CSE 531: Computability and Complexity — Autumn 2004
Problem Set #3 (On P and NP) — Instructor: Venkatesan Guruswami
Due on Wednesday, **November 16, 2004** in class.

Instructions: Same as Problem Set 1.

---

1. Assuming P = NP, describe a polynomial time algorithm that, given a Boolean formula ϕ, actually produces a satisfying assignment for ϕ if it is satisfiable.
2. Define MAJ-3SAT to be the language of 3-CNF formulas ϕ for which there is an assignment to the variables which satisfies *at least two* out of the three literals in *every* clause. For example, $(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee \bar{z}) \wedge (y \vee z \vee w)$ is in MAJ-3SAT since the assignment $x = 0$, $y = 1$, $z = 0$ and $w = 1$ satisfies at least two literals in each clause. Prove that MAJ-3SAT is in P. (Hint: Look at Problem 7.37 in Sipser's book.)
3. Let $f : \Sigma^* \to \Sigma^*$ be a function such that $|f(w)| \leq p(|w|)$ for some polynomial $p(n)$. Describe a language A_f based on f (but not necessarily over the alphabet Σ) which satisfies the following: $A_f \in$ P if and only if f is computable in polynomial time.
4. Problem 7.34, Sipser's book. (3COLOR is NP-complete)
5. A bipartite graph is a graph whose vertices can be partitioned into two disjoint parts each of which is an independent set. Formally a bipartite graph $H = (X, Y, E)$ has vertex set $X \cup Y$ for disjoint sets X, Y and each edge in its edge set E has one endpoint in X and one in Y. A k-bipartite clique of H is a pair of subsets $S \subseteq X$ and $T \subseteq Y$ with $|S| = |T| = k$ such that $(s, t) \in E$ for each $s \in S$ and $t \in T$ (informally all "cross-edges" exist between S and T). Define the language

   $$\text{BIPARTITE-CLIQUE} = \{\langle H, k\rangle \mid H \text{ is a bipartite graph that has a } k\text{-bipartite clique}\} .$$

   Prove that BIPARTITE-CLIQUE is NP-complete.
   (Hint: This problem is reasonably tricky and the obvious reduction from CLIQUE that one would be tempted to try does *not* work.)
6. Define the Boolean function $\text{MAJORITY}_n : \{0,1\}^n \to \{0,1\}$ as:

   $$\text{MAJORITY}_n(x_1, x_2, \ldots, x_n) = 1 \text{ if and only if } \sum_{i=1}^{n} x_i \geq n/2 .$$

   Thus MAJORITY_n returns the majority vote of its inputs. Show that MAJORITY_n can be computed with $O(n)$ size Boolean circuits (with NOT gates and fan-in 2 AND and OR gates). (Hint: Divide and Conquer)
7. * **(Optional Problem for Fun/Extra Credit)** Suppose P $\neq$ NP. Then there are languages in P and languages that are NP-complete, and these sets are disjoint. Is there anything else in NP? A 1975 theorem due to our very own Richard Ladner shows that the answer to this question is yes. In this exercise, try to prove Ladner's theorem, which asserts the following: If P $\neq$ NP, then there is a language in NP that neither belongs to P nor is NP-complete.

   (**Disclaimer:** This is obviously a difficult problem for which good hints are warranted. At the time of this writing, however, I did not fully figure out how to turn the proof idea I've

attempted to sketch below into a complete proof. You can discuss this with me further in person, or better still get suggestions from the horse's mouth by asking Ladner (not that I've checked that this would be okay with him)!

One way to prove this is, loosely put, by blowing holes in SAT (but feel free to ponder about other attacks). Define $A = \text{SAT} \cap \{w \mid f(|w|) \text{ is even}\}$ where $f : \mathbb{N} \to \mathbb{N}$ is a *very slowly growing* function on natural numbers (think $\log^* n$ for intuition). Thus A has long, well-separated regions that are identical to SAT. If $f(n)$ is computable in time polynomial in n, then clearly A is in NP. To see why we might hope for $A \notin \text{P}$, note that A has large regions that look like SAT (namely for inputs of lengths n for which $f(n)$ is even). To see why we might hope that A isn't NP-complete, suppose there is a mapping reduction g from SAT to A. Pick $x \in \text{SAT} - A$. We must have $g(x) \in A$, and since $x \notin A$, $g(x)$ must be in some other "region" of A, and thus must (typically) be exponentially longer or shorter than x. g being a polytime reduction the former is not possible, and thus $g(x)$ must be exponentially smaller than x. Since A can be decided in exponential time, this means we can tell if $x \in \text{SAT}$ in polynomial time for a large fraction of x, something we don't expect.

Note that the above is just hope and not a correct proof since there is no guarantee on how hard instances of SAT are distributed just because $\text{P} \neq \text{NP}$.

To get a proof out of this high level idea, we have to make use of diagonalization against all polytime TMs M_i and polytime reductions R_i from SAT. The machine computing f will do this. On input n it will run for fixed number of steps, say n, computing $f(0), f(1), \ldots$ in sequence. If last value computed is k it will try to diagonalize (running for another n steps, say) against M_i if $k = 2i$ and against R_i if $k = 2i + 1$, and if it manages to do so it will output $k+1$ as $f(n)$, else $f(n) = k$. The key point is that eventually n will be large enough to accomplish this diagonalization within the time bound. The proof is completed by deriving the consequence $\text{P} = \text{NP}$ under the assumption that M_i solves A or R_i reduces SAT to A for any i.)