Beispielaufgaben zur Midterm-Klausur — Grundzüge von Algorithmen und Datenstrukturen, WS 12/13

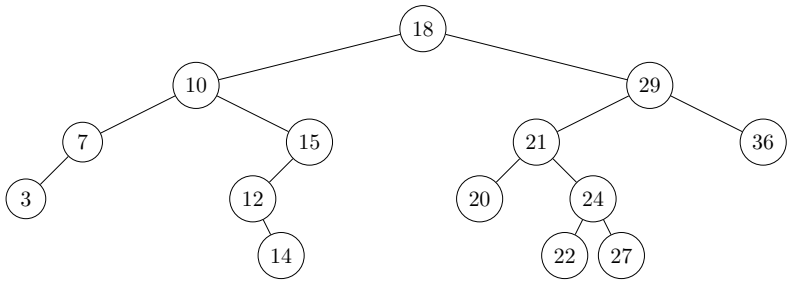
Prof. Dr. Markus Bläser, Radu Curticapean, Christian Engels
http://www-cc.cs.uni-sb.de/course/38/

Abgabe: nie

Sollten Ihnen noch Punkte zur Zulassung fehlen, so können Sie, nach Rücksprache mit Ihrem Bremser, durch eine gute Bearbeitung des Zettels die Zulassung evtl. noch erreichen.

Aufgabe 1. Führen Sie die folgenden Operationen auf dem unten dargestellten binären Suchbaum in der angegebenen Reihenfolge nacheinander aus und zeichnen Sie die dabei entstehenden Bäume:

- 18 löschen,
- 11 einfügen,
- 24 löschen,
- 24 einfügen,
- 7 löschen.



Aufgabe 2. Sei T ein binärer Baum. Zeigen sie, dass folgende Aussagen äquivalent sind:

a) T erfüllt die binäre Suchbaum-Eigenschaft.

b) Inorder-walk(T) erzeugt eine aufsteigend sortierte Sequenz.

Aufgabe 3. Der folgende Algorithmus ist bekannt als "Bubble Sort":

Input: array $a[1..n]$

```
1: Finished ← FALSE
2: while Finished = FALSE do
3:     Finished ← TRUE
4:     for i ← 1 to n − 1 do
5:         if a[i] > a[i + 1] then
6:             swap(a[i], a[i + 1])
7:             Finished ← FALSE
```

Zeigen Sie, dass dieser Algorithmus immer nach einer endlichen Anzahl von Iterationen das Array sortiert hinterlässt. Benutzen Sie dazu eine Invariante. Wie viele Schritte benötigt der Algorithmus im schlimmsten Fall?

Aufgabe 4. Gegeben seien n Arrays mit jeweils k Einträgen. Zeigen Sie, dass man mindestens $\Omega(nk \log k)$ viele Vergleiche braucht, um die n Arrays jeweils zu sortieren. Warum reicht es nicht aus zu argumentieren, dass das Sortieren eines Arrays $\Omega(k \log k)$ viele Vergleich braucht und man n-mal sortieren muss?

Aufgabe 5. Ein Intervall-Baum speichert geschlossene Intervalle der Form $[a, b]$, $a, b \in \mathbb{Q}$, $a \leq b$. Folgende Operationen sollen dabei unterstützt werden:

- Interval_Insert(T, x) – fügt einen Knoten x, der ein Intervall speichert, in den Intervall-Baum T ein.
- Interval_Search$(T, [a, b])$ – liefert entweder einen Zeiger/eine Referenz auf einen Knoten x im Intervallbaum T, so dass das in x gespeicherte Intervall und $[a, b]$ sich überschneiden, oder NULL, wenn es keinen solchen Knoten x in T gibt.

Ziel dieser Aufgabe ist es, binäre Suchbäume so zu erweitern, dass diese Operationen durchgeführt werden können. Dazu speichern wir in jedem Knoten die untere Grenze, low(x), und die obere Grenze, high(x), des Intervalls, das x repräsentiert. Der Schlüssel, nach dem der Baum sortiert wird, ist low(x), die untere Grenze des jeweiligen Intervalls. Außerdem speichern wir in jedem Knoten x den Wert max(x), die größte obere Grenze aller Intervalle, die in $T(x)$, dem Unterbaum mit Wurzel x, gespeichert sind.

(a) Beschreiben Sie, wie man Interval_Insert(T, x) implementieren kann. Begründen Sie, warum das Ergebnis Ihrer Methode wieder ein Intervall-Baum ist wie oben beschrieben.

(b) Betrachten Sie die folgende Methode für Interval_Search.

Input: Intervall-Baum T, Intervall $[a, b]$, $a, b \in \mathbb{Q}$, $a \leq b$.
Output: Knoten x, dessen Intervall sich mit $[a, b]$ überschneidet bzw. NULL, falls es keinen solchen Knoten gibt

```
1: x := root(T)
2: while x ≠ NULL und [a,b] ∩ [low(x), high(x)] = ∅ do
3:     if Left(x) ≠ NULL und max(Left(x)) ≥ a then
4:         x := Left(x)
5:     else
6:         x := Right(x)
   return x
```

Beweisen Sie die folgende Invariante für die Schleife:

> Wenn T ein Intervall $[c, d]$ enthält mit $[c, d] \cap [a, b] \neq \emptyset$, dann gibt es ein solches auch in $T(x)$.

Begründen Sie damit, dass die Methode das angegebene Ausgabeverhalten hat.