计算概论

《什么是“计算”-2》

裘宗燕

北京大学数学学院信息科学系

2015, 春季

现实的计算

- 问题：现实中的计算机真的等价于通用图灵机吗？
- 两个重要不同点：
  - 图灵机有无穷长的存储带，现实计算机的存储器是有穷的
  - 图灵机没有时间概念，而实际计算机操作需要时间。无论一次操作多快，大量操作累计可能超过任何给定时间
- 这两个问题是本质性的，与所有计算问题有关，因为
  - 在每个计算问题的求解过程中都需要存储相关的数据
  - 每个计算问题都要通过一系列操作完成，操作需要时间
- 这些问题为计算领域提出了新的研究课题：

  数据存储量和操作时间将怎样影响计算？怎样思考/理解能用实际计算机（或其他计算工具）解决的问题和计算的局限性？

问题，实例，算法

- 为抽象地研究现实的计算问题，假定
  - 计算中的数据存储有一种基本单元，每个单元只能保存固定的一点有限数据（一个单位的空间）
  - 计算中的一次操作要消耗一个单位的时间
- 考虑计算问题时，需要区分三个概念
  - 问题：一个问题 **W** 是一个需要解决的需求（需用计算求解），如求实数的平方根，找出大于整数 **m** 的第一个素数等
  - 问题实例：问题 **W** 的一个实例 **w** 是该问题的一个具体例子，如求 **2.0** 的平方根，找出大于 10^{10} 的第一个素数
  - 算法：解决问题 **W** 的一个算法 **A**，是能求解 **W** 的所有实例的一个计算过程的严格描述。对 **W** 的实例 **w**，实施算法 **A** 描述的计算过程能得到所需结果。如平方根算法能求出 **2.0** 的平方根，求素数算法能得到大于 10^{10} 的第一个素数

算法设计与分析

- 用计算机解决问题，需要开发解决问题的方法（算法）。解决重要问题的算法不仅具有理论价值，也有实际价值。例如：
  - google 的网络检索算法，有效利用大量计算机，ranking
  - 高安全和高效率的加密解密算法
- 算法是计算机领域最重要的研究课题（遍及所有研究/应用领域）

  两件工作：总结/发现需要解决的问题，开发相关的高效算法
- 人们也考查算法的设计与分析工作，总结算法研究的经验
  - 总结出一批设计算法的重要模式（思路）
  - 分析具体算法的性质，主要是时间和空间开销
- 算法设计是创造性工作，需要人的智慧，不能自动化

  算法设计：从问题的说明性描述到解决问题的过程性描述

算法的复杂性（复杂性）

- 对解决某个具体计算问题 W 的具体算法 A
  - 考虑 A 计算中的存储开销和时间开销
  - 算法 A 应该能处理问题 Q 的任一个实例，具体实例的规模不同，具体计算的实际开销与实例的规模有关
- 为反映问题实例的具体规模对计算代价的影响，我们把一个算法的计算（时间和空间）代价定义为问题实例规模的函数

  可能有一些问题，其时间（或空间）开销函数随规模扩大增长得很快，另一些问题的开销函数增长较慢

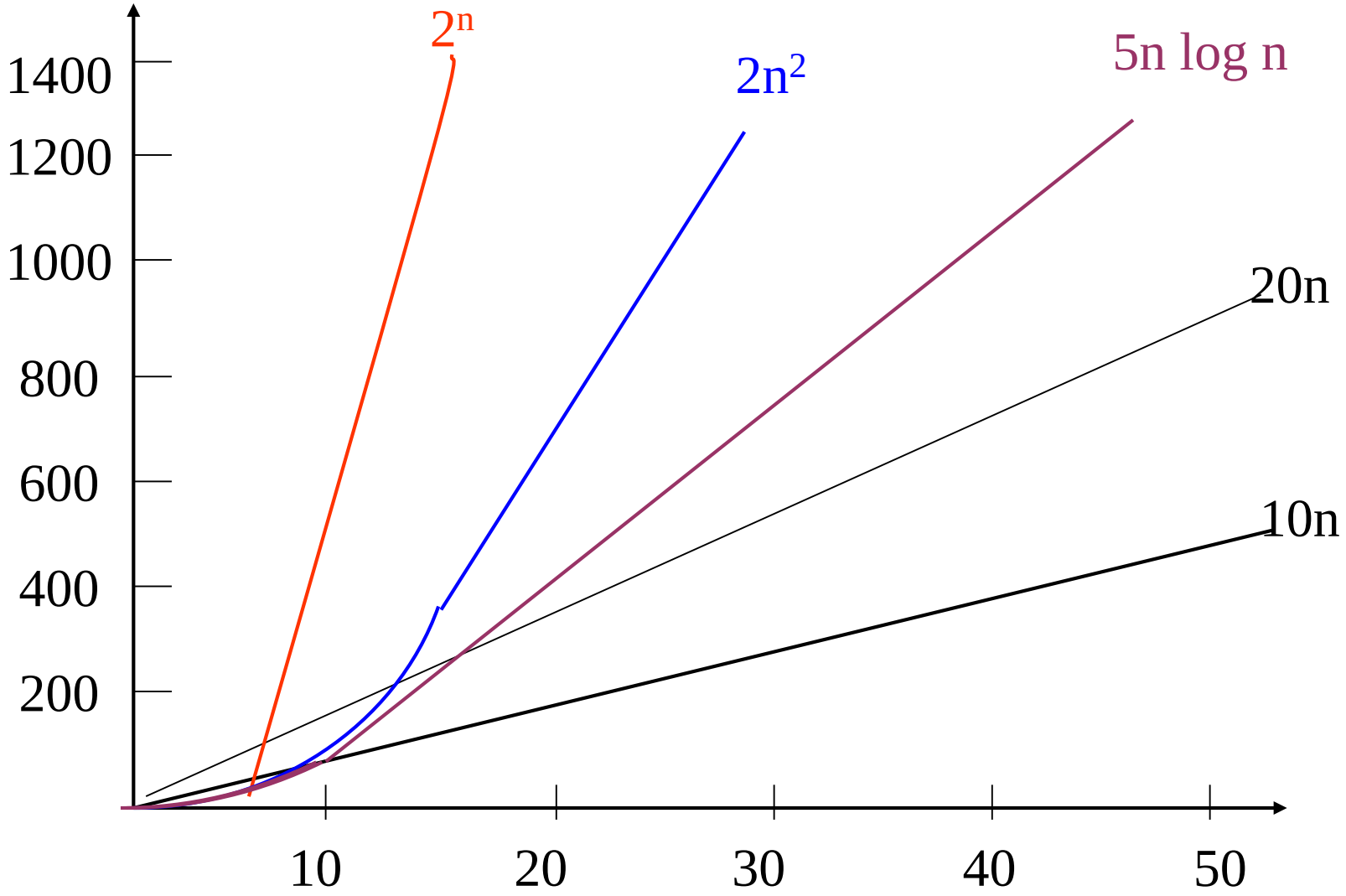
  用函数关系反映计算的性质，实际上是描述一种增长趋势。这两个函数关系称为算法的时间代价和空间代价
- 但不同实际机器的操作执行速度、存储单元大小都可能不同。理论研究应该抽象，应排除具体机器的非普遍性特征，反映问题的共性和本质

算法的复杂性

- 采用的方法与数学分析中有关无穷大的讨论类似
  - 忽略具体算法的常量特征，只考虑其量级
  - 例如 $2\times n^2$ 和 $5\times n^2$，认为属于同一量级
- 算法的复杂性度量，算法在处理规模为 n 的实例时
  - 所需存储空间的量级，称为算法的空间复杂性
  - 所需要的计算时间的量级，称为算法的时间复杂性
- 例如，如果一个算法的时间代价随着实例规模的增长而线性增长（成比例增长），就说这个算法具有线性的时间复杂性

  例如，求一个表中的元素之和或平均值
- 常见复杂性：常量的，O(1)；对数的，O(log n)；线性的，O(n)；O(n log n)；平方的，$O(n^2)$；立方的，$O(n^3)$；指数的，$O(2^n)$

不同复杂性的增长


2^n
$2n^2$
5n log n
20n
10n
1400
1200
1000
800
600
400
200
10
20
30
40
50

算法复杂性的考量

- 用同一个算法解决问题，同样规模实例的计算开销可能不同
  - 求一组数据的平均值，同样大小的组开销差不多
  - 在一组数据里找第一个小于 0 的项，情况不同
- 几种可能有用的考虑（以时间为例），对规模为 n 的数据：
  - ① 考虑完成算法 A 最少需要多少时间
  - ② 考虑完成算法 A 最多需要多少时间
  - ③ 考虑完成算法 A 平均需要多少时间
- ① 价值较小，没提供太多有用信息（最乐观的估计用处不大）
- ② 是一种保证，某时间内工作一定完成（最坏情况时间复杂性）
- ③ 是对算法 A 的全面评价，但没有保证。它还依赖于实例的实际分布，较难确定（平均时间复杂性）

算法和问题的复杂性

- 解决同一个问题，可能存在许多不同的方法（算法），不同算法的时间和空间开销可能不同

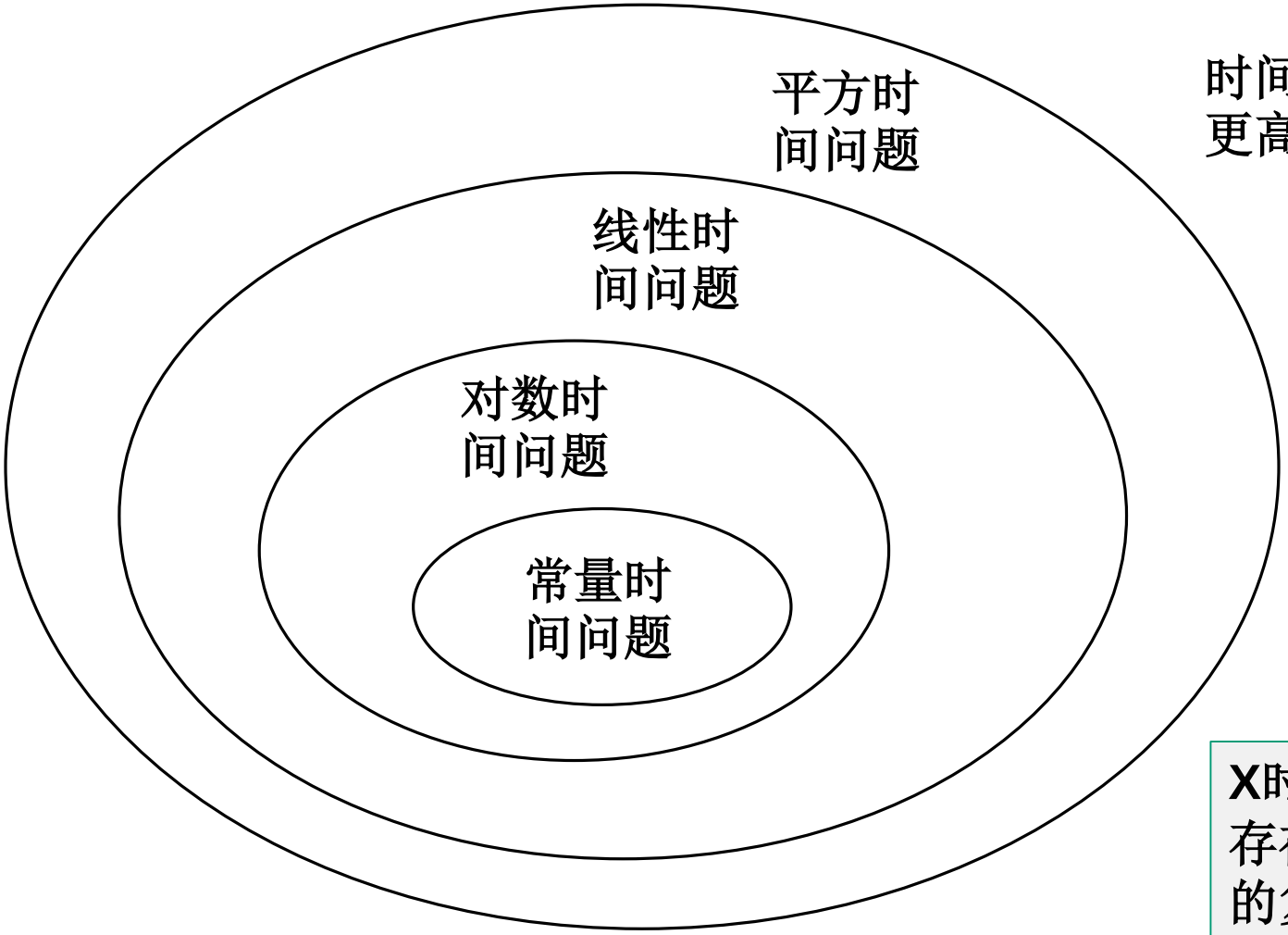
  例，求斐波那契数列的项，复杂性可能与 n 为指数或线性关系

- 分析和理解一个算法，得到的结论应该不受算法处理的具体问题实例的规模变化的影响（对所有规模都有效）
- 分析和理解一个问题，得到的结论也应该超越可能的不同求解算法，直指问题本质，消除解决同一问题的具体算法之间的差异
- 人们把问题的时间复杂性（问题的空间复杂性）定义为能解决该问题的最快可能算法的时间复杂性
- 注意：解决一个问题的所有可能算法未必都已知
  - 求解问题W的一个具体算法，给出该问题复杂性的一个上界
  - 将来可能找到更好的算法，得到更好（更紧）的上界

问题的复杂性（计算复杂性）

- 另一可能做法：通过理论研究弄清问题的复杂性
  - 人们已经在理论上证明了一些问题的时间（空间）复杂性
  - 有些很显然。例如，求一个表中所有数值的平均值
- 问题的计算复杂性使我们可以对计算问题做一个分类
  - 有些问题很简单，计算时间与规模无关；有些问题的计算开销与实例规模成正比，复杂性是问题规模的线性函数，有些问题的复杂性是平方的，或立方的，或更高次多项式的
  - 有些问题更复杂，计算复杂性是指数的或者更高
- 人们经常说“这件事很难”，但没有给出“难”的严格定义
  - 计算复杂性概念严格定义了计算问题的难易
  - 复杂性高的问题就是更难的问题

问题的复杂性分类



时间复杂性
更高的问题

.....

X时间问题类：
存在解决该问题的复杂性不超过X时间的算法

计算复杂性

- 人们提出的一种简化认识是：
  - 具有多项式时间复杂性的问题，可以算作是能用计算机在合理时间内处理的问题（直至一定的规模）
  - 具有指数及指数以上复杂性的问题，用计算机只能处理规模很小的问题实例，因此是难问题
- 从计算复杂性研究中产生了二十一世纪的第一大数学问题：

$$P =?= NP$$

P 指在确定性图灵机上多项式时间可以解决的问题类

NP 指在非确定性图灵机上多项式时间可以解决的问题类

非确定性图灵机：每步转换都可以若干个（有穷个）选择，而且该图灵机在工作中的每一步都能够得到一个提示，做出最终将得到问题的解的最好的转换选择

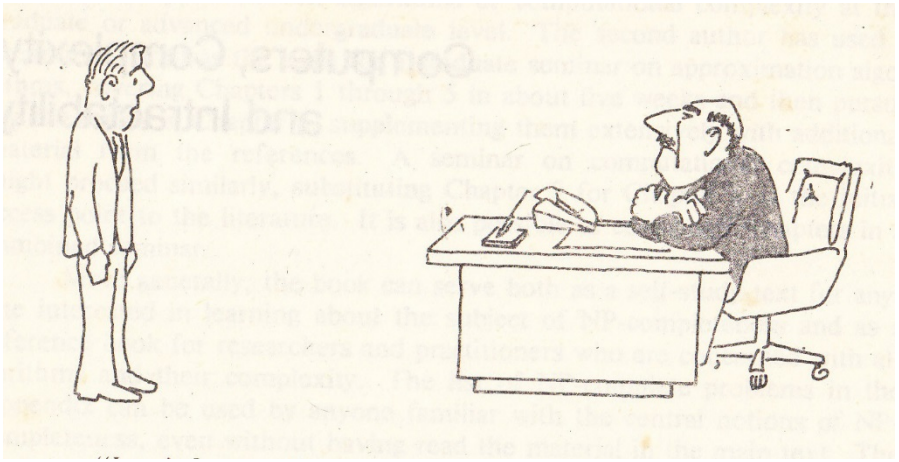
复杂性的意义

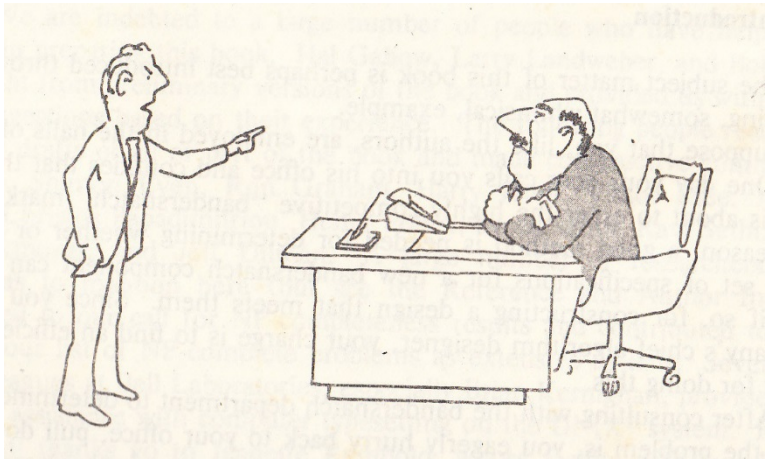
- 计算复杂性决定了一个问题能不能考虑用计算机解决

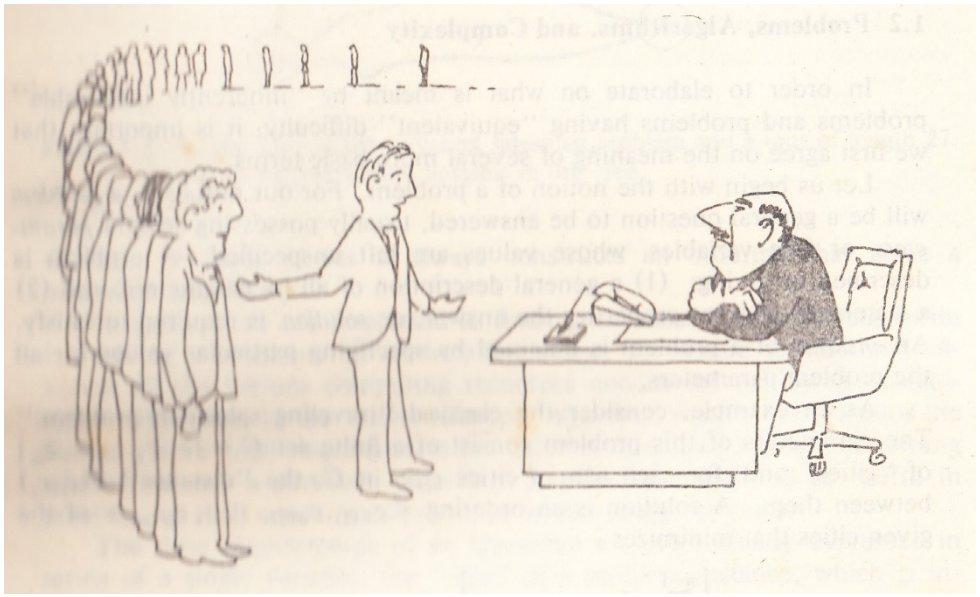
  如果一个问题复杂性太高，要处理的实例又有一定规模，就不能期望通过计算机写程序去解决它

- 人们已经发现了许多在实际中非常重要的问题具有 NP 类中最高的复杂性，称为 NP 完全性问题
  - 任何一个 NP 问题都可以通过一个多项式时间的转换算法，归结到一个 NP 完全性问题
  - 任何一个 NP 完全性问题，都可以通过一个具有多项式时间复杂性的变换转换到任何一个 NP 完全性问题
  - 只要有一个 NP 完全性问题存在多项式时间的算法，所有 NP 问题都有多项式时间算法
  - 只要任何一个 NP 完全性问题有多项式算法，则 P == NP

NP 完全性问题







计算复杂性的两面

- 问题的复杂性说明了实际计算设备的局限性，很多人们关心的问题无法在合理时间内求解：
  - 一些是人们希望做的工作：许多优化问题、求最优解的问题、安排和调度问题等的复杂性很高，很多是NP完全问题
  - 一些是一部分人不希望别人做的事情：例如，人们希望自己的加密方法不能被轻易破解。问题的复杂性能给出一种保证，说明有些问题采用当前设备，不能在合理时间内解决
- 实际上，社会上的各种安全性都依靠复杂性
  - 锁具、安全门、保险箱、金库，等等
  - 原则上说，任何安全保障都可能打破，问题是需要付出多少代价。代价高到一定程度，就没人认为值得做了
  - 安全性的度量：破坏它需要的资源（包括时间、成本等）

复杂性的意义

- 现实中大量重要信息都保存在计算机系统里
  - 数十亿台计算机连在世界范围互通的 intenet 上
  - 计算机之间存在“物理连接”，使从一部计算机取得存储在另一计算机里的信息成为可能
  - 许多信息具有经济、政治、国家利益方面的重要价值
  - 如何保护计算机信息安全，是今天最重要的问题之一
- 计算机（包括网络等）信息保护的基础就是计算的复杂性
  - 破解安全屏障的问题也是一个计算问题
  - 只要破解的复杂性足够高，用目前可行的计算工具在可预料的（有实际价值的）时间内无法破解，就是安全的

图灵机与复杂性

- 前面提出问题的复杂性，是以图灵机上作为衡量的基础。从这个角度思考，可以提出一个问题：

  会不会存在本质上比图灵机更快的计算模型，使得在图灵机上计算的难问题，在该模型上不是难问题？

- 首先，确实存在可以在 **P** 时间里求解 **NP** 完全性问题的模型

  具有无穷并行度的图灵机具有这种能力

- 如果只考虑顺序机器模型（也就是说，这种模型每个时刻只能执行一个动作），人们的研究提出了一个扩展的图灵论题：

  对任一种确定性的计算模型 **M**，都存在一个 **k**。如果对一个问题存在 **T(n)** 时间的 **M** 算法，那么就存在一台确定性的图灵机，可以在 $T(n)^k$ 时间解决该问题

  潜在结论：不存在本质上比图灵机更快的确定性计算模型

图灵机与复杂性

- 对已有的重要计算模型的研究，证明了扩展的图灵论题都有效
- 问题：k 可能会有多大？

  已经证明，对随机访问机器（我们用的计算机都可以归于这种模型），k = 2

  对已经考察过的其他模型，k 值也不大

- 有关 P =?= NP 问题的研究还在继续，一些数学家在研究它。这里收集和一百多篇分别证明 P == NP 和 P=/= NP 的文章

  http://www.win.tue.nl/~gwoegi/P-versus-NP.htm

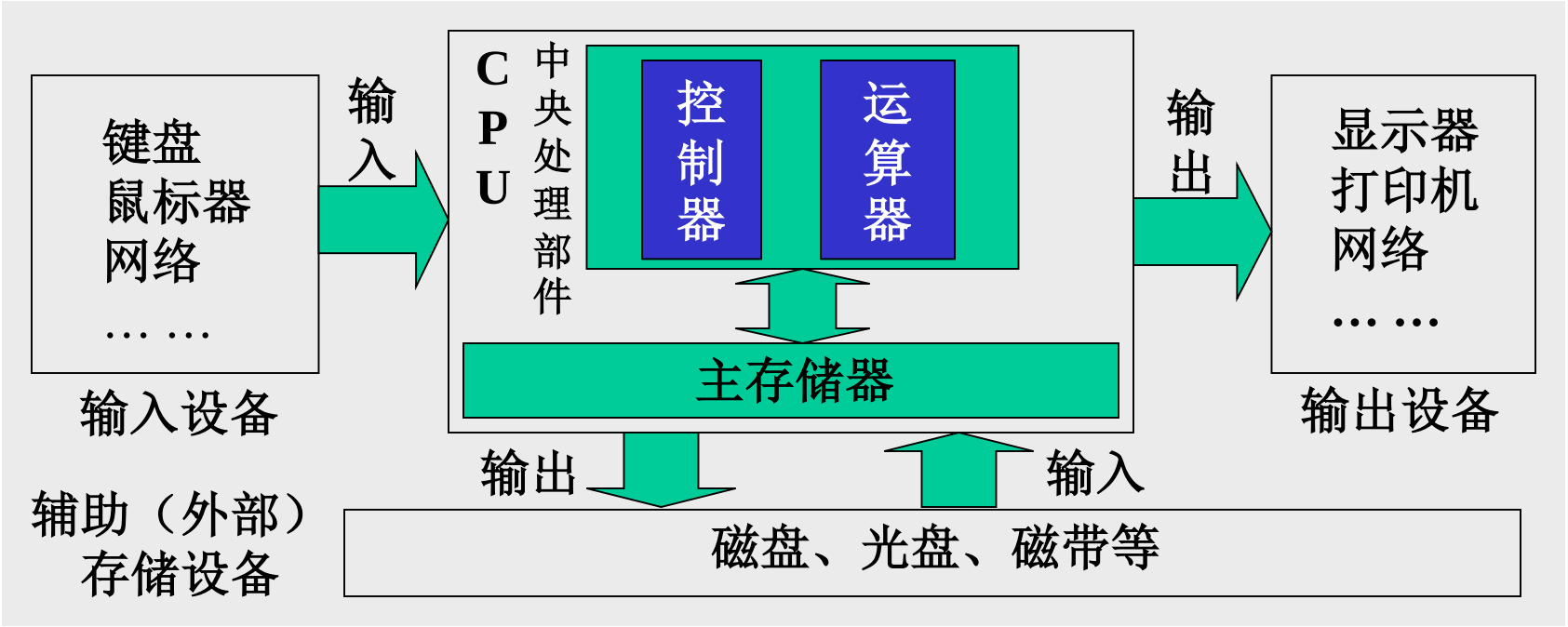
- Gödel 给 von Neumann 的一封信中涉及这个问题。他写道，如果某问题（后证明是 NP 完全问题）有 P 时间算法（P == NP），那么数学家能做的事情都能由机器完成了。
- 量子计算？是一种能行的计算模型吗？

现实的计算机

- 现实世界中的电子计算机（通用电子数字计算机）是用超大规模集成电路（VLSI）和其他元器件构造起来的一种复杂电子设备

  它可以看作通用图灵机的一种实现

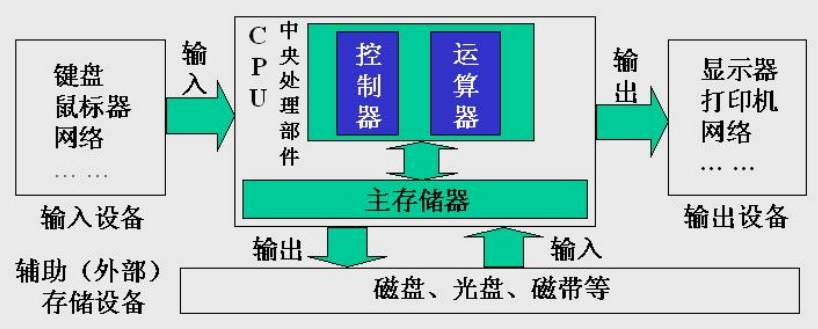
- 电子计算机由一些部件构成，简化的结构图：



现实的计算机

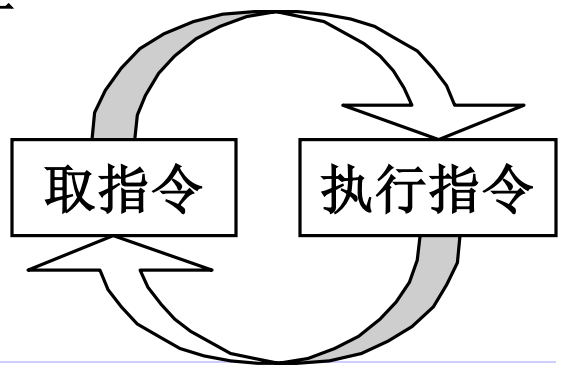
- CPU（中央处理器）处理数据，对应于图灵机的控制器，包括：
  - 一个控制器解释程序的指令（“脑”）
  - 一个运算器执行指令，完成实际操作（“手”）
  - 能执行（不多的）一组指令，每条指令对应一个特定操作。例如移动一个单元的数据，或者一次算术运算等
- 主存储器(内存)存放运行的程序和数据，对应于图灵机存储带

  CPU 可以直接访问。有关存储器的一些细节后面介绍
- 外部存储器保存备用的程序与数据。需要时装入内存供 CPU 使用
- 输入输出设备实现与外部的信息交换：
  - 与用户（使用者，人）
  - 与其他计算机系统
  - 与各种其他设备和网络等



计算机和程序

- 程序就是指令的序列，指挥CPU完成动作。但程序从哪里来？
- ENIAC (1946)：程序由插板等表示(硬件)，速度受限，换程序很困难
- 冯·诺依曼提出“存储程序原理”，为计算机结构奠定基础
  - 原来在存储器里只保存数据。冯·诺依曼的方案是把程序（的某种编码形式）也存入存储器，由CPU自主提取和执行
  - 这样，计算机就摆脱了外界拖累，以电子电路的速度自动运行
  - 按此原理构造的计算机称为“存储程序计算机”（或称冯·诺依曼计算机）。所有主流计算机都是这种计算机
- 实际上，通用图灵机已经采用了“存储程序原理”
- 计算机运行是两步循环（CPU基本循环）
  - 从内存取下条指令，完成指令要求的动作
  - 直至遇到停机指令，或者永远运行下去



计算机和程序

- CPU 绝对服从指挥，严格按指令工作
  - 一个 CPU 的指令一般只有几十到一二百种
  - 实际领域计算的需要千差万别、错综复杂。计算机怎么能满足所有计算的需要呢？
- 答案：通过程序。通过指令的排列，能写出的程序是（可数）无穷多
  - 类似于数字和数，汉字和中文文章的关系
  - 程序对应于通用图灵机存储带上的特定图灵机的编码
- 计算机的工作原理带来两方面优势：
  - 通用性：只需要一种或几种机器，就能满足整个社会无穷无尽、丰富多彩的需要。支持大工业生产，提高性能，减低成本
  - 专用性：运行不同程序时，表现为不同的专用信息处理机，甚至同时表现为多种信息处理机（同时运行多个不同程序）
- 通用性和专用性的完美统一，使计算机成为信息时代的最基础的设备和最锐利的武器

程序行为的实现

- 计算机 CPU 可以执行的指令用一套规定形式的二进制编码表示，构成了一种描述程序的语言，称为机器语言
- 最基本层面的程序就是 CPU 指令简单序列（机器语言程序）

  CPU 控制器能直接执行用机器语言程序

  在 CPU 执行循环的每一步，控制器顺序提取一条指令（当前指令），分析并解释它，命令运算器去执行相应动作

  执行指令时，还可能牵涉到有关的数据，指令里可能包括如何取得被操作数据的信息（同样用二进制编码表示）
- 二进制形式的机器语言不适合人使用，人们开发了与机器语言直接对应的符号形式的语言，称为汇编语言，具有易读的指令形式，例如

  add r1 r2
  mov r1 r2

  从汇编语言到机器语言的翻译完全是机械的，是一种简单“计算”。完成这一工作的程序称为“汇编程序”

程序行为的实现

- 虽然汇编语言的指令意义比较清晰，但仍然是面向计算机的语言，概念低级，与实际应用距离太远，难以描述复杂的计算过程

  1954年IBM发布John Backus领导开发发第一个完全符号形式的高级语言Fortran，开创了计算机程序设计的新时代

- 计算机不能直接执行高级语言程序（设为 P），有两种基本处理方式
  - 把（任一个）P翻译为机器语言程序，然后交计算机执行

    这种翻译是机械的，工作繁琐且细节很多，应该由计算机完成

    完成这种翻译的程序称为编译程序，这种实现方式称为编译。例如，C编译程序能把用C语言写的程序翻译为CPU可执行的程序
  - 在计算机上实现一个程序，它能识别任何P，模拟其计算过程，实现其行为。这个程序称为解释程序，这种实现方式称为解释

    抽象地看，就是用一台通用图灵机模拟另一台通用图灵机，只是两台机器的输入字符集和图灵机编码方式不同

程序行为的实现

- 我们用的是Python Software Foundation支持下用 C 语言开发的官方的 Python 系统，该系统解释我们用 Python 语言编写的程序
  - 这个系统用 C 语言描述，通过 C 语言编译器翻译为 CPU 可以直接执行的机器语言程序，安装后保存在我们的计算机里
  - 启动Python系统后，该系统能识别和执行 Python 语言写的程序
  - 在下层，CPU直接解释执行这个Python系统（也是一个程序）
  - 实际情况比这还要复杂，Python系统并不是直接读取和执行我们写的Python程序，而是先将其翻译为另一种更容易解释和高速执行的内部形式，而后用一个低级解释器（称为虚拟机）执行它
- 这里的每一层都可以看作“等价于”通用图灵机的机器

  在一部通用图灵机上模拟另一部通用图灵机

  在每个层次上，都可以定义任意多不同的专用机器

内存结构模型

- 要理解计算机处理数据的情况，还需要对计算机内存、存储管理方面一些重要基本问题有所了解。现在介绍这方面的一些基本情况
- 基本内存结构：
  - 内存是线性排列的一批存储单元，单元有唯一编号，称为单元地址
  - 单元地址从 0 开始址连续排列，可用地址是一个连续整数区间
  - 对内存单元的访问（存取其中的数据）都通过单元地址进行。因此，要访问一个单元，必须先掌握其地址
  - 基于地址访问单元是常量时间操作，与单元位置或内存大小无关

程序运行中构造、使用、处理的对象，都在这种线性结构里安排位置

内存和对象存储

- 程序运行中建立/存在的每个对象都要占用（或大或小的）一块内存
  - 建立的每个对象都有确定的唯一标识（例如内存位置），在对象的存续期间其标识保持不变，这是一个基本原则
  - 知道一个对象的位置就能访问（使用）它，已知位置访问相应对象的操作可以在常量时间完成
- 如果一个组合对象包含一组元素，它们在一块元素存储区里连续存储，每个元素的存储量相同，有下面情况
  - 设对象的元素存储区的起始位置是 **p**，每个元素占用 **a** 个内存单元，再假设第一个元素编号为 **0**
  - 要访问编号为 **k** 的元素，其位置 **loc(p, a, k)** 可以通过下式计算

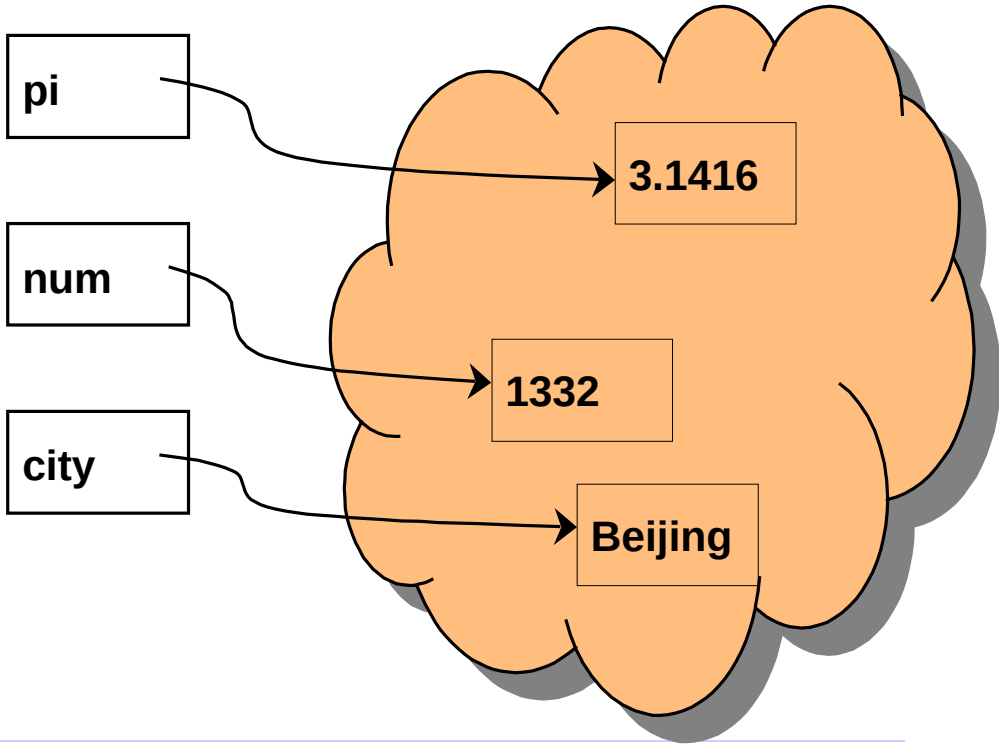
$$loc(p, a, k) = p + k*a$$

  - 显然，计算元素位置（及访问）所需时间与元素编号无关，与组合对象的元素个数无关（访问连续存储的数据，是常量时间操作）

变量和对象

- 程序里的变量（全局的、局部的，以及函数参数）有系统化的存储安排方式，是另一套专门机制，下面讨论中不考虑。实际上：
  - 变量也在内存安排位置，每个变量占用若干存储单元
  - 程序运行中总能找到按作用域可见的那些变量，取得或修改其值

- 在 **Python** 里，可以
  - 通过初始化（或提供实参）给变量（或函数参数）约束一个值（对象）
  - 用赋值修改变量的约束值
- 给变量约束一个值对象，就是把该对象的标识（例如内存位置）存入变量。如图所示



变量和值

- Python 程序里变量的值都是对象，可以是：
  - 基本类型（如逻辑值、浮点数等）的对象，大小固定且比较小
  - 复杂的对象，例如 list 等，可能比较大（包含一组成分对象），需要的存储单元可能不同（不同的 list 有长有短），可能有复杂的内部结构（例如，其元素又可能是复杂的数据对象），等等
- Python 程序运行时内部有一个专门的存储管理系统，负责管理程序可用的内存，支持灵活高效的内存使用
  - 程序中要求建立对象时，为这些对象安排存储
  - 当某些对象不再有用时回收它们占用的存储
  - 存储管理系统屏蔽了具体内存使用细节，减少编程人员的负担
- 在写 Python 程序时，通常不需要关心存储管理的具体细节

  但也应注意，运行中存在的对象都需要存储，过多的对象有可能用完所有可用存储，这种情况下程序只能崩溃

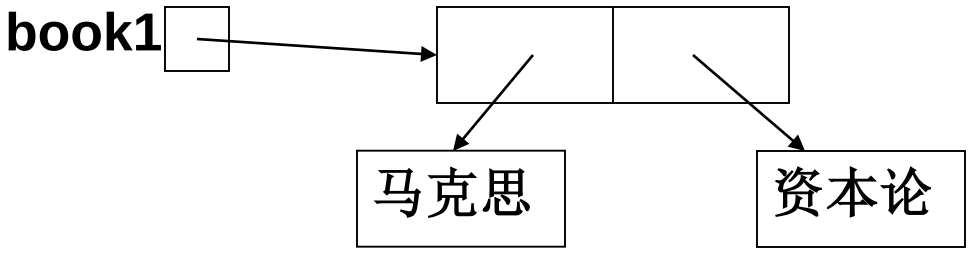
对象创建和变量约束

- 假设要给变量 s 赋值一个新字符串，系统需要：
  - 找一块足够大的内存块，把字符串的内容复制进去
  - 把内存块的地址信息存入变量 s

- 这样做通常不够
  - 内存单元里存储的都是二进制编码，仅从单元里存储的内容无法判断这一字符串到哪里结束
  - 需要有一种安排（约定）。由于字符串可以有任意的长度，一种可能安排是在相应存储块的开始记录字符串长度，例如：

“表示”及其设计

- 程序中生成的对象都要以某种方式保存，因此Python解释器的实现者需要为每种对象设计一种存储方式。这种方式及其效果称为该对象的“表示”（representation）。前例就是为字符串设计了一种表示
- 字符串的结构简单，可以用一块连续存储区表示，类似的情况如数学里的 n 维向量。但并非所有对象都如此
- 假设要表示一类对象，它有两个成员对象，都是字符串，一个表示作者（作者名），一个表示图书标题（书名）
  - 书名和作者都可以用字符串表示，但两个字符串长度不定
  - 作为整体的对象怎样表示，才能支持灵活方便的处理？
- 一种做法是用三块内存（例如做成元组）：



- 变量保存元组的地址
- 元组里存字符串地址
- 这种结构称为链接结构，用于表示复杂的数据

Python 的对象表示

- Python 系统的实现基于一套精心设计的链接结构
  - 各种复杂对象，甚至 Python 程序等，都基于独立的存储块实现，通过链接相互关联
  - 各种数据对象的表示方式，对相关结构上各种操作的效率有着简单性的影响，也间接影响着用 Python 做的程序
  - 理解这些结构，可以帮助我们更有效地使用 Python
- 一般而言
  - 基于较低级的编程语言（例如 C）工作时，可以根据需要设计数据结构的表示（实现）方法。常规数据结构课程关注这方面问题
  - 基于 Python 等高级编程系统工作时，特别是做复杂工作时，也常需要自己设计一些数据结构，还需要对语言提供的各种结构的基本原理有很好理解，才能更有效地使用它们
  - 本学期主要考虑如何利用语言提供的数据构造机制，下学期另一门课程还会更多讨论数据组织的问题

内存和外存

- 计算机内存速度较快，常量时间访问，但成本较高。相对便宜而且存储密度高的实现DRAM技术具有易失性，一旦失电，全部信息将消失
- 为长期保存大量信息，人们开发了另一些技术，主要有基于电磁原理的磁盘/磁带，基于光电原理的光盘，基于半导体技术的闪存（U盘）等
- 与内存相比，外存储器的速度慢，不适合 CPU 直接访问。但单位容量的成本较低，存储数据具有持续性（无电存储）。由于这些特点
  - 外存通常用于保存大量需要长期存储的数据（包括可执行的程序）
  - 需要时，先将外存信息装入内存，而后CPU在内存使用它们
  - 程序执行中需要长期保存的数据，也需要转存到外存
- 外存存储空间由计算机的管理程序（操作系统）管理，保存在外存的数据按使用的逻辑关系组织为一个个命名文件
  - 正在运行的程序可以通过文件名访问外存文件
  - 下一节课，将讨论如何开发使用外存文件信息的 Python 程序

计算机的发展

- CPU 原理简单，而先进 CPU 极端复杂，是最复杂的产品。原因：
  - 需要计算机完成的计算日趋复杂，要执行更多指令才能完成，因此要求更高性能的 CPU。提高 CPU 速度需要更精深的技术和特殊的结构，这些都使 CPU 的复杂性大大增加
  - 多核 CPU 的多个执行部件需要通过复杂的机制（部件）相互协调
- 有些书籍说计算机经历了电子管/晶体管/集成电路/大规模集成电路 4 个阶段，称为第 1、2、3、4 代计算机。这种分类实际上已经毫无价值

  电子器件技术的这些变化不具根本性（虽很有意义：降低成本、减小体积等），是探索计算机制造方式和工艺的过程，历经二十余年
- 人们一直在研究真正的“新型”计算机，如量子计算机/生物计算机等
  - 与今天计算机本质上不同的信息处理工具会出现吗？能取代目前流行的电子计算机吗？目前还看不到明确的替代者
  - 看来还需要很长时间努力。可以拭目以待，也可以积极参与