CS244 Theory of Computation
Homework 4 Solution

Problem 1

(a) Show that $A_{\mathsf{LBA}} = \{\langle B, w\rangle \mid B \text{ is an } \mathsf{LBA} \text{ that accepts input } w\}$ is PSPACE-complete.

Proof. To show that A_{LBA} is in PSPACE, recall the book's algorithm for A_{LBA}. It simulates the input LBA on the input string for qng^n steps where q is the number of states, g is the size of the tape alphabet and n is the length of the input string, by maintaining the current configuration of the LBA and a counter of the number of steps performed. Both can be stored by using polynomial space.

To show that A_{LBA} is PSPACE-hard, we give a reduction from *TQBF*. Note that *TQBF* is decidable with an LBA. The book's algorithm for *TQBF* uses linear space, and so it can run on an LBA B by encoding some constant numbers of tape cells into a single cell with a larger alphabet. Given a formula $\langle\phi\rangle$, the reduction outputs $\langle B, \langle\phi\rangle\rangle$. Therefore, $\langle\phi\rangle \in TQBF \iff B$ accepts $\langle\phi\rangle \iff \langle B, \langle\phi\rangle\rangle \in A_{\mathsf{LBA}}$. □

(b) Show that $E_{\mathsf{DFA}} = \{\langle A\rangle \mid A \text{ is a } \mathsf{DFA} \text{ and } L(A) = \emptyset\}$ is NL-complete.

Proof. To prove that E_{DFA} is NL-complete, we can show instead that E_{DFA} is coNL-complete, because NL = coNL. Showing that E_{DFA} is coNL-complete is equivalent to showing that $\overline{E_{\mathsf{DFA}}}$ is NL-complete.

First we show that $\overline{E_{\mathsf{DFA}}} \in$ NL. The NL-algorithm guesses a path along transitions from the start state to an accept state and accepts if it finds one.

Second we give a log space reduction from *PATH* to $\overline{E_{\mathsf{DFA}}}$. The reduction maps $\langle G, s, t\rangle$ to a DFA B which has a state q_i for every node v_i in G and one additional state d. DFA B has input alphabet $\{a_1, \ldots, a_k\}$ where k is the number of nodes of G. If G has an edge from v_i to v_j we put a transition in B from state q_i to q_j on input a_j. If G has no transition from v_i to v_j we put a transition from q_i to d on input a_j. In addition we put transitions from d to itself on all input symbols. The state corresponding to node s is the start state and the state corresponding to node t is the only accept state. The role of d in this construction is to make sure that every state has an exiting transition for every input symbol. We argue that this construction works by showing that G has a path from s to t iff $L(B) = \emptyset$. If G has a path $s, v_{i_1}, \ldots, v_{i_l}, t$ from s to t, B accepts the input $a_{i_1} \ldots a_{i_l}$. Similarly, if B accepts some input $a_{i_1} \ldots a_{i_l}$, then it must correspond in G to the path $s, v_{i_1}, \ldots, v_{i_l}, t$. The reduction can be carried out in log space because it is a very simple transformation from the input. □

Problem 2

Describe a deterministic, polynomial-time SAT-oracle Turing machine M^{SAT} that takes as input a directed graph G and nodes s and t, and outputs a Hamiltonian path from s to t if one exists. If none exist, then M^{SAT} outputs **No Hamiltonian path**.

Solution

Recall that $HAMPATH \in \text{NP}$ so $HAMPATH \leq_{\text{P}} SAT$.

M^{SAT} on input $\langle G, s, t \rangle$:

1. Use the reduction from $HAMPATH$ to SAT to produce a formula $\langle \phi \rangle$ and test $\langle \phi \rangle \in SAT$ using the oracle. If $\langle \phi \rangle \notin SAT$ then output **No Hamiltonian path** and halt. [poly time]
2. Output s as the first node of the path and let $r = s$. [$O(1)$]
3. Repeat until $r = t$: [$O(n)$ iterations]
4. Let $r_1, \ldots, r_k$ be the nodes that have edges from r. Permanently remove r from G and test whether a Hamiltonian path exists in the new graph G from each r_i to t using the above reduction and oracle. One of these tests must succeed. [poly time]
5. Output r_i and let $r = r_i$. [$O(1)$]

M^{SAT} is deterministic, polynomial-time.

Problem 3

Say that a probabilistic algorithm ***uses randomness r(n)*** if it uses at most $r(n)$ coin tosses on each computation thread.

(a) Recall the probabilistic algorithm for EQ_{ROBP} we presented. How much randomness does it use when it is run on two branching programs that have m input variables? Give your answer as a function of m using big-O notation. Explain your reasoning.

(b) Let $\text{BPP}[f(n)] = \{A \mid A$ is decided by a probabilistic, polynomial time TM that uses at most $O(f(n))$ randomness on all inputs of length $n\}$. Show that $\text{BPP}[\log(n)] \subseteq \text{P}$.

Solution

(a) The probabilistic algorithm we presented for EQ_{ROBP} selects one value for each of the m variables from the field F_q where q has slightly more than $3m$ elements. Each field element can be selected with $O(\log m)$ coin tosses. So the total number of coin tosses used by the algorithm is $O(m \log m)$.

(b) *Proof.* If M is a probabilistic, polynomial time TM that uses at most $O(\log n)$ coin tosses on each computation thread, we can simulate M in deterministic polynomial time by running M on each of its computation branches and computing the probability that it accepts. The total number of computation branches is at most $2^{O(\log n)} = n^k$ for some constant k. Hence the running time of the simulation is polynomial. □

Problem 4

For any positive integer x, let $x^{\mathcal{R}}$ be the integer whose binary representation is the reverse of the binary representation of x. (Assume no leading 0s in the binary representation of x.) Define the function $\mathcal{R}^+ : \mathcal{N} \to \mathcal{N}$ where $\mathcal{R}^+(x) = x + x^{\mathcal{R}}$.

(a) Let $A_2 = \{\langle x, y\rangle \mid \mathcal{R}^+(x) = y\}$. Show $A_2 \in \mathrm{L}$.

Proof. Assume x is k-bit long, then y is at most $(k+1)$-bit long. Write $x = x_{k-1} \dots x_1 x_0$ and $y = y_k \dots y_1 y_0$ where y_k can be 0. Start from the least significant bit of x to the most significant bit. Add x_i and x_{k-i-1} to get r_i the i-th least significant bit of $\mathcal{R}^+(x)$ and the carry. If $r_i \neq y_i$, *reject*. At the most significant bit, let the carry be r_k. If all $r_i = y_i$, *accept*.

The algorithm only needs space to store the index i, the result of the current bit, and the carry, which is $O(\log n)$ space. Thus $A_2 \in \mathrm{L}$. □

(b) Let $A_3 = \{\langle x, y\rangle \mid \mathcal{R}^+(\mathcal{R}^+(x)) = y\}$. Show $A_3 \in \mathrm{L}$.

Proof. Calculate $\mathcal{R}^+(x)$ as in (a) for k times, and only store the i-th and the $(k-i-1)$-th bit of $\mathcal{R}^+(x)$ instead of the entire $\mathcal{R}^+(x)$ on the i-th iteration. Add these two bits to get the i-th bit of $\mathcal{R}^+(\mathcal{R}^+(x))$ and the carry. Compare this bit to y_i. If they are not equal, *reject*. If they are equal on all iterations, *accept*.

For the calculation of the outer $\mathcal{R}^+$, we need to store the index i, the result of the current bit, and the carry. For the calculation of the inner $\mathcal{R}^+$, we need to store the index of the current bit, the result of the current bit, the carry, and another bit. Once the i-th and the $(k-i-1)$-th bit of $\mathcal{R}^+(x)$ are both calculated, the space to store the two bits will contain the i-th and the $(k-i-1)$-th bit of $\mathcal{R}^+(x)$. This only requires $O(\log n)$ space. Thus $A_3 \in \mathrm{L}$. □

Problem 5

For branching program B and $w = w_1 \dots w_m$, where each $w_i \in \{0, 1\}$, let $B(w)$ be the output of B when its input variables $x_1, \dots, x_m$ are set $x_i = w_i$ for each i.

(a) Let $ALL_{\mathsf{ROBP}} = \{\langle B\rangle \mid B \text{ is a read-once branching program and } B(w) = 1 \text{ on all } w\}$. Show that $ALL_{\mathsf{ROBP}} \in \mathrm{P}$.

Proof. In a read-once branching program B, if a path exists from the start node to the output 0 node, then $B(w) = 0$ for the corresponding input w and so $B \notin ALL_{\mathsf{ROBP}}$. If no such path exists then $B \in ALL_{\mathsf{ROBP}}$. Testing for such a path can be done in P. □

(b) Let $ALL_{\mathsf{BP}} = \{\langle B\rangle \mid B \text{ is a branching program and } B(w) = 1 \text{ on all } w\}$. Show that ALL_{BP} is coNP-complete.

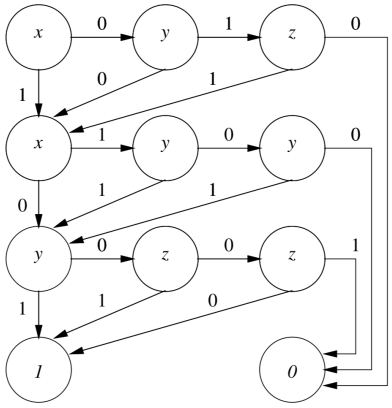
Proof. Prove equivalently that $\overline{ALL_{\mathsf{BP}}}$ is NP-complete.

First, to show $\overline{ALL_{\mathsf{BP}}} \in \mathrm{NP}$, observe that a certificate is an input w for which $B(w) = 0$, which can be checked in polynomial time.

To show that $\overline{ALL_{\mathsf{BP}}}$ is NP-hard, give a reduction from *3SAT* to $\overline{ALL_{\mathsf{BP}}}$. For 3cnf boolean formula ϕ, construct a branching program B_ϕ that computes the same boolean function. The nodes of B_ϕ correspond to each occurrence of a literal in ϕ, grouped by clauses. The computation of B_ϕ flows

Problem 5 continued on next page...

through these groups. From each group, B_ϕ goes to the output 0 node whenever it detects an unsatisfied clause, and B_ϕ goes to the first node in the next clause group when it determines that the current clause is satisfied. B_ϕ is roughly as large as ϕ and can be computed in polynomial time. The details are illustrated in the following example. Let $\phi = (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee y \vee y) \wedge (y \vee z \vee \overline{z})$. Then B_ϕ is:



By interchanging the 0 and 1 output nodes, B_ϕ becomes a branching program which outputs 1 on all w whenever ϕ is unsatisfiable. That completes the reduction. □

Problem 6

Prove that if $A \subseteq \{0,1\}^*$ is a regular language, a family of branching programs $(B_1, B_2, \dots)$ exists where each B_n accepts exactly the strings in A of length n and is bounded in size by a constant times n.

Proof. Let $\Sigma = \{0,1\}$ and let DFA $D = (Q, \Sigma, \delta, q_0, F)$ recognize a language A. Construct B_n, the n-th member of a family of branching programs $(B_1, B_2, \dots)$ as follows. The nodes of B_n are (q, i) for $q \in Q$ and $1 \le i \le n$, and those nodes are labeled to read x_i, the i-th symbol of the input x. Additionally, B_n has two output nodes **Accept** and **Reject**. Draw an edge with label $a \in \Sigma$ from (q, i) to $(r, i+1)$ if $\delta(q, a) = r$ for $1 \le i < n-1$, and from $(q, n-1)$ to **Accept** if $\delta(q, a) \in F$ and from $q, n-1)$ to **Reject** if $\delta(q, a) \notin F$. The start node is $(q_0, 1)$. The number of nodes of B_k is $|Q| \times n = O(n)$, and thus satisfies the size requirement. □