3 Complexity
计算复杂性

杨雅君

yjyang@tju.edu.cn

智能与计算学部

2022

Outline

1. 度量复杂性
2. P 类
3. NP 类
4. NP 完全性
5. 几个 NP 完全问题
6. NP 难问题

Outline

1. 度量复杂性
   - 大 O 和小 o 记法
   - 分析算法
   - 模型间的复杂性关系
2. P 类
3. NP 类
4. NP 完全性
5. 几个 NP 完全问题

度量复杂性

考察递归语言 $A = \{0^k 1^k | k \geqslant 0\}$, 单带图灵机需要多少时间来判定 A?

度量复杂性

考察递归语言 $A = \{0^k 1^k | k \geqslant 0\}$, 单带图灵机需要多少时间来判定 A?

Example (判定语言 A 的图灵机 M_1)

$M_1 =$ 对输入串 w

1. 扫描纸带, 如果在 1 的右边发现 0, 就拒绝
2. 如果纸带上既有 0 也有 1, 就重复下一步
3. 扫描纸带, 删除一个 0 和一个 1.
4. 如果所有 1 都被删除以后还有 0, 或者所有 0 都被删除以后还有 1, 就拒绝. 否则, 如果在纸带上既没有剩下 0 也没有剩下 1, 就接受.

度量复杂性

本章节我们将学会分析判定 A 的图灵机 M_1 的算法所需的时间. 在一个特定的输入上, 算法所使用的步数可能与若干参数有关. 下面给出时间复杂度的定义:

度量复杂性

本章节我们将学会分析判定 A 的图灵机 M_1 的算法所需的时间. 在一个特定的输入上, 算法所使用的步数可能与若干参数有关. 下面给出时间复杂度的定义:

Definition (时间复杂度)

令 M 是一个总停机的确定型图灵机. M 的**运行时间** (running time) 或者**时间复杂度** (time complexity) 是一个函数 $f : \mathbf{N} \to \mathbf{N}$, 其中 N 是非负整数集合. $f(n)$ 是 M 在所有长度为 n 的输入上运行时所经过的最大步数. 若 $f(n)$ 是 M 的运行时间, 则称 M 在时间 $f(n)$ 内运行, M 是 $f(n)$ 的时间图灵机. 通常使用 n 表示输入的规模.

大 O 和小 o 记法

因为算法的精确运行时间通常是一个复杂的表达式, 所以一般只是估计它的趋势和级别.

大 O 和小 o 记法

因为算法的精确运行时间通常是一个复杂的表达式, 所以一般只是估计它的趋势和级别.

- 函数 $f(n) = 6n^3 + 2n^2 + 20n + 45$, 称 f 渐近地不大于 n^3.

大 O 和小 o 记法

因为算法的精确运行时间通常是一个复杂的表达式, 所以一般只是估计它的趋势和级别.

- 函数 $f(n) = 6n^3 + 2n^2 + 20n + 45$, 称 f 渐近地不大于 n^3.

这种关系的表达方式被称为**渐近记法或大 O 记法**, 记为 $f(n) = O(n^3)$

Definition (上界)

设 f 和 g 是两个函数 $f, g : \mathbf{N} \to \mathbf{R}^+$. 称 $f(n) = O(g(n))$, 若存在正整数 c 和 n_0, 使得对所有 $n \geqslant n_0$ 有

$$f(n) \leqslant cg(n)$$

当 $f(n) = O(g(n))$ 时, 称 $g(n)$ 是 $f(n)$ 的上界 (upper bound), $g(n)$ 是 $f(n)$ 的渐近上界, 以强调没有考虑常数因子.

大 O 和小 o 记法

Example

设 $f_1(n)$ 是函数 $5n^3 + 2n^2 + 22n + 6$. 保留最高次项 $5n^3$, 并且舍去它的系数 5, 得到 $f_1(n) = O(n^3)$.

验证一下这个结果是否满足上面的的形式化定义.

大 O 和小 o 记法

Example

设 $f_1(n)$ 是函数 $5n^3 + 2n^2 + 22n + 6$. 保留最高次项 $5n^3$, 并且舍去它的系数 5, 得到 $f_1(n) = O(n^3)$.

验证一下这个结果是否满足上面的的形式化定义.

令 $c = 6$, $n_0 = 10$, 则对于所有 $n \geqslant 10$, 有 $5n^3 + 2n^2 + 22n + 6 \leqslant 6n^3$.

大 O 和小 o 记法

Example

设 $f_1(n)$ 是函数 $5n^3 + 2n^2 + 22n + 6$. 保留最高次项 $5n^3$, 并且舍去它的系数 5, 得到 $f_1(n) = O(n^3)$.

验证一下这个结果是否满足上面的的形式化定义.

令 $c = 6$, $n_0 = 10$, 则对于所有 $n \geqslant 10$, 有 $5n^3 + 2n^2 + 22n + 6 \leqslant 6n^3$.

此外, $f_1(n) = O(n^4)$, 因为 n^4 比 n^3 大, 所以它也是 f_1 的一个渐近上界.

大 O 和小 o 记法

Example

设 $f_1(n)$ 是函数 $5n^3 + 2n^2 + 22n + 6$. 保留最高次项 $5n^3$, 并且舍去它的系数 5, 得到 $f_1(n) = O(n^3)$.

验证一下这个结果是否满足上面的的形式化定义.

令 $c = 6$, $n_0 = 10$, 则对于所有 $n \geqslant 10$, 有 $5n^3 + 2n^2 + 22n + 6 \leqslant 6n^3$.

此外, $f_1(n) = O(n^4)$, 因为 n^4 比 n^3 大, 所以它也是 f_1 的一个渐近上界.

但是, $f_1(n) \neq O(n^2)$, 无论 c 和 n_0 如何赋值, 始终无法满足定义.

大 O 和小 o 记法

Example

设 $f_1(n)$ 是函数 $5n^3 + 2n^2 + 22n + 6$. 保留最高次项 $5n^3$, 并且舍去它的系数 5, 得到 $f_1(n) = O(n^3)$.

验证一下这个结果是否满足上面的的形式化定义.

令 $c = 6$, $n_0 = 10$, 则对于所有 $n \geqslant 10$, 有 $5n^3 + 2n^2 + 22n + 6 \leqslant 6n^3$.

此外, $f_1(n) = O(n^4)$, 因为 n^4 比 n^3 大, 所以它也是 f_1 的一个渐近上界.

但是, $f_1(n) \neq O(n^2)$, 无论 c 和 n_0 如何赋值, 始终无法满足定义.

Example

因 $\log_b n = \frac{\log_2 n}{\log_2 b}$, 故 $f(n) = O(\log n)$ 不必指明基数, 因为忽略常数因子.

令 $f_2(n) = 3n \log_2 n + 5n \log_2 \log_2 n + 2$, 此时有 $f_2(n) = O(n \log n)$.

大 O 和小 o 记法

与大 O 记法相伴的有小 o 记法. 大 O 记法指一个函数渐近地不大于另一个函数. 要说一个函数渐近地小于另一个函数, 则用小 o 记法.

大 O 和小 o 记法

与大 O 记法相伴的有小 o 记法. 大 O 记法指一个函数渐近地不大于另一个函数. 要说一个函数渐近地小于另一个函数, 则用小 o 记法. 大 O 记法与小 o 记法的区别类似于 $\leqslant$ 和 $<$ 之间的区别.

Definition (上界)

设 f 和 g 是两个函数 $f, g : \mathbf{N} \to \mathbf{R}^+$. 如果

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$$

则称 $f(n) = o(g(n))$. 换言之, $f(n) = o(g(n))$ 意味着对于任何实数 $c > 0$, 存在一个数 n_0, 使得对所有的 $n \geqslant n_0$, $f(n) < cg(n)$.

大 O 和小 o 记法

容易验证下面的等式

Example

1. $\sqrt{n} = o(n)$
2. $n = o(n \log(\log n))$
3. $n \log(\log n) = o(n \log n)$
4. $n \log n = o(n^2)$
5. $n^2 = o(n^3)$

大 O 和小 o 记法

容易验证下面的等式

Example

1. $\sqrt{n} = o(n)$
2. $n = o(n \log(\log n))$
3. $n \log(\log n) = o(n \log n)$
4. $n \log n = o(n^2)$
5. $n^2 = o(n^3)$

但是, 以上例子中 $f(n)$ 不会等于 $o(f(n))$.

分析算法

Example (判定语言 A 的图灵机 M_1)

$M_1 =$ 对输入串 w

1. 扫描纸带, 如果在 1 的右边发现 0, 就拒绝
2. 如果纸带上既有 0 也有 1, 就重复下一步
3. 扫描纸带, 删除一个 0 和一个 1.
4. 如果所有 1 都被删除以后还有 0, 或者所有 0 都被删除以后还有 1, 就拒绝. 否则, 如果在纸带上既没有剩下 0 也没有剩下 1, 就接受.

分析算法

Example (判定语言 A 的图灵机 M_1)

$M_1 =$ 对输入串 w

1. 扫描纸带, 如果在 1 的右边发现 0, 就拒绝
2. 如果纸带上既有 0 也有 1, 就重复下一步
3. 扫描纸带, 删除一个 0 和一个 1.
4. 如果所有 1 都被删除以后还有 0, 或者所有 0 都被删除以后还有 1, 就拒绝. 否则, 如果在纸带上既没有剩下 0 也没有剩下 1, 就接受.

为了分析 M_1, 把他的四个步骤分开来考虑

分析算法

为了分析 M_1, 把他的四个步骤分开来考虑:

分析算法

为了分析 M_1, 把他的四个步骤分开来考虑:

Example (分析图灵机 M_1)

1. 步骤 1: 扫描纸带验证输入合法, 执行此次扫描需要 n 步. 将读写头重新放置在纸带左端需要另外 n 步. 所以这一步骤共需 $2n$ 步, 用大 O 记法为 $O(n)$ 步.

分析算法

为了分析 M_1, 把他的四个步骤分开来考虑:

Example (分析图灵机 M_1)

1. 步骤 1: 扫描纸带验证输入合法, 执行此次扫描需要 n 步. 将读写头重新放置在纸带左端需要另外 n 步. 所以这一步骤共需 $2n$ 步, 用大 O 记法为 $O(n)$ 步.
2. 步骤 2 和步骤 3: 机器反复扫描纸带, 每次扫描中删除一个 0 和一个 1. 每次扫描需 n 步. 因为每次扫描删除两个符号, 所以至多扫描 $\frac{n}{2}$ 次, 故步骤 2 和步骤 3 需要的全部时间为 $\frac{n}{2}O(n) = O(n^2)$.

分析算法

为了分析 M_1, 把他的四个步骤分开来考虑:

Example (分析图灵机 M_1)

1. 步骤 1: 扫描纸带验证输入合法, 执行此次扫描需要 n 步. 将读写头重新放置在纸带左端需要另外 n 步. 所以这一步骤共需 $2n$ 步, 用大 O 记法为 $O(n)$ 步.
2. 步骤 2 和步骤 3: 机器反复扫描纸带, 每次扫描中删除一个 0 和一个 1. 每次扫描需 n 步. 因为每次扫描删除两个符号, 所以至多扫描 $\frac{n}{2}$ 次, 故步骤 2 和步骤 3 需要的全部时间为 $\frac{n}{2}O(n) = O(n^2)$.
3. 步骤 4: 机器扫描一次决定接受或拒绝, 此步骤需时至多是 $O(n)$.

分析算法

为了分析 M_1, 把他的四个步骤分开来考虑:

Example (分析图灵机 M_1)

1. 步骤 1: 扫描纸带验证输入合法, 执行此次扫描需要 n 步. 将读写头重新放置在纸带左端需要另外 n 步. 所以这一步骤共需 $2n$ 步, 用大 O 记法为 $O(n)$ 步.
2. 步骤 2 和步骤 3: 机器反复扫描纸带, 每次扫描中删除一个 0 和一个 1. 每次扫描需 n 步. 因为每次扫描删除两个符号, 所以至多扫描 $\frac{n}{2}$ 次, 故步骤 2 和步骤 3 需要的全部时间为 $\frac{n}{2}O(n) = O(n^2)$.
3. 步骤 4: 机器扫描一次决定接受或拒绝, 此步骤需时至多是 $O(n)$.

M_1 在规模为 n 的输入上的运行时间为 $O(n) + O(n^2) + O(n) = O(n^2)$.

分析算法

Definition (时间复杂性类)

令 $t : \mathbf{N} \to \mathbf{R}^+$ 是一个函数. 定义**时间复杂性类** TIME$(t(n))$ 为由 $O(t(n))$ 时间的图灵机判定的所有语言的集合

分析算法

Definition (时间复杂性类)

令 $t : \mathbf{N} \to \mathbf{R}^+$ 是一个函数. 定义**时间复杂性类** TIME$(t(n))$ 为由 $O(t(n))$ 时间的图灵机判定的所有语言的集合

回忆语言 $A = \{0^k 1^k | k \geqslant 0\}$. M_1 在时间 $O(n^2)$ 内判定 A, 因为 TIME(n^2) 包括所有在时间 $O(n^2)$ 内可以判定的语言, 故 $A \in$ TIME(n^2).

分析算法

Definition (时间复杂性类)

令 $t : \mathbf{N} \to \mathbf{R}^+$ 是一个函数. 定义**时间复杂性类** TIME$(t(n))$ 为由 $O(t(n))$ 时间的图灵机判定的所有语言的集合

回忆语言 $A = \{0^k 1^k | k \geqslant 0\}$. M_1 在时间 $O(n^2)$ 内判定 A, 因为 TIME(n^2) 包括所有在时间 $O(n^2)$ 内可以判定的语言, 故 $A \in$ TIME(n^2).

是否存在渐进更快地判定语言 A 的图灵机呢?

分析算法

M_2 采用不同的方法, 可以渐近更快地判定 A. 它表明 $A \in \mathsf{TIME}(n \log n)$

分析算法

M_2 采用不同的方法, 可以渐近更快地判定 A. 它表明 $A \in \mathsf{TIME}(n \log n)$

Example

M_2 对输入串 w:

1. 扫描纸带, 如果在 1 的右边发现 0, 就**拒绝**.
2. 只要在纸带上还有 0 和 1 , 就重复下面的步骤.
3. 扫描纸带, 检查剩余的 0 和 1 的总数是偶数还是奇数. 若是奇数, 就**拒绝**.
4. 再次扫描纸带, 从第一个 0 开始, 隔一个删除一个 0, 然后从第一个 1 开始, 隔一个删除一个 1 .
5. 如果纸带上不再有 0 和 1 , 就**接受**. 否则, **拒绝**.

分析算法

如果图灵机有第二条纸带, 就可以在 $O(n)$ 时间 (也称为线性时间) 内判定语言 A.

分析算法

如果图灵机有第二条纸带, 就可以在 $O(n)$ 时间 (也称为线性时间) 内判定语言 A.

Example

M_3 对输入串 w:

1. 扫描纸带 1, 如果在 1 的右边发现 0, 就**拒绝**.
2. 扫描纸带 1 上的 0, 直到第一个 1 时停止, 同时把 0 复制到纸带 2 上.
3. 扫描纸带 1 上的 1 直到输入的末尾. 每次从纸带 1 上读到一个 1, 就在纸带 2 上删除一个 0, 如果在读完 1 之前所有的 0 都被删除, 就**拒绝**.
4. 如果所有的 0 都被删除, 就**接受**. 如果还有 0 剩下, 就**拒绝**.

模型间的复杂性关系

考察三种模型：单带图灵机，多带图灵机和非确定型图灵机

模型间的复杂性关系

考察三种模型：单带图灵机, 多带图灵机和非确定型图灵机

Theorem

设 $t(n)$ 是一个函数, $t(n) \geqslant n$. 则每一个 $t(n)$ 时间的多带图灵机都和某一个 $O(t^2(n))$ 时间的单带图灵机等价.

模型间的复杂性关系

考察三种模型：单带图灵机, 多带图灵机和非确定型图灵机

Theorem

设 $t(n)$ 是一个函数, $t(n) \geqslant n$. 则每一个 $t(n)$ 时间的多带图灵机都和某一个 $O(t^2(n))$ 时间的单带图灵机等价.

Definition (非确定型图灵机的运行时间)

设 N 是非确定型图灵机, 且是判定器. N 的运行时间是函数 $f : \mathbf{N} \to \mathbf{N}$, 其中 $f(n)$ 是在任何长度为 n 的输入上所有分支中的最大步数.

模型间的复杂性关系

考察三种模型：单带图灵机, 多带图灵机和非确定型图灵机

Theorem

设 $t(n)$ 是一个函数, $t(n) \geqslant n$. 则每一个 $t(n)$ 时间的多带图灵机都和某一个 $O(t^2(n))$ 时间的单带图灵机等价.

Definition (非确定型图灵机的运行时间)

设 N 是非确定型图灵机, 且是判定器. N 的运行时间是函数 $f : \mathbf{N} \to \mathbf{N}$, 其中 $f(n)$ 是在任何长度为 n 的输入上所有分支中的最大步数.

Theorem

设 $t(n)$ 是一个函数, $t(n) \geqslant n$. 则每一个 $t(n)$ 时间的非确定型图灵机都和某一个 $2^{O(t(n))}$ 时间的单带图灵机等价.

Outline

1 度量复杂性

2 P 类
- 多项式时间
- P 中的问题举例

3 NP 类

4 NP 完全性

5 几个 NP 完全问题

多项式时间

运行时间的多项式差异可以认为是较小的, 而是指数差异被认为是大的.

多项式时间

运行时间的多项式差异可以认为是较小的, 而是指数差异被认为是大的.

说明:

所有合理的确定型计算模型都是**多项式等价的**, 也就是说, 它们中任何一个模型都可以模拟另一个, 而运行时间只增长多项式倍.

多项式时间

运行时间的多项式差异可以认为是较小的, 而是指数差异被认为是大的.

说明:

所有合理的确定型计算模型都是**多项式等价的**, 也就是说, 它们中任何一个模型都可以模拟另一个, 而运行时间只增长多项式倍.

现在给出复杂性理论中的一个重要定义.

Definition (P 类)

P 是确定型单带图灵机在多项式时间内可判定的语言类. 换言之,

$$P = \bigcup_k \mathsf{TIME}(n^k)$$

多项式时间

在复杂性理论中，P 类扮演核心的角色，它的重要性在于：

1. 对于所有与确定型单带图灵机多项式等价的计算模型来说，P 是不变.
2. P 大致对应于在计算机上实际可解的那一类问题.

多项式时间

在复杂性理论中, P 类扮演核心的角色, 它的重要性在于:

1. 对于所有与确定型单带图灵机多项式等价的计算模型来说, P 是不变.
2. P 大致对应于在计算机上实际可解的那一类问题.

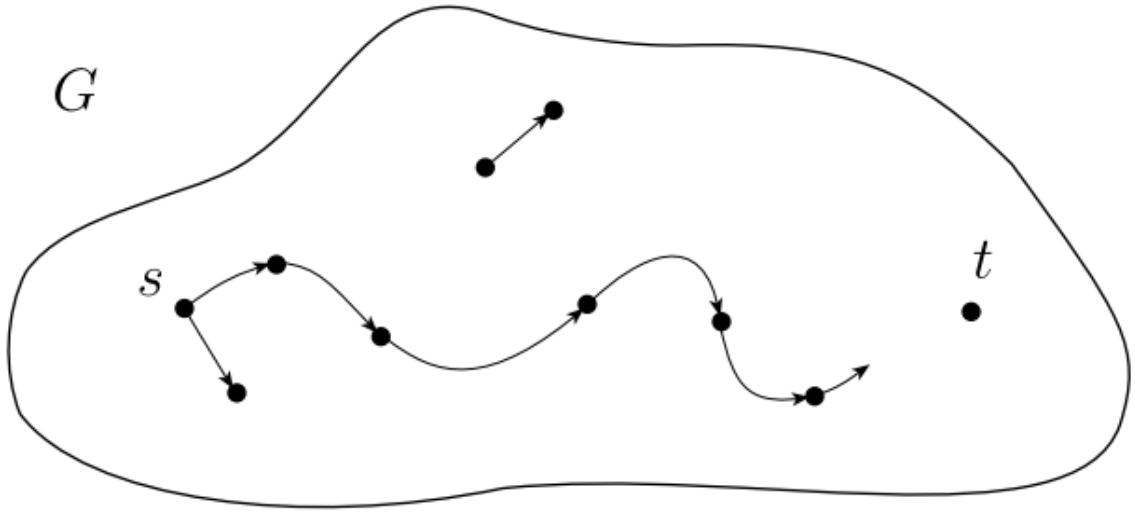
说明:

- 第 1 条表明, 在数学上, P 是一个稳健的类, 它不受所采用的的具体计算模型的影响.
- 第 2 条表明, 从实用的观点看, P 是恰当的. 当一个问题在 P 中的时候, 就有办法在时间 n^k (k 是常数) 内求解它.

P 中的问题举例

有向图 G 包含节点 s 和 t, 如下图所示, PATH 问题就是要确定是否存在从 s 到 t 的有向路径. 令

$$PATH = \{\langle G, s, t\rangle | G \text{ 是具有从 } s \text{ 到 } t \text{ 的有向路径的有向图}\}$$



P 中的问题举例

Theorem

$PATH \in P$

P 中的问题举例

Theorem

$PATH \in P$

证明：*PATH* 的一个多项式时间算法 M 运行如下:

P 中的问题举例

Theorem

$PATH \in P$

证明：*PATH* 的一个多项式时间算法 M 运行如下:

M 对输入 $\langle G, s, t\rangle$, G 是包含结点 s 和 t 的有向图：

1. 在结点 s 上做标记.
2. 重复下面步骤 3, 直到不再有结点被标记.
3. 扫描 G 的所有边. 如果找到一条边 (a, b), a 被标记而 b 没有被标记, 那么标记 b.
4. 若 t 被标记, 则**接受**; 否则, **拒绝**.

P 中的问题举例

令 *RELPRIME* 代表检查两个数是否互素的问题, 即

P 中的问题举例

令 *RELPRIME* 代表检查两个数是否互素的问题, 即

$$RELPRIME = \{\langle x, y\rangle | x \text{ 与 } y \text{ 互素}\}$$

P 中的问题举例

令 *RELPRIME* 代表检查两个数是否互素的问题, 即

$$RELPRIME = \{\langle x, y\rangle | x \text{ 与 } y \text{ 互素}\}$$

Theorem

$RELPRIME \in P$

P 中的问题举例

令 *RELPRIME* 代表检查两个数是否互素的问题, 即

$$RELPRIME = \{\langle x, y\rangle | x \text{ 与 } y \text{ 互素}\}$$

Theorem

$RELPRIME \in P$

Theorem

每一个上下文无关语言都是 P 的成员.

Outline

1 度量复杂性

2 P 类

3 NP 类
- NP 问题
- NP 中的问题举例
- P 与 NP 的关系

4 NP 完全性

5 几个 NP 完全问题

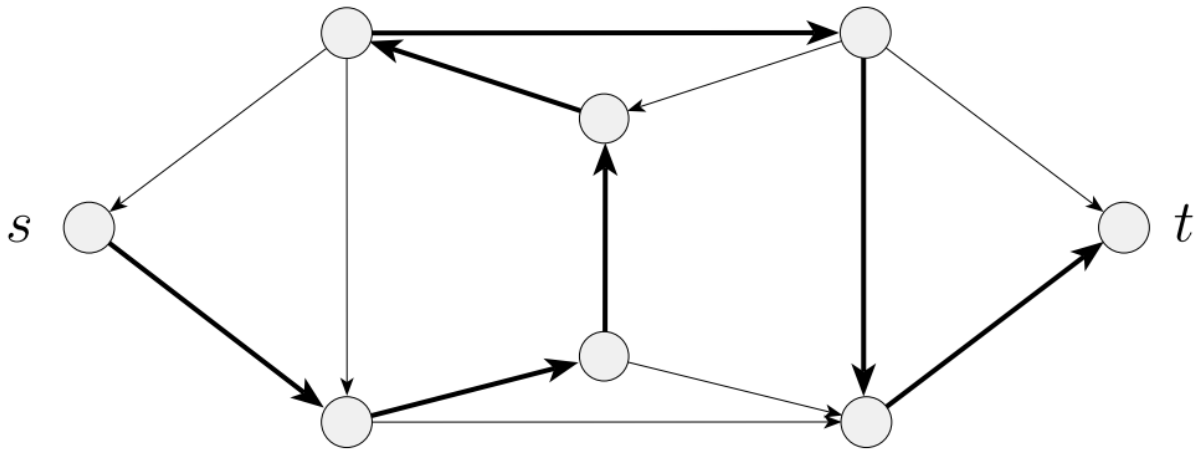
NP 问题

有向图 G 中的**哈密顿路径**是通过每个结点恰好一次的有向路径. 考虑这样一个问题: 验证一个有向图是否包含一条哈密顿路径连接着两个指定的结点. 令

NP 问题

有向图 G 中的**哈密顿路径**是通过每个结点恰好一次的有向路径. 考虑这样一个问题: 验证一个有向图是否包含一条哈密顿路径连接着两个指定的结点. 令

$$HAMPATH = \{\langle G, s, t\rangle | G \text{ 是包含从 } s \text{ 到 } t \text{ 的哈密顿路径的有向图}\}$$



NP 问题

没有人知道 *HAMPATH* 是否能在多项式时间内求解, 但其具有一个特点, 称为**多项式可验证性**.

NP 问题

没有人知道 *HAMPATH* 是否能在多项式时间内求解, 但其具有一个特点, 称为**多项式可验证性**.

Definition (验证机)

语言 A 的验证机是一个算法 V, 这里

$$A = \{w | \text{对某个字符串 } c, V \text{ 接受 } \langle w, c \rangle\}$$

因为只根据 w 的长度来度量验证机的时间, 所以多项式时间验证机在 w 的长度的多项式时间内运行. 若语言 A 有一个多项式验证机, 则称它为**多项式时间可验证的**.

NP 问题

Definition (NP)

NP 是具有多项式时间验证机的语言类.

NP 问题

Definition (NP)

NP 是具有多项式时间验证机的语言类.

定理

一语言在 NP 中, 当且仅当它能被某个非确定型多项式时间图灵机判定.

NP 问题

Definition (NP)

NP 是具有多项式时间验证机的语言类.

定理

一语言在 NP 中, 当且仅当它能被某个非确定型多项式时间图灵机判定.

Definition (非确定型时间复杂类)

NTIME$(t(n)) = \{L | L$ 是被 $O(t(n))$ 时间的非确定型图灵机判定的语言$\}$

NP 问题

Definition (NP)

NP 是具有多项式时间验证机的语言类.

定理

一语言在 NP 中, 当且仅当它能被某个非确定型多项式时间图灵机判定.

Definition (非确定型时间复杂类)

$\mathsf{NTIME}(t(n)) = \{L | L$ 是被 $O(t(n))$ 时间的非确定型图灵机判定的语言$\}$

推论

$\mathsf{NP} = \bigcup_k \mathsf{NTIME}(n^k)$

NP 中的问题举例

Example (团问题)

无向图中的一个团是一个子图, 其中每两个结点都有边相连. k 团就是包含 k 个结点的团. 团问题旨在判定一个图是否包含指定大小的团. 令:

$$\mathit{CLIQUE} = \{\langle G, k\rangle | G \text{ 是包含 } k \text{ 团的无向图}\}$$

团问题 *CLIQUE* 属于 NP.

NP 中的问题举例

Example (团问题)

无向图中的一个团是一个子图, 其中每两个结点都有边相连. k 团就是包含 k 个结点的团. 团问题旨在判定一个图是否包含指定大小的团. 令:

$$\mathit{CLIQUE} = \{\langle G, k\rangle | G \text{ 是包含 } k \text{ 团的无向图}\}$$

团问题 *CLIQUE* 属于 NP.

Example (*SUBSET-SUM* 问题)

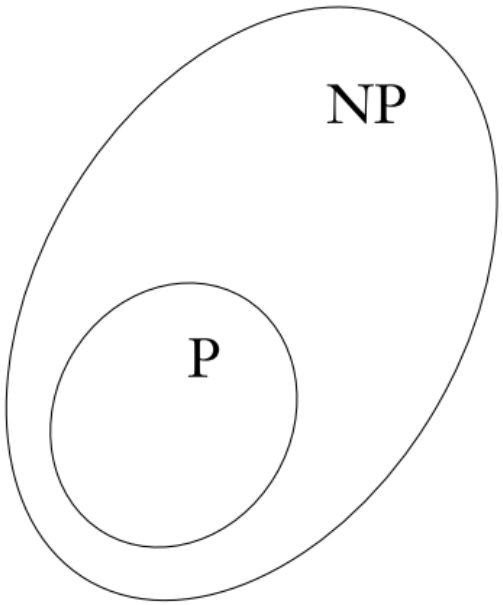
判定一个集合中是否存在一个加起来等于 t 的子集, 即

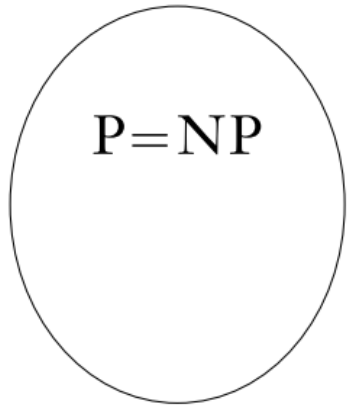
$$\mathit{SUBSET\text{-}SUM} = \{\langle s, t\rangle | s = \{x_1, \cdots, x_k\}, \text{且存在 } \{y_1, \cdots, y_l\} \subseteq \{x_1, \cdots, x_k\} \text{ 使得 } \sum y_i = t\}$$

SUBSET-SUM 问题属于 NP.

P 与 NP 的关系

$P = NP$ 是否成立的问题是理论计算机科学和当代数学中最大的悬而未决的问题之一. 如果这两个类相等, 那么所有多项式可验证的问题都将是多项式可判定的. 大多数研究者相信这两个类是不相等的.





Outline

1 度量复杂性

2 P 类

3 NP 类

4 NP 完全性
- 多项式时间可归约性
- NP 完全性的定义
- 库克-列文定理

5 几个 NP 完全问题

NP 完全性

在 P 与 NP 问题上的一个重大进展是在 20 世纪 70 年代由 Stephen Cook 和 Leonid Levin 完成的. 他们发现 NP 中某些问题的复杂性与整个类的复杂性相关联.

NP 完全性

在 P 与 NP 问题上的一个重大进展是在 20 世纪 70 年代由 Stephen Cook 和 Leonid Levin 完成的. 他们发现 NP 中某些问题的复杂性与整个类的复杂性相关联.

- 这些问题中任何一个如果存在多项式时间算法, 那么所有 NP 问题都是多项式时间可解的.

NP 完全性

在 P 与 NP 问题上的一个重大进展是在 20 世纪 70 年代由 Stephen Cook 和 Leonid Levin 完成的. 他们发现 NP 中某些问题的复杂性与整个类的复杂性相关联.

- 这些问题中任何一个如果存在多项式时间算法, 那么所有 NP 问题都是多项式时间可解的.
- 这些问题被称为 NP 完全的 (NP-complete).

NP 完全性

在 P 与 NP 问题上的一个重大进展是在 20 世纪 70 年代由 Stephen Cook 和 Leonid Levin 完成的. 他们发现 NP 中某些问题的复杂性与整个类的复杂性相关联.

- 这些问题中任何一个如果存在多项式时间算法, 那么所有 NP 问题都是多项式时间可解的.
- 这些问题被称为 NP 完全的 (NP-complete).

NP 完全问题对于理论和实践都具有重要意义:

- 在理论方面, 试图证明 P 不等于 NP 可以将注意力集中到一个 NP 完全问题上.
- 在实践方面, NP 完全性可以防止为某一具体问题浪费时间去寻找本不存在的多项式时间算法.

NP 完全性

下面给出第一个 NP 完全问题：**可满足性问题**

NP 完全性

下面给出第一个 NP 完全问题：**可满足性问题**

布尔可满足性问题

布尔公式是包含布尔变量和运算的表达式. 例如：$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ 是一个布尔公式. 如果对变量的某个 0,1 赋值使得一个公式的值等于 1, 则该布尔公式是**可满足的**. 上面的公式是可满足的, 因为赋值 $x = 0$, $y = 1$,$z = 0$ 使得 ϕ 的值为 1. 称该赋值满足 ϕ. 可满足性问题就是判定一个布尔公式是否是可满足的. 令 $SAT = \{\langle\phi\rangle | \phi$ 是可满足的布尔公式$\}$

NP 完全性

下面给出第一个 NP 完全问题：**可满足性问题**

布尔可满足性问题

布尔公式是包含布尔变量和运算的表达式. 例如：$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ 是一个布尔公式. 如果对变量的某个 0,1 赋值使得一个公式的值等于 1, 则该布尔公式是**可满足的**. 上面的公式是可满足的, 因为赋值 $x = 0$, $y = 1$,$z = 0$ 使得 ϕ 的值为 1. 称该赋值满足 ϕ. 可满足性问题就是判定一个布尔公式是否是可满足的. 令 $SAT = \{\langle\phi\rangle | \phi$ 是可满足的布尔公式$\}$

现给一个把 *SAT* 问题的复杂性与 NP 所有问题的复杂性联系起来的定理

NP 完全性

下面给出第一个 NP 完全问题：**可满足性问题**

布尔可满足性问题

布尔公式是包含布尔变量和运算的表达式. 例如：$\phi = (\overline{x} \wedge y) \vee (x \wedge \overline{z})$ 是一个布尔公式. 如果对变量的某个 0,1 赋值使得一个公式的值等于 1, 则该布尔公式是**可满足的**. 上面的公式是可满足的, 因为赋值 $x = 0$, $y = 1$,$z = 0$ 使得 ϕ 的值为 1. 称该赋值满足 ϕ. 可满足性问题就是判定一个布尔公式是否是可满足的. 令 $SAT = \{\langle\phi\rangle | \phi$ 是可满足的布尔公式$\}$

现给一个把 *SAT* 问题的复杂性与 NP 所有问题的复杂性联系起来的定理

Theorem

$SAT \in P$ 当且仅当 $P = NP$.

多项式时间可归约性

在分析 NP 完全性过程中一个重要的概念是**多项式时间可归约性**

多项式时间可归约性

在分析 NP 完全性过程中一个重要的概念是**多项式时间可归约性**

Definition (多项式时间可计算函数)

若存在多项式时间图灵机 M, 使得在任何输入 w 上, M 停机时 $f(w)$ 恰好在带子上, 则称函数 $f : \Sigma^* \to \Sigma^*$ 为多项式时间可计算函数.

多项式时间可归约性

在分析 NP 完全性过程中一个重要的概念是**多项式时间可归约性**

Definition (多项式时间可计算函数)

若存在多项式时间图灵机 M, 使得在任何输入 w 上, M 停机时 $f(w)$ 恰好在带子上, 则称函数 $f : \Sigma^* \to \Sigma^*$ 为多项式时间可计算函数.

Definition (多项式时间归约)

语言 A 称为**多项式时间映射可归约**到语言 B, 或简称为多项式时间可归约到 B, 记为 $A \leqslant_P B$, 若存在多项式时间可计算函数 $f : \Sigma^* \to \Sigma^*$, 对于每一个 w, 有

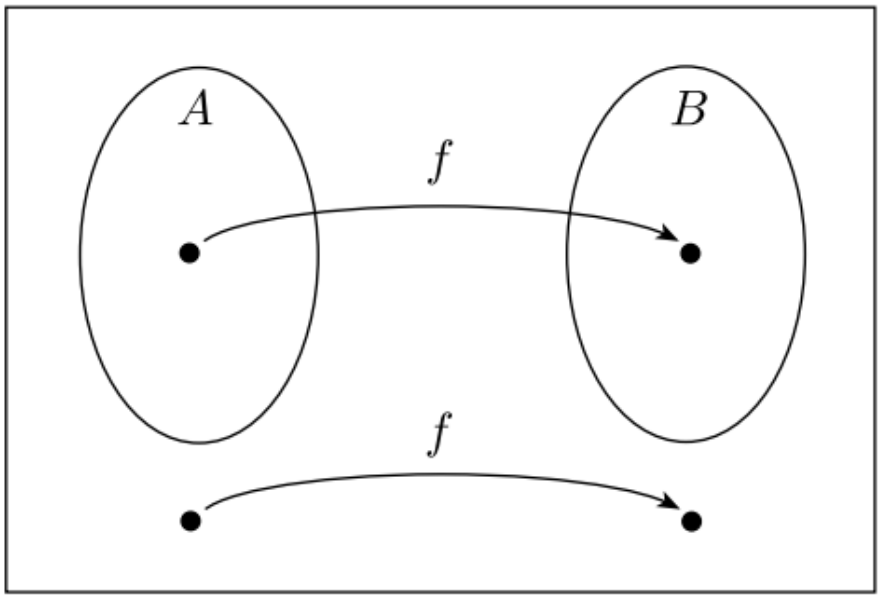
$$w \in A \Leftrightarrow f(w) \in B$$

函数 f 称为 A 到 B 的多项式时间归约.

多项式时间可归约性

多项式时间可归约性是映射可归约性的有效近似.

- 如同一般的映射归约一样, A 到 B 的多项式时间归约提供了一种方法, 把 A 的成员资格判定转化为 B 的成员资格判定,
- 只是现在这种转化是有效完成的.



多项式时间可归约性

Theorem

若 $A \leqslant_P B$ 且 $B \in P$, 则 $A \in P$.

多项式时间可归约性

Theorem

若 $A \leqslant_P B$ 且 $B \in P$, 则 $A \in P$.

证明：设 M 是判定 B 的多项式时间算法, f 是从 A 到 B 的多项式时间归约. 判定 A 的多项式时间算法 N 的描述如下：

多项式时间可归约性

Theorem

若 $A \leqslant_P B$ 且 $B \in P$, 则 $A \in P$.

证明：设 M 是判定 B 的多项式时间算法, f 是从 A 到 B 的多项式时间归约. 判定 A 的多项式时间算法 N 的描述如下：

N 对输入 w:

多项式时间可归约性

Theorem

若 $A \leqslant_P B$ 且 $B \in P$, 则 $A \in P$.

证明：设 M 是判定 B 的多项式时间算法, f 是从 A 到 B 的多项式时间归约. 判定 A 的多项式时间算法 N 的描述如下：

N 对输入 w:

1. 计算 $f(w)$
2. 在输入 $f(w)$ 上运行 M, 输出 M 的输出

多项式时间可归约性

Theorem

若 $A \leqslant_P B$ 且 $B \in P$, 则 $A \in P$.

证明：设 M 是判定 B 的多项式时间算法, f 是从 A 到 B 的多项式时间归约. 判定 A 的多项式时间算法 N 的描述如下：

N 对输入 w:

1. 计算 $f(w)$
2. 在输入 $f(w)$ 上运行 M, 输出 M 的输出

若 $w \in A$, 则 $f(w) \in B$, 因为 f 是从 A 到 B 的归约. 于是, 只要 $w \in A$, M 就接受 $f(w)$. 另外, 因为 N 的两个步骤都在多项式时间内运行, 故 N 在多项式时间内运行.

多项式时间可归约性

我们在看多项式归约的一个例子之前，首先介绍 $3SAT$，它是可满足性问题的一种特殊情况，因为其中所有公式都具有一种特殊形式

多项式时间可归约性

我们在看多项式归约的一个例子之前，首先介绍 $3SAT$，它是可满足性问题的一种特殊情况，因为其中所有公式都具有一种特殊形式

- 文字是一个布尔变量或布尔变量的非，如 x 或 $\overline{x}$

多项式时间可归约性

我们在看多项式归约的一个例子之前，首先介绍 3*SAT*，它是可满足性问题的一种特殊情况，因为其中所有公式都具有一种特殊形式

- **文字**是一个布尔变量或布尔变量的非，如 x 或 $\overline{x}$
- **子句**是由 $\vee$ 连接起来的若干文字，如 $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$

多项式时间可归约性

我们在看多项式归约的一个例子之前，首先介绍 3*SAT*，它是可满足性问题的一种特殊情况，因为其中所有公式都具有一种特殊形式

- **文字**是一个布尔变量或布尔变量的非，如 x 或 $\overline{x}$
- **子句**是由 $\vee$ 连接起来的若干文字，如 $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$
- 一个布尔公式若是由 $\wedge$ 连接的若干子句组成，则为**合取范式**，称它为 cnf 公式，

多项式时间可归约性

我们在看多项式归约的一个例子之前，首先介绍 3*SAT*，它是可满足性问题的一种特殊情况，因为其中所有公式都具有一种特殊形式

- **文字**是一个布尔变量或布尔变量的非，如 x 或 $\overline{x}$
- **子句**是由 $\vee$ 连接起来的若干文字，如 $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$
- 一个布尔公式若是由 $\wedge$ 连接的若干子句组成，则为**合取范式**，称它为 cnf 公式，如：$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_3 \vee \overline{x_5} \vee \overline{x_6}) \wedge (x_3 \vee \overline{x_6})$

多项式时间可归约性

我们在看多项式归约的一个例子之前，首先介绍 3*SAT*，它是可满足性问题的一种特殊情况，因为其中所有公式都具有一种特殊形式

- **文字**是一个布尔变量或布尔变量的非，如 x 或 $\overline{x}$
- **子句**是由 $\vee$ 连接起来的若干文字，如 $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$
- 一个布尔公式若是由 $\wedge$ 连接的若干子句组成，则为**合取范式**，称它为 cnf 公式，如：$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_3 \vee \overline{x_5} \vee \overline{x_6}) \wedge (x_3 \vee \overline{x_6})$
- 若所有子句都有三个文字，则为 3cnf 公式，如：

多项式时间可归约性

我们在看多项式归约的一个例子之前，首先介绍 3*SAT*，它是可满足性问题的一种特殊情况，因为其中所有公式都具有一种特殊形式

- **文字**是一个布尔变量或布尔变量的非，如 x 或 $\overline{x}$
- **子句**是由 $\vee$ 连接起来的若干文字，如 $(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee x_4)$
- 一个布尔公式若是由 $\wedge$ 连接的若干子句组成，则为**合取范式**，称它为 cnf 公式，如：$(x_1 \vee \overline{x_2} \vee \overline{x_3} \vee \overline{x_4}) \wedge (x_3 \vee \overline{x_5} \vee \overline{x_6}) \wedge (x_3 \vee \overline{x_6})$
- 若所有子句都有三个文字，则为 3cnf 公式，如：

$$(x_1 \vee \overline{x_2} \vee \overline{x_3}) \wedge (x_3 \vee \overline{x_5} \vee x_6) \wedge (x_3 \vee \overline{x_6} \vee x_4) \wedge (x_4 \vee x_5 \vee x_6)$$

- 令 $3SAT = \{\langle\phi\rangle | \phi$是可满足的 3cnf 公式$\}$. 如果一个赋值满足一个 cnf 公式，那么每一个子句必须至少包含一个值为 1 的文字.

多项式时间可归约性

下面的定理给出 $3SAT$ 问题到 $CLIQUE$ 问题的多项式时间归约.

多项式时间可归约性

下面的定理给出 *3SAT* 问题到 *CLIQUE* 问题的多项式时间归约.

Theorem

3SAT 多项式时间可归约到 *CLIQUE*.

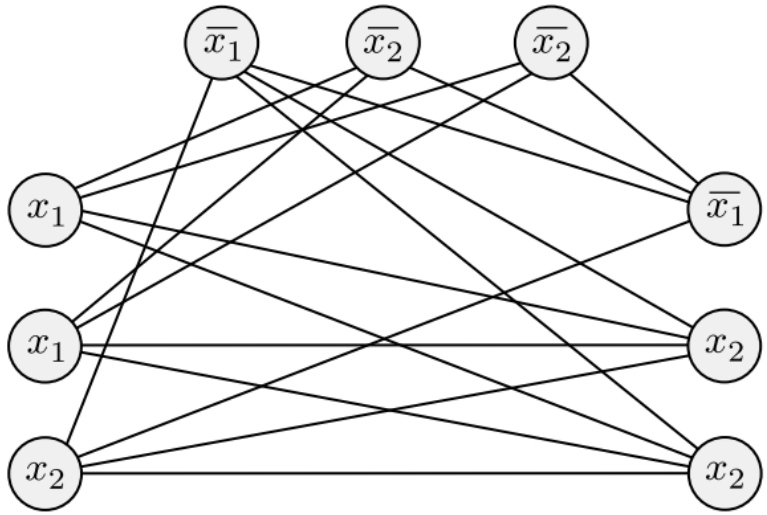
多项式时间可归约性

下面的定理给出 3*SAT* 问题到 *CLIQUE* 问题的多项式时间归约.

Theorem

3SAT 多项式时间可归约到 CLIQUE.

证明思路：给出从 3*SAT* 到 *CLIQUE* 的多项式时间归约 f, 把公式转化为图. 在构造的图中, 指定大小的团对应于公式的满足赋值.



归约从 $\phi = (x_1 \vee x_1 \vee x_2) \wedge (\overline{x_1} \vee \overline{x_2} \vee \overline{x_2} \wedge (\overline{x_1} \vee x_2 \vee x_2))$ 生成的图.

NP 完全性的定义

Definition (NP 完全的)

如果语言 B 满足下面两个条件, 就成为 ***NP* 完全的** (NP-complete):

1. B 属于 NP, 并且
2. NP 中的每个 A 都多项式时间可归约到 B.

NP 完全性的定义

Definition (NP 完全的)

如果语言 B 满足下面两个条件, 就成为 ***NP* 完全的** (NP-complete):

1. B 属于 NP, 并且
2. NP 中的每个 A 都多项式时间可归约到 B.

Theorem

若上述的 B 是 *NP* 完全的, 且 $B \in P$, 则 $P = NP$

NP 完全性的定义

Definition (NP 完全的)

如果语言 B 满足下面两个条件, 就成为 ***NP* 完全的** (NP-complete):

1. B 属于 NP, 并且
2. NP 中的每个 A 都多项式时间可归约到 B.

Theorem

若上述的 B 是 *NP* 完全的, 且 $B \in P$, 则 $P = NP$

Theorem

若上述的 B 是 *NP* 完全的, 且 $B \leqslant_P C$, C 属于 *NP*, 则 C 是 *NP* 完全的.

库克-列文定理

库克-列文定理给出了第一个 NP 完全问题

库克-列文定理

库克-列文定理给出了第一个 NP 完全问题

Theorem

SAT 是 NP 完全的

库克-列文定理

库克-列文定理给出了第一个 NP 完全问题

Theorem

SAT 是 NP 完全的

一旦有了一个 NP 完全问题, 就可以从它出发, 通过多项式时间归约得到其他 NP 完全问题.

库克-列文定理

库克-列文定理给出了第一个 NP 完全问题

Theorem

SAT 是 NP 完全的

一旦有了一个 NP 完全问题, 就可以从它出发, 通过多项式时间归约得到其他 NP 完全问题.

NPC 问题的重要意义

- 在理论方面, 试图证明 P 不等于 NP 可以将注意力集中到一个 NP 完全问题上. 试图证明 P 等于 NP 可以证明一个 NPC 问题属于 P.
- 在实践方面, NP 完全性可以防止为某一具体问题浪费时间去寻找本不存在的多项式时间算法.

Outline

1 度量复杂性

2 P 类

3 NP 类

4 NP 完全性

5 几个 NP 完全问题

6 NP 难问题

几个 NP 完全问题

Corollary 1

3*SAT* 是 NP 完全的.

几个 NP 完全问题

Corollary 1

3*SAT* 是 NP 完全的.

Corollary 2

CLIQUE 是 NP 完全的.

几个 NP 完全问题

Corollary 1

3*SAT* 是 NP 完全的.

Corollary 2

CLIQUE 是 NP 完全的.

顶点覆盖问题

若 G 是无向图, 则 G 的**顶点覆盖** (vertex-cover) 是结点的一个子集, 使得 G 的每条边都与子集中的结点之一相关联. 顶点覆盖问题旨在确定图中是否存在指定规模的顶点覆盖:

$$\textit{VERTEX-COVER} = \{\langle G, k\rangle | G \text{ 是具有 } k \text{ 个结点的顶点覆盖的无向图}\}$$

顶点覆盖问题是 NP 完全的

几个 NP 完全问题

独立集问题

若 G 是无向图, 如果在 G 的顶点子集 I 中任何亮点都没有 G 的边相连, 则称 I 是独立集. 如果一个独立集不小于 (顶点数不少于) 原图的任何独立集, 则这个独立集就是最大的.

$$IS = \{\langle G, k\rangle | G \text{ 是具有 } k \text{ 个结点的独立集的无向图}\}$$

独立集问题是 NP 完全的

几个 NP 完全问题

独立集问题

若 G 是无向图, 如果在 G 的顶点子集 I 中任何亮点都没有 G 的边相连, 则称 I 是独立集. 如果一个独立集不小于 (顶点数不少于) 原图的任何独立集, 则这个独立集就是最大的.

$$IS = \{\langle G, k\rangle | G \text{ 是具有 } k \text{ 个结点的独立集的无向图}\}$$

独立集问题是 NP 完全的

Theorem

哈密顿路径问题 *HAMPATH* 是 *NP* 完全的

几个 NP 完全问题

独立集问题

若 G 是无向图, 如果在 G 的顶点子集 I 中任何亮点都没有 G 的边相连, 则称 I 是独立集. 如果一个独立集不小于 (顶点数不少于) 原图的任何独立集, 则这个独立集就是最大的.

$$IS = \{\langle G, k\rangle | G \text{ 是具有 } k \text{ 个结点的独立集的无向图}\}$$

独立集问题是 NP 完全的

Theorem

哈密顿路径问题 *HAMPATH* 是 *NP* 完全的

Theorem

子集和问题 *SUBSET-SUM* 是 *NP* 完全的

Outline

1 度量复杂性

2 P 类

3 NP 类

4 NP 完全性

5 几个 NP 完全问题

6 NP 难问题

NP 难问题

Definition (NP 难问题)

有些问题 L 是如此困难的, 以至于虽然能证明 NP 完全性定义中条件 (2) (NP 中每个语言都在多项式时间里归约到 L), 但不能证明条件 (1) (L 属于 NP). 如果是这样, 就称 L 为 NP 难的

NP 难问题

Definition (NP 难问题)

有些问题 L 是如此困难的, 以至于虽然能证明 NP 完全性定义中条件 (2) (NP 中每个语言都在多项式时间里归约到 L), 但不能证明条件 (1) (L 属于 NP). 如果是这样, 就称 L 为 NP 难的

证明 L 是 NP 难的, 就足以证明 L 非常可能需要指数时间或更糟糕. 但是如果 L 不属于 NP, 则证明 L 明显的困难性并不支持论证所有 NP 完全问题都是困难的. 也就是说, 也许事实是 $P = NP$, 而 L 仍然需要指数时间.

NP 难问题

Definition (NP 难问题)

有些问题 L 是如此困难的, 以至于虽然能证明 NP 完全性定义中条件 (2) (NP 中每个语言都在多项式时间里归约到 L), 但不能证明条件 (1) (L 属于 NP). 如果是这样, 就称 L 为 NP 难的

证明 L 是 NP 难的, 就足以证明 L 非常可能需要指数时间或更糟糕. 但是如果 L 不属于 NP, 则证明 L 明显的困难性并不支持论证所有 NP 完全问题都是困难的. 也就是说, 也许事实是 $P = NP$, 而 L 仍然需要指数时间.

NP 难问题与 NP 问题、NP 完全问题的关系

- NP 难问题与 NP 问题有交集
- NPC 难问题包含 NP 完全问题

总结

- 度量复杂性
  - 大 O 和 小 o 记法
  - 计算复杂性类
  - 模型间的复杂性关系
- P 问题
  - 多项式时间
  - P 中的问题举例
- NP 问题
  - NP 问题的定义
  - NP 中的问题举例
  - P 与 NP 的关系

总结

- 度量复杂性
  - 大 O 和 小 o 记法
  - 计算复杂性类
  - 模型间的复杂性关系
- P 问题
  - 多项式时间
  - P 中的问题举例
- NP 问题
  - NP 问题的定义
  - NP 中的问题举例
  - P 与 NP 的关系
- NP 完全性
  - 多项式时间可归约性
  - NP 完全性定义
  - 库克-列文定理
- 几个 NP 完全问题
  - 最大团问题
  - 最小顶点覆盖问题
  - 最大独立集问题
  - 哈密顿路径问题
- NP 难问题
  - NP 难问题的定义和关系