Klausur Algorithmen und Datenstrukturen
Sommersemester 2007
9. 7. 2007
Forschungszentrum Jülich
Zentralinstitut für Angewandte Mathematik
Dr. Helmut Schumacher, Hans Joachim Pflug

Name:	
Matrikel-Nr.:	

			max.Punktzahl
Aufgabe	1)		(8)
Aufgabe	2)		(8)
Aufgabe	3)		(8)
Aufgabe	4)		(9)
Aufgabe	5)		(17)
Aufgabe	6)		(17)
Aufgabe	7)		(17)
Aufgabe	8)		(8)
Aufgabe	9)		(8)
maximale Summe:			100

Gesamtpunkte:	Note:

Die folgenden Hinweise bitte sorgfältig durchlesen!

- Zum Bestehen der Klausur sind **50 Punkte** notwendig!
- Zur Bearbeitung stehen Ihnen **2 Stunden** zur Verfügung.
- Was nicht bewertet werden soll, kennzeichnen Sie bitte durch Durchstreichen.
- Beginnen Sie jede Aufgabe auf einer neuen Seite.
- Tragen Sie auf jedem weiteren Blatt oben ihren Namen und ihre Matrikelnummer sowie die Nummer der Aufgabe, zu der dieses Blatt gehört, ein.
- Es sind (außer Schreibzeug) keinerlei Hilfsmittel erlaubt.
- Sollte ein Täuschungsversuch beobachtet werden, so gilt die Klausur als nicht bestanden.
- Bleiben Sie am Ende der Bearbeitungszeit bitte auf Ihrem Platz, bis die Aufsicht alle Klausurblätter (sowie etwaige Zusatzblätter) eingesammelt hat.
- Wenn Ihrer Meinung nach Angaben fehlen, machen Sie Annahmen und dokumentieren diese.

Aufgabe 1:

Gegeben ist ein Feld von 15 unsortierten Integer-Werten:

34, 13, 39, 14, 36, 49, 40, 37, 41, 15, 12, 16, 38, 21, 3

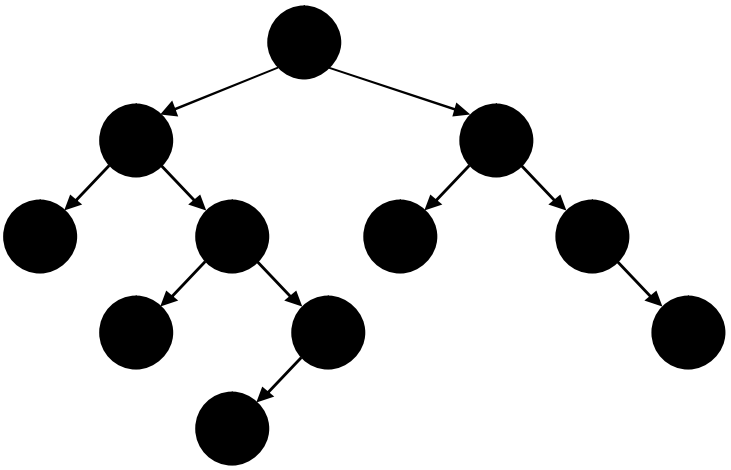
a) Sortieren Sie die Werte nach dem Quicksort-Verfahren. Kennzeichnen Sie die Pivot-Elemente und dokumentieren Sie die Vertauschungen. Verwenden Sie als Pivot-Element den „median of three“, d.h. das wertmäßig mittlere des ersten, des mittleren und des letzten Elements.
Für Teilfelder von 2 Elementen brauchen Sie den Quicksort-Algorithmus nicht mehr zu verwenden, sondern dürfen die Elemente gegebenenfalls einfach umtauschen.

34	***13***	***39***	***14***	***36***	***49***	***40***	***37***	***41***	***15***	***12***	***16***	***38***	***21***	***3***

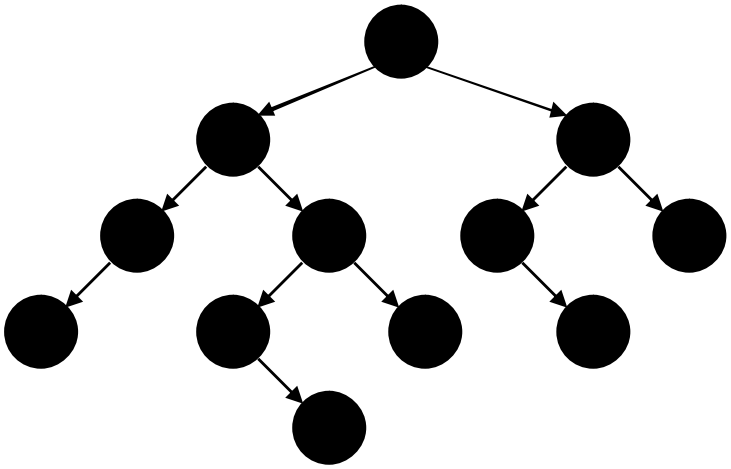
Aufgabe 2:

Geben Sie bei den folgenden Bäumen für jeden Knoten den Balance-Index an. Welcher dieser Bäume ist ein AVL-Baum? An welcher Stelle ist der Balance-Index verletzt?

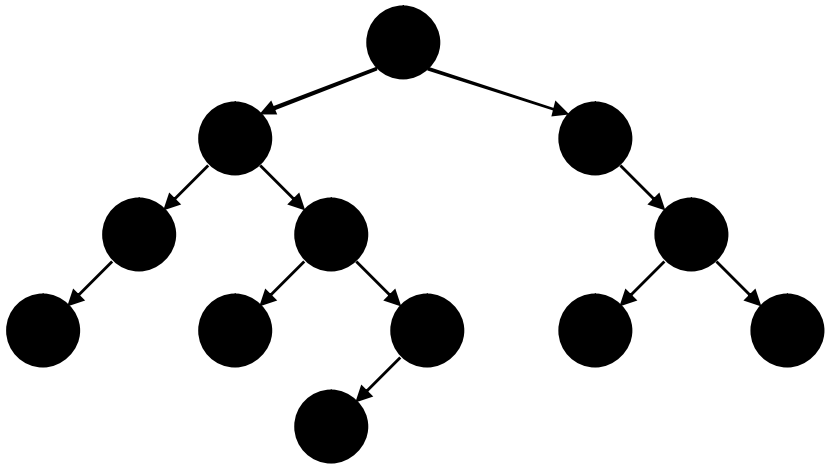
a)



b)



c)



Aufgabe 3:

Gegeben sei folgende Adjazenzliste:

A → C	35
A → D	4
B → C	5
D → C	30
D → E	2
D → F	4

E → B	12
E → C	26
E → F	1
F → B	10
F → C	24

a.) Zeichnen Sie den zugehörigen Graphen
b.) Bestimmen Sie nach der Methode von Dijkstra die kleinsten Entfernungen vom Knoten A zu allen anderen Knoten. Füllen Sie dazu die untenstehende Tabelle aus.

		d						p				
V_i		B	C	D	E	F		B	C	D	E	F

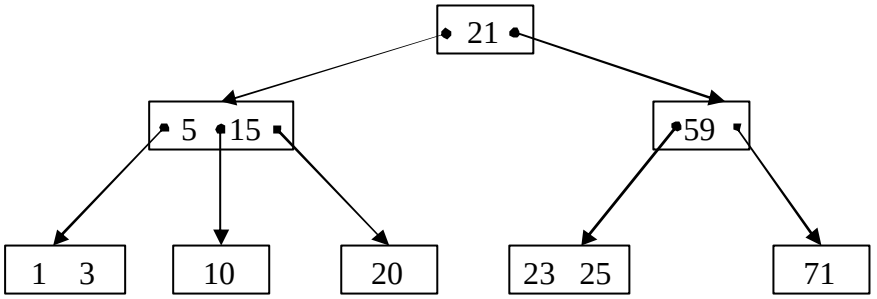
(Vorlage zum Entwurf. Bitte streichen Sie am Ende Ihren Entwurf komplett durch.)

		d						p				
V_i		B	C	D	E	F		B	C	D	E	F

Aufgabe 4:

Führen Sie die unter a) bis i) beschriebenen Einfüge- bzw. Löschoperationen aus und zeichnen Sie den resultierenden Baum auf.
Achtung: Die Bäume besitzen teilweise Ordnung 1 und teilweise Ordnung 2.

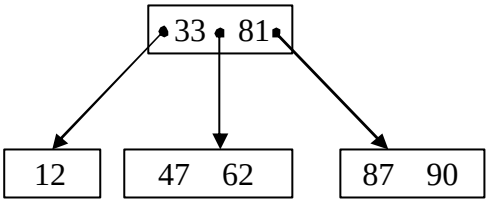
a) Gegeben sei folgender B-Baum der Ordnung 1:



Fügen Sie die **9** ein.

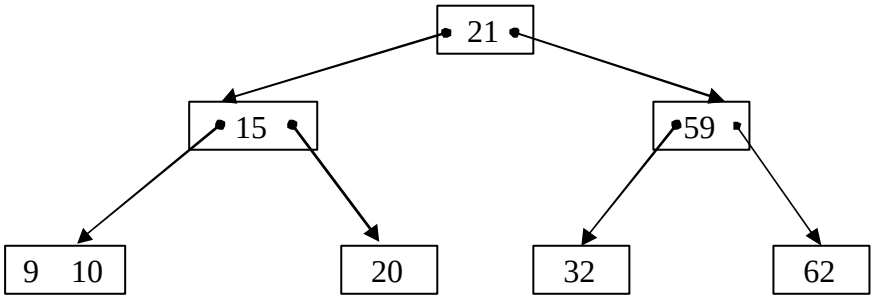
b) Fügen Sie in den unter a) abgebildeten Baum die **41** ein.

c) Gegeben sei folgender B-Baum der Ordnung 1:



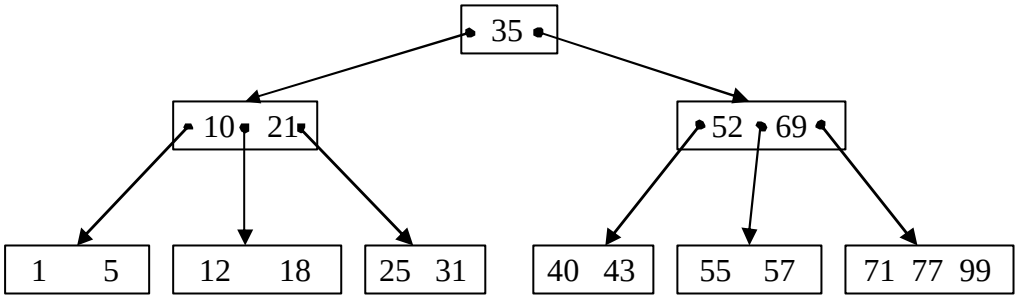
Fügen Sie die **82** ein.

d) Gegeben sei folgender B-Baum der Ordnung 1:



Entfernen Sie das Element **21**.

e) Gegeben sei folgender B-Baum der Ordnung **2**:

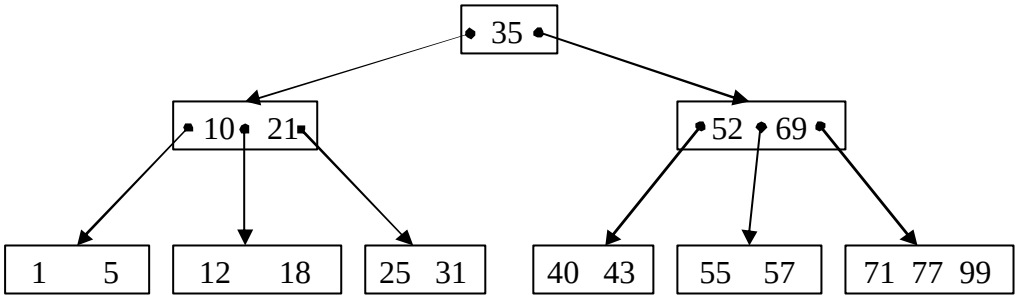


Löschen Sie die **77**.

f) Löschen Sie aus dem unter e) abgebildeten Baum die **55**.

g) Löschen Sie aus dem unter e) abgebildeten Baum die **25**.

h) Gegeben sei folgender B-Baum der Ordnung **2**:



Löschen Sie die **69**.

i) Löschen Sie aus dem unter h) abgebildeten Baum die **52**.

Aufgabe 5:

Gegeben sei ein Binärbaum, der beliebige Objekte aufnehmen soll.

a.) Schreiben Sie die Klasse *Node* für einen Knoten des Binärbaums.
b.) Eine Klasse *BinaryTree*, die einen Binärbaum darstellt, ist folgendermaßen gegeben:

```
public class BinaryTree {
  private Node root, current;
  public BinaryTree(Node root) {
    this.root = root;
    current = root;
  }
}
```

Fügen Sie einen zweiten Konstruktor
`public BinaryTree(BinaryTree b)`
hinzu, der eine Kopie des übergebenen Binärbaums erzeugt. Verwenden Sie zum Kopieren eine zusätzliche rekursive Methode.

Aufgabe 6:

Ein String bestehe aus einer Folge der Ziffern 0-9. Folgendes Gerüst ist gegeben:

```
public class Zahlenfolge {

     private String text;    //Die Zahlenfolge;

}
```

Fügen Sie die unten beschriebenen Methoden hinzu, die es ermöglichen, in der Zahlenfolge nach einer Teilfolge mit dem in der Vorlesung behandelten Boyer-Moore-Verfahren zu suchen.

```
//Gibt die skip-Tabelle fuer den String pat zurueck. Der Rueckgabe-
//wert hat 10 Elemente, die fuer die Zahlen 0-9 stehen.
private int[] getSkipTabelle(String pat)

//Ueberprueft, ob der String pat mit dem Teilstring von text ab
//der Position pos uebereinstimmt.
private boolean fits(String pat, int pos)

//Sucht den String pat nach dem Boyer-Moore-Verfahren und gibt die
//Index-Position des 1. Vorkommens zurueck. Kommt der String nicht
//vor, wird -1 zurueckgegeben.
public int findFirst(String pat)
```

Bei allen Methoden wird davon ausgegangen, dass die String-Übergabeparameter tatsächlich nur die Zahlen 0-9 enthalten.

Aufgabe 7:

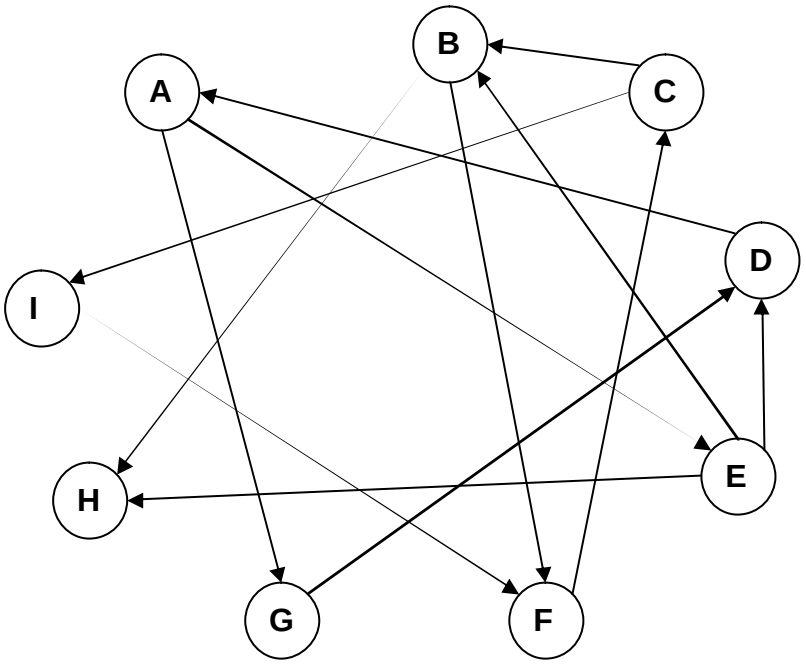
Schreiben Sie eine Klasse *MyStack*, die einen Stack für double-Werte darstellt. Der Stack soll durch ein Feld implementiert werden, das sich dynamisch vergrößert und verkleinert (keine verkettete Liste).
Falls beim Hinzufügen eines neuen Elements das Feld „überläuft“, soll es um das doppelte vergrößert werden. Falls nach dem Entnehmen eines Elements weniger als ein Drittel der Feldelemente besetzt ist, soll das Feld um die Hälfte verkleinert werden.
Falls aus einem leeren Stack ein Wert entnommen werden soll, wird eine *EmptyStackException* geworfen. Die Exception-Klasse muss nicht selbst geschrieben werden. Sie ist bereits im Paket *java.util* vorhanden.
Die Startgröße des Feldes ist 10.

Benutzen Sie folgendes Grundgerüst:

```
public class MyStack {

private double[] stack = new double[10]; //Feld fuer Stack
//evtl. weitere Attribute

//Fuegt ein Element hinzu
public void push(double d) {
  //zu implementieren
}

//Entfernt ein Element
public double pop() throws EmptyStackException {
  //zu implementieren
}

}
```

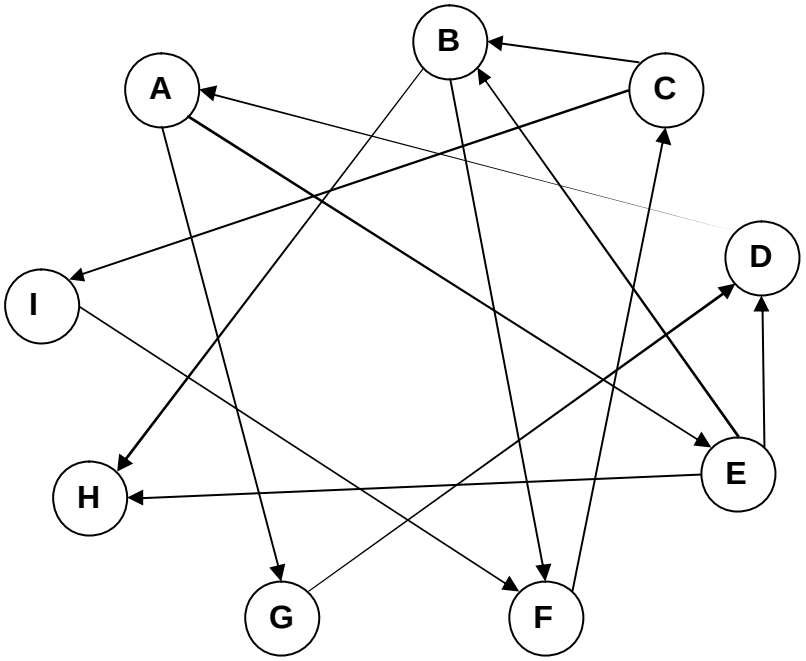
Aufgabe 8:

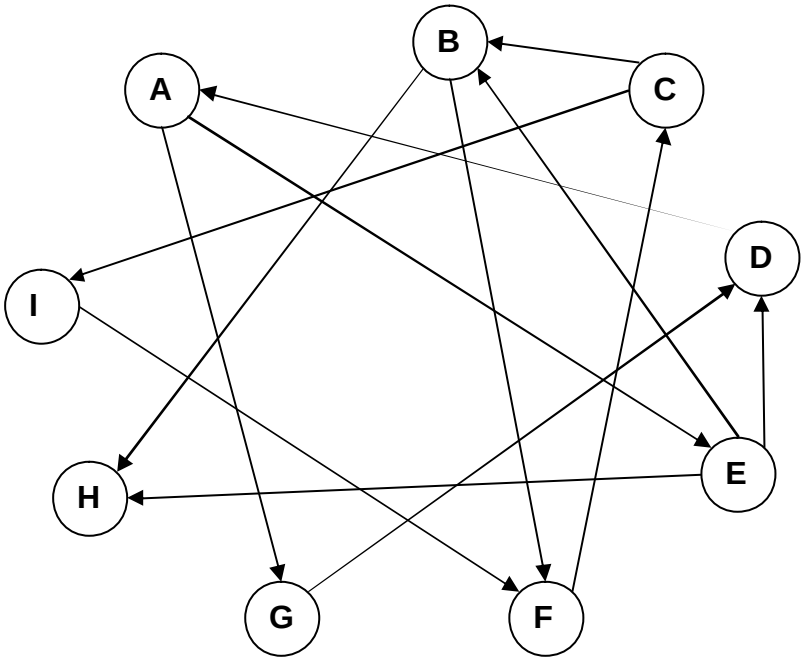
Ein gerichteter Graph sei wie folgt definiert:



a) Stellen Sie zu diesem Graph die Knotenliste auf.
b) Durchlaufen Sie den Graph nach der Tiefensuche und zählen Sie die Knoten in der Reihenfolge auf, in der Sie sie besuchen. Beginnen Sie beim Knoten A. Besuchen Sie die Knoten zuerst, die weiter vorne im Alphabet stehen.
c) Durchlaufen Sie den Graph nach der Breitensuche und zählen Sie die Knoten in der Reihenfolge auf, in der Sie sie besuchen. Beginnen Sie beim Knoten A. Besuchen Sie die Knoten zuerst, die weiter vorne im Alphabet stehen.

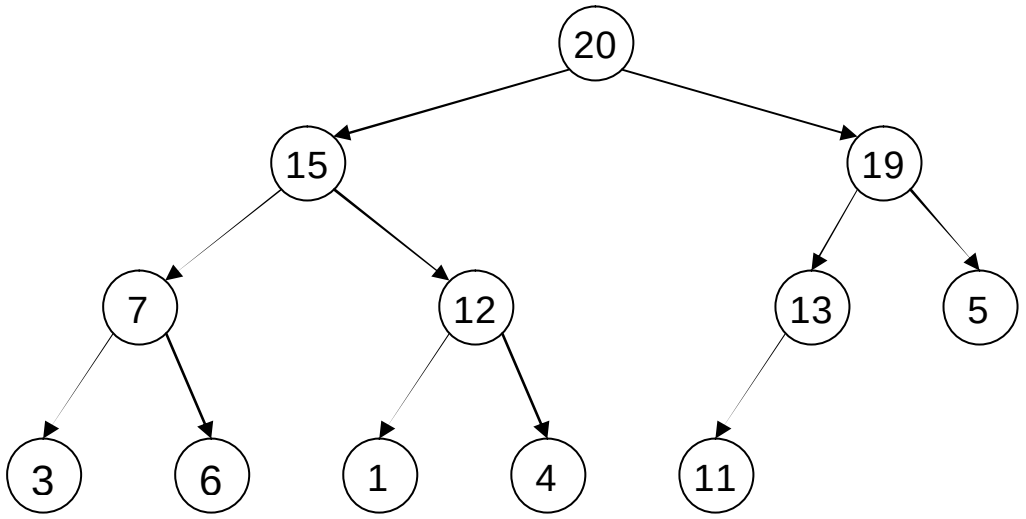
Auf der Rückseite befinden sich weitere Graphen als Vorlage.


B
A
C
D
I
E
H
G
F


B
A
C
D
I
E
H
G
F

Aufgabe 9:

Folgender Heap sei gegeben:



a) Entfernen Sie nacheinander 5 Mal das größte Element und zeichnen Sie nach jedem Schritt den neuen Heap-Baum.
b) Fügen Sie anschließend den Wert 10 ein und zeichen Sie den neuen Heap-Baum.