Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Jeremias Weihmann

Sommersemester 2012
Lösungen der Klausur
28. Juli 2012

Grundlagen: Algorithmen und Datenstrukturen

Name	Vorname	Studiengang	Matrikelnummer
.....	.....	☐ Diplom ☐ Bachelor ☐ Master ☐ Lehramt ☐ Inform. ☐ BioInf. ☐ WirInf. ☐ Mathe ☐ Games	.....

Hörsaal	Reihe	Sitzplatz	Unterschrift
.....	.....	.....	.....

Allgemeine Hinweise zur Klausur

- Versehen Sie bitte alle von Ihnen genutzten Blätter mit Vornamen, Namen und Matrikelnummer.
- Bitte legen Sie Ihren Studentenausweis und einen amtlichen Lichtbildausweis auf die Hörsaalbank.
- Bitte schreiben Sie *nicht* in roter oder grüner Farbe bzw. mit Bleistift.
- Außer Ihrem Schreibgerät und einem handbeschriebenen DIN-A4-Blatt sind keine weiteren Hilfsmittel erlaubt.
- Die Bearbeitungszeit beträgt 150 Minuten.

Hörsaal verlassen von ..... bis ..... / von ..... bis .....

Vorzeitig abgegeben um .....

Besondere Bemerkungen:

	A1	A2	A3	A4	A5	A6	A7	A8	A9	B	Σ	Korrektor
Gesamt	8	12	9	6	10	8	10	8	9	5	80+5	
Erstkorrektur												
Zweitkorrektur												

Aufgabe 1 (8 Punkte)

Beantworten Sie die folgenden Fragen.

(a) Nennen Sie eine (für (a, b)-Bäume charakteristische) Invariante, die bei (a, b)-Bäumen stets erfüllt ist.

(b) Geben Sie eine jeweils möglichst gute obere und untere Schranke für die Baumtiefe eines (a, b)-Baumes für n Elemente (d.h. mit $n + 1$ Blättern) an.

(c) Wir betrachten ein dynamisches Array mit $\beta = 2$ und $\alpha = 4$ definiert wie in der Vorlesung. Geben Sie jeweils an, wie groß der Anteil der belegten Felder des Arrays ist, unmittelbar nachdem eine Größenänderung des Arrays stattgefunden hat, die durch eine

(i) pushBack-Operation ausgelöst wurde.

(ii) popBack-Operation ausgelöst wurde.

(d) Nennen Sie zwei Probleme, die mithilfe einer modifizierten Tiefensuche gelöst werden können.

(e) Geben Sie für MergeSort und für QuickSort jeweils asymptotisch optimale obere und untere Schranken für die Laufzeit an.

Lösungsvorschlag

(a) Die Zahl der Kindknoten jedes inneren Knotens außer der Wurzel ist mindestens a und maximal b. Ein Splitschlüssel entspricht stets dem größten Blatt im zu dem Schlüssel korrespondieren Unterbaum. Alle Blätter befinden sich auf derselben Ebene.

(b) Ein (a, b)-Baum mit n Elementen hat Baumtiefe mindestens $\lceil \log_b(n+1) \rceil$ und maximal $1 + \left\lfloor \log_a \left(\frac{n+1}{2} \right) \right\rfloor$.

(c) Der Anteil der belegten Felder ist jeweils genau 1/2.

(d) DAG-Erkennung, Finden der Zusammenhangskomponenten, Finden der Blöcke, Finden der starken Zusammenhangskomponenten

(e) MergeSort: Untere Schranke $\Omega(n \log n)$, obere Schranke $\mathcal{O}(n \log n)$. QuickSort: Untere Schranke $\Omega(n \log n)$, obere Schranke $\mathcal{O}(n^2)$.

Aufgabe 2 (12 Punkte)

Gegeben seien die folgenden Funktionen von $\mathbb{N}$ nach $\mathbb{R}$:

$$f(n) = n^{\sqrt{n}}, \quad g(n) = \ln(2n), \quad h(n) = n + [1 + (-1)^n] \cdot n^3, \quad k(n) = n^2$$

Kreuzen Sie in den Zeilen (a) bis (f) jeweils „$\Delta = \mathcal{O}$“ an, wenn $a(n) \in \mathcal{O}(b(n))$ gilt, kreuzen Sie „$\Delta = \Omega$“ an, wenn $a(n) \in \Omega(b(n))$ gilt, und kreuzen Sie „Unvergleichbar“ an, wenn keine der beiden anderen Antwortmöglichkeiten zutrifft.

(a)	$f(n) \in \Delta(g(n))$	☐ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar
(b)	$f(n) \in \Delta(h(n))$	☐ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar
(c)	$f(n) \in \Delta(k(n))$	☐ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar
(d)	$g(n) \in \Delta(h(n))$	☐ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar
(e)	$g(n) \in \Delta(k(n))$	☐ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar
(f)	$h(n) \in \Delta(k(n))$	☐ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar

Begründen Sie Ihre Antworten jeweils mit einem kurzen Beweis.

Jede korrekt angekreuzte Zeile gibt einen Punkt, jede korrekte Begründung gibt *zusätzlich* einen Punkt. Eine falsch angekreuzte Zeile oder eine falsche Begründung gibt *keine* Punktabzüge.

Lösungsvorschlag

(a)	$f(n) \in \Delta(g(n))$	☐ $\Delta = \mathcal{O}$	☒ $\Delta = \Omega$	☐ Unvergleichbar
(b)	$f(n) \in \Delta(h(n))$	☐ $\Delta = \mathcal{O}$	☒ $\Delta = \Omega$	☐ Unvergleichbar
(c)	$f(n) \in \Delta(k(n))$	☐ $\Delta = \mathcal{O}$	☒ $\Delta = \Omega$	☐ Unvergleichbar
(d)	$g(n) \in \Delta(h(n))$	☒ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar
(e)	$g(n) \in \Delta(k(n))$	☒ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☐ Unvergleichbar
(f)	$h(n) \in \Delta(k(n))$	☐ $\Delta = \mathcal{O}$	☐ $\Delta = \Omega$	☒ Unvergleichbar

Zunächst zeigen wir, dass $f(n) \in \Omega(n^3)$ gilt. Dazu ist

$$\exists c > 0 \; \exists n_0 \; \forall n \geq n_0 : n^{\sqrt{n}} \geq c \cdot n^3$$

zu zeigen. Dies ist offensichtlich wahr, wie man an der Wahl $c = 1$ und $n_0 = 9$ ablesen kann.

Wir zeigen (b), d.h. $f(n) \in \Omega(h(n))$. Wegen der Transitivität von Ω ist es hinreichend zu zeigen, dass $n^3 \in \Omega(h(n))$, d.h.

$$\exists c > 0 \; \exists n_0 \; \forall n \geq n_0 : n^3 \geq c \cdot n + [1 + (-1)^n] \cdot n^3$$

gilt. Hierfür stellen wir fest, dass

$$n^3 = 1/3 \cdot (n^3 + 2n^3) \geq 1/3 \cdot (n + 2n^3) \geq 1/3 \cdot (n + [1 + (-1)^n] \cdot n^3)$$

für alle $n \geq 1$ gilt, d.h. wir können $c = 1/3$ und $n_0 = 1$ wählen.

Wir zeigen (c), d.h. $f(n) \in \Omega(k(n))$: Dies ist offensichtlich, da für $c = 1$ und alle $n \geq n_0 = 1$ gerade $n^3 = c \cdot n \cdot n^2 \geq c \cdot n^2$ gilt.

Nun zeigen wir zunächst $g(n) \in \mathcal{O}(n)$. Hinreichend dafür ist $g'(n) \in \mathcal{O}(\frac{d}{dn}n)$. Die Ableitung von $\ln(2n) = \ln(n) + \ln(2)$ ist $1/n$, die von n ist 1. Wegen

$$\lim_{n\to\infty} \frac{1/n}{1} = 0$$

folgt die Behauptung.

Wir zeigen (d), d.h. $g(n) \in \mathcal{O}(h(n))$: Wegen der Transitivität von $\mathcal{O}$ reicht es aus, $n \in \mathcal{O}(n)$ zu zeigen, d.h.:

$$\exists c > 0\ \exists n_0\ \forall n \geq n_0 : n \leq c \cdot n + [1 + (-1)^n] \cdot n^3$$

Dies ist wahr, wie man an der Wahl $c = 1$ und $n_0 = 1$ leicht erkennen kann.

Daraus folgt automatisch $h(n) \in \Omega(g(n))$ woraus wegen der Transitivität von Ω sofort $f(n) \in \Omega(g(n))$, **d.h. (a) folgt.**

Wir zeigen (e), d.h. $g(n) \in \mathcal{O}(k(n))$: Wegen der Transitivität von $\mathcal{O}$ reicht es aus, $n \in \mathcal{O}(n)$ zu zeigen, d.h.:

$$\exists c > 0\ \exists n_0\ \forall n \geq n_0 : n \leq c \cdot n^2$$

Wir zuvor erkennt man dies leicht an der Wahl $c = 1$ und $n_0 = 1$.

Wir zeigen (f), d.h., dass $h(n)$ und $k(n)$ bezüglich $\mathcal{O}$ und Ω unvergleichbar sind. Dafür müssen wir

$$\forall c > 0\ \forall n_0\ \exists n \geq n_0 : n + [1 + (-1)^n] \cdot n^3 > c \cdot n^2$$

und

$$\forall c > 0\ \forall n_0\ \exists n \geq n_0 : n^2 > c \cdot n + [1 + (-1)^n] \cdot n^3$$

zeigen. Zunächst die erste Aussage: Seien $c > 0$ und n_0 beliebig gewählt. Wir wählen $n \geq n_0$ als gerade Zahl, d.h. $n + [1 + (-1)^n] \cdot n^3 = n + 2n^3$. Für n muss $n + 2n^3 > c \cdot n^2$ gelten. Dies ist erfüllt, wenn wir ein beliebiges gerades $n \geq n_0$ wählen, das $1/n + 2n > c$ erfüllt, also zum Beispiel

$$n := \max\{n_0, \lceil c/2 \rceil + 1\} + \max\{n_0, \lceil c/2 \rceil + 1\} \bmod 2.$$

Nun die zweite Aussage: Seien $c > 0$ und n_0 beliebig gewählt. Wir wählen $n \geq n_0$ als ungerade Zahl, d.h. $n + [1 + (-1)^n] \cdot n^3 = n$. Für n muss $n^2 > c \cdot n$ gelten. Dies ist erfüllt, wenn wir ein beliebiges ungerades $n \geq n_0$ wählen, das $n > c$ erfüllt, also zum Beispiel

$$n := \max\{n_0, \lceil c \rceil + 1\} + [\max\{n_0, \lceil c \rceil + 1\} + 1] \bmod 2.$$

Wir merken an dieser Stelle an, dass alle Beweise auch mithilfe von Limes, Limes Superior und Limes Inferior geführt werden können.

Aufgabe 3 (9 Punkte)

Wir betrachten eine Schlüsselmenge Key $= \{\text{blau}, \text{rot}, \text{gelb}, \text{schwarz}, \text{weiß}\}$, sowie eine Hashtabelle der Größe $m = 4$. Es seien folgende Hashfunktionen gegeben:

f_1:	blau $\to 0$	rot $\to 0$	gelb $\to 2$	schwarz $\to 2$	weiß $\to 3$
f_2:	blau $\to 1$	rot $\to 2$	gelb $\to 1$	schwarz $\to 3$	weiß $\to 3$
f_3:	blau $\to 0$	rot $\to 2$	gelb $\to 2$	schwarz $\to 0$	weiß $\to 1$
f_4:	blau $\to 0$	rot $\to 3$	gelb $\to 1$	schwarz $\to 3$	weiß $\to 1$
f_5:	blau $\to 1$	rot $\to 0$	gelb $\to 1$	schwarz $\to 1$	weiß $\to 3$
f_6:	blau $\to 2$	rot $\to 0$	gelb $\to 0$	schwarz $\to 3$	weiß $\to 2$
f_7:	blau $\to 1$	rot $\to 1$	gelb $\to 0$	schwarz $\to 3$	weiß $\to 3$

Geben Sie für jede der folgenden drei Familien von Hashfunktionen jeweils an, ob sie im Kontext der gegebenen Schlüsselmenge und Hashtabelle 1-universell ist. Begründen Sie Ihre Aussagen.

(a) $\mathcal{H}_1 = \{f_1, f_2, f_3, f_4\}$

(b) $\mathcal{H}_2 = \{f_4, f_5, f_6, f_7\}$

(c) $\mathcal{H}_3 = \{f_1, f_2, f_3, f_4, f_5, f_6, f_7\}$

Lösungsvorschlag

Eine Familie $\mathcal{H}$ von Hashfunktionen ist im Kontext einer Schlüsselmenge Key und einer Hashtabelle der Größe m nach Definition 1-universell, wenn für alle Werte $x, y \in$ Key mit $x \neq y$ gilt

$$\frac{|\{f \in \mathcal{H} \mid f(x) = f(y)\}|}{|\mathcal{H}|} \leq \frac{1}{m} .$$

Für $m = 4$ und $|\mathcal{H}| = 4$ bedeutet dies, dass wir nicht mehr als eine Kollision pro Paar x, y haben dürfen. Für $|\mathcal{H}| = 7$ dürfen wir nicht mehr als $7/4$ also auch nur eine Kollision pro Paar haben. Wir stellen zunächst eine Tabelle der Paare und ihrer Kollisionen auf:

Paar	f_1	f_2	f_3	f_4	f_5	f_6	f_7
blau/rot	x						x
blau/gelb		x			x		
blau/schwarz			x		x		
blau/weiß						x	
rot/gelb			x			x	
rot/schwarz				x			
rot/weiß							
gelb/schwarz	x				x		
gelb/weiß				x			
schwarz/weiß		x					x

Da sowohl $\mathcal{H}_1$ als auch $\mathcal{H}_2$ in jeder Zeile höchstens eine Kollision haben, sind beide Familien von Hashfunktionen 1-universell. Die Familie $\mathcal{H}_3$ ist nicht 1-universell, da beispielsweise das Paar blau/rot in f_1 und f_7 jeweils eine Kollision also insgesamt 2 Kollisionen hat.

Aufgabe 4 (6 Punkte)

Sortieren Sie die Zahlenfolge $523, 126, 67, 1, 500, 34, 21, 229, 9, 123$ mit RadixSort. Geben Sie für jede Sortierphase den Inhalt der Buckets sowie die durch die Sortierphase entstehende Zahlenfolge an.

Lösungsvorschlag

Wir sortieren nach der letzten Ziffer.

0	1	2	3	4	5	6	7	8	9
500	1		523	34		126	67		229
	21		123						9

Wir erhalten daraus das folgende Zwischenresultat: $500, 1, 21, 523, 123, 34, 126, 67, 229, 9$

Wir sortieren nach der vorletzten Ziffer.

0	1	2	3	4	5	6	7	8	9
500		21	34			67			
1		523							
9		123							
		126							
		229							

Wir erhalten daraus das folgende Zwischenresultat: $500, 1, 9, 21, 523, 123, 126, 229, 34, 67$

Wir sortieren nach der drittletzten Ziffer.

0	1	2	3	4	5	6	7	8	9
1	123	229			500				
9	126				523				
21									
34									
67									

Wir erhalten daraus das folgende Endresultat: $1, 9, 21, 34, 67, 123, 126, 229, 500, 523$

Aufgabe 5 (10 Punkte)

Für eine Menge von $n > 0$ verschiedenen Zahlen ist der Median definiert als das $\lceil n/2 \rceil$-kleinste Element dieser Menge. Wir betrachten die folgende Modifikation von QuickSelect gegeben in Pseudo-Code. Diese findet in einem Eingabearray $(x_1, x_2, \ldots, x_n)$ bestehend aus n verschiedenen Zahlen das k-kleinste Element, wobei $k \in [n]$:

Algorithmus 1: Modifizierter QuickSelect

```
1  quickSelect(int[] (x_1, x_2, ..., x_n), int k)
2  {
3    Sei m der Median von (x_1, x_2, ..., x_n).
4    Sei L ein Array mit genau den Elementen x_i < m, i ∈ {1, ..., n}.
5    Sei R ein Array mit genau den Elementen x_i > m, i ∈ {1, ..., n}.
6    Falls L genau k − 1 Elemente enthält, dann gib m aus.
7    Falls L mindestens k Elemente enthält, dann gib das Ergebnis von
8      quickSelect(L, k) aus.
9    Andernfalls gib das Ergebnis von quickSelect(R, k − |L| − 1) aus.
10 }
```

Angenommen, man kann den Median einer n-elementigen Menge in linearer Zeit in n finden (d.h. in Zeit maximal $cn + d$ für zwei Konstanten $c, d > 0$). Zeigen Sie, dass unter dieser Annahme die gegebene Modifikation von QuickSelect Laufzeit $\mathcal{O}(n)$ hat. Argumentieren Sie dabei exakt, d.h. erklären Sie unter anderem, welche Laufzeit die einzelnen Code-Zeilen bei geeigneter Implementierung haben.

Lösungsvorschlag

Zunächst zeigen wir, welche Laufzeiten die einzelnen Zeilen bei geeigneter Implementierung haben:

Der Aufruf der Funktion ist maximal linear. Die dritte Zeile ist unserer Annahme entsprechend maximal linear. Wir betrachten nun die vierte Zeile: Man läuft das Eingabearray durch und zählt, wie viele Elemente kleiner als m sind. Anschließend wird ein entsprechend großes neues Array L erstellt und diese Zahlen in L kopiert. Der Aufwand für diese Prozedur ist maximal linear. Entsprechendes gilt für die fünfte Zeile. Die sechste Zeile besitzt konstante Kosten. Die siebte und achte Zeile besitzt insgesamt konstante Kosten für den Test und die Einleitung des rekursiven Aufrufs. Entsprechendes gilt für die neunte Zeile.

Wir können damit eine obere Schranke $T(n)$ für die Laufzeit bei einem Eingabearray der Länge n als Rekursion formulieren. Eine kritische Beobachtung dabei ist, dass die Länge von L und R jeweils maximal $n/2$ ist, und damit die Eingabe beim rekursiven Aufruf maximal halb so groß ist wie ursprüngliche Eingabe. Wir definieren unsere obere Laufzeitschranke entsprechend:

$$T(n) = \begin{cases} T(n/2) + c'n, & \text{falls } n > 1 \\ a, & \text{falls } n = 1 \end{cases}$$

Hierbei ist $c' > 0$ eine geeignet gewählte Konstante ist, sodass $c'n$ eine obere Schranke für die Gesamtlaufzeit der Prozedur ohne die Zeit für den rekursiven Aufruf ist. $a > 0$ wiederum ist der konstante Aufwand bei einer Eingabe der Länge 1.

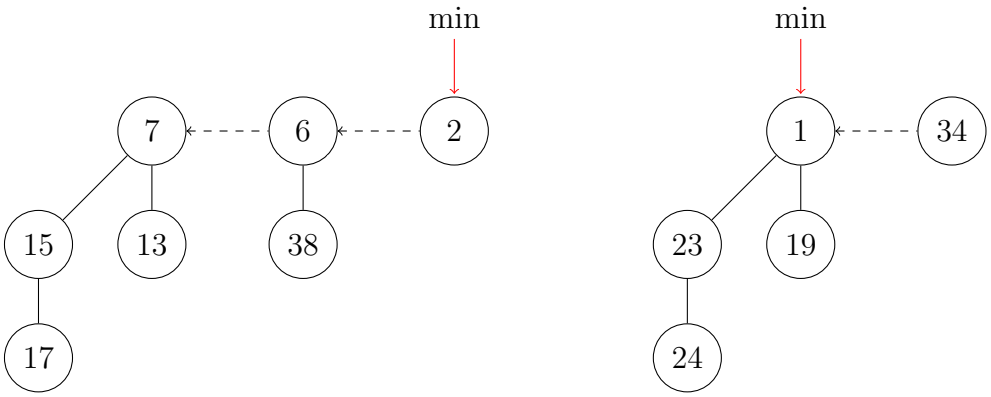
Da die Zahl 1 der rekursiven Aufrufe größer als der Verkleinerungsfaktor $1/2$ ist, kann der erste Fall des vereinfachten Master-Theorems angewendet werden. Dieses liefert nun $T(n) \in \Theta(n)$. Der Algorithmus hat also Laufzeit $\mathcal{O}(n)$.

Man kann auch ohne das Master-Theorem argumentieren: Es finden maximal $\mathrm{ld}(n)$ rekursive Aufrufe statt. Im i-ten rekursiven Aufruf haben wir Aufwand maximal $\max\{c, a\} \cdot n/2^i$. Die Gesamtlaufzeit ist also nach oben beschränkt durch

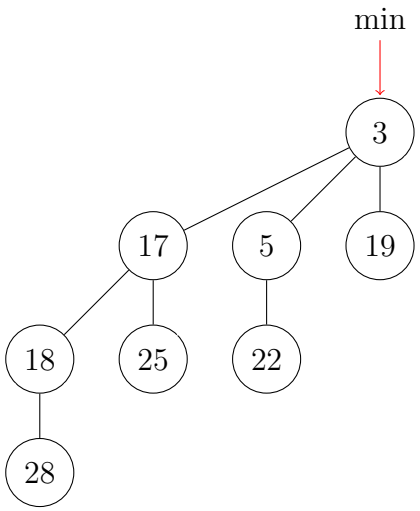
$$\begin{aligned}\sum_{i=0}^{\lfloor \mathrm{ld}(n) \rfloor} \max\{c, a\} \cdot \frac{n}{2^i} &= \max\{c, a\} \cdot n \cdot \sum_{i=0}^{\lfloor \mathrm{ld}(n) \rfloor} \frac{1}{2^i} < \max\{c, a\} \cdot n \cdot \sum_{i=0}^{\infty} \frac{1}{2^i} \\ &= \max\{c, a\} \cdot n \cdot 2 \in \mathcal{O}(n).\end{aligned}$$

Aufgabe 6 (8 Punkte)

(a) Führen Sie eine Merge-Operation auf den folgenden beiden Binomial-Heaps durch. Zeichnen Sie den resultierenden Binomial-Heap.



(b) Führen Sie hintereinander **zwei** DeleteMin-Operationen auf dem folgenden Binomial-Heap durch. Zeichnen Sie den Binomial-Heap nach jeder DeleteMin-Operation.

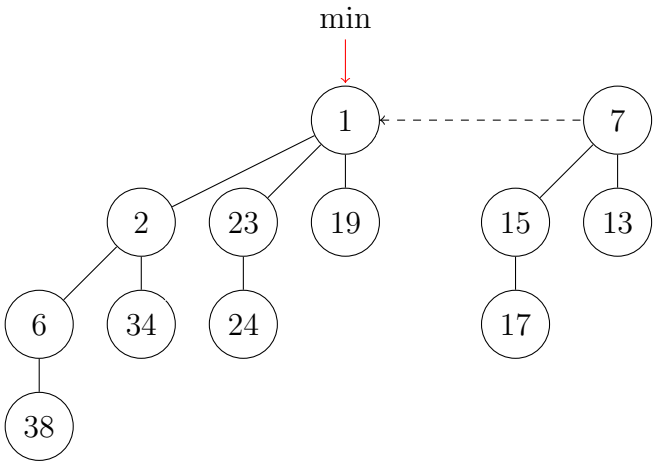


(c) Gegeben seien drei Binomial-Heaps mit 493, 275 bzw. 294 Elementen. Geben Sie an, welche Ränge die in den Heaps enthaltenen Binomialbäume besitzen. Durch zwei Merge-Operationen können die drei Binomial-Heaps zu einem neuen Binomial-Heap vereinigt werden. Berechnen Sie auch für diesen Binomial-Heap die Ränge der im Heap enthaltenen Binomialbäume.

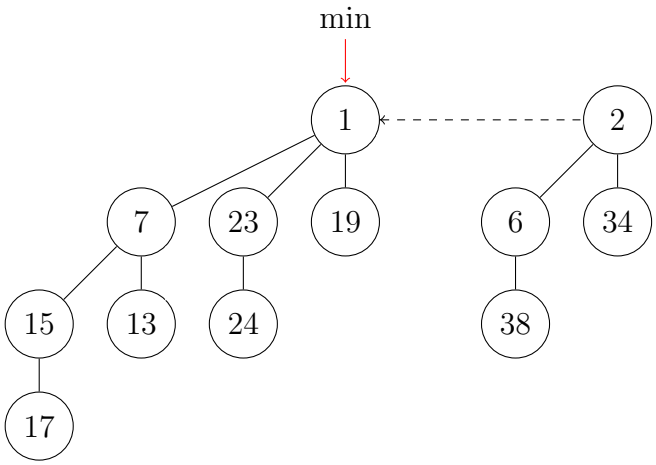
Lösungsvorschlag

(a) Die folgenden drei Binomial-Heaps sind möglich.

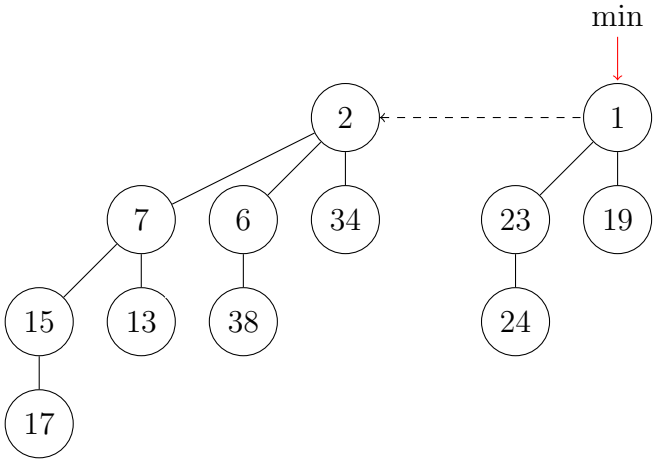
(i)



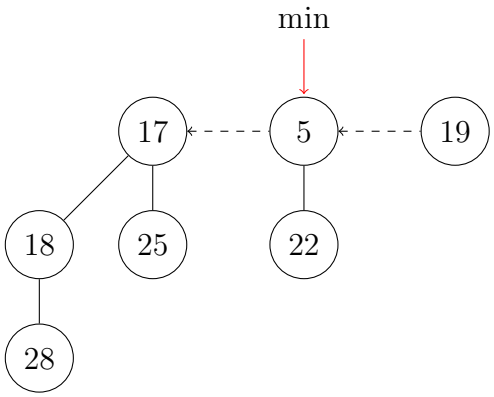
(ii)



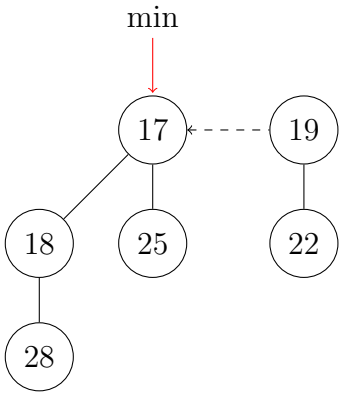
(iii)



(b) Nach der ersten DeleteMin-Operation hat der Binomial-Heap die folgende Gestalt:



Nach der zweiten DeleteMin-Operation erhalten wir:



(c) Die Ränge lassen sich an der Binärdarstellung der Zahl der enthaltenen Elemente ablesen. Daher hat der erste Heap mit $493 = 2^8 + 2^7 + 2^6 + 2^5 + 2^3 + 2^2 + 2^0$ Elementen die Ränge $8, 7, 6, 5, 3, 2, 0$, der zweite Heap die Ränge $8, 4, 1, 0$ und der dritte Heap die Ränge $8, 5, 2, 1$. Der durch die Merge-Operationen entstehende Heap hat 1062 Elemente und daher die Ränge $10, 5, 2, 1$.

Aufgabe 7 (10 Punkte)

Zeigen Sie, dass die Worst-Case-Laufzeit für das Einfügen von n Elementen in einen anfangs leeren **Binär-Heap** in $\Omega(n \log n)$ liegt. Verwenden Sie in Ihrem Beweis eine in geeigneter Weise in Abhängigkeit von n definierte Folge von einzufügenden Elementen.

Lösungsvorschlag

Wir betrachten für alle $n \geq 1$ die absteigend sortierte Folgen $n, n-1, \ldots, 2, 1$. Fügen wir die Elemente in einer solchen Reihenfolge ein, dann müssen wir in jedem Schritt das neue Element bis zur Wurzel aufsteigen lassen. Die Laufzeit für das Aufsteigen ist damit mindestens so groß wie $t-1$, wobei t die Baumtiefe des aktuellen Heaps ist. Befinden sich i Elemente im Heap, dann ist die Baumtiefe des Binär-Heaps $\lceil \mathrm{ld}(i+1) \rceil - 1$. Damit gilt für die Laufzeit

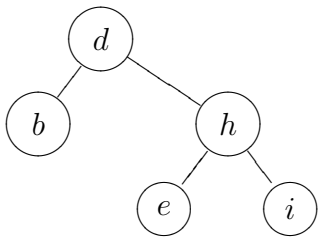
$$\begin{aligned} T(n) &\geq \sum_{i=0}^{n-1} (\lceil \mathrm{ld}(i+1) \rceil - 1) \geq \sum_{i=0}^{n-1} (\mathrm{ld}(i+1) - 1) = \sum_{i=1}^{n} (\mathrm{ld}(i) - 1) = -n + \sum_{i=1}^{n} \mathrm{ld}(i) \\ &= -n + \mathrm{ld}(n!) \geq -n + \mathrm{ld}(n^{n/2}) = -n + \frac{n}{2} \mathrm{ld}(n) \in \Omega(n \log n) \, . \end{aligned}$$

Falls man die Formel $\mathrm{ld}(n!) \geq \mathrm{ld}(n^{n/2})$ oder eine artverwandte Formel nicht kennt, kann man auch wie folgt argumentieren: Sei $n \geq n_0 := 4$ und sei m die größte Zweierpotenz, die kleiner als $n/2$ ist. Nach Einfügung der ersten m Elemente ist der Binär-Heap ein vollständiger Binärbaum mit m Blättern und Baumtiefe $\mathrm{ld}(m) \geq \mathrm{ld}(n/4) = \mathrm{ld}(n) - 2$. Das bedeutet, dass jedes der restlichen mindestens $n/2$ Elemente für das Aufsteigen eine Laufzeit von mindestens $\mathrm{ld}(n) - 2$ hat. Die Gesamtlaufzeit ist also mindestens $n/2 \cdot (\mathrm{ld}(n) - 2) \in \Omega(n \log n)$.

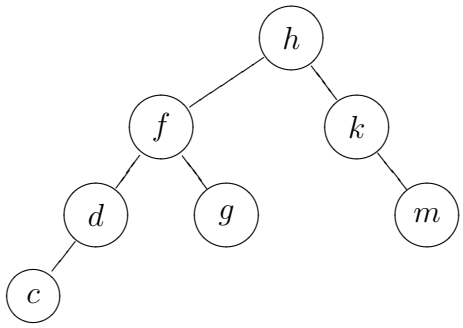
Aufgabe 8 (8 Punkte)

In dieser Aufgabe sind die Schlüssel lexikographisch geordnet. Zeichnen Sie nach **jeder** Operation den durch die Operation entstandenen AVL-Baum, und schreiben Sie dazu, ob keine Rotation, ob eine Rotation oder ob eine Doppelrotation durchgeführt wurde.

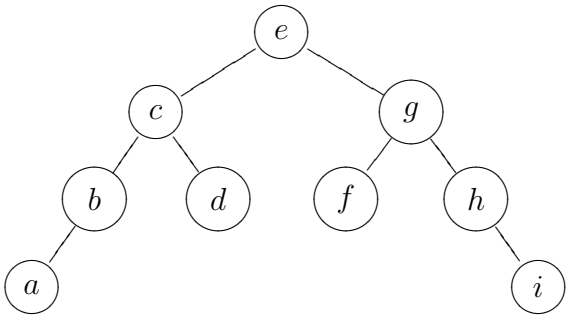
(a) Führen Sie auf dem folgenden AVL-Baum eine *insert*-Operation des Schlüssels f durch.



(b) Führen Sie auf dem folgenden AVL-Baum eine *insert*-Operation des Schlüssels a durch. Führen Sie dann auf dem dadurch entstandenen AVL-Baum eine *insert*-Operation des Schlüssels i durch.

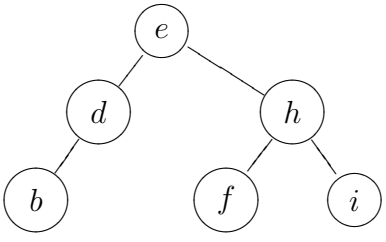


(c) Führen Sie auf dem folgenden AVL-Baum eine *delete*-Operation des Schlüssels e durch.

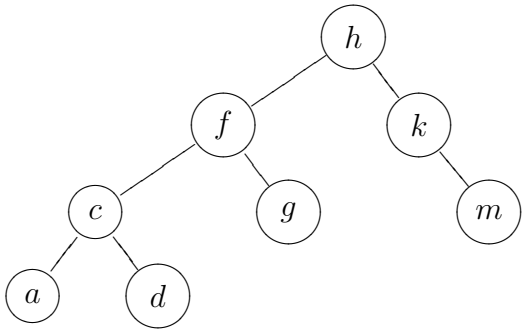


Lösungsvorschlag

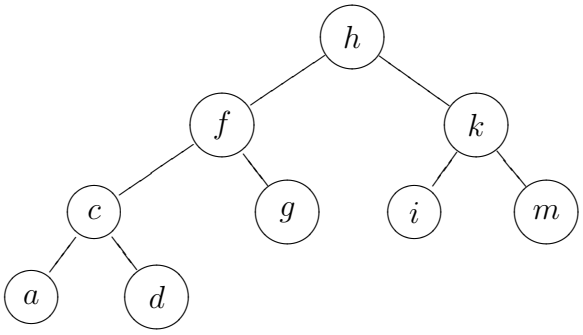
(a) f wird mit Doppelrotation eingefügt.



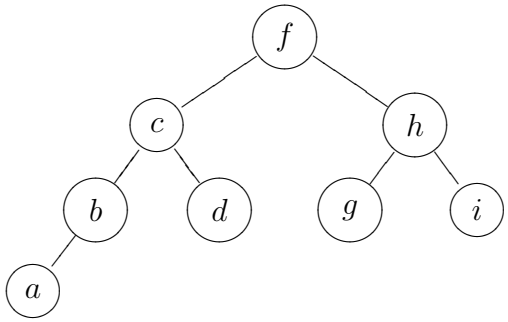
(b) a wird mit Einfachrotation eingefügt.

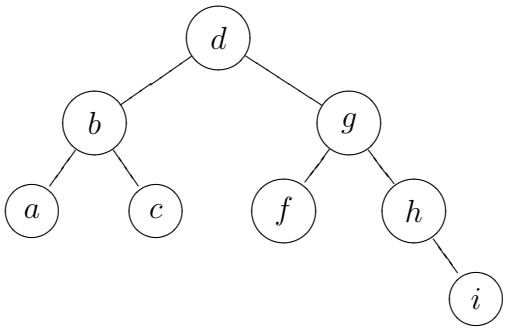


i wird ohne Rotation eingefügt.



(c) e wird mit Einfachrotation gelöscht. Je nachdem, welche Variante man wählt, erhält man einen der folgenden Bäume:

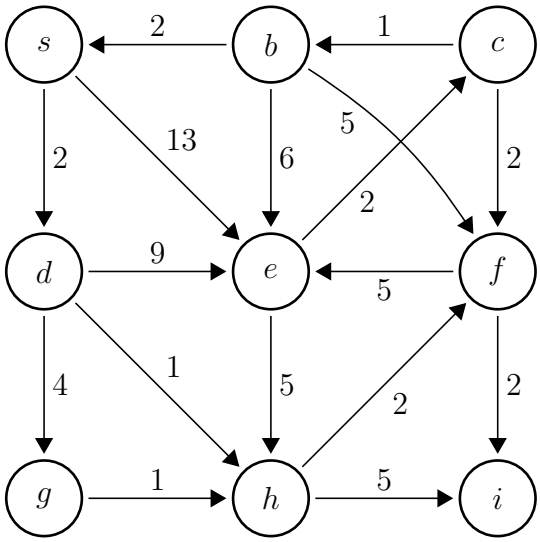




Aufgabe 9 (9 Punkte)

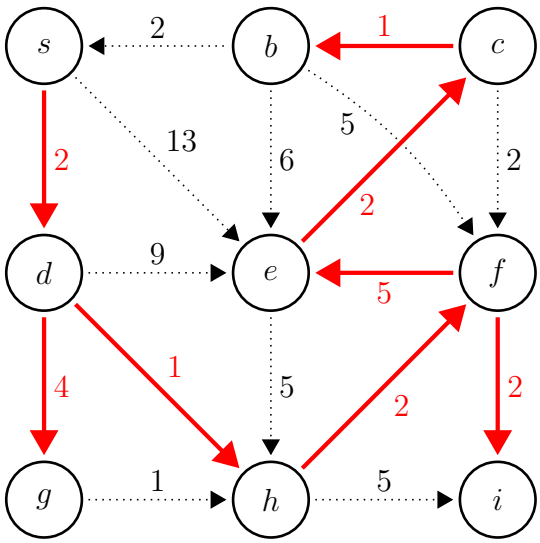
Führen Sie den Algorithmus von Dijkstra auf dem folgenden Graphen beginnend bei Knoten s durch. Füllen Sie dafür die Zeilen der Schritte 2. bis 9. der Berechnungstabelle aus. Markieren Sie zum Schluss alle Kanten, die zum berechneten Kürzeste-Wege-Baum gehören.

Anmerkung: Die Spalte *Inhalt der Priority-Queue* enthält Paare bestehend aus Schlüssel und zugehöriger Priorität. $(d, 2)$ bedeutet also, dass Schlüssel d mit Priorität 2 in der Queue ist.



Schritt	Aktueller Knoten	pred bzw. parent von b	c	d	e	f	g	h	i	Inhalt der Priority-Queue
1.	s			s	s					$(d, 2),\ (e, 13)$
2.										
3.										
4.										
5.										
6.										
7.										
8.										
9.										

Lösungsvorschlag



Schritt	Aktueller Knoten	pred bzw. parent von b	c	d	e	f	g	h	i	Inhalt der Priority-Queue
1.	s			s	s					$(d,2),\ (e,13)$
2.	d			s	d		d	d		$(h,3),\ (g,6),\ (e,11)$
3.	h			s	d	h	d	d	h	$(f,5),\ (g,6),\ (i,8),\ (e,11)$
4.	f			s	f	h	d	d	f	$(g,6),\ (i,7),\ (e,10)$
5.	g			s	f	h	d	d	f	$(i,7),\ (e,10)$
6.	i			s	f	h	d	d	f	$(e,10)$
7.	e		e	s	f	h	d	d	f	$(c,12)$
8.	c	c	e	s	f	h	d	d	f	$(b,13)$
9.	b	c	e	s	f	h	d	d	f	

Bonusaufgabe (5 Punkte)

Ein Binomial-Heap für n Elemente kann dadurch aufgebaut werden, dass wir mit einem einelementigen Binomial-Heap beginnen, und dann die restlichen $n-1$ Elemente durch insert-Operationen in den Heap einfügen.

Zeigen Sie mithilfe einer amortisierten Analyse, dass die Worst-Case-Laufzeit für die so definierte build-Funktion in $\mathcal{O}(n)$ liegt.

Anmerkung: Wie üblich nehmen wir an, dass der Vergleich von zwei Elementen nur konstant viel Zeit benötigt. Weiterhin nehmen wir an, dass jedes mal, wenn durch die Vereinigung zweier Binomialbäume ein neuer Binomialbaum entsteht, sofort der Minimumszeiger mithilfe eines Vergleichs des alten Minimums mit der Wurzel des neuen Baumes aktualisiert wird.

Lösungsvorschlag

Wird ein neues Element hinzugefügt, entstehen für jede Merge-Operation zweier Binomialbäume konstante Kosten: Es muss eine neue Referenz vom Baum mit der kleineren Wurzel zum Baum mit der größeren Wurzel gesetzt werden, sowie einige Referenzen der Wurzelliste müssen aktualisiert werden, was konstante Kosten verursacht. Zudem muss das Minimum aktualisiert werden. Dazu genügt es, die Wurzel des neuen Baumes zu betrachten und mit der alten Minimumswurzel zu vergleichen. Je nachdem, welche Wurzel kleiner ist, bleibt der Minimumszeiger gleich oder zeigt auf die Wurzel des neuen Baumes. Beim Spezialfall, dass der Baum mit der Minimumswurzel Teil des neuen Baumes ist, zeigt der Minimumszeiger stets auf die Wurzel des neuen Baumes. Diese Vergleiche und das Setzen der Referenz verursachen ebenfalls nur konstante Kosten. Es kann noch weiterer Overhead wie zum Beispiel durch das Ablaufen der Wurzelelemente durch iterative Merge-Operationen bzw. das Vergleichen von Wurzelelementen dazukommen. Diese Kosten und damit die Gesamtkosten sind aber linear in der Zahl der Vereinigungen der Binomialbäume. Zusätzlich entsteht noch konstanter Overhead.

Wir führen nun eine amortisierte Analyse mithilfe der Bankkontomethode durch. Wir haben im letzten Absatz argumentiert, dass die tatsächlichen Kosten einer insert-Operation maximal $mc+d$ sind, wobei $c, d > 0$ positive Konstanten sind und m die Zahl der Vereinigungen ist. Jedes Mal, wenn ein Binomialbaum mit einem Grad entsteht, sodass dieser Grad vor der insert-Operation noch nicht da war, werden c Token auf das Konto eingezahlt. Jede Vereinigungsoperation zweier Binomialbäume hebt c Token vom Konto ab.

Wir beobachten, dass bei jeder Insert-Operation genau einmal ein solcher Binomialbaum mit neuem Grad entsteht. Daher betragen die amortisierten Kosten maximal $(mc+d)+(c-mc) = c+d$. Wir müssen nun noch zeigen, dass das Tokenkonto stets nichtnegativ ist. Dazu genügt die Beobachtung, dass jede Vereinigung zweier Binomialbäume mit Grad g nur genau diejenigen c Token vom Konto abhebt, die bei der Entstehung des ersten Binomialbaumes mit Grad g auf das Konto eingezahlt wurden. Die Gesamtkosten von n konsekutiven insert-Operationen und damit die Kosten der build-Operation sind also maximal $n(c+d) \in \mathcal{O}(n)$.