Binärbaum

Aufgabe 1: Konstruktion eines Binärbaumes

1. Konstruieren Sie einen vollständigen, geordneten Binärbaum (auf Papier), der die Werte 4, 6, 7, 8, 10, 12, 14 und 17 enthält.
2. Überlegen Sie sich, wie Sie in welcher Reihenfolge Sie die obigen Werte in den Baum einfügen müssen, damit ein möglichst ausgeglichener Baum entsteht.

Diese Aufgabe 1 muss nicht abgegeben werden.

Aufgabe 2: Traversierung von Bäumen

Sie bekommen die Datenstruktur eines Binärbaumes in Java vorgegeben. Erweitern Sie diese um die folgenden Traversierungsmethoden:

- preOrder
- inOrder
- postOrder
- levelOrder

Hinweis:
Verwenden Sie für die levelorder-Methode (siehe Klasse TreeTraversal) die Queue-Implementation des JDKs.

Aufgabe 3: Rangliste mit binärem Suchbaum

Die Verwaltung von Datensätzen mittels Listen ist nur bis zu einer bestimmte Anzahl Elemente effizient genug, da zum Finden und geordneten Einfügen eines Elements im Schnitt über die Hälfte der Elemente hinweg iteriert werden muss (O(n)). Aus diesem Grund werden bei grösseren Datenmengen effizientere Datenstrukturen wie zum Beispiel sortierte Binärbäume vorgezogen. Sortierte Binärbäume haben die Eigenschaft, dass Elemente im Mittel in Log_2 Schritten eingefügt und wieder gefunden werden können (O(log(n)).

Es soll nun die Ranglisten-Aufgabe nochmals mittels eines sortierten Binärbaumes einer echten Teilnehmerliste gelöst werden.

Aufgabe 4: Bestimmen der Höhe und Grösse des Baumes

Implementieren Sie die Methode height und size des SortedBinaryTree mittels der Sie die Höhe des Baumes und die Anzahl Knoten bestimmen können (als rekursive Methoden). Bestimmen Sie die Höhe des Baumes bei der Rangliste. Was fällt auf?

Aufgabe 5 Ausschnitt der Elemente aus einem binären Suchbaum

Eine weitere Aufgabe, die mit Bäumen elegant gelöst werden kann, ist, alle Elemente zu finden, die innerhalb eines vorgegebenen Intervalls liegen. Man könnte zwar auch den ganzen Baum traversieren und alle Elemente überprüfen (filtern), was aber bei grossen Bäumen nicht effizient genug ist. Erweitern Sie Ihre Traversal-/Interface & Klasse um eine optimierte Methode interval(Comparable<T> min, Comparable<T> max, Visitor<T> v), die nur die Knoten besucht, die innerhalb des angegebenen Intervalls liegen.Ergänzen Sie hierzu die Methode intervall aus in Ihrer TreeTraversal-Klasse.

Hinweis:
Bei der Interval-Traversierung kann entschieden werden, in welchem Teilbaum sich sicherlich keine Knoten des angegebenen Intervalls mehr befinden, d.h. ob es sich z.B. im untenstehenden Beispiel lohnt, den linken Teilbaum des Knotens C zu traversieren.