Klausur Algorithmen und Datenstrukturen

Sommersemester 2016

20. September 2016

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

Hinweis: $i = 2$ heißt: nach Einfügen des neuen Keys und nach **einer** Vertauschung.

Aus der Vorlesung kennen Sie den Algorithmus 'insert' der Datenstruktur Heap. Gegeben sei nun folgender Heap mit N = 15:

Index	1	2	3	4	5	6	7	8	9	10	11	12
ID	10	11	3	9	2	12	7	8	4	1	5	6
Key	**1**	**12**	**17**	**64**	**17**	**60**	**32**	**95**	**86**	**49**	**24**	**88**

Fügen Sie den Schlüssel key = 0 ein. Geben Sie den Heap und das Array Positions nach der Iteration i := 2 an.

Hinweis: Bezeichnen Sie im Array Positions ungenutzte Positionen mit '-1'

TheHeap:

Index	1	2	3	4	5	6	7	8	9	10	11	12	13
ID													
Key													

Positions:

Index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Position															

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. Sehen Sie anhand von generierten Beispielen, unter welchen Umständen die Anwendung von Double hashing zu wesentlich weniger besuchten Positionen im Array führt.

$$F(i, 13, K) := (F_1(13, K) + (i-1) \cdot F_2(13, K)) \; mod \; 13$$
$$F_1(N_{max}, K) := (K \; mod \; N_{max})$$
$$F_2(N_{max}, K) := K + (K \; mod \; N_{max})$$

Fügen Sie die folgenden Werte in der festen Reihenfolge ein:

$$K_1 = 7$$
$$K_2 = 54$$
$$K_3 = 49$$
$$K_4 = 19$$
$$K_5 = 3$$
$$K_6 = 45$$
$$K_7 = 32$$
$$K_8 = 41$$

Index	Key
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

Freie Seite für Ihre Nebenrechnungen (werden nicht gewertet):

Aufgabe 3: 10 Punkte

Aus der Vorlesung kennen Sie den Algorithmus Bucket Sort. Gegeben sei folgende Menge von Strings:

Geben Sie den Zustand des Algorithmus vor und nach der Iteration # 7 an.
Geben Sie ebenfalls die Belegung der Buckets während der Iteration an.
Verwenden Sie als Eingaben bitte nur die IDs der Strings.
Die String-IDs sollen sequentiell und durch Komma getrennt eingetragen werden. Achten Sie ggf. auf die korrekte Reihenfolge.

Strings:

ID	String	Länge
0	x d j y s n g t y a u d	12
1	q r k i g q g h s b e h s y	14
2	r u j r g h h z t n u	11
3	m x o z d q z x v s o k d	13
4	y k q n u z p i s a n p k v g	15
5	y k q n u z p i s	9
6	t r v x	4
7	t r v	3

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Vorher
A

Index	Mengen mit String IDs
0	{ }
1	{ }
2	{ }
3	{ }
4	{ }
5	{ }
6	{ }
7	{ }
8	{ }
9	{ }
10	{ }
11	{ }
12	{ }
13	{ }
14	{ }

S'

Nachher
A

Index	Mengen mitString IDs
0	{ }
1	{ }
2	{ }
3	{ }
4	{ }
5	{ }
6	{ }
7	{ }
8	{ }
9	{ }
10	{ }
11	{ }
12	{ }
13	{ }
14	{ }

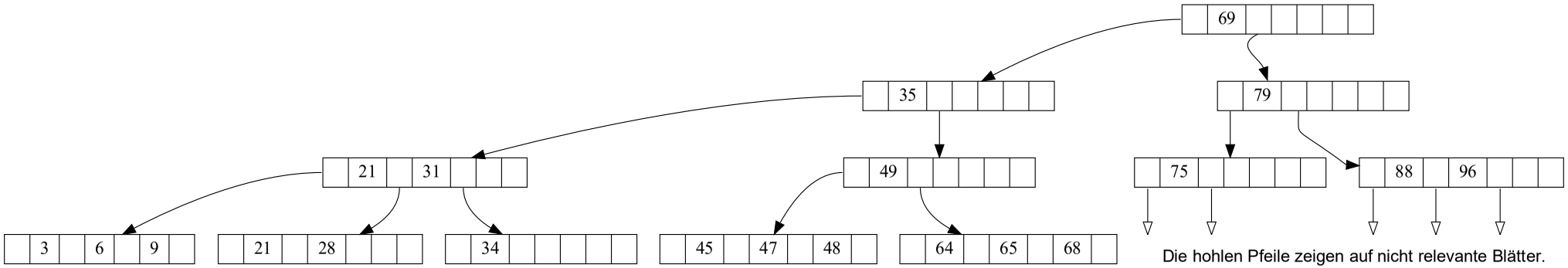
S'

Buckets

Bucket	String IDs
a	
b	
c	
d	
e	
f	
g	
h	
i	
j	
k	
l	
m	
n	
o	
p	
q	
r	
s	
t	
u	
v	
w	
x	
y	
z	

Aufgabe 4: 10 Punkte

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



Die hohlen Pfeile zeigen auf nicht relevante Blätter. Sie müssen diese und Ihre Referenzen in Ihrer Lösung nicht angeben!

Geben Sie den resultierenden B-Baum nach dem Löschen des Schlüssels 34 an.

Platz für Ihre zeichnerische Lösung (weiterer Platz nächste Doppelseite):

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Bitte verwenden Sie das folgende bekannte Eingabeformat, indem Sie eine neue Tabelle anlegen ODER zeichen Sie den resultierenden Baum.

Für Knoten: Knoten-ID, Schlüssel 1, Schlüssel 2, ... , Schlüssel n
Die Knoten-ID ist eine frei wählbare Zahl. Ausnahme: die Wurzel hat immer ID = 0.
Beispiel: 0,3,4,5
für einen Wurzelknoten mit der Schlüsselfolge 3, 4, 5

Für Kanten: Ursprungsknoten-ID, Kind-Zeiger-ID, Zielknoten-ID.
Beispiel: 0,0,1
Für eine Kante von Wurzelknoten, Kind-Zeiger 0, zum Knoten mit ID = 1.

Knoten	Kanten
0,69	0,0,15
15,35	15,0,5
5,21,31	5,0,1
1,3,6,9	5,1,4
4,21,28	5,2,11
11,34	15,1,6
6,49	6,0,3
3,45,47,48	6,1,7
7,64,65,68	0,1,16
16,79	16,0,14
14,75	14,0,12
12,74,75	14,1,10
10,76	16,1,9
9,88,96	9,0,2
2,81,84,86	9,1,8
8,89,96	9,2,13
13,99	

Platz für Ihre zeichnerische Lösung:

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;
 }
```

Konkrete Aufgabe: Schreiben Sie eine Methode

`ListItem<T> merge ( ListItem<T> list, T[] array, Comparator<T> cmp )`

Die Methode darf ohne Überprüfung davon ausgehen, dass `list` auf den Kopf einer korrekt gebildeten, nichtleeren Liste verweist, und dass `array` auf ein Arrayobjekt verweist (insbesondere sind beide ungleich `null`). Weiter können Sie davon ausgehen, dass das Attribut `key` in keinem Knoten `null` ist, dass auch keine Komponente von `array` gleich `null` ist, und dass sowohl die Liste als auch das Array aufsteigend sortiert gemäß `Comparator.compare` sind. Natürlich ist auch `cmp` nicht `null`. Zum Kopieren von Werten des Typs `T` dürfen Sie „=“ verwenden.

Die Komponenten von `array` sollen als weitere Schlüsselwerte in die Liste eingefügt werden, so dass die Ergebnisliste wiederum aufsteigend sortiert ist. Zurückgeliefert wird ein Verweis auf den Kopf der Ergebnisliste. Zu implementieren ist der aus der Vorlesung bekannte Algorithmus `merge`.

Verbindliche Anforderungen:

- Die Methode muss *iterativ* sein, das heißt, Rekursion ist nicht erlaubt.
- Die Anzahl der Aufrufe von `Comparator.compare` muss im Worst Case linear bleiben in der Gesamtzahl an Schlüsselwerten, die am Ende von `merge` in der Liste sind, auf die die Rückgabe verweist.
- Außer `Comparator.compare` darf keine Funktionalität aus der Standardbibliothek von Java oder anderen Bibliotheken verwendet werden.

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

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 6: 15 Punkte

Aus der Vorlesung und vom Java-Übungsblatt kennen Sie Vielwegsuchbäume 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

```
ListItem<T>  traverse ( TreeNode<T> root )
```

auf Basis der Klasse `ListItem<T>` von Aufgabe 5. Die Methode darf ohne Überprüfung davon ausgehen, dass `root` auf einen korrekt gebildeten Vielwegbaum verweist (der auch leer sein kann), und dass dieser Baum die Suchbaumeigenschaft erfüllt. Insbesondere sind `theKeys` und `theSuccessors` ungleich `null`. Die Methode `traverse` soll eine aufsteigend sortierte Liste aller Schlüsselwerte in diesem Baum zurückliefern. Zum Kopieren von Werten des Typs `T` dürfen Sie „=“ verwenden.

Verbindliche Anforderungen:

- Die Aufgabe soll vollständig durch Rekursion gelöst werden, das heißt, Schleifen sind nicht zulässig (auch nicht zum Durchlauf von `theKeys` und `theSuccessors`).
- Die asymptotische Komplexität soll linear in der Gesamtzahl aller Schlüsselwerte im Baum sein.
- Es darf keine Funktionalität aus der Standardbibliothek von Java oder anderen Bibliotheken verwendet werden.

Unverbindliche Hinweise:

- Es bietet aich an, die beiden Rekursionen in zwei Hilfsmethoden auslagern.
- Richten Sie als erstes ein Listenelement („Dummyelement“) ein, an das Sie alle „wirklichen“ Listenelemente hängen, das `traverse` aber nicht mit zurückliefert. Die rekursiven Methoden haben einen Verweis auf das momentan *letzte* Listenelement als (einen ihrer) Parameter und liefern einen Verweis auf das Element zurück, das nach dem Anhängen neuer Elemente das nunmehr letzte ist.

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

Intuitiv gesprochen, berechnet `foo`, wie weit rechts oder links von der Hauptdiagonale (also von den Indizes `[0][0]`, `[1][1]`, `[2][2]` ...) entfernt noch Einträge ungleich 0 vorkommen:

```
01        public int foo ( long[][] a )  {
02           return bar ( a, 0 );
03        }

04        public int bar ( long[][] a, int i ) {
05             if ( i == a.length )
06                return -1;
07             int max1 = fooBar ( a[i], i );
08             int max2 = bar ( a, i+1 );
09             if ( max1 > max2 )
20                return max1;
11             return max2;
12        }

13        public int fooBar ( long[] b, int i ) {
14             int k = -1;
15             for ( int j = 0; j < b.length; j++ ) {
16                 if ( b[j] != 0 && k < i - j )
17                     k = i - j;
18                 if ( b[j] != 0 && k < j - i )
19                     k = j - i;
20             }
21             return k;
22         }
```

Beachten Sie dabei, dass die einzelnen Arrays `a[`i`]` nicht unbedingt gleiche Länge haben müssen. Sie können aber davon ausgehen, dass `a` selbst und auch jedes einzelne Array `a[`i`]` ungleich `null` ist.

Konkrete Aufgaben (a) – (d):

(a) Formulieren Sie **kurz und bündig, aber präzise und unmissverständlich** den Output jeder der Methoden `foo`, `bar` und `fooBar` (bei `foo` also eine Präzisierung der obigen intuitiven Umschreibung). **3 Punkte**

Hinweise: Nennen Sie den Rückgabewert jeweils r und formulieren Sie so etwas wie „die maximale ganze Zahl r, so dass ...“. Ihre Formulierung ist erst dann präzise,

wenn sie die Terme `a.length` und `a[`i`].length` bzw. `b.length` geeignet enthält. Für `bar` ist ein weiterer Index h zweckmäßig, der die Indizes von `a` durchläuft, da i hier schon eine andere Rolle hat.

(b) Formulieren Sie **kurz und bündig, aber präzise und unmissverständlich** die Invariante der Schleife in `fooBar` sowie den Korrektheitsbeweis (also Induktionsanfang und Induktionsschritt) für diese Invariante. Arbeiten Sie die Induktionsvoraussetzung *explizit* in den Induktionsschritt ein. **4 Punkte**

Hinweise: Beginnen Sie die Invariante analog zum Wiki mit „Nach $j \geq 0$ Iterationen gilt:“. Nehmen Sie wieder wie in Teil (a) einen zusätzlichen Index h zu Hilfe. Benennen Sie die Indizes i, j und h überall *explizit*, wo diese von Bedeutung sind.

(c) Formulieren Sie **kurz und bündig, aber präzise und unmissverständlich** die Invariante der Rekursion in `bar` sowie den Korrektheitsbeweis (also Induktionsanfang und Induktionsschritt) für diese Invariante. Arbeiten Sie wieder die Induktionsvoraussetzung *explizit* ein. **4 Punkte**

Hinweise: Für Invariante und Induktionsanfang machen Sie sich klar, dass der Wert `-1` bei nichtnegativen Zahlen als neutrales Element der Maximumsbildung verwendet werden kann. Beim Induktionsschritt leiten Sie die Werte von `max1` und `max2` her und kombinieren diese beiden Teilergebnisse zu einer Gesamtaussage.

(d) Sei n die Länge von `a` und m das Maximum aus `a[0].length, a[1].length,...,` `a[`$n-1$`].length`. Geben Sie an, unter welchen Umständen der Best Case bzw. der Worst Case bei festgehaltenen Werten n und m eintritt, und begründen Sie, warum die asymptotische Gesamtkomplexität der Methode `foo` (inklusive aller Aufrufe von `bar` und `fooBar`) im Best Case in $\Theta(n+m)$ und im Worst Case in $\Theta(n \cdot m)$ ist. Geben Sie die Nummern der Zeilen an, die durch ihre wiederholte Ausführung den höchsten Beitrag zur asymptotischen Gesamtkomplexität leisten. **4 Punkte**

Von den Korrektoren auszufüllen:

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

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) Warum ist die Anzahl paarweiser Vergleiche bei Quicksort in $\Theta(n^2)$ im Worst Case? Identifizieren Sie dazu, unter welchen Umständen der Worst Case eintritt. Begründen Sie, warum das der Worst Case ist, und begründen Sie, warum die Anzahl Vergleiche für diesen Fall in $\Theta(n^2)$ ist. **(5 Punkte)**

(b) Überlegen Sie sich eine möglichst kleines Beispieleingabe für den Algorithmus von Dijkstra, die zeigt, dass dieser Algorithmus nicht unbedingt korrekte Ergebnisse liefert, wenn negative Kantenlängen in der Eingabe vorkommen. Erläutern Sie, wie der Algorithmus auf Ihrem Beispiel vorgehen würde, welcher suboptimale Pfad dabei herauskommt und welcher Pfad kürzer wäre. **(5 Punkte)**

(c) Was bedeutet es konkret zu sagen, dass das TSP in $\mathcal{NP}$ ist, und warum ist diese konkrete Aussage korrekt? Verwenden Sie die Begriffe *Entscheidungsproblem*, *Zertifikat* und *Ja-Instanz* geeignet (diese Begriffe müssen Sie nicht erläutern). **(5 Punkte)**

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 diese und die folgenden Seiten:

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____

Ihre Matrikelnummer (zu Ihrer Sicherheit):_____