Zürcher Hochschule für Angewandte Wissenschaften

Algorithmen und Datenstrukturen

Bäume

Aufgabe 1: Konstruktion eines Binärbaumes [2 Punkt]

a) Konstruieren Sie einen vollen geordneten Binärbaum (auf Papier), der die Werte 4,6,7,8,10,12,14,17 enthält.

b) Überlegen Sie sich, wie Sie in welcher Reihenfolge Sie die obigen Werte in den Baum einfügen müssen, damit ein möglichst ausgeglichener Baum entsteht.

Aufgabe 2: Traversierung von Bäumen [2 Punkte]

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 die Queue Implementation des JDKs

Abgabe

Praktikum: ADS5.2

Filename: TreeTraversal.java

Aufgabe 3: Rangliste mit binärem Suchbaum [2 Punkte]

Die Verwaltung von Datensätzen mittels Listen ist nur bis zu einer bestimmten Anzahl Elemente effizient, da zum 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 werden können.

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

Abgabe

Praktikum: ADS5.3

Filename: RankingTreeServer.java

Aufgabe 4: Bestimmen der Höhe und Grösse des Baumes [2 Punkte]

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?

Abgabe

Praktikum: ADS5.4

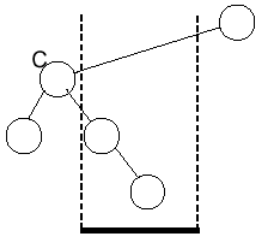
Filename: SortedBinaryTree.java

Aufgabe 5 Ausschnitt der Elemente aus einem binären Suchbaum [2 Punkte]

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, i.e. den Aufwand O(n) hätte. Erweitern Sie Ihre Traversal-/Interface & Klasse um eine optimierte Methode interval(T min, T max, Visitor<T> v), die nur die Knoten besucht, die innerhalb des angegebenen Intervalls liegen.
Aufgabe: ergänzen Sie 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 unten stehenden Beispiel lohnt, den linken Teilbaum des Knotens C zu traversieren.



Abgabe

Praktikum: ADS5.5

Filename: TreeTraversal.java