TECHNISCHE UNIVERSITÄT MÜNCHEN
FAKULTÄT FÜR INFORMATIK
Lehrstuhl für Sprachen und Beschreibungsstrukturen SS 2015
Grundlagen: Algorithmen und Datenstrukturen **Übungsblatt 11**
Prof. Dr. Helmut Seidl, J. Kranz, A. Reuss, R. Vogler 21.06.15

Abgabe: 29.06.15 (vor 10 Uhr)

Hinweis: In der Vorlesung wurden Suchbäume als sortierte Liste zusammen mit einer Navigationsstruktur (also dem eigentlichen Baum) eingeführt. Der Einfachheit halber besteht bei uns jeder Suchbaum nur aus der Navigationsstruktur.

Aufgabe 11.1 (P) Binomialheap

Führen Sie auf einem anfangs leeren Binomialheap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({15, 20, 9, 1, 11, 4, 13, 17, 42, 7}),
- deleteMin,
- deleteMin,
- deleteMin,
- deleteMin

Die build-Operation ist dabei durch iterative Ausführung von der insert-Operation auf den Elementen realisiert. Die insert-Operation wiederum besteht aus einer merge-Operation auf dem aktuellen, anfangs leeren Binomialheap und einem einelementigen Binomialheap, wobei letzterer das einzufügende Element enthält.

Aufgabe 11.2 (P) AVL-Bäume

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

Aufgabe 11.3 (P) (a,b)-Bäume

Führen Sie auf einem anfangs leeren $(2, 4)$-Baum folgende Operationen aus und zeichnen Sie die Zwischenergebnisse: `insert(23)`, `insert(30)`, `insert(13)`, `insert(6)`, `insert(40)`, `insert(80)`, `insert(62)`, `insert(75)`, `insert(28)`, `insert(21)`, `insert(29)`, `remove(62)`, `remove(75)`, `remove(13)`.

Anmerkung: Zum leichteren Verständnis der Operationen beobachten wir, dass ein Split-Schlüssel stets dem größten (d.h. rechtesten) Blatt entspricht, das sich in dem zu dem Split-Schlüssel korrespondierenden Unterbaum befindet.

Aufgabe 11.4 (P) Zusatzaufgabe

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

Aufgabe 11.5 [5 Punkte] **(H) Binomial-Heap**

Bilden Sie mittels

- build({12,16,24}) und
- build({55,43,42,30,25})

zwei Binomialheaps und führen Sie dann folgende Operationen aus:

- merge der beiden Binomialheaps,
- deleteMin,
- deleteMin,
- deleteMin

Hierbei soll die build-Operation durch mehrfache Ausführung von insert für die Elemente in der genannten Reihenfolge umgesetzt werden. Die insert-Operation ihrerseits besteht aus einer merge-Operation des aktuellen, anfangs leeren Binomialheaps und eines einelementigen Binomialheaps, welcher das einzufügende Element enthält.

Stellen Sie alle Zwischenergebnisse graphisch dar!

Aufgabe 11.6 [5 Punkte] **(H) (a,b)-Bäume**

Führen Sie auf einem anfangs leeren $(2, 3)$-Baum die folgenden Operationen aus:

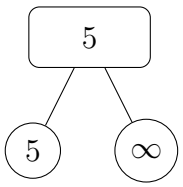
1. `insert(5)`
2. `insert(10)`
3. `insert(6)`
4. `insert(9)`
5. `insert(7)`
6. `insert(8)`
7. `insert(42)`
8. `insert(4)`
9. `insert(3)`
10. `insert(2)`
11. `insert(1)`

und stellen Sie jeweils das Zwischenergebnis graphisch dar. Führen Sie anschließend die folgenden Operationen aus:

1. `remove(6)`
2. `remove(5)`
3. `remove(4)`
4. `remove(8)`
5. `remove(10)`
6. `remove(9)`
7. `remove(1)`

und stellen Sie wieder die Zwischenergebnisse graphisch dar.

Hinweis: Nach der ersten insert-Operation sieht der Baum so aus:



Aufgabe 11.7 [10 + 5 Punkte] **(H) Programmieraufgabe: Avella**

In dieser Aufgabe geht es darum, einen AVL-Baum zu implementieren. Ein Programmgerüst des in Java zu implementierenden Programms wird als separates Archiv mit diesem Übungsblatt verteilt; die Abgabe erfolgt wie üblich über TUMjudge. 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 Werte im linken Teilbaum eines Knotens sind kleiner als der oder gleich dem Wert des Knotens.
- Die Werte im rechten Teilbaum eines Knotens sind größer als der oder gleich dem Wert des Knotens.
- Die Balancierung jedes Knotens entspricht der Höhendifferenz der Teilbäume entsprechend der Vorlesung.

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

d) Implementieren Sie die Methode `void insert(int value)` der Klasse `AVLTree`, die einen Wert 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 value)` der Klasse `AVLTree`, die einen Wert 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 Wert 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 dürfen davon ausgehen, dass keine Duplikate in den Baum eingefügt werden.

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.

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

Sie können in dieser Aufgabe insgesamt 5 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 Parallelen zwischen den verschiedenen Rotationen weiter zu kürzen.