Introduction to Theoretical Computer Science

Lecture 16: The Rest of Space Complexity

Dr. Liam O'Connor

LFCS, University of Edinburgh
CECS, Australian National University
Semester 1, 2023/2024

Logarithmic Space

Definition

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

where $\mathbf{SPACE}(f(n))$ (resp. $\mathbf{NSPACE}(f(n))$) are the classes of problems decidable in $f(n)$-bounded space by a deterministic (resp. non-deterministic) Turing machine.

Logarithmic Space

Definition

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

where $\mathbf{SPACE}(f(n))$ (resp. $\mathbf{NSPACE}(f(n))$) are the classes of problems decidable in $f(n)$-bounded space by a deterministic (resp. non-deterministic) Turing machine.

How can a Turing machine have a sublinear space bound?

Logarithmic Space

Definition

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

where $\mathbf{SPACE}(f(n))$ (resp. $\mathbf{NSPACE}(f(n))$) are the classes of problems decidable in $f(n)$-bounded space by a deterministic (resp. non-deterministic) Turing machine.

How can a Turing machine have a sublinear space bound?

Revised Bounded Turing Machine

Define a $f(n)$-space-bounded Turing machine with two tapes:

1. the *input tape* is read-only, and just contains the input of size n.
2. the *working tape*, which is read-write and bounded by $f(n)$.

Problems in **L**

Example

$\{0^k 1^k \mid k \in \mathbb{N}\} \in \mathbf{L}$ Why?

Problems in **L**

Example

$\{0^k 1^k \mid k \in \mathbb{N}\} \in \mathbf{L}$ Why?

Example

$PATH = \{\langle G, s, t\rangle \mid t$ reachable from s in **directed** graph $G\} \in \mathbf{P}$

Is it in **L**?

Problems in **L**

Example

$\{0^k 1^k \mid k \in \mathbb{N}\} \in \mathbf{L}$ Why?

Example

$PATH = \{\langle G, s, t\rangle \mid t$ reachable from s in **directed** graph $G\} \in \mathbf{P}$

Is it in **L**?

- We don't know.

Problems in **L**

Example

$\{0^k 1^k \mid k \in \mathbb{N}\} \in \mathbf{L}$ Why?

Example

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

Is it in **L**?

- We don't know.
- **Undirected** version is in **L** (Reingold 2005), but the proof is not easy (because **SL** = **L**).
- What about **NL**?

Problems in **NL**

$PATH \in$ **NL**

On input $\langle (V, E), s, t \rangle$:

1. store $v \leftarrow s$ on the working tape
2. repeat up to $|V| - 1$ times:
3. nondeterministically 'guess' v' where $(v, v') \in E$
4. if $v' = t$ accept, else set $v \leftarrow v'$
5. reject

Problems in **NL**

$PATH \in$ **NL**

On input $\langle (V, E), s, t \rangle$:

1. store $v \leftarrow s$ on the working tape
2. repeat up to $|V| - 1$ times:
3. nondeterministically 'guess' v' where $(v, v') \in E$
4. if $v' = t$ accept, else set $v \leftarrow v'$
5. reject

Why is this in **NL**?

Problems in **NL**

$PATH \in$ NL

On input $\langle (V, E), s, t \rangle$:

1. store $v \leftarrow s$ on the working tape
2. repeat up to $|V| - 1$ times:
3. nondeterministically 'guess' v' where $(v, v') \in E$
4. if $v' = t$ accept, else set $v \leftarrow v'$
5. reject

Why is this in **NL**?

Question

L $\subseteq$ **NL**, but is **NL** $\subseteq$ **L**? We don't know.

Log-space transducers

Definition

A *log-space transducer* is a Turing machine with three tapes:

1. The input tape, which is read-only.
2. The working tape, which is read-write and log-bounded.
3. The output tape, which is write-only.

A *log-space reduction* is a reduction computable by a log-space transducer.

Hardness

Definition

A problem P_1 is *log-space reducible* to P_2, written $P_1 \leq_L P_2$, if there is a log-space reduction from P_1 to P_2.

Hardness

Definition

A problem P_1 is *log-space reducible* to P_2, written $P_1 \leq_L P_2$, if there is a log-space reduction from P_1 to P_2.

Recall:
To prove that a problem P_2 is *hard*, show that there is an *easy* reduction from a known hard problem P_1 to P_2.

Hardness

Definition

A problem P_1 is *log-space reducible* to P_2, written $P_1 \leq_L P_2$, if there is a log-space reduction from P_1 to P_2.

Recall:
To prove that a problem P_2 is *hard*, show that there is an *easy* reduction from a known hard problem P_1 to P_2.

Definition

A problem P is ***NL**-Hard* if, for every $A \in \mathbf{NL}$, $A \leq_L P$

- If a problem P_1 is **NL**-hard and $P_1 \leq_P P_2$ then P_2 is **NL**-Hard.
- To prove that a problem P_2 is **NL**-hard, show that there's a log-space reduction from a known **NL**-hard P_1 to P_2.

Completeness

Question

If any **NL**-hard problem is shown to be **L**, what does that mean?

Completeness

Question

If any **NL**-hard problem is shown to be **L**, what does that mean?

Definition

A problem is ***NL-complete*** if it is both **NL**-hard and in **NL**.

Example

PATH is **NL**-complete.

- We already know *PATH* $\in$ **NL**.
- Why is it **NL**-hard?

NL-hardness of *PATH*

Let $P \in$ **NL**. Given a nondeterministic log-space Turing machine M that computes P, we:

[1] W.l.o.g. we say there is just one accepting configuration.

NL-hardness of *PATH*

Let $P \in \mathbf{NL}$. Given a nondeterministic log-space Turing machine M that computes P, we:

1. Construct a control-flow graph G of all the reachable configurations of M for the given input.

[1] W.l.o.g. we say there is just one accepting configuration.

NL-hardness of *PATH*

Let $P \in$ **NL**. Given a nondeterministic log-space Turing machine M that computes P, we:

1. Construct a control-flow graph G of all the reachable configurations of M for the given input.
2. Ask if there is a *PATH* from the start configuration s to the accept configuration t[1].

[1] W.l.o.g. we say there is just one accepting configuration.

NL-hardness of *PATH*

Let $P \in$ **NL**. Given a nondeterministic log-space Turing machine M that computes P, we:

1. Construct a control-flow graph G of all the reachable configurations of M for the given input.
2. Ask if there is a *PATH* from the start configuration s to the accept configuration t[1].

The transducer needs only log space on the working tape to produce the graph on the output tape.

Thus..

As *PATH* $\in$ **NL** and *PATH* is **NL**-hard, *PATH* is **NL**-complete.

[1] W.l.o.g. we say there is just one accepting configuration.

NL-hardness of *PATH*

Let $P \in$ **NL**. Given a nondeterministic log-space Turing machine M that computes P, we:

1. Construct a control-flow graph G of all the reachable configurations of M for the given input.
2. Ask if there is a *PATH* from the start configuration s to the accept configuration t[1].

The transducer needs only log space on the working tape to produce the graph on the output tape.

Thus..

As *PATH* $\in$ **NL** and *PATH* is **NL**-hard, *PATH* is **NL**-complete.

Because *PATH* $\in$ **P**, we conclude **L** $\subseteq$ **NL** $\subseteq$ **P**!

[1] W.l.o.g. we say there is just one accepting configuration.

L vs NL

We hypothesise that we pay an exponential time penalty when we simulate nondeterministic machines with deterministic ones, but what about space?

L vs **NL**

We hypothesise that we pay an exponential time penalty when we simulate nondeterministic machines with deterministic ones, but what about space?

Note: We don't know that **NL** $\not\subseteq$ **L**, so it's possible there's no penalty.

L vs NL

We hypothesise that we pay an exponential time penalty when we simulate nondeterministic machines with deterministic ones, but what about space?

Note: We don't know that **NL** $\not\subseteq$ **L**, so it's possible there's no penalty.

Savitch's Theorem

Define a recursive algorithm $\mathrm{kpath}(s, t, k)$ that returns `true` iff there is a path of length k from s to t in a graph $G = (V, E)$.

- If $k = 0$, return $s = t$.
- If $k = 1$, return $(s, t) \in E$.
- If $k > 1$, for each $u \in V$:
  - If $\mathrm{kpath}(s, u, \lfloor \frac{k}{2} \rfloor) \wedge \mathrm{kpath}(u, t, \lceil \frac{k}{2} \rceil)$, return `true`.

L vs NL

We hypothesise that we pay an exponential time penalty when we simulate nondeterministic machines with deterministic ones, but what about space?
Note: We don't know that **NL** $\not\subseteq$ **L**, so it's possible there's no penalty.

Savitch's Theorem

Define a recursive algorithm $\mathrm{kpath}(s, t, k)$ that returns `true` iff there is a path of length k from s to t in a graph $G = (V, E)$.

- If $k = 0$, return $s = t$.
- If $k = 1$, return $(s, t) \in E$.
- If $k > 1$, for each $u \in V$:
  - If $\mathrm{kpath}(s, u, \lfloor \frac{k}{2} \rfloor) \wedge \mathrm{kpath}(u, t, \lceil \frac{k}{2} \rceil)$, return `true`.

kpath can compute *PATH* in $\log^2(|G|)$ space, so **NL** $\subseteq$ **L**^{2}. In general **NSPACE**$(f(n)) \subseteq$ **SPACE**$(f^2(n))$.

Certificates

Just as with **NP**, we can also characterise **NL** in terms of a *verifier* for *certificates* (candidate solutions):

Certificates

Just as with **NP**, we can also characterise **NL** in terms of a *verifier* for *certificates* (candidate solutions):

Theorem

A problem $P \in$ **NL** iff there is a *log-space verifier* for P-certificates.

A log-space verifier has three tapes:

1. A *input tape* that is read-only.
2. A *working tape* that is log-bounded.
3. A *certificate tape* that is *read-once* (left to right).

The size of the certificate we are verifying must be polynomial in the size of the input.

Certificates

Just as with **NP**, we can also characterise **NL** in terms of a *verifier* for *certificates* (candidate solutions):

Theorem

A problem $P \in$ **NL** iff there is a *log-space verifier* for P-certificates.
A log-space verifier has three tapes:

1. A *input tape* that is read-only.
2. A *working tape* that is log-bounded.
3. A *certificate tape* that is *read-once* (left to right).

The size of the certificate we are verifying must be polynomial in the size of the input.

Exercise: Show that this is equivalent to our **NSPACE** definition previously.

$PATH \in \mathbf{NL}$

Example

A certificate for *PATH* is a list of vertices $v_0, v_1, \ldots, v_k$ forming an acyclic path from s to t in a graph $G = (V, E)$. We can check with a log-space verifier that:

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

† Node names can be binary digits

$PATH \in \mathbf{NL}$

Example

A certificate for *PATH* is a list of vertices $v_0, v_1, \ldots, v_k$ forming an acyclic path from s to t in a graph $G = (V, E)$. We can check with a log-space verifier that:

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

We only read the certificate once, left to right, and it suffices to store two nodes in our working tape, so this is log space†.

† Node names can be binary digits

NL vs coNL

coNL is all problems whose complement is in **NL**.

Immerman-Szelepcsényi Theorem

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

More generally:

$$\mathbf{NSPACE}(f(x)) = \mathbf{coNSPACE}(f(x))$$

Thus:

$$\mathbf{PSPACE} = \mathbf{coPSPACE}$$

Proof of Immerman-Szelepcsényi

We prove this by showing $\overline{PATH} \in$ **NL**.

Intuition

Say I want to convince you (a verifier) that in a graph $G = (V, E)$, there is no path from s to t. I can do this by convincing you of the following two statements:

1. There are exactly $m_{|V|}$ distinct vertices reachable from s by paths of length $\leq |V|$.

Proof of Immerman-Szelepcsényi

We prove this by showing $\overline{PATH} \in$ **NL**.

Intuition

Say I want to convince you (a verifier) that in a graph $G = (V, E)$, there is no path from s to t. I can do this by convincing you of the following two statements:

1. There are exactly $m_{|V|}$ distinct vertices reachable from s by paths of length $\leq |V|$.
2. The target vertex t is *not* one of those $m_{|V|}$ vertices.

Proof of Immerman-Szelepcsényi

We prove this by showing $\overline{PATH} \in$ **NL**.

Intuition

Say I want to convince you (a verifier) that in a graph $G = (V, E)$, there is no path from s to t. I can do this by convincing you of the following two statements:

1. There are exactly $m_{|V|}$ distinct vertices reachable from s by paths of length $\leq |V|$.
2. The target vertex t is *not* one of those $m_{|V|}$ vertices.

So, what are the certificates?

Proof of Immerman-Szelepcsényi

We prove this by showing $\overline{PATH} \in \mathbf{NL}$.

Intuition

Say I want to convince you (a verifier) that in a graph $G = (V, E)$, there is no path from s to t. I can do this by convincing you of the following two statements:

1. There are exactly $m_{|V|}$ distinct vertices reachable from s by paths of length $\leq |V|$.
2. The target vertex t is *not* one of those $m_{|V|}$ vertices.

So, what are the certificates?

- For Part 2, we just give a list of $m_{|V|}$ distinct vertices that are not t, along with a certificate for each vertex v in our list that v is reachable from s by paths of length $\leq |V|$

Proof of Immerman-Szelepcsényi

We prove this by showing $\overline{PATH} \in \mathbf{NL}$.

Intuition

Say I want to convince you (a verifier) that in a graph $G = (V, E)$, there is no path from s to t. I can do this by convincing you of the following two statements:

1. There are exactly $m_{|V|}$ distinct vertices reachable from s by paths of length $\leq |V|$.
2. The target vertex t is *not* one of those $m_{|V|}$ vertices.

So, what are the certificates?

- For Part 2, we just give a list of $m_{|V|}$ distinct vertices that are not t, along with a certificate for each vertex v in our list that v is reachable from s by paths of length $\leq |V|$
- For Part 1, we do inductive counting...

Inductive Counting

I want to convince you (the verifier) of the following:

Certify this:

There are exactly $m_{|V|}$ distinct vertices reachable from s by paths of length $\leq |V|$.

To do this, I'll make an inductive argument:

Steps

For each $k = 0, \ldots, |V| - 1$, I'll show you (the verifier) that:
"if m_k vertices are reachable by paths of length $\leq k$,
then m_{k+1} vertices are reachable by paths of length $\leq k + 1$."

Sub-certificates

Steps

For each $k = 0, \ldots, |V| - 1$, I'll show you (the verifier) that:
"if m_k vertices are reachable by paths of length $\leq k$,
then m_{k+1} vertices are reachable by paths of length $\leq k + 1$."

The certificate for each step takes the form of a sub-certificate for each vertex $v \in V$:

Sub-certificates

Steps

For each $k = 0, \ldots, |V| - 1$, I'll show you (the verifier) that:
"if m_k vertices are reachable by paths of length $\leq k$,
then m_{k+1} vertices are reachable by paths of length $\leq k + 1$."

The certificate for each step takes the form of a sub-certificate for each vertex $v \in V$:

- If v is reachable by paths of length $\leq k + 1$, then it is just a path from s to v of length $\leq k + 1$

Sub-certificates

Steps

For each $k = 0, \ldots, |V| - 1$, I'll show you (the verifier) that:
"if m_k vertices are reachable by paths of length $\leq k$,
then m_{k+1} vertices are reachable by paths of length $\leq k + 1$."

The certificate for each step takes the form of a sub-certificate for each vertex $v \in V$:

- If v is reachable by paths of length $\leq k + 1$, then it is just a path from s to v of length $\leq k + 1$
- If v is not reachable by paths of length $\leq k + 1$, then it is a list of m_k distinct vertices that do not have an edge to v, and a certificate for each vertex v' in our list that v' is reachable from s by paths of length $\leq k$.

Sub-certificates

Steps

For each $k = 0, \ldots, |V| - 1$, I'll show you (the verifier) that:
"if m_k vertices are reachable by paths of length $\leq k$,
then m_{k+1} vertices are reachable by paths of length $\leq k + 1$."

The certificate for each step takes the form of a sub-certificate for each vertex $v \in V$:

- If v is reachable by paths of length $\leq k + 1$, then it is just a path from s to v of length $\leq k + 1$
- If v is not reachable by paths of length $\leq k + 1$, then it is a list of m_k distinct vertices that do not have an edge to v, and a certificate for each vertex v' in our list that v' is reachable from s by paths of length $\leq k$.

There should be exactly m_{k+1} "reachable" sub-certificates (our verifier will check this).

Sub-polynomiality

We have a finer-grained notion of reduction now, so we can make distinctions smaller than **P**:

Sub-polynomiality

We have a finer-grained notion of reduction now, so we can make distinctions smaller than **P**:

P-Completeness

A problem is **P**-complete iff it is in **P** and all problems in **P** can be log-space reduced to it.
Examples: Emptiness of CFGs, True Boolean Circuit Value, etc.

Sub-polynomiality

We have a finer-grained notion of reduction now, so we can make distinctions smaller than **P**:

P-Completeness

A problem is **P**-complete iff it is in **P** and all problems in **P** can be log-space reduced to it.
Examples: Emptiness of CFGs, True Boolean Circuit Value, etc.

Logarithmic Hierarchy

We can imagine a *logarithmic hierarchy* like the polynomial hierarchy, i.e. the languages decided by an alternating Turing machine in logarithmic space with a bounded number of alternations.

Sub-polynomiality

We have a finer-grained notion of reduction now, so we can make distinctions smaller than **P**:

P-Completeness

A problem is **P**-*complete* iff it is in **P** and all problems in **P** can be log-space reduced to it.
Examples: Emptiness of CFGs, True Boolean Circuit Value, etc.

Logarithmic Hierarchy

We can imagine a *logarithmic hierarchy* like the polynomial hierarchy, i.e. the languages decided by an alternating Turing machine in logarithmic space with a bounded number of alternations.
By Immerman-Szelepcsényi, the hierarchy collapses, i.e. $\Sigma_j^{\mathbf{L}} = \mathbf{NL}$ for all j.

Sub-polynomiality

We have a finer-grained notion of reduction now, so we can make distinctions smaller than **P**:

P-Completeness

A problem is **P**-complete iff it is in **P** and all problems in **P** can be log-space reduced to it.
Examples: Emptiness of CFGs, True Boolean Circuit Value, etc.

Logarithmic Hierarchy

We can imagine a *logarithmic hierarchy* like the polynomial hierarchy, i.e. the languages decided by an alternating Turing machine in logarithmic space with a bounded number of alternations.
By Immerman-Szelepcsényi, the hierarchy collapses, i.e. $\Sigma_j^{\mathbf{L}} = \mathbf{NL}$ for all j. But for unbounded alternations, $\mathbf{AL} = \mathbf{P}$.