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.

- $\sum_{i=0}^{5}\left(i^2 \cdot n^i\right) \in \_\_\_\_\left(n^7\right)$
- $n^7 \in \_\_\_\_\left(\frac{1}{20}n^7 + 5000 \cdot n^2\right)$
- $\frac{n^2}{\log_8(n)} \in \_\_\_\_\left(n^2\right)$
- $n \in \_\_\_\_\left(2^{12978313297}\right)$
- $\sum_{i=0}^{7}\left(n^i\right) \in \_\_\_\_\left(n^8\right)$
- $n! \in \_\_\_\_\left(n^n\right)$
- $3^n \in \_\_\_\_\left(4^n\right)$
- $\sum_{i=0}^{\sqrt{n}} i \in \_\_\_\_\left(n\right)$
- $\sqrt{n} \in \_\_\_\_\left(\log_2(n)\right)$
- $\sum_{i=1}^{n}\left(\frac{i^2+1}{i}\right) \in \_\_\_\_\left(n^2\right)$

b) Beweisen oder widerlegen Sie, dass für Funktionen $f\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ und $g\colon \mathbb{N} \to \mathbb{R}_{\geq 0}$ gilt:

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

Lösung:

a)

- $\sum_{i=0}^{5}\left(i^2 \cdot n^i\right) \in \_o\_\left(n^7\right)$
- $n^7 \in \_\Theta\_\left(\frac{1}{20}n^7 + 5000 \cdot n^2\right)$
- $\frac{n^2}{\log_8(n)} \in \_o\_\left(n^2\right)$
- $n \in \_\omega\_\left(2^{12978313297}\right)$
- $\sum_{i=0}^{7}\left(n^i\right) \in \_o\_\left(n^8\right)$

- $n! \in \underline{\quad o \quad}(n^n)$
- $3^n \in \underline{\quad o \quad}(4^n)$
- $\sum_{i=0}^{\sqrt{n}} i \in \underline{\quad \Theta \quad}(n)$
- $\sqrt{n} \in \underline{\quad \omega \quad}(\log_2(n))$
- $\sum_{i=1}^{n} \left(\frac{i^2+1}{i} \right) \in \underline{\quad \Theta \quad}(n^2)$

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

Beweis:
Gegenbeispiel: Sei $f(n) = 0$ und $g(n) = n$. Dann gilt $f(n) + g(n) = 0 + n = n$ und $f(n) \cdot g(n) = 0 \cdot n = 0$. Es gibt aber offensichtlich kein c und n_0, sodass für alle $n \geq n_0$ gilt:

$$f(n) + g(n) = n \leq 0 = c \cdot 0 = c \cdot f(n) \cdot g(n)$$

□

Aufgabe 2 (Rekursionsgleichungen): (10 + 8 = 18 Punkte)

a) Geben Sie für das Programm

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

    int value = func(n);

    int k = 7;
    while (k >= 5) {
        value = 4 * berechne(n/k);
        k = k - 1;
    }

    return value;
}

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

    while (n > 1) {
        res = res + 1;
        n = n/2;
    }

    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

$$T(n) = 8 \cdot T\left(\frac{n}{4}\right) + 5 \cdot n + \sqrt{n}$$

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

Lösung:

a) `func(n)` hat logarithmische Laufzeit im Parameter n, da in der `while`-Schleife der aktuelle Wert von n immer wieder durch zwei geteilt wird (was größenordnungsmäßig logarithmisch oft möglich ist, bis 1 erreicht wird).

Jeder Aufruf von `berechne(n)` führt nacheinander zu den Aufrufen `berechne(n/7)`, `berechne(n/6)` und `berechne(n/5)`, da in der `while`-Schleife `k` die Werte 7, 6 und 5 annimmt. Somit ergibt sich die folgende Rekursionsgleichung für die asymptotische Laufzeit von `berechne`(n):

$$T(n) \;=\; T\left(\frac{n}{7}\right) + T\left(\frac{n}{6}\right) + T\left(\frac{n}{5}\right) + \log(n) \qquad T(0) = T(1) = 1$$

b) Es sind $b = 8$, $c = 4$ und $f(n) = 5 \cdot n + \sqrt{n}$. E wird nun wie folgt bestimmt:

$$E \;=\; \frac{\log(8)}{\log(4)} \;=\; \frac{3}{2} \;=\; 1.5$$

Damit gilt $n^E = n^{1.5}$. Wähle nun $\varepsilon = 1.5 - 1 = 0.5$. Wir bestimmen nun den folgenden Grenzwert:

$$\begin{aligned} & \lim_{n\to\infty} \frac{f(n)}{n^{E-\varepsilon}} \\ = & \lim_{n\to\infty} \frac{5 \cdot n + \sqrt{n}}{n^{1.5-0.5}} \\ = & \lim_{n\to\infty} \frac{5 \cdot n + \sqrt{n}}{n} \\ = & \lim_{n\to\infty} 5 + \frac{1}{\sqrt{n}} \\ = & \ 5 + 0 \ = \ 5 \end{aligned}$$

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

$$T(n) \in \Theta\left(n^{1.5}\right) \ .$$

Aufgabe 3 (Sortieren): (4 + 6 + 5 = 15 Punkte)

a) Sortieren Sie das folgende Array mithilfe von Mergesort. Geben Sie dazu das Array nach jeder Merge-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

7	4	3	1	6	8	5	2	9

b) Sortieren Sie das folgende Array mithilfe von Heapsort. Geben Sie dazu das Array nach jeder Swap-Operation an. Die vorgegebene Anzahl an Zeilen muss nicht mit der benötigten Anzahl an Zeilen übereinstimmen.

3	6	7	4	1	8

c) Eine Firma hat ein Fließband, auf welchem eine Folge von Gegenständen vorwärts und rückwärts befördert werden kann. Jeder Gegenstand ist mit einem Barcode versehen, welcher eine Zahl repräsentiert. An einer Stelle des Fließbandes befindet sich eine Maschine, welche zwei sich hintereinander an dieser Stelle auf dem Fließband befindliche Gegenstände bzgl. der durch ihren Barcode repräsentierten Zahl miteinander vergleichen und, falls sich die Gegenstände nicht in richtiger Reihenfolge befinden, die Position der beiden Gegenstände auf dem Fließband miteinander vertauschen kann. Die ausgelesenen Zahlen kann die Maschine jedoch nicht nach außen kommunizieren (d. h. der Maschine kann nur ein Signal gegeben werden, ihre Vergleichs- und Tauschoperation durchzuführen, aber die Maschine sendet keine andere Information zurück als diejenige, dass sie ihre Operation beendet hat).

Die Firma möchte eine Steuerung des Fließbandes und der Maschine einbauen, welche das Fließband jeweils einen Gegenstand vorwärts bzw. rückwärts bewegen und die Maschine veranlassen kann, ihre Vergleichs- und Tauschoperation durchzuführen. Die Steuerung soll dafür sorgen, dass die Folge der Gegenstände sortiert wird.

Welches Sortierverfahren aus der Vorlesung würden Sie der Steuerung zugrunde legen? Begründen Sie Ihre Antwort kurz (d. h. mit nicht mehr als 50 Worten).

Lösung:

a)

7	4	3	1	6	8	5	2	9
4	7	3	1	6	8	5	2	9
3	4	7	1	6	8	5	2	9
3	4	7	1	6	8	5	2	9
1	3	4	6	7	8	5	2	9
1	3	4	6	7	5	8	2	9
1	3	4	6	7	5	8	2	9
1	3	4	6	7	2	5	8	9
1	2	3	4	5	6	7	8	9

b)

3	6	7	4	1	8
3	6	8	4	1	7
8	6	3	4	1	7
8	6	7	4	1	3
3	6	7	4	1	8
7	6	3	4	1	8
1	6	3	4	7	8
6	1	3	4	7	8
6	4	3	1	7	8
1	4	3	6	7	8
4	1	3	6	7	8
3	1	4	6	7	8
1	3	4	6	7	8

c) Für dieses Szenario eignet sich nur Bubblesort, da es das einzige Verfahren aus der Vorlesung ist, welches lediglich Vergleiche und Tauschoperationen auf benachbarten Elementen benötigt.

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

a) Fügen Sie die folgenden Werte in das unten stehende Array a der Länge 11 unter Verwendung der *Divisionsmethode* mit *linearer Sondierung* ein:

4, 17, 15, 27, 26, 14.

b) Fügen Sie die folgenden Werte in das unten stehende Array a der Länge 11 unter Verwendung der *Divisionsmethode* mit *quadratischer Sondierung* ($c_1 = 0.0$, $c_2 = 1.0$) ein:

6, 29, 17, 9, 20, 28.

Lösung:

a) m = 11:

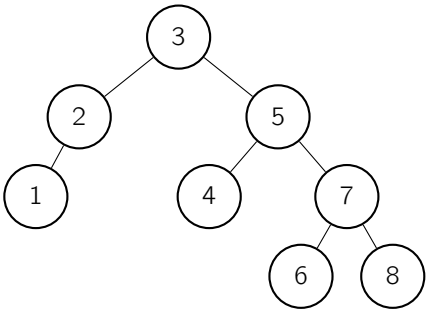
			14	4	15	17	27	26		

b) m = 11, c_1 = 0.0, c_2 = 1.0:

		20		28		6	29		9	17

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

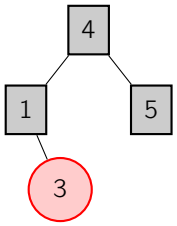
a) Löschen Sie den Wert 3 aus dem folgenden *AVL-Baum* und geben Sie die entstehenden Bäume nach jeder *Löschoperation* sowie jeder *Rotation* an. Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird:



b) Fügen Sie den Wert 2 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.



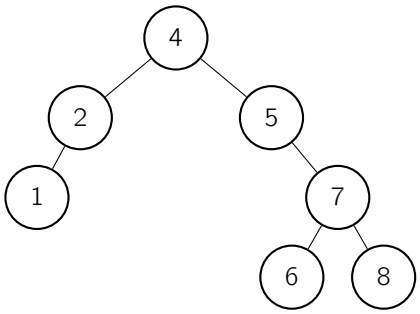
c) Zeigen Sie per Induktion über h: Hat ein Knoten v eines Rot-Schwarz-Baums die Höhe h, so besitzt der Teilbaum mit Wurzel v mindestens $2^{bh(v)} - 1$ innere Knoten.

Hinweise:

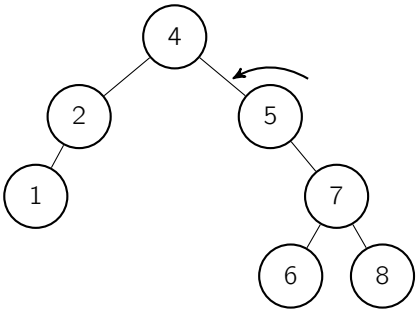
- *$bh(v)$ bezeichnet die* **Schwarzhöhe** *des Knoten v, also die Anzahl von schwarzen Knoten auf einem Pfad von v zu einem externen Blatt, wobei der Knoten v* **nicht mitzählt**.
- *Die Höhe $h(v)$ eines Knoten v ist Anzahl der Kanten des längsten Pfades von v bis zu einem externen Blatt.*
- *Innere Knoten sind Knoten des Baumes, die keine externen Blätter sind.*
- *Es ist hilfreich, bei der Induktionsannahme davon auszugehen, dass die Aussage für alle Höhen h' mit $0 \leq h' < h$ gilt.*

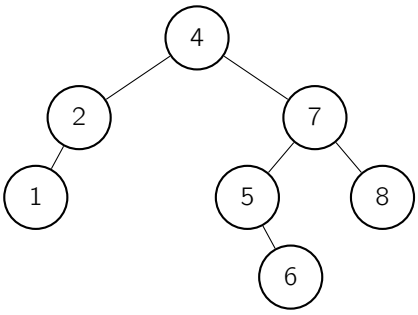
Lösung:

a) entferne 3

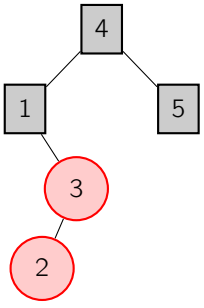


rotiere 5 nach links

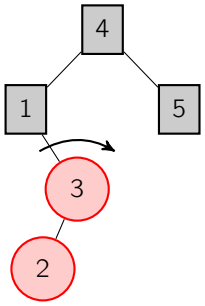


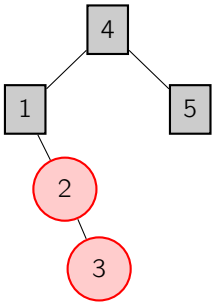


b) füge 2 ein

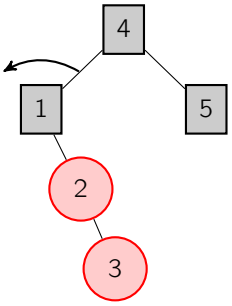


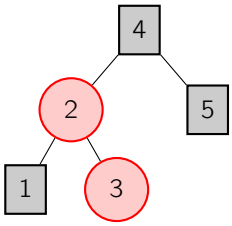
Fall 2 (Onkel ist null und damit schwarz und aktueller Knoten liegt innen): rotiere 3 nach rechts



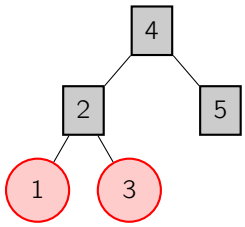


Fall 3 (Onkel ist null und damit schwarz und aktueller Knoten liegt außen): rotiere 1 nach links





Fall 3 (Fortsetzung): umfärben



c)

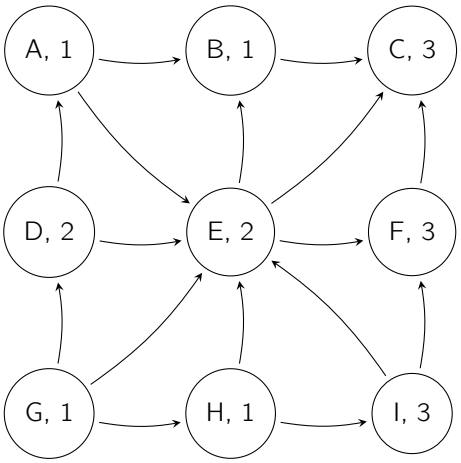
- Induktionsanfang: $h = 0$: Die einzigen Knoten v mit Höhe 0 sind die externen Knoten. Damit ist die Anzahl der inneren Knoten des Teilbaums 0 und ebenso ist $2^{bh(v)} - 1 = 2^0 - 1 = 0$.
- Induktionshypothese: Für alle Knoten v' eines Rot-Schwarz-Baums mit Höhe $0 \leq h(v') < h$, hat der Teilbaum mit Wurzel v' mindestens $2^{bh(v')} - 1$ innere Knoten.
- Induktionsschritt: Betrachten wir einen Knoten v mit Höhe $h > 0$ und seine Nachfolger v_ℓ und v_r. Ist v_ℓ rot, dann gilt $bh(v_\ell) = bh(v)$ und sonst $bh(v_\ell) = bh(v) - 1$. Analoges gilt für v_r. Wegen $h(v_\ell) < h$ und $h(v_r) < h$ ist nach Induktionsvoraussetzung die Anzahl der inneren Knoten in den beiden Teilbäumen von v also jeweils mindestens $2^{bh(v)-1} - 1$. Da v ein innerer Knoten ist, hat der Teilbaum mit Wurzel v mindestens

$$2 \cdot \underbrace{(2^{bh(v)-1} - 1)}_{\text{je Teilbaum}} + \underbrace{1}_{v} = 2^{bh(v)} - 2 + 1 = 2^{bh(v)} - 1$$

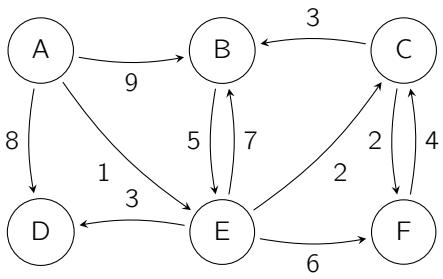
  innere Knoten.

Aufgabe 6 (Graphen): (5 + 5 + 10 = 20 Punkte)

a) Bestimmen Sie eine *topologische Sortierung* unter Verwendung des in der Vorlesung vorgestellten Algorithmus für den folgenden Graphen. Die Knoten in diesem Graphen sind mit jeweils einem Schlüssel und einer Dauer beschriftet. Im gesamten Algorithmus werden Knoten in aufsteigender alphabetischer Reihenfolge ihrer Schlüssel berücksichtigt. Geben Sie als Ergebnis die Liste der Knotenschlüssel zusammen mit ihrem jeweiligen *frühesten Endzeitpunkt (eft)* in aufsteigender Reihenfolge der Topologiewerte an. Markieren Sie außerdem einen *kritischen Pfad*, indem Sie die zugehörigen Knotenschlüssel unterstreichen, und geben Sie den gesamten *frühesten Endzeitpunkt (eft)* an.



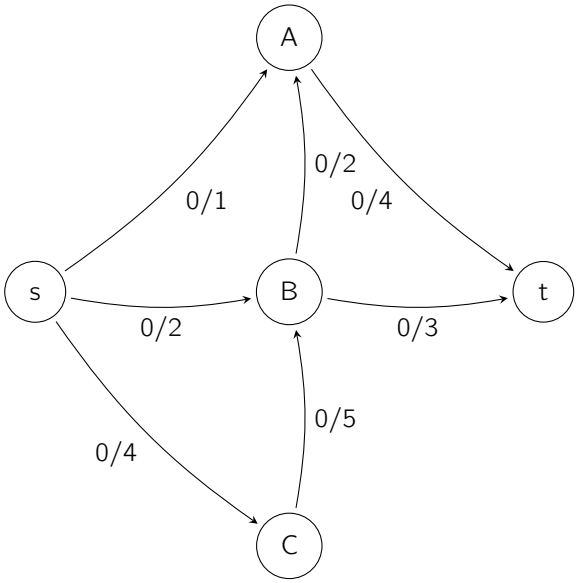
b) Betrachten Sie den folgenden Graphen:



Führen Sie den *Dijkstra* Algorithmus auf diesem Graphen mit dem *Startknoten A* aus. Falls mehrere Knoten für die nächste Iteration zur Wahl stehen, werden die Knoten dabei in alphabetischer Reihenfolge betrachtet. Füllen Sie dazu die nachfolgende Tabelle aus:

Knoten	A				
B					
C					
D					
E					
F					

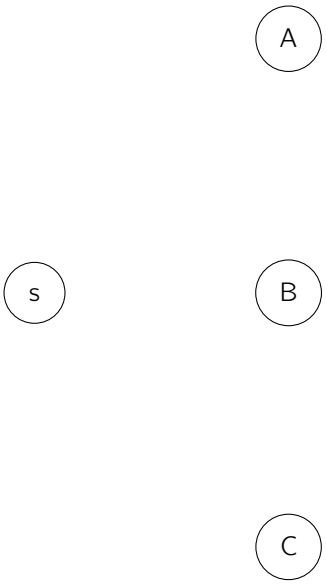
c) Betrachten Sie das folgende Flussnetzwerk mit Quelle s und Senke t:



Berechnen Sie den maximalen Fluss in diesem Netzwerk mithilfe der *Ford-Fulkerson Methode*. Geben Sie dazu *jedes Restnetzwerk (auch das initiale)* sowie *nach jeder Augmentierung* den aktuellen Zustand des Flussnetzwerks an. Geben Sie außerdem den *Wert des maximalen Flusses* an. Die vorgegebene Anzahl an Lösungsschritten muss nicht mit der benötigten Anzahl solcher Schritte übereinstimmen.

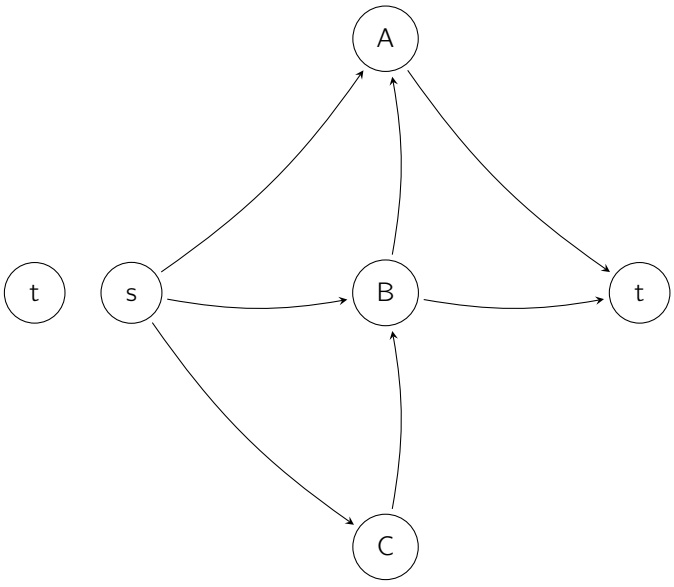
Schritt 1:

Restnetzwerk:



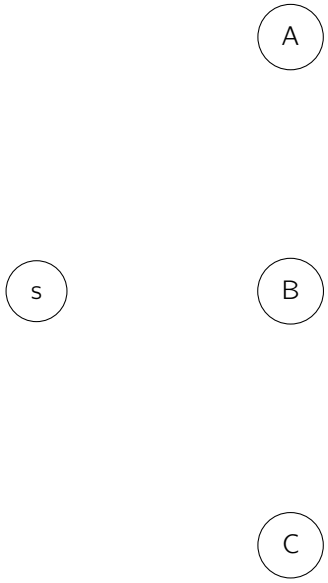
Schritt 2:

Nächstes Flussnetzwerk mit aktuellem Fluss:



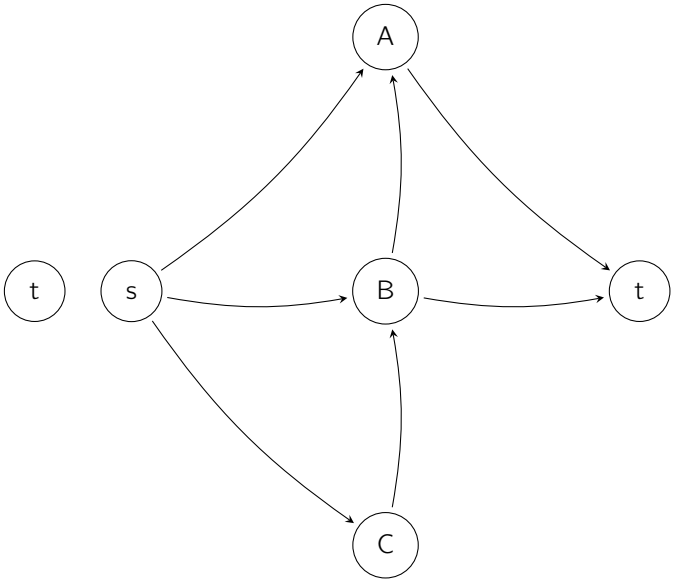
Schritt 3:

Restnetzwerk:



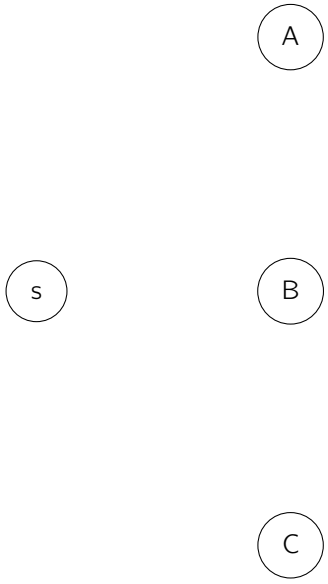
Schritt 4:

Nächstes Flussnetzwerk mit aktuellem Fluss:



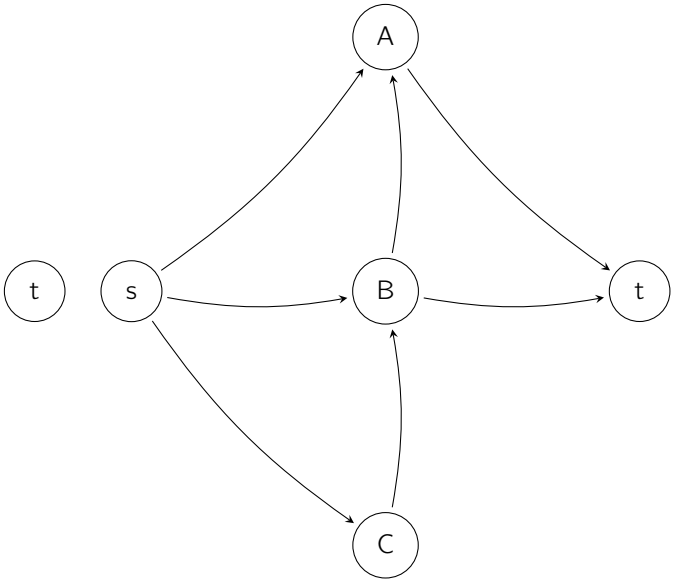
Schritt 5:

Restnetzwerk:



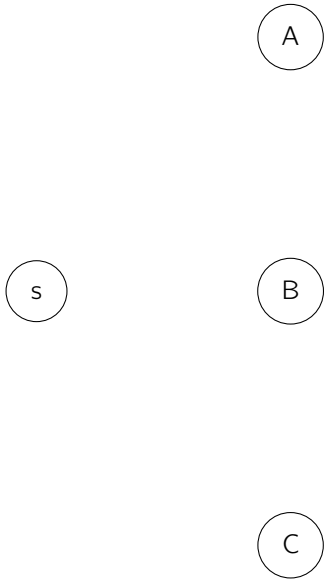
Schritt 6:

Nächstes Flussnetzwerk mit aktuellem Fluss:



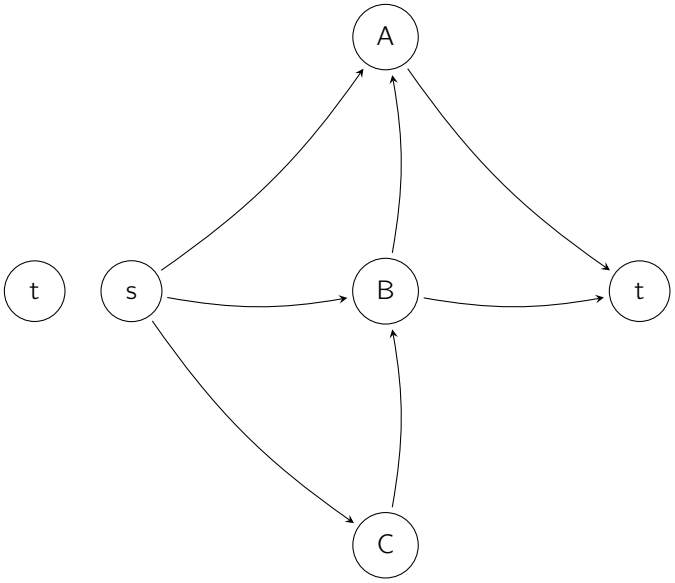
Schritt 7:

Restnetzwerk:



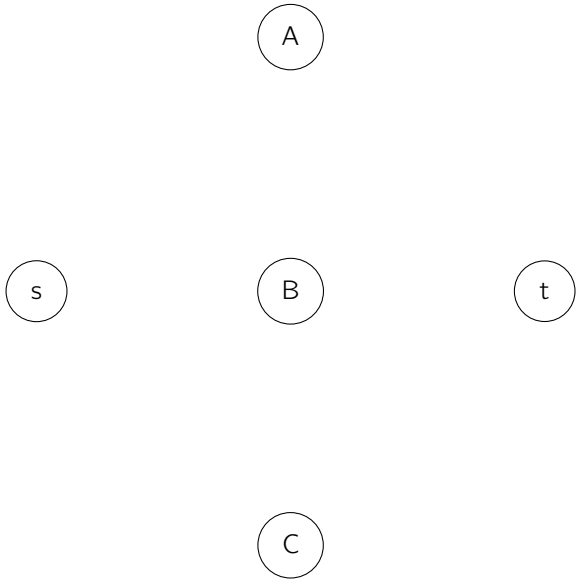
Schritt 8:

Nächstes Flussnetzwerk mit aktuellem Fluss:



Schritt 9:

Restnetzwerk:



Der maximale Fluss hat den Wert:

Lösung:

a) Der gegebene Graph hat die folgende topologische Sortierung:

C (3), B (4), F (6), E (8), A (9), D (11), I (11), H (12), G (13)
eft: 13

b)

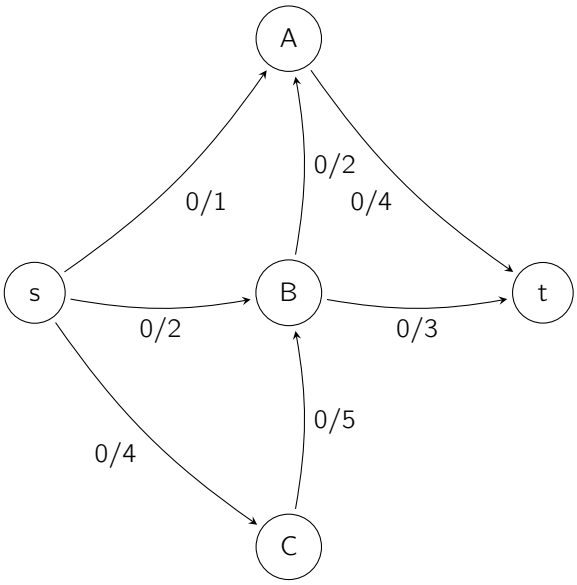
Knoten	A	E	C	D	F
B	9	8	6	6	6
C	∞	3	–	–	–
D	8	4	4	–	–
E	1	–	–	–	–
F	∞	7	5	5	–

Die grau unterlegten Zellen markieren, an welcher Stelle für welchen Knoten die minimale Distanz sicher berechnet worden ist.

Schritt 0:

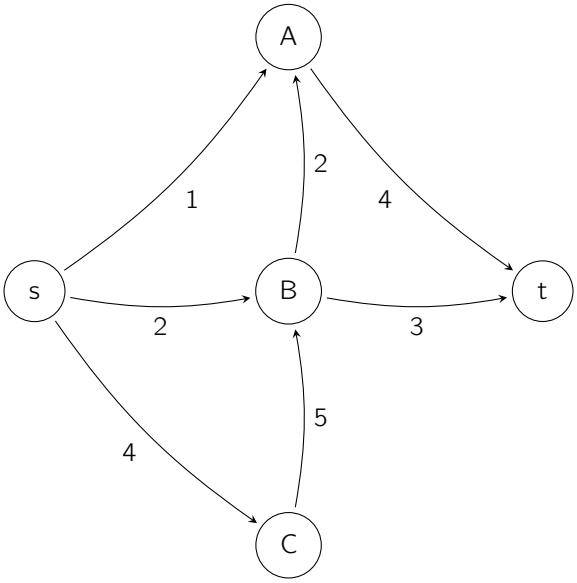
c)

Initiales Flussnetzwerk:



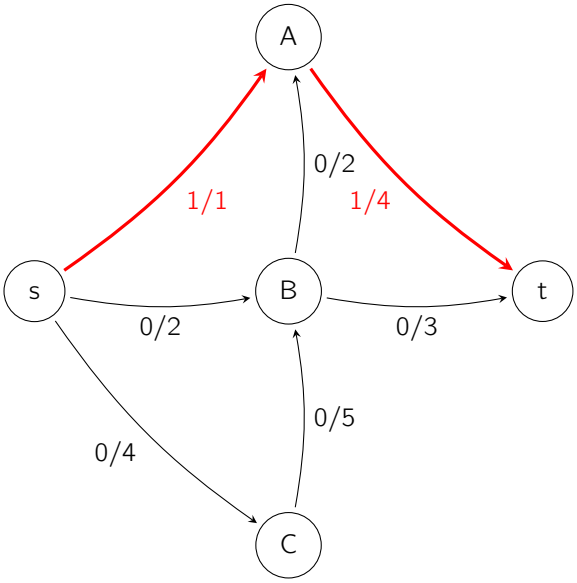
Schritt 1:

Restnetzwerk:



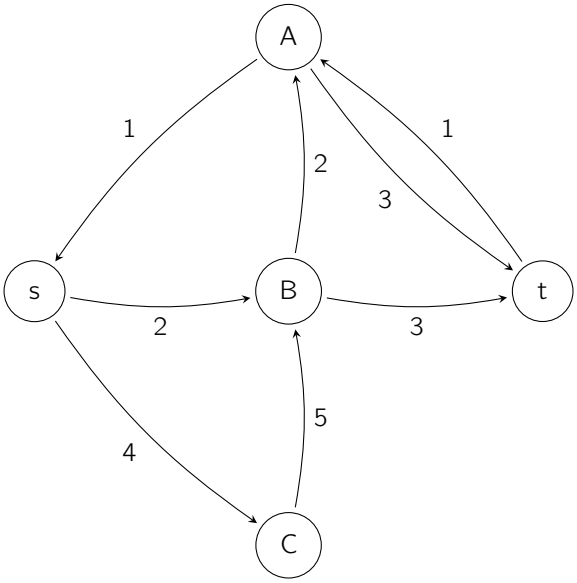
Schritt 2:

Nächstes Flussnetzwerk mit aktuellem Fluss:



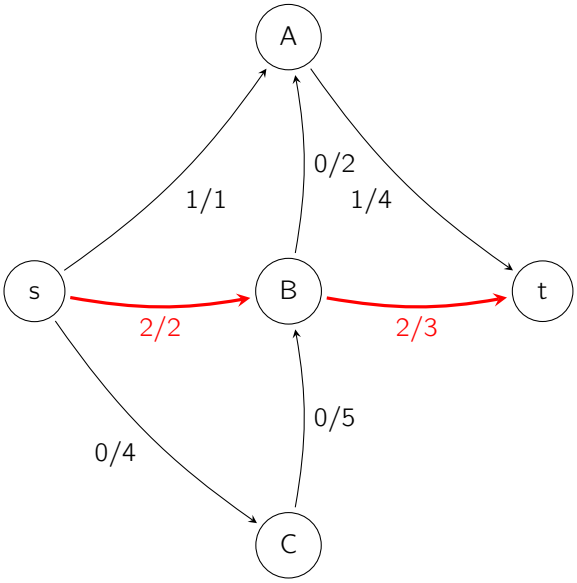
Schritt 3:

Restnetzwerk:



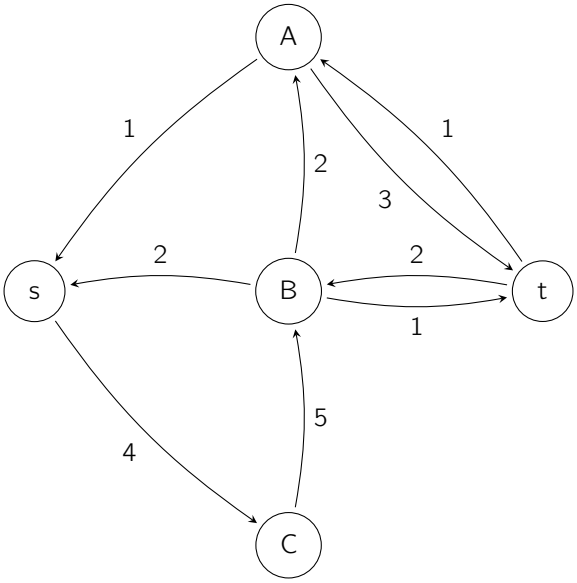
Schritt 4:

Nächstes Flussnetzwerk mit aktuellem Fluss:



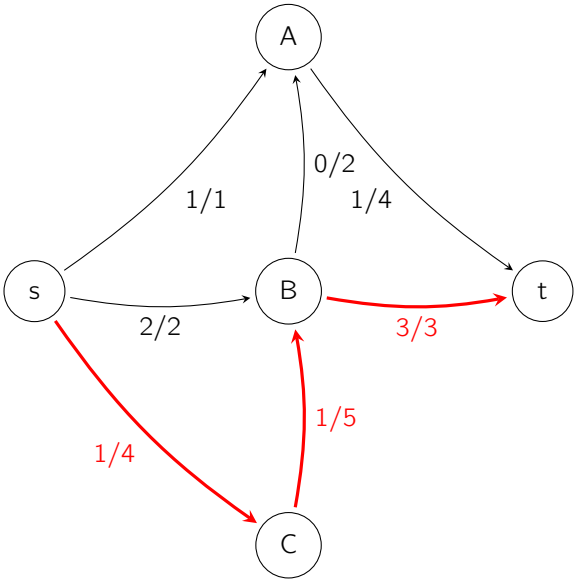
Schritt 5:

Restnetzwerk:



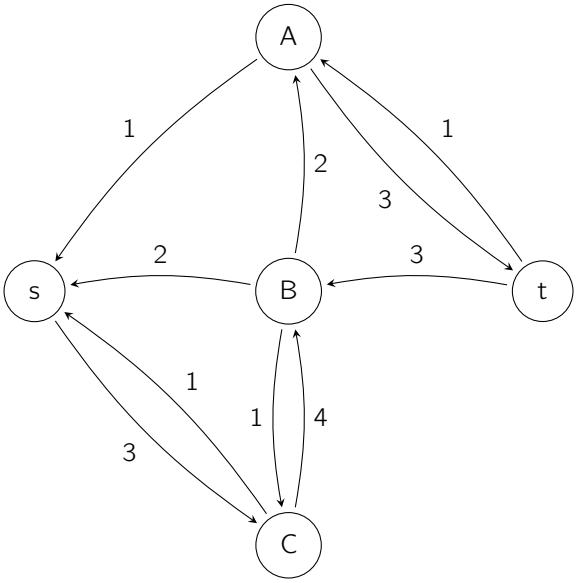
Schritt 6:

Nächstes Flussnetzwerk mit aktuellem Fluss:



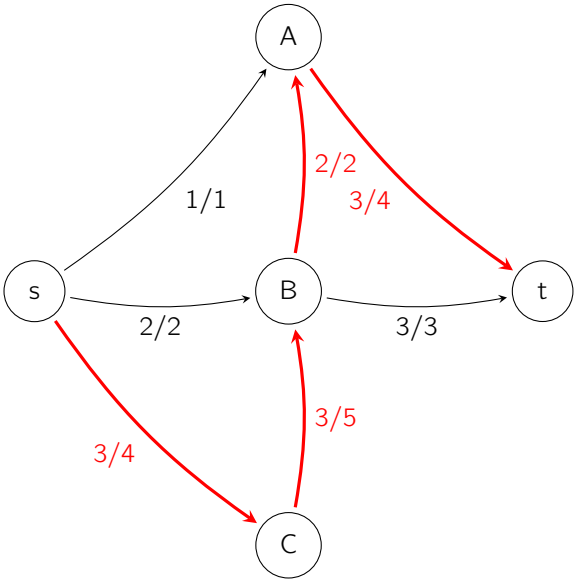
Schritt 7:

Restnetzwerk:



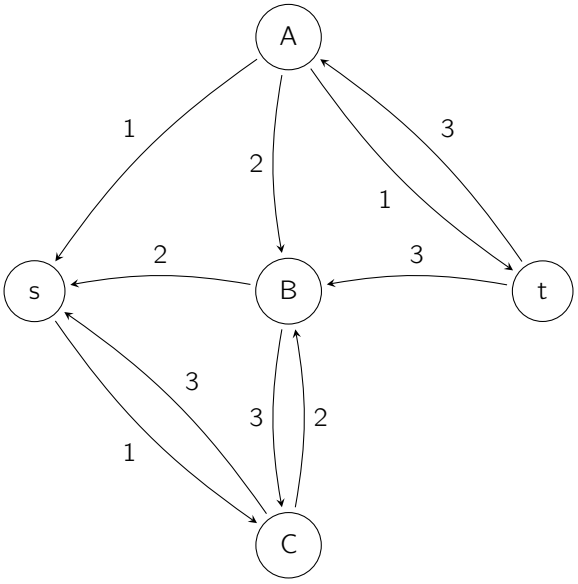
Schritt 8:

Nächstes Flussnetzwerk mit aktuellem Fluss:



Schritt 9:

Restnetzwerk:



Der maximale Fluss hat den Wert: 6

Aufgabe 7 (Algorithmenentwurf): (2 + 8 = 10 Punkte)

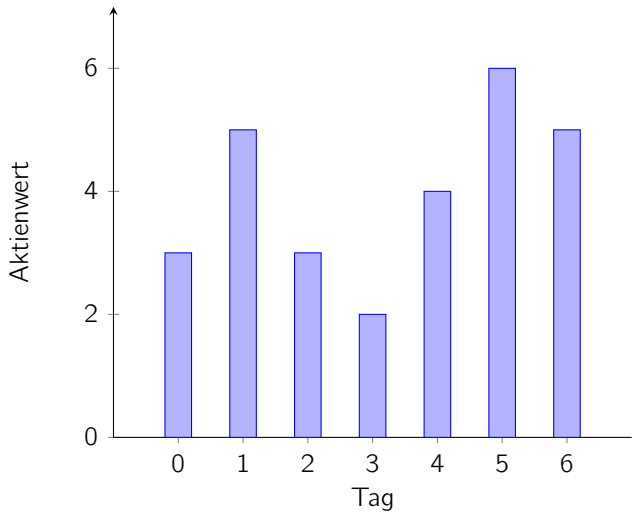
Gegeben seien die Werte einer Aktie an n aufeinanderfolgenden Tagen $\{0, \ldots, n-1\}$ in Form eines textttint Arrays E der Länge n. Steht also in $E[i]$ ein Wert v, so hatte die Aktie an Tag i den Wert v. Der *Aktienspann* eines Tages i ist die maximale Anzahl der **direkt aufeinanderfolgenden, vorhergehenden** Tage (inklusive des aktuellen Tags), an dem der Aktienwert **höchstens so hoch** wie am Tag i war. Formal ist der Aktienspann eines Tages $i \in \{0, \ldots, n-1\}$ also die maximale Anzahl s_i der Tage $t_1, \ldots, t_{s_i}$ mit $t_j \in \{0, \ldots, n-1\}$ für $j \in \{1, \ldots, s_i\}$, für die gilt:

- $t_{s_i} = i$ (d. h. der letzte Tag der Folge $t_1, \ldots, t_{s_i}$ ist der Tag i selbst)
- $t_j + 1 = t_{j+1}$ für $j \in \{1, \ldots, s_i - 1\}$ (d. h. die Tage in $t_1, \ldots, t_{s_i}$ folgen ohne Unterbrechung direkt aufeinander)
- $E[t_j] \leq E[i]$ für $j \in \{1, \ldots, s_i\}$ (d. h. an jedem Tag dieser Folge hatte die Aktie einen höchstens so hohen Wert wie am Tag i)

a) Betrachten Sie das folgende Eingabearray E, das die Werte einer Aktie an 7 aufeinanderfolgenden Tagen wiedergibt:

$$[3, 5, 3, 2, 4, 6, 5]$$

Eine graphische Repräsentation dieser Werte sieht wie folgt aus:



Berechnen Sie für alle Tage den Aktienspann und tragen Sie ihn in das nachfolgende Array ein, sodass an Index i der Aktienspann des Tages i steht. Die Lösung für Tag 0 ist bereits vorgegeben.

1						

b) Geben Sie einen Algorithmus in Pseudocode an, der zu einem gegebenen Eingabearray E der Länge n von Aktienwerten ein Ausgabearray S berechnet, sodass an $S[i]$ der Aktienspann des Tages i steht. Die Laufzeit Ihres Algorithmus soll dabei eine Worst-Case-Komplexität in $\mathcal{O}(n)$ besitzen, wobei für die Komplexität arithmetische Operationen, Vergleiche und Speicherzugriffe zu berücksichtigen sind. Begründen Sie kurz (maximal 70 Worte), wieso ihr Algorithmus die Laufzeitschranke einhält.

Hinweise: Es kann hierbei hilfreich sein,

- *von* **rechts nach links** *über das Array zu iterieren (wobei bereits eine einzige Iteration ausreicht) sowie*
- *einen* **Stack** *zu benutzen. Zur Erinnerung: der ADT* Stack *bietet die folgenden Operationen:*
  - `bool isEmpty(Stack s)` *(liefert* `true` *gdw. der Stack* `s` *leer ist)*
  - `void push(Stack s, Element e)` *(legt das Element* `e` *oben auf den Stack* `s`*)*

– *`Element pop(Stack s)` (entfernt das oberste Element des nicht-leeren Stacks `s` und liefert dieses zurück)*

Zusätzlich dürfen Sie davon ausgehen, dass der ADT Stack *die folgende Operation zur Verfügung stellt:*

– *`Element peek(Stack s)` (liefert das oberste Element des nicht-leeren Stacks `s` zurück, ohne es von `s` zu entfernen)*

Alle diese Operationen besitzen eine Worst-Case-Laufzeit in $\mathcal{O}(1)$.

Lösung:

a) Die Aktienspänne lauten wie folgt:

1	2	1	1	3	6	1

b)
```
void aktienspann(int E[], int S[], int n) {
    Stack<int> positionStack;
        for (int j = n - 1; j >= 0; j--) {
            while (!isEmpty(positionStack)) {
                if (E[peek(positionStack)] < E[j]) {
                    int topElement = pop(positionStack);
                    S[topElement] = topElement - j;
                }
            }
        push(positionStack, j);
    }

    // Behandle alle Tage an denen der Wert hoeher
    // war als an allen vorherigen Tagen.
    while (!isEmpty(positionStack)) {
        topElement = pop(positionStack);
        S[topElement] = topElement + 1;
    }
}
```

Die
textttfor-Schleife wird n-mal durchlaufen. Die beiden
textttwhile-Schleifen werden insgesamt ebenfalls n-mal durchlaufen, da jedes Element genau einmal auf den Stack gelegt und wieder davon entfernt wird. Da ansonsten alle Schleifen nur konstant viele Anweisungen enthalten, deren Laufzeit jeweils in $\mathcal{O}(1)$ ist, ergibt sich eine Gesamtlaufzeit in $\mathcal{O}(n)$.

Aufgabe 8 (Dynamische Programmierung): (7 + 6 = 13 Punkte)

a) Bestimmen Sie die *längste gemeinsame Teilsequenz* der Sequenzen `ACKER` und `AKTE`. Benutzen Sie hierfür den in der Vorlesung vorgestellten Algorithmus mit dynamischer Programmierung und füllen Sie die folgende Tabelle aus. Eine leere Sequenz können Sie durch - angeben.

b) Eine CD hat eine verfügbare Laufzeit von $k \in \mathbb{N}_{>0}$ Minuten. Außerdem haben wir eine Liste $S = [s_1, \ldots, s_n]$ von Songlaufzeiten in Minuten (der i-te Song hat also eine Laufzeit von exakt $s_i \in \mathbb{N}_{>0}$ Minuten). Wir möchten nun eine Auswahl der Songs auf die CD brennen, sodass die Laufzeit der CD möglichst gut ausgenutzt wird (d. h. die ungenutzte Restzeit soll minimiert werden ohne die verfügbare Laufzeit zu überschreiten) und jeder Song höchstens einmal auf der CD enthalten ist (während gleiche Songlaufzeiten durchaus mehrfach in S auftreten können, sind die Songs, die dieser Liste zugrunde liegen, paarweise verschieden).

Beispiel: Sei $k = 10$ und $S = [3, 4, 6, 5]$ die Liste der Songlaufzeiten. Brennt man die Songs mit den Laufzeiten 3 und 4 auf die CD, so erhält man eine Restlaufzeit von 3 Minuten, welche nicht weiter verwendet werden kann. Brennt man hingegen die Songs mit den Laufzeiten 4 und 6 auf die CD, so bleibt keine Restlaufzeit übrig. Dies wäre eine optimale Lösung für das gegebene Problem (es kann mehrere optimale Lösungen für eine Probleminstanz geben, wobei die entsprechende Restlaufzeit natürlich gleich sein muss).

Geben Sie eine Rekursionsgleichung für das Teilproblem $C(i, \ell)$ an, wobei $C(i, \ell)$ die minimale Restlaufzeit für eine CD mit verfügbarer Laufzeit $0 \leq \ell \leq k$ und den Songs mit Laufzeiten $s_1, \ldots, s_i$ ist, wobei $0 \leq i \leq n$ gilt (bei $i = 0$ ist die Liste der Songlaufzeiten leer).

Lösung:

a) Die Tabelle wird vom Algorithmus wie folgt gefüllt:

	∅	A	K	T	E
∅	0	0	0	0	0
A	0	↖ 1	1	1	1
C	0	↑ 1	1	1	1
K	0	1	↖ 2	← 2	2
E	0	1	2	2	↖ 3
R	0	1	2	2	↑ 3

Längste gemeinsame Teilsequenz: `AKE`

b)

$$C(i,\ell) = \begin{cases} \ell & \text{für } i = 0, \\ min\{C(i-1,\ell), C(i-1,\ell-s_i)\} & \text{für } i > 0, \ell \geq s_i, \\ C(i-1,\ell) & \text{sonst.} \end{cases}$$