COMP4141 Theory of Computation

Log Space

Ron van der Meyden

CSE, UNSW

Revision: 2015/05/07

(Credits: M Sipser, K Engelhardt, R van Glabbeek, P Hofner)

Sublinear Space

Consider 2-tape TMs where the tape containing the input tape is read-only.

Change the definition of space complexity to ignore the input tape. (*Tute:* This matters at most for sublinear space.)

Definition

$$\mathbf{L} = \mathbf{SPACE}(\log n) \qquad \mathbf{NL} = \mathbf{NSPACE}(\log n)$$

Example

$\{\ 0^k 1^k \mid k \in \mathbb{N}\ \} \in \mathbf{L}$: Use a single binary counter initialised to 0 to first count the 0s up and then the 1s down.

Example

Recall that

$$PATH = \{ \langle G, s, t \rangle \mid t \text{ is reachable from } s \text{ in } \textbf{directed} \text{ graph } G \}$$

is in **P**. For $PATH \in \mathbf{NL}$ we build:
$M =$ "On input $\langle (V, E), s, t \rangle$

1. store $v \leftarrow s$ on the 2nd tape
2. repeat up to $|V| - 1$ times:
3. non-deterministically guess v' with $(v, v') \in E$
4. if $v' = t$, *accept* else store $v \leftarrow v'$
5. *reject*."

Whether $\mathbf{L} = \mathbf{NL}$ is open. Whether $PATH \in \mathbf{L}$ is unclear, but the **undirected** version is known to be in **L** (Reingold 2005).

Log Space Reducibility

Polynomial-time reducibility ($\leq_{\mathbf{P}}$) is too coarse a measure to define **NL**-completeness. Instead, we'll use *log space reducibility* ($\leq_{\mathbf{L}}$) based on log space transducers:

Definition

A *log space transducer* is a 3-tape TM with

1. a read-only *input* tape,
2. a read-write *working* tape, and
3. a write-only *output* tape

that uses only $\mathcal{O}(\log n)$ space on the working tape.

Theorem

If $A \leq_{\mathbf{L}} B$ *and* $B \in \mathbf{L}$*, then* $A \in \mathbf{L}$.

NL-completeness

Definition

A language B is **NL**-*complete* if $B \in \mathbf{NL}$ and for every $A \in \mathbf{NL}$ we have $A \leq_{\mathbf{L}} B$.

Theorem

PATH is **NL**-*complete.*

Corollary

$\mathbf{NL} \subseteq \mathbf{P}$

Certificate Definition of **NL**

Recall that **NP** is the class of languages for which **P** verifiers exist. A similar characterisation can be given for **NL**.

Theorem

*$L \in$ **NL** if L has a logspace verifier, that is, a 3-tape TM M and a polynomial p such that for all x there exists a certificate u of size $p(|x|)$ and M accepts iff $x \in L$ when started as follows:*

1. *tape 1 is read-once (from left to right) and contains the certificate u,*
2. *tape 2 is read-only and contains the input x, and*
3. *tape 3 is read-write work tape of size $\mathcal{O}(\log|x|)$.*

Proof.

As for the two characterisations of **NP**, we show one direction by using a description of an accepting run of the NTM as certificate and the other direction by guessing the certificate symbol-by-symbol when we need it. □

None of this is in [Sipser2006].

Example

To show once again that $PATH \in \mathbf{NL}$ we let the certificate for $\langle((V, E), s, t)\rangle \in PATH$ be a list $[v_0, \ldots, v_k]$ of nodes forming an acyclic path from s to t in (V, E). The verifier checks that

1. $v_0 = s$,
2. $(v_j, v_{j+1}) \in E$, for all $0 \leq j < k$, and
3. $v_k = t$.

This takes at most logspace because (a) it suffices to store 2 nodes and (b) node names are binary representations of $1, \ldots, |V|$.

Theorem (Immerman-Szelepcsényi)

$\mathbf{NL} = \mathbf{coNL}$

Proof.

Show $\overline{PATH} \in \mathbf{NL}$ by providing a logspace verifier and certificates. On input $\langle (V, E), s, t, u \rangle$ our verifier uses two procedures to certify that:

1. $v \notin C_i$ given $|C_i|$
2. $|C_i| = c$, given $|C_{i-1}|$

where $C_0 = \{s\}$ and $C_{i+1} = C_i \cup E(C_i)$, i.e., C_i is the set of nodes reachable from s in at most i steps.

Applying the second procedure $|V| - 1$ times yields the number of nodes reachable from s and the first procedure can then certify $t \notin C_{|V|-1}$. $\square$

Proof details I

What can we do with a logspace verifier?

Given $v \in V$ and $i \leq |V|$ we can certify $v \in C_i$.

Same as for *PATH* plus counting steps.

Given $v \in V$, $i \leq |V|$, and $|C_i|$ we can certify $v \notin C_i$.

cert = ordered list of $|C_i|$ certificates for the $u \in C_i$.
Check that:

1. each sub-certificate is valid as per the previous point,
2. certificates are ordered,
3. none of the certified nodes is v, and
4. the number of certificates is $|C_i|$

Proof details II

Given $v \in V$, $i \leq |V|$, and $|C_{i-1}|$ we can certify $v \notin C_i$.

Ordered list of $|C_{i-1}|$ certificates for the $u \in C_{i-1}$.
Check that:

1. each sub-certificate is valid as per the previous point,
2. certificates are ordered,
3. the certified nodes are neither v nor neighbours of v, and
4. the number of certificates is $|C_{i-1}|$.

Proof details III

Given $|C_{i-1}|$ and c we can certify $|C_i| = c$.

cert = ordered list of $|V|$ certificates for the $v \in V$ of either $v \in C_i$ or $v \notin C_i$ as described above.
Check that:

1. each sub-certificate is valid as per the previous points while counting the total number of certificates and the number of $v \in C_i$ certificates
2. Accept if those counts are $|V|$ and c, respectively.