Aufgabe 1 (Laufzeitanalyse): (5 + 7 + 5 + 8 = 25 Punkte)

a) Beweisen oder widerlegen Sie: $\frac{n^2 \cdot (\sqrt{n})^3}{n^3+1} \in \Omega(n)$

b) Beweisen oder widerlegen Sie: $\Pi_{i=1}^{n}(i+n) \in \mathcal{O}(n^{2n})$

c) Bestimmen Sie die Komplexitätsklasse der folgenden Rekursionsgleichung mithilfe des **Mastertheorems** oder begründen Sie, warum das Mastertheorem nicht anwendbar ist.

$$T(n) = \begin{cases} 16 \cdot T(\frac{n}{4}) + \sqrt{n^3} & \text{für } n > 1 \\ 5 & \text{sonst} \end{cases}$$

Erinnerung Mastertheorem: $E = \log_b(a)$ (alternativ $b^E = a$) und

$f(n) \in \mathcal{O}(n^{E-\epsilon}), \epsilon > 0$	$\rightarrow T(n) \in \Theta(n^E)$
$f(n) \in \Theta(n^E)$	$\rightarrow T(n) \in \Theta(n^E \cdot \log(n))$
$f(n) \in \Omega(n^{E+\epsilon}), \epsilon > 0$ und $a \cdot f(n/b) \leq d \cdot f(n), d < 1$	$\rightarrow T(n) \in \Theta(f(n))$

d) Geben Sie die **Rekursionsgleichung** für die Laufzeit des folgenden Algorithmus an. Dabei sind die elementaren Operationen $\{+, -, *, /\} \in \mathcal{O}(1)$. Zuweisungen und Vergleiche können vernachlässigt werden.

- Geben Sie an, welche der Parameter einen Einfluss auf die Laufzeit haben.
- Für Verzweigungen im Kontrollfluss nutzen Sie bitte die Fallunterscheidung wie in der Vorlesung vorgestellt.

```
int function(int a, int b) {
  int n = b;
  int twoB = b;
  while(n > 0) {
    twoB = twoB + 1;
    n = n - 1;
  }
  if(a > 10) {
    return function(a/2,b) + function(a/2,b) + 2*function(a/2,b);
  } else if (a < 10 && a > 0) {
    return function(a/3,b-1) + function(a/3,b-1);
  } else {
    return 43;
  }
}
```

Lösung:

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

Beweis:
Wir haben $\lim_{n\to\infty} \frac{\frac{n^2 \cdot (\sqrt{n})^3}{n^3+1}}{n} = \lim_{n\to\infty} \frac{n \cdot (\sqrt{n})^3}{n^3+1} = \lim_{n\to\infty} \frac{n^{\frac{5}{2}}}{n^3+1} = \lim_{n\to\infty} \frac{1}{n^{\frac{1}{2}}+n^{-\frac{5}{2}}} = 0$. Damit gilt auch $\liminf_{n\to\infty} \frac{\frac{n^2 \cdot (\sqrt{n})^3}{n^3+1}}{n} = 0$, aber $\frac{n^2 \cdot (\sqrt{n})^3}{n^3+1} \in \Omega(n) \iff \liminf_{n\to\infty} \frac{\frac{n^2 \cdot (\sqrt{n})^3}{n^3+1}}{n} > 0$. □

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

Beweis:
Zu zeigen: $\exists c \in \mathbb{R}^{>0}.\ \exists n_0 \in \mathbb{N}.\ \forall n \in \mathbb{N}.\ n \geq n_0 \rightarrow (\Pi_{i=1}^{n}(i+n) \leq c \cdot n^{2n})$.
Es gilt für alle $n \geq 2$:

$$\begin{aligned} \Pi_{i=1}^{n}(i+n) &\leq \Pi_{i=1}^{n}(2n) \\ &= (2n)^n \\ &\leq (n^2)^n \\ &= n^{2n} \end{aligned}$$

Wähle also $c = 1$ und $n_0 = 2$. Damit ist die Aussage bewiesen. □

c) **Behauptung:** $T(n) \in \Theta(n^2)$

Beweis:
Wir haben $a = 16, b = 4$ und $E = 2$.
Wir zeigen $\sqrt{n^3} \in \mathcal{O}(n^{2-\epsilon})$ für $\epsilon = \frac{1}{2}$:

$$\lim_{n\to\infty} \frac{\sqrt{n^3}}{n^{2-\epsilon}} = \lim_{n\to\infty} \frac{n^{\frac{3}{2}}}{n^{2-\frac{1}{2}}} = \lim_{n\to\infty} 1 < \infty$$

Somit ist die Behauptung gültig (1. Fall des Mastertheorems). □

d)

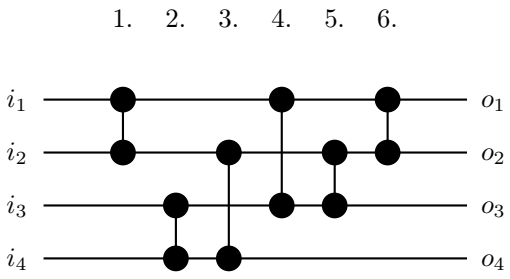
$$T_{function}(a,b) = \begin{cases} 3 \cdot T_{function}(a/2, b) + 2 \cdot b + c_1, & \text{wenn } a > 10 \wedge b > 0 \ (c_1 = 6) \\ 3 \cdot T_{function}(a/2, b) + c_1, & \text{wenn } a > 10 \wedge b \leq 0 \ (c_1 = 6) \\ 2 \cdot T_{function}(a/3, b-1) + 2 \cdot b + c_2, & \text{wenn } 0 < a < 10 \wedge b > 0 \ (c_2 = 5) \\ 2 \cdot T_{function}(a/3, b-1) + c_2, & \text{wenn } 0 < a < 10 \wedge b \leq 0 \ (c_2 = 5) \\ 2 \cdot b, & \text{wenn } (a = 10 \vee a \leq 0) \wedge b > 0 \\ 0, & \text{sonst, d.h. wenn } (a = 10 \vee a \leq 0) \wedge b \leq 0 \end{cases}$$

Aufgabe 2 (Sortierverfahren): (3 + 5 + 5 = 13 Punkte)

a) Sortieren Sie das Array `[ 8,5,1,9,7,3,2 ]` durch Anwendung des **Insertionsort**-Algorithmus **aus der Vorlesung**. Geben Sie dazu das Array **nach jeder Iteration der äußersten Schleife** an. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

b) Sortieren Sie das Array `[ 7,6,5,9,4,8,2,3 ]` durch Anwendung des **Quicksort**-Algorithmus **aus der Vorlesung**. Geben Sie dazu das Array **nach jeder Partition-Operation** an. Denken Sie daran, dass im Algorithmus aus der Vorlesung sowohl die Wahl des Pivot-Elements als auch das Verfahren zur Partitionierung festgelegt sind. Die Anzahl der vorgegebenen Zeilen muss nicht mit der Anzahl benötigter Schritte übereinstimmen.

c) Ein Sortiernetzwerk ist ein Schaltkreis für n Eingangssignale (Schlüssel) $i_1, \ldots, i_n$, welcher diese Signale sortiert auf n Ausgangssignale $o_1 \leq \cdots \leq o_n$ abbildet. Dazu dürfen lediglich Komparatorschaltungen als senkrechte Verbindungen zwischen jeweils zwei Leitungen i und j mit $i < j$ genutzt werden, welche die beiden Signale tauschen, wenn das Signal auf Leitung i größer als das Signal auf Leitung j ist. Ansonsten werden die Signale unverändert weitergeleitet. Das folgende Netzwerk ist zum Beispiel ein Sortiernetzwerk für 4 Signale:



Hier werden zunächst die Signale auf den Leitungen 1 und 2 miteinander verglichen und in korrekter Reihenfolge weitergeleitet. Anschließend werden die Signale auf den Leitungen 3 und 4 verglichen usw., sodass am Ende die Ausgangssignale o_1 bis o_4 den Schaltkreis in sortierter Reihenfolge verlassen. Zur Illustration wenden wir das Netzwerk auf die Beispieleingabe $i_1 = 5$, $i_2 = 3$, $i_3 = 4$, $i_4 = 1$ an und geben die Werte auf den Leitungen 1 bis 4 unmittelbar nach den Schritten 1. bis 6. an:

1.) $3, 5, 4, 1$

2.) $3, 5, 1, 4$

3.) $3, 4, 1, 5$

4.) $1, 4, 3, 5$

5.) $1, 3, 4, 5$

6.) $1, 3, 4, 5$

Geben Sie ein Sortiernetzwerk für 5 Signale an, welches die 5 Eingangssignale gemäß dem **Bubblesort**-Algorithmus **aus der Vorlesung** sortiert auf die 5 Ausgangssignale abbildet (Sie können dazu annehmen, dass `lengthOfE` aus dem Algorithmus in jeder Iteration um genau 1 reduziert wird).

Lösung:

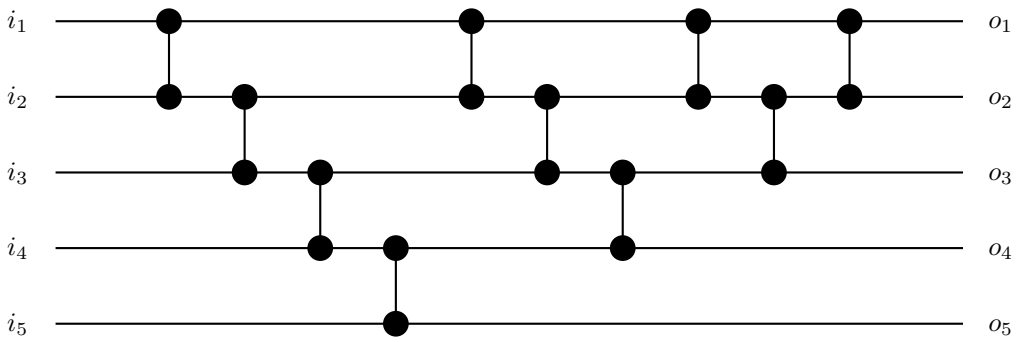
a)

8	5	1	9	7	3	2
5	8	1	9	7	3	2
1	5	8	9	7	3	2
1	5	8	9	7	3	2
1	5	7	8	9	3	2
1	3	5	7	8	9	2
1	2	3	5	7	8	9

b)

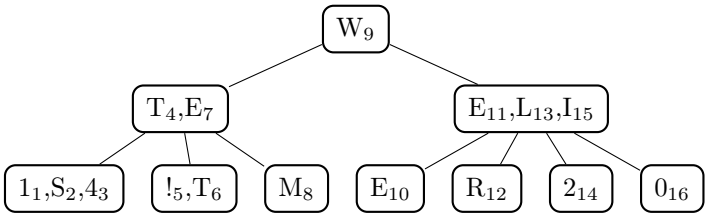
7	6	5	9	4	8	2	3
2	3	5	9	4	8	7	6
2	3	5	4	6	8	7	9
2	3	4	5	6	8	7	9
2	3	4	5	6	8	7	9
2	3	4	5	6	7	8	9

c)

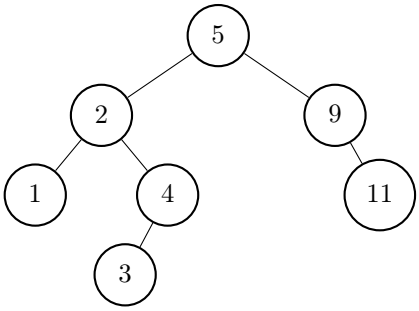


Aufgabe 3 (Bäume): (4 + 5 + 5 + 7 + 3 + 3 = 27 Punkte)

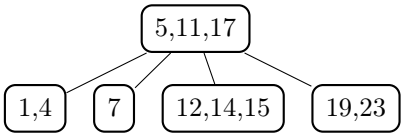
a) **Überführen** Sie den folgenden **2-3-4-Baum** in einen **Rot-Schwarz-Baum**. Nehmen Sie beim Überführen von einem 3-er Knoten den **rechten** Schlüssel als Wurzel. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.
Hinweis: Die Schlüssel sind die kleinen Zahlen rechts neben den im Knoten enthaltenen Daten.



b) Löschen Sie den Wert 9 aus dem folgenden **AVL-Baum** und geben Sie die entstehenden Bäume nach jeder **Löschoperation** sowie jeder **Rotation** an:



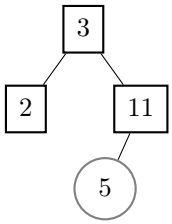
c) Fügen Sie den Wert 13 in den folgenden **2-3-4-Baum** ein und geben Sie den dabei entstehenden Baum an:



d) Fügen Sie den Wert 7 in den folgenden **Rot-Schwarz-Baum** ein und geben Sie die entstehenden Bäume nach

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

Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



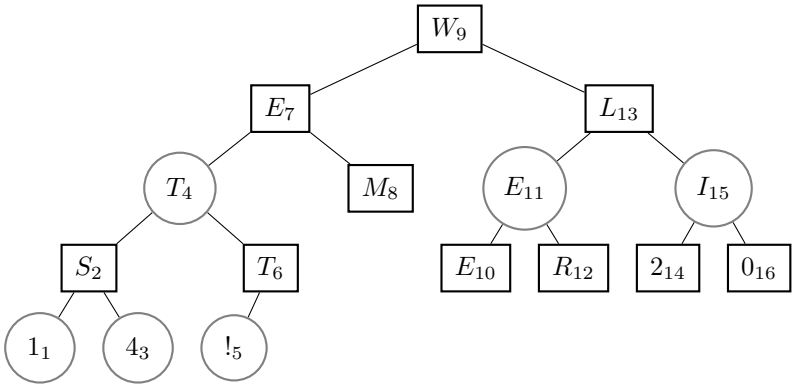
e) Geben Sie einen Baum der Tiefe 4 mit minimaler Knotenanzahl an, welcher **2-höhenbalanciert** ist.

f) Geben Sie einen Binärbaum mit den Schlüsselwerten von 0 bis 6 an, der bei **Inorder-Traversierung** die Schlüssel in folgender Reihenfolge ausgibt:

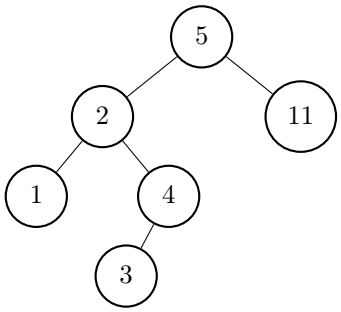
0, 5, 2, 3, 4, 1, 6

Lösung:

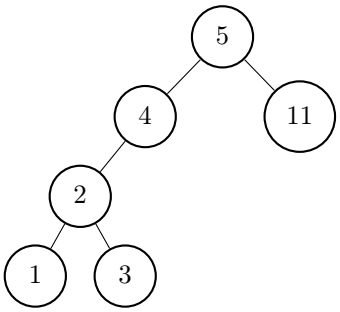
a)



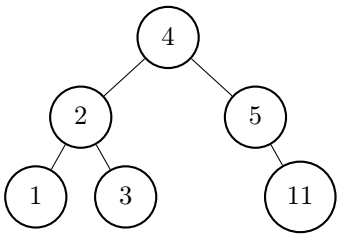
b) entferne 9



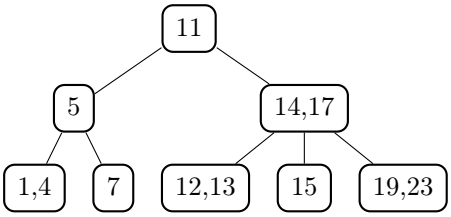
rotiere 2 nach links



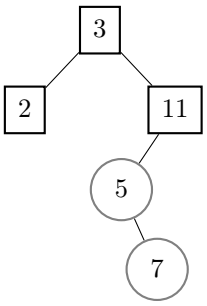
rotiere 5 nach rechts



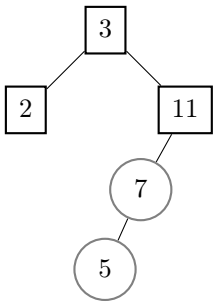
c) Schritt 1: Füge 13 ein



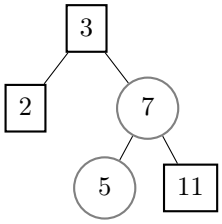
d) füge 7 ein



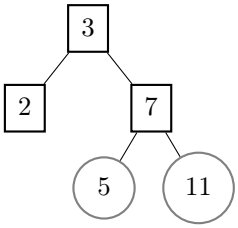
Fall 2: rotiere 5 nach links



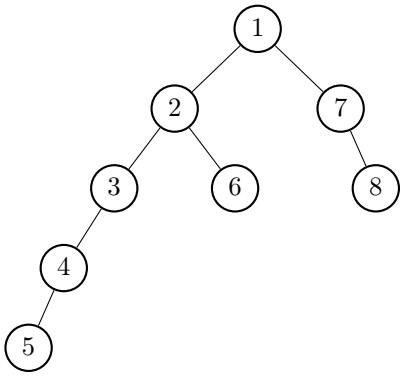
Fall 3: rotiere 11 nach rechts



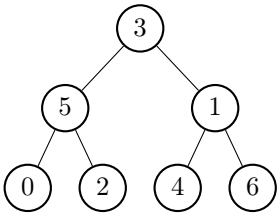
Fall 3: umfärben



e)



f)



Aufgabe 4 (Hashing): (3 + 3 + 4 = 10 Punkte)

a) Fügen Sie die folgenden Werte in das unten stehende Array `a` der Länge 7 unter Verwendung der **Divisionsmethode** mit **linearer Sondierung** ein ($f(n,i) = ((n \mod 7) + i) \mod 7$):

5, 7, 4, 11, 0, 12.

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 ($f(n,i) = ((n \mod 11) + 0.0 \cdot i + 1.0 \cdot i^2) \mod 11$):

3, 7, 14, 25, 1, 12.

c) Nennen Sie die **vier Eigenschaften**, nach denen eine **Hashfunktion qualitativ beurteilt** werden kann.

Lösung:

a) m = 7:

7	0	12		4	5	11

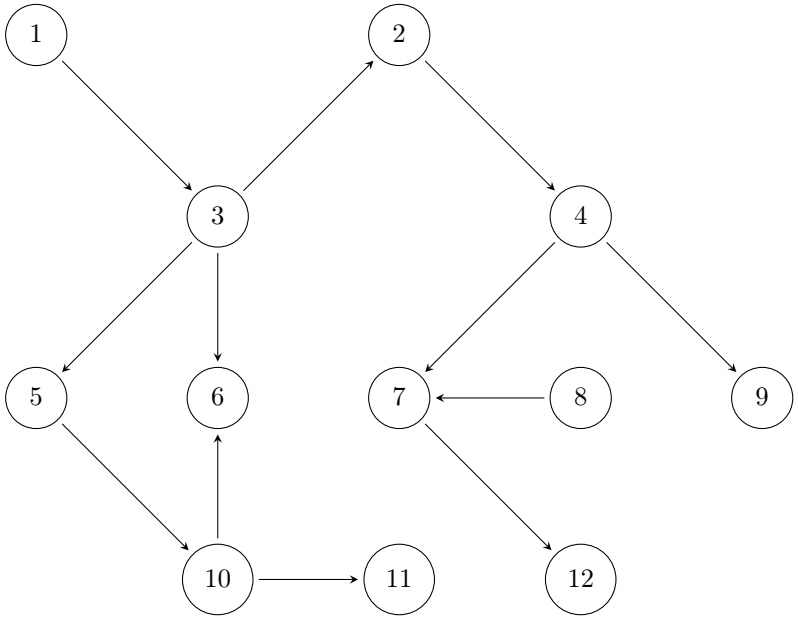
b) m = 11, $c_1 = 0.0$, $c_2 = 1.0$:

	25	1	3	14	12		7			

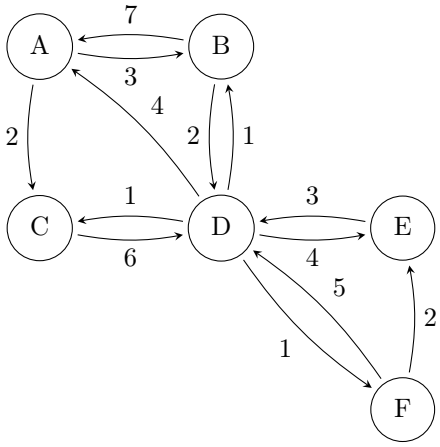
c)
- Komplexität
- Gleichverteilung der Schlüssel
- Surjektivität
- ähnliche Schlüssel sollen nicht nah beieinander liegen

Aufgabe 5 (Graphen): (4 + 7 + 10 + 5 = 26 Punkte)

a) Bestimmen Sie eine **topologische Sortierung** unter Verwendung des **in der Vorlesung** vorgestellten Algorithmus für den folgenden Graphen. Im gesamten Algorithmus werden Knoten in aufsteigender Reihenfolge ihrer Schlüssel berücksichtigt. Als Ergebnis geben Sie die Liste der Knotenschlüssel in aufsteigender Reihenfolge der Topologiewerte an.

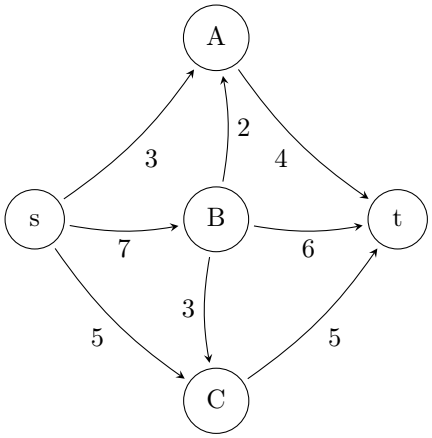


b) Betrachten Sie den folgenden Graphen:



Führen Sie den **Dijkstra** Algorithmus auf diesem Graphen mit dem **Startknoten A** aus. Füllen Sie dazu die nachfolgende Tabelle aus, indem Sie den Wert von `v` und `key` **nach jeder Iteration** der `while`-Schleife eintragen:

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.

d) Beweisen oder widerlegen Sie:

Der **Kondensationsgraph** $G{\downarrow} = (V',\ E')$ eines gerichteten Graphen $G = (V,\ E)$ ist **azyklisch**.

Lösung:

a) Der gegebene Graph hat die folgende topologische Sortierung:

12(1), 7(2), 9(3), 4(4), 2(5), 6(6), 11(7), 10(8), 5(9), 3(10), 1(11), 8(12)

b) Der Dijkstra Algorithmus füllt die Tabelle wie folgt aus:

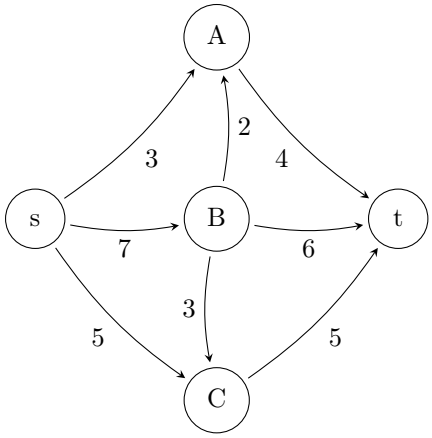
v	A	C	B	D	F
key[A]	0	0	0	0	0
key[B]	3	3	3	3	3
key[C]	2	2	2	2	2
key[D]	∞	8	5	5	5
key[E]	∞	∞	∞	9	8
key[F]	∞	∞	∞	6	6

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

c) Die Ford-Fulkerson Methode wird wie folgt ausgeführt:

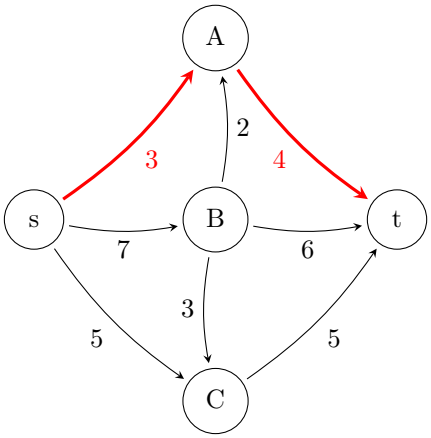
Schritt 0:

Initiales Flussnetzwerk:



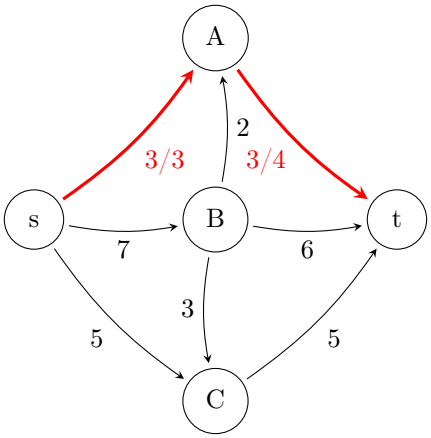
Schritt 1:

Restnetzwerk:



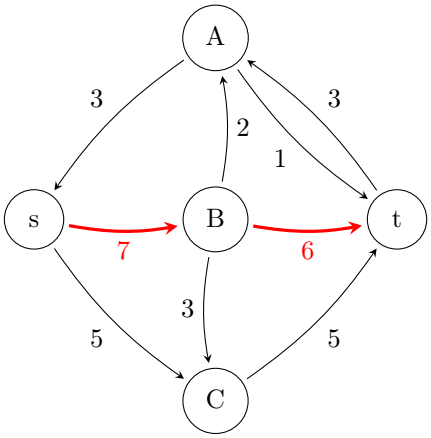
Schritt 2:

Nächstes Flussnetzwerk mit aktuellem Fluss:



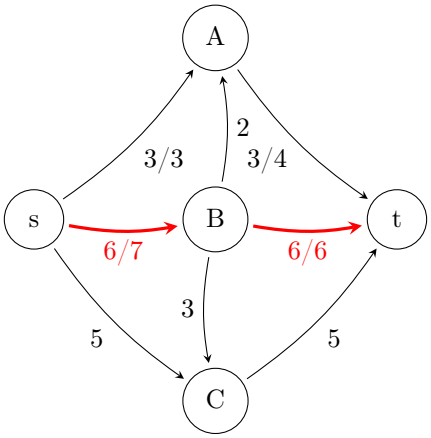
Schritt 3:

Restnetzwerk:



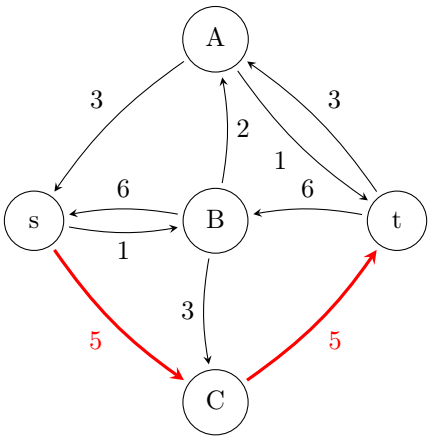
Schritt 4:

Nächstes Flussnetzwerk mit aktuellem Fluss:



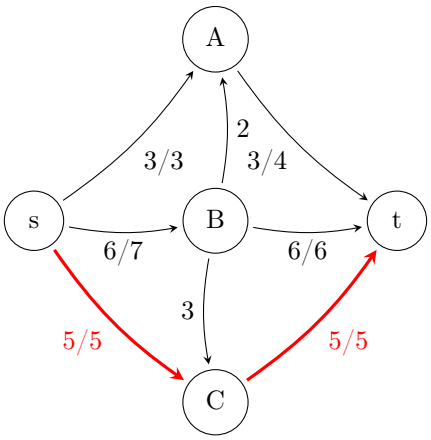
Schritt 5:

Restnetzwerk:



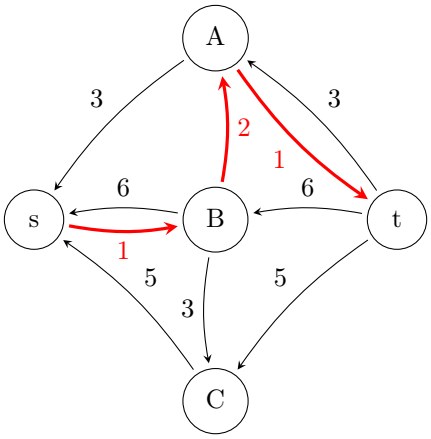
Schritt 6:

Nächstes Flussnetzwerk mit aktuellem Fluss:



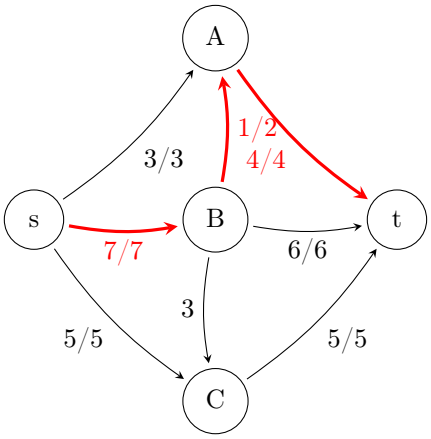
Schritt 7:

Restnetzwerk:



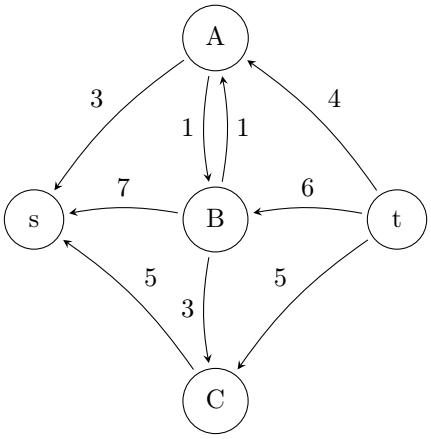
Schritt 8:

Nächstes Flussnetzwerk mit aktuellem Fluss:



Schritt 9:

Restnetzwerk:



Der maximale Fluss hat den Wert: 15

d) Behauptung: Die Aussage gilt.

Beweis:
Angenommen es gibt einen Zyklus in $G\downarrow$. Da $G\downarrow$ per Konstruktion keine Schlingen enthält, existieren zwei Knoten $v \neq v'$ im Zyklus, sodass man von v aus v' in $G\downarrow$ erreichen kann und umgekehrt. Da $G\downarrow$ der Kondensationsgraph von G ist, korrespondieren v und v' zu zwei verschiedenen starken Zusammenhangskomponenten S und S' von G. Aus der Konstruktion von $G\downarrow$ folgt dann, dass jeder Knoten aus S jeden Knoten in S' erreichen kann und umgekehrt. Dies steht im Widerspruch zur Maximaliät einer starken Zusammenhangskomponente per Definition, da S um die Knoten von S' zu einer starken Zusammenhangskomponente erweitert werden kann (analog kann S' um die Knoten von S zu einer starken Zusammenhangskomponente erweitert werden). □

Aufgabe 6 (Dynamische Programmierung): (2 + 10 = 12 Punkte)

a) Was ist **Memoization** und worauf kann man es anwenden?

b) Gegeben sei ein Rucksack mit **maximaler Tragkraft** 10 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 $w_0 = 4$, $w_1 = 5$, $w_2 = 3$ und $w_3 = 6$ und die **Werte** $c_0 = 3$, $c_1 = 4$, $c_2 = 2$ und $c_3 = 5$. Geben Sie zudem die vom Algorithmus bestimmte Tabelle `C` an und beschreiben Sie anhand der Tabelle wie man die mitzunehmenden Gegenstände bestimmen kann, um den maximalen Wert zu erreichen.

Lösung:

a) Memoization wird auf rekursive Algorithmen angewandt. Die Ergebnisse von Funktionsaufrufen werden gespeichert. Vor jedem neuen Funktionsaufruf wird geprüft, ob das Ergebnis bereits berechnet wurde. Wenn dem so ist, dann findet kein Aufruf statt sondern es wird das gespeicherte Ergebnis verwendet. Sonst wird das Ergebnis berechnet und gespeichert.

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

	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	↑ 0	← 0	← 0	← 0	← 3	3	3	3	3	3	3
2	0	0	0	0	↑ 3	4	4	4	4	7	7
3	0	0	0	2	↑ 3	4	4	5	6	7	7
4	0	0	0	2	↑ 3	← 4	← 5	← 5	← 6	← 7	← 8

Damit ergibt sich der maximale Wert 8 für den Fall, dass der 0. und 3. Gegenstand mitgenommen werden.

Dies lässt sich von der Tabelle wie folgt ablesen: Wenn die i-te Zeile einen Pfeil nach links enthält dann wird der $(i-1)$-te Gegenstand mitgenommen. Die Pfeile zeigen dabei an wie der folgende Algorithmus durch die Tabelle läuft:

```
int i = n; int j = M;
while (i > 0 && j > 0) {
    if (C[i][j] != C[i-1][j])
        for (int k = 0; k < w[i-1]; k++) j--;
    i--;
}
```

Aufgabe 7 (Geometrische Algorithmen): (2 + 5 = 7 Punkte)

a) Geben Sie die Definition einer **konvexen Hülle** an.

b) Berechnen Sie die konvexe Hülle der folgenden Punktmenge. Benutzen Sie dafür **Grahams Scan** wie **in der Vorlesung** vorgestellt und geben Sie die Teilschritte **nach jeder Iteration** (also nach jedem neu hinzugefügten Punkt) an. Umkreisen Sie die Punkte, die vom Algorithmus in der Iterationsschleife nicht betrachtet werden.

Lösung:

a) **Alternative I:** Die konvexe Hülle einer endlichen Menge Q von Punkten ist das kleinste konvexe Polygon P, für das sich jeder Punkt in Q entweder auf dem Rand von P oder in seinem Inneren befindet.

Alternative II: Die konvexe Hülle einer endlichen Menge Q von Punkten ist das kleinste Polytop (konvexes Polygon) P mit $Q \subseteq P$.

Alternative III: Gegeben sei ein Vektorraum V. Die konvexe Hülle einer Menge $Q \subseteq V$ ist die kleinste konvexe Menge $P \subseteq V$ mit $Q \subseteq P$.

Alternative IV: Gegeben sei ein Vektorraum V. Die konvexe Hülle einer Menge $Q \subseteq V$ ist der Schnitt $\bigcup_{\substack{Q \subseteq P \subseteq V \\ P \text{ konvex}}} P$ aller konvexen Obermengen von Q in V.

b)