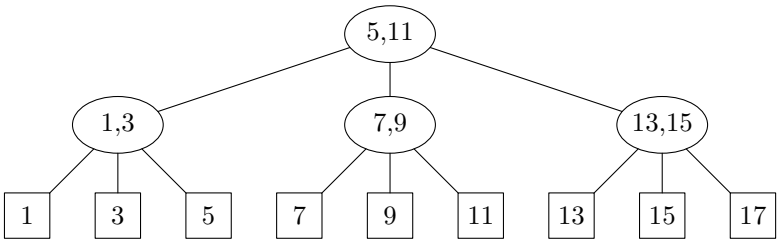
**RWTH Aachen** **SoSe 2007**
**Lehrgebiet Theoretische Informatik** Blatt 6
Kneis—Mölle—Richter—Rossmanith—Thiemann 14.5.2007

# Übungen zur Vorlesung Datenstrukturen und Algorithmen

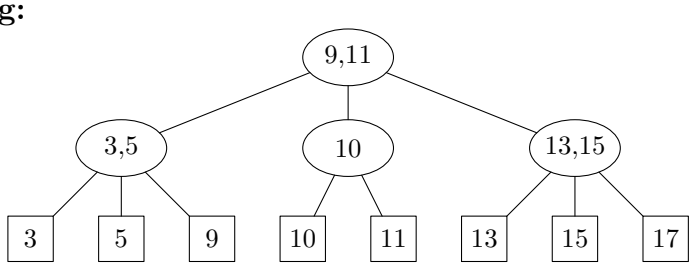
## T13

Vorgegeben sei der folgende $(2,3)$-Baum:




Was passiert, wenn Sie die 10 einfügen, dann die 7 löschen und abschließend auch die 1 löschen?

**Lösungsvorschlag:**




## T14

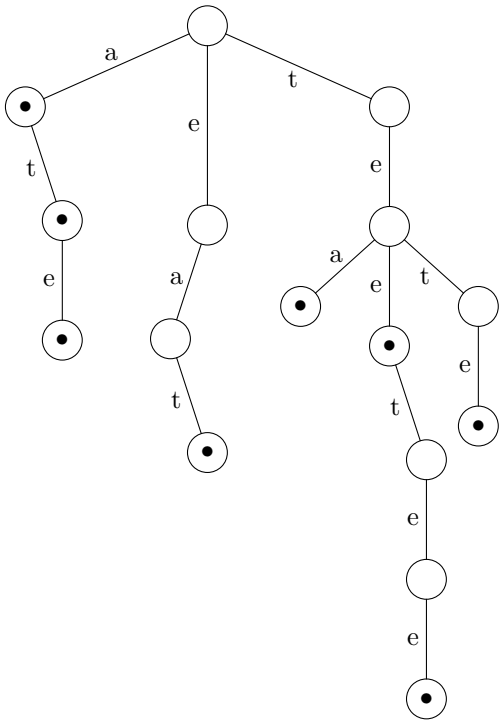
Englische Wörter über dem Alphabet $\{a, e, t\}$ sind

$$\{\text{a, at, ate, eat, tea, tee, teetee, tete}\}.$$

Was ist der längste sinnvolle und korrekte Satz, den Sie aus diesen Wörtern bilden können? Wie sieht ein Trie für diese Wörter aus?
Entwerfen Sie Algorithmen für das Einfügen und Löschen in Tries! Löschen Sie „tea“, „eat“ und „teetee“ aus dem obigen Trie und fügen Sie abschließend noch deutsche Wörter aus diesen drei Buchstaben ein.

**Lösungsvorschlag:**

Unser Vorschlag zur ersten Teilaufgabe: At tea, a teetee ate a tete. Der Trie sieht wie folgt aus:



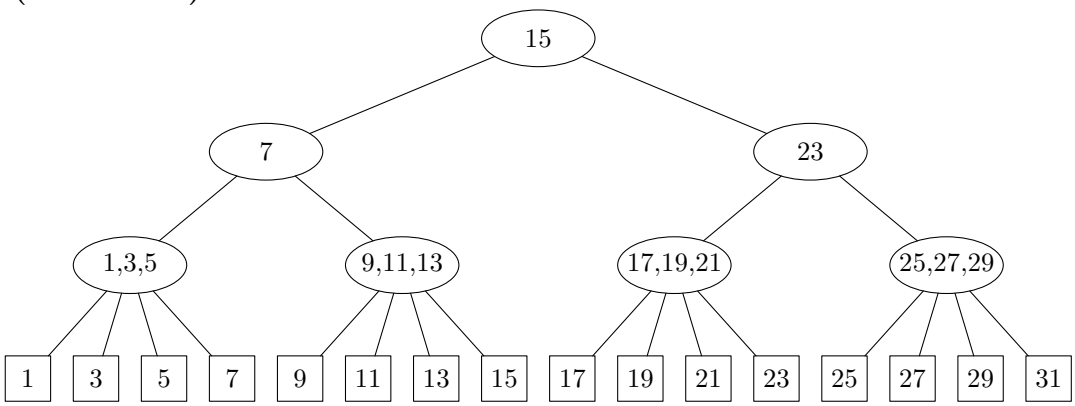

Zum Löschen wird zuerst der Knoten gesucht und seine Markierung entfernt. Dann werden iterativ alle Blätter ohne Markierung entfernt. Beim Einfügen wird der notwendige Teilbaum ergänzt und die entsprechende Markierung gesetzt. Die Laufzeit ist linear in der Wortlänge.
Deutsche AET-Wörter sind „taete“, „Tat“, „Tee“ und etwa „Tattee“ oder „Teetat“.

**T15**
Diskutieren Sie, welche Eigenschaften eine gute Hashfunktion für Strings haben sollte.
Es sei $m$ die Größe der Hashtabelle und $s = a_1 a_2 \dots a_n$ ein String, wobei wir die Zeichen mit ihren ASCII-Codes identifizieren. Wie gut sind die Hashfunktionen

$$\Big(\sum_{k=1}^{n} a_k\Big) \bmod m, \qquad \Big(\sum_{k=1}^{n} k \cdot a_k\Big) \bmod m, \qquad a_1^{a_2^{a_3}} \bmod m?$$

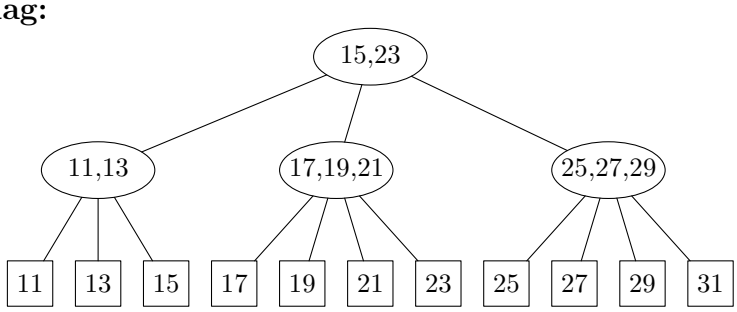
**Lösungsvorschlag:**
Natürlich sollten ähnliche Strings möglichst auf verschiedene Hashwerte abgebildet werden. Weiterhin sollte der Werteraum möglichst gleichmäßig beansprucht werden.
Kurze Wörter werden bei den ersten beiden Funktionen stets auf kleine Werte abgebildet, was insbesondere bei großen $m$ ungünstig ist. Beispiel: Es sei $m = 128^2$, und wir speichern alle Wörter der Länge zwei. Der Lastfaktor ist genau eins, aber die erste Hashfunktion bildet 128 Wörter auf den Wert 127 ab. Bei einer erfolgreichen Suche braucht man im Mittel etwa 32 Vergleiche, während dieser Wert bei einer guten Hashfunktion knapp über eins liegen sollte. Die zweite Hashfunktion verhält sich hier nicht deutlich besser als die erste.
Alle Wörter, die mit dem ASCII-Code 1 beginnen, werden von der dritten Hashfunktion auf den Hashwert 1 abgebildet. Außerdem werden alle Strings mit gleichem Präfix der Länge drei auf den gleichen Hashwert abgebildet, beispielsweise alle Sätze, die mit „Die“ beginnen, sowie – wenn wir Bytes statt ASCII-Codes betrachten – alle JPEG-Dateien (`FF D8 FF`).

**H12 (10 Punkte)**




Wie sieht dieser $(2, 4)$-Baum nach dem Löschen der Schlüssel 1, 3, 5, 7 und 9 aus? Wieviele Schlüssel können höchstens eingefügt werden, ohne daß die Höhe zunimmt?

**Lösungsvorschlag:**




Beweisen wir folgendes Lemma: Falls wir in einen Unterbaum einfügen, dessen Wurzel weniger als vier Kinder hat, dann wird die Wurzel nicht geteilt. Der Beweis ist einfach, wenn wir Induktion über die Höhe des Unterbaums verwenden. Hat der Unterbaum nur Höhe eins und weniger als vier Kinder, dann hat er nach dem Einfügen höchstens vier Kinder und wird daher nicht geteilt.

Ist seine Höhe größer, dann kann es passieren, daß ein Kind der Wurzel geteilt wird. Dann hat aber die Wurzel wieder höchstens vier Kinder und wird nicht geteilt.

Die Höhe des Baums nimmt nur zu, wenn die Wurzel geteilt wird. Solange es also einen inneren Knoten im Baum gibt, der weniger als vier Kinder hat, können wir unterhalb dieses Knotens gefahrlos einfügen. Wenn es keinen mehr gibt, dann hat er 64 Blätter. Wir können also 48 Schlüssel ohne Höhenzunahme hinzufügen.

**H13 (10 Punkte)**

Die Größe einer Hashtabelle wird gemäß folgender Strategie angepaßt:

- Falls sie mehr als 50 Elemente enthält und der Lastfaktor mindestens zwei ist, wird die Hashtabelle durch eine Tabelle doppelter Größe ersetzt.
- Falls sie mehr als 50 Elemente enthält und der Lastfaktor höchstens $1/2$ ist, wird die Hashtabelle durch eine Tabelle halber Größe ersetzt.
- Bei höchstens 50 Elementen wird eine Tabelle der Größe 100 verwendet.

Finden Sie eine geeignete Potentialfunktion, bezüglich welcher die amortisierten Kosten des Einfügens und Löschens konstant sind. Beweisen Sie dies.

Folgern Sie, daß $k$ Operationen auf einer Hashtabelle der Anfangsgröße $m$ bei universellem Hashing im Durchschnitt nur $O(m+k)$ Schritte benötigen.

**Lösungsvorschlag:**

Wir verwenden als Potentialfunktion die Anzahl der Operationen seit dem letzten Rehash mal einer Konstante $c$.

Falls kein Rehash stattfindet, dann sind die tatsächlichen Kosten im Erwartungswert durch $O(1+\alpha)$ beschränkt, da die Anzahl der Vergleiche im Durchschnitt höchstens $1+\alpha$ ist. Außerdem steigt das Potential um $c$, so daß die amortisierten Kosten höchstens $O(1+\alpha+c)$ betragen. Da $\alpha$ niemals größer als zwei ist, sind die amortisierten Kosten also höchstens $O(1)$.

Im Falle eines Rehash müssen alle Elemente der Hashtabelle durchgegangen werden, was $O(n+m)$ Zeit benötigt, falls $n$ die Anzahl der gespeicherten Elemente und $m$ die augenblickliche Größe der Hashtabelle ist. Für das Speichern in die neue Hashtabelle wird wiederum $O(n)$ Zeit benötigt, insgesamt also $O(n)$, da $m = O(n)$.

Seit dem letzten Rehash wurden aber mindestens $n$ Elemente gelöscht oder $n/2$ Elemente eingefügt. Die Potentialfunktion fällt auf 0, nimmt also um mindestens $cn/2$ ab.

Die amortisierten Kosten sind daher durch $O(n) - cn/2$ beschränkt, wobei die Konstante im $O(n)$ nicht von $c$ abhängt. Wenn wir $c$ groß genug wählen, sinken die amortisierten Kosten sogar unter null.