OPERATIONS RESEARCH
KACKERTSTR. 7
52072 AACHEN

Prof. Dr. Marco Lübbecke
Dipl.-Math. oec. Michael Bastubbe
Christian Puchert, M.Sc.
Annika Thome, M.Sc.

Johannes Gocke
Dustin Hütter
Sascha Kurowski

Programmieren, Algorithmen, Datenstrukturen
9. Programmieraufgabe

Abnahme bis spätestens 24.01.2014

AVL-Bäume

Diese Aufgabenstellung wird euch im Vergleich zu den vorherigen sehr knapp vorkommen. Ihr sollt dadurch lernen, noch selbstständiger zu programmieren und euch mehr eigene Gedanken zum Konzept und Design von Programmen zu machen. Schaut euch aber bevor ihr losprogrammiert in jedem Fall nochmal die Folien der Übung vom 16.01.2014 an - dort findet ihr wichtige Hinweise und nützliche Tipps zur Implementation von Suchbäumen.

Programmiert zunächst einen unbalancierten binären Suchbaum und dann einen AVL-Baum und testet ihre Korrektheit und Performance mit Hilfe des nicht von euch zu verändernden Testprogramms in

`SearchTreeTest.java`.

Eure Baumklassen müssen dafür `padTree.SearchTree` und `padTree.AVLTree` heißen und beide das in

`TraversableSortedContainer.java`

definierte Interface `padInterfaces.TraversableSortedContainer` implementieren. Die beiden genannten Dateien stehen im Lernraum als `PA09.zip` zum Download bereit.

Die einzige weitere explizit geforderte Funktionalität ist eine öffentliche Methode `checkBalances` in der Klasse `AVLTree`, die die Balancen im gesamten Baum überprüft und nur dann `true` zurückgibt, wenn der Check fehlerfrei verlaufen ist. Diese Methode wird auch vom Testprogramm aufgerufen. Denkt beim Programmieren aber in eurem eigenen Interesse auch an die Integration weiterer Mechanismen zur Fehlersuche, z. B. sinnvolle `toString`-Methoden in allen Klassen.

Desweiteren erwarten wir von euch, dass euer Design allen in der PAD erlernten Prinzipien der schönen Programmierung entspricht. Insbesondere heißt das:

- Der intelligente Einsatz von Vererbung sorgt dafür, dass keine gleichen oder sehr ähnlichen Vorgänge oder Daten mehrfach implementiert werden.
- Generics werden überall dort sauber verwendet, wo es für die Typsicherheit sinnvoll ist. Die einzigen so genannten `[unchecked]`-Compiler-Warnungen, die wir bei der Abnahme akzeptieren werden, sind solche, die beim Casten zwischen einem Typ `T` und dem zugehörigen Interface `Comparable<T>` auftreten.

- Alle Funktionalitäten sind in Hinblick auf bestmögliche Performance wohl durchdacht. Insbesondere müssen alle Operationen auf Datenstrukturen auch wirklich die in der Vorlesung gezeigte Komplexität aufweisen.
- Euer Programm weist eine sinnvolle Package-Struktur auf: Interfaces liegen getrennt von Bäumen und die einzige Klasse außerhalb eines Packages (also im Default-Package) ist das Testprogramm.

Das Testprogramm sollte übrigens schon kompilieren und für den unbalancierten Suchbaum mit passenden Optionen auch funktionieren, wenn eure AVL-Baum-Klasse nur mit den notwendigen Dummy-Methoden (sprich ohne bzw. mit minimalen Rümpfen) ausgestattet ist.

Die Performance eurer Bäume könnt Ihr mit dem Testprogramm nur ohne die Debug-Option richtig testen. Dann sollte für 50000 Elemente und je 100 Zugriffe der AVL-Baum in etwa doppelt so schnell sein wie der unbalancierte Suchbaum (ca. 10s vs. ca. 20s). Im Debug-Modus machen die diversen Tests das Programm für beide Baumarten gleichermaßen langsam.

Sehr wichtig: Bei dieser Programmieraufgabe kommt es mehr denn je darauf an, dass ihr euch eures Designs sicher seid, bevor ihr anfangt es zu implementieren! Räumt jegliche Zweifel und Unklarheiten rechtzeitig aus: Zuerst durch selbstständiges kritisches Begutachten eurer eigenen Arbeit, dann durch den Austausch von Ideen und Erfahrungen mit anderen Gruppen, schließlich durch Fragen in Tutorien oder Rechnerbetreuung und durch Fragen im Forum.

Also, alles rauf auf die Bäume und nicht die Balance verlieren...
Viel Spaß und Erfolg!