

## Probeklausur ALP3 WS 05/06

Achtung dies ist eine Probeklausur. Leiten Sie nicht zuviel über die echte Klausur aus der vorliegenden Aufgabensammlung ab.

90 Punkte == 90 Minuten Zeit

### Verständnisfragen (15 Punkte)

Richtig oder Falsch? Richtige Antworten zählen 1,5 Punkte. Falsche Antworten 1 Punkt Abzug. Die Summe der Punkte dieser Aufgabe kann aber nicht unter 0 sinken.

	Richtig	Falsch
1.) Wird beim Hashing Kollisionsbehandlung mit offener Speicherung betrieben, dann braucht man keine Sondierung nach einem freien Speicherplatz zu betreiben.		
2.) Im Vergleich zwischen sortierten Feld und Hashfeld schneidet letzterer bei der Suche solange besser ab, bis die Belegung 60% nicht übersteigt.		
3.) Im Gegensatz zur Tiefensuche macht es bei der Breitensuche keinen Sinn von Präordnung, Inordnung, Postordnung zu sprechen.		
4.) Vielwegbäume lassen sich als Binärbäume darstellen, welche dann mindestens genauso hoch sind, aber eine kleiner-gleiche Anzahl von Blätter haben		
5.) Ein vollständiger Baum mit Höhe $h$ hat $2^h - 1$ Knoten.		
6.) Ein interner Iterator überlässt dem Aufrufer die Kontrolle über die Traversierung der Datenstruktur.		
7.) Die Abstraktionsfunktion ist eine surjektive Abbildung von Werten der konkreten Repräsentation auf Werte des abstrakten Modells.		
8.) In einer korrekten Repräsentation muss für jedes Element aus der konkreten Wertemenge, welches der konkreten Invariante genügt, auch die Abbildung dieses Elements durch die Abstraktionsfunktion der abstrakten Invariante genügen.		
9.) Durch eine Spezifikation ist genau eine konkrete Repräsentation vorgegeben.		
10.) Durch eine konkrete Repräsentation ist genau eine Implementierung vorgegeben.		

## **Aufgabe Typverträglichkeit, Vererbung, Generizität (20 Punkte)**

Die folgenden Klassen seien vorgegeben:

a) (4 P)

```
class A {
    static int x;

    static void op(int a) {
        System.out.println(x++);
    }
}
```

```
class B extends A {
    static int x;

    void op(char y) {
        System.out.println(x-1);
    }
}
```

```
static void test() {
    B b;
    A a = (b = new B());
    a.x = 6;
    a.op('a');
    b.op('a');
}
}
```

b) (4 P)

```
class C<E> {

    E[] a = new E[8];
    boolean[] b = new boolean[8];
    static class D {
        boolean m(int i) {
            return !b[i];
        }
    }

    void m(byte b) {
        System.out.println(
            (new D()).m(b));
    }

    static void test() {
        C<Float> c = new C<Float>();
        int x = 8;
        c.m(x);
    }
}
```

Entscheiden Sie jeweils, ob es sich hierbei um gültigen Java-(1.5-)Code handelt. Geben Sie an, welche Ausgabe bei Ausführung der Methode test erzeugt wird, bzw. erklären Sie alle statischen Fehler.

(Diese Aufgabe stammt aus der Nachklausur 2005.)

### ***Algebraische Spezifikation (10 Punkte)***

Sei folgende Signaturen von Operationen auf binären Bäumen mit Typparameter T gegeben:

EmptyTree: Tree

Construct: T X Tree X Tree -> Tree

Left: Tree -> Tree

Right: Tree -> Tree

Root: Tree -> T

Height: Tree -> Int

Size: Tree -> Int

Formulieren Sie 10 Axiome als algebraische Spezifikation für diesen ADT.

### **Spezifikation (30 Punkte):**

a.) Eine Multimenge (Bag)  $\{ | t | \}$ , sei durch eine Abbildung repräsentiert:

Modell: `type Bag t = t -> Int`

Invariante: `inv bag = all(\x -> bag x >= 0)t`

Geben Sie eine Abstraktionsfunktion an:

b.) Ein Menge  $\{t\}$  sei durch ein Bitfeld repräsentiert:

```
class IntSet { // { t } (model)
  protected final boolean[] set;
```

Geben Sie eine Abstraktionsfunktion an:

c.) Geben Sie die Invariante für einen AVL-Baum umgangssprachlich an:

## **Baum-Traversierung (15 Punkte)**

Gegeben sei folgender Rot-Schwarz-Baum in zusätzlich gefädelter Darstellung:

```
public class ThreadedRedBlackTree<T extends Comparable<T>> {
class Node {
    private T val;
    private Node l, r;
    private boolean isBlack;
    private boolean rthread = true;
    // true -> "Faden", false -> echtes Kind
    Node(T val) {
        this.val = val;
    }
}
```

```
private Node root;
```

```
public int countIf(Predicate<T> valuePredicate,
Predicate<Boolean> blackPredicate){
```

Implementieren Sie die Methode `countIf`, welche die Anzahl aller Knoten zurückgeben soll, die beide Prädikate (eins bezogen auf den Wert des Knoten und das andere bezogen auf die Farbe) wahr werden lassen.

Um zum Beispiel alle roten Knoten zu zählen, könnte man aufrufen:

```
public static void main(String[] args){
    ThreadedRedBlackTree<Integer> tree =
        new ThreadedRedBlackTree<Integer>();
    tree.count(
        new Predicate<Integer>(){
            public boolean decide(Integer i){return true;}
        },
        new Predicate<Boolean>(){
            public boolean decide(Boolean isBlack){
                return false;
            }
        }));
}
```

### ***O-Kalkül (10 Punkte)\****

Aus einem Feld lässt sich ein Heap „in-place“ erstellen, wenn man mit der rechten Hälfte des Feldes als Teilheaps beginnt und dann Elemente iterativ einfügt (vgl. 8.4.2. Version 2, rearrangieren A, Variante 2). Für jedes Element muss man hierbei seine Einordnungen in den Teilheap korrigieren, wie es auch beim Entfernen von Elementen für das nach oben gesetzte Element der Fall ist („runtertrickeln“). Beweisen Sie, dass diese Operation in  $O(n)$  und nicht in den erwarteten  $O(n \log n)$  durchgeführt werden kann.

\* Extra-Aufgabe für Theoretiker.