CPSC 421: Introduction to Theory of Computing
Tutorial #2, Not to be handed in

Note: $\mathbb{N} = \{0, 1, \dots\}$, $[n] = \{1, \dots, n\}$ for integers $n \geq 1$.

1. (Time hierarchy theorem)

   Time Hierarchy Theorem. If f, g are time-constructible functions satisfying $f(n) \log f(n) = o(g(n))$ then

   $$\mathsf{TIME}\big(f(n)\big) \subsetneq \mathsf{TIME}\big(g(n)\big).$$

   Equivalently, for all time-constructible functions $f : \mathbb{N} \to \mathbb{N}$,

   $$\mathsf{TIME}\left(o\left(\frac{f(n)}{\log f(n)}\right)\right) \subsetneq \mathsf{TIME}\big(f(n)\big)$$

   Note: f is in $o(g)$ if $\lim_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| = 0$, and f is $O(g)$ if $\limsup_{n\to\infty} \left|\frac{f(n)}{g(n)}\right| < \infty$.

   (a) Show that $\mathsf{TIME}(2^n) = \mathsf{TIME}(2^{n+1})$.

   Solution. This is because $2^{n+1} = O(2^n)$.

   (b) Show that $\mathsf{TIME}(2^n) \subsetneq \mathsf{TIME}(2^{2n})$.

   Solution. Both 2^n and 2^{2n} are time-constructible functions. Since $2^n = o\big(2^{2n}/\log(n)\big)$, the claim follows by the time hierarchy theorem.

   (c) Show that at least one of the following must be true. Either $\mathsf{P} \neq \mathsf{NP}$ or $\mathsf{NP} \neq \mathsf{EXP}$.

   Solution. By the time hierarchy theorem, we have $\mathsf{P} \neq \mathsf{EXP}$. Hence, if $\mathsf{P} = \mathsf{NP}$ then it follows that $\mathsf{NP} \neq \mathsf{EXP}$.

   (d) We saw in homework that $\mathsf{TIME}(1) = \mathsf{TIME}(f(n))$ whenever $f(n) = o(n)$. Explain why this does not contradict the time hierarchy theorem.

   Solution. The time hierarchy theorem says nothing about what happens when $f(n) = o(n)$. In fact, the proof of the time hierarchy required the simulating TM to know the length of its input. This is not possible in time $o(n)$!

2. (Easy warm-up on P and NP) Let X and Y be languages and suppose that X can be reduced to Y in polynomial time. Which are the following are necessarily true?

   (a) If Y is NP-complete then so is X.

   Solution. False.

   (b) If X is NP-complete then so is Y.

   Solution. False.

   (c) If Y is NP-complete and X is in NP then X is NP-complete.

   Solution. False.

   (d) If X is NP-complete and Y is in NP then Y is NP-complete.

   Solution. True.

(e) X and Y cannot both be NP-complete.

Solution. False.

(f) If X is in P then Y is in P.

Solution. False.

(g) If Y is in P then X is in P.

Solution. True.

3. Is P closed under:

(a) union?

Solution. Suppose $L_1, L_2 \in \mathsf{P}$. Let M_i be a polynomial time decider for L_i. For any x, run M_1 on x then M_2 on x. If either accepts then accept. Else reject. The running time is polynomial since each M_i runs in polynomial time.

(b) concatenation?

Solution. Suppose $L_1, L_2 \in \mathsf{P}$. We want to show that $L_1 \circ L_2 \in \mathsf{P}$. Given an input string x and for every splitting of $x = yz$, test if $y \in L_1$ and $z \in L_2$. The running time is polynomial since each M_i runs in polynomial time and we only do $O(n)$ tests.

(c) complement?

Solution. If $L \in \mathsf{P}$ and M decides L in polynomial time then the TM that negates the output of M is a polynomial time decider for $\overline{L}$.

Do parts (a) and (b) for NP. If you solve part (c) for NP as well, please let the instructors know.

Solution. Parts (a) and (b) are pretty much the same as above. For part (b), we can even split the string nondetemrinistically. Part (c) is a major open question.

4. Recall the definition of the complexity class NP: A language $L \subseteq \{0,1\}^*$ is in NP if there exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a polynomial-time TM M such that for every $x \in \{0,1\}^*$,

$$x \in L \Leftrightarrow \exists u \in \{0,1\}^{p(|x|)} \text{ s.t. } \text{Result}(M, \langle x, u\rangle) = \text{yes}.$$

If $x \in L$ and $u \in \{0,1\}^{p(|x|)}$ satisfy $\text{Result}(M, \langle x, u\rangle) = \text{yes}$, then we call u a *certificate* for x (with respect to the language L and machine M).

(a) Argue that the verifier definition above of NP is *asymmetric* with respect to verification of yes/no instances.

Solution.

If $x \in L$; i.e., x is a "yes" instance, then it has a short certificate (polynomial in $|x|$) u that can be verified in time that is $\text{poly}(|x|, |u|)$. In particular, just because M rejects a particular $\langle x, u\rangle$, it does not mean that we may conclude that $x \notin L$; instead, it just means that we have not yet verified that $x \in L$, and perhaps it may be impossible to verify this, but we cannot be sure yet. In short, the definition does not require that the "no" instances have succinct certificates.

(b) Recall that $\mathsf{EXP} = \bigcup_{c>1} \mathsf{TIME}\big(2^{n^c}\big)$. Here is a proof that $\mathsf{NP} = \mathsf{EXP}$. Consider any language L in EXP. Let us construct a verifier M that takes as input $\langle x, u\rangle$, where u is a string of ones of length $2^{|x|^k}$, $k \in \mathbb{N}$. Then the verifier is allowed to run for $2^{|x|^k}$ time, and so it can run any exponential time algorithm for L. But our Alternative Characterization (above) of NP then proves that L is in NP. So $\mathsf{NP} = \mathsf{EXP}$.

Is this proof correct ? If not, what is the flaw ?

Solution.

It is not correct. The flaw: $|u|$ needs to be poly($|x|$).

(c) Show that $\mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{EXP}$ using the Alternative Characterization of NP (above) as the *definition* of the class NP. Recall that $\mathsf{P} = \bigcup_{c\geq 0} \mathsf{TIME}(n^c)$.

Solution. ($\mathsf{P} \subseteq \mathsf{NP}$): Suppose $L \in \mathsf{P}$ is decided in polynomial-time by a TM N. Then $L \in \mathsf{NP}$, since we can take N as the machine M in the definition of NP above and make $p(x)$ the zero polynomial (in other words, u is an empty string).

($\mathsf{NP} \subseteq \mathsf{EXP}$): If $L \in \mathsf{NP}$ and M, $p(\cdot)$ are as in the definition of NP above, then we can decide L in time $2^{O(p(n))}$ by enumerating all possible u with $|u| \leq p(|x|)$ and using M to check whether u is a valid certificate for the input x. The machine accepts iff such a u is ever found. Since $p(n) = O(n^c)$ for some $c \geq 0$, this machine runs in $2^{O(n^c)}$ time.

(d) We say that NTM M runs in $T(n)$ time if for every input $x \in \{0,1\}^*$ and *every* sequence of non-deterministic choices, M reaches either q_{accept} or q_{reject} (halts) within $T(|x|)$ steps. That is, $T(n)$ is the maximum number of steps that M uses on *every branch* of its computation on any input of length n. For every function $T : \mathbb{N} \to \mathbb{N}$ and $L \subseteq \{0,1\}^*$, we say that $L \in \mathsf{NTIME}\big(T(n)\big)$ if there is a constant $c > 0$ and a $cT(n)$-time NTM M such that for every $x \in \{0,1\}^*$, $x \in L \Leftrightarrow \text{Result}(M, x) = \text{yes}$. Use the characterization of NP above as the *definition* of the class NP to show that $\mathsf{NP} = \bigcup_{c\geq 0} \mathsf{NTIME}(n^c)$.

Solution.

The main idea is that the sequence of nondeterministic choices made by an accepting computation of an NTM can be viewed as a certificate that the input is in the language, and vice versa.

- $\mathsf{NP} \subset \bigcup_{c\geq 0} \mathsf{NTIME}(n^c)$: If $L \in \mathsf{NP}$ according to the verifier definition above, then we describe a polynomial-time NTM N that decides L. On input x, it uses the ability to make non-deterministic choices to write down a string u of length $p(|x|)$. Then it runs the deterministic verifier M of the Alternative Characterization definition above to verify that u is a valid certificate for x, and if so, enters q_{accept}. TM N enters q_{accept} on x if and only if a valid certificate exists for x. Since $p(n) = O(n^c)$ for some $c > 1$, we conclude that $L \in \mathsf{NTIME}(n^c)$.
- $\bigcup_{c\geq 0} \mathsf{NTIME}(n^c) \subset \mathsf{NP}$: Suppose $p : \mathbb{N} \to \mathbb{N}$ is a polynomial and L is decided by a NTM N that runs in time $p(n)$. For every $x \in L$, there is a sequence of nondeterministic choices that makes N reach q_{accept} on input x. We can use this sequence as a certificate for x. Notice, this certificate has length $p(|x|)$ and can be verified in polynomial time by a deterministic machine, which checks that N would have entered q_{accept} after using these nondeterministic choices. In more detail, the following is a description of the verifier M:

$M =$ "On input $\langle w, c\rangle$, where w and c are strings:

i. Simulate N on input w, treating each symbol of c as a description of the nondeterministic choice to make at each step (as in the proof of Sipser, Theorem 3.16).

ii. If this branch of N's computation accepts, accept; otherwise, reject."

Thus $L \in \mathsf{NP}$ according to the Alternative Characterization above.

(e) Show that each of following decision problems is in NP by exhibiting (1) a poly-time NTM that decides it (clearly describe the non-deterministic choices), and (2) a poly-time verifier (according to the Alternative Characterization above;) clearly describe the succinct certificates.

i. Given a list of m linear inequalities with rational coefficients over n variables $u_1, \ldots, u_n$, find out if there is an assignment of zeros and ones to $u_1, \ldots, u_n$ satisfying all the inequalities.

Solution.
This is the decision version of 0/1 Integer Programming. First we define a language corresponding to this problem. We will represent rational numbers as pairs of integers: $\ell/k \mapsto (\ell, k)$, where ℓ, k are integers and $k \neq 0$. To avoid trivialities, we assume that every integer m is of the form $m/1$ and thus maps to the pair $(m, 1)$. Denote as $c_{i,j}$ the coefficient of variable u_i in the jth constraint, $i \in [n]$, $j \in [m]$, and let b_j be the RHS of the jth constraint. Wlog we shall assume that the constraints are all "$\leq$", and that $c_{i,j}$ is always given even if it is 0. Then $c_{i,j} = \ell_{i,j}/k_{i,j}$ for some integer $\ell_{i,j}$ and non-zero integer $k_{i,j}$, and $b_j = \ell_j/k_j$.
So a possible language is

$$
\begin{aligned}
L = \big\{ & \ell_{1,1}, k_{1,1} \# \ell_{2,1}, k_{2,1} \# \cdots \# \ell_{n,1}, k_{n,1} \# \# \ell_1, k_1 \$ \cdots \\
& \ell_{1,m}, k_{1,m} \# \ell_{2,m}, k_{2,m} \# \cdots \# \ell_{n,m}, k_{n,m} \# \# \ell_m, k_m \$: n, m \text{ positive integers}, \\
& \ell_{i,j}, k_{i,j}, \ell_j, k_j \text{ integers and } k_{i,j}, k_j \neq 0 \ \ \forall i \in [n], j \in [m], \\
& \exists u_1, \ldots, u_n \in \{0,1\} \text{ s.t. } \sum_{i=1}^{n} (\ell_{i,j}/k_{i,j}) u_i \leq (\ell_j/k_j) \ \ \forall j \in [m] \big\}.
\end{aligned}
$$

(1) **NTM:** We may assume that there is a DTM ADDQ that carries out rational addition, and a DTM CQ that performs rational comparison. Both ADDQ and CQ receive as inputs two rational numbers a and b, each described as pairs of integers as above. On input $\langle a, b\rangle$, ADDQ writes $c = a + b$ on its tape as a pair of integers. CQ is such that $\text{Result}(\text{CQ}, \langle a, b\rangle) = \text{yes}$ iff $a \leq b$. Let N denote our NTM. Whenever N encounters a #, it branches, writing 0 or 1 on its tape at the first available blank cell (past the right-most \$). When N reads the first \$, it starts reading the n-bits it has just written $(u_1, \ldots, u_n)$. Let $S_1 \ldots, S_m$ be counters that are initially set to 0. For each $i \in [n]$, if $u_i = 0$, then for every $j \in [m]$, N uses DTM ADDQ to add $c_{i,j}$ to S_j, then it runs CQ on $\langle S_j, b_j\rangle$ (note that S_j is a sum of rationals so it is rational): If CQ rejects, then reject and halt; otherwise continue. Our NTM accepts iff CQ never rejects.

(2) The certificate is the assignment.

ii. Given two $n \times n$ adjacency matrices M_1, M_2, decide if M_1 and M_2 define the same graph, up to renaming of vertices.

Solution. This is the Graph Isomorphism problem.

(1) **NTM:** The NTM must guess a permutation $\pi : [n] \to [n]$ such that M_2 is equal to M_1 after reordering M_2's indices according to π. A permutation may be written as a list of n pairs of integers $(1, k_1), \ldots, (n, k_n)$, where $k_i \in [n]$, $k_i \equiv \pi(i)$ for all $i \in [n]$, and $k_1, \ldots, k_n$ are distinct. We may choose to represent an adjacency matrix as a string consisting of the rows as they appear in the matrix, each separated by #; i.e., consecutive elements of the rows of the matrix appear contiguously, so the encoding string has the form

$$b_{1,1}b_{1,2}\cdots b_{1,n}\#b_{2,1}b_{2,2}\cdots b_{2,n}\#\cdots\#b_{n,1}b_{n,2}\cdots b_{n,n}\#,$$

where $b_{i,j} \in \{0, 1\}$ is the element in the ith row and jth column. This representation is called "row-major."
Initially, the NTM chooses any number $k_1 \in [n]$ with which to pair 1, so there are n choices for k_1 (branches). Suppose that the NTM has made $i - 1$ choices $k_1, \ldots, k_{i-1}$, $i \geq 2$. Then, in making its ith choice, our NTM chooses an integer k_i to pair with i such that k_i is **not** equal to any of the machine's previous choices $k_1, \ldots, k_{i-1}$; thus there are $(n - i + 1)$ possible choices for k_i. Once $k_1, \ldots, k_n$ have been chosen, our NTM checks whether $M_1(i, j) = M_2(k_i, k_j) \equiv M_2\big(\pi(i), \pi(j)\big)$ for every $i, j \in [n]$, and accepts iff the latter holds.

(2) **Verifier:** The certificate is the permutation $\pi : [n] \to [n]$ such that M_2 is equal to M_1 after reordering M_2's indices according to π. The certificate can be written as a list of n pairs $\big\langle\langle 1, k_1\rangle, \ldots, \langle n, k_n\rangle\big\rangle$, where $k_i \in [n]$, $k_i \equiv \pi(m_i)$ for all $i \in [n]$, and $k_1, \ldots, k_n$ are distinct.

5. (Reductions)

(a) Show that $\mathsf{SAT} \leq_{\mathsf{P}} \mathsf{HALT}$. Conclude that HALT is NP-hard. Is HALT NP-complete?

Solution. Let M be the Turing machine that on input $\phi = \phi(x_1, \ldots, x_n)$ tries every possible assignment of $x_1, \ldots, x_n$. If any are accepting, it accepts. Otherwise, it loops forever. Hence, M halts on ϕ if and only if ϕ is satisfiable.

By the Cook-Levin Theorem, SAT is NP-hard so HALT is also NP-hard. However, HALT cannot be in NP since it is undecidable.

(b) Let $\mathsf{CNF}_3 = \{\ \phi : \phi$ is a satisfiable cnf-formula and each variable appears in at most 3 places $\}$. Show that CNF_3 is NP-complete.

Solution. Let ϕ be a Boolean formula and initialize ψ to be the empty Boolean formula. The reduction is as follows. Go through the formula ϕ and if we find a variable x in at least two clauses, replace one of them with a new variable x' and append the condition $(x \vee \neg x') \wedge (\neg x \vee x') = (x = x')$ to ϕ'. Repeat this process until no variable appears more than twice in ϕ. Call this new formula, with no duplicated variables, ϕ'. Note that ϕ' contains each variable only once and ψ contains each variable at most twice. So $\phi' \wedge \psi$ is a cnf-formula where each variable appears in at most 3 places. Finally ϕ is satisfiable if and only $\phi' \wedge \psi$ is satisfiable since ψ only enforces that the new variables we created are equal to the original variables.

(c) Let $\mathsf{DFA}_\cap = \{ \langle D_1, \ldots, D_m \rangle : m \geq 1, D_1, \ldots, D_m \text{ are DFAs and } \cap_{i=1}^m L(D_i) \neq \emptyset \}$. Show that $\mathsf{DFA}_\cap$ is NP-hard.

6. Show that NP is closed under Kleene star. Do the same for P.

(a) Show that NP is closed under Kleene star.

Solution. Let $L \in \mathsf{NP}$. For a given input x, guess a decomposition $x = x_1 \ldots x_n$ and check that $x_i \in L$ for each $i \in [n]$.

(b) (Tricky) Do the same for P.

Solution. Proving that P is closed under Kleene star is a bit more tricky since we cannot just "guess" a decomposition. Fix $L \in \mathsf{P}$. Observe that if a string $x = x_1 \ldots x_n \in L^*$ then either $x \in L$ or there exists k for which $x_1 \ldots x_k \in L^*$ and $x_{k+1} \ldots x_n \in L^*$. Thus, dynamic programming seems like a natural approach.

It will be slightly nicer to write $x = x_0 x_1 \ldots x_n x_{n+1}$ where $x_0 = x_{n+1} = \varepsilon$. We will construct a table $V[i, j]$ where $V[i, j] = 1$ if and only $x_i \ldots x_j \in L^*$. (Note that for $j < i$, we can set $V[i, j] = 0$.) If $x_i \ldots x_j \in L$ (which we can check in polynomial time) then set $V[i, j] = 1$. Otherwise, we recurse by testing if there is some $0 \leq k \leq n$ for which $V[i, k] = V[k + 1, j] = 1$. If so, then set $V[i, j] = 1$.

We can compute $V[i, i]$ for all i in polynomial time then compute $V[i, i + 1]$ in polynomial time, etc. The overall running time is thus polynomial.