CS 181d, Complexity Theory
Fall 2012
Homework 3
Due Wednesday, September 26, 11:59 PM

1. **[20 Points] NP and NP-completeness!**

   (a) Prove that if a language L is in NP then it is decided by some deterministic TM in exponential time. Recall that a function is said to be exponential if it is in $O(2^{p(n)})$ for some polynomial $p(n)$. Be careful in your analysis here, deriving an actual exponential bound for the deterministic TM based on the polynomial for the non-deterministic TM. In particular, let the nondeterministic TM for L be $M_L = (Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}})$ where Q is finite the state set, Σ is the finite input alphabet, Γ is the finite tape alphabet, δ is the non-deterministic transition function (let b denote the maximum number of nondeterministic choices made by the δ function), and the three special states are the initial state, accepting state, and rejecting state. Let $q(n)$ be the polynomial running time of M_L. The polynomial that you derive for the deterministic simulator will be a function of some of the parameters in M_L and $q(n)$. Explain your derivation.

   (b) Let

   $$L_1 = \{< M >, w, 0^t | \ M \text{ is nondeterministic TM that accepts string } w \text{ within } t \text{ time steps}\}$$

   Prove that language L_1 is NP-complete by explicitly showing a reduction from *every* problem in NP to L_1. (This is what Cook's Theorem does but it is not the "usual way" of doing an NP-completeness proof; normally we would perform a reduction from a single known NP-complete problem. Nonetheless, here I would like you to perform a proof from every possible language in NP because it's not too often that we get to do that!)

2. **[20 Points] Cook's Theorem Revisited!** In class we examined Cook's Theorem that SATISFIABILITY is NP-complete. The proof begins by first observing that for any language L in NP there exists a non-deterministic Turing Machine M and a polynomial $p(n)$ such that on input string w, M runs in time $p(|w|)$ and determines whether or not w is in the language. Then, we showed that for any string w of length n we can build (in polynomial time) an instance of the SATISFIABILITY problem that is satisfiable if and only if M accepts w in time $p(n)$.

The reduction involved 6 groups of clauses called $G_1, \ldots, G_6$. Looking back at your notes, you will recall that G_5 attempted to enforce the constraint that the contents of the tape at time $i+1$ follow from the contents at time i according to the δ function of the Turing Machine M. We didn't quite finish the details of that clause group.

(a) Using the variable notation that we defined in class ($Q_{i,k}, H_{i,j}$, and $S_{i,j,k}$) describe *all* of the clauses needed for clause group G_5.

(b) Derive an upper-bound on the **total** number of variables and clauses in the SAT expression as a function of $p(n), r$, and g. This should end up being polynomial in n or our reduction is not polynomial-time.

3. **[20 Points] Cliques Revisited!**

Do this problem before the next one. This problem builds some technical tools that we'll need on the next problem. First, recall that a language L is NP-complete if it is in NP and every problem L' in NP is polynomial-time reducible to L (this latter condition is called "NP-hardness").

(a) Recall that the CLIQUE decision problem goes like this: Given an undirected graph G and a positive integer k, is there a clique (a fully connected subgraph) of size at least k in G. Prove that CLIQUE is NP-complete via a reduction from 3SAT. (You may have seen a reduction from VERTEX COVER in another course, but you're asked here to describe a different reduction.) In the interest of brevity, describe your reduction precisely but keep the "iff" part of the proof brief – a few sentence for each direction of the proof should suffice.

(b) Let G and H be two undirected graphs. The product[1] of these graphs $G \cdot H$ is a new graph defined as follows: Replace each vertex of G with a full copy of H. Then if two vertices g and g' in G are adjacent, then every vertex in the full copy of H that replaced g is connected to every vertex in the full copy of H that replaced g'. More precisely, the product $G \cdot H$ has a vertex (g, h) for each vertex $g \in V(G)$ and $h \in V(H)$ where (g, h) is adjacent to (g', h') if either $g = g'$ and h is adjacent to h' OR g is adjacent to g'. Finally, let $m(G)$ denote the size of the largest clique in graph G. **Prove** that $m(G)m(H) = m(G \cdot H)$.

[1]There are actually 2^8 natural ways to define the product of two graphs. See `mathworld.wolfram.com/GraphProduct.html` for details. We are using what MathWorld calls "graph lexicographic product".

4. **[40 Points] DP: A New Complexity Class!** In this problem you will investigate a "new" complexity class called DP. DP is the collection of all languages L such that L can be expressed as $A - B$ where A and B are both in NP. Recall that $A - B$ means the collection of items in A and not in B.

   (a) First, show that DP contains all of NP and also all of co-NP. Aha, so DP includes some pretty hard problems!

   (b) A decision problem L is defined to be *DP-complete* if it satisfies **two** conditions: L is in DP *and* every problem in DP is polynomial-time reducible to L (this latter condition is called "DP-hardness"). Now, consider the language:

   $$L = \{< G_1, k_1, G_2, k_2 > \mid \ G_1 \text{ has a clique of size } k_1 \text{ and } G_2 \text{ doesn't have a clique of size } k_2\}$$

   Prove that L is DP-complete.

   (c) Now, modify your reduction above so that when your reduction constructs an instance G_1, k_1, G_2, k_2 of L, this instance always has the property that $k_1 > k_2$, there is a clique of size $k_1 - 1$ in G_1, and there is a clique of size $k_2 - 1$ in G_2.

   (d) Finally, consider the problem MAX-CLIQUE defined as the language:

   $$\{< G, k > \mid \text{the largest clique in } G \text{ is of size exactly } k\}$$

   Prove that MAX-CLIQUE is DP-complete. You may find it useful to use the earlier results of this problem and the previous one.

5. **[20 Point OPTIONAL Bonus Problem] Primality Testing is in NP!** Usually, proving that a problem is in NP is pretty easy. Sometimes though, it's actually quite interesting and challenging! This problem explores one such problem.

   PRIMES is the language of all prime numbers in binary. Prove that PRIMES is in NP. *Hint:* The only number theory that you need to do this is the following result which you may use without proof: "A number $p > 1$ is prime iff there exists a number $1 < r < p$ such that $r^{p-1} = 1 \bmod p$ and for all prime divisors q of $p - 1$, $r^{\frac{p-1}{q}} \neq 1 \bmod p$.