CS 181d, Complexity Theory
Fall 2012
Homework 5
Due Wednesday, October 10, at 11:59 PM

This is the penultimate homework assignment! The last assignment will be somewhat shorter and will be due on Tuesday, October 16 in class in order to allow you the rest of the week to complete the take-home final exam.

1. **[40 Points] Finishing Up the NL = co-NL Proof!** Your task is to write out *the entire proof* that NL = co-NL. Our approach was to show that co-PATH is in NL.

   In class we started the proof that co-PATH is in NL. The funny thing that we assumed in that proof is that the non-deterministic log space TM for co-PATH was given not just $< G, s, t >$, but also r, the number of vertices reachable from s in G. In order to complete the proof, you'll need to show that r can be computed by the non-deterministic log space TM by itself before launching into the step that we described in class. In other words, the non-deterministic log space TM for co-PATH will construct a tree that has two "parts". The "upper part" will compute r and the lower part will use r to determine that there is no path from s to t.

   (a) Professor I. Lai of P.I.T. has proposed the following approach for computing r: "It's easy!" explains Professor Lai nonchalantly. "You simply have your non-deterministic machine first use non-determinism to guess every possible value of r and place this guess in a counter. Writing r down takes log space. Now, each guess of the counter's value appears in a different subtree of our non-deterministic tree. In each subtree, we now verify that guess as follows: Use non-determinism again to sequentially guess the vertices that we believe to be reachable from s. For each such guessed vertex we first verify that we can reach that vertex (in the same way that we checked if s can reach t in our proof that PATH is in NL) and, if the answer is "yes", we decrement the counter. If we get to a point that the counter is zero, we conclude that we guessed the correct value of r and we now have r to use in the rest of the construction!" Now "r-gue" that Professor Lai is wrong!

   (b) Here's an approach that does work. Let S_i denote the set of vertices that are reachable from s and at distance i or less from s. If there are ℓ vertices in the graph, we'd like to find S_ℓ since this will be the set of all vertices reachable from s. The size of S_ℓ is the value r that we need. Unfortunately, even one such set may be too large for us to store (in its entirety) on tape since we are log-space bounded! So, instead we'll compute the *sizes* of these sets where $r_i = |S_i|$ and we're seeking $r = r_\ell$.

   We know that $r_0 = 1$ since there is exactly one vertex reachable from s using 0 or fewer edges (it's s itself!). Now, we'll try to iteratively compute r_1 and then

r_2, etc. Each r_i will be computed based on the previous r_{i-1} value that we've computed. How do we do that? Notice that the vertices in set S_i are those that are reachable from s in i or fewer hops. Thus, they are all of the vertices in S_{i-1} and all of the vertices reachable in one hop from S_{i-1}. Another easier and more convenient way of characterizing this, is that a vertex is in S_i if it is either s or is reachable from some vertex in S_{i-1} in one hop (notice that this includes the vertices that are at distance i from s as well as those that are at distance less than i from s).

So now what? Again, we're assuming that we've computed r_{i-1} in the previous iteration. We'll now use non-determinism to guess and verify the vertices in S_{i-1} (but without ever writing all of them down at once since that takes too much space) and use that to compute r_i.

In your write-up first describe the process by which r_i is computed from r_{i-1}, then put this together to explain how $r = r_k$ is computed, and then briefly explain how this fits together with what we showed in class to conclude that co-PATH is in NL.

2. **[30 Points] 2SAT is NL-complete!** In class, we briefly reviewed the proof that 2SAT is in P (which we did rigorously in a homework assignment in CS 140). You may assume that result here without reproving it. Now, you'll show that 2SAT is NL-complete. Amazing, but true!

   (a) In class we showed that PATH is NL-complete. Consider the very closely-related problem, DAG-PATH which is the language of triples G, s, t where G is a directed acyclic graph (DAG) and there is a path from s to t in G. Briefly explain why our proof that PATH is NL-complete actually proved that DAG-PATH is NL-complete.

   (b) Show that 2SAT is in NL.

   (c) Now prove that 2SAT is NL-hard. Keep in mind that NL = coNL; this may be helpful!

   (d) Conclude that 2SAT is NL-complete.

3. **[30 Points] The Strange Complexity Class P/log!**

   Here's an interesting "new" complexity class: It's called P/log. This class comprises all languages (unary, binary, or over any other alphabet) with the following property: There exists a polynomial time deterministic verifier V, a constant k, and an omnipotent prover P; On input of length n, the prover P sends V an advice string of length at most $k \log n$. This advice string is the same for all inputs of length n. In other words, the advice depends only on the length of the input and not on the particular symbols in the input. The verifier V then computes deterministically in time polynomial in n and chooses to accept or reject its input.

Notice that we are *not* saying that the verifier accepts its input if there exists some advice string that would make it accept. Rather, we are saying that there exists a specific "trusted" prover that works with this verifier to issue just the right advice string. In other words, just because some advice string of length $O(\log n)$ makes the verifier accept does not mean that it correctly decided whether its input string is in the language. This is different from the class NP, where the "prover" that provides our certificate is not trusted; in NP the verifier verifies the certificate and can't be tricked by an evil prover into accepting an input string that is not in the language.

(a) Show that the the class P/log contains undecidable languages. Aha, so short advice strings can be very useful! Funky!

(b) Since P/log contains undecidable languages, we might be tempted to believe that it contains all of NP. However, in part (c) below you'll show the amazing fact that if SATISFIABILITY is in P/log then P = NP. **Before** doing this though, prove that if we had a polynomial time decider for the SATISFIABILITY problem then we could construct a polynomial time algorithm for finding an actual satisfying assignment (a "valuation") for any instance of SATISFIABILITY if one exists. This idea is called "self reducibility" and you may recall that we talked about this in the "Algorithms" course. (*Recall:* SATISFIABILITY is the general form of 3SAT: The expression is in conjunctive normal form but there can be any number of literals per clause.)

(c) **OK, now show that** if SATISFIABILITY is in P/log then P = NP. *Note:* Without loss of generality, you may assume – if you wish – that an instance of SATISFIABILITY whose ending is malformed, that is it ends with an "AND" symbol or the last clause is only partially formed (i.e. the last clause is malformed because it ends with an "OR" symbol or is missing a closing parenthesis, etc.), is interpreted as an instance with the malformed ending omitted. This can just make your "housekeeping" a bit simpler. This assumption is actually not necessary.