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

20. Oktober 2016

Datenstrukturen & Algorithmen Lösungen zu Blatt 4 HS 16

Lösung 4.1 *Verschiedenes.*

a) Hier gibt es zwei mögliche Lösungen, nämlich $x = 1511$, $y = 2609$ sowie $x = 926$, $y = 1115$.

b) Nach einer Iteration:

2	5	9	10	15	1	6	11	12	8	7	20
1	2	3	4	5	6	7	8	9	10	11	12

Nach zwei Iterationen:

1	2	5	9	10	15	6	11	12	8	7	20
1	2	3	4	5	6	7	8	9	10	11	12

c) Da wir annehmen dürfen, dass n eine Potenz von 3 ist, gilt $n = 3^k$ für ein $k \in \mathbb{N}$. Wir teleskopieren, um auf eine Formel für $T(n)$ zu kommen:

$$\begin{aligned}
T(n) &= 9 + 4T(n/3) \\
&= 9 + 4(9 + 4T(n/3^2)) \\
&= 9 + 4(9 + 4(9 + 4T(n/3^3))) \\
&= 9 + 4 \cdot 9 + 4^2 \cdot 9 + 4^3 \cdot T(n/3^3) \\
&= 9 \cdot (4^0 + 4^1 + 4^2) + 4^3 \cdot T(n/3^3) \\
&= \dots \\
&\stackrel{!}{=} 9 \cdot \sum_{i=0}^{k-1} 4^i + 4^k \cdot 6 = 9 \cdot \frac{4^k - 1}{4 - 1} + 4^k \cdot 6 = 9 \cdot 4^k - 3 = 9 \cdot 4^{\log_3(n)} - 3.
\end{aligned}$$

Wir beweisen nun unsere Annahme durch vollständige Induktion über n.

Induktionsanfang ($n = 1$): Für $n = 1$ ist $T(1) = 6 = 9 \cdot 4^{\log_3(1)} - 3$.

Induktionshypothese: Wir nehmen an, es gelte $T(n) = 9 \cdot 4^{\log_3(n)} - 3$.

Induktionsschritt ($n \to 3n$): Für $n > 1$ gilt

$$T(3n) = 9 + 4 \cdot T(n) \stackrel{\text{I.H.}}{=} 9 + 4 \cdot (9 \cdot 4^{\log_3(n)} - 3) = 9 + 9 \cdot 4^{\log_3(n)+1} - 12 = 9 \cdot 4^{\log_3(3n)} - 3.$$

Hinweis: Alternativ hätte man die Aussage natürlich auch durch vollständige Induktion über k beweisen können.

Lösung 4.2 *Algorithmenentwurf: Zahlensummen.*

a) Wir prüfen für jede mögliche Wahl von a (es gibt n Kandidaten), ob die Zahl $z - a$ im Array vorkommt (in diesem Fall hätten wir zwei Zahlen mit $a + b = z$ gefunden). Da A sortiert ist, können wir mittels binärer Suche in $\mathcal{O}(\log n)$ Schritten feststellen, ob $z-a$ in A vorkommt. Falls die Zahl vorkommt, haben wir eine Lösung gefunden, ansonsten probieren wir die nächste Wahl von a aus, usw. Damit ergibt sich ein Algorithmus mit Kosten in $\mathcal{O}(n \log n)$.

b) Eine Laufzeit von $\mathcal{O}(n)$ erreichen wir mit folgender Überlegung: Seien l, r die Indizes des linken bzw. rechten Endes des Arrays (initial: $l = 1$ und $r = n$). Falls $A[l]+A[r] = z$, geben wir $A[l]$ und $A[r]$ aus und beenden das Verfahren. Wenn nun $A[l]+A[r] > z$, dann gilt sicher $A[k]+A[r] > z$ für jedes k mit $l \leq k \leq r$ (es gilt $A[k] \geq A[l]$, da A sortiert ist). Folglich gibt es im Array von Position l bis Position r keine Zahl, die zu $A[r]$ zusammengezählt genau z ergibt. Es genügt also, im Array nur Elemente mit Index $\leq r-1$ zu berücksichtigen. Daher setzen wir in diesem Fall $r \leftarrow r - 1$ und wiederholen das Verfahren.

   Ist dagegen $A[l]+A[r] < z$, dann gilt sicher $A[l]+A[k] < z$ für jedes k mit $l \leq k \leq r$. Folglich gibt es im Array von Position l bis Position r keine Zahl, die zu $A[l]$ zusammengezählt genau z ergibt (es gilt $A[k] \leq A[r]$, da A sortiert ist). Also genügt es, im Array nur Elemente mit Index $\geq l + 1$ zu berücksichtigen. Daher setzen wir in diesem Fall $l \leftarrow l + 1$ und wiederholen das Verfahren.

   Wir brechen ab, falls entweder ein Paar gefunden wurde, oder falls $l = r$. Wir finden so immer ein Paar a, b mit $a + b = z$, falls ein solches existiert. Die Laufzeit von $\mathcal{O}(n)$ können wir beweisen, indem wir den Verlauf von $r-l$ betrachten: In jedem Schritt, in dem das Verfahren nicht abbricht, wird entweder r um eins erniedrigt oder l um eins erhöht (aber niemals beides gleichzeitig). Das heisst, $r - l$ wird in jedem Schritt eins kleiner. Ursprünglich ist $r - l = n - 1$, und somit terminiert das Verfahren nach höchstens $n - 1$ Schritten.