Klausur Algorithmen und Datenstrukturen

Wintersemester 2017

11. April 2017

Ihre Matrikelnummer:

Achtung: Geben Sie keine weiteren persönlichen Daten außer der Matrikelnummer an!

Genereller Hinweis: Sie können selbstverständlich auch Zwischen- und Nebenergebnisse in Ihre Abgabe schreiben, um leichter zum Ergebnis zu kommen. Stellen Sie dann aber die eigentliche Lösung klar heraus.

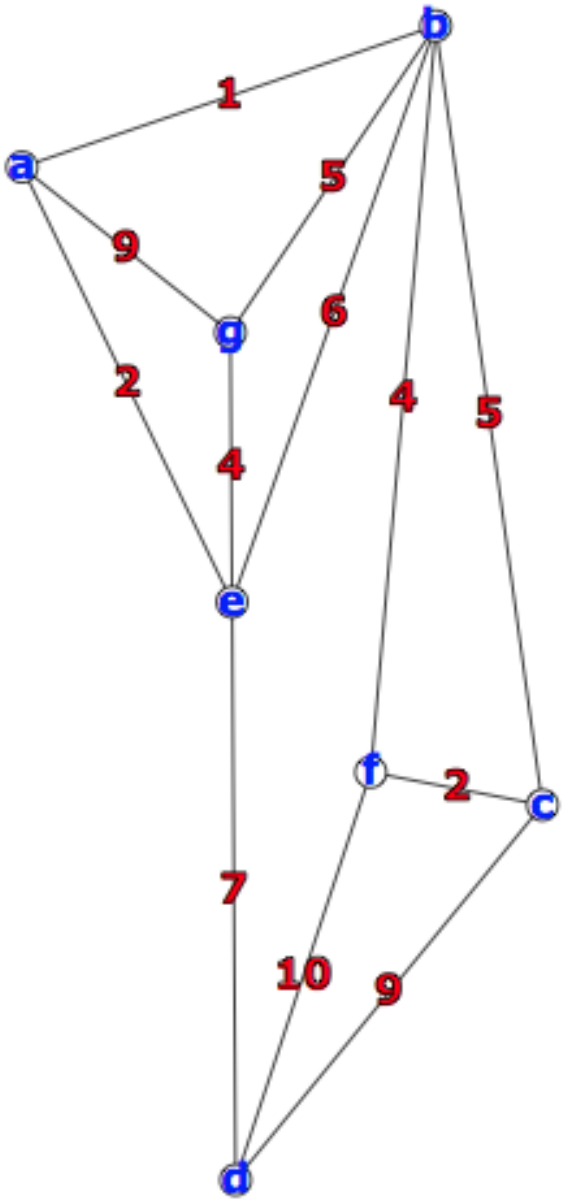
Diese Seite ab hier nur von den Korrektoren auszufüllen!

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**Summe**
Maximale Punktzahl	**10**	**10**	**10**	**10**	**15**	**15**	**15**	**15**	**100**
Erreichte Punktzahl									

Aufgabe 1: 10 Punkte

In der Vorlesung haben Sie den Floyd-Warshall-Algorithmus zur Berechnung der kürzesten Pfadpaare kennen gelernt.

Gegeben sei nun folgender Graph:



Geben Sie zum Iterationszeitpunkt i := 5 die Matrix an. Geben Sie unendlich, falls nötig, mit ∞ oder INF an. Verwenden Sie die folgende Liste von Knoten: $U = \{f, d, e, g, c, a, b\}$.

	a	b	c	d	e	f	g
a							
b							
c							
d							
e							
f							
g							

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Freie Seite für Ihre Nebenrechnungen (werden nicht gewertet):

Aufgabe 2: 10 Punkte

In der Vorlesung haben Sie das Verfahren Double Hashing als typisches Design für Hashfunktionen kennengelernt:

$$F(i, 18, K) := (F_1(18, K) + (i-1) \cdot F_2(18, K)) \bmod 18$$
$$F_1(N_{\max}, K) := (K \bmod N_{\max})$$
$$F_2(N_{\max}, K) := K + (K \bmod N_{\max})$$

Fügen Sie die folgenden Werte in dieser Reihenfolge ein: 16, 28, 51, 24, 38, 9, 23, 41, 13, 20.

Ihr Ergebnis:

0	9
1	10
2	11
3	12
4	13
5	14
6	15
7	16
8	17

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Freie Seite für Ihre Nebenrechnungen (werden nicht gewertet):

Aufgabe 3: 10 Punkte

In der Vorlesung haben Sie den Sortieralgorithmus „Quicksort“ und damit verbunden das Verfahren „pivot partitioning by scanning“ kennen gelernt. In dieser Aufgabe sollen Sie nun den Zustand des Algorithmus nach einer bestimmten Iteration angeben. Sei nun folgende Liste gegeben:

Index	0	1	2	3	4	5	6	7	8	9	10	11
Wert	-6	-13	15	-10	1	-14	-3	10	4	-5	3	7

Index	12	13	14	15	16	17	18	19	20	21	22
Wert	2	-8	-15	-10	-8	0	-8	-14	11	9	9

Sei außerdem das Pivotelement p:= -1.

Geben Sie den Zustand der Liste nach der Iteration i:=5 an. Geben Sie außerdem die Zeigerpositionen i1, i2 und i3 an.

Hinweis: Die Indizes beginnen bei 0.

Eingabe Ergebnisliste und Zeiger:

Index	0	1	2	3	4	5	6	7
Ergebnisliste								

Index	8	9	10	11	12	13	14	15
Ergebnisliste								

Index	16	17	18	19	20	21	22
Ergebnisliste							

Zeiger i1: _____

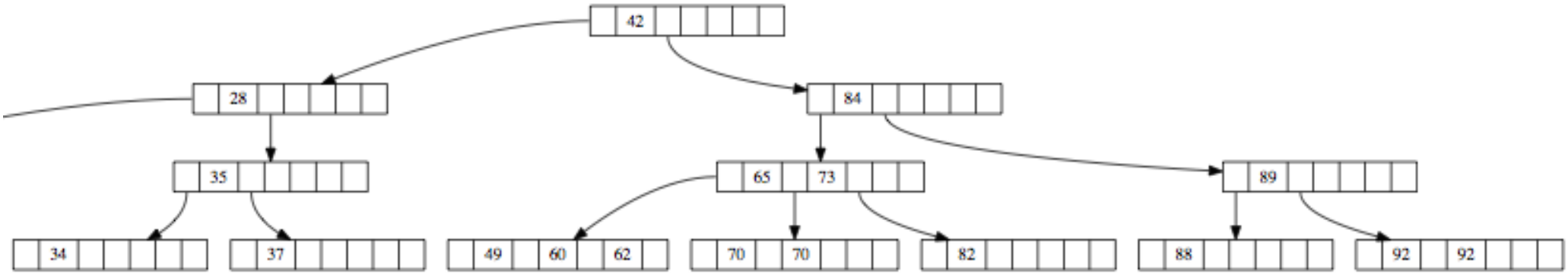
Zeiger i2: _____

Zeiger i3: _____

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Aufgabe 4: 10 Punkte

Aus der Vorlesung kennen Sie die Datenstruktur B-Baum. Gegeben sei nun folgender Ausschnitt eines B-Baumes der Ordnung M = 2:



Zeichnen Sie den resultierenden B-Baum (selbstverständlich ohne den abgeschnittenen Teil) nach dem Löschen des Schlüssels 88.

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Platz für Ihre zeichnerische Lösung:

Aufgabe 5: 15 Punkte

Aus der Vorlesung und vom Java-Übungsblatt kennen Sie lineare Listen und diese Klasse für Listenelemente:

```
public class ListItem <T>  {
     public T            key;
     public ListItem<T>  next;
 }
```

Vom Java-Übungsblatt wissen Sie, was ein *Run* in einer linearen Sequenz ist: eine maximale Teilsequenz von aufeinanderfolgenden Elementen, die aufsteigend sortiert ist. *Maximal* heißt dabei, dass unmittelbar vor und nach dieser Teilsequenz entweder gar kein Schlüsselwert ist oder ein Schlüsselwert, nach dessen Hinzunahme zur Teilsequenz diese nicht mehr aufsteigend sortiert ist (siehe auch das Beispiel unten).

Konkrete Aufgabe: Schreiben Sie eine Methode

```
ListNode<T> reverseRuns ( ListItem<T> head, Comparator<T> cmp )
```

Die Methode darf ohne Überprüfung davon ausgehen, dass `head` auf den Kopf einer korrekt gebildeten, nichtleeren Liste verweist (also dass `head!=null` ist) und dass `cmp` auf ein Objekt einer Klasse verweist, die `Comparator<T>` implementiert. Die Methode soll die Elemente der Liste, auf die `head` verweist, umordnen. Und zwar soll jeder Run in sich unverändert bleiben, aber die Runs sollen ihre Reihenfolge umkehren. Zurückgeliefert wird ein Verweis auf den Kopf der Ergebnisliste.

Beispiel: $(\underbrace{3,5,5,7}, \underbrace{2,4}, \underbrace{3}, \underbrace{2,2,6,8,8}) \longrightarrow (\underbrace{2,2,6,8,8}, \underbrace{3}, \underbrace{2,4}, \underbrace{3,5,5,7})$

Verbindliche Anforderungen:

- Die Methode muss *iterativ* sein, das heißt, Rekursion ist nicht erlaubt.
- Es dürfen keine neuen Objekte mit Operator `new` erzeugt werden, und kein Attribut `key` darf verändert werden, das heißt, die Aufgabe muss durch Änderungen des Attributs `next` in den einzelnen Listenelementen gelöst werden.
- Außer Methode `compare` von `Comparator<T>` darf keine Funktionalität aus der Standardbibliothek von Java oder anderen Bibliotheken verwendet werden.
- Die asymptotische Komplexität muss insgesamt linear bleiben.

Erinnerung: Methode `compare` von `Comparator<T>` liefert $+1$ (bzw. -1) zurück, falls der erste Parameter größer (bzw. kleiner) als der zweite ist, bei Gleichheit 0.

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Von den Korrektoren auszufüllen:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Ihre Lösung schreiben Sie auf diese und die folgenden Seiten bis zur nächsten Aufgabe:

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Aufgabe 6: 15 Punkte

Aus der Vorlesung und vom Java-Übungsblatt kennen Sie Vielwegbäume und diese Klasse für Baumknoten:

```
public class TreeNode <T> {
    public T[]             theKeys;
    public TreeNode<T>[]   theSuccessors;
    public int             numberOfKeys;
}
```

Konkrete Aufgabe: Schreiben Sie eine Methode

```
void mirrorSymmetric ( TreeNode<T> root )
```

Die Methode darf ohne Überprüfung davon ausgehen, dass `root` auf einen korrekt gebildeten Vielwegbaum verweist (der auch leer sein kann, das heißt, `root==null` ist möglich). Insbesondere sind `theKeys` und `theSuccessors` ungleich `null`, und es ist `numberOfKeys`>0. In dem Baum, dessen Wurzel `root` ist, sollen links und rechts spiegelsymmetrisch vertauscht werden. Das heißt: In jedem Knoten `node` dieses Baumes werden die Elemente von `node.theSuccessors` in ihrer Reihenfolge genau umgedreht. An den Schlüsselwerten in einem Knoten soll sich hingegen nichts ändern.

Verbindliche Anforderungen:

- Die Aufgabe soll vollständig durch Rekursion gelöst werden, das heißt, Schleifen sind nicht zulässig (auch nicht zum Durchlauf von `theSuccessors`).
- Es darf keine Funktionalität aus der Standardbibliothek von Java oder anderen Bibliotheken verwendet werden (Attribut `length` von Arrays gehört zur Sprache Java selbst und darf daher verwendet werden).
- Die asymptotische Komplexität muss insgesamt linear bleiben.

Erinnerung: `numberOfKeys` enthält die momentane Anzahl von tatsächlichen Schlüsselwerten im Knoten (die auch kleiner als `theKeys.length` sein kann).

Von den Korrektoren auszufüllen:

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O

Ihre Lösung schreiben Sie auf die folgenden Seiten bis zur nächsten Aufgabe:

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Aufgabe 7: 15 Punkte

Grob gesprochen, komprimiert `foo` zweidimensionale Arrays (die nicht unbedingt Matrizen sein müssen):

```
01      public class E {
02         int r;
03         double x;
04      }

05      public E[][]  foo ( double[][] a )  {
06         E[][] e = bar ( a );
07         e = foobar ( a, e, 0 );
08         return e;
09      }

10      public E[][] bar ( double[][] a ) {
11           E[][] e = new E [ a.length ] [ ];
12           for ( int i = 0; i < a.length; i++ ) {
13              int m = 0;
14              for ( int j = 0; j < a[i].length; j++ )
15                 if ( a[i][j] != 0 )
16                    m++;
17              e[i] = new E [ m ];
18           }
19           return e;
20       }

21      public E[][] foobar ( double[][] a, E[][] e, int i ) {
22         if ( i == a.length )
23            return;
24         foobar ( a, e, i+1 );
25         int k = 0;
26         for ( int j = 0; j < a[i].length; j++ )
27            if ( a[i][j] != 0 ) {
28                e[i][k] = new E();
29                e[i][k].x = a[i][j];
30                e[i][k].r = j;
31                k++;
32            }
33         return e;
34      }
```

Konkrete Aufgaben (a) – (d):

(a) Formulieren Sie **kurz und bündig, aber präzise und unmissverständlich** die Voraussetzungen, die der Parameter `a` von `foo` erfüllen muss, damit alle drei Methoden wohldefiniert sind (sprich: damit keine `RunTimeException` geworfen wird). **2 Punkte**

(b) Formulieren Sie **kurz und bündig, aber präzise und unmissverständlich** den Output der Methoden `foo`, `bar` und `foobar` (bei `foo` also eine Präzisierung der einleitenden Umschreibung vor dem Java-Code). **5 Punkte**

Unverbindlicher Hinweis: Zur Vermeidung von Dopplungen müssen Sie bei `fooBar` Ihre Antwort für `bar` nicht wiederholen, und Ihre Antwort für `foo` dürfen Sie als Spezialfall Ihrer Antwort für `fooBar` formulieren.

(c) Formulieren Sie **kurz und bündig, aber präzise und unmissverständlich** die Invariante der `for`-Schleife in `foobar` sowie den Korrektheitsbeweis (also Induktionsanfang und Induktionsschritt) für diese Invariante. Arbeiten Sie die Induktionsvoraussetzung *explizit* in den Induktionsschritt ein. **5 Punkte**

Verbindliche Hinweise: Beginnen Sie die Invariante analog zum Wiki mit „Nach $h \geq 0$ Iterationen gilt: ...“. Benennen Sie die Indizes `i`, `j`, `h` und `k` überall *explizit*, wo diese von Bedeutung sind.

(d) Sei n die Länge von `a` und m das Maximum aus `a[0].length, a[1].length,...,` `a[`n`−1].length`. Geben Sie die Nummern der Zeilen an, die durch ihre wiederholte Ausführung den höchsten Beitrag zur asymptotischen Gesamtkomplexität leisten. Für beliebige Werte von n und m geben Sie Fälle an, in denen die asymptotische Gesamtkomplexität der Methode `foo` (inklusive aller Aufrufe von `bar` und `foobar`) den Best Case $\Theta(n + m)$ bzw. den Worst Case $\Theta(n \cdot m)$ erreicht. **3 Punkte**

Von den Korrektoren auszufüllen:

(b)		(b)					(c)					(d)		
A	B	A	B	C	D	E	A	B	C	D	E	A	B	C

Ihre Lösung schreiben Sie auf die folgenden Seiten bis zur nächsten Aufgabe:

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Aufgabe 8: 15 Punkte

(a) *Min-Spanning Tree*: Konstruieren Sie ein Gegenbeispiel mit möglichst wenigen Knoten und Kanten für folgende falsche(!) Behauptung:

Man bekommt *immer* einen minimalen spannenden Baum in einem zusammenhängenden ungerichteten Graphen mit Knotenmenge $\{v_1, \ldots, v_n\}$, indem man für beliebiges $i \in \{1, \ldots, n-1\}$ einen minimalen spannenden Baum auf den Knoten $\{v_1, \ldots, v_i\}$ und einen minimalen spannenden Baum auf den Knoten $\{v_{i+1}, \ldots, n\}$ konstruiert und dann von allen Kanten, die einen der Knoten in $\{v_1, \ldots, v_i\}$ mit einem der Knoten in $\{v_{i+1}, \ldots, v_n\}$ verbinden, eine Kante mit geringstem Gewicht hinzufügt.

Begründen Sie, dass Ihr Beispiel tatsächlich ein Gegenbeispiel ist und auch geringstmögliche Knoten- und Kantenzahl hat. **(5 Punkte)**

(b) Warum gilt $\mathcal{P} \subseteq \mathcal{NP}$ und $\mathcal{NPC} \subseteq \mathcal{NP}$? Nennen Sie die Definitionen dieser drei Klassen und argumentieren Sie damit. **(5 Punkte)**

(c) Aus der Vorlesung kennen Sie das algorithmische Konzept *Dynamische Programmierung*. Sei Σ ein beliebiges Alphabet. Die *Distanz* $\Delta(S, T)$ zweier Strings $S = (s_1, s_2, \ldots, s_m)$ und $T = (t_1, t_2, \ldots, t_n)$ mit $s_1, \ldots, s_m, t_1, \ldots, t_n \in \Sigma$ ist die Anzahl der Lösch- und Einfügeoperationen, die mindestens nötig ist, um S in T zu transformieren.

Beispiel: Die Distanz von „algorithmen“ und „datenstrukturen“ ist 16:

„algorithmen“ $\xrightarrow{-6}$ „arten“ $\xrightarrow{+10}$ „datenstrukturen“

Konkrete Aufgabe: Beschreiben Sie **kurz und bündig, aber präzise und unmissverständlich** in Umgangssprache (**nicht** in Programmiersprache) eine Rekursionsgleichung und einen darauf basierenden dynamischen Programmieransatz, der zu zwei gegebenen Strings die Distanz berechnet. **(5 Punkte)**

Hinweis: Seien $S' = (s_1, \ldots, s_{m-1})$ und $T' = (t_1, \ldots, t_{n-1})$. Überlegen Sie sich, wie $\Delta(S, T)$ sich durch $\Delta(S, T')$, $\Delta(S', T)$ und $\Delta(S', T')$ in Abhängigkeit von s_m und t_n ausdrücken lässt. (Vergessen Sie den Rekursionsanker nicht!)

Von den Korrektoren auszufüllen:

(a)					(b)					(c)				
A	B	C	D	E	A	B	C	D	E	A	B	C	D	E

Ihre Lösung schreiben Sie auf die folgenden Seiten:

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____