CSE 421 Winter 2025
Lecture 23:
NP

Nathan Brunelle

http://www.cs.uw.edu/421

Polynomial Time Reduction

Defn: We write $\boldsymbol{A} \leq_{\boldsymbol{P}} \boldsymbol{B}$ iff there is an algorithm for $\boldsymbol{A}$ using a 'black box' (subroutine or method) that solves $\boldsymbol{B}$ that

- uses only a polynomial number of steps, and
- makes only a polynomial number of calls to a method for $\boldsymbol{B}$.

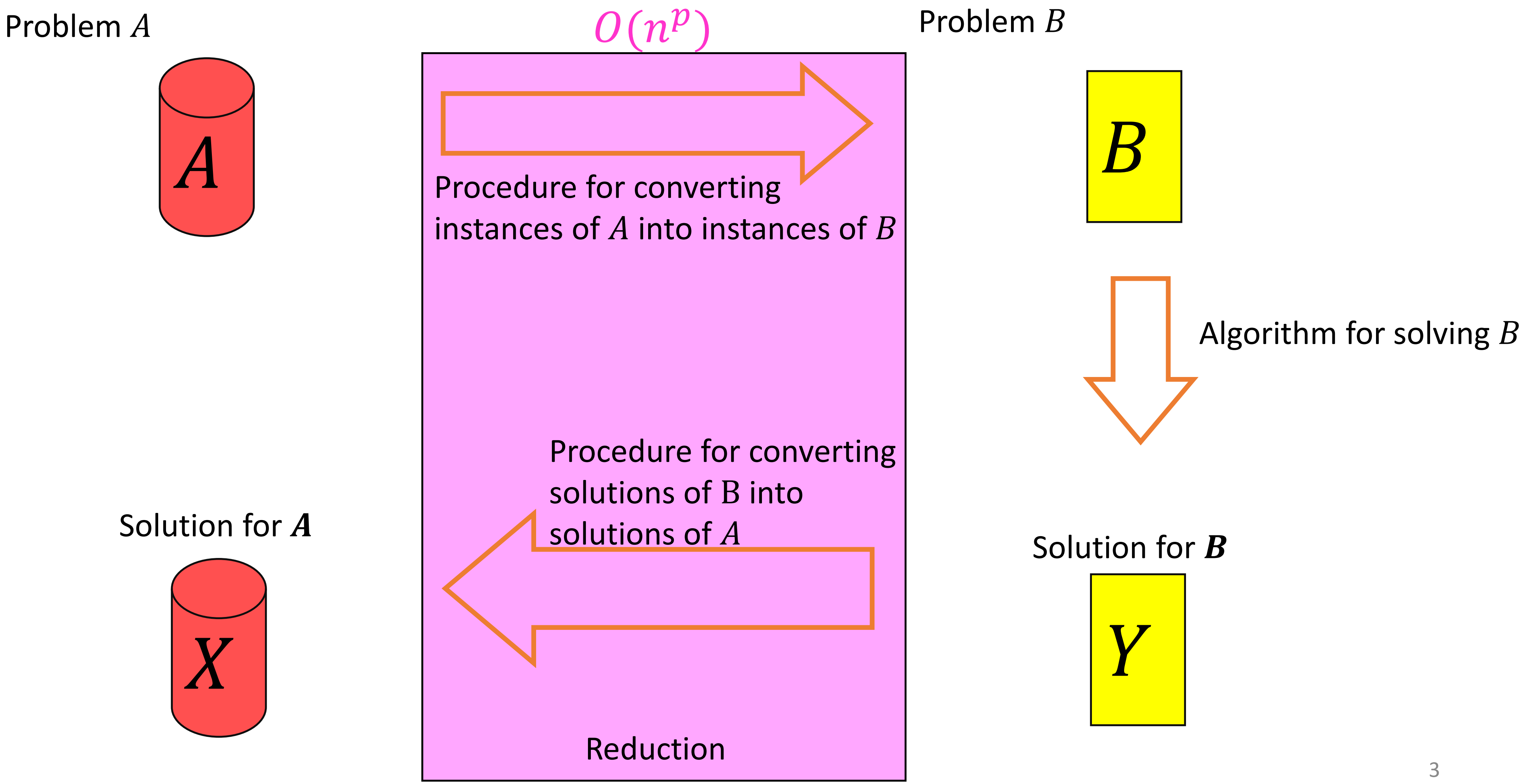
Theorem: If $\boldsymbol{A} \leq_{\boldsymbol{P}} \boldsymbol{B}$ then a poly time algorithm for $\boldsymbol{B}$ $\Rightarrow$ poly time algorithm for $\boldsymbol{A}$

Proof: Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!

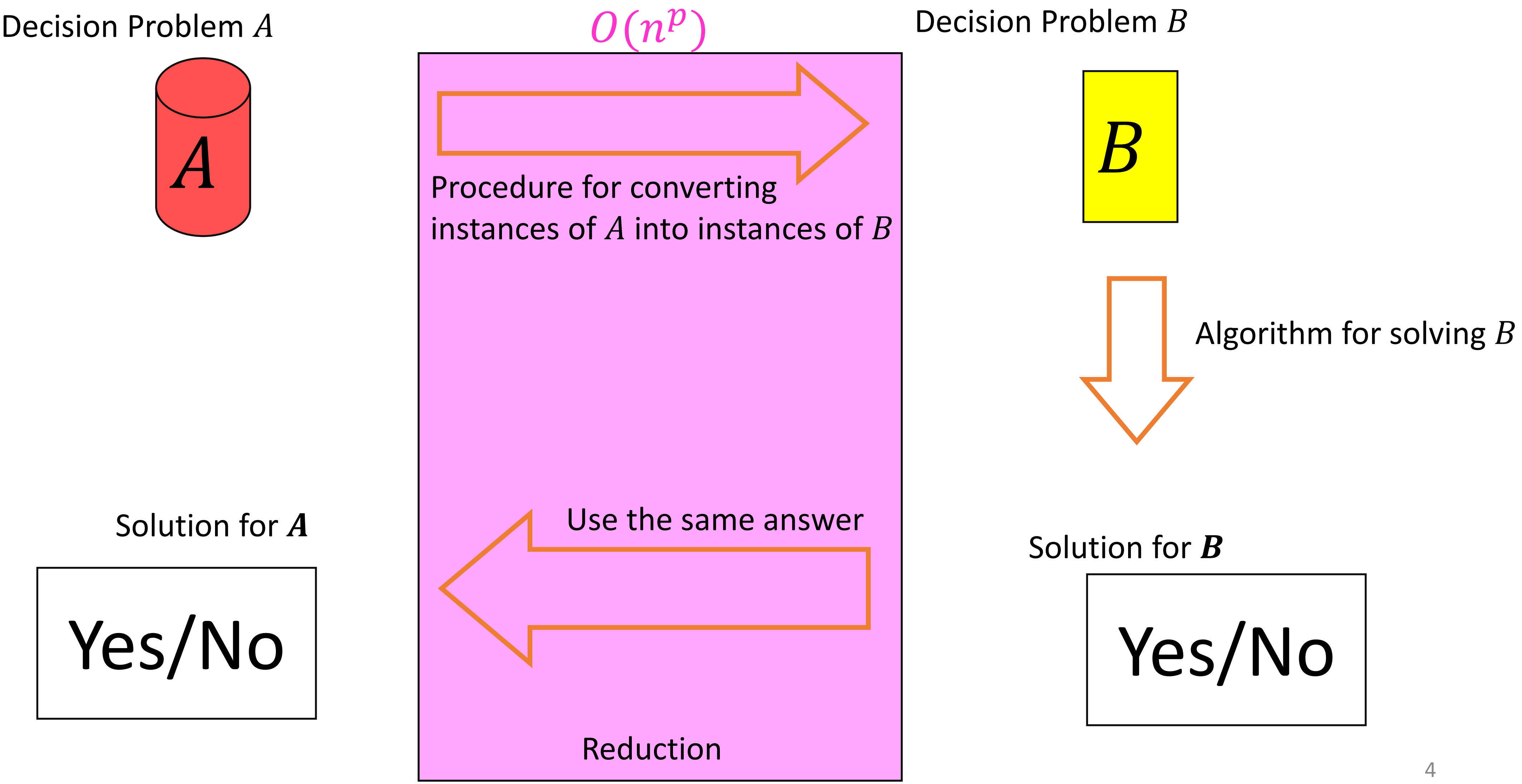
Corollary: If you can prove there is no fast algorithm for $\boldsymbol{A}$, then that proves there is no fast algorithm for $\boldsymbol{B}$.

Intuition for "$\boldsymbol{A} \leq_{\boldsymbol{P}} \boldsymbol{B}$": "$\boldsymbol{B}$ is at least as hard* as $\boldsymbol{A}$" *up to polynomial-time slop.

Polynomial Time Reductions



Polynomial Time Reductions (Decision Problems)



Let’s do a reduction

4 steps for reducing (decision problem) $\boldsymbol{A}$ to problem $\boldsymbol{B}$

1. Describe the reduction itself
   - i.e., the function that converts the input for $\boldsymbol{A}$ to the one for problem $\boldsymbol{B}$.
   - i.e., describe what the top arrow in the pink box does
2. Make sure the running time would be polynomial
   - In lecture, we’ll sometimes skip writing out this step.
3. Argue that if the correct answer (to the instance for $\boldsymbol{A}$) is **YES**, then the input we produced is a **YES** instance for $\boldsymbol{B}$.
4. Argue that if the correct answer (to the instance for $\boldsymbol{A}$) is **NO**, then the input we produced is a **NO** instance for $\boldsymbol{B}$.

Let's do a reduction

4 steps for reducing (decision problem) $\boldsymbol{A}$ to problem $\boldsymbol{B}$

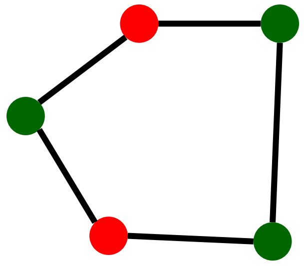
1. Describe the reduction itself
   - i.e., the function that converts the input for $\boldsymbol{A}$ to the one for problem $\boldsymbol{B}$.
   - i.e., describe what the top arrow in the pink box does
2. Make sure the running time would be polynomial
   - In lecture, we'll sometimes skip writing out this step.
3. Argue that if the correct answer (to the instance for $\boldsymbol{A}$) is **YES**, then the input we produced is a **YES** instance for $\boldsymbol{B}$.
4. Argue that if the input we produced is a **YES** instance for $\boldsymbol{B}$ then the correct answer (to the instance for $\boldsymbol{A}$) is **YES**.

Contrapositive

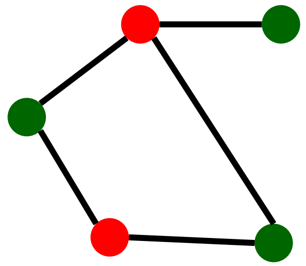
Reduce 2Color to 3Color

Defn: A undirected graph $G = (V, E)$ is k-colorable iff
we can assign one of k colors to each vertex of V s.t. for
$(u, v) \in E$, their colors, $\chi(u)$ and $\chi(v)$, are different.
"edges are not monochromatic"

2Color: Given: an undirected graph G
Is G 2-colorable?

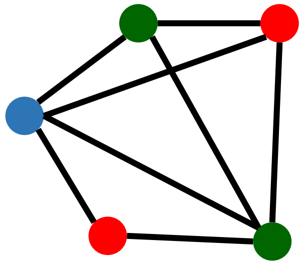


No

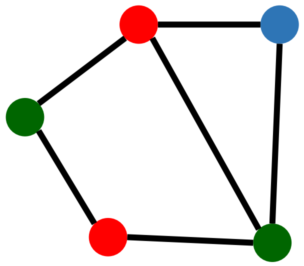


Yes

3Color: Given: an undirected graph G
Is G 3-colorable?



No



Yes

2Color $\leq_P$ 3Color

- Given a graph G figure out whether it can be 2-colored, by using an algorithm that figures out whether it can be 3-colored.

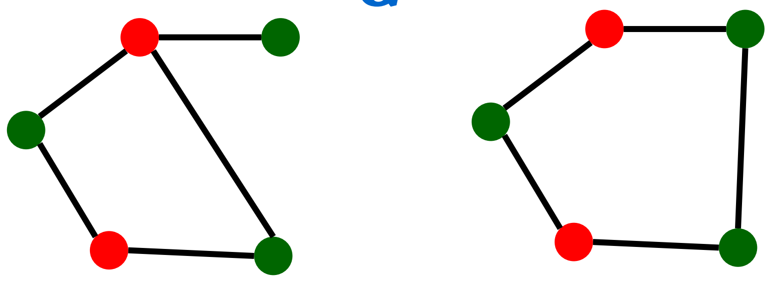
Usual outline:

- Transform G into an input for the **3Color** algorithm
- Run the **3Color** algorithm
- Use the answer from the **3Color** algorithm as the answer for G for **2Color**

Reducing 2Color to 3Color

2Color

G



$O(n^p)$

Transform given graph ***G*** such that the output graph ***H*** was 3-colorable if and only if ***G*** was 2-colorable

Use the same answer

Reduction

3Color

H

Algorithm for solving **3Color**

Solution for **2Color**

Yes/No

Solution for **3Color**

Yes/No

Reduction

If we just ask the **3Color** algorithm about $\boldsymbol{G}$, if $\boldsymbol{G}$ is 3-colorable but not 2-colorable it will give the wrong answer because it has the 3rd color available.

Idea: Add extra vertices and edges to to **G** to force the 3rd color to be used there but not on $\boldsymbol{G}$

Reduction $\boldsymbol{f}$: Add one extra vertex $\boldsymbol{v}$ and attach it to **everything** in $\boldsymbol{G}$.

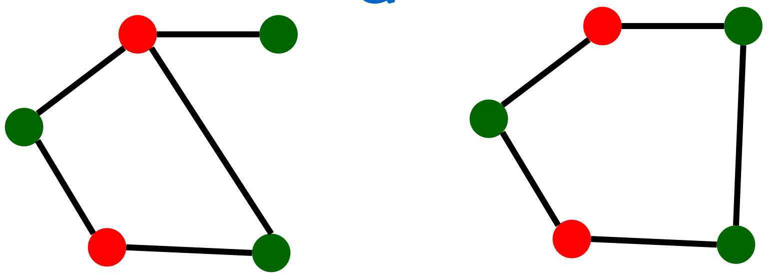
Write $\boldsymbol{H} = \boldsymbol{f}(\boldsymbol{G})$.

($\boldsymbol{f}$ is polynomial time computable.)

Reducing 2Color to 3Color

2Color

$\boldsymbol{G}$



$O(n^p)$

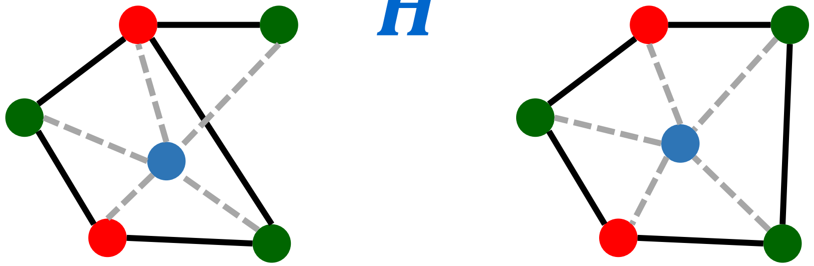
Add a new node to $\boldsymbol{G}$, connect every node to it

Use the same answer

Reduction

3Color

$\boldsymbol{H}$



Algorithm for solving **3Color**

Solution for **2Color**

Yes/No

Solution for **3Color**

Yes/No

Let's do a reduction

4 steps for reducing (decision problem) $\boldsymbol{A}$ to problem $\boldsymbol{B}$

1. Describe the reduction itself
   - i.e., the function that converts the input for $\boldsymbol{A}$ to the one for problem $\boldsymbol{B}$.
2. Make sure the running time would be polynomial
   - In lecture, we'll sometimes skip writing out this step.
3. Argue that if the correct answer (to the instance for $\boldsymbol{A}$) is **YES**, then the input we produced is a **YES** instance for $\boldsymbol{B}$.
4. Argue that if the input we produced is a **YES** instance for $\boldsymbol{B}$ then the correct answer (to the instance for $\boldsymbol{A}$) is **YES**.

Correctness

Two statements to prove (two directions):

If $\boldsymbol{G}$ is a **YES** for **2Color** ($\boldsymbol{G}$ is 2-colorable) then $\boldsymbol{H}$ is a **YES** for **3Color** ($\boldsymbol{H}$ is 3-colorable)

Suppose $\boldsymbol{G}$ is 2-colorable: $\boldsymbol{G}$ has a 2-coloring $\boldsymbol{\chi}$ so edges of $\boldsymbol{G}$ have different colored endpoints. We get a 3-coloring of $\boldsymbol{H}$ by using $\boldsymbol{\chi}$ for all the copies of original vertices of $\boldsymbol{G}$ and a 3rd color for the extra vertex $\boldsymbol{v}$: Original edges of $\boldsymbol{G}$ in $\boldsymbol{H}$ have different colored endpoints; the extra edges too. So $\boldsymbol{H}$ is 3-colorable.

If $\boldsymbol{H}$ is a **YES** for **3Color** ($\boldsymbol{H}$ is 3-colorable) then $\boldsymbol{G}$ is a **YES** for **2Color** on ($\boldsymbol{G}$ is 2-colorable)

Suppose $\boldsymbol{H}$ is 3-colorable: Consider a 3-coloring $\boldsymbol{\chi'}$ of $\boldsymbol{H}$. Consider the extra vertex $\boldsymbol{v}$ in $\boldsymbol{H}$ that was added to $\boldsymbol{G}$. For every vertex $\boldsymbol{u}$ of $\boldsymbol{G}$, we have an edge $(\boldsymbol{u}, \boldsymbol{v})$ so $\boldsymbol{\chi'(u) \neq \chi'(v)}$. This means that every vertex $\boldsymbol{u}$ of $\boldsymbol{G}$ is colored with one of the two colors other than $\boldsymbol{\chi'(v)}$. So we can use $\boldsymbol{\chi'}$ as a 2-coloring of $\boldsymbol{G}$ since all those edges had different colored endpoints in $\boldsymbol{H}$. So $\boldsymbol{G}$ is 2-colorable.

Write two separate arguments

The two directions we covered actually prove an if and only if.

To make sure you handle both directions, I **strongly** recommend:

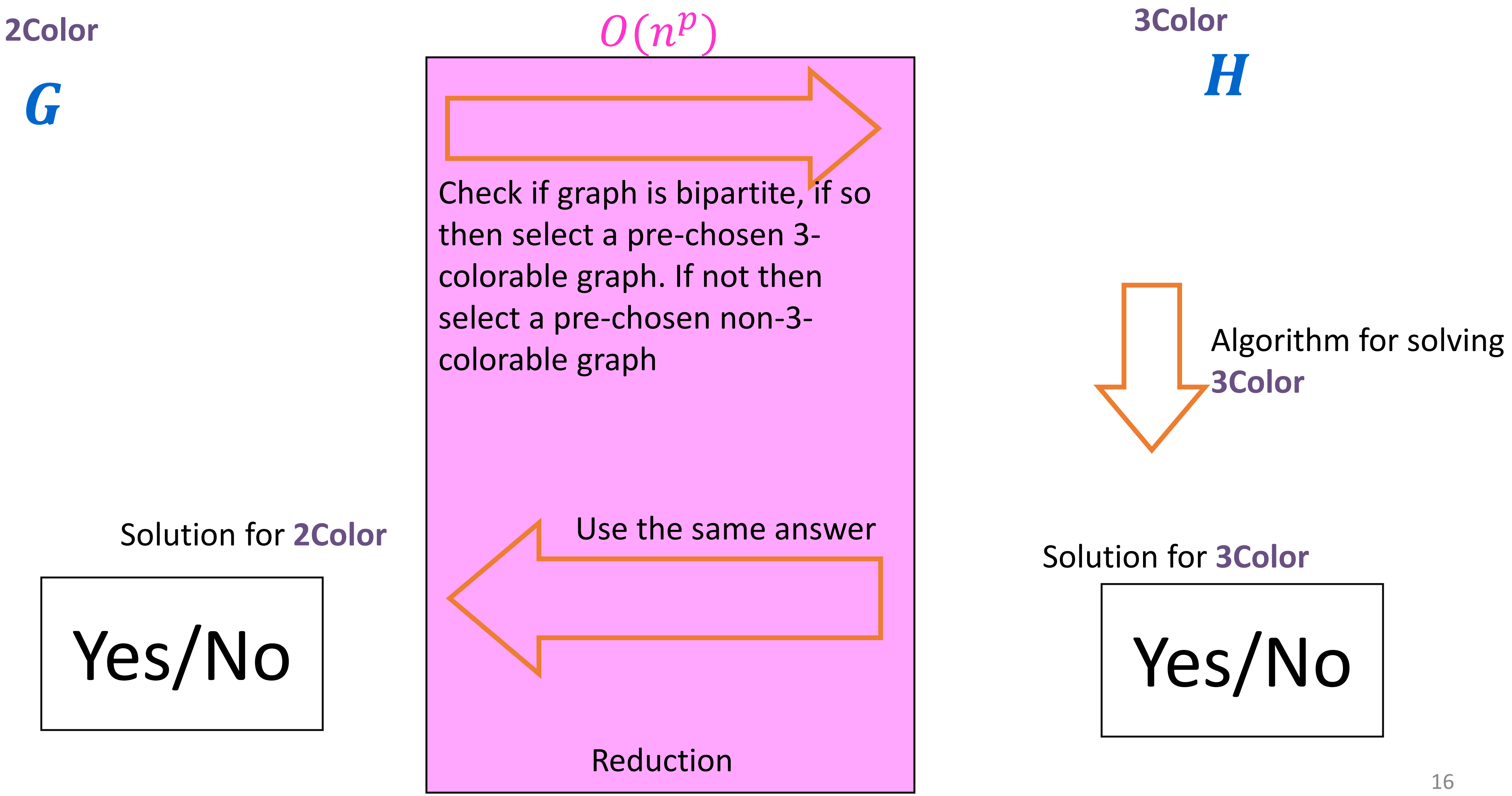
- Always do two separate proofs! (Don’t try to prove both directions at once, don’t refer back to the prior proof and say “for the same reason”. There are usually subtle differences.)
- Don’t use contradiction! (It’s easy to start from the wrong spot and accidentally prove the same direction twice without realizing it.)

Another proof of 2Color $\leq_P$ 3Color

We had an $O(n + m)$ time algorithm for **2Color** based on BFS.

Simply solve the **2Color** problem without making any calls to a **3Color** method!

Reducing 2Color to 3Color



Two More Reductions

Independent-Set:

Given a graph $G = (V, E)$ and an integer k

Is there a $U \subseteq V$ with $|U| \geq k$ such that no two vertices in U are joined by an edge? (U is called an independent set.)

Clique:

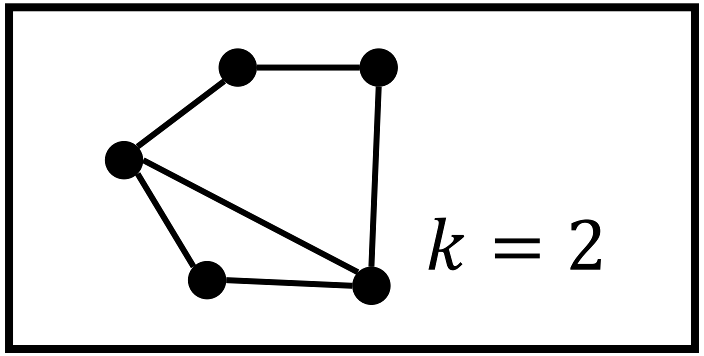
Given a graph $G = (V, E)$ and an integer k

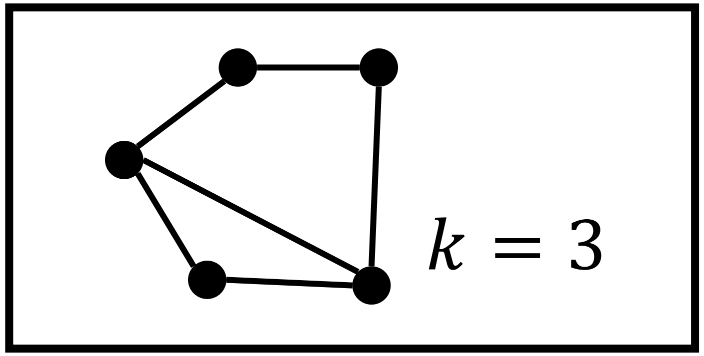
Is there a $U \subseteq V$ with $|U| \geq k$ such that every pair of vertices in U is joined by an edge? (U is called a clique.)

Claim: **Independent-Set** $\leq_P$ **Clique**

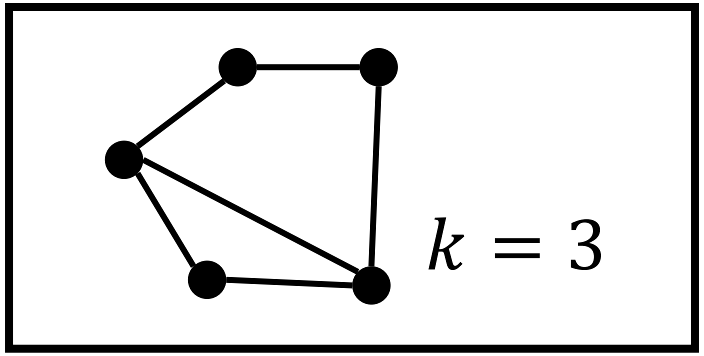
Examples

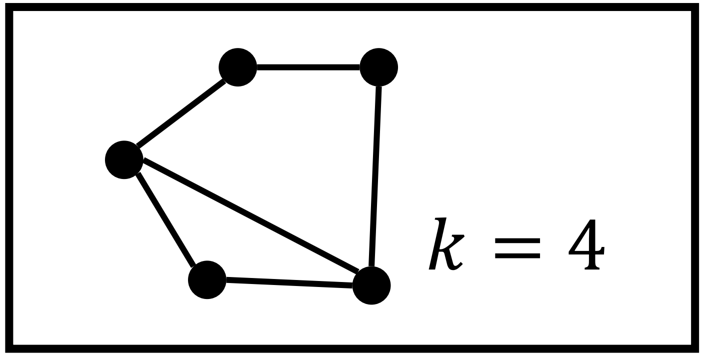
- Independent Set





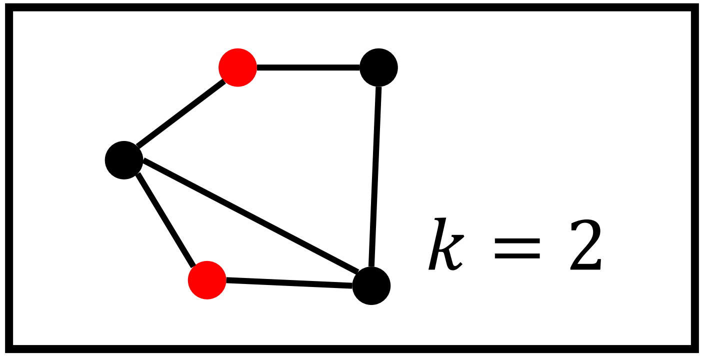
- Clique



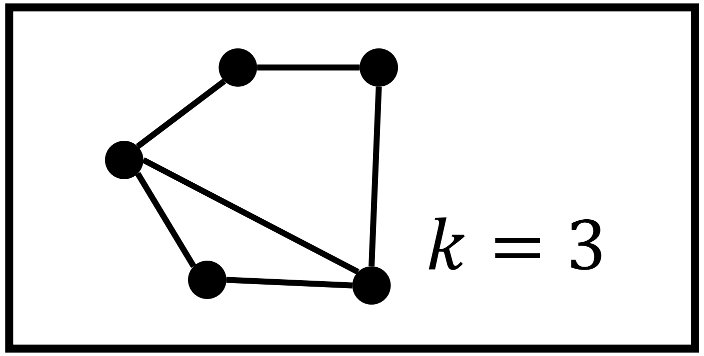


Examples

- Independent Set

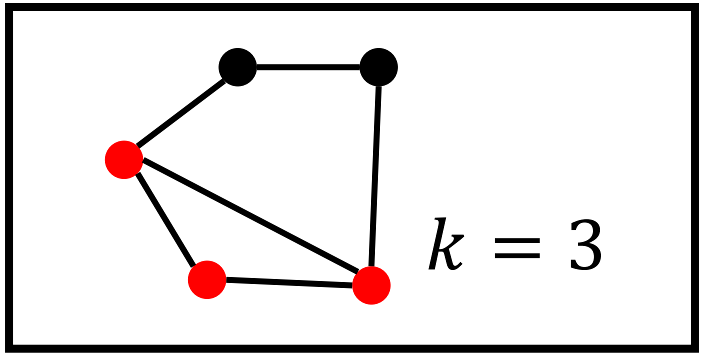


Yes

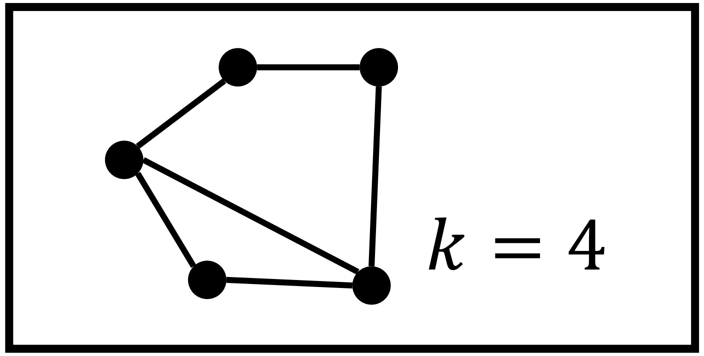


No

- Clique



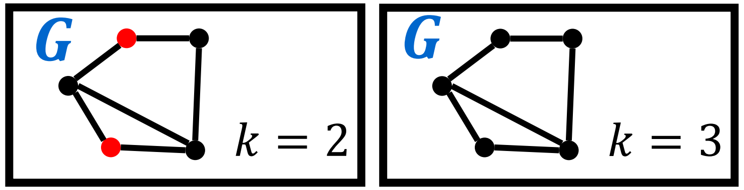
Yes



No

Reducing Independent Set to Clique

Independent Set



$O(n^p)$

Transform given graph G and number k into G' and k' such that G has an Independent Set of size k iff G' has a Clique of size k'

Use the same answer

Reduction

Clique

G', k'

Algorithm for solving **Clique**

Solution for **Clique**

Yes/No

Solution for **Independent Set**

Yes/No

Independent-Set $\leq_P$ Clique

Given:

- (G, k) as input to **Independent-Set** where $G = (V, E)$

Use function f that transforms (G, k) to (G', k) where

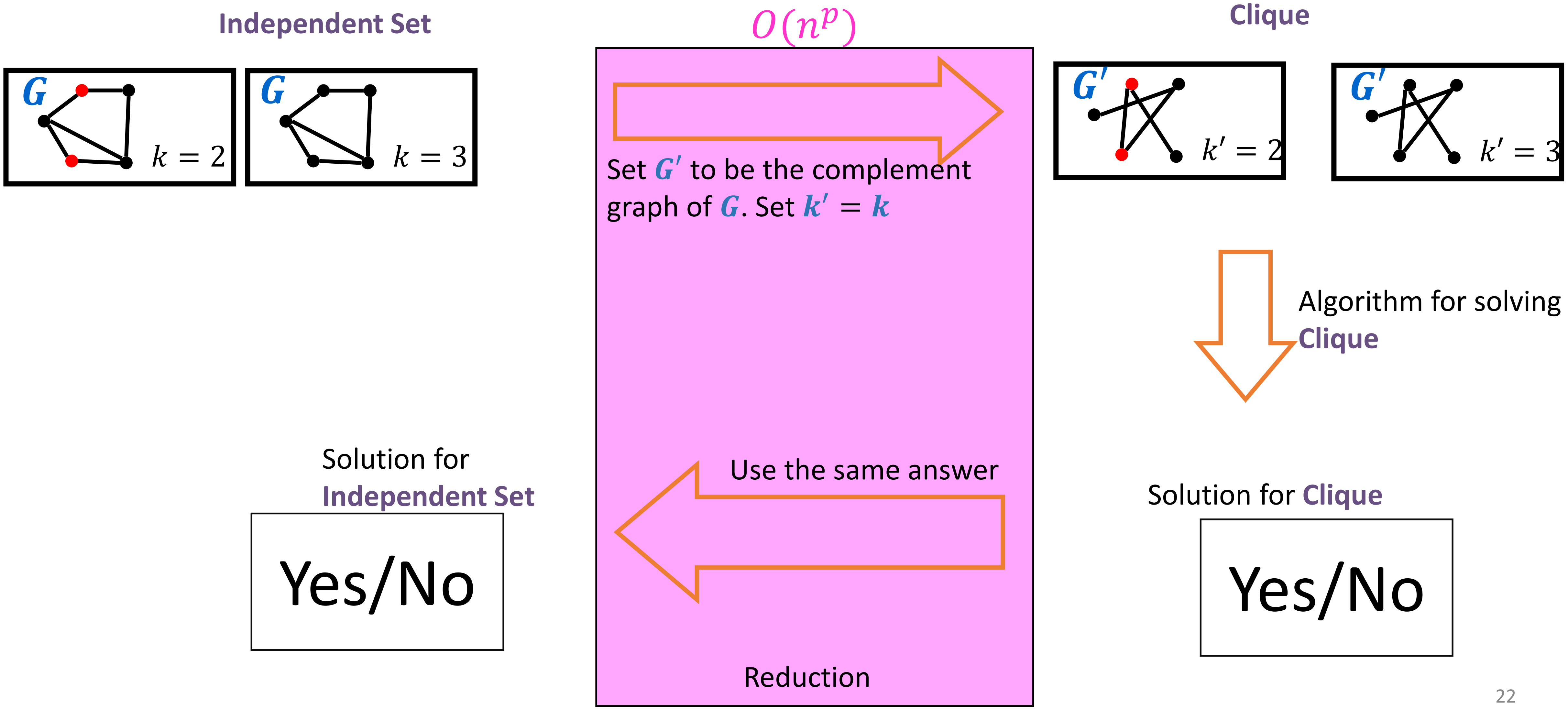
- $G' = (V, E')$ has the same vertices as G but E' consists of **precisely** those edges on V that are not edges of G.

graph complement

From the definitions, U is an independent set in G

$\Leftrightarrow$ U is a clique in G'

Reducing Independent Set to Clique



Clique $\leq_P$ Independent Set

Given:

- (G, k) as input to **Clique** where $G = (V, E)$

Use function f that transforms (G, k) to (G', k) where

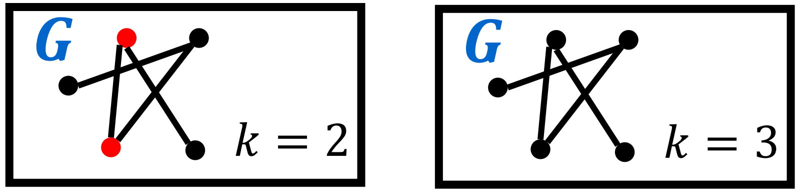
- $G' = (V, E')$ has the same vertices as G but E' consists of **precisely** those edges on V that are not edges of G.

From the definitions, U is an clique in G

$\Leftrightarrow$ U is an independent set in G'

Reducing Clique to Independent Set

Clique



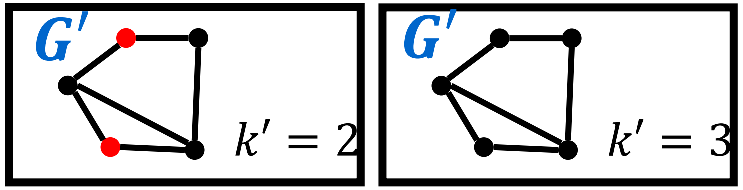
$O(n^p)$

Set G' to be the complement graph of G. Set $k' = k$

Use the same answer

Reduction

Independent Set



Algorithm for solving **Independent Set**

Solution for **Clique**

Yes/No

Solution for **Independent Set**

Yes/No

Another Reduction

Vertex-Cover:

Given a graph $G = (V, E)$ and an integer k

Is there a $W \subseteq V$ with $|W| \leq k$ such that every edge of G has an endpoint in W?

(W is a vertex cover, a set of vertices that covers E.)

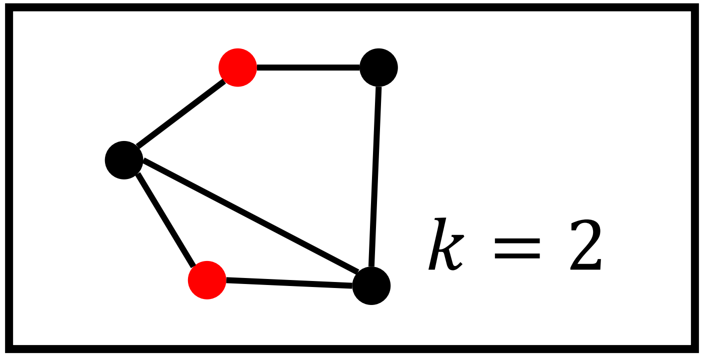
Claim: **Independent-Set** $\leq_P$ **Vertex-Cover**

Lemma: In a graph $G = (V, E)$ and $U \subseteq V$

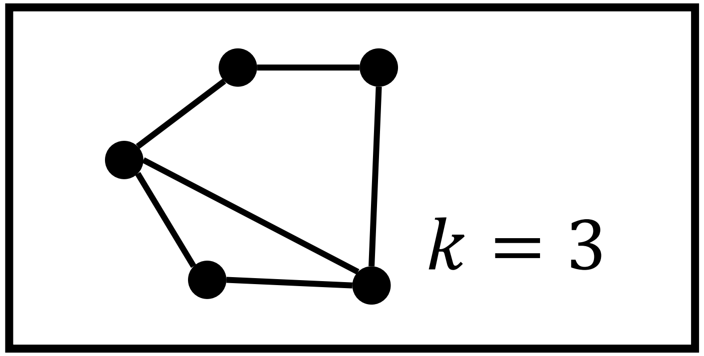
U is an independent set $\Leftrightarrow$ $V - U$ is a vertex cover

Examples

- Independent Set

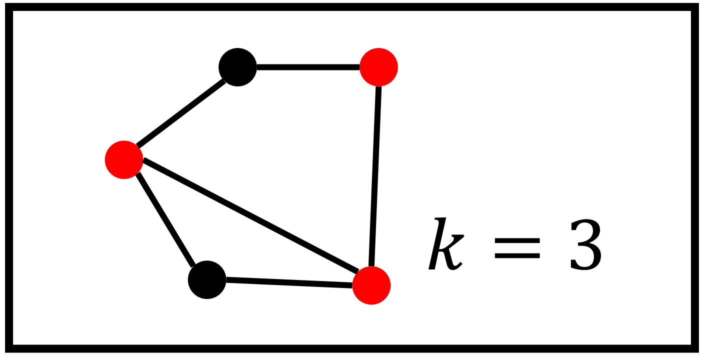


Yes

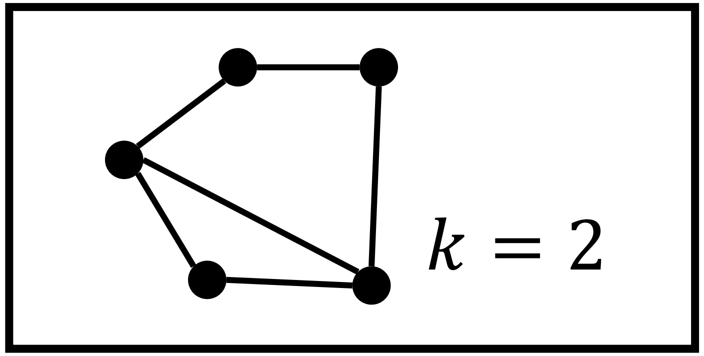


No

- Vertex Cover



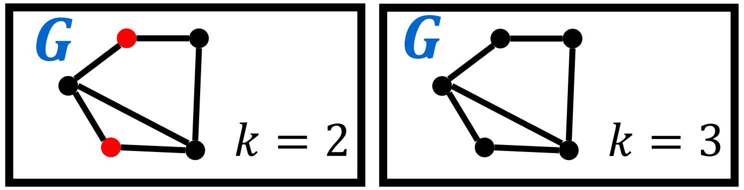
Yes



No

Reducing Independent Set to Vertex Cover

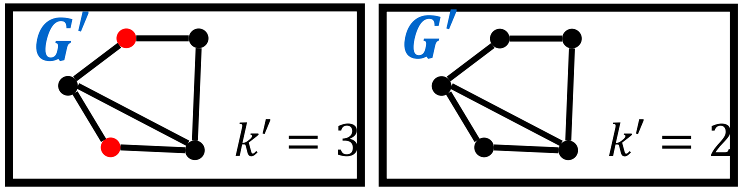
Independent Set



$O(n^p)$

Set $G' = G$. Set $k' = |V| - k$

Vertex Cover



Algorithm for solving **Vertex Cover**

Solution for **Vertex Cover**

Yes/No

Use the same answer

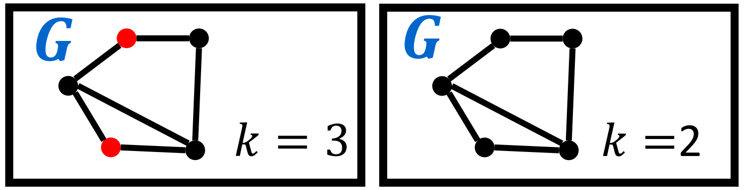
Solution for **Independent Set**

Yes/No

Reduction

Reducing Vertex Cover to Independent Set

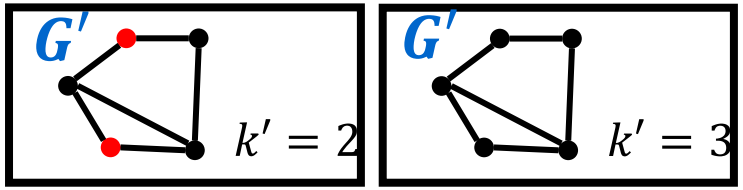
Vertex Cover



$O(n^p)$

Set $G' = G$. Set $k' = |V| - k$

Independent Set



Algorithm for solving **Independent Set**

Use the same answer

Solution for **Vertex Cover**

Yes/No

Solution for **Independent Set**

Yes/No

Reduction

Reduction Idea

Lemma: In a graph $G = (V, E)$ and $U \subseteq V$

U is an independent set $\Leftrightarrow$ $V - U$ is a vertex cover

Proof:

($\Rightarrow$) Let U be an independent set in G

Then for every edge $e \in E$,

U contains at most one endpoint of e

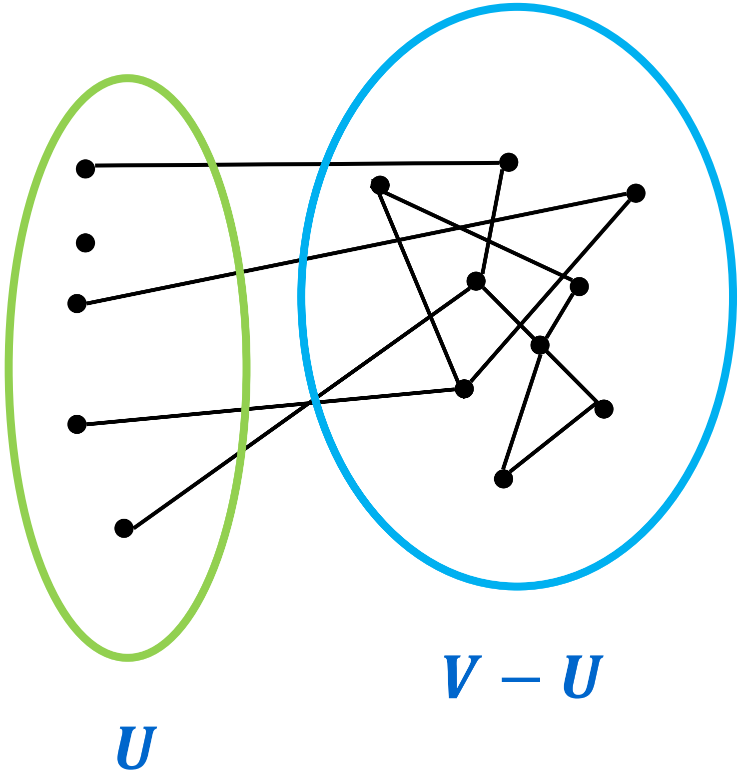
So, at least one endpoint of e must be in $V - U$

So, $V - U$ is a vertex cover

($\Leftarrow$) Let $W = V - U$ be a vertex cover of G

Then U does not contain both endpoints of any edge (else W would miss that edge)

So U is an independent set



Reduction for Clique $\leq_P$ Vertex-Cover

Clique:

Given a graph $G = (V, E)$ and an integer k

Is there a $U \subseteq V$ with $|U| \geq k$ such that every pair of vertices in U is joined by an edge?

(U is called a clique.)

Vertex-Cover:

Given a graph $G = (V, E)$ and an integer k

Is there a $W \subseteq V$ with $|W| \leq k$ such that every edge of G has an endpoint in W?

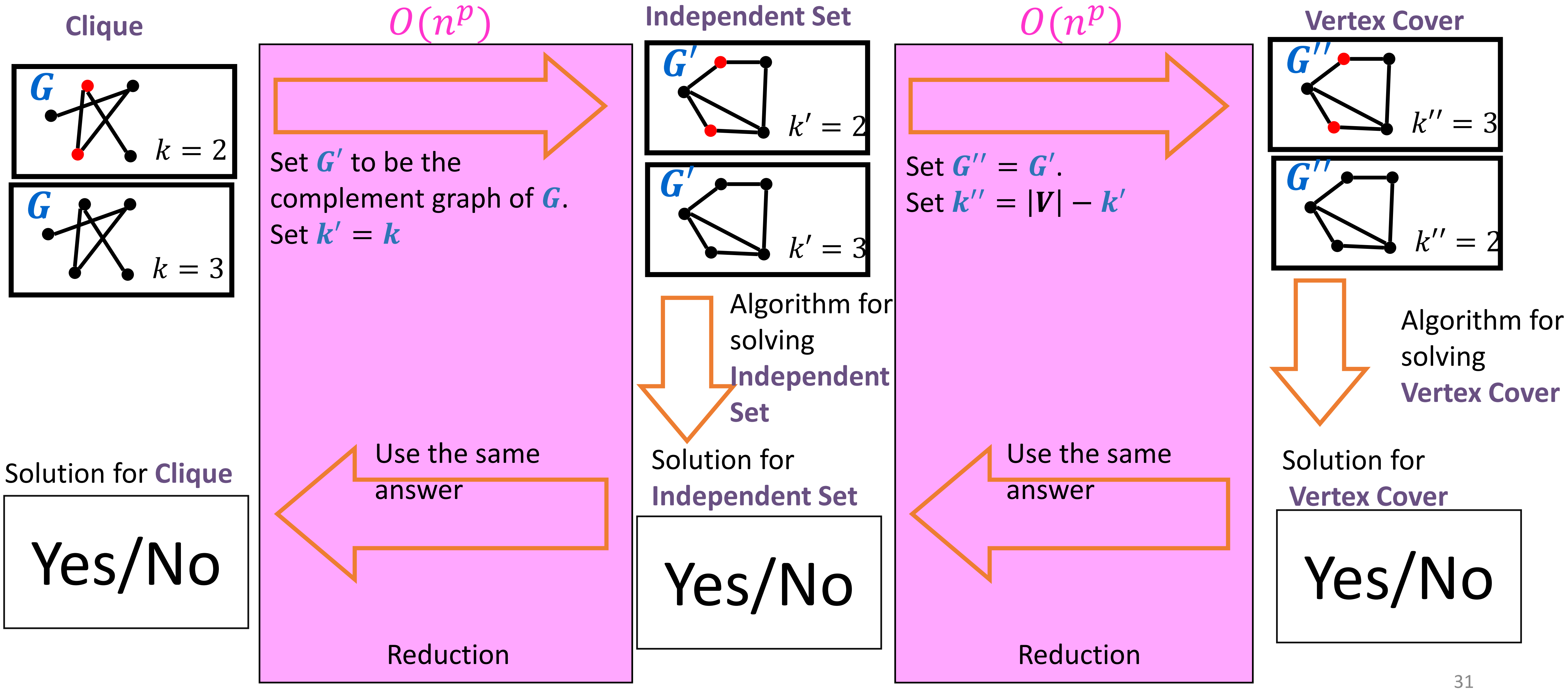
(W is a vertex cover, a set of vertices that covers E.)

Claim: **Clique $\leq_P$ Vertex-Cover**

Idea:

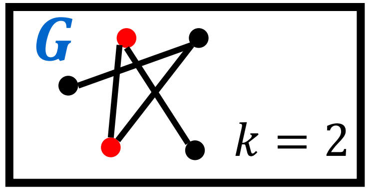
Use **Clique $\leq_P$ Independent-Set** and **Independent-Set $\leq_P$ Vertex-Cover**

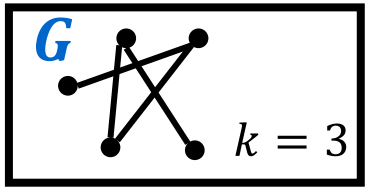
Reducing Clique to Vertex Cover



Reducing Clique to Vertex Cover

Clique





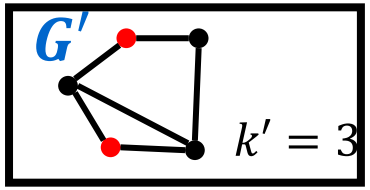
$O(n^p)$

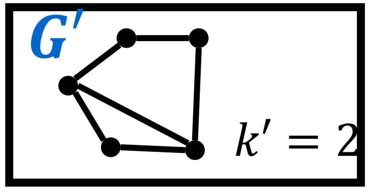
Set G' to be the complement graph of G. Set $k' = |V| - k$

Use the same answer

Reduction

Vertex Cover





Algorithm for solving **Vertex Cover**

Solution for **Clique**

Yes/No

Solution for **Vertex Cover**

Yes/No

Polynomial time

Defn: Let **P** (polynomial-time) be the set of all decision problems solvable by algorithms whose worst-case running time is bounded by some polynomial in the input size.

This is the class of decision problems whose solutions we have called "efficient".

Polynomial Time Reduction

Defn: We write $\boldsymbol{A} \leq_{\mathbf{P}} \boldsymbol{B}$ iff there is an algorithm for $\boldsymbol{A}$ using a 'black box' (subroutine or method) that solves $\boldsymbol{B}$ that

- uses only a polynomial number of steps, and
- makes only a polynomial number of calls to a method for $\boldsymbol{B}$.

Theorem: If $\boldsymbol{A} \leq_{\mathbf{P}} \boldsymbol{B}$ then $\boldsymbol{B} \in \mathbf{P} \Rightarrow \boldsymbol{A} \in \mathbf{P}$

Proof: Not only is the number of calls polynomial but the size of the inputs on which the calls are made is polynomial!

Corollary: If $\boldsymbol{A} \leq_{\mathbf{P}} \boldsymbol{B}$ then $\boldsymbol{A} \notin \mathbf{P} \Rightarrow \boldsymbol{B} \notin \mathbf{P}$.

Theorem: If $\boldsymbol{A} \leq_{\mathbf{P}} \boldsymbol{B}$ and $\boldsymbol{B} \leq_{\mathbf{P}} \boldsymbol{C}$ then $\boldsymbol{A} \leq_{\mathbf{P}} \boldsymbol{C}$

Proof: Compose the reductions: Plug in "the algorithm for $\boldsymbol{B}$ that uses $\boldsymbol{C}$" in place of $\boldsymbol{B}$.

Relationship among the problems so far

Using polynomial time reductions we have found:

- 2Color $\leq_P$ 3Color
- Independent-Set $\leq_P$ Clique
- Clique $\leq_P$ Independent-Set
- Vertex-Cover $\leq_P$ Independent-Set
- Independent-Set $\leq_P$ Vertex-Cover
- Clique $\leq_P$Vertex-Cover
  - By composing the reductions for Clique $\leq_P$Independet-Set and Independent-Set $\leq_P$ Vertex-Cover

- We could also conclude Vertex-Cover $\leq_P$ Clique
- Because we can reduce any of Independent-Set, Vertex-Cover, and Clique to any other:
  - If any one of those belongs to class P then all of them do.
  - In any one of those does not belong to class P then none of them do

Beyond **P**?

Independent-Set, **Clique**, **Vertex-Cover**, and **3Color** are examples of natural and practically important problems for which we don't know any polynomial-time algorithms.

There are many others such as...

DecisionTSP:

Given a weighted graph $\boldsymbol{G}$ and an integer $\boldsymbol{k}$,

Is there a simple path that visits all vertices in $\boldsymbol{G}$ having total weight at most $\boldsymbol{k}$?

and...

Satisfiability

- Boolean variables $x_1, \dots, x_n$
  - taking values in $\{0, 1\}$. 0=false, 1=true
- Literals
  - x_i or $\neg x_i$ for $i = 1, \dots, n$. ($\neg x_i$ also written as $\overline{x_i}$.)
- Clause
  - a logical OR of one or more literals
  - e.g. $(x_1 \vee \neg x_3 \vee x_7 \vee x_{12})$
- CNF formula
  - a logical AND of a bunch of clauses
- k-CNF formula
  - All clauses have exactly k variables

Satisfiability

CNF formula example:

$$(\boldsymbol{x_1} \vee \neg \boldsymbol{x_3} \vee \boldsymbol{x_4}) \wedge (\neg \boldsymbol{x_4} \vee \boldsymbol{x_3}) \wedge (\boldsymbol{x_2} \vee \neg \boldsymbol{x_1} \vee \boldsymbol{x_3})$$

Defn: If there is some assignment of 0's and 1's to the variables that makes it true then we say the formula is satisfiable

- $(\boldsymbol{x_1} \vee \neg \boldsymbol{x_3} \vee \boldsymbol{x_4}) \wedge (\neg \boldsymbol{x_4} \vee \boldsymbol{x_3}) \wedge (\boldsymbol{x_2} \vee \neg \boldsymbol{x_1} \vee \boldsymbol{x_3})$ is satisfiable: $\boldsymbol{x_1 = x_3 = 1}$
- $\boldsymbol{x_1} \wedge (\neg \boldsymbol{x_1} \vee \boldsymbol{x_2}) \wedge (\neg \boldsymbol{x_2} \vee \boldsymbol{x_3}) \wedge \neg \boldsymbol{x_3}$ is not satisfiable.

3SAT: Given a CNF formula $\boldsymbol{F}$ with exactly $\boldsymbol{3}$ variables per clause, is $\boldsymbol{F}$ satisfiable?

Common property of these problems

- There is a special piece of information, a short certificate or proof, that allows you to **efficiently verify** (in polynomial-time) that the **YES** answer is correct. This certificate might be very hard to find.

  - **3Color**: the assignment of a color to each node.
  - **Independent-Set, Clique**: the set of vertices
  - **Vertex-Cover**: the set of vertices
  - **Decision-TSP**: the path
  - **3SAT**: a truth assignment that makes the CNF formula true.

The complexity class **NP**

NP consists of all decision problems where

- You can **verify** the **YES** answers efficiently (in polynomial time) given a short (polynomial-size) ***certificate***

and

- **No fake certificate** can fool your polynomial time verifier into saying **YES** for a **NO** instance

More precise definition of $\mathbf{NP}$

A decision problem **A** is in $\mathbf{NP}$ iff there is

- a polynomial time procedure **VerifyA**(.,.) and
- a polynomial $\boldsymbol{p}$

s.t.

- for every input $\boldsymbol{x}$ that is a **YES** for **A** there is a string $\boldsymbol{t}$ with $|\boldsymbol{t}| \leq \boldsymbol{p}(|\boldsymbol{x}|)$ with **VerifyA**($\boldsymbol{x}$, $\boldsymbol{t}$) = **YES**

and

- for every input $\boldsymbol{x}$ that is a **NO** for **A** there does not exist a string $\boldsymbol{t}$ with $|\boldsymbol{t}| \leq \boldsymbol{p}(|\boldsymbol{x}|)$ with **VerifyA**($\boldsymbol{x}$, $\boldsymbol{t}$) = **YES**

- A string $\boldsymbol{t}$ on which **VerifyA**($\boldsymbol{x}$, $\boldsymbol{t}$) = **YES** is called a ***certificate*** for $\boldsymbol{x}$ or a ***proof*** that $\boldsymbol{x}$ is a **YES** input

Verifying the certificate is efficient

3Color: the coloring

- Check that each vertex has one of only 3 colors and check that the endpoints of every edge have different colors

Independent-Set, Clique: the set $\boldsymbol{U}$ of vertices

- Check that $|\boldsymbol{U}| \geq \boldsymbol{k}$ and either no (**IS**) or all (**Clique**) edges on present on $\boldsymbol{U}$

Vertex-Cover: the set $\boldsymbol{W}$ of vertices

- Check that $|\boldsymbol{W}| \leq \boldsymbol{k}$ and $\boldsymbol{W}$ touches every edge.

Decision-TSP: the path

- Check that the path touches each vertex and has total weight $\leq \boldsymbol{k}$.

- **3-SAT**: a truth assignment $\boldsymbol{\alpha}$ that makes the CNF formula $\boldsymbol{F}$ true.
  - Evaluate $\boldsymbol{F}$ on the truth assignment $\boldsymbol{\alpha}$.

Keys to showing that a problem is in **NP**

1. Must be decision probem (**YES**/**NO**)
2. For every given **YES** input, is there a certificate (i.e., a hint) that would help?
   - OK if some inputs don't need a certificate
3. For any given **NO** input, is there a fake certificate that would trick you?
4. You need a polynomial-time algorithm to be able to tell the difference.

Solving **NP** problems without hints

There is an obvious algorithm for all **NP** problems:

Brute force:

Try all possible certificates and check each one using the verifier to see if it works.

Even though the certificates are short, this is exponential time

- $\mathbf{2^n}$ truth assignments for $\boldsymbol{n}$ variables
- $\binom{\boldsymbol{n}}{\boldsymbol{k}}$ possible $\boldsymbol{k}$-element subsets of $\boldsymbol{n}$ vertices
- $\boldsymbol{n}!$ possible TSP tours of $\boldsymbol{n}$ vertices
- etc.

What We Know

- Every problem in $\mathbf{NP}$ is in exponential time
- Every problem in $\mathbf{P}$ is in $\mathbf{NP}$
  - You don't need a certificate for problems in $\mathbf{P}$ so just ignore any hint you are given
- Nobody knows if all problems in $\mathbf{NP}$ can be solved in polynomial time; i.e., does $\mathbf{P} = \mathbf{NP}$?
  - one of the most important open questions in all of science.
  - huge practical implications
- Most CS researchers believe that $\mathbf{P} \neq \mathbf{NP}$
  - $1M prize either way
  - but we don't have good ideas for how to prove this ...

NP-hardness & **NP**-completeness

Notion of hardness we **can** prove that is useful unless $\mathbf{P} = \mathbf{NP}$:

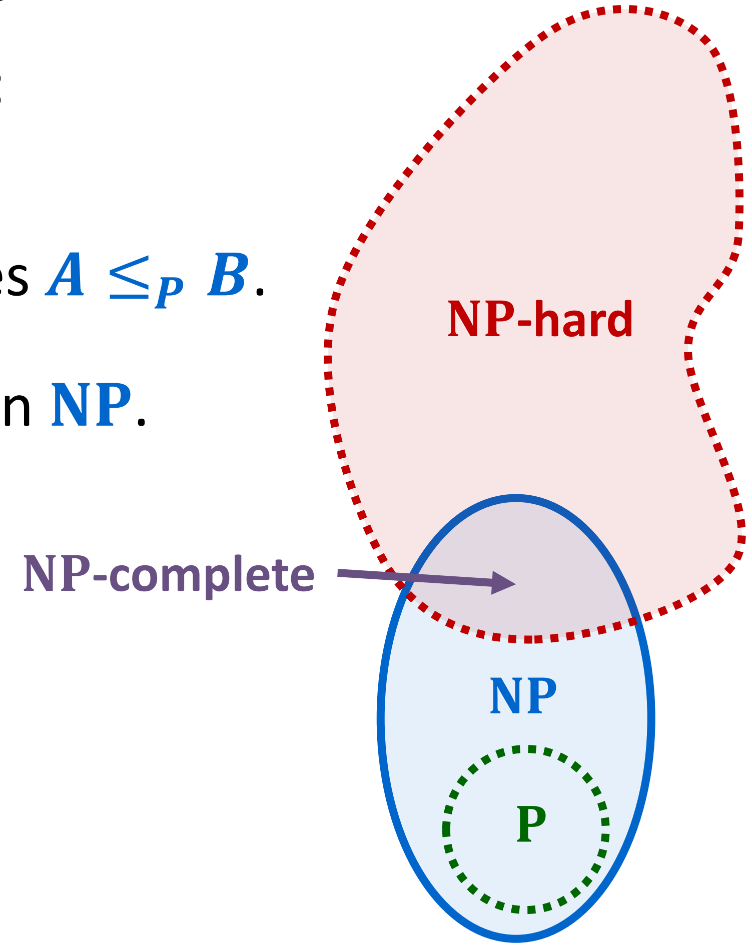
Defn: Problem $\boldsymbol{B}$ is **NP**-hard iff **every** problem $\boldsymbol{A} \in \mathbf{NP}$ satisfies $\boldsymbol{A} \leq_{\boldsymbol{P}} \boldsymbol{B}$.

This means that $\boldsymbol{B}$ is at least as hard as every problem in **NP**.

Defn: Problem $\boldsymbol{B}$ is **NP**-complete iff

- $\boldsymbol{B} \in \mathbf{NP}$ and
- $\boldsymbol{B}$ is **NP**-hard.

This means that $\boldsymbol{B}$ is a hardest problem in **NP**.



Not at all obvious that any **NP**-complete problems exist!

Cook-Levin Theorem

Theorem [Cook 1971, Levin 1973]: **3SAT** is **NP**-complete

Proof: See CSE 431.

Corollary: If **3SAT** $\leq_P$ **B** then **B** is **NP**-hard.

Proof: Let **A** be an arbitrary problem in **NP**.
Since **3SAT** is **NP**-hard we have **A** $\leq_P$ **3SAT.**

Then **A** $\leq_P$ **3SAT** and **3SAT** $\leq_P$ **B** imply that **A** $\leq_P$ **B**.

Therefore every problem **A** in **NP** has **A** $\leq_P$ **B**
so **B** is **NP**-hard.

Cook & Levin did the hard work.

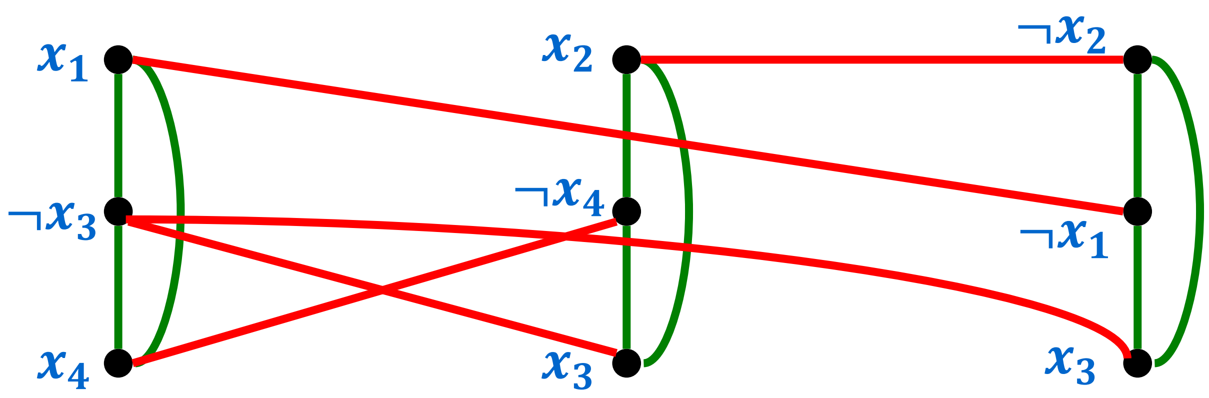
We only need to give one reduction to show that a problem is NP-hard!

Another $\mathbf{NP}$-complete problem: 3SAT $\leq_P$ Independent-Set

1. The reduction:
   - Map CNF formula F to a graph G and integer k
   - Let m = # of clauses of F
   - Create a vertex in G for each literal occurrence in F
   - Join two vertices u, v in G by an edge iff
     - u and v correspond to literals in the same clause of F (green edges) or
     - u and v correspond to literals x and $\neg x$ (or vice versa) for some variable x (red edges).
   - Set $k = m$
2. Clearly polynomial-time computable

Another $\mathbf{NP}$-complete problem: 3SAT $\leq_P$ Independent-Set

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



$m = 3$

G has both kinds of edges.

The color is just to show why the edges were included.

$k = m$

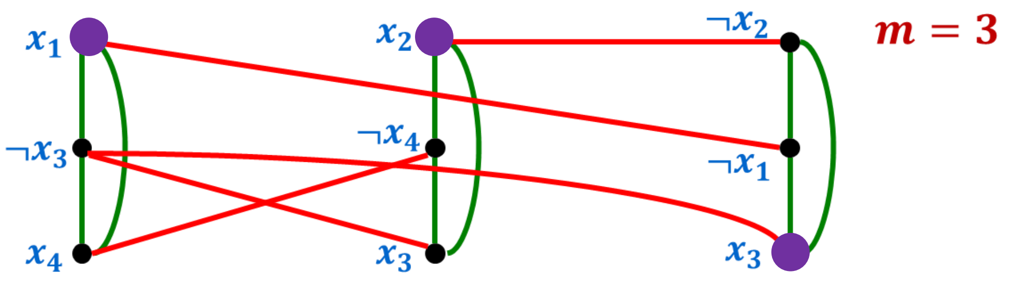
Correctness (⇒)

Suppose that F is satisfiable (**YES** for **3SAT**)

- Let α be a satisfying assignment; it satisfies at least one literal in each clause.
- Choose the set U in G to correspond to the **first satisfied literal in each clause**.
  - $|U| = m$
  - Since U has $\mathbf{1}$ vertex per clause, no green edges inside U.
  - A truth assignment never satisfies both x and $\neg x$, so no red edges inside U.
  - Therefore U is an independent set of size m

Therefore (G, m) is a **YES** for **Independent-Set**.

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



Satisfying assignment α:

$$\alpha(x_1) = \alpha(x_2) = \alpha(x_3) = \alpha(x_4) = 1$$

Set U marked in purple is independent.

Correctness (⇐)

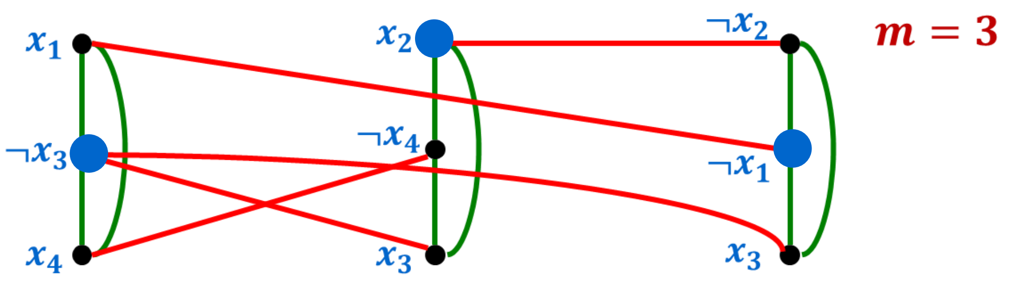
Suppose that G has an independent set of size m

((G, m) is a **YES** for **Independent-Set**)

- Let U be the independent set of size m;
- U must have one vertex per column (green edges)
- Because of red edges, U doesn't have vertex labels with conflicting literals.
- Set all literals labelling vertices in U to true
- This may not be a total assignment but just extend arbitrarily to a total assignment α.
  - This assignment satisfies F since it makes at least one literal per clause true.

Therefore F is satisfiable and a **YES** for **3SAT**.

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



Given independent set U of size m

Satisfying assignment α: Part defined by U:

$\alpha(x_1) = 0, \alpha(x_2) = 1, \alpha(x_3) = 0$

Set $\alpha(x_4) = 0$.

Many **NP**-complete problems

Since **3SAT** $\leq_P$ **Independent-Set**, **Independent-Set** is **NP**-hard.

We already showed that **Independent-Set** is in **NP**.

⇒ **Independent-Set** is **NP**-complete

Corollary: **Clique** and **Vertex-Cover** are also **NP**-complete.

Proof: We already showed that all are in **NP**.

We also showed that **Independent-Set** polytime reduces to all of them.

Combining this with **3SAT** $\leq_P$ **Independent-Set** we get that all are **NP**-hard.