1 Background

Recall that an NL-certifier is a Turing Machine with three tapes:

- Input Tape: Read-Only.
- Working Memory: Read-Write but is small $O(\log(n))$, where $n$ is the input size.
- Certificate/Proof Tape: Read-Once: the pointer only moves to the right (every memory cell gets read at most once).

A language $L$ is in NL if and only if there is an NL-certifier $M$ such that

$$x \in L \iff \exists y, M(x, y) = \text{accept}.\]$$

In the previous lecture we showed that $PA_0H = \{(G, s, t) : \exists \text{ an } st\text{-path in the directed graph } G \} \in NL$

Indeed it is NL-complete. To see why $PATH \in NL$, note that if we are provided with an $st$-path, $u_1, \ldots, u_k$ such that $u_1 = s$ and $u_k = t$, we can verify the validity of this certificate using only $O(\log(n))$ memory: As we read the path, we check that it starts at $s$ and ends at $t$, and for every $i$, we check with the input that there is an edge from $u_i$ to $u_{i+1}$. At any moment our working memory essentially only contains two values $u_i, u_{i+1}$ (each requiring only $O(\log(n))$ space) plus maybe some extra bits for the computation.

2 Immerman-Szelepcsényi’s Theorem

In this section we prove that $NL = CoNL$. The key element of the prove is showing that the complement of $PATH$ is in $NL$. That is

$\overline{PATH} = \{(G, s, t) : \exists \text{ an } st\text{-path in the directed graph } G \} \in NL$

Note that this is not trivial! How can we come up with an easily verifiable proof for the fact that there are no paths from $s$ to $t$?

**Theorem 1.** We have $NL = CoNL$.

**Proof.** In the next lemma we will prove that $PATH \in NL$. Then since $PATH$ is NL-complete, $PATH$ is CoNL-complete. Hence $PATH \in NL$ implies that $CoNL \subseteq NL$. On the other hand $PATH \in NL$ also implies (by the definition of CoNL) that $PATH \in CoNL$, which then by completeness of $PATH$ implies $NL \subseteq CoNL$. 

Lemma 1. $\text{PATH} \in \text{NL}$.

Proof. Let $m = |V(G)|$, and let us label the vertices as $u_1, \ldots, u_m$. The input size is roughly $n = \Theta(m^2)$ if we present the graph with its adjacency matrix, and $\log(n) = \Theta(\log(m))$. Note that each vertex can be represented with $\log(m)$ bits.

For $k = 0, \ldots, m$, define

$$A_k = \{ u : \exists s \text{ path of length } \leq k \}.$$ 

Now let us try to build the certificate. For each $k = 0, \ldots, n$, define

$$P_k = [B^k_1, \ldots, B^k_n],$$

where the blocks are defined as

- If $u_i \in A_k$, then
  $$B^k_i = [u_i \in A_k, \text{ One } su_i \text{ path of length at most } k].$$

Note that here the path is a proof for $u_i \in A_k$.

- If $u_i \notin A_k$, then
  $$B^k_i = [u_i \notin A_k] \text{ (with no proof).}$$

We make two key observations

1. If we know the size $c_k = |A_k|$, then we can verify the correctness of $P_k$. Indeed if the vertices in $A_k$ are given to us correctly (which we can verify), and the total count matches $c_k$, then we shall know that the non-memberships $[u_i \notin A_k]$ are also given to us correctly.

2. If we have a fixed vertex $v$ in mind, then as we are verifying $P_k$ (assuming we know $c_k$), we can find out whether $v \in A_{k+1}$ or not. Each time we see a vertex $u_i \in A_k$, we check to see if it has an edge to $v$. If it has, then $v \in A_{k+1}$.

Algorithm 1 Verifying correctness of $P_k$, and $v \in A_{k+1}$ given $c_k$

1: procedure VERIFY($c_k, k, P_k, v$) \hfill \Comment{Only remembers $c, k, c_k, i, v, B$}
2: Initialize $c \leftarrow 0$ and $B \leftarrow [v \notin A_{k+1}]$
3: for $i = 1, \ldots, k$ do
4:   if $u_i \in A_k$ according to $P_k$ then
5:     Verify it by following the path \hfill \Comment{requires only $O(\log(m))$ bits}
6:     Increase $c$
7:   if $u_i \backslash v$ is an edge then
8:     $B \leftarrow [v \in A_{k+1}]$
9: Verify that $c = c_k$ and return $B$

Now it remains to compute $c_k$ from the certificate. The full certificate that is provided to us is going to be the following:

$$C = P_0 \ldots P_0 P_1 \ldots P_1 P_2 \ldots P_2 \ldots P_m \ldots P_m$$

$m$ times $m$ times $m$ times $m$ times

Computing $c_k$ is going to be straightforward using the above observations. Indeed as we read the $m$ copies of $P_k$, we can compute $c_{k+1}$. We will the $i$-th copy to find out whether $u_i \in A_{k+1}$. Note that $c_0 = 1$ and $A_0 = \{s\}$. Note also that there is no $st$-path if and only if $t \notin A_m$. The verification of $Y$ is illustrated in Algorithm 2. 

$\square$
Algorithm 2 Verifying the Full Certificate $Y$

1: procedure $\text{Verify}(Y)$
2: Set $c_0 \leftarrow 1$.
3: for $k = 0, \ldots, m$ do
4:   Initialize $c_{k+1} \leftarrow 0$
5:   for $j = 1, \ldots, m$ do
6:     $\text{VerifyP}(c_k, k, P_k, u_j)$
7:     if it returns $u_j \in A_{k+1}$ then
8:       Increase $c_{k+1}$
9: Return whether $t \notin A_m$

$\triangleright$ Only remembers $k, c_k, c_{k+1}, j$
$\triangleright$ At this point we know $c_k$, we forget $c_{k-1}$