



Julius-Maximilians-

**UNIVERSITÄT
WÜRZBURG**

Lehrstuhl für

INFORMATIK I

Algorithmen & Komplexität



Institut für Informatik

Algorithmen und Datenstrukturen

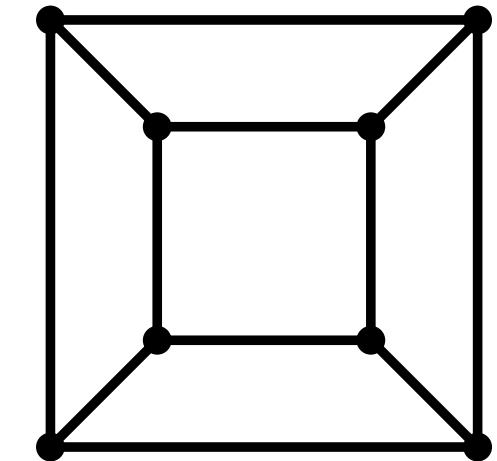
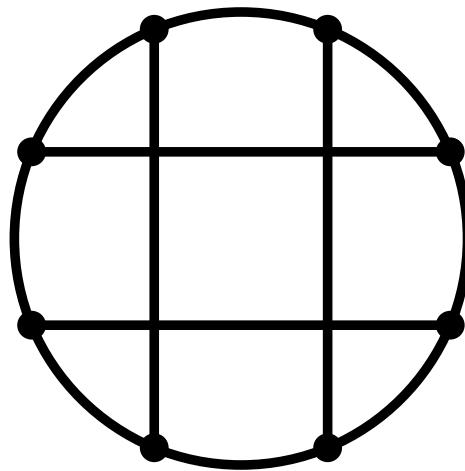
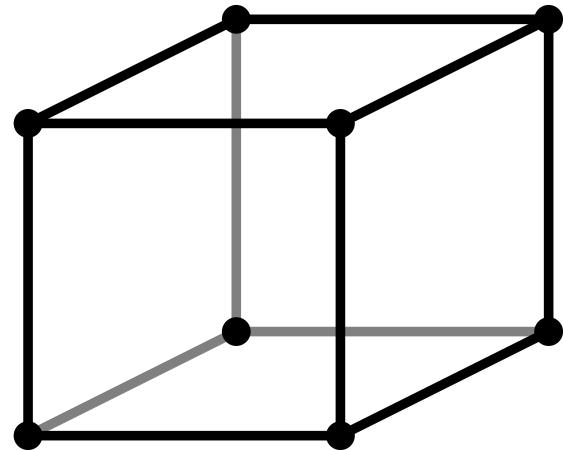
Wintersemester 2020/21

18. Vorlesung

Graphen:

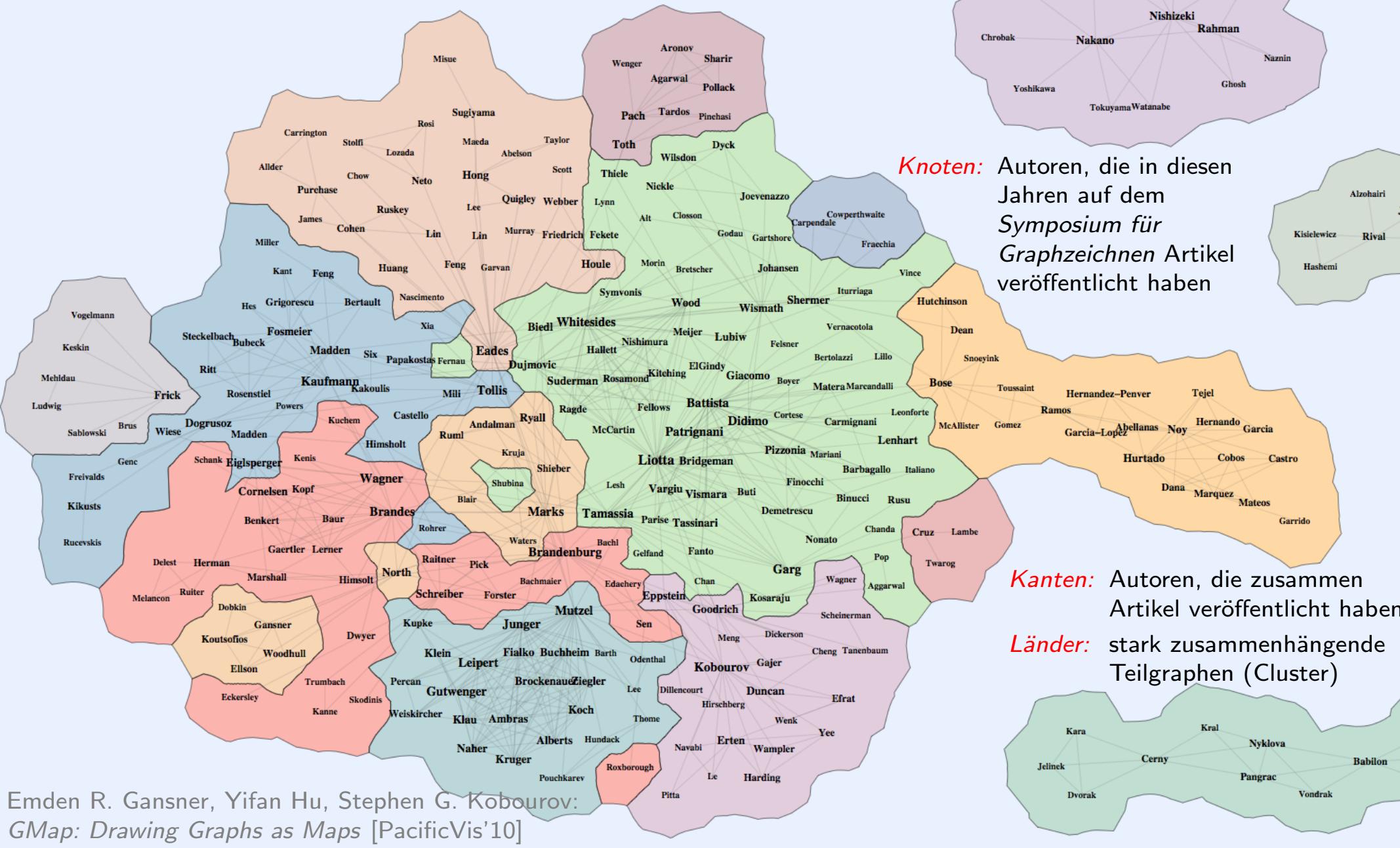
Repräsentation und Durchlaufstrategien

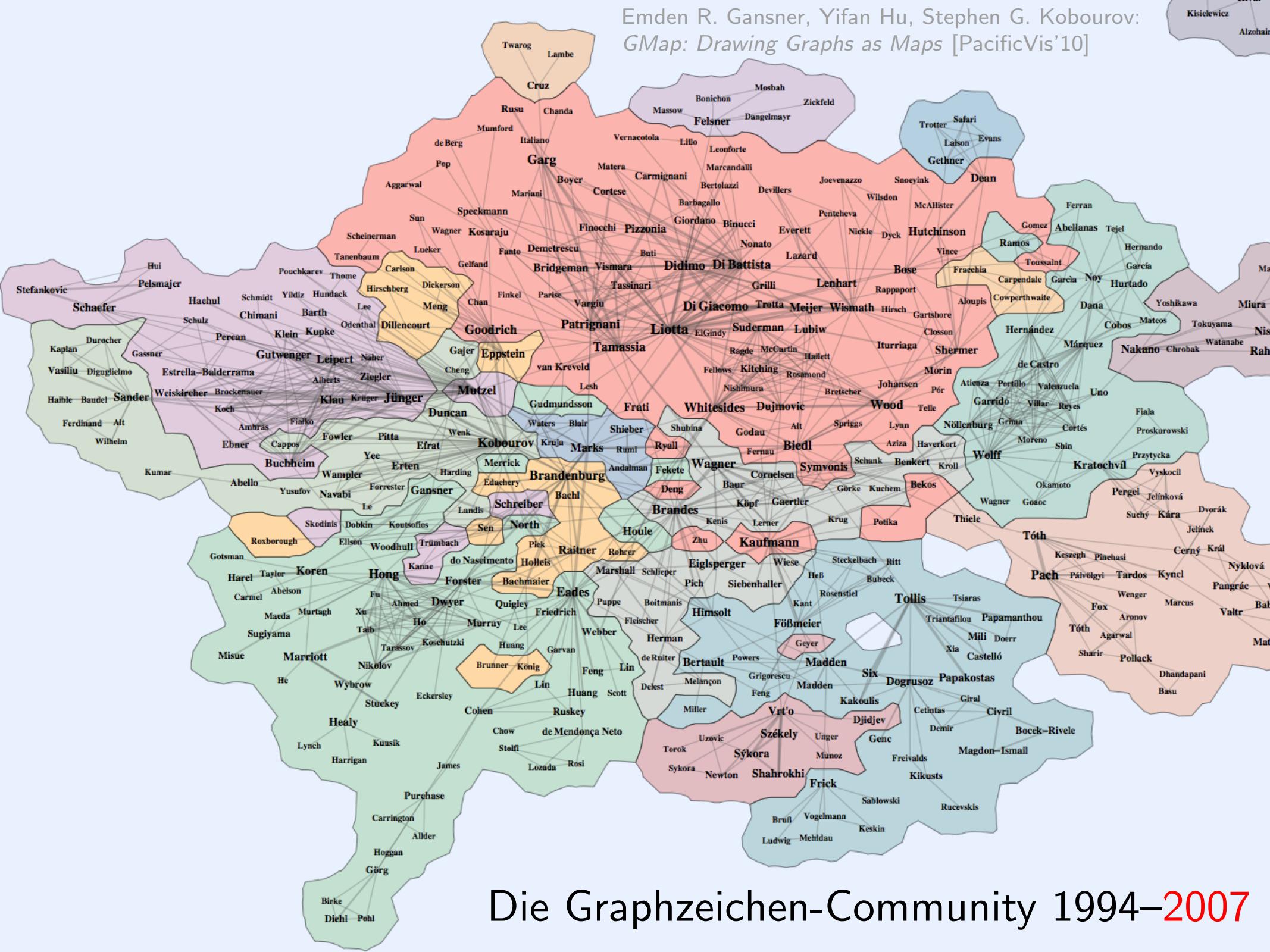
Was ist das?



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

Die Graphzeichen-Community 1994–2004





Die Graphzeichen-Community 1994–2007

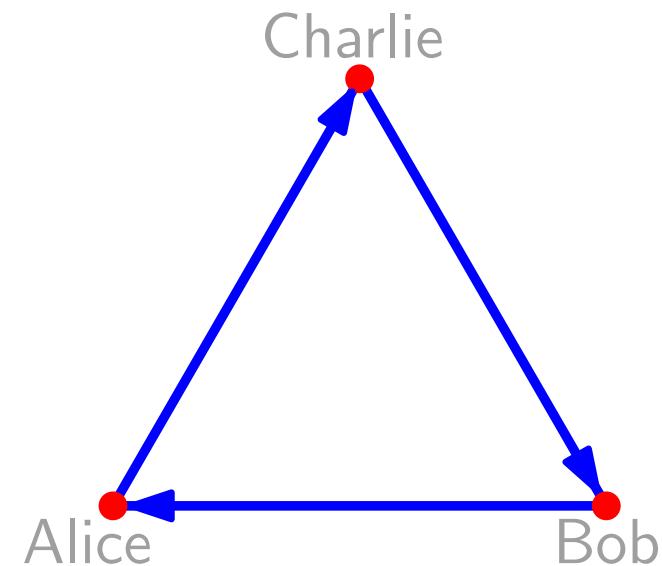
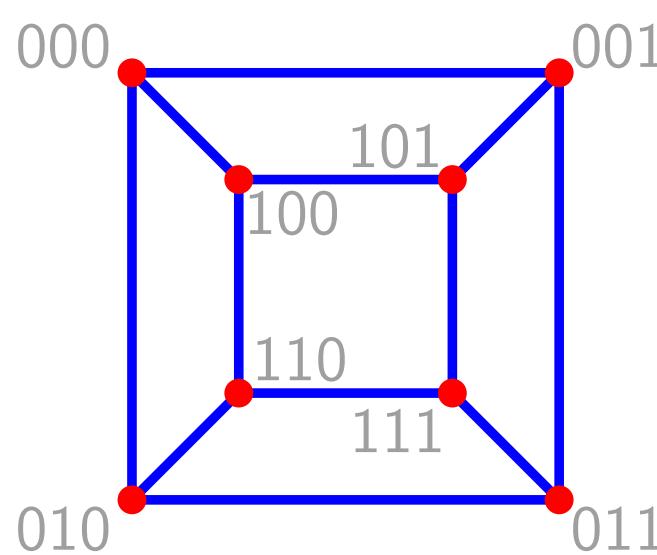
F: Was ist ein Graph?

A₁: 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.

$$V = \{000, 001, \dots, 111\}$$

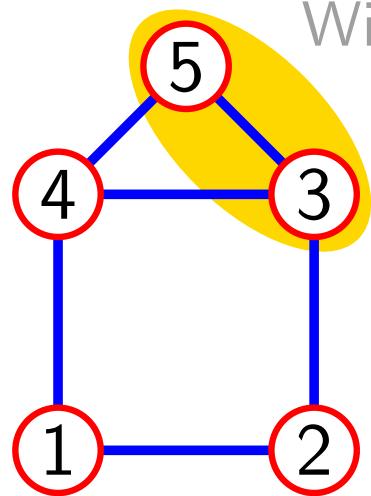
$$\{u, v\} \in E \Leftrightarrow ?$$



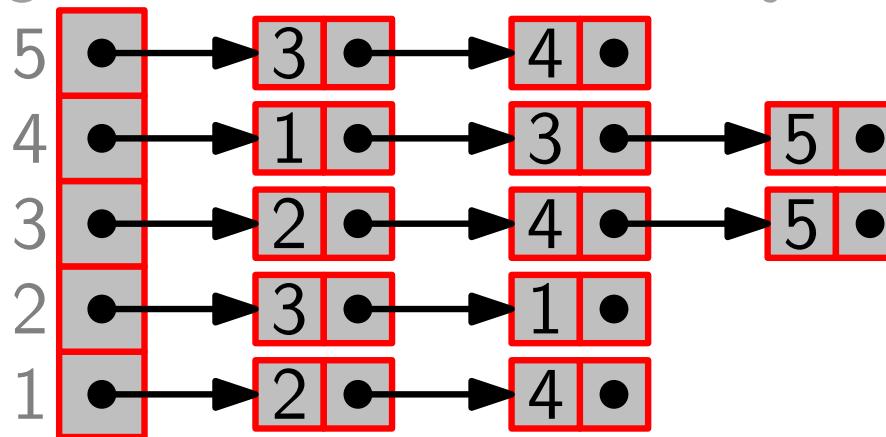
A₂: Ein gerichteter Graph ist ein Paar (V, E) , wobei

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

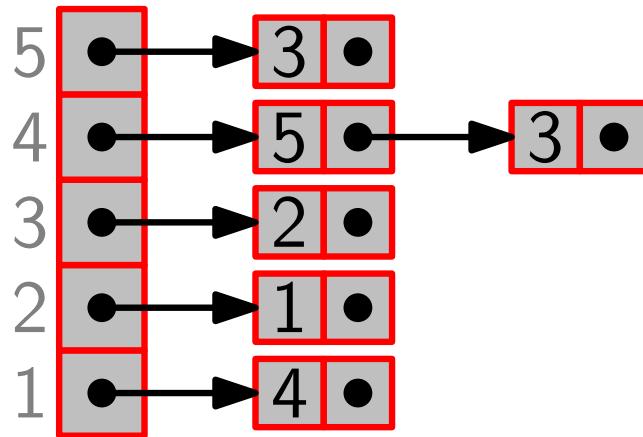
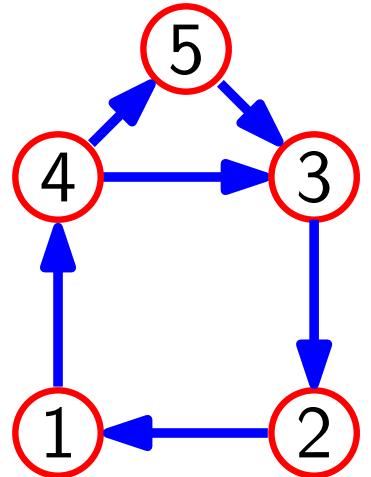
F: Wie repräsentiere ich einen Graphen?



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



	1	2	3	4	5
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

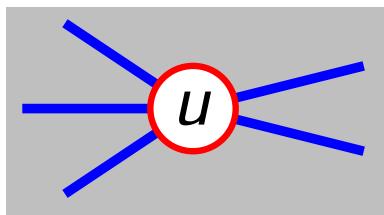


	1	2	3	4	5
1	0	0	0	1	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	1	0	1
5	0	0	1	0	0

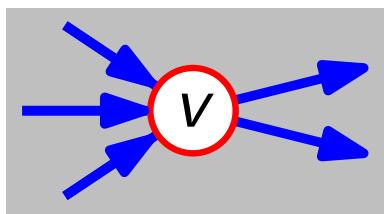
$$a_{ij} = 1 \Leftrightarrow (i, j) \in E$$

Grad eines Knotens

Def.



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



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

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

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.

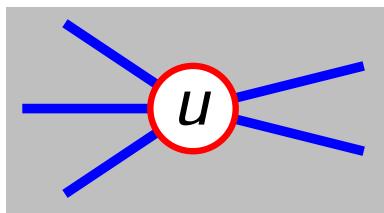
Ein Knoten ist *inzident* zu allen Kanten, deren Endknoten er ist.

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

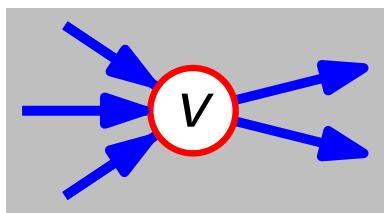
Aus Sicht der Kanten: $2 \cdot |E|$ also gleich!

Grad eines Knotens

Def.



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



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

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

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.

$$2 \cdot |E| = \sum_{v \in V} \deg v = \sum_{v \in V_{\text{ger}}} \deg v + \sum_{v \in V_{\text{ung}}} \deg v$$

gerade! *gerade!* *gerade!* \Rightarrow *gerade!*

$$\sum_{v \in V_{\text{ung}}} \deg v \text{ gerade } \Rightarrow |V_{\text{ung}}| \text{ ist gerade!}$$

□

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?

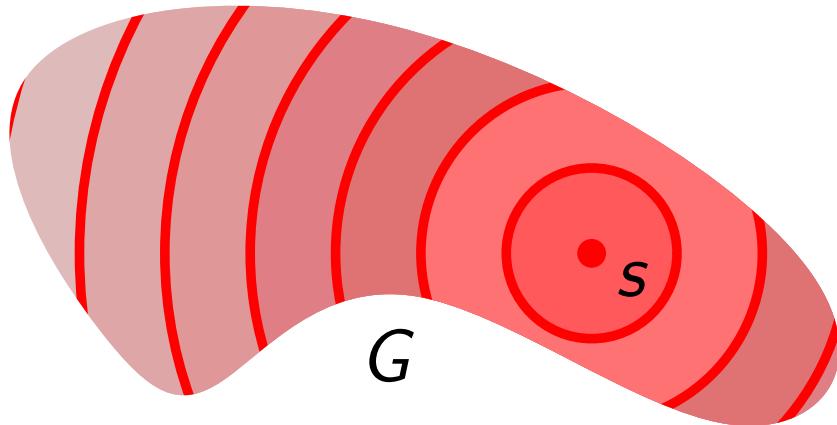
2. Durchlaufe einen Graphen auf einem Kreis,
so dass jeder **Knoten** genau einmal durchlaufen wird.

Charakterisierung: Bei welchen Graphen geht das (nicht)?

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

F: Wie durchlaufe ich einen Graphen?

Ideen?



1. wellenförmige Ausbreitung ab einem gegebenen Startknoten s

Breitensuche (breadth-first search, BFS) ← jetzt!

2. vom Startknoten s möglichst schnell weit weg

Tiefensuche (depth-first search, DFS)

nächstes Mal!

Breitensuche

BFS(Graph G , Vertex s)

Initialize(G, s)

$Q = \text{new Queue}()$

$Q.\text{Enqueue}(s)$

while not $Q.\text{Empty}()$ **do**

$u = Q.\text{Dequeue}()$

foreach $v \in \text{Adj}[u]$ **do**

if $v.\text{color} == \text{white}$ **then**

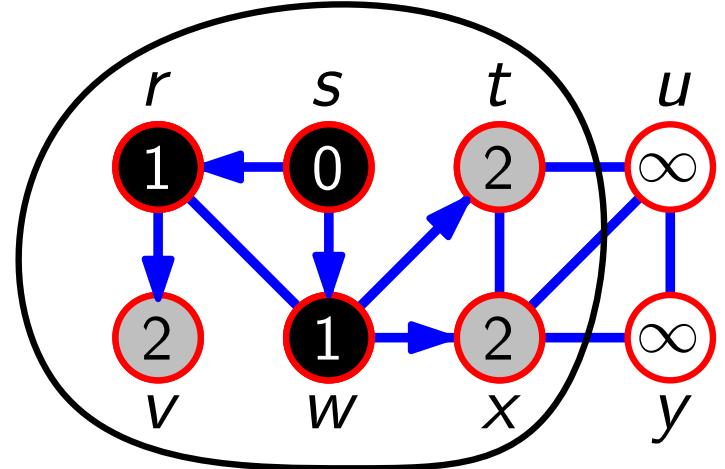
$v.\text{color} = \text{gray}$

$v.d = u.d + 1$

$v.\pi = u$

$Q.\text{Enqueue}(v)$

$u.\text{color} = \text{black}$



$w \quad r \quad t \quad x \quad v \quad \quad \quad usw.$

Initialize(Graph G , Vertex s)

foreach $u \in V$ **do**

$u.\text{color} = \text{white}$

$u.d = \infty$

$u.\pi = \text{nil}$

$s.\text{color} = \text{gray}$

$s.d = 0$

Laufzeit?

Initialize	En-/Dequeues	Adjazenzlisten (foreach-Schleifen)
$O(V)$	$+ O(V)$	$+ O(E) = O(V + E)$
<small>[Beob. über Knotengrade!]</small>		

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 $\text{BFS}(G, s)$ für alle $v \in V$ gilt:

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

berechneter Abstand von s *tatsächlicher Abstand von s*

Lemma 1.

(Eigenschaft kürzester Wege)

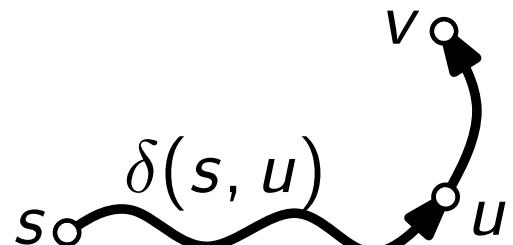
Sei $G = (V, E)$ ein (un)gerichteter Graph, $s \in V$.

Dann gilt für jede Kante $(u, v) \in E$:

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

Beweis.

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



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



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

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 $\text{BFS}(G, s)$ gilt für alle $v \in V$: $v.d \geq \delta(s, v)$.

Beweis. Induktion über die Anz. k von Enqueue-Oper.

```
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 = gray
        v.d = u.d + 1
        v.π = u
        Q.Enqueue(v)
    u.color = black
```

$k = 1$: Situation nach $Q.\text{Enqueue}(s)$:

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

$k > 1$: Situation nach $Q.\text{Enqueue}(v)$:

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 grau. $\Rightarrow v.d$ ändert sich nicht mehr.



Korrektheit von BFS – Fortsetzung

Lemma 2. Sei $G = (V, E)$ ein (un)gerichteter Graph, $s \in V$. Nach $\text{BFS}(G, s)$ 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 $\text{BFS}(G, s)$ 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 . π -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.: $v.d \leq \delta(s, v)$.

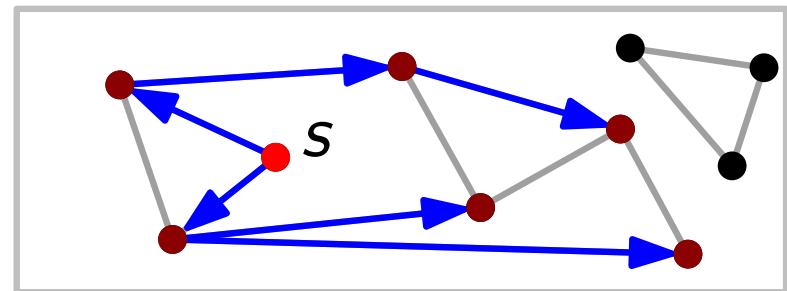
Widerspruchsbeweis mit Wahl des „kleinsten Schurken“.

Siehe Kapitel 22.2 [CLRS].

BFS-Bäume

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

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



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

Beh.: G_π ist ein *Kürzeste-Wege-Baum* (oder *BFS-Baum*), d.h.

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

Bew.: Folgt aus (ii) und (iii) im Hauptsatz. □