Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel
Peter Widmayer
Thomas Tschager
Tobias Pröger

13. Oktober 2016

Datenstrukturen & Algorithmen Lösungen zu Blatt 3 FS 16

Lösung 3.1 *Asymptotische Laufzeit einschätzen I.*

a) Um die asymptotische Anzahl der Aufrufe von f zu bestimmen, schätzen wir zunächst die Anzahl der Aufrufe nach oben ab. Die äussere Schleife wird offenbar weniger als n Mal durchlaufen. Die innere Schleife hat genau i Durchläufe, und da i nie grösser als $n/3$ ist, wird sie höchstens $n/3$ Mal durchlaufen. Die Anzahl der Aufrufe von f beträgt also höchstens $n \cdot n/3 = n^2/3 \in \mathcal{O}(n^2)$.

Andererseits beobachten wir aber auch, dass die äussere Schleife mindestens $n/9$ Mal durchlaufen wird. Bei etwa der Hälfte der Durchläufe ist $i > n/6$, also wird die innere Schleife mindestens $n/6$ Mal durchlaufen. Die Gesamtanzahl der Aufrufe von f ist damit also mindestens $(n/18) \cdot (n/6) = n^2/108 \in \Omega(n^2)$.

Da die Anzahl der Aufrufe von f sowohl in $\mathcal{O}(n^2)$ als auch in $\Omega(n^2)$ liegt, folgt, dass f asymptotisch $\Theta(n^2)$ Mal aufgerufen wird.

b) Die äussere Schleife wird exakt n Mal durchlaufen. Die erste innere Schleife startet bei $j = 100$, dekrementiert j in jedem Schritt um 1 und terminiert genau dann, wenn $j^2 < 1$, also $j < 1$ ist. Die Schleife wird also lediglich konstant oft durchlaufen! Die zweite innere Schleife verdoppelt den Wert von k in jedem Schritt. Daher wird sie lediglich $\Theta(\log(n))$ Mal durchlaufen. Also wird f asymptotisch $\Theta(n \cdot (1 + \log(n))) = \Theta(n \log n)$ Mal aufgerufen.

c) Die äussere Schleife wird exakt n Mal durchlaufen. Zur Bestimmung der Laufzeit der mittleren Schleife beobachten sie, dass sie terminiert, wenn $j^2 > n$, also $j > \sqrt{n}$ ist. Daher wird sie lediglich $\Theta(\sqrt{n})$ Mal durchlaufen. Die innerste Schleife wird $n-1$ Mal durchlaufen. Also wird f asymptotisch $\Theta(n \cdot \sqrt{n} \cdot n) = \Theta(n^{5/2})$ Mal aufgerufen.

Lösung 3.2 *Asymptotische Laufzeit einschätzen II.*

a) Wir beobachten zunächst folgendes:

- Für $n = 0$ wird die Funktion g genau zweimal aufgerufen, also ist $c(0) = 2$.
- Für $n = 1$ wird die Funktion g genau einmal aufgerufen, also ist $c(1) = 1$.
- Für $n \geq 2$ gibt es in Schritt 9 einen Aufruf von g. Zusätzlich rufen wir die Hauptfunktion f aber auch noch einmal auf der Eingabe $n - 1$ auf, und noch zweimal auf der Eingabe $n-2$. Schritt 6 erzeugt $c(n-1)$ viele Aufrufe von g, die Schritte 7 und 8 erzeugen jeweils $c(n-2)$ viele Aufrufe von g. Damit ist also $c(n) = 1 + c(n-1) + 2 \cdot c(n-2)$.

Wir zeigen nun mittels verallgemeinerter Induktion über n, dass $c(n) \geq 2^{n-1}$ gilt. Da $c(n)$ zwei explizite Verankerungen hat, müssen wir hier *zwei* Induktionsanfänge benutzen. Analog müssen wir verallgemeinerte statt vollständige Induktion benutzen, da wir Induk-

tionshypothesen für $c(n-2)$ *und* $c(n-1)$ benötigen (die klassische vollständige Induktion erlaubt nur Hypothesen für *ein* n).

Induktionsanfang I ($n = 0$): Es gilt $c(0) = 2 \geq \frac{1}{2} = 2^{0-1}$.

Induktionsanfang II ($n = 1$): Es gilt $c(1) = 1 \geq 1 = 2^{1-1}$.

Induktionshypothese: Es gibt ein $n \in \mathbb{N}$, sodass *für alle* $0 \leq n' \leq n$ gilt, dass $c(n') \geq 2^{n'-1}$ ist.

Induktionsschritt ($n \to n+1$):

$$\begin{aligned} c(n+1) &= 1 + c(n) + 2 \cdot c(n-1) \\ &\overset{I.H.}{\geq} 1 + 2^{n-1} + 2 \cdot 2^{(n-1)-1} \\ &= 1 + 2^{n-1} + 2^{n-1} > 2 \cdot 2^{n-1} = 2^n = 2^{(n+1)-1}. \end{aligned}$$ ∎

b) Wir lösen die Rekursion auf, indem wir die berechneten Zwischenergebnisse in einem Array $Z[0..n]$ abspeichern. Dabei speichere $Z[i]$ den von $f(i)$ berechneten Wert (für $i = 0, \ldots, n$). Dies führt zu folgendem Codefragment:

Funktion $f'(n)$

1 Setze $Z[0] \leftarrow 2 \cdot g(0)$.
2 Setze $Z[1] \leftarrow g(1)$.
3 Für $i = 2, \ldots, n$:
4 Berechne $x \leftarrow Z[i-1] + 2 \cdot Z[i-2]$.
5 Setze $Z[i] \leftarrow g(x)$.
6 Gib $Z[n]$ zurück.

Wir beobachten, dass die Anzahl der Aufrufe von g nur noch $\mathcal{O}(n)$ beträgt.

Lösung 3.3 *Algorithmenentwurf: Divide-and-Conquer.*

a) Ein naiver Algorithmus würde für jedes $i = 1, \ldots, n$ das Objekt O_i mit allen anderen Objekten O_j, $j \in \{1, \ldots, n\} \backslash \{i\}$ vergleichen und so prüfen, ob es mindestens $\lfloor n/2 \rfloor$ weitere Objekte O_j mit $O_j = O_i$ gibt. Die Laufzeit dieses Verfahrens ist in $\mathcal{O}(n^2)$. Man könnte einwenden, dass es reicht, i bis $\lceil n/2 \rceil$ laufen zu lassen, was aber die quadratische Laufzeit im schlechtesten Fall nicht verbessern würde.

b) Mittels Divide-and-Conquer kommt man auf folgende Lösung: Wir teilen die Objekte $O_1, \ldots, O_n$ in zwei gleich grosse Mengen M_1 und M_2 auf. Nun beobachten wir: Wenn die Eingabe $O_1, \ldots, O_n$ ein Mehrheitselement hat, es also mehr als $n/2$ Mal vorkommt, dann muss genau dieses Element in mindestens einer der beiden Mengen mehr als $n/4$ Mal vorkommen (wäre es in beiden Mengen höchstens $n/4$ Mal enthalten, dann auch in $\{O_1, \ldots, O_n\}$ höchstens $n/2$ Mal). Daraus folgt, dass ein Mehrheitselement von $O_1, \ldots, O_n$ für mindestens eine der beiden Mengen M_1 und M_2 ebenfalls ein Mehrheitselement ist.

Wir bestimmen daher einfach rekursiv in beiden Mengen das Mehrheitselement, falls eines existiert. Dadurch erhalten wir keinen, einen oder zwei Kandidaten (d.h., mögliche Objekte) für das Mehrheitselement von $O_1, \ldots, O_n$. Für diese Kandidaten prüfen wir einzeln, ob sie das Mehrheitselement von $O_1, \ldots, O_n$ sind, indem wir über die Objekte O_i iterieren und dabei zählen, wie oft ein Kandidat vorkommt.

Analyse: Sei $T(n)$ die Laufzeit des Algorithmus für eine Eingabe mit n Objekten. Um zu prüfen, ob ein Kandidat ein Mehrheitselement ist, reichen $\mathcal{O}(n)$ viele Schritte. Also erhalten wir $T(n) = 2T(n/2) + a \cdot n$ und $T(1) = c$, wobei a und c Konstanten sind. Wie in der Vorlesung gesehen, hat diese Formel die Auflösung $c \cdot n + a \cdot \log(n) \cdot n$, also hat der Algorithmus Kosten $\Theta(n \log n)$.

c) Es ist möglich, das Mehrheitselement in Linearzeit (d.h. in Zeit $\mathcal{O}(n)$) zu finden. Dazu durchlaufen wir betrachten wir die Objekte $O_1, \ldots, O_n$ in dieser Reihenfolge und merken uns 1) einen aktuellen Kandidaten für das Mehrheitselement, und 2) einen Zähler. Anfangs setzen wir den Zähler auf 0 und den Kandidaten auf etwas beliebiges (er spielt keine Rolle, solange der Zähler 0 ist). Wir sehen uns nun die Objekte $O_1, \ldots, O_n$ nacheinander an. Hier unterscheiden wir drei Fälle:

1. Fall: Das aktuelle Objekt ist ungleich dem Kandidaten und der Zähler ist gleich 0: Dann setzen wir den Kandidaten auf das aktuell betrachtete Objekt und den Zähler auf 1.

2. Fall: Das aktuelle Objekt ist ungleich dem Kandidaten und der Zähler ist grösser als 0: Dann reduzieren wir den Zähler um 1.

3. Fall: Das aktuelle Objekt ist gleich dem Kandidaten: Dann erhöhen wir den Zähler um 1.

Nun zeigen wir mittels verallgemeinerter Induktion über die Anzahl der Objekte n die folgende Aussage: *Ist m das Mehrheitselement von $O_1, \ldots, O_n$, dann ist unser Kandidat am Ende m und unser Zähler grösser als 0.* Achtung: Andersherum gilt dies nicht!

Induktionsanfang ($n = 1$): Ist nur $n = 1$ Objekt gegeben, dann ist die Aussage trivialerweise wahr.

Induktionshypothese: Wir nehmen an, die Aussage stimmt für alle Eingaben, die aus höchstens n Objekten bestehen.

Induktionsschritt ($n \to n + 1$): Wir benutzen nun die Induktionshypothese, um zu zeigen, dass die Aussage auch für Eingaben mit $n + 1$ Objekten stimmt. Dazu unterscheiden wir zwei Fälle. Entweder der Zähler wird nach dem ersten Schritt nie wieder 0. Dann muss das erste Objekt O_1 öfter vorkommen als alle anderen zusammen. Da dieses Objekt in unserem Algorithmus von Anfang an der Kandidat war, ist die Aussage in diesem Fall korrekt. Andernfalls nehmen wir an, der Zähler wird nach genau k Schritten 0. Nach Induktionsannahme wissen wir also, dass kein Element die Mehrheit unter den ersten k Objekten hat. Falls es ein Mehrheitselement m gibt, so muss es folglich die Mehrheit unter den restlichen $n - k$ Objekten haben. Da unser Zähler nach k Schritten wieder im Anfangszustand ist, können wir die restlichen Objekte getrennt betrachten. Nach Induktionsannahme ist der Zähler nach dem Durchlaufen der letzten $n - k$ Objekte grösser als 0 und m ist dann unser Kandidat. ∎

Was hilft uns dies nun? Wir haben gesehen, wir einen möglichen Kandidaten für ein Mehrheitselement in Zeit $\mathcal{O}(n)$ finden können. Nun müssen wir nun noch einmal über alle Objekte O_i iterieren, um zu prüfen, ob dieser Kandidat tatsächlich das Mehrheitselement für $O_1, \ldots, O_n$ ist.