

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.

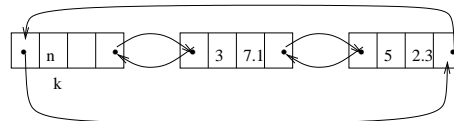
Lösung: Die Zusicherung (C) ist die Invariante.

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 $(a_0, a_1, \dots, a_{n-1})$ 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
}

```

- Wie lautet die Repräsentationsinvariante?
- Geben Sie bitte die Abstraktionsfunktion an.
- Implementieren Sie den Konstruktor der Klasse Vector.
- Implementieren Sie die Methode get.
- Implementieren Sie die Methode put.

Lösung:

- $I \equiv \forall e_1, e_2 \text{ in Vector} \wedge e_1 \neq \text{head} \wedge e_1.\text{next} == e_2 : e_1.\text{position} < e_2.\text{position} \wedge e_1.\text{value} \neq 0$
- $\alpha(\text{position}, \text{value}, \text{head}, \text{head}) = [0|i \leftarrow [1..\text{head}.\text{position}]$
 $\alpha(\text{position}, \text{value}, \text{prev}, \text{head}) = [\text{value}] ++ [0|i \leftarrow [1..\text{head}.\text{position} - \text{position} - 1]$
 $\alpha(\text{position}, \text{value}, \text{prev}, \text{next}) = [\text{value}] ++ [0|i \leftarrow [1..\text{next}.\text{position} - \text{position} - 1]] ++ \alpha(\text{next}.\text{position}, \text{next}.\text{value}, \text{next}.\text{prev}, \text{next}.\text{next})$
- Konstruktor:

```

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

```
- 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
}

```
- put-Methode:

```

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

    if (x == 0) return;

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

```

```

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 ausgeführt:

```

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()));

```

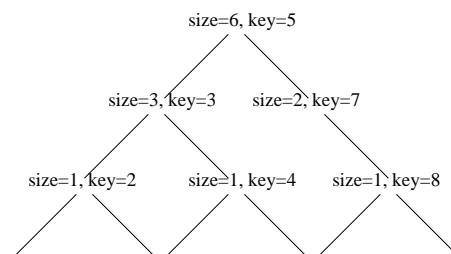
Richtig oder falsch: Das Objekt besitzt wegen der `Cast`-Operation nicht alle Felder, die es bei der Erzeugung hatte.

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 | 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:

```

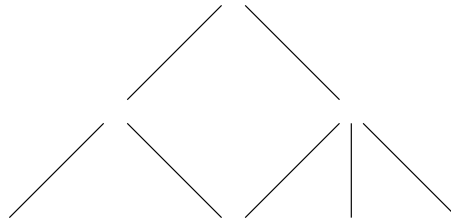
findKth :: Int -> BinSTreeSized -> Int
findKth k (Node size (Node lsize lleft lkey lright) key (Node rsize rleft rkey rright))
  | size - rsize == k           = key
  | lsize < k                   = findKth (k - lsize - 1) (Node rsize rleft rkey rright)
  | lsize >= k                  = findKth k (Node 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 unzulässig"

```

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?
- 5
 - 6
 - 7
 - 8
 - 9

Lösung:

- (a) Minimalgrad 2 \rightarrow min. Anzahl der Schlüssel pro Blatt 1 $\rightarrow 4+5=9$
 \rightarrow max. Anzahl der Schlüssel pro Blatt 3 $\rightarrow 4+15=19$
- (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 $v \implies v$ hat mind. ein Kind, das kleiner als v ist $\implies T$ ist kein Heap.

(c) Beweis 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;
}
```

7. Aufgabe (20 Min.)

5 Punkte

Gegeben sei ein Datentyp D , dessen Elemente Integer-Zahlen sind. Der Datentyp D besitze drei Methoden `insert`, `delete`, `findclosest`. Die Methode `findclosest(y)` (y aus der aktuellen Elementenmenge) liefert ein Element x der aktuellen Elementenmenge zurück, für das folgendes gilt: $|x - y| \leq |z - y|$, für alle z in der aktuellen Elementenmenge. Seien weiter T_{insert} , T_{delete} und $T_{findclosest}$ die Laufzeiten der entsprechenden Methoden im schlimmsten Fall, und die Größe T definiert als: $T = \max(T_{insert}, T_{delete}, T_{findclosest})$.

Bei welcher der folgenden Datenstrukturen für D ist T minimal?

- (A) eine sortierte Liste
- (B) eine nicht sortierte Liste
- (C) ein Heap
- (D) ein AVL-Baum
- (E) eine hash-Tabelle mit den durch verkettete Listen gelösten Konflikten

Begründen Sie Ihre Antwort.

Lösung:

Die Antwort: (D).

Begründung:

- (A) $T_{insert} = O(n), T_{delete} = O(n), T_{findclosest} = O(1)$ $\rightarrow T = O(n)$
 (B) $T_{insert} = O(1), T_{delete} = O(n), T_{findclosest} = O(n)$ $\rightarrow T = O(n)$
 (C) $T_{insert} = O(\log_2 n), T_{delete} = O(\log_2 n), T_{findclosest} = O(n)$ $\rightarrow T = O(n)$
 (D) $T_{insert} = O(\log_2 n), T_{delete} = O(\log_2 n), T_{findclosest} = O(\log_2 n)$ $\rightarrow T = O(\log_2 n)$
 (E) $T_{insert} = O(1), T_{delete} = O(n), T_{findclosest} = O(n)$ $\rightarrow T = O(n)$

(**) **Aufgabe (20 Min.)**

5 Punkte

Bekanntlich enthält ein AVL-Baum der Höhe h mindestens $N(h) = Fib(h+2) - 1$ Knoten und höchstens $2^{h+1} - 1$ Knoten. Zeigen Sie: für jedes n mit $Fib(h+2) - 1 \leq n \leq 2^{h+1} - 1$ gibt es einen AVL-Baum mit n Knoten und Höhe h . (Hinweis: die vollst. Induktion) **Lösung:** Zu zeigen: $\forall n \in N_0$ mit $Fib(h+1) - 1 \leq n \leq 2^{h+1} - 1 \exists$ ein AVL-Baum mit n Knoten und der Höhe h .

Ind. Anf.

$n=1, h=0$ - klar (ein Knoten ist ein AVL-Baum und es gilt $Fib(0+1) - 1 \leq n \leq 2^{0+1} - 1$)

Bemerkung: manche Autoren fangen bei $n=1$ mit $h=1$ an.

Ind. Ann.

$\forall i \in \{1, \dots, n\}$ mit $Fib(h+1) - 1 \leq i \leq 2^{h+1} - 1 \exists$ ein AVL-Baum mit i Knoten und der entsprechenden Höhe h .

Ind. Schritt

$n \leftarrow n+1$, weiter sei n_l die Anzahl der Knoten im linken Teilbaum und n_r die Anzahl der Knoten im rechten Teilbaum

Fallunterscheidung:

1. Fall: die Höhe h des Baumes ändert sich nicht. O.B.d.A. sei der eingefügte Knoten im linken Teilbaum. Nach der Ind. Ann. gilt

$$\begin{aligned} Fib(h+1) - 1 &\leq n_l + 1 \leq 2^h - 1 \\ &+ \\ Fib(h+1) - 1 &\leq n_r \leq 2^h - 1 \\ &+ \\ 1 &\leq 1 \leq 1 \\ \hline Fib(h+1) + Fib(h) + Fib(h-1) - 1 - 1 + 1 &\leq n + 1 \leq 2^{h+1} - 1 \end{aligned}$$

$$Fib(h+1) + Fib(h) + Fib(h-1) - 1 - 1 + 1 \leq n + 1 \leq 2^{h+1} - 1 \implies Fib(h+2) - 1 < n + 1 \leq 2^{h+1} - 1$$

2. Fall: die Höhe h des Baumes ändert sich. O.B.d.A. sei der eingefügte Knoten im linken Teilbaum. Nach der Ind. Ann. gilt

$$\begin{aligned} Fib(h+2) - 1 &\leq n_l + 1 \leq 2^{h+1} - 1 \\ &+ \\ Fib(h+1) - 1 &\leq n_r \leq 2^h - 1 \\ &+ \\ 1 &\leq 1 \leq 1 \\ \hline Fib(h+1+2) - 1 &\leq n + 1 \leq 3 * 2^h - 1 \end{aligned}$$

$$Fib((h+1)+2) - 1 \leq n + 1 \leq 3 * 2^h - 1 \implies Fib((h+1)+2) - 1 \leq n + 1 < 2^{(h+1)+1} - 1$$

(*) **Aufgabe (20 Min.)**

1+4 Punkte

Gegeben sei ein Teil der Implementierung des ADTs Zahlenmenge:

```
class IntSet {
    final int Max;
    int last = 0;
    int[] v;
```

...

```

public void insert (int e) throws InsertException {
    if (last < Max) {
        v[last] = e;
        last++;
    } else throw new InsertException();
}
...
}

```

Mit der konkreten Einfüge-Operation für ein Objekt S speichert man den Wert des einzufügenden Elementes in $S.v[S.last]$ und inkrementiert $S.last$, falls $last$ ungleich $S.Max - 1$ ist. Mit dieser Implementierung ist es möglich, auch Duplikate in Objekt Menge zu speichern. 1) Sei A die Abstraktionsfunktion für $IntSet$. Wie kann $A(S)$ definiert werden?

- (α) $\{i \mid i \text{ in } 0 \dots S.last-1\}$
- (β) $\{i \mid i \text{ in } S.last \dots S.Max-1\}$
- (χ) $\{S.v[i] \mid i \text{ in } 0 \dots S.last-1\}$
- (δ) $\{S.v[i] \mid i \text{ in } 0 \dots S.Max-1\}$
- (e) $\{S.v[i] \mid i \text{ in } 0 \dots S.v[S.last]-1\}$

2) Das folgende Programmfragment verändert konkrete $IntSet$ -Objekte:

```

public void mutation (IntSet S, int x) {
    int k;

    k = 0;
    while (k <= S.last) {
        if (S.v[k] == x){
            S.v[k] = S.v[S.last];
            S.last = S.last - 1;
        }
        else k = k + 1;
    }
}

```

(i) Geben Sie die Vorbedingung und den Effekt der while-Schleife an.

(ii) Welche der folgenden abstrakten Operationen des ADTs „Menge der Integer-Zahlen“ implementiert die Funktion `mutation`:

- (a) Addieren x zu S
- (b) Entfernen x aus S
- (c) Bilden des Durchschnitts von x und S
- (d) Bilden der Vereinigung von x und S
- (e) Kopieren von S
- (f) keine

Lösung:

1) Die Antwort: (χ) $\{S.v[i] \mid i \text{ in } 0 \dots S.last - 1\}$

2) (i) $P = \{k == 0\}$, $Q = \{\forall i \text{ in } 0 \dots S.last - 1 : S.v[i] \neq x\}$

(ii) Die Antwort ist (b) Entfernen x aus S (unabhängig davon ob die Bedingung der while-Schleife $k \leq S.last$ oder $k < S.last$ lautet).

Bemerkung: als richtig wurde auch die Antwort (f) keine gesehen.