TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK

Lehrstuhl für Sprachen und Beschreibungsstrukturen
Grundlagen: Algorithmen und Datenstrukturen
Prof. Dr. Helmut Seidl, J. Kranz, A. Reuss, R. Vogler

SS 2016
Übungsblatt 8
04.06.2016

Abgabe: **12.06.2016** (bis 23:59 Uhr)

Wiederholung einiger Definitionen: Ein Knoten eines gewurzelten Baumes ist ein Blatt, wenn er keine Kinder hat. Andernfalls ist er ein innerer Knoten.

(In der Literatur wird ein gewurzelter Baum, der nur aus der Wurzel besteht, nicht einheitlich gehandhabt - in einigen Fällen ist dieser einzelne Knoten per Definition ein innerer Knoten, in anderen Fällen ein Blatt, und in wieder anderer Literatur ein Spezialfall, der weder innerer Knoten, noch Blatt ist. Bei unserer Definition ist ein solcher Knoten ein Blatt.)

Die Tiefe eines Knotens ist die Länge des (eindeutigen) Pfades von dem Knoten zur Wurzel, gemessen in der Zahl der Kanten des Pfades. D.h. insbesondere, dass die Wurzel Tiefe 0 hat. (Dies ist nicht einheitlich in der Literatur: in einigen Fällen wird in der Zahl der Knoten des Pfades gemessen, was dann gerade 1 mehr ist als die Zahl der Kanten).

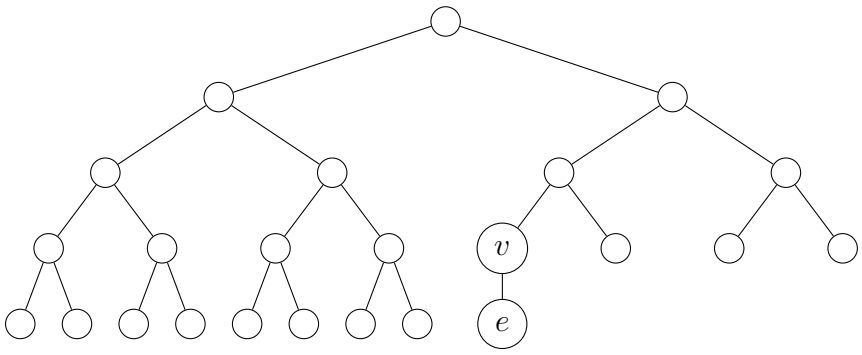
Die Baumtiefe eines gewurzelten Baumes ist das Maximum der Tiefen aller enthaltenen Knoten. Das i-te Level eines gewurzelten Baumes besteht aus allen Knoten, die Tiefe i besitzen.

Ein gewurzelter Baum ist ein Binärbaum, wenn jeder innere Knoten maximal zwei Kinder hat. Wenn jeder innere Knoten genau zwei Kinder hat, ist er ein echter Binärbaum. Ein Binärbaum ist vollständig, wenn er ein echter Binärbaum ist und alle Blätter dieselbe Tiefe haben.

Sei t die Tiefe eines Binärbaumes T. Der Baum T ist ein fast vollständiger Binärbaum, wenn T entweder nur aus der Wurzel besteht oder wenn die ersten $t-1$ Level gemeinsam einen vollständigen Binärbaum bilden, und die Knoten von T so angeordnet werden können, dass es auf dem Level t einen Knoten e mit dem Vaterknoten v gibt, sodass gilt:

- Alle Knoten auf dem Level $t-1$, die „links“ von v stehen, haben zwei Kinder.
- Alle Knoten auf dem Level $t-1$, die „rechts“ von v stehen, haben keine Kinder.

Die folgende Abbildung zeigt einen fast vollständigen Binärbaum mit Baumtiefe 4.



Aufgabe 8.1 (P) AVL-Bebaumung

Gegeben sei ein AVL-Baum der nur aus einem Knoten mit Schlüssel 10 besteht. Fügen Sie nacheinander die Schlüssel 5, 17, 3, 1, 4 ein. Löschen Sie dann den Schlüssel 4, und fügen Sie dann die Schlüssel 8, 2, 7, 6, 9 ein. Löschen Sie dann die Knoten mit den Schlüsseln 2, 1, 8. Zeichnen Sie den AVL-Baum für jede Einfüge- bzw. Löschoperation und geben Sie an, ob Sie keine, eine Einfach- oder Doppelrotation durchgeführt haben.

Aufgabe 8.2 (P) Fast vollständige Birnbäume

In der Vorlesung wurde die folgende Aussage verwendet: Jeder fast vollständige Binärbaum mit $n \geq 1$ Knoten und Baumtiefe $t \geq 0$ erfüllt $2^t \leq n \leq 2^{t+1} - 1$.

Beweisen Sie diese Aussage.

Aufgabe 8.3 (P) Zusatzaufgabe

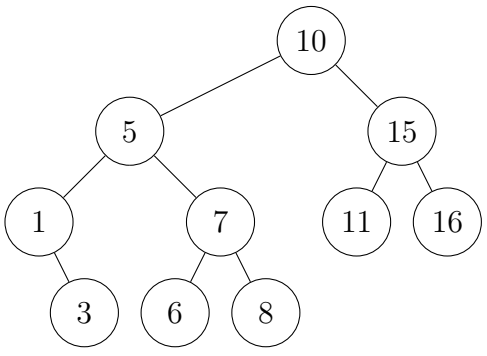
Wir betrachten die Anzahl der Blocktransfers, die beim externen Sortieren auftreten. Die Komplexität ist in der Vorlesung mit $\mathcal{O}(\frac{n}{B} \log_{M/B} \frac{n}{M})$ hergeleitet worden. Hierbei ist M die Größe des internen Speichers, B die Größe eines Blocks und n die Zahl der zu sortierenden Elemente im externen Speicher. Man kann für externes Sortieren zeigen, dass im Worst-Case $\Omega(\frac{n}{B} \log_{M/B} \frac{n}{B})$ Blocktransfers benötigt werden. Ist damit der in der in der Vorlesung gezeigte Algorithmus asymptotisch optimal?

Aufgabe 8.4 [4 Punkte] **(H) Binärbäume – (nur) echt mit b Blättern**

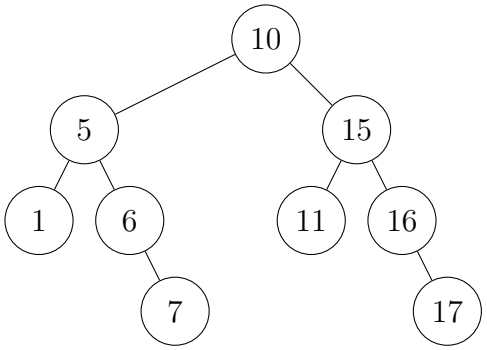
Zeigen Sie, dass es für jedes $b \geq 1$ einen fast vollständigen echten Binärbaum mit b Blättern gibt.

Aufgabe 8.5 [4 Punkte] **(H) AVL-Bäume**

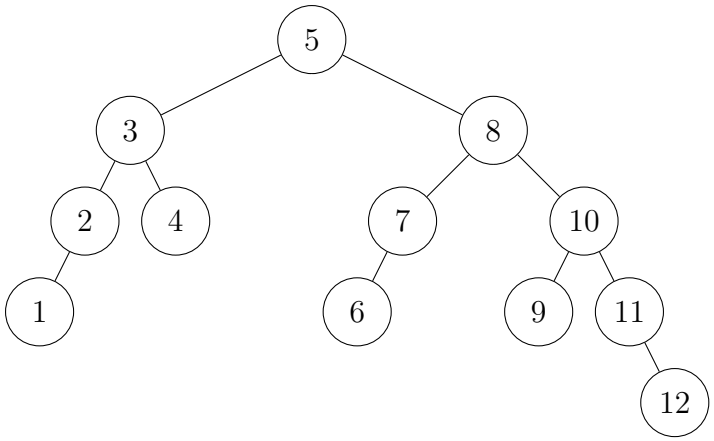
(a) Führen Sie auf dem folgenden AVL-Baum eine `insert`-Operation des Schlüssels 9 durch. Zeichnen Sie den durch die Operation entstehenden AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.



(b) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 10 durch. Zeichnen Sie den durch die Operation entstehenden AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.



(c) Führen Sie auf dem folgenden AVL-Baum eine `remove`-Operation des Schlüssels 4 durch. Zeichnen Sie **für jede dabei durchgeführte Rotation** den durch die Rotation entstehenden Baum.



Aufgabe 8.6 [12 + 3 Punkte] **(H) Programmieraufgabe: Avella**

In dieser Aufgabe geht es darum, einen AVL-Baum zu implementieren. Es soll hier nur der Baum selbst, nicht die komplette Suchstruktur implementiert werden. Im Baum werden nur die Schlüssel abgelegt, auf assoziierte Werte wird verzichtet. Eine Vorlage und Startpunkt finden Sie im Ordner `avella-angabe/`. Kommentare im gegebenen Quelltext enthalten wichtige Hinweise und Beispiele.

Gehen Sie wie folgt vor:

a) Implementieren Sie die Methode `int height()` der Klasse `AVLTreeNode`. Die Methode ermittelt die Höhe des Teilbaums, an dessen Wurzel die Methode aufgerufen wird.

b) Implementieren Sie die Methode `boolean validAVL()`, die Ihnen beim Debuggen hilft. Sie testet, ob der AVL-Baum alle Forderungen an AVL-Bäume erfüllt. Konkret wird folgendes geprüft:

- Die Höhe des linken Teilbaumes eines Knotens unterscheidet sich von der Höhe des rechten Teilbaumes um höchstens eins.
- Die Schlüssel im linken Teilbaum eines Knotens sind kleiner als der oder gleich dem Schlüssel des Knotens.
- Die Schlüssel im rechten Teilbaum eines Knotens sind größer als der Schlüssel des Knotens.
- Die Balancierung jedes Knotens entspricht der Höhendifferenz der Teilbäume mit der Darstellung aus der Vorlesung.

c) Implementieren Sie die Methode `boolean find(int key)` der Klasse `AVLTree`, die einen Schlüssel im AVL-Baum sucht. Sie gibt zurück, ob der Schlüssel gefunden wurde.

d) Implementieren Sie die Methode `void insert(int key)` der Klasse `AVLTree`, die einen Schlüssel in den AVL-Baum einfügt. Achten Sie darauf, dass der Baum nach dem Einfügen alle Forderungen an AVL-Bäume erfüllt.

e) Implementieren Sie die Methode `boolean remove(int key)` der Klasse `AVLTree`, die einen Schlüssel aus dem AVL-Baum löscht. Achten Sie darauf, dass der Baum nach dem Löschen alle Forderungen an AVL-Bäume erfüllt. Die Methode gibt zurück, ob der Schlüssel im Baum gefunden und gelöscht wurde oder nicht.

Bitte beachten Sie, dass Sie für eine funktionierende Implementierung die Klasse `AVLTree` (und auch die eingebettete Klasse `AVLTreeNode`) gegebenenfalls noch an anderen Stellen als innerhalb der jeweiligen Methoden verändern müssen. Es empfiehlt sich, die Aufgaben in Teilprobleme zu zerlegen und diese in Hilfsmethoden zu implementieren. Sie können für Ihre Lösung annehmen, dass der gleiche Schlüssel nicht mehrfach in den Baum eingefügt wird.

Die Klasse `AVLTree` enthält die Methode `String dot()`, die den Baum in das Graphviz-Format[1] umwandelt. Es ist sehr empfehlenswert, sich die eigenen Graphen z.B. mit Xdot[2] anzusehen, um Fehler besser zu verstehen.

Sie können in dieser Aufgabe insgesamt 3 Bonuspunkte erreichen. Dazu ist es notwendig, sämtliche nötige Rotationen (also zum Einfügen und zum Löschen von Elementen) in einer einzigen Methode `void rebalance(...)` der Klasse `AVLTreeNode` zu implementieren. Die Methode darf keine Code-Duplikation enthalten, darf also fast ausschließlich zwischen Einfach- und Doppelrotation unterscheiden. Den vollen Bonus erhält man, wenn keine sinnvolle Möglichkeit mehr besteht, den Code der Methode durch Ausnutzung von Ähnlichkeit zwischen den verschiedenen Rotationen weiter zu kürzen.

[1] `http://www.graphviz.org/`
[2] `https://github.com/jrfonseca/xdot.py`