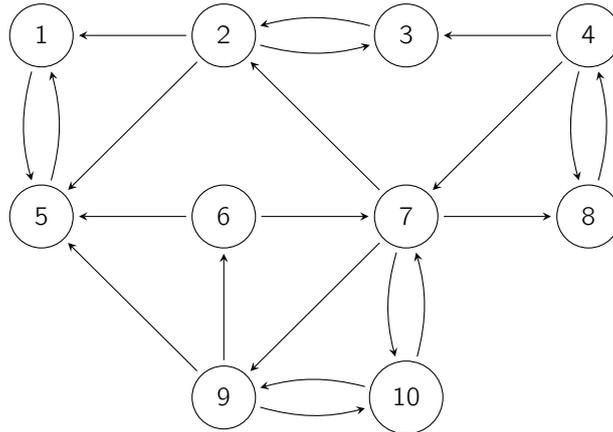


Tutoraufgabe 1 (Starke Zusammenhangskomponenten):

Wenden Sie *Sharir's Algorithmus* aus der Vorlesung an, um die starken Zusammenhangskomponenten des folgenden Graphen zu finden. Geben Sie das Array `color` und den Stack `S` nach jeder Schleifeniteration der ersten und zweiten Phase (also nach Zeile 17 und nach Zeile 22 im Code auf den Folien) an, falls DFS1 bzw. DFS2 ausgeführt wurde. Geben Sie zudem das Array `scc` nach jeder Schleifeniteration der zweiten Phase (also nach Zeile 22) an, falls DFS2 ausgeführt wurde. Nehmen Sie hierbei an, dass `scc` initial mit Nullen gefüllt ist und die Knoten im Graphen und in allen Adjazenzlisten aufsteigend nach ihren Schlüsselwerten sortiert sind, also der Knoten mit Schlüssel 1 vom Algorithmus als erstes berücksichtigt wird usw.



Lösung: _____

Phase 1:

S: 5, 1

color: (1, s), (2, w), (3, w), (4, w), (5, s), (6, w), (7, w), (8, w), (9, w), (10, w)

S: 5, 1, 3, 2

color: (1, s), (2, s), (3, s), (4, w), (5, s), (6, w), (7, w), (8, w), (9, w), (10, w)

S: 5, 1, 3, 2, 8, 6, 10, 9, 7, 4

color: (1, s), (2, s), (3, s), (4, s), (5, s), (6, s), (7, s), (8, s), (9, s), (10, s)

Phase 2:

S: 5, 1, 3, 2, 8, 6, 10, 9, 7

color: (1, w), (2, w), (3, w), (4, s), (5, w), (6, s), (7, s), (8, s), (9, s), (10, s)

scc: (1, 0), (2, 0), (3, 0), (4, 4), (5, 0), (6, 4), (7, 4), (8, 4), (9, 4), (10, 4)

S: 5, 1, 3

color: (1, w), (2, s), (3, s), (4, s), (5, w), (6, s), (7, s), (8, s), (9, s), (10, s)

scc: (1, 0), (2, 2), (3, 2), (4, 4), (5, 0), (6, 4), (7, 4), (8, 4), (9, 4), (10, 4)

S: 5

color: (1, s), (2, s), (3, s), (4, s), (5, s), (6, s), (7, s), (8, s), (9, s), (10, s)

scc: (1, 1), (2, 2), (3, 2), (4, 4), (5, 1), (6, 4), (7, 4), (8, 4), (9, 4), (10, 4)

Der gegebene Graph hat die folgenden starken Zusammenhangskomponenten:

{1, 5}
{2, 3}
{4, 6, 7, 8, 9, 10}

Tutoraufgabe 2 (Kondensationsgraph):

- a) Zeigen Sie, dass für jeden gerichteten Graphen G gilt, dass der transponierte Graph des Kondensationsgraphen von G gleich dem Kondensationsgraphen des transponierten Graphen von G ist:

$$(G \downarrow)^T = (G^T) \downarrow$$

- b) Entwerfen Sie einen Algorithmus, der für einen gegebenen gerichteten Graphen $G = (V, E)$ in Laufzeit $\mathcal{O}(|V| + |E|)$ den Graph $G \downarrow$ berechnet. Stellen Sie insbesondere sicher, dass es in $G \downarrow$ höchstens eine Kante von einem Knoten v zu einem anderen Knoten w gibt (d.h. es darf keine doppelten Kanten geben).

Lösung: _____

- a) Die Transposition G^T von G ist

$$G^T = (V, E^T = \{(w, v) \mid \exists (v, w) \in E\}).$$

Wir zeigen nun, dass die Knotenmengen der starken Zusammenhangskomponenten eines beliebigen gerichteten Graphen $G = (V, E)$ und seiner Transposition gleich sind. Sei dazu $S = \{v_1, \dots, v_k\} \subseteq V$ eine starke Zusammenhangskomponente von G . Seien $v_i, v_k \in S$. Per Definition gilt, dass es einen Pfad von v_i nach v_k in G gibt, d.h.

$$v_i = v_k \vee (v_i, v_k) \in E \vee \exists u_1, \dots, u_m \in S : (v_i, u_1), (u_1, u_2), \dots, (u_m, v_k) \in E \quad (1)$$

und dass es einen Pfad von v_k nach v_i gibt:

$$v_k = v_i \vee (v_k, v_i) \in E \vee \exists u'_1, \dots, u'_\ell \in S : (v_k, u'_1), \dots, (u'_\ell, v_i) \in E \quad (2)$$

Betrachten wir nun G^T . (1) gilt genau dann wenn

$$v_k = v_i \vee (v_k, v_i) \in E^T \vee \exists u_1, \dots, u_m \in S : (v_k, u_m), \dots, (u_2, u_1), (u_1, v_i) \in E^T \quad (3)$$

und es existiert somit ein Pfad von v_k nach v_i in G^T und (2) gilt genau dann wenn

$$v_i = v_k \vee (v_i, v_k) \in E^T \vee \exists u'_1, \dots, u'_\ell \in S : (v_i, u'_1), (u'_1, u'_2), \dots, (u'_\ell, v_k) \in E^T \quad (4)$$

und es existiert somit ein Pfad von v_i nach v_k in G^T . Es gilt also, dass v_i von v_k sich genau dann gegenseitig in G erreichen können, wenn sie sich in G^T erreichen können. Die Knotenmengen der starken Zusammenhangskomponenten sind also in G und G^T die gleichen.

Daher ist der Kondensationsgraph $(G^T) \downarrow$ von $G^T = (V, E^T)$

$$(G^T) \downarrow = (\{S_1, \dots, S_p\}, \{(S_j, S_i) \mid i, j \in \{1, \dots, p\}, \exists (w, v) \in E^T : w \in S_j, v \in S_i\})$$

Wegen $(w, v) \in E^T \Leftrightarrow (v, w) \in E$ gilt

$$(G^T) \downarrow = (\{S_1, \dots, S_p\}, \{(S_j, S_i) \mid \exists (v, w) \in E : w \in S_j, v \in S_i\}) = (G \downarrow)^T$$

Damit ist die Aussage gezeigt.

- b) Der Algorithmus könnte wie folgt vorgehen.

- Wir benutzen den Algorithmus aus der Vorlesung, um die starken Zusammenhangskomponenten von G in Zeit $\mathcal{O}(|V| + |E|)$ berechnet. Wir schreiben das Ergebnis in ein Array `scc`, so dass `scc[u]` $\in \{1, \dots, |V|\}$ die Zusammenhangskomponente von `u` ist.
- Sortiere das Array `scc` "in" ein Array `scc'` mit Hilfe von **Countingsort**. Da die zu sortierenden Zahlen im Bereich $1, \dots, |V|$ liegen, ist die Laufzeit beschränkt durch $\mathcal{O}(|V| + |V|) = \mathcal{O}(|V|)$.

- Iteriere über scc' und füge alle Zahlen, die nicht gleich dem Vorgänger (im Array) sind, zu $V \downarrow$ hinzu. Die Gesamtlaufzeit hierfür ist $\mathcal{O}(|V|)$.
- Fülle ein Array S der Länge (max.) $|E|$ mit den geordneten Paaren

$$\{(x, y) \mid (u, v) \in E \wedge x = scc[u] \wedge y = scc[v] \wedge x \neq y\}$$

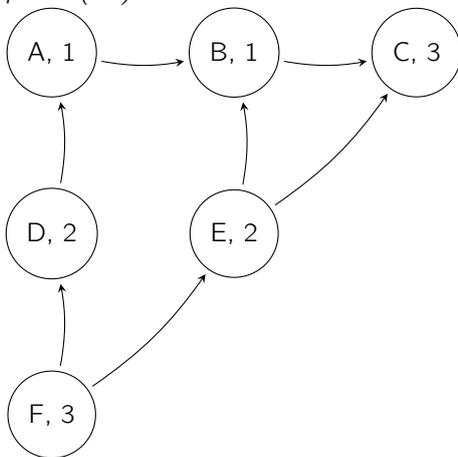
in Zeit $\mathcal{O}(|E|)$.

- Sortiere die Elemente in S zweimal mit Hilfe von Countingsort: das erste Mal nach der zweiten Komponente, danach nach der ersten Komponente. Da Countingsort stabil ist, ist das Ergebnis nach der ersten Komponente sortiert **und** Elemente mit gleicher erster Komponente sind nach der zweiten Komponente sortiert. Die zwei Durchläufe haben eine Laufzeit von $\mathcal{O}(|V| + |E|)$, da wir $\mathcal{O}(|E|)$ Elemente mit Schlüsseln im Bereich $1, \dots, |V|$ sortieren.
- Iteriere über das (sortierte) Array S und füge eine Kante (x, y) genau dann zu $E \downarrow$ hinzu, wenn es verschieden von dem vorhergehenden Element (in S) ist. Dies verhindert, dass wir eine Kante (x, y) zweimal zu $E \downarrow$ hinzufügen. Die Laufzeit hierfür ist beschränkt durch $\mathcal{O}(|E|)$.

Insgesamt ergibt sich eine Laufzeit von $\mathcal{O}(|V| + |E|)$.

Tutoraufgabe 3 (Kritischer Pfad):

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.



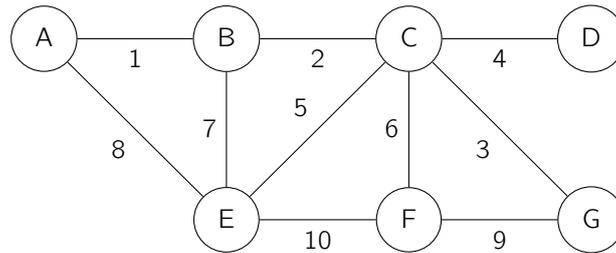
Lösung: _____

Der gegebene Graph hat die folgende topologische Sortierung:

C (3), B (4), A (5), D (7), E (6), F (10)
eft: 10

Tutoraufgabe 4 (Minimaler Spannbaum):

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,

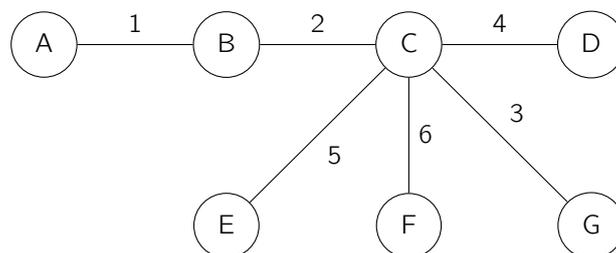
1. welche Kosten die Randknoten haben (d. h. für jeden Knoten v in pq die Priorität von v , wobei ∞ angibt, dass der entsprechende Knoten noch nicht zum Randbereich gehört)
2. und welchen Knoten `pq.getMin()` wählt, indem Sie den Kosten-Wert des gewählten Randknoten in der Tabelle unterstreichen (wie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

Geben Sie zudem den vom Algorithmus bestimmten minimalen Spannbaum an.

Lösung: _____

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

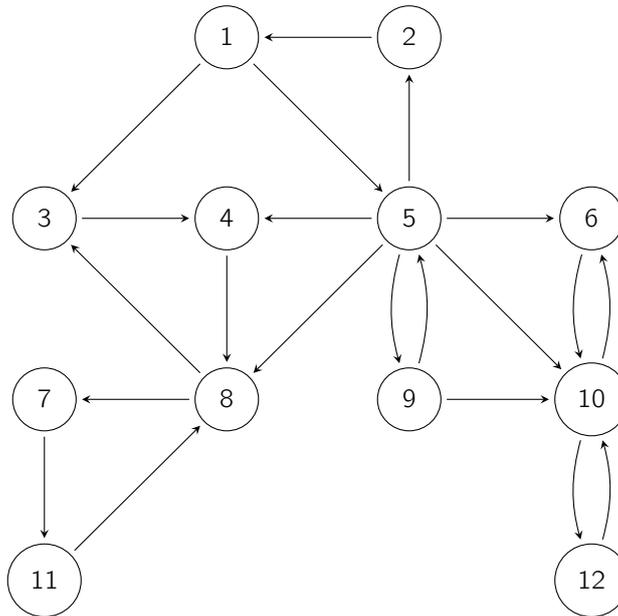


Aufgabe 5 (Starke Zusammenhangskomponenten):

(8 Punkte)

Wenden Sie *Sharir's Algorithmus* aus der Vorlesung an, um die starken Zusammenhangskomponenten des folgenden Graphen zu finden. Geben Sie das Array `color` und den Stack S nach jeder Schleifeniteration der ersten und zweiten Phase (also nach Zeile 17 und nach Zeile 22 im Code auf den Folien) an, falls DFS1 bzw. DFS2 ausgeführt wurde. Geben Sie zudem das Array `scc` nach jeder Schleifeniteration der zweiten Phase (also nach Zeile 22) an, falls DFS2 ausgeführt wurde. Nehmen Sie hierbei an, dass `scc` initial mit Nullen gefüllt ist und die Knoten im

Graphen und in allen Adjazenzlisten aufsteigend nach ihren Schlüsselwerten sortiert sind, also der Knoten mit Schlüssel 1 vom Algorithmus als erstes berücksichtigt wird usw.



Lösung: _____

Phase 1:

S: 11, 7, 8, 4, 3, 2, 12, 10, 6, 9, 5, 1

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

Phase 2:

S: 11, 7, 8, 4, 3, 2, 12, 10, 6, 9, 5

color: (1, s), (2, s), (3, w), (4, w), (5, s), (6, w), (7, w), (8, w), (9, s), (10, w), (11, w), (12, w)

scc: (1, 1), (2, 1), (3, 0), (4, 0), (5, 1), (6, 0), (7, 0), (8, 0), (9, 1), (10, 0), (11, 0), (12, 0)

S: 11, 7, 8, 4, 3, 2, 12, 10

color: (1, s), (2, s), (3, w), (4, w), (5, s), (6, s), (7, w), (8, w), (9, s), (10, s), (11, w), (12, s)

scc: (1, 1), (2, 1), (3, 0), (4, 0), (5, 1), (6, 6), (7, 0), (8, 0), (9, 1), (10, 6), (11, 0), (12, 6)

S: 11, 7, 8, 4

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

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

Der gegebene Graph hat die folgenden starken Zusammenhangskomponenten:

{1, 2, 5, 9}
{3, 4, 7, 8, 11}
{6, 10, 12}

Aufgabe 6 (Starke Zusammenhangskomponenten II):

(4 + 7 = 11 Punkte)

- a) Professor Cartoon (der böse Zwillingbruder von Professor Katoen) behauptet, dass der Algorithmus zum Erkennen starker Zusammenhangskomponenten vereinfacht werden kann, indem man in der zweiten Tiefensuche den eigentlichen Graphen G statt des transponierten Graphen benutzt und den Stack des Algorithmus

durch eine Queue ersetzt (und damit die Reihenfolge der Startknoten der Tiefensuche in der zweiten Phase "umkehrt").

Legen Sie Professor Cartoon das Handwerk indem Sie ihn widerlegen.

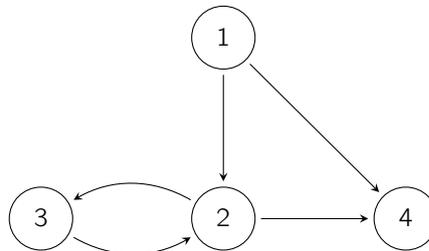
- b)** Geben Sie einen Algorithmus (Pseudocode oder präzise Beschreibung) an, welcher für einen gerichteten Graphen $G = (V, E)$ in Laufzeit $\mathcal{O}(|V| + |E|)$ einen Graphen $G' = (V, E')$ berechnet, so dass
- (a) G' die gleichen starken Zusammenhangskomponenten wie G hat,
 - (b) $G' \downarrow = G \downarrow$, und
 - (c) $|E'|$ minimal ist (über alle Graphen, die die ersten beiden Bedingungen erfüllen).

Begründen Sie, wieso Ihr Algorithmus die Laufzeitschranke einhält und wieso er korrekt arbeitet.

Hinweis: Sie dürfen das Ergebnis von Tutoraufgabe 2 b) benutzen.

Lösung: _____

- a)** Wir zeigen Professor Cartoon folgenden Graphen.



Nach der ersten Phase des Algorithmus erhalten wir den Queue-Inhalt $\langle 3, 4, 2, 1 \rangle$, wobei die 3 am *head* der Queue steht. Eine Tiefensuche von 3 identifiziert (fälschlicherweise) die "starke Zusammenhangskomponente" $\{2, 3, 4\}$, die aber keine SCC ist.

- b)** Idee: ersetze die Kanten innerhalb einer SCC durch einen Kreis.

Ein Algorithmus könnte wie folgt aussehen:

- Berechne die SCCs von G und zeitgleich ein Array `leader` welches an Index i den Leader der SCC V_i speichert. Dies geht in Laufzeit $\mathcal{O}(|V| + |E|)$.
- Berechne $G \downarrow$ in Laufzeit $\mathcal{O}(|V| + |E|)$ (siehe Tutoraufgabe 2).
- Setze $E' = \emptyset$.
- Für jede SCC $V_i = \{v_1, \dots, v_k\}$, erweitere E' durch

$$E' = E' \cup \{(v_1, v_2), (v_2, v_3), \dots, (v_{k-1}, v_k), (v_k, v_1)\}$$

Die insgesamt benötigte Laufzeit hierfür ist $\mathcal{O}(|V|)$.

- Für jede Kante (V_i, V_j) in $G \downarrow$, wähle jeweils einen Repräsentanten füge $(\text{leader}[i], \text{leader}[j])$ zu E' hinzu. Die dafür benötigte Laufzeit ist in $\mathcal{O}(|E|)$.

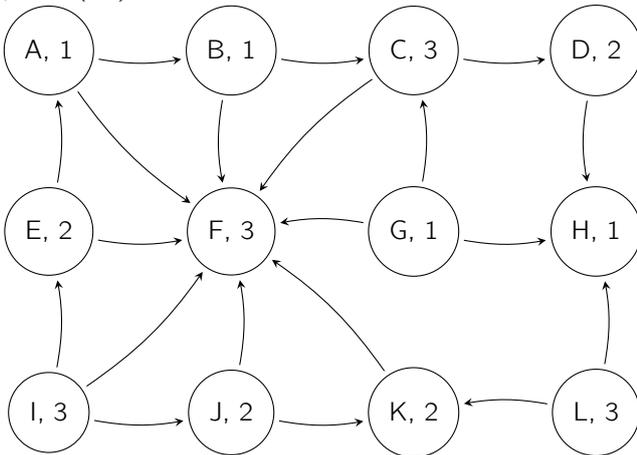
Insgesamt ergibt sich eine obere Laufzeitschranke von $\mathcal{O}(|V| + |E|)$. Es bleibt, die Korrektheit zu begründen.

G' hat die gleichen starken Zusammenhangskomponenten wie G , da alle Knoten einer SCC aus G durch einen Kreis verbunden sind und G' nicht größere SCCs haben kann als G , da die zusätzlich hinzugefügten Kanten "aus $G \downarrow$ " stammen und daher keine Zyklen erzeugen können ($G \downarrow$ ist azyklisch). Aus diesem Grund ist auch $G' \downarrow = G \downarrow$. Die Kantenanzahl von G' ist außerdem minimal unter den Graphen, die diese beiden Eigenschaften erfüllen, da mindestens k Kanten benötigt werden, um eine SCC mit k Knoten zu bilden und G' dafür genau k Kanten benutzt. Wird auch nur eine Kante gelöscht, die "aus dem Kondensationsgraphen stammt", so ist die Bedingung $G' \downarrow = G \downarrow$ verletzt.

Aufgabe 7 (Kritischer Pfad):

(5 Punkte)

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.



Lösung: _____

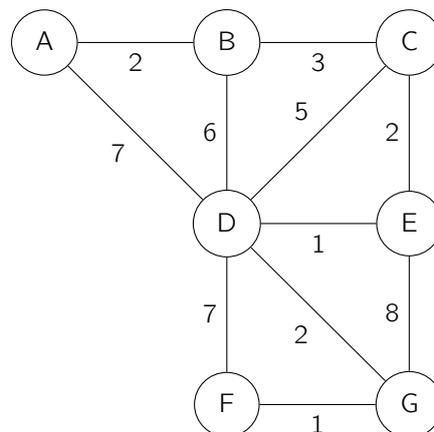
Der gegebene Graph hat die folgende topologische Sortierung:

H (1), D (3), E (3), C (6), B (7), A (8), E (10), G (7), K (5), J (7), I (13), L (8)
eft: 13

Aufgabe 8 (Minimaler Spannbaum):

(4 Punkte)

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,

1. welche Kosten die Randknoten haben (d. h. für jeden Knoten v in p_q die Priorität von v , wobei ∞ angibt, dass der entsprechende Knoten noch nicht zum Randbereich gehört)

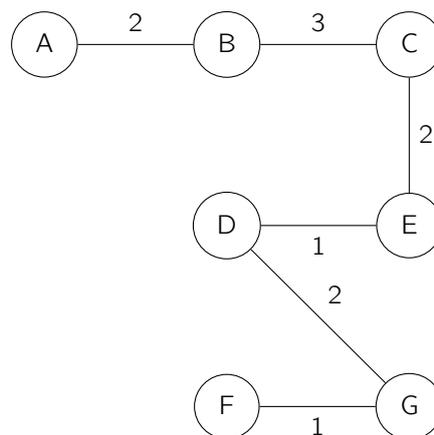
2. und welchen Knoten `pq.getMin()` wählt, indem Sie den Kosten-Wert des gewählten Randknoten in der Tabelle unterstreichen (wie es in der ersten Zeile bereits vorgegeben ist).

Geben Sie zudem den vom Algorithmus bestimmten minimalen Spannbaum an.

Lösung: _____

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

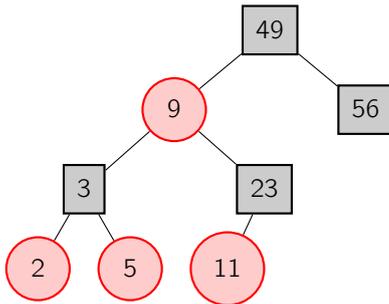
Hierbei gibt eine unterstrichene Zahl an, in welcher Iteration (zugehöriger Zeilenkopf) welcher Knoten (zugehöriger Spaltenkopf) durch `pq.getMin()` gewählt wurde. Wir erhalten den folgenden minimalen Spannbaum:



Aufgabe 9 (Rot-Schwarz-Bäume): (15 Minuten, 2 + 1,5 + 2,5 + 2 = 8 Punkte)

- a) 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.

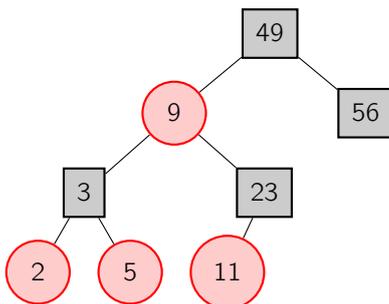
Markieren Sie außerdem zu jeder Rotation, welcher Knoten in welche Richtung rotiert wird. Mehrere Umfärbungen können Sie in einem Schritt zusammenfassen. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



b) Fügen Sie den Wert 10 in den folgenden *Rot-Schwarz-Baum* ein und geben Sie die entstehenden Bäume nach

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

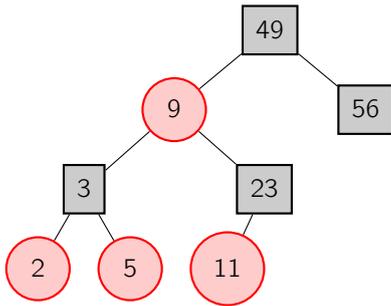
Markieren Sie außerdem zu jeder *Rotation*, welcher Knoten in welche Richtung rotiert wird. Mehrere *Umfärbungen* können Sie in einem Schritt zusammenfassen. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



c) Löschen Sie den Wert 56 aus dem folgenden *Rot-Schwarz-Baum* und geben Sie die entstehenden Bäume nach

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

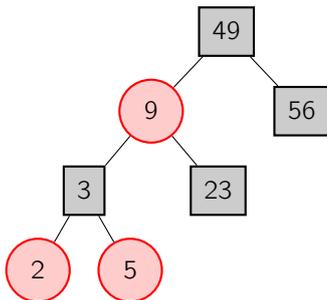
Markieren Sie außerdem zu jeder *Rotation*, welcher Knoten in welche Richtung rotiert wird. Mehrere *Umfärbungen* können Sie in einem Schritt zusammenfassen. Beachten Sie, dass rote Knoten rund und schwarze Knoten eckig dargestellt werden.



d) Löschen Sie den Wert 9 aus dem folgenden *Rot-Schwarz-Baum* und geben Sie die entstehenden Bäume nach

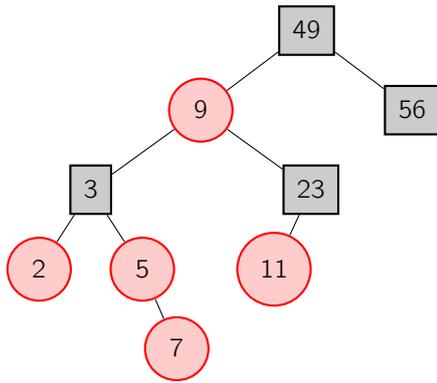
- jeder *Löschoperation*,
- jeder *Rotation* sowie
- jeder *Umfärbung* an.

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.

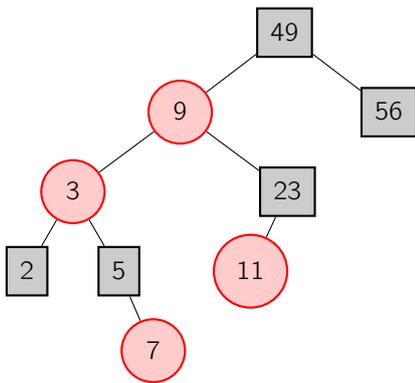


Lösung: _____

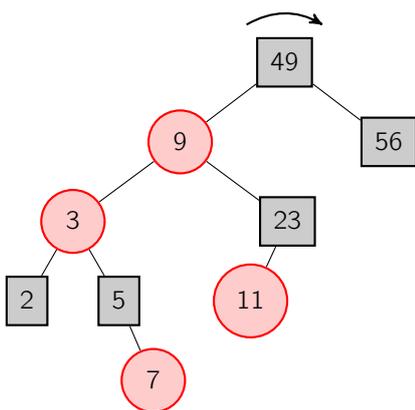
a) füge 7 ein

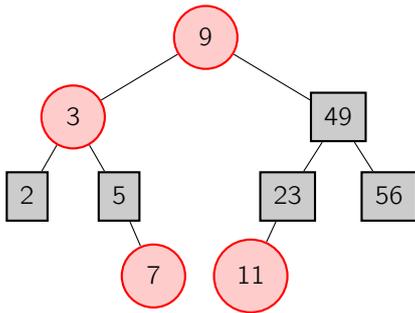


Fall 1: umfärben

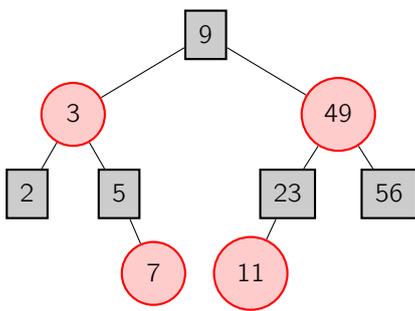


rotiere 49 nach rechts

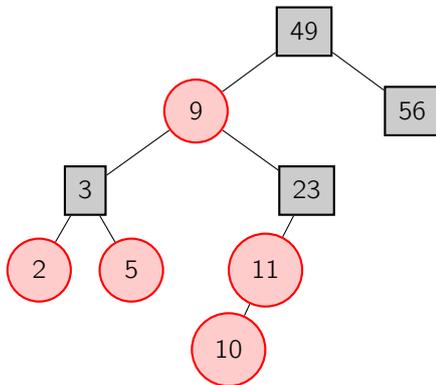




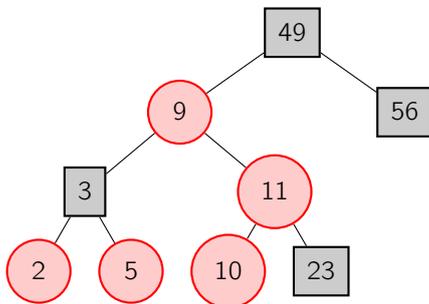
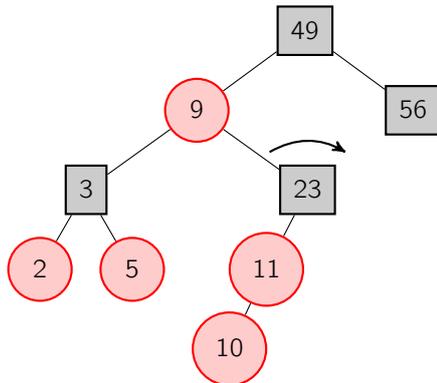
Fall 3: umfärben



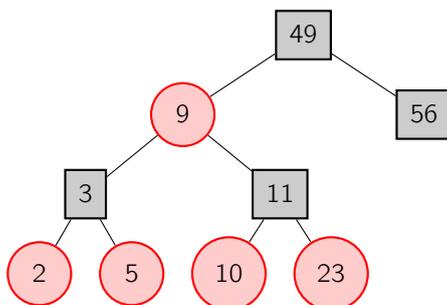
b) füge 10 ein



rotiere 23 nach rechts

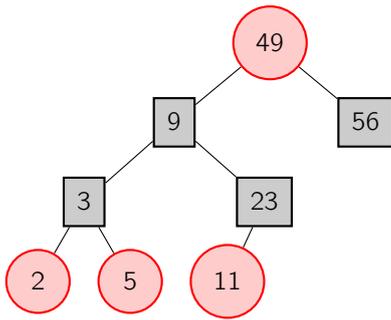


Fall 3: umfärben

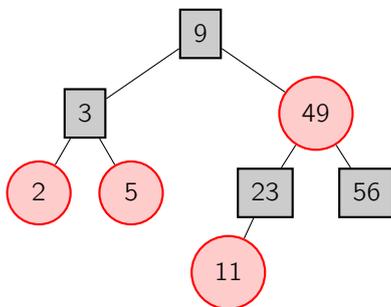
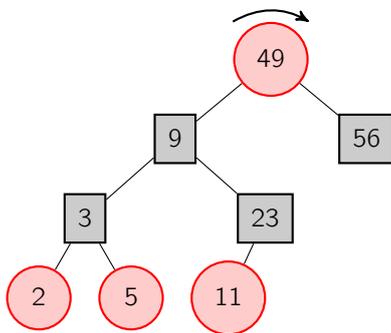


c) Lösche 56:

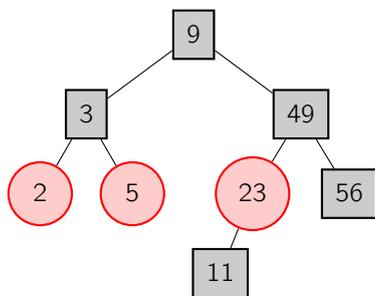
Fall 1: umfärben



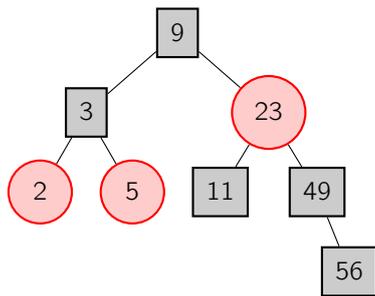
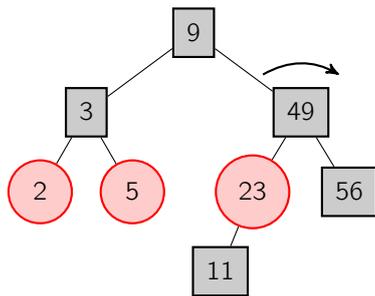
rotiere 49 nach rechts



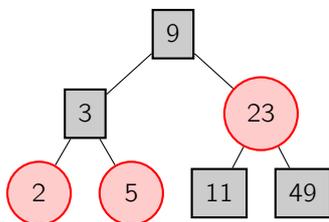
Fall 4: umfärben



rotiere 49 nach rechts



ersetze 56 durch rechtes Kind

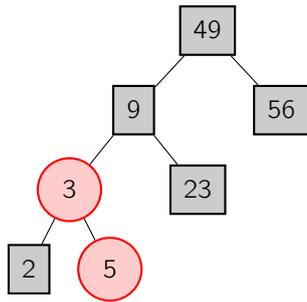


d) Lösche 9:

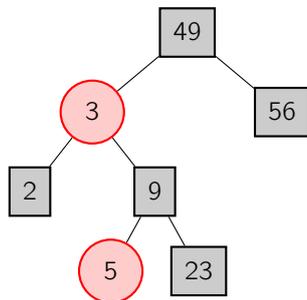
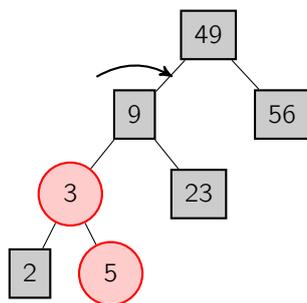
Wir löschen den Knoten mit dem nächst größeren Wert 23 und fügen diesen Wert dann in den zu löschenden Knoten ein.

Lösche 23:

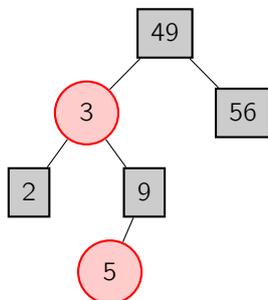
Fall 4: umfärben



rotiere 9 nach rechts



ersetze 23 durch rechtes Kind



füge 23 in den zu löschenden Knoten ein

