Problem Set 10

Due: **10:00pm, Monday, April 25**

This problem set focuses on complexity. Namely, it revisits time complexity, polynomial-time reductions, and $\mathbf{NP}$-completeness. Notice that $\mathbf{NP}$-completeness is not yet covered on April 15 but will be covered in the next class.

Write your answers in the ps10.tex LaTeX template. You will submit your solutions in GradeScope as a PDF file with your answers to the questions in this template. There are four "required" problems and one "bonus" practice, where the bonus gives extra points.

Collaboration Policy: You may discuss the problems with anyone you want. You are permitted to use any resources you find for this assignment **other than solutions from previous/concurrent CS3120 courses**. You should write up your own solutions and understand everything in them, and submit only your own work. You should note in the *Collaborators and Resources* box below the people you collaborated with and any external resources you used. You shall explicitly state the *content*, e.g., the main message in your collaborated discussion, the search keywords, the LLM/AI prompts, or the section in a book.

Collaborators and Resources: TODO: replace this with your collaborators and resources (if you did not have any, replace this with *None*)

To do this assignment:

1. Open this read-only Overleaf project located at https://www.overleaf.com/read/gfmwyrgtzwdb#bce762, and then copy this project.
2. Open your copy of the project and in the left side of the browser, you should see a file directory containing ps10.tex. Click on ps10.tex to see the LaTeX source for this file, and enter your solutions in the marked places.
3. The first thing you should do in ps10.tex is set up your name as the author of the submission by replacing the line, `\submitter{TODO: your name}`, with your name and UVA id, e.g., `\submitter{Haolin Liu (srs8rh)}`.
4. Write insightful and clear answers to all of the questions in the marked spaces provided.
5. Before submitting your PDF file, also remember to:
   - List your collaborators and resources, replacing the TODO in `\collaborators{TODO: replace ...}` with your collaborators and resources. (Remember to update this before submitting if you work with more people.)
   - Replace the second line in ps10.tex, `\usepackage{uvatoc}` with `\usepackage[response]{uvatoc}` so the directions do not appear in your final PDF. Starting from this Problem Set, we may deduct 3pt if you forgot this step.

Problem 1 (6pt) *Running Time Analysis*

Consider the *INCREMENT* problem defined below:

> **Input:** A natural number, x, encoded using binary representation with *least significant bit first*.
>
> **Output:** A binary encoding, least significant bit first, of $x + 1$.

(a) Characterize using asymptotic notation the *average* running time cost for solving *INCREMENT*, where cost is the number of steps required by a standard Turing Machine. (The average running time depends on the distribution of inputs. For this, you should assume for any input length n, all binary strings of length n are equally likely as input.)

(b) Characterize using asymptotic notation the *worst-case* running time cost for solving *INCREMENT*, where cost is the number of steps required by a standard Turing Machine. (You should be able to get a tight bound using Θ notation. Use n to represent the size of the input (that is, the number of input cells needed to represent the input x in binary notation) in your answer.)

(c) Does your answer to either sub-question change if the input and output are represented using *most significant bit first* instead of with *least significant bit first*?

Answer:

(a)

(b)

(c)

Problem 2 (6pt) *Polynomial-Time Reductions*

For each sub-problem, indicated if the state proposition is **True** or **False**, and provide a brief (one or two sentences) justification for your answer.

We use the notations in the book: let $F, G : \{0,1\}^* \to \{0,1\}^*$. We say that F reduces to G, denoted by $F \leq_p G$, if there is a polynomial-time computable $R : \{0,1\}^* \to \{0,1\}^*$ such that for every $x \in \{0,1\}^*$, $F(x) = G(R(x))$.

(a) $F \leq_p G$ and $G \in \mathbf{P}$ implies $F \in \mathbf{P}$.

(b) $F \leq_p G$ and $F \in \mathbf{P}$ implies $G \in \mathbf{P}$.

(c) $F \leq_p G$ and $G \in \mathbf{EXP}$ implies $F \in \mathbf{EXP}$.

(d) $F \leq_p G$ and $G \leq_p F$ implies $F \in \mathbf{P}$.

(e) $F \leq_p G$ and G is computable implies F is computable.

(f) $F \leq_p G$ and F is computable implies G is computable.

Answer:

(a)

(b)

(c)

(d)

(e)

(f)

More Powerful Polynomial-Time Reductions. The reduction definition in the book (and in Problem 2) is called a *mapping reduction*, *many-to-one reduction* or a *Karp reduction*. The reason for this naming is that there are *other*, perhaps less restricted, forms of reductions as well. A Karp reduction can be interpreted as follows. Suppose we have a *subroutine* (e.g., from a library) that computes G correctly. Then, using this subroutine, we can solve F by using the reduction R. In doing so, the reduction R will process the input x (for which we want to know $F(x)$) to $R(x) = y$ in such a way that $G(y)$ will *exactly* provide the answer $F(x)$ that we would like to find.

Note that this definition of reduction only allows G to be used once, on a single input $R(x)$, and the output of G cannot be processed to produce the desired output $F(x)$. We can do more than the Karp reduction definition allows and still have a valid polynomial-time reduction. The requirement is that everything done to construct the reduction must be something that will not change whether the execution takes a polynomial number of steps. For example, we can allow more than one call to the subroutine $G(\cdot)$, so long as the number of calls is polynomial in the input size. We could also use a function Q to post-process the output to produce $F(x)$ so long as we know Q runs in polynomial-time in the size of the original input. This more general form of reduction is known as a *Cook* reduction (we will talk about Stephen Cook and why it is named after him in a future class).

Problem 3 (4pt) *Silly Reductions*

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

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

(a) Show that $MINIMUM$ can be polynomial-time reduced to $SORTING$ (read the note above on more powerful notions of reduction, and you can use the Cook reduction definition here). Recalling the discussion in the previous problem, this means we are looking for way to finding minimum, assuming that we can use a subroutine that computes the sorting. Try to find a reduction that makes only one call to the sorting subroutine.

(b) Show that $SORTING$ can be reduced to $MINIMUM$ using a *Cook* reduction. The difference with the previous part is that this time the subroutine that provides answers to $MINIMUM$ can be used *multiple times* by the reduction, and it is the job of the reduction to use the answers to finally obtain the correct sorted output.

(c) Does this mean that $MINIMUM$ and $SORTING$ are equivalently hard problems?

(d) We can solve $SORTING$ *without* using a subroutine the solves $MINIMUM$, and vice versa. Would that be a reduction? Briefly explain your answer.

Answer:

(a)

(b)

(c)

(d)

Problem 4 (5pt) *Jeffersonian Paths*

The Hamlitonian Path problem (not named after Alexander), defined below, is known to be $\mathbf{NP}$-complete.

> *HAMILTONIAN PATH*
> **Input:** A description of an undirected, finite graph, $G = (V, E)$.
>
> **Output:** **True** if there exists a path in G that visits each vertex in V exactly once; otherwise **False**.

Despite his declarations to the contrary, Jefferson does not consider all vertices equal, and defines the *JEFFERSONIAN PATH* problem as:

> *JEFFERSONIAN PATH*
> **Input:** A description of an undirected, finite graph, $G = (V, E)$, and a partitioning of V into two subsets V_1 and V_2 such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$.
>
> **Output:** **True** if there exists a path in G that visits each vertex in V exactly once, where all vertices in V_1 are visited before any vertex in V_2; otherwise **False**.

Prove *JEFFERSONIAN PATH* is $\mathbf{NP}$-complete.

Answer:

Problem 5 (4pt) *Turing Machines and Concrete Computers*

(a) Give three reasons why a Turing Machine is better than the computer you are typing on now.

(b) Give three reasons why the computer you are typing on now is better than a Turing Machine.

Answer:

(a)

(b)

Do not write anything on this page; leave this page empty.

This is the end of the problems for PS10. Remember to follow the last step in the directions on the first page to prepare your PDF for submission.