berlin

Fakultät IV

Klausur MPGI 2
31.7.2007

Pepper
Frank / Kleeblatt / Wimmer

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 120 Minuten

- Es sind beliebige Papier-Unterlagen als Hilfsmittel zugelassen. Nicht zugelassen sind elektronische Hilfsmittel, wie z. B. Taschenrechner, Handys oder Laptops.
- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier geschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseiten. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!** Wenn Sie zusätzliche, von uns ausgegebene Blätter verwenden, geben Sie unbedingt an, zu welcher Aufgabe die Lösung gehört!
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen werden nicht gewertet! Streichen Sie gegebenenfalls eine Lösung durch!
- Schreiben Sie nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, können nicht gewertet werden!
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Sollten Sie im Softwareteil eine Teilaufgabe nicht lösen können, so dürfen Sie die dort geforderte Funktion in anderen Teilaufgaben verwenden.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

Aufgabe	Punkte	erreicht
1	6	
2	5	
3	3	
4	7	
5	2	
6	9	
7	8	
8	7	
9	3	

1. Aufgabe (6 Punkte): Java-Programmierung (Arrays)

In einer Jugendherberge gibt es Zimmer für Übernachtungsgäste. Auf jedem Zimmer steht eine bestimmte Anzahl von Betten für die Gäste zur Verfügung.

```
public class Gaestezimmer {
    public int betten;
    public int gaeste;
}
```

Das Attribut `betten` gibt an, wie viele Betten in dem Zimmer vorhanden sind. Das Attribut `gaeste` gibt an, wie viele der Betten zur Zeit von Gästen belegt sind.

Die Jugendherberge hat mehrere Etagen, auf jeder Etage gibt es mehrere Zimmer. Die Zimmer sind nach Etage und Nummer in einem 2-dimensionalen Array erfasst (1. Dimension Etage, 2. Dimension Nummer). Gehen Sie davon aus, dass alle Felder im Array mit Zimmern initialisiert wurden (d.h. sie sind nicht mit `null` belegt).

Erweitern Sie in den folgenden Aufgaben die Klasse `Jugendherberge`.

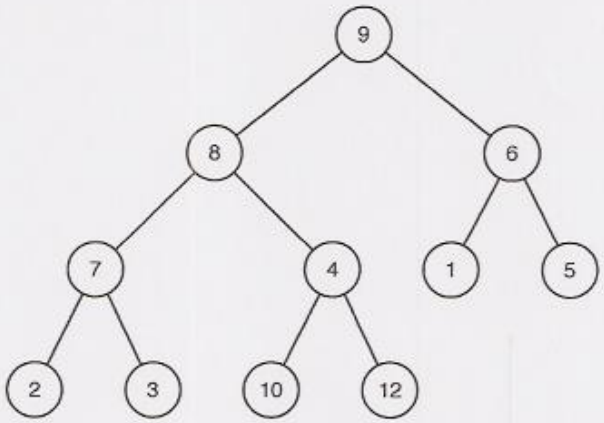
```
public class Jugendherberge {
    public Gaestezimmer [][] zimmer;
    // ... Ihre Methoden
}
```

1.1. Zimmer mit freien Betten (3 Punkte) Implementieren Sie die Methode `sucheZimmer`, die ein Gästezimmer der Jugendherberge zurückliefert, in dem es noch mindestens eine bestimmte Anzahl von freien Betten gibt. Die Mindest-Anzahl freier Betten soll als Parameter übergegeben werden. Wählen Sie einen geeigneten Rückgabewert für den Fall, dass kein Zimmer diese Bedingung erfüllt.

1.2. Freie Betten auf einer Etage (3 Punkte) Implementieren Sie die Methode `freieBettenAufEtage`, welche die Anzahl nicht belegter Betten auf einer bestimmten Etage zurückgibt. Die Nummer der Etage wird als Parameter übergeben. Falls die Etage nicht existiert, soll die Methode 0 als Ergebnis liefern.

2. Aufgabe (5 Punkte): Sortierverfahren

2.1. Heap-Eigenschaften (2 Punkte) Ist der folgende Baum ein Heap? Begründen Sie Ihre Antwort!



2.2. Sortieralgorithmen (3 Punkte) Gegeben sind die Implementierungen von den Sortieralgotihmen *A* und *B* durch die gleichgenannten Methoden:

```
public static void A (long [] a) {
        for (int i = 1; i < a.length; i++) {
            helpA (a, i);
        }
}

public static void helpA (long [] a, int h) {
        for (int i = h; i>=1; i--) {
            if (a[i-1] <= a[i])
                break;
            swap (a, i-1, i);
        }
}
```

```
public static void B (long [] a) {
        for(int i = 0; i <a .length; i++) {
            int j = helpB (a,i);
            swap (a,i,j);
        }
}

public static int helpB (long [] a, int h) {
        int result = h;
        for (int i = h+1; i < a.length; i++)
            if (a[i] < a[result])
                result = i;
        return result;
}
```

Beide Implementierungen benutzen die Hilfmethode `static void swap(long [] a, int i, int j)`, die die Zahlen an Positionen `i` und `j` im Array `a` vertauscht.

Geben Sie für jede der Methoden `A` und B den Namen des Sortieralgorithmus an, den sie implementiert. Geben Sie außerdem für jede Implementierung an, ob sie in-situ ist und ob sie stabil arbeitet.

Algorithmus	in-situ (ja/nein)	stabil (ja/nein)	Name des Verfahrens
A			
B			

3. Aufgabe (3 Punkte): Hoare-Kalkül

3.1. Verzweigungsregel (3 Punkte) Beweisen Sie für das folgende Programmfragment die partielle Korrektheit bezüglich der gegebenen Nachbedingung POST und ermitteln Sie eine passende Vorbedingung PRE.

Hinweis: Die Modulo-Funktion $mod(x, y)$ gibt den Rest der Division x/y an.

$//PRE:$

```
if (a % 2 == 0) {

    e = e + a;

} else {

    e = e + a + 1;

}
```

$//POST: mod(e, 2) = 0$

4. Aufgabe (7 Punkte): Numerik

4.1. Implementierung der Exponentialfunktion (6 Punkte) Die Exponentialfunktion e^x definieren wir aus numerischen Gründen folgendermaßen :

$$e^x = \begin{cases} \sum_{n=0}^{\infty} \frac{x^n}{n!} & \text{falls } x \geq 0 \\ \frac{1}{1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\frac{x^4}{4!}-\cdots+\ldots} & \text{falls } x < 0 \end{cases}$$

Implementieren Sie eine Methode `static double exponential(double x)`, welche die Exponentialfunktion anhand der oben angegebenen Formeln berechnet. Brechen Sie den Algorithmus ab, falls das aktuelle Ergebnis sich um weniger als 10^{-4} geändert hat.

Hinweis: Falls nötig, kann die Methode `Math.abs()` verwendet werden. Andere Methoden aus der Klasse `Math` sind nicht erlaubt.

```
static double exponential(double x) {

}
```

4.2. Genauigkeit der Näherung (1 Punkt) Implementieren Sie die Methode

```
static boolean isNearMathExp (double x, double epsilon),
```

die als Ergebnis einen booleschen Wert liefert, der angibt, ob Ihre Methode `double exponential(double x)` für das Argument `x` sich um weniger als `epsilon` von der Java-Implementierung `Math.exp(x)` unterscheidet.

```
static boolean isNearMathExp (double x, double epsilon) {




}
```

5. Aufgabe (2 Punkte): Hashing

5.1. Hashtabellen (2 Punkte) Fügen Sie die drei Schlüssel 10, 15, 21 in der angegebenen Reihenfolge in eine anfangs leere Hashtabelle ein. Die Tabelle hat 5 Speicherplätze. Die folgende Hashfunktion h und Sondierungsfunktion g sind zu verwenden:

$$\begin{aligned} h(k) &= k \bmod 5 \\ g(k,j) &= (j+3) \bmod 5 \end{aligned}$$

Hierbei ist k der einzufügende Schlüssel und j der zuletzt getestete Speicherplatz. Geben Sie für jeden Schlüssel an, welche Positionen getestet werden, wo Kollisionen auftreten, und wo der Schlüssel letzendlich gespeichert wird.

6. Aufgabe (9 Punkte): Java-Programmierung (Listen)

Im Rahmen dieser Aufgabe soll mit einer einfach verketteten Liste gearbeitet werden.

Gegeben sind die Klassen `Rechteck` und `ListElement`.

```
1  public class Rechteck {
2
3          private double a;
4          private double b;
5
6          public Rechteck(double a, double b) {
7                  this.a = a;
8                  this.b = b;
9          }
10
11         public double flaeche() {
12                 return a * b;
13         }
14 }
15
16 public class ListElement {
17
18         private Rechteck rechteck;
19         private ListElement naechstes;
20
21         public ListElement( Rechteck rechteck, ListElement naechstes ) {
22                 this.rechteck = rechteck;
23                 this.naechstes = naechstes;
24         }
25
26         public void setRechteck( Rechteck rechteck ) {
27                  this.rechteck = rechteck;
28         }
29
30         public Rechteck getRechteck() {
31                  return this.rechteck;
32         }
33
34         public ListElement getNaechstes() {
35                  return this.naechstes;
36         }
37
38         public void setNaechstes( ListElement naechstes ) {
39                  this.naechstes = naechstes;
40         }
41 }
```

6.1. Listenklasse (2 Punkte) Implementieren Sie zunächst eine Klasse `RechteckListe`.

Die Klasse repräsentiert eine einfach verkettete Liste mit Objekten der Klasse `ListElement` und soll die Attribute *Listenanfang* und *Länge der Liste* beinhalten. Achten Sie darauf, Kapselung zu verwirklichen.

Ergänzen Sie eine *get-Methode* für den Zugriff auf die Listenlänge und eine Methode `loeschen`, die alle Elemente aus der Liste entfernt.

```
public class RechteckListe {

}
```

6.2. Gefilterte Liste (3 Punkte)

Ergänzen Sie in der Klasse `RechteckListe` eine Methode `filter`, die aus der Rechteckliste alle Rechtecke entfernt, deren Fläche kleiner ist als der übergebene Wert `minFlaeche`.

Hinweis: Für die Lösung dieser Aufgabe stehen Ihnen *keine* weiteren als die Methoden aus der vorgegebenen Klasse `ListElement` zur Verfügung.

```
public void filter (double minFlaeche) {

}
```

6.3. Liste sortieren - Insertion sort (4 Punkte)

Implementieren Sie eine Methode `sort()` in der Klasse `RechteckListe`, welche die Rechtecke in der Liste nach ihren Flächeninhalten mit dem *Insertion-Sort*-Verfahren aufsteigend sortiert. Die Methode `sort` soll eine sortierte Liste zurückliefern und die ursprüngliche Liste unverändert lassen.

```
public RechteckListe sort() {

}
```

7. Aufgabe (8 Punkte): Suchbäume

7.1. Einfügen in Suchbäume (2 Punkte) Fügen Sie in einen binären, geordneten Baum (Suchbaum) mit Werten nur in den Blättern die folgenden Werte in gegebener Reihenfolge ein:

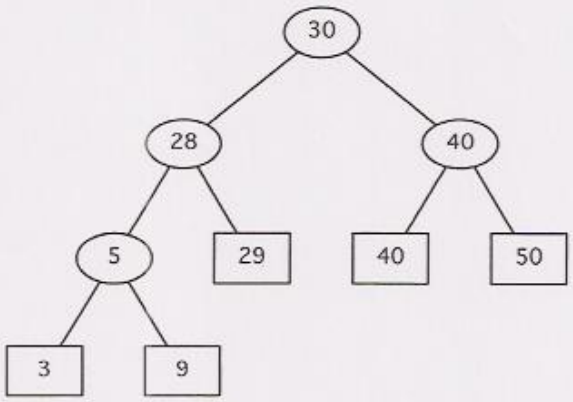
15, 22, 44, 43, 42.

Der Baum ist vor dem Einfügen des Wertes 15 leer. Beim Erzeugen innerer Knoten soll gelten: Der Schlüssel gibt den größten Wert des linken Teilbaums an. Geben Sie den Baum nach jedem eingefügten Wert an.

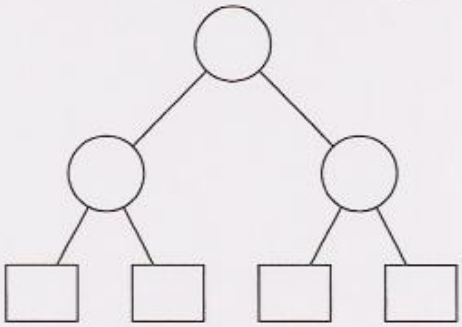
7.2. Löschen in Suchbäumen (2 Punkte) Löschen Sie aus dem gegebenen binären, geordneten Baum (Suchbaum) mit Werten nur in den Blättern nacheinander die folgenden Blätter:

29, 9, 50

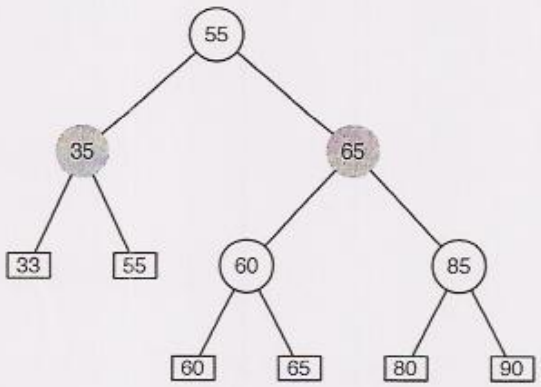
Geben Sie den Baum nach jedem gelöschten Blatt an.



7.3. Binäre Suchbäume – Aufbau (2 Punkte) Gegeben ist die folgende Struktur eines binären, geordneten Baums (Suchbaum). Die Werte sind ganze Zahlen und befinden sich nur in den Blättern. In welcher Reihenfolge müssen die vier Elemente 1, 2, 3 und 4 in einen leeren Baum eingefügt werden, um diese Struktur zu ergeben? Geben Sie alle geeigneten Einfügereihenfolgen an. Es gilt dieselbe Anforderung beim Einfügen wie in Teilaufgabe 1: Der Wert eines inneren Knotens gibt den größten Wert des linken Teilbaums an.



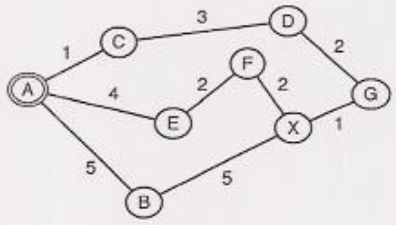
7.4. Rot-Schwarz-Bäume (2 Punkte) Ist der folgende Baum ein Rot-Schwarz-Baum? Begründen Sie Ihre Antwort! *Achtung: In der folgenden Abbildung sind die roten Knoten grau gezeichnet und die schwarzen Knoten sind nicht gefärbt!*



8. Aufgabe (7 Punkte): Graphen

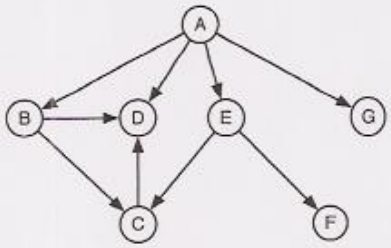
8.1. Dijkstra Algorithmus (3 Punkte) Bestimmen Sie mit Hilfe des Dijkstra-Algorithmus die kürzesten Wege vom Startknoten A zu allen anderen Knoten. Verwenden Sie dazu die folgende Tabelle, in der Sie Felder, in denen Relaxation stattfindet, umkreisen und Felder, die die endgütigen Kosten angeben, unterstreichen.

Hinweis: Bei gleichen Kosten sei die Knotenordnung so, dass kleinere Buchstaben jeweils vor größeren kommen (alphabetische Ordnung), d.h. z.B. wenn B und C beide die Kosten 5 haben, dann wählen Sie B.



Iteration	A	B	C	D	E	F	G	X
0								
1								
2								
3								
4								
5								
6								
7								
8								

8.2. Topologische Sortierung (2 Punkte) Geben Sie eine topologische Sortierung der Knoten des folgenden Graphen an:



8.3. Eindeutigkeit (1 Punkt) Ist die topologische Sortierung des Graphen aus der letzten Unteraufgabe (8.2.) eindeutig? Begründen Sie Ihre Antwort.

8.4. Topologische Sortierbarkeit (1 Punkt) Unter welcher Bedingung ist eine topologische Sortierung *nicht* möglich?

9. Aufgabe (3 Punkte): Sudoku

Beim Sudoku-Spiel gibt es folgendes Teilproblem: Gegeben ist ein Array `Feld[] zeile`. Dabei ist `Feld` durch eine Klasse folgender Art definiert:

```
class Feld {
    ...
    public Set<Integer> kandidaten; // möglicherweise leer
    ...
}
```

Beispiel:

{}	{2,3}	{6}	{3,5}	{5}	{3,5,6}	{3}	{}

9.1. Einzelnes Vorkommen einer Zahl (3 Punkte) Implementieren Sie eine Methode, die prüft, ob es eine Zahl zwischen 1 und 9 (jeweils einschließlich) gibt, die in genau einer der Mengen `kandidaten` vorkommt (im Beispiel ist das die „2"). Falls ja, liefern Sie das entsprechende `Feld`-Objekt, ansonsten `null`. Sollten mehrere Felder diese Anforderung erfüllen, geben Sie ein beliebiges davon zurück.

Hinweis: Für die Klasse Set kann man die neue for-Schleife von Java 5 benutzen. Ob ein Element in einer Menge enthalten ist, kann mit der Methode `boolean contains(Object o)` getestet werden, die für Objekte vom Typ `Set` zur Verfügung steht.