Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Simulation und Graphik
Prof. Dr. Klaus-Dietz Tönnies

Magdeburg, 14.07.03

Prüfungsklausur

Einführung in die Informatik/Algorithmen und Datenstrukturen

Name:	Matrikelnummer:
Studiengang:	
Blattzahl:	
Unterschrift Student/in:	Unterschrift Aufsicht:

Aufg. 1	Aufg. 2	Aufg. 3	Aufg. 4	Aufg. 5	Aufg. 6	Aufg. 7	Summe
(von 10)	(von 7)	(von 7)	(von 10)	(von 10)	(von 7)	(von 9)	(von 60)

- **Allgemeine Hinweise**
  Schreiben Sie auf jedes Blatt Ihren Namen, Ihre Matrikelnummer und eine Seitennummer.
- Die Programme sind gut zu kommentieren!
- Beginn und Ende der Lösung zu einer Aufgabe sind durch einen waagerechten Strich deutlich zu kennzeichnen.
- Zugelassene Hilfsmittel: nur Schreibmaterial
- **Nicht** zugelassene Hilfsmittel: Mitschriften, Bücher, Skript, Taschenrechner, Handy
- Die Klausur besteht aus 7 Aufgaben, die Bearbeitungszeit beträgt 180 Minuten.
- Zum Bestehen der Klausur sind 24 Punkte erforderlich.

Aufgabe 1 (10 Punkte)

a) Entwickeln Sie ein JAVA-Interface *GeomVector* für Klassen von Geometrievektoren beliebiger Dimension. Das Interface soll folgende Methoden umfassen:

Dimension:	Eingabe: -, Ausgabe: int
Komponente:	Eingabe: int, Ausgabe: double
Länge:	Eingabe: -, Ausgabe: double
Addition:	Eingabe: *GeomVector,* Ausgabe: *GeomVector*, Exception *DimensionMismatch* bei ungleicher Dimension

b) Schreiben Sie JAVA-Methoden für die Implementierung der Methoden zur Berechnung der Länge eines Vektors und zur Addition zweier Vektoren. Gehen Sie dabei von einer Klasse aus, die das obige Interface unter Verwendung eines JAVA-Array umsetzt, bei der die Implementierungen der Methoden zur Dimension und Komponente vorgegeben sind.

Hinweis:
Die Vektoraddition erfolgt durch komponentenweise Addition und die Berechnung der Länge durch die Wurzel aus der Summe der Quadrate seiner Komponenten.

c) Geben Sie weiterhin eine Methode für die beschriebene abgeleitete Klasse an, die aus einem Vektor der Dimension *n* einen um *m* Dimensionen erweiterten Vektor erzeugt, wobei die neuen zusätzlichen Dimensionen durch Nullwerte aufgefüllt werden.

Aufgabe 2 (7 Punkte)

Gegeben sei folgender Algorithmus:

Algorithmus: pot(a, n)
Eingabe: a (reelle Zahl)
n ≥ 1 (ganze Zahl)
Ausgabe: a^n (reelle Zahl)

```
i ← 1
result ← a
while i < n do
   result ← a*result
   i ← i+1
return result
```

a) Geben Sie die Schleifeninvariante an und weisen Sie nach, dass es sich tatsächlich um eine Invariante handelt.

b) Weisen Sie nach, dass der Algorithmus terminiert.

Aufgabe 3 (7 Punkte)

Gegeben Sei die Implementierung eines Interface für den abstrakten Datentyp *List* mit folgenden Methoden:

Position first(), // gibt erstes Element von *List* zurück
Position last(), // gibt letztes Element von *List* zurück
Position before(Position p), // gibt Element zurück, das vor *p* liegt (**null**, falls p=first())
Position after(Position p), // gibt Element zurück, das hinter p liegt (**null**, falls p=last())
Position insertBefore(Position p, Object elem), // fügt neues Element *elem* vor Position *p* ein
Position insertAfter(Position p, Object elem). // fügt neues Element *elem* hinter Position *p* ein

Alle Methoden seien in einer doppelt verketteten Liste mit dem Aufwand O(1) realisiert.

Hinweis:
Sie können zur Vereinfachung annehmen, dass *DNode* (Knoten mit *getNext()*, *get*P*rev(), ...*) eine Implementierung von *Position* ist und dass die Listen nicht leer sind.

a) Erweitern Sie die obige Implementierung um die folgenden zwei Java-Methoden. Sie können zur Realisierung obige Methoden nutzen.

```
public Position elementAtRank(int i) throws InvalidRankException{
    // Rückgabe des Elementes an Rang i (Zählung beginnt mit 0)
    … }
public void insertAtRank(Object elem, int i) throws InvalidRankException {
    // Einfügen des Elementes elem an Rang i
    ... }
```

b) Bestimmen Sie den Aufwand für beide Methoden.

Aufgabe 4 (10 Punkte)

a) Wie ist ein regulärer binärer Baum definiert? Was versteht man unter der Höhe eines Baums?

b) Geben Sie in Pseudocode eine Methode zur Berechnung der Höhe eines binären Baumes an. Es stehen die folgenden Methoden zur Verfügung:
Position root(),
Position parent(Position p),
Position leftChild(Position p),
Position rightChild(Position p),
boolean isExternal(Position p),
boolean isInternal(Position p).

c) Wie hoch muss ein binärer Baum mit *n* Knoten mindestens sein? Begründen Sie Ihre Antwort!

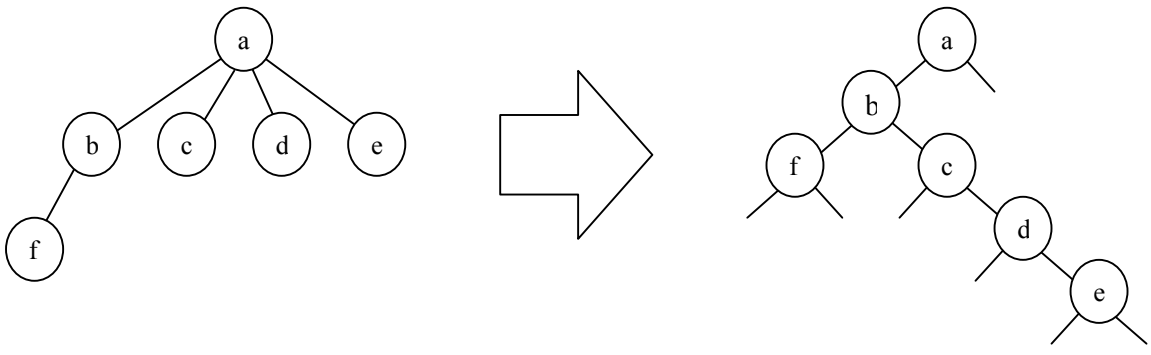
d) Geben Sie eine geeignete Datenstruktur für einen binären Baum an. Eine mögliche Implementierung auf Basis dieser Datenstruktur ist in Java zu skizzieren. Es sind Java-Methoden anzugeben, die zu einem Knoten die jeweiligen Kinder liefern.

Aufgabe 5 (10 Punkte)

Gegeben sei durch den *ADT Tree* ein n-ärer Baum und die folgenden Zugriffsmethoden:
Position root(),
PositionIterator children(Position p),
Position parent(Position p),
Object element(Position p).

a) Schreiben Sie eine Methode in Pseudocode, die diesen Baum traversiert.

b) Schreiben Sie in Anlehnung an die Traversierung in Pseudocode eine Methode, welche die Knoten des Baumes in einen binären Baum kopiert. Dabei sollen Elternknoten im n-ären Baum Vorfahren im binären Baum werden. Die Transformation soll dabei entsprechend folgender in der Vorlesung vorgestellten Vorgehensweise geschehen:
Für einen Knoten im n-ären Baum wird jeweils der erste Kindknoten ein linker Kindknoten im binären Baum (*b* im Beispiel). Alle weiteren Kindknoten des Knotens im n-ären Baum werden vom ersten Kindknoten (*b*) ausgehend jeweils rechte Kindknoten (*c, d* und *e* im Beispiel).



Nutzen Sie für die Realisierung den Abstrakten Datentyp (*ADT) ChangeableBinaryTree* mit folgenden Methoden:

ChangeableBinaryTree() *// Konstruktor,*
Position root(), Position parent(Position p),
Position leftChild(Position p),
Position rightChild(Position p),
boolean isInternal(Position p),
boolean isExternal(Position p),
boolean isRoot(Position p),
expandExternal(Position p) *// ersetzt externen Knoten durch Knoten mit 2 Kindern,*
replaceElement(Position p, Object e)

Aufgabe 6 (7 Punkte)

MergeSort (Sortieren durch Mischen) ist eine schnelle Methode zur Sortierung, die nach dem Divide-and-Conquer-Prinzip arbeitet.

a) Schreiben Sie eine Methode in Pseudocode, durch die *MergeSort* realisiert wird.

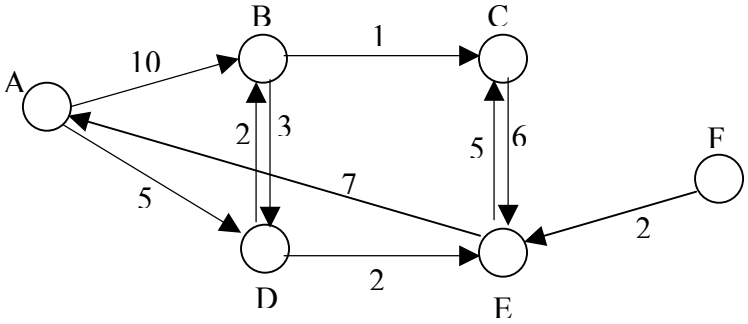
b) Was versteht man unter *stabilem* Sortieren. Ist *MergeSort* stabil? Begründen Sie Ihre Aussage!

c) Geben Sie den Aufwand für *MergeSort* im schlechtesten Fall in O-Notation an. Begründen Sie Ihre Angabe.

Aufgabe 7 (9 Punkte)

Gegeben sei der nachfolgend gezeichnete gerichtete Graph mit sechs Knoten und bewerteten Kanten *w(u,v).*

a) Zeichnen Sie am Beispiel dieses Graphen alle kürzeste Pfade vom Startknoten *A* zu allen anderen Knoten ein.



b) Was versteht man unter Erreichbarkeit eines Knoten? Wie kann man alle nicht erreichbaren Knoten ermitteln?

c) Wandeln Sie den *Dijkstra-Algorithmus* so ab, dass die kürzesten Pfade zu allen vom gegebenen Startknoten aus erreichbaren Knoten berechnet werden. Geben Sie den Algorithmus im Pseudocode an.

d) Begründen Sie, warum bei der Pfadsuche nach dem Dijkstra-Algorithmus die Kanten keine negative Bewertung haben dürfen.