Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Institut für Theoretische Informatik
Peter Widmayer
Thomas Tschager
Antonis Thomas

23. März 2016

Datenstrukturen & Algorithmen Blatt 5 FS 16

Aufgabe 5.1 *AVL-Bäume.*

a) Zeichnen Sie den entstehenden Baum, wenn die Schlüssel $5, 8, 16, 10, 12, 13, 15, 9, 11$ in dieser Reihenfolge in einen initial leeren AVL-Baum eingefügt werden.

b) Zeichnen Sie den entstehenden AVL-Baum, wenn aus dem Baum in a) der Schlüssel 5 gelöscht wird.

c) Gegeben sei ein AVL-Baum T, der eine Menge von Schlüsseln S enthält. Gibt es eine Reihenfolge für die Schlüssel in S, sodass beim Einfügen in einen anfangs leeren Baum keine Rotationen stattfinden und genau T entsteht? Falls ja, beweisen Sie die Aussage, und geben Sie andernfalls ein Gegenbeispiel an.

Aufgabe 5.2 *Erweiterte Suchbäume.*

In dieser Aufgabe soll die Funktionalität eines natürlichen Suchbaums erweitert werden. Wir nehmen an, dass als Schlüssel ganze Zahlen verwaltet werden. Modifizieren Sie den Suchbaum, sodass zusätzlich zum Auffinden einer Zahl die folgenden Anfragen beantwortet werden können.

a) Wie viele Schlüssel im Baum sind *geradzahlig* und kleiner als eine gegebene Zahl k?

b) Wie viele *geradzahlige* Schlüssel im Baum liegen (echt) zwischen gegebenen k_1 und k_2 mit $k_1 < k_2$?

Überlegen Sie, welche zusätzlichen Informationen jeder Knoten speichern soll und wie das Einfügen und Entfernen von Schlüsseln angepasst werden muss, damit die obigen Anfragen effizient beantwortet werden können. Geben Sie auch die Laufzeit aller Operationen an.

Aufgabe 5.3 *Amortisierte Analyse.*

In dieser Aufgabe betrachten wir dynamische Arrays, die je nach Bedarf wachsen können (wie z.B. `java.util.Vector` in der Java-Standardbibliothek). Diese werden vom Anfang her der Reihe nach mit Werten belegt. Sobald mehr als n Elemente gespeichert sind, wird ein neues Array fester Länge $k > n$ erzeugt, die alten Inhalte umkopiert und das neue Element hinzugefügt. Ein Array der Länge k kann in k Zeiteinheiten angelegt werden, und das Kopieren eines Elements erfolgt in konstanter Zeit.

a) Beschreiben Sie, wie Sie k wählen würden, damit jede Einfügeoperation amortisiert konstante Laufzeit hat und das Einfügen von n Elementen also in $\Theta(n)$ Schritten erfolgt. Beweisen Sie durch amortisierte Analyse, dass Ihre Wahl pro Einfügeoperation amortisiert konstante Zeit benötigt.

b) Wir betrachten nun Löschoperationen, bei denen jeweils *das letzte* Element im Array entfernt wird. Bei einer Mischung solcher Einfüge- und Löschoperationen kann es aus Platzgründen sinnvoll sein, das Array auch schrumpfen zu lassen. Beschreiben Sie, wie Sie das Array schrumpfen lassen würden, und zeigen Sie, dass beide Operationen auch gemischt immer in amortisiert konstanter Zeit erfolgen können.

Bitte wenden.

Aufgabe 5.4 *AVL-Bäume (**Programmieraufgabe**).*

In dieser Aufgabe sollen Sie einen AVL-Baum in Java implementieren. Wir stellen auf der Webseite zur Vorlesung eine Vorlage für Ihren Code zur Verfügung. Die Vorlage enthält bereits einen Grossteil der Implementierung und Ihre Aufgabe ist es, folgende Funktionen zu ergänzen:

- `rotateLeft`. Diese Funktion führt eine Rotation nach links durch. Beachten Sie, dass die Funktion `rotateRight` bereits gegeben ist und Ihnen als Vorlage dienen kann.
- `insert`. Diese Funktion soll einen gegebenen Schlüssel an der richtigen Position einfügen und die Funktion zur Rebalancierung des Baumes aufrufen. Falls ein Schlüssel bereits vorhanden ist, soll er nicht erneut eingefügt werden.
- `delete`. Diese Funktion soll einen gegebenen Schlüssel löschen (verwenden Sie dabei den *symmetrischen Nachfolger* falls der zu löschende Schlüssel zwei Kinder hat) und wiederum die Rebalancierung des Baumes veranlassen.

Eingabe Die erste Zeile der Eingabe enthält lediglich die Anzahl der Testfälle t. Danach folgen die t Testfälle, die wie folgt repräsentiert werden: Die erste Zeile enthält die Anzahl der Operationen $n \leq 100$. Danach folgt eine Zeile mit n ganzzahligen Zahlen ungleich 0. Die i-te Zahl x_i soll eingefügt werden wenn $x_i > 0$. Falls $x_i < 0$, soll der Schlüssel $|x_i|$ gelöscht werden. Beispielsweise soll für $x_i = 10$ der Schlüssel 10 eingefügt werden und für $x_j = -10$ derselbe Schlüssel gelöscht werden (falls er im AVL-Baum gespeichert ist).

Ausgabe Für jeden Testfall soll eine Zeile ausgegeben werde mit allen Balancefaktoren der Knoten des AVL-Baumes, d.h. n Zahlen $b_i \in \{-1, 0, 1\}$ für $i = 1, \ldots, n$, sodass b_i den Balancefaktor des i-ten Knoten des Baumes gemäss der Inorder-Reihenfolge entspricht. Benutzen Sie für diese Ausgabe die vorgegebene Funktion `printBalance`.

Beispiel

Eingabe:

```
2
4
7 10 9 1
7
6 8 3 1 9 -6 2
```

Ausgabe:

```
0 -1 -1 0
0 0 0 -1 0
```

Bemerkungen Der Balancefaktor eines Knoten ist definiert als die Höhe seines rechten minus der Höhe seines linken Teilbaumes (siehe Code in der Vorlage).

Für diese Aufgabe gibt es zwei Kategorien von Testfällen, für die insgesamt 100 Punkte vergeben werden:

- **Einfügen** (50 Punkte): Es werden nur Schlüssel zum Baum hinzugefügt.
- **Einfügen-und-Löschen** (50 Punkte): Schlüssel können sowohl hinzugefügt, als auch gelöscht werden.

Abgabe: Am Mittwoch, den 6. April 2016 in Ihrer Übungsgruppe.