Problem 1 Solution

To show A_{NFA} ∈NL, we should create a non-deterministic Turing machine to decide it in log-space:

Input: $(M = (Q, \delta, q_0, F, \Sigma), w)$
  Let $n = |w|, s = q_0$
  for $i = 1, ..., n$
    Guess new $s \in \delta(s, w_i)$ where w_i is the i^{th} character of w or ϵ
  if $s \in F$ accept, else reject

This just simulates M on w non-deterministically and so will accept if and only if $M(w)$ does. Furthermore, the only things it has to remember are n, s, and i, which each take $log(|input|)$ to store, so our machine is log-space.

To show A_{NFA} ∈NL-hard, we will reduce from a known NL-hard problem, ST-REACH. Specifically, we want a log-space transducer that takes input $G = (V, E), s, t$ and outputs M, w such that

$$(G, s, t) \in \text{ST-REACH} \Leftrightarrow (M, w) \in A_{NFA}$$

Although $(M = (Q, \delta, q_0, F, \Sigma), w)$ may end up being longer that $log(|(G, s, t)|)$, we can output $Q, \delta, q_0, F, \Sigma$, and w piece by piece only using log-space the entire time. We will do this as follows:

Let $Q = V,\ q_0 = s,\ F = \{t\},\ \Sigma = \emptyset,\ w = \epsilon$, and $\delta(u, \epsilon) = \{v | (u, v) \in E\}$. The state machine of this NFA is identical to the directed graph we were given and since we can traverse any edge on a whim (since arc is an ϵ-transition) then M accepts $w = \epsilon$ if and only if we can get from the start state (which is exactly s) to *the* final state (which is exactly encoded as t). So, this makes our reduction a valid one; we only have to argue that it is a log-space reduction now.

To see that we can have a log-space transducer to make this mapping note that we can output $Q, \delta, q_0, F, \Sigma$, and w sequentially without having to remember all of them. For example, we can output Q by going through V in the input piece-by-piece with a pointer (and a pointer only takes log space) and copying the single node we see there to the output tape to populate Q. Since we do this locally for a single node and then move on without having to remember what we've done, this is only takes log space to do. Similarly, we can construct the rest piece by piece in the manner only focusing on the current node or edge we're currently on and so we only ever use log space at any given time.

Therefore, ST-REACH$\leq_m^L$ A_{NFA} and A_{NFA} ∈NL so A_{NFA} is NL-complete.

Problem 2 Solution

To show DAG-REACH∈NL, we should create a non-deterministic Turing machine to decide it in log-space. We will assume every graph we're giving is a DAG and that we don't have to check for this as mentioned on Piazza (although I give the blueprint on Piazza of how we could solve this without that assumption).

Input: $(G = (V, E), s, t)$
  Let $n = |V|, p = s$
  for $i = 1, ..., n$
    if $p == t$ accept
    Guess p =neighbor of p
  reject

This will find an $s-t$ path if there is one and the only things it has to remember are the pointers/counters n, p, and i, so our machine is log-space.

And to show DAG-REACH∈NL-hard we will show ST-REACH$\leq_m^L$DAG-REACH. That is, we want a log-space transducer that takes input $G = (V, E), s, t$ and outputs $G' = (V', E'), s', t'$ such that

$$(G, s, t) \in \text{ST-REACH} \Leftrightarrow (G', s', t') \in \text{DAG-REACH}$$

To do this let's take (G, s, t) and create a layered/hierarchical graph G' so that every node can only have edges going to the layer directly above. Note that any graph of this form must be acyclic since there are no edges with both endpoints in a layer and there is only forward edges (no backwards traveling). So now we just need to make G' clever enough so that our iff statement holds.

Let V' be $|V|$ copies of V, where each copy of V constitutes a layer. Then draw an edge from node i at each level to node j in the next level if G contained an edge from i to j in the original graph. Finally draw an edge from node i in each level to node i in the next level. And if we let s' be s in the first layer and t' be t in the last layer, then our claim that there is a path from s to t in G iff there is a path from s' to t' in G' holds in the straightforward way of taking an $s - t$ path in G and translating it to the corresponding layered $s' - t'$ path in G' and vice versa.

Finally, this can be done in log space by an almost identical argument to problem 1. For example, to show that creating V' for the output tape can be done in log-space, note that the following algorithm writes all of V' to the output tape and only uses a constant number of pointers/counters and so only needs log-space memory:

```
Let n = |V|, v = v_0
for i = 1, ..., n
  for j = 1, ..., n
    write v on output tape as v_i to signify it as node v on layer i
    let v =next node on input tape
  let v = v_0
```

E' can be created similarly and outputting s' and t' are trivial.

So DAG-REACH is in NL and NL-hard and so is NL-complete.

Problem 3 Solution

To show StrongConn$\in$NL, we should create a non-deterministic Turing machine to decide it in log-space.

```
Input: G = (V, E)
  Let n = |V|
  for i = 1, ..., n
    for j = 1, ..., n
      let s = v_i and t = v_j
      if s == t continue
      for k = 1, ..., n
        guess s =neighbor of s
        if s == t break
      if s! = t reject
  accept
```

As before this only uses a constant number of pointers/counters and so is log space.

Now we will reduce DAG-REACH to StrongConn by showing a log-space transducer is possible that takes input $G = (V, E), s, t$ and outputs $G' = (V', E')$ such that

$$(G, s, t) \in \text{DAG-REACH} \Leftrightarrow G' \in \text{StrongConn}$$

To do this let's take $(G = (V, E), s, t)$ let $V' = V$ and then supplement E so that for every $v \in V - \{s, t\}$ we add the directed edges (v, s) and (t, v).

This is in log-space, similar to before, since we just visit each vertex and add these edges accordingly without having to keep the vertices in memory. Now we want to argue our iff claim:

If $(G, s, t) \in$DAG-REACH, then there there is a path from s to t. Then for any two vertices $u, v \in V$, we have our edge (u, s) that can be concatenated with the path that exists from s to t and then finally we

finish our path with the edge (t, v). Since u and v were arbitrary, there is a path from each vertex to every other vertex and so $G' \in$StrongConn.

Alternatively, if $G' \in$StrongConn, then there is certainly a path from s to t by the definition of strong connectivity, but we must also argue that that path can be made with only edges from our original G: If an edge (v, s) is in the path, then there must have been a preceding subpath from s to v; but that means there is a cycle through v which we can simply remove from the path. And if an edge (t, v) is in the path, then there must be a preceding sub-path from s to t; this means (t, v) and everything after it are superfluous and can just be removed. So we are left with an $s - t$ path with edges only from our original G and so $(G, s, t) \in$DAG-REACH.

And so StrongConn is both NL and NL-hard and therefore NL-complete.