Announcements:  
- Project will be released this week. Start figuring out project teams. (up to 3 per team)
- New section: Mondays 2PM  
  Annamira & Adnana.  

See Piazza for more details.
Recap: Reductions

"a problem \( A \) reduces to a problem \( B \) if any subroutine to solve \( B \) can be used to solve \( A \)"

In more detail:

\[ \begin{align*}
&\text{pre-processing} \\
&\text{efficient} \\
&x \rightarrow \rightarrow y \\
&\text{efficient} \\
\end{align*} \]

\[ \begin{align*}
B(y) &\rightarrow A(x) \\
\text{post-processing} \\
\text{efficient} \\
\end{align*} \]

\( \circ \) an efficient alg for \( B \) \( \Rightarrow \) an efficient alg for \( A \).

A reduction = \( \text{pre-processing} + \text{post-processing} \).

\( \circ \) \( \text{if an efficient alg for } B \leq_\mathcal{A} \text{ an efficient alg for } A \)
Recap: Rudrata Cycle & Rudrata Path

**Rudrata Cycle Problem:**
Given an undirected graph $G = (V, E)$
Is there a cycle that visits each vertex exactly once?

**a.k.a. Hamiltonian Cycle**

**Rudrata st-path problem:**
Given an undirected graph $G = (V, E)$, nodes s & t
Is there a path from s to t that visits each vertex exactly once?

---

**RudPath ≤ RudCycle**

To prove correctness:

- If $G$ has an $st$-Rudrata Path $\Rightarrow G'$ has a Rudrata Cycle
- If $G$ has no $st$-Rudrata Path $\Rightarrow G'$ has no Rudrata Cycle
Exercise from last time

**RudCycle ≤ RudPath?**

Given $G$

Pick some vertex $v = G$ duplicate it $v', v''$

and both $v', v''$ have same neighbors as $v$

Claim: $G$ has RudCycle $\iff G'$ has a RudPath from $v'$ to $v''$.
Search, Decision & Optimization Problems

So far we talked mainly about optimization problems.

For Example: 1. Find shortest path from s to t.
               2. Find Best prefix-free Encoding.
               3. Find Maximum Flow.

Search Problems:

Examples  1. Given $G, s, t$ & Budget $B$, find a path of length $\leq B$ from $s$ to $t$ (if one exists)
            2. Given $f_1, \ldots, f_m$ & A Budget $B$, find a prefix-free tree of cost $\leq B$ (if one exists)
            3. Given $G$ find a Rudrada Cycle (if one exists)

Decision Problems: Given $G, s, t$ & Budget $B$, is there a path of length $\leq B$ from $s$ to $t$?
**Search Problems**

Def'n: A language \( L \) is a subset \( \{0,1\}^* \).

Def'n: A binary relation \( R \) is a subset \( R \subseteq \{0,1\}^* \times \{0,1\}^* \).

We say that a binary relation is "efficiently verifiable" if given \((x,y)\) there exists an efficient algorithm that decides whether \((x,y) \in R\).

- \((x,y) \in R \Rightarrow |y| \leq \text{poly}(|x|) \) (witnesses are \( \text{poly}(|x|) \) -length).
- Runtime of algorithm (verifier): at most \( \text{poly}(|x|) \).

**Example:** \[ R = \{ (G, c) \mid G \text{ is an undirected graph, } c \text{ is a cycle in } G \text{ that visits every vertex once} \} \]

Def'n: \( L(R) = \{ x \mid \exists y, (x,y) \in R \} \) Language assoc. with \( R \).

Decide \((R) = \text{Given } x \text{ decide whether } \exists y : (x,y) \in R \).

Search \((R) = \text{Given } x \text{ find } y : (x,y) \in R \text{ if one exists.} \)
**Observe:** If $R$ is "efficiently verifiable" then $\text{Decide}(R)$ can be solved $2^{\text{poly}(|x|)}$ time.

**Proof:** Given $x$:

1. For every possible $y \in \{0, 1\}^{\text{poly}(|x|)}$:
   - check whether $(x, y) \in R$. (poly-time).
     - accept \hspace{1cm} continue

2. Reject.

$2^{\text{poly}(|x|)} \cdot \text{poly}(|x|)$ time.
$\mathsf{P} = \{ L(R) \text{ s.t. } \text{Decide}(R) \text{ can be solved efficiently} \}$

$\mathsf{NP} = \{ L(R) \text{ s.t. } R \text{ is efficiently verifiable} \}$

$\mathsf{P} \subset \mathsf{NP}$

$\mathsf{NP}$ is non-deterministic.

$R = \{ ((G,s,t,B),P) : P \text{ is a path from } s \text{ to } t \text{ with length } \leq B \}$

Decide$(R)$ efficiently? Yes, Dijkstra $L(R) \in \mathsf{P}$.

$R = \{ (G,C) : C \text{ is a Hamiltonian cycle in } G \}$

$L(R) \in \mathsf{NP}$

$L(R) \in \mathsf{P}$.

$L(R) \in \mathsf{NP}$

$L(R) \in \mathsf{P}$. $L(R) \in \mathsf{P}$.
Does \( P = NP \)?
**NP Completeness**

**Def:** A problem \( A \) is **NP-Hard** if \( \forall B \in \text{NP} \) \( B \rightarrow A \) (\( B \leq_A A \)).

**Def:** A problem \( A \) is **NP-complete** if \( A \) is NP-hard and \( A \in \text{NP} \).

There exist NP-complete problems!

\[
\begin{align*}
B &\rightarrow A \text{ is NPC} \quad \Rightarrow \quad C \text{ is NPC.} \\
A &\rightarrow C \\
B &\rightarrow A \rightarrow C
\end{align*}
\]

\( B \rightarrow A \rightarrow C \)

- Pre-process
- Post-process

\( C \text{ is in NP} \)
\[ B \leq A \leq C \]
\[ \Rightarrow B \leq C. \]
Def'n: A circuit is a directed acyclic graph with input nodes marked by $x_1, \ldots, x_n$ & gates: OR gate, AND gate, Not gate.

Example:

\[ \begin{array}{ccc}
\neg & & \\
\lor & & \\
\lor & & \\
\end{array} \]

1 output.

Def'n: CSAT (Circuit Satisfiability). Given circuit $C$ on inputs $y_1, \ldots, y_m$ decide whether $\exists y \in \{0,1\}^m$ s.t. $C(y) = 1$. 

size = # of gates
"Claim": suppose algorithm $A$ runs on inputs of length $n$ in time $t$. Then, there exists a circuit of size $O(t^2 \cdot n)$ that "simulates" $A$.

**Idea:**

$\text{layer } t$

$\text{layer } t-1$

$\vdots$

$\text{layer } 1$

$\text{layer } 0$:

input & initial state of $A$

State of memory & registers after $i$ steps
\[ R = \{ (c, y) : c(y) = 1 \} \]

\[ L(R) = \text{CSAT}. \]

\[ \text{NPC} = \text{NP}^H + \text{NP} \]