Technische Hochschule Brandenburg

Komplexpraktikum Sommersemester 2023

Prof. Dr. Sven Buchholz

Aufgabenstellung

Wie üblich kürzen wir einen Binären Suchbaum mit BST (für binary search tree) ab.

Jeder Knoten eines Binarbaums und somit auch eines BSTs ist ein Objekt der Klasse `Node`.

```
public class Node {
    NodeData data;
    Node left, right;
}
```

Das Datum eines Knotens ist dabei allgemein als vom Datentyp `NodeData` definiert. Wir werden im Versuch nur mit Worten (Strings) arbeiten.

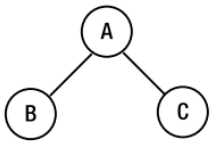
```
public class NodeData {
    String w;
}
```

Bäume werden durch die Klasse `Tree` beschrieben.

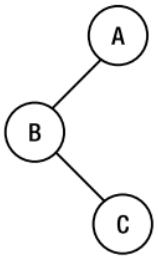
```
public class Tree {
    Node root;
}
```

Aufgabe 1

Schreiben Sie für `Tree` eine Klassenmethode `buildTree`, die mit Hilfe der Scanner-Klasse aus der Java API Eingaben von der Konsole liest, und daraus einen Baum aufbaut. Wir beginnen mit dem Datum für die Wurzel, und jedem Knoten folgt dann sein linker und dann sein rechter Unterbaum. Falls ein Unterbaum leer ist, wird dies durch das Zeichen @ ausgedrückt. Ein Baum mit nur einem Knoten mit dem Datum A ist so also durch A @ @ gegeben. Der Baum



wird durch A B @ @ C @ @ erzeugt, der Baum



hingegen durch A B @ C @ @.

Aufgabe 2

Implementieren Sie in der Klasse `Tree` die folgenden drei Methoden `preOrderTraversal`, `inOrderTraversal` und `postOrderTraversal` für die entsprechenden Traversierungen. Wenn ein Knoten `node` besucht wird, soll für diesen `node.data.visit()` aufgerufen werden. Erweitern Sie daher `NodeData` um eine Methode `visit`, die das Datum des Knoten auf der Konsole ausgibt. Schreiben Sie einen Testtreiber `TesttreiberAufgabe2.java`, der für einen Baum jede Traversierung aufruft und das Ergebnis auf der Konsole ausgibt. Eine Ausgabe des Testtreibers könnte zum Beispiel so aussehen:

```
The pre-order traversal is: C E F H B G A N J K

The in-order traversal is: F H E B C A G J N K

The post-order traversal is: H F B E A J K N G C
```

Aufgabe 3

In dieser Aufgabe geht es nun schließlich tatsächlich um BSTs. Erweitern Sie die Klasse `Tree` um eine Methode `findOrInsert`. Die Methode sucht in einem Baum für ein Datum `d` vom Typ `NodeData` nach einem Knoten mit diesem Datum. Dabei wird der Baum als BST angenommen. Wird das Datum im Baum gefunden, so gibt die Methode den zugehörigen Knoten zurück. Falls nicht, fügt Sie einen neuen Knoten mit Datum `d` so ein, dass die BST Eigenschaft erhalten bleibt und gibt diesen zurück. Ein binärer Baum ist ein BST falls für jeden seiner Knoten gilt: jedes Datum in seinem linken Unterbaum ist kleiner als sein Datum, und jedes im rechten Unterbaum größer. Für Strings ist dabei die lexikographische Ordnung gemeint.

Aufgabe 4

Benutzen und erweitern Sie alle Ihre bisherigen Klassen nun, um die folgende Aufgabe zu lösen. Gegeben ist ein Textfile. Wir nehmen an, dass es nur Buchstaben und Leerzeichen enthält. Ein Wort darin ist definiert als eine Folge von Buchstaben. Wir nehmen der Einfachheit halber an, dass alle Wörter klein geschrieben sind. Schreiben Sie ein Programm `TesttreiberAufgabe4.java` , dass beim Aufruf ein solches Files als Argument bekommt, dieses scannt, einen BST für die gefunden Wörter aufbaut und dabei das Vorkommen eines jeden Wortes mitzählt. Ihr Programm soll dann den Baum mittels `inOrderTraversal` durchlaufen, und dabei jedes Wort mit seiner Häufigkeit ausgeben.

Abgaben

bitte alles am Tag des Versuches bis spätestens Ende der Versuchszeit in Moodle hochladen, also: `TesttreiberAufgabe2.java`, `TesttreiberAufgabe4.java` und die finalen Versionen der Klassen `Tree`, `NodeData` und `Node`.