Algorithms – CMSC-27200
http://alg15.cs.uchicago.edu Homework set #8.
Posted 3-1. Problems 8.1(c2) and 8.2 updated 3-2 12:30pm.
Due ~~Wednesday~~ Friday, March 6, 2015

Read the homework instructions on the website. The instructions that follow here are only an incomplete summary.

Hand in your solutions to problems marked "HW." Do not hand in problems marked "DO." If you hand in homework marked **"XC" (extra credit)**, do so on **separate and separately stapled sheets, please**. PRINT YOUR NAME and SECTION NUMBER ON EVERY SHEET you submit. **Use LaTeX to typeset your solutions**. Hand in your solutions on paper, do not email.

When writing pseudocode, **explain the meaning of your variables.** Use comments to explain what is happening in each line. Also, give a short explanation of the idea behind the algorithm. **Unless otherwise stated, describe your algorithms in pseudocode. Elegance of your code matters.**

Carefully study the policy (stated on the website) on collaboration, internet use, and academic integrity. **State collaborations and sources both in your paper and in email to the instructors.**

8.1 **(Configuration space)** Onaedo designs an algorithm that involves an array of n variables, $x_1, \ldots, x_n$, taking decimal digits as values ($0 \leq x_i \leq 9$). The body of her algorithm consists of a **while** loop that updates these variables.

(a) **HW (4 points)** Let $\mathcal{C}$ denote the configuration space. Determine $|\mathcal{C}|$. Your answer should be a very simple formula. (Recall that a *configuration* is a setting of all variables; for instance if $n = 5$ then $(9, 0, 6, 6, 1)$ is a configuration in this problem. The *configuration space* is the set of all configurations. So you need to count the configurations.)

(b) **XC (4 points)** Onaedo discovers that the following predicate is a loop invariant for her **while** loop:

$$x_1 \leq x_2 \leq \cdots \leq x_n. \tag{1}$$

Let us call this predicate M for "monotonicity," so $M(x_1, \ldots, x_n) = 1$ if and only if Eq. (1) holds. Let $\mathcal{C}_M$ denote the set of those configurations that satisfy M. Prove: $|\mathcal{C}_M| = \binom{n+9}{n}$.

(c) **HW (5+4 points)** Onaedo's algorithm needs to discover a configuration in $\mathcal{C}_M$ that satisfies a given predicate $T(x_1, \ldots, x_n)$ or show that no such configuration exists; let's call this "Onaedo's problem." Onaedo has a polynomial-time algorithm that decides, given a configuration $a = (a_1, \ldots, a_n)$, whether or not a satisfies T. (So T is a polynomial-time decidable predicate.) Prove that Onaedo's problem can be solved in polynomial time (polynomial in terms of the bit-length of the description of a single configuration when we encode each decimal digit by four binary digits). (c1) Write your algorithm in pseudocode. Explain what your algorithm does in English, and comment each line of your code to explain what happens in that line. No rigorous proof of correctness of your algorithm is required, just a brief indication why it works. (c2) Prove that your algorithm runs in polynomial time as specified above. Determine the exponent in the polynomial bound (the smallest constant C such that your algorithm runs in $O(N^C)$ where N is the bit-length of the description of a configuration). (Use part (b) for this; assume T can be computed in time $O(N^k)$.)

8.2 **HW (7 points) (transitivity of Karp reduction)** Recall the definition of Karp reduction: Let Σ_1 and Σ_2 be finite alphabets, $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$ be languages. (Σ^* denotes the set of all finite strings over the alphabet Σ.) A *Karp reduction* from L_1 to L_2 is a function $f : \Sigma_1^* \to \Sigma_2^*$ such that (i) f is computable in polynomial time; and (ii) $(\forall x \in \Sigma_1^*)(x \in L_1 \iff f(x) \in L_2)$.

We say that L_1 is *Karp-reducible* to L_2 if such an f exists; notation: $L_1 \preccurlyeq_{\text{Karp}} L_2$. We say that L_1 is Karp-reducible *with exponent* c to L_2 if the Karp reduction f can be computed in time $O(n^c)$ (where n is the bit-length of the input to f).

Prove that Karp-reducibility is transitive: if $L_1 \preccurlyeq_{\text{Karp}} L_2 \preccurlyeq_{\text{Karp}} L_3$ then $L_1 \preccurlyeq_{\text{Karp}} L_3$. (LaTeX for the curly inequality is `\preccurlyeq`.) Determine the exponent: Assuming L_1 is Karp-reducible to L_2 with exponent c and L_2 is Karp-reducible to L_3 with exponent d, what is the exponent of the reduction of L_1 to L_3? [Notation updated 3-4 12:30pm]

8.3 DO: Let NPC denote the class of NP-complete languages. Prove: if $\mathsf{NPC} \cap \mathsf{P} \neq \emptyset$ then $\mathsf{NP} = \mathsf{P}$. In other words, if there is an NP-complete language that can be decided in polynomial time then all languages

in NP can be decided in polynomial time. ("Deciding" a language L means deciding the membership problem in L, i. e., given $x \in \Sigma^*$, we need to decide whether or not $x \in L$.)

Comment: It is conjectured that $\mathsf{P} \neq \mathsf{NP}$. This exercise then says that, according to this conjecture, none of the NP-complete problems can be solved in polynomial time. So for instance it follows from the conjecture that 3-colorability cannot be decided in polynomial time. It does not follow from the conjecture that integers cannot be factored in polynomial time, although we believe that to be the case.

8.4 **HW (6 points)**: Prove: If 3-colorability of graphs can be decided in polynomial time then RSA can be broken in polynomial time. (Use the theorem that the class of 3-colorable graphs is NP-complete.) Note that breaking RSA is not a decision problem. Do not write any pseudocode for this problem; just explain in clear English the steps it will take to break RSA using a polynomial-time algorithm for 3-colorability.

8.5 DO: Prove: If $\mathsf{NPC} \cap \mathsf{coNP} \neq \emptyset$ then $\mathsf{NP} = \mathsf{coNP}$. In other words, if there is a well-characterized NP-complete language then all languages in NP are well characterized. (Check the "Material covered" link on the course website for the notion of good characterization.)

Comment: It is conjectured that $\mathsf{NP} \neq \mathsf{coNP}$. This exercise then says that, according to this conjecture, none of the NP-complete problems is well characterized. So for instance there are no polynomial-time verifiable certificates for non-3-colorability in the sense that there are infinitely many graphs that are not 3-colorable yet this fact cannot be certified by polynomial-time verifiable certificates. (Of course for some graphs it is easy to certify non-3-colorability: for instance if the graph contains K_4. But a graph need not contain K_4 in order to not be 3-colorable. This exercise says a graph may not be 3-colorable without any apparent reason.)

8.6 DO: Recall the decision version of the problem of factoring integers (into their prime factors); the corresponding language[1] is

$$\text{FACTOR} = \{(x, y) \mid (\exists d)(d \mid x \text{ and } 2 \leq d \leq y)\}$$

[1]The originally posted definition of the FACTOR language erroneously omitted the $2 \leq d$ constraint.

Prove: if FACTOR $\in$ NPC then NP = coNP. In other words, it follows from the NP $\neq$ coNP conjecture that FACTOR is NOT NP-complete.

Comment: While we believe FACTOR is a hard problem (FACTOR $\notin$ P), the problem of factoring is not as hard as 3-colorability in this structural sense. It is our focus on *decision problems* that allowed us to notice the asymmetry between "yes" and "no" answers, allowed us to define the complexity classes NP and coNP, and made it possible to make such statements of stuctural complexity like FACTOR $\in$ NP $\cap$ coNP and therefore, conjecturally, FACTOR $\notin$ NPC, thuis placing the factoring problem into a lower complexity class than the NP-complete problems. This distinction could not be observed based merely on the fact that the best algorithms known for factoring take exponential time (exponential in $\sqrt{n}$ where n is the bit-length of the input).

8.7 **HW (3 points)** Prove: If you prove that RSA cannot be broken in polynomial time then you are eligible for a million-dollar prize (unless someone else has already claimed that prize). You may use any of the exercises above and any result proved in class; and you may also use the theorem that 3-colorability is NP-complete. Name the institution that funds the prize.

8.8 **HW (5 points)** We are given an edge-weighted digraph $G = (V, E, w)$ where $w : E \to \mathbb{R}$ is the weight function. Assume $E \neq \emptyset$. Decide, in time $O(|V||E|)$, whether or not G contains a negative cycle. — Instructions. If we are also given a source vertex s then Bellman–Ford decides (in round $|V|$) whether or not a negative cycle, *accessible from* s, exists. So we could decide whether or not G has a negative cycle by runinng Bellman–Ford for all $s \in V$. But this would require $\Theta(|V|^2|E|)$ time ($|V|$ applications of $|V|$-round Bellman–Ford; each round scans the entire edge list). Solve the problem with a single application of Bellman–Ford to a weighted digraph other than G. Your job is to construct this other weighted digraph, H. Give a clear description of H. No pseudocode required.