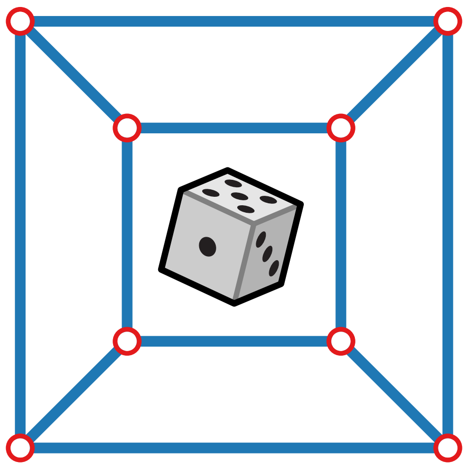
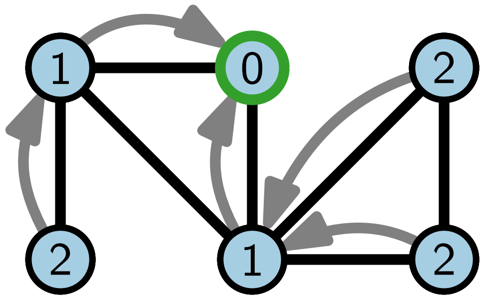
# Algorithmen und Datenstrukturen

## Vorlesung 17:
## Graphen und Breitensuche

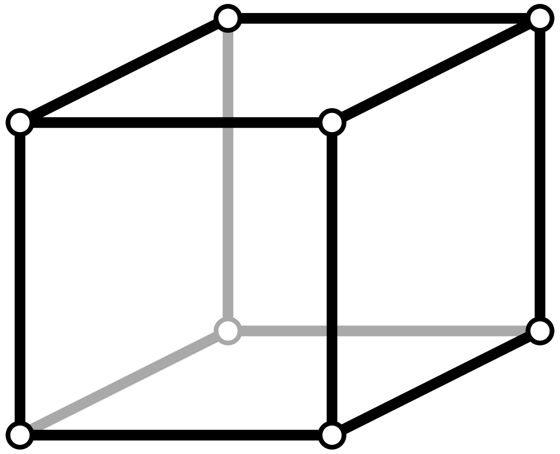


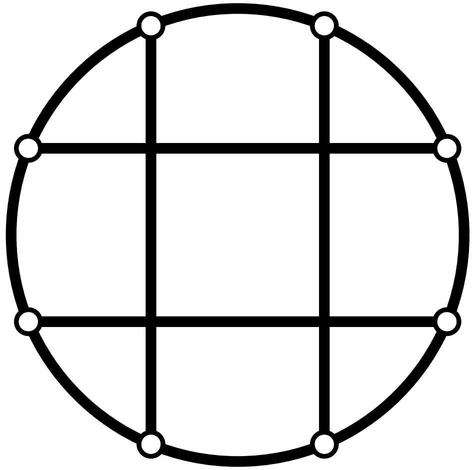


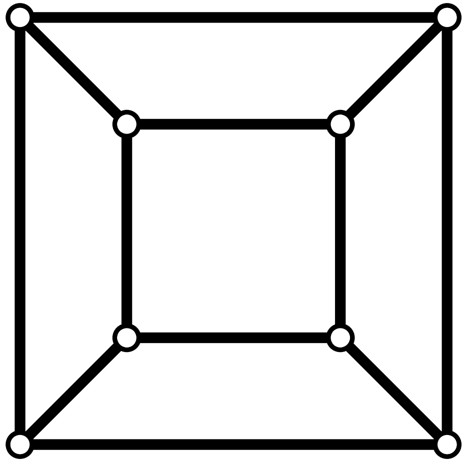

*Alexander Wolff*

*Wintersemester 2025*

# Was ist das?







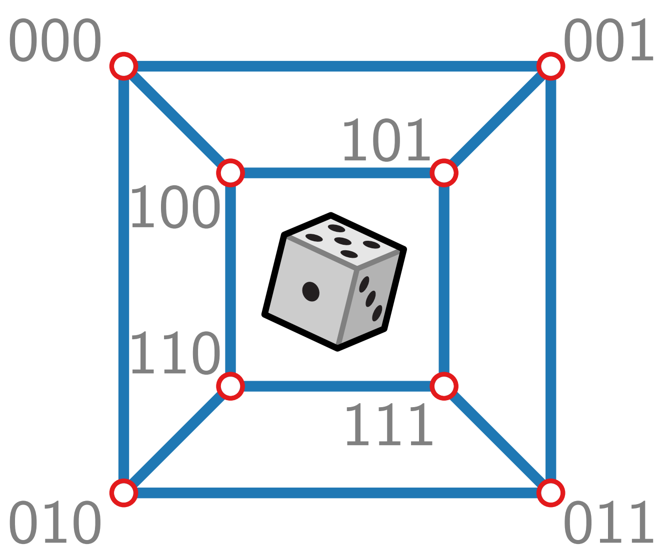
Ein (und derselbe) **Graph**; der dreidimensionale Hyperwürfel.

# Was ist ein Graph?

Ein (ungerichteter) Graph ist ein Paar $(V, E)$, wobei

- $V$ **Knotenmenge** und
- $E \subseteq \binom{V}{2} = \{\{u, v\} \subseteq V \mid u \neq v\}$ **Kantenmenge**.

$V = \{000, 001, \ldots, 111\}$
$\{u, v\} \in E \quad \Leftrightarrow \quad ?$

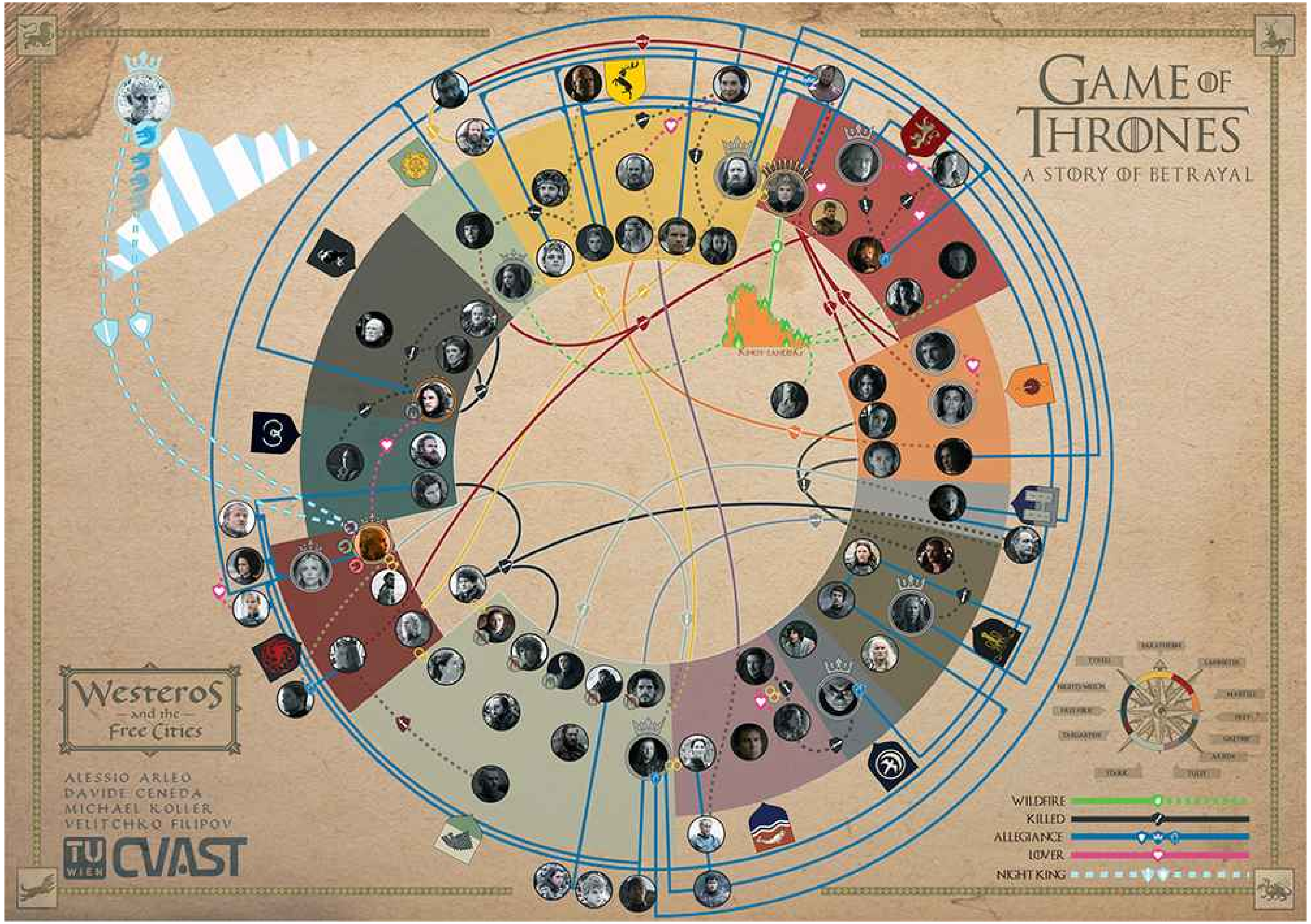




Ein **gerichteter** Graph ist ein Paar $(V, E)$ , wobei

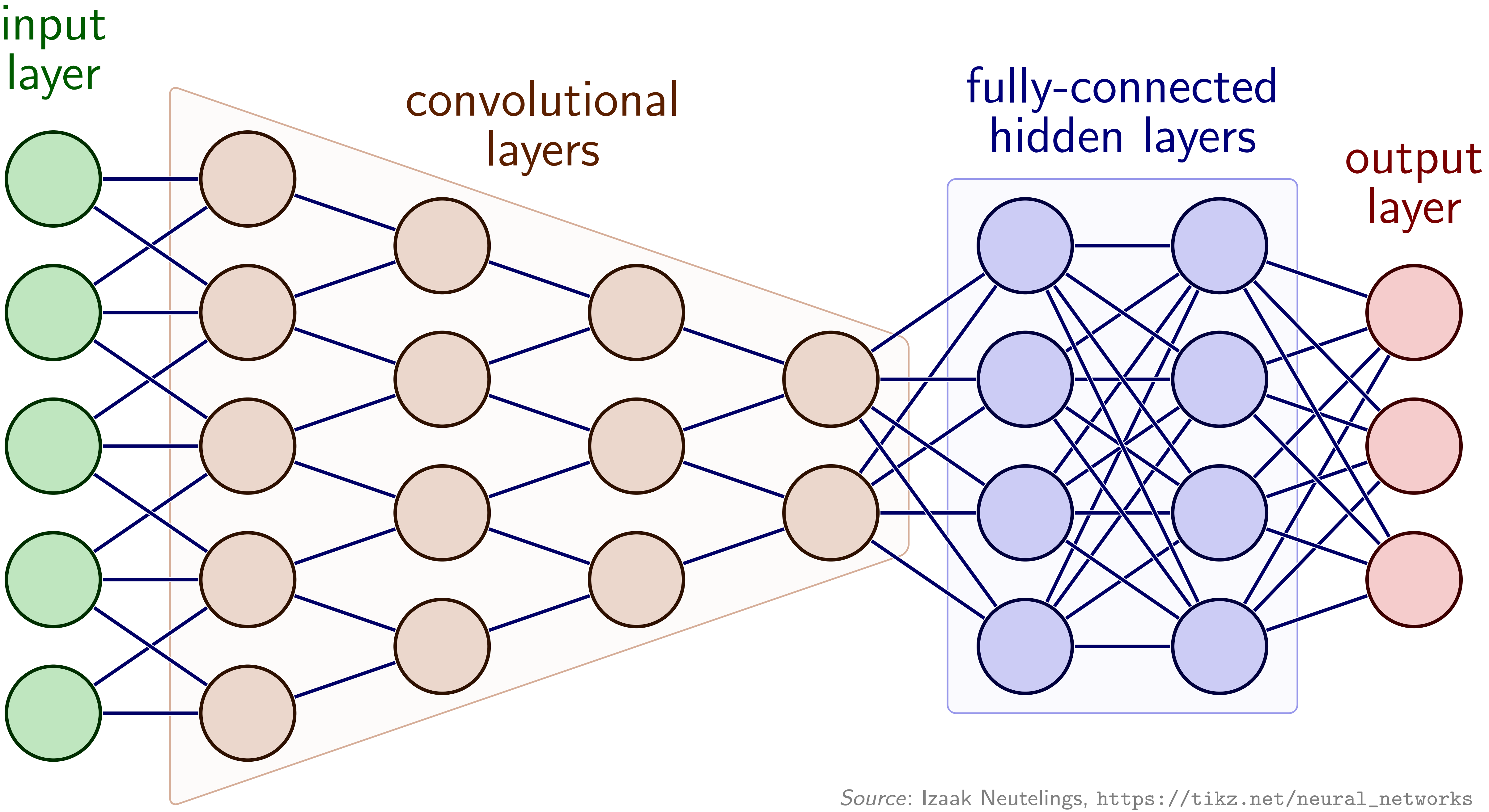
- $V$ **Knotenmenge** und
- $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ **Kantenmenge**.

# Soziale Netzwerke – Verhältnisse



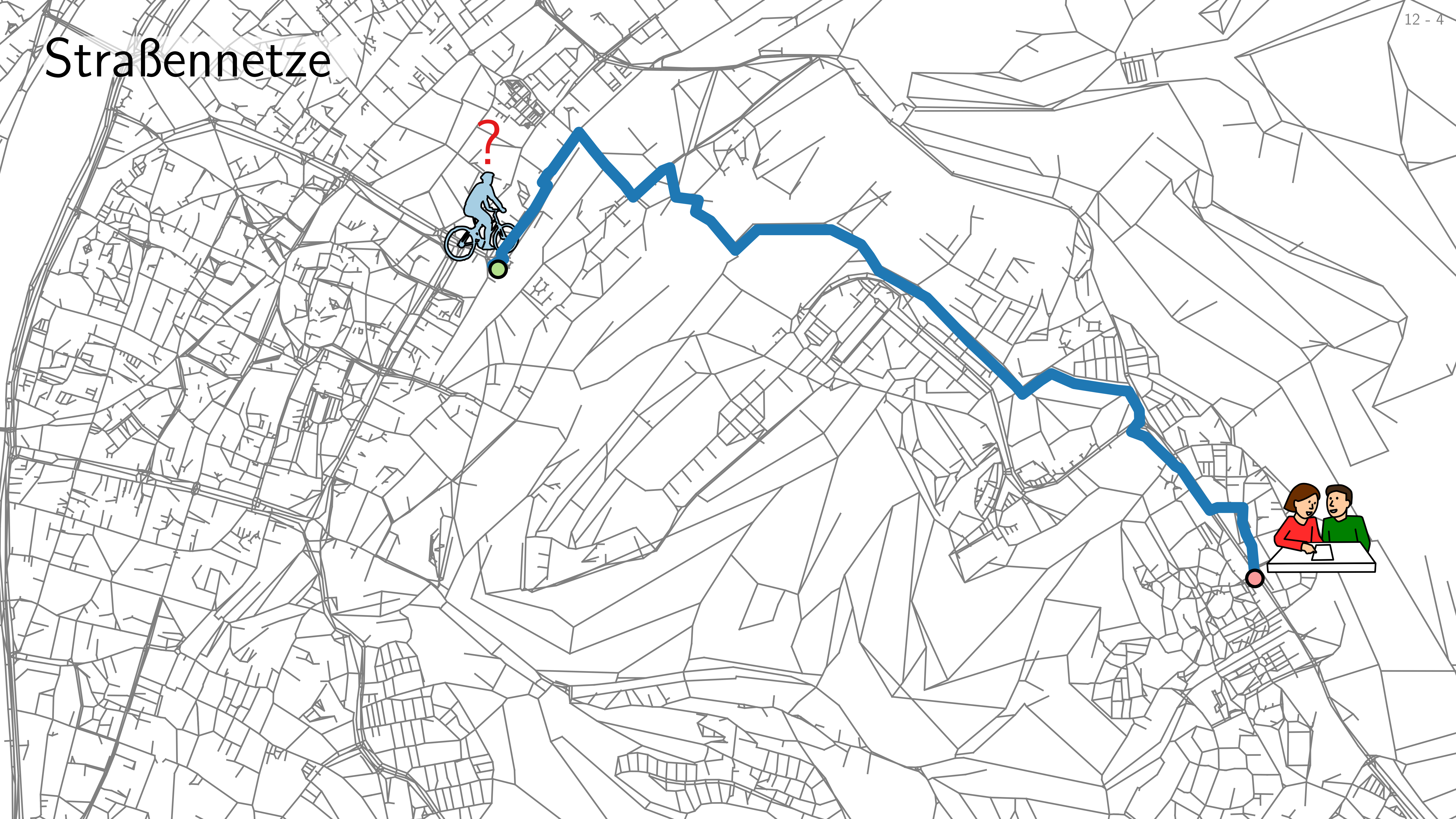

Velitchko Filipov, Davide Ceneda, Michael Koller, Alessio Arleo, and Silvia Miksch, GD Contest 2018

# Neuronale Netzwerke



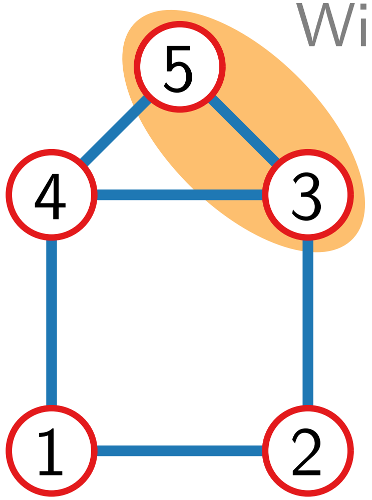

*Source*: Izaak Neutelings, `https://tikz.net/neural_networks`

# Straßennetze

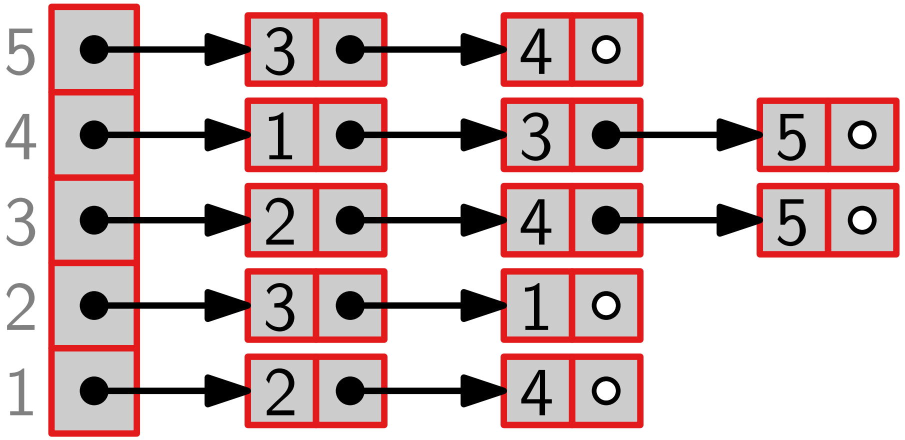


# Wie repräsentiere ich einen Graphen?

Wir sagen: Knoten 3 und 5 sind **adjazent**.



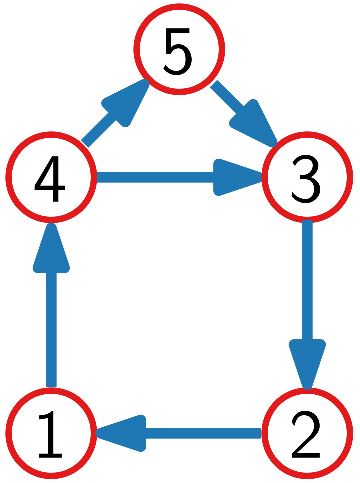

ungerichteter Graph



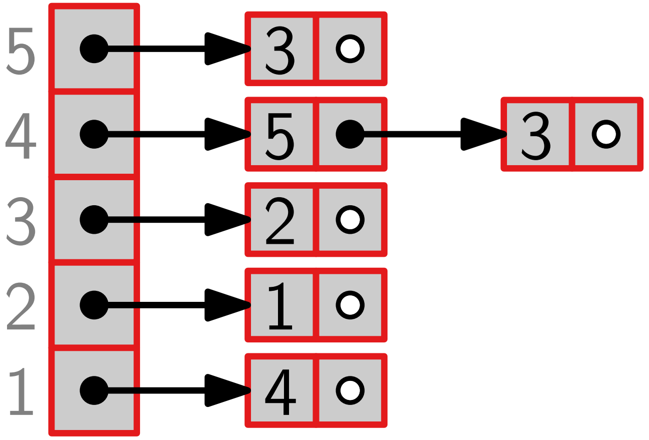

**Adjazenzlisten**

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 1 & 1 \\ 4 & 1 & 0 & 1 & 0 & 1 \\ 5 & 0 & 0 & 1 & 1 & 0 \end{array}$$

**Adjazenzmatrix**




gerichteter Graph



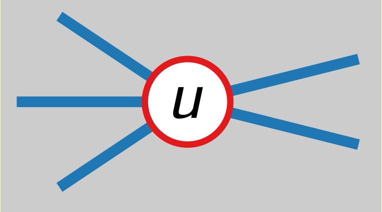

$$\begin{array}{c|ccccc} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 & 1 \\ 5 & 0 & 0 & 1 & 0 & 0 \end{array}$$

$\mathrm{Adj}[i] = \{j \in V \mid (i,j) \in E\}$

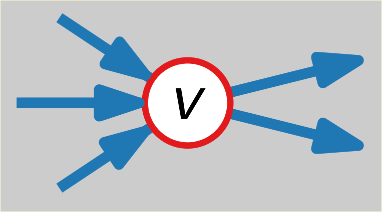
$a_{ij} = 1 \Leftrightarrow (i,j) \in E$

# Grad eines Knotens

**Def.**




$$\deg(u) = |\mathrm{Adj}[u]|$$




$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$

$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

**Beob.** Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$ .

**Beweis.** Technik des **zweifachen Abzählens:**

Zähle alle Knoten-Kanten-Inzidenzen.

Eine Kante ist **inzident** zu ihren Endknoten.

Ein Knoten ist **inzident** zu allen Kanten, deren Endknoten er ist.

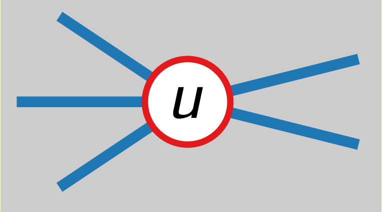
Aus Sicht der Knoten: $\sum_{v \in V} \deg(v)$

Aus Sicht der Kanten: $2 \cdot |E|$

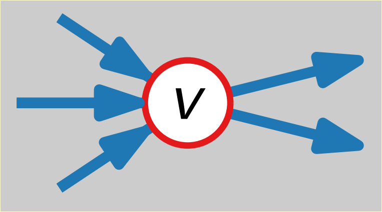
also gleich!

# Grad eines Knotens

**Def.**




$$\deg(u) = |\mathrm{Adj}[u]|$$




$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$

$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

**Beob.** Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= \; 2 \cdot |E|$ .

✓

**Sätzle.** Die Anzahl der Knoten ungeraden Grades ist gerade.

**Beweis.** $\underbrace{2 \cdot |E|}_{\textit{gerade!}} = \underbrace{\sum_{v \in V} \deg(v)}_{\textit{gerade!}} = \underbrace{\sum_{v \in V_{\mathrm{ger}}} \deg(v)}_{\textit{gerade!}} + \underbrace{\sum_{v \in V_{\mathrm{ung}}} \deg(v)}_{\Rightarrow \; \textit{gerade!}}$

$\sum_{v \in V_{\mathrm{ung}}} \deg(v)$ *gerade* $\Rightarrow$ $|V_{\mathrm{ung}}|$ ist *gerade!* $\square$

# Rundlaufstrategien für ungerichtete Graphen

1. Durchlaufe einen Graphen auf einem Kreis,
   so dass jede **Kante** genau einmal durchlaufen wird.

   **Charakterisierung:** Bei welchen Graphen geht das (nicht)?

   **Konstruktion:** Wie (und in welcher Zeit) finde ich einen solchen Rundlauf, falls er existiert?

   $\mathcal{P}$

2. Durchlaufe einen Graphen auf einem Kreis,
   so dass jeder **Knoten** genau einmal durchlaufen wird.

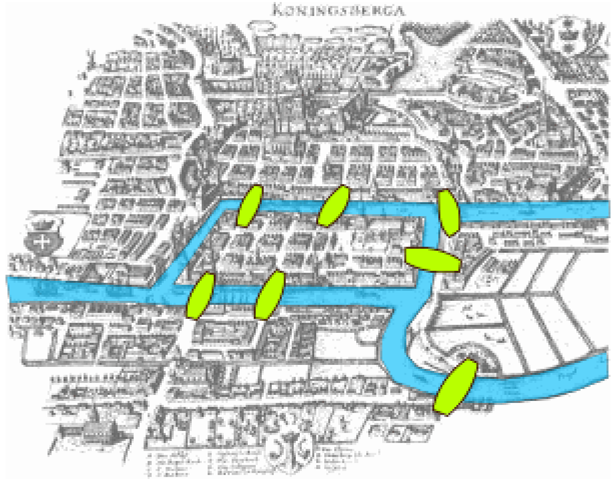
   **Charakterisierung:** Bei welchen Graphen geht das (nicht)?

   **Konstruktion:** Wie (und in welcher Zeit) finde ich einen solchen Rundlauf, falls er existiert?

   $\mathcal{NP}$-schwer

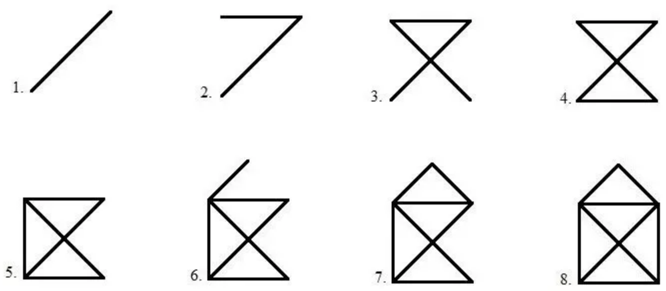

Kapitän Nemo, Public domain, via Wikimedia Commons

Königsberger Brückenproblem



Bogdan Giuşcă, CC BY-SA 3.0, via Wikimedia Commons

Das Haus vom Nikolaus

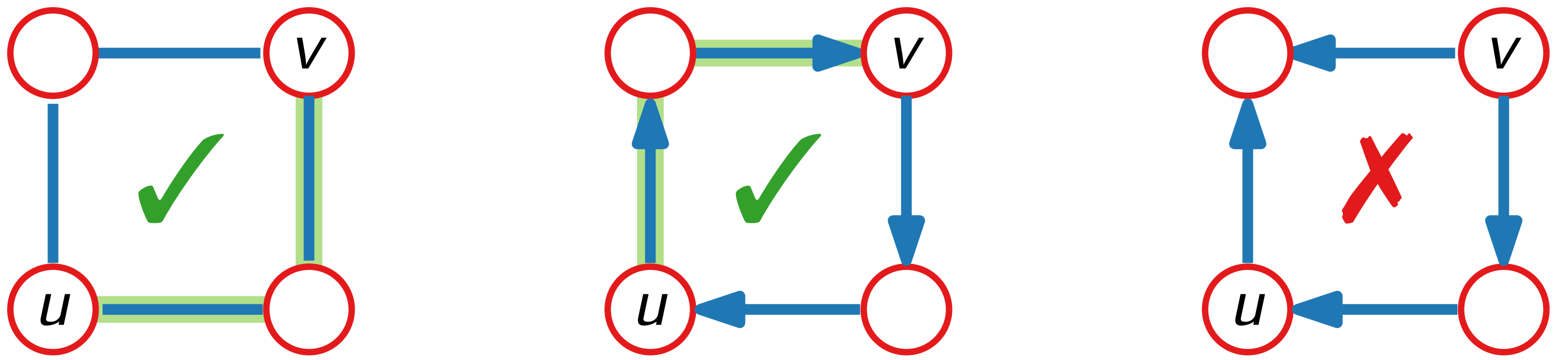



https://www.skizzen-zeichnungen.de/anleitung-zeichnen-vom-haus-vom-nikolaus/

# Zusammenhang

**Def.** Ein (un)gerichteter **Pfad** von einem Knoten $u$ zu einem Knoten $v$ ist eine Folge von Kanten, die in $u$ beginnt und in $v$ endet.

**Def.** Ein Knoten $v$ ist von einem Knoten $u$ aus **erreichbar**, wenn es einen (un)gerichteten Pfad von $u$ nach $v$ gibt.



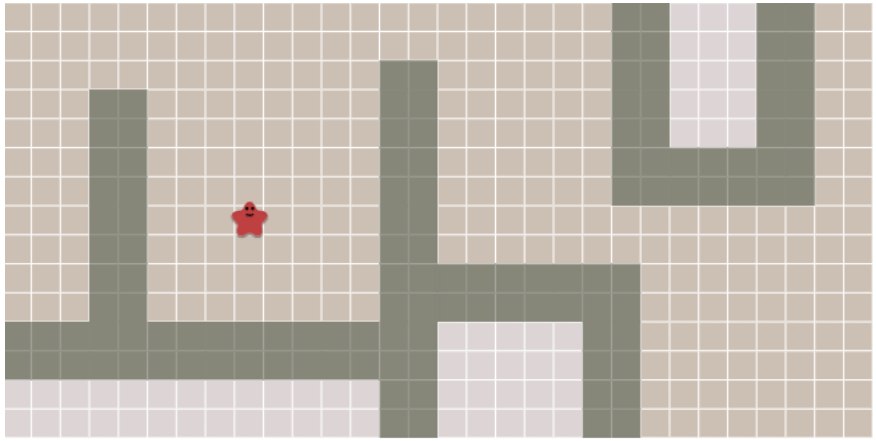

**Def.** Ein ungerichteter Graph heißt **zusammenhängend**, wenn jedes Knotenpaar voneinander aus erreichbar ist.

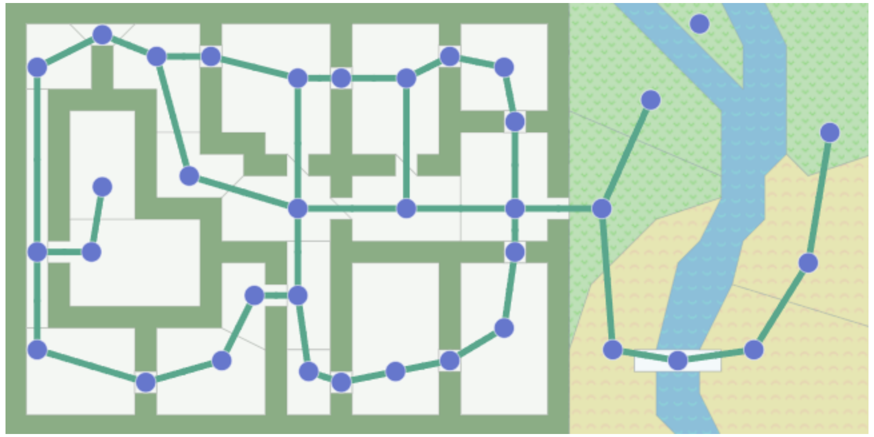
**Def.** Ein gerichteter Graph heißt **(schwach) zusammenhängend**, wenn der zugehörige ungerichtete Graph zusammenhängend ist.

**Def.** Einen maximalen zusammenhängenden Teilgraphen nennt man eine **Zusammenhangskomponente**.

# Wie durchlaufe ich einen Graphen?

Wie finde ich heraus, welche Knoten von einem Startknoten $s$ aus erreichbar sind?



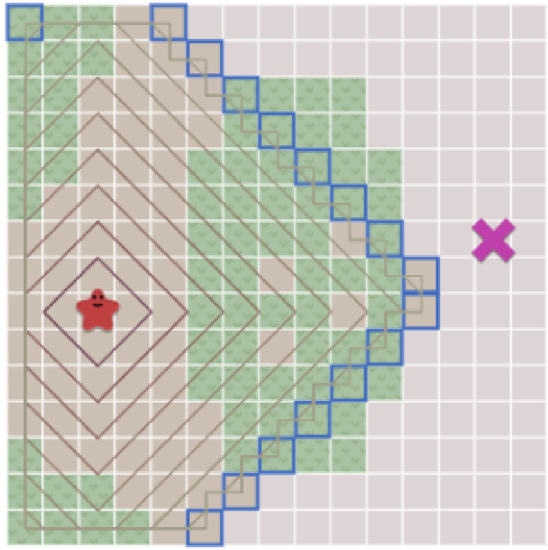


Amit Patel, "Introduction to the $A^*$ Algorithm", Red Blob Games, 2014, https://www.redblobgames.com/pathfinding/a-star/introduction.html

1. wellenförmige Ausbreitung ab $s$

   **Breitensuche (breadth-first search, BFS)**

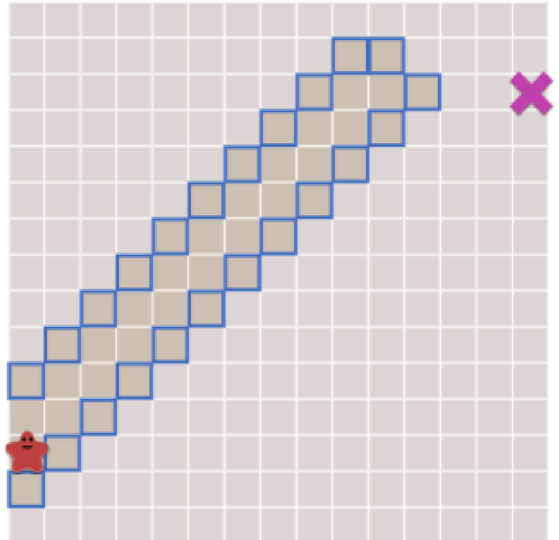
   heute



2. von $s$ möglichst schnell weit weg

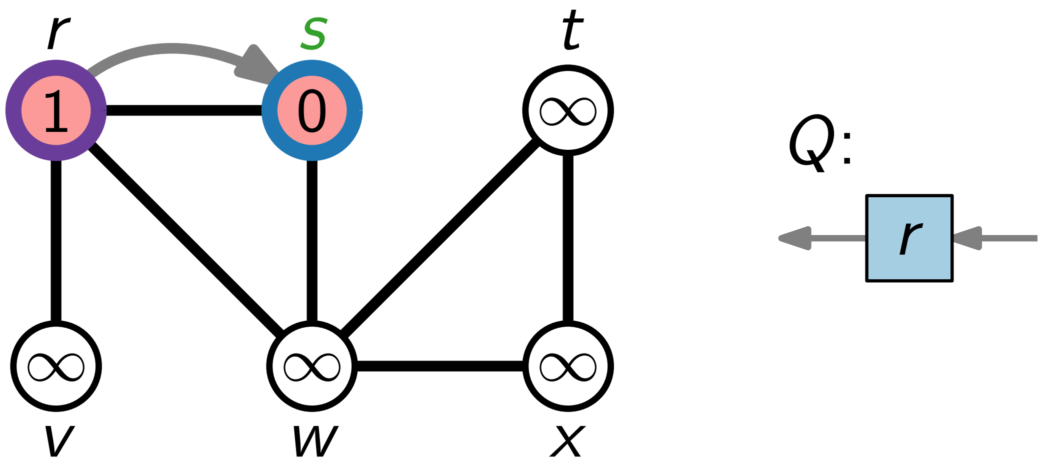
   **Tiefensuche (depth-first search, DFS)**

   nächstes Mal



# Breitensuche

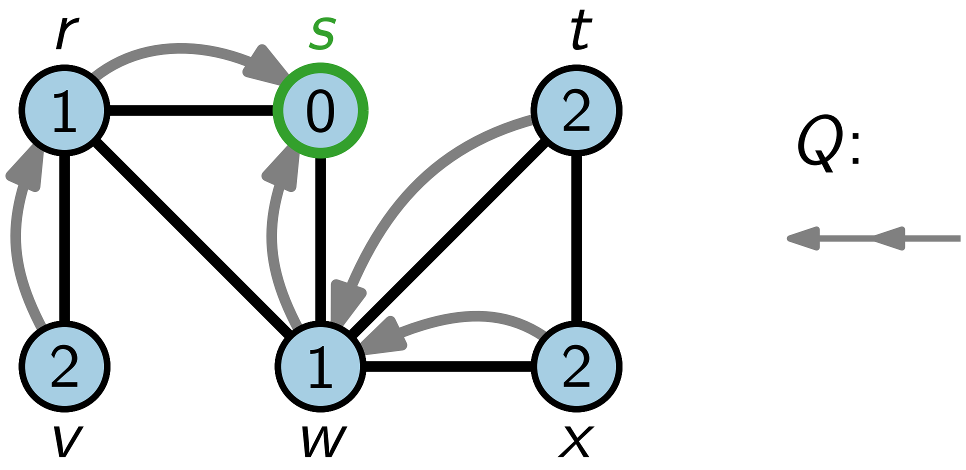
BFS(Graph $G$, Vertex $s$)
  Initialize($G, s$)
  $Q$ = new Queue()
  $Q$.Enqueue($s$)
  **while not** $Q$.Empty() **do**
    $u = Q$.Dequeue()
    **foreach** $v \in \mathrm{Adj}[u]$ **do**
      **if** $v.color == white$ **then**
        $v.color = red$
        $v.d = u.d + 1$
        $v.\pi = u$
        $Q$.Enqueue($v$)




Initialize(Graph $G$, Vertex $s$)
  **foreach** $u \in V$ **do**
    $u.color = white$
    $u.d = \infty$
    $u.\pi = nil$ Vorgänger
  $s.color = red$
  $s.d = 0$

# Breitensuche

BFS(Graph $G$, Vertex $s$)
  INITIALIZE($G, s$)
  $Q$ = new QUEUE()
  $Q$.ENQUEUE($s$)
  **while not** $Q$.EMPTY() **do**
    $u = Q$.DEQUEUE()
    **foreach** $v \in \mathrm{Adj}[u]$ **do**
      **if** $v.color == white$ **then**
        $v.color = red$
        $v.d = u.d + 1$
        $v.\pi = u$
        $Q$.ENQUEUE($v$)
    $u.color = blue$

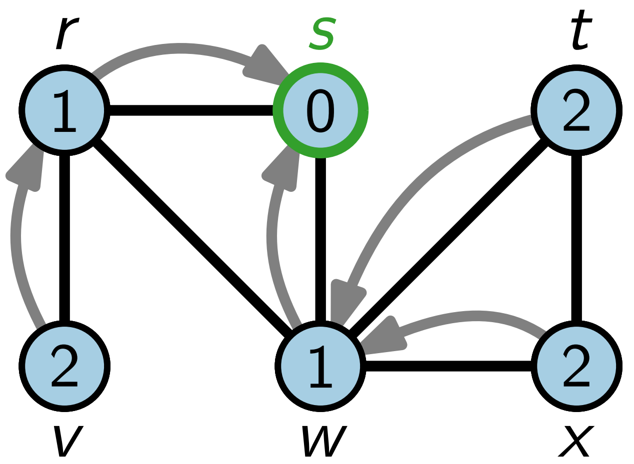



INITIALIZE(Graph $G$, Vertex $s$)
  **foreach** $u \in V$ **do**
    $u.color = white$
    $u.d = \infty$
    $u.\pi = nil$   Vorgänger
  $s.color = red$
  $s.d = 0$

**Demo.**
https://algo.uni-trier.de/demos/graphtraversal.html

# Breitensuche

$\text{BFS}(\text{Graph } G, \text{Vertex } s)$
- $\textsc{Initialize}(G, s)$
- $Q = \text{new } \textsc{Queue}()$
- $Q.\textsc{Enqueue}(s)$
- **while not** $Q.\textsc{Empty}()$ **do**
  - $u = Q.\textsc{Dequeue}()$
  - **foreach** $v \in \text{Adj}[u]$ **do**
    - **if** $v.\mathit{color} == \mathit{white}$ **then**
      - $v.\mathit{color} = \mathit{red}$
      - $v.d = u.d + 1$
      - $v.\pi = u$
      - $Q.\textsc{Enqueue}(v)$
  - $u.\mathit{color} = \mathit{blue}$




$Q$:

$\textsc{Initialize}(\text{Graph } G, \text{Vertex } s)$
- **foreach** $u \in V$ **do**
  - $u.\mathit{color} = \mathit{white}$
  - $u.d = \infty$
  - $u.\pi = \mathit{nil}$ Vorgänger
- $s.\mathit{color} = \mathit{red}$
- $s.d = 0$

**Laufzeit?**

$$\underbrace{\mathcal{O}(|V|)}_{\textsc{Initialize}} + \underbrace{\mathcal{O}(|V|)}_{\textsc{En-/Dequeues}} + \underbrace{\mathcal{O}(|E|)}_{\text{Adjazenzlisten (foreach-Schleifen)}} = \mathcal{O}(|V| + |E|)$$

Beob. über Knotengrade!

# Korrektheit von BFS – Vorbereitung

**Definition.** Sei $G = (V, E)$ (un)gerichteter Graph, $u, v \in V$.
$\delta(u, v) :=$ Länge eines kürzesten $u$-$v$-Wegs,
(falls $v$ von $u$ erreichbar; sonst $\delta(u, v) := \infty$).

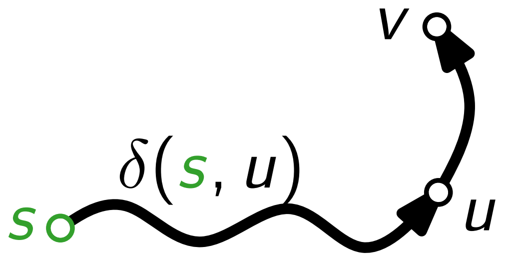
**Ziel:** Zeige, dass nach $\mathrm{BFS}(G, s)$ für alle $v \in V$ gilt:

$$\underbrace{v.d}_{\text{berechneter Abstand von } s} = \underbrace{\delta(s, v)}_{\text{tatsächlicher Abstand von } s}.$$

**Lemma 1.** (Eigenschaft kürzester Wege)
Sei $G = (V, E)$ ein (un)gerichteter Graph, $s \in V$.
Dann gilt für jede Kante $(u, v) \in E$:

$$\delta(s, v) \leq \delta(s, u) + 1.$$

**Beweis.** 1. Fall: $u$ ist von $s$ erreichbar (d.h. $\exists$ $s$-$u$-Weg)




**Dieser** $s$-$v$-Weg hat Länge $\delta(s, u) + 1$.

$\Downarrow$

**Kürzester** $s$-$v$-Weg hat Länge $\leq \delta(s, u) + 1$.

# Korrektheit von BFS – Vorbereitung

**Definition.** Sei $G = (V, E)$ (un)gerichteter Graph, $u, v \in V$.
$\delta(u, v) :=$ Länge eines kürzesten $u$-$v$-Wegs,
(falls $v$ von $u$ erreichbar; sonst $\delta(u, v) := \infty$).

**Ziel:** Zeige, dass nach $\mathrm{BFS}(G, s)$ für alle $v \in V$ gilt:

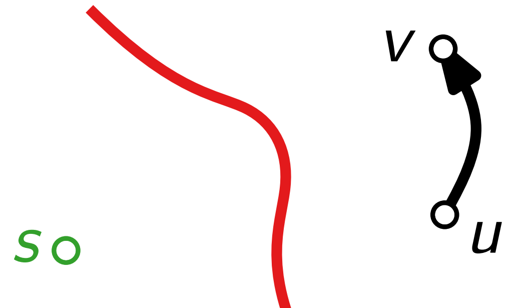
$$v.d = \delta(s, v).$$

**berechneter** Abstand von $s$ — **tatsächlicher** Abstand von $s$

**Lemma 1.** (Eigenschaft kürzester Wege)
Sei $G = (V, E)$ ein (un)gerichteter Graph, $s \in V$.
Dann gilt für jede Kante $(u, v) \in E$:

$$\delta(s, v) \leq \delta(s, u) + 1.$$

**Beweis.** 2. Fall: $u$ ist **nicht** von $s$ erreichbar (d.h. $\nexists$ $s$-$u$-Weg)




$\Rightarrow \delta(s, u) = \infty$
$\Rightarrow \delta(s, v) \leq \infty + 1$ ✓

# Korrektheit von BFS – Fortsetzung

**Lemma 1.** Sei $s \in V$. Dann gilt für jede Kante $(u, v) \in E$:
$$\delta(s, v) \leq \delta(s, u) + 1.$$
✓

**Lemma 2.** Sei $G = (V, E)$ ein (un)gerichteter Graph, $s \in V$.
Nach $\mathrm{BFS}(G, s)$ gilt für alle $v \in V$: $v.d \geq \delta(s, v)$.

**Beweis.** Induktion über die Anzahl $k$ von ENQUEUE-Operationen.

```
BFS(Graph G, Vertex s)
  INITIALIZE(G, s)
  Q = new QUEUE()
  Q.ENQUEUE(s)
  while not Q.EMPTY() do
    u = Q.DEQUEUE()
    foreach v ∈ Adj[u] do
      if v.color == white then
        v.color = red
        v.d = u.d + 1
        v.π = u
        Q.ENQUEUE(v)
    u.color = blue
```

$k = 1$: Situation nach $Q.\mathrm{ENQUEUE}(s)$: ✓

- $s.d = 0 = \delta(s, s)$
- für alle $v \in V \setminus \{s\}$ gilt $v.d = \infty \geq \delta(s, v)$

$k > 1$: Situation nach $Q.\mathrm{ENQUEUE}(v)$: ✓

$v$ war gerade noch weiß und ist benachbart zu $u$.

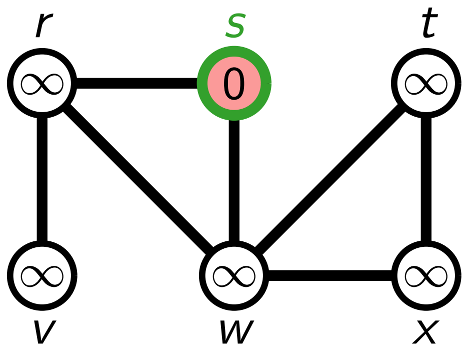
$$v.d = u.d + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$$

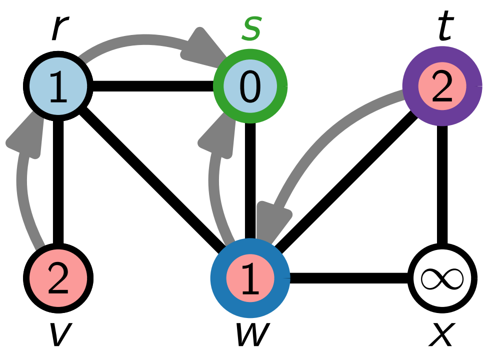
Induktionsannahme für $u$

Lemma 1

($u.d$ wurde gesetzt, als Anz. ENQUEUE-Oper. $< k$)

Jetzt ist $v$ rot. $\Rightarrow$ $v.d$ ändert sich nicht mehr. □



# Korrektheit von BFS – Fortsetzung

**Lemma 2.** Sei $G = (V, E)$ ein (un)gerichteter Graph, $s \in V$. Nach BFS$(G, s)$ gilt für alle $v \in V$: $v.d \geq \delta(s, v)$. ✓

**Lemma 3.** Sei $Q = \langle v_1, v_2, \dots, v_r \rangle$ während BFS. Dann gilt:
(A) $v_r.d \leq v_1.d + 1$ und
(B) $v_i.d \leq v_{i+1}.d$ für $i = 1, \dots, r-1$.

Also $d$-Werte der Knoten in $Q$ z.B. $\langle 3, 3, 4, 4, 4 \rangle$.

**Korollar.** Angenommen $u$ wird früher als $v$ in $Q$ eingefügt, dann gilt $u.d \leq v.d$, wenn $v$ in $Q$ eingefügt wird.

**Beweis.** Folgt aus Lemma 3 und der Tatsache, dass jeder Knoten $\leq 1\times$ einen endlichen $d$-Wert bekommt.

# Korrektheit von BFS – Hauptsatz

**Satz.** Sei $G$ ein (un)gerichteter Graph, $s$ ein Knoten von $G$. Nach BFS$(G, s)$ gilt:

(i) Für alle Knoten $v \in V$ gilt $v.d = \delta(s, v)$.

(ii) Jeder von $s$ erreichbare Knoten wird entdeckt.

(iii) Für jeden von $s$ erreichbaren Knoten $v \neq s$ gilt: es gibt einen kürzesten $s$-$v$-Weg, der aus einem kürzesten $s$-$v.\pi$-Weg und der Kante $(v.\pi, v)$ besteht.

**Beweis.** (i) $\Rightarrow$ (ii), (iii). Es genügt also (i) zu zeigen.

Lemma 2 $\Rightarrow v.d \geq \delta(s, v)$. Noch z.z.: $v.d \leq \delta(s, v)$.

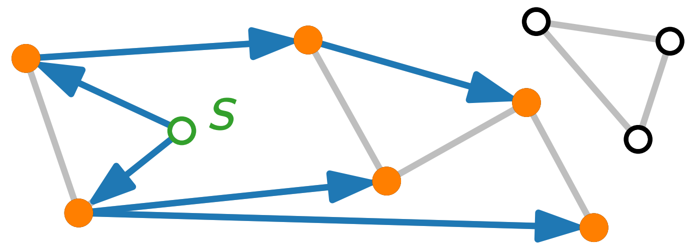
Widerspruchsbeweis mit Wahl des „kleinsten Schurken“.

Siehe Kapitel 22.2 [CLRS].

# BFS-Bäume

Betrachte den **Vorgänger-Graphen** $G_\pi = (V_\pi, E_\pi)$ von $G$:

- $\boxed{V_\pi} = \{v \in V : v.\pi \neq nil\} \cup \{s\}$
- $\boxed{E_\pi} = \{(v.\pi, v) : v \in V_\pi \setminus \{s\}\}$




Klar: $G_\pi$ ist ein Baum (da zshg. und $|E_\pi| = |V_\pi| - 1$).

**Behauptung:** $G_\pi$ ist ein **Kürzeste-Wege-Baum** (oder **BFS-Baum**), d.h.

- $V_\pi = \{v \in V : v \text{ erreichbar von } s\}$
- für alle $v \in V_\pi$ enthält $G_\pi$ einen eindeutigen Weg von $s$ nach $v$, der ein kürzester $s$-$v$-Weg ist.

**Beweis:** Folgt aus (ii) und (iii) im Hauptsatz. $\square$