Introduction to Problem Hardness

CS/MCS 401: Computer Algorithms I
Lecture 22
Professor: Jonathan Liu

Icebreaker: What has been the hardest problem from class so far?

Today's Overview

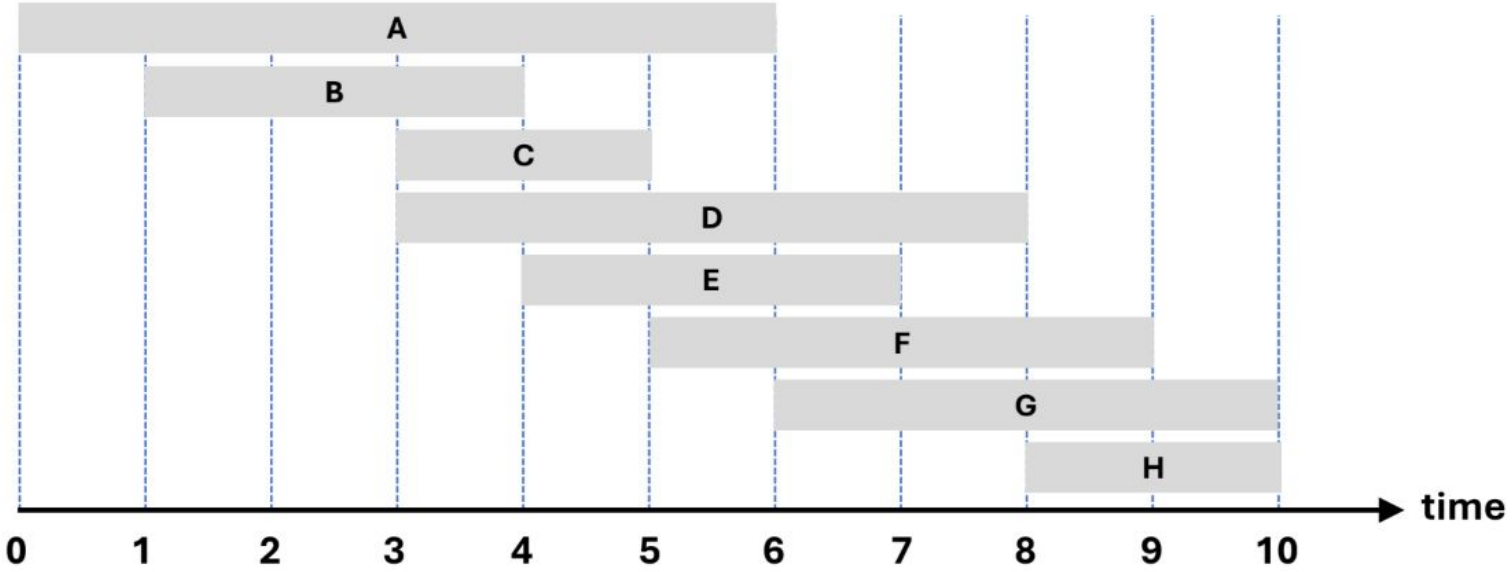
Today is the start of a new unit! In this unit, we're going to be focused primarily on one question: How can we tell if a problem can't be solved efficiently?

Today we're going to mainly build a framework for talking about this by answering a simpler question: how do you compare the hardness of two different problems?

Problem Hardness

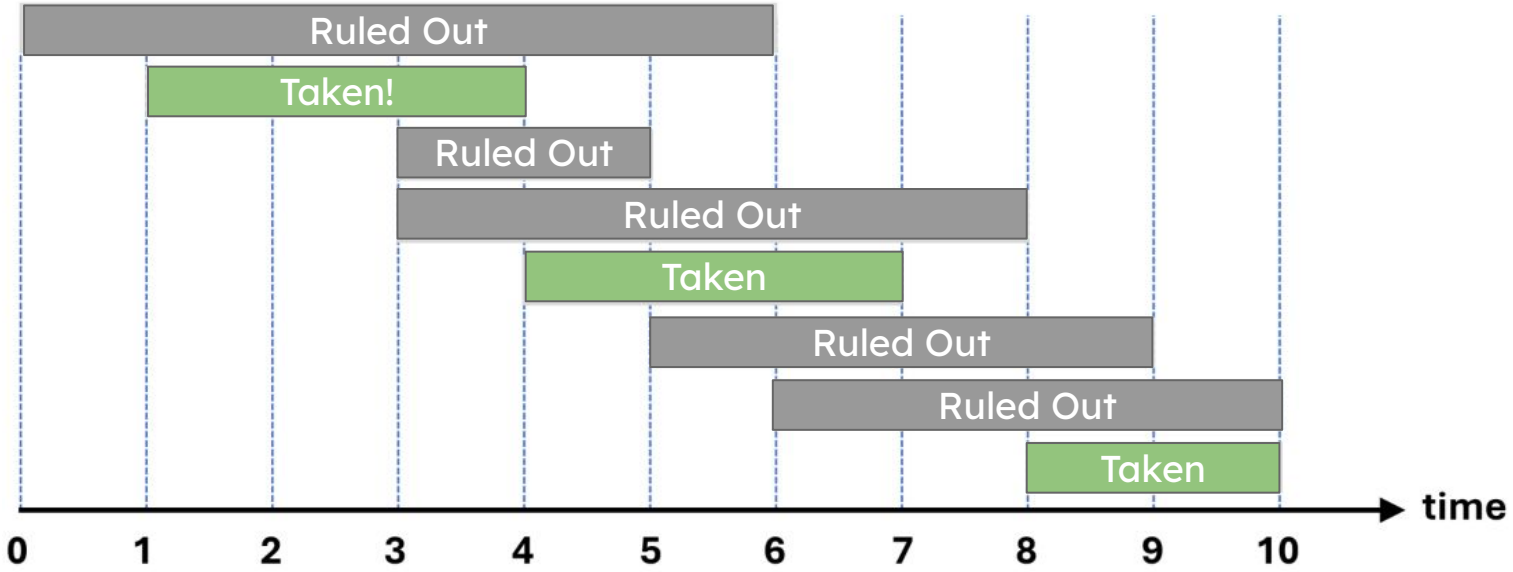
Course Scheduling

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.



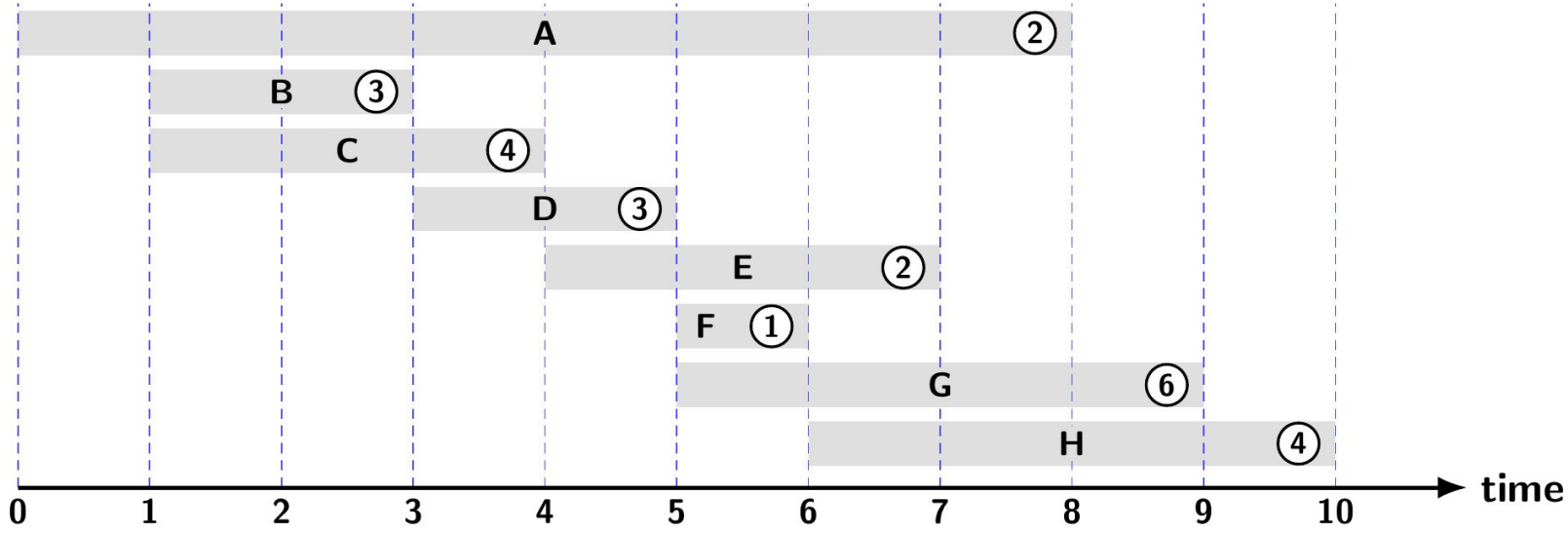
Course Scheduling

You're only willing to come onto campus one day per week, so you'd like to squeeze as many classes as you can into the one day.



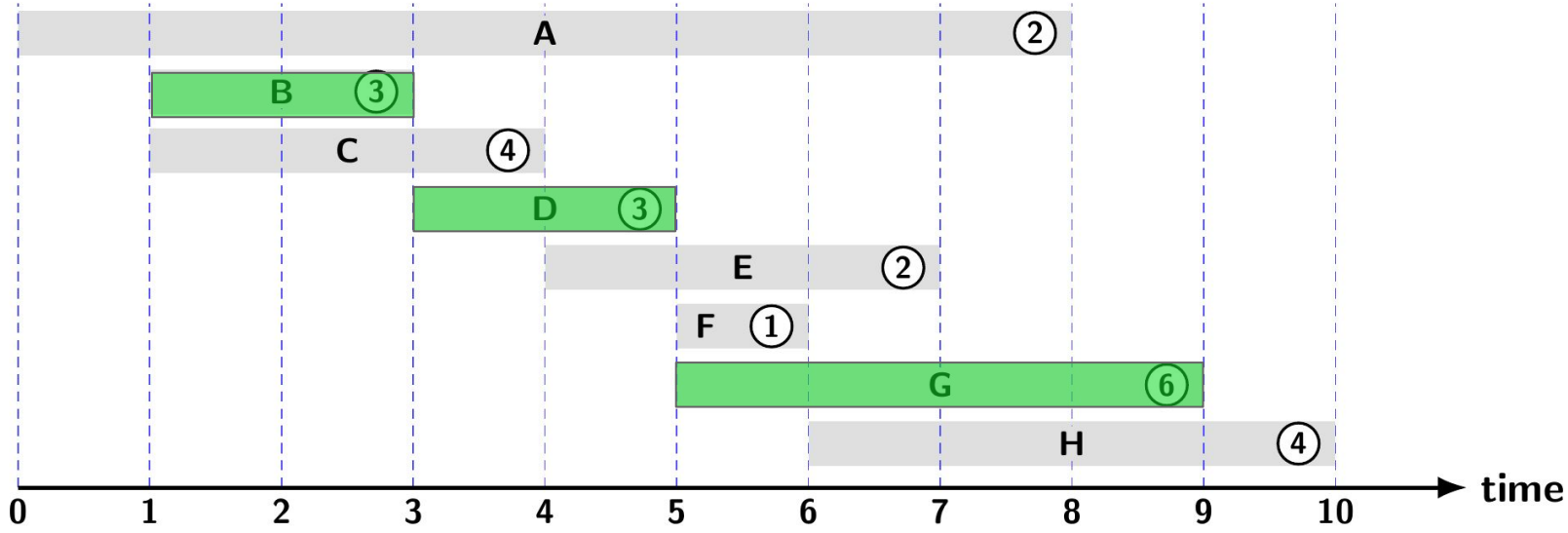
Weighted Course Scheduling

You're only willing to come onto campus one day per week. Each class has an importance level displayed. Find the feasible schedule with maximum importance.



Weighted Course Scheduling

You're only willing to come onto campus one day per week. Each class has an importance level displayed. Find the feasible schedule with maximum importance.



Some Examples: Which is harder?

Course Scheduling	$\leq$	Weighted Course Scheduling
Shortest Paths with Negative Edges	$\geq$	Shortest Paths with no Negative Edges
Find the Minimum Spanning Tree	$\leq$	Teleporters

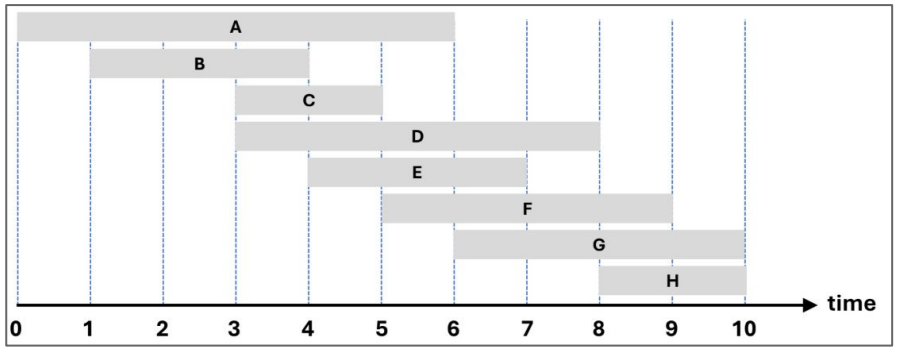
What does it mean for a problem to be hard?

- More details in the problem
- More steps in the problem
- More constraints
- **If one is a subproblem of the other then the subproblem is easier**
-
- If certain parts of the input are dependent on each other
- *Whether there is an efficient way to solve it*
  - Large size/domain of the input
  - Runtime to brute force

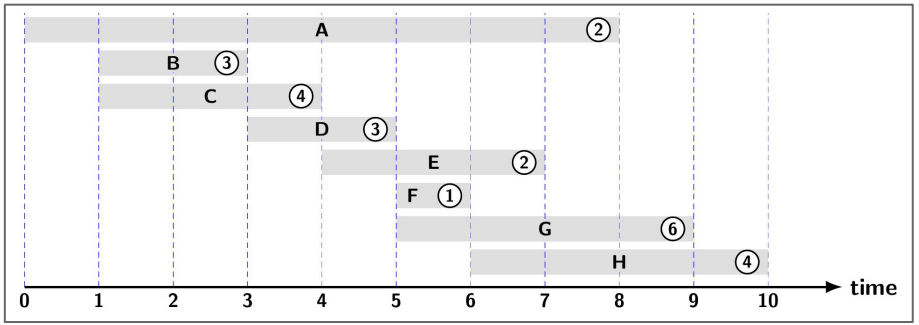
Reductions: Our Measurement of Difficulty

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

Course Scheduling



Weighted Course Scheduling

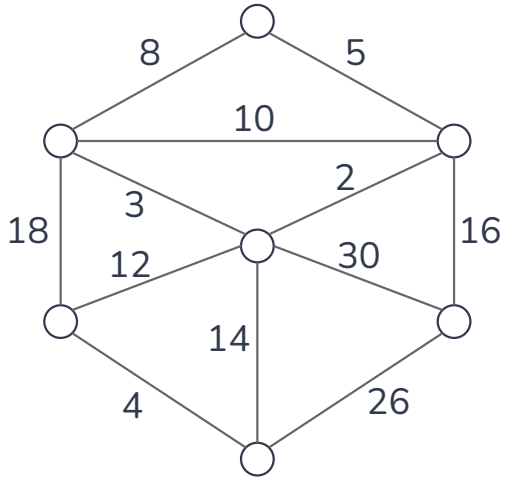


A Couple of Definitions

A **problem**, for this course, is a set of inputs and a set of desired outputs.

An **instance** is a single input to a problem.

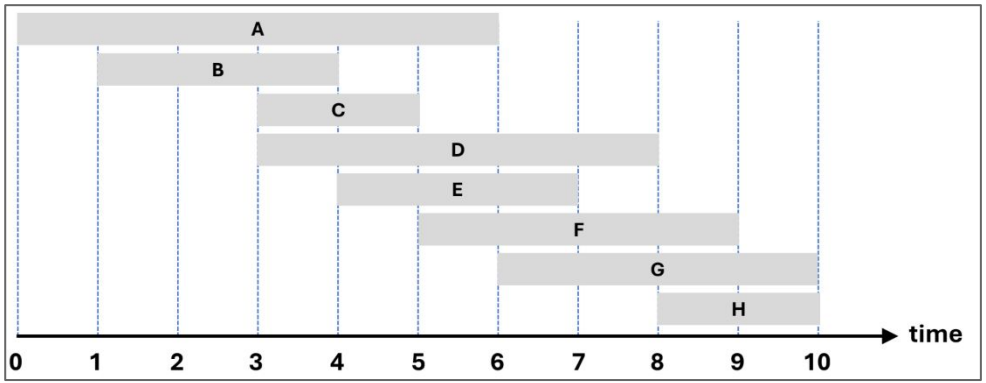
"Find the minimum spanning tree of a graph."



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 on the right to solve it?

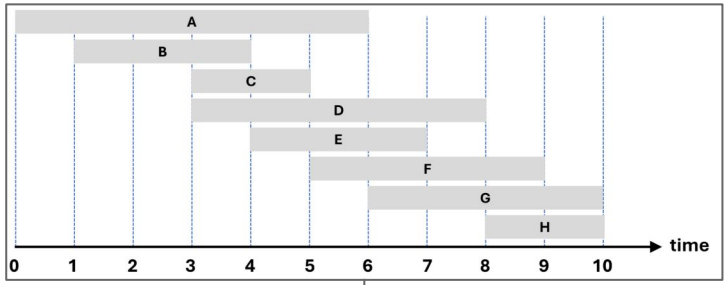


An Algorithm that solves *Weighted* Course Scheduling

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

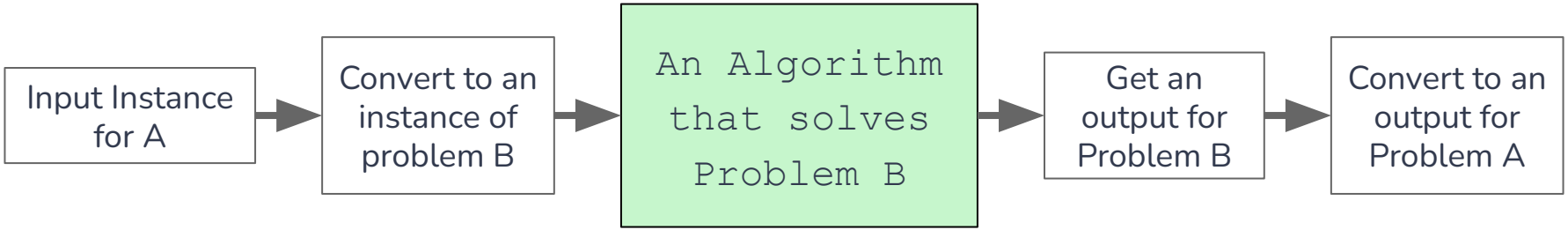
The highest-weight set of classes

Output that set!

Reductions

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

Note: A reduction may call the algorithm multiple times if needed.



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.**

Worksheet Problems

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.**

We use an algorithm for MST to solve Teleporters. Thus, Teleporters reduces to MST, so **MST is harder than teleporters**.

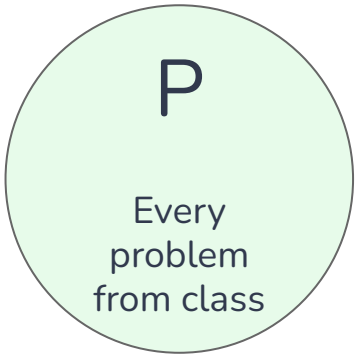
If I have an algorithm for Teleporters, I can use that to find an MST of any graph. So, MST reduces to Teleporters, so **Teleporters is harder than MST**.

P, NP, and NP-completeness

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.



A Couple of Definitions

A **search problem** is a problem that is looking for an output satisfying something.

A **decision problem** is a problem whose answer is "yes" or "no".

Decision problems are just as hard as search problems. For example, the two problems can be reduced to each other. You'll see why in the homework.

For the rest of today, we'll only be talking about decision problems.

"Find the minimum spanning tree of a graph."

"Is there a spanning tree with weight 40 or less?"

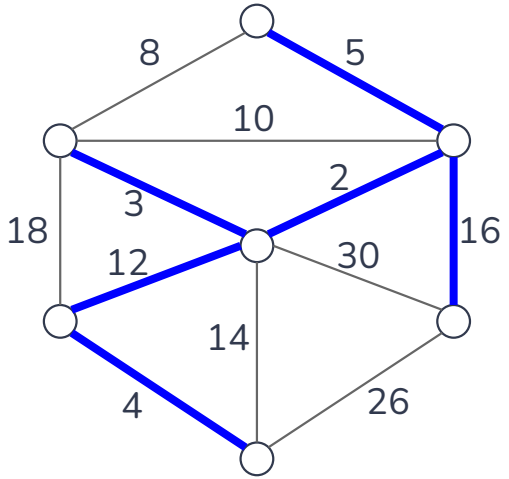
NP (Complexity Class)

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?”



Solving vs. Verifying

Solving

9							7	5
7			4	2				
						8		4
	9	3		8		7		
			2	3	1			
	5				9	2	4	8
					4		5	
		7					6	

Verifying

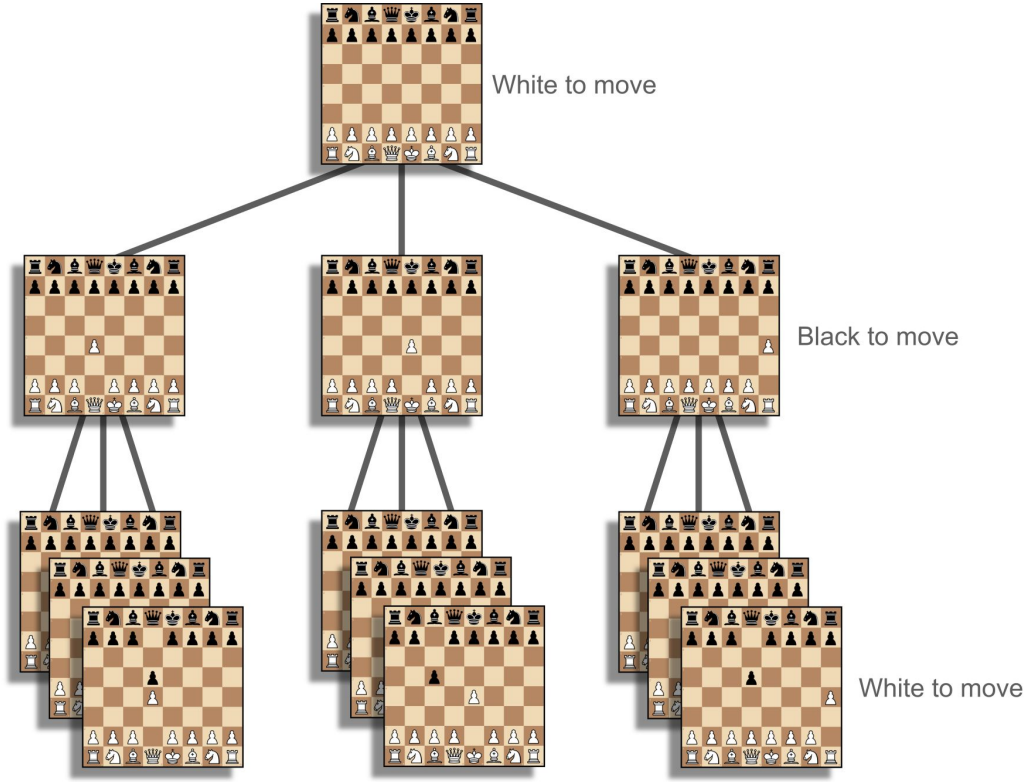
9	2	4	6	1	8	3	7	5
7	3	8	4	2	5	6	9	1
5	6	1	7	9	3	8	2	4
4	9	3	5	8	6	7	1	2
6	7	5	2	3	1	4	8	9
8	1	2	9	4	7	5	3	6
3	5	6	1	7	9	2	4	8
2	8	9	3	6	4	1	5	7
1	4	7	8	5	2	9	6	3

Can everything be verified efficiently?

Q: Is there a way to always win in chess?

Yes. Here's my strategy:

There are around 10^{40} "sensible" chess games. You'd have to check that my strategy works for them all!



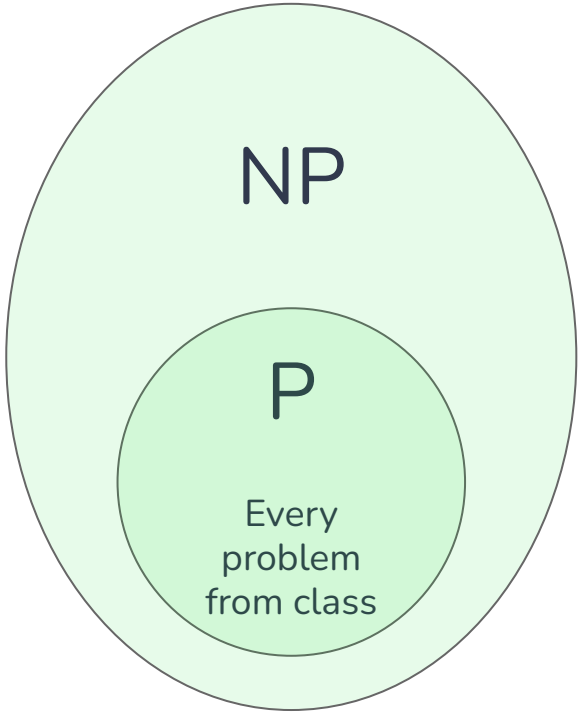
P vs. NP

If a problem can be solved in polynomial time, then it can definitely be verified in polynomial time. You don't even need a certificate anymore – you can just run the algorithm yourself! So **P is in 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!



Lecture Recap

Today we introduced **reductions**, our tool for comparing the hardness of algorithmic problems.

Then, we introduced the complexity classes P and NP, as well as the P vs. NP debate.