Aufgaben mit Lösung zur Vorlesung "*Algorithmen und Datenstrukturen*"

Aufgabe 1:

(a) Lösen Sie die Rekurrenz-Gleichung

$$a_{n+2} = a_n + 2$$

für die Anfangs-Bedingungen $a_0 = 2$ und $a_1 = 1$. (10 Punkte)

(b) Lösen Sie die Rekurrenz-Gleichung

$$a_{n+2} = 2 \cdot a_n - a_{n+1}$$

für die Anfangs-Bedingungen $a_0 = 0$ und $a_1 = 3$. (10 Punkte)

Lösung:

(a) Es handelt sich um eine lineare, inhomogene Rekurrenz-Gleichung der Ordnung 2. Die zugehörige homogene Rekurrenz-Gleichung lautet

$$a_{n+2} = a_n$$

Zur Lösung machen wir den Ansatz $a_n = \lambda^n$. Das führt auf die Gleichung

$$\lambda^{n+2} = \lambda^n.$$

Division durch λ^n liefert (nach Umstellen) die Gleichung

$$\lambda^2 - 1 = 0.$$

Wegen $\lambda^2 - 1 = (\lambda - 1) \cdot (\lambda + 1)$ ist diese Gleichung äquivalent zu

$$(\lambda - 1) \cdot (\lambda + 1) = 0$$

und daraus folgt

$$\lambda = 1 \vee \lambda = -1.$$

Um eine spezielle Lösung der inhomogenen Rekurrenz-Gleichung $a_{n+2} = a_n + 2$ zu erhalten, versuchen wir zunächst den Ansatz $a_n = c$. Dieser Ansatz führt auf die Gleichung

$$c = c + 2 \quad \text{aus der sofort die offensichtlich falsche Gleichung} \quad 0 = 2$$

folgen würde. Daher führt dieser Ansatz nicht zum Ziel und wir versuchen statt dessen den Ansatz

$$a_n = c \cdot n.$$

Damit erhalten wir

$$c \cdot (n + 2) = c \cdot n + 2.$$

Die Lösung dieser Gleichung ist offenbar $c = 1$. Damit lautet die allgemeine Lösung der Rekurrenz-Gleichung

$$a_n = \alpha \cdot 1^n + \beta \cdot (-1)^n + n.$$

Die Koeffizienten α und β bestimmen wir durch Einsetzen der Anfangsbedingungen. Das führt auf das Gleichungs-System

$$\left\{ \begin{array}{l} 2 = \alpha + \beta \\ 1 = \alpha - \beta + 1 \end{array} \right\} \quad \Leftrightarrow \quad \left\{ \begin{array}{l} 3 = 2 \cdot \alpha + 1 \\ 1 = 2 \cdot \beta - 1 \end{array} \right\} \quad \Leftrightarrow \quad \left\{ \begin{array}{l} 1 = \alpha \\ 1 = \beta \end{array} \right\}$$

Damit haben wir folgende Lösung der gegebenen Rekurrenz-Gleichung gefunden:

$$a_n = (-1)^n + n + 1.$$

(b) Wir machen den Ansatz $a_n = \lambda^n$ und erhalten die Gleichung

$$\lambda^{n+2} = 2 \cdot \lambda^n - \lambda^{n+1}.$$

Division durch λ^n und anschließendes Umstellen liefert

$$\lambda^2 + \lambda - 2 = 0 \quad \Leftrightarrow \quad (\lambda - 1) \cdot (\lambda + 2) = 0.$$

Damit gilt also

$$\lambda = 1 \vee \lambda = -2$$

und die allgemeine Lösung lautet

$$a_n = \alpha \cdot 1^n + \beta \cdot (-2)^n.$$

Wir bestimmen die Konstanten α und β durch Einsetzen der Anfangsbedingungen:

$$\left\{ \begin{array}{l} 0 = \alpha + \beta \\ 3 = \alpha - 2 \cdot \beta \end{array} \right\} \quad \Leftrightarrow \quad \left\{ \begin{array}{l} \alpha = -\beta \\ 3 = -3 \cdot \beta \end{array} \right\} \quad \Leftrightarrow \quad \left\{ \begin{array}{l} -1 = \beta \\ 1 = \alpha \end{array} \right\}$$

Damit erhalten wir die Lösung

$$a_n = 1 - (-2)^n.$$

Hinweis: Bei der Lösung der folgenden Aufgabe sind selbstverständlich die in der Vorlesung vorgestellten Algorithmen zu verwenden.

Aufgabe 2: Der AVL-Baum t sei durch den folgenden Term gegeben, wobei zur Vereinfachung auf die Angabe der Werte, die mit den Schlüsseln assoziiert sind, verzichtet wurde.

$$t = \mathit{node}(17, \mathit{node}(8, \mathit{node}(2, \mathit{nil}, \mathit{nil}), \mathit{node}(10, \mathit{nil}, \mathit{nil})), \mathit{node}(23, \mathit{nil}, \mathit{nil}))$$

(a) Fügen Sie in diesem Baum den Schlüssel `13` ein und geben Sie den resultierenden Baum an. (6 Punkte)

(b) Fügen Sie in dem Baum aus Teil (b) den Schlüssel `15` ein und geben Sie den resultierenden Baum an. (3 Punkte)

(c) Entfernen Sie den Schlüssel 2 aus dem unter Teil (b) berechneten Baum und geben Sie den resultierenden Baum an. (4 Punkte)

Lösung: Die folgende Lösung ist sehr ausführlich. In der Klausur reicht es aus, die entsprechenden Bäume anzugeben.

Das Einfügen und Löschen in einem AVL-Baum unterscheidet sich von dem Einfügen und Löschen in einem binären Baum durch die zusätzliche Anwendung der Funktion *restore*(). Diese Funktion ist durch die folgenden Gleichungen spezifiziert:

1. $\mathit{nil}.\mathit{restore}() = \mathit{nil}$,
2. $|l.\mathit{height}() - r.\mathit{height}()| \leq 1 \rightarrow \mathit{node}(k, v, l, r).\mathit{restore}() = \mathit{node}(k, v, l, r)$,
3. $$\begin{array}{ll} & l_1.\mathit{height}() = r_1.\mathit{height}() + 2 \\ \wedge & l_1 = \mathit{node}(k_2, v_2, l_2, r_2) \\ \wedge & l_2.\mathit{height}() \geq r_2.\mathit{height}() \\ \rightarrow & \mathit{node}(k_1, v_1, l_1, r_1).\mathit{restore}() = \mathit{node}\big(k_2, v_2, l_2, \mathit{node}(k_1, v_1, r_2, r_1)\big) \end{array}$$
4. $$\begin{array}{ll} & l_1.\mathit{height}() = r_1.\mathit{height}() + 2 \\ \wedge & l_1 = \mathit{node}(k_2, v_2, l_2, r_2) \\ \wedge & l_2.\mathit{height}() < r_2.\mathit{height}() \\ \wedge & r_2 = \mathit{node}(k_3, v_3, l_3, r_3) \\ \rightarrow & \mathit{node}(k_1, v_1, l_1, r_1).\mathit{restore}() = \mathit{node}\big(k_3, v_3, \mathit{node}(k_2, v_2, l_2, l_3), \mathit{node}(k_1, v_1, r_3, r_1)\big) \end{array}$$

5. $r_1.height() = l_1.height() + 2$
   $\wedge$ $r_1 = node(k_2, v_2, l_2, r_2)$
   $\wedge$ $r_2.height() \geq l_2.height()$
   $\rightarrow$ $node(k_1, v_1, l_1, r_1).restore() = node\big(k_2, v_2, node(k_1, v_1, l_1, l_2), r_2\big)$

6. $r_1.height() = l_1.height() + 2$
   $\wedge$ $r_1 = node(k_2, v_2, l_2, r_2)$
   $\wedge$ $r_2.height() < l_2.height()$
   $\wedge$ $l_2 = node(k_3, v_3, l_3, r_3)$
   $\rightarrow$ $restore\big(node(k_1, v_1, l_1, r_1)\big) = node\big(k_3, v_3, node(k_1, v_1, l_1, l_3), node(k_2, v_2, r_3, r_2)\big)$

Damit lautet die Lösung der Aufgaben:

(a) Zunächst fügen wir den Schlüssel 13 ein, ohne auf die Balancierungs-Bedingung zu achten. Wir erhalten den folgenden Term, wobei wir die Knoten noch mit ihren Höhen annotieren, um später die Balancierungs-Bedingung überprüfen zu können:

$$t = n(17, n(8, n(2, *, *) : 1, n(10, n(13, *, *) : 1, *) : 2) : 3, \\ n(23, *, *) : 1) : 4$$

Damit sehen wir, dass die Balancierungs-Bedingung an der Wurzel dieses Knotens verletzt ist, denn der linke Teilbaum hat eine Tiefe von drei, während der rechte Teilbaum eine Tiefe von 1 hat. Da der rechte Teilbaum des linken Teilbaums eine größere Tiefe hat als der linke Teilbaum, liegt die in Gleichung 4 beschriebene Situation vor. Im einzelnen gilt:

1. $k_1 = 17$,
2. $l_1 = n(8, n(2, *, *), n(10, n(13, *, *), *))$,
3. $k_2 = 8$,
4. $l_2 = n(2, *, *)$,
5. $r_2 = n(10, n(13, *, *), *))$,
6. $k_3 = 10$,
7. $l_3 = n(13, *, *)$,
8. $r_3 = *$,
9. $r_1 = n(23, *, *))$.

Damit erhalten wir den AVL-Baum

$$t = n(10, n(8, n(2, *, *), *), \\ n(17, n(13, *, *), n(23, *, *)))$$

(b) Fügen wir den Schlüssel 15 ein, so erhalten wir

$$t = n(10, n(8, n(2, *, *), *), \\ n(17, n(13, *, n(15, *, *)), n(23, *, *)))$$

Dies ist bereits ein AVL-Baum.

(c) Nachdem wir den Schlüssel 2 entfernt haben, hat der Baum die Form

$$t = n(10, n(8, *, *), \\ n(17, n(13, *, n(15, *, *)), n(23, *, *)))$$

Jetzt ist die Balancierungs-Bedingung an der Wurzel verletzt. Ein Aufruf von *restore*() liefert

$$t = n(13, n(10, n(8, *, *), *), \\ n(17, n(15, *, *), n(23, *, *))).$$

Aufgabe 3: Betrachten Sie das folgende Programm:

```
sum := procedure(n) {
    i := 0;
    s := 0;
    while (i <= n) {
        s := i + s;
        i := i + 1;
    }
    return s;
}
```

Die Funktion *sum* soll die folgende Spezifikation erfüllen:

$$sum(n) = \tfrac{1}{2} \cdot n \cdot (n+1)$$

(a) Weisen Sie mit Hilfe des Hoare-Kalküls nach, dass das Programm korrekt ist.

(b) Beweisen Sie mit Hilfe der Methode der symbolischen Programm-Ausführung nach, dass das Programm korrekt ist.

Lösung:

(a) Zunächst der Hoare-Kalkül:

1. Wir zeigen als erstes, dass die `while`-Schleife der Invariante

$$I := \big(s = \tfrac{1}{2} \cdot i \cdot (i-1)\big)$$

genügt. Für die erste Zuweisung in der Schleife gilt

$$\{I \wedge i \leq n\} \quad \texttt{s := s + i;} \quad \{(I \wedge i \leq n)[s \mapsto s - i]\}$$

Wir formen den Ausdruck $\big(I \wedge i \leq n\big)[s \mapsto s - i]$ um:

$$\begin{aligned}
\big(I \wedge i \leq n\big)[s \mapsto s - i] &\leftrightarrow \big(s = \frac{1}{2} \cdot i \cdot (i-1) \wedge i \leq n\big)[s \mapsto s - i] \\
&\leftrightarrow s - i = \frac{1}{2} \cdot i \cdot (i-1) \wedge i \leq n \\
&\leftrightarrow s = \frac{1}{2} \cdot i \cdot (i-1) + i \wedge i \leq n \\
&\leftrightarrow s = \frac{1}{2} \cdot i \cdot (i+1) \wedge i \leq n
\end{aligned}$$

Als nächstes betrachten wir die Zuweisung "`i := i + 1;`":

$$\{s = \tfrac{1}{2} \cdot i \cdot (i+1) \wedge i \leq n\} \quad \texttt{i := i + 1;} \quad \{\big(s = \tfrac{1}{2} \cdot i \cdot (i+1) \wedge i \leq n\big)[i \mapsto i-1]\}$$

Es gilt

$$\begin{aligned}
&\big(s = \frac{1}{2} \cdot i \cdot (i+1) \wedge i \leq n\big)[i \mapsto i-1] \\
\leftrightarrow \quad & s = \frac{1}{2} \cdot (i-1) \cdot i \wedge i - 1 \leq n \\
\leftrightarrow \quad & s = \frac{1}{2} \cdot i \cdot (i-1) \wedge i \leq n + 1
\end{aligned}$$

und damit haben wir die Invariante nachgewiesen.

2. Die Invariante ist zu Beginn der Schleife erfüllt, denn zu Beginn der Schleife gilt $s = 0$ und $i = 0$ und offenbar gilt

   $$i = 0 \wedge s = 0 \rightarrow s = \tfrac{1}{2} \cdot (i-1) \cdot i.$$

3. Nach Beendigung der Schleife gilt $i = n + 1$ und damit hat die Invariante die Form

   $$s = \tfrac{1}{2} \cdot \big((n+1)-1\big) \cdot (n+1) = \tfrac{1}{2} \cdot n \cdot (n+1).$$

   Das ist aber genau die Behauptung.

(b) Jetzt die symbolische Programm-Ausführung:

```
1  unsigned sum(unsigned n) {
2      unsigned i_0 = 0;
3      unsigned s_0 = 0;
4      while (i_0 <= n) {
5          s_{k+1} = i_k + s_k;
6          i_{k+1} = i_k + 1;
7      }
8      return s_K;
9  }
```

Wir zeigen nun, dass für s_k die folgende Invariante gilt:

$$s_k = \tfrac{1}{2} \cdot (i_k - 1) \cdot i_k.$$

I.A.: $k = 0$.
Es gilt $s_0 = 0$ und $i_0 = 0$ und damit folgt sofort

$$s_0 = \tfrac{1}{2} \cdot (i_0 - 1) \cdot i_0.$$

I.S.: $k \mapsto k + 1$
Offenbar gilt $i_{k+1} = i_k + 1$ und damit haben wir

$$\begin{aligned} s_{k+1} &= i_k + s_k \\ &\overset{IV}{=} i_k + \frac{1}{2} \cdot (i_k - 1) \cdot i_k \\ &= \frac{1}{2} \cdot i_k \cdot (i_k + 1) \\ &= \frac{1}{2} \cdot (i_{k+1} - 1) \cdot i_{k+1}. \end{aligned}$$

Die Schleife wird offenbar $n + 1$ mal durchlaufen und es gilt $i_K = n + 1$. Daraus folgt

$$s_K = s_{n+1} = \tfrac{1}{2} \cdot (i_{n+1} - 1) \cdot i_{n+1} = \tfrac{1}{2} \cdot n \cdot (n+1).$$

Aufgabe 4: Im Skript werden Gleichungen angegeben, die das Einfügen und Löschen in einem Heap beschreiben. In diesem Zusammenhang sollen Sie in dieser Aufgabe einige zusätzliche Methoden auf binären Bäumen durch bedingte Gleichungen spezifizieren.

(a) Spezifizieren Sie eine Methode *isHeap*, so dass für einen binären Baum $b \in \mathcal{B}$ der Ausdruck $b.\texttt{isHeap}()$ genau dann den Wert `true` hat, wenn b die *Heap-Bedingung* erfüllt. (10 Punkte)

(b) Implementieren Sie eine Methode *isBalanced*, so dass für einen binären Baum $b \in \mathcal{B}$ der Ausdruck $b.\texttt{isBalanced}()$ genau dann den Wert `true` hat, wenn b die *Balancierungs-Bedingung* für *Heaps* erfüllt. (5 Punkte)

Lösung:

(a) Um die Methode *isHeap* leicht spezifizieren zu können, definieren wir zunächst eine Hilfsfunktion

$$\mathit{isBigger} : \mathcal{B} \times \mathit{Key} \to \mathbb{B}.$$

Für einen binären Baum b und einen Schlüssel k soll $b.\mathit{isBigger}(k)$ genau dann gelten, wenn alle Schlüssel, die in b auftreten, größer-gleich k sind. Diese Funktion wird durch Gleichungen spezifiziert:

1. $\mathit{nil}.\mathit{isBigger}(k) = \texttt{true}$,
2. $\mathit{node}(k_1, v_1, l, r).\mathit{isBigger}(k) = \big(k_1 \geq k_1 \wedge l.\mathit{isBigger}(k) \wedge r.\mathit{isBigger}(k)\big)$.

Damit läßt sich jetzt die Methode *isHeap*() durch Gleichungen spezifizieren:

1. $\mathit{nil}.\mathit{isHeap}() = \texttt{true}$,
2. $\mathit{node}(k, v, l, r).\mathit{isHeap}() = \big(l.\mathit{isBigger}(k) \wedge r.\mathit{isBigger}(k) \wedge l.\mathit{isHeap}() \wedge r.\mathit{isHeap}()\big)$.

(b) Wir definieren die Methode *isBalanced*() induktiv.

1. $\mathit{nil}.\mathit{isBalanced}() = \texttt{true}$,
2. $\mathit{node}(k, v, l, r).\mathit{isBalanced}() =$
$$\big(|l.\mathit{count}() - r.\mathit{count}()| \leq 1 \wedge l.\mathit{isBalanced}() \wedge r.\mathit{isBalanced}()\big).$$

Hier haben wir eine Hilfsfunktion *count* benutzt, die wie folgt spezifiziert werden kann:

1. $\mathit{Nil}.\mathit{count}() = 0$
2. $\mathit{Node}(p, v, l, r).\mathit{count}() = 1 + l.\mathit{count}() + r.\mathit{count}()$.

Aufgabe 5: Es sei $f(n) := \left(\sum_{i=1}^{n} \frac{1}{i}\right) - \ln(n)$. Zeigen Sie $f(n) \in \mathcal{O}(1)$. (12 Punkte)

Hinweis: Zeigen Sie

$$0 \leq \left(\sum_{i=1}^{n} \frac{1}{i}\right) - \ln(n) \leq 1.$$

Lösung: Für alle $x \in [i-1, i]$ gilt:

$$\begin{aligned}
& x \leq i \\
\Rightarrow \quad & \frac{1}{x} \geq \frac{1}{i} \\
\Rightarrow \quad & \int_{i-1}^{i} \frac{1}{x} dx \geq \int_{i-1}^{i} \frac{1}{i} dx \\
\Rightarrow \quad & \int_{i-1}^{i} \frac{1}{x} dx \geq \frac{1}{i} \\
\Rightarrow \quad & \sum_{i=2}^{n} \int_{i-1}^{i} \frac{1}{x} dx \geq \sum_{i=2}^{n} \frac{1}{i} \\
\Rightarrow \quad & \int_{1}^{n} \frac{1}{x} dx \geq \sum_{i=2}^{n} \frac{1}{i} \\
\Rightarrow \quad & \ln(n) - \ln(1) \geq \sum_{i=2}^{n} \frac{1}{i} \\
\Rightarrow \quad & \ln(n) + 1 \geq \sum_{i=1}^{n} \frac{1}{i} \\
\Rightarrow \quad & \sum_{i=1}^{n} \frac{1}{i} - \ln(n) \leq 1
\end{aligned}$$

Analog gilt für alle $x \in [i-1, i]$:

$$\begin{aligned}
& x \geq i-1 \\
\Rightarrow \quad & \frac{1}{x} \leq \frac{1}{i-1} \\
\Rightarrow \quad & \int_{i-1}^{i} \frac{1}{x} dx \leq \int_{i-1}^{i} \frac{1}{i-1} dx \\
\Rightarrow \quad & \int_{i-1}^{i} \frac{1}{x} dx \leq \frac{1}{i-1} \\
\Rightarrow \quad & \sum_{i=2}^{n} \int_{i-1}^{i} \frac{1}{x} dx \leq \sum_{i=2}^{n} \frac{1}{i-1} \\
\Rightarrow \quad & \int_{1}^{n} \frac{1}{x} dx \leq \sum_{i=2}^{n} \frac{1}{i-1}
\end{aligned}$$

$$\Rightarrow \quad \ln(n) - \ln(1) \leq \sum_{i=2}^{n} \frac{1}{i-1}$$

$$\Rightarrow \quad \ln(n) \leq \sum_{i=1}^{n-1} \frac{1}{i}$$

$$\Rightarrow \quad 0 \leq \sum_{i=1}^{n-1} \frac{1}{i} - \ln(n)$$

$$\Rightarrow \quad 0 \leq \sum_{i=1}^{n} \frac{1}{i} - \ln(n)$$

Insgesamt haben wir damit die Ungleichungs-Kette

$$0 \leq \left(\sum_{i=1}^{n} \tfrac{1}{i} \right) - \ln(n) \leq 1$$

bewiesen und daraus folgt die Behauptung unmittelbar.

Aufgabe 6: Es sei $\mathcal{B}$ die Menge der binären Bäume, die im Skript definiert wird.

(a) Spezifizieren Sie eine Methode

$$\mathit{isOrdered} : \mathcal{B} \to \mathbb{B}$$

durch bedingte Gleichungen. Für einen binären Baum b soll der Aufruf $b.\mathit{isOrdered}()$ genau dann `true` zurück liefern, wenn $b \in \mathcal{B}_<$ gilt. (8 Punkte)

Hinweis: Definieren Sie sich geeignete Hilfsfunktionen.

(b) Es sei *insert*() die in Abschnitt 7.2 des Skripts definierte Methode. Nehmen Sie an, dass Sie für alle $b \in \mathcal{B}_<$, alle Schlüssel k und alle Werte v die Gleichung

$$b.\mathit{insert}(k, v).\mathit{isOrdered}() = \texttt{true}$$

beweisen sollen. Geben Sie an, welche Lemmata über die in Teil (a) definierten Hilfsfunktionen zu einem solchen Beweis benötigt werden. (4 Punkte)

(c) Zeigen Sie nun für geordnete binäre Bäume die Gleichung

$$b.\mathit{insert}(k, v).\mathit{isOrdered}() = \texttt{true}$$ (12 Punkte)

Sie dürfen dabei die Lemmata, die Sie in Teil (b) angeben sollen, benutzen.

Lösung:

(a) Wir definieren zunächst zwei Hilfsfunktionen

$$\mathit{smaller} : \mathcal{B} \times \mathit{Key} \to \mathbb{B} \quad \text{und} \quad \mathit{bigger} : \mathcal{B} \times \mathit{Key} \to \mathbb{B}.$$

Der Aufruf $b.\mathit{smaller}(k)$ soll als Ergebnis genau dann `true` liefern, wenn alle in dem Baum b auftretenden Schlüssel kleiner als der Schlüssel k sind. Analog liefert der Aufruf $b.\mathit{bigger}(k)$ als Ergebnis genau dann `true`, wenn alle in dem Baum b auftretenden Schlüssel größer als der Schlüssel k sind. Die beiden Funktionen werden durch die folgenden Gleichungen definiert:

1. $\mathit{nil}.\mathit{smaller}(k) = \texttt{true}$.
2. $\mathit{node}(k_1, v_1, l, r).\mathit{smaller}\big(k\big) = \big(k_1 < k \wedge l.\mathit{smaller}(k) \wedge r.\mathit{smaller}(k)\big)$.
3. $\mathit{nil}.\mathit{bigger}(k) = \texttt{true}$.
4. $\mathit{node}(k_1, v_1, l, r).\mathit{bigger}\big(k\big) = \big(k_1 > k \wedge l.\mathit{bigger}(k) \wedge r.\mathit{bigger}(k)\big)$.

Damit kann nun die Funktion *isOrdered*() wie folgt spezifiziert werden:

1. $nil.isOrdered() = \texttt{true}$.
2. $node(k, v, l, r).isOrdered() = \big(l.smaller(k) \wedge r.bigger(k) \wedge l.isOrdered() \wedge r.isOrdered()\big)$.

(b) Wir benötigen die folgenden beiden Eigenschaften der Funktionen *smaller* und *bigger*:

1. $k < k_1 \wedge l.smaller(k_1) \rightarrow l.insert(k, v).smaller(k_1)$,
2. $k > k_1 \wedge r.bigger(k_1) \rightarrow r.insert(k, v).bigger(k_1)$.

(c) Wir führen den Beweis durch Wert-Verlaufsinduktion.

1. Fall: $nil.insert(k, v).isOrdered() = node(k, v, nil, nil).isOrdered() = \texttt{true}$.
2. Fall: Es gelte $k < k_1$. Dann können wir voraussetzen, dass

   $$node(k_1, v_1, l, r).isOrdered()$$

   gilt und müssen zeigen, dass daraus

   $$node(k_1, v_1, l, r).insert(k, v).isOrdered()$$

   folgt. Dies sehen wir wie folgt:

   $$\begin{aligned} & node(k_1, v_1, l, r).insert(k, v).isOrdered() \\ = \;& node(k_1, v_1, l.insert(k, v), r).isOrdered() \\ = \;& \big(l.insert(k, v).smaller(k_1) \wedge r.bigger(k_1) \wedge \\ & \quad l.insert(k, v).isOrdered() \wedge r.isOrdered()\big) \end{aligned}$$

   Wir zeigen nun, dass alle vier Bestandteile dieser Konjunktion den Wert `true` haben.

   i. Aus der Voraussetzung, dass $node(k_1, v_1, l, r).isOrdered()$ gilt, folgt zunächst

   $$l.smaller(k_1).$$

   Nach Teil (b) folgt wegen $k < k_1$ dann auch

   $$l.insert(k, v).smaller(k_1).$$

   ii. Aus der Voraussetzung $node(k_1, v_1, l, r).isOrdered()$ folgt unmittelbar

   $$r.bigger(k_1).$$

   iii. Aus der Voraussetzung $node(k_1, v_1, l, r).isOrdered()$ folgt zunächst

   $$l.isOrdered().$$

   Nach Induktions-Voraussetzung haben wir dann auch

   $$l.insert(k, v).isOrdered().$$

   iv. Aus der Voraussetzung $node(k_1, v_1, l, r).isOrdered()$ folgt sofort

   $$r.isOrdered().$$

   Damit haben wir insgesamt gezeigt, dass $node(k_1, v_1, l.insert(k, v), r).isOrdered()$ gilt.
3. Fall: Es gelte $k > k_1$. Dann können wir voraussetzen, dass

   $$node(k_1, v_1, l, r).isOrdered()$$

   gilt und müssen zeigen, dass daraus

   $$node(k_1, v_1, l, r).insert(k, v).isOrdered()$$

   folgt, was darauf herausläuft zu zeigen, dass

   $$node\big(k_1, v_1, l, r.insert(k, v)\big).isOrdered()$$

   gilt. Da die Rechnung analog zum zweiten Fall ist, führen wir sie nicht weiter aus.

Aufgabe 7: Es gelte $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}, \texttt{f}\}$. Die Häufigkeit, mit der diese Buchstaben in dem zu kodierenden String s auftreten, sei durch die folgende Tabelle gegeben:

Buchstabe	a	b	c	d	e	f
Häufigkeit	8	9	10	11	12	13

(a) Berechnen sie einen optimalen Kodierungs-Baum für die angegebenen Häufigkeiten.

(b) Geben die Kodierung der einzelnen Buchstaben an, die sich aus diesem Baum ergibt.

Lösung:

(a) Wir wenden den Huffman-Algorithmus an und erhalten die folgenden Mengen. Zur Abkürzung schreiben wir dort $l(a, f)$ statt $\mathit{leaf}(a, f)$ und $n(l, r)$ statt $\mathit{node}(l, r)$.

1. $\{l(a,8), l(b,9), l(c,10), l(d,11), l(e,12), l(f,13)\}$
2. $\{l(c,10), l(d,11), l(e,12), l(f,13), n(l(a,8), l(b,9)) : 17\}$
3. $\{l(e,12), l(f,13), n(l(a,8), l(b,9)) : 17, n(l(c,10), l(d,11)) : 21\}$
4. $\{n(l(a,8), l(b,9)) : 17, n(l(c,10), l(d,11)) : 21, n(l(e,12), l(f,13)) : 25\}$
5. $\{n(l(e,12), l(f,13)) : 25, n(n(l(a,8), l(b,9)) : 17, n(l(c,10), l(d,11)) : 21) : 38\}$
6. $\{n(n(l(e,12), l(f,13)) : 25, n(n(l(a,8), l(b,9)) : 17, n(l(c,10), l(d,11)) : 21) : 38) : 63\}$

Damit ist

$$n(n(l(e,12), l(f,13)), n(n(l(a,8), l(b,9)), n(l(c,10), l(d,11))))$$

der gesuchte Kodierungsbaum.

(b) Also ergibt sich die folgende Kodierung für die einzelnen Buchstaben:

```
e = 00, f = 01, a = 100, b = 101, c = 110, d = 111.
```