Notes of Computational Complexity — Introduction

ArtsEpiphany

June 1, 2022

Abstract

这是我整理的《Computational Complexity — A Modern Approach》读书笔记的 Introduction 部分，主要介绍了计算相关的一些基本概念，计算理论的研究内容以及相关的问题。

本节内容较为简单通俗，希望能够让人们对于计算理论有一定的了解并产生兴趣。如果愿意，欢迎阅读接下来的部分。当然，遇到不懂的地方也不要忘了可以参考原书，毕竟我的笔记可能会遗漏自以为众所周知的内容，或是有其他的问题。翻看原书或许能有所帮助。

如果发现问题请联系 ArtsEpiphany@gmail.com。

1 计算的概念

计算（Computation）的概念早已存在，但长期没有被严谨地定义。在上一世纪，人们提出了图灵机、lambda 演算、元胞自动机等在可计算性上相互等价的计算模型（如果一个问题可以被其中之一求解，则可以被这些之中任何一个模型求解），成功地定义了计算。具体说来，问题，可以视为输入到输出的函数，而在某个计算模型下计算这个函数的方法，我们称之为算法（Algorithm）。邱奇-图灵论题（Church-Turing Thesis）声称，所有计算或算法都可以由一台图灵机来执行，也就是说，我们可以通过这些模型来研究计算相关的问题。同样地，当我们用自然语言叙述出一个求解问题的计算方法时，我们也可以自然地相信，它也对应着一个图灵机上的算法。

计算的概念引入了两个研究方向。第一个方向是，哪些问题是可（被图灵机等）计算的。之所以有必要研究，是因为存在一些不可计算的问题：任何一个求解该问题的（正确）算法，都存在特定的一些输入，使得该算法无

法在有限时间内给出答案。[1] 这一部分不是这本书的主要内容。第二个方向是计算复杂度理论（Computational Complexity Theory），计算复杂度可以通俗理解为问题的“难度”，我们会稍后给出更严谨的定义。

2 计算复杂度

我们来考虑两个整数的乘法。我们可能会选择列竖式求解，这就是一个求解的算法，这其中，一位数的加法与乘法可以被视为一个基本操作。[2] 现在考虑一个 a 位整数与一个 b 位整数的乘法。首先，我们需要进行 ab 次一位数之间的乘法，得到 ab 个一位或两位数。我们将它们展开，就会得到如下所示的至多 $2ab$ 个一位数，我们需要将它们按位求和。

[1]不难发现，对于每个输入 I，一定存在一个算法，在输入为 I 时输出正确的结果，而在输入不为 I 时陷入无限循环。（这很容易做到，我们只需要将输入输出编码进算法之中即可，“如果输入为 I 则输出 ***，否则死循环”。）这样的算法们拼凑在一起，是不是可以用来求解不可计算问题呢？当然不行。这是因为算法必须是有限大小的，我们不能将无限个算法拼接在一起。所以我们也可以理解为，假设 $I(A)$ 是使得（正确）算法 A 能够在有限时间内输出的所有输入的集合，那么对于不可计算问题而言，对于任意有限算法集 S，有 $\bigcup_{A\in S} I(A)$ 不等于所有输入构成的集合，即“全体输入的集合不可被有限个 $I(A)$ 集合覆盖”。

[2]我们可以将一位数之间的加法与乘法的结果全部写入算法中去，例如“3 乘 5 的结果是 15”，但是不能把所有的整数之间的乘法的结果全部写入算法，因为后者是无限的。当然，如果愿意，也可以把所有一位数及两位数之间互相的加法与乘法的结果全部写入算法中去（这是有限的），并视为基本操作，这样在计算时可以两位两位地去计算。然而，这并没有为算法的效率带来本质的提升。

$$
\begin{array}{ccccccc}
 & & & 9 & 8 & 7 & 6 \\
\times & & & & 5 & 4 & 3 \\
\hline
 & & & & & 1 & 8 \\
 & & & & 2 & 1 & \\
 & & & 2 & 4 & & \\
 & & 2 & 7 & & & \\
 & & & & 2 & 4 & \\
 & & & 2 & 8 & & \\
 & & 3 & 2 & & & \\
 & 3 & 6 & & & & \\
 & & & 3 & 0 & & \\
 & & 3 & 5 & & & \\
 & 4 & 0 & & & & \\
4 & 5 & & & & &
\end{array}
\tag{1}
$$

这些数需要多少次加法呢？我们首先把这至多 $2ab$ 个数按位逐个加起来，而每次进位就简单地在高位上写一个 1（先不去加它）。这个时候我们已经进行了不超过 $2ab$ 次加法，还有至多 $2ab$ 个 1 等待求和。接下来我们逐轮去处理这些进位：每一轮加上上一轮进位产生的 1，而这一轮同样可能产生进位，我们把这些 1 写在对应的位上，但在这一轮中不加上它们。注意到，这次进位变得不那么常见了：在某位发生进位之后，还要再加 10 个 1 才会再次进位。我们这样去统计进位的次数：如果是某位的第一次进位，我们单独统计，由于只有 $a+b$ 位，这一部分一共有至多 $a+b$ 次进位；否则的话，每 10 次进位才会带来一次更高位的进位（它们还会继续带来进位），所以 $2ab$ 次进位只会带来至多 $2ab/9$ 次进位。这样一来，我们不难发现存在一个常数 c 使得加法与乘法的总次数不超过 $c \cdot ab$。

现在考虑另外一种计算方法，想要计算 x 与 y 的乘积，我们也可以直接把 x 个 y 加起来。同样假设 x 有 a 位，y 有 b 位，那么我们不难发现存在一个常数 c 使得我们需要的基本操作的数量不少于 $c \cdot b \cdot 10^a$。

不难发现，随着 a 的增长，第二种算法所需要的基本操作的数量终将远远多于第一种算法所需要的基本操作的数量。这一特性是与 c 的具体取值

无关的，而计算设备的快慢与否，差异就类似这个常数，比如“这台设备比那台设备快一倍”，就是只需要使用一半的时间就可以求出答案。这里我们称第一种算法“更高效”，而算法的高效与否对求解用时的影响是决定性的：只要问题规模足够大，手工计算答案采用更高效算法的人也可以算得比采用低效算法的计算机更快。[3]

如此一来，我们就可以衡量问题的复杂程度：是否存在一个求解算法使得对于规模为 n 的输入只需要 $c \cdot f(n)$ 次基本操作呢？我们还可以据此将问题依据它们的复杂程度划分为不同的类别。这里展示一个复杂的问题：给定 n 个人，每两个人之间要么是朋友要么不是朋友，现在已知哪些人之间是朋友，求一个包含了最多可能的人的集合，使得其中的人两两都是朋友。我们的导航软件可以从数千条可行道路里面找到最为便捷的，但上述问题在 $n = 1000$ 的特定输入下，即使是目前最先进的计算设备也无法在合理的时间[4]内求出精确解，这就是问题复杂度的差异。

3 相关研究

回顾那个问题：对于一个问题，是否存在一个求解算法使得对于规模为 n 的输入只需要 $c \cdot f(n)$ 次基本操作呢？如果答案为是的话，往往相对容易，我们只需要提出一个符合要求的算法即可。因此，这一领域入门较为便捷，可以不借助大量高深的数学知识，凭借自己的天赋与汗水取得一定的成果。而否的那一部分就比较困难了，我们需要证明“不存在一个算法怎样怎样”，这往往是需要技巧的。好在，我们也不完全是漫无目的，因为根据邱奇-图灵论题，我们只需要证明在某种计算模型上的算法均不符合要求即可。尽管如此，这方面的证明依然是很有难度的，所以我们的另一个主要任务就是探究不同问题之间的关系，哪些问题是相同难度的，哪些问题更容易，哪些问题更困难。这也会是本书的主要内容。

下面是几个计算理论相关的有趣问题。

- 上文所述的朋友问题有高效的求解算法吗？

[3]乘法有更高效的算法吗？其实是有的，对于两个 n 位数的乘法，快速傅立叶变换可以达到 $c \cdot n \log n \log \log n$ 的计算效率。

[4]比如：1 年。

- 随机化能带来更高效的算法吗?
- 如果我们只寻求近似解，或者允许以一个非零概率输出错误解，可以找到一个更高效的算法吗?
- 复杂的问题的存在会带来哪些好处?
- 量子计算机比经典计算机优越多少?
- 数学定理的证明和对证明的验证可以由程序进行吗?

这一领域还有更多有趣的问题等待探索。