NP-Completeness

CS/MCS 401: Computer Algorithms I
Lecture 23
Professor: Jonathan Liu

Icebreaker: Does The Bean still need to be moved?

Announcements

- HW5 is due tomorrow!
- Project checkpoint is due tomorrow!
- Final Exam is scheduled:
  - Wednesday, 12/9 at 10:30am, here!

Today's Overview

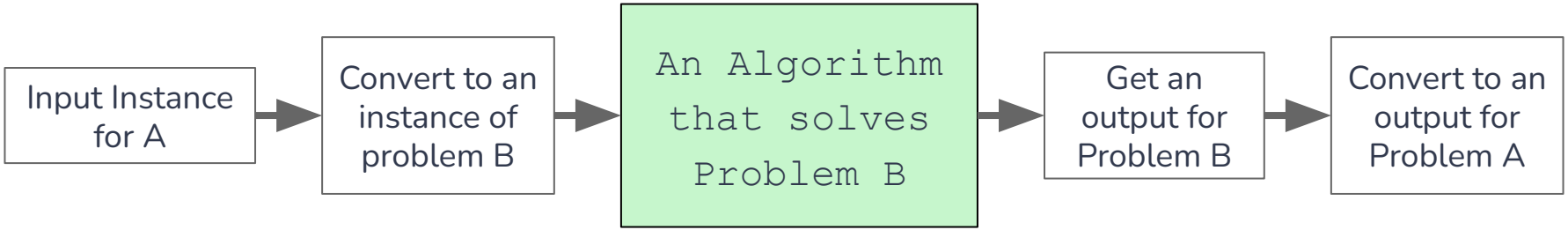
Today, we're going to start talking about problems that we *believe* to be too hard. We'll talk about what that means formally, and then we'll see some examples.

Recap

Poly-Time Reductions

A reduction is a solution to problem A that uses an algorithm for solving problem B.

In this class, we only care about reductions that can occur in poly-time.

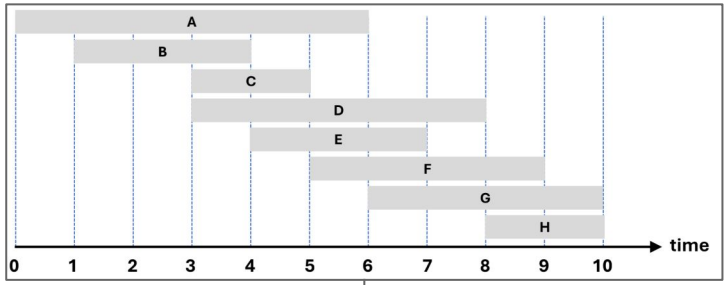


If you can solve problem A using an algorithm that solves problem B, we say **A reduces to B.** This also means that **A is easier (or at least no harder) than B,** or **A ≤ B in difficulty.**

Reductions

A reduction is a solution to problem A that uses an algorithm for solving problem B.

Given an instance of a course scheduling problem, how do I use the algorithm I have to solve it?



Here, we reduced Course Scheduling to Weighted Course Scheduling. We said "Course Scheduling is easy if we can solve Weighted Course Scheduling." **Therefore, Course Scheduling is easier than Weighted Course Scheduling.**

An Algorithm that solves Course Scheduling!

Give all the classes weight 1

Classes with Weights

An Algorithm that solves *Weighted* Course Scheduling

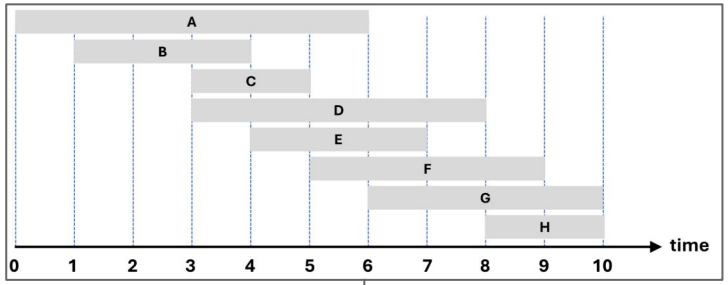
The highest-weight set of classes

Output that set!

Reductions

A reduction is a solution to problem A that uses an algorithm for solving problem B.

Given an instance of a course scheduling problem, how do I use the algorithm I have to solve it?



Here, we reduced Course Scheduling to Weighted Course Scheduling. Therefore, Course Scheduling is easier than Weighted Course Scheduling.

Output that set!

An Algorithm that solves Course Scheduling!

Give all the classes weight 1

Classes with Weights

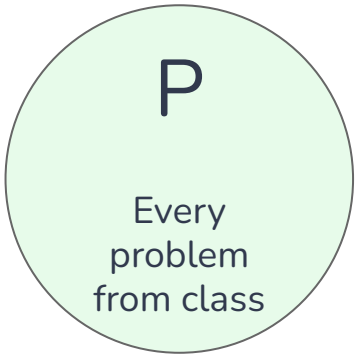
An Algorithm that solves *Weighted* Course Scheduling

The highest-weight set of classes

P (Complexity Class)

P (**P**olynomial) is the set of all problems that can be **solved** in polynomial time. Its runtime is $O(n^p)$ for some constant p.

Every problem we've seen in this class so far (except Knapsack!) is in the set P.

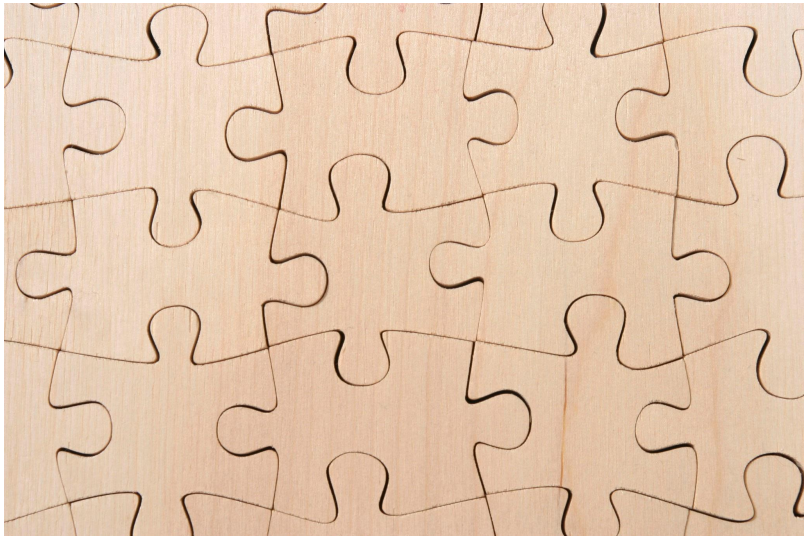


Solving vs. Verifying

Solving



Verifying



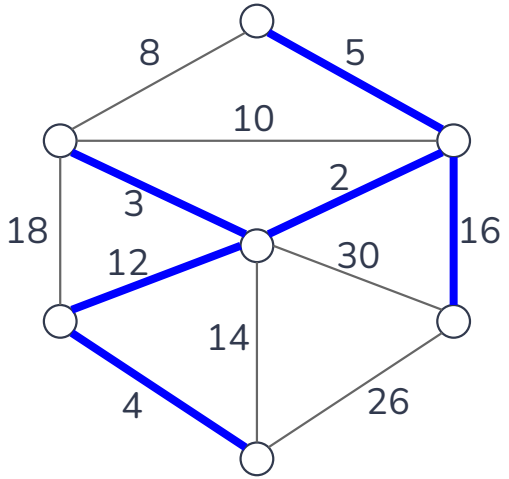
NP (Complexity Class)

NP (**N**ondeterministic **P**olynomial) is the set of all decision problems that can be **verified** in polynomial time.

If the answer to a decision problem is “yes,” there is a **certificate** that can be checked in polynomial time.

Here, if I show you a tree of weight 60, it’s really easy for you to **verify** that that’s true.

“Is there a spanning tree with weight 60 or less?”

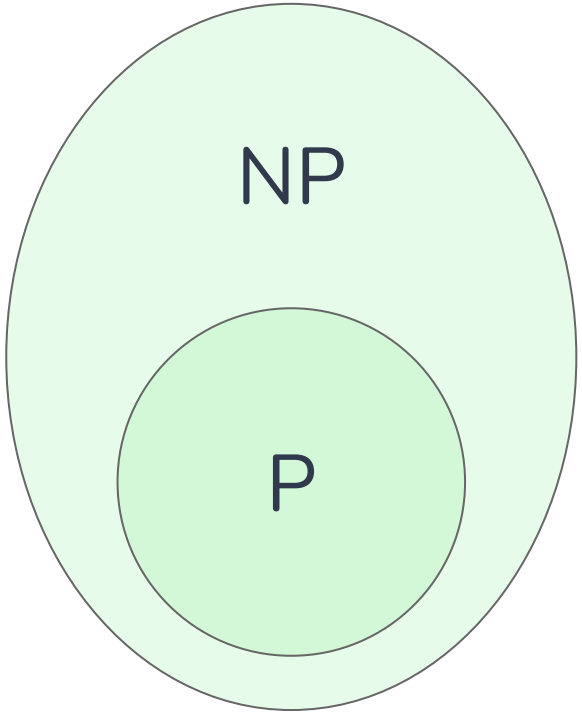


P vs. NP

The Million-Dollar Question: Does P = NP?

In other words, if a problem can be verified quickly, does that mean it can be solved quickly?

We don’t really know. But intuitively, it feels like discovering something is harder than checking it!



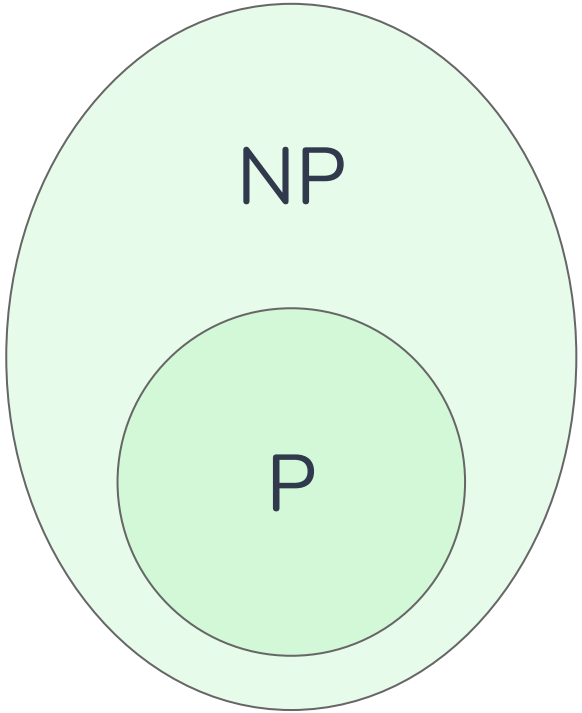
NP-Completeness

P vs. NP

In order to prove that P=NP, we'd have to prove that *every* problem in NP can be solved in polynomial time.

Alternatively, to prove that P ≠ NP, we'd have to find some problem that can be verified in polynomial time and prove that *no* polynomial-time algorithm can solve it.

Both of these tasks seem quite difficult...



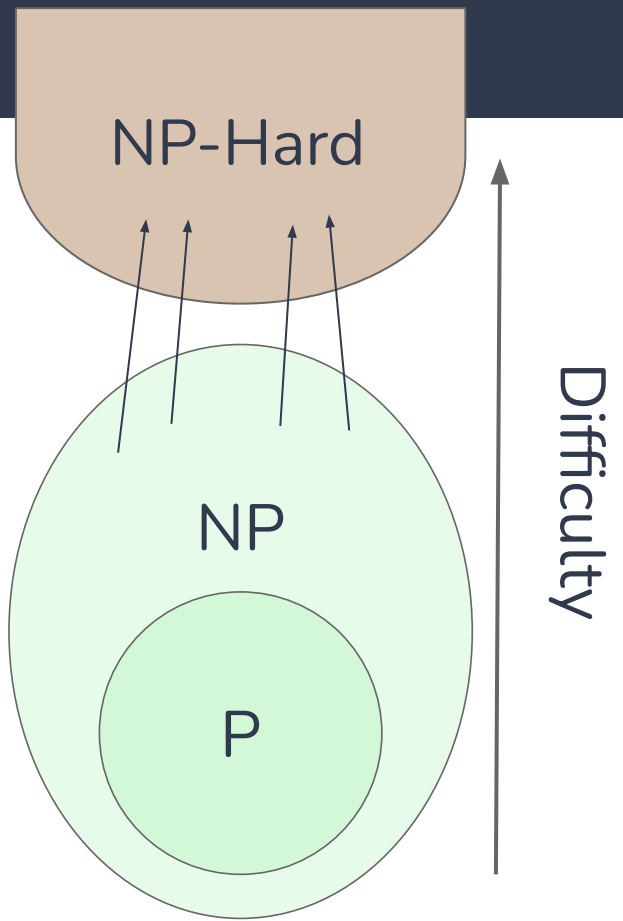
NP-Hard Problems

NP-Hard is the set of all problems where, *if the problem can be solved in polynomial time, then P=NP.*

In other words, **if the problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time.**

Do NP-Hard problems even exist...?

Remarkably, yes!



NP-Complete Problems

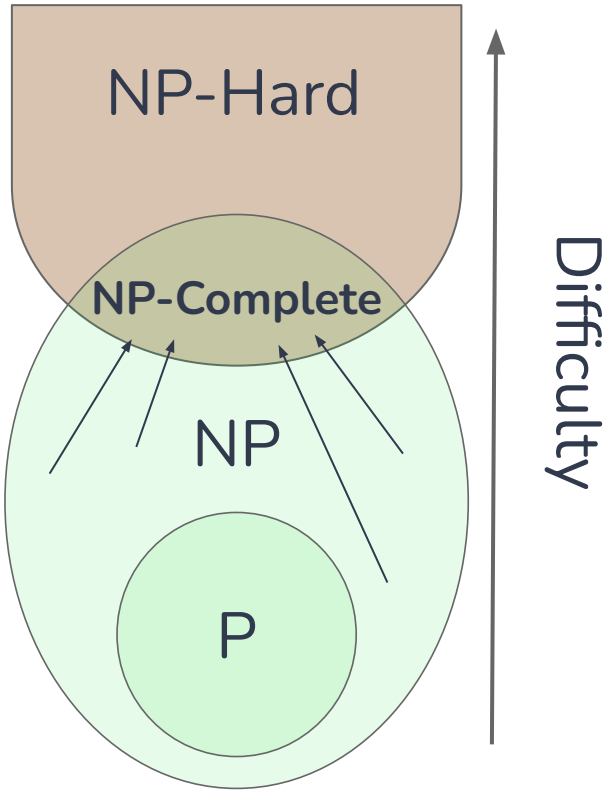
NP-Hard is the set of all problems where **if the problem can be solved in polynomial time, then every problem in NP can be solved in polynomial time.**

A problem is NP-Complete if it is in NP and NP-Hard.

A problem is NP-Complete if *every problem in NP has a poly-time reduction to it.*

NP-Complete problems are the *hardest* problems in NP.

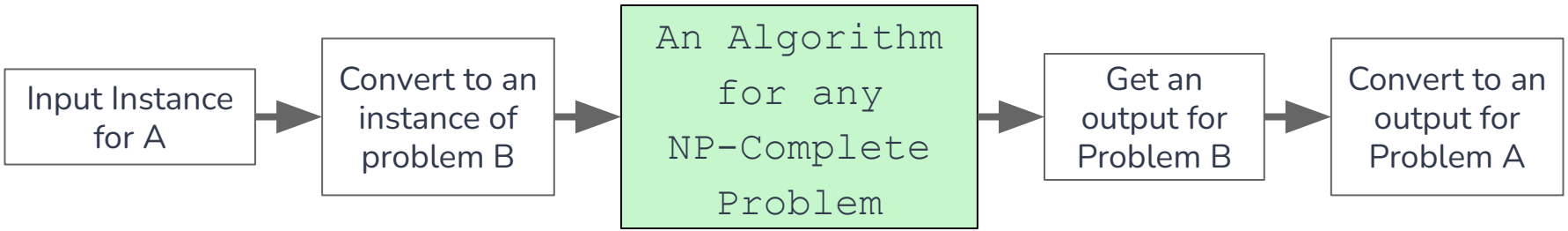
If any NP-Complete problem has a polynomial-time algorithm, then P=NP.



NP-Completeness: The Reduction Picture

All problems in NP have a polynomial-time reduction to any NP-complete problem.

So for any NP problem A and **any** NP-Complete problem B, you can do the following:



So if the algorithm for B is poly-time, then this is an algorithm for A in poly-time.

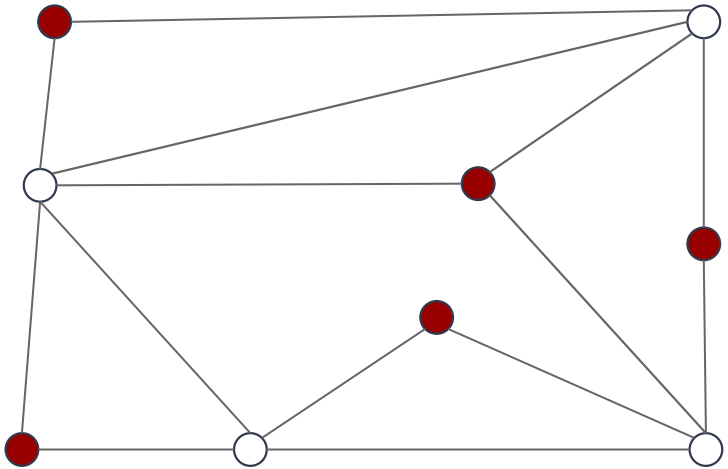
Your First NP-Complete Problems

Independent Set

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find the largest independent set.

Decision Problem: Is there an independent set of size 3? 4? 5?

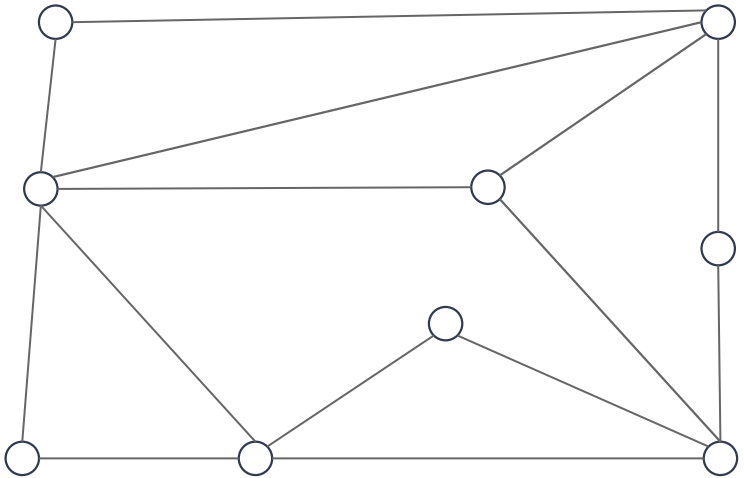


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

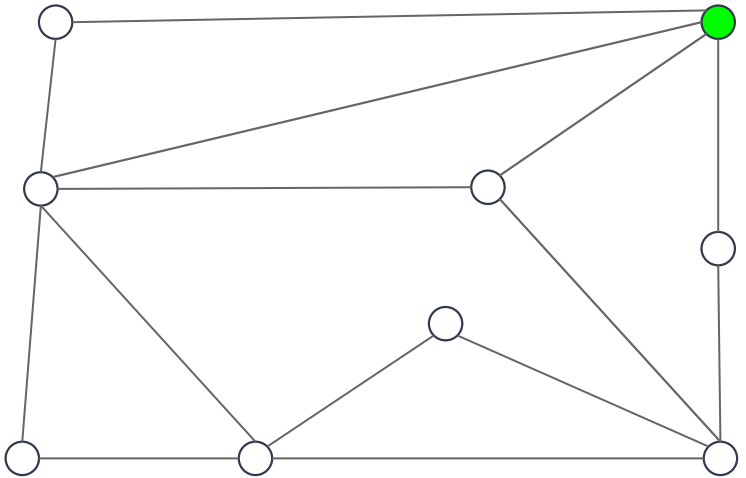


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

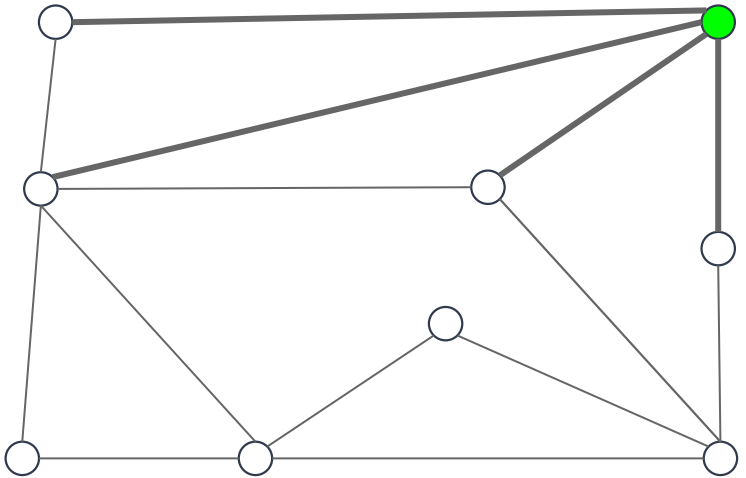


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

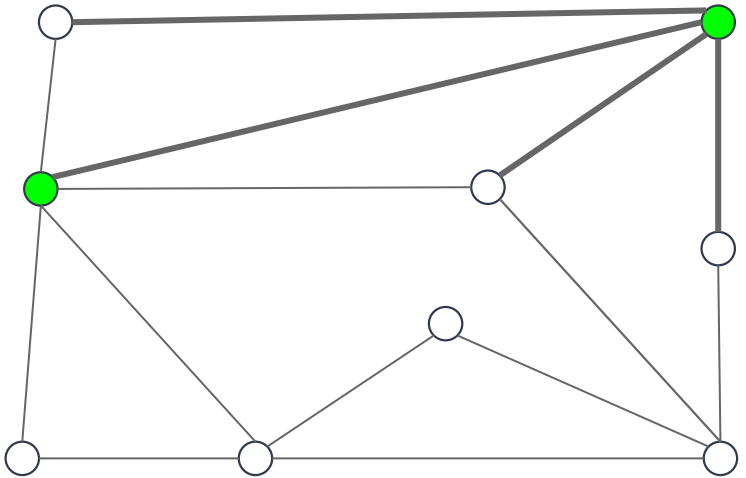


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

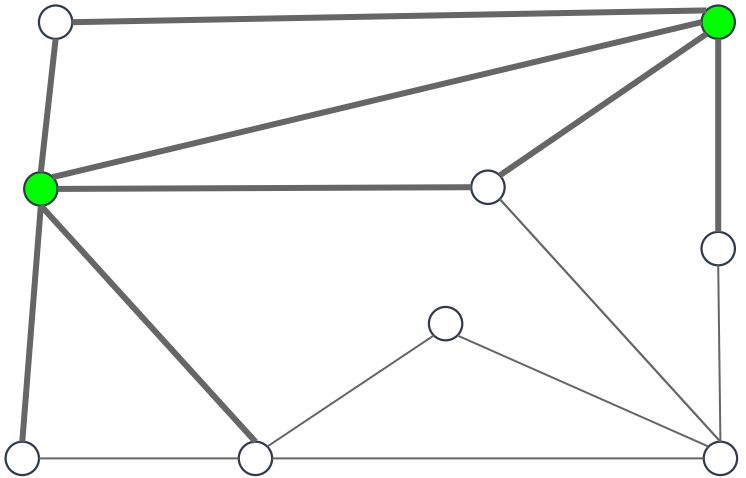


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

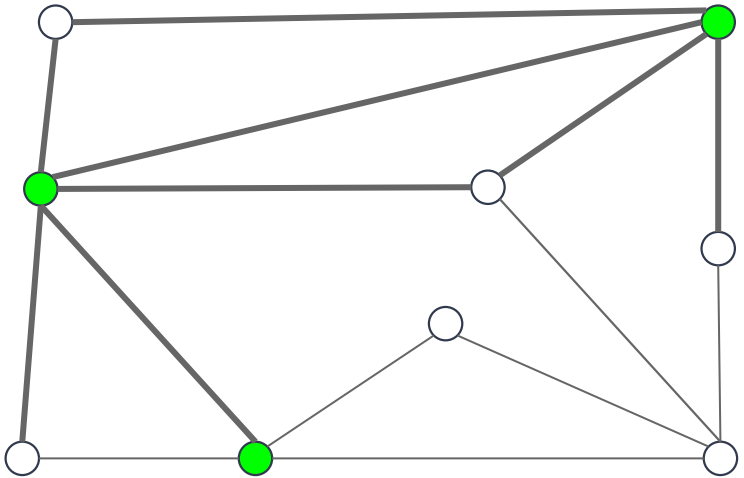


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

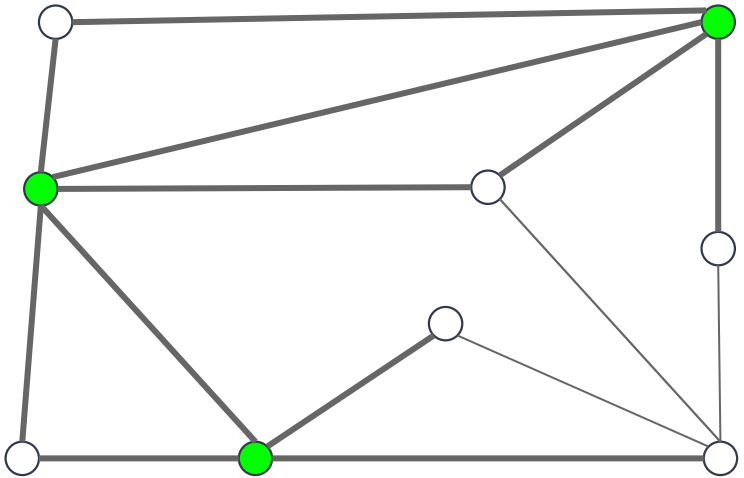


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

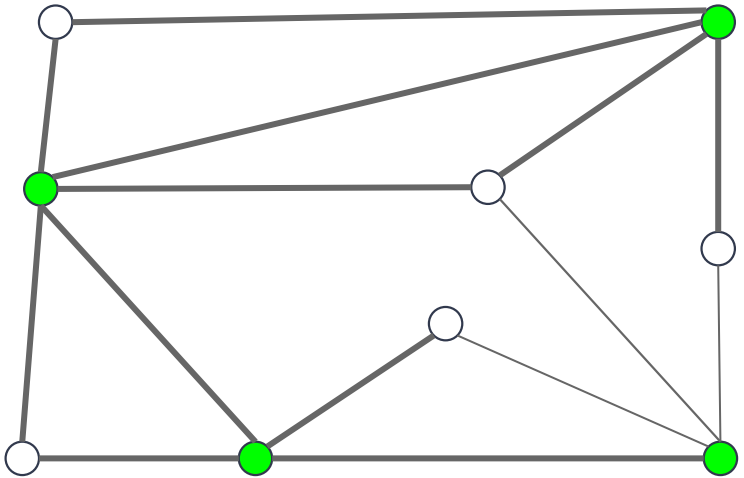


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?

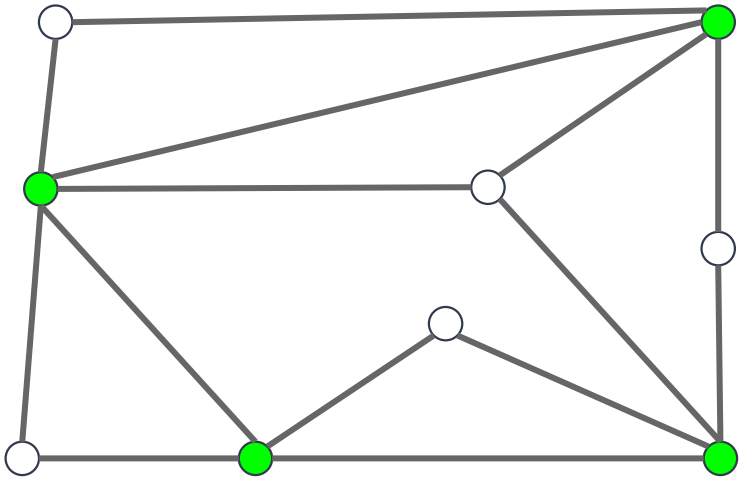


Vertex Cover

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **vertex cover problem** asks you to find the smallest vertex cover.

Decision Problem: Is there an vertex cover of size 3? 4? 5?



There is a vertex cover of size 4.

Independent Set vs. Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find an independent set of size j.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **Vertex Cover problem** asks you to find a vertex cover of size k.

Which problem is harder?

Reduce one problem to the other.

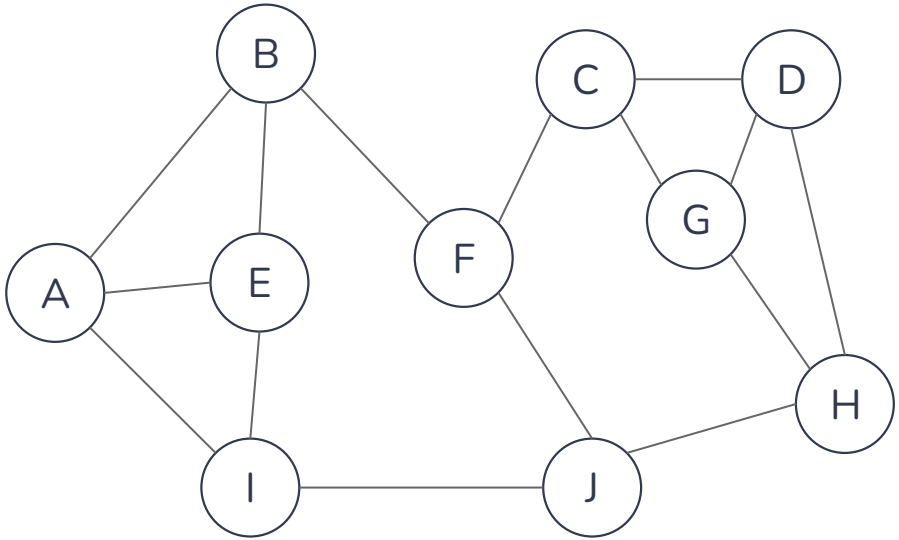
Independent Set vs. Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find an independent set of size j.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **Vertex Cover problem** asks you to find a vertex cover of size k.



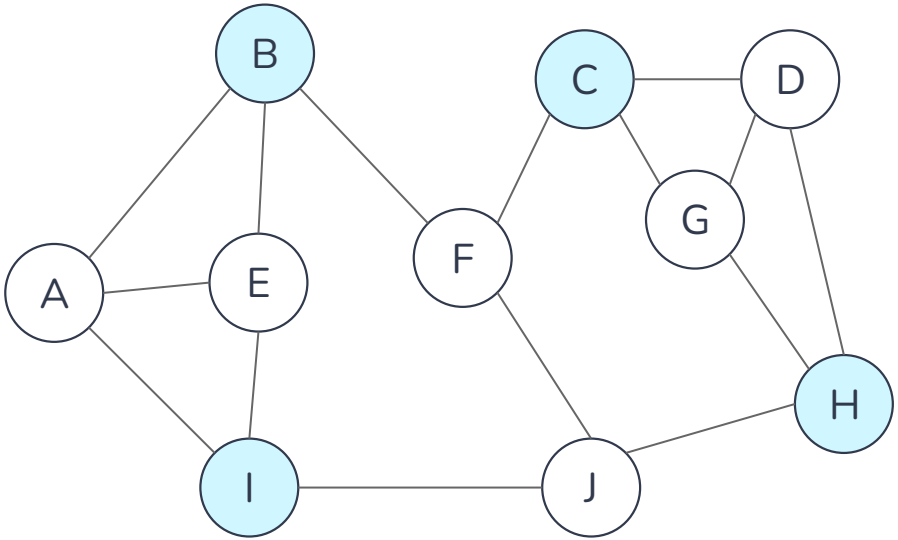
Independent Set vs. Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find an independent set of size j.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **Vertex Cover problem** asks you to find a vertex cover of size k.



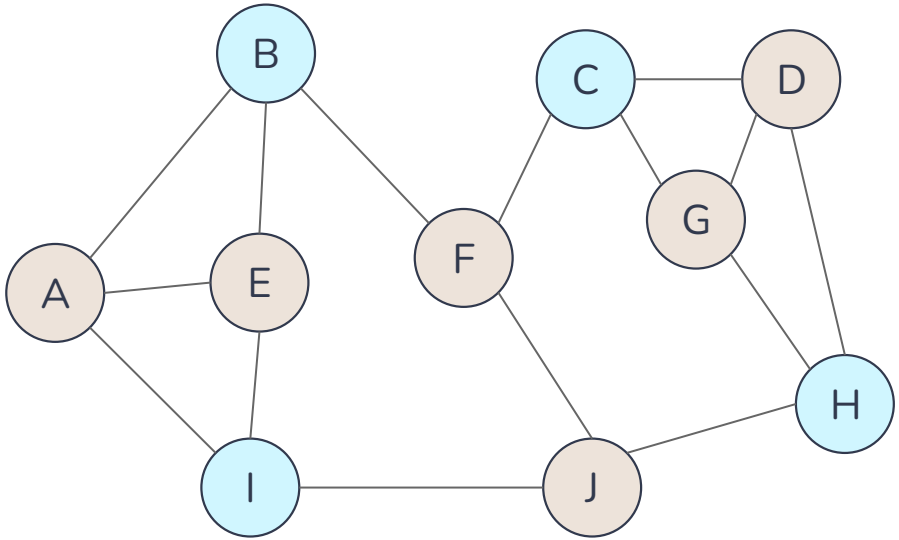
Independent Set vs. Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find an independent set of size j.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **Vertex Cover problem** asks you to find a vertex cover of size k.



If U is the best independent set, then V \ U is the best vertex cover. **Why?**

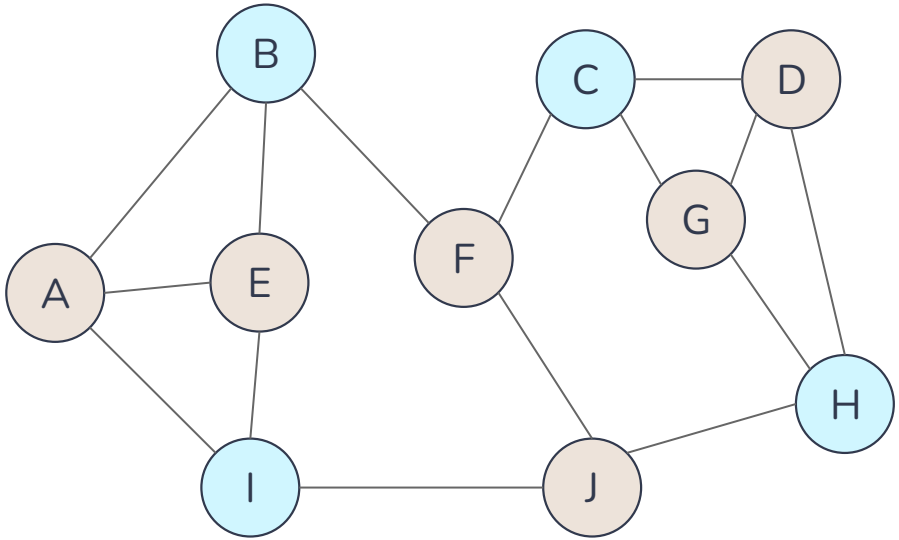
Independent Set vs. Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find an independent set of size j.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **Vertex Cover problem** asks you to find a vertex cover of size k.



If U is the best independent set, then V \ U is the best vertex cover. **Why?**

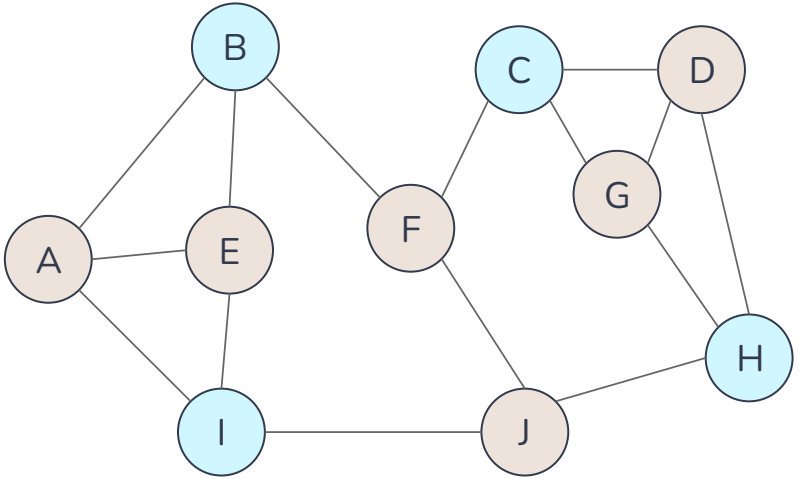
Independent Set vs. Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find an independent set of size j.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **Vertex Cover problem** asks you to find a vertex cover of size k.



Take any independent set. This set contains at most one vertex per edge. Therefore, every edge is next to a vertex not in the independent set. So, the set of vertices not in the independent set is a vertex cover.

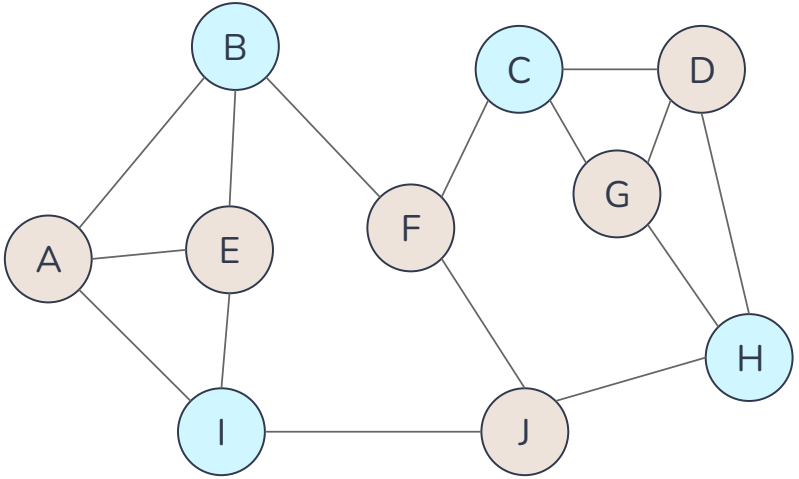
Independent Set vs. Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

The **Independent Set problem** asks you to find an independent set of size j.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

The **Vertex Cover problem** asks you to find a vertex cover of size k.

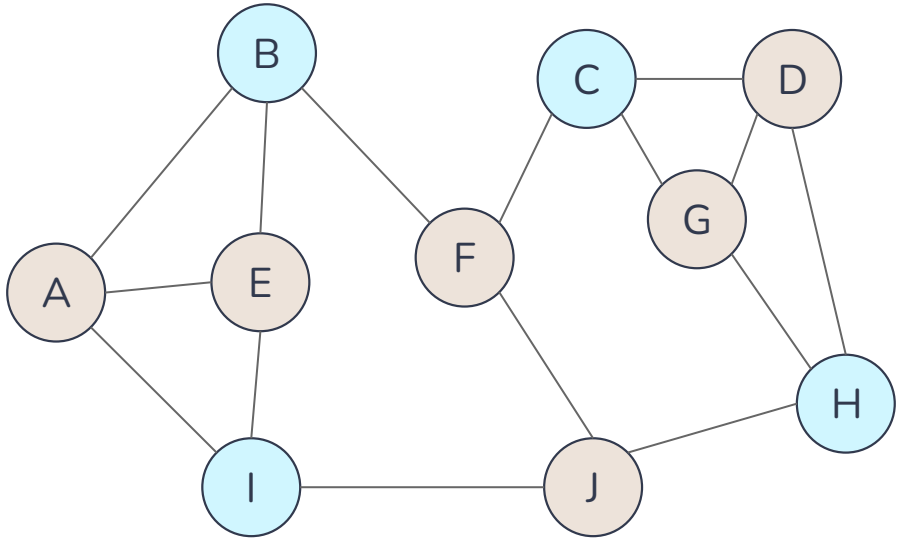


Take any vertex cover. If two vertices not in the vertex cover are adjacent, then the edge between them is not covered. Thus, not two vertices not in the vertex cover are adjacent, so they make an independent set.

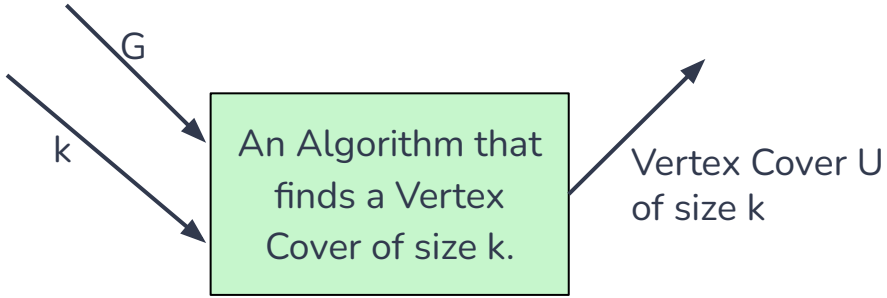
Reducing Independent Set to Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

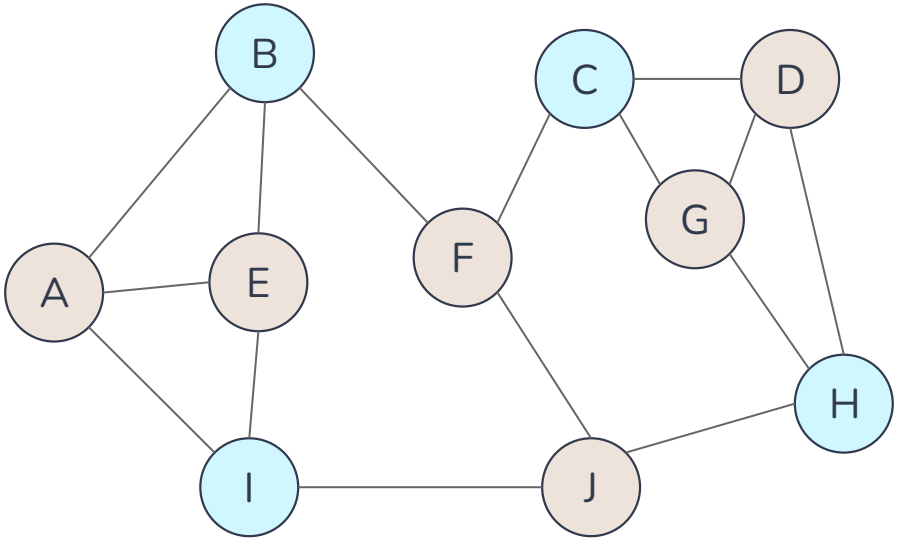


Want: Independent Set of size j.



Reducing Independent Set to Vertex Cover

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.



Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

Want: Independent Set of size j.

Take G, j

G

k=n-j

An Algorithm that finds a Vertex Cover of size k.

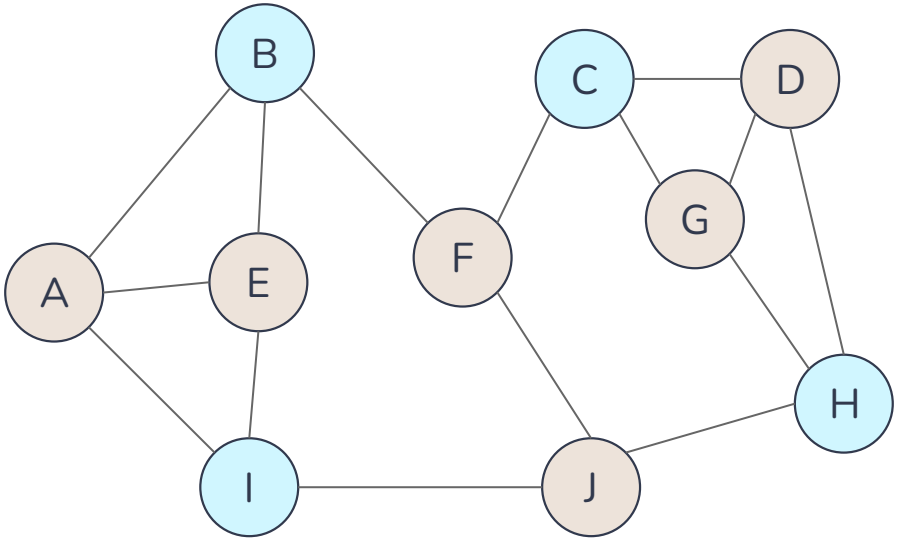
Vertex Cover U of size n-j

Output V \ U as the Independent Set!

This is a reduction from IndSet to Vertex Cover.

Reducing Vertex Cover to Independent Set

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.



Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.

Want: Vertex Cover of size k.

Take G, k

G

j=n-k

An Algorithm that finds an Independent Set of size j.

Output V \ U as the Vertex Cover!

Independent Set U of size n-k

This is a reduction from Vertex Cover to IndSet.

The Picture So Far

Given a graph G=(V, E), an **independent set** is a set of vertices U where no two vertices in U are adjacent.

Given a graph G=(V, E), a **vertex cover** is a set of vertices U where every edge is adjacent to a vertex in U.



These problems are equally hard!

3SAT

Constraint Satisfaction Problems

Consider a set of **boolean** variables w, x, y, z.

A **literal** is either a boolean variable x or its negation $\overline{x}$.

A **clause** is a set of *literals* connected by OR operations: $C = w \vee \overline{x} \vee y$.

A **CNF Formula** is a set of *clauses* connected by AND operations: $C_1 \wedge C_2 \wedge C_3$.

A **3-SAT Formula** is a CNF Formula where every *clause* has exactly 3 *literals*.

Example: $(\overline{w} \vee x \vee y) \wedge (w \vee x \vee \overline{y}) \wedge (\overline{x} \vee y \vee z)$

3-SAT Problem: Is there an assignment of booleans that makes this formula evaluate to TRUE?

One Possibility: w = TRUE, x = FALSE, y = TRUE, z = FALSE

3SAT Worksheet Problems

(a) No. The second clause is false, so the formula is FALSE.

(b) T,T,F,F: first clause works, second works, third works, fourth works

T,F,T,T: works!

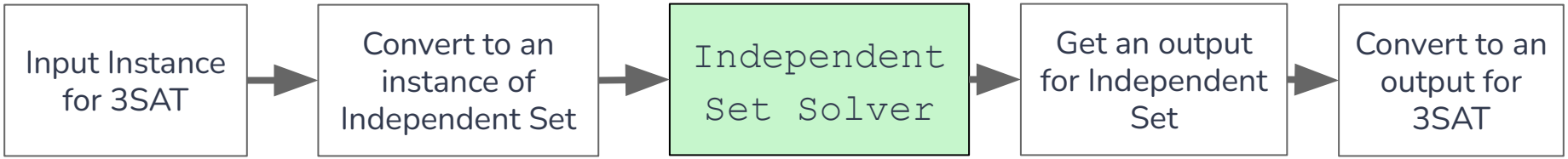
F,T,T,F: works!

T,F,F,T: works!

3SAT -> Independent Set

Reducing 3SAT to Independent Set

We're going to try something brave: we're going to create a reduction between two problems that seem to be quite different from one another.



A first step: Independent Set is on a graph with vertices and edges, while 3SAT input is a formula of literals and clauses. What might an vertex represent?

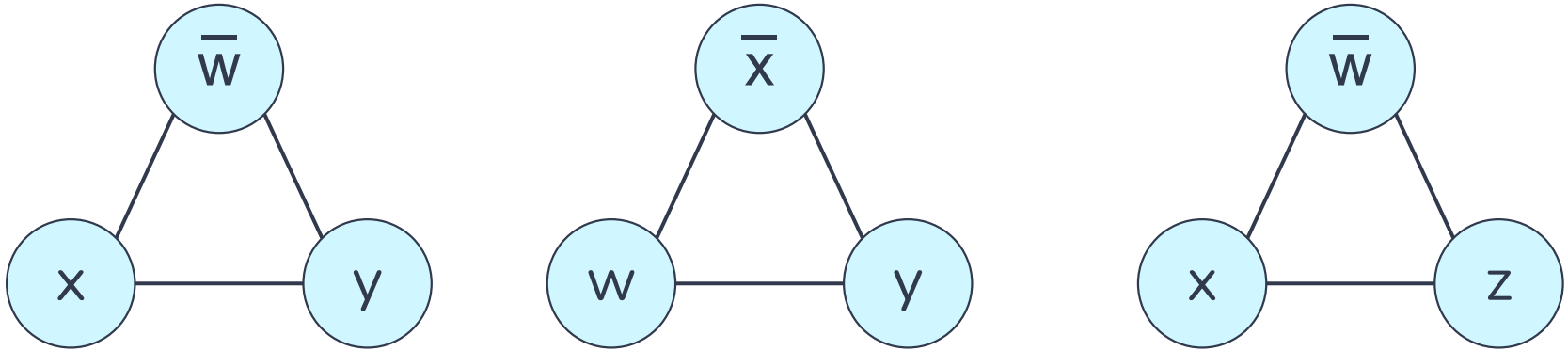
$$(\overline{w} \vee x \vee y) \wedge (w \vee \overline{x} \vee y) \wedge (\overline{w} \vee x \vee z)$$

A vertex might be a literal!

Reducing 3SAT to Independent Set: Construction

Convert each clause into a triangle of its literals.

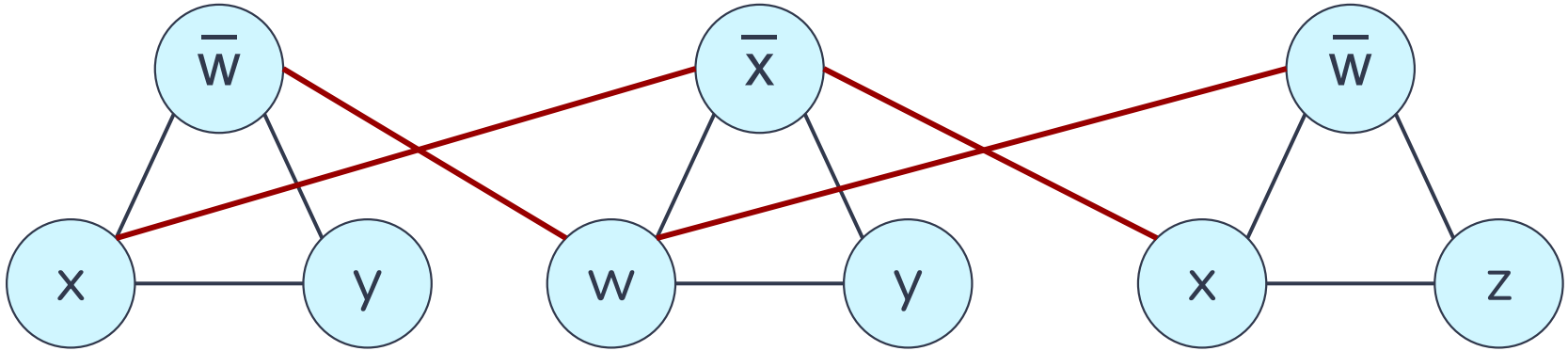
$(\overline{w} \vee x \vee y) \wedge (w \vee \overline{x} \vee y) \wedge (\overline{w} \vee x \vee z)$



Reducing 3SAT to Independent Set: Construction

Convert each clause into a triangle of its literals.
Connect each literal to its negations.

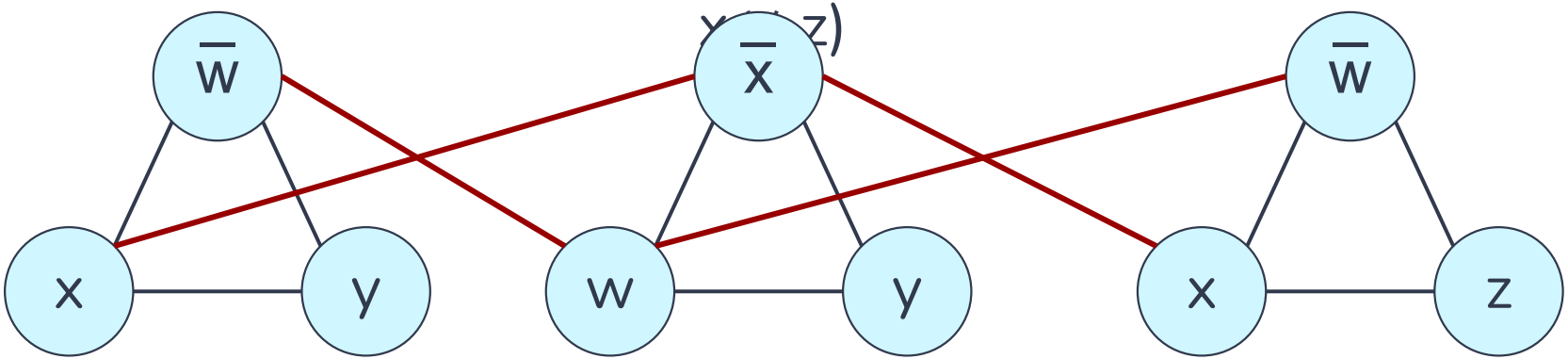
$(\overline{w} \vee x \vee y) \wedge (w \vee \overline{x} \vee y) \wedge (\overline{w} \vee x \vee z)$



Reducing 3SAT to Independent Set: Interpretation

Convert each clause into a triangle of its literals. Connect each literal to its negations.	If a vertex is in the ISet, then its value is True. No value and its negation can both be picked.

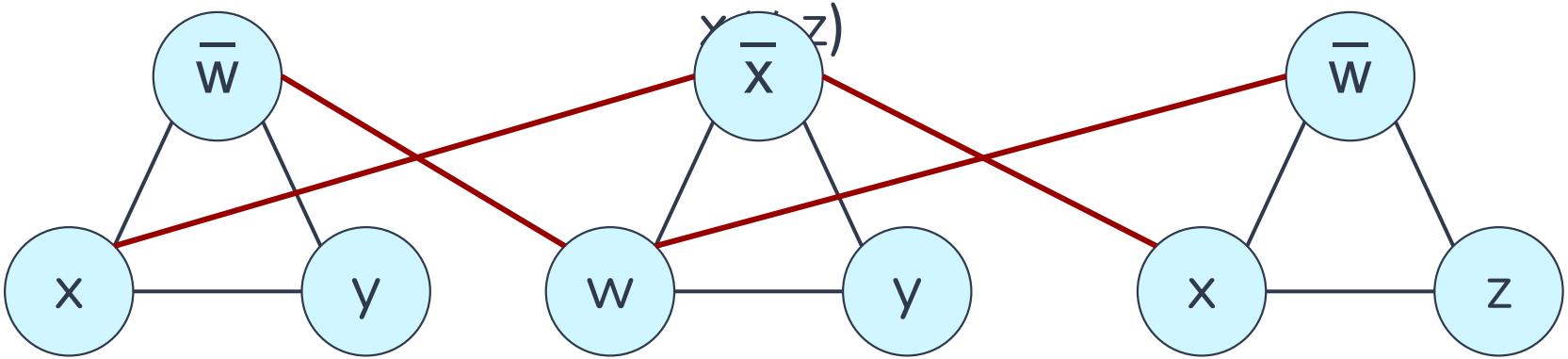
$$(\overline{w} \vee x \vee y) \wedge (w \vee \overline{x} \vee y) \wedge (\overline{w} \vee x \vee z)$$



Reducing 3SAT to Independent Set: Proof of Correctness

Need to Prove: (1) If this graph has an independent set equal to the number of clauses, then the 3SAT problem is satisfiable. (2) If it doesn't, then the 3SAT problem is not satisfiable.

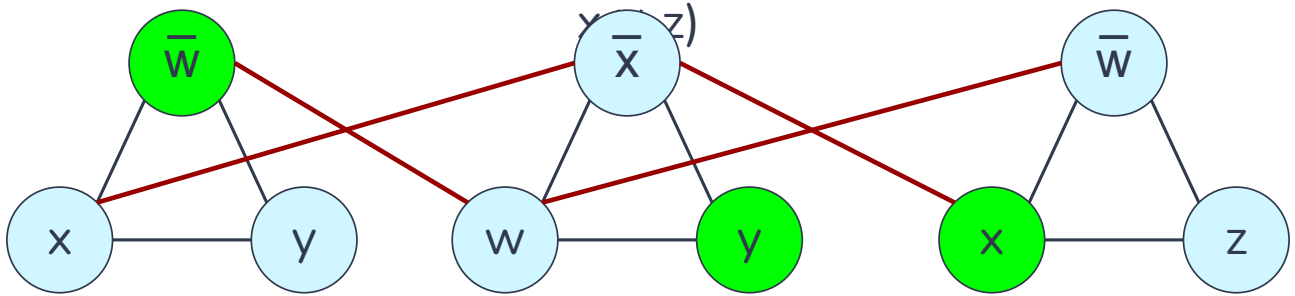
$(\overline{w} \vee x \vee y) \wedge (w \vee \overline{x} \vee y) \wedge (\overline{w} \vee$ $z)$



Reducing 3SAT to Independent Set: Proof of Correctness

Claim (1) If this graph has an independent set equal to the number of clauses, then the 3SAT problem is satisfiable.

$(\overline{w} \vee x \vee y) \wedge (w \vee \overline{x} \vee y) \wedge (\overline{w} \vee$ $x \vee z)$



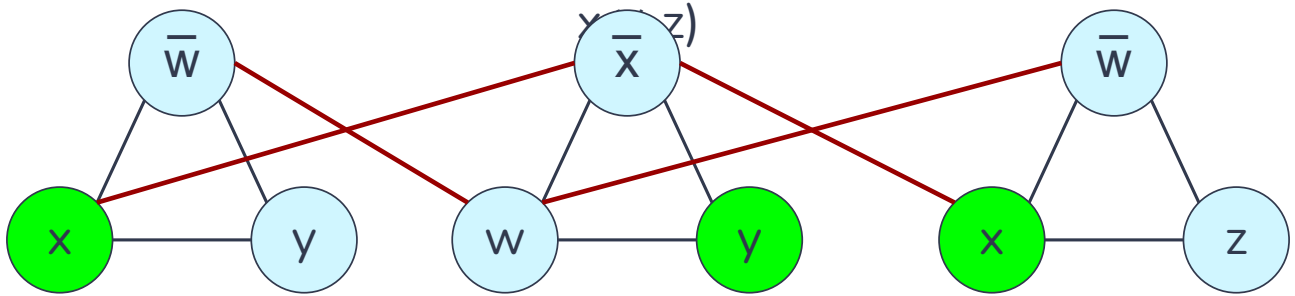
Set $\overline{w}$=T
Set y=T
Set x=T

Proof: Say there is an independent set equal to the number of clauses. Then, because each clause is a triangle there is only one selected vertex per clause. We set the literal of that vertex to T. Then every clause evaluates to T, and there is no contradiction because a literal and its negation can't both be selected. Therefore, this makes a satisfying assignment.

Reducing 3SAT to Independent Set: Proof of Correctness

Claim (2) If it doesn't, then the 3SAT problem is not satisfiable. We prove the contrapositive: if the 3SAT problem is satisfiable, there is an independent set equal to the number of clauses.

$(\overline{w} \vee x \vee y) \wedge (w \vee \overline{x} \vee y) \wedge (\overline{w} \vee$ $x \vee z)$

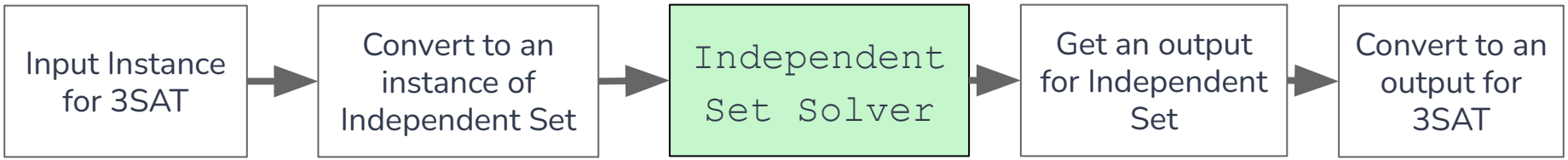


Satisfying Assignment: (w, x, y, z) = (T, T, T, F)

Proof: Say there is a satisfying assignment. Then, in each clause there is at least one literal that evaluates to true. Select the corresponding vertex on the graph to add to the Independent Set. These vertices are all independent from one another because a vertex and its negation are never both selected. Therefore, there is an independent set equal to the number of clauses.

Reducing 3SAT to Independent Set

We're going to try something brave: we're going to create a reduction between two problems that seem to be quite different from one another.

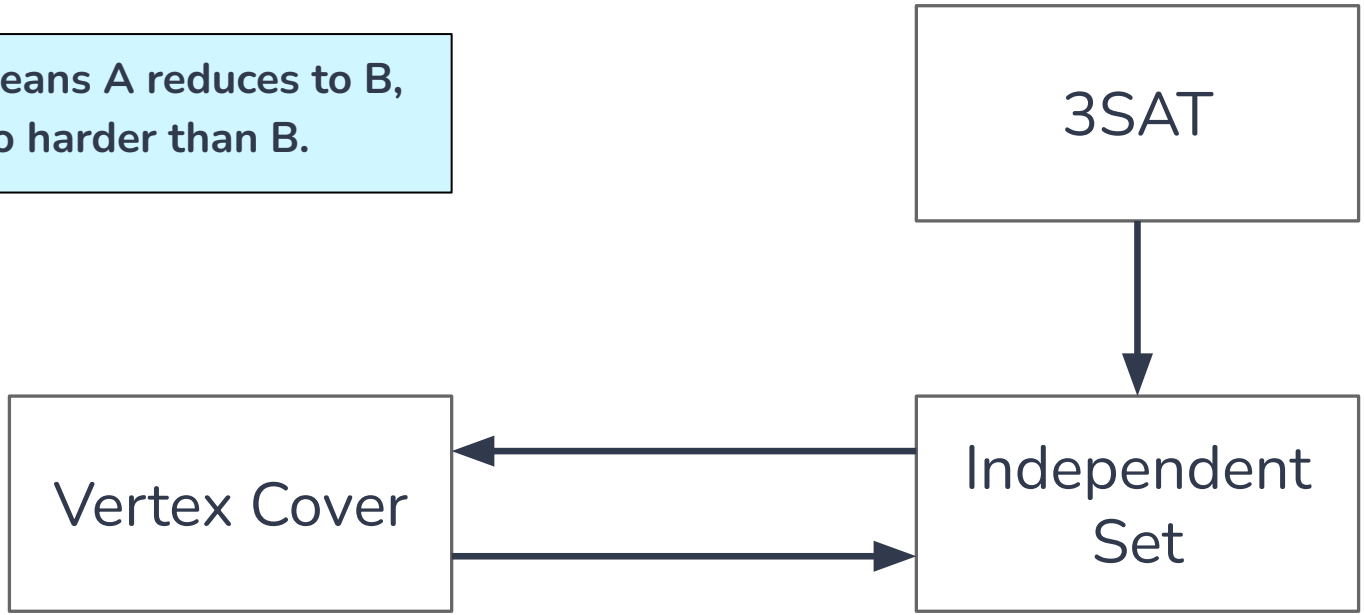


Using this construction, we have shown that 3SAT reduces to Independent Set. So what do we conclude?

3SAT is no harder than Independent Set.

The Picture Today

A -> B means A reduces to B, so A is no harder than B.



Lecture Recap

Today we introduced the concept of NP-Completeness. Then, we talked about our first three NP-complete problems, and showed some reductions between some of them.