Algorithmen und Datenstrukturen INF3/ICS3

Wintersemester 2023/24

Prof. Dr. Georg Schied

Aufgabenblatt 8

Abgabetermin: Mo. 11. Dezember 2023, 23:59 Uhr

Zum Bestehen müssen 11 von 22 Punkten erreicht werden.

Aufgabe 8.1

Können **verkettete Listen** effizient **vergleichsbasiert sortiert** werden, ohne die Elemente in ein Array zu kopieren und ohne dabei neue Listenknoten zu erzeugen (d.h. nur durch Umhängen der Knoten)? Welche der für Arrays vorgestellten effizienten Sortieralgorithmen lassen sich gut auf verkettete Listen übertragen?

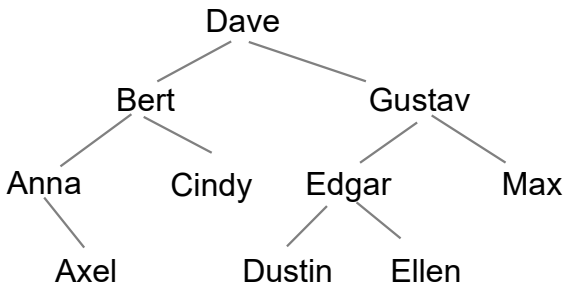
Aufgabe 8.2

a) Geben Sie alle Suchbäume an, die für die Wertemenge {1, 2, 3} möglich sind.

b) Binäre Suchbäume sollen dazu eingesetzt werden, um Artikelbezeichnungen zu speichern. Als Ordnungskriterium wird die übliche lexikographische Sortierung verwendet. Fügen Sie nacheinander folgende Bezeichnungen in einen am Anfang leeren Suchbaum ein.

`Datteln`, `Gurke`, `Brot`, `Ananas`, `Marmelade`, `Eis`, `Dosenmilch`, `Axt`, `Ente`, `Chili`

c) Löschen Sie `Anna`, `Dave` und `Ellen` aus folgendem Baum:

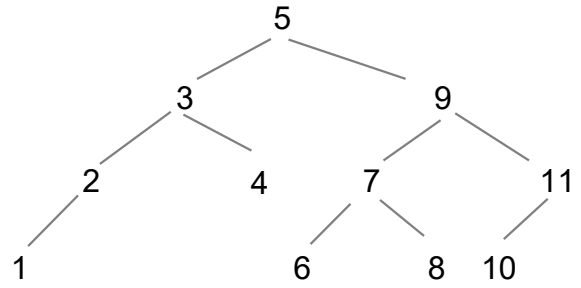


Kommentieren Sie kurz die Vorgehensweise.

Aufgabe 8.3 - Scheinaufgabe (4 P)

a) Fügen Sie die Zeichen G, L, H, S, C, R, M, E, J, B, U, F, D nacheinander in einen zu Beginn leeren binären Suchbaum ein.

b) Löschen Sie nacheinander die Werte 2 und 9 aus dem folgenden Suchbaum:



Aufgabe 8.4 - Scheinaufgabe (12 P)

In Moodle finden Sie die Klassen `SearchTree` als Implementierung eines binären Suchbaums für Werte vom Typ `double`.

a) Erweitern Sie die Klasse `SearchTree` um folgende Methoden:

`public double` `prodLeaves()`

Berechnet das Produkt der Werte aller *Blätter* des Baums. Der leere Baum soll den Wert 1.0 liefern.

`public double` `extractMin()`

Entfernt den Knoten mit dem minimalen Wert aus dem Baum und gibt dessen Wert zurück. Liefert eine `RuntimeException`, falls der Baum leer ist.

`public` `ArrayList<Double> toSortedList()`

Liefert die im Baum gespeicherten Werte *aufsteigen sortiert* als `ArrayList` (Paket `java.util`) zurück.

`public boolean` `equals(SearchTree other)`

Prüft, ob der Baum `other` genau die gleichen Werte enthält, unabhängig von der Struktur des Baums. Die Prüfung sollte effizient implementiert sein.

In Moodle finden Sie dazu die Klasse `SearchTree` sowie eine JUnit-Testklasse `JuTestSearchTree`.

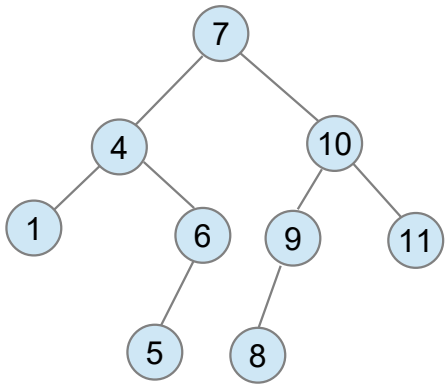
b) Welche Laufzeitkomplexität hat Ihre Operation `equals` im mittleren und schlechtesten Fall? Gehen Sie bei dabei davon aus, dass beide Bäume jeweils *n* Werte enthalten.

c) Programm `SearchTreeMeasurement` (siehe Moodle), misst die Laufzeiten für `equals()` bei verschiedenen Problemgrößen mit zufällig generierten Daten. Messen Sie die Laufzeiten für Ihre Implementierung und erläutern Sie, ob die erwartete Laufzeitkomplexität bei den gemessenen Zeiten erkennbar ist (starten Sie das Programm mit Option `-Xint`).

Aufgabe 8.5

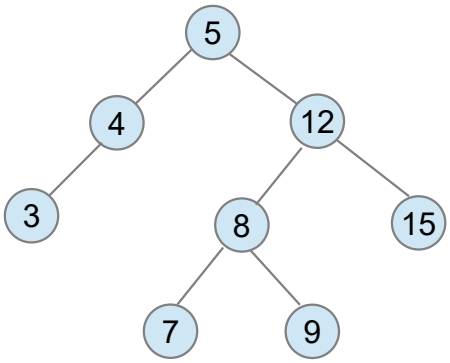
a) Fügen Sie in einen am Anfang leeren **AVL-Baum** nacheinander die Werte
1, 4, 7, 10, 8, 6, 2, 5
ein. Geben Sie den Baum nach jeder Einfügeoperation an und erläutern Sie, welche Rotationen dabei zur Restrukturierung verwendet werden.

b) Löschen Sie nacheinander die Werte 4, 1 und 11 aus dem folgenden AVL-Baum.



Aufgabe 8.6 - Scheinaufgabe (6 P)

a) Fügen Sie die Werte 2 und 10 in den folgenden **AVL-Baum** ein. Stellen Sie den Baum nach jeder Einfügeoperation dar und geben Sie an, welche Rotationen dabei ggf. durchgeführt werden.



b) Löschen Sie den Wert 5 aus folgendem AVL-Baum. Kommentieren Sie dabei kurz die Vorgehensweise.