READINGS: Sections 34.4, 34.5

Review:

- We can think of a decision problem as a set A of strings over some finite alphabet. We may as well assume the alphabet is $\{0,1\}$, since we can code other symbols as bit strings (e.g. ASCII code).

  Thus if $D(x)$ is a decision problem, then the corresponding set $A = \{x \mid D(x) = 1\}$.

- **Reducibility**: We define $A \leqslant_p B$ (A is polytime reducible to B) as follows:

  $A \leqslant_p B$ iff there is a polytime computable functions $f : \{0,1\}^* \to \{0,1\}^*$ such that

$$x \in A \Leftrightarrow f(x) \in B \text{ for all } x \in \{0,1\}^*$$

- Think $A \leqslant_p B$ means A is no harder than B.
- **Facts about $\leqslant_p$**
  1. $\leqslant_p$ is **transitive**. That is, $A \leqslant_p B$ and $B \leqslant_p C$ implies $A \leqslant_p C$.
  2. $A \leqslant B$ and $B \in \mathbf{P}$ implies $A \in \mathbf{P}$
     $A \leqslant B$ and $B \in \mathbf{NP}$ implies $A \in \mathbf{NP}$
  3. **Definition**: A is **NP**-*hard* if $B \leqslant_p A$ for all $B \in \mathbf{NP}$.
  4. **Definition**: A is **NP**-*complete* if A is **NP**-hard and $A \in \mathbf{NP}$.
- **Important Theorem:** If A is **NP** hard and $A \leqslant_p B$ then B is **NP** hard.

  Proof: This follows immediately from the definition of **NP**-hard and the transitivity of $\leqslant_p$.

The above Theorem is important, since once we know that one problem **NP**-complete, we can use it to show many other problems are also **NP**-complete. The original problem shown to be **NP**-complete is SAT (the Boolean satisfiability problem). Here

$$\text{SAT} = \{\varphi \mid \varphi \text{ is a satisfiable Boolean formula}\}$$

Cook-Levin Theorem:[1971] SAT is **NP**-complete.

It is easy to see that SAT is in **NP**: The certificate for a YES instance for a formula φ is a truth assignment to the variables of φ that satisfies φ (i.e. makes φ true).

The hard part of the proof is to show that SAT is **NP**-hard (every **NP** problem is polytime reducible to SAT). We will not prove this part in this course . (However last week we gave an intuitive argument that the related problem Circuit-SAT is **NP-complete**.)

Recall that a CNF (Conjunctive Normal Form) formula is a conjunction $C_1 \wedge C_2 \wedge \cdots \wedge C_m$ of clauses ($m \geqslant 1$) where a clause $(\ell_1 \vee \cdots \vee \ell_j)$ ($j \geqslant 1$) is a disjunction of literals, where each ℓ_i has the form x or $\bar{x}$, where x is a Boolean variable.

CNF-SAT is the problem SAT restricted to CNF formulas, and 3SAT is CNF-SAT restricted to formulas with exactly 3 distinct literals in each clause.

We will show that 3SAT is **NP**-complete. This is a useful result, since it can be used to show many other problems are **NP**-hard.

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

Fact: 2SAT is in **P**. It is an interesting excercise to give a polytime algorithm for 2SAT (not easy, but not that hard).

The easy part of the 3SAT Theorem is to show that 3SAT is in **NP**. The argument is that same as for SAT: A certificate for φ is a truth assignment to the variables of φ which make φ true.

To show that 3SAT is **NP**-hard we use both the Important Theorem above and the Cook-Levin Theorem. Thus it suffices to prove the following:

Lemma 1 SAT $\leqslant_p$ 3SAT.

Our definition above of 3SAT requires that the input formula φ has exactly 3 literals in every clause. It is useful to relax this requirement a little:

Definition: 3SAT' is the same as 3SAT except we allow each clause to have *at most* 3 literals.

Lemma 2: 3SAT' $\leqslant_p$ 3SAT.

This is an easy result. Suppose for example that an φ' is an input to 3SAT', and one of the clauses $(\ell_1 \vee \ell_2)$ has only two literals. Then we can replace this clause by two clauses $(\ell_1 \vee \ell_2 \vee y) \wedge (\ell_1 \vee \ell_2 \vee \bar{y})$, where y is a new variable. It is easy to see that the resulting formula is satisfiable iff the orginal formula φ is satisfiable. We then apply this idea repeatedly to convert all of the small clauses to conjuctions of 3-literal clauses.

Lemma 3 SAT $\leqslant_p$ 3SAT'

Note that **Lemma 1** follows from Lemmas 1 and 2, because $\leqslant_p$ is transitive. Thus it suffices to prove **Lemma 3**.

Definition: If φ and φ' are two formulas, then we say φ is *equivalent* to φ' (written $(\varphi \equiv \varphi')$) if $\tau(\varphi) = \tau(\varphi')$ for every truth assignment τ.

To prove **Lemma 3** we need a polytime algorithm which does the following:

Transform a Boolean formula φ to a 3CNF' formula φ' so φ is satisfiable iff φ' is satisfiable. (1)

To do this, it is tempting to use the fact that every Boolean formula φ is equivalent to a CNF formula. But there are two problems with this approach. The first difficulty is that there are formulas such as $x_1 \vee x_2 \vee x_3 \vee x_4$ that our not equivalent to *any* 3CNF formula. The second is that the CNF form φ' of φ could be exponentially bigger than φ, so there is no possibility of a polytime algorithm for producing φ' on input φ.

So instead of trying to make φ' equivalent to φ, we add many new variables to φ' that do not occur in φ, and instead of making φ' equivalent to φ we just want φ' to be satisfiable iff φ is satisfiable.

Here is an outline of the proof of **Lemma** 3, showing how to translate φ to φ' in polynomial time.

We start by removing each even length string of $\neg$'s in φ, so $\neg\neg$ does not occur in φ. For example $\neg\neg\neg x$ is changed to $\neg x$. Note that this modified form of φ is equivalent to the original φ.

Now we perform the following steps.

1. For each binary subformula α of φ (i.e. one of the form $(\beta \wedge \gamma)$ or $(\beta \vee \gamma)$) associate a new variable $x_\alpha = \ell_\alpha$. (The idea is that we will add clauses to φ' which will force the literal ℓ_α to have the same truth value as α.)
2. For each subformula $\alpha' = \neg\alpha$ of φ, let the literal $\ell_{\alpha'} = \overline{x_\alpha}$.
3. For each binary subformula $\alpha = (\gamma \circ \delta)$ of φ, where $\circ$ is either $\wedge$ or $\vee$, we define a 3CNF' formula D_α using the variable x_α and the literals ℓ_γ and ℓ_δ such that D_α is equivalent to the formula $(x_\alpha \leftrightarrow (\ell_\gamma \circ \ell_\delta))$.

4. Let $\varphi' = \ell_\varphi \wedge D_{\alpha_1} \wedge \cdots \wedge D_{\alpha_m}$ where $\alpha_1, \ldots, \alpha_m$ are the binary subformulas of φ.

We need to specify the 3CNF' formula D_α of step 3. Consider the case that $\circ$ is $\vee$, so α is $(\gamma \vee \delta)$.

Let D_α be $(x_\alpha \vee \overline{\ell_\gamma}) \wedge (x_\alpha \vee \overline{\ell_\delta}) \wedge (\overline{x_\alpha} \vee \ell_\gamma \vee \ell_\delta)$. It is easy to verify that D_α is equivalent to $(x_\alpha \leftrightarrow (\ell_\gamma \vee \ell_\delta))$

The case α is $(\gamma \wedge \delta)$ is similar.

Now it is clear that the above is a polytime algorithm converting φ to φ', and φ' is a CNF formula with at most 3 literals in each clause.

It remains to prove that φ is satisfiable iff φ' is satisfiable, where φ' is defined in step 4.

Suppose that the assignment τ' satisfies φ'. By induction on the depth of nesting of the subformulas α of φ, using the property of D_α stated in step 3 we see that $\tau'(\ell_\alpha) = \tau'(\alpha)$. In particular $\tau'(\ell_varphi) = \tau'(\varphi)$. Since $\tau'(\varphi') = 1$ and ℓ_φ is a conjunct of φ' it follows that $\tau'(\varphi) = \tau'(\ell_\varphi) = 1$.

Conversely, suppose that τ is an assignment to the variables of φ, and $\tau(\varphi) = 1$. Then we can extend τ to an assignment τ' to the variables of φ' by defining $\tau'(x_\alpha) = \tau(\alpha)$ for each binary subformula α of φ (and so $\tau'(\neg\alpha) = \tau'(\overline{x_\alpha})$). In particular $\tau'(\ell_\varphi) = \tau(\varphi) = 1$.

Finally, for each binary subformula α of φ, by the property of D_α given in Step 3 we see that $\tau'(D_\alpha) = 1$. Thus $\tau'(\varphi') = 1$, so φ' is satisfiable.

This completes the proof that 3SAT is NP-complete.

Now we use 3SAT to prove that other problems are NP-complete.

Definition: Let $G = (V, E)$ be an undirected graph. Then an *independent set* of G is a subset $V' \subseteq V$ of vertices such that if $u, v \in V'$ then (u, v) is **not** an edge of G.

Define the decision problem IND-SET as follows:

Input: $\langle G, k\rangle$, where G is an undirected graph and k is a positive integer.

Question: Does G have an independent set of size k?

Theorem: IND-SET in NP-complete.

Proof

Is is easy to see that IND-SET is in NP: The certificate for a YES instance $\langle G, k\rangle$ is an independent set V' of size k. It easy to check in polytime whether V' is an independent set of size k.

To show that IND-SET is NP-hard, by the Important Theorem at the beginning of the lecture, it suffices to show that 3SAT $\leqslant_p$ IND-SET.

Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be an instance of 3SAT, where each C_i is a clause $(\ell_{i1} \vee \ell_{i2} \vee \ell_{i3})$ where each ℓ_{ij} is a literal (i.e. a variable or a negated variable).

We transform φ to an instance $G_\varphi = (V, E), k_\varphi$ as follows.

Let $V = \{\langle i, j\rangle \mid 1 \leqslant i \leqslant m, 1 \leqslant j \leqslant 3\}$. Here we think if each node $\langle i, j\rangle$ as the *occurance* of the literal ℓ_{ij} in clause C_i. Note that if $i \neq i'$ then ℓ_{ij} and $\ell_{i'j'}$ might be the same literal, but they are distinct nodes in G.

We define the edge set $E = E_1 \cup E_2$. Here

$$E_1 = \{(\langle i,1\rangle, \langle i,2\rangle), (\langle i,1\rangle, \langle i,3\rangle), (\langle i,2\rangle, \langle i,3\rangle) \mid 1 \leqslant i \leqslant m\}$$

$$E_2 = \{(\langle i,j\rangle, \langle i',j'\rangle \mid \ell_{ij} = \overline{\ell_{i'j'}}, 1 \leqslant i, i' \leqslant m, 1 \leqslant j, j' \leqslant 3\}$$

Let $k_\varphi = m$

Clearly the transformation from φ to $\langle G_\varphi, k_\varphi\rangle$ can be carried out in polynomial time.

We prove correctness of the transformation as follows.

Suppose that φ is satisfiable. Let τ be an assignment that satisfies φ. Then τ satisfies each clause C_i. For each $i, 1 \leqslant i \leqslant m$ choose j_i so that τ satisfies the literal ℓ_{ij_i}. Then the set

$$V' = \{\langle i, j_i\rangle \mid 1 \leqslant i \leqslant m\}$$

is an independent set of G of size $m = k_\varphi$. This is because no two distinct elements of V' have the same i value, so they cannot be connected by an edge in E_1, and no two literals can be both true but complementary, so no edge in E_2 can connect them.

Conversely, let V' be an independent set of size $k_\varphi = m$. Then no two distinct elements $\langle i, j\rangle, \langle i', j'\rangle$ can have $i = i'$, because then they would be connected by an edge in E_1. Hence for $1 \leqslant j \leqslant m$ there is an element in V' of the form $\langle i, j\rangle$. We can define an assignment τ which makes every literal ℓ_{ij} such that $\langle i, j\rangle \in V'$ true, because no two of these literals are complementary because of the edges E_2. Thus τ satisfies every clause in φ, and hence it satisfies φ.

Recall that a subset $V' \subset V$ in an undirected graph $G = (V, E)$ is a *vertex cover* if every edge in E has at least one endpoint in V'.

Let

$$VC = \{\langle G, k\rangle \mid G \text{ is an undirected graph with a vertex cover of size } k\}$$

Theorem VC is NP-complete.

Clearly VC is in NP. To show VC is NP hard, it suffices to show that IND-SET $\leqslant_p$ VC. This is an easy reduction, using the following easy result:

Lemma: Let $V' \subseteq V$ be a subset of V in an undirected graph $G = (V, E)$. Then V' is a vertex cover of G iff $V - V'$ is an independent set in G.