Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Wissens- und Sprachverarbeitung
Prof. Dr. A. Nürnberger

Magdeburg, 02.08.07

Prüfungsklausur

Algorithmen und Datenstrukturen

Name:	Matrikelnummer:
Vorname:	Studiengang:
Blattanzahl:	
Unterschrift Student/in:	Unterschrift Aufsicht:

Aufg. 1	Aufg. 2	Aufg. 3	Aufg. 4	Aufg. 5	Aufg. 6	Aufg. 7	Summe
(von 7)	(von 6)	(von 7)	(von 10)	(von 13)	(von 7)	(von 10)	(von 60)

Allgemeine Hinweise

- Schreiben Sie auf jedes Blatt Ihren Namen, Ihre Matrikelnummer und die Seitennummer.
- Die Programme sind gut zu kommentieren! Programmstrukturen sind je nach Aufgabenstellung als Java-ähnlicher Pseudocode oder in Java anzugeben.
- Beginn und Ende einer Aufgabenlösung sind durch einen waagerechten Strich deutlich zu kennzeichnen. Ungültige Lösungen sind durchzustreichen.
- Zugelassene Hilfsmittel: ausschließlich Schreibmaterialien
- Die Klausur besteht aus 7 Aufgaben, die Bearbeitungszeit beträgt 180 Minuten.
- Bitte **nicht** mit grünem Stift schreiben.

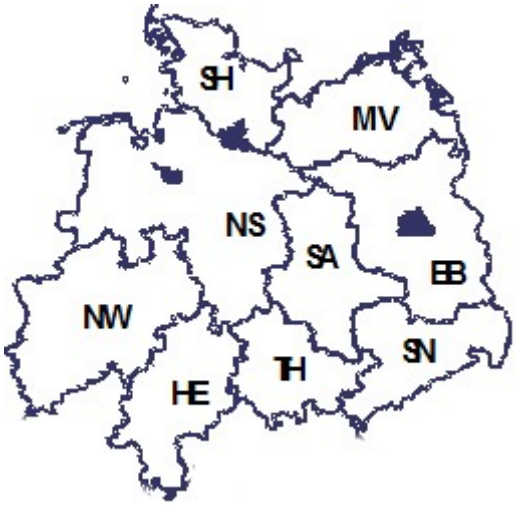
Aufgabe 1 (7 Punkte)
Definieren Sie in *Java* eine Klasse *Queue<T>* mit maximal 100 Elementen vom Typ *T* auf der Grundlage eines **Arrays** mit geeignetem Konstruktor und den Methoden:

- *enqueue (*oder *offer)* fügt ein neues Element am Ende der *Queue* ein
- *dequeue (*oder *poll)* entfernt das erste Element aus der *Queue* und liefert es zurück
- *size* liefert die Anzahl der Elemente in der *Queue*

Es ist unbedingt abzusichern, dass alle Methoden mit der Komplexität O(1) arbeiten. Beschreiben Sie kurz Ihre Realisierungsidee.

Hinweis:
Vergessen Sie nicht, bei leerer bzw. voller *Queue* entsprechende *Exceptions* auszulösen. Die Klassen *QueueFullException* und *QueueEmptyException* können als bekannt angenommen werden.

Aufgabe 2 (1 + 2 + 2 + 1 = 6 Punkte)
In der folgenden Abbildung sehen Sie einen Ausschnitt aus Karte der Bundesrepublik Deutschland mit neun Flächen-Bundesländern:



BB	Brandenburg
HE	Hessen
MV	Mecklenburg-Vorpommern
NS	Niedersachsen
NW	Nordrhein-Westfalen
SA	Sachsen-Anhalt
SH	Schleswig-Holstein
SN	Sachsen
TH	Thüringen

a. Zeichnen Sie einen Graphen, wobei jeder Knoten ein Flächen-Bundesland präsentiert und die Kanten für das Vorhandensein einer gemeinsamen Grenze stehen.
b. Geben Sie die Adjazenzliste dieses Graphen an, wobei die Listen alphabetisch (nach dem Kürzel) geordnet sein sollen.
c. Geben Sie die Reihenfolge der Knoten bei einem Tiefen- **und** einem Breitendurchlauf des Graphen an, wobei der Startknoten Sachsen-Anhalt(SA) sei. Verarbeiten Sie dabei die Kanten eines Knotens in der sortierten Reihenfolge der Adjazenzliste.
d. Begründen Sie, warum der Graph keinen Eulerkreis (geschlossenen Eulerzug) bildet. Welche Kanten müsste man entfernen, damit er einen Eulerkreis bildet?

Aufgabe 3 (3 + 4 = 7 Punkte)
a. Gegeben sei der folgende Algorithmus:

```
XYZ:  var Q, R : int
      input X, Y
      /* Vorbedingungen */
      R := X; Q:= 0;
      while R  >= Y
         do
            R = R - Y;
            Q = Q + 1;
          od
      output Q, R
      /* Nachbedingung */
```

- Demonstrieren Sie die Arbeitsweise am Beispiel x =13 und y=5.
- Was berechnet dieser Algorithmus?
- Nennen Sie die Vorbedingungen und die Nachbedingung.

b. Mehrere Gegenstände mit unterschiedlichen Größen und Werten sollen in einen Rucksack gepackt werden, wobei der Gesamtwert möglichst groß sein soll. Jeder Gegenstand ist nur einmal vorhanden und darf nicht geteilt werden.
   - Beschreiben Sie die Lösung dieses Rucksackproblems mit einem Greedy-Algorithmus und ermitteln Sie die Lösung für folgendes Beispiel. Das Fassungsvermögen des Rucksacks sei 12:

Gegenstand	A	B	C	D	E
Größe	2	4	5	2	3
Wert	3	7	6	6	2

   - Nennen Sie Strategien, um das Rucksackproblem optimal zu lösen.

Aufgabe 4 (4 + 1 + 1 + 2 + 2 = 10 Punkte)

a. Gegeben sei die Klasse TreeNode<T>, die einen Knoten eines binären Baumes beschreibt.

```
public class TreeNode<T> {
    protected T key;
    protected TreeNode<T> left, right;
    public TreeNode(T x) {
        key = x;
        left = null;
        right = null;
    }
    public TreeNode<T> copy() {
        …
    }

    public int count () {
        …
    }
}
```

Ergänzen Sie die Klasse *TreeNode<T>* um die Methoden

- *copy* kopiert den Knoten und seine beiden Teilbäume und
- *count* zählt die Knoten mit **genau einem** Kind in diesem Teilbaum

b. Welche Datenstruktur ist geeignet, wenn Sie einen Baum in einem Array speichern wollen? Sie suchen eine stets ausgeglichene Baumstruktur mit einer garantiert kompakten Anordnung der Knoten, die sich ohne Lücken im Array speichern lässt.

c. Ein Projekt erfordert eine kompakte Datenstruktur mit einem garantierten Suchaufwand von O(log n) für beliebige Knoten im schlechtesten Fall. Welche Struktur können Sie vorschlagen?

d. Durch welche Eigenschaften ist ein AVL-Baum gekennzeichnet? Welche Vorteile hat ein AVL-Baum? Wie viele Blätter hat ein AVL-Baum der Höhe h maximal?

e. Fügen Sie nacheinander die Werte 4, 10, 9, 13, 22, 24 in einen AVL- **oder** Rot-Schwarz-Baum ein. Skizzieren Sie alle notwendigen Rotationen bzw. Umfärbungen.

Aufgabe 5 (2 + 1 + 6 + 3 + 1 = 13 Punkte)

a. Was ist ein *Min-Heap*? Gehen sie auf Heap-Form und Heap-Eigenschaften ein.
b. Skizzieren Sie einen (beliebigen) Min-Heap aus den Zahlen *13, 55, 22, 49, 27, 11, 52 und 14*, die Werte brauchen nicht in dieser Reihenfolge eingefügt werden.
c. Deklarieren Sie einen *Heap*, wählen Sie eine geeignete Datenstruktur dafür.
   Schreiben Sie in Java-ähnlichem Pseudocode die Methode *T extractMin()* zum Entfernen des kleinsten Objektes aus dem Heap. Kommentieren Sie die einzelnen Schritte.
   Wie ist die Komplexität dieses Algorithmus in O-Notation?
d. Beschreiben Sie den Algorithmus für das Sortierverfahren Heap-Sort in Java-ähnlichem Pseudocode. Sie können die Methoden *createHeap* zum Aufbau eines Heaps und *extractMin* als gegeben voraussetzen.
e. Ist Heap-Sort ein stabiles Sortierverfahren? Begründen Sie die Antwort.

Aufgabe 6 (1 + 3 + 3 = 7 Punkte)

Ein Objekt *Person* sei durch seinen Vornamen und das Geburtsdatum bestimmt (*name, tag, monat, jahr*). 1000 solcher Objekte sollen in eine Hashtabelle eingefügt werden.

a. Wie groß wählen Sie die Tabelle und begründen Sie Ihre Wahl.
b. Schreiben Sie eine Klasse *Person*, die obiges Objekt repräsentiert. Außer den Attributen soll ein Konstruktor angegeben werden.
   Formulieren Sie in dieser Klasse jeweils eine gute und eine schlechte Hashfunktion *int hashCode()* und begründen Sie dies.
c. Wann ist es sinnvoll, Hashverfahren einzusetzen?
   Nennen Sie die Vor- und Nachteile gegenüber dem Speichern in einer Liste und in einem Baum.

Aufgabe 7 (2 + 7 + 1 = 10 Punkte)

In eine einfach verkettete Liste (vom Typ *SimpleList<Integer>*) wurden ganze Zahlen eingefügt. Mit einem "Filter" sollen aus dieser Liste alle geraden Zahlen entfernt und in eine neue Liste „eingesammelt“ werden. Das Filtern soll nur durch Umhängen der Zeiger erfolgen.

a. Skizzieren Sie für eine Liste mit den Werten: 25, 34, 37, 32, 30, 21 das Umhängen der Zeiger.
b. Beschreiben Sie in Stichpunkten Ihre Lösungsidee.
   Schreiben Sie eine Methode *SimpleList<Integer> filter(SimpleList<Integer> list)* die eine gefilterte Liste zurückgibt und die übergebene Liste so ändert, dass nur die ungeraden Zahlen enthalten sind.
   Zur Verfügung stehen die Methoden *getHead()* aus der Klasse *SimpleList* und *getNext(), setNext(obj)* und *getElement()* aus der Klasse *Node*.
c. Wie groß ist der Aufwand für das Filtern in O-Notation, begründen Sie kurz?

Wir wünschen Ihnen viel Erfolg!