Solution

Computational Complexity – Homework 1

Discussed on 26.04.2019.

Exercise 1.1

Recall the definition of the Landau notation for $f, g : \mathbb{N} \to \mathbb{N}$:

$$\begin{array}{lll} f \in \mathcal{O}(g) & :\Leftrightarrow & \exists c \in (0,\infty)\exists n_0 \in \mathbb{N}\forall n > n_0 \,:\, f(n) \leq c \cdot g(n). \\ f \in \Omega(g) & :\Leftrightarrow & g \in \mathcal{O}(f) \\ f \in \Theta(g) & :\Leftrightarrow & f \in \mathcal{O}(g) \wedge f \in \Omega(g) \\ f \in o(g) & :\Leftrightarrow & \forall \epsilon \in (0,\infty)\exists n_0 \in \mathbb{N}\forall n > n_0 \,:\, f(n) \leq \epsilon \cdot g(n) \\ f \in \omega(g) & :\Leftrightarrow & g \in o(f). \end{array}$$

Remark: Some authors prefer to write $f = \mathcal{O}(g)$ instead of $f \in \mathcal{O}(g)$. As $\mathcal{O}(g)$ is set of functions, while f is a function, the latter is more precise than the former.

(a) Assume f, g are strictly positive functions, i.e., $f(n), g(n) > 0$ for all $n \in \mathbb{N}$. Show or disprove:

- $f \in \Theta(g)$ if and only if there exist $c_1, c_2 \in (0,\infty)$ such that $c_1 \leq f(n)/g(n) \leq c_2$ for almost all $n \in \mathbb{N}$. ("almost all" is equivalent to "except for finitely many").
- $f \in o(g)$ if and only if $\lim_{n\to\infty} f(n)/g(n) = 0$.

(b) Let f and g be any two of the following functions. Describe their relation using the Landau notation.

$$\begin{array}{lll} (a)\, n^2 & (b)\, n^3 & (c)\, n^2 \log n \\ (d)\, 2^n & (e)\, n^n & (f)\, n^{\log n} \\ (g)\, 2^{2^n} & (h)\, 2^{2^{n+1}} & (j)\, n^2 \text{ if } n \text{ is odd}, 2^n \text{ otherwise.} \end{array}$$

(c) Describe (and prove) the relations between $2^{\mathcal{O}(n)}$, $\mathcal{O}(2^n)$ and $2^{n^{\mathcal{O}(1)}}$.

Solution:

- $$\begin{array}{ll} & f \in \Theta(g) \\ \Leftrightarrow & f \in \mathcal{O}(g) \wedge g \in \mathcal{O}(f) \\ \Leftrightarrow & \exists c_f > 0 \exists n_f \forall n \geq n_f :\, f(n) \leq c_f g(n) \wedge \exists c_g > 0 \exists n_g \forall n \geq n_g :\, g(n) \leq c_g f(n) \\ \overset{*}{\Leftrightarrow} & \exists c_f, c_g > 0 \exists n_0 \forall n \geq n_0 :\, f(n) \leq c_f g(n) \wedge g(n) \leq c_g f(n) \\ \Leftrightarrow & \exists c_f, c_g > 0 \exists n_0 \forall n \geq n_0 :\, \frac{1}{c_g} \leq \frac{f(n)}{g(n)} \leq c_f \\ \overset{**}{\Leftrightarrow} & \exists c_1, c_2 > 0 \exists n_0 \forall n \geq n_0 :\, c_1 \leq \frac{f(n)}{g(n)} \leq c_2 \end{array}$$

  *: ($\Rightarrow$) set $n_0 := \max(n_f, n_g)$. ($\Leftarrow$) set $n_f := n_g := n_0$.

  **: $c_f = c_2$, $c_1 = 1/c_f$.

- $$\begin{array}{ll} & f \in o(g) \\ \Leftrightarrow & \forall c > 0 \exists n_c \forall n \geq n_c \,:\, f(n) \leq c g(n) \\ \Leftrightarrow & \forall c > 0 \exists n_c \forall n \geq n_c \,:\, \frac{f(n)}{g(n)} \leq c \\ \overset{*}{\Leftrightarrow} & \forall \epsilon > 0 \exists n_\epsilon \forall n \geq n_\epsilon \,:\, \left|\frac{f(n)}{g(n)}\right| < \epsilon \\ \Leftrightarrow & \lim_{n\to\infty} \frac{f(n)}{g(n)} = 0. \end{array}$$

  *: Note that (i) $f(n), g(n) > 0$ and (ii) ($\Rightarrow$) set $c := 0.9\epsilon$, ($\Leftarrow$) $\epsilon := c$.

- Without any guarantee! Lower half defined by symmetry.

	n^2	n^3	$n^2 \log n$	2^n	n^n	$n^{\log n}$	2^{2^n}	$2^{2^{n+1}}$	$f(n) := (n \text{ odd? } n^2 : 2^n)$
n^2	$\Theta(n^2)$	$o(n^3)$	$o(n^2 \log n)$	$o(2^n)$	$o(n^n)$	$o(n^{\log n})$	$o(2^{2^n})$	$o(2^{2^{n+1}})$	$\mathcal{O}(f(n))$
n^3		$\Theta(n^3)$	$\omega(n^2 \log n)$	$o(2^n)$	$o(n^n)$	$o(n^{\log n})$	$o(2^{2^n})$	$o(2^{2^{n+1}})$	$--$
$n^2 \log n$			$\Theta(n^2 \log n)$	$o(2^n)$	$o(n^n)$	$o(n^{\log n})$	$o(2^{2^n})$	$o(2^{2^{n+1}})$	$--$
2^n				$\Theta(2^n)$	$o(n^n)$	$\omega(n^{\log n})^*$	$o(2^{2^n})$	$o(2^{2^{n+1}})$	$\Omega(f(n))$
n^n					$\Theta(n^n)$	$\omega(n^{\log n})$	$o(2^{2^n})$	$o(2^{2^{n+1}})$	$\omega(f(n))$
$n^{\log n}$						$\Theta(n^{\log n})$	$o(2^{2^n})$	$o(2^{2^{n+1}})$	$--$
2^{2^n}							$\Theta(2^{2^n})$	$o(2^{2^{n+1}})$	$\omega(f(n))$
$2^{2^{n+1}}$								$\Theta(2^{2^{n+1}})$	$\omega(f(n))$
$f(n)$									$\Theta(f(n))$

*:

$$2^n \in \omega(n^{\log n}) \Leftrightarrow n^{\log n} \in o(2^n) \Leftrightarrow \lim_{n\to\infty} \frac{n^{\log n}}{2^n} = 0 \Leftrightarrow \lim_{n\to\infty} 2^{(\log n)^2 - n} = 0 \Leftrightarrow \lim_{n\to\infty} (\log n)^2 - n = -\infty \Leftrightarrow \lim_{n\to\infty} \frac{(\log n)^2}{n} = 0.$$

Using l'Hospital:

$$\lim_{n\to\infty} \frac{(\log n)^2}{n} = 0 \Leftrightarrow \lim_{n\to\infty} \frac{2(\log n)\frac{1}{n}}{1} = 0 \Leftrightarrow \lim_{n\to\infty} \frac{\log n}{n} = 0 \Leftrightarrow \lim_{n\to\infty} \frac{1/n}{1} = 0.$$

Remark: Similarly, one shows that $(\log n)^k \in o(n)$ for any $k \in \mathbb{N}$.

- We have $\mathcal{O}(2^n) \subsetneq 2^{\mathcal{O}(n)} \subsetneq 2^{n^{\mathcal{O}(1)}}$. Proof: Let $f \in \mathcal{O}(2^n)$, then there exists $c \geq 1$ such that $f(n) \leq c2^n$ for all large enough n. Hence $f(n) \leq 2^{\log c + n} \leq 2^{cn}$ for all large enough n and thus $f \in 2^{\mathcal{O}(n)}$. Similarly we have $2^{cn} \leq 2^{n^c}$ for $c \geq 1$ and large enough n which shows the second inclusion. Observe that the inclusions are strict, since for example $2^{3n} \notin \mathcal{O}(2^n)$ and $2^{n^5} \notin \mathcal{O}(2^{\mathcal{O}(n)})$

Exercise 1.2

Consider the following language on $\{0,1\}$:

$$L = \{u0v0w \in \{0,1\}^* \mid u,v,w \in \{1\}^* \wedge |v| \leq |w| \leq |u| \wedge \exists k \in \{|v|, \ldots, |w|\} : k \text{ divides } |u|\}.$$

Its characteristic function f_L is then

$$f_L : \{0,1\}^* \to \{0,1\} : x \mapsto \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

Construct a Turing machine which computes f_L in time $\mathcal{O}(n^k)$ for some fixed $k > 0$.

Solution: We give an informal description of the behaviour of a TM deciding L:

- 1. Step: Check that the input x is of the form 1*01*01*.

  If x is not of the required from, output 0 and halt.

- 2. Step: Copy u, v, and w parts of x to work tapes 1 to 3.
- 3. Step: Check that $|v| \leq |w| \leq |u|$.

  If x does not satisfy the requirement on u, v, w, output 0 and halt.

- 4. Step: As long as work tape 4 contains less 1s than work tape 1 (u) append the content of work tape 2 (v) to the content of work tape 4.
- 5. Step: Check whether work tapes 1 and 4 contain the same number of 1s.

  If this is the case, output 1 and halt.

- 6. Step: Empty work tape 4.
- 7. Step: Append an 1 to the content of work tape 2.
- 8. Step: Check that work tape 2 contains at most as many 1s as work tape 3.

  If this does not hold, output 0 and halt.

- Go to Step 4.

One easily checks that every "macro step" can be done by a TM using at most $\mathcal{O}(|x|)$ many steps.

Exercise 1.3

If $f : \{0,1\}^* \to \{0,1\}$ is computable by a TM with a finite alphabet Γ then it is also computable by a TM with alphabet $\Sigma = \{0, 1, \Box, \triangleright\}$, moreover, with only a polynomial overhead.

Prove the statement above. Does the same hold for infinite Γ? Does the same hold for $\Sigma = \{1, \Box, \triangleright\}$?

Solution: In the lecture, you have seen that a k-tape TM can be simulated by a single tape TM with only a polynomial overhead. We will make use of this fact.

First, note that any element of Γ can be encoded using $k = \lceil \log |\Gamma| \rceil$ letters of binary alphabet. We can thus simulate the working tape with symbols of Γ by k tapes with symbols of Σ.

Exercise 1.4

Call a Turing machine M *oblivious* if the positions of its heads at the i^{th} step of its computation on input x *depend only* on i and $|x|$, but not x itself.

Let $L \in \mathbf{DTIME}(T)$ with $T : \mathbb{N} \to \mathbb{N}$ time-constructible. Show that there is an oblivious Turing machine which decides L in time $O(T^2)$.

Solution: Let M be a Turing machine deciding L in time $T(n)$. Further, let M_T be a Turing machine calculating T. As T is required to be time-constructible, we find such a M_T.

We sketch how to construct from M and M_T an oblivious Turing machine O which decides L in time $\mathcal{O}(T(n)^2)$. For simplicity, we assume that M is a one-tape TM; for this, we allow M to also write to the input tape. O is not required to have only a single tape, still we allow O to write to its input tape, too.

The behaviour of O is as follows:

(a) First, O reads the input once from left to right, copies for every symbol read an 1 to the input tape of M_T, and, finally, moves all heads back to the left-most position.

(b) It then starts M_T on input $1^{|x|}$. For every step done by M_T, O also writes an 1 to two tapes, called **space** and **time** in the following. After M_T has terminated, the content of both **space** and **time** is $\triangleright 1^{T(|x|)}$.

(c) Then, O simulates exactly $T(|x|)$ steps of M, i.e., after simulating a single step of M, O moves the head of **time** one place to the left, the simulation terminates when the head of **time** hits $\triangleright$.

    A single step of M is simulated as follows:

    O remembers the position of the head of M on the input tape by some apropriate symbol, e.g., if Γ is the tape alphabet used by M, then O might use the symbols $\Gamma \cup \hat{\Gamma}$ where $\hat{\Gamma} = \{\hat{\gamma} \mid \gamma \in \Gamma\}$.

    In order for O to be able to simulate a step of M, O needs to remember the control state of M and (at most three) symbols $\mu\hat{\gamma}\nu$ within the 1-step vicinity of the head of the M. (This is finite information and therefore can be stored in the control of O. Check this by yourself!)

    As M is time-bounded by T, O knows that the head of M can never move more than $T(|x|)$ steps to the right. Hence, O can scan the whole tape content of M by moving its input head $T(|x|)$ steps to the right and then back again. The **space** tape can be used for this.

    Within this scan, O can remember the three symbols $\mu\hat{\gamma}\nu$, determine from the next step of M and change its tape content accordingly.

    E.g.: assume that a given point of time M is in the configuration $(q, \triangleright ab\hat{c}d))$ with $\delta_M(q, c) = (q', e, \rightarrow)$, i.e., M makes the following step:

    $$(q, \triangleright ab\hat{c}d) \to (q', \triangleright abe\hat{d}).$$

    O simulates this step as follows: it remebers in its control state the control state q of M plus the last three symbols read. O scans its input tape from left to right until one step after $\hat{c}$ is encountered. Then O remembers the necessary symbols $b\hat{c}d$ and the state q so it can determine the next step of M. As M moves right, O can immediately replace d to $\hat{d}$, then it moves on to the right until $T(n)$ steps are made (reading **space** in lockstep). O then moves its input head back to the left-most position. On its way back O waits on $\hat{d}$ so it can replace the symbol $\hat{c}$ left of it by e. Similarly, O can simulate a step where M moves its head to the left.

It is left to the reader to check that O is indeed oblivious.