Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Simulation und Graphik
Prof. Dr. Holger Theisel

Magdeburg, 26.07.2011

Klausur „Algorithmen und Datenstrukturen“

Matrikelnummer:		
Name:	Vorname:	Studiengang:
Abschluss mit unbenotetem Schein:	☐ ja	☐ nein
Blattanzahl:	Unterschrift Student/in:	Unterschrift Aufsicht:

bitte **nicht** ausfüllen:

Aufg. 1: AVL-Bäume	**Aufg. 2:** Rot-Schwarz- und 2-3-4-Baum	**Aufg. 3:** Lauflängen-kodierung	**Aufg. 4:** Dynamische Programmierung	**Aufg. 5:** Hashing	**Aufg. 6:** Binärbaum
(von 6)	(von 4)	(von 7)	(von 4)	(von 4,5)	(von 5)

Aufg. 7: Traversierung	**Aufg. 8:** Dijkstra	**Aufg. 9:** Heapsort	**Aufg. 10:** Huffman - Kodierung	**Aufg. 11:** Wahr oder falsch?	**Aufg. 12:** Bonus-aufgabe	**Summe**
(von 2)	(von 4,5)	(von 3)	(von 3)	(von 5)	(von 2 Zusatzpkt.)	(von 48)

Hinweise für die Klausurbearbeitung

- Überprüfen Sie die Klausur auf Vollständigkeit (13 Blätter). Füllen Sie das Klausurdeckblatt gewissenhaft mit Namen, Vornamen und Studiengang aus. Schreiben Sie auf jedes Blatt Ihren Namen.
- Legen Sie alle für die Klausur benötigten Dinge, insbesondere Lichtbildausweis (Studentenausweis zur Anwesenheitskontrolle), Stifte, Verpflegung auf Ihren Tisch. Schalten Sie Ihre Handys aus.
- Verwenden Sie für Ihre Antworten den freien Platz nach den Aufgaben und ggf. die Rückseiten der Blätter. Melden Sie sich, wenn Sie zusätzliche leere Blätter benötigen.
- Benutzung roter / grüner Stiftfarbe und Bleistift ist untersagt. Geben Sie bei Rechenaufgaben vor jedem Schritt an, was Sie gerade ausrechnen. Bewertet wird der Rechenvorgang, das Endergebnis alleine reicht nicht.
- Die Benutzung unerlaubter Hilfsmittel (Taschenrechner, Bücher, Folien zur Vorlesung, Mobiltelefone) gilt als Täuschungsversuch und führt zu einer Bewertung der Prüfung mit „nicht ausreichend“ (5,0).
- Schreiben Sie deutlich! Unleserliche Passagen können nicht korrigiert werden.
- Beschränken Sie sich auf die geforderten Angaben und halten Sie Ihre Antworten kurz und präzise. Nicht geforderte Angaben ergeben keine zusätzlichen Punkte.

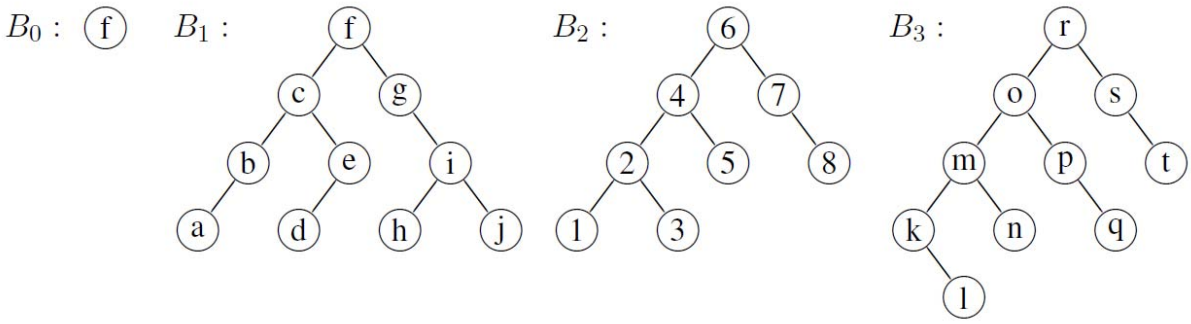
Viel Erfolg!

Name:_____

Aufgabe 1: AVL-Bäume (6 Punkte)

a) Durch welche Eigenschaften ist ein AVL-Baum gekennzeichnet? Welchen Vorteil hat ein AVL-Baum gegenüber einem einfachen binären Suchbaum?

b) Welche der vier Bäume B_0, B_1, B_2 und B_3 sind AVL-Bäume, welche nicht? Begründen Sie kurz Ihre Antwort.



c) Fügen Sie die Zahlen 2, 6, 7, 4, 3, 5, 1 (in dieser Reihenfolge) in einen zunächst leeren AVL-Baum ein. Skizzieren Sie den Baum nach jedem Einfügen eines Elementes.

Name:_____

Aufgabe 2*:* Rot-Schwarz- und 2-3-4-Baum (4 Punkte)

a) Fügen Sie die Zahlen 5, 6, 7, 1, 2, 3, 4 (in dieser Reihenfolge) in einen zunächst leeren Rot-Schwarz-Baum ein. Skizzieren Sie den Baum nach jedem Einfügen eines Elementes.
b) Wie sehen die sieben entsprechenden 2-3-4-Bäume aus?

Name:_____

Aufgabe 3: Lauflängenkodierung (7 Punkte)

Die Lauflängenkodierung (Run-Length Encoding) ist ein sehr einfacher verlustfreier Kompressionsalgorithmus für digitale Daten bei dem Wiederholungen von gleichen Zeichen verkürzt dargestellt werden.
Dabei wird jede Teilfolge, die aus mehr als zwei gleichen Zeichen besteht, durch ihre Länge und dem Zeichen kodiert.
Die Eingabe *"ABBCCCKKKKKKKKKKK"* wird zu *"ABB3C12K"*.

Schreiben Sie eine Java-Methode

public static String kompr(String a),

um einen gegebenen String nach diesem Verfahren zu kodieren.

Hinweis:
Aus der Klasse *String* dürfen Sie die Funktionen *public char charAt(int index)* und *public int length()* nutzen.

Name:_____

Aufgabe 4: Dynamische Programmierung (4 Punkte)

a) Schreiben Sie eine Java-Methode

public static int f (int a, int b),

die die durch die Rekursionsformel

$f(a, b) = f(a-1, b) + 2*f(a, b-1)$ mit $f(a, 0) = f(0, b) = 1$,

definierte Funktion *f* mit Hilfe der **dynamischen** Programmierung berechnet.

b) Begründen Sie kurz (anhand der Komplexität in *O*-Notation), wieso diese Lösung mittels dynamischer Programmierung effizienter ist als eine entsprechende rekursive Lösung.

Name:_____

Aufgabe 5: Hashing (4,5 Punkte)

Gegeben sind die unten aufgeführten Hashtabellen mit den angegebenen Hashfunktionen *h(v).* Die Hashtabellen wurden mit den angegebenen Werten in der Reihenfolge von links nach rechts gefüllt. Bestimmen Sie jeweils, ob beim Füllen der Hashtabelle zur Kollisionsbehandlung jeweils entweder

- lineare Sondierung
  $H(v,i) = (h(v) + i) \bmod 5$,
- lineare Sondierung mit alternierendem Vorzeichen
  $H(v,i) = (h(v) + (-1)^i * i) \bmod 5$
- oder quadratische Sondierung
  $H(v,i) = (h(v) + i^2) \bmod 5$

benutzt wurde.

Hinweise:

- Sondiertechniken können mehrfach vorkommen.
- Für negative x gilt *x mod n = (n+x) mod n*

a) Werte = [17, 7, 12, 20, 11], *h(v) = (2∗v + 1) mod 5*, Hashtabelle = [17, 20, 12, 11, 7]

b) Werte = [12, 7, 10, 15, 18], *h(v) = 3∗v mod 5*, Hashtabelle = [10, 12, 7, 18, 15]

c) Werte = [3, 13, 18, 20, 12], *h(v) = (4∗v + 2) mod 5*, Hashtabelle = [13, 12, 20, 18, 3]

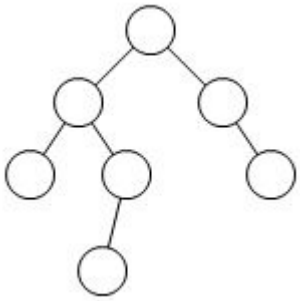
Name:_____

Aufgabe 6: Binärbaum (5 Punkte)

Erweitern Sie die Klasse BinaryTree<T> um die Methode

```
public int maxWidth(),
```

die in einem gegebenen binären Baum die größte Breite (d.h. die maximale Anzahl von Knoten in der gleichen Ebene) zurückgibt.
Für folgenden Binärbaum ist drei die größte Breite.



Hinweise:

- Die Klasse TreeNode<T> steht Ihnen mit folgenden Methoden zur Verfügung:

```
public TreeNode();
public boolean isNullNode();
public boolean hasLeftChild();
public boolean hasRightChild();
public TreeNode<T> getLeft();
public TreeNode<T> getRight();
```

- Die Klasse BinaryTree<T> steht Ihnen mit folgenden Methoden zur Verfügung:

```
public BinTree();
public boolean isEmpty();
public int height();
public TreeNode<T> getRoot();
```

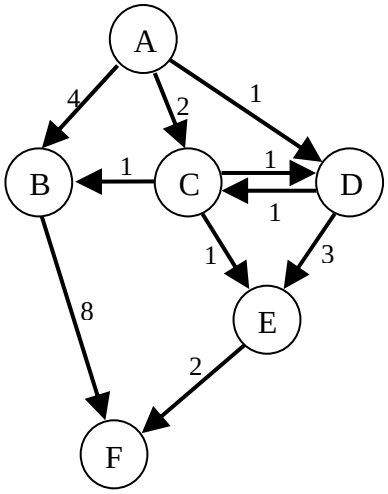
Name:_____

Aufgabe 7: Traversierung (2 Punkte)

Zeigen Sie, dass es keinen binären Suchbaum mit den fünf Knoten *a, b, c, d* und *e* geben kann, der sowohl in Inorder- als auch in Postorder- als auch in Preorder-Traversierung die gleiche Reihenfolge *a, b, c, d, e* der Knoten ausgibt.

Name:_____

Aufgabe 8: Dijkstra (4,5 Punkte)



a) Geben Sie den Graphen in den folgenden Formen an:
Adjazenzmatrix:

Adjazenzliste:

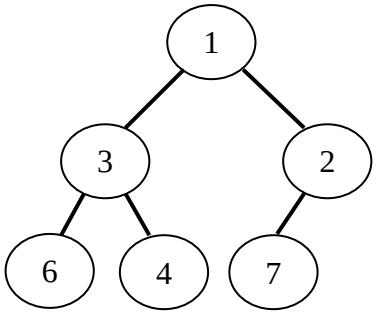
Knotenliste:

b) Berechnen Sie für den gegebenen Graphen die Entfernungen aller Knoten zum Startknoten *A*. Nutzen Sie dafür den Dijkstra-Algorithmus. Geben Sie dafür die einzelnen Zustände der Prioritätswarteschlange an.

Name:_____

Aufgabe 9: Heapsort (3 Punkte)

Sortieren Sie mittels des Heapsort-Verfahrens. Zeichen Sie für jeden Schritt den zu veränderten Min-Heap.



Name:_____

Aufgabe 10: Huffman-Kodierung (3 Punkte)

Erstellen Sie die Huffman Kodierung für den String „abcdefabcdeabcdabcaba“.

Name:_____

Aufgabe 11: Wahr oder falsch? (5 Punkte, wenigstens 0 Punkte)

Kreuzen Sie jeweils (ohne Begründung!) an, ob die Aussage *"wahr"* oder *"falsch"* ist. Für jede richtige Antwort gibt es einen halben Punkt, für jede falsche Antwort wird ein halber Punkt abgezogen. Beantworten Sie also lieber nur Fragen, bei denen Sie sich sicher sind!

	ja	nein
1. Genetische Algorithmen sind nicht deterministisch.		
2. In einem Heap steht das größte Element stets in der untersten Ebene am weitesten rechts.		
3. Jeder Rot-Schwarz-Baum ist ein AVL-Baum.		
4. Hashfunktionen müssen stets streng monoton wachsend sein.		
5. Topologisches Sortieren eines azyklischen gerichteten Graphen ist nicht eindeutig, es kann also verschiedene Lösungen geben.		
6. Wird ein Graph als Adjazenzmatrix dargestellt, so ist der Aufwand zum Löschen einer Kante unabhängig von der Größe des Graphen.		
7. Für gerichtete Graphen mit ausschließlich positiven Gewichten liefern der Dijkstra-Algorithmus und der Bellman-Ford-Algorithmus stets das gleiche Ergebnis.		
8. Die Entropie eines Binärbildes (d.h. ein Bild, dessen Pixel nur die Farben "schwarz" und "weiß" beinhalten) ist $\log_2\left(\frac{1}{2}\right)$.		
9. Beim Fünf-Philosophen-Problem können maximal zwei Philosophen gleichzeitig essen.		
10. Ein Semaphor eines Prozesses mit mehreren Threads kann nicht gleichzeitig verschiedene Zustände in verschiedenen Threads haben.		

Name:_____

Bonus-Aufgabe: Minimaler Spannbaum mit Bellman-Ford-Algorithmus (2 Punkte)

Ein *minimaler Spannbaum M* ist ein Baum mit folgenden Eigenschaften:

- *M* ist ein zyklenfreier Teilgraph eines gewichteten (und gerichteten) Graphen *G*.
- *M* enthält alle Knoten *V* von *G*.
- Die Pfade in *M* ausgehend von einem ausgezeichneten Startknoten *s* zu allen anderen Knoten des Baumes entsprechen den *kürzesten Pfaden* von *s* zu allen Knoten im Graphen *G*.

Erweitern Sie den Algorithmus von Bellman-Ford (siehe *Algorithm 1*) zur Bestimmung kürzester Pfade in Graphen so, dass für einen gegebenen Startknoten *s* der minimale Spannbaum in *G* extrahiert werden kann. Sie können dazu zusätzliche Attribute in den Elementen des Graphen speichern.

Algorithm 1 BellmanFord (Vertices $\mathcal{V}$, Edges $\mathcal{E}$, Vertex s)

```
1: for vertex v in V do {Step1: initialize graph}
2:   if v is s then
3:     v.distance = 0
4:   else
5:     v.distance = ∞
6:   end if
7: end for
8: for i from 1 to |V| − 1 do {Step 2: relax edges repeatedly}
9:   for edge e_uv in E do {e_uv is the edge from u to v}
10:    u = e_uv.source
11:    v = e_uv.destination
12:    if u.distance + e_uv.weight < v.distance then
13:      v.distance = u.distance + e_uv.weight
14:    end if
15:  end for
16: end for
17: for edge e_uv in E do {Step 3: check for negative-weight cycles}
18:   u = e_uv.source
19:   v = e_uv.destination
20:   if u.distance + e_uv.weight < v.distance then
21:     print "Error: Graph contains a negative-weight cycle."
22:   end if
23: end for
```