CS46 Homework 10

This homework is due at 10pm Sunday, May 3. Submit on github as a file called **hw10.tex**.

For this homework, you will work with a partner or alone. It's ok to discuss approaches at a high level with other students, but most of your discussions should just be with your partner. Your partnership's write-up is your own: do not share it, and do not read other teams' write-ups. If you use any out-of-class references (anything except class notes, the textbook, or asking Lila), then you **must** cite these in your post-homework survey. Please refer to the course webpage or ask me any questions you have about this policy. or ask me any questions you have about this policy.

1. Recall that a **vertex cover** in a graph G is a subset of vertices where every edge of G has at least one endpoint in the subset.

   $$\textsc{VertexCover} = \{\langle G, k\rangle \mid G \text{ has a } k\text{-node vertex cover}\}$$

   Theorem 7.44 says that $\textsc{VertexCover}$ is NP-COMPLETE.

   An **independent set** in a graph G is a subset of vertices with no edges between them.

   $$\textsc{IndependentSet} = \{\langle G, k\rangle \mid G \text{ contains an independent set of } k \text{ vertices}\}$$

   We will show that $\textsc{IndependentSet}$ is NP-COMPLETE.

   (a) Prove that $\textsc{IndependentSet} \in \text{NP}$.

   (b) Prove that $\textsc{IndependentSet}$ is NP-HARD.
   (Hint: reduce from $\textsc{VertexCover}$. This is *not* the same direction you did in lab, but you might be able to use the same idea as the core of your reduction.)

2. Show that if P = NP, then every language $A \in \text{P}$ is NP-complete except $A = \emptyset$ and $A = \Sigma^*$.

3. A regular expression is ***-free** (pronounced "star-free") if it does not include any Kleene stars, so for example the regular expression "$(1 \cup 0)00$"' is *-free but "$0^*(1 \cup 11)$" is not *-free.

   Consider the language:

   $$L = \{\langle R_1, R_2\rangle \mid R_1 \text{ and } R_2 \text{ are *-free regular expressions and } L(R_1) \neq L(R_2)\}$$

   We will prove that L is NP-COMPLETE, using a reduction from $\textsc{Sat}$.

   (a) Show that $L \in \text{NP}$ by giving a deterministic polynomial-time verifier and describing the certificate strings it uses to check membership in L. (Make sure your verifier can't be fooled by a bad certificate!)

   (b) Given a formula ϕ in conjunctive normal form, write a regular expression that matches the language:

   $$\{w \mid w \text{ encodes a truth assignment for } \phi\}$$

   (c) Given a set of n literals $\{\alpha_1, \alpha_2, \ldots, \alpha_n\}$, consider the clause:

   $$c = \alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n$$

   Write a regular expression that matches the language:

   $$\{w \mid w \text{ encodes a truth assignment which does } \textit{not} \text{ satisfy } c\}$$

(d) Given a formula ϕ in conjunctive normal form, write a regular expression that matches the language:

$$\{w \mid w \text{ encodes a truth assignment which does } \textit{not} \text{ satisfy } \phi\}$$

(Use part (c).)

(e) Use parts (b) and (d) to give a polynomial-time reduction from SAT to L. Conclude that L is NP-COMPLETE. (**Hint:** If you want to, you may assume throughout this problem that formulas in SAT are always in conjunctive normal form.)

4. **(extra credit)**

$$2\text{-SAT} = \left\{ \langle\varphi\rangle \;\middle|\; \begin{array}{l} \varphi \text{ is a satisfiable formula in conjunctive normal form} \\ \text{with exactly two literals per clause} \end{array} \right\}$$

We will prove that 2-SAT $\in$ P.

Any clause $(x \vee y)$ with two literals can be thought of as two implications $\overline{x} \Rightarrow y$ and $\overline{y} \Rightarrow x$. The clause $(x \vee x)$ can be thought of as $\overline{x} \Rightarrow x$. If we then consider $x \Rightarrow y$ as a directed edge from vertex x to vertex y, we can construct an "implication graph" from any 2-CNF formula φ.

(a) Show that a 2-CNF formula is unsatisfiable if and only if there is a variable x such that in the implication graph, there is a path from x to $\overline{x}$ and from $\overline{x}$ to x.

(b) Design an algorithm based on this fact to show that 2-SAT $\in$ P.

5. (**extra credit**) Show that if P $\cap$ NP-HARD $\neq \emptyset$, then P $=$ NP.