Practice Final

1. (Sipser 1.45) Let $A/B = \{w \mid wx \in A \text{ for some } x \in B\}$. Show that if A is regular and B is any language, then A/B is regular.

   Solution Outline: Let $M = (Q, \Sigma, \delta, q_0, F)$ be the DFA for A, where Σ is the union of the alphabets for A and B. We define F' as

   $$F' = \{q \in Q \mid \exists x \in B \; s.t. \; M \text{ goes from } q \text{ to some state in } F \text{ on reading } x\}$$

   Then $M' = (Q, \Sigma, \delta, q_0, F')$ is a DFA for A/B. Note that it might be hard to *construct* M' depending on how hard it is to decide B, but we are only required to show its existence.

2. Let M be a 1-tape Turing machine with q states, and let w be a string of length n. Prove that if on input w the machine M does not move its head left in the first $n + q + 1$ steps, then it *never* moves its head left on this input.
   Clarification: Assume that the machine only moves its head left or right (i.e. it cannot choose to stay put).

   Solution Outline: After $n - 1$ steps, the head will have moved across the entire input and the machine will just read blank cells for the next $q + 2$ steps (since its head is only moving right). However, then there must exist a state q_0 such that the machine enters q_0 twice during these $q + 2$ steps. But then, its configuration is exactly the same when it comes to q_0 the second time as it was the first time ($\textit{input cell} = \textit{blank}, \textit{state} = q_0$). But if reading a blank cell on q_0 brings the machine back to q_0 again, it will go into an infinite loop. Since the machine moved its head right during the first run of this loop, it will always move its head right subsequently.

3. A boolean formula is said to be in Monotone 2-CNF if it is the conjunction of clauses, each of which has exactly 2 literals and all the literals in the formula are positive (i.e. no negations). Note that such a formula can be easily satisfied by setting all variables to `true`.

   Consider the following version of the satisfiability problem for Monotone 2-CNF formulas:

   $$k - MON - 2SAT = \{\langle \phi, k \rangle \mid \phi \text{ is in Monotone 2-CNF and can be satisfied by setting at most } k \text{ variables to true}\}$$

   Prove that k-MON-2SAT is **NP**-complete.

   Solution Outline: k-MON-2SAT is easily seen to be in NP, since given an assignment with at most k variables set to `true`. we can easily verify if it satisfies the formula. To see the NP-hardness, we reduce VERTEX COVER to k-MON-2SAT. Let $G = (V, E)$ be a graph. For each vertex $v \in V$, we define a variable x_v (with the intention that $x_v =$ `true` iff v is in the vertex cover). Since, for each edge (u, v), at least one vertex must be in the vertex cover, we add the clauses $(x_u \vee x_v)$ for each edge $(u, v) \in E$. The formula φ is thus given by

   $$\varphi = \bigwedge_{(u,v)\in E} (x_u \vee x_v)$$

   Then the formula φ has a satisfying assignment with k variables set to true if and only if G has a vertex cover of size k.

4. Define

$$\text{CYCLE-LENGTH} = \{\langle G, c\rangle \mid 3 \le c \le |V(G)|, G \text{ is a directed graph and the length of the shortest cycle in } G \text{ is } c.\}$$

Prove that CYCLE-LENGTH is **NL**-complete.

SOLUTION OUTLINE: To see the **NL** hardness, we reduce an instance of PATH to CYCLE-LENGTH. Given $\langle G = (V, E), s, t\rangle$ as an instance of PATH, we construct n copies $G_1, \dots, G_n$ of the graph G. However, we delete all the edges within each copy and instead add the edges $(u_i, v_{i+1})\ \forall (u,v) \in E \forall i \in \{1, \dots, n-1\}$ and $(u_i, u_{i+1})\ \forall u \in V \forall i \in \{1, \dots, n-1\}$. Thus, we connect each vertex in the ith copy to itself and all its neighbors in the $(i+1)$th copy. Note that this new graph (call it H) has no cycles (since all edges go into a higer numbered copy) . Finally, we add the edge (t_n, s_1). This edge will create a cycle (of length n) if and only if it is possible to reach t_n from s_1 in H. But then, because of the way edges were added, this also gives a path from s to t in G of length at most n. Thus, G has an $s-t$ path if and only if the shortest cycle in H is of lenght n.

To see that CYCLE-LENGTH $\in$ **NL**, consider the following languages:

- $A_1 = \{\langle G, c_1\rangle \mid G \text{ has a cycle of length at most } c_1\}$
- $A_2 = \{\langle G, c_2\rangle \mid G \text{ has no cycle of length less than } c_2\}$

$A_1 \in$ **NL** since we can guess a cycle of length c_1 by moving from vertex to vertex. Also, $A_2 = \overline{A_2} \in$ **coNL** $=$ **NL**. Since $\langle G, c\rangle in$CYCLE-LENGTH if and only if $\langle G, c\rangle \in A_1$ and $\langle G, c-1\rangle \in A_2$, we have CYCLE-LENGTH $\in$ **NL**.

5. Consider the language

$$EQ_{NFA} = \{\langle N, N'\rangle \mid N, N' \text{ are NFAs with the same alphabet and } L(N) = L(N')\}$$

Show that $EQ_{NFA} \in$ **PSPACE**.
(*Hint:* Can you convert this to an appropriate reachability problem?)

SOLUTION OUTLINE: Suppose N and N' both have at most n states. We can then convert them into DFAs D_N and $D_{N'}$ with at most $m = 2^n$ states each using space polynomial in n. Finally, we can construct a DFA S, which is the product of D_N and $D_{N'}$ (with at most $m^2 = 2^{2n}$ states) and accepts $L(D_N)\Delta L(D_{N'})$ (strings that are in exactly one of the languages). Now, $L(N) = L(N')$ iff $L(S) = \emptyset$ i.e. none of the final states are reachable from the start state in S.

Since this is a reachability problem, it can be decided nondeterministically using space logarithmic in the size of the graph (because $PATH \in$ **NL**). Thus, this problem can be decided in $NSPACE(\log(m^2)) = NSPACE(n) \subseteq SPACE(n^2) \subseteq$ **PSPACE**.

Madhur's Note: Please ingnore the next problem. I think there is a mistake in the problem as stated - apologies.

6. We define the class Universal Simulator Perfect Zero-Knowledge (USPZK) as the class of zero knowledge protocols for which there is a single universal simulator U, which given the input to the protocol and the code of the any verifier, simulates the verifier's view of the interaction.

Sipser gives the following interactive protocol for Graph Non-Ispmorphism, which is is actually in Honest Verifier Perfect Zero Knowledge:

INPUT: Two graphs G_1 and G_2.
Verifier: Picks a random $i \in \{1, 2\}$ and a random permutation π. Sends $H = \pi(G_i)$.
Prover: Sends i i.e. identifies if H is a permutated copy of G_1 or G_2.

Prove that if the above protocol is in USPZK i.e. there exists a single universal simulator for all verifiers (not just honest ones), then there is a randomized polynomial time algorithm for Graph Isomorphism.