Non-Deterministic Space
Non-deterministic Machines

Recall that if $NT$ is a non-deterministic Turing Machine, then $NT(x)$ denotes the tree of configurations which can be entered with input $x$, and $NT$ accepts $x$ if there is some accepting path in $NT(x)$.

**Definition**

The space complexity of a non-deterministic Turing Machine $NT$ is the function $\text{NSpace}_{NT}$ such that $\text{NSpace}_{NT}(x)$ is the minimal number of cells visited in an accepting path of $NT(x)$ if there is one, otherwise it is the minimal number of cells in the rejecting paths.

(If not all paths of $NT(x)$ halt, then $\text{NSpace}_{NT}(x)$ is undefined)
Nondeterministic Space Complexity

Definition
For any function $f$, we say that the nondeterministic space complexity of a decidable language $L$ is in $O(f)$ if there exists a nondeterministic Turing Machine $NT$ which decides $L$, and constants $n_0$ and $c$ such that for all inputs $x$ with $|x| > n_0$

$$\text{NSpace}_{NT}(x) \leq cf(|x|)$$

Definition
The nondeterministic space complexity class $\text{NSPACE}[f]$ is defined to be the class of all languages with nondeterministic space complexity in $O(f)$
Definition of NPSPACE

\[
\text{NPSPACE} = \bigcup_{k \geq 0} \text{NSPACE}[n^k]
\]
Savitch’s Theorem

Unlike time, it can easily be shown that non-determinism does not reduce the space requirements very much:

**Theorem (Savitch)**

If \( s(n) \geq \log n \), then

\[
\text{NSPACE}[s] \subseteq \text{SPACE}[s^2]
\]

**Corollary**

\[
PSPACE = \text{NPSPACE}
\]
Proof (for $s(n) \geq n$)

- Let $L$ be a language in $\text{NSPACE}[s]$
- Let $NT$ be a non-deterministic Turing Machine that decides $L$ with space complexity $s$
- Choose an encoding for the computation $NT(x)$ that uses $ks(|x|)$ symbols for each configuration
- Let $C_0$ be the initial configuration, and $C_a$ be the accepting configuration
- Define a Boolean function $reach(C,C',j)$ which is $\text{true}$ if and only if configuration $C'$ can be reached from configuration $C$ in at most $2^j$ steps
- To decide whether or not $x \in L$ we must determine whether or not $reach(C_0,C_a,ks(|x|))$ is $\text{true}$
We can calculate $reach(C_0, C_a, ks(|x|))$ in $O(s(|x|)^2)$ space, using a divide-and-conquer algorithm:

$reach(C, C', j)$

1. If $j=0$ then if $C=C'$, or $C'$ can be reached from $C$ in one step, then return true, else return false.

2. For each configuration $C''$, if $reach(C, C'', j-1)$ and $reach(C'', C', j-1)$, then return true.

3. Return false

The depth of recursion is $O(s(|x|))$ and each recursive call requires $O(s(|x|))$ space for the parameters.
Logarithmic Space

Since polynomial space is so powerful, it is natural to consider more restricted space complexity classes

Even linear space is enough to solve Satisfiability

**Definition**

\[
L = \text{SPACE}[\log n]
\]

\[
NL = \text{NSPACE}[\log n]
\]
Problems in L and NL

What sort of problems are in \textbf{L} and \textbf{NL}?

In logarithmic space we can store:

- a fixed number of counters (up to length of input)
- a fixed number of pointers to positions in the input string

Therefore in deterministic log-space we can solve problems that require a fixed number of counters and/or pointers for solving; in non-deterministic log-space we can solve problems that require a fixed number of counters/pointers for verifying a solution
Examples (L)

Palindromes:

We need to keep two counters

\[ L = \{0^k 1^k \mid k \in \mathbb{N} \} \]

First count the number of 0s, then count 1s, subtracting from the previous number one by one. If the result is 0, accept; otherwise, reject.

Brackets (if brackets in an expression positioned correctly):

We need only a counter of brackets currently open. If this counter gets negative, reject; otherwise accept if and only if the last value of the counter is zero
Examples (NL)

The first problem defined on this course was Reachability¹

This can be solved by the following non-deterministic algorithm:

- Define a counter and initialize it to the number of vertices in the graph
- Define a pointer to hold the ``current vertex” and initialize it to the start vertex
- While the counter is non-zero
  - If the current vertex equals the target vertex, return yes
  - Non-deterministically choose a vertex which is connected to the current vertex
  - Update the pointer to this vertex and decrement the counter
- Return no

¹Also known as Path
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

$$A \leq_L B \implies A \leq B$$
Completeness

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

Definition
A language $L$ is said to be **P-complete** if $A \in \text{P}$ and, for any $A \in \text{P}$, $A \leq_L L$
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 \( L = NL \)
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 $P$.

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:

$$\neg X_1 \lor X_2 \lor \neg X_3 \equiv (X_1 \land X_3) \rightarrow X_2$$

$$\neg X_1 \lor \neg X_2 \equiv (X_1 \land X_2) \rightarrow \text{false}$$

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
Time and Space

Complexity

Decidable Languages

P

PSPACE

NL

NP

L

All Languages

NL-complete

P-complete

NP-complete
ERROR: undefined
OFFENDING COMMAND:

STACK: