Informatik für Informationsmanger I

Universität
Koblenz-Landau
Institut für Informatik

Prof. Dr. Steffen Staab
Dr. Manfred Jackel

SS 2006

Nachklausur

13.04.2007

Lösung

Bitte in Druckschrift leserlich ausfüllen!

Name _____

Vorname _____

E-Mail-Adresse _____ j k l @uni-koblenz.de

Matrikelnummer _____

Studiengang:

☐ Computervisualistik (Diplom)
☐ Informatik (Diplom)
☐Computervisualistik (BSc)
☐Informatik (BSc)
☐ Anglistik & Medienmanagement (BSc)
☐ Informationsmanagement (BSc)

☐ Diese Prüfungsleistung melde ich verbindlich als Freiversuch im Sinne der Prüfungsordnung an.

☐ Diese Prüfungsleistung ist mein 2. Versuch (Nachklausur). Erstversuch im _____ (Semester).

Auswertung:

	1	2	3	4	5	6	GESAMT
Punkte							

Aufgabe 1 (12 Punkte)

Diese Aufgabe umfasst 3 Multiple-Choice-Unteraufgaben. Innerhalb jeder Unteraufgabe gilt: wenn alle 4 Kreuze an der richtigen Stelle stehen, gibt es 4 Punkte für die Unteraufgabe. Ein falsches Kreuz gibt einen Punkt Abzug. Wer 2 richtige und 2 falsche Kreuzchen in einer Unteraufgabe macht, erhält 1+1-1-1=0 Punkte. Es gibt keine negativen Gesamtpunktzahlen pro Unteraufgabe, jede Unteraufgabe bringt 0 bis 4 Punkte.

Aufgabe 1a (4 Punkte):

		Ja	Nein	Frage
a)				**Interface**
	1.	☐	☒	`class A extends B,C implements D,E {…}` ist korrekter Java-Code
	2.	☐	☒	Eine Klasse, die abstrakte Methoden besitzt, ist ein Interface.
	3.	☒	☐	Die Klasse Integer besitzt eine Methode compareTo.
	4.	☐	☒	Ein Interface darf keine überladenen Methoden enthalten.

Aufgabe 1b (4 Punkte):

				Sortieren
	1.	☒	☐	Sortieren durch Einfügen ist ein greedy-Algorithmus.
	2.	☐	☒	Quicksort hat –bei gleicher Anzahl der Elemente- immer den gleichen Aufwand, unabhängig von der konkreten Zahlenfolge.
	3.	☐	☒	Mit n Vergleichen und n Vertauschungen kann man n/2 Zahlen sortieren.
	4.	☒	☐	Mergesort ist besser als $O(n^2)$.

Aufgabe 1c (4 Punkte):

c)				**Suchen**
	1.	☒	☐	Lineares Suchen ist bei kleinen Suchmengen empfehlenswert.
	2.	☒	☐	Suchen in einer Hashtabelle ohne Kollisionen hat den asymptotischen Aufwand O(1).
	3.	☒	☐	Der Aufwand für binäres Suchen liegt zwischen O(1) und O(log(n)).
	4.	☐	☒	Binärsuche ist ein greedy-Verfahren.

Aufgabe 2 (12 Punkte)

Die folgende Klasse implementiert generische Listen. Ergänzen Sie die Klasse um eine Methode `public ObjectNode getSecond()`, die die Knoteninformation des zweiten Listenelementes der Liste liefert. Ist die Liste leer oder hat nur 1 Element, wird null zurückgegeben.

```
public class List {
      private ObjectNode head;

      public List() {
            head = null;
      }

      public List(ObjectNode head) {
            this.head = head;
      }

      public void insertFirst(Object data) {
            head = new ObjectNode(data, head);
      }

      public Object getHead() {
            return head.data;
      }

      public boolean empty() {
            return head == null;
      }

      public List getTail() {
            if (head == null)
                  return new List(null);
            else if (head.getNext() == null)
                  return new List(null);
            else
                  return new List(head.getNext());
      }
```

```
public class ObjectNode {
  Object data;        Ob-
jectNode next;

  Object getData() {
    return data;
  }

  void setData(Object data)
{
    this.data = data;
  }

  ObjectNode getNext() {
    return next;
  }

  void setNext(ObjectNode
next) {
    this.next = next;
  }

  public ObjectNode(Object
a) {
    this(a, null);
  }

  public ObjectNode(Object
a, ObjectNode n) {
    data = a;
    next = n;
```

```
/**
*
* Bitte fügen Sie den Code in den Methodenrumpf ein:
**/

      public ObjectNode getSecond() {


            if (empty()) return null;
            else {
                  List s = getTail();
                  if (s.empty()) return null;
                  else { s = s. getTail();
                         return s.getHead();
                       }
            }









      }
}
```

Aufgabe 3 (12 Punkte)

Gegeben sei folgendes Code-Fragment:

```
          int i=n,j, sum=0;
          …
(1)  while (i>0) {
(2)       j = 0;
(3)       while (j<i) {
(4)          sum+=1;
(5)          j = j+1;
          }
(6)       i = i-1;
     }
```

a) Berechnen Sie den exakten Aufwand in Abhängigkeit von n unter der Maßgabe, dass jede Zeile genau eine Aufwandseinheit erfordert.

(1) n+1
(2) n
(3) n+1 + n + … +2)
b) (4) (n + n-1 + … +1)
(5) (n + n-1 + … +1)
(6) n

= n+1 + n + n +3(n*(n+1)/2) + n = 3(n*(n+1)/2) +4n+1

b) Welche Werte nimmt sum in Zeile 6 bei jedem Schleifendurchlauf an für n=10?

Durchlauf	1	2	3	4	5	6	7	8	9	10
sum	10	19	27	34	40	45	49	52	54	55

c) Zeigen Sie, dass f(n)=4 n^2+6n+10 zu O(n^2) gehört **(Diese Teilaufgabe braucht für A&M nicht bearbeitet zu werden.**

für n>=1 gilt: f(n) = 4n^2+6n+10 <= 4n^2+6$n^2$10n^2= 20 n^2 , also
$f(n) \in O(n^2)$

Aufgabe 4 (12 Punkte)

Sortieren Sie die folgenden Zahlen Array-basiert, indem Sie die nachstehenden Tabellen vervollständigen.

a) Sortieren durch Auswählen (selection sort)

4	**8**	**2**	**1**	**7**	**5**	**3**	**6**
1	8	2	4	7	5	3	6
1	2	8	4	7	5	3	6
1	2	3	4	7	5	8	6
1	2	3	4	5	7	8	6
1	2	3	4	5	6	8	7
1	2	3	4	5	6	7	8

b) Sortieren durch Austauschen (Bubblesort ohne Optimierung)

4	**8**	**2**	**1**	**7**	**5**	**3**	**6**
4	2	1	7	5	**3**	6	8
2	1	4	5	3	6	7	8
1	2	4	3	5	6	7	8
1	2	3	4	5	6	7	8

Aufgabe 5 (12 Punkte)

Aus dem Paket Kapitel13 kennen Sie die Klasse Tree und das

```
public interface NodeActionInterface {
  /**
   * Methode, deren Implementierung auf einem Knoten arbeitet.
   * @param n Der zu bearbeitende Knoten
   */
  public void action(Node n);
}
```

Wir gehen im Folgenden von einem Baum aus, dessen Knoten Integer-Objekte beinhalten. Die Knoten sehen dann so aus:

```
Public class Node {
      Object data;
      Node left;
      Node right;
…
}
```

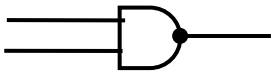
Ist n ein Objekt von Typ `Node`, so wird mit `n.data = new Integer(7)` der Wert 7 als Objekt gespeichert. Der int-Wert kann so ausgelesen und beispielsweise der Variablen i zugewiesen werden: `int i = ((Integer)n.data).intValue()`.

Leiten Sie aus `NodeActionInterface` eine Klasse `public class AverageNodeValue` ab, die das arithmetische Mittel der Knoteninhalte aller Knoten ermittelt, auf die die Methode action angewandt wird. Wir geben den Rahmen vor, Sie brauchen nur die Methode action auszuprogrammieren!

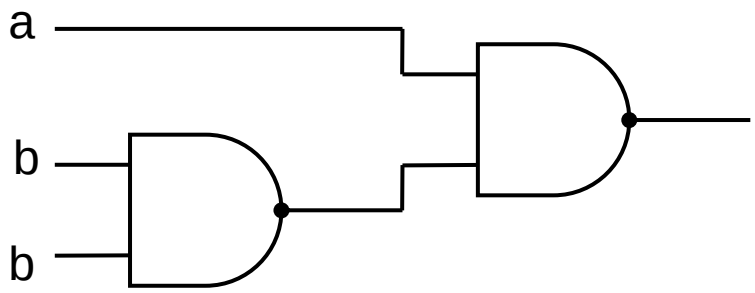
```
   public class AverageNodeValue implements NodeActionInterface {

      private int sum=0;  // Summe der Knoteninhalte
      private int count=0; // Anzahl der Knoten

      public void action(Node n) { // hier programmieren!

            count++;
            sum += ((Integer)n.data).intValue();









      }

      public String toString() {
            if (count!=0)
            return “Durchschnitt=“+sum/count; else return“Leerer Baum“;

}
}
```

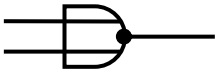
Aufgabe 6 (12 Punkte)

a) Geben Sie eine Gatter-Schaltung an, die die Verknüpfung $a \Rightarrow b$ ausschließlich mit NAND-Gattern realisiert.



$$a \Rightarrow b \equiv \neg a \vee b \equiv \neg(a \wedge \neg b) \equiv \neg(a \wedge \neg(b \wedge b))$$





b) Geben Sie eine Gatter-Schaltung an, die die Verknüpfung $a \Rightarrow b$ ausschließlich mit NOR-Gattern realisiert.

$$a \Rightarrow b \equiv \neg a \vee b \equiv \neg\neg(\neg a \vee b)$$