DAP2 – Präsenzübung 3

Besprechung: 10.05.2017 — 12.05.2017

Präsenzaufgabe 3.1: (Rekursionsgleichung)

Gegeben sei die Rekursionsgleichung:

$$T(n) = \begin{cases} 4 \cdot T(n/2) + n^3 & \text{wenn } n > 1 \\ 1 & \text{sonst} \end{cases}$$

a) Bestimmen Sie eine asymptotische obere Schranke für $T(n)$ und beweisen Sie sie mittels Induktion. Sie dürfen annehmen, dass n von der Form 2^k für ein $k \in \mathbb{N}$ ist.

b) Begründen Sie, warum die Annahme über n keine Beschränkung der Allgemeinheit darstellt.

Lösung:

a) Aus dem Master-Theorem (aus der Vorlesung) würden wir haben, dass $f(n) = n^3$, $\gamma \cdot n^3 = 4 \cdot \left(\frac{n}{2}\right)^3 \Leftrightarrow \gamma = \frac{1}{2}$ ist, wovon folgt $T(n) \in O(f(n) = O(n^3)$. Da wir aber keine Konstante für diese Behauptung kennen, sollten wir genaueren Ausdrück herleiten.

Behauptung: $T(n) \leq 2 \cdot n^3 - n^2 = O(n^3)$ (wir nehmen an, dass n eine 2er Potenz ist).

$$\begin{aligned} T(n) &= 4 \cdot T\left(\frac{n}{2}\right) + n^3 \\ &= 4 \cdot \left(4 \cdot T\left(\frac{n}{2^2}\right) + \frac{n^3}{2^3}\right) + n^3 = 4^2 \cdot T\left(\frac{n}{2^2}\right) + \frac{n^3}{2} + n^3 \\ &= 4^3 \cdot T\left(\frac{n}{2^3}\right) + \frac{n^3}{2^2} + \frac{n^3}{2^1} + \frac{n^3}{2^0} \\ &= \ldots = 4^k \cdot T\left(\frac{n}{2^k}\right) + n^3 \sum_{i=0}^{k-1} \frac{1}{2^i} \end{aligned}$$

für alle k, so dass $n > 2^k$. Wenn $k = \log_2 n$, gilt weiter:

$$\begin{aligned} T(n) &= 4^{\log_2 n} \cdot T(1) + n^3 \sum_{i=0}^{\log_2 n - 1} \frac{1}{2^i} = n^2 \cdot 1 + n^3 \cdot \frac{1 - \frac{1}{2^{\log_2 n}}}{1 - \frac{1}{2}} \\ &= n^2 + 2n^3 \cdot \left(1 - \frac{1}{n}\right) = 2 \cdot n^3 - n^2 = O(n^3) \end{aligned}$$

Die bisherige Analyse ist eher ein intuitives, schlaues Erraten der Lösung. Die Behauptung ist mittels vollständiger Induktion zu beweisen:

Beweis:

(I.A.) Für $n = 1$ ist $T(n) = 1 \leq 2 \cdot 1^3 - 1^2 = 2 - 1 = 1$.

(I.V.) Die Behauptung gelte für ein beliebiges, aber festes $n_0 = 2^{k_0}$ (n_0 ist eine Zweierpotenz).

(I.S.) Betrachte $n = 2 \cdot n_0$ (die nächste 2er Potenz). Dann gilt

$$\begin{aligned} T(n) &= 4 \cdot T\left(\frac{2 \cdot n_0}{2}\right) + (2 \cdot n_0)^3 = 4 \cdot T(n_0) + 8 \cdot n_0^3 \\ &\overset{\text{(IV)}}{\leq} 4 \cdot \left(2 \cdot n_0^3 - n_0^2\right) + 8 \cdot n_0^3 = 16 \cdot n_0^3 - 4 \cdot n_0^2 \\ &= 2 \cdot (2 \cdot n_0)^3 - (2 \cdot n_0)^2 = 2 \cdot n^3 - n^2. \end{aligned}$$

Damit folgt die Behauptung für alle natürliche Zahlen n.

Anmerkung: der induktive Beweis könnte auch für die Aussage $T(n) \leq c \cdot n^2$, für alle $c \geq 2$, durchgeführt werden. Was man aber nicht beweisen kann, ist dass $T(n) \leq c' \cdot n^2$ für eine $c' < 2$ ist. Wenn man aus $T(n) \in O(n^3)$ behaupten würde, dass $T(n) \leq n^3$ ist, könnte man dies nicht beweisen, weil es falsch ist.

b) **Potenzenbehauptung**: Es bleibt noch zu begründen, warum die Annahmen über n (2er-Potenzen) keine Beschränkung der Allgemeinheit darstellen. Wir haben für diese Rekursionsgleichung gezeigt, dass

$$T(n) \leq c \cdot n^3$$

für eine Konstante $c > 0$ für alle Werte von n gilt, die Zweierpotenzen sind. Ein einfacher Induktionsbeweis (den wir hier nicht führen wollen) zeigt, dass $T(n)$ monoton wachsend ist. Damit folgt für n mit $2^k \leq n < 2^{k+1}$:

$$\begin{aligned} T(n) &\leq T\left(2^{k+1}\right) \\ &\leq c \cdot \left(2^{k+1}\right)^3 = c \cdot 2^3 \cdot 2^{3k} \\ &= 8 \cdot c \cdot \left(2^k\right)^3 \leq 8 \cdot c \cdot n^3, \end{aligned}$$

d.h., es existiert eine Konstante $c' > c$, nämlich $c' = 8 \cdot c$, so dass auch für alle n (zwischen zwei Potenzen $2^k \leq n < 2^{k+1}$) $T(n) \leq c' \cdot n^2$ gilt.

Wenn wir keine Behauptung über 2er-Potenzen machen würden, müssten wir entweder behaupten, dass die Teilung in der Rekursionsgleichung ($T(\frac{n}{2})$) ganzzählig ist, oder dass bei Teilung Ab- bzw. Aufrunden ($T(\lfloor \frac{n}{2} \rfloor)$ bzw. $T(\lceil \frac{n}{2} \rceil)$) gegeben ist.

□

Präsenzaufgabe 3.2: (Teile und Herrsche)

Gegeben sei ein Array $A[1..n]$ von positiven ganzen Zahlen. Wir wollen die größte Wertdifferenz $A[i] - A[j]$ für $i \leq j$ bestimmen. Z.B. im Array $\{1, 8, 4, 7\}$ ist die größte Wertdifferenz $8 - 4 = 4$.

a) Entwickeln Sie einen Teile-und-Herrsche Algorithmus, der die größte Wertdifferenz berechnet und zurückgibt, und beschreiben Sie ihn mit eigenen Worten. Geben Sie eine Implementierung Ihres Algorithmus in Pseudocode an. Für die volle Punktzahl wird ein Algorithmus erwartet, dessen Worst-Case-Laufzeit durch $\mathcal{O}(n)$ beschränkt ist.

b) Analysieren Sie die Laufzeit Ihres Algorithmus. Stellen Sie hierzu eine Rekursionsgleichung für die Laufzeit Ihres Algorithmus auf und lösen Sie diese. Hier dürfen Sie behaupten, dass n eine Zweierpotenz ist.

c) Zeigen Sie die Korrektheit Ihres Algorithmus.

Lösung:

a) Die größte Wertdifferenz muss mindestens 0 sein, da wir für $i = j$ also $A[i] - A[i] = 0$ haben. Somit gilt, dass wenn wir das Array der Länge 1 haben, können wir einfach als größte Wertdifferenz 0 zurückgeben. Ansonsten können wir das Array in zwei (gleichgrößen) Teilarrays teilen, und als größte Wertdifferenz kann eine der folgenden drei Fälle vorkommen:

- die größte Wertdifferenz $A[i] - A[j]$ ist im linken Teilarray,
- die größte Wertdifferenz $A[i] - A[j]$ ist im rechten Teilarray,
- die größte Wertdifferenz hat $A[i]$ im linken Teilarray und $A[j]$ im rechten Teilarray.

Deswegen müssten wir für beiden Teilarrays A_L und A_R vom Array A die Werte $\max_L$ (den maximalen Wert im A_L), $\min_L$ (den minimalen Wert im A_L) und w_L (die größte Wertdifferenz im A_L) berechnen und zurückgeben (und auch für A_R). Nun folgt es dass es gilt:

- $\max_A = \max\{\max_L, \max_R\}$,
- $\min_A = \min\{\min_L, \min_R\}$,
- $w_A = \max\{w_L, w_R, \max_L - \min_R\}$.

Für das Array A der Länge 1 gilt: $(\max_A, \min_A, w_A) = (A[1], A[1], 0)$.

Diese Beschreibung lässt sich einfach in den Pseudocode **MaxDif(Array A, int l, int r)** umwandeln, wobei **l** und **r** die Indizes der Grenzen des betrachteten Teilarrays sind, und das erste Aufruf mit **MaxDif(A,1,n)** erfolgt.

```
StartMaxDif(array A):
1. n ← Length(A)
2. if n > 0 then
3.     (m1, m2, w) ← MaxDif(A,1,n)
4.     return w
```

MaxDif(**array** A, **int** StartIndex, **int** EndIndex):

1. **if** EndIndex = StartIndex **then**
2. **return** $(A[\text{StartIndex}], A[\text{StartIndex}], 0)$
3. **else**
4. $k \leftarrow \lfloor(\text{EndIndex} + \text{StartIndex})/2\rfloor$ `/* Index des mittleren Elements */`
5. $(\max_L, \min_L, w_L) \leftarrow$ MaxDif(A, StartIndex, k)
6. $(\max_R, \min_R, w_R) \leftarrow$ MaxDif(A, $k+1$, EndIndex)
7. $\max_A \leftarrow \max\{\max_L, \max_R\}$
8. $\min_A \leftarrow \min\{\min_L, \min_R\}$
9. $w_a \leftarrow \max\{w_L, w_R, \max_R - \min_L\}$
10. **return** $(\max_A, \min_A, w_A)$

b) Wir können annehmen, dass n eine Zweierpotenz ist, und damit dass alle Teilarrays auch die Größe gleich einer Zweierpotenz haben. Die Rekursionsabbruchbehandlung (Zeilen 1 und 2) lässt sich in konstanter Zeit absolvieren. Ansonsten ruft man bei jedem Rekursionsaufruf zwei weitere Aufrufe mit halbierter Größe des Arrays, was man höchstens $\log_2 n$ mal tun kann. Die Restanweisungen und Rekursionsabbruchsbehandlung eine konstante Anzahl der Rechenschritte verlangt. Die Laufzeit des Algorithmus ist durch folgende Rekursionsgleichung gegeben

$$T(n) = \begin{cases} c_1 & \text{für } n = 1 \\ 2 \cdot T\left(\frac{n}{2}\right) + c_2 & \text{für } n > 1 \end{cases}$$

wobei c_1 und c_2 zwei positive Konstanten sind. Die Lösung davon ist (nach Master-Theorem mit $\gamma = 2 > 1$) $T(n) \in O(n)$, was die erwünschte Laufzeit des Algorithmus aus der Teilaufgabe a) ist.

c) **Behauptung:** Algorithmus **MaxDif** findet das maximale Element, das minimale Element und die größte Wertdifferenz korrekt bei Eingabe des Arrays der Länge n, $\forall n \in \mathbb{N}$.

(I.A.) Wenn $n = 1$ ist, dann gibt es nur ein Element in dem Array, das das maximale und minimale Element ist, und das die größte Wertdifferenz $A[1] - A[1] = 0$ ergibt. Das wird korrekt in den Zeilen 1 und 2 berechnet und zurückgegeben. Die Behauptung ist deswegen im Induktionsanfang korrekt.

(I.V.) Es gelte dass der Algorithmus **MaxDif** das maximale Element, das minimale Element und die größte Wertdifferenz korrekt findet, für alle Arrays deren Eingabelängen kleiner als beliebige aber feste $n_0 \in \mathbb{N}$, $n_0 > 1$.

(I.S.) Sei die Länge des Arrays A gleich n_0, mit $n_0 > 1$. Die Bedingung in der Zeile 1. ist falsch. Wir finden das mittleren Element mit Index $k \leftarrow \lfloor(\text{EndIndex+StartIndex})/2\rfloor$ in der Zeile 4. Jetzt rufen wir unseren Algorithmus **MaxDif** auf der ersten Hälfte des Arrays A_L und der zweiten Hälfte des Arrays A_R auf (in den Zeilen 5. und 6.). Da die Größe der Teilarrays A_L und A_R kleiner als n_0 ist, werden nach der Induktionsvoraussetzung die Werte von $\max_L$, $\min_L$, w_L, $\max_R$, $\min_R$ und w_R korrekt berechnet. Denn das maximale Element in A ist größeres aus $\max_L$ und $\max_R$ (korrekt in Zeile 7.); das minimale Element in A ist kleineres aus $\min_L$ und $\min_R$ (korrekt in Zeile 8.); und nach Vorüberlegungen muss die größte Wertdifferenz in A eine der w_L, w_R und $\max_L - \min_R$ sein (korrekt berechnet in Zeile 9). Somit werden die korrekte Werte in der Zeile 10. auch für das Array der Länge n_0 zurückgegeben.

Somit gilt die Behauptung für alle natürlichen Zahlen n, d.h. die Korrektheit des Algorithmus ist damit bewiesen.