Klausur
Theoretische Informatik II
Algorithmen und Komplexität
TINF18IT2 – Sommersemester 2019
13. Juni 2019, 12:30 Uhr

DHBW
Duale Hochschule
Baden-Württemberg

Lösungsvorschlag

Aufgabe	**1**	**2**	**3**	**4**	**5**	**6**	**7**	$\sum_K$	Note_K
Punkte									

Ü-Blatt	**1**	**2**	**3**	**4**	**5**	$\sum_Ü$	$\text{Note}_Ü$
Punkte							

Gesamtnote: $0.3 \cdot \text{Note}_Ü + 0.7 \cdot \text{Note}_K =$ ☐

Bitte beachten Sie

- Schreiben Sie auf jedes Blatt Ihre Matrikelnummer, auch auf etwaige zusätzlich verwendete Blätter.
- Wenn Sie zusätzliches Papier verwenden, schreiben Sie auch die Aufgabennummer auf jedes Blatt.
- Die Klausur hat 16 Seiten mit 7 Aufgaben.
- Überprüfen Sie, dass der ausgehändigte Klausurbogen 7 verschiedene Aufgaben enthält!
- Die Bearbeitungszeit beträgt 120 Minuten.
- Es sind keine Hilfsmittel zugelassen.
- Geben Sie für jede Aufgabe nur eine Lösung an. Bei mehreren angegebenen Lösungen wird die jeweils schlechteste zur Wertung herangezogen.
- Die maximal für eine Aufgabe erreichbaren Punkte stehen hinter der Aufgabennummer. Bei Aufgaben mit mehreren Teilaufgaben werden die Punkte als Summe der maximal in den Teilaufgaben erreichbaren Punkte dargestellt.
- Zum Bestehen der Prüfung muss eine Gesamtnote (aus Übungsnote und Klausurnote zusammengesetzt) von 4.0 erreicht werden.
- Zum Erreichen der Note 4.0 in der Klausur sind 30 Punkte hinreichend.
- Für die Indizierung von Arrays mit n Elementen wird die C-Konvention verwendet, dass der erste Eintrag im Array den Index 0 hat, und der letzte Eintrag den Index $n - 1$.

Viel Erfolg!

Aufgabe 1 (10 Punkte)

Kreuzen Sie bei den folgenden Fragen die korrekten Antworten an.

Jede korrekt angekreuzte Antwort gibt dabei einen Punkt, jede falsch angekreuzte Antwort führt zu einem Punktabzug. Wird keine Antwort angekreuzt, so gibt es 0 Punkte für die Frage. Insgesamt kann es für die ganze Aufgabe nicht weniger als 0 Punkte geben.

		Wahr	Falsch
1.	Alan Turing hat mathematisch bewiesen, dass ein **NP**-vollständiges Problem nicht durch einen Algorithmus in polynomialer Zeit gelöst werden kann.		X
2.	Ein vergleichsbasierter Sortieralgorithmus kann keinen besseren asymptotischen Aufwand als $O(n \log(n))$ haben.	X	
3.	Angenommen, jede Folge F von m Operationen auf einer Datenstruktur D habe den Gesamtaufwand $T(F) \leq 4m + 2$. Dann haben die einzelnen Operationen auf D konstanten amortisierten Aufwand.	X	
4.	Im ungünstigsten Fall (worst case) ist der Aufwand von MERGESORT in $\Theta(n^2)$.		X
5.	Ein großer Vorteil von Hash-Tabellen ist ihre effiziente Speichernutzung.		X
6.	Es sei T ein minimaler Spannbaum eines Graphen G. Wenn jedes Kantengewicht von G mit einer Konstanten $c > 0$ multipliziert wird, so ist T weiterhin ein minimaler Spannbaum.	X	
7.	Ein Baum mit n Knoten hat genau $n - 1$ Kanten.	X	
8.	Bei der Aufwandsanalyse von Kruskals Algorithmus ist der Aufwand für die Vorsortierung der Kanten vernachlässigbar.		X
9.	Das Problem des Handelsreisenden (TSP) ist **NP**-schwer, aber nicht **NP**-vollständig.		X
10.	Jeder Wald mit n Knoten, der aus k Teilbäumen besteht, hat genau $n - k$ Kanten.	X	

Aufgabe 2 (10 = 2 + 2 + 2 + 2 + 2 Punkte)

Wir betrachten Funktionen $f, g : \mathbb{N} \to \mathbb{R}_+$, die für alle Eingaben $n \in \mathbb{N}$ positiv sind.

(a) Ergänzen Sie in den Klammern die korrekte Definition von $\Omega(f(n))$:

$$\Omega(f(n)) = \left\{ g(n) \;\middle|\; \boxed{\text{es gibt } c > 0 \text{ und } n_0 \in \mathbb{N}, \text{ so dass für alle } n \geq n_0 \text{ gilt: } g(n) \geq c f(n)} \right\}.$$

(b) Vervollständigen Sie: Aus der Vorlesung ist bekannt, dass für ein Polynom $p(n)$ vom Grad k gilt:

$$p(n) \in \Theta(\boxed{n^k}).$$

(c) Zeigen Sie oder widerlegen Sie:

$$4n^3 - 5n + 3 \in \Omega(n^2).$$

Antwort:
Schreibe $p(n) = 4n^3 - 5n + 3$. Nach Teil (b) gilt $p(n) \in \Theta(n^3)$. Da $\Theta(n^3) = O(n^3) \cap \Omega(n^3)$, gilt insbesondere

$$p(n) \in \Omega(n^3) \subset \Omega(n^2).$$

Ausführlicher (nicht verlangt): Da $p(n) \in \Omega(n^3)$, gibt es eine Konstante $c > 0$ und einen Index $n_0 \in \mathbb{N}$, so dass für $n \geq n_0$ gilt:

$$p(n) \geq cn^3 > cn^2,$$

wobei die zweite Ungleichung natürlich daraus folgt, dass $n^3 > n^2$ gilt für $n > 0$. Dies bedeutet aber nichts anderes als $p(n) \in \Omega(n^2)$.

(d) Zeigen Sie oder widerlegen Sie:

$$f(n) + g(n) \in \mathrm{O}(f(n)), \text{ falls } g(n) \in \mathrm{O}(f(n)).$$

Antwort:
Da $g(n) \in \mathrm{O}(f(n))$, gibt es eine Konstante $c > 0$ und ein $n_0 \in \mathbb{N}$, so dass für $n \geq n_0$ gilt:

$$f(n) + g(n) \leq f(n) + cf(n) = (1 + c)f(n).$$

Dies bedeutet nichts anderes, als dass $f(n) + g(n) \in \mathrm{O}(f(n))$ liegt (verwende das selbe n_0 und als Konstante $c' = 1 + c$).

(e) Zeigen Sie oder widerlegen Sie:

$$n + \sin(n) \in \mathrm{O}(n).$$

Antwort:
Für alle $n \geq 1$ gilt $\sin(n) \leq 1$ und somit

$$n + \sin(n) \leq n + 1 \leq n + n = 2n.$$

Dies bedeutet $n + \sin(n) \in \mathrm{O}(n)$.

Aufgabe 3 (10 = 1 + 2 + 3 + 1 + 1 + 2 Punkte)

(a) Geben Sie die Gaußsche Summenformel an (für $n \in \mathbb{N}$):

$$1 + 2 + 3 + \ldots + (n-1) + n = \boxed{\frac{n(n+1)}{2}}$$

(b) Betrachten Sie das folgende Programmfragment:

```
EINEFUNKTION(int n)
    for i = 1 to n do
        for j = i to n do
            MACHWAS( )
        end_for
    end_for
```

Wie oft wird, abhängig von der Eingabe n, innerhalb der Prozedur EINEFUNKTION die Prozedur MACHWAS aufgerufen? Begründen Sie Ihre Antwort.

Antwort:

MACHWAS wird (für festes i) in der inneren Schleife $n - i$-mal aufgerufen, und i selbst läuft von 0 bis n. Also:

$$\begin{aligned} n &\text{ Aufrufe in der ersten inneren Schleife, } i = 1 \\ n-1 &\text{ Aufrufe in der zweiten inneren Schleife, } i = 2 \\ &\vdots \\ 2 &\text{ Aufrufe in der vorletzten inneren Schleife, } i = n-1 \\ 1 &\text{ Aufruf in der letzten inneren Schleife, } i = n \end{aligned}$$

Somit haben wir insgesamt $n + (n-1) + \ldots + 2 + 1 = \frac{n(n+1)}{2}$ Aufrufe (verwende Teil (a)).

(c) Vervollständigen Sie die Formulierung des Master-Theorems:

Für vier Konstanten $a, b, c, d > 0$ und $n = b^k$ für ein $k \in \mathbb{N}$ sei die folgende Rekurrenz gegeben:

$$T(n) = \begin{cases} a & \text{falls } n = 1, \\ c \cdot n + d \cdot T(\frac{n}{b}) & \text{falls } n > 1 \end{cases}$$

Dann gilt:

$$T(n) \in \begin{cases} \boxed{\Theta(n)} & \text{falls } d < b, \\ \boxed{\Theta(n \log(n))} & \text{falls } d = b, \\ \boxed{\Theta(n^{\log_b(d)})} & \text{falls } d > b. \end{cases}$$

(d) Beweisen oder widerlegen Sie: Gilt

$$T(n) = \begin{cases} 1, & \text{falls } n = 1, \\ 25 \cdot T(\frac{n}{5}) + n, & \text{falls } n > 1, \end{cases}$$

so folgt daraus

$$T(n) \in \mathrm{O}(n^2).$$

(Sie dürfen dabei annehmen, dass n eine Potenz von 5 ist.)

Antwort:
Dies ist der Fall $d = 25$ und $b = 5$ im Master-Theorem. Also $d > b$, und somit

$$T(n) \in \mathrm{O}(n^{\log_5(25)}) = \mathrm{O}(n^2).$$

(e) Geben Sie die Definition der Komplexitätsklasse **NP** an.

Anwort:
NP besteht aus allen (entscheidbaren) Prädikaten, die von einer nicht-deterministischen Turing-Maschine in polynomialer Zeit entschieden werden können.
(Äquivalent:
Ein Problem $\mathcal{Q}$ liegt in **NP**, wenn für eine gegebene Probleminstanz q von $\mathcal{Q}$ und einen Lösungsvorschlag x dafür von einer deterministischen Turing-Maschine in polynomialer Zeit geprüft werden kann, ob x eine Lösung von q ist.)

(f) Geben Sie die Definition von **NP**-*Vollständigkeit* eines Problems $\mathcal{P}$ an. (Falls Sie dazu den Begriff der **NP**-Schwere verwenden, geben Sie dessen Definition ebenfalls an.)

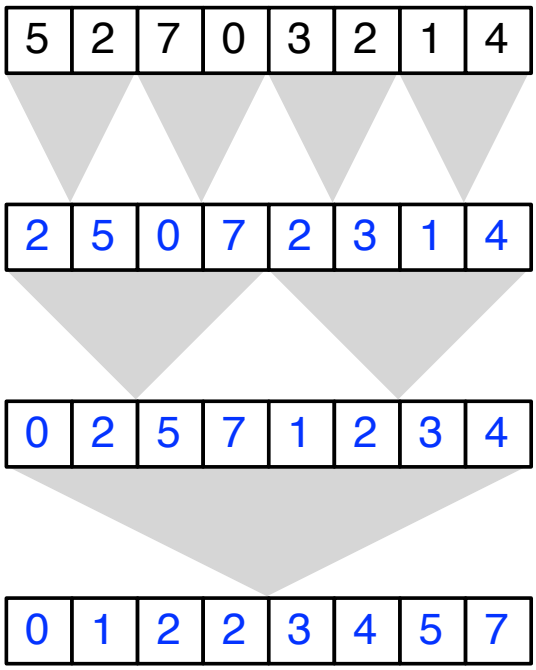
Anwort:
Ein Problem $\mathcal{P}$ ist **NP**-vollständig, wenn

(a) jedes Problem aus **NP** in polynomialer Zeit auf $\mathcal{P}$ reduziert werden kann (d.h. $\mathcal{P}$ ist **NP**-schwer),

(b) und $\mathcal{P}$ selbst in **NP** liegt.

Aufgabe 4 (10 = 3 + 1 + 4 + 1 + 1 Punkte)

(a) Sortieren Sie den folgenden Array mit MERGESORT. Tragen Sie dazu in den Kästchen unten für jeden Schritt die aktuelle Anordnung der Zahlen ein.



(b) Sie wollen ein neues Element in einen bereits sortierten Array einsortieren. Welchen Sortieralgorithmus schlagen Sie dazu vor? Begründen Sie Ihre Antwort.

Antwort:
Von den Algorithmen aus der Vorlesung ist INSERTIONSORT am besten geeignet, da der Fall eines vorsortierten Arrays für INSERTIONSORT der günstigste Fall ist, und dann der Aufwand in $O(n)$ liegt.
(Die anderen Algorithmen aus der Vorlesung haben auch im vorsortierten Fall den Aufwand $O(n \log(n))$ oder schlechter.)

(c) Der folgende Sortieralgorithmus erhält als Eingabe einen Array A der Länge n.

```
1:  FIZZYSORT(A)
2:      n := LENGTH(A)
3:      for i = 0 to n − 1 do
4:          for j = n − 1 downto i + 1
5:              if A[j] < A[j − 1] then SWAP(A[j], A[j − 1]) end_if
6:          end_for
7:      end_for
```

Die Prozedur SWAP vertauscht hierbei die Werte ihrer beiden Argumente.

Ergänzen Sie: Der asymptotische Aufwand (in Anzahl der Vergleiche von Arrayeinträgen) von FIZZYSORT im *ungünstigsten Fall* ist

$$\Theta(\boxed{n^2}).$$

Geben Sie eine kurze Begründung für Ihre Antwort.

Antwort:
Ein ähnliches Zählargument wie in Aufgabe 3 (b):

$n-1$ Vergleiche in der ersten inneren Schleife, $i = 0$
$n-2$ Vergleiche in der zweiten inneren Schleife, $i = 1$
$\vdots$
1 Vergleich in der vorletzten inneren Schleife, $i = n-2$
0 Vergleiche in der letzten inneren Schleife, $i = n-1$

Somit haben wir insgesamt $(n-1)+(n-2)\ldots+2+1 = \frac{(n-1)n}{2}$ Vergleiche (verwende erneut Aufgabe 3 (a)). Es gilt aber $\frac{(n-1)n}{2} \in \Theta(n^2)$ (z.B. nach Aufgabe 2 (b)).

(d) Wie unterscheidet sich der asymptotische Aufwand von FIZZYSORT im *günstigsten* Fall vom *ungünstigsten* Fall?

Antwort:
Es gibt keinen Unterschied. Die Anzahl der Schleifendurchläufe, und somit der Vergleiche, ist vom Algorithmus fest vorgegeben und hängt nicht von der Eingabe A ab.

(e) Ist FIZZYSORT in-place? Begründen Sie kurz Ihre Antwort.

Antwort:
Ja. Alle Datenbewegungen werden direkt innerhalb des Arrays A ausgeführt. Die Hilfsfunktion SWAP benötigt höchstens $O(1)$ temporären Speicher zum Vertauschen zweier Werte.

Aufgabe 5 (10 Punkte)

Wir betrachten erneut den Sortieralgorithmus FIZZYSORT.

```
1:    FIZZYSORT(A)
2:        n := LENGTH(A)
3:        for i = 0 to n − 1 do
4:            for j = n − 1 downto i + 1
5:                if A[j] < A[j − 1] then SWAP(A[j], A[j − 1]) end_if
6:            end_for
7:        end_for
```

Zeigen Sie, dass nach Beendigung von FIZZYSORT der Array sortiert ist (d.h. es gilt $A[0] \leq A[1] \leq \ldots \leq A[n-1]$), indem Sie das folgende Beweisfragment an den markierten Stellen ergänzen:

- Ohne Beweis verwenden wir: Es gilt die folgende Schleifeninvariante **(J)** für die innere **for**-Schleife (Zeilen 4-6):

  Am Ende der Iteration für Index j *ist* $A[j-1]$ *kleiner-gleich den Elementen* $A[j]$ *bis* $A[n-1]$.

- Wir zeigen nun, dass die folgende Schleifeninvariante **(I)** für die äußere **for**-Schleife (Zeilen 3-7) gilt:

  Am Ende der i*-ten Iteration sind die Elemente* $A[\boxed{0}]$ *bis* $A[\boxed{i}]$

  (i) *sortiert und*

  (ii) *kleiner-gleich den Elementen* $A[\boxed{i+1}]$ *bis* $A[\boxed{n-1}]$.

- Der Beweis ist durch Induktion über i.

  Induktionsanfang $i = 0$:
  Für Bedingung (i) der Invariante (I) ist hier nichts zu zeigen.

  Die innere **for**-Schleife terminiert beim Index $j = \boxed{1}$. Wegen der Schleifeninvariante (J) für die innere **for**-Schleife gilt somit, dass $A[\boxed{0}]$ kleiner-gleich den Elementen $A[\boxed{1}]$ bis $A[\boxed{n-1}]$ ist, was Bedingung (ii) entspricht.

  Induktionsvoraussetzung:
  Wir nehmen an, dass nach der i-ten Iteration der äußeren Schleife die Schleifeninvariante (I) gilt (für ein gewisses $i \geq 1$).

  Induktionsschritt $i \rightsquigarrow i+1$:
  Wir stellen fest, dass in der $i+1$-ten Iteration die innere **for**-Schleife beim

Wert $j = \boxed{i+2}$ terminiert. Aufgrund der Schleifeninvariante (J) gilt somit *am Ende* der $i+1$-ten Iteration der äußeren **for**-Schleife:

$$A[i+1] \leq A[\,\boxed{i+2}\,], \ldots, A[\,\boxed{n-1}\,].$$

Wegen der Induktionsvoraussetzung folgt damit zunächst

$$A[\,\boxed{0}\,] \leq \ldots \leq A[\,\boxed{i}\,] \leq A[i+1],$$

was Teil (i) der Schleifeninvariante (I) für $i+1$ ist, und außerdem

$$A[\,\boxed{k}\,] \leq A[i+2], \ldots, A[n-1] \quad \text{für } k = 0, \ldots, i+1,$$

also Teil (ii) der Invariante (I) für $i+1$.

Die Schleifeninvariante (I) gilt somit für $i+1$ und daher nach dem Induktionsprinzip für alle $i \geq 1$.

- Die Korrektheit von FIZZYSORT folgt nun aus der Gültigkeit der Schleifeninvariante (I) der äußeren Schleife für $i = \boxed{n-1}$. ◇

Aufgabe 6 (10 = 2 + 4 + 2 + 2 Punkte)

(a) Es sei H eine Hash-Tabelle. Was wird in H gespeichert...

(i) ... beim verketteten Hashing?

Antwort:
Die Daten werde nicht in der Hash-Tabelle selbst, sondern in verketteten Listen gespeichert („Kollisionslisten"). Die Hash-Tabelle enthält die Zeiger auf die Kopfelemente dieser verketteten Listen.

(ii) ... beim Hashing mit linearer Sondierung?

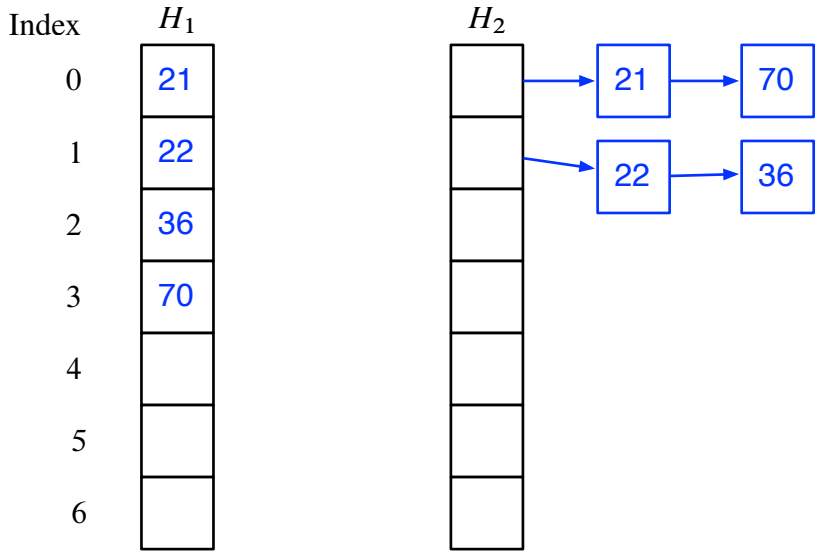
Antwort:
Die Hash-Tabelle enthält selbst die Daten, die gespeichert werden sollen.

(b) Wir betrachten zwei Hash-Tabellen H_1 und H_2 mit der Hash-Funktion

$$h(x) = x \bmod 7.$$

Kollisionen werden dabei von H_1 über *lineare Sondierung* aufgelöst, und von H_2 über *verkettetes Hashing* (wobei neue Einträge am Ende der jeweiligen Liste angehängt werden sollen).

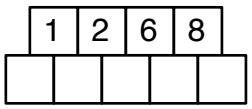
Zeigen Sie in der Skizze unten, wo die Elemente 21, 22, 36, 70 (in dieser Reihenfolge) jeweils in den beiden Hash-Tabellen eingefügt werden.



(c) Nennen Sie einen Vor- und einen Nachteil von doppelt verketteten Listen gegenüber einfach verketteten Listen.

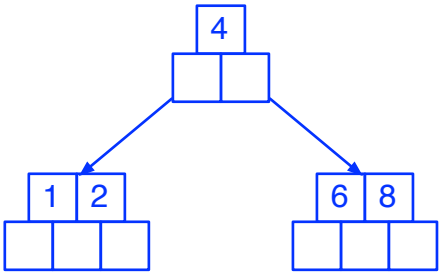
Antwort:
Vorteil: Zugriff auf Vorgängerelemente ist möglich, und dadurch das direkte Löschen eines Elements (ohne das Vorgängerelemente gegeben bekommen zu müssen).
Nachteil: Höherer Speicheraufwand, höherer Zeitaufwand bei Ändern der Liste, da mehr Zeiger umgehängt werden müssen.

(d) Gegeben sei ein B-Baum mit Parametern $(a, b) = (2, 5)$, der als einzigen Knoten die Wurzel enthält, wie unten dargestellt.



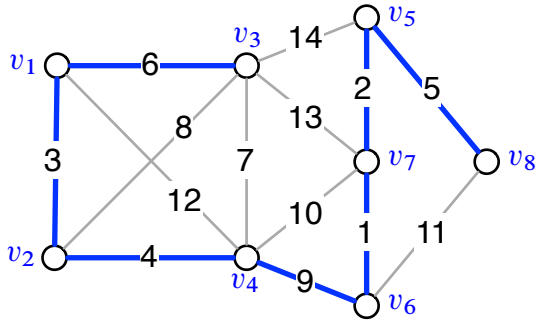
Nun soll ein Element mit Schlüssel $\boxed{4}$ eingefügt werden. Zeichnen Sie den resultierenden B-Baum.

Antwort:



Aufgabe 7 (10 = 4 + 6 Punkte)

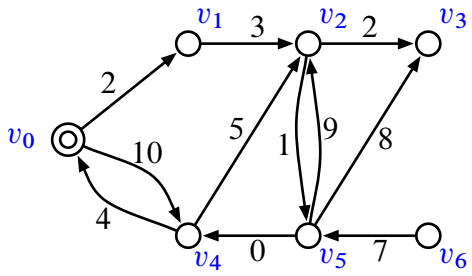
(a) Bestimmen Sie mit Kruskals Algorithmus einen minimalen Spannbaum für den folgenden Graphen (die Zahlen an den Kanten sind die Kantengewichte). Geben Sie dabei die Reihenfolge an, in der die Kanten zum minimalen Spannbaum hinzugefügt werden.



Antwort:

(v_6, v_7)
(v_5, v_7)
(v_1, v_2)
(v_2, v_4)
(v_5, v_8)
(v_1, v_3)
(v_4, v_6)

(b) Bestimmen Sie mit Dijkstras Algorithmus die Längen der kürzesten Pfade vom Startknoten v_0 zu den anderen Knoten. Geben Sie dabei die Reihenfolge an, in der die Knoten des Graphen von Dijkstras Algorithmus erkundet werden.



Knoten	Distanz zu v_0	Reihenfolge
v_0	0	0
v_1	2	1
v_2	5	2
v_3	7	5
v_4	~~10~~, 6	4
v_5	6	3
v_6	∞	–