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

Sommersemester 2014
Lösungen der Klausur
19. Juli 2014

Grundlagen: Algorithmen und Datenstrukturen

Name	Vorname	Studiengang	Matrikelnummer
.....	.....	☐ Diplom ☐ Inform. ☐ Bachelor ☐ BioInf. ☐ Master ☐ WirInf. ☐ Lehramt ☐ 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	Σ	Korrektor
Gesamt	8	12	10	8	8	10	8	8	8	80	
Erstkorrektur											
Zweitkorrektur											

Aufgabe 1 (8 Punkte)

Kreuzen Sie für die folgenden Fragen alle korrekten Antworten an. Für jede Frage sind zwischen null und alle Antworten korrekt. Für jede Frage gibt es genau dann einen Punkt, wenn alle richtigen Antworten angekreuzt und alle falschen Antworten nicht angekreuzt sind, ansonsten gibt es für die Frage null Punkte.

(a) In welchen Wachstumsordnungen liegt die erwartete Laufzeit des QuickSelect-Algorithmus? ☐ $\mathcal{O}(n)$ ☐ $\Theta(n \log n)$ ☐ $\Theta(n^2)$

(b) In welchen Wachstumsordnungen liegt die Worst-Case-Laufzeit einer Folge von m Operationen auf einem zu Beginn leeren dynamischen Array (definiert wie in der Vorlesung)? ☐ $\mathcal{O}(m)$ ☐ $\Omega(m)$ ☐ $\Theta(m)$ ☐ $o(m)$ ☐ $\omega(m)$

(c) Welche der folgenden Voraussetzungen sind jeweils *hinreichend*, um das kleinste von n Elementen in Zeit $\mathcal{O}(\log(n))$ finden zu können?

☐ Die Elemente sind in einer doppelt verketteten unsortierten Liste enthalten.
☐ Die Elemente sind in einem AVL-Suchbaum enthalten.
☐ Die Elemente sind in einem Binomial-Min-Heap enthalten.

(d) Welche der folgenden Voraussetzungen sind jeweils *hinreichend*, damit der Dijkstra-Algorithmus das korrekte Resultat liefert?

☐ Alle Kantengewichte des Eingabegraphen sind nichtnegativ.
☐ Der Eingabegraph enthält keinen negativen Kreis.
☐ Der Eingabegraph ist ein DAG.

(e) Welche der folgenden Voraussetzungen sind jeweils *hinreichend*, damit die Find-Operation in einem binären Suchbaum mit n Elementen in Zeit $\mathcal{O}(\log n)$ durchgeführt werden kann?

☐ Die Tiefe des Baums ist maximal $15 \log n$.
☐ Für jeden inneren Knoten unterscheiden sich die Tiefen des linken und rechten Teilbaums um maximal 1.
☐ Der Baum hat $b - 1$ innere Knoten, wobei b die Zahl seiner Blätter ist.

(f) Welche Vorteile hat statisches perfektes Hashing gegenüber Hashing mit Chaining, wenn die Eingabe aus $n \in \mathbb{N}$ gleichverteilten Eingabeschlüsseln besteht?

☐ Die Worst-Case-Laufzeit für die Konstruktion der Hash-Tabelle ist linear in n.
☐ Die Worst-Case-Laufzeit für die Find-Operation ist konstant.
☐ Man kann nach Konstruktion der Hash-Tabelle für die n gegebenen Elemente zusätzliche Elemente in erwartet konstanter Zeit einfügen.

(g) Welche der folgenden Eigenschaften hat die Adjazenzmatrix als Graph-Repräsentation für einen Graphen mit n Knoten und m Kanten?

☐ Der Speicherbedarf ist $\mathcal{O}(n + m)$.
☐ Kanten können in Zeit $\mathcal{O}(1)$ gelöscht und eingefügt werden.
☐ Alle Nachbarn eines Knotens können in Zeit $\mathcal{O}(d)$ gefunden werden, wobei d die Zahl der Nachbarn ist.

(h) Welche der folgenden Algorithmen lösen das All-Pairs-Shortest-Path-Problem? ☐ Dijkstra-Alg. ☐ Bellman-Ford-Alg. ☐ Johnson-Alg.

Lösungsvorschlag

(a) In welchen Wachstumsordnungen liegt die erwartete Laufzeit des QuickSelect-Algorithmus? ☒ $\mathcal{O}(n)$ ☐ $\Theta(n \log n)$ ☐ $\Theta(n^2)$

(b) In welchen Wachstumsordnungen liegt die Worst-Case-Laufzeit einer Folge von m Operationen auf einem zu Beginn leeren dynamischen Array (definiert wie in der Vorlesung)? ☒ $\mathcal{O}(m)$ ☒ $\Omega(m)$ ☒ $\Theta(m)$ ☐ $o(m)$ ☐ $\omega(m)$

(c) Welche der folgenden Voraussetzungen sind jeweils *hinreichend*, um das kleinste von n Elementen in Zeit $\mathcal{O}(\log(n))$ finden zu können?

☐ Die Elemente sind in einer doppelt verketteten unsortierten Liste enthalten.
☒ Die Elemente sind in einem AVL-Suchbaum enthalten.
☒ Die Elemente sind in einem Binomial-Min-Heap enthalten.

(d) Welche der folgenden Voraussetzungen sind jeweils *hinreichend*, damit der Dijkstra-Algorithmus das korrekte Resultat liefert?

☒ Alle Kantengewichte des Eingabegraphen sind nichtnegativ.
☐ Der Eingabegraph enthält keinen negativen Kreis.
☐ Der Eingabegraph ist ein DAG.

(e) Welche der folgenden Voraussetzungen sind jeweils *hinreichend*, damit die Find-Operation in einem binären Suchbaum mit n Elementen in Zeit $\mathcal{O}(\log n)$ durchgeführt werden kann?

☒ Die Tiefe des Baums ist maximal $15 \log n$.
☒ Für jeden inneren Knoten unterscheiden sich die Tiefen des linken und rechten Teilbaums um maximal 1.
☐ Der Baum hat $b - 1$ innere Knoten, wobei b die Zahl seiner Blätter ist.

(f) Welche Vorteile hat statisches perfektes Hashing gegenüber Hashing mit Chaining, wenn die Eingabe aus $n \in \mathbb{N}$ gleichverteilten Eingabeschlüsseln besteht?

☐ Die Worst-Case-Laufzeit für die Konstruktion der Hash-Tabelle ist linear in n.
☒ Die Worst-Case-Laufzeit für die Find-Operation ist konstant.
☐ Man kann nach Konstruktion der Hash-Tabelle für die n gegebenen Elemente zusätzliche Elemente in erwartet konstanter Zeit einfügen.

(g) Welche der folgenden Eigenschaften hat die Adjazenzmatrix als Graph-Repräsentation für einen Graphen mit n Knoten und m Kanten?

☐ Der Speicherbedarf ist $\mathcal{O}(n + m)$.
☒ Kanten können in Zeit $\mathcal{O}(1)$ gelöscht und eingefügt werden.
☐ Alle Nachbarn eines Knotens können in Zeit $\mathcal{O}(d)$ gefunden werden, wobei d die Zahl der Nachbarn ist.

(h) Welche der folgenden Algorithmen lösen das All-Pairs-Shortest-Path-Problem?
☐ Dijkstra-Alg. ☐ Bellman-Ford-Alg. ☒ Johnson-Alg.

Aufgabe 2 (12 Punkte)

Kreuzen Sie in den Zeilen (a) bis (f) jeweils das zu den Funktionen stärkste passende Symbol an. Das heißt, wenn $\Delta = o$ (bzw. $\Delta = \Theta$) möglich ist, wählen Sie $\Delta = o$ (bzw. $\Delta = \Theta$) und nicht $\Delta = \mathcal{O}$. Falls die Funktionen unvergleichbar sind, kreuzen Sie „u.“ an.

Bsp.:	n	$\in \Delta(n^2)$	☒ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.
(a)	$7n^3$	$\in \Delta(3n^7)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.
(b)	$\sqrt[3]{\sqrt{3n}}$	$\in \Delta\left(\sqrt{\sqrt[3]{2n}}\right)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.
(c)	$n!$	$\in \Delta(4^n)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.
(d)	n	$\in \Delta((2+(-1)^n)n)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.
(e)	$\sqrt{n}$	$\in \Delta(\ln n)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.
(f)	$(0.5n)^{\operatorname{ld} n}$	$\in \Delta\left(2^{(\operatorname{ld} n)^2}\right)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.

Begründen Sie Ihre Antworten jeweils mit einem kurzen Beweis.

Jede korrekt angekreuzte Zeile gibt einen Punkt. Ist das angekreuzte Symbol prinzipiell richtig, aber nicht das stärkste passende Symbol, gibt es einen halben Punkt. Jeder korrekte kurze Beweis für das von Ihnen angekreuzte Symbol gibt *zusätzlich* einen Punkt. Sie müssen allerdings nicht beweisen, dass das von Ihnen angekreuzte Symbol das stärkst mögliche ist. Eine falsch angekreuzte Zeile oder ein falscher Beweis gibt *keine* negativen Punkte.

Lösungsvorschlag

(a)	$7n^3$	$\in \Delta(3n^7)$	☒ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.
(b)	$\sqrt[3]{\sqrt{3n}}$	$\in \Delta\left(\sqrt{\sqrt[3]{2n}}\right)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☒ Θ	☐ u.
(c)	$n!$	$\in \Delta(4^n)$	☐ o	☐ $\mathcal{O}$	☒ ω	☐ Ω	☐ Θ	☐ u.
(d)	n	$\in \Delta((2+(-1)^n)n)$	☐ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☒ Θ	☐ u.
(e)	$\sqrt{n}$	$\in \Delta(\ln n)$	☐ o	☐ $\mathcal{O}$	☒ ω	☐ Ω	☐ Θ	☐ u.
(f)	$(0.5n)^{\operatorname{ld} n}$	$\in \Delta\left(2^{(\operatorname{ld} n)^2}\right)$	☒ o	☐ $\mathcal{O}$	☐ ω	☐ Ω	☐ Θ	☐ u.

(a) $\lim\limits_{n\to\infty} \dfrac{7n^3}{3n^7} = \lim\limits_{n\to\infty} \dfrac{7}{3n^4} = 0$

(b) $\lim\limits_{n\to\infty} \dfrac{\sqrt[3]{\sqrt{3n}}}{\sqrt{\sqrt[3]{2n}}} = \lim\limits_{n\to\infty} \dfrac{\sqrt[6]{3}\cdot\sqrt[6]{n}}{\sqrt[6]{2}\cdot\sqrt[6]{n}} = \sqrt[6]{\dfrac{3}{2}}$

(c) Es ist bekannt, dass $n! \geq n^{n/2} = \sqrt{n}^n$ gilt, woraus $n! \in \Omega(n^{n/2})$ folgt. Aus $\lim_{n\to\infty} \frac{\sqrt{n}^n}{4^n} = \lim_{n\to\infty} \left(\frac{\sqrt{n}}{4}\right)^n \geq= \lim_{n\to\infty} \left(\frac{8}{4}\right)^n = \infty$ erhalten wir $n^{n/2} \in \omega(4^n)$. Aus den Transitivitätsregeln folgt damit $n! \in \omega(4^n)$.

(d) Für alle n gilt $n \leq (2 + (-1)^n)n$, woraus $n \in \mathcal{O}((2 + (-1)^n)n)$ folgt. Für alle n gilt außerdem $n \geq \frac{1}{2} \cdot (2 + (-1)^n)n$, woraus sich $n \in \Omega((2 + (-1)^n)n)$ ergibt. Insgesamt erhalten wir also $n \in \Theta((2 + (-1)^n)n)$.

(e) $\lim_{n\to\infty} \frac{\sqrt{n}}{\ln n} = \lim_{n\to\infty} \frac{0.5n^{-0.5}}{1/n} = \lim_{n\to\infty} \frac{0.5n}{\sqrt{n}} = \lim_{n\to\infty} 0.5\sqrt{n} = \infty$

(f) $\lim_{n\to\infty} \frac{(0.5n)^{\operatorname{ld} n}}{2^{(\operatorname{ld} n)^2}} = \lim_{n\to\infty} \frac{(0.5n)^{\operatorname{ld} n}}{n^{\operatorname{ld} n}} = \lim_{n\to\infty} 0.5^{\operatorname{ld} n} = \lim_{n\to\infty} \frac{1}{n} = 0$

Aufgabe 3 (10 Punkte)

(a) Nennen Sie zwei in der Vorlesung behandelte vergleichsbasierte Sortieralgorithmen und nennen Sie die jeweiligen asymptotisch optimalen oberen Schranken für deren Worst-Case-Laufzeiten.

(b) Erklären Sie kurz, wie man einen binären Heap dafür verwenden kann, um eine gegebene Zahlenfolge in aufsteigender Reihenfolge zu sortieren.

(c) Sortieren Sie die Zahlenfolge $741, 147, 17, 57, 45, 55, 5, 141, 145, 11$ in aufsteigender Reihenfolge mit RadixSort. Geben Sie für jede Sortierphase den Inhalt der Buckets sowie die durch die Sortierphase entstehende Zahlenfolge an.

Lösungsvorschlag

(a) Zum Beispiel MergeSort mit $\mathcal{O}(n \log n)$ und QuickSort mit $\mathcal{O}(n^2)$.

(b) Man fügt mit der `insert`-Operation iterativ alle Zahlen in den Heap ein und verwendet anschließend iterativ die `deleteMin`-Operation, um die Zahlen in aufsteigender Reihenfolge zu erhalten.

(c) Wir sortieren nach der letzten Ziffer.

0	1	2	3	4	5	6	7	8	9
	741				45		147		
	141				55		17		
	11				5		57		
					145				

Es ergibt sich das folgende Zwischenresultat: $741, 141, 11, 45, 55, 5, 145, 147, 17, 57$

Wir sortieren nach der vorletzten Ziffer.

0	1	2	3	4	5	6	7	8	9
5	11			741	55				
	17			141	57				
				45					
				145					
				147					

Es ergibt sich das folgende Zwischenresultat: $5, 11, 17, 741, 141, 45, 145, 147, 55, 57$

Wir sortieren nach der drittletzten Ziffer.

0	1	2	3	4	5	6	7	8	9
5	141						741		
11	145								
17	147								
45									
55									
57									

Wir erhalten daraus das folgende Endresultat: $5, 11, 17, 45, 55, 57, 141, 145, 147, 741$

Aufgabe 4 (8 Punkte)

Wir betrachten eine Folge von $m \geq 1$ Inkrement-Operationen iterativ angewandt auf 0, das heißt, dass die k-te Inkrementoperation σ_k die Zahl $k-1$ zur Zahl k inkrementiert. Wir nehmen an, dass alle Zahlen in *Dezimalschreibweise* geschrieben werden, also mit den Ziffern $0, 1, \ldots, 9$. Außerdem nehmen wir (vereinfachend) an, dass die Änderung einer Stelle (von x zu $x+1$ für alle $x \in \{0, \ldots, 8\}$ bzw. von 9 zu 0) Laufzeit 1 hat und die Laufzeit jeder Inkrement-Operation gerade die Anzahl der Stellen ist, die durch die Operation geändert werden.

(a) Definieren Sie ein zulässiges Amortisationsschema $\Delta : \{\sigma_1, \sigma_2, \ldots\} \to \mathbb{R}$ der Bankkonto-Methode, sodass für jede Inkrementoperation σ_k die amortisierte Laufzeit $A(\sigma_k) = T(\sigma_k) + \Delta(\sigma_k)$ gerade $\frac{10}{9}$ beträgt (hierbei sei $T(\sigma_k)$ die tatsächliche Laufzeit von σ_k).

Sie erhalten einen Teil der Punkte, wenn Sie eine amortisierte Laufzeit herleiten, die zwar mehr als $\frac{10}{9}$ beträgt, aber zumindest konstant ist.

(b) Zeigen Sie, dass Ihr Amortisationsschema zulässig ist, indem Sie nachweisen, dass das Tokenkonto stets nichtnegativ ist.

Hinweis: Verwenden Sie in Ihrem Amortisationsschema für jede Operation σ_k die Anzahl $S_{x \to x+1}(\sigma_k)$ der Stellen, die durch σ_k von x zu $x+1$ geändert werden, sowie die Anzahl $S_{9 \to 0}(\sigma_k)$ der Stellen, die durch σ_k von 9 zu 0 geändert werden.

Lösungsvorschlag

(a) Wird eine Zahl x inkrementiert, dann müssen die letzten Stellen der Binärdarstellung solange geändert werden, bis eine Ziffer x, $x \in \{0, 1, \ldots, 8\}$ in eine Ziffer $x+1$ geändert wurde. Die Kosten hiefür betragen gerade die Anzahl der Änderungen von 9 zu 0 zuzüglich einer Änderung von x zu $x+1$. Wenn also jede Änderung von x zu $x+1$ gerade $\frac{1}{9}$ Token auf das Konto einbezahlt, kann von diesem Konto die nächste später auftretende Änderung derselben Stelle von 9 zu 0 bezahlt werden.

Wir definieren also

$$\Delta(\sigma_k) := \frac{S_{x \to x+1}(\sigma_k)}{9} - S_{9 \to 0}(\sigma_k) = \frac{1}{9} - S_{9 \to 0}(\sigma_k).$$

Die tatsächliche Laufzeit von σ_k beträgt nach Angabe

$$T(\sigma_k) = S_{x \to x+1}(\sigma_k) + S_{9 \to 0}(\sigma_k) = 1 + S_{9 \to 0}(\sigma_k).$$

Die amortisierte Laufzeit beträgt daher

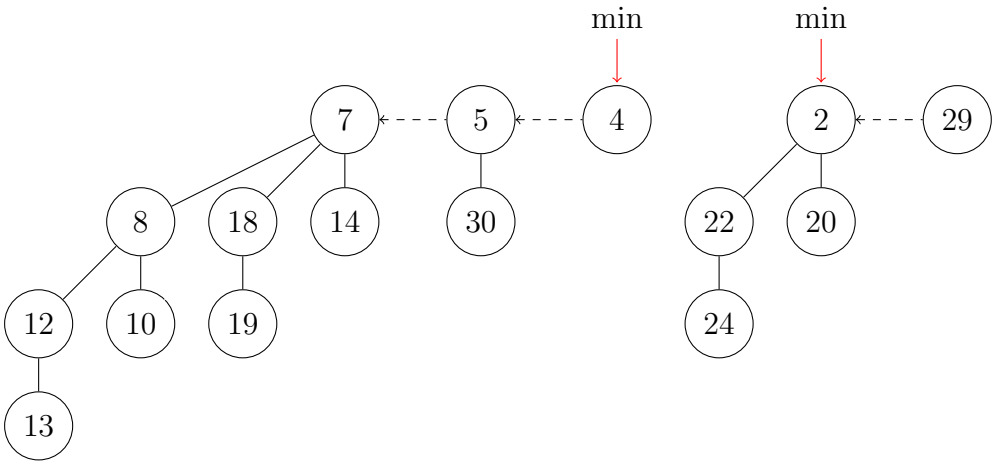
$$A(\sigma_k) = T(\sigma_k) + \Delta(\sigma_k) = (1 + S_{9 \to 0}(\sigma_k)) + \left(\frac{1}{9} - S_{9 \to 0}(\sigma_k)\right) = \frac{10}{9}.$$

(b) Dieses Schema ist zulässig, weil der Betrag auf dem Konto stets $\frac{1}{9}$ mal die Summe aller Ziffern in der Dezimaldarstellung der aktuellen Zahl beträgt, und damit nie negativ ist.

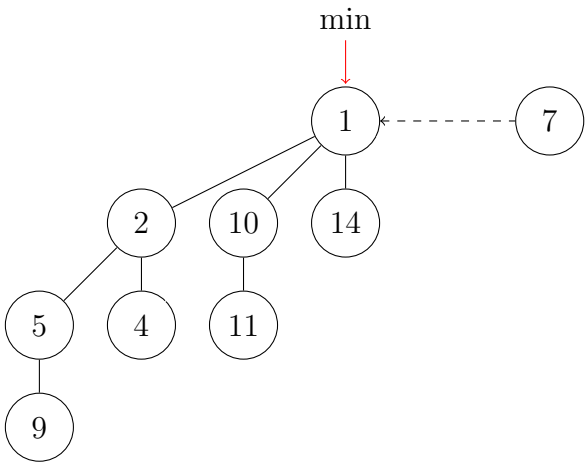
Wir bemerken, dass daher die Laufzeit einer Folge von Inkrementoperationen der Länge m maximal $\frac{10}{9}m$ beträgt.

Aufgabe 5 (8 Punkte)

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



(b) Führen Sie eine `deleteMin`-Operationen auf dem folgenden Binomial-Heap durch. Zeichnen Sie den resultierenden Binomial-Heap.



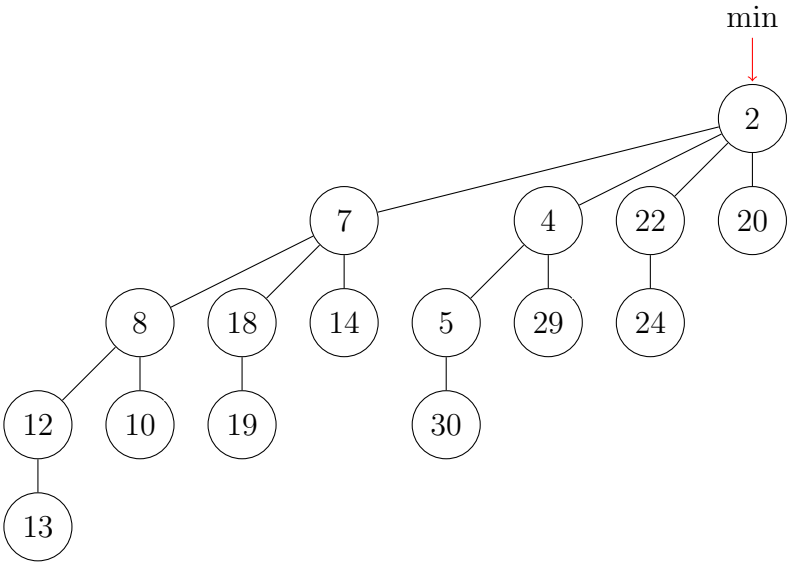
(c) Geben Sie für die Operationen `merge`, `insert`, `min` und `deleteMin` von Binomial-Heaps jeweils asymptotisch optimale obere Laufzeitschranken an.

(d) Gegeben sei ein Binomial-Heap, der aus Binomialbäumen mit den Rängen $0, 1, 2, 3, 4, 5, 6, 7$ besteht. Auf diesem Heap werden nacheinander 162 `deleteMin`-Operationen ausgeführt. Berechnen Sie die Ränge der Bäume im resultierenden Binomial-Heap B_1.

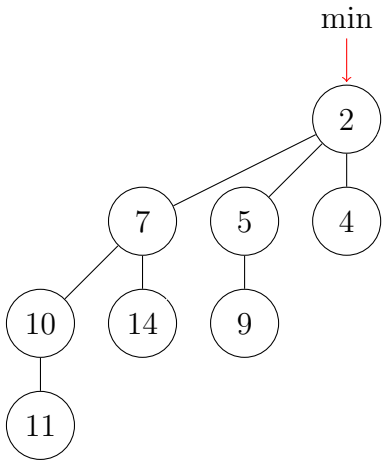
Nun wird B_1 mit insgesamt 127 weiteren Binomial-Heaps $B_2, \ldots, B_{128}$ vereinigt, die jeweils genauso viele Elemente wie B_1 besitzt. Berechnen Sie auch für den daraus resultierenden Binomial-Heap die Ränge der enthaltenen Binomial-Bäume.

Lösungsvorschlag

(a)



(b)



(c) `merge`: $\mathcal{O}(\log n)$, `insert`: $\mathcal{O}(\log n)$, `min`: $\mathcal{O}(1)$, `deleteMin`: $\mathcal{O}(\log n)$

(d) Die Elementzahl im ursprünglichen Binomial-Heap ist 255. Der entstehende Binomial-Heap enthält damit $93_{10} = 1011101_2$ Elemente. Die Ränge der enthaltenen Binomialbäume sind also 0,2,3,4,6. Wenn wir nun diesen Baum mit 127 Bäumen mergen, die alle ebenfalls 93 Elemente besitzen, besitzt der entstehende Binomial-Heap genau $93 \cdot 2^7 = 93 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2$ Elemente. Da die Multiplikation einer Binärzahl gerade einem Bitshift um eine Position nach links entspricht, ist die Bitdarstellung von $93 \cdot 2^7$ gerade 10111010000000_2, d.h. der resultierende Binomial-Heap enthält Binomial-Bäume mit den Rängen $7, 9, 10, 11, 13$.

Aufgabe 6 (10 Punkte)

(a) Ein ungerichteter Graph $G = (V, E)$ mit Knotenmenge V und Kantenmenge E ist bipartit, wenn es zwei Knotenmengen $V_1, V_2 \subseteq V$ mit $V_1 \cap V_2 = \emptyset$ und $V_1 \cup V_2 = V$ gibt, sodass jede Kante des Graphen einen Knoten aus V_1 mit einem Knoten aus V_2 verbindet (d.h., keine Kante verläuft innerhalb von V_1 oder innerhalb von V_2). Zeigen Sie mit (starker) vollständiger Induktion, dass jeder Baum bipartit ist.

(b) Gegeben ist die folgende Rekursionsgleichung:

$$f(n) = 2f\left(\frac{n}{3}\right) + 21$$
$$f(1) = 4$$

Zeigen Sie mit vollständiger Induktion, dass es ein $c > 0$ gibt, sodass für alle Dreierpotenzen $n = 3^k > 0$ mit $k \in \mathbb{N}_0$ die Ungleichung $f(n) \leq cn$ erfüllt ist. Verwenden Sie nicht das Master-Theorem.

Lösungsvorschlag

(a) Für den Baum, der $n = 1$ Knoten hat, also nur aus der Wurzel w besteht, ist die Aussage klar, da $V_1 = \{w\}$, $V_2 = \emptyset$ eine Möglichkeit ist, die Knotenmengen bipartit zu partitionieren. (Wenn man möchte, kann man auch den leeren Baum als Induktionsanfang verwenden.)

Sei $n > 1$. Unsere Induktionsannahme ist, dass die Aussage für alle gewurzelten Bäume mit $n-1$ Knoten gilt. Betrachte einen beliebigen gewurzelten Baum T mit n Knoten. Dieser muss ein Blatt b mit Vaterknoten v haben. Durch Entfernen von b erhalten wir einen Baum T' mit $n-1$ Knoten. Nach Induktionsannahme ist T' bipartit, d.h., es gibt Mengen V_1' und V_2' mit den entsprechenden Eigenschaften. Sei o.B.d.A. $v \in V_1'$. Wir definieren $V_1 := V_1'$ und $V_2 := V_2' \cup \{b\}$. Alle Kanten aus T' verlaufen zwischen V_1 und V_2. Ebenso verläuft die Kante von v nach b zwischen V_1 und V_2. Daher ist T bipartit.

Für ungewurzelte Bäume kann der Beweis natürlich analog geführt werden. Alternativ kann man einen ungewurzelten Baum auch vorher einfach wurzeln und dann direkt obigen Beweis verwenden. (Wir erlauben hierbei die Annahme, dass jeder Baum gewurzelt werden kann.)

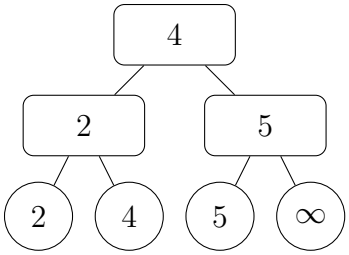
(b) Wir wählen $c = 21$. Für $n = 1$ gilt $f(n) = f(1) = 4 \leq 21 \cdot 1 = 21n$. Sei $n \geq 3$ eine Dreierpotenz. Angenommen, die Aussage gilt für die Dreierpotenz $\frac{n}{3}$. Dann finden wir

$$f(n) = 2f\left(\frac{n}{3}\right) + 21 \leq 2 \cdot 21 \cdot \frac{n}{3} + 21 = 14n + 21 \leq 14n + 7n = 21n.$$

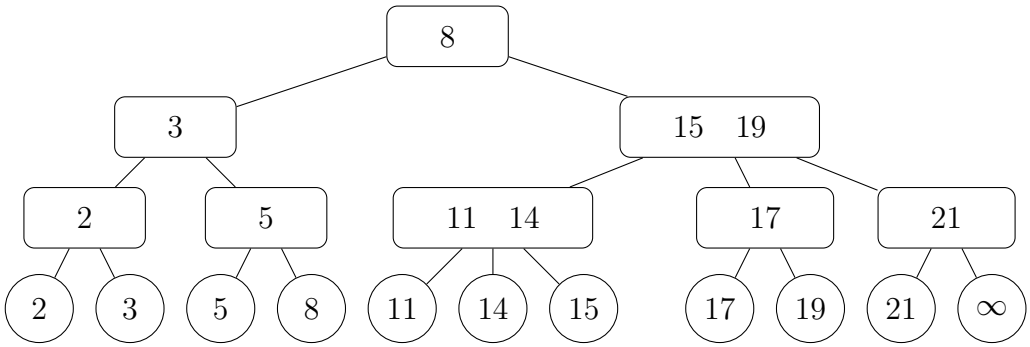
Aufgabe 7 (8 Punkte)

In dieser Aufgabe sind die Schlüssel aufsteigend geordnet. Zeichnen Sie nach **jeder** Operation den durch die Operation entstandenen $(2,3)$-Baum.

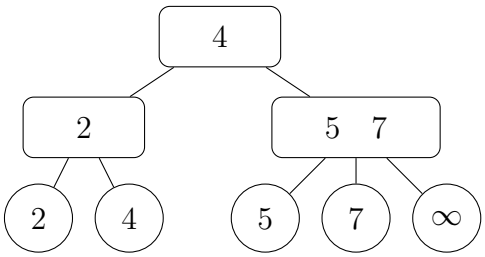
(a) Führen Sie auf dem folgenden $(2,3)$-Baum eine `insert`-Operation des Schlüssels 3 durch.



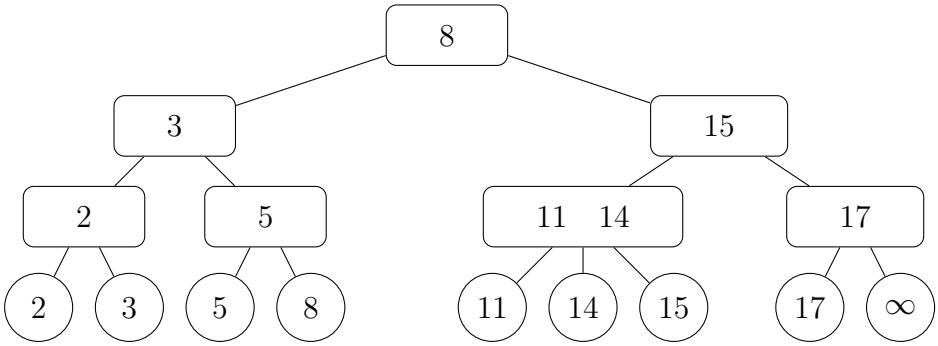
(b) Führen Sie auf dem folgenden $(2,3)$-Baum eine `insert`-Operation des Schlüssels 12 durch.



(c) Führen Sie auf dem folgenden $(2,3)$-Baum eine `delete`-Operation des Schlüssels 4 durch.

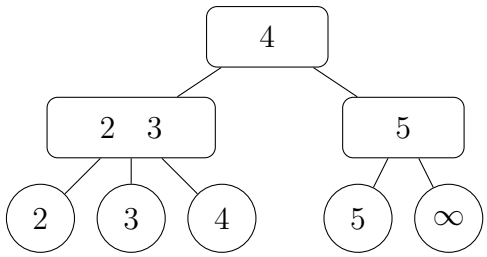


(d) Führen Sie auf dem folgenden $(2,3)$-Baum eine `delete`-Operation des Schlüssels 8 durch.

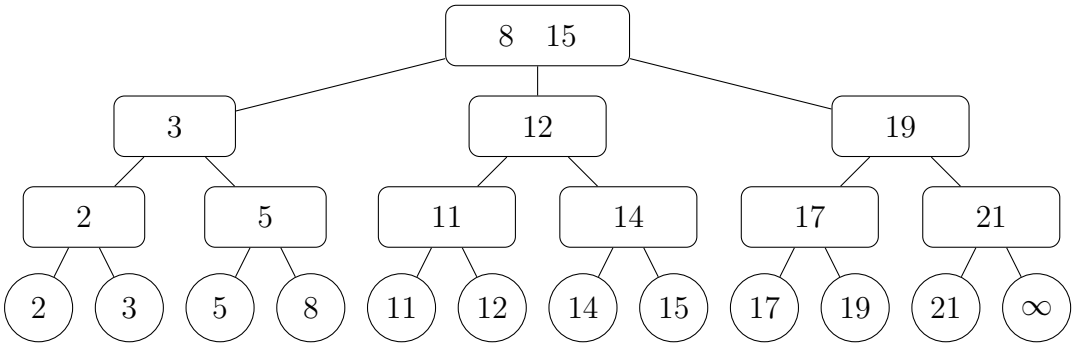


Lösungsvorschlag

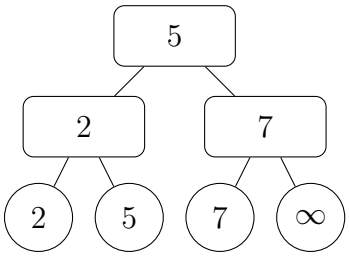
(a)



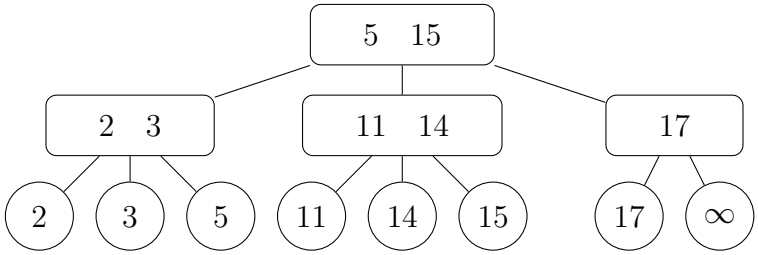
(b)



(c)



(d)



Aufgabe 8 (8 Punkte)

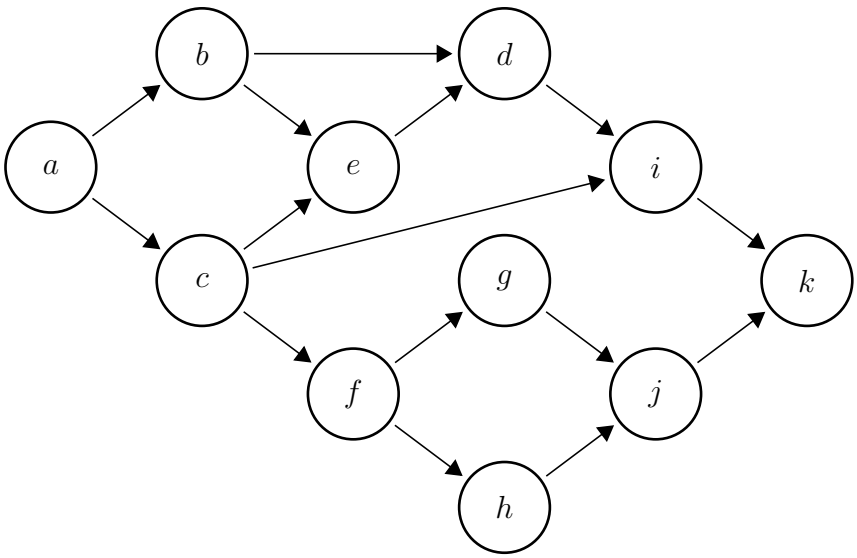
Gegeben ist der folgende Graph.

(a) Nennen Sie die Reihenfolge, in der eine Tiefensuche, die bei Knoten a gestartet wird, die Knoten des Graphen besucht, wenn die Nachbarn eines Knotens in aufsteigender alphabetischer Reihenfolge bearbeitet werden.

(b) Da der Graph kreisfrei ist, können die Knoten topologisch sortiert werden. Nennen Sie die topologische Sortierung der Knoten, die sich aus der Reihenfolge ergibt, in der die von Ihnen in Aufgabe (a) durchgeführte Tiefensuche die Knoten fertig bearbeitet hat.

(c) Nennen Sie die Reihenfolge, in der eine Breitensuche, die bei Knoten a gestartet wird, die Knoten des Graphen besucht, wenn die Nachbarn eines Knotens in aufsteigender alphabetischer Reihenfolge bearbeitet werden.

(d) Nennen Sie, von topologischer Sortierung abgesehen, insgesamt zwei beliebige Anwendungen von (entsprechend modifizierter) Tiefen- oder Breitensuche.



Lösungsvorschlag

(a) $a, b, d, i, k, e, c, f, g, j, h$

(b) Die Knoten werden in der folgenden Reihenfolge fertiggestellt: $k, i, d, e, b, j, g, h, f, c, a$. Indem wir diese Reihenfolge umdrehen, erhalten wir die gewünschte topologische Sortierung: $a, c, f, h, g, j, b, e, d, i, k$

(c) $a, b, c, d, e, f, i, g, h, k, j$

(d) Zum Beispiel SSSP in ungewichteten Graphen oder das Finden von Artikulationsknoten und Blöcken in Graphen.

Aufgabe 9 (8 Punkte)

Die folgende Abbildung enthält zweimal denselben Graphen. Auf diesem wird gerade der Dijkstra-Algorithmus zur Lösung des Single-Source-Shortest-Path-Problems ausgeführt. Der Algorithmus wurde bei Knoten s (links unten) gestartet. Die quadratischen Knoten sind diejenigen, die bereits fertig bearbeitet wurden (für die der Algorithmus also bereits die korrekten Distanzen und Vorgängerknoten ermittelt hat).

Bitte lesen Sie alle Teilaufgaben vollständig, bevor Sie mit der Bearbeitung beginnen. Es muss klar erkennbar sein, welche Markierung was bedeutet.

Bearbeiten Sie folgende Aufgaben im *linken* Graphen:

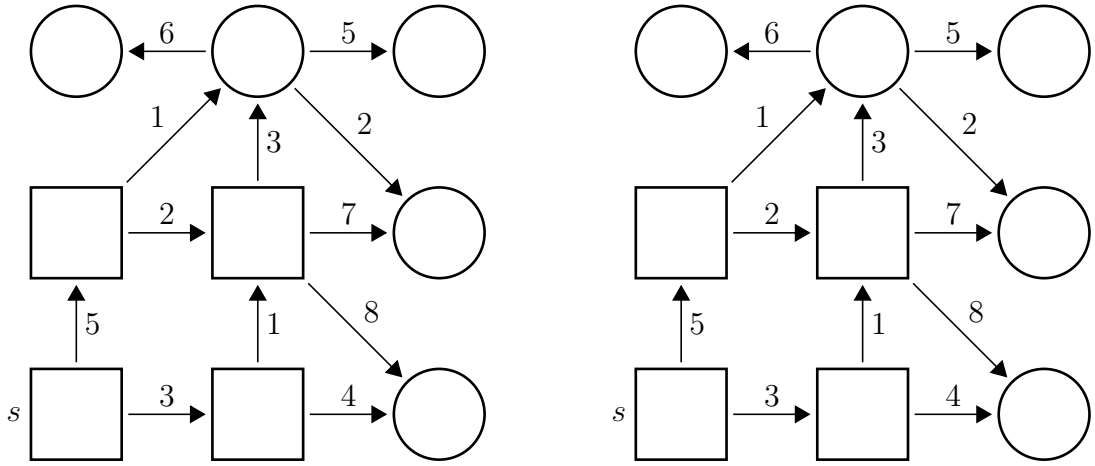
(i) Für jeden fertig bearbeiteten Knoten x, markieren Sie die Kante zwischen x und dem Vorgängerknoten von x im Kürzeste-Wege-Baum, und schreiben Sie die jeweilige Distanz vom Startknoten s zu x in den Knoten x.

(ii) Für jeden Knoten x in der Priority-Queue, markieren Sie x und die Kante vom Vorgängerknoten zu x, und schreiben Sie den aktuellen Prioritätswert von x in den Knoten x (zum Zeitpunkt *bevor* der nächste Knoten aus der Queue entnommen und bearbeitet wird).

Sei v der Knoten, der als nächster aus der Queue entnommen und bearbeitet wird. Bearbeiten Sie folgende Aufgaben im *rechten* Graphen:

(iii) Markieren Sie v und schreiben Sie die Distanz von s zu v in den Knoten v.

(iv) Für jeden Knoten x, der nach der Bearbeitung von v in der Priority-Queue ist, markieren Sie x. Für die Situation nach der Bearbeitung von v, markieren Sie die Kante vom Vorgängerknoten zu x und schreiben Sie den Prioritätswert von x in den Knoten x.



Lösungsvorschlag