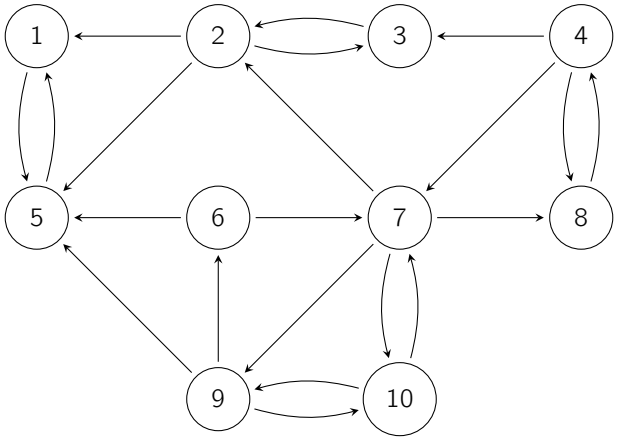
Tutoraufgabe 1 (Starke Zusammenhangskomponenten):

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 berücksichtig 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 $S = \{v_1, \ldots, v_k\} \subseteq V$ 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.

$$v_i = v_k \vee (v_i, v_k) \in E \vee \exists u_1, \ldots, u_m \in S : (v_i, u_1), (u_1, u_2), \ldots, (u_m, v_k) \in E \tag{1}$$

und dass es einen Pfad von v_k nach v_i gibt:

$$v_k = v_i \vee (v_k, v_i) \in E \vee \exists u'_1, \ldots, u'_\ell \in S : (v_k, u'_\ell), \ldots, (u_2, u'_1), (u'_1, v_i) \in E \tag{2}$$

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

$$v_k = v_i \vee (v_k, v_i) \in E^T \vee \exists u_1, \ldots, u_m \in S : (v_k, u_m), \ldots, (u_2, u_1), (u_1, v_i) \in E^T \tag{3}$$

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

$$v_i = v_k \vee (v_i, v_k) \in E^T \vee \exists u'_1, \ldots, u'_\ell \in S : (v_i, u'_1), (u'_1, u'_2), \ldots, (u'_\ell, v_k) \in E^T \tag{4}$$

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 $G^T = (V, E^T)$

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

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

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

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]` $\in \{1, \ldots, |V|\}$ die Zusammenhangskomponente von `u` ist.
- Sortiere das Array `scc` "in" ein Array `scc'` mit Hilfe von **Countingsort**. Da die zu sortierenden Zahlen im Bereich $1, \ldots, |V|$ 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

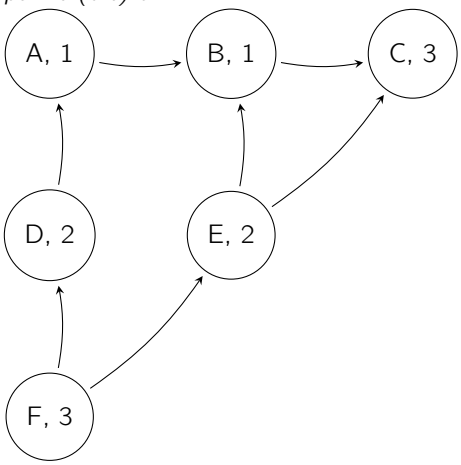
  $$\{(x, y) \mid (u, v) \in E \wedge x = \texttt{scc}[\texttt{u}] \wedge y = \texttt{scc}[\texttt{v}] \wedge x \neq y\}$$

  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 $1, \ldots, |V|$ sortieren.
- Iteriere über das (sortiere) 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.



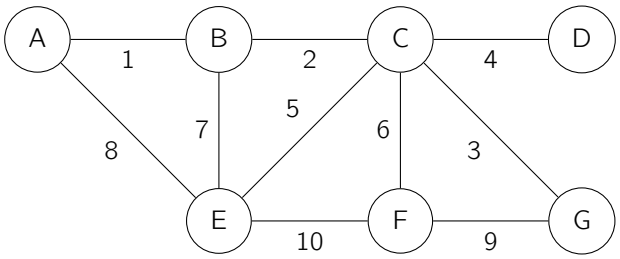
Lösung:

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: