Reminder

- We defined the classes of languages that can be solved by Turing Machines in deterministic and non-deterministic logarithmic space to be \( L \) and \( NL \). Formally these Turing Machines have a read-only input tape and a separate tape to perform computation.

- \( NL \subseteq P \) since all \( O(\log n) \) space-configurations can be computed in \( 2^{O(\log n)} = n^{O(1)} \) time, \( NL \subseteq SPACE(\log^2 n) \) by Savitch’s Theorem.

- A log-space transducer is a Turing Machine with an input and output tape along with a work tape limited to logarithmic space. We say \( A \leq_L B \) i.e. \( A \) is log-space reducible to \( B \) if and only if a log-space transducer reduces \( A \) to \( B \).

- \( A \) if \( NL \)-complete if and only if \( A \in NL \) and every \( NL \) language is log-space reducible to \( A \).

- \( PATH \in NL \)-complete by modeling the tape configurations of an \( NL \) language with a graph computable in log-space.

- \( NL = coNL \) by showing \( PATH \in NL \) and using the fact that every \( coNL \) language is log-space reducible to \( PATH \).

- To show \( PATH \in NL \), we note that if we knew the number of nodes \( c \) reachable from \( s \) we can test for no path between \( s \) and \( t \) with the following algorithm:

  "On input \( \langle G, s, t \rangle \) :
  1. Initialize \( d = 0 \)
  2. For \( v \in V \)
     a. Run a non-deterministic branch of \( PATH(G, s, v) \) and increment \( d \) by 1 if it accepts.
     b. If it did accept and \( v = t \), reject.
  3. If \( d = c \), accept."

The above algorithm works since can determine whether it only accepts when it does not find a path from \( s \) to \( t \) AND has accounted for all the reachable vertices from \( s \).

- To calculate \( c \) in log-space we iteratively calculate \( R_i \) the set of vertices reachable from \( s \) within \( i \) steps and let \( c = |R_{|V|}| \)
Example 1: Strongly-Connected is NL-complete

We say that a directed graph is strongly connected if every two nodes are connected by a directed path in each direction.

Let \( SC = \{ (G) \mid G \text{ is a directed strongly connected graph} \} \)

Show that \( SC \) is NL-complete.

We first show \( SC \in NL \) with the following algorithm.

“On input \( \langle G \rangle \):

1. For every two nodes \( u \) and \( v \):
   (a) Run \( \text{PATH}(G, u, v) \)
   (b) If reject, reject.

2. Else at the end of the loop, accept.”

Now we show \( \text{PATH} \leq_L SC \).

Given \( \langle G, s, t \rangle \) we construct \( \langle G' \rangle \) by adding an edge from \( v \) to \( s \) and from \( t \) to \( v \) for all \( v \in V \).

If \( \langle G, s, t \rangle \in \text{PATH} \) then for arbitrary vertices \( u, v \) in \( G' \) we can form the path \( u \to s \leadsto t \to v \) therefore \( \langle G' \rangle \in SC \).

If \( \langle G' \rangle \in SC \) then there must exist a simple path (does not repeat vertices) from \( s \) to \( t \) in \( G' \). Since the edges we added in the reduction either take us out of \( t \) or into \( s \) they cannot exist in this simple path. Therefore a path between \( s \) and \( t \) in \( G' \) is also a path in \( G \) so \( \langle G, s, t \rangle \in \text{PATH} \).
Example 2: Bipartiteness is in NL

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.

Let \[ \text{BIPARTITE} = \{ \langle G \rangle \mid G \text{ is an undirected bipartite graph} \} \]

Show that BIPARTITE ∈ NL.

Convince yourself that the following lemma is true.

**Lemma:** \( G \) is bipartite iff \( G \) does not contain an odd-length cycle.

Since NL = coNL thus allows us to provide a non-deterministic algorithm that accepts if \( G \) has an odd-length cycle to show BIPARTITE ∈ coNL hence the result. We can detect odd-length cycles with the following algorithm

"On input \( \langle G \rangle \):

1. Initialize \( d = 0 \)

2. Non-deterministically select \( v \in V \) and let \( \text{start} = v \)

3. While \( d \leq V \)
   a. Increment \( d \) by 1
   b. Reassign \( v \) non-deterministically to a neighbor of \( v \) i.e. set \( v = u \) where \((u, v) \in E\)
   c. If \( u = \text{start} \) and \( d \) is odd, accept.

2. Else at end of loop, reject."