# Algorithmen

Michael Kaufmann

24/11/2020 – 7. Vorlesung
Graphenalgorithmen -Grundbegriffe

# Gliederung

**I. Einführung**
**II. Grundlegende Datenstrukturen**
- Arrays und Listen
- Bäume
- Keller und Warteschlangen
- Heaps und Prioritätswarteschlangen
- Hashing

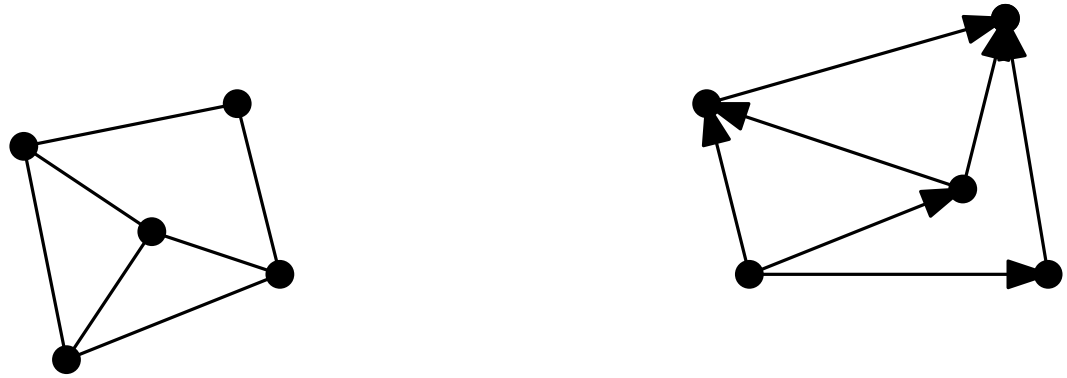
**III. Graphenalgorithmen**
- Grundlegendes
- Kürzeste Wege
- Graphdurchmusterung

**IV. Sortieren**
**V. Suchen**
**VI. Generische algorithmische Methoden**
**VII. Algorithmen auf Zeichenketten**

# III. Graphenalgorithmen

# Grundbegriffe

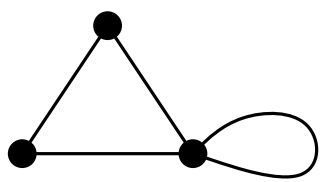
Ein Graph $G = (V, E)$ besteht auf einer Menge von Knoten $V$ und einer Menge von Kanten $E \subset V \times V$.



- Kanten können ungerichtet oder gerichtet sein. Zugehörige Notation $\{u, v\}$ oder $(u, v)$
- $u$ heißt adjazent (benachbart) zu $v$, falls es eine Kante zwischen $u$ und $v$ gibt. Die Kante $\{u, v\}$ ist inzident zu $u$ und $v$.

# Grundbegriffe

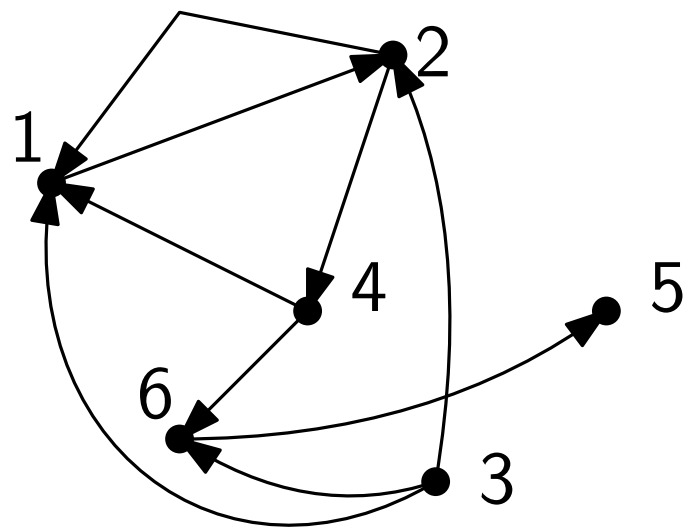
- In einem ungerichteten Graph schreiben wir $u \sim v$, falls $u$ adjazent zu $v$ ist
- In einem gerichteten Graph schreiben wir $u \to v$, falls es Kante $(u, v) \in E$ gibt
- Sind Gewichte oder Kosten auf den Kanten, so hat jede Kante $(u, v)$ ein Gewicht $w(u, v)$ oder Kosten $c(u, v)$.
- Manchmal sehen wir ungewichtete Graphen als speziell gewichtete Graphen an, wo alle Kantengewichte entweder $= 0$ (keine Kante) oder $= 1$ (Kante existiert) sind.
- Eine Kante $e = (u, v)$ ist eine Selbstschleife, wenn $u = v$.



# Grundbegriffe

Ausgangsgrad eines Knoten $v$ sind die Zahl seiner ausgehenden Kanten,
Formal: $outdeg(v) = |\{w \in V \mid (v, w) \in E\}|$

Analog $indeg(v) = |\{w \mid (w, v) \in E\}|$ Eingangsgrad
sowie $deg(v) = indeg(v) + outdeg(v)$ bzw.
$= |\{w \mid \{v, w\} \in E\}|$ Grad von $v$ (ungerichtet)

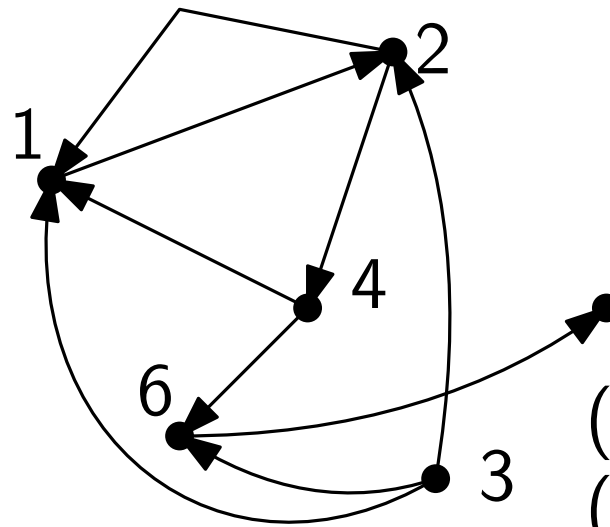



$outdeg(2) = 2$
$indeg(1) = 3$
$deg(3) = 3$

# Grundbegriffe

Sei $G = (V, E)$ ein Graph.
Eine Folge $(v_0, v_1, \ldots, v_k)$ heißt **Pfad**, falls für alle $0 \leq i \leq k-1$ Kanten $(v_i, v_{i+1})$ existieren.
Der Pfad startet in $v_0$, endet in $v_k$.

Ein Pfad $(v_0, v_1, \ldots, v_k)$ heißt **Zykel**, falls $v_k = v_0$.




$(1, 2, 4, 6, 5)$ ist Pfad.
$(1, 2, 4, 1)$ ist ein Zykel
Kein Pfad von 1 nach 3

# Grundlegendes

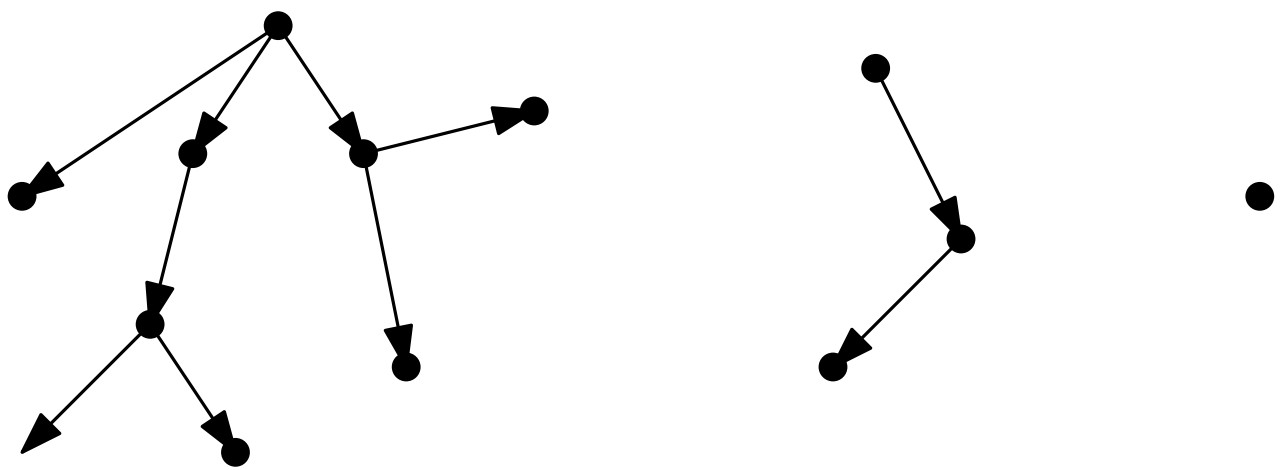
Graph $G = (V, E)$ heißt Baum, falls

a) $V$ enthält genau ein $v_0$ mit $indeg(v_0) = 0$

b) Für alle $v \in V \backslash \{v_0\}$ : $indeg(v) = 1$

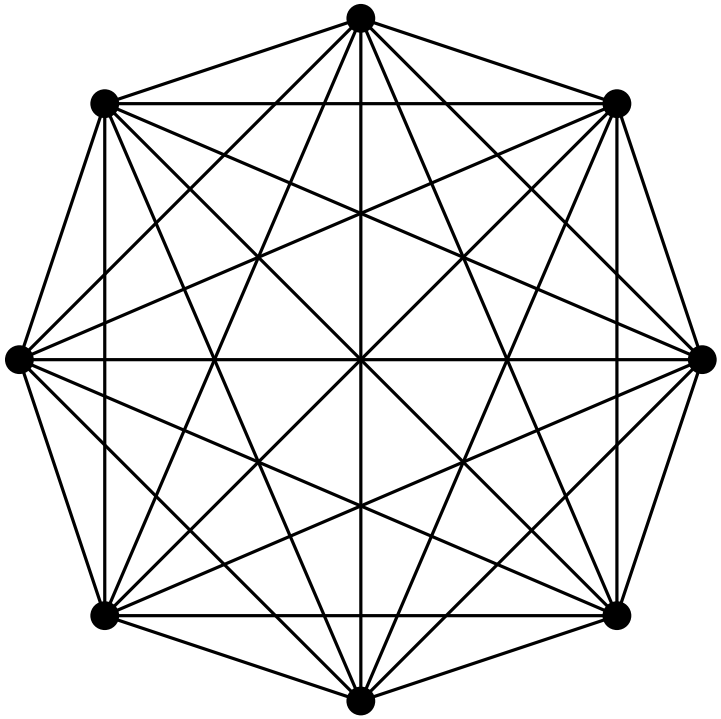
c) $G$ ist azyklisch (ohne Zykel)

$G = (V, E)$ heißt Wald, falls $G$ aus mehreren disjunkten Bäumen besteht. ($G = G_1 \cup ... \cup G_k$, jedes $G_i$ ist Baum)

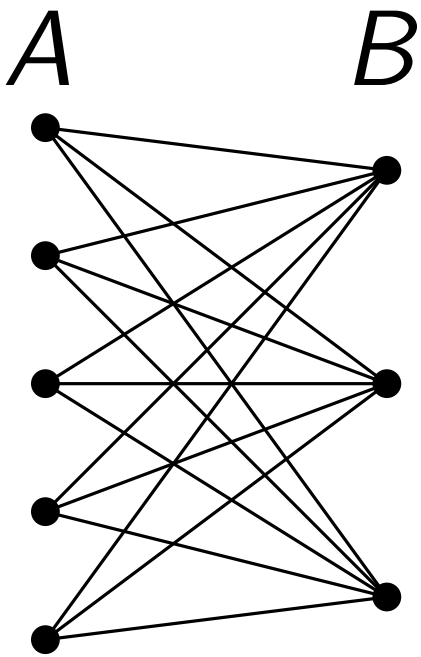


# Grundlegendes

**Vollständige Graphen**
sind Graphen, für die für beliebige Knoten $u, v \in V$ gilt:
Kante $\{u, v\} \in E$



$K_8$: der vollst. Graph
mit 8 Knoten

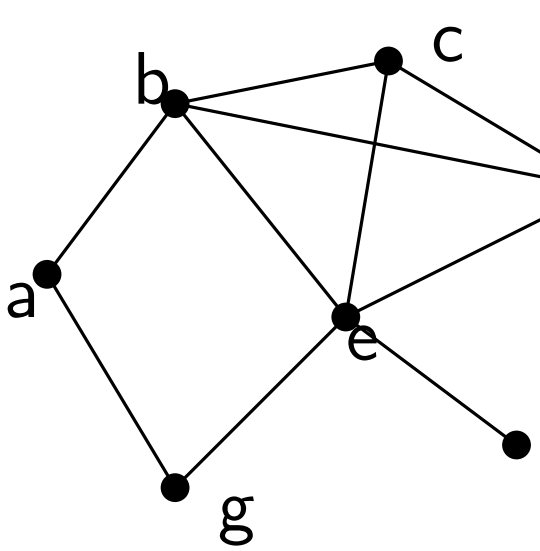



$K_{5,3}$: der vollst. bipartite Graph
mit 5 und 3 Knoten

# Noch mehr Begriffe

Ein Graph $G = (V, E)$ heißt **bipartit**, wenn $V = A \cup B$ und $\emptyset \neq A \neq V$ und für alle Kanten $u, v$ gilt: Falls $u \in A$, dann $v \in B$, sowie falls $u \in B$, dann $v \in A$.

Für einen Graph $G = (V, E)$ heißt ein Teilgraph $G' = (V', E')$ **induziert**, falls $V' \subseteq V$ und für alle Kanten $\{u, v\} \in E$ mit $u, v \in V'$ gilt: $\{u, v\} \in E'$



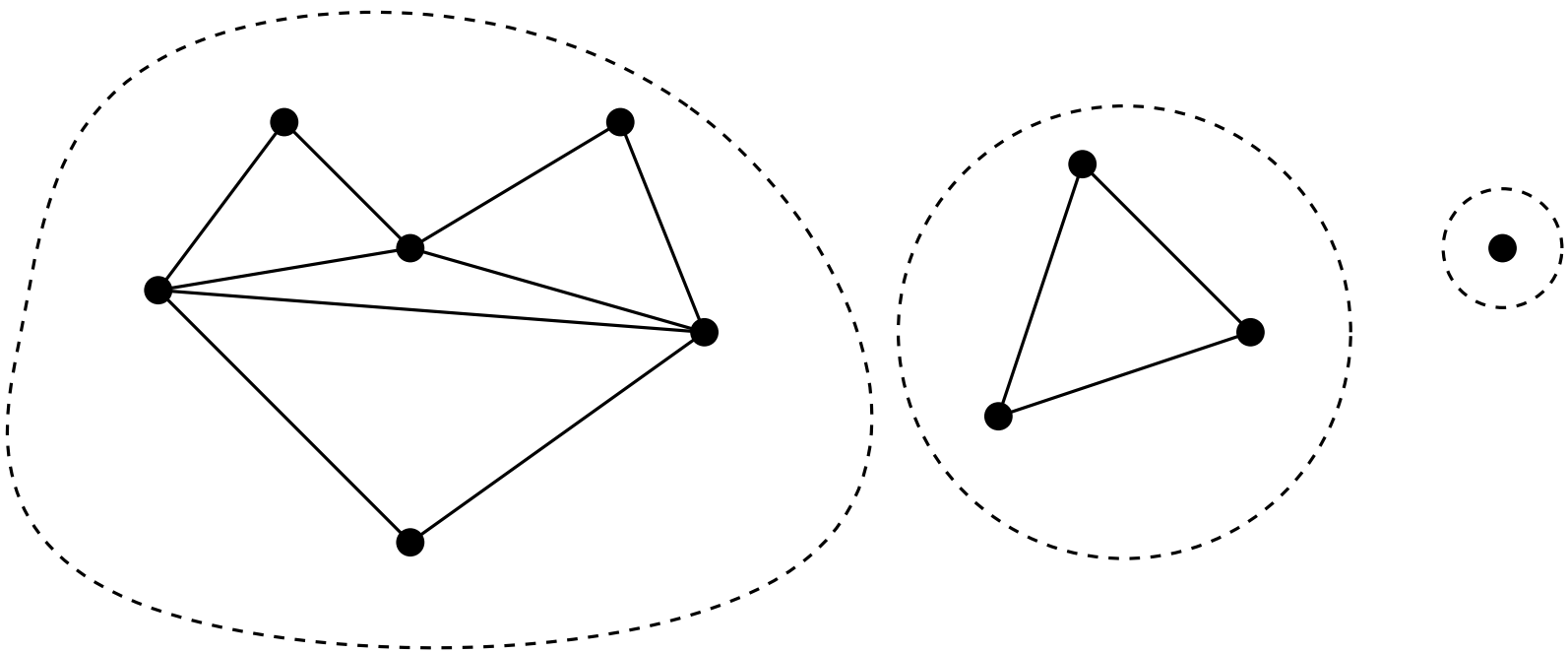

$G$

$G' =$
$(\{b, c, d, e\}, \{\{b, c\}, \{c, d\}, \{d, e\}, \{e, b\}\})$
ist NICHT induziert für $G$.

# Noch mehr Begriffe (2)

Ein ungerichteter Graph $(V, E)$ heißt **zusammenhängend**, falls es zwischen zwei beliebigen Knoten $u, v \in V$ einen Pfad gibt mit den Endpunkten $u$ und $v$.

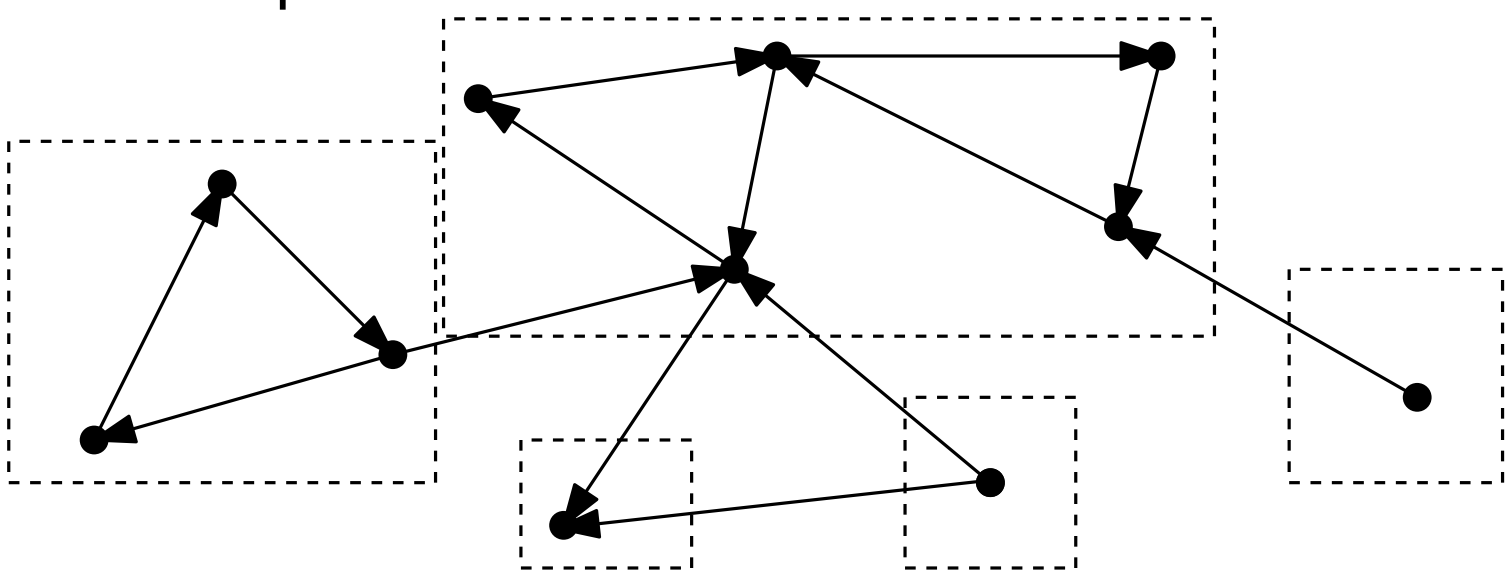
Eine **Zusammenhangskomponente** (ZK) eines ungerichteten Graphen $G$ ist ein maximaler zusammenhängender Teilgraph von $G$.



# Noch mehr Begriffe (3)

Ein **gerichteter** Graph $G = (V, E)$ heißt **stark zusammenhängend**, falls für alle Knoten $u \neq v \in V$ gilt: Es gibt einen Pfad von $u$ nach $v$ UND es gibt einen Pfad von $v$ nach $u$.

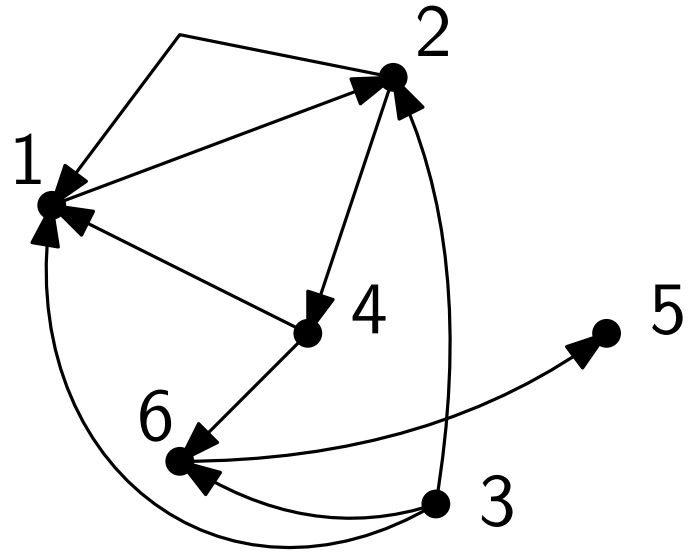
Eine **starke Zusammenhangskomponente** (SZK) ist ein maximaler stark zusammenhängender Teilgraph eines gerichteten Graphen $G$



# Darstellung

## 1. Adjazenzmatrix $A = (a_{i,j})$

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




$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Bei ungerichteten Graphen existieren beide Richtungen jeder Kante.
$\rightarrow$ Matrix ist symmetrisch.

# Notationen und Adjazenzmatrix

Sei $|V| = n$ und $|E| = m$.

**Platzbedarf**: $O(n^2)$.
**Zugriffszeit**: $O(1)$.

Platz eventuell nicht effizient. Insbesondere bei 'dünnen' Graphen, wo $m = O(n)$.

**Typische Fragen**:

- Welcher Knoten hat die meisten eingehenden Kanten? (beliebteste Person)
- Welches Knotenpaar ist am weitesten auseinander? ('Durchmesser')
- Welcher Knoten ist 'zentral'?

# 2. Adjazenzlisten

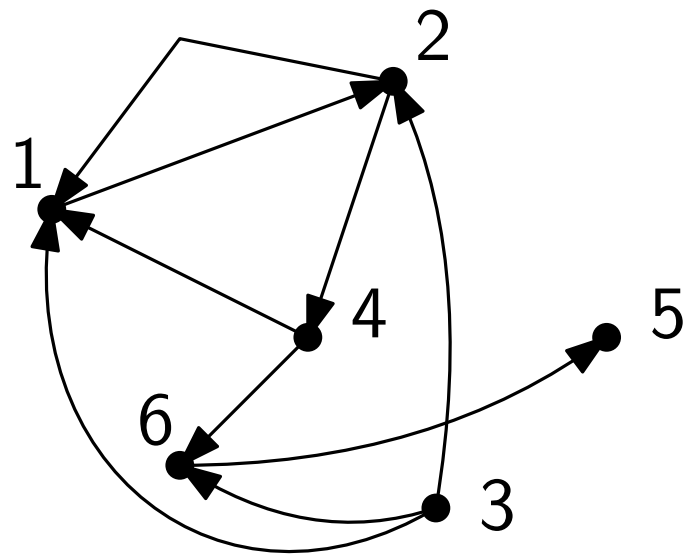
Speichern für jeden Knoten $v$ die Nachbarknoten.

**Falls $G$ gerichtet**:

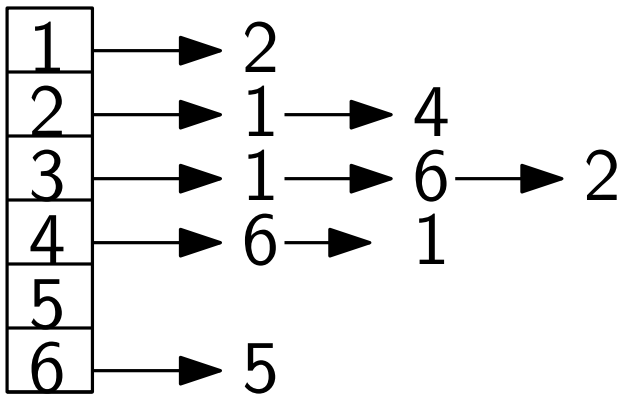
$$InAdj(v) = \{w \in V \mid (w, v) \in E\}$$
$$OutAdj(v) = \{w \in V \mid (v, w) \in E\}$$

**Falls $G$ ungerichtet**:

$$Adj(v) = \{w \in V \mid \{v, w\} \in E\}$$




**Bsp**: $OutAdj$ als Adjazenzliste




**Platz**: $O(n + m)$

# Adjazenzlisten

Zugriff auf Kante $(v, w)$ in $O(outdeg(v))$,
wobei $outdeg(v) = |\{w \in V \mid (v, w) \in E\}|$

**Platzbedarf**: $O(n + m)$.
**Zugriffszeit**: $O(outdeg(v))$.

Platz sehr effizient. Insbesondere bei 'dünnen' Graphen, wo $m = O(n)$ viel besser als Matrixdarstellung. Zugriff evtl. schlechter...

**Viele Algorithmen auf Adjazenzlistendarstellung ausgelegt.**

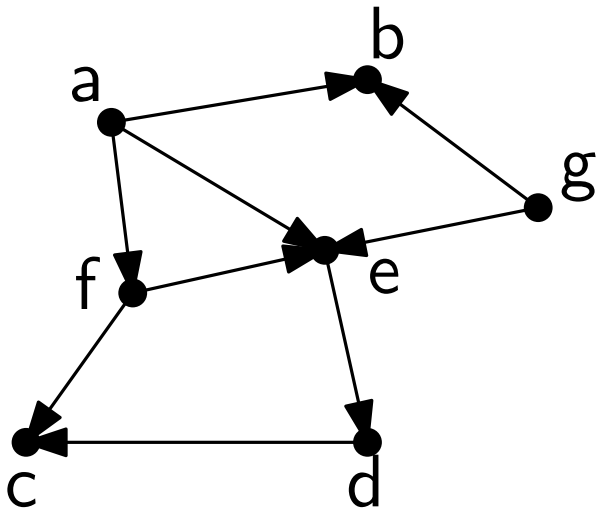
# Algorithmus: Topologisches Sortieren

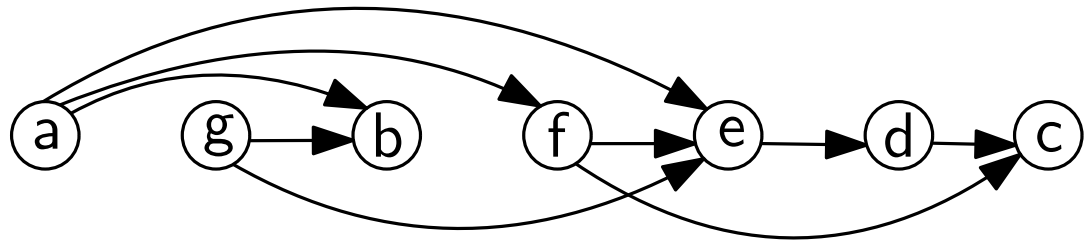
Sei $G = (V, E)$ ein gerichteter Graph (DAG).
Eine Abbildung

$$num : V \to \{1, 2, ..., n\}$$

mit $n = |V|$ heißt topologische Sortierung, falls für alle $(v, w) \in E$ gilt:

$$num(v) < num(w).$$




Pfeile alle nach rechts gerichtet.

# Topologisches Sortieren

**Lemma**:
Graph $G$ besitzt genau dann eine topologische Sortierung, wenn $G$ azyklisch ist.

**Beweis**:
'$\Rightarrow$' Annahme: $G$ zyklisch. Dann sei $(v_0, ..., v_k)$ ein Zykel mit $v_0 = v_k$. Es muss gelten:
$num(v_0) < num(v_1) < ... < num(v_k) = num(v_0)$ Wid.!

'$\Leftarrow$': Sei $G$ azyklisch.
**Beh.**: $G$ enthält Knoten $v$ mit $indeg(v) = 0$.

$\rightarrow$ $v$ kriegt die $num(v) = 1$. Lösche $v$ und dann induktiv

**Bew.**: Starte bei belieb. $w$. Laufe eingeh. Kanten rückwärts. Nach spätestens $n - 1$ Schritten ist $v$ gefunden (oder Zykel)

# Algorithmus *TopSort*

**TopSort**$((V, E), i)$

**If** $(|V| = 1)$

$num(v) \leftarrow i$ für $v \in V$

**If** $(|V| > 1)$ {

$v \leftarrow$ Knoten aus $V$ mit indegree 0

$num(v) \leftarrow i$

**TopSort**$((V \setminus \{v\}, E \setminus \{(v, w) \mid (v, w) \in E\}), i + 1)$

}

# Algorithmus *TopSort*

$count \leftarrow 0$
**while** $(\exists v \in V$ mit $indeg(v) = 0)$ {
    count++
    $num(v) \leftarrow$ count
    streiche $v$ mit ausgehenden Kanten
}
**if** $(count < |V|)$
    **return** '$G$ zyklisch'

**Ziel**: Laufzeit $O(n + m)$

# Implementierungsfragen

1. **benutzen Adjazenzlisten**: Platz $O(n+m)$
   - Lösche $v$ und Kanten mit Durchlaufen von $OutAdj(v)$ in $O(outdeg(v))$
   - insgesamt Laufzeit $\sum_v O(outdeg(v)) = O(n+m)$
2. **Finden geeignetes $v$**:
   - Benutze `inZaehler[]` für InDegrees und Menge `ZERO`
   - Wird $(v,w)$ gelöscht, dekrementiere `inZaehler[`$w$`]`. Nimm eventuell $w$ in `ZERO` auf
   - Geeignetes $v$ wird in `ZERO` gezogen und gelöscht $\rightarrow$ `ZERO` als Stack $\rightarrow$ Laufzeit $O(1)$
3. **Initialisierung von `inZaehler[]` und `ZERO`**:
   - Durchlaufe Adj.listen. Erhöh `inZaehler[`$w$`]` für Kante $(v,w)$
   - Initialisierung ZERO ?
   - Insgesamt Laufzeit $O(n+m)$

# TopSort

**Satz**:
Gegeben ein gerichteter Graph $G = (V, E)$.
In Zeit $O(n + m)$ kann festgestellt werden, ob $G$ einen Zykel hat, und wenn nicht, kann $G$ topologisch sortiert werden,