# 第3章 计算思维和学科基础

计算科学专家迪杰斯特拉说过：**“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我们的思维能力。”**计算科学的发展也影响着人类的思维方式，从最早的结绳记数到目前的电子计算机，人类思维方式发生了相应的改变，如计算生物学改变了生物学家的思维方式，计算博弈论改变了经济学家的思维方式，计算社会科学改变了社会学家的思维方式，量子计算改变了物理学家的思维方式，等等。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。

## 3.1 计 算 思 维

### 3.1.1 计算思维的特征

**1. 计算思维的定义**

计算思维是美国哥伦比亚大学周以真(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。**

**2. 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。

计算思维是人的思维方式，不是计算机的。计算思维是人类求解问题的思维方法，而不是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相互融合。**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，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？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一个细节的情况下，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到，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

如图 3-1 所示，计算思维包含的基本概念很多。周以真教授认为，**计算思维可以分为数学思维和工程思维两部分**。数学思维的基本概念包括复杂性、抽象、可计算性、数学模型、算法、数据结构、一致性和完备性、不确定性等。

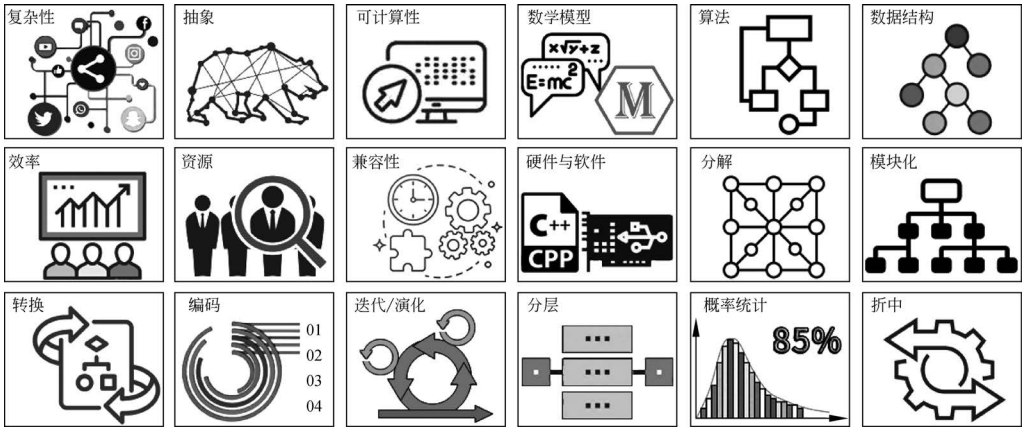



图 3-1　计算思维的基本概念

**1. 复杂性**

复杂问题的不确定包括二义性（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、教师排课问题等）、指数组合爆炸（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。计算科学专家迪杰斯特拉曾经指出，**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《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 $A=\{aaaaaaa\cdots\}$

案例分析：这是一个简单系统，相应程序为，在每一个 $a$ 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 $B=\{aabaabaabaab\cdots\}$

案例分析：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 $C=\{aabaababbaabaababb\cdots\}$

案例分析：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 $D=\{aababbababbbabaaababbab\cdots\}$

案例分析：字符排列毫无规律，如果希望编程解决，就必须将字符串全部列出。因此，**一旦程序大小与试图描述的系统大小相当，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，称该系统为根本复杂系统。在达到根本复杂系统之前，人们可以编写出解决问题的程序。

（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,“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。自然语言的同一语句在不同语境下会有不同的理解(二义性),例如,“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?”。程序语言不允许语句出现二义性,因此程序语言都有一套人为规定的语法规则。

解决程序语句的二义性有以下方法:一是设置一些规则,规定在出现二义性的情况下,指出哪一个是正确的,例如,程序语句中出现整数与浮点数混合运算时,都按浮点数处理;二是强制修改为正确格式,例如,程序语句出现错误时,提示用户进行强制修改。

(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(见图3-2、表3-1)。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。

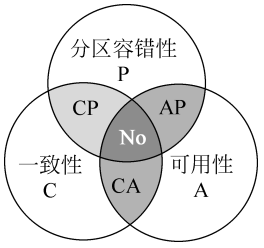



图3-2 CAP组合方式

**表3-1 CAP组合方式的不同特征**

| CAP组合 | 满足特性 | 业务场景 |
|---|---|---|
| **AP** | 放弃一致性C,保证分区容忍性和可用性,即在短时间内不能保证数据的一致性 | 例如,管理商品订单时,今日退款成功,明日退款到账,只要用户可以接受在一定时间内到账即可,这是很多分布式系统设计时的选择 |
| **CP** | 放弃可用性A,保证数据的一致性和数据分区的容错性 | 例如,跨银行转账,转账事务要等待双方系统都完成,整个事务才算完成,追求数据一致性,放弃可用性 |
| **CA** | 放弃分区容错性P,即不进行数据的分区容错性管理 | 例如,不考虑网络不通、服务节点宕机等问题,最常用的关系型数据库就满足了CA特征 |

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会涉及众多的程序模块,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导致系统复杂性的增加**。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】** 维数灾难是指随着多项式变量维数的增加,会使解题计算量呈指数增加。

例如，天气预报的数学模型是一组复杂的非线性方程组，在 10 个变量组成的 10 阶多项式中，计算式的个数超过了 100 万个。例如，图像识别中，一张分辨率为 32×32×3(像素×像素×色彩)的图片，它的维数是 3072，也就是说每张图片有 3072 个自由度。如此高维的函数，用多项式拟合的复杂性不可想象。幸好神经网络提供了很好的高维解决方案，神经网络是一类特殊的函数，它对高维函数提供了一种有效的逼近方法。

**程序的复杂性来自大量的不确定性**，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。著名计算科学家布莱恩·W. 克尼汉(Brian W. Kernighan，UNIX 和 C 语言开发者之一，“hello，world”程序作者)在 *Software Tools* 一书中总结了软件开发的性质，他说：**“控制复杂性是软件开发的根本。”**

(5) 简单性原则。**计算技术的发展遵循了简单性原则**，一些复杂的技术往往被简单技术取代，例如，复杂的 Ada 语言被简单的 C 语言取代；复杂的 IBM OS/360 操作系统被 UNIX 取代，而更加简单和开放的 Linux 又取代了 UNIX 系统；复杂的大规模并行处理(massively parallel processing，MPP)计算机结构被简单的计算机集群结构取代；复杂的异步传输模式(asynchronous transfer mode，ATM)网络传输技术被简单的以太网技术取代；等等。系统设计应当遵循保持简单原则(KISS)，**应当推崇简单就是美，任何没有必要的复杂都需要避免**(奥卡姆剃刀原则)。

**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。有才华的画家视抽象艺术为最美但又最难画，其中包含的艺术内涵太丰富。

**【例 3-6】** 如图 3-3 所示，毕加索终生喜欢画牛，他年轻时画的牛体形庞大，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。

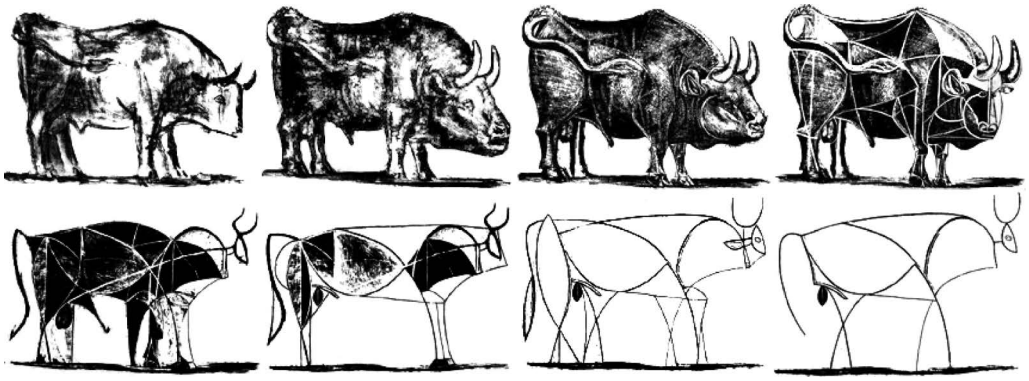


图 3-3 毕加索画牛的抽象过程

(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。计算自动化要求对事物进行某种程度的抽象，抽象的目标是最终能够用机器进行自动计算。抽象是对实际事物进行人为处理，抽取共同的、本质的特征，并对这些特征进行描述，从而适合计算机的处理方式。例如，词语“三个苹果、三本书、三辆车”三个短语中，人们先将名词(苹果、书、车)抽去，再将量词抽去(个、本、辆)，这样就抽象出了数字“三”。计算思维的抽象方法有简化、分

解、替代、分层、编码、公式化、图表化等。

**【例3-7】** 数据的抽象有以下方法：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、树状、连通图等类型，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；等等。

**【例3-8】** 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硬件抽象层(hardware abstraction layer，HAL)；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；等等。

**【例3-9】** **数学建模过程就是将实际问题抽象成数学形式的过程**。为了实现程序的自动化计算，可以将问题抽象为一个数学模型。例如，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；将计算机体系结构抽象成不同的层次结构；将语言翻译抽象为统计语言模型；等等。

**3. 分解**

笛卡儿在《谈谈方法》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，然后分别解决。”**

(1) 用等价关系进行系统简化。复杂系统可以看成一个集合，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个子系统，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) 用分治法进行分解。分而治之是指把一个复杂问题分解成若干个简单的问题，逐个解决，这种朴素的思想来源于人们生活与工作的经验。编程人员采用分治法时，应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、兼容性、硬件与软件、编码(转换)、时间和空间、模块化、资源、复用、安全、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、重复执行指令优先等，它们体现了效率优先原则；而队列、网络数据包转发等，体现了平等优先原则。效率与平等的选择需要根据实际问题进行权衡分析。例如，绝大部分算法都采用效率优先原则，但是也有例外。例如，对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；棋类博弈程序也是这样。

但是，效率是一个双刃剑，经济学家奥肯(Arthur M. Okun)在《平等与效率——重大的

抉择》一书中断言:**“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】** 老式阴极射线管(CRT)采用电子束逐行扫描方式显示图像,而新型液晶显示器(LCD)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 也沿用了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,例如,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。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,大幅降低了产品成本。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远高于软件,因而,软件硬件化能够大幅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、神经网络计算等,目前都采用专用芯片处理,这是软件硬件化的典型案例。**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 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件和硬件融合。例如,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。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;只使用软件,会使程序变得很复杂,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

机和路由器是软件和硬件相互融合的经典案例。

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,每一项指标都很重要,**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,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错误检查和纠正(ECC)校验,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校验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

计算工程解决问题一般需要经过以下步骤: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;三是从问题中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,直至得到最终解答。

**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) 解题条件。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。

**2. 对象的离散化**

计算机处理的对象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化为离散型问题后(数字化),才能被计算机处理。

**【例3-16】** 在计算机屏幕上显示一幅图像时,计算机必须将图像在水平和垂直方向分解成一定分辨率的像素点(离散化);然后将每个像素点分解成红绿蓝(red green blue,

RGB)三种基本颜色；再将每种颜色的亮度分解为0～255(1字节)个等级；这样计算机就会得到一大批有特定规律的离散化数字，也就能够任意处理这张图像了，如图像的放大、缩小、旋转、变形、变换颜色等操作。

**3. 数学模型和算法**

**数学模型是解决实际问题的数学形式**(如数学方程、二维表格、逻辑关系、抽象图形等)，**算法是实现数学模型的计算方法和计算步骤**。数学模型和算法密切相关。

(1) 构建数学模型。首先对需要解决的问题用数学形式描述它，通过这种描述来寻找问题的性质，看看这种描述是不是合适，如果不合适，则换一种方式。通过反复地尝试、不断地修正来达到一个满意的结果。数学模型应当满足三个方面的要求：一是能较好地模拟现实问题；二是应当容易理解；三是要便于程序实现。

(2) 选择合适的算法。算法的描述形式有自然语言、伪代码、程序流程图等，所有算法都可以转换为程序代码。一个问题有多种解决方法，因此解决同一问题也有不同的算法(参见例4-4赝品金币问题)。每一种算法都有适宜解决问题的类型，例如，穷举法适用于查找没有顺序关系的对象，而二分查找法则适用于查找按序排列的对象。

**4. 程序设计**

**程序设计的本质就是将解决问题的思想(算法)翻译成计算机能够执行的指令(程序)**。图灵在“计算机器与智能”论文中指出：“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，用程序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程序语言表述的算法。

## 3.2 数学建模

### 3.2.1 数学模型的构建方法

**1. 数学模型的基本概念**

**数学模型是指用数学语言描述一个问题，使它便于程序处理**。数学模型大部分情况下用数学方程表达(如例3-18商品提价模型、例4-22蒙特卡洛算法模型)，但是，数学方程仅仅是数学模型的主要表达形式之一，数学模型也可以用表格(如例3-19囚徒困境模型)、抽象图形(如二叉树、联通图、例3-36细胞自动机模型、图3-21哥尼斯堡七桥示意图)、符号(如例3-35布尔检索模型)、逻辑关系(参见5.4.5节数理逻辑应用中加法器电路的真值表)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，而非数值型问题的数学建模则相对复杂。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，需要将这些问题抽象化，然后建立它们的数学模型。

**【例3-17】** 观众对一部电影的评论五花八门，可以通过“情感计算”将这些评价进行分类统计。即评论中有好、推荐、顶呱呱、精彩等词语时，归类于很好，用字母A表示(符号化)；评论中有还行、一般、有点意思等词语时，归类于一般，用字母B表示；评论中如果充满了垃圾、弱智、浪费时间等词语时，归类于差，用字母C表示。然后对评论关键词按ABC

分类进行统计,就可以得出观众对影片观感的量化指标。由以上分析可见,对社会活动进行数学建模并不是一件很困难的事情。

**2. 数学建模的一般方法**

笛卡儿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,把所有问题转化为数学问题;第二,把所有数学问题转化为一个代数问题;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的对模型进行综合;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. 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 3-18】** 商品提价问题的数学模型。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,销售量大但利润低;定价高时,利润高但销售量减少。假设某商场销售的某种商品单价 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。如果要使年度总销售收入不少于 75 万元,求该商品的最高提价。

数学建模步骤如下。

(1) 已知条件:单价 25 元×销售 3 万件=年度销售收入 75 万元。

(2) 约束条件 1:每件商品提价 1 元,则销售量减少 0.1 万件。

(3) 约束条件 2:保持年度总销售收入不低于 75 万元。

(4) 设最高提价为 $x$ 元,提价后的商品单价:$(25+x)$元。

(5) 提价后的销售量:$(30\,000-1000x)$件。

(6) 组合以上条件:$(25+x)\times(30\,000-1000x)\geqslant 750\,000$。

(7) 简化后的数学模型:$(25+x)\times(30-x)\geqslant 750$。

**4. 编程求解**

求解例 3-18 问题的 Python 程序如下。

```
>>> from sympy import symbols, solve                    # 导入第三方包
>>> 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。

## 3.2.2 囚徒困境:博弈策略建模

**1. 冯·诺依曼与博弈论**

博弈是双方通过不同策略相互竞争的游戏,棋类活动是最经典的博弈行为。1944 年,冯·诺依曼和奥斯卡·摩根斯特恩(Oskar Morgenstern)发表了著作《博弈论和经济行为》,首次从讨论经济行为出发,说明了建立博弈论的重要性。囚徒困境说明了为什么合作对双

方有利时，保持合作也非常困难。囚徒困境也反映了个人最佳选择并非团体最佳选择。虽然囚徒困境只是一个数学模型，但现实中的商业竞争、社会谈判、国际合作等，都会频繁出现类似的情况。

**2. 囚徒困境问题描述**

**【例 3-19】** 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，并且向囚徒提出：如果囚徒双方都认罪，则双方都获 3 年刑期；如果囚徒双方都不认罪，则双方都获 1 年刑期；如果囚徒 A 不认罪，而囚徒 B 认罪，则囚徒 A 不获刑，但是囚徒 B 会获刑 5 年，反之也是如此。

案例分析：根据题意，囚徒双方的选择如表 3-2 所示。

**表 3-2 囚徒困境中双方的选择**

| 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 |

**3. 囚徒的策略选择困境**

困境中，两名囚徒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**

以下是囚徒困境建模的一般形式。

策略符号化。如表 3-3 所示，我们对囚徒困境中的各种行为以符号表示。

**表 3-3 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 | 单独背叛所获 |

从表 3-3 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。

根据表 3-2，假设囚徒认罪时选择为 1；囚徒不认罪时选择为 2(用数字 1、2 表示选择是为了简化下面的程序设计)。囚徒困境数学模型的表格形式如表 3-4 所示。

**表 3-4 囚徒困境问题的数学模型**

| 博 弈 方 案 | 囚徒双方的博弈策略 | 囚徒选择的逻辑表达式 | 囚徒 A、B 的刑期 |
|---|---|---|---|
| 1 | A 认罪；B 认罪 | a=1　and　b=1 | A 三年，B 三年 |
| 2 | A 不认罪；B 认罪 | a=2　and　b=1 | A 零年，B 五年 |
| 3 | A 认罪；B 不认罪 | a=1　and　b=2 | A 五年，B 零年 |
| 4 | A 不认罪；B 不认罪 | a=2　and　b=2 | A 一年，B 一年 |

**5. 囚徒困境的程序实现**

根据表3-4所示的数学模型,我们可以用多条件选择结构来设计程序。

**【例3-20】** 用程序实现囚徒困境的博弈。Python程序如下。

```
while True:                                                    # 建立循环判断
    a = input('囚徒A选择【1=认罪;2=不认;0=退出】:')              # 输入A的博弈策略
    b = input('囚徒B选择【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判三年,唉;A判三年,唉')                        # 打印博弈结果
    elif a == '2' and b == '1':                                # A不认罪,B认罪
        print('A判零年,哈哈;B判五年,呜呜')                    # 打印博弈结果
    elif a == '1' and b == '2':                                # A认罪,B不认罪
        print('A判五年,呜呜;B判零年,哈哈')                    # 打印博弈结果
    elif a == '2' and b == '2':                                # A、B都不认罪
        print('A判一年,@|@;B判一年@|@')                        # 打印正确选择
    else:                                                      # 否则
        break                                                  # 退出循环,结束程序
>>> 囚徒A选择【1=认罪;2=不认;0=退出】:…                       # 程序输出(略)
```

**6. 囚徒困境模型的最佳策略**

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和分析。

单次和多次囚徒困境博弈的结果会有所不同。对一次性囚徒困境博弈来说,最佳策略是背叛。重复多次的囚徒困境博弈中,最佳博弈策略是“以牙还牙”,这是阿纳托尔·拉波波特(Anatol Rapoport)发明的方法。策略如下:在博弈最开始选择合作,然后每次采用对手前一回合的策略。即如果对手上一次为合作,则你选择合作;如果对手上一次为背叛,则你选择背叛(即以牙还牙)。

## 3.2.3 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。计算科学专家自1950年开始,一直致力于研究如何让机器对语言做更好的理解和处理。

**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。

**2. 基于语法分析的机器翻译**

1956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】** 用形式语言表示短语“那个穿红衣服的女孩是我的女朋友”。

案例分析:$S$=短语,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。用形式语言表示如图3-4所示。

**【例3-22】** 用形式语言表示程序的条件语句:if x==2 then {x=a+b}。

案例分析:用形式语言(抽象语法树)表示的程序语句如图3-5所示。

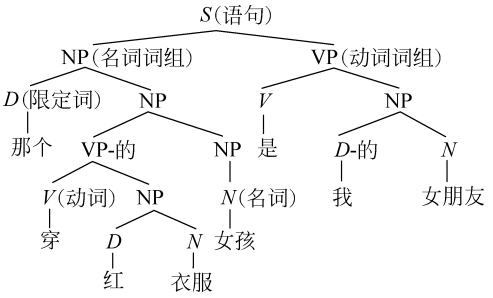



图 3-4　用形式语言表示短语

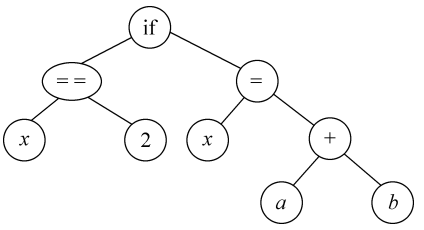



图 3-5　用形式语言表示程序语句

乔姆斯基提出形式语言理论后，人们想到了让机器学习语法、分析语句等。遗憾的是，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。

**3. 贝叶斯定理**

(1) 正向概率。假设一个口袋里有 $P$ 个红球、$Q$ 个白球，它们除了颜色外，其他属性完全一样。当人们伸手进去摸一个球时，摸到红球的概率可以计算出来。

(2) 逆向概率。如果人们并不知道口袋里红球和白球的比例，从口袋里摸出一些球，然后根据手中红球和白球的比例，对口袋里红球和白球的比例进行推测，这就是逆向概率问题。也就是说，当人们不能准确知道某个事物的本质时，可以依靠经验去判断事务的本质和属性。

(3) 贝叶斯公式。数学家托马斯·贝叶斯(Thomas Bayes)于 1763 年发表了解决“逆向概率”的论文。贝叶斯公式建立在主观判断的基础上，首先估计一个值，然后根据事实不断修正这个值。贝叶斯定理为：如果 $k$ 个独立事件 A1，A2，…，A$k$ 发生的概率为 $P$(A1)＋$P$(A2)＋…＋$P$(A$k$)＝1，在已知事件 B 的概率时，朴素贝叶斯定理如下

$$P(\mathrm{A}\mid\mathrm{B})=(P(\mathrm{B}\mid\mathrm{A})\times P(\mathrm{A}))/P(\mathrm{B}) \tag{3-1}$$

式中，$P$(A|B)是 B 情况下 A 事件发生的概率(后验概率)；$P$(B|A)为 A 事件下得到数据 B 的概率(似然度)；$P$(A)是 A 事件发生的概率(先验概率)；$P$(B)是 B 事件发生的概率(标准化常量)。简单地说，贝叶斯公式为：后验概率＝先验概率×似然度。

**【例 3-23】** 1788 年，亚历山大·汉密尔顿(Alexander Hamilton)、约翰·杰伊(John Jay)、詹姆斯·麦迪逊(James Madison)共同发表了《联邦党人文集》。该论文集是匿名发表的，在 85 篇文章中，有 73 篇文章的作者较为明确，其余 12 篇存在争议。1955 年，哈佛大学统计学教授莫斯特(Fredrick Mosteller)等用统计学的方法，对《联邦党人文集》的作者进行鉴定。他们采用以贝叶斯公式为核心的分类算法，经过 10 多年的研究，推断出了 12 篇文章的真实作者，他们的研究方法在统计学界引发了轰动。

**4. 基于概率统计的机器翻译**

贝叶斯定理名扬天下，主要得益于人工智能领域的应用，特别是自然语言识别领域。用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？当检测到的语音信号为(o1，o2，o3)时，根据这组信号推测发送的短语是(s1，s2，s3)。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(o1，o2，o3，…)的情况下，求概率 $P$(o1，o2，o3，…|s1，s2，s3，…)达到最大值的短语(s1，s2，s3，…)，即

$$P(\mathrm{o1},\mathrm{o2},\mathrm{o3},\cdots\mid\mathrm{s1},\mathrm{s2},\mathrm{s3},\cdots)\times P(\mathrm{s1},\mathrm{s2},\mathrm{s3},\cdots) \tag{3-2}$$

式中,$P(o1,\cdots|s1,\cdots)$是条件概率,它表示在 s1 发生的条件下,o1 发生的概率,其中 o1 和 s1 具有相关性,读作"在 s1 条件下 o1 的概率"。$P(s1,s2,s3,\cdots)$是联合概率,表示 s1、s2、s3 等事件同时发生的概率,其中 s1、s2、s3 是相互独立的事件。因此,在式(3-2)中,$P(o1,o2,o3,\cdots|s1,s2,s3,\cdots)$表示某个短语(s1,s2,s3,…)被读成(o1,o2,o3,…)的可能性(概率值)。而 $P(s1,s2,s3,\cdots)$表示字串 s1,s2,s3,…成为合理短语的可能性(概率值)。

如果我们把(s1,s2,s3,…)当成中文,把(o1,o2,o3,…)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(o1,o2,o3,…)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。

**5. N 元统计语言模型的数学建模**

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

$$P(S)=\text{argmax}_y P(x\mid y)\times P(y) \tag{3-3}$$

式中,$P(S)$是翻译语句的概率值;$\text{argmax}_y$ 是求变量 $y$ 最大值;$P(x|y)$是翻译模型,用于分析单词和短语如何翻译,它从数据统计中学习规律;$P(y)$是语言模型,用于描述一个短语合理性的概率,它从英语语料库中学习(训练)知识。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,这大幅减少了计算量。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,还会涉及很多其他复杂的细节。

统计语言模型依赖单词的上下文(本词与上个词和下个词的关系)概率分布。例如,当一个短语为"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 3-24】** 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个不同的切分方法时,我们希望找到概率最大的那个分词**。

假设任意一个词 $w_i$ 的出现概率只与它前面的词 $w_{i-1}$ 有关(马尔可夫假设),于是问题得到了简化。这时短语 S 出现的概率为

$$P(S)=P(w_1)\times P(w_2\mid w_1)\times P(w_3\mid w_1\ w_2)\times\cdots\times P(w_n\mid w_1w_2\cdots w_{n-1}) \tag{3-4}$$

简单地说,统计语言模型的计算思维就是:短语 S 翻译成短语 F 的概率是短语 S 中每一个单词翻译成 F 中对应单词概率的乘积。根据式(3-4),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。

(1) 一元模型为:$P(w_1w_2w_3w_4)=P(w_1)\times P(w_2)\times P(w_3)\times P(w_4)$。

(2) 二元模型为:$P(w_1w_2w_3w_4)=P(w_1)\times P(w_2|w_1)\times P(w_3|w_2)\times P(w_4|w_3)$。

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

**【例 3-25】** 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 3-26】 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)，符号“≫”为远大于。

【例 3-27】 拼写纠错。$P$(about fifteen **minutes** from)＞ $P$(about fifteen **minuets** from)。

【例 3-28】 语音识别。$P$(I saw a van)≫ $P$(eyes awe of an)。

【例 3-29】 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 3-6 所示。

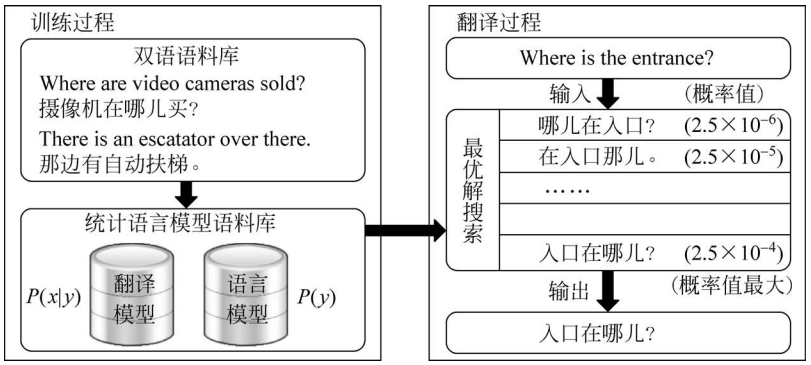



图 3-6 统计语言模型的机器翻译过程

说明：在线简单语句翻译的案例，参见本书配套教学资源程序 F3_1.py。

## 3.2.4 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，我们先介绍以下实际生活中的例子。

【例 3-30】 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？

例 3-30 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 3-31】 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？

案例分析：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，最后一个人将收到的数加上自己的收入后传给第一个人。第一个人从收到的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

$$\mathrm{SR}=(S-X)/n \tag{3-5}$$

式中,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。

例 3-31 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 3-31 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?我们将在 7.3.6 节安全计算的"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 $1 \leqslant Mi \leqslant N$)的财富,在不透露富翁财富的情况下,如何进行财富排名。

**【例 3-32】** 姚氏百万富翁问题的商业应用。假设书生向酒家的店小二买一壶酒,书生愿意支付的最高金额为 $x$ 元;店小二希望的最低卖出价为 $y$ 元。书生和店小二都希望知道 $x$ 与 $y$ 哪个大。如果 $x > y$,他们就可以开始讨价还价;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。

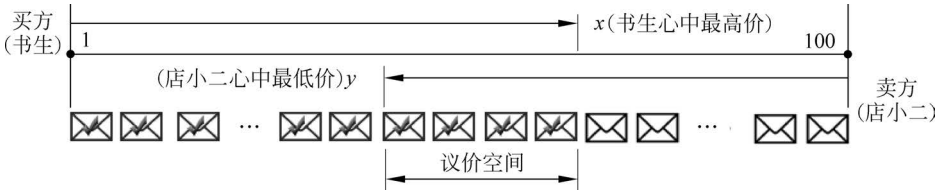



图 3-7 商品买卖中的多方安全计算问题

案例分析:假设书生和店小二设想的价格都低于 100 元,而且双方都不会撒谎(避免囚徒困境)。如图 3-7 所示,准备 100 个编号信封顺序放好,店小二回避,书生在小于和等于 $x$(最高买价)的所有信封内部做一个记号(如打√),并且顺序放好;然后书生回避,店小二把顺序放好的第 $y$(最低卖价)个信封取出,并将其余信封收起来。书生和店小二共同打开这个信封,如果信封内部有书生做的记号,说明 $x \geqslant y$,此时商品有议价空间;如果信封内没有记号,说明 $x < y$,此时商品没有议价空间。

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。

## 3.2.5 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(T)；否则赋值逻辑值假(F)。

**【例 3-33】** 利用搜索引擎查找有关“原子能应用”的文献。在搜索引擎的搜索栏中输入查询关键词“原子能 AND 应用”，这表示需要搜索的文献必须满足两个条件：一是包含“原子能”关键字，二是包含“应用”关键字。

**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(例如，100 篇文献要 100b)，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、第 5、第 10、第 11、第 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运算(AND)，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、第 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。最便宜的微型计算机都可以一次进行 32 位布尔运算，每秒进行 20 亿次以上运算。当然，由于这些二进制数中绝大部分位数都是零，我们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 3-34】** 假设网页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 3-5 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 3-5 “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 |

续表

| 文　　档 | T1＝病毒 | T2＝电脑 | T3＝计算机 | T4＝感冒 | T5＝医生 | T6＝生物 |
|---|---|---|---|---|---|---|
| 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号等,如表3-6所示。

**表3-6　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。目前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3-35】** 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3-8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。

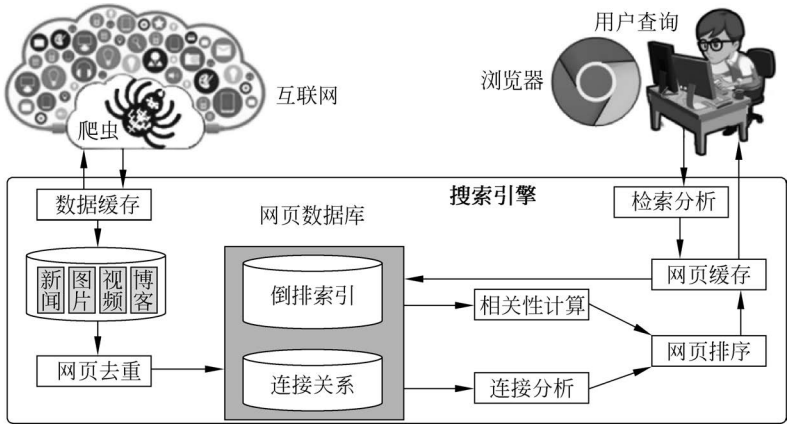



图3-8　搜索引擎工作过程

然后将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”“医生”“生物”词语的文档。

(2) 进行布尔位运算。对表 3-6 进行布尔位运算，符合查询要求的网页为 D2、D3、D4、D6，这个排序显然不太合理，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。所有网页排序确定后，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。

### 3.2.6 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**。每一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·诺依曼是最早提出机器自我复制概念的科学家之一。

1948 年，冯·诺依曼在论文“自动机的通用逻辑理论”中提出了“细胞自动机”(也译为元胞自动机)理论，细胞自动机是冯·诺依曼为计算机考虑的一种新的体系结构。冯·诺依曼对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，并提出了细胞自动机的自繁殖和自修复等理论。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程序，美国趣味数学大师马丁·加德纳(Martin Gardner)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，吸引了大批科学家的注意。

**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 MATLIB 软件下，输入“life”命令就可以运行“生命游戏”程序。

**【例 3-36】** 如图 3-9 所示，生命游戏是一个二维网格游戏，这个网格中每个方格都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的细胞的数量(见图 3-10)。如果相邻方格活细胞数量过多，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。

案例分析：如图 3-10 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 3-9 的方格网中，我们用黑色圆点表示该细胞为“生”，空格(白色)表示该细胞为“死”。或者说方格网中黑色圆点表示某个时候某种“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地变化。

**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。

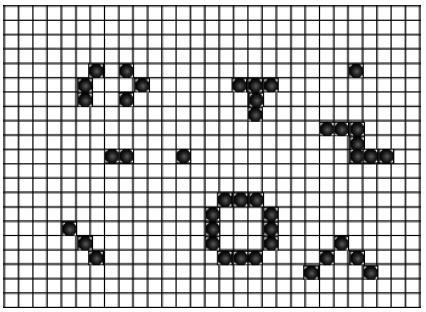


图 3-9　生命游戏(初始状态)

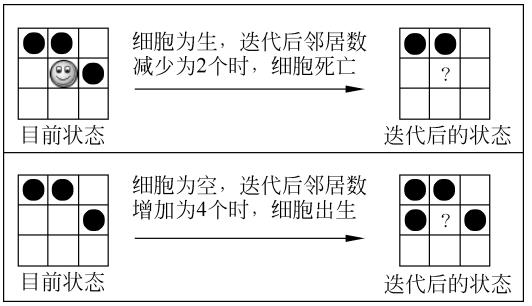



图 3-10　细胞的生死取决于邻居细胞的状态

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一个细胞都遵循完全一样的生存定律；再进一步,我们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(见图 3-10)。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,周而复始。

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,直至无穷。如图 3-11 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),根据上述生存定律,可以得到第 2、第 3、第 4 代的情况。

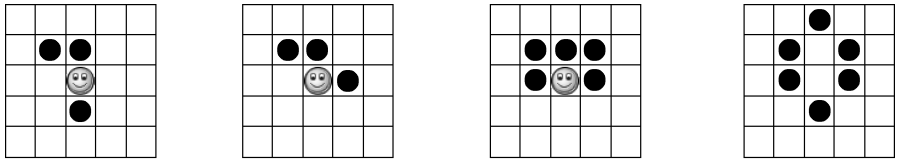



图 3-11　生命游戏中 4 代细胞的演化过程

生命游戏中，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。尽管生命游戏中每一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。这说明了一个计算思维的方法：**“复杂的事物（如生命）也可以用几条简单的规则表示。”**生命游戏提供了一个用简单方法处理复杂问题的经典案例。

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

$$A=(L_d,S,N,f) \tag{3-6}$$

式中，$L$ 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的集合，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3-12所示。

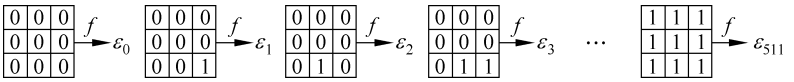



图3-12　二维细胞自动机的局部规则

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**：每一个等式对应图3-12中的一个框图，如 $f(111111111)=\varepsilon_{511}$ 对应图3-12最后的框图，$f(111111111)$的函数值为 $\varepsilon_{511}$，其中 $\varepsilon_{511}$ 或者为1或者为0。

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3-13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 $C_2$，$C_3$，…。

| | | |
|---|---|---|
| $X_1$ | $X_2$ | $X_3$ |
| $X_4$ | $X_0$ | $X_5$ |
| $X_6$ | $X_7$ | $X_8$ |

图3-13　细胞 $X_0$ 的邻居细胞

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，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

$$\text{如果}\quad S_t=1,\text{则}\ S_{t+1}=\begin{cases}1, & S=2,3\\ 0, & S\neq 2,3\end{cases} \tag{3-7}$$

$$\text{如果}\quad S_t=0,\text{则}\ S_{t+1}=\begin{cases}1, & S=3\\ 0, & S\neq 3\end{cases} \tag{3-8}$$

式中,$S_t$ 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。

以上公式化的数学模型看上去比较抽象,简单来说,假设活细胞(cell)为1,死细胞(cell)为0,则邻居细胞的累加和(sum)决定了本细胞下一轮的生存状态,我们可以根据这些逻辑关系,建立如表3-7所示的表格型数学模型。

**表3-7 生命游戏中本细胞状态与邻居细胞状态的数学模型**

| 生命属性 | 本细胞现在状态 $S_t$ | 邻居细胞总数 sum | 本细胞下一轮状态 $S_{t+1}$ |
|---|---|---|---|
| 繁殖 | 死,cell=0 | 如果 sum=3 | 则 cell=1 |
| 稀少 | 活,cell=1 | 如果 sum≤2 | 则 cell=0 |
| 过多 | 活,cell=1 | 如果 sum>3 | 则 cell=0 |
| 正常 | 活,cell=1 | 如果 sum=2 或 3 | 则 cell=1 |

**6. 康威"生命游戏"算法步骤**

(1) 对本细胞周围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 $ci$,它在 $t$ 时刻的状态为$(si,t)$,它两个邻居的状态为$(si-1,t)$,$(si+1,t)$,则细胞 $ci$ 在下一时刻的状态为$(si,t+1)$,可以用函数表示为

$$(si,t+1)=f((si-1,t),(si,t),(si+1,t))$$

其中,$(si,t)\in\{0,1\}$。

**7. 细胞自动机的应用**

**细胞自动机的重要特征是能与外界交换信息,并根据交换来的信息改变自己的动作,甚至改变自己的结构,以适应外界的变化**。细胞自动机的应用几乎涉及自然科学和社会科学的各个领域。在生物学领域,细胞自动机可以用来研究生命体的新陈代谢和遗传变异,以及研究进化和自然选择的过程。在数学领域,细胞自动机可以定义可计算函数,研究各种算法,如遗传算法等。2019年澳大利亚森林大火时,有专家用细胞自动机模型来预测大火蔓延趋势;2022年有专家用它来预测新冠病毒的流行趋势等。

说明:细胞自动机的案例,参见本书配套教学资源程序 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、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)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岁的图灵发表了论文"论可计算数及其在判定问题中的应用"。在论

文中，图灵构造了一台想象中的计算机，科学家称它为图灵机。图灵机与巴贝奇的分析机一样，也可以用来计算，只是图灵根本没有打算去建造这台机器。图灵机是一种结构十分简单但计算能力很强的计算模型，它用来计算所有有效可计算函数。

如图 3-14 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)。

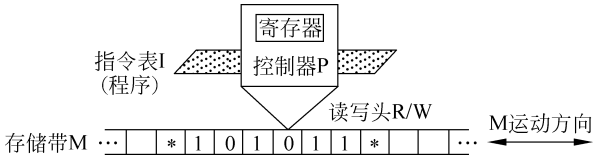



图 3-14　图灵机基本结构

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哪一条指令，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)。

是否存在一台图灵机，能够模拟所有其他图灵机？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?这与冯·诺依曼提出的细胞自动机理论有异曲同工之妙。美国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复杂的组织。

**4. 图灵机的重大意义**

图灵机是一种计算理论模型。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设计模型(图灵本人没有给出图灵机结构图)。图灵机虽然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。例如,计算机能做什么?不能做什么?这是很难直接回答的问题。如果将问题转换为哪些问题是图灵机可识别、可判定的,哪些问题图灵机不可判定?这样问题就变得容易解决了。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,然后根据算法编程并在图灵机上运行,如果图灵机能在有限步骤内停机,则这个问题就能解决。如果找不到这样的算法,或者算法在图灵机上运行时不能停机(无穷循环),则这个问题无法用图灵机解决。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,任何算法也解决不了。"**可见计算机的极限能力就是图灵机的计算能力。

(2) 图灵机引入了读/写、算法、程序等概念,极大地突破了过去计算机的设计理念。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。

**5. 图灵完备的程序语言**

图灵完备性用来衡量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,非图灵完备程序语言一般没有判断(if)、循环(for)、计算等指令,或者是这些指令功能很弱,因此无法模拟出图灵机。绝大部分程序语言都是图灵完备的,如 C/C++、Java、Python 等;也有少部分程序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),而非图灵完备的程序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,则停机问题可解;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?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。

案例分析：如图 3-15 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。如果输入程序是自终止的，则程序 T 中的 $x=1$；如果输入程序不能自终止，则程序 T 中的 $x=0$。如图 3-16 所示，修改停机程序为 P，程序 P＝停机程序 T＋循环结构。

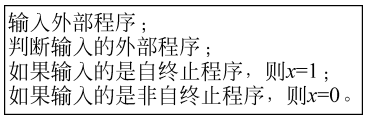



图 3-15　停机程序

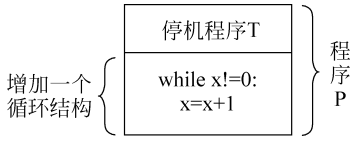



图 3-16　构造程序 P

如图 3-17 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。

如图 3-18 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。

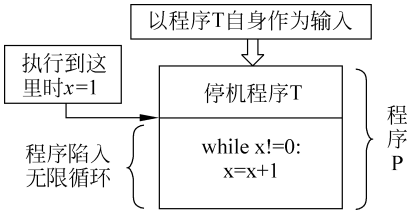



图 3-17　悖论 A：程序不能自停机

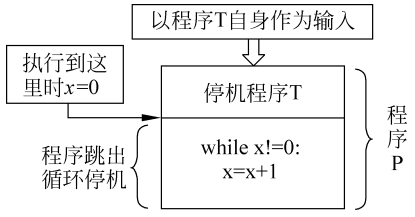



图 3-18　悖论 B：程序能够自停机

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 预言程序不会停机，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）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，他的鼻子会不会变长？

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不可解。

**2. 莱斯定理与程序的不可判定性**

莱斯定理（Rice's theorem）是可计算性理论中一条重要的定理。它可以简单表述为：**递归可枚举语言的所有非平凡性质都不可判定**。

莱斯定理比较抽象。通俗地说，“递归可枚举语言”可以理解为通常使用的 C、Java、Python 等程序语言；“非平凡性质”可以理解为程序具有的某些属性，例如，有些程序可能存在数组越界问题，有些程序则不存在这个问题；“不可判定”简单来说就是找不到一个完美的算法，来判定程序是否具有某个性质。莱斯定理可以通俗地理解为：一个程序是否具

备某种非平凡性质,这个问题如同停机问题一样,是不可判定的。**不存在一个程序可以判定任意程序具有某种非平凡性质**。

**【例 3-39】** 老师布置了一个作业,要求学生编写一个程序:对任意给定的整数 x,输出 2x,一个学生程序写完后马上交给了老师。这门课程有 200 名学生选修,这时老师有了一个偷懒的想法:他能不能写一个程序,自动判断学生上交的程序是否满足要求呢?莱斯定理打破了这个幻想:不存在这种具有"非平凡性质"的程序。

**3. 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,我们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?这个问题被莱斯定理证明与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,输入某些复杂系统的规则,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,唯一的办法就是让系统实际运行,没有任何一种算法能够事先给出这个复杂系统的运行结果。

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而人能做的事情。

## 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)曾编写过了一个神话传说: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 3-40】** 只有 3 个盘子的汉诺塔问题解决过程如图 3-19 所示。3 个盘子的最佳搬移次数为:$2^3-1=7$ 次。

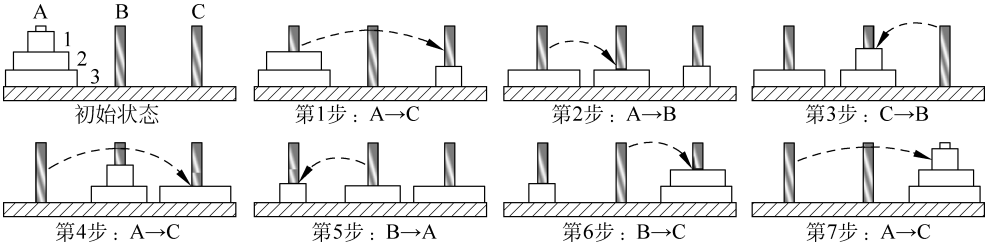



图 3-19 3 个盘子时汉诺塔的移动过程

汉诺塔 3 个圆柱为 A、B、C，$n$ 为盘子数，递归求解的 Python 程序如下。

```
1   def hanoi(n, a, b, c):                  # n 圆盘数,A 初始柱,B 过渡柱,C 目标柱
2       if n == 1:                          # 如果只有一个盘子
3           print(f'盘号{n}:{a}->{c}')      # 将盘子从 A 移到 C
4       else:                               # 否则
5           hanoi(n-1, a, c, b)             # 递归,将 n-1 个圆盘,从 A 经过 C 移到 B
6           print(f'盘号{n}:{a}->{c}')      # 将第 n 个盘子(最底层盘子)从 A 移到 C
7           hanoi(n-1, b, a, c)             # 递归,将 n-1 个圆盘,从 B 经过 A 移到 C
8   n = int(input('请输入盘子数:'))          # 接收用户输入数据,并转换为整数
9   hanoi(n, 'A', 'B', 'C')                 # 调用 hanoi()递归函数,并传送实参
>>> 请输入盘子数:3                          # 程序输出,计算时间随盘子数呈指数增长
    盘号 1:A->C…                            #(略)
```

递归程序虽然简单，但是程序理解困难，需要反复琢磨。汉诺塔有 64 个盘子时，盘子移动次数为：$2^{64}-1=18\,446\,744\,073\,709\,551\,615$。汉诺塔递归程序是一种串行计算，第 $n$ 步运算依赖于第 $n-1$ 步运算，因此在理论上不存在并行计算的可能性。汉诺塔问题虽然在理论上是可计算的，但是计算时间、递归深度和内存容量，都大大超出了当前计算机的能力。**可见即使是理论上可计算的问题，也要考虑计算量是否超出了目前计算机的能力**。

### 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。

**1. 哥德尔不完备性定理**

完备的中文语义是完整的意思，简单地说就是该有的都有了。例如，整数集合中，加减乘运算是完备的，因为它们的运算结果仍然是整数；但是除法在整数集中是不完备的，因为两个整数相除的结果不一定是整数。数学公理体系的完备性是该体系中有足够个数的公理，以此为依据可推导(或判定)出该体系的全部结论或命题。

20 世纪以前，大多数数学家认为所有数学问题都有算法(完备性)，只是有些问题的算法目前还没有找到。1928 年国际数学大会上，数学家希尔伯特(David Hilbert)提出了数学的三个精辟问题：**数学是完备的吗？数学是一致的吗？数学是可判定的吗？**

哥德尔首先回答了希尔伯特的前两个问题。1931 年，奥地利数学家哥德尔发表了不完备性定理。哥德尔第一定理认为：**任何逻辑自洽的算术公理系统，必定是不完备的**；其中一定有真命题，但是在体系中不能被证明(即体系是不完备的)。

哥德尔第二定理认为：任何逻辑自洽的算术公理系统，不能用于证明本身的自洽性(即不能自己证明自己)。

哥德尔不完备性定理说明，**任何一个形式化的系统，它的一致性和完备性不可兼得**。公理不完备可能是公理个数太少，导致有些正确的命题不能被公理推出；公理的不一致可能是公理个数太多，导致公理之间互相矛盾。这样一致性和完备性互为悖论。

数学可判定性问题是能否找到一种方法，仅仅通过机械的计算就能判定某个数学陈述是对是错？图灵回答了希尔伯特的第三个问题，图灵证明了一个不可计算的数，实际上就是一个不可判定的问题，数学的不可判定性来源于有些数是不可计算的。

**2. 哪些数是可计算数**

图灵提出了一个在他之前几乎没有人考虑过的问题：所有数都是可计算的吗？图灵在一篇论文中指出："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。1995年，贝利、波尔温、普劳夫(David Bailey、Peter Borwein、Simon Plouffe，BBP)共同发明了BBP公式，它可以计算圆周率中的某一位，BBP公式如下

$$\pi=\sum_{k=0}^{\infty}\left[\frac{1}{16^k}\left(\frac{4}{8k+1}-\frac{2}{8k+4}-\frac{1}{8k+5}-\frac{1}{8k+6}\right)\right] \tag{3-9}$$

式中，$k$是π需要计算的小数位。BBP公式的特点是求π的第$n$位小数时，不需要计算出它的前$n-1$位；而且，BBP公式是线性收敛的，这使得它非常适合多线程并行计算。因此，BBP公式在计算圆周率时具有较高的效率和实用性。

**【例3-41】** 用BBP公式计算π值到小数点后第100位。Python程序如下。

```
from decimal import Decimal, getcontext                    # 导入标准模块-精确计算函数
getcontext().prec = 102                                    # 设计算精度为102位(很重要)
N = int(input('请输入需要计算到小数点后第n位:'))            # 输入计算位数
pi = Decimal(0)                                            # 初始化pi值为精确浮点数
for n in range(N):                                         # 循环计算pi精确值
    k = Decimal(n)                                         # 迭代变量为精确浮点数(很重要)
    pi += Decimal(1/pow(16,k)*(4/(8*k+1)-2/(8*k+4)-1/(8*k+5)-1/(8*k+6)))
                                                           # BBP公式
print(f'小数点后第{N}位的pi值为:', pi)                      # 打印pi值
>>>
请输入需要精确到小数点后第n位:100                          # 程序输出
小数点后第100位的pi值为:3.1415926535  8979323846  2643383279  5028841971
  6939937510  5820974944  5923078164  0628620899  8628034825  3421170679  8
```

**3. 不可计算的蔡廷常数**

圆周率π、毕达哥拉斯数$\sqrt{2}$、黄金比例$\phi$等都是可计算数，尽管它们是无理数。可计算数可以定义为：可以通过有限终止算法计算到任何所需精度的实数。

1975年，计算科学专家格里高里·蔡廷(Gregory Chaitin)提出了一个问题：选择任意一种程序语言，随机输入一段程序，这段程序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"。

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。蔡廷常数是一个明确定义的数，目前已在理论上证明了蔡廷常数就是不可计算数，它永远也无法计算出来。

图灵在论文"论可计算数及其在判定问题中的应用"中证明，对于所有可能的程序输入，解决停机问题的一般算法是不存在的，即停机的概率无法计算。因为停机问题是不可判定的，所以蔡廷常数无法计算。

**4. 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。"在数学上也存在同样类似的问题：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，就能够得到数学问题的最终答案呢？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。1930 年前后，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。

可计算问题都可以通过编码的方法，用函数的形式表示。因此可以通过定义在自然数集上的有效可计算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有效可计算函数。

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有效可计算函数相同。

1936 年，哥德尔、丘奇等定义了递归函数。丘奇指出：有效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，在有限步骤内可以计算结果的函数都是递归函数。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可计算函数相同。

一个显而易见的事实是：数学精确定义的可计算函数都是可计算的。问题是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：**任何能有效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。**

由于有效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，也因此称其为“丘奇—图灵猜想”，该论题被普遍假定为真。

**5. 不可计算问题的类型**

不可计算的问题有以下类型：**第一类是理论上不可计算的问题**，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、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，又称丢番图方程)等，这些问题都是计算能力的理论限制。**第二类是现实中不可计算的问题**，计算机的速度和存储空间都有限制，例如，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。**第三类是定义模糊难以计算的问题**，例如，“做一个西红柿炒鸡蛋”这样一个概念模糊的命题也是不可计算的。

## 3.3.5 计算科学难题：P=NP?

**1. P 问题**

有些问题是确定性的，例如，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**多项式(polynomially，P)**问题。**P 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

重之和最小的生成树)等,这些问题都能在多项式时间内解决。

**2. NP 问题**

非确定性多项式(nondeterministic polynomially,NP)问题中的 N 指非确定的算法。有些问题是非确定性问题,例如,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 3-42】** 验证某个连通图中的一条路径是不是哈密顿回路(Hamilton)很容易;而问某个连通图中是否不存在哈密顿回路则非常困难,除非穷举所有可能的哈密顿路径,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 NP 完全(NP-complete,NPC)问题。NPC 是比 NP 更难的问题。

**【例 3-43】** 对任意的布尔可满足性问题(SAT),总能写成以下形式

$$(x_1 \vee x_2 \vee x_3) \wedge (x_4 \vee \bar{x}_5 \vee x_n)\cdots$$

式中,∨表示逻辑或运算;∧表示逻辑与运算;－表示逻辑取反运算;表达式中 $x$ 的值只能取 0 或者 1。那么当 $x$ 取什么值的时候,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:$(x_1 \vee \bar{x}_2 \vee x_3) \wedge (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee x_3)$,当 $x_1=0$,$x_2=0$,$x_3=$任意值(0 或 1)时,表达式的值为真。一个简单的 SAT 问题处理起来尚且如此麻烦,可见对任意 SAT 问题的求解非常困难。SAT 的典型应用有数独游戏、扫雷游戏等。

NPC 问题目前没有多项式算法,只能用穷举法一个一个地检验,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呈指数级增长,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、哈密顿回路问题、旅行商问题、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 难(NP-hard)问题。NP-hard 太难了,例如,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 3-20 所示。

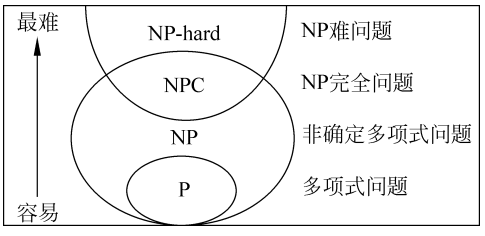



图 3-20 计算科学中的难解问题

**【例 3-44】** 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;中国象棋估计有 $10^{150}$ 种博弈状态;围棋达到了惊人的 $10^{360}$ 种博弈状态。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。1971 年，斯蒂文考克(Stephen Cook)提出了 P=NP? 猜想。究竟 P=NP? 还是 P≠NP? 迄今为止这个问题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的问题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性的问题，而是通过弱化有关条件，将问题限制得特殊一些，再解决这个普遍性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。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

数学家希尔伯特指出："**任何一门学科，只要它能提供丰富的问题，它就是有生命力的；**相反地，如果问题贫乏，那就预示着这一学科的独立发展已经趋向消亡和终止。"计算科学的发展中，科学家们提出过许多具有深远意义的问题和经典案例，对这些问题的深入研究为计算学科的发展起到了十分重要的推动作用。

### 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(见图 3-21)。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，七座桥所有的走法一共有 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，有几名大学生写信给瑞士数学家欧拉(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(见图 3-22)，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。

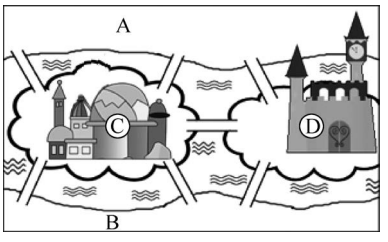


图 3-21 哥尼斯堡七桥问题

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2个奇点的连通图,以奇点为起点可以一笔画成此图(见图3-23)。

(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。

说明:如图3-23所示,某个顶点的边是奇数个称为奇点;反之称为偶点。

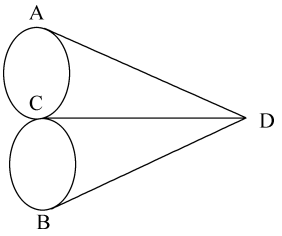



图3-22 抽象后的七桥路径

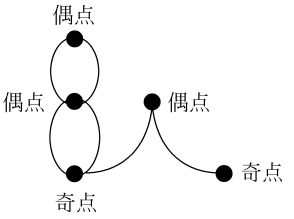



图3-23 图的奇点和偶点

由以上分析可见,哥尼斯堡七桥的路径抽象图中(见图3-22),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,或者说它不是欧拉图。

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,如深度优先搜索(depth first search,DFS)、广度优先搜索(breath first search,BFS)、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 哈密顿回路:计算复杂性

计算科学中,很多问题都涉及复杂系统的求解。例如,天气预测、癌细胞基因突变等问题,都与复杂系统有关。由于复杂系统求解时容易引入不确定性,这会导致需要极大的计算资源。1857年,数学家威廉·罗恩·哈密顿(William Rowan Hamilton)设计了一个名为"周游世界"的木制玩具(见图3-24),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3-25),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,这就形成了一条哈密顿回路(见图3-26)。

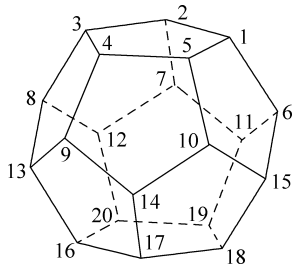



图3-24 周游世界玩具

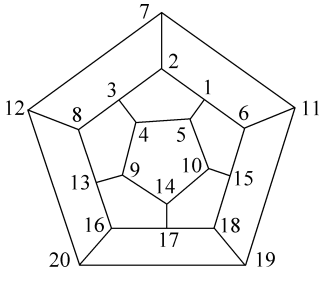



图3-25 抽象后的平面图

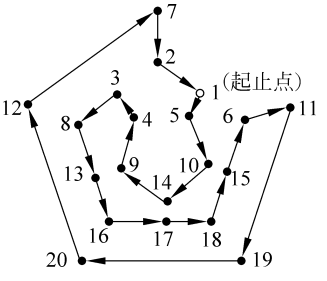



图3-26 一条哈密顿回路

哈密顿回路与欧拉回路看似相同，本质上是完全不同的问题。哈密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 3-22 所示不存在欧拉回路，但是存在多个哈密顿回路（如 A→C→B→D→A）。

如果经过图中的每个顶点恰好一次后能够回到出发点，这样的图称为哈密顿图。所经过的闭合路径就形成了一个圈，它称为哈密顿回路或者哈密顿圈（见图 3-27、图 3-28）。图中一个顶点的度数是指这个顶点所连接的边数，如图 3-27 所示，顶点 B 的度为 4。哈密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，则此图是哈密顿图；若图中每一对不相邻顶点的度数之和不小于顶点数，则为哈密顿图。也有一些图中不存在哈密顿回路（见图 3-29）。

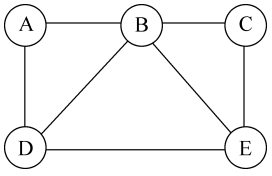


图 3-27　有哈密顿回路示例 1

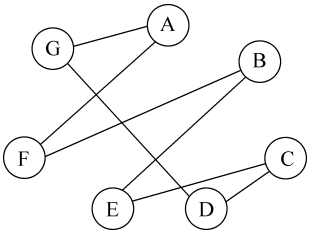


图 3-28　有哈密顿回路示例 2

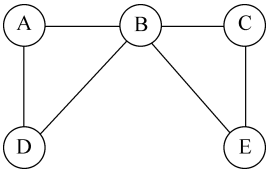


图 3-29　没有哈密顿回路

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

欧拉回路和哈密顿回路在任务排队、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

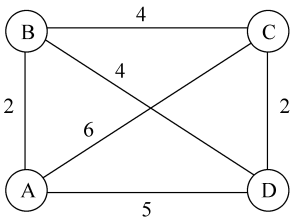
旅行商问题（traveling salesman problem，TSP）由哈密顿和英国数学家柯克曼（T. P. Kirkman）共同提出，它是指有若干个城市，任意两个城市之间的距离可能不同或相同；如果一个旅行商从某一个城市出发，依次访问每个城市一次，最后回到出发地，问旅行商应当如何行走经过的路程总长才会最短。

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，列出所有路径，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 3-30 所示，从城市 A 出发，再回到城市 A 的路径有 6 条，最短路径为：A→D→C→B→A。

求解旅行商问题比求哈密顿回路更困难。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，路径组合数会呈现指数级增长，计算时间的增长率也呈指数增长，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，或称为“维数灾难”（随着多项式中变量维数地增加，计算量呈指数倍增长）。

如图 3-30 所示，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




| 序号 | 路　径 | 路径长度 | 最短路径 |
|---|---|---|---|
| 1 | A→B→C→D→A | 13 | 是 |
| 2 | A→C→B→D→A | 19 | 否 |
| 3 | A→D→C→B→A | 13 | 是 |
| 4 | A→B→D→C→A | 14 | 否 |
| 5 | A→C→D→B→A | 16 | 否 |
| 6 | A→D→B→C→A | 19 | 否 |

图 3-30　4 个城市的旅行商问题示意图

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 3-45】** 城市数为 26 时,求所有城市的全部路径数。Python 程序如下。

```
>>> import math                          # 导入标准模块
>>> math.factorial(25)                   # 路径数为(26-1)的阶乘值
15511210043330985984000000               # 所有路径数 = 1.5×10^25,计算量发生了组合爆炸
```

说明:很多旅行商求解程序都简化了问题,这些程序只考虑一个城市与周边 2～3 个城市有连接路线,而不是考虑每个城市与其他所有城市都有连接路线。求解程序见配套教学资源程序 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精力的事情,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。如果能理解蜜蜂怎样做到这一点,对解决旅行商问题将有很大帮助。

旅行商问题有以下应用,例如,车载全球定位系统(global positioning system,GPS)中,经常需要规划行车路线,怎样做到行车路线距离最短,就需要对 TSP 问题求解;在印制电路板(printed circuit board,PCB)设计中,怎样安排众多的导线使线路距离最短,也需要对 TSP 问题求解。旅行商问题在其他领域也有普遍的应用,如物流运输规划、网络路由节点设置、遗传学领域、航天航空领域等。

计算科学中图论经典问题之间的区别如表 3-8 所示。

**表 3-8　计算科学图论经典问题的区别**

| 问 题 要 求 | 哥尼斯堡七桥问题 | 哈密顿回路问题 | 旅行商问题(TSP) |
|---|---|---|---|
| 点遍历 | 全部遍历,允许重复 | 全部遍历,不允许重复 | 全部遍历,不允许重复 |
| 边遍历 | 全部遍历,不允许重复 | 部分遍历,不允许重复 | 部分遍历,不允许重复 |
| 边权值遍历 | 边长无关,权值=1 | 边长无关,权值=1 | 边长不一,权值不同 |
| 求解问题 | 求是否存在遍历路径 | 求节点遍历最短回路 | 求节点遍历最短距离 |

## 3.4.4　单向函数:公钥密码的基础

### 1. 单向函数的概念

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,要比把牙膏挤出来困难得多。类似的,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 3-46】 如图 3-31 所示，已知 $x$ 计算 $f(x)$容易；但是已知 $f(x)$，求 $x$ 很难。

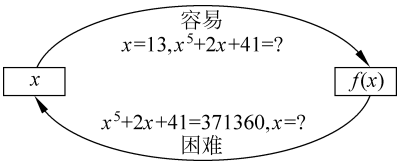


图 3-31 单向函数求解

**2. 单向函数的假设**

单向函数听起来很好，但事实上严格地按数学定义，我们不能从理论上证明单向函数的存在性。因为单向函数的存在性证明将意味着计算科学中最具挑战性的猜想 P=NP，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，因此我们假定存在单向函数。

最简单的单向函数就是整数相乘。不管两个多大的整数，我们很容易计算出它们的乘积；但是对两个 100 位的素数之积，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**

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，就能很容易在另一个方向计算这个函数。也就是说，已知 $x$，易于计算 $f(x)$；而已知 $f(x)$，却难于计算 $x$。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，把钟表拆成数百个小零件很容易，而把这些小零件组装成能够工作的钟表非常困难，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。

【例 3-47】 表达式 $88^7 \bmod 187 = 11$ 中，假设明文 $M$ 为 88，密文 $C$ 为 11，公钥 KU（陷门）为(7,187)，私钥 KR（陷门）为(23,187)。

(1) 表达式 $88^{(a)} \bmod (b) = 11$ 中，已知陷门 $a=7, b=187$ 时，求密文 $C=11$ 很容易。

(2) 表达式 $11^{(a)} \bmod (b) = 88$ 中，已知密文为 $C=11^{(a)}$，明文为 $M=88$，但是不知道陷门（私钥）$a$，$b$ 时，利用表达式求陷门 $a$、$b$ 非常困难。

(3) 表达式 $11^{(a)} \bmod 187 = (M)$中，已知密文 $C=11^{(a)}$，公钥 KU=187，但是不知道陷门（私钥 KR）$a$ 时，利用表达式求明文 $M$ 也非常困难。

**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。**所有公钥加密算法的关键在于它们有各自独特的单向陷门函数**。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**（Ron Rivest、Adi Shamir、Leonard Adleman 三人的姓氏）**密码算法就是根据单向陷门函数设计的**。

## 3.4.5 哲学家就餐问题：死锁控制

**1. 哲学家就餐问题**

1965 年，迪杰斯特拉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：有五个哲学家，他们的生活方式是交替地进行思考和进餐。哲学家共用一张圆桌，分别坐在周围的五张椅子上，圆桌上有五个碗和五支餐叉（见图 3-32）。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。

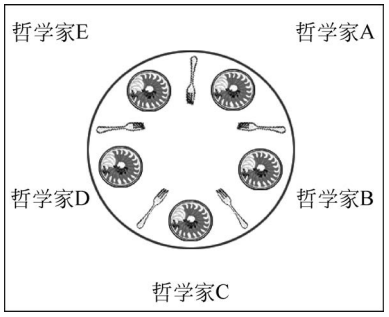



图 3-32　哲学家就餐问题

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只餐叉时才能吃饭;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只餐叉前,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,导致死锁现象发生。

**2. 哲学家就餐问题的解决方法**

(1) 资源加锁。资源加锁可以使资源只被一个进程访问,当一个进程想要使用的资源被另一个进程锁定时,它必须等待资源解锁。当多个进程涉及加锁资源时,在某些情况下仍然有可能会发生死锁;而且,这个算法的资源利用率较低。

(2) 服务生法。引入一个餐厅服务生(监督进程),哲学家必须通过他的允许才能拿起餐叉(分配资源),因为服务生知道哪只餐叉正在使用,他能作出判断,避免死锁。这个算法实现简单,但效率不高,当多个哲学家同时需要餐叉时,服务员只好随机分配。

(3) 资源分级法。为餐叉(资源)分配一个优先级,约定优先级最高的餐叉优先分配,这样就不会有两个资源同时被同一个进程获取。这种方法能避免死锁,但并不实用,特别是事先并不知道进程所需资源的情况下,难以实现资源分配。

由以上分析可知,系统死锁没有完美的解决方案。**解决系统死锁需要频繁地进行死锁检测,这会严重降低系统运行效率,以至于得不偿失**。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。可见鸵鸟算法只是一种折中策略。

## 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? Tanenbaum 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 3-33 所示,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。

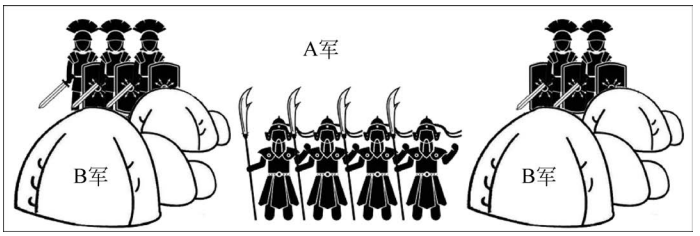



图 3-33　两军通信问题：B 军之间如何实现安全可靠的通信

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。**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 军通信兵被 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。**通信信道不可靠时，没有确定型算法能实现进程之间的协同**。所以，TCP 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，例如，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 1～10s）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　为什么圆周率 π 是可计算数？

3-4　不可计算的问题有哪些类型？

3-5　简要说明不可计算问题的解决方法。

3-6　简要说明计算复杂性理论的研究内容。

3-7　开放题：例 3-40 中，用计算机模拟搬运汉诺塔中的盘子，如果计算机 1ms 可以计算搬运一次盘子，搬运一个有 64 个盘子的汉诺塔，一共需要计算多长时间？

3-8　开放题：写出“石头—剪子—布”游戏的博弈策略矩阵。

3-9　开放题：“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？

3-10　实验题：参考本书程序案例，实现囚徒困境的博弈。