Matrikelnummer:			
DHBW Duale Hochschule Baden-Württemberg Stuttgart **ÜBUNGSKLAUSUR**		Fakultät Studiengang: Jahrgang / Kurs : Studienhalbjahr:	**Technik** **Informatik** **2018 A/B/C/D** **2. Semester**
Datum:	**KW29/2019**	Bearbeitungszeit:	**90 Minuten**
Modul: Unit:	**T3INF1003.1** **Algorithmen**	Dozent:	**Schulz, Dietzsch,** **Kötter, Kochanowski**
Hilfsmittel:	**Zwei gebundene Dokumente, z.B. Vorlesungsskript, eigene Notizen**		

Aufgabe	erreichbar	erreicht
1	9	
2	9	
3	10	
4	10	
5	10	
6	12	
7	16	
Summe	76	

1. Sind Sie gesund und prüfungsfähig?
2. Sind Ihre Taschen und sämtliche Unterlagen, insbesondere alle nicht erlaubten Hilfsmittel, seitlich an der Wand zum Gang hin abgestellt und nicht in Reichweite des Arbeitsplatzes?
3. Haben Sie auch außerhalb des Klausurraumes im Gebäude keine unerlaubten Hilfsmittel oder ähnliche Unterlagen liegen lassen?
4. Haben Sie Ihr Handy ausgeschaltet und abgegeben?

(Falls Ziff. 2 oder 3 nicht erfüllt sind, liegt ein Täuschungsversuch vor, der die Note „nicht ausreichend“ zur Folge hat.)

Aufgabe 1 (5+2+2 Punkte)
Betrachten Sie die Folge $S = (3, 15, 22, 2, 37, 7, 111, 9, 12)$

a) Sortieren Sie die Folge S aufsteigend gemäß der normalen Ordnung $<$ auf den natürlichen Zahlen. Verwenden Sie das in der Vorlesung gezeigte *Selection-Sort*-Verfahren. Geben Sie den Zustand von S nach jedem Durchlauf der äußersten Schleife an.

b) Wie viele *Vergleiche* von Elementen der Folge S benötigen Sie?

c) Betrachten Sie nun die *lexikographische Ordnung* $<_{lex}$, bei der Zahlen wie Namen im Telefonbuch verglichen werden - zunächst die erste Ziffer, und bei Gleichheit genau so die folgenden. Dabei ist z.B. $11 <_{lex} 3$ und $3 <_{lex} 9$.

Wie viele Vergleiche würden Sie benötigen, wenn sie die Folge S in Bezug auf $<_{lex}$ aufsteigend sortieren würden?

Lösung

a)

3	15	22	2	37	7	111	9	12
2	15	22	3	37	7	111	9	12
2	3	22	15	37	7	111	9	12
2	3	7	15	37	22	111	9	12
2	3	7	9	37	22	111	15	12
2	3	7	9	12	22	111	15	37
2	3	7	9	12	15	111	22	37
2	3	7	9	12	15	22	111	37
2	3	7	9	12	15	22	37	111

b) 36

c) Ebenfalls 36 - bei Selection Sort hängt die Anzahl der Vergleiche nur von der Anzahl der Elemente ab, nicht von der Ordnung.

Fortsetzung

Aufgabe 2 (2+3+4 Punkte)
Für die folgenden Funktionen sei $x \in \mathbb{R}$.

a) Seien $f : \mathbb{R} \to \mathbb{R}$, $g : \mathbb{R} \to \mathbb{R}$ zwei Funktionen definiert durch $f(x) = x^3, g(x) = 2^x$.

a1) Bestimmem Sie ein beliebiges $k > 3$ für das $g(k) \geq f(k)$ gilt. Begründen Sie ihre Aussage.

a2) Gilt $f \in \mathcal{O}(g)$? Beweisen Sie ihre Aussage.

b) Sei $f : \mathbb{R} \to \mathbb{R}$ mit $f(x) = \sqrt[3]{x}$ und $g : \mathbb{R} \to \mathbb{R}$ mit $g(x) = \ln(x) + \frac{2}{x}$.
Entscheiden Sie, ob $f \in \mathcal{O}(g)$ oder $g \in \mathcal{O}(f)$. Beweisen Sie Ihre Behauptung.

Lösung

a1) Es gilt $10^3 = 1000, 2^{10} = 1024$, also reicht $k = 10$. (oder auch k beliebig ≥ 10)

a2) Es gilt $f \in \mathcal{O}(g)$. Beweis mit dem Grenzwert-Kriterium:

$$\begin{aligned}
& \lim_{x\to\infty} \frac{x^3}{2^x} \\
= \quad & \lim_{x\to\infty} \frac{3!}{(\ln 2)^3 2^x} \qquad \text{(3 mal Ableiten mit l'Hopital)} \\
= \quad & \frac{3!}{(\ln 2)^3} \lim_{x\to\infty} \frac{1}{2^x} \\
= \quad & 0 \in \mathbb{R}
\end{aligned}$$

b) Behauptung: $g \in \mathcal{O}(f)$. Beweis mit dem Grenzwertkriterium:

$$\begin{aligned}
& \lim_{x\to\infty} \frac{\ln x + \frac{2}{x}}{\sqrt[3]{x}} \\
= \quad & \lim_{x\to\infty} \left(\frac{\ln x}{\sqrt[3]{x}} + \frac{\frac{2}{x}}{\sqrt[3]{x}}\right) \\
= \quad & \lim_{x\to\infty} \left(\frac{\ln x}{\sqrt[3]{x}} + \frac{2}{x\sqrt[3]{x}}\right) \\
= \quad & \lim_{x\to\infty} \frac{\ln x}{\sqrt[3]{x}} + \lim_{x\to\infty} \frac{2}{x\sqrt[3]{x}} \\
= \quad & \lim_{x\to\infty} \frac{\ln x}{x^{\frac{1}{3}}} \\
= \quad & \lim_{x\to\infty} \frac{\frac{1}{x}}{\frac{1}{3}x^{-\frac{2}{3}}} \qquad \text{l'Hopital} \\
= \quad & \lim_{x\to\infty} \frac{\frac{1}{x}}{\frac{1}{3x^{\frac{2}{3}}}} \\
= \quad & \lim_{x\to\infty} \frac{3x^{\frac{2}{3}}}{x} \\
= \quad & \lim_{x\to\infty} \frac{3}{\sqrt[3]{x}} \\
= \quad & 0 \in \mathbb{R}
\end{aligned}$$

Fortsetzung

Aufgabe 3 (1+1+2+5+1 Punkte)
Betrachten Sie die folgende C-Funktion.

```
int tuwas(int n)
{
    int i, res = 0;

    if(n != 0)
    {
        for(i=0; i<n; i++)
        {
            res = res+(2*i);
        }
        res = res+tuwas(n-1);
    }
    return res;
}
```

a) Bestimmen Sie den Rückgabewert für die Eingaben $n = 1, n = 2, n = 3$.

b) Bestimmen Sie das kleinste $k \in \mathbb{N}$, so dass die *Laufzeitkomplexität* von `tuwas()` in $\mathcal{O}(n^k)$ ist. Sie können davon ausgehen, dass n nicht negativ ist. Begründen Sie Ihre Antwort.

c) Was passiert, wenn n negativ ist?

Lösung

a) $1 \mapsto 0, 2 \mapsto 2, 3 \mapsto 8$

b) $k = 2$, d.h. die Laufzeitkomplexität von `tuwas()` ist in $\mathcal{O}(n^2)$.

Begründung:

- Die Rekursion terminiert für $n = 0$.
- Die Schleife läuft n Durchläufe bei Eingabe von n, und der Aufwand in der Schleife ist konstant. Damit ist die Rekurrenzrelation für die Laufzeit $t(n) = c \cdot n + t(n-1)$.
- Damit sind die Kosten $t(n) = c \cdot \sum_{i=0}^{n} i$, und mit Gauß $t(n) = c \cdot \frac{n(n+1)}{2} = c \cdot \frac{n^2+n}{2}$.

c) Auf einem idealen Computer läuft die Funktion endlos. Auf einem realen Computer wird n irgendwann einen Underflow erleiden und dann eventuell terminieren. Auf einem noch realeren Computer läuft der Stack über und es gibt einen Segmentation Fault. (Volle Punktzahl schon für den idealen Fall)

Aufgabe 4 (3+4+3 Punkte)

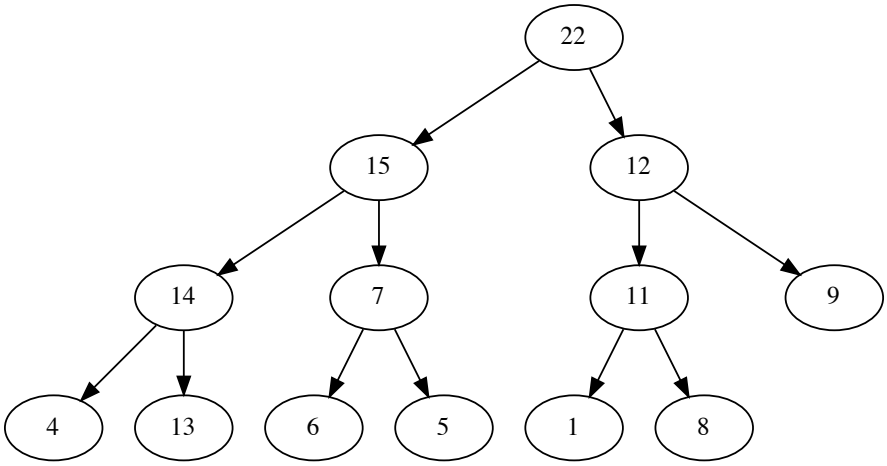
a) Betrachten sie einen binären Max-Heap mit 123 Elementen.

a1) Wie tief ist der Heap als Baum? D.h. wie viele Ebenen hat der Heap? Begründen Sie ihre Aussage!

a2) Wie viele Knoten hat der Heap auf der letzten (tiefsten) Ebene? Begründen Sie ihre Aussage!

b) Betrachten Sie den Max-Heap H unten auf der Seite. Fügen Sie nacheinander die Elemente 3, 20 und 42 in den Heap ein und stellen Sie jeweils die Heap-Eigenschaft mit dem in der Vorlesung gezeigten Verfahren wieder her. Zeichnen Sie den Heap nach jeder Einfügeoperation!

c) Betrachten Sie wieder den Original-Heap H (nicht das Ergebnis von Teil b!). Extrahieren Sie das größte Element des Heaps und stellen Sie die Heap-Eigenschaft wieder her. Zeichnen Sie das Ergebnis.



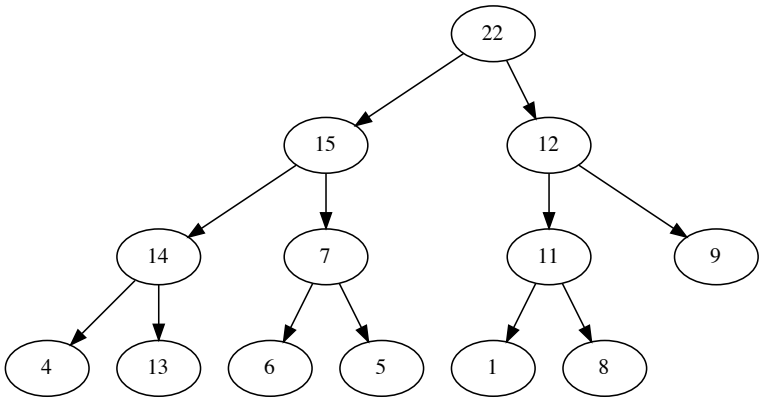
Heap H (auf den nächsten Seiten wiederholt)

Lösung

a1) Ein binärer Heap ist ein fast vollständiger Binärbaum. Ein solcher mit 123 Elementen hat immer 7 Ebenen (6 volle Ebenen mit insgesamt 63 Elementen und die unvollständige Ebene. (1.5P)

a2) Ein vollständiger Binärbaum mit 6 Ebenen hat 63 Elemente, also bleiben 60 Elemente für die letzte Ebene. (1.5P)

Fortsetzung



Lösung

b) (je 1 Punkt für 3, 20, 2 Punkte für 42)

```
Mit 3:
              22
      15              12
  14       7      11       9
 4  13   6   5   1   8   3
```

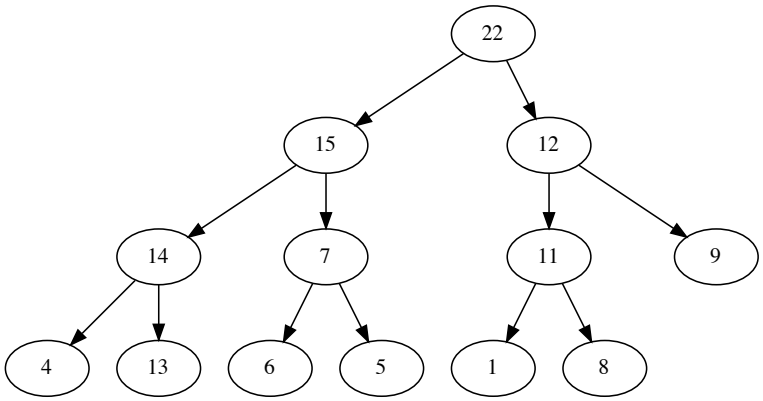
```
Mit 3 und 20:
              22               1
      15              20
  14       7      11      12
 4  13   6   5   1   8   3   9
```

```
Mit 3, 20, 42:
                            42
              22                                20
      15              7              11              12
  14      13       6       5       1       8       3       9
 4
```

c) Ohne 22:

```
              15
      14              12
  13       7      11       9
 4   8   6   5   1
```

Fortsetzung

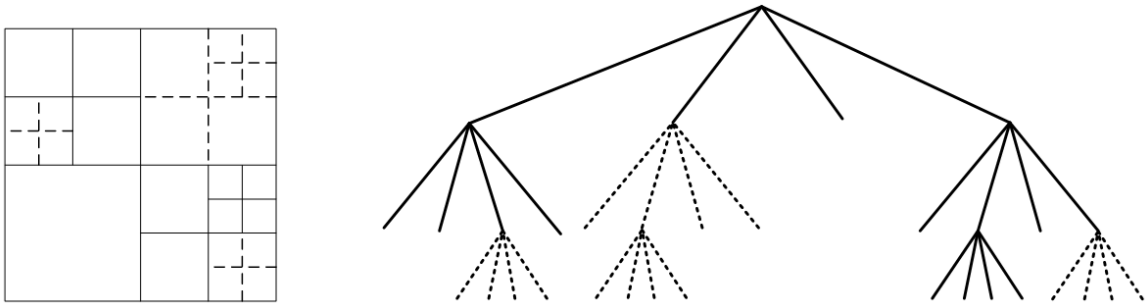


Aufgabe 5 (2+4+4 Punkte)

a) Geben Sie für die Rekurrenzrelation G eine $\mathcal{O}$-Lösung an.

$$G(n) = 8 \cdot G(\left\lfloor \frac{n}{2} \right\rfloor) + 3n^4 \quad \text{für } n \in \mathbb{N}$$

b) Ein *QuadTree* ist eine Datenstruktur zum schnellen Finden von Objekten anhand von kartesischen Koordinaten in der Ebene (also anhand der Position in X,Y). Jeder Knoten des QuadTrees repräsentiert ein Quadrat in der Ebene. Der Wurzelknoten repräsentiert den gesamte relevanten Ausschnitt der Ebene. In einem Knoten werden - direkt oder indirekt - alle Objekte gespeichert, die sich in diesem Quadrat befinden. Wenn die Anzahl der direkt in einem Quadrat gespeicherten Objekte zu groß wird, so wird das Quadrat in 4 Teilquadrate mit jeweils halber Seitenlänge gespalten, und nur diese direkt im Knoten gespeichert, während die Objekte selbst in den Knoten zum dem passenden Unterquadrat verlagert werden. Ein Quadrat enthält also immer entweder nur genau 4 Unterquadrate oder bis zu k Objekte in einer linearen Liste. Sie können im folgenden davon ausgehen, dass der gesamte QuadTree n Objekte enthält, und dass die gespeicherten Objekte gleichmäßig in der Ebene verteilt sind.



Beispiel für einen QuadTree (©Sheng-Hsiang Chang wikimedia commons)

- b1) Bestimmen Sie (im Sinne der $\mathcal{O}$-Notation) die Komplexität der Operation `find(tree, x, y)`, die zu einem Satz Koordinaten herausfindet, ob ein Objekt an diesen Koordinaten existiert. Sie können davon ausgehen, dass die Suche nach einem Objekt in einem Blatt-Quadrat Komplexität $\mathcal{O}(1)$ hat (es sind ja höchstens k Objekte zu prüfen).
- b2) Bestimmen Sie (im Sinne der $\mathcal{O}$-Notation) die Komplexität der Operation `collect(tree)`, die eine Liste aller gespeicherten Objekte erzeugt. Sie können davon ausgehen, dass das Zusammenfassen von Teilergebnissen mit insgesamt n Objekten Kosten $\mathcal{O}(n)$ hat.

Lösung

a) Master-Theorem mit $a = 8, b = 2, d = 4$, also Fall 1: $G \in \Theta(n^4)$

b1) Die Suche erfolgt rekursiv. Auf jeder Ebene wird das Problem auf ein Teilproblem mit der Größe 1/4 reduziert. Das Durchreichen des Ergebnisses (gefunden oder nicht) hat Kosten $\mathcal{O}(1)$. Damit ergibt sich als Rekurrenzrelation

$$F(n) = 1 \cdot F(\left\lfloor \frac{n}{4} \right\rfloor) + c$$

Mit dem Master-Theorem ergibt sich $a = 1, b = 4, d = 0$, damit Fall 2 und $F(n) \in \mathcal{O}(\log_4 n * n^0) = \mathcal{O}(\log_4 n)$

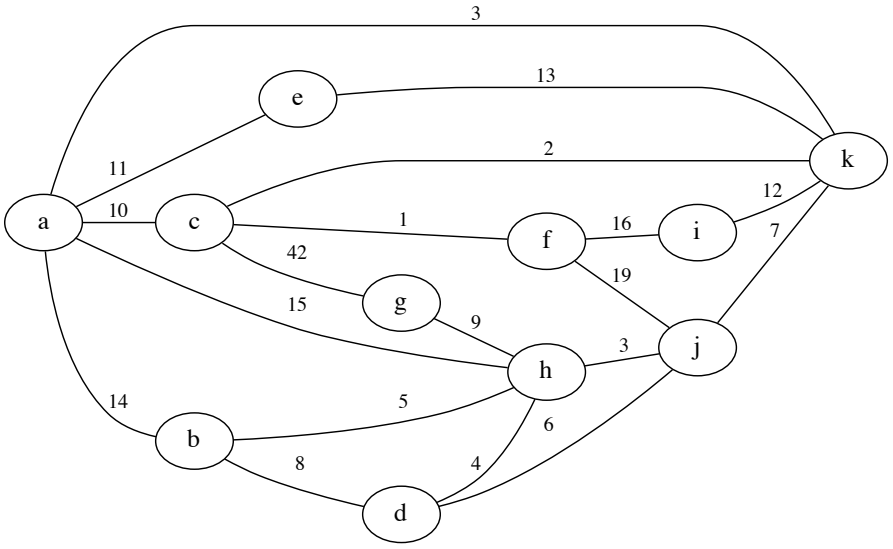
b2) Um alle Objekte eines Knotens zu sammeln muss der Algorithmus rekursiv alle 4 Teilknoten durchsuchen. Die gefundenen n Objekte müssen jeweils mit konstanten Kosten pro Objekt in die Ergebnisliste eingefügt werden. Damit ergibt sich:

$$C(n) = 4 \cdot C(\left\lfloor \frac{n}{4} \right\rfloor) + cn$$

Master-Theorem mit $a = 4, b = 4, d = 1$, also Fall 2 und $C(n) \in \mathcal{O}(n \log_4 n)$.

Fortsetzung

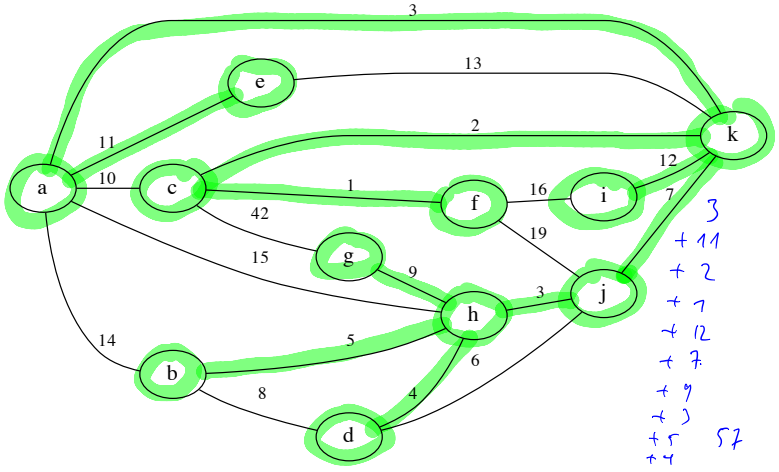
Aufgabe 6 (5+7 Punkte)
Gegeben sei der folgende Graph G.



a) Bestimmen Sie für G ausgehend vom Knoten **g** einen minimalen Spannbaum mit Hilfe des Prim-Algorithmus. Sie können hierzu die benutzten Kanten im Bild *sauber* markieren oder eine Liste der verwendeten Kanten angeben. Geben Sie die Reihenfolge an, in der Sie die Knoten dem Spannbaum hinzufügen. Wie hoch ist das Gesamtgewicht der Kanten des minimalen Spannbaums?

b) Verwenden Sie den Algorithmus von Dijkstra, um die minimale Entfernung aller Knoten in G vom Knoten **a** zu bestimmen. Geben Sie die Knoten in der Reihenfolge an, in der sie im Algorithmus durchlaufen werden. Auf der nächsten Seite finden Sie eine Kopie des Graphen und eine Tabelle für das Ergebnis.

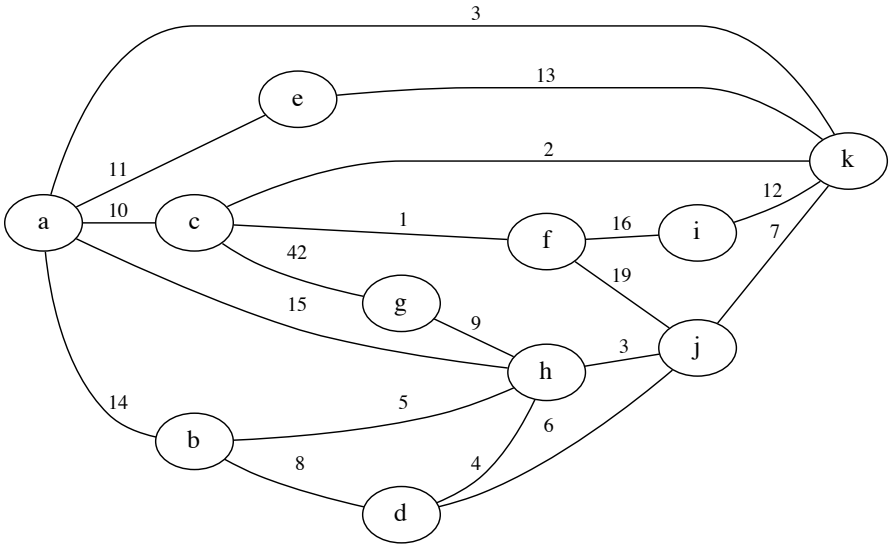
Lösung



a) Gesamtgewicht ist 57.

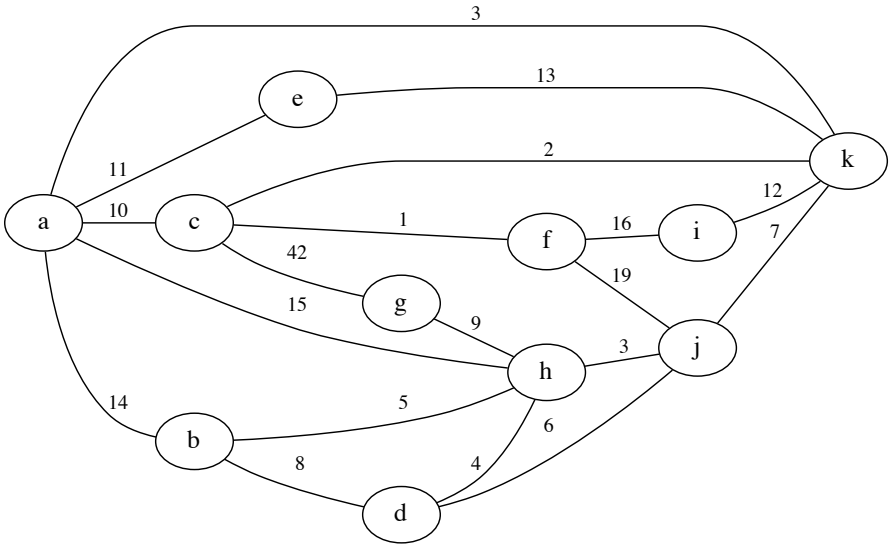
- Reihenfolge ist g (Start),h (9), j (3), d (4), b (5), k (7), c (2), f (1), a (3), e (11), i (12)

Fortsetzung

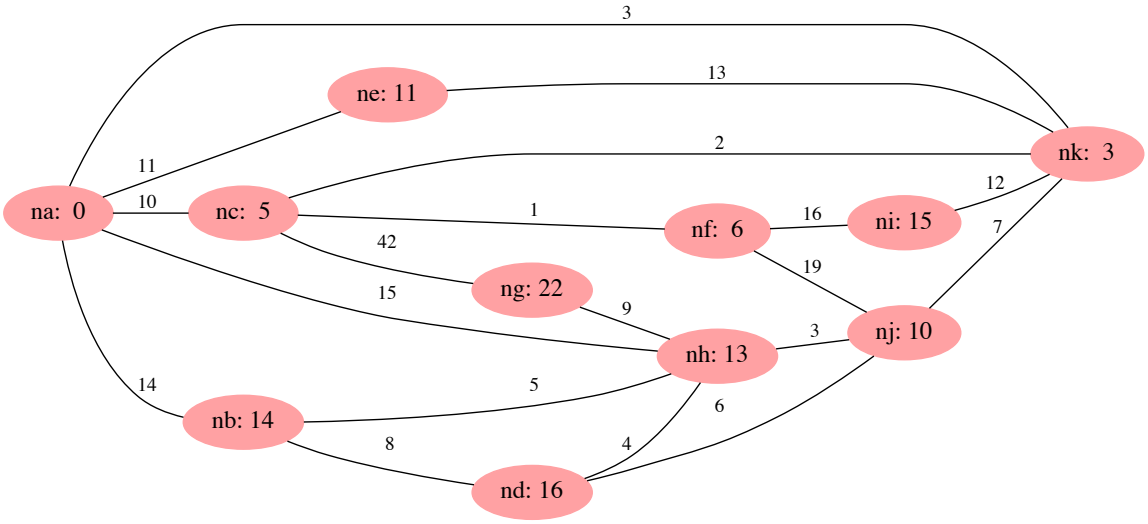


Knoten	Abstand
a	0

Fortsetzung



Lösung



b)

Mit Reihenfolge:

Knoten	Abstand
a	0
k	3
c	5
f	6
j	10
e	11
h	13
b	14
i	15
d	16
g	22

Alphabetisch:

Knoten	Abstand
a	0
b	14
c	5
d	16
e	11
f	6
g	22
h	13
i	15
j	10
k	3

Aufgabe 7 (7+2+3+4 Punkte)

a) Gegeben sei die Schlüsselfolge

$$S = (16, 55, 17, 68, 72, 18, 33, 28, 15)$$

Fügen Sie die Elemente von S in dieser Reihenfolge in eine Hash-Tabelle der Größe 13 (Indizes 0-12) ein. Verwenden Sie als Hash-Funktion $h(x) = ((x + 3) \mod 13)$. Lösen Sie Kollisionen mit Hilfe von Linear Probing auf.

Geben Sie den Inhalt der Hash-Tabelle nach jeder Einfüge-Operation an. Die Tabelle unten enthält hierzu eine Spalte für jede Einfüge-Operation.

Markieren Sie bitte außerdem in jedem Schritt die Kollisionen, indem Sie die Werte einkreisen, mit denen der einzufügende Wert kollidiert (z.B. (16)). Wie viele Kollisionen treten auf? (Für jeden Wert kann es mehrere Kollisionen geben.)

b) Betrachten Sie den AVL-Baum A auf der nächsten Seite.

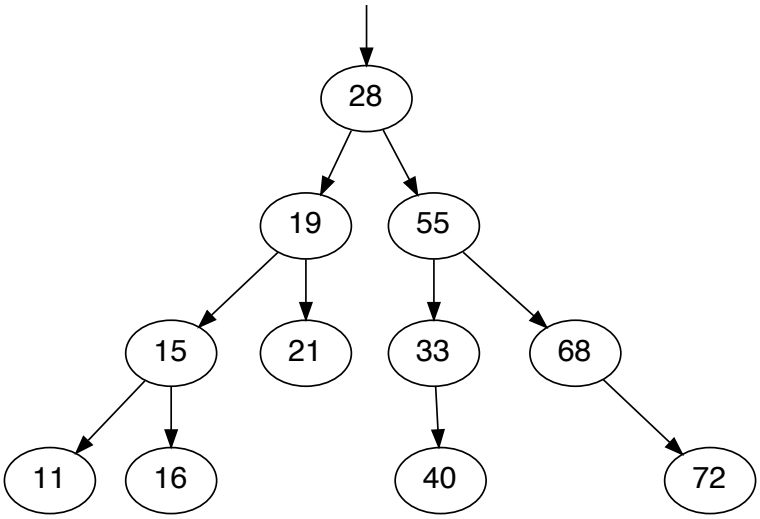
b1) Bestimmen Sie die Höhenbalance an jedem Knoten. Sie können die Werte in den vorhandenen Baum neben die Knoten schreiben.

b2) Fügen Sie den Wert 75 in den Baum A ein und rebalancieren ihn entsprechend dem AVL-Algorithmus. Geben Sie jede Einzelrotation in der Form „links-/rechts-Rotation mit X als Wurzel und Y als Pivot“ an. Zeichnen Sie das Ergebnis.

b3) Fügen Sie nun den Wert 17 in das Ergebnis von b2) ein und rebalancieren Sie den resultierenden Baum entsprechend dem AVL-Algorithmus. Geben Sie jede Einzelrotation in der Form „links-/rechts-Rotation mit X als Wurzel und Y als Pivot“ an. Zeichnen Sie das Ergebnis.

Feld	16	55	17	68	72	18	33	28	15
0									
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									

Fortsetzung



AVL Baum A (Einzufügen sind 75 (b2) und 17 (b3))

Lösung

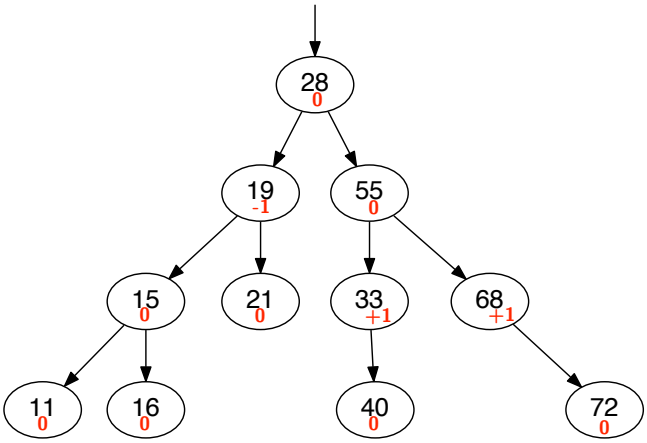
a) Je 0.5 Punkte für ie ersten 6, 1 Punkt für die verbleibenden 3, 1 Punkt für die Kollisionen

	16	55	17	68	72	18	33	28	15
0									15
1									
2									
3									
4									
5								28	(28)
6	16	(16)	16	(16)	16	16	16	16	(16)
7		55	(55)	(55)	55	55	55	55	(55)
8			17	(17)	17	(17)	17	17	(17)
9				68	68	(68)	68	68	(68)
10					72	(72)	(72)	72	(72)
11						18	(18)	18	(18)
12							33	33	(33)

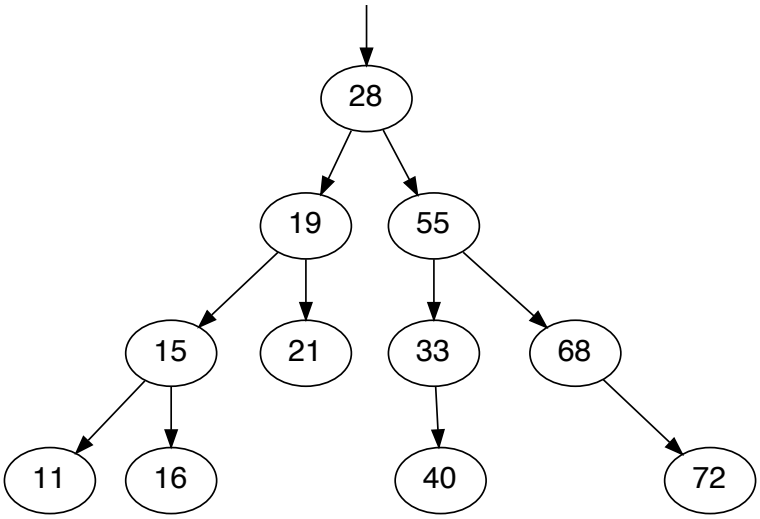
18 Kollisionen: 1x55, 1x17, 3x68, 3x18, 2x33, 8x15

b1) Balance-Werte:

```
: 0: 0:  1:        11
: 0: 0:  2:     15
: 0: 0:  1:        16
:-1:-1:  3:   19
: 0: 0:  1:     21
: 0: 0:  4: 28
: 1: 1:  2:     33
: 0: 0:  1:        40
: 0: 0:  3:   55
: 1: 1:  2:     68
: 0: 0:  1:        72
```



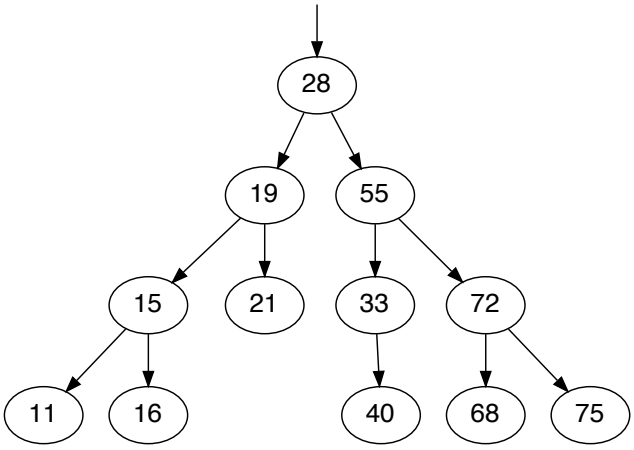
Fortsetzung



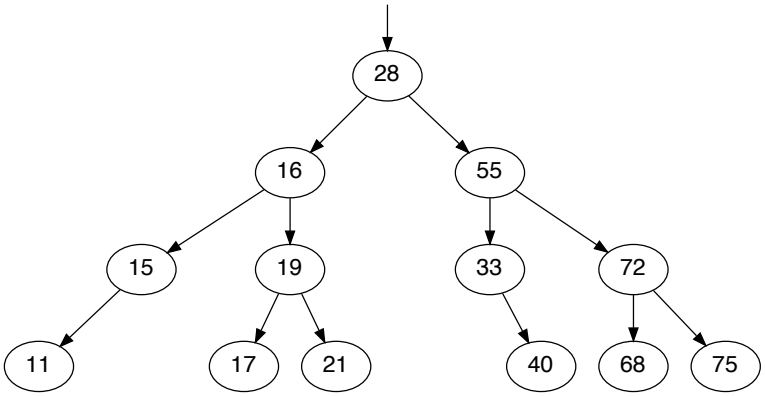
AVL Baum A (Einzufügen sind 75 (b2) und 17 (b3))

Lösung

b2) Links-Rotation mit 68 als Root, 72 als Pivot



b3) Erst Links-Rotation mit 15 als Root, 16 als Pivot, dann Rechts-Rotation mit 19 als Root, 16 als Pivot



– Ende der Klausur –