Prof. Dr. Gerd Stumme, Folke Eisterlehner, Dominik Benz
Fachgebiet Wissensverarbeitung

15.06.2009

10. Hausübung „Algorithmen und Datenstrukturen“

Sommersemester 2009

Abgabetermin: Montag, 22.06.2009, 10:00 Uhr

1 Anzahl Knoten in Bäumen mit Verweigungsgrad k

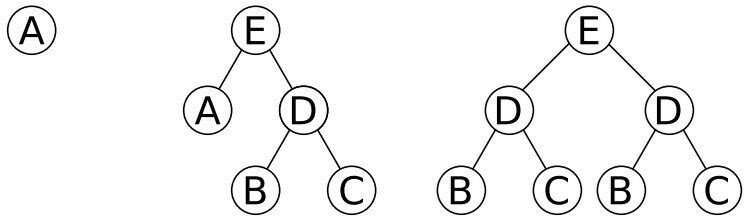
Bäume mit Verweigungsgrad k**:** Sei $k > 2$ eine natürliche Zahl und $\mathcal{S}$ eine abzählbare Menge von Schlüsselwerten. Die Menge $\mathsf{kB}_{\mathcal{S}}$ der *Bäume* (mit Verzweigungsgrad k) über $\mathcal{S}$ ist wie folgt definiert:

(B) Jeder Schlüssel $s \in \mathcal{S}$ ist ein Baum T der Höhe $\mathrm{h}(T) := 1$ mit Wurzel s (s ist ein *äußerer Knoten* oder auch *Blatt*).

(R) Sind $T_1, \ldots, T_k$ Bäume der Höhen $h_1, \ldots, h_k$ und ist $\ell \in \mathcal{S}$ ein Schlüssel, so ist $T := (\ell, T_1, \ldots, T_k)$ ein Baum der Höhe $\mathrm{h}(T) := 1 + \max\{\mathrm{h}(T_1), \ldots, \mathrm{h}(T_k)\}$ mit Wurzel ℓ (ℓ ist ein *innerer Knoten* und *direkter Vorgänger der Wurzelknoten von* $T_1, \ldots, T_k$).

(A) Nichts sonst ist ein Baum.

Beispiele für Bäume mit Verzweigungsgrad 2 sind A, $(\mathsf{E}, \mathsf{A}, (\mathsf{D}, \mathsf{B}, \mathsf{C}))$ und $(\mathsf{E}, (\mathsf{D}, \mathsf{B}, \mathsf{C}), (\mathsf{D}, \mathsf{B}, \mathsf{C}))$. Analog zu Bäumen aus Termen (siehe Folie 6-5), lassen sich Bäume graphisch darstellen:



Als *Tiefe* eines Knotens v in einem Baum T bezeichnet man die Anzahl von „übergeordneten Knoten“ von v, d.h. die Anzahl von Vorgängern. Ein Baum T heißt *vollständig*, falls alle Blätter in T die gleiche Tiefe haben. Im obigen Beispiel hat jeder mit B beschriftete Knoten die Tiefe 2 und jeder mit D beschriftete Knoten die Tiefe 1. Der erste und der letzte Baum sind vollständig.

Seien beliebige natürliche Zahlen $h > 0, k > 2$ gegeben.

a) Wie viele Blätter hat ein *vollständiger* Baum der Höhe h mit Verzweigungsgrad k?

b) Wie viele innere Knoten hat ein *vollständiger* Baum der Höhe h mit Verzweigungsgrad k?

c) Wie viele Knoten (innere und äußere) hat ein beliebiger Baum der Höhe h mit Verzweigungsgrad k *maximal*?

Beweisen Sie die von Ihnen aufgestellten Behauptungen.

(12 Punkte)

Traversierung von Baümen: Betrachten Sie folgenden Baum mit Verzweigungsgrad 3:

(1,(2,A,B,C),(2,(3,E,F,(4,G,H,I)),B,C),(2,A,(3,E,F,G),C))

Bezeichne für jeden Baum der Höhe $h > 1$ mit Wurzel W

- L den linken Unterbaum,
- M den mittleren Unterbaum
- und R den rechten Unterbaum.

a) Geben Sie in Analogie zur *Preorder*-Traversierung aus der Vorlesung die Ausgabe der Baumtraversierung gemäß der Reihenfolge $W - L - M - R$ an.

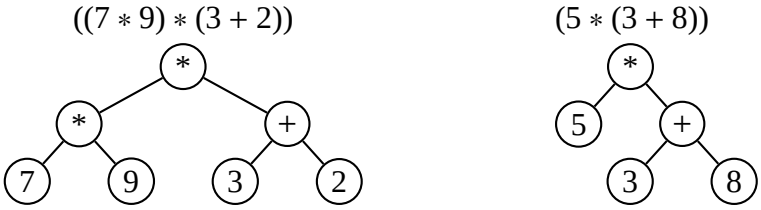
b) Geben Sie in Analogie zur *Inorder*-Traversierung aus der Vorlesung die Ausgabe der Baumtraversierung gemäß der Reihenfolge $L - M - W - R$ an.

c) Geben Sie in Analogie zur *Postorder*-Traversierung aus der Vorlesung die Ausgabe der Baumtraversierung gemäß der Reihenfolge $L - M - R - W$ an.

(6 Punkte)

2 Binäre Bäume zur Auswertung arithmethischer Ausdrücke

Binäre Bäume können auch zur Dartellung von vollständig geklammerten arithmetischen Ausdrücken verwendet werden wie in folgendem Beispiel veranschaulicht:



Hierbei handelt es sich um Binärbäume mit zwei Knotentypen:

- `Operator`: Ein arithmetischer Operator; in dieser Übung beschränken wir uns auf die Operatoren Multiplikation (**`*`**) und Addition (**`+`**).

- **Operand**: Ein Zahlenwert; in unserem Fall sollen nur ganzzahlige Werte (Integers) betrachtet werden.

Die Auswertung eines arithmetischen Ausdrucks in Baumform ist rekursiv definiert:

- Der Wert eines *Operanden*-Knotens entspricht dem Zahlenwert der in diesem Knoten gespeicherten Zahl (Beispiel: 7 für den am weitesten links stehenden Blatt-Knoten in der Abbildung oben).
- Der Wert eines *Operator*-Knotens entspricht dem Wert seines linken Teilbaumes verknüpft mit dem Wert seines rechten Teilbaumes über den Operator, der in diesem Knoten gespeichert ist (Beispiel: $7 * 9 = 63$ für den am weitesten links stehenden Operator-Knoten in der Abbildung oben).

Man kann sich leicht veranschaulichen, dass bei der Auswertung eines Ausdrucks die Werte in einer Art "bottom-up"-Prozedur von den Blättern nach oben propagiert werden, bis schliesslich im Wurzelknoten der Wert des gesamten Ausdrucks steht.

Datenstruktur: Programmieren Sie eine Klasse **ArithmeticTree**, um die oben beschriebene Datenstruktur eines Binärbaumes zur Repräsentation arithmetischer Ausdrücke abzubilden. Der **ArithmeticTree** soll hierbei Knoten des Typs **ArithmeticTreeNode** besitzen. In diesen Knoten sollen entweder ein Operator oder ein Operand gespeichert werden. Programmieren Sie hierzu zwei Konstruktoren

1. **ArithmeticTreeNode(String operator)** sowie
2. **ArithmeticTreeNode(Integer operand)**

In ersterem Fall soll ein Operator-Knoten erzeugt werden, in letzterem ein Operanden-Knoten. Alle übrigen Funktionen sind analog zur Definition eines binären Baums in Java von Folie 6-10. Die Klasse **ArithmeticTreeTest.java** von der Vorlesungsseite enhält Code, der einen kleinen Beispielbaum in dieser Datenstruktur erzeugt.

Die Klasse **ArithmeticTree** selbst soll einen Konstruktur haben, der einen leeren Baum erzeugt (**ArithmeticTree()**) sowie einen Konstruktor, der einen arithmetischen Baum mit einem gegebenen Wurzelknoten erzeugt
(**ArithmeticTree(ArithmeticTreeNode root)**).

(6 Punkte)

Ausgabe in Infix / Postfix Notation, Auswertung des Ausdrucks: Fügen Sie der Klasse **ArithmeticTree** folgende Funktionen hinzu:

- **Integer evaluate()**: Auswertung des arithmetischen Ausdrucks, der in diesem Baum abgespeichert ist. Der Rückgabewert soll dem Wert des arithmetischen Ausdrucks entsprechen.

- **`void printInfix()`**: Bildschirmausgabe des arithmetischen Ausdrucks, der in diesem Baum gespeichert ist, in vollständig geklammerter Infix-Notation (wie oben in der Abbildung). Die Reihenfolge der Ausgabe soll hierbei jeweils **`(operand1 operator operand2)`** sein.
- **`void printPostfix()`**: Bildschirmausgabe des arithmetischen Ausdrucks in Postfix-Notation (siehe Hausübung 6 für die Definition der Postfix-Notation).

Die Ausführung der Klasse **`ArithmeticTreeTest`** von der Vorlesungsseite sollte eine Ausgabe in folgendem Format erzeugen:

```
Postfix-Notation:
 2 2 5 4 * + *
Infix-Notation:
(2*(2+(5*4)))
Wert des Ausdrucks:
44
```

(12 Punkte)

ZUSATZ: Parsen von Postfix-Notation: Schreiben Sie eine Klasse **`PostfixParser`**, die arithmetische Ausdrücke in Postfix-Notation parst und in einem arithmetischen Binärbaum abspeichert. Die Klasse **`TreeCalculator.java`** von der Vorlesungsseite enthält ein Gerüst, um die Postfix-Ausdrücke von Hausübung 6 einzulesen und diese in Postfix- und Infix-Notation auszugeben sowie ihren Wert anzugeben. Sie sollte folgende Ausgabe erzeugen (beispielhaft für die ersten beiden Ausdrücke):

```
Parsing expression 1
Postfix-Notation:
 5 5 +
Infix-Notation:
(5+5)
Wert des Ausdrucks:
10

Parsing expression 2
Postfix-Notation:
 7 3 5 6 * + *
Infix-Notation:
(7*(3+(5*6)))
Wert des Ausdrucks:
231
```

(12 Punkte)

Auf Fehlerbehandlung kann in allen Teilaufgaben verzichtet werden, d.h. Sie brauchen nicht zu prüfen, ob ihre Datenstruktur einen gültigen arithmetischen Ausdruck enthält oder nicht.
Hinweis: Die Zusatz-Programmieraufgabe ist *keine* verpflichtende Aufgabe, durch ihre Bearbeitung können allerdings fehlende Punkte bei anderen Aufgaben von diesem Blatt ausgeglichen werden. Insgesamt können auf diesem Aufgabenblatt 36 Punkte erreicht werden; wie Sich diese Punkte zusammensetzen, bleibt Ihnen überlassen.

Viel Spaß und viel Erfolg!