Week 12

Collaboration: You should work on the problems yourself, before discussing with others, including your cohorts at your cohort meeting. By the Assessed Cohort Meeting, you and all of your cohortmates, should be prepared to present and discuss solutions to all of the assigned problems. In addition to discussing with your cohortmates, you may discuss the problems with any other current CS3102 students you want, and use any resources you want except for any materials from previous offerings of this course or complete solutions that might be available on the web, which are not permitted. **You may not collaborate on the assigned writeup with your cohort-mates or anyone else, but you may use notes taken before or during your assessed cohort meeting.**

You writeups for this week will work similarly to last week's. At the end of your assessed cohort meeting, your Cohort Coach will assign a *new* problem for the writeup that you must do on your own. Since we will have a final exam in the course (and a midterm-like activity during the week of fall break) we wanted to give you experience solving problems on your own so that you can check your progress. As such, this week's writeup will not be taken directly from this week's problem set, and instead will be a new problem that will expect you to demonstrate your ability do a reduction proof on your own.

Problem 1: Flawed Reductions

Definition 1 (k-CNF) We say that a formula is in conjunctive normal form (CNF for short) if it is an AND of ORs of variables or their negations. E.g. $(x_7 \vee \overline{x_{22}} \vee x_{15}) \wedge (x_{37} \vee x_{22} \vee \overline{x_7})$ is in CNF. We say that it is k-CNF if there are exactly k variables per clause (group of variables combined with OR).

Recall that 3-SAT is in *NP-Hard*, where 3-SAT requires determining whether there exists at least one way to assign Boolean values to each variable in a 3-CNF formula so that the formula evaluates to True.

The 4-SAT problem requires determining whether there exists at least one way to assign Boolean values to each variable in a 4-CNF formula so that the formula evaluates to True.

Below we've presented several different flawed approaches for demonstrating that 4-SAT is in *NP-Hard*. Identify the main flaw or flaws in each approach, and explain things the proposed approach is misunderstanding.

(a) Convert a given 3-CNF formula into a 4-CNF formula by putting `False` into each clause. For example, if the 3-CNF formula given to 3-SAT was $(x_7 \vee \overline{x_{22}} \vee x_{15}) \wedge (x_{37} \vee x_{22} \vee \overline{x_7})$, we would add `False` to each clause to create the 4-CNF formula $(x_7 \vee \overline{x_{22}} \vee x_{15} \vee \texttt{False}) \wedge (x_{37} \vee x_{22} \vee \overline{x_7} \vee \texttt{False})$. This new 4-CNF formula will be satisfiable if and only if the original 3-CNF formula is satisfiable.

(b) Convert a given 4-CNF formula into a 3-CNF formula by breaking each clause of 4 variables into two clauses of 3 variables in such a way that both of the new clauses are satisfiable if and only if the original clause was satisfiable. For example, if the 4-CNF formula that was given to 4-SAT was $(x_7 \vee \overline{x_{22}} \vee x_{15} \vee x_9) \wedge (x_{37} \vee x_{22} \vee \overline{x_7} \vee x_{12})$, we would introduce two new variables (one per original clause) and break up each clause to create the 3-CNF formula $(x_7 \vee \overline{x_{22}} \vee n_1) \wedge (\overline{n_1} \vee x_{15} \vee x_9) \wedge (x_{37} \vee x_{22} \vee n_2) \wedge (\overline{n_2} \vee \overline{x_7} \vee x_{12})$.

(c) In the _Polynomial Time Reductions_, we showed that 2-SAT (the problem of satisfying a 2-CNF formula) was easier than 3-SAT (the problem of satisfying a 3-CNF formula). This demonstrates that it is harder to satisfy CNF formulas with more variables per clause. Since 3-SAT is in *NP-Complete*, and 4-CNF has more clauses than 3-CNF, it must be harder, so 4-SAT is in *NP-Hard*.

Problem 2: Silly Reductions

Consider the $SORTING$ and $MINIMUM$ problems defined below:

> *SORTING*
> **Input:** A list of n natural numbers, $x_1, x_2, x_3, \ldots, x_n$.
> **Output:** An ordering of the input list, $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ where $\{i_1\} \cup \{i_2\} \cup \ldots \{i_n\} = \{1, 2, \ldots, n\}$ and for all $k \in \{1, 2, \ldots, n-1\}$, $x_{i_k} \leq x_{i_{k+1}}$.
>
> *MINIMUM*
> **Input:** A list of n natural numbers, $x_1, x_2, x_3, \ldots, x_n$.
> **Output:** A member, x_m, such that $x_m \in \{x_1, x_2, \ldots, x_n\}$ and for all $k \in \{1, 2, \ldots, n\}$, $x_m \leq x_k$.

(a) Show that $MINIMUM$ is polynomial-time reducible to $SORTING$.

(b) Show that $SORTING$ is polynomial-time reducible to $MINIMUM$.

(c) Does this mean that solving $MINIMUM$ and solving $SORTING$ require algorithms with the same asymptotic running time?

Problem 3: TAUT

Definition 2 (DNF) We say that a formula is in disjunctive normal form (DNF for short) if it is an OR of ANDs of variables or their negations. E.g. $(x_7 \wedge \overline{x_{22}} \wedge x_{15}) \vee (x_{37} \wedge x_{22} \wedge \overline{x_7})$ is in DNF. We say that it is k-DNF if there are exactly k variables per clause (group of variables combined with AND).

We know that 3-SAT is in *NP-Complete*, where 3-SAT requires determining whether there exists at least one way to assign Boolean values to each variable in a 3-CNF formula so that the formula evaluates to True.

The 3-TAUT problem requires determining whether *every* assignment causes a 3-DNF formula to evaluate to True (i.e., no assignments will cause the formula to evaluate to False).

(a) Show that 3-TAUT is in *NP-Hard*.

(b) 3-TAUT is not known to belong to *NP*. Give an intuitive reason why it is difficult to show that 3-TAUT belongs to *NP*.

(For the historical significance of the 3-TAUT problem, see the optional video on the history of the Cook-Levin Theorem at the end of the Week 12 guide.)

Problem 4: Fact Checking

This *P vs NP on TV* Computerphile video (https://www.youtube.com/watch?v=dJUEkjxylBw) with Simon Singh (who normally does a great job presenting technically challenging topics in a very accessible, but essentially correct, way) contains a faulty discussion of the P vs NP question. Watch this video, and identify as many incorrect or misleading statements as you can (we were able to identify 2 between time stamps 0:25 and 0:30 alone). Propose corrections for all such statements you identify.

Problem 5: Who Wants To Be A Millionaire?

For each of the following statements below, indicate whether it would resolve the $\mathrm{P} = \mathrm{NP}$ problem. If it would resolve it, indicate the direction it would be resolved. If it would not resolve it, explain why.

(a) A problem from class EXP is found to be in class P.

(b) A problem from class EXP is found to be in class *NP-Hard*.

(c) A problem from class *NP-Hard* is found to be in class P.

(d) A problem from class *NP-Hard* is found to be in class NP.

(e) A problem from class *NP-Hard* is found to be outside of the class P

Problem 6: $\mathrm{P} = \mathrm{NP}$ for RAM Machines?

We discussed in the *"Difficulty" of Functions* video that we measure the "difficulty" or "complexity" of a function as the minimum amount of a resource required to implement that function using a given model of computing. This idea leads into the discussion in the *RAM Model* video that the "difficulty" of a problem might depend on the model of computing used.

We will call the class P the set of all functions computable by a tape Turing Machine in polynomial time, and call the class P_{RAM} the set of all functions computable by a RAM Turing Machine in polynomial time. We will similarly define NP and NP_{RAM}.

(a) Show that $\mathrm{P} = \mathrm{P}_{RAM}$

(b) Show that $\mathrm{NP} = \mathrm{NP}_{RAM}$

