第3章

计算复杂性

计算复杂性是计算机科学中极其重要的一个领域，它不但包含了一个完整独立且内容丰富的理论，同时也对许多其他有关的计算机和应用数学领域产生了重大影响。计算复杂性理论使用数学方法对计算中所需的各种资源的耗费做定量的分析，并研究各类问题之间在计算复杂程度上的相互关系和基本性质，是算法分析的理论基础。

计算复杂性理论在过去 30 多年中发展迅猛，自 20 世纪 90 年代以来取得了令人瞩目的结果。这些结果涉及的领域非常广泛，包括经典复杂性类的概率型新定义（**IP=PSPACE** 和各种 PCP 定理）以及它们在近似算法中的应用、Shor 因子分解算法、对于 **P≠NP** 问题的理解、去随机化理论、基于计算难度的伪随机性、随机性提取器、伪随机对象的构造。

算法是计算机科学的核心思想，算法的基本模型是图灵机（Turing Machine），它是由英国数学家艾伦·图灵（Alan Turing）提出的一种抽象计算模型，用来精确执行算法。图灵机不仅可以衡量可计算性，而且可以衡量问题的计算复杂性。研究算法所要面临的一个基本问题是完成计算任务需要多少资源。这个问题包括两个方面：第一，计算可能性，也就是说什么样的任务是能计算的，若能计算，则存在求解特定问题的具体算法，例如升序排序一组数的算法；第二，计算最优性，它可以衡量计算任务的计算效率，例如给出升序排序一组数的任何算法所要执行的操作步数的下限。最理想的情况应该是找到解决计算问题的算法正好达到解决计算问题的能力极限，但实际上，这两者之间存在较大差距。

引入计算机科学中的计算复杂度理论对于量子计算的研究具有重要的意义。第一，经典计算机科学提供了大量的概念和技术，它们可以非常成功地被用到量子计算研究中，量子计算的许多成功来源于计算机科学现有思想和量子力学新思想的结合。例如很多量子算法都基于量子傅里叶变换，而量子傅里叶变换实现了对经典傅里叶变换的加速。第二，经典计算机上执行给定计算任务需要资源量可以作为量子计算研究的参照物，例如因子分解问题已引起很多关注。人们认为在经典计算机上，这个问题没有有效解，然而这个问题在量子计算机上存在有效解法。我们感兴趣的是关于求素因子的问题，经典计算机和量子计算机之间的计算能力存在什么样的差距。这类差距可能存在于一大类计算问题中，而非仅仅是寻

找因子，通过对这个特殊问题的进一步研究，也许能够找出它在量子计算机上比在经典计算机上更容易求解的特征，并根据这些灵感继续寻找解决其他问题的量子算法。第三，以计算机科学家的方式思考问题。计算机科学家和物理学家或其他自然科学家的思维方式相当不同，而且量子计算是多学科交叉的领域，通过结合计算机科学家的思考方式将有助于更深入地理解量子计算。

本章的安排如下：3.1 节介绍两类计算模型，即图灵机模型和线路模型；3.2 节讨论计算复杂性，研究解决特定计算问题时所需的时间和空间资源，它根据求解的难度对问题进行大致分类，包括 **P** 类和 **NP** 类等；3.3 节通过回顾计算科学的发展以及存在的问题结束本章。

3.1　计算模型

建立数学模型可谓难上加难，这是由于历史上人类在解决各种计算任务的过程中用尽了各种各样的方法——从直觉和灵感到算盘或计算尺，再到现代计算机。此外，自然界中其他生物或系统也时刻需要处理各种计算任务，而它们的解决之道也是纷乱繁杂。怎样才能找出一个能抓住这些计算方法共性的简洁数学模型呢？如果再考虑本书要关注的计算效率问题，则建模问题就更加无从下手了。考虑计算效率问题似乎必须小心地选择计算模型，因为即便是儿童也知道一款新的游戏是否能“高效运行”将依赖于他的计算机硬件。

本节将采用图灵机模型和线路模型，引入两种计算模型的原因是由于不同的计算模型在处理特定问题时会提供不同的视角，从两种（或更多）途径思考同一概念比只通过一种途径更好。为了更好地表示这些模型，下面首先给出一些记号约定。

1. 对象的字符串表示

本章所讨论的基本计算任务主要是针对函数的计算，而且这些函数大多是输入和输出均是长度有限的 0-1 位串（即属于 $\{0,1\}^*$，这里的 $*$ 表示位串长度）函数。仅考虑位串上的操作并未真正限制讨论的范围，因为只需要通过编码就可以将整数、整数对、图、向量、矩阵等一般对象表示为位串。例如，整数可以表示为二进制形式（比如，40 可以表示为 101000），图可以表示为邻接矩阵（即一个包含 n 个顶点的图 G 可以表示为一个 $n \times n$ 的 0/1 矩阵 $\boldsymbol{A}$，其中 $\boldsymbol{A}_{ij}=1$，当且仅当图 G 中存在边 ij）。忽略详细的底层处理的表示过程，而简单地用 $\underline{x}$ 表示对象 x 在未明确指定的某种规范方法下的二进制表示。

2. 判定问题或语言

在所有位串映射为位串的函数中，布尔函数的输出只有一个位。把根据布尔函数 f 确定的一个集合 $L_f=\{x\in\{0,1\}^*|f(x)=1\}$，称之为语言或判定问题。

3. O、Θ 和 Ω 记号

算法的计算效率一般是将该算法执行基本操作的个数，表达为该算法输入长度的函

数，即 $T(n)$ 是算法在所有长度为 n 的输入上执行基本操作的最大个数，其中函数 T 的形式依赖于基本操作的基本定义。

定义 3.1 设函数 $f\colon N \to N$，$g\colon N \to N$。

（1）如果存在常数 c，使得 $f(n) \leqslant c \cdot g(n)$ 对充分大的 n 均成立，则称 $f = O(g)$。

（2）如果 $g = O(f)$，则称 $f = \Omega(g)$。

（3）如果 $f = O(g)$ 且 $g = O(f)$，则称 $f = \Theta(g)$。

3.1.1 图灵机

下面介绍 k-带图灵机的概念，其工作示意图如图 3.1 所示。

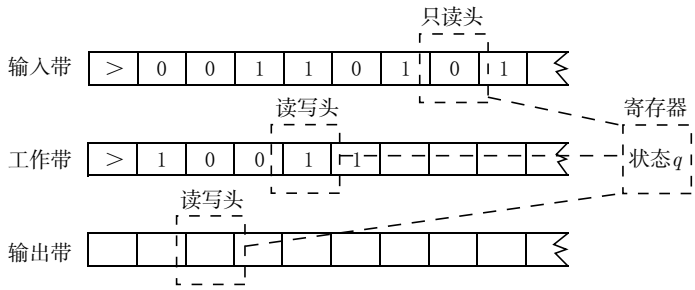

图 3.1 具有输入带、工作带和输出带的 3-带图灵机

演草纸包含 k 条带，每条带均由单向无限延伸的很多单元构成，每个单元能够存储称为机器的字母表的有限集 Γ 中的一个符号。每条带均有一个带头，它具有在带上每次读或写一个符号的能力。机器的计算划分为一系列离散的时间步骤，带头在每个步骤中能够向左或向右移动一个单元。机器的第一条带是输入带，其带头只能从该带上读取符号，而不能写符号，因此这是一条只读带。另外的 $k-1$ 条读写带称为工作带，其中最后一条带是输出带，机器在计算终止前把最终计算结果写在输出带上。

图灵机有有限种状态，表示为 Q。机器有一个“寄存器”，它能够在任何时刻记录机器处于 Q 中的何种“状态”。状态确定了机器在下一个计算步骤要采取的动作，包括：①直接读取 k 个带头所在存储单元的符号；② 在 $k-1$ 条读写带上将带头所在存储单元的符号替换为新的符号（也可以通过再次写下原来的符号而不改变带）；③ 修改寄存器使其记录来自有限集 Q 中的另一种状态（状态也可以保持不变，只需要选择与之前相同的状态）；④ 将每个带头向左或向右移动一个单元（或保持不动）。

图灵机 M 的数学形式定义为一个三元组 (Γ, Q, δ)。其中，有限集 Γ 包含 M 的存储带上允许出现的所有符号。同时，假设 Γ 中还包含：一个特定的“空白符”，记为 $\square$；一个特定的“开始符号”，记为 $\triangleright$；以及 0 和 1。Γ 称为 M 的字母表。有限集 Q 包含 M 的寄存器中可能出现的所有状态。此外，假设 Q 还包含一个特定的开始状态 q_{start} 和一个特定的终止状态 q_{halt}。函数 $\delta : Q \times \Gamma^k \to Q \times \Gamma^{k-1} \times \{L, S, R\}^k$ 描述了 M 在各个步骤中使用的规则，其中，函数 δ 也称 M 的转移函数。

如果图灵机的状态为 $q \in Q$，k 条带上当前读到的符号是 $(\sigma_1, \sigma_2, \cdots, \sigma_k)$，且满足

$$\delta(q, (\sigma_1, \sigma_2, \cdots, \sigma_k)) = (q', (\sigma'_2, \cdots, \sigma'_k), z) \tag{3.1.1}$$

其中，$z \in \{L, S, R\}^k$，则在下一个步骤中，后 $k-1$ 条带上的各个符号 σ 将被相应地替换为 σ'，图灵机将进入状态 q'，且 k 个带头将根据 z 相应地向左（L）、向右（R）移动或保持不动（S）（如果带头要在位于带的最左端的位置向左移动，则保持不动）。除输入带之外，所有带的第一个单元初始化为开始符号 $\triangleright$，而其余单元则初始化为空白符 $\square$。输入带的第一个单元初始化为开始符号 $\triangleright$，后续单元存储长度有限的非空符号串 x（即“输入”），其余所有单元都是空白符 $\square$。所有带头位于带的左端，而机器处于特殊开始状态 q_{start}。这就是图灵机在输入 x 上的初始格局。计算过程的每个步骤就是如前段所述那样应用一次转移函数 δ。机器一旦处于特定的终止状态 q_{halt} 下，转移函数 δ 将不再允许机器修改带上的内容或改变机器的状态。显然，机器进入状态 q_{halt} 就已经停机。

在复杂性理论中，人们通常只关注在任意输入上经过有限个操作步骤必然停机的图灵机。因此，我们给出图灵机在函数计算任务上运行时间的定义。

定义 3.2　令 $f: \{0,1\}^* \to \{0,1\}^*$ 和 $T: N \to N$ 是两个函数，M 是一个图灵机。如果将 M 初始化为任意输入 $x \in \{0,1\}^*$ 上的初始格局，则 M 停机时将 $f(x)$ 写在它的输出带上，就称 M 计算 f。如果 M 在任意输入 x 上计算 $f(x)$ 时最多只需要 $T(|x|)$ 个步骤，则称 M 在 $T(|x|)$ 时间内计算 f。

根据图灵机的定义，存在很多可能的图灵机，每个图灵机都根据转换函数计算特定的任务。然而图灵（Turing）注意到了通用图灵机的存在，他证明了给定任意图灵机 M 的位串表示作为输入，通用图灵机可以模拟 M 的运行。通用图灵机的各种参数均是固定的，包括字母表的大小、状态的数量和带的数量。被模拟的图灵机的各项参数均可能比通用图灵机的大得多。这之所以不是障碍，得益于编码的能力。即使通用图灵机的字母表很简单，其他图灵机的状态和转移函数也可以在编码后存放于通用图灵机的带上，然后由通用图灵机一步一步地模拟执行。

接下来，我们不加证明地给出亨尼（Hennie）和斯特恩斯（Stearns）提出的高效通用图灵机构造。

定理 3.1　存在图灵机 U 使得 $U(x, \alpha) = M_\alpha(x)$ 对于任意 $x, \alpha \in \{0,1\}^*$ 成立，其中 M_α 为 α 表示的图灵机。而且，如果 M_α 在输入 x 上至多运行 T 步之后就停机，则 $U(x, \alpha)$ 将在 $CT \log T$ 步内停机，其中 C 是一个独立于 $|x|$ 而仅依赖于 M_α 的字母表大小，即带和状态数量的常数。

通用图灵机 U 使用字母表 $\{\triangleright, \square, 0, 1\}$，除了输入带和输出带之外，它还使用 3 条工作带（见图 3.2）。U 按照与 M 相同的方式使用其输入带、输出带和一条工作带。此外，U 用第 2 条工作带存储 M 的转移函数值的表，用第 3 条工作带存储 M 的当前状态。为了模拟 M 的一个计算步骤，U 先通过扫描 M 的转移函数表和当前状态确定 M 的下一个状态和要写出的符号以及带头移动方向，然后执行相应的动作。M 的每个计算步骤都需要通过 U 的 C 个计算步骤进行模拟，其中 C 依赖于 M 转移函数表的规模。

图 3.2　通用图灵机

通用图灵机的存在自然而然地导致了不可计算性这一问题。如果一个函数不能由图灵机计算，那么它就是不可计算的，也就是说，对于任何给定的输入，没有图灵机停止并输出正确的答案。下面给出不可计算函数的存在性定理。

定理 3.2　*存在不能被任意图灵机计算的函数* $\mathrm{UC}:\ \{0,1\}^* \to \{0,1\}^*$。

证明　函数 UC 定义如下：对任意 $\alpha \in \{0,1\}^*$，如果 $M_\alpha(\alpha)=1$，则 $\mathrm{UC}(\alpha)=0$；否则，若 M_α 为其他值或进入无限循环，则 $\mathrm{UC}(\alpha)=1$。

用反证法假设 UC 是可计算的，因此存在图灵机 M 使得 $M_\alpha(\alpha)=\mathrm{UC}(\alpha)$ 对任意的 $\alpha \in \{0,1\}^*$ 成立。于是，有 $M(\underline{M})=\mathrm{UC}(\underline{M})$。但是，由函数 UC 的定义可知，$\mathrm{UC}(\underline{M})=1 \Leftrightarrow M(\underline{M}) \neq 1$，产生矛盾。∎

下面介绍一个更自然的不可计算函数。函数 HALT 以序对 $\langle \alpha, x \rangle$ 为输入，它输出 1 当且仅当 α 表示的图灵机 M 在输入 x 上会在有限步骤内停机。这肯定是人们需要计算的函数。因为在给定一个计算机程序和输入之后，人们肯定希望知道该程序在该输入上是否会陷入无限循环。如果计算机能够计算函数 HALT，则设计无 Bug 的计算机软件和硬件将变得容易得多。遗憾的是，现在可以证明计算机计算不了这个函数，即使运行时间允许任意的长。

定理 3.3　*函数 HALT 不能被任何图灵机计算。*

证明　用反证法。假设存在计算函数 HALT 的图灵机 M_{HALT}。用 M_{HALT} 构造一个计算函数 UC 的图灵机 M_{UC}，这与定理 3.2 矛盾。

图灵机 M_{UC} 的构造如下：对于输入 α，M_{UC} 运行 $M_{\mathrm{HALT}}(\alpha,\alpha)$；若结果为 0，则意味着 M_α 在 α 上不停机，M_{UC} 输出 1；否则 M_{UC} 用通用图灵机 U 计算 $b=M_\alpha(\alpha)$。如果 $b=1$，则 M_{UC} 输出 0；否则 M_{UC} 输出 1。

在 $M_{\mathrm{HALT}}(\alpha,\alpha)$ 于有限步内输出 $\mathrm{HALT}(\alpha,\alpha)$ 的假设下，图灵机 M_{UC} 输出 $\mathrm{UC}(\alpha)$。∎

我们已经介绍过了经典图灵机，然而经典图灵机并不能有效地模拟量子力学的过程，因此迫切需要发展量子计算模型以模拟量子系统的演化过程。Deutsch 提出了量子图灵机模型以及量子计算复杂性理论。随后, Bernstein 和 Vazirani 对该模型进行了数学形式上的严格描述。

量子图灵机（Quantum Turing Machine, QTM）的数学形式定义为一个七元组 $(Q,\Sigma,\Gamma,$

$\delta, q_0, q_Y, q_N)$，其中 Q 是有限状态集，$q_0, q_Y, q_N \in Q$ 为初始状态、接受状态和拒绝状态，Γ 为包含带中允许出现的所有符号的一个有限集，Σ 为输入字符表集合，δ 为量子状态转移函数。函数 δ 的映射为

$$\delta : Q \times \Gamma \times Q \times \Gamma \times \{L, S, R\} \to \bar{C} \tag{3.1.2}$$

满足 $\sum_{(q_2,a_2,d)\in Q\times\Gamma\times\{L,S,R\}} \|\delta(q_1,a_1,q_2,a_2,d)\|^2 = 1$。$\bar{C}$ 为复数集的子集，其实部和虚部都是多项式时间可计算的，即存在确定性多项式时间函数 $f(n)$ 的算法精确计算复数 $z \in \bar{C}$ 的实部和虚部，使得 $\|f(n) - z\| < 2^{-n}$。$\{L, S, R\}$ 表示读写头的移动方向为向左移动、不移动或向右移动，称 $\delta(q_1, a_1, q_2, a_2, d)$ 为格局。量子图灵机在状态 q_1 读取字符 a_1，沿方向 d 进入状态 q_2 读取字符 a_2。转移函数指定了无限维空间格局叠加态的线性映射 M_δ（时间演化算子）。

量子图灵机的工作原理如下所述。设 S 为量子图灵机的格局且 S 是有限线性组合上满足欧几里得归一化条件的复内积空间，称 S 中的每个元素为 M 的一个叠加。量子有限状态转移函数 δ 诱导一个时间演化算子 $U_M : S \to S$。量子图灵机以格局 c 起始，当前状态为 p，且扫描标识符 σ，下个动作 M 将被置为格局叠加：$\psi = \sum_i z_i c_i$，其中，每个非零的 z_i 都与一个量子转移函数 $\delta(q_1, a_1, q_2, a_2, d)$ 对应，c_i 是向 c 施行转化得到的新格局。通过线性时间演化算子 U_M 可以将这种操作扩展到整个 S 空间。

此外还有学者相继提出广义量子图灵机（GQTM）、随机语言量子图灵机（ROQTM）以及多带量子图灵机模型（MQTM），它们可以看成是量子图灵机的变形，在计算能力上存在着一定的差异。GQTM 的计算能力涵盖了 QTM，QTM 的计算能力涵盖了 ROQTM，在二次多项式时间内能够通过一个单带的 QTM 模拟 MQTM。

量子图灵机继承了经典的图灵机（包括概率图灵机）的一些要素，但也有自身的特性。量子图灵机实质上是由量子读写头和一条无限长度的量子带构成的。量子带上的每个单元格代表一个量子记数位，其状态可以是 0 和 1 的叠加形式，以便在量子带上对编码问题的多个输入进行并行计算，计算得到的结果为所有输入相应结果的一个叠加，最后进行测量得到经典的结果。另一个不同之处是状态转移函数的变化，量子图灵机中，量子状态转移函数的存在使得执行 T 步计算仅需精度为 $O(\log T)$ 位的转移幅度。量子图灵机与经典图灵机的本质差异是实现量子并行计算，这与量子相干性在量子图灵机中的作用密切相关。

3.1.2　线路模型

图灵机是相当理想化的计算模型，实际的计算机的大小总是有限的，我们假设的图灵机的大小是无限的。本节使用另一个通用模型（称为布尔线路模型），该模型在计算能力上等同于图灵机，但在许多应用中更方便也更接近实际，特别地，计算的线路模型是对量子计算机的研究有特殊重要性的预备知识。一个布尔线路是一个有向非循环图，其结

点与布尔函数相关联，这些结点有时称作逻辑门。一个具有 n 条输入线和 m 条输出线的结点与函数 $f:\{0,1\}^n \to \{0,1\}^m$ 关联。这里有一个简单的例子，如图 3.3 所示。

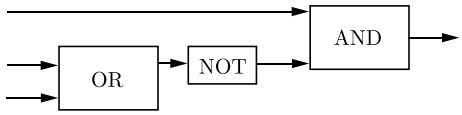

图 3.3　一个简单的布尔线路例子

给定一些位串作为输入，这些线携带位的值，直到到达一个结点为止。结点是计算位的逻辑函数（该结点为非门、或门、与门等）。结点的输出线将输出位传送到下一个结点，直到计算在输出线处结束。输入线可以携带常数，这些常数不会随线路的不同输入而变化，而是线路硬件的一部分。在图灵机中，转移函数是局部的，因此操作是一系列基本步骤。在线路模型中，同样要求逻辑门是局部的，也就是说，每个结点操作的导线数是有限的。

现在观察这些逻辑门的通用性，事实上可以证明用固定数目的门就可以计算任意函数 $f:\{0,1\}^n \to \{0,1\}^m$，但是为了简化，我们给出输出为单比特的布尔函数 $f:\{0,1\}^n \to \{0,1\}$ 的证明过程，通用性的一般证明可以从布尔函数的特殊情况得到。

定理 3.4　用固定数量的门就可以计算布尔函数 $f:\{0,1\}^n \to \{0,1\}$。

证明（归纳法）　当 $n=1$ 时，恒等函数可由一条单线构成；比特翻转函数可由一个非门构成；把输入比特替换为 0 的函数可由包含一个初态为 0 的工作比特与门构成；把输入替换为 1 的函数可由包含一个初态为 1 的工作比特或门构成。

假设任意 n 比特函数可由一条线路计算，另外 f 是 $n+1$ 比特函数。定义两个 n 比特函数 f_0 和 f_1 分别为 $f_0(x_1,x_2,\cdots,x_n)=f(0,x_1,x_2,\cdots,x_n)$ 和 $f_1(x_1,x_2,\cdots,x_n)=f(1,x_1,x_2,\cdots,x_n)$，由归纳假设知道可以用线路计算。 ∎

由上述定理的证明过程可知，根据第 1 位比特的输入 0 或 1，以及后 n 个比特计算函数 f_0 和 f_1，我们可以设计出计算 f 的线路（见图 3.4）。

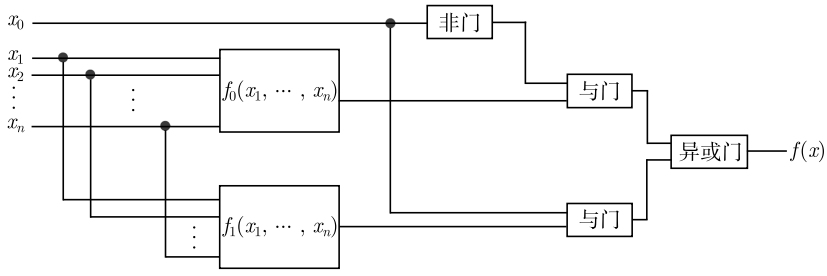

图 3.4　归纳法中计算 $n+1$ 比特函数 f 的线路

通用线路设计中包含以下 5 种元素。

① 保持比特状态不变的连线。

② 用于 $n=1$ 时证明的处于标准状态的辅助比特。

③ 把输入的单比特输出为其两个复制的扇出运算。

④ 交换两个比特值的交叉运算。

⑤ 与、非和异或门。

后续章节将参照经典线路模型定义量子计算的线路模型。然而，将这 5 个元素从经典线路模型推广到量子线路模型存在很多挑战：量子不可克隆原理；扇出运算在量子力学中不能以直接的方式进行；与门和异或门不可逆，因此无法直接用酉算子实现。

为了衡量计算的开销，可以使用不同的参数：线路中门的数量 N 或线路的时间 T。在本书中，我们将主要考虑门的数量 N。我们对消耗 S 感兴趣，其中 S 作为输入大小（即线路中输入线的数量，通常用 n 表示）的函数。为了找到消耗函数 $S(n)$，把函数 f 看作由线路族 $\{C_n\}_{n=1}^{\infty}$ 计算的 $\{f_n\}_{n=1}^{\infty}$ 函数族，其中线路 C_n 作用在 n 个输入位上，$S(n)$ 为线路 C_n 的大小。

这里讨论图灵机模型和线路模型之间的一个重要区别。许多信息可以通过硬件进入线路。如果不指定设计硬件需要多长时间，则这样的线路甚至可以计算不可计算的函数（如停机问题）。线路的这种不合理的计算能力是由于我们没有指定构造线路的硬件。要避免这样的荒谬，把注意力集中在有趣和现实的情况上。因此把要求计算 $\{f_n\}_{n=1}^{\infty}$ 的线路硬件用一台图灵机以多项式消耗进行设计。如果图灵机以整数 n 作为输入，输出线路 C_n 的具体描述，那么该线路族 $\{C_n\}_{n=1}^{\infty}$ 的模型被称作一致线路模型，否则就称为非一致线路模型。一致布尔线路和图灵机的模型是多项式等价的，这意味着给定一个在多项式时间内计算 $f(x)$ 的图灵机，存在一个由多项式图灵机指定的线路族 $\{C_n\}_{n=1}^{\infty}$，使得 C_n 计算 f_n。这种对应关系反过来也成立，也就是说，给定线路族，存在一个图灵机模拟它们。因此，计算的复杂性不取决于所使用的模型（多项式因子除外）。

3.2　计算复杂性类

完成一项计算需要什么样的时间和空间资源？许多情况下，这是我们对一个计算问题能够提出的最重要的问题。计算复杂性研究就是解决计算问题需要的时间和空间资源。计算复杂性的任务是证明解决问题最好算法所需要资源的下界估计，即使算法不是具体已知的。形式化描述计算复杂性的理论的一个困难在于，解决同一个问题的不同计算模型可能需要不同的资源。例如，多带图灵机求解许多问题比单带图灵机快得多。下面介绍一个粗略的解决方法。设一个问题由输入的 n 比特给定，例如我们可能想知道一个特定的 n 比特数是否为素数，各问题计算复杂性的主要差别是以 n 的多项式为界的资源解决，还是需要以比 n 的任意多项式增长更快的资源解决。后一种情况我们常说资源要求问题规模是指数性（exponential）的，这是对指数说法的滥用，因为存在如 $n^{\log n}$ 类的数比任何多项式增长快（因此按这个惯例是指数的），但比任何真正的指数增长都要慢。如果存在一个算法，用多项式资源求解这个问题，则该问题将被视为容易、可解或可行的，而如果已知的最好算法需要指数资源，则称之为难、不可解或不可行的。

多项式与指数的分类相当粗糙。在实践中，一个用 $2^{n/1000}$ 个操作的求解算法可能比一个运行 n^{1000} 个操作的算法更有用。仅对非常大的输入长度 $(n \approx 10^8)$，这个有效的多

项式算法才比非有效的指数算法更可取，但对许多情形来说，选择这里的非有效指数算法更实际。

不过，主要基于多项式与指数分类研究计算复杂性有许多原因。首先，历史上，多项式算法几乎毫无例外地远远快于指数算法。第二个原因也是更根本的，强调多项式与指数分类的原因来自强 Church-Turing 论题（任何计算模型都可用基本操作次数至多为多项式增长的概率型图灵机模拟）。强 Church-Turing 论题意味着如果一个问题在概率型图灵机上没有多项式解法，那么在任何计算设备上都不会有有效解法，这对“多项式可解”与“有效可解”等同起来的看法产生了强烈的推动作用。然而，由于一些问题在经典计算机 (包括概率型图灵机) 上不可解决，而其在量子计算机上可以得到有效解决，这对强 Church-Turing 问题提出了质疑。

最后，对问题的多项式与指数的分类只是弄明白了问题难度的最初步和粗略的步骤，它说明了计算机科学资源问题的很多一般要点，但是它也存在严重的问题：很难证明一个有意义的问题是否存在仅需指数资源的求解方法。

现在，给出问题按照需要资源大小分类的新方式。复杂性类是在给定资源界限下能被计算的所有函数构成的集合，本节将综述各种计算复杂性类。

3.2.1 P 类和 NP 类

许多计算题可以非常清楚地描述为判定问题——答案为是或否的问题。例如，一个给定的数 m 是否为素数？这是素性判定问题，计算复杂性很容易被描述为判定问题形式，这主要出于两个原因：这种理论的形式最简单也最优雅，同时又可以以自然的方式推广到更复杂的情形；历史上的计算复杂性理论基本上是从判定问题的研究而产生的。

字母表 Σ 上的判定问题/语言 L 定义为 Σ 上所有有限字符串集合 Σ^* 的一个子集。例如就素性判断问题而言，就能用二进制字母表 $\Sigma=\{0,1\}$ 编码，语言 $L=\{10,11,101,111,1011,\cdots\}$。

为解决素性判定问题，我们希望一台图灵机从给定的数 n 开始，最终输出与当 n 为素数时为“肯定”、当 n 不为素数时为“否定”相等价的结果。为了使这个想法精确化，稍微修改一下我们之前的图灵机模型，把 q_{halt} 换成分别表示“肯定”和“否定”的答案的两个状态 q_{yes} 和 q_{no}，其他方面机器的行为完全不变，并且在进入 q_{yes} 或 q_{no} 时停止，更一般地，一个语言 L 由图灵机判定，假设机器可以判断带子上的输入 x 是否为 L 的一个元素，如果 $x\in L$，最终会停在 q_{yes} 状态，如果 $x\notin L$，最终会停在 q_{no} 状态，根据出现的两种情况，我们说机器接受或拒绝。

可以多快地判断一个数是否为素数呢？也就是说，判定表示素性判定问题的语言的最快的图灵机是怎样的？

定义 3.3（DTIME 类） 设 $T\colon N\to N$ 是一个函数，称语言 $L\in\mathbf{DTIME}(T(n))$ 当且仅当存在运行时间为 $k\cdot T(n)$ 的图灵机判定语言 L，其中 $k>0$ 为常数。

现在，将“高效计算”的概念精确化。将高效计算等同于多项式运行时间，即对某个常数 $c>0$ 而言，至多为 n^c 的运行时间。这一概念由下面的复杂性类刻画，其中 **P** 表示

“多项式的”。

定义 3.4（**P** 类）　$\mathbf{P}=\bigcup_{c\geqslant 1}\mathbf{DTIME}(n^c)$。

P 是复杂性类的第一个例子，现在给出属于 **P** 的一个问题——图的连通性问题。在图的连通性问题中，给定图 G 以及它的两个顶点 s 和 t，要求判定 s 在 G 上是否可以连通到 t。证明该问题属于 **P** 可以采用深度优先搜索算法，算法从 s 出发，逐一检查 G 的边，并对所有访问过的边进行标记，后续步骤需要访问相邻接的边，最多经过 $\binom{n}{2}$ 步后，所有的边要么全部访问过，要么永远无法访问。

更一般地，一个复杂性类定义为一个由语言构成的集合。现在，定义复杂性类 NP 类，由此将“可被高效验证的解”这一直觉概念形式化。我们曾说一个问题是“可被高效求解的”，如果它可以被图灵机在多项式时间内求解，则问题的解是“可被高效验证的”，即该问题的解可以在多项式时间内得到验证。由于图灵机每步只能读一个位，因此这意味着给定的解不能太长——至多是输入长度的多项式。

定义 3.5（NP 类）　语言 $L\subseteq\{0,1\}^*$ 属于 NP，如果存在多项式函数 $p\colon N\to N$ 和一个多项式时间图灵机 M（称为 L 的验证器），使得对任意 $x\in\{0,1\}^*$ 有

$$x\in L\Leftrightarrow\exists u\in\{0,1\}^{p(|x|)}\text{ 满足 }M(x,u)=1$$

如果 $x\in L$ 和 $u\in\{0,1\}^{p(|x|)}$ 满足 $M(x,u)=1$，则称 u 是 x（关于语言 L 和图灵机 M）的见证。

下面给出属于 NP 类的几个判定问题。

（1）旅行商问题：给定 n 个结点的集合，$\binom{n}{2}$ 个任意结点对之间的距离 $d_{i,j}$ 以及一个数 k，判定是否存在一个封闭回路，使得访问每个结点恰好一次且代价不超过 k。见证是该回路中所有顶点的序列。

（2）子集和问题：给定数集 $\{A_1,A_2,\cdots,A_n\}$ 和数 T，判定是否存在一个和等于 T 的子集。见证是这种子集中所有数的列举。

（3）线性规划问题：给定 n 个变量 $u_1,u_2,\cdots u_n$ 上的 m 个有理系数线性不等式（每个线性不等式形如 $a_1u_1+a_2u_2+\cdots+a_nu_n\leqslant b$，其中 $a_1,a_2,\cdots a_n$，b 为有理数），判定是否存在变量 $u_1,u_2,\cdots,u_n$ 上的 m 个有理数赋值满足所有不等式。见证是这样的一个赋值。

（4）0/1 整数规划问题：给定 n 个变量 $u_1,u_2,\cdots u_n$ 上 m 个有理系数线性不等式，能否将变量 $u_1,u_2,\cdots u_n$ 赋值为 0 或 1，使得所有不等式成立。见证是这样的一个赋值。

（5）图的同构问题：给定两个 $n\times n$ 的邻接矩阵 $\boldsymbol{M}_1$ 和 $\boldsymbol{M}_2$，判定在顶点重新命名的意义下 $\boldsymbol{M}_1$ 和 $\boldsymbol{M}_2$ 是否定义了同一个图。见证是如下的一个置换 $\pi:[n]\to[n]$，它将 $\boldsymbol{M}_1$ 的行和列按 π 进行调整后使得 $\boldsymbol{M}_1$ 等于 $\boldsymbol{M}_2$。

（6）合数问题：给定一个整数 N，判定 N 是否是一个合数。见证是 N 的一个因数分解。

（7）因数问题：给定 3 个整数 N、L、U，判定 N 是否有一个素因数 p 属于区间

$[L,U]$。见证是这样的素因数 p。

（8）图连通性问题：给定图 G 以及它的两个顶点 s 和 t，要求判定 s 和 t 在 G 上是否连通。见证是 G 中从 s 到 t 的一条路径。

上述列举的问题中，图的连通性问题、合数问题和线性规划问题已被证明属于 P。目前还无法判断列举的其他问题是否属于 P，尽管也无法证明它们不属于 P。

现在我们感兴趣的是 P 类和 NP 类是什么关系。

定理 3.5 $\mathrm{P}\subseteq\mathrm{NP}$。

证明

设语言 $L\in\mathrm{P}$ 被多项式时间图灵机 M 判定，则将类 NP 定义中的 $p(x)$ 视为零多项式（即 u 为空串），于是 $L\in\mathrm{NP}$。∎

定理 3.5 表明 P 是 NP 的子集。然而计算机科学中最著名的未解决问题就是是否存在不在 P 中的 NP 问题，常简记为 $\mathrm{P}\neq\mathrm{NP}$ 问题。大多数计算机科学家相信 $\mathrm{P}\neq\mathrm{NP}$，但几十年过去了，仍没有人能证明它，$\mathrm{P}\neq\mathrm{NP}$ 的可能性仍然存在。

现在我们思考在什么情况下 P=NP，这需要引入归约、NP-难和 NP-完全性的概念。

定义 3.6（归约，NP-难和 NP-完全性） 称语言 $L\subseteq\{0,1\}^*$ 多项式时间卡普归约到语言 $L'\subseteq\{0,1\}^*$，记为 $L\leqslant_p L'$，如果存在多项式时间可计算的函数 $f:\{0,1\}^*\to\{0,1\}^*$ 使得对于任意的 $x\in\{0,1\}^*$ 都有 $x\in L$ 当且仅当 $f(x)\in L'$。多项式时间卡普归约简称为多项式时间归约。如果 $L\leqslant_p L'$ 对任意 $L\in\mathrm{NP}$ 成立，则称 L' 是 NP-难的。如果 L' 是 NP-难的且 $L'\in\mathrm{NP}$，则称 L' 是 NP-完全的。

根据多项式归约的定义可以得到以下结论。

定理 3.6

（1）如果 $L\leqslant_p L'$ 且 $L'\in\mathrm{P}$，则 $L\in\mathrm{P}$。

（2）如果 $L\leqslant_p L'$ 且 $L'\leqslant_p L''$，则 $L\leqslant_p L''$。

证明

（1）可以结合图 3.5 简单地验证。

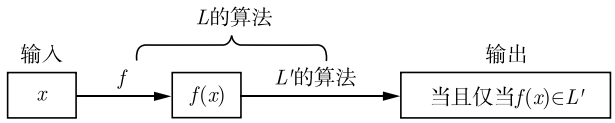

图 3.5 定理 3.6 中（1）的证明过程示意

（2）假设 f_1 是从 L 到 L' 的归约且 f_2 是从 L' 到 L'' 的归约，由于在给定的 x 上 $f_2(f_1(x))$ 在多项式时间内完成计算，同时 $f_2(f_1(x))\in L''\Leftrightarrow f_1(x)\in L'\Leftrightarrow x\in L$，那么映射 $x\mapsto f_2(f_1(x))$ 是从 L 到 L'' 的归约。∎

下面我们不加证明地给出 P=NP 的一些充分条件。

定理 3.7

（1）如果语言 L 是 NP- 难的且 $L\in\mathrm{P}$，则 P=NP。

（2）如果语言 L 是 NP-完全的，则 $L\in \mathrm{P}$ 当且仅当 P=NP。

NP-完全语言真的存在吗？换句话说，NP类中真的有一个语言与该类中的其他任意语言一样难吗？下面给出了这种语言的一个简单例子。

定理 3.8　设 TMSAT=$\{\langle \alpha, x, 1^n, 1^t\rangle : \exists u\in\{0,1\}^n$ 使得 M_α 以 $\langle x,u\rangle$ 为输入时将在 t 步内输出 1$\}$，其中 M_α 是位串 α 表示的图灵机，则 TMSAT 是 NP-完全的。

证明　设 L 是一个 NP 语言，根据 NP 类的定义知道，存在多项式 p 和一个验证器图灵机 M 使得 $x\in L$ 当且仅当存在位串 $u\in\{0,1\}^{q(x)}$ 且满足 $M(x,u)=1$ 且 M 的运行时间是多项式时间 $q(n)$。为了将 x 归约到 TMSAT，需要将任意位串 $x\in\{0,1\}^*$ 映射到元组 $\langle \underline{M}, x, 1^{p|x|}, 1^{q|m|}\rangle$，其中 $m=|x|+p(|x|)$，$\underline{M}$ 是图灵机 M 的位串表示。根据 TMSAT 的定义和 M 的选择，该映射显然可以在多项式时间内完成，且 $\langle \underline{M}, x, 1^{p|x|}, 1^{q|m|}\rangle \in$ TMSAT $\Leftrightarrow \exists u\in\{0,1\}^{q(x)}$ 使得 $M(x,u)$ 在 $q(m)$ 步内输出 $1\Leftrightarrow x\in L$。 ∎

TMSAT 不是一个有用的 NP 完全问题，因为该语言的定义与图灵机概念的联系过于密切。下面将给出更“自然”的一些 NP-完全问题的例子。首先研究来自命题逻辑领域的 NP-完全问题。

变量 $u_1,u_2,\cdots,u_n$ 上的一个布尔公式由变量和逻辑运算符与（AND）($\wedge$)、或（OR）($\vee$)、非（NOT）($\neg$) 构成。例如，$(u_1\wedge u_2)\vee(u_2\wedge u_3)\vee(u_3\wedge u_1)$ 是一个布尔公式。如果 φ 是变量 $u_1,u_2,\cdots u_n$ 上的一个布尔公式且 $z\in\{0,1\}^n$，则 $\varphi(z)$ 表示依次赋予 z 中的各个值后 φ 的取值，z 中的 1 表示真而 0 表示假。如果存在赋值 z 使得 $\varphi(z)$ 为真，则称 φ 是可满足的，否则称 φ 是不可满足的。之前给出的公式 $(u_1\wedge u_2)\vee(u_2\wedge u_3)\vee(u_3\wedge u_1)$ 是可满足的，因为赋值 $u_1=z_1,u_2=z_2,u_3=z_3$ 满足该公式当且仅当至少有两个 z_i 的值是 1。

如果 $u_1,u_2,\cdots,u_n$ 上的布尔公式是在变量（或变量的非）上 OR 操作所得的若干公式上的 AND 操作，则称该公式是合取范式（Conjunctive Normal Form，CNF）。例如下面的公式就是一个 3CNF（这里的 $\bar{u}_i$ 表示 $\neg u_i$）。

$$(\bar{u}_1\vee u_2\vee u_3)\wedge(u_1\vee\bar{u}_2\vee u_3)\wedge(u_1\vee u_2\vee\bar{u}_3) \tag{3.2.1}$$

更一般地，合取范式形如

$$\wedge_i\left(\vee_j v_{i_j}\right) \tag{3.2.2}$$

其中每个 v_{i_j} 要么是一个变量 u_k，要么是变量的非 $\bar{u}_k$，项 v_{i_j} 称为公式的文字，而项 $\vee_j v_{i_j}$ 称为公式的子句。如果一个公式中的每个子句至多包括 k 个文字，则称该公式为一个 k 合取范式或 kCNF。所有可满足的 CNF 公式构成的语言记为 SAT，所有可满足的 3CNF 公式构成的语言记为 3SAT。

现在给出一个自然的 NP-完全问题。

定理 3.9　（Cook-Levin 定理）

（1）SAT 问题是 NP-完全问题。

（2）3SAT 问题是 NP-完全问题。

定理 3.9 是于 20 世纪 70 年代由 Cook 和 Levin 证明的，由于证明过程烦琐，这里仅给出证明思路而不予证明，感兴趣的读者可以阅读相关参考文献。证明思路为先证明

SAT 问题是 NP- 难的，再证明 SAT 多项式时间归约到 3SAT。

读者可能会问，为什么 3SAT 问题的 NP-完全性比 TMSAT 的 NP-完全性更自然、更有意义呢？第一个原因是 3SAT 问题在证明其他问题的 NP-完全性时非常有用：3SAT 问题具有极其简单的组合结构，便于归约过程采用。第二个原因是命题逻辑在数理逻辑中具有中心地位，这正是 Cook 和 Levin 首先研究 3SAT 问题的原因所在。第三个原因是 3SAT 具有重要的实践意义：实际上，3SAT 问题是约束满足问题的一个简单特例，而约束满足问题广泛存在于包含人工智能在内的许多领域中。

NP 类的重要性部分来自无数计算问题已知为 NP-完全的事实。下面是一些不同数学领域中 NP-完全问题的例子。

（1）团问题：一个团是一个无向图 G 的顶点子集，该集中每对顶点均有边连接。团的大小就是它包含顶点的个数。给定一个整数 m 和一个图 G，G 是否包含一个大小为 m 的团？

（2）顶点覆盖问题：一个无向图 G 的一个顶点覆盖是顶点的一个子集 V'，使得图中每条边至少有一个顶点属于 V'。给定一个整数 m 和一个图 G，图 G 是否有一个顶点覆盖包含 m 个顶点？

（3）子集和问题。

（4）0/1 整数规划问题。

设 $\mathrm{P} \neq \mathrm{NP}$，则可证明存在一个既不是多项式资源可解，又不是 NP-完全的非空问题类 NPI（NP 中间问题）。很显然，没有已知的NPI问题（否则我们就知道 $\mathrm{P} \neq \mathrm{NP}$ 了），但有些问题被认为是可能的候选问题，因子问题和图同构问题是候选问题中最有可能的两个。

量子计算研究者对 NPI 类中的问题感兴趣有两个原因。首先，希望找到用以求解不在 P 类中的问题的快速量子算法；其次，很多人怀疑量子计算机不能有效求解 NP 中的全部问题，从而排除了 NP-完全问题，因此很自然地把注意力集中到 NPI 类上。实际上已经发现了因子问题的快速量子算法，并且已经激发起对其他可能是 NPI 问题的快速量子算法的寻找。

3.2.2 其他复杂性类

我们已经考察了某些重要的复杂性类（如 P、NP 和 NP 完全类）的基本性质。复杂性类的类型非常丰富，在这些复杂性类之间已知或可能存在许多不平凡的关系。对学习量子计算而言，不需要明白所有这些不同的复杂性类，不过，对重要的复杂性类稍做了解还是有用的，因为许多复杂性类在量子计算研究中有自然的对应物，而且如果要了解量子计算机的能力如何，也应该明白量子计算机所能解的问题类，以及在经典计算机所定义复杂性类的家族中处于什么位置。

现在先介绍其他复杂性类，包括 EXP 类、PSPACE 类以及 L 类，然后给出所有介绍过的复杂类之间的包含关系。

定义 3.7 （EXP类）$\mathrm{EXP} = \bigcup_{c>1} \mathrm{DTIME}\left(2^{n^c}\right)$，也就是说，复杂性类EXP表示包含

所有由图灵机在指数时间内可以判定的问题。

前面所介绍的复杂类都是依据资源在时间上的定义，接下来介绍资源在空间上所定义的复杂性类。

定义 3.8（空间受限计算）　设 $S\colon N \to N$ 且 $L \subseteq \{0,1\}^*$。如果存在常数 c 和判定 L 的图灵机 M，使得对任意长度为 n 的输入，M 在完成计算过程中带头至多只访问除输入带之外各条工作带上的 $c \cdot S(n)$ 个存储单元，则称 $L \in \mathbf{SPACE}(S(n))$。

定义 3.9（**PSPACE** 类）　$\mathrm{PSPACE} = \bigcup_{c>0} \mathrm{SPACE}(n^c)$，即复杂性类PSPACE是在图灵机上用多项式数量的工作比特，在不限制时间的情况下可以求解的问题类。

定义 3.10（L类）　$\mathrm{L} = \mathrm{SPACE}(\log n)$，也就是说，复杂性类L包含所有由图灵机在对数空间内可以判定的判定问题。

我们不加证明地给出目前已知的各类空间受限复杂性类和时间受限复杂性类之间的关系：

定理 3.10　$\mathrm{L} \subseteq \mathrm{P} \subseteq \mathrm{NP} \subseteq \mathrm{PSPACE} \subseteq \mathrm{EXP}$。

最后介绍另一类我们感兴趣的复杂性类BPP，为此需要引入概率型图灵机的概念。概率型图灵机是有两个转移函数 δ_0、δ_1 的图灵机。当概率型图灵机 M 在输入 x 上运行时，每个步骤以 1/2 的概率选用转移函数 δ_0，以 1/2 的概率选用转移函数 δ_1。每一步的选择均独立于之前所做的所有选择。概率型图灵机只能输出 1（接受）或 0（拒绝），输出结果是一个随机变量。

定义 3.11（BPP类）　复杂性类BPP是包含所有具有如下性质的语言 L，即存在一个概率型图灵机 M 对任何输入均在多项式时间后停止，而且使得：若 $x \in L$，则 M 以至少 2/3 的概率接受 x；若 $x \notin L$，则 M 以至少 2/3 的概率拒绝 x。

在这里，定义里面的常数 2/3 可以是任意给定的，它可以是在 1/2 与 1 之间的任意常数，而BPP集合维持不变。原因在于，虽然该算法有错误的机率，但是只要我们多运行几次算法，那么多数的答案都是错误的机率会呈指数级衰减，这个结果可由 Chernoff 界导出。

定理 3.11（Chernoff 界）　设 $X_1, X_2, \cdots, X_n$ 是独立同分布随机变量，每个随机变量取 1 的概率为 $1/2+\varepsilon$，取 0 的概率为 $1/2-\varepsilon$，则

$$p\left(\sum_{i=1}^{n} X_i \leqslant \frac{n}{2}\right) \leqslant \mathrm{e}^{-2\varepsilon^2 n} \tag{3.2.3}$$

证明　考虑任意包含最多 $n/2$ 个 1 的序列 $x_1, x_2, \cdots, x_n$，当含有 $\lfloor n/2 \rfloor$ 个 1 时，这种序列出现的概率达到最大，因此

$$p(X_1 = x_1, x_2, \cdots, x_n = x_n) \leqslant \left(\frac{1}{2}-\varepsilon\right)^{\frac{n}{2}} \left(\frac{1}{2}+\varepsilon\right)^{\frac{n}{2}} = \frac{\left(1-4\varepsilon^2\right)^{\frac{n}{2}}}{2^n} \tag{3.2.4}$$

总共最多有 2^n 个这样的序列，于是

$$p\left(\sum_{i=1}^{n} X_i \leqslant \frac{n}{2}\right) \leqslant 2^n \times \frac{\left(1-4\varepsilon^2\right)^{\frac{n}{2}}}{2^n} = \left(1-4\varepsilon^2\right)^{\frac{n}{2}} \tag{3.2.5}$$

最后由不等式 $1-x \leqslant \exp(-x)$，所以

$$p\left(\sum_{i=1}^{n} X_i \leqslant \frac{n}{2}\right) \leqslant \mathrm{e}^{-4\varepsilon^2 n/2} = e^{-2\varepsilon^2 n} \tag{3.2.6}$$

∎

BPP 类甚至比 P 类更适合作为在经典计算机上可以有效解决的判定问题类。BPP 类的量子对应称为 BQP 类，它是人们在研究量子算法时最感兴趣的类。

定义 3.12（BQP类） 复杂性类 BQP 是包含所有具有如下性质的语言 $L \subseteq \{0,1\}^*$，即存在一个多项式函数 p 以及一个一致量子线路族 $\{C_n\}_n$，使得对任意 $x \in \{0,1\}^n$ 满足：若 $x \in L$，则 $C_n(x)$ 以至少 a 的概率接受；若 $x \notin L$，则 $C_n(x)$ 以至多 b 的概率接受。其中 $a-b \geqslant 1/p(n)$ 且 $|C_n| \leqslant p(n)$。

人们通常把能解决 BPP（BQP）问题的机器称为 BPP 机器（BQP 机器）。

人们将数学结论的正确性证明用一系列符号写在纸上，验证者可以查验证明者写出的证明是否有效。当然，只有正确的结论才存在有效证明。但是，更一般地，人们还通常通过交流让大家相信结论的正确性。也就是说，查验证明的人（称为验证者）要求提供证明的人（称为证明者）先给出一系列解释，然后才相信结论是正确的。这个过程也称为交互证明过程。下面介绍交互式证明中的复杂性类。

首先我们感兴趣的一个类是 MA，它由一些判定问题组成：当给定一个多项式大小的比特串（即见证）时，BPP 机器能够检查这些决策问题的结果为“是”的情况，具体定义如下。

定义 3.13（MA 类） 复杂性类 MA 是包含所有具有如下性质的语言 $L \subseteq \{0,1\}^*$，即存在一个多项式函数 p 以及一个 BPP 机器 V，使得对于任意 $x \in \{0,1\}^n$ 满足：若 $x \in L$，则存在一个串（见证）$w \in \{0,1\}^{\leqslant p(n)}$，使得 $V(x,w)$ 以至少 a 的概率接受；若 $x \notin L$，则对任意串 $w \in \{0,1\}^{\leqslant p(n)}$，使得 $V(x,w)$ 以至多 b 的概率接受。其中，$a-b \geqslant 1/p(n)$。

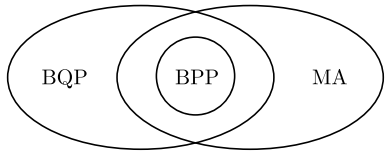

图 3.6 BPP、BQP 和 MA 的包含关系

我们有 BPP $\subseteq$ MA，因为任何 BPP 问题相当于见证为空的 MA 问题。MA 的量子对应叫作 QMA，它的证明串是一种量子状态，如图 3.6 所示。

定义 3.14（QMA 类） 复杂性类 QMA 是包含所有具有如下性质的语言 $L \subseteq \{0,1\}^*$，即存在一个多项式函数 p 以及一个 BQP 机器 V，使得对于任意 $x \in \{0,1\}^n$ 满足：若

$x \in L$，则存在一个最多包含 $p(n)$ 量子比特的量子态 $|\psi\rangle$，使得 $V(x, |\psi\rangle)$ 以至少 a 的概率接受；若 $x \notin L$，则对任意最多包含 $p(n)$ 量子比特的量子态 $|\psi\rangle$，使得 $V(x, |\psi\rangle)$ 以至多 b 的概率接受。其中，$a - b \geqslant 1/p(n)$。

显然 BQP $\subseteq$ QMA，因为 BQP 机器可以简单地忽略证明态 $|\psi\rangle$。接下来，给出 IP 类的定义。

定义 3.15（IP 类）　复杂性类 IP 是包含所有具有如下性质的语言 $L \subseteq \{0,1\}^*$，即存在一个多项式函数 p 以及一个概率型多项式时间图灵机 V，使得对于任意 $x \in \{0,1\}^n$ 满足：若 $x \in L$，则存在一个证明者 P 与验证者 V 交互最多包含长度 $p(n)$ 的信息，使得 V 以至少 a 的概率接受；若 $x \notin L$，则对任意证明者 P，P 与验证者 V 交互最多包含长度 $p(n)$ 的信息，使得 V 以至多 b 的概率接受。其中，$a - b \geqslant 1/p(n)$。

前面的类是相当经典的复杂度类，现在描述一个更加非经典类的定义。

定义 3.16（QPIP 类）　复杂性类 QPIP 是包含所有具有如下性质的语言 $L \subseteq \{0,1\}^*$，即存在一个多项式函数 p 以及一个概率型多项式时间图灵机 V（增强它准备和测量一组常数量子比特的能力），使得对于任意 $x \in \{0,1\}^n$ 满足：若 $x \in L$，则存在一个 BQP 证明者 P 与验证者 V 交互最多包含长度 $p(n)$ 的经典或量子信息，使得 V 以至少 a 的概率接受；若 $x \notin L$，对任意 BQP 证明者 P，P 与验证者 V 交互最多包含长度 $p(n)$ 的经典或量子信息，使得 V 以至多 b 的概率接受。其中，$a - b \geqslant 1/p(n)$。

3.3　计算科学的发展与展望

计算科学是一个快速增长的多学科领域，使用先进的计算能力理解和解决复杂的问题。计算作为数学的研究对象已有几千年了。计算不等于数学，但数学确实起源于对计算的研究。计算的渊源可以深入扩展到数学和工程。数学为计算提供了理论、方法和技术，而工程为实际计算和应用提供了可以自动计算的设备，并为更有效地完成计算和应用任务提供了工程方法和技术。从类型上讲，计算主要有两大类：数值计算和符号推导。数值计算包括实数和函数的加减乘除、幂运算、开方运算、方程的求解等。符号推导包括代数与各种函数的恒等式、不等式的证明及几何命题的证明等。但无论是数值计算还是符号推导，它们在本质上是等价的、一致的，即二者是密切关联的，可以相互转化，具有共同的计算本质。随着数学的不断发展，还可能出现新的计算类型。

电子计算机在其发展过程中惊人地遵循摩尔定律，为人类文明社会的发展做出了巨大贡献。但是，半个世纪以来，科学家却一直在考虑新型计算机模型的研制。究其原因，主要有两点：第一，电子计算机的工艺制造技术即将达到极限，如著名的理论物理学家 Kaku 在 2012 年预言，10 年内电子计算机的工艺制造技术将达到极限；第二，由于图灵机模型所致，电子计算机一直不能处理规模较大的 NP-完全问题。在探索非传统的新型计算机模型研究中，相继提出了仿生计算（人工神经网络、进化计算、PSO 计算等）、光计算、量子计算及生物计算等。

我们之前的讨论都是基于图灵机模型的经典计算模型，下面介绍一些非经典计算模

型，如并行计算和分布式计算，对比它们和图灵机模型的计算能力，最后延伸到量子计算。首先介绍并行计算的概念。并行计算（Parallel Computing）是指在并行机上将一个应用分解成多个子任务，分配给不同的处理器，各个处理器之间相互协同，并行地执行子任务，从而达到加速求解速度或者提高求解应用问题规模的目的。为了实现并行计算，必须具备 3 个条件。

（1）并行机。并行机至少包含两台或两台以上的处理机，这些处理机通过互连网络相互连接，相互通信。

（2）应用问题必须具有并行度。也就是说，应用可以分解为多个子任务，这些子任务可以并行地执行。将一个应用分解为多个子任务的过程称为并行算法的设计。

（3）并行编程。在并行机提供的并行编程环境上，具体实现并行算法，编制并行程序并运行该程序，从而达到并行求解应用问题的目的。对于具体的应用问题，采用并行计算技术的主要目的在于两个方面：① 加速求解问题的速度；② 提高求解问题的规模。

并行计算机的优势在于时间资源上的节省，但是在空间资源上却有损失。图灵机可以用与多项式等价的物理资源（时间和空间资源总和）实现并行计算机的功能。

并行计算机的一个特例就是生物计算机。生物计算是指以生物大分子作为“数据”的计算模型。首先，生物计算机在存储方面与传统电子计算机相比具有巨大优势，1 克 DNA 的存储信息量可与 1 万亿张 CD 相当，存储密度是通常使用的磁盘存储器的 1000 亿到 10 000 亿倍。其次，生物计算机还具有超强的并行处理能力，通过一个狭小区域的生物化学反应即可实现逻辑运算，数百亿个 DNA 分子构成大批 DNA 计算机并行操作，生物计算机的传输数据与通信过程简单，其并行处理能力可与超级电子计算机媲美，通过 DNA 分子碱基不同的排列次序作为计算机的原始数据，对应的酶通过生物化学变化对 DNA 碱基进行基本操作，能够实现电子计算机的各种功能。然后，蛋白质分子可以自我组合，能够新生出微型电路，具有活性，因此生物计算机拥有生物特性，生物计算机不再像电子计算机那样在芯片损坏后无法自动修复，生物计算机能够发挥生物调节机能，自动修复受损芯片。最后，由于 DNA 链的另一个重要性质是双螺旋结构，A 碱基与 T 碱基、C 碱基与 G 碱基形成碱基对，因此每个 DNA 序列有一个互补序列，这种互补性是生物计算机具备的独特优势。如果错误发生在 DNA 的某一对螺旋序列中，则修改醇能够参考互补序列对错误进行修复。生物计算机自身具备修改错误的特性，因此生物计算机的数据错误率较低。

下面介绍分布式计算（Distributed Computing）的概念。分布式计算通过网络相互连接的两个以上的处理机相互协调，各自执行相互依赖的不同应用，从而达到协调资源访问，提高资源使用效率的目的。但是，分布式计算无法达到并行计算所倡导的提高求解同一个应用的速度或者问题规模的目的。分布式计算的优点是超大规模、虚拟化、高可靠性、通用性、高可伸缩性、按需服务、极其廉价和容错性，缺点则是多点故障（一台或多台计算机的故障，或者一条或多条网络线路的故障，都会导致分布式系统出现问题）；安全性（分布式系统为非授权用户的攻击提供了更多机会）。分布式计算的一个重要特例就是区块链——一种分布式数据存储、点对点传输、共识机制、加密算法等计算机技术的新

型应用模式。

由于分布式计算的原理是将一个大型计算任务拆分成许多部分，并分别交给其他计算机处理，同时将所有的计算结果合并为原问题的解决方案，所以它也可以用图灵机有效模拟。

最后我们延伸至量子计算。量子计算是一种遵循量子力学规律，通过调控量子信息单元进行计算的新型计算模式，它的计算速度指数倍地快于经典的确定型图灵机和概率型图灵机。研究量子计算机的一个重要原因是它构成了对强 Church-Turing 论题的严重挑战：该猜想断言，任意可物理实现的计算装置都可以被图灵机模拟，而计算速度最多下降一个多项式因子。在量子计算机上存在整数分解问题的多项式时间算法；然而，在概率型图灵机和确定型图灵机上，人们经过长期努力仍不能为整数分解问题找到多项式时间算法。如果整数分解问题根本不存在高效的经典算法（目前，现实社会严重依赖于这一猜想，因为诸如 RSA 等密码方案的安全性全部建立在该猜想的基础之上），或者量子计算机是可以物理实现的，则强 Church-Turing 论题就是错误的。此外，物理学家也对量子计算机感兴趣，因为这有助于他们理解量子力学理论。

习题

1. 给出一个计算布尔函数 PAL 的图灵机 M，其中布尔函数 PAL 的定义如下：对于任意 $x \in \{0,1\}^*$，如果 x 是回文，则 $\mathrm{PAL}(x)$ 等于 1，否则等于 0。也就是说，$\mathrm{PAL}(x)=1$ 当且仅当 x 从右到左读是一样的（即 $x_1x_2\cdots x_n = x_nx_{n-1}\cdots x_1$）。

2. 给出一个将图灵机表示成二进制的完整方案。也就是说，给出一个能将任意图灵机 M 转换为二进制串 $\underline{M}$ 的过程，它要保证能够从 $\underline{M}$ 还原为图灵机 M，或者说至少能够还原出一个功能等价的图灵机 $\widetilde{M}$（即 $\widetilde{M}$ 能够在相同运行时间内计算 M 能计算的函数）。

3. 证明与非门的通用性，即如果可以使用连线、辅助比特和扇出运算，则与非门可以模拟非门、与门和异或门。

4. 证明图论中的下列语言/判定问题属于 P 类（可以使用邻接矩阵表示图，也可以使用邻接表表示图，表示形式不影响结论）。

（1）所有连通图的集合（如果图 G 的每对顶点均通过一条路径连通，则 G 称为连通图）。

（2）所有不含三角形的图的集合（三角形是指相互连通的 3 个不同顶点）。

（3）所有二分图的集合。如果图 G 的顶点可以划分成两个集合 A 和 B，使得 G 的所有边均是从 A 中的一个顶点连接到 B 中的一个顶点（A 中的任意两个顶点之间或 B 中的任意两个顶点之间不存在边），那么称图 G 为二分图。

（4）所有树的集合。如果一个图是连通的且没有环，则称该图为一棵树。

5. 证明下列语言属于 NP 类。

（1）2 着色：$\mathrm{2COL} = \{G : \text{图}G\text{可以用 2 种颜色着色}\}$。

（2）3 着色：3COL = $\{G$: 图G可以用 3 种颜色着色$\}$。

（3）所有连通图构成的集合。

其中，用 k 种颜色对图 G 着色指的是给图 G 的每个顶点赋予 $[k]$ 中的一个数，使得图 G 上相邻顶点上的数字均不相等。

6. 我们已经在所有语言中定义了关系 $\leqslant_p$，注意到它是自反的（$L\leqslant_p L$ 对任意语言 L 成立）和传递的（如果 $L\leqslant_p L'$ 且 $L'\leqslant_p L''$，则有 $L\leqslant_p L''$）。证明这个关系不是对称的，即 $L\leqslant_p L'$ 并不蕴含 $L'\leqslant_p L$。

7. 证明

（1）如果语言 L 是 NP- 难的且 $L\in$ P，则 P=NP。

（2）如果语言 L 是 NP-完全的，则 $L\in$ P 当且仅当 P=NP。

8. 证明旅行商问题是 NP-完全的。

9. 证明 3 着色是 NP-完全的。

10. 证明复杂性类的关系 L $\subseteq$ P $\subseteq$ NP $\subseteq$ PSPACE $\subseteq$ EXP。

参考文献

[1] Hennie F C, Stearns R E. Two-tape simulation of multitape Turing machines[J]. Journal of the ACM, 1966,13(4):533-546.

[2] Bernstein E, Vazirani U. Quantum Complexity Theory[J]. SIAM Journal on Computing, 1997, 26(5):1411-1473.

[3] 张焕国, 毛少武, 吴万青, 等. 量子计算复杂性理论综述 [J]. 计算机学报, 2016, 39(12):2403-2428.

[4] Church A. An unsolvable problem of elementary number theory[J]. American Journal of Mathematics, 1936, 58(2): 345-363.

[5] Gheorghiu A, Kapourniotis T, Kashefi E. Verification of Quantum Computation: An Overview of Existing Approaches[J]. Theory of Computing Systems, 2019, 63:715-808.

[6] Aharonov D, Benor M, Eban E. Interactive Proofs for Quantum Computations[OL]. 2008, https://arxiv.org/abs/0810.5375.

[7] 张林波, 迟学斌, 莫则尧, 等. 并行计算导论 [M]. 北京：清华大学出版社, 2006.