Recap

- Polynomial Time Reductions ($X \leq P Y$)
  - Key Problems
    - Independent Set, Vertex Cover, Set Cover, 3-SAT etc...
  - Example Reductions
    - Independent Set \leq P Vertex Cover (Simple Equivalence)
    - Vertex Cover \leq P Independent Set (Simple Equivalence)
    - Independent Set \leq P Set Cover (Special Case to General)
    - 3-SAT \leq P Independent Set (Gadgets)
  - Decision Problems vs Search Problems
  - Self-Reducibility

Decision Problems

- Decision problem.
  - $X$ is a set of strings.
  - Instance: string $s$.
  - Algorithm $A$ solves problem $X$: $A(s) = yes$ if $s \in X$.

Polynomial time. Algorithm $A$ runs in poly-time if for every string $s$, $A(s)$ terminates in at most $p(|s|)$ “steps”, where $p(\cdot)$ is some polynomial.

- PRIMES: $X = \{2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, \ldots\}$


Definition of $P$

- Decision problems for which there is a poly-time algorithm.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Algorithm</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>MULTIPLE</td>
<td>Is $x$ a multiple of $y$?</td>
<td>Grade school division</td>
<td>51, 17</td>
<td>51, 16</td>
</tr>
<tr>
<td>RELPRIME</td>
<td>Are $x$ and $y$ relatively prime?</td>
<td>Euclid (300 BCE)</td>
<td>34, 35</td>
<td>26, 51</td>
</tr>
<tr>
<td>PRIMES</td>
<td>Is $x$ prime?</td>
<td>AKS (2002)</td>
<td>53</td>
<td>51</td>
</tr>
<tr>
<td>EDITDISTANCE</td>
<td>Is the edit distance between $x$ and $y$ less than 5?</td>
<td>Dynamic programming</td>
<td>neither</td>
<td>accept</td>
</tr>
<tr>
<td>LSEQOLVE</td>
<td>Is there a vector $x$ that satisfies $Ax = b$?</td>
<td>Gauss-Edmonds elimination</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
NP

Certification algorithm intuition.
- Certifier views things from "managerial" viewpoint.
- Certifier doesn't determine whether \( s \in X \) on its own; rather, it checks a proposed proof \( t \) that \( s \in X \).

**Def.** Algorithm \( C(s, t) \) is a certifier for problem \( X \) if for every string \( s \), \( s \in X \) iff there exists a string \( t \) such that \( C(s, t) = \text{yes} \).

**NP.** Decision problems for which there exists a poly-time certifier.

**Remark.** NP stands for nondeterministic polynomial-time.

\[ C(s, t) \] is a poly-time algorithm and \( |t| \leq p(|s|) \) for some polynomial \( p(\cdot) \).

"Certificate" or "witness"

Certifiers and Certificates: Composite

**COMPOSITES.** Given an integer \( s \), is \( s \) composite?

**Certificate.** A nontrivial factor \( t \) of \( s \). Note that such a certificate exists iff \( s \) is composite. Moreover \( |t| \leq |s| \).

**Certifier.**

```java
boolean C(s, t) {
    if (t < 1 or t > s)
        return false
    else if (s is a multiple of t)
        return true
    else
        return false
}
```

**Instance.** \( s = 437,669 \).
**Certificate.** \( t = 541 \) or \( 809 \).

**Conclusion.** COMPOSITES is in NP.

Certifiers and Certificates: 3-Satisfiability

**SAT.** Given a CNF formula \( \Phi \), is there a satisfying assignment?

**Certificate.** An assignment of truth values to the \( n \) boolean variables.

**Certifier.** Check that each clause in \( \Phi \) has at least one true literal.

**Ex.**

\[
(\neg x_1 \vee x_2 \vee x_3) \land (x_1 \vee \neg x_2 \vee x_3) \land (x_1 \vee x_2 \vee x_3) \land (\neg x_1 \vee \neg x_2 \vee \neg x_3) 
\]

\[ x_1 = 1, \ x_2 = 0, \ x_3 = 0 \]

**Certificate.**

**Conclusion.** SAT is in NP.

The Main Question: P Versus NP

**Does P = NP?** [Cook 1971, Edmonds, Levin, Yablonski, Gödel]
- Is the decision problem as easy as the certification problem?
- Clay $1 million prize.

If yes: Efficient algorithms for 3-COLOR, TSP, FACTOR, SAT, ...

If no: No efficient algorithms possible for 3-COLOR, TSP, SAT, ...

Consensus opinion on \( P = NP \)? Probably no.

Certifiers and Certificates: Hamiltonian Cycle

**HAM-CYCLE.** Given an undirected graph \( G = (V, E) \), does there exist a simple cycle \( C \) that visits every node?

**Certificate.** A permutation of the \( n \) nodes.

**Certifier.** Check that the permutation contains each node in \( V \) exactly once, and that there is an edge between each pair of adjacent nodes in the permutation.

**Conclusion.** HAM-CYCLE is in NP.
8.4 NP-Completeness

Polynomial Transformation

Def. Problem X polynomial reduces (Cook) to problem Y if arbitrary instances of problem X can be solved using:
- Polynomial number of standard computational steps, plus
- Polynomial number of calls to oracle that solves problem Y.

Def. Problem X polynomial transforms (Karp) to problem Y if given any input x to X, we can construct an input y such that x is a yes instance of X iff y is a yes instance of Y.

Note. Polynomial transformation is polynomial reduction with just one call to oracle for Y, exactly at the end of the algorithm for X. Almost all previous reductions were of this form.

Open question. Are these two concepts the same with respect to NP?

NP-Complete

NP-complete. A problem Y in NP with the property that for every problem X in NP, X ≤p Y.

Theorem. Suppose Y is an NP-complete problem. Then Y is solvable in poly-time iff P = NP.

Pf. \[ \text{If } P = NP \text{ then } Y \text{ can be solved in poly-time since } Y \text{ is in NP.} \]
\[ \text{If } Y \text{ can be solved in poly-time, let } X \text{ be any problem in NP.} \]
\[ \text{Since } X \leq_p Y, \text{ we can solve } X \text{ in poly-time. This implies } NP \subseteq P. \]

Fundamental question. Do there exist "natural" NP-complete problems?
### The "First" NP-Complete Problem

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

**Pf.** (sketch)
- Any algorithm that takes a fixed number of bits $n$ as input and produces a yes/no answer can be represented by such a circuit. Moreover, if algorithm takes poly-time, then circuit is of poly-size.
- Consider some problem $X$ in NP. It has a poly-time certifier $C(s, t)$.
  - To determine whether $s$ is in $X$, need to know if there exists a certificate $t$ of length $p(|s|)$ such that $C(s, t) = \text{yes}$.
  - View $C(s, t)$ as an algorithm on $|s| + p(|s|)$ bits (input $s$, certificate $t$)
    - first $|s|$ bits are hard-coded with $s$
    - remaining $p(|s|)$ bits represent bits of $t$
  - Circuit $K$ is satisfiable iff $C(s, t) = \text{yes}$.

### 3-SAT is NP-Complete

**Theorem.** 3-SAT is NP-complete.

**Pf.** Suffices to show that CIRCUIT-SAT $\leq_p$ 3-SAT since 3-SAT is in NP.
- Let $K$ 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: $x_2 \lor \neg x_3$
    - $x_1 = x_4 \lor x_5 \implies$ add 3 clauses: $x_1 \lor x_4 \lor x_5$
    - $x_0 = x_1 \land x_2 \implies$ add 3 clauses: $x_0 \land x_1 \land x_2$
  - Hard-coded input values and output value:
    - $x_0 = 0 \implies$ add 1 clause: $\neg x_0$
    - $x_0 = 1 \implies$ add 1 clause: $x_0$
  - Final step: turn clauses of length $< 3$ into clauses of length exactly 3.

### Circuit Satisfiability

**CIRCUIT-SAT.** Given a combinational circuit built out of AND, OR, and NOT gates, is there a way to set the circuit inputs so that the output is 1?

### Establishing NP-Completeness

**Remark.** Once we establish first "natural" NP-complete problem, others fall like dominoes.

**Recipe to establish NP-completeness of problem $Y$.**
- Step 1. Show that $Y$ is in NP.
- Step 2. Choose an NP-complete problem $X$.
- Step 3. Prove that $X \leq_p Y$.

**Justification.** If $X$ is an NP-complete problem, and $Y$ is a problem in NP with the property that $X \leq_p Y$ then $Y$ is NP-complete.

**Pf.** Let $W$ be any problem in NP. Then $W \leq_p X \leq_p Y$.
- By transitivity, $W \leq_p Y$.
- Hence $Y$ is NP-complete.

---

**Example.** Construction below creates a circuit $K$ whose inputs can be set so that $K$ outputs true iff graph $G$ has an independent set of size 2.

**Observation.** All problems below are NP-complete and polynomial reduce to one another!
Some NP-Complete Problems

Six basic genres of NP-complete problems and paradigmatic examples.

- Packing problems: SET-PACKING, INDEPENDENT SET.
- Covering problems: SET-COVER, VERTEX-COVER.
- Constraint satisfaction problems: SAT, 3-SAT.
- Sequencing problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3D-MATCHING, 3-COLOR.
- Numerical problems: SUBSET-SUM, KNAPSACK.

Practice: Most NP problems are either known to be in P or NP-complete.

Notable exceptions: Factoring, graph isomorphism, Nash equilibrium.

8.9 co-NP and the Asymmetry of NP

Asymmetry of NP
We only need to have short proofs of yes instances.

Ex 1. SAT vs. TAUTOLOGY.
- Can prove a CNF formula is satisfiable by giving such an assignment.
- How could we prove that a formula is not satisfiable?

Ex 2. HAM-CYCLE vs. NO-HAM-CYCLE.
- Can prove a graph is Hamiltonian by giving such a Hamiltonian cycle.
- How could we prove that a graph is not Hamiltonian?

Remark: SAT is NP-complete and SAT \equiv P TAUTOLOGY, but how do we classify TAUTOLOGY? not even known to be in NP

NP and co-NP

NP: Decision problems for which there is a poly-time certifier.
Ex. SAT, HAM-CYCLE, COMPOSITES.

Def. Given a decision problem X, its complement \overline{X} is The same problem with the yes and no answers reverse.

Ex. X = \{ 0, 1, 4, 6, 8, 9, 10, 12, 14, 15, \ldots \}
\overline{X} = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots \}

co-NP: Complements of decision problems in NP.
Ex. TAUTOLOGY, NO-HAM-CYCLE, PRIMES.
**Fundamental question.** Does NP = co-NP?
- Do yes instances have succinct certificates iff no instances do?
- Consensus opinion: no.

**Theorem.** If NP = co-NP, then P = NP.
- P is closed under complementation.
- If P = NP, then NP is closed under complementation.
- In other words, NP = co-NP.
- This is the contrapositive of the theorem.

**Good Characterizations**

**Good characterization.** ([Edmonds 1965])\( NP \cap \text{co-NP} \).
- If problem X is in both NP and co-NP, then:
  - for yes instance, there is a succinct certificate
  - for no instance, there is a succinct disqualifier
- Provides conceptual leverage for reasoning about a problem.

**Ex.** Given a bipartite graph, is there a perfect matching.
- If yes, can exhibit a perfect matching.
- If no, can exhibit a set of nodes S such that \(|N(S)| < |S|\).

**Observation.** \( P \subseteq NP \cap \text{co-NP} \).
- Proof of max-flow min-cut theorem led to stronger result that max-flow and min-cut are in P.
- Sometimes finding a good characterization seems easier than finding an efficient algorithm.

**Fundamental open question.** Does \( P = NP \cap \text{co-NP} \)?
- Mixed opinions.
- Many examples where problem found to have a non-trivial good characterization, but only years later discovered to be in P.
  - linear programming [Khachiyan, 1979]
  - primality testing [Agrawal-Kayal-Saxena, 2002]

**Fact.** Factoring is in \( NP \cap \text{co-NP} \), but not known to be in P.
- If poly-time algorithm for factoring, our break RSA crypotosystem.

**PRIMES is in NP \cap co-NP**

**Theorem.** PRIMES is in \( NP \cap \text{co-NP} \).
**Pf.** We already know that PRIMES is in co-NP, so it suffices to prove that PRIMES is in NP.

**Pratt’s Theorem.** An odd integer \( s \) is prime iff there exists an integer \( t \) such that
\[
\begin{align*}
\frac{s-1}{p} & \equiv 1 \pmod{s} \\
\frac{s-1}{3} & \equiv 1 \pmod{s} \\
\frac{s-1}{36,473} & \equiv 1 \pmod{s} \\
\end{align*}
\]
for all prime divisors \( p \) of \( s-1 \).

**Certifier.**
- Check \( s-1 = 2 \times 2 \times 3 \times 36,473 \).
- Check \( 17(s-1) = 36,473 \).
- Check \( 17(s-1)/2 = 329,415 \).

**Input.** \( s = 437,677 \)

**FACTOR is in NP \cap co-NP**

**Theorem.** FACTOR is in \( NP \cap \text{co-NP} \).
**Pf.**
- Certificate: a factor \( p \) of \( x \) that is less than \( y \).
- Disqualifier: the prime factorization of \( x \) (where each prime factor is less than \( y \)), along with a certificate that each factor is prime.

**Primality Testing and Factoring**

**We established:** PRIMES \( \subseteq \) COMPOSITES \( \subseteq \) FACTOR.

**Natural question.** Does FACTOR \( \subseteq \) PRIMES?

**Consensus opinion.** No.

**State-of-the-art.**
- PRIMES is in P, ... proved in 2001.
- FACTOR not believed to be in P.

**RSA cryptosystem.**
- Based on dichotomy between complexity of two problems.
- To use RSA, must generate large primes efficiently.
- To break RSA, suffixes to find efficient factoring algorithm.