计算理论

- 计算理论
  - 什么是计算
  - 计算理论的核心问题
  - Kolmogorov 复杂度

什么是计算

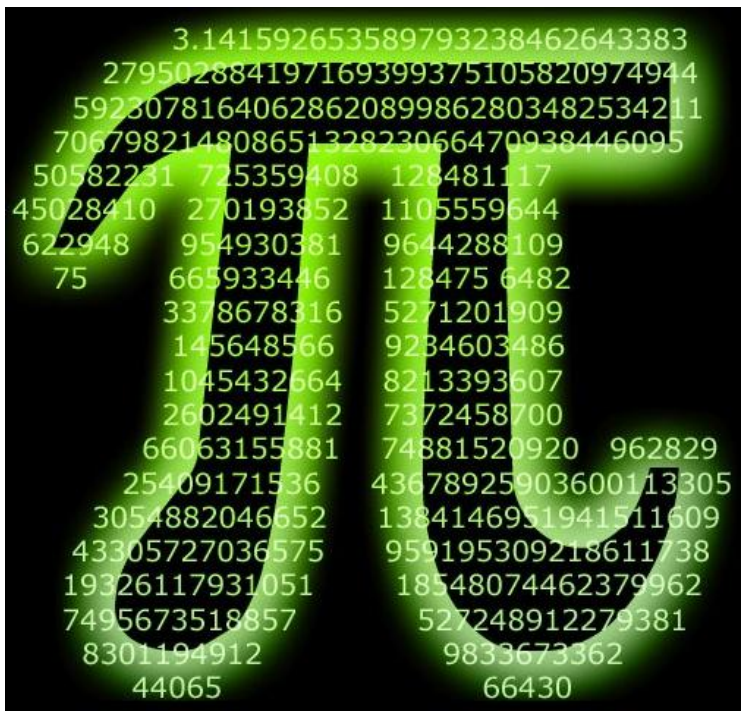
从计数到计算

- 实物计数–屈指计数–结绳计数–刻符计数–发明数字

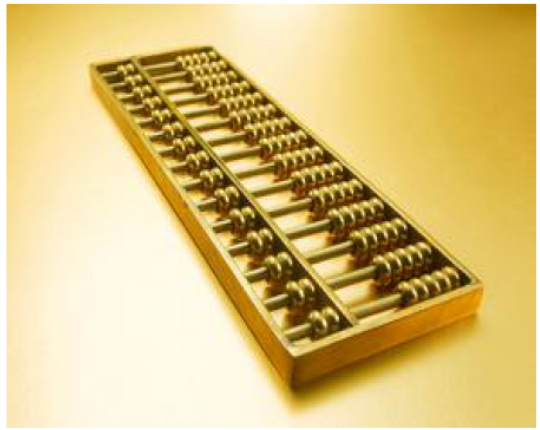
什么是计算

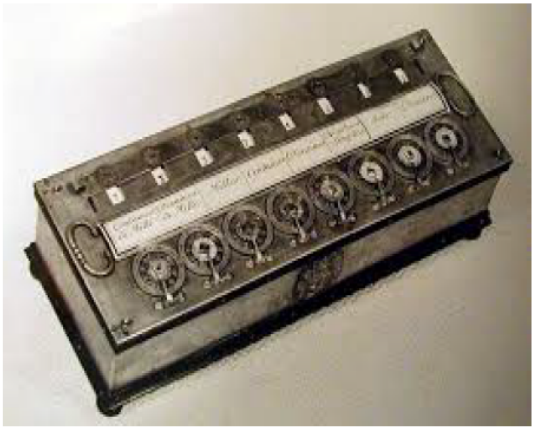
从计数到计算

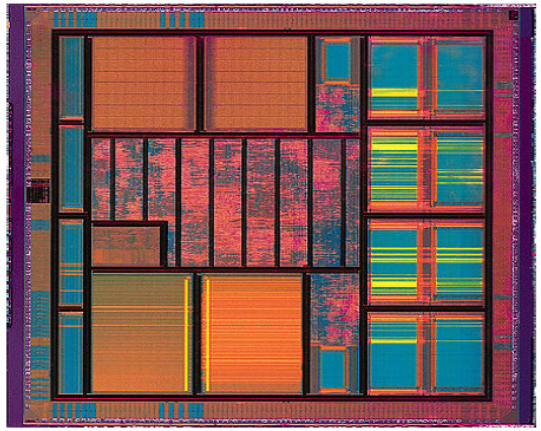
- 实物计数–屈指计数–结绳计数–刻符计数–发明数字
- 算筹–算术（整数运算）



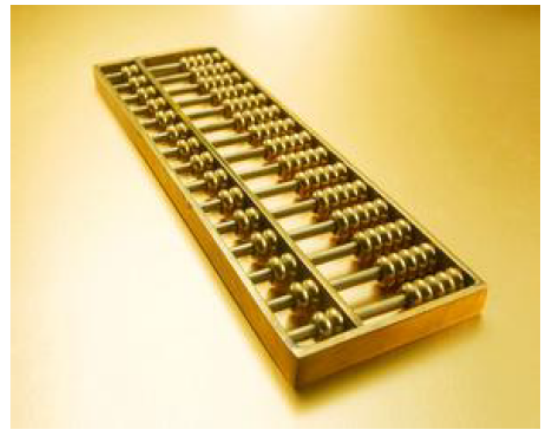
什么是计算

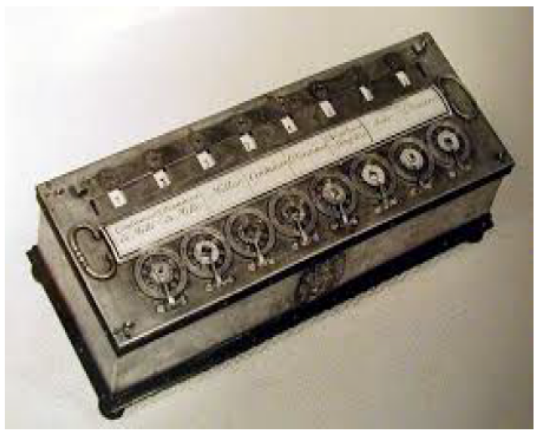


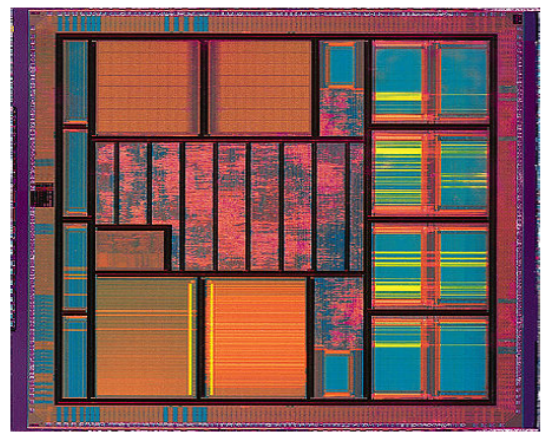




什么是计算







计算的最基本概念：对于一个给定输入，获得期望输出的过程

什么是计算

从逻辑到计算

- 古希腊哲学家和数学家发展逻辑学和逻辑演绎方法

什么是计算

从逻辑到计算

- 古希腊哲学家和数学家发展逻辑学和逻辑演绎方法
- 十九世纪数理逻辑问世将逻辑与计算联系起来

什么是计算

从逻辑到计算

- 古希腊哲学家和数学家发展逻辑学和逻辑演绎方法
- 十九世纪数理逻辑问世将逻辑与计算联系起来
  - 通过计算进行逻辑演绎

什么是计算

从逻辑到计算

- 古希腊哲学家和数学家发展逻辑学和逻辑演绎方法
- 十九世纪数理逻辑问世将逻辑与计算联系起来
  - 通过计算进行逻辑演绎
  - 通过逻辑推理实现计算-符号运算

什么是计算

从逻辑到计算

- 古希腊哲学家和数学家发展逻辑学和逻辑演绎方法
- 十九世纪数理逻辑问世将逻辑与计算联系起来
  - 通过计算进行逻辑演绎
  - 通过逻辑推理实现计算-符号运算

为什么要研究数理逻辑呢？

什么是计算

从逻辑到计算

- 古希腊哲学家和数学家发展逻辑学和逻辑演绎方法
- 十九世纪数理逻辑问世将逻辑与计算联系起来
  - 通过计算进行逻辑演绎
  - 通过逻辑推理实现计算-符号运算

为什么要研究数理逻辑呢？

- 程序＝算法＋数据结构

什么是计算

从逻辑到计算

- 古希腊哲学家和数学家发展逻辑学和逻辑演绎方法
- 十九世纪数理逻辑问世将逻辑与计算联系起来
  - 通过计算进行逻辑演绎
  - 通过逻辑推理实现计算-符号运算

为什么要研究数理逻辑呢？

- 程序＝算法＋数据结构
- 算法＝逻辑＋控制

什么是计算

算法与计算

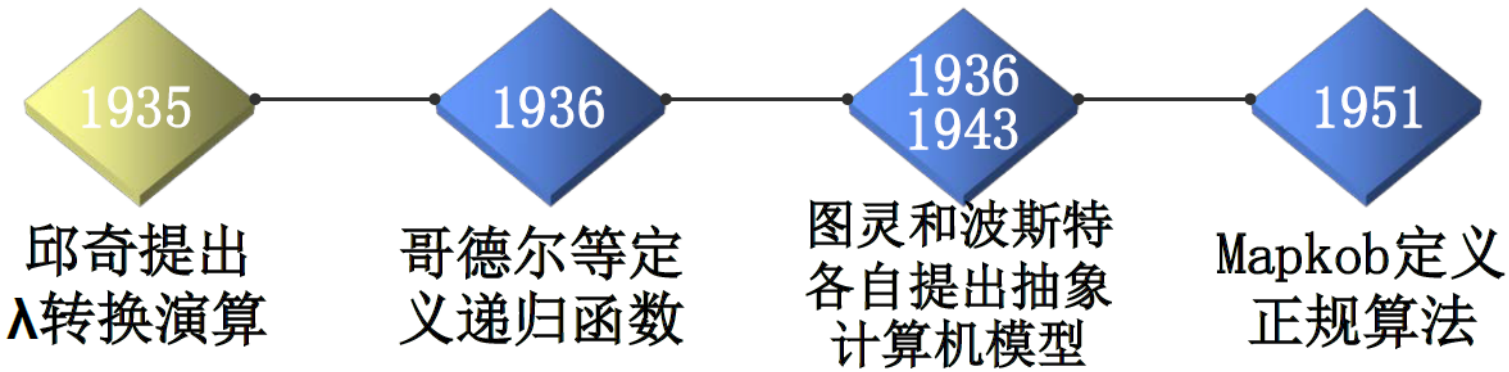
- 算法是接收输入并产生输出的计算过程；算法就是计算的办法

什么是计算

算法与计算

- 算法是接收输入并产生输出的计算过程；算法就是计算的办法

算法概念精确化定义:

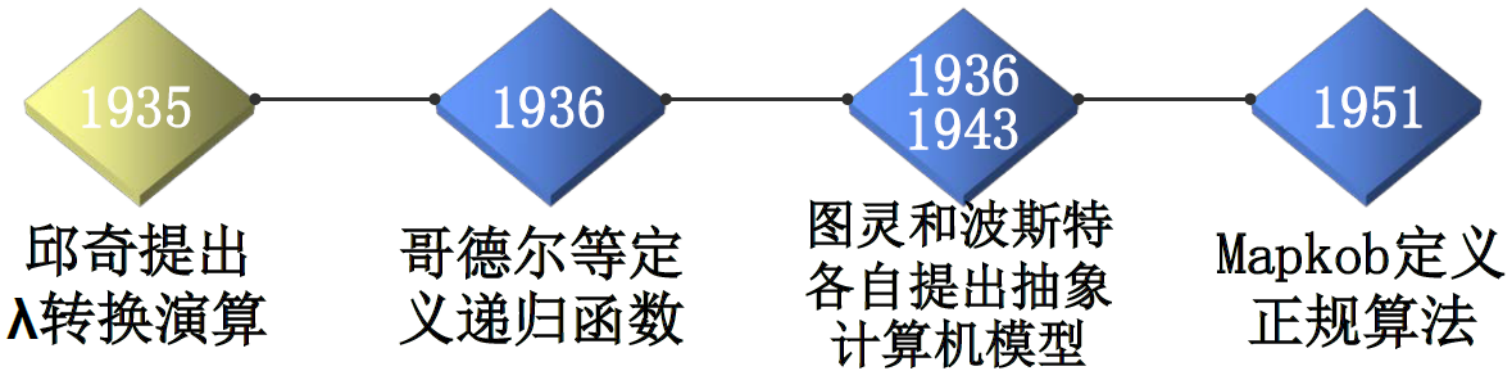


什么是计算

算法与计算

- 算法是接收输入并产生输出的计算过程；算法就是计算的办法

算法概念精确化定义:



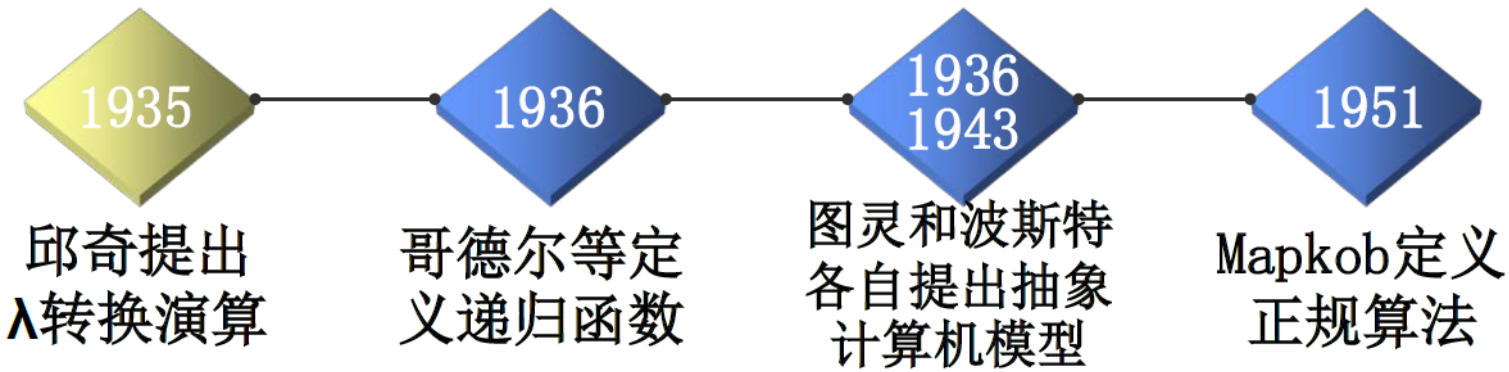
性质:

什么是计算

算法与计算

- 算法是接收输入并产生输出的计算过程；算法就是计算的办法

算法概念精确化定义:



性质:

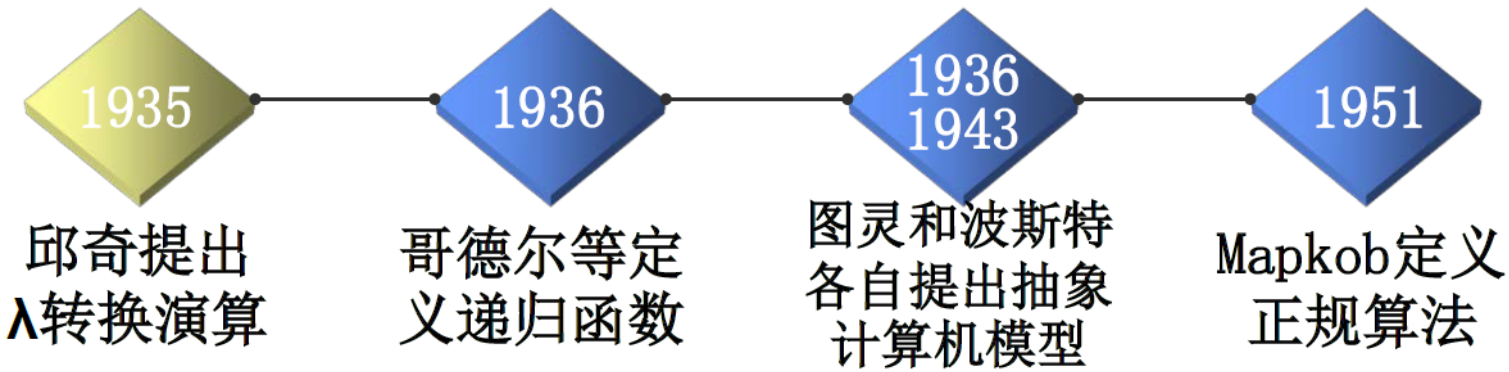
- 可行性-有明确的有限的求解步骤

什么是计算

算法与计算

- 算法是接收输入并产生输出的计算过程；算法就是计算的办法

算法概念精确化定义:



性质:

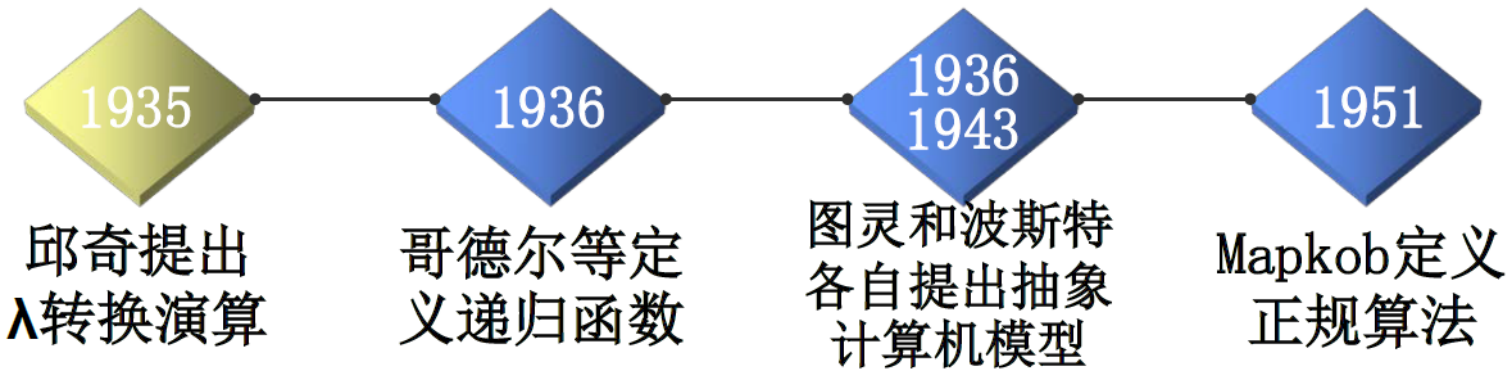
- 可行性-有明确的有限的求解步骤
- 有穷性-对某些输入在有限步内结束，并给出结果

什么是计算

算法与计算

- 算法是接收输入并产生输出的计算过程；算法就是计算的办法

算法概念精确化定义:



性质:

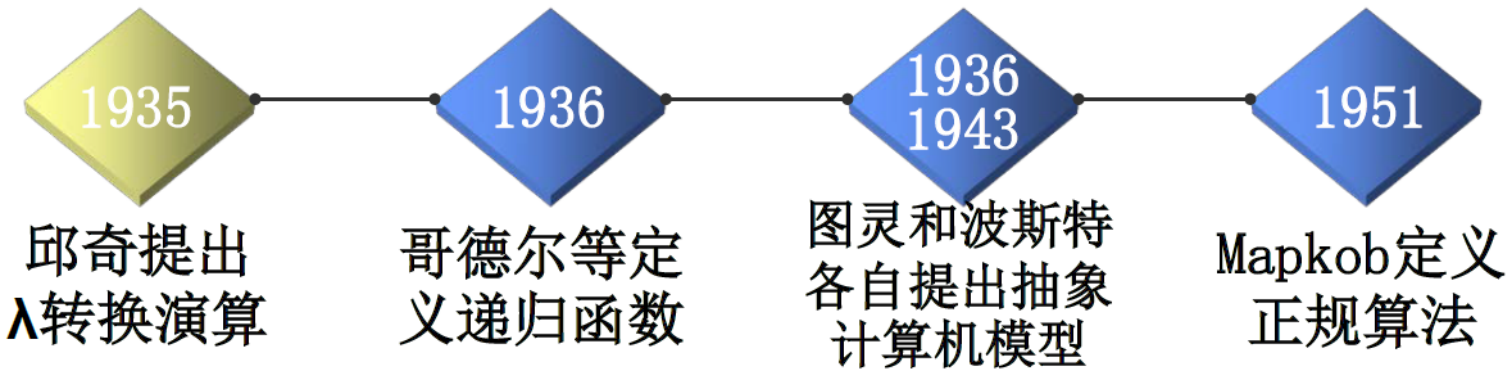
- 可行性-有明确的有限的求解步骤
- 有穷性-对某些输入在有限步内结束，并给出结果
- 通用性-对于某类问题的求解，总是给出正确的答案

什么是计算

算法与计算

- 算法是接收输入并产生输出的计算过程；算法就是计算的办法

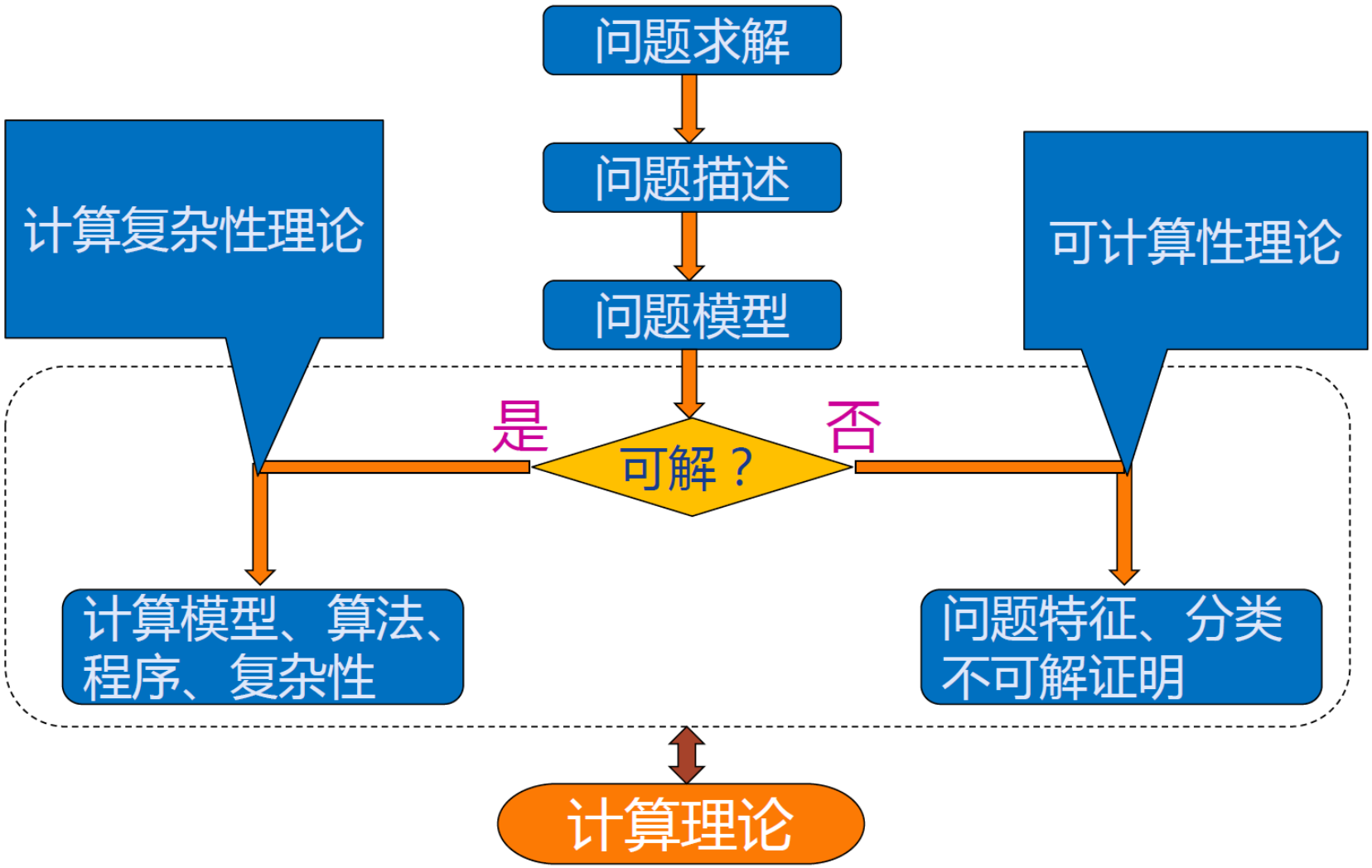
算法概念精确化定义:



性质:

- 可行性-有明确的有限的求解步骤
- 有穷性-对某些输入在有限步内结束，并给出结果
- 通用性-对于某类问题的求解，总是给出正确的答案
- 确定性-每个步骤都是机械的、明确的，无歧义

什么是计算理论


问题求解
问题描述
问题模型
计算复杂性理论
可计算性理论
是
可解？
否
计算模型、算法、程序、复杂性
问题特征、分类不可解证明
计算理论

计算理论的核心问题

- **计算理论**：关于计算和计算机械的数学理论，它研究计算的过程与功效。

计算理论的核心问题

- **计算理论**：关于计算和计算机械的数学理论，它研究计算的过程与功效。
- **核心问题**:
  - 自动机理论：计算模型和计算能力

计算理论的核心问题

- **计算理论**：关于计算和计算机械的数学理论，它研究计算的过程与功效。
- **核心问题**:
  - 自动机理论：计算模型和计算能力
  - 可计算性理论：问题是否可解

计算理论的核心问题

- **计算理论**：关于计算和计算机械的数学理论，它研究计算的过程与功效。
- **核心问题**:
  - 自动机理论：计算模型和计算能力
  - 可计算性理论：问题是否可解
  - 计算复杂性理论：问题是否难解

图灵机

- **计算模型**：刻画计算的抽象的形式系统或数学系统。在计算科学中，计算模型是指**具有状态转换特征，能对所处理对象的数据或信息进行表示、加工、变换和输出的数学机器**。

图灵机

- **计算模型**：刻画计算的抽象的形式系统或数学系统。在计算科学中，计算模型是指**具有状态转换特征，能对所处理对象的数据或信息进行表示、加工、变换和输出的数学机器**。

- **图灵机**：
  - 1936 年，图灵在其“论可计算数及其在判定问题中的应用”论文中提出著名的图灵机(Turing Machine) 的设想。

图灵机

- **计算模型**：刻画计算的抽象的形式系统或数学系统。在计算科学中，计算模型是指**具有状态转换特征，能对所处理对象的数据或信息进行表示、加工、变换和输出的数学机器**。

- **图灵机**：
  - 1936 年，图灵在其“论可计算数及其在判定问题中的应用”论文中提出著名的图灵机(Turing Machine) 的设想。
  - **现代计算机的理论模型**：
    - 相同之处：程序与数据混合在一起，由控制器控制执行。

图灵机

- **计算模型**：刻画计算的抽象的形式系统或数学系统。在计算科学中，计算模型是指**具有状态转换特征，能对所处理对象的数据或信息进行表示、加工、变换和输出的数学机器**。

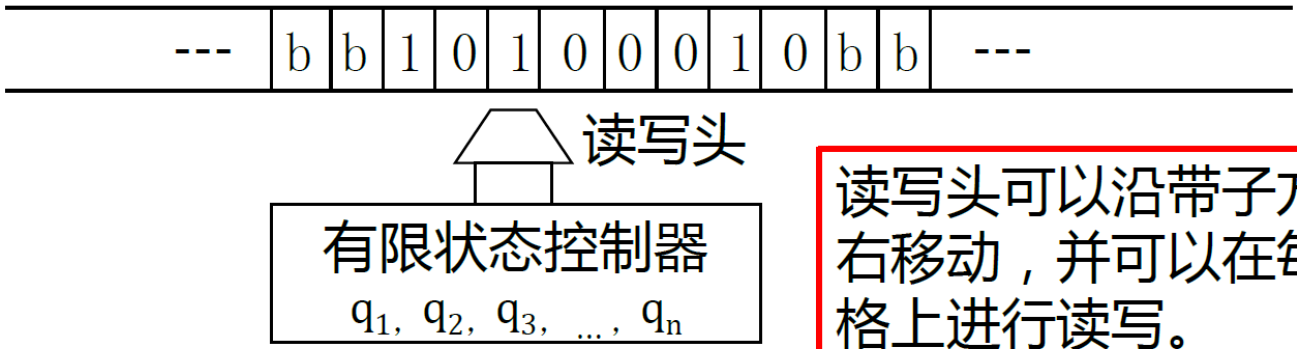
- **图灵机**：
  - 1936 年，图灵在其“论可计算数及其在判定问题中的应用”论文中提出著名的图灵机(Turing Machine) 的设想。
  - **现代计算机的理论模型**：
    - 相同之处：程序与数据混合在一起，由控制器控制执行。
    - 不同之处：内存无限大！没有考虑输入与输出！（所有信息都在子带上）

图灵机—直观描述

- **图灵机的计算装置**：一条两端可无限延长的带子，一个读写头，一个有限状态控制器。

图灵机—直观描述

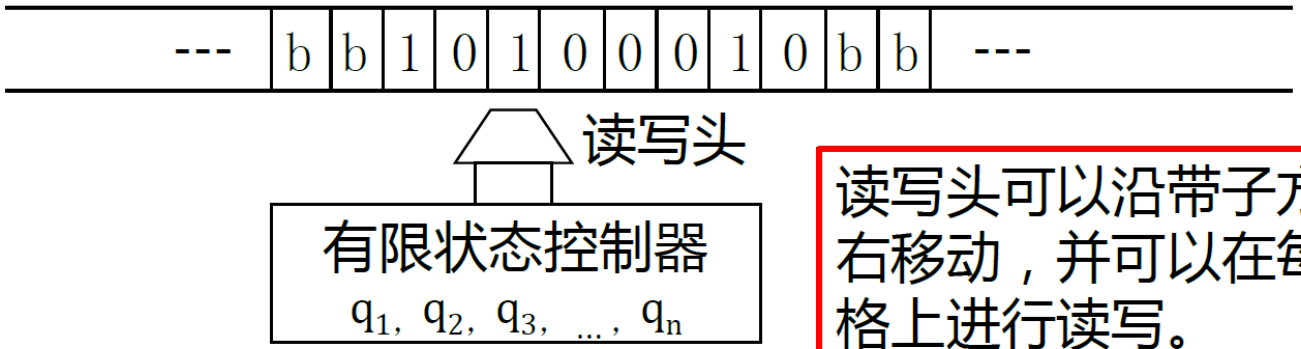
- **图灵机的计算装置**：一条两端可无限延长的带子，一个读写头，一个有限状态控制器。



读写头可以沿带子方向左右移动，并可以在每个方格上进行读写。

图灵机—直观描述

- **图灵机的计算装置**：一条两端可无限延长的带子，一个读写头，一个有限状态控制器。



- **有穷字母表**：

$$\{S_0, S_1, S_2, \cdots, S_p\}$$

通常仅有 S_0、S_1 两个字符，其中：

$$S_0 \to 0, S_1 \to 1$$

图灵机—直观描述

- **机器的控制状态**：

$$\{q_1, q_2, q_3, \cdots, q_n\}$$

图灵机的初始状态设为 q_1，结束状态设为 q_n

图灵机—直观描述

- **机器的控制状态**:

$$\{q_1, q_2, q_3, \cdots, q_n\}$$

图灵机的初始状态设为 q_1，结束状态设为 q_n

- **五元组指令集合**:

$$(q_i, S_j, S_k, R/L/N, q_n)$$

  - q_i – 机器目前所处的状态

图灵机—直观描述

- **机器的控制状态**:

$$\{q_1, q_2, q_3, \cdots, q_n\}$$

图灵机的初始状态设为 q_1，结束状态设为 q_n

- **五元组指令集合**:

$$(q_i, S_j, S_k, R/L/N, q_n)$$

  - q_i – 机器目前所处的状态
  - S_j – 机器从方格中读入的符号

图灵机—直观描述

- **机器的控制状态**:

$$\{q_1, q_2, q_3, \cdots, q_n\}$$

图灵机的初始状态设为 q_1，结束状态设为 q_n

- **五元组指令集合**:

$$(q_i, S_j, S_k, R/L/N, q_n)$$

  - q_i – 机器目前所处的状态
  - S_j – 机器从方格中读入的符号
  - S_k – 机器用来代替 S_j 写入方格的符号

图灵机—直观描述

- **机器的控制状态**:

$$\{q_1, q_2, q_3, \cdots, q_n\}$$

图灵机的初始状态设为 q_1, 结束状态设为 q_n

- **五元组指令集合**:

$$(q_i, S_j, S_k, R/L/N, q_n)$$

  - q_i – 机器目前所处的状态
  - S_j – 机器从方格中读入的符号
  - S_k – 机器用来代替 S_j 写入方格的符号
  - R, L, N – 右移一格, 左移一格, 不移动

图灵机—直观描述

- **机器的控制状态**:

$$\{q_1, q_2, q_3, \cdots, q_n\}$$

图灵机的初始状态设为 q_1, 结束状态设为 q_n

- **五元组指令集合**:

$$(q_i, S_j, S_k, R/L/N, q_n)$$

  - q_i – 机器目前所处的状态
  - S_j – 机器从方格中读入的符号
  - S_k – 机器用来代替 S_j 写入方格的符号
  - R, L, N – 右移一格, 左移一格, 不移动
  - q_n – 下一步机器的状态

图灵机——工作原理

工作原理：

图灵机——工作原理

工作原理:

- 机器从给定带子上的某起点（初始状态）出发

图灵机——工作原理

工作原理:

- 机器从给定带子上的某起点（初始状态）出发
- 其动作完全由其机内五元组指令集来决定

图灵机——工作原理

工作原理:

- 机器从给定带子上的某起点（初始状态）出发
- 其动作完全由其机内五元组指令集来决定
- 计算结果是从机器停止时带子上的信息得到

图灵机——工作原理

工作原理:

- 机器从给定带子上的某起点（初始状态）出发
- 其动作完全由其机内五元组指令集来决定
- 计算结果是从机器停止时带子上的信息得到

指令的设计问题:

图灵机——工作原理

工作原理:

- 机器从给定带子上的某起点（初始状态）出发
- 其动作完全由其机内五元组指令集来决定
- 计算结果是从机器停止时带子上的信息得到

指令的设计问题:

- 指令死循环：$q_1 S_2 S_2 R q_3; q_3 S_3 S_3 L q_1$

图灵机——工作原理

工作原理:

- 机器从给定带子上的某起点（初始状态）出发
- 其动作完全由其机内五元组指令集来决定
- 计算结果是从机器停止时带子上的信息得到

指令的设计问题:

- 指令死循环：$q_1 S_2 S_2 R q_3; q_3 S_3 S_3 L q_1$
- 指令二义性：$q_3 S_2 S_2 R q_4; q_3 S_2 S_4 L q_6$

图灵机——应用实例

- [例] 假设：
  - b：表示空格

图灵机——应用实例

- [例] 假设：
  - b：表示空格
  - q_1：表示机器的初始状态

图灵机——应用实例

- [例] 假设：
  - b：表示空格
  - q_1：表示机器的初始状态
  - q_4：表示机器的结束状态

图灵机——应用实例

- [例] **假设**:
  - b：表示空格
  - q_1：表示机器的初始状态
  - q_4：表示机器的结束状态

- **条件**：如果带子上的**输入信息**：10100010，读写头位对准最右边第一个为 0 的方格，且状态为 q_1。

图灵机——应用实例

- [例] **假设**:
  - b：表示空格
  - q_1：表示机器的初始状态
  - q_4：表示机器的结束状态

- **条件**：如果带子上的**输入信息**：10100010，读写头位对准最右边第一个为 0 的方格，且状态为 q_1。

- **问**：按照以下**五元组指令集**执行后，输出正确的计算结果是什么？

$q_1 01 \mathrm{L} q_2$；$q_1 10 \mathrm{L} q_3$；$q_1 \mathrm{bbN} q_4$；

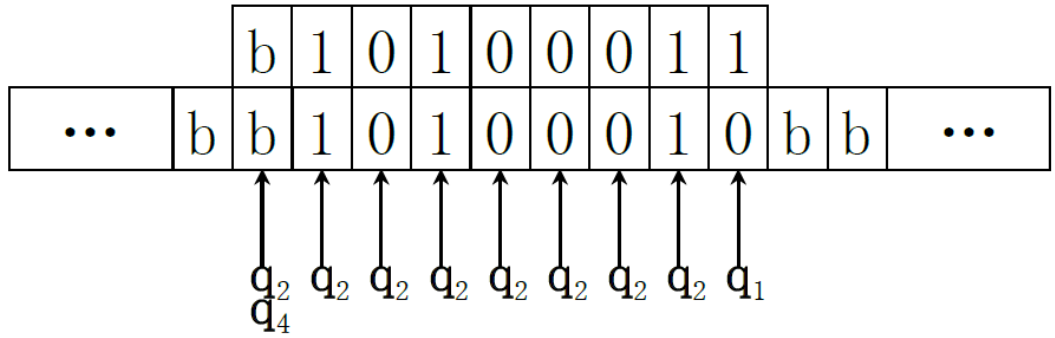
$q_2 00 \mathrm{L} q_2$；$q_2 11 \mathrm{L} q_2$；$q_2 bbN q_4$；

$q_3 01 \mathrm{L} q_2$；$q_3 10 \mathrm{L} q_3$；$q_3 bbN q_4$

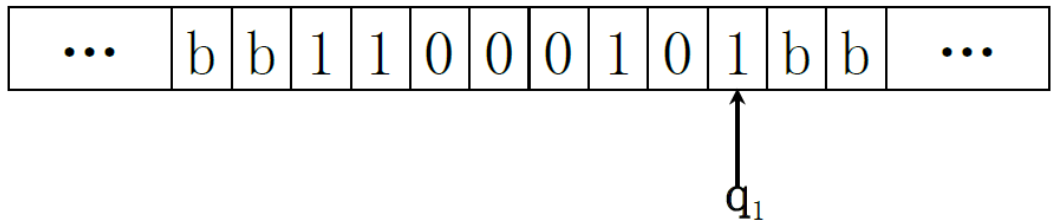
图灵机——应用实例

指令集

$q_1 01 L q_2$
$q_1 10 L q_3$
$q_1 bbN q_4$
$q_2 00 L q_2$
$q_2 11 L q_2$
$q_2 bbN q_4$
$q_3 01 L q_2$
$q_3 10 L q_3$
$q_3 bbN q_4$



计算函数是：$S(x) = x + 1$



图灵机——应用实例

[例] 图灵机 Mz：其中 $Q = q_1, q_2, q_n$

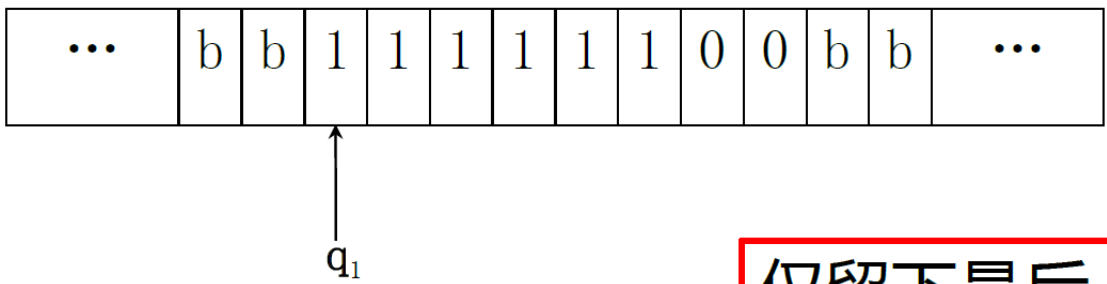
五元组指令集为：

$$q_1 10 R q_1$$

$$q_1 00 L q_2$$

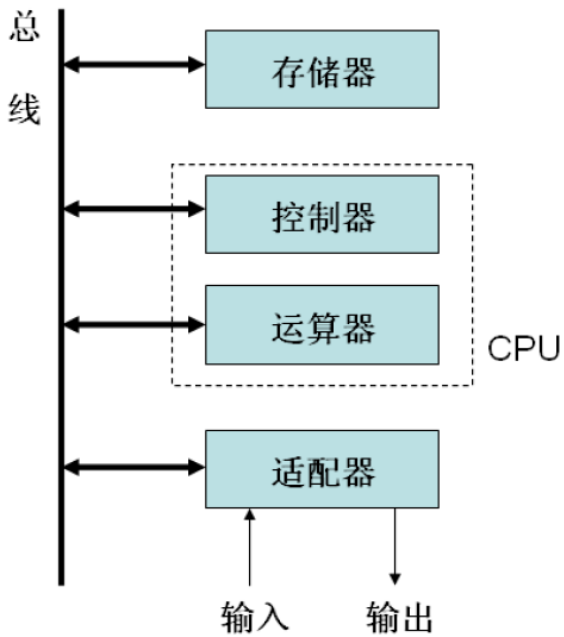
$$q_2 01 N q_n$$

求 Mz 对任何一串“1”的作用是什么？



仅留下最后一个“1”

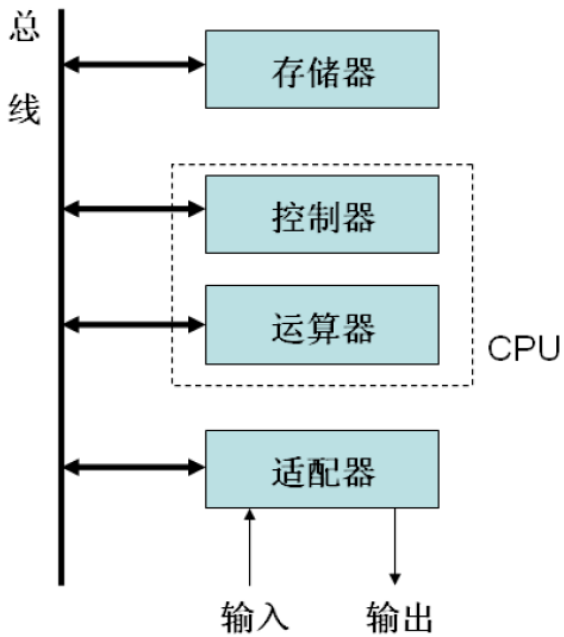
冯·诺依曼结构计算机



冯·诺依曼结构

- 计算机由控制器、运算器、存储器、输入设备和输出设备组成。

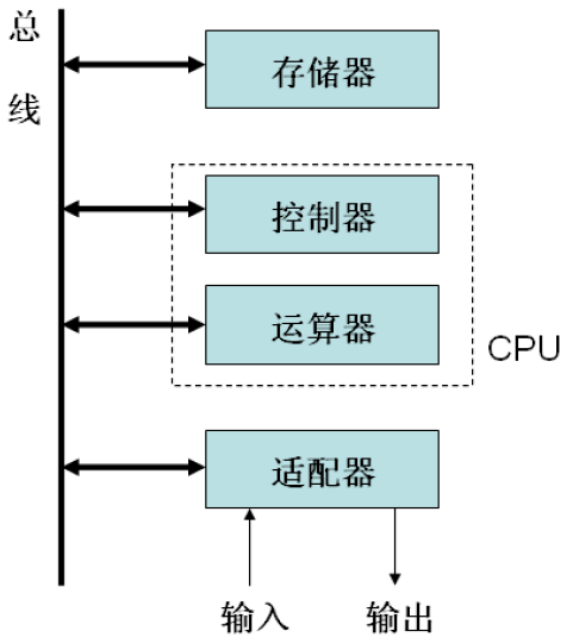
冯 · 诺依曼结构计算机



冯 · 诺依曼结构

- 计算机由控制器、运算器、存储器、输入设备和输出设备组成。
- **基本原理**: 存储程序（Stored Program）并按地址顺序执行。

冯 · 诺依曼结构计算机

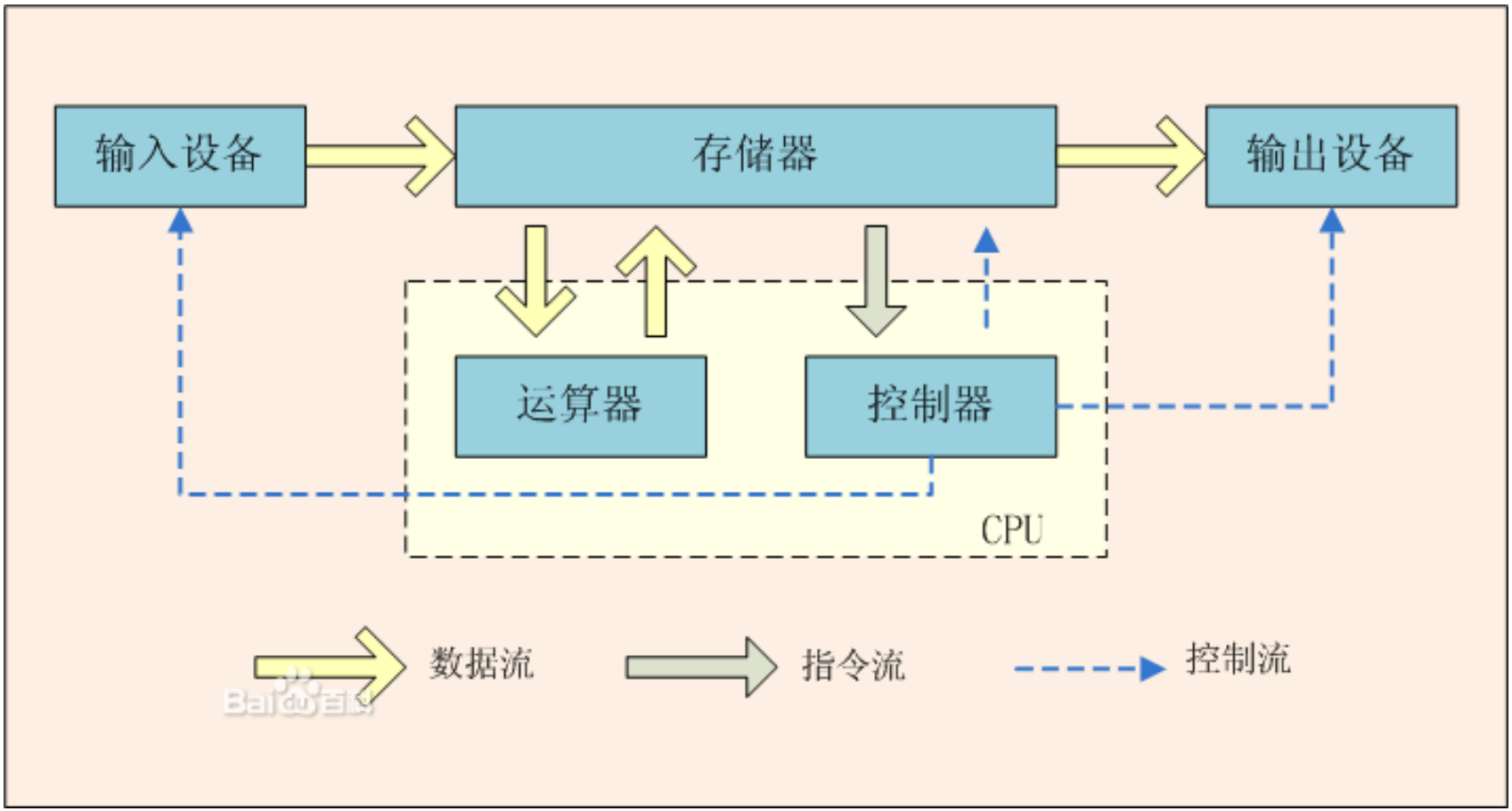


冯 · 诺依曼结构

- 计算机由控制器、运算器、存储器、输入设备和输出设备组成。
- **基本原理**：存储程序（Stored Program）并按地址顺序执行。
- 控制器按照程序顺序，逐条把指令和数据从存储器中取出并加以执行，自动完成由程序所描述的处理工作。

冯 · 诺依曼结构计算机

工作原理：先将程序 (一组指令) 和数据存入计算机，启动程序就能按照程序指定的逻辑顺序把指令读取并逐条执行，自动完成指令规定的操作。



冯 · 诺依曼结构计算机

冯 · 诺依曼机的特点

- 以运算器为中心，输入输出设备与存储器之间的数据传送都要过运算器

冯·诺依曼结构计算机

冯·诺依曼机的特点

- 以运算器为中心，输入输出设备与存储器之间的数据传送都要过运算器
- 采用存储程序原理

冯·诺依曼结构计算机

冯·诺依曼机的特点

- 以运算器为中心，输入输出设备与存储器之间的数据传送都要过运算器
- 采用存储程序原理
- 指令是由操作码和地址码组成

冯·诺依曼结构计算机

冯·诺依曼机的特点

- 以运算器为中心，输入输出设备与存储器之间的数据传送都要过运算器
- 采用存储程序原理
- 指令是由操作码和地址码组成
- 数据以二进制表示，并采用二进制运算

冯·诺依曼结构计算机

冯·诺依曼机的特点

- 以运算器为中心，输入输出设备与存储器之间的数据传送都要过运算器
- 采用存储程序原理
- 指令是由操作码和地址码组成
- 数据以二进制表示，并采用二进制运算
- 硬件与软件完全分开，硬件在结构和功能上是不变的，完全靠编制软件来适应用户需要

冯·诺依曼结构计算机

冯·诺依曼机的特点

- 以运算器为中心，输入输出设备与存储器之间的数据传送都要过运算器
- 采用存储程序原理
- 指令是由操作码和地址码组成
- 数据以二进制表示，并采用二进制运算
- 硬件与软件完全分开，硬件在结构和功能上是不变的，完全靠编制软件来适应用户需要

冯·诺依曼机结构的局限性

冯·诺依曼结构计算机

冯·诺依曼机的特点

- 以运算器为中心，输入输出设备与存储器之间的数据传送都要过运算器
- 采用存储程序原理
- 指令是由操作码和地址码组成
- 数据以二进制表示，并采用二进制运算
- 硬件与软件完全分开，硬件在结构和功能上是不变的，完全靠编制软件来适应用户需要

冯·诺依曼机结构的局限性

- **瓶颈**：存储器与中央处理单元之间的通路太狭窄，每次执行一条指令，所需的指令和数据都必须经过这条通路。

可计算性

- 机器是否可计算？
- 如果可以，代价是多少？

丢番图方程–维基百科

丢番图方程 [编辑]

文A 52种语言

条目 讨论 汉 漢 大陆简体 阅读 编辑 查看历史 工具

> 此条目已列出参考文献，但因为没有文内引注而使来源仍然不明。 *(2019年12月12日)*
> 请加上合适的文内引注来改善这篇条目。

丢番图方程（英语：Diophantine equation），又称**不定方程**，是未知数只能使用整数的整数系数多项式等式；即形式如$a_1 x_1^{b_1} + a_2 x_2^{b_2} + \ldots\ldots + a_n x_n^{b_n} = c$ 的等式，并且其中所有的a_j、b_j和c均是整数。若其中能找到一组整数解$m_1, m_2 \ldots m_n$者则称之有整数解。

丢番图问题一般可以有数条等式，其数目比未知数的数目少；丢番图问题要求找出对所有等式都成立的整数组合。换言之，丢番图问题定义了代数曲线或者代数曲面，或更为一般的几何形，要求找出其中的栅格点。对丢番图问题的数学研究称为**丢番图分析**。线性丢番图方程为线性整数系数多项式等式，即此多项式为次数为0或1的单项式的和。

丢番图方程的名字来源于3世纪希腊数学家亚历山大城的丢番图，他曾对这些方程进行研究，并且是第一个将符号引入代数的数学家。

关于丢番图方程的理论的形成和发展是二十世纪数学一个很重要的发展。丢番图方程的例子有裴蜀等式、勾股定理的整数解、佩尔方程、四平方和定理和费马最后定理等。

可计算性理论

- 希尔伯特第 10 问题：判定丢番图方程的可解性

可计算性理论

- 希尔伯特第 10 问题：**判定丢番图方程的可解性**
  - 给定一个系数均为有理整数，包含任意个未知数的丢番图方程：设计一个过程，通过有限次的计算，能够判定该方程在有理数整数上是否可解。

可计算性理论

- 希尔伯特第 10 问题：**判定丢番图方程的可解性**
  - 给定一个系数均为有理整数，包含任意个未知数的丢番图方程：设计一个过程，通过有限次的计算，能够判定该方程在有理数整数上是否可解。
  - **马蒂雅谢维奇定理**：不可能存在一个演算法能够判定任意丢番图方程是否有解；甚至，在任何相容于皮亚诺算数的系统当中，都能具体构造出一个丢番图方程，使得没有任何办法可以判断它是否有解。

可计算性理论

- 希尔伯特第 10 问题：**判定丢番图方程的可解性**
  - 给定一个系数均为有理整数，包含任意个未知数的丢番图方程：设计一个过程，通过有限次的计算，能够判定该方程在有理数整数上是否可解。
  - **马蒂雅谢维奇定理**：不可能存在一个演算法能够判定任意丢番图方程是否有解；甚至，在任何相容于皮亚诺算数的系统当中，都能具体构造出一个丢番图方程，使得没有任何办法可以判断它是否有解。

- **丘奇-图灵论题**：任何可由算法有效演算的问题同样可由图灵机有效计算。

  In simple terms, the Church–Turing thesis states that a function on the natural numbers is computable in an informal sense (i.e., computable by a human being using a pencil-and-paper method, ignoring resource limitations) if and only if it is computable by a Turing ma-

可计算性理论

停机问题：重要的不可判定问题

可计算性理论

停机问题：重要的不可判定问题

> 1936 年，Turing 发表“论可计算数及其在判定问题中的应用”论文中提出一般性停机问题的不可判定性，并用形式化方法证明其为一个不可计算问题。

可计算性理论

停机问题：重要的不可判定问题

1936 年，Turing 发表“论可计算数及其在判定问题中的应用”论文中提出一般性停机问题的不可判定性，并用形式化方法证明其为一个不可计算问题。

一般性的停机问题：对于任意的图灵机和输入，是否存在一个算法，用于判定图灵机在接收初始输入后可达停机状态。若能找到这种算法，停机问题可解；否则不可解。

可计算性理论

通俗地说，停机问题就是判断任意一个程序是否在有限的时间内结束运行的问题。

可计算性理论

通俗地说，停机问题就是判断任意一个程序是否在有限的时间内结束运行的问题。

例如：
```
main()
{  int i=1;
   while ( i<10 )
   {  i=i+1;
   }
   return;
}
```

又如：
```
main()
{  int i=1;
   while ( i>0 )
   {  i=i+1;
   }
   return;
}
```

可计算性理论

通俗地说，停机问题就是判断任意一个程序是否在有限的时间内结束运行的问题。

例如：
```
main()
{  int i=1;
   while ( i<10 )
   {  i=i+1;
   }
   return;
}
```

又如：
```
main()
{  int i=1;
   while ( i>0 )
   {  i=i+1;
   }
   return;
}
```

程序简单时容易做出判断，当例子复杂时会遇到较大的困难，而在某些情况下则无法预测。

可计算性理论

- **停机问题的关键**：能否找到一个测试程序，这个测试程序能判定任何一个程序在给定的输入下能否终止

可计算性理论

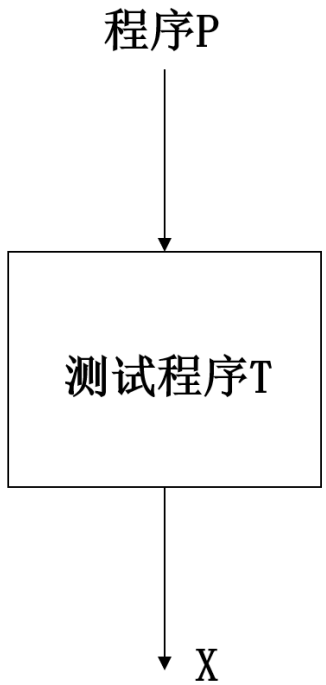
- **停机问题的关键**：能否找到一个测试程序，这个测试程序能判定任何一个程序在给定的输入下能否终止
- **数学反证法证明**：先假设存在这样的测试程序，然后再构造一个程序，该测试程序不能测试

可计算性理论

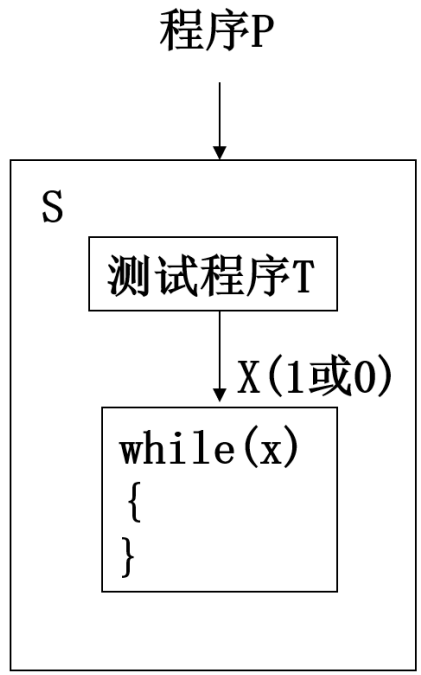
- **停机问题的关键**：能否找到一个测试程序，这个测试程序能判定任何一个程序在给定的输入下能否终止
- **数学反证法证明**：先假设存在这样的测试程序，然后再构造一个程序，该测试程序不能测试

> - 假设存在一个测试程序 T，它能接受任何输入
> - 当输入程序 P 能终止，输出 1；P 不能终止，输出 0

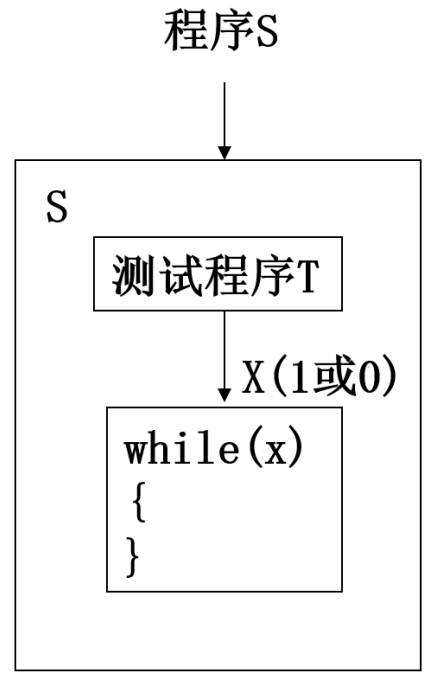
可计算性理论



P能终止, X→1
P不终止, X→0

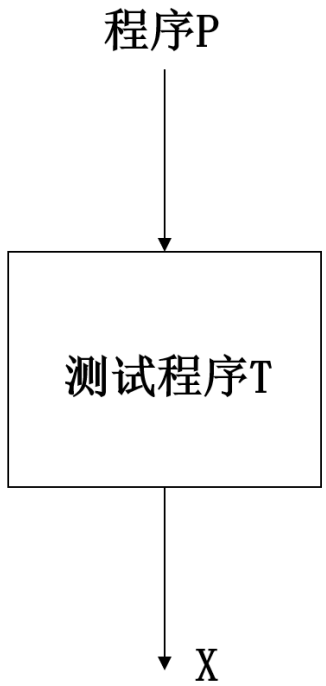


P终止, X→1, S→不终止
P不终止, X→0, S→终止

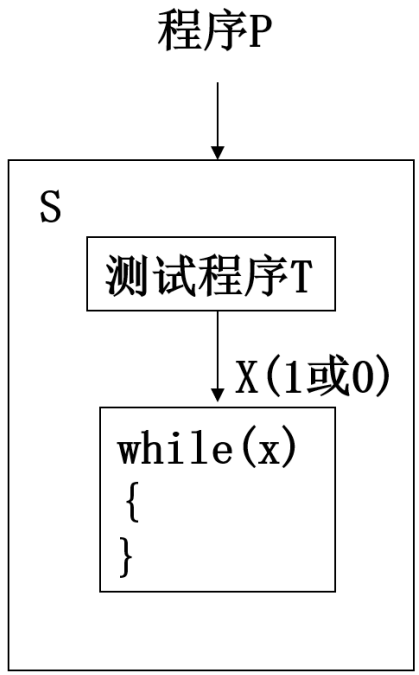


S终止, X→1, S→不终止
S不终止, X→0, S→终止

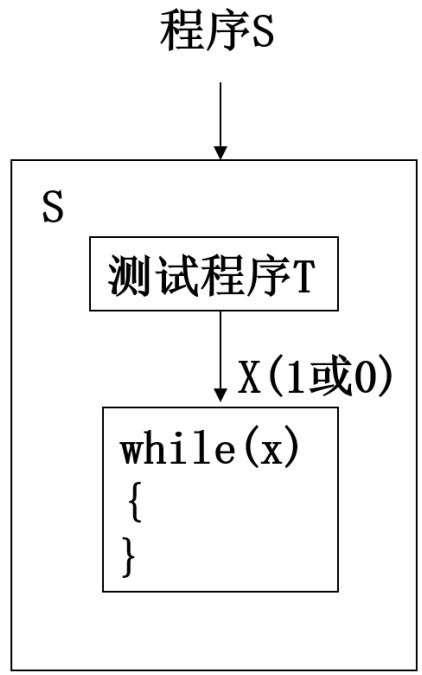
可计算性理论



P能终止, X→1
P不终止, X→0



P终止, X→1, S→不终止
P不终止, X→0, S→终止



S终止, X→1, S→不终止
S不终止, X→0, S→终止

结论：若 S 终止，则 S 不终止；若 S 不终止，则 S 终止，结论矛盾故可以确定这样的测试程序不存在，从而证明**停机问题不可解**

可计算性理论

- **理发师悖论**：一个理发师的招牌“城里所有**不自己刮脸的男人都由我给他们刮脸，我也只给这些人刮脸**。”

可计算性理论

- **理发师悖论**：一个理发师的招牌“城里所有**不自己刮脸的男人都由我给他们刮脸，我也只给这些人刮脸**。”
- **问题是**：
  - 谁给这位理发师刮脸呢？如果他自己刮脸，那他就属于自己刮脸的那类人。但是，他说他不给这类人刮脸，因此他不能自己来刮。

可计算性理论

- **理发师悖论**：一个理发师的招牌“城里所有**不自己刮脸的男人都由我给他们刮脸，我也只给这些人刮脸**。”
- **问题是**：
  - 谁给这位理发师刮脸呢？如果他自己刮脸，那他就属于自己刮脸的那类人。但是，他说他不给这类人刮脸，因此他不能自己来刮。
  - 如果另外一个人来给他刮脸，那他就是不自己刮脸的人。但是，他说他要给所有这类人刮脸。因此，其他任何人也不能给他刮脸。

计算复杂性理论

- 例子 1：**排序问题**

  升序排列一张具有 100 万个数的数字表

计算复杂性理论

- 例子 1：**排序问题**

  升序排列一张具有 100 万个数的数字表

- 例子 2：**时间表问题**

  制定某大学的课程表，课程表必须满足某些合理的限制（比如不能有两个班在同一时间使用同一教室）。

计算复杂性理论

- 例子 1：**排序问题**

  升序排列一张具有 100 万个数的数字表

- 例子 2：**时间表问题**

  制定某大学的课程表，课程表必须满足某些合理的限制（比如不能有两个班在同一时间使用同一教室）。

复杂性理论核心问题：是什么使得某些问题很难计算，而又使另一些问题容易计算呢？

计算复杂性理论

计算复杂性理论：用数学方法研究各类问题的计算复杂性的学科

计算复杂性理论

计算复杂性理论：用数学方法研究各类问题的计算复杂性的学科

- 研究各种可计算问题在计算过程中资源 (如时间、空间等) 的耗费情况

计算复杂性理论

计算复杂性理论：用数学方法研究各类问题的计算复杂性的学科

- 研究各种可计算问题在计算过程中资源 (如时间、空间等) 的耗费情况
- 各类问题复杂性的本质特性和相互关系

计算复杂性理论

计算复杂性理论：用数学方法研究各类问题的计算复杂性的学科

- 研究各种可计算问题在计算过程中资源 (如时间、空间等) 的耗费情况
- 各类问题复杂性的本质特性和相互关系

计算复杂性理论的主要创始人

- Rabin、Cook 和 Karp: 复杂性理论研究的先驱
- Hartmanis 和 Stearns: 提出计算复杂性的理论体系，并首次正式命名了计算复杂性
- Blum: 布卢姆公理系统 (为复杂度类提供通用定义，且不依赖具体的机器模型，wiki/chatgpt)

计算复杂性

计算复杂性：用计算机求解问题的难易程度

计算复杂性

计算复杂性：用计算机求解问题的难易程度

- 算法复杂性 → 针对特定算法

计算复杂性

计算复杂性：用计算机求解问题的难易程度

- 算法复杂性 → 针对特定算法
- 计算复杂性 → 针对特定问题，反映问题的固有难度

计算复杂性

计算复杂性：用计算机求解问题的难易程度

- 算法复杂性 → 针对特定算法
- 计算复杂性 → 针对特定问题，反映问题的固有难度
- 计算复杂性＝最佳的算法复杂性

计算复杂性

计算复杂性：用计算机求解问题的难易程度

- 算法复杂性 → 针对特定算法
- 计算复杂性 → 针对特定问题，反映问题的固有难度
- 计算复杂性＝最佳的算法复杂性

对比信息度量

计算复杂性

计算复杂性：用计算机求解问题的难易程度

- 算法复杂性 → 针对特定算法
- 计算复杂性 → 针对特定问题，反映问题的固有难度
- 计算复杂性＝最佳的算法复杂性

对比信息度量

度量标准：

- **时间复杂度** → 计算所需的步数或指令条数

计算复杂性

计算复杂性：用计算机求解问题的难易程度

- 算法复杂性 → 针对特定算法
- 计算复杂性 → 针对特定问题，反映问题的固有难度
- 计算复杂性＝最佳的算法复杂性

对比信息度量

度量标准：

- **时间复杂度** → 计算所需的步数或指令条数
- **空间复杂度** → 计算所需的存储空间大小

时间复杂度

- **定义**：令 M 是一个确定型图灵机。M 的运行时间（时间复杂度）是一个函数 $f(n) : N \to N$，其中 N 是非负整数集合，$f(n)$ 是 M 在所有长度为 n 的输入上运行时所经过的最大步数。

时间复杂度

- **定义**：令 M 是一个确定型图灵机。M 的运行时间（时间复杂度）是一个函数 $f(n): N \to N$，其中 N 是非负整数集合，$f(n)$ 是 M 在所有长度为 n 的输入上运行时所经过的最大步数。

- **复杂度的渐进记法（大 O 记法）**

  - $f(n) = 6n^3 + 4n^2 + 7 = O(n^3)$　例：LMMSE 检测

时间复杂度

- **定义**：令 M 是一个确定型图灵机。M 的运行时间（时间复杂度）是一个函数 $f(n): N \to N$，其中 N 是非负整数集合，$f(n)$ 是 M 在所有长度为 n 的输入上运行时所经过的最大步数。

- **复杂度的渐进记法（大 O 记法）**
  - $f(n) = 6n^3 + 4n^2 + 7 = O(n^3)$　例：LMMSE 检测
  - $f(n) = 6n\log_b n + n\log_b \log_a n + 7 = O(n\log n)$　例：FFT 算法

时间复杂度

- **定义**：令 M 是一个确定型图灵机。M 的运行时间（时间复杂度）是一个函数 $f(n): N \to N$，其中 N 是非负整数集合，$f(n)$ 是 M 在所有长度为 n 的输入上运行时所经过的最大步数。

- **复杂度的渐进记法（大 O 记法）**
  - $f(n) = 6n^3 + 4n^2 + 7 = O(n^3)$　例：LMMSE 检测
  - $f(n) = 6n \log_b n + n \log_b \log_a n + 7 = O(n \log n)$　例：FFT 算法
  - 多项式界：$O(n^c)$

时间复杂度

- **定义**：令 M 是一个确定型图灵机。M 的运行时间（时间复杂度）是一个函数 $f(n): N \to N$，其中 N 是非负整数集合，$f(n)$ 是 M 在所有长度为 n 的输入上运行时所经过的最大步数。

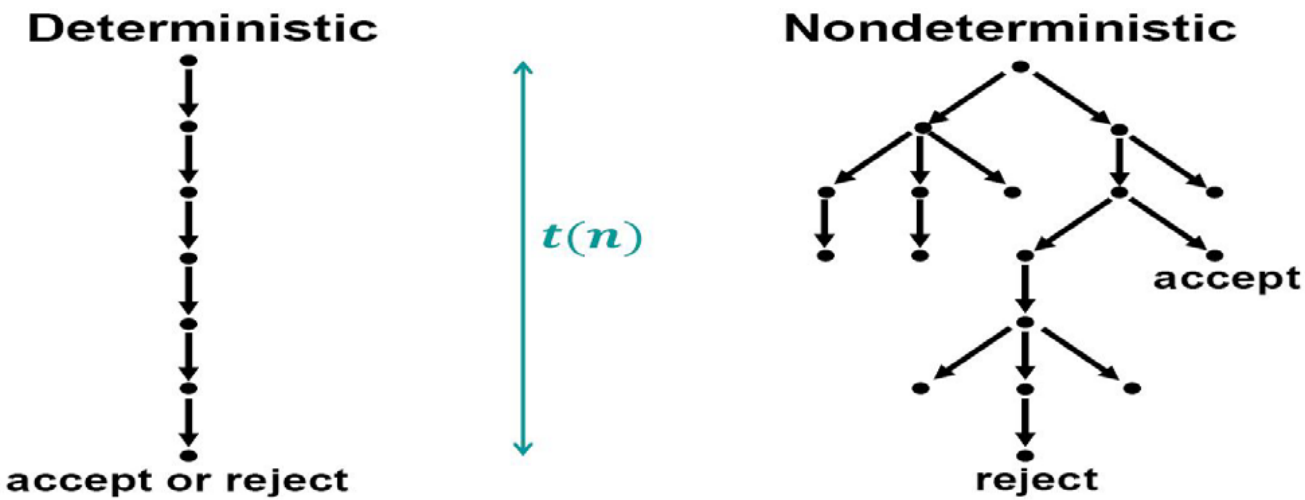
- **复杂度的渐进记法（大 O 记法）**
  - $f(n) = 6n^3 + 4n^2 + 7 = O(n^3)$　例：LMMSE 检测
  - $f(n) = 6n \log_b n + n \log_b \log_a n + 7 = O(n \log n)$　例：FFT 算法
  - 多项式界：$O(n^c)$
  - 指数界：$O(2^{n^\delta})$，其中 $\delta > 0$　例：MIMO MAP 检测

模型间复杂性关系

定义：设 N 是一个非确定型图灵机（每一步均按由输入和状态确定的方向产生输出并转移到下一个状态）。N 的运行时间是函数 $f(n) : N \to N$，其中 $f(n)$ 是在任意长度为 n 的输入上所有计算分支中的最大步数。

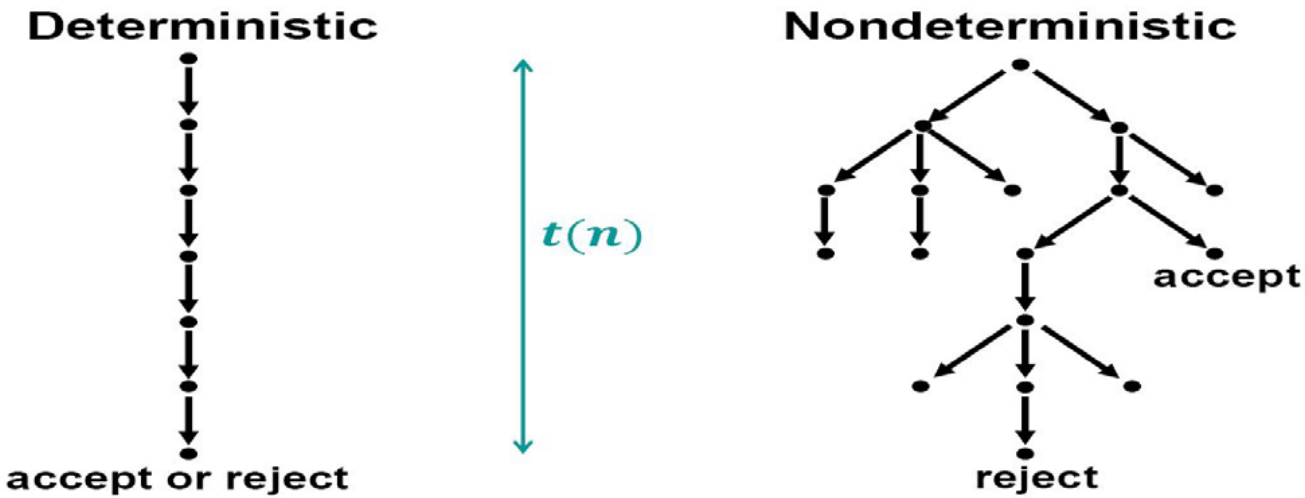
模型间复杂性关系

定义：设 N 是一个非确定型图灵机（每一步均按由输入和状态确定的方向产生输出并转移到下一个状态）。N 的运行时间是函数 $f(n) : N \to N$，其中 $f(n)$ 是在任意长度为 n 的输入上所有计算分支中的最大步数。



模型间复杂性关系

定义：设 N 是一个非确定型图灵机（每一步均按由输入和状态确定的方向产生输出并转移到下一个状态）。N 的运行时间是函数 $f(n) : N \to N$，其中 $f(n)$ 是在任意长度为 n 的输入上所有计算分支中的最大步数。



定理：设 $t(n)$ 是一个函数，$t(n) \geq n$。则每一个 $t(n)$ 时间的非确定型单带图灵机都与某一个 $2^{O(t(n))}$ 的确定型单带图灵机等价。

P 类问题

- **可判定**：对所有输入都停机的图灵机

P 类问题

- **可判定**：对所有输入都停机的图灵机
- **定义**：P 类是确定型单带图灵机在多项式时间内可判定的问题。

P 类问题

- **可判定**：对所有输入都停机的图灵机
- **定义**：P 类是确定型单带图灵机在多项式时间内可判定的问题。
- 例子：有向图 PATH 问题—$O(n)$ 复杂度，递归问题

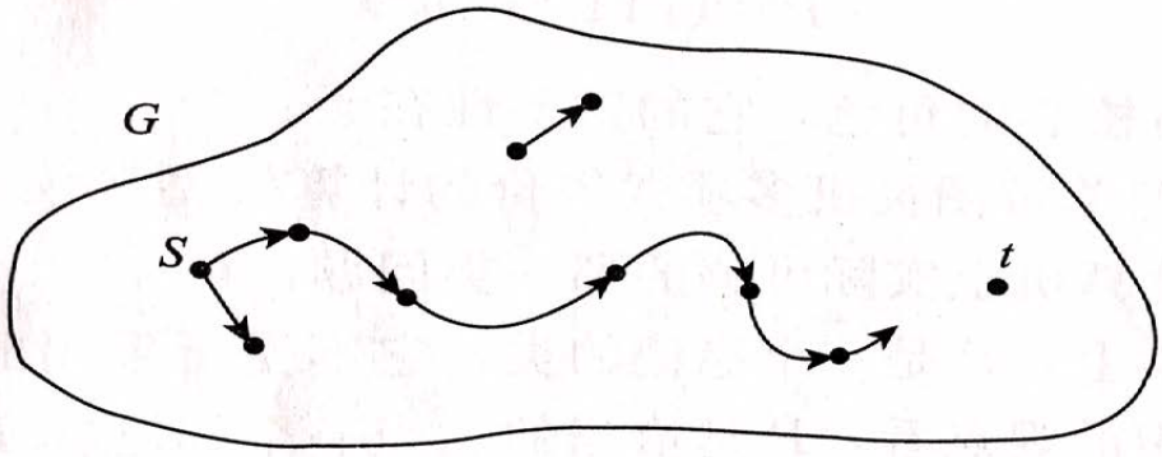


图 7-2　*PATH* 问题：存在从 s 到 t 的路径吗

NP (Non-deterministic Polynomial) 类问题

- **验证机**：验证给定输入是否可被接受

NP (Non-deterministic Polynomial) 类问题

- **验证机**：验证给定输入是否可被接受
- 定义：NP 类是具有多项式时间验证机的问题，也是非确定型图灵机在多项式时间内可解的问题

NP (Non-deterministic Polynomial) 类问题

- **验证机**：验证给定输入是否可被接受
- 定义：NP 类是具有多项式时间验证机的问题，也是非确定型图灵机在多项式时间内可解的问题
- 例子：哈密顿路径问题（邮差问题；电器设计、网络优化，$n!$）
  - 有向图哈密顿路径是通过每个结点恰好一次的有向路径

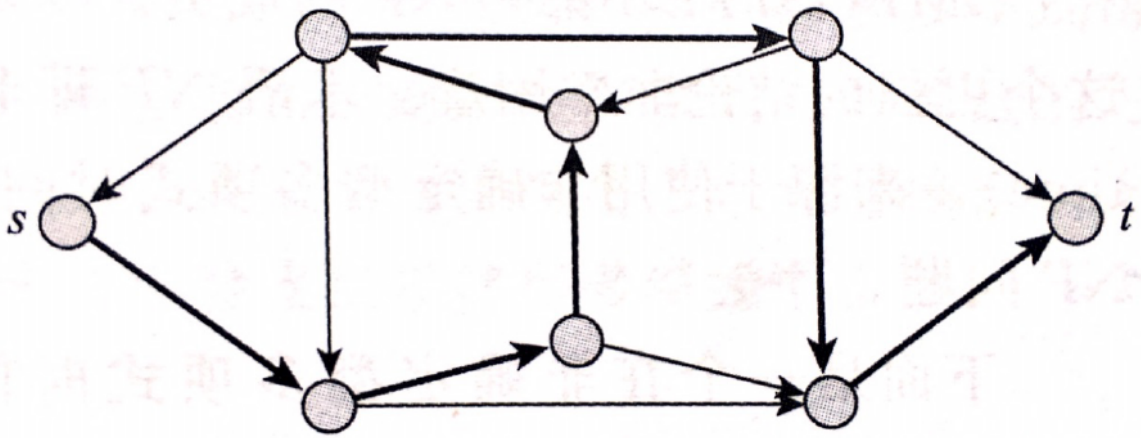
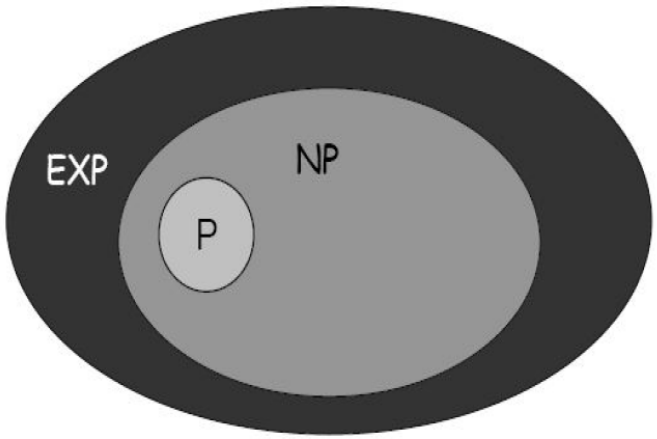
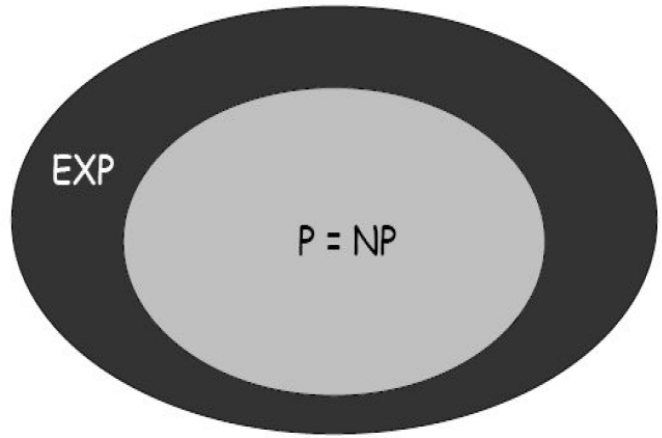


图 7-3　哈密顿路径经过每个结点恰好一次

P=NP ?



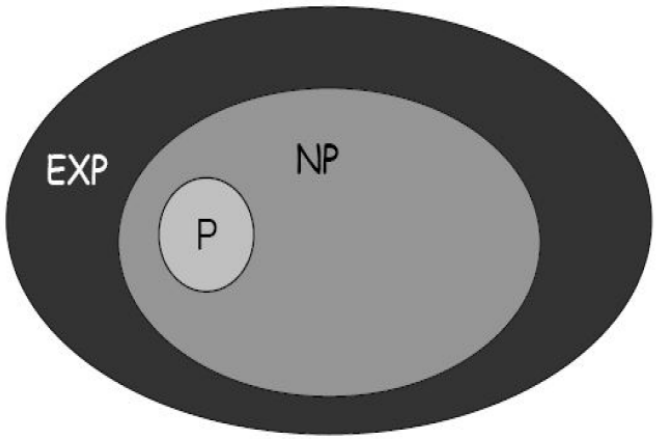
If $P \neq NP$



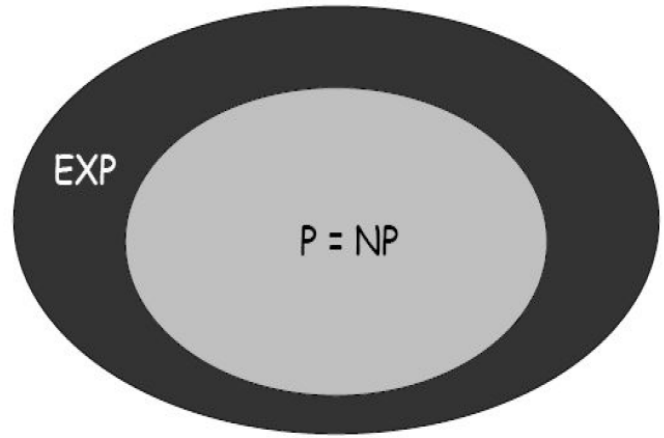
If $P = NP$

美国克雷-Clay 数学研究所：1 million dollar question

P=NP ?



If $P \neq NP$

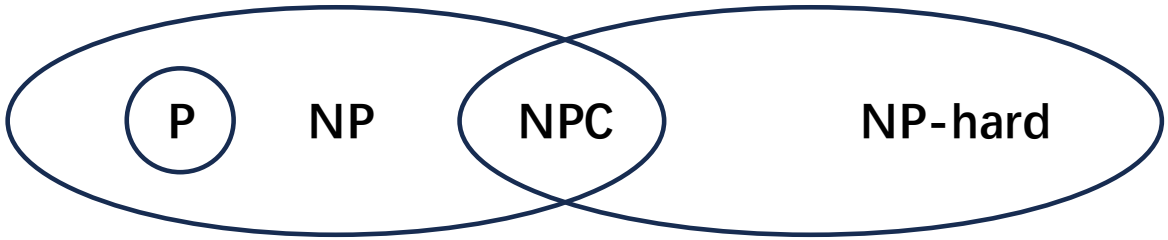


If $P = NP$

美国克雷-Clay 数学研究所：1 million dollar question

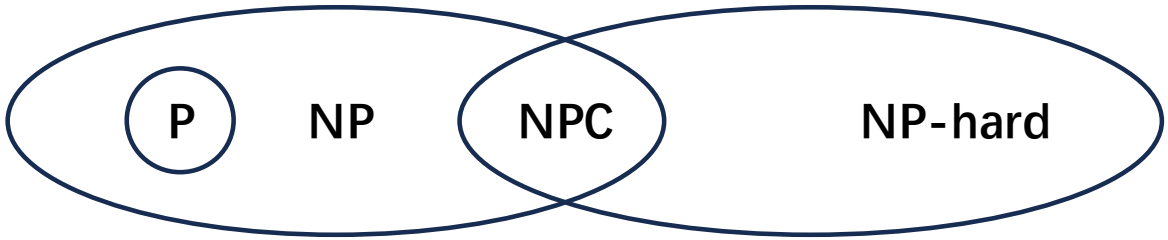
- **研究**：把原来认为是 NP 类的问题变成 P 类问题!
- **例子**：排序问题、MIMO MAP 检测

NP-Hard & NPC



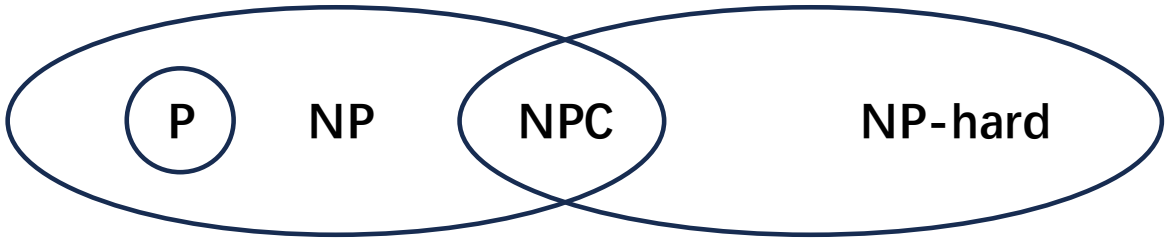
- NP 完全（NPC）问题：
  - 是 NP 问题

NP-Hard & NPC



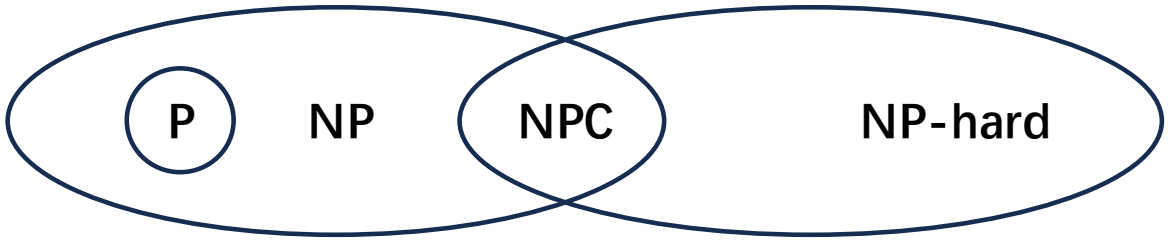
- **NP 完全（NPC）问题**：
  - 是 NP 问题
  - 所有的 NP 问题都可以在多项式时间内归约到该问题

NP-Hard & NPC



- NP **完全**（NPC）**问题**：
  - 是 NP 问题
  - 所有的 NP 问题都可以在多项式时间内归约到该问题
- NP **难**（NP-Hard）**问题**：
  - 所有的 NP 问题都可以在多项式时间内归约到该问题

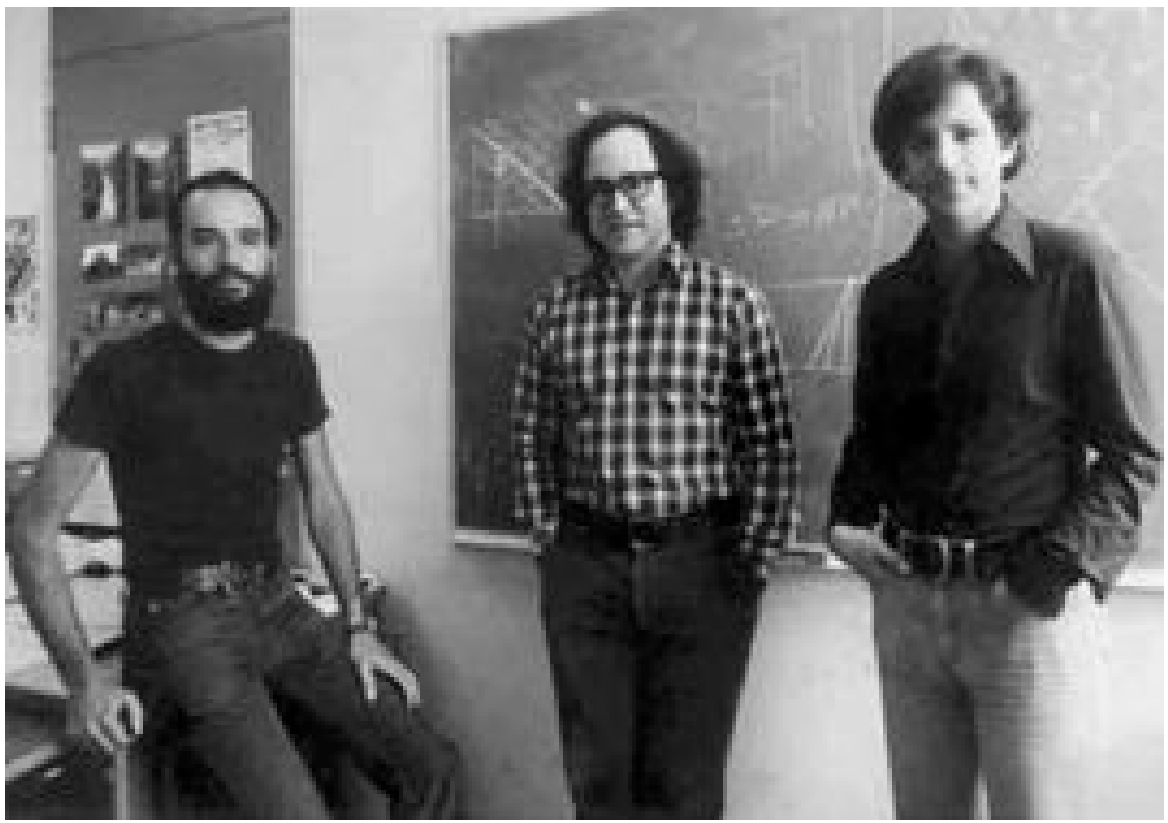
NP-Hard & NPC



- NP **完全**（NPC）**问题**：
  - 是 NP 问题
  - 所有的 NP 问题都可以在多项式时间内归约到该问题
- NP **难**（NP-Hard）**问题**：
  - 所有的 NP 问题都可以在多项式时间内归约到该问题
  - **注**：不一定是 NP 问题

复杂度应用

- **密码学**
  - RSA 算法—WIFI、手机加密
    - 整数因子分解: 不能在有限时间完成整数因子分解



Kolmogorov 复杂度

- 字符串的描述复杂度

1. 0101010101010101010101010101010101010101010101010101010101010101
2. 0110101000001001111001100110011111111001110111100110010010000100 0
3. 1101111001110101111101101111101110101101111000101110010100111011

Kolmogorov 复杂度

- 字符串的描述复杂度

1. 01010101010101010101010101010101010101010101010101010101010101
2. 01101010000010011110011001100111111100111011110011001001000010 00
3. 11011110011101011111011011111011101011011110001011100101001110 11

- 一些观察:
  - 序列 1 可描述为:“打印“01”30 次”

Kolmogorov 复杂度

- 字符串的描述复杂度

1. 01010101010101010101010101010101010101010101010101010101010101
2. 01101010000010011110011001100111111100111011110011001001000010 00
3. 11011110011101011111011011111011101011011110001011100101001110 11

- 一些观察:
  - 序列 1 可描述为:“打印“01”30 次”
  - 序列 2 可描述为:“打印 2—1 的 2 进制表示的前 60 位”

Kolmogorov 复杂度

- 字符串的描述复杂度

1. 0101010101010101010101010101010101010101010101010101010101010101
2. 0110101000001001111001100110011111111001110111100110010010000100 0
3. 1101111001110101111101101111101110101101111000101110010100111011

- 一些观察:
  - 序列 1 可描述为:“打印“01”30 次”
  - 序列 2 可描述为:“打印 2—1 的 2 进制表示的前 60 位”
  - 序列 3 可描述为:“统计序列中 1 的个数,在按字典序排列的序列中的位置”

Kolmogorov 复杂度

定义：关于一个**通用计算机**$\mathcal{U}$，二元字符串 x 的科尔莫戈罗夫复杂度 $K_{\mathcal{U}}(x)$ 定义为

$$K_{\mathcal{U}}(x) = \min_{p:\mathcal{U}(p)=x} l(p)$$

即能够输出 x 并且停止的所有程序的最小长度。(最好的情况)

科尔莫戈罗夫复杂度$K_{\mathcal{U}}(x)$ 在一定程度上反应了字符串 x

- 信息内容
- 冗余度
- 结构化程度

Kolmogorov 复杂度的通用性

通用性：如果 $\mathcal{U}$ 是一个通用计算机，那么对于任意其他的计算机 $\mathcal{A}$，对所有的二元串 $x \in \{0,1\}^*$，均有

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}}$$

其中 $c_{\mathcal{A}}$ 不依赖于 x。

Kolmogorov 复杂度的通用性

通用性：如果 $\mathcal{U}$ 是一个通用计算机，那么对于任意其他的计算机 $\mathcal{A}$，对所有的二元串 $x \in \{0,1\}^*$，均有

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}}$$

其中 $c_{\mathcal{A}}$ 不依赖于 x。

证明：

1. 假设对于计算机 $\mathcal{A}$，有一个输出 x 的程序 $p_{\mathcal{A}}$，于是 $\mathcal{A}(p_{\mathcal{A}}) = x$

Kolmogorov 复杂度的通用性

通用性：如果 $\mathcal{U}$ 是一个通用计算机，那么对于任意其他的计算机 $\mathcal{A}$，对所有的二元串 $x \in \{0,1\}^*$，均有

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}}$$

其中 $c_{\mathcal{A}}$ 不依赖于 x。

证明：

1. 假设对于计算机 $\mathcal{A}$，有一个输出 x 的程序 $p_{\mathcal{A}}$，于是 $\mathcal{A}(p_{\mathcal{A}}) = x$

2. **图示**：在该程序之前加一个模拟程序 $s_{\mathcal{A}}$，告诉计算机 $\mathcal{U}$ 如何模拟计算机 $\mathcal{A}$，因此有：$p = s_{\mathcal{A}} p_{\mathcal{A}}$，其长度为

$$l(p) = l(s_{\mathcal{A}}) + l(p_{\mathcal{A}}) = c_{\mathcal{A}} + l(p_{\mathcal{A}})$$

Kolmogorov 复杂度的通用性

通用性：如果 $\mathcal{U}$ 是一个通用计算机，那么对于任意其他的计算机 $\mathcal{A}$，对所有的二元串 $x \in \{0,1\}^*$，均有

$$K_{\mathcal{U}}(x) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}}$$

其中 $c_{\mathcal{A}}$ 不依赖于 x。

证明：

1. 假设对于计算机 $\mathcal{A}$，有一个输出 x 的程序 $p_{\mathcal{A}}$，于是 $\mathcal{A}(p_{\mathcal{A}}) = x$

2. **图示**：在该程序之前加一个模拟程序 $s_{\mathcal{A}}$，告诉计算机 $\mathcal{U}$ 如何模拟计算机 $\mathcal{A}$，因此有：$p = s_{\mathcal{A}} p_{\mathcal{A}}$，其长度为

$$l(p) = l(s_{\mathcal{A}}) + l(p_{\mathcal{A}}) = c_{\mathcal{A}} + l(p_{\mathcal{A}})$$

3. 对所有二元串 x，有

$$K_{\mathcal{U}}(x) = \min_{p:\, U(p)=x} l(p) \leq \min_{s_{\mathcal{A}}, p_{\mathcal{A}}:\, \mathcal{A}(p_{\mathcal{A}})=x} (l(p_{\mathcal{A}}) + c_{\mathcal{A}}) \leq K_{\mathcal{A}}(x) + c_{\mathcal{A}}$$

条件复杂度

条件复杂度定理：条件复杂度小于序列的长度，即

$$K(x|l(x)) \leq l(x) + c$$

条件复杂度

条件复杂度定理：条件复杂度小于序列的长度，即

$$K(x|l(x)) \leq l(x) + c$$

证明： 输出 x 的一个程序可以是:

Print the following l-bit sequence: $x_1 x_2 \cdots x_{l(x)}$

复杂度的上界和下界

Kolmogorov 复杂度上界（无条件）：

$$K(x) \leq K(x|l(x)) + 2\log l(x) + c$$

注： $2\log l(x)$ 为未知 x 长度的代价，重复让机器知道这些比特表示 x 长度

复杂度的上界和下界

Kolmogorov **复杂度上界（无条件）**：

$$K(x) \leq K(x|l(x)) + 2\log l(x) + c$$

注： $2\log l(x)$ 为未知 x 长度的代价，重复让机器知道这些比特表示 x 长度

Kolmogorov **复杂度下界**：复杂度 $K(x) < k$ 的二进字符串 x 的数目满足

$$|\{x \in \{0,1\}^* : K(x) < k\}| < 2^k$$

证明：所有长度小于 k 的程序总数为

$$1 + 2 + 4 + \cdots + 2^{k-1} < 2^k$$

若干例子

- 例 1：n 个 0 的序列

$$K(000\cdots0|n) = c, \forall n$$

若干例子

- 例 1：n 个 0 的序列

$$K(000\cdots0|n) = c, \forall n$$

- 例 2：π 前 n 位的复杂度

$$K(\pi_1\pi_2\pi_3\cdots\pi_n|n) = c, \forall n$$

若干例子

- 例 1：n 个 0 的序列

$$K(000\cdots 0|n) = c, \forall n$$

- 例 2：π 前 n 位的复杂度

$$K(\pi_1\pi_2\pi_3\cdots\pi_n|n) = c, \forall n$$

- 例 3：整数 n

$$K(n) \leq c + \log^* n, \forall n$$

若干例子

- 例 4: 含 k 个 1 的 n 比特长度序列

$$K(x_1 x_2 x_3 \cdots x_n | n) \leq nH_0\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + \frac{1}{2}\log n + c$$

若干例子

- 例 4: 含 k 个 1 的 n 比特长度序列

$$K(x_1 x_2 x_3 \cdots x_n | n) \leq nH_0\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + \frac{1}{2}\log n + c$$

证：1 **的个数** + **字典序的二进制表示**，程序总长度：

若干例子

- 例 4: 含 k 个 1 的 n 比特长度序列

$$K(x_1 x_2 x_3 \cdots x_n | n) \leq nH_0\left(\frac{1}{n}\sum_{i=1}^{n} x_i\right) + \frac{1}{2}\log n + c$$

证：1 **的个数** + **字典序的二进制表示**，程序总长度：

$$l(p) = c + \log n + \log \binom{n}{k}$$

若干例子

- 例 4: 含 k 个 1 的 n 比特长度序列

$$K(x_1x_2x_3\cdots x_n|n) \le nH_0\left(\frac{1}{n}\sum_{i=1}^{n}x_i\right)+\frac{1}{2}\log n+c$$

证：1 **的个数** + **字典序的二进制表示**，程序总长度：

$$\begin{aligned} l(p) &= c+\log n+\log\binom{n}{k} \\ &\le c'+\log n+nH_0\left(\frac{k}{n}\right)-\frac{1}{2}\log n \end{aligned}$$

注：不等式

$$\binom{n}{k} \le \frac{1}{\sqrt{\pi npq}}2^{nH_0(p)}, \quad p=\frac{k}{n},\ q=1-p$$

Kolmogorov 复杂度与熵

定理：假设随机过程 $\{X_i\}$ 为 i.i.d. 且服从概率密度函数 $f(x), x \in \mathcal{X}$，其中 $\mathcal{X}$ 是一个有限字母表。令 $f(x^n) = \prod_{i=1}^{n} f(x_i)$，那么对于任意的 n，存在一个常数 c，使得

$$H(X) \leq \frac{1}{n} \sum_{x^n} f(x^n) K(x^n|n) \leq H(X) + \frac{(|X|-1)\log n}{n} + \frac{c}{n}$$

从而

$$\mathrm{E}\left[\frac{1}{n} K(X^n|n)\right] \to H(X)$$

Kolmogorov 复杂度与熵

证明:

引理: 对于任意的计算机 $\mathcal{U}$,

$$\sum_{p:u(p)\text{停止}} 2^{-l(p)} \leq 1$$

1) 由**信源编码定理**可知: 码字长度的期望不小于熵, 因此

$$\sum_{x^n} f(x^n) K(x^n|n) \geq H(X_1, X_2, \cdots, X_n) = nH(X)$$

Kolmogorov 复杂度与熵

证明:

引理: 对于任意的计算机 $\mathcal{U}$,

$$\sum_{p:u(p)\text{停止}} 2^{-l(p)} \leq 1$$

1) 由**信源编码定理**可知: 码字长度的期望不小于熵, 因此

$$\sum_{x^n} f(x^n) K(x^n|n) \geq H(X_1, X_2, \cdots, X_n) = nH(X)$$

2) **二元字母表**情况, 即 X_i 服从 i.i.d. Bernoulli(θ). 由例 4 可得

$$\mathrm{E}[K(X_1 X_2 \cdots X_n|n)] \leq nH_0\left(\frac{1}{n}\sum_{i=1}^{n} E[X_i]\right) + \frac{1}{2}\log n + c$$

不可压缩序列

设 $X_1, X_2, \cdots, X_n$ 为服从 Bernoulli$(\frac{1}{2})$ 的一个随机过程。则

$$P(K(X_1 X_2 \cdots X_n | n) < n - k) < 2^{-k}$$

不可压缩序列

设 $X_1, X_2, \cdots, X_n$ 为服从 Bernoulli$(\frac{1}{2})$ 的一个随机过程。则

$$P(K(X_1 X_2 \cdots X_n|n) < n-k) < 2^{-k}$$

证明:

$$\begin{aligned}
&P(K(X_1 X_2 \cdots X_n|n) < |n-k) \\
&= \sum_{x_1 x_2 \cdots x_n : K(x_1 x_2 \cdots x_n|n) < n-k} p(x_1, x_2, \cdots, x_n) \\
&= \sum_{x_1 x_2 \cdots x_n : K(x_1 x_2 \cdots x_n|n) < n-k} 2^{-n} \\
&= |\{x_1 x_2 \cdots x_n : K(x_1 x_2 \cdots x_n|n) < n-k\}| 2^{-n} \\
&< 2^{n-k} 2^{-n} = 2^{-k}
\end{aligned}$$

不可压缩序列

定义：如果

$$\lim_{n\to\infty} \frac{K(x_1 x_2 \cdots x_n | n)}{n} = 1$$

则无限串 x 是不可压缩的 (incompressible)

不可压缩序列

定义：如果

$$\lim_{n\to\infty} \frac{K(x_1 x_2 \cdots x_n | n)}{n} = 1$$

则无限串 x 是不可压缩的 (incompressible)

如果无限串 $x_1, x_2, \cdots$ 是不可压缩的，则

$$\frac{1}{n}\sum_{i=1}^{n} x_i \to \frac{1}{2}$$

不可压缩序列

定义：如果

$$\lim_{n\to\infty} \frac{K(x_1 x_2 \cdots x_n | n)}{n} = 1$$

则无限串 x 是不可压缩的 (incompressible)

如果无限串 $x_1, x_2, \cdots$ 是不可压缩的，则

$$\frac{1}{n}\sum_{i=1}^{n} x_i \to \frac{1}{2}$$

在任何不可压缩的 0-1 串中，0 和 1 的比例几乎相等

作业

符号解释：$\wedge$ =AND; $\neg$ =NOT; $\vee$ =OR

1. 令 x, y 为任意长度的 0,1 序列，证明 $K(x, y) \leq K(x) + K(y) + c$
2. 整数和的复杂度

   a) 证明：$K(n) \leq \log n + 2 \log \log n + c$

   b) 证明：$K(n_1 + n_2) \leq K(n_1) + K(n_2) + c$
3. 给定两个正态分布 $p(x|C_1) \sim \mathcal{N}(\mu_1, \sigma_1^2), p(x|C_2) \sim N(\mu_2, \sigma_2^2), P(C_1)$ 和 $P(C_2)$，计算贝叶斯判别点
4. 设计一个两输入的感知器来实现布尔函数 A$\wedge\neg$B
5. 设计一个两层的感知器网络来实现异或布尔函数 A XOR B。

   提示：A XOR B=(A$\wedge\neg$B)$\vee$($\neg$A $\wedge$ B)
6. 推导输出为 o 的单个单元的梯度下降训练法则，其中

   $o = w_0 + w_1 x_1 + w_1 x_1^2 + \cdots + w_n x_n + w_n x_n^2$