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. Das charakteristische Polynom lautet $\chi(x) = x^2 - 1 = (x-1) \cdot (x+1)$. Es gilt $\chi(1) = 0$. Wegen $\chi'(x) = 2 \cdot x$ und $\chi'(1) = 2 \neq 0$ erhalten wir eine spezielle Lösung mit der Formel

$$a_n = \frac{2}{2} \cdot n = n.$$

Wegen $\chi(x) = (x-1) \cdot (x+1)$ lautet die allgemeine Lösung

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

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 lautet die Lösung

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

(b) Es handelt sich um eine lineare, homogene Rekurrenz-Gleichung der Ordnung 2. Das charakteristische Polynom lautet $\chi(x) = x^2 + x - 2 = (x-1) \cdot (x+2)$. Damit lautet die allgemeine Lösung

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

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 lautet die Lösung

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

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

$$t = \mathit{node}(11, \mathit{node}(10, \mathit{nil}, \mathit{nil}), \mathit{node}(15, \mathit{nil}, \mathit{node}(18, \mathit{node}(17, \mathit{nil}, \mathit{nil}), \mathit{node}(24, \mathit{nil}, \mathit{nil}))))$$

(a) Fügen Sie in diesem Baum den Schlüssel `16` ein und geben Sie den resultierenden Term an. (3 Punkte)

(b) Fügen Sie in dem in Teil (a) berechneten Baum den Schlüssel `13` ein und geben Sie den resultierenden Term an. (3 Punkte)

(c) Löschen Sie aus dem in Teil (b) berechneten Baum den Schlüssel `15` und geben Sie den resultierenden Term an. (4 Punkte)

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

Lösung: Wir wiederholen zunächst die Gleichungen, die das Einfügen und Löschen beschreiben:

1. Die Gleichungen für das Einfügen lauten:
   1. $\mathit{nil}.\mathit{insert}(k, v) = \mathit{node}(k, v, \mathit{nil}, \mathit{nil})$,
   2. $\mathit{node}(k, v_2, l, r).\mathit{insert}(k, v_1) = \mathit{node}(k, v_1, l, r)$,
   3. $k_1 < k_2 \rightarrow \mathit{node}(k_2, v_2, l, r).\mathit{insert}(k_1, v_1) = \mathit{node}\big(k_2, v_2, l.\mathit{insert}(k_1, v_1), r\big)$,
   4. $k_1 > k_2 \rightarrow \mathit{node}(k_2, v_2, l, r).\mathit{insert}(k_1, v_1) = \mathit{node}\big(k_2, v_2, l, r.\mathit{insert}(k_1, v_1)\big)$.
2. Die Gleichungen für das Löschen lauten:
   1. $\mathit{nil}.\mathit{delete}(k) = \mathit{nil}$,
   2. $\mathit{node}(k, v, \mathit{nil}, r).\mathit{delete}(k) = r$,
   3. $\mathit{node}(k, v, l, \mathit{nil}).\mathit{delete}(k) = l$,
   4. $l \neq \mathit{nil} \wedge r \neq \mathit{nil} \wedge r.\mathit{delMin}() = [r', k_{min}, v_{min}] \rightarrow$
      $\mathit{node}(k, v, l, r).\mathit{delete}(k) = \mathit{node}(k_{min}, v_{min}, l, r')$,
   5. $k_1 < k_2 \rightarrow \mathit{node}(k_2, v_2, l, r).\mathit{delete}(k_1) = \mathit{node}\big(k_2, v_2, l.\mathit{delete}(k_1), r\big)$,
   6. $k_1 > k_2 \rightarrow \mathit{node}(k_2, v_2, l, r).\mathit{delete}(k_1) = \mathit{node}\big(k_2, v_2, l, r.\mathit{delete}(k_1)\big)$.

Damit können wir nun die Teilaufgaben lösen. Um die Notation übersichtlich zu halten, kürzen wir *node*() durch `n`() ab und statt *nil* schreiben wir $*$

(a)
```
t =  n(11, n(10, *, *),
           n(15, *,
                 n(18, n(17, n(16, *, *), *),
                       n(24, *, *))))
```

(b)
```
t =  n(11, n(10, *, *),
           n(15, node(13, *, *),
                 n(18, n(17, n(16, *, *), *),
                       n(24, *, *))))
```

(c)
```
t =  n(11, n(10, *, *),
           n(16, node(13, *, *),
                 n(18, n(17, *, *),
                       n(24, *, *))))
```

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

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:

$$\begin{aligned} t = n(17, & n(8, n(2, *, *) : 1, n(10, n(13, *, *) : 1, *) : 2) : 3, \\ & n(23, *, *) : 1) : 4 \end{aligned}$$

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. $l_3 = *$,
8. $r_3 = n(13, *, *)$,
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

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

Das ist kein AVL-Baum mehr, denn an der Wurzel ist die Balancierungs-Bedingung verletzt. Wenn wir diesen Baum rebalancieren, erhalten wir:

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

Aufgabe 4:

(a) Zeigen Sie, dass für die Funktion *merge*, die wir im Skript definiert haben, folgende Gleichung gilt:

$$count(x, merge(L_1, L_2)) = count(x, L_1) + count(x, L_2) \qquad \text{(8 Punkte)}$$

(b) Zeigen Sie, dass für die Funktion *split*, die wir im Skript definiert haben, folgende Gleichung gilt:

$$split(L) = [L_1, L_2] \rightarrow count(x, L) = count(x, L_1) + count(x, L_2) \qquad \text{(7 Punkte)}$$

Lösung:

(a) Wir führen den Nachweis durch Wertverlaufs-Induktion nach der Definition von *merge*(). Um diese Induktion durchführen zu können, schlagen wir zunächst die Definition der Funktion *merge*() im Skript nach. Wir finden dort:

1. $\texttt{merge}([], L_2) = L_2$,
2. $\texttt{merge}(L_1, []) = L_1$,
3. $x \preceq y \rightarrow \texttt{merge}\big([x] + R_1, [y] + R_2\big) = [x] + \texttt{merge}\big(R_1, [y] + R_2\big)$.
4. $\neg x \preceq y \rightarrow \texttt{merge}\big([x] + R_1, [y] + R_2\big) = [y] + \texttt{merge}\big([x] + R_1, R_2\big)$.

Damit können wir jetzt die Induktion durchführen:

1. $L_1 = []$.
   Wir formen zunächst die linke Seite der Behauptung um.

   $$count(x, merge(L_1, L_2)) = count(x, merge([], L_2)) = count(x, L_2)$$

   Jetzt formen wir die rechte Seite der Behauptung um.

   $$\begin{aligned} count(x, L_1) + count(x, L_2) &= count(x, []) + count(x, L_2) \\ &= 0 + count(x, L_2) \\ &= count(x, L_2) \end{aligned}$$

   Da wir das selbe Ergebnis erhalten wie bei der Umformung der linken Seite der Gleichung, ist der erste Fall damit abgeschlossen.

2. $L_2 = []$. Dieser Fall ist analog zum ersten Fall.
3. $L_1 = [u] + R_1 \wedge L_2 = [v] + R_2 \wedge u \preceq v$.
   Wir formen zunächst die linke Seite der Behauptung um.

$$\begin{aligned} count(x, merge([u] + R_1, [v] + R_2)) &= count(x, [u] + merge(R_1, [v] + R_2)) \\ &= eq(x, u) + count(x, merge(R_1, [v] + R_2)) \\ &\overset{IV}{=} eq(x, u) + count(x, R_1) + count(x, [v] + R_2) \end{aligned}$$

   Jetzt formen wir die rechte Seite der Behauptung um.

$$count(x, [u] + R_1) + count(x, [v] + R_2) = eq(x, u) + count(x, R_1) + count(x, [v] + R_2)$$

   Da wir das selbe Ergebnis erhalten wie bei der Umformung der linken Seite der Gleichung, ist der dritte Fall damit abgeschlossen.
4. $L_1 = [u] + R_1 \wedge L_2 = [v] + R_2 \wedge \neg(u \preceq v)$.
   Dieser Fall ist analog zum dritten Fall.

(b) Wir führen den Nachweis durch Wertverlaufs-Induktion nach der Definition von *split*(). Um diese Induktion durchführen zu können, schlagen wir zunächst die Definition der Funktion *split*() im Skript nach. Wir finden dort:

1. $\texttt{split}(\texttt{[]}) = [\texttt{[]}, \texttt{[]}]$.
2. $\texttt{split}([x]) = [\texttt{[}x\texttt{]}, \texttt{[]}]$.
3. $\texttt{split}(R) = [R_1, R_2] \rightarrow \texttt{split}\big([x, y] + R\big) = \big[[x] + R_1, [y] + R_2\big]$.

Damit können wir die Induktion durchführen:

1. $L = []$. Dann gilt $L_1 = []$ und $L_2 = []$. Wir formen die linke Seite der zu beweisenden Gleichung $count(x, L) = count(x, L_1) + count(x, L_2)$ um:

   $$count(x, L) = count(x, []) = 0.$$

   für die rechte Seite gilt:

   $$count(x, L_1) + count(x, L_2) = count(x, []) + count(x, []) = 0 + 0 = 0.$$
2. $L = [u]$. Dann gilt $L_1 = [u]$ und $L_2 = []$. Wir formen die linke Seite der zu beweisenden Gleichung um:

   $$count(x, L) = count(x, [u]) = eq(x, u).$$

   für die rechte Seite gilt:

$$\begin{aligned} count(x, L_1) + count(x, L_2) &= count(x, [u]) + count(x, []) \\ &= eq(x, u) + 0 \\ &= eq(x, u). \end{aligned}$$

3. $L = [u, v] + R$. für den rekursiven Aufruf von *split*() gelte

   $$split(R) = [R_1, R_2].$$

   Nach der Induktions-Voraussetzung gilt dann zunächst

   $$count(x, R) = count(x, R_1) + count(x, R_2).$$

   Jetzt formen wir die linke Seite der zu beweisenden Gleichung um:

$$\begin{aligned} count(x, L) &= count(x, [u, v] + R) \\ &= eq(x, u) + eq(x, v) + count(x, R) \\ &\overset{IV}{=} eq(x, u) + eq(x, v) + count(x, R_1) + count(x, R_2) \end{aligned}$$

Nun formen wir die rechte Seite der zu beweisenden Gleichung um:

$$\begin{aligned} count(x, L_1) + count(x, L_2) &= count(x, [u] + R_1) + count(x, [v] + R_2) \\ &= eq(x, u) + count(x, R_1) + eq(x, v) + count(x, R_2) \end{aligned}$$

Da dieser Ausdruck bis auf die Reihenfolge der Summanden mit dem für die linke Seite gefundenen Ausdruck übereinstimmt, ist der Beweis damit abgeschlossen.

Aufgabe 5: Betrachten Sie das folgende Programm:

```
double sum(unsigned n) {
    unsigned i = 0;
    unsigned 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

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

   genügt. (Diese Invariante ist etwas stärker als in der Vorlesung. Dadurch läßt sich dann später zeigen, dass beim Abbruch der Schleife die Gleichung $i = n+1$ gilt. In der Vorlesung hatten wir an dieser Stelle nur anschaulich argumentiert.)
   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:

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

   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

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

3. Nach Beendigung der Schleife gilt dann

   $$s = \tfrac{1}{2} \cdot (i-1) \cdot i \wedge i \leq n+1 \wedge \neg(i \leq n)$$

   Es gilt

   $$i \leq n+1 \wedge \neg(i \leq n) \rightarrow i = n+1$$

   Damit erfüllt das Programm die Spezifikation

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

(b) Jetzt die symbolische Programm-Ausführung:

```
1  double 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 die `while`-Schleife die folgende Invariante hat:

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

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 6: Im Abschnitt 7.2 des Skriptes 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.`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.`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

$$\leq\colon \mathit{Key} \times \mathcal{B} \to \mathbb{B}.$$

für einen Schlüssel k und einen binären Baum b soll $k \leq b$ genau dann gelten, wenn k kleinergleich als alle Schlüssel sind, die in b auftreten. Diese Funktion wird durch Gleichungen spezifiziert:

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

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

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

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

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

Aufgabe 7:

(a) Zeigen Sie $\log_2(n) \in \mathcal{O}\big(\ln(\sqrt{n})\big)$. (4 Punkte)

(b) Zeigen Sie $\big|\sin(n)\big| \in \mathcal{O}\big(1\big)$. (4 Punkte)

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

Hinweis: Zeigen Sie

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

Lösung:

(a) zunächst bemerken wir, dass gilt:

$$\log_2(n) = \frac{\ln(n)}{\ln(2)} \quad \text{und} \quad \ln(\sqrt{n}) = \ln\left(n^{\frac{1}{2}}\right) = \tfrac{1}{2} \cdot \ln(n).$$

Wir zeigen die Behauptung nun mit dem Grenzwertsatz:

$$\begin{aligned}
\lim_{n\to\infty} \frac{\log_2(n)}{\ln(\sqrt{n})} &= \lim_{n\to\infty} \frac{\frac{\ln(n)}{\ln(2)}}{\frac{1}{2}\ln(n)} \\
&= \lim_{n\to\infty} \frac{\frac{1}{\cancel{\ln(2)}}}{\frac{1}{2}} \\
&= \lim_{x\to\infty} \frac{2}{\ln(2)} \\
&= \frac{2}{\ln(2)}
\end{aligned}$$

Da der Grenzwert existiert, folgt die Behauptung aus dem Grenzwertsatz.

(b) Das ist eigentlich eine Scherzaufgabe. Aus der Geometrie wissen wir, dass

$$0 \leq |\sin(n)| \leq 1 \quad \text{für alle } n \in \mathbb{N}$$

gilt, woraus die Behauptung unmittelbar folgt. Die Aufgabe ist ein Beispiel, bei dem der Grenzwertsatz versagt.

(c) 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 $x \in [i-1, i]$:

$$
\begin{aligned}
& x \geq i-1 \\
\Rightarrow \quad & \frac{1}{x} \leq \frac{1}{i-1} \\
\Rightarrow \quad & \int_{i-1}^{i} \frac{1}{x} dx \leq \int_{i-1}^{i} \frac{1}{i-1} dx \\
\Rightarrow \quad & \int_{i-1}^{i} \frac{1}{x} dx \leq \frac{1}{i-1} \\
\Rightarrow \quad & \sum_{i=2}^{n} \int_{i-1}^{i} \frac{1}{x} dx \leq \sum_{i=2}^{n} \frac{1}{i-1} \\
\Rightarrow \quad & \int_{1}^{n} \frac{1}{x} dx \leq \sum_{i=2}^{n} \frac{1}{i-1} \\
\Rightarrow \quad & \ln(n) - \ln(1) \leq \sum_{i=2}^{n} \frac{1}{i-1} \\
\Rightarrow \quad & \ln(n) \leq \sum_{i=1}^{n-1} \frac{1}{i} \\
\Rightarrow \quad & 0 \leq \sum_{i=1}^{n-1} \frac{1}{i} - \ln(n) \\
\Rightarrow \quad & 0 \leq \sum_{i=1}^{n} \frac{1}{i} - \ln(n)
\end{aligned}
$$

Insgesamt haben wir damit die Ungleichung

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

bewiesen und daraus folgt die Behauptung unmittelbar.

Aufgabe 8: Es sei $\mathcal{B}$ die Menge der binären Bäume, die im Kapitel 6 des Skriptes 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 6.1 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 b die Gleichung

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

Lösung:

(a) zunächst spezifizieren wir zwei Methoden

$$\leq\colon Key \times \mathcal{B} \to \mathbb{B} \quad \text{und} \quad \leq\colon Key \times \mathcal{B} \to \mathbb{B}$$

durch bedingte Gleichungen. Die Beziegung $k \leq b$ soll für einen Schlüssel k und einen Baum b genau dann gelten, wenn k kleiner-gleich alle Schlüssel aus b ist und $b \leq k$ gilt genau dann, wenn alle Schlüssel aus b k kleiner-gleich k sind. Die Beziegung $k \leq b$ haben wir bereits in Aufgabe 6 angegeben. Analog definieren wir jetzt $b \leq k$:

1. $nil \leq k$,
2. $node(k_1, v_1, l, r) \leq k = \big(k_1 \leq k \wedge l \leq k \wedge r \leq k\big)$.

Damit können wir nun Gleichungen für *isOrdered*() angeben.

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

(b) Wir benötigen die folgenden beiden Lemmata:

1. $k \leq b.insert(k_0, v_0) = k \leq k_0 \wedge k \leq b$,
2. $b.insert(k_0, v_0) \leq k = k_0 \leq k \wedge b \leq k$.

(c) Wir beweisen die Behauptung durch Wertverlaufs-Induktion nach der Definition von *insert*(). Dazu betrachten wir die folgenden Gleichungen aus dem Skript:

1. $nil.insert(k, v) = node(k, v, nil, nil)$,
2. $node(k, v_2, l, r).insert(k, v_1) = node(k, v_1, l, r)$,
3. $k_1 < k_2 \to node(k_2, v_2, l, r).insert(k_1, v_1) = node\big(k_2, v_2, l.insert(k_1, v_1), r\big)$,
4. $k_1 > k_2 \to node(k_2, v_2, l, r).insert(k_1, v_1) = node\big(k_2, v_2, l, r.insert(k_1, v_1)\big)$.

Jetzt führen wir die Induktion durch:

1. $b = nil$

$$\begin{aligned} nil.insert(k, v).isOrdered() &= node(k, v, nil, nil).isOrdered() \\ &= nil \leq k \wedge k \leq nil \wedge nil.isOrdered() \wedge nil.isOrdered() \\ &= \texttt{true} \wedge \texttt{true} \wedge \texttt{true} \wedge \texttt{true} \\ &= \texttt{true} \end{aligned}$$

2. $b = node(k, v_0, l, r)$
   Da $b \in \mathcal{B}_<$ ist, wissen wir, dass $b.isOrdered() = \texttt{true}$ ist. Daraus folgt sofort

   $$l \leq k \wedge k \leq r \wedge l.isOrdered() \wedge r.isOrdered()$$

   Damit können wir nun den Nachweis führen:

$$\begin{aligned} & node(k, v_0, l, r).insert(k, v).isOrdered() \\ = \;& node(k, v, l, r).isOrdered() \\ = \;& l \leq k \wedge k \leq r \wedge l.isOrdered() \wedge r.isOrdered() \end{aligned}$$

   und die letzte Formel haben wir oben schon aus der Tatsache gefolgert, dass b ein geordneter binärer Baum ist.

3. $b = node(k_0, v_0, l, r)$ und $k \leq k_0$. Dann gilt

$$\begin{aligned}
& node(k_0, v_0, l, r).insert(k, v).isOrdered() \\
= \;& node(k_0, v_0, l.insert(k, v), r).isOrdered() \\
= \;& l.insert(k, v) \leq k_0 \wedge k_0 \leq r \wedge l.insert(k, v).isOrdered() \wedge r.isOrdered() \\
\overset{IV}{=} \;& l.insert(k, v) \leq k_0 \wedge k_0 \leq r \wedge \texttt{true} \wedge r.isOrdered() \\
= \;& l \leq k_0 \wedge k \leq k_0 \wedge k_0 \leq r \wedge \texttt{true} \wedge r.isOrdered() \quad \text{nach Lemma aus (b)} \\
= \;& \texttt{true} \qquad \text{wegen } b.isOrdered()
\end{aligned}$$

4. $b = node(k_0, v_0, l, r)$ und $\neg(k \leq k_0)$.
   Dieser Fall ist analog zum 3. Fall.

Aufgabe 9: 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 $leaf(a, f)$ und $n(l, r)$ statt $node(l, r)$.

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

   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) Damit ergibt sich die folgende Kodierung für die einzelnen Buchstaben:

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