1. NP-hard 问题.

→ 多项式时间解决 ⇒ $P=NP$

① 难题 NP-hard.
② 快（多项式时间）.
③ 准确. 正确解.

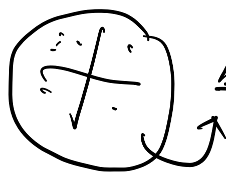
② ③ → P问题
① ② → 近似算法
→ 固定参数算法. FPT. ↑ 是否能快

2. 基本思路:

对参数是EXP. 对问题是Poly.

└ ① 非负整数. $k(2)$ ← 问题子输入.
② 最好可以简化计算.

⇒ 参数化问题 = problem + 参数.



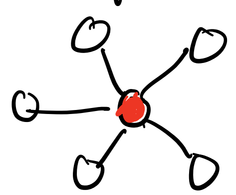
⇒ 算法目标, 多项式时间 in 问题规模 n.
EXP in 参数 k

例1. k-顶点覆盖. NP-hard.

给定一个图 $G=(V,E)$, 和一个非负整数 k.

求问. 是否存在一个集合 S, $|S| \le k$. 可 cover 所有的边.

参数 k. k ? $|V|$



方案一 暴力搜索.

在NP ← {
- 所有有 k 个顶点的组合. $\binom{|V|}{k} + \binom{|V|}{k-1} + \cdots \binom{|V|}{1}$ = $O(|V|^k)$
- 对每一个组合衡量. $O(|E|)$
}

⇒ $O(V^k \cdot E)$

分析 $O(V^k \cdot E)$. 对应固定参数 k. poly.

输入规模. L 可能很大. $O(V^{100} \cdot E)$ 不好.

$\Rightarrow n^{f(k)}$. X. 不够好.

v.s.

$f(k) \cdot n^{O(1)}$ ✓ 比较好.

$f: \mathbb{N} \to \mathbb{N}$ 参数. 需知 k 以 n 无关的参数.

方案二 bounded search-tree 算法. (类似分支限界).

1). 选择任一边 $e = (u, v) \in E$.

至少为 S. u 在 S. or $v \in S$. or $u, v \in S$.

2) 局部暴力搜索: 减/猜.

① $S = S \cup \{u\}$. $|S| \le k$.

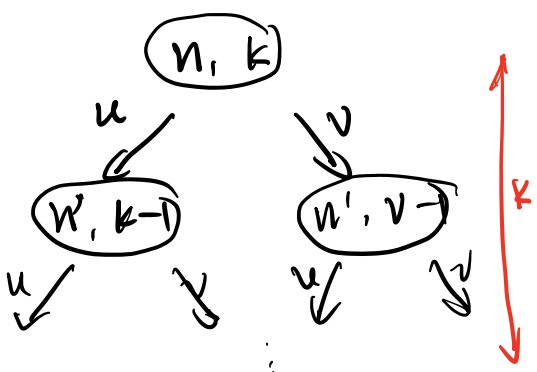
↓ 删除 u & 所有 u 相连的边.

$O(V)$ → A_1 $k \leftarrow k-1$ 递归调用.

② $S = S \cup \{v\}$.

↓ A_2 同上 递归.

- return. $A_1 \cup A_2$.



DP = recursive + guess + memo.

3). 递归出口. 成功 $E = \emptyset$

失败. 走到叶结点. $k = 0$ && $E \neq \emptyset$

4). $O(2^k)$ 个节点. 每个节点 $O(V)$. $\Rightarrow O(2^k \cdot V)$.

分析 $O(2^k \cdot V)$. → $f(k)\cdot n^{O(1)}$ $k = O(\lg V)$.

└ 给定 k，不变的.

3. Good: $f(k)\cdot n^{O(1)}$, 不是 $f(k)+n^{O(1)}$?

定理: $f(k)\cdot n^{c_1} \equiv g(k) + n^{c_2}$

证明: $\Leftarrow$ 假设 $g(k)$. $n^{c_2} \geq 1$.

$\Rightarrow$ if $n \leq f(k)$, 即 $f(k)\cdot n^{c_1} \leq f(k)^{c_1+1}$

if $f(k) \leq n$. 即 $f(k)\cdot n^{c_1} \leq n^{c_1+1}$

$\Rightarrow f(k)\cdot n^{c_1} \leq \max\{f(k)^{c_1+1}, n^{c_1+1}\}$

$\leq \underbrace{f(k)^{c_1+1}}_{=g(k)} + n^{\underbrace{c_1+1}_{=c_2}}$

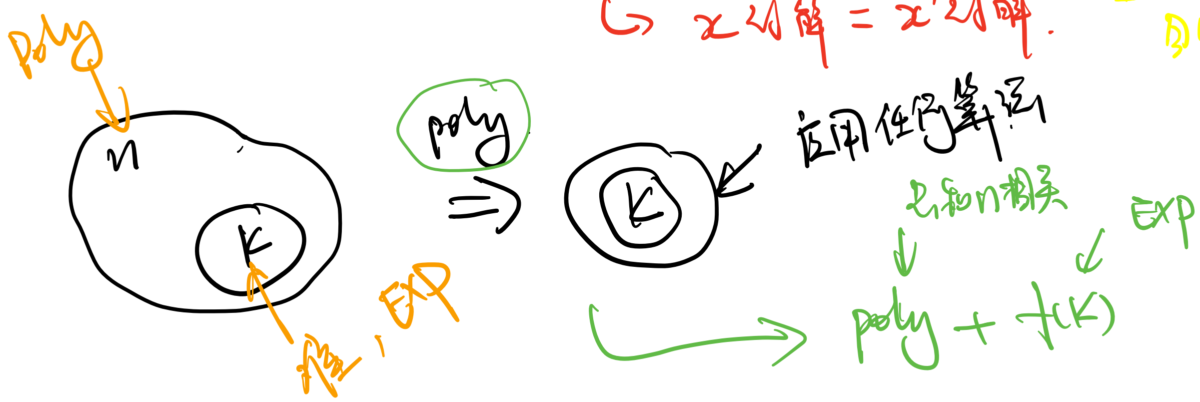
证明2. $xy \leq x^2+y^2$ → $g(k) = f^2(k)$ $c_2 = 2c_1$

4. 核心化. Kernelization, 一个自动简化.

⇒ 对 input 进行预处理, 类似归约 (多项式时间). → 是可能的,

input (x, k) 变换为 等价的 (x', k') $|x'| \leq f(k)$

└> x 的解 = x' 的解.



定理. FPT $\Longleftrightarrow$ 存在核心化.

证明: $\Leftarrow$. 存在核心化, 核心化后新的问题规模 n'

$n' \le f(k)$ 解决 $f(k)$ 新问题的时间

$n^{O(1)} + g(f(k))$

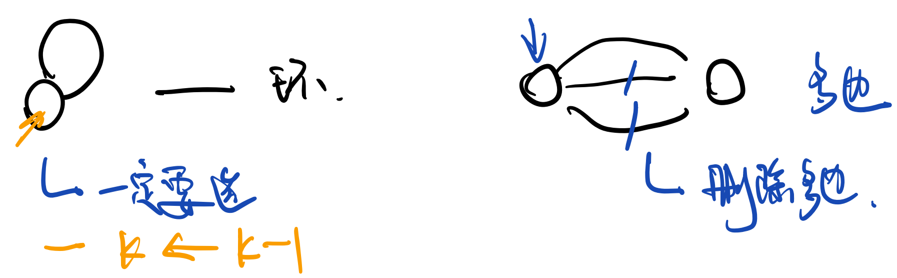
$\Rightarrow$ 假设存在算法 A, $f(k)\cdot n^c$

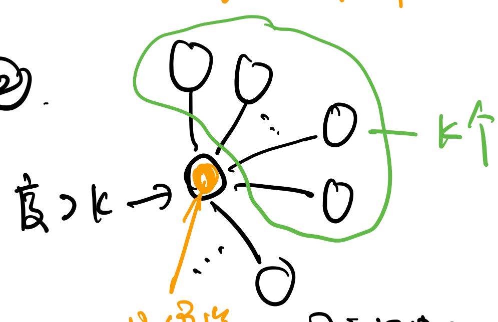
k已知:
- 对 $n \le f(k)$: 相当于已做核心化.
- 对 $n \ge f(k)$:
  - A的时间复杂度. $f(k)\cdot n^c \le n^{c+1}$
  - 输出 $O(1)$.

若k未知, A运行 n^c

例2. k-顶点覆盖的核心化方法.

核心化:

① 

② 

(问题化简).

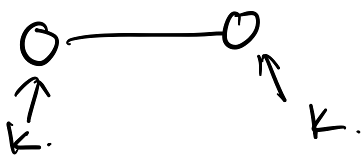
1) 选择度 > k 的顶点,
2) 删掉该顶点和相邻边.
3) $k \leftarrow k-1$

$|V|$

→ ③ 剩下的图：简单图 & $\max$(顶点的度) $= k$.

核心化后

⇒ 最多可 cover k^2 条边.



⇒ if $|E'| > k^2$, False

else $|E'| \le k^2$ → 新问题 $G = (V', E')$

$|V'| \le 2k^2$

⇒ 归约为 输入规模为 $O(3k^2)$ 的新问题

└ $|V'| + |E'|$

时间复杂度分析：① 核心化用时 $O(|V'| + |E'|)$.

② 暴力搜索, $O(V^k \cdot E') = O(V' + E' + 2^k \cdot k^{k})$

搜索树. $O(V' + 2^k) = O(V' + E' + 2^k \cdot k^2)$

当前最优. $O(kV + c^k)$

△ → 1.xx.

FPT vs. 近似算法？

$f(1/\varepsilon) \cdot n^{O(1)}$

定理：如果一个优化问题有 EPTAS.

{ PTAS $O(n^{2/\varepsilon})$
FPTAS $O(n/\varepsilon^2)$ }

⇒ 对应的判断性问题一定有 FPT.