Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dr. Jörn Müller-Quade
Björn Kaidel, Sascha Witt

2. Tutorenblatt zu Algorithmen I im SoSe 2017

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

Im Folgenden findet ihr einige unverbindliche Vorschläge zur Gestaltung des zweiten Tutoriums.

1. Master-Theorem, Lösen von Rekurrenzen

Wiederholung des Master-Theorems, gebt verschieden Rekurrenzen an, bei denen das Master-Theorem anwendbar ist und andere, die durch vollständige Induktion bewiesen werden müssen (entweder nur für Potenzen oder für alle natürlichen Zahlen). Beispiele:

Aufgabe 1 (*Master-Theorem*)

Einfache Aufgaben, bei denen das Master-Theorem direkt oder indirekt anwendbar ist (vergleiche Übung und Übungsblatt). Insbesondere auch Aufgaben, bei denen die Rekurrenz nicht direkt in das Schema des Theorems passt und es „geschaltet“ angewandt werden muss. Siehe dazu auch das zweite Übungsblatt aus dem letzten Jahr.

Aufgabe 2 (*Master-Theorem*)

Die Laufzeit eines Algorithmus $\mathcal{A}$ wird beschrieben durch die Rekurrenzgleichung $T(1) = 1$ und $T(n) = 7 \cdot T(\lceil n/2 \rceil) + n$ für alle anderen $n \in \mathbb{N}$. Ein weiterer Algorithmus $\mathcal{B}$ hat Laufzeit $T'(1) = 1$ und $T'(n) = a \cdot T'(\lceil n/4 \rceil) + 5 \cdot n$ sonst. Was ist der größte Wert $a \in \mathbb{N}$ sodass $\mathcal{B}$ asymptotisch schneller als $\mathcal{A}$ ist?

Aufgabe 3 (*Vollständige Induktion*)

Gegeben sei folgende Rekurrenz für alle $n = 4^k$, wobei $k \in \mathbb{N}_0$:

$$T(n) = \begin{cases} 2 & \text{falls } n = 1, \\ 2T(n/4) & \text{falls } n > 1. \end{cases}$$

Finden Sie eine Funktion $f\colon \mathbb{N} \to \mathbb{R}^+$ und Konstanten $c_1, c_2 \in \mathbb{R}^+$, so dass $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ für alle $n = 4^k$ (wobei $k \in \mathbb{N}_0$) gilt und beweisen Sie Ihre Behauptung. (Es gilt sogar $T(n) = 2\sqrt{n}$.)

2. Weitere Aufgaben

Aufgabe 4 (*Schleifeninvariante, Korrektheit, Aufwandsanalyse*)

Das Merging-Problem ist folgendermaßen definiert:

Gegeben: zwei aufsteigend sortierte Arrays $A[1..n_1], B[1..n_2]$ von natürlichen Zahlen
Gesucht: das aufsteigend sortierte Array $C[1..(n_1 + n_2) =: n]$ von natürlichen Zahlen, das genau die Zahlen von A und B enthält

Der folgende Algorithmus löst das Problem:

1: **function** *merge*(A : Array $[1..n_1]$ of $\mathbb{N}_{\geq 0}$, B : Array $[1..n_2]$ of $\mathbb{N}_{\geq 0}$)
2: **assert** $A[i] \leq A[j] \quad \forall i \leq j$ mit $i, j \in \{1, ..., n_1\}$
3: **assert** $B[i] \leq B[j] \quad \forall i \leq j$ mit $i, j \in \{1, ..., n_2\}$
4: $A[n_1 + 1] := \infty$, $B[n_2 + 1] := \infty$
5: $n := n_1 + n_2$
6: $j_A := 1$, $j_B := 1$;

7: **for** $i := 1$ **to** n **do**
8: $C[i] = \min(A[j_A], B[j_B])$
9: **if** $A[j_A] < B[j_B]$ **then**
10: $j_A = j_A + 1$
11: **else**
12: $j_B = j_B + 1$
13: **invariant** $C[1..i]$ enthält genau $A[1..j_A - 1], B[1..j_B - 1]$
14: **invariant** $B[k] \leq A[j_A] \quad \forall k \in \{1..j_B - 1\}$, $A[k] \leq B[j_B] \quad \forall k \in \{1..j_A - 1\}$
15: **invariant** $C[1..i]$ ist sortiert
16: **assert** $j_A = n_1 + 1, j_B = n_2 + 1$
17: **assert** $C[i] \leq C[j] \quad \forall i \leq j,\ i, j \in \{1, ..., n\}$
18: **assert** $C[1..n]$ enthält genau $A[1..n_1], B[1..n_2]$
19: **return** C

Beweisen Sie die Korrektheit des vorgegebenen Algorithmus, in dem Sie die vorgebenen Invarianten und Assertions beweisen. Beweisen Sie außerdem, dass der vorgegebene Algorithmus linearen Zeitverbrauch hat.

Musterlösung:

a) **Laufzeit:** Jeder Aufruf der Schleife in Zeile 6 benötigt konstant viel Zeit. Da die Schleife von 1 bis n läuft ist daher die Laufzeit offensichtlich $\Theta(n)$.
Korrektheit: Wir zeigen zunächst per Induktion die Invarianten und leiten daraus dann die Korrektheit unseres Algorithmus ab.

- *Invariante in Zeile 13:*
Induktionsanfang $i = 1$: Es wird genau eins der Arrays ausgewählt. Danach wird vom ausgewählten Array der Pointer um eins erhöht. D.h. im Fall $A[j_A] < B[j_B]$ wird j_A zu 2 und j_B bleibt 1. Dann gilt Behauptung. Der andere Fall geht analog.
Induktionsvoraussetzung: $C[1..i-1]$ enthält genau $A[1..j_A - 1], B[1..j_B - 1]$
Induktionsschluss $i - 1 \rightsquigarrow i$: Im Fall $A[j_A] < B[j_B]$ wurde $A[j_A]$ hinzugefügt und j_A wurde um eins erhört. Also gilt die Behauptung. Der andere Fall gilt analog.
- *Invariante in Zeile 14:*
Induktionsanfang $i = 1$: zu dem Zeitpunkt befindet sich nur ein Elementi $C[1]$ im Array C. Wegen Zeile 8 ist $C[1] = \min(A[1], B[1])$. Im Fall $C[1] = A[1]$ wird j_A um eins erhöht und es gilt offensichtlich $A[1] \leq B[1]$. Die Behauptung gilt also. Der andere Fall geht analog.
Induktionsvoraussetzung: Die Invariante ist zum Zeitpunkt $i - 1$ erfüllt. D.h. $B[k] \leq A[j_A] \quad \forall k \in \{1..j_B - 1\}$, $A[k] \leq B[j_B] \quad \forall k \in \{1..j_A - 1\}$
Induktionsschluss $i - 1 \rightsquigarrow i$: Wir betrachten den Fall, dass in Zeile 8 $C[i] = A[j_A]$ gewählt wird. In diesem Fall ist $j_{A_{\text{neu}}} = j_{A_{\text{alt}}} + 1$ und $A[j_{A_{\text{alt}}}] \leq B[j_{B_{\text{alt}}}]$. Umstellen liefert $A[j_{A_{\text{neu}}} - 1] \leq B[j_{B_{\text{alt}}}]$. Insbesondere impliziert das zusammen mit der Induktionsvoraussetzung die Behauptung $A[k] \leq B[j_{B_{\text{alt}}}] \quad \forall k \in \{1..j_{A_{\text{neu}}} - 1\}$. Der andere Fall geht analog.
- *Invariante in Zeile 15:*
Induktionsanfang $i = 1$: zu dem Zeitpunkt befindet sich nur ein Element im Array C. Also ist C sortiert.
Induktionsvoraussetzung: Die Invariante ist zum Zeitpunkt $i - 1$ erfüllt. D.h. $C[1..i-1]$ ist sortiert.
Induktionsschluss $i - 1 \rightsquigarrow i$: Da die beiden Arrays A, B aufsteigend sortiert sind, ist $A[k] \leq A[j_A] \quad \forall k \leq j_A$ und $B[k] \leq B[j_B] \quad \forall k \leq j_B$. Mit den vorherigen Invarianten folgt dann, $C[k] \leq A[j_A], C[k] \leq B[j_B] \quad \forall k \leq i - 1$. Also folgt insbesondere $C[k] \leq C[i] = \min(A[j_A], B[j_B]) \quad \forall k \leq i - 1$. Also ist $C[1..i]$ aufsteigend sortiert.
- *Assertion in Zeile 16:* Nach Ausführung der Schleife wurde j_A n_1 mal erhöht und j_B wurde n_2 mal erhört. Da beide werde mit 1 initialisiert wurden gilt die Behauptung.

Insgesamt folgt nach Ausführung der Schleife: $C[1..n]$ enthält genau $A[1..n_1]$ und $B[1..n_2]$ sowie $C[1..n]$ ist sortiert. Unser Algorithmus arbeitet also korrekt.