CMPUT 675 - Winter 2019
Take Home Final Assignment
Due Friday, April 26 by 3:00pm

All **Exercises** have the same weight, regardless of their difficulty. You are allowed to skip **two** exercises freely with no penalty. As always, let me know if something is not clear.

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 a "skipped" question).

Unlike the assignments which adopted the collaboration model, you must complete everything individually without discussion of any form with anyone else except the instructor.

Finally, if you don't find me in person to drop off the exam then please slide your exam under my door (ATH 3-06) and email me if you have done so.

Pages: 5

Exercise 1: Designing A Turing Machine (Again)

For a string $s \in \{0,1\}^*$, consider the unique way to break s into its maximal substrings $s^1, \ldots, s^m$ each consisting of only 0s or only 1s. For example, if $s = \texttt{0011110100011}$ then $m = 6$ and

$$s^1 = \texttt{00}, \quad s^2 = \texttt{1111}, \quad s^3 = \texttt{0}, \quad s^4 = \texttt{1}, \quad s^5 = \texttt{000}, \quad s^6 = \texttt{11}.$$

Call such a string **valid** if $|s^1| = 1$ and $|s^{i-1}| \leq |s^i| \leq |s^{i-1}| + 1$ for every other $2 \leq i \leq m$. Observe the example above is not valid. The empty string is not valid.

Examples of valid strings:

$$\texttt{0101100111}, \quad \texttt{0}, \quad \texttt{101010}, \quad \texttt{1001110000111110000000}.$$

More examples of invalid strings:

$$\texttt{1100111}, \quad \texttt{10110}, \quad \texttt{10011100001110000}.$$

The first of these is not valid because it begins with a run of length 2. The second is not valid because the last run has length 1 whereas the previous run has length 2. The last example is not valid because the last run of 1s has length 3 but the preceding run has length 4.

Let $\textsc{Valid} \subseteq \{0,1\}^*$ be the language of all valid strings. Design a Turing machine to decide $\textsc{Valid}$.

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 any number of tapes to do this[1]; I suggest 2 tapes. 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: Indecisive Turing Machines

Define an **indecisive** verifier to be a verifier with three halting states: q_{ACCEPT}, q_{REJECT} and q_{UNSURE}. Say that an indecisive verifier V decides a language $\mathcal{L}$ if the following is true for each $x \in \{0,1\}^*$:

- $x \in L \Rightarrow \exists \pi \in \{0,1\}^*$ such that $V(x,\pi) = q_{\text{ACCEPT}}$ and $\forall \pi' \in \{0,1\}^*$, $V(x,\pi') \in \{q_{\text{ACCEPT}}, q_{\text{UNSURE}}\}$.
- $x \notin L \Rightarrow \exists \pi \in \{0,1\}^*$ such that $V(x,\pi) = q_{\text{REJECT}}$ and $\forall \pi' \in \{0,1\}^*$, $V(x,\pi') \in \{q_{\text{REJECT}}, q_{\text{UNSURE}}\}$.

If you prefer, a nondeterministic variant where π is replaced by nondeterministic transitions works as well. Just indicate if you are taking this view in your solution.

Let **INP** be the class of all languages that are decided by an indecisive verifer V whose running time is poly($|x|$).

Prove $\mathbf{INP} = \mathbf{NP} \cap \textbf{co-NP}$.

Exercise 3: Another NL-Complete Problem

In the class, we saw that determining if there is a directed $s-t$ path in a graph is **NL**-complete with respect to implicitly log-space computable reductions. Now consider the following language:

$$\mathcal{L}_{\text{SC}} = \{A : A \text{ is the } \{0,1\}\text{-adjacency matrix of a strongly connected digraph}\}.$$

Recall that a digraph is strongly connected if and only if for every two vertices u, v, there is a directed path from u to v.

Show that $\mathcal{L}_{\text{SC}}$ is **NL**-complete with respect to implicitly log-space computable reductions.

Exercise 4: Randomized Logspace Complexity

For this exercise, you should take the view of probabilistic TMs as being those with two transition functions, where each step randomly picks one (i.e. do not take the verifier view). This is especially important in the second part.

Let **BPL** be all languages $\mathcal{L}$ that can be decided by a probabilistic Turing machine M that always runs in polynomial time and always uses $O(\log n)$ work space such that for $x \in \{0,1\}^*$,

- $x \in \mathcal{L} \Rightarrow \mathbf{Pr}[M(x) = \text{ACCEPT}] \geq 2/3$, and

[1] Open a 2- or 3-tape example and then edit it yourself to start a multi-tape project.

- $x \notin \mathcal{L} \Rightarrow \mathbf{Pr}[M(x) = \text{REJECT}] \geq 2/3$.

So **BPL** is to **L** as **BPP** is to **P**.

- Show that $\mathbf{BPL} \subseteq \mathbf{P}$.

  Hint: Compute the probability the computation ends up in an accepting state for a given input x either with dynamic programming or matrix multiplication.

Now consider another complexity class **ZPL** consisting of languages $\mathcal{L}$ decidable by a probabilistic Turing machine M that always uses $O(\log n)$ work space and never terminates with the incorrect solution. But their expected running time is not required to be bounded, we simply have the guarantee that M halts with probability 1.

- Give an example of such a machine where the expected running time is exponential in n.

  Hint: First try building a directed graph with, say, m nodes (m has nothing to do with the problem at hand, it is just for this hint) with out-degree 2 at each node such that the expected number of steps of a random walk to get from node 1 to node m is at least 2^m.

  Then try to emulate this behaviour with the computation of some probabilistic TM. It is ok if the machine performs superfluous work that does not seem to be helping it decide the problem (*i.e.* the only purpose of the extra work is to answer this question).

- Still, argue $\mathbf{ZPL} \subseteq \mathbf{P}$.

Exercise 5: Adaptive PCPs

Suppose we allowed PCP verifiers to be adaptive. That is, consider an $(r(n), q(n))$-PCP verifier V taking inputs x, y, π where x is an instance of the language to be decided, y is the random string, and π is the proof string (into which V has random access). The first proof position i_1 is a function of the input x and the random string y, but each subsequent queried proof position i_j is a function of $(x, y, \pi_{i_1}, \pi_{i_2}, \ldots, \pi_{i_{j-1}})$.

That is, V can decide the next position to query after looking at a few bits. We still require V to run in polynomial time.

For each $r(n) \in \cup_{c \geq 0} O(n^c)$ and $q(n) \in O(\log n)$, show that if $\mathcal{L}$ is decided by an adaptive $(r(n), q(n))$-PCP verifier then it is also decided by a nonadaptive (*i.e.* standard) $(r(n), 2^{q(n)} - 1)$-PCP verifier.

Note: This means the statement of the main PCP theorem with $q(n) \in O(1)$ remains (qualitatively) the same whether we use adaptive or nonadaptive verifiers.

Exercise 6: Short Keys and Perfect Secrecy

We saw that if $\mathbf{P} = \mathbf{NP}$, then for any efficient encryption scheme that uses keys that are even just one bit shorter than the message, there is an efficient algorithm that can distinguish between some messages with good probability.

Here, you will show that without the complexity assumption, true perfect secrecy is still not possible using encryption schemes with short keys. That is, suppose (E, D) is an encryption/decryption scheme using keys of length $K(n)$. Show for any n such that $K(n) < n$ that there are messages $x, y \in \{0,1\}^n$ such that

$$\mathbf{Pr}_{k \sim \{0,1\}^{K(n)}}[E_k(x) = z] \neq \mathbf{Pr}_{k \sim \{0,1\}^{K(n)}}[E_k(y) = z]$$

for some $z \in \{0,1\}^*$.

Exercise 7: Reducing Random Bits for BPP Algorithms

Let $G = (V; E)$ be an (n, d, λ)-expander. Consider some $B \subseteq V$ and say $\beta = |B|/|V|$. Also consider a standard random walk $v_0, v_1, \ldots, v_k$ of length k in G. That is, v_0 is chosen uniformly from V and then v_i is chosen uniformly among the neighbours of v_{i-1} for $1 \leq i \leq k$. By this, we mean we first uniformly sample a random edge exiting v_{i-1} and set v_i to be the other endpoint of this edge: it could be $v_{i-1} = v_i$ if this edge is a loop.

- For every $I \subseteq \{0, \ldots, k\}$, argue $\mathbf{Pr}[\bigwedge_{i \in I}(v_i \in B)] \leq ((1 - \lambda) \cdot \sqrt{\beta} + \lambda)^{|I|-1}$.

  You can just mention the relevant parts that need to be modified in the proof was saw for a similar result.

- Conclude if $|B| \leq n/100$ and $\lambda < 1/100$, the probability there is some $I \subseteq \{0, \ldots, k\}$ such that $|I| > k/10$ and $v_i \in B$ for all $i \in I$ is at most $2^{-k/100}$. The constants (eg. 1/100, 1/10) here do not matter too much, so long as whatever you do prove will allow you to correctly answer the next part.

- Use this to show that every **BPP** algorithm A that flips m coins and outputs the correct answer (for the language it is deciding) with probability $\geq 2/3$ can be turned into another **BPP** algorithm that, for any k being bounded by a polynomial in the input size, uses $O(m+k)$ coins and outputs the correct answer with probability $\geq 1 - 2^{-k}$. Note this is an improvement over our earlier result for improving error bounds for **BPP** that gets the same error bound, but using $O(m \cdot k)$ coins.

You may use, without proof, that for every $0 < \lambda < 1$ there is some d such that for every $n \geq 1$ there is a strongly-explicit (n, d, λ)-expander. If you are curious, this can be proven by piecing together parts of what we covered in the course plus the proof of Theorem 21.19 from the book (we did not cover this).

Exercise 8: Beyond The Halting Problem

For any constant $k \geq 0$, consider the following complexity class $\mathbf{RE}_k$. A language $\mathcal{L} \subseteq \{0,1\}^*$ lies in $\mathbf{RE}_k$ if there is a deterministic Turing machine M that halts on every input such that for $x \in \{0,1\}^*$:

- $x \in \mathcal{L}$ if and only if
  $\exists y_1 \in \{0,1\}^* \; \forall y_2 \in \{0,1\}^* \; \exists y_3 \in \{0,1\}^* \ldots \; Q_k y_k \in \{0,1\}^* \; M(x, y_1, y_2, \ldots, y_k) = \text{ACCEPT}$.

Here, Q_k will be $\exists$ if k is odd and Q_k will be $\forall$ if k is even. Similarly, define **co-RE**$_k$ to be all languages $\mathcal{L}$ such that $\overline{\mathcal{L}} \in \mathbf{RE}_k$ which can, alternatively, be defined in the same way as $\mathbf{RE}_k$ except the order of quantifiers is $\forall \; \exists \; \forall \; \exists \ldots$.

Your mark for this exercise will be based on your best answers to 5 of the parts below, so feel free to skip a part if you want (it will not count against your ≤ 2 free questions).

1. Show $\textsc{Halt} \in \mathbf{RE}_1$ where $\textsc{Halt} = \{(\alpha, x) : M_\alpha(x) \text{ halts after a finite number of steps}\}$.

2. Call a language $\mathcal{L} \subseteq \{0,1\}^*$ **recursively enumerable** if there is some Turing machine M such for $x \in \{0,1\}^*$ we have $M(x) = \textsc{accept}$ if and only if $x \in \mathcal{L}$. But if $x \notin L$ we do not require $M(x)$ to halt.

   Argue that a language $\mathcal{L}$ is recursively enumerable if and only if $\mathcal{L} \in \mathbf{RE}_1$.

3. Consider the language $\textsc{AlwaysHalts} = \{\alpha \in \{0,1\}^* : M_\alpha \text{ halts on every input}\}$.
   Show $\textsc{AlwaysHalts} \in \textbf{co-RE}_2 - \textbf{co-RE}_1$.

   Hint: It might help if you can prove why $\textsc{Halt}$ itself is not in $\textbf{co-RE}_1$.

4. Show $\mathbf{RE}_{k+1} \neq \mathbf{RE}_k$ for all $k \geq 0$.
   Hint: The case $k = 0$ was already proven in the lectures (using different terminology). Simply generalize the proof.

5. Show there is some $\mathcal{L} \subseteq \{0,1\}^*$ such that $\mathcal{L} \notin \mathbf{RE}_k$ for any $k \geq 0$.

6. Show $\mathbf{RE}_1^{\textsc{Halt}} = \mathbf{RE}_2$. Here, $\mathbf{RE}_1^{\textsc{Halt}}$ is the same definition as $\mathbf{RE}_1$ except the TM M in the definition has oracle access to $\textsc{Halt}$.
   Hint: To show $\mathbf{RE}_1^{\textsc{Halt}} \subseteq \mathbf{RE}_2$, try using quantified variables to "guess" the result of each oracle call and to certify this guess. Partial credit if you show $\mathbf{RE}_1^{\textsc{Halt}} \subseteq \mathbf{RE}_k$ for some $k \geq 3$ if you can't get the inclusion with $k = 2$.