# Algorithmen und Datenstrukturen 1

Ausgearbeitetes Übungsblatt 3

Datum: 29. April 2005
Erstellt mit LaTeX

## Aufgabe 3.1

**Aufgabenstellung:**

Geben Sie den Pseudocode eines Algorithmus an, der folgendes Problem löst:
Gegeben sind ein binärer Suchbaum und ein Wert $x$. Finde den Knoten mit dem größten Schlüssel in dem Baum, der höchstens so groß ist wie $x$. Falls ein solcher Knoten nicht existiert, soll die Funktion den Wert NULL zurückgeben. Die Laufzeit Ihrer Funktion soll $O(h)$ sein, wobei $h$ die Höhe des Baumes ist. Achtung: Auch falls kein Knoten mit Schlüssel x im Baum existiert, aber ein Knoten mit kleinerem Schlüssel enthalten ist, muss Ihre Funktion einen Knoten zurückgeben.

**Lösung:**

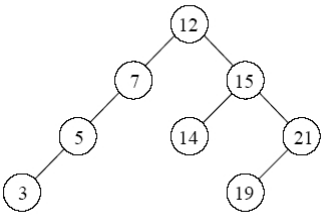
```
FindeKnoten(r, x) {
    p = NULL;
    while(r != NULL) {
        if(r.key <= x) {
            if(p != NULL) {
                if(r.key > p.key) {
                    p = r;
                }
            }
            else {
                p = r;
            }
        }
        if(r.key >= x) {
            r = r.leftson;
        }
        else {
            r = r.rightson;
        }
    }
    if(p == NULL) {
        return -1;
    }
    else {
        return p;
    }
}
```

---

## Aufgabe 3.2

**Aufgabenstellung:**

Betrachten Sie den folgenden binären Suchbaum:



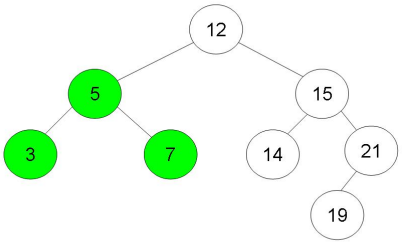

(a) Ist dieser binäre Suchbaum ein AVL-Baum? - Falls nein, setzen Sie Schritte, die ihn in einen AVL-Baum verwandeln, und zeichnen Sie den Baum in seinem korrigierten Zustand.

(b) Führen Sie danach folgende Operationen in diesem AVL-Baum in der angegebenen Reihenfolge durch und zeichnen Sie den Baum nach jeder Operation; achten Sie darauf, dass die AVL-Eigenschaft immer erhalten bleibt:

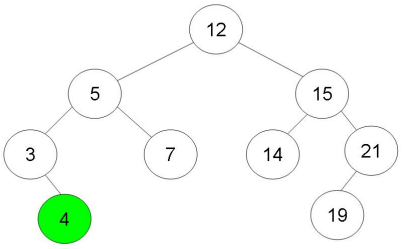
- fügen Sie 14 ein;
- fügen Sie 20 ein;
- löschen Sie 21;
- fügen Sie 13 ein.

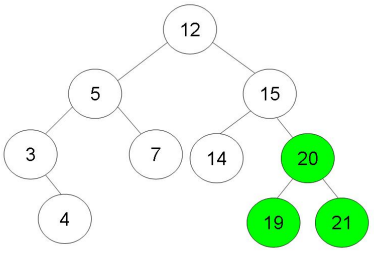
**Lösung:**

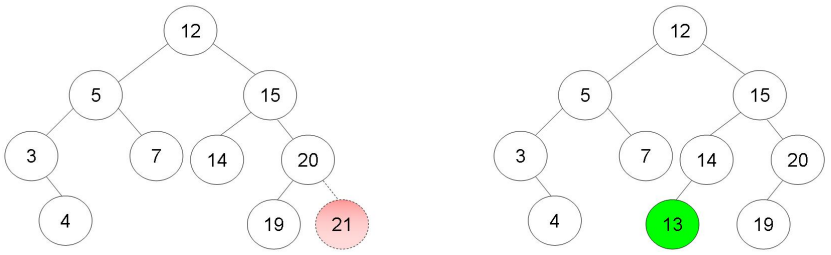
(a) Dieser Baum ist kein gültiger AVL-Baum, da die Differenz zwischen Höhe des rechten Unterbaums und Höhe des linken Unterbaums beim Knoten „7“ $-2$ beträgt. Eine Rotation von 5 um 7 sollte das Problem beheben:




(b) Durchführung der einzelnen Operationen; grün sind die veränderten Knoten gekennzeichnet, rot die Knoten, die gelöscht wurden. Beim Einfügen von 20 muss eine Doppelrotation um 21 durchgeführt werden, ansonsten sind keine Aktionen notwendig, um die AVL-Eigenschaft des Baumes zu gewährleisten.






---

# Aufgabe 3.3

**Aufgabenstellung:**

Beschreiben Sie die Vorteile eines AVL-Baumes gegenüber einem natürlichen binären Baum anhand eines selbst entwickelten Beispiels. Ihr Baum soll genau neun Knoten enthalten und im Fall des natürlichen binären Baumes zu einer Mindesthöhe von sechs Ebenen führen, die durch den AVL-Baum verhindert wird. Geben Sie die Knotenwerte in der Folge des Einfügens an, und zeichnen Sie den jeweiligen Zustand beider Bäume nach dem Einfügen jedes einzelnen Wertes. Geben Sie dabei beim AVL-Baum auch die Zwischenschritte an, die eventuell nötig werden.

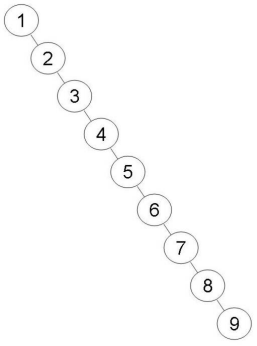
**Lösung:**

Im Folgenden soll die Zahlenfolge

$$\langle 1, 2, 3, 4, 5, 6, 7, 8, 9 \rangle$$

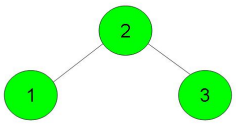
verwendet werden, um zunächst einen natürlichen binären Suchbaum damit zu erstellen, und anschließend, um einen AVL-Baum zu erstellen.

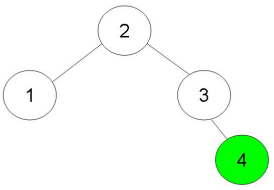
Der natürliche binäre Baum entartet zur linearen Liste (ohne Angabe der Zwischenschritte):



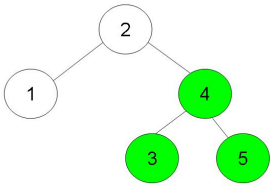

Und nun zum AVL-Baum:


Beim Einfügen von 3 wird eine Rotation von 2 um 1 erforderlich.

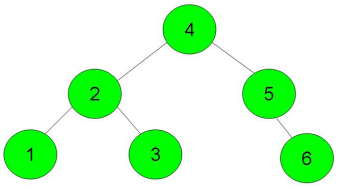


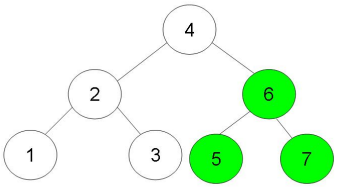

Beim Einfügen von 5 muss 4 um 3 rotiert werden.

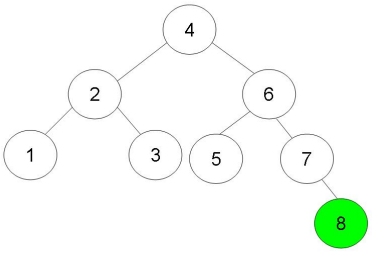



Das Einfügen von 6 macht eine Rotation von 4 um 2 erforderlich.

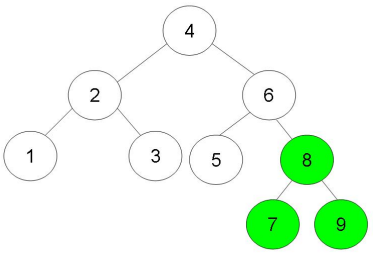



Rotation von 6 um 5 beim Einfügen von 7.




Und schließlich eine Rotation von 8 um 7 beim Einfügen von 9.




---

# Aufgabe 3.4

**Aufgabenstellung:**

Gegeben sei ein binärer Suchbaum. Jeder Knoten $A$ des Baumes besteht aus dem Schlüssel $A.schluessel$ (ein Character-String) und den Verweisen $A.links$ und $A.rechts$ auf den linken bzw. rechten Unterbaum. Sei $T$ ein Verweis auf den Wurzelknoten des Baumes.

(a) Schreiben Sie einen möglichst effizienten rekursiven Algorithmus *PrüfeKnotenbalancierung* mit geeigneten Parametern, der für den Baum T feststellt, ob dieser knotenbalanciert ist, und das Ergebnis (wahr/falsch) als Rückgabewert liefert.
Ein Baum ist genau dann knotenbalanciert, wenn für jeden Knoten gilt, dass sich die Anzahl der Knoten im linken bzw. rechten Unterbaum maximal um eins unterscheidet.

(b) Geben Sie an, wie Ihr Algorithmus von einem Hauptprogramm aus für Baum $T$ aufgerufen werden muss.

(c) Wie hoch ist der Zeitaufwand des Algorithmus in $\Theta$-Notation in Abhängigkeit von der Knotenanzahl $n$? Begründen Sie Ihre Antwort mit wenigen Worten.

**Lösung:**

(a)

```
PrüfeKnotenbalancierung(T, var Anzahl) {
    if(T == NULL) {
        Anzahl = 0;
        return true;
    }
    Balance = true;
    AnzahlLinks = 0;
    AnzahlRechts = 0;
    if(T.links != NULL) {
        if(PrüfeKnotenbalancierung(T.links, AnzahlLinks) == false) {
            Balance = false;
        }
    }
    if(Balance == true) {
        if(T.rechts != NULL) {
            if(PrüfeKnotenbalancierung(T.rechts, AnzahlRechts) == false) {
                Balance = false;
            }
        }
    }
    if(Balance == true) {
        if(|AnzahlLinks - AnzahlRechts| <= 1) {
            Anzahl = AnzahlLinks + AnzahlRechts + 1;
            return true;
        }
    }
    Anzahl = 0;
    return false;
}
```

(b) Aufruf durch

```
PrüfeKnotenbalancierung(T, Anzahl);
```

(c) Für jeden Knoten ist im Worst Case eine Ausführung des Algorithmus notwendig (in allen anderen Fällen nicht, da dann nicht der ganze Baum durchgesehen wird). Die Ausführungszeit für den Algorithmus ohne Berücksichtigung der Rekursionen) liegt in $\Theta(1)$, liegt also für $n$ Knoten in $\Theta(n)$.

---

# Aufgabe 3.5

**Aufgabenstellung:**

(a) Füugen Sie die Elemente der Folge

$$\langle 10, 18, 19, 20, 35, 34, 21, 23, 25, 28, 26 \rangle$$

in dieser Reihenfolge in einen anfangs leeren B-Baum der Ordnung 3 ein. Zeichnen Sie den B-Baum nach dem Einfügen jedes einzelnen Elements.

(b) Fügen Sie zum Vergleich dieselben Elemente in derselben Reihenfolge in einen anfangs leeren natürlichen binären Suchbaum ein. Zeichnen Sie den Baum in seinem endgültigen Zustand. Welche Vor- bzw. Nachteile der beiden Datenstrukturen sind im direkten Vergleich zu erkennen?

**Lösung:**

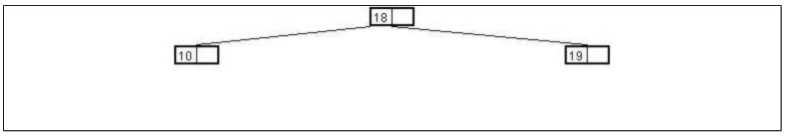
(a) Die folgenden Abbildungen wurden mit Hilfe eines B-Baum-Applets auf

http://www.gruntz.ch/courses/inf3/slides/BBaum/BBaum.html

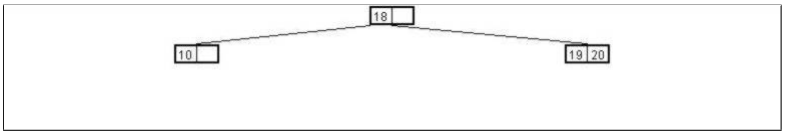
erstellt.

Einfügen von 10 und 18:

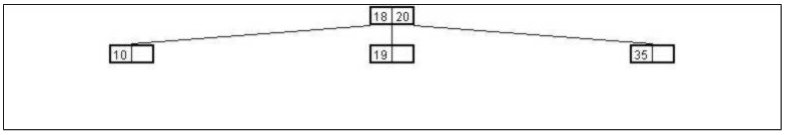

Beim Einfügen von 19 muss die Wurzel geteilt werden, 18 (der mittlere Schlüssel) wird neue Wurzel, 10 und 19 Kinder davon:



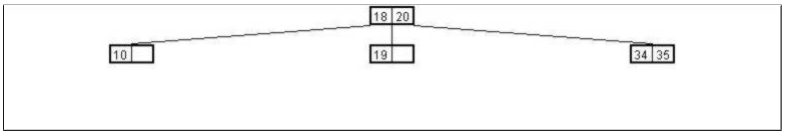

Einfügen von 20:



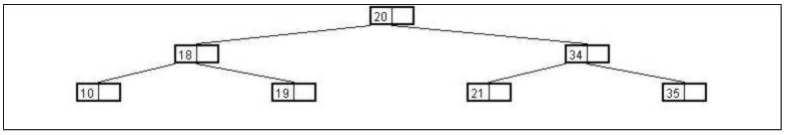
Beim Einfügen von 35 wird der Knoten geteilt, 20 (der mittlere Schlüssel) kommt in die Wurzel, 19 und 35 werden getrennt, werden jeweils Kind der Wurzel.



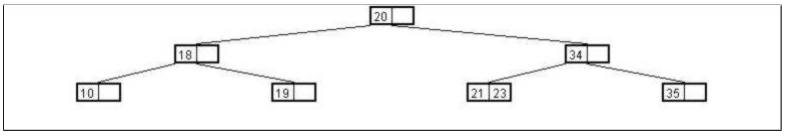

Einfügen von 34:



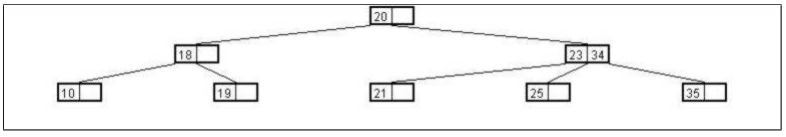

Beim Einfügen von 21 wird der Knoten wieder geteilt, 34 (mittlerer Schlüssel) kommt in die Wurzel, 21 und 35 werden Kinder der Wurzel; nun enthält allerdings die Wurzel drei Schlüssel, wird daher geteilt, 20 (der mittlere Schlüssel wird neue Wurzel, 18 und 34 Kinder davon.



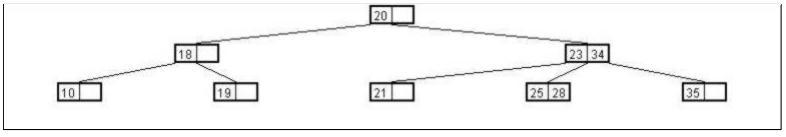

Einfügen von 23:



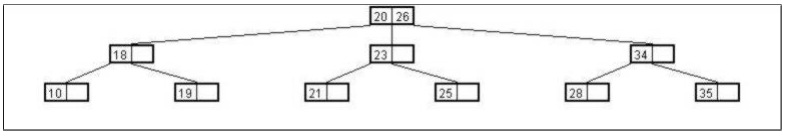

Beim Einfügen von 25 wird der Knoten wieder geteilt, der mittlere Schlüssel 23 kommt in den Elternknoten.




Einfügen von 28:




Beim Einfügen von 26 wird der Knoten wieder geteilt, 26 kommt in den Elternknoten; dieser wird ebenfalls geteilt, 26 schafft es bis in die Wurzel.




(b) Der natürliche binäre Baum entartet mehr oder weniger in eine lineare Liste (ohne Abbildung). Es zeigt sich daher im Vergleich zwischen B-Baum und natürlichem binären Baum: Das Einfügen in einen natürlichen binären Suchbaum ist schneller, der Baum entartet im schlechtesten Fall - so wie in diesem Beispiel jedoch in einen Baum mit der Höhe $h = n - 1$, wobei $n$ die Anzahl der Knoten in diesem Baum darstellt. Ein Element in einen B-Baum einzufügen, ist zwar aufwändiger, doch ist ein B-Baum immer - so auch in diesem Beispiel - gut balanciert, was Suchoperationen beschleunigt.

# Aufgabe 3.6

**Aufgabenstellung:**

Eine Hashtabelle soll maximal 1500 Einträge aufnehmen; die Schlüssel sind *character strings*, interpretiert als $2^8$-adische Zahlen. Die Hashfunktion ist nach der Divisions-Rest-Methode implementiert. Die Kollisionsbehandlung erfolgt durch *Lineares Sondieren*. Geben Sie für jede der Zahlen 1413, 1664, 1789, 2048 und 10000 an, ob sie eine gute Wahl für die Tabellengröße $m$ wäre, und begründen Sie Ihre Entscheidungen.

**Lösung:**

- **1413** ist nicht geeignet, da eine solche Hashtabelle nicht alle Schlüssel aufnehmen kann.
- **1664** wäre geeignet, kann aufgrund der geringen Tabellengröße allerdings zu einer großen Anzahl von Kollisionen führen; außerdem ist zu beachten, dass 1664 eine gerade Zahl ist, daher durch 2 teilbar ist und daher unnötig zu Kollisionen führen kann.
- **1789** ist weder zu klein, noch zu groß, liegt optimal zwischen den beiden Zweierpotenzen $2^{10} = 1024$ und $2^{11} = 2048$ und ist außerdem noch eine Primzahl. Diese Zahl ist daher am besten von allen genannten geeignet.
- **2048** ist ungeeignet, da $2^{11} = 2048$.
- **10000** ist ungeeignet, da aufgrund der großen Tabellengröße im Verhältnis zur Anzahl der höchstens abgespeicherten Schlüssel unnötigerweise Speicherplatz verschwendet wird.

---

# Aufgabe 3.7

**Aufgabenstellung:**

Gegeben ist eine Hashtabelle in Form eines Arrays *table* mit der festgelegten Größe $n$. Die Kollisionsbehandlung soll durch *Lineares Sondieren* erfolgen.
Jedes Element *table[i], i = 0, . . .n - 1*, des Arrays besteht aus folgenden Komponenten:

- *table[i].key* enthält den Schlüssel des Datensatzes;
- *table[i].data* enthölt die eigentlichen Daten;
- *table[i].status* enthölt einen der drei Werte *used*, *unused* und *reusable*, je nachdem, ob das fragliche Element gerade einen Datensatz enthält, noch nie einen Datensatz enthalten hat oder bereits einen Datensatz enthalten hat, aufgrund der mittlerweile erfolgten Löschung dieses Datensatzes aber wieder zur Verfügung steht.

(a) Schreiben Sie eine Prozedur in Pseudocode, die diese Hashtabelle für die Verwendung mit *Linearem Sondieren* korrekt und vollständig initialisiert, aber keine überflüssigen Initialisierungen vornimmt.

(b) Nehmen Sie an, dass eine Hashfunktion $a(k)$ existiert, die aus einem Schlüssel $k$ einen Hashindex für die oben beschriebene Tabelle berechnet.
Schreiben Sie eine Funktion in Pseudocode, die feststellt, ob ein Datensatz mit dem Schlüssel $k$ in der Tabelle enthalten ist. Falls ja, soll der Index des Datensatzes zurückgegeben werden; falls nein, der Wert $-1$.

(c) Nehmen Sie an, dass die Schlüssel der in der Hashtabelle zu speichernden Datensätze Zeichenketten sind, dass Sie auf die einzelnen Elemente eines Schlüssels $k$ als Zeichen *k.char[j], j = 0, . . .m - 1* zugreifen können, und dass eine Funktion $ord(c)$ existiert, die den numerischen Wert eines Zeichens $c$ liefert.
Schreiben Sie eine Hashfunktion $a'(k)$ in Pseudocode, welche die Divisionsrestmethode verwendet, um aus den Zeichen des Schlüssels $k$ einen Hashindex zu berechnen.

**Lösung:**

(a)

```
Initialisieren(table, n) {
    for(i=0; i<n; i++) {
        table[i].status = unused;
    }
}
```

(b)

```
Suche(table, n, k) {
    index = a(k);
    if(table[index].key == k) {
        return index;
    }
    b = index + 1;
    while(b != index && table[b].status == used) {
        if(table[b].key == k) {
            return b;
        }
        b = (b + 1) \% n;
    }
    return -1;
}
```

(c)

```
Hashindex(k, m, n) {
    index = 0;
    for(i=0; i<m; i++) {
        index += 128^(l-i+1) * ord(k.char[i]);
    }
    return index mod n;
}
```

---

# Aufgabe 3.8

**Aufgabenstellung:**

Skizzieren Sie eine Hashtabelle mit $m = 11$ Feldern und tragen Sie mittels *Quadratischem Sondieren* folgende Werte in der gegebenen Reihenfolge ein:

$$\langle 13, 2, 5, 10, 25, 16, 12 \rangle .$$

Die Hashfunktion und die Konstanten für die Sondierungsfunktion lauten:

$$h'(k) = (2k + 3) \bmod m \text{ mit } c1 = 4,\ c2 = 3.$$

**Lösung:**

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** |
|---|---|---|---|---|---|---|---|---|---|---|
| 16 | 10 | 5 | 2 | | 12 | | 13 | | 25 | |

---

# Aufgabe 3.9

**Aufgabenstellung:**

(a) Entwerfen Sie eine Hashtabelle mit *Double Hashing* zur Speicherung von bis zu zehn ganzen Zahlen im Bereich von -120 bis +230.
Dimensionieren Sie die Tabelle so, dass einerseits nicht zu viele Kollisionen auftreten, andererseits aber auch nicht zu viel Speicher vergeudet wird, und achten Sie darauf, dass Ihre Wahl der Tabellengröße einer „guten“ Wahl für die beiden Hashfunktionen nicht im Weg steht.
Geben Sie außerdem zwei „gute“ Hashfunktionen basierend auf Divisions-Rest-Methode bzw. Multiplikationsmethode an. Begründen Sie Ihre Entscheidungen.

(b) Testen Sie Ihre Tabelle, indem Sie die Zahlen der Folge

$$\langle 12, 13, 28, 10, 14, 32, 43 \rangle$$

in der angegebenen Reihenfolge in Ihre Tabelle einfügen. Zeichnen Sie dabei den Zustand der Tabelle nach dem Einfügen jeder einzelnen Zahl.

**Lösung:**

Hashfunktion für *Double Hashing*:

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \, mod \, m$$

Die Listengröße sei 13; es handelt sich dabei um eine Primzahl, außerdem ist diese Zahl etwas größer als die Anzahl der maximal in der Hashtabelle zu speichernden Einträge (11 Einträge).
Hashfunktionen nach der Divisionsrestmethode:

$$h_1(k) = k \, mod \, m$$

$$h_2(k) = 1 + (k \, mod \, m')$$

wobei $m' = m - 2 = 11$.
Ergebnis nach dem Einfügen der vorgegebenen Werte in die Hashtabelle:

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | 14 | 28 | - | 43 | - | 32 | - | - | - | 10 | - | 12 |

Hashfunktionen nach der Multiplikationsmethode:

$$h_1(k) = \lfloor m \cdot (k \cdot a \, mod \, 1) \rfloor$$

mit $A = \frac{\sqrt{5}-1}{2}$ (goldener Schnitt)

$$h_2(k) = 1 + \lfloor m \cdot (k \cdot a \, mod \, 1) \rfloor$$

Ergebnis nach dem Einfügen der vorgegebenen Werte in die Hashtabelle:

| **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 13 | - | 10 | 28 | - | 12 | - | 43 | 14 | - | 32 | - | - |

---

# Aufgabe 3.10

**Aufgabenstellung:**

Gegeben sei eine Hashtabelle $H_2$ mit $m_2 = 7$ Feldern. Die Kollisionsbehandlung erfolgt mittels *Double Hashing*. Die beiden Hashfunktionen lauten:

$$h_1(k) = (5 + k3k) \bmod 7$$
$$h_2(k) = (k - 1) \bmod 5$$

Wie lautet die entsprechende Hashfunktion $h(k,i)$? Die Hashtabelle $H_2$ befindet sich nach dem Eintragen einiger Elemente in folgendem Zustand:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 3 | | | 4 | 2 | | 12 |

Tragen Sie nun das Element 9 unter Verwendung der *Verbesserung nach Brent* in die Hashtabelle $H_2$ ein und geben Sie alle dabei berechneten Hashwerte an.

**Lösung:**

Hashfunktion:

$$((5 + 3k) \, mod 7 + (i \cdot (k-1) \, mod 5)) \, mod 7$$

Entstehende Hashtabelle:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 3 | | | 4 | 9 | 2 | 12 |