第3章 计算思维和学科基础

计算思维通过广义的计算来描述各类自然过程和社会过程，从而解决各个学科的问题。本章主要从复杂性、抽象、数学建模、仿真、可计算性等计算思维概念出发，讨论各种问题的建模方法、基本算法思想，以及学科经典问题等内容。

3.1 计 算 思 维

3.1.1 计算思维的特征

1. 计算工具与思维方式的相互影响

计算科学专家迪科斯彻(Edsger Wybe Dijkstra)说过：**“我们使用的工具影响着我们的思维方式和思维习惯，从而也将深刻地影响着我们的思维能力”**。计算的发展也影响着人类的思维方式，从最早的结绳计数，发展到目前的电子计算机，人类思维方式发生了相应的改变。如：计算生物学改变着生物学家的思维方式；计算博弈论改变着经济学家的思维方式；计算社会科学改变着社会学家的思维方式；量子计算改变着物理学家的思维方式。计算思维已成为利用计算机求解问题的一个基本思维方法。

2. 计算思维的定义

“计算思维”是哥伦比亚大学(CU)周以真(Jeannette M. Wing)教授提出的一种理论。周以真教授认为：**计算思维是运用计算科学的基础概念去求解问题、设计系统和理解人类行为，它涵盖了计算科学的一系列思维活动**。

国际教育技术协会(ISTE)和计算机研究教师协会(CSTA)2011年给计算思维做了一个可操作性的定义，即计算思维是一个问题解决的过程，该过程包括以下特点：

(1) 拟定问题，并能够利用计算机和其他工具的帮助来解决问题；

(2) 要符合逻辑地组织和分析数据；

(3) 通过抽象，如模型、仿真等再现数据；

(4) 通过算法思想(一系列有序的步骤)，支持自动化的解决方案；

(5) 分析问题，找到最有效的方案，并且有效地应用这些方案和资源；

(6) 将该问题的求解过程进行推广，并移植到更广泛的问题中。

3. 计算思维的特征

周以真教授在《计算思维》论文中提出了以下计算思维的基本特征。

计算思维是人的，不是计算机的思维方式。计算思维是人类求解问题的思维方法，而不是要使人类像计算机那样思考。

计算思维是数学思维和工程思维的融合。计算科学本质上来源于数学思维，但是受计算设备的限制，迫使计算科学专家必须进行工程思考，不能只是数学思考。

计算思维建立在计算过程的能力和限制之上。需要考虑哪些事情人类比计算机做得好，哪些事情计算机比人类做得好，其中最根本的问题是：什么是可计算的。

为了有效地求解一个问题，我们可能要进一步问：一个近似解是否就够了呢？是否允许漏报和误报？计算思维就是通过简化、转换和仿真等方法，把一个看起来困难的问题，重新阐释成一个我们知道怎样解决的问题。

计算思维采用抽象和分解的方法，将一个庞杂的任务分解成一个适合计算机处理的问题。计算思维是选择合适的方式对问题进行建模，使它易于处理。在我们不必理解系统每个细节的情况下，就能够安全地使用或调整一个大型的复杂系统。

根据以上周以真教授的分析可以看出：计算思维以设计和构造为特征。计算思维是运用计算科学的基本概念进行问题求解、系统设计的一系列思维活动。

3.1.2 数学思维的概念

计算思维包含哪些最基本的概念？目前专家们尚无统一的意见。一般来说，计算思维包含数学思维和工程思维两部分。数学思维的基本概念有复杂性、抽象、模型、算法、数据结构、可计算性、一致性和完备性等。

1. 复杂性

大问题的复杂性包括二义性（如语义理解等）、不确定性（如哲学家就餐问题、混沌问题等）、关联（如操作系统死锁问题、大学教师排课问题等）、指数爆炸（如汉诺塔问题、旅行商问题等）、悖论（如罗素理发师悖论、图灵停机问题等）等概念。计算科学专家迪科斯彻曾经指出**“编程的艺术就是处理复杂性的艺术”**。

（1）程序的复杂性。德国科学家克拉默（Friedrich Cramer）在著作《混沌与秩序——生物系统的复杂结构》一书中给出了几个简单例子，用于分析程序的复杂性。

【例 3-1】 序列 A＝{aaaaaaa…}

这是一个简单系统，相应程序为：在每个 a 后续写 a。这个短程序使得这个序列得以随意复制，不管多长都可以编程。

【例 3-2】 序列 B＝{aabaabaabaab…}

与上例相比，该例要复杂一些，这是一个准复杂系统，但仍可以很容易地写出程序：在两个 a 后续写 b 并重复这一操作。

【例 3-3】 序列 C＝{aabaababbaabaababb…}

与上例相似，也可以用短程序来描述：在两个 a 后续写 b 并重复，每当第 3 次重写 b 时，将第 2 个 a 替换为 b。这样的序列具有可定义的结构，可用相应的程序表示。

【例 3-4】 序列 D＝{aababbababbbabaaababbab…}

以上案例中的信息排列毫无规律，如果希望编程解决，就必须将字符串全部列出。这就得出一个结论：**一旦程序大小与试图描述的系统大小相提并论，编程就变得没有意义**。当系统结构不能描述，或者说描述它的最小算法与系统本身具有相同的信息比特数时，则称该系统为根本复杂系统。在达到复杂系统之前，人们可以编写能解决问题的程序。

（2）语义理解的二义性。人们视为智力挑战的问题，计算机做起来未必困难；反而是

一些人类觉得简单平常的脑力活动，机器实现起来可能非常困难。例如，速算曾经是人类智力超群的象征，而计算机无论在速度还是准确率上都毫无争议地超过了人类。但是，“为我做一个西红柿炒鸡蛋”这样简单的问题，计算机理解起来就非常困难。因为这个问题存在太多的二义性，如：“西红柿”要什么品种？成熟到什么程度？是否要大小基本相同？“一个”的语义是什么？是1个西红柿还是1斤西红柿？“做”的语义是什么，是“炒”吗？“炒”的语义又是什么，是不停地翻动吗？翻动的频率要多大？炒多长时间，等等。可见概念模糊的命题不可计算。程序语言与自然语言有所不同，最大的区别是自然语言的同一语句在不同语境下有不同的理解，例如，短语“女朋友很重要吗”，可以理解为“女朋友/很重要吗?”，也可以理解为“女朋友很重/要吗?”。程序语言决不允许出现以上歧义。因此，任何一种程序设计语言都有一套规定的程序语法。

二义性问题在程序设计中会经常遇到。在语言中，二义性是一种语法不完善的说明，在程序设计中应当避免这种情况。解决二义性有两种方法：方法一是设置一些规则，该规则可在出现二义性的情况下指出哪个语法是正确的，它的优点是无须修改文法（可能会很复杂）就能够消除二义性；方法二是将文法改变成一个强制正确的格式。

（3）复杂系统的CAP理论。在分布式系统中，一致性（Consistency）指数据复制到系统N台机器后，如果数据有更新，则N台机器的数据需要一起更新；可用性（Availability）指分布式系统有很好的响应性能。分区容错性（Partition Tolerance）指分布式系统部分机器出现故障时，系统可以自动隔离到故障分区，并将故障机器的负载分配到正常分区继续工作（容错）。埃瑞克·布鲁尔（Eric Brewer）教授指出：对于分布式系统，数据一致性、系统可用性、分区容错性三个目标（合称CAP）不可能同时满足，最多只能满足其中两个。CAP理论给人们以下启示：事物的多个方面往往是相互制衡的，在复杂系统中，冲突不可避免。CAP理论是NoSQL数据库的理论基础。

在系统设计中，常常需要在各方面达成某种妥协与平衡，因为凡事都有代价。例如，分层会对性能有所损害，不分层又会带来系统过于复杂的问题。很多时候结构就是平衡的艺术，明白这一点，就不会为无法找到完美的解决方案而苦恼。复杂性由需求所决定，既要求容量大，又要求效率高，这种需求本身就不简单，因此很难用简单的算法解决。

（4）大问题的不确定性。大型网站往往有成千上万台机器，在这些系统上部署软件和管理服务是一项非常具有挑战性的任务。大规模用户服务往往涉及众多程序模块，很多操作步骤。简单性原则就是要求每个阶段、每个步骤、每个子任务都尽量采用最简单的解决方案。这是由于**大规模系统存在的不确定性会增加系统复杂性**。即使做到了每个环节最简单，但是由于不确定性的存在，整个系统还是会出现不可控的风险。

【例3-5】 计算科学专家杰夫·迪恩（Jeff Dean）在介绍大规模数据中心遇到的难题时指出：大部分机器处理请求的平均响应时间为1ms左右，只有1%机器的请求处理时间会大于1s。如果一个请求需要由100个节点机器并行处理，那么就会出现63%的请求响应时间大于1s，这完全不可接受。面对这个复杂的不确定性问题，Jeff Dean和Google公司做了很多工作来解决这个问题。

程序的复杂性来自于大量的不确定性，如需求不确定、功能不确定、输入不确定、运行环境不确定等，这些不确定性无法避免。程序的需求会以各种方式变化，而且往往会向程序员没有预料到的方向发展。由于不确定性的存在，在系统设计中，应当遵循KISS（Keep It

Simple，Stupid，保持简单）原则，推崇**简单就是美，任何没有必要的复杂都需要避免**（奥卡姆剃刀原则）。但是要做到 KISS 原则并不容易，人们遇到问题时，往往会从各个方面去考虑，其中难免包含了问题的各种细枝末节，这种思维方式会导致问题变得非常复杂。

2. 抽象

（1）艺术的抽象。在美术范畴内，抽象的表现最简单省力，也最复杂费力。从理论上讲，抽象体现人的主观意识，因此只要画得怪，都可以称为抽象画。有才华的画家认为抽象艺术最美但又最难画，其中包含的艺术内涵太丰富。

【例 3-6】 如图 3-1 所示，毕加索终生喜欢画牛，年轻时他画的牛体形庞大，有血有肉，威武雄壮。但随着年龄的增长，他画的牛越来越突显筋骨。到他八十多岁时，他画的牛只有寥寥数笔，乍看上去就像一副牛的骨架。而牛外在的皮毛、血肉全部没有了，只剩一副具有牛神韵的骨架了。

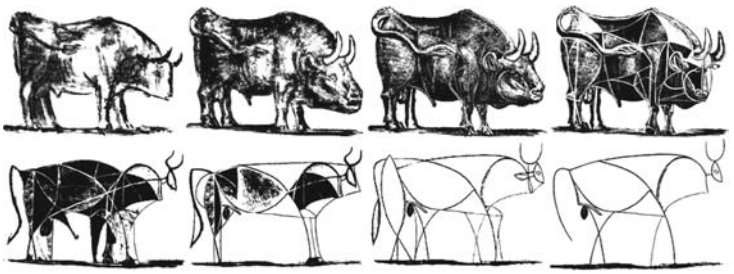


图 3-1　毕加索画牛的抽象过程

（2）计算思维的抽象。计算的根本问题是什么能被有效地自动进行。自动化要求对进行计算的事物进行某种程度的抽象，计算思维中的抽象最终要能够利用机器一步步自动执行。为了确保机器的自动化，就需要在抽象过程中进行精确和严格的符号转换和建立数学模型。抽象是对实际事物进行人为处理，抽取所关心的、共同的、本质特征的属性，并对这些事物及其特征属性进行描述，从而大大降低系统元素的数量。计算思维的抽象方法有：分解、简化、剪枝、替代、分层、模型化、公式化、形式化等。

【例 3-7】 为了实现程序的自动化计算，需要将问题抽象为一个数学模型。如将不可计算问题抽象为停机问题；欧拉将哥尼斯堡七桥问题抽象为“图论”问题；将解决问题的步骤抽象为算法；将机器语言翻译抽象为统计语言模型等。

【例 3-8】 对数据的抽象方法有：对数值、字符、图形、音频、视频等信息，抽象为二进制数字；数据之间的关系有顺序、层次、树形、图形等类型，数据结构就是对这些关系的抽象。程序设计中的抽象方法有：将数据存储单元地址抽象为变量名；将复杂的函数程序抽象为简单的应用程序接口（API）；对运行的程序抽象为进程等。

【例 3-9】 计算机硬件中的抽象方法有：将集成电路的设计抽象为布尔逻辑运算；将不同的硬件设备抽象为 HAL（Hardware Abstraction Layer，硬件抽象层）；对 I/O 设备的操作抽象为文件操作；将不同体系结构的计算机抽象为虚拟机等。

3. 分解

笛卡儿（René Descartes）在《谈方法》一书中指出：**“如果一个问题过于复杂以至于一下子难以解决，那么就将原问题分解成足够小的问题，然后再分别解决”**。

(1) 利用等价关系进行系统简化。复杂系统可以看成是一个集合,降低集合复杂性的最好办法是使它有序,也就是按“等价关系”对系统进行分解。通俗地说,就是将一个大系统划分为若干子系统,使人们易于理解和交流。这样,子系统不仅具有某种共同的属性,而且可以完全恢复到原来的状态,从而大大降低系统的复杂性。

(2) 利用分治法思想进行分解。分而治之是指把一个复杂问题分解成若干简单的问题,然后逐个解决。这种朴素的思想来源于人们生活与工作的经验,并且完全适用于技术领域。编程人员采用分治法时,应着重考虑:复杂问题分解后,每个子问题能否用程序实现?所有程序最终能否集成为一个软件系统?软件系统能否解决这个复杂的问题?

3.1.3 工程思维的概念

工程思维的基本概念有效率、资源、兼容性、硬件与软件、模型和结构、时间和空间、编码(转换)、模块化、复用、安全、演化、折中与结论等。

1. 效率

效率始终是计算领域重点关注的问题。例如,为了提高程序的执行效率,采用并行处理技术;为了提高网络传输效率,采用信道复用技术;为了提高 CPU 利用率,采用流水线技术;为了提高 CPU 处理速度,采用高速缓存技术等。

【例 3-10】 计算领域“优先”技术有:系统进程优先、中断优先、重复执行的指令优先等,它们体现了效率优先的原则;而队列、网络数据包转发等,体现了平等优先的原则。效率与平等的选择需要根据实际问题进行权衡分析。如绝大部分算法都采用效率优先原则,但是也有例外。如“树”的广度搜索和深度搜索中,采用了平等优先原则,即保证树中每个节点都能够被搜索到,因而搜索效率很低;而启发式搜索则采用效率优先原则,它会对树进行“剪枝”处理,因此不能保证树中每个节点都会被搜索到。在实际应用中,搜索引擎同样不能保证因特网中每个网页都会被搜索到,棋类博弈程序也是如此。

但是,效率是一把双刃剑,经济学家奥肯(Arthur M. Okun)在《平等与效率——重大的抉择》中断言:**“为了效率就要牺牲某些平等,并且为了平等就要牺牲某些效率”**。奥肯的论述同样适用于计算领域。

2. 兼容性

计算机硬件和软件产品遵循向下兼容的设计原则。在计算机产品中,新一代产品总是在老一代产品的基础上进行改进。新设计的计算机软件和硬件应当尽量兼容过去设计的软件系统,兼容过去的体系结构,兼容过去的组成部件,兼容过去的生产工艺,这就是“向下兼容”。计算机产品无法做到“向上兼容”(或向前兼容),因为老一代产品无法兼容未来的系统,只能是新一代产品来兼容老产品。

【例 3-11】 老式 CRT(Cathode Ray Tube,阴极射线管)采用电子束逐行扫描方式显示图像,而新型 LCD(Liquid Crystal Display,液晶显示器)没有电子束,原理上也不需要逐行扫描,一次就能够显示整屏图像。但是为了保持与显卡、图像显示程序的兼容性,LCD 不得不沿用老式的逐行扫描技术。

兼容性降低了产品成本,提高了产品可用性,同时也阻碍了技术发展。各种老式的、正在使用的硬件设备和软件技术(如 PCI 总线、复杂指令系统、串行编程方法等),它们是计算机领域发展的沉重负担。如果不考虑向下兼容问题,设计一个全新的计算机时,完全可以采

用现代的、艺术的、高性能的结构和产品，如苹果 iPad 就是典型案例。

3. 硬件与软件

早期计算机中，硬件与软件之间的界限十分清晰。随着技术发展，软件与硬件之间的界限变得模糊不清了。特兰鲍姆(Andrew S. Tanenbaum)教授指出：**“硬件和软件在逻辑上是等同的”。“任何由软件实现的操作都可以直接由硬件来完成，任何由硬件实现的指令都可以由软件来模拟”**。某些功能既可以用硬件技术实现，也可以用软件技术实现。

【例 3-12】 硬件软件化。硬件软件化是将硬件的功能由软件来实现，它屏蔽了复杂的硬件设计过程，大大降低了产品成本。例如，在 x86 系列 CPU 内部，用微指令来代替硬件逻辑电路设计。微指令技术增加了指令设计的灵活性，同时也降低了逻辑电路的复杂性。另外，冯·诺依曼计算机结构中的“控制器”部件，目前已经由操作系统取代。目前流行的虚拟机、虚拟仪表、软件定义网络(Software Defined Network，SDN)等，都是硬件设备软件化的典型案例。

【例 3-13】 软件硬件化。软件硬件化是将软件实现的功能设计成逻辑电路，然后将这些电路制造到集成电路芯片中，由硬件实现其功能。硬件电路的运行速率要大大高于软件，因而，软件硬件化能够大大提升系统的运行速率。例如，实现两个符号的异或运算时，软件实现的方法是比较两个符号的值，再经过 if 控制语句输出运算结果；硬件实现的方法是直接利用逻辑门电路实现异或运算。视频数据压缩与解压缩、3D 图形的几何建模和渲染、数据奇偶检验、网络数据打包与解包等，目前都采用专业芯片处理，这是软件技术硬件化的典型案例。**可见硬件和软件的界限可以人为划定，并且经常变化**。

【例 3-14】 软件与硬件的融合。在 TCP/IP(Transmission Control Protocol/Internet Protocol，传输控制协议/网际协议)网络中，信号比特流通过物理层硬件设备高速传输。而网络层、传输层和应用层的功能是控制比特传输，实现传输的高效性和可靠性等。实际中，应用层的功能主要由软件实现，而传输层和网络层则是软硬件相互融合。如传输层的设备是交换机，网络层的设备是路由器，在这两台硬件设备上，都需要加载软件(如数据成帧、地址查表、路由算法等)，以实现对传输的控制。如果只用硬件设备，会使设备复杂化，而且不一定能很好地实现控制功能；如果只使用软件，程序也会变得很复杂，某些接口功能实现困难，而且程序运行效率较低，这对有实时要求的应用(如数据中心)是致命缺陷。交换机和路由器是软硬件相互融合、协同工作的经典案例。

一般来说，**硬件实现某个功能时，具有速度快，占用内存少等优点，但是可修改性差，成本高；而软件实现某个功能时，具有可修改性好，成本低等优点，但是速度低，占用内存多**。具体采用哪种设计方案实现功能，需要对软件和硬件进行折中考虑。

4. 折中与结论

在计算领域产品设计中，经常会遇到：性能与成本、易用性与安全性、纠错与效率、编程技巧与可维护性、可靠性与成本、新技术与兼容性、软件实现与硬件实现、开放与保护等相互矛盾的设计要求。单方面看，每项指标都很重要，在鱼与熊掌不可兼得的情况下，计算科学专业人员必须做出折中和结论。

【例 3-15】 计算机工作过程中，由于电磁干扰、时序失常等原因，可能会出现数据传输和处理错误。如果每个步骤都进行数据错误校验，则计算机设计会变得复杂无比。因此，是否进行数据错误校验，数据校验的使用频度如何，需要进行性能与复杂性方面的折中考虑。例如，在个人微机中，性能比安全性更加重要，因此内存条一般不采用奇偶校验和 ECC 功

能，以提高内存的工作效率；但是在服务器中，一旦系统崩溃将造成重大损失（如股票交易服务器的崩溃），因此服务器内存条的安全性要求大于工作效率，奇偶校验和 ECC 是服务器内存必不可少的设计要求。

3.1.4 问题求解的方法

利用计算技术解决问题时，一般需要经过以下几个步骤：一是理解问题，寻找解决问题的条件；二是对一些具有连续性质的现实问题，进行离散化处理；三是从问题抽象出一个适当的数学模型，然后设计或选择一个解决这个数学模型的算法；四是按照算法编写程序，并且对程序进行调试和测试，最后运行程序，直至得到最终答案。

1. 寻找解决问题的条件

（1）界定问题。解决问题首先要对问题进行界定，弄清楚问题到底是什么，不要被问题的表象迷惑。只有正确地界定了问题，才能找准应该解决的目标，后面的步骤才能正确地执行。如果找不准目标，就可能劳而无获，甚至南辕北辙。

（2）寻找解题的条件。在**“简化问题，变难为易”**的原则下，尽力寻找解决问题的必要条件，以缩小问题求解范围。当遇到一道难题时，可以尝试从最简单的特殊情况入手，找出有助于简化问题、变难为易的条件，逐渐深入，最终分析归纳出解题的步骤。

例如，在一些需要进行搜索求解的问题中，一般可以采用深度优先搜索和广度优先搜索。如果问题的搜索范围太大（如棋类博弈），减少搜索量最有效的手段就是“剪枝”（删除一些对结果没有影响的分支问题），即建立一些限制条件，缩小搜索的范围。如果问题错综复杂，可以尝试从多个侧面分析和寻找必要条件；或者将问题分解后，根据各部分的本质特征，再来寻找各种必要条件。

2. 对象的离散化

计算机处理的对象有一部分本身就是离散化的，如数字、字母、符号等；但是在很多实际问题中，信息都是连续的，如图像、声音、时间、电压等自然现象和社会现象。凡是“可计算”的问题，处理对象都是离散型的，因为计算机建立在离散数字计算的基础上。所有连续型问题必须转换为离散型问题后（数字化），才能被计算机处理。

【例 3-16】 在计算机屏幕上显示一张图片时，计算机必须将图片在水平和垂直方向分解成一定分辨率的像素点（离散化）；然后将每个像素点再分解成红绿蓝（RGB）三种基本颜色；再将每种颜色的变化分解为 0～255（1 字节）个色彩等级。这样计算机就会得到一大批有特定规律的离散化数字，计算机也就能够任意处理这张图片了，如图片的放大、缩小、旋转、变形、变换颜色等操作。

3. 解决问题的算法（数学建模）

求解一个问题时，可能会有多种算法可供选择，选择标准首先是算法的正确性、可靠性、简单性；其次是算法所需要的存储空间和执行速度等。

（1）问题的抽象描述。遇到实际问题时，首先将其形式化，将问题抽象为一个一般性的数学问题。对需要解决的问题用数学形式描述它，先不要管是否合适。然后通过这种描述来寻找问题的结构和性质，看看这种描述是不是合适，如果不合适，再换一种方式。通过反复地尝试、不断地修正来达到一个满意的结果。遇到一个新问题时，通常都是先用各种各样的小例子去不断地尝试，从中发现问题的关键性质。

(2) 理解算法的适应性。需要观察问题的结构和性质,每个实际问题都有它相应的性质和结构。每种算法技术和思想,如穷举法、分治算法、贪心算法、动态规划、遗传算法、蒙特卡罗算法等,都有它们适宜解决的问题。例如,动态规划适宜解决的问题需要有最优子结构和重复性子问题。一旦我们观察出问题的结构和性质,就可以用现有的算法去解决它。用数学方式表述问题,有利于总结出问题的结构和性质。

(3) 建立算法。建立数学模型时,找出问题的已知条件、求解的目标,以及已知条件和目标之间的联系。算法描述形式有数学模型、数据表格、结构图形、伪代码、程序流程图等。获得了算法并不等于问题可解,问题是否可解还取决于算法的复杂性,即算法所需要的时间和空间在数量级上能否接受。

4. 程序设计

图灵在论文《计算机器与智能》中指出:"如果一个人想让机器模仿计算员执行复杂的操作,他必须告诉计算机要做什么,并把结果翻译成某种形式的指令表。这种构造指令表的行为称为编程"。算法对问题求解过程的描述比程序简单,用编程语言对算法经过细化编程后,可以得到计算机程序,而执行程序就是执行用编程语言表述的算法。

3.1.5 数学模型的构建

1. 数学模型

模型是将研究对象通过抽象、归纳、演绎、类比等方法,用适当形式描述的表达方式。简单地说,模型是系统的简化表示,每个模型都是对现实世界的近似模拟,没有完美的模型。模型的类型有实体模型(如汽车模型、城市规划模型等)、仿真模型(如飞行器实验仿真、天气预测模型等)、抽象模型(如数学模型、结构模型、思维模型等)。计算科学经常采用仿真模型和抽象模型来解决问题。

数学模型是用数学语言描述的问题。数学模型可以是一个数学公式、一组代数方程,也可以是它们的某种组合。数学表达式仅仅是数学模型的主要形式之一,但切不可误认为"数学模型就是数学表达式"。数学模型也可以用符号、图形、表格等形式进行描述。

所有数学模型均可转换为算法和程序。数值型问题相对容易建立数学模型,而非数值型问题建模则相对复杂。一些无法直接建立数学模型的系统,如抽象思维、社会活动、人类行为等,需要先将这些问题符号化,然后再建立它们的数学模型。数学模型应用日益广泛的原因在于:社会生活各个方面日益数字化;计算技术的发展为精确化提供了条件;很多无法试验或费用很大的试验(如天气预报),用数学模型进行研究是一条捷径。

2. 数学建模的一般方法

计算领域解决问题时,建立数学模型是十分关键的一步。笛卡儿(René Descartes)设计了一种希望能够解决各种问题的万能方法,它的大致模式是:第一,把所有问题转换为数学问题;第二,把所有数学问题转换为一个代数问题;第三,把所有代数问题归结到解一个方程式。这也是现代数学建模思想的来源。

数学建模方法有以下两大类。一是采用原理分析方法建模,选择常用算法有针对性地对模型进行综合;二是采用统计分析方法建模,通过随机化等方法得到问题的近似数学模型(如语音识别),数学模型出错概率会随计算次数的增加而显著减少。

3. 商品提价的数学建模案例

如果将问题抽象为数学模型,问题就可以用计算方法求解。例如,将"讨价还价"行为看

作一场博弈，则可以将问题抽象成数学模型，然后用程序求解。

【例 3-17】 商品提价问题建立的数学模型。商场经营者既要考虑商品的销售额、销售量，同时也要考虑如何在短期内获得最大利润。这个问题与商品定价有直接关系，定价低时，销售量大但利润小；定价高时，利润大但销售量减少。假设某商场销售的某种商品单价为 25 元，每年可销售 3 万件。设该商品每件提价 1 元，则销售量减少 0.1 万件。如果要使总销售收入不少于 75 万元，求该商品的最高提价。数学模型建立方法如下。

(1) 分析问题。

已知条件：单价 25 元×销售 3 万件＝销售收入 75 万元；

约束条件 1：每件商品提价 1 元，则销售量减少 0.1 万件；

约束条件 2：保持总销售收入不少于 75 万元。

(2) 建立数学模型。

设最高提价为 x 元，提价后的商品单价为：$(25+x)$元；

提价后的销售量为：$(30\,000-1000x/1)$件；

则：$(25+x)\times(30\,000-1000x/1)\geqslant 750\,000$；

简化后数学模型为：$(25+x)\times(30-x)\geqslant 750$。

4. 编程求解

求解例 3-17 问题的 Python 程序如下。

```
>>> from sympy import symbols, solve                    # 导入第三方包 - 符号计算
>>> x = symbols('x')                                    # 设置 x 为符号
>>> f = solve((25 + x) * (30 - x) >= 750)               # 计算不等式取值范围
>>> print('x 的取值范围是:', f)                          # 打印结果
x 的取值范围是: (0 <= x) & (x <= 5)                     # x 大于或等于 0,而且小于或等于 5
```

对以上问题编程求解后：$x\leqslant 5$，即提价最高不能超过 5 元。

3.2 建模案例

建模是对问题求解的抽象过程。有人认为，建立数学建模需要专业的数学知识，其实很多数学模型并不涉及高深的数学知识，如“平均收入”的安全计算。即便是对复杂系统的研究(如细胞自动机)，有时只需要几条简单的规则。

3.2.1 囚徒困境：博弈策略建模

1. 博弈论概述

如果有两人以上参与，且双方可以通过不同策略相互竞争的游戏，而且一方采用的策略会对另一方的行为产生影响，这种情况称为博弈。1944 年，冯·诺依曼和奥斯卡·摩根斯特恩(Oskar Morgenstern)发表了《博弈论和经济行为》著作，首次介绍了博弈论(Game Theory)，他们希望博弈论能为经济问题提供数学解答。

博弈论的基本元素有参与者、行动、信息、策略、收益、均衡和结果。博弈包括同时行动和顺序行动两种类型。在同时行动博弈中，参与者在不了解对方行动的情况下采取行动(如

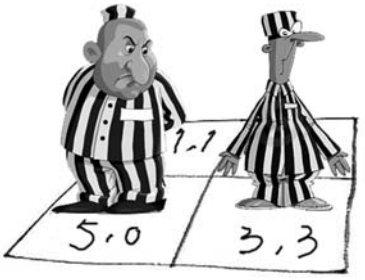
囚徒困境、工程竞标等)。顺序行动博弈采用轮流行动方式,先行动者的动作会明确告知后行动者,博弈参与者轮流行动(如象棋比赛、商业谈判、学术辩论等)。

囚徒困境是两个囚徒之间的一种特殊博弈,它说明了为什么在合作对双方都有利时,保持合作也非常困难。囚徒困境也反映了个人最佳选择而并非团体最佳选择。虽然囚徒困境只是一个模型,但现实中的价格竞争、商业谈判等,都会频繁出现类似的情况。

2. 囚徒困境问题描述

1950年,兰德公司的梅里尔·弗勒德(Merrill Flood)和梅尔文·德雷希尔(Melvin Dresher)根据博弈论拟定出相关困境的理论,后来由艾伯特·塔克(Albert Tucker)以"囚徒困境"的命题进行阐述。经典的囚徒困境描述如例3-18所示。

【例3-18】 警方逮捕了A、B两名嫌疑犯,但没有足够证据指控二人有罪。于是警方分开囚禁嫌疑犯,单独与二人见面,并向双方提供以下相同的选择(如图3-2所示)。



策略	A认罪	A不认罪
B认罪	A=3, B=3	A=0, B=5
B不认罪	A=5, B=0	A=1, B=1

图3-2 囚徒困境的博弈

(1) A、B都认罪并检举对方(互相背叛),则A、B同判3年(A=3,B=3)。

(2) A不认罪并检控B(A背叛),B认罪(B合作),则A获释,B判5年。

(3) B不认罪并检控A(B背叛),A认罪(A合作),则B获释,A判5年。

(4) A、B都不认罪(互相合作),则A、B均判1年(A=1,B=1)。

【例3-19】 用Python程序实现囚徒困境的博弈,代码如下。

```
# E0319.py                                                    # 【建模-囚徒困境】
while True:                                                   # 建立循环判断
    a = input('A,你认罪吗?【1=认罪;2=不认;0=退出】:')          # 输入A的博弈策略
    b = input('B,你认罪吗?【1=认罪;2=不认;0=退出】:')          # 输入B的博弈策略
    if a == '1' and b == '1':                                 # A、B都认罪
        print('A、B都得判3年,唉')                             # 打印博弈结果
    elif a == '2' and b == '1':                               # A不认罪,B认罪
        print('A判0年,B判5年,唉')                             # 打印博弈结果
    elif a == '1' and b == '2':                               # A认罪,B不认罪
        print('A判5年,B判0年,唉')                             # 打印博弈结果
    elif a == '2' and b == '2':                               # A、B都不认罪
        print('A判1年,B判1年.')                               # 打印正确选择
    else:                                                     # 否则
        break                                                 # 退出循环,结束程序
```

```
>>>...(输出略)                                               # 程序运行结果
```

3. 囚徒的策略选择困境

囚徒到底应该选择哪个策略，才能将自己的刑期缩至最短？两名囚徒由于隔绝监禁，并不知道对方的选择；即使他们能够交谈，也未必能够尽信对方，或者会存心欺骗对方。就个人理性选择而言，检举对方（背叛）得到的刑期，总比沉默（合作）要低。困境中两名理性囚徒的可能会做出如下选择。

（1）若对方沉默，背叛会让我获释，所以我会选择背叛。

（2）若对方背叛我，我也要指控对方才能得到较低刑期，所以也选择背叛。

两个囚徒的理性思考都会得出相同的结论：选择背叛。结果二人都要服刑。

在囚徒困境博弈中，如果两个囚徒选择合作，双方都保持沉默，总体利益会更高。而两个囚徒只追求个人利益，都选择背叛时，总体利益反而较低，这就是“困境”所在。

4. 囚徒困境的数学建模

根据囚徒困境的基本博弈思想，可以建立一个数学模型，然后进行编程求解。在社会学和经济学中，经常采用博弈形式分析各种论题，以下是囚徒困境建模的一般形式。

（1）策略符号化。对囚徒困境中的各种行为以 T、R、P、S 符号表示，将囚徒由于各种选择而获得的收益和支付转换为数值，这就获得了表 3-1 所示的符号表。

表 3-1　囚徒困境的符号表

符　号	分　数	英　文	中　文	说　明
T	5	Temptation	背叛收益	单独背叛成功所得
R	3	Reward	合作报酬	共同合作所得
P	1	Punishment	背叛惩罚	共同背叛所得
S	0	Suckers	受骗支付	单独背叛所得

从表 3-1 可见：5＞3＞1＞0，从而得出不等式：T＞R＞P＞S。一个经典的囚徒困境问题必须满足这个不等式，不满足这个条件的问题就不是囚徒困境。但是，以整体获分最高而言，将得出以下不等式：2R＞T＋S（如 2×3 ＞5＋0）或 2R＞2P（如 2×3 ＞2×1），由此可见合作（2R）比背叛（T＋S 或 2P）得分高。如果重复进行囚徒困境的博弈，参与者的策略将会从注重“T＞R＞P＞S”转变为注重“2R＞T＋S”（即“合作优于背叛”），尤其要避免 2P（两人均背叛）的出现。

（2）建立收益和支付矩阵。假设根据以下规则来确定博弈双方的收益和支付值：

一人背叛，一人合作时，背叛者得 5 分（背叛收益），合作者得 0 分（受骗支付）；

二人都合作时，双方各得 3 分（合作报酬）；

二人都背叛时，各得 1 分（背叛惩罚）。

由此得到的收益和支付矩阵如表 3-2 所示。

表 3-2　囚徒困境的收益和支付矩阵表

囚徒的收益和支付矩阵			以符号表示的策略		
策略	A 合作	A 背叛	策略	A 合作	A 背叛
B 合作	A＝3，B＝3	A＝5，B＝0	B 合作	R，R	T，S
B 背叛	A＝0，B＝5	A＝1，B＝1	B 背叛	S，T	P，P

(3) 建立数学模型。由表 3-2 可以建立以下数学模型：

$$\begin{cases} A=R,B=R\text{ 时},A=3,B=3 \\ A=T,B=S\text{ 时},A=5,B=0 \\ A=S,B=T\text{ 时},A=0,B=5 \\ A=P,B=P\text{ 时},A=1,B=1 \end{cases}$$

5. 囚徒困境的博弈策略

密歇根大学的罗伯特·阿克斯罗德(Robert Axelrod)为了研究囚徒困境问题，在 1979 年组织了一场计算机程序比赛。比赛设定了两个前提：一是每个人都是自私的；二是没有权威干预个人决策，每个人完全按照自己利益最大化进行决策。他研究的主要问题是：人为什么要合作？什么时候合作，什么时候不合作？如何使别人与自己合作？

参加博弈的有 14 个程序，每个程序循环对局 300 次，得分最高的是加拿大学者阿纳托尔·拉波波特(Anatol Rapoport)编写的“一报还一报”程序。这个程序的博弈策略是：第一次对局采取合作策略，以后每次对局都采用和对手上一次相同的策略。即对手上一次合作，我这次就合作；对手上一次背叛，我这次就背叛。

6. 囚徒困境模型的应用

在人类社会或大自然都可以找到类似囚徒困境的例子。经济学、政治学、动物行为学、进化生物学等学科，都可以用囚徒困境模型进行研究分析。

【例 3-20】 商业活动中会出现各种囚徒困境的案例。以广告竞争为例：两家公司互相竞争，两家公司的广告互相影响，即一公司的广告被顾客接受则会夺取对方公司的市场份额。如果两家公司同时发出质量类似的广告，则收入增加很少但成本增加较大。两家公司都面临两种选择：好的策略是互相达成协议，减少广告开支(合作)；差的策略是增加广告开支，设法提升广告质量，压倒对方(背叛)。

3.2.2 机器翻译：统计语言建模

长期以来，人们一直梦想能让机器代替人类翻译语言、识别语音、理解文字。计算科学专家从 1950 年开始，一直致力于研究如何让机器对语言做最好的理解和处理。

1. 基于词典互译的机器翻译

早期人们认为只要用一部双向词典和一些语法知识就可以实现两种语言文字间的机器互译，结果遇到了挫折。例如，将英语句子 Time flies like an arrow(光阴似箭)翻译成日语，然后再翻译回来时，竟变成了“苍蝇喜欢箭”；英语句子 The spirit is willing but the flesh is weak(心有余而力不足)翻译成俄语，然后再翻译回来时，竟变成了 The wine is good but the meat is spoiled(酒是好的，肉变质了)。

这些问题出现后，人们发现机器翻译并非想象的那么简单，人们认识到单纯地依靠“查字典”的方法不可能解决机器翻译问题，只有在对语义理解的基础上，才能做到真正的翻译。因此，机器翻译当时被认为是一个完整性问题，即为了解决其中一个问题，你必须解决全部的问题，哪怕是一个简单和特定的任务。

2. 基于语法分析的机器翻译

1957 年，乔姆斯基(Avram Noam Chomsky)提出了“形式语言”理论。形式语言是用数学方法研究自然语言(如英语)和人工语言(如程序语言等)的理论，乔姆斯基提出了形式语

言的表达形式和递归生成方法。

【例 3-21】 用形式语言表示短句“那个穿红衣服的女孩是我的女朋友”。

假设：S＝短句，线条＝改写，V＝动词，VP＝动词词组，N＝名词，NP＝名词词组，D＝限定词，A＝形容词，P＝介词，PP＝介词短语。用形式语言表示的短句如图 3-3 所示。

【例 3-22】 上例用形式语言表示程序的条件语句：if x＝＝2 then {x＝a＋b}。用形式语言表示的程序语句如图 3-4 所示。

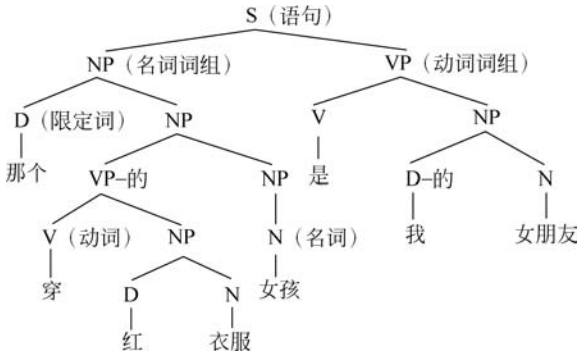


图 3-3 用形式语言表示的短句

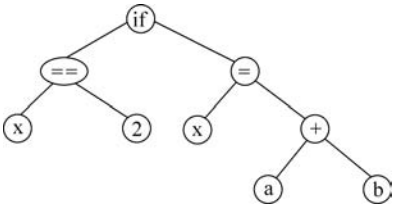


图 3-4 用形式语言表示的程序语句

乔姆斯基提出“形式语言”理论后，人们更坚定了利用语法规则进行机器翻译的信念。为了解决机器翻译问题，人们想到了让机器模拟人类进行学习：这就需要让机器理解人类语言，学习人类的语法，分析语句等。遗憾的是，依靠计算机理解自然语言遇到了极大的困难。机器翻译的某些语句，在语法上很正确，但是语义上无法理解或存在矛盾。

【例 3-23】 The pen is in the box：钢笔在盒子里。

【例 3-24】 The box is in the pen：盒子在钢笔里（正确翻译：盒子在围栏里）。

3. 基于概率统计的机器翻译

两种不同语言交谈的人们，怎样根据信息推测说话者的意思呢？以语音识别为例，当检测到的语音信号为 o_1,o_2,o_3 时，根据这组信号推测发送的短语是 s_1,s_2,s_3。显然，这是在所有可能的短语中，找到可能性最大的一个短语。用数学语言描述就是：在已知 $o_1,o_2,o_3,\cdots$ 的情况下，求概率 $P(o_1,o_2,o_3,\cdots|s_1,s_2,s_3,\cdots)$ 达到最大值的短语 $s_1,s_2,s_3,\cdots$。即

$$P(o_1,o_2,o_3,\cdots \mid s_1,s_2,s_3,\cdots)\cdot P(s_1,s_2,s_3,\cdots) \tag{3-1}$$

式中：$P(o_1,\cdots|s_1,\cdots)$ 是条件概率，它表示在 s_1 发生的条件下，o_1 发生的概率，其中 o_1 和 s_1 具有相关性，读作“在 s_1 条件下 o_1 的概率”；$P(s_1,s_2,s_3,\cdots)$ 是联合概率，表示 s_1、s_2、s_3 等事件同时发生的概率，其中 s_1、s_2、s_3 是相互独立的事件。因此，在式(3-1)中，$P(o_1,o_2,o_3,\cdots|s_1,s_2,s_3,\cdots)$ 表示某个短语 $s_1,s_2,s_3,\cdots$ 被读成 $o_1,o_2,o_3,\cdots$ 的可能性（概率值）。而 $P(s_1,s_2,s_3,\cdots)$ 表示字串 $s_1,s_2,s_3,\cdots$ 成为合理短语的可能性（概率值）。

如果把 $s_1,s_2,s_3,\cdots$ 当成中文，把 $o_1,o_2,o_3,\cdots$ 当成对应的英文，那么就能利用这个模型解决机器翻译问题；如果把 $o_1,o_2,o_3,\cdots$ 当成手写文字得到的图像特征，就能利用这个模型解决手写体文字的识别问题。

4. *N* 元统计语言模型的数学建模

1972 年，美国计算科学专家贾里尼克（Fred Jelinek）用两个隐马尔可夫模型（Markov

Model，一种隐含未知参数的统计模型）建立了统计语音识别数学模型。马尔可夫模型假定下一个词出现的概率只与前一个词有关，这大大减少了计算量。利用马尔可夫模型时，需要先对一个海量语料库进行统计分析，这个过程称为“训练”，它需要把任意两个词语之间关联的概率计算出来。在实际操作中，还会牵涉很多其他复杂的细节。

统计语言模型是基于概率的模型，计算机借助大容量语料库的概率参数，估计出自然语言中每个短语出现的可能性（概率），而不是判断该短语是否符合语法。常用统计语言模型有：N 元文法模型（N-gram Model）、隐马尔可夫模型、最大熵模型等。统计语言模型依赖于单词的上下文（本词与上一个词和下一个词的关系）概率分布。例如，当一个短语片段为“他正在认真……”时，下一个词可以是“学习、工作、思考”等，而不可能是“美丽、我、中国”等。学者们发现，许多词对后面出现的词有很强的预测能力，英语这类有严格语序的语言更是如此。汉语语序较英语灵活，但是这种约束关系依然存在。

【例 3-25】 对短语“南京市长江大桥”进行分词时，可以切分为“南京市/长江/大桥”和“南京/市长/江大桥”，我们会认为前者的切分更合理，因为“长江大桥”与“江大桥”两个分词中，后者在语料库中出现的概率很小。所以，同一个短语中出现若干不同的切分方法时，希望找到概率最大的那个分词。

统计语言模型描述了任意语句 S 属于某种语言集合的可能性（概率 $P(S)$）。例如：P（他/认真/学习）$=0.02$（概率值），P（他/认真/读书）$=0.03$，P（他/认真/坏）$=0$ 等。这里并不要求语句 S 在语法上是完备的，该模型需对任意语句 S 都给出一个概率统计值。

假设一个语句 S 中第 w_i 个词出现的概率，依赖于它前面的 $N-1$ 个词，这样的语言模型称为 N-gram 模型（N 元语言统计模型）。语句 S 出现的概率 $P(S)$ 等于每个词（w_i）出现的概率相乘。语句 S 出现的概率 $P(S)$ 可展开为

$$P(S)=P(w_1)P(w_2\mid w_1)P(w_3\mid w_1\,w_2)\cdots P(w_n\mid w_1\,w_2\cdots w_{n-1}) \tag{3-2}$$

其中，$P(w_1)$ 表示第 1 个词 w_1 出现的概率；$P(w_2|w_1)$ 是在已知第 1 个词的前提下，第 2 个词出现的概率；其余以此类推。不难看出，词 w_n 的出现概率取决于它前面所有词。为了预测词 w_n 的出现概率，必须已知它前面所有词的出现概率。从计算来看，各种可能性太多，太复杂了。因此人们假定任意一个词 w_i 的出现概率只与它前面的词 w_{i-1} 有关（马尔可夫假设），于是问题得到了很大的简化。这时 S 出现的概率就变为

$$P(S)=P(w_1)P(w_2\mid w_1)P(w_3\mid w_1\,w_2)\cdots P(w_n\mid w_1w_2\cdots w_{n-1}) \tag{3-3}$$

接下来的问题是如何估计 $P(w_i|w_{i-1})$。有了大量文本语料库后，这个问题变得很简单，只要数一数这对词（w_{i-1}，w_i）在统计文本中出现了多少次，以及 w_{i-1} 本身在同样文本中前后相邻出现了多少次。简单地说，统计语言模型的计算思维就是：短句 S 翻译成短句 F 的概率是短句 S 中每个单词翻译成 F 中对应单词概率的乘积。根据式（3-3），可以推导出常见的 N 元模型（假设只有 4 个单词的情况）如下。

一元模型为 $P(w_1w_2w_3w_4)=P(w_1)P(w_2)P(w_3)P(w_4)$；

二元模型为 $P(w_1w_2w_3w_4)=P(w_1)P(w_2|w_1)P(w_3|w_2)P(w_4|w_3)$；

三元模型为 $P(w_1w_2w_3w_4)=P(w_1)P(w_2|w_1)P(w_3|w_1w_2)P(w_4|w_2w_3)$。

N 元统计语言模型应用广泛。如以下案例所示，前面语句出现的概率，大大高于后面的语句，因此，根据语言统计模型，正确的选择是前面的语句。

【例 3-26】 中文分词。对语句“已结婚的和尚未结婚的青年”进行分词时：P（已/结

婚/的/和/尚未/结婚/的/青年)>P(已/结婚/的/和尚/未/结婚/的/青年)。

【例 3-27】 机器翻译。对语句 The box is in the pen 进行翻译时：P(盒子在围栏里)≫P(盒子在钢笔里)。

【例 3-28】 拼写纠错。P(about fifteen **minutes** from)>P(about fifteen **minuets** from)。

【例 3-29】 语音识别。P(I saw a van)≫P(eyes awe of an)。

【例 3-30】 音字转换。将输入的拼音 gong ji yi zhi gong ji 转换为汉字时：P(共计一只公鸡>P(攻击一致公鸡)。

统计语言模型的机器翻译过程如图 3-5 示。

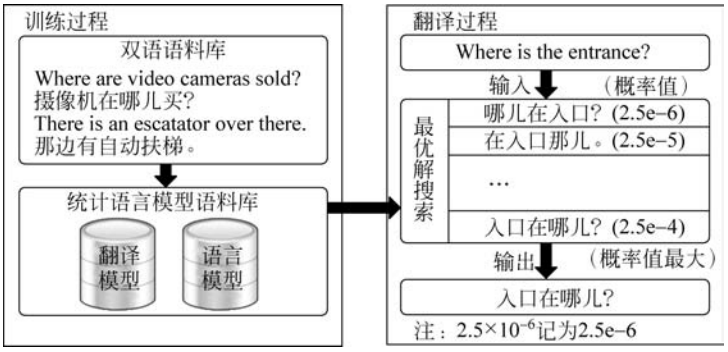


图 3-5 统计语言模型的机器翻译过程

5. 三元统计语言模型的应用

N-gram 模型的缺点在于当 N 较大时，需要规模庞大的语料文本统计库(如 Google N-gram 语料库为 1TB 左右，本书的配套教学资源中附有小型专题语料库)来确定模型的参数。目前使用的 N-gram 模型通常是 N=2 的二元语言模型，或是 N=3 的三元语言模型。以三元模型为例，可以近似地认为任意词 w_i 的出现概率只与它前面两个词有关。

20 世纪 80 年代，李开复博士用统计语言模型成功地开发了世界上第一个大词汇量连续语音识别系统 Sphinx。哈尔滨工业大学研发的微软拼音输入法也是基于三元统计语言模型。三元统计语言模型现在仍然是实际应用中表现最佳的语言模型。据调查，目前市场上的语音听写系统和拼音输入法都是基于三元模型实现的。

早期时，基于概率的“愚蠢”翻译方法居然比语言学家设计的规则系统翻译得更好，这让每个人都感到惊讶。但是，概率统计的翻译方法不可能做到百分之百准确。目前，更好的方法是采用深度神经网络进行机器翻译。

说明：在线简单语句翻译案例，参见本书配套教学资源程序 F3-1.py。

3.2.3 平均收入：安全计算建模

1. 什么是安全多方计算

为了说明什么是安全多方计算(SMC)，先介绍几个实际生活中的例子。

【例 3-31】 很多证券公司的金融研究组用各种方法，试图统计基金的平均仓位，但得出的结果差异较大。为什么不直接发送调查问卷呢？因为基金经理都不愿意公开自己的真实仓位。那么如何在不泄露每个基金真实数据的前提下，统计出平均仓位的精确

值呢？

以上案例具有以下共有特点：一是两方或多方参与基于他们各自私密输入数据的计算；二是他们都不想其他方知道自己输入的数据。问题是在保护输入数据私密性的前提下如何实现计算？这类问题是安全多方计算中的“比特承诺”问题。

2. 简单安全多方计算的数学建模

【例 3-32】 假设一个班级的大学同学毕业 10 年后聚会，大家对毕业后同学们的平均收入水平很感兴趣。但基于各种原因，每个人都不想让别人知道自己的真实收入数据。是否有一个方法，在每个人都不会泄露自己收入的情况下，大家一块算出平均收入呢？

方法是：同学们围坐在一桌，先随便挑出一个人，他在心里生成一个随机数 X，加上自己的收入 $N1$ 后（$S \leftarrow X+N1$）传递给邻座，旁边这个人在接到这个数（S）后，再加上自己的收入 $N2$ 后（$S \leftarrow S+N2$），再传给下一个人，依次下去，最后一个人将收到数加上自己的收入后传给第一个人。第一个人从收到数里减去最开始的随机数，就能获得所有人的收入之和。该和除以参与人数就是大家的平均收入。以上方法的数学模型为

$$\mathrm{SR}=(S-X)/n$$

其中，SR 为平均收入；S 为全体同学收入累计和；X 为初始随机数；n 为参与人数。

以上是美国数学教授大卫·盖尔（David Gale）提出的案例和数学模型。

3. 合谋问题

在例 3-32 的方法中，存在以下几个问题：

（1）模型假设所有参与者都是诚实的，如果有参与者不诚实，则会出现计算错误。

（2）第 1 个人可能谎报结果，因为他是“名义上”的数据集成者。

（3）如果第 1 个人和第 3 个人串通，第 1 个人把自己告诉第 2 个人的数据同时告诉第 3 个人，那么第 2 个人的收入就被泄露了。

问题（1）和问题（2）属于游戏策略问题。问题（3）是否存在一种数学模型，使得对每个人而言，除非其他所有人一起串通，否则自己的收入不会被泄露呢？将在 7.3.5 节的“同态加密”中讨论数据加密计算问题。

4. 姚氏百万富翁问题

“百万富翁问题”是安全多方计算中著名的问题，它由华裔计算科学家、图灵奖获得者姚期智教授提出：两个百万富翁想要知道他们谁更富有，但他们都不想让对方知道自己财富的任何信息。如何设计这样一个安全协议？姚期智教授在 1982 年的国际会议上提出了解决方案，但是这个算法的复杂度较高，它涉及“非对称加密”算法。

百万富翁问题推广为多方参与的情况是：有 n 个百万富翁，每个百万富翁 Pi 拥有 Mi 百万（其中 $1 \leqslant \mathrm{Mi} \leqslant N$）的财富，在不透露富翁财富的情况下，如何进行财富排名。

【例 3-33】 姚氏百万富翁问题的商业应用。假设张三希望向李四购买一些商品，但他愿意支付的最高金额为 x 元；李四希望的最低卖出价为 y 元。张三和李四都希望知道 x 与 y 哪个大。如果 $x>y$，他们都可以开始讨价还价；如果 $x<y$，他们就不用浪费口舌了。但他们都不想告诉对方自己的出价，以免自己在讨价还价中处于不利地位。

解决问题的算法思想是：假设张三和李四设想的价格都低于 100 元，而且双方都不会撒谎（避免囚徒困境）。如图 3-6 所示，准备 100 个编号信封顺序放好，李四回避，张三在小于和等于 x（最高买价）的所有信封内部做一个记号，并且顺序放好；张三回避，李四把顺序

放好的第 y(最低卖价)个信封取出,并将其余信封收起来。张三和李四共同打开这个信封,如果信封内部有张三做的记号,则说明 $x>y$ 或 $x=y$;如果无记号则 $x<y$。由此双方都可以判断商品有没有议价空间。

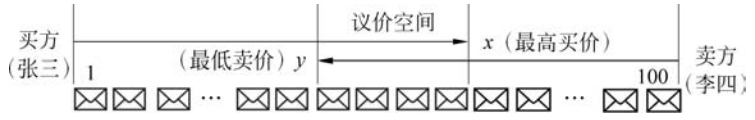


图 3-6　商品买卖中的多方安全计算问题示意图

说明:姚氏百万富翁问题的案例,参见本书配套教学资源程序 F3-2.py。

3.2.4　网页搜索:布尔检索建模

1. 信息检索与布尔运算

当用户在搜索引擎中输入查询语句后,搜索引擎要判断后台数据库中每篇文献是否含有这个关键词。如果一篇文献含有这个关键词,计算机相应地给这篇文献赋值为逻辑值"真"(True);否则赋值逻辑值"假"(False)。

【例 3-34】 当我们查找"原子能应用"文献,但并不想知道如何造原子弹时,可以输入查询关键词"原子能 AND 应用 AND (NOT 原子弹)",表示符合要求的文献必须同时满足三个条件:一是包含"原子能",二是包含"应用",三是不包含"原子弹"。

2. 布尔检索的基本工作原理

网页信息搜索不可能将每篇文档都扫描一遍,检查它是否满足查询条件,因此需要建立一个索引文件。最简单的索引文件是用一个很长的二进制数,表示一个关键词是否出现在每篇文献中。有多少篇文献就需要多少位二进制数(如 100 篇文献要 100bit),每位二进制数对应一篇文献,1 表示文献有这个关键词,0 表示没有这个关键词。

例如,关键词"原子能"对应的二进制数是 01001000 01100001 时,表示第 2、5、10、11、16(左起计数)号文献包含了这个关键词。同样,假设"应用"对应的二进制数是 00101001 10000001 时,那么要找到同时包含"原子能"和"应用"的文献时,只要将这两个二进制数进行逻辑与(AND)运算,即

```
      0100 [1] 000 0110000 [1]      # 1 为 1～16 号文献中包含关键词"原子能"的文献
AND   0010 [1] 001 1000000 [1]      # 1 为 1～16 号文献中包含关键词"应用"的文献
-------------------------------------------------------------
      0000 [1] 000 0000000 [1]      # 1 为 1～16 号文献中包含关键词"原子能 + 应用"的文献
```

根据以上布尔运算结果,表示第 5、16(左起计数)号文献满足查询要求。

计算机做布尔运算的速度非常快。最便宜的微机都可以一次进行 32 位布尔运算,一秒进行 20 亿次以上。当然,由于这些二进制数中绝大部分位数都是 0,只需要记录那些等于 1 的位数即可。

3. 布尔查询的数学模型

(1) 建立"词-文档"关联矩阵数学模型。

【例 3-35】 假设语料库中的记录内容如下。

D1	据报道,感冒病毒近日猖獗……
D2	小王是医生,他对研究电脑病毒也很感兴趣,最近发现了一种……
D3	计算机程序发现了艾滋病病毒的传播途径……
D4	最近我的电脑中病毒了……
D5	病毒是处于生命与非生命物体交叉区域的存在物……
D6	生物学家尝试利用计算机病毒来研究生物病毒……

为了根据关键词检索网页,首先建立文献数据库索引文件。表 3-3 就是文献与关键词的关联矩阵数学模型,其中 1 表示文档中有这个关键词,0 表示没有这个关键词。

表 3-3 “词-文档”关联矩阵数学模型

文　　档	T1＝病毒	T2＝电脑	T3＝计算机	T4＝感冒	T5＝医生	T6＝生物
D1	1	0	0	1	0	0
D2	1	1	0	0	1	0
D3	1	0	1	0	0	0
D4	1	1	0	0	0	0
D5	1	0	0	0	0	0
D6	1	0	1	0	0	1
…	…	…	…	…	…	…

(2) 建立倒排索引文件。为了通过关键词快速检索网页,搜索引擎往往建立了关键词倒排索引表。表每行一个关键词,后面是关键词的文献 ID 等,如表 3-4 所示。

表 3-4　关键词倒排索引表

关键词	文档 ID	文档 ID	文档 ID	文档 ID	文档 ID	文档 ID	附加信息
病毒	D1	D2	D3	D4	D5	D6	出现频率等
电脑		D2		D4			
计算机			D3			D6	
感冒	D1						
医生		D2					
生物						D6	
…	…	…	…	…	…	…	…

4. 网页搜索过程

(1) 建立布尔查询表达式。早期的文献查询系统大多基于数据库,严格要求查询语句符合布尔运算。今天的搜索引擎相比之下要聪明得多,它会自动把用户的查询语句转换成布尔运算的关系表达式。

【例 3-36】 假设用户在浏览器中输入的查询语句为“查找计算器病毒的资料”。搜索引擎工作过程如图 3-7 所示,搜索引擎首先对用户输入的关键词进行检索分析。如:进行中文分词(将语句切分为“查找/计算器/病毒/的/资料”);过滤停止词(清除“查找、的、资料”等);纠正用户拼写错误(如将“计算器”改为“计算机”)等操作。

然后,关键词转换为布尔表达式:Q＝病毒 AND(计算机 OR 电脑)AND((NOT 感冒)OR(NOT 医生)OR(NOT 生物))。也就是需要搜索包含“计算机”“病毒”,不包含“感冒”

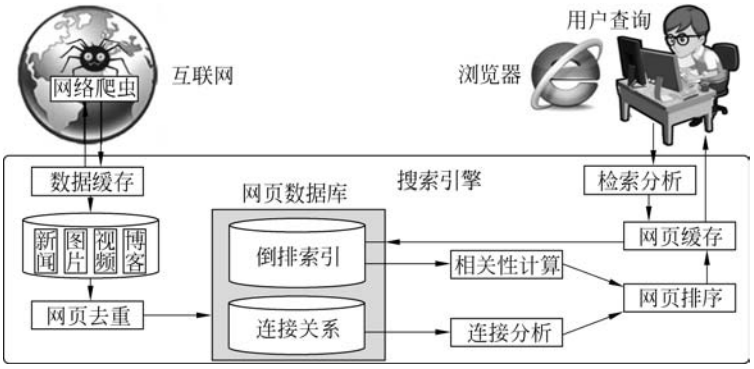


图 3-7　搜索引擎工作过程

“医生”“生物”词语的文档。

(2) 进行布尔位运算。对表 3-4 进行布尔位运算,符合查询要求的网页为 D2、D3、D4、D6,如果按这个排名显示网页显然不太合理。因此,网页还需要经过相关性计算后再排序显示。

(3) 网页排序显示。搜索引擎对查询语句进行关联矩阵运算后,就可以找出含有所有关键词的文档。但是搜索的文档经常会有几十万甚至上千万份,通常搜索引擎只计算前 1000 个网页的相关性就能满足要求。搜索引擎对网页相关性的计算包括:关键词的常用程度;词的频度及密度;关键词的位置;关键词的链接分析及页面权重等内容。经过相关性计算后的排序网页,还需要进行过滤和调整,对于一些有作弊嫌疑的页面,虽然按照正常的权重和相关性计算排名靠前,但搜索引擎可能在最后把这些页面调到后面去。所有排序确定后,搜索引擎将原始页面的标题、说明标签、快照日期等数据发送到用户浏览器。

布尔运算最大的好处是容易实现、速度快,这对于海量信息查找至关重要。它的不足是只能给出是与否的判断,而不能给出量化的度量。因此,所有搜索引擎在内部检索完毕后,都要对符合要求的网页根据相关性排序,然后才返回给用户。

说明:简单搜索引擎的案例,参见本书配套教学资源程序 F3-3.py。

3.2.5　生命游戏:细胞自动机建模

1. 细胞自动机的研究

什么是生命?生命的本质就是可以自我复制、有应激性并且能够进行新陈代谢的机器。每个细胞都是一台自我复制的机器,应激性是对外界刺激的反应,新陈代谢是和外界的物质能量进行交换。冯·诺依曼是最早提出自我复制机器概念的科学家之一。

1948 年,冯·诺依曼在《自动机的通用逻辑理论》论文中,为模拟生物细胞的自我复制提出了“细胞自动机”(也译为元胞自动机)理论。冯·诺依曼对各种人造自动机和天然自动机进行了比较,提出了自动机的一般理论和它们的共同规律,并提出了自繁殖和自修复等理论。但是,冯·诺依曼的细胞自动机理论在当时并未受到学术界的重视。直到 1970 年,剑桥大学何顿·康威(John Horton Conway)教授设计了一个叫作“生命游戏”的计算机程序,美国趣味数学大师马丁·加德纳(Martin Gardner,1914—2010)通过《科学美国人》杂志,将康威的生命游戏介绍给学术界之外的广大读者,生命游戏大大简化了冯·诺依曼的思想,吸

引了各行各业一大批人的兴趣，这时细胞自动机课题才吸引了科学家的注意。

2. 生命游戏概述

生命游戏没有游戏玩家各方之间的竞争，也谈不上输赢，可以把它归类为仿真游戏。在游戏进行中，杂乱无序的细胞会逐渐演化出各种精致、有形的结构。这些结构往往有很好的对称性，而且每一代都在变化形状。一些形状一经锁定，就不会逐代变化。有时，一些已经成形的结构会因为一些无序细胞的“入侵”而被破坏。但是形状和秩序经常能从杂乱中产生出来。在 MATLAB 软件下，输入 life 命令就可以运行“生命游戏”程序。

如图 3-8(a)所示，生命游戏是一个二维网格游戏，这个网格中每个方格居住着一个活着或死了的细胞。一个细胞在下一个时刻的生死取决于相邻 8 个方格中活着或死了细胞的数量。如果相邻方格活着的细胞数量过多，这个细胞会因为资源匮乏而在下一个时刻死去；相反，如果周围活细胞过少，这个细胞会因为孤单而死去(见图 3-8(b))。在游戏初始阶段，玩家可以设定周围活细胞(邻居)的数目和位置。如果邻居细胞数目设定过高，网格中大部分细胞会因为找不到资源而死去，直到整个网格都没有生命；如果邻居细胞数目设定过低，世界中又会因为生命稀少而得不到繁衍。实际中，邻居细胞数目一般选取 2 或者 3，这样整个生命世界才不至于太过荒凉或拥挤，而是一种动态平衡。游戏规则是：当一个方格周围有 2 或 3 个活细胞时，方格中的活细胞在下一个时刻继续存活。即使这个时刻方格中没有活细胞，在下一个时刻也会“诞生”活细胞。在这个游戏中，还可以设定一些更加复杂的规则，例如，当前方格的状态不仅由父代决定，而且还考虑到祖父代的情况。

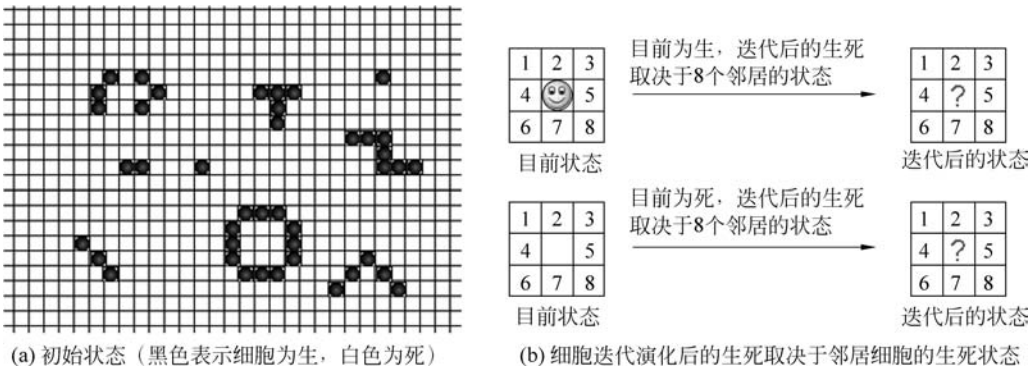


图 3-8　生命游戏是一个二维的细胞自动机

如图 3-8(a)所示，每个方格中都可放置一个生命细胞，每个生命细胞只有两种状态：生或死。在图 3-8(a)的方格网中，用黑色方格表示该细胞为“生”，空格(白色)表示该细胞为“死”。或者说方格网中黑色部分表示某个时候某种“生命”的分布图。生命游戏想要模拟的是：随着时间的流逝，这个分布图将如何一代一代变化。

3. 生命游戏的生存定律

游戏开始时，每个细胞随机地设定为“生”或“死”之一的某个状态。然后，根据某种规则，计算出下一代每个细胞的状态，画出下一代细胞的生死分布图。

应该规定什么样的迭代规则呢？我们需要一个简单而且能够反映生命之间既协同，又竞争的生存定律。为简单起见，最基本的考虑是假设每个细胞都遵循完全一样的生存定律；再进一步，把细胞之间的相互影响只限制在最靠近该细胞的 8 个邻居中，如图 3-8(b)所示。

也就是说，每个细胞迭代后的状态由该细胞及周围 8 个细胞目前的状态所决定。作了这些限制后，仍然还有很多方法来规定“生存定律”的具体细节。例如，在康威的生命游戏中，规定了如下生存定律：

（1）当前细胞为死亡状态时，当周围有 3 个存活细胞时，则迭代后该细胞变成存活状态（模拟繁殖）；若原先为生，则保持不变。

（2）当前细胞为存活状态时，当周围的邻居细胞低于 2 个（不包含 2 个）存活时，该细胞变成死亡状态（模拟生命数量稀少）。

（3）当前细胞为存活状态时，当周围有 2 个或 3 个存活细胞时，该细胞保持原样。

（4）当前细胞为存活状态时，当周围有 3 个以上的存活细胞时，该细胞变成死亡状态（模拟生命数量过多）。

可以把最初的细胞结构定义为种子，当所有种子细胞按以上规则处理后，可以得到第 1 代细胞图。按规则继续处理当前的细胞图，可以得到下一代的细胞图，循环往复。

上面的生存定律当然可以任意改动，发明出不同的“生命游戏”。

4. 生命游戏的迭代演化过程

设定了生存定律之后，根据网格中细胞的初始分布图，就可以决定每个格子下一代的状态。然后，同时更新所有的状态，得到第 2 代细胞的分布图。这样一代一代地迭代下去，以至无穷。如图 3-9 所示，从第 1 代开始，画出了 4 代细胞分布的变化情况。第 1 代时，图中有 4 个活细胞（黑色格子，笑脸为最后一步的连接细胞），然后，读者可以根据上述生存定律，得到第 2、3、4 代的情况，观察并验证图 3-9 的结论。

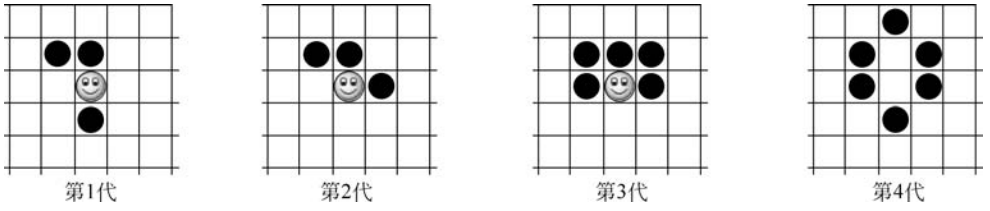


图 3-9　生命游戏中 4 代细胞的演化过程

图 3-10 画出了几种典型的图案演化分布情形。在生命游戏的程序中，随意试验其他一些简单图案就会发现：某些图案经过若干代演化后，会成为静止、振动、运动中的一种，或者是它们的混合物。

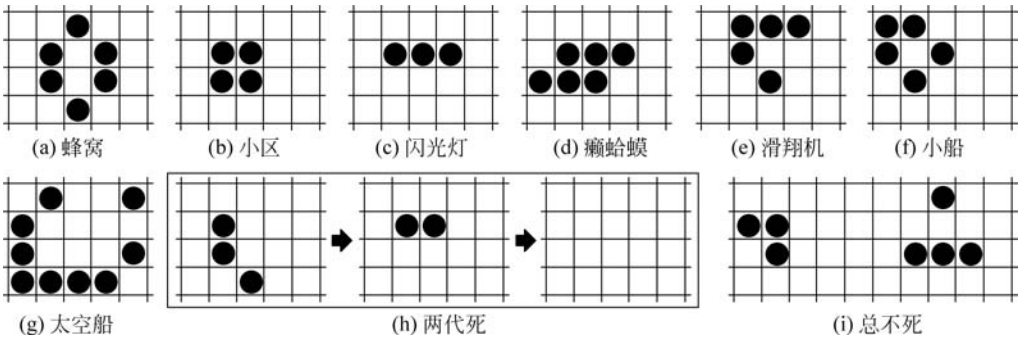


图 3-10　生命游戏中几种特别类型的分布图案

尽管生命游戏中每个细胞所遵循的生存规律都相同，但它们演化形成的图案却各不相同。这又一次说明了一个计算思维的方法："复杂的事物（即使生命）也可以用几条简单的规则表示"。生命游戏提供了一个观察从简单到复杂的方式。

5. 二维细胞自动机数学模型

细胞自动机不是严格定义的数学方程或函数，细胞自动机的构建没有固定的数学公式，而是由一系列规则构成。凡是满足这些规则的模型都可以称为细胞自动机模型。细胞自动机在时间、空间、状态上，都采用离散变量，每个变量只取有限个状态，而且只在时间和空间的局部进行状态改变。

（1）细胞自动机数学模型。细胞自动机（A）由细胞、细胞状态、邻域和状态更新规则构成，数学模型为

$$A=(L_d,S,N,f) \tag{3-4}$$

其中，L 为细胞空间；d 为细胞空间的维数；S 是细胞有限的、离散的状态集合；N 为某个邻域内所有细胞的集合；f 为局部映射或局部规则。

一个细胞在一个时刻只取一个有限集合的一种状态，如{0，1}。细胞状态可以代表个体的态度、特征、行为等。空间上与细胞相邻的细胞称为邻元，所有邻元组成邻域。

（2）二维细胞自动机数学模型。生命游戏是一个二维细胞自动机。二维细胞自动机的基本空间是二维直角坐标系，坐标为整数的所有网格集合，每个细胞都在某一网格中，可以用坐标(a,b)表示1个细胞。每个细胞只有0或1两种状态，其邻居是由$(a\pm1,b)$、$(a,b\pm1)$、$(a\pm1,b\pm1)$和(a,b)共9个细胞组成的集合。状态函数 f 用图示法表示时如图3-11所示。

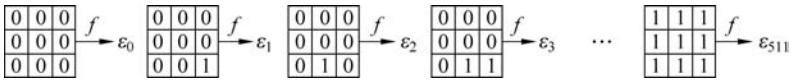


图3-11　二维细胞自动机的局部规则

从左上角先水平后竖直，到右下角给9个位置排序，状态函数 f 可以用：$f(000000000)=\varepsilon_0$，$f(000000001)=\varepsilon_1$，$f(000000010)=\varepsilon_2$，…，$f(111111111)=\varepsilon_{511}$ 表示。

注意：每个等式对应于图3-11中的一个框图，如 $f(111111111)=\varepsilon_{512}$ 对应于图3-11最后的框图，$f(111111111)$的函数值为 ε_{511}，其中 ε_{511} 或者为1或者为0。

图3-12　细胞 X_0 的邻居细胞

设 $t=0$ 时刻的初始配置是 C_0，对任一细胞 X_0，假设邻域内细胞的状态如图3-12所示。

那么，这个细胞在 $t=1$ 时的状态就是 $f(X_0,X_1,X_2,X_3,X_4,X_5,X_6,X_7,X_8)$。所有细胞在 $t=1$ 时的状态都可以用这种方法得到，合在一起就是 $t=1$ 时刻的配置 C_1。并依次可得到 $t=2$，$t=3$，…时的配置 C_2，C_3，…。

从数学模型的角度看，可以将生命游戏模型划分成网格棋盘，每个网格代表一个细胞。网格内的细胞状态为：0=死亡，1=生存；细胞的邻居半径为：$r=1$；邻居类型=Moore（摩尔）型。则二维生命游戏的数学模型为

$$\text{如果 } S_t=1,\text{则 } S_{t+1}=\begin{cases}1, & S=2,3\\ 0, & S\neq 2,3\end{cases} \tag{3-5}$$

$$\text{如果 } S_t=0,\text{则 } S_{t+1}=\begin{cases}1, & S=3\\ 0, & S\neq 3\end{cases} \tag{3-6}$$

其中，S_t 表示 t 时刻细胞的状态；S_{t+1} 表示 $t+1$ 时刻细胞的状态；S 为邻居活细胞数。

6. 康威“生命游戏”算法步骤

康威“生命游戏”算法步骤如下：

(1) 对本细胞周围的 8 个近邻的细胞状态求和。

(2) 如果邻居细胞总和为 2，则本细胞下一时刻的状态不改变。

(3) 如果邻居细胞总和为 3，则本细胞下一时刻的状态为 1；否则状态=0。

假设细胞为 c_i，它在 t 时刻的状态为(s_i, t)，它两个邻居的状态为(s_i-1, t)，(s_i+1, t)，则细胞 c_i 在下一时刻的状态为$(s_i, t+1)$，可以用函数表示为

$$(s_i,t+1)=f((s_i-1,t),(s_i,t),(s_i+1,t)),\quad (s_i,t)\in\{0,1\}$$

7. 细胞自动机的应用

细胞自动机是一种动态模型，它可以作为一种通用性建模的方法。研究内容包括：信息传递、构造、生长、复制、竞争与进化等。同时，它为动力学系统理论中，有关秩序、紊动、混沌、非对称、分形等系统整体行为与复杂现象的研究，提供了一个有效的模型。细胞自动机的应用几乎涉及社会科学和自然科学的各个领域。

说明：细胞自动机的案例，参见本书配套教学资源中的程序 F3-4.py。

3.3 计算科学基础：可计算性

计算科学的基础理论包括：计算理论(可计算性理论、复杂性理论、算法设计与分析、计算模型等)、离散数学(集合论、图论与树、数论、离散概率、抽象代数、布尔代数)、程序语言理论(形式语言理论、自动机理论、形式语义学、计算语言学等)、人工智能(机器学习、模式识别、知识工程、机器人等)、逻辑基础(数理逻辑、多值逻辑、模糊逻辑、组合逻辑等)、数据库理论(关系理论、关系数据库、NoSQL 数据库等)、计算数学(符号计算、数学定理证明、计算几何等)、并行计算(分布式计算、网格计算等)。

3.3.1 图灵机计算模型

1. 图灵机的基本结构

1936 年，年仅 24 岁的图灵发表了《论可计算数及其在判定问题中的应用》论文。论文中，图灵构造了一台抽象的“计算机”，科学家称它为“图灵机”。图灵机是一种结构十分简单但计算能力很强的计算模型，它可以用来计算所有能想象到的可计算函数。

如图 3-13 所示，图灵机由控制器(P)、指令表(I)、读写头(R/W)、存储带(M)组成。其中，存储带是一个无限长的带子，可以左右移动，带子上划分了许多单元格，每个单元格中包含一个来自有限字母表的符号。控制器中包含了一套指令表(控制规则)和一个状态寄存器，指令表就是一个图灵机程序，状态寄存器则记录了机器当前所处的状态，以及下一个新

状态。读写头则指向存储带上的格子，负责读出和写入存储带上的符号，读写头有写 1、写 0、左移、右移、改写、保持、停机 7 种行为状态(有限状态机)。

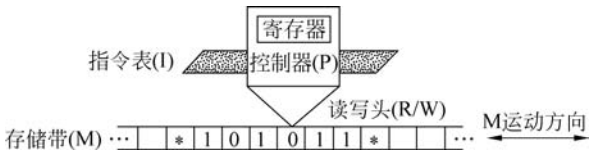


图 3-13　图灵机的基本结构

图灵机的工作原理是：存储带每移动一格，读写头就读出存储带上的符号，然后传送给控制器。控制器根据读出的符号以及寄存器中机器当前的状态(条件)，查询应当执行程序的那一条指令，然后根据指令要求，将新符号写入存储带(动作)，以及在寄存器中写入新状态。读写头根据程序指令改写存储带上的符号，最终计算结果就在存储带上。

2. 图灵机的特点

在上面案例中，图灵机使用了 0、1、* 等符号，可见图灵机由有限符号构成。如果图灵机的符号集有 11 个符号，如{0,1,2,3,4,5,6,7,8,9,*}，那么图灵机就可以用十进制来表示整数值。但这时的程序要长得多，确定当前指令要花更多的时间。符号表中的符号越多，用机器表示的困难就越大。

图灵机可以依据程序对符号表要求的任意符号序列进行计算。因此，同一个图灵机可以进行规则相同、对象不同的计算，具有数学上函数 $f(x)$的计算能力。

如果图灵机初始状态(读写头的位置、寄存器的状态)不同，那么计算的含义与计算的结果就可能不同。每条指令进行计算时，都要参照当前的机器状态，计算后也可能改变当前的机器状态。而状态是计算科学中非常重要的一个概念。

在图灵机中，虽然程序按顺序来表示指令序列，但是程序并非顺序执行。因为指令中关于下一状态的指定说明了指令可以不按程序的顺序执行。这意味着，程序的三种基本结构——“顺序、判断、循环”在图灵机中得到了充分体现。

3. 通用图灵机

专用图灵机将计算对象、中间结果和最终结果都保存在存储带上，程序保存在控制器中(程序和数据分离)。由于控制器中的程序是固定的，那么专用图灵机只能完成规定的计算(输入可以多样化)。

存在一台图灵机能够模拟所有其他图灵机吗？答案是肯定的，能够模拟其他所有图灵机的机器称为“通用图灵机”。通用图灵机可以把程序放在存储带上(程序和数据混合在一起)，而控制器中的程序能够将存储带上的指令逐条读进来，再按照要求进行计算。

通用图灵机一旦能够把程序作为数据来读写，就会产生很多有趣的情况。首先，会有某种图灵机可以完成自我复制，例如，计算机病毒就是这样。其次，假设有一大群图灵机，让它们彼此之间随机相互碰撞。当碰到一块时，一个图灵机可以读入另一个图灵机的编码，并且修改这台图灵机的编码，那么在这个图灵机群中会产生什么情况呢？圣塔菲研究所的实验得出了惊人的结论：在这样的系统中，会诞生自我繁殖的、自我维护的、类似生命的复杂组织，而且这些组织能进一步联合起来构成更大的组织。

4. 图灵机的重大意义

图灵机不是一台具体的机器，它是一种理论思维模型。图灵机完全忽略了计算机的硬

件特征，考虑的核心是计算机的逻辑结构。图灵机的内存是无限的，而实际机器的内存是有限的，所以图灵机并不是实际机器的准确模型（图灵本人也没有给出图灵机结构图），图灵机模型也并没有直接带来计算机的发明。但是图灵机具有以下重大意义。

（1）图灵机证明了通用计算理论，肯定了计算机实现的可能性。图灵机可以分析什么是可计算的，什么是不可计算的。一个问题能不能解决，在于能不能找到一个解决这个问题的算法，然后根据这个算法编制程序在图灵机上运行，如果图灵机能够在有限步骤内停机，则这个问题就能解决。如果找不到这样的算法，或者这个算法在图灵机上运行时不能停机，则这个问题无法用计算机解决。图灵指出：**“凡是能用算法解决的问题，也一定能用图灵机解决；凡是图灵机解决不了的问题，任何算法也无法解决”**。

（2）图灵机模型引入了读/写、算法、程序语言等概念，极大突破了过去计算机器的设计理念。通用图灵机与现代计算机的相同之处是：程序可以和数据混合在一起。图灵机与现代计算机的不同之处在于：图灵机的内存无限大，并且没有考虑输入和输出设备（所有信息都保存在存储带上）。

（3）图灵机模型是计算科学最核心的理论，因为计算机的极限能力就是图灵机的计算能力，很多复杂的理论问题都可以转换为图灵机来考虑。进行理论研究时，一般以图灵机为计算模型，因为它能够模拟实际计算机中的所有计算行为，并且足够简单明确。

5. 图灵完备的编程语言

“图灵完备性”用来衡量某个计算模型的计算能力，如果一个计算模型具有和图灵机同等的计算能力，它就可以称为是图灵完备的。简单的方法是看该程序语言能否模拟出图灵机，非图灵完备编程语言一般没有判断（if）、循环（for）、计算等指令（如 HTML、正则表达式等），或者是这些指令功能很弱（如 SQL 语言等），因此无法模拟出图灵机。绝大部分编程语言都是图灵完备的，如 C/C++、Java、Python 等；也有少部分编程语言是非图灵完备的，如 HTML、正则表达式、SQL 等。具有图灵完备性的语言不一定都有用（如 Brainfuck 语言），而非图灵完备的编程语言也不一定没有用（如 HTML、SQL、正则表达式等）。程序语言并不需要与图灵机完全等价的计算能力，只要程序语言能够完成某种计算任务，它就是一种有用的程序语言。

3.3.2 停机问题：理论上不可计算的问题

1. 什么是停机问题

停机问题是：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，可达到停机状态。若能找到这种算法，停机问题可解；否则不可解。通俗地说，停机问题就是：能不能编写一个用于检查并判定另一个程序是否会运行结束的程序呢？图灵在 1936 年证明了：解决停机问题不存在通用算法。

【例 3-37】 用反证法证明“停机问题”超出了图灵机的计算能力，无算法解。

如图 3-14(a)所示，设计一个停机程序 T，在 T 中可以输入外部程序，并对外部程序进行判断。如果输入的程序是自终止的，则程序 T 中的 x=1；如果输入的程序不能自终止，则程序 T 中的 x=0。如图 3-14(b)所示，修改停机程序为 P，程序 P=停机程序 T+循环结构。

如图 3-14(c)所示，在图灵机中运行程序 P，并将程序 T 自身作为输入；如果输入的外部程序为自终止程序，则 x=1，程序 P 陷入死循环。这导致悖论 A：自终止程序不能停机。

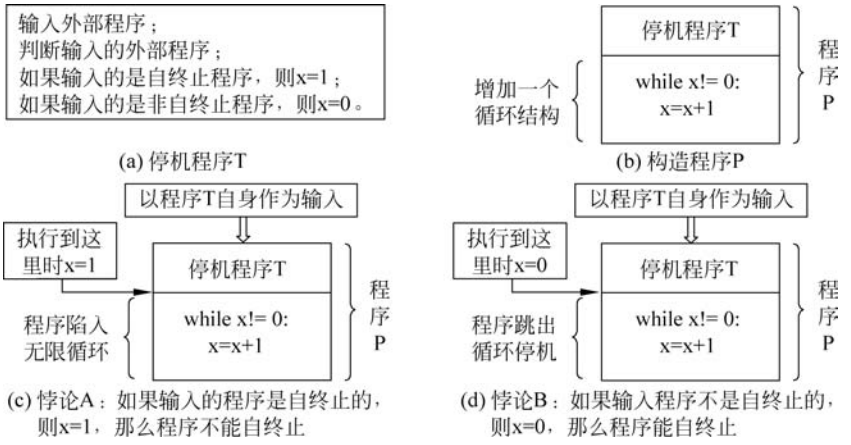


图 3-14　简要证明停机程序悖论的示意图

如图 3-14(d)所示，如果输入的外部程序为非自终止程序，则 x＝0，这时程序 P 在循环条件判断后，结束循环进入停机状态。这导致悖论 B：不能自终止的程序可以停机。

以上结论显然自相矛盾，因此，可以认为停机程序是不可计算的。

【例 3-38】 以下用一个更加通俗易懂的案例来说明停机问题。假设 A 是一个万能的程序，那么 A 自然可以构造一个与 A 相反的程序 B。如果 A 预言程序会停机，则 B 就预言程序不会停机；如果 A 预测程序不会停机，则 B 就预言程序会停机。这样 A 能判断 B 会停机吗？这显然是不能的，这也就是说程序 A 和 B 都不存在。

停机程序悖论(逻辑矛盾)在于存在“指涉自身”的问题。简单地说，人们不能判断匹诺曹(《木偶奇遇记》中的主角)说“我在说谎”这句话时，他的鼻子是否会变长。

停机问题说明了计算机是一个逻辑上不完备的形式系统。计算机不能解一些问题并不是计算机的缺点，因为停机问题本质上是不可解的。

2. 停机问题的实际意义

是否存在一个程序能够检查所有其他程序会不会出错？这是一个非常实际的问题。为了检查程序的错误，必须对这个程序进行人工检查。那么能不能发明一种聪明的程序，输进去任何一段其他程序，这个程序就会自动帮你检查输入的程序是否有错误？这个问题被证明和图灵停机问题实质上相同，不存在这样的聪明程序。

图灵停机问题与复杂系统的不可预测性有关。我们总希望能够预测出复杂系统的运行结果。那么能不能发明一种聪明程序，然后输入某些复杂系统的规则，然后输出这些规则运行的结果呢？从原理上讲，这种事情是不可能的，因为它与图灵停机问题等价。因此，要想弄清楚某个复杂系统运行的结果，唯一的办法就是让此类系统实际运作，没有任何一种算法能够事先给出这个系统的运行结果。

以上强调的是**不存在一个通用程序能够预测所有复杂系统的运行结果**，但并没有说不存在一个特定程序能够预测某个或者某类特定复杂系统的结果。那怎么得到这种特定的程序呢？这就需要人工编程，也就是说存在某些机器做不了的事情，而人能做。

3.3.3　汉诺塔：现实中难以计算的问题

法国数学家爱德华·卢卡斯(Édouard Anatole Lucas)曾编写过一个神话：印度教天神

汉诺(Hanoi)在创造地球时建了一座神庙,神庙里竖有三根宝石柱子,柱子由一个铜座支撑。汉诺将 64 个直径大小不一的金盘子按照从大到小的顺序依次套放在第一根柱子上,形成一座金塔(即汉诺塔)。天神让庙里的僧侣们将第一根柱子上的 64 个盘子借助第二根柱子全部移到第三根柱子上,即将整个塔迁移,同时定下了三条规则:一是每次只能移动一个盘子;二是盘子只能在三根柱子上来回移动,不能放在他处;三是在移动过程中,三根柱子上的盘子必须始终保持大盘在下,小盘在上。汉诺塔问题全部可能的状态数为 3^n 个(n 为盘子数),最少搬动次数为 2^n-1。

【例 3-39】 只有 3 个盘子的汉诺塔问题解决过程如图 3-15 所示。3 个盘子的最佳搬移次数为 $2^3-1=7$ 次。

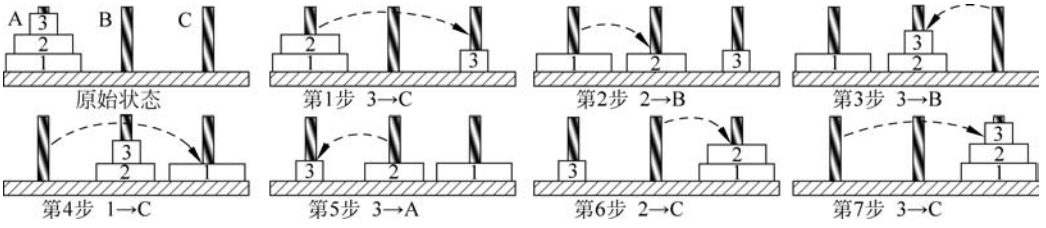


图 3-15 3 个盘子时汉诺塔的解题过程

如果汉诺塔 3 个柱子名为 A、B、C,n 个盘子递归求解的 Python 程序如下:

```
1  # E0339.py                                  # 【复杂性 - 汉诺塔】
2  def hanoi(n, a, b, c):                     # 定义递归函数,n 为盘子数,a,b,c 为柱子名
3      if n == 1:                             # 如果只有一个盘子
4          print(a,'-->', c)                  # 将盘子从 a 柱移到 c 柱
5      else:                                  # 否则
6          hanoi(n-1, a, c, b)                # 递归,将 a 柱 n-1 号盘子移到 b 柱,c 柱为辅助柱
7          hanoi(1, a, b, c)                  # 递归,将 a 柱最底层最大的盘子移到 c 柱
8          hanoi(n-1, b, a, c)                # 递归,将 b 柱 n-1 号盘子移到 c 柱,a 柱为辅助柱
9  n = int(input('请输入汉诺塔的盘子数:'))    # 接收用户输入的数据,并转换为整数
10 hanoi(n, 'a', 'b', 'c')                    # 调用递归函数 hanoi( )
```

```
>>>请输入汉诺塔的盘子数:3                    # 程序运行结果(运算时间呈指数增长,测试
a-->c a-->b c-->b a-->c...(输出略)           # 表明,15 个盘子的运算时间约为 5 分钟)
```

汉诺塔递归求解程序虽然非常简单,但是难以理解,需要反复琢磨。汉诺塔有 64 个盘子时,盘子最少移动次数为 $2^{64}-1=1.8\times10^{19}$。由此可知,**并不是所有问题都可以计算,即使是可计算问题,也要考虑计算量是否超过了目前计算机的计算能力**。

说明:汉诺塔动画案例,参见本书的配套教学资源程序 F3-3.py。

3.3.4 不完备性与可计算性

计算理论研究的三个核心领域为自动机、可计算性和复杂性。通过"计算机的基本能力和局限性是什么?"这一问题将这三个领域联系在一起。在计算复杂性理论中,将问题分成容易计算和难计算。在可计算理论中,把问题分成可解和不可解。

1. 哥德尔不完备性定理

20 世纪以前,大部分数学家认为所有问题都有算法,关于计算问题的研究就是找出解

决各类问题的算法。1928 年，德国著名数学家戴维·希尔伯特(David Hilbert)提出一个问题：是否有一个算法能对所有的数学原理自动给予证明。这个美好的希望不久就被打破了，1931 年，奥地利数学家哥德尔(Kurt Gödel)给出了证明：任何无矛盾的公理体系，只要包含初等数论，则必定存在一个不可判定命题，用这组公理不能判定其真假；或者说：一个理论如果不自相矛盾，那么这种性质在该理论中不可证明。

哥德尔不完备性定理说明，**任何一个形式系统，它的一致性和完备性不可兼得**。或者说，如果一个形式系统是一致的，那么这个系统必然是不完备的，二者不可兼得。

不完备性也可以用一个古老的哲学观点“一个人不可能完全理解自身”进行验证：如果你能做到完全理解自身，你就可以预知自己在未来十秒钟内做什么，而刻意做出不符合预测的行为就可以推翻“你能够完全理解自身”这个前提。

2. 图灵机的不完备性

哥德尔不完备性定理说明，在任何一个数学系统内，总有一些命题的真伪无法通过算法来确定。总是存在这样的函数，由于过于复杂以致没有严格定义的、逐步计算的过程能够根据输入值来确定输出值。也就是说，这种函数的计算超出了任何算法的能力。

图灵提出图灵机模型后也发现了有些问题图灵机无法计算。例如，定义模糊的问题，如“人生有何意义”；或者缺乏数据的问题，如“明天彩票的中奖号是多少”，它们的答案无法计算出来；还有一些定义完美的问题也是不可计算的，如“停机问题”。

3. 什么是可计算性

物理学家阿基米德曾经宣称：“给我足够长的杠杆和一个支点，我就能撬动地球”。在数学上也同样存在类似的问题：是不是只要给数学家足够长的时间，通过“有限次”简单而机械的演算步骤，就能够得到最终答案呢？这就是“可计算性”问题。

什么是可计算的？什么又是不可计算的呢？要回答这一问题，关键是要给出“可计算性”的精确定义。20 世纪 30 年代，一些著名数学家和逻辑学家从不同角度分别给出了“可计算性”概念的确切定义，为计算科学的发展奠定了重要基础。

可计算的问题都可以通过自然数编码的方法，用函数的形式表示。因此可以通过定义在自然数集上的“直观可计算函数”(也称为一般递归函数)来理解“可计算性”的概念。凡是从某些初始符号串开始，在有限步骤内得到计算结果的函数都是直观可计算函数。

1935 年，丘奇(Alonzo Church)为了定义可计算性，提出了 λ 演算理论。丘奇认为，λ 演算可定义的函数与直观可计算函数相同。

1936 年，哥德尔、丘奇等定义了递归函数。丘奇指出：一切直观可计算函数都是递归函数。简单地说，计算就是符号串的变换。凡是可以从某些初始符号串开始，在有限步骤内可以得到计算结果的函数都是一般递归函数。可以从简单的、直观上可计算的一般函数出发，构造出复杂的可计算函数。1936 年，图灵也提出：图灵机可计算函数与直观可计算函数相同。

一个显而易见的事实是：数学精确定义的直观可计算函数都是可计算的。问题是直观可计算函数是否恰好就是这些精确定义的可计算函数呢？对此丘奇认为：凡直观可计算函数都是 λ 可定义的；图灵证明了图灵可计算函数与 λ 可定义函数是等价的，著名的“丘奇-图灵论题”(丘奇是图灵的老师)认为：**任何能直观计算的问题都能被图灵机计算。如果证明了某个问题使用图灵机不可计算，那么这个问题就是不可计算的**。

由于直观可计算函数不是一个精确的数学概念，因此丘奇-图灵论题不能加以证明，也因此称为“丘奇-图灵猜想”，该论题被普遍假定为真。

4. 哪些数不可计算？

如何判断一个数是否可以计算？图灵在一篇论文中提出：“当一个实数所有的位数，包括小数点后的所有位，都可以在有限步骤内用某种算法计算出来，它就是可计算数；如果一个实数不是可计算数，那它就是不可计算数”。例如，圆周率 π 可以在有限时间内计算到小数点后的任何位置，所以 π 是可计算数。

1975 年，计算科学专家格里高里·蔡廷(Gregory Chaitin)提出了一个有趣的问题：选择任意一种编程语言，随机输入一段代码，这段代码能成功运行并且会在有限时间里终止(不会无限运行下去)的概率是多大？他把这个概率值命名为“蔡廷常数”。目前已经在理论上证明了，永远无法求出“蔡廷常数”的值。

理论上，任何一段程序的运行结果，只有终止和永不终止两种情况。可终止程序的数量一定占有全体程序总数的一个固定比例，所以这个停机概率一定存在，它的上限为 1，下限为 0。在真实环境中，任意编程语言书写的程序有无穷多个，要想用统计方法得到这个概率值，需要的程序测试也是无穷多次，显然蔡廷常数是一个不可计算数。

5. 不可计算问题的类型

不可计算的问题有以下类型：第一类是理论意义上不可计算问题，由丘奇-图灵论题确定的所有非递归函数都是不可计算的。具体问题有：停机问题、蔡廷常数、彭罗斯拼图(Penrose Diagram)、蒂博尔·拉多(Tibor Radó)提出的忙碌海狸函数 BB(n)、不定方程的整数解(希尔伯特第 10 个问题，也称为丢番图方程)等，这些问题都是计算能力的理论限制。第二类是现实计算机的速度和存储空间都有一定限制，如某个问题在理论上可计算，但是计算时间如果长达几百年，那么这个问题实际上还是无法计算。如汉诺塔问题、长密码破解问题、旅行商问题、大数因式分解问题等。第三类是现实意义上难以计算的问题，如“西红柿炒鸡蛋”这样一个概念模糊的命题也不可计算。

3.3.5 P=NP？计算科学难题

1. P 问题

有些问题是确定性的，如加、减、乘、除计算，只要按照公式推导，就可以得到确定的结果，这类问题是 P 问题。**P(Polynomially，多项式)问题是指算法能在多项式时间内找到答案的问题**，这意味着计算机可以在有限的时间内完成计算。

对一个规模为 n 的输入，最坏情况下(穷举法)，P 问题求解的时间复杂度为 $O(n^k)$。其中 k 是某个确定的常数，我们将这类问题定义为 P 问题，直观上，P 问题是在确定性计算模型下的易解问题。P 问题有：计算最大公约数、排序问题、图搜索问题(在连通图中找到某个对象)、单源最短路径问题(两个节点之间的最短路径)、最小生成树问题(连通图中所有边权重之和最小的树)等，这些问题都能在多项式时间内解决。

2. NP 问题

NP(Nondeterministic Polynomially，非确定性多项式)问题中的 N 指非确定的算法。有些问题是非确定性问题，如寻找大素数问题，目前没有一个现成的推算素数的公式，需要用穷举法进行搜索，这类问题就是 NP 问题。**NP 问题不能确定是否能够在多项式时间内找**

到答案，但是可以确定在多项式时间内验证答案是否正确。

【例 3-40】 验证某个连通图中的一条路径是不是哈密尔顿(Hamilton)回路很容易，而问某个连通图中是否不存在哈密尔顿回路则非常困难，除非穷举所有可能的哈密尔顿路径，否则无法验证这个问题的答案，因此这是一个 NP 问题。

3. NPC 问题

如果任何一个 NP 问题能通过一个多项式时间算法转换为某个其他 NP 问题，那么这个 NP 问题就称为 NPC(NP-Complete，NP 完全)问题。NPC 是比 NP 更难的问题。

【例 3-41】 对任意的布尔可满足性问题(SAT)，总能写成以下形式：

$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee x_5 \vee x_n) \quad \cdots$$

其中，$\vee$ 表示逻辑或运算；$\wedge$ 表示逻辑与运算；表达式中 x 的值只能取 0 或者 1。那么当 x 取什么值时，这个表达式为真？或者根本不存在一个取值使表达式为真？这就是 SAT 问题，任意 SAT 是一个典型的 NPC 问题。

一个简单的 SAT 问题，如：$(x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2) \wedge (\overline{x_2} \vee x_3)$，当 $x_1=0$，$x_2=0$，$x_3=$任意值(0 或 1)时，表达式的值为真，$\overline{\ }$表示逻辑取反运算。

NPC 问题目前没有多项式算法，只能用穷举法逐一检验，最终得到答案。但是穷举算法的复杂性为指数关系，计算时间随问题的复杂程度成指数级增长，很快问题就会变得不可计算了。目前已知的 NPC 问题有 3000 多个，如布尔可满足性问题、国际象棋 N 皇后问题、密码学中大素数分解问题、多核 CPU 流水线调度问题、哈密尔顿回路问题、旅行商问题(Travelling Salesman Problem，TSP)、最大团问题、最大独立集合问题、背包问题等。

4. NP-hard 问题

除 NPC 问题外，还有一些问题连验证解都不能在多项式时间内解决，这类问题被称为 NP-hard(NP 难)问题。NP-hard 太难了，围棋或象棋的博弈问题；怎样找到一个完美的女朋友等，都是 NP-hard 问题。计算科学中难解问题之间的关系如图 3-16 所示。

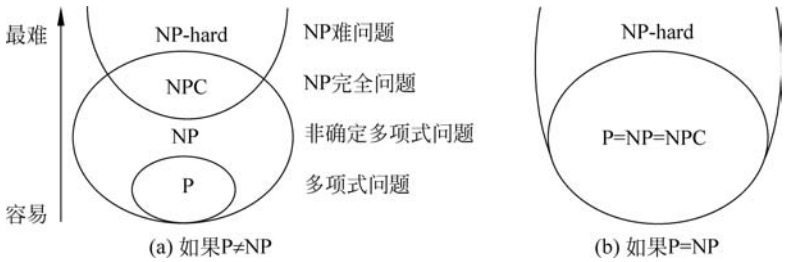


图 3-16 计算科学中的难解问题

【例 3-42】 棋类博弈问题。考察棋局所有可能的博弈状态时，国际跳棋有 10^{31} 种博弈状态，香农在 1950 年估计出国际象棋的博弈状态大概有 10^{120} 种；中国象棋的博弈状态估计有 10^{150} 种；围棋的博弈状态达到了惊人的 10^{360} 种。事实上大部分棋类的计算复杂度都呈指数级上升。作为比较，目前可观测到宇宙中的原子总数估计有 $10^{78}\sim10^{82}$ 个。

5. P=NP？问题

直观上看计算复杂性时：P<NP<NPC<NP-hard，问题的难度不断递增。但是目前只证明了 P 属于 NP，究竟 P=NP？还是 P≠NP？迄今为止这个问题还没有找到一个有效的

证明。P=NP 是一个既没有证实，也没有证伪的命题。

计算科学专家认为：**找一个问题的解很困难，但验证一个解很容易（证比解易）**，用数学公式表示就是 P≠NP。问题难于求解，易于验证，这与人们的日常经验相符。因此，人们倾向于接受 P≠NP 这一猜想。

6. 不可计算问题的解决方法

无论是理论上不可计算还是现实中难以计算的问题，都是指无法得到公式解、解析解、精确解或最优解，但是这并不意味着不能得到近似解、概率解、局部解或弱解。计算科学专家对待理论上或现实中不可解的问题时，通常采取两个策略：一是不去解决一个过于普遍的问题，而是通过弱化有关条件，将问题限制得特殊一些，再解决这个普遍问题的一些特例或范围窄小的问题；二是寻求问题的近似算法、概率算法等。也就是说，对于不可计算或不可判定的问题，人们并不是束手无策，而是可以从计算的角度有所作为。

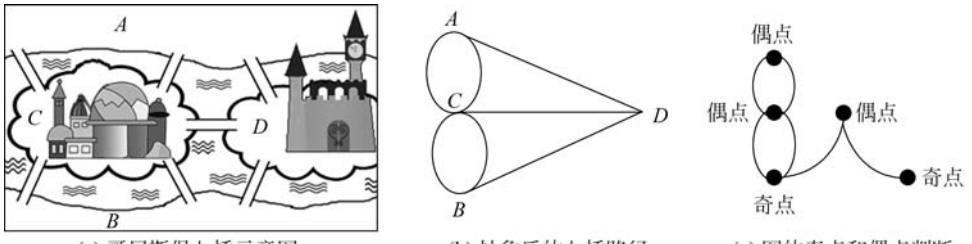
3.4 学科经典问题：计算复杂性

计算科学发展中，人们提出过许多具有深远意义的问题和典型实例。这些典型问题的研究不仅有助于我们深刻地理解计算科学，而且对学科的发展有十分重要的推动作用。计算科学的经典问题除停机问题、汉诺塔问题、中文屋子问题外，还有哥尼斯堡七桥问题、哈密尔顿回路问题、旅行商问题、哲学家就餐问题、两军通信问题等。

3.4.1 哥尼斯堡七桥问题：图论

18 世纪初，普鲁士的哥尼斯堡（今俄罗斯加里宁格勒）有一条河穿过，河上有两个小岛，有七座桥把两个岛与河岸联系起来，如图 3-17(a)所示。有哥尼斯堡市民提出了一个问题：一个步行者怎样才能不重复、不遗漏地一次走完七座桥，最后回到出发点？问题提出后，很多哥尼斯堡市民对此很感兴趣，纷纷进行试验，但在相当长的时间里都始终未能解决。从数学知识来看，七座桥所有的走法共有 7!=5040 种，这么多种走法要一一尝试，将会是一个很大的工作量。

1735 年，有人写信给当时在俄罗斯彼得堡科学院任职的瑞士数学家欧拉（Leonhard Euler），请他帮忙解决这一问题。欧拉把哥尼斯堡七桥问题抽象成几何图形，如图 3-17(b)所示，他圆满地解决了这个难题，同时开创了“图论”的数学分支。欧拉把一个实际问题抽象成合适的“数学模型”，这并不需要深奥的理论，但想到这一点却是解决难题的关键。



(a) 哥尼斯堡七桥示意图　(b) 抽象后的七桥路径　(c) 图的奇点和偶点判断

图 3-17　哥尼斯堡七桥问题

如果图中存在一条路径，经过图中每条边一次且仅一次，则该路径称为欧拉回路，具有欧拉回路的图称为欧拉图。欧拉回路的“边”不能重复经过，但是“顶点”可以重复经过。欧拉回路可以用“一笔画”来说明，要使图形一笔画出，就必须满足以下条件。

(1) 全部由偶点组成的连通图，选任一偶点为起点，可以一笔画成此图。

(2) 只有 0 或 2 个奇点，其余为偶点的连通图，以奇点为起点可以一笔画成此图。

(3) 其他情况的图不能一笔画出，奇点数除以 2 可以算出此图需几笔画成。

注：如图 3-17(c)所示，顶点的边是奇数称为“奇点”；反之称为“偶点”。

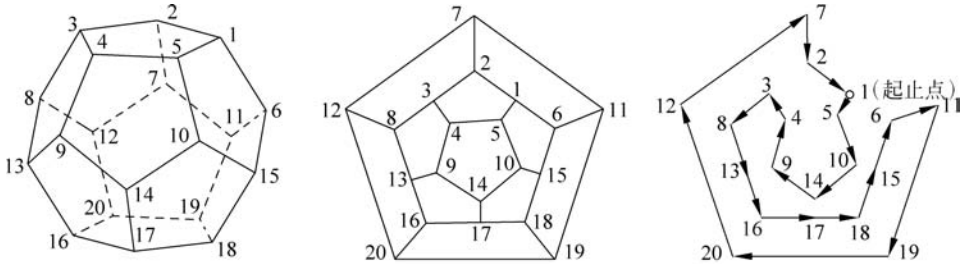
由以上分析可见，哥尼斯堡七桥的路径图中(见图 3-17(b))，4 个点全是奇点，因此图形不能一笔画出(最少需要 2 笔画)。**欧拉回路是从起点一笔画到终点的路径集合**，由此可见哥尼斯堡七桥不存在欧拉回路(不是欧拉图)。

图论中的图由若干点和边构成，一系列由边连接起来的点称为路径。图论常用来研究事物之间的联系。一般用点代表事物，用边表示两个事物之间的联系，如互联网中的节点与线路、社交网络中明星与粉丝之间的关系、电路中的节点与电流、旅行商问题中的城市与道路等，都可以用图论进行分析和研究。在计算科学领域，有各种各样的图论算法，如深度优先遍历、广度优先遍历、图最短路径、图最小生成树、网络最大流等。

说明：哥尼斯堡程序案例，参见本书配套教学资源程序 F3-6.py。

3.4.2 哈密尔顿回路：计算复杂性

1857 年，爱尔兰数学家哈密尔顿(William Rowan Hamilton)设计了一个名为“周游世界”的木制玩具(见图 3-18(a))，玩具是一个正 12 面体，有 20 个顶点和 30 条边。如果将周游世界的玩具抽象为一个二维平面图(见图 3-18(b))，图中每个顶点看作一个城市，正 12 面体的 30 条边看成连接这些城市的路径。假设从某个城市出发，经过每个城市(顶点)恰好一次，最后又回到出发点(见图 3-18(c))，这就形成了一条哈密尔顿回路。



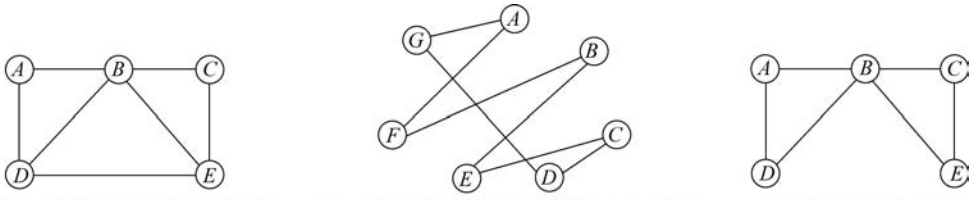
(a) 哈密尔顿“周游世界”玩具　(b) 抽象后的平面图　(c) 一条哈密尔顿回路

图 3-18　哈密尔顿图和哈密尔顿回路

哈密尔顿回路与欧拉回路看似相同，但本质上是完全不同的问题。哈密尔顿回路是访问每个顶点一次，欧拉回路是访问每条边一次；哈密尔顿回路的边和顶点都不能重复通过，但是有些边可以不经过，欧拉回路的边不能重复通过，顶点可以重复通过。如图 3-17(b)中不存在欧拉回路，但是存在多个哈密尔顿回路(如 $A \rightarrow C \rightarrow B \rightarrow D \rightarrow A$)。

如果经过图中的每个顶点恰好一次后能够回到出发点，这样的图称为哈密尔顿图。所经过的闭合路径就形成了一个圈，它称为哈密尔顿回路或者哈密尔顿圈，如图 3-19(a)、

图 3-19(b)所示。图中一个顶点的度数是指这个顶点所连接的边数，图 3-19(a)中，顶点 B 的度为 4。哈密尔顿图有很多充分条件，例如，图的最小度不小于顶点数的一半时，则此图是哈密尔顿图；若图中每对不相邻顶点的度数之和不小于顶点数，则为哈密尔顿图。也有一些图中不存在哈密尔顿回路，如图 3-19(c)所示。



(a) 一条哈密尔顿回路={A,B,C,E,D,A}　(b) 一条哈密尔顿回路={A,G,D,C,E,B,F,A}　(c) 没有哈密尔顿回路

图 3-19　哈密尔顿回路(哈密尔顿圈)

目前还没有找到判断一个图是不是哈密尔顿图的充分和必要条件。寻找一个图的哈密尔顿回路是 NP 难问题。通常用穷举搜索的方法来判定一个图中是否含有哈密尔顿回路。目前还没有找到哈密尔顿回路的多项式算法。

欧拉回路和哈密尔顿回路在任务排队、内存碎片回收、并行计算等领域均有应用。

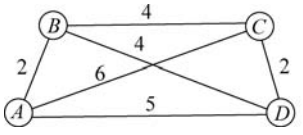
说明：哈密尔顿回路案例，参见本书配套教学资源程序 F3-7.py。

3.4.3　旅行商问题：计算组合爆炸

旅行商问题是哈密尔顿和英国数学家柯克曼(T. P. Kirkman)提出的问题，它是指有若干城市，任何两个城市之间的距离可能不同或相同，现在一个旅行商从某一个城市出发，依次访问每个城市一次，最后回到出发地，问如何行走路程最短。

旅行商问题的“边”和“顶点”都不能重复通过，但是有些“边”可以不经过，因此旅行商问题与哈密尔顿回路有相似性；不同的是旅行商问题增加了边长的权值。

解决旅行商问题的基本方法是：对给定的城市路径进行排列组合，首先列出所有的路径，然后计算出每条路径的总里程，最后选择一条最短的路径。如图 3-20 所示，从城市 A 出发，再回到城市 A 的路径有 6 条，其中距离最短的路径是：A→B→C→D→A。



路径	总距离	路径	总距离
A→B→C→D→A	(13)	A→B→D→C→A	(14)
A→C→B→D→A	(19)	A→C→D→B→A	(16)
A→D→C→B→A	(13)	A→D→B→C→A	(19)

图 3-20　4 个城市的旅行商问题示意图

求解旅行商问题比求哈密尔顿回路更困难。当城市数不多时，找到最短路径并不难。但是随着城市数的增加，路径组合数会呈指数级增长，计算时间的增长率也呈指数级增长，一直达到无法计算的地步，这种情况称为计算的“组合爆炸”问题。

图 3-20 中，旅行商从 A 起，访问全部城市后再回到 A 点时，路径数为(4－1)!＝6 条。旅行商问题目前还没有找到搜索最短路径的优秀算法，只能采用穷举法一个一个城市地去搜索尝试。因此，n 个城市旅行商的全部路径有(n－1)! 条。

【例 3-43】 当城市数为 26 时，计算旅行商访问所有城市的全部路径数，代码如下。

```
>>> import math                          # 导入标准模块 - 数学
>>> math.factorial(25)                   # 计算 25 的阶乘值
155112100433309859840000000              # 所有路径数 = 1.5×10^25，计算量发生组合爆炸
```

说明：一个简化的旅行商问题求解案例，参见本书配套教学资源程序 F3-8.py。

2010 年，英国伦敦大学奈杰尔·雷恩(Nigel Raine)博士在《美国博物学家》发表论文指出，蜜蜂每天都要在蜂巢和花朵之间飞来飞去，在不同花朵之间飞行是一件很耗体力的事情，因此蜜蜂每天都在解决“旅行商问题”。雷恩博士利用人工控制的假花进行实验，结果显示，不管怎样改变假花的位置，蜜蜂稍加探索后，很快就可以找到在不同花朵之间飞行的最短路径。如果能理解蜜蜂是怎样做到这一点的，将对解决旅行商问题有很大帮助。

旅行商问题应用广泛，如在车载 GPS 中，经常需要规划行车路线，如何做到行车路线距离最短，就需要对旅行商问题进行求解；在印制电路板(Printed Circuit Board，PCB)设计中，如何安排众多的导线使线路距离最短，也需要对旅行商问题进行求解。旅行商问题在其他领域也有广泛的应用，如物流运输规划、网络路由节点设置、遗传学领域、航空航天等领域。

3.4.4 单向函数：公钥密码的基础

1. 单向函数的概念

在现实世界中，单向函数的例子非常普遍。例如，把盘子打碎成数百块碎片是很容易的事情，然而要把所有碎片再拼成一个完整的盘子，却是非常困难的事情。例如，将挤出的牙膏塞回牙膏管子，这要比把牙膏挤出来困难得多。类似地，将两个大素数相乘要比将它们的乘积因式分解容易得多。

【例 3-44】 计算 41×71＝2911 很容易，但是将 2911 因数分解为 41 和 71 却很难。

2. 单向函数的假设

单向函数听起来没什么问题，但若严格地按数学定义，并不能从理论上证明单向函数的存在，因为这将意味着计算科学中最具挑战性的猜想 P＝NP 成立，而目前的 NP 完全性理论不足以证明存在单向函数。即便是这样，还是有很多函数看起来像单向函数，我们能够有效地计算它们，而且至今还不知道有什么办法能容易地求出它们的逆，因此假定存在单向函数。

最简单的单向函数就是整数相乘。不管两个多大的整数，都可以很容易地计算出它们的乘积；但是对于两个 100 位的素数之积而言，很难在合理的时间内分解出两个素数因子。

3. 单向陷门函数

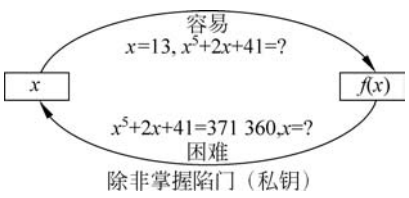


图 3-21 单向陷门函数求解示意图

单向陷门函数是一类特殊的单向函数，它在一个方向上易于计算而反方向难于计算。但是，如果你知道某个秘密，就能很容易地在另一个方向上计算这个函数。也就是说，已知 x，易于计算 $f(x)$；而已知 $f(x)$，却难于计算 x，如图 3-21 所示。然而，如果有一些秘密信息 y，一旦给出 $f(x)$和 y，就很容易计算

出 x。例如，钟表拆卸和安装就像一个单向陷门函数，很容易把钟表拆成数百个小零件，而把这些小零件组装成能够工作的钟表却非常困难；然而，通过某些秘密信息（如钟表装配手册），就能把钟表安装还原。

4. 单向陷门函数在密码中的应用

单向函数不能直接用作密码体制，因为如果用单向函数对明文进行加密，即使是合法的接收者也不能还原出明文，因为单向函数的逆运算非常困难。密码体制更关心的是单向陷门函数。毫不夸张地说，**公钥密码体制的设计就是寻找单向陷门函数**。注意，单向陷门函数不是单向函数，它只是对那些不知道陷门的人表现出单向函数的特性。**著名的 RSA（Rivest，Shamir，Adleman）密码体制就是根据单向陷门函数而设计**。

3.4.5 哲学家就餐问题：死锁控制

1965 年，迪科斯彻（Dijkstra）在解决操作系统的“死锁”问题时，提出了“哲学家就餐”问题，他将问题描述为：有五位哲学家，他们的生活方式是交替地进行思考和进餐。哲学家们共用一张圆桌，分别坐在周围的五把椅子上，圆桌上有五个碗和五支餐叉，如图 3-22 所示。平时哲学家进行思考，饥饿时哲学家试图取左、右最靠近他的餐叉，只有在他拿到两支餐叉时才能进餐；进餐完毕，放下餐叉又继续思考。

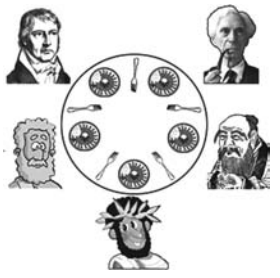


图 3-22 哲学家就餐问题

在哲学家就餐问题中，有如下约束条件：一是哲学家只有拿到两支餐叉时才能吃饭；二是如果餐叉已被别人拿走，哲学家必须等别人吃完之后才能拿到餐叉；三是哲学家在自己未拿到两支餐叉前，不会放下手中已经拿到的餐叉。

哲学家就餐问题用来说明在并行计算中（如多线程程序设计），多线程同步时产生的问题；它还用来解释计算机死锁和资源耗尽问题。

哲学家就餐问题形象地描述了多进程以互斥方式访问有限资源的问题。计算机系统有时不能提供足够多的资源（CPU、内存等），但是又希望同时满足多个进程的使用要求。如果只允许一个进程使用计算机资源，系统效率将非常低下。研究人员采用一些有效的算法，尽量满足多进程对有限资源的需求，同时尽量减少死锁和进程饥饿现象的发生。

当五个哲学家都左手拿着餐叉不放，同时去取他右边的餐叉时，就会无限等待下去，引起死锁现象发生。在实际问题中，常用的解决方法是资源加锁，使资源只能被一个程序或一段代码访问。当一个程序想要使用的资源被另一个程序锁定时，它必须等待资源解锁。当多个程序涉及加锁资源时，在某些情况下仍然有可能发生死锁。

计算机死锁没有完美的解决方法。解决系统死锁需要频繁地进行死锁检测，这会严重降低系统的运行效率，得不偿失。由于死锁并不经常发生，大部分操作系统采用了“鸵鸟算法”，即尽量避免发生死锁，一旦真正发生了死锁，就假装什么都没有看见，重新启动死锁的程序或系统。可见鸵鸟算法是为了效率优先而采用的一种折中策略。

3.4.6 两军通信：信号不可靠传输

网络通信能不能做到理论上可靠？特南鲍姆教授在《计算机网络》一书中提出了一个经典的“两军通信”问题。

如图 3-23 所示，一支 A 军在山谷里扎营，在两边山坡上驻扎着 B 军。A 军比两支 B 军中的任意一支都要强大，但是两支 B 军加在一起就比 A 军强大。如果一支 B 军单独与 A 军作战，它就会被 A 军击败；如果两支 B 军协同作战，他们能够把 A 军击败。两支 B 军要协商一同发起进攻，他们唯一的通信方法是派通信员步行穿过山谷，通过山谷的通信员可能躲过 A 军的监视，将发起进攻的信息传送到对方 B 军；但是通信员也可能被山谷中的 A 军俘虏，从而将信息丢失（信道不可靠）。问题：是否存在一个方法（协议）能实现 B 军之间的可靠通信，使两军协同作战而取胜？这就是两军通信问题。

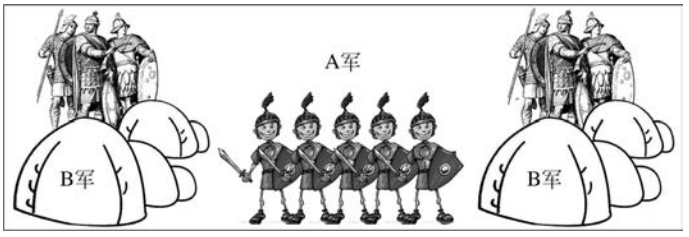


图 3-23　两军通信问题：B 军之间如何实现安全可靠的通信

特南鲍姆教授指出，不存在一个通信协议使山头两侧的 B 军达成共识。因为在信道不可靠的情况下，永远无法确定最后一次通信是否送达了对方。如果更深入一步讨论，当 B 军通信员被 A 军截获，并且通信内容被 A 军修改后（黑客攻击），情况将会变得更加复杂。可见，**网络通信是在不可靠的环境中实现尽可能可靠的数据传输**。

两军通信问题本质上是一致性确认问题，也就是说通信双方都要确保信息的一致性。因特网 TCP（传输控制协议）采用“三次握手”进行通信确认，这是一个通信安全和通信效率兼顾的参数。我们说“TCP 是可靠通信协议”，仅仅是说它成功的概率较高而已。

两军通信问题在计算机通信领域应用广泛，如数据的发送方如何确保数据被对方正确接收；在计算机集群系统中，如果在指定时间内（如数秒）没有收到计算节点的“心跳”信号时，怎样确定对方是“宕机”了，还是通信网络出现了问题；在密码学中，最困难的工作是如何将密钥传送给接收方（密钥分发），如果在理论或实际中有一个安全可靠的方法将密钥传送给接收方，则加密和解密系统就显得多此一举。

习　题　3

3-1　简要说明什么是计算思维。

3-2　简要说明计算领域解决问题的主要步骤。

3-3　简要说明计算领域哪些问题不可计算以及如何解决这类问题。

3-4　写出“石头剪刀布”游戏的博弈策略矩阵。

3-5　简要说明如何利用统计语音数学模型，将宠物狗的肢体语言翻译为人类语言。

3-6　简要说明汉诺塔问题递归求解过程中，盘子的移动有哪些规律。

3-7　一个黑白图像局部区域有 1000 个随机数据，如 10001101…，现在需要 1 转换为 0，将 0 转换为 1，请用图灵机编写程序实现以上功能（要求进行程序注释）。

3-8　“现代计算机与图灵机在计算性能上是等价的”这句话正确吗？

3-9　简要说明不可计算的问题有哪些类型。

3-10　简要说明 P 问题。

3-11　简要说明 NP(非确定性多项式)问题。

3-12　简要说明不可计算问题的解决方法。

3-13　简要说明计算复杂性理论的研究内容。

3-14　实验：参考例 3-19 的程序案例，实现囚徒困境的博弈。

3-15　实验：参考例 3-39 的程序案例，用递归求解汉诺塔问题。