Klausur Algorithmen und Datenstrukturen

Erstprüfer: Dr.-Ing. Marius Feldmann
Zweitprüfer: Prof. Dr.-Ing. habil. Heiko Vogler

4. Februar 2019, 14:50–16:20 Uhr

Ihre Daten – unverzüglich in Blockschrift ausfüllen –

Vorname: .....

Nachname: .....

Matrikelnummer: ..... Fachsemester: .....

Studiengang: ☐ B.Sc. Inf. ☐ B.Sc. Med.inf. ☐ Dipl.-Inf.

☐ Dipl.-IST ☐ Lehramt Inf. ☐ anderer: .....

Hinweise zur Bearbeitung

- Es sind keine Hilfsmittel zugelassen.
- Tragen Sie alle Lösungen in die dafür vorgesehenen Bereiche ein; im Ausnahmefall können Sie die leere(n) Reserveseite(n) am Ende nutzen.
- Sie können die karierten Blätter für Notizen oder Nebenrechnungen verwenden; diese werden *nicht* zur Korrektur herangezogen!
- Geben Sie am Ende *nur die Aufgabenblätter* ab.

Bewertung – von den Korrekteuren auszufüllen –

1	Programmieren in C	/ 18
2	Pulsierender Speicher	/ 17
3	Heapsort	/ 9
4	AVL-Bäume	/ 13
5	Tiefen- und Breitensuche	/ 14
6	Algebraisches Pfadproblem	/ 18
7	EBNF & Syntaxdiagramme	/ 11
$\sum$	Summe	/100

....

Erstprüfer Zweitprüfer

1 Programmieren in C

Punkte	
(a)	/ 6
(b)	/ 6
(c)	/ 6
$\sum$	/18

(a) Es soll geprüft werden, ob eine Zeichenkette, welche als Array von Symbolen (`char`) vorliegt, ein Palindrom ist. Eine Zeichenkette ist ein *Palindrom*, wenn sie von vorne und von hinten gelesen gleich ist. Vervollständigen Sie die Funktion `palindrom`, so dass 1 zurückgegeben wird, falls die Zeichenkette `a` der Länge `l` ein Palindrom ist, und sonst 0 zurückgegeben wird.

```
int palindrom(char a[], int l) {




}
```

(b) Gegeben sei der folgende Datentyp zur Darstellung von Listen mit Werten vom Typ `int`:

```
typedef struct elem* list;
struct elem {
  int key;
  list next;
};
```

Geben Sie eine Funktion `void append(list *lp, int n)` an, welche ein neues Element e mit Schlüsselwert `n` erzeugt, dieses hinten an eine gegebene Liste `*lp` anhängt, und den Nachfolger von e auf `NULL` setzt. Gehen Sie davon aus, dass `lp != NULL` ist.

```
void append(list *lp, int n) {




}
```

(c) Gegeben sei der folgende Datentyp zur Darstellung von Binärbäumen mit Schlüsselwerten vom Typ `int`.

```
typedef struct node* tree;
struct node {
  int key;
  tree left, right;
};
```

Schreiben Sie eine Funktion `treeToList`, welche aus einem beliebigen Binärbaum t des oben genannten Typs eine Liste l der geraden Schlüsselwerte (`key`) von t generiert. Dabei soll l die Schlüsselwerte in folgender Reihenfolge enthalten:

- zunächst die geraden Schlüsselwerte im linken Teilbaum von t,
- dann gegebenenfalls den Schlüsselwert am Wurzelknoten, und
- zuletzt die geraden Schlüsselwerte im rechten Teilbaum von t.

Geben Sie alle dazu erforderlichen Variablendeklarationen und einen **Aufruf** Ihrer Funktion an.

Hinweis: Nutzen Sie die Funktion `append` aus Teilaufgabe **(b)**.

2 Pulsierender Speicher

Punkte	
(a)	/ 5
(b)	/12
$\sum$	/17

```
1  #include <stdio.h>
2
3  int z = 0;
4  void g(int*, int);
5
6  void f(int* a, int b) {
7    /* label1 */
8    if (b == 0)
9      *a = 1;
10   else {
11     f(&z, b - 1);     /* $1 */
12     /* label2 */
13     g(&z, z);         /* $2 */
14     /* label3 */
15     *a = b - z;
16   }
17 }
18
19 void g(int* c, int d) {
20   /* label4 */
21   if (d == 0)
22     *c = 0;
23   else {
24     g(&z, d - 1);    /* $3 */
25     /* label5 */
26     f(&z, z);        /* $4 */
27     /* label6 */
28     *c = d - z;
29   }
30 }
31
32 int main() {
33   int x, y;
34   x = 2;
35   /* label7 */
36   f(&y, x);         /* $5 */
37   /* label8 */
38   printf("%d\n", y);
39   return 0;
40 }
```

Objektname	Gültigkeitsbereich

(a) Tragen Sie den Gültigkeitsbereich jedes Objektes in die Tabelle neben dem Programm ein. Nutzen Sie dazu die Zeilennummern.

(b) Führen Sie jedes der drei folgenden Speicherbelegungsprotokolle um jeweils vier Schritte weiter. Dokumentieren Sie die aktuelle Situation beim Passieren der Marken (*label1* bis *label8*). Geben Sie jeweils den Rücksprungmarkenkeller und die *sichtbaren* Variablen mit ihrer Wertebelegung an. Die Inhalte von Speicherzellen nicht sichtbarer Variablen müssen Sie nur bei Änderungen eintragen. Die bereits festgelegten Rücksprungmarken sind *$1* bis *$5*.

Haltepunkt	RM (wächst nach links)	Umgebung 1	2	3	4	5	6	7	8	9	10	11
label7	–	z 0	x 2	y ?								

Haltepunkt	RM (wächst nach links)	Umgebung 1	2	3	4	5	6	7	8	9	10	11
label4	2 : 1 : 5	z 1	2	?	3	2	1	1	c 1	d 1		

Haltepunkt	RM (wächst nach links)	Umgebung 1	2	3	4	5	6	7	8	9	10	11
label5	2 : 5	z 0	2	?	3	2	c 1	d 1				

3 Heapsort

Punkte	
$\sum$	/9

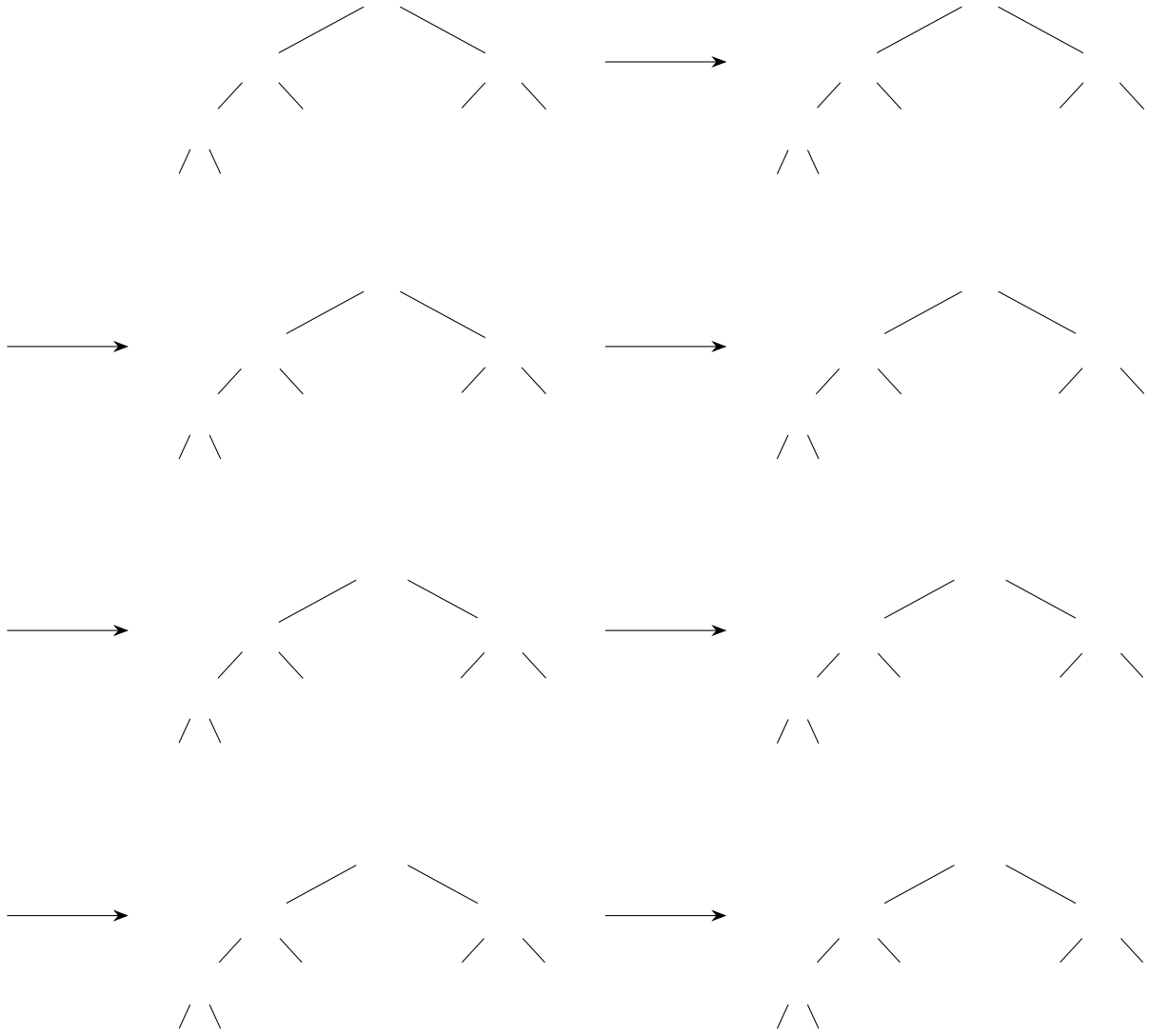
Gegeben sei die Folge 4, 2, 7, 1, 3, 11, 9, 6, 15.

Wenden Sie auf diese Folge den Heapsort-Algorithmus an.

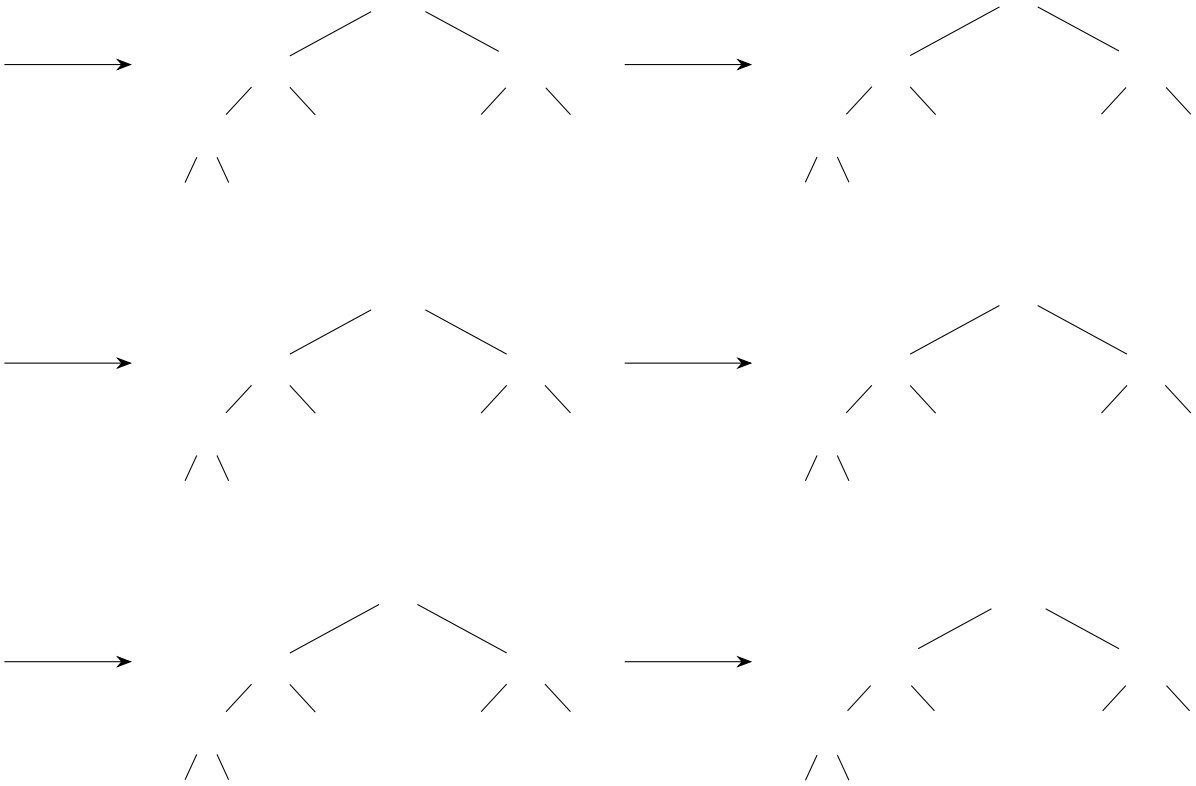
Dokumentieren Sie dazu in der Phase 1 das schrittweise Herstellen der Heap-Eigenschaft und dabei insbesondere die Veränderungen durch die Funktion „sinkenlassen“. Das Sinkenlassen in mehreren *unabhängigen* Teilbäumen darf in einem Schritt protokolliert werden.

In der Phase 2 brauchen Sie nur **zwei** Sortierschritte auszuführen. Ein Sortierschritt besteht aus einem Tausch- und einem Sinkenlassen-Schritt, die jeweils einzeln zu notieren sind.

Verwenden Sie die untenstehenden Vordrucke zur Dokumentation des Ablaufs des Algorithmus. Möglicherweise benötigen Sie nicht alle Vordrucke.



(weitere Vordrucke auf der nächsten Seite)



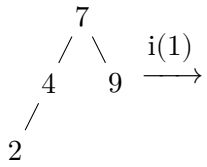
4 AVL-Bäume

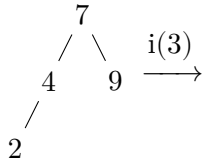
Punkte	
$\sum$	/13

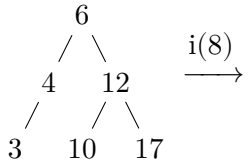
Fügen Sie in die folgenden AVL-Bäume den jeweils angegebenen Schlüssel ein. Ein einzufügender Schlüssel x wird durch i(x) notiert. Stellen Sie nach jedem Einfügen die AVL-Eigenschaft her und dokumentieren Sie hierbei die vom Einfüge- und Balancierungsalgorithmus ausgeführten Operationen. Nutzen Sie dabei die folgenden Abkürzungen:

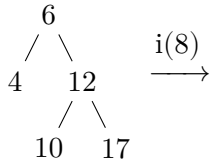
- L(x) – für die Linksrotation um den Knoten mit dem Schlüsselwert x,
- R(x) – für die Rechtsrotation um den Knoten mit dem Schlüsselwert x.

Geben Sie unmittelbar nach jedem i(x)-Schritt die Balancefaktoren für alle relevanten Knoten auf dem Pfad von x zur Wurzel an.





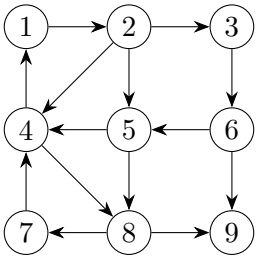




5 Tiefen- und Breitensuche

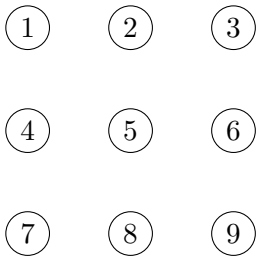
Punkte	
(a)	/ 8
(b)	/ 6
$\sum$	/14

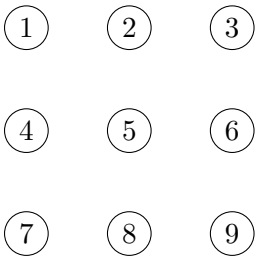
Der gerichtete Graph G sei durch folgende Darstellung gegeben:

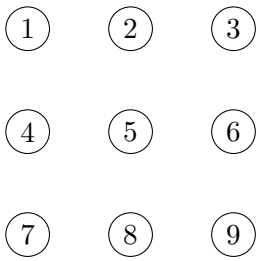


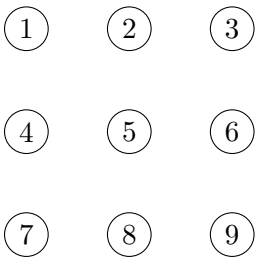
In den folgenden Teilaufgaben müssen Sie lediglich Kanten in den Vordruck eintragen. Zwischenschritte zu den Lösungen brauchen Sie nicht anzugeben.

(a) Wenden Sie auf G wiederholt den DFS-Algorithmus mit dem **Startknoten 2** an und bestimmen Sie auf diese Weise **vier** unterschiedliche depth-first-trees.

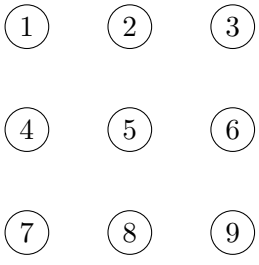


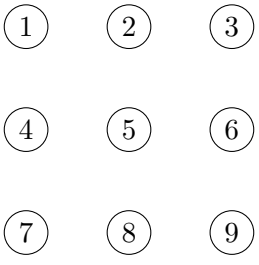


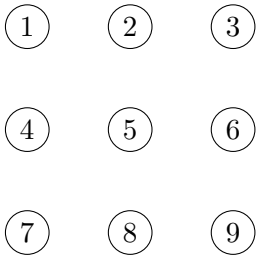




(b) Wenden Sie auf G wiederholt den BFS-Algorithmus mit dem **Startknoten 2** an und bestimmen Sie auf diese Weise **drei** unterschiedliche breadth-first-trees.



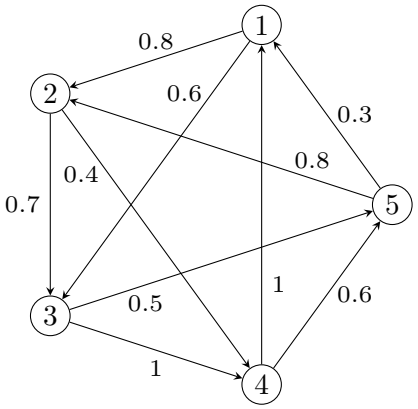




6 Algebraisches Pfadproblem

Punkte	
(a)	/ 5
(b)	/ 6
(c)	/ 1
(d)	/ 6
$\sum$	/18

Gegeben ist der Viterbi-Semiring $([0,1], \max, \cdot, 0, 1)$ und der gewichtete Graph G über dem Viterbi-Semiring in der nebenstehenden Abbildung.



(a) Geben Sie die modifizierte Adjazenzmatrix von G vollständig an.

$$mA_G = \begin{pmatrix} \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \\ \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots & \ldots\ldots \end{pmatrix}$$

(b) Die Matrix $D_G^{(2)}$ ist gegeben. Vervollständigen Sie die Einträge der Matrix $D_G^{(3)}$!

$$D_G^{(2)} = \begin{pmatrix} 1 & 0.8 & 0.6 & 0.32 & 0 \\ 0 & 1 & 0.7 & 0.4 & 0 \\ 0 & 0 & 1 & 1 & 0.5 \\ 1 & 0.8 & 0.6 & 1 & 0 \\ 0.3 & 0.8 & 0.56 & 0.32 & 1 \end{pmatrix} \qquad D_G^{(3)} = \begin{pmatrix} 1 & 0.8 & 0.6 & \ldots\ldots & \ldots\ldots \\ 0 & 1 & 0.7 & \ldots\ldots & \ldots\ldots \\ 0 & 0 & 1 & 1 & 0.5 \\ 1 & 0.8 & 0.6 & 1 & \ldots\ldots \\ 0.3 & 0.8 & 0.56 & \ldots\ldots & 1 \end{pmatrix}$$

(c) Welche Änderungen ergeben sich in der Berechnung von $D_G^{(5)}$ im Vergleich zu $D_G^{(4)}$?

(d) Gegeben sei das Alphabet $\Sigma = \{a, b, c\}$, der Semiring der formalen Sprachen $(\mathcal{P}(\Sigma^*), \cup, \cdot, \emptyset, \{\varepsilon\})$ über dem Alphabet Σ und die Matrix $D^{(2)}_{G'}$ eines gewichteten Graphen G' über diesem Semiring. Füllen Sie die fehlenden Einträge der Matrix $D^{(3)}_{G'}$ aus!

$$D^{(2)}_{G} = \begin{pmatrix} \{\varepsilon\} & \emptyset & \{a\} \\ \emptyset & \{\varepsilon\} & \{d\} \\ \emptyset & \{c\} & \{\varepsilon, b, cd\} \end{pmatrix}$$

$$D^{(3)}_{G} = \begin{pmatrix} \{\varepsilon\} & \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\ \emptyset & \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \\ \emptyset & \ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots \end{pmatrix}$$

7 EBNF & Syntaxdiagramme

Punkte	
(a)	/ 6
(b)	/ 5
$\sum$	/11

(a) Gegeben sei die Sprache

$$L = \{(ab)^{n+1}a^m c^n (aa)^k bb (cc)^k \mid n, m \geq 0, k \geq 1\}.$$

Geben Sie ein System von Syntaxdiagrammen $\mathcal{U}$ an, das die Sprache L erzeugt. Notieren Sie den Namen des Startdiagramms von $\mathcal{U}$.

Startdiagramm:

(b) Gegeben sei die EBNF-Definition $\mathcal{E} = (V, \Sigma, A, R)$ mit

$$V = \{S, A, B\},$$
$$\Sigma = \{a, b, c, d\} \qquad \text{und}$$
$$R = \{\ S ::= A\,\hat{[}B\hat{]},\qquad A ::= \hat{\{}\,A\,\hat{(}ab\hat{|}c\hat{)}\,\hat{\}},\qquad B ::= d\,\hat{[}\,B\hat{]}\,c\ \}.$$

Übersetzen Sie $\mathcal{E}$ gemäß der Übersetzungsvorschrift *trans* aus der Vorlesung in ein System von Syntaxdiagrammen und geben Sie das Startdiagramm an. Sie müssen *keine Zwischenschritte* angeben.

Startdiagramm: