



# 数据结构与算法 (Python) -01/概论

刘云淮 Yunhuai.liu@pku.edu.cn

<http://www.yunhuai.net/DSA2023/CoursePage/DSA2023.html>

北京大学计算机学院

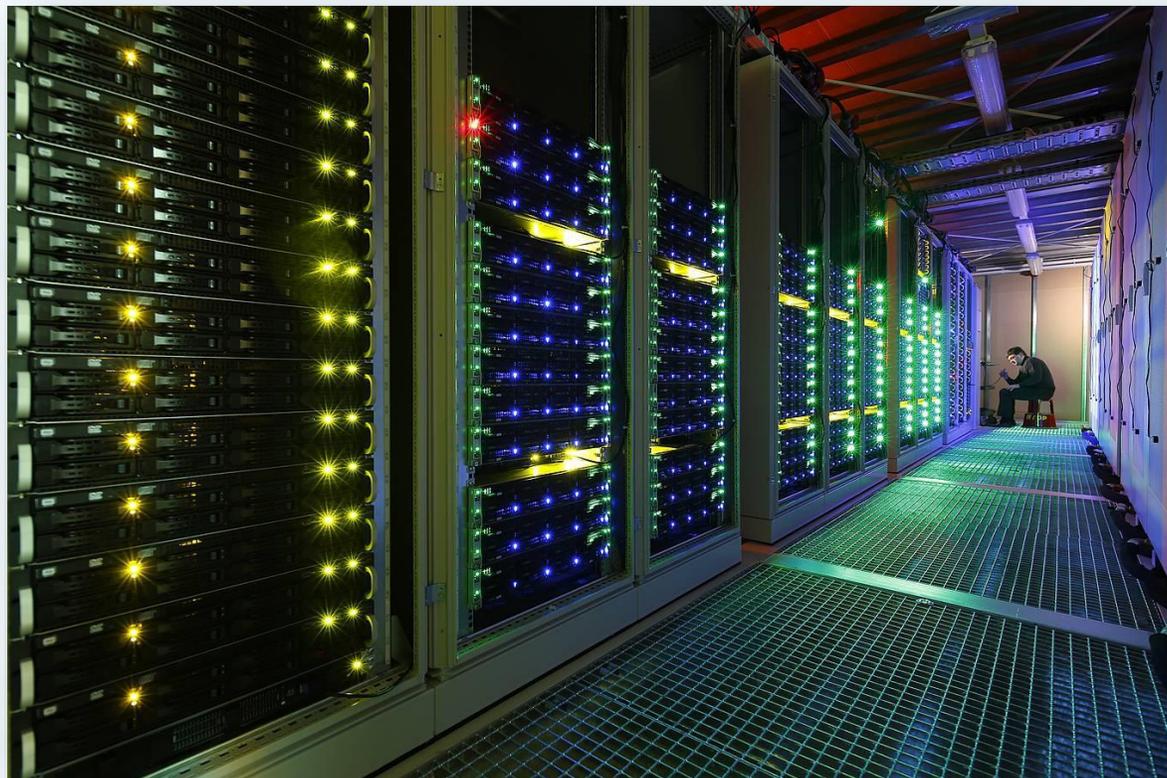
# 目录

- › **关于计算**  
计算的定义，可计算性，计算复杂度
- › **什么是计算机科学**
- › **什么是数据结构**
- › **什么是算法**
- › **编程与算法的区别**
- › **为什么学习数据结构与抽象数据类型**
- › **为什么学习算法**



# 关于计算

- › 问题，以及如何解决问题
- › 图灵机
- › 可以通过“计算”解决的问题
- › 计算复杂性
- › 不可计算问题
- › 突破极限



# 问题，以及如何解决问题1/4

- 人们在生活、生产、学习、探索、创造过程中会遇到各种未知的事物  
云是什么？这种草（虫子）可以吃么？什么是无理数？什么是万物的起源？  
为什么会下雨？为什么食物放久了会发霉？为什么 $\sqrt{2}$ 是无理数？生命的意义是什么？  
怎么让粮食长得更多？怎么将楼房建到101层？怎么求最大公约数？怎么维护公平与正义？

## 问题解决之道：从未知到已知

感觉、经验（实验范式）

占卜、求神

逻辑、数学、实验（理论范式）

工程、计算

模型、模拟、仿真（仿真范式）

哲学？



你脑袋里是不是充满了问号呢？

# 问题，以及如何解决问题2/4

› **有些问题已经解决，很多问题尚未解决，有些问题似乎无法解决**  
尚未解决和无法解决问题的共性：表述含混、标准不一、涉及主观、结果不确定

› **数学：解决问题的终极工具**

在长期的发展过程中，人们把已经解决的问题逐渐表述为数学命题与模型；

尚未解决的问题，人们试图通过数学建模，采用数学工具来解决；

无法解决的问题，人们试图转换表述、明晰问题来部分解决。

› **为什么是数学？**

数学具有清晰明确的符号表述体系；

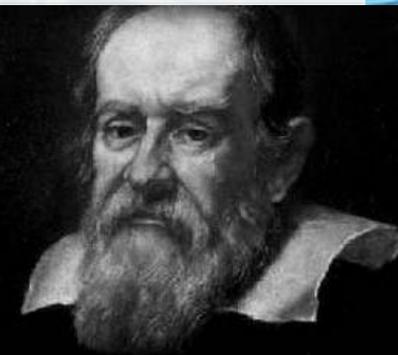
严密确定的推理系统；

但正如科学不是万能的，数学也不是万能的

- 有些问题天然无法明确表述（主观、价值观、意识形态、哲学问题等）
- 有些可明确表述的问题仍然无法解决（留后待述）

**Mathematics is the alphabet in  
which God has written the universe**

Galileo, Italian astronomer, mathematician and philosopher (1564 - 1642)



# 问题，以及如何解决问题3/4

## › 问题的分类

- What: 是什么? 面向判断与分类的问题;
- Why: 为什么? 面向求因与证明的问题;
- How: 怎么做? 面向过程与构建的问题。

## › 问题的核心

- 给定一个特定的输入
- 通过某些中间过程
- 得到一些特定的输出

# 希尔伯特和他著名的判定问题

## › 数学是完备的么?

是否所有的正确的数学陈述，都可以被证明?

## › 数学是一致的么?

是否存在又对又不对的数学陈述?

## › 数学是可判定的么?

是否所有的数学陈述，都是可以被简单的逻辑判定对错?



	大卫·希尔伯特 (1886年)
出生	1862年1月23日 普鲁士柯尼斯堡 (今俄罗斯加里宁格勒)
逝世	1943年2月14日 (81岁) 纳粹德国哥廷根
居住地	德国
国籍	德国
研究领域	数学和哲学
机构	柯尼斯堡大学 哥廷根大学

# 问题，以及如何解决问题

## › 何谓简单的方法判断对错-

“能否找到一种基于有穷、确定、可行观点的方法，来判定任何一个数学命题的真假”——抽象的“计算”概念提出

- 由有限数量的明确有限指令构成；
- 指令执行在有限步骤后终止；
- 指令每次执行都总能得到正确解；
- 原则上可以由人单独采用纸笔完成，而不依靠其它辅助；
- 每条指令可以机械地被精确执行，而不需要**智慧**和**灵感**。

# 问题，以及如何解决问题4/4

## › 问题解决的“计算”之道

20世纪30年代，几位逻辑学家几乎同时各自独立提出了几个关于“计算”的数学模型

- 奥地利逻辑学家、数学家哥德尔(K.F. Godel,1906-1978)和美国逻辑学家、数学家克莱尼(S.C. Kleene,1909-1994)的递归函数模型
- 美国逻辑学家、数学家丘奇(A. Church,1903-1995)的Lambda演算模型
- 波兰裔美国逻辑学家、数学家波斯特(E.L. Post,1897-1954)的Post机模型
- 英国逻辑学家、数学家图灵(A.M. Turing,1912-1954)的图灵机模型

后续研究证明，这几个“基于有穷观点的能行方法”的计算模型，全都是等价的，在某个模型下“可计算”的问题，在另外的模型下也是“可计算”的

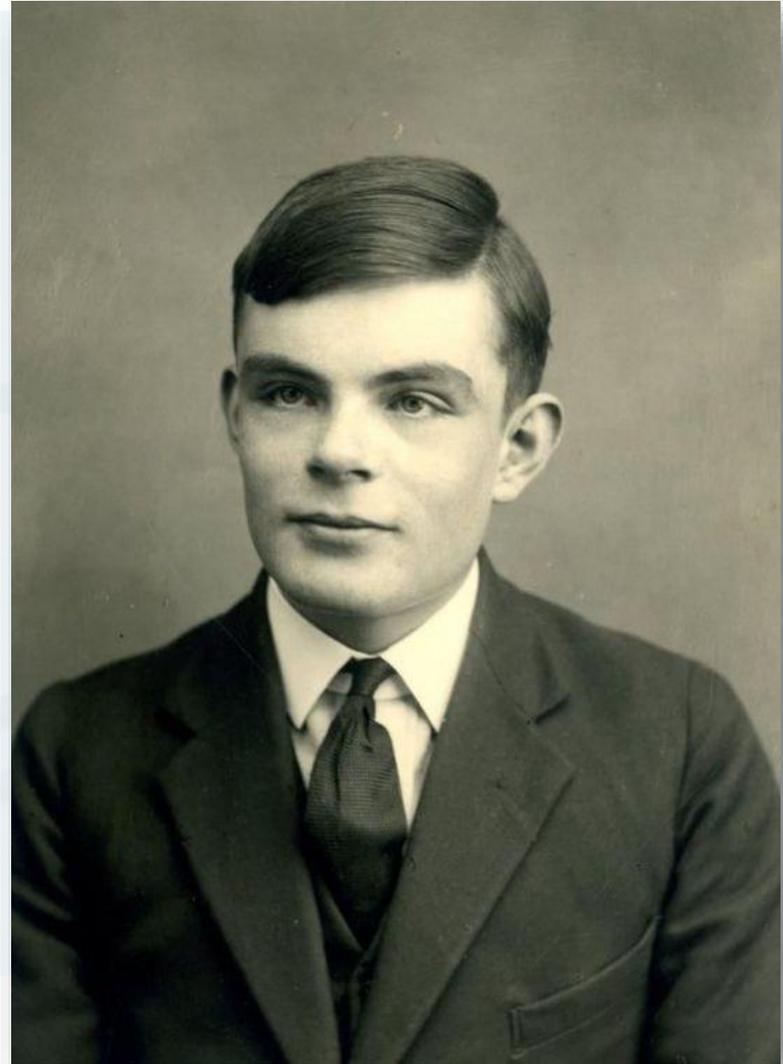
虽然希尔伯特的计划最终被证明无法实现，即不存在“能行方法”来判定任何一个数学命题的真假，总有数学命题，其真假是无法证明的

但“能行可计算”的概念，成为了计算理论的基础，其中的一些数学模型（如图灵机）也成为现代计算机的理论基础

计算机是数学家一次失败思考的产物。  
——无名氏

# 图灵机 Turing Machine

- › **1936年, Alan Turing提出的一种抽象计算模型**  
基本思想是用机器模拟人们用纸笔进行数学运算的过程, 但比数值计算更为简单
- › **在纸上写上或擦除某个符号;**
- › **把注意力从纸的一个位置转向另一个位置;**
- › **在每个阶段, 人要决定下一步的动作, 依赖于:**
  - (a) 此人当前所关注的纸上某个位置的符号和
  - (b) 此人当前思维的状态。



# 艾伦·图灵和他的图灵机

## 论可计算数及其在判定问题中的应用

ON COMPUTABLE NUMBERS, WITH AN APPLICATION TO  
THE ENTSCHIEDUNGSPROBLEM

By A. M. TURING.

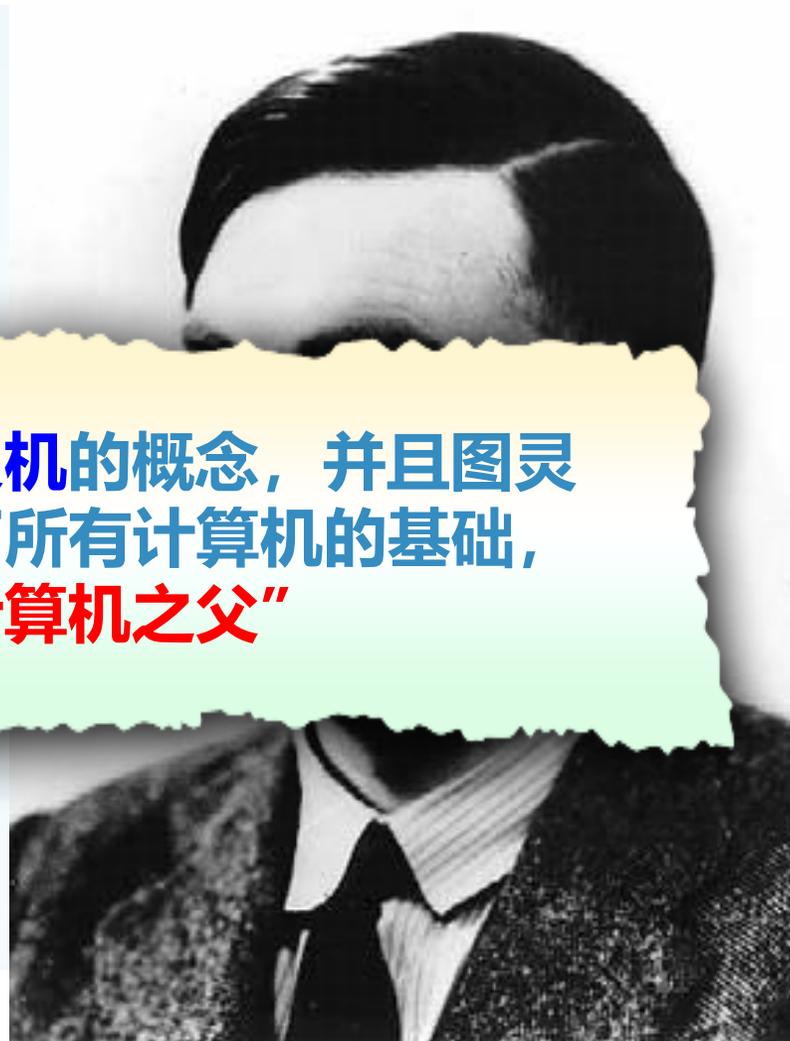
[Received 1936, published November, 1936.]

The “computable numbers whose expressions are finite. Although the subject is almost equally elementary as that of an integral variable, it is almost equally elementary as that of an integral variable, and so, however, the same in effect, for explicit treatment, shortly to give an account of functions, and so forth of the theory of functions.

computable numbers. According to my definition, a number is computable if its decimal can be written down by a machine.

In §§ 9, 10 I give some arguments with the intention of showing that the computable numbers include all numbers which could naturally be regarded as computable. In particular, I show that certain large classes of numbers are computable. They include, for instance, the real parts of all algebraic numbers, the real parts of the zeros of the Bessel functions, the numbers  $\pi$ ,  $e$ , etc. The computable numbers do not, however, include all definable numbers, and an example is given of a definable number which is not computable.

这篇论文，首次提出了图灵机的概念，并且图灵机是工程可实现的，奠定了所有计算机的基础，图灵也被称为“计算机之父”



# Can Machines Think?

A. M. Turing (1950) *Computing Machinery and Intelligence*. *Mind* 49: 433-460.

## COMPUTING MACHINERY AND INTELLIGENCE

### 1. The Imitation Game

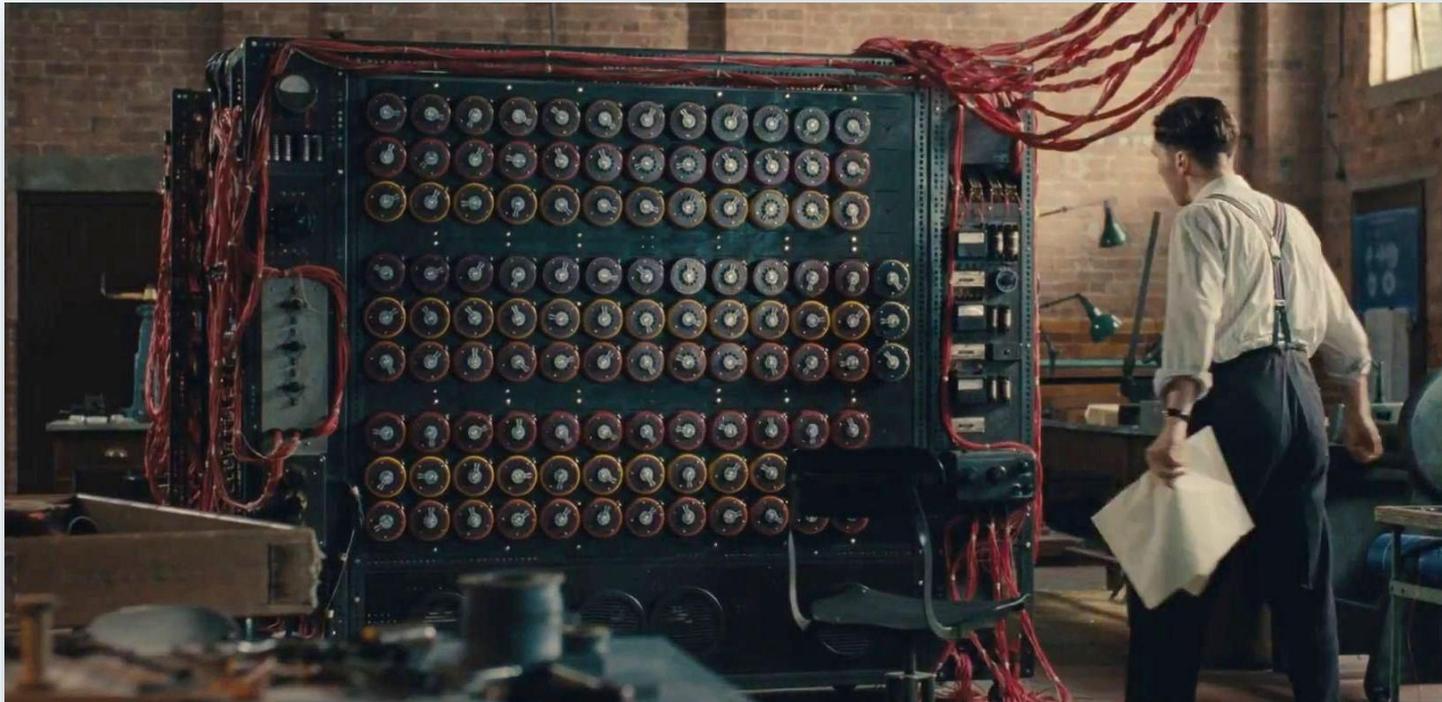
I propose to consider the question "Can machines think?" This should be framed so as to reflect so far as possible the natural language usage of the word "think". It is dangerous, if the meaning of the word "think" is not clear, to examine how they are computed. The answer to the question is to be given in the form of a statistical survey such as a Galton question. In the present definition I shall replace the question by a definition expressed in relatively unambiguous words.

The new form of the problem can be described in terms of a game which we call the "imitation game." It is played with three people, a man (A), a woman (B), and an interrogator (C) who may be of either sex. The interrogator stays in a room apart from the other two. The object of the game for the interrogator is to determine which of the other two is the man and which is the woman. He knows them by labels X and Y, and at the end of the game he says either "X is A and Y is B" or "X is B and Y is A." The interrogator is allowed to put questions to A and B thus:

这篇论文，首次提出了图灵测试的概念，奠定了所有人工智能的基础，图灵也被称为“人工智能之父”



# The.Imitation.Game.2014



# 图灵及图灵奖



艾伦·麦席森·图灵



ACM图灵奖杯

ACM图灵奖 (A.M. Turing Award) , 又称“A.M.图灵奖”, 由国际计算机协会 (ACM) 于1966年设立, 专门奖励对计算机事业作出重要贡献的个人。其名称取自计算机科学先驱、英国科学家艾伦·麦席森·图灵 (A.M. Turing) 。图灵奖对获奖条件要求极高, 评奖程序极严, 它是计算机界最负盛名、最崇高的一个奖项, 有“**计算机诺贝尔奖**”之称。

# 图灵机 Turing Machine

## 图灵机由以下几部分构成

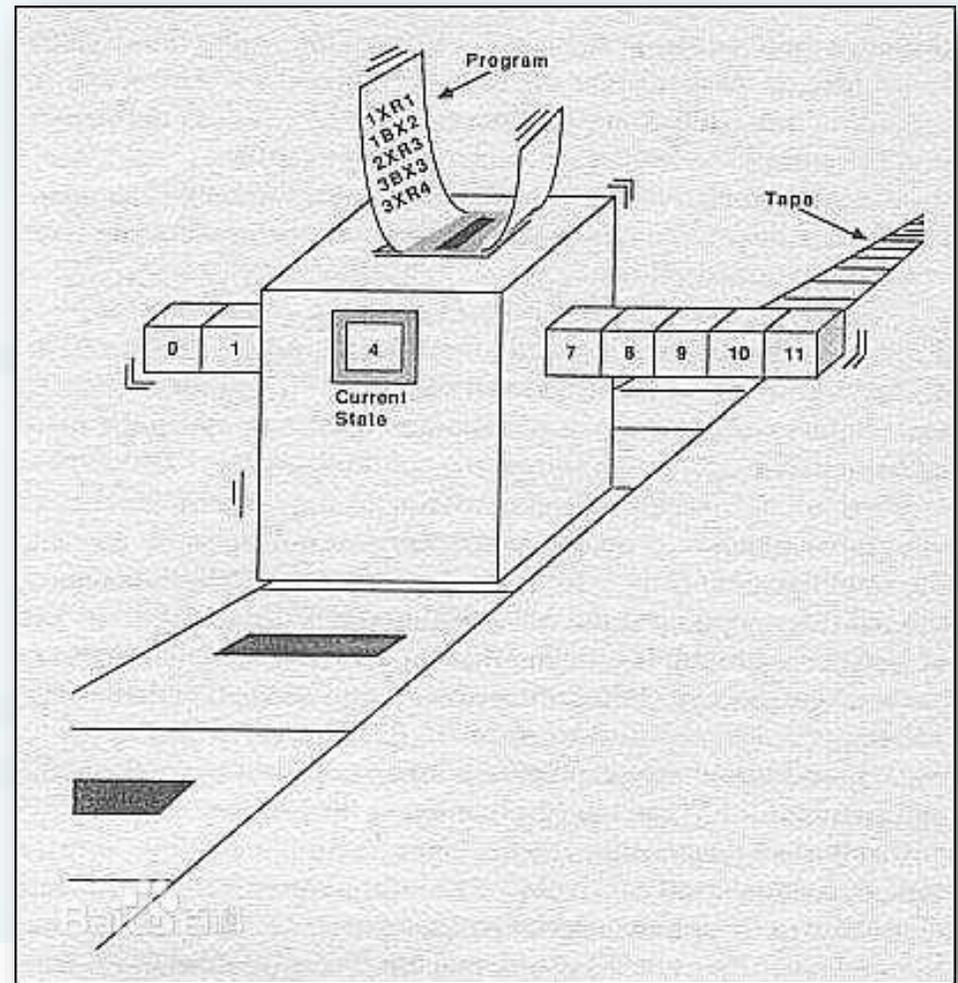
一条无限长的纸带，分为一个个相邻的格子，每个格子可以记录一个符号

一个读写头，可以在纸带上左右移动，能读出和擦写格子的字符

一个状态寄存器，记录机器所在的状态，状态的数量是有限的

一系列有限的控制规则

- 每条规则指明了在当前状态下，根据读写头读入的字符
- 来确定读写头擦写格子的字符，是否移动，是否改变状态



# 图灵机 Turing Machine: 例子

- › 判定  $\{a^m b^m \mid m \geq 0\}$ : 左半部全是a, 右半部全是b, 且ab数量相等的字符串  
如: ab、aabb、aaaabbbb, 进入“接受”状态, 如: b、ba、abb, 进入“拒绝”状态
- › 规则思路: 将a和b一一对消, 如果最后剩下空白B则接受, 否则拒绝
  - <s0, a, B, s1, R>: 初始碰到a, 消去, s1, 右移
  - <s1, a, a, s1, R>: 消去1个a的状态, 继续右移, 找最后一个b
  - <s1, b, b, s1, R>: 继续右移
  - <s1, B, B, s2, L>: 右移到头状态s2, 回移
  - <s2, b, B, s3, L>: 如果有b, 消去, 进入左移状态s3
  - <s3, b, b, s3, L>: 左移
  - <s3, a, a, s3, L>: 左移

# 图灵机 Turing Machine: 例子

## › 规则

$\langle s_3, B, B, s_0, R \rangle$ : 左移到头变初始状态  $s_0$ , 右移看下个字符

$\langle s_0, B, B, s_Y, N \rangle$ : a, b 都能一一消完, 则进入“接受”状态, 停机

$\langle s_0, b, b, s_N, R \rangle$ : b 多了, 或者在 a 前, 进入“拒绝”状态, 停机

$\langle s_2, a, a, s_N, R \rangle$ : a 多了, 或者在 b 后, 进入“拒绝”状态, 停机

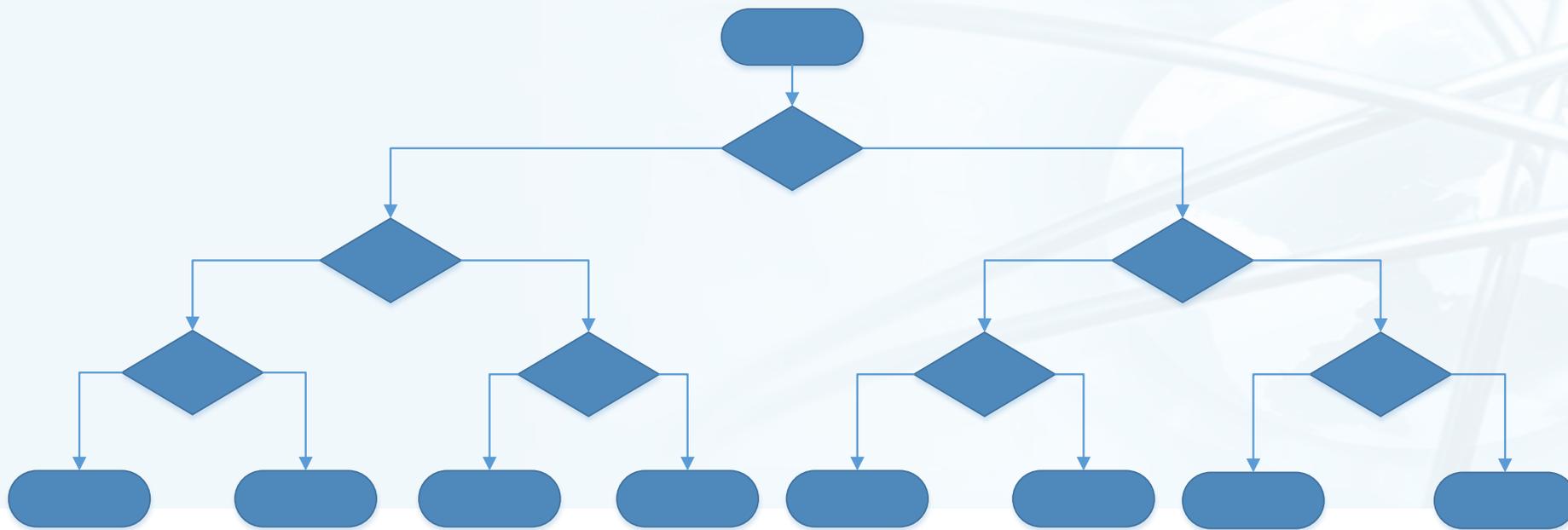
$\langle s_2, B, B, s_N, R \rangle$ : a 多了, 进入“拒绝”状态, 停机

## › 网上有很多虚拟图灵机, 大家可以试试

## › Virtual Turing Machine

# 可以通过“计算”解决的问题1/3

- › 如果用任何一个“有限能行方法”下的计算模型可以解决的问题，都算是“可计算”的
- › What: 分类问题，可以通过树状的判定分支解决

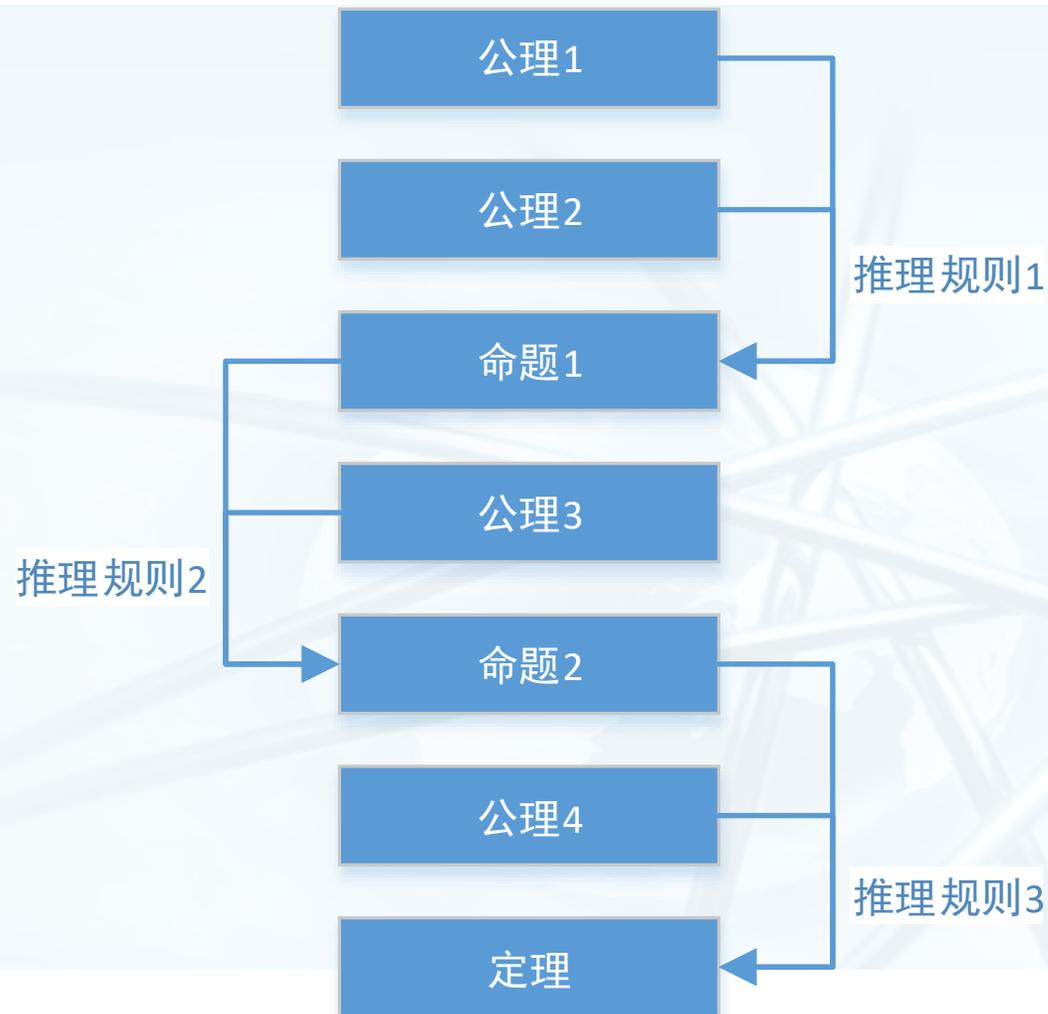


# 可以通过“计算”解决的问题2/3

## Why: 证明问题, 可以通过有限的公式序列来解决

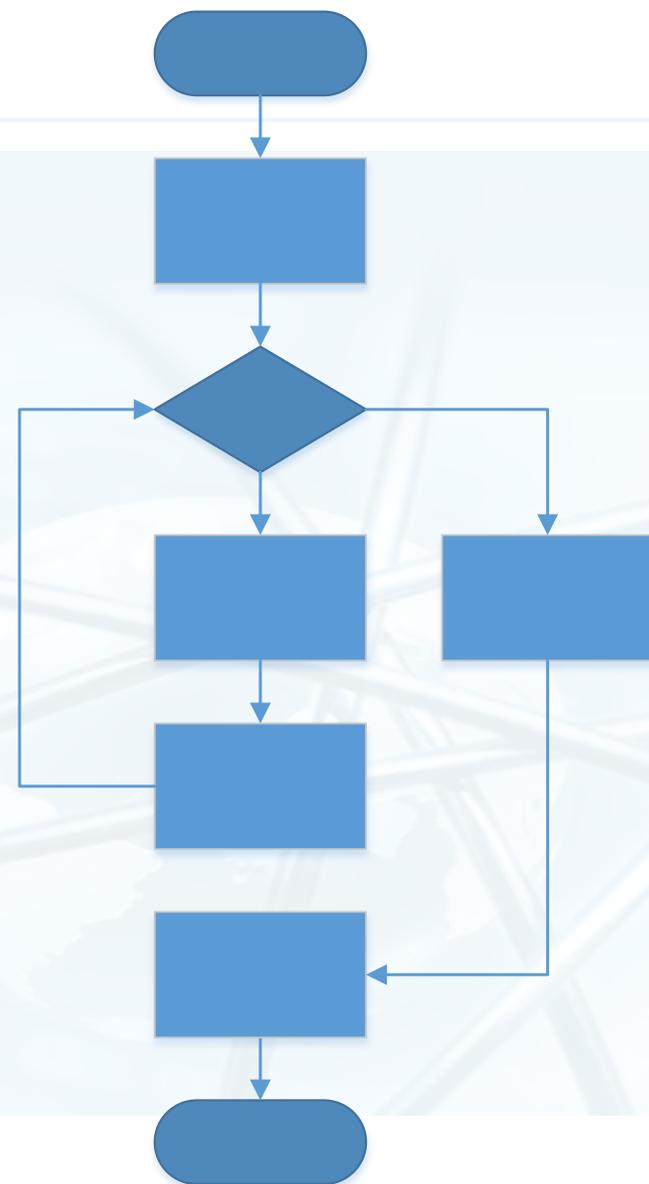
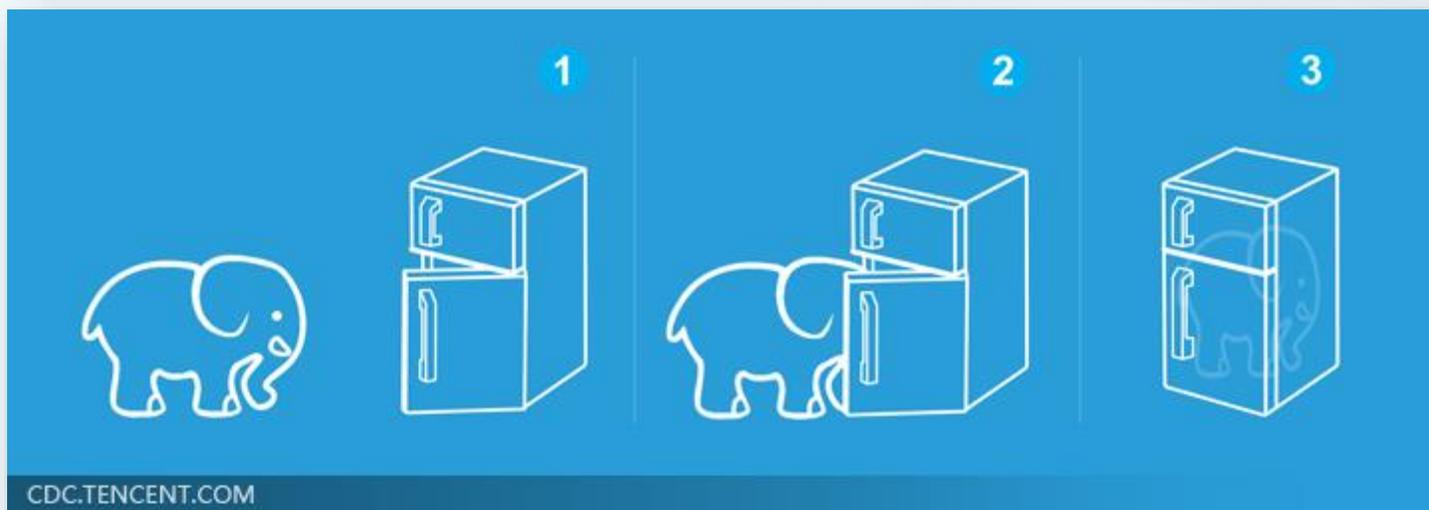
数学定理证明采用符号语言, 从不证自明的公理出发, 一步步推理得出最后待证明的定理

我们在以往学习过的定理证明即为此类解决方法



# 可以通过“计算”解决的问题

› How: 过程问题, 可以通过流程来解决



# 世界上最早的算法：欧几里德算法（最大公约数）

› 公元前3世纪，记载于《几何原本》

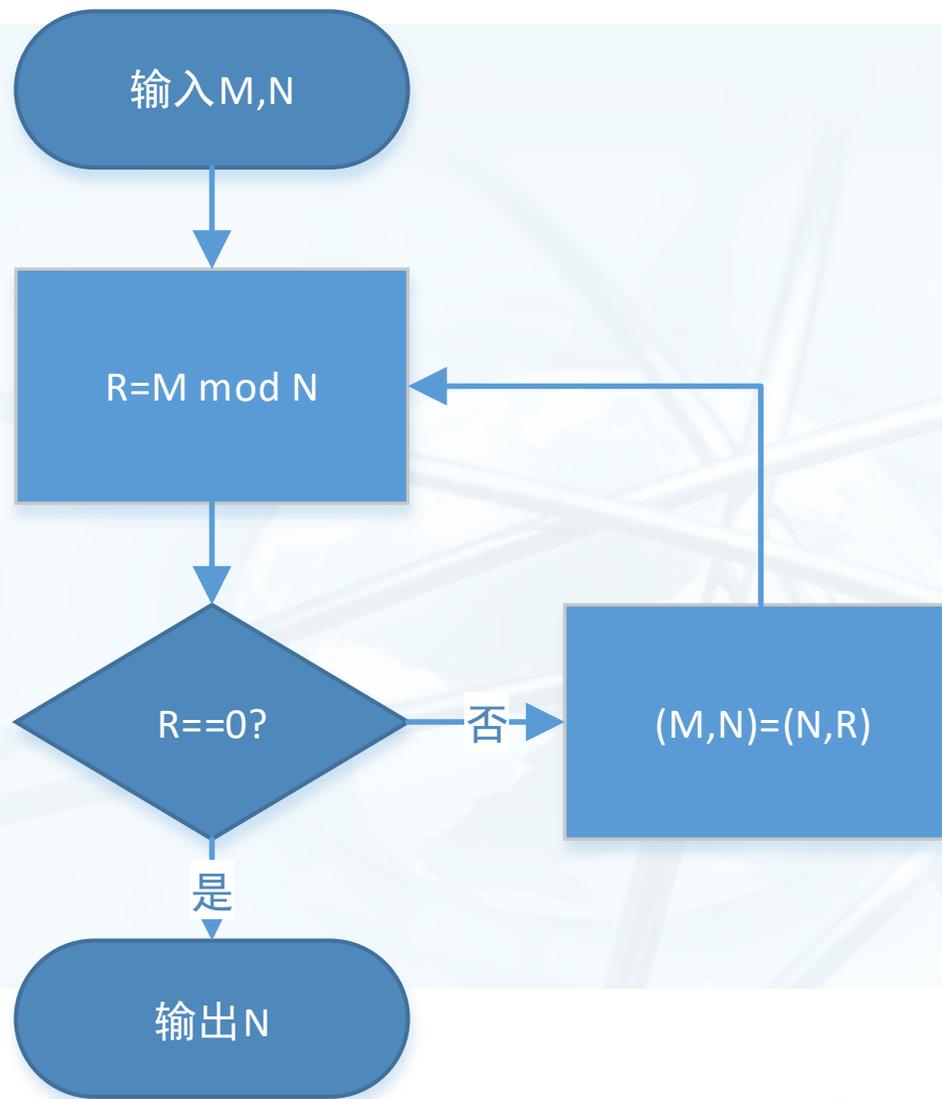
辗转相除法求最大公约数

› 辗转相除法处理大数时非常高效

它需要的步骤不会超过较小数的位数（十进制下）的五倍

加百利·拉梅(Gabriel Lamé)于1844年证明了这点，

并开创了计算复杂性理论。



# 计算复杂性

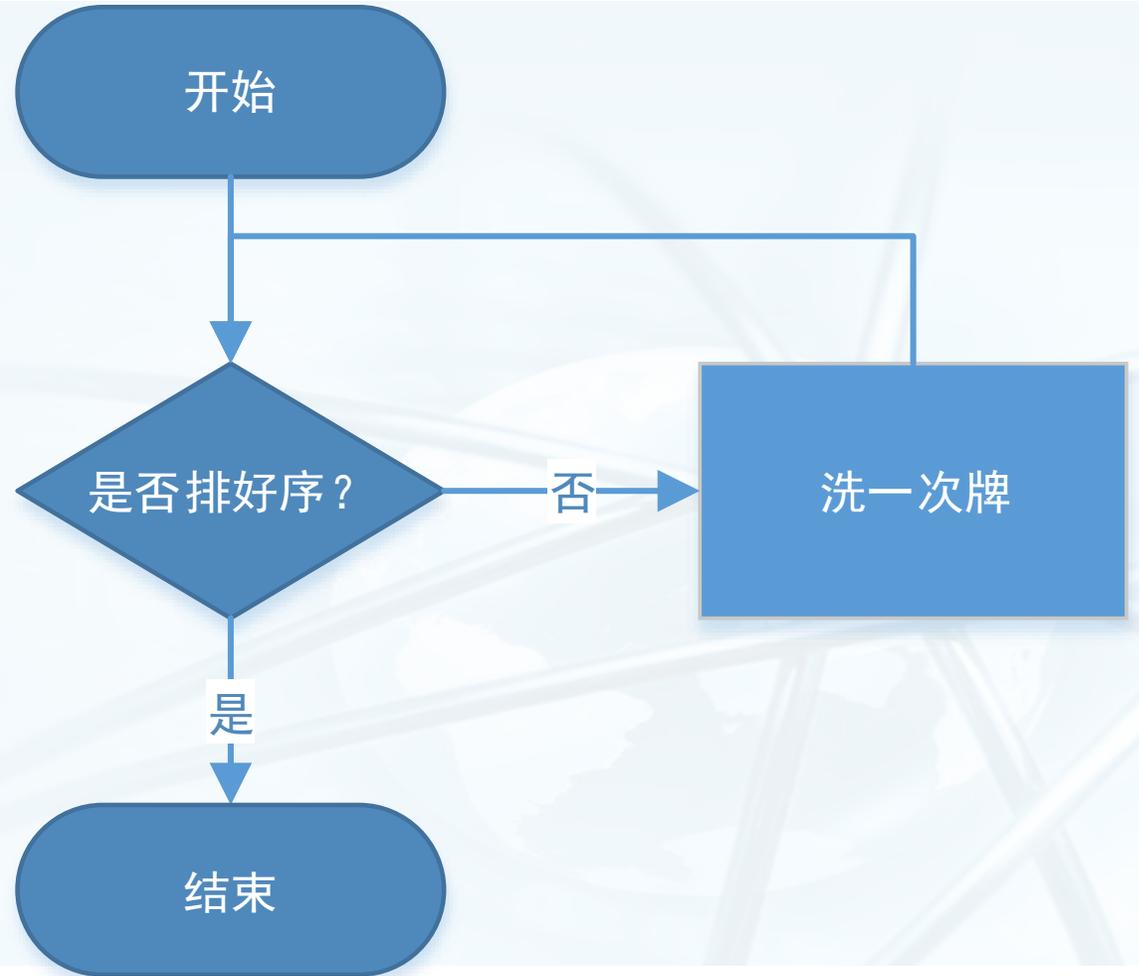
- › “基于有穷观点的能行方法”的“可计算”概念仅仅涉及到问题的解决**是否能在有限资源内**（时间/空间）完成，并不关心具体要花费**多少**计算步骤或多少存储空间
- › 由于人们对资源（时间/空间）的拥有相当有限，对于问题的解决需要考虑其可行性如何，人们发现各种不同的问题，其难易程度是不一样的
  - 有些问题非常容易解决，如基本数值计算；
  - 有些问题的解决程度尚能令人满意，如表达式求值、排序等；
  - 有些问题的解决会爆炸性地吞噬资源，虽**有解法**，但没什么可行性，如哈密顿回路、货郎担问题等
- › 定义一些衡量指标，对问题的难易程度（所需的执行步骤数/存储空间大小）进行分类，是计算复杂性理论的研究范围

# 计算复杂性

› 但对于同一个问题，也会有不同的解决方案，其解决效率上也是千差万别，例如排序问题，以 $n$ 张扑克牌作为排序对象

一般人们会想到的是“冒泡”排序，即每次从牌堆里选出一张最小的牌，这样全部排完大概会需要 $n^2$ 量级的比较次数

😊 另一种有趣的“Bogo”排序方法，洗一次牌，看是否排好序，没有的话，接着洗牌，直到排序成功！这样全部排完，平均需要 $n*n!$ 量级的比较次数（最坏的情况是永远都无法完成排序）



# 计算复杂性

- › **计算复杂性理论研究问题的本质，将各种问题按照其难易程度分类，研究各类问题之间的难度级别，并不关心解决问题的具体方案**
- › **而数据结构与算法，则研究问题在不同现实资源约束情况下的不同解决方案，致力于找到具体的计算资源条件下，效率最高的那个算法方案**
  - 不同的硬件配置（手持设备、平板电脑、PC设备、超级计算机）
  - 不同的运行环境（单机环境、多机环境、网络环境、小内存）
  - 不同的应用领域（消费级别、工业控制、生命维持系统、航天领域）
  - 甚至不同的使用状况（正常状况、省电状况）
- › **如何对具体的算法进行分析，并用衡量指标评价其复杂度，我们在后面的课程里还会详细介绍**

# 不可计算问题

## › 有不少定义清晰，但无法解决的问题

并不是目前尚未找到，而是在“基于有穷观点的能行方法”的条件下，已经被证明并不存在解决方案

## › “停机问题”：判定任何一个程序在任何一个输入情况下是否能够停机

## › 不可计算数：几乎所有的无理数，都无法通过算法来确定其任意一位是什么数字

可计算数很少：如圆周率 $\pi$ ，自然对数的底 $e$

$$\pi = \frac{1}{2^6} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{10n}} \left( -\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9} \right)$$

$$e = \sum_{n=0}^{\infty} \frac{1}{n!} = \frac{1}{0!} + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \frac{1}{4!} + \dots$$

# 去除bug的人工智能

- › 原来Yelp本打算利用深度学习来消除已有的Bug、优化提升用户体验
- › 神经网络将所有的内容都做了删除。不得已，Yelp只能将版本进行了回滚。



Yelp: Local Food & Services

Restaurant & Delivery Finder

UPDATE

4.4 ★★★★★  
241K Ratings

#3  
Travel

12+  
Age

## What's New

[Version History](#)

Version 12.27.0

1d ago

We apologize to anyone who had problems with the app this week. We trained a neural net to eliminate all the bugs in the app and it deleted everything. We had to roll everything back. **To be fair, we were 100% bug-free... briefly.**

# 突破极限



## › 新的计算处理技术

硅光芯片、DNA计算、量子计算、人列计算机

## › 新的计算思路?

超大规模分布式计算: SETI@Home (Search for ExtraTerrestrial Intelligence)

能够突破“基于有穷观点的能行方法”么?

## › 一个有57000位作者的Nature大作的故事

“如果无数多的猴子在无数多的打字机上随机地乱敲, 并持续无限久的时间, 那么在某个时候, 必然有只猴子会打出莎士比亚的全部著作。”

FoldIt这款游戏就是让你来组装蛋白。玩家要做的, 就是在给定一个目标蛋白的情况下, 用各种氨基酸进行组装, 最终拼凑出这个蛋白的完全体, 完成蛋白质结构解析的工作 (详见末尾参考阅读)



## Predicting protein structures with a multiplayer online game

Seth Cooper<sup>1</sup>, Firas Khatib<sup>2</sup>, Adrien Treuille<sup>1,3</sup>, Janos Barbero<sup>1</sup>, Jeehyung Lee<sup>3</sup>, Michael Beenen<sup>1</sup>, Andrew Leaver-Fay<sup>2,4</sup>, David Baker<sup>2</sup>, Zoran Popović<sup>1</sup>, and >57,000 Foldit players<sup>5</sup>

# 计算机科学研究什么

- › 计算机科学不仅仅是对计算机的研究，虽然计算机是非常重要的计算工具
- › 计算机科学主要研究的是问题、问题解决过程，以及问题的解决方案  
包括了前述的计算复杂性理论，以及对算法的研究
- › 为了更好地处理机器相关性或独立性，引入了“抽象abstraction”的概念，用以从“逻辑logical”或者“物理physical”的不同层次上看待问题及解决方案
- › 一个关于“抽象”的例子：汽车  
从司机观点看来，汽车是一台可以带人去往目的地的代步工具，司机上车、插钥匙、点火、换档、踩油门加速、刹车。从抽象的角度说，司机看到的是汽车的“逻辑”层次，司机可以通过操作各个机构来达到运输的目的，这些操纵机构（方向盘、油门、档位）就称为“接口interface”

# 计算机科学研究什么

## 一个关于“抽象”的例子：汽车

另一方面，从汽车修理工的角度来看同一辆汽车，就会相当不同，他不仅要会驾驶汽车，而且还需要清楚每项功能是如何实现的，如发动机工作原理，档位操作的机械结构，发动机舱内各处温度如何测量和控制等等，这些构成了汽车的“物理”层次。



# 计算机科学研究什么

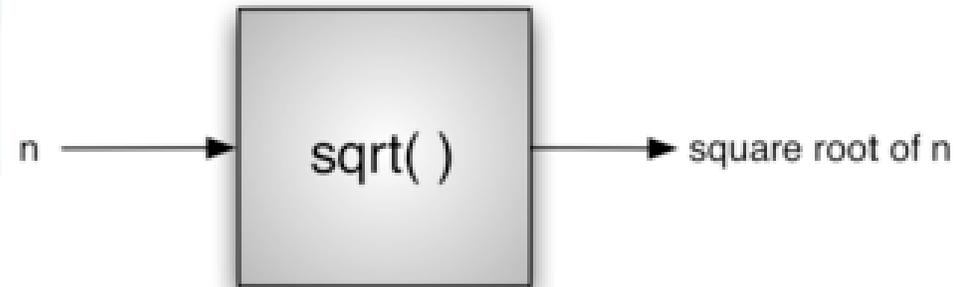
## › 在我们熟悉的计算机使用上也是如此

从一般大众用户观点看来，计算机可以用来编辑文档、收发邮件、上网聊天、处理照片等等，这些用户都不需要具备对计算机内部如何处理的知识，实现这些功能是计算机的“逻辑”层次，而对于计算机科学家、程序员、技术支持、系统管理员来说，就必须要了解从硬件结构、操作系统原理到网络协议等各方面的低层次细节。

## › “抽象”发生在各个不同层次上，即使对于程序员来说，使用编程语言进行编程，也会涉及到“抽象”

如计算一个数的平方根，程序员可以调用编程语言的库函数`math.sqrt()`，直接得到结果，而无需关心其内部是如何实现，这种功能上的“黑盒子”称作“过程抽象procedural abstraction”

```
>>> import math
>>> math.sqrt(16)
4.0
>>>
```



# 什么是数据

- › **数据**：计算机所能处理的**所有符号**的集和  
数字、字符、图像、声音、文本
- › **数据元素**：数据这个集和中的一个**个体**
- › **数据对象**：一类数据元素，组成一个数据对象  
所有的桌子、所有的同学
- › **数据项**：一个数据元素中有若干个数据项，是有意义的**最小单元**  
员工号、手机号、身份证号好

# 什么是数据结构 (1)

› 举例：160687700118616021531110108197705316333

员工号：1606877001

手机号：18616021531

身份证号：110108197705316333

**杂乱的数据是没有意义的，不能传达信息和交流**

# 什么是数据结构 (2)

› 例2: 电话号码本  $\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle, \dots, \langle a_n, b_n \rangle$

其中:  $a_i$  为姓名,  $b_i$  为电话号码

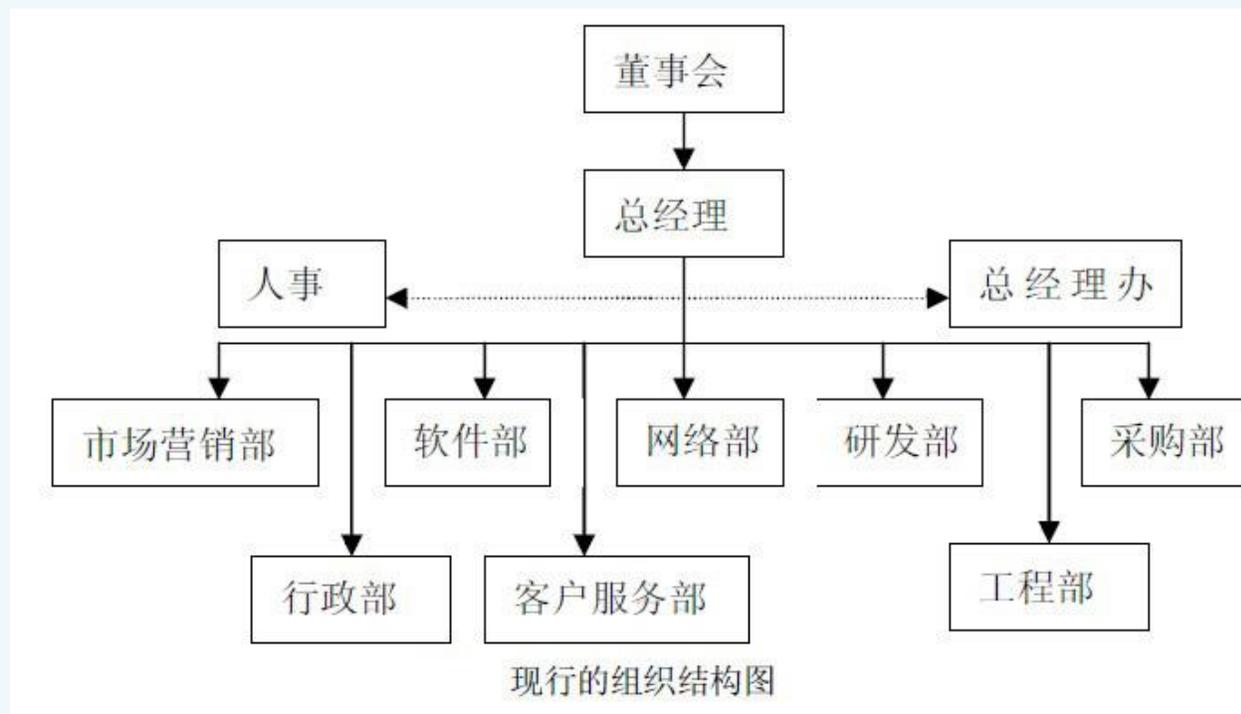
要求: 设计一个算法, 给定姓名, 找到号码

- 如果姓名和号码排列次序无规律  
则只能一个一个查找
- 如果姓名按字典顺序组织, 则查找快的多

**数据之间是有联系的, 合理的利用联系,  
常常影响算法的效率和选择**

# 什么是数据结构 (3)

例3:



数据之间是有结构的，  
比如分层结构（树状结构）

# 什么是数据结构 (4)

## 例3：学校学籍管理

每个同学包括：姓名、性别、出生年月、学号、院系、学分

对学籍管理有如下操作

- 查找：查找某个同学的信息
- 加入：新学年入学
- 删除：高年级毕业

**在某种数据结构上，可以定义一组运算**

# 数据结构的4个核心问题

- › 数据结构：按照逻辑关系组织起来的一批数据，按一定的存储结构把它存储在计算机中，并在这些数据上定义了相关运算的集合
- 数据的各种逻辑结构（关系）和存储（物理）结构，以及对应关系
  - 每种结构所适应的各种运算
  - 针对问题，设计相应的算法
  - 能分析算法的效率，包括时间和空间复杂度



# 数据结构三要素

- › **逻辑结构:** 定义了结构中的基本元素之间的相互关系。  
数据元素之间的逻辑关系；二元组  $(D, R)$ ，其中  $D$  是数据元素的有限集合， $R$  是  $D$  上的关系的有限集。
- › **存储结构:** 给出了结构中的基本元素之间的存储方式  
包括元素的表示和关系的表示。
- › **数据的运算:** 这个结构具有的行为特征  
体现为在存储结构上的具体实现算法。



# 数据的存储结构

- › **顺序存储结构**：它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。
- › **链式存储结构**：它不要求逻辑上相邻的结点在物理位置上亦相邻，结点间的逻辑关系是由附加的指针字段表示的。
- › **索引存储结构**：除建立存储结点信息外，还建立附加的索引表来标识结点的地址。
- › **散列存储结构**：就是根据结点的关键字直接计算出该结点的存储地址。

四种存储结构既可单独使用，又可组合使用

# 顺序存储结构

**顺序存储结构：**它是把逻辑上相邻的结点存储在物理位置相邻的存储单元里，结点间的逻辑关系由存储单元的邻接关系来体现。

逻辑地址      数据元素

0	$k_0$
1	$k_1$
...	...
$i$	$k_i$
...	...
$n-1$	$k_{n-1}$

存储地址      数据元素

$\text{Loc}(k_0)$	$k_0$
$\text{Loc}(k_0)+c$	$k_1$
...	...
$\text{Loc}(k_0)+i*c$	$k_i$
...	...
$\text{Loc}(k_0)+(n-1)*c$	$k_{n-1}$

# 链接 (链状) 存储结构

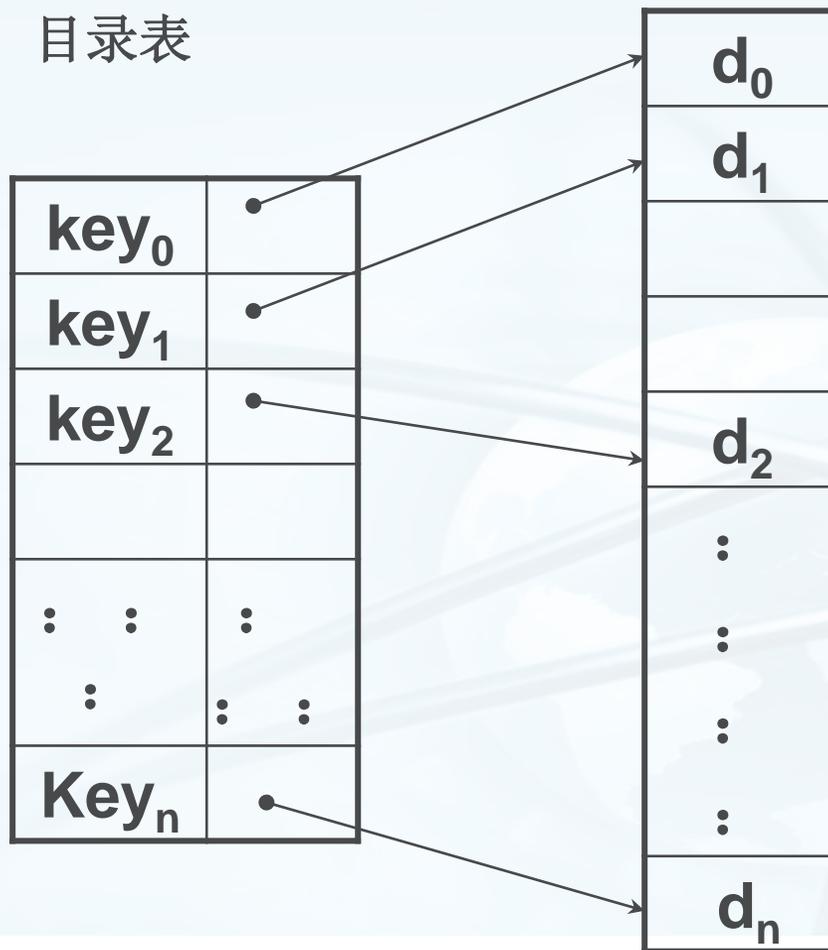
**链接 (链状) 存储结构:** 它不要求逻辑上相邻的结点在物理位置上亦相邻, 结点间的逻辑关系是由附加的指针字段表示的。



存储地址	数据域	指针域
1	Li	43
7	Qian	13
13	Sun	1
19	Wang	NULL
25	Wu	37
31	Zhao	7
37	Zheng	19
43	Zhou	25

# 索引存储结构

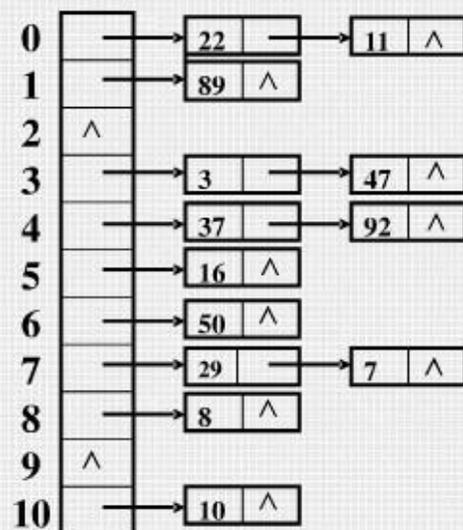
**索引存储结构：**除建立存储结点信息外，还建立附加的索引表来标识结点的地址。



# 散列 (hash, 哈希) 存储结构

› **散列存储结构：**就是根据结点的关键字直接计算出该结点的存储地址。

例：设 { 47, 7, 29, 11, 16, 92, 22, 8, 3, 50, 37, 89 } 的哈希函数为： $\text{Hash}(\text{key}) = \text{key} \bmod 11$ ，用**拉链法**处理冲突，建表。



有冲突的元素可以插在表尾，也可以插在表头。

# 数据的运算

› **定义在逻辑结构上的一系列操作以及这些操作在存储结构上的实现；**  
数据的运算是定义在逻辑结构上的，而具体的实现是基于存储结构。

› **常用的运算：**

检索/定位

插入

删除

修改

排序

# 主要介绍的数据结构

- **线性表**：线性表中各元素之间是一种简单的“线性”关系。
  - **顺序表和链表**：是两种常用的实现线性表的数据结构。
  - **字符串**：字符串也是一种特殊的线性结构，以字符为元素。
  - **栈**：栈元素的存入和取出按照后进先出原则，最先取出的总是在此之前最后放进去的那个元素。
  - **队列**：队列实现先进先出的原则，最先到达的元素也最先离开队列。

# 主要介绍的数据结构

- **树与二叉树**：树和二叉树都属“树形结构”，在逻辑上表示了结点的层次关系。
- **字典**：字典是一种二元组的集合，每个二元组包含着一个关键码和一个值。
  - 按关键码进行**检索**是字典中最常用的操作。
- **排序**：针对由一组记录组成的文件，每个记录由若干字段组成，以**排序码**为依据进行排序。
- **图**：包括一个结点集合和一个边集合，边集合中每条边联系着两个结点。

# 数据结构的4个核心问题

- 数据的各种逻辑结构（关系）和存储（物理）结构，以及对应关系
- 每种结构所适应的各种运算
- 针对问题，设计相应的算法
- 能分析算法的效率，包括时间和空间复杂度



# 什么是算法

› **算法是利用一个有限的指令集，遵循指令流完成特定的功能**

计算的狭义解释

完成从输入=》输出的过程

算法的基本特性：

- **有穷性**：算法经过有限步以后结束
- **确定性**：算法的下一步必须是明确的
- **可行性**：每一步都可行、且最终是正确的

# 什么是编程Programming

- › **编程是通过程序设计语言，将算法变为计算机可以执行的代码的过程**  
没有算法，编程无从谈起  
图灵奖获得者Niklaus Wirth的名言：算法+数据结构=程序
- › **程序设计语言需要为算法的实现提供实现“过程”和“数据”的机制**  
具体表现为程序设计语言中的“控制结构”和“数据类型”
- › **实现算法所需要的基本控制结构，程序设计语言均有语句相对应**  
顺序处理、分支选择、循环迭代
- › **程序设计语言也提供了最基本的数据类型来表示数据，如整数、字符等**  
但对于复杂的问题而言，这些基本数据类型不利于算法的表达
- › **还需要引入一种控制复杂度的方法，便于清晰高效地表达算法**

# 算法≠程序

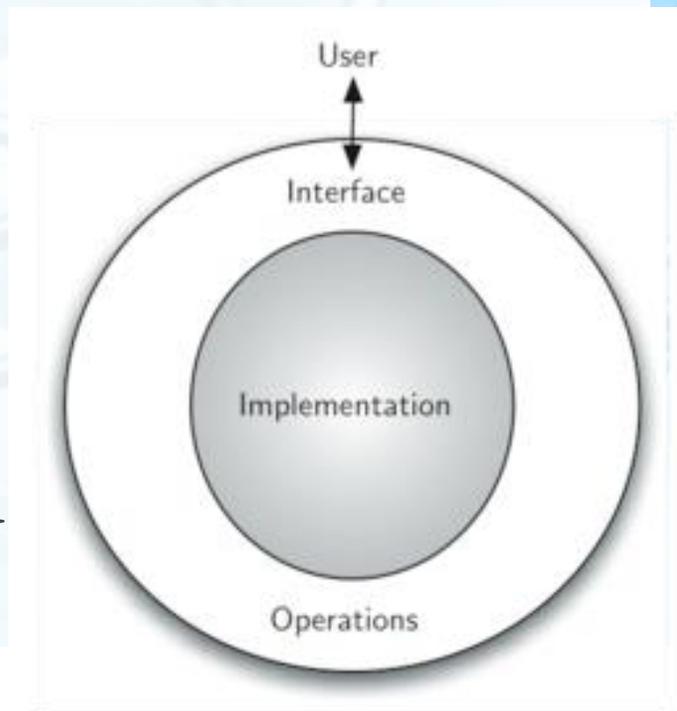
- › **算法**是解决问题的一个方法或者过程，一个问题可以有多个算法
- › **程序**是算法在某种程序语言下的实现  
程序=算法+数据结构

主要区别在于：有穷性、确定性、正确性

- 程序可以是无穷的，例如OS
- 程序可以使错误的，算法必须正确
- 程序一定基于某种语言，在某种机器上可以执行
- 算法可以用框图、自然语言描述

# 为什么要学习数据结构和抽象数据类型

- › 为了控制问题和问题解决过程的复杂度，我们需要利用抽象来保持问题的“**整体感**”而不会陷入到过多的细节中去
- › 这要求对现实问题进行建模的时候，对算法所要处理的数据，也要保持与问题本身的一致性，不要有太多与问题无关的细节
- › 前面谈到的“过程抽象”启发我们进行“**数据抽象**”  
相对于基本数据类型的“抽象数据类型ADT:Abstract Data Type”  
ADT是对数据进行处理的一种逻辑描述，并不涉及如何实现这些处理  
ADT建立了一种对数据的“封装encapsulation”  
封装技术将可能的处理实现细节隐蔽起来，能有效控制算法的复杂度



# 为什么要学习数据结构和抽象数据类型

- › **数据结构Data Structure，是对ADT的具体实现，同一种ADT可以采用不同的数据结构来实现**

数据结构采用程序设计语言的控制结构和基本数据类型来实现ADT所提供的逻辑接口，属于ADT的“物理”层次

对数据实现“逻辑”层次和“物理”层次的分离，可以定义复杂的数据模型来解决问题，而不需要考虑此模型如何实现

由于对抽象数据类型可以由多种实现方案，这种独立于实现的数据模型让底层程序员专注于实现和优化数据处理，而无须改变数据的使用接口，让用户专注于问题的解决过程

- › **如电动车与汽油车，底层动力实现不同，但开车的操作接口（方向盘、油门、刹车、档位）基本都是相同的**

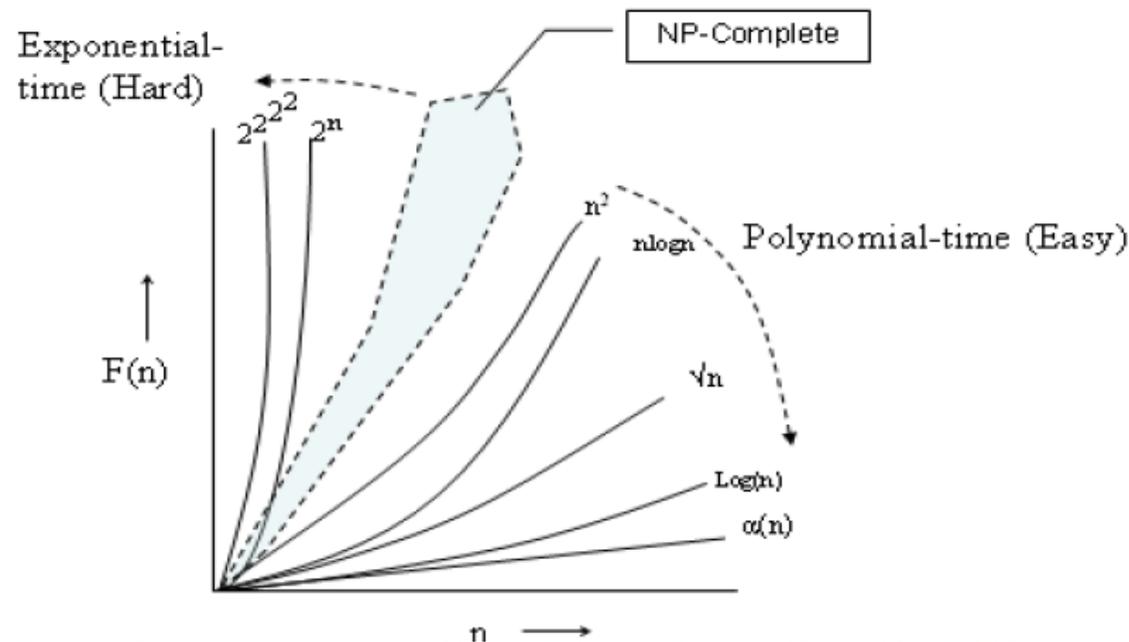
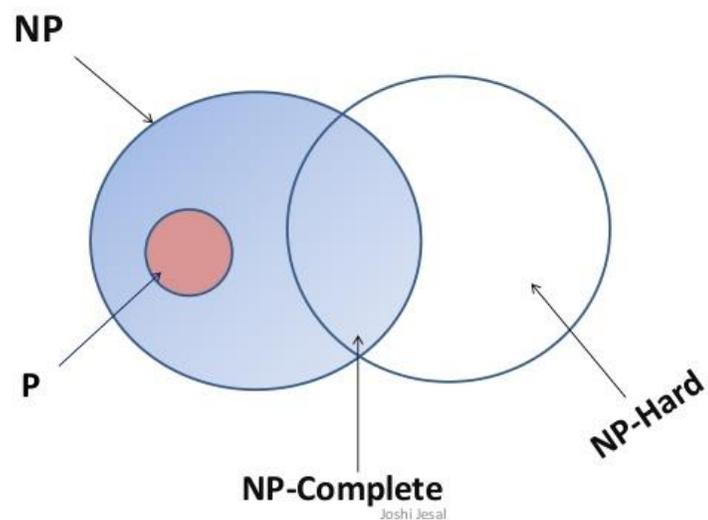
# 为什么要学习算法

- › 首先，通过学习各种不同问题的解决方案，有助于我们在面对未知问题的时候，能够根据**类似问题**的解决方案来更好解决
- › 其次，各种算法通常有较大的差异，我们可以通过算法分析技术来**评判算法**本身的特性，而不仅仅根据算法在特定机器和特定数据上运行的表现来评判它  
即使同一个算法，在不同的运行环境和输入数据的情况下，其表现的差异可能也会很大
- › 在某些情况下，我们或许会碰到棘手的难题，得能**区分**这种问题是根本不存在算法，还是能找到算法，但需要耗费大量的资源
- › 某些问题的解决可能需要一些折衷的处理方式，我们需要学会在不同算法之间进行**选择**，以适合当前条件的要求



# 计算机可解问题的大致难度分类

Relationship among P, NP, NP-Complete and NP-Hard



# 参考阅读

- › <http://blog.sciencenet.cn/blog-2371919-866686.html>
- › [http://en.wikipedia.org/wiki/Effective\\_method](http://en.wikipedia.org/wiki/Effective_method)
- › <http://mindhacks.cn/2006/10/15/cantor-godel-turing-an-eternal-golden-diagonal/>
- › <http://www.matrix67.com/blog/archives/4812>
- › P vs. NP: 从一则数学家谋杀案说起  
<http://www.guokr.com/article/437662/>
- › bogo排序:  
<http://zh.wikipedia.org/wiki/Bogo%E6%8E%92%E5%BA%8F>
- › <http://www.matrix67.com/blog/archives/901>

# 参考阅读

- › **背包问题:** <http://baike.baidu.com/view/841810.htm>
- › **哈密顿回路:** <http://baike.baidu.com/view/1031680.htm>
- › **货郎担问题:** <http://baike.baidu.com/view/267558.htm>
- › **睡眠排序:** <http://blog.csdn.net/zmazon/article/details/8514088>
- ›  **$\pi$ 里包含了所有可能的数字组合吗?**  
<http://www.guokr.com/article/439682/>
- › **57000人完成的Nature大作 世界上作者最多的论文**  
<http://www.biodiscover.com/news/research/117459.html>