Karlsruher Institut für Technologie — Prof. Dr. Jörn Müller-Quade
Institut für Theoretische Informatik — Björn Kaidel, Sebastian Schlag, Sascha Witt

1. Übungsblatt zu Algorithmen I im SoSe 2017

http://crypto.iti.kit.edu/index.php?id=799
{bjoern.kaidel,sascha.witt}@kit.edu

Musterlösungen

Aufgabe 0.1 (*Wichtig – Einige Worte zur Übungsblattabgabe*)

Für die Übungsblattabgabe vereinbaren wir einige Regeln, um die Organisation und Korrektur zu vereinfachen, sowie um Ihnen transparent einen klaren und eindeutigen Rahmen zur Abgabe zu geben. Es wird einen Klausurbonus für erreichte Punkte geben (siehe Absprachen dazu in der Vorlesung und Übung) – bitte halten Sie sich daher an diese Regeln:

- Bitte geben Sie ihr Übungsblatt rechtzeitig ab. Die Abgabe muss immer spätestens bis Dienstags eine Woche nach der Ausgabe, 12:45 Uhr, erfolgt sein. Der genaue Zeitpunkt ist jeweils eindeutig auf dem Übungsblatt angegeben (dieses erste Übungsblatt stellt eine Ausnahme dar, da wir es aufgrund des Feiertages früher veröffentlicht haben. Sie müssen es bis zum 09.05.2017 abgeben). Später abgegebene Blätter werden nicht bewertet.
- Sie können ihr Übungsblatt in (vorher bei ihrem/r Tutor/in gemeldeten) Zweiergruppen abgeben. Ihr/e Partner/in muss dabei im gleichen Tutorium wie Sie sein. Siehe hierzu auch die Absprachen in der Vorlesung/Übung.
- Bitte heften Sie ihre Lösungen zur Abgabe zusammen.
- Verwenden Sie das angehängte Deckblatt (siehe Ende des Übungsblatts), auf dem Sie *deutlich* Ihren Namen, Matrikel- und Tutoriennummer angeben. Unterschreiben Sie das Deckblatt an angegebener Stelle. Wir behalten uns einen Punkteabzug vor, falls Sie wiederholt Übungsblätter ohne Deckblatt abgeben sollten.
- Markieren Sie bitte jedes Blatt mit Ihrem Namen, Matrikel- und Tutoriennummer.
- Bitte verwenden Sie dokumentenechte Stifte und insbesondere keine Bleistifte und rote Stifte.
- Geben Sie ihr Übungsblatt bitte in einer handschriftlichen Reinschrift ab. Wir behalten uns vor unleserliche Abgaben nicht zu korrigieren. Sie müssen Ihre Lösungen nicht extra nochmals sauber abschreiben. Wir bitten Sie aber darum, kein Schmierpapier abzugeben und Lösungsversuche, die nicht gewertet werden sollen, eindeutig als solche zu kennzeichnen (z.B. durch deutliches Durchstreichen).
- Abschreiben ist untersagt. Ein erster Verstoß wird *mindestens* mit 0 Punkten für die abgeschriebene Aufgabe und einer Verwarnung geahndet. Wir behalten uns bei Verstößen gegen diese Regel vor, Ihnen keinen Klausurbonus zu gewähren.

Aufgabe 0.2 (*Kommunikationsmöglichkeiten*)

Machen Sie sich mit den Kommunikationsmöglichkeiten auf der Vorlesungswebsite unter `https://crypto.iti.kit.edu/index.php?id=799` vertraut. Es gibt ein ILIAS-Forum, einen anonymen Feedbackkasten und eine Mailingliste. Bitte melden Sie sich für die Mailingliste an, damit die Übungsleitung eine Möglichkeit hat, Sie bei dringenden Informationen direkt erreichen zu können. Das ILIAS-Forum ist zum Austausch und zur Diskussion gedacht. Das Posten von vollständigen Aufgabenlösungen ist

verboten, Hilfestellungen und Diskussionen sind aber erwünscht. Der Feedbackkasten dient dazu, dass Sie eine anonyme Kontaktmöglichkeit zu uns haben, falls Ihnen bezüglich der Vorlesung oder der Übung Kritik, Verbesserungsvorschläge usw. auf dem Herzen liegen, die Sie lieber anonym vorbringen möchten. Bitte verschicken Sie über den Feedbackkasten keine Fragen an uns. Da er anonym ist, haben wir keine Möglichkeit, Ihnen auf ihre Fragen zu antworten. Benutzen Sie für Fragen am Besten das Forum, falls die Frage von allgemeinem Interesse ist und ansonsten kontaktieren Sie bitte ihre/n Tutor/in oder die Übungsleitung per Mail.

Aufgabe 1 (*$\mathcal{O}$-Kalkül,* $3+4+1=8$ *Punkte*)

Geben Sie jeweils an, ob $\mathcal{O}$, Ω oder Θ für die jeweiligen Kästchen zutreffend ist und beweisen Sie die entstandene Aussage. Verwenden Sie wann immer möglich Θ.

a) $n^3 \in \square\,(n^2)\,, \quad 2^n \in \square\,(4^n)\,, \quad n+\log n \in \square\,(n)$

b) $\log(n+1) \in \square\,(\log n)\,, \quad \log n \in \square\,(\log(n)^2)\,, \quad n^n \in \square\,\left(2^{(n^2)}\right)$

c) Gilt die folgende Implikation? Beweisen Sie ihre Antwort!

$$\forall f, g\colon \mathbb{N} \to \mathbb{N}\colon\ f(n) \in \mathrm{o}(g(n)) \Rightarrow \log(f(n)) \in \mathrm{o}((\log(g(n))))$$

Musterlösung:

a)
- Es gilt $n^3 \in \Omega\,(n^2)$. Zu zeigen ist $\exists c>0\ \exists n_0>0\ \forall n \geq n_0\colon c\cdot n^2 \leq n^3$. Wir zeigen die Aussage für $c=1$ und $n_0=2$ (zur Übung) mit vollständiger Induktion:
  - Induktionsanfang: Es gilt $2^2 = 4 < 8 = 2^3$.
  - Induktionsvoraussetzung: Für ein $n \in \mathbb{N}$ gilt $n^2 \leq n^3$.
  - Induktionsschritt: Es gilt $(n+1)^2 = n^2+2n+1 \leq n^3+3n^2+3n+1 = (n+1)^3$.
- Es gilt $2^n \in \mathcal{O}(4^n)$, da
$$\lim_{n\to\infty} \frac{2^n}{4^n} = \lim_{n\to\infty} \frac{2^n}{2^n\cdot 2^n} = \lim_{n\to\infty} \frac{1}{2^n} = 0.$$
- Es gilt $n+\log n \in \Theta(n)$. Zu zeigen ist $\exists c_1, c_2>0\ \exists n_0>0\ \forall n \geq n_0\colon 0 \leq c_1\cdot n \leq n+\log n \leq c_2\cdot n$. Wir zeigen die Aussage für $c_1=1, c_2=2$ und $n_0=1$ mit vollständiger Induktion und gehen dafür vom Logarithmus zur Basis 2 aus (wie in der Übung argumentiert, ist die Basis des Logarithmus bei Betrachtungen im O-Kalkül nicht wichtig, wir verwenden sie hier aber konkret in den Umformungen):
  - Induktionsanfang: Es gilt $0 \leq 1 = 1+\log 1 \leq 2\cdot 1$.
  - Induktionsvoraussetzung: Für ein $n \in \mathbb{N}$ gilt $1\cdot n \leq n+\log n \leq 2\cdot n$.
  - Induktionsschritt: Es gilt $n+1 \overset{IV}{\leq} (n+\log n)+1 \leq (n+1)+\log(n+1) \overset{n\geq 1}{\leq} n+1+\log(2n) = n+1+\log(2)+\log(n) = n+\log(n)+2 \overset{IV}{\leq} 2n+2 \leq 2(n+1)$.

b)
- Es gilt $\log(n+1) \in \Theta(\log(n))$, da
$$\begin{aligned}\frac{\log(n+1)}{\log n} &= \frac{\log\left((1+\frac{1}{n})\cdot n\right)}{\log n}\\ &= \frac{\log(1+\frac{1}{n})+\log n}{\log n}\\ &= \frac{\log(1+\frac{1}{n})}{\log n}+1.\end{aligned}$$
und somit gilt, dass
$$0 < \lim_{n\to\infty} \frac{\log(n+1)}{\log n} = \lim_{n\to\infty} \frac{\overbrace{\log(1+\frac{1}{n})}^{\to 0}}{\log n}+1 = 1 < \infty.$$

- Es gilt $\log n \in \mathcal{O}((\log n)^2)$, da $\lim_{n\to\infty} \frac{\log n}{(\log n)^2} = \lim_{n\to\infty} \frac{1}{\log n} = 0$.
- Es gilt $n^n \in \mathcal{O}\left(2^{(n^2)}\right)$, da

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

c) Die Aussage gilt nicht, ein Gegenbeispiel ist gegeben durch $f\colon \mathbb{N} \to \mathbb{N},\ n \mapsto n$ und $g\colon \mathbb{N} \to \mathbb{N},\ n \mapsto n^2$. Es gilt $n \in \mathrm{o}(n^2)$, da $\lim_{n\to\infty} \frac{n}{n^2} = 0$, aber es gilt

$$\lim_{n\to\infty} \frac{\log n}{\log(n^2)} = \lim_{n\to\infty} \frac{\log n}{\log(n^2)} = \lim_{n\to\infty} \frac{\log n}{2 \cdot \log(n)} = \frac{1}{2} \neq 0$$

und somit $\log n \notin \mathrm{o}(\log(n^2))$.

Aufgabe 2 (*Schleifeninvariante,* 4 *Punkte*)

Gegeben sei folgender Algorithmus in Pseudocode. Gehen Sie dabei vereinfachend davon aus, dass alle auftretenden Zahlen jeweils in eine Speicherzelle passen.

Function $f(n, m : \mathbb{N}) : (\mathbb{N}_0, \mathbb{N}_0)$
  a=0 : $\mathbb{N}_0$
  b=m : $\mathbb{N}_0$
  c=1 : $\mathbb{N}_0$
  while $m - c \cdot n \geq 0$ **do**
    $a := c$
    $c := c + 1$
    $b := m - a \cdot n$
  return (a, b)

a) Berechnen Sie die Laufzeit des Algorithmus im $\mathcal{O}$-Kalkül in Abhängigkeit von m. Begründen Sie ihre Behauptung.

b) Was berechnet der Algorithmus?

c) Nennen und beweisen Sie eine Schleifeninvariante für die while-Schleife. Diese soll m in Abhängigkeit von n beschreiben. Sie können für diese Aufgabe von $m \geq n$ ausgehen.

Musterlösung:

a) Im Worst-Case ist $n = 1$, da $n \in \mathbb{N}$. Da c bei jedem Schleifendurchlauf um 1 erhöht wird, ist die Bedingung $m - c \cdot n > 0$ also spätestens nach m Schleifendurchläufen nicht mehr erfüllt. Somit ist die Laufzeit des Algorithmus in $\mathcal{O}(m)$.

b) Ist $m < n$ gibt der Algorithmus $(0, m)$ aus. Ansonsten berechnet er eine Division von m durch n mit Rest, d.h. er berechnet (a, b) so, dass $m = a \cdot n + b$ und $b = m \bmod n$.

c) Eine mögliche Schleifeninvariante ist $m = a \cdot n + b$. Wir beweisen diese:

- Vor der Schleife und am Beginn des ersten Schleifendurchlaufs gilt $a = 0$ und $b = m$.
- Es bezeichne a_i, b_i, c_i die Werte der Variablen a, b und c am Anfang des i-ten Schleifendurchlaufs. Wir müssen zeigen, dass $m = a_i \cdot n + b_i$ gilt. Am Anfang des i-ten Schleifendurchlaufes gilt $a_i = c_{i-1}$ und $b_i = m - a_i \cdot n = m - c_{i-1} \cdot n$. Damit folgt:

$$a_i \cdot n + b_i = c_{i-1} \cdot n + (m - c_{i-1} \cdot n) = m.$$

- Da c in jedem Durchgang um einen Zähler erhöht wird und n, m natürliche Zahlen sind, terminiert die Schleife immer. Bei Beendigung der Schleife gilt, dass $a = \max(\{i \ : \ m - i \cdot n > 0\}$, woraus auch $b = m \bmod n$ folgt.

Aufgabe 3 (*Karatsuba-Ofman-Multiplikation,* 3 *Punkte*)

Multiplizieren Sie 1652 mit 5789 (das sind Dezimalzahlen!) nach dem Algorithmus von Karatsuba und Ofman. Verwenden Sie das unten angegebene Schema zum Aufschrieb eines Multiplikationsaufrufs und stellen Sie die Multiplikation wie in der Übung als Baum dar.
Hinweis: Führen Sie nur Additionen sowie einstellige Multiplikationen direkt aus. Mehrstellige Multiplikationen müssen rekursiv mit Karatsuba-Ofman berechnet werden.
Schema zum Aufschrieb eines Multiplikationsaufrufs:

$a \cdot b$	$a_1 \cdot b_1 =$	$\text{recMult}(a_1, b_1)$
	$a_0 \cdot b_0 =$	$\text{recMult}(a_0, b_0)$
e	$(a_1 + a_0) \cdot (b_1 + b_0) =$	$\text{recMult}((a_1 + a_0), (b_1 + b_0))$

Musterlösung:
Der Algorithmus:

1: **Function** *recMult*(a, b)
2: **assert** a und b und es sei $k = \lceil n/2 \rceil$
3: **if** $n = 1$ **then return** $a \cdot b$
4: Schreibe a als $a_1 \cdot B^k + a_0$
5: Schreibe b als $b_1 \cdot B^k + b_0$
6: $c_{11} :=$ recMult(a_1, b_1)
7: $c_{00} :=$ recMult(a_0, b_0)
8: $c_{0110} :=$ recMult$((a_1 + a_0), (b_1 + b_0))$
9: $e := c_{11} \cdot B^{2k} + (c_{0110} - c_{11} - c_{00})B^k + c_{00}$
10: **return** e

Berechnung:
Siehe Rekursionsbaum in Abbildung 1.

Aufgabe 4 (*Algorithmenentwurf,* 3 *Punkte*)

Gegeben sei ein Array M von natürlichen Zahlen mit $|M| \geq 2$. Entwerfen Sie einen Algorithmus in Pseudocode, der zwei Indizies $i, j \in \{0, \ldots, |M| - 1\}$ mit $i \neq j$ berechnet, sodass

$$M[i] + M[j] = \max(\{M[x] + M[y] \mid x, y \in \{0, \ldots, |M| - 1\}, x \neq y\}).$$

Ihr Algorithmus muss asymptotisch Laufzeit $\mathcal{O}(n)$ haben ($n := |M|$). Begründen Sie, warum der Algorithmus diese Laufzeit erreicht.

Musterlösung:
Die Forderung an i und j ist genau dann erfüllt, wenn oBdA gilt, dass

$$M[i] = \max(M) \text{ und}$$
$$M[j] = \max(M \setminus M[i]).$$

Der Algorithmus bestimmt also den Index i des größten Elements, sowie den Index j des zweitgrößten Elements in M. Dazu durchläuft der Algorithmus das Array in einer Schleife und merkt sich den Index des bisher größten Elements. Findet er ein Element das größer als das aktuelle Maximum ist, so aktualisiert er diese Variablen und speichert den alten Index als Index des aktuell zweitgrößten Elements.

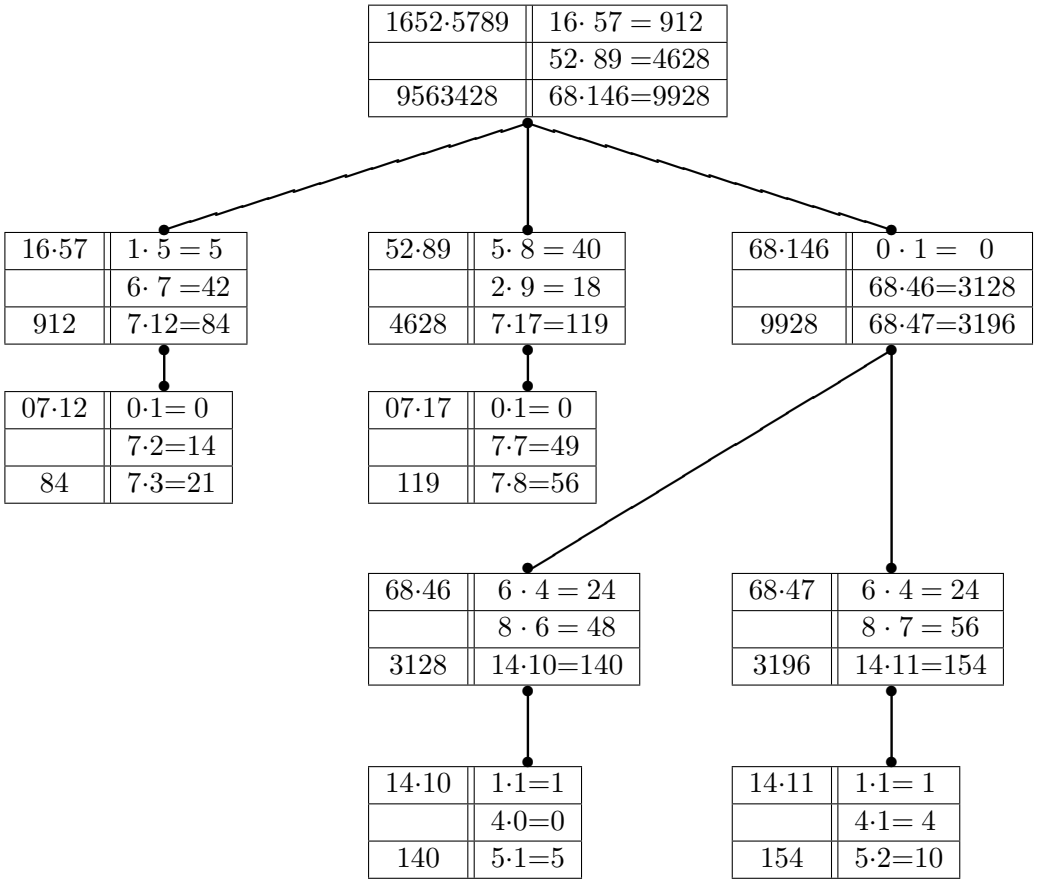

Abbildung 1: Berechnungsbaum

Findet er ein Element welches größer als das zweitgrößte, aber nicht größer als das größte Element ist, so aktualisiert er den Index für das zweitgrößte Element. Diese Schleife betrachtet jedes Element von M maximal einmal, entsprechend läuft der Algorithmus in $\mathcal{O}(n)$.

Der Pseudocode dazu könnte so aussehen:

Function findMaxAndSecondMax(M : **Array** $[0..n-1]$ of $\mathbb{N}$) :
    assert $n \geq 2$
    $i := 1$ −− i = Index of (current) max
    $j := 0$ −− j = Index of second biggest element
    if $M[0] > M[1]$ **then**
        $i := 0$
        $j := 1$
    endif

    for $u := 2$ **to** $n-1$ **do**
        if $M[u] > M[i]$ **then**
            j := i
            i := u
        else if $M[u] > M[j]$ **then** −− new „second maximum“ found
            j := u
        endif
    return (i, j)