COMS 3261, Computer Science Theory (Fall 2023): Assignment 4 Solutions

Problems

1. (5 points) Prove that the class P is closed under complement.

   Solution: The proof is analogous to the argument showing that the class of decidable languages is closed under complement. Let L be a language in P, and let A be a TM that decides L and runs in polynomial time. Then we can also decide the complement of L in polynomial time by simulating A on an input w and if A accepts w then halt and reject w and if A rejects w then halt and accept w.

2. (10 points) Let FACTOR be the function that takes as input a natural number x in decimal notation, and outputs the prime factorization of x, also in decimal. For example, if the input is 13, then the output would be 13, and if the input is 12, the output would be 2,2,3. (You can output the prime factors in any order.) Prove that if $P = NP$ then there is a polynomial-time algorithm for FACTOR. Note that NP is a class of *languages* (or decision problems), so factoring is not in NP. Therefore, you should give an algorithm for FACTOR, assuming that every language in NP is solvable in polynomial-time.

   Solution: For this problem we will also assume that PRIME is in P since as discussed in class it is known to be in P (and even if not, PRIME is in NP so under the assumption P=NP we get that PRIME is in P).

   Let EXISTS-PRIME-FACTOR be the language consisting of triples (x, y_1, y_2) such that $1 < y_1 < y_2 < x$ and such that there exists a prime number z, $y_1 \leq z \leq y_2$, such that z divides x. First it is easy to see that EXISTS-PRIME-FACTOR is in NP: the verifier V on input $((x, y_1, y_2), z)$ checks whether (i) z is a prime in the interval (y_1, y_2) and (ii) that z divides x, and if so then V accepts and otherwise V rejects. Therefore assuming $P = NP$, EXISTS-PRIME-FACTOR is in P.

   Let FIND-PRIME-FACTOR take as input x, and output either x if x is prime, or it outputs a prime factor $y < x$ of x. We first describe an algorithm to solve FIND-PRIME-FACTOR using EXISTS-PRIME-FACTOR as a subroutine. FIND-PRIME-FACTOR on input x: If PRIME(x) accepts (so x is prime), then halt and output x. Otherwise, call EXISTS-PRIME-FACTOR$(x, 2, x/2)$. If it accepts then we know that there is a prime factor x' in the interval $(2, x/2)$. In this case recursively call EXISTS-PRIME-FACTOR on the interval $(2, x/4)$. Otherwise we know that there is a prime factor factor in the interval $(x/4, x/2)$ so we recursively call EXISTS-PRIME-FACTOR on the interval $(x/4, x/2)$. When the recursive calls finish, we will have found some x' that is a nontrivial prime factor of x.

   The above algorithm runs in polynomial time assuming that EXISTS-PRIME-FACTOR runs in polynomial time in $\log x$. (Note that the length of the input is $O(\log x)$. This is because each time we call EXISTS-PRIME-FACTOR, we are recursing on an interval of that is half as big as the previous interval, so after $\log x$ iterations, we are guaranteed to have found a prime factor.

   Lastly we will show how to solve FACTOR given the above algorithm for FIND-PRIME-FACTOR. On input x to FACTOR, we repeatedly call FIND-PRIME-FACTOR.

We start by calling it on x. If it returns x then x is prime so we are done. Otherwise, if it returns a prime factor $x' < x$, then we call FIND-PRIME-FACTOR on x/x'. Since $x' \geq 2$, each time we call FIND-PRIME-FACTOR, it is called on a number that is at most half of what it was in the previous call, and therefore we call FIND-PRIME-FACTOR at most $\log x$ times. Since each call is polynomial time in $\log x$, the total runtime to solve FACTOR is polynomial in $\log x$, (assuming that $P = NP$)

3. (10 points) State True or False for each question and give a one sentence justifcation of your answer.

   (a.) For any language L, if L is in NP, then the complement of L is also in NP.

   Solution: If for every NP language L, its complement is also in NP then P is equal to NP intersect coNP. This is currently unknown and we expect that the answer is no. Therefore the statement is not known to be true or false but it is likely False.

   (b.) For any language L, if L is NP-complete, then the complement of L is NP-hard.

   Solution: This is true. Let L be an NP-complete language. Then we will show that if the complement of L is in P then $P = NP$. Assume that the complement of L is in P. Then L itself is also in P since by problem 1 above. Then since L is NP-complete this implies that if L is in P then every other language in NP is also in P.

4. (12 points) A Boolean formula is in CNF form if it is the logical AND of a set of clauses, where each clause is the OR of a set of literals, and each literal is a variable or its negation. A CNF formula f over variables $x_1, \ldots, x_n$ is *satisfiable* if there exists a Boolean assignment $\alpha \in \{0,1\}^n$ to the variables such that $f(\alpha)$ evaluates to true.

   (a.) Is the following formula satisfiable? Prove your answer.

   $$f = (x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_4) \wedge (x_4)$$

   Solution: This formula is satisfiable. A satisfying assignment is: $\alpha = 0111$.

   (b.) Suppose that you are given a Boolean formula f in CNF form over n Boolean variables, where each clause contains at most 3 literals. Prove that f can be converted into another CNF formula, g, such that: (i) g has exactly 3 distinct literals per clause; (ii) the number of clauses in g is polynomial in n; and (iii) g is satisfiable if and only if f is satisfiable. (Note that the variables of g can include new variables in addition to the variables of f.)

   Solution: Let f be a CNF formula where each clause contains at most 3 literals, and let the underlying variables of f be $x_1, \ldots, x_n$. We will replace each clause C_i in f by a new conjunction of zero or more equivalent clauses as follows:

   (1) if C_i contains three distinct literals then let $C_i' = C_i$.
   (2) Otherwise let $C_i = (l_1 \vee l_2)$ where l_1, l_2 are distinct literals. Then we replace C_i by $C_i' = (l_1 \vee l_2 \vee y_i) \wedge (l_1 \vee l_2 \vee \neg y_i)$, where y_i is a new variable.
   (3) Otherwise C_i contains only one literal, so $C_i = (l_1)$. Then we replace C_i by $C_i' = (l_1 \vee y_i \vee z_i) \wedge (l_1 \vee \neg y_i \vee z_i) \wedge (l_1 \vee y_i \vee \neg z_i) \wedge (l_1 \vee \neg y_i \neg z_i)$, where y_i, z_i are new variables.

Let g be the conjunction of all of C_i's. It is left to prove that f is satisfiable if and only if g is satisfiable. Given an assignment to the original x variables that satisfies f, when we plug this assignment into g it will also be satisfiable since all we did was add variables to some original clauses which were already satisfied. On the other hand, given an assignment to all of the x, y, z variables of g, we claim that if this assignment satisfies g, then the x-part of the assignment will satisfy f. This holds because the only way to satisfy C_i' is to satisfy C_i itself. Finally the reduction from f to g is polynomial-time in the length of f since we convert each clause C_i one-by-one and each conversion is linear-time.

5. (15 points) The problem k-minSAT takes as input a CNF formula f over n Boolean variables $x_1, \ldots, x_n$ and a number $k \leq n$, and accepts if and only if f has a satisfying assignment with at most k 1's.

   (a.) Prove for any constant k, k-minSAT is in P. (e.g., 3-minSAT is in P.)

   Solution: For constant k, the number of assignments to $x_1, \ldots, x_n$ with at most k 1's is: $\sum_{i=0}^{k} \binom{n}{k}$ which is $O(n^k)$ when k is constant, and thus is polynomial in n. Thus we can solve k-minSAT on f by enumerating over all assignments with at most k 1's (they can be enumerated in polynomial-time) and then check for each one whether or not it satisfies f; if at least one satisfies f then we accept and otherwise we reject.

   (b.) Prove that k-minSAT is NP-complete.

   Solution: To see that k-minSAT is NP-complete, we first note that k-minSAT is in NP since a nondeterministic polytime can guess an assignment with at most k 1's and verify whether or not it satisfies f. Thus it is left to prove that it is NP-hard. We will prove this by showing that 3SAT polynomial-time reduces to k-minSAT. Given an input f to 3SAT, it is in 3SAT if and only if iff $(f, k = n)$ is in k-minSAT. Therefore since 3SAT is NP-hard this implies that k-minSAT is also NP-hard.