| | |
|---|---|
| Name: | Klausur-ID: |
| Vorname: | |
| Matrikelnummer: | |

# Karlsruher Institut für Technologie

Institut für Theoretische Informatik

Prof. Jörn Müller-Quade

11. April 2018

# Klausur Algorithmen I

| | | |
|---|---|---|
| Aufgabe 1. | Kleinaufgaben | 15 Punkte |
| Aufgabe 2. | Sortieren | 7 Punkte |
| Aufgabe 3. | Spannbäume | 12 Punkte |
| Aufgabe 4. | Heaps | 7 Punkte |
| Aufgabe 5. | Hashing | 9 Punkte |
| Aufgabe 6. | Dynamische Programmierung | 10 Punkte |

Bitte beachten Sie:

- Schreiben Sie auf **alle Blätter** der Klausur und Zusatzblätter Ihren Namen und Ihre Matrikelnummer.
- Schreiben Sie auf das Deckblatt Ihre Klausur-ID und nehmen Sie nach der Klausur den Zettel mit der Klausur-ID mit. Sie benötigen diese, um später online Ihre Note zu erfahren.
- Die Klausur enthält 18 Blätter. Die Bearbeitungszeit beträgt 120 Minuten.
- Die durch Übungsblätter gewonnenen Bonuspunkte werden erst nach Erreichen der Bestehensgrenze hinzugezählt. Die Bonuspunkte entscheiden nicht über das Bestehen der Klausur.
- Als Hilfsmittel ist ein beidseitig handbeschriebenes DIN-A4 Blatt zugelassen.
- Bitte kennzeichnen Sie deutlich, welche Aufgabe gewertet werden soll. **Bei mehreren angegebenen Möglichkeiten wird jeweils die schlechteste Alternative gewertet.**

| Aufgabe | mögliche Punkte | | | | | | | erreichte Punkte | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | a | b | c | d | e | f | Σ | a | b | c | d | e | f | Σ |
| 1 | 1 | 4 | 1 | 3 | 3 | 3 | 15 | | | | | | | |
| 2 | 2 | 2 | 3 | - | - | - | 7 | | | | - | - | - | |
| 3 | 2 | 2 | 1 | 2 | 2 | 3 | 12 | | | | | | | |
| 4 | 1 | 2 | 2 | 2 | - | - | 7 | | | | | - | - | |
| 5 | 3 | 2 | 4 | - | - | - | 9 | | | | - | - | - | |
| 6 | 1 | 2 | 5 | 2 | - | - | 10 | | | | | - | - | |
| Σ | | | | | | | 60 | | | | | | | |

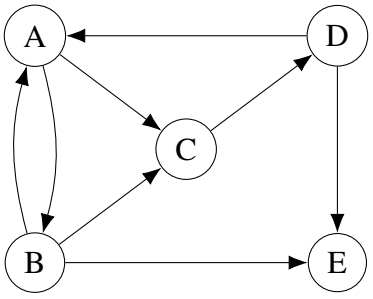
| Bonuspunkte: | | Gesamtpunktzahl: | |
|---|---|---|---|

**Aufgabe 1.** Kleinaufgaben [15 Punkte]

Bearbeiten Sie die folgenden Aufgaben. Begründen Sie Ihre Antworten jeweils kurz.
*Reine Ja/Nein-Antworten ohne Begründung geben keine Punkte.*

**a.** Gilt $2n \in o(n)$? Beweisen Sie Ihre Antwort! [1 Punkt]

**b.** Im folgenden Graphen wird eine Breitensuche ausgehend vom Knoten *A* ausgeführt, sodass bei mehreren Möglichkeiten jeweils die Knoten in alphabetischer Reihenfolge ausgewählt werden. Geben Sie die Knoten in der Reihenfolge an, wie sie von dieser Breitensuche jeweils zum ersten Mal betrachtet werden. Geben Sie außerdem an, bei welchen Kanten es sich um *tree*-, *backward*- bzw. *cross*-Kanten handelt.

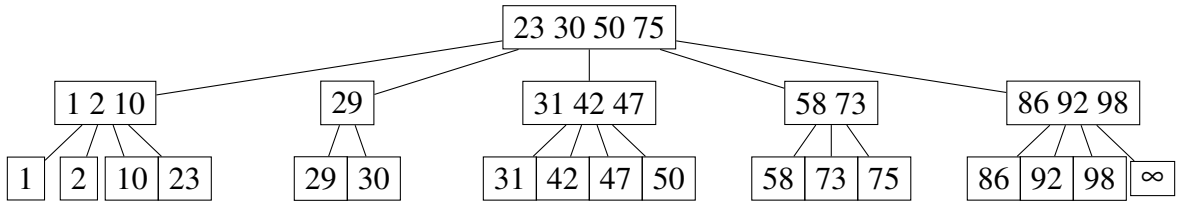



[4 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 1**

**c.** Ist der folgende Baum ein (2,4)-Baum nach Definition aus der Vorlesung? Begründen Sie Ihre Antwort!




[1 Punkt]

**d.** Betrachten Sie die folgende Aussage über Rekurrenzen:
Seien $a, b, c, d$ positive Konstanten und $n \in \mathbb{N}$. Ist

$$T(n) = \begin{cases} a & \text{für } n = 1 \\ d \cdot T(\lceil n/b \rceil) + c \cdot n & \text{für } n > 1 \end{cases},$$

dann gilt

$$T(n) \in \begin{cases} \Theta(n) & \text{falls } d \leq b \\ \Theta(n^{\log_b d}) & \text{falls } d > b \end{cases}.$$

Ist diese Aussage wahr? Beweisen Sie Ihre Antwort! [3 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 1**

**e.** Es sei $G = (V, E)$ ein gerichteter und zusammenhängender Graph mit Kantengewichten aus $\mathbb{N}$. Es seien $A, B \in V$ zwei Knoten aus $G$ und $P$ die Menge aller Pfade von $A$ nach $B$. Es sei $P_{\min}^{A,B}$ die Menge aller kürzesten Pfade von $A$ nach $B$. Ein *zweitkürzester Pfad* von $A$ nach $B$ ist definiert als ein kürzester Pfad von $A$ nach $B$ aus der Menge $P \setminus P_{\min}^{A,B}$. Betrachten Sie folgenden Algorithmus:

1. Berechne Dijkstras Algorithmus in $G$ ausgehend von $A$. Sei $p$ der berechnete kürzeste Pfad von $A$ nach $B$.
2. Bestimme die Kante $e$ mit geringstem Gewicht auf $p$.
3. Setze $E' = E \setminus \{e\}$.
4. Berechne Dijkstras Algorithmus auf $G' = (V, E')$ ausgehend von $A$ und gebe den hierbei berechneten kürzesten Pfad von $A$ nach $B$ aus.

Berechnet dieser Algorithmus immer einen zweitkürzesten Pfad zwischen $A$ und $B$? Beweisen Sie Ihre Antwort! [3 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 1**

**f.** Es sei $L$ eine doppelt verkette Liste mit Head-Element, deren Elemente natürliche Zahlen enthalten. Geben Sie im Pseudocode einen Algorithmus *insertMiddle(L,a)* an, der als Übergabeparameter eine Zahl $a \in \mathbb{N}$ und $L$ erhält. Der Algorithmus soll in der Mitte von $L$ ein neues Listenelement mit Eintrag $a$ einfügen. Genauer sei $n$ die Anzahl Einträge in $L$ bevor das neue Element eingefügt wird (Head-Element nicht mitgezählt). Dann soll der Algorithmus das neue Element an Position $\lfloor n/2 \rfloor + 1$ eintragen. Die Laufzeit des Algorithmus soll in $O(n)$ liegen.

Sie können davon ausgehen, dass $L$ einen Zeiger *head* auf das Head-Element in $L$ bereit stellt und mindestens einen Eintrag enthält. Die Liste stellt keine anderen Funktionen oder Zeiger zu Verfügung, auch die Anzahl an Elementen in $L$ ist nicht bekannt. Sie müssen weder die Laufzeit noch die Korrektheit ihres Algorithmus beweisen. Kommentieren Sie Ihren Algorithmus zur besseren Verständlichkeit. [3 Punkte]

**Aufgabe 2.** Sortieren [7 Punkte]

**a.** Sortieren Sie das Array $\langle 15, 2, 13, 27, 1, 9, 12, 4 \rangle$ mit dem MergeSort-Algorithmus. Verwenden Sie zur Darstellung das Schema aus der Vorlesung und geben Sie bei jedem Schritt an, ob es sich um einen split- oder merge-Schritt handelt. [2 Punkte]

**b.** Nennen Sie zwei Vorteile von vergleichsbasiertem Sortieren im Vergleich zu ganzzahligem Sortieren! [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

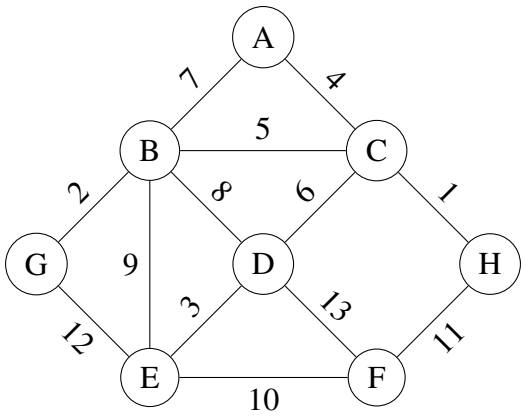
**Fortsetzung von Aufgabe 2**

**c.** Ein Sortieralgorithmus wird *stabil* genannt, wenn er die Reihenfolge von gleichwertigen Elementen nicht verändert, d.h. sind die Elemente $A$ und $B$ gleich und $A$ steht vor dem Sortieren vor $B$, so wird $A$ auch nach dem Sortieren vor $B$ stehen.

Der QuickSort-Algorithmus aus der Vorlesung ist nicht stabil. Geben Sie eine Modifikation von QuickSort an, sodass der Algorithmus stabil ist. Hierbei soll die grundsätzliche Funktionsweise und Laufzeit von QuickSort erhalten bleiben. Ihre Modifikation muss die In Place Eigenschaft des QuickSort Algorithmus *nicht* erhalten. Begründen Sie kurz, warum Ihre Modifikation funktioniert.
Für diese Aufgabe ist kein Pseudocode notwendig, es genügt, wenn Sie Ihre Modifikation eindeutig beschreiben. [3 Punkte]

**Aufgabe 3.** Spannbäume [12 Punkte]

**a.** Berechnen Sie einen minimalen Spannbaum des unten angegebenen Graphen mit dem Algorithmus von Kruskal. Geben Sie dabei die Kanten in der Reihenfolge an, in der sie der Algorithmus von Kruskal ausgibt.



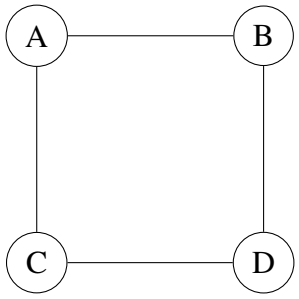

[2 Punkte]

**b.** Geben Sie je ein Pro-Argument für den Algorithmus von Jarník-Prim und den Algorithmus von Kruskal an. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 3**

**c.** Ergänzen Sie den folgenden Graphen um Kantengewichte, sodass bei der Berechnung eines minimalen Spannbaums durch die Algorithmen von Kruskal und Jarník-Prim beide Algorithmen die Kanten in unterschiedlicher Reihenfolge auswählen, wenn der Algorithmus von Jarník-Prim beim Knoten *A* beginnt. [1 Punkt]




Es sei im Folgenden $G = (V, E)$ ein ungerichteter und zusammenhängender Graph mit ganzzahligen Kantengewichten. Ein *maximaler* Spannbaum von *G* ist ein Spannbaum von *G*, bei dem die Summe der Kantengewichte maximal unter allen möglichen Spannbäumen von *G* ist.

**d.** Beweisen Sie: Die leichteste Kante auf einem Kreis in *G* wird nicht für einen maximalen Spannbaum von *G* benötigt. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 3**

**e.** Beweisen Sie: Für alle $S \subset V$ kann die schwerste Kante im Schnitt $C = \{\{u,v\} \in E : u \in S, v \in V \setminus S\}$ für einen maximalen Spannbaum verwendet werden. [2 Punkte]

**f.** Beschreiben Sie einen Algorithmus, der in Zeit $O(|E| \log |E|)$ in $G$ einen maximalen Spannbaum berechnet. Erklären Sie kurz, wieso Ihr Algorithmus wirklich einen maximalen Spannbaum berechnet und in der gewünschten Zeit läuft. Für diese Aufgabe ist kein Pseudocode und kein formaler Beweis notwendig. [3 Punkte]

**Aufgabe 4.** Heaps [7 Punkte]

*Hinweis:* In dieser Aufgabe handelt es sich bei Heaps stets um Min-Heaps.
**a.** Gegeben sei folgender binärer Heap in impliziter Darstellung. Zeichnen Sie den Heap in Baumdarstellung. [1 Punkt]

| 2 | 3 | 7 | 4 | 5 | 12 | 11 |
|---|---|---|---|---|---|---|

**b.** Führen Sie auf folgendem Array die Methode *buildHeapBackwards* aus der Vorlesung aus, um einen binären Heap aufzubauen. Geben Sie dabei das Array nach jeder Veränderung an. Einträge, die sich nicht verändern, dürfen Sie dabei leer lassen. In diesem Fall werten wir die letzte Zahl, die in einer der darüberliegenden Spalten steht. [2 Punkte]

| 10 | 4 | 6 | 2 | 5 | 9 | 3 |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

**Platz für Korrekturen:**

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | | | | |
| | | | | | | |
| | | | | | | |

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 4**

**c.** Führen Sie auf folgendem binärem Heap die Methode *deleteMin* aus der Vorlesung aus. Geben Sie dabei das Array nach jeder Veränderung an. Einträge, die sich nicht verändern, dürfen Sie dabei leer lassen. In diesem Fall werten wir die letzte Zahl, die in einer der darüberliegenden Spalten steht. [2 Punkte]

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 6 | 9 | 8 | 5 | 7 | 11 | 10 |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

**Platz für Korrekturen:**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 4 | 6 | 9 | 8 | 5 | 7 | 11 | 10 |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

**d.** Geben Sie möglichst genau die Zeitkomplexität der Methoden *min*, *insert*, *deleteMin* und *buildHeap* eines binären Heaps im O-Kalkül an. [2 Punkte]

**Aufgabe 5.** Hashing [9 Punkte]

**a.** Gegeben sei folgende Hashtabelle, welche lineare Suche zur Kollisionsauflösung verwendet:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 34 | | | | 24 | 65 | 44 | 75 | 18 | 37 |

Es ist bekannt, dass 18 als *erstes* eingefügt wurde und dass 65 zeitlich *vor* 24 eingefügt wurde. Als Hashfunktion wurde

$$h(x) := x \; mod \; 10$$

verwendet.

Bestimmen Sie die vollständige Reihenfolge, in welcher die Schlüssel in die Hashtabelle eingefügt wurden. Begründen Sie ihre Antwort.
Sie können folgende Tabellen als Hilfe benutzen:

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |

[3 Punkte]

**b.** Nennen Sie zwei Vorteile von Hashing mit verketteten Listen gegenüber Hashing mit linearer Suche. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 5**

**c.** In einem Lebensmittelladen wurde ein automatisches Bestellungssystem installiert, das mit einer FIFO Queue arbeitet. Jedes mal, wenn ein Kunde eine Bestellung aufgibt, werden die Bestelldaten (Produkt, Menge, Name des Kunden) in die Queue eingefügt. Der Besitzer des Ladens arbeitet die einzelnen Aufträge in der Queue nacheinander ab. Um zu vermeiden, dass ein Produkt ausverkauft ist, möchte er jeder Zeit in seinem System abfragen können, wie viele Einheiten eines Produktes sich gerade in der Queue befinden.

Beschreiben Sie, wie Sie das Bestellsystem so modifiziern würden, dass folgende Operationen in erwartet $O(1)$ Zeit ausgeführt werden können.

- `enqueue`($B$): Fügt eine Bestellung $B := \langle$Produkt $p$, Menge $m$, Name des Kunden$\rangle$ an das Ende der Queue ein.
- `dequeue`(): Extrahiert die aktuelle Bestellung $B$ aus der Queue.
- `query`($p$): Gibt die Menge von Produkt $p$ zurück, die sich aktuell in der Queue befindet.

*Hinweis:* Es ist bekannt, dass der Laden insgesamt $n$ verschiedene Produkte führt. Die im Bestellungssystem installierte Queue kann maximal $q$ Bestellungen enthalten und es gilt $q < n$. Ihre Datenstrukturen dürfen $O(q)$ Platz verwenden. Für diese Aufgabe ist kein Pseudocode notwendig.

[4 Punkte]

**Aufgabe 6.** Dynamische Programmierung [10 Punkte]

Gegeben sei eine binäre Matrix $A$ der Größe $M \times N$, d.h., eine Matrix, die nur 0 und 1 enthält. Gesucht ist die Größe $\mathrm{lcss}_1(A)$ der größten zusammenhängenden **quadratischen** Teilmatrix, die nur 1 enthält.
Beispielsweise ist $\mathrm{lcss}_1(A) = 3$ für die folgende Matrix $A \in \{0,1\}^{5\times 6}$:

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ 1 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

**a.** Markieren Sie eine größte zusammenhängende quadratische 1er-Teilmatrix in der untenstehenden $6 \times 8$ Matrix und geben Sie deren Größe an. [1 Punkt]

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

**Korrektur:**

$$\begin{pmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \end{pmatrix}$$

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 6**

**b.** Angenommen, Sie kennen zu einer gegebenen Matrix $A$ lediglich Position und Größe der **eindeutig bestimmten** größten zusammenhängenden quadratischen 1er-Teilmatrix, die restlichen Einträge sind Ihnen nicht bekannt. Dies sei hier beispielhaft visualisiert (ein „?“ steht dabei für ein unbekanntes Bit, die angegebenen Bits stellen die größte zusammenhängende quadratische 1er-Teilmatrix dar):

$$\begin{pmatrix} ? & ? & ? & ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & ? & ? & ? & ? & ? & ? \end{pmatrix} \qquad \begin{pmatrix} ? & ? & ? & ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & ? & ? & ? & ? & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & 1 & 1 & 1 & 1 & ? & ? \\ ? & ? & ? & ? & ? & ? & ? & ? & ? & ? \end{pmatrix}$$

Können Sie in beiden Fällen eindeutig die

1. Positionen
2. Größen

der jeweils größten zusammenhängenden quadratischen 1er-Teilmatrizen der grau markierten Flächen angeben, obwohl die restlichen Einträge der Matrizen nicht bekannt sind? Begründen Sie Ihre Antworten. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

**Fortsetzung von Aufgabe 6**

**c.** Entwerfen Sie nach dem Entwurfsprinzip der dynamischen Programmierung einen Algorithmus, welcher zu einer gegebenen Matrix $A \in \{0,1\}^{M \times N}$ die gesuchte Größe $\mathrm{lcss}_1(A)$ berechnet. Die Laufzeit des Algorithmus soll dabei in $\mathrm{O}(M \cdot N)$ liegen. Für diese Aufgabe ist kein Pseudocode notwendig, es genügt, wenn Sie Ihren Algorithmus eindeutig beschreiben. [5 Punkte]

**d.** Analysieren Sie die Laufzeit und den Speicherverbrauch Ihres Algorithmus im O-Kalkül. [2 Punkte]

**Konzeptpapier für Nebenrechnungen.**