CS 535: Complexity Theory, Fall 2023

Homework 1

Due: 11:59PM, Tuesday, September 12, 2023.

Reminder. Homework must be typeset with LaTeX preferred. Make sure you understand the course collaboration and honesty policy before beginning this assignment. Collaboration is permitted, but you must write the solutions *by yourself without assistance*. You must also identify your collaborators. Assignments missing a collaboration statement will not be accepted. Getting solutions from outside sources such as the Web or students not enrolled in the class is strictly forbidden.

Problem 0 (Housekeeping)**.** Please fill out the first day survey (`https://forms.gle/dcLKu6JuAE8QeFJa7`). Read and sign the course Collaboration and Honesty Policy (`https://cs-people.bu.edu/mbun/courses/535_F23/handouts/collaboration-policy.pdf`) and upload it to Gradescope. An electronic signature is fine.

Problem 1 (Encoding Graphs)**.** An encoding is an unambiguous way to represent an object, e.g., a natural number, a graph, or a Turing machine, as a string in Σ^* for some alphabet Σ. Encodings allow us to present complex objects as inputs to Turing machines and other computational devices. One way to formalize an encoding of a set of objects D is as a one-to-one (injective) function $\mathrm{Enc} : D \to \Sigma^*$.

A **canonical directed graph** G is a pair (V, E) where the vertex set $V = [n]$ for some natural number $n \in \mathbb{N}$, and the edge set $E \subseteq V \times V$ is a set of ordered pairs of vertices.

(a) Describe an explicit encoding of the set D consisting of all canonical directed graphs using the alphabet $\Sigma = \{0, 1\}$. You don't have to go into excruciating detail here, but it should be unambgiuous to the person reading your solution how any given directed graph G maps to its encoding $\mathrm{Enc}(G)$. (3 points)

(b) How efficient is your encoding? That is, what is the (asymptotically) best upper bound $T(n)$ you can give on $\mathrm{Enc}(G)$ whenever G is a canonical directed graph on vertex set $V = [n]$? Express your answer using big-O notation and briefly justify it. (3 points)

(c) Show that for *every* encoding function Enc, there exists an n-vertex canonical directed graph G whose encoding length is $|\mathrm{Enc}(G)| \geq \binom{n}{2}$. How does your upper bound from part (b) compare to this lower bound? (4 points)

Problem 2 (Arithmetizing Turing Machines)**.** In this class, we usually think of Turing machines as computing functions $f : \{0,1\}^* \to \{0,1\}^*$, but TMs and their variants are also naturally suited to computing functions $f : \mathbb{N} \to \mathbb{N}$ over the natural numbers. In this problem you'll explore a few such TM variants that historically helped bridge the machine view of computability with the recursive function / λ-calculus view.

(a) A **binary arithmetic machine** maintains a tuple of counters $(C_1, \ldots, C_m)$ and computes a function $f : \mathbb{N} \to \mathbb{N}$ as follows. At initialization, the first counter C_1 is set to the input $z \in \mathbb{N}$ and all other counters are set to 0. In one step of computation, the machine "reads" the contents of each counter by testing whether it is equal to zero. Based on this information, and its finite control state, it may then (possibly independently for each counter) update each counter using one of the following arithmetic operations:

$C_i \leftarrow C_i$	[no change]
$C_i \leftarrow 0$	
$C_i \leftarrow 1$	
$C_i \leftarrow 2C_i + C_j$	for some other counter C_j
$C_i \leftarrow \lfloor C_i/2 \rfloor$	
$C_i \leftarrow C_j \mod 2$	for some other counter C_j.

When the machine halts, its output is the natural number loaded in the last counter C_m.

Show (using enough detail to convince your human reader) that a binary arithmetic machine using at most $3k$ counters can simulate an arbitrary k-tape Turing machine. That is, every k-tape TM can be converted into a $3k$-counter binary arithmetic machine that solves the same problem. Here you may convert between natural number inputs / outputs and binary strings in any reasonable way of your choice. (10 points)

(b) (*Bonus Problem*) A **counter machine** also maintains a tuple of counters $(C_1, \ldots, C_m)$, but in each step of computation it is only allowed to read its counters by testing whether they are equal to zero, and then update each counter by either incrementing by one, decrementing by one, or leaving it the same. The output of the machine is the content of counter C_m when the machine halts.

(i) Show that a counter machine can simulate a binary arithmetic machine, with a constant-factor blowup in the number of counters used.

(ii) What is the runtime overhead of your simulation? That is, given a binary arithmetic machine running in time $T(n)$, what is the runtime of your counter machine as a function of $T(n)$?

(iii) Is it possible to significantly improve your runtime overhead? Prove your answer.