Chapter 8, Part 2

NL and L
NL-Completeness

A logspace transducer is a TM with a read-only input tape, a write-only output tape, and a read/write work tape, in which only $O(\log n)$ tape cells of the work tape can be used.

A logspace transducer $M$ computes a function $f$ if for every $w$, $M$ on $w$ halts with $f(w)$ on the output tape.

A language $A$ is logspace reducible, write $A \leq_L B$, if there is a logspace computable mapping reduction from $A$ to $B$.

A language $L$ is NL-complete if $A \in \text{NL}$ and every $A \in \text{NL}$ is logspace reducible to $L$. 
Properties of logspace reductions

**Theorem.** If $A \leq_L B$ and $B \in L$ then $A \in L$.

If $A \leq_L B$ and $B \in \text{NL}$ then $A \in \text{NL}$.

**Theorem.** If $A$ is \text{NL}-complete and $A \in L$ then $L = \text{NL}$.
Theorem. \textit{PATH} is \textbf{NL}-complete.

Proof. \textit{PATH} $\in$ \textbf{NL}. Given an instance $(G, s, t)$ of \textit{PATH} with $n$ nodes, repeat the following $n - 1$ times with $x = s$ at the beginning:

- Nondeterministically select a node $y$ from $1, \ldots, n$,
- If $(x, y)$ is in $G$, then set $x$ to $y$. If not, reject.
- If $y = t$, then accept.

This method correctly decides whether $(G, s, t) \in \textit{PATH}$ and requires $O(\log n)$ space.
Let $L$ be decided by a nondeterministic $c \log n$ space machine $N$. We may assume that $N$ has the unique accepting configuration for each input. Let $x$ be an input of some length $n$. Define the graph $G$ as follows:

- The nodes of $G$ are the configurations of $M$ on $x$. Here each configuration is the concatenation of the state, head positions, and the work tape contents.
- $s$ is the initial configuration
- $t$ is the accepting configuration.
- For every pair of nodes $u$ and $v$, there is an arc from $u$ to $v$ if and only if $v$ is one of the next possible configurations of $u$.

Then $(G, s, t) \in PATH$ if and only if $x \in L$. 
Computation of \((G, s, t)\) in logspace

Let \(\ell\) be the encoding length of each configuration.

```plaintext
for \(u = 0^\ell, \ldots, 1^\ell\) do
  for \(v = 0^\ell, \ldots, 1^\ell\) do
    if \(u\) and \(v\) are configurations then
      if \(u \Rightarrow v\) then output 1 else output 0
  \(C \leftarrow 0;\)
  for \(u = 0^\ell, \ldots, 1^\ell\) do
    if \(u\) is a configuration then
      \(C \leftarrow C + 1;\)
      if \(u\) = the initial config. then output \(“s = C”\)
      if \(u\) = the accepting config. then output \(“t = C”\)
```

The algorithm works in \(O(\ell) = O(\log n)\) space.
**Theorem.** \( \overline{PATH} \in \text{NL} \).

**Proof** Let \((G, s, t)\) be an instance of \(PATH\) with \(n\) nodes. For each \(i, 0 \leq i \leq n\), define \(A_i\) to be the set of all nodes reachable from \(s\) within \(i\) steps and \(c_i = \|A_i\|\).

Given \(c_i\) it is possible to nondeterministically enumerate all the nodes in \(A_i\) with the following \text{ENUMERATE}(i, c_i):

1. Set counter \(d\) to 0;
2. for \(j = 0, \ldots, n\) do the following:
   (a) guess an \(s\)-to-\(j\) path of length at most \(i\);
   (b) if successful increment \(d\) and output \(j\);
3. if \(d = c_i\) output “SUCCESSFUL”; otherwise, output “FAILURE”
Computing $c_{i+1}$ knowing $c_i$

1. Set counter $e$ to 0;

2. For $j = 0, \ldots, n$ do the following:
   (a) Set a variable $r$ to false.
   (b) Call ENUMERATE($i, c_i$). For each node $u$ output by ENUMERATE, check if $u \Rightarrow j$; if so, set $r$ to true.
   (c) If ENUMERATE has output “FAILURE” at the end output “FAILURE”.

Otherwise, increment $e$ if and only if $r = true$.

3. Output $e$. 
Testing Unreachability

1. Set $i$ to 0 and $c_0$ to 1.
2. For $i = 0, \ldots, n - 1$, compute $c_{i+1}$ from $c_i$.
3. (Check if $t \not\in A_n$ by calling ENUMERATE($n$, $c_n$).) Accept if the enumeration is “SUCCESSFUL” and $t$ is not output.

The method uses only $O(\log n)$ space.