Cook's Theorem: SAT is NP-complete.

- SAT in NP:
  Given F,c, where c is a setting of values (True/False) for the variables of F:

  Output the value of F under the setting given by c.

  This can be carried out in polynomial time: given a formula F and a setting of its variables, just substitute the values for each variable and then evaluate each connective one-by-one, from the inside out.
  Moreover, if F is satisfiable, then there is some value of c that will make this verifier output yes (when $c = a$ setting that makes F true); and if F is not satisfiable, then this verifier will output *no* for every possible value of c (since no setting makes F true).

  The same reasoning shows that Circuit-SAT, CNF-SAT and 3SAT also belong to NP.

- SAT is NP-hard (main idea):
  Let D be any problem in NP. By definition, there is a polytime verifier $V(x, c)$ for D. This polytime verifier can be implemented as a circuit with input gates representing the values of x and c. For any input x for D, we can hard-code the value of x into this circuit in such a way that there is a value of the certificate for which the verifier outputs *yes* iff there is some setting of the input gates corresponding to c that make the circuit output 1. It's possible to show that this transformation can be carried out in polynomial time (as a function of the size of x), and it's also possible to show that this circuit can then be translated into a formula in CNF (in polytime) such that settings of the circuit's input gates correspond to settings of the formula's variables.

  This shows that Circuit-SAT, SAT, and CNF-SAT are all NP-hard.

NP-completeness examples:
VERTEX-COVER: $\{< G, k > :$ G is a graph that contains a vertex cover of size k, *i.e.*a set C of k vertices such that each edge of G has at least one endpoint in $C\}$

VERTEX-COVER (VC) is NPC:

- VC in NP: Given G, k, c, we can verify in polytime that c represents a vertex cover of size k in G.
- VC is NP-hard: 3SAT $\leqslant_p$ VC.
  Given $F = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_r \vee b_r \vee c_r)$, where $a_i, b_i, c_i \in \{x_1, \sim x_1, x_2, \sim x_2, \cdots, x_s, \sim x_s\}$, construct $G = (V, E)$ and k such that F satisfiable iff G contains vertex cover of size k, as follows:

  $k = s + 2r$

  $V = \{a_1, b_1, c_1, \cdots, a_r, b_r, c_r, x_1, \sim x_1, \cdots, x_s, \sim x_s\}$

  $E = \{(x_i, \sim x_i) : 1 \leqslant i \leqslant s\} \cup \{(a_i, b_i), (b_i, c_i), (c_i, a_i) : 1 \leqslant i \leqslant r\} \cup \{(l, x) : l = a_i \text{ or } b_i \text{ or } c_i, \text{ and } x = x_j \text{ or } \sim x_j \text{ corresponding to } l\}$

  For example, if $F = (x_1 \vee \sim x_2 \vee \sim x_4) \wedge (x_2 \vee \sim x_3 \vee x_1) \wedge (\sim x_3 \vee x_4 \vee \sim x_2)$, then $a_1 = x_1, b_1 =\sim x_2, c_1 =\sim x_4, a_2 = x_2, b_2 =\sim x_3, c_2 = x_1, a_3 =\sim x_3, b_3 = x_4, c_3 =\sim x_2$ so

  $k = 4 + 2 \times 3 = 10$

  $V = \{a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3, x_1, \sim x_1, x_2, \sim x_2, x_3, \sim x_3, x_4, \sim x_4\}$

  $E = \{(x_1, \sim x_1), (x_2, \sim x_2), (x_3, \sim x_3), (x_4, \sim x_4), (a_1, b_1), (b_1, c_1), (c_1, a_1), (a_1, x_1), (b_1, \sim x_2), (c_1, \sim x_4), (a_2, b_2), (b_2, c_2), (c_2, a_2), (a_2, x_2), (b_2, \sim x_3), (c_2, x_1), (a_3, b_3), (b_3, c_3), (c_3, a_3), (a_3, \sim x_3), (b_3, x_4), (c_3, \sim x_2)\}$

Clearly, construction can be done in polytime (with one scan of F).

Also, if F is satisfiable, then there is an assignment of truth values that make at least one literal in each clause true. Pick a cover C as follows: for each variable, C contains x_i or $\sim x_i$, whichever is true under the truth assignment; for each clause, C contains every literal except one that's true (pick arbitrarily if more than one true literal). C contains exactly $s+2r$ vertices and is a cover: all edges $(x_i, \sim x_i)$ are covered; all edges in clause triangles are covered (because we picked two vertices from each triangle); all edges between "clauses" and "variables" are covered (two from inside triangle, one from true literal for that clause).

Finally if G contains a cover C of size $k = s + 2r$, C must contain at least one of x_i or $\sim x_i$ for each i (because of edges $(x_i, \sim x_i)$) and at least two of a_i, b_i, c_i for each i (because of triangle), so only way for C to have size $s + 2r$ is to contain exactly one of x_i or $\sim x_i$ and exactly two of a_i, b_i, c_i, for each i. Since C covers all edges with only two vertices per triangle, the third vertex in each triangle must have its "outside" edge covered because of x_i or $\sim x_i$. If we set literals according to choices of x_i or $\sim x_i$ in C, this will make formula F true: at least one literal will be true in each clause (because at least one edge from "variables" to "clauses" is covered by the variable in C).

SUBSET-SUM: Given a set of positive integers S and a positive integer target t, is there some subset S' of S whose sum is exactly t, i.e., $\sum_{x \in S'} x = t$?

SUBSET-SUM (SS) is NPC:

- SS is in NP because it takes polytime to verify that the certificate represents a subset of S whose sum is t (1- check if all items in the certificate c is in S. 2- check if sum of the items in c is t).

- SS is NP-hard because 3SAT $\leqslant_p$ SS:
  Given formula $F = (a_1 \vee b_1 \vee c_1) \wedge \cdots \wedge (a_r \vee b_r \vee c_r)$ where $a_i, b_i, c_i \in \{x_1, \sim x_1, \cdots, x_s, \sim x_s\}$, construct numbers as follows:

  - For $j = 1, ..., s$:
    number $x_j = 1$ followed by $s - j$ 0s followed by r digits where k-th next digit equals 1 if x_j appears in clause C_k, 0 otherwise;
    number $\sim x_j = 1$ followed by $s - j$ 0s followed by r digits where k-th next digit equals 1 if $\sim x_j$ appears in clause C_k, 0 otherwise.

  - For $j = 1, ..., r$:
    number $C_j = 1$ followed by $r - j$ 0s and
    number $D_j = 2$ followed by $r - j$ 0s.

  - Target t = s 1s followed by r 4s.

  Clearly, this can be constructed in polytime.

  Example of reduction for $F = (x_1 \vee \sim x_2 \vee \sim x_4) \wedge (x_2 \vee \sim x_3 \vee x_1) \wedge (\sim x_3 \vee x_4 \vee \sim x_2)$:

  So the numbers are:

	x_1	x_2	x_3	x_4	C_1	C_2	C_3
x_1	1	0	0	0	1	1	0
$\sim x_1$	1	0	0	0	0	0	0
x_2	0	1	0	0	0	1	0
$\sim x_2$	0	1	0	0	1	0	1
x_3	0	0	1	0	0	0	0
$\sim x_3$	0	0	1	0	0	1	1
x_4	0	0	0	1	0	0	1
$\sim x_4$	0	0	0	1	1	0	0
C_1	0	0	0	0	1	0	0
D_1	0	0	0	0	2	0	0
C_2	0	0	0	0	0	1	0
D_2	0	0	0	0	0	2	0
C_3	0	0	0	0	0	0	1
D_3	0	0	0	0	0	0	2
t	1	1	1	1	4	4	4

dummies to get clause columns to sum to 4

$$S = \left\{ \begin{array}{rr} x_1 = & 1000110, \\ \sim x_1 = & 1000000, \\ x_2 = & 100010, \\ \sim x_2 = & 100101, \\ x_3 = & 10000, \\ \sim x_3 = & 10011, \\ x_4 = & 1001, \\ \sim x_4 = & 1100, \\ D_1 = & 200, \\ C_1 = & 100, \\ D_2 = & 20, \\ C_2 = & 10, \\ D_3 = & 2, \\ C_3 = & 1. \end{array} \right\}$$

and $\quad t = \; 1111444$

If F is satisfiable, then there is a setting of variables such that each clause of F contains at least one true literal. Consider the subset $S' = \{$numbers that correspond to true literals$\}$. By construction, $\sum_{x \in S'} x = s$ 1s followed by r digits, each one of which is either 1, 2, or 3 (because each clause contains at least one true literal). This means it is possible to add suitable numbers from $\{C_1, D_1, ..., C_r, D_r\}$ so that the last r digits of the sum are equal to 4, i.e., there is a subset S' such that $\sum_{x \in S'} x = t$.

If there is a subset S' of S such that $\sum_{x \in S'} x = t$, then S' must contain exactly one of $\{xj, \sim xj\}$ for $j = 1, ..., n$, because that is the only way for the numbers in S' to add to the target (with a 1 in the first s digits). Then, F is satisfied by setting each variable according to the numbers in S': for each clause j, the corresponding digit in the target is equal to 4 but the numbers C_j and D_j together only add up to 3 in that digit; this means that the selection of numbers in S' must include some literal with a 1 in t.

Template for proofs of NP-completeness: To show A is NPC, prove that

- A in NP: Describe a polytime verifier for A.
  "Given (x, c), check c has correct format and properties..."
  Argue that verifier runs in polytime and that x is a yes-instance iff verifier outputs "yes" for some c.

  Note that all problems in NP we've seen so far have a similar structure to their definition: "the answer for object A is Yes iff there is some related object B such that some property holds about A and B" –

for example, for CLIQUE: "the answer for undirected graphs G and integers k is Yes iff there is a subset of vertices C that forms a k-clique in G". For all such problems, the verifier will also have a common structure: "on input (A, c), check that c encodes an object B and that A and B have the required property". Because of the way these decision problems are defined, this guarantees (A, c) is accepted for some c iff A is a yes-instance. All that remains is to ensure checking property of A, B can be done in polytime.

- A is NP-hard: Show $B \leqslant_p A$ for some NP-hard problem B.
  "Given x, construct y_x as follows: ..."
  Argue that construction can be carried out in polytime and that x yes-instance iff y_x yes-instance (often by showing x yes-instance $\Rightarrow$ y_x yes-instance and y_x yes-instance $\Rightarrow$ x yes-instance)
  In more detail, this involves:
  - starting with arbitrary input y for B (i.e., without making any assumption about whether y is a yes-instance or a no-instance),
  - describing explicit construction of specific input x_y for A,
  - arguing construction can be carried out in polytime,
  - arguing if y is a yes-instance, then so is x_y,
  - arguing if x_y is a yes-instance, then so was y (or equivalently, if y is a no-instance, then so is x_y).

  Watch last step! Argument starts from x_y constructed earlier (not from arbitrary input x for A), and relates it to arbitrary y that x_y was constructed from.

Traps to watch out for:

- Direction of reduction: must start from arbitrary input x for B (cannot place any restrictions on input; reduction must work with all possible inputs) and explicitly construct specific input y_x for A.
- "Reduction" that does something different for yes-instances vs. no-instances: this would involve telling the difference, which can't be done in polytime when B is NP-hard.

Some NP-Complete problems: