G22.3520: Honors Analysis of Algorithms

Problem Set 6+7

Solutions for most of the problems

For all problems, the alphabet $\Sigma = \{0, 1\}$.

Problem 1

Solve Problem 4 from: http://www.cs.nyu.edu/courses/fall07/G22.3520-001/ps2.pdf

Note: This is the problem about the algorithm $A(S)$.

Problem 2 (Pumping Lemma for regular languages)

Consider a DFA with n states and suppose x is a string accepted by the DFA with $|x| > n$. Show that x can be written as $x = uvw$ (i.e. concatenation of three strings u, v, w) such that $|v| > 0$ and uv^iw is accepted by the DFA for every integer $i \geq 0$ (i.e. concatenation of u, v repeated i times, and w).

Using this lemma, prove that $\{0^k1^k \mid k \geq 1\}$ is not a regular language.

Proof: Consider the path of length $|x|$ that takes the DFA from the starts state q_0 to an accept state q_f. Since $|x| > n$, there must be a state q that repeats itself on the path. Write $x = uvw$ where u takes the DFA from q_0 to q, then v takes the DFA from q to itself, and then w takes the DFA from q to q_f. Clearly, $|v| > 0$, and for every $i \geq 0$, the string uv^iw takes the DFA from q_0 to q_f, and hence is accepted by the DFA.

Now suppose on the contrary that $L = \{0^k1^k \mid k \geq 1\}$ is regular and is accepted by a DFA with n states. Consider the string $x = 0^n1^n \in L$. By the pumping lemma, one can write $x = uvw, |v| > 0$ and $uv^iw \in L \;\forall\; i \geq 0$. Note however that this is a contradiction since no matter what the partition $x = uvw$ is, the string $uv^2w \notin L$. Depending on what/where v is, the string uv^2w either contains more zeros than ones, or more ones than zeros, or a substring of type 011*0.

Problem 3

The standard procedure for converting an NFA to an equivalent DFA yields an exponential blowup in the number of states. That is, if the original NFA has n states, then the resulting DFA has 2^n states. In this problem, you will show that such an exponential blowup is necessary in the worst case.

Define $L_n = \{w :$ The nth symbol from the right is $1\}$.

1. Give an NFA with $n + 1$ states that recognizes L_n.
2. Prove that any DFA with fewer than 2^n states cannot recognize L_n. *Hint: Let M be any DFA with fewer than 2^n states. Start by showing that there exist two different strings of length n that drive M to the same state.*

Proof: Let M be any DFA with fewer than 2^n states. We will show that M cannot recognize L_n. Since there are 2^n strings of length n, by the Pigeonhole Principle, there are two different strings $x = x_1x_2\cdots x_n$ and $y = y_1y_2\cdots y_n$ that drive M to the same state. Since $x \neq y$, there is some i such that $x_i \neq y_i$. Without loss of generality, say that $x_i = 1$ and $y_i = 0$. Let $x' = x0^{i-1}$ and $y' = y0^{i-1}$. It is easy to see that $x' \in L_n$ and $y' \notin L_n$. However, since x and y drive M to the same state, it is clear that $x' = x0^{i-1}$ and $y' = y0^{i-1}$ also drive M to the same state, yet $x' \in L_n$ and $y' \notin L_n$. Therefore M cannot recognize L_n.

Problem 4

If A is a language, let $A_{-\frac{1}{2}}$ be the set of all first halves of strings in A, i.e.

$$A_{-\frac{1}{2}} = \{x \mid \text{for some } y, |x| = |y|, \text{and } xy \in A\}.$$

Show that if A is regular, so is $A_{-\frac{1}{2}}$. *Hint: Run two DFAs "in parallel", one forward and the other backward.*

Solution: Given a DFA M that accepts A, we construct an NFA M' that accepts $A_{\frac{1}{2}-}$. The basic idea is as follows: to decide whether a string $x \in A_{\frac{1}{2}-}$, we non-deterministically choose y such that $|x| = |y|$. We simulate M on x and at the same time simulate M backwards on string y. The simulation on x starts with the start state of M (call it q_0) whereas the simulation on y starts with some accept state of M (call it q_f). We accept iff both simulations reach the same state of M (call it q). Thus we accept iff x takes the DFA from q_0 to q and y takes it from q to q_f. Since q_f is an accept state of M, we ensure that $xy \in A$. The simultaneous simulation on x and y is carried out by the cartesian product construction, similar to proof of Theorem 1.2 in Sipser's book.

Formally, if $(Q, \Sigma, \delta, q_0, F)$ describes the DFA M, then the NFA $M' = (Q', \Sigma, \delta', q_{start}, F')$ is defined as follows:

- $Q' = Q \times Q \cup \{q_{start}\}$.
- $F' = \{(q, q) \mid q \in Q\}$.
- There is ϵ-move from q_{start} to all the states in $\{(q_0, q_f) \mid q_f \in F\}$. These are the only moves possible from q_{start}.
- There is a move from (q_1, q_2) to (q_3, q_4) on input symbol $a \in \Sigma$ iff $\delta(q_1, a) = q_3$ and $\delta(q_4, b) = q_2$ for some $b \in \Sigma$. Formally,

$$(q_3, q_4) \in \delta'((q_1, q_2), a) \quad \text{iff} \quad \delta(q_1, a) = q_3 \text{ and } \exists\, b \in \Sigma \text{ s.t. } \delta(q_4, b) = q_2$$

Problem 5

For a language A, let SUFFIX(A) denote the set of all suffixes of strings in A, i.e.

$$\text{SUFFIX}(A) = \{v \mid uv \in A \text{ for some string } u\}.$$

Show that if A is a context-free language, so is SUFFIX(A). *Hint: Design a PDA.*

Solution: Let A be a context-free language recognized by a PDA M. We will construct a PDA R that recognizes SUFFIX(A). On input v, the PDA R works in two phases. The first phase operates without looking at the input. The PDA non-deterministically generates a symbol $a \in \Sigma$ and simulates (one or more) steps of M until the symbol a is read. The PDA repeats this sequence of moves (as many times as it wishes). It non-deterministically decides when to switch to second phase. In the second phase, the PDA looks at the input v and simulates M on v.

Note that R accepts v if and only if there exists $u \in \Sigma^*$ such that M accepts uv. The string u (and its length!) is "guessed".

Problem 6

Show that the following languages are decidable:

1. $\text{INFINITE}_{\text{DFA}} = \{\langle D\rangle \mid D \text{ is a DFA such that } L(D) \text{ is infinite}\}$.
2. $L = \{\langle R, S\rangle \mid R \text{ and } S \text{ are regular expressions such that } L(R) \subseteq L(S)\}$.

Solution: Using the proof of pumping lemma from Problem 2, it is clear that $L(D)$ is infinite if and only if there is a state q of the DFA such that (i) q is reachable from the start state q_0 (ii) some accept state is reachable from q (iii) q is on some non-trivial cycle. All three conditions can be checked for every state q using standard algorithms.

Solution: We observe that $L(R) \subseteq L(S)$ if and only $L(R) \cap \overline{L(S)} = \emptyset$. Therefore the following TM decides

$$L = \{\langle R, S\rangle : R \text{ and } S \text{ are regular expressions and } L(R) \subseteq L(S)\}\,.$$

On input $\langle R, S\rangle$:

1. Construct a DFA D such that $L(D) = L(R) \cap \overline{L(S)}$.
2. Run a TM T that decides E_{DFA} on input $\langle D\rangle$ (i.e. decides whether $L(D)$ is empty. We did this in class).
3. If T accepts, output ACCEPT. If T rejects, output REJECT.

Problem 7

In this problem, we explore the notion of *oracle reducibility.* If A is a language, then a *Turing machine with oracle* A is a Turing machine with a "magical" subroutine that decides membership in A. In other words, the subroutine, when given a string w, tells the machine whether or not $w \in A$. Let

$$\text{HALT}_{\text{TM}} = \{\langle M, x\rangle \mid M \text{ is a Turing machine that halts on } x\}.$$

Show that there is a Turing machine with oracle HALT_{TM} that decides the following problem with only *two* questions to the oracle: Given three (machine, input) pairs $\langle M_1, x_1\rangle, \langle M_2, x_2\rangle, \langle M_3, x_3\rangle$, decide for each pair whether the Turing machine halts on the corresponding input.

Note: This is trivial if one allows *three* questions. Just ask the oracle whether $\langle M_i, x_i\rangle \in \text{HALT}_{\text{TM}}$ for $i = 1, 2, 3$.

Solution: Suppose that exactly k of the machines $\{M_1, M_2, M_3\}$ halt on corresponding inputs. The idea is to figure out the value of k. Once we know k, we can decide which ones halt: simply simulate the three machines on corresponding inputs simultaneously (one step of every machine at a time), and halt when exactly k of them halt. Since we know k, we are guaranteed to halt.

This is how we figure out the value of k. First construct a machine M that simulates the three machines on corresponding inputs simultaneously and halts if at least two of them halt. Now decide whether M halts by asking the oracle.

(Case 1): If M halts, we know that $k \geq 2$. By running the machines simultaneously again, we figure out which two of the machines halt. Then by asking the oracle, we can figure out whether the third machine halts.

(Case 2): If M does not halt, we know that $k \leq 1$. Construct a machine M' that simulates the three machines on corresponding inputs simultaneously and halts if at least one of them halts. Decide whether M' halts by asking the oracle. If M' halts, then we know that $k = 1$. If M' does not halt, we know that $k = 0$.

Problem 8

Show that the collection of Turing-recognizable languages is closed under the operation of (a) union (b) concatenation (c) star and (d) intersection. What about complementation operation ?

Solution:

(a) Let L_1 and L_2 be two Turing-recognizable languages, and let M_1 and M_2 be TMs that recognizes L_1 and L_2 respectively. We construct a TM M that recognizes $L_1 \cup L_2$: On input w,

1. Run M_1 and M_2 on w "in parallel". That is, in each step, M runs one step of M_1 and one step of M_2.
2. If either of M_1 and M_2 accepts, output ACCEPT. If both reject, output REJECT.

If M_1 or M_2 accepts w, then M will halt and accept w since M_1 and M_2 are run in parallel and an accepting TM will halt and accept w in a finite number of steps. If both M_1 and M_2 reject w, then M will reject w. If neither M_1 nor M_2 accepts w and one of them loops on w, then M will loop on w. Thus $L(M) = L_1 \cup L_2$, and Turing-recognizable languages are closed under union.

(b) Let L_1 and L_2 be two Turing-recognizable languages, and let M_1 and M_2 be TMs that recognizes L_1 and L_2 respectively. We construct a NTM N that recognizes $L_1 \circ L_2$: On input w,

1. Nondeterministically split w into two parts $w = xy$.
2. Run M_1 on x. If it rejects, halt output REJECT. If it accepts, go to Step 3.
3. Run M_2 on w. If it accepts, output ACCEPT. If it rejects, output REJECT.

If $w \in L_1 \circ L_2$, then there is a way to split w into two parts $w = xy$ such that $x \in L_1$ and $y \in L_2$, thus, M_1 halts and accepts x, and M_2 halts and accepts y. Hence at least one branch of N will accept w. On the other hand, if $w \notin L_1 \circ L_2$, then for every possible splitting $w = xy$, $x \notin L_1$ or $y \notin L_2$, so M_1 does not accept x or M_2 does not accept y. Thus, all branch of N will reject w. Therefore, $L(N) = L_1 \circ L_2$, and Turing-recognizable languages are closed under concatenation.

(c) Let L be a Turing-recognizable languages, and let M be a TM that recognizes L. We construct a TM M' that recognizes L^*: On input w,

1. Nondeterministically choose an integer k and split w into k parts $w = w_1 w_2 \cdots w_k$.
2. Run M on each w_i. If M accepts all of $w_1, \ldots, w_k$, output ACCEPT. If M rejects one of $w_1, \ldots, w_k$, output REJECT.

If $w \in L^*$, then there is a splitting of $w = w_1 w_2 \ldots w_k$ such that $w_i \in L$ for each i, and thus at least one branch of M' will accept w. On the other hand, if $w \notin L^*$, then for every possible splitting $w = w_1 w_2 \ldots w_k$, $w_i \notin L$ for at least one i, thus all branches of M' will reject w. Therefore, $L(M') = L^*$, and Turing-recognizable languages are closed under the star operation.

(d) Let L_1 and L_2 be two Turing-recognizable languages, and let M_1 and M_2 be TMs that recognizes L_1 and L_2 respectively. We construct a TM M that recognizes $L_1 \cap L_2$: On input w,

1. Run M_1 on w. If it rejects, halt and output REJECT. If it accepts, go to Step 2.
2. Run M_2 on w. If it accepts, output ACCEPT. If it rejects, output REJECT.

Clearly $L(M) = L_1 \cap L_2$, and thus Turing-recognizable languages are closed under intersection.

(e) Not closed under complementation. Suppose it were true that whenever L is Turing-recognizable, so is $\overline{L}$. So whenever L is Turing-recognizable, then both $L, \overline{L}$ would be Turing-recognizable, and hence L would be decidable (as proved in class). This is a contradiction since there exists a language that is Turing-recognizable but not decidable, e.g. the language A_{TM}.

Problem 9

Show that every context free language is in P. In other words, for any fixed grammar G, design a polynomial time algorithm to test whether a given string w is generated by the grammar. The algorithm should run in time polynomial in $|w|$. The size of the grammar itself is thought of as a constant. *Hint: Use dynamic programming. Assume w.l.o.g. that the grammar is in Chomsky normal form.*

Problem 10

Show that P is closed under the star operation (*Hint: Use dynamic programming.*) Recall that for a language L,

$$L^* = \{x_1 x_2 \ldots x_k \mid k \geq 0,\ x_i \in L\ \forall 1 \leq i \leq k\}$$

Solution: We show that **P** is closed under the star operation by dynamic programming. Let $L \in \mathbf{P}$, and let A be a polynomial-time algorithm that decides L. We construct a polynomial-time algorithm B that decides L^*, as follows. On input $w = w_1 \cdots w_n$, algorithm B constructs a table T such that $T[i,j] = 1$ if $w_i \cdots w_j \in L^*$, and $T[i,j] = 0$ otherwise. Details follow.

Algorithm B:

On input $w = w_1 \cdots w_n$,

1. If $w = \varepsilon$, output ACCEPT and halt.

```
2. Initialize $T[i,j] = 0$ for each $0 \le i \le j \le n$.
3. For $i = 1$ to $n$: [1]
   (a) Run $M$ on $w_i$ to decide whether $w_i \in L$.
   (b) If $w_i \in L$, then set $T[i,i] = 1$.
4. For $\ell = 2$ to $n$: [2]
   For $i = 1$ to $n - \ell + 1$: [3]
   (a) Let $j = i + \ell - 1$. [4]
   (b) Run $M$ on $w_i \cdots w_j$ to decide whether $w_i \cdots w_j \in L$.
   (c) If $w_i \cdots w_j \in L$, then set $T[i,j] = 1$.
   (d) For $k = i$ to $j - 1$: [5]
          If $T[i,k] = 1$ and $T[k,j] = 1$, then set $T[i,j] = 1$.
5. Output ACCEPT if $T[1,n] = 1$; else output REJECT. [6]
```

It is not hard to see that Algorithm B correctly decides L^* provided that Algorithm A correctly decides L. Moreover, Algorithm B runs in $O(n^3)$ stages, and each stage takes polynomial-time as A runs in polynomial-time. Therefore, Algorithm B is a polynomial-time algorithm that decides L^*.

Problem 11 (Do not submit)

Show that these problems are NP-complete (you can assume that SAT and CLIQUE are NP-complete).

1. DOUBLE-SAT $= \{\langle \phi \rangle \mid \phi$ is a boolean formula that has at least two satisfying assignments$\}$.
2. HALF-CLIQUE $= \{\langle G \rangle \mid G$ is a graph with a clique of size at least $|G|/2\}$.

Solution: Clearly Double-SAT $\in$ **NP**, since a NTM can decide Double-SAT as follows: On input a Boolean formula $\varphi(x_1, \ldots, x_n)$, nondeterministically guess 2 assignments and verfify whether both satisfy φ. To show that Double-SAT is **NP-Complete**, we give a reduction from SAT to Double-SAT, as follows:

On input $\varphi(x_1, \ldots, x_n)$:

1. Introduce a new variable y.
2. Output formula $\varphi'(x_1, \ldots, x_n, y) = \varphi(x_1, \ldots, x_n) \wedge (y \vee \overline{y})$.

[1]Test each substring of length 1
[2]ℓ is the length of the substring
[3]i is the start position of the substring
[4]j is the end position of the substring
[5]k is the split position
[6]$T[1,n] = 1$ if and only if $w = w_1 \cdots w_n \in L^*$.

If $\varphi(x_1, \ldots, x_n) \in SAT$, then φ has at least 1 satisfying assignment, and therefore $\varphi'(x_1, \ldots, x_n, y)$ has at least 2 satisfying assignments as we can satisfy the new clause $(y \vee \overline{y})$ by assigning either $y = 1$ or $y = 0$ to the new variable y, so $\varphi'(x_1, \ldots, x_n, y) \in$ Double-SAT. On the other hand, if $\varphi(x_1, \ldots, x_n) \notin SAT$, then clearly $\varphi'(x_1, \ldots, x_n, y) = \varphi(x_1, \ldots, x_n) \wedge (y \vee \overline{y})$ has no satisfying assignment either, so $\varphi'(x_1, \ldots, x_n, y) \notin$ Double-SAT. Therefore, SAT $\leq_P$ Double-SAT, and hence Double-SAT is **NP-Complete**.

Solution: Clearly Half-CLIQUE $\in$ **NP**, since a NTM can decide Half-CLIQUE as follows: On input a graph G, nondeterministically choose at least $n/2$ nodes and verfiy whether they form a clique. To show that Half-CLIQUE is **NP-Complete**, we give a reduction from CLIQUE to Half-CLIQUE, as follows:

On input $\langle G, k \rangle$, where G is a graph on n vertices and k is an integer:

1. If $k = n/2$, then output $\langle G \rangle$.
2. If $k < n/2$, then construct a new graph G' by adding a complete graph with $n - 2k$ vertices and connecting them to all vertices in G, and output $\langle G' \rangle$.
3. If $k > n/2$, then construct a new graph G'' by adding $2k - n$ isolated vertices[7] to G, and output $\langle G'' \rangle$.

When $k = n/2$, it is clear that $\langle G, n/2 \rangle \in$ CLIQUE if and only if $\langle G \rangle \in$ Half-CLIQUE.

When $k < n/2$, if G has a k-clique, then G' has a clique of size $k + (n - 2k) = (2n - 2k)/2$, and therefore $\langle G' \rangle \in$ Half-CLIQUE as G' is a graph with $2n - 2k$ vertices. Conversely, if $\langle G' \rangle \in$ Half-CLIQUE, that is, if G' has a clique of size $n - k = k + (n - 2k)$, then at most $n - 2k$ of the clique come from the $n - 2k$ new vertices. Therefore the remaining at least k vertices form a clique in G, and hence $\langle G, k \rangle \in$ CLIQUE.

When $k > n/2$, if G has a k-clique, then G'' has a clique size $k = 2k/2$, and therefore $\langle G' \rangle \in$ Half-CLIQUE as G'' is a graph with $n + 2k - n = 2k$ vertices. Conversely, if $\langle G'' \rangle \in$ Half-CLIQUE, that is, if G'' has a clique of size k, then the clique does not contain any of the new vertices as they are isolated. Thus the clique is a k-clique of G, and hence $\langle G, k \rangle \in$ CLIQUE.

Problem 12

Let $\{x_1, x_2, \ldots, x_n\}$ be boolean variables. A literal is either a variable or its negation, i.e. x_i or $\overline{x}_i$. A clause is logical OR of one or more distinct literals. The size of a clause is the number of literals in it. A 2CNF formula ϕ is a collection of m clauses (possibly with repetition),

$$\phi = (C_1, C_2, \ldots, C_m)$$

where each C_i is of size at most two. Let MAX-2SAT be the following decision problem: Given a pair (ϕ, k) where ϕ is a 2CNF formula with n variables and m clauses, and k is a positive integer such that $k \leq m$, decide whether there exists an assignment to the n boolean variables that satisfies at least k clauses. Show that MAX-2SAT is NP-complete, by giving a polynomial time reduction from VERTEX COVER.

[7] An isolated vertice is one which is not adjacent to any other vertex.

Recall that VERTEX COVER is a problem where, given an undirected graph $G = (V, E)$, with $|V| = n$, and given a positive integer $\ell \leq n$, one needs to decide whether G has a vertex cover of size at most ℓ. A vertex cover is a subset $V' \subseteq V$ such that for every edge in E, at least one of its endpoints is included in V'.

Hint: To every vertex in the graph, assign a boolean variable which is intended to be TRUE if and only if the vertex is included in the vertex cover. Add clauses of size two corresponding to the edges, and clauses of size one corresponding to the vertices. The clauses corresponding to edges may need to be repeated a number of times.

Solution: It appears as Problem-2 here:
http://www.cs.nyu.edu/web/Academic/Graduate/exams/sample/algs_f07_ans.pdf