Problem 1: (Sipser 8.20) Let MULT = {a#b#c | a, b, c are binary numbers and a × b = c}. Show that MULT L.

Idea:

Individually calculate each bit of c using a counter to iterate through each bit in c, counters to iterate through bits in a and b, counters to track carryover from the previous bit and also carrover to the next bit, and of course a bit to track the current bit. If the current bit is different than what it is supposed to be in c then we reject.

M on input w:

If w not of form $a\#b\#c$ reject

Initialize ctr_c to 0 (first bit in c)

Initialize $ctr_{carry} = 0, ctr_{prev} = 0$ and $bit = 0$

Initialize ctr_b to 0 and ctr_a to 0

For each bit in c

Add ctr_{prev} to bit, if bit becomes 1 and ctr_{prev} is not zero then incriment ctr_{carry} by 1 and make $bit = 0$. (Add previous carryover to current bit and update the carryover)

Iterate i from 0 to ctr_b and j from ctr_a to 0 at the same time

If $i \geq |b|$ then $b_i = 0$ and if $j \geq |a|$ then $a_j = 0$

Otherwise add $a_j \times b_i$ to bit if $bit = 1$ then make bit 0 and incriment ctr_{carry} by 1.

Check if $bit = c$, if not reject else continue

M accepts (if all $a \times b$ bits are same as c bits)

The space complexity is $O(\log n)$ because all of these counters only count up to $\log n$ so it is a constant times $\log n$ space so $O(\log n)$.

Proof of correctness $L(M) = MULT$

$x \in MULT \rightarrow M$ accepts w

> If $x \in MULT$ then $a \times b = c$. Our machine checks that each bit of $a \times b$ is the same as the bits of c so then M accepts

$x \notin MULT \rightarrow M$ rejects w

> If $x \notin MULT$ then x not of form $a\#b\#c$ (so M rejects) or $a \times b \neq c$ in which case there is at least one bit of $a \times b$ that doesn't match with c so M rejects.

M accepts $w \rightarrow x \in MULT$

> If M accepts w then w is of the form $a\#b\#c$ and $a \times b = c$ so $x \in MULT$

M rejects $w \rightarrow x \notin MULT$

> If M rejects w then w is not of the form $a\#b\#c$ or one bit of $a \times b$ is not the same as c so $a \times b \neq c$ so $x \notin MULT$

Therefore since $L(M) = MULT$ and M uses log space then $MULT \in L$

Problem 2: (Second half of Sipser 8.25) An undirected graph is bipartite if its nodes may be divided into two sets so that all edges go from a node in one set to a node in the other set. This is equivalent to stating that the graph does not contain a cycle that has an odd number of vertices. Let BIPARTITE = { G | G is bipartite}. Prove that BIPARTITE NL.

Idea:

Similar to the UNDIRECTED CYCLE algorithm but this time we keep a parity checker and if at any point we return to a starting vertex with a different parity then we reject. This needs 2 counters (current start node and current start edge), constant space for the current vertex, constant space for the previous vertex, constant space for the parity bit.

This machine would be done deterministically since there are no choices or guesses to be done nondeterministically, resulting in $BIPARTITE \in L$ and $L \subset NL$ so $BIPARTITE \in NL$

M on input w

If w not a description of a graph G reject

Initialize a counter to iterate through every vertex

Initialize a parity bit to 0

For each vertex, go to the next vertex (iterate through all adjacent vertices) and change parity to 1

Then always take the first edge in sequence from our next vertex (take next edge after the edge we just took in the description of G, if the edge we just took was the last edge in our vertex then use the first edge for that vertex)

Continuously take the first edge in sequence and alternate the parity between 0 and 1 until we reach out starting vertex.

Once we reach the starting vertex, if the parity is not 0 reject, clear tape (except for the counters) repeat steps for next vertex adjacent to start.

If we have gone through all adjacent vertices then we move to the next starting vertex and repeat above.

After we have done the above for every possible starting vertex and adjacent vertex pair and none of them rejected then M accepts.

The space complexity is $O(\log n)$ because we have a counter to iterate through each starting vertex $O(\log n)$, a counter for starting edges $O(\log n)$ and constant space for current vertex and parity vertex. So total space is $O(\log n)$

Proof of correctness $L(M) = BIPARTITE$:

$x \in BIPARTITE \rightarrow M$ accepts on input w

This means that the graph can be split into 2 sets such that one set is parity 0 and parity 1, no matter what path there is there is no path that results in returning to a vertex at a different parity. Thus for each starting vertex we will return with a parity of 0 so our machine accepts.

$x \notin BIPARTITE \rightarrow M$ rejects on input w

This means there is at least one edge that connects vertices in the same parity set. This would result in these verticies having a different parity when our machine M returns to it. Thus M will reject.

M accepts on input $w \rightarrow x \in BIPARTITE$

This means all vertices returned to itslef at the same parity. This means all edges connected vertices of different parities. This means $x \in BIPARTITE$.

M rejects on input $w \to x \notin BIPARTITE$

This means one vertex was able to return to itslef at a different parity. This means somewhere an edge connected two vertices of the same parity. So $x \ni nBIPARTITE$

Therefore since we proved $L(M) = BIPARTITE$ and M uses log space then $BIPARTITE \in L \subset NL$ so $BIPARTITE \in NL$

Problem 3: Let ACYCLIC-PATH = { G, a, b | G is a directed graph with no cycles and there exists path from a to b in G}. Prove that ACYCLIC-PATH is NL-complete. (Hint: when doing the reduction, don't try to remove cycles from the initial graph, but rather expand it in a way that gets rid of cycles.)

$ACYCLIC - PATH \in NL$

Idea

$PATH \in NL$, prove $DIRECTED - CYCLE \in NL$ therefore $NO - DIRECTED - CYCLE \in coNL$ and $CO - NL = NL$

Define NTM M to decide $DIRECTED - CYCLE$ in log space

M on input w:

If w is not the description of a directed graph reject

For each vertex v in G:

If $PATH$ from v to v exists then accept

Else continue

reject

Space is $O(\log n)$ because we have one counter to iterate through vertices and $PATH \in NL$ so $O(\log n)$ space. Therefore space is $O(\log n)$

Since M is guarenteed to nondeterministically find a cycle if it exists, $DIRECTED - CYCLE \in NL$ therefore $NO - DIRECTED - CYCLE \in coNL$ and $CO - NL = NL$ so $NO - DIRECTED - CYCLE \in NL$.

Create M' a NTM to decide $ACYCLIC - PATH$ in log space

M' on input w:

> If w is not the description of a directed graph and a, b reject
>
> If $PATH(G, a, b)$ rejects then reject, else continue
>
> If $NO - DIRECTED - CYCLE(G)$ rejects then reject, else continue
>
> accept

Space is $O(\log n)$ because both machines $PATH, NO - DIRECTED - CYCLE$ are log space.

Therefore M' accepts iff there is a path from a to b and there are no cycles. So $L(M') = ACYCLIC - PATH$ and M' is a NTM in logspace so $ACYCLIC - PATH \in NL$

$PATH \leq_p ACYCLIC - PATH$

Idea:

Generate a graph that has different layers, with vertices $u_{v,i}$ where $v \in G$ and i is the layer or number of steps from a. i is bound from 0 to $n-1$ because 0 is only a and any path consisting of more than $n-1$ edges has a loop.

Create M on input w:

> If w is not of the form G, a, b reject
>
> Create logspace transducer f to generate a new graph for $ACYCLIC - PATH$.
>
> Create layer L_0 with vertex a_0 and add this vertex to simulated output
>
> Create layer L_1 with vertices $u_{v,1}$ for each $v \in G | \exists e \in G \wedge e = a \rightarrow v$ add these verticies to simulated output.
>
> For each layer i up to $n-1$:
>
> > create layer L_i with verticies $u_{v,i}$ for each $v \in G | \exists e \in G \wedge \exists u \in L_{i-1} \wedge e = u \rightarrow v$, add these verticies to simulated output.

This only ever results in a single vertex on the working tape at a time, and a counter to indicate the current layer

At each vertex insertion add a correct corresponding edge

$\forall j|$ add an edge $e = u_{b,j} \to b^*$.

Run $ACYCLIC - PATH(f(x))$ where $f(x)$ outputs the new graph with path inputs a_0, b^*

M accepts / rejects if $ACYCLIC - PATH(f(x))$ accepts / rejects

Proof of correctness

$x \in PATH \to f(x) \in ACYCLIC - PATH$

If $x \in PATH$ then there exists a path within n steps from a to b. Since $f(x)$ contains all possible paths of length n then $f(x)$ contains the path from a to b represented as the path from a_0 to $u_{b,j}$ for some j. $f(x)$ contains no cycles as we only move from one layer to the next. $u_{b,j}$ has an edge to b^* so then there is a path from a_0 to b^* so $ACYCLIC - PATH$ accepts on $f(x)$.

$x \notin PATH \to f(x) \notin ACYCLIC - PATH$

If $x \notin PATH$ then there does not exist a path within n steps from a to b. Since $f(x)$ contains all possible paths of length n then $f(x)$ contains no path from a_0 to $u_{b,j}$ for some j. $f(x)$ contains no cycles as we only move from one layer to the next. Since a does not reach b, there exists no $u_{b,j}$ so $ACYCLIC - PATH$ rejects on $f(x)$.

$f(x) \in ACYCLIC - PATH \to x \in PATH$

If $f(x) \in ACYCLIC - PATH$ then there is a path from a_0 to b^*, which means theres a path from a to b in G. Therefore $x \in PATH$.

$f(x) \notin ACYCLIC - PATH \to x \notin PATH$

If $f(x) \notin ACYCLIC - PATH$ then there is no path from a_0 to b^*, which means theres no path from a to b in G. Therefore $x \notin PATH$.

Therefore since $PATH \leq_p ACYCLIC - PATH \wedge ACYCLIC - PATH \in NL$ then $ACYCLIC - PATH$ is NL-complete.