Algorithm Design Patterns and Anti-Patterns

Algorithm design patterns:
- Greedy.
- Divide-and-conquer.
- Dynamic programming.
- Duality.
- Reductions.
- Local search.
- Randomization.

Ex.
- O(n log n) interval scheduling.
- O(n log n) quicksort.
- O(n^2) edit distance.
- O(n^2) bipartite matching.

Algorithm design anti-patterns:
- NP-completeness.
- PSPACE-completeness.
- Undecidability.

- O(n^k) algorithm unlikely.
- O(n^k) certification algorithm unlikely.
- No algorithm possible.

8.1 Polynomial-Time Reductions
Classify Problems According to Computational Requirements

Q. Which problems will we be able to solve in practice?


Those with polynomial-time algorithms.

<table>
<thead>
<tr>
<th>Yes</th>
<th>Probably no</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shortest path</td>
<td>Longest path</td>
</tr>
<tr>
<td>Matching</td>
<td>3D-matching</td>
</tr>
<tr>
<td>Min cut</td>
<td>Max cut</td>
</tr>
<tr>
<td>2-SAT</td>
<td>3-SAT</td>
</tr>
<tr>
<td>Planar 4-color</td>
<td>Planar 3-color</td>
</tr>
<tr>
<td>Bipartite vertex cover</td>
<td>Vertex cover</td>
</tr>
<tr>
<td>Primality testing</td>
<td>Factoring</td>
</tr>
</tbody>
</table>

Polynomial-Time Reduction

Suppose we could solve X in polynomial-time. What else could we solve in polynomial time?

Reduction. Problem X polynomial reduces 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.

Notation. \( X \leq_P Y \).

That is, if Y’s oracle is B, we may design a polynomial time algorithm A which uses B to solve X (the time spent by B is not counted).

Remarks.

- Note: This is Cook reducibility.
- Karp reduction only applies to Decision Problems X and Y: 
  \( X \leq_K Y \) if there is an algorithm A for X such that A(x) is true if and only if B(f(x)) is true, where both A and f run in poly-time.

Purpose. Classify problems according to relative difficulty.

Design algorithms. If \( X \leq_P Y \) and Y can be solved in polynomial-time, then X can also be solved in polynomial time.

Establish intractability. If \( X \leq_P Y \) and X cannot be solved in polynomial-time, then Y cannot be solved in polynomial-time.

Establish equivalence. If \( X \leq_P Y \) and \( Y \leq_P X \), we use notation \( X \equiv_P Y \).
Reduction By Simple Equivalence

Basic reduction strategies:
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.

Independent Set

INDEPENDENT SET: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \geq k$, and for each edge at most one of its endpoints is in $S$?

Ex. Is there an independent set of size $\geq 6$? Yes.
Ex. Is there an independent set of size $\geq 7$? No.

Vertex Cover

VERTEX COVER: Given a graph $G = (V, E)$ and an integer $k$, is there a subset of vertices $S \subseteq V$ such that $|S| \leq k$, and for each edge, at least one of its endpoints is in $S$?

Ex. Is there a vertex cover of size $\leq 4$? Yes.
Ex. Is there a vertex cover of size $\leq 3$? No.
Claim. \textsc{Vertex-Cover} \iff \textsc{Independent-Set}.

\textbf{Pf.} We show $S$ is an independent set iff $V - S$ is a vertex cover.

$\implies$

- Let $S$ be any independent set.
- Consider an arbitrary edge $(u, v)$.
- $S$ independent $\Rightarrow u \notin S$ or $v \notin S$ $\Rightarrow u \in V - S$ or $v \in V - S$.
- Thus, $V - S$ covers $(u, v)$.

$\impliedby$

- Let $V - S$ be any vertex cover.
- Consider two nodes $u \in S$ and $v \in S$.
- Observe that $(u, v) \in E$ since $V - S$ is a vertex cover.
- Thus, no two nodes in $S$ are joined by an edge $\Rightarrow$ $S$ independent set.

---

**Reduction from Special Case to General Case**

Basic reduction strategies:
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.
**Set Cover**

*SET COVER:* Given a set $U$ of elements, a collection $S_1, S_2, \ldots, S_m$ of subsets of $U$, and an integer $k$, does there exist a collection of $\leq k$ of these sets whose union is equal to $U$?

**Sample application.**
- $m$ available pieces of software.
- $S_i$ of $n$ capabilities that we would like our system to have.
- $S_i$ piece of software provides the set $S_i \subseteq U$ of capabilities.
- Goal: achieve all $n$ capabilities using fewest pieces of software.

Ex:

<table>
<thead>
<tr>
<th>$U = {1, 2, 3, 4, 5, 6, 7}$</th>
<th>$k = 2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1 = {3, 7}$</td>
<td>$S_2 = {2, 4}$</td>
</tr>
<tr>
<td>$S_3 = {3, 4, 5, 6}$</td>
<td>$S_4 = {5}$</td>
</tr>
<tr>
<td>$S_5 = {1}$</td>
<td>$S_6 = {1, 2, 6, 7}$</td>
</tr>
</tbody>
</table>

**Vertex Cover Reduces to Set Cover**

*Claim.* $\text{VERTEX-COVER} \leq_p \text{SET-COVER}$.

*Proof.* Given a $\text{VERTEX-COVER}$ instance $G = (V, E)$, $k$, we construct a set cover instance whose size equals the size of the vertex cover instance.

**Construction.**
- Create $\text{SET-COVER}$ instance:
  - $k = k$, $U = E$, $S_v = \{e \in E : e \text{ incident to } v\}$
- $\text{Set-cover of size} \leq k$ if $\text{vertex cover of size} \leq k$.

**Polynomial-Time Reduction**

**Basic strategies.**
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction by encoding with gadgets.
### 8.2 Reductions via "Gadgets"

**Basic reduction strategies:**
- Reduction by simple equivalence.
- Reduction from special case to general case.
- Reduction via "gadgets."

**Satisfiability**

- **Literal:** A Boolean variable or its negation.
  \[ x_i \text{ or } \neg x_i \]
- **Clause:** A disjunction of literals.
  \[ C_j = x_1 \lor x_2 \lor x_3 \]
- **Conjunctive normal form:** A propositional formula \( \Phi \) that is the conjunction of clauses.
  \[ \Phi = C_1 \land C_2 \land C_3 \land C_4 \]

**SAT:** Given CNF formula \( \Phi \), does it have a satisfying truth assignment?

**3-SAT:** SAT where each clause contains exactly 3 literals.

\[ \Phi = (x_1 \lor x_2 \lor x_3) \land (x_4 \lor x_5 \lor x_6) \land (x_7 \lor x_8 \lor x_9) \]

**Ex:**
- Yes: \( x_1 = \text{true}, x_2 = \text{true}, x_3 = \text{false} \)
- \( x_1 = \text{true}, x_2 = \text{false}, x_3 = \text{false} \)

**Claim:** 3-SAT \( \leq \) P INDEPENDENT-SET.

**Proof:** Given an instance \( \Phi \) of 3-SAT, we construct an instance \((G, k)\) of INDEPENDENT-SET that has an independent set of size \( k \) iff \( \Phi \) is satisfiable.

**Construction:**
- \( G \) contains 3 vertices for each clause, one for each literal.
- Connect 3 literals in a clause in a triangle.
- Connect literal to each of its negations.

**Example:**
- \( \Phi = (x_1 \lor x_2 \lor x_3) \land (x_4 \lor x_5 \lor x_6) \land (x_7 \lor x_8 \lor x_9) \)
- \( k = 3 \)
3 Satisfiability Reduces to Independent Set

Claim. \( G \) contains independent set of size \( k = |\Phi| \) iff \( \Phi \) is satisfiable.

Pf. \( \Rightarrow \) Let \( S \) be independent set of size \( k \).
- \( S \) must contain exactly one vertex in each triangle.
- Set these literals to true.
- Truth assignment is consistent and all clauses are satisfied.

Pf. \( \Leftarrow \) Given satisfying assignment, select one true literal from each triangle. This is an independent set of size \( k \).

\[
G \\
\begin{array}{c}
\text{\( \Phi = (\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_3) \land (\overline{x_1} \lor x_2 \lor \overline{x_3}) \)}
\end{array}
\]

Review

Basic reduction strategies.
- Simple equivalence: \textsc{Independent-Set} \( \leq_p \) \textsc{Vertex-Cover}.
- Special case to general case: \textsc{Vertex-Cover} \( \leq_p \) \textsc{Set-Cover}.
- Encoding with gadgets: \textsc{3-Sat} \( \leq_p \) \textsc{Independent-Set}.

Transitivity. If \( X \leq_p Y \) and \( Y \leq_p Z \), then \( X \leq_p Z \).

Pf idea. Compose the two algorithms.

Ex: \textsc{3-Sat} \( \leq_p \) \textsc{Independent-Set} \( \leq_p \) \textsc{Vertex-Cover} \( \leq_p \) \textsc{Set-Cover}.

8.3 Definition of NP
Decision Problems

Decision problem.
- X is a set of strings.
- Instance:  string s.
- Algorithm A solves problem X:  \( A(s) = \text{yes} \) iff \( s \in X \).

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:  \( X = \{ 2, 3, 5, 7, 11, 13, 17, 23, 29, 31, 37, ... \} \)
Algorithm.  [Agrawal-Kayal-Saxena, 2002] \( p(|s|) = |s|^8 \).

Decision Problem vs. Optimization Problem

Decision problem.  Does there exist a vertex cover of size \( \leq k \)?
Optimization problem.  Find vertex cover of minimum cardinality.

Self-reducibility.  Optimization problem \( \rightarrow \) decision version.
- Applies to all (NP-complete) problems in this chapter.
- Justifies our focus on decision problems.

Ex:  to find min cardinality vertex cover.
- (Binary) search for cardinality \( k^* \) of min vertex cover.
- Find a vertex \( v \) such that \( G - \{ v \} \) has a vertex cover of size \( \leq k^* - 1 \).
- any vertex in any min vertex cover will have this property
- Include \( v \) in the vertex cover.
- Recursively find a min vertex cover in \( G - \{ v \} \).

Definition of P

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

<table>
<thead>
<tr>
<th>Problem</th>
<th>Description</th>
<th>Algorithm</th>
<th>Year</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>11, 17</td>
<td>11, 16</td>
</tr>
<tr>
<td>RELPRIME</td>
<td>Are x and y relatively primes?</td>
<td>Euclid (300 BCE)</td>
<td>34, 39</td>
<td>34, 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>EDIT-DISTANCE</td>
<td>Is the edit distance between s and y less than 5?</td>
<td>Dynamic programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LSOLVE</td>
<td>Is there a vector x that satisfies ( Ax = b )?</td>
<td>Gauss-Edmonds elimination</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

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

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

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 if \( s \) is composite. Moreover \( |t| < |s| \).

**Certifier.**

```java
boolean C(s, t) {
    if (t < 1 or t > s)
        return false
    else if (remainder(s, t) == 0)
        return true
    else
        return false
}
```

**Instance.** \( s = 437,669 \).

**Certificate.** \( t = 541 \) or \( 809 \).

**Conclusion.** COMPOSITES is in NP.

Every problem in P is also 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.**

\[
(\overline{x_1} \lor x_2 \lor x_3) \land (x_1 \lor \overline{x_2} \lor x_4) \land (x_1 \lor x_2 \lor \overline{x_5})
\]

**Instance s**

\[ s_1 = 1, s_2 = 1, s_3 = 0, s_4 = 1 \]

**Certificate t**

**Conclusion.** SAT is in NP.
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 (and the first and the last nodes).

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

---

P, NP, EXP

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

**EXP.** Decision problems for which there is an exponential-time algorithm.

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

**Claim.** $P \subseteq NP$.

**Pf.** Consider any problem $X$ in $P$.
- By definition, there exists a poly-time algorithm $A(s)$ that solves $X$.
- Certificate: $t = \varepsilon$, certifier $C(s, t) = A(s)$.

**Claim.** $NP \subseteq EXP$.

**Pf.** Consider any problem $X$ in $NP$.
- By definition, there exists a poly-time certifier $C(s, t)$ for $X$.
- To solve input $s$, run $C(s, t)$ on all strings $t$ with $|t| \leq p(|s|)$.
- Return yes, if $C(s, t)$ returns yes for any of these.

---

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.
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 in polynomial time an input y from x 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.

Question. Are these two concepts the same?

we abuse notation ≤ and blur distinction

NP-hard and NP-Complete

NP-hard. A problem Y is NP-hard if, for every problem X in NP, X ≤p Y.

NP-complete. A problem Y is in NP and is NP-hard.

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

Pf. ⇒ If P = NP then Y can be solved in poly-time since Y is in NP.

Pf. ⇐ Suppose Y can be solved in poly-time.

- Let X be any problem in NP. Since X ≤p Y, we can solve X in poly-time. This implies NP ⊆ P.
- We already know P ⊆ NP. Thus P = NP.

Fundamental question. Do there exist "natural" NP-complete problems?
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?

![Circuit Diagram]

Yes: 1 0 1

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 polynomial time, then circuit is of polynomial size.
- Consider some problem X in NP. It has a polynomial 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) = yes,
  - View C(s, t) as an algorithm on |s| + p(|s|) bits (input s, certificate t) and convert it into a polynomial size circuit K.
  - First |s| bits are hard-coded with s
  - Remaining p(|s|) bits represent bits of t
  - Circuit K is satisfiable if C(s, t) = yes.

Example

**Ex.** 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.

![Example Diagram]
Establishing NP-Completeness

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

Recipe to establish NP-completeness of problem Y.

1. Show that Y is in NP.
2. Choose an NP-hard problem X.
3. Prove that X \leq_p Y.

Justification. If X is an NP-hard 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.

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.

1. Let K be any circuit.
2. Create a 3-SAT variable x_i for each circuit element i.
3. Make circuit compute correct values at each node:
   - x_2 = \neg x_3 \Rightarrow add 2 clauses: x_2 \lor x_3 \lor \bar{x}_2
   - x_1 = x_4 \lor x_5 \Rightarrow add 3 clauses: \bar{x}_1 \lor x_4 \lor \bar{x}_5
   - x_0 = x_1 \land x_2 \Rightarrow add 3 clauses: \bar{x}_0 \lor x_0 \lor x_1 \lor x_2
4. Hard-coded input values and output value.
   - x_5 = 0 \Rightarrow add 1 clause: \bar{x}_5
   - x_0 = 1 \Rightarrow add 1 clause: x_0
5. Final step: turn clauses of length < 3 into
   clauses of length exactly 3 by introducing new
   variables.

Example

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

Example diagram

Diagram showing the circuit and its connections.
Observation. All problems below are NP-complete and polynomial reduce to one another!

CIRCUIT-SAT
3-SAT
DIRECTED HAM-CYCLE
INDEPENDENT SET
GRAPH 3-COLOR
HAM-CYCLE
TSP
SUBSET-SUM
SCHEDULING

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, 3-DNF.
- Sequenting problems: HAMILTONIAN-CYCLE, TSP.
- Partitioning problems: 3-D-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.

Extent and Impact of NP-Completeness

Extent of NP-completeness. (Papadimitriou 1995)

- Prime intellectual export of CS to other disciplines.
- 6,000 citations per year (title, abstract, keywords).
- More than "compiler", "operating system", "database"
- Broad applicability and classification power.
- "Captures vast domains of computational, scientific, mathematical endeavors; and seems to roughly delimit what mathematicians and scientists had been aspiring to compute feasibly."
More Hard Computational Problems

Aerospace engineering: optimal mesh partitioning for finite elements.
Biology: protein folding.
Chemical engineering: heat exchanger network synthesis.
Civil engineering: equilibrium of urban traffic flow.
Economics: computation of arbitrage in financial markets with friction.
Electrical engineering: VLSI layout.
Environmental engineering: optimal placement of contaminant sensors.
Financial engineering: find minimum risk portfolio of given return.
Genomics: phylogeny reconstruction.
Mechanical engineering: structure of turbulence in sheared flow.
Medicine: reconstructing 3-D shape from biplane angiogram.
Operations research: optimal resource allocation.
Physics: partition function of 3-D Ising model in statistical mechanics.
Politics: Shapley-Shubik voting power.
Pop culture: Minesweeper consistency.
Statistics: optimal experimental design.

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, 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. $X = \{0, 1, 4, 6, 8, 9, 10, 12, 14, 15, \ldots \}$
$\overline{X} = \{2, 3, 5, 7, 11, 13, 17, 23, 29, \ldots \}$
Equivalence: Since $X \leq_P X$ and $X \leq_P X$, we have $X \equiv_P X$.
(Cook's definition only)
co-NP. Complements of decision problems in NP.
Ex. TAUTOLOGY, NO-HAM-CYCLE, PRIMES.

NP = co-NP?

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.
Pf idea.
• 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 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 $|N(S)| < |S|$.  

Good Characterizations

Observation. $P \subseteq NP \cap co-NP$.
- 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 $P = NP \cap co-NP$?
- 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 $NP \cap co-NP$, but not known to be in $P$.

If poly-time algorithm for factoring, can break RSA cryptosystem.

Fact. $PRIMES$ is in $NP \cap co-NP$.

Theorem. $PRIMES$ is in $NP \cap co-NP$.

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 $1 < t < s$ s.t.

\[
\begin{align*}
1^r & \equiv 1 \pmod{s} \\
(t^{(s-1)/r}) & \equiv 1 \pmod{s} \\
\text{for all prime divisors } p \text{ of } s-1
\end{align*}
\]

Certifier.
- Check $s-1 = 2 \times 3 \times 36,473$.
- Check $17(s-1) = 1 \pmod{s}$.
- Check $17(s-1)/2 \equiv 437,676 \pmod{s}$.
- Check $17(s-1)/3 \equiv 329,415 \pmod{s}$.
- Check $17(s-1)/36,473 \equiv 305,452 \pmod{s}$.

Input. $s = 437,677$
Certificate. $t = 17, 2 \times 3 \times 36,473$

prime factorization of $s-1$
does need a recursive certificate to assert that 3 and 36,473 are prime

FACTOR is in $NP \cap co-NP$

FACTORIZE. Given an integer $x$, find its prime factorization.

FACTOR. Given integers $x$ and $y$, does $x$ have a nontrivial factor $> y$?

Theorem. FACTOR $\equiv_P$ FACTORIZE.

Theorem. FACTOR is in $NP \cap co-NP$.

Pf.
- 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 ≤ COMPOSITES ≤ FACTOR.

Natural question: Does FACTOR ≤ 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, must find efficient factoring algorithm.