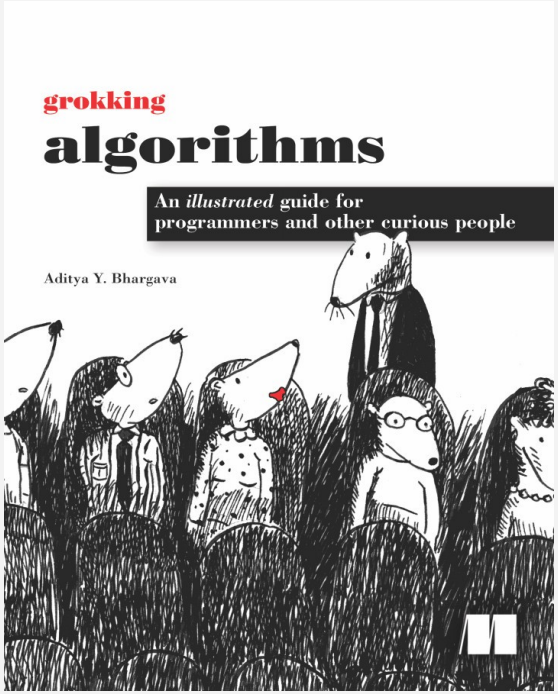
计算机科学与技术学科研究生教材：计算机算法基础 (第 1 版) (*Essentials of Computer Algorithms*)

机械工业出版社, 2014



主要参考书目 (cont.)

Aditya Y. Bhargava
Grokking Algorithms
Manning Publications, 2016
中译本:《算法图解》. 袁国忠译, 人民邮电出版社, 2017
https://www.manning.com/books/grokking-algorithms



实践工具

- 一种面向对象、解释型计算机编程语言 (高级动态编程语言)
- 具有近二十年的发展历史，成熟且稳定
- 语法简捷和清晰，尽量使用无异义的英语单词
- 设计哲学：“优雅”、“明确”、“简单”
- 具备垃圾回收功能，能够自动管理内存使用
- 虚拟机本身几乎可以在所有的操作系统中运行
- 入门：“简明 Python 教程” (http://woodpecker.org.cn/)

考核方式及其他

- 考核方式：
  - 期末考试 (60%)：课堂开卷
  - 课后习题 (40%)
    - 每一讲都会留一些课后习题，需要在 deadline 之前提交到课程网站或 ceital@163.com，或提交纸质版
    - 邮件主题及附件命名规则：算法作业 _<序号>_<姓名>_<学号>
    - 尽量采用 pdf 格式文件
    - 允许合作，严禁抄袭！

- 任课教师：薛健 (工程科学学院)
  - 电话：88256650 (office)，13810307841 (cell)
  - 邮箱：xuejian@ucas.ac.cn
  - Office Hour：每周日上午 9:30~10:30，人文楼 128
  - 课程资源：
    课程网站：http://sep.ucas.ac.cn

考核方式及其他

- 考核方式：
  - 期末考试 (60%)：课堂开卷
  - 课后习题 (40%)
    - 每一讲都会留一些课后习题，需要在 deadline 之前提交到课程网站或 ceital@163.com，或提交纸质版
    - 邮件主题及附件命名规则：算法作业 _<序号>_<姓名>_<学号>
    - 尽量采用 pdf 格式文件
    - 允许合作，严禁抄袭！
  - 期末成绩 $= \max(\text{考试成绩}, \text{考试成绩} \times 60\% + \text{作业成绩} \times 40\%)$
- 任课教师：薛健 (工程科学学院)
  - 电话：88256650 (office)，13810307841 (cell)
  - 邮箱：xuejian@ucas.ac.cn
  - Office Hour：每周日上午 9:30~10:30，人文楼 128
  - 课程资源：
    课程网站：http://sep.ucas.ac.cn

主要内容

1 引言

2 背景知识
- 集合论
- 逻辑
- 概率
- 代数

3 基本数据结构

4 算法分析基础

主要内容

集合 (Set)

- **集合**：具有某种属性的事物的全体，或是一些确定对象的汇合
- $a \in \mathbf{S}$；$a \notin \mathbf{S}$
- **集合的表示**：
  - $\mathbf{S} = \{a, b, c\}$
  - $\mathbf{S} = \{x | x > 3, x \in \mathbf{Z}\}$
- **空集**：$\mathbf{S} = \{\} = \varnothing$
- **子集**：$\forall x \in \mathbf{S}_1, x \in \mathbf{S}_2 : \mathbf{S}_1 \subseteq \mathbf{S}_2$
- **交集**：$\mathbf{S} \cap \mathbf{T} = \{x | x \in \mathbf{S} \text{ 且 } x \in \mathbf{T}\}$
- **并集**：$\mathbf{S} \cup \mathbf{T} = \{x | x \in \mathbf{S} \text{ 或 } x \in \mathbf{T}\}$
- **集合的特点**：确定性、互异性、无序性、完备性

集合的基数（势，Cardinality）

- **有限集**：由有限个元素组成的集合
  - 存在一个自然数 n，使得集合 $\mathbf{S}$ 中的元素与集合 $\{1, 2, \cdots, n\}$ 中的元素一一对应，记作 $|\mathbf{S}| = n$
  - 一个包含 n 个元素的集合共有 2^n 个不同子集
  - 一个包含 n 个元素的集合共有 $C_n^k = \frac{n!}{(n-k)!k!}$ 个不同的势为 k 的子集
- **无限集**：由无限个元素组成的集合
  - **可数集**：可以与自然数（正整数）集合 $\{1, 2, 3, \cdots\}$ 建立一一映射
  - **不可数集**：与自然数集合不存在一一映射（基数大于自然数集的基数）
  - **无限集的基数**：阿列夫数 $\aleph$。自然数集（可数）的基数记作 $\aleph_0$；实数集 $\mathbf{R}$（不可数）的基数记作 $2^{\aleph_0}$ 或 $\beth_1$

序列 (Sequence)

- **定义**:
  - 序列是按照某种顺序排成一列的对象
  - 其项属于集合 $\mathbf{S}$ 的**有限序列**是一个从 $\{1, 2, \cdots, n\}$ 到 $\mathbf{S}$ 的函数，这里 $n \geq 0$，也称做 n 元组
  - 其项属于 $\mathbf{S}$ 的**无限序列**是从 $\{1, 2, \cdots\}$ (自然数集合) 到 $\mathbf{S}$ 的函数
- **表示**:
  - $\mathbf{S}_1 = (a, b, c)$; $\mathbf{S}_2 = (b, c, a)$; $\mathbf{S}_3 = (a, a, b, c)$; $\mathbf{S}_4 = (a_1, a_2, \cdots, a_n) = (a_n)$
- **一些定义和结论**:
  - 一个给定序列的**子序列**是从给定序列中去除一些元素，而不改变其他元素之间相对位置而得到的
  - 一个元素互不相同的有限序列称做包含相同元素有限集的一个**排列**; 一个包含 n 个元素的有限集有 $n!$ 个不同排列
  - 若序列的项属于一个偏序集，则**单调递增序列**就是其中每个项都大于等于之前的项；若每个项都严格大于之前的项，这个序列就是**严格单调递增**的

元组和笛卡儿积

- 元组 (Tuples)是一个有限序列：
  - 有序对 (x, y)，三元组 (x, y, z)，四元组，五元组
  - k-元组：包含 k 个元素的有限序列
- 集合 $\mathbf{S}$ 和 $\mathbf{T}$ 的笛卡儿积 (Cartesian/Cross Product)定义为 $\mathbf{S} \times \mathbf{T} = \{(x, y) \mid x \in \mathbf{S}, y \in \mathbf{T}\}$
- 笛卡儿积的性质：
  - 对于任意集合 $\mathbf{S}$，根据定义有 $\mathbf{S} \times \varnothing = \varnothing \times \mathbf{S} = \varnothing$
  - 一般来说笛卡儿积不满足交换律和结合律
  - 笛卡儿积对集合的并和交满足分配律：
    - $\mathbf{A} \times (\mathbf{B} \cup \mathbf{C}) = (\mathbf{A} \times \mathbf{B}) \cup (\mathbf{A} \times \mathbf{C})$
    - $(\mathbf{B} \cup \mathbf{C}) \times \mathbf{A} = (\mathbf{B} \times \mathbf{A}) \cup (\mathbf{C} \times \mathbf{A})$
    - $\mathbf{A} \times (\mathbf{B} \cap \mathbf{C}) = (\mathbf{A} \times \mathbf{B}) \cap (\mathbf{A} \times \mathbf{C})$
    - $(\mathbf{B} \cap \mathbf{C}) \times \mathbf{A} = (\mathbf{B} \times \mathbf{A}) \cap (\mathbf{C} \times \mathbf{A})$
  - $|\mathbf{S} \times \mathbf{T}| = |\mathbf{S}||\mathbf{T}|$

关系 (Relations) 和函数 (Functions)

- 关系 (或二元关系) 是两个集合笛卡尔积的子集，即 $\mathbf{R} \subseteq \mathbf{S} \times \mathbf{T}$
- 例如：实数集上的“小于”关系可定义为
  $\{(x, y) \mid x \in \mathbf{R}, y \in \mathbf{R}, x < y\}$
- 关系的性质 ($\mathbf{R} \subseteq \mathbf{S} \times \mathbf{S}$):
  - 自反性 (reflexive): $\forall x \in \mathbf{S}, (x, x) \in \mathbf{R}$
  - 对称性 (symmetric): $\forall x, y \in \mathbf{S}, (x, y) \in \mathbf{R} \Rightarrow (y, x) \in \mathbf{R}$
  - 传递性 (transitive): $\forall x, y, z \in \mathbf{S}, (x, y) \in \mathbf{R}, (y, z) \in \mathbf{R} \Rightarrow (x, z) \in \mathbf{R}$
- 等价关系：同时满足自反性、对称性和传递性的二元关系

关系 (Relations) 和函数 (Functions) (cont.)

- 等价类：给定一个集合 $\mathbf{S}$ 和在 $\mathbf{S}$ 上的一个等价关系 $\mathbf{R}$，则 $\mathbf{S}$ 中的一个元素 x 的等价类是在 $\mathbf{S}$ 中等价于 x 的所有元素构成的子集，即 $[x] = \{y \in \mathbf{S} \mid x\mathbf{R}y\}$
- 从输入元素集合 $\mathbf{X}$ 到可能的输出元素集合 $\mathbf{Y}$ 的函数 f (记作 $f : \mathbf{X} \to \mathbf{Y}$) 是 $\mathbf{X}$ 与 $\mathbf{Y}$ 的关系，满足如下条件：
  - f 是完全的：对集合 $\mathbf{X}$ 中任一元素 x 都有集合 $\mathbf{Y}$ 中的元素 y 满足 xfy
  - f 是多对一的：若 xfy 且 xfz，则 $y = z$

主要内容

逻辑

- 逻辑是研究“有效推论和证明的原则与标准”的一门学科，做为一门形式科学，逻辑透过对推论的形式系统与自然语言中的论证等来研究并分类命题 (statement) 与论证的结构
- 最简单的命题称为原子式 (atomic formula)
- 更复杂的命题可以由原子式和逻辑运算符 (或连接符，logical connectives) 组合生成，基本运算符包括：
  - 非 ($\neg$)：$\neg A$ 为真当且仅当 A 为假
  - 与 ($\wedge$)：$A \wedge B$ 为真当且仅当 A 为真且 B 为真
  - 或 ($\vee$)：$A \vee B$ 为真当且仅当 A 为真或 B 为真或二者都为真
  - 蕴涵 ($\rightarrow$ 或 $\Rightarrow$，读作 “implies”)：$A \rightarrow B$ 读作 “A implies B” 或 “if A then B”
- $A \rightarrow B$ 等价于 $\neg A \vee B$
- De Morgan’s laws
  - $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
  - $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$

量词 (Quantifier，$\forall$ 和 $\exists$)

- 全称量词 $\forall$，“for all ...”:
  - $\forall x P(x)$ 为真 $\Leftrightarrow$ $P(x)$ 对于任意 (所有) x 为真
- 存在量词 $\exists$，“there exist ...”:
  - $\exists x P(x)$ 为真 $\Leftrightarrow$ 存在 x 使 $P(x)$ 为真
- $\forall x P(x) \Leftrightarrow \neg \exists x (\neg P(x))$
- $\exists x P(x) \Leftrightarrow \neg \forall x (\neg P(x))$
- $\forall x (A(x) \rightarrow B(x))$：对于任意 x，如果 $A(x)$ 成立则 $B(x)$ 成立

证明：反例，逆否命题，反证

- **反例证明**：
  - 要证明 $\forall x(A(x) \to B(x))$ 为假，只需举出一个反例，即存在某个 x 使得 $A(x)$ 为真而 $B(x)$ 为假
  - $\neg(\forall x(A(x) \to B(x))) \Leftrightarrow \exists x(A(x) \wedge \neg B(x))$
- **逆否命题证明**：
  - 要证明 $A \to B$，只需证明 $\neg B \to \neg A$
- **反证**：
  - 要证明 $A \to B$，只需假设 $\neg B$，然后证明 B 本身，即证明 $(A \wedge \neg B) \to B$
  - $A \to B \Leftrightarrow (A \wedge \neg B) \to B$
  - $A \to B \Leftrightarrow (A \wedge \neg B)$ 为假
  - 假设 $(A \wedge \neg B)$ 为真，找到矛盾 (如 $A \wedge \neg A$)，然后可得结论 $(A \wedge \neg B)$ 为假，即 $A \to B$ 成立

反证法例子

$$(B \wedge (B \to C)) \to C$$

反证法例子

$(B \wedge (B \to C)) \to C$

Proof.

假设 $\neg C$

$\neg C \wedge (B \wedge (B \to C))$

$\Rightarrow \neg C \wedge (B \wedge (\neg B \vee C))$

$\Rightarrow \neg C \wedge ((B \wedge \neg B) \vee (B \wedge C))$

$\Rightarrow \neg C \wedge ((B \wedge C))$

$\Rightarrow \neg C \wedge C \wedge B$

$\Rightarrow$ 矛盾

$\Rightarrow C$ □

推理规则

- 推理规则 (推论规则) 是构造有效推论的方案。这些方案建立在一组叫做前提的公式和叫做结论的断言之间的语法关系。这些语法关系用于推理过程中，新的真的断言从其他已知的断言得出

- 证明系统形成自一组规则，它们可以被链接在一起形成证明或推导。任何推导都只有一个最终结论，它是要证明或推导的陈述。如果在推导中留下了未满足的前提，则推导就是假言陈述："如果前提成立，那么结论成立。"

- 常用规则：
  - 肯定前件式 (Modus ponens)：if $\{B \to C \text{ and } B\}$ then $\{C\}$
  - 否定后件式 (Modus tollens)：if $\{B \to C \text{ and } \neg C\}$ then $\{\neg B\}$
  - 三段论法 (Syllogism)：if $\{A \to B \text{ and } B \to C\}$ then $\{A \to C\}$
  - 否定消除：if $\{B \to C \text{ and } \neg B \to C\}$ then $\{C\}$

布尔(代数)逻辑

- 在包含两个元素 0(逻辑假)和 1(逻辑真)的集合 $\mathbf{B}$ 上定义了两个二元运算：$\wedge$(或记作 $\cdot$，逻辑与)和 $\vee$(或记作 $+$，逻辑或)以及一个一元运算 $\neg$(或记作 $\sim$，逻辑非)
- 闭合：如果 $x, y \in \mathbf{B}$，则 $x \wedge y \in \mathbf{B}$, $x \vee y \in \mathbf{B}$, $\neg x \in \mathbf{B}$
- 单位元：$x \wedge 1 = x$, $x \vee 0 = x$
- 交换律：$x \wedge y = y \wedge x$, $x \vee y = y \vee x$
- 结合律：$x \wedge (y \wedge z) = (x \wedge y) \wedge z$, $x \vee (y \vee z) = (x \vee y) \vee z$
- 分配律：$x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$, $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$
- 吸收律：$x \vee (x \wedge y) = x$, $x \wedge (x \vee y) = x$
- 互补律：$x \wedge \neg x = 0$, $x \vee \neg x = 1$

重言式 (Tautology) 和真值表

一个逻辑公式，不论其组成部分的命题变项为真的可能性怎么样，它总是为真。在布尔代数中发现重言式的最简单的方法是使用真值表。

重言式 (Tautology) 和真值表

一个逻辑公式，不论其组成部分的命题变项为真的可能性怎么样，它总是为真。在布尔代数中发现重言式的最简单的方法是使用真值表。

Example

$(B \wedge (B \to C)) \to C$ 是重言式。

B	C	$B \to C$	$(B \wedge (B \to C))$	$(B \wedge (B \to C)) \to C$
0	0	1	0	1
0	1	1	0	1
1	0	0	0	1
1	1	1	1	1

用推理规则证明

$$(B \wedge (B \to C)) \to C$$

用推理规则证明

$(B \wedge (B \to C)) \to C$

Proof.

$(B \wedge (B \to C)) \to C$
$\Rightarrow \neg(B \wedge (B \to C)) \vee C$
$\Rightarrow \neg(B \wedge (\neg B \vee C)) \vee C$
$\Rightarrow \neg((B \wedge \neg B) \vee (B \wedge C)) \vee C$
$\Rightarrow \neg(B \wedge C) \vee C$
$\Rightarrow \neg B \vee \neg C \vee C$
$\Rightarrow$ True (tautology) □

主要内容

事件 (Event)

- **单位事件**：在一次随机试验中可能发生的不能再细分的结果
- **事件空间**：在随机试验中可能发生的所有单位事件的集合 $\mathrm{U} = \{s_1, s_2, \cdots, s_k\}$
- **单位事件 s_i 概率**：取实数 $P(s_i)$ 使得
  1. $0 \leq P(s_i) \leq 1,\ 1 \leq i \leq k$
  2. $P(s_1) + P(s_2) + \cdots + P(s_k) = 1$
- **随机事件**：事件空间的子集 $\mathrm{S} \subseteq \mathrm{U}$，其概率 $P(\mathrm{S}) = \sum_{s_i \in \mathrm{S}} P(s_i)$
- **必然事件**：$\mathrm{U} = \{s_1, s_2, \cdots, s_k\},\ P(\mathrm{U}) = 1$
- **不可能事件**：$\varnothing,\ P(\varnothing) = 0$
- **互补事件**：“$\bar{\mathrm{S}}$”，$\mathrm{U} - \mathrm{S}$，$P(\bar{\mathrm{S}}) = P(\mathrm{U} - \mathrm{S}) = 1 - P(\mathrm{S})$

条件概率

- **条件概率**：事件 S 在另外一个事件 T 已经发生条件下的发生概率，表示为 $P(\mathrm{S} \mid \mathrm{T})$，读作“在 T 条件下 S 的概率”
- **联合概率**：两个事件共同发生的概率，表示为 $P(\mathrm{S} \cap \mathrm{T})$ 或 $P(\mathrm{S}, \mathrm{T})$
- **条件概率的计算**：

$$P(\mathrm{S} \mid \mathrm{T}) = \frac{P(\mathrm{S} \cap \mathrm{T})}{P(\mathrm{T})} = \frac{\sum_{s_i \in \mathrm{S} \cap \mathrm{T}} P(s_i)}{\sum_{s_j \in \mathrm{T}} P(s_j)}$$

- **统计独立性**：当且仅当两个随机事件 S 与 T 满足 $P(\mathrm{S} \cap \mathrm{T}) = P(\mathrm{S})P(\mathrm{T})$，称它们是统计独立 (statistically independent) 或随机独立 (stochastically independent)

随机变量及其期望值

- **随机变量**：事件空间上的实值函数 $f : \mathrm{U} \to \mathbf{R}$，即为事件空间中的每一个单位事件 e 赋予一个实数值 $f(e)$
  - 例如：随机掷两个骰子，整个事件空间可以由 36 个元素组成：$\mathrm{U} = \{(i, j) \mid i = 1, \cdots, 6; j = 1, \cdots, 6\}$，这里可以构成多个随机变量，比如随机变量 X：获得的两个骰子的点数和；或者随机变量 Y：获得的两个骰子的点数差绝对值。随机变量 X 可以有 11 个整数值，而随机变量 Y 只有 6 个整数值

- **随机变量的期望值**：是试验中每次可能结果的概率乘以其结果的总和；换句话说，期望值是随机试验在同样的机会下重复多次的结果计算出的等同“期望”的平均值

$$E(f) = \sum_{e \in \mathrm{U}} f(e)P(e)$$

条件期望和期望定律

- **条件期望**：在给定事件 S 已经发生的条件下，随机变量 f 的期望值

$$E(f \mid \mathrm{S}) = \sum_{e \in \mathrm{S}} f(e) P(e \mid \mathrm{S})$$

- **期望定律**：f 和 g 是定义在事件空间 U 上的随机变量，则对任意随机事件 S，有

$$E(\alpha f + \beta g) = \alpha E(f) + \beta E(g)$$

$$E(f) = P(\mathrm{S}) E(f \mid \mathrm{S}) + P(\bar{\mathrm{S}}) E(f \mid \bar{\mathrm{S}})$$

主要内容

不等式和取整函数

- 不等式的基本性质
  - 传递性：if $(a \leq b)$ and $(b \leq c)$ then $(a \leq c)$
  - 可加性：if $(a \leq b)$ and $(c \leq d)$ then $(a + c \leq b + d)$
  - 乘法 (正向缩放)：if $(a \leq b)$ and $(\alpha > 0)$ then $(\alpha a \leq \alpha b)$
- 取整函数 (Floor and Ceiling Functions)
  - 向下取整：$\lfloor x \rfloor$ 或记作 $Floor(x)$：小于等于 x 的最大整数
  - 向上取整：$\lceil x \rceil$ 或记作 $Ceiling(x)$：大于等于 x 的最小整数

关于取整函数

- 常用计算规则
  - $x-1<\lfloor x\rfloor \le x \le \lceil x\rceil < x+1$
  - $\lfloor -x\rfloor = -\lceil x\rceil$; $\lceil -x\rceil = -\lfloor x\rfloor$
  - $\lfloor x\rfloor = n \iff n \le x < n+1$
    $\lfloor x\rfloor = n \iff x-1 < n \le x$
    $\lceil x\rceil = n \iff n-1 < x \le n$
    $\lceil x\rceil = n \iff x \le n < x+1$
  - $\lfloor x+n\rfloor = \lfloor x\rfloor + n$; $\lceil x+n\rceil = \lceil x\rceil + n$
  - $x < n \iff \lfloor x\rfloor < n$
    $n < x \iff n < \lceil x\rceil$
    $x \le n \iff \lceil x\rceil \le n$
    $n \le x \iff n \le \lfloor x\rfloor$
  - $\left\lfloor \sqrt{\lfloor x\rfloor}\right\rfloor = \lfloor \sqrt{x}\rfloor$; $\left\lceil \sqrt{\lceil x\rceil}\right\rceil = \lceil \sqrt{x}\rceil$

关于取整函数 (cont.)

Theorem

$f(x)$ 连续、递增，且满足：$f(x) \in \mathbf{N} \longrightarrow x \in \mathbf{N}$，则：

1. $\lfloor f(\lfloor x \rfloor) \rfloor = \lfloor f(x) \rfloor$
2. $\lceil f(\lceil x \rceil) \rceil = \lceil f(x) \rceil$

由此，可得：

$$\left\lfloor \frac{\lfloor x \rfloor + m}{n} \right\rfloor = \left\lfloor \frac{x+m}{n} \right\rfloor; \quad \left\lceil \frac{\lceil x \rceil + m}{n} \right\rceil = \left\lceil \frac{x+m}{n} \right\rceil$$

参考：

Ronald L. Graham, Donald E. Knuth, Oren Patashnik
Concrete Mathematics: A Foundation for Computer Science
《具体数学：计算机科学基础》(英文版 · 第 2 版)
机械工业出版社, 2002.8

对数

- **对数**：$\log_b x$ 称为为以 b 为底的 x 的对数，其中 $b > 0, b \neq 1, x > 0$，它的值是使得 $b^L = x$ 成立的实数 L

- **对数的性质**：
  - 对数函数 $\log_b x$ 是严格单调函数，当 $0 < b < 1$ 时单调递减，当 $b > 1$ 时单调递增
  - 对数函数是单射，即 $\log_b x = \log_b y \Rightarrow x = y$，其反函数为 b^x
  - $\log_b 1 = 0;\ \log_b b = 1;\ \log_b x^a = a \log_b x$
  - $\log_b xy = \log_b x + \log_b y$
  - $x^{\log y} = y^{\log x}$
  - $\log_b x = \dfrac{\log_a x}{\log_a b}$

数列与级数

- **数列**：按照某种规则排列的一组数，如 $a_1, a_2, \cdots, a_n, \cdots$
- **级数**：数列求和，即 $a_1 + a_2 + \cdots + a_n + \cdots$，记作 $\sum_{k=1}^{\infty} a_k$
- **常用级数的部分和**
  - $\sum_{i=1}^{n} i = \frac{1}{2} n(n+1)$
  - $\sum_{i=1}^{n} i^2 = \frac{1}{6} n(n+1)(2n+1) \approx \frac{n^3}{3}$, $\sum_{i=1}^{n} i^k \approx \frac{n^{k+1}}{k+1}$
  - $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$
  - $\sum_{i=1}^{n} i2^i = (n-1)2^{n+1} + 2$

单调函数与凸函数

Definition (单调函数 (monotonic function))

$f(x)$ 为单调增函数，当且仅当 $x \leq y \Rightarrow f(x) \leq f(y)$；$f(x)$ 为单调减函数，当且仅当 $-f(x)$ 为单调增函数。

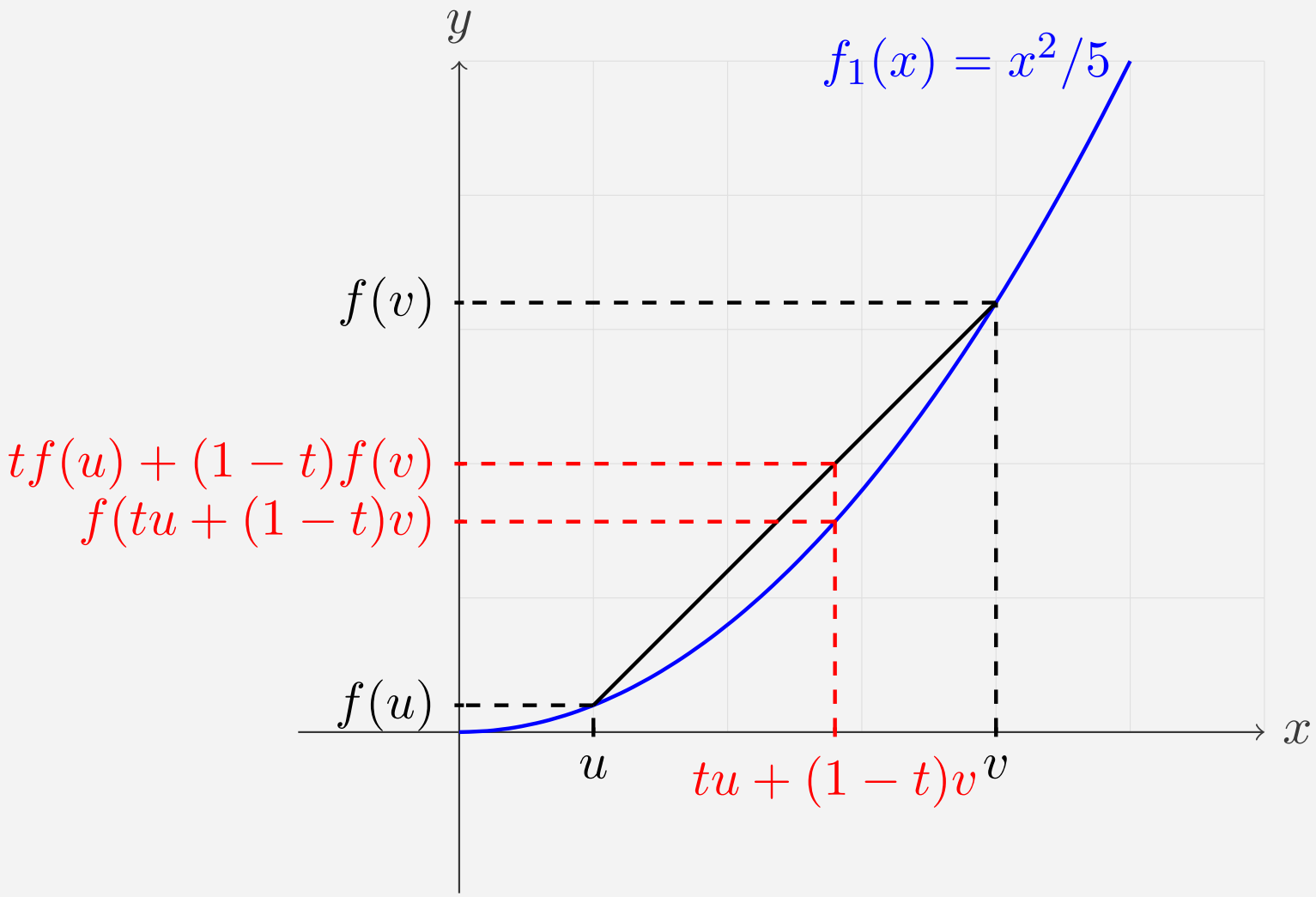
Definition (凸函数 (convex function))

若函数 $f(x)$，$\forall u, v \in$ 定义域 $\mathbf{D}, t \in [0,1]$ 都有

$$f(tu + (1-t)v) \leq tf(u) + (1-t)f(v)$$

则称 $f(x)$ 为凸 (convex) (下凸，上凹) 函数；若 $-f(x)$ 为凸函数，则称 $f(x)$ 为凹 (concave) (上凸，下凹) 函数

单调函数与凸函数(图示)



凸函数的判定

Lemma

1. $f(x)$ 是实数集 $\mathbf{R}$ 上的连续函数，则 $f(x)$ 为凸函数当且仅当 $\forall u, v \in \mathbf{R}$

$$f\big(\frac{1}{2}(u+v)\big) \leq \frac{1}{2}\big(f(u)+f(v)\big)$$

2. 定义在整数集 $\mathbf{Z}$ 上的函数 $f(n)$ 为凸函数，当且仅当 $\forall n \in \mathbf{Z}$ 有

$$f(n+1) \leq \frac{1}{2}\big(f(n)+f(n+2)\big)$$

单调函数和凸函数的判定

Lemma

1. $f(n)$ 是定义在整数集上的函数，$f^*(x)$ 是 $f(n)$ 采用相邻点的线性插值在实数集上的扩展，则有
   (a) $f(n)$ 是单调增 (减) 函数当且仅当 $f^*(x)$ 是单调增 (减) 函数
   (b) $f(n)$ 是凸函数当且仅当 $f^*(x)$ 是凸函数
2. 如果 $f(x)$ 的一阶导数 $f'(x)$ 存在且非负，则 $f(x)$ 为单调增函数
3. 如果 $f(x)$ 的一阶导数 $f'(x)$ 存在且为单调增函数，则 $f(x)$ 为凸函数
4. 如果 $f(x)$ 的二阶导数 $f''(x)$ 存在且非负，则 $f(x)$ 为凸函数

用积分求和

- 一些常用积分公式

$$\int_0^n x^k dx = \frac{1}{k+1} n^{k+1}, \quad \int_0^n e^{ax} dx = \frac{1}{a}(e^{an} - 1),$$

$$\int_1^n x^k \ln(x) dx = \frac{1}{k+1} n^{k+1} \ln(n) - \frac{1}{(k+1)^2} n^{k+1}$$

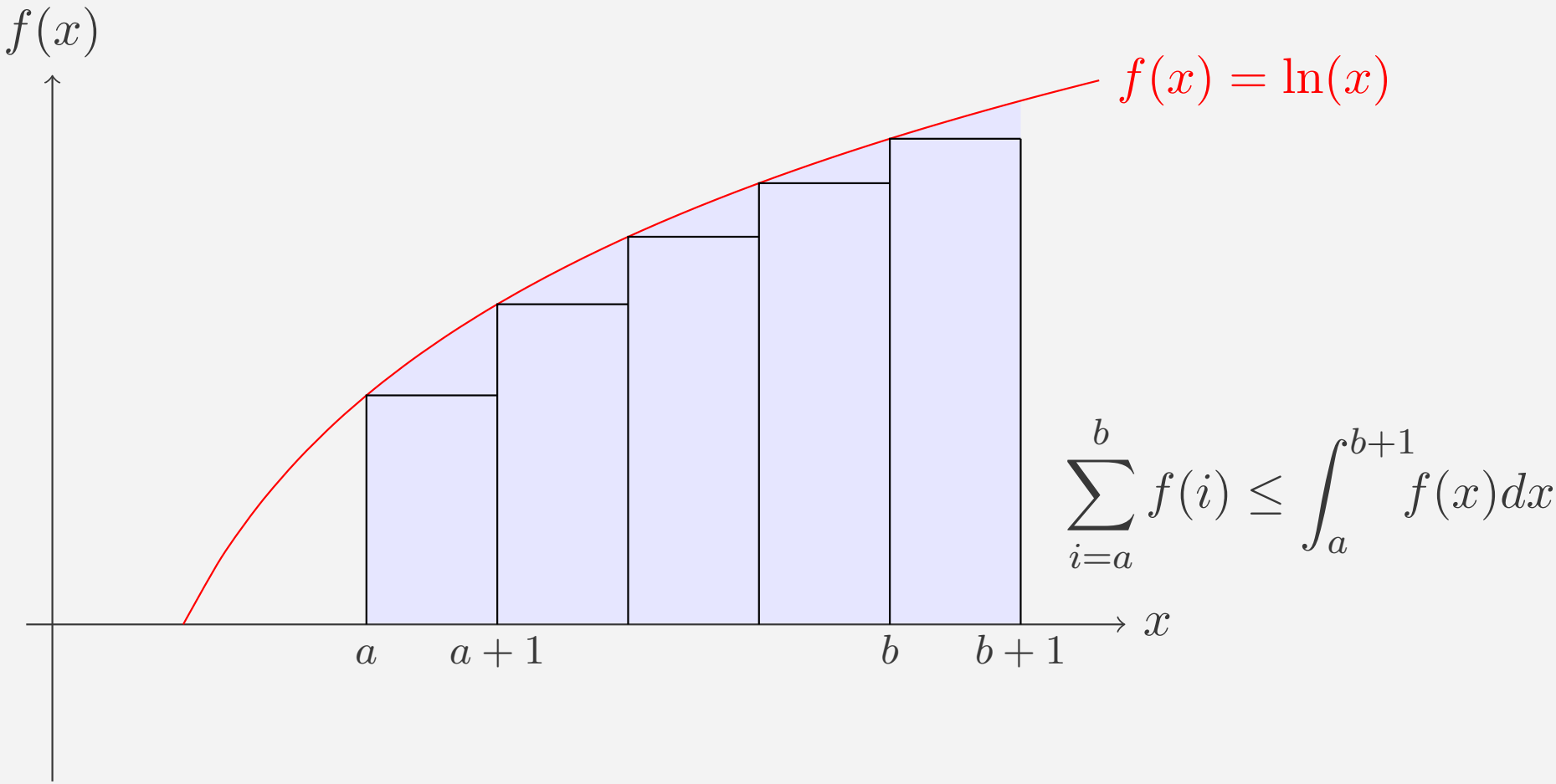
- 对于一些数列的求和，使用以下结论往往可以用积分来计算其近似值(或给出其上下界)
  - 若 $f(x)$ 单调递增，即 $x \le y \Rightarrow f(x) \le f(y)$，则

$$\int_{a-1}^{b} f(x)dx \le \sum_{i=a}^{b} f(i) \le \int_{a}^{b+1} f(x)dx$$

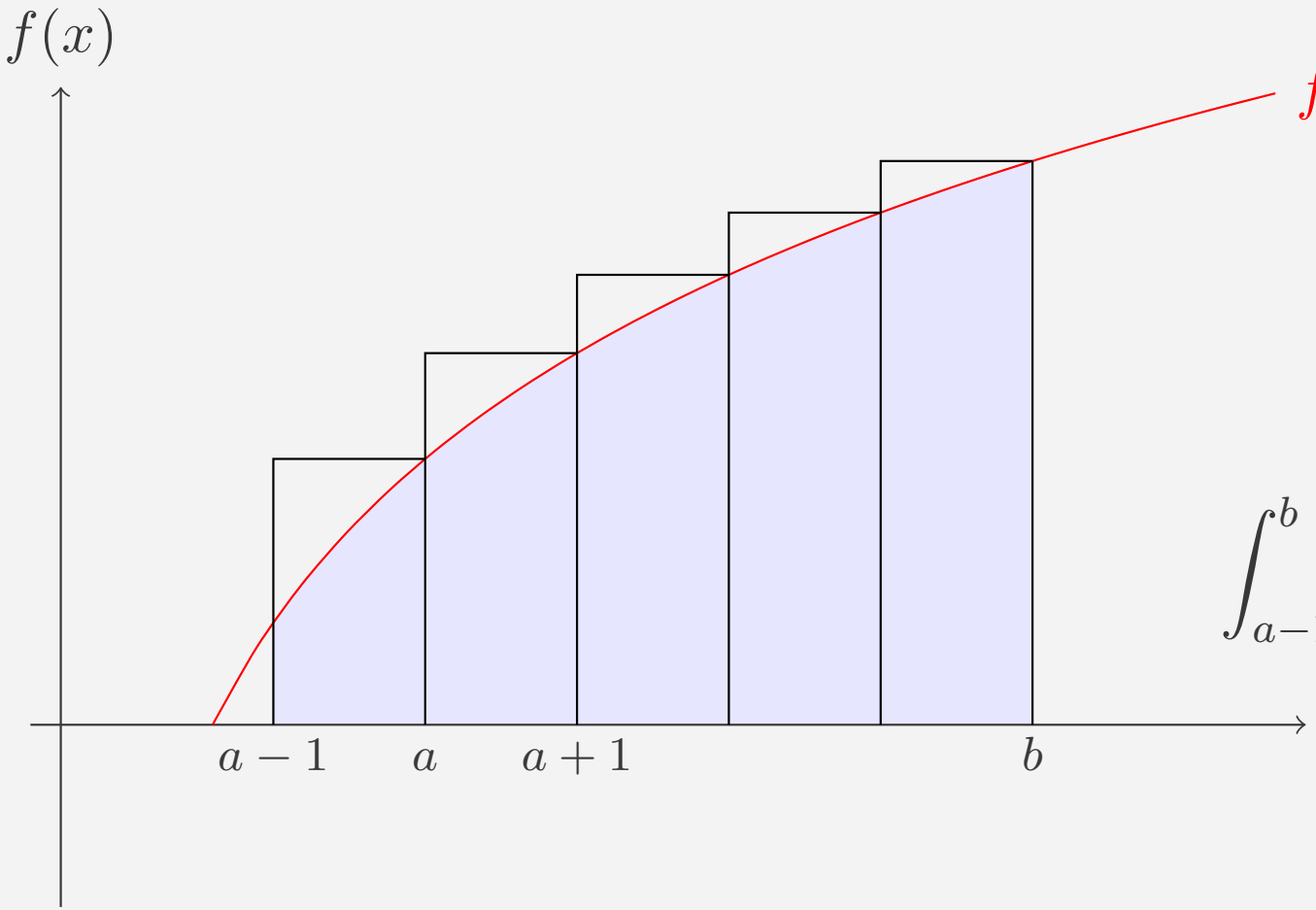
  - 类似地，若 $f(x)$ 单调递减，则

$$\int_{a}^{b+1} f(x)dx \le \sum_{i=a}^{b} f(i) \le \int_{a-1}^{b} f(x)dx$$

用积分求近似和 (图示)



用积分求近似和 (图示)



$$\int_{a-1}^{b} f(x)dx \leq \sum_{i=a}^{b} f(i)$$

用积分求近似和

Example

求 $\sum_{i=1}^{n} \frac{1}{i}$ 近似上下界：

用积分求近似和

Example

求 $\sum_{i=1}^{n} \frac{1}{i}$ 近似上下界：

$$\sum_{i=1}^{n} \frac{1}{i} \leq 1 + \int_{1}^{n} \frac{dx}{x} = 1 + \ln x \Big|_{1}^{n} = 1 + \ln n - \ln 1 = \ln n + 1$$

用积分求近似和

Example

求 $\sum_{i=1}^{n} \frac{1}{i}$ 近似上下界：

$$\sum_{i=1}^{n} \frac{1}{i} \leq 1 + \int_{1}^{n} \frac{dx}{x} = 1 + \ln x \Big|_{1}^{n} = 1 + \ln n - \ln 1 = \ln n + 1$$

$$\sum_{i=1}^{n} \frac{1}{i} \geq \int_{1}^{n+1} \frac{dx}{x} = \ln x \Big|_{1}^{n+1} = \ln(n+1) - \ln 1 = \ln(n+1)$$

用积分求近似和

Example

求 $\sum_{i=1}^{n} \frac{1}{i}$ 近似上下界：

$$\sum_{i=1}^{n} \frac{1}{i} \leq 1 + \int_{1}^{n} \frac{dx}{x} = 1 + \ln x \Big|_{1}^{n} = 1 + \ln n - \ln 1 = \ln n + 1$$

$$\sum_{i=1}^{n} \frac{1}{i} \geq \int_{1}^{n+1} \frac{dx}{x} = \ln x \Big|_{1}^{n+1} = \ln(n+1) - \ln 1 = \ln(n+1)$$

$$\ln(n+1) \leq \sum_{i=1}^{n} \frac{1}{i} \leq \ln n + 1$$

关于调和级数 $\sum_{i=1}^{\infty} \frac{1}{i}$

- 其部分和称为调和数 (Harmonic number)，记作 $H_n = \sum_{i=1}^{n} \frac{1}{i}$
- Euler 给出了其积分形式：$H_n = \int_0^1 \frac{1-x^n}{1-x} dx$ (用数学归纳法可以证明)
- 一个更精确的近似：$\sum_{i=1}^{n} \frac{1}{i} \approx \ln(n) + \gamma$ 其中 $\gamma = \lim_{n\to\infty} \left[\left(\sum_{i=1}^{n} \frac{1}{i} \right) - \ln(n) \right] \approx 0.5772156649\ldots$ 称为欧拉-马修罗尼常数 (Euler-Mascheroni constant)

用积分求近似和

Example

求 $\sum_{i=1}^{n} \lg i$ 下界：

用积分求近似和

Example

求 $\sum_{i=1}^{n} \lg i$ 下界：

$$\begin{aligned}
\sum_{i=1}^{n} \lg i &= 0 + \sum_{i=2}^{n} \lg i \geq \int_1^n \lg x dx \\
\int_1^n \lg x dx &= \int_1^n (\lg e)(\ln x) dx = (\lg e) \int_1^n \ln x dx \\
&= (\lg e)(x \ln x - x)\Big|_1^n = (\lg e)(n \ln n - n + 1) \\
&= n \lg n - (n-1) \lg e
\end{aligned}$$

用积分求近似和

Example

求 $\sum_{i=1}^{n} \lg i$ 下界：

$$\begin{aligned}
\sum_{i=1}^{n} \lg i &= 0 + \sum_{i=2}^{n} \lg i \geq \int_1^n \lg x dx \\
\int_1^n \lg x dx &= \int_1^n (\lg e)(\ln x) dx = (\lg e) \int_1^n \ln x dx \\
&= (\lg e)(x \ln x - x)\Big|_1^n = (\lg e)(n \ln n - n + 1) \\
&= n \lg n - (n-1) \lg e
\end{aligned}$$

$$\sum_{i=1}^{n} \lg i \geq n \lg n - 1.443(n-1)$$

函数分类

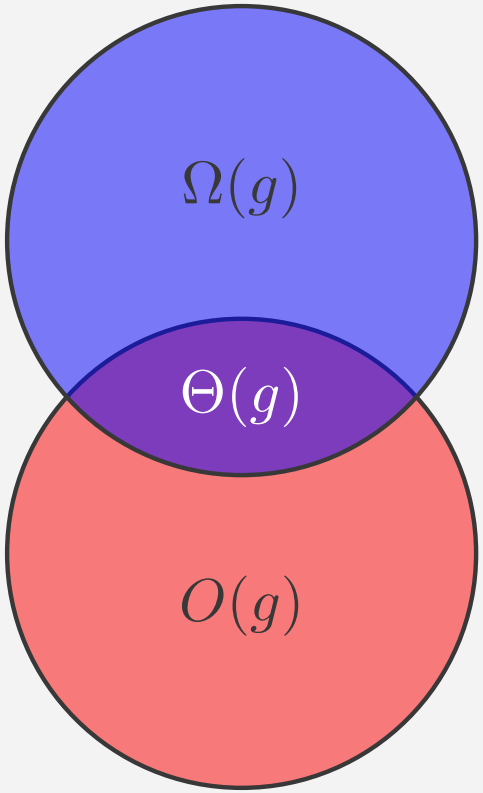
- 根据函数的渐近增长速度 (asymptotic growth rate)、渐近阶 (asymptotic order)或简称阶 (order) 来对函数进行分类，常数项或增长较慢对结果影响不大的项通常在分析中被忽略

- $\Omega(g)$、$\Theta(g)$ 和 $O(g)$
  - $\Omega(g)$：增长速度比 g 快，至少一样快的函数
  - $\Theta(g)$：增长速度与 g 一样快的函数
  - $O(g)$：增长速度比 g 慢，至多一样快的函数

函数分类

- 根据函数的渐近增长速度 (asymptotic growth rate)、渐近阶 (asymptotic order)或简称阶 (order) 来对函数进行分类，常数项或增长较慢对结果影响不大的项通常在分析中被忽略

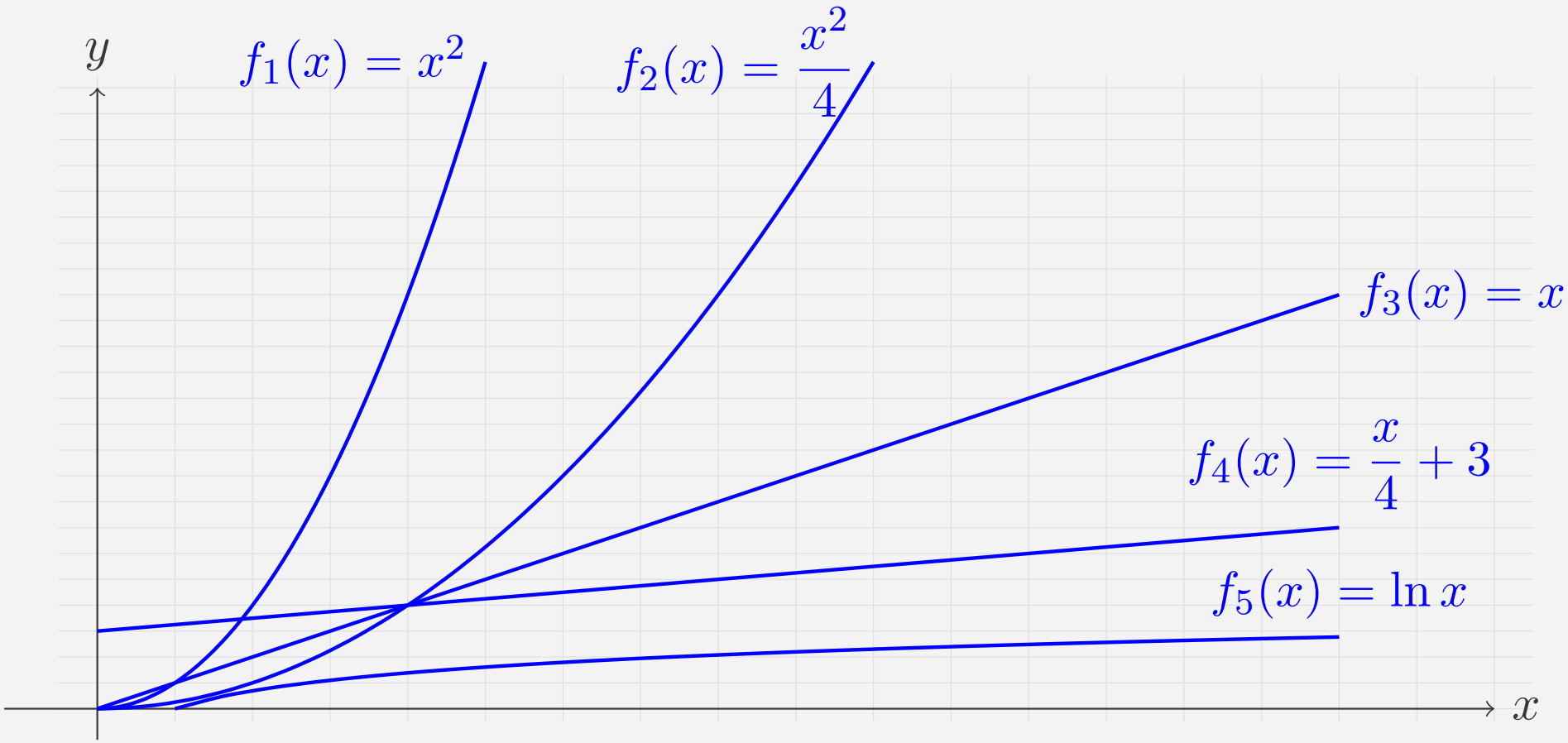
- $\Omega(g)$、$\Theta(g)$ 和 $O(g)$
  - $\Omega(g)$：增长速度比 g 快，至少一样快的函数
  - $\Theta(g)$：增长速度与 g 一样快的函数
  - $O(g)$：增长速度比 g 慢，至多一样快的函数



$\Omega(g)$、$\Theta(g)$ 和 $O(g)$

- 在描述算法复杂度时所使用的函数通常是从非负整数集合到非负实数集合的函数
- 函数 f 和 g 是从非负整数集合到非负实数集合的函数，存在实数 $c > 0$ 和非负整数 n_0：
  - $O(g)$ 是所有满足 $\forall n \geq n_0, f(n) \leq cg(n)$ 的函数 f 构成的集合
  - $\Omega(g)$ 是所有满足 $\forall n \geq n_0, f(n) \geq cg(n)$ 的函数 f 构成的集合
  - $\Theta(g) = O(g) \cap \Omega(g)$

渐近阶比较



渐近阶比较

Example

$f(n) = \frac{n^3}{2}$, $g(n) = 37n^2 + 120n + 17$

1. 当 $n \geq 78$ 时，$g(n) < 1f(n)$，所以 $g \in O(f)$
2. $\lim\limits_{n\to\infty} \frac{g(n)}{f(n)} = \lim\limits_{n\to\infty} (74/n + 240/n^2 + 34/n^3) = 0$
3. 反证法：假设 $f \in O(g)$，

   则 $\exists c > 0, n_0 \in \mathbf{N}$, $\forall n \geq n_0$, $\frac{n^3}{2} \leq 37cn^2 + 120cn + 17c$,

   即 $\frac{n}{2} \leq 37c + \frac{120c}{n} + \frac{17c}{n^2} \leq 174c$，因为 c 是常数，所以当 n 很大时该不等式显然不成立，因此得出矛盾，所以 $f \notin O(g)$

渐近阶比较

Lemma

1. $f \in O(g)$ 如果 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$，包括 $c = 0$
2. $f \in \Omega(g)$ 如果 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$，包括 $c = \infty$
3. $f \in \Theta(g)$ 如果 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$，且 $0 < c < \infty$

渐近阶比较

Lemma

1. $f \in O(g)$ 如果 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c < \infty$，包括 $c = 0$
2. $f \in \Omega(g)$ 如果 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c > 0$，包括 $c = \infty$
3. $f \in \Theta(g)$ 如果 $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$，且 $0 < c < \infty$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0 \Rightarrow f \in o(g)$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty \Rightarrow f \in \omega(g)$$

渐近阶比较

Theorem (洛必达法则 (L'Hôpital's Rule))

函数 f 和 g 可导，且满足 $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0$ 或 $\pm\infty$，则有

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

渐近阶比较

Theorem (洛必达法则 (L'Hôpital's Rule))

函数 f 和 g 可导，且满足 $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = 0$ 或 $\pm\infty$，则有

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

Example

$f(n) = n, g(n) = \lg n$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n}{\lg n} = \lim_{n\to\infty} \frac{n}{\ln n/ln2} = \lim_{n\to\infty} \frac{\ln 2}{1/n} = \lim_{n\to\infty} n \ln 2 = \infty$$

$\therefore f \in \Omega(g)$

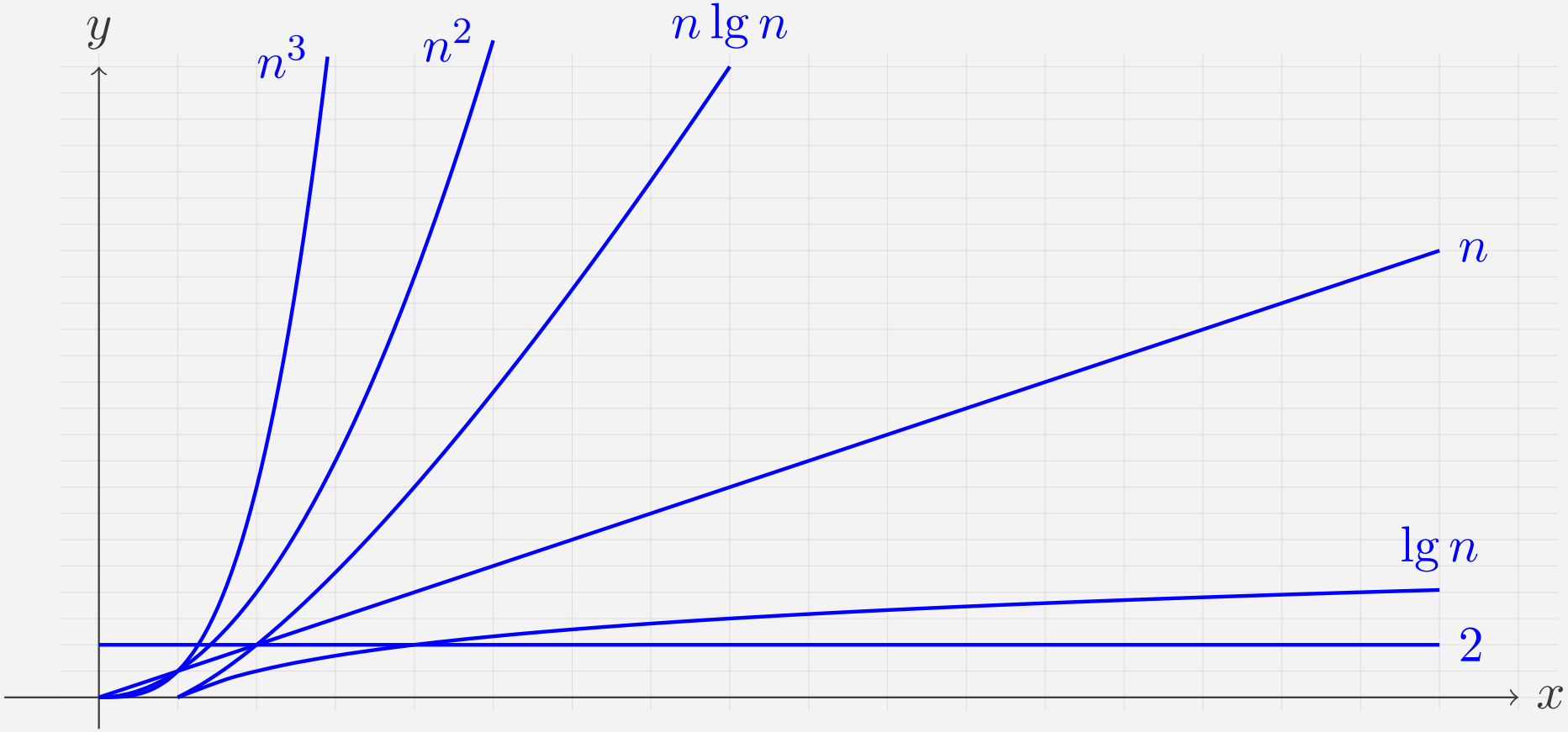
$\Omega(g)$、$\Theta(g)$ 和 $O(g)$ 的性质

- 传递性：$f \in O(g)$ and $g \in O(h) \Longrightarrow f \in O(h)$
- 自反性：$f \in \Theta(f)$
- 对称性：$f \in \Theta(g) \Longrightarrow g \in \Theta(f)$
- Θ 定义了函数空间上的一个等价关系，每一个 $\Theta(f)$ 函数集合是一个等价类
- $f \in O(g) \Longleftrightarrow g \in \Omega(f)$
- $O(f+g) = O(\max(f,g))$，对于 Θ 和 Ω 有类似结论

函数分类

- $O(1)$：渐近阶至多为常数的函数
- $f \in \Theta(n)$：线性阶函数
- $f \in \Theta(n^2)$：平方阶函数
- $f \in \Theta(n^3)$：立方阶函数；
- 对任意常数 $\alpha > 0$ (包括分数)，$\log n \in o(n^\alpha)$
- 对任意常数 $k > 0$ 和 $c > 1$，$n^k \in o(c^n)$
- $\sum_{i=1}^{n} i^d \in \Theta(n^{d+1})$　$\sum_{i=1}^{n} \log i \in \Theta(n \log n)$
- $\sum_{i=a}^{b} r^i \in \Theta(\text{序列最大项})$，$r > 0$ 且 $r \neq 1$，a 和 b 可以是 n 的函数

函数分类



函数渐近阶分析的意义

	解题时间 (假定单位时间为微秒)			
输入规模 (n)	$33n$	$460n \lg n$	$13n^2$	2^n
10	0.00033sec.	0.015sec.	0.0013sec.	0.001sec.
100	0.0033sec.	0.3sec.	0.13sec.	4×10^{16}yr.
1,000	0.033sec.	4.5sec.	13sec.	
10,000	0.33sec.	61sec.	22min.	
100,000	3.3sec.	13min.	1.5days	

主要内容

1 引言

2 背景知识

3 基本数据结构
- 列表
- 栈与队列
- 二叉树
- 抽象数据类型

4 算法分析基础

主要内容

线性表 (Linear List)

- 线性表是由 $n(n \geq 0)$ 个数据元素 (结点) $a_0, a_1, a_2, \cdots, a_{n-1}$ 组成的有限序列
  - 数据元素的个数 n 定义为表的长度 $= list.\text{length}()$ ($list.\text{length}() = 0$ (表里没有一个元素) 时称为空表)
  - 非空的线性表记作: $(a_0, a_1, a_2, \cdots, a_{n-1})$
  - 数据元素 a_i $(0 \leq i \leq n-1)$ 只是个抽象符号，其具体含义在不同情况下可以不同
- 一个数据元素可以由若干个数据项组成。数据元素称为记录，含有大量记录的线性表又称为文件。
- 这种结构具有下列特点:
  - 存在一个唯一的没有前驱的 (头) 数据元素
  - 存在一个唯一的没有后继的 (尾) 数据元素
  - 每一个数据元素均有一个直接前驱和一个直接后继数据元素
- 线性表的常用存储结构: 顺序表 (数组); 链表 (单链表、双链表、循环链表)

广义表

- 广义表一般记作 $LS = (a_0, a_1, \cdots, a_{n-1})$，$n$ 是它的长度，a_i 可以是单个元素 (原子)，也可以是广义表 (子表)
- 当广义表非空时，称第一个元素 a_0 为表头，称其余元素组成的表为表尾。
- 表头是元素 (可以是原子，也可以是广义表)，表尾一定是广义表。
- 广义表是一种非线性的数据结构。但如果广义表的每个元素都是原子，它就变成了线性表

广义表

- 广义表一般记作 $LS = (a_0, a_1, \cdots, a_{n-1})$，$n$ 是它的长度，a_i 可以是单个元素 (原子)，也可以是广义表 (子表)
- 当广义表非空时，称第一个元素 a_0 为表头，称其余元素组成的表为表尾。
- 表头是元素 (可以是原子，也可以是广义表)，表尾一定是广义表。
- 广义表是一种非线性的数据结构。但如果广义表的每个元素都是原子，它就变成了线性表

Example

$E = (a, E)$ 是一个递归的表；
$D = ((), (e), (a, (b, c, d)))$ 是多层次的广义表，长度为 3，深度为 3；
$((a), a)$ 的表头是 (a)，表尾是 (a)，$((a))$ 的表头是 (a)，表尾是 $()$。

主要内容

栈 (Stack)

- 栈 (堆栈)是后进先出 (LIFO, Last In First Out) 的线性表
- 栈只允许在线性表的一端 (称为栈顶，top) 进行添加元素和删除元素的操作，在具体应用中常用数组来实现
- 堆栈的两种基本操作：推入 (push) 和弹出 (pop):
  - 推入 (push)：将数据放入堆栈的顶端，栈顶指针加一。
  - 弹出 (pop)：将顶端数据资料输出 (回传)，栈顶指针减一。

队列 (Queue)

- 队列是先进先出 (FIFO, First In First Out) 的线性表
- 队列只允许在后端 (rear) 进行插入操作，在前端 (front) 进行删除操作，在具体应用中通常用链表或者数组来实现
- 队列的两种基本操作：入队 (enqueue) 和出队 (dequeue)
- 优先队列 (Priority Queue)是不同于先进先出队列的另一种队列。每次从队列中取出的是具有最高优先权的元素

主要内容

二叉树

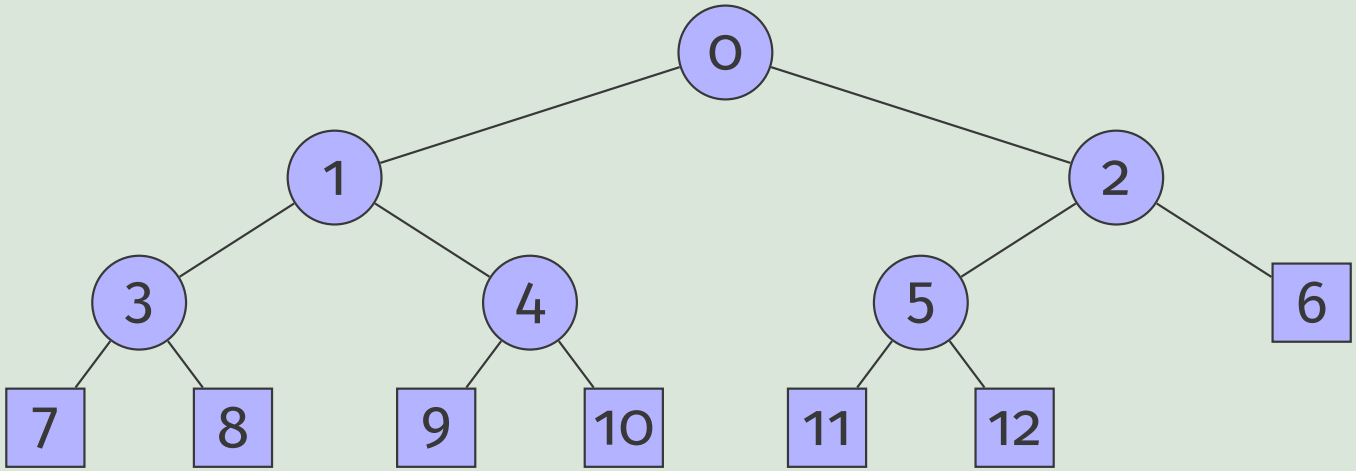
- 二叉树是一组结点的集合，该集合是空集或满足下列条件的非空集合：
  - 有且仅有一个根结点
  - 其余结点分为两个不相交的子集，每个子集都是一棵二叉树，分别称为根结点的左子树 (left subtree) 和右子树 (right subtree)
- 结点的深度 (depth)：根结点的深度为 0，其他结点的深度等于其父结点深度 $+1$
- 二叉树的高度 (height)：叶结点的最大深度
- 结点的度 (degree)：结点的子树个数 (≤ 2)；0 度结点称为叶结点 (leaf)
- 二叉树的性质：
  - 任一二叉树至多包含 2^d 个深度为 d 的结点
  - 高度为 h 的二叉树至多有 $2^{h+1}-1$ 个结点
  - 包含 n 个结点的二叉树其高度至少为 $\lceil \lg(n+1) \rceil - 1$
  - 若非空二叉树有 n_0 个叶结点和 n_2 个度为 2 的结点，则 $n_0 = n_2 + 1$
- 注意：二叉树和树是两个不同的概念，尽管二者有许多相同的性质

完全二叉树

- 满二叉树 (full/proper binary tree)：每个结点都恰有 0 个或 2 个子结点 (除叶结点外每个结点都有两个子结点)
- 完美二叉树 (perfect binary tree)：高度为 h 且有 $2^{h+1}-1$ 个结点的满二叉树
- 完全二叉树 (complete binary tree)：若除最后一层外其余层都是满的，并且最后一层或者是满的，或者是在右边缺少连续若干结点，则此二叉树为完全二叉树
  - 高度为 h 的完全二叉树至少有 2^h 个结点，至多有 $2^{h+1}-1$ 个结点
  - 完全二叉树可以按深度 (层次) 顺序存储在数组中，结点 k 的父结点是 $\lfloor (k-1)/2 \rfloor$，左子结点是 $2k+1$，右子结点是 $2k+2$
  - 有 n 个结点的完全二叉树其叶结点个数为 $\lceil n/2 \rceil$

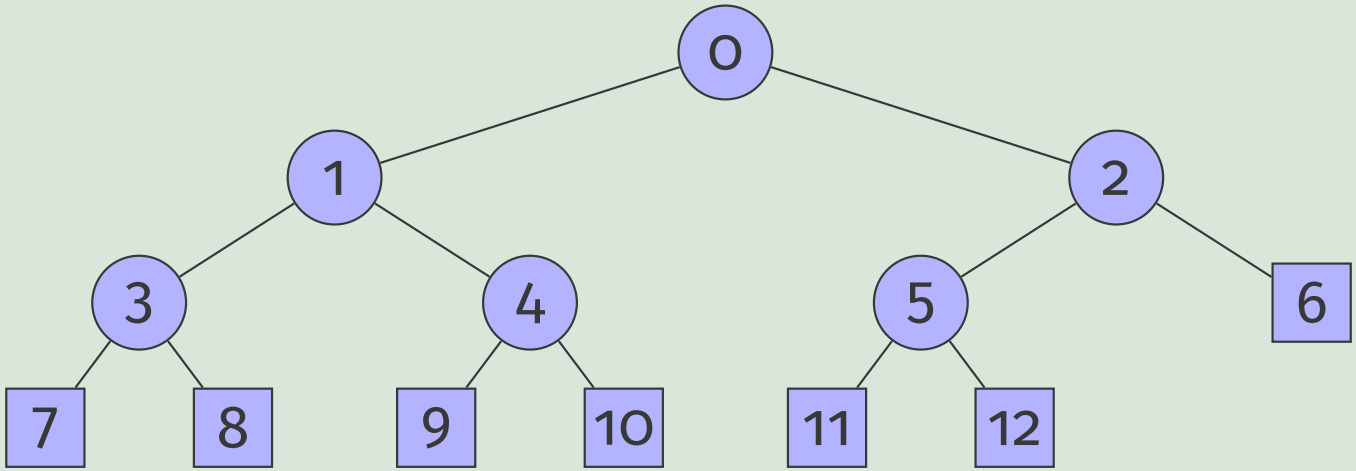
例子

Example (完全二叉树)

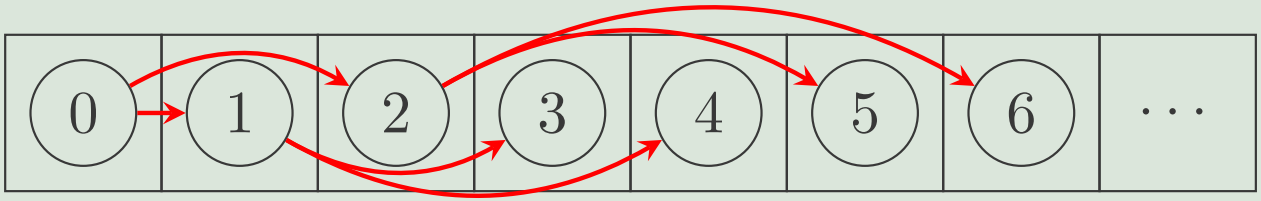


例子

Example (完全二叉树)



完全二叉树的顺序存储方式:



二叉堆 (Binary Heap)

- **堆 (Heap)**：基于树的数据结构，其满足堆特性：若 B 是 A 的子结点，则 $\mathrm{key}(A) \geq \mathrm{key}(B)$ (大根堆)
- **二叉堆**：用完全二叉树来表达的堆结构，可以说是实现优先队列的最有效的数据结构之一
- **二叉堆的基本操作**：
  - 往堆中添加元素 (heapify-up, up-heap, bubble-up)：
    1. 将新元素插入堆的尾部
    2. 将新元素与其父结点比较，若满足堆特性，则结束
    3. 否则，将其与父结点交换位置，并重复上一步骤
  - 提取并删除堆首元素 (heapify-down, down-heap, bubble-down)：
    1. 将堆尾元素放到堆首代替已删除的堆首元素
    2. 将其与子结点比较，若满足堆特性，则结束
    3. 否则，将其与子结点中的一个交换位置，使三者满足堆特性，并重复上一步骤
- ✠ **效率**：入队和出队的复杂度 $W(n) \in O(\log n)$

主要内容

抽象数据类型 (Abstract Data Type)

- 基本数据结构经常采用抽象数据类型来表达
- 抽象数据类型的定义包括：
  - 结构 (Structures)：数据结构的声明
  - 函数 (Functions)：数据操作方法的定义
- 抽象数据类型通常用类 (Class) 来实现 (C++，Java，...)
- 算法的设计通常建立在对抽象数据类型的操作上

主要内容

1 引言

2 背景知识

3 基本数据结构

4 算法分析基础
- 分析的目的
- 算法的正确性
- 算法的复杂度
- 算法的最优性

主要内容

算法分析的目的

- 分析算法的目的：
  - 提高算法性能（更快的速度和更少的额外存储空间占用）；
  - 从几个可能的算法中挑出最好的算法；
  - 设计一个新的算法
- 在分析一个算法时，通常要考察这个算法的
  - 正确性
  - 效率（时间复杂度）和存储空间占用情况（空间复杂度）
  - 最优性和简洁性

主要内容

算法正确性证明

- 一个完整的算法包含一系列步骤 (操作、指令、语句)，将输入 (前件，precondition) 转换成输出 (后件/效果，postcondition)
- 证明的过程：如果前件被满足，则当算法所有步骤按序执行完毕，后件必为真

算法正确性证明

- 一个完整的算法包含一系列步骤 (操作、指令、语句)，将输入 (前件，precondition) 转换成输出 (后件/效果，postcondition)
- 证明的过程：如果前件被满足，则当算法所有步骤按序执行完毕，后件必为真

Example

阶乘的结果总是大于等于 1 的整数

因此一个正确的计算阶乘的算法，对于满足要求的输入，其输出结果必须是大于等于 1 的整数

算法正确性证明

- 一个完整的算法包含一系列步骤 (操作、指令、语句)，将输入 (前件，precondition) 转换成输出 (后件/效果，postcondition)
- 证明的过程：如果前件被满足，则当算法所有步骤按序执行完毕，后件必为真
- 证明工具：反证法、数学归纳法

Example

阶乘的结果总是大于等于 1 的整数

因此一个正确的计算阶乘的算法，对于满足要求的输入，其输出结果必须是大于等于 1 的整数

主要内容

算法工作量的度量 (效率问题)

- 算法的效率：时间复杂度 (time complexity)
  - 需要一种方法来计算算法解题所需的工作量，以便衡量算法的效率高低 (执行时间的长短)
  - 这一度量方法必须是与所使用的计算机、程序设计语言、程序员等一切实现细节无关
  - 工作量的大小通常依赖于输入数据的多少 (工作量是输入规模的函数)
- 度量单位：基本操作 (basic operation)
  - 针对某一特定问题，确定解决该问题所需的最基本的操作是什么
  - 算法为解决问题所执行的所有操作的总和大致正比于执行基本操作的数量
  - 例如：排序算法中最重要且最基本的操作是比较，因此我们一般用比较次数的多少来衡量一个排序算法效率的高低
- 注意输入数据的某些特性是否会影响算法的行为 (边界条件、输入规模...)

最坏情况时间复杂度 (Worst-case Complexity)

- 最坏情况复杂度定义:

$$W(n) = \max\{t(I) | I \in \mathbf{D}_n\}$$

  - $\mathbf{D}_n$ 是算法输入规模为 n 的输入的集合 (通常是无限集)
  - I 是 $\mathbf{D}_n$ 中的某一个输入
  - $t(I)$ 表示在输入 I 下算法执行基本操作的数量
  - 即: 最坏情况复杂度标志了一个算法在各种输入条件下最长 (阶最大) 的执行时间

- 对于特定问题, 最坏情况下的输入 I 依赖于特定算法, 即算法不同, 最坏情况的输入也可能不同

平均时间复杂度 (Average Complexity)

- 平均复杂度定义：

$$A(n) = \sum_{I \in \mathbf{D}_n} P(I)t(I)$$

  - $P(I)$ 是输入 I 出现的概率
  - $P(I)$ 一般难以用解析的方法得到

- 如果考虑算法执行失败的情况：

$$A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$$

- $\mathbf{D}_n$ 中的元素 I 可以是以同样方式影响算法行为的一个集合或等价类

一个例子

Example (在一个无序数组中搜索给定键值)

Algorithm SeqSearch($E[], n, K$)

```
1 for i ← 0 to n − 1 do
2 |   if E[i] = K then
3 |   |   return i;
4 |   end
5 end
6 return −1;
```

一个例子

Example (在一个无序数组中搜索给定键值)

Algorithm SeqSearch($E[], n, K$)

```
1 for i ← 0 to n − 1 do
2 |   if E[i] = K then
3 |   |   return i;
4 |   end
5 end
6 return −1;
```

最坏情况复杂度：$W(n) = n$

一个例子

Example (在一个无序数组中搜索给定键值)

Algorithm SeqSearch($E[], n, K$)

```
1 for i ← 0 to n − 1 do
2 |   if E[i] = K then
3 |   |   return i;
4 |   end
5 end
6 return −1;
```

最坏情况复杂度：$W(n) = n \quad W(n) \in \Theta(n)$

return

平均复杂度分析(例)

- $A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$

平均复杂度分析(例)

- $A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$
- 在 $\mathbf{D}_n$ 中共有 $n+1$ 类输入情况
  - 搜索成功情况(K 在数组中):有 n 类($I_i: K = E[i]$)
  - 搜索失败情况(K 不在数组中):1 类

平均复杂度分析(例)

- $A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$
- 在 $\mathbf{D}_n$ 中共有 $n+1$ 类输入情况
  - 搜索成功情况(K 在数组中):有 n 类(I_i : $K = E[i]$)
  - 搜索失败情况(K 不在数组中):1 类
- 假定 K 等于数组 n 个元素中任一元素的概率相同,即有 $P(I_i|succ) = 1/n$

平均复杂度分析(例)

- $A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$
- 在 $\mathbf{D}_n$ 中共有 $n+1$ 类输入情况
  - 搜索成功情况(K 在数组中):有 n 类(I_i : $K = E[i]$)
  - 搜索失败情况(K 不在数组中):1 类
- 假定 K 等于数组 n 个元素中任一元素的概率相同,即有 $P(I_i|succ) = 1/n$
- $t(I_i) = i + 1,\ i = 0, 1, \cdots, n-1$

平均复杂度分析(例)

- $A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$
- 在 $\mathbf{D}_n$ 中共有 $n+1$ 类输入情况
  - 搜索成功情况(K 在数组中):有 n 类(I_i : $K = E[i]$)
  - 搜索失败情况(K 不在数组中):1 类
- 假定 K 等于数组 n 个元素中任一元素的概率相同,即有 $P(I_i|succ) = 1/n$
- $t(I_i) = i+1,\ i = 0, 1, \cdots, n-1$
- $A_{succ} = \sum_{i=0}^{n-1} P(I_i)t(I_i) = \sum_{i=0}^{n-1}(1/n)(i+1) = (n+1)/2$

平均复杂度分析(例)

- $A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$
- 在 $\mathbf{D}_n$ 中共有 $n+1$ 类输入情况
  - 搜索成功情况(K 在数组中):有 n 类(I_i : $K = E[i]$)
  - 搜索失败情况(K 不在数组中):1 类
- 假定 K 等于数组 n 个元素中任一元素的概率相同,即有 $P(I_i|succ) = 1/n$
- $t(I_i) = i+1,\ i = 0, 1, \cdots, n-1$
- $A_{succ} = \sum_{i=0}^{n-1} P(I_i)t(I_i) = \sum_{i=0}^{n-1}(1/n)(i+1) = (n+1)/2$
- $P(I|fail) = 1,\ t(I) = n,\ \therefore A_{fail}(n) = P(I|fail)t(I) = n$

平均复杂度分析(例)

- $A(n) = P(succ)A_{succ}(n) + P(fail)A_{fail}(n)$
- 在 $\mathbf{D}_n$ 中共有 $n+1$ 类输入情况
  - 搜索成功情况(K 在数组中):有 n 类(I_i : $K = E[i]$)
  - 搜索失败情况(K 不在数组中):1 类
- 假定 K 等于数组 n 个元素中任一元素的概率相同,即有 $P(I_i|succ) = 1/n$
- $t(I_i) = i+1,\ i = 0, 1, \cdots, n-1$
- $A_{succ} = \sum_{i=0}^{n-1} P(I_i)t(I_i) = \sum_{i=0}^{n-1}(1/n)(i+1) = (n+1)/2$
- $P(I|fail) = 1,\ t(I) = n,\ \therefore A_{fail}(n) = P(I|fail)t(I) = n$
- 假定 K 在数组中出现的概率为 q,则有

$$A(n) = q\frac{n+1}{2} + (1-q)n = n(1-\frac{1}{2}q) + \frac{1}{2}q$$

空间复杂度

- 如果可以确定输入数据的存储单元，则可分析最坏情况下或平均情况下算法运行所需占用的存储空间
- 时间与空间的折衷：
  - 存储空间有限 $\Rightarrow$ 算法复杂 $\Rightarrow$ 时间复杂度高
  - 提高算法效率(降低时间复杂度) $\Rightarrow$ 额外的处理或预处理步骤 $\Rightarrow$ 更多的存储空间

Exercise (1)

已知一个长度为 n 的数组和一个正整数 k，并且最多只能使用一个用于交换数组元素的附加空间单元，试设计算法得到原数组循环右移 k 次的结果并分析算法的时间复杂度。

deadline: 2019.11.24

主要内容

算法的最优化

- 问题的复杂度：
  - 问题也存在复杂度的概念，每一类问题都有一个固有的时间复杂度，即解决该类问题所需的最少工作量，无论使用何种算法都不可能少于这个工作量
- 问题复杂度分析：
  - 选择用于解决该问题的一类算法
  - 基于对该类算法复杂度的分析建立并证明解决该问题所需的基本操作数量的下界 (lower bound)
  - 该下界即为该问题的复杂度
- 问题复杂度的分析一般只针对最坏情况

证明算法的最优性

证明某个算法 A 是否是解决这类问题的最优算法

- 分析算法 A 的最坏情况复杂度 $W_A(n)$
- 证明对于任意解决同类问题的算法，在所有规模为 n 的输入中，存在某些输入情况使这些算法的时间复杂度至少为 $W_{[A]}(n)$
- 如果 $W_A(n) = W_{[A]}(n)$，则证明算法 A 是最优的，同时问题复杂度为 $W_{[A]}(n)$，否则：
  - 还存在比 A 更优的算法；
  - 或者存在比 $W_{[A]}(n)$ 更准确的下界

例子

Example (查找一个无序数组中的最大元素)

Algorithm FindMax($E[], n$)

```
1 max ← E[0];
2 for i ← 1 to n − 1 do
3 |   if max < E[i] then
4 |   |   max ← E[i];
5 |   end
6 end
7 return max;
```

复杂度分析

- 基本操作
  - 数组元素之间的比较

复杂度分析

- 基本操作
  - 数组元素之间的比较
- 最坏情况分析
  - 对于元素数为 n 的数组，需要 $n-1$ 次比较操作
  - $W_A(n) = n - 1$

复杂度分析

- 基本操作
  - 数组元素之间的比较
- 最坏情况分析
  - 对于元素数为 n 的数组，需要 $n-1$ 次比较操作
  - $W_A(n) = n - 1$
- 问题的复杂度
  - 假定数组元素各异 (符合最坏情况的前提)
  - 在包含 n 个各不相同元素的数组中，$n-1$ 个元素必然不是最大的
  - 对每一个元素来说，要证明其不是最大元素，必须存在另一个元素比它大，即至少需要一次比较操作
  - 要确定 $n-1$ 个元素不是最大元素，至少需要 $n-1$ 次比较
  - $\therefore W_{[A]}(n) = n - 1$

复杂度分析

- 基本操作
  - 数组元素之间的比较
- 最坏情况分析
  - 对于元素数为 n 的数组，需要 $n-1$ 次比较操作
  - $W_A(n) = n - 1$
- 问题的复杂度
  - 假定数组元素各异 (符合最坏情况的前提)
  - 在包含 n 个各不相同元素的数组中，$n-1$ 个元素必然不是最大的
  - 对每一个元素来说，要证明其不是最大元素，必须存在另一个元素比它大，即至少需要一次比较操作
  - 要确定 $n-1$ 个元素不是最大元素，至少需要 $n-1$ 次比较
  - $\therefore W_{[A]}(n) = n - 1$
- $W_A(n) = W_{[A]}(n) \Longrightarrow$ 上述算法是最优的

查找(搜索)算法

- 回顾：顺序查找算法 ▸ recall
- 对于无序数组，$W_{[\texttt{SeqSearch}]}(n) = n$，即算法 SeqSearch 是最优的

查找(搜索)算法

- 回顾：顺序查找算法 recall
- 对于无序数组，$W_{[\texttt{SeqSearch}]}(n) = n$，即算法 SeqSearch 是最优的
- 对于有序数组，SeqSearch 是否还是最优的呢？

查找(搜索)算法

- 回顾：顺序查找算法 ▸ recall
- 对于无序数组，$W_{[\texttt{SeqSearch}]}(n) = n$，即算法 SeqSearch 是最优的
- 对于有序数组，SeqSearch 是否还是最优的呢？
  - SeqSearch 未利用数组的有序性

查找(搜索)算法

- 回顾：顺序查找算法 ▸ recall
- 对于无序数组，$W_{[\texttt{SeqSearch}]}(n) = n$，即算法 SeqSearch 是最优的
- 对于有序数组，SeqSearch 是否还是最优的呢？
  - SeqSearch 未利用数组的有序性
  - 是否有比 SeqSearch 更优的算法？

查找(搜索)算法

- 回顾：顺序查找算法 ▸ recall
- 对于无序数组，$W_{[\texttt{SeqSearch}]}(n) = n$，即算法 SeqSearch 是最优的
- 对于有序数组，SeqSearch 是否还是最优的呢？
  - SeqSearch 未利用数组的有序性
  - 是否有比 SeqSearch 更优的算法？
- 对有序数组查找算法的一种改进策略：
  - 将 K 与第 ik 处的元素比较，$i = 1, 2, \cdots, n/k$
  - 若结果为小于，则往回比较第 $ik-1, ik-2, \cdots, ik-k+1$ 处的元素
  - 若结果为大于，则往前跳过 $k-1$ 个元素，比较第 $(i+1)k$ 处的元素
  - 则在最坏情况下要比较 $n/k+k$ 次，即 $W(n) = n/k+k$
  - 当 $k = \sqrt{n}$ 时，$W(n)$ 取得最小值 $2\sqrt{n}$
  - 我们将复杂度从 $O(n)$ 降到了 $O(\sqrt{n})$

查找(搜索)算法

- 回顾：顺序查找算法 ▸ recall
- 对于无序数组，$W_{[\texttt{SeqSearch}]}(n) = n$，即算法 SeqSearch 是最优的
- 对于有序数组，SeqSearch 是否还是最优的呢？
  - SeqSearch 未利用数组的有序性
  - 是否有比 SeqSearch 更优的算法？
- 对有序数组查找算法的一种改进策略：
  - 将 K 与第 ik 处的元素比较，$i = 1, 2, \cdots, n/k$
  - 若结果为小于，则往回比较第 $ik-1, ik-2, \cdots, ik-k+1$ 处的元素
  - 若结果为大于，则往前跳过 $k-1$ 个元素，比较第 $(i+1)k$ 处的元素
  - 则在最坏情况下要比较 $n/k+k$ 次，即 $W(n) = n/k+k$
  - 当 $k = \sqrt{n}$ 时，$W(n)$ 取得最小值 $2\sqrt{n}$
  - 我们将复杂度从 $O(n)$ 降到了 $O(\sqrt{n})$
- 还有没有更优的算法？

二分查找 (Binary Search) 算法

假定数组元素按不减 (nondecreasing) 序排列

Algorithm `BinarySearch`$(E[], n, K)$

```
1 first ← 0;
2 last ← n − 1;
3 while first ≤ last do
4 |   mid ← (first + last)/2;
5 |   if K = E[mid] then return mid;
6 |   else if K < E[mid] then last ← mid − 1;
7 |   else first ← mid + 1;
8 end
9 return −1;
```

BinarySearch 最坏情况复杂度分析

- 基本操作：键值 K 与数组元素的比较
- 算法在每一次比较处产生三个分支
- 在最坏情况下每次比较都不相等，而要查找的部分数组则被一分为二，根据比较结果选择前一半或后一半作为下面要继续查找的部分
- 在待查找部分仅剩一个元素之前需要经过多少次对半分？
  - $n/(2^d) \geq 1 \Rightarrow d \leq \lg n$
- $W(n) = \lfloor \lg n \rfloor + 1 = \lceil \lg(n+1) \rceil \in \Theta(\lg n)$

BinarySearch 平均复杂度分析

- 按 K 在数组中出现的位置将所有输入分为 $n+1$ 种情况，其中有 1 种为查找失败情况
- 不妨设 $n = 2^d - 1$，则 $A_{fail}(n) = \lg(n+1)$
- 设在查找成功情况下 K 在数组各元素位置出现的概率为 $P(I_i|succ) = 1/n$
  - 将数组的 n 个元素位置分组：$\mathbf{S}_t$ 表示恰比较 t 次可知与 K 相等的元素位置集合
  - 则显然有：$|\mathbf{S}_1| = 1 = 2^0$, $|\mathbf{S}_2| = 2^1$, $|\mathbf{S}_3| = 2^2$, $\cdots$, $|\mathbf{S}_t| = 2^{t-1}$
  - 所以，$A_{succ}(n) = \sum_{t=1}^{d}(|\mathbf{S}_t|/n)t = ((d-1)2^d + 1)/n = \lg(n+1) - 1 + \lg(n+1)/n$
- $A(n) = qA_{succ}(n) + (1-q)A_{fail}(n) = \lg(n+1) - q + \dfrac{q\lg(n+1)}{n}$

最优性

- **查找算法类**：在大小为 n 的数组 E 中查找与给定键值 K 相等的元素，返回其位置；对数组元素除比较外无其他操作

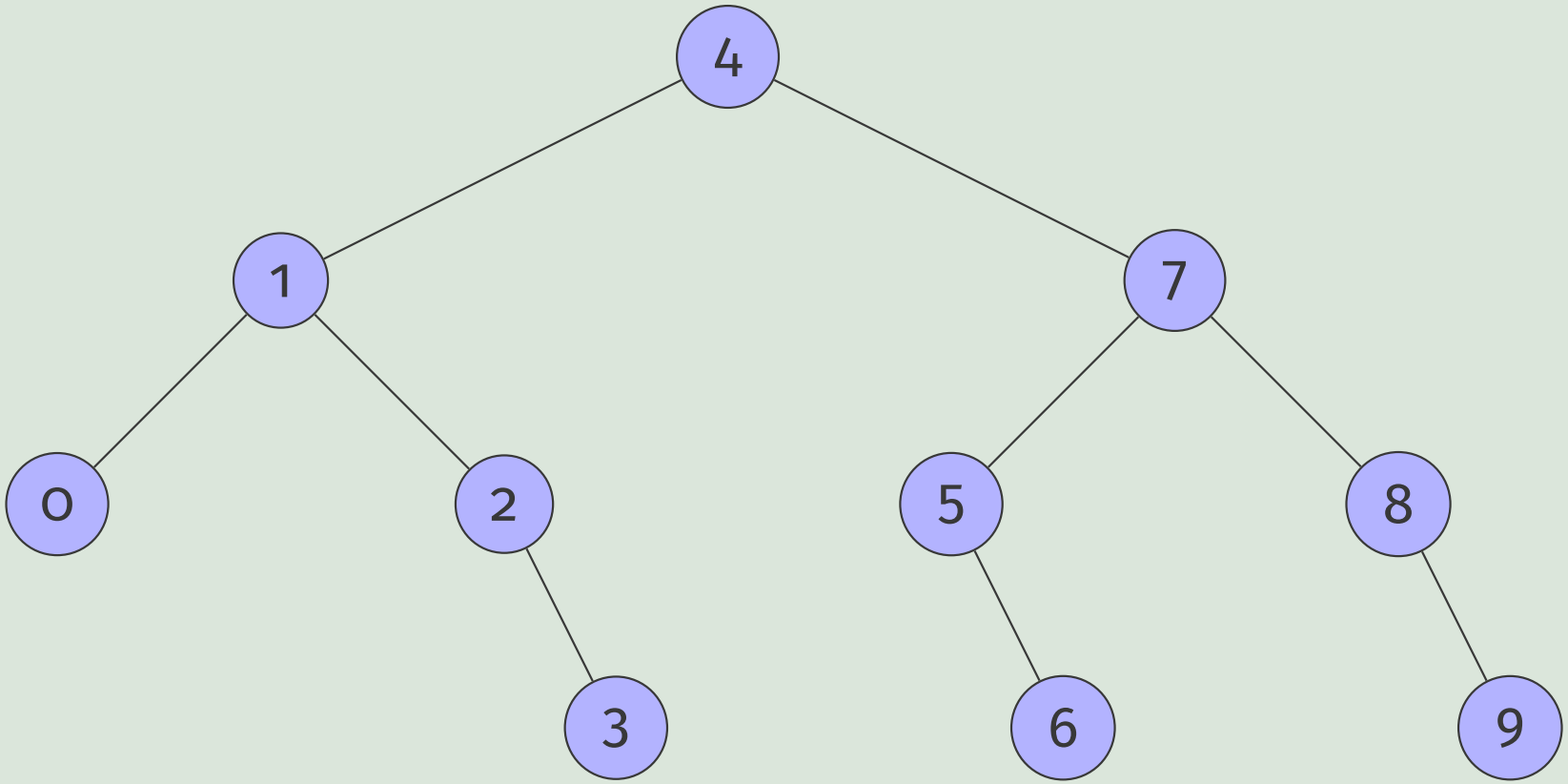
- **决策树 (decision tree)**：对查找算法类中的某个输入规模为 n 的算法，其决策树是包含标有 $0 \sim n-1$ 结点的二叉树，其构造规则如下
  1. 树根标记第一个与键值 K 比较的元素位置
  2. 对标记为 i 的结点，其左子结点标记为当 $K < E[i]$ 时下一个与 K 比较的元素位置，其右子结点标记为当 $K > E[i]$ 时下一个与 K 比较的元素位置
  3. 一个结点若无左子结点或右子结点，则表示在得到 $K < E[i]$ 或 $K > E[i]$ 的结果后算法终止

最优性 (cont.)

- 决策树中从根开始到叶子结点的每条路径代表了其对应算法在给定输入下的比较操作序列
- 对于给定算法，最坏情况下的比较操作序列对应于其决策树中从根到叶子结点的最长路径
- 显然，任何合理的查找算法都可以根据以上规则构造决策树；而其比较次数不会多于其决策树的层数

决策树实例

Example ($n = 10$ 时 BinarySearch 的决策树)



查找算法类的复杂度

- 查找算法最坏情况下的比较操作序列对应于其决策树中从根到叶子结点的最长路径，设该路径经过的结点数 (即最坏情况下比较次数) 为 p，而决策树共包含 N 个结点，则有：
  - $N \leq 1 + 2 + 4 + \cdots + 2^{p-1} = 2^p - 1 \Longrightarrow 2^p \geq N + 1$

查找算法类的复杂度

- 查找算法最坏情况下的比较操作序列对应于其决策树中从根到叶子结点的最长路径，设该路径经过的结点数 (即最坏情况下比较次数) 为 p，而决策树共包含 N 个结点，则有：
  - $N \leq 1 + 2 + 4 + \cdots + 2^{p-1} = 2^p - 1 \Longrightarrow 2^p \geq N + 1$
- 如何得到 p 与输入规模 n 之间的关系？

查找算法类的复杂度

- 查找算法最坏情况下的比较操作序列对应于其决策树中从根到叶子结点的最长路径，设该路径经过的结点数 (即最坏情况下比较次数) 为 p，而决策树共包含 N 个结点，则有：
  - $N \leq 1+2+4+\cdots+2^{p-1}=2^p-1 \Longrightarrow 2^p \geq N+1$
- 如何得到 p 与输入规模 n 之间的关系？$N \overset{?}{\geq} n$

查找算法类的复杂度

- 查找算法最坏情况下的比较操作序列对应于其决策树中从根到叶子结点的最长路径，设该路径经过的结点数 (即最坏情况下比较次数) 为 p，而决策树共包含 N 个结点，则有：
  - $N \leq 1+2+4+\cdots+2^{p-1} = 2^p - 1 \Longrightarrow 2^p \geq N+1$
- 如何得到 p 与输入规模 n 之间的关系？$N \overset{?}{\geq} n$
- 即需要证明：对从 0 到 $n-1$ 的每一个 i，决策树中至少有一个结点标记为 i

查找算法类的复杂度

- 查找算法最坏情况下的比较操作序列对应于其决策树中从根到叶子结点的最长路径，设该路径经过的结点数 (即最坏情况下比较次数) 为 p，而决策树共包含 N 个结点，则有：
  - $N \leq 1+2+4+\cdots+2^{p-1} = 2^p - 1 \Longrightarrow 2^p \geq N+1$
- 如何得到 p 与输入规模 n 之间的关系？$N \overset{?}{\geq} n$
- 即需要证明：对从 0 到 $n-1$ 的每一个 i，决策树中至少有一个结点标记为 i
- $2^p \geq N+1 \geq n+1 \Longrightarrow p \geq \lg(n+1)$

查找算法类的复杂度

- 查找算法最坏情况下的比较操作序列对应于其决策树中从根到叶子结点的最长路径，设该路径经过的结点数 (即最坏情况下比较次数) 为 p，而决策树共包含 N 个结点，则有：
  - $N \leq 1+2+4+\cdots+2^{p-1} = 2^p - 1 \Longrightarrow 2^p \geq N+1$
- 如何得到 p 与输入规模 n 之间的关系？$N \overset{?}{\geq} n$
- 即需要证明：对从 0 到 $n-1$ 的每一个 i，决策树中至少有一个结点标记为 i
- $2^p \geq N+1 \geq n+1 \Longrightarrow p \geq \lg(n+1)$

Theorem

任何使用比较的方法在包含 n 个元素的数组中查找键值为 K 的元素的算法至少要做 $\lceil \lg(n+1) \rceil$ 次比较

证明 $N \geq n$

反证法证明.

- 假设从 0 到 $n-1$ 中存在某一位置 i 在决策树中没有对应的结点标注
- 构造两个输入数组 E_1 和 E_2 如下：
  - $E_1[i] = K,\ E_2[i] = K' > K$
  - $\forall j < i,\ E_1[j] = E_2[j] < K$
  - $\forall j > i,\ E_1[j] = E_2[j] > K'$
- 因为决策树中不存在 i 结点，算法将永远不会比较 K 与 $E_1[i]$ 或 $E_2[i]$，这样，对于 E_1 和 E_2 这两种不同的输入，同一算法处理的元素完全相同，因而其运行过程和结果都是一样的
- 所以，E_1 和 E_2 中至少有一个输入，算法给出的结果是错误的，与算法的正确性构成矛盾

□

进一步思考

- 若只需要判定某一给定键值是否在数组中，而不需要给出其在数组中出现的具体位置，则是否存在更快的算法？
  - 数组无序；
  - 数组有序；
  - 数组规模非常大；
  - 使用更复杂的数据结构；
  - 空间复杂度。
- 如果允许出现少量错判，则情况又如何？

本讲回顾

1. 引言
   - 算法简介
   - 关于本课程
2. 背景知识
   - 集合论
   - 逻辑
   - 概率
   - 代数
3. 基本数据结构
   - 列表
   - 栈与队列
   - 二叉树
   - 抽象数据类型
4. 算法分析基础
   - 分析的目的
   - 算法的正确性
   - 算法的复杂度
   - 算法的最优性