Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Lehrstuhl für
INFORMATIK I
Algorithmen & Komplexität

Institut für Informatik

Algorithmen und Datenstrukturen

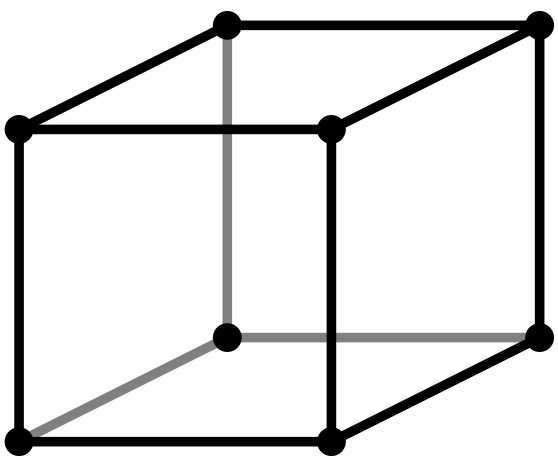
Wintersemester 2023/24

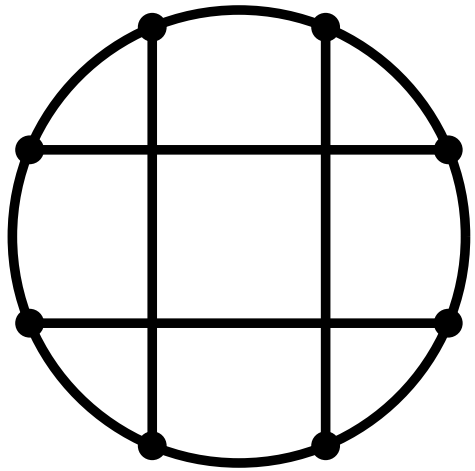
18. Vorlesung

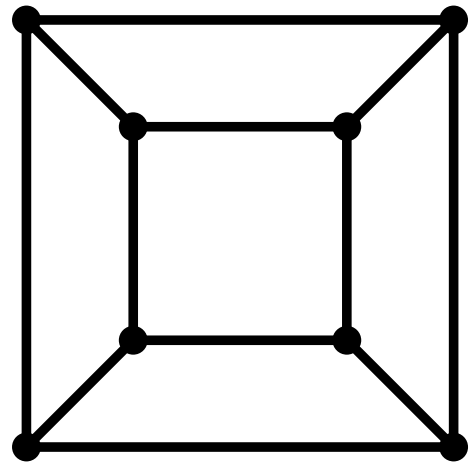
Graphen: Repräsentation und Durchlaufstrategien

Prof. Dr. Alexander Wolff *Lehrstuhl für Informatik I*

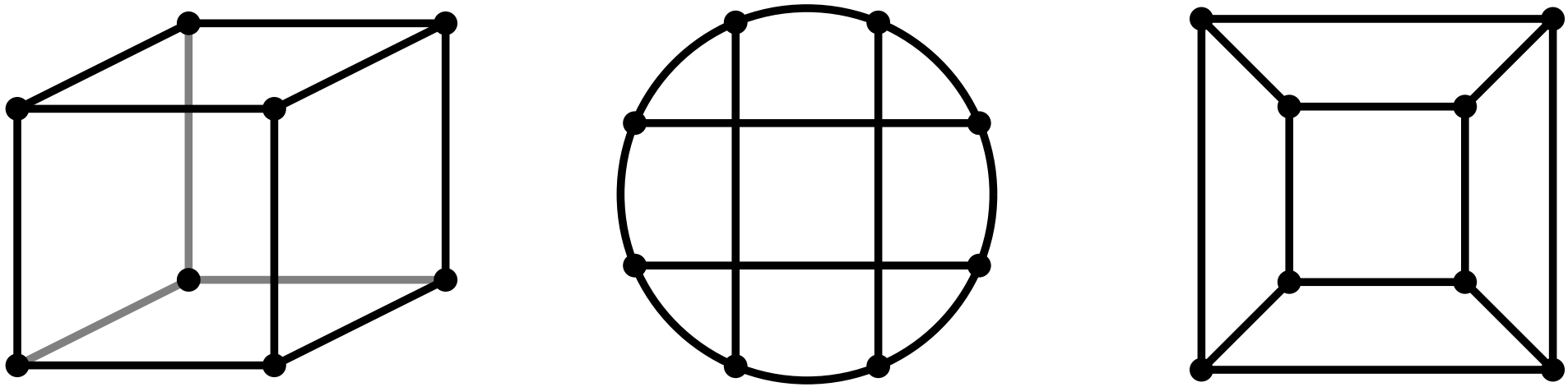
Was ist das?





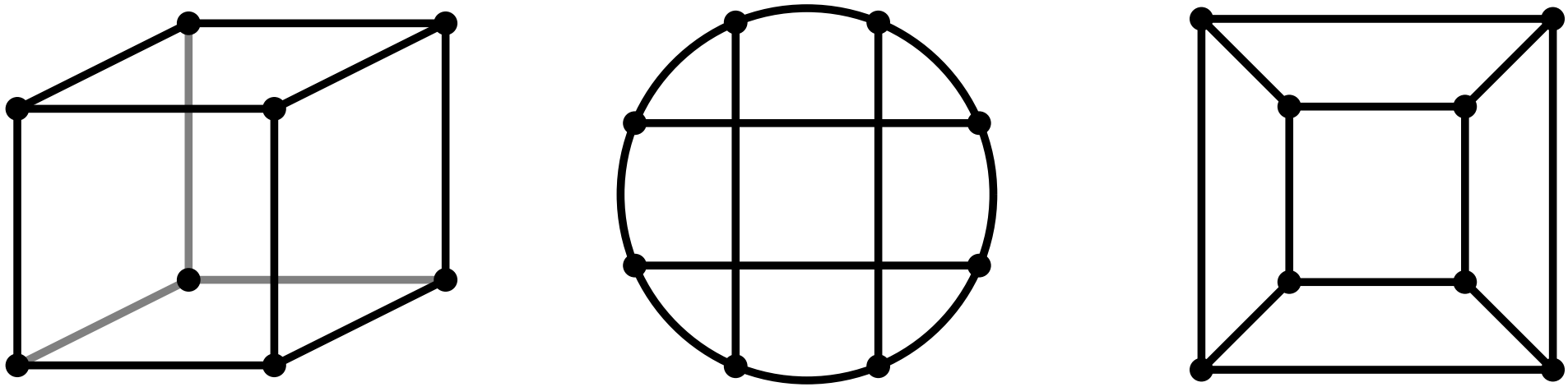


Was ist das?



Ein (und derselbe) *Graph*.

Was ist das?



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

StudiVZ am 9. Dezember 2006

Insgesamt 1.074.574 Profile (davon 1.035.890 öffentlich)

Davon 430.000 *aktiv*; ein Profil ist aktiv, wenn

- das Profil öffentlich ist,
- die Person mindestens zwei Freunde hat,
- in mindestens einer Gruppe ist und
- das Profil innerhalb des letzten Monats aktualisiert wurde.

StudiVZ am 9. Dezember 2006

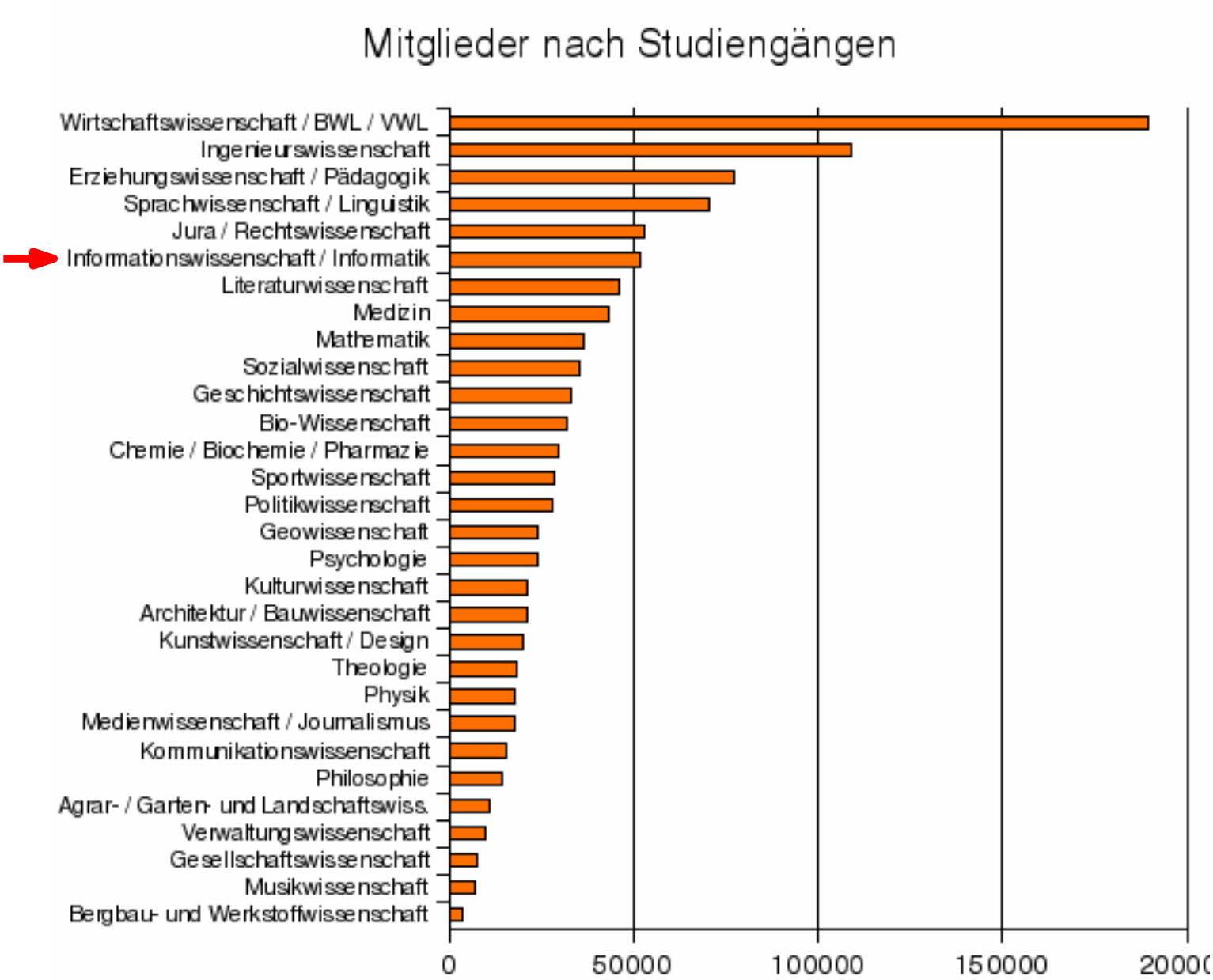
Insgesamt 1.074.574 Profile (davon 1.035.890 öffentlich)

Davon 430.000 *aktiv*; ein Profil ist aktiv, wenn

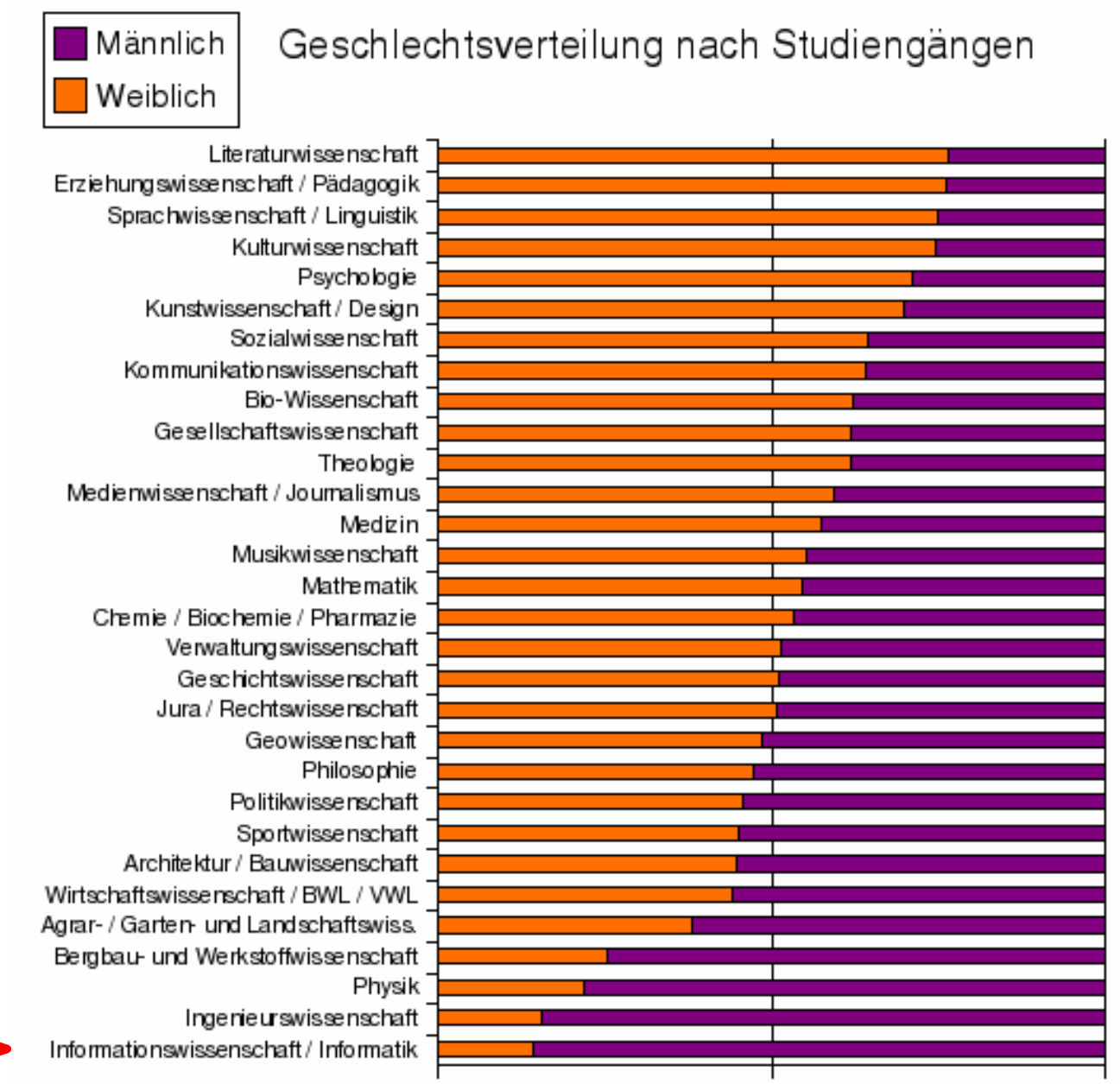
- das Profil öffentlich ist,
- die Person mindestens zwei Freunde hat,
- in mindestens einer Gruppe ist und
- das Profil innerhalb des letzten Monats aktualisiert wurde.

Ein wenig Statistik über die Mitglieder, sortiert nach Studienfach...

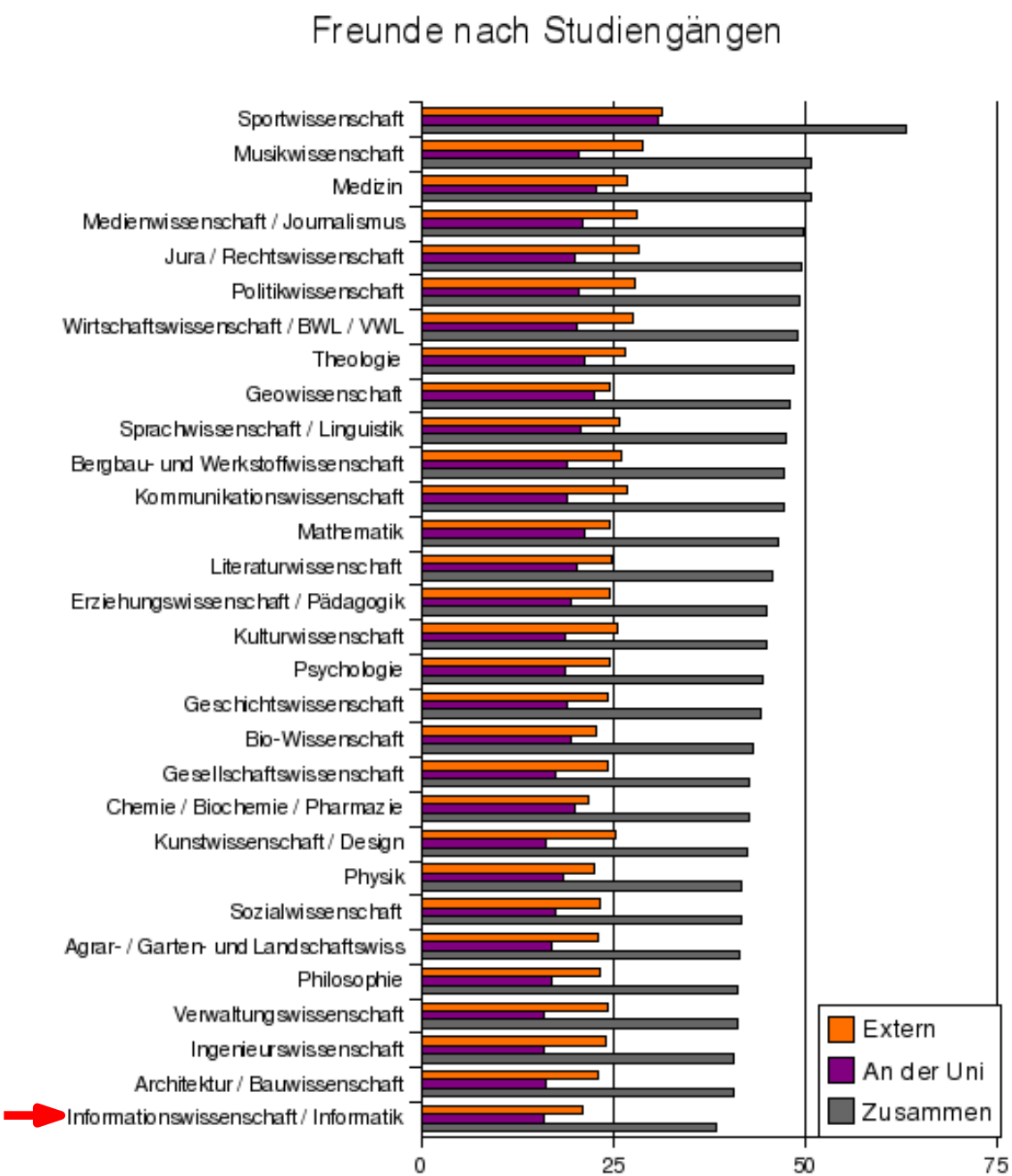
StudiVZ-Statistiken



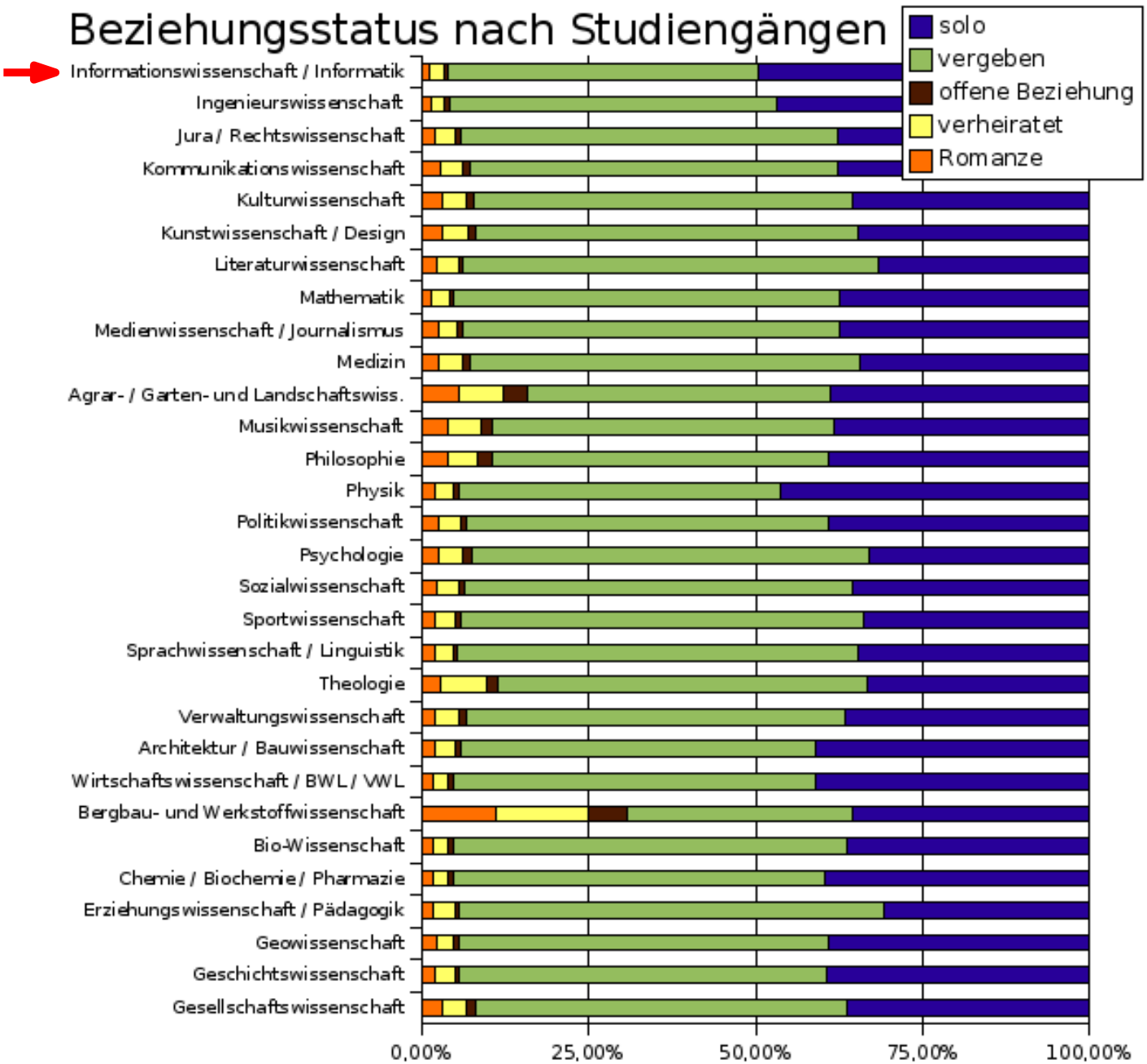
StudiVZ-Statistiken



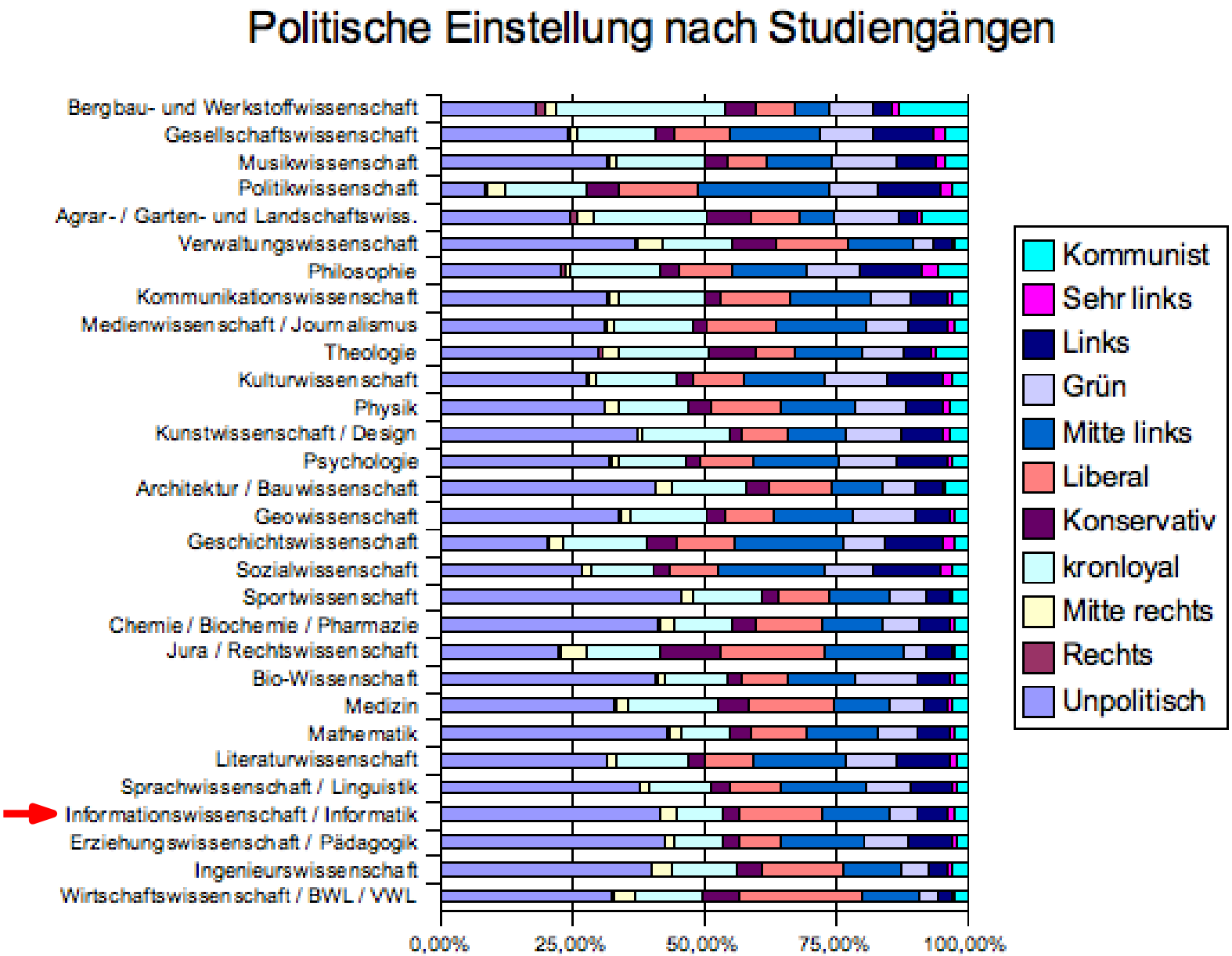
StudiVZ-Statistiken



StudiVZ-Statistiken



StudiVZ-Statistiken



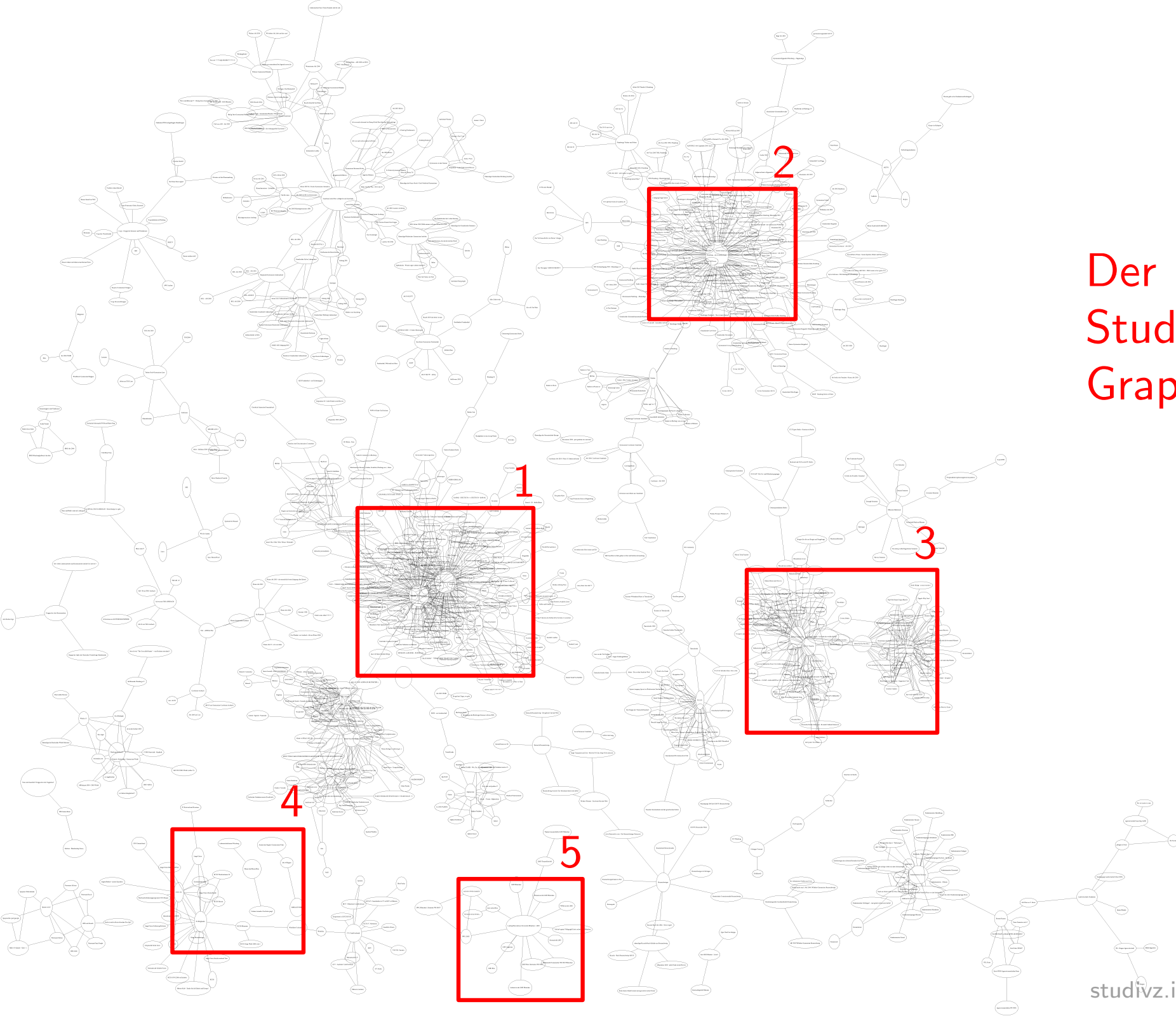
Der „StudiVZ-Graph“

Der „StudiVZ-Graph“ enthält nur Gruppen, die

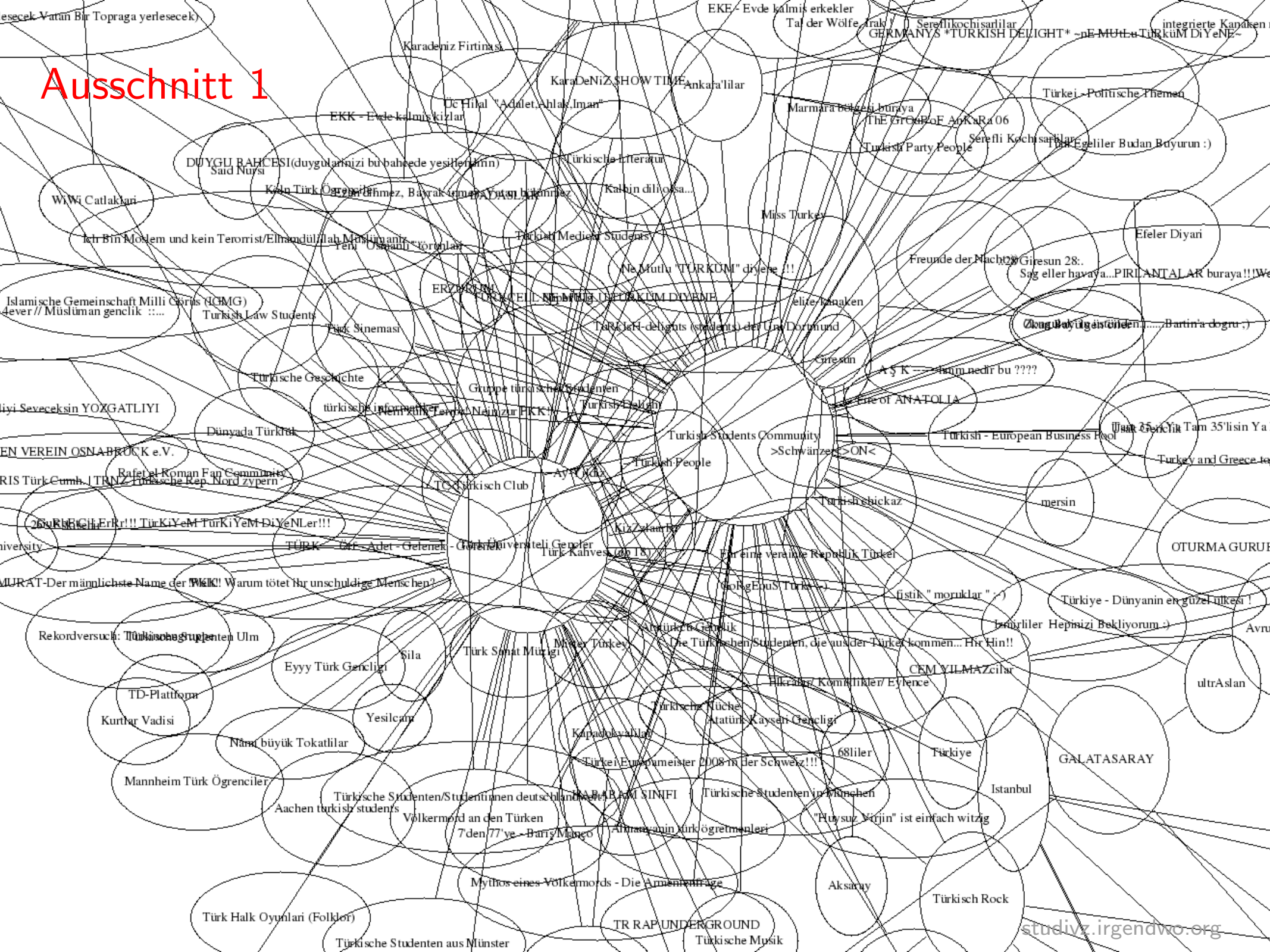
- mindestens 10 Mitglieder haben und
- zu einem Cluster (= stark verbundener Teilgraph) gehören.

Zwei Gruppen sind durch eine Kante verbunden, wenn $\geq 45\%$ der Mitglieder einer Gruppe auch Mitglieder der anderen sind.

Der StudiVZ-Graph

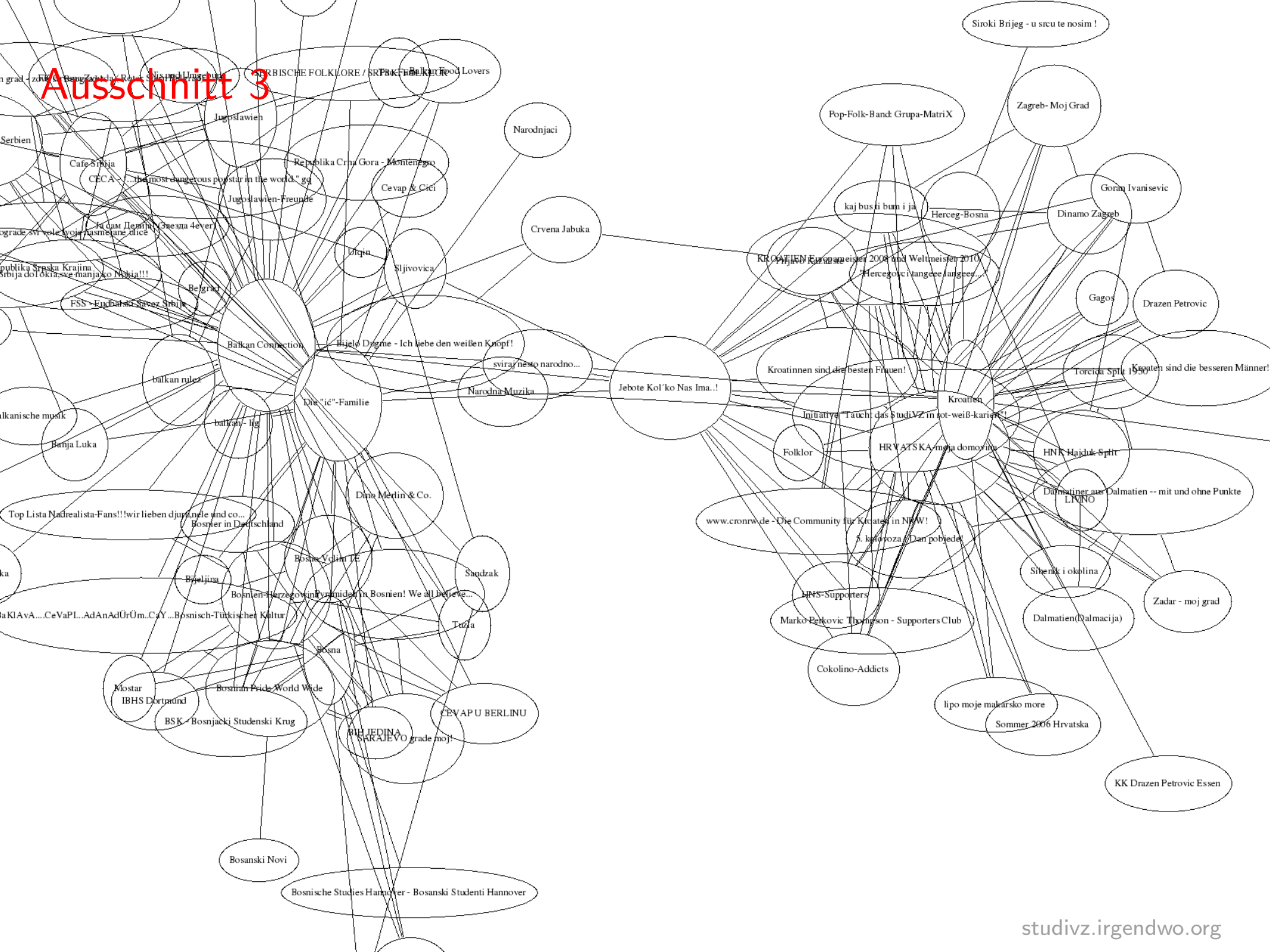


Der StudiVZ-Graph

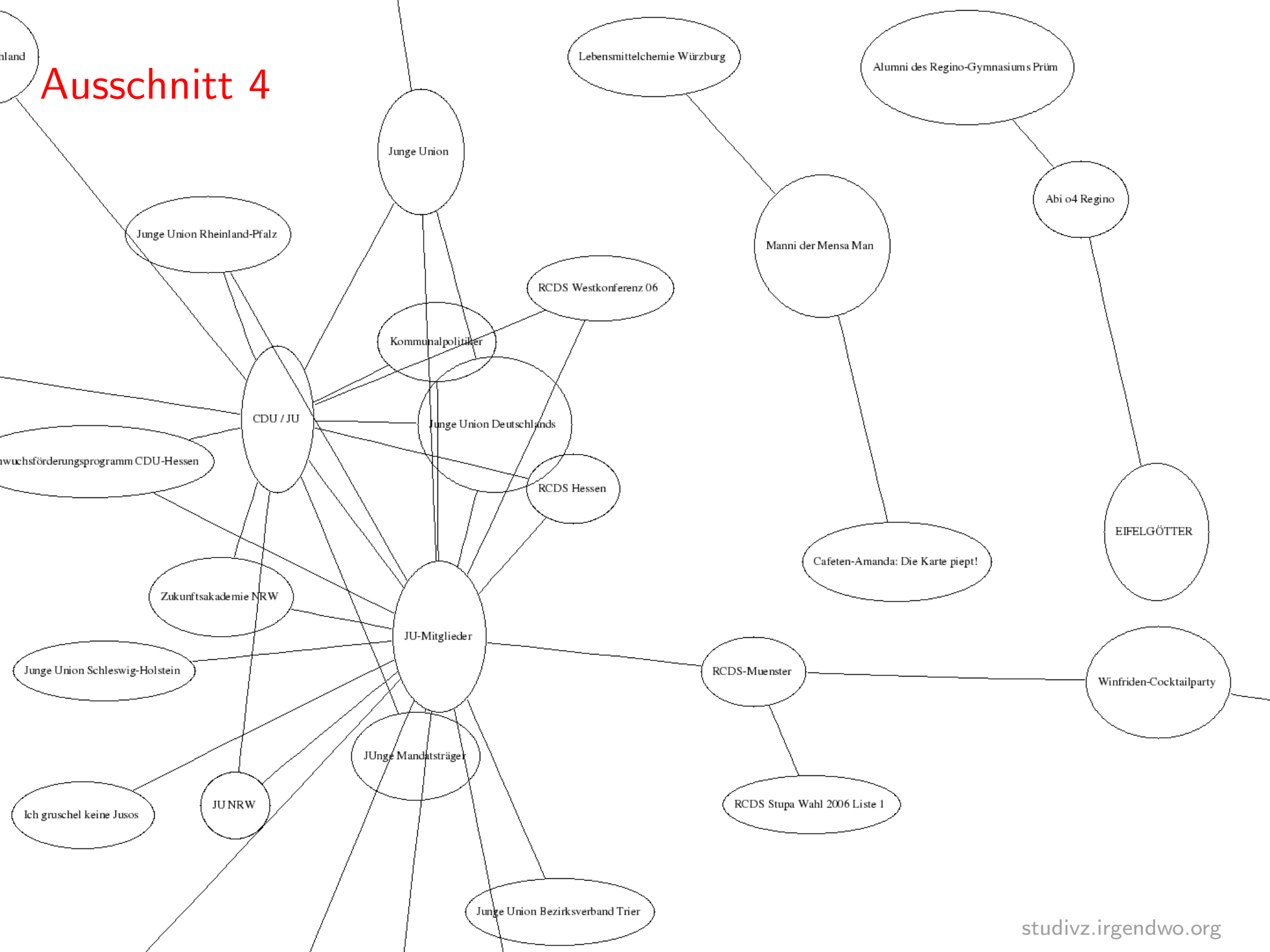

Ausschnitt 1
Karadeniz Firtinasi
KaraDeNiZ SHOW TIME
Ankara'lilar
Üç Hilal "Adalet,Ahlak,Iman"
EKK - Evde kalmis kizlar
Marmara bölgesi buraya
ThE GrEaTeSt AnKaRa 06
Türkei - Politische Themen
Serefli Kochisarliler
Türk Egeliler Budan Buyurun :)
Turkish Party People
DUYGU BAHCESI(duygularinizi bu bahcede yesillendirin)
Said Nursi
Türkische Literatur
Kalbin dili olsa...
Miss Turkey
Efeler Diyari
Freunde der Nacht
Sag eller havaya...PIRLANTALAR buraya!!!
Islamische Gemeinschaft Milli Görüs (ICMG)
Turkish Law Students
elite/kanaken
Türk Sinemasi
Türkische Geschichte
Gruppe türkischer Studenten
Turkish Delight
Dünyada Türklük
Turkish Students Community
Turkish - European Business Pool
Turkey and Greece
Rafet el Roman Fan Community
Türkisch People
TC Türkisch Club
Turkish chickaz
mersin
Türk Üniversiteli Gençler
Türk Kahvesi (on TR)
Für eine vereinte Republik Türkei
OTURMA GURUP
Türkiye - Dünyanin en güzel ülkesi !
Rekordversuch: Türkische Studenten Ulm
Sila
Türk Sanat Müzigi
Alle Türkischen Studenten, die aus der Türkei kommen... Hin Hin!!
CEM YILMAZcilar
Eyyy Türk Gencligi
TD-Plattform
Kurtlar Vadisi
Yesilcam
Türkische Küche
Atatürk Kayseri Gençligi
ultrAslan
Nami büyük Tokatlilar
Kapadokyalilar
Türkei Europameister 2008 in der Schweiz!!!
68liler
Türkiye
GALATASARAY
Mannheim Türk Ögrenciler
Türkische Studenten/Studentinnen deutschlandweit
Türkische Studenten in München
Istanbul
Aachen turkish students
Völkermord an den Türken
7'den 77'ye - Barış Manço
"Huysuz Virjin" ist einfach witzig
Mythos eines Völkermords - Die Armenierfrage
Aksaray
Türkisch Rock
Türk Halk Oyunlari (Folklor)
TR RAP UNDERGROUND
Türkische Musik
Türkische Studenten aus Münster
studivz.irgendwo.org

Ausschnitt 2

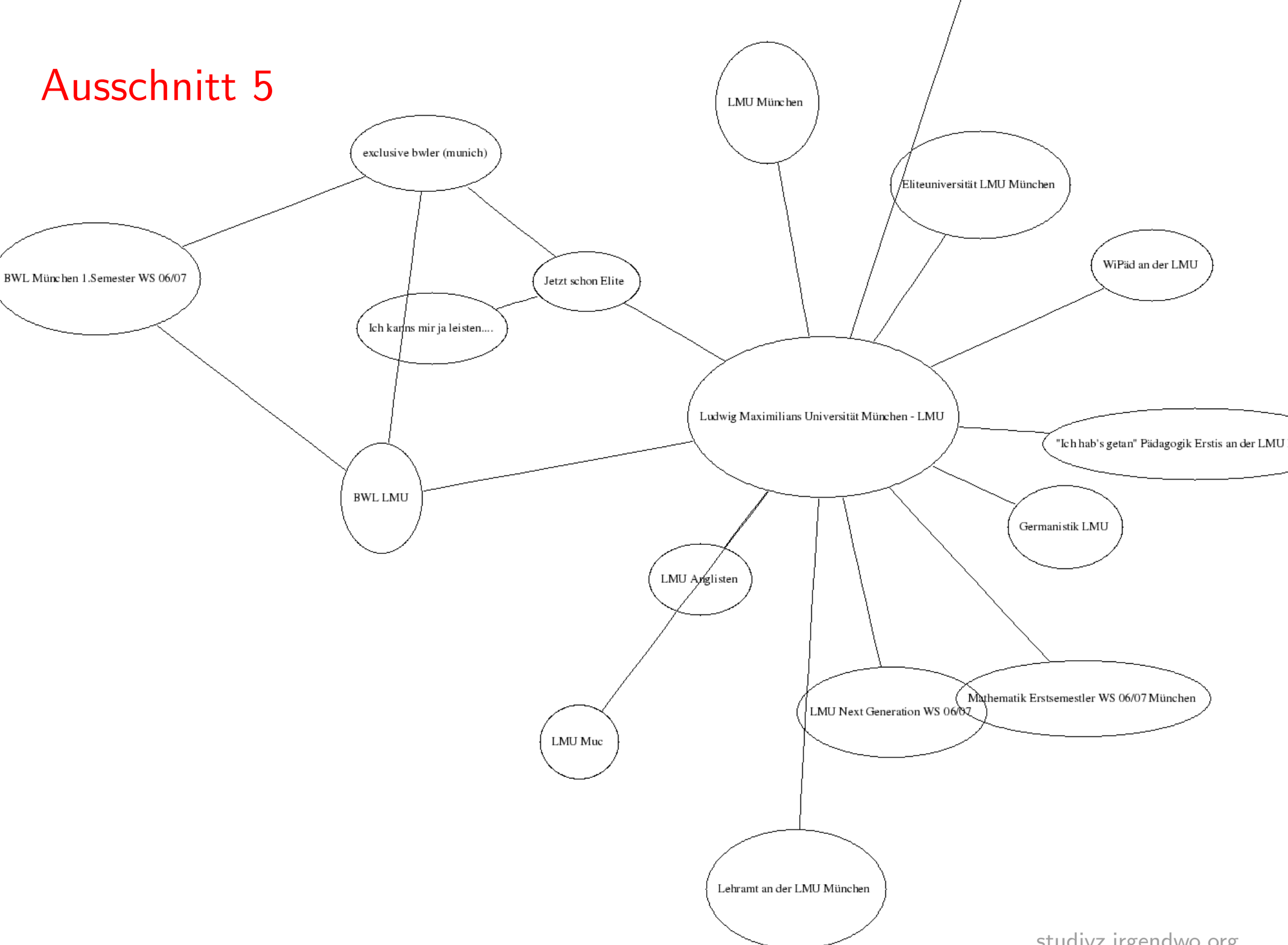
Ausschnitt 3



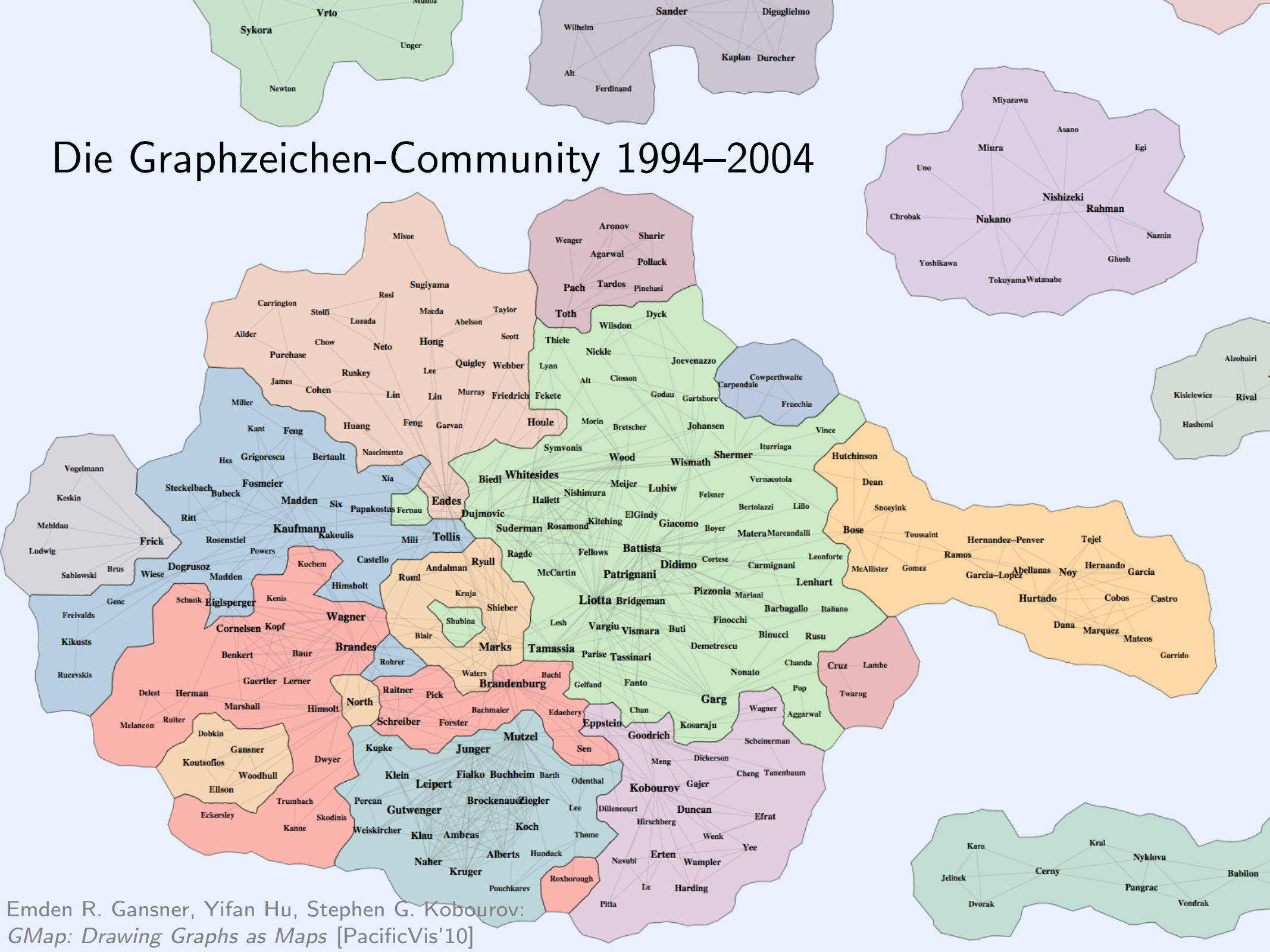
Ausschnitt 4



Ausschnitt 5

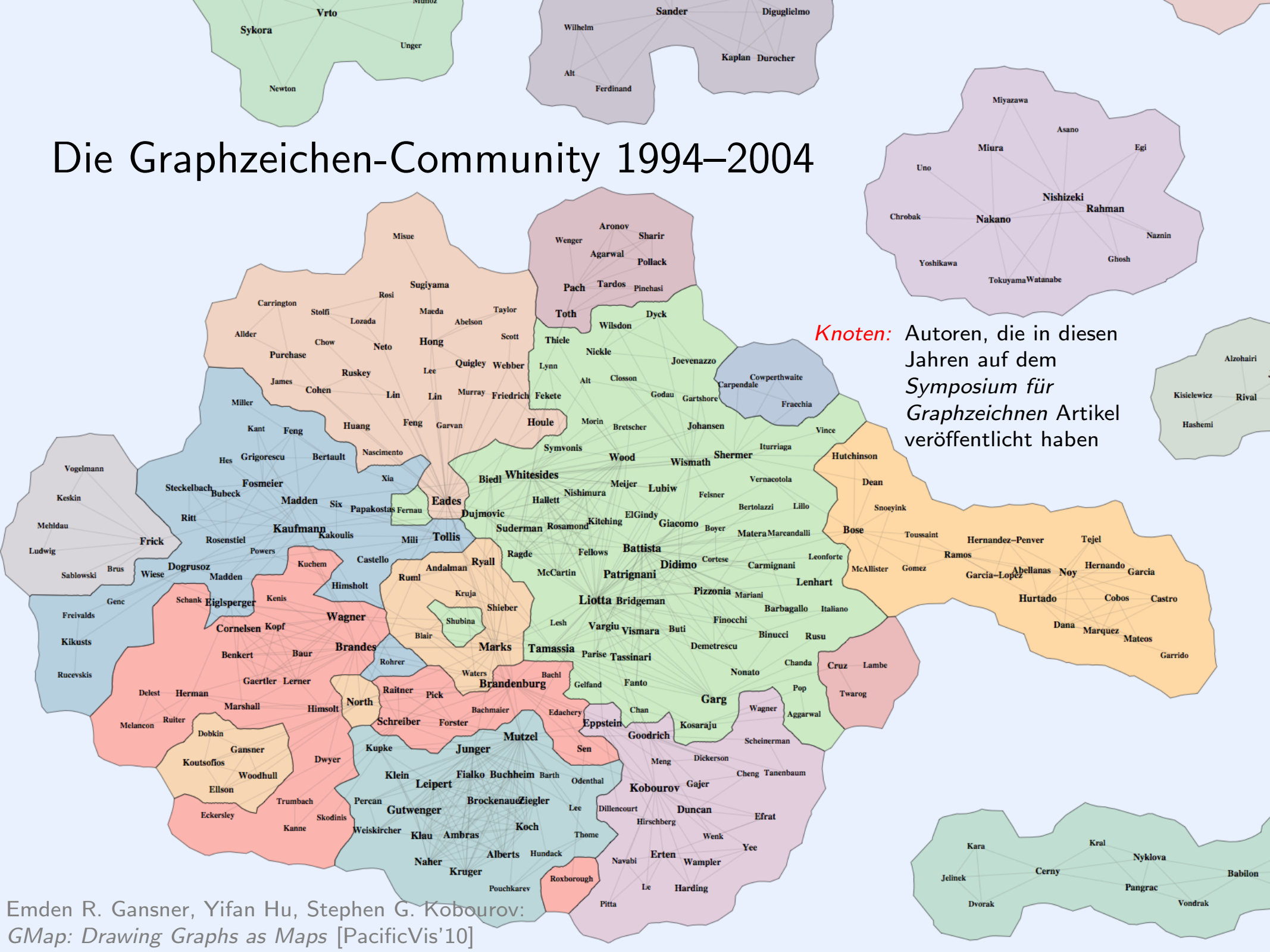


Die Graphzeichen-Community 1994–2004



Emden R. Gansner, Yifan Hu, Stephen G. Kobourov:
GMap: Drawing Graphs as Maps [PacificVis'10]

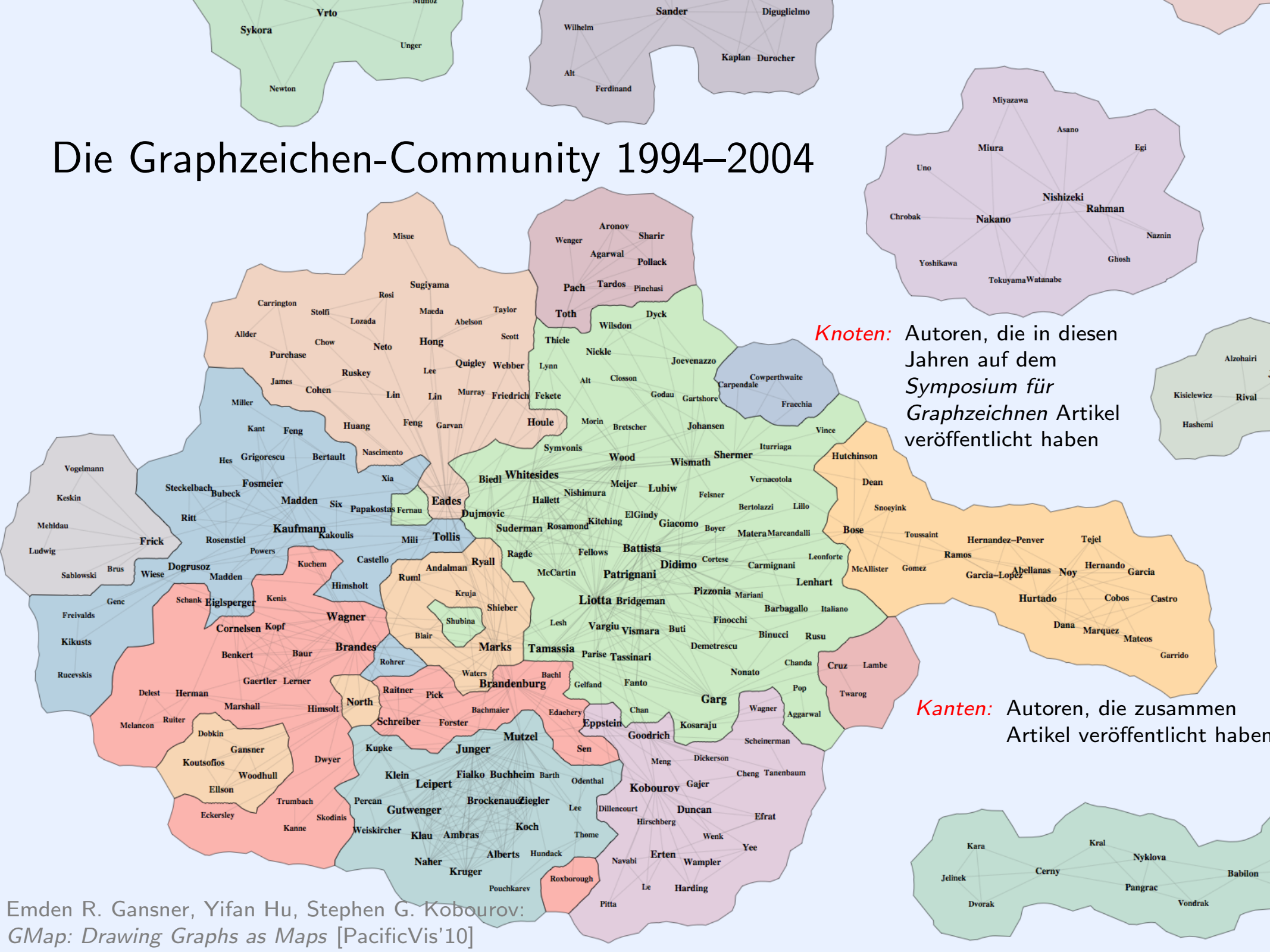
Die Graphzeichen-Community 1994–2004



Knoten: Autoren, die in diesen Jahren auf dem *Symposium für Graphzeichnen* Artikel veröffentlicht haben

Emden R. Gansner, Yifan Hu, Stephen G. Kobourov:
GMap: Drawing Graphs as Maps [PacificVis'10]

Die Graphzeichen-Community 1994–2004

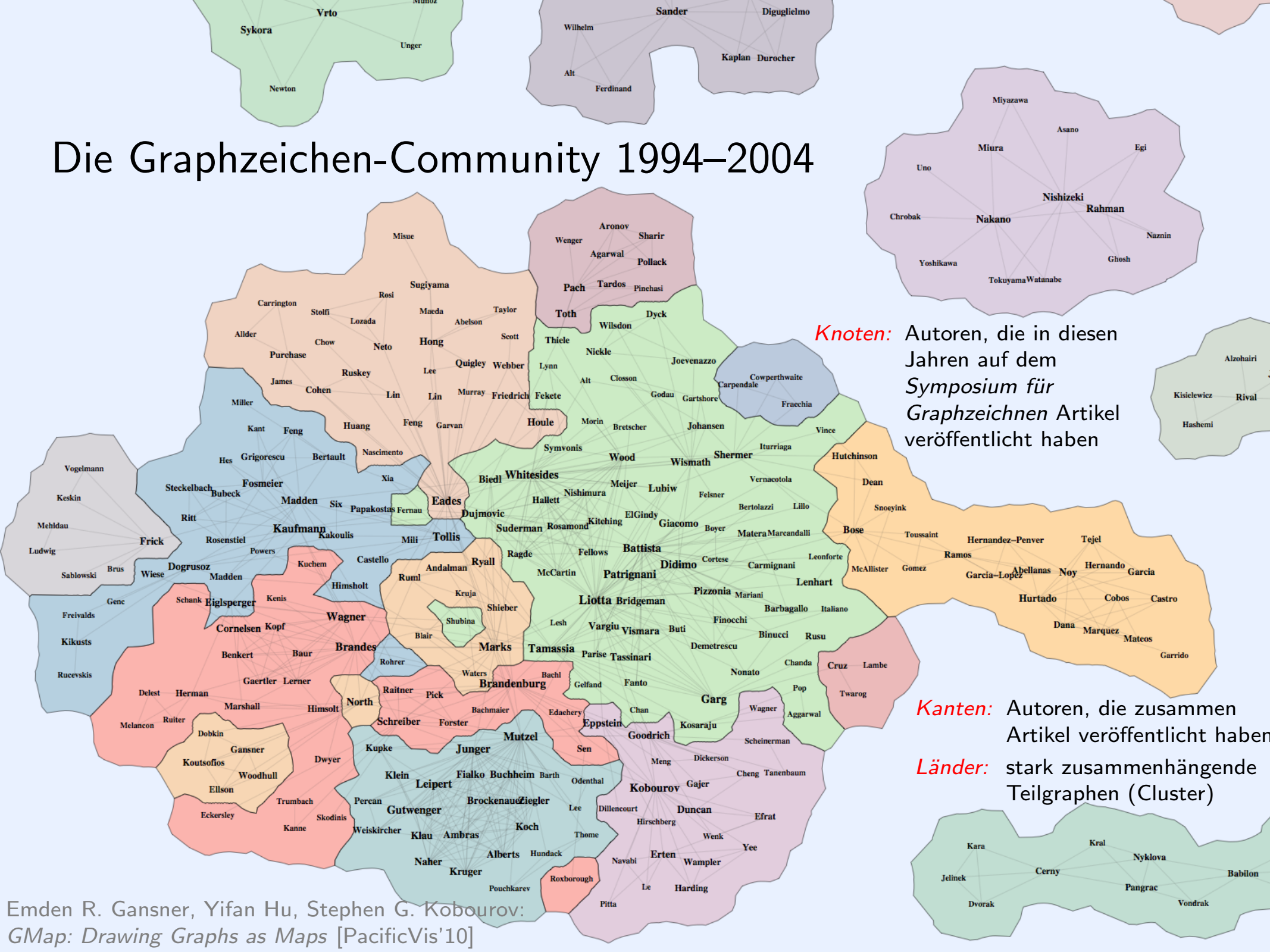


Knoten: Autoren, die in diesen Jahren auf dem *Symposium für Graphzeichnen* Artikel veröffentlicht haben

Kanten: Autoren, die zusammen Artikel veröffentlicht haben

Emden R. Gansner, Yifan Hu, Stephen G. Kobourov:
GMap: Drawing Graphs as Maps [PacificVis'10]

Die Graphzeichen-Community 1994–2004



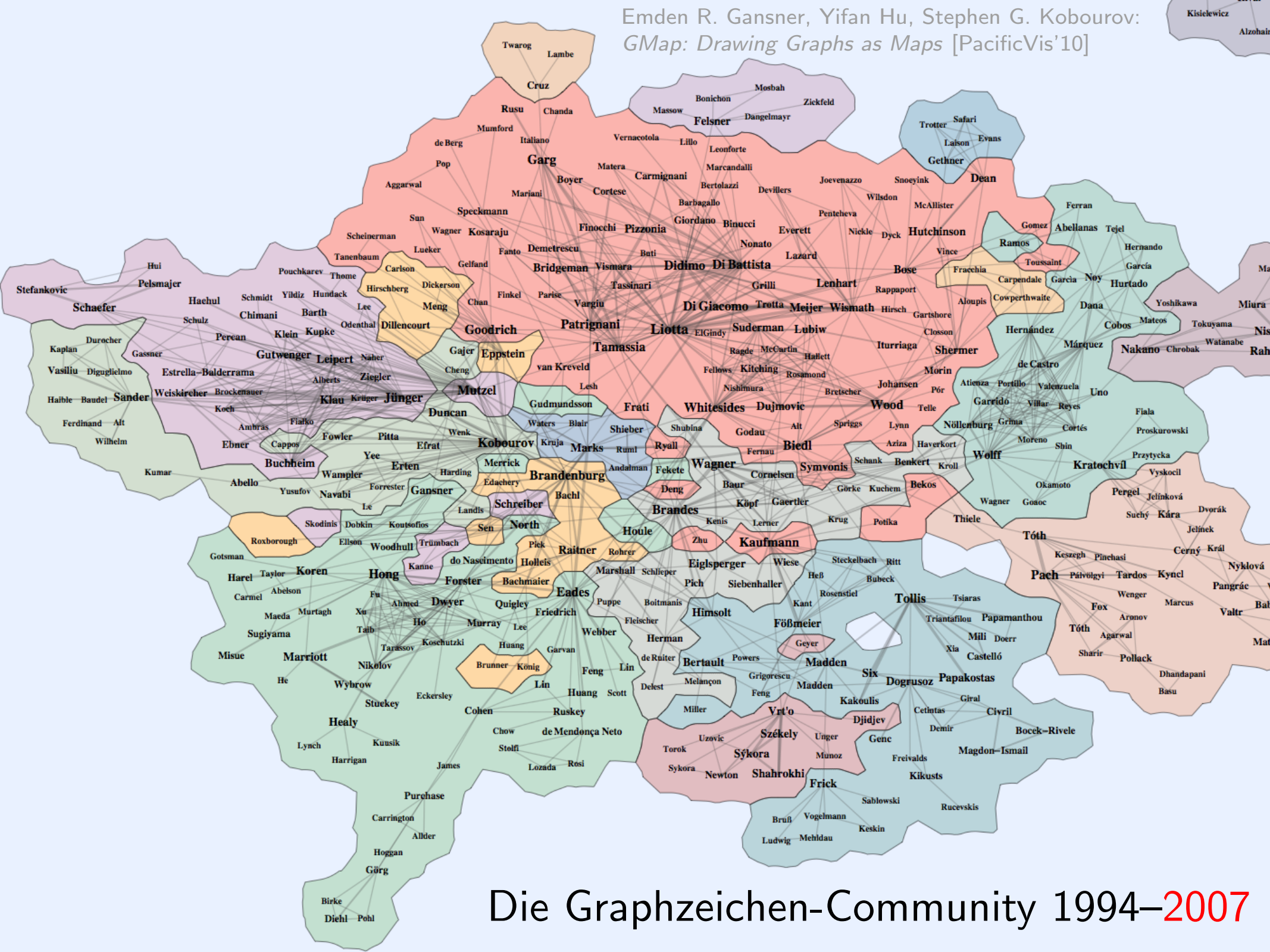
Knoten: Autoren, die in diesen Jahren auf dem *Symposium für Graphzeichnen* Artikel veröffentlicht haben

Kanten: Autoren, die zusammen Artikel veröffentlicht haben

Länder: stark zusammenhängende Teilgraphen (Cluster)

Emden R. Gansner, Yifan Hu, Stephen G. Kobourov:
GMap: Drawing Graphs as Maps [PacificVis'10]

Emden R. Gansner, Yifan Hu, Stephen G. Kobourov:
GMap: Drawing Graphs as Maps [PacificVis'10]



Die Graphzeichen-Community 1994–2007

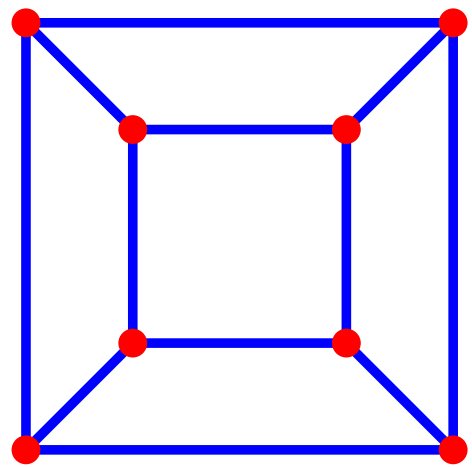
F: Was ist ein Graph?

F: Was ist ein Graph?

A_1: Ein (ungerichteter) Graph ist ein Paar (V, E)

F: Was ist ein Graph?

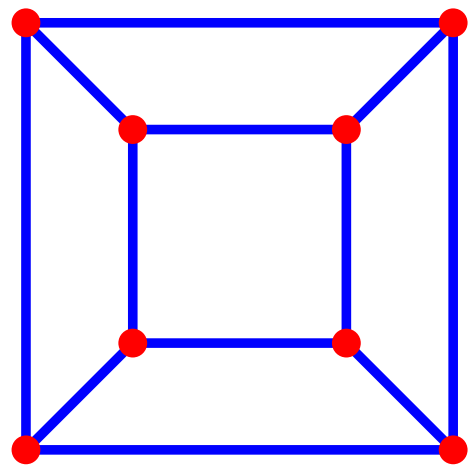
A_1: Ein (ungerichteter) Graph ist ein Paar (V, E)



F: Was ist ein Graph?

A_1: Ein (ungerichteter) Graph ist ein Paar (V, E), wobei

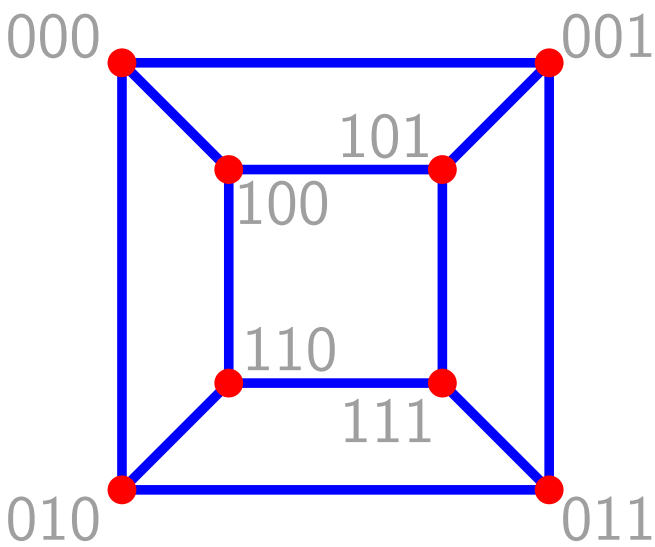
- V *Knotenmenge* und
- $E \subseteq \binom{V}{2} = \{\{u, v\} \subseteq V \mid u \neq v\}$ *Kantenmenge*.



F: Was ist ein Graph?

A_1: 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*.

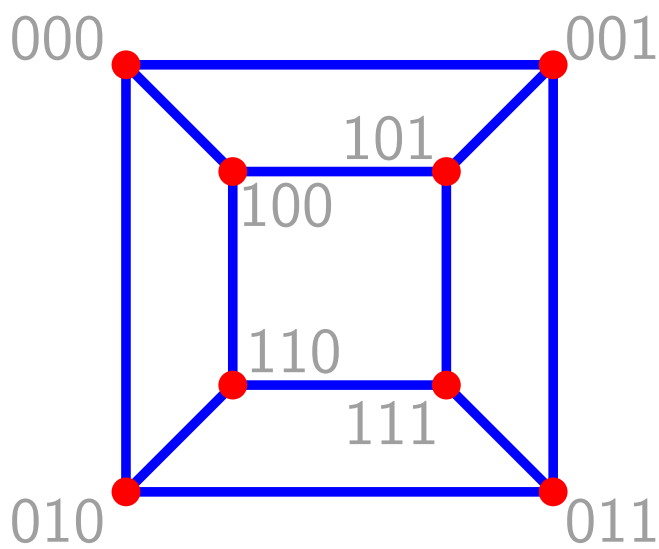
$V = \{000, 001, \ldots, 111\}$
$\{u, v\} \in E \quad \Leftrightarrow$?

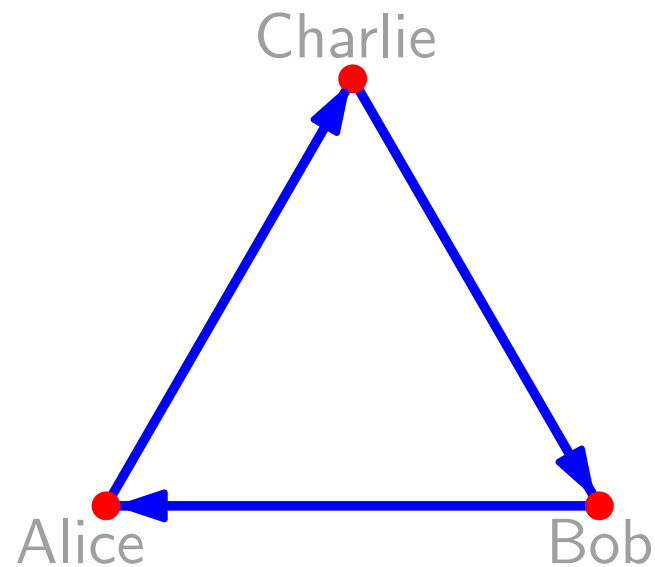


F: Was ist ein Graph?

A_1: 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*.

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

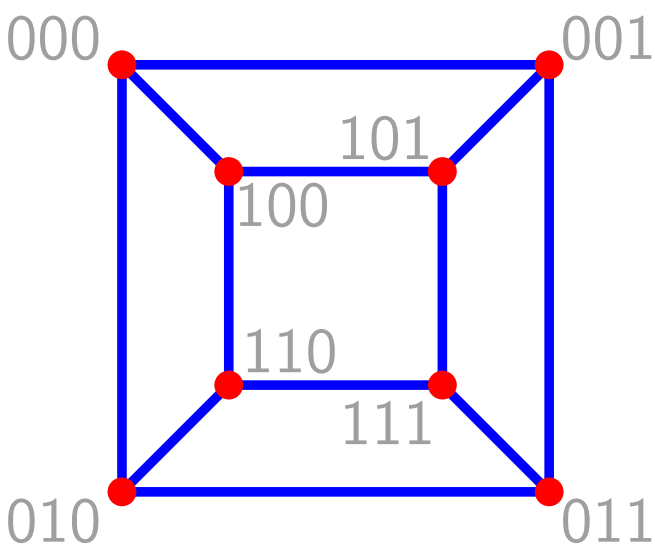


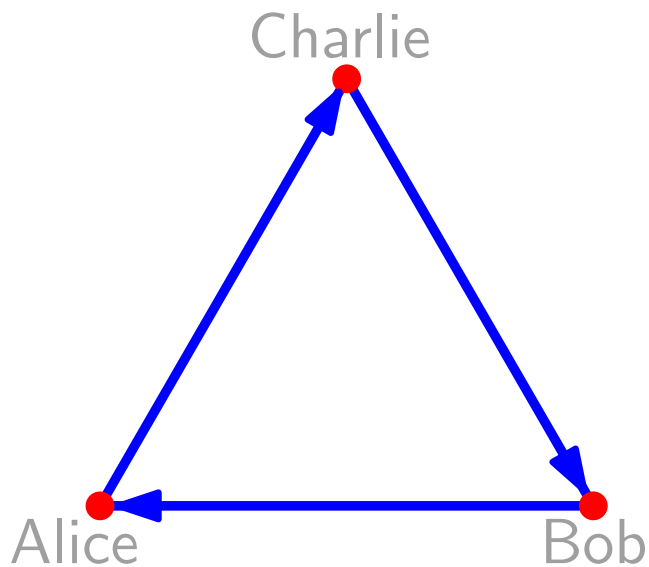


F: Was ist ein Graph?

A_1: 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*.

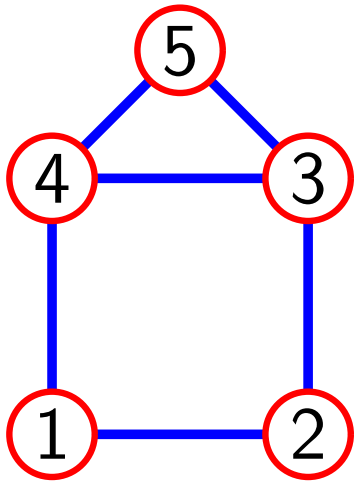
$V = \{000, 001, \ldots, 111\}$
$\{u, v\} \in E \quad \Leftrightarrow \quad ?$





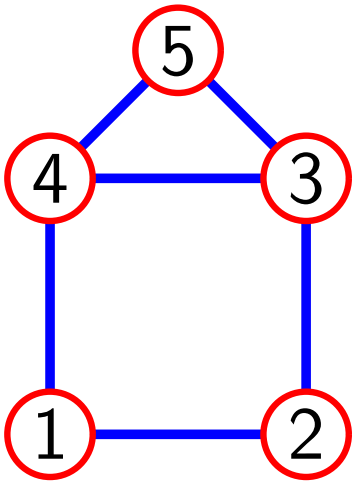
A_2: Ein *gerichteter* Graph ist ein Paar (V, E), wobei
- V *Knotenmenge* und
- $E \subseteq V \times V = \{(u, v) \mid u, v \in V\}$ *Kantenmenge*.

F: Wie repräsentiere ich einen Graphen?

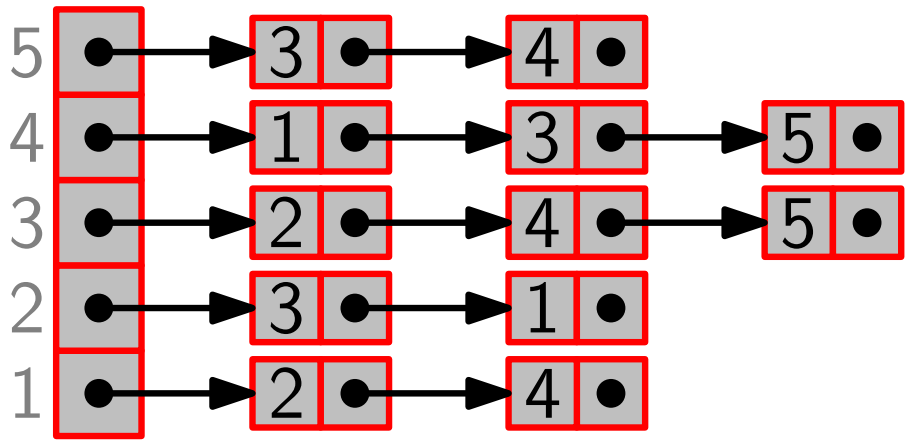


ungerichteter Graph

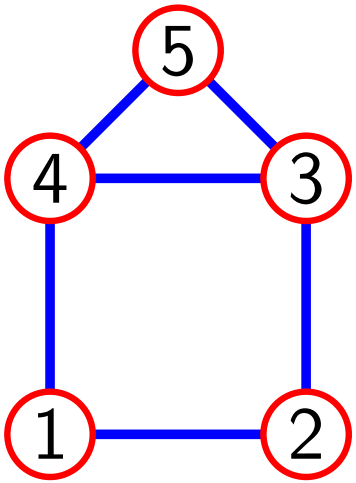
F: Wie repräsentiere ich einen Graphen?



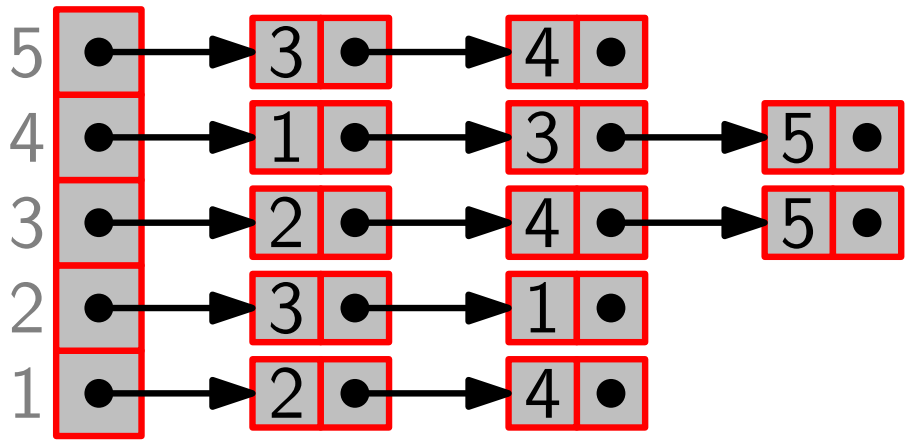
ungerichteter Graph



F: Wie repräsentiere ich einen Graphen?



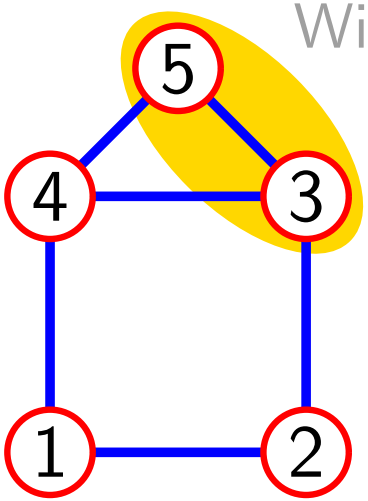
ungerichteter Graph



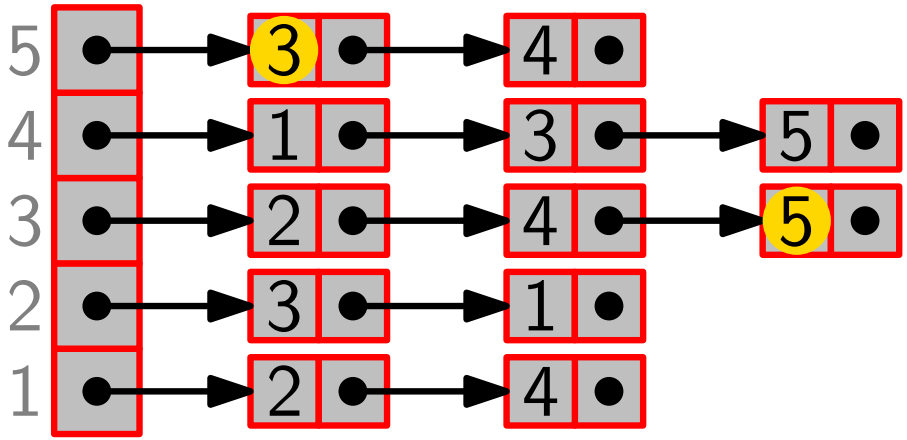
Adjazenzlisten

F: Wie repräsentiere ich einen Graphen?

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

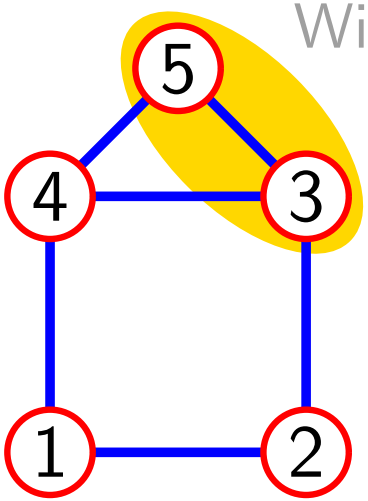


ungerichteter Graph



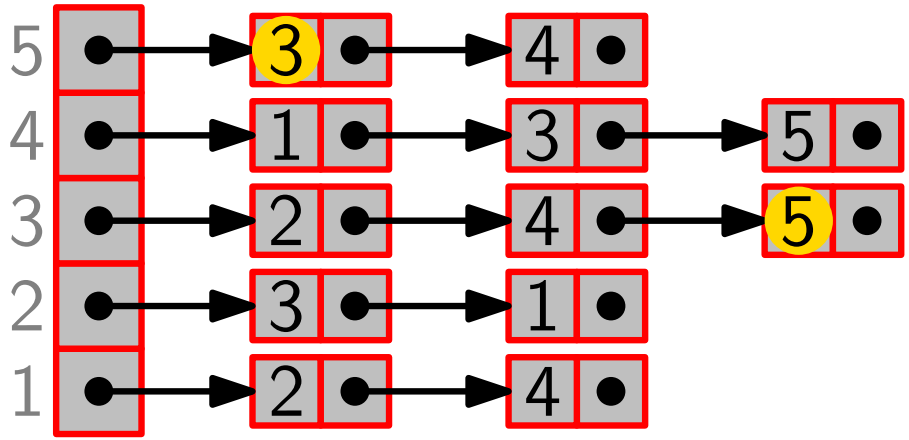
Adjazenzlisten

F: Wie repräsentiere ich einen Graphen?

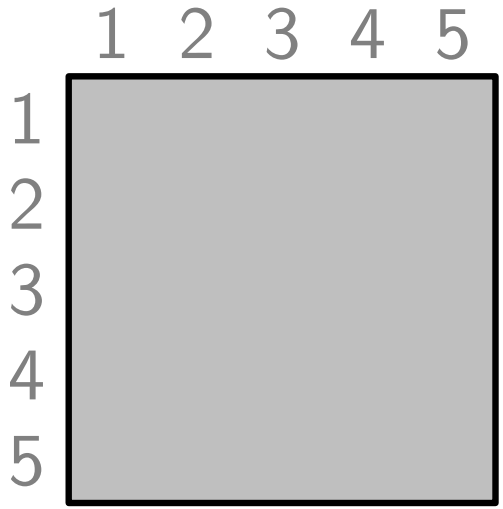


ungerichteter Graph

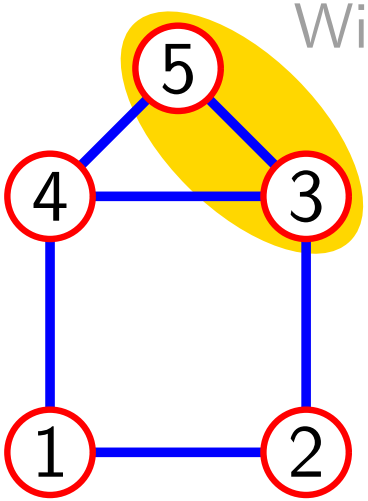
Wir sagen: Knoten 3 und 5 sind *adjazent*.



Adjazenzlisten

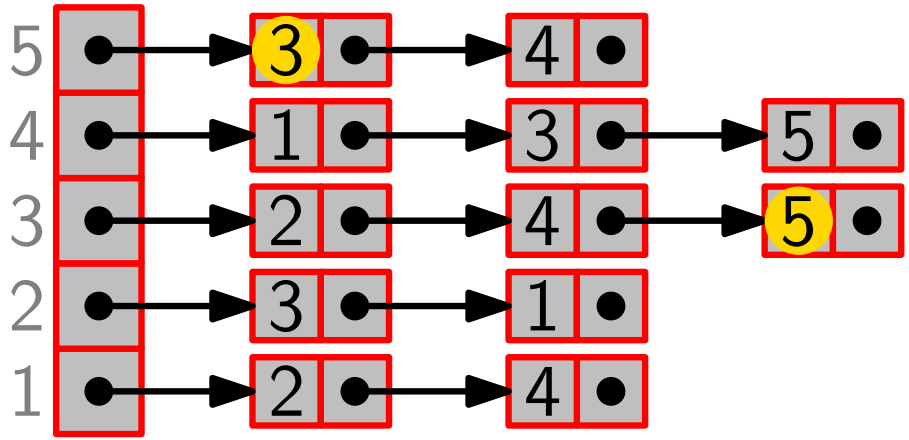


F: Wie repräsentiere ich einen Graphen?



ungerichteter Graph

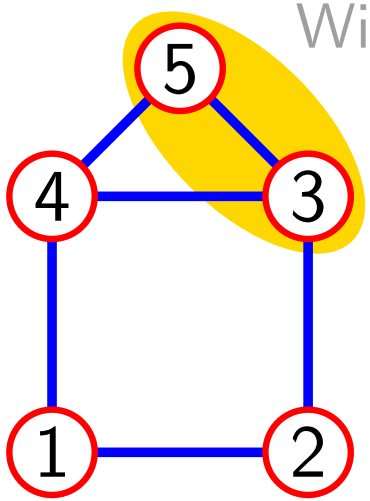
Wir sagen: Knoten 3 und 5 sind *adjazent*.



Adjazenzlisten

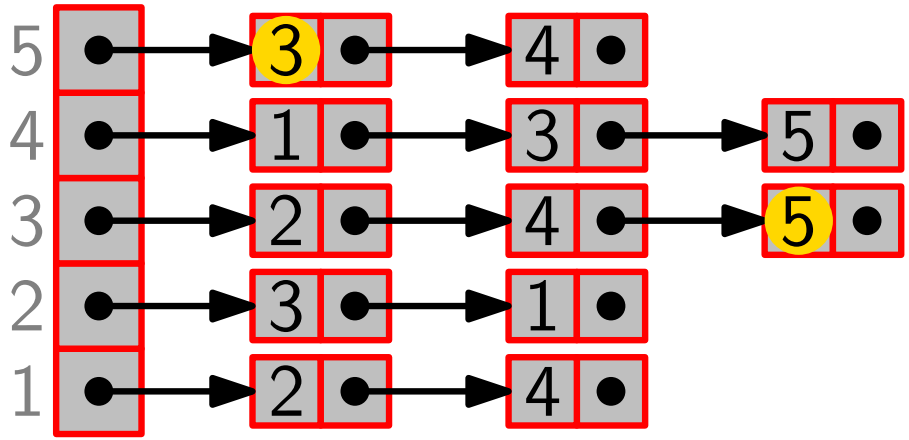
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	1
5	0	0	1	1	0

F: Wie repräsentiere ich einen Graphen?



ungerichteter Graph

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



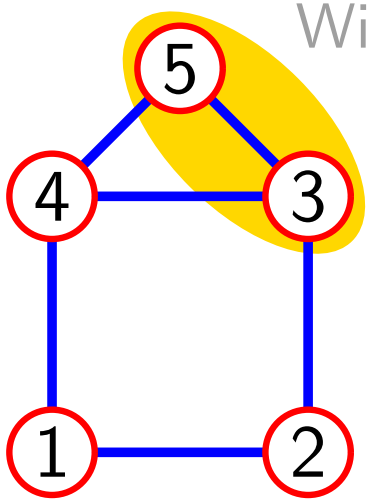
Adjazenzlisten

	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

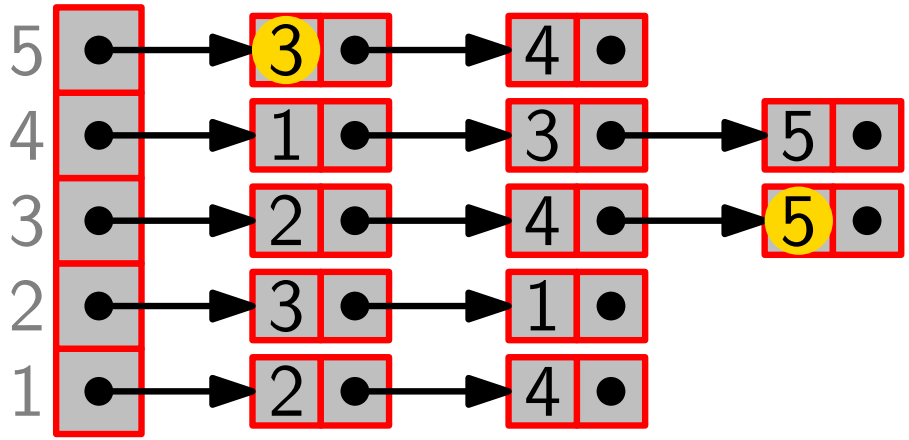
Adjazenzmatrix

F: Wie repräsentiere ich einen Graphen?

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



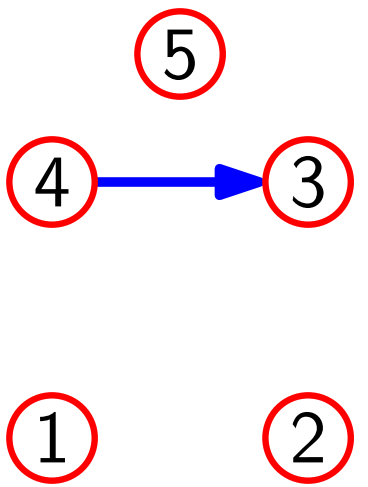
ungerichteter Graph



Adjazenzlisten

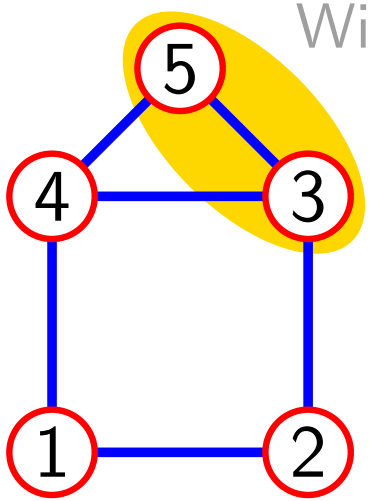
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	1
5	0	0	1	1	0

Adjazenzmatrix

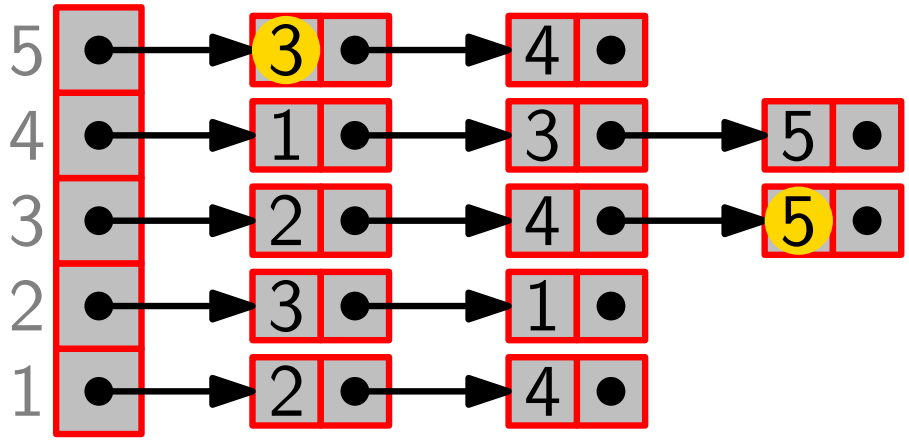


F: Wie repräsentiere ich einen Graphen?

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



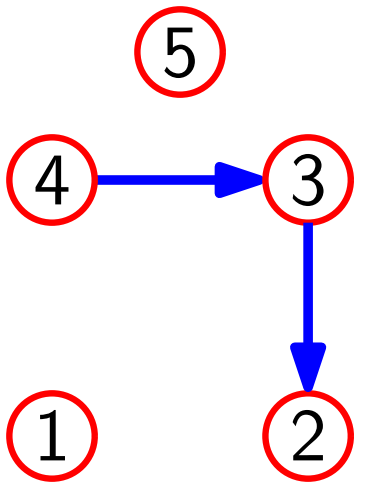
ungerichteter Graph



Adjazenzlisten

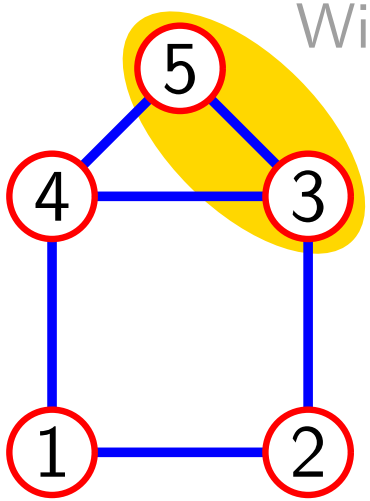
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	1
5	0	0	1	1	0

Adjazenzmatrix

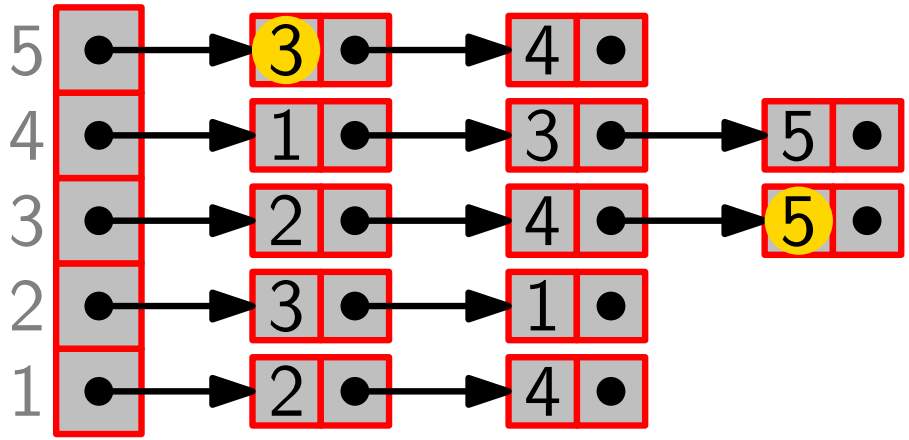


F: Wie repräsentiere ich einen Graphen?

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



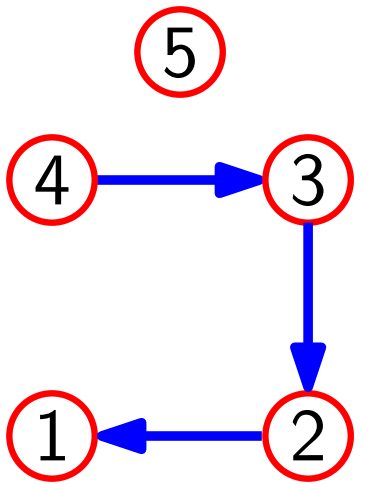
ungerichteter Graph



Adjazenzlisten

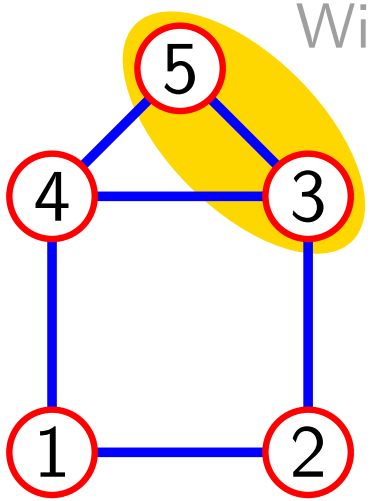
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	1
5	0	0	1	1	0

Adjazenzmatrix

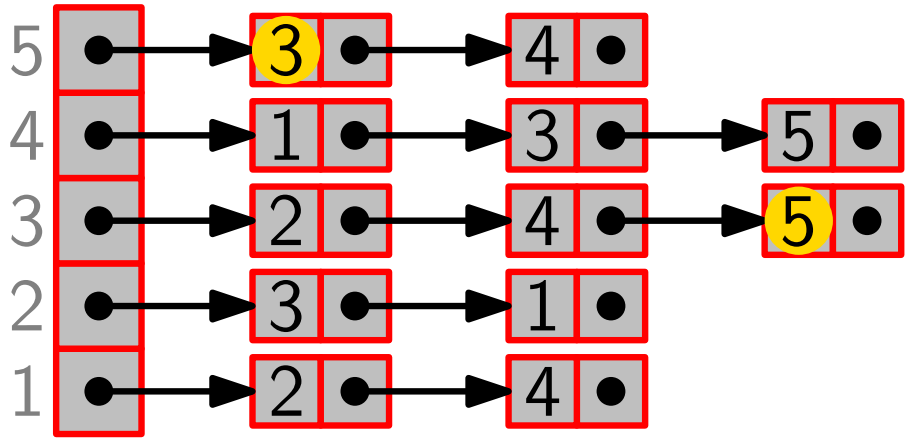


F: Wie repräsentiere ich einen Graphen?

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



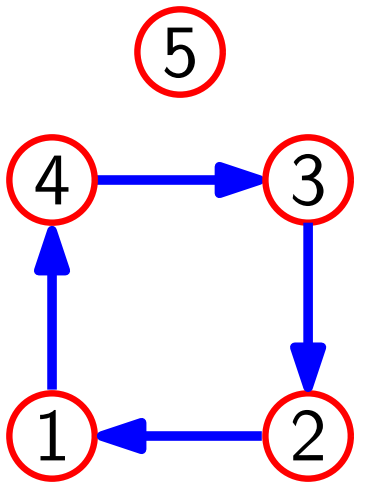
ungerichteter Graph



Adjazenzlisten

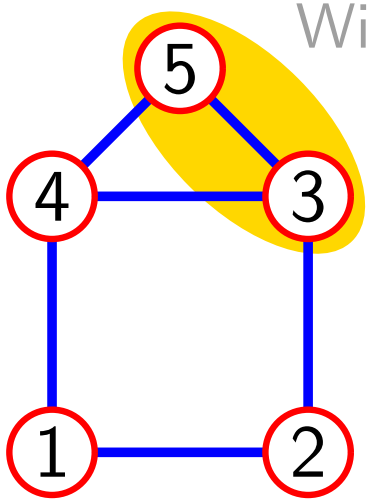
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	1
5	0	0	1	1	0

Adjazenzmatrix

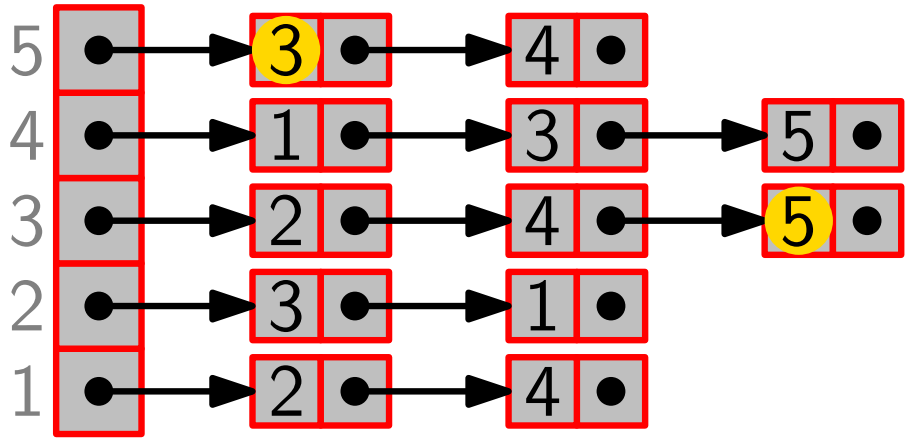


F: Wie repräsentiere ich einen Graphen?

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



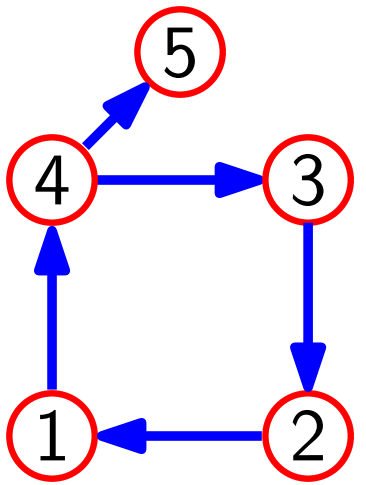
ungerichteter Graph



Adjazenzlisten

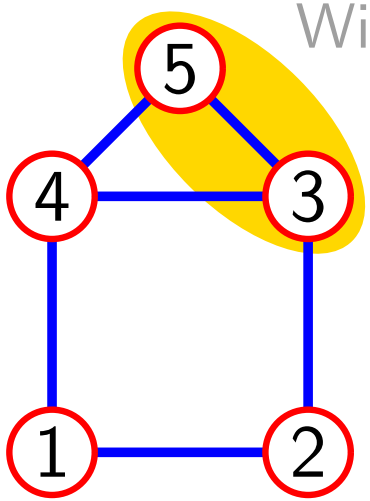
	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	0
3	0	1	0	1	1
4	1	0	1	0	1
5	0	0	1	1	0

Adjazenzmatrix

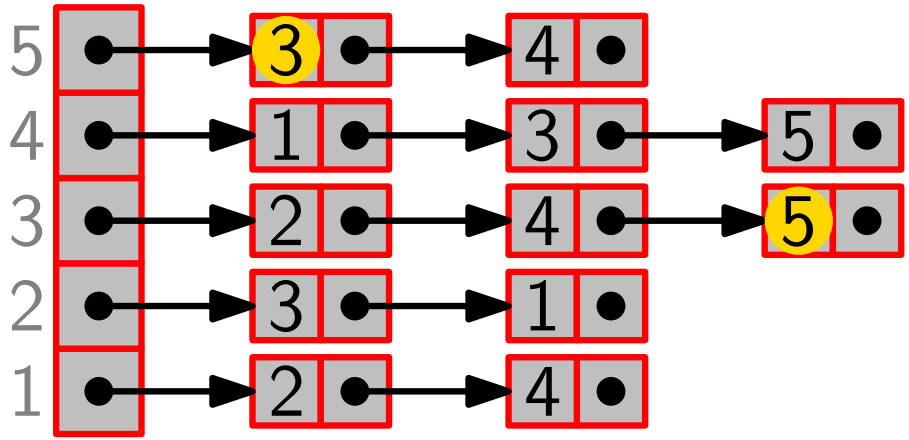


F: Wie repräsentiere ich einen Graphen?

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



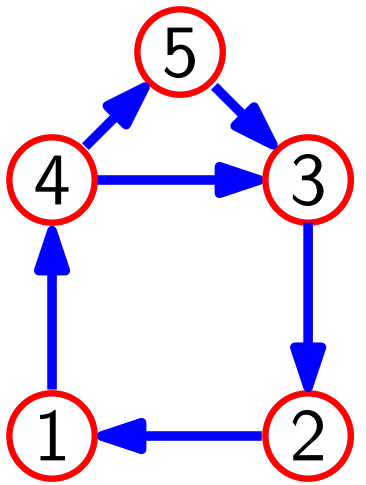
ungerichteter Graph



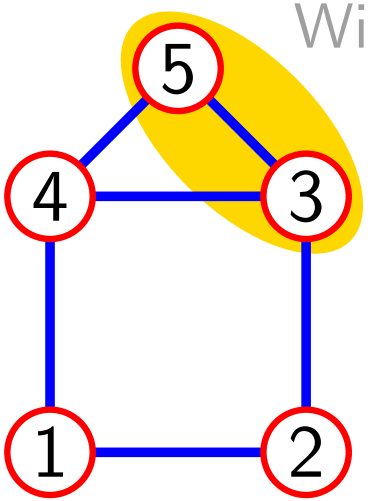
Adjazenzlisten

	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

Adjazenzmatrix

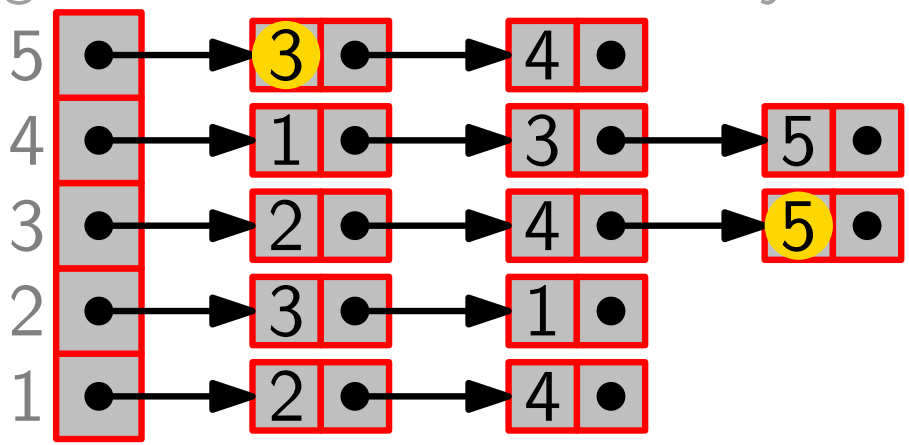


F: Wie repräsentiere ich einen Graphen?



ungerichteter Graph

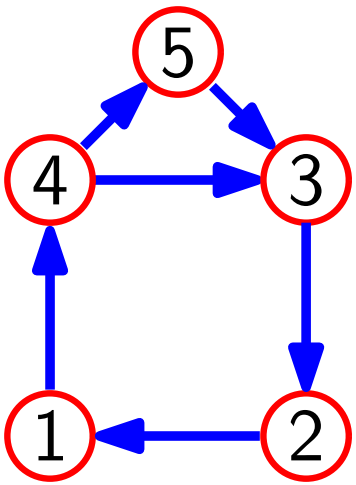
Wir sagen: Knoten 3 und 5 sind *adjazent*.



Adjazenzlisten

	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

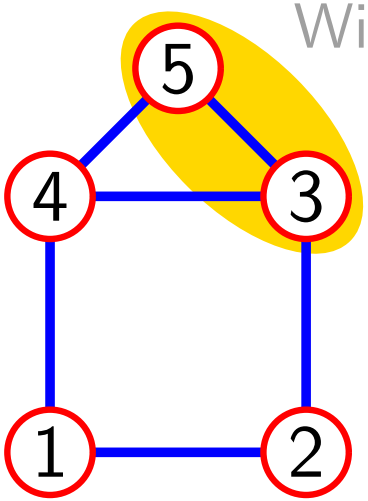
Adjazenzmatrix



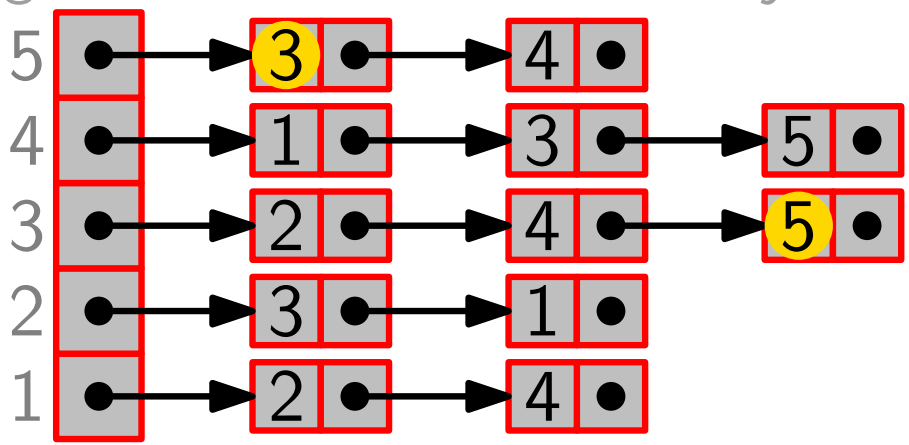
gerichteter Graph

F: Wie repräsentiere ich einen Graphen?

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



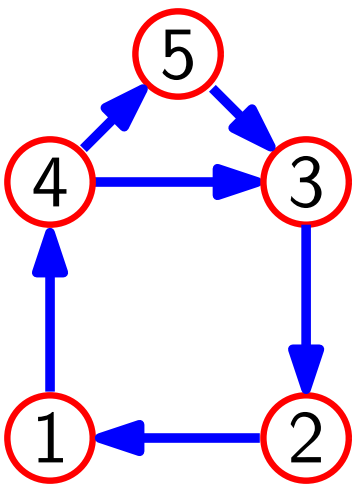
ungerichteter Graph



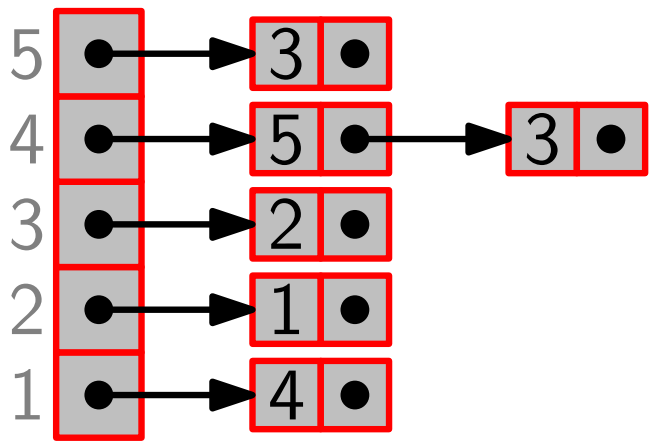
Adjazenzlisten

	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

Adjazenzmatrix



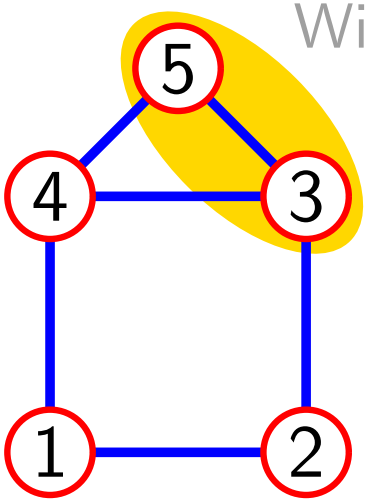
gerichteter Graph



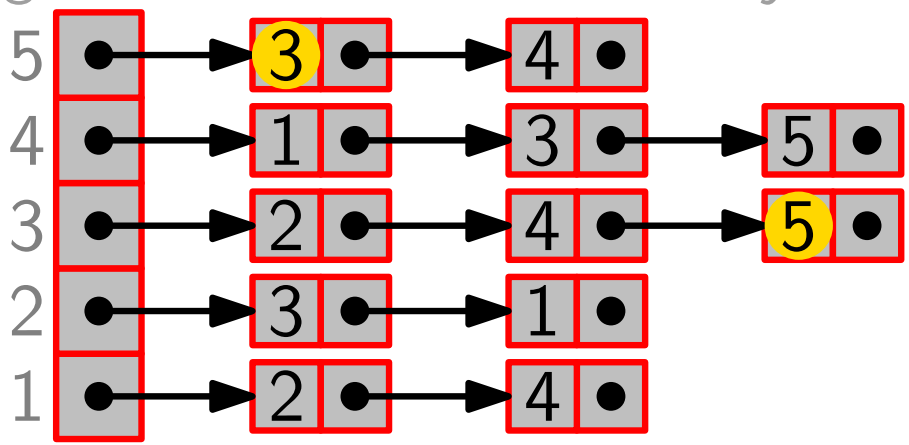
$\text{Adj}[i] = \{j \in V \mid (i,j) \in E\}$

F: Wie repräsentiere ich einen Graphen?

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



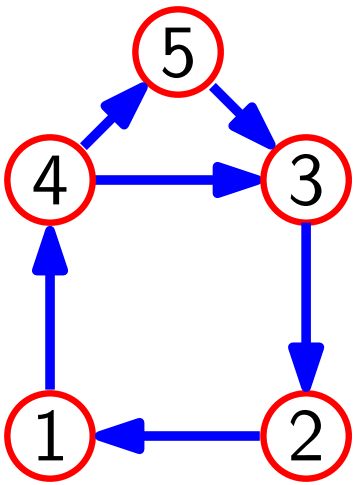
ungerichteter Graph



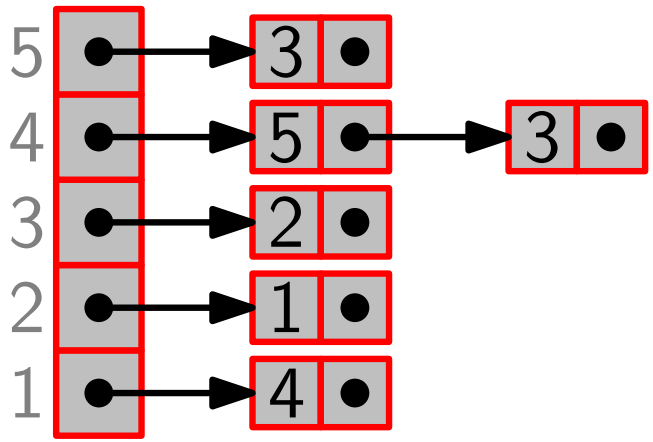
Adjazenzlisten

	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

Adjazenzmatrix



gerichteter Graph



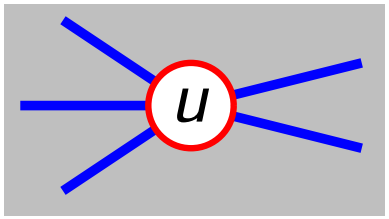
$\mathrm{Adj}[i] = \{j \in V \mid (i,j) \in E\}$

	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

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

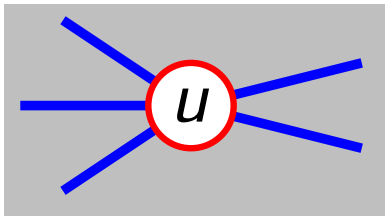
Grad eines Knotens

Def.



Grad eines Knotens

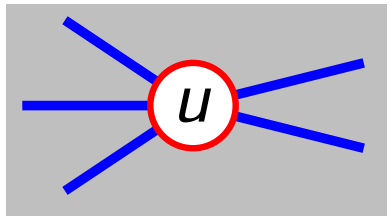
Def.

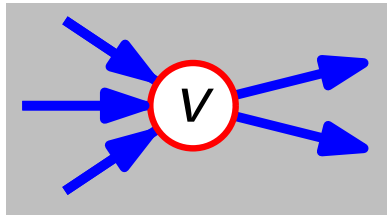


$$\deg(u) = |\mathrm{Adj}[u]|$$

Grad eines Knotens

Def.

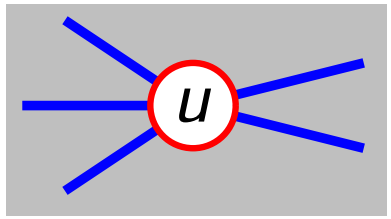




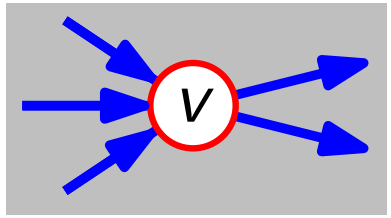
$$\deg(u) = |\mathrm{Adj}[u]|$$

Grad eines Knotens

Def.



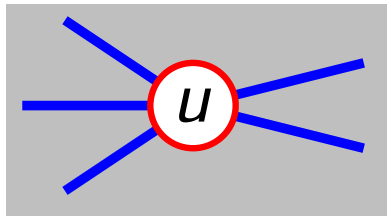
$$\deg(u) = |\mathrm{Adj}[u]|$$



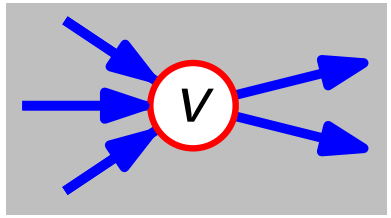
$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$

Grad eines Knotens

Def.



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

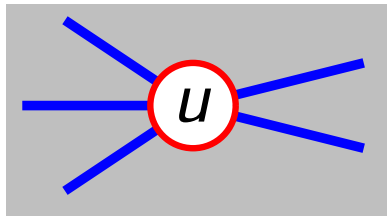


$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$

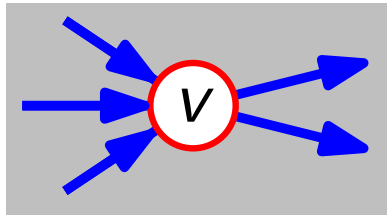
$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

Grad eines Knotens

Def.



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

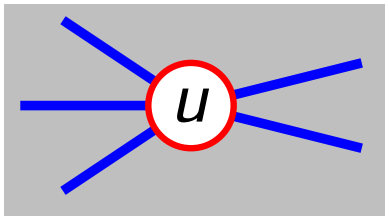


$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$
$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

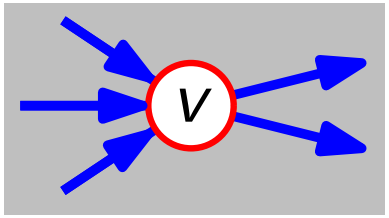
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $=$ 

Grad eines Knotens

Def.



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

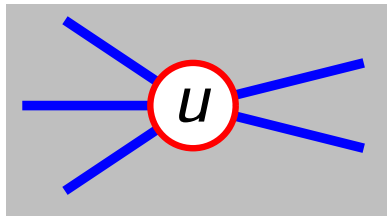


$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$
$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

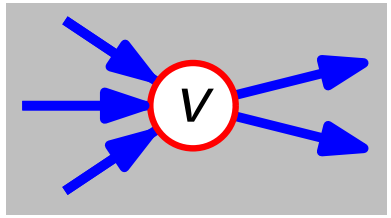
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Grad eines Knotens

Def.



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



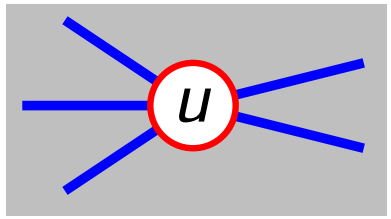
$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$
$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

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

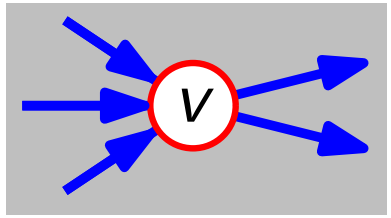
Beweis. Technik des *zweifachen Abzählens:*

Grad eines Knotens

Def.



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



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

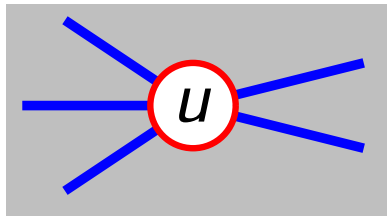
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Beweis. Technik des *zweifachen Abzählens:*

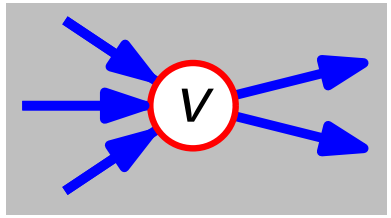
Zähle alle Knoten-Kanten-Inzidenzen.

Grad eines Knotens

Def.



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



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

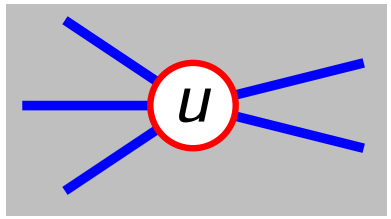
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Beweis. Technik des *zweifachen Abzählens:*

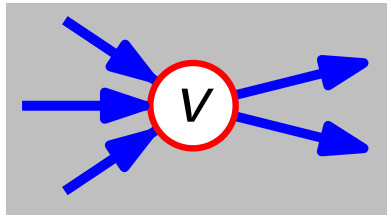
Zähle alle Knoten-Kanten-Inzidenzen.

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 $= 2 \cdot |E|$.

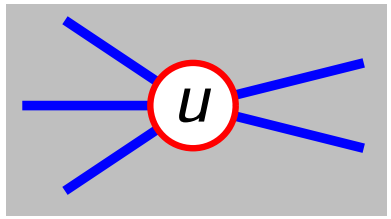
Beweis. Technik des *zweifachen Abzählens:*

Zähle alle Knoten-Kanten-Inzidenzen.

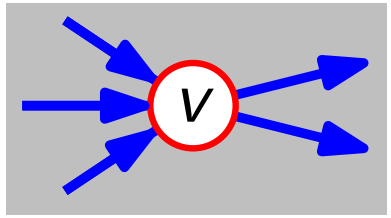
Eine Kante ist *inzident* zu ihren Endknoten.

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:*

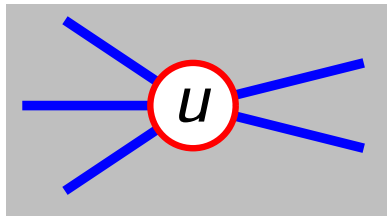
Zähle alle Knoten-Kanten-Inzidenzen.

Eine Kante ist *inzident* zu ihren Endknoten.

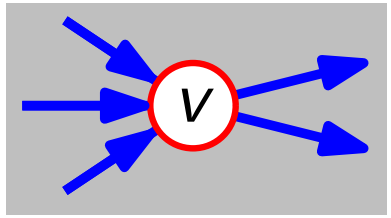
Ein Knoten ist *inzident* zu allen Kanten, deren Endknoten er ist.

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.

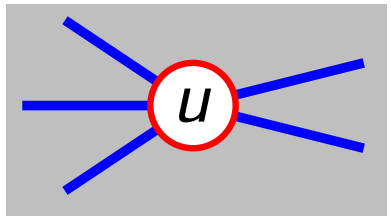
Ein Knoten ist *inzident* zu allen Kanten, deren Endknoten er ist.

Aus Sicht der Knoten:

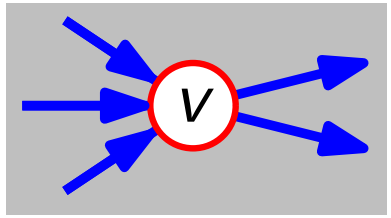
Aus Sicht der Kanten:

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.

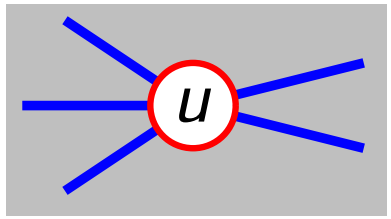
Ein Knoten ist *inzident* zu allen Kanten, deren Endknoten er ist.

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

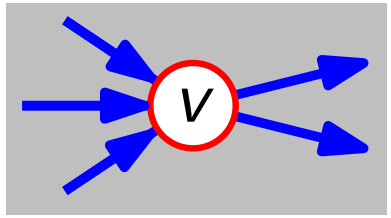
Aus Sicht der Kanten:

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.

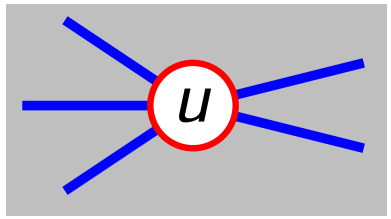
Ein Knoten ist *inzident* zu allen Kanten, deren Endknoten er ist.

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

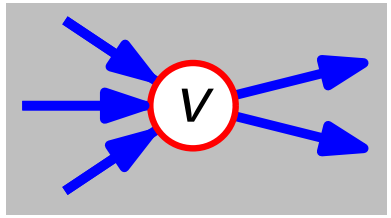
Aus Sicht der Kanten: $2 \cdot |E|$

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.

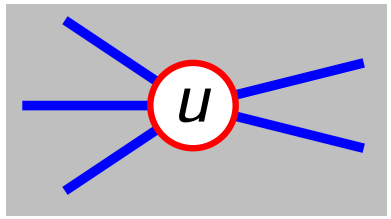
Ein Knoten ist *inzident* zu allen Kanten, deren Endknoten er ist.

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

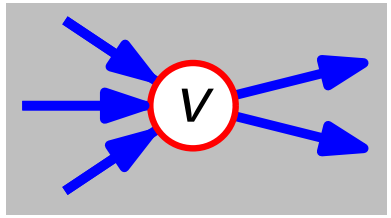
Aus Sicht der Kanten: $2 \cdot |E|$ also gleich!

Grad eines Knotens

Def.



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

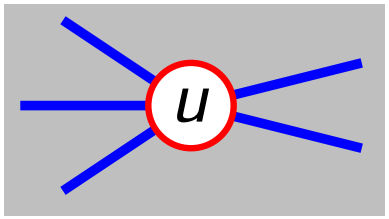


$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$
$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

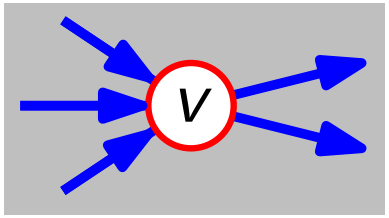
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Grad eines Knotens

Def.



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



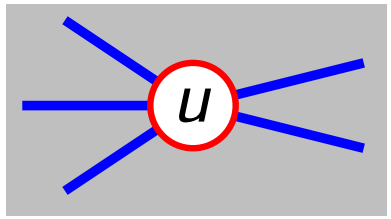
$$\mathrm{outdeg}(v) = |\mathrm{Adj}[v]|$$
$$\mathrm{indeg}(v) = |\{u \in V : (u, v) \in E\}|$$

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

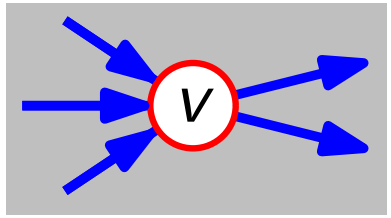
Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

Grad eines Knotens

Def.



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



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

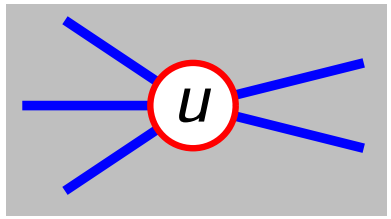
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

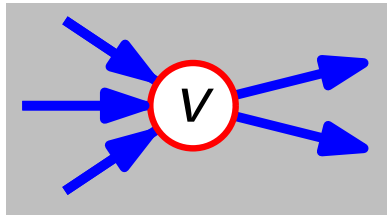
Beweis. $2 \cdot |E| = \sum_{v \in V} \deg v$

Grad eines Knotens

Def.



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



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

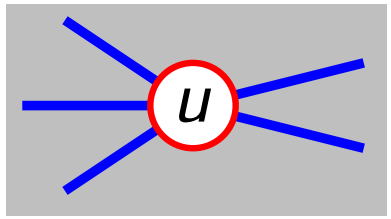
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

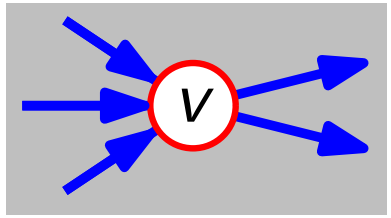
Beweis. $2 \cdot |E| = \sum_{v \in V} \deg v = \sum_{v \in V_{\text{ger}}} \deg v + \sum_{v \in V_{\text{ung}}} \deg v$

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 $= 2 \cdot |E|$.

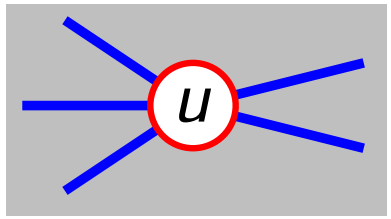
Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

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

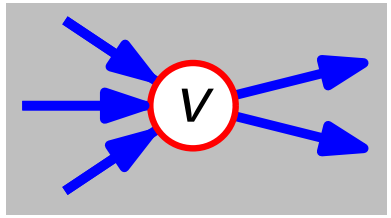
gerade!

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 $= 2 \cdot |E|$.

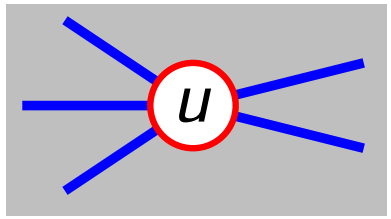
Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

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

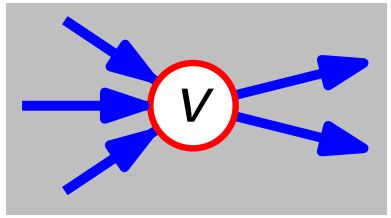
gerade! *gerade!*

Grad eines Knotens

Def.



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



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

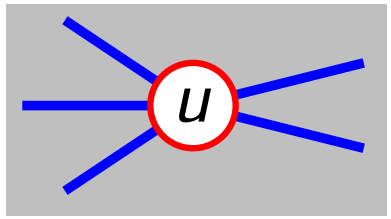
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

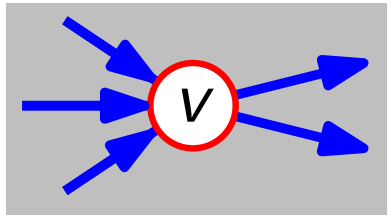
Beweis. $\underbrace{2 \cdot |E|}_{\textit{gerade!}} = \underbrace{\sum_{v \in V} \deg v}_{\textit{gerade!}} = \underbrace{\sum_{v \in V_{\mathrm{ger}}} \deg v}_{\textit{gerade!}} + \sum_{v \in V_{\mathrm{ung}}} \deg v$

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 $= 2 \cdot |E|$.

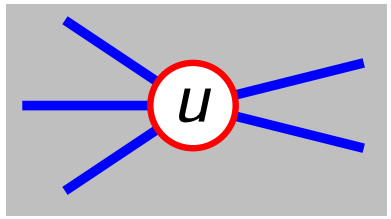
Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

Beweis.

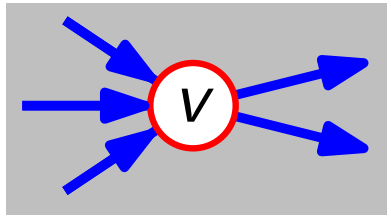
$$\underbrace{2 \cdot |E|}_{\textit{gerade!}} = \underbrace{\sum_{v \in V} \deg v}_{\textit{gerade!}} = \underbrace{\sum_{v \in V_{\mathrm{ger}}} \deg v}_{\textit{gerade!}} + \sum_{v \in V_{\mathrm{ung}}} \deg v$$

Grad eines Knotens

Def.



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



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

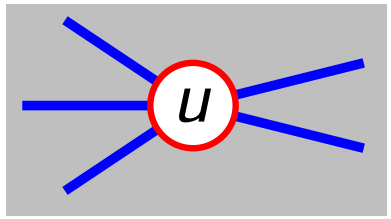
Beob. Sei $G = (V, E)$ ein ungerichteter Graph.
Dann ist die Summe aller Knotengrade $= 2 \cdot |E|$.

Sätzle. Die Anzahl der Knoten ungeraden Grades ist gerade.

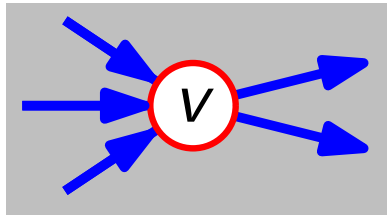
Beweis. $\underbrace{2 \cdot |E|}_{\textit{gerade!}} = \underbrace{\sum_{v \in V} \deg v}_{\textit{gerade!}} = \underbrace{\sum_{v \in V_{\mathrm{ger}}} \deg v}_{\textit{gerade!}} + \underbrace{\sum_{v \in V_{\mathrm{ung}}} \deg v}_{\Rightarrow \ \textit{gerade!}}$

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.

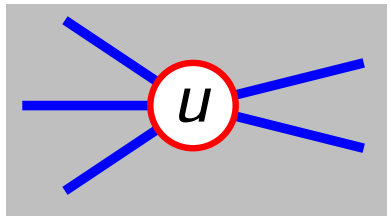
Beweis.

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

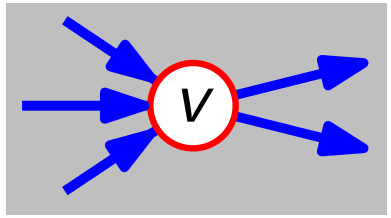
$$\sum_{v \in V_{\mathrm{ung}}} \deg v \quad \textit{gerade} \Rightarrow$$

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.

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)?

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?

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.

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)?

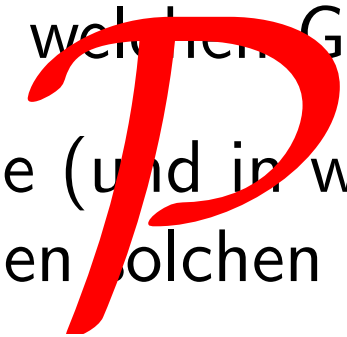
   Konstruktion: Wie (und in welcher Zeit) finde ich einen solchen Rundlauf, falls er existiert?

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?

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

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

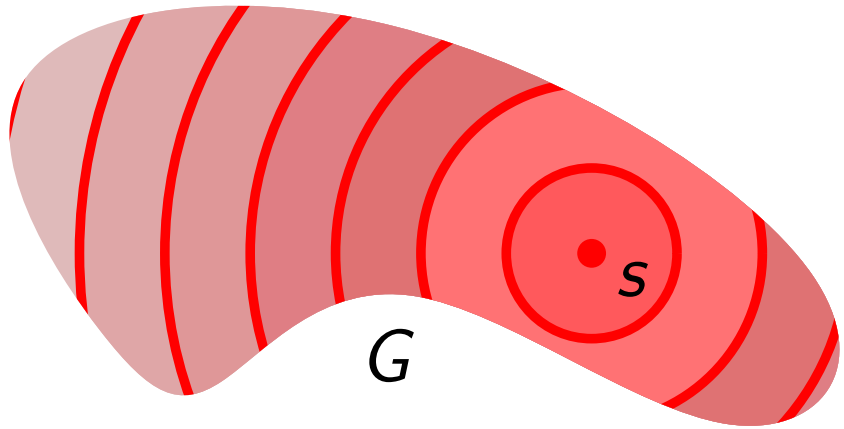
→ Vorlesung *Algorithmische Graphentheorie* (nächstes Semester!)

F: Wie durchlaufe ich einen Graphen?

Ideen?

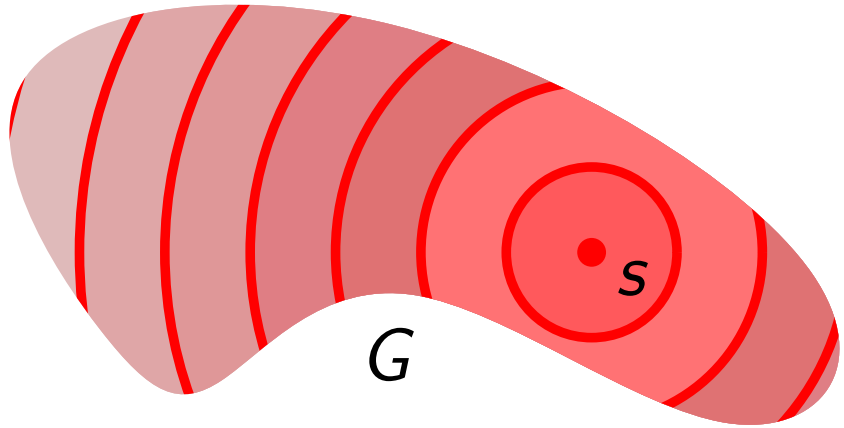
F: Wie durchlaufe ich einen Graphen?

Ideen?



F: Wie durchlaufe ich einen Graphen?

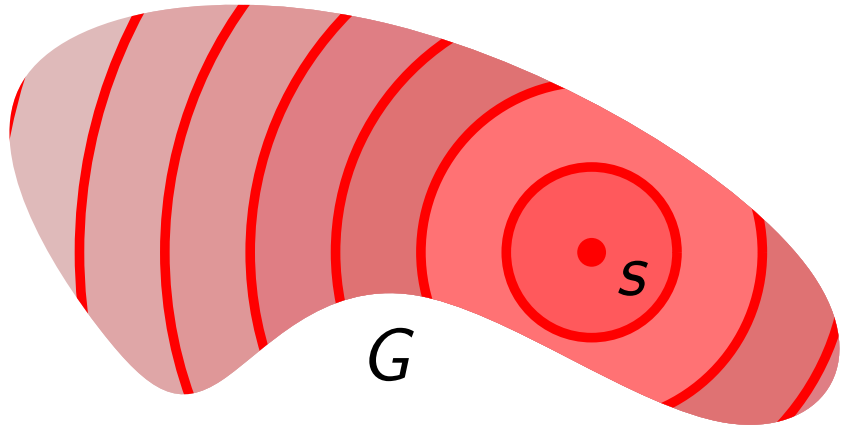
Ideen?



1. wellenförmige Ausbreitung ab einem gegebenen Startknoten s

F: Wie durchlaufe ich einen Graphen?

Ideen?

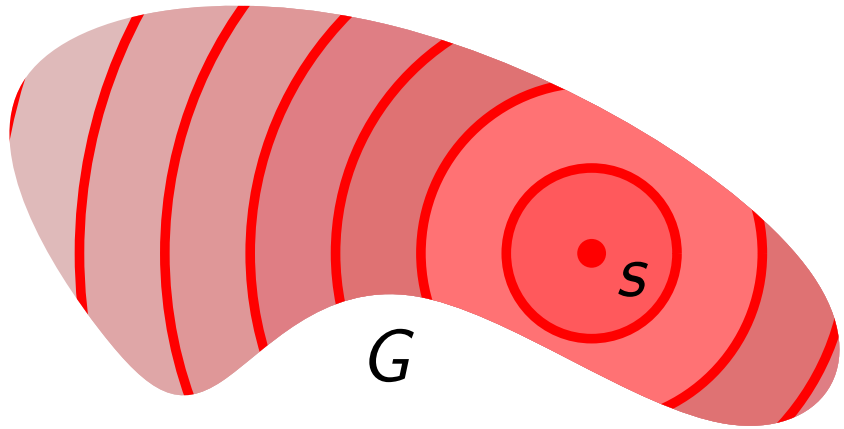


1. wellenförmige Ausbreitung ab einem gegebenen Startknoten s

   Breitensuche (breadth-first search, BFS)

F: Wie durchlaufe ich einen Graphen?

Ideen?



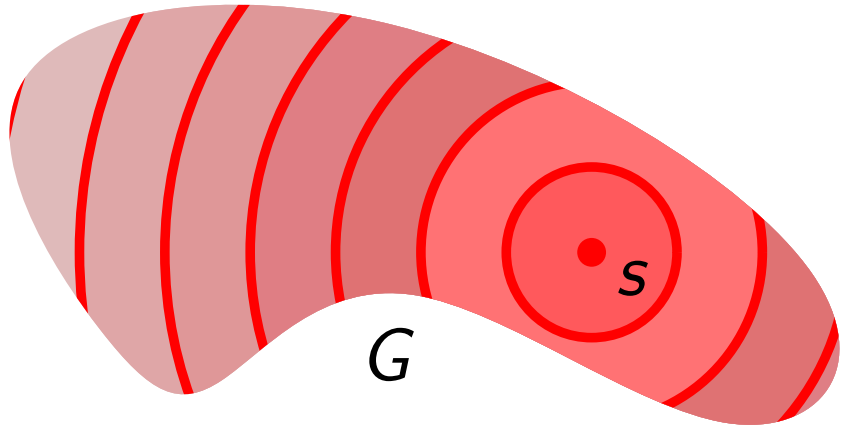
1. wellenförmige Ausbreitung ab einem gegebenen Startknoten s

   Breitensuche (breadth-first search, BFS)

2. vom Startknoten s möglichst schnell weit weg

F: Wie durchlaufe ich einen Graphen?

Ideen?



1. wellenförmige Ausbreitung ab einem gegebenen Startknoten s

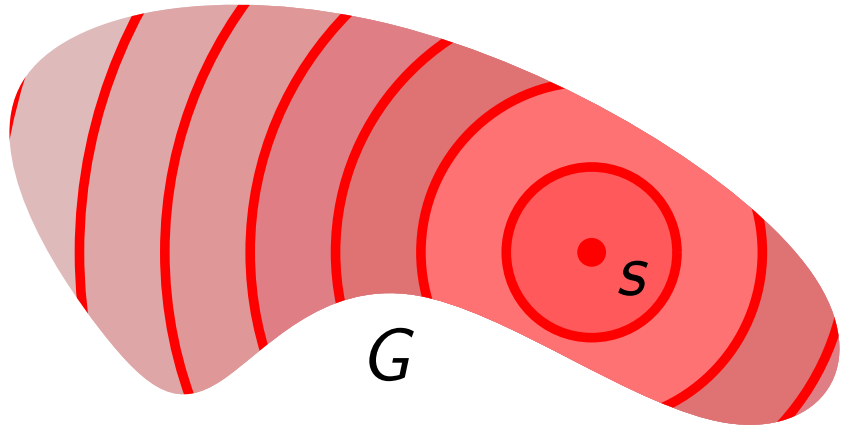
   Breitensuche (breadth-first search, BFS)

2. vom Startknoten s möglichst schnell weit weg

   Tiefensuche (depth-first search, DFS)

F: Wie durchlaufe ich einen Graphen?

Ideen?



1. wellenförmige Ausbreitung ab einem gegebenen Startknoten s

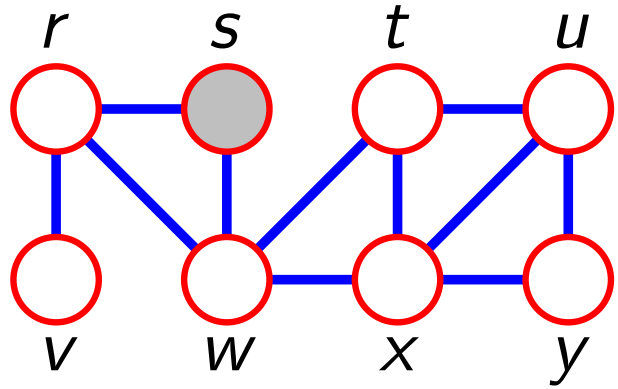
   Breitensuche (breadth-first search, BFS) ← *jetzt!*

2. vom Startknoten s möglichst schnell weit weg

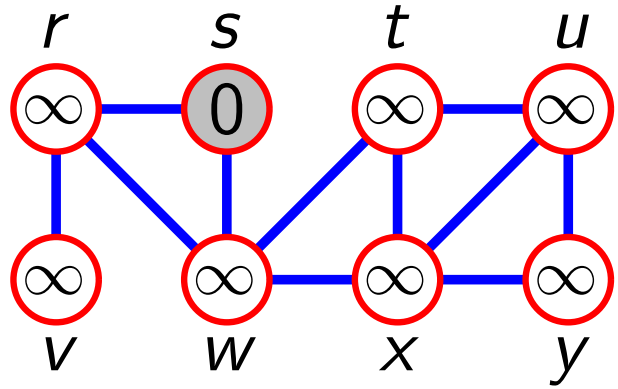
   Tiefensuche (depth-first search, DFS)

   nächstes Mal!

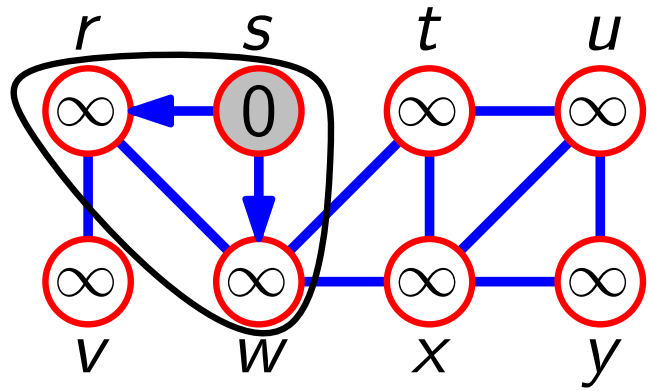
Breitensuche



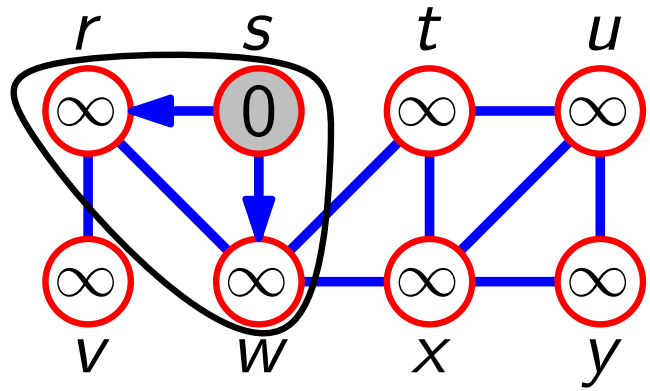
Breitensuche

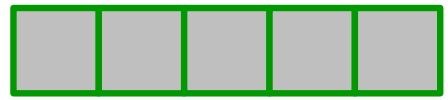


Breitensuche

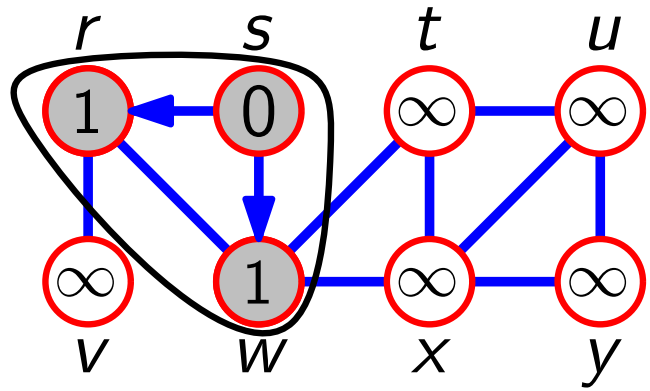


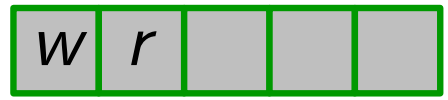
Breitensuche



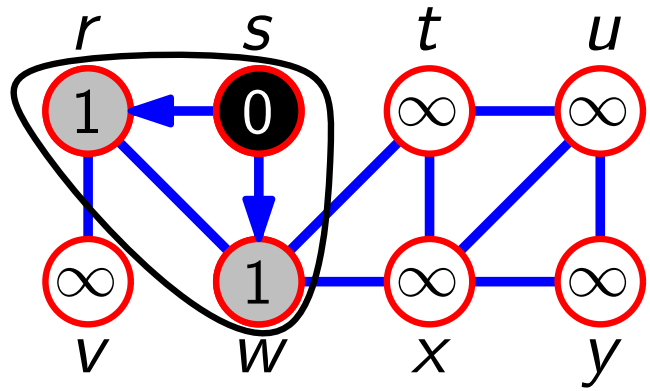


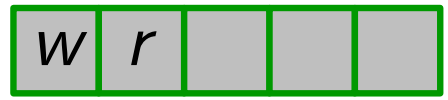
Breitensuche



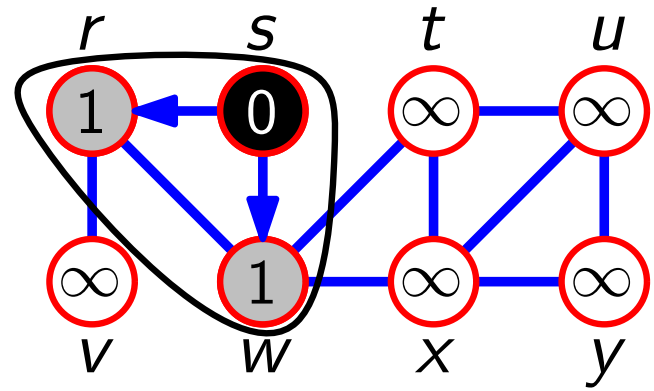


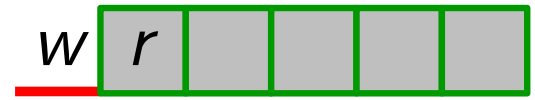
Breitensuche



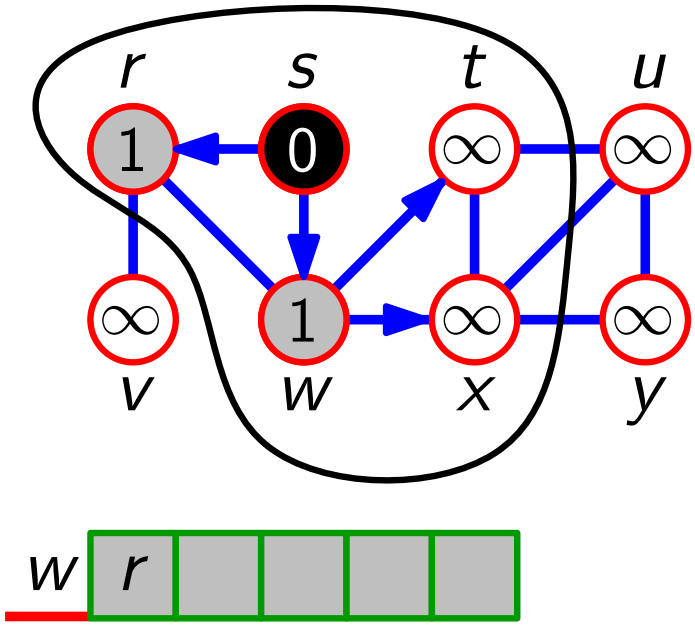


Breitensuche

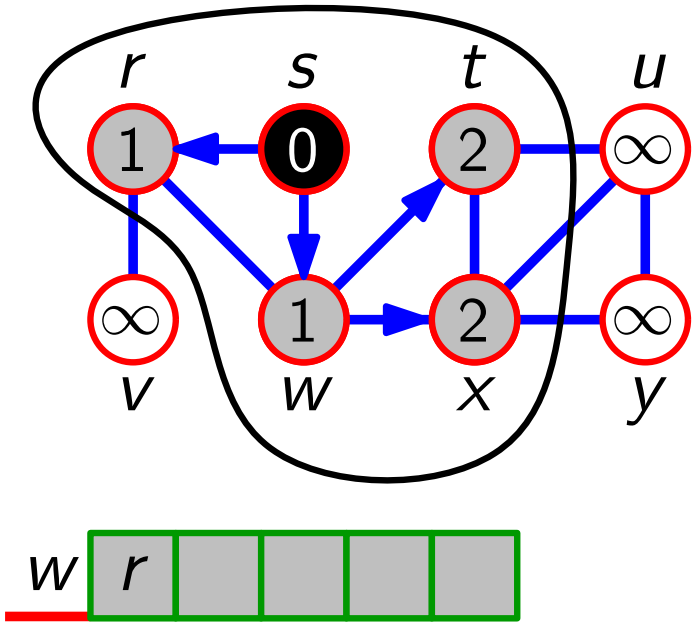




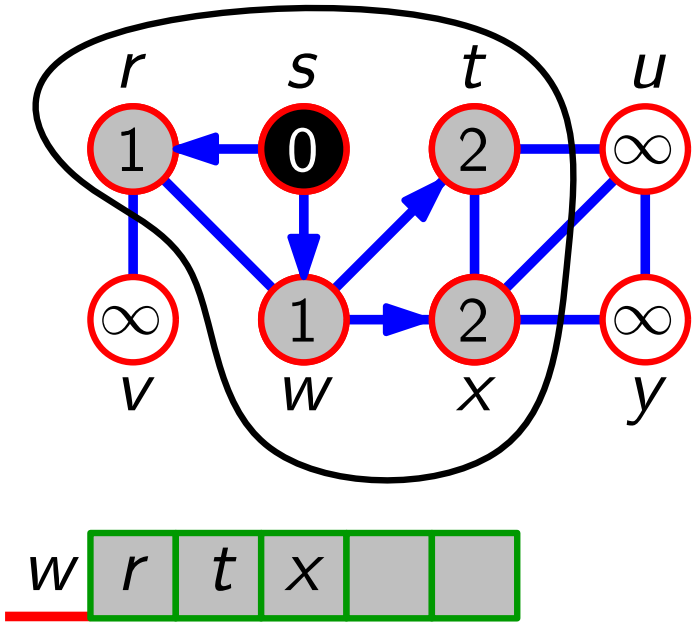
Breitensuche



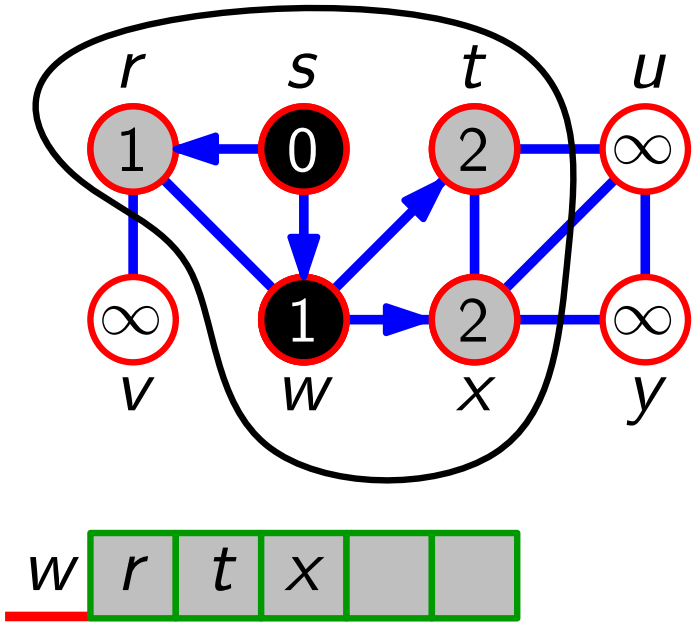
Breitensuche



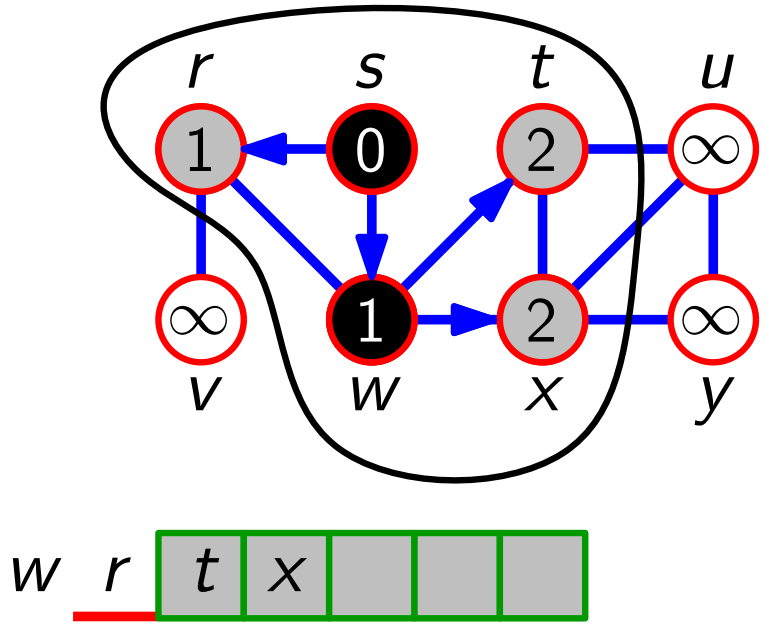
Breitensuche



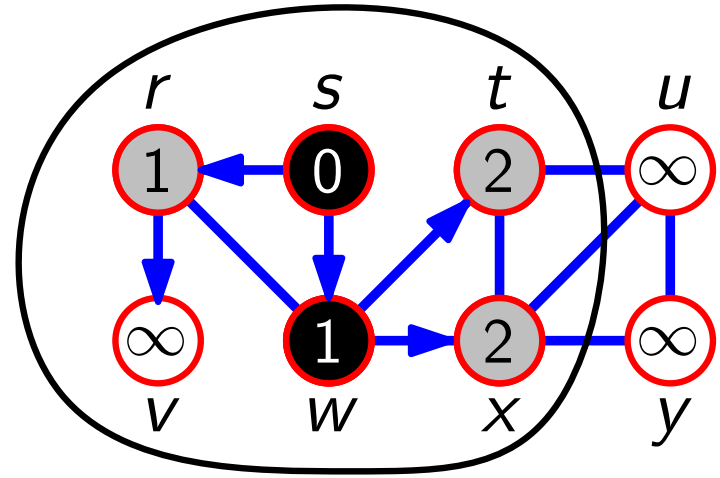
Breitensuche

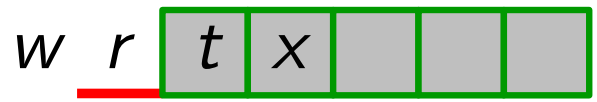


Breitensuche

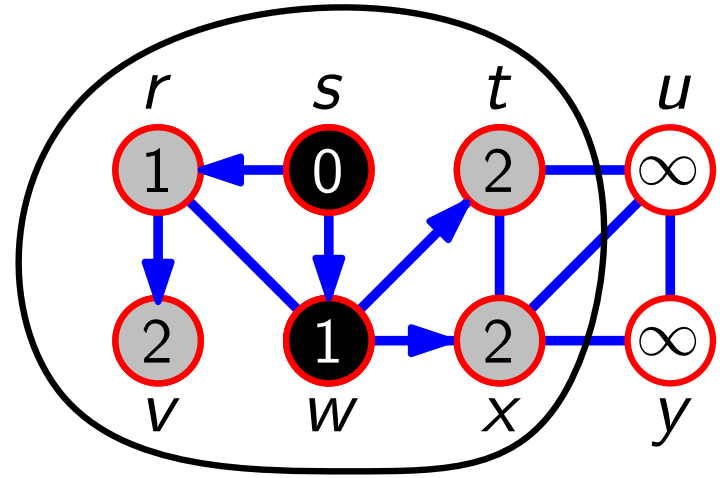


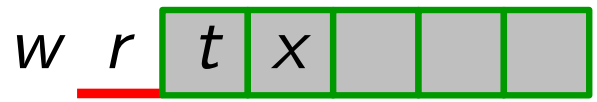
Breitensuche



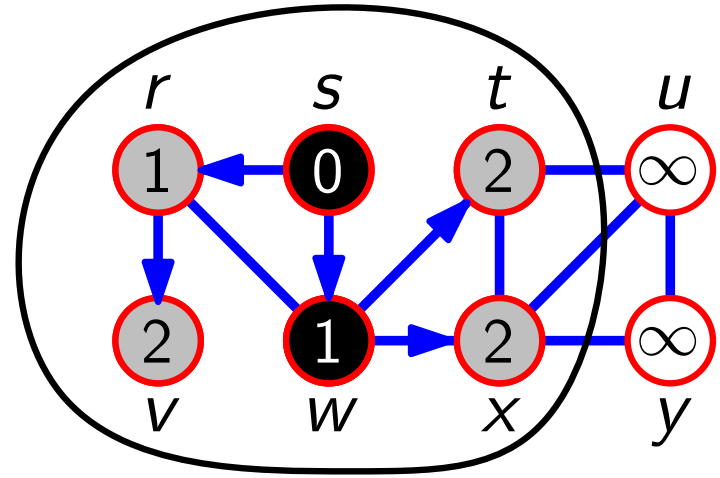


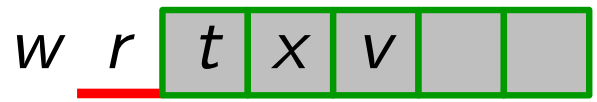
Breitensuche



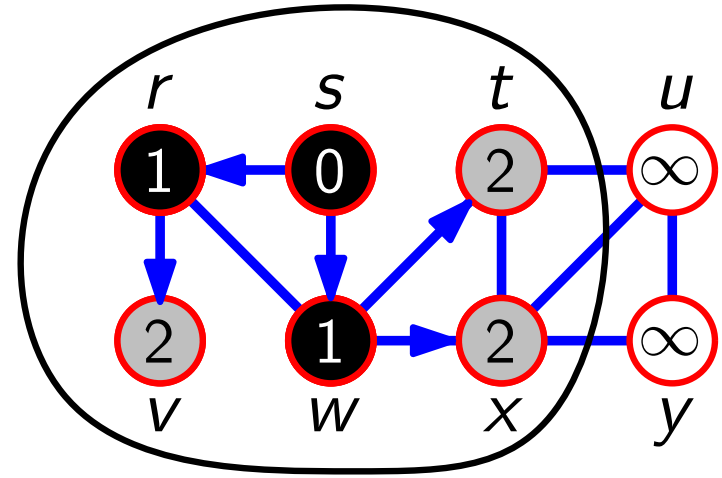


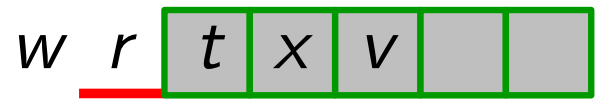
Breitensuche



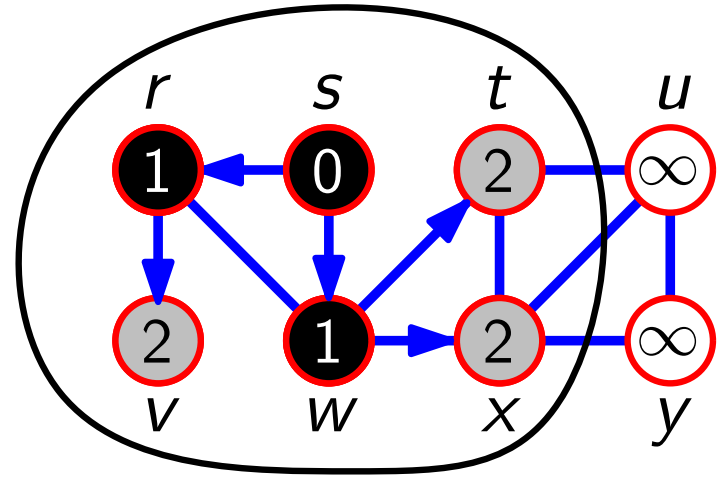


Breitensuche

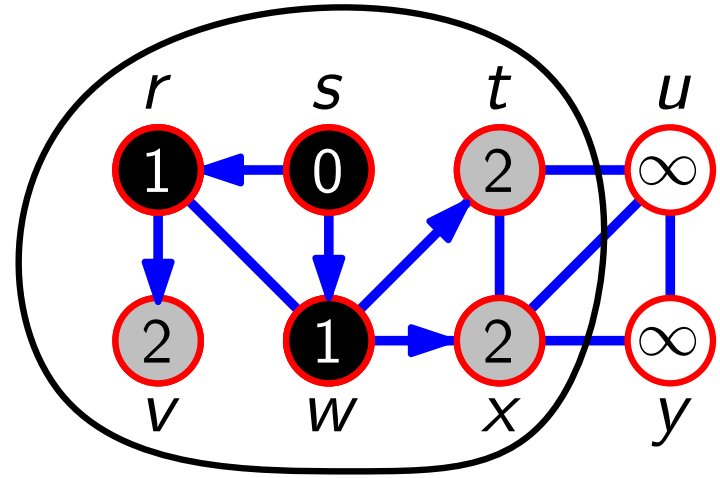




Breitensuche



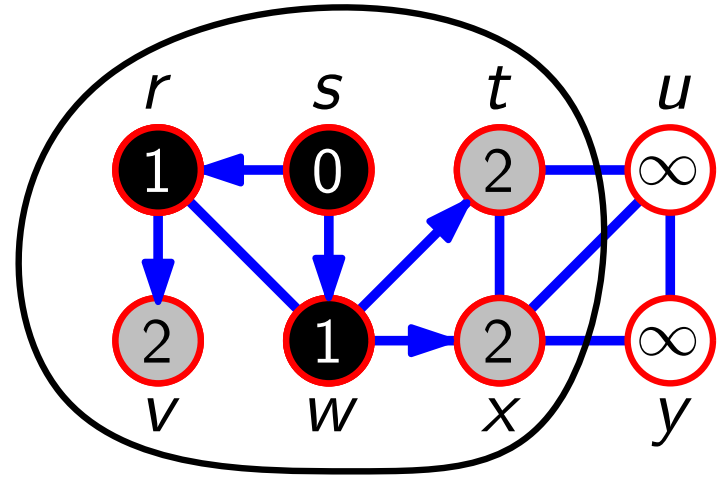
Breitensuche



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

Breitensuche

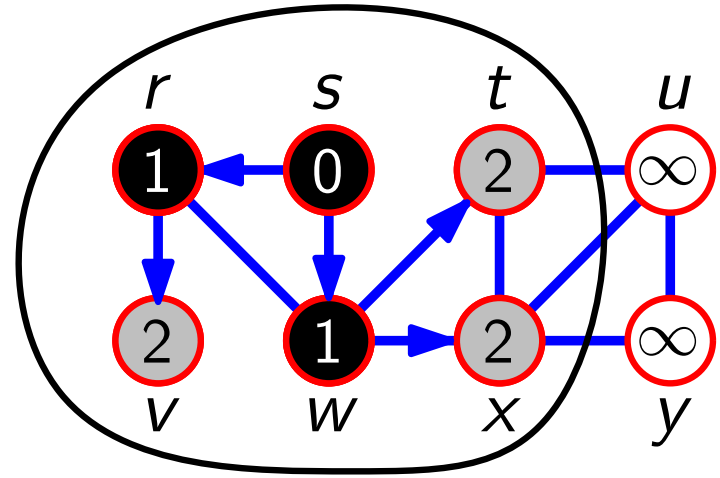
BFS(Graph G, Vertex s)
  Initialize(G, s)
  ■ = new ■■■■■ *Welche Datenstruktur??*



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

Breitensuche

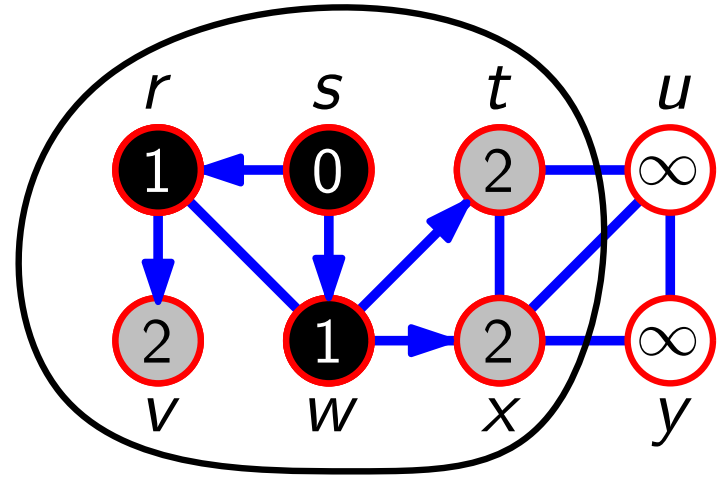
```
BFS(Graph G, Vertex s)
  Initialize(G, s)
  Q = new Queue()
```




```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.color = white
      u.d = ∞
      u.π = nil
  s.color = gray
  s.d = 0
```

Breitensuche

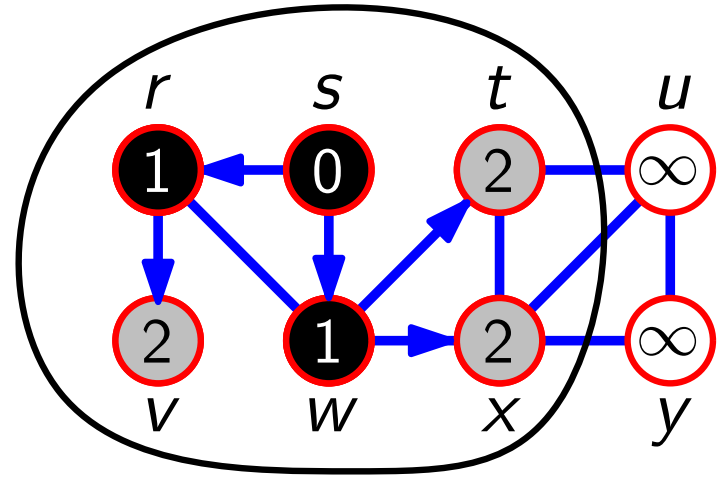
```
BFS(Graph G, Vertex s)
  Initialize(G, s)
  Q = new Queue()
  Q.Enqueue(s)
```




```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.color = white
      u.d = ∞
      u.π = nil
  s.color = gray
  s.d = 0
```

Breitensuche

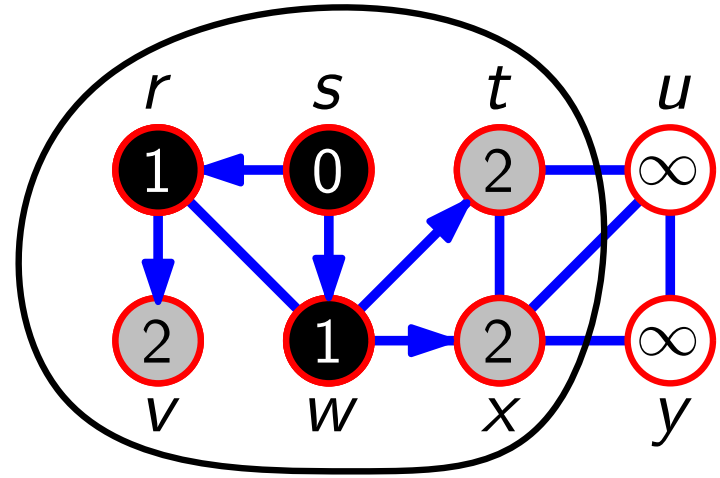
```
BFS(Graph G, Vertex s)
  Initialize(G, s)
  Q = new Queue()
  Q.Enqueue(s)
  while not Q.Empty() do
```




```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.color = white
      u.d = ∞
      u.π = nil
  s.color = gray
  s.d = 0
```

Breitensuche

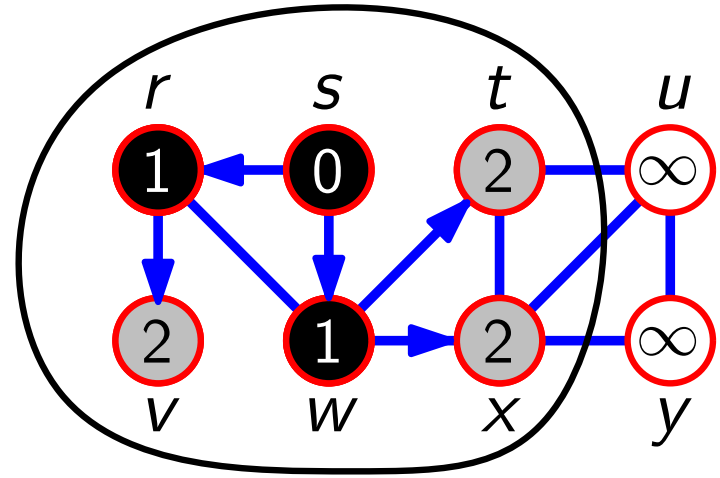
```
BFS(Graph G, Vertex s)
    Initialize(G, s)
    Q = new Queue()
    Q.Enqueue(s)
    while not Q.Empty() do
        u = Q.Dequeue()
```




```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.color = white
        u.d = ∞
        u.π = nil
    s.color = gray
    s.d = 0
```

Breitensuche

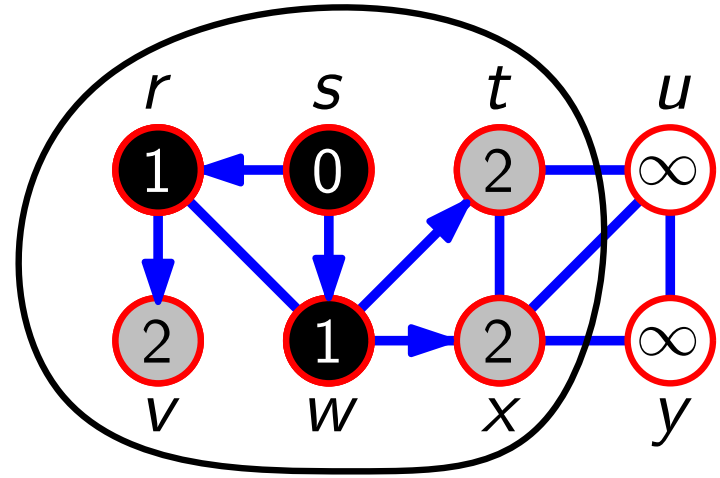
```
BFS(Graph G, Vertex s)
    Initialize(G, s)
    Q = new Queue()
    Q.Enqueue(s)
    while not Q.Empty() do
        u = Q.Dequeue()
        foreach v ∈ Adj[u] do
```

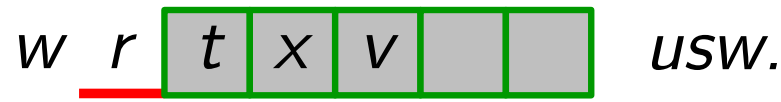



```
Initialize(Graph G, Vertex s)
    foreach u ∈ V do
        u.color = white
        u.d = ∞
        u.π = nil
    s.color = gray
    s.d = 0
```

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 ∈ Adj[u] do
          Aufgabe:
          Schreiben Sie Pseudocode, so dass:
          v.d = Länge eines kürzesten
                s-v-Weges über u, falls ...
          v.π = Vorgänger auf diesem Weg
```

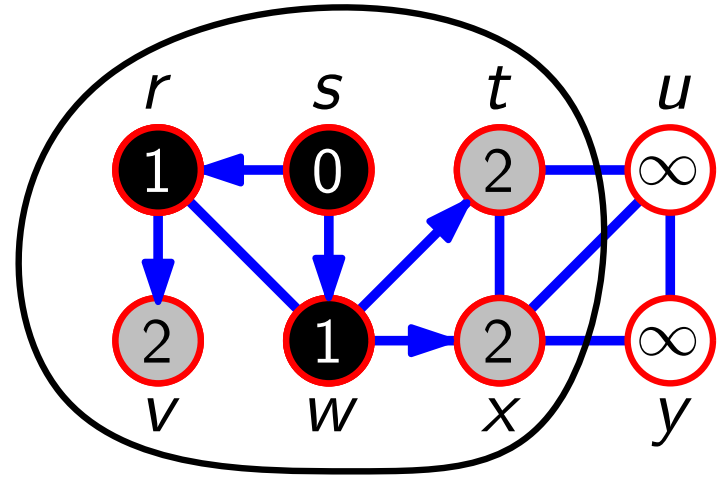





```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.color = white
      u.d = ∞
      u.π = nil
  s.color = gray
  s.d = 0
```

Breitensuche

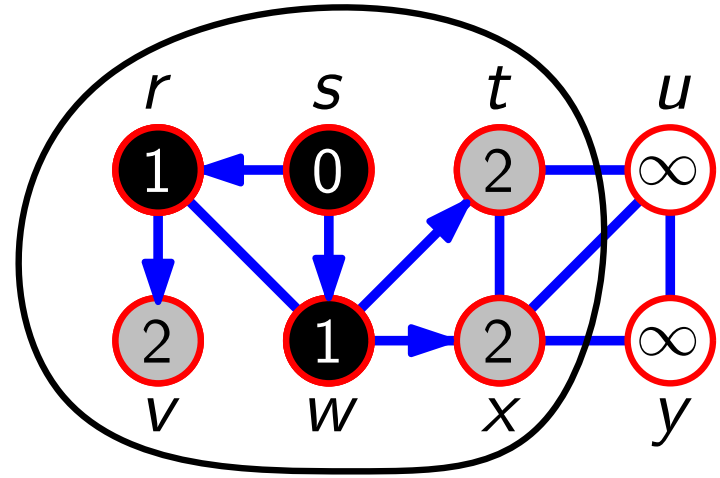
```
BFS(Graph G, Vertex s)
  Initialize(G, s)
  Q = new Queue()
  Q.Enqueue(s)
  while not Q.Empty() do
      u = Q.Dequeue()
      foreach v ∈ Adj[u] do
          if v.color == white then
              v.color = gray
              v.d = u.d + 1
              v.π = u
              Q.Enqueue(v)
```

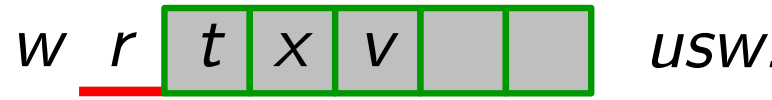



```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.color = white
      u.d = ∞
      u.π = nil
  s.color = gray
  s.d = 0
```

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 ∈ 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
```

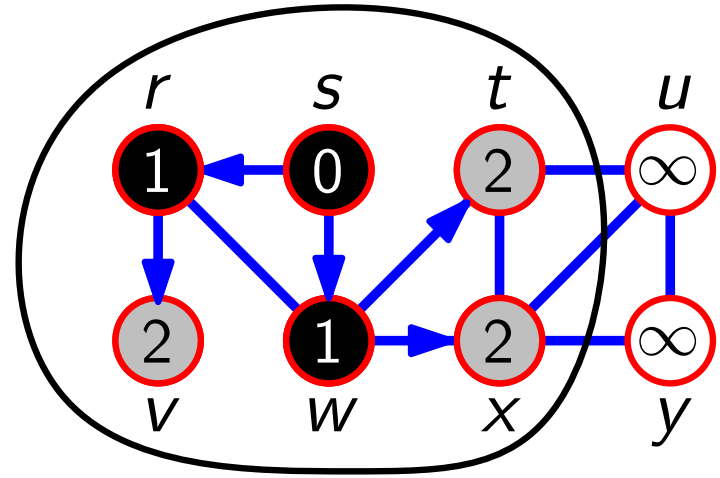





```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

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 ∈ 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
```

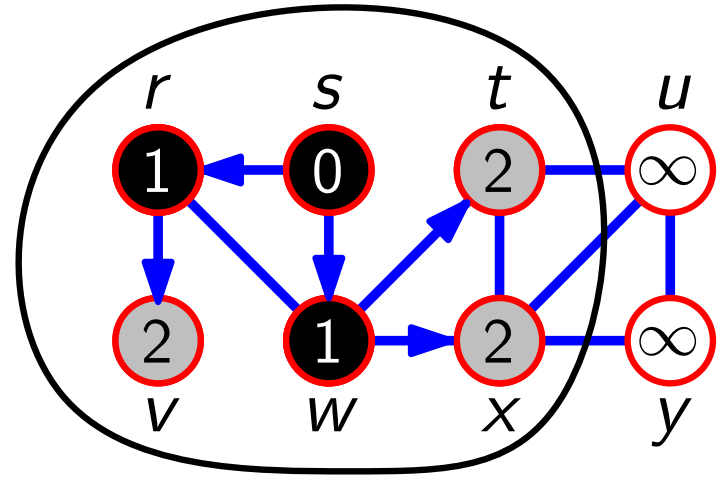



```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

Laufzeit?

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 ∈ 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
```



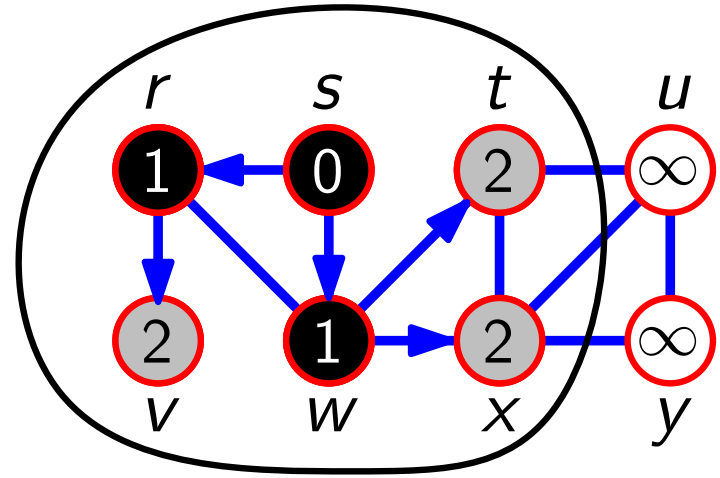

```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

Initialize

Laufzeit?

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 ∈ 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
```

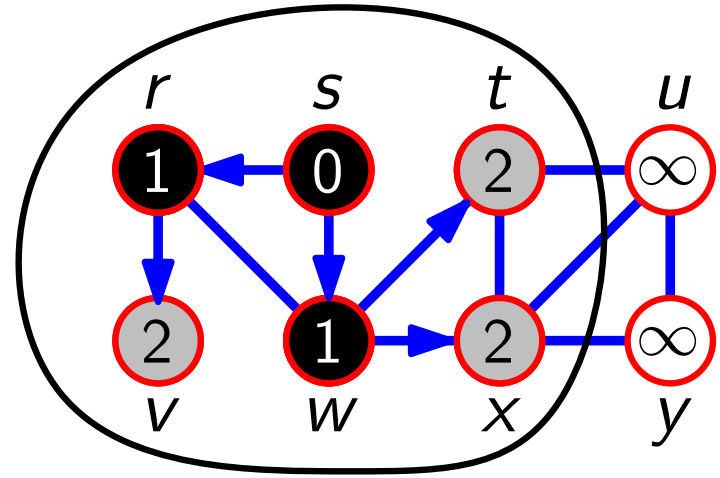



```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

Laufzeit? Initialize $O(|V|)$

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 ∈ 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
```

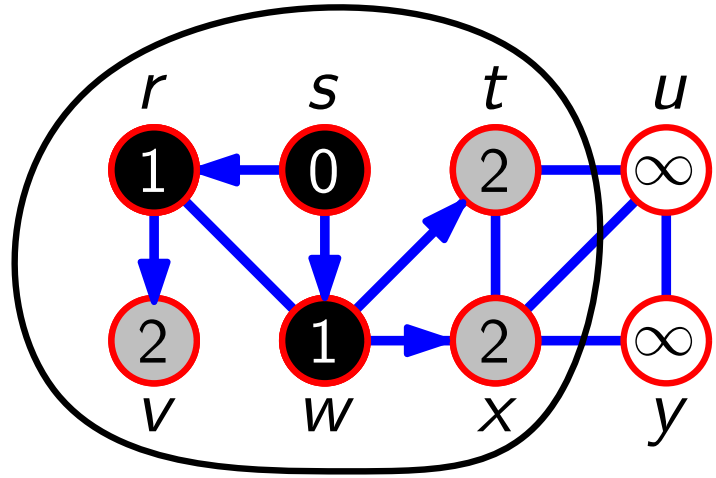



```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

Laufzeit? Initialize $O(|V|)$ En-/Dequeues

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 ∈ 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
```

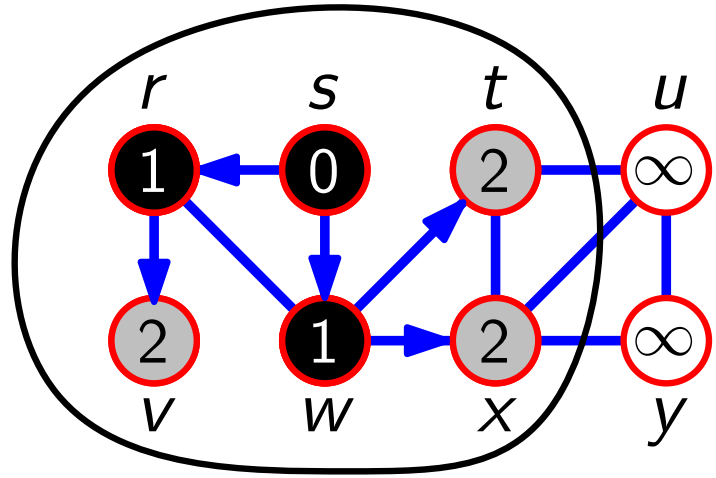



```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
      u.color = white
      u.d = ∞
      u.π = nil
  s.color = gray
  s.d = 0
```

Laufzeit? $O(|V|)$ (Initialize) $+$ $O(|V|)$ (En-/Dequeues)

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 ∈ 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
```

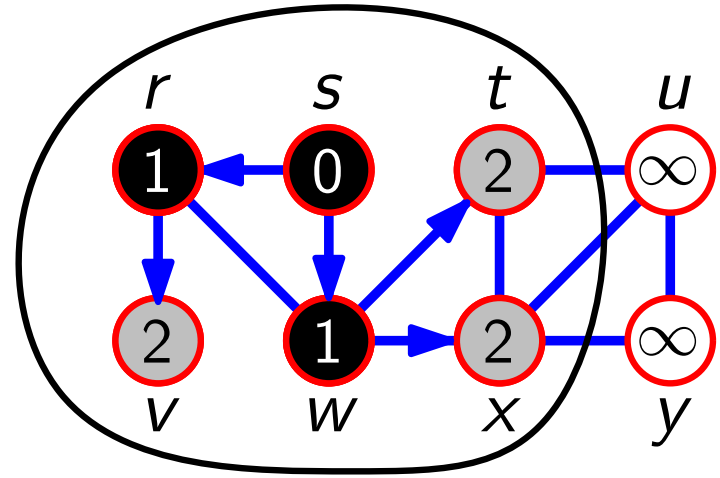



```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

Laufzeit? $O(|V|) + O(|V|)$ (Initialize; En-/Dequeues; Adjazenzlisten (foreach-Schleifen))

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 ∈ 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
```




```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

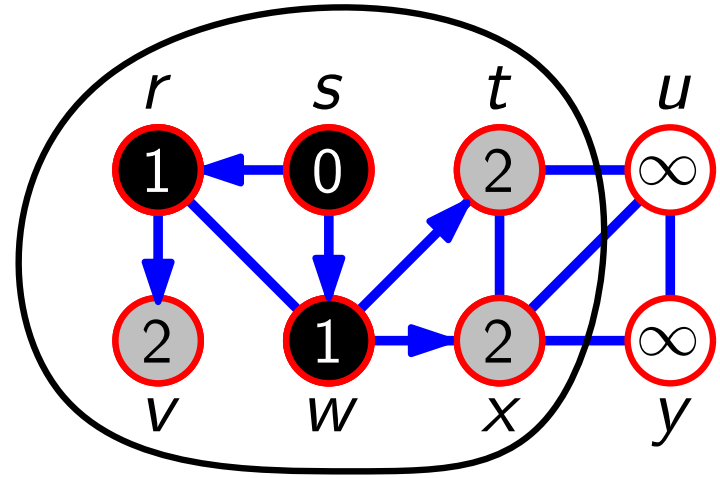
Laufzeit? $O(|V|) + O(|V|) + O(|E|)$

Initialize — En-/Dequeues — Adjazenzlisten (foreach-Schleifen)

[Beob. über Knotengrade!]

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 ∈ 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
```




```
Initialize(Graph G, Vertex s)
  foreach u ∈ V do
    u.color = white
    u.d = ∞
    u.π = nil
  s.color = gray
  s.d = 0
```

Laufzeit? $O(|V|) + O(|V|) + O(|E|) = O(|V| + |E|)$

Initialize — En-/Dequeues — Adjazenzlisten (foreach-Schleifen)

[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 $\delta(u, v) := \infty$).

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 BFS(G, s) für alle $v \in V$ gilt:
$$v.d = \delta(s, v).$$

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 BFS(G, s) für alle $v \in V$ gilt:

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

berechneter Abstand von s — *tatsächlicher* Abstand von s

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 BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *tatsächlicher* Abstand von s

Lemma 1. (Eigenschaft kürzester Wege)

Korrektheit von BFS – Vorbereitung

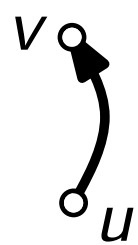
Definition. Sei $G = (V, E)$ (un)gerichteter Graph, $u, v \in V$.
$\delta(u, v) :=$ Länge eines kürzesten u-v-Wegs,
(falls v von u erreichbar; sonst $\delta(u, v) := \infty$).

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

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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 $(u, v) \in E$:

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



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 BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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.

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 BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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)

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 $\delta(u, v) := \infty$).

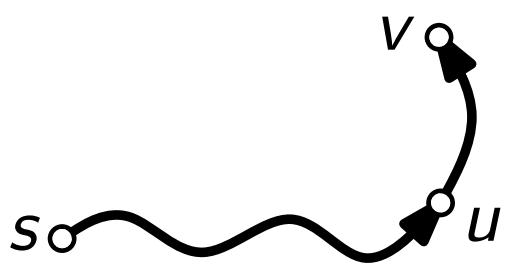
Ziel: Zeige, dass nach BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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)



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 $\delta(u, v) := \infty$).

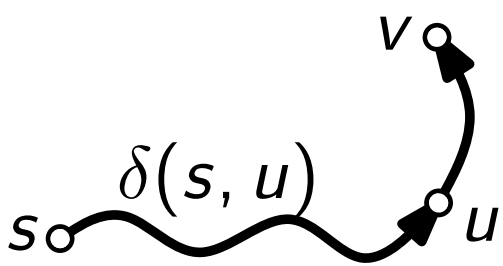
Ziel: Zeige, dass nach BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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)



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 $\delta(u, v) := \infty$).

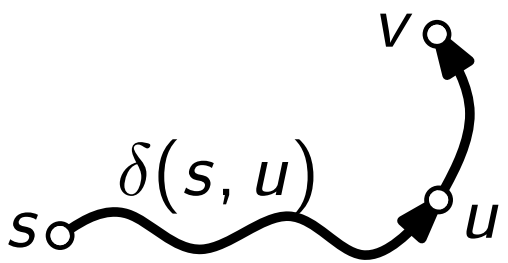
Ziel: Zeige, dass nach BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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 $\delta(s, u) + 1$.

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 $\delta(u, v) := \infty$).

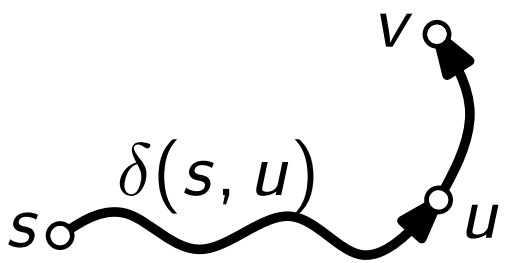
Ziel: Zeige, dass nach BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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 $\delta(s, u) + 1$.

$\Downarrow$

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 $\delta(u, v) := \infty$).

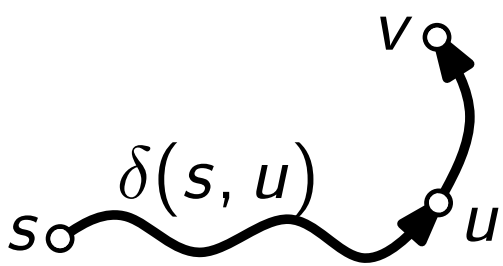
Ziel: Zeige, dass nach BFS(G, s) für alle $v \in V$ gilt:

berechneter Abstand von s — $v.d = \delta(s, v)$. — *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 – Fortsetzung

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

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

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

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

Beweis.

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

Beweis.

```
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
```

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 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
```

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 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
```

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 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
```

$k = 1$:

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 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.Enqueue(s):

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 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.Enqueue(s):

- $s.d = 0 = \delta(s, s)$

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 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.Enqueue(s):

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

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 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.Enqueue(s):

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

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 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.Enqueue(s):

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

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 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.Enqueue(s): ✓

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

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 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.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.Enqueue(v):

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 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.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.Enqueue(v):

v war gerade noch weiß und ist benachbart zu u.

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 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.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.Enqueue(v):

v war gerade noch weiß und ist benachbart zu u.
$v.d = u.d + 1$

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 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.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.Enqueue(v):

v war gerade noch weiß und ist benachbart zu u.
$v.d = u.d + 1 \geq \delta(s, u) + 1$

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 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.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.Enqueue(v):

v war gerade noch weiß und ist benachbart zu u.

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

Induktionsannahme für u

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 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.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.Enqueue(v):

v war gerade noch weiß und ist benachbart zu u.

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

Induktionsannahme für u
($u.d$ wurde gesetzt, als Anz. Enqueue-Oper. $< k$)

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

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 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.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.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 ($u.d$ wurde gesetzt, als Anz. Enqueue-Oper. $< k$) — Lemma 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 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.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.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 ($u.d$ wurde gesetzt, als Anz. Enqueue-Oper. $< k$) — Lemma 1

Jetzt ist v grau.

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 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.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.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 ($u.d$ wurde gesetzt, als Anz. Enqueue-Oper. $< k$) — Lemma 1

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

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 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.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.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 ($u.d$ wurde gesetzt, als Anz. Enqueue-Oper. $< k$) — Lemma 1

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

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 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.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.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 ($u.d$ wurde gesetzt, als Anz. Enqueue-Oper. $< k$); Lemma 1

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

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 $Q = \langle v_1, v_2, \ldots, v_r \rangle$ während BFS. Dann gilt:

(A) $v_r.d \leq v_1.d + 1$ und

(B) $v_i.d \leq v_{i+1}.d$ für $i = 1, \ldots, r-1$.

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 $Q = \langle v_1, v_2, \ldots, v_r \rangle$ während BFS. Dann gilt:

(A) $v_r.d \leq v_1.d + 1$ und

(B) $v_i.d \leq v_{i+1}.d$ für $i = 1, \ldots, r-1$.

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

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 $Q = \langle v_1, v_2, \ldots, v_r \rangle$ während BFS. Dann gilt:

(A) $v_r.d \leq v_1.d + 1$ und

(B) $v_i.d \leq v_{i+1}.d$ für $i = 1, \ldots, r-1$.

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.

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 $Q = \langle v_1, v_2, \ldots, v_r \rangle$ während BFS. Dann gilt:

(A) $v_r.d \leq v_1.d + 1$ und

(B) $v_i.d \leq v_{i+1}.d$ für $i = 1, \ldots, r-1$.

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

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.

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.

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.

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).

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.

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 $\Rightarrow v.d \geq \delta(s, v)$.

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.: $v.d \leq \delta(s, v)$.

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.

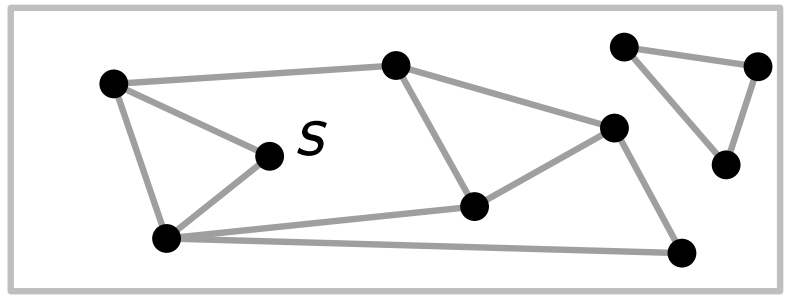
Lemma 2 $\Rightarrow v.d \geq \delta(s, v)$. Noch z.z.: $v.d \leq \delta(s, v)$.

Widerspruchsbeweis mit Wahl des „kleinsten Schurken“.

Siehe Kapitel 22.2 [CLRS].

BFS-Bäume

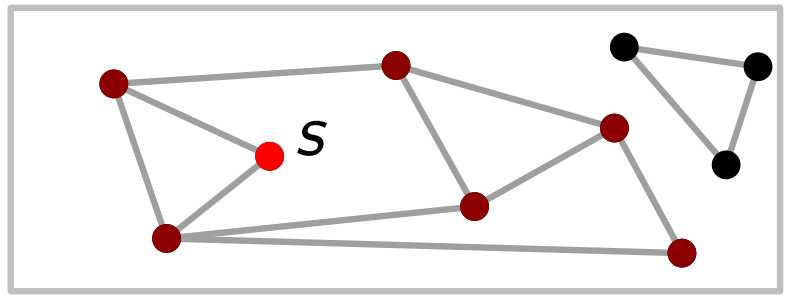
Betrachte den *Vorgänger-Graphen* $G_\pi = (V_\pi, E_\pi)$ von G:



BFS-Bäume

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

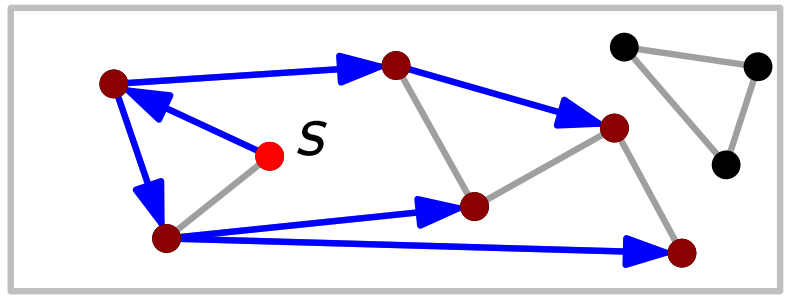
- $\boxed{V_\pi} = \{v \in V : v.\pi \neq nil\} \cup \{s\}$



BFS-Bäume

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

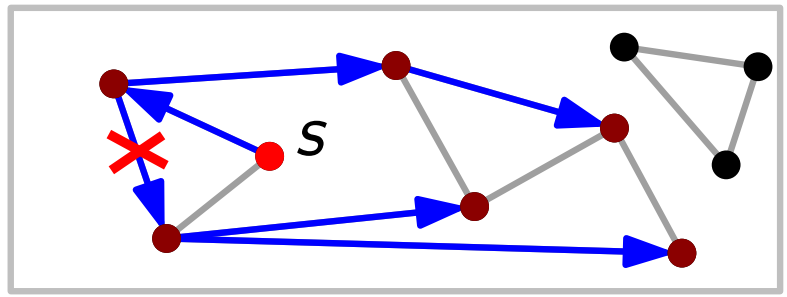
- $V_\pi = \{v \in V : v.\pi \neq nil\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) : v \in V_\pi \setminus \{s\}\}$



BFS-Bäume

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

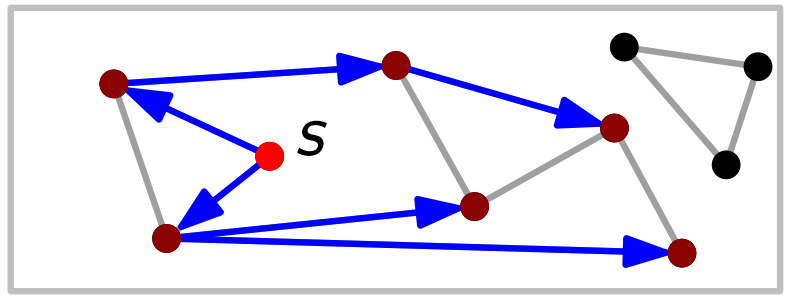
- $V_\pi = \{v \in V : v.\pi \neq nil\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) : v \in V_\pi \setminus \{s\}\}$



BFS-Bäume

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

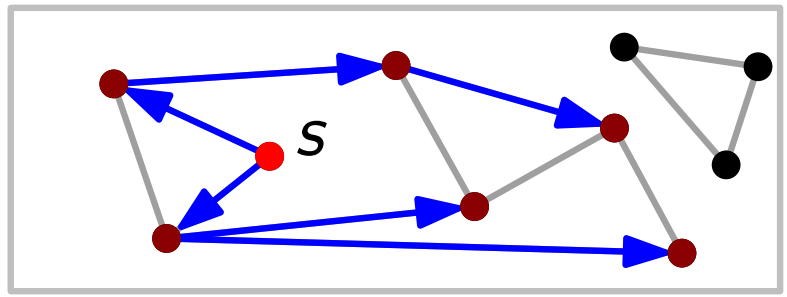
- $V_\pi = \{v \in V : v.\pi \neq nil\} \cup \{s\}$
- $E_\pi = \{(v.\pi, v) : v \in V_\pi \setminus \{s\}\}$



BFS-Bäume

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

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

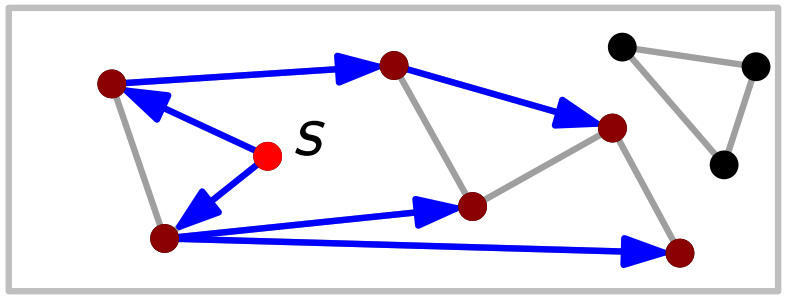


Klar: G_π ist ein Baum

BFS-Bäume

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

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

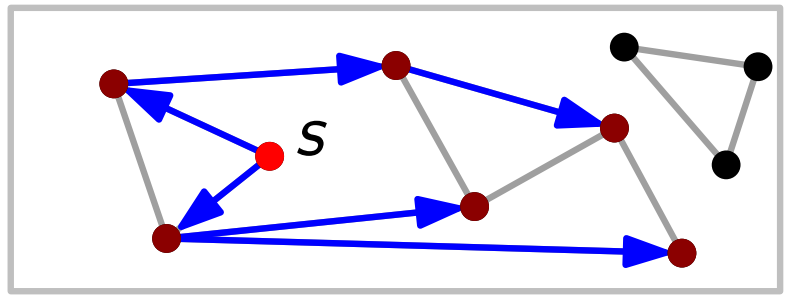


Klar: G_π ist ein Baum (da zshg. und $|E_\pi| = |V_\pi| - 1$).

BFS-Bäume

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

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



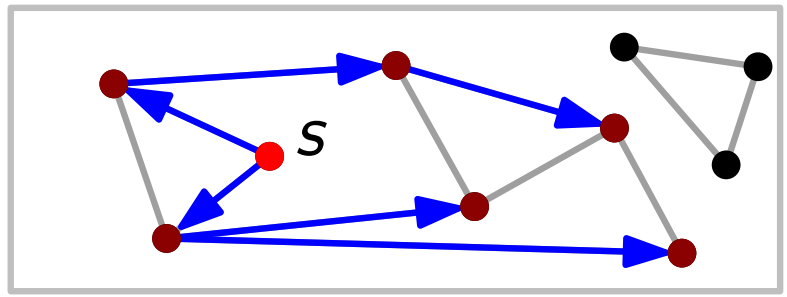
Klar: G_π ist ein Baum (da zshg. und $|E_\pi| = |V_\pi| - 1$).

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

BFS-Bäume

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

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



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

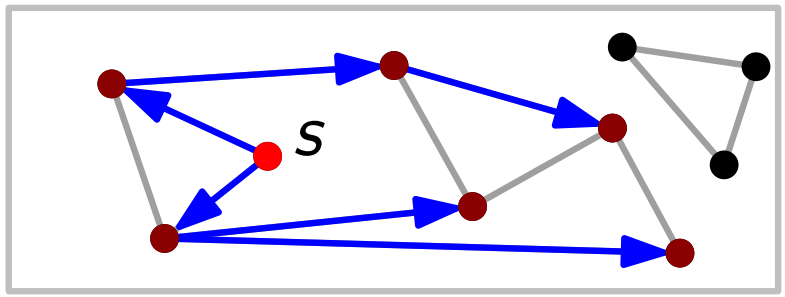
Beh.: G_π ist ein *Kürzeste-Wege-Baum* (oder *BFS-Baum*), d.h.

- $V_\pi = \{v \in V : v$ erreichbar von $s\}$

BFS-Bäume

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

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



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

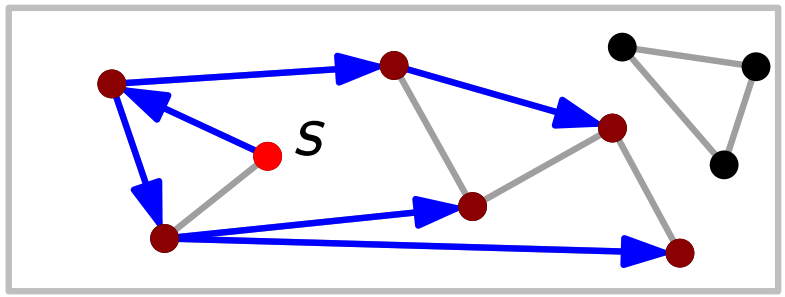
Beh.: G_π ist ein *Kürzeste-Wege-Baum* (oder *BFS-Baum*), d.h.

- $V_\pi = \{v \in V : v$ 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.

BFS-Bäume

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

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



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

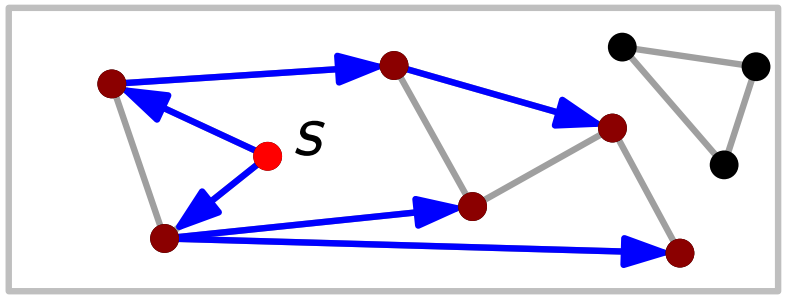
Beh.: G_π ist ein *Kürzeste-Wege-Baum* (oder *BFS-Baum*), d.h.

- $V_\pi = \{v \in V : v$ 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.

BFS-Bäume

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

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



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.

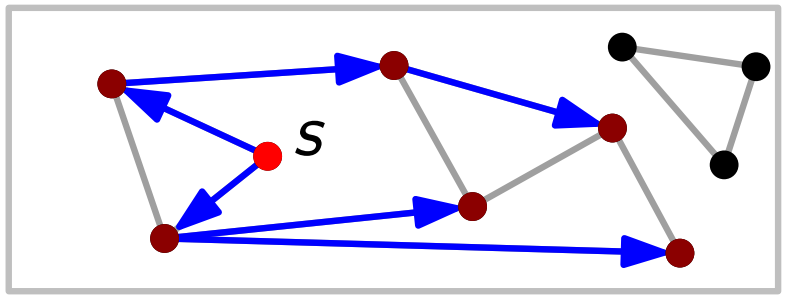
- $V_\pi = \{v \in V : v$ erreichbar von $s\}$
- für alle $v \in V_\pi$ enthält G_π einen eindeutigen Weg von s nach v, der ein kürzester s-v-Weg ist.

Bew.:

BFS-Bäume

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

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



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. □