# CMSC 303 Introduction to Theory of Computation, VCU
# Spring 2017, Assignment 7
# Due: Thursday, April 27, 2017 in class
# Steven Hernandez

Total marks: 52 marks + 5 marks bonus for typing your solutions in LaTeX using the provided A7 template file.

Unless otherwise noted, the alphabet for all questions below is assumed to be $\Sigma = \{0, 1\}$. This assignment focuses on the complexity classes P and NP, as well as polynomial-time reductions.

1. [12 marks]

   (a) [4 marks] Let $f(n) = 4n^2 - 30n + 6$. Prove that $f(n) \in O(n^2)$. In your proof, give explicit values for $c$ and $n_0$.

   ---

   Proof:
   $f(n) = 4n^2 - 30n + 6$
   $\leq 4n^2 + 6$ (since $n \geq 0$)
   $\leq 4n^2 + 6n^2$ (since $n \geq 0$)
   $= 10n^2$
   $\therefore$ if $c = 10$ and $n_0 = 8$, implies $f(n) = O(n^2)$.

   ---

   (b) [4 marks] Let $f(n) = n^{999}$ and $g(n) = (\sqrt{\log n})^{\sqrt{\log n}}$. Precisely one of the following two claims is true: $f(n) \in O(g(n))$, $g(n) \in O(f(n))$. Prove whichever one is true. In your proof, give explicit values for $c$ and $n_0$. (Hint: Note that an arbitrary function $f(n)$ can be rewritten as $2^{\log_2(f(n))}$.)

   ---

   $g(n) \in O(f(n))$ where $c = 1$ and $n_0 = 2$

   ---

   (c) [4 marks] This question tests an important subtlety in the definition of "polynomial-time". One of the most famous open problems in classical complexity theory is whether the problem of factoring a given integer $N$ into its prime factors is solvable in polynomial time on a classical computer[1]. Given positive integer $N$ as input, why is the following naive approach to the factoring problem not polynomial-time?

   ```
   1: Set m := 2.
   2: Set S := ∅ for S a multi-set.
   3: while m ≤ N do
   4:     if m divides N then
   5:         Set S ∪ {m}.
   6:         Set N = N/m.
   7:     else
   8:         Set m = m + 1.
   ```

[1] In fact, many popular cryptosystems are based on the assumption that this problem is *not* efficiently solvable. Recall from class, however, that in 1994 it was shown by Peter Shor that *quantum* computers can efficiently solve this problem!

9: **end if**
10: **end while**
11: Return the set $S$ of divisors found.

---

The input to this approach is always a single integer, thus $n = 1$. However, the amount of time it takes to process this algorithm is dependant on the value for the input $N$. Based on line three, the time complexity of this algorithm increases as the value of $N$ increases. This increase is exponential and thus, cannot be done in polynomial time.

Line 7 is a clever, never thought of it that way.

---

2. [6 marks] Let co-NP denote the complement of NP. In other words, intuitively, co-NP is the class of languages $L$ for which if input $x \notin L$, then there is an efficiently verifiable proof of this fact, and if $x \in L$, then no proof can cause the verifier to accept.

   Prove that if P = NP, then NP is closed under complement, i.e. NP = co-NP. How can closure of NP under complement hence potentially be used to resolve the P vs NP question?

---

Proof: We understand that if $P = NP$, then our TM $S \in NP$ will complete in polynomial time. Thus, we can build a verifier $V_{co-NP}$ which verifies some input $x \in L$ a Language by simply running the $S$ within $V_{co-NP}$.

$$V_{co-NP} = \text{"On input } \langle M, x \rangle \\ \quad 1. \text{ Simply run } M \text{ on } x, \text{ reject if } M \text{ accepts, otherwise accept.} \tag{1}$$

Because we know a TM can either $accept$, $reject$ or loop forever, having a verifier $V_{NP}$ to show that a machine will $accept$ and a verifier $V_{co-NP}$ to show that a machine will $reject$ on some input would mean we could infer all three states for the TM.

For example, if $V_{NP}$ accepts and $V_{co-NP}$ rejects on $z$, we know that $z$ is in the language. If $V_{NP}$ rejects and $V_{co-NP}$ accepts, the input is not in the language. However, if $V_{NP}$ rejects and $V_{co-NP}$ rejects, we know that the TM never accepts or rejects and thus, must loop forever.

---

3. [10 marks] In class, we introduced the language

$$\text{CLIQUE} = \{\langle G, k \rangle \mid G \text{ is a graph containing a clique of size at least } k\}.$$

   Consider now two further languages:

$$\begin{aligned} \text{INDEPENDENT-SET} &= \{\langle G, k \rangle \mid G \text{ is a graph containing an independent set of size at least } k\} \\ \text{VERTEX-COVER} &= \{\langle G, k \rangle \mid G \text{ is a graph containing a vertex cover of size at most } k\}. \end{aligned}$$

   Here, for graph $G = (V, E)$, an *independent set* $S \subseteq V$ satisfies the property that for any pair of vertices $u, v \in S$, $(u, v) \notin E$. A *vertex cover* $S \subseteq V$ satisfies the property that for any edge $(u, v) \in E$, at least one of $u$ or $v$ must be in $S$.

   (a) [4 marks] Prove that CLIQUE $\leq_p$ INDEPENDENT-SET. (Hint: Given a graph $G$, think about its *complement*. Google "complement graph" for a definition.)

---

Proof: Suppose we have $G = (V, E)$ which is the graph we wish to find an independent set of size at least $k$. We construct a graph $G_{inv} = (V, E_{inv})$ where $V$ (of $G$) $= V$ of $G_{inv}$. $E_{inv}$ is build based on the edges $E$ of $G$ in this way:

$\forall v, w \notin E$, we add an edge to $E_{inv}$. This will create an inversed graph of $G$ where all nodes which were not connected in $G$ are now connected in $G_{inv}$ and where all nodes were connected in $G$ are not connected in $G_{inv}$.

Proof of Correctness: INDEPENDENT-SET is satisfiable if and only if $G_{inv}$ has a clique of at least k.

Let $x = (x_1, ..., x_m)$ be a satisfying set of vertexes in $G$ for INDEPENDENT-SET for size $k$ where $|x| \geq k$. For $\forall y, z \in x$ we know that $(y, z) \notin E$, thus by our construction above, $(y, z) \in E_{inv}$. Thus now, we know that $G_{inv}$ would have edges between all $y, z \in x$. Therefore $x \in G_{inv}$ is a clique.

Suppose $G_{inv}$ has a k-clique. Let $x = (x_1, ..., x_m)$ satisfy CLIQUE. Thus, there must be an edge between each vertices $q, r \in G_{inv}$. By the reverse of the construction above, that means for our $G$, $q, e \notin G$. Therefore our vertices would be an independent-set.

---

(b) [6 marks] Prove that INDEPENDENT-SET $\leq_p$ VERTEX-COVER.

---

Reference for more information on vertex-cover:

https://www8.cs.umu.se/kurser/TDBA77/VT06/algorithms/BOOK/BOOK3/NODE108.HTM

The idea here is that if we have a

Proof: Suppose we have $G = (V, E)$ a graph which we wish to find a vertex-cover of size at most $k$. If we have a vertex-cover in $G$ of size $k$, then we must have an independent-set in $G$ of size $|V| - k$. Why is this? Because by definition of our independent-set, we are finding any vertices which have edges that are in $E$, but do not connect to any other vertices in our independent-set. What would happen if a vertex in the independent-set connected to another vertex in the independent-set? This show that the edge does not connect to the vertex-cover and thus, we would have a vertex-cover that does not cover all edges and thus is not a vertex-cover by our definition.

We need to construct a TM $S_{vertex-cover}$ which uses our other TM $S_{independent-set}$ to decide.

$$
\begin{aligned}
S_{vertex-cover} = \text{"On input } & G(V, E), k \\
& \text{Let } k' = |V| - k \\
& \text{Run } S_{independent-set} \text{ on } G, k' \\
& \text{accept if } S_{independent-set} \text{ accepts, otherwise reject."}
\end{aligned}
\tag{2}
$$

---

4. [8 marks] Let $CNF_k = \{\langle\phi\rangle \mid \phi \text{ is a satisfiable CNF-formula where each variable appears in at most } k \text{ places}\}$. Show that $CNF_3$ is NP-complete. (Hint: Suppose a variable $x$ appears (say) twice. Replace the first and second occurrences of $x$ with new distinct variables $y_1$ and $y_2$, respectively. Next, add clauses to ensure that $y_1 = y_2$. To do this in CNF form, recall that $y_1 = y_2$ is logically equivalent to $(y_1 \implies y_2) \wedge (y_2 \implies y_1)$, and that $(a \implies b)$ can be rewritten as $(\bar{a} \vee b)$. How can this trick be applied more generally when $x$ appears $m > 2$ times?)

---

Proof: First we show that $\text{CNF}_3 \in NP$. We can do this easily by setting showing a satisfying value for $\phi$ which can be verified in polynomial time. $\phi = (a \vee a \vee a) \wedge (b \vee b \vee b)$. Our verifier $V$ which takes in input $\phi$ simply runs as follows:

$$
\begin{aligned}
V = \text{"On input } & \phi, k \\
& \text{For each variable } v \in \phi \\
& \qquad \text{Keep a tally of the number of times } v \text{ appears} \\
& \qquad \text{If tally} > k, \text{ reject} \\
& \text{accept"}
\end{aligned}
\tag{3}
$$

Now, we can show that CNF$_k$ is NP-hard. We can show this by creating a reduction from 3-SAT.

We create a TM $S$ which simulates CNF$_3$ with TM $3SAT$ as well as TM $V$ from above.

$$
\begin{aligned}
S = \text{"On input } & \phi, k \\
& \text{We simply run } \phi \text{ on TM } 3SAT\text{, reject if the TM rejects} \\
& \text{Run } V\text{, reject if } V \text{ rejects} \\
& \text{accept"}
\end{aligned}
\tag{4}
$$

Because $V$ can run in polynomial time, it can be disregarded in checking if CNF$_k \in$ NP. Thus, what we are left with is $3SAT$ which we've shown in the past is NP-hard. Thus CNF$_k$ is NP-complete.

---

5. [8 marks] It is possible for a problem to be NP-hard without actually being in NP itself. For example, the halting problem from class is clearly not in NP, since it is undecidable. However, in this question, you will show that the language $HALT = \{\langle M, x\rangle \mid M \text{ is a TM which halts on input } x\}$ is NP-hard. Is the halting problem hence NP-complete?

---

A problem NP-hard means that all problems in NP are reducible to this problem in polynomial time. By this, obviously all problems are reducible HALT because all problems can be modeled into a language which can be defined by a TM. To answer the problem, we would have to be able to decide if this TM will halt, and thus HALT could be considered a 'parent' or 'harder'' problem.

This does not mean that the halting problem is NP-complete, because as stated in the question, "the halting problem is clearly not in NP". To be NP-complete, a problem must initially be in NP.

---

6. [8 marks] In class, we have focused on *decision* problems, i.e. deciding whether $x \in L$ or not. For example, given a 3-CNF formula $\phi$, recall that the decision problem 3SAT asks whether $\phi$ is satisfiable or not. In practice, however, we may not just want a YES or NO answer, but also an actual assignment which satisfies $\phi$ whenever $\phi$ is satisfiable. It turns out that in certain settings, being able to solve the *decision* version of the problem (e.g. 3SAT) in polynomial time implies we can also solve the *search* version (e.g. find the satisfying assignment itself) in polynomial time. Problems satisfying this property are called *self-reducible*.

Your task is as follows: Show that 3SAT is self-reducible. In other words, given any 3-CNF formula $\phi$ and a polynomial-time black-box $M$ for determining if $\phi$ is satisfiable, show how to find a satisfying assignment for $\phi$ in polynomial time. You may assume that $M$ is able to decide all CNF formulae with at most 3 variables per clause.

---

Reference for more information on 'self-reducibility': http://www.cs.toronto.edu/ fpitt/20109/CSC363/lectures/LN11.txt

Suppose we have a 3SAT with k clauses (eg. $(x_1 \vee \bar{x_2} \vee x_3) \wedge ...$). We can create a TM $3CNF_{search}$ modeled:

$$
\begin{aligned}
3CNF_{search} = \text{"On input } & \phi \\
& \text{For each clause } c \in \phi\text{:} \\
& \quad \text{For each variable } y \in c\text{:} \\
& \quad\quad \text{Let } \phi_1 = \phi - y \text{ from clause } c \\
& \quad\quad \text{Run } M \text{ on } \phi_1 \\
& \quad\quad \text{If } M \text{ accepts, then set } \phi = \phi_1 \\
& \text{return } \phi
\end{aligned}
\tag{5}
$$

$3CNF_{search}$ runs through all 3 variables in all $k$ clauses. Thus, it is completed in $O(3k) \times O(M)$.

The idea of this machine is that we remove variables from clauses until the clause no longer returns true. If the clause no longer returns true, this means that the removed variable was the only required variable for the clause.

---