

Algorithmen und Programmierung III

WS 04/05

Klausur am 14.2.05, 12:15 – 13:45

Zu erreichende Mindestpunktzahl: 20 (von insgesamt 40)

Aufgabe 1 (4 Punkte, 10 min)

Die folgenden Klassen seien vorgegeben:

```
class Equal<X> {
    boolean equal(X x) { return !unequal(x); }
    boolean unequal(X x) { return !equal(x); }
}
class Num extends Equal<Integer> {
    int value;
    boolean equal(Num n) { return value==n.value; }
}
```

Die Auswertung des Ausdrucks `new Num().equal(0)` führt zu einem Laufzeitfehler. Um welchen Fehler handelt es sich, und was ist die Ursache?

Aufgabe 2 (11 Punkte, 25 min)

Multimengen seien mit Hilfe von binären Suchbäumen wie folgt implementiert:

```
class TreeBag<X extends Comparable<X>> {
    private class Node { X value; Node left, right; ..... }
    private Node root; .....
    public boolean isSet() { ..... }
}
```

Für jeden Knoten im Baum ist sein Wert ungleich `null` und größer *oder gleich* allen Werten im linken Teilbaum und kleiner als alle Werte im rechten Teilbaum. `isSet` ist gesucht. Diese Methode soll Antwort auf die Frage geben, ob eine Menge vorliegt (d.h. keine Duplikate vorkommen). *Wichtige Einschränkungen:* Die Methode soll keine größere Komplexität als $O(n)$ haben. Die Werte sollen *nicht* – etwa in eine andere Repräsentation – kopiert werden; es ist lediglich der Baum geeignet zu analysieren. (Ansonsten kann der obige Code nach Belieben geeignet erweitert werden.)

Aufgabe 3 (9 Punkte, 20 min)

Ein *2-3-Baum* zur Repräsentation einer Menge von `Int`-Werten sei wie folgt vereinbart:

```
data ZDBaum = Empty | T1 ZDBaum Int ZDBaum
              | T2 ZDBaum Int ZDBaum Int ZDBaum
```

(vgl. Vorlesung 7.2.3). Geben Sie die konkrete Invariante (7 Punkte) und die Abstraktionsfunktion (2 Punkte) als Funktionen in (Pseudo-) Haskell an.

Aufgabe 4 (9 Punkte, 20 min)

Gegeben ist

```
class Pair<X extends Comparable<X>, Y>
    implements Comparable<Pair<X,Y>> {
public X first; Y second;
public int compareTo(Pair<X,Y> p){
    return first.compareTo(p.first); }
}
```

Gesucht ist eine Methode `existsGap`, die für eine gegebene Menge von abgeschlossenen Intervallen $[a,b]$, $a,b \in \mathbf{R}$, prüft, ob es zwischen den Intervallen eine Lücke gibt (oder stets zwei benachbarte Intervalle einander überlappen):

```
static boolean existsGap(SortedSet<Pair<Float,Float>> inter)
```

Hinweise: 1) Ein Intervall kann vollständig in einem anderen enthalten sein. 2) Es bietet sich an, mit einem Iterator über `inter` zu arbeiten und `Float.MAX_VALUE` einzusetzen.

Aufgabe 5 (7 Punkte, 15 min)

Um für einen Text sehr schnell zu entscheiden, ob eine gegebene Zeichenkette darin vorkommt, kann man einen *Suffixbaum* aufbauen. Dies ist ein Vielwegbaum, der jedes Suffix des Textes als Folge der Markierungen auf dem Weg von der Wurzel zu einem Blatt enthält.

```
      /  /  |  \
e    n    a    b
    / \    |    |
   e  a  n  a
      | / \    |
      n e  a  n
      |    |  |
      e    n  a
          |  |
          e  n
              |
              e
```

Der Suffixbaum für das Wort "Banane" sieht z.B. so aus.

Ein solcher Suffixbaum sei wie folgt in Java repräsentiert:

```
public class SuffixTree {
    static class Node {
        char c;
        List<Node> subtrees = new Vector<Node>();
    } .....
    private Node root = new Node('.'); // ignored in search
    public boolean contains(String s) { ..... }
}
```

Entwickeln Sie die Methode `contains`, die die Frage beantwortet, ob die gesuchte Zeichenkette in dem zum Suffixbaum gehörigen Text enthalten ist.