More examples of problems in NP.

**Example. 2-COLORABILITY.**

**Instance:** A simple undirected graph \(G=(V,E)\).

**Question:** Is \(G\) 2-colorable? That is, does there exist a function \(f: V \rightarrow \{1, 2\}\) such that \(f(u) \neq f(v)\) for all \(\{u, v\}\in E\)? (Equivalently, is \(G\) bipartite?)

**Certificate:** A function \(f : V \rightarrow \{1, 2\}\). This has size \(O(|V|)\), which is polynomial in the size of the instance.

**Verifier:**

- For every \(v \in V\), if \(f(v) \in \{1, 2\}\), then output "no" and stop.
- For every \(\{u, v\} \in E\), if \(f(u) = f(v)\), then output "no" and stop.
- Output "yes."

This verification can be done in \(O(|V| + |E|)\) time.

**Example. 3-COLORABILITY.**

[Just like 2-COLORABILITY.]
Example. HAMILTONIAN CIRCUIT.

Instance: A simple undirected graph $G=(V,E)$.

Question: Does $G$ contain a Hamiltonian circuit (i.e., a spanning cycle)?

Certificate: A list of integers $(v_0, v_1, v_2, ..., v_n)$ naming the vertices in order around a Hamiltonian circuit. This has size $O(|V|)$. (Or $O(|V| \log |V|)$, if you want to count bits.)

Verifier:
- If $n \neq |V|$, output "no" and stop.
- If $v_0 \neq v_n$, output "no" and stop.
- For each $i \in \{0, 1, 2, ..., n-1\}$, if $v_i \notin \{1, 2, ..., n\}$, output "no" and stop.
- For each $i \in \{0, 1, 2, ..., n-2\}$, for each $j \in \{i+1, i+2, ..., n-1\}$, if $v_i = v_j$, output "no" and stop.
- For each $i \in \{0, 1, 2, ..., n-1\}$, if $\exists v_i, v_{i+1} \notin E$, output "no" and stop.
- If $n < 3$, output "no" and stop.
- Output "yes."

This verification can be done in $O(|V|^2)$ time.
Example. SAT. [P&S Example 15.7, §15.3]

Instance: A propositional formula $F$ on the Boolean variables $x_1, \ldots, x_n$.

Question: Is $F$ satisfiable? That is, does there exist an assignment of truth values to $x_1, \ldots, x_n$ such that the resulting truth value of $F$ is TRUE?

Certificate: Truth values for all variables. This has size $O(n)$.

Verifier:
- If the certificate does not consist of exactly $n$ bits, output "no" and stop.
- Evaluate $F$ using the given truth values for the variables $x_1, \ldots, x_n$. If the result is FALSE, output "no" and stop.
- Output "yes."

This verification can be done in $O(m)$ time, where $m$ is the length of the formula $F$. (Note: $m$ is not the number of variables, because each variable may appear many times in $F$.)

2 July
Example. ILP. [P&S Example 15.8, §15.3]

Instance: An $m \times n$ matrix $A$ of integers and a vector $b$ of $m$ integers.

Question: Does there exist a vector $x$ of $n$ integers such that $Ax = b$ and $x \geq 0$?

Certificate: A vector $x$ of $n$ integers.
[See P&S Example 15.8, and P&S Thm 13.4 in §13.3, for careful justification that a feasible IP always has a polynomial-size feasible solution.]

Verifier: Output "yes" iff all entries of $x$ are integers, $Ax = b$, and $x \geq 0$. This verification can be done with $O(mn)$ arithmetic operations.
Aside: The class \textit{co-NP} [P&S §16.1]

**Defn.** The complement of a decision problem \( A \) is the decision problem \( \overline{A} \) in which an instance is the same as an instance of \( A \) and in which the answer to an instance \( x \) is "yes" if and only if the answer to \( x \) in \( A \) is "no."

**Defn.** The class \textit{co-NP} is the class of decision problems whose complement is in \( NP \).

---

So, a decision problem is in \( co-NP \) iff all "no" instances have polynomial-size "co-certificates" proving that the answer is "no," verifiable by a "co-verifier" in polynomial time.

---

Intuitively:

- A decision problem is in \( NP \) when you can efficiently prove "yes" answers.
- A decision problem is in \( co-NP \) when you can efficiently prove "no" answers.

**Example.** The complement of \textit{COMPOSITENESS} is \textit{PRIMALITY} (well, counting 1 as prime). \textit{COMPOSITENESS} is in \( NP \), so \textit{PRIMALITY} is in \( co-NP \).
Example. 2-COLORABILITY is in co-NP, because a "co-certificate" to prove a "no" answer is an odd cycle. (A graph is bipartite if and only if it contains no odd cycle.)

Example. 3-COLORABILITY \notin co-NP.
Nobody knows an efficient "co-certificate" to prove that a graph is not 3-colorable. (But, on the other hand, it is also true that nobody has proven 3-COLORABILITY is not in co-NP.)

Example. HAMILTONIAN CIRCUIT \notin co-NP.
Same situation as for 3-COLORABILITY. Nobody knows an efficient way to prove that a general graph does not have a Hamiltonian circuit.

Example. P \subseteq NP \cap co-NP.
P \subseteq co-NP for the same reason that P \subseteq NP: the "co-certificate" can be nothing, and the "co-verifier" can verify a "no" answer by just solving the instance.

Polynomial-time reductions [P&S §15.4]

Defn. Let $A_1$ and $A_2$ be decision problems. We say that $A_1$ reduces in polynomial time to $A_2$ iff there exists a polynomial-time algorithm $A_1$ for $A_1$ that uses a (hypothetical) algorithm $A_2$ for $A_2$ as a subroutine at unit cost. We call $A_1$ a polynomial-time reduction from $A_1$ to $A_2$.

Note: The phrase "at unit cost" in this definition means that in measuring the running time of $A_1$ we are counting the execution of $A_2$ as a single elementary operation.

In reality, of course, such an algorithm $A_2$ for $A_2$ almost certainly takes many elementary operations. But counting $A_2$ as a single elementary operation is justifiable in light of the following:

Proposition. [P&S Prop. 15.1] If $A_1$ polynomially reduces to $A_2$ and there exists a polynomial-time algorithm for $A_2$, then there exists a polynomial-time algorithm for $A_1$. 

2 July
Proof. Let the polynomial \( p_1(n) \) bound the running time of \( A_1 \) (with the assumption of unit-cost invocation of \( A_2 \)), and let the polynomial \( p_2(n) \) bound the running time of \( A_2 \). Then the actual number of elementary operations used to run \( A_1 \) on an instance of size \( n \), counting all operations used by the calls to \( A_2 \), is bounded by

\[
p(n) = p_1(n) \cdot p_2(p_1(n))
\]

because \( A_1 \) makes at most \( p_1(n) \) calls to \( A_2 \), and the largest possible input to \( A_2 \) is \( p_1(n) \) even if \( A_1 \) used all of its steps just to write that input, so each call to \( A_2 \) takes at most \( p_2(p_1(n)) \) elementary operations. Since \( p(n) \) is a polynomial, this is a polynomial-time algorithm for \( A_1 \). \( \square \)

In a polynomial-time reduction, \( A_2 \) may be called many times (well, only polynomially many times) by \( A_1 \), and the operation of \( A_1 \) may depend on the results of earlier calls to \( A_2 \). But there is a particularly interesting kind of polynomial-time reduction in which \( A_1 \) calls \( A_2 \) only once, at the very end, and then directly returns the result from \( A_2 \)
Poly-time reductions -2

Defn. We say that a decision problem $A_1$ polynomially transforms to another decision problem $A_2$ if there is a polynomial-time algorithm to convert any instance $x$ of $A_1$ to an instance $y$ of $A_2$ such that the answer to $x$ is "yes" if and only if the answer to $y$ is "yes".

Example. CNF-SAT polynomially transforms to ILP.

(CNF-SAT is a special case of SAT in which the instances are restricted to be formulas in conjunctive normal form.)

We saw an IP formulation for CNF-SAT in the lecture on June 24. For example, the CNF-SAT instance

$$(x_1 \lor \overline{x}_2 \lor \overline{x}_3) \land (\overline{x}_1 \lor x_3) \land (x_2 \lor x_3 \lor \overline{x}_4)$$

can be converted (in polynomial time) to the IP

$$\text{max } 0$$
$$\text{s.t. } x_1 + (1-x_2) + (1-x_3) \geq 1$$
$$\quad (1-x_1) + x_3 \geq 1$$
$$\quad x_2 + x_3 + (1-x_4) \geq 1$$
$$\quad x_i \in \{0,1\} \text{ for all } i.$$
ILP is the decision version of integer programming; the question is, "Is this IP feasible?" (Or, equivalently here, "Does this IP have a feasible solution with objective value \( \geq 0 \)?)

The answer to ILP for the IP formulation of a CNF-SAT instance is "yes" if and only if the original CNF formula is satisfiable. So this is a polynomial-time transformation. ✓

Example. HAMILTONIAN CIRCUIT polynomially transforms to TSP.

Recall the decision version of TSP:
Instance: \( n \) cities, the cost of each arc \((i,j)\), and a value \( L \in \mathbb{R} \).
Question: Does there exist a tour through all cities having total cost \( \leq L \)?

Given an instance of HAMILTONIAN CIRCUIT, i.e., a graph \( G=(V,E) \), construct an instance of TSP as follows: Set \( n = |V| \), set the cost of arc \((i,j)\) to 0 if \( (i,j) \in E \) or 1 otherwise, set \( L = 0 \). Then the answer to the TSP instance is "yes" (i.e., there exists a tour of total cost 0) if and only if \( G \) has a Hamiltonian circuit. This conversion can be done in polynomial time, so this is a polynomial-time transformation. ✓
Example. CLIQUE polynomially transforms to INDEPENDENT SET.

**CLIQUE:**
- Instance: Graph $G=(V,E)$, integer $k$.
- Question: Does $G$ contain a clique of size $k$, i.e., a subset $K \subseteq V$ with $|K|=k$ such that every two vertices in $K$ are adjacent?

**INDEPENDENT SET:**
- Instance: Graph $G=(V,E)$, integer $k$.
- Question: Does $G$ contain an independent set of size $k$, i.e., a subset $S \subseteq V$ with $|S|=k$ such that no two vertices in $S$ are adjacent?

Given an instance $(G, k)$ of CLIQUE, convert $G$ to its complement $\overline{G}$ (change edges to non-edges and vice versa) to get an instance $(\overline{G}, k)$ of INDEPENDENT SET. The graph $\overline{G}$ has an independent set of size $k$ if and only if $G$ has a clique of size $k$. √
Defn. A decision problem $A$ is called NP-complete if

- $A \in \text{NP}$ and
- all other problems in NP polynomially transform to $A$.

At the moment it is not clear that any such problems exist (this is the result of Cook's theorem—tomorrow's lecture), but

- if a decision problem $A$ is NP-complete, and
- if there exists a polynomial-time algorithm for $A$,

then, as a consequence of the proposition from earlier, we would have a polynomial-time algorithm for all problems in NP!

This would mean $P=NP$, which appears not to be true (because no one has ever been successful in finding a polynomial-time algorithm for any NP-complete problem).

So, in a meaningful sense, NP-complete problems are the hardest problems in NP: if we could solve any NP-complete problem in poly time, then we could solve all problems in NP in poly time.