Class NL
Reducing Problems

We have seen that polynomial time reduction between problems is a very useful concept for studying relative complexity of problems. It allowed us to distinguish a class of problems, \textbf{NP}, which includes many important problems and is viewed as the class of hard problems.

We are going to do the same for space complexity classes: \textbf{NL} and \textbf{PSPACE}.

There is a problem:

Polynomial time reduction is too powerful.
Log-Space Reduction

A transducer is a 3-tape Turing Machine such that

- the first tape is an input tape, it is never overwritten
- the second tape is a working tape
- the third tape is an output tape, no instruction of the transition function uses the content of this tape

The space complexity of such a machine is the number of cells on the working tape visited during a computation

A function $f : \Sigma^* \rightarrow \Sigma^*$ is said to be log-space computable if there is a transducer computing $f$ in $O(\log n)$
**Definition** A language $A$ is log-space reducible to a language $B$, denoted $A \leq_L B$, if a log-space computable function $f$ exists such that for all $x \in \Sigma^*$

$$x \in A \iff f(x) \in B$$

Note that a function computable in log-space is computable in polynomial time, so

$$A \leq_L B \Rightarrow A \leq B$$
Completeness

Definition
A language $L$ is said to be $\mathsf{NL}$-complete if $L \in \mathsf{NL}$ and, for any $A \in \mathsf{NL}$,

$$A \leq_L L$$

Definition
A language $L$ is said to be $\mathsf{P}$-complete if $L \in \mathsf{P}$ and, for any $A \in \mathsf{P}$,

$$A \leq_L L$$
NL-Completeness of REACHABILITY

Theorem
Reachability is NL-complete

Corollary
NL ⊆ P

Proof Idea
For any non-deterministic log-space machine $NT$, and any input $x$, construct the graph $NT(x)$. Its vertices are possible configurations of $NT$ using at most $\log(|x|)$ cells on the working tape; its edges are possible transitions between configurations.

Then $NT$ accepts the input $x$ if and only if the accepting configuration is reachable from the initial configuration
Proof

• Let $A$ be a language in $\text{NL}$

• Let $NT$ be a non-deterministic Turing Machine that decides $A$ with space complexity $\log n$

• Choose an encoding for the computation $NT(x)$ that uses $k\log(|x|)$ symbols for each configuration

• Let $C_0$ be the initial configuration, and $C_a$ be the accepting configuration

• We represent $NT(x)$ by giving first the list of vertices, and then a list of edges
• Our transducer $T$ does the following

- $T$ goes through all possible strings of length $k\log(|x|)$ and, if the string properly encodes a configuration of $NT$, prints it on the output tape

- Then $T$ goes through all possible pairs of strings of length $k\log(|x|)$. For each pair $(C_1, C_2)$ it checks if both strings are legal encodings of configurations of $NT$, and if $C_1$ can yield $C_2$. If yes then it prints out the pair on the output tape

• Both operations can be done in log-space because the first step requires storing only the current string (the strings can be listed in lexicographical order). Similarly, the second step requires storing two strings, and (possibly) some counters

• $NT$ accepts $x$ if and only if there is a path in $NT(x)$ from $C_0$ to $C_a$
Log-Space reductions and \( L \)

We take it for granted that \( P \) is closed under polynomial-time reductions.

We can expect that \( L \) is closed under log-space reductions, but it is much less trivial.

**Theorem**

If \( A \leq_L B \) and \( B \in L \), then \( A \in L \)

**Corollary**

If any NL-complete language belongs to \( L \), then \( L = \text{NL} \)
Proof

Let \( M \) be a Turing Machine solving \( B \) in log-space, and let \( T \) be a log-space transducer reducing \( A \) to \( B \).

It is not possible to construct a log-space decider for \( A \) just combining \( M \) and \( T \), because the output of \( T \) may require more than log-space.

Instead, we do the following:

Let \( f \) be the function computed by \( T \).

On an input \( x \), a decider \( M' \) for \( A \):

- Simulates \( M \) on \( f(x) \).
- When it needs to read the \( l \)-th symbol of \( f(x) \), \( M' \) simulates \( T \) on \( x \), but ignores all outputs except for the \( l \)-th symbol.
P-completeness

Using log-space reductions we can study the finer structure of the class \( \mathsf{P} \).

A clause \( Z_1 \lor Z_2 \lor \ldots \lor Z_k \) is said to be a Horn if it contains at most one positive literal

\[
\neg X_1 \lor X_2 \lor \neg X_3 \quad \equiv \ (X_1 \land X_3) \rightarrow X_2
\]

\[
\neg X_1 \lor \neg X_2 \quad \equiv \ (X_1 \land X_2) \rightarrow \text{true}
\]

A CNF is said to be Horn if every its clause is Horn

**Horn-SAT**

Instance: A Horn CNF \( \Phi \).

Question: Is \( \Phi \) satisfiable?
Theorem

Horn-SAT is P-complete
Computability and Complexity

Time and Space

- All Languages
- Decidable Languages
- NP-completeness
- P-completeness
- NL-completeness
- Time and Space
**NL and coNL**

For a language $L$ over an alphabet $\Sigma$, we denote $\overline{L}$ the complement of $L$, the language $\Sigma^* - L$.

**Definition**

The class of languages $L$ such that $\overline{L}$ can be solved by a non-deterministic log-space Turing machine verifier is called **coNL**.

**Theorem**

$NL = coNL$
Proof

The properties of \textbf{NL} and \textbf{coNL} are similar to those of \textbf{NP} and \textbf{coNP}:

- if \( L \) is NL-complete then \( \overline{L} \) is coNL-complete
- if a coNL-complete problem belongs to \textbf{NL} then \( \text{NL} = \text{coNL} \)

Reachability is NL-complete.

Therefore it is enough to show that \textbf{No-Reachability} is in NL.

In order to do this, we have to find a non-deterministic algorithm that proves in log-space that there is no path between two specified vertices in a graph.
Counting the number of reachable vertices

Given a graph $G$ and two its vertices $s$ and $t$; let $n$ be the number of vertices of $G$

First, we count the number $c$ of vertices reachable from $s$

Let $A_i$ be the set of all vertices connected to $s$ with a path of length at most $i$, and $c_i = |A_i|$

Clearly, $\{s\} = A_0 \subseteq A_1 \subseteq \ldots \subseteq A_n$, and $c_n = c$

We compute the numbers $c_0, c_1, \ldots, c_n$ inductively
Suppose $c_i$ is known. The following algorithm non-deterministically either compute $c_{i+1}$ or reject

- set $c_{i+1} = 0$
- for every vertex $v$ from $G$ do
  - set $m = c_i$
  - for every vertex $w$ from $G$ non-deterministically do or not do
    - check whether or not $w \in A_i$ using random walk
    - if not then reject
    - if yes then set $m = m - 1$
    - if there is the edge $(w, v)$ and a witness for $v$ is not found yet, set $c_{i+1} = c_{i+1} + 1$
- if $m \neq 0$ reject
- output $c_{i+1}$
Checking Reachability

Given $G$, $s$, $t$ and $c$

- set $m = c$
- for every vertex $v$ from $G$ non-deterministically do or not do
  - check whether or not $v$ is reachable from $s$ using random walk
  - if not then reject
  - if yes then set $m = m - 1$
  - if $v = t$ then reject
- if $m \neq 0$ reject
- accept