CSE 421 Winter 2025
Lecture 22:
Reductions

Nathan Brunelle

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

Q: Does every problem have a polynomial time algorithm?

A: NO. The Halting problem is undecidable so it doesn't have an algorithm at all [Turing]

Q: If there is an algorithm for a problem is there is always one that runs in polynomial time?

A: NO. There are problems that require exponential time to solve. (See CSE 431)

Q: What about some of the problems we've seen so far?

How do we know that a problem is hard?

At this point in the quarter, you've probably at least once been banging your head against a problem…

… for so long that you began to think "there's no way there's actually an efficient algorithm for this problem."

That wasn't true for any of the problems we have assigned you to solve (so far).

- But we think that it **is** true for certain types of problems, including one where you showed how some algorithms failed to work.

- Over the next week we will look at how you can figure out that some problem you encounter is just as hard as those.

Some definitions

Defn: A problem is a set of inputs and their associated correct outputs.

- “Find a Minimum Spanning Tree” is a problem.
- Input is a graph, output is the MST.

- “Tell whether a graph is bipartite” is a problem.
- Input is a graph, output is “yes” or “no”

- “Find the ‘maximum subarray sum’” is a problem.
- Input is an array, output is the number that represents the largest sum of a subarray.

Some definitions

Defn: An instance is a single input to a problem.

- A single, particular graph is an instance of the MST problem
- A single, particular graph is an instance of the bipartiteness-checking problem.
- A single, particular array is an instance of the maximum subarray sum problem.

Relative Hardness of Problems

- Want to ***compare*** the hardness of problems
  - Want to be able to say

“Problem **B** is solvable in polynomial time ⇒ problem **A** is solvable in polynomial time”

“Problem **B** is at least as hard as problem **A**”

Polynomial Time Reduction

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

- uses only a polynomial number of steps, and
- makes only a polynomial number of calls to a method for B.

Theorem: If $A \leq_P B$ then a poly time algorithm for B ⇒ poly time algorithm for 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 A, then that proves there is no fast algorithm for B.

Intuition for "$A \leq_P B$": "B is at least as hard* as A" *up to polynomial-time slop.

Now the weird part…

We read “$A \leq_P B$” as “A is polynomial-time **reducible** to B” or

“A can be **reduced** to B in polynomial time”

- It means “we can solve A using at most a polynomial amount of work on top of solving B.”
- But word reducible seems to go in the opposite direction of the $\leq$ sign.

The general motivation for the terminology is:

- “To solve A we can reduce our attention from all possible things just to solving B.”

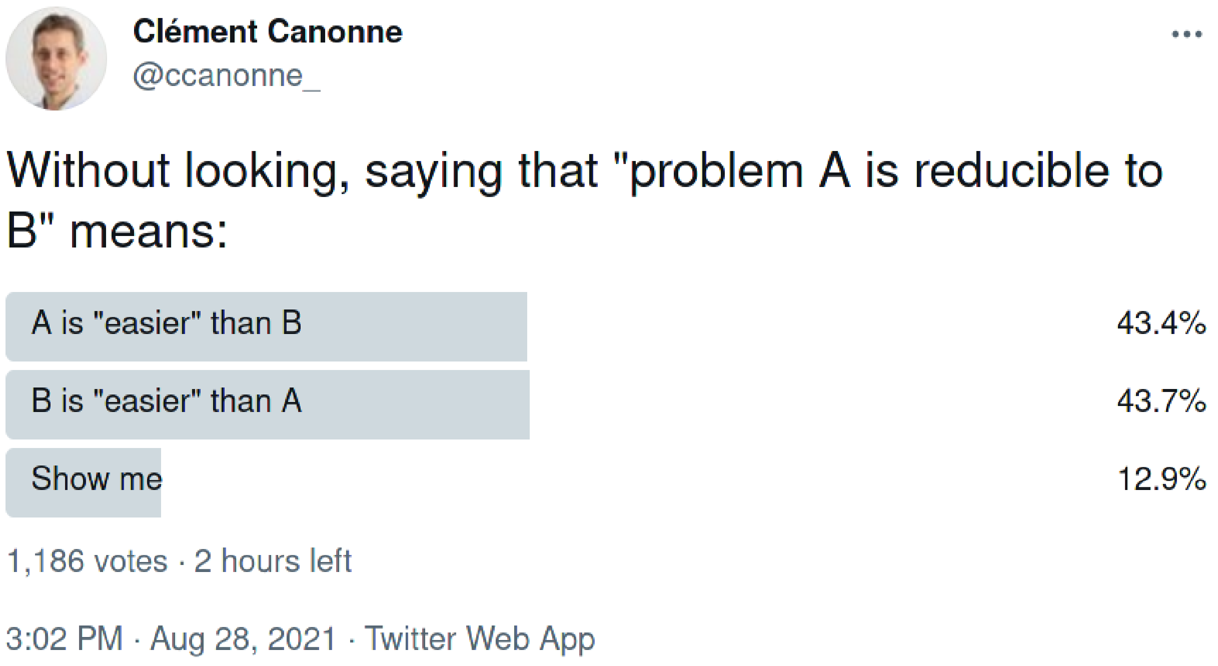
- Often we have easy problem $\leq_P$ harder problem. (e.g. bipartite matching $\leq_P$ flow)
- Sometimes we can show general case $\leq_P$ special case (e.g. stable matching)
  - In this case we really use the extra polytime work we’re allowed.

Some Previous Examples

- On Homework 1, you reduced “stable matchings with different numbers of applicants and jobs with only some unacceptable“ to “[standard] stable matching”.
- On Homework 2, you (might have) reduced
  “labelling bear photographs” to “2-coloring”.
- We reduced “Bipartite Matching” to “Network Flow”.

Getting the wording right

Lots of people mess this up!



Clément Canonne
@ccanonne_

Without looking, saying that "problem A is reducible to B" means:

A is "easier" than B	43.4%
B is "easier" than A	43.7%
Show me	12.9%

1,186 votes · 2 hours left

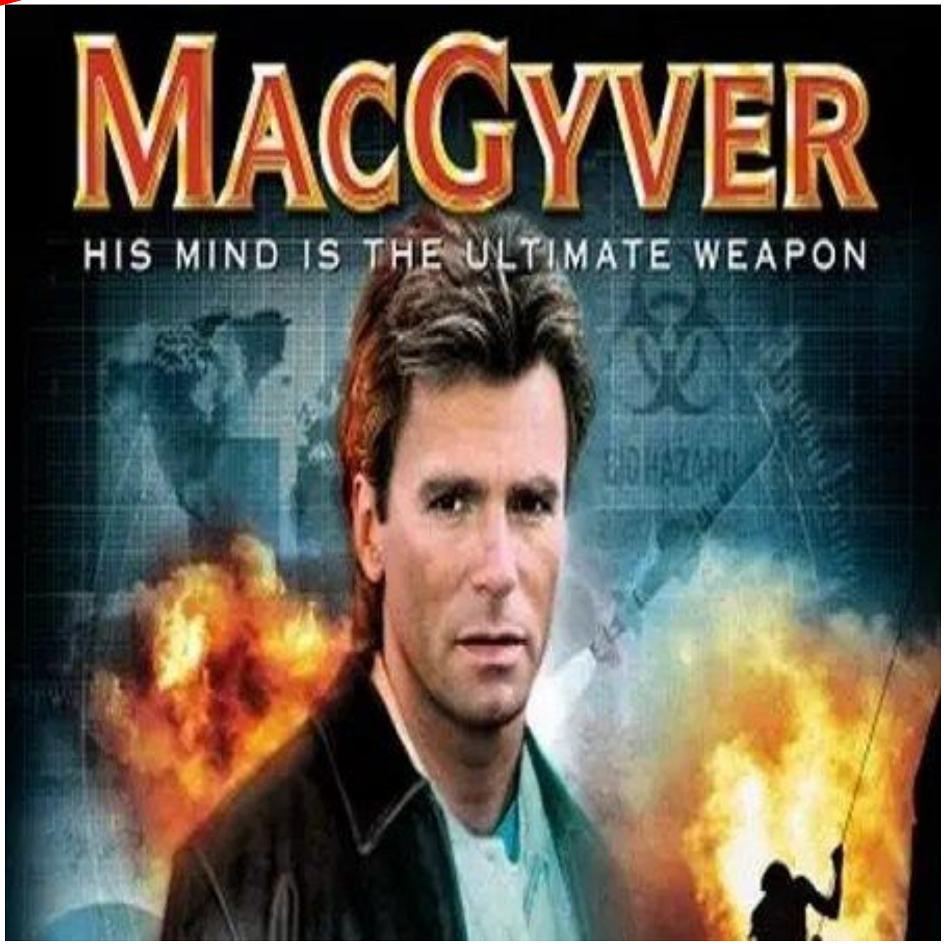
3:02 PM · Aug 28, 2021 · Twitter Web App

Tl;dr check the direction you're going every time. It's going to take a while to be intuitive.

Reductions

Shows how two different problems relate to each other

MOVIE TIME!



MacGyver's Reduction

Problem we don't know how to solve

A

Opening a door

Aim duct at door, insert keg

Problem we do know how to solve

B

Lighting a fire

How?

Solution for $\boldsymbol{B}$

Alcohol, wood, matches

Put fire under the Keg

Reduction

Solution for $\boldsymbol{A}$

Keg cannon battering ram

Using the word “reduction”

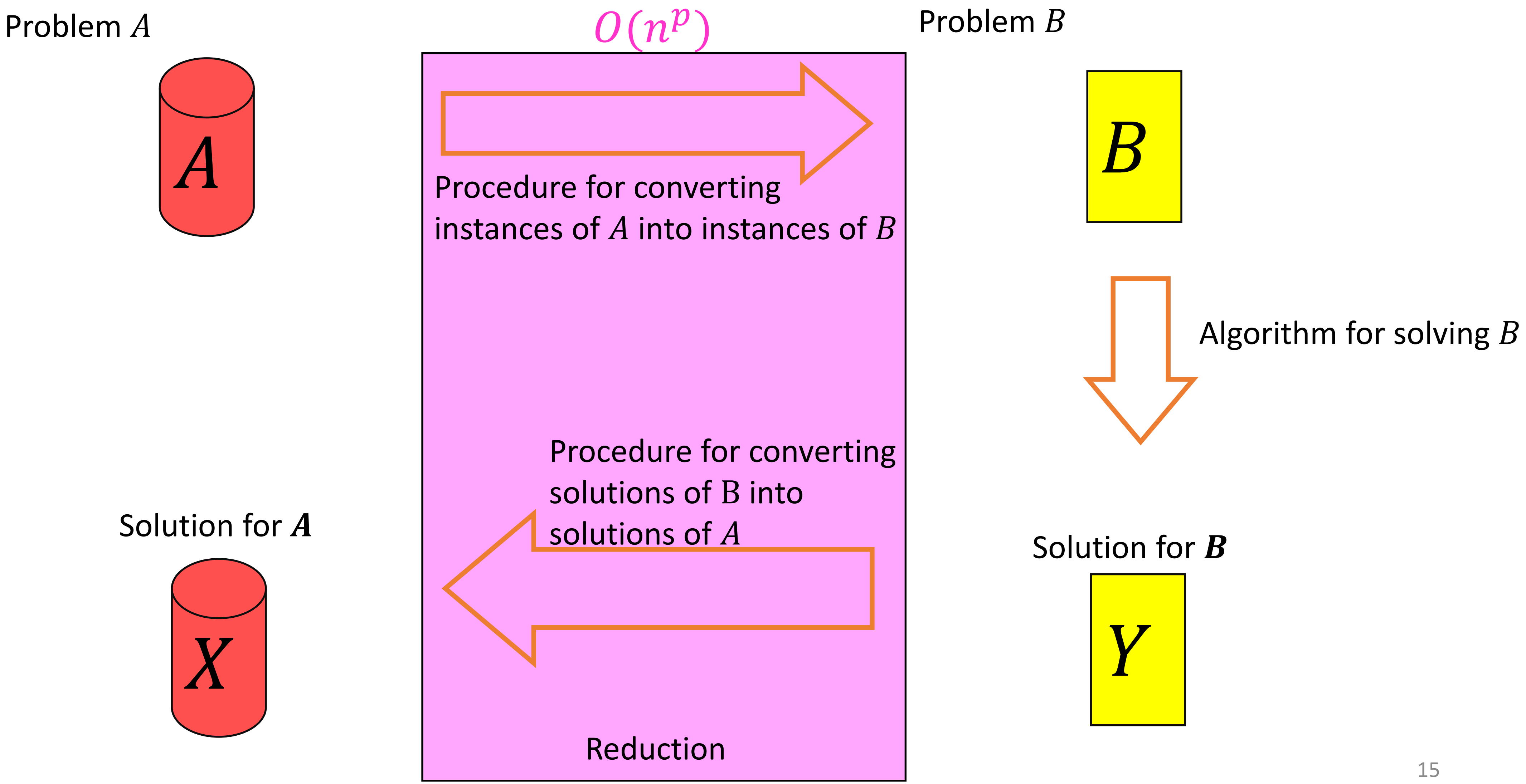
- Problem A: open a door
- Problem B: light a fire
- MacGyver reduced A to B
  - Meaning he used a solution to B to produce a solution for A
- Which statements are correct?
  1. $A \leq B$
  2. $B \leq A$
  3. A is “easier” than B
  4. B is “easier” than A

Using the word "reduction"

- Problem A: open a door
- Problem B: light a fire
- MacGyver reduced A to B
  - Meaning he used a solution to B to produce a solution for A
- Which statements are correct?

1. $\boldsymbol{A \leq B}$
2. ~~$B \leq A$~~
3. **$\boldsymbol{A}$ is "easier" than $\boldsymbol{B}$**
4. ~~B is "easier" than A~~

Polynomial Time Reductions



Decision Problems

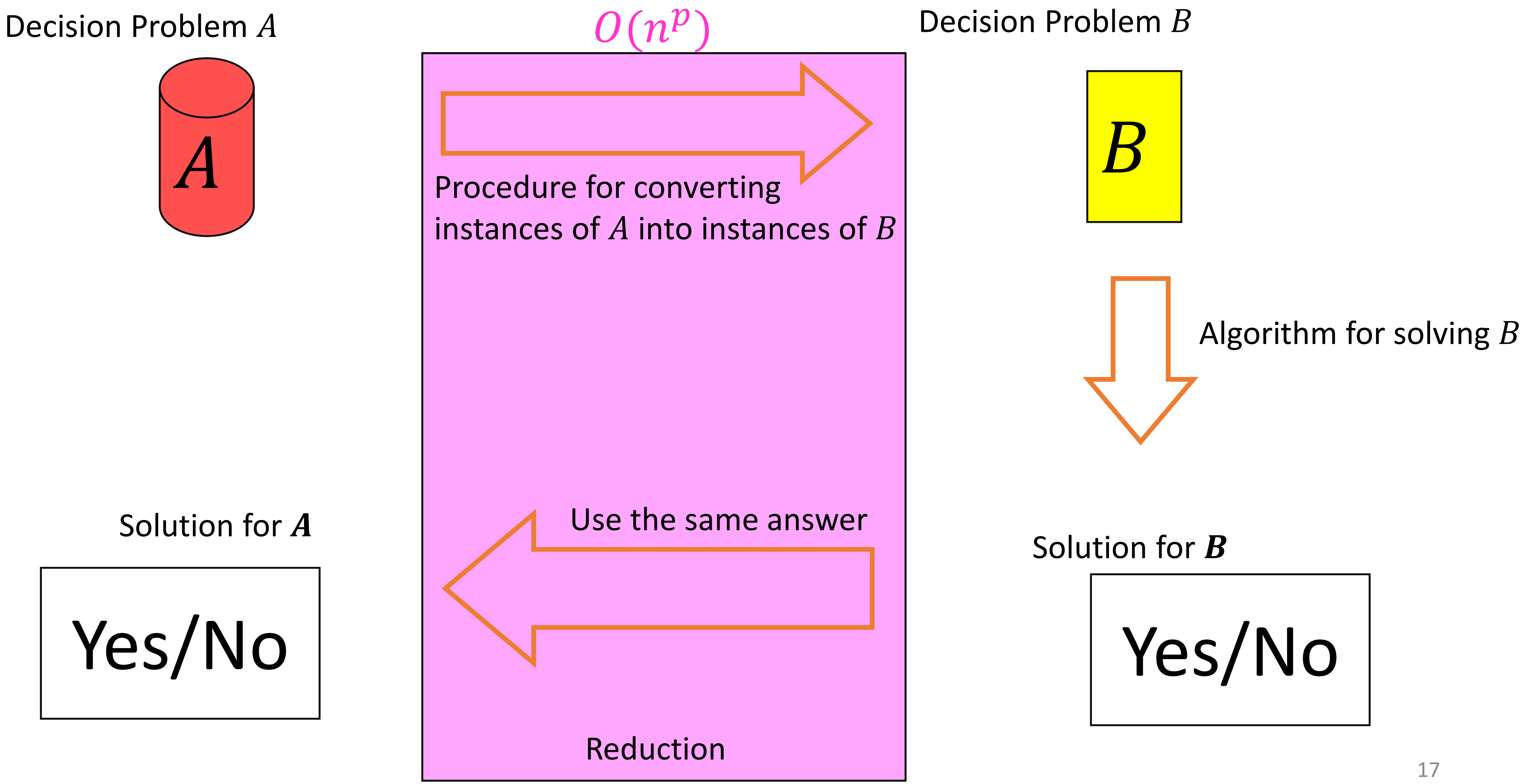
Defn: A decision problem is a problem that has a "**YES**" or "**NO**" answer.

- A correct algorithm has a Boolean return type

Example: Is this polytope empty?

- Problems can be rephrased in terms of very similar decision problems.
  - Instead of "Find the shortest path from s to t"
    ask "Is there a path from s to t of length at most k?"
  - Can do binary search to find exact value.
  - If a problem is easy then all of its individual output bits must be easy
  - If a problem is hard then at least one of its output bits must be hard.

Polynomial Time Reductions (Decision Problems)



A Special Kind of Polynomial-Time Reduction

We will often use a restricted form of $\boldsymbol{A \leq_P B}$ often called a Karp or many-one reduction...

Defn: $\boldsymbol{A \leq_P^1 B}$ iff there is an algorithm for $\boldsymbol{A}$ given a black box solving $\boldsymbol{B}$ that on input $\boldsymbol{x}$ that

- Runs for polynomial time computing $\boldsymbol{y = f(x)}$
- Makes $\boldsymbol{1}$ call to the black box for $\boldsymbol{B}$ on input $\boldsymbol{y}$
- Returns the answer that the black box gave

We say that the function $\boldsymbol{f}$ is the reduction.

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

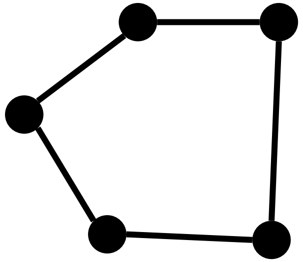
4 steps for reducing (decision problem) A to problem B

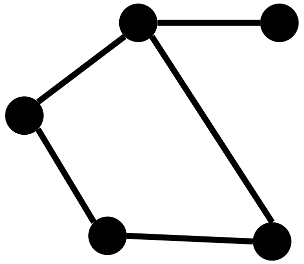
1. Describe the reduction itself
   - i.e., the function that converts the input for A to the one for problem B.
   - i.e., describe what the top arrow in the pink box does
2. Make sure the running time would be polynomial
   - In lecture, we’ll sometimes skip writing out this step.
3. Argue that if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

Contrapositive

Reduce 2Color to 3Color

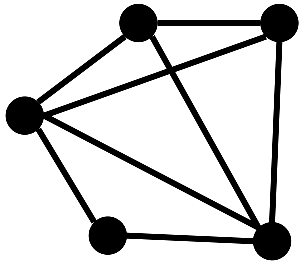
Defn: A undirected graph $\boldsymbol{G = (V, E)}$ is $\boldsymbol{k}$-colorable iff
we can assign one of $\boldsymbol{k}$ colors to each vertex of $\boldsymbol{V}$ s.t. for
$\boldsymbol{(u, v) \in E}$, their colors, $\boldsymbol{\chi(u)}$ and $\boldsymbol{\chi(v)}$, are different.
"edges are not monochromatic"

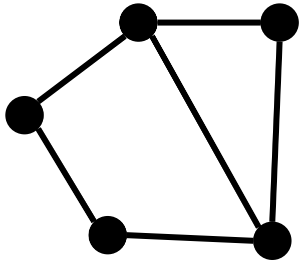




2Color: Given: an undirected graph $\boldsymbol{G}$
Is $\boldsymbol{G}$ 2-colorable?

3Color: Given: an undirected graph $\boldsymbol{G}$
Is $\boldsymbol{G}$ 3-colorable?

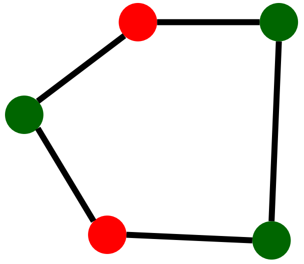




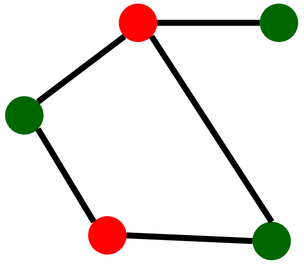
Reduce 2Color to 3Color

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?

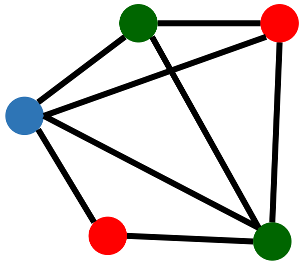


No

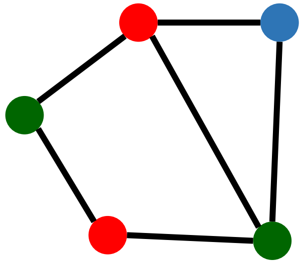


Yes

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.

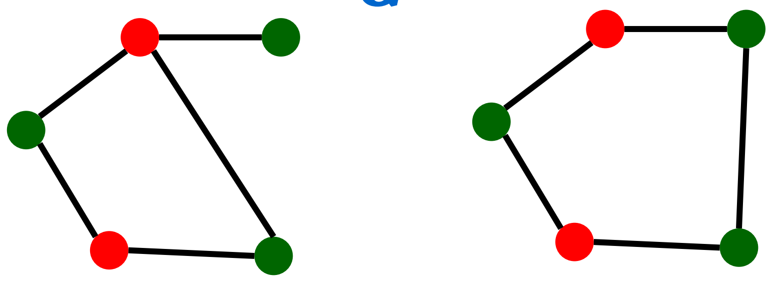
Usual outline:

- 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 G, if 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 G

Reduction f: Add one extra vertex v and attach it to **everything** in G.

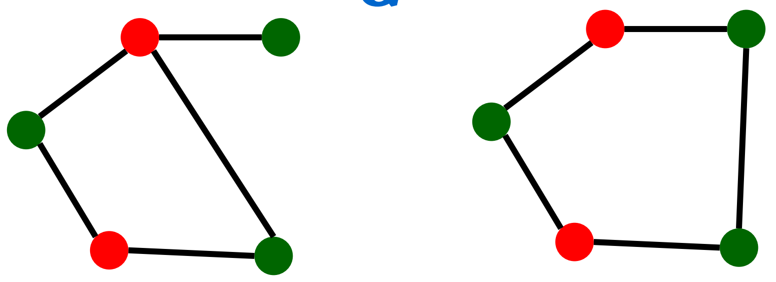
Write $H = f(G)$.

(f is polynomial time computable.)

Reducing 2Color to 3Color

2Color

G

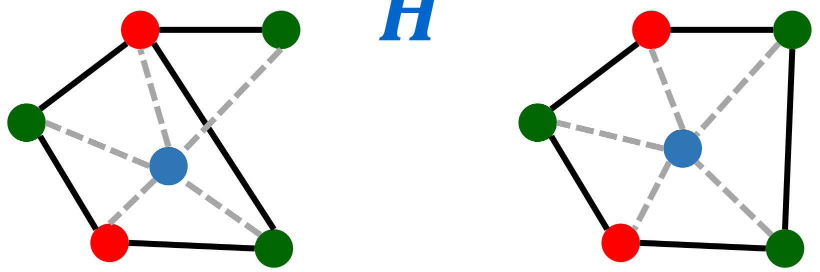


$O(n^p)$

Add a new node to ***G***, connect every node to it

3Color

H



Algorithm for solving 3Color

Solution for 2Color

Yes/No

Use the same answer

Reduction

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:

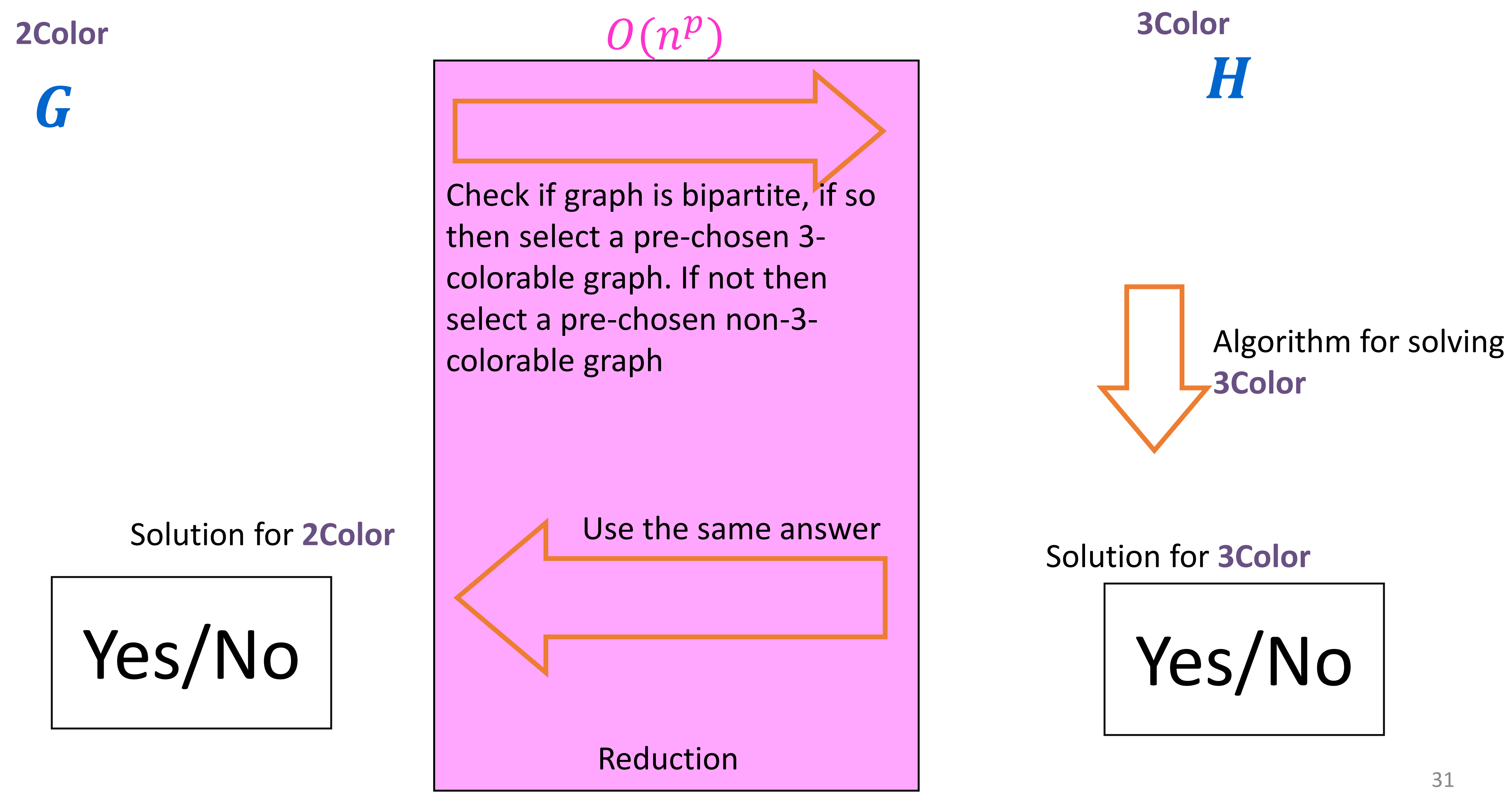
- Always do two separate proofs! (Don't try to prove both directions at once, don't refer back to the prior proof and say "for the same reason". There are usually subtle differences.)
- Don't use contradiction! (It's easy to start from the wrong spot and accidentally prove the same direction twice without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

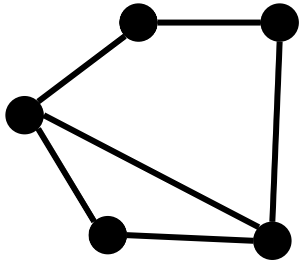
Given a graph $G = (V, E)$ and an integer k

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.)

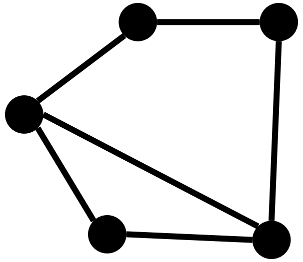
Claim: **Independent-Set** $\leq_P$ **Clique**

Examples

- Independent Set

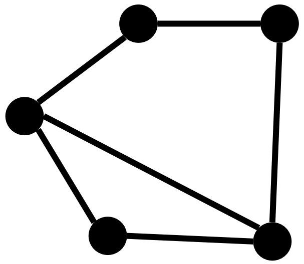


$k = 2$

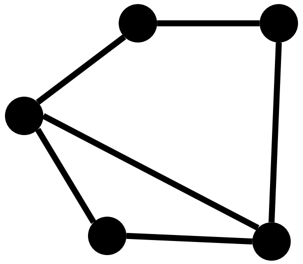


$k = 3$

- Clique



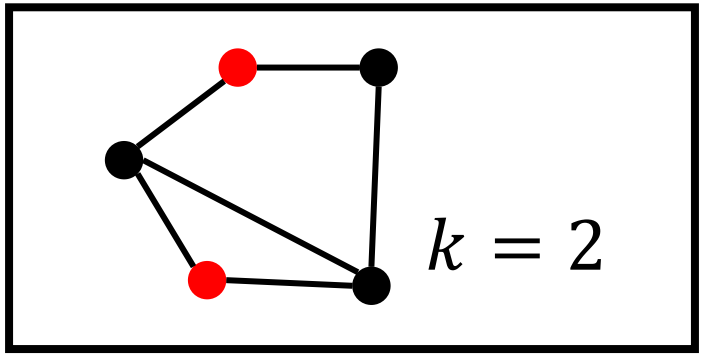
$k = 3$



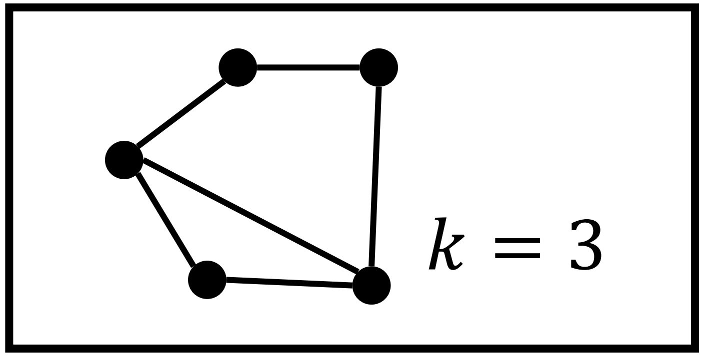
$k = 4$

Examples

- Independent Set

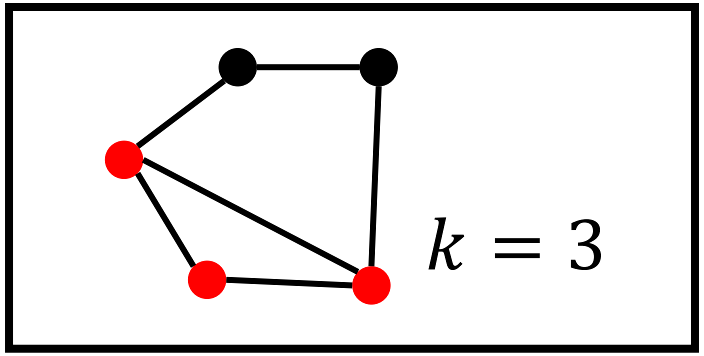


Yes

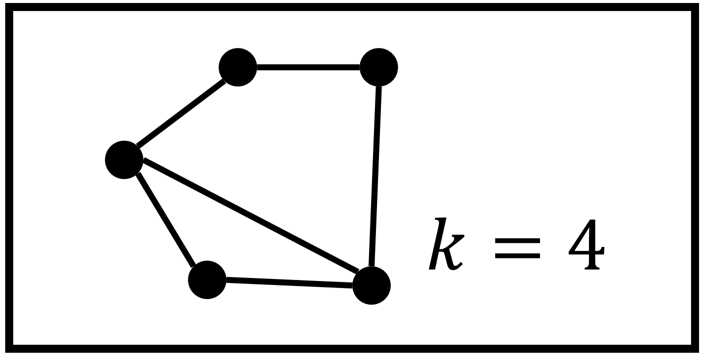


No

- Clique



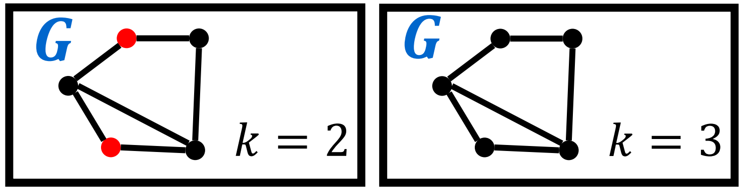
Yes



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 **Independent Set**

Yes/No

Solution for **Clique**

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.

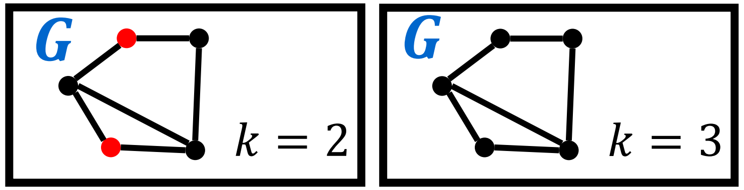
graph complement

From the definitions, U is an independent set in G

$\Leftrightarrow$ U is a clique in G'

Reducing Independent Set to Clique

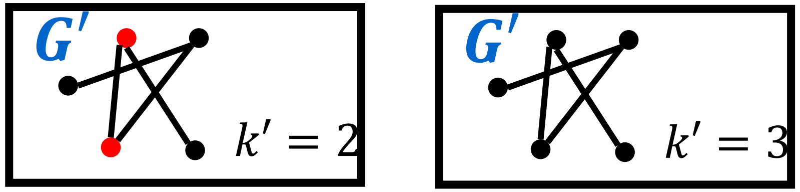
Independent Set



$O(n^p)$

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

Clique



Algorithm for solving **Clique**

Solution for **Independent Set**

Yes/No

Use the same answer

Reduction

Solution for **Clique**

Yes/No

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

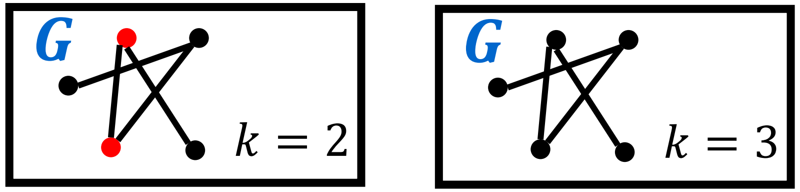
- $G' = (V, E')$ has the same vertices as G but E' consists of **precisely** those edges on V that are not edges of G.

From the definitions, U is an clique in G

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

Reducing Clique to Independent Set

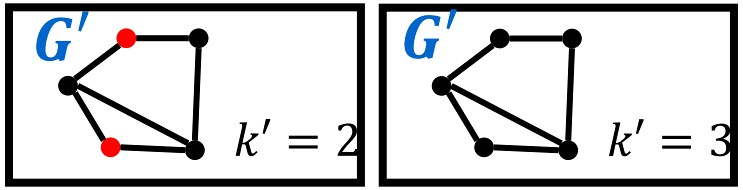
Clique



$O(n^p)$

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

Independent Set



Algorithm for solving **Independent Set**

Use the same answer

Solution for **Clique**

Yes/No

Solution for **Independent Set**

Yes/No

Reduction

Another Reduction

Vertex-Cover:

Given a graph $\boldsymbol{G = (V, E)}$ and an integer $\boldsymbol{k}$

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

($\boldsymbol{W}$ is a vertex cover, a set of vertices that covers $\boldsymbol{E}$.)

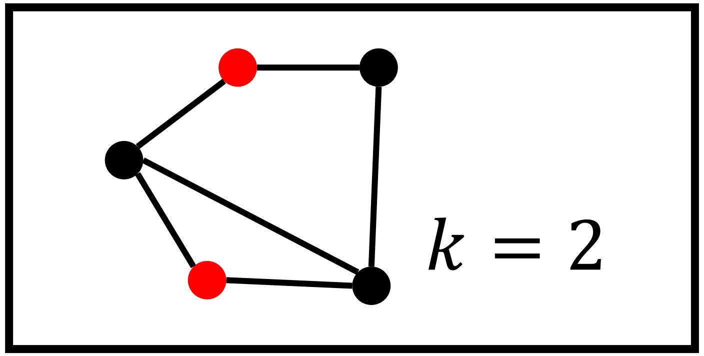
Claim: **Independent-Set** $\leq_P$ **Vertex-Cover**

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

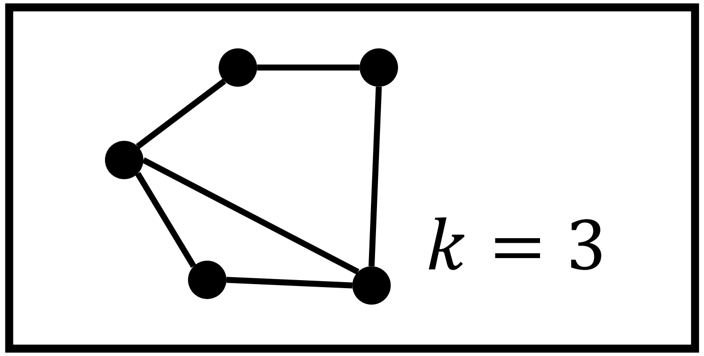
$\boldsymbol{U}$ is an independent set $\Leftrightarrow$ $\boldsymbol{V - U}$ is a vertex cover

Examples

- Independent Set

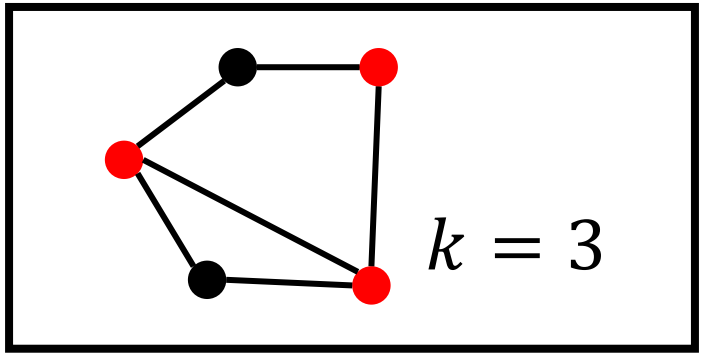


Yes

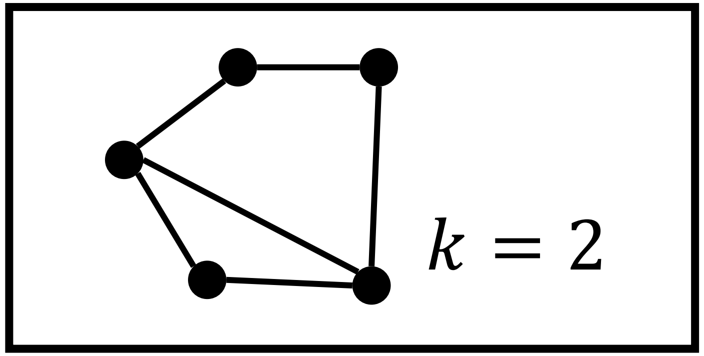


No

- Vertex Cover



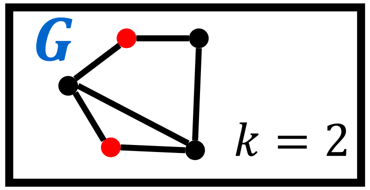
Yes

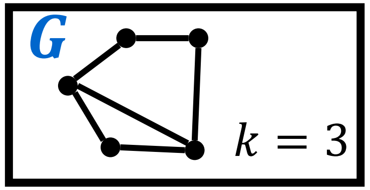


No

Reducing Independent Set to Vertex Cover

Independent Set

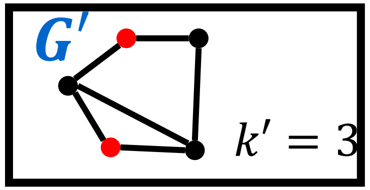


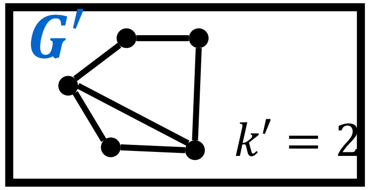


$O(n^p)$

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

Vertex Cover





Algorithm for solving **Vertex Cover**

Solution for **Vertex Cover**

Yes/No

Use the same answer

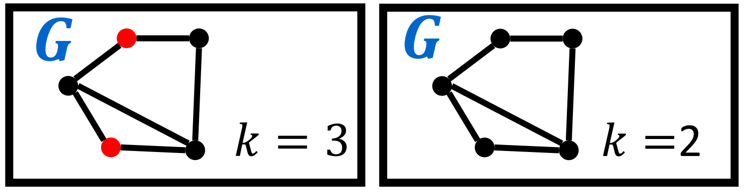
Solution for **Independent Set**

Yes/No

Reduction

Reducing Vertex Cover to Independent Set

Vertex Cover



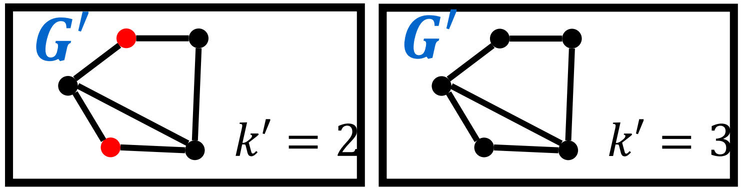
$O(n^p)$

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

Use the same answer

Reduction

Independent Set



Algorithm for solving **Independent Set**

Solution for **Independent Set**

Yes/No

Solution for **Vertex Cover**

Yes/No

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

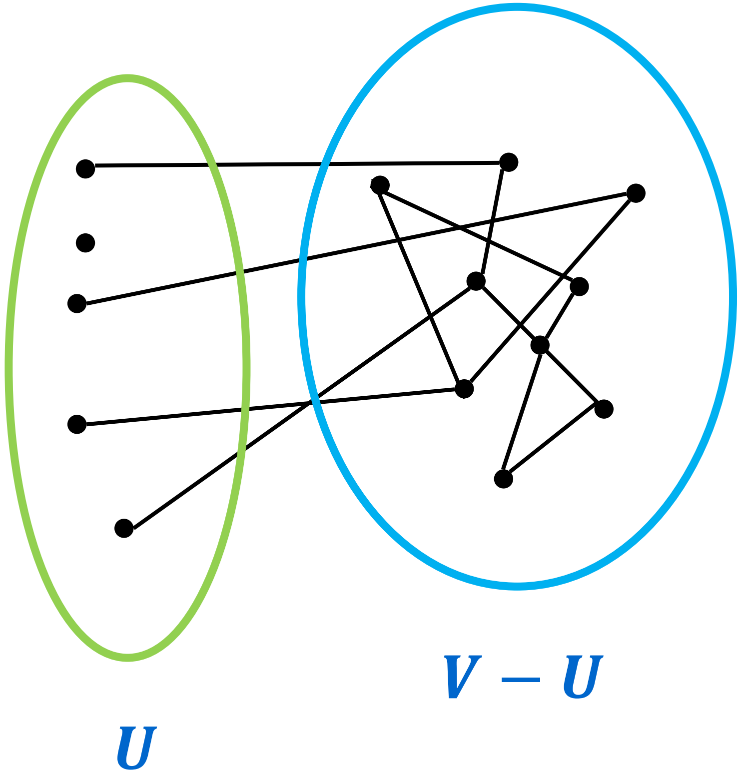
So, at least one endpoint of e must be in $V - U$

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?

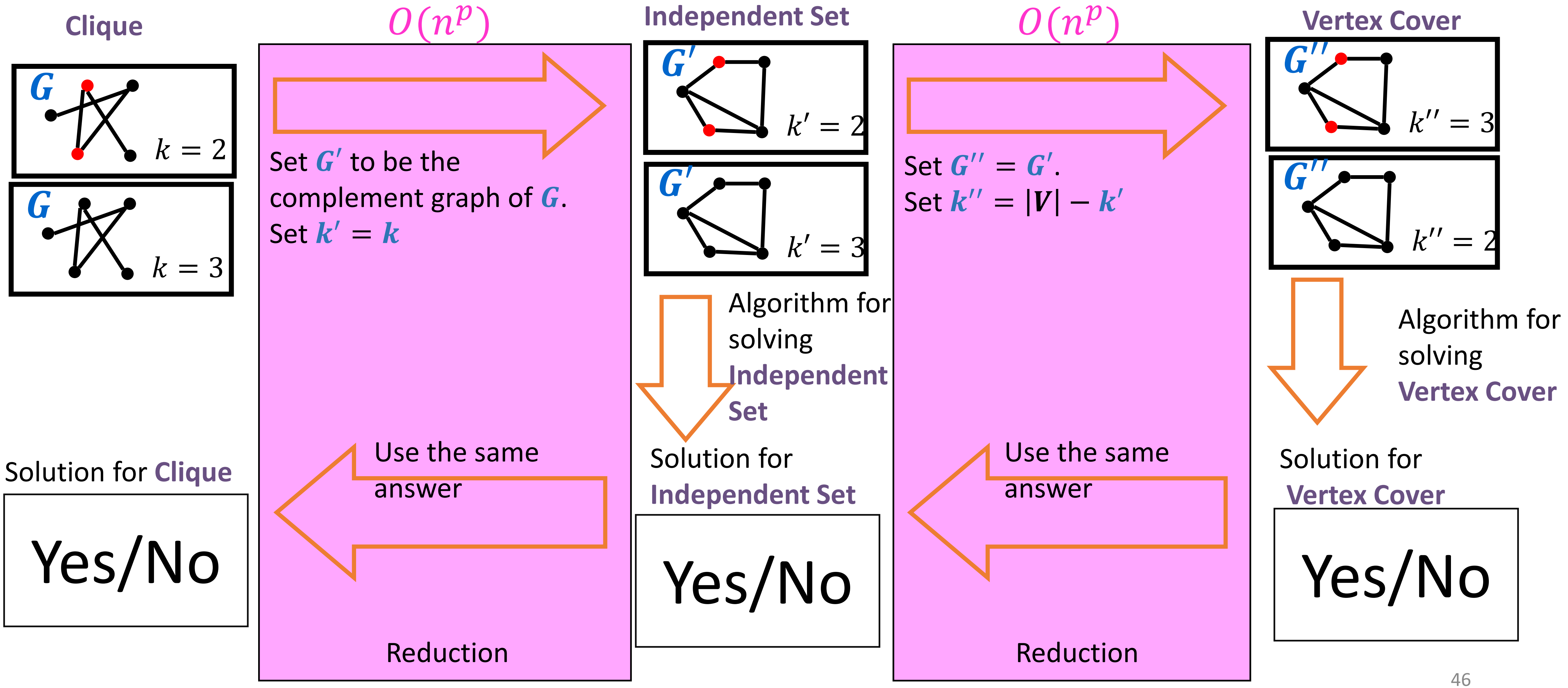
(W is a vertex cover, a set of vertices that covers E.)

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

Idea:

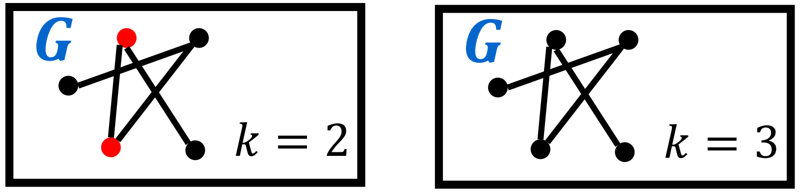
Use **Clique $\leq_P$ Independent-Set** and **Independent-Set $\leq_P$ Vertex-Cover**

Reducing Clique to Vertex Cover



Reducing Clique to Vertex Cover

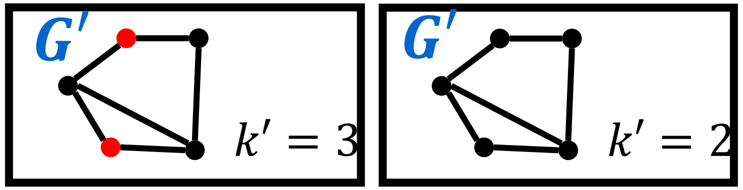
Clique



$O(n^p)$

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

Vertex Cover



Algorithm for solving **Vertex Cover**

Solution for **Vertex Cover**

Yes/No

Use the same answer

Reduction

Solution for **Clique**

Yes/No