Readings: Sipser, Sections 8.5, 8.6, and 10.2.

Problem 1: (Sipser 8.13) Show that TQBF restricted to formulas where the part following the quantifiers is in conjunctive normal form is still PSPACE-complete.

Problem 2: (Sipser 8.27) Recall that a directed graph is strongly connected if every two nodes are connected by a directed path in each direction. Let

\[ \text{STRONGLY-CONNECTED} = \{ (G) \mid G \text{ is a strongly connected graph} \} \]

Show that STRONGLY-CONNECTED is NL-complete.

Problem 3: This problem uses the ideas in the proof of Theorem 8.27.

Describe a nondeterministic log-space Turing machine \( M \) that decides the language

\[ L = \{ (G, s, m, k) \mid G \text{ is a directed graph}, s \text{ is a node in } G, m, k \in \mathbb{N}, \text{ and exactly } m \text{ nodes of } G \text{ are reachable from } s \in G \text{ by paths consisting of at most } k \text{ edges} \} \]

That is, if exactly \( m \) nodes are reachable from \( s \in G \) by paths of length at most \( k \), then \( M \) must accept \( (G, s, m, k) \) on some computation path. On the other hand, if more or fewer than \( m \) nodes are reachable from \( s \in G \) by paths of length at most \( k \), then \( M \) must reject \( (G, s, m, k) \) on all computation paths.

Explain why your Turing machine \( M \) works correctly and why it works in log space.

Problem 4: Define the language class PP as follows: A language \( L \in \text{PP} \) if and only if there exists a probabilistic polynomial time Turing machine such that:

- If \( w \in L \), then \( \Pr[M \text{ accepts } w] \geq \frac{1}{2} \).
- If \( w \notin L \), then \( \Pr[M \text{ accepts } w] < \frac{1}{2} \).

Prove that:

1. \( \text{BPP} \subseteq \text{PP} \).
2. \( \text{NP} \subseteq \text{PP} \).
3. \( \text{PP} \subseteq \text{PSPACE} \).

Hint for (2): Consider a nondeterministic TM for \( L \), and replace rejections with probabilistic decisions.

Problem 5: The class RP is the class of languages \( L \) for which there is a probabilistic Turing machine \( M \) that always terminates in polynomial time, and such that for all \( w \notin L \), \( M \) always reject \( w \), and for all \( w \in L \), \( M \) accepts \( w \) with probability at least \( \frac{1}{2} \). The class coRP is the class of languages whose complement is in RP.

So far, we have only discussed machines that always terminate in polynomial time, but that give a correct answer only with some probability. Here we consider machines that always give the right answer when they terminate, but that run in time that is only polynomial on average.

Show that for any language \( L \in \text{RP} \cap \text{coRP} \) there is a probabilistic Turing machine \( M \) that runs in expected polynomial time (i.e., the expected number of steps until \( M \) terminates is bounded by a polynomial), and that when \( w \) terminates it accepts if and only if \( w \in L \).
This class \( \text{RP} \cap \text{coRP} \) is called ZPP for “zero probability polynomial”.

**Problem 6:** (Fermat’s test) Sipser 10.15. Prove Fermat’s little theorem. That is, prove that

If \( p \) is prime, and \( a \in \mathbb{Z}_p^+ \), then \( a^{p-1} \equiv 1 \pmod{p} \)

(Hint: Consider the sequence \( a, a^2, \ldots \). What must happen, and how?)

**Problem 7:** (Branching program example) Show that the majority function can be computed by a branching program that has \( O(n^2) \) nodes.

**Problem 8:** (Branching program equivalence test)

1. Give a read-once branching program \( B_1 \) that computes the function of three Boolean variables, \( x_1, x_2, \) and \( x_3 \), that has value 1 if and only if exactly one or exactly three of the variables have value 1.

2. Give a different read-once branching program \( B_2 \) that computes the same function as in part (a).

3. Compute the polynomials \( p_1 \) and \( p_2 \) associated with the output 1 box for programs \( B_1 \) and \( B_2 \), respectively, using the rules given in Sipser’s book, p. 378.

4. Choose arbitrary values from \( \mathbb{Z}_7 \) for the three variables, and evaluate \( p_1 \) and \( p_2 \) to check that they indeed give the same result.