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

2. Übungsblatt zu Algorithmen I im SoSe 2017

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

Musterlösungen

Aufgabe 1 (*Rekurrenzen,* $2+3=5$ *Punkte*)

a) Gegeben sei folgende Rekurrenz:

$$T(n) = \begin{cases} 2 & \text{für } n < 4, \\ 6 \cdot T\left(\frac{n}{4}\right) + n^2 & \text{für } n \geq 4 \end{cases}$$

Zeigen Sie durch vollständige Induktion, dass $T(n) \leq 2n^2 + 2n$ gilt, falls sich n als $n = 4^k$, $n = 2 \cdot 4^k$ oder $n = 3 \cdot 4^k$ (mit $k \in N_0$) schreiben lässt. Verwenden Sie dazu im Induktionsschritt den Schritt $n \rightsquigarrow 4n$ und bedenken Sie, dass Sie im Induktionsanfang mehrere Basisfälle betrachten müssen.

b) Gegeben sei folgende Rekurrenz:

$$T(n) = \begin{cases} 1 & \text{falls } n < 3, \\ T(n-1) + 2 \cdot T(n-2) + n & \text{falls } n \geq 3. \end{cases}$$

Finden Sie eine Funktion f, sodass $T(n) \in \mathcal{O}(f(n))$ gilt und beweisen Sie Ihre Behauptung.

Musterlösung:

(a) Bei Zahlen, die sich wie oben beschrieben schreiben lassen, landet man bei wiederholtem Teilen durch 4 irgendwann entweder bei 1, 2 oder 3. Also müssen drei Induktionsanfänge gezeigt werden:
Induktionsanfang $n = 1$**:** $T(1) = 2 \leq 2 \cdot 1^2 + 2 \cdot 1 = 4$
Induktionsanfang $n = 2$**:** $T(2) = 2 \leq 2 \cdot 2^2 + 2 \cdot 2 = 12$
Induktionsanfang $n = 3$**:** $T(3) = 2 \leq 2 \cdot 3^2 + 2 \cdot 3 = 24$
Induktionsvoraussetzung: $T(n) \leq 2n^2 + 2n$ gilt für ein n (mit $n = 4^k$, $n = 2 \cdot 4^k$ oder $n = 3 \cdot 4^k$)
Induktionsschluss $n \rightsquigarrow 4n$**:**

$$\begin{aligned} T(4n) &= 6 \cdot T\left(\frac{4n}{4}\right) + (4n)^2 \\ &\overset{\text{IV}}{\leq} 6 \cdot (2n^2 + 2n) + (4n)^2 \\ &= 12n^2 + 12n + (4n)^2 \\ &= 12n^2 + 3 \cdot (4n) + (4n)^2 \\ &\leq 12n^2 + 4n^2 + 2 \cdot (4n) + (4n)^2 \\ &= 16n^2 + (4n)^2 + 2 \cdot (4n) \\ &= 2 \cdot (4n)^2 + 2 \cdot (4n) \end{aligned}$$

□

(b) Es gilt: $T(n) \in \mathcal{O}(n!)$. Wir beweisen hierzu induktiv, dass für alle $n \in \mathbb{N}$ gilt, dass $T(n) \leq n!$:
Induktionsanfang $n < 3$**:** $T(1) = 1 = 1!$ und $T(2) = 1 < 2 = 2!$.
Induktionsvoraussetzung: Für ein $n \in \mathbb{N}$, $n \geq 2$, gilt $T(k) \leq k! \quad \forall k \leq n$.
Induktionsschluss $n \rightsquigarrow n+1$**:**

$$\begin{aligned} T(n+1) &= T((n+1)-1) + 2 \cdot T((n+1)-2) + (n+1) \\ &= T(n) + 2 \cdot T(n-1) + (n+1) \\ &\overset{IV}{\leq} n! + 2 \cdot (n-1)! + (n+1) \\ &\overset{n\geq 2}{\leq} n! + n \cdot (n-1)! + n! \\ &= 3 \cdot n! \\ &\overset{n\geq 2}{\leq} (n+1) \cdot n! \\ &= (n+1)! \end{aligned}$$

Aufgabe 2 (*Master-Theorem, 4 Punkte*)

Zeigen Sie mit Hilfe der gerundeten Version des Master-Theorems (siehe unten) scharfe asymptotische Schranken für folgende Rekurrenzen:

a) $A(1) = 1$ und für $n \in \mathbb{N}$: $A(n) = 49 \cdot A(\lceil n/7 \rceil) + n$

b) $B(1) = 3$ und für $n \in \mathbb{N}$: $B(n) = 3 \cdot B(\lceil n/3 \rceil) + 3n + 3\sqrt{n} + 3$

c) $C(1) = 23$ und für $n \in \mathbb{N}$: $C(n) = C(\lceil n/2 \rceil) + 3n$

d) $D(1) = 3$ und für $n \in \mathbb{N}$: $D(n) = 2 \cdot D(\lceil n/4 \rceil) + C(n)$

Die gerundete Version des Master-Theorems: Für positive Konstanten a, b, c, d und $n \in \mathbb{N}$ sei

$$T(n) = \begin{cases} a & \text{falls } n = 1 \text{ Basisfall} \\ d \cdot T(\lceil n/b \rceil) + c \cdot n & \text{falls } n > 1 \text{ teile und herrsche.} \end{cases}$$

Dann gilt

$$T(n) \in \begin{cases} \Theta(n) & \text{falls } d < b \\ \Theta(n \log n) & \text{falls } d = b \\ \Theta\left(n^{\log_b d}\right) & \text{falls } d > b. \end{cases}$$

Musterlösung:

a) $a = 1$, $b = 7$, $c = 1$, $d = 49 \overset{d>b}{\Rightarrow} A(n) \in \Theta\left(n^{\log_7 49}\right) = \Theta(n^2)$

b) Die gegebene Rekurrenz B passt aufgrund des Summanden „$3n + 3\sqrt{n} + 3$“ nicht genau in das Schema des Master-Theorems. Um eine scharfe Schranke zu bestimmen, betrachten wir nun zwei weitere Rekurrenzen B_1 und B_2, sodass für alle $n \in \mathbb{N}$ gilt:

$$B_1(n) \leq B(n) \leq B_2(n).$$

Dazu halten wir zuerst fest, dass für $n \geq 1$ gilt: $3n \leq 3n + 3\sqrt{n} + 3 \leq 9n$.
Definiere nun $B_1(1) = 3$, $B_1(n) = 3 \cdot B_1(n/3) + 3n$ und $B_2(1) = 3$, $B_2(n) = 3 \cdot B_2(n/3) + 9n$. Dann gilt $B_1(n) \leq B(n) \leq B_2(n)$. Immer gilt $a = 3$, $b = 3$, $d = 3$. Nach dem Master-Theorem gilt nun $B_1 \in \Theta(n \log n)$ und $B_2 \in \Theta(n \log n)$, da $b = 3 = d$. Insgesamt ergibt sich damit $B(n) \in \Theta(n \log n)$, da $B_1(n) \leq B(n) \leq B_2(n)$.

c) $a = 23,\ b = 2,\ c = 3,\ d = 1 \overset{d<b}{\Rightarrow} C(n) \in \Theta(n)$

d) Da $C(n) \in \Theta(n)$ existieren $c_1, c_2, n_0 \in \mathbb{N}$, sodass $c_1 n \leq C(n) \leq c_2 n$ für alle $n \geq n_0$. Definiere damit $D_1(1) = 3, D_1(n) = 2 \cdot D_1(n/4) + c_1 n$ und $D_2(1) = 3, D_2(n) = 2 \cdot D_2(n/4) + c_2 n$. Dann gilt $D_1(n) \leq D(n) \leq D_2(n)$ für alle $n \geq n_0$. Immer gilt $a = 3, b = 4, d = 2$. Da $d < b$ folgt $D_1(n) \in \Theta(n)$ und $D_2(n) \in \Theta(n)$. Daraus folgt $D(n) \in \Theta(n)$.

Aufgabe 3 (*Algorithmenanalyse & -entwurf, 6 Punkte*)

Betrachten Sie den folgenden Algorithmus:

```
Function func(a : ℕ; A : Array [0, ..., n − 1] of ℕ; start : ℕ0; end : ℕ0) : ℕ0 ∪ {−1}
    assert start ≤ end
    if start = end do
        if A[start] = a do
            return start
        else
            return −1
    else
        middle = ⌈(start + end + 1)/2⌉
        i := func(a, A, start, middle − 1)
        j := func(a, A, middle, end)
        if i ≠ −1 do
            return i
        else
            return j
```

Im Folgenden bezeichne a immer ein Element in $\mathbb{N}$, A ein Array mit Elementen aus $\mathbb{N}$ und n die Größe von A. Sie können für die gesamte Aufgabe vereinfachend davon ausgehen, dass n von der Form $n = 2^k$ mit $k \in \mathbb{N}$ ist.

a) Was berechnet der Algorithmus func bei Aufruf von func$(a, A, 0, n-1)$?

b) Zeigen Sie: Die Laufzeit des Algorithmus func bei Aufruf von func$(a, A, 0, n-1)$ liegt in $\mathcal{O}(n)$.

c) Angenommen, dass Array A ist aufsteigend sortiert. Geben Sie einen Algorithmus func_2 an, der für diesen Fall das gleiche berechnet, wie func$(a, A, 0, n-1)$, aber Laufzeit $\mathcal{O}(\log n)$ hat. Beweisen Sie, dass ihr Algorithmus diese Laufzeit erreicht.

Musterlösung:

a) Der Algorithmus sucht nach dem Element a im Array A und gibt den kleinsten Index i aus, sodass $A[i] = a$ gilt. Enthält A das Element a nicht, so gibt der Algorithmus -1 aus.

b) Da der Algorithmus rekursiv ist, stellen wir eine Rekurrenz auf, um seine Laufzeit zu untersuchen. Zuerst stellen wir fest, dass durch die Veränderungen von start und end bestimmt wird, welcher Bereich vom Array durchsucht wird.

Der Basisfall ist dann erreicht, wenn start = end gilt. Hier wird ein einelementiges Array untersucht und es werden nur zwei einfache Vergleiche ausgeführt, sodass wir die Kosten mit 2 abschätzen können.

Ansonsten ruft sich der Algorithmus zweimal selbst auf, wobei dabei das Array „halbiert“ wird, indem start und end angepasst werden. Die „Teilarrays“ haben zwar nicht exakt die gleiche Größe, wir können sie aber jeweils mit $n/2$ abschätzen (da wir nur von n der Form 2^k ausgehen, müssen

wir hier nicht Runden). Nach den rekursiven Aufrufen folgt ein einfacher Vergleich, dessen Kosten wir mit 1 abschätzen können.

Wir erhalten damit die Rekurrenz

$$T(n) = \begin{cases} 2 & \text{falls } n = 1 \\ 2 \cdot T(n/2) + 1 & \text{sonst.} \end{cases}$$

Auf diese Rekurrenz ist das Master-Theorem (in der Variante aus der Vorlesung) nicht anwendbar. Daher beweisen wir induktiv $T(n) \leq 3 \cdot n - 1$ für alle n mit $n = 2^k$, $k \in \mathbb{N}$.

Induktionsanfang $n = 2$**:** $T(2) = 2 \cdot T(1) + 1 = 5 = 3 \cdot 2 - 1$

Induktionsvorraussetzung: $T(n) \leq 3 \cdot n - 1$ gilt für ein n (mit $n = 2^k$)

Induktionsschritt $n \rightsquigarrow 2n$**:**

$$\begin{aligned} T(2n) &= 2 \cdot T(2n/2) + 1 \\ &= 2 \cdot T(n) + 1 \\ &\overset{IV}{\leq} 2 \cdot (3n - 1) + 1 \\ &= 3 \cdot (2n) - 1 \end{aligned}$$

c) Wir können uns zunutze machen, dass das Array aufsteigend sortiert ist. Wenn wir das „mittlere“ Element $A[\text{middle}]$ betrachten und beispielsweise gilt, dass $a \leq A[\text{middle}]$, muss der Algorithmus das Array nur noch „links von“ $A[\text{middle}]$ durchsuchen und kann die „rechte Hälfte“ komplett ignorieren. Analog muss er bei $a \geq A[\text{middle}]$ nur die rechte Hälfte betrachten. Damit ergibt sich der bekannte und wichtige Algorithmus der **binären Suche**:

```
Function binarySearch(a : ℕ; A : Array [0, ..., n − 1] of ℕ; start : ℕ0; end : ℕ0) : ℕ0 ∪ {−1}
    assert start ≤ end
    assert A is in ascending order
    if start = end do
        if A[start] = a do
            return start
        else
            return −1
    else
        middle = ⌊(start + end)/2⌋
        if a ≤ A[middle] do
            return binarySearch(a, A, start, middle)
        else
            return binarySearch(a, A, middle + 1, end)
```

Für diesen Algorithmus ergibt sich die Rekurrenz

$$T(n) = \begin{cases} 2 & \text{falls } n = 1 \\ T(n/2) + 1 & \text{sonst.} \end{cases}$$

Auch auf diese Rekurrenz ist das Master-Theorem (in der Variante aus der Vorlesung) nicht anwendbar. Daher beweisen wir induktiv $T(n) \leq \log_2(n) + 2$ für alle n mit $n = 2^k$, $k \in \mathbb{N}$.

Induktionsanfang $n = 2$**:** $T(2) = T(1) + 1 = 2 + 1 = 2 + \log_2(2)$

Induktionsvorraussetzung: $T(n) \leq \log_2(n) + 2$ gilt für ein n (mit $n = 2^k$)

Induktionsschritt $n \rightsquigarrow 2n$:

$$\begin{aligned} T(2n) &= T(2n/2) + 1 \\ &= T(n) + 1 \\ &\overset{IV}{\leq} (\log_2(n) + 2) + 1 \\ &= \log_2(n) + 2 + \log_2(2) \\ &= \log_2(2n) + 2 \end{aligned}$$

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

Betrachten Sie den folgenden Algorithmus:

```
Function sieve(n : ℕ \ {1}) : Array [2,...,n] of Boolean
    isPrime : Array [2,...,n] of Boolean
    for i=2 to n do
        isPrime[i] := true

    for i=2 to n do
        invariant ???
        if isPrime[i] = true do
            step := i
            while i · step ≤ n do
                isPrime[i·step] := false
                step := step + 1
    return isPrime
```

a) Was berechnet der Algorithmus sieve?

b) Geben Sie eine geeignete Schleifeninvariante für die im Algorithmus mit „???“ markierte Stelle an und beweisen Sie diese!

Musterlösung:

a) Der Algorithmus erstellt ein Array isPrime für alle Zahlen 2 bis n, sodass isPrime[i] genau dann gleich true ist, wenn die Zahl i eine Primzahl ist.

b) Eine geeignete Invariante ist „$\forall i$ mit $2 \leq i \leq n$: A[i] = false $\Rightarrow$ i ist nicht prim“.

Bevor wir die Invariante beweisen, halten wir folgenden Fakt fest: Eine Zahl $i \in \mathbb{N}$ ist genau dann eine Primzahl, wenn 1 und i die einzigen Teiler von i sind. Umgekehrt ist i also genau dann *keine* Primzahl, wenn es zwei Zahlen $a, b \in \mathbb{N}$ mit $1 < a, b < i$ und $i = a \cdot b$ gibt.

Vor Beginn der Schleife ist die Invariante erfüllt, da noch kein Element des Arrays auf false gesetzt wurde. Die **for** -Schleife berechnet nun für jedes i die Werte $i \cdot i, i \cdot (i+1), i \cdot (i+2), ...$ und setzt die entsprechenden Einträge des Arrays auf false. Dies erhält die Schleifeninvariante, da $i > 1$ und diese Zahlen somit keine Primzahlen sind. Es genügt bei $i \cdot i$ zu beginnen, da alle „Kombinationen“ $i \cdot a$ mit $a < i$ bereits bei einem vorherigen Schleifendurchlauf betrachtet wurden (nämlich im Schleifendurchlauf mit $i = a$). Zusätzlich setzt die Schleife niemals isPrime[p] auf false, wenn p eine Primzahl ist, da keine Kombination i·step= p existiert und niemals für den Schleifenzähler i selbst isPrime[i] verändert wird. Da die **for** - und **while** -Schleifen jeweils bis $i = n$ laufen, wird jeder Eintrag des Arrays betrachtet. Somit ist die Schleifeninvariante auch am Ende der Schleife erfüllt.