CSE491/596, Fall 2020 Problem Set 6 Due Mon. Nov. 23, 11:59pm
Plus topics for presentations on Nov. 19–20

Lectures and Reading: Next week's lectures will prove the PSPACE-completeness of the TQBF problem and prove Savitch's Theorem. Along with observing that the Circuit Value Problem (CVP) is complete for P under $\leq_m^{\log}$ and reiterating that GAP is complete for NL under $\leq_m^{\log}$, which will round out the major completeness classes. After one lecture on oracle Turing machines, things will move on to classical probabilistic computation and the class BPP. All of these topics are in the last sections 17–19 of Debray's notes, though not in that order, and we will follow the completeness results from the ALR notes, section 5 of Chapter 28. The remaining reading for the lectures after Thanksgiving will be from selected chapters of my textbook with Richard Lipton on quantum computation.

I have not heard of any issues with the date of **Monday, Nov. 30**, in the regular 3pm class period (with one exception), so we can consider that fixed. It will cover material up through the domain of this problem set. There may be a choice of kind of reduction problem.

Assignment 6, written part due Mon. Nov. 23, 11:59pm

(1) (12 + 30 + 3 = 45 pts.)

Consider the following decision problem:

EDGE-DISJOINT PATHS
INSTANCE: A *directed* graph G and nodes $s_1, s_2, t_1, t_2 \in V(G)$.
QUESTION: Do there exist a path P_1 from s_1 to t_1 and a path P_2 from s_2 to t_2 such that no edge connects a vertex used in P_1 to a vertex used in P_2?

(a) Find and explain the flaw(s) in this attempt to classify the problem into NL: "Design an NTM N with two worktapes used to guess the next vertex u in the path P_1 and the next vertex v in the path P_2. Initially $u = s_1$ and $v = s_2$, and N can also write t_1 and t_2 on separate tapes as the targets without violating the $O(\log n)$ space bound. Using one or two more tapes to maintain indices i, j of nodes in the input graph, N can guess a u' that has an out-edge from u as the next step of P_1, and then guess a v' going out from v as the next step of P_2. Finally, N can go once more to the input graph to check if there is an edge between u' and v' (in either direction). If so, then the current nondeterministic branch dies (maybe some other branch will make luckier guesses); while if this never happens and we eventually get $u' = t_1$ and $v' = t_2$ then N accepts."

(b) Prove that this problem is NP-complete, using a reduction from 3SAT. You must use a standard "rungs and clause gadgets" type architecture, though it will be different from the "size k" problems used in lectures. It will need some extra framework so that you can run one path "up the ladder" and the other path through the clause gadgets.

(c) Name one or more "drastic" consequences if the language of this problem were to belong to NL after all.

(d) **Added:** And for *15 points extra credit*, explain why the logical analysis of your answer to (b) would go wrong if G is an undirected graph.

(2) $3 \times 6 =$ **18 pts. total**

A collection $\{ \mathcal{C}_i \}$ of complexity classes forms a *proper hierarchy* if given any $\mathcal{C}_i$ and $\mathcal{C}_j$ with $i \neq j$, one of them is properly contained in the other. Which of the following collections are proper hierarchies? Justify your answers, mainly by verifying the relevant "little-o" or "Θ" relations between time bounds. Here $\mathbf{Q}^+$ stands for the positive rational numbers.

(a) $\{ \mathsf{DTIME}[n^c] : c \in \mathbf{Q}^+, c \geq 1 \}$. **Not graded: see presentation option (2)**

(b) $\{ \mathsf{DTIME}[(n+c)^3] : c \in \mathbf{Q}^+, c \geq 1 \}$.

(c) $\{ \mathsf{DTIME}[2^{cn}] : c \in \mathbf{Q}^+ \}$.

(d) $\{ \mathsf{DTIME}[2^{n^{1/c}}] : c \in \mathbf{Q}^+, c \geq 1 \}$.

(3) (18 + 6 + 15 = 39 pts., for 102 on the set)

Consider the following decision problem:

CYCLING DFA
INSTANCE: A DFA $M = (Q, \Sigma, \delta, s, F)$ and a string $x \in \Sigma^n$ where $Q = \{1, 2, \ldots, n\}$.
QUESTION: Does M on input x visit every one of its states and end up back at s?

(a) Sketch a deterministic Turing machine M that decides this problem in $O(\log n)$ space. It is enough to diagram the worktapes of M and say what information each one maintains while sketching the algorithm in pseudocode.

(b) Also estimate the worst-case running time of your M. (It is AOK to ignore $O(\log n)$ factors by using $\tilde{O}$ notation, meanwhile ignoring the difference between n and the true instance length which is $N = \Theta(n \log n)$.)

(c) Now sketch a faster algorithm that uses linear space. You may use the fact that Turing machines can execute mergesort at full efficiency. Estimate its running time and compare with your answer to (b).

Presentation Options For 11/19–20, (5) and (6) teamable

(1) For those who like numerics: Prove the final $t' = O(t \log n)$ estimate in the k-tapes-to-2 simulation. Define J_1 to be the 4-step path $0 \to 1 \to 0 \to -1 \to 0$. For $k \geq 2$, define J_k to be

$$J_{k-1} \to 1 \to 2 \to \cdots \to 2^k{-}1 \to 2^k{-}2 \to \cdots \to 1 \to 0 \to -1 \to \cdots \to -(2^k{-}1) \to -1 \to J_{k-1}.$$

Note that J_{k-1} appears twice, and you should substitute it into this definition recursively to make one long path. Or you can picture the process iteratively going forward:

$$J_2 = 0 \to 1 \to 0 \to -1 \to 0 \to 1 \to 2 \to 3 \to 2 \to 1 \to 0 \to -1 \to -2 \to -3 \to -2 \to -1 \to 0 \to 1 \to 0 \to -1 \to 0.$$

$$\begin{aligned} J_3 \;=\; & 0 \to 1 \to 0 \to -1 \to 0 \to 1 \to 2 \to 3 \to 2 \to 1 \to 0 \to -1 \to -2 \to -3 \to -2 \to -1 \to 0 \to 1 \to 0 \to -1 \to 0 \\ & \to 1 \to 2 \to 3 \to 4 \to 5 \to 6 \to 7 \to 6 \to 5 \to 4 \to 3 \to 2 \to 1 \to 0 \to -1 \to -2 \to -3 \to -4 \to -5 \to -6 \\ & \to -7 \to -6 \to -5 \to -4 \to -3 \to -2 \to -1 \to 0 \\ & \to 1 \to 0 \to -1 \to 0 \to 1 \to 2 \to 3 \to 2 \to 1 \to 0 \to -1 \to -2 \to -3 \to -2 \to -1 \to 0 \to 1 \to 0 \to -1 \to 0 \end{aligned}$$

What makes this a little tricky is that you don't immediately get t' as a function of t. Instead you get both as functions of k: t' is the number of steps in J_k, and t is (proportional to) the number of times the subsequence $0 \to 1 \to 0 \to -1 \to 0$ appears. Then combine the equations so k goes away and you can estimate t' in terms of t. Be precise enough so that you can give the constant factor in the "O."

You are welcome to simplify by dropping the negative-number portions so everything is halved—this should not change the constant in the O-notation. Finish by summarizing how this allows a Turing machine M running in time $t(n)$ first to be simulated by an *oblivious* two-worktape TM M' running in time $t'(n) = O(t(n) \log t(n)$, and then by Boolean circuits C_n of size $O(t(n) \log t(n)$.

(2) For those who like (explaining away) apparent paradoxes: Given any real number $c \geq 1$, let DQ_c abbreviate $\mathsf{DTIME}[\tilde{O}(n^c)]$. The case $c = 1$ is deterministic quasilinear time, which the "scholia" to the Cook-Levin theorem called DQL.

- Show that whenever $c < d$, DQ_c is properly contained in DQ_d. (This has essentially the same proof as (a) of problem (2) on the homework's written part.) This means there is a language A_d in DQ_d that is not in DQ_c.
- From the fact that there are *uncountably* many real numbers, conclude that there are uncountably many different complexity classes of the form DQ_d.
- But wait—there are only countably many decidable languages in all. How can there be uncountably many languages A_d??

If this doesn't perplex you, don't choose this. If this does perplex you but you resolve it, fine. If this perplexes you and you stay perplexed, you can contact me for a hint.

(3) Show that the following problem is complete for $\mathsf{co\text{-}NP}$ under $\leq^p_m$:

$\mathrm{ALL}_{REG,n}$
INSTANCE: A regular expression α using only $0, 1, +, \cdot$ (no Kleene star), and a number n.
QUESTION: Does α match every string in $\{0,1\}^n$?

For a hint, take a 3DNF formula $\psi(x_1, \ldots, x_n) = T_1 \vee \cdots \vee T_m$. Then ψ is a tautology if and only if every assignment $a \in \{0,1\}^n$ satisfies one of the terms. (Or if you prefer, think of a 3CNF formula $\phi(x_1, \ldots, x_n) = C_1 \wedge \cdots \wedge C_m$ and note that ϕ is *un*satisfiable if and only if every assignment $a \in \{0,1\}^n$ "unsatisfies" one of the clauses.) Show how to create α from ψ (or from ϕ) and give analysis to show that the reduction from 3TAUT (or from the complement of 3SAT) is correct. (For membership in $\mathsf{co\text{-}NP}$, note that powering abbreviations like $(0+1)^{17}$ are disallowed, or you can require n to be given as 0^n in unary notation.)

(4) Show that the following problem is NP-complete:

BINARY LINEAR EQUATIONS
INSTANCE: A set $E_1, \ldots, E_m$ where each E_j is a linear equation in three of the variables $x_1, \ldots, x_n$.
QUESTION: Is there a solution to the equations in which each x_i is 1 or 0?

Here is where using a reduction from Exactly One 3SAT (rather than from vanilla 3SAT) may come in especially handy. Note that the equations can have nonzero constant terms—this is the case when solving linear equations in the form $Ax = b$ where A is an $m \times n$ matrix, $x = (x_1, \ldots, x_n)$ is the n-vector being solved for, and $b = (b_1, \ldots, b_m)$ is the m-vector of constants in the equations. "Wait a second: $Ax = b$ is solvable in polynomial time by methods taught in MTH309. Why doesn't that prove $\mathsf{NP} = \mathsf{P}$?" Explain...

(5) Let us revisit the palindrome language, specifically its subset $P = \{v10^r1w : v, w \in \{0,1\}^r, w = v^R\}$ from the HW4 presentation option (D). Let M be a deterministic Turing machine *with a read-only input tape* such that $L(M) = P$, M runs in time $t(n)$, and M runs in space $s(n)$. Prove that

$$t(n)s(n) = \Omega(n^2),$$

where $n = 3r + 2$ is the whole length of the input as before. Note that there are two ways of doing no worse than this: One is to copy v to a tape, go to the end of the input tape, and compare v against w^R while going right-to-left on the input tape. This uses linear time but also linear space. At the other extreme is the idea of comparing bits of v and w one at a time, each time taking the k-th bit of v from the front and the k-th bit of w from the end. Without changing any input tape characters, this can be managed by calculating r and maintaining r, k, and a counter j going from 1 to k, each in binary notation on a separate tape. This uses $O(\log n)$ space but quadratic time, giving $t(n)s(n) = O(n^2 \log n)$. We can make the time-space product exactly quadratic by comparing v in $\log n$-sized chunks at a time, rather than single bits. There are ways in-between, such as using $O(\sqrt{n})$ space to compare $\sqrt{n}$-sized chunks at a time, but this takes order-$n^{1.5}$ time, giving $t(n)s(n) = \Omega(n^2)$ again.

To prove $t(n)s(n) = \Omega(n^2)$ in general, the trick is that once an input length n is given, we can (wlog.) mark off the $s(n)$ cells that M is allowed to use in advance. Then we can consider every possible contents $u \in \Gamma^{s(n)}$ of these cells. For any fixed n, we can imagine M instead as a single-tape TM M' with state set $Q' = Q \times \Gamma^{s(n)}$ allowing for every possible "state" of the worktapes too. Redo the calculation of presentation option (D) with Q' in place of Q. You may freely use the answer key of (D). You may use other sources as well, but should make your presentation follow this outline and make the calculation look like that of (D).

(6) With reference to (5), now suppose we allow Turing machines M'' a source of randomness. Specifically, we can allow them to compute hash functions $h_\rho : \{0,1\}^r \to \{0,1\}^\ell$ from random seeds ρ of length $O(\ell)$ such that for all distinct $v, w \in \{0,1\}^r$,

$$\Pr_\rho[h_\rho(v) = h_\rho(w)] = \frac{1}{2^\ell}.$$

Suppose this holds for all $\ell \leq r$ and that $h_\rho(v)$ can always be computed in $O(r)$ time and $O(\ell)$ space. Show that now we can arrange for M'' to decide whether a given string $v10^r1w$ belongs to P while beating the time-space tradeoff in (5), on pain of a slight $\frac{1}{2^\ell}$ chance of erroneously accepting when $v10^r1w \notin P$.

[This problem can be done independent of (5) but makes a nice combo with it. This is reminiscent of how many password systems are implemented. The system does not store your whole password w but rather a hash of it, $h(w)$. There may be a chance that an intruder can gain access by hitting on a string $v \neq w$ such that $h(v) = h(w)$, but by choosing ℓ wisely—maybe $\ell > r$, especially if your password is short—it can be made minuscule.]