Logarithmic space

Evgenij Thorstensen

V18

Journey below

Unlike for time, it makes sense to talk about sublinear space.

This models computations on input that does not fit in memory.

To make sense of it, we need to change our computational model.

Read-only tape

Our NTMs and DTMs will have two tapes. One is a read-only input tape, and the other is a regular working tape.

The machine starts with the working tape blank, and the input on the input tape.

The space used by such a TM is the number of tape cells on the working tape. We ignore the input tape.

The classes L and NL

Define

$$L = \mathrm{SPACE}(\log n)$$

and

$$NL = \mathrm{NSPACE}(\log n)$$

Why not $(\log n)^2$ or $\bigcup_{k \in \mathbb{N}} (\log n)^k$?

See also
https://cstheory.stackexchange.com/questions/3439/why-do-we-consider-log-space-as-a-model-of-efficient-

Why these classes?

We can define

$$\mathsf{polyL} = \bigcup_{k \in \mathbb{N}} \mathsf{SPACE}((\log n)^k)$$

However, little is known about this class.

Not clear whether $\mathsf{SPACE}((\log n)^2) \subseteq \mathsf{P}$.

Why these classes?

We can define

$$\mathsf{polyL} = \bigcup_{k \in \mathbb{N}} \mathsf{SPACE}((\log n)^k)$$

However, little is known about this class.

Not clear whether $\mathsf{SPACE}((\log n)^2) \subseteq \mathsf{P}$.

We have

$$\mathsf{SPACE}(\log n) \subseteq \bigcup_{c \in \mathbb{N}} \mathsf{TIME}(2^{c(\log n)})$$

due to number of configurations of a space-bounded DTM.

Since $2^{(\log n)} = n$, $2^{c(\log n)} = n^c$, and so $\mathsf{L} \subseteq \mathsf{P}$. However, $2^{(\log n)^2} = n^{\log n}$.

Some properties

L is the smallest class containing log space and being closed under subroutines.

Obvious that NL $\subseteq$ NP, by configurations argument.

Not obvious that NL $\subseteq$ P — Savitch gives a quadratic space increase!

We will prove that NL $\subseteq$ P later.

Hands-on computation

The language $\{0^k 1^k \mid k \geqslant 0\}$ is in L.

To decide it, count the number of zeroes and ones, check if they are equal.

To do that, need to store two numbers. Each number is $\leqslant n$.

Size of a number k in binary is $\log k$.

More generally, what can I compute?

$\log n$ space allows me to have $c \log n$ bits.

$\log n$ bits can store numbers up to n.

I am thus allowed a constant number of pointers into the input, and $\log n$ boolean flags.

A pointer is the cell number on the input tape.

An interesting problem for L

Relational joins over a fixed query.

Q = SELECT * FROM R JOIN S ON (R.a = S.b) JOIN T ON (...)

Input: Tables R, S, T... mentioned.

Accept if the join is nonempty.

Join nonempty if and only if the tables have a tuple each that satisfies the equalities.

Need to store a pointer for each table.

A problem in NL

Recall PATH, "is there a path from s to t in this directed graph?"

This language is in NL, but we need to be clever.

Length of path can be n, so can't store it.

However, at each step I can record the current node, reusing this space.

Nondeterministically select next node from those current one points to.

Accept if t is found, reject if not after n steps.

Certificate definition of NL

NP has a verifier definition, using polynomial-time DTM.

Can we do the same for NL with a logspace-bounded verifier?

Yes, but not in the naive way.

Can't just have a DTM $M(w, c)$ using log space. Can solve 3SAT with that.

Can't restrict to a log-size certificate, either.

Read-once verifier

A read-once verifier is a DTM $M(w, c)$ with a read-once tape, head can only move right on it.

$w \in A$ if exists some c for which $M(w, c)$ accepts.

For NL, we have three tapes. A read-once certificate tape, a read-only input tape, and a logspace-bounded working tape.

NL and verifiers

NL is the class of languages with logspace-bounded read-once verifiers.

If I have a logspace NTM decider, I can use a computation branch as a certificate.

Since the NTM uses log space, I can redo the computation one step at a time, not exceeding log space.

If I have a verifier, I can build an NTM to guess the next symbol of the certificate I need. Store at most log such, so this is fine.

Some inclusions

$$\mathsf{L} \subseteq \mathsf{NL} = \mathsf{coNL} \subseteq \mathsf{P}$$

We will prove both of these. $\mathsf{NL} \subseteq \mathsf{P}$ will follows from a completeness theorem for NL.

By analogy with P and NP, we will define NL-complete problems using *logspace reductions*.

A logspace reduction is a function f from strings to strings computable in log space.

Formally, a log space transducer, with a read-only input tape, a write-only output tape, and a log-bounded working tape.

NL-completeness

Definition

A language W is NL-complete iff $W \in \mathsf{NL}$ and all problems in NL are logspace-reducible to W.

We write $W_1 \leqslant_{\mathsf{L}} W_2$ when W_1 is logspace-reducible to W_2.

For P, we had closure under $\leqslant_{\mathsf{P}}$. The same is true of L, but less obvious.

L is closed under $\leqslant_L$

Theorem

If $B \in L$ *and* $A \leqslant_L B$, *then* $A \in L$.

Idea: Same as for P, given w for A, compute $f(w)$ and run the machine for B on that.

Problem: Space does not bound time and output size! Could have $|f(w)| \neq O(\log |w|)$.

Need to avoid having to store all of $f(w)$.

L closed under $\leqslant_L$, proof

If $B \in L$ and $A \leqslant_L B$, then $A \in L$.

We can trade time for space. The DTM deciding A will not compute and store $f(w)$.

Instead, we will compute symbols of $f(w)$ on demand.

Every transition of $M(B)$, we recompute $f(w)$, and store only the symbol we need.

Store only current symbol and cell number of that symbol.

Closure, consequences

L being closed under $\leqslant_{\mathrm{L}}$ implies that if an NL-complete problem is in L, then $\mathrm{L} = \mathrm{NL}$.

The proof also implies that a logspace DTM can use other logspace DTMs as subroutines.

This is despite the fact that a product of logs is not itself a log!

This property is more generally called "a class being low for itself".