计算机系统的局限性结课总结

张梓卫

（10235101526 软件工程学院 软件工程专业）

摘要： 本文通过多方面探讨计算机系统在理论和实际应用中的局限性及挑战，强调基础理论研究的重要性和长期价值，讨论计算机广泛应用背后的科学研究和探索精神。本文通过对计算机系统局限性的深入探讨，揭示了在面对复杂问题时的种种挑战，并强调了继续探索新计算范式和优化算法的重要性。 接着，介绍了新发展范式，包括量子计算和生物计算，展示了这些新计算模型在提升计算能力和解决复杂问题上的潜力。本文后续还探讨了软件设计的困难和计算机系统在处理复杂问题时的局限性，提出了算法复杂性和不可计算性的问题，并通过课程讲解中的实例说明了这些问题在实际应用中的影响。根据图灵机及其在计算理论中的重要性，讨论了不可计算问题及其数学证明。在探讨新计算范式时，文章介绍了并行计算和随机算法的基本原理及其在解决计算难题中的应用。最后，量子计算作为一种新型计算模式，展示了其在信息处理和密码学中的巨大潜力和挑战。通过零知识证明的讨论，展示了计算机科学在保护隐私和确保信息安全方面的重要进展。

关键词： 计算机局限性, 算法复杂度, 零知识证明, 新计算范式

1 计算机系统的局限性学习的目的

课程第一章指出，计算机在各个领域逐渐得到了更多应用，例如超级计算机、象棋比赛、游戏等。而牛顿的三大运动定律、波尔代数、彭佳莱猜想等的科学定律的重要性，让人们试图使用计算机对其验证。

而中国在基础理论研究上的相对不足，历史上有许多技术发明，但缺乏理论创新，科学研究不仅有经济价值，更重要的是为民族留下宝贵的精神财富。许多科学研究最初似乎无用，但例如伯约伦课程最初的两轮游戏在股票市场中的应用，都为人类留下了深刻的影响。

1.1 新发展范式

量子计算：量子计算的基本原理及其在处理信息方面的优势，例如并行处理能力和解决复杂问题的潜力。

生物计算：利用生物分子的化学反应来处理信息，生物芯片的应用。

计算能力的极限：强调探索新计算模型不仅是为了提升计算能力，也是为了理解计算的极限，并推动技术的进一步发展。

1.2 未知问题探索

计算机难题的应用：提到计算机难以解决的问题可以转化为优势，例如在密码学中的应用，把大数分解为质数的问题用于保密协议。

数字签名：数字签名技术及其在电子政务中的应用，解决了电子签名的真实性和不可抵赖性问题。
零知识证明：即无需透露具体内容即可证明拥有某信息的能力。

2 计算机系统局限性

2.1 计算机应用的广泛性

计算机不仅用于计算，还用于各种日常生活中的应用，如排课、交通控制等。虽然计算机看起来无所不能，但实际上有许多局限性。

2.1.1 计算机的局限性

计算机并不是万能的，有些问题它无法解决。这些问题的无法解决并不是因为计算机的硬件不够先进或程序员的技能不足，而是因为问题本身的性质使得计算机无法用算法解决。例如，某些物理系统的模拟问题，这些问题无法通过计算机在有限时间内解决。

2.1.2 有限问题与无限问题

对于有限集合的问题，计算机总是可以通过查表等简单方法解决。对于无限集合的问题，计算机则可能无法解决。

2.2 算法局限性

时间复杂度

- 计算时间：某些算法在处理大规模数据时，计算时间呈指数级增长，导致无法在合理时间内获得结果。
- 最优解的计算：在许多情况下，算法只能找到近似解，而无法保证找到最优解。

空间复杂度

- 存储需求：某些算法需要大量的存储空间，可能超过实际可用的资源。
- 适用范围：
  - 特定问题：许多算法只能解决特定类型的问题，不能通用应用于所有问题。
  - 数据依赖：算法的性能和结果往往依赖于输入数据的特性，不同的数据集可能导致显著不同的表现。

2.2.1 算法设计

- 正确性：设计和实现正确的算法具有挑战性，容易出现逻辑错误。
- 效率：设计高效的算法需要深入理解问题和优化技巧，不易实现。

2.2.2 算法极限

- 不可计算问题：存在一些问题（如停机问题）无法通过算法解决，是计算理论中的基本限制。
- 随机性和不确定性：某些问题需要处理随机性和不确定性，增加了算法设计的复杂性。

3 计算模型

3.1 可计算性

算法问题的分类方法：如果一个问题没有算法解决，它就是不可计算的，反之，则是可计算的。

课程中提到的铺瓷砖例子中，我们设想一种算法来验证瓷砖是否能铺满任何大小的房间，但指出这样的问题因涉及无限次测试而不可行。在计算机中处理无限问题的有非常大的难度，计算机在有限时间内只能处理有限的信息，无法完成无限次的测试。

3.2 早期计算模型的建立

计算机处理：

- **历史背景**：在计算机发明前，数学家已经在讨论可计算性。早期的计算理论研究了哪些问题可以通过自动化手段解决。
- **基本模型**：数学家构建了简单的机器模型，用于研究计算能力。在这个模型下，提出了判断什么是可计算的，什么是不可计算的。
- **数的表示**：机器模型需要处理对象（如整数），这些对象需要用符号表示。基本的符号包括 0 到 9，扩展可以表示小数点和有理数。
- **文本的处理**：处理文本时，使用字母和空格表示不同的单词和句子。
- **图像的处理**：现代技术通过将图像分割为点阵，记录每个点的信息，将图像转化为计算机可处理的数字形式。
- **数字化过程**：所有要处理的信息都需要通过编码转换为计算机内部的符号。
- **模型的局限性**：计算机处理的符号总数是有限的，这是其能力的一个局限性。

带子模型：

- **具体内容**：使用一个很长的带子分成一个一个的小格，每个格子存放一个符号。带子两端可以无限延伸。
- **符号表**：带子上的符号属于一个有限的符号表（Alpha）。
- **读头**：机器有一个读头，可以读取带子上的符号，进行分类，并决定是否改写符号，移动读头位置。
- **机器操作**：该机器可以执行四类操作：读取符号、判断符号、决定是否改写符号、移动读头。
- **理论证明**：简单的机器模型实际可执行现代计算机的所有操作，具备现代计算机的所有基本要素。
- **总结**：展示了早期计算理论中的关键概念，证明简单机器模型具有强大的计算能力。

3.3 计算模型和图灵机

创始人艾伦·图灵，指出图灵机是用来描述和解决各种问题的模型。如果一个问题在图灵机上找不到相应的程序，则该问题就是不可计算的。

图灵机的工作原理是根据当前状态（带子上存储的内容）决定下一步操作。操作依赖于当前状态，计算机能够做的所有事情都基于这一点。图灵机模型看似简单，但实际上能够执行加法等操作。例如，图灵机可以读取两个数并通过一系列步骤完成加法操作，将结果写回磁带。

除了图灵机，还有其他数学家提出了不同的计算模型，包括 Lambda 演算、生成系统和递归函数等。这些模型虽然形式不同，但计算能力与图灵机等价，能够解决相同的计算问题。

图灵机能够解决任何有效可解的算法问题。这一结论虽然看似模糊，但在数学和计算理论中有重要意义。有效可解问题的定义需要进一步讨论和明确。

虽然不同的计算模型有各自的特点和复杂度，但在计算能力上都是等价的，能够解决相同的计算问题。数学家通过比较这些模型，最终接受了这一结论。

4 不可计算问题

4.1 不可计算问题中的瓷砖铺设问题

4.1.1 讨论要点

在无限平面上有两个点 V 和 W，需要使用特定形状的瓷砖将它们连接。问题是给定这种瓷砖形状，是否能铺设出一条连接 V 和 W 的路径。

如果允许瓷砖在所有方向（上下左右）移动，该问题是可判定的。如果限制瓷砖只能向上或向下移动，该问题则是不可判定的。[1]

数学描述

设平面上的两个点为 $V(x_1,y_1)$ 和 $W(x_2,y_2)$，我们希望找到一种方法使用给定形状的瓷砖连接这两个点。

点的定义

$$V = (x_1,y_1), \quad W = (x_2,y_2) \tag{1}$$

瓷砖的定义

假设瓷砖的形状可以通过一组矢量定义，例如：

$$T = \{t_1,t_2,\ldots,t_n\} \tag{2}$$

其中每个 t_i 都是一个二维向量 $t_i = (a_i,b_i)$。

路径的定义

路径可以表示为一系列的瓷砖位置，每个位置是由前一个位置加上一个瓷砖矢量组成的：

$$P = \{V, V+t_1, V+t_1+t_2, \ldots, V+t_1+t_2+\cdots+t_k = W\} \tag{3}$$

其中 $t_i \in T$。

不可计算性证明

通过构造一个对应的图灵机问题，可以证明瓷砖铺设问题是不可计算的。

4.2 3n+1 问题（角谷猜想）

在不可计算问题的章节中，提及了 3n+1 问题，又称 Collatz 猜想，是一个著名的数学问题。其描述如下：给定一个正整数 n，进行以下操作：

$$n \to \begin{cases} \frac{n}{2} & \text{如果} n \text{ 是偶数} \\ 3n+1 & \text{如果} n \text{ 是奇数} \end{cases} \tag{1}$$

如此反复进行，最终是否总会回到 1？

4.2.1 公式表示

用递归的形式表示 3n+1 问题，可以写成：

$$a_{k+1} = \begin{cases} \frac{a_k}{2} & \text{如果} a_k \text{ 是偶数} \\ 3a_k+1 & \text{如果} a_k \text{ 是奇数} \end{cases} \tag{2}$$

其中 $a_0 = n$。

4.2.2 示例

假设 $n = 6$，计算过程如下：

$$6 \to 3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1 \tag{3}$$

4.3 停机问题

停机问题是计算机科学中的一个重要问题，证明了存在一些程序无法判断其自身是否会停止运行。[2]

问题描述

设计算一个算法，输入一个程序 P 和一个输入 I，判断程序 P 在输入 I 时是否会停止运行。

图灵机描述

可以用图灵机来形式化描述停机问题：
给定图灵机 M 和输入字符串 w，停机问题问是否存在一个算法可以决定 M 在输入 w 时是否会停机。

公式表示

用数学符号表示停机问题：

$$H(M,w) = \begin{cases} 1 & \text{如果} M \text{ 在输入} w \text{ 时会停机} \\ 0 & \text{如果} M \text{ 在输入} w \text{ 时不会停机} \end{cases}$$

不可判定性证明

通过对角线化和归谬法，可以证明停机问题是不可判定的。即：

假设存在一个算法 H 能够解决停机问题，我们可以构造一个程序 D，它对自身进行检查并导致矛盾：

$$D(x) = \begin{cases} \text{停止} & \text{如果} H(x,x) = 0 \\ \text{无限循环} & \text{如果} H(x,x) = 1 \end{cases}$$

如果 $H(D,D) = 1$，则 D 无限循环，矛盾；如果 $H(D,D) = 0$，则 D 停止，矛盾。

因此，停机问题不可判定。

5 算法复杂度

复杂度分为时间复杂度和空间复杂度，而时间复杂度是计算机的局限性占比最大的一部分，因为许多问题中，计算机的运行效率即使已经比人类高出很多，但仍旧由于问题的复杂性而无法解决。

5.1 二分法

逐个检查名字是否在名单中，最坏情况下需要 $2N$ 次操作。而通过在名单末尾添加目标名字的方法，将复杂度减少为 $N+1$。

可是，这样效率对于计算机而言仍然是慢的。于是，人们发明了二分搜索法，通过对名单进行排序，每次查找中间的名字，从而将复杂度降为 $logN$。[3]

5.2 排序问题

- **上界：**
  - 表示通过实际设计的算法能达到的最好效果，即最坏情况下的最高代价。
- **下界：**
  - 通过数学证明得出的最优解，即问题在理论上的最小代价。
- Closed Problem：
  - 如果找到的算法复杂度与下界一致，问题被称为“封闭问题”，无需再优化。
- Open Problem：
  - 如果找到的算法复杂度与下界差距很大，问题仍有优化空间。这类问题具有研究价值，吸引计算机程序设计者的关注。
- **排序问题的代价：**
  - 为了进行有效的查找，必须先对无序名单进行排序，这也需要时间和资源。
  - 传统排序方法的复杂度为 $O(N^2)$。
- **改进排序方法：**
  - 提出了改进的排序方法，通过分组排序，然后合并，提高效率，复杂度为 $O(N\log N)$。

5.3 汉诺塔问题

机器性能与计算问题：虽然更快的机器可以解决一些计算问题，但某些问题的复杂性使得机器性能提升无法完全解决。

汉诺塔问题介绍：汉诺塔问题涉及将一个塔从一个柱子移到另一个柱子，同时遵守大的不能放在小的上面的规则。

具体操作步骤：详细讲解了如何逐步移动塔层，确保每次都遵守规则。三个塔层的移动需要 7 步。

递归算法：汉诺塔问题的通用解决方法是递归算法，其复杂性为 $O(2^N)$。

实际应用中的复杂性：假设塔有 64 层，每层移动花费 5 秒，完成所有移动需要约 3 万亿年，远超宇宙年龄，显示出问题的巨大复杂性。[4]

6 易解性问题和难解性问题

6.1 复杂性的代价

即使拥有速度极快的计算机（每秒一万亿次运算），有些问题由于算法复杂性太高，仍然需要极长的时间才能解决。

6.2 具体实例

举例说明不同复杂性的算法在处理不同规模问题时所需的时间，强调指数增长和阶乘增长的不可行性。

例如，一个复杂度为 $O(n^2)$ 的算法在处理规模为 n 的问题时，其运行时间为：

$$T(n) = k \cdot n^2$$

其中 k 是常数。

然而，一个复杂度为 $O(2^n)$ 的算法在处理相同规模的问题时，其运行时间为：

$$T(n) = k \cdot 2^n$$

计算时间会长达几亿年甚至更久。

6.3 好的算法和坏的算法

好的算法：在多项式时间内解决问题，成本可接受。即，复杂度为 $O(n^k)$，其中 k 是常数。

坏的算法：需要指数时间或阶乘时间，成本高不可接受。即，复杂度为 $O(2^n)$ 或 $O(n!)$。

6.4 计算模型无关性

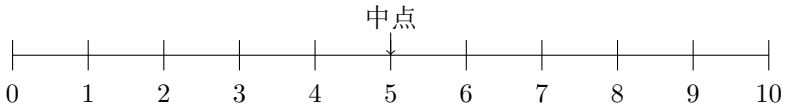
好的算法在不同的计算模型中都表现良好，坏的算法在任何模型中都表现差。这种性质称为鲁棒性。不仅问题的可计算性具有鲁棒性（即在不同模型中可计算性不变），好的算法的判定也具有鲁棒性。

算法优化和寻找最佳算法在计算机科学中占据重要地位，就像体育运动员追求更快更高更强一样，科学家们也在不断追求更高效的算法。

7 难解性问题实例

7.1 查找问题的重要性

在数据库中查找关键字是耗时的任务，但有效的分类算法可以显著提高效率。例如，通过二分查找法，查找一百万个用户的电话本只需进行约 $\log_2 1000000 \approx 20$ 次查找。



7.2 数据挖掘（Data Mining）

没有分类的数据中查找信息的困难。数据挖掘算法的有效性通常较差。使用计算机辅助工具提高软件质量是很重要的，如通过计算机进行测试和验证。

7.3 三重指数问题

三重指数问题及其计算复杂性，举例说明当 $n = 4$ 时，计算结果的复杂性极高。即使是简单的整数和加法操作，其复杂性也可能非常高，涉及到指数级别的复杂性。当加入更多操作如乘法时，问题的复杂性进一步增加，导致不可判定的问题。

7.4 空间换时间

当前自动化工具面临的最大挑战是运行速度慢，虽然有解，但执行速度非常慢。设计高效算法的重要性，尤其在处理复杂问题时。

是否可以通过增加空间来换取时间，但指出在某些情况下，这样的换取并不可行。举例说明，当问题规模达到一定程度时，即使整个宇宙的质子都用来存储信息，也无法解决问题。

7.5 排课问题

教务部门需要安排课程，确保每个老师不会在同一时间出现在不同教室，每个学生不会同时上两门课。使用尝试和回溯的方法（backtracking）来解决问题，即一步步尝试，如果遇到冲突则退回重试。

7.6 拼图问题

类似于拼图的排列问题，需要将九块小块按颜色匹配排列。计算这种排列的复杂性非常高，比如 3x3 的拼图有 $9! = 362880$ 种可能性，如果是 5x5 的拼图则复杂性更高，达到 $25! \approx 1.55 \times 10^{25}$，可能需要 533 万亿年才能计算完。

7.7 回溯算法（backtracking）

回溯算法广泛应用于探索和设计复杂问题的解决方案。每次尝试一个可能性，如果行不通则退回一步重新尝试。

7.8 复杂性的代价

回溯算法提供了一种设计程序的方法，从理论上可以找到解决方案，但需要权衡计算代价。强调了算法设计中复杂性和实用性的平衡。虽然回溯算法可以找到问题的解决方案，但其计算代价可能非常高，实际应用中需要考虑到这个代价。提到在中学玩过的类似八卦图的游戏，通过探索和回溯找到解决方案，这也是回溯算法的一个例子.

8 新计算范式

8.1 并行计算

8.1.1 并行计算的基本概念

将复杂任务分解为多个小任务，并同时进行处理，从而提高计算效率。在处理大规模数据和复杂计算问题时，显著减少计算时间。

8.1.2 并行计算的挑战

包括任务分解的复杂性、任务之间的同步和通信，以及资源分配的问题。在实际应用中的案例如气象预报、大数据分析和高性能计算等领域的应用。

8.2 随机算法

8.2.1 并发技术和计算复杂性的问题

引入第二种方法，即随机算法，用于解决一些计算难题。随机算法的原理是通过抛掷硬币来决定计算路径。讨论随机算法的两种类型分别是拉斯维加斯算法和蒙特卡罗算法。

- **拉斯维加斯算法：**保证正确答案，通常能提高计算效率，但不保证每次输入都有效率提升。
- **蒙特卡罗算法：**在高概率下有效，能减少计算错误，但不能完全避免错误。

8.2.2 随机数生成的重要性

提出使用物理手段（如机械手臂抓取沙子）或伪随机数生成器来模拟随机数的生成。

8.3 量子计算

8.3.1 量子状态的不可确定性

通过例子解释量子叠加态，例如薛定谔的猫。量子比特（Qubit）的概念及其在量子计算中的重要性。量子比特可以同时处于多个状态，从而实现海量并发计算。

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

其中，α 和 β 是复数，且满足 $|\alpha|^2 + |\beta|^2 = 1$。

8.3.2 量子计算的独特性质

量子纠缠现象及其在瞬间通信中的应用。量子纠缠如何实现超越传统通信速度的瞬时状态传递。

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \tag{1}$$

8.3.3 量子计算的应用和挑战

量子计算在解决复杂计算问题上的潜力，如提高 NP 问题的求解效率。量子计算在密码学中的应用和潜在风险，特别是因数分解问题对现有密码学的挑战。

8.3.4 当前量子计算的发展状况

世界各地对量子计算的实验和研究进展。现有的量子计算机仍在发展的早期阶段，但已展现出巨大的潜力。经典计算机与量子计算结合的仿真方法。

8.4 零知识证明

零知识证明是一种方法，可以让某人证明自己有某种能力或信息，而不泄露该能力或信息的具体细节。

- **投票的例子：**通过零知识证明，投票系统可以统计总票数和投票结果，但无法追踪具体是谁投了哪种票，从而保护隐私。
- **共享账户的例子：**在多个用户共享一个账户的情况下，管理者可以知道总的收入和支出，但无法知道具体是哪一个用户在何时进行了操作。
- **任务完成能力的例子：**某人可以证明自己有能力完成某个任务（如三色图着色），但不透露具体完成任务的方法。

8.4.1 零知识证明的应用

零知识证明在实际应用中有部分应用，如密码学和保密系统。具体例子说明，使用 Alice 和 Bob 的例子，说明如何通过零知识证明来验证 Alice 有能力完成任务，而 Bob 无法学习到具体方法。多次随机抽取和测试，使得 Bob 逐渐相信 Alice 的能力，但无法获取任何有用的信息。

零知识证明算法的原理，通过多次随机测试来验证被证明者的能力，而不泄露任何细节。强调了零知识证明在计算机科学中的重要性，特别是在密码学和保密系统中的应用。

参考文献

[1] Turing, A. M. (1936). "On Computable Numbers, with an Application to the Entscheidungsproblem". Proceedings of the London Mathematical Society. 42: 230–265.

[2] Sipser, M. (2006). Introduction to the Theory of Computation. Cengage Learning.

[3] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press.

[4] Gerety, C., & Cull, P. (1986). "Time complexity of the Towers of Hanoi problem". ACM SIGACT News. 18(1): 80-87.