Last time
• Polynomial-time reductions

Today
• Polynomial-time reductions
• NP-completeness
Classify Problems

- **Desiderata:** classify problems according to those that can be solved in polynomial-time and those that cannot.

- Some problems *provably require exponential time* (next month):
  - Given a Turing machine, does it halt in at most $k$ steps?
  - Given a board position in an $n$-by-$n$ generalization of chess, can black guarantee a win?

- **Frustrating news:** huge number of fundamental problems have defied classification for decades.

- **Chapters 7.4-7.5 (NP-completeness):** Show that these fundamental problems are "computationally equivalent" and appear to be different manifestations of one really hard problem.
Given languages $A$ and $B$, 
$A \leq_p B$

if there is a \textit{poly-time} computable function $f$, 
such that for all strings $w$, 
$w \in A$ iff $f(w) \in B$. 

\begin{equation}
A \xrightarrow{f} B
\end{equation}
Implication of poly-time reductions

**Theorem.** If $A \leq_p B$ and $B \in P$ then $A \in P$.

(So, if $A \leq_p B$ and $A \notin P$ then $B \notin P$.)

**Theorem.** If $A \leq_p B$ and $B \leq_p C$ then $A \leq_p C$.

(Poly-time reductions compose.)
Basic reduction strategies

- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.
Independent Set

Given an undirected graph $G$, an **independent set** in $G$ is a set of nodes, which includes at most one endpoint of every edge.

$$\text{INDEPENDENT SET} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph which has an independent set with } k \text{ nodes} \}$$

- Is there an independent set of size $\geq 6$?
  - Yes.

- Is there an independent set of size $\geq 7$?
  - No.
Vertex Cover

Given an undirected graph $G$, a vertex cover in $G$ is a set of nodes, which includes at least one endpoint of every edge.

$$\text{VERTEX COVER} = \{ \langle G, k \rangle \mid G \text{ is an undirected graph which has a vertex cover with } k \text{ nodes} \}$$

- Is there vertex cover of size $\leq 4$?
  - Yes.

- Is there a vertex cover of size $\leq 3$?
  - No.
Set Cover

Given a set U, called a universe, and a collection of its subsets \( S_1, S_2, \ldots, S_m \), a set cover of U is a subcollection of subsets whose union is U.

- \( \text{SET COVER} = \{ \langle U, S_1, S_2, \ldots, S_m; k \rangle \mid U \text{ has a set cover of size } k \} \)

U = \{ 1, 2, 3, 4, 5, 6, 7 \}  
\( k = 2 \)  
\( S_1 = \{3, 7\} \)  
\( S_2 = \{3, 4, 5, 6\} \)  
\( S_3 = \{1\} \)  
\( S_4 = \{2, 4\} \)  
\( S_5 = \{5\} \)  
\( S_6 = \{1, 2, 6, 7\} \)

- Sample application.
  - m available pieces of software.
  - Set U of n capabilities that we would like our system to have.
  - The \( i \)th piece of software provides the set \( S_i \subseteq U \) of capabilities.
  - Goal: achieve all n capabilities using fewest pieces of software.
Satisfiability

- **Boolean variables**: variables that can take on values T/F (or 1/0)
- **Boolean operations**: \( \lor, \land, \text{ and } \lnot \)
- **Boolean formula**: expression with Boolean variables and ops

\[
\text{SAT} = \{ \langle \Phi \rangle \mid \Phi \text{ is a satisfiable Boolean formula} \}
\]

- **Literal**: A Boolean variable or its negation. \( x_i \text{ or } \overline{x_i} \)
- **Clause**: OR of literals. \( C_j = x_1 \lor x_2 \lor x_3 \)
- **Conjunctive normal form (CNF)**: AND of clauses. \( \Phi = C_1 \land C_2 \land C_3 \land C_4 \)

3SAT = \{ \langle \Phi \rangle \mid \Phi \text{ is a satisfiable Boolean CNF formula, where each clause contains exactly 3 literals} \}

Each corresponds to a different variable

Ex: \[
\left( \overline{x_1} \lor x_2 \lor x_3 \right) \land \left( x_1 \lor \overline{x_2} \lor x_3 \right) \land \left( x_2 \lor x_3 \right) \land \left( \overline{x_1} \lor \overline{x_2} \lor \overline{x_3} \right)
\]

Yes: \( x_1 = \text{true}, x_2 = \text{true} \text{ and } x_3 = \text{false} \).
Summary of reduction strategies

• Basic reduction strategies.
  – Simple equivalence: \( \text{INDEPENDENT-SET} \equiv_p \text{VERTEX-COVER} \).
  – Special case to general case: \( \text{VERTEX-COVER} \leq_p \text{SET-COVER} \).
  – Encoding with gadgets: \( 3\text{-SAT} \leq_p \text{INDEPENDENT-SET} \).

• Composition. If \( X \leq_p Y \) and \( Y \leq_p Z \), then \( X \leq_p Z \).
• Thus,

\[ 3\text{SAT} \leq_p \text{INDEPENDENT-SET} \leq_p \text{VERTEX-COVER} \leq_p \text{SET-COVER}. \]

Are we done now?
A language $B$ is **NP-complete** if

1. $B \in \text{NP}$
2. $B$ is **NP-hard**, i.e., every language in NP is poly-time reducible to $B$. 

**Diagram:**

- $P$ is a subset of $NP$.
- $B$ is connected to all elements in $NP$.

**Legend:**
- $P$ represents **P**.
- $NP$ represents **NP**.
- $B$ represents **NP-complete**.
Implication of poly-time reductions

**Theorem.** If

- B is $\text{NP}$-complete,
- C $\in \text{NP}$ and
- $B \leq_p C$

then C is $\text{NP}$-complete.
Implication of poly-time reductions

**Theorem.** If

- $B$ is $NP$-complete,
- $C \in NP$ and
- $B \leq_P C$

then $C$ is $NP$-complete.

**Theorem.** If $B$ is $NP$-complete and $B \in P$ then

$P = NP$.

(So, if $B$ is $NP$-complete and $P \neq NP$
then there is no poly-time algorithm for $B$.)
$BA_{\text{NTM}} = \{ \langle M, x, t \rangle \mid M \text{ is an NTM that accepts } x \text{ in at most } t \text{ steps} \}$

**Theorem.** $BA_{\text{NTM}}$ is NP-Complete.

1. $BA_{\text{NTM}} \in \text{NP}$:
   
   The list of guesses $M$ makes to accept $x$ in $t$ steps is the certificate that $\langle M, x, t \rangle \in BA_{\text{NTM}}$.

2. For all $A \in \text{NP}$, $A \leq_{\text{p}} BA_{\text{NTM}}$.

   $A \in \text{NP}$ iff there is an NTM $N$ for $A$ that runs in time $O(n^k)$.

   Let $f_A(w) = \langle N, w, c \mid w \mid^k \rangle$.

   $\langle N, w, c \mid w \mid^k \rangle \in BA_{\text{NTM}}$ iff $N$ accepts $w$, iff $w \in A$.

Technical detail: $n$ denotes $1^n$. 

4/18/2016
A **circuit** is built out of AND, OR and NOT gates. A circuit is **satisfiable** if one can set the circuit inputs, so that the output is 1. \[ \text{CIRCUIT-SAT = \{\langle C \rangle | C \text{ is a satisfiable circuit}\}}. \]
The "First" NP-Complete Problem

**Theorem.** CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

**Proof sketch.**

1. **CIRCUIT-SAT is in NP**
   
   certificate: input on which circuit is 1.
The "First" NP-Complete Problem

**Theorem.** CIRCUIT-SAT is NP-complete. [Cook 1971, Levin 1973]

**Proof sketch.**

2. For all $A \in \text{NP}$, $A \leq_p \text{CIRCUIT-SAT}$.
   - A TM that takes a fixed number of bits as input can be represented by a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.
   - Consider some problem $A \in \text{NP}$. It has a poly-time verifier $V(w, c)$. To determine whether $w \in A$, need to know if there exists a certificate $c$ of length $p(|w|)$ such that $V(w, c)$ accepts.
   - View $V(w, c)$ as an algorithm on $|w| + p(|c|)$ bits (input $w$, certificate $c$) and convert it into a poly-size circuit $C$.
     - first $|w|$ bits are hard-coded with $w$
     - remaining $p(|s|)$ bits represent bits of $t$

**Correctness:** Circuit $C$ is satisfiable iff $V(w, c)$ accepts.
Idea #1 (on board): Algorithms are circuits

• Every circuit is an “algorithm” with
  – Fixed length input and
  – Fixed running time

• Every algorithm with fixed length input $m$, and upper bound $t$ on running time can be converted into a circuit with $O(t(m + t))$ gates
  – Space usage is at most $m + t$
  – Imagine a $(m + t) \times t$ table where column $i$ is the state of the algorithms memory at time $i$
  – Circuit just needs to compute state at time $i + 1$ from state at time $i$. 
A circuit $C$ whose inputs can be set so that $C$ outputs true iff graph $G$ has an independent set of size 2.

$G = (V, E), \ n = 3$

\[
\binom{n}{2} \text{ hard-coded inputs (graph description)} \quad n \text{ inputs (nodes in independent set)}
\]
3SAT is NP-complete

Proof. Suffices to show that CIRCUIT-SAT \leq_p 3SAT since 3SAT is in NP.

- Let C be any circuit.
- Create a 3-SAT variable \( x_i \) for each circuit element \( i \).
- Make circuit compute correct values at each node:

  - \( x_2 = \neg x_3 \implies \) add 2 clauses: \( x_2 \lor x_3, \neg x_2 \lor x_3 \)
  
  - \( x_1 = x_4 \lor x_5 \implies \) add 3 clauses: \( x_1 \lor \neg x_4, x_1 \lor \neg x_5, x_1 \lor x_4 \lor x_5 \)
  
  - \( x_0 = x_1 \land x_2 \implies \) add 3 clauses: \( \neg x_0 \lor x_1, \neg x_0 \lor x_2, x_0 \lor \neg x_1 \lor \neg x_2 \)

- Hard-coded input values and output value.

  - \( x_5 = 0 \implies \) add 1 clause: \( \neg x_5 \)
  
  - \( x_0 = 1 \implies \) add 1 clause: \( x_0 \)

- Final step: turn clauses of length < 3 into clauses of length exactly 3.
Idea #2 (on board): Verifying is easier than computing

- Given a circuit $C$ with $n$ input wires and $m$ gates, the reduction constructs a 3CNF formula $\Phi$ with $O(n + m)$ variables and $O(m)$ clauses
  - One variable $\Phi$ for each wire in $C$
  - Given a satisfying assignment $a$ for $C$, get a satisfying assignment to $\Phi$:
    - Set variable $x_e$ to the value on wire $e$ when $C$ is executed on $a$
- Notice the power of nondeterminism
  - Checking that each gate is executed correctly only requires looking at 3 wires at a time