Agenda

- Space complexity
- Classes PSPACE and NPSPACE
- Logspace reductions
- Class NL
- Alternation
Time vs. Space

- Computation is limited by:
  - Time
  - Memory

- So far, our focus has been on time complexity
- This lecture we will look at space complexity
Definition (Space usage)
Let $M$ be a Turing machine that halts on all inputs. We say that $M$ uses $S(n)$ space if for all inputs $x \in \{0, 1\}^*$, the machine $M$ visits at most $S(|x|)$ cells on the non-input tapes of $M$.

Notes on time and space:
- TM using $T(n)$ time can use at most $T(n)$ space
- For space, *sublinear* complexities makes sense
Space Complexity

Definition (Class SPACE)

Let $S: \mathbb{N} \rightarrow \mathbb{N}$ be a function. The class $\text{SPACE}(S(n))$ is the set of languages $L$ for which there exists a Turing machine $M$ and a constant $c > 0$ such that $M$ decides $L$ and uses $c \cdot S(n)$ space.

- $\text{DTIME}(T(n)) \subseteq \text{SPACE}(T(n))$
Definition (Class NSPACE)

Let $T: \mathbb{N} \rightarrow \mathbb{N}$ be a function. The class $\text{NSPACE}(S(n))$ is the set of languages $L$ for which there exists a nondeterministic Turing machine $M$ and a constant $c > 0$ such that $M$ decides $L$ and uses at most $c \cdot S(n)$ tape locations in any execution on an input of length $n$.

- $\text{SPACE}(S(n)) \subseteq \text{NSPACE}(S(n))$
Time vs. Space

Definition (Space-constructible function)
Let $S : \mathbb{N} \rightarrow \mathbb{N}$ be a function. We say that $S$ is space-constructible if there is a TM $M$ that computes the function $x \mapsto \llcorner S(|x|) \lrcorner$ in space $O(S(n))$, where $\llcorner n \lrcorner$ denotes the binary representation of the number $n$.

Theorem
For any space-constructible function $S : \mathbb{N} \rightarrow \mathbb{N}$, we have

$$\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))})$$
Configuration Graphs

- Let $M$ be a NDTM that uses $S(n)$ space and let $x \in L \subseteq \{0, 1\}^*$
- Define a directed configuration graph $G_{M,x}$ such that
  - Vertices represent possible configurations of $M$ on input $x$
  - There is a directed edge from $u$ to $v$ if $M$ can get from the configuration corresponding to $u$ to the configuration corresponding to $v$ in one step
- Each configuration can be encoded in $O(S(n))$ bits
- Thus, the configuration graph has at most $2^{O(S(n))}$ vertices
- Each vertex has two outgoing edges
- We can assume $G_{M,x}$ has only one accepting configuration by modifying $M$
**Time vs. Space**

**Theorem**

*For any space-constructible function $S: \mathbb{N} \rightarrow \mathbb{N}$, we have*

$$\text{NSPACE}(S(n)) \subseteq \text{DTIME}(2^{O(S(n))}).$$

**Proof:**

- We can now decide a language $L \in \text{NSPACE}(S(n))$ in time $2^{O(S(n))}$ as follows
- Let $M$ be NDTM witnessing $L \in \text{NSPACE}(S(n))$
- Construct the configuration graph $G_{M,x}$ in time $2^{O(S(n))}$
- Decide if we can reach the accepting configuration with a linear-time algorithm
Space Complexity Classes

Definition

- \( \text{PSPACE} = \bigcup_{c > 0} \text{SPACE}(n^c) \)
- \( \text{NPSPACE} = \bigcup_{c > 0} \text{NSPACE}(n^c) \)
- \( \text{L} = \text{SPACE}(\log n) \)
- \( \text{NL} = \text{NSPACE}(\log n) \)

- Relationships between time and space:
  - \( \text{L} \subseteq \text{NL} \subseteq \text{P} \)
  - \( \text{NP} \subseteq \text{PSPACE} \subseteq \text{NPSPACE} \subseteq \text{EXP} \)
PSPACE-completeness

Definition

- We say that a language $L \subseteq \{0, 1\}^*$ is PSPACE-hard if for any $L' \in \text{PSPACE}$ we have $L' \leq_p L$.
- We say that a language $L \subseteq \{0, 1\}^*$ is PSPACE-complete if $L$ is PSPACE-hard and $L \in \text{PSPACE}$. 
PSPACE-completeness: Examples

Definition (SPACE-TMSAT)

- **Instance:** A tuple $(M, x, 1^n)$, where $M$ is a Turing machine and $x \in \{0, 1\}^*$.
- **Question:** Does $M$ accept $x$ in space $n$?

**SPACE-TMSAT is PSPACE-complete**

- **Proof:** Easy.

**Many logic problems are PSPACE-complete**

**Generalised versions** of many games are PSPACE-complete

- What distinguishes PSPACE-complete and EXP-complete?
PSPACE-completeness: Examples

- A *quantified Boolean formula* (QBF) is a formula of form
  \[ Q_1 x_1 \ Q_2 x_2 \ldots, Q_n x_n \varphi(x_1, x_2, \ldots, x_n), \]
  where each \( Q_i \) is either \( \exists \) or \( \forall \) and \( \varphi \) is a Boolean formula over variables \( x_1, x_2, \ldots, x_n \)

  - **Example:** \( \forall x \exists y (x \land y) \lor (\neg x \land \neg y) \)
  - A QBF is always *true* or *false*

**Definition (TQBF)**

- **Instance:** A QBF \( \psi \).
- **Question:** Does \( \psi \) evaluate to true?
TQBF is PSPACE-complete

Basic idea for reducing $L \in \text{PSPACE}$ to TQBF:

- Let $M$ be a TM deciding $L$ in polynomial space $S(n)$ and let $x$ be an instance of $L$
- Define a QBF formula encoding the edges of the configuration graph $G_{M,x}$
- Use that to define a QBF formula encoding the reachability question from the starting state to the accepting state
- The final formula can be made to have size $O(S(n)^2)$ with some work
PSPACE-completeness: Examples

- Similar idea works for $L \in \text{NPSPACE}$
- TQBF is $\text{NPSPACE-complete}$
- It follows that $\text{PSPACE} = \text{NPSPACE}$!
Savitch’s Theorem

Theorem (W. Savitch 1970)

For any space-constructible function $S: \mathbb{N} \rightarrow \mathbb{N}$ with $S(n) > \log n$, we have that

$$\text{NSPACE}(S(n)) \subseteq \text{SPACE}(S(n)^2).$$

Proof idea:

- Solve reachability problem in the configuration graph $G_{M,x}$
- Can be done in space $O(S(n)^2)$ if the original NDTM uses space $O(S(n))$
Next, we want to discuss the $L$ vs. $NL$ question. We are working in the very restricted setting of logarithmic space:

- $O(\log n)$ bits can be used to count up to $n^c$.
- $O(\log n)$ bits can be used to refer to a single object from a collection with $n$ objects.
- In logarithmic space, we can store constant number of such counters.
Logspace Reductions

- Polynomial-time reductions are much stronger than logarithmic space
- Logarithmic space is not even enough to write the output of a polynomial reduction

Basic idea:

- Compute the reduction \( x \mapsto f(x) \) *implicitly* with logarithmic overhead
- Specifically, given \( x \) and \( i \leq |x| \), we can compute the \( i \)th bit of \( f(x) \) with logarithmic memory
- Memory used by the reduction can be re-used between subsequent calls to the reduction
Logspace Reductions

**Definition**

A function \( f : \{0, 1\}^* \rightarrow \{0, 1\}^* \) is *implicitly logspace computable* if there is \( c > 0 \) such that \( |f(x)| \leq |x|^c \) for all \( x \in \{0, 1\}^* \) and the languages

\[
L_f = \{(x, i) : f(x)_i = 1\}, \text{ and } \\
L'_f = \{(x, i) : |f(x)| \leq i\}
\]

are in \( L \).

**Definition**

A *logspace reduction* from \( L_1 \) to \( L_2 \) is an implicitly logspace computable function \( R : \{0, 1\}^* \rightarrow \{0, 1\}^* \) such that \( x \in L_1 \) if and only if \( R(x) \in L_2 \). Logspace reducibility is denoted by \( L_1 \leq_l L_2 \).
Logspace Reductions

Lemma

- If $L_1 \leq_l L_2$ and $L_2 \leq_l L_3$, then $L_1 \leq_l L_2$.
- If $L_1 \leq_l L_2$ and $L_2 \in L$, then $L_1 \in L$.

Proof:

- If $g$ and $f$ are implicitly logspace computable, then $h(x) = g(f(x))$ is implicitly logspace computable.
- This implies both of the claims.
**Definition**

A language $L \subseteq \{0, 1\}^*$ is in NL if there exists a deterministic Turing machine $M$ (called *logspace verifier*) with an additional special read-once input tape, and a polynomial $p : \mathbb{N} \to \mathbb{N}$ such that for all $x \in \{0, 1\}^*$ we have $x \in L$ if and only if there is $u \in \{0, 1\}^*$ with $|u| \leq p(|x|)$ such that $M(x, u) = 1$, where

- $M(x, u)$ denotes the output of $M$ when $x$ is written on the input tape and $u$ is written on the special read-once input tape, and
- $M$ uses at most $O(\log |x|)$ space on its working tapes.
**NL-completeness**

**Definition**

- We say that a language $L \subseteq \{0, 1\}^*$ is **NL-hard** if for any $L' \in \text{NL}$ we have $L' \leq_L L$.
- We say that a language $L \subseteq \{0, 1\}^*$ is **NL-complete** if $L$ is NL-hard and $L \in \text{NL}$. 
PATH

- **Instance**: Directed graph $G = (V, E)$, two vertices $s$ and $t$.
- **Question**: Is there a path from $s$ to $t$ in $G$?

- PATH is clearly in NL
- Corresponding problem for *undirected* graphs is in L
  - Very complicated proof
PATH is NL-complete

Theorem

PATH is NL-complete.

Proof sketch:

- Let $L \in \text{NL}$ be a language decided by a logspace NDTM $M$
- **Reduction from $L$ to PATH:** map $x$ to the path problem on configuration graph $G_{M,x}$
- Vertices of $G_{M,x}$ can be described with $O(\log |x|)$ bits; each bit of the adjacency matrix of $G_{M,x}$ can be computed in logarithmic space
**Definition**

\[ \text{coNL} = \left\{ L \subseteq \{0, 1\}^* : \overline{L} \in \text{NL} \right\} \]

- Complete languages for coNL are the complements of NL-complete languages

**Theorem**

PATH is NL-complete.

- **Non-existence** of a path can be verified in logarithmic space
- NL = coNL
Complementary Space Classes

Theorem (N. Immerman, R. Szelepcsényi 1987)

For any space-constructible $S : \mathbb{N} \to \mathbb{N}$ with $S(n) > \log n$, we have that

$$\text{NSPACE}(S(n)) = \text{coNSPACE}(S(n)).$$

Proof idea:

- For a no-instance of $L \in \text{NSPACE}(S(n))$, prove that there is no path from starting configuration to accepting configuration in the configuration graph.
- Almost the same proof as for NL-completeness of PATH.
Alternating Turing Machines

- Alternation is an important generalisation of nondeterminism.
- In a nondeterministic computation each configuration is an implicit **OR** of its successor configurations: i.e. a configuration “leads to acceptance” iff at least one of its successors does.
- The idea is to allow both **OR** and **AND** configurations in a tree of configurations generated by a NTM $N$ computing on input $x$. 
Definition
An *alternating* Turing machine $N$ is a nondeterministic Turing machine where the set of states $K$ is partitioned into two sets $K = K_{\text{AND}} \cup K_{\text{OR}}$.

Given the tree of configurations of $N$ on input $x$, the *eventually accepting configurations* of $N$ are defined recursively:

1. Any leaf configuration with state “yes” is eventually accepting.
2. A configuration with state in $K_{\text{AND}}$ is eventually accepting iff all its successors are.
3. A configuration with state in $K_{\text{OR}}$ is eventually accepting iff at least one of its successors is.

$N$ *accepts* $x$ iff its initial configuration is eventually accepting.
Alternation-Based Complexity Classes

Definition

An alternating Turing machine $N$ decides a language $L$ iff $N$ accepts all strings $x \in L$ and rejects all strings $x \notin L$.

- It is straightforward to define $\text{ATIME}(f(n))$ and $\text{ASPACE}(f(n))$; and using them, e.g. $\text{AP} = \text{ATIME}(n^k)$, $\text{AL} = \text{ASPACE}(\log n)$ etc.
- Roughly speaking, alternating time classes correspond to deterministic space and alternating space classes correspond to deterministic time but one exponential higher.

Theorem

$\text{AL} = \text{P}$, $\text{AP} = \text{PSPACE}$, $\text{APSPACE} = \text{EXP}$, …
Alternation and The Polynomial Time Hierarchy

Denote by $\Sigma_i P$ (resp. $\Pi_i P$), $i \geq 1$, the family of languages decided by polynomially time-bounded alternating Turing machines whose every computation satisfies the following conditions:

- The initial state belongs to $K_{OR}$ (resp. $K_{AND}$).
- The computation *alternates* from a state in $K_{OR}$ to a state in $K_{AND}$ or vice versa at most $i - 1$ times.

By definition, set also $\Sigma_0 P = \Pi_0 P = P$.

**Theorem**

For every $i \geq 0$, $\Sigma_i P = \Sigma_i^P$ and $\Pi_i P = \Pi_i^P$. 
Lecture 10: Summary

- Space complexity
- Configuration graphs
- \( \text{PSPACE} \) and \( \text{PSPACE} \)-completeness
- \( \text{PSPACE} = \text{NPSPACE} \)
- \( \text{L} \) and \( \text{NL} \)
- Logspace reductions
- \( \text{NL} = \text{coNL} \)
- Alternation