

算法设计与分析

Computer Algorithm Design & Analysis
2019.11

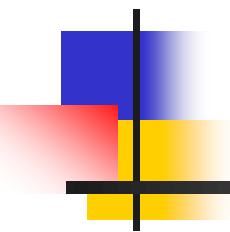
王多强

dqwang@mail.hust.edu.cn

群名称: 2019-算法
群 号: 835135560



群名称: 2019-算法
群 号: 835135560



Chapter 34

NP-Completeness

NP 完全性

易解问题和难解问题

- 如果一个问题 Π 存在一个运行时间为 $O(n^k)$ 的算法，其中 k 为常数，则称问题 Π 存在多项式时间算法。
- 通常将存在多项式时间算法的问题看做是易解问题，而将需要超多项式时间算法的问题看做是难解问题。
 - 注：难解问题并不一定是不可解问题，可解但时间复杂度高(超多项式时间)的问题也是难解问题。

多项式时间的性质

- 多项式时间复杂性具有**闭包性**
 - 多项式函数通过算术运算的组合或重叠，得到的函数仍然是多项式阶的。
- 多项式时间复杂性具有**独立性**
 - 在某一计算模型下有多项式时间，在其它模型下也一定具有多项式时间，如RAM模型、Turing模型。

判定性问题(Decision-problem)

- ◆ **判定性问题**：仅要求回答Yes或No的问题。

如：17422是素数吗？

- ◆ **判定性问题也有易解的和难解的**

如：

- 17422可以写成两个质数的和吗？ **困难的问题**
 - 7能表示成两个质数之和：2+5。是吗？ **简单的问题**
 - 任一大于2的偶数都可写成两个质数之和，是吗？ **难解问题**
- 易解问题
- 难解问题
- **功能性问题**：一般由一个函数 f 给出，目标是计算 f 在某种定义下的值。
- 如：最优化问题，求目标函数的极值。

P类问题和NP类问题

- ◆ P类问题和NP类问题的形式化定义来源于**形式语言**的识别问题。
 - **语言识别**：某个句子是否能够为某种形式语言的**自动机**识别，能则输出Yes，否则输出No；
 - 语言识别问题是一种特殊的**判定问题**，基于某种计算模型给出，如图灵机模型、RAM模型。而判定问题可以容易地表述为语言的识别问题，从而方便在某种计算模型上求解。
- ◆ 这里从算法的角度对P类问题和NP类问题给出一种非形式化的解释。

一些判定性问题

- **图灵停机问题**： halting problem, 判断任意一个程序是否会在有限的时间之内结束运行并停机。是一个不可解的问题。
- **哈密顿回路问题**： 在一个图 $G=(V,E)$ 中, 求从某个顶点出发, 经过所有顶点一次且仅一次, 且回到出发点的回路称为**哈密顿回路问题**。
 - 确定图 G 中是否有一个哈密顿回路的问题是一个判定问题。
- **TSP问题**： 旅行商问题/货郎担问题, 在一个带权图 $G=(V,E)$ 中, 求经过所有顶点一次且仅一次, 再回到出发点, 且路径长度最短的回路 —— TSP是最优化问题, 非判定性问题。
 - 对给定的带权图 G 和一个正整数 k , 确定是否有一个经过所有顶点一次且仅一次再回到出发点的回路, 其**总距离小于 k** 的问题是一个判定问题。

判定问题的一个重要特征

相对原问题，判定一个**待定解**是否是该问题的一个解相对简单。

如：

- 求解哈密顿回路是个难解问题，但**验证一个给定的顶点序列是不是哈密顿回路**却是很容易的。
 - 对TSP问题有类似结论。
- 求解一个线性方程组可能是困难的，但**验证一组解是不是方程组的解**却很容易。

判定问题与最优化问题

最优化问题：该类问题的每一种可能的解都有一个相关的值，
我们的目标是找出其中具有**最优值**的可行解。

- 如，**SHORTEST-PATH问题**：已知一个无向图 G 及顶点 u 和 v ，求 u 到 v 之间经过最少边的一条路径。

■ NP完全性适合于判定问题，不直接适合于最优化问题。

- 但最优化问题和判定问题之间有着方便的联系：
 - ◆ 通过对待优化的问题**强加一个界**，就可以把一个给定的最优化问题转化为一个相关的判定问题。

如，SHORTEST-PATH问题，其相关的判定问题 $PATH_k$ 可以描述为：
对给定的有向图和顶点 u 和 v 以及一个整数 k ，判定在 u 和 v 之间是否存在一条至多包含 k 条边的路径。

因为判定问题相对“更容易一些”，因此，我们证明最优化问题是一个“困难的问题”时，可以利用这种“难-易”关系来说明：

如果我们能提供足够的证据表明某个判定问题是个困难问题，就等于提供了证据表明与其相关的最优化问题也是困难的

——因为判定问题“更容易一些”。

- 注：引入判定问题是为了说明最优化问题是困难的，而不是为了说明判定问题可解就一定可以解最优化问题。

确定性算法和P类问题

定义1 **确定性算法** (deterministic algorithm)

设A是求解问题 Π 的一个算法，如果在算法的整个执行过程中，**每一步只有一个确定的选择**，则称算法A是**确定性算法**。

如：计算 $2+3$ 的值。

定义2 **P类问题** (Polynomial Problem)

如果对于某个**判定问题** Π ，存在一个非负整数 k ，对于输入规模为 n 的实例，能够以 **$O(n^k)$** 的时间运行一个**确定性算法**得到 (yes或no) 的答案，则问题 Π 是一个**P类问题**。

◆ P类问题可以理解为具有**多项时间复杂度的确定性判定问题**。

非确定性算法和NP类问题

定义3 **非确定性算法** (Non-deterministic algorithm)

设A是求解问题 Π 的一个算法，如果算法A以如下**猜测并验证**的方式工作，就称算法A是一个**非确定性算法**。

(1)**猜测阶段**：在这个阶段，对问题的输入实例，猜测一个解 y （待验证的解，证书），而且猜测是**以一种非确定性的方式工作**，故在算法的每一次运行时， y 的值可能不同。

(2)**验证阶段**：在这个阶段，用一个**确定性算法**验证两件事：

首先，检查在猜测阶段产生的 y 是否具有合适（合法）的形式，如果不是，则算法停下来并得到no；

否则， y 是合适的形式，那么验证它是否是问题的解，如果是问题的解，则算法停下来并得到yes。否则，算法停下来并得到no。

例，TSP问题的判定形式：

假定算法A是求解TSP判定问题的非确定性算法。


首先，算法A以非确定性的形式猜测一个路径是TSP判定问题的解——待验证的解。

然后，用确定性算法检查这个路径是否经过所有结点一次且仅一次并返回出发点。

- 如果答案为yes，则继续验证这个回路的总长度是否 $\leq k$ ；
 - 如果答案仍是yes，则算法A输出yes，否则输出no。
- 但如果算法A输出no，并不意味着不存在一个满足要求的回路，因为算法的猜测可能是不正确的；

可见，算法输出yes，当且仅当对于TSP判定问题的某个输入实例至少存在一条满足要求的回路。

- 试想，如果总能猜中，那也很“神”啊。

- 
- ◆ **非确定性算法**不是一个可以实际运行的算法，引入非确定性算法的目的在于给出NP类问题的定义，从而将验证过程为多项式时间的问题归为一类进行研究——NP类问题。

定义4 **NP类问题**

如果对于某个判定问题 Π ，存在一个非负整数 k ，对于输入规模为 n 的实例，能够以 $O(n^k)$ 的时间运行一个非确定性算法，得到yes或no的答案，则该判定问题是一个**NP类问题**（Non-deterministic Polynomial, **多项式复杂程度的非确定性问题**）。

要点：对于NP类判定问题，必须存在一个确定性算法，能够以多项式时间来检查和验证在非确定的猜测阶段所产生的解是否是问题的一个正确答案。

因此，NP问题不能简单理解为“非多项式时间问题”，而是“具有多项式时间非确定性算法的问题”，因此NP类问题也仅是难解问题的一个子类。

例：并不是任何一个在常规计算机上需要指数时间的问题都是NP类问题。如：汉诺塔问题

汉诺塔的传说：在印度，有一个古老的传说：在世界中心**贝拿勒斯**（在印度北部）的圣庙里，有一块黄铜板上插着三根宝石针。印度教的主神**梵天**在创造世界的时候，在其中一根针上从下到上地穿好了由大到小的64片金片，这就是所谓的**汉诺塔**。

不论白天黑夜，总有一个僧侣按照下面的法则移动这些金片：

- 一次只移动一片
- 不管在哪根针上，小片必须在大片上面。

僧侣们预言，当所有的金片都从梵天穿好的那根针上移到另外一根针上时，世界就将在一声霹雳中消灭，而梵塔、庙宇和终生也都将同归于尽。

不管这个传说的可信度有多大，如果考虑一下把64片金片从一根针移到另外一根针上，并且始终保持**上小下大**的顺序，这需要多少次移动呢？

用数学分析的方法：假设对n片金片，移动次数是 **$f(n)$**

则，显然 $f(1)=1$ ， $f(2)=3$ ， $f(3)=7$ ，

可以证明， $f(k+1)=2*f(k)+1$ 。

则有： **$f(n)=2^n-1$** 。

则，当 $n=64$ 时， $f(64)=2^{64}-1=18446744073709551615$ 。

设想，即使僧侣搬金片每秒钟可以移一次，则要需5800多亿年才能将64片金片从梵天穿好的那根针上移到另外一根针上。——宇宙可能真的毁灭了。


汉诺塔问题不是NP类问题。Why?

对于任意给定的 n （代表层数），正确的移动方案有 2^n 个**移动步**。这样，“汉诺塔验证”需要对 $O(2^n)$ 步进行验证——不是一个可以在多项式时间内完成的问题。

因此对汉诺塔问题，**不能在多项式时间内猜测并验证一个答案**，所以汉诺塔问题不是NP类问题。

尽管NP类问题是对于**判定问题**定义的，但理论已证明，对那些**可以在多项式时间应用非确定性算法解决**的所有问题都属于NP类问题。

- P类问题和NP类问题的区别在于：
 - P类问题可以用**多项式时间的确定性算法**进行判定或求解。
 - 而NP类问题则是用**多项式时间的非确定性算法**进行判定或求解。



- ◆ 一般，P类问题的算法具有一个计算公式，将输入代入公式，就可以算出结果，如 $x+y$ 。

- ◆ NP类问题没有明确的计算公式可以求解。

比如没有一个公式可以计算出最大的质数是多少或者能够求解哈密顿回路问题。

哈密顿回路问题

- 难解问题：图G中是否有一条哈密顿回路？

已知一个问题实例，一种可能的判定算法是罗列出G的顶点的所有排列，然后对每一种排列进行检查，以确定它是否是一条哈密顿回路。

设顶点数为 n ，则总共有 $n!$ 种可能的顶点排列，因此算法的运行时间为指数形式，不是多项式的。

易解问题：哈密顿回路判定问题

假设有人说某个给定的图 G 是哈密顿图，并给出一个代表回路的顶点序列，说这是该图的一条哈密顿回路。

证明该顶点序列是不是哈密顿回路是容易的：

仅需要检查所提供的回路是否是 V 中顶点的一个排列，以及沿回路的每条连续的边是否在图中存在。这样就可以验证所提供的回路是否是哈密顿回路。

- 该验证算法可以在 $O(n^2)$ 的时间内实现，因此可以在多项式时间内验证一个结点序列是不是图的一条哈密顿回路。

P类问题与NP类问题的关系 **P=NP?**

- 如果问题 Π 属于P类问题，则存在一个多项式时间的确定性算法来对它进行判定或求解。显然也可以构造一个多项式时间的非确定性算法，来验证其正确性。因此，问题 Π **也属于NP类问题**，即 $P \subseteq NP$ 。
- 而如果问题 Π 属于NP类问题，则存在一个多项式时间的非确定性算法，来猜测并验证它的解。但是不一定能构造一个多项式时间的确定性算法对它进行求解或判定。因此，人们猜测 **$P \neq NP$** ，但至今无法证明。

问题变换与计算复杂性归约 (reduce)

定义5 假设问题 Π' 存在一个算法 A ，对于问题 Π' 的输入实例 I' ，算法 A 可以得到一个输出 O' ；另外问题 Π 的输入实例是 I ，对应于输入 I ，问题 Π 有一个输出 O 。

则问题 Π 变换到 Π' 是一个3个步骤的过程：

- **输入转换**：把问题 Π 的输入 I 转换为问题 Π' 的适当输入 I' ；
- **问题求解**：对问题 Π' ，应用算法 A 对输入实例 I' 产生一个输出 O' ；
- **输出转换**：把问题 Π' 的输出 O' 转换为问题 Π 对应输入实例 I 的正确输出 O 。

- 若在 $O(\tau(n))$ 的时间内完成上述输入和输出的转换，则称问题 Π 以 $\tau(n)$ 时间变换（**归约**）到问题 Π' ，记为 $\Pi \propto_{\tau(n)} \Pi'$ ，其中 n 为问题规模；
- 若能以**多项式时间**完成上述转换，则称问题 Π 以**多项式时间**转换到问题 Π' ，记为 $\Pi \propto_p \Pi'$ 。

例：设算法A可以求解排序问题。输入 I' 是一组整数 $X=x_1, x_2, \dots, x_n$ ，输出 O' 是这组整数的一个排列，使得 $x_{i1} \leq x_{i2} \leq \dots \leq x_{in}$ 。

下面考虑配对问题：输入 I 是两组整数 $X=x_1, x_2, \dots, x_n$ 和 $Y=y_1, y_2, \dots, y_n$ 。输出 O 是两组整数的元素配对，即 X 中的最小值与 Y 中的最小值配对，次小值与次小值配对，依次类推。

解决配对问题的一种办法是使用已存在的排序算法A：首先将这两组整数排序，然后根据它们的位置进行配对。所以，经过下述3步，可以将配对问题转换为排序问题：

- (1) 把配对问题的输入I转化为排序问题的两个输入 I_1' 和 I_2' ；
- (2) 应用算法A对两个输入 I_1' 和 I_2' 分别排序得到两个有序序列 O_1' 和 O_2' ；
- (3) 把排序问题的输出 O_1' 和 O_2' 转化为配对问题的输出O，这可以通过配对两组整数的第一个元素、第二个元素、...来得到。

配对问题到排序问题的转换有助于建立配对问题代价的一个上限：上述输入和输出的转换都是多项式时间内完成的，所以如果排序问题有多项式时间的算法，则配对问题也一定有多项式时间的算法（闭包性）。

需要强调的是：问题变换的主要目的不是给出解决问题的算法，而是给出通过另一个问题来**理解当前问题计算时间上下限**的一种方式。

定理2 若已知问题 Π 的计算时间下限是 $T(n)$ 且问题 Π 可 $\tau(n)$ 变换到问题 Π' ，即 $\Pi \propto_{\tau(n)} \Pi'$ ，则 $T(n) - O(\tau(n))$ 为问题 Π' 的一个计算时间下限。

定理3 若已知问题 Π' 的时间上限是 $T(n)$ ，且问题 Π 可 $\tau(n)$ 变换到问题 Π' ，即 $\Pi \propto_{\tau(n)} \Pi'$ ，则 $T(n) + O(\tau(n))$ 为问题 Π 的一个计算时间上限。

定理4 设 Π 、 Π' 和 Π'' 是3个判定问题，若 $\Pi \propto_p \Pi'$ ，且 $\Pi' \propto_p \Pi''$ ，则 $\Pi \propto_p \Pi''$ 。

证明：

设A是一个对于某个多项式p，在 $p(n)$ 步内实现 $\Pi \propto_p \Pi'$ 的算法；

设B是一个对于某个多项式q，在 $q(n)$ 步内实现 $\Pi' \propto_q \Pi''$ 的算法。

设x是算法A的一个规模为n的输入。

显然，算法A以输入x运行（转换）时，其输出的大小不会超过 $cp(n)$ 。如果算法B接受规模大小为 $cp(n)$ 的输入时，则存在某个多项式r，它的运行（转换）时间 $O(q(cp(n)))=O(r(n))$ ，由此，从问题 Π 到问题 Π'' 是一个多项式变换

证毕。

一般来说，问题变换不是一个可逆的过程，如果问题 Π 和问题 Π' 可相互转换，即 $\Pi \propto_{r(n)} \Pi'$ 且 $\Pi' \propto_{r(n)} \Pi$ ，则称问题 Π 和问题 Π' 是 $r(n)$ 时间等价的。

NP完全问题 (NPC, Non-deterministic Polynomial complete problem)

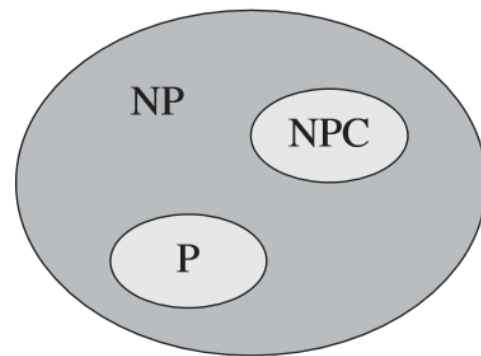
NP类问题包括了实际应用领域的几乎所有判定问题，这些问题的算法在最坏情况下的时间复杂性都是一些形如 2^n 、 $n!$ 等快速增长的指数函数。

我们希望能够在NP类问题的内部找到一种方法，比较两个问题的计算复杂性，并进一步找到NP类问题中最难的问题。

- 比较两个问题的计算复杂性的方法就是问题变换（归约）。

定义6 令 Π 是一个判定问题，如果 Π 属于NP类问题，并且对于NP类问题中的每一个问题 Π' ，都有 $\Pi' \leq_p \Pi$ ，则称判定问题 Π 是一个NP完全问题，简称NPC（NP-Complete）。

- ◆ NPC问题是NP类问题的一个子类，同时也是NP类问题中最难的一类，解决了它，则NP类问题就都可以求解了。
- ◆ 但到目前为止任何一个NPC问题都还没有找到多项式时间算法。

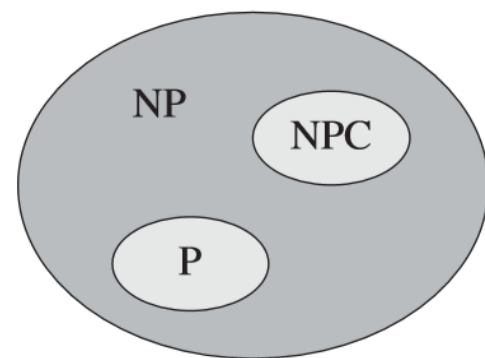


NPC问题有一种非常令人惊奇的性质：即如果一个NPC问题能在多项式时间内得到解决，那么，NP中的每一个问题都可以在多项式时间内求解，即会有 $NP=P$ 。

- 但是目前谁也没有找出关于任何NP完全问题的多项式时间的算法

- 对 $P=NP?$ 问题的研究都是以NPC问题为中心开展的。

- 大部分从事理论研究的计算机科学家都认为 $P \neq NP$ ，即NPC类中的所有问题，都不存在多项式时间算法 即**所有NP完全问题都不是多项式时间可解的。**



大多数理论计算机科学家眼中的P、NP和NPC三者之间的关系，P和NP都完全包含在NP内，且 $P \cap NPC = \emptyset$

NP-hard:

NP类问题是可以利用多项式时间的非确定性算法来进行判定的问题，但还不是难解问题的全部。难解问题中还有一类问题，虽然也能证明所有的NP类问题可以在多项式时间内变换为问题 Π ，但是并不能证明问题 Π 是NP类问题（不能用确定性算法进行判定），所以问题 Π 不是NPC的，但是问题 Π 至少与任意NP类问题有同样的难度。

定义7 如果对于NP类问题中的每一个问题 Π' 都有 $\Pi' \propto_p \Pi$ ，
则称问题 Π 是一个NP-hard问题（NP-难问题）。

◆ **NPC \neq NP-hard**，但 **NPC \in NP-hard**，NP难中还包括一类比NP类问题更难的问题。

- ◆ 也可以认为 “NP难” 问题 是比NP问题 “更难” 的问题。

一般而言 若判定问题属于NP完全问题，则相应的最优化问题属于NP难问题。

- 判定图G中是否存在汉密顿回路是NPC问题，而求解汉密顿回路中最短路径的TSP问题是NP难问题。
- 判定图 $G=(V, E)$ 中是否存在k个顶点的团问题是NPC问题，而求图G中顶点数最多的团问题则是NP难问题。

算法导论中的关键描述：

- P类中包含的是**多项式时间内可解**的问题，即一些可以在 $O(n^k)$ 时间内求解的问题。
 - 此处 k 为某个常数， n 是问题的输入规模。
- NP类中包含的是在**多项式内“可验证”**的问题，此处是指给出某一解决方案的“证书”，就能够在关于问题输入规模的多项式时间内，验证该证书是否是正确的。
- P中的任何问题都属于NP，因为如果某一问题是属于P的，则可以在不给出证书的情况下，在多项式时间内解决它。

算法导论中的关键描述：

- 关于形式语言体系的描述（略）
- 如果一个问题属于NP，且与NP中的任何问题是一样“难”的，则说它属于NPC类 也称为NP完全的。

“一样难”的含义：别的问题至少都可以以多项式时间规约为该问题。

■ 证明一个问题是NP完全问题

- 首先明确：证明某一特定问题是NP完全问题时，我们并不是要证明存在某个有效的算法去求解该问题，而是要**证明不太可能存在一个有效的算法。**
- 我们是在陈述它是一个多么困难的问题。

如何证明一个问题是NP完全问题？

证明一个判定问题 Π 是**NP完全问题**需要经过两个步骤：

(1)**证明问题 Π 属于NP类问题**。即可以在多项式时间以确定性算法验证一个猜测的解是不是问题的真正解。

(2)**证明NP类问题中的每一个问题都能在多项式时间变换为问题 Π** 。由于**多项式问题变换具有传递性**，所以，只需证明一个已知的NPC问题能够在多项式时间变换为问题 Π 即可。

- **证明是NP的**：即可以在多项式时间以确定性算法验证一个解。
- **证明是NP难的**：即证明一个已知的NP完全问题可以在多项式时间内转化为要证明的问题。

第一个NPC问题

如果我们证明了至少有一个问题是NP完全问题，就可以用多项式时间可归约性作为工具，来证明其他问题也具有NP完全性。

- 可满足性问题就是这第一个NPC问题。
- 1971年，Cook证明了可满足性问题（SAT，布尔可满足问题）是NP完全的（**Cook定理**）。
- 1972年，Karp证明了十几个问题都是NP完全的。
- 这些NP完全问题的证明为以后问题的证明奠定了基础。

34.3 电路可满足性 (CIRCUIT-SAT)

布尔组合电路是由**布尔组合元素**通过线路互联后构造而成的电路。

布尔组合元素是指任何一种电路元素，它有着固定数目的输入和输出，执行的是某种良定义(well-defined)的函数功能——**布尔函数**。

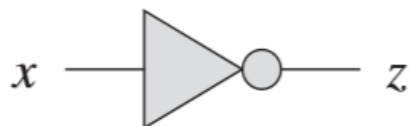
布尔函数的值取自**布尔集合** $\{0,1\}$ 。

➤ 这些元素称为逻辑门。有三种基本的逻辑门：

NOT门（非门）、**AND门**（与门）、**OR门**（或门）

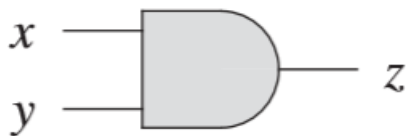
每一种门及其操作都可以用一个真值表来描述。

三种基本的逻辑门



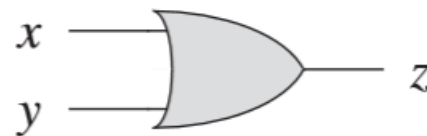
x	$\neg x$
0	1
1	0

(a)



x	y	$x \wedge y$
0	0	0
0	1	0
1	0	0
1	1	1

(b)



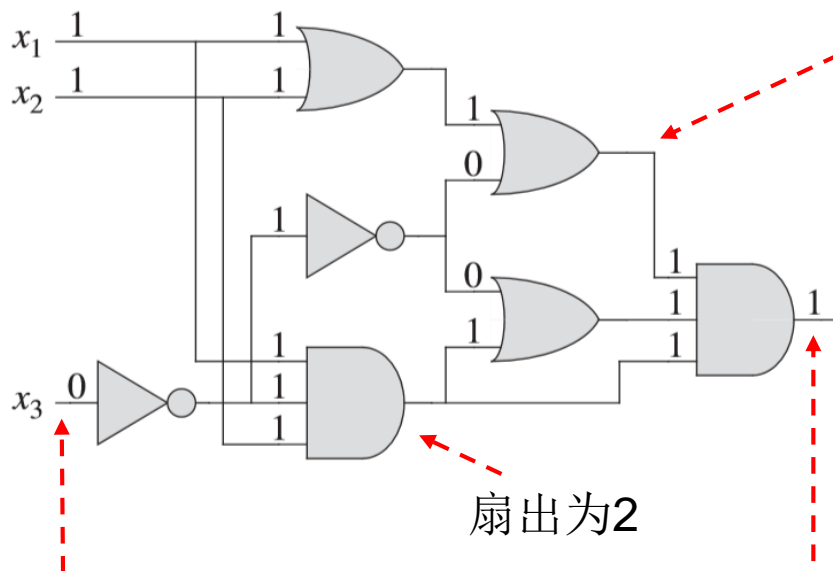
x	y	$x \vee y$
0	0	0
0	1	1
1	0	1
1	1	1

(c)

Figure 34.7 Three basic logic gates, with binary inputs and outputs. Under each gate is the truth table that describes the gate's operation. **(a)** The NOT gate. **(b)** The AND gate. **(c)** The OR gate.

电路可满足性 (CIRCUIT-SAT) :

布尔组合电路由一个或多个布尔组合元素通过线路连接而成。



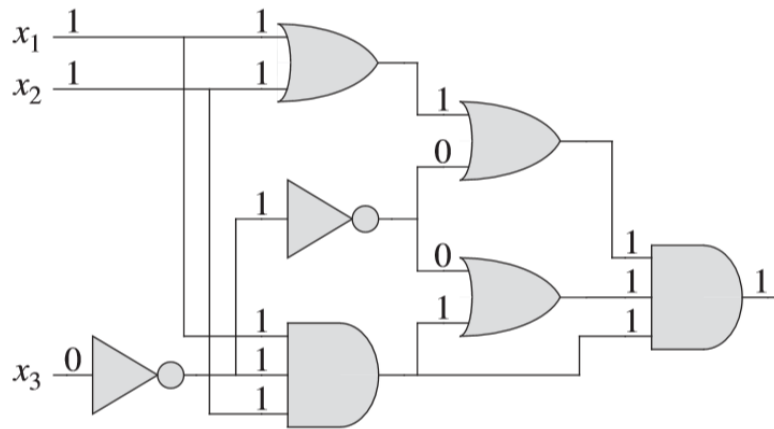
线路将某一元素的输出与另一个元素的输入连接起来。

- 虽然一根线上不可能有多于一个的布尔元素的输出与其相连，但它可以作为其他几个元素的输入。
- 由一个接线提供输入的元素个数称为该接线的**扇出**(fan-out)
- 布尔电路不包含回路

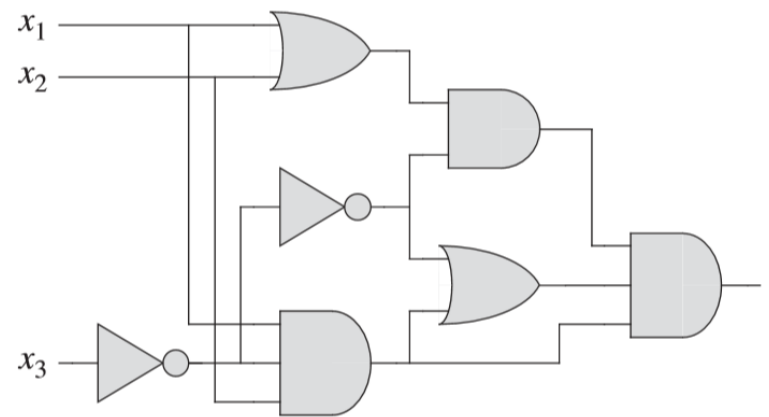
如果没有哪一个元素的输出接到某个接线上的话，则称该接线为**电路的输入**，它接受来自外部的数据。

如果没有哪一个元素的输入连接到某根接线上的话，则称该接线为**电路的输出**。

- 一个布尔组合电路的**真值赋值**是指一组布尔输入值。
- 如果一个**单输出**布尔组合电路具有一个**可满足性赋值**，即**可以使得电路输出为1的一个真值赋值**，称该布尔组合电路是**可满足的**。



(a)



(b)

- 电路(a)是可满足的：对电路(a)的输入赋值(1,1,0)，使得电路的输出为1。
- 电路(b)是不可满足的：对电路(b)的任何一种赋值都不能使得电路的输出为1。

电路可满足性问题：对一个给定的布尔组合电路，判断它是否是可满足电路。

- 布尔组合电路的**规模**是指其中布尔组合元素的个数，再加上电路中接线的条数。
- **电路可满足性问题是算法导论证明的第一个NP完全问题。**
- 后面假设该问题已经被证明过。
 - (证明过程略)

即证明了电路可满足性问题是NP类的且是NP难的。

34.4 布尔公式可满足性 (SAT问题)

布尔公式可满足性问题 (SAT) 描述如下:

- SAT的一个实例是一个由下列成分组成的布尔公式 ϕ :
 - n 个**布尔变量**: x_1, x_2, \dots, x_n ;
 - m 个**布尔连接词**: 具有一个或两个输入和一个输出的任何布尔函数, 如 \wedge (与), \vee (或), \neg (非), \rightarrow (蕴含), $\boxed{\leftrightarrow}$ (当且仅当)
 - **括号**。(不失一般性, 假设没有冗余的括号)
- 一个布尔公式 ϕ 的真值赋值是为 ϕ 中各个变量所取的一组布尔值; **可满足性赋值**是指使公式 ϕ 的值为1的真值赋值。
- 具有可满足性赋值的公式就是**可满足公式**。

合取范式的可满足性问题(CNF-SAT问题, conjunctive normal form)

- 一个合取范式是形如 $A_1 \wedge A_2 \wedge \dots \wedge A_n$ 的表达式,
 - 其中, **子句** A_i 是形如 $a_1 \vee a_2 \vee \dots \vee a_n$ 的式子,
 - 其中 a_i 称为文字, 为某一布尔变量或该布尔变量的非。
- **CNF-SAT问题**: 判定是否存在一组对所有布尔变量的赋值, 使得一个合取范式取值为真。
- **3合取范式的可满足问题(3CNF-SAT问题)**
 - 每个 $A_i (i=1, \dots)$ 恰好都有三个不同的文字
- **析取范式**: 交和并对换

定理34.9 布尔公式的可满足性问题是NP完全的。

证明：首先证明 $\text{SAT} \in \text{NP}$ ，接着证明 $\text{CIRCUIT-SAT} \leq_p \text{SAT}$

——根据引理34.8，这将证明定理成立。

(1) 证明 $\text{SAT} \in \text{NP}$

证明SAT属于NP，即是证明对于输入公式 ϕ ，它一个待测的可满足性赋值所组成的证书可以在多项式时间内得到验证。

可表述为：验证算法将公式中的每个变量对应赋值，再对表达式求值。这一任务**很容易在多项式时间内完成**。

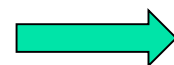
——布尔变量数、子句数等都是多项式的。

如果表达式的值为1，则说明公式是可满足的。

(2) 证明SAT是NP难度的

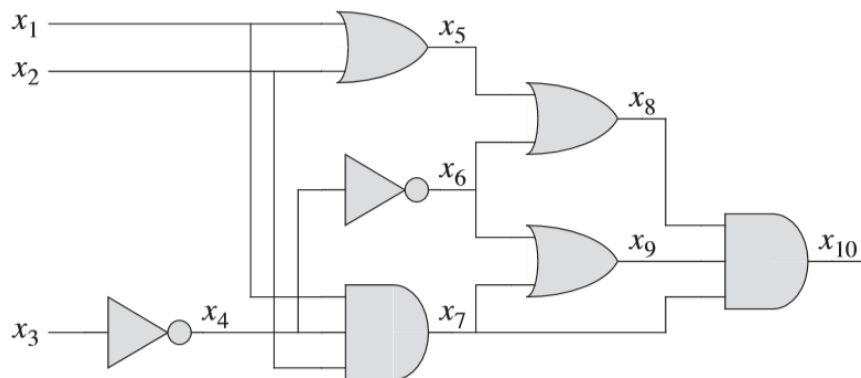
根据前面的描述，CIRCUIT-SAT是我们已知的“第一个”NPC问题，亦即所有NP问题都可以归约为CIRCUIT-SAT问题。因此，y要证明SAT问题是NP难度的问题，只要证明

$$\text{CIRCUIT-SAT} \leq_p \text{SAT}$$

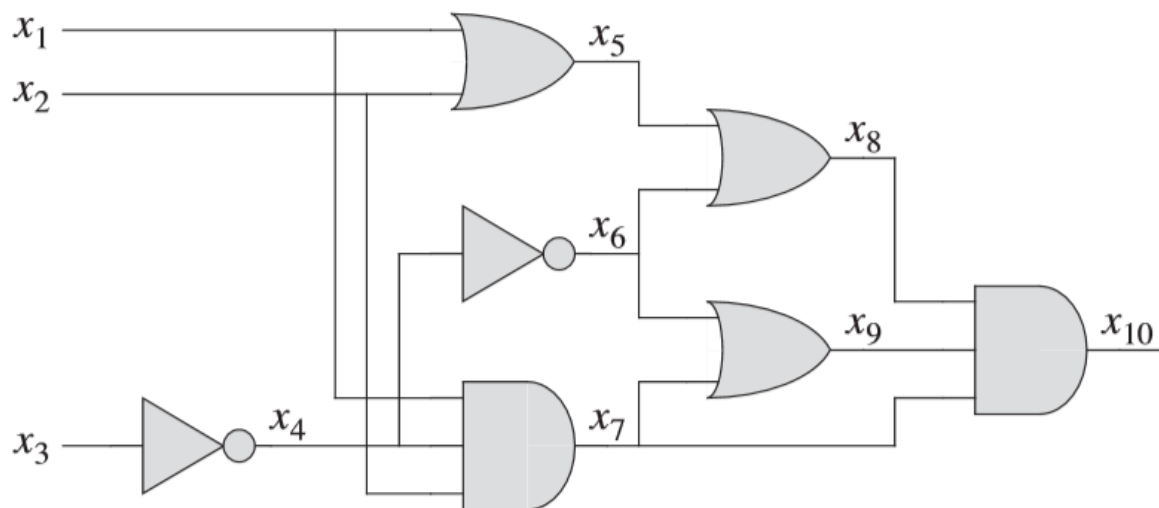


即电路可满足性问题的任何实例可以在多项式时间内归约为公式可满足性问题的一个实例。

下图说明了由CIRCUIT-SAT归约为SAT的基本思想：



下图说明了由CIRCUIT-SAT归约为SAT的基本思想：



- 对电路C中的每一根接线 x_i ，公式 φ 都有一个对应的变量 x_i 。
- 每个逻辑门操作都可以表示为关于其附属线路变量的公式。

例如：输出 x_{10} 的门操作为： $x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)$ 。（与门）

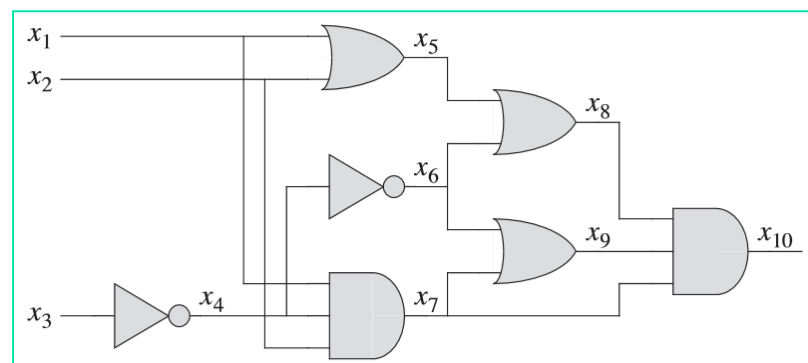
这里，把这些小式子（small formula）称为“子句”
（clause）

记根据上述归约算法产生的公式为 ϕ :

➤ 它是电路输出变量与描述每个门操作的子句的“与连接”的“与”。

➤ 对图中的电路，相应的公式为：

$$\begin{aligned}\phi = & x_{10} \wedge (x_4 \leftrightarrow \neg x_3) \\ & \wedge (x_5 \leftrightarrow (x_1 \vee x_2)) \\ & \wedge (x_6 \leftrightarrow \neg x_4) \\ & \wedge (x_7 \leftrightarrow (x_1 \wedge x_2 \wedge x_4)) \\ & \wedge (x_8 \leftrightarrow (x_5 \vee x_6)) \\ & \wedge (x_9 \leftrightarrow (x_6 \vee x_7)) \\ & \wedge (x_{10} \leftrightarrow (x_7 \wedge x_8 \wedge x_9)) .\end{aligned}$$



电路的规模是门电路数量和线路数的线性函数，所以这一转换是P的

上述过程实现了转换，但转换的对不对呢？

—— 即，需要说明**解的可归约性**，即SAT的解是不是CIRCUIT-SAT的解？

■ 下面说明该过程满足解的可归约性：

如果电路C具有一个可满足性赋值，则电路的每条线路都有一个良定义的值，并且整个电路的输出为1。因此，用线路的值对公式 ϕ 中的每个变量赋值后，就使得 ϕ 中每个子句的值为1，因而，所有子句的合取值也为1——可满足。

反之，如果存在一个赋值使得 ϕ 的值为1，则类似可证电路C是可满足的。

这样，我们就证明了CIRCUIT-SAT \leq_p SAT。

证毕

34.4 3-CNF可满足性 (3SAT)

- **文字**：布尔公式中的一个**文字**(literal)是指一个**变量**或**变量的“非”**。
- **合取范式**：如果一个布尔公式可以表示为所有子句的“与”，且每个子句都是一个或多个文字的“或”，则称该布尔公式为**合取范式** (CNF, conjunctive normal form)。
- **3合取范式**：如果公式中每个子句恰好都有**三个不同的文字**，则称该布尔公式为**3合取范式** (3-CNF)。

例：布尔公式

$$(x_1 \vee \neg x_1 \vee \neg x_2) \wedge (x_3 \vee x_2 \vee x_4) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4)$$

是一个三合取范式。

定理34.10 3-合取范式形式的布尔公式的可满足性问题是NPC问题

证明：

(1) 由于3SAT是SAT的特例，因此3SAT属于NP。

(可采用证明 $\text{SAT} \in \text{NP}$ 一样的方法进行证明)

(2) 因此我们只需要证明 $\text{SAT} \leq_p \text{3-CNF-SAT}$ 。

归约可以分为三个步骤，每一步逐渐使输入的公式向3-合取范式接近。

首先，为输入的公式 ϕ 构造一棵二叉“语法分析树”：

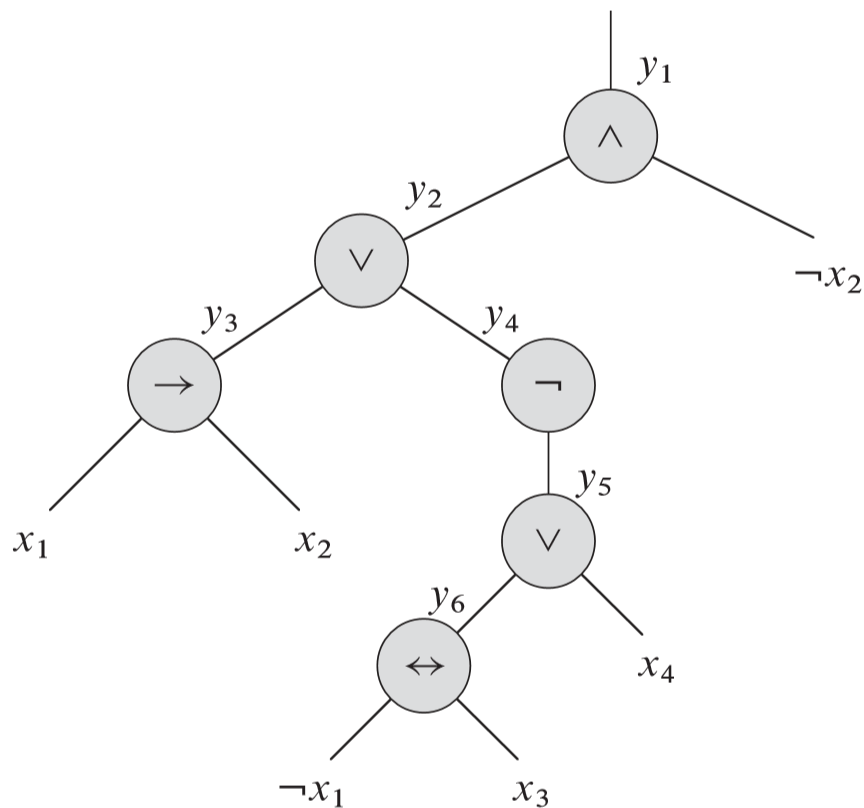
—— 以文字作为树叶，以连接词作为内部节点。

如下图所示。

如公式

$$\phi = ((x_1 \rightarrow x_2) \vee \neg((\neg x_1 \leftrightarrow x_3) \vee x_4)) \wedge \neg x_2$$

的语法分析树如下：

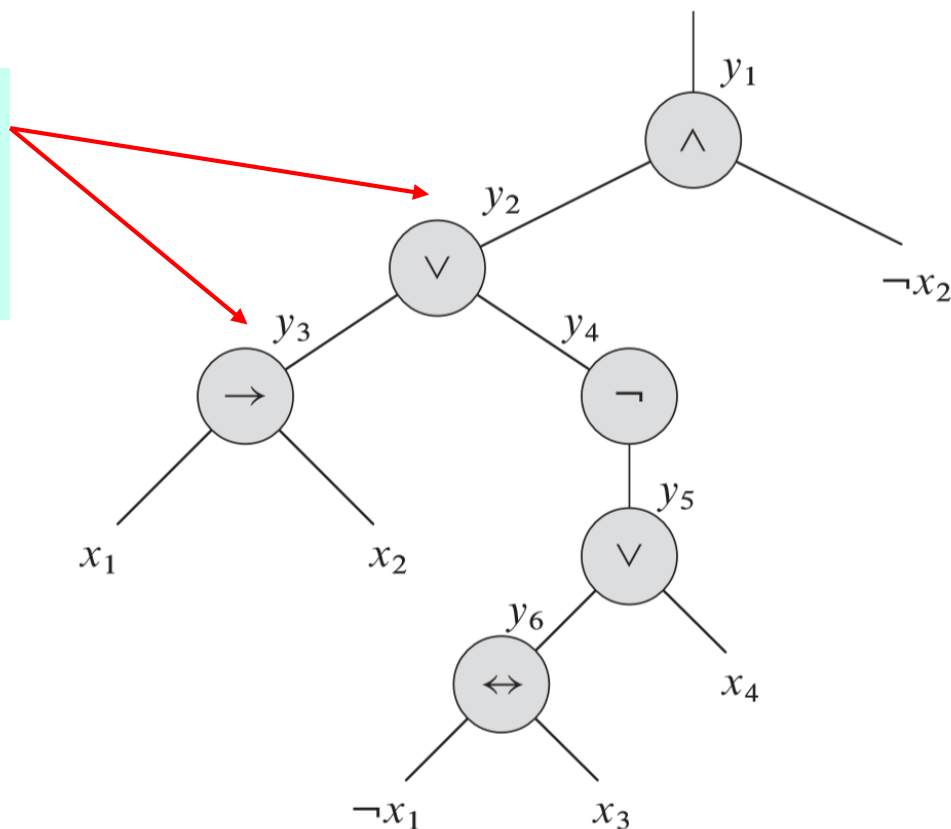


如果输入公式中包含数个文字的“或”等子句，就可以利用结合律对表达式加上括号，使得在所产生的树中的每一个内部节点均有一个或两个孩子。

把二叉语法分析树看作是计算该函数的一个电路：

—— 从叶子结点输入，向根方向变换，根代表电路的输出。

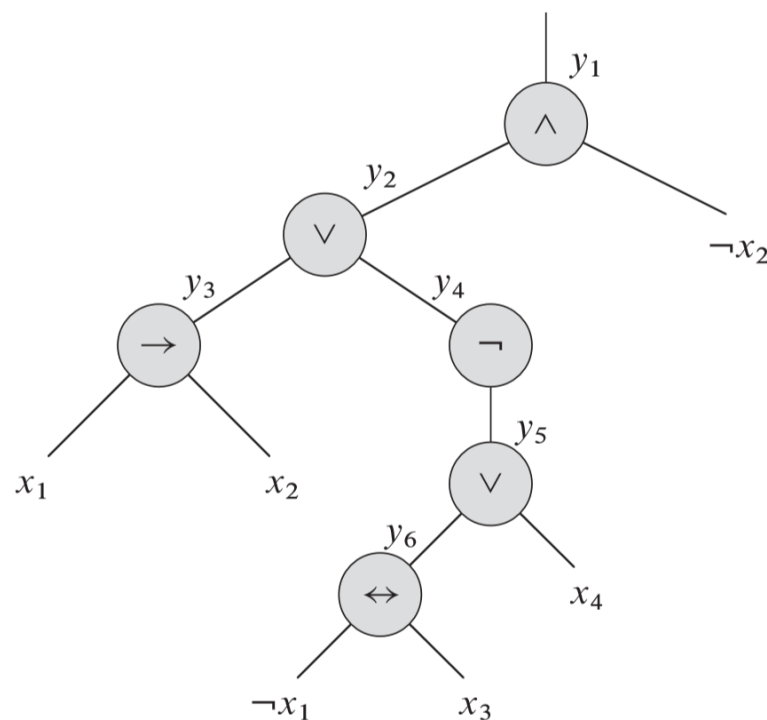
为每个内部结点的输出引入一个变量 y_i 。



从而把原始公式 φ 改写为根变量与描述每个结点操作的子句的“与”连接的“与”。

经改写后得到的表达式为：

$$\begin{aligned}\phi' = & y_1 \wedge (y_1 \leftrightarrow (y_2 \wedge \neg x_2)) \\ & \wedge (y_2 \leftrightarrow (y_3 \vee y_4)) \\ & \wedge (y_3 \leftrightarrow (x_1 \rightarrow x_2)) \\ & \wedge (y_4 \leftrightarrow \neg y_5) \\ & \wedge (y_5 \leftrightarrow (y_6 \vee x_4)) \\ & \wedge (y_6 \leftrightarrow (\neg x_1 \leftrightarrow x_3))\end{aligned}$$



注：这样得到的公式 Φ' 是所有子句 Φ_i' 的合取式，每个子句 Φ_i' 至多有3个文字。

第二步是把 Φ' 中每个子句 φ_i' 变换为合取范式。

- 通过对 φ_i' 中变量的所有可能的赋值进行计算，我们构造出 φ_i' 的**真值表**。
- 真值表中的每一行由子句变量的一种可能的赋值和根据这一赋值所计算出的子句的值组成。
- 然后利用真值表中**值为0的项**构造出一个的析取范式。
- 最后运用**DeMorgan定律**把所有文字取补，并把“或”变成“与”，“与”变成“或”，就可以把该公式变成CNF公式 φ_i'' ：

例：把子句 $\varphi_1' = (y_1 \leftrightarrow (y_2 \wedge \neg x_2))$ 变换为CNF。

下图为 φ_1' 的真值表。

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	<u>0</u>
1	1	0	1
1	0	1	<u>0</u>
1	0	0	<u>0</u>
0	1	1	1
0	1	0	<u>0</u>
0	0	1	1
0	0	0	1

取其中值为0的项，与 $\neg\varphi_1'$ 等价的DNF公式为：

$$\begin{aligned} & (y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \\ & \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2) \end{aligned}$$

y_1	y_2	x_2	$(y_1 \leftrightarrow (y_2 \wedge \neg x_2))$
1	1	1	0
1	1	0	1
1	0	1	0
1	0	0	0
0	1	1	1
0	1	0	0
0	0	1	1
0	0	0	1

取其中值为0的项，与 $\neg \phi_1'$ 等价的DNF公式为：

$$(y_1 \wedge y_2 \wedge x_2) \vee (y_1 \wedge \neg y_2 \wedge x_2) \\ \vee (y_1 \wedge \neg y_2 \wedge \neg x_2) \vee (\neg y_1 \wedge y_2 \wedge \neg x_2)$$

取反，并利用DeMorgan定律，得到CNF公式如下：

$$\phi_1'' = (\neg y_1 \vee \neg y_2 \vee \neg x_2) \wedge (\neg y_1 \vee y_2 \vee \neg x_2) \\ \wedge (\neg y_1 \vee y_2 \vee x_2) \wedge (y_1 \vee \neg y_2 \vee x_2),$$

显然，该式等价于原子句 ϕ_1'

这样公式 ϕ' 的每个子句 ϕ_i' 被转换为一个CNF公式 ϕ_i'' ，因此 ϕ' 等价于由 ϕ_i'' 的合取组成的CNF公式 ϕ'' 。此外， ϕ'' 的每个子句最多包含3个文字。

第三步，对公式进一步进行变换，使得每个子句恰好有三个不同的文字。最后利用CNF公式 ϕ'' 的子句构造出的3-CNF公式 ϕ''' 。具体如下：

公式 ϕ''' 中使用两个辅助变量 p 和 q 。对 ϕ'' 中的每个子句 C_i ，使 ϕ''' 包含下列子句：

- 如果 C_i 有三个不同的文字 则直接作为 ϕ''' 的一个子句；
- 如果 C_i 有两个不同的文字，即如果 $C_i = (l_1 \vee l_2)$ ，其中 l_1 和 l_2 为文字，则把 $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ 作为 ϕ''' 的子句。
 - 加入 p 和 $\neg p$ 仅仅为了满足每个子句必须有三个不同的文字的语法要求。
 - 无论 $p=0$ 或 $p=1$ ， $(l_1 \vee l_2 \vee p) \wedge (l_1 \vee l_2 \vee \neg p)$ 都等价于 $(l_1 \vee l_2)$

- 如果 C_i 中仅有一个文字 l ，则把

$$(1 \vee p \vee q) \wedge (1 \vee p \vee \neg q) \wedge (1 \vee \neg p \vee q) \wedge (1 \vee \neg p \vee \neg q)$$

作为 ϕ''' 的一个子句。

- 同理，对 p 和 q 的任意取值，上式都等价于 l 。

现在可以看出3-CNF公式 ϕ''' 是可满足的，当且仅当上述三个步骤的每一步中， ϕ 都是可满足的：

如同从CIRCUIT-SAT归约为SAT的过程一样，第一步根据 ϕ 构造 ϕ' 的过程保持可满足性，第二步产生的CNF公式 ϕ'' 在代数上等价于 ϕ' ，第三步产生的3-CNF公式 ϕ''' 也等价于 ϕ'' ，这是因为对变量 p 和 q 的任意赋值所产生的公式代数上与 ϕ'' 等价。

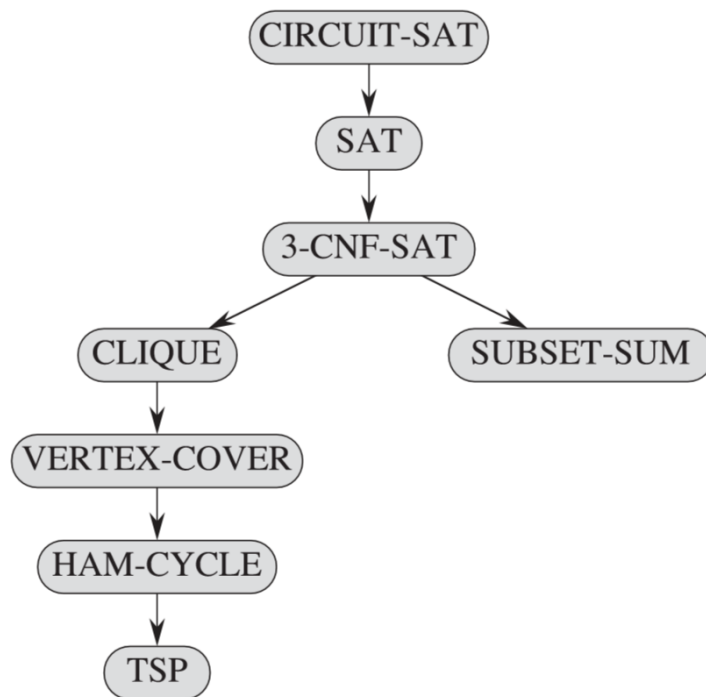
此外，还必须证明归约可以在多项式时间内完成。

- 从 ϕ 构造 ϕ' 中的每个连接词至多引入了一个变量 (y_i) 和一个子句 (ϕ_i') ;
- 从 ϕ' 构造 ϕ'' 的过程对 ϕ' 中的每一个子句至多在 ϕ'' 中引入8个子句，这是因为 ϕ'' 中的每个子句至多有3个变量，因此每个子句的真值表至多有 $2^3=8$ 行。
- 从 ϕ'' 构造 ϕ''' 的过程对 ϕ'' 中的每个子句至多在 ϕ''' 中引入4个子句。

因此，所产生的公式 ϕ 的规模是关于原始公式长度的多项式，
可以容易地在多项式时间内完成每一步构造过程。证毕！

34.5 NP完全问题

下图给出了在上一节和本节中进行的NP完全性证明的流程结构：



对每个问题进行归约，从而证明其NP完全性。其中，根是CIRCUIT-SAT，是在定理34.7中证明了的第一个NPC问题。

34.5.1 团问题 (The clique problem)

- **团**：无向图 $G = \langle V, E \rangle$ 中的团是一个顶点子集 $V' \subseteq V$ ，其中每一对顶点之间都由 E 中的一条边相连。
- **团的规模**：是指它所包含的顶点数。
- **团问题**：寻找图中规模最大的团（优化问题）。
 - 判定问题：在图中是否存在一个给定规模为 k 的团？

形式定义为：

$$C = \{ \langle G, k \rangle : G \text{ 是一个包含规模为 } k \text{ 的团的图} \}$$

团问题的一种朴素解 (The clique problem) :

列出V的所有k-子集, 对其中的每一个进行检查, 看它是否形成一个团。

- 时间复杂度: $\Omega(k^2 \binom{|V|}{k})$
 - 如果k为常数, 则该算法是多项式时间的。
 - 但一般情况下, k可能接近于 $|V|/2$, 则算法的运行时间就是超多项式的——事实上, 团问题的有效算法是不大可能存在的。

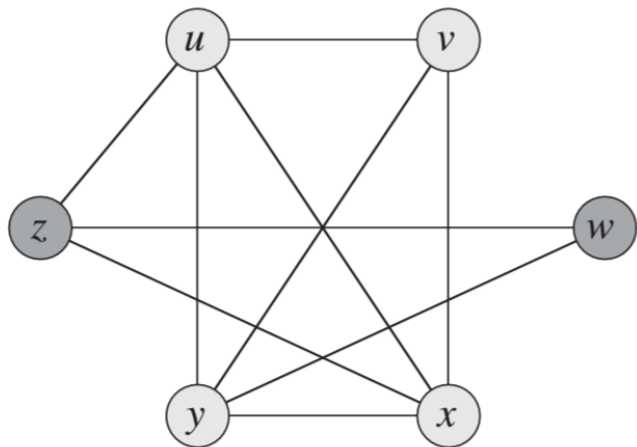
定理34.11 团问题是NP完全的。

证明略。

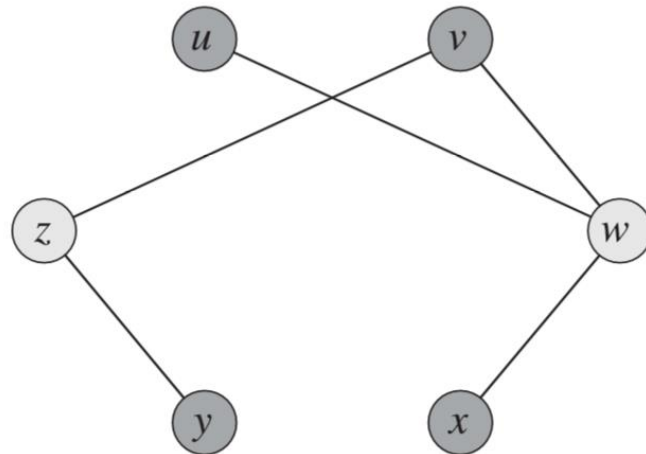
34.5.2 顶点覆盖问题

无向图 $G=(V,E)$ 的顶点覆盖是指子集 $V' \subseteq V$ 满足对所有的边 $(u, v) \in E$, 有 $u \in V'$ 或 $v \in V'$ (或两者都成立)。

- ▶ 即，每个顶点“覆盖”与其相关联的边，并且 G 的顶点覆盖是覆盖 E 所有边的顶点所组成的集合。
- ▶ 顶点覆盖的**规模**是指它所包含的顶点数。



(a)



(b)

顶点覆盖问题：

在给定的图中，找出具有最小规模的顶点覆盖。

把问题重新表述为判定问题：

$\text{VERTEX-COVER} = \{ \langle G, k \rangle : \text{图}G\text{具有规模为}k\text{的顶点覆盖} \}$

定理34.12 顶点覆盖问题是NP完全的

证明：（略）

34.5.3 汉密顿回路问题

定理34.12 汉密顿回路问题是NP完全问题

证明：（略）

34.5.4 旅行商问题

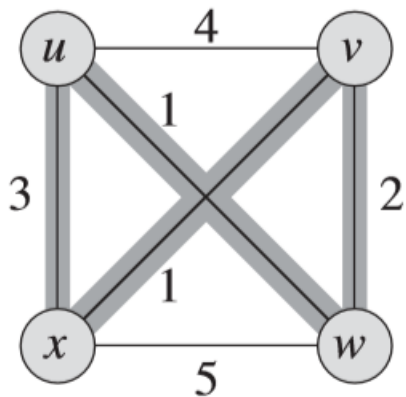
一个售货员必须访问 n 个城市。经过汉密顿回路，恰好访问每个城市一次，并最终回到出发的城市。

从城市 i 到城市 j 的旅行费用为一个整数 $c(i,j)$ ，售货员旅行所需的全部费用是他旅行经过的各边费用之和。售货员希望整个旅行费用最低。

与旅行商问题对应的判定问题是：

$$\text{TSP} = \{ \langle G, c, k \rangle : G = (V, E) \text{ 是一个完全图, } c \text{ 是 } V \times V \rightarrow \mathbb{Z} \text{ 上的一个整数, } k \in \mathbb{Z}, G \text{ 是否包含一个费用至多为 } k \text{ 的旅行回路。} \}$$

例：



旅行商问题的一个实例，阴影覆盖的边表示费用最低的旅行路线，其费用为7。

定理34.14 旅行商问题是NP完全的

证明：（略）

34.5.5 子集和问题

已知一个正整数的有限集 S 和一个整数目标 $t > 0$ ，问是否存在一个子集 $S' \subseteq S$ ，其元素和为 t 。

例：设 $S = \{1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993\}$ ， $t = 138457$ 。

则子集 $S' = \{1, 2, 7, 98, 343, 686, 2409, 17206, 117705\}$ 是该问题的一个解。

子集和问题的定义：

$\text{SUBSET-SUM} = \{ \langle S, t \rangle : \text{存在一个子集 } S' \subseteq S, \text{ 满足 } t = \sum_{s \in S'} s \}.$

定理34.15 子集和问题是NP完全的。

NP完全问题的计算机处理

(1)采用先进的算法设计技术：寻找多项式时间算法

(2)充分利用限制条件

许多问题虽然被归结为NP完全问题，但是它会包含某些限制条件，而有些问题加上限制条件后可能会改变性质。

如：0/1背包问题中，限定物品的重量和价值均为正整数，则利用动态规划，背包问题存在一个伪多项式时间 $O(nW)$ 的算法。

令 $S^i = \{ (P_j, W_j) \mid \text{为从第1件物品到第} i \text{件物品, 在总放置重量不超过} W \text{的前提下, 所有可能的放置方案所获得的效益值和所占重量} \}$

若每件物品的重量 w_i 和效益值 p_i 均为 **整数**, 则 S^i 中每个序偶 (P, W) 的 P 值和 W 值也是整数, 且有 $P \leq \sum_{0 \leq j \leq i} |p_j|$, $W \leq M$

问题: 在 $1 \sim n$ 范围内有多少个互异的整数?

答案: n 个

在任一 S^i 中, 所有序偶具有 **互异** P 值和 W 值。

故有: $|S^i| \leq 1 + \sum_{0 \leq j \leq i} |p_j|$ $|S^i| \leq 1 + \min\{ \sum_{0 \leq j \leq i} |w_j|, M \}$

至少有一个 $(0,0)$

故，在所有 w_j 和 p_j 均为整数的情况下，0/1背包问题的时间和空间复杂度将为：

$$O(\min \{2^n, n \sum_{1 \leq i \leq n} \lfloor p_i \rfloor, nW\})$$

■ 尽管背包问题的时间复杂度为 $O(nW)$ ，但它仍然是一个NP完全问题。这是因为 W 同问题的输入大小并不成线性关系。

- 问题的输入大小仅取决于表示输入所需的比特数。事实上， $\log W$ ，即表示 W 所需的比特数，同问题的输入长度成线性关系。
- 背包问题存在完全逼近多项式时间的方案，这使得在计算机科学领域，人们对背包问题很感兴趣。但作为NP完全问题，背包问题没有一种既准确又快速（多项式时间）的算法。

(3)近似算法

很多问题的输入是近似的，同时很多问题的解允许有一定误差。采用近似算法可以在一定时间内得到问题的近似解。

(4)概率算法

(5)并行算法

利用多台处理机共同完成一项计算，是解决计算密集型问题的必经之路。

(6)智能算法

遗传算法、人工神经算法、DNA算法、蚁群算法、免疫算法、模拟退火算法。