算法以及复杂性

回顾 & 思考

- 计算机能做什么?
  - 可计算问题
  - 不可计算问题
- 问题:
  - 可计算问题都能够被有效解决吗?

理论计算机中关心的话题

如何快速计算问题

- 一个问题是不是能被不错地解决
  - 不错：以一个OK的速度，即多项式时间复杂度。
  - 哪些问题是能够被不错地解决
  - 一个重要理论问题：P? NP。
  - 一些相关的概念：P, NP, NP-Hard.....

已搞定

- 我已经知道他能被不错得解决之后，可不可以更快。
  - 如何在更快得解决各种P问题（设计更好的算法）。

今天我们讨论: 问题难度的分类

类似于可计算问题，不可计算问题。

我们关心的问题

- 是不是**所有问题**都能被快速（多项式时间）解决。
- 答案：不是
  - 停机问题不行
- 进一步：那么有没有问题是可计算的，但是不是可快速计算的？
  - 修改版的停机问题不行
  - 简单描述：问一个图灵机在某个输入 x 下，能不能在 $2^{|x|}$ 步内停机。
- 证明关键：对角线法则。

对角线法则够用吗？

许多实际问题的分类困难

- 有许多问题我们**既不能**证明他快速，
  - 没有找到多项式时间的算法。
- 我们**也不能**利用对角线发现法则证明他们不能快速计算。

搜索类问题

搜索类问题

- 举例：
  - 给你一串数字，问你其中是否存在长度为k的上升子串。
  - 给你一个图，问你图里是否存在某个起点到某个终点长度为k的路。
  - 给你一个图，问你能不能在图里选k个点把所有边都覆盖。
- 总结
  - 问题中存在着**许多选项**（指数数量，但多项式规模）。
  - 我们可以通过**搜索**所有选项，来得到问题的答案。
  - 疑问：是不是能有**聪明**的办法，**不用搜索**所有选项，就得到答案呢？

搜索类问题

- 疑问：是不是能有**聪明**的办法，**不用搜索**所有选项，就得到答案呢？
- 有些我们已经知道可以了（大家以后都会学到）
  - 给你一串数字，问你其中是否存在长度为k的上升子串。
  - 给你一个图，问你图里是否存在某个起点到某个终点长度为k的路。
- 但有许多问题，我们还没那么聪明。

给这类问题一个名字

- 这就是NP，我们给出两种定义（他们等价）。
- NP：非确定型图灵机能做多项式时间内解决的问题。
  - 非确定性图灵机当面临某个东西（一串数中的某个数）选或者不选的时候，可以**同时**去尝试选或者不选，所以他在多项式步骤内就能完成全部搜索。
- NP：确定型图灵机能在多项式时间内**验证**的问题。
  - 给你一个选项，你能判断这个选项符合不符合要求。
  - 符合要求，整个问题的答案应该就是yes；不符合，一切还都有可能。
- 对比P的定义：确定型图灵机在多项式时间内可**判定**的问题。
  - 在复杂度理论里，大家喜欢讨论判定问题（yes or no），所以可以解决相当于可以判定（准确的回答yes or no）。

我们关注第二个定义，把他formal一点。

- 对一个判定问题 f，什么时候$f \in NP$呢?
- 存在一个多项式时间的**图灵机（验证器）** g，输入是x和y。
  - x： 原问题的输入。
  - y：某个多项式规模的选项（$|x|$的多项式）。
- 该**图灵机（验证器）**能做到
  - $f(x) = yes$时，存在一个y，使得 $g(x, y) = yes$。
  - $f(x) = no$时，对于所有的y，$g(x, y) = no$。
- 对比 $f \in P$ 的定义
  - 存在一个多项式时间的**图灵机（判定器）**。

帮助我们理解定义的小插曲

- 证明：如下问题都是NP问题
  - 给你一串数字，问你其中是否存在长度为k的上升子串。
  - 给你一个图，问你图里是否存在某个起点到某个终点长度为k的路。
  - 给你一个图，问你能不能在图里选k个点把所有边都覆盖。
- 证明：所有P问题都是NP问题。

重新认识我们的疑问

- 疑问：是不是所有搜索问题都能被**聪明**的解决？
- 相当于在问，是不是$P = NP$？
- 但研究者普遍相信 $P \subsetneq NP$。
  - 做数学证明题和验证人家证明的对不对的区别。

为了研究这个问题

我们想去盯着那些比较难的NP问题下手。

定义难度

- 根据我们目前会不会快速解决来定义问题的难度？
- 不严格。
- 那么，我们该怎么比较两个问题的难度呢？
- 没错，就要用**规约**的思路来定义！

回顾

- 图灵归约回顾：
- $f \leq_T g$ 表示存在一种可以调用g的图灵机，可以用来计算f。
- 我们在讨论**多项式时间**内的事情，所以我们的故事一定要只能在多项式时间内发生。
- 多项式时间图灵归约定义（cook reduction)。
- $f \leq_T^P g$ 表示存在一种**多项式时间**的可以调用g的图灵机，可以用来计算f。
  - 这个图灵机的运行步骤是多项式的。
  - 这个图灵机调用g的次数也是多项式的。

小插曲

- $f \leq_T^P g$ 表示存在一种多项式时间的可以调用g的图灵机，可以用来计算f。
  - 这个图灵机的运行步骤是多项式的。
  - 这个图灵机**调用g的次数**也是多项式的。
- 在P，NP的研究中，还有另一种规约的定义叫做Karp规约$(f \leq_k g)$，它规定：
  - 这个图灵机运行步骤是多项式的。
  - 这个图灵机只能在**最后调用一次g**，然后直接输出结果。
- 其实很多理论都是基于Karp规约的，但是我们这门课先带大家理解Cook规约。

寻找一个难的问题下手

- 什么样的问题足够难?
- 如果一个问题比所有NP问题都难，它够难了吧!
- 如果我们把这个问题在多项式时间内解决，说明什么?
- 另一个角度，如果我们想证明 $P \neq NP$，我们应该关注
  - 最难的NP问题。
  - 它比所有NP问题都难。
  - 它是一个NP问题。

给这些足够难的问题一个分类

- NP-hard问题
  - 我们称一个问题f是NP-Hard，如果对于**所有**NP问题g，$g \leq_? f$。
  - 在这门课中，我们可以用$\leq_T^P$定义。
  - 在复杂度理论中，常见的是使用$\leq_k$定义。
  - 但他们**都达到**了效果：
  - 如果有我们可以多项式解决任何一个NP-Hard问题，我们就能多项式解决所有NP问题。
- NP-complete问题
  - 如果f是NP-Hard，
  - 并且f是NP，
  - 我们称f是NP-Complete。

这意味着什么？

- 如果我们证明了一个问题是NP-hard，
- 一方面，如果我们想研究复杂度，
  - 我们可以看看他到底难在哪里。
- 一方面，如果我们只是想对这个问题设计算法，
  - 假如我们相信$P \neq NP$，这就说明我们想快速且准确得解决这个问题是不可能的。

一切的开始

[Cook-Levin Theorem] SAT 问题是 NP-complete.

什么是SAT问题?

- 布尔表达式：一个由变量，与(∧)、或(∨)、非(¬)构成的表达式。
  - 举例：$(x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2)$
- 合取范式(CNF)：许多括号组成的布尔表达式
  - 括号之间由与(∧)相连
  - 括号之内存在一些正变量(x_1)或者逆变量$(\neg x_1)$，并由或(∨)相连。
  - 举例：$(x_1 \vee x_3 \vee \neg x_4)$, $(x_2 \vee \neg x_3)$ and $(\neg x_1 \vee \neg x_2)$
- SAT问题：输入一个CNF，问你是否有一种$x_1 \sim x_n$的赋值，使得这个CNF最后答案是true。
  - 对以上的例子：$x_1 = \text{true}, x_2 = \text{false}, x_3 = \text{false}.$

为什么SAT问题是NP-complete

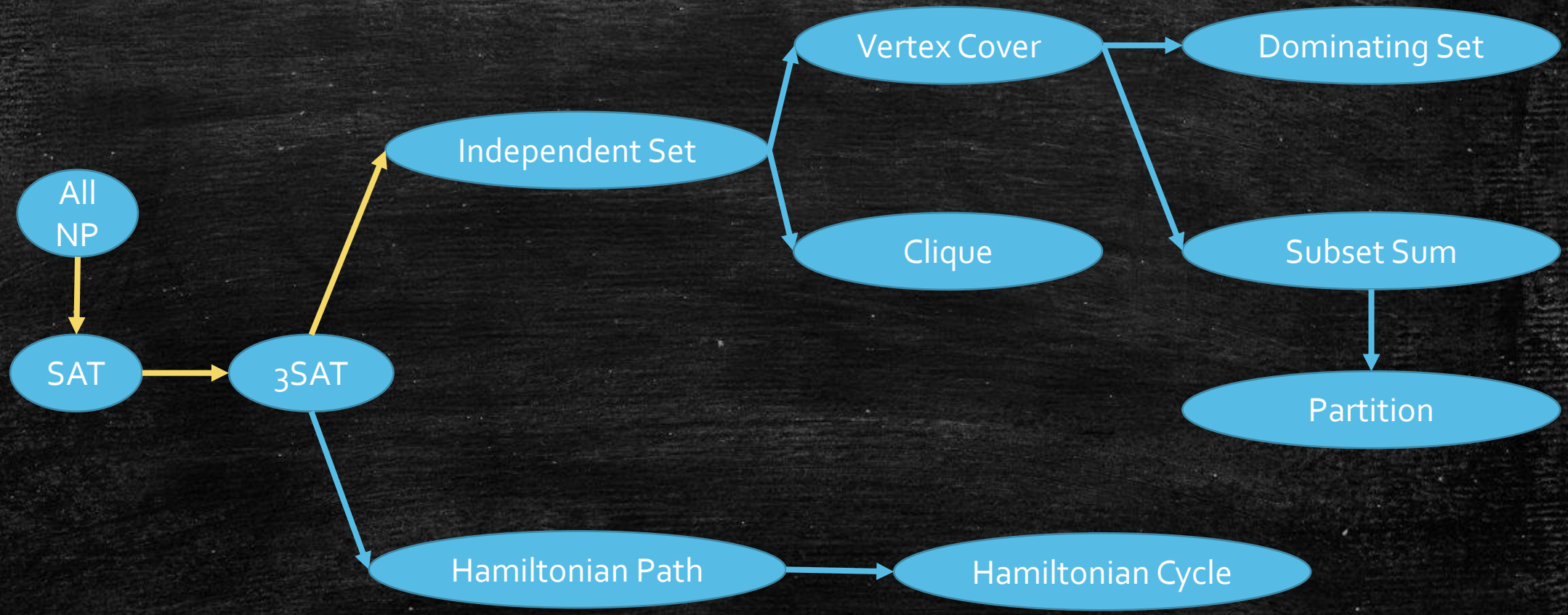
- 问题1：证明它是NP的。
- 问题2：证明它是NP-hard。
- 问题2能成功的简单原因，如果我们会做SAT，我们就能模拟非确定型图灵机，或者说，我们能模拟同步运行所有不同输入的验证器。
- 由于时间问题，我们就假设这件事正确！

如何利用这个结论证明别的问题是NP-hard?

任务变得简单了

- 如何证明某个问题f是NP-hard?
- 关键性质：这两种规约都具有传递性。
- 寻找一个已有的NP-hard问题，如SAT，然后证明 SAT$\leq_T^P f$，或 SAT$\leq_k f$。

一些规约的展示


All
NP
SAT
3SAT
Independent Set
Vertex Cover
Dominating Set
Clique
Subset Sum
Partition
Hamiltonian Path
Hamiltonian Cycle

来看两个小例子

3-SAT 问题

什么是3-SAT问题

- 规定输入的CNF一定是一个3-CNF。
  - 一个括号里最多只有三个正（逆）变量。
  - 例子：
    - 3CNF：$(x_1 \vee x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2)$
    - 不是3CNF：$(x_1 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4)$
- 提问，你觉得以下哪个题目证明起来简单
  - $\mathrm{SAT} \leq_T^P 3-\mathrm{SAT}$
  - $3-\mathrm{SAT} \leq_T^P \mathrm{SAT}$

证明SAT$\leq_T^P$3-SAT

- 备注：其实我们证明了$\leq_K$
- 基本思路：
  - 输入一个CNF，我们构造一个与之等价的3-CNF，使得我们的3-SAT判定器能够通过判定这个3-CNF，来达到判定CNF的效果。
- 关键技巧：
  - 表达式的分割：
  - $(x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4) = (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee \neg x_3 \vee \neg x_4)$
  - $(x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4 \vee x_5 \vee x_6) = (x_1 \vee x_2 \vee y_1) \wedge (\neg y_1 \vee \neg x_3 \vee y_2) \wedge (\neg y_2 \vee \neg x_4 \vee y_3) \wedge (\neg y_3 \vee x_5 \vee x_6)$

转换规则

- $(\ell_1 \vee \cdots \vee \ell_k) = (\ell_1 \vee \ell_2 \vee y_1) \wedge (\neg y_1 \vee \ell_3 \vee y_2) \wedge \cdots \wedge (\neg y_{k-2} \vee \ell_{k-1} \vee \ell_k)$
- 如果存在一种赋值让$(\ell_1 \vee \cdots \vee \ell_k)$为真，那么我们可以通过设置$y$让右式也为真。
- 如果存在一种赋值让右式为真，那么这个赋值直接就能让左式为真（取出x的部分）。

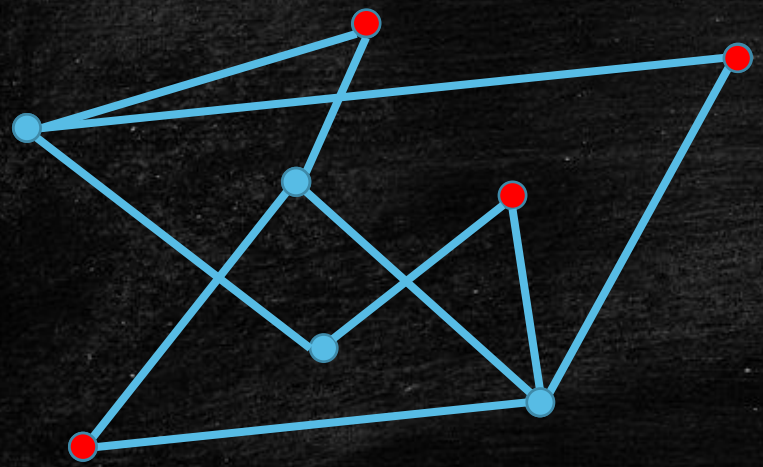
总结

- 我们的规约如下
- 收到输入CNF
- 通过添加y，把CNF中所有大于三个元素的括号进行分割。
- 每个括号会需要一个y和一个$\neg y$，转换时间和规模都保证在多项式以内。
- 得到3CNF表达式通用3-SAT判定其为yes或者no。
- 可以证明，3-SAT回答yes，代表该CNF是yes输入，3-SAT回答no，该CNF是no输入。

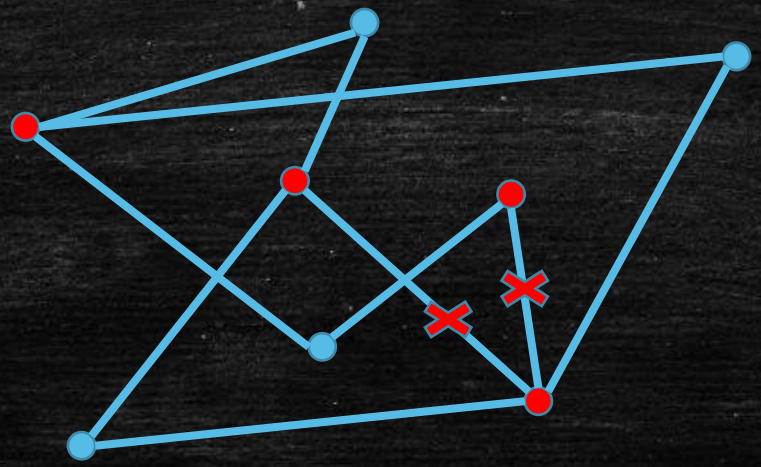
独立集问题

独立集问题

- 输入一个图$G = (V, E)$，以及一个数k。
- 输出该图是否存在一个k个点的独立集合。
- 独立集的定义：
  - 一个顶点的子集$S \subset V$，且S中的顶点互相不相连。



独立集



非独立集

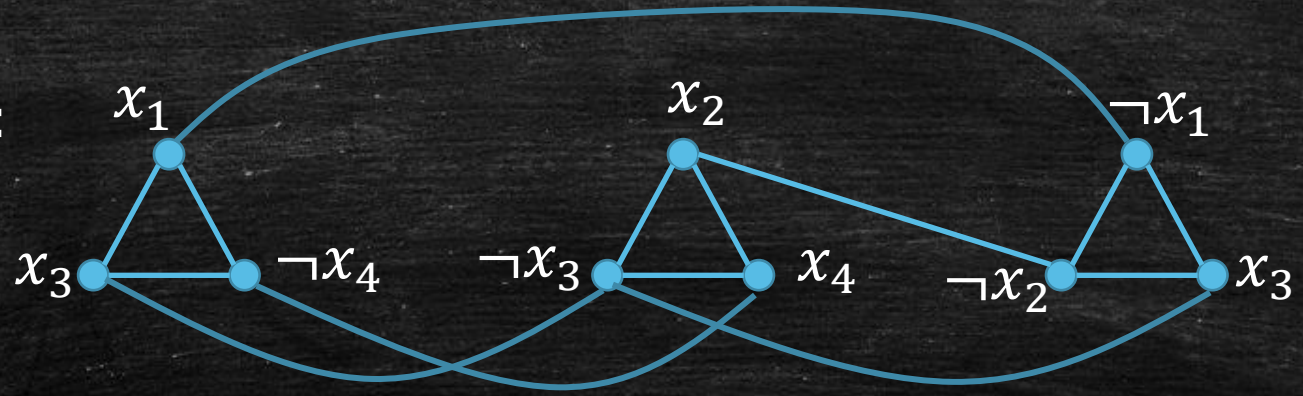
证明 3-SAT$\leq_T^P$ 独立集

- 依然备注：其实我们证明了$\leq_K$
- 基本思路：
  - 输入一个n个变量m个括号的3-CNF，我们构建一个等价的图，如果该图存在m个点的独立集，就代表3-CNF存在一组赋值为真，反之亦然。

规约展示

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

G:

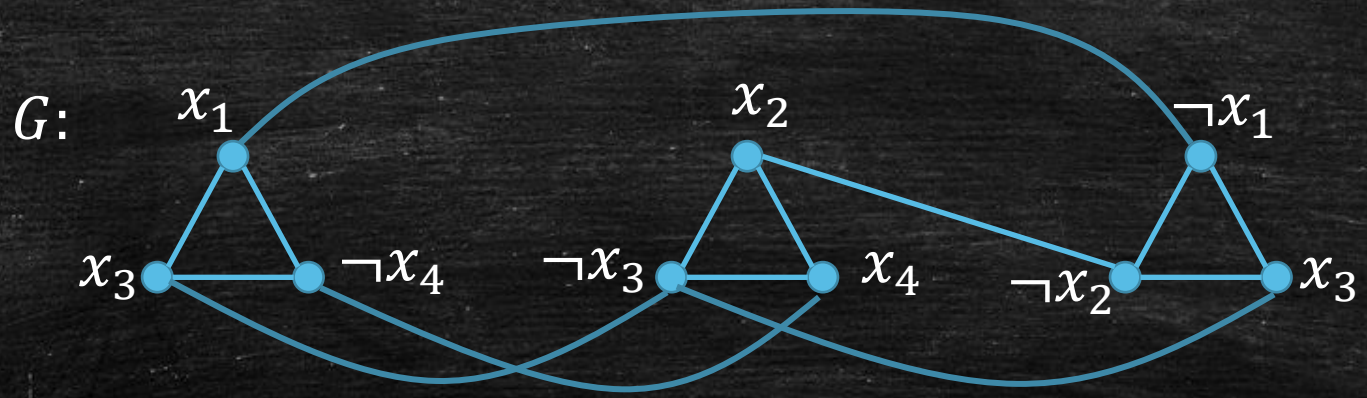


$k = 3 = m$

- 第一个方向的证明：
- 如果存在一种一个k个点的独立集，我们把设定x使得这k个元素都为真，其他随便。
- 他们不会矛盾，且每个括号一定有一个元素为真，所以这个赋值为真。

规约展示

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



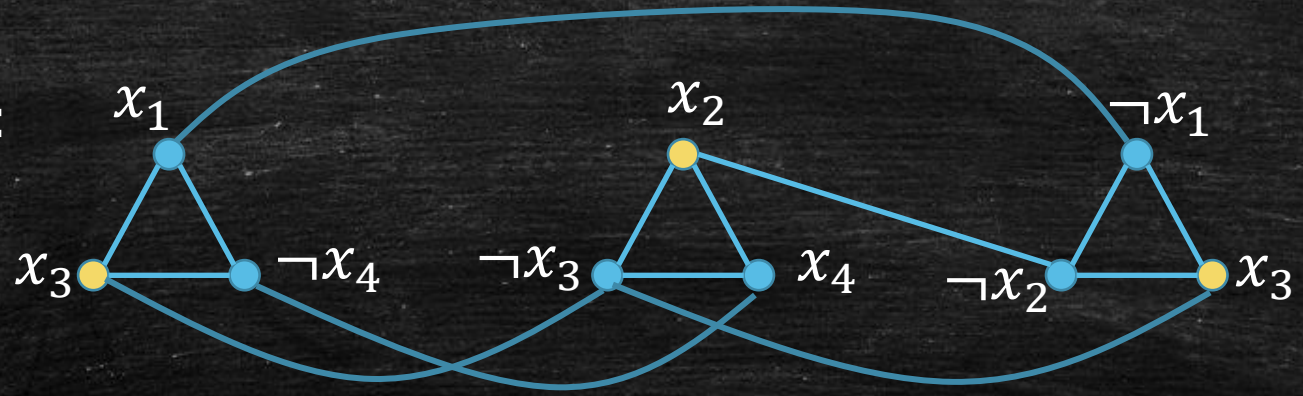
$k = 3 = m$

- 第二个方向的证明：
- 如果存在一种赋值使得3-CNF为真，那种赋值里肯定能够使得每个括号里至少一个元素为真，我任意选择其中一个。
- 这样我就选出了m个顶点，且他们不会矛盾。

规约展示

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

G:



$k = 3$

- 例子: $x_1 = x_2 = x_3 = x_4 = \text{true}$ makes $\phi = \text{true}$
- 每个括号选出一个代表
  - $(x_1 \vee x_3 \vee \neg x_4)$
  - $(x_2 \vee \neg x_3 \vee x_4)$
  - $(\neg x_1 \vee \neg x_2 \vee x_3)$