1. Define the language

\[ \text{ShortestPath} = \{(G, k, s, t) \mid \text{the shortest path from } s \text{ to } t \text{ in } G \text{ has length } k\} \]

(a) Prove that \( \text{ShortestPath} \) is in \( \text{NL} \). (15 points)

(b) Prove that \( \text{ShortestPath} \) is in \( \text{L} \) if and only if \( \text{L} = \text{NL} \). (15 points)

(a) Solution 1: We construct a \( \text{NL} \)-machine for \( \text{ShortestPath} \) as follows: on input \( \langle G, k, s, t \rangle \), first compute \( r_{k-1} \) (the number of vertices reachable from \( s \) in at most \( k - 1 \) steps). Then, on input \( \langle G = (V, E), k, s, t \rangle \) and \( r_{k-1} \) on the work tape,

\[
\begin{align*}
  d &\leftarrow 0 \\
  \text{flag} &\leftarrow \text{FALSE} \\
  \text{for all } w &\in V \text{ do} \\
  &\quad p \leftarrow s \\
  &\quad \text{for } i \leftarrow 1 \text{ to } k - 1 \text{ do} \\
  &\quad \quad \text{non-deterministically pick a neighbor } q \text{ of } p \\
  &\quad \quad \text{if } p = w \text{ then} \\
  &\quad \quad \quad d \leftarrow d + 1 \\
  &\quad \quad \quad \text{if } w = t \text{ reject} \\
  &\quad \quad \quad \text{if } w \text{ is a neighbor of } t \text{ then} \\
  &\quad \quad \quad \quad \text{flag} \leftarrow \text{TRUE} \\
  &\quad \quad \text{if } d < r_{k-1} \text{ reject} \\
  &\quad \text{if flag then accept else reject}
\end{align*}
\]

(b) Solution 2: Observe that \( \text{NL} \) is closed under intersection, and that \( \text{ShortestPath} = L_1 \cap L_2 \) where \( L_1 = \{ \langle G, k, s, t \rangle \mid \text{there is a path from } s \text{ to } t \text{ of length at most } k \} \) and \( L_1 = \{ \langle G, k, s, t \rangle \mid \text{there is no path from } s \text{ to } t \text{ of length at most } k - 1 \} \). On the other hand, it is clear that \( L_1 \in \text{NL} \) and that \( L_2 \in \text{coNL} = \text{NL} \).

(b) Solution 1: It suffices to prove that \( \text{PATH} \leq_L \text{ShortestPath} \), since \( \text{NL} = \text{coNL} \) and \( \text{PATH} \) is \( \text{coNL} \)-complete. Given an instance \( \langle G, s, t \rangle \) of \( \text{PATH} \), the log-space transducer for this reduction outputs \( \langle G', n + 1, s, t \rangle \) where \( n \) is the number of vertices in \( G \), and \( G' \) is constructed from \( G \) by adding \( n \) new vertices and a path from \( s \) to \( t \) of length \( n + 1 \) that goes through these new vertices.

Solution 2: If \( \text{ShortestPath} \in \text{L} \), then we can solve \( \text{PATH} \) in logarithmic space by invoking the logarithmic space machine for \( \text{ShortestPath} \) for \( k \) from 0 to the number of vertices in the graph.

2. Define the language

\[ \#\text{SAT} = \{ \langle \varphi, k \rangle \mid \varphi \text{ is a 3CNF that has precisely } k \text{ satisfying assignments} \} \]

Prove that if \( \#\text{SAT} \in \text{NP} \) then \( \text{NP} = \text{coNP} \). (20 points)
First, it is clear that $\overline{3\text{SAT}} \in \text{coNP}$. In addition, observe that if $L \in \text{coNP}$, then $\overline{L} \in \text{NP}$ and thus $\overline{L} \leq_P 3\text{SAT}$ (since $3\text{SAT}$ is NP-complete). It follows that $L \leq_P \overline{3\text{SAT}}$, and therefore $\overline{3\text{SAT}}$ is coNP-complete. Now, if $\#\text{SAT} \in \text{NP}$, then $\overline{3\text{SAT}} \in \text{NP}$ via the reduction to $\#\text{SAT}$, namely $\varphi \mapsto (\varphi, 0)$. Moreover, since $3\text{SAT}$ is coNP-complete, we have coNP \subseteq NP. Complementation yields NP \subseteq coNP, and hence NP = coNP.

3. Prove that $E_{DFA}$ is NL-complete. (25 POINTS)

Since NL = coNL, it suffices to show that $E_{DFA}$ is coNL-complete, or equivalently, $\overline{E_{DFA}}$ is NL-complete.

To show that $\overline{E_{DFA}} \in \text{NL}$, we give a nondeterministic TM that decides $\overline{E_{DFA}}$ in logarithmic space that is similar to that one for PATH: on input a DFA $D$, the machine starts from $D$’s start state and nondeterministically guesses an alphabet at each step and follows a sequence of states in $D$ until it hits an accept state, in which case it accepts, or until the number of states that it has visited exceeds the number of states in $D$, in which case it rejects. The machine only needs to keep track of the current state and a counter on its work tape, which takes up only logarithmic space.

To show that $\overline{E_{DFA}}$ is NL-hard, we prove that PATH $\leq_L \overline{E_{DFA}}$. Given an instance $(G, s, t)$ of PATH, the log-space transducer for this reduction constructs a DFA $D$ as follows: the nodes of $G$ are the states of $D$, $s$ is the start state, $t$ is the unique accept state, and $\{1, 2, \ldots, d\}$ is the alphabet, where $d$ is the maximum out-degree of $G$ (a common mistake is to assume that the out-degree of each node in $G$ is 1 and use an alphabet of size 1). Finally, the transitions are given by the edges of $G$ (each edge leaving a given node is labeled by a different alphabet character; if a state has fewer than $d$ outgoing edges, on all remaining symbols, $D$ loops back to the same state). Clearly, $L(D)$ is non-empty if there is a path from $s$ to $t$ in $G$. Moreover, this reduction can be computed using logarithmic space.

4. Define $EQ_{NFA} = \{(N_1, N_2) \mid N_1, N_2$ are NFAs and $L(N_1) = L(N_2)\}$. Prove that $EQ_{NFA} \in \text{PSPACE}$. (25 POINTS)

First, we show that if $N_1, N_2$ are NFAs each with at most $n$ states where $L(N_1) \neq L(N_2)$, then there exists a string $s$ of length at most $2^{2n}$ in $L(N_1) \triangle L(N_2)$. This is because we can construct DFAs with at most $2^n$ states for the languages $L(N_1)$ and $L(N_2)$ and thus a DFA with at most $2^n$ states for the language $L(N_1) \triangle L(N_2)$. In particular, if the language $L(N_1) \triangle L(N_2)$ is non-empty, then it contains a string $s$ of length at most $2^{2n}$

Now, we give a non-deterministic linear space machine $M$ which decides $\overline{EQ_{NFA}}$. This implies $EQ_{NFA} \in \text{coNPSPACE} = \text{PSPACE}$. The machine incrementally guesses a string of length at most $2^{2n}$ that causes exactly one of $N_1$ and $N_2$ to accept. $M$ works as follows: on input $(N_1, N_2)$,

for $i \leftarrow 1$ to $2^{2n}$ do

- Nondeterministically guess an input symbol.
- Update the set of states each NFA could be in.
- Accept if one of $N_1, N_2$ accepts and the other does not accept.
- Reject.

Note that $M$ uses only linear space to store $i$, and the set of states each NFA $N_1, N_2$ could be in.