Universität Basel

Algorithmische Mathematik

Graphen & Anwendungen

Skript zur Vorlesung
im
FS 2025

Helmut Harbrecht, Michael Multerer & Marc Schmidlin

Version vom 5. Juni 2025

Informationen ...

zu den Übungen

Auf den Übungsblättern kommen Theorie- und Programmieraufgaben vor. Die Ausgabe und Abgabe der Übungsblätter findet jeweils am Montag statt. Für die Theorieaufgaben gelten folgende Regeln:

- Die Lösungen sind handschriftlich (Papier oder Tablet) zu schreiben.
- Die Abgaben erfolgen dabei entweder physisch an der Spiegelgasse 1 oder digital über ADAM durch jeweils **ein** PDF-File pro Übungsblattserie. Abgegebene PDF-Files müssen direkt A4-Papier druckbar sein.

Für die Programmieraufgaben gelten folgende Regeln:

- Die Lösungen sind in `MATLAB` zu programmieren.
- Die Abgabe der Codes erfolgt dabei immer digital über ADAM durch jeweils **ein** ZIP-File pro Übungsblattserie. Die ZIP-Files enthalten dabei jeweils alle notwendigen Sourcecode-Files. Falls in den Programmieraufgaben Fragen zu beantworten sind, werden allfällige Antworten mit den Theorieaufgaben abgegeben.

Die Rückgabe der korrigierten Abgaben erfolgt physisch in der Übungsstunde.

zu der Leistungsüberprüfung

Die Vorlesung *Algorithmische Mathematik: Graphen und Anwendungen* wird als eine kombinierte Veranstaltung "Vorlesung mit Übung" (6KP) mit lehrveranstaltungs-begleitender Leistungsüberprüfung gehalten, welche wie folgt angedacht ist:

- In den Übungsblätter sind 50% der Punkte, von den abzugebenden Übungsaufgaben, zu erreichen, um die Zulassung für die mündliche Prüfung zu erhalten.
- Es ist *keine* schriftliche Prüfung vorgesehen.
- Die mündliche Prüfung (Dauer 30min) wird benotet (Skala 1-6 0,5). Die mündlichen Prüfungen finden vom 04-06.06.2025 statt.

Wenn Sie die Zulassung für die mündliche Prüfung erhalten, wird die Veranstaltung für Sie mit der Note der mündlichen Prüfung benotet. Haben Sie jedoch die Zulassung für die mündliche Prüfung nicht erhalten, dürfen Sie nicht an der mündlichen Prüfung teilnehmen und die Veranstaltung wird für Sie mit einer ungenügenden Note bewertet.

Vorwort

Diese Mitschrift kann und soll nicht ganz den Wortlaut der Vorlesung wiedergeben. Sie soll das Nacharbeiten des Inhalts der Vorlesung erleichtern.

Literatur zur Vorlesung:

- H. Harbrecht und M. Multerer: *Algorithmische Mathematik: Graphen, Numerik und Probabilistik*, Springer Spektrum
- N. Blum: *Algorithmen und Datenstrukturen*, Oldenbourg-Verlag
- B. Korte und J. Vygen: *Combinatorial Optimization: Theory and Algorithms*, Springer-Verlag

Inhaltsverzeichnis

1 GRUNDLAGEN

1.1 Graphen

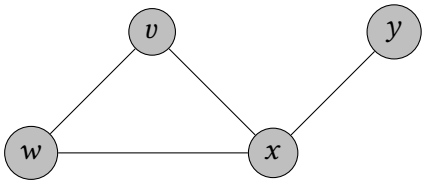
Definition 1.1 Ein *Graph* ist ein Paar $G = (V, E)$, bestehend aus endlicher Menge V von *Knoten* (vertices) und einer endlichen Menge von ungeordneten Paaren $\{v, w\} \subseteq V$,

$$E \subseteq \big\{ X \subseteq V \,:\, |X| = 2 \big\},$$

den *Kanten* (edges). Eine Kante $e = \{v, w\} \in E$ steht für eine Verbindungen zwischen den verschiedenen Knoten v und w, da $\{v, w\} = \{w, v\}$ gilt, ist dieser Verbindung keine Richtung zugeordnet; sprich die Kante e ist ungerichtet.

Achtung: Wie zumindest in einigen Bereichen in der mathematischen Literatur nicht unüblich, kann die Terminologie in der Graphentheorie variieren. Dabei kann der gleiche Term, je nach Quelle, verschiedene (aber eben auch ähnliche) Bedeutungen annehmen und, umgekehrt verschiedene Terme exakt die gleiche Bedeutung haben. In der Literatur wird zum Beispiel, was hier als Graph bezeichnt wird, auch *ungerichteter Graph* genannt.

Beispiel 1.2 Graph $G = \big(\{v, w, x, y\}, \big\{\{v, w\}, \{w, x\}, \{v, x\}, \{x, y\}\big\}\big)$:



♣

Definition 1.3 Sei $G = (V, E)$ ein Graph. Ein *Weg* in G ist eine Knotenfolge

$$\pi = v_0, v_1, \ldots, v_r$$

mit $r \geq 1$ und $\{v_i, v_{i+1}\} \in E$, $i = 0, 1, \ldots, r-1$. Wir sprechen von einem Weg, der *von v nach w* führt, wenn der Anfangsknoten $v_0 = v$ und der Endknoten $v_r = w$ ist. Die *Länge* $|\pi|$ von π ist r, das heisst, die Anzahl der Kanten, die in π durchlaufen werden. Ein Knoten w heisst von einem Knoten v *erreichbar*, falls ein Weg $\pi = v_0, v_1, \ldots, v_r$ in G existiert, so dass $v = v_0$ und $w = v_r$.

Beachte: In der Literatur wird ein Weg auch *Pfad* genannt.

Beispiel 1.4 Beispiele für Wege in dem Graphen von Beispiel 1.2 sind

- $\pi_1 = v, w$ (Länge 1),
- $\pi_2 = v, w, x$ (Länge 2),
- $\pi_3 = v, w, x, y, x, y$ (Länge 5) und
- $\pi_4 = x, v, w, x, y$ (Länge 4). ♣

Definition 1.5 Sei $G = (V, E)$ ein Graph und ein Weg

$$\pi = v_0, v_1, \dots, v_r$$

mit $r \geq 1$ gegeben, dann nennen wir den Weg π einen *Rundweg*, falls $v_0 = v_r$ gilt.

Definition 1.6 Sei $G = (V, E)$ ein Graph und ein Weg

$$\pi = v_0, v_1, \dots, v_r$$

mit $r \geq 1$ gegeben. Der Weg π heisst *einfach*, falls gilt:

- $v_0, v_1, \dots, v_r$ sind paarweise verschieden
- oder π ist ein Rundweg und $v_0, v_1, \dots, v_{r-1}$ sind paarweise verschieden.

Beispiel 1.7 Der Graph aus Beispiel 1.2 besitzt die einfachen Rundwege $\pi_1 = v, w, x, v$ und $\pi_2 = x, y, x$. Dementgegen ist $\pi_3 = v, w, x, y, x, v$ ein Rundweg, welcher nicht einfach ist. ♣

Definition 1.8 Es sei $G = (V, E)$ ein Graph und $v \in V$. Wir definieren:

- die Menge der *(direkten) Nachbarn* von v

$$\text{post}(v) := \{w \in V : \{v, w\} \in E\},$$

- die Menge aller von v erreichbaren Knoten

$$\text{post}^\star(v) := \{w \in V : \text{es existiert ein Weg von } v \text{ nach } w\}.$$

Beispiel 1.9 Für den Graphen aus Beispiel 1.2 ist

$$\text{post}(x) = \{v, w, y\}, \qquad \text{post}^\star(x) = \{v, w, x, y\}.$$ ♣

Definition 1.10 Es sei $G = (V, E)$ ein Graph und $v \in V$. Der *Grad* von v ist gegeben durch

$$\deg(v) := |\text{post}(v)|.$$

Beachte: Insbesondere gilt

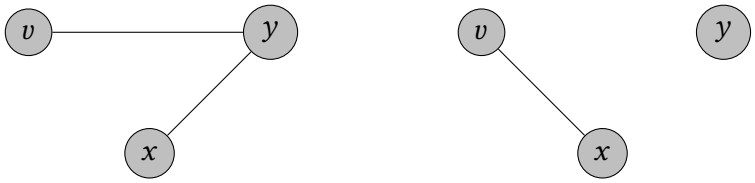
$$\sum_{v \in V} \deg(v) = 2|E|.$$

Definition 1.11 Es sei $G = (V, E)$ ein Graph. Ein Graph $G' = (V', E')$ heisst *Teilgraph von* G, falls $V' \subseteq V$ und $E' \subseteq E$ gelten.

Beachte: Da G' selber ein Graph sein muss, gilt

$$E' \subseteq E \cap \{X \subseteq V' \,:\, |X| = 2\}.$$

Beispiel 1.12 Der Graph rechts ist ein Teilgraph des Graphs von Beispiel 1.2, der links keiner:



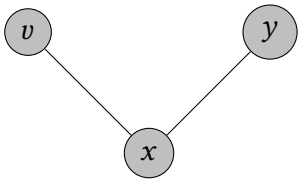
♣

Definition 1.13 Es sei $G = (V, E)$ ein Graph. Für eine Knotenteilmenge $V' \subseteq V$ definieren wir den *von V' induzierten Teilgraphen von G*, durch $G[V'] = (V', E')$ mit

$$E' := E \cap \left\{ X \subseteq V' \,:\, |X| = 2 \right\}.$$

Beachte: Offensichtlich ist jeder induzierter Teilgraph immer ein Teilgraph.

Beispiel 1.14 Der von $\{v, x, y\}$ induzierte Teilgraph des Graphs von Beispiel 1.2 ist:



♣

Definition 1.15 Es seien $G = (V, E)$ und $G' = (V', E')$ zwei Graphen. Die Graphen G und G' heissen *isomorph*, falls es eine Abbildung $\phi : V \to V'$ gibt, wobei

- ϕ bijektiv ist und
- $E' = \left\{ \{\phi(v), \phi(w)\} \,:\, \{v, w\} \in E \right\}$ erfüllt ist.

Eine solche Abbildung ϕ nennen wir einen *Graphenisomorphismus* von G und G'.

Grundlegend am Graphenisomorphismus ist, dass er die Graphenstruktur des einen Graphen exakt auf den anderen abbildet. Das heisst wenn man nur die Graphenstruktur betrachtet kann man zwei isomorphe Graphen nicht unterscheiden: in dem Sinne sie sind also lediglich unterschiedliche Beschreibungen der gleichen abstrakten Graphenstrukur.

Beispiel 1.16 Der Graph $G = \big(\{v, w, x, y\}, \big\{\{v, w\}, \{w, x\}, \{v, x\}, \{x, y\}\big\}\big)$ ist isomorph zu dem Graphen $G' = \big(\{b, d, a, c\}, \big\{\{a, b\}, \{b, d\}, \{a, c\}, \{a, d\}\big\}\big)$. ♣

Um zu überprüfen, ob zwei Graphen isomorph sind, muss man einen Isomorphismus zwischen den Graphen finden. A priori kommt gemäss Definition jede bijektive Abbildung zwischen V und V' potenziell in Frage. Folgende Aussage schränkt die potenziellen bijektiven Abbildung $\phi : V \to V'$ für einen allfälligen Graphenisomorphismus weiter ein.

Satz 1.17 *Seien $G = (V, E)$ und $G' = (V', E')$ zwei isomorphe Graphen. Dann erfüllt jeder Graphenisomorphismus $\phi : V \to V'$*

$$\deg(v) = \deg\big(\phi(v)\big)$$

für alle $v \in V$.

Beweis. Trivial. ♠

1.2 Digraphen

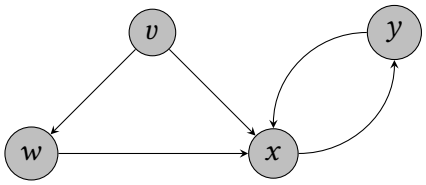
Definition 1.18 Ein *Digraph* ist ein Paar $G = (V, E)$, bestehend aus einer endlicher Menge V von *Knoten* (vertices) und einer endlichen Menge von geordneten Paaren $(v, w) \in V \times V$,

$$E \subseteq \big\{(v, w) \in V \times V \,:\, v \neq w\big\},$$

den *Kanten* (edges). Eine Kante $e = (v, w) \in E$ steht für eine Verbindungen zwischen den verschiedenen Knoten v und w, da $(v, w) \neq (w, v)$ gilt, ist dieser Verbindung eine Richtung zugeordnet; sprich sie ist gerichtet. Wir nennen v den *Anfangsknoten* und w den *Endknoten* der Kante e.

Beachte: In der Literatur wird ein Digraph auch *gerichteter Graph* genannt.

Beispiel 1.19 Digraph $G = \big(\{v, w, x, y\}, \{(v, w), (w, x), (v, x), (x, y), (y, x)\}\big)$:



♣

Bemerkung In Digraphen werden manchmal auch Kanten der Form (v, v) zugelassen. Man spricht dann von *Schleifen*. ♦

Definition 1.20 Sei $G = (V, E)$ ein Digraph. Ein *Weg* in G ist eine Knotenfolge

$$\pi = v_0, v_1, \dots, v_r$$

mit $r \geq 1$ und $(v_i, v_{i+1}) \in E$, $i = 0, 1, \dots, r-1$. Wir sprechen von einem Weg, der *von v nach w* führt, wenn der Anfangsknoten $v_0 = v$ und der Endknoten $v_r = w$ ist. Die *Länge* $|\pi|$ von π ist r, das heisst, die Anzahl der Kanten, die in π durchlaufen werden. Ein Knoten w heisst von einem Knoten v *erreichbar*, falls ein Weg $\pi = v_0, v_1, \dots, v_r$ in G existiert, so dass $v = v_0$ und $w = v_r$.

Beispiel 1.21 Beispiele für Wege in dem Digraphen von Beispiel 1.19 sind

- $\pi_1 = v, w$ (Länge 1),
- $\pi_2 = v, w, x$ (Länge 2) und
- $\pi_3 = v, w, x, y, x, y$ (Länge 5). ♣

Definition 1.22 Sei $G = (V, E)$ ein Digraph und ein Weg

$$\pi = v_0, v_1, \dots, v_r$$

mit $r \geq 1$ gegeben, dann nennen wir den Weg π einen *Rundweg*, falls $v_0 = v_r$ gilt.

Definition 1.23 Sei $G = (V, E)$ ein Digraph und ein Weg

$$\pi = v_0, v_1, \dots, v_r$$

mit $r \geq 1$ gegeben. Der Weg π heisst *einfach*, falls gilt:

- $v_0, v_1, \dots, v_r$ sind paarweise verschieden
- oder π ist ein Rundweg und $v_0, v_1, \dots, v_{r-1}$ sind paarweise verschieden.

Beispiel 1.24 Der Digraph aus Beispiel 1.19 besitzt als einzigen einfachen Rundweg den Rundweg $\pi_1 = x, y, x$ respektive mit y als Anfangsknoten $\pi_1 = y, x, y$. In dem wir den Rundweg π_1 mehrmals aneinander hängen können wir die weiteren Rundwege $\pi_2 = y, x, y, x, y$, $\pi_3 = y, x, y, x, y, x, y$ etc. generieren. Diese sind aber jeweils per definition nicht einfach. Weiter gibt es keinen Rundweg, der die Knoten v oder w enthält. ♣

Definition 1.25 Es sei $G = (V, E)$ ein Digraph und $v \in V$. Wir definieren:

- die Menge der *(direkten) Nachfolger* von v

$$\text{post}(v) := \{w \in V \,:\, (v, w) \in E\},$$

- die Menge der *(direkten) Vorgänger* von v

$$\mathrm{pre}(v) := \{w \in V \,:\, (w,v) \in E\},$$

- die Menge aller von v erreichbaren Knoten

$$\mathrm{post}^\star(v) := \{w \in V \,:\, \text{es existiert ein Weg von } v \text{ nach } w\},$$

- die Menge aller Knoten, die v erreichen können,

$$\mathrm{pre}^\star(v) := \{w \in V \,:\, v \in \mathrm{post}^\star(w)\}.$$

Ein Knoten $w \in \mathrm{post}(v) \cup \mathrm{pre}(v)$ ist ein *Nachbar* von v.

Beachte: $\mathrm{post}(v)$ und $\mathrm{pre}(v)$ können völlig verschieden sein, ebenso $\mathrm{post}^\star(v)$ und $\mathrm{pre}^\star(v)$.

Beispiel 1.26 Für den Digraphen aus Beispiel 1.19 ist

$$\mathrm{post}(v) = \{w, x\}, \qquad \mathrm{post}^\star(v) = \{w, x, y\}, \qquad \mathrm{pre}(v) = \mathrm{pre}^\star(v) = \emptyset$$

und

$$\mathrm{post}(y) = \{x\}, \qquad \mathrm{post}^\star(y) = \{x, y\}.$$ ♣

Definition 1.27 Es sei $G = (V, E)$ ein Digraph und $v \in V$. Der *Eingangsgrad* von v ist gegeben durch

$$\mathrm{indeg}(v) := |\mathrm{pre}(v)|$$

und der *Ausgangsgrad* durch

$$\mathrm{outdeg}(v) := |\mathrm{post}(v)|.$$

Schliesslich definieren wir den *Grad* von v vermittels

$$\deg(v) := \mathrm{indeg}(v) + \mathrm{outdeg}(v).$$

Beachte: Es gilt zwingend:

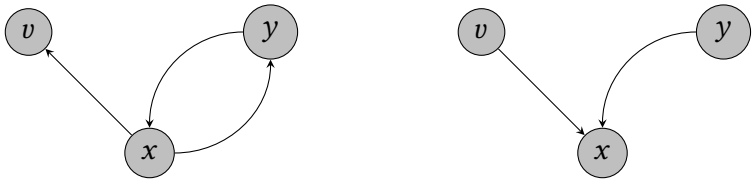
$$\sum_{v \in V} \mathrm{outdeg}(v) = \sum_{v \in V} \mathrm{indeg}(v) = |E|.$$

Definition 1.28 Es sei $G = (V, E)$ ein Digraph. Ein Digraph $G' = (V', E')$ heisst *Teildigraph von* G, falls $V' \subseteq V$ und $E' \subseteq E$ gelten.

Beachte: Da G' selber ein Digraph sein muss, gilt

$$E' \subseteq E \cap \big\{(v,w) \in V' \times V' \,:\, v \neq w\big\}.$$

Beispiel 1.29 Der Digraph rechts ist ein Teildigraph des Digraphs von Beispiel 1.19, der links keiner:



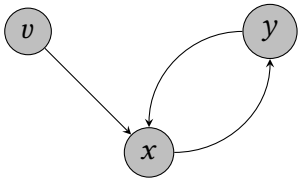
♣

Definition 1.30 Es sei $G = (V, E)$ ein Digraph. Für eine Knotenteilmenge $V' \subseteq V$ definieren wir den *von V' induzierten Teildigraphen von G*, durch $G[V'] = (V', E')$ mit

$$E' := E \cap \big\{(v,w) \in V' \times V' \,:\, v \neq w\big\}.$$

Beachte: Offensichtlich ist jeder induzierter Teildigraph immer ein Teildigraph.

Beispiel 1.31 Der von $\{v, x, y\}$ induzierte Teildigraph des Digraphs von Beispiel 1.19 ist:



♣

Definition 1.32 Es seien $G = (V, E)$ und $G' = (V', E')$ zwei Digraphen. Die Digraphen G und G' heissen *isomorph*, falls es eine Abbildung $\phi : V \to V'$ gibt, wobei

- ϕ bijektiv ist und
- $E' = \big\{(\phi(v), \phi(w)) \,:\, (v,w) \in E\big\}$ erfüllt ist.

Eine solche Abbildung ϕ nennen wir einen *Digraphenisomorphismus* von G und G'.

Satz 1.33 *Seien $G = (V, E)$ und $G' = (V', E')$ zwei isomorphe Digraphen. Dann erfüllt jeder Digraphenisomorphismus $\phi : V \to V'$*

$$\mathrm{indeg}(v) = \mathrm{indeg}\big(\phi(v)\big), \qquad \mathrm{outdeg}(v) = \mathrm{outdeg}\big(\phi(v)\big)$$

für alle $v \in V$.

Beweis. Trivial. ♠

Definition 1.34 Es sei $G = (V, E)$ ein Digraph. Dann ist der von G induzierte Graph G' gegeben durch $G' = (V, E')$ mit

$$E' := \big\{\{v, w\} : (v, w) \in E\}.$$

Beispiel 1.35 Der von dem Digraph von Beispiel 1.19 induzierte Graph ist genau der Graph von Beispiel 1.2. ♣

Definition 1.36 Es sei $G = (V, E)$ ein Graph. Dann ist der von G induzierte Digraph G' gegeben durch $G' = (V, E')$ mit

$$E' := \big\{(v, w) : \{v, w\} \in E\}.$$

Beispiel 1.37 Der von dem Graph von Beispiel 1.2 induzierte Digraph ist nicht der Digraph von Beispiel 1.19. Der Digraph von Beispiel 1.19 ist lediglich ein echter Teildigraph. ♣

1.3 Implementation von Graphen & Digraphen

Die einfachste Art Digraphen im Rechner zu speichern ist die Verwendung von *Adjazenzmatrizen*.

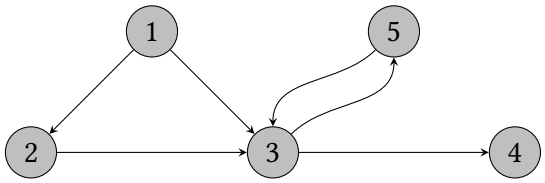
Definition 1.38 Ein Digraph $G = (V, E)$ mit $V = \{1, 2, \dots, n\}$ kann durch eine *Adjazenzmatrix* oder *Nachbarschaftsmatrix* $\mathbf{A} = [a_{i,j}]_{i,j=1}^{n} \in \mathbb{R}^{n \times n}$ mit

$$a_{i,j} = \begin{cases} 1, & \text{falls } (i, j) \in E, \\ 0, & \text{sonst,} \end{cases}$$

dargestellt werden.

Definition 1.39 Ein Graph $G = (V, E)$ mit $V = \{1, 2, \dots, n\}$ kann durch die Adjazenzmatrix des durch G induzierten Digraphen dargestellt werden.

Beispiel 1.40 Der Digraph G



und sein induzierten Graphen G' besitzen die Adjazenzmatrizen

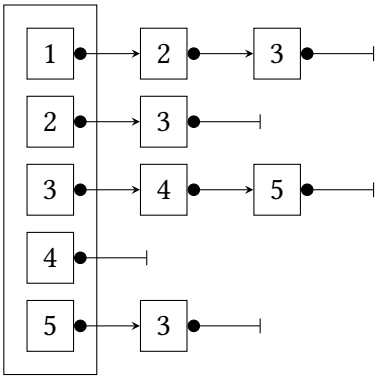
$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \qquad \mathbf{A}' = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}. \qquad \clubsuit$$

Bemerkung Bei Graphen ist $\mathbf{A}$ stets symmetrisch, das heisst es gilt $a_{i,j} = a_{j,i}$ für alle $1 \leq i, j \leq n$. ♦

Die Adjazenzmatrix besitzt immer den Speicherplatzbedarf $|V|^2$, unabhängig von der Anzahl $|E|$ der Kanten. Platzeffizienter ist die Darstellung von Digraphen $G = (V, E)$ durch *Adjazenzlisten*:

Definition 1.41 Die *Adjazenzliste* oder *Nachbarschaftsliste* zu einem Knoten $v \in V$ enthält einen Knoten $w \in V$ genau dann, wenn $w \in \text{post}(v)$ gilt.

Beispiel 1.42 Die Adjazenzlisten zum Digraphen aus Beispiel 1.40 werden wie folgt dargestellt:



♣

Adjazenzlisten können gespeichert werden als *einfach verkettete Listen* mit Elementen der Form

```
struct node
  {  unsigned int  number;
     struct node  *next;
  };
```

Sind `A` und `B` vom Typ `struct node`, so enthält `A.number` beziehungsweise `B.number` die Nummer des Knotens, während die Zuweisung

```
A.next = &B;
```

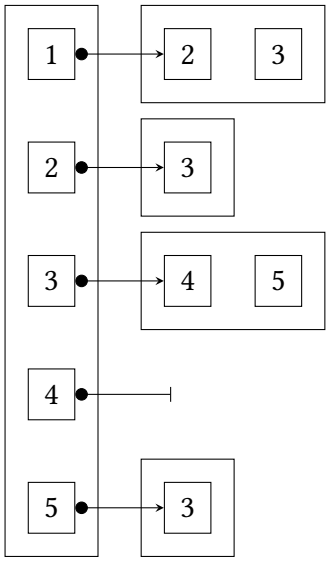
`B` als Nachfolger von `A` definiert. Anstelle von `B` kann man dann auch `A.next` schreiben. Das Ende der Liste wird durch

```
B.next = NULL;
```

markiert. Merken muss man sich jeweils den Anfang der Liste, auch *Listenkopf* (head) genannt. Dies geschieht meist durch ein Feld der Länge $|V|$.

Für Digraphen haben Adjazenzlisten den Seicherplatzbedarf $|V| + |E|$. Für Graphen ist der Platzbedarf $|V| + 2|E|$, da jede Kante in zwei Adjazenzlisten vorkommt.

Bemerkung Eine weitere Möglichkeit ist es, jede Adjazenzliste statt als einfach verkette Liste direkt als Feld zu speichern. Für einen Knoten $v \in V$ muss das Feld dann jeweils die Länge $\text{outdeg}(v)$ haben. Beispielsweise wuerden die Adjazenzlisten zum Digraphen aus Beispiel 1.40 dann wie folgt dargestellt werden:



In MATLAB kann man diese Variante umsetzen in dem man die Adjazenzlisten als Vektoren in ein `cell array` der Länge $|V|$ speichert. ♦

Unter der Annahme, dass der maximale Ausgangsgrad eines Digraphen nicht zu gross ist, kann man anstatt den einfach verketteten Listen oder den Vektoren, die Adjazenzliste auch als Spalten in eine rechteckige Matrix speichern. Dabei hat die Matrix dann $|V|$ Spalten und $\max_{v \in V} \text{outdeg}(v)$ Reihen. Um die Adjazenzlisten alle gleich lang zu machen ergänzt man sie mit Nullen. Für den Digraphen aus Beispiel 1.40 erhält man dann:

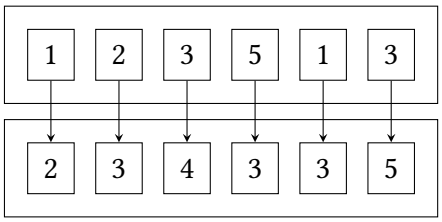
$$\begin{bmatrix} 2 & 3 & 4 & 0 & 3 \\ 3 & 0 & 5 & 0 & 0 \end{bmatrix}.$$

Während wir diese einfache Variante in den Uebungen benutzen werden, ist es auch angebracht zu bemerken, dass es auch raffiniertere Darstellungen von Digraphen gibt, welche insbesondere auch für die Speicherung von dünnvernetzen Digraphen gebräulich sind:

Bemerkung

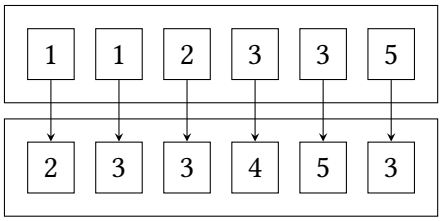
- Beim *coordinate list* (COO) Format speichert man in zwei Felder der Länge $|E|$ jeweils den Anfangsknoten in dem einen Feld und den Endknoten in dem

anderen Feld an der gleichen Stelle und dies für alle Kanten. Für den Digraphen aus Beispiel 1.40 hat man zum Beispiel:

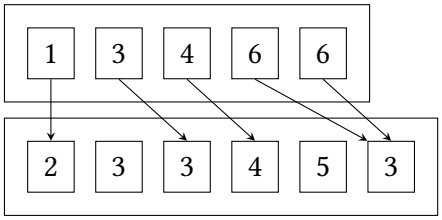


Ein Nachteil dieser Darstellung ist, dass man jeweils Suchen muss, um herauszufinden, ob eine Kante im Graph existiert oder nicht.

- Um das COO Format zu verbessern kann man die Reihenfolge sortieren. Dabei möchte man die Reihenfolge so wählen, dass die Anfangsknoten aufsteigend sortiert sind, und die Endknoten zum gleichen Anfangsknoten auch aufsteigend sortiert sind. Für den Digraphen aus Beispiel 1.40 hat man zum Beispiel:



- Im sortierten COO Format sieht man, dass die Sortierung der Anfangsknoten eine Redundanz offenbart. Es ist nicht nötig die Anfangsknoten als Feld zu merken, sondern es reicht sich für jeden Knoten zu merken, wo der potenziell erste Eintrag in dem Endknotenfeld ist. Dies ergibt genau das *compressed sparse row* (CSR) Format. Für den Digraphen aus Beispiel 1.40 hat man zum Beispiel:



Der Vorteil dieses Formats ist, dass es sehr speichereffizient ist und die Existenz von Kanten auch effizient bestimmt werden kann. Ein Nachteil ist, dass das Einfügen oder Löschen von Kanten generell ineffizient ist. ♦

Bemerkung Leicht modifiziert eignen sich die Formate COO und CSR wie auch andere, um dünn besetzte Matrizen effizient abzuspeichern. Die Indices jedes Nichtnull-Eintrages wird als Kante in einem Digraphen verstanden, wobei nun der Kante der Werst des Eintrages zugeordnet wird. ♦

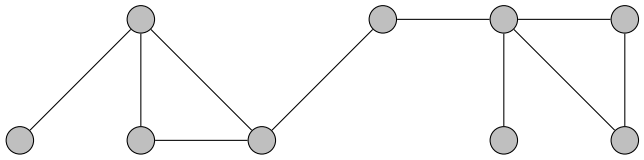
2 Etwas Graphentheorie

2.1 Zusammenhang

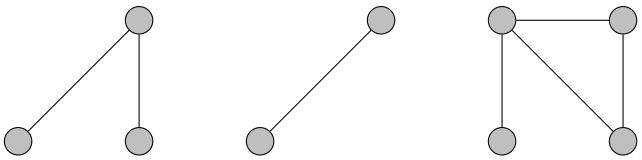
Definition 2.1 Sei $G = (V, E)$ ein Graph und $C \subseteq V$. C heisst *zusammenhängend*, falls je zwei Knoten $v, w \in C$, $v \neq w$, voneinander *erreichbar* sind, das heisst, falls gilt $w \in \text{post}^\star(v)$ (oder äquivalent $v \in \text{post}^\star(w)$). C heisst *Zusammenhangskomponente* von G, falls C eine nicht-leere maximale zusammenhängende Knotenmenge ist. Maximalität bedeutet hier, dass C in keiner anderen zusammenhängenden Menge $C' \subset V$ echt enthalten ist (also $C \neq C'$). Der Graph G heisst zusammenhängend, falls V zusammenhängend ist.

Beispiel 2.2

- Ein zusammenhängender Graph ist:



- Ein unzusammenhängender Graph mit drei Zusammenhangskomponenten ist:



♣

Offenbar sind die Zusammenhangskomponenten eines Graphs die Äquivalenzklassen der Knotenmenge V unter der Erreichbarkeits-Äquivalenzrelation "$\equiv$", wobei

$$v \equiv w \;:\Longleftrightarrow\; \{v\} \cup \text{post}^\star(v) = \{w\} \cup \text{post}^\star(w).$$

Insbesondere zerfällt G in paarweise disjunkte Zusammenhangskomponenten $C_1, C_2, \ldots, C_r$ mit

$$V = \bigcup_{i=1}^{r} C_i, \qquad E = \bigcup_{i=1}^{r} E_i,$$

wobei $E_i := E \cap \{X \subseteq C_i \;:\; |X| = 2\}$.

Satz 2.3 *Sei $G = (V, E)$ ein Graph mit $n = |V| \geq 1$ Knoten sowie $m = |E|$ Kanten, dann gilt: Aus G zusammenhängend folgt $m \geq n - 1$.*

Beweis. Wir beweisen die Aussage durch Induktion nach n. Für $n = 1$ folgt $m = 0 = n - 1$; im Fall $n = 2$ ist G zusammenhängend genau dann, wenn $m = 1 = n - 1$. Wir nehmen nun an, dass $n \geq 3$ und G zusammenhängend ist. Wähle $v \in V$, so dass

$$\deg(v) = \min_{w \in V} \deg(w) =: k.$$

Es gilt $k > 0$, denn sonst wäre v ein isolierter Knoten, was im Widerpruch zu G zusammenhängend steht. Im Fall $k \geq 2$ folgt

$$2m = 2|E| = \sum_{w \in V} \underbrace{\deg(w)}_{\geq k} \geq n \cdot k \geq 2 \cdot n$$

und folglich $m \geq n \geq n - 1$.

Für $k = 1$ ergibt sich die Aussage wie folgt: Es sei $G' = (V', E')$ der induzierte Graph, der durch Streichen des Knotens v sowie der ausgehenden Kante entsteht. Mit G ist auch G' zusammenhängend und nach Induktionsvoraussetzung folgt wegen $|V'| = n-1$ und $|E'| = m-1$

$$m - 1 = |E'| \geq (n - 1) - 1 = n - 2,$$

das heisst $m \geq n - 1$. ♠

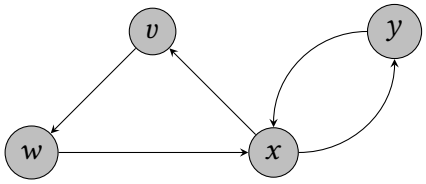
Definition 2.4 Sei $G = (V, E)$ ein Digraph und $C \subseteq V$. Dann heisst C zusammenhängend, falls C in dem von G induzierten Graphen zusammenhängend ist.

Satz 2.5 *Sei $G = (V, E)$ ein Digraph mit $n = |V| \geq 1$ Knoten sowie $m = |E|$ Kanten, dann gilt: Aus G zusammenhängend folgt $m \geq n - 1$.*

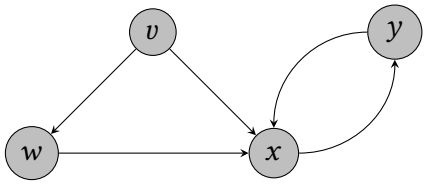
Beweis. Trivial. ♠

Definition 2.6 Ein Digraph $G = (V, E)$ heisst *stark zusammenhängend*, falls für jedes Paar von Knoten $v, w \in V$ mit $v \neq w$ gilt $v \in \text{post}^\star(w)$ und $w \in \text{post}^\star(v)$, das heisst, es gibt einen Weg von v nach w und einen Weg von w nach v. Die *starken Zusammenhangskomponenten* sind die Kontenmengen der maximalen stark zusammenhängenden Teildigraphen.

Beispiel 2.7 Der Digraph



ist stark zusammenhängend. Hingegen besteht der nicht stark zusammenhängender Digraph



aus den starken Zusammenhangskomponenten $\{v\}, \{w\}, \{x, y\}$. ♣

Hinsichtlich der starken Zusammenhangskomponenten ist es oft nützlich, den so genannten *kondensierten Digraphen* zu betrachten. Dieser fasst die Knotenmengen der starken Zusammenhangskomponenten zu einzelnen Knoten zusammen.

Definition 2.8 Für einen Digraphen $G = (V, E)$ seien die starken Zusammenhangskomponenten gegeben durch

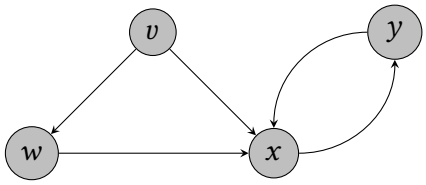
$$G_i := \big(V_i, E \cap (V_i \times V_i)\big) \quad \text{für } i = 1, 2, \dots, p$$

mit $V_1, V_2, \dots, V_p \subseteq V$. Der Digraph $G^\star = (V^\star, E^\star)$ mit

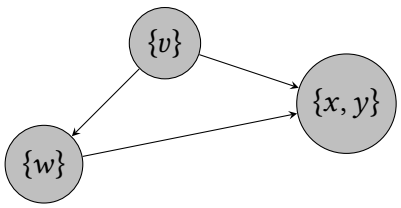
$$V^\star := \{V_1, V_2, \dots, V_p\} \quad \text{und} \quad E^\star := \big\{(V_i, V_j) \in V^\star \times V^\star \,:\, i \neq j \text{ und } E \cap (V_i \times V_j) \neq \emptyset\big\}$$

heisst *kondensierter Digraph* zu G.

Beispiel 2.9 Der Digraph



hat den kondensierten Digraph



bestehend aus den starken Zusammenhangskomponenten $\{v\}, \{w\}, \{x, y\}$. ♣

2.2 Rundweg

Definition 2.10 Ein *Eulerscher Rundweg* in einem Graphen oder Digraphen $G = (V, E)$ ist ein Rundweg, der jede Kante $e \in E$ genau einmal enthält. Ist G ein Graph, so nennen wir G *Eulersch*, falls der Grad jedes Knotens gerade ist. Ein Digraph ist *Eulersch*, falls $\text{indeg}(v) = \text{outdeg}(v)$ für alle $v \in V$ gilt.

Basierend auf dieser Definition haben wir den folgenden berühmten Satz, der Leonhard Euler zugeschrieben wird.

Satz 2.11 *Ein zusammenhängender Graph oder Digraph $G = (V, E)$ besitzt genau dann einen Eulerschen Rundweg, wenn er Eulersch ist.*

Beweis. Die Bedingung ist notwendig, da ein Knoten $v \in V$, der k-mal in einem Eulerschen Rundweg vorkommt (oder $k + 1$ mal, wenn es sich um den Anfangs- und Endknoten handelt), im gerichteten Fall

$$\text{indeg}(v) = \text{outdeg}(v) = k$$

und im ungerichteten Fall

$$\deg(v) = 2k$$

erfüllen muss.

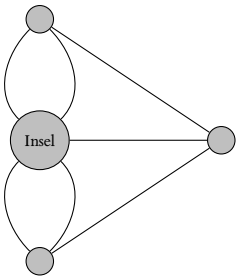
Dass die Bedingung auch hinreichend ist, sieht man wie folgt ein: Sei

$$\pi = v_0, v_1, \dots, v_r$$

der längste Weg, in dem jede Kante aus E höchstens einmal vorkommt. Insbesondere muss in diesem Weg jede Kante enthalten sein, die v_r verlässt. Das bedeutet aber sofort $v_0 = v_r$ wegen der Bedingung an den Knotengrad. Angenommen π enthält nicht alle Kanten, das heisst, es gibt eine Kante $e = (w_1, w_2) \in E$ oder $e = \{w_1, w_2\} \in E$, sodass $e \neq (v_i, v_{i+1})$ beziehungsweise $e \neq \{v_i, v_{i+1}\}$ für alle $i = 0, \dots, r-1$. Da G zusammenhängend ist, muss nun entweder w_1 oder w_2 in π enthalten sein. Ist nun beispielsweise $w_1 = v_i$ in π enthalten, so ist

$$\tilde{\pi} = v_i, v_{i+1}, \dots, v_r, v_1, \dots v_{i-1}, v_i, w_2$$

ein längerer Weg, in dem jede Kante genau einmal vorkommt. ♠



Zu Zeiten Eulers floss durch die Stadt Königsberg der Fluss Pregel. In diesem Fluss gab es eine Insel, hinter der sich der Fluss teilte. Die vier resultierenden Landstücke waren durch insgesamt sieben Brücken verbunden. Hieraus ergab sich die Fragestellung, ob es möglich wäre einen Rundweg zu finden, der jede Brücke nur genau einmal passiert. Die vorliegend Situation ist im vorangestellten *Multigraphen* (hierin sind auch parallele Kanten zugelassen) schematisch dargestellt. Der soeben bewiesene Satz sagt nun aus, dass in der vorliegenden Konfiguration kein solcher Rundweg existiert.

Satz 2.12 *Sei $G = (V, E)$ ein Graph oder Digraph und*

$$\pi = v_0, v_1, \dots, v_r$$

mit $r \geq 1$ ein Weg. Wenn π nicht einfach ist, dann kann π aus einem einfachen Weg gewonnen werden, indem wiederholt einfache Rundwege eingefügt werden. Insbesondere kann daher jeder nicht einfache Rundweg aus einfachen Rundwegen zusammengesetzt werden.

Beweis. Es ist einfach einzusehen, dass jeder Weg der Länge ≤ 2 einfach sein muss. Daher reicht es per Induktion zu zeigen, dass ein nicht einfacher Weg aus einem Weg und einem Rundweg zusammengestzt werden kann, welche beide notwendigerweise eine echt kleinere Länge aufweisen. Sei also

$$\pi = v_0, v_1, \dots, v_r$$

mit $r \geq 3$ ein nicht einfacher Weg. Dann existieren $0 \leq i < j \leq r$, sodass $v_i = v_j$ und insbesondere auch $(i, j) \neq (0, r)$ gelten. Da wir keine Schleifen in Graphen und Digraphen erlauben, muss aber insbesondere auch $j > i + 1$ gelten.

- Ist $i > 0$, so besteht π aus dem Weg

  $$\tilde{\pi} = v_0, \dots, v_i, v_{j+1}, \dots, v_r$$

  und dem Rundweg

  $$\rho = v_i, \dots, v_j.$$

- Andernfalls gelten $i = 0$ und $j \neq r$. Dann besteht π aus dem Weg

  $$\tilde{\pi} = v_0, v_{j+1}, \dots, v_r$$

  und dem Rundweg

  $$\rho = v_0, \dots, v_j.$$

Da in beiden Fällen die Länge von $\tilde{\pi}$ und ρ echt kleiner als die Länge von π ist, sind wir fertig. ♠

Beachte: Ein nicht einfacher Weg ist im Allgemeinen nicht eindeutig aus einem einfachen Weg und einfachen Rundwegen zusammengesetzt.

Beispiel 2.13 Der Rundweg $v_1, v_5, v_3, v_6, v_2, v_5, v_4, v_6, v_1$ lässt isch einerseits durch die Zusammensetzung der einfachen Rundwege v_1, v_5, v_3, v_6, v_1 und v_6, v_2, v_5, v_4, v_6 generieren, aber auch durch v_1, v_5, v_4, v_6, v_1 und v_5, v_3, v_6, v_2, v_5. ♣

2.3 Zyklus

Definition 2.14 Ein einfacher Rundweg heisst genau dann *pathologisch*, wenn er Länge 2 hat und er in einem Graphen ist. Ein Rundweg in einem Graphen oder Digraphen heisst *pathologisch*, wenn jede Art den Rundweg aus einfachen Rundwegen zusammenzusetzen, mindestens einen pathologischen einfachen Rundweg benötigt. Wir nennen einen Rundweg, der nicht pathologisch ist, *Zyklus*.

Beachte: In der Literatur wird statt Zyklus auch *Kreis* benutzt. Per Definition ist klar, dass pathologische Rundweg nur in Graphen aber *nicht* in Digraphen vorkommen können.

Bemerkung Offensichtlich bedeutet diese Definition also:

(i) jeder nicht pathologische einfache Rundweg ist ein Zyklus,

(ii) sind $\pi_1 = v_0, v_1, \dots, v_r$ und $\pi_2 = w_0, w_1, \dots, w_\ell$ Zyklen mit $v_i = w_0 = w_\ell$, so ist auch

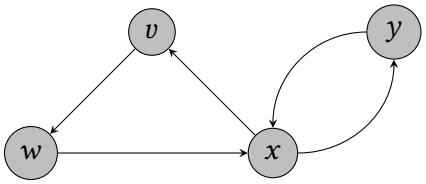
$$\pi = v_0, v_1, \dots, v_{i-1}, w_0, w_1, \dots, w_\ell, v_{i+1}, v_{i+2}, \dots, v_r$$

ein Zyklus,

(iii) nur die durch (i) und (ii) generierbaren Rundwege sind Zyklen. ♦

Beispiel 2.15

- Der Digraph



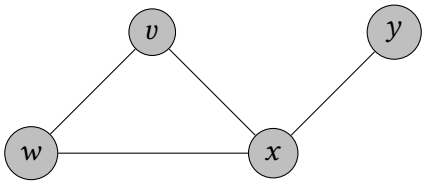
besitzt die einfachen Zyklen

$$\pi_1 = x, y, x, \qquad \pi_2 = v, w, x, v,$$

und die nicht einfachen Zyklen

$$\pi_3 = x, y, x, y, x, \qquad \pi_4 = v, w, x, y, x, v.$$

- Der Graph

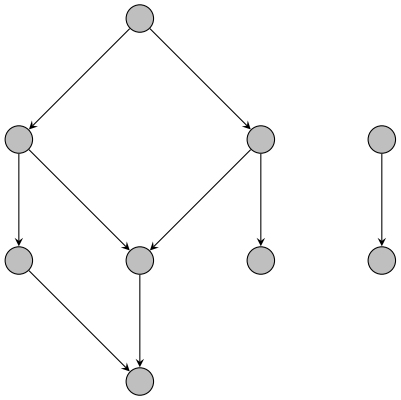


besitzt den (einfachen) Zyklus $\pi_1 = v, w, x, v$. Dementgegen sind die zwei Rundwege $\pi_2 = x, y, x$ und $\pi_3 = v, w, x, y, x, v$ keine Zyklen! ♣

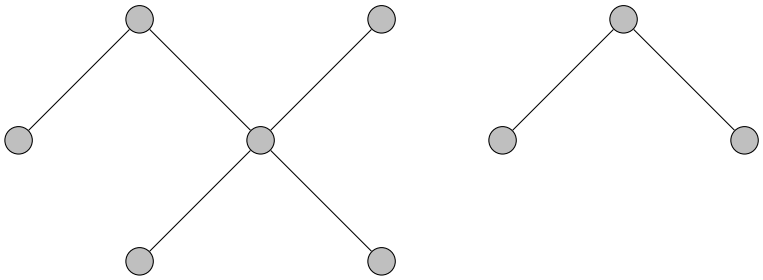
Definition 2.16 Ein Graph oder Digraph G heisst *azyklisch* oder *zyklenfrei*, falls es keine Zyklen in G gibt. Ein azyklischer und zusammenhängender Graph ist ein *Baum*.

Beispiel 2.17

- Azyklischer Digraph:



- Azyklischer Graph:



♣

Satz 2.18 *Sei $G = (V, E)$ ein Graph mit n Knoten. Dann sind folgende Aussagen äquivalent:*

1. *G ist ein Baum.*
2. *G hat $n - 1$ Kanten und ist zusammenhängend.*

3. *G hat $n-1$ Kanten und ist azyklisch.*
4. *G ist azyklisch und das Hinzufügen einer beliebigen Kante erzeugt einen Zyklus.*
5. *G ist zusammenhängend und das Entfernen einer Kante erzeugt einen unzusammenhängenden Graphen.*
6. *Jedes Paar von verschiedenen Knoten in G ist durch genau einen einfachen Weg miteinander verbunden.*

Beweis.

$1 \Rightarrow 6$: Dies folgt aus der Tatsache, dass die Vereinigung zweier disjunkter einfacher Wege mit gleichen Anfangs- und Endpunkten ein Zyklus ist.

$6 \Rightarrow 5$: G ist zusammenhängend gemäss Voraussetzung. Das Entfernen der Kante $\{v, w\}$ macht w unerreichbar von v.

$5 \Rightarrow 4$: G ist azyklisch, denn sonst kann eine Kante entfernt werden, so dass G weiterhin zusammenhängend ist. Da es in G stets einen Weg von v nach w gibt, liefert das Hinzufügen einer Kante $\{v, w\}$ einen Zyklus.

$4 \Rightarrow 3 \Rightarrow 2$: Die Behauptung folgt, falls für einen azyklischen Graphen gilt

$$n = m + p, \tag{2.1}$$

wobei $m = |E|$ und p die Anzahl der Zusammenhangskomponenten ist. Da (2.1) klar ist für $m = 0$, nehmen wir an, (2.1) gilt für ein $|E| = m$. Fügen wir eine zusätzliche Kante hinzu, dies bedeutet $|E| = m + 1$, so muss sich p um eins reduzieren, denn sonst würde ein Zyklus entstehen.

$2 \Rightarrow 1$: Wir zerstören Zyklen aus G durch Entfernen von Kanten. Haben wir etwa k Kanten entfernt, so folgt aus (2.1)

$$\underbrace{n - 1 - k}_{\text{Kanten}} + \underbrace{p}_{=1} = n,$$

das heisst $k = 0$. ♠

3
GRAPHENDURCHMUSTERUNG

Häufig muss ein Graph oder Digraph durchmustert werden. Populäre Graphendurchmusterungsmethoden sind die *Tiefensuche* und die *Breitensuche*. Beide lassen sich auf folgenden Algorithmus zurückführen, der alle von einem Startknoten s erreichbaren Knoten durchsucht. Der Algorithmus sowie die drei nachfolgenden Sätze sind nur für Graphen aufgeführt, sie gelten mit den üblichen Modifikationen aber auch für Digraphen.

Algorithmus 3.1 (algorithmische Suche)
input: Graph $G = (V, E)$ und Startknoten $s \in V$
output: azyklischer Graph $G' = (R, T)$ mit $R = \{s\} \cup \text{post}^\star(s)$ und $T \subseteq E$

① Initialisierung: $R := \{s\}$, $Q := \{s\}$, $T = \emptyset$.

② Falls $Q = \emptyset$ dann `stop`, sonst wähle $v \in Q$.

③ Wähle $w \in V \setminus R$ mit $e = \{v, w\} \in E$. Falls kein solches w existiert, dann setze $Q := Q \setminus \{v\}$ und gehe nach ②.

④ Setze $R := R \cup \{w\}$, $Q := Q \cup \{w\}$, $T := T \cup \{e\}$ und gehe nach ②.

Je nachdem, wie die Menge Q verwaltet wird, ergeben sich verschiedene Resultate. Wir unterscheiden:

Definition 3.2 Bei der *Tiefensuche* oder *Depth-First-Search (DFS)* wird derjenige Knoten $v \in Q$ ausgewählt, der zuletzt zu Q hinzugefügt wurde (Stack-Speicherung). Bei der *Breitensuche* oder *Breadth-First-Search (BFS)* wird derjenige Knoten $v \in Q$ ausgewählt, der zuerst zu Q hinzugefügt wurde (Queue-Speicherung).

Satz 3.3 *Algorithmus 3.1 liefert einen azyklischen Graphen $G' = (R, T)$ mit $R = \{s\} \cup \text{post}^\star(s)$ und $T \subseteq E$.*

Beweis. Zu jedem Zeitpunkt des Algorithmus ist (R, T) zusammenhängend. Insbesondere ist (R, T) azyklisch, denn eine neue Kante $e = \{v, w\}$ verbindet stets Knoten $v \in Q \subseteq R$ und $w \in V \setminus R$.

Angenommen, am Ende existiert ein von s erreichbarer Knoten $w \in V \setminus R$. Dann gibt es einen einfachen s-w-Weg $\pi = s, v_1, \ldots, v_r, w$ in G und weiter eine Kante $\{x, y\} \in E$ aus π mit $x \in R$ und $y \notin R$. Da $x \in R$ ist, muss irgendwann bei Ausführung des Algorithmus $x \in Q$ gelten. Der Algorithmus terminiert jedoch nicht, bevor x aus Q entfernt ist. Dies geschieht aber nur, falls $\{x, y\} \notin E$ gilt. ♠

Satz 3.4 *Die Ausführung von "wähle $v \in Q$" und "wähle $w \in V \setminus R$ mit $e = \{v, w\} \in E$" sei in $\mathcal{O}(1)$ durchführbar. Dann besitzt Algorithmus 3.1 die Komplexität $\mathcal{O}(|V| + |E|)$.*

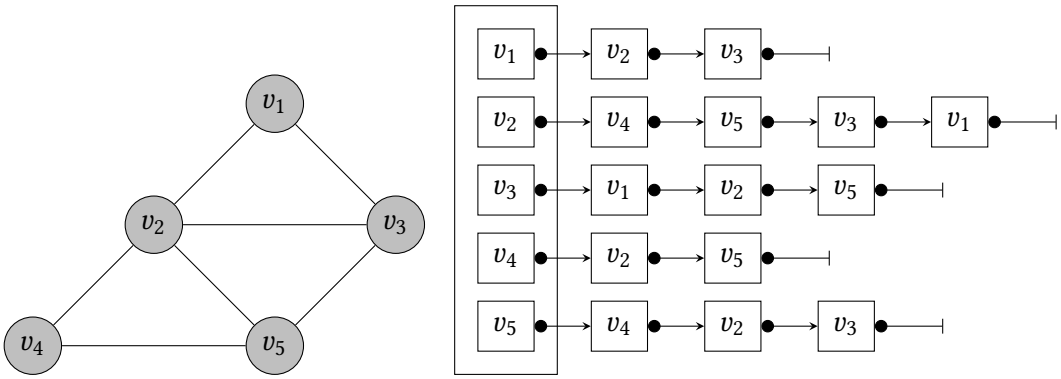
Beweis. Jeder Knoten $v \in V$ wird höchstens $(|\operatorname{post}(v)| + 1)$-mal und jede Kante $e \in E$ höchstens einmal betrachtet. ♠

Bemerkung In der Regel gehen wir davon aus, dass der betrachtete Graph $G = (V, E)$ zusammenhängend ist. Das bedeutet

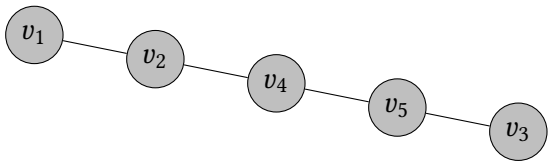
$$|V| - 1 \leq |E| \leq |V|^2.$$

Somit ist die Laufzeit der Graphendurchmusterung immer $\mathcal{O}(|E|)$. ♦

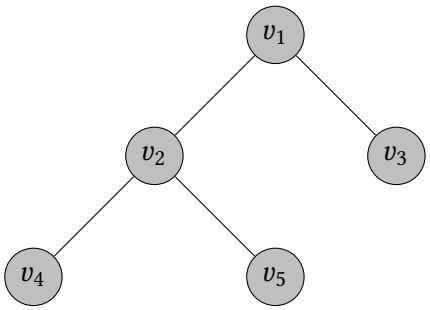
Beispiel 3.5 Der Graph $G = (V, E)$ sei



und $s = v_1$. Bei der Tiefensuche ergibt sich die Besuchsreihenfolge v_1, v_2, v_4, v_5, v_3 und somit der Baum



Die Breitensuche liefert die Besuchsreihenfolge v_1, v_2, v_3, v_4, v_5, das heisst den Baum



♣

Satz 3.6 *Seien $G = (V, E)$ ein Graph, $s, v \in V$ und*

$$\mathrm{dist}_G(s, v) := \min\{|\pi| : \pi = s, u_1, \ldots, u_r, v \text{ Weg in } G\},$$

wobei wir $\mathrm{dist}_G(s, v) := \infty$ setzen, falls kein s-v-Weg existiert. Dann enthält der BFS-Graph $G' = (R, T)$ zum Startknoten $s \in V$ einen kürzesten Weg zu jedem $v \in \mathrm{post}^\star(s)$. Dies bedeutet, der einfache Weg $\pi = s, u_1, \ldots, u_r, v$ in G' erfüllt $|\pi| = \mathrm{dist}_G(s, v)$.

Beweis. Zuerst bemerken wir, dass π eindeutig bestimmt ist, da $G' = (R, T)$ azyklisch ist. Wir modifizieren nun Algorithmus 3.1 wie folgt: In ① setzen wir $\ell(s) := 0$ und in ④ $\ell(w) := \ell(v)+1$. Dann gilt offenbar zu jedem Zeitpunkt

$$\ell(v) = \mathrm{dist}_{(R,T)}(s, v) \quad \text{für alle } v \in R.$$

Weiterhin gibt es für kein $v \in Q$, das in ② ausgewählt wird, ein $w \in R$ mit

$$\ell(w) > \ell(v) + 1. \tag{3.1}$$

Angenommen, der Algorithmus bricht ab und es existiert ein Knoten $w \in V$ mit

$$\mathrm{dist}_G(s, w) < \mathrm{dist}_{G'}(s, w). \tag{3.2}$$

Falls es mehr als einen solchen Knoten gibt, so wählen wir denjenigen mit dem kleinsten Abstand $\mathrm{dist}_G(s, w)$. Sei $\pi = s, u_1, \ldots, u_r, v, w$ ein kürzester Weg in G. Dann gilt $\mathrm{dist}_G(s, v) = \mathrm{dist}_{G'}(s, v)$, da sonst v ein Knoten mit kleinerem Abstand wäre, der (3.2) erfüllt. Ausserdem gilt $\{v, w\} \in E$. Weiter ist

$$\begin{aligned}\ell(w) = \mathrm{dist}_{G'}(s, w) &> \mathrm{dist}_G(s, w) = \mathrm{dist}_G(s, v) + 1 = \mathrm{dist}_{G'}(s, v) + 1 \\ &= \ell(v) + 1.\end{aligned}$$

Gemäss (3.1) gilt $w \notin R$ zu dem Zeitpunkt, zu dem $v \in Q$ entfernt wird. Dies widerspricht jedoch ③, denn $\{v, w\} \in E$. ♠

Bemerkung Gemäss Satz 3.4 berechnet der obige Algorithmus die Werte der Distanzen $\mathrm{dist}_{G'}(s, v) = \mathrm{dist}_G(s, v)$ für alle $v \in R$ in Komplexität $\mathcal{O}(|V| + |E|)$. ♦

Nachfolgender, auf der Tiefensuche basierender Algorithmus bestimmt die starken Zusammenhangskomponenten eines Digraphen.

Algorithmus 3.7 (Bestimmung starker Zusammenhangskomponenten)
input: Digraph $G = (V, E)$
output: eine Funktion $\mathrm{comp} : V \to \mathbb{N}$, die die Zugehörigkeit zu einer starken Zusammenhangskomponente kennzeichnet

① setze $R := \emptyset$, $N := 0$

② für alle $v \in V$:
falls $v \notin R$, dann `visit1`(v)

③ setze $R := \emptyset$, $K := 0$

④ für alle $j = |V|, |V|-1, \dots, 1$:
falls $\iota^{-1}(j) \notin R$, dann setze $K := K+1$ und `visit2`$\left(\iota^{-1}(j)\right)$

Hilfsprogramm `visit1`(v):

① setze $R := R \cup \{v\}$

② für alle $w \in V \setminus R$ mit $(v,w) \in E$: `visit1`(w)

③ setze $N := N+1$, $\iota(v) := N$ und $\iota^{-1}(N) := v$

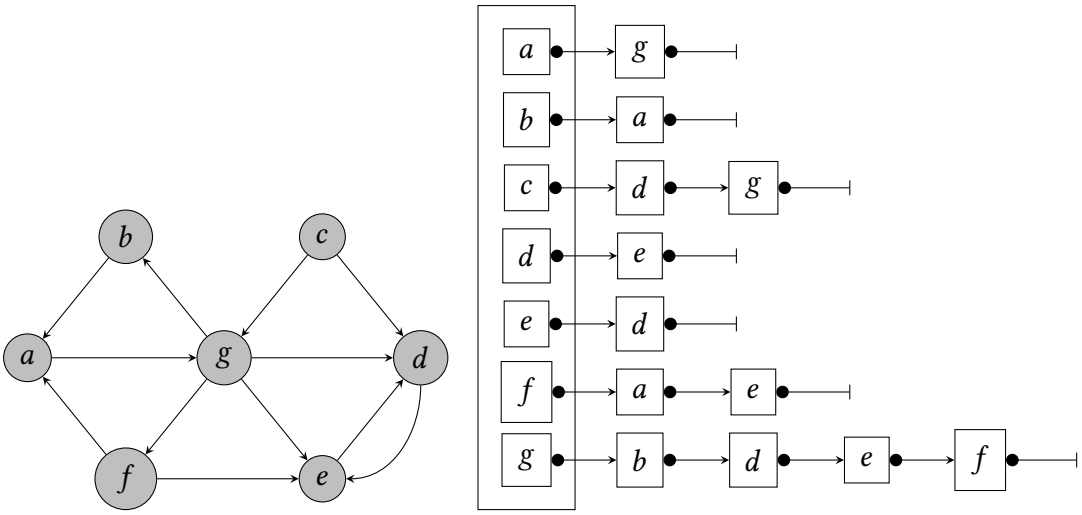
Hilfsprogramm `visit2`(v):

① setze $R := R \cup \{v\}$

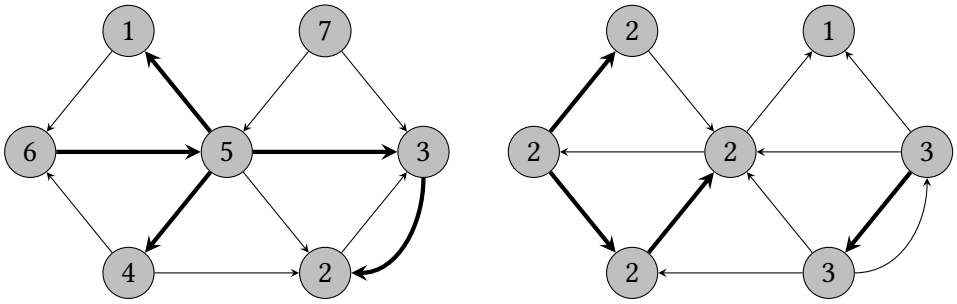
② für alle $w \in V \setminus R$ mit $(w,v) \in E$: `visit2`(w)

③ setze comp$(v) := K$

Beispiel 3.8 Gegeben sei der Digraph



Dann ergibt sich in ② für die erste Tiefensuche `visit1` die Besuchsreihenfolge a, g, b, d, e, f. Als einziger Knoten ist nur noch c nicht in R. Die Tiefensuche für c bricht somit sofort ab und er bekommt die Nummer 7.



Die Tiefensuche `visit2` ist eine Tiefensuche im inversen Digraphen $G^{-1} := (V, E^{-1})$, wobei $E^{-1} := \{(w, v) : (v, w) \in E\}$. In ④ startet die erste Tiefensuche `visit2` mit c, da $\iota(c) = 7$, kann aber keine weiteren Knoten erreichen. Folglich wird nun die Tiefensuche für a gestartet, da $\iota(a) = 6$. Es können b, f, g erreicht werden. Schliesslich wird e von d aus erreicht. Damit ergeben sich die starken Zusammenhangskomponenten $\{c\}$, $\{a, b, f, g\}$, $\{d, e\}$. ♣

Satz 3.9 *Algorithmus 3.7 identifiziert die starken Zusammenhangskomponenten in linearem Aufwand* $\mathcal{O}(|V| + |E|)$.

Beweis. Der Aufwand ergibt sich analog zu Satz 3.4.

Seien nun $v, w \in V$ zwei Knoten derselben starken Zusammenhangskomponente, das heisst, in G gibt es einen Weg von v nach w und umgekehrt. Somit gibt es in G^{-1} ebenfalls einen Weg von v nach w und umgekehrt. Die Tiefensuche `visit2` markiert somit beide Knoten als zur selben Zusammenhangskomponente gehörig, also $\text{comp}(v) = \text{comp}(w)$.

Es verbleibt zu zeigen, dass zwei Knoten $v, w \in V$ mit $\text{comp}(v) = \text{comp}(w)$ auch zur selben starken Zusammenhangskomponente gehören. Dazu sei $r(v)$ beziehungsweise $r(w)$ derjenige von v respektive w erreichbare Knoten mit dem höchsten ι-Wert. Wegen $\text{comp}(v) = \text{comp}(w)$ liegen beide Knoten im selben durch `visit2` erzeugten DFS-Baum. Dessen Startknoten r erfüllt $r = r(v) = r(w)$. Da r von v erreichbar ist und r einen höheren ι-Wert erhalten hat, muss r vor v zu R hinzugefügt worden sein bei der Tiefensuche `visit1`. Daher erhält der entsprechende von `visit1` erzeugte DFS-Baum einen r-v-Weg, das heisst, v ist auch von r erreichbar. Analog ist auch w von r erreichbar. Zusammengefasst ist v von w erreichbar und umgekehrt, was zu zeigen war. ♠

Für azyklische Digraphen bestimmt Algorithmus 3.7 eine spezielle Sortierung der Knoten. Diese werden wir im folgenden benötigen.

Definition 3.10 Es sei $G = (V, E)$ ein Digraph. Eine Numerierung

$$V = \{v_1, v_2, \ldots, v_n\}$$

der Knoten heisst *topologische Ordnung*, falls für alle Kanten $(v_i, v_j) \in E$ gilt $i < j$.

Lemma 3.11 *Der Digraph $G = (V, E)$ besitzt eine topologische Ordnung genau dann, wenn er azyklisch ist.*

Beweis. Übung. ♠

Satz 3.12 *Zu einem Digraphen $G = (V, E)$ bestimmt Algorithmus 3.7 eine topologische Ordnung des kondensierten Digraphen $G^\star = (V^\star, E^\star)$ in linearem Aufwand $\mathcal{O}(|V| + |E|)$.*

Beweis. Seien $V_i, V_j \subseteq V$ die Knotenmengen zweier starker Zusammenhangskomponenten mit $\text{comp}(v_i) = i$, $\text{comp}(v_j) = j$ für alle $v_i \in V_i$, $v_j \in V_j$. Ohne Beschränkung der Allgemeinheit gelte $i < j$. Wir zeigen, dass in G keine Kanten $e = (v_j, v_i) \in E$ existieren mit $v_i \in V_i$, $v_j \in V_j$.

Angenommen eine solche Kante existiert. Alle Knoten aus V_i werden in der Tiefensuche `visit2` vor den Knoten aus V_j zu R hinzugefügt. Insbesondere gilt $v_i \in R$ und $v_j \notin R$ beim überprüfen der Kante $e = (v_j, v_i)$. Dies bedeutet jedoch, dass v_j zu R hinzugefügt wird, bevor K erhöht wird, was $\text{comp}(v_i) \neq \text{comp}(v_j)$ widerspricht. ♠

Korollar 3.13 *Der kondensierte Digraph $G^\star = (V^\star, E^\star)$ zu einem Digraphen $G = (V, E)$ ist zyklenfrei.*

Beweis. Dies folgt sofort aus Lemma 3.11. ♠

Korollar 3.14 *Algorithmus 3.7 bestimmt eine topologische Ordnung des Digraphen $G = (V, E)$ in linearem Aufwand $\mathcal{O}(|V| + |E|)$, falls diese existiert. Gibt es eine solche Ordnung nicht, so erfährt man dies ebenfalls in linearem Aufwand.*

Beweis. Da eine topologische Ordnung nur dann existiert, wenn der Digraph azyklisch ist, ergibt sich eine topologische Ordnung genau dann, wenn alle starken Zusammenhangskomponenten einknotig sind, das heisst $G = G^\star$. ♠

4 Kürzeste-Wege-Probleme

Definition 4.1 Sei $G = (V, E)$ ein Digraph. Eine *Gewichtsfunktion*, manchmal auch *Kostenfunktion* genannt, für die Kanten von G ist eine Abbildung $\omega : E \to \mathbb{R}$. Ist $\pi = v_0, v_1, \ldots, v_r$ ein Weg in G, dann wird der Wert

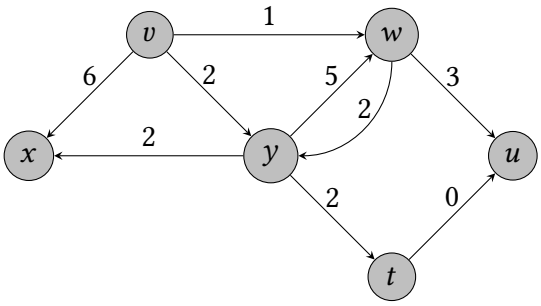
$$\omega(\pi) = \sum_{i=0}^{r-1} \omega(v_i, v_{i+1})$$

die *Weglänge* von π bezüglich w genannt. Das Tripel $G = (V, E, \omega)$ heisst *gewichteter Digraph*.

Definition 4.2 Sei $G = (V, E, \omega)$ ein gewichteter Digraph und $v, w \in V$. Ein *kürzester Weg* von v nach w in G bezüglich ω ist ein v-w-Weg π mit $\omega(\pi) \leq \omega(\pi')$ für jeden v-w-Weg π'. Die *kürzeste Weglänge* $\delta(v, w)$ von v nach w ist definiert durch

$$\delta(v, w) := \begin{cases} \min\{\omega(\pi) : \pi \text{ ist Weg von } v \text{ nach } w\}, & \text{falls ein solcher Weg existiert,} \\ \infty, & \text{sonst.} \end{cases}$$

Beispiel 4.3 Gegeben sei der gewichtete Digraph $G = (V, E, \omega)$:



Ein kürzester Weg von v nach x ist $\pi = v, y, x$. Seine Weglänge ist $\omega(\pi) = 2 + 2 = 4$ und ist kürzer als $\omega(v, x) = 6$. Zum Knoten u gibt es von v zwei kürzeste Wege, nämlich $\pi_1 = v, w, u$ und $\pi_2 = v, y, t, u$ mit $\omega(\pi_1) = \omega(\pi_2) = \delta(v, u) = 4$. ♣

Man unterscheidet verschiedene Varianten des *kürzeste-Wege-Problems*. Gegeben sei stets ein gewichteter Digraph $G = (V, E, \omega)$.

1. **Einzelpaar-kürzeste-Wege-Problem** (*single pair shortest path problem*): Gesucht ist für $v, w \in V$ ein kürzester Weg von v nach w.

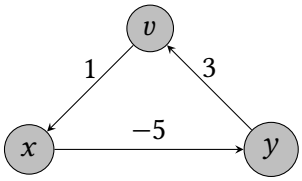
2. **Einzelquelle-kürzeste-Wege-Problem** (*single source shortest path problem*): Für $v \in V$ berechne einen kürzesten Weg zu allen $w \in \text{post}^\star(v)$.

3. **Alle-Paare-kürzeste-Wege-Problem** (*all pair shortest path problem*): Finde für jedes Paar $v, w \in V$ einen kürzesten Weg von v nach w.

Natürlich kann das erste Problem durch das zweite gelöst werden. Es ist auch kein asymptotisch besseres Verfahren bekannt. Aus diesem Grund werden wir gleich dieses allgemeine Problem betrachten.

Negative Gewichte sind gemäss Definition 4.1 zugelassen. Problematisch sind jedoch (einfache) Zyklen mit negativer Weglänge wie folgendes Beispiel zeigt:

Beispiel 4.4 Gegeben sei folgender gewichtete Digraph $G = (V, E, \omega)$:



In diesem Digraphen gibt es keine kürzesten Wege, denn mit zum Beispiel

$$\begin{aligned} \omega(v, x, y) &= -4, \\ \omega(v, x, y, v, x, y) &= -5, \\ \omega(v, x, y, v, x, y, v, x, y) &= -6, \\ &\vdots \end{aligned}$$

können beliebig kurze Wege generiert werden. ♣

Lemma 4.5 *Sei $G = (V, E, \omega)$ ein gewichteter Digraph. Falls es in G keine Zyklen mit negativer Weglänge gibt, dann gibt es für je zwei Knoten $v, w \in V$ mit $w \in \text{post}^\star(v)$ einen kürzesten Weg π mit*

$$\delta(v, w) = \omega(\pi) > -\infty.$$

Beweis. Da es keine negativen Zyklen gibt, genügt es alle einfachen Wege von v nach w zu betrachten. Weil $|V|$ und $|E|$ endlich sind, sind dies nur endlich viele, woraus die Behauptung folgt. ♠

Lemma 4.6 *Sei $G = (V, E, \omega)$ ein gewichteter Digraph ohne negative Zyklen. Weiter sei nun $\pi = v_0, v_1, \ldots, v_{r-1}, v_r$ ein kürzester Weg von v_0 nach v_r. Dann ist für alle $0 \leq i < j \leq r$ der Teilweg $\pi_{i,j} = v_i, v_{i+1}, \ldots, v_j$ von π ein kürzester Weg von v_i nach v_j.*

Beweis. Angenommen, es existiert ein kürzester Weg $\pi'_{i,j}$ von v_i nach v_j mit $\omega(\pi'_{i,j}) < \omega(\pi_{i,j})$. Dann erfüllt der zusammengesetzte Weg $\widehat{\pi} = \pi_{1,i}, \pi'_{i,j}, \pi_{j,r}$ die Abschätzung

$$\begin{aligned}\omega(\widehat{\pi}) &= \omega(\pi_{1,i}) + \omega(\pi'_{i,j}) + \omega(\pi_{j,r}) \\ &< \omega(\pi_{1,i}) + \omega(\pi_{i,j}) + \omega(\pi_{j,r}) \\ &= \omega(\pi).\end{aligned}$$

Dies ist ein Widerspruch zur Voraussetzung, dass π ein kürzester Weg von v_0 nach v_r ist. ♠

Korollar 4.7 *Seien $G = (V, E, \omega)$ ein gewichteter Digraph ohne negative Zyklen und $\pi = v_0, v_1, \dots, v_r$ ein kürzester Weg von v_0 nach v_r. Dann gilt*

$$\delta(v_0, v_r) = \delta(v_0, v_{r-1}) + \omega(v_{r-1}, v_r).$$

Beweis. Gemäss Lemma 4.6 ist $\pi' = v_0, v_1, \dots, v_{r-1}$ ein kürzester Weg von v_0 nach v_{r-1}, das heisst $\omega(\pi') = \delta(v_0, v_{r-1})$. Dies bedeutet

$$\delta(v_0, v_r) = \omega(\pi) = \omega(\pi') + \omega(v_{r-1}, v_r).$$ ♠

Einzelquelle-kürzeste-Wege-Problem im Fall *nicht-negativer* Gewichte:

Algorithmus 4.8 (Dijkstra)

input: Ein gewichteter Digraph $G = (V, E, \omega)$ mit nicht-negativen Gewichten und ein Startknoten $s \in V$.

output: Kürzeste Wege von s zu allen $v \in V$ samt Weglänge $\ell(v)$. Genauer ist $\ell(v)$ die Länge eines kürzesten s-v-Wegs, der aus einem kürzesten s-$p(v)$-Weg und der Kante $\big(p(v), v\big)$ besteht. Für $v \notin \text{post}^\star(s)$ ist $\ell(v) = \infty$ und $p(v)$ undefiniert.

① setze $\ell(s) := 0$ und $\ell(v) := \infty$ für alle $v \in V \setminus \{s\}$, setze $R := \emptyset$

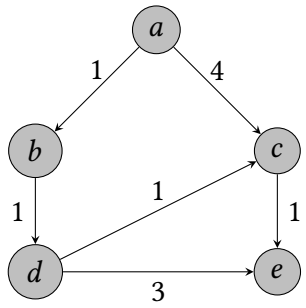
② finde $u \in V \setminus R$ mit $\ell(u) = \min_{v \in V \setminus R} \ell(v)$

③ setze $R := R \cup \{u\}$

④ für alle $v \in V \setminus R$ mit $(u, v) \in E$:
falls $\ell(v) > \ell(u) + \omega(u, v)$, dann
setze $\ell(v) := \ell(u) + \omega(u, v)$ und $p(v) := u$

⑤ falls $R \neq V$ gehe nach ②

Beispiel 4.9 Gegeben sei folgender gewichtete Digraph $G = (V, E, \omega)$:



Ausgehend vom Startknoten ist die Arbeitsweise dieses Algorithmus wie folgt:

Iteration	a	b	c	d	e	u	$\ell(u)$	$p(u)$
0	0	∞	∞	∞	∞	a	0	—
1	—	1	4	∞	∞	b	1	a
2	—	—	4	2	∞	d	2	b
3	—	—	3	—	5	c	3	d
4	—	—	—	—	4	e	4	c

♣

Satz 4.10 (Dijkstra) *Der Algorithmus von Dijkstra arbeitet korrekt, wobei seine Laufzeit $\mathcal{O}(n^2)$ mit $n = |V|$ ist.*

Beweis. Der Übersichtlichkeit halber schreiben wir den Iterationsindex als Suffix an alle Variablen des Dijkstra-Algorithmus. Wir beweisen, dass folgende Aussagen bei jeder Ausführung von ② gelten:

(a) Für alle $v \in R^{(k)}$ und alle $y \in V \setminus R^{(k)}$ gilt $\ell^{(k)}(v) \leq \ell^{(k)}(y)$.

(b) Für alle $v \in R^{(k)}$ ist $\ell^{(k)}(v)$ die kürzeste Weglänge von s nach v. Ist $\ell^{(k)}(v) < \infty$ und $v \neq s$, dann existiert ein kürzester Weg von s nach v mit Knoten aus $R^{(k)}$ und letzter Kante $\big(p^{(k)}(v), v\big)$.

(c) Für alle $v \in V \setminus R^{(k)}$ ist $\ell^{(k)}(v)$ die kürzeste Weglänge von s nach v im aus den Knoten $R^{(k)} \cup \{v\}$ bestehenden Teildigraphen von G. Ist $\ell^{(k)}(v) < \infty$, dann ist $p^{(k)}(v) \in R^{(k)}$ und

$$\ell^{(k)}(v) = \ell^{(k)}\big(p^{(k)}(v)\big) + \omega\big(p^{(k)}(v), v\big).$$

Da diese Aussagen für $k = 0$ gelten, das heisst, nach Ausführung von ①, zeigen wir, dass ③ und ④ die Aussagen erhalten, das heisst, wir zeigen den Induktionsschritt $k \mapsto k + 1$. Dazu sei $k \geq 0$ beliebig und u der in ② ausgewählte Knoten.

Für beliebige $v \in R^{(k)}$ und $y \in V \setminus R^{(k)}$ gilt wegen (a)

$$\ell^{(k+1)}(v) = \ell^{(k)}(v) \leq \ell^{(k)}(u) = \min_{x \in V \setminus R^{(k)}} \ell^{(k)}(x) \leq \ell^{(k+1)}(y).$$

Folglich gilt (a) auch nach ③ und ④, da $R^{(k+1)} = R^{(k)} \cup \{u\}$.

Um zu zeigen, dass (b) nach ④ gilt, müssen wir nur den Knoten u betrachten. Da (c) für k gilt, genügt es zu zeigen, dass in G kein Weg π von s nach u existiert mit einem Knoten $y \in V \setminus R^{(k+1)}$ und $\omega(\pi) < \ell^{(k)}(u) = \ell^{(k+1)}(u)$.

Angenommen, es gibt einen solchen Weg π, etwa

$$\pi = \underbrace{s, v_1, \dots, v_r,}_{\in R^{(k)}} \underbrace{y}_{\notin R^{(k)}}, v_{r+1}, \dots, v_{r+m}, u.$$

Da (c) für k gilt, ist $\ell^{(k)}(y) = \omega(s, v_1, \dots, v_r, y)$, und es folgt wegen der Nicht-Negativität der Gewichte

$$\omega(s, v_1, \dots, v_r, y) \leq \omega(\pi) < \ell^{(k)}(u).$$

Dies bedeutet $\ell^{(k)}(y) < \ell^{(k)}(u)$, was $\ell^{(k)}(u) = \min_{x \in V \setminus R^{(k)}} \ell^{(k)}(x)$ widerspricht.

Nun zeigen wir, dass ③ und ④ auch Aussage (c) erhalten. Falls für ein $v \in V \setminus R^{(k+1)}$ in ④

$$p^{(k+1)}(v) := u, \qquad \ell^{(k+1)}(v) := \ell^{(k+1)}(u) + \omega(u, v)$$

gesetzt wird, muss ein Weg von s nach v existieren im von den Knoten $R^{(k+1)} \cup \{v\}$ aufgespannten Teildigraphen $G'(v)$ von G mit Länge $\ell^{(k+1)}(u) + \omega(u, v)$ und letzter Kante (u, v).

Angenommen, es existiert ein $v \in V \setminus R^{(k+1)}$ und ein Weg π von s nach v in $G'(v)$ mit $\omega(\pi) < \ell^{(k+1)}(v)$. Der Knoten u muss in π enthalten sein, da nur u zu $R^{(k+1)}$ hinzugefügt worden ist und sich sonst ein Widerspruch zu (c) vor Ausführung von ③ und ④ ergäbe (denn $\ell^{(k+1)}(v)$ ist höchstens kleiner als $\ell^{(k)}(v)$ geworden).

Sei x der Vorgänger von v in π. Wegen $x \in R^{(k)} \subset R^{(k+1)}$ folgt aus (a)

$$\ell^{(k+1)}(x) \leq \ell^{(k+1)}(u)$$

und aus ④

$$\ell^{(k+1)}(v) \leq \ell^{(k+1)}(x) + \omega(x, v) \leq \ell^{(k+1)}(u) + \omega(x, v) \leq \omega(\pi).$$

Hierin gilt die letzte Ungleichung, da π den Knoten u enthält und die Kante (x, v). Die Ungleichung

$$\ell^{(k+1)}(v) \leq \omega(\pi)$$

widerspricht jedoch unserer Annahme.

Folglich gelten zu jedem Zeitpunkt k die Aussagen (a)–(c), insbesondere gilt (b) bei Abbruch des Algorithmus, das heisst, der Algorithmus arbeitet korrekt.

Die Aufwandsabschätzung ist offensichtlich: Es werden $n = |V|$ Iterationen ausgeführt, die jeweils einen Aufwand $\mathcal{O}(n)$ haben. ♠

Bemerkung Mit Hilfe so genannter *Fibonacci Heaps* lässt sich der Aufwand des Algorithmus von Dijkstra auf $\mathcal{O}(m + n \log n)$ reduzieren. Hierbei gilt $m = |E|$ und $n = |V|$. ♦

Im Falle von Digraphen mit *negativen* Gewichten, aber ohne negativen Zyklen, muss folgender teurerer Algorithmus verwendet werden. Er ist der schnellste bisher bekannte Algorithmus für dieses Problem.

Algorithmus 4.11 (Moore-Bellman-Ford)
input: Ein gewichteter Digraph $G = (V, E, \omega)$ ohne negative Zyklen und ein Startknoten $s \in V$.
output: Kürzeste Wege von s zu allen $v \in V$ samt Weglänge $\ell(v)$. Genauer ist $\ell(v)$ die Länge eines kürzesten s-v-Wegs, der aus einem kürzesten s-$p(v)$-Weg und der Kante $\big(p(v), v\big)$ besteht. Für $v \notin \mathrm{post}^\star(s)$ ist $\ell(v) = \infty$ und $p(v)$ undefiniert.

① setze $\ell(s) := 0$ und $\ell(v) := \infty$ für alle $v \in V \setminus \{s\}$

② für alle $k = 1, 2, \ldots, n-1$:
  für jede Kante $(u, v) \in E$:
    falls $\ell(v) > \ell(u) + \omega(u, v)$, dann
      setze $\ell(v) := \ell(u) + \omega(u, v)$ und $p(v) := u$

Satz 4.12 (Moore-Bellman-Ford) *Der Moore-Bellman-Ford-Algorithmus arbeitet korrekt, wobei seine Laufzeit $\mathcal{O}(m \cdot n)$ ist mit $m = |E|$ und $n = |V|$.*

Beweis. Die Aufwandsabschätzung ist offensichtlich.

Zu jedem Zeitpunkt des Algorithmus bezeichne

$$R := \{v \in V \,:\, \ell(v) < \infty\},$$
$$F := \{(u, v) \in E \,:\, u = p(v)\},$$

dann zeigen wir:

(a) $\ell(v) \geq \ell(u) + w(u, v)$ für alle $(u, v) \in F$,

(b) der Digraph (R, F) ist azyklisch,

(c) der Digraph (R, F) ist ein gerichteter Baum mit Wurzel s, das heisst, jeder Knoten $v \in R$ ist von s aus über genau einen Weg erreichbar.

Wenn in ② $p(v) := u$ gesetzt wird, dann gilt gerade

$$\ell(v) = \ell(u) + \omega(u, v).$$

Da $\ell(u)$ danach höchstens verkleinert wird, folgt Aussage (a).

Um (b) zu zeigen, nehmen wir an, dass zu einem Zeitpunkt ein Zyklus

$$\pi = v_0, v_1, \ldots, v_{r-1}, v_r, \quad v_0 = v_r$$

entsteht durch Setzen von $p(v_r) := v_{r-1}$. Dann galt aber zuvor

$$\ell(v_0) = \ell(v_r) > \ell(v_{r-1}) + \omega(v_{r-1}, v_r)$$

und gemäss (a)

$$\ell(v_i) \geq \ell(v_{i-1}) + \omega(v_{i-1}, v_i), \quad i = 1, 2, \ldots, r-1.$$

Aufsummieren ergibt

$$\sum_{i=1}^{r} \omega(v_{i-1}, v_i) = \left(\sum_{i=1}^{r-1} \underbrace{\omega(v_{i-1}, v_i)}_{\leq \ell(v_i) - \ell(v_{i-1})} \right) + \underbrace{\omega(v_{r-1}, v_r)}_{< \ell(v_r) - \ell(v_{r-1})} < \sum_{i=1}^{r} \big(\ell(v_i) - \ell(v_{i-1}) \big) = 0,$$

das heisst, der Zyklus ist negativ, was der Voraussetzung widerspricht.

Schliesslich folgt aus $x \in R \setminus \{s\}$ auch $p(x) \in R$, dies ist die Aussage (c).

Gemäss (a)–(c) ist also zu jedem Zeitpunkt $\ell(y)$ mindestens die Länge des (eindeutigen) s-y-Wegs im Digraphen (R, F). Wir zeigen nun, dass nach k Iterationen $\ell(y)$ auch höchstens die Länge eines kürzesten s-y-Wegs in G mit höchstens k Kanten ist. Da für $k = 1$ die Aussage klar ist, nehmen wir an, sie gilt auch nach Iteration $k-1$. Nun sei

$$\pi = s, v_1, \ldots, v_r, x, y, \quad r \leq k-2$$

ein kürzester s-y-Weg in G mit höchstens k Kanten. Dann ist $\pi' = s, v_1, \ldots, v_r, x$ ein kürzester s-x-Weg mit höchstens $k-1$ Kanten. Nach Induktionsannahme folgt $\ell(x) \leq \omega(\pi')$ und somit

$$\ell(y) \leq \ell(x) + \omega(x, y) \leq \omega(\pi') + \omega(x, y) = \omega(\pi).$$

Da kein Weg in (R, F) mehr als $n-1$ Kanten besitzt, impliziert dies die Korrektheit des Algorithmus. ♠

Wir betrachten schliesslich einen Algorithmus zur Lösung des Alle-Paare-kürzeste-Wege-Problems. Ohne Einschränkung der Allgemeinheit sei dabei die Knotenmenge $V = \{1, 2, \ldots, n\}$.

Algorithmus 4.13 (Floyd-Warshall)

input: Ein gewichteter Digraph $G = (V, E, \omega)$ mit $V = \{1, 2, \ldots, n\}$ ohne negative Zyklen.

output: Matrizen $[\ell_{i,j}]_{1 \leq i,j \leq n}$ und $[p_{i,j}]_{1 \leq i,j \leq n}$ mit der kürzesten Weglänge $\ell_{i,j}$ von i nach j und der letzten Kante $(p_{i,j}, j)$ eines i-j-Wegs, falls ein solcher existiert.

① setze $\ell_{i,j} := \omega(i, j)$ für alle $(i, j) \in E$
setze $\ell_{i,j} := \infty$ für alle $(i, j) \in (V \times V) \setminus E$ mit $i \neq j$
setze $\ell_{i,i} := 0$ für alle $i \in V$
setze $p_{i,j} := i$ für alle $i, j \in V$

② für alle $j = 1, 2, \ldots, n$:
  für alle $i = 1, 2, \ldots, n$ mit $i \neq j$:
    für alle $k = 1, 2, \ldots, n$ mit $k \neq j$:
    falls $\ell_{i,k} > \ell_{i,j} + \ell_{j,k}$, dann
      setze $\ell_{i,k} := \ell_{i,j} + \ell_{j,k}$ und $p_{i,k} := p_{j,k}$

Satz 4.14 (Floyd-Warshall) *Der Algorithmus von Floyd und Warshall arbeitet korrekt, wobei seine Laufzeit $\mathcal{O}(n^3)$ mit $n = |V|$ ist.*

Beweis. Die Aussage zur Laufzeit ist klar. Wir schreiben wieder den Iterationsindex j als Suffix an alle Variablen und zeigen: Nach j_0 äusseren Iterationen ist $\ell_{i,k}^{(j_0)}$ die Länge eines kürzesten i-k-Wegs bestehend nur aus den Zwischenknoten $v \in \{1, \dots, j_0\}$ und mit Endkante $(p_{i,k}^{(j_0)}, k)$. Diese Aussage beweisen wir mit vollständiger Induktion über j_0.

Für $j_0 = 0$ gilt sie gemäss ① und für $j_0 = n$ impliziert sie die Korrektheit des Algorithmus. Wir nehmen an, dass obige Aussage gilt für ein $j_0 \in \{0, 1, \dots, n-1\}$, das heisst, für alle $i, k \in V$ ist $\ell_{i,k}^{(j_0)}$ die Länge eines kürzesten i-k-Wegs bestehend nur aus Zwischenknoten $v \in \{1, \dots, j_0\}$.

In der (j_0+1)-ten Iteration wird $\ell_{i,k}^{(j_0)}$ durch $\ell_{i,j_0+1}^{(j_0)} + \ell_{j_0+1,k}^{(j_0)}$ ersetzt, falls dieser Wert kleiner ist. Es verbleibt daher zu zeigen, dass dann die entsprechenden Wege

$$\pi_1 = i, u_1, \dots, u_r, j_0 + 1,$$
$$\pi_2 = j_0 + 1, v_1, \dots, v_s, k$$

keine gemeinsamen inneren Knoten haben.

Angenommen beide Wege haben den gemeinsamen Knoten $u_p = v_q$, dann ist

$$\pi = i, u_1, \dots, u_p, v_{q+1}, \dots, v_s, k$$

ein Weg von i nach k, bestehend nur aus Knoten $v \in \{1, \dots, j_0\}$. Wegen

$$\omega(u_p, u_{p+1}, \dots, u_r, j_0 + 1, v_1, \dots, v_q) \geq 0$$

ist

$$\ell_{i,k}^{(j_0)} \leq \omega(\pi) \leq \omega(\pi_1 \cup \pi_2) \leq \ell_{i,j_0+1}^{(j_0)} + \ell_{j_0+1,k}^{(j_0)},$$

was ein Widerspruch zu $\ell_{i,k}^{(j_0)} > \ell_{i,j_0+1}^{(j_0)} + \ell_{j_0+1,k}^{(j_0)}$ ist. ♠

5 Netzwerkflussprobleme

Motivation: In den Seehäfen $A_1, A_2, \ldots, A_p$ liegen $r_1, r_2, \ldots, r_p$ Tonnen Bananen zum Verschiffen bereit. In den Zielhäfen $B_1, B_2, \ldots, B_q$ besteht die Nachfrage nach $d_1, d_2, \ldots, d_q$ Tonnen. Die Kapazität der Schifffahrtslinie von Hafen A_i nach Hafen B_j ist maximal $c(A_i, B_j)$.

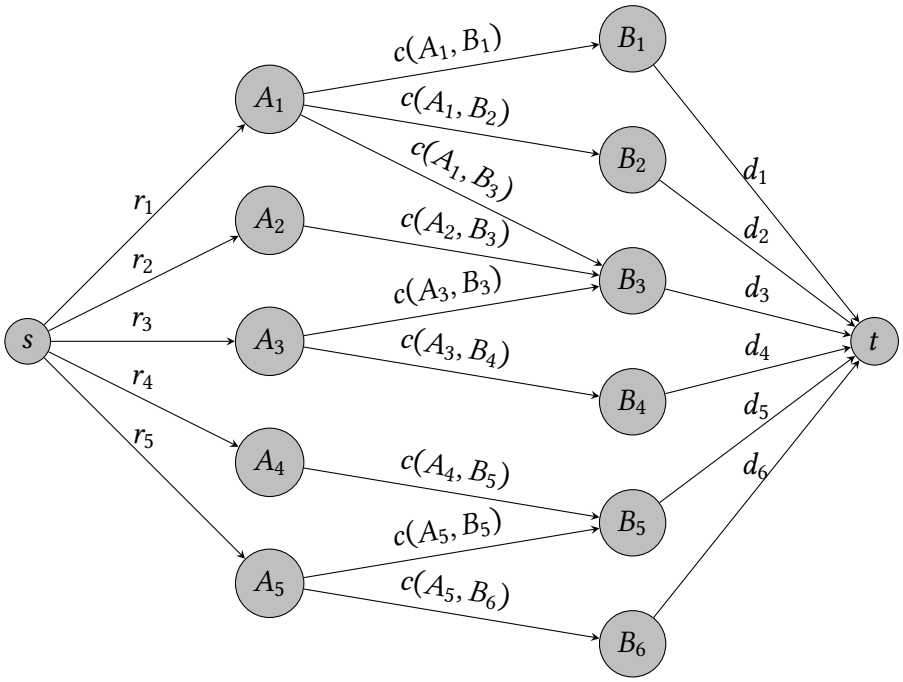
Es stellen sich die folgenden Fragen:

1. Ist es möglich, alle Anforderungen zu befriedigen?
2. Falls nein, wie viele Bananen können maximal zu den Zielhäfen gebracht werden?
3. Von wo nach wo sollen wieviele Bananen verschifft werden?

Zur Lösung konstruieren wir einen Digraphen $G = (V, E)$ mit

$$V = \{A_1, A_2, \ldots, A_p, B_1, B_2, \ldots, B_q\}, \qquad E = \{(A_i, B_j) \,:\, 1 \leq i \leq p,\ 1 \leq j \leq q\}.$$

Der Kante (A_i, B_j) ordnen wir die Kapazität $c(A_i, B_j)$ zu. Um Angebots- und Nachfragemengen zu modellieren, führen wir zwei weitere Knoten s, t und Kanten (s, A_i) beziehungsweise (B_j, t) mit Kapazität $c(s, A_i) = r_i$ beziehungsweise $c(B_j, t) = d_j$ ein.



Zur Beantwortung der drei Fragen lösen wir folgendes Problem: Was ist der maximale Fluss von s nach t in G und wie sieht dieser aus? Dabei ist der Fluss auf einer Kante durch ihre Kapazität beschränkt. Des Weiteren muss der gesamte Fluss, der einen Knoten A_i oder B_j betritt, diesen auch wieder verlassen.

Definition 5.1 Ein *Netzwerk* ist ein Tupel $N = (V, E, c, s, t)$ bestehend aus

- einem Digraphen $G = (V, E)$,
- einer *Kapazitätsfunktion* $c : E \to \mathbb{R}_{\geq 0}$,
- einer *Quelle* $s \in V$ mit $\operatorname{pre}(s) = \emptyset$,
- einer *Senke* $t \in V$ mit $\operatorname{post}(t) = \emptyset$.

Ein *Fluss* $f : E \to \mathbb{R}_{\geq 0}$ ist eine Funktion, die folgende Bedingungen erfüllt:

1. *Kapazitätsbedingung*:

$$f(v, w) \leq c(v, w) \quad \text{für alle } (v, w) \in E,$$

2. *Kirchhoffsches Gesetz*:

$$\sum_{u \in \operatorname{pre}(v)} f(u, v) = \sum_{w \in \operatorname{post}(v)} f(v, w) \quad \text{für alle } v \in V \setminus \{s, t\}.$$

Der *Wert* des Flusses ist

$$\operatorname{flow}(f) = \sum_{w \in \operatorname{post}(s)} f(s, w).$$

Definition 5.2 Der *maximale Fluss* eines Netzwerkes N ist gegeben durch

$$\operatorname{MaxFlow}(N) := \max\{\operatorname{flow}(f) : f \text{ ist Fluss für } N\}.$$

Eine Flussfunktion f für N wird *optimal* genannt, falls

$$\operatorname{flow}(f) = \operatorname{MaxFlow}(N).$$

Definition 5.3 Ein *Schnitt* für ein Netzwerk $N = (V, E, c, s, t)$ ist eine Knotenmenge $S \subseteq V$ mit $s \in S$ und $t \notin S$. Die *Kapazität* eines Schnitts ist gegeben durch

$$\operatorname{cap}(S) = \sum_{\substack{v \in S \\ w \in \operatorname{post}(v) \setminus S}} c(v, w).$$

Die *minimale Schnittkapazität* von N ist

$$\operatorname{MinCut}(N) := \min\{\operatorname{cap}(S) : S \text{ ist Schnitt für } N\}.$$

Lemma 5.4 *Sei S ein Schnitt eines Netzwerkes $N = (V, E, c, s, t)$, dann gilt für jeden Fluss f*

(i) $$\operatorname{flow}(f) = \sum_{\substack{v \in S \\ w \in \operatorname{post}(v) \setminus S}} f(v, w) - \sum_{\substack{v \in S \\ u \in \operatorname{pre}(v) \setminus S}} f(u, v),$$

(ii) $\operatorname{flow}(f) \leq \operatorname{cap}(S)$.

Beweis. Aussage (i) folgt aus dem Kirchhoffschen Gesetz

$$\begin{aligned}
\operatorname{flow}(f) &= \sum_{w \in \operatorname{post}(s)} f(s, w) \\
&= \sum_{v \in S} \left(\sum_{w \in \operatorname{post}(v)} f(v, w) - \sum_{w \in \operatorname{pre}(v)} f(u, v) \right) \\
&= \sum_{\substack{v \in S \\ w \in \operatorname{post}(v) \setminus S}} f(v, w) - \sum_{\substack{v \in S \\ u \in \operatorname{pre}(v) \setminus S}} f(u, v) \\
&+ \underbrace{\sum_{\substack{v \in S \\ w \in \operatorname{post}(v) \cap S}} f(v, w) - \sum_{\substack{v \in S \\ u \in \operatorname{pre}(v) \cap S}} f(u, v)}_{=0}
\end{aligned}$$

Da $0 \leq f(e) \leq c(e)$ für alle $e \in E$, folgt weiter

$$\operatorname{flow}(f) \overset{\text{(i)}}{\leq} \sum_{\substack{v \in S \\ w \in \operatorname{post}(v) \setminus S}} f(v, w) \leq \sum_{\substack{v \in S \\ w \in \operatorname{post}(v) \setminus S}} c(v, w) = \operatorname{cap}(S),$$

dies ist Aussage (ii). ♠

Satz 5.5 (Max-Flow-Min-Cut-Theorem) *Sei $N = (V, E, c, s, t)$ ein Netzwerk, dann gilt*

$$\operatorname{MinCut}(N) = \operatorname{MaxFlow}(N).$$

Beweis. Aus Lemma 5.4 folgt $\operatorname{MaxFlow}(N) \leq \operatorname{MinCut}(N)$. Daher genügt es zu zeigen, dass ein Schnitt S existiert mit $\operatorname{MaxFlow}(N) = \operatorname{cap}(S)$. Hierzu geben wir eine Prozedur an, die für einen gegebenen Fluss f mit $\operatorname{flow}(f) = \operatorname{MaxFlow}(N)$ einen Schnitt S mit $\operatorname{cap}(S) = \operatorname{flow}(f)$ konstruiert.

Wir starten mit $S = \{s\}$. In jedem Schritt erweitern wir S um einen Knoten $y \in V \setminus S$, der benötigt wird, damit die Behauptung überhaupt erfüllt sein kann:

① setze $S := \{s\}$

② solange $x \in S$, $y \in V \setminus S$ existieren mit

$$\begin{aligned}
c(x, y) &> f(x, y), \text{ falls } (x, y) \in E, \\
f(y, x) &> 0, \qquad \text{falls } (y, x) \in E,
\end{aligned}$$

setze $S := S \cup \{y\}$.

Wir zeigen zunächst, dass S stets ein Schnitt für N ist, das heisst, es gilt stets $t \notin S$.

Angenommen, es gilt $t \in S$, dann gibt es einen Knoten $v_{r-1} \in S$, der dafür verantwortlich ist, dass $t = v_r$ zu S hinzugenommen wurde, das heisst, $c(v_{r-1}, v_r) > f(v_{r-1}, v_r)$ oder $f(v_r, v_{r-1}) > 0$. Genauso gibt es einen Knoten $v_{r-2} \in S$, der dafür verantwortlich ist, dass v_{r-1} hinzugenommen worden ist, usw. Folglich existiert ein ungerichteter Weg

$$\pi = v_0, v_1, \dots, v_r, \quad v_i \in S \text{ für alle } 0 \leq i \leq r,$$

wobei $v_0 = s$. Setzen wir für alle $i = 0, 1, \dots, r-1$

$$\varepsilon_i := \begin{cases} c(e) - f(e), & \text{falls } e = (v_i, v_{i+1}) \in E \text{ und } e^{-1} = (v_{i+1}, v_i) \notin E, \\ f(e^{-1}), & \text{falls } e = (v_i, v_{i+1}) \notin E \text{ und } e^{-1} = (v_{i+1}, v_i) \in E, \\ \max\{c(e) - f(e), f(e^{-1})\}, & \text{falls } e = (v_i, v_{i+1}) \in E \text{ und } e^{-1} = (v_{i+1}, v_i) \in E, \end{cases} \tag{5.1}$$

so folgt nach Konstruktion stets $\varepsilon_i > 0$. Wir setzen

$$\varepsilon := \min_{0 \leq i < r} \varepsilon_i > 0. \tag{5.2}$$

und führen einen Widerspruch herbei, indem wir nun einen Fluss $f^\star$ konstruieren mit

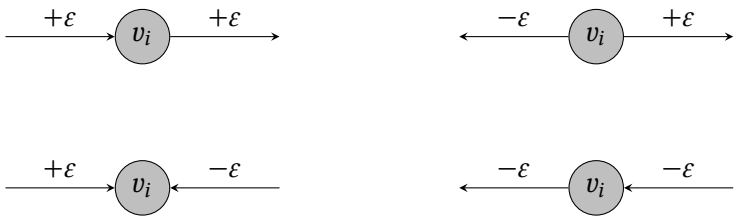
$$\text{flow}(f^\star) = \text{MaxFlow}(N) + \varepsilon.$$

Hierzu definieren wir $f^\star$ für alle $0 \leq i < r$ wie folgt:

$$f^\star(e) := f(e) + \varepsilon, \quad \text{falls } e = (v_i, v_{i+1}) \in E \text{ und } e^{-1} = (v_{i+1}, v_i) \notin E,$$
$$f^\star(e^{-1}) := f(e^{-1}) - \varepsilon, \text{ falls } e = (v_i, v_{i+1}) \notin E \text{ und } e^{-1} = (v_{i+1}, v_i) \in E.$$

Gilt $e = (v_i, v_{i+1}) \in E$ und $e^{-1} = (v_{i+1}, v_i) \in E$, so erhöhen wir $f(e)$ um ε falls $c(e) - f(e) > f(e^{-1})$, ansonsten verringern wir $f(e^{-1})$ um ε.

(5.1), (5.2) garantieren, dass $f^\star$ die Kapazitätsbedingung nicht verletzt. Das Kirchhoffsche Gesetz bleibt beim Übergang $f \mapsto f^\star$ erhalten, da es nur folgende vier Möglichkeiten der Flussänderung pro Knoten v_i gibt:



Also ist $f^\star$ ein Fluss. Weiter gilt

$$\begin{aligned} \text{flow}(f^\star) &= \sum_{v \in \text{post}(s)} f^\star(s, v) \\ &= \sum_{u \in \text{pre}(t)} f^\star(u, t) \\ &= \sum_{u \in \text{pre}(t) \setminus \{v_{r-1}\}} f(u, t) + \underbrace{f^\star(v_{r-1}, t)}_{= f(v_{r-1}, t) + \varepsilon} \\ &= \text{flow}(f) + \varepsilon, \end{aligned}$$

was ein Widerspruch zu $\text{flow}(f) = \text{MaxFlow}(N)$ ist. Damit ist S ein Schnitt in N und gemäss Konstruktion gilt für alle $x \in S, y \in V \setminus S$ dass $f(x,y) = c(x,y)$ beziehungsweise $f(y,x) = 0$. Damit folgt $\text{flow}(f) = \text{cap}(S)$. ♠

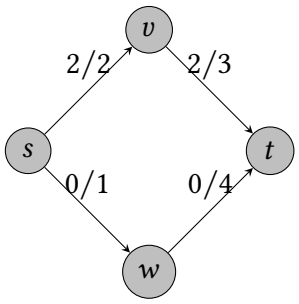
Der im Beweis des Max-Flow-Min-Cut-Theorems konstruierte Weg π heisst *augmentierter Weg*.

Definition 5.6 Sei f ein Fluss im Netzwerk $N = (V, E, c, s, t)$. Eine Kante $e = (x,y) \in E$ heisst *Vorwärtskante*, falls $f(e) < c(e)$. Eine Kante $e = (x,y)$ mit $e^{-1} = (y,x) \in E$ heisst *Rückwärtskante*, falls $f(e^{-1}) > 0$. Der *Restdigraph* für f ist der Digraph $G' = (V, E')$ mit

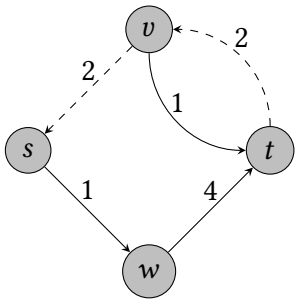
$$E' = \{(x,y) \in V \times V \,:\, (x,y) \text{ ist Vorwärts- oder Rückwärtskante}\}.$$

Die Grössen $c(e) - f(e)$ beziehungsweise $f(e^{-1})$ heissen *Restkapazitäten*. Ein *augmentierter Weg* $\pi = v_0, v_1, \ldots, v_r$ ist ein Weg im Restdigraphen mit $v_0 = s$ und $v_r = t$.

Beispiel 5.7 Gegeben sei folgendes Netzwerk mit Fluss/Kapazitäten:



Der Restdigraph ist



wobei Vorwärtskanten durch durchgezogene und Rückwärtskanten durch gestrichelte Pfeile markiert sind. ♣

Im Fall, dass der Fluss f nicht maximal ist, kann mit Hilfe eines augmentierten Weges der Fluss vergrössert werden. Somit erhalten wir folgenden Algorithmus:

Algorithmus 5.8 (Ford-Fulkerson)
input: Netzwerk $N = (V, E, c, s, t)$
output: Fluss f mit $\text{flow}(f) = \text{MaxFlow}(N)$

① Setze $f(e) = 0$ für alle $e \in E$.

② Suche einen augmentierten Weg π von s nach t. Falls keiner existiert, dann `stop`.

③ Berechne ε gemäss (5.1), (5.2). Augmentiere f um ε und gehe nach ②.

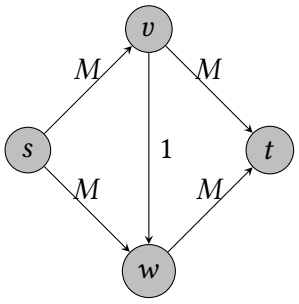
Ford und Fulkerson haben anhand eines Beispiels gezeigt, dass bei irrationalen Kapazitäten der Algorithmus möglicherweise nicht terminiert. Im Fall ganzzahliger Kapazitäten ist dies jedoch nicht der Fall.

Satz 5.9 (Integral-Flow-Theorem) *Sei $N = (V, E, c, s, t)$ ein Netzwerk mit ganzzahligen Kapazitäten. Dann terminiert der Ford-Fulkerson-Algorithmus nach maximal $\sum_{e \in E} c(e)$ Augmentierungsschritten mit einem ganzzahligen maximalen Fluss.*

Beweis. Da alle Kapazitäten ganzzahlig sind und wir mit dem Nullfluss starten, ist während der Durchführung des Algorithmus $\text{flow}(f)$ stets ganzzahlig. Da ein Augmentierungsschritt die Grösse des Flusses mindesten um 1 erhöht, ergibt sich die Behauptung. ♠

Auch bei ganzzahligen Kapazitäten kann der Ford-Fulkerson-Algorithmus viele Augmentierungsschritte benötigen:

Beispiel 5.10 Wie betrachten das Netzwerk



Offensichtlich gilt $\text{MaxFlow}(N) = 2M$. Starten wir mit dem Nullfluss und augmentieren stets entlang eines s-t-Wegs der Länge 3, erhöht sich $\text{flow}(f)$ jeweils nur um $\varepsilon = 1$. Folglich werden $2M$ Schritte benötigt. ♣

Das Beispiel zeigt, dass bei willkürlicher Wahl des augmentierenden Wegs (also Wege von s nach t im Restdigraphen) die Anzahl der Augmentierungsschritte sehr gross sein kann. Polynomielle Laufzeitbeschränkung im Ford-Fulkerson-Algorithmus kann durch die Wahl eines *kürzesten* augmentierten Weg erreicht werden. Hierbei bezieht sich die Länge auf die Anzahl der Kanten.

Algorithmus 5.11 (Edmonds-Karp)
input: Netzwerk $N = (V, E, c, s, t)$
output: Fluss f mit $\text{flow}(f) = \text{MaxFlow}(N)$

① Setze $f(e) = 0$ für alle $e \in E$.

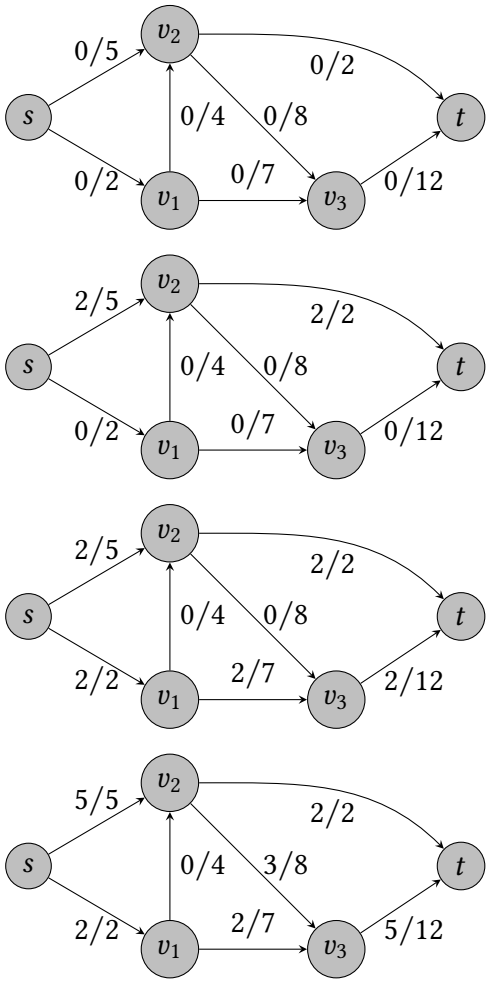
② Suche einen kürzesten augmentierten Weg π von s nach t. Falls keiner existiert, dann `stop`.

③ Berechne ε gemäss (5.1), (5.2). Augmentiere f um ε und gehe nach ②.

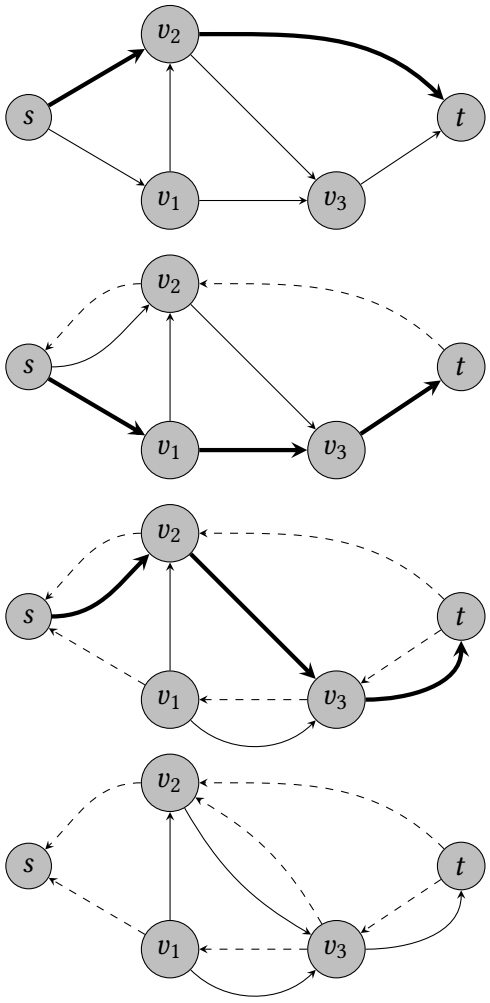
Bemerkung Schritt ② kann durch eine Breitensuche im Restdigraphen realisiert werden, vergleiche Satz 3.6. ♦

Beispiel 5.12 Wir illustrieren den Edmonds-Karp-Algorithmus anhand des folgenden konkreten Beispiels.

Netzwerk mit aktuellem Fluss:



Restdigraph und kürzester augmentierter Weg:



♣

Um die Korrektheit des Algorithmus von Edmonds und Karp zu zeigen, werden wir folgendes Lemma benötigen:

Lemma 5.13 *Sei $(f_0, \pi_0), (f_1, \pi_1), (f_2, \pi_2), \ldots$ die von Algorithmus 5.11 erzeugte Folge von Flussfunktionen f_i und zugehörigen kürzesten augmentierten Wegen π_i im Restdigraphen von f_i. Dann gelten die folgenden Aussagen:*

1. *Für alle i gilt $|\pi_i| \leq |\pi_{i+1}|$.*
2. *Kommt $e = (v, w)$ in π_i und $e^{-1} = (w, v)$ in π_j vor mit $i < j$, so gilt*

$$|\pi_i| + 2 \leq |\pi_j|.$$

Beweis. Es sei $\ell_i(x, y)$ die Länge eines kürzesten Wegs von x nach y im Restdigraphen von f_i. Insbesondere gilt also $|\pi_i| = \ell_i(s, t)$. Wir zeigen zuerst, dass für alle $v \in V$ gilt

$$\ell_{i+1}(s, v) \geq \ell_i(s, v). \tag{5.3}$$

Falls $\ell_{i+1}(s, v) = \infty$, dann ist die Ungleichung trivialerweise erfüllt. Wir können also annehmen, dass v von s im Restdigraphen von f_{i+1} erreichbar ist, das heisst $\ell_{i+1}(s, v) = r < \infty$. Sei $\pi = s, v_1, v_2, \ldots, v_r$ ein kürzester Weg von s nach $v = v_r$ im Restdigraphen von f_{i+1}. Wir zeigen, dass dann gilt

$$\ell_i(s, v_{j+1}) \leq \ell_i(s, v_j) + 1, \quad 1 \leq j < r. \tag{5.4}$$

Falls (v_j, v_{j+1}) eine Kante im Restdigraphen von f_i ist, gilt (5.4) offensichtlich. Ist (v_j, v_{j+1}) keine Kante im Restdigraphen von f_i, dann muss sich der Flusswert der inversen Kante (v_{j+1}, v_j) im Augmentierungsschritt $f_i \mapsto f_{i+1}$ verändert haben. Andernfalls könnte (v_j, v_{j+1}) im Restdigraphen von f_{i+1} nicht vertreten sein. Folglich liegt die Kante (v_{j+1}, v_j) auf dem Weg π_i. Da π_i ein kürzester Weg von s nach t ist und v_{j+1} unmittelbar vor v_j in π_i vorkommt, ergibt sich

$$\ell_i(s, v_{j+1}) = \ell_i(s, v_j) - 1,$$

das heisst, es gilt ebenfalls (5.4).

Aus (5.4) folgt dann (5.3), denn

$$\begin{aligned}
\ell_i(s, v) &= \ell_i(s, v_r) \\
&\overset{(5.4)}{\leq} \ell_i(s, v_{r-1}) + 1 \\
&\overset{(5.4)}{\leq} \ell_i(s, v_{r-2}) + 2 \\
&\;\;\vdots \\
&\overset{(5.4)}{\leq} \ell_i(s, v_1) + r - 1 \\
&\overset{(5.4)}{\leq} \ell_i(s, s) + r \\
&= \ell_{i+1}(s, v).
\end{aligned}$$

Insbesondere liefert die Wahl $v = t$ Aussage 1.

Analog zu (5.3) zeigt man, dass auch

$$\ell_{i+1}(v,t) \geq \ell_i(v,t) \tag{5.5}$$

gilt für alle $v \in V$.

Sei nun $e = (v,w)$ bzw. $e^{-1} = (w,v)$ eine Kante im Weg π_i bzw. π_j mit $i < j$, das heisst

$$\pi_i = s, \dots, v, w, \dots, t, \qquad \pi_j = s, \dots, w, v, \dots, t.$$

Da beide jeweils kürzeste Wege im entsprechenden Restdigraphen sind, gilt

(i) $|\pi_i| = \ell_i(s,v) + \ell_i(v,t)$,

(ii) $|\pi_j| = \ell_j(s,w) + 1 + \ell_j(v,t)$,

(iii) $\ell_i(s,w) = \ell_i(s,v) + 1$,

während (5.3) und (5.5) wegen $i < j$ implizieren

(iv) $\ell_j(s,w) \geq \ell_i(s,w), \quad \ell_j(v,t) \geq \ell_i(v,t)$.

Kombination der Beziehungen (i)–(iv) liefert Aussage 2:

$$\begin{aligned} |\pi_j| &\overset{\text{(ii)}}{=} \ell_j(s,w) + 1 + \ell_j(v,t) \\ &\overset{\text{(iv)}}{\geq} \ell_i(s,w) + 1 + \ell_i(v,t) \\ &\overset{\text{(iii)}}{=} \ell_i(s,v) + 2 + \ell_i(v,t) \\ &\overset{\text{(i)}}{=} |\pi_i| + 2. \end{aligned}$$

♠

Satz 5.14 (Edmonds-Karp) *Unabhängig von den Kapazitäten terminiert Algorithmus 5.11 nach höchstens $(n \cdot m)/2$ Augmentierungsschritten, wobei $n = |V|$ und $m = |E|$ ist.*

Beweis. Sei $(f_0, \pi_0), (f_1, \pi_1), (f_2, \pi_2), \dots$ die von Algorithmus 5.11 erzeugte Folge von Flussfunktionen f_i und zugehörigen kürzesten Wegen π_i im Restdigraphen von f_i. In jedem Augmentierungsschritt wird mindestens eine Kante $e = (v,w)$ des Wegs π_i voll ausgeschöpft, das heisst, dass eine Flussveränderung um die Restkapazität stattfindet:

- Ist e eine Vorwärtskante im Restdigraphen von f_i, so ist $f_{i+1}(e) = c(e)$.
- Ist e eine Rückwärtskante im Restdigraphen von f_i, so ist $f_{i+1}(e^{-1}) = 0$.

In keinem der beiden Fälle ist e eine Kante im Restdigraphen von f_{i+1}. Bevor dieselbe Kante in einem späteren Augmentierungsschritt $f_k \mapsto f_{k+1}$ in π_k vorkommt und wieder voll ausgeschöpft wird, muss die inverse Kante $e^{-1} = (w, v)$ im Weg π_j mit $i < j < k$ vorgekommen sein. Aus Lemma 5.13 folgt

$$|\pi_i| \leq |\pi_j| - 2 \leq |\pi_k| - 4.$$

Wird also e in den Wegen $\pi_{i_0}, \pi_{i_1}, \pi_{i_2}, \dots, \pi_{i_\ell}$ voll ausgeschöpft, dann existiert eine Indexfolge $j_0, j_1, \dots, j_{\ell-1}$ derart, dass

- $i_0 < j_0 < i_1 < j_1 < \cdots < i_{\ell-1} < j_{\ell-1} < i_\ell$,
- $e^{-1} = (w, v)$ kommt in $\pi_{j_0}, \pi_{j_1}, \dots, \pi_{j_\ell - 1}$ vor,
- $1 \leq |\pi_{i_0}| \leq |\pi_{j_0}| - 2 \leq |\pi_{i_1}| - 4 \leq |\pi_{j_1}| - 6 \leq \cdots \leq |\pi_{i_\ell}| - 4\ell$.

Da kürzeste Wege stets einfach sind, ist π_{i_k} stets ein einfacher Weg im Restdigraphen von f_{i_k} und es folgt

$$|\pi_{i_\ell}| < n.$$

Hieraus folgt jedoch, dass jede Kante $e \in E \cup E^{-1}$ höchstens $n/4$-mal voll ausgeschöpft werden kann, das heisst, $\ell < n/4$. Da nur $|E \cup E^{-1}| \leq 2|E|$ Kanten vorhanden sind, werden maximal

$$2m \cdot \frac{n}{4} = \frac{m \cdot n}{2}$$

Augmentierungsschritte durchgeführt. ♠

Bemerkung Wir haben soeben die Existenz einer Lösung des Netzwerkflussproblems gezeigt! ♦

Korollar 5.15 *Der Aufwand des Edmonds-Karp-Algorithmus ist $\mathcal{O}(m^2 n)$, wobei $n = |V|$ und $m = |E|$.*

Beweis. Gemäss Satz 5.14 benötigen wir höchstens $(m \cdot n)/2$ Augmentierungsschritte. Da hierzu jeweils eine Breitensuche benötigt wird, die den Aufwand $\mathcal{O}(m)$ besitzt, ergibt sich das Behauptete. ♠

6 Bipartites Matching

Definition 6.1 Sei $G = (V, E)$ ein Graph. Ein *Matching* von G ist eine Kantenmenge $M \subseteq E$, so dass jeder Knoten von G höchstens auf einer Kante von M liegt, das heisst, wenn für alle Kanten $e = \{v, w\}, e' = \{x, y\} \in M$ gilt

$$e \neq e' \Rightarrow \{v, w\} \cap \{x, y\} = \emptyset.$$

Ein Matching M heisst *maximal*, wenn $|M| \geq |M'|$ für alle Matchings M' von G.

Wir wollen uns im folgenden darauf beschränken, ein maximales Matching in einem *bipartiten* Graphen zu suchen.

Definition 6.2 Ein Graph $G = (V, E)$ heisst *bipartit* oder *zweigeteilt*, falls nichtleere Knotenmengen V_L und V_R existieren, so dass

(i) $V = V_L \cup V_R, V_L \cap V_R = \emptyset$,

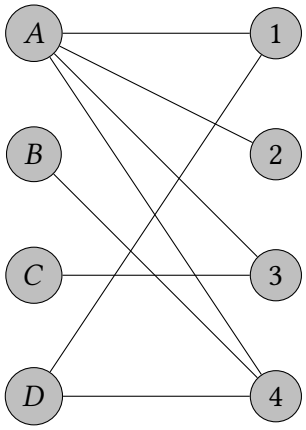
(ii) für jede Kante $\{v, w\} \in E$ ist

$$\{v, w\} \cap V_L \neq \emptyset, \ \{v, w\} \cap V_R \neq \emptyset.$$

Die Mengen V_L, V_R heissen *(Bi-) Partition*.

Viele Anwendungen führen auf die Bestimmung eines maximalen Matchings in einem bipartiten Graphen. Wir betrachten exemplarisch das sogenannte "Heiratsproblem".

Beispiel 6.3 Vier Frauen $V_L = \{A, B, C, D\}$ haben unter vier Männern $V_R = \{1, 2, 3, 4\}$ diejenigen ausgewählt, die sie sich als Ehepartner wünschen, und umgekehrt. Eine Heiratsagentur soll anhand dieser Information potentielle Paare bilden. Gesucht ist folglich eine Paarbildung, bei der nur Wunschpaare zulässig sind und die Zahl der Heiratsvermittlungen maximal ist. Wir erhalten beispielsweise den Graphen



wobei die Kanten für Wunschpaare stehen. Ein maximales Matching ist

$$M = \big\{\{A,2\},\{B,4\},\{C,3\},\{D,1\}\big\}.$$ ♣

Wir führen das Matchingprobem auf ein äquivalentes Flussproblem zurück.

Definition 6.4 Sei $G = (V, E)$ ein bipartiter Graph mit Partition $V = V_L \cup V_R$. Wir definieren das zugehörige Netzwerk $N_G = (V \cup \{s,t\}, E', c, s, t)$ gemäss:

- $s, t \notin V$ und $s \neq t$,
- $E' = \vec{E} \cup \{(s,v) : v \in V_L\} \cup \{(w,t) : w \in V_R\}$,
- $c(e) = 1$ für alle $e \in E'$.

Hierbei ist $\vec{E} := \big\{(v,w) \in V_L \times V_R : \{v,w\} \in E\big\}$ eine *Orientierung* für G, die nur die Kanten von V_L nach V_R enthält. Eine *0-1-Flussfunktion* für N_G ist eine Flussfunktion f für N_G mit $f(e) \in \{0,1\}$ für alle $e \in E'$.

Satz 6.5 *Sei $G = (V, E)$ ein bipartiter Graph mit Partition $V = V_L \cup V_R$, dann gilt:*

(i) Zu jedem Matching M gibt es eine 0-1-Flussfunktion f_M für N_G mit $\mathrm{flow}(f_M) = |M|$.

(ii) Zu jeder 0-1-Flussfunktion f für N_G gibt es ein Matching M_f für G mit $\mathrm{flow}(f) = |M_f|$.

Beweis. (i) Sei f_M definiert gemäss

$$f_M(v,w) := \begin{cases} f_M(s,v) = f_M(w,t) = 1, & \text{falls } v \in V_L, w \in V_R, \{v,w\} \in M, \\ 0, & \text{sonst.} \end{cases}$$

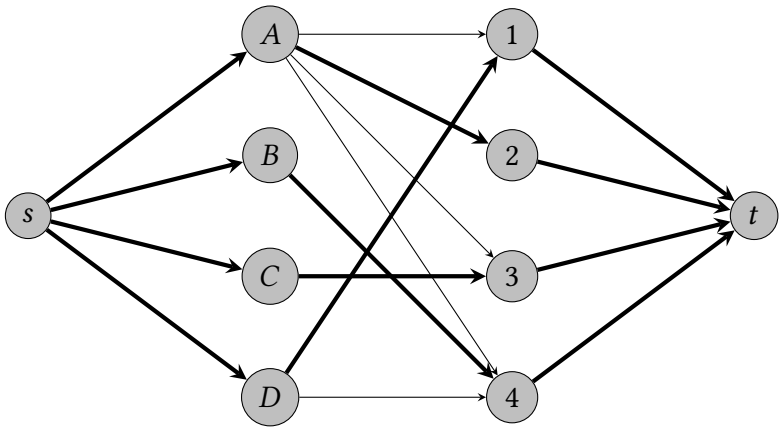
Da aufgrund der Matchingbedingungen jeder Knoten auf höchstens einer Kante von E liegt, erfüllt f_M das Kirchhoffsche Gesetz. Folglich ist f_M eine 0-1-Flussfunktion, insbesondere gilt $\mathrm{flow}(f_M) = |M|$.

(ii) Definiere

$$M_f := \big\{\{v,w\} \in E \,:\, f(v,w) = 1\big\}.$$

Das Kirchhoffsche Gesetz für f entspricht der Matchingeigenschaft von M_f und offensichtlich gilt $\text{flow}(f) = |M_f|$. ♠

Beispiel 6.6 Für das Heiratsproblem aus Beispiel 6.3 sind N_G und f_M gegeben durch:



Hierin entsprechen dicke Pfeile $f_M(v,w) = 1$, dünne Pfeile $f_M(v,w) = 0$. ♣

Korollar 6.7 *Für G wie zuvor gilt*

$$\text{MaxFlow}(N_G) = \max\{|M| \,:\, M \text{ ist Matching für } G\}.$$

Eine maximale 0-1-Flussfunktion für N_G kann mit dem Ford-Fulkerson-Algorithmus in Laufzeit $\mathcal{O}(n \cdot m)$ bestimmt werden, wobei $m = |E|$ und $n = |V|$.

Beweis. Die ersten Aussagen folgen sofort aus Satz 6.5. Die Aussage hinsichtlich der Laufzeit folgt aus der Tatsache, dass die Anzahl der Flusserhöhungsschritte beschränkt ist durch $\text{MaxFlow}(N_G)$ und

$$\text{MaxFlow}(N_G) \le \sum_{v \in \text{post}(s)} c(s,v) = \sum_{v \in V_L} \underbrace{c(s,v)}_{=1} = |V_L| \le n.$$ ♠

Bemerkung Die Suche nach augmentierten Wegen kann mit einer Tiefen- oder Breitensuche in Laufzeit $\mathcal{O}(m)$ durchgeführt werden. ♦

Definition 6.8 Sei $G = (V,E)$ ein bipartiter Graph mit Partition $V = V_L \cup V_R$ und $|V_L| \le |V_R|$. Ein Matching M heisst *perfekt*, falls $|M| = |V_L|$ gilt.

Satz 6.9 (Heiratssatz von Hall) *Sei $G = (V, E)$ ein bipartiter Graph mit Partition $V = V_L \cup V_R$ und $|V_L| \leq |V_R|$. Dann existiert ein perfektes Matching genau dann, wenn*

$$|\operatorname{post}(W)| \geq |W| \quad \text{für alle } W \subseteq V_L.$$

Hierbei gilt $\operatorname{post}(W) := \cup_{w \in W} \operatorname{post}(w)$.

Beweis. "⇒" Ist M perfekt und $W \subseteq V_L$, so enthält M für alle $w \in W$ genau eine Kante $\{w, w_M\} \in E$. Die Knoten w_M liegen also in $\operatorname{post}(W)$ und sind paarweise verschieden, dies bedeutet

$$|\operatorname{post}(W)| \geq |W|.$$

"⇐" Sei $|\operatorname{post}(W)| \geq |W|$ für alle $W \subseteq V_L$. Seien N_G das zum Matchingproblem gehörige Netzwerk und f eine maximale 0-1-Flussfunktion. Gilt

$$\operatorname{flow}(f) = |V_L|,$$

so ist das entsprechende Matching gemäss Korollar 6.7 perfekt.

Sei S_f die Menge aller von s erreichbaren Knoten im Restdigraphen von f. Wegen $t \notin S_f$ und

$$\begin{aligned}
\operatorname{cap}(S_f) &= \sum_{\substack{v \in S_f \\ w \in \operatorname{post}(v) \setminus S_f}} c(v, w) \\
&= \sum_{\substack{v \in S_f \\ w \in \operatorname{post}(v) \setminus S_f}} f(v, w) - \sum_{\substack{w \in S_f \\ v \in \operatorname{pre}(w) \setminus S_f}} \underbrace{f(v, w)}_{=0} \\
&= \operatorname{flow}(f)
\end{aligned}$$

ist S_f ein Schnitt mit minimaler Schnittkapazität, vergleiche den Beweis des Max-Flow-Min-Cut-Theorems. Wir zeigen zunächst, dass

$$\operatorname{post}(V_L \cap S_f) \subseteq S_f \tag{6.1}$$

gilt. Sei hierzu $v \in S_f \cap V_L$ beliebig und $w \in \operatorname{post}(v)$. Angenommen, (6.1) gilt nicht, dann ist $w \notin S_f$. Folglich ist (v, w) keine Kante im Restdigraphen von f. Die Kante (v, w) ist somit gesättigt, das heisst

$$f(v, w) = 1.$$

Da $\operatorname{pre}(v) = \{s\}$, folgt aus dem Kirchhoffschen Gesetz, dass $f(s, v) = 1$. Damit ist auch (s, v) keine Kante im Restdigraphen. Der Knoten v kann also im Restdigraphen nur über eine Rückwärtskante von der Quelle s erreicht werden. Daher gibt es ein $u \in \operatorname{post}(v) \cap S_f$ mit

$$f(v, u) = 1.$$

Es gibt folglich zwei direkte Nachfolger von v mit

$$f(v, w) = f(v, u) = 1.$$

Dies widerspricht aber dem Kirchhoffschen Gesetz, da (s, v) die einzige zu v führende Kante ist.

Damit gilt (6.1) und es folgt für alle Kanten (u, v) in N_G mit $u \in S_f$ und $v \in V \setminus S_f$, dass $u = s$ oder $v = t$ ist. Dies impliziert

$$\begin{aligned}
\operatorname{cap}(S_f) &= \sum_{\substack{u \in S_f \\ v \in \operatorname{post}(u) \setminus S_f}} \underbrace{c(u,v)}_{=1} \\
&= |\{(s,v) : (s,v) \in E' \text{ und } v \notin S_f\}| \\
&\qquad + |\{(v,t) : (v,t) \in E' \text{ und } v \in S_f\}| \\
&= |V_L \setminus S_f| + |\underbrace{S_f \cap V_R}_{\overset{(6.1)}{\supseteq} \operatorname{post}(V_L \cap S_f)}| \\
&\geq |V_L \setminus S_f| + |\operatorname{post}(V_L \cap S_f)|.
\end{aligned}$$

Nach Voraussetzung ist $|\operatorname{post}(W)| \geq |W|$, insbesondere

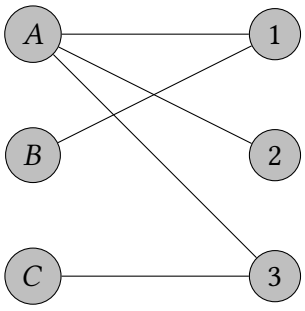
$$|\operatorname{post}(V_L \cap S_f)| \geq |V_L \cap S_f|,$$

und wir erhalten

$$\begin{aligned}
\operatorname{flow}(f) &= \operatorname{cap}(S_f) \\
&\geq |V_L \setminus S_f| + |\operatorname{post}(V_L \cap S_f)| \\
&\geq |V_L \setminus S_f| + |V_L \cap S_f| \\
&= |V_L|.
\end{aligned}$$

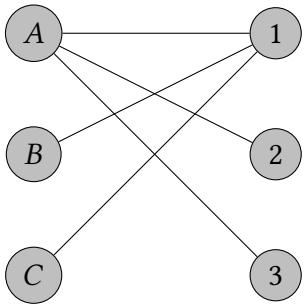
Andererseits gilt offensichtlich auch $\operatorname{flow}(f) \leq |V_L|$ und somit $\operatorname{flow}(f) = |V_L|$. ♠

Beispiel 6.10 Beim bipartiten Graphen



ist das Hallsche Kriterium erfüllt, denn A, B, C haben jeweils mindestens einen Nachfolger, $\{A, B\}, \{B, C\}, \{A, C\}$ mindestens zwei Nachfolger und $\{A, B, C\}$ drei Nachfolger.

Zum bipartiten Graphen



existiert kein perfektes Matching, denn $\{B, C\}$ hat nur einen Nachfolger. ♣

Der Heiratssatz von Hall liefert kein zufriedenstellendes algorithmisches Kriterium für die Existenz eines perfekten Matchings, da alle Teilmengen $W \subseteq V_L$ betrachtet werden müssen. Allerdings ermöglicht er diesen Nachweis in Graphen, in denen alle Knoten denselben Grad haben. Solche Graphen heissen auch *regulär*.

Satz 6.11 *Sei $G = (V, E)$ ein bipartiter Graph mit Partition $V = V_L \cup V_R$ und $|V_L| \leq |V_R|$. Gilt $|\operatorname{post}(v)| = k > 0$ für alle $v \in V$, so existiert ein perfektes Matching.*

Beweis. Seien $W \subseteq V_L$ und

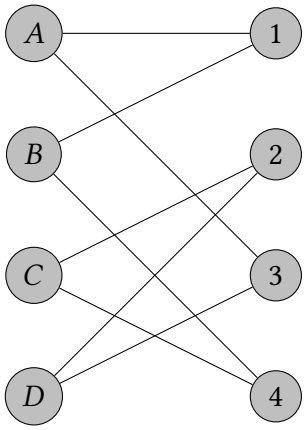
$$E_1 := \big\{\{v, w\} \in E : v \in W\big\},$$
$$E_2 := \big\{\{v, w\} \in E : w \in \operatorname{post}(W)\big\}.$$

Aus $E_1 \subseteq E_2$ folgt

$$k|W| = |E_1| \leq |E_2| = k|\operatorname{post}(W)|,$$

das heisst $|W| \leq |\operatorname{post}(W)|$. Satz 6.9 liefert dann die Behauptung. ♠

Beispiel 6.12 Der bipartite Graph



erfüllt $|\operatorname{post}(v)| = 2$ für alle Knoten $v \in V$. Ein perfektes Matching ist zum Beipiel

$$M = \big\{\{A, 3\}, \{B, 1\}, \{C, 4\}, \{D, 2\}\big\}.$$ ♣

Index