数据分析与算法设计
第11章 算法能力的极限
（Limitations of Algorithm Power）

李旻
百人计划研究员
浙江大学 信息与电子工程学院
Email: min.li@zju.edu.cn

算法的能力

- 算法的能力并不是没极限的
  - 有些问题是无法用任何算法求解的
    - 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
  - 有些问题可以用算法求解，但是无法在多项式时间内获得答案
    - 汉诺塔问题
  - 有些问题可以在多项式的时间内算法求解，但往往局限于最优情况

算法的效率

- 对于给定的某个问题，以及给定的某个算法，我们可以分析其基本操作的执行次数来评估其算法的效率，确定其渐进的效率类型
- 对于给定的某个问题，我们也可以根据该问题的其它算法，来评估某个具体算法的效率如何（横向比较）
  - 需要知道该问题的任意算法可能具有的最佳效率（下界）
  - 如果当前算法和下界的效率类型相同，则该界是紧密的（下确界），即当前算法的改进空间仅为一个常量因子
  - 如果当前算法与下界之间还有不少差距，则仍可改进：要么存在一个匹配下界的更快算法，要么可证明一个更好的下界

目录

- 如何求下界
  - 平凡下界
  - 信息论下界
  - 敌手下界
  - 问题化简
- 决策树
  - 排序算法的决策树
  - 查找有序数组的决策树
- P、NP和NP完全问题
  - P和NP问题
  - NP完全问题

平凡下界

- 任何算法至少要“读取”所有它必须要处理的项，并“写出”它的全部输出
- 平凡下界(trivial lower bound)：对问题的输入中必须要处理的项进行计数，同时对必须要输出的项计数
  - 计算n次多项式：$p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_0$, 任意算法都要处理所有系数$a_n, , \ldots, a_0$, 平凡下界属于$\Omega(n)$（紧密的）
  - 任何生成n个不同项所有排列的算法其输出规模是n！，因此平凡下界属于$\Omega(n!)$（紧密的）
  - 计算两个n阶方阵乘积的算法的平凡下界属于$\Omega(n^2)$(尚未知道该界是否紧密)
- 平凡下界往往过小，因而用处不是非常大

信息论下界

- 信息论下界：根据算法必须处理的信息量来建立效率的下界
  - 比如，“猜”数字游戏：某人在1和n间选择了一个正整数，我们可以向他提问一些能够回答是或否的问题，来推断出该数。任何求解算法都必须解决一定数量的不确定信息，即为了在n种可能选择中确实某个数所需要的比特数$\lceil log_2 n \rceil$

- 许多涉及比较操作的问题，如排序和查找，信息论下界是非常用的
- 决策树的机制可更精确地实现信息论方法的内在思想

敌手下界

- 敌手下界：基于一种恶意而又诚实的敌手逻辑—恶意使它不断地把算法推向最消耗时间的路径，而诚实又迫使它必须和已做出的选择保持一致

- 举例：合并两个规模为n的有序列表

$$a_1 < a_2 < \cdots < a_n \quad \text{and} \quad b_1 < b_2 < \cdots < b_n$$

  - 该问题任何基于比较算法的键值比较次数的下界是2n−1

敌手下界

- 用敌手法证明前述结论
  - 敌手规则：当且仅当i<j时，对$a_i < b_j$的比较返回真
  - 跟这种规则一致，任何正确的合并算法只能产生一种合并列表：

$$b_1 < a_1 < b_2 < a_2 < \cdots < b_n < a_n$$

  - 为了生成这种合并列表，任何正确的算法不得不一次不差地进行2n−1次相邻元素对的比较，即b_1和a_1比，a_1和b_2比，以此类推
  - 因此，对于任何合并算法需要的键值比较次数来说，2n−1是一个下界

问题化简

- 假定问题Q的下界已知
- 欲证明问题P至少和问题Q一样复杂（Q的下界也是P的一个下界），则需把Q转化为P（而不是P转化为Q），即Q的任意实例都可以转化为P的一个实例，所以求解P的算法都可以用来求解Q

- 举例：两个对称矩阵的乘法(P)和两个任意方阵的乘法(Q)复杂性相同
  - P是Q的特例
  - Q可以转化为P

$$XY = \begin{bmatrix} 0 & A \\ A^T & 0 \end{bmatrix} \begin{bmatrix} 0 & B^T \\ B & 0 \end{bmatrix} = \begin{bmatrix} AB & 0 \\ 0 & A^T B^T \end{bmatrix}$$

问题化简

表 11.1　建立下界时，常在问题化简中使用的问题

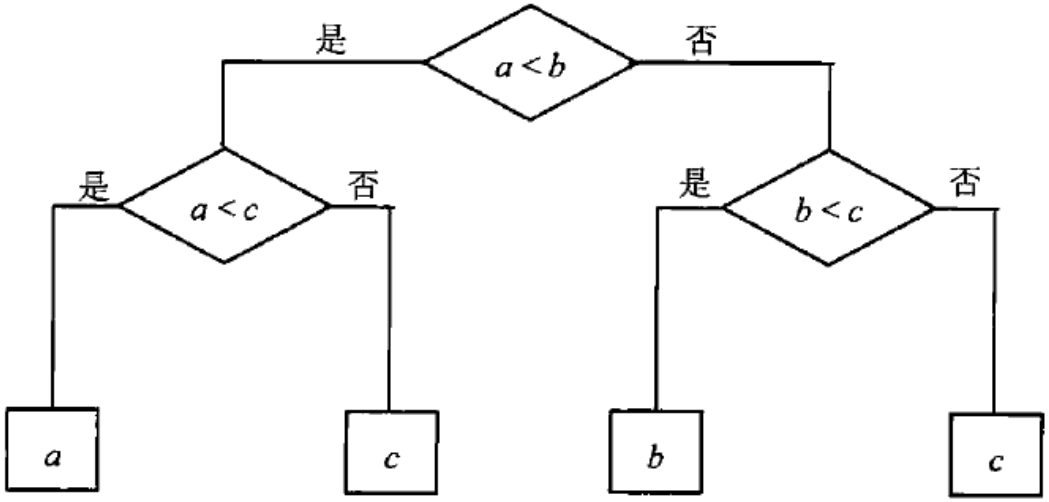
问　题	下　界	紧密与否
排序	$\Omega(n\log n)$	是
在有序数组中查找	$\Omega(\log n)$	是
元素唯一性问题	$\Omega(n\log n)$	是
n 位整数的乘法	$\Omega(n)$	未知
方阵的乘法	$\Omega(n^2)$	未知

目录

- 如何求下界
  - 平凡下界
  - 信息论下界
  - 敌手下界
  - 问题化简
- 决策树
  - 排序算法的决策树
  - 查找有序数组的决策树
- P、NP和NP完全问题
  - P和NP问题
  - NP完全问题

决策树

- 引例：求解a, b, c三个数中最小数的决策树



  - 每个叶子对应一种可能的输出
  - 叶子的数量必须至少和可能的输出数量一样多

决策树

- 一棵决策树如果具有给定数量的叶子，而这个数量是由可能的输出数量决定的，那么这棵树必须有足够的高度来容纳这些叶子

- 对于任何具有l个叶子，高度为h的二叉树，则

$$h \geq \lceil \log_2 l \rceil$$

- 该不等式给出了二叉决策树的一个下界，也给出了在最坏情况下，任何所讨论问题的基于比较的算法在比较次数上的下界

排序算法的决策树

- 对于基于比较的排序算法，可以通过研究它的决策树的特性，来分析算法在时间效率上的重要下界
- 一个排序算法的输出可以看成对一个输入列表的元素下标的某种排列。因此，对任意的n元素列表排列后，可能的输出的数量等于n!
- 因此，算法在最坏情况下的比较次数：

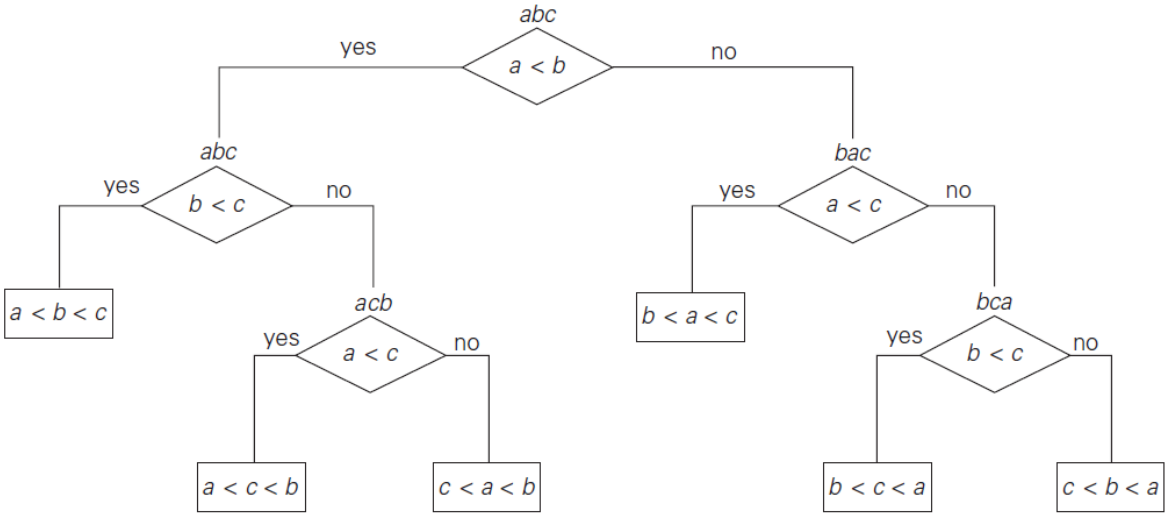
$$C_{worst}(n) \geq \lceil \log_2 n! \rceil .$$

$$\lceil \log_2 n! \rceil \approx \log_2 \sqrt{2\pi n}(n/e)^n = n \log_2 n - n \log_2 e + \frac{\log_2 n}{2} + \frac{\log_2 2\pi}{2} \approx n \log_2 n.$$

- 该下界是紧密的，合并排序的最差效率与该界吻合

排序算法的决策树

- 也可用决策树来分析基于比较的排序算法的平均性能
  - 举例：三个元素的插入排序算法，其决策树叶子的平均深度=平均比较次数=(2+3+3+2+3+3)/6

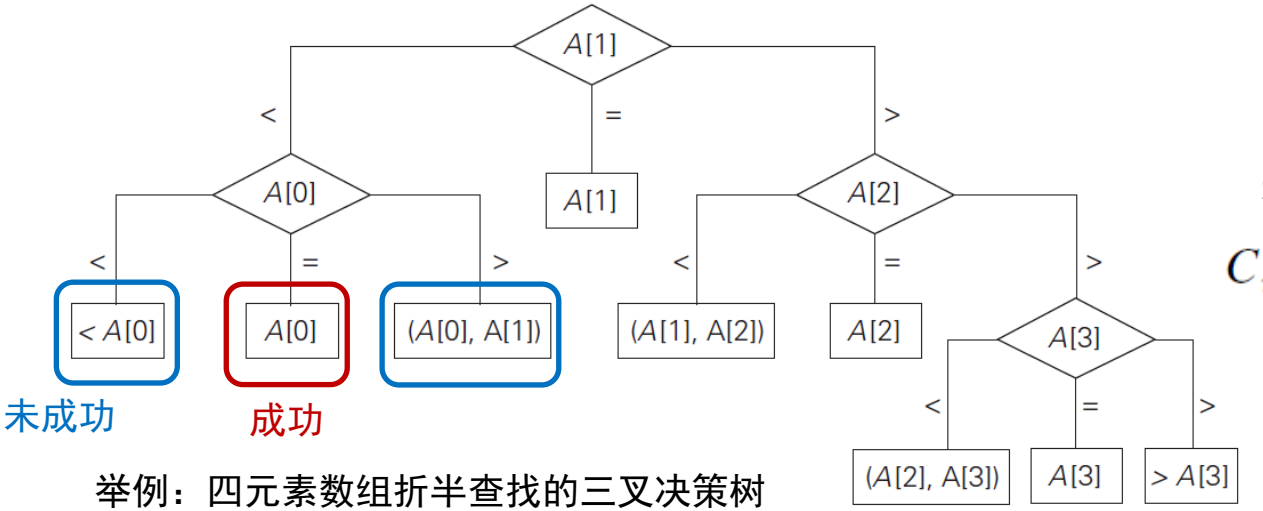


  - 对于一个n元素列表排序，算法的平均比较次数的下界是

$$C_{avg}(n) \geq \log_2 n!.$$

查找有序数组的决策树

- 对于包含n个键$A[0] < A[1] < \cdots < A[n-1]$的有序数组的查找问题，可以构建相应的决策树来确定键值比较次数的下界
- 主要考虑的算法是折半查找
- 一种直接的构造方式是三叉决策树



举例：四元素数组折半查找的三叉决策树

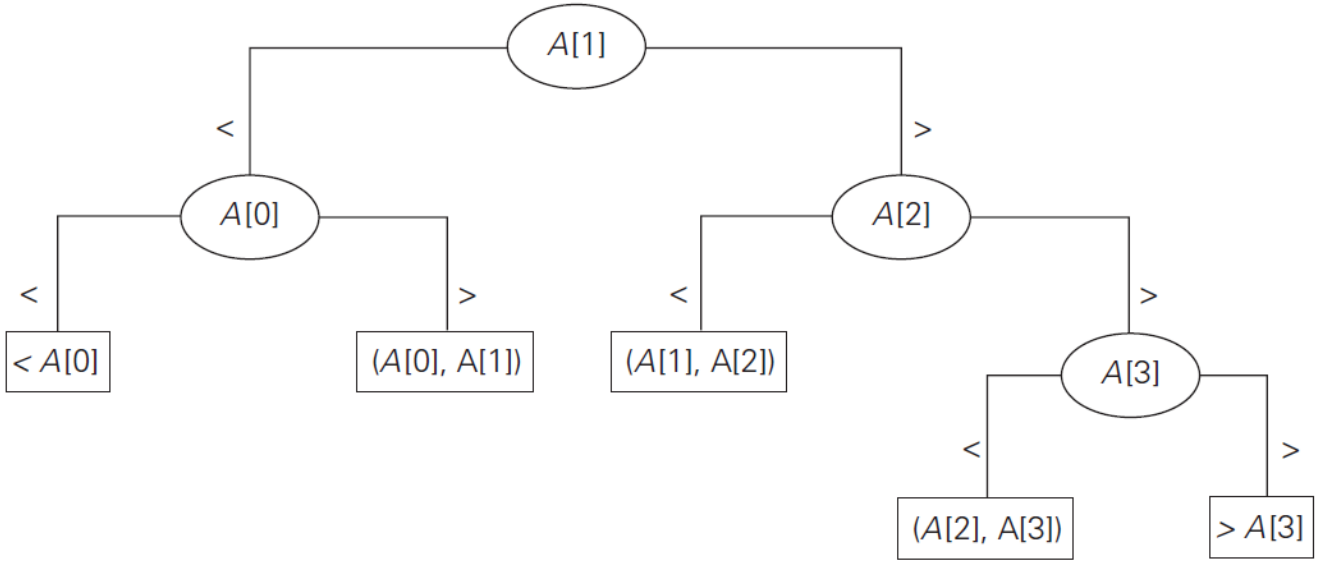
成功查找的叶子数：n
未成功查找的叶子数：n+1

$$C_{worst}(n) \geq \lceil \log_3(2n+1) \rceil.$$

（下界不紧密！）

查找有序数组的决策树

- 改进的构造方式是二叉决策树
  - 删除三叉树中的中间子树



$$C_{worst}(n) \geq \lceil \log_2(n+1) \rceil.$$ （下界紧密！）

目录

- 如何求下界
  - 平凡下界
  - 信息论下界
  - 敌手下界
  - 问题化简
- 决策树
  - 排序算法的决策树
  - 查找有序数组的决策树
- P、NP和NP完全问题
  - P和NP问题
  - NP完全问题

易解与难解的问题

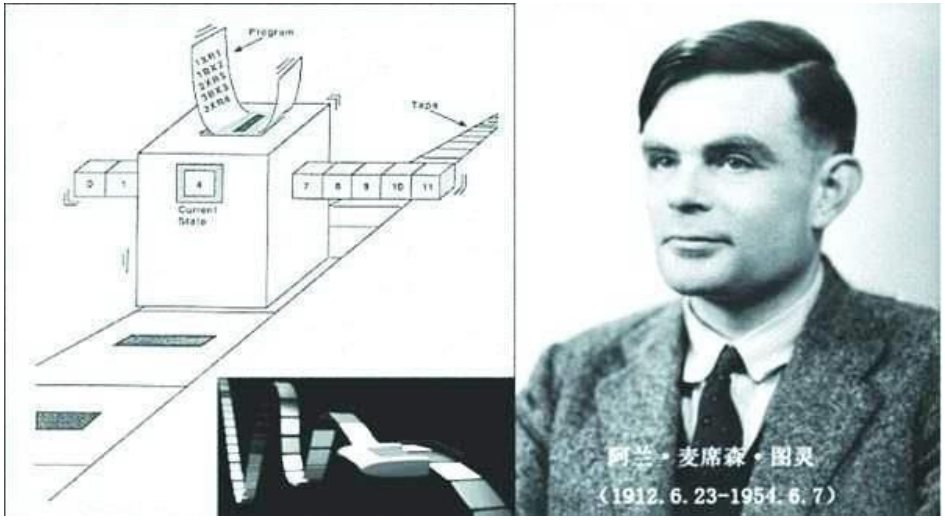
- 定义：假定$p(n)$是问题输入规模的一个多项式函数，如果一个算法的最差时间效率属于$O(p(n))$，我们说该算法能够在多项式时间内对问题求解
  - 注意：对数时间内求解的问题也属于$O(p(n))$
- 易解问题：可以在多项式时间内求解的问题
- 难解问题：不能在多项式时间内求解的问题
- 采用多项式时间来区分问题难易的原因
  - 无法保证能够在合理的时间内对难解问题的所有实例求解
  - 对于实用的多项式类型的算法，它们的多项式次数很少会大于3，作为算法下界的多项式，它们的系数也不会很大
  - 多项式函数有很多方便的数学特性
  - 可以发展出一种称为计算复杂性的理论

P类问题

- P类问题：是一类能够用（确定性的）算法在多项式的时间内求解的判定问题(decision problems)。这类问题类型也称为多项式类型
  - 查找、元素唯一性
  - 图的连通性、无环性等
- 把P类问题约束为判定问题的主要原因：
  - 排除不能在多项式时间内求解的问题，因为它们往往会产生指数级的输出
  - 虽然许多重要的问题以它们最自然的形式出现时并不是判定问题，但它们可以化简为一系列更容易研究的判定问题（如：图的着色问题）

不可判定问题

- 是否每个判定问题都可以在多项式时间内求解？否
- 某些判定问题无法用任何算法求解，这种问题称为不可判定问题（undecidable; 相对于能用算法解的可判定问题而言）
- 举例（停机问题）：给定一段计算机程序和一个输入，判断该程序对于该输入是会中止，还是会无限运行下去



不可判定问题

- 证明（停机问题）：反证法
  - 假设A是一个可求解该问题的算法，则对程序P和输入I有

$$A(P,I)=\begin{cases}1, & \text{如果程序}P\text{对于输入}I\text{会停机}\\ 0, & \text{如果程序}P\text{对于输入}I\text{不会停机}\end{cases}$$

  - 把程序P看成它自己的一个输入，则可构造如下的程序

$$Q(P)=\begin{cases}\text{停机}, & \text{如果}A(P,P)=0,\ \text{即程序}P\text{对于输入}P\text{不会停机}\\ \text{不停机}, & \text{如果}A(P,P)=1,\ \text{即程序}P\text{对于输入}P\text{会停机}\end{cases}$$

  - 用Q来替代P，得到

$$Q(Q)=\begin{cases}\text{停机}, & \text{如果}A(Q,Q)=0,\ \text{即程序}Q\text{对于输入}Q\text{不会停机}\\ \text{不停机}, & \text{如果}A(Q,Q)=1,\ \text{即程序}Q\text{对于输入}Q\text{会停机}\end{cases}$$

    - 产生矛盾，上述两种输出都不可能

另一些重要的问题

- 还有许多问题，既没有找到它们的多项式类型算法，也无法证明这样的算法不存在
  - **哈密顿回路问题** 确定一个给定的图中是否包含一条哈密顿回路(一条起止于相同顶点的路径，并且只经过其他所有顶点一次)。
  - **旅行商问题** 对于相互之间距离为已知正整数的 n 座城市，求最短的漫游路径(求一个权重为正整数的完全图的最短哈密顿回路)。
  - **背包问题** 对于 n 个重量和价值都为给定正整数的物品和一个承重量为给定正整数的背包，求这些物品中一个最有价值的子集，并且要能够装到背包中。
  - **划分问题** 给定 n 个正整数，判断是否能把它们划分成两个不相等的子集，并且和相等。
  - **装箱问题**(bin-packing problem) 给定 n 个物品，它们的大小都是不超过 1 的有理数，把它们装进数量最少的大小为 1 的箱子中。
  - **图的着色问题**(graph-coloring problem) 对于一个给定的图，求使得任何两个相邻顶点的颜色都不同时需要分配给图顶点的最少颜色数量。
  - **整数线性规划问题**(integer linear programming problem) 求一个线性函数的最大(或最小)值，函数包含若干个整数变量，并且满足线性等式和(或)不等式形式的有限约束。

另一些重要的问题

- 上述问题有些是判定问题，有些不是判定问题（自然形式出现）但可转化为等价的判定问题

- 共同点：
  - 都有着按照指数增长的候选项，其规模是输入规模的函数，需要在这些候选项中寻找问题的最终解
  - 虽然在计算上对问题求解可能是困难的，但是在计算上判定一个待定解是否解决了该问题却是简单的；这种判定可在多项式时间内完成

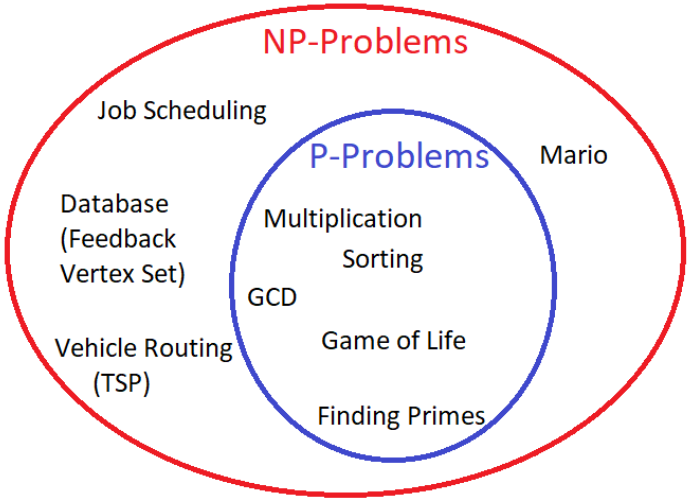
NP类问题

- NP类问题：是一类可以用不确定多项式算法求解的判定问题。这类问题类型也称为不确定多项式类型
  - 包含前述哈密顿回路问题、旅行商问题等组合问题
- 不确定算法：是一个两阶段的过程，把一个判定问题的实例l作为输入，并进行下面的操作
  - 非确定（“猜测”）阶段：生成一个任意串S，把它当作给定实例l的一个候选解
  - 确定（“验证”）阶段：确定性算法把l和S都作为它的输入，如果S的确是l的一个解，则输出“是”；如果S不是l的一个解，则要么输出“否”，要么停不下来
  - 不确定算法在验证阶段的时间效率是多项式级的，则我们说它是不确定多项式类型的

P与NP

- $P \subseteq NP$
  - 忽略在不确定阶段生成的串S，而采用确定多项式时间的算法求解
- 未解之谜：

$$P = NP?$$



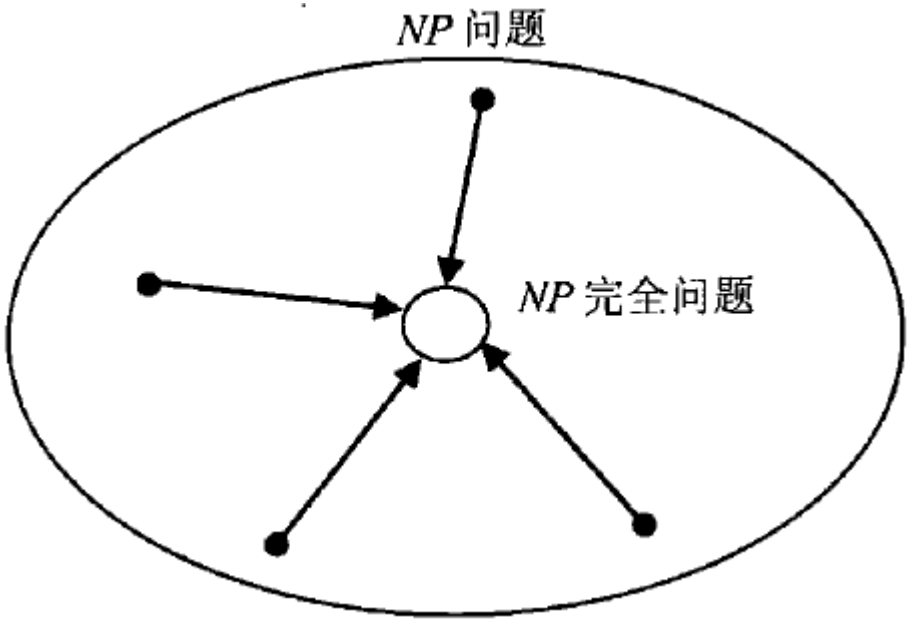
NP完全问题

- 定义：一个判定问题D_1可以多项式化简为一个判定问题D_2，条件是存在一个函数t能够把D_1的实例转化为D_2的实例，使得
  - t把D_1的所有真实例映射为D_2的真实例，把D_1的所有假实例映射为D_2的假实例
  - t可以用一个多项式算法计算

- 上述定义意味着如果D_1可以多项式化简为某些能够在多项式时间内求解的问题D_2，则问题D_1就可以在多项式时间内求解

NP完全问题

- NP完全问题：一个判定问题D是NP完全问题，条件是
  - 它属于NP类型
  - NP中的任何问题都能够在多项式时间内化简为D



NP完全问题

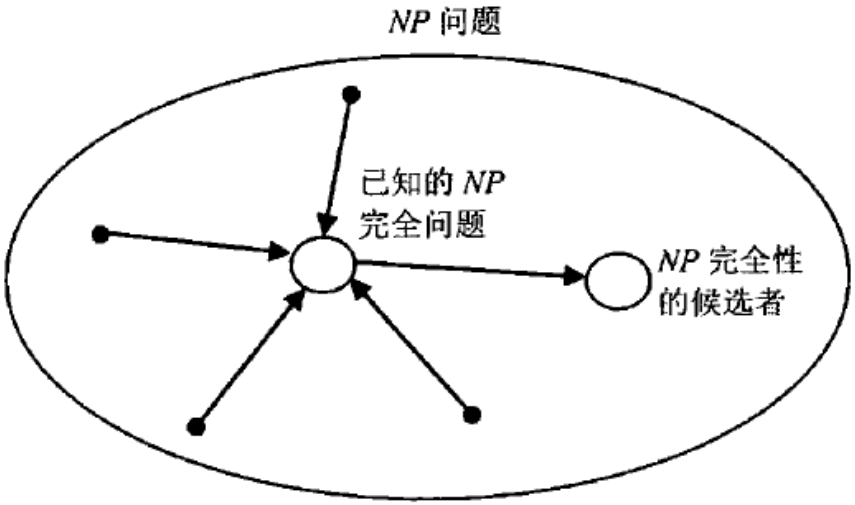
- 第一个NP完全问题
  - 由美国的Stephen Cook和苏联的Leonid Levin分别独立发现
  - 合取范式可满足性问题
    - 它与布尔表达式有关。每个布尔表达式都能被表示成合取范式，如

$$(x_1 \vee \overline{x}_2 \vee \overline{x}_3) \& (\overline{x}_1 \vee x_2) \& (\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3)$$

    - 合取范式可满足性：是否可以把真或假赋给一个给定的合取范式类型的布尔表达式中的变量，使得整个表达式都为真
- 后续发现了几百种其它的NP完全问题，包括前述的哈密顿回路问题、旅行商问题、划分问题等

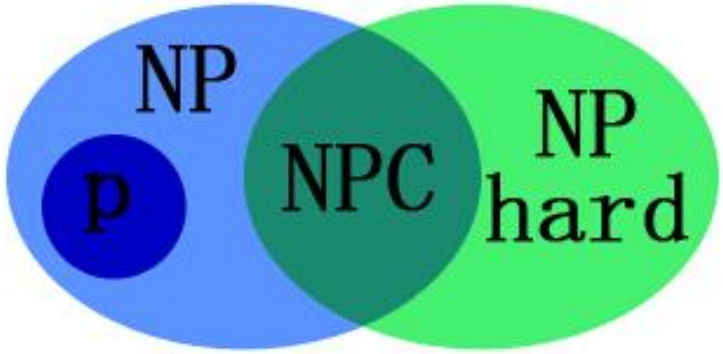
NP完全问题

- 基于已知的NP完全问题，我们可以采用如下的步骤证明一个新的判定问题是NP完全问题
  - 步骤1：证明该问题是NP问题
  - 步骤2：证明一个已知的NP完全问题能够在多项式时间内转化为所讨论的问题



NP难问题

- NP难问题：一个判定问题D是NP难问题，条件是NP中的任何问题都能在多项式时间内化简为D，但它不一定是NP类问题
  - 如：停机问题



课后作业

章 X	节 X. Y	课后作业题 Z	思考题 Z
11	11.1	3	1,2,12
	11.2	3,6	9,10,11
	11.3	2,11	12

注：只需上交“课后作业题”；以“学号姓名_chX.pdf ”规范命名，提交到“学在浙大”指定文件夹。DDL：2024年5月14日