Admin: (No recitations tomorrow—Thanksgiving!)

Reading: CLRS §34.3, pages 984-985

Outline: Special lecture on reductions

☐ intro
☐ VC ≤p clique
☐ MSSP ≤p SSSP
☐ SAT ≤p VC
**Reductions**

"Problem" maps inputs to outputs (e.g. function or predicate)

E.g. product of two matrices ← function
is graph connected? ← predicate (T/F)

is formula satisfiable?

We are interested in relation between problems.

In particular, can we solve problem A using procedure for problem B?
I.e. can we reduce A to B?

**Notation:** $A \leq B$ for such a reduction

Several different flavors of such reduction...

Suppose $A$ maps $x$ to $A(x)$

$B$ maps $y$ to $B(y)$

**Figure:**

```
\begin{tikzpicture}
  \node (A) at (0,0) {A};
  \node (B) at (2,-2) {B};
  \node (A_x) at (0,-2) {A(x)};
  \node (y) at (2,-3) {y};

  \draw[->] (A) -- (B);
  \draw[->] (A_x) -- (B);
  \draw[->] (B) -- (A_x);
  \draw[->] (A_x) -- (y);
\end{tikzpicture}
```
Considerations:

- Are A & B functions or predicates?
  (works either way, just need to be clear)

- Is there a bound on reduction work (exclusive of calls to B?)
  Typically want this to be "small"

- Is A=B? ("self-reduction" to smaller problem instances, e.g. mergesort or dyn. programming or gcd)
  We'll ignore self-reduction today...

- Is B used once, or multiple times, for a single input to A?

- Is reduction a "mapping reduction"? That is, reduction has form
  \[ A(x) = B(f(x)) \]
  We map x to f(x), then apply B once, and take answer from B as final output for A(x).
  Many common reductions are mapping reductions.

- Notation: \( A \leq_m B \) mapping reduction

  \( A \leq_p B \) polynomial time mapping reduction (aka "Karp reduction")

  \( A \leq_T B \) "Turing reduction" (can call B many times)

  \( A \leq_{Cook} B \) "Cook reduction": polynomial time Turing reduction (can call B many times)
Motivations for studying reductions

1. **Better upper bounds**
   If we show $A \leq B$ via an efficient reduction, and we have an efficient algorithm for $B$, then we have an efficient algorithm for $A$.

2. **Lower bounds**
   If we show $A \leq B$ via an efficient reduction, and we have reason to believe $A$ is hard, then we have reason to believe $B$ is hard.

(Used in computational complexity theory and theory of NP-completeness.)
Example: \( A = \) does input graph \( G \) of \( n \) vertices contain a vertex cover of size \( k \)?

\( B = \) does input graph \( G \) of \( n \) vertices contain a clique of size \( l \)? (clique = \( l \) mutually adjacent vertices)

\[ \Delta \quad \Box \quad \ldots \]

\( l = 3 \quad l = 4 \)

Claim: \( A \leq_p B \) (poly-time mapping reduction)

Proof: Given \( G = (V, E) \) and \( k \), map to \( G' = (V, \overline{E}) \) and \( l = n-k \)

where \( \overline{E} \) is complement of \( E \).

(Can reverse to show \( B \leq_p A \) as well.)
**Fact:** Vertex cover is “NP-hard”  

**Fact:** If A is NP-hard \& A \leq_p B, then B is NP-hard.

\[ \therefore \text{Clique problem is NP-hard (actually, NP-complete)} \]

(NP-hard \equiv at least as hard as satisfiability of Boolean formula, SAT \equiv I, e. SAT \leq_p A if A NP-hard.)

(NP-complete \equiv NP-hard \& solvable in nondet polynomial time...)

**Fact:** If A \leq_p B \& B \leq_p C, then A \leq_p C.

**Example:**

\[ A = \text{mult-source shortest path} : \text{given } G=(V,E) \text{ with edge weights } w \& \text{ start vertices } s_1, s_2, ..., s_k \& \text{ target vertex } t, \text{ find shortest path from } \text{ some start to target } t. \]

\[ B = \text{std SSSP}. \]

\[ A \leq_p B \]
Proof: 

\[ G' \]

\[ \text{shortest path in } G' \text{ from } s \rightarrow t \]

\[ \text{yields shortest } s_i \rightarrow t \text{ path in } G. \]

Thus, good alg for SSSP yields good alg for M SSP (since reduction is efficient).

**Emphasize:** Reduction by itself \( A \leq_p B \) doesn't solve \( A \) or \( B \), it just relates \( A \) to \( B \).

You have to know that \( A \) is hard, or that \( B \) is easy, for this to be meaningful.
Thm \[ \text{SAT} \leq^p \text{VC (vertex cover)} \]

Given \( \Psi = (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor \overline{x_3}) \land (x_1 \lor x_2 \lor x_3) \),

\[ \downarrow \text{transform to } G \]

\[ \overline{x_1}, x_1, x_2, x_3 \]

\[ x_1, x_2, x_3, x_3 \]

\[ \text{draw edge if incompatible or in same clause} \]

Look for V.C. of size \( L - C \)

\[ \Rightarrow \text{C vertices with no edges} \]

\[ \Rightarrow \text{must be one from each clause} \]

\[ \Rightarrow \text{satisfying assignment} \]

\[ \text{(This suffices to show VC is NP-hard...)} \]