Algorithmen und Datenstrukturen 1

SoSe 2020 · Prof. Dr. U. Meyer
27. Juli 2020
Klausurnummer: **0001**

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Nachname (Druckschrift)**:** _____

Vorname (Druckschrift)**:** _____

Matrikelnummer: _____

Studiengang: _____

Bitte Hinweise beachten:

- Schreiben Sie Ihren **Namen nur auf dieses Titelblatt**. Sollten Sie Zusatzpapier bekommen, vermerken Sie darauf unbedingt die Klausurnummer **0001**.
- Diese Klausur ist für die Variante **Algo1 (vollständig)**.
- Die Klausur besteht aus den Aufgaben **1 − 8**.
- Merken oder notieren Sie sich Ihre Klausurnummer **0001**, da nur unter dieser Nummer die Ergebnisse veröffentlicht werden.
- Es dürfen nur **dokumentenechte Stifte** in den Farben blau und schwarz verwendet werden. Insb. ist die Nutzung von Tintenlöschern untersagt. Zugelassene Hilfsmittel: 1 Blatt DIN A4 mit handschriftlichen Notizen (zweiseitig).
- Das Mitbringen nicht zugelassener Hilfsmittel stellt einen Täuschungsversuch dar und führt zum Nichtbestehen der Klausur. **Schalten Sie bitte deshalb alle elektronischen Geräte, insbesondere Handys und Smartwatches, vor Beginn der Klausur aus und packen Sie diese weg.**
- Werden zu einer Aufgabe zwei oder mehr Lösungen angegeben, so gilt die Aufgabe als nicht gelöst. Entscheiden Sie sich also immer für **eine** Lösung. Begründungen sind nur dann notwendig, wenn die Aufgabenformulierung dies verlangt.
- Die Klausur ist mit Sicherheit bestanden, wenn (ohne Bonifikation aus den Übungspunkten) mindestens **50%** der Höchstpunktzahl erreicht wird.
- Die Klausur dauert **180 Minuten**.

2 0011000 100018

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Diese Seite ist für den internen Gebrauch.
Bitte leer lassen.

Aufgabe	1	2	3	4	5	6	7	8
Erreichbar	22	20	20	24	20	12	24	18
Erreicht								

Klausur	Bonfikation

2 0 1 1 0 0 0 1 0 0 0 1 8

Aufgabe 1: O-Notation und Codeanalyse — $8 + 8 + 6 = 22$ Punkt(e)

a) Geben Sie jeweils eine Funktion f an, welche die angegebenen Wachstumseinschränkungen einhält.

$f \in \omega(\sqrt{n})$ $\quad$ $f \in o(n)$ $\quad$ $f(n) =$ _____

$f \in \omega(n)$ $\quad$ $f \in o(n \log n)$ $\quad$ $f(n) =$ _____

$4^f \in \Theta(2^f)$ $\quad$ $f(n) =$ _____

$f \in \Theta(n)$ $\quad$ $f \in o(2^n)$ $\quad$ $f(n) =$ _____

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

b) Lösen Sie die folgenden Rekursionsgleichungen auf. Für alle Rekursionsgleichungen gilt $T(n) = 1$ für alle $n \leq 1$. Nehmen Sie vereinfachend an, dass n so gewählt wurde, so dass sämtliche Divisionen restfrei aufgehen.

$T(n) = 8 \cdot T(\frac{n}{2}) + 3$ $\quad$ $T(n) = \Theta($_____$)$

$T(n) = 2 \cdot T(\frac{n}{4}) + 3$ $\quad$ $T(n) = \Theta($_____$)$

$T(n) = 1 \cdot T(\frac{n}{2}) + 3$ $\quad$ $T(n) = \Theta($_____$)$

$T(n) = 9 \cdot T(\frac{n}{3}) + n^2$ $\quad$ $T(n) = \Theta($_____$)$

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

c) Geben Sie für die folgenden Algorithmen jeweils die Laufzeit in Θ-Notation an.

```
Algorithmus loop1(n)
  for i = 1 to n
    j = i
    while j > 1
      j = ⌊j/2⌋
```

```
Algorithmus loop2(n)
  s = 2
  while s ≤ n
    s = s * s
```

```
Algorithmus loop3(n)
  i = n
  while i ≥ 1
    j = i
    while j ≤ n
      j = 2 * j
    i = i − 1
```

loop1: $\Theta($_____$)$

loop2: $\Theta($_____$)$

loop3: $\Theta($_____$)$

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

Aufgabe 2: Hashing — $10 + 10 = 20$ Punkt(e)

a) Gegeben ist folgende Hashtabelle der Größe 13. Es wird doppeltes Hashing mit den folgenden Funktionen verwendet:

$$f_i(k) = (h_1(k) + i \cdot h_2(k)) \bmod 13$$
$$h_1(k) = 6k \bmod 13$$
$$h_2(k) = 5 - (k \bmod 5)$$

0	1	2	3	4	5	6	7	8	9	10	11	12
	24	35		18		14			21			28

Fügen Sie die folgenden Elemente in der gegebenen Reihenfolge in die Hashtabelle ein.

$$1, 2, 3, 4, 5$$

Rechenhilfe:

k	1	2	3	4	5
$h_1(k)$	6	12	5	11	4

⇑⇑⇑ _____ / 10 Punkt(e) ⇑⇑⇑

b) Gegeben ist folgende Hashtabelle der Größe 13. Es wird Hashing mit linearem Austesten mit den Hashfunktionen $h_i(k) = (4k + i) \bmod 13$ verwendet.

0	1	2	3	4	5	6	7	8	9	10	11	12
			30	14			44	28	25		32	29

Fügen Sie die folgenden Elemente in der gegebenen Reihenfolge in die Hashtabelle ein.

$$1, 2, 3, 4, 5$$

Rechenhilfe:

k	1	2	3	4	5
$h_0(k)$	4	8	12	3	7

⇑⇑⇑ _____ / 10 Punkt(e) ⇑⇑⇑

Aufgabe 3: Erdős-Zahl — $8 + 8 + 4 = 20$ Punkt(e)

Paul Erdős war ein ungarischer Mathematiker, der etwa 1 500 Artikel gemeinsam mit über 500 Koautoren[1] publizierte. Ihm zu Ehren wurde die Erdős-Zahl $z(\cdot)$ definiert. Sei $A = \{a_1, \ldots, a_n\}$ die Menge der Autoren. Paul Erdős wird durch $a_1 \in A$ repräsentiert.

- Dann hat Paul Erdős die Zahl $z(a_1) = 0$.

Für jeden anderen Autor $a_i \in A$ mit $i > 1$ gilt:

- a_i hat die Zahl $z(a_i) = k{+}1$ falls k die kleinste Erdős-Zahl unter den Koautoren von a_i ist.
- a_i hat die Zahl $z(a_i) = \infty$, wenn a_i keinen Koautor mit endlicher Erdős-Zahl hat.

Im folgenden wollen wir die Erdős-Zahl von allen Autoren A berechnen. Als **Eingabe** dient ein ungerichteter Graph $G = (A, E)$. Es existiert eine Kante $\{a, b\} \in E$, genau dann wenn Autoren a und b Koautoren sind, d.h. mindestens einen Artikel gemeinsam verfassten. Die **Ausgabe** soll ein Array $Z[1 \ldots n]$ sein, wobei $Z[i]$ die Erdős-Zahl $z(a_i)$ von Autor a_i speichert.

a) Beschreiben Sie **natürlich-sprachlich** einen möglichst effizienten Algorithmus Erdos, der dieses Problem löst. Dieser soll auf einem aus der Vorlesung bekannten Algorithmus Vl aufbauen. **Nennen Sie Vl und diskutieren Sie die notwendigen Modifikationen** um Z zu berechnen. Begründen Sie kurz die **Korrektheit Ihres Ansatzes** (kein Beweis notwendig).

   Hierbei können Sie Vl als Blackbox verwenden, das heißt es müssen insbesondere keine Datenstrukturen (o.Ä.) diskutiert werden.

[1] Zwei Autoren sind genau dann Koautoren, wenn sie mindestens ein Papier gemeinsam publizierten.

b) Beschreiben Sie Ihren effizienten Algorithmus in **Pseudo-Code**. Definieren und initialisieren Sie zunächst alle genutzten Datenstrukturen (inkl. der Graph-Repräsentation).

c) Analysieren Sie die Laufzeit Ihres Algorithmus **asymptotisch exakt** für den Fall, das alle Autoren eine endliche Erdős-Zahl haben.

Aufgabe 4: Augensumme mehrerer Würfel — $4 + 6 + 8 + 2 + 4 = 24$ Punkt(e)

Gegeben seien n Spielwürfel mit Ziffern 1 bis 6. Wir werfen diese nacheinander, und notieren die Augenzahlen $(z_1, \ldots, z_n)$ mit $z_i \in \{1, 2, 3, 4, 5, 6\}$ für alle $1 \leq i \leq n$. Gesucht ist die Anzahl $N[n, k]$ an Kombinationen $(z_1, \ldots, z_n)$, so dass die Augensumme $\sum_{i=1}^{n} z_i$ exakt den Wert k mit $n \leq k \leq 6n$ annimmt, d.h.

$$N[n,k] \quad = \quad \left| \left\{ (z_1, \ldots, z_n) \;\middle|\; z_i \in \{1,2,3,4,5,6\} \; \forall 1 \leq i \leq n \text{ mit } \sum_{i=1}^{n} z_i = k \right\} \right|.$$

Hinweis: Sie können annehmen, dass die Eingabe $n \geq 1$ und $n \leq k \leq 6n$ erfüllt.

a) Betrachten Sie folgenden naiven Algorithmus: wir testen *jede* Kombination $(z_1, \ldots, z_n)$ mit $1 \leq z_i \leq 6$ für alle $1 \leq i \leq n$, und zählen wie viele Kombination die Augensumme k ergeben. Geben Sie eine möglichst scharfe untere Schranken für die Laufzeit dieses Algorithmus an. Begründen Sie Ihre Antwort kurz.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Wir möchten das Problem nun mit Hilfe eines **dynamischen Programms** lösen. Hierfür berechnen wir $N[n, k]$ **rekursiv**. Geben Sie eine geeignete Rekursionsgleichung inklusive minimaler notwendiger Basisfälle an.

Rekursion: $N[n, k] =$ _____

Basisfälle:

⇑⇑⇑ _____ / 6 Punkt(e) ⇑⇑⇑

c) Beschreiben Sie einen Algorithmus in **Pseudo-Code**, der Ihre Rekursionsgleichung $N[n, k]$ mittels einer $n \times k$ Matrix effizient löst. Nutzen Sie `return` um das Endergebnis zurückzugeben.

`return` _____

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

d) Leiten Sie die Laufzeit Ihres Algorithmus asymptotisch exakt her.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

e) Beschreiben Sie kurz natürlich-sprachlich, wie sich der Speicherverbrauch von $\Theta(n \cdot k)$ auf $\mathcal{O}(n)$ reduzieren lässt.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

Aufgabe 5: Doppelt-verkettete Liste — $2 + 2 + 2 + 14 = 20$ Punkt(e)

In der Vorlesung implementierten wir eine doppelt-verkette Liste zum Speichern von Ganzzahlen (`int`) sinngemäß wie folgt:

```
struct Element {
  int value;
  Element* prev;
  Element* next;
};
```

```
struct List {
  Element* head;
  Element* tail;
};
```

Hier beschreibt `Element*` einen Zeiger (Pointer) auf ein Objekt vom Typ `Element`. Ein Zeiger, der ins „Nichts“ zeigt, hat den Wert `nullptr`. Alle Zeiger haben initial den Wert `nullptr`.

a) Betrachten Sie folgende Funktionen `size` und `is_empty`:

```
// Zähle Elemente in lst
int size(List* lst) {
  Element* it = lst->head;
  int k = 0;
  while(it != nullptr) {
    it = it->next;
    k = k + 1;
  }
  return k;
}
```

```
// Ermittle ob lst leer ist
bool is_empty(List* lst) {
  if (size(lst) == 0) {
    return true;
  } else {
    return false;
  }
}
```

Ermitteln Sie die Laufzeit von `is_empty` asymptotisch exakt[2] und definieren Sie alle verwendeten Variablen.

Laufzeit von `is_empty`: _____

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Implementieren Sie `is_empty_opt` mit best-möglicher Laufzeit in **Pseudocode**.

```
bool is_empty_opt(List* lst) {




}
```

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

c) Ermitteln Sie die Laufzeit von `is_empty_opt` asymptotisch exakt und definieren Sie alle verwendeten Variablen.

Laufzeit von `is_empty_opt`: _____

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

[2] Übereinstimmende obere und untere Schranken.

d) Zeigen Sie, dass wir mit Hilfe von `List` einen **Stack** (Stapelspeicher) simulieren können.

Implementieren Sie hierfür ein effizientes Funktionspaar `push`/`pop` in **Pseudocode** durch explizites Setzen von Zeigern. Analysieren Sie die Laufzeit Ihrer Implementierungen.

```
void push(List* lst, int value) {
  // Gleiches Verhalten wie push auf einem Stack.
  Element* x = new Element();
  x->value = value;









}
```

Laufzeit von `push`: _____

```
int pop(List* lst) {
  // Gleiches Verhalten wie pop auf einem Stack.
  // Gibt den Wert des entfernten Elements zurück.
  // Sie können annehmen, dass die Liste nicht leer ist.









}
```

Laufzeit von `pop`: _____

⇑⇑⇑ _____ / 14 Punkt(e) ⇑⇑⇑

Aufgabe 6: Huffman-Codes — 12 Punkt(e)

Geben Sie zu jedem Zeichen das Codewort eines Huffman-Codes an der zu folgender Häufigkeitstabelle gehört.

a	b	c	d	e	f	g
3	3	1	1	7	2	4

Immer wenn zwei Teilbäume vereint werden, können Sie beliebig anordnen. Sollten Sie zwischen zwei gleichwertigen Knoten auswählen müssen, können Sie beliebig entscheiden.

Der Huffman-Baum muss nicht angegeben werden und wird auch nicht bewertet.

a $\hat{=}$ _____

b $\hat{=}$ _____

c $\hat{=}$ _____

d $\hat{=}$ _____

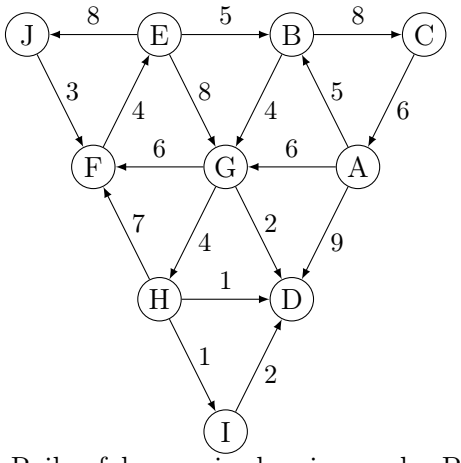
e $\hat{=}$ _____

f $\hat{=}$ _____

g $\hat{=}$ _____

⇑⇑⇑ _____ / 12 Punkt(e) ⇑⇑⇑

a) Wenden Sie Dijkstras Algorithmus auf folgenden gewichteten, gerichteten Graph mit **Startknoten** A an.



Geben Sie die Knoten in der Reihenfolge an, in der sie aus der Prioritätswarteschlange entfernt werden.

Reihenfolge: A,_____

Geben Sie den ersten Knoten an für den die Distanz von einem Wert kleiner ∞ erneut reduziert wird, bevor er aus der Prioritätswarteschlange entfernt wird.

Knoten: _____

⇑⇑⇑ _____ / 12 Punkt(e) ⇑⇑⇑

b) Sei $G = (V, E)$ ein gewichteter, gerichteter Graph und $w : E \to \mathbb{R}_{\geq 1}$ eine Kantengewichtsfunktion. Wir betrachten die folgende Transformation der Kantengewichte:

$$w'_a(e) = 2^{\log_a(w(e))}$$

i) **(4 Punkte)** Geben Sie ein $a \in \mathbb{N}_{>1}$ an, sodass die kürzesten Wege bzgl. w unter der Transformation w'_a erhalten bleiben und begründen Sie warum.

ii) **(4 Punkte)** Geben Sie ein Beispiel für ein $a \in \mathbb{N}_{>1}$ an, sodass die kürzesten Wege bzgl. w unter der Transformation w'_a *nicht* erhalten bleiben und begründen Sie an einem Beispiel warum.

iii) **(4 Punkte)** Verallgemeinern Sie das Beispiel aus ii) auf alle a außer das, das in Aufgabenteil i) verwendete. Beachten Sie, dass das in Aufgabenteil i) verwendete a das Einzige ist für das die kürzesten Wege erhalten bleiben.

⇑⇑⇑ _____ / 12 Punkt(e) ⇑⇑⇑

Aufgabe 8: Graphen

$8 + 10 = 18$ Punkt(e)

a) Gegeben ist ein ungerichteter Graph als Elternarray.

1	2	3	4	5	6	7	8	9	10
2	3	8	8	8	10	4	0	4	4

Geben Sie eine Adjazenzliste für diesen ungerichteten Graph an.

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

b) Zeigen oder Widerlegen Sie folgende Aussage:

Für jedes $n \in \mathbb{N}_{>0}$ existiert eine Reihenfolge in der die Zahlen $1, \ldots, n$ in einen binären Suchbaum eingefügt werden können, sodass der resultierende Baum auch ein gültiger Max-Heap ist.

⇑⇑⇑ _____ / 10 Punkt(e) ⇑⇑⇑

Wichtig: Lösungen auf dieser Seite werden nur dann berücksichtigt, wenn bei der entsprechenden Aufgabe ein Hinweis auf Seite 15 platziert wurde.