Recap from Last Time
The Cobham-Edmonds Thesis

A language $L$ can be \textit{decided efficiently} if there is a TM that decides it in polynomial time.

Equivalently, $L$ can be decided efficiently if it can be decided in time $O(n^k)$ for some $k \in \mathbb{N}$.

Like the Church-Turing thesis, this is \textit{not} a theorem! It's an assumption about the nature of efficient computation, and it is somewhat controversial.
The Complexity Class $\text{P}$

- The *complexity class* $\text{P}$ (for *polynomial* time) contains all problems that can be solved in polynomial time.

- Formally:
  
  $$\text{P} = \{ L \mid \text{There is a polynomial-time decider for } L \}$$

- Assuming the Cobham-Edmonds thesis, a language is in $\text{P}$ if it can be decided efficiently.
Polynomial-Time Verifiers

• A **polynomial-time verifier** for $L$ is a TM $V$ such that
  • $V$ halts on all inputs.
  • $w \in L$ iff $\exists c \in \Sigma^*$. $V$ accepts $\langle w, c \rangle$.
  • $V$'s runtime is a polynomial in $|w|$ (that is, $V$'s runtime is $O(|w|^k)$ for some integer $k$)
The Complexity Class \textbf{NP}

• The complexity class \textbf{NP} (\textit{nondeterministic polynomial time}) contains all problems that can be verified in polynomial time.

• Formally:

\[
\text{NP} = \{ L \mid \text{There is a polynomial-time verifier for } L \}\]

• The name \textbf{NP} comes from another way of characterizing \textbf{NP}. If you introduce \textit{nondeterministic Turing machines} and appropriately define “polynomial time,” then \textbf{NP} is the set of problems that an NTM can solve in polynomial time.
**Theorem (Baker-Gill-Solovay):** Any proof that purely relies on universality and self-reference cannot resolve $P \neq NP$.

**Proof:** Take CS154!
So how are we going to reason about \textbf{P} and \textbf{NP}?
New Stuff!
A Challenge
Problems in \textbf{NP} vary widely in their difficulty, even if \textbf{P} = \textbf{NP}.

How can we rank the relative difficulties of problems?
Reducibility
Maximum Matching

• Given an undirected graph \( G \), a **matching** in \( G \) is a set of edges such that no two edges share an endpoint.

• A **maximum matching** is a matching with the largest number of edges.

A matching, but not a maximum matching.
Maximum Matching

- Given an undirected graph $G$, a **matching** in $G$ is a set of edges such that no two edges share an endpoint.

- A **maximum matching** is a matching with the largest number of edges.
Maximum Matching

- Given an undirected graph $G$, a matching in $G$ is a set of edges such that no two edges share an endpoint.
- A maximum matching is a matching with the largest number of edges.
Maximum Matching

• Jack Edmonds' paper “Paths, Trees, and Flowers” gives a polynomial-time algorithm for finding maximum matchings.

  • (This is the same Edmonds as in “Cobham-Edmonds Thesis.”)

• Using this fact, what other problems can we solve?
Solving Domino Tiling
Solving Domino Tiling
Solving Domino Tiling
In Pseudocode

boolean canPlaceDominoes(Grid G, int k) {
    return hasMatching(gridToGraph(G), k);
}
Based on this connection between maximum matching and domino tiling, which of the following statements would be more proper to conclude?

A. Finding a maximum matching isn’t any more difficult than tiling a grid with dominoes.

B. Tiling a grid with dominoes isn’t any more difficult than finding a maximum matching.

Answer at PollEv.com/cs103 or text CS103 to 22333 once to join, then A or B.
Intuition:

Tiling a grid with dominoes can't be “harder” than solving maximum matching, because if we can solve maximum matching efficiently, we can solve domino tiling efficiently.
Another Example
Reachability

• Consider the following problem:

  Given an directed graph \( G \) and nodes \( s \) and \( t \) in \( G \), is there a path from \( s \) to \( t \)?

• It's known that this problem can be solved in polynomial time (use DFS or BFS).

• Given that we can solve the reachability problem in polynomial time, what other problems can we solve in polynomial time?
Converter Conundrums

• Suppose that you want to plug your laptop into a projector.

• Your laptop only has a VGA output, but the projector needs HDMI input.

• You have a box of connectors that convert various types of input into various types of output (for example, VGA to DVI, DVI to DisplayPort, etc.)

• Question: Can you plug your laptop into the projector?
### Converter Conundrums

**Connectors**
- RGB to USB
- VGA to DisplayPort
- DB13W3 to CATV
- DisplayPort to RGB
- DB13W3 to HDMI
- DVI to DB13W3
- S-Video to DVI
- FireWire to SDI
- VGA to RGB
- DVI to DisplayPort
- USB to S-Video
- SDI to HDMI
In Pseudocode

```java
boolean canPlugIn(List<Plug> plugs) {
    return isReachable(plugsToGraph(plugs), VGA, HDMI);
}
```
Based on this connection between plugging a laptop into a projector and determining reachability, which of the following statements would be more proper to conclude?

A. Plugging a laptop into a projector isn’t any more difficult than computing reachability in a directed graph.

B. Computing reachability in a directed graph isn’t any more difficult than plugging a laptop into a projector.
**Intuition:**

Finding a way to plug a computer into a projector can't be "harder" than determining reachability in a graph, since if we can determine reachability in a graph, we can find a way to plug a computer into a projector.
bool solveProblemA(string input) {
    return solveProblemB(transform(input));
}

**Intuition:**

Problem A can't be “harder” than problem B, because solving problem B lets us solve problem A.
```cpp
bool solveProblemA(string input) {
    return solveProblemB(transform(input));
}
```

- If $A$ and $B$ are problems where it's possible to solve problem $A$ using the strategy shown above*, we write
  \[ A \leq_p B. \]

- We say that $A$ is polynomial-time reducible to $B$.

* Assuming that `transform` runs in polynomial time.
Polynomial-Time Reductions

- If $A \leq_p B$ and $B \in \text{P}$, then $A \in \text{P}$.
- If $A \leq_p B$ and $B \in \text{NP}$, then $A \in \text{NP}$. 
This $\leq_p$ relation lets us rank the relative difficulties of problems in \textbf{P} and \textbf{NP}.

What else can we do with it?
Time-Out for Announcements!
Please evaluate this course on Axess.

Your feedback makes a difference.
Problem Set Nine

• Problem Set Nine is due this Friday at 2:30PM.
  • *No late submissions can be accepted.* This is university policy – sorry!
• As always, if you have questions, stop by office hours or ask on Piazza!
Final Exam Logistics

• Our final exam is Monday, March 19th from 3:30PM – 6:30PM, location Hewlett 200 & 201 (no special last name assignments).
  • Sorry about how soon that is – the registrar picked this time, not us. If we had a choice, it would be on the last day of finals week.
• The exam is cumulative. You’re responsible for topics from PS1 – PS9 and all of the lectures.
• As with the midterms, the exam is closed-book, closed-computer, and limited-note. You can bring one double-sided sheet of 8.5” × 11” notes with you to the exam, decorated any way you’d like.
• Students with OAE accommodations: if we don’t yet have your OAE letter, please send it to us ASAP.
Preparing for the Final

- On the course website you’ll find
  - six practice final exams, which are all real exams with minor modifications, with solutions, and
  - a giant set of 46 practice problems (EPP3), with solutions.
- Our recommendation: Look back over the exams and problem sets and redo any problems that you didn’t really get the first time around.
- Keep the TAs in the loop: stop by office hours to have them review your answers and offer feedback.
Practice Final Exam

- We will be holding a practice final exam in room 380-380X tonight from 7PM – 10PM.
- We’ll print out copies of a few of the different practice exams and you can pick whichever one you’d like!
Back to CS103!
NP-Hardness and NP-Completeness
Question: What makes a problem hard to solve?
**Intuition:** If $A \leq_p B$, then problem $B$ is at least as hard* as problem $A$.

* for some definition of “at least as hard as.”
**Intuition:** To show that some problem is hard, show that lots of other problems reduce to it.
**NP-Hardness**

- A language $L$ is called *NP-hard* if for every $A \in \text{NP}$, we have $A \leq_P L$.

  Intuitively: $L$ has to be at least as hard as every problem in $\text{NP}$, since an algorithm for $L$ can be used to decide all problems in $\text{NP}$.
A language $L$ is called **NP-hard** if for every $A \in \text{NP}$, we have $A \leq_p L$.

A language in $L$ is called **NP-complete** if $L$ is NP-hard and $L \in \text{NP}$.

The class $\text{NPC}$ is the set of NP-complete problems.
A language \( L \) is called **NP-hard** if for every \( A \in \text{NP} \), we have \( A \leq_p L \).

A language in \( L \) is called **NP-complete** if \( L \) is NP-hard and \( L \in \text{NP} \).

The class **NPC** is the set of NP-complete problems.
The Tantalizing Truth

**Theorem:** If any NP-complete language is in P, then P = NP.

**Proof:** Suppose that L is NP-complete and L ∈ P. Now consider any arbitrary NP problem A. Since L is NP-complete, we know that A ≤ₚ L. Since L ∈ P and A ≤ₚ L, we see that A ∈ P. Since our choice of A was arbitrary, this means that NP ⊆ P, so P = NP. ■
**Theorem:** If any \( \text{NP} \)-complete language is not in \( \text{P} \), then \( \text{P} \neq \text{NP} \).

**Proof:** Suppose that \( L \) is an \( \text{NP} \)-complete language not in \( \text{P} \). Since \( L \) is \( \text{NP} \)-complete, we know that \( L \in \text{NP} \). Therefore, we know that \( L \in \text{NP} \) and \( L \notin \text{P} \), so \( \text{P} \neq \text{NP} \). \( \square \)
How do we even know NP-complete problems exist in the first place?
Satisfiability

- A propositional logic formula $\varphi$ is called **satisfiable** if there is some assignment to its variables that makes it evaluate to true.
  - $p \land q$ is satisfiable.
  - $p \land \neg p$ is unsatisfiable.
  - $p \rightarrow (q \land \neg q)$ is satisfiable.
- An assignment of true and false to the variables of $\varphi$ that makes it evaluate to true is called a **satisfying assignment**.
The boolean satisfiability problem (SAT) is the following:

Given a propositional logic formula $\varphi$, is $\varphi$ satisfiable?

Formally:

$$SAT = \{ \langle \varphi \rangle \mid \varphi \text{ is a satisfiable PL formula} \}$$
The language SAT happens to be in \textbf{NP}. Think about how a polynomial-time verifier for SAT might work. Which of the following would work as certificates for such a verifier, given that the input is a propositional formula \( \phi \)?

A. The truth table of \( \phi \).
B. One possible variable assignment to \( \phi \).
C. A list of all possible variable assignments for \( \phi \).
D. None of the above, or two or more of the above.
**Theorem (Cook-Levin)**: SAT is $\textbf{NP}$-complete.

**Proof Idea**: To see that $\textbf{SAT} \in \textbf{NP}$, show how to make a polynomial-time verifier for it. Key idea: have the certificate be a satisfying assignment.

To show that $\textbf{SAT}$ is $\textbf{NP}$-hard, given a polynomial-time verifier $V$ for an arbitrary $\textbf{NP}$ language $L$, for any string $w$ you can construct a polynomially-sized formula $\varphi(w)$ that says “there is a certificate $c$ where $V$ accepts $\langle w, c \rangle$.” This formula is satisfiable if and only if $w \in L$, so deciding whether the formula is satisfiable decides whether $w$ is in $L$.

**Proof**: Take CS154!
Why All This Matters

• Resolving $\mathbf{P} \equiv \mathbf{NP}$ is equivalent to just figuring out how hard SAT is.
  • If $\text{SAT} \in \mathbf{P}$, then $\mathbf{P} = \mathbf{NP}$.
    If $\text{SAT} \notin \mathbf{P}$, then $\mathbf{P} \neq \mathbf{NP}$.

• We've turned a huge, abstract, theoretical problem about solving problems versus checking solutions into the concrete task of seeing how hard one problem is.

• You can get a sense for how little we know about algorithms and computation given that we can't yet answer this question!
Why All This Matters

• You will almost certainly encounter \textbf{NP}-hard problems in practice – they're everywhere!

• If a problem is \textbf{NP}-hard, then there is no known algorithm for that problem that
  • is efficient on all inputs,
  • always gives back the right answer, and
  • runs deterministically.

• \textit{Useful intuition:} If you need to solve an \textbf{NP}-hard problem, you will either need to settle for an approximate answer, an answer that's likely but not necessarily right, or have to work on really small inputs.
Sample NP-Hard Problems

- **Computational biology:** Given a set of genomes, what is the most probable evolutionary tree that would give rise to those genomes? *(Maximum parsimony problem)*
- **Game theory:** Given an arbitrary perfect-information, finite, twoplayer game, who wins? *(Generalized geography problem)*
- **Operations research:** Given a set of jobs and workers who can perform those tasks in parallel, can you complete all the jobs within some time bound? *(Job scheduling problem)*
- **Machine learning:** Given a set of data, find the simplest way of modeling the statistical patterns in that data *(Bayesian network inference problem)*
- **Medicine:** Given a group of people who need kidneys and a group of kidney donors, find the maximum number of people who can end up with kidneys *(Cycle cover problem)*
- **Systems:** Given a set of processes and a number of processors, find the optimal way to assign those tasks so that they complete as soon as possible *(Processor scheduling problem)*
**Coda:** What if $P \neq NP$ is resolved?
Intermediate Problems

• With few exceptions, every problem we've discovered in \( \text{NP} \) has either
  • definitely been proven to be in \( \text{P} \), or
  • definitely been proven to be \( \text{NP} \)-complete.

• A problem that's \( \text{NP} \), not in \( \text{P} \), but not \( \text{NP} \)-complete is called \textbf{NP-intermediate}.

• \textbf{Theorem (Ladner):} There are \( \text{NP} \)-intermediate problems if and only if \( \text{P} \neq \text{NP} \).
What if $P \neq NP$?
A Good Read:

“A Personal View of Average-Case Complexity” by Russell Impagliazzo
What if $P = NP$?
And a Dismal Third Option
Next Time

- The Big Picture
- Where to Go from Here
- A Final “Your Questions”
- Parting Words!