21120491：高级数据结构与算法分析 **2023-2024 学年春夏学期**

Lecture 10: NP 完全性

教师：刘金飞，助教：吴一航 日期: 2024 年 4 月 30 日

10.1 引言

激动人心的，我们来到了这门课的一个新的转折点。在之前的讨论中，我们介绍了几种算法设计策略，看到了很多能巧妙利用问题结构从而可以快速解决的问题，介绍了非常多如何 "' 解决' 某个问题的方法。现在，我们需要考虑的问题是，一个问题有多难，是否存在一种方法使得它能够被高效地解决? 这个问题显然比先前的考虑更加抽象，也需要更多的技巧和思考。在通常的意义下，我们认为能 "快速解决" 或 "高效解决" 的问题是指存在一个多项式时间的算法能够解决的问题（后面会解释原因）。而所谓多项式时间，是指算法的时间复杂度与输入的长度之间是多项式关系。我们来看三个经典的存在多项式时间算法问题：

1. 最短路径问题：给定一个有向图 $G = (V, E)$，即使是带负权的，我们可以在 $O(|V||E|)$ 的时间内找到从单一源顶点开始的最短路径；这是多项式时间的，因为图中我们的输入规模可以视为 $|V|+|E|$（如果我们使用邻接表）；

2. 欧拉回路问题：是否存在一个回路（即起点终点是同一个顶点）使得它恰好经过图中每条边一次? 这个问题可以在 $O(|E|)$ 的时间内用深度优先搜索解决，这在数据结构基础课程中已经熟悉；

3. 2-CNF 可满足性问题（简称 2-SAT 问题）：回忆离散数学，我们称一个逻辑表达式是 k-CNF（即 $k-$ 合取范式）的，如果它是由 k 个子句的合取（$\wedge$）构成的，每个子句是由 k 个变量或者它们的否定构成的析取（$\vee$）。2-SAT 问题就是判断一个 2-CNF 逻辑表达式是否存在对其变量的某种 0 和 1 的赋值，使得它的值为 1，这个问题可以在多项式时间内解决。例如

   $$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

   满足赋值条件 $x_1 = 1, x_2 = 0, x_3 = 1$。这一问题与图论中的强连通分量有关，我们在后续介绍了归约的概念后会详细讨论，总之也是多项式时间可解的。

然而，有一个经典的容易出现的混淆的问题是 0-1 背包问题：回忆 0-1 背包问题的动态规划算法，我们最终的时间复杂度为 $O(nC)$，其中 n 是输入的物品个数，C 是背包最大容量。乍一看这是给出了多项式时间的算法，然而我们需要注意到，C 在输入时是以二进制的方式表达的，其二进制表示的长度是 $\log C$，因此这个算法的时间复杂度实际上是指数级别的（当然这个复杂度的确非常有迷惑性，因此我们称其为伪多项式的）。当然对于 n 而言，n 个物品实际上对应于 n 个二进制物品价值和物品重量输

入，前面的图问题中 n 个顶点和边实际上也是 n 个二进制编码，所以不会发生 C 这种单独一个数字导致的的窘境。

然而，这一窘境引出了一个自然的问题：0-1 背包问题是否存在一个多项式时间内可以解决的算法，很遗憾，至今我们无法对这个问题给出答案，这也就引入了我们今天要讨论的关于问题的困难程度的话题。前面的三个多项式时间的例子以及前面关于算法设计的讨论仿佛给了我们很大的信心，仿佛所有问题都能找到高效的算法，但我们只要对上面三个问题稍微做一点点变化，就会发现事情并不是那么简单：

1. 最短路问题变体：如果带负圈的最短路问题中我们的要求不是找到负圈就结束，而是给出具有最短路的无环路径，那么这一问题就变得困难了，目前无法在多项式时间内解决这个问题；
2. 哈密顿回路问题：起点和终点是一个顶点，途中经过图中所有其他节点且只经过一次。即将欧拉回路每条边经过一次改为了每个点经过一次，然而判断一个图是否存在哈密顿回路目前也不存在多项式时间的算法，尽管看起来和欧拉回路区别不大；
3. 3-CNF 可满足性问题（简称 3-SAT 问题）：判断一个 3-CNF 逻辑表达式是否存在对其变量的某种 0 和 1 的赋值，使得它的值为 1。这个问题目前也没有多项式时间的算法，尽管和 2-SAT 问题仅有一字之差。

由此我们发现，似乎问题与问题之间亦有差别：有的问题能找到很巧妙的方法在多项式时间内解决，而有的问题虽然看起来和前者差不多，但至今也没有找到多项式时间的算法。这也就带来了这一节讨论的核心：我们如何形式化地定义问题的困难程度，如何对这些问题按照困难程度进行分类，这些困难程度背后又有什么有趣的内涵？为了讨论这些问题，我们需要首先形式化地定义问题以及计算这些问题的形式化计算模型。接下来就让我们开始正式的讨论。

10.2 形式语言

为了形式化地定义问题以及计算模型，我们首先需要给出某种编码。这就是形式语言要干的事情。首先，考虑一个至多可数的集合 Σ，称为字母表（alphabet），其上的字符串（string）定义为有限个元素（包括 0 个）的有序连接。我们称所有这样的字符串记作 Σ^*。

但往往，这样的东西本身并不具备含义。一方面，我们需要一个解释来表明它到底在表达什么，比方说，下面这个二进制字符串（$\{0,1\}$ 上的字符串）：

$$0100100001100101011011000110110001101111$$

可以表示十进制整数 310939249775 或者 ASCII 字符串 Hello：需要明确指出它表示的是什么东西。另一方面，这样的结构过于简单，以至于不能表达任何有意义的东西，这就像是语言的无意义连接，比方

说：

军进犯大林嘴鸟军强脾气穷就怕嗲气

这是一段脸滚键盘打出来的字，它也是中文字符上面的字符串，但我们认为它毫无意义。因此我们取所有字符串 Σ^* 的一个子集 L，并将其称为形式语言（formal language）。这个子集的定义在我们这里是由自然语言给出的，而如果是在描述复杂度那里，我们需要对它给出更加精细的刻画。

下面我们主要要讨论两种问题：

1. 给定语言 L，判断某个字符串 ω 是否在 L 中，这被称为判定问题（decision problem）；
2. 给定语言 L，寻找某个字符串 ω 使得 ω 在 L 中，这被称为搜索问题（search problem）；
3. 给定语言 L_1, L_2，计算某个从 L_1 到 L_2 的函数，这在后面会以另外一种方式实现。

在此，我们需要注意形式语言的几个性质：

1. 至多可数的集合上的有限字符串是可数的；这就是为什么我们在此定义的时候用了至多可数，而课件上用的是有限——在我们的语境下，这不会有什么影响，而且至多可数和后面讨论的逻辑系统能够联系起来。
2. 语言可以做衔接、并、交、补、Kleene 星等运算，这些运算的定义参见课件。

最后，我们来看一个例子：如果给出一个自然语言下的问题，如何将其转变成一个形式语言的判定问题？显然，能够完成翻译的问题只有那些答案是 “是” 或者 “否” 的问题，例如：

给定一个图，判定它是否有 Hamilton 回路。

首先，我们表明：

引理 10.39 *所有图构成的集合是可数的。*

这个引理的证明不难。显然，它的顶点数有限，固定顶点数的图个数有限；可数个有限集的并集依然是可数的，因此我们可以构造一个 Σ^* 包含所有图而不包含所有别的东西：只要把这个可数集到自然数的双射变成到 $\{1\}^*$ 的双射即可，即取 $\Sigma = \{1\}$，这样字符串由几个 1 组成就意味着它是编号为几的图。而我们的形式语言 L 就定义为所有有 Hamilton 回路的图对应的字符串构成的集合。在这里我们看到，它是 Σ^* 的一个子集，而它的定义是通过一个谓词（predicate）“有 Hamilton 回路” 给出的，即这一形式语言可以写为

$$L = \{x \in \Sigma^* \mid Q(x)\},$$

其中 $Q(x)$ 是谓词 “x 是一个有 Hamilton 回路的图”。因此，我们的判定问题就是给定一个字符串 x，判定它是否在 L 中。于是，一个判定问题的复杂度就被转变成了另一个问题：一个可数集的子集能有 “多复杂”？这就需要我们接下来进一步定义计算模型后进行讨论。

10.3 计算模型

考虑计算过程，我们最基本的想法就是所谓的“算法”的概念：一个有限的指令集中取出的有限条指令，能在有限时间内确定性地完成。在 1930 年以前，从来没有过可计算性这样的概念，当我们给出一个问题，我们自然假定它是可计算的。然而，自 Hilbert 的幻梦被 Gödel 打破以来，其他一些要求给出某种算法的问题也遭到质疑：是否有一些问题用算法是不能解决的？但是，这样的直觉想要得到严格的证明，天才的图灵发明了一种泛用的计算模型用来描述可计算函数：

定义 10.40 (图灵机) *考虑一个双向无限的磁带（tape），将其划分成一个一个的方格（cell），有一个磁头（head），磁头中有一个有限状态机，其中包含有限个状态 $q \in Q$。每个方格要么是空的（$\square$），要么是写有字符 $s \in S$ 的（S 为有限集）。在每一步操作中，它可以完成：*

- *磁头内部状态的转变；*
- *把扫描到的符号 s 改写成 $s' \in S \cup \{\square\}$；*
- *往左（L）或往右（R）移动磁头；*

这由一个偏函数（只在定义域的一个子集上有定义的函数）$\delta : Q \times (S \cup \{\square\}) \to Q \times (S \cup \{\square\}) \times \{L, R\}$ 决定（即当我们处于某个状态、读取到某个字符时，这个函数告诉我们下一个状态、应该写下的字符以及应该往哪个方向移动磁头），我们将这个偏函数称为图灵程序（Turing program）。

每一个计算开始之前，我们都将磁头置于起始方格（starting cell），它是写有内容的最左侧方格，并让其处于起始状态（starting state），然后将输入放在起始方格右侧，并保证纸带其余部分为空。如果磁头抵达状态 $q_h \in Q$，称作停机状态（halting state），则视作计算结束，并将磁带上从起始方格开始的内容视作输出。我们将图灵机的构型（configuration），记作：

$$c = t_m t_{m-1} \dots t_2 t_1 \underline{q_i} s_1 s_2 \dots s_k$$

其中 s_i 和 t_i 分别是一直到最右侧（最左侧）的非空格子上的内容，当然，如果没有则可以写个 $\square$；q_i 指当前磁头所处的状态，当前磁头指向 s_1 所在的方格。

如果磁头进入状态 $q \neq q_h$ 而 δ 未定义（不移动），则称计算中断，不将其视作停机。

所谓的图灵计算（Turing computation）就指图灵机的一列构型 $c_0, c_1, \dots, c_n$，它按照如上的叙述由一个图灵程序所描述。

我们来看两个基本的图灵机的例子：

例 10.41 *我们要计算函数 $f(x) = x + 2$，如果用纸带上连续的 1 的个数表示数字大小（例如输入 x 就是纸带上有 x 个连续的 1），于是要计算这一函数，我们实际上是希望图灵机能在输入后面多放两个 1。*

当前状态	当前读入字符	下一状态	写下的字符	磁头移动方向
q_1	1	q_1	1	R
q_1	□	q_2	1	R
q_2	□	q_h	1	R

我们可以用如下的图灵机状态转移函数计算这一函数：从上表可以看出 q_1 是起始状态，如果我们看到 1 则说明现在还在输入 x 的区域内，直到我们读到了一个空格 □ 时，我们就要开始写入 1 来做加法了：我们写下一个 1 就进入状态 q_2，然后 q_2 时再写下一个 1，此时就完成了加法，进入停机状态 q_h。

例 10.42 因为我们需要讨论判定问题，因此图灵机最终应该告诉我们“是”或“否”。实际上就是最终图灵机会在纸带上留下一个 1 或 0 代表“是”或“否”。在有些图灵机的定义中，我们会规定写下 1 时代表一个 q_{accept} 状态表示接收，写下 0 时代表一个 q_{reject} 状态表示拒绝，实际上如果我们在进入接受和拒绝状态后再加一个转移函数到停机状态 q_h，就喝我们的定义相容了。

事实上前面的定义的图灵机我们称为确定性图灵机（deterministic Turing machine），还有一种非确定性图灵机（non-deterministic Turing machine）。所谓的确定性图灵机和非确定性图灵机的差异就在于偏函数上。按照上面的定义，它可以给每个状态都赋予**一个**转移，也就是说，确定原来的 $(q, s) \in Q \times (S \cup \{\square\})$，它就只能达到唯一的一个 $Q \times (S \cup \{\square\}) \times \{L, R\}$ 中的元素。非确定性图灵机中，这个函数的值域是原来值域的幂集，也就是说，它有许多可能的路径可选，如果有一条路径可以停机，那它称就会停机。在判定性问题中，只要有一条路径接受，那么整个图灵机就接受，也就是说只有所有路径都拒绝，整个图灵机才拒绝。在后面我们会看到，非确定性图灵机和确定性图灵机在可计算性上是等价的，但是在复杂度上可能大相径庭。

定理 10.43 任何一个非确定性图灵机都可以被一个确定性图灵机模拟，即它们可以计算的函数是一致的。

证明:[sketch] 为了实现这一模拟，最直接的想法就是：因为非确定性图灵机执行时，每一步会产生多种选择，因此所有路径会生成一颗树，确定性图灵机只需要进行 BFS 搜索这棵树即可。

我们用一个三条纸带的图灵机来模拟即可：第一条纸带放输入，第二条纸带模拟非确定性图灵机读到输入的时候的状态和行动，但 BFS 事实上是有一个特定顺序的，图灵机并不知道，所以我们还需要第三条纸带来记录 BFS 的进程，提示第二条纸带下一步应该模拟哪条路径。这样的话，我们就可以模拟非确定性图灵机的行为了。然后我们说明三条纸带的图灵机（更广泛的，多条纸带的图灵机）实际上与单条纸带的是等价的，这一点比较显然，最简单的想法就是把纸带分成三个部分即可。上述说明的详细版本都是比较技术性的，读者可以参考计算理论的教材。

因为非确定性图灵机的所有计算都可以被确定性图灵机模拟，所以显然二者可以计算的函数是一致的。

■

定理 10.44 (Church-Turing 论题) *存在一个现实可行的计算某个函数的方式当且仅当存在一个图灵机计算这个函数。*

通俗而言就是每个算法都有一个图灵机实现。事实上 Church-Turing 论题或多或少是个猜想，所以这还是个论题，并不能称之为定理。并且这一论题不全是数学问题，其中还有丰富的物理内涵。这一论题还有一个推广形式。

定理 10.45 (推广的 Church-Turing 论题) *现实可行的计算系统都可以被图灵机高效地模拟。*

10.4 Gödel 数

下面我们介绍一种典范的方式对形式语言进行编码。我们记 $\mathbb{P} \subset \mathbb{N}$ 为所有素数的集合。则对于至多可数的字母表 Σ，我们可以建立双射 $f : \Sigma \to \mathbb{N}$。然后，我们将每个字符串映到：

$$x_1 x_2 ... x_n \mapsto p_1^{f(x_1)} p_2^{f(x_2)} \cdots p_n^{f(x_n)}$$

其中 p_n 表示第 n 个 $\mathbb{P}$ 中的素数。这样的话，我们就能建立一个有限长度字符串到 $\mathbb{N}$ 的对应，而且它是单的（质因数分解的唯一性）。这样，我们就建立了一个 $L \to \mathbb{N}$ 的对应，其中 L 是任意语言。这样的对应（在更广泛的意义上讲，任何 $C \to \mathbb{N}$ 的单射，其中 C 为可数个数学对象的集合）就被称为一个 Gödel 数。

例 10.46 *图灵机是可以被指派 Gödel 数的，因为图灵程序的每一个分量都是可数的，所以只有可数个图灵机。我们记 M_α 为编码为 α 的图灵机。不难发现：*

1. *对于任意计算，都存在无穷多个 $\alpha \in \mathbb{N}$ 对应完成这种计算的图灵程序，这只要通过插入一些无意义状态就能实现。这个结论通常被称为指标集（indice set）的无限性。*
2. *存在一个图灵机模拟任意编号 α 的图灵机的执行。这个图灵机称为通用图灵机（universial Turing machine）。可以表明，如果原来的图灵机计算时间是 $f(n)$，那么它花费的时间是 $O(f(n) \log f(n))$，这在后面较复杂的几节会用到。*

Gödel 定义这套东西的主要目标是完成 Gödel 不完备性的证明。在此，我们干脆呈现 Tarski 的版本，称为 Tarski 不可定义性定理（Tarski's undefinability theorem），它蕴含了 Gödel 的结果：

定义 10.47 *称 $T : L \to L$ 是语言 L 中的一个谓词（predicate），如果它将字符串 x 映到*

$$s_1 x s_2 x \cdots s_{n-1} x s_n,$$

其中 $s_n \in \Sigma^*$，且对于任意 $x \in L$ 都有 $T(x) \in L$。

定理 10.48 (Tarski 不可定义性定理) *设 L 是一个形式语言，存在 $\mathbb{N}$ 到 Σ^* 的嵌入，记作 ι. 其中的字符串可以用 $g : L \to \mathbb{N}$ 对应到其 Gödel 数，满足以下条件：*

1. *存在函数 $f : L \to \{0, 1\}$，称为字符串的真值；*
2. *存在符号 $\neg \in \Sigma$，使得 $f(\neg x) = 1 - f(x)$；*
3. *对角线引理成立：对于任意谓词 $B : L \to L$，都有一个字符串 a 使得 $a = B(\iota(g(a)))$；*

那么，不存在谓词 $T : L \to L$ 满足 $f(T(\iota(g(x)))) = f(x), \forall x \in L$。

证明： 我们采用反证法证明这个结果。根据对角线引理，考虑一个 $s \in L$，使得 $s = \neg T(\iota(g(s)))$（即取 $B = \neg T$，先调用 T 然后在字符串前面加一个 $\neg$），然后两边应用 f 求真值，就有 $f(s) = 1 - f(T(\iota(g(s)))) \neq f(T(\iota(g(s))))$。 ■

其中的对角线引理暗示的是这个语言可以自指。也就是说，我们可以构造一个谓词，它的真值是它自己的 Gödel 数的真值的否定。这个定理的证明是一个典型的应用对角线法的证明，它的证明方法是构造一个谓词，使得它的真值是它自己的 Gödel 数的真值对应的谓词的否定。

这个定理意味着：

1. 不存在一个谓词 T 使得 $f \circ T \circ \iota \circ g$ 能够判定任意一个字符串的真值；也就是说，$\iota \circ g$ 的 “逆” 是不可完全在 L 中定义的；
2. Gödel 不完备性定理：一个包含基本算数（自然数及其等姓）的公理系统中，存在一个不可证明也不可证伪的命题：取 L 为所有可能的逻辑公式，它可以叙述自然数的性质。而一个逻辑公式可以用 g 来编码。所以，对角线引理就是表明，我们有一个谓词 $S(n)$ 表达 $\varphi(n)$ 为假，其中 φ 是 Gödel 数为 n 的逻辑公式。然后，将 S 应用于它自身的 Gödel 数 $g(S)$，即可得出矛盾，因为若 $S(g(S))$ 为真，则 $\varphi(g(S))$ 为假，其中 φ 是 Gödel 数为 $g(S)$ 的逻辑公式，又 g 是单射，故 $S = \varphi$，所以若 $S(g(S))$ 为真，则 $\varphi(g(S)) = S(g(S))$ 为假，这显然是荒谬的；
3. 停机问题：存在一个不可被图灵机计算的问题：判定任意一个图灵机是否会停机。这个问题的证明是通过构造一个图灵机，它会在某个 Gödel 数 g 上停机当且仅当这个 Gödel 数对应的图灵机 M_g 在自己 Gödel 数 g 上不停机，这个图灵机的存在性就是对角线引理的推论。实际上，证明也是如出一辙的。

上面的证明方法，对角线法，是一种非常重要的方法。我们在后面会在复杂度的语境下重新使用这种方法，并得出一些有意思的结果。

当然，我想提到哥德尔，大家都不会拒绝那段有趣的数学史。在二十世纪初，野心勃勃的 Hilbert 提出了一句振聋发聩的口号：我们必须知道，我们终将知道。因此，他的二十三个问题中的第二个就是算数公理系统的一致性。但是，这个命题很快被 Gödel 所证伪，虽然 Hempel 等当时知名的哲学家并不认

可他的证明，Gödel 不完备性定理很快成为了一个重要的结果，并随后由 Church 的 λ-验算和 Turing 的图灵机，进一步表明了算数公理系统不但存在不可证明的命题，也不存在一种算法来判断一个命题是否为真。进一步地，他们提出了 Church-Turing 论题，就此划定了可计算性的疆界。而我们要讨论的复杂度理论肇源于 Hartmanis 和 Stearns 的论文，我们在后面有所提及，而且这构成了我们在本讲以及以后几讲中谈论复杂度理论的基本框架。另一条涉及复杂度的理论大约起源于 1970s 早期，是由 Blum 等人提出的公理化的方法，这套方法我们鲜有涉及，但也是饶有趣味的。

实际上，对复杂度的意识早在 1950 年代就有苏联科学家做出过探讨，他们探讨了暴力搜索优化问题的解的方法下界。这个问题在西方国家受到关注应当是大约 1965 年 Edmonds 的论文。有意思的是，Gödel 在他与 von Neumann 1956 年的通信中，提出了 $\mathsf{P} = \mathsf{NP}$ 的一个原始形式。而在本讲中最具代表性的人物是 Cook (1971) 和 Karp (1972)，有兴趣的读者可以参见 Sipser 在 1992 年一篇综述 *The history and status of the P versus NP question*，其中详细解释了 $\mathsf{P} = \mathsf{NP}$ 问题的历史，并且对 Gödel 的信件做出了翻译。

10.5 复杂类

在有了前面几节的铺垫后，我们将正式开始讨论我们最开始提出的问题，即如何形式化地定义问题的困难程度，也就是我们需要定义一些困难程度可能不同复杂类，将问题归于这些复杂类中。下面我们定义几个经典的复杂度类，对于读者来说，前两个类大概是最熟悉的，也是最容易接受的，但也是目前看来相当复杂的两个复杂度类。

10.5.1 DTIME 和 NTIME 系列

我们考虑一族函数 $f(n)$。称一个问题是：

1. $\mathsf{DTIME}(f(n))$ 的，如果求解规模为 n 的问题的确定性图灵机能在 $f(n)$ 步之内停机；
2. $\mathsf{NTIME}(f(n))$ 的，如果求解规模为 n 的问题的非确定性图灵机能在 $f(n)$ 步之内停机。

一个形式语言 L 的判定问题的规模指的是输入的长度。这两个类的定义来源于 Hartmanis & Stearns 的论文，在这里我们给出的形式是稍有不同的，忽略了常数的问题，只要注意到他们证明的一个加速定理：

定理 10.49 (Hartmanis-Stearns, 1966) *如果 f 可以被图灵机 M 在 $T(n)$ 时间内计算，那么对于任意常数 $c \geqslant 1$，都有一个图灵机 $\tilde{M}$ 能够在 $T(n)/c$ 的时间内完成同样的计算。*

证明:[sketch] 我们仅证明 $c = 2$ 的情况，其余情况类似。我们将图灵机的磁头中的有限状态机的两步状态转移合并成一步转移，这样的话我们的字母表也需要同步拓宽，即将原先的字符（包括 $\square$）两两组

合。显然我们的计算时间减半。 ■

在此基础上，我们定义几个复杂度类：

1. $\mathsf{P} = \bigcup_{k\geqslant 1} \mathsf{DTIME}(n^k)$；
2. $\mathsf{NP} = \bigcup_{k\geqslant 1} \mathsf{NTIME}(n^k)$；
3. $\mathsf{EXP} = \bigcup_{k\geqslant 1} \mathsf{DTIME}(2^{n^k})$；
4. $\mathsf{NEXP} = \bigcup_{k\geqslant 1} \mathsf{NTIME}(2^{n^k})$。

结合前面的定义理解此处的定义，P 就是确定性图灵机能在多项式时间内停机解决的问题，其余类似。

基于此，我们不难发现 $\mathsf{P} \subset \mathsf{NP}, \mathsf{EXP} \subset \mathsf{NEXP}$，因为确定性图灵机就是一种特殊的非确定性图灵机。而另一个平凡的性质 $\mathsf{NP} \subset \mathsf{EXP}$ 则需要注意到用确定性图灵机模拟非确定性图灵机所需的开销是指数级别的，或者使用另一个等价定义：

定理 10.50 *一个语言 L 是 NP 的当且仅当存在多项式 p 和一个多项式时间的确定性图灵机 M，使得对于任意 x 都有*

$$x \in L \iff \exists u \in \{0,1\}^{p(|x|)}, s.t. M(x,u) = 1.$$

u 被称为 x 关于语言 L 的证明（certificate）。故上式的含义为 x 在 L 中当且仅当存在一个多项式长度的证明使得 M 在多项式时间内接受。

这个定理的证明（也就是两个 NP 的定义等价）实际上并不复杂，我们在此给出一个直觉描述。如果 L 是 NP 的，那么给出一个非确定性图灵机的选择序列即可作为证明，这个序列长度一定是多项式规模的，因为非确定性图灵机在多项式时间内至多完成多项式次分支。而验证只要通过一个确定性图灵机模拟这个选择序列下的执行即可完成。如果存在这样一个证明，那么非确定性图灵机只要不断分支，一定能在多项式时间内分支多项式次，进而凑出这样的一个证明。

除了这几个平凡的关系以外，其它层级的关系（相等还是真包含）大多还是模糊不清的（P 和 EXP 除外）。我们很快会重新回到这个问题上来。

在这一节的最后，我们给出为什么 EXP 和 NEXP 类重要的一个原因：

定理 10.51 *如果 $\mathsf{EXP} \neq \mathsf{NEXP}$，那么 $\mathsf{P} \neq \mathsf{NP}$。*

证明不难，但为了缩减篇幅在此省略。这表明，指数级别的复杂度类研究实际上有助于厘清多项式时间的复杂度类之间的关系。最后，我们给出一个例子：

例 10.52

$$L = \{(m,r) \mid \exists s < r, s \mid m\}$$

这个语言是 NP 的。这用等价定义很容易看出来，因为 s 就是我们所需的证明。因为问题的输入是 m, r，那么输入的长度就是 $\log m + \log r$，而证明 s 是一个小于 r 的数，它的长度是 $O(\log r)$ 的，所以 s 作为证明首先是多项式长度的。然后我们可以在 $\log r$ 的多项式时间内做除法验证，这样就能验证给出的证明是否正确。

如果用非确定性图灵机，直观而言我们直接在第一步尝试所有可能的 s 即可。因为这些尝试只要有一个成功，那么我们就能得到一个正确的分解。

尽管我们知道这个问题是 NP 的，但是我们并不知道它是否是 NP-完全（后面会提及这一概念）的。值得一提的是，质因数分解的困难性是当前许多密码学基础设施（如 RSA 等）被认为安全的基础。因此如果 $\mathsf{P} = \mathsf{NP}$，现在的大量密码的攻击难度将会大大降低。

10.5.2 co-NP 类

下面这个类比较特殊，它是由形式语言的补集所定义的。实际上，我们不妨定义一个更广泛的东西。回顾一下，一个形式语言 L 的补（complement），记作 $\overline{L}$，是所有在 Σ^* 中而不在 L 中的字符串构成的语言。

定义 10.53 设 C 是一个复杂度类，则 *co*-C 称为它的补类（*complement class*），其中的语言定义为：

$$co\text{-}\mathsf{C} = \{L \mid \overline{L} \in \mathsf{C}\}$$

注意，co-C 并不意味着在所有问题意义上的补，而是其中所有语言的补。下面的结果显然：

定理 10.54 $\mathsf{P} = co\text{-}\mathsf{P}$。

这是因为我们只要将回答中的 “是” 和 “否” 互换就可以了——这是确定性图灵机的特权。对于非确定性图灵机，问题就变得没那么简单了。如果有一条路径给出了 “是” 的回答，那么整体回答就是 “是”；而如果简单翻转是和否，那么如果原来有一条路径给出了 “否” 的回答，整体的回答依然是 “是”。因此，co-NP 的结构就显得扑朔迷离了起来。事实上，用另外一种方式给出定义可能更加轻松，这对应 NP 的等价定义：

定理 10.55 一个语言 L 是 *co*-NP 的当且仅当存在多项式 p 和一个多项式时间的确定性图灵机 M，使得对于任意 x 都有

$$x \in L \iff \forall u \in \{0,1\}^{p(|x|)}, s.t. M(x,u) = 0.$$

注意这里把 $\exists$ 换成了 $\forall$。实际上，$\mathsf{P} \subset \mathsf{NP} \cap \text{co-NP}$，这个结果很简单，留作读者练习。更进一步地，我们有 $\mathsf{P} = \mathsf{NP} \implies \mathsf{NP} = \text{co-NP}$。这里同样给出一个例子：

例 10.56 在布尔公式中，所有永真式的集合是 *co*-NP 的。这是因为我们可以用确定性图灵机在多项式时间内完成一个布尔公式的求值。如果它是永真式，那么任意求值的结果都是 1，然后翻转 0 和 1 即可匹配我们上面的定义。

10.5.3 空间复杂度类 PSPACE

在这里，我们需要考虑的是图灵机的纸带花费多大的空间。我们同样定义一个问题是：

1. $\mathsf{DSPACE}(f(n))$ 的，如果求解规模为 n 的问题的确定性图灵机花费 $f(n)$ 长度的纸带；
2. $\mathsf{NSPACE}(f(n))$ 的，如果求解规模为 n 的问题的非确定性图灵机花费 $f(n)$ 长度的纸带。

注意，非确定性图灵机花费的纸带长度是最多的那个分支花费的长度。当然，这里的 “加速定理” 更加显然，只需 “几步并成一步”，也就是在一格纸带里多填充一些东西，扩充字母表即可。因此，这样的定义是良定的。

在这里，我们首先应该考虑以下结果：

定理 10.57 (Savitch 定理)

$$\mathsf{NSPACE}(f(n)) = \mathsf{DSPACE}(f^2(n))$$

这个估计的证明并不困难，只要考虑怎么用确定性图灵机模拟非确定性图灵机即可。类似地，我们定义：

1. $\mathsf{PSPACE} = \bigcup_{k\geqslant 1} \mathsf{DSPACE}(n^k)$；
2. $\mathsf{NPSPACE} = \bigcup_{k\geqslant 1} \mathsf{NSPACE}(n^k)$。

但是这个时候，Savitch 定理就告诉我们，$\mathsf{PSPACE} = \mathsf{NPSPACE}$。而由于确定性图灵机的对称性，我们也有 co-$\mathsf{PSPACE} = \mathsf{PSPACE}$。所以，空间复杂度层级中，PSPACE 就成为了几乎唯一最重要的一个。当然，还有一些亚线性复杂度的问题，在此略过不表。

接下来，我们要考虑的就是空间复杂度和时间复杂度之间的关系。很显然，我们有

$$\mathsf{P} \subset \mathsf{NP} \subset \mathsf{PSPACE} \subset \mathsf{EXP}$$

为什么？因为一方面，图灵机在 n 步内之内至多能够经过 n 个纸带格子，所以 $\mathsf{NP} \subset \mathsf{NPSPACE} = \mathsf{PSPACE}$；另一方面，一个图灵机如果只经过 n^k 个格子，那么它的构型（可能的状态）的总数只有 $O(2^{(n^k)})$ 个，而如果它进入了完全相同的构型，那它就会循环（永不停机）。因此，$\mathsf{PSPACE} \subset \mathsf{EXP}$。但是，到目前为止，我们并不知道 PSPACE 和 EXP 之间是否是真包含关系。最后，我们依然以一个例子收尾：

例 10.58 (TQBF 问题) 在全量词（所有变元都是受限变元）的布尔公式中，永真式的集合是 PSPACE 的。

例 10.59 *TQBF 以及 TQBF 博弈与 PSPACE 完全性*

10.5.4 其它有点意思的复杂度类

在 P 和 NP 中间的小复杂度类很多。其中，有一些直接以问题的名字命名，比如我们以后可能会提到一点的 UGC（unique game conjecture），以及 PPAD（polynomial parity arguments on directed graphs），后者在算法博弈论中有一些作用。在这里，我们不会深入介绍这些复杂度类之间复杂的结构，因为这样的定义多到不可胜数。感兴趣的读者可以参照 Complexity Zoo 网站给出的图示。

10.6 归约与 NP 完全性

讨论了这么多抽象的复杂度类之后，我们终归是要回到问题层面上来。我们要讨论的依然是一个问题有多复杂。现在，已经建立的理论告诉我们，可以用一个相对简单的方法来完成描述，也就是将其看成某一个类的成员。而接下来我们要考虑的问题就是：

1. 在某一个类中，不同问题的难度如何比较?
2. 如何更好地确定不同的类之间的包含关系?

这两个问题就是在这一部分我们想要探讨的重点。我们会首先定义 Karp 归约，然后指出在一个复杂度类中，有一些问题是相对来说“最为困难的”，通过对它们的研究，就可以给出复杂度类层级的研究。然后，我们会表明如何建立一个复杂度层级，即对角线方法，并且藉此证明一个有趣的定理：Ladner 定理。最后，我们将通过另一种方式表述归约，并表明对角线法所面临的障碍，同时建立多项式复杂度分层。

10.6.1 Karp 归约

首先，让我们从一个生活中的例子开始。考虑一道数学题 A，如果告诉你它等价于 B，那么你觉得问题 A 难还是问题 B 难？从直观上，我们一般会说问题 A 更难，这是因为我们在做数学题的时候，往往会把难题归约到简单的问题。但再仔细想想，这其实是有问题的：我们不能排除还有一种转化 C 使得问题 A 变成平凡的；而如果问题 B 能解决，那么问题 A 就能解决，因此，问题 A 的难度一定不超过问题 B——这是一个反直觉的事实。另一个更平易近人的例子是，如果想要在 ADS 课上考高分，那么我们可以认真学习，也可以黑了 PTA；但是，认真学习和黑了 PTA 都比考高分难，因为这二者都可

以保证考高分，而考高分不一定真的认真学了，也不一定真把 PTA 黑了。这样，下面的定义就会显得比较自然：

定义 10.60 *称一个语言 A 可被多项式地归约（或 Karp 归约，reduction）到 B，如果存在一个可以在多项式时间内计算的函数 f 使得*

$$x \in A \iff f(x) \in B$$

记作 $A \leq_P B$。

实际上这就是说，当我们用解决问题 B 的方法去解决问题 A 时，可以在多项式时间内完成问题的转化，并且保证给出的答案是正确的（不会错判漏判）。有的时候，这种函数也被称为是一个转译器（transducer）。很显然，归约关系是自反的、传递的。因此，它就自然给出了一种层级：

定义 10.61 *考虑复杂度类 C。如果问题 P 满足 $\forall C \in \mathsf{C}, C \leq_P P$，则称 P 是 C-难的（C-hard）；如果同时 $P \in \mathsf{C}$，则称 P 是 C-完全的（C-complete）。*

结合我们前面对归约中问题难度的讨论，完全问题显然就是一个复杂度类中 “最复杂” 的那些问题。因此，我们不难把 $\mathsf{P} = \mathsf{NP}$ 的问题归约成以下几个步骤：

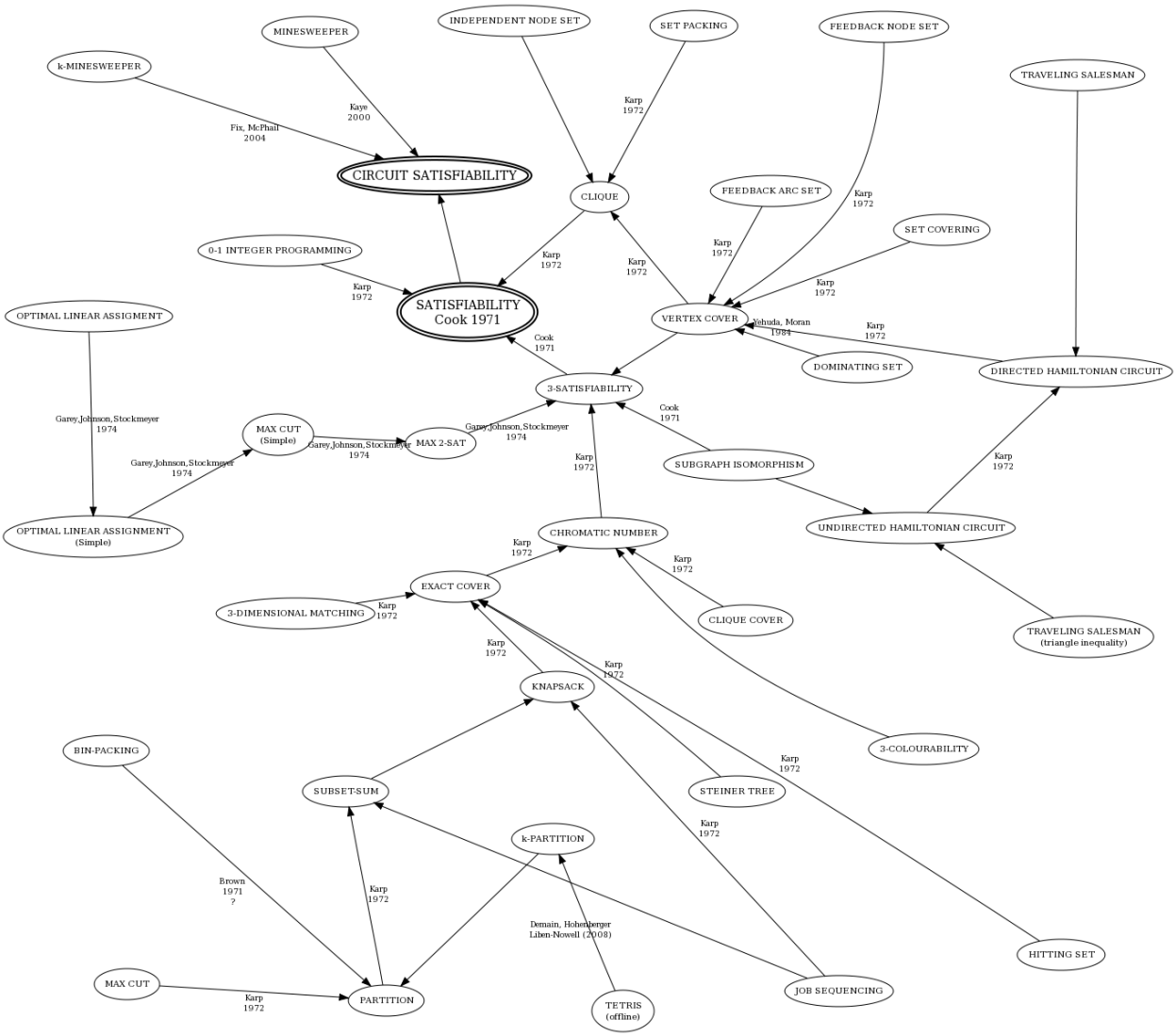
1. 找到一个 NP-完全的问题（我们下一节就会做到）；
2. 证明这个问题是 P 的（很不幸，还没人能做到）。

这样因为所有的 NP 问题都可以在多项式时间内归约到 NP-完全的问题，因此如果 NP-完全的问题可以在多项式时间内解决，那么所有的 NP 问题都可以在多项式时间内解决，那么 NP 就是 P 了。反之，$\mathsf{P} \neq \mathsf{NP}$ 的问题归约成以下几个步骤：

1. 找到一个 NP-完全的问题；
2. 证明这个问题不是 P 的（很不幸，也还没人能做到）。

当然，正如我们一直强调的，归约并不意味着一定要这么来处理这个问题，所以证明 $\mathsf{P} \neq \mathsf{NP}$ 的方式其实也不止一种。同学们闲着没事也可以想想，万一就拿了图灵奖呢（不是）。

有的读者可能会疑惑，本节的标题为 Karp 归约，其中有什么背景呢？事实上，这源于 Richard M. Karp 在 1972 年发表的文章 *Reducibility among Combinatorial Problems*，其中介绍了 21 个 NP 完全问题之间的归约。下面的大图给出了更多的 NP 完全以及 NP 困难问题（回忆上面的定义，NP 困难就是指所有的 NP 问题都能归约到它，但它本身可能不在 NP 中）的归约路径。


MINESWEEPER
INDEPENDENT NODE SET
SET PACKING
FEEDBACK NODE SET
k-MINESWEEPER
TRAVELING SALESMAN
Kaye 2000
Karp 1972
Fix, McPhail 2004
CIRCUIT SATISFIABILITY
CLIQUE
FEEDBACK ARC SET
Karp 1972
0-1 INTEGER PROGRAMMING
Karp 1972
Karp 1972
SET COVERING
Karp 1972
Karp 1972
SATISFIABILITY
Cook 1971
OPTIMAL LINEAR ASSIGMENT
VERTEX COVER
Yehuda, Moran 1984
Karp 1972
Cook 1971
DOMINATING SET
DIRECTED HAMILTONIAN CIRCUIT
3-SATISFIABILITY
Garey,Johnson,Stockmeyer 1974
MAX CUT (Simple)
Garey,Johnson,Stockmeyer 1974
MAX 2-SAT
Garey,Johnson,Stockmeyer 1974
Cook 1971
Garey,Johnson,Stockmeyer 1974
Karp 1972
SUBGRAPH ISOMORPHISM
Karp 1972
OPTIMAL LINEAR ASSIGNMENT (Simple)
CHROMATIC NUMBER
UNDIRECTED HAMILTONIAN CIRCUIT
Karp 1972
Karp 1972
EXACT COVER
3-DIMENSIONAL MATCHING
Karp 1972
CLIQUE COVER
TRAVELING SALESMAN (triangle inequality)
Karp 1972
Karp 1972
KNAPSACK
3-COLOURABILITY
BIN-PACKING
Karp 1972
SUBSET-SUM
STEINER TREE
Karp 1972
k-PARTITION
Brown 1971 ?
Karp 1972
Demain, Hohenberger Liben-Nowell (2008)
HITTING SET
MAX CUT
Karp 1972
PARTITION
TETRIS (offline)
JOB SEQUENCING

10.6.2 完全问题及案例

言归正传，我们首先要想办法找到一个完全问题。实际上，我们已经给出了一个例子：质因数分解（我们将在后面给出证明）。但是，最初的 NP-完全问题实际上并不是这个，而是 Cook-Levin 定理给出的 SAT 问题——它考虑所有 CNF 公式，在其中定义语言：

$$\text{SAT} = \{L \mid L \text{ 可满足}\}$$

实际上这和我们最开始定义的 2-SAT 和 3-SAT 问题类似，只是我们这里不再限制合取范式中每个子句的长度。在清楚问题的定义后，我们就可以开始我们本节的一个核心定理的讲解：

定理 10.62 (Cook 定理) SAT 问题是 NP-完全的。

接下来的很大篇幅都将是这个定理的证明。不感兴趣的读者当然可以跳过这个部分，因为这个部分确实显得有些枯燥。显然，SAT 问题是 NP 的，所以下面我们要表明它是 NP-困难的，即所有 NP 问题都能归约到它：

证明: 下面我们考虑任意一个 NP 中的语言 L。根据定义，存在一个非确定性图灵机能够在 $p(n)$ 的时间内完成对它的判定。接下来，我们要为这个语言 L 中的元素 x 构造一个 CNF，形成一个多项式归约。在此为了简洁起见，仅给出对应的 CNF 的构造而不去解释这为什么是多项式的，因为这应当是自明的。

设这个图灵机在某条计算路径上得到的前 $p(n)$ 个构型是 $C_0, C_1, \cdots, C_{p(n)}$，我们定义以下逻辑谓词，它们都是用来描述这一串状态转移的：

1. $S(q,t)$：构型 C_t 的状态是 q；
2. $H(h,t)$：构型 C_t 的读写头在纸带上的第 h 格；
3. $T(b,h,t)$：构型 C_t 的纸带第 h 格写有符号 b；
4. $I(j,t)$：在 t 到 $t+1$ 的状态转移过程中，完成了操作 j。

这些逻辑谓词都是不那么正式的，其中的 q, h, b, j 实际上都以布尔变量编码。注意，因为我们至多用到 $O(p(n))$ 个格子，所以 h 是有多项式的范围的。在下面的描述中，为了不那么冗长，我们也采用不那么形式化的写法，读者应当可以自己用这些谓词构造出下面描述中对应的 CNF。我们要求它满足：

1. 在时刻 0，状态是起始状态 q_0，此时的纸带只有前 n 格放了输入 x，其他部分都是空白；
2. 在时刻 i，状态唯一、纸带的每一格上的符号唯一、读写头的位置唯一；
3. 在时刻 $p(n)$，状态是接受状态；
4. 在时刻 i，没有被读写头访问的方格 h 上的符号与 $i+1$ 时刻相同；

5. 在时刻 i，仅有一个操作被执行，且它符合有限状态机的转移规则。

事实上这样构造的 CNF 长度为 $O(p(n)^3)$，它是满足我们的要求的。这里的长度主要来源于第二条要求的唯一性的要求。例如我们考虑状态唯一性，设状态集合为 $\{q_1, \cdots, q_k\}$，其中 k 是 $O(p(n))$ 级别的（因为最多只能跑这么多步，故只能有这么多状态），那么我们要求某一时刻的状态唯一，其实就是要求以下 CNF 成立（这显然等价于下面出现的谓词 $S(q_i, t)$ 只能有一个为真）：

$$(S(q_1,t) \vee \cdots \vee S(q_k,t)) \wedge \left(\bigwedge_{1 \leqslant i < j \leqslant k} (\neg S(q_i,t) \vee \neg S(q_j,t)) \right)$$

这个 CNF 的长度是 $O(p(n)^2)$ 的，其它的唯一性要求同理，并且我们一共有 $O(p(n))$ 个时刻都需要满足这样的唯一性，所以这个要求的长度是 $O(p(n)^3)$ 的。

总而言之，根据前面的描述，我们将这一问题根据图灵机构型需要满足的要求转化为了一个 CNF，并且根据上述要求第三条显然有 $x \in L \iff$ CNF 可满足（CNF 可满足等价于存在一系列构型组成的转移路径使得最后一步接受），所以我们就完成了证明。 ■

下一步就是完成 Cook-Levin 定理：

引理 10.63 SAT $\leq_P$ 3-SAT。

证明: 考虑 x 是一个 CNF，对每一个子句 $F = u_1 \vee u_2 \vee \cdots \vee u_m$，我们给出一族：

$$(u_1 \vee u_2 \vee z_1) \wedge (\neg z_1 \vee u_3 \vee z_2) \wedge (\neg z_2 \vee u_4 \vee z_3) \wedge \cdots \wedge (\neg z_{m-3} \vee u_{m-1} \vee u_m)$$

其中 z_i 是新引入的变量，我们不难验证这样的拆分是合理的。如果 $m < 3$，新引入变量即可，在此略过。这个族是可满足的当且仅当 F 是可满足的。 ■

结合上述定理和引理，我们有 Cook-Levin 定理：

定理 10.64 (Cook-Levin 定理) 3-SAT 是 NP-完全的。

证明: 首先显然 3-SAT 是 NP 的。然后，由于 SAT $\leq_P$ 3-SAT，并且因为 SAT 是 NP-完全的，所以所有的 NP 问题都能归约到 SAT，根据归约的传递性可知所有的 NP 问题都能归约到 3-SAT，所以 3-SAT 是 NP-完全的。 ■

因此这一定理也给出了一种证明 NP-完全性的方法：只要证明一个问题是 NP 的，然后证明它可以从一个已知的 NP-完全问题归约而来即可。现在，让我们重新看一眼这个让人头晕的证明。注意到：

1. 它给出的转化 $x \in L \iff f(x) \in$ SAT 实际上给出了从 x 的证明到 $f(x)$ 的证明的一个单射，因为 x 的证明作为非确定性图灵机的一个分支在证明中被给出了。这种归约被称为 Levin 归约；
2. 实际上，它给出的是所有 x 的可能证明到所有 $f(x)$ 的可能证明的集合的双射，这表明它们大小相同。这种规约被称为约俭归约（parsimonious reduction）。

许多类似的 NP-完全问题都有从任意语言 $L \in \mathsf{NP}$ 出发的约俭 Levin 归约，这在有的时候是一个非常有用的事实。下面就是一个有意思的例子：

推论 10.65 *如果 $\mathsf{P} = \mathsf{NP}$，那么对于任意 $L \in \mathsf{NP}$，存在一个图灵机能够在多项式时间内计算出任意一个字符串 $x \in L$ 的证明。*

证明: 因为 L 到 SAT 的归约是约俭 Levin 归约，所以我们只要表明对于 SAT 这个命题成立即可。为此，我们考虑一个确定性图灵机 M 在多项式时间内给出任意 CNF 的可满足性，这是 $\mathsf{P} = \mathsf{NP}$ 的直接推论，然后我们将利用它来构造一个能够搜索出证明的图灵机：首先，我们用 M 来检查输入 φ 是否是可满足的，如果不是，那当然没有证明；如果是，那考虑固定第一个变量，判断剩下的是否是可满足的，以此类推。 ■

在接下来关于 NP 完全的讨论中，我们将看到的问题基本都是判定问题，但显然不是所有问题都是判定问题，例如我们讲义前面提到的搜索问题。对于一个搜索问题（最优化、排序事实上都可以视为搜索一个解的问题），我们可以将其转化为对应的判定问题。根据上述引理，如果我们要搜索某个 NP 语言中的成员（这是搜索问题），我们可以写下对应的判定问题。然后我们可以把这个 NP 语言归约到 SAT，这个归约是 Levin 归约，因此 SAT 的一个证明可以转化为该判定问题的一个证明，然后这个证明可以在多项式时间内转化为搜索问题的一个解。

举个例子，我们考虑团问题，我们有一个 NP 语言

$$\{(G,k) \mid G\text{有一个大小为}k\text{的团}\}$$

我们有一个搜索问题：找到一个大小为 k 的团，对应的判定问题为 G 是否存在一个大小为 k 的团。我们可以将这个判定问题归约到 SAT，然后根据 Levin 归约的性质，SAT 的证明可以转化为团判定问题的证明，然后这个证明就是一个大小为 k 的团。

除此之外，这个证明事实上表明 SAT 是向下自归约的（downward self-reducible）。给定一个解决长度小于 n 的算法，我们就能给出解决长度 n 的算法。实际上，Levin 归约的性质使得所有 NP-完全问题都有类似的特性，但由于其形式化相对麻烦，在此不做过多的解释。

另外，我们上面对 co-NP 和 PSPACE 给出的两个例子也都是完备的。接下来我们给出一些比较经典的归约的例子供读者参考，体会归约的一些基本方法。

首先是 PPT 上的例子：

例 10.66 *已知哈密顿回路问题是 NP-完全的，现在我们要证明旅行商问题（TSP）也是 NP-完全的。*

- *哈密顿回路问题：给定一个图 $G = (V, E)$，判断是否存在一个哈密顿回路，即一个遍历所有节点恰好一次的回路；*
- *旅行商问题：给定一个完全图 $G = (V, E)$ 和一个整数 k，判断是否存在一个哈密顿回路，使得其长度不超过 k。*

证明:

1. 第一步：证明 TSP 是 NP 的。这是显然的，因为我们如果给了一个哈密顿回路，我们可以在多项式时间内验证它的长度是否不超过 k。

2. 第二步：证明哈密顿回路问题可以归约到 TSP，也就是说我们已经有一个可以解决 TSP 的万能方法，现在要解决哈密顿回路问题，我们要想办法把解决哈密顿回路问题转化为解决 TSP 问题，让转化而来的 TSP 返回给我的答案就是我想知道的哈密顿回路问题的答案——也就是说，存在哈密顿回路等价于转化后的旅行商问题的答案是 “是”。

   如何归约呢？其实是比较平凡的。当我们收到一个哈密顿回路的输入 $G = (V, E)$，为了转化为 TSP 问题，我们需要构造一个带权完全图 $G' = (V, E')$，然后我们把 G 中的边权重都设为 1，而原先不在 G 中的边（因为 G 不一定是完全图）权重设为 2。这样归约后答案是否一致呢？事实上也是显然的：

   (a) 如果 G' 中存在一个哈密顿回路的长度不超过 $|V|$，这意味着存在一个边权全为 1 的哈密顿回路，而边权为 1 表明这条边原先就在 G 中，因此这就意味着 G 中存在一个哈密顿回路；

   (b) 如果 G' 不存在长度不超过 $|V|$ 的哈密顿回路，那么 G 中必定也不存在哈密顿回路，否则若存在哈密顿回路，那么 G' 中也存在哈密顿回路只经过边权为 1 的边，故长度不超过 $|V|$，与我们的假设 G' 不存在长度不超过 $|V|$ 的哈密顿回路矛盾，故 G 中必定也不存在哈密顿回路。

   因此，通过上面的归约，我们把一个图 G 的哈密顿回路问题转化为了图 G' 在 $k = |V|$ 的情况下的 TSP 问题，并且只要 TSP 返回是，那就的确有哈密顿回路，返回否就的确没有。并且归约的过程显然也是多项式时间的，因为只需把图赋权重即可，这样我们就完成了归约。

■

PPT 的最后还给出了一个团问题和顶点覆盖问题的归约，读者可以自行阅读 PPT，应当是容易理解的。下面我们来看一个归约，尽管不是 NPC 问题，但是给出了一种 SAT 问题和图问题之间的归约的经典方法。

例 10.67 2-SAT 问题是多项式时间可解的。

证明: 我们显然需要将 2-SAT 问题转化为多项式时间可解的图问题。为了转化为图，我们假设全部出现的逻辑变量为 $x_1, \cdots, x_n$，于是每个变量和它的反一共有 $2n$ 个变量，将它们作为图的 $2n$ 个顶点即可。我们心里有一个声音，就是这些点之间边的连接情况应该就和可满足性有些联系。如何表达出来呢？事实上我们只需要利用一个基本的逻辑等价式：

$$x_1 \vee x_2 \iff (\neg x_1 \to x_2) \wedge (\neg x_2 \to x_1)$$

于是我们可以将 2-SAT 的所有 $x_i \vee x_j$ 字句转化为图中的两条边，即 $(\neg x_i \to x_j)$ 和 $(\neg x_j \to x_i)$（其它形式的子句类似）。这样我们就把图的顶点和边都构造好了，然后我们只需要用多项式时间找强联通分

量即可，因为我们表达式可满足当且仅当图中强连通分量不存在矛盾，即不会有一个变量和它的反在同一个强连通分量中：

1. 如果对于某个 i 有 x_i 和 $\neg x_i$ 在同一个强连通分量中，则意味着有一条路径从 x_i 到 $\neg x_i$，也有一条路径从 $\neg x_i$ 到 x_i，根据逻辑蕴含的传递性可知，$x_i \to \neg x_i$ 和 $\neg x_i \to x_i$ 都成立，即我们有 $\neg x_i \vee \neg x_i$ 且 $x_i \vee x_i$，这显然是不可能的；

2. 如果对于所有的 i，x_i 和 $\neg x_i$ 都不在同一个强连通分量中，那么我们直接尝试赋值即可，并且一定是可以实现的，因为不会出现矛盾的赋值。

所以我们用多项式时间把可满足性问题转化为了图的强连通分量问题，并且强联通分量也有多项式时间算法，这样我们就完成了证明。 ■

10.6.3 对角线方法和 Ladner 定理

接下来，我们要考虑怎么充分地建立一种复杂度分层。对角线方法（diagonalization）在这里是重要的——当然，在这里，我们只能给出一些粗浅的认识，更深层次的形式化需要的抽象超越了在此可以讨论的范围。首先，我们定义：

定义 10.68 *称 f 是一个时间可构造函数（time-constructible function），如果存在一个图灵机，对于任意输入 n，可以在 $O(f(n))$ 的时间内输出 $f(n)$ 个 1。*

这个定义粗看似乎有点莫名其妙。直观上看，最明显的反例是不可计算的函数，它当然不是时间可构造的，而可计算的反例在现在我们还给不出来。粗略地理解的话，这个定义想要保证的是有一台图灵机可以对这个 $f(n)$ 进行计时。在下面的定理证明过程中，我们会注意到它的重要性：

定理 10.69 (Time Hierachy Theorem, THT, Seiferas-Fischer-Meyer, 1978, Žák, 1983) *设 $f(n)$ 和 $g(n)$ 都是时间可构造的，如果 $f(n) = o(g(n))$，则 $\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}(g(n)\log g(n))$。*

证明： 为了证明这个定理，我们需要考虑构造一个图灵机，然后证明它不在 $\mathsf{DTIME}(f(n))$ 中，但是在 $\mathsf{DTIME}(g(n)\log g(n))$ 中。前者，我们需要采用对角线方法，而后者只需要完成一个模拟。定义函数：

$$p(\alpha) = \begin{cases} 1 - M_\alpha(\alpha) & \text{如果模拟 } M_\alpha \text{ 的通用图灵机在 } g(|\alpha|)\log g(|\alpha|) \text{ 步以内停机} \\ 0 & \text{否则} \end{cases}$$

一方面，p 在 $O(g(n)\log g(n))$ 的时间内是可计算的。这是因为我们直接用一台通用图灵机模拟 M_α 的执行即可，其花费的开销正好是这个值，这样就完成了这个模拟。另一方面，假定 p 在 $O(f(n))$ 中可以计

算，花费的时间是 $c_1 f(n)$，用通用图灵机模拟它执行的开销是 $c_2 c_1 f(n) \log f(n)$，而由于 $f(n) = o(g(n))$，总存在一个整数 n_0 使得对于任意 $n \geqslant n_0$ 都有

$$g(n) \log g(n) > c_2 c_1 f(n) \log f(n).$$

取一个 $|\beta| > n_0$ 长度的能计算函数 p 的图灵机 M_β，即 $M_\beta(\alpha) = p(\alpha)$，它花费的时间是 $c_1 f(n)$，则根据上面的不等式，模拟 M_β 的通用图灵机能在 $g(|\beta|) \log g(|\beta|)$ 的时间内停机，这是由我们的假定得出的，因此我们用两种方式计算 $p(\beta)$，根据 p 的定义，我们有 $p(\beta) = 1 - M_\beta(\beta)$，但是因为 M_β 的定义，$p(\beta) = M_\beta(\beta) \neq 1 - M_\beta(\beta)$。这就产生了矛盾，因此 p 不可能在 $O(f(n))$ 的时间内计算，即 $p \notin \mathsf{DTIME}(f(n))$，但是 $p \in \mathsf{DTIME}(g(n) \log g(n))$，因此 $\mathsf{DTIME}(f(n)) \subsetneq \mathsf{DTIME}(g(n) \log g(n))$。 ■

这个定理有一个直接的推论，即 $\mathsf{P} \subsetneq \mathsf{EXP}$。因此此前我们研究的包含关系链尽管中间的包含是否是真包含未知，但是 P 和 EXP 之间的关系是确定的。这也从某种层面上给出了为什么我们认为多项式时间算法是高效的算法的原因。

为了表明可构造性的意义，我们陈述（但不证明）以下定理：

定理 10.70 (Borodin–Trakhtenbrot) *对于任意一个可计算函数 $g : \mathbb{N} \to \mathbb{N}$ 满足 $g(n) \geqslant n$，存在一个函数 $f : \mathbb{N} \to \mathbb{N}$ 使得 $\mathsf{DTIME}(f(n)) = \mathsf{DTIME}(g(f(n)))$。*

显然，取 g 是时间可构造的，如果 f 也是时间可构造的，那么它和 THT 是矛盾的。对于非确定性图灵机的版本，我们有：

定理 10.71 *设 $f(n)$ 和 $g(n)$ 都是时间可构造的，则 $\mathsf{NTIME}(f(n)) \subsetneq \mathsf{NTIME}(g(n))$，如果 $f(n+1) = o(g(n))$。*

对于空间复杂度的版本，我们有类似的空间可构造和以下定理：

定理 10.72 *设 $f(n)$ 和 $g(n)$ 都是空间可构造的，则 $\mathsf{DSPACE}(f(n)) \subsetneq \mathsf{DSPACE}(g(n))$，如果 $f(n) = o(g(n))$。*

这几个结果的证明是类似的，我们在此略过。现在以一个对角线方法不太寻常的应用收尾：

定理 10.73 (Ladner) *如果 $\mathsf{P} \neq \mathsf{NP}$，则一定存在一个语言 $L \in \mathsf{NP} - \mathsf{P}$，且不是 NP-完全的。*

证明: 很显然，如果 $\mathsf{P} \neq \mathsf{NP}$，那么 NP-完全的语言一定在 $\mathsf{NP} - \mathsf{P}$ 中，这个定理表明，其中事实上不只有 NP 完全的问题。所以，我们希望从 SAT 出发去构造一个不是 NP-完全的语言。首先定义：

$$\mathrm{SAT}_H = \{x 1^{n^{H(n)}} \mid x \in \mathrm{SAT}, n = |x|\}$$

这个语言类看起来很寻常，它无非就是对 SAT 做了一点加边的操作。当 H 有界时，它显然也是 NP-完全的，因为它的加边无非是多项式的。而当 H 无界时，我们声称它不是 NP-完全的：

如果存在一个从 SAT 到 SAT_H 的多项式归约 f，它的时间复杂度是 $O(n^i)$，那么它一定是把一个长度 n 的表达式映到一个长度 $O(n^i)$ 的表达式 $x1^{|x|^{H(|x|)}}$，他的长度是 $|x|+|x|^{H(|x|)}$。因此 $|x|=o(n)$，形象地说，我们就将长度为 n 的表达式在多项式时间内越归约越短了，这样的话最多 n 次归约就能完成对这个表达式的判定任务，因为长度已经变成 1 了，这意味着 SAT 是 P 的，与 $\mathsf{P}\neq\mathsf{NP}$ 的假设矛盾。

因此，我们要做的就是构造函数 H，使得它在无界（保证不是 NP-完全的）的同时增长不是太快（保证不是 P）的。事实上，我们取 $H(n)$ 为最小的、小于 $\log\log n$ 的整数 i，使得对于任意长度不超过 $\log n$ 的字符串，指标 i 的图灵机 M_i 都会在 $i|x|^i$ 步之内停机，而且 $M_i=1 \iff x\in\mathrm{SAT}_H$，也就是说 i 是判定 $\mathrm{SAT}_H\cap\{x\mid |x|\leqslant\log n\}$ 的图灵机的指标集中的元素。如果这样的 i 不存在，那么取 $H(n)=\log\log n$。

接下来，我们表明 $\mathrm{SAT}_H\notin\mathsf{P}$。如果它属于 P，那么存在一个图灵机 M 在至多 cn^c 步中解决这个问题。这意味着存在一个整数 $i>c$ 使得 $M=M_i$。由 H 的定义，对于任意 $n>2^{2^i}$，都有 $H(n)\leqslant i$，则 $n^{H(n)}$ 有多项式界，这意味着对于足够长的输入（因为我们要求 $i>c$ 且 $n>2^{2^i}$），SAT_H 就是 SAT 加上多项式长度的边，所以这就表明 SAT 是 P 的。

为了表明它不是 NP-完全的，我们只要表明 H 在 n 趋于无穷时趋于无穷。也就是说，对于任意整数 i，只有有限个 n 使得 $H(n)=i$。因为 $\mathrm{SAT}_H\notin\mathsf{P}$，对于任意 i，存在一个输入 x 使得给定 $i|x|^i$ 的时间，M_i 不能给出正确的结果。故结合 H 的定义，对于任意满足 $\log n>|x|$ 的整数 n，$H(n)\neq i$，故只有有限个 n 使得 $H(n)=i$。 ■

10.7 NP 困难

我们之前已经提到了 NP-困难问题，它是一个非常重要的概念。很遗憾，限于时间与篇幅我们无法在这里展开太多，但非常建议有兴趣的同学去阅读《Algorithm Design》的第 10 章，其中介绍了一些处理 NP 困难问题的方法。在这里，我们只给出一个例子：

例 10.74 回顾 0-1 背包问题：给定 n 个物品，每个物品有一个重量 s_i 和一个价值 v_i，以及一个背包容量 C，问如何选择物品放入背包使得总价值最大，但总重量不超过 C。

1. 动态规划一讲中给出了一个算法，请问那个算法是多项式时间的吗？

2. 如果每个物品的重量都小于等于 n^2，请问 0-1 背包问题是否存在多项式时间算法；

3. 0-1 背包问题判定版本：给定 n 个物品，每个物品有一个重量 s_i 和一个价值 v_i，以及一个背包容量 C 和一个价值 V，问是否存在一种选择使得总价值不小于 V 且总重量不超过 C。证明：0-1 背包问题判定问题是 NP 完全的（提示：可以利用互联网搜索任意可能的归约）。

10.8 致谢

在此我需要非常感谢刘泓健同学为本章讲义的绝大部分内容提供的基本框架。我们真诚地希望这几章的讲义的作用不仅仅是让读者理解课程需要掌握的内容，更重要的是在这片理论计算机科学尚不繁荣的土地上——毕竟在这里笔者感兴趣的算法博弈论都能算得上很理论的——种下一颗种子，让读者去探索更多相关的知识。也由衷感谢 Prof. Arora 和 Prof. Barak 的 *Computational Complexity: A Modern Approach* 一书，本章的许多内容都参考了这本书。

作为世界一流大学的世界一流专业，在计算机科学与技术学院没有复杂度理论的专题课程是非常令人惊讶的。这个领域本应该是计算机科学的重要组成部分，其中也反映了许多计算机科学的基本思想和方法，诞生了许多图灵奖得主（包括图灵本人、Cook 和 Karp 等我们在本节中见到的名字）。但是，我们遗憾地看到，计算机科学与技术的学生对科学一无所知，对技术也知之甚少。我们希望，这份讲义（尤其是其中的对角线方法）能够让大家体会到计算机科学本身的精妙之处，并且能够更多地了解相关的内容，比方说可计算性理论、数理逻辑等等。在这一讲中，我们尚未能明确给出数理逻辑和复杂度理论之间的直接关联，虽然我们通过几个例子将其展现了一部分。有兴趣的读者可以寻找一些多项式层级、描述复杂度等关键词的文献，它更明显地表达了复杂度和数理逻辑之间的联系，对于自动定理证明、理论求解等领域也有诸多应用。此外，复杂度理论尚有一种更加现代化、形式化的表述方式，即所谓复杂度测度。一些物理层面的复杂度，比如 Kolgomorov 复杂度等也能在某种意义上得到统一，并且得到一些热动力学的结果。此外，谈到数理逻辑，值得一提的就是与之并称的类型论和范畴论。也有一套基于范畴逻辑的所谓综合复杂度理论（synthetic complexity theory）正在形成，虽然它远未能称得上成型。但是，这种未定的形态更适合那些有志于计算机科学的研究者，毕竟，这种仿佛来自虚空的和谐之美，是我们不变的追求。