问题

Problem

刘 铎

liuduo@bjtu.edu.cn

易解性

- 易解的问题[Cobham 1964, Edmonds 1965, Rabin 1966]：存在关于输入大小（例如，字节数）的多项式时间算法
- 难解的问题：不存在关于输入大小的多项式时间算法

指数增长

- 指数增长让算法技术的变革相形见绌
  - 假设宇宙中的每个电子都相当于一台超级计算机
  - 且每一个都自宇宙诞生至今不断工作
  - 来求解一个 ***n*!** 复杂度的算法

数值参考	
不同的量	数量级
超级计算机每秒可执行的指令数	10^{15}
每年的秒数	10^{8}
宇宙的（估计）年龄（单位：年）	10^{10}
宇宙中电子的（估计）数目	10^{79}

- 但是都无法处理有 **1,000** 个城市的TSP！
- $\mathbf{1000! = 4.02\times10^{2567} >> 10^{1000} >> 10^{79}\times10^{10}\times10^{8}\times10^{15}}$

对问题进行分类

- 迫切需求
  - 根据多项式时间内可以解决的问题和不能解决的问题来对问题进行分类

问题的类型

Types of Problems

问题的类型

- 最优化问题（**optimization problem**）：构造一个解可将某个目标函数最大化或最小化
- 判定性问题（**decision problem**）：仅要求回答为“是”或“否”（YES/NO）的问题
- 示例：哈密顿回路问题
  - 判定性问题为：给定的无向图是否存在哈密顿回路

问题的类型

- 许多问题都会有最优化版本和判定性版本
- 例如：TSP
  - 最优化问题：给定一个赋权图，求总权值最小的哈密顿回路
  - 判定性问题：给定一个赋权图和一个整数 k，是否存在总权值不超过 k 的哈密顿回路？

问题的类型

- 许多问题都会有最优化版本和判定性版本
- 例如：背包问题
  - 最优化问题：找到可放入背包的一个对象子集，使得总价值达到最大
  - 判定性问题：给定 k，是否存在一个可放入背包的对象子集，总价值至少为 k？

问题的类型 —— 自归约

- 显然，如果我们可以在（关于 $\log n + n \times \log(\max(v_i, w_i))$的）多项式时间内求解背包问题，那么就也可以通过比较最优解 O 和阈值 k 来轻松地解决判定性背包问题

问题的类型 —— 自归约

- 反过来，如果我们可以在时间 T 内求解判定性版本，那么就可以使用类似二分查找的方法来计算最优解
- 使用类似二分查找的方法来计算最优解
- 设所有的量都是整数，则最佳收益的值在 $[1, n \cdot v_{\max}]$ 范围内，其中 $v_{\max}$ 是任一物品的最大收益
- 因此，我们可以通过不超过 $O(\log n + \log p_{\max})$ 次调用判定性过程来找到最佳收益
  - 这关于输入大小（ $\log n + n \times \log(\max(v_i, w_i))$ ）是线性的
- 因此，这两种版本在计算有效性方面是密切相关的
- 本部分理论主要是围绕判定性问题发展起来的

多项式时间归约

Polynomial-Time Reductions

可归约性（*reducibility*）

- 两个问题之间可归约性（reducibility）的直观概念是：如果我们能够有效地解决其中一个问题，那么我们也能有效地解决另一个问题
- 如果你需要解决一个问题，那么可以把它归约到另一个你知道如何解决的问题
- 示例
  - 通过计算 LCS 来得到 SCS
  - 通过计算 GCD 来得到 LCM
  - 将（一些）最大化问题转化为最小化问题，反之亦然

多项式时间归约

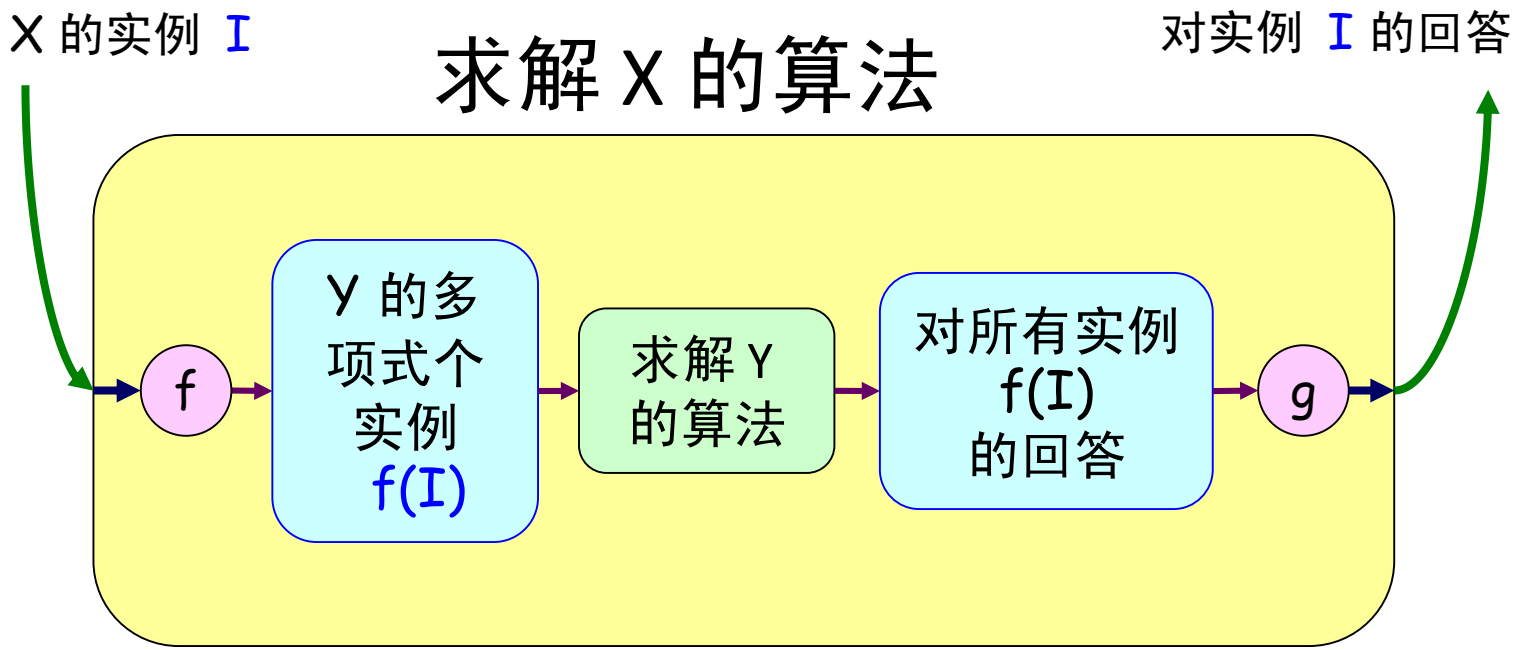
- 迫切需求
  - 假设我们可以在多项式时间内求解问题 Y 了，那么我们还可以（由此）在多项式时间内求解哪些问题呢？
- 多项式时间归约
  - 如果问题 X 的任意实例可以使用以下方法解决，则问题 X 是多项式规约到问题 Y 的
    - 多项式次数个标准计算步骤，以及
    - 对求解问题 Y 的预言（oracle）的多项式次调用
      - 有些模型中只允许调用求解 Y 的预言一次
- 记做 $X \leq_P Y$

多项式时间归约

- 预言（oracle）
  - 假设我们有一个黑盒子（一个算法），可以解决问题 Y 的实例
  - 如果我们输入 Y 的一个实例，那么在单个步骤/单次操作中，黑盒将返回正确的回答

多项式时间归约（示意图）

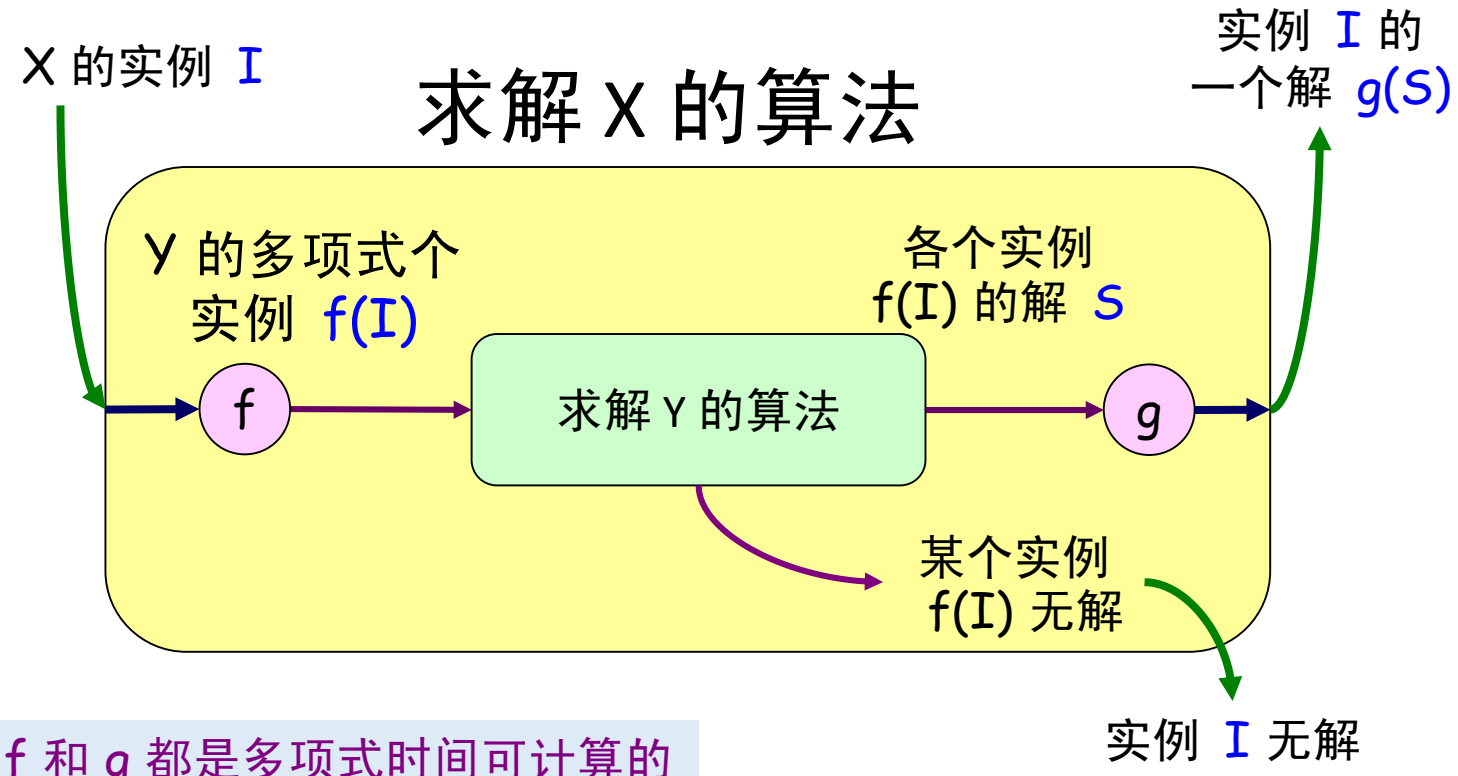
- $X \leq_P Y$
- X 可以多项式时间归约到 Y



f 和 g （关于 X 的输入规模）都是多项式时间可计算的

多项式时间归约（示意图）

- $X \leq_P Y$
- X 可以多项式时间归约到 Y



f 和 g 都是多项式时间可计算的

多项式时间归约

- 自反性
  - $X \leq_P X$
- 证明的梗概
  - 事实上并没有进行“归约”

多项式时间归约

- 传递性
  - 若 $X \leq_P Y$ 且 $Y \leq_P Z$，则 $X \leq_P Z$
- 证明的梗概
  - 将两个归约过程进行“复合”
  - “多项式的多项式还是多项式”

多项式时间归约

- 目的：根据相对困难程度难度对问题进行分类

- 设计算法：如果 $X \leq_P Y$，而且 Y 可以在多项式时间内求解，那么 X 也可以在多项式时间内求解
- 确立难解性：如果 $X \leq_P Y$，而且 X 不能在多项式时间内求解，那么 Y 也不能在多项式时间内求解
- 建立等价关系：如果 $X \leq_P Y$ 和 $Y \leq_P X$，则记 $X \equiv_P Y$

基本思路

- 建立等价关系

→ 进行等价类划分

→ 寻找代表元

→ 对代表元进行研究

→ 进而对同一个等价类中的问题进行归约，解决一批问题

- 例如算法复杂度的阶就是定义和使用了等价关系Θ

多项式时间归约示例

Reduction Examples

点独立集问题

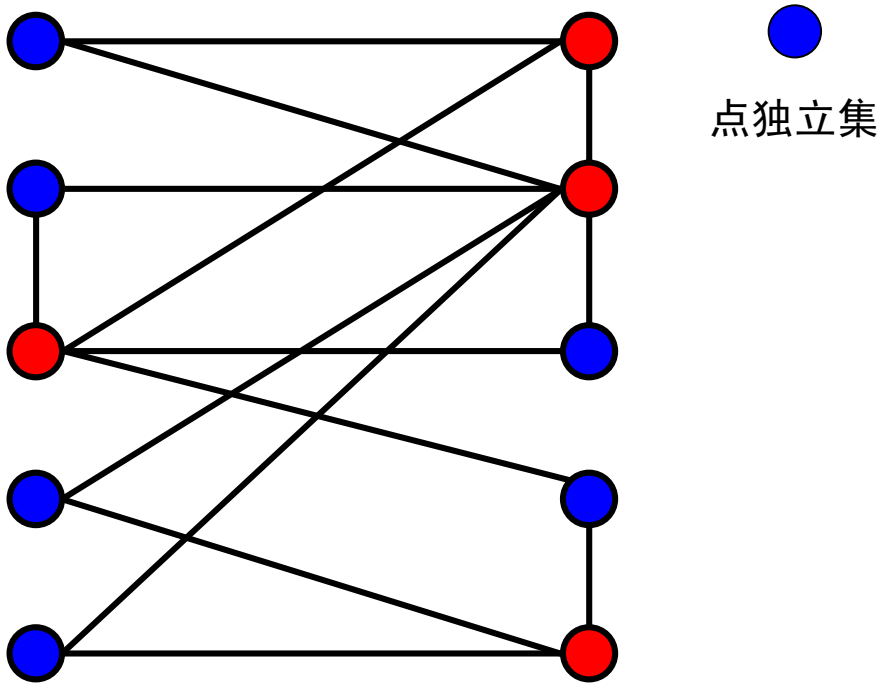
- 点独立集（INDEPENDENT-SET）：给定简单图 $G = (V, E)$ 以及一个整数 k，是否存在顶点的子集 $S \subseteq V$ 使得 $|S| \geq k$，且图中任一条边都至多只有一个端点在 S 中？

图中是否有基数至少为6的点独立集？

回答为 Yes

图中是否有基数至少为7的点独立集？

回答为 No



点独立集

顶点覆盖问题

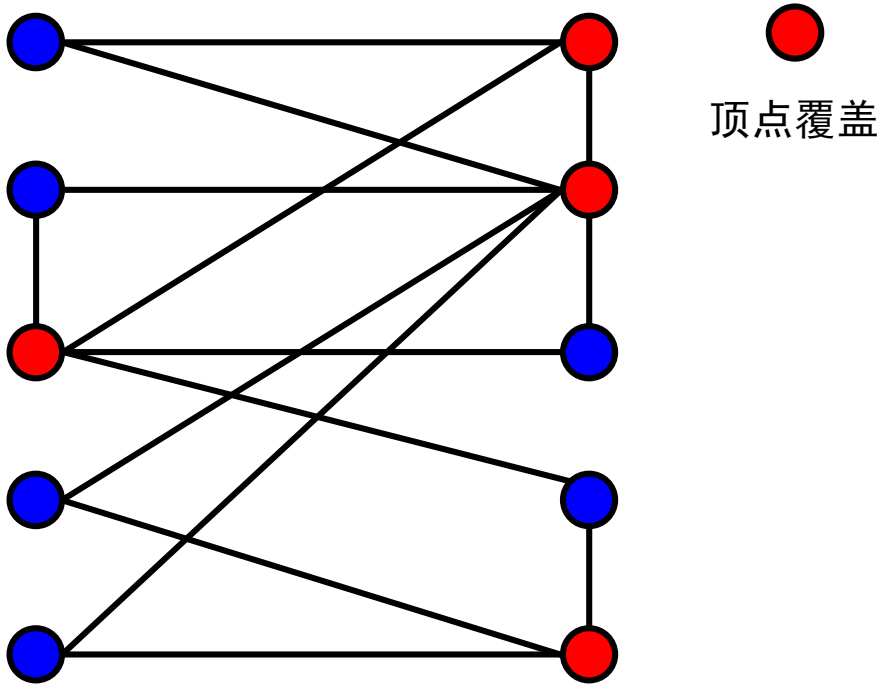
- 顶点覆盖（VERTEX-COVER）：给定简单图 $G = (V, E)$ 以及一个整数 k，是否存在顶点的子集 $S \subseteq V$ 使得 $|S| \leq k$，且图中任一条边都至少有一个端点在 S 中？

图中是否有基数至多为4的顶点覆盖？

回答为 Yes

图中是否有基数至多为3的顶点覆盖？

回答为 No



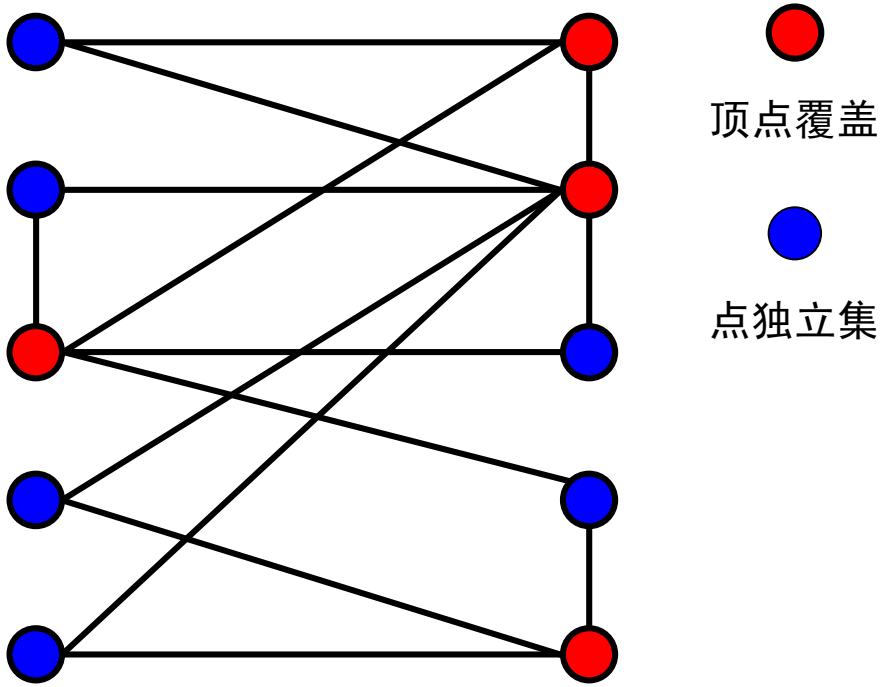
顶点覆盖

顶点覆盖问题

- 断言

  VERTEX-COVER $\equiv_P$ INDEPENDENT-SET

- 证明
  - *S* 是点独立集当且仅当 *V* – *S* 是顶点覆盖

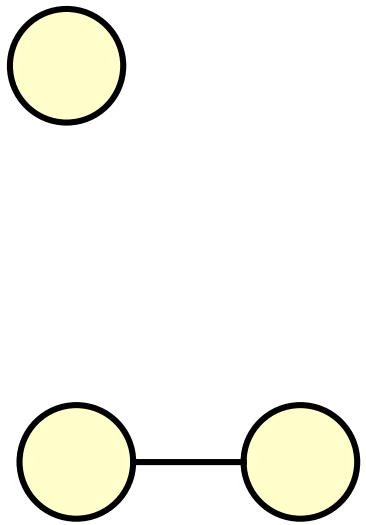


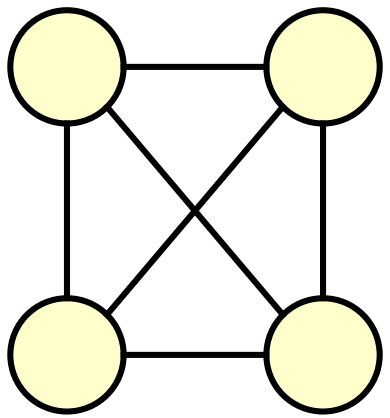
顶点覆盖与点独立集

- 断言 **VERTEX-COVER $\equiv_P$ INDEPENDENT-SET**
- 证明 S 是点独立集当且仅当 $V-S$ 是顶点覆盖
  - 必要性“$\Rightarrow$”
    - 设 S 为任一点独立集
    - 考虑任意边 (u, v)
    - 由点独立集的定义可得 $u\notin S$ 或 $v\notin S$，继而得到 $u\in V-S$ 或 $v\in V-S$
    - 因此，$V-S$ 覆盖边 (u, v)
  - 充分性“$\Leftarrow$”
    - 令 $V-S$ 为任一顶点覆盖
    - 考虑两个顶点 $u\in S$ 和 $v\in S$
    - 由于 $V-S$ 是顶点覆盖，因此 $(u, v)\notin E$
    - 因此，S 中任意两点之间都不存在边，于是 S 是点独立集

k-团

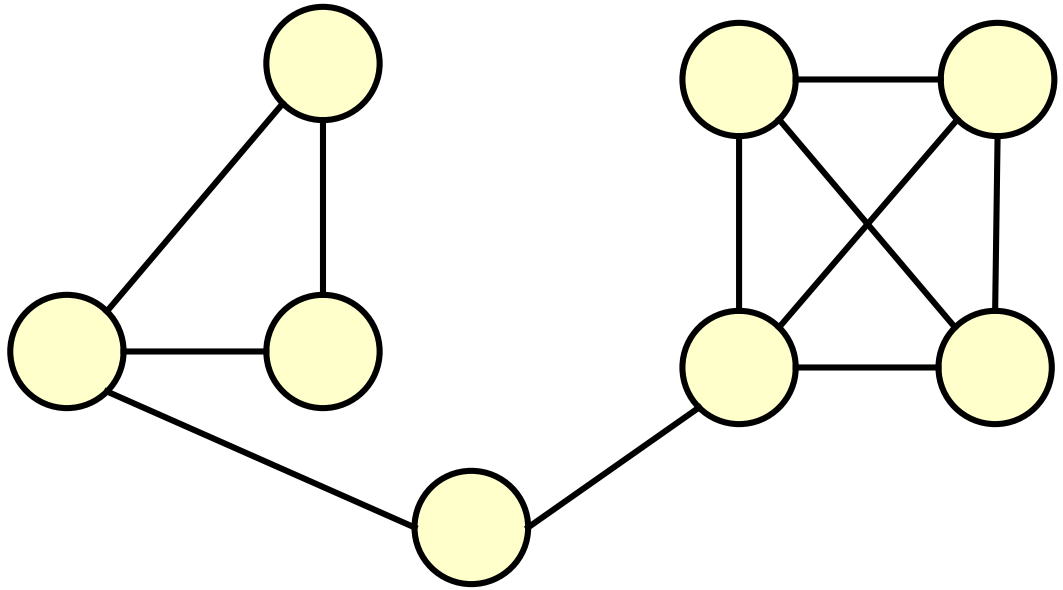
- ***k*-团（*k*-clique）是一组 *k* 个顶点，它们之间有 $k(k-1)/2$ 条边（即完全图 K_k）**





k-团

- 下图含有一个4-团作为子图



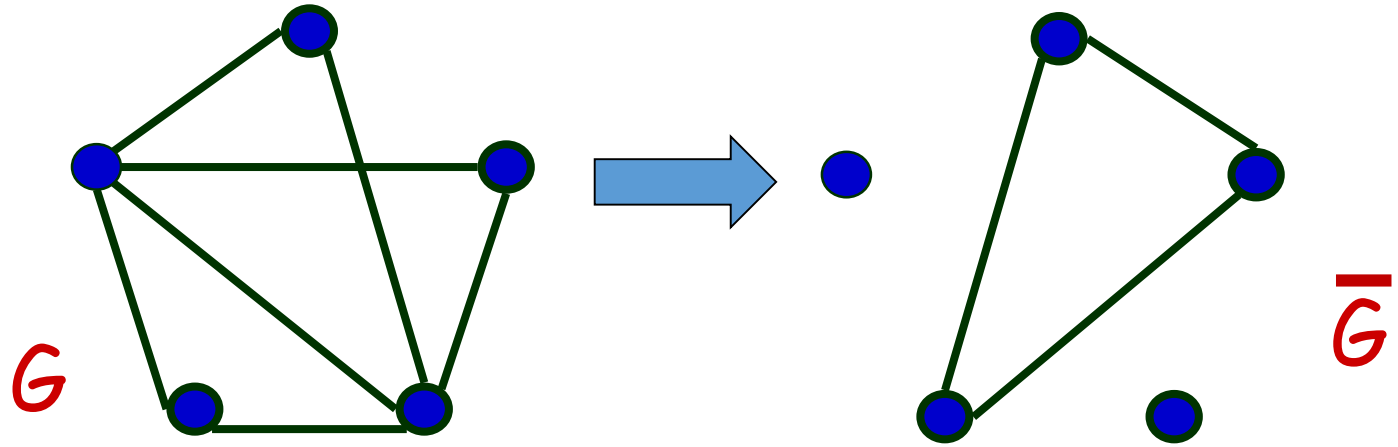
- **CLIQUE问题**
  - 给定图 ***G*** 和值 ***k***，判断图 ***G*** 中是否含有 ***k***-团作为子图

k-团与点独立集

- 断言

$$\textbf{INDEPENDENT-SET} \equiv_p \textbf{CLIQUE}$$

- 证明
  - S 是图 G 的一个大小为 k 的点独立集当且仅当S 是图 G 的补图 $\overline{G}$ 的一个k-团



集合覆盖

- 集合覆盖问题（SET-COVER）：给定一个基础集合 U、它的一些子集 $S_1, S_2, \ldots, S_m$ 以及整数 k，是否可以从中选取至多 k 个子集，使得它们的并集恰好是 U?
- 例

U = { 1, 2, 3, 4, 5, 6, 7 }

k = 2

S_1 = {3, 7}

S_2 = {3, 4, 5, 6}

S_3 = {1}

S_4 = {2, 4}

S_5 = {5}

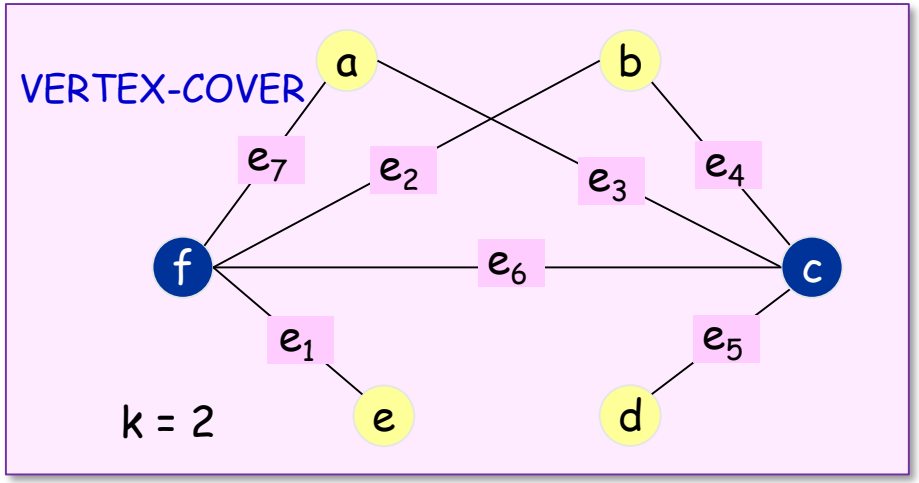
S_6 = {1, 2, 6, 7}

顶点覆盖归约至集合覆盖

- 断言：**VERTEX-COVER** $\leq_p$ **SET-COVER**
- 证明：
  - 给定 **VERTEX-COVER** 的一个实例 $G = (V, E)$ 和 k，可以构造一个集合覆盖实例

顶点覆盖归约至集合覆盖

- 构造方法：
  - 创建一个 **SET-COVER** 的实例：$k=k$，$U=E$，$S_v = \{e \in E : v \text{ 是 } e \text{ 的端点}\}$
  - 存在基数不超过 k 的集合覆盖当且仅当存在基数不超过 k 的顶点覆盖



SET-COVER

U = { 1, 2, 3, 4, 5, 6, 7 }
k = 2

$S_a = \{3, 7\}$　　$S_b = \{2, 4\}$
$S_c = \{3, 4, 5, 6\}$　　$S_d = \{5\}$
$S_e = \{1\}$　　$S_f = \{1, 2, 6, 7\}$

注记

- 传递性
  - 若 $X \leq_P Y$ 且 $Y \leq_P Z$，则 $X \leq_P Z$

- 示例：
  - **INDEPENDENT-SET $\leq_P$ VERTEX-COVER $\leq_P$ SET-COVER**
  - **VERTEX-COVER $\leq_P$ INDEPENDENT-SET $\leq_P$ CLIQUE**

千禧年大奖难题

(Millennium Prize Problems)

The P vs NP Question

千禧年大奖难题

- 千禧年大奖难题（Millennium Prize Problems），又称世界七大数学难题，是七个由美国克雷数学研究所（Clay Mathematics Institute, CMI）于2000年5月24日公布的数学猜想
- 根据克雷数学研究所订定的规则，任何一个猜想的解答，只要发表在数学期刊上，并经过两年的验证期，解决者就会被颁发一百万美元奖金

千禧年大奖难题

- http://www.claymath.org/millennium-problems
- https://www.claymath.org/millennium/p-vs-np/

P vs NP

If it is easy to check that a solution to a problem is correct, is it also easy to solve the problem? This is the essence of the P vs NP question. Typical of the NP problems is that of the Hamiltonian Path Problem: given N cities to visit, how can one do this without visiting a city twice? If you give me a solution, I can easily check that it is correct. But I cannot so easily find a solution.

课程中引用的图片素材来源于网络，仅供教学使用

P 类与 NP 类

P & NP

对问题的复杂性进行分类

是否存在多项式时间算法可以求解决此问题？

- 可能的回答：
  - 是的，存在
  - 不，不存在
    - 因为可以证明所有的算法都需要指数时间
    - 因为可以证明根本不存在解决这个问题的算法
  - 不知道。但如果能找到一个这样的算法，那么它将提供一种在多项式时间内求解许多其他问题的方法

缘起

- 重要的研究课题：对于一些著名的问题——例如哈密顿回路问题，证明其不存在多项式时间算法
  - 如果你能证明这一点，无疑你将获得图灵奖
  - 为了回答上述问题，人们定义了两类问题，**P类和NP类**

P 类与 *NP* 类

- 所有多项式时间可解的判定问题组成的问题类称作 **P** 类
- **P** 类：可在 $O(p(n))$ 时间求解的一类判定性问题，其中 $p(n)$ 是关于输入规模 n 的多项式
- 即
- **P** 类：有多项式时间算法的判定性问题

P 类与 NP 类

- **NP 类** 由多项式时间内可验证的判定问题组成
- 定义
  - 称算法 ***C*(*s*, *t*)** 是问题 ***X*** 的验证器（**certifier**），指的是对于每个实例 ***s***，该实例存在解当且仅当存在该实例的一个证据（**certificate/witness**）***t*** 使得***C*(*s*, *t*)** 输出为“是/ Yes”
  - 若一个判定性问题存在多项式时间验证器，则称该问题属于**NP**类

P 类与 NP 类

- **NP 类** 由多项式时间内可验证的判定问题组成
  - 粗略地讲：如果我们以某种方式得到了问题的一个实例（instance）和该实例的一个候选解，那么我们可以在（关于输入）的多项式时间内对该候选解的正确性进行验证
  - 示例——哈密顿回路问题：
    - 给定 n 个不同顶点构成的序列 $v_1, v_2, \ldots, v_n$，我们可以在$O(n)$（输入大小的多项式）时间内验证
      - 是否包含图中所有顶点
      - (v_i, v_{i+1})以及(v_n, v_1)是否是输入图 G 中的边

(*) NP 类

- **NP** 事实上代表非确定（nondeterministic）多项式时间
- **NP**：一类可以用非确定多项式（**NP**）算法求解的判定问题
  - 这些算法在任一步骤上都可以同时从多个可能动作中进行选择，并且这些动作都可能不依赖于先前的任何信息

可满足性（*Satisfiability*）问题

- 文字：布尔变量或其否定 x_i or $\overline{x_i}$
- 子句：文字的析取（析取式）$C_j = x_1 \vee \overline{x_2} \vee x_3$
- 合取范式（**CNF**）：子句的合取构成的命题公式Φ $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$
- SAT：给定CNF公式，判定是否存在成真指派

示例: $(\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (x_2 \vee x_3) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$

Yes： x_1 = 1/true, x_2 = 1/true, x_3 = 0/false

- 3-SAT：每个子句恰好包含3个文字的**SAT**

可满足性问题的验证器与证据

- 给定 n 个布尔变量的真值指派
- 检查公式中的每个子句，是否至少有一个为真的文字
- 例：

$$\left(\overline{x_1} \vee x_2 \vee x_3\right) \wedge \left(x_1 \vee \overline{x_2} \vee x_3\right) \wedge \left(x_1 \vee x_2 \vee x_4\right) \wedge \left(\overline{x_1} \vee \overline{x_3} \vee \overline{x_4}\right)$$

实例 s

$$x_1 = 1, x_2 = 1, x_3 = 0, x_4 = 1$$

证据 t

- 结论：SAT属于**NP**类

P类与NP类

- 粗略地讲
  - 如果你能在多项式时间内验证一个“解”是否正确，那么这个问题就属于**NP**类
  - 这是一个——或者说至少看起来是一个比能够在多项式时间内找到那个正确的解要宽松得多的条件

P 类与 NP 类

- “My favourite example here is a jigsaw puzzle.
- Solving the puzzle can be very hard, but if someone claims they've solved it, it usually takes no more than a quick glance to check whether they're right.
- To get a quantitative estimate of the running time, just look at each piece in turn and make sure that it fits the limited number of neighbours that adjoin it.
- The number of calculations required to do this is roughly proportional to the number of pieces, so the check runs in polynomial time.
- But you can't solve the puzzle that way. Neither can you try every potential solution in turn and check each as you go along, because the number of potential solutions grows much faster than any fixed power of the number of pieces. ”

——伊恩•斯图尔特（Ian Stewart）

P 类与 NP 类

- **P 类** 由所有多项式时间可解的判定问题组成
  - **P类**：存在多项式时间算法的判定问题
- **NP 类** 由多项式时间内可验证的判定问题组成
  - **NP类**：存在多项式时间验证器的判定问题
- **NP** 并非意为 “not **P**”

P 类与 NP 类

- 断言：**P** $\subseteq$ **NP**
- 定义
  - 称算法 ***C*(*s*, *t*)** 是问题 ***X*** 的验证器，指的是对于每个实例 ***s***，该实例存在解当且仅当存在该实例的一个证据 ***t*** 使得***C*(*s*, *t*)** 输出为“是/ Yes”
  - 若一个判定性问题存在多项式时间验证器，则称该问题属于**NP**类
- 若一个判定性问题属于**P**类，则存在多项式时间算法 ***A*(*s*)**
  - 验证器取为 ***C*(*s*, *t*) = *A*(*s*)**，***t*** 可取任一证据——例如空字符串

P 类与 NP 类

- 断言：$\mathbf{P} \subseteq \mathbf{NP}$
- **P = NP?**
  - 可能是不成立的
  - 但也可能成立的，因为
    - 目前也无人证明 $\mathbf{P} \neq \mathbf{NP}$
- **P = NP** 与否关系着大量问题是否存在多项式时间算法（快速算法）

NP 完全性理论

NP-Completeness

NP-困难与NP-完全

- 如果问题 Π 满足对于所有 $\Pi' \in NP$，都有 $\Pi' \leq_p \Pi$，则 Π 称作在多项式归约下是**NP-困难的（NP-hard）**
  - 此处并不要求 $\Pi \in NP$
- 如果问题 Π 是NP-困难的且 $\Pi \in NP$，则称它是**NP-完全的（NP-complete，NPC）**
  - 因此，这些是NP类中最困难的问题
  - 这些问题形成了多项式时间归约意义下的一个等价类

NP-完全问题

- **NPC**： Y 是一个NP问题，且对于任一NP问题 X，都有 $X \leq_P Y$
- **NP完全性（NP-completeness）**这一术语用来形容NP问题中最难的那些
  - 如果一个NPC问题有一个多项式时间算法，那么NP中的每个问题都有多项式时间算法

NP-完全问题

- 定理
  - 假设 Y 是一个NPC问题。则 Y 存在多项式时间算法当且仅当 P=NP
- 证明
  - “←”若 P=NP，则由于 Y 在 NP 中，所以 Y 在 P 中——即 Y 可以在多时间内求解
  - “→”若 Y 存在多项式时间算法，则
    - 设 X 是任一NP问题。由于 $X \leq_P Y$，因此可以在多项式时间内求解 X。这意味着 NP⊆P
    - 再由 P⊆NP 即得 P=NP
- 对 P≠NP 的研究的焦点可以放在NPC问题上

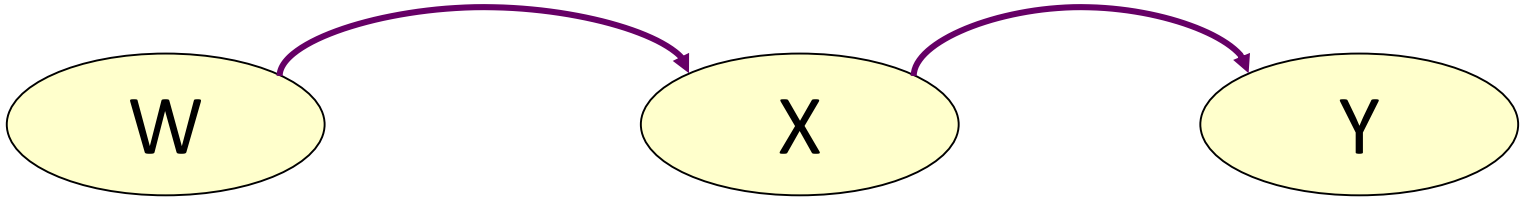
建立NP-完全性

- 一旦建立了第一个“自然的”NP完全问题，其他问题就会像推倒多米诺骨牌一样
- 建立问题 *Y* 的NP完全性的方法
  - 步骤1：证明 *Y* 是NP问题
  - 步骤2：选择一个NP完全问题 *X*
  - 步骤3：证明 $X \leq_P Y$

建立NP-完全性

- 如果 X 是一个NP完全问题，而 Y 是一个NP问题，且 $X \leq_P Y$，则 Y 是NP完全的
- 证明：
  - 设 W 是任一NP问题，则 $W \leq_P X \leq_P Y$
  - 由多项式归约的传递性可知 $W \leq_P Y$
  - 因此 Y 是NP完全的

由NP-完全性的定义　由上述假设

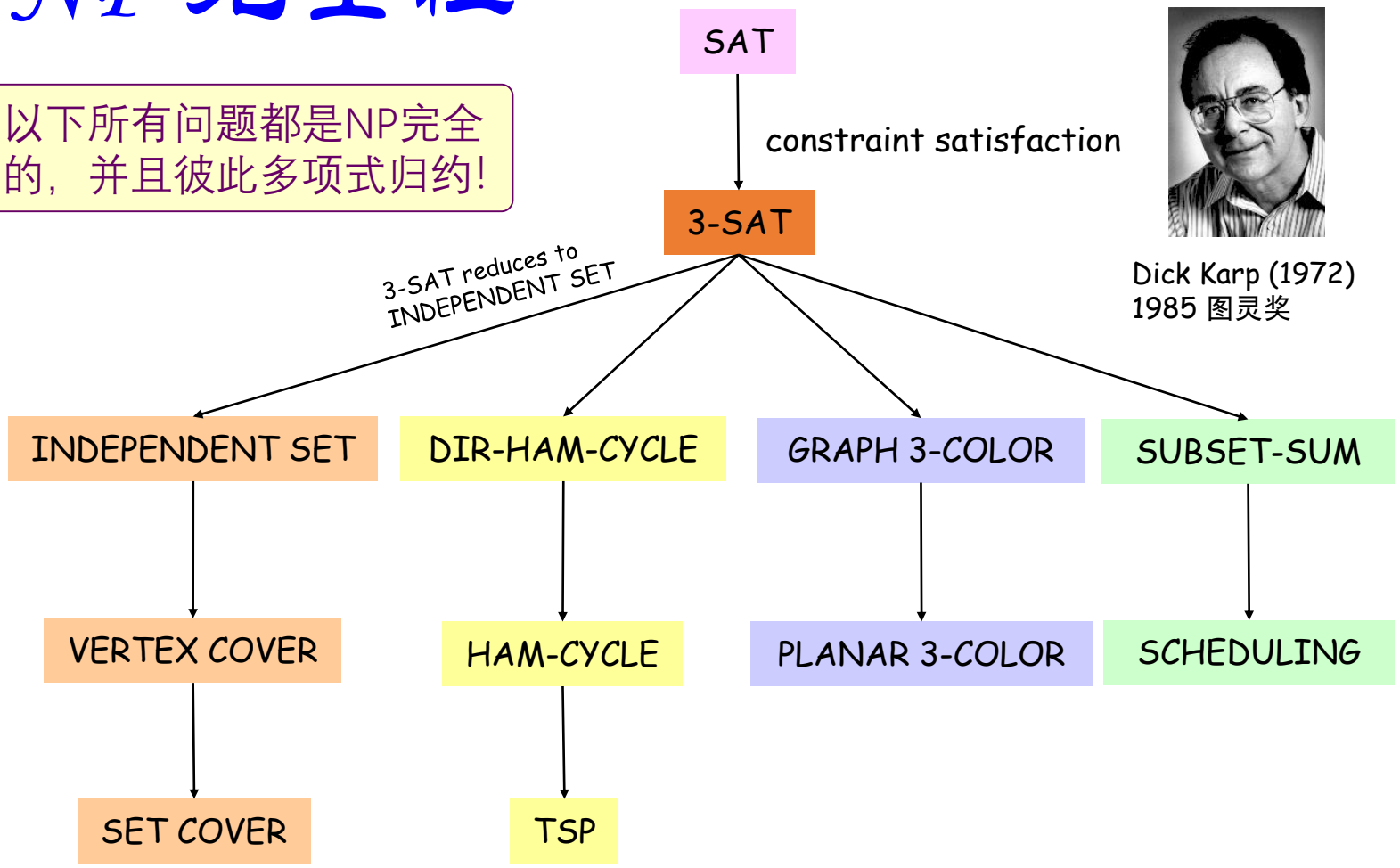


NP-完全问题

- 然而，此时出现了一个关键性问题：
  NPC问题是否确实存在？
- Cook-Levin定理[Cook 1971, Levin 1973]
  - 在多项式时间归约下，**CNF**可满足性问题是NP完全的
  - Cook于1982年获得图灵奖
- 对该问题的肯定回答导致了理论计算机科学中最引人入胜的领域之一的发展

NP-完全性

以下所有问题都是NP完全的，并且彼此多项式归约!



Dick Karp (1972)
1985 图灵奖

注记

- 在实践中，大多数NP问题要么是P问题，要么是NP完全问题
- 但也存在值得注意的例外情况
  - 例如因子分解，图同构等

应对NP-完全性

- 希望最差情况不要发生
  - 复杂性理论研究最差情况下的算法表现，但要解决的实例却可能是“很容易”的
- 让问题产生变化
  - 设计一个近似算法，即可以确保在多项式时间内找到高质量的解的算法
    - 活跃的研究领域，但并非总能做到——除非P=NP！
- 利用这种困难性
  - 例如密码学
- 继续努力证明 **P=NP**

End