CSE 200 - Computability and Complexity
Homework 2
Due Wednesday, November 1, 11:59pm

Instructions: You may work individually or in a team of 2 people. You may switch teams for different assignments. Please ensure your name(s) and PID(s) are clearly visible on the first page of your submission, and then upload the PDF to Gradescope. If working in a group, submit only one submission per group: one partner uploads the submission through their Gradescope account and then adds the other group member to the Gradescope submission by selecting their name in the "Add Group Members" dialog box. You will need to re-add your group member every time you resubmit a new version of your assignment.

It is highly recommended (though not required) that you type your answers. It is your responsibility to make any handwriting clear and legible for grading. For ease of grading, please start each new problem on a separate page.

Problems:

1. **NP-complete?** For any undirected graph G, let $\omega(G)$ denote its *clique number*—i.e., the number of vertices in the largest clique of G. Consider the following languages:

$$\textsc{Clique} := \{\langle G, k\rangle \mid \omega(G) \geq k\}$$
$$\textsc{BiggerClique} := \{\langle G, H\rangle \mid \omega(G) > \omega(H)\}$$

In class, we showed that Clique is NP-complete.

(a) Prove that BiggerClique is NP-hard by reduction from Clique. That is, show $\textsc{Clique} \leq_p \textsc{BiggerClique}$.

(b) Give some intuition for why BiggerClique might not be NP-complete. Your answer can be informal.

(c) Suppose there is a special kind of algorithm that can solve Clique in constant time. Such an algorithm is usually called an "oracle," and running the oracle on some input is called "querying" the oracle. Show that there is a polynomial-time algorithm that can solve BiggerClique when it has the ability to query the Clique oracle polynomially many times.

The class of languages solvable by algorithms of the type in (c) is usually denoted as P^{NP} since there is polynomial-time machine with access to an oracle that can solve an NP-complete problem (in this case, Clique). In other words, in this problem, we have shown that BiggerClique is NP-hard and that $\textsc{BiggerClique} \in \mathsf{P}^{\mathsf{NP}}$.

As it turns out, BiggerClique is actually complete for the complexity class $\mathsf{P}^{\mathsf{NP}[\log]}$, that is, the class of languages which can be solved by a polynomial time machine that can make *logarithmically* many queries to an NP oracle. As a challenge (not graded), improve your algorithm in (c) to only use $O(\log n)$ queries, where n is the maximum number vertices in the input graphs G and H.

2. **A different kind of constraint satisfaction problem** Let's define a variant of 3SAT called 3NAE-SAT where the NAE stands for "not all equal." Recall that in 3SAT, we want each clause of our Boolean formula to contain at least one true literal. In 3NAE-SAT, we require that within a given clause the literals are neither all false *nor* all true. For example, the clause $\text{NAE}(x_1, x_2, \overline{x_3})$ is satisfied under an assignment where x_1, x_2, x_3 are all true, but not satisfied under an assignment where x_1 and x_2 are true, but x_3 is false. A 3NAE-SAT formula is satisfiable if all of its clauses are satisfiable. Once again, the "3" means that each clause has at most 3 literals. Let

$$\text{3NAE-SAT} = \{\varphi \mid \varphi \text{ is a satisfiable 3NAE-SAT formula}\}.$$

We will show that 3NAE-SAT is NP-hard by a reduction from 3SAT. It will be simpler two break the reduction into two steps by using an intermediate problem. First, we need to show that this approach is consistent with our notion of reduction.

(a) Let A, B, C be *any* languages. Show that if $A \leq_p B$ and $B \leq_p C$, then $A \leq_p C$. In other words, polynomial-time Karp reductions are transitive.

We now define the intermediate language 3NAE-SAT-CONST which is the same as the language 3NAE-SAT except the clauses can also have the literals T and F which are *always* true and false, respectively. Recall that a usual literal corresponding to some variable can be either true or false.

(b) Prove that 3SAT $\leq_p$ 3NAE-SAT-CONST.

(c) Prove that 3NAE-SAT-CONST $\leq_p$ 3NAE-SAT.

Notice that combining (b) and (c) with (a) shows 3NAE-SAT is NP-hard.

3. **Complexity consequences** Prove that if NP = PSPACE then coNP = NP.

4. **Bootstrapping NP algorithms for the polynomial hierarchy**

Assume that $\text{NP} \subseteq \cup_{c \geq 1} \text{TIME}(n^{c \log n})$. Show that for every $i \in \mathbb{N}$, there exists a $k \in \mathbb{N}$ such that $\Sigma_i \subseteq \cup_{c \geq 1} \text{TIME}(n^{c(\log n)^k})$.

5. **Small space algorithms**

Define the language $\textsc{Sum} := \{x\#y\#z \mid x, y, z \in \{0,1\}^* \text{ and } x + y = z\}$. The interpretation of the language is that x, y, and z are integers written in binary and we would like to know if $x + y$ is equal to z. The "#" symbol is used to separate the numbers. Prove that $\textsc{Sum} \in \text{L}$. In other words, Sum is computable in logarithmic space.