Karlsruher Institut für Technologie

Algorithmen 1 – Sommersemester 2018
Zuverlässige Softwaresysteme im Kontext der Automobilindustrie
`http://verialg.iti.kit.edu`
Prof. Dr. Carsten Sinz – Markus Iser

Übungsblatt 2

Ausgabe: 02.05.2018 – 15:30
Abgabe: 09.05.2018 – 13:00

A Rekurrenzen

A.1 Obere Schranke beweisen (2 Punkte)

Zeigen Sie durch vollständige Induktion, dass $T(n) \leq 7n^2 - 6n$, falls n eine Zweierpotenz ist und $T(n)$ durch folgende Rekurrenzrelation beschrieben wird:

$$T(n) \leq \begin{cases} 1 & \text{falls } n = 1, \\ 4 \cdot T(\lceil \frac{n}{2} \rceil) + 6 \cdot n & \text{falls } n \geq 2. \end{cases}$$

Lösungsvorschlag:

Induktionsanfang $n = 1:\ T(1) \leq 1 \leq 7 \cdot 1^2 - 6 \cdot 1 = 1$

Induktionsvoraussetzung $T(n) \leq 7n^2 - 6n$ gelte für eine beliebige, aber feste Zweierpotenz n.

Induktionsschluss $n \rightsquigarrow 2n$:

$$T(2n) \leq 4 \cdot T(n) + 6 \cdot 2n \leq 4 \cdot (7n^2 - 6n) + 6 \cdot 2n = 28n^2 - 24n + 6 \cdot 2n = 28n^2 - 12n = 7((2n)^2) - 6(2n)$$

A.2 Asymptotische Schranken finden und beweisen (2 Punkte)

Seien c_0 und c_1 positive Konstanten und

$$T(n) = \begin{cases} c_0 n & \text{falls } n \leq n_0, \\ T(\lfloor \frac{n}{5} \rfloor) + T(\lfloor \frac{7}{10} n + 2.5 \rfloor) + c_1 n & \text{falls } n > n_0. \end{cases}$$

für geeignet gewähltes $n_0 > 35$. Finden Sie eine Funktion f (in geschlossener Form), so dass $T(n) = \Theta(f(n))$ gilt und beweisen Sie ihre Behauptung.

Lösungsvorschlag:

Wähle z.B. $f(n) := n$. Offensichtlich ist $T(n) \geq \min(c_0, c_1)n$, also $T(n) \in \Omega(n)$. Es bleibt zu zeigen, dass $T(n) \in \mathcal{O}(n)$. Hierzu zeigen wir induktiv, dass ein $c \in \mathbb{R}^+$ existiert, sodass $\forall n \in \mathbb{N} : T(n) \leq cn$.

Induktionsanfang $n \leq n_0$:

Dann ist $T(n) = c_0 n$. Wählt man $c \geq c_0$, so ist die Aussage wahr, da $T(n) = c_0 n \leq cn$.

Induktionsvoraussetzung Es sei $n \in \mathbb{N}$ beliebig, aber fest. Es gelte $\forall k < n : T(k) \leq ck$.

Induktionsschluss $n-1 \rightsquigarrow n,\ n > 35$

$$\begin{aligned} T(n) &= T(\underbrace{\lfloor \frac{n}{5} \rfloor}_{<n}) + T(\underbrace{\lfloor \frac{7}{10}n + 2.5 \rfloor}_{<n}) + c_1 n \\ &\overset{\text{IV}}{\leq} c\lfloor \frac{n}{5} \rfloor + c\lfloor \frac{7}{10}n + 2.5 \rfloor + c_1 n \\ &\leq c\frac{n}{5} + c\frac{7}{10}n + 2.5c + c_1 n \\ &= c\frac{9}{10}n + 2.5c + c_1 n \end{aligned}$$

Hier muss also neben $c \geq c_0$ noch c so gewählt werden können, dass $c\frac{9}{10}n + 2.5c + c_1 n \leq cn$ oder analog $c_1 n \leq c(\underbrace{\frac{n}{10} - 2.5}_{>0 \text{ nötig}})$ gilt. Daher musste $n_0 > 25$ gewählt werden.

Mit $n_0 > 25$ folgt also die Existenz des gesuchten $c \in \mathbb{R}^+$ mit $\forall n \in \mathbb{N} : T(n) \leq cn$. Hiermit ist offensichtlich $T(n) \in \mathcal{O}(n)$. Es gilt damit insgesamt $T(n) \in \Omega(n) \cap \mathcal{O}(n) = \Theta(n) = \Theta(f(n))$.

B Master-Theorem (2 Punkte)

Zeigen Sie asymptotische Schranken für die folgenden Rekurrenzen unter Anwendung des Mastertheorems.

$$T_1(n) = \begin{cases} 1 & \text{für } n = 1 \\ \sqrt{2}T_1(\frac{n}{2}) + c^2 n & \text{für } n = 2^k, k \in \mathbb{N} \end{cases} \tag{1}$$

$$T_2(n) = \begin{cases} 1 & \text{für } n = 1 \\ 4T_2(\frac{n}{4}) + 4n & \text{für } n = 4^k, k \in \mathbb{N} \end{cases} \tag{2}$$

$$T_3(n) = \begin{cases} 1 & \text{für } n = 1 \\ 8T_3(\frac{n}{2}) + n & \text{für } n = 2^k, k \in \mathbb{N} \end{cases} \tag{3}$$

$$T_4(n) = \begin{cases} 1 & \text{für } n = 1 \\ 3(T_4(\frac{n}{5}) + 2n + 1) + 3 & \text{für } n = 5^k, k \in \mathbb{N} \end{cases} \tag{4}$$

Lösungsvorschlag:
Master-Theorem (einfache Form): Für positive Konstanten a, b, c, d sei $n = b^k$ für ein $k \in \mathbb{N}$. Sei $T(n)$ eine Laufzeitfunktion der Form

$$T(n) = \begin{cases} a & \text{für } n = 1 \\ d \cdot T(n/b) + cn & \text{für } n > 1 \end{cases}$$

Dann gilt

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

1. $a = 1, b = 2, c_{\text{Def.}} = c^2_{\text{Aufg.}}, d = \sqrt{2} \overset{\text{d} \leq \text{b}}{\Rightarrow} T_1(n) \in \Theta(n)$

2. $a = 1, b = 4, c = 4, d = 4 \overset{d=b}{\Rightarrow} T_2(n) \in \Theta(n \log n)$
3. $a = 1, b = 2, c = 1, d = 8 \overset{d>b}{\Rightarrow} T_3(n) \in \Theta(n^{\log_2 8}) = \Theta(n^3)$
4. Nach Umformung: $T_4(n) = 3T_4(n/5) + 6n + 6$.
   Es gilt $\forall n \in \mathbb{N}^+ : 6n \leq 6n + 6 \leq 12n$.
   Definiere $T_4^<(1) := 1, T_4^<(n) := 3T_4^<(n/5) + 6n$ und $T_4^>(1) := 1, T_4^>(n) := 3T_4^>(n/5) + 12n$.
   Dann gilt $T_4^<(n) \leq T_4(n) \leq T_4^>$. Immer gilt $a = 1, b = 5, d = 3$.
   Mit $c = 6 \overset{d<b}{\Rightarrow} T_4^<(n) \in \Theta(n) \Rightarrow T_4 \in \Omega(n)$ und mit $c = 12 \overset{d<b}{\Rightarrow} T_4^>(n) \in \Theta(n) \Rightarrow T_4 \in \mathcal{O}(n)$ und zusammen $T_4(n) \in \Theta(n)$.

C Merge-Algorithmus (4 Punkte)

Der Merge-Algorithmus verschmilzt zwei sortierte Zahlenreihen zu einer sortierten Zahlenreihe. Gegeben sind also zwei aufsteigend sortierte Arrays $A[1, \ldots, n_1]$ und $B[1, \ldots n_2]$ von natürlichen Zahlen, und gesucht ist das aufsteigend sortierte Array $C[1, \ldots, (n_1 + n_2)]$ von Zahlen in A und B.

Geben Sie einen Algorithmus an, der das Merging-Problem in $\Theta(n)$ löst (mit $n = n_1 + n_2$), und beweisen Sie die Laufzeit Ihres Algorithmus. Beweisen Sie außerdem die Korrektheit Ihres Algorithmus mit Hilfe von Invarianten. (Tipp: Versuchen Sie eine Schleifeninvariante über eine sortierte Teilliste zu beweisen.)

Lösungsvorschlag:

Algorithm 1: Merge

Data: A : Array $[1 \ldots n_1]$ of $\mathbb{N}_{\geq 0}$, B : Array $[1 \ldots n_2]$ of $\mathbb{N}_{\geq 0}$
Result: Sortierte Liste C : Array $[1 \ldots n_1 + n_2]$ von Zahlen in A und B

```
1  Precondition: A[i] ≤ A[j]    ∀i ≤ j mit i, j ∈ {1, ..., n_1}
2  Precondition: B[i] ≤ B[j]    ∀i ≤ j mit i, j ∈ {1, ..., n_2}
3  A[n_1 + 1] ← ∞
4  B[n_2 + 1] ← ∞
5  n ← n_1 + n_2
6  j_A ← 1
7  j_B ← 1
8  for i ← 1 ... n do
9  |  C[i] = min(A[j_A], B[j_B])
10 |  if A[j_A] < B[j_B] then
11 |  |  j_A = j_A + 1
12 |  else
13 |  |  j_B = j_B + 1
14 |  Invariant: C[1 ... i] enthält genau A[1 ... j_A − 1], B[1 ... j_B − 1]
15 |  Invariant: B[k] ≤ A[j_A]    ∀k ∈ {1 ... j_B − 1}
16 |  Invariant: A[k] ≤ B[j_B]    ∀k ∈ {1 ... j_A − 1}
17 |  Invariant: C[1 ... i] ist sortiert
18 Assert: j_A = n_1 + 1 ∧ j_B = n_2 + 1
19 Postcondition: C[i] ≤ C[j], ∀i ≤ j, i, j ∈ {1, ..., n}
20 Postcondition: C[1 ... n] enthält genau A[1 ... n_1], B[1 ... n_2]
21 return C
```

Korrektheit von Merge:

Laufzeit: Jeder Aufruf der Schleife in Zeile 8 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 14:***
  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: $C[1 \dots i-1]$ enthält genau $A[1 \dots j_A - 1]$, $B[1 \dots j_B - 1]$
  Induktionsschluss: $i-1 \rightsquigarrow i$: Im Fall $A[j_A] < B[j_B]$ wurde $A[j_A]$ hinzugefugt und j_A wurde um eins erhört. Also gilt die Behauptung. Der andere Fall gilt analog.

- ***Invariante in Zeile 15 und 16:***
  Induktionsanfang: $i = 1$: zu dem Zeitpunkt befindet sich nur ein Elementi C[1] im Array C. Wegen Zeile 9 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 $i-1$ erfullt. D.h. $B[k] \leq A[j_A]$ $\forall k \in 1 \dots j_B - 1$, $A[k] \leq B[jB]$ $\forall k \in 1 \dots j_A - 1$
  Induktionsschluss: $i-1 \rightsquigarrow i$: Wir betrachten den Fall, dass in Zeile 9 $C[i] = A[j_A]$ gewählt wird. In diesem Fall ist $j_{A_{neu}} = j_{A_{alt}} + 1$ und $A[j_{A_{alt}}] \leq B[j_{B_{alt}}]$. Umstellen liefert $A[j_{A_{neu}} - 1]$ $\leq B[j_{B_{alt}}]$. Insbesondere impliziert das zusammen mit der Induktionsvoraussetzung die Behauptung $A[k] \leq B[j_{B_{alt}}]$ $\forall k \in 1 \dots j_{A_{neu}} - 1$. Der andere Fall geht analog.

- ***Invariante in Zeile 17:***
  Induktionsanfang: $i = 1$: zu dem Zeitpunkt befindet sich nur ein Element im Array C. Also ist C sortiert.
  Induktionsvoraussetzung: Die Invariante ist zum Zeitpunkt $i-1$ erfullt. D.h. $C[1 \dots i-1]$ ist sortiert.
  Induktionsschluss: $i-1 \rightsquigarrow i$: Da die beiden Arrays A, B aufsteigend sortiert sind, ist $A[k]$ $\leq A[j_A]$ $\forall k \leq j_A$ und $B[k] \leq B[j_B]$ $\forall k \leq j_B$. Mit den vorherigen Invarianten folgt dann, $C[k] \leq A[j_A]$, $C[k] \leq B[j_B]$ $\forall k \leq i-1$. Also folgt insbesondere $C[k] \leq C[i] = \min(A[j_A], B[j_B])$ $\forall k \leq i-1$. Also ist $C[1 \dots i]$ aufsteigend sortiert.

- ***Assertion in Zeile 18:***
  Nach Ausführung der Schleife wurde j_A n_1 mal erhöht und j_B n_2 mal. Da beide Werte mit 1 initalisiert wurden gilt die Behauptung.

Insgesamt folgt nach Ausführung der Schleife $C[1 \dots n]$ ist sortiert. Unser Algorithmus arbeitet also korrekt. □