Aufgabe 1 ($\mathcal{O}$-Notation): (15 + 6 = 21 Punkte)

a) Tragen Sie in die durch ____ gekennzeichneten freien Felder entweder o, ω oder Θ ein, sodass die entsprechende Aussage gilt. Beispielsweise wäre bei der Aussage $f(n) \in$ ____$(f(n))$ ein Θ einzutragen, da die Aussage $f(n) \in \Theta(f(n))$ gilt, die Aussagen $f(n) \in o(f(n))$ und $f(n) \in \omega(f(n))$ jedoch beide nicht gelten.

- $n \in \underline{\quad\quad} \left(\sum_{i=0}^{n} \frac{i}{n} \right)$
- $n^n \in \underline{\quad\quad} (n!)$
- $\sum_{i=0}^{n} \frac{i}{n} \in \underline{\quad\quad} (n^2)$
- $n! \in \underline{\quad\quad} \left(\sum_{i=0}^{3} i^n \right)$
- $\frac{\log_{14}(n)}{n} \in \underline{\quad\quad} \left(\frac{1}{4} \right)$
- $n^{155} + 3^n \in \underline{\quad\quad} \left(\sum_{i=0}^{3} i^n \right)$
- $n^2 \in \underline{\quad\quad} \left(\sum_{i=0}^{n} i \right)$
- $\sum_{i=0}^{3} i^n \in \underline{\quad\quad} (n^2)$
- $\left(\frac{8}{7} \right)^n \in \underline{\quad\quad} (n^2)$
- $\log_2(n) \in \underline{\quad\quad} (n)$

b) Beweisen oder widerlegen Sie:

$$f(n) \cdot g(n) \in \mathcal{O}\left(f(n)^{g(n)} \right)$$

Lösung:

a)

- $n \in \underline{\ \Theta\ } \left(\sum_{i=0}^{n} \frac{i}{n} \right)$
- $n^n \in \underline{\ \omega\ } (n!)$
- $\sum_{i=0}^{n} \frac{i}{n} \in \underline{\ o\ } (n^2)$
- $n! \in \underline{\ \omega\ } \left(\sum_{i=0}^{3} i^n \right)$

- $\frac{\log_{14}(n)}{n} \in \underline{\ o\ } \left(\frac{1}{4}\right)$
- $n^{155} + 3^n \in \underline{\ \Theta\ } \left(\sum_{i=0}^{3} i^n\right)$
- $n^2 \in \underline{\ \Theta\ } \left(\sum_{i=0}^{n} i\right)$
- $\sum_{i=0}^{3} i^n \in \underline{\ \omega\ } (n^2)$
- $\left(\frac{8}{7}\right)^n \in \underline{\ \omega\ } (n^2)$
- $\log_2(n) \in \underline{\ o\ } (n)$

b) **Behauptung:** Die Aussage gilt nicht.

Beweis:
Gegenbeispiel: Sei $f(n) = 1$ und $g(n) = n$. Dann gilt $f(n) \cdot g(n) = 1 \cdot n = n$ und $f(n)^{g(n)} = 1^n = 1$. Es gibt aber offensichtlich kein c und n_0, sodass für alle $n \geq n_0$ gilt:

$$f(n) \cdot g(n) = n \leq c = c \cdot 1 = c \cdot f(n)^{g(n)}$$

$\square$

Aufgabe 2 (Rekursionsgleichungen): (10 + 8 + 8 = 26 Punkte)

a) Geben Sie für das Programm

```
int berechne(int n) {
    if (n <= 1)
        return 5000;

    int value = func(n);

    int k = 5;
    while (k >= 1) {
        value = value + value * berechne(n/3);
        k = k - 1;
    }

    return value;
}

int func(int n) {
    int res = 0;

    while (n > 0) {
        int m = n;
        while (m > 0) {
            res = res + m;
            m = m - 1;
        }
        n = n - 1;
    }

    return res;
}
```

eine Rekursionsgleichung für die **asymptotische** Laufzeit des Aufrufes `berechne(n)` in Abhängigkeit von `n` an. Die elementaren, also die für die asymptotische Laufzeit relevanten, Operationen sind alle arithmetischen Operationen sowie Vergleiche. Sie brauchen die Basisfälle der Rekursionsgleichung *nicht* anzugeben.

b) Bestimmen Sie für die Rekursionsgleichung

$$T(n) = 8 \cdot T\left(\frac{n}{2}\right) + n^3 + 4 \cdot n + \frac{1}{n}$$

die Komplexitätsklasse Θ mit Hilfe des Master-Theorems. Begründen Sie Ihre Antwort.

c) Sei $T(n)$ rekursiv wie folgt definiert:

$$T(n) = \begin{cases} 1, & \text{falls } n = 0 \\ 2 \cdot T(n-1), & \text{andernfalls.} \end{cases}$$

Beweisen Sie die folgende Aussage mittels vollständiger Induktion:

$$T(n) \in \Theta(2^n)$$

Lösung: ______

a) `func(n)` hat quadratische Laufzeit im Parameter n, da die äußere `while`-Schleife n mal und die innere `while`-Schleife im Durchschnitt $\frac{n}{2}$ mal durchlaufen wird.

Jeder Aufruf von `berechne(n)` führt zu 5 rekursiven Aufrufen (da die `while`-Schleife 5 mal durchlaufen wird), jeweils mit Parameter $n/3$ Somit ergibt sich die folgende Rekursionsgleichung für die asymptotische Laufzeit von `berechne`(n):

$$T(n) \;=\; 5 \cdot T\left(\frac{n}{3}\right) + n^2 + c \qquad T(0) = T(1) = 1$$

b) Es sind $b = 8$, $c = 2$ und $f(n) = n^3 + 4 \cdot n + \frac{1}{n}$. E wird nun wie folgt bestimmt:

$$E \;=\; \frac{\log(8)}{\log(2)} \;=\; \frac{3}{1} \;=\; 3$$

Damit gilt $n^E = n^3$. Wir bestimmen nun den folgenden Grenzwert:

$$\begin{aligned}
& \lim_{n\to\infty} \frac{f(n)}{n^E} \\
= \; & \lim_{n\to\infty} \frac{n^3 + 4 \cdot n + \frac{1}{n}}{n^3} \\
= \; & \lim_{n\to\infty} \frac{1 + \frac{4}{n^2} + \frac{1}{n^4}}{1} \\
= \; & \lim_{n\to\infty} 1 + \frac{4}{n^2} + \frac{1}{n^4} \\
= \; & \lim_{n\to\infty} 1 + \lim_{n\to\infty} \frac{4}{n^2} + \lim_{n\to\infty} \frac{1}{n^4} \\
= \; & 1 + 0 + 0 \;=\; 1
\end{aligned}$$

Damit gilt $f(n) \in \Theta\left(n^E\right)$. Somit findet der *zweite Fall* des Master-Theorems Anwendung und es ergibt sich die Komplexitätsklasse $\Theta\left(n^E \cdot \log(n)\right)$ für $T(n)$, also

$$T(n) \in \Theta\left(n^3 \cdot \log(n)\right) \; .$$

c) **Behauptung:** Die Aussage gilt.

Beweis:
Es ist also zu zeigen: Es existieren c_1, c_2 und n_0, sodass für alle $n \geq n_0$ gilt:

$$c_1 \cdot 2^n \;\leq\; T(n) \;\leq\; c_2 \cdot 2^n$$

Wähle $c_1 = c_2 = 1$ und $n_0 = 0$. Wir zeigen nun mittels vollständiger Induktion die Aussage

$$1 \cdot 2^n \;\leq\; T(n) \;\leq\; 1 \cdot 2^n$$

Induktionsanfang: $n = 0$.

$$1 \cdot 2^0 = 1 = T(0) \;\leq\; T(0) \;\leq\; T(0) = 1 = 1 \cdot 2^0$$

Induktionshypothese. Für beliebiges aber festes n gilt:

$$1 \cdot 2^n \;\leq\; T(n) \;\leq\; 1 \cdot 2^n$$

Induktionsschritt: $\boldsymbol{n} \rightarrow \boldsymbol{n}+1$ Laut Induktionshypothese gilt:

$$1 \cdot 2^n \;\leq\; T(n) \;\leq\; 1 \cdot 2^n$$

Durch Multiplizieren der Ungleichung mit 2 erhalten wir:

$$\begin{aligned} & 1 \cdot 2^n \cdot 2 \;\leq\; T(n) \cdot 2 \;\leq\; 1 \cdot 2^n \cdot 2 \\ \iff & 1 \cdot 2^{n+1} \;\leq\; T(n+1) \;\leq\; 1 \cdot 2^{n+1} \end{aligned}$$

Damit ist die Aussage gezeigt. $\square$

Aufgabe 3 (Sortieren): (4 + 5 + 3 = 12 Punkte)

a) Sortieren Sie das folgende Array mithilfe von Insertionsort. Geben Sie dazu das Array nach jeder Iteration der äußeren Schleife an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

4	3	8	7	2

b) Sortieren Sie das folgende Array mithilfe von Quicksort. Geben Sie dazu das Array nach jeder Partition-Operation an und markieren Sie das jeweils verwendete Pivot-Element. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

9	7	3	1	6	8	2	5

c) Wenden Sie die buildHeap Operation aus der Vorlesung (also den Anfang von Heapsort) auf das folgende Array an, um darauf die Max-Heap-Eigenschaft herzustellen. Geben Sie dazu das Array nach jeder Swap-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

4	7	8	3	2	9

Lösung:

a)

4	3	8	7	2
3	4	8	7	2
3	4	8	7	2
3	4	7	8	2
2	3	4	7	8

b)

9	7	3	1	6	8	2	5
2	1	3	**5**	6	8	9	7
2	1	**3**	5	6	8	9	7
1	2	3	5	6	8	9	7
1	2	3	5	6	**7**	9	8
1	2	3	5	6	7	**8**	9

c)

4	7	8	3	2	9

4	7	9	3	2	8

9	7	4	3	2	8

9	7	8	3	2	4

Aufgabe 4 (Hashing): (3 + 3 = 6 Punkte)

a) Fügen Sie die folgenden Werte in das unten stehende Array a der Länge 10 unter Verwendung der *Divisionsmethode* mit *linearer Sondierung* ein:

3, 15, 13, 24, 23, 12.

b) Fügen Sie die folgenden Werte in das unten stehende Array a der Länge 10 unter Verwendung der *Divisionsmethode* mit *quadratischer Sondierung* ($c_1 = 0.0$, $c_2 = 1.0$) ein:

7, 28, 17, 10, 20, 27.

Lösung:

a) m = 10:

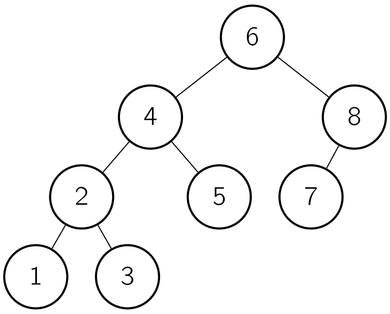
		12	3	13	15	24	23		

b) m = 10, $c_1 = 0.0$, $c_2 = 1.0$:

10	17			20		27	7	28	

Aufgabe 5 (Bäume): (4 + 7 + 6 = 17 Punkte)

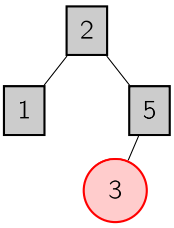
a) Löschen Sie den Wert 8 aus dem folgenden *AVL-Baum* und geben Sie die entstehenden Bäume nach jeder *Löschoperation* sowie jeder *Rotation* an. Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird:



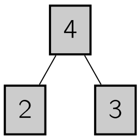
b) Fügen Sie den Wert 4 in den folgenden *Rot-Schwarz-Baum* ein und geben Sie die entstehenden Bäume nach

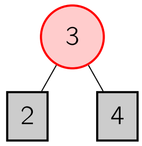
- jeder *Einfügeoperation*,
- jeder *Rotation* sowie
- jeder *Umfärbung* an.

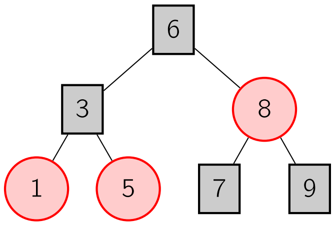
Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird. Mehrere Umfärbungen können Sie in einem Schritt zusammenfassen. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.

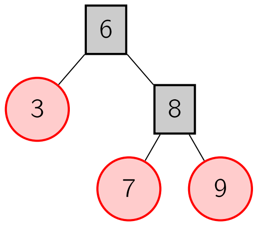


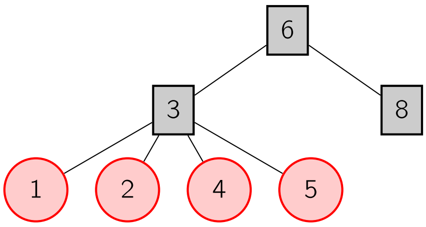
c) Geben Sie zu den folgenden Bäumen an, ob es sich dabei jeweils um einen gültigen Rot-Schwarz-Baum handelt. Falls dies nicht der Fall sein sollte, geben Sie mindestens eine Eigenschaft eines Rot-Schwarz-Baums an, die der jeweilige Baum verletzt. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.

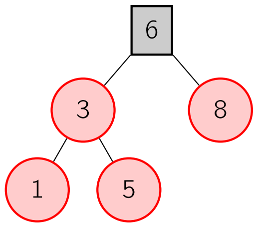






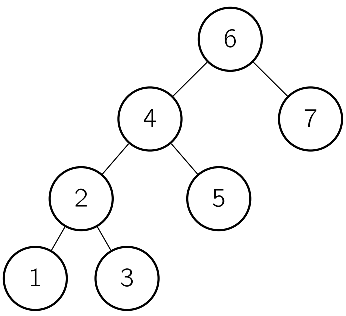




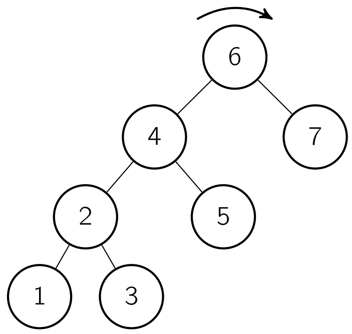


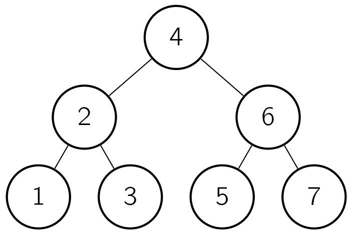
Lösung:

a) entferne 8

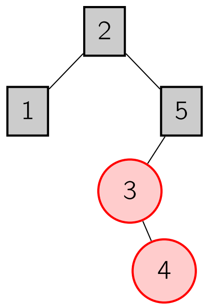


rotiere 6 nach rechts

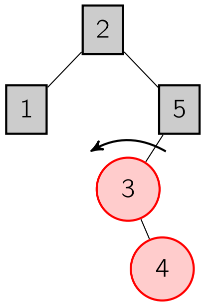


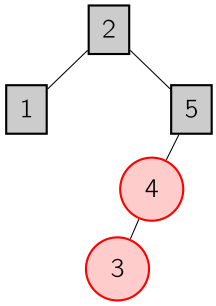


b) füge 4 ein

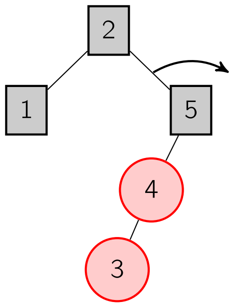


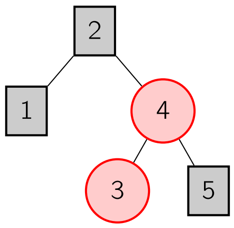
Fall 2 (Onkel ist null und damit schwarz und aktueller Knoten liegt innen): rotiere 3 nach links



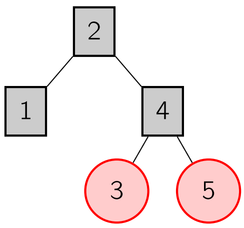


Fall 3 (Onkel ist null und damit schwarz und aktueller Knoten liegt außen): rotiere 5 nach rechts

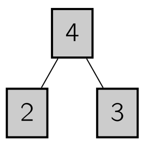




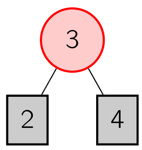
Fall 3 (Fortsetzung): umfärben



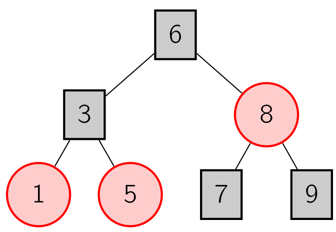
c)



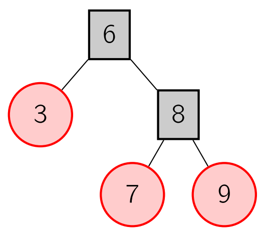
Der Baum ist kein Suchbaum (Knoten 3).



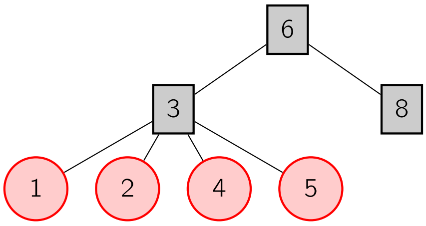
Die Wurzel ist nicht schwarz.



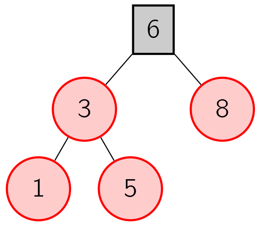
Gültiger Rot-Schwarz-Baum.



Schwarzhöhen-Verletzung an Knoten 6.



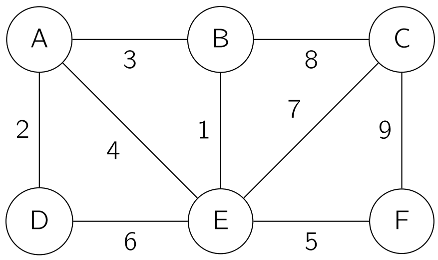
Der Baum ist kein Binärbaum (Knoten 3).



Rot-Rot-Verletzung an Knoten 3.

Aufgabe 6 (Graphen): (6 + 8 + 3 + 6 = 23 Punkte)

a) Führen Sie Prim's Algorithmus auf dem folgenden Graphen aus.



Der Startknoten hat hierbei den Schlüssel A. Geben Sie dazu *vor* jedem Durchlauf der äußeren Schleife an,

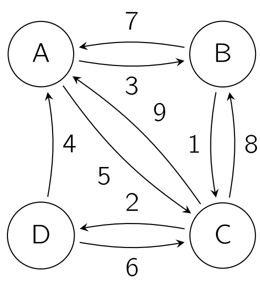
a) welche Kosten die Randknoten haben (d. h. für jeden Knoten `v` in `pq` die Priorität von `v`, wobei ∞ angibt, dass der entsprechende Knoten noch nicht zum Randbereich gehört)
b) und welchen Knoten `pq.getMin()` wählt, indem Sie den Kosten-Wert des gewählten Randknoten in der Tabelle unterstreichen (wie es in der ersten Zeile bereits vorgegeben ist).

Geben Sie zudem den vom Algorithmus bestimmten minimalen Spannbaum an.

#Iteration	A	B	C	D	E	F
1	$\underline{0}$	∞	∞	∞	∞	∞
2						
3						
4						
5						
6						

Minimaler Spannbaum:

b) Betrachten Sie den folgenden Graphen:



Führen Sie den *Algorithmus von Floyd* auf diesem Graphen aus. Geben Sie dazu nach jedem Durchlauf der äußeren Schleife die aktuellen Entfernungen in einer Tabelle an. Die erste Tabelle enthält bereits die Adjazenzmatrix nach Bildung der reflexiven Hülle. Der Eintrag in der Zeile *i* und Spalte *j* ist also ∞, falls es keine Kante vom Knoten der Zeile *i* zu dem Knoten der Spalte *j* gibt, und sonst das Gewicht dieser Kante. Beachten Sie, dass in der reflexiven Hülle jeder Knoten eine Kante mit Gewicht 0 zu sich selbst hat.

1	A	B	C	D
A	0	3	5	∞
B	7	0	1	∞
C	9	8	0	2
D	4	∞	6	0

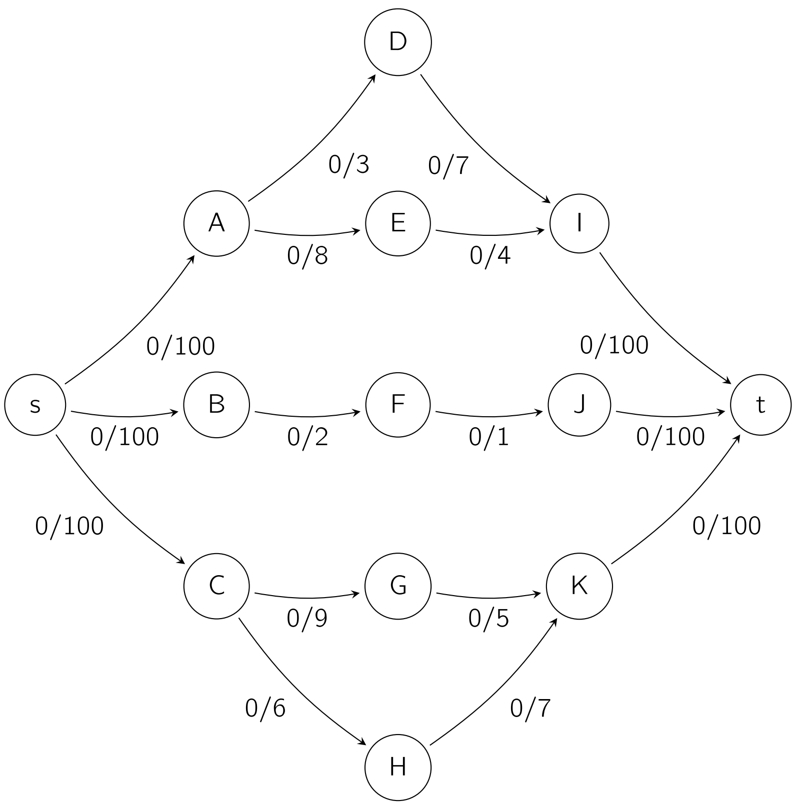
2	A	B	C	D
A				
B				
C				
D				

3	A	B	C	D
A				
B				
C				
D				

4	A	B	C	D
A				
B				
C				
D				

5	A	B	C	D
A				
B				
C				
D				

c) Betrachten Sie das folgende Flussnetzwerk mit Quelle s und Senke t:



Geben Sie einen minimalen Schnitt sowie den Wert des maximalen Flusses in diesem Flussnetzwerk an.

d) Gegeben ist eine endliche Menge von Servern S und eine Funktion $b\colon S \times S \to \mathbb{N}$. Für zwei Server $s_1, s_2 \in S$ gibt der Funktionswert $b(s_1, s_2)$ die Bandbreite an, mit der Daten vom Server s_1 *direkt* (d. h. ohne Weiterleitung über einen anderen Server) zum Server s_2 gesendet werden können (es gilt hierbei nicht zwingend $b(s_1, s_2) = b(s_2, s_1)$).

Sollen nun Daten von s_1 nach s_k gesendet werden, so kann es jedoch sein, dass die maximale Bandbreite entlang eines Pfades von Servern $s_1, \ldots, s_k$ höher ist als die Bandbreite $b(s_1, s_k)$ der direkten Verbindung. Da Bandbreiten Flaschenhälse darstellen, ist die maximale Bandbreite entlang des Pfades $s_1, \ldots, s_k$ durch das Minimum

$$\min\left\{b(s_1, s_2), b(s_2, s_3), \ldots, b(s_{k-1}, s_k)\right\}$$

gegeben.

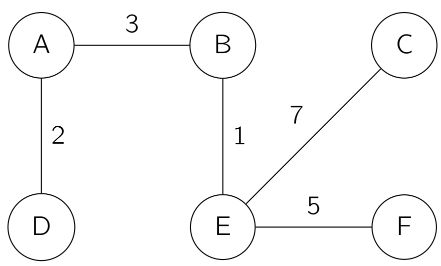
Entwerfen Sie (in Stichpunkten) einen Algorithmus, der zwischen allen geordneten Paaren von Servern die maximale Bandbreite ermittelt. Der Algorithmus soll dabei insgesamt in Zeit $\Theta\left(|S|^3\right)$ ausgeführt werden können, d. h. nach $\Theta\left(|S|^3\right)$ Schritten sollen zwischen *allen* Paaren die maximalen Bandbreiten errechnet worden sein. Der Algorithmus soll eine Abwandlung eines Ihnen aus der Vorlesung bekannten Algorithmus zur Berechnung *kürzester Pfade* sein. Geben Sie insbesondere an, welchen Algorithmus Sie abwandeln. Begründen Sie kurz, warum ihr Algorithmus das vorliegende Problem löst.

Lösung:

a)

#Iteration	A	B	C	D	E	F
1	<u>0</u>	∞	∞	∞	∞	∞
2		3	∞	<u>2</u>	4	∞
3		<u>3</u>	∞		4	∞
4			8		<u>1</u>	∞
5			7			<u>5</u>
6			<u>7</u>			

Hierbei gibt eine unterstrichene Zahl an, in welcher Iteration (zugehöriger Zeilenkopf) welcher Knoten (zugehöriger Spaltenkopf) durch `pq.getMin()` gewählt wurde. Wir erhalten den folgenden minimalen Spannbaum:



b)

1	A	B	C	D
A	0	3	5	∞
B	7	0	1	∞
C	9	8	0	2
D	4	∞	6	0

2	A	B	C	D
A	0	3	5	∞
B	7	0	1	∞
C	9	8	0	2
D	4	7	6	0

3	A	B	C	D
A	0	3	4	∞
B	7	0	1	∞
C	9	8	0	2
D	4	7	6	0

4	A	B	C	D
A	0	3	4	6
B	7	0	1	3
C	9	8	0	2
D	4	7	6	0

5	A	B	C	D
A	0	3	4	6
B	7	0	1	3
C	6	8	0	2
D	4	7	6	0

c) Ein minimaler Schnitt ist $(\{s, A, B, C, E, F, G\}, \{D, H, I, J, K, t\})$. Der Wert des maximalen Flusses ist 19.

d) Zur Lösung des Problems wird der Algorithmus von Floyd wie folgt abgewandelt:

- In der initialen Tabelle ($k = 0$):
  - In der initialen Tabelle werden auf der Diagonalen ∞-Symbole (statt 0-en) eingetragen, da die Bandbreite von einem Server zu sich selbst im Prinzip unbeschränkt ist – es muss ja keine Verbindung überbrückt werden.
  - Hat ein Server s_i eine direkte Verbindung zu einem Server s_j, so wird in Zeile i, Spalte j die Bandbreite der direkten Verbindung eingetragen.
  - Hat ein Server s_i *keine* direkte Verbindung zu einem Server s_j, so wird in Zeile i, Spalte j eine 0 (statt ∞) eingetragen, da bzgl. einer direkten Verbindung gar keine Bandbreite zur Verfügung steht.
- In den Iterationsschritten ($k > 0$): Es wird exakt wie beim Algorithmus von Floyd verfahren, allerdings wird statt der Rekursionsformel

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$

  die Rekursionsformel

$$d_{ij}^{(k)} = \max\left(d_{ij}^{(k-1)}, \min\left(d_{ik}^{(k-1)}, d_{kj}^{(k-1)}\right)\right)$$

  verwendet.

Letzteres ist deshalb korrekt, da statt der Länge des Pfades, die *maximale Weite* (daher der max-Operator) des *Flaschenhalses* (daher der min-Operator) zwischen zwei beliebigen Servern gesucht wird.

Aufgabe 7 (Dynamische Programmierung): (7 + 8 = 15 Punkte)

a) Gegeben sei ein Rucksack mit *maximaler Tragkraft* 7 sowie 4 *Gegenstände*. Der i-te Gegenstand soll hierbei ein Gewicht von w_i und einen Wert von c_i haben. Bestimmen Sie mit Hilfe des in der Vorlesung vorgestellten Algorithmus zum Lösen des Rucksackproblems mit dynamischer Programmierung den *maximalen Gesamtwert* der Gegenstände, die der Rucksack tragen kann (das Gesamtgewicht der mitgeführten Gegenstände übersteigt nicht die Tragkraft des Rucksacks). Die *Gewichte* seien dabei $w_1 = 4$, $w_2 = 5$, $w_3 = 3$ und $w_4 = 6$ und die *Werte* $c_1 = 3$, $c_2 = 4$, $c_3 = 2$ und $c_4 = 5$. Geben Sie zudem die vom Algorithmus bestimmte Tabelle `C` und die *mitzunehmenden Gegenstände* an.

Zur Erinnerung: Die Rekursionsgleichung für das Rucksackproblem lautet:

$$C[i,j] = \begin{cases} 0 & \text{für } i = 0, j \geq 0 \\ -\infty & \text{für } j < 0 \\ max(C[i-1,j], c_i + C[i-1, j-w_i]) & \text{sonst} \end{cases}$$

b) Um weiterhin hochqualitative Vorlesungsvideos anzubieten, will die Video AG Werbespots in die Vorlesungsvideos einbauen. Um die Aufmerksamkeit der Zuschauer nicht zu oft zu unterbrechen, ist entschieden worden, dass es genau zwei Werbeunterbrechungen in einem Vorlesungsvideo geben soll, deren Länge jeweils maximal k Zeiteinheiten beträgt. Der Video AG liegen Angebote für Werbespots aus der Menge $W = \{w_1, \ldots, w_n\}$ vor. Die Länge des Werbespots w_i ist durch ℓ_i Zeiteinheiten gegeben. Da die beiden Werbeblöcke als unterschiedlich wirksam erachtet werden, wird ein Preis $p_{i,1}$ erzielt, wenn der Werbespot w_i in der ersten Werbepause gesendet wird, und ein Preis $p_{i,2}$, wenn der Werbespot in der zweiten Werbepause gesendet wird. Jeder Werbespot darf aber insgesamt höchstens einmal während eines Vorlesungsvideos gesendet werden. Außerdem dürfen Werbespots nur in voller Länge gesendet werden (eine teilweise Sendung ist also nicht erlaubt).

Helfen Sie der Video AG, den Gewinn durch die Werbesendungen zu maximieren.

Gehen Sie dazu wie folgt vor:

(i) Stellen Sie eine Rekursionsgleichung $G(i, t_1, t_2)$ auf, die angibt, welcher Gewinn maximal erzielt werden kann, wenn die noch verfügbaren Zeiten der beiden Werbeblöcke t_1 bzw. t_2 Zeiteinheiten betragen und nur die Werbespots $W' = \{w_1, \ldots, w_i\}$ zur Verfügung stehen.

(ii) Geben Sie an, für welche Werte i, t_1, t_2 der Wert von $G(i, t_1, t_2)$ das gesuchte Ergebnis (also die maximalen Werbeeinnahmen unter den oben beschriebenen Randbedingungen) liefert. Anders formuliert: Welcher "Aufruf" der Rekursionsgleichung liefert das gewünschte Ergebnis?

Lösung:

a) Die Tabelle `C` wird vom Algorithmus wie folgt gefüllt:

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	$\uparrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 3	3	3	3
2	0	0	0	0	$\uparrow$ 3	4	4	4
3	0	0	0	2	$\uparrow$ 3	$\leftarrow$ 4	$\leftarrow$ 4	$\leftarrow$ 5
4	0	0	0	2	3	4	5	$\uparrow$ 5

Damit ergibt sich der maximale Wert 5 für den Fall, dass der 1. und 3. Gegenstand mitgenommen werden.

b) Die gesuchte Rekursionsgleichung ist gegeben durch

$$G(i, t_1, t_2) = \begin{cases} 0 & \text{wenn } i = 0 \land t_1 \geq 0 \land t_2 \geq 0 \\ -\infty & \text{wenn } t_1 < 0 \lor t_2 < 0 \\ \max\{G(i-1, t_1, t_2), & \text{sonst} \\ p_{i,1} + G(i-1, t_1 - \ell_i, t_2), & \\ p_{i,2} + G(i-1, t_1, t_2 - \ell_i)\} & \end{cases}$$

wobei der "Aufruf" $G(n, k, k)$ das gewünschte Ergebnis liefert.