CMPUT 675 - Winter 2019
Assignment #1 - Due Feb 7 by 12:30pm

All **Exercises** have the same weight, regardless of their difficulty. You are allowed to skip one exercise freely with no penalty. If you answer all questions, I will drop the one with the lowest mark when computing your mark for this assignment.

It is highly recommended that you typeset your solutions in LaTeX. I still want hard copies of your solution, so submit a printout if you do typeset it. The only exception is the first question, everyone must submit it by email (unless you are using it as your "skipped" question).

Pages: 4

Exercise 1: Designing A Turing Machine

Design a Turing machine that decides the following language.

$$\mathcal{L} = \{x \in \{0,1\}^* : x = yy \text{ for some } y \in \{0,1\}^*\}$$

For example, 101101 and 00110011 are in $\mathcal{L}$ but 110011 and 111 are not in $\mathcal{L}$.

You must implement your solution as a complete and correct program that runs on the Turing machine simulator found at

`https://turingmachinesimulator.com/`

Consult the documentation and examples on that page to understand how to specify and run a TM on the site. You may use up to 2 tapes to do this[1]. You may only use 0, 1, and blank symbols. **Do not forget to recompile before testing if you make changes to the "code" ☺.**

If you don't want to create an account, you can just copy/paste the code into your own plain text file to save it.

Submit your code to me by email attached as a plain-text `.txt` file. You should have some comments in the code explaining how your solution works. I should be able to test it by copy/pasting into the simulator.

Exercise 2: Busy Beaver and Some Number Theory

Let $\lambda \in \{0,1\}^*$ denote the empty string, so $|\lambda| = 0$. For $n \geq 1$, let $\mathcal{M}_n^{\text{halt}}$ be all single-tape Turing machines M with $\leq n$ states (apart from the halting states) and alphabet $\Gamma = \{0, 1, \Box\}$ such that M halts when given λ as input.

For a Turing Machine $M \in \mathcal{M}_n^{\text{halt}}$, let $\textsc{steps}(M)$ be the number of steps taken by M when given the empty string as input.

Consider the **busy beaver** function $\text{BB} : \mathbb{N}_{\geq 1} \to \mathbb{N}$ given by

$$\text{BB}(n) = \max\{\textsc{steps}(M) : M \in \mathcal{M}_n^{\text{halt}}\}.$$

[1] Open a 2-tape example and then edit it yourself to start a 2-tape project.

Note that $\mathrm{BB}(n)$ is well-defined: it is easily seen that $\mathcal{M}_n^{\mathrm{halt}} \neq \emptyset$: *eg.* some TMs immediately transition to a halting state.

- Show that no TM can compute $\mathrm{BB}(n)$ for all $n \geq 1$.
- Consider *Goldbach's Conjecture*: Every even integer $n \geq 4$ can be expressed as the sum of two primes. For example, $4 = 2 + 2, 24 = 7 + 17$ and $3572 = 101 + 3461$. Resolving this conjecture is still an open problem.

  Show there is a TM M such that $M(\lambda) = \textsc{accept}$ if Goldbach's conjecture is true and $M(\lambda) = \textsc{false}$ if Goldbach's conjecture is false.

  Furthermore, show there is some constant n_0 such that if we have a value x and a proof that $\mathrm{BB}(n_0) = x$ then we can use these to algorithmically generate either a proof or a refutation of Goldbach's conjecture in finite time!

Exercise 3: Running-Time Bounds

Question 3.1 from the book.

Show that the following language is not decidable.

$$\{\alpha \in \{0,1\}^* : \text{for some } c, d > 0, M_\alpha \text{ halts within } c \cdot |x|^2 + d \text{ steps on every input}\}.$$

Hint: Show how to decide HALT if you had a TM deciding this language.

Exercise 4: Exactly One 3SAT

Question 2.17 from the book.

In the problem EXACTLY-ONE-3SAT, we are given a 3CNF instance $\varphi = (X, \mathcal{C})$, just like with E3SAT. The goal is to determine if there is a truth assignment $\tau : X \to \{\textsc{True}, \textsc{False}\}$ such that *exactly* one literal is TRUE under τ in each clause of φ .

Show that EXACTLY-ONE-3SAT is **NP**-complete.

Hint from the textbook
Replace each occurrence of a literal ℓ_i in a clause $C \in \mathcal{C}$ of an E3SAT instance by a new variable $z_{i,C}$. Add clauses and further auxiliary variables to the EXACTLY-ONE-3SAT instance you are constructing to ensure that if ℓ_i is TRUE then $z_{i,C}$ is allowed to be either TRUE or FALSE but if ℓ_i is FALSE then $z_{i,C}$ must be FALSE.

Exercise 5: Cook Reductions

Part of this is found in Question 2.14 from the book.

A language L is **Cook reducible** to a language L' if $L \in \mathbf{P}^{L'}$. Here, $\mathbf{P}^{L'}$ is the set of all languages that are decidable by polynomial-time oracle TMs with oracle access to L'.

Recall, from the lectures, that $L \leq_{\mathbf{p}} L'$ means L is polynomial-time Karp reducible to L'. That is, there is a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for $x \in \{0,1\}^*$, $x \in L$ if and only if $f(x) \in L'$.

- Show that the notion of Cook reductions is weaker than the notion of Karp reductions. That is, for two languages L, L' prove that $L \leq_{\mathbf{p}} L'$ implies L is Cook reducible to L'.
- Say a language L' is **NP**-hard *with respect to Cook reductions* if every language $L \in \mathbf{NP}$ is Cook reducible to L'. Show if there is some $L' \in \mathbf{P}$ that is also **NP**-hard with respect to Cook reductions, then $\mathbf{P} = \mathbf{NP}$.
- Show that the notion of Cook reductions is transitive.
- Show that SAT is Cook reducible to TAUTOLOGY (the language of all Boolean expressions that are true under every truth assignment).
- Show that if SAT $\leq_{\mathbf{p}}$ TAUTOLOGY then $\mathbf{NP} = \mathbf{co\text{-}NP}$.

Exercise 6: Quadratic Equations

Question 2.20 from the book.

Let QUADEQ be the language of all satisfiable systems of *quadratic equations* over $\mathbb{Z}/2\mathbb{Z}$ (integers modulo 2). That is, an instance has variables $x_1, \ldots, x_n$ and constraints of the form

$$\sum_{i,j} a_{i,j} \cdot x_i \cdot x_j = b$$

for some given $a_{i,j} \in \mathbb{Z}/2\mathbb{Z}$ and some $b \in \mathbb{Z}/2\mathbb{Z}$.

For example, consider the following instance with variables x_1, x_2, x_3, x_4 and three constraints:

$$\begin{aligned} x_1 \cdot x_3 + x_2 \cdot x_2 + x_3 \cdot x_4 &= 1 \\ x_1 \cdot x_1 + x_3 \cdot x_3 &= 0 \\ x_1 \cdot x_4 + x_1 \cdot x_2 + x_2 \cdot x_3 + x_2 \cdot x_4 &= 0 \end{aligned}$$

Here, equality is modulo 2: $1+1 = 0$ when working over $\mathbb{Z}/2\mathbb{Z}$. Note the left side of each expression has each term being a product of variables. There are no linear terms.

An instance is **satisfiable** if it is possible to assign values in $\mathbb{Z}/2\mathbb{Z}$ to the variables $x_1, \ldots, x_n$ so all equalities hold. For example, the assignment $(x_1, x_2, x_3, x_4) \to (1, 0, 1, 0)$ satisfies the three quadratic equalities above.

Show QUADEQ is **NP**-complete.

Note: This is not just an artificial problem, it will be an important starting point when we talk about how to probabilistically verify certificates for languages in **NP** by only querying a *constant* number of bits.

Hint
Exploit the fact that $x_i \cdot x_i = x_i \pmod 2$.

Exercise 7: Very Sparse Languages

Question 2.10 from the book.

Call a language L *unary* if for each $n \geq 0$, $L \cap \{0,1\}^n$ is empty or only contains 1^n. Show that if some unary language is **NP**-hard then $\mathbf{P} = \mathbf{NP}$.

Hint
Given a CNF instance ϕ, generate a set of pairs $\{(f(\psi_i), \psi_i)\}$ where each ψ_i is a CNF instance such that ϕ is satisfiable if and only if one of the ψ is satisfiable. Evolve this set by trying both ways to fix the assignment to a particular variable but be careful to make sure the set of pairs continues to have polynomial size throughout this process.

Exercise 8: Ladner's Function is Efficiently Computable

This is essentially Question 3.6.a) from the book. All logarithms are base-2 logarithms.

Recall the function H used in the proof of Ladner's theorem. That is, $H(n)$ is the minimum $i \leq \log \log n$ such that for every $x \in \{0,1\}^*$ with $|x| \leq \log n$, the computation of $M_i(x)$ halts within $i \cdot |x|^i$ steps and correctly decides if $x \in \mathrm{SAT}_H$. If there was no such i, we use $H(n) = \log \log n$. Here, we are sayings that M_i is M_α where $\alpha \in \{0,1\}^*$ is the binary encoding of i.

The use of $\log \log n$ is slightly informal as it is not defined for all natural numbers n and does not always produce an integer. For the sake of concreteness, use $g(n) := \lceil \log(\lceil \log(n+2) \rceil) \rceil$ which is computable in $\mathrm{poly}(n)$ time. You don't have to show this, just assume it. The proof of Ladner's theorem works with this concrete function[2].

Show that $H(n)$ can be computed in $\mathrm{poly}(n)$ time.

The book gives a really strong hint (copied below). Your job is just to put the pieces together by describing the algorithm with some care and providing a good accounting of the running time.

Hint from the book
The essential ingredients are (1) we need to recursively compute $H(i)$ on every $i \leq \log n$, (2) simulate $O(\log \log n)$ machines, each on various inputs of length $O(\log n)$ for at most $g(n) \cdot (\log n)^{g(n)} = o(n)$ steps (you should prove this asymptotic bound), and (3) decide SAT on instances of size $O(\log n)$. One can design such an algorithm deciding $H(n)$ with running time $T(n)$ satisfying $T(n) \leq n \cdot T(\log n) + O(n^2)$, which simplifies to $O(n^2)$.

[2] Of course, if we are being picky we have to tweak a other things to make $H(n)$ precise, such as the problem that $i \cdot |x|^i = 0$ for the empty string x but you don't have to do this.