Efficient Algorithms and Problem Complexity
– Introduction to Problem Complexity –

Frank Drewes
Department of Computing Science
Umeå University
Today’s Menu

1. About Problem Complexity

2. Decision Problems and the Class P
What is Problem Complexity?

- Obviously (?) some problems can be solved more efficiently than others.
- As always, we measure efficiency depending on input size. [WHY?]
- Efficiency can refer to very different aspects:
  - solving the problem quickly,
  - solving it in little memory space,
  - solving it on a parallel computer using few processors,
  - solving it in a distributed way with little communication,
  - solving it with a small circuit,
  - solving it using a short program,
  - …
- There are at least 3 versions of each of these:
  - worst case
  - average case
  - best case
But What Is a Problem?

A problem is a function \( A: \mathcal{I} \rightarrow \mathcal{O} \) from a set \( \mathcal{I} \) of inputs to a set of outputs \( \mathcal{O} \).

- If \( \mathcal{O} = \{\text{yes}, \text{no}\} \), then the problem is a decision problem.
- A decision problem \( A \) can be identified with \( \{I \in \mathcal{I} \mid A(I) = \text{yes}\} \).
- If we want to stress that \( A \) is not necessarily a decision problem, we call it a function problem.
- More about both types of problems later...
About Problem Complexity

And, by the Way, What Is an Algorithm?

When studying problem complexity, we must be somewhat more precise about what constitutes an algorithm:

- We do not discuss concrete algorithms, but algorithms we do not know ⇒ it is important to agree on a suitable model of computation.
- The model must allow us to measure size and time in a reasonable way.
- We may use pseudocode, but must be aware of the underlying model.
- Traditional choice:
  - Model: Turing machine
  - Input/output: strings
  - Size (of input): length of string
  - Time consumed: number of steps
- The random access machine (RAM) is more similar to a modern computer.
Deterministic Random Access Machines (RAMs)

A deterministic random access machine \( M \) has a program counter \( \kappa \) and registers \( R_0, R_1, R_2 \ldots \) storing non-negative integers \( R_0, R_1, R_2, \ldots \).

An address is an expression of the form \( i \) or \( \langle i \rangle \) \((i \in \mathbb{N})\), where \( R_{\langle i \rangle} = R_{R_i} \).

A value is an expression of the form \( i \) \((i \in \mathbb{N})\) or \( R_\alpha \), where \( \alpha \) is an address.

\( M \) is given by its program, a finite sequence \( I_1 \cdots I_m \) of instructions of the following types, where \( \alpha \) is a address, \( \beta \) is a value, and \( k \in \{1, \ldots, m+1\} \):

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_\alpha \leftarrow \beta )</td>
<td>store ( \beta ) in ( R_\alpha )</td>
</tr>
<tr>
<td>add ( \beta R_\alpha )</td>
<td>add ( \beta ) to ( R_\alpha )</td>
</tr>
<tr>
<td>sub ( \beta R_\alpha )</td>
<td>subtract ( \beta ) from ( R_\alpha ) (0 if ( \beta \geq R_\alpha ))</td>
</tr>
<tr>
<td>if ( \beta ) then goto ( k )</td>
<td>set program counter to ( k ) if ( \beta &gt; 0 )</td>
</tr>
</tbody>
</table>
Deterministic Random Access Machines (RAMs)

A random access machine works as follows:

- The input is a sequence $x = a_1 \cdots a_n \in \mathbb{N}^*$ which is stored in registers $R_1, \ldots, R_n$, whereas $R_0$ initially contains $n$ and $R_i = 0$ for all $i > n$.
- The program counter $\kappa$ is initially set to 1.
- Each step executes $I_\kappa$ and increases $\kappa$ by 1 (unless a jump occurs).
- The program terminates when $\kappa = m + 1$.
- The output is $R_1 \cdots R_{R_0}$, which is denoted by $M(x)$.

Measuring input length and running time:

- The input length is $\sum_{j=1}^{n} |\text{bin}(a_i)|$, where $\text{bin}(a_i)$ is $a_i$ in binary notation.
- The running time is the number of steps executed until $\kappa = m + 1$.

**Question:** Are these two conventions reasonable?
Polynomial-time RAMs

- A RAM runs in polynomial time if there is some \( k \in \mathbb{N} \) such that the running time of \( M \) is \( O(n^k) \) for all inputs \( x \) of length \( n \).
- Easy exercise: Show that, in this case, \( M \) runs in time \( n^l + c \) for some \( l, c \in \mathbb{N} \) (i.e., we may assume that the constant factor is 1).
- A decision problem \( A \) is decided by a RAM \( M \) if, for all inputs \( x \),

\[
M(x) = \begin{cases} 
1 & x \in A \\
0 & \text{otherwise}.
\end{cases}
\]

- The class of all problems that can be decided in polynomial time by a RAM is denoted by \( \mathbb{P} \).
A Few Remarks about Encodings

- Encodings are important to be aware of, even though they are usually not made explicit.
- A RAM that computes \( f(n) \) in time \( O(n^k) \) is not polynomial \( \Rightarrow \) be careful when encoding numbers in unary (or don’t do that)!
- All decision problems can be encoded as sets of strings (languages).
- Encoding each symbol \( \sigma_i \) in an alphabet \( \Sigma = \{\sigma_1, \ldots, \sigma_k\} \) as \( i \), strings can be given as input to a RAM. (Think ASCII or UTF.)
- For problems involving numbers, other encodings may be more natural.
- Reasonable encodings can be transformed into each other in polynomial time (usually \( O(n) \) or \( O(n^2) \)).
A Few Remarks about Encodings

But we usually talk about instances of a problem, disregarding non-instances. So, what about inputs that are not valid instances at all?

Theorem (disregarding invalid inputs)

Let $A$ be a decision problem. If there is a RAM such that

- for all inputs $x$, $M(x) = 1$ if and only if $x \in A$, and
- $M$ runs in polynomial time on inputs in $A$,

then $A$ can be decided in polynomial time (i.e., $A \in P$).

Proof sketch: Suppose $M$ runs in time less than $n^k + c$ on inputs in $A$. Construct a new RAM $M'$ that works as follows:

1. With input $x$, start by computing $|x|^k + c$ in some register $R_z$ (easy to do in time $O(n^k)$).
2. Continue precisely like $M$, but decrease $R_z$ by 1 in each step.
3. Moreover, if $R_z$ ever reaches 0, stop with output 0.
Another important thing is to be able to combine algorithms from already existing ones. For $P$ to be a robust class, it should be closed under such constructions.
Composition Theorem

If $M_1$ and $M_2$ are polynomial-time RAMs, the composition $M_2 \circ M_1$, where $M_2$ is applied to the output of $M_1$, can be computed by a polynomial-time RAM as well.

Proof sketch: Suppose $M_1$ and $M_2$ run in time $O(n^k)$ and $O(n^l)$, for some $k, l \geq 1$. Construct a new RAM $M$ that works in the obvious way:

1. With input $x$, start by computing $M_1(x)$.
2. When $M_1$ has terminated, continue by working like $M_2$.

Clearly, phase 1 runs in time $O(n^k)$

⇒ the length of $M_1(x)$ is $O(n^{2k})$ [WHY? (Compare with the textbook)]

⇒ phase 2 runs in time $O(n^{2kl}) = O(n^{2kl})$

⇒ the total time is $O(n^{2kl})$. 

Frank Drewes (Umeå University)
Please read Section 10.1 in the textbook.