Z. Kanaeva

Aufgabenstellungen (Aufgaben 1-6) und Lösungen
Klausur zu ALP III (WS01/02),19. Februar 2002

1. **Aufgabe (10 Min.)** *3 Punkte*
   Welche der Zusicherungen (A - D) kann als Effekt für das folgende Programmfragment gelten:

$$z = z * a;$$
$$y = y - 1;$$

   (A) $a * y == z$

   (B) $z == a^y$

   (C) $const == z * a^y$

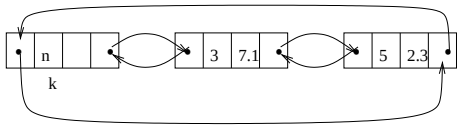
   (D) $y > 0 \wedge z == a^k$

   Begründen Sie bitte kurz Ihre Antwort.

   Lösung: *Die Zusicherung (C) ist die Invariante.*

2. **Aufgabe (35 Min.)** *9 Punkte*
   Ein Vektor v im n-dimensionalen Raum kann als eine Folge von reellen Zahlen dargestellt werden: v in R. In imperativen Programmen wird man dafür in der Regel ein Feld vom Typ float (also float[0 .. n-1]) verwenden.

   Ein dünn besetzter Vektor aus R_n enthält nur weniger von Null verschiedene Elemente. Solche Vektoren als n-elementige Felder zu repräsentieren wäre eine Verschwendung von Speicherplatz. Daher zieht man häufig eine kompaktere Repräsentation vor, bei der nur von Null verschiedene Elemente berücksichtigt werden. Die Idee ist, eine entsprechende Anzahl von Zellen in einem verketteten Ring wie folgt zusammenzufassen:



   Diese doppelt verkettete Ringliste mit Kopf k stellt einen Vektor $(a_0, a_1, \ldots, a_{n-1})$ dar, bei dem $a_3 = 7.1$ und $a_5 = 2.3$ ist, während alle übrigen Komponenten Null sind. Die Zellen sind nach aufsteigenden Indizes angeordnet. Wenn der Vektor nur aus Nullen besteht, besteht die Ringliste nur aus dem Kopf.

   Für die Verwaltung derart repräsentierter Vektoren soll eine java-Klasse mit folgender Spezifikation bereitgestellt werden.

```
class Vector {
  class Element {
    int position;
    float value;
    Element next;
    Element prev;
  }
```

```
  Element head;

  public Vector(int n);
     // requires: -
     // effect: ein Vector-Objekt in n Dimensionen, mit allen
     // Elementen, die 0 sind
  public float get(int i);
     // requires: i ist kleiner als die Anzahl der Dimensionen von v
     // effect: x ist das i-te Element von v, v bleibt unverndert

  public void put(float x,int i);
     // requires: i ist kleiner als die Anzahl der Dimensionen von v
     // effect: das i-te Element von v ist x, alle anderen Elemente
     // bleiben unverndert
}
```

(a) Wie lautet die Repräsentationsinvariante?

(b) Geben Sie bitte die Abstraktionsfunktion an.

(c) Implementieren Sie den Konstruktor der Klasse `Vector`.

(d) Implementieren Sie die Methode `get`.

(e) Implementieren Sie die Methode `put`.

Lösung:

(a) $I \equiv \forall_{e_1,e_2\ in\ Vector \wedge e_1 \neq head \wedge e_1.next==e_2} : e_1.position < e_2.position \wedge e_1.value \neq 0$

(b) kommt noch

(c) Konstruktor:

```
Vector(int n) {
  head = new Element();
  head.position = n;
  head.next = head;
  head.prev = head;
}
```

(d) get-Methode:

```
public float get(int i) {
  Element current = head.next;

  while (current != head && current.position <= i) {
    if (current.position == i) return current.value;
    current = current.next;
  }
  return 0;  // Der einzige Wert (value), der nicht vorkommen darf
}
```

(e) put-Methode:

```
public void put(float x,int i) {
  Element aux = new Element();
  Element current = head.next;

  aux.position = i;
  aux.value = x;
  while (current != head && current.position < i) {
    current = current.next;
  }
  current.prev.next = aux;
  aux.prev          = current.prev;
  aux.next          = current;
  current.prev      = aux;
}
```

3. **Aufgabe (10 Min.)** *3 Punkte*

(a) Angenommen `class B extends A {...}` und `class C extends A {...}`. Welche der Anweisungen ist (sind) falsch?
a) `B b = new A();` **falsch**
b) `A a = new B();`
c) `C a = (C) new B();` **falsch**
d) keine

(b) Gegeben sei eine Klasse `class Animal` und eine Subklasse `class Cat extends Animal`. In beiden wird die Methode `noise()` vereinbart. Welche Methode wird in folgendem Codestück ausgefhrt:

```
Animal s = new Cat();
System.out.println(s.noise());
```

Lösung: *Mit s.noise() führt man die Methode noise() der Klasse Cat aus. Cat ist in dem Fall der dynamische Typ von s.*

(c) Die Klasse `Y` erweitert die Klasse `X`: `class Y extends X {...}`. Welche der folgenden cast-Operationen ist falsch /überflüssig /richtig?
`X x = (X) new Y();` **überflüssig**
`Y y = (Y) x;` **richtig**

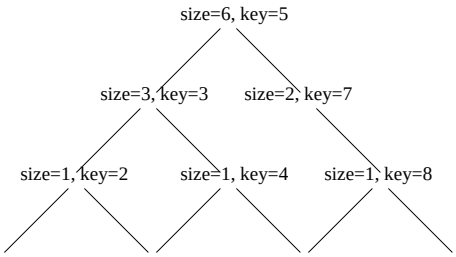
(d) Voraussetzung wie oben

```
Y a = (Y) ((X) (new Y()));
```

Richtig oder falsch: Das Objekt besitzt wegen der Cast-Operation nicht alle Felder, die es bei der Erzeugung hatte.
Lösung: *falsch*

4. **Aufgabe (30 Min.)** *5 Punkte*

Wir betrachten binäre Suchbäume, die in jedem Knoten x die Größe des Unterbaums enthalten, dessen Wurzel x ist. (Größe: Anzahl nichtleerer Knoten). Beispiel:



$tree = Node\ 6\ (Node\ 3\ (Node\ 1\ Leaf\ 2\ Leaf)\ 3\ (Node\ 1\ Leaf\ 4\ Leaf))\ 5\ (Node\ 2\ Leaf\ 7\ (Node\ 1\ Leaf\ 8\ Leaf))$

$dataBinSTreeSized = Leaf \mid Node\ Int\ BinSTreeSized\ Int\ BinSTreeSized$

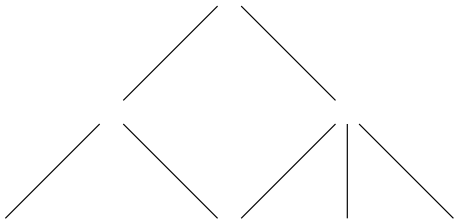
Schreiben Sie eine Haskell-Funktion $findKth\ k\ tree$, die das k-te Element in aufsteigender Sortierfolge der Werte (hier als Integer angenommen) liefert. $findKth\ 1\ tree$ liefert den kleinsten Wert im Baum, $findKth\ root\_size\ tree$ den größten. Ihr Algorithmus soll linear in der Höhe des Baumes sein. Machen Sie sich vorab klar, in welchem Teilbaum man den k-ten Wert findet. Dabei sind die Größen ($size$) des linken und rechten Teilbaums wichtig.

Lösung:

```
findKth :: Int → BinSTreeSized → Int
findKth k (Node size (Node lsize lleft lkey lright) key (Node rsize rleft rkey rright))
          | size − rsize == k                       = key
          | lsize < k                               = findKth (k − lsize − 1) (Node rsize rleft rkey rright)
          | lsize >= k                              = findKth k (Node lsize lleft lkey lright)
findKth k (Node size Leaf key (Node rsize rleft     rkey rright))
          | size − rsize == k                       = key
          | otherwise                               = findKth (k − 1) (Node rsize rleft rkey rright)
findKth k (Node size (Node lsize lleft lkey lright) key Leaf)
          | size == k                               = key
          | otherwise                               = findKth k (Node lsize lleft lkey lright)
findKth k (Node size Leaf key Leaf)
          | size == k                               = key
          | otherwise                               = error "k ist unzulaessig"
```

5. **Aufgabe (15 Min.)** *2+3 Punkte*

   (a) Gegeben sei der folgende B-Baum mit dem Minimalgrad 2. Wie viele Werte (minimal und maximal) kann der folgende B-Baum enthalten?



   (b) Wie viele innere Knoten kann ein B-Baum mit dem Minimalgrad 2 enthalten, der 9 Blätter besitzt?

   i. 5
   ii. 6
   iii. 7
   iv. 8
   v. 9

Lösung:

(a) Minimalgrad 2 $\longrightarrow$ min. Anzahl der Schlüssel pro Blatt 1 $\longrightarrow$ 4+5=9
$\longrightarrow$ max. Anzahl der Schlüssel pro Blatt 3 $\longrightarrow$ 4+15=19

(b) Die Antwort: 5 und 7. Die Überlegungen: Insgesamt hat man 9 Blätter, somit gibt es folgende Verteilung der Verweise der Knoten der überliegenden Ebene: 2 2 2 3. Mit allen anderen (3 3 3 oder 4 2 3) hat man nur einen Vater-Knoten und damit 4 innere Knoten. Also bei der (2 2 2 3)-Verteilung hat man entweder eine zusätzliche Ebene mit einem Knoten, der 3 Schlüssel und dem entsprechend 4 Verweise hat, oder zwei zusätzliche Ebenen (eine Ebene mit zwei Knoten und die andere mit einem Knoten, die jeweils einen Schlüssel haben).

6. **Aufgabe (30 Min.)** *1+2+3+4 Punkte*

   (a) Ist die Folge F = [4,7,10,9,16,12,8,13,11,20] die Darstellung eines binären Heaps? Zeichnen Sie bitte den durch F angegebenen Heap. Stellen Sie gegebenenfalls die Heap-Eigenschaft her.

   (b) Zeigen Sie: Das maximale Element in einem Heap ist ein Blatt.

(c) Zeigen Sie: Ein Heap mit n Elementen besitzt $\lceil \frac{n}{2} \rceil$ Blätter

(d) Schreiben Sie eine Methode `Comparable getMax()`, die den maximalen Wert bestimmt, und dabei die möglichst kleine Anzahl von Vergleichen macht.

Lösung:

(a) Nein.

(b) Beweis per Widerspruch. Sei T ein Heap und sei weiter das maximale Element ein innerer Knoten v $\Longrightarrow$ v hat mind. ein Kind, das kleiner als v ist $\Longrightarrow$ T ist kein Heap.

(c) $Beweis \quad mittels\ vollst.\ Induktion$
$Ind.\ Anfang.: \ n = 2, \ der\ Heap\ T\ hat\ 1\ Blatt.\ Nach\ der\ Formel\ \lceil \frac{2}{2} \rceil \ = \ 1\ hat\ T\ auch\ ein\ Blatt$
$Ind.\ Ann.: \ der\ Heap\ T\ hat\ n\ Knoten\ und\ somit \lceil \frac{n}{2} \rceil \ Blätter$
$Ind.\ Schritt.: \ der\ Heap\ T\ hat\ n+1\ Knoten$
$Fallunterscheidung: \ n{-}gerade\ (also\ n = 2m), \ das\ bedeutet\ ein\ Knoten\ hat\ ein\ Kind\ (Blatt).$
$Mit\ dem\ Erhöhen\ von\ n\ um\ 1\ erhöht\ sich\ auch\ die\ Anzahl\ der\ Blätter\ um\ 1.$
$Somit\ hat\ man \lceil \frac{n}{2} \rceil + 1 \ = \ \lceil \frac{2m}{2} + \frac{1}{2} \rceil \ = \ \lceil \frac{2m+1}{2} \rceil \ = \ \lceil \frac{n+1}{2} \rceil$
$n{-}ungerade\ (also\ n = 2m{+}1), \ das\ bedeutet, \ dass\ alle\ Knoten\ der\ vorletzten\ Ebene\ 2\ Kinder, \ die\ Blätter\ sind, \ haben.$
$Mit\ dem\ Erhöhen\ von\ n\ um\ 1\ erhöht\ sich\ die\ Anzahl\ der\ Blätter\ nicht.$
$Somit\ hat\ man \lceil \frac{n}{2} \rceil \ = \ \lceil \frac{2m+1}{2} + \frac{1}{2} \rceil \ = \ \lceil \frac{2m+2}{2} \rceil \ = \ \lceil \frac{n+1}{2} \rceil$

(d) getMax-Methode:

```
Comparable getMax() {
  Comparable auxmax=arrayBT[0]; // in arrayBT stehen die Heap-Elemente
  int i=count-1;                // count ist die Anzahl der Elemente im Heap

  while (i >= count/2) {
    if (auxmax < arrayBT[i]) {
      auxmax = arrayBT[i];
      i--;
    }
  }
  return auxmax;
}
```