Harvard University Extension School
Computer Science E-207

Problem Set 10

Due Friday, December 7, 2012 at 11:59 PM Eastern Time.
Submit your solutions in a single PDF called lastname+ps10.pdf emailed to cscie207@seas.harvard.edu.
LATE PROBLEM SETS WILL NOT BE ACCEPTED.

Problem set by **ENTER YOUR NAME HERE**

Collaboration Statement: **FILL IN YOUR COLLABORATION STATEMENT HERE
(See the syllabus for information)**

See syllabus for collaboration policy.

PROBLEM 1 (10 points)

Recall the definition of a homomorphism from Problem Set 2. We define a *non-erasing homomorphism* ψ to be a homomorphism such that $\psi(s) = \varepsilon$ if and only if $s = \varepsilon$. Prove that **NP** is closed under non-erasing homomorphisms. (*Hint:* Use characterization of **NP** in terms of polynomial-time verifiers.)

PROBLEM 2 (10 points)

Let DoubleSAT $= \{\langle\phi\rangle : \phi$ is a boolean formula with at least two satisfying assignments$\}$. Show that DoubleSAT is **NP**-complete.

PROBLEM 3 (10 + 5 points)

In the last problem of Problem Set 9, you were consulted by a business for advice on throwing the most fun party. You were told that a party would be no fun if any employee and his or her direct supervisor were both there.

Over the next ten years, the business underwent numerous personnel changes, e.g. the VP of Engineering became the President, and Engineer A was promoted to VP of Engineering. It is now time for another party. Remembering how successful it was last time, they come to you for help again. This time, however, things are a little more complicated: a party is no fun if someone meets any of the direct supervisors he or she has *ever* had.

Suppose you are given $(e_0, f_0), (e_1, f_1), \ldots, (e_n, f_n)$, where employee e_i has fun index f_i, and a directed graph G where the vertices are employees and there is an edge from e_i to e_j if e_i was ever the direct supervisor of e_j.

(A) Show that the decision problem $\{\langle(e_0, f_0), \ldots, (e_n, f_n), G, k\rangle :$ there is a party with fun index at least $k\}$ is **NP**-complete. (*Hint*: Reduce from Clique.)

(B) Show that if $\mathbf{P} \neq \mathbf{NP}$, there is no polynomial-time algorithm that given given any $(e_0, f_0), \ldots, (e_n, f_n)$ and G computes the most fun party.

PROBLEM 4 (5+5+10 points)

We have seen in class that **P** and **NP** are classes of decision problems. Now consider a different type of problem: A *search problem* is a mapping $S : \Sigma^* \to P(\Delta^*)$ from strings ("instances") to sets of strings ("valid solutions"). An algorithm M *solves* a search problem S if on every input $x \in \Sigma^*$ such that $S(x) \neq \emptyset$, M outputs some solution in $S(x)$.

An **NP** *search problem* is a search problem S for which every solution $y \in S(x)$ is of length at most polynomial in $|x|$, and for which there exists a polynomial-time verifier V that accepts $\langle x, y \rangle$ if and only if $y \in S(x)$.

(A) Show that the SAT-SEARCH problem "given a satisfiable boolean formula φ, find a satisfying assignment" is an **NP** search problem.

(B) Show that the FACTORING problem "given a number N (in binary), find its prime factorization" is an **NP** search problem. (You may use the fact that PRIMES $\in$ **P**.)

(C) Prove that SAT $\in$ **P** if and only if SAT-SEARCH can be solved in polynomial time. (Hint: think one variable at a time.)

PROBLEM 5 ((Challenge!) 1 points)

Write down an *explicit* algorithm with the following property. If **P** = **NP**, then your algorithm runs in polynomial time and solves TSP on all sufficiently long inputs. By "explicit" we mean that your algorithm should not contain or reference any unknown constant.

(*Hint*: If **P** = **NP**, then there exists a Turing machine M and a constant c such that M solves the Traveling Salesman Problem TSP in time at most n^c. Your algorithm should find one such pair (M, c) by enumeration. Note that, on input x of length n, while you may not be able to test in time polynomial in n whether a TM M correctly solves TSP on input x, in time n^c, you *can* check that this property holds for all inputs of length significantly smaller than n, by comparing M against any exponential-time algorithm for TSP.)