Algorithmic Discrete Mathematics
2. Exercise Sheet

TECHNISCHE UNIVERSITÄT DARMSTADT

Department of Mathematics
Andreas Paffenholz
Silke Horn

SS 2013
2 May 2013

Groupwork

Exercise G1

(a) Show that every tree T has at least $\Delta(T)$ leaves.

(b) Show that a tree without a vertex of degree 2 has more leaves than other vertices.

Solution:

(a) From a vertex of maximal degree $\Delta(T)$ follow a path in each direction until you reach a leave. These leaves are pairwise distinct.

(b) Let n and m denote the number of vertices and edges, respectively. Then $m = n - 1$. If we denote by n_i the number of vertices of degree i we get

$$2(n-1) = 2m = \sum_{v \in V} \deg v = n_1 + \sum_{i \geq 3} i n_i \geq n_1 + 3(n - n_1)$$

and hence $2n_1 \geq n + 2$.

Exercise G2

Recall that an automorphism of a graph $G = (V, E)$ is a bijective map $\phi : V \to V$ such that $\{u, v\} \in E \Leftrightarrow \{\phi(u), \phi(v)\} \in E$ for any $u, v \in V$.

Show that every automorphism of a tree fixes a vertex or an edge.

Solution: Let $T = (V, E)$ be a tree. We prove the result by induction over $n = |V|$.

For $n = 1$ the assertion is trivial.

For $n = 2$ there are two automorphisms of T: The identity map fixes everything; the map that switches the two vertices fixes the edge.

Now assume $n \geq 3$. Let ϕ be an automorphism of T. Then ϕ permutes the leaves of T since automorphisms preserves the degree. *I.e.*, if $v_1, \ldots, v_k \in V$ are the leaves of T, then $\phi(v_i) \in \{v_1, \ldots, v_k\}$ for each i.

Now consider the tree $T' = T - \{v_1, \ldots, v_k\}$. If we restrict ϕ to T' we obtain an automorphism of T'. By the induction hypothesis ϕ fixes a vertex or an edge of T' and hence ϕ also fixes a vertex or an edge of T.

Exercise G3

Let $f, g : \mathbb{N} \to \mathbb{N}$ be two functions such that $g \in \Omega(f)$. Assume that two algorithms are given:

- Algorithm A has a running time of $\mathcal{O}(f)$.
- Algorithm B has a running time of $\mathcal{O}(g)$.

Consider the following two algorithms:

```
Algorithm 1:
   Input: $n \in \mathbb{N}$
 1 $s \leftarrow 0$
 2 for $i = 0, \ldots, 100$ do
 3     Run Algorithm $A$
 4 for $i = 0, \ldots, 2n$ do
 5     Run Algorithm $B$
```

```
Algorithm 2:
   Input: $n \in \mathbb{N}$
 1 if $n \geq 100$ then
 2     Run Algorithm $A$
 3 else
 4     Run Algorithm $B$
```

Estimate the running times as accurately as possible.

Solution: The running time of Algorithm 1 is given by $100\mathcal{O}(f) + 2n\mathcal{O}(g) = \mathcal{O}(f) + \mathcal{O}(ng) = \mathcal{O}(ng)$ since $g \in \Omega(f)$.

For the running time of Algorithm 2 we have to keep in mind that we are only interested in the asymptotic behavior as $n \to \infty$ and hence only the first part of the If-statement matters. The running time is thus $\mathcal{O}(f)$.

Exercise G4

Let $f, g : \mathbb{N} \to \mathbb{N}$ be two functions and a a constant. Prove:

(a) $f \in \mathcal{O}(f)$

(b) $a \cdot \mathcal{O}(f) \subseteq \mathcal{O}(f)$

(c) $\mathcal{O}(f) + \mathcal{O}(f) \subseteq \mathcal{O}(f)$

(d) $\mathcal{O}(f) \cdot \mathcal{O}(g) \subseteq \mathcal{O}(fg)$

(e) $f \cdot \mathcal{O}(g) \subseteq \mathcal{O}(fg)$

(f) $\max(f, g) \in \Theta(f + g)$

Hint: For two sets A, B addition and multiplication are defined point-wise, *e.g.* for $A = \{a, b\}$ and $B = \{c, d\}$: $A + B = \{a + c, a + d, b + c, b + d\}$, $A \cdot B = \{ac, ad, bc, bd\}$.

Solution:

(a) With $c = 1$ we have $f(n) \leq c \cdot f(n)$ for all $n \geq 0$.

(b) Let $h \in a \cdot O(f)$. Then there is $h' \in \mathcal{O}(f)$ such that $h = ah'$. Moreover, there are n_0, c such that $h'(n) \leq cf(n)$ for all $n \geq n_0$. With $c' = ac$ this yields $h(n) = ah'(n) \leq acf(n) = c'f(n)$.

(c) Let $h \in \mathcal{O}(f) + \mathcal{O}(f)$. Then there are $h_1, h_2 \in \mathcal{O}(f)$ with $h = h_1 + h_2$. Moreover, there are $c_1, c_2, n_{0,1}, n_{0,2}$ such that $h_i(n) \leq c_i f(n)$ for $n \geq n_{0,i}$ and $i = 1, 2$. With $n_0 = \max\{n_{0,1}, n_{0,2}\}$ and $c = c_1 + c_2$ we then have $h(n) = h_1(n) + h_2(n) \leq c_1 f(n) + c_2 f(n) = cf(n)$ for all $n \geq n_0$.

(d) Let $h \in \mathcal{O}(f) \cdot \mathcal{O}(g)$. Then there are $h_1 \in \mathcal{O}(f), h_2 \in \mathcal{O}(g)$ with $h = h_1 h_2$. Moreover, there are $c_1, c_2, n_{0,1}, n_{0,2}$ such that $h_1(n) \leq c_1 f(n)$ für $n \geq n_{0,1}$ and $h_2(n) \leq c_2 g(n)$ for $n \geq n_{0,2}$. With $n_0 = \max\{n_{0,1}, n_{0,2}\}$ and $c = c_1 c_2$ we then get $h(n) = h_1(n) h_2(n) \leq c_1 f(n) c_2 g(n) = c(fg)(n)$ for all $n \geq n_0$.

(e) Because of $f \in \mathcal{O}(f)$ this assertion follows from (d).

(f) We have $f(n) + g(n) \geq \max(f, g)(n) \geq \frac{1}{2}(f(n) + g(n))$ for all $n \in \mathbb{N}$. Hence $c_1(f + g)(n) \geq \max(f, g)(n) \geq c_2(f + g)(n)$ for $c_1 = 1, c_2 = \frac{1}{2}$. Thus $\max(f, g) \in \Theta(f + g)$.

Homework

Exercise H1 (5 points)

Recall that a *subgraph* of a graph $G = (V, E)$ is a graph $H = (W, F)$ with $W \subseteq V, F \subseteq E$. H is *induced* if $F = E \cap \binom{W}{2}$. Assume that a graph $G = (V, E)$ with $|V| = n$ and $|E| \geq 3$ without isolated vertices does not have an induced subgraph with two edges. Show that $G = K_n, n \geq 3$, *i.e.,* G is a complete graph on n vertices.

Solution: Assume that there are two vertices u, v that are not connected by an edge. Since G has no induced subgraph with two edges, there is no vertex that is connected with both u and v. Since G does not have isolated vertices, there are two edges $\{a, u\}, \{b, v\} \in E$. If $\{a, b\} \notin E$, then the induced subgraph on a, b, u, v has two edges. Otherwise the induced subgraph on u, a, b has two edges. This yields a contradiction. Hence G is a complete graph.

Exercise H2 (5 points)

Sort the functions

$$n, \quad n^3, \quad \sqrt{n}, \quad n!, \quad 2^n, \quad n^n$$

according to their complexity in ascending order using o-notation.

Reminder:

$$f \in o(g) \iff \forall c > 0 \ \exists n_0 \in \mathbb{N} \ \forall n \geq n_0 : 0 \leq f(n) < cg(n)$$

Solution: The order is $\sqrt{n}, n, n^3, 2^n, n!, n^n$.

$\sqrt{n} \in o(n)$: Choose $n_0 = \left\lceil \frac{1}{c^2} \right\rceil + 1$.

$n \in o(n^3)$: Choose $n_0 = \left\lceil \sqrt{\frac{1}{c}} \right\rceil + 1$.

$n^3 \in o(2^n)$: We have to solve $n^3 < c2^n$ for n. For $n \geq 30$ we have $n^3 < 2^{n/2}$. Thus, we can consider

$$\frac{2^n}{n^3} > 2^{n/2} > \frac{1}{c},$$

which holds for $n > 2\log_2 \frac{1}{c}$. Hence we may choose $n_0 = \max\left\{\left\lceil 2\log_2 \frac{1}{c} \right\rceil, 30\right\}$.

$2^n \in o(n!)$: For $n \geq 6$ it holds that $2^n < (n-1)!$. For $n > \frac{1}{c}$ then also $cn! > \frac{n!}{n} = (n-1)! > 2^n$. We may thus choose $n_0 = \max\{6, \left\lceil \frac{1}{c} \right\rceil\}$.

$n! \in o(n^n)$: Choose $n_0 = \left\lceil \frac{1}{c} \right\rceil + 1$.

Exercise H3 (5 points)
Consider the following algorithm:

```
Algorithm 3:
Input: $n \in \mathbb{N}$
$d \leftarrow 2$
$q \leftarrow n$
while $q > d$ do
    $q \leftarrow n/d$
    if $\lceil q \rceil = q$ then
        return $d$
    else
        $d \leftarrow d + 1$
return 0
```

What does it do? Estimate its running time.

Solution: The algorithms returns 0 if n is prime (or 1) and its smallest (positive) non-trivial divisor otherwise. To this end, the algorithm test whether n can be divided by $d = 2, 3, \ldots$ without remainder. If this is the case for some $d \in \mathbb{N}$, the algorithm stops and outputs d. Otherwise the algorithm stops if $\frac{n}{d} < d$. Hence we always have $d \leq \sqrt{n}$ and the running time is $\mathcal{O}(\sqrt{n})$. (In fact, one can even show that it is $\Theta(\sqrt{n})$.)

Exercise H4 (5 points)
Let $G = (V, E)$ be a connected Eulerian graph. Devise an algorithm that returns an Eulerian tour in G, prove its correctness and estimate its running time in $\mathcal{O}$-notation.

Solution: We assume that the graph is given as a (mutable) adjacency list A, *i.e.,* $A(v)$ is the list of neighbors of v.

```
Algorithm 4:
Input: adjacency list $A$ of a connected Eulerian graph $G = (V, E)$
$v \leftarrow$ some vertex
$W \leftarrow [v]$
repeat
    $W' \leftarrow [v]$
    $w \leftarrow v$
    while $A(w)$ is non-empty do
        $w' \leftarrow A(w)(0)$
        append $w'$ to $W'$
        delete $w'$ from $A(w)$ and $w$ from $A(w')$
        $w \leftarrow w'$
    insert $W'$ into $W$ at $v$
    while $A(v)$ is empty and $v$ is not the last element of $W$ do
        $v \leftarrow$ next vertex in $W$
until $v = W(0)$
return $W$
```

The algorithm constructs an Eulerian trail W by building it up from shorter trails W'.

In each iteration it starts at some vertex v and then constructs a longest possible trail W' along previously ununsed edges. This always yields a closed trail, that is inserted into W at vertex v. *I.e.*, if $W = (v_0, \dots, v, \dots, v_n = v_0)$ before, then we set $W = (v_0, \dots, v, W', v, \dots, v_n = v_0)$. ($W$ could be implemented as a linked list to make this efficient.)

Now we move along W until we find a new vertex that has unused edges left and repeat the process until we reach the end of W.

The running time is $\mathcal{O}(m)$ since we traverse every edge once when inserting it into W. Moreover, in the search for vertices with unused edges we traverse W a second time.

Note that in the proof of the running we need to carefully choose the data structures: First of all, we could store our trails W and W' in linked lists, so that insertion can be done in constant time. Moreover, we need to ensure that deleting vertices from adjacency lists can be done efficiently. To this end, we could store adjacency lists as linked lists (so that deletion works in constant time) and endow each entry v' of an adjacency list $A(v)$ with a pointer to the element v in the adjacency list $A(v)$ (the same edge). This makes it possible to delete w from $A(w')$ in constant time in line 9.