#1 Stuff I forgot to cover—The P vs NP problem

I wanted to touch on this a bit, because it is important you understand the relationship between P and NP.

Firstly,

Theorem 1. *P ⊆ NP.*

Proof. Let's do this as a class. The gist is that every polynomial time decider can be thought of as a poly-time verifier that does not use its certificate. □

A question for you: is this theorem true?

Theorem 2. *NP ⊆ P?*

Next, let EXPTIME $= \bigcup_k^\infty \text{TIME}(2^{n^k})$. Then:

Theorem 3. *NP ⊆ EXPTIME.*

Proof. Hint: given a polynomial time verifier V, can we "guess" the certificate? In what running time? Further hint: what is an upper bound on the length of the certificate? And how much time does it take to generate each 0-1 string of a certificate of that length? □

In the end, what we know is this:

$$\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXPTIME}$$

Which helps explain why we think of things in P as "efficient" and things in NP as "inefficient".

#2 SAT and 3SAT

A few definitions and then another NP-complete problem for ya.
First:

Definition 1. *A formula ϕ is in conjunctive-normal form if ϕ is composed of the conjunction of n clauses*

$$C_1 \wedge C_2 \wedge \ldots \wedge C_n$$

where each clause C_i has the form

$$(x_1 \vee x_2 \vee \ldots x_m)$$

for boolean literals $x_1 \ldots x_m$, each of which are either (i) a variable or (ii) a negated variable.

For example, the formula

$$\phi = (x_1 \vee \neg x_2) \wedge (\neg x_3 \vee \neg x_2 \vee x_4 \vee x_5)$$

is in conjunctive-normal form because:

1. ϕ is the conjunction of two clauses; and
2. each clause is the disjunction of (possibly negated) variables.

Next,

Definition 2. *a <u>3CNF-formula</u> is a formula ϕ in conjunctive normal form such that each clause has exactly three literals.*

E.g.,

$$\phi = (x_1 \vee \neg x_2 \vee \neg x_1) \wedge (\neg x_3 \vee \neg x_2 \vee x_4)$$

is a 3CNF-formula.

The reason we do this all this is that all formulae can be transformed to 3CNF-formulae. And 3CNF-formulae are easier to reason and think about. It is fairly easy to describe how to transform an arbitrary formulae ϕ to a 3CNF-formula inductively over the grammar $B \to \Sigma^* \mid \neg B \mid B \wedge B \mid B \vee B$. I would like to describe this process because it's fundamentally a PL problem (it is a transformation on an AST)—which usually means I should omit it for time.

Finally, if

$$\text{3SAT} = \{\langle \phi \rangle \mid \phi \text{ is a satisfiable 3CNF-formula. }\}$$

then:

Theorem 4. 3SAT *is* NP*-complete.*

Proof. Modify the proof that SAT is NP-complete, or reduce SAT to 3SAT (take your pick). $\square$

What is important here is that we have another NP-complete language to use in showing *other* languages are NP-complete—and 3SAT is often easier to use to do so.

#3 Other NP-Complete problems

For example...

#3.1 clique

This one actually comes from section 7.4 (not 7.5), but I skipped it.

First,

Definition 3. *Let $G = (V, E)$ be a graph. A <u>k-clique</u> in G is a subgraph of G (with $\geq k$ vertices) in which every two vertices are connected by an edge. (Or, a k-<u>clique</u> is a <u>completely-connected</u> subgraph of G of size $\geq k$.)*

Then

$$\text{CLIQUE} = \{\langle G, k \rangle \mid G \text{ is an undirected graph with a } k\text{-clique.}\}$$

Now,

Theorem 5. CLIQUE *is* NP-*complete.*

Proof. First, is CLIQUE in NP? Yes. Our certificate is simply the clique. Then we can verify if (i) the clique has at least k nodes, and (ii) the clique is a clique.
Next, we show that 3SAT $\leq_p$ CLIQUE. Consider an arbitrary formula ϕ in 3SAT.

$$\phi = (a_1 \vee b_1 \vee c_1) \wedge (a_2 \vee b_2 \vee c_2) \wedge ...(a_1 \vee b_1 \vee c_1)$$

Our reduction f will generate the string $\langle G, k \rangle$, with G defined as follows.

1. The vertices in G are organized into k groups of three vertices each called the triples $T_1, ..., T_k$. Each triple corresponds to a clause in ϕ; each vertex in a triple corresponds to a literal in a clause. Label each vertex by its literal.

2. The edges of G connect all but two types of pairs of vertices in G.

   (a) No edge is present between vertices in the same triple, and

   (b) no edge is present between two nodes with contradictory labels—e.g., if v_1 is labeled x_1 and v_2 is labeled $\neg x_1$, then do not draw an edge between v_1 and v_2.

Let's consider what this looks like on:

$$\phi = (x_1 \vee x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_2)$$

(I will draw the picture from pp. 303.)
For correctness:

1. Suppose ϕ has a satisfying assignment—then at least one literal is true in each clause. Select one true literal from each clause—they will have an edge drawn between them, so they form a clique of size k.

2. Suppose G has a k-clique. No two of the clique's vertices occur in the same triple, by construction. Therefore, each of the k triples contains exactly one of the k clique nodes. Assign truth values to the variables in ϕ so that each literal labeling a clique node is made true; this will be satisfiable because no two contradictory nodes are connected by an edge. The result is an assignment to ϕ in which each clause has at least one true literal (so $\phi \in$ 3SAT).

□

#4 Stuff from §7.5 I had to cut for time

Please use §7.5 as a reference for (i) how to write NP-completeness proofs, and (ii) the following problems.

1. VERTEX-COVER is NP-complete (7.44).

2. The (directed and undirected) Hamiltonian Path problem is NP-complete. (A Hamiltonian Path in a graph G is a path p that visits each vertex exactly once.)

3. SUBSET-SUM is NP-complete.