

## Organisatorisches

# Graphentheorie: Organisatorisches

## Programmieren und Software-Engineering Theorie

2. September 2025

- Dieser Foliensatz enthält alle organisatorischen Informationen zu dieser Doppelstunde POS Theorie
- Bei etwaigen Fragen zur Benotung, Leistungsüberprüfungen etc. bitte immer als erstes in diesem Dokument nachsehen!

## Organisatorisches

- Die Gesamtnote POS ergibt sich durch POS (Programmieren) und POS (Theorie)
- 3/4 KIF/AIF:
  - 5h Java
  - 2h Graphentheorie (WS), Algorithmen, Formale Sprachen (SS)
- 5/6 BIF/CIF:
  - 2h Java
  - 2h Graphentheorie (WS), Algorithmen, Formale Sprachen (SS)
- Für positive POS Modulnote: beide Teile müssen positiv bestanden werden!

## Beurteilung

- Zwei schriftliche Leistungsüberprüfungen (SLÜs) pro Semester
- Bei jeder SLÜ sind 40 Punkte zu erreichen.
- Semesternote (Teilbereich):

$$\frac{\text{Punkte 1. SLUE} + \text{Punkte 2. SLUE}}{2}$$

mit folgendem Punkteschlüssel:

Punkte	Note
[0, 20]	Nicht Genügend
(20, 25]	Genügend
(25, 30]	Befriedigend
(30, 35]	Gut
(35, 40]	Sehr Gut

## Mitarbeit

- Mitarbeit: Präsentation von Beispielen an der Tafel, laufende konstruktive Beiträge zum Unterricht, etc.
- Durch sehr gute Mitarbeit können Sie die bessere Note erreichen, wenn Sie
  - zwischen zwei Noten stehen, bzw.
  - äußerst knapp an der Grenze zur besseren Note stehen.

POS (Theorie)

Organisatorisches

5 / 12

## Schriftliche Leistungsüberprüfungen (SLÜs)

- **Anmerkung:** Es müssen nicht unbedingt beide SLÜs positiv sein (theoretisch möglich 40 Punkte + 1 Punkt  $\Rightarrow$  41 Punkte insgesamt  $\Rightarrow$  dividiert durch 2 ergibt 20.5 Punkte  $\Rightarrow$  positiv im Teilbereich).
- Werden beide SLÜs *nicht* absolviert  $\Rightarrow$  Note *nicht beurteilt*.
- Bei nachweislicher Verhinderung<sup>1</sup> bei einer SLÜ besteht die Möglichkeit diese nachzuholen. Nehmen Sie unverzüglich<sup>2</sup> Kontakt per Mail auf.
- Bei negativem Abschluss im regulären Semester besteht die Möglichkeit eines *Kolloquiums*.

<sup>1</sup>Krankmeldung, ärztliche Zeitbestätigung, Zeitbestätigung zu nicht aufschiebbarem Amtsweg

<sup>2</sup>z.B. nach Ende des Krankenstandes

POS (Theorie)

Organisatorisches

7 / 12

## Schriftliche Leistungsüberprüfungen (SLÜs)

- Die SLÜs sind “**open book**”: d.h. alle *schriftlichen* Unterlagen können verwendet werden. **Alle elektronischen Hilfsmittel sind verboten!**
- Bitte: möglichst viel auf die Angabe schreiben und möglichst wenige (selbst mitzubringende) Zusatzblätter verwenden.
- Dies spart Ressourcen: Papier, Gewicht, Dateigröße beim Digitalisieren, etc.
- Als Zusatzblätter sind ausschließlich A4-Blätter mit ordentlichen Kanten zulässig, da nur diese vom Mehrblatteinzugs des Scanners korrekt verarbeitet werden.
- Die Verwendung von Bleistiften ist erlaubt, rote Farbe ist jedoch zu vermeiden.

POS (Theorie)

Organisatorisches

6 / 12

## Kommunikation, Kolloquien

- Auskünfte zu Noten werden ausschließlich persönlich im Unterricht erteilt. (Idealerweise zu Beginn oder kurz vor Ende der Unterrichtseinheit)
- Bei Anfragen per Mail ist die
  - schulinterne Mailadresse verwenden,
  - die besuchte Klasse und Fach anzuführen.
- Terminvereinbarungen zu Kolloquien:
  - Nutzen Sie nach Möglichkeit einen der Sammel-Termine (September, November, Jänner).
  - Führen Sie zusätzlich das Jahr/Semester an auf das sich das Kolloquium bezieht.
  - Machen Sie zwei bis drei konkrete Terminvorschläge, die mit meinem Stundenplan vereinbar sind (also kein Unterricht zu dieser Zeit).

POS (Theorie)

Organisatorisches

8 / 12

## Unterlagen

- Die Vortragsfolien decken die Inhalte ab!
- Eigene (ergänzende) Mitschrift, insbesondere zu Beispielen und Erklärungen, wird empfohlen!

### Achtung!

Die Präsentationsfolien enthalten Animationen die zu gewissen Beispielen/Algorithmen eine Schritt-für-Schritt Erklärung enthalten. Diese Animationen sind am Besten im Vollbildmodus anzusehen! Im gedruckten Handout finden sich lediglich verkürzte Darstellungen mit weniger Zwischenschritten.

## Gestaltung des Unterrichts

- Vortrag/Inputphase zur Theorie
- Beispiele zur Veranschaulichung, schrittweisen selbstständigen Erarbeitung des Themengebietes
- Online Graphen-Tool:  
[graphen.theoretische-informatik.at](http://graphen.theoretische-informatik.at)
- Übungsaufgaben (ohne Abgabe, d.h. freiwillig)

## Motivation

- *Inhalte* der Graphentheorie
- Jedoch auch:
  - Abstraktes Denken
  - Problemlösungskompetenz
  - Formales, analytisches und logisches Denken
- Zu den Inhalten des Wintersemesters (WS) gibt es im Sommersemester (SS) eine **Programmieraufgabe!**
- Programm zu *komplexen* und *abstrakten* Themen soll hierbei *selbstständig* von Grund auf konzeptioniert, entwickelt und getestet werden.
- Umsetzung von Programmiertechniken (aus Java-Teil) von graphentheoretischen Inhalten

## Weiterführende Inhalte (\*)

- Manche Inhalte sind speziell für besonders Interessierte gedacht
- Diese Inhalte zählen nicht zum Kernstoffgebiet, und müssen somit nicht gelernt/gekonnt-verstanden werden
- Die Kennzeichnung dieser Inhalte erfolgt durch die blaue Titel- und Fußzeile in den Folien!

# Graphentheorie: Einleitung

## Programmieren und Software-Engineering Theorie

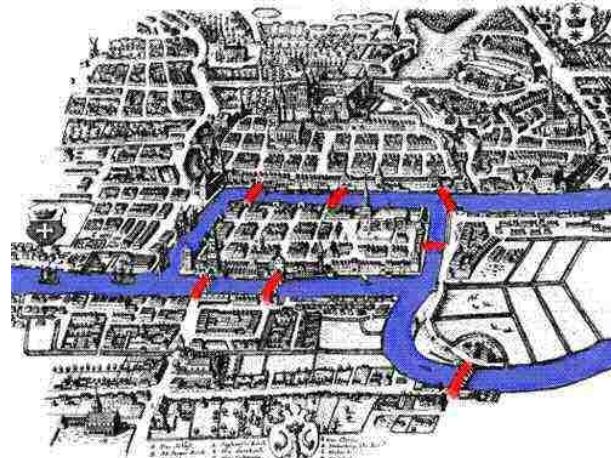
2. September 2025

## GESCHICHTE

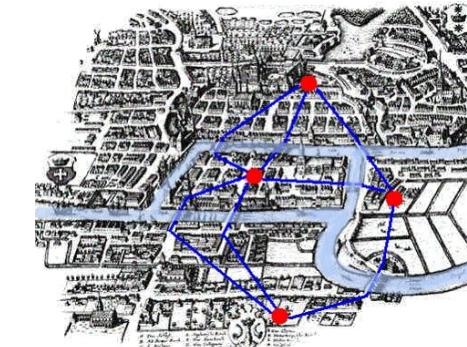
Anfänge der Graphentheorie 1736:

Leonhard Euler untersucht das "Königsberger Brückenproblem"

**Fragestellung:** Gibt es einen Weg der jede der sieben Brücken über den Fluss Pregel genau *einmal* benutzt?



Euler modellierte erstmals eine Problemstellung mit einem **Graphen**



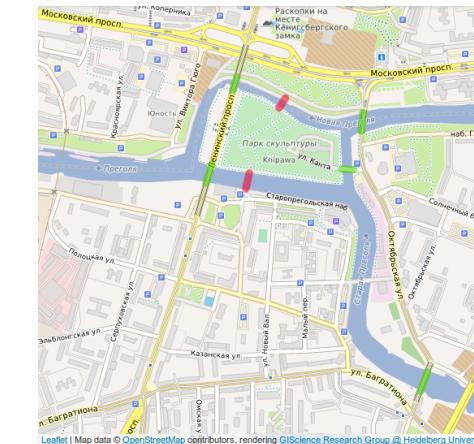
Die roten Kreise werden als *Knoten* bezeichnet, die blauen Verbindungen als *Kanten*.

**Euler fand heraus: so ein Rundgang existiert nicht!**

Seit 1946 trägt Königsberg den Namen *Kaliningrad* und ist eine russische Exklave nördlich von Polen.



Von den historischen sieben Brücken existieren heute nur noch fünf!



## Navigationssysteme

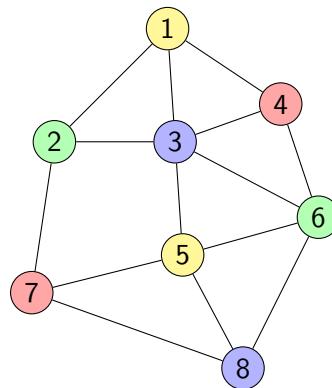
- Berechnung von kürzesten Wegen in Navigationssystemen
- Straßennetz wird als Graph repräsentiert

# ANWENDUNGEN



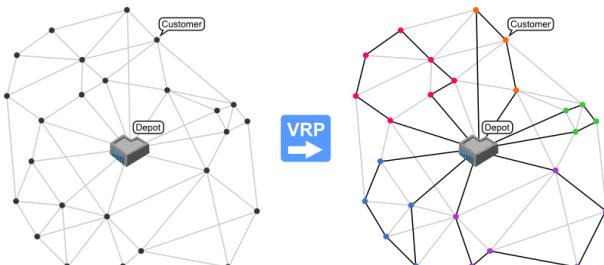
## Mobilfunk

- Frequenzplanung in der Mobilfunkindustrie
- Interferenzen bei gleichen Frequenzen und nahen Senderstandorten
- Modellierung durch Graphen:
  - Ordne jedem Knoten eine Frequenz ("Farbe") zu
  - Benachbarte Knoten dürfen nicht die selbe Farbe haben
  - Finde minimale Anzahl an Farben: Optimierungsproblem



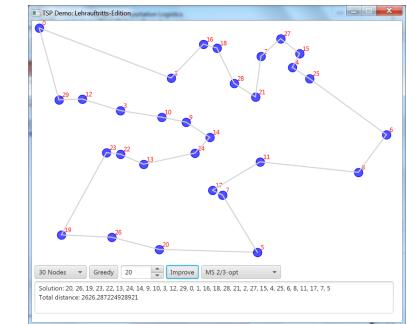
## Vehicle Routing Problem

- Ähnlich zu TSP, jedoch ist hier auszuwählen welches Fahrzeug einer vorgegebenen Flotte welchen Ort (Kunden) anfährt
- Häufiges Problem in der Transportlogistik
- Ebenso extrem schwierig zu lösen: deshalb meist Anwendung von Heuristiken (teilweise auf Zufallsoperationen basierende Näherungsverfahren)



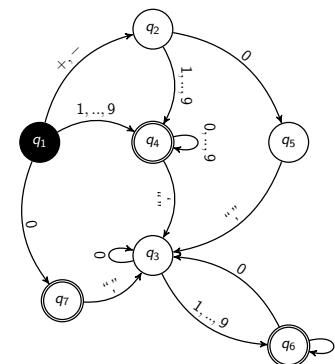
## Problem des Handlungsreisenden

- "Travelling Salesman Problem" (TSP)
- Archetypisches Optimierungsproblem auf Graphen
- Problemstellung: Handlungsreisender möchte bestimmte Orte jeweils einmal besuchen
- Gesucht: Kürzester Weg
- Extrem schwierig (optimal) zu lösen.
- Fundamentale Bedeutung für viele Anwendungen in Logistik, Verbindungen auf Leiterplatten



## Endlicher Automat (Finite State Machine)

Erkennung einer rationalen Zahl (z.B. -0,425) durch einen endlichen Automaten. Nicht gültig wären ("00,4" oder "-,3" oder "3,000" oder "01,")



⇒ Mehr dazu im Sommersemester!



## Mengentheoretische Begriffe und Notation anhand von Beispielen

- **Schnittmenge:** enthält alle Elemente die in beiden Mengen enthalten sind.

$$A \cap B = \{4, 5\}$$

- **Vereinigungsmenge:** enthält alle Elemente die in einer der beiden Mengen (oder beiden) enthalten sind.

$$A \cup B = \{1, 2, 3, 4, 5, 6, 7, 8\}$$

- **Differenzmenge:** enthält alle Elemente der ersten Menge die jedoch nicht in der zweiten Menge enthalten sind.

$$A \setminus B = \{1, 2, 3\}$$

$$B \setminus A = \{6, 7, 8\}$$

## Kardinalität

Unter *Kardinalität* versteht man die *Größe* oder *Mächtigkeit* einer Menge. Bei endlichen Mengen entspricht dies der Anzahl der darin enthaltenen Elemente. Die Schreibweise sind die vertikalen Striche.

Bemerkung: dies hat nichts mit dem Betrag (Zahlen) oder der Längen von Vektoren zu tun. Diese Begriffe sind für Mengen gar nicht sinnvoll anwendbar.

*Beispiel:*

$$F = \{2, 4, 6, 8, 11\}$$

$$|F| = 5$$

*Beispiel:*

$$|\{\}| = 0$$

## Mengentheoretische Begriffe und Notation anhand von Beispielen

- **(Echte) Teilmenge:** (alle Elemente einer Menge sind auch in einer anderen Menge enthalten, die Mengen sind *nicht* gleich)

$$E \subset (B \setminus A)$$

- **(Unechte) Teilmenge:** entweder (echte) Teilmenge, oder die Mengen sind gleich!

$$D \subseteq (B \setminus A)$$

Im konkreten Fall gilt nicht  $D \subset (A \setminus B)$ , da  $D$  keine echte Teilmenge von  $(B \setminus A)$  ist.

## Potenzmenge (\*)

Die Potenzmenge einer Menge enthält alle ihre Teilmengen als Elemente.

$$\mathcal{P}(X) = \{U \mid U \subseteq X\}$$

Die Potenzmenge einer Menge  $X$  wird entweder durch  $\mathcal{P}(X)$  oder oft auch als  $2^X$  angeschrieben.

*Beispiel:*

$$M = \{1, 2, 3\}$$

$$\mathcal{P}(M) = \{\emptyset, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$$

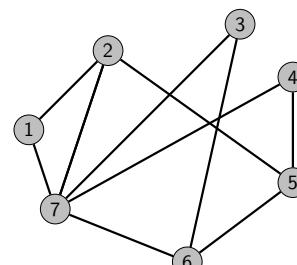


## Ungerichtete Graphen

### Definition (Ungerichteter Graph)

Ein (ungerichteter) Graph  $G = (V, E)$  besteht aus:

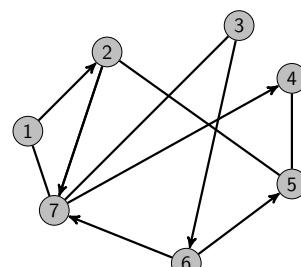
- einer (endlichen) Menge an *Knoten*  
 $V(G) = \{1, 2, \dots, n\}$ ; die Knoten werden meist mit Großbuchstaben oder Zahlen bezeichnet.
- einer (endlichen) Menge an *Kanten*  $E(G)$ ; Eine Kante zwischen  $i$  und  $j$  wird mit  $[i, j]$  oder  $\{i, j\}$  bezeichnet.



**Anmerkung:** Die Bezeichnung  $V$  für die Knotenmenge stammt von der englischen Bezeichnung "vertices" (pl.), bzw. "vertex" (sing.), was wörtlich übersetzt *Eckpunkt* bedeutet. Dennoch ist die Bezeichnung *Knoten* auf Deutsch gebräuchlicher. Die Kantenmenge  $E$  stammt vom englischen Begriff "edge(s)".

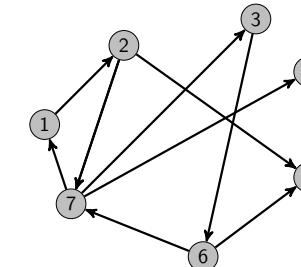
## Gemischte Graphen

- *Gemischte Graphen* enthalten sowohl gerichtete als auch ungerichtete Kanten



## Gerichtete Graphen

- Ungerichtete Kanten haben wir in eckigen Klammern notiert (z.B.  $[1, 3]$ ): Reihenfolge der Knoten spielt keine Rolle!
- Ein *gerichteter Graph* enthält nur *gerichtete Kanten*
- **Def. gerichtete Kante:**  $(i, j)$  ist eine gerichtete Verbindung vom Knoten  $i$  zum Knoten  $j$
- Eine gerichtete Kante wird auch *Bogen* oder *Pfeil* genannt. (engl.: *arc*)



## Adjazenz

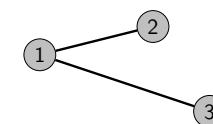
### Definition (Adjazente Knoten)

Zwei Knoten  $i$  und  $j$  sind adjazent, wenn eine Kante  $[i, j]$  existiert.

### Definition (Adjazente Kanten)

Zwei Kanten  $[i, j]$  und  $[j, k]$  sind adjazent wenn sie einen gemeinsamen Knoten  $j$  besitzen.

Beispiel: im folgenden Graph sind die Kanten  $[1, 2]$  und  $[1, 3]$  adjazent. Ebenso sind die Knoten 1 und 2, sowie die Knoten 1 und 3 adjazent.

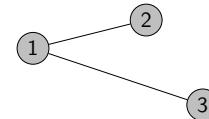


## Inzidenz

### Definition (Inzidenz)

Ein Knoten ist *inzident* mit einer Kante wenn der Knoten Endpunkt dieser Kante ist.

*Beispiel:*

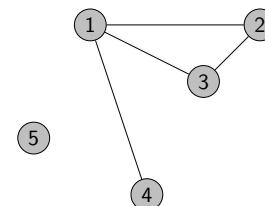


Hier ist die Kante  $[1, 2]$  und  $[1, 3]$  jeweils inzident mit dem Knoten 1.

## Isolierter Knoten

Einen Knoten  $v$  mit  $d(v) = 0$  nennt man *isolierten Knoten*.

*Beispiel:*  $d(5) = 0$



## Knotengrad

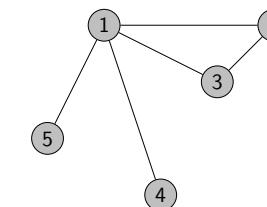
### Definition (Knotengrad)

Unter dem Grad (engl. degree)  $d(v)$  eines Knotens  $v \in V(G)$  versteht man die Anzahl der Kanten  $e \in E(G)$  die mit  $v$  verbunden sind.

### Definition (Endpunkt)

Ist  $d(v) = 1$  nennt man  $v$  einen *Endpunkt* des Graphen.

*Beispiel:*  $d(1) = 4, d(2) = 2, d(3) = 2, d(4) = 1, d(5) = 1;$

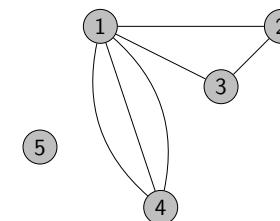


## Mehrfachkante

### Definition (Mehrfachkante)

Verlaufen zwischen zwei Knoten mindestens zwei Kanten, so heißt diese Menge von Kanten *Mehrfachkante* oder *Multikante*.

*Beispiel:* Mehrfachkante zwischen Knoten 1 und 4

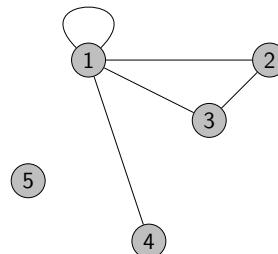


## Schlinge

### Definition (Schlinge)

Als *Schlinge* wird in einem Graphen eine Kante bezeichnet, die einen Knoten mit sich selbst verbindet.

**Beispiel:** Schlinge [1, 1]



## Beispiel

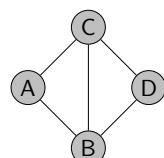
**Anmerkung:** Ein Graph  $G$  ist ausschließlich durch die Menge  $V$  der Knoten und die Menge  $E$  der Kanten bestimmt. Wie genau die Knoten und Kanten aufgezeichnet werden, ist nicht relevant!

**Beispiel:** Graph  $G = (V, E)$

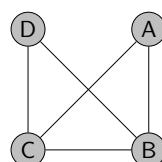
mit  $V = \{A, B, C, D\}$

und  $E = \{[A, B], [A, C], [B, C], [B, D], [C, D]\}$

Eine mögliche Darstellung des Graphen sieht wie folgt aus:



Eine andere Darstellung ist:



## Schlichter Graph

### Definition (Schlichter Graph)

Unter einem *schlichten Graphen* (auch: einfacher Graph) versteht man einen Graphen ohne Schlingen und Mehrfachkanten.

**Anmerkung:** Wir betrachten, sofern nicht anders angegeben, immer schlichte Graphen!



## Konstruktive Erweiterung von 4-regulären Graphen

### Übungsaufgabe

Finden Sie ein Verfahren (Algorithmus) zur Erweiterung von 4-regulären Graphen, d.h. weitere Knoten sollen hinzugefügt werden, und der Graph 4-regulär bleiben!

POS (Theorie)

Spezielle Graphen

5 / 23

## Reguläre Graphen

### Aufgabe: 3-regulärer Graph

Konstruieren Sie einen schlichten und zusammenhängenden 3-regulären Graphen mit 5 Knoten.

## Konstruktive Erweiterung von 3-regulären Graphen

### Übungsaufgabe

Wie kann dieser Algorithmus zur konstruktiven Erweiterung von 3-regulären Graphen angepasst werden?

POS (Theorie)

Spezielle Graphen

6 / 23

## Vollständiger Graph (1)

### Definition (Vollständiger Graph $K_n$ )

Ein Graph  $G$  mit  $n = |V|$  Knoten heißt vollständiger Graph  $K_n$ , wenn er regulär vom Grad  $n - 1$  ist.

**Bemerkung:** In einem vollständigen Graphen sind somit alle Knoten direkt miteinander verbunden.

**Eigenschaft:** Jeder  $K_n$  hat genau  $\frac{n \cdot (n-1)}{2}$  Kanten:

$$|E| = \frac{n \cdot (n-1)}{2}$$

POS (Theorie)

Spezielle Graphen

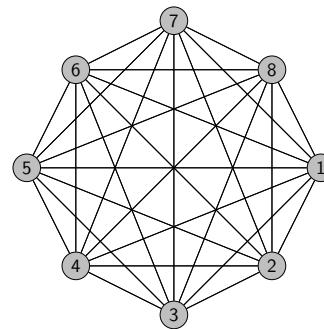
7 / 23

POS (Theorie)

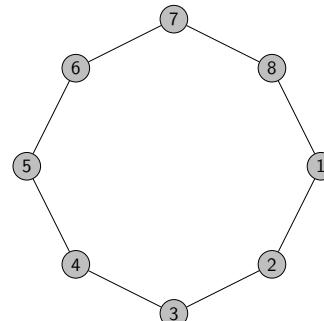
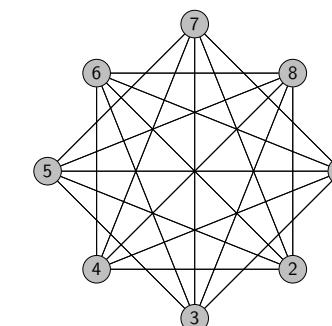
Spezielle Graphen

8 / 23

## Vollständiger Graph (2)

**Beispiel:** Vollständiger Graph  $G = K_8$ 

## Komplementärgraph (2)

**Beispiel:** Graph  $G$ **Beispiel:** der komplementäre Graph  $G'$  zu  $G$  sieht folgendermaßen aus:

## Bipartite Graphen (1)

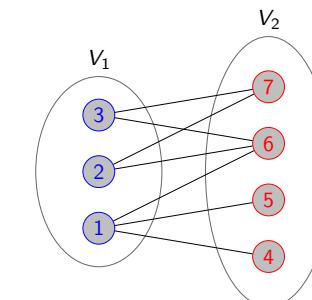
### Definition (Bipartiter Graph)

Ein Graph  $G = (V, E)$  heißt *bipartit* wenn eine Partitionierung der Knoten  $V$  in zwei disjunkte Teilmengen  $V_1$  und  $V_2$  existiert, sodaß für jede Kante  $[i, j] \in E$  gilt, dass entweder  $i \in V_1$  und  $j \in V_2$ , oder andererseits  $i \in V_2$  und  $j \in V_1$ .

**Bemerkung:** Man stellt sich am besten eine Knotenmenge auf der linken Seite, und eine auf der rechten Seite vor. Kanten verlaufen nur von links nach rechts (und umgekehrt), aber niemals zwischen Knoten einer der beiden Mengen.

## Bipartite Graphen (2)

### Beispiel:



## Übungsaufgaben

### Übungsaufgaben

- 2.1 Konstruieren sie einen schlichten Graphen mit 12 Knoten der jeweils 3 Knoten vom Grad 3, 4, 5 und 6 enthält. Knoten gleichen Grades dürfen nicht adjazent sein.
- 2.2 Konstruieren Sie einen schlichten, zusammenhängenden Graphen mit 14 Knoten, der jeweils
  - 4 Knoten vom Grad 6,
  - 2 Knoten vom Grad 5,
  - 4 Knoten vom Grad 4,
  - 2 Knoten vom Grad 3, und
  - 2 Knoten vom Grad 2 hat.

Knoten gleichen Grades dürfen nicht adjazent sein, Knoten vom Grad 2 sollen nicht adjazent zu Knoten vom Grad 3 und Grad 4 sein.

## Teilgraphen (1)

### Definition (Teilgraph)

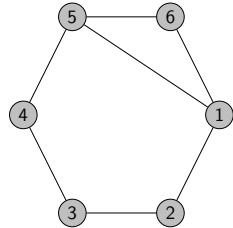
Ein Graph  $G' = (V', E')$  heißt *Teilgraph*, *Subgraph* oder *Untergraph* von  $G = (V, E)$ , wenn  $V' \subseteq V$  und  $E' \subseteq E$  (und für alle  $[i, j] \in E'$  gilt  $i \in V'$  und  $j \in V'$ )

### Definition (Obergraph)

Ein Graph  $G$  ist *Obergraph* eines Graphen  $G'$  wenn  $G'$  Teilgraph von  $G$  ist.

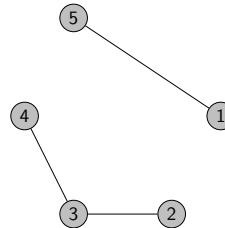
## Teilgraphen (2)

**Beispiel:** Graph  $G$



POS (Theorie)

**Beispiel:** Teilgraph  $G'$  von  $G$



Spezielle Graphen

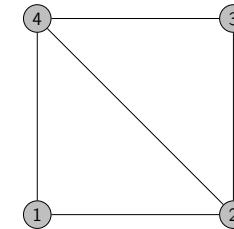
## Unterteilungsgraph

**Definition (Unterteilungsgraph)**

Ein Unterteilungsgraph ist ein Graph, der durch Kantenunterteilung aus einem anderen Graph entstanden ist.

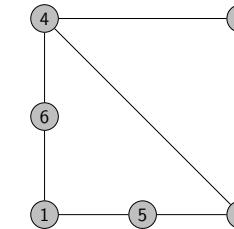
**Anmerkung:** In einem Unterteilungsgraphen wird "auf einer Kante" ein neuer Knoten eingefügt. Formal wird eine Kante  $[u, v]$  entfernt, ein neuer Knoten  $w$  dem Graphen hinzugefügt, und dann Kanten  $[u, w]$  und  $[w, v]$  hinzugefügt.

**Beispiel:** Graph  $G$



POS (Theorie)

**Beispiel:** Unterteilungsgraph von  $G$



Spezielle Graphen

18 / 23

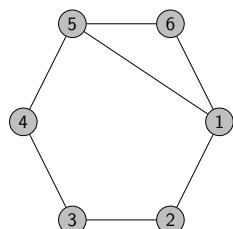
## Spannender Teilgraph

**Definition (Spannender Teilgraph)**

Ein Teilgraph  $G' = (V', E')$  heißt *spannender Teilgraph* von  $G = (V, E)$  wenn  $V' = V$  und  $E' \subseteq E$ .

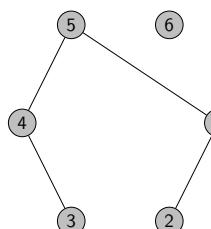
**Bemerkung:** Ein spannender Teilgraph  $G'$  besitzt also alle Knoten von  $G$ , jedoch nicht unbedingt alle Kanten von  $G$  (eventuell auch gar keine).

**Beispiel:** Graph  $G$



POS (Theorie)

**Beispiel:** Teilgraph  $G'$  von  $G$



Spezielle Graphen

19 / 23

**Definition (Gesättigter Teilgraph)**

Sei  $G' = (V', E')$  ein Teilgraph von  $G = (V, E)$ . Enthält  $E'$  alle Kanten aus  $E$  deren Endpunkte in  $V'$  liegen, so sagt man, dass  $G'$  von  $V' \subseteq V$  in  $G$  aufgespannt wird, oder dass  $G'$  ein *gesättigter Teilgraph* von  $G$  ist.

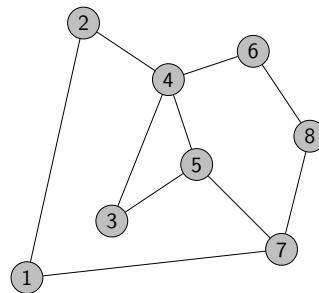
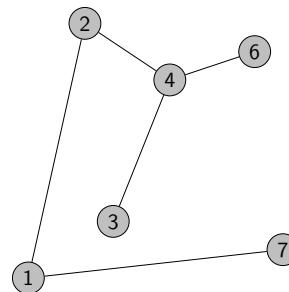
**Bemerkung:** Ein gesättigter Teilgraph enthält also bezüglich einer Teilmenge von Knoten alle Kanten des ursprünglichen Graphen. Es fehlen also alle Kanten, die zu den fehlenden Knoten inzident sind.

POS (Theorie)

Spezielle Graphen

20 / 23

## Gesättigter Teilgraph (2)

Graph  $G$  mit 8 KnotenGesättigter Teilgraph  $G'$  mit 6 Knoten von  $G$ 

### Übungsaufgaben

**2.3** Konstruieren Sie einen schlichten 4-regulären Graphen  $G$  mit  $|V| = 8$  mit Knotenbeschriftungen (A, B, C, ...).

- Zeichnen Sie einen spannenden Teilgraphen  $G'$  von  $G$  mit  $|E'| = 6$ .
- Zeichnen Sie einen gesättigten Teilgraphen  $G''$  von  $G$  mit  $|V''| = 5$ .

**2.4** Hintergrund: Ein Springer soll auf einem Schachbrett so gezogen werden, dass er jedes Feld genau einmal besucht, und wieder zum Ausgangspunkt zurückkommt. Wie sieht der Graph  $G$  aus, der dieses Problem beschreibt, also der Graph der alle möglichen Züge des Springers auf dem Schachbrett enthält. Fragestellung: Wieviele Kanten hat der Graph?

### Aufgabe 2.5: In einer Kleinstadt gibt es 10 Märkte...

- ① Vom Bäckermarkt führen breite Straßen in alle vier Himmelsrichtungen zu vier weiteren Märkten. Diese sind wiederum ringförmig miteinander verbunden.
- ② In den Fischmarkt münden vier Straßen, davon aber keine vom Pferdemarkt.
- ③ Nur an zwei Märkten münden mehr als drei Straßen.
- ④ Es gibt keine Sackgassen.
- ⑤ Der Schweinemarkt, Milchmarkt, Naschmarkt und Heumarkt sind jeweils nicht direkt durch eine Straße miteinander verbunden.
- ⑥ Vom Getreidemarkt erreicht man direkt den Heumarkt, den Rindermarkt und den Milchmarkt.
- ⑦ Vom Kohlmarkt führen drei Straßen zum Heumarkt, Rindermarkt und Fischmarkt, die jeweils aber nicht direkt durch eine Straße miteinander verbunden sind.

**Konstruieren Sie einen Graphen, der einer Skizze des Stadtplanes der Kleinstadt nach den obigen Angaben entspricht.**

# Graphentheorie: Planare Graphen

## Programmieren und Software-Engineering Theorie

2. September 2025

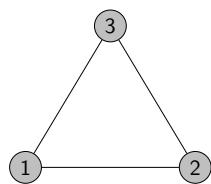
POS (Theorie)

Planare Graphen

1 / 7

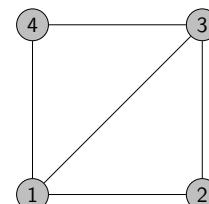
## Planare Graphen (2)

**Beispiel:** Graph mit 3 Knoten und 3 Kanten:



Wir erhalten 3 Knoten - 3 Kanten + 2 Flächen = 2

**Beispiel:** Graph mit 4 Knoten und 5 Kanten:



Wir erhalten 4 Knoten - 5 Kanten + 3 Flächen = 2

## Planare Graphen (1)

### Definition (Planarer Graph)

Ein *planarer* oder *plättbarer* Graph kann in der Ebene ohne überkreuzende Kanten gezeichnet werden.

### Eulerscher Polyedersatz

Der *Eulerscher Polyedersatz* lautet auf zusammenhängende und planare Graphen übertragen

$$|V| - |E| + |F| = 2,$$

wobei  $|F|$  die Anzahl der "Flächen" (engl. Facets) bedeutet.

**Wichtig:** Die "äußere" Fläche (alles um den Graphen herum) ist mitzuzählen!

POS (Theorie)

Planare Graphen

3 / 7

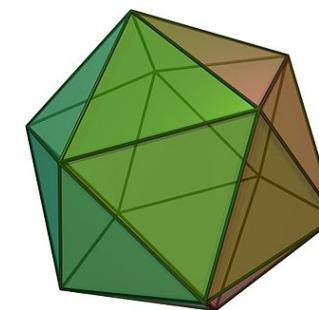
POS (Theorie)

Planare Graphen

4 / 7

## Planare Graphen (3)

Der Eulersche Polyedersatz gilt allgemein für konvexe Polyeder, z.B. dem *Ikosaeder*

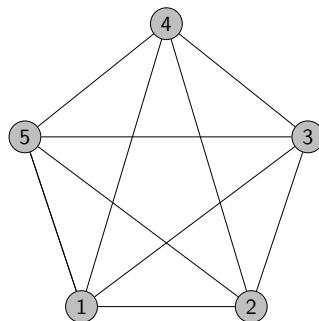


## Satz von Kuratowski

### Satz von Kuratowski

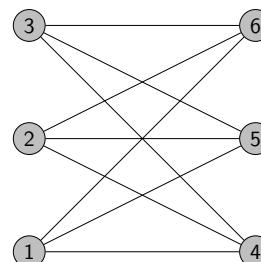
Ein Graph ist genau dann planar, wenn er keinen Teilgraphen besitzt, der ein Unterteilungsgraphen des  $K_5$  oder  $K_{3,3}$  ist.

Vollständiger Graph  $K_5$ :



POS (Theorie)

Vollständiger bipartiter Graph  $K_{3,3}$ :



Planare Graphen

5 / 7

## Beweis: $K_5$ nicht planar (\*)

**Beweis (indirekt):** für jeden planaren Graphen gilt der Eulersche Polyedersatz  $|V| - |E| + |F| = 2$ .

Wir nehmen also an, dass der  $K_5$  planar ist, und somit die Eulersche Formel gilt.<sup>1</sup> Es gilt  $|V| = 5$ ,  $|E| = 10$  und somit  $|F| = 7$ . Jede Fläche wird von mindestens drei Kanten begrenzt. Andererseits kommt jede Kante genau zwei mal als Begrenzung vor. Es gilt also:

$$|E| \geq \lceil \frac{3 \cdot |F|}{2} \rceil.$$

Setzen wir nun die konkreten Werte ein, erhalten wir  $|E| \geq 11$ , was im Widerspruch zu  $|E| = 10$  steht.

Somit ist der  $K_5$  nicht planar. □

<sup>1</sup> Vergleiche mit Aussagenlogik:  $\frac{\neg a \rightarrow b, \neg b}{a}$ . In diesem Fall kann der Graph planar sein oder eben nicht. Es gibt keine weitere Möglichkeit. Wenn somit die Annahme er sei planar mit korrekten logischen Umformungen zu einem Widerspruch führt, dann muss genau das Gegenteil der Annahme wahr sein: er ist nicht planar.

POS (Theorie)

Planare Graphen

6 / 7

## Beweis: $K_{3,3}$ nicht planar (\*)

**Beweis (indirekt):** Der Beweis verläuft analog zum vorigen Beweis. Es gilt  $|V| = 6$ ,  $|E| = 9$  und somit muss gelten  $|F| = 5$ .

Im bipartiten Graphen muss jedoch jede Fläche von mindestens vier Kanten begrenzt werden.  $|E| \geq \frac{4 \cdot |F|}{2}$  führt für den  $K_{3,3}$  somit zu  $|E| \geq 10$ . Dies steht wieder im Widerspruch zu  $|E| = 9$ . □

Wir haben bisher gezeigt:

**Ist  $G$  planar  $\rightarrow G$  enthält nicht  $K_{3,3}$  oder  $K_5$ .**

Um den Satz von Kuratowski tatsächlich zu beweisen, muss auch die "andere Richtung" bewiesen werden:

**Ist  $G$  planar  $\leftarrow G$  enthält nicht  $K_{3,3}$  oder  $K_5$  (bzw. deren Unterteilungsgraphen)**

Dies ist jedoch etwas schwieriger. Der Beweis kann über vollständige Induktion geführt werden. Erst dann ist tatsächlich gezeigt, dass gilt:

**Ist  $G$  planar  $\leftrightarrow G$  enthält nicht  $K_{3,3}$  oder  $K_5$  (bzw. deren Unterteilungsgraphen)**

POS (Theorie)

Planare Graphen

7 / 7

# Graphentheorie: Kantenfolgen, Eulersche und Hamiltonsche Graphen

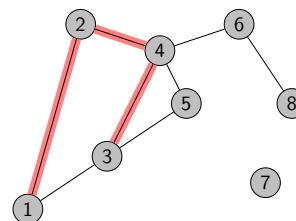
## Programmieren und Software-Engineering Theorie

2. September 2025

## Kantenfolgen

**Bemerkung:** In einer Kantenfolge können bestimmte Kanten auch mehrfach vorkommen!

**Beispiel:** Kantenfolge  $K = [1, 2], [2, 4], [4, 3], [3, 4], [4, 2]$



**Beispiel:** Keine Kantenfolgen wären  $[1, 2], [2, 3]$  oder  $[1, 2], [4, 3]$ .

## Kantenfolgen

### Definition (Kantenfolge)

Eine Kantenfolge von  $v_1$  nach  $v_n$  (aus  $V$ ) ist eine endliche Folge von Kanten  $[v_1, v_2], [v_2, v_3], \dots, [v_{n-1}, v_n]$ , sodass jeweils zwei aufeinanderfolgende Kanten adjazent sind.

### Definition (Offene Kantenfolge)

Eine Kantenfolge  $[v_1, v_2], \dots, [v_{n-1}, v_n]$  heißt *offen*, falls  $v_1 \neq v_n$ .

### Definition (Geschlossene Kantenfolge)

Eine Kantenfolge  $[v_1, v_2], \dots, [v_{n-1}, v_n]$  heißt *geschlossen*, falls  $v_1 = v_n$ .

## Kantenzug

### Definition (Kantenzug)

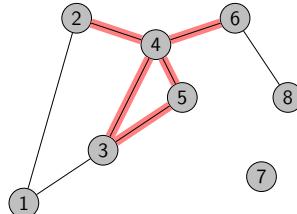
Ein *Kantenzug* ist eine Kantenfolge bei der alle Kanten paarweise verschieden sind.

**Anmerkung 1:** “paarweise verschieden” ... typische (mathematische) Formulierung, die ausdrückt, dass alle Elemente unterschiedlich sind, d.h. es keine zwei gleichen Elemente gibt!

**Anmerkung 2:** Analog zur Kantenfolge existieren *offene* und *geschlossene* Kantenzüge

## Kantenzug

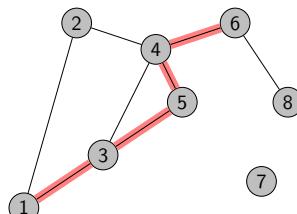
**Beispiel:** Kantenzug  $K = [2, 4], [4, 5], [5, 3], [3, 4], [4, 6]$



**Anmerkung:** der Knoten 4 wird in diesem Kantenzug mehrfach besucht!

## Weg, bzw. Pfad (2)

**Beispiel:** der Pfad  $P(1, 6) = [1, 3], [3, 5], [5, 4], [4, 6]$  mit  $|P(1, 6)| = 4$  ist rot markiert



Allgemein können mehrere (verschiedene) Wege von  $s$  nach  $t$  existieren.

### Definition (Kürzester Weg/Pfad)

Jene Wege von  $s$  nach  $t$  mit minimaler Länge  $|P(s, t)|$  werden **kürzeste Wege** (oder kürzeste Pfade) genannt.

## Weg, bzw. Pfad (1)

### Definition (Weg, Pfad)

Ein Weg oder Pfad (engl. path)  $P(s, t)$  ist ein offener Kantenzug von  $s \in V$  nach  $t \in V$  bei dem alle Knoten paarweise verschieden sind.

### Achtung

In der Literatur wird oft *Weg* so definiert, daß gleiche Knoten im Kantenzug vorkommen dürfen. bei einem *Pfad* müssen dann die Knoten voneinander verschieden sein. Allerdings existieren auch andere Definitionen in welchen dieser "Weg" als *Pfad* bezeichnet wird, und der zuvor genannte "Pfad" als *elementarer Pfad*.

### Definition (Weglänge)

Die Länge  $|P(s, t)|$  ist die Anzahl der Kanten in  $P(s, t)$

## Kreis, Zyklus

### Definition (Zyklus)

Ein *Zyklus*  $(v_1, v_2, \dots, v_{n-1}, v_n)$  ist ein geschlossener Kantenzug, d.h.  $v_1 = v_n$ .

### Definition (Kreis)

Ein *Kreis* ist ein Zyklus bei dem alle Knoten, bis auf den Start- und Endknoten verschieden sind.

### Achtung

Auch die Begriffe Kreis und Zyklus sind leider in der Literatur nicht immer einheitlich definiert.

## Eulersche Linie, Euler-Zyklus

### Definition (Eulersche Linie)

Eine *Eulersche Linie* ist ein Kantenzug, der alle Kanten des Graphen genau einmal enthält.

**Bemerkung:** Man beachte, daß hier Knoten mehrfach durchlaufen werden können!

### Definition (Euler-Zyklus)

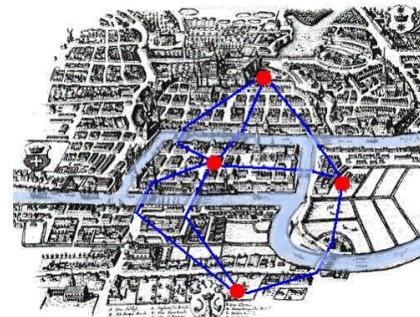
Bei einer geschlossenen *Eulerschen Linie*, bzw. ein **Euler-Zyklus** sind Start- und Endknoten gleich!

### Achtung

Oft wird von Eulerschen Kreisen oder Eulerschen Zyklen gesprochen. Ob diese Bezeichnungen tatsächlich zutreffend sind hängt wiederum von den konkreten Definitionen von diesen Begriffen ab, also insbesondere ob sie das mehrfache Durchlaufen von Knoten erlauben.

## Königsberger Brückenproblem

Wir haben nun die allgemeine Lösung des eingangs erwähnten Königsberger Brückenproblems



**Anmerkung:** Ein Graph besitzt genau dann eine offene Eulersche Linie, wenn alle Knoten bis auf zwei einen geraden Grad aufweisen.

## Eulersche Graphen

### Definition (Eulerscher Graph)

Besitzt ein Graph einen *Euler-Zyklus*, so bezeichnet man ihn als *Eulersch*.

### Satz von Euler-Hierholzer

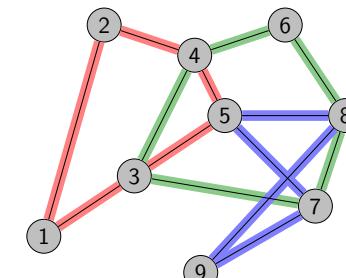
Sei  $G$  ein zusammenhängender Graph. Dann sind folgende Aussagen äquivalent:

- $G$  ist *Eulersch*
- Jeder Knoten in  $G$  hat geraden Grad
- Die Kantenmenge von  $G$  ist die Vereinigung aller Kanten von paarweise disjunkten Zyklen <sup>a</sup>

<sup>a</sup>disjunkt bezüglich ihrer Kanten!

## Euler-Zyklus

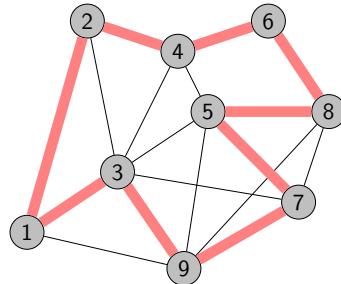
**Beispiel:** Beispiel, indem die Kantenmenge von  $G$  die Vereinigung von paarweise disjunkten Zyklen ist (und damit  $G$  eulersch)





## Hamiltonkreis

**Beispiel:**



POS (Theorie)

Kantenfolgen

17 / 30

## Hamiltonkreis (\*)

Ein kurzer Blick hinter die Kulissen: Komplexität des Problems, Komplexitätstheorie:

- Das Hamiltonkreis-Problem liegt in der Komplexitätsklasse **NP**–vollständig:
  - NP: “*nondeterministic polynomial*”
  - Es kann nicht in *polynomieller Zeit* in Abhängigkeit von den Eingabedaten entschieden werden ob eine Lösung existiert
  - Ist enthalten in Liste der 21 klassischen *NP*-vollständigen Probleme (Richard Karp, 1972)

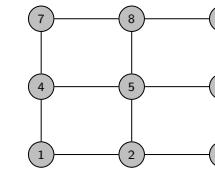
## Hamiltonscher Graph

### Definition (Hamiltonscher Graph)

Einen Graphen der einen Hamiltonkreis enthält nennt man *Hamiltonschen Graph*.

- Nicht alle Graph sind Hamiltonsch!
- Es existiert keine einfache Charakterisierung ob ein Graph Hamiltonsch ist!**
- Im Gegensatz zum leicht lösbar Problem des Euler-Zyklus, ist das Hamiltonkreisproblem im Allgemeinen sehr schwierig zu lösen.

**Beispiel:**



Dieser Graph ist *nicht* Hamiltonsch!

POS (Theorie)

Kantenfolgen

18 / 30

## Exkurs: Komplexitätstheorie (1) (\*)

- Viele Probleme können *effizient* berechnet werden
- Die *Laufzeit* des Algorithmus wächst mit zunehmend großen Eingabedaten nur moderat.
- Sei  $n$  die Größe der Eingabedaten, dann läuft das Lösungsverfahren (Algorithmus) in  $O(n^k)$  Schritten ab
  - $O(n)$  ist zu lesen als: in der Größenordnung von  $n$
  - $n^k$  bedeutet hier eine beliebige Potenz von  $n$  (Exponent  $k$ )
  - Beispiel: Wir sortieren  $n = 10$  Zahlen aufsteigend  $\Rightarrow$  dafür benötigen wir  $O(n^2)$ , also ca. 100 Schritte<sup>1</sup>
- Alle Probleme wo es einen derartigen Exponenten  $k$  gibt liegen in der **Komplexitätsklasse P**
  - Sie sind *polynomiell* berechenbar

<sup>1</sup>Sortieren geht auch “besser”!

POS (Theorie)

Kantenfolgen

19 / 30

POS (Theorie)

Kantenfolgen

20 / 30

## Exkurs: Komplexitätstheorie (2) (\*)

- Jetzt stark vereinfacht: bei bestimmten anderen Problemen weiß man, dass sie in  $O(k^n)$  berechenbar sind!
- Man beachte:  $n$  steht jetzt selbst im Exponenten  $\Rightarrow$  exponentielles Wachstum der Laufzeit
- Die Folge ist, das Programm wird nicht in sinnvoll kurzer Zeit fertig.
- Man nennt dies die **Komplexitätsklasse NP**
- Bei bestimmten Problemen weiß man, dass sie in  $NP$  liegen, und zu den schwierigsten Problemen in  $NP$  zählen  $\Rightarrow$  **Komplexitätsklasse NP-vollständig**

## Exkurs: Komplexitätstheorie (3) (\*)

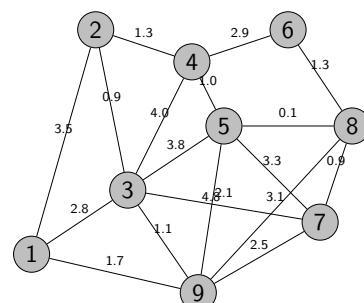
- Diese  $NP$ -vollständigen Probleme sind alle aufeinander abbildbar (d.h. hat man einen Algorithmus für ein Problem gefunden, kann man damit grundsätzlich alle derartigen Probleme lösen).
- Großes Problem: Es ist nicht bewiesen, dass  $P \neq NP$
- Dies ist eines der großen (wahrscheinlich das größte) ungelöste Problem der theoretischen Informatik.
- Nahezu alle Forscher sind jedoch überzeugt, dass tatsächlich  $P \neq NP$ , aber es fehlt der Beweis.
- Wäre  $P = NP$ , so könnten es plötzlich effiziente Algorithmen für schwierige aber praxisrelevante Aufgabestellungen geben!

## (Kanten-)Gewichtete Graphen

## Definition ((Kanten-)Gewichteter Graph)

Sei  $w : E \rightarrow \mathbb{R}$  eine Abbildung die jeder Kante  $e \in E$  eine reelle Zahl zuordnet. Man bezeichnet  $w$  als die Gewichtsfunktion.

**Beispiel:** mit  $w(e) \geq 0, \forall e \in E$ :



## Kürzester Hamiltonkreis (\*)

- Betrachten wir nun einen vollständigen Graphen mit Kantengewichten  $w(e) \geq 0$  für alle  $e \in E$ .
- Existiert hier ein Hamiltonkreis?
- In diesem Graphen existiert auf jeden Fall ein Hamiltonkreis! Es gibt tatsächlich sogar

$$\frac{(|V| - 1)!}{2}$$

verschiedene Hamiltonkreise!

- Alle Permutationen der Knoten ergeben einen gültigen Hamiltonkreis!

## Kürzester Hamiltonkreis (\*)

- Im Gegensatz zum **Entscheidungsproblem** "gibt es einen Hamiltonkreis" wollen wir nun das **Optimierungsproblem** betrachten: Finde hinsichtlich der Kantengewichte  $w(e)$  den **kürzesten Hamiltonkreis  $H$** , also

$$\min_H \sum_{e \in H} w(e).$$

- Da das Entscheidungsproblem *NP*-vollständig ist, ist das Optimierungsproblem **NP-schwierig**
- Diese Variante des Problems nennt man **Travelling Salesman Problem (TSP)**
- Wie schon in der Einleitung erwähnt, ist das TSP von immens großer Bedeutung für Wirtschaft, Industrie und Informatik

## Übungsaufgaben

### Aufgabe 4.1

Ein Bauer steht mit einem Wolf, einer Ziege und einem Kohlkopf an einem Flussufer und möchte mit einem Boot an die andere Seite des Flusses gelangen. Das Boot ist jedoch nur groß genug um ein weiteres Tier, bzw. den Kohlkopf mitzunehmen.

Weiters gelten die folgenden Einschränkungen: ist die Ziege alleine mit dem Kohlkopf, so frisst sie diesen! Ebenso frisst der Wolf die Ziege, wenn er alleine mit ihr ist. Wie kann der Bauer schnellstmöglich mit allen Dreien (lebendig!) an das andere Ufer gelangen?

**Aufgabe:** Modellieren Sie die Aufgabenstellung graphentheoretisch. Was entspricht den Knoten, was entspricht den Kanten? Sie sollen also *nicht* die Lösung finden, sondern eine korrekte Modellierung angeben!

## Travelling Salesman Problem (\*)

**Beispiel:** Beliefere ausgehend von einem Lagerstandort 50 Kunden in Wien Modellierung:

- Die Kunden werden durch Knoten  $V$  modelliert
- Wir betrachten den vollständigen Graphen  $G$  bezüglich  $V$
- Kantengewichte  $w(e_{ij})$  ergeben sich durch Berechnung der kürzesten Wege im Straßengraphen vom Kunden  $i$  zum Kunden  $j$
- Lösung: **kürzester Hamiltonkreis** in  $G$



## Übungsaufgaben

### Aufgabe 4.2

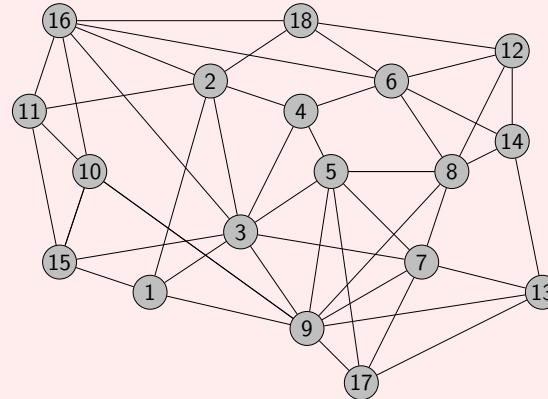
Vier Personen wollen eine Höhle verlassen. Sie haben nur noch eine Lampe, die aber nicht mehr lange brennt. Es können immer nur maximal zwei Personen mit einer Lampe gehen, für drei Personen wäre das Licht nicht ausreichend hell.

Die Personen weisen jedoch stark unterschiedliche Fähigkeiten im Klettern auf. Die erste Person (A) benötigt 1 Zeiteinheit (ZE) für den Weg, die zweite Person (B) 2 ZE, die dritte (C) 5 ZE und die vierte Person (D) 10 ZE.

Wie können die vier Personen am schnellsten die Höhle verlassen? Ziel der Aufgabenstellung ist wiederum die korrekte *Modellierung* und nicht primär die Lösung!

**Aufgabe 4.3**

Berechnen Sie einen Euler-Zyklus für den folgenden Graphen. Wenn der Graph nicht Eulersch ist, dann ändern Sie eine Kante (hinzufügen, entfernen), sodaß der Graph dann Eulersch ist.



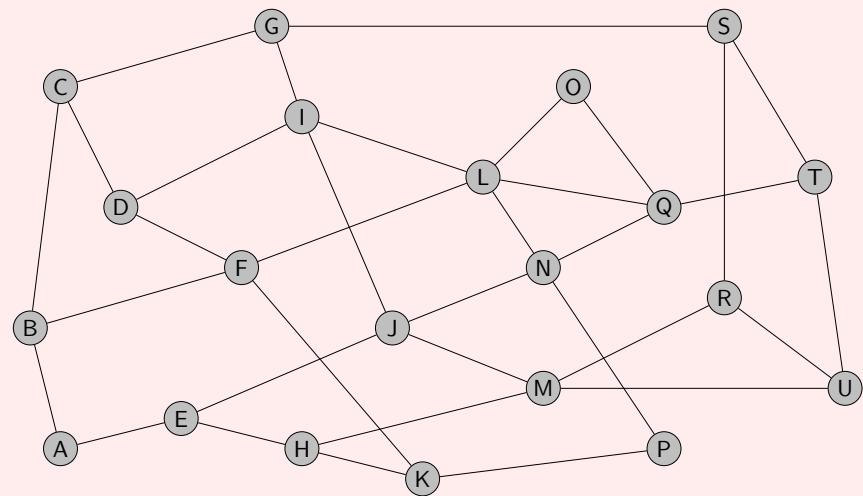
POS (Theorie)

Kantenfolgen

29 / 30

**Aufgabe 4.4**

Geben Sie einen Hamilton-Kreis für den folgenden Graphen an.



POS (Theorie)

Kantenfolgen

30 / 30

# Graphentheorie: Eigenschaften

## Programmieren und Software-Engineering Theorie

2. September 2025

## Distanz (gewichteten Graphen)

In einem gewichteten Graphen gilt für die Distanz

$$d(u, v) = \min_{P(u,v)} \sum_{e \in P(u,v)} w(e),$$

also die Summe der Kantengewichte über den kürzesten Pfad von  $u$  nach  $v$ . Die folgenden Definitionen können für beide Varianten (ungewichtete und gewichtete Graphen) verwendet werden.

## Distanz (ungewichteter Graph)

### Definition (Distanz)

Die Distanz (Abstand) zweier Knoten  $u$  und  $v$  in einem Graphen ist die Länge des kürzesten Weges  $P(u, v)$ , also

$$d(u, v) = \min_{P(u,v)} |P(u, v)|.$$

- Falls kein Weg zwischen  $u$  und  $v$  existiert, so gilt

$$d(u, v) = \infty$$

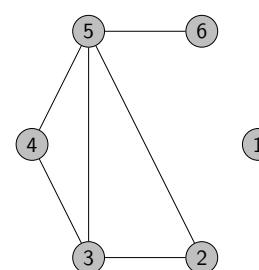
- Der Abstand eines Knoten zu sich selbst ist 0, also

$$d(u, v) = 0 \leftrightarrow u = v.$$

- In ungerichteten Graphen gilt  $d(u, v) = d(v, u)$  für alle  $u, v \in V$ .

## Distanz (Beispiel)

### Graph $G$ mit 6 Knoten



$$d(1, 1) = 0$$

$$d(1, 2) = \infty$$

$$d(2, 6) = 2$$

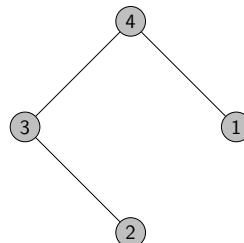
$$d(3, 4) = 1$$

## Exzentrizität

### Definition (Exzentrizität)

Die Exzentrizität  $ex(v)$  eines Knoten  $v \in V$  eines ungerichteten, zusammenhängenden Graphen  $G = (V, E)$  ist die Distanz zum entferntesten Knoten von  $v$  in  $G$ .

Zur Ermittlung der Exzentrizitäten muss der maximale Abstand jedes Knoten im Graph  $G$  bestimmt werden.

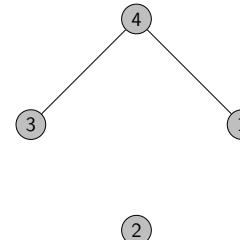


POS (Theorie)

Knoten	Exzentrizität	Entferntester Knoten
1	3	2
2	3	1
3	2	1
4	2	2

Eigenschaften

5 / 20



POS (Theorie)

In einem nicht zusammenhängenden Graphen gibt es zu jedem Knoten mindestens einen Knoten der nicht erreichbar ist.

Der Wert für die Exzentrizität ist in diesem Fall für alle Knoten mit  $\infty$  definiert.

$$\forall v \in V : ex(v) = \infty$$

## WH: Chinese Postman Problem

- (Geschlossene) Eulersche-Linien haben große praktische Bedeutung für
  - Abfallentsorgung
  - Schneeräumung
  - Briefzustellung
- Ein Straßengraph ist jedoch nicht unbedingt *Eulersch*
- Ebenso spielt die Länge der einzelnen Straßen eine Rolle!
- Chinese Postman Problem:**
  - Benannt nach chinesischem Mathematiker Mei Ko Kwan (\*1934)
  - Gegeben:** gewichteter Graph (nicht notwendigerweise Eulersch)
  - Ziel:** Finde kürzesten Zyklus im Graphen, der jede Kante *mindestens* einmal enthält

POS (Theorie)

Eigenschaften

7 / 20

## Exzentrizität (2)

POS (Theorie)

Eigenschaften

6 / 20

## Einschub: Paarung/Matching

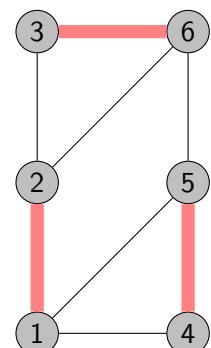
- Bildung von 2-elementigen Teilmengen ("Paaren") von Knoten

### Definition (Paarung/Matching)

Gegeben sei ein Graph  $G = (V, E)$ . Eine Menge  $M \subseteq E$  heißt *Matching* (oder Paarung), wenn kein Knoten aus  $V$  zu mehr als einer Kanten aus  $M$  inzident ist.

- Paarungen heißen **vollständig**, wenn alle Knoten gepaart sind.
- In gewichteten Graphen sind meist Paarungen minimalen oder maximalen Gewichts von Interesse, wobei

$$w(M) = \sum_{e \in M} w(e)$$



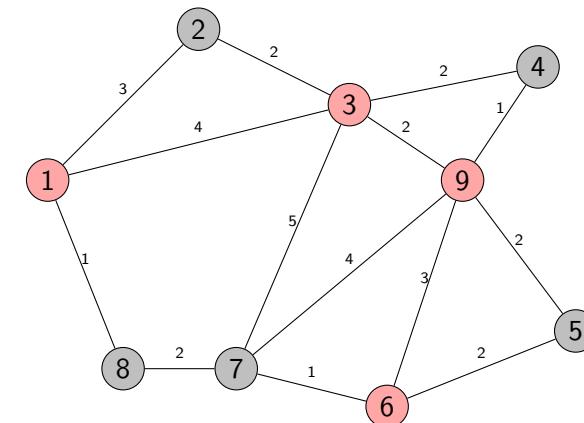
Beispiel für ein (vollständiges) Matching in einem ungewichteten Graphen

8 / 20

## Chinese Postman Problem

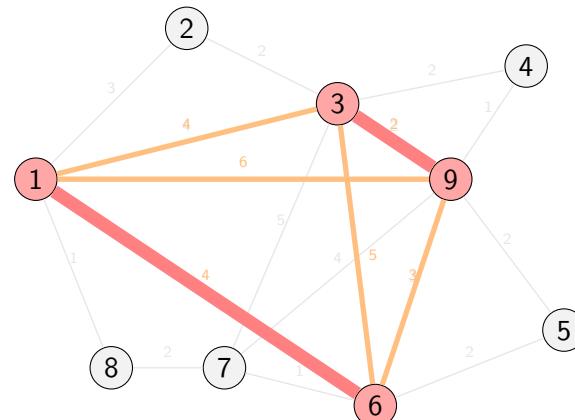
- Wähle alle Knoten mit ungeradem Grad
- Bilde vollständigen Graphen aus diesen Knoten
- Jeder Kante werden Kosten zugeordnet, die der Distanz der Knoten im ursprünglichen Graphen entsprechen
- Bilde kostenminimale *Paarung* von Knoten (ohne Details)
- Dupliziere Kanten entlang der kürzesten Wege der gepaarten Knoten im ursprünglichen Graphen
- Der resultierende Graph ist nun Eulersch, der Euler-Zyklus entspricht der kürzest möglichen geschlossenen Kantenfolge im ursprünglichen Graphen, die alle Kanten mindestens einmal enthält!

## Beispiel: Chinese Postman Problem



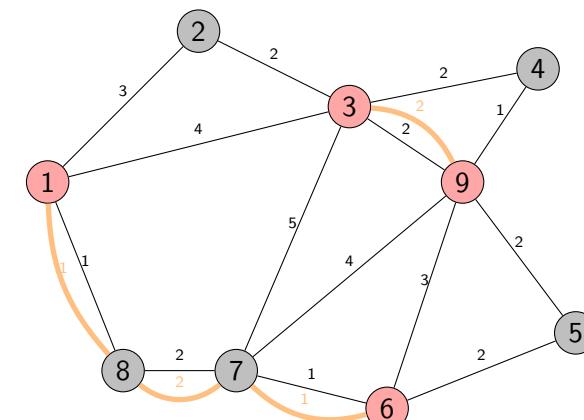
Die **markierten Knoten** haben ungeraden Grad, wodurch der Graph insgesamt *nicht* Eulersch ist.

## Beispiel: Chinese Postman Problem



Bezüglich der "ungeraden" Knoten wird ein **vollständiger Graph** gebildet, dessen Kantengewichte den Distanzen im ursprünglichen Graphen entsprechen. ⇒ Bildung von **kostenminimalem Matching**.

## Beispiel: Chinese Postman Problem



Alle Kanten entlang der kürzesten Pfade zwischen den gepaarten Knoten werden nun im ursprünglichen Graphen verdoppelt, wodurch dieser nun Eulersch wird!

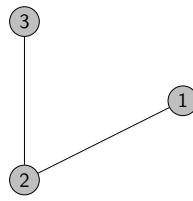
## Durchmesser

### Definition (Durchmesser)

Der Durchmesser  $dm(G)$  eines Graphen  $G$  ist das Maximum der Exzentrizitäten aller Knoten von  $G$ .

Wenn  $G$  nicht zusammenhängend ist, so gilt  $dm(G) = \infty$ .

**Beispiel:** Graph mit  $dm(g) = 2$



POS (Theorie)

Knoten	Exzentrizität
1	2
2	1
3	2

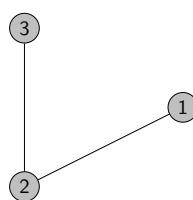
**Anmerkung:** Das Maximum aller Exzentrizitäten im Beispielgraphen beträgt 2. Somit beträgt der Durchmesser  $dm(G) = 2$

Eigenschaften

13 / 20

## Radius (2)

**Beispiel:** Graph mit  $rad(G) = 1$



POS (Theorie)

Knoten	Exzentrizität
1	2
2	1
3	2

**Anmerkung:** Das Minimum aller Exzentrizitäten im Beispielgraphen beträgt 1. Somit beträgt der Radius  $rad(G) = 1$

Eigenschaften

15 / 20

## Radius (1)

### Definition (Radius)

Der Radius  $rad(G)$  eines Graphen  $G$  ist das Minimum aller Exzentrizitäten aller Knoten von  $G$ .

Für alle ungerichteten zusammenhängenden Graphen gilt:

$$rad(G) \leq dm(G) \leq 2 \cdot rad(G)$$

sowie

$$dm(G) \geq rad(G) \geq dm(G)/2$$

**Anmerkung:** Das heißt, der Radius kann maximal gleich groß sein wie der Durchmesser und muss mindestens halb so groß wie der Durchmesser sein.

**Anmerkung:** Wenn der Graph nicht zusammenhängend ist, ist der Radius unendlich ( $rad(G) = \infty$ ).

POS (Theorie)

Eigenschaften

14 / 20

## Zentrum (1)

### Definition (Zentrum)

In einem ungerichteten, zusammenhängenden Graphen  $G$  ist das Zentrum  $Z(G)$  die Menge der Knoten  $v \in V$  deren Exzentrizität dem Radius entspricht:

$$Z(G) = \{v \in V(G) : ex(v) = rad(G)\}$$

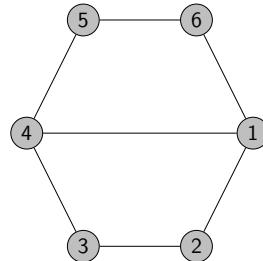
POS (Theorie)

Eigenschaften

16 / 20

## Zentrum (2)

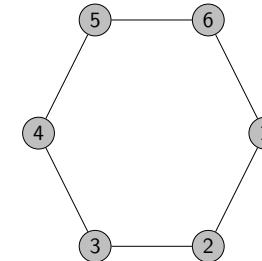
**Beispiel:** Graph  $G_1$  mit 6 Knoten



Somit gilt:  $dm(G_1) = 3$ ,  $rad(G_1) = 2$ ,  $Z(G_1) = \{1, 4\}$

## Zentrum (3)

**Beispiel:** Graph  $G_2$  mit 6 Knoten

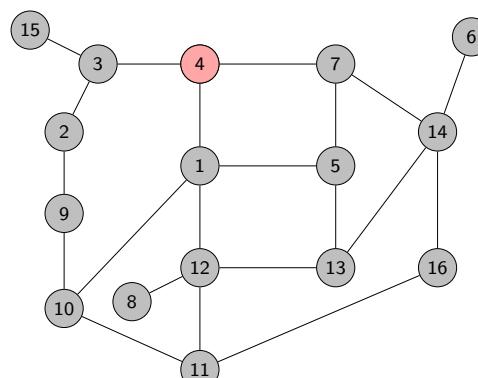


Somit gilt:  $dm(G_2) = 3$ ,  $rad(G_2) = 3$ ,  $Z(G_2) = \{1, 2, 3, 4, 5, 6\}$

## Zentrum (4)

**Beispiel:** In einer Stadt soll eine neue Feuerwache gebaut werden. Um im Brandfall eine optimale Anfahrtszeit zu gewährleisten, soll der Standort der Feuerwache so gewählt werden, dass auch das am weitesten entfernte Haus schnell zu erreichen ist.

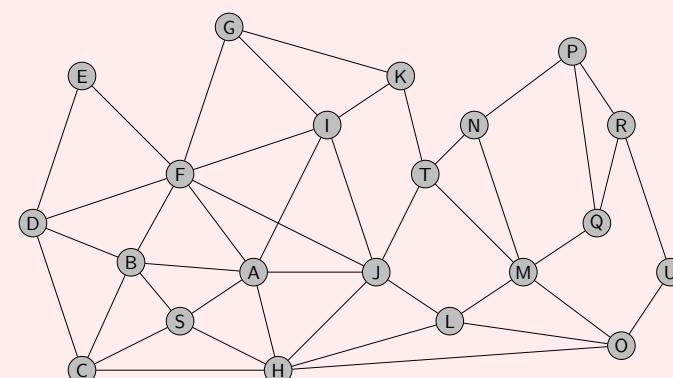
Der Graph stellt die Stadt dar. Ein Haus wird durch einen Knoten und eine Straßenverbindung durch eine Kante symbolisiert. Aus diesem Graph lässt sich sehr rasch das Zentrum der Stadt berechnen.



$v$	$ex(v)$	$v$	$ex(v)$
1	4	9	5
2	5	10	4
3	4	11	5
4	3	12	4
5	4	13	5
6	5	14	4
7	4	15	5
8	5	16	5

### Aufgabe 5.11

Gegeben sei der Graph  $G$ :



- 1 Geben Sie die Exzentrizitäten zu allen Knoten an.
- 2 Bestimmen Sie den Radius  $rad(G)$ .
- 3 Bestimmen Sie den Durchmesser  $dm(G)$ .
- 4 Bestimmen Sie das Zentrum, und schreiben Sie die Knotenmenge  $Z(G)$  auf.

# Graphentheorie: Zusammenhang

## Programmieren und Software-Engineering Theorie

2. September 2025

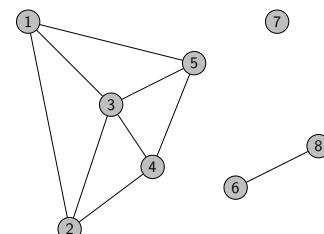
## Zusammenhangskomponenten

### Definition (Zusammenhangskomponenten)

Unter einer *Zusammenhangskomponenten*  $K(v)$  versteht man eine Teilmenge von Knoten maximaler Größe die mit  $v$  durch einen Weg verbunden sind.

$$K(v) = \{u \in V(G) \mid v \rightsquigarrow u\}$$

**Beispiel:** Graph  $G_3$  mit 3 Zusammenhangskomponenten:



$$Z_1 = \{1, 2, 3, 4, 5\}$$

$$Z_2 = \{6, 8\}$$

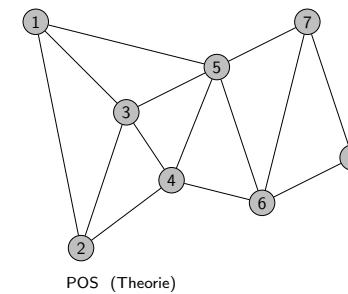
$$Z_3 = \{7\}$$

## Zusammenhang

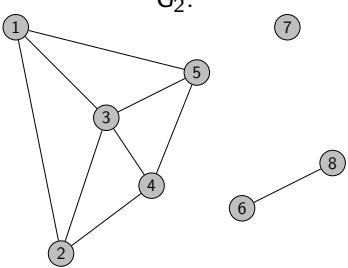
### Definition (Zusammenhang)

Ein ungerichteter Graph  $G = (V, E)$  heißt genau dann *zusammenhängend*, wenn für alle Paare  $u, v \in V$  gilt, dass sie durch einen Weg verbunden sind (kurz:  $u \rightsquigarrow v, \forall u, v \in V(G)$ ).

**Beispiel:** Zusammenhängender Graph  $G_1$ :



**Beispiel:** Nicht zusammenhängender Graph  $G_2$ :



## Komponenten

### Definition (Komponenten)

Die von den Zusammenhangskomponenten aufgespannten, gesättigten Teilgraphen von  $G$  heißen die *Komponenten* des Graphen  $G$ .

### Definition (Anzahl der Komponenten)

Die Anzahl der Komponenten eines Graphen  $G$  wird mit  $c(G)$  bezeichnet.

**Beispiel:**  $c(G_3) = 3$

**Beispiel:** Die Komponenten von  $G_3$  lauten:

$$K_1 = (\{1, 2, 3, 4, 5\}, \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\})$$

$$K_2 = (\{6, 8\}, \{\{6, 8\}\})$$

$$K_3 = (\{7\}, \emptyset)$$

## Artikulation

### Definition (Artikulation)

Ein Knoten  $v$  heißt *Artikulation* wenn die Anzahl der Komponenten von  $G - \{v\}$  größer ist als jene von  $G$ , also

$$c(G - \{v\}) > c(G).$$

- Artikulationen haben niemals Grad 0 oder 1.
- Artikulationen sind Schnittstellen zwischen größer gleich zwei Blöcken.
- Nach Entfernung einer Artikulation ist der Graph nicht (mehr) zusammenhängend.

## Blöcke

### Definition (Block)

Ein *Block* ist ein zusammenhängender Teilgraph, der keine Artikulationen hat und es keinen Obergraph zu diesem Teilgraph gibt, der ebenfalls keine Artikulationen hat.

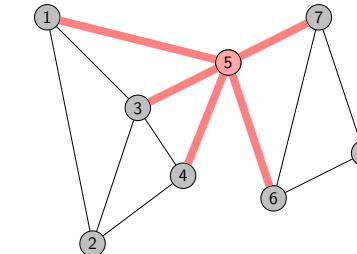
**Bemerkung:** ein Block ist also ein *maximaler* Teilgraph ohne Artikulationen (maximal bezüglich der Anzahl seiner Knoten).

### Folgerungen:

- Jede Kante und jeder Kreis liegen in genau einem Block von  $G$ .
- Es gibt keine Kante die in zwei Blöcken liegen kann.
- Zwei Blöcke eines Graphen haben höchstens einen gemeinsamen Knoten und dieser ist eine Artikulation.
- Der kleinste Block eines Graphen ist ein isolierter Knoten.

## Artikulation

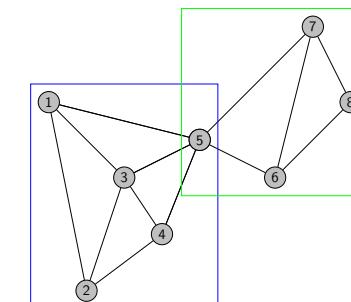
### Beispiel: Graph $G_4$ :



$G - \{5\}$  besteht nun aus den Komponenten  $K_1 = (K(1), \{[1, 2], [1, 3], [2, 3], [2, 4], [3, 4]\})$  und  $K_2 = (K(6), \{[6, 7], [6, 8], [7, 8]\})$ .

## Blöcke

### Beispiel: Graph $G_5$ mit zwei Blöcken:



Die Knotenmengen der Blöcke lauten:

$$B_1 = \{1, 2, 3, 4, 5\}$$

$$B_2 = \{5, 6, 7, 8\}$$

Der Block selbst ist der durch diese Knotenmengen definierte spannende, gesättigte Teilgraph.

## Blöcke

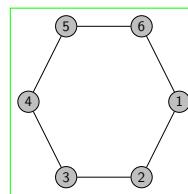
**Beispiel:** Der Kleinstmögliche Block besteht nur aus einem Knoten.

Graph  $G_6$ :

①

Knotenmenge zum Block:  $B_1 = \{1\}$

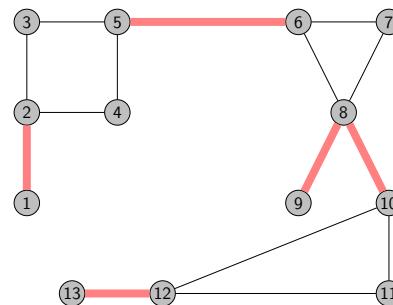
**Beispiel:** Ein zusammenhängender Graph ohne Artikulation ist selbst ein Block. Graph  $G_7$ :



Knotenmenge zum Block:  $B_1 = \{1, 2, 3, 4, 5, 6\}$

## Brücken

**Beispiel:** Graph  $G_8$  mit 5 Brücken



## Brücke

### Definition (Brücke)

Eine Kante  $e$  eines Graphen  $G$  heißt *Brücke*, wenn sich nach Entfernung dieser Kante die Anzahl der Komponenten des Graphen erhöht, also

$$c(G - \{e\}) > c(G).$$

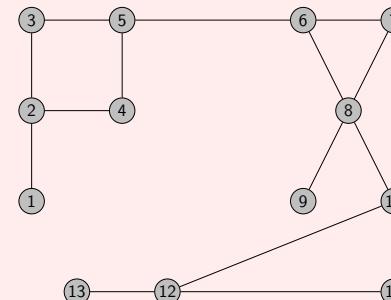
### Anmerkung:

- Eine Brücke ist selbst ein Block.
- Eine Brücke ist damit, wie auch eine Artikulation, eine Schwachstelle im Graphen.

## Brücken

### Beispiel 6.01

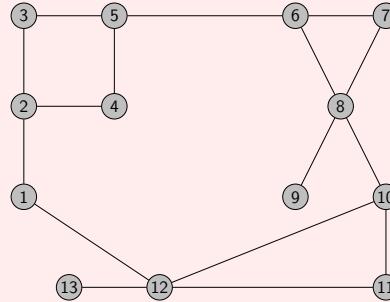
Gegeben sei der folgende Graph  $G_8$ . Bestimmen Sie alle Artikulationen, Brücken und Blöcke.



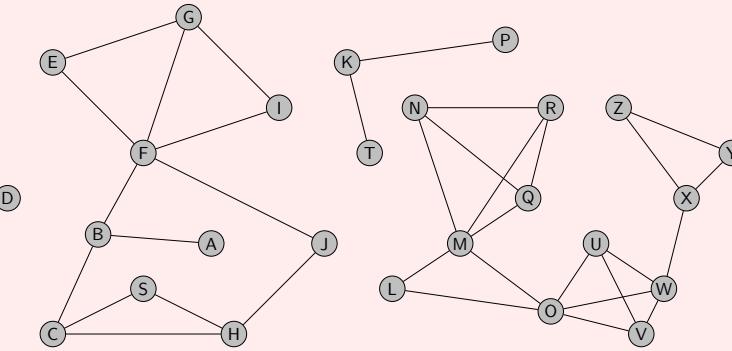
## Brücken

## Beispiel 6.02

Gegeben sei der folgende Graph  $G_9$ . Bestimmen Sie alle Artikulationen, Brücken und Blöcke.



Beispiel 6.11: Gegeben sei der folgende Graph  $G_{10}$ :

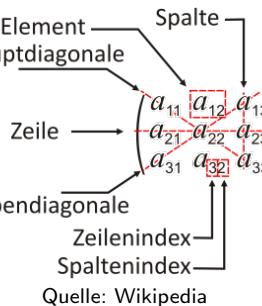


- ① Geben Sie alle Zusammenhangskomponenten an.
  - ② Bestimmen Sie alle Artikulationen.
  - ③ Bestimmen Sie alle Brücken.
  - ④ Bestimmen Sie alle Blöcke, und schreiben Sie diese jeweils als Knotenmenge auf.

# Graphentheorie: Matrizenbasierte Algorithmen

## Programmieren und Software-Engineering Theorie

2. September 2025



## Matrixmultiplikation

Ist  $A$  eine  $n \times m$ -Matrix

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{pmatrix}$$

und  $B$  eine  $m \times p$ -Matrix

$$B = \begin{pmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1p} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & b_{m3} & \dots & b_{mp} \end{pmatrix}$$

dann ist das Matrizenprodukt

$$A \cdot B = C$$

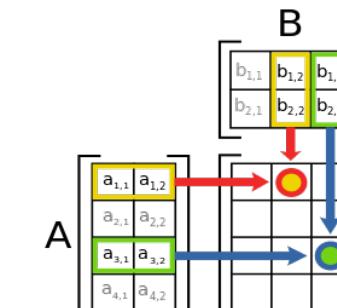
gegeben durch die  $n \times p$ -Matrix

$$C = \begin{pmatrix} c_{11} & c_{12} & c_{13} & \dots & c_{1p} \\ c_{21} & c_{22} & c_{23} & \dots & c_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & c_{n3} & \dots & c_{np} \end{pmatrix}$$

mit

$$c_{ij} = \sum_{k=1}^m a_{ik} b_{kj}$$

## Matrixmultiplikation



Quelle: Wikipedia

- Man stelle sich die Matrizen  $A$  und  $B$  bei der Multiplikation so angeordnet vor, wie in der Grafik links dargestellt (oder schreibe sie tatsächlich so auf!)

- In

$$c_{ij} = \sum_{k=1}^m a_{ik} b_{kj}$$

läuft der Index über die Elemente einer Zeile der Matrix  $A$  und über die Elemente einer Spalte in der Matrix  $B$

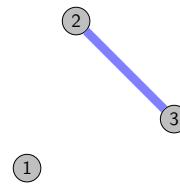
- Es wird das jeweils  $k$ -te Element der Zeile, bzw. Spalte *multipliziert* und zur bisherigen Summe *addiert*



## Werte in der Adjazenzmatrix

In einem *ungerichteten* Graphen erhält man eine obere Dreiecksmatrix. Die Einträge in grau sind bei derartigen Graphen nicht notwendig.

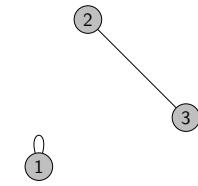
		zum Knoten	1	2	3
vom Knoten		1	0	0	0
	1	2	0	0	1
	2	3	0	1	0



## Adjazenzmatrix

Werte in der Hauptdiagonale entsprechen Schlingen

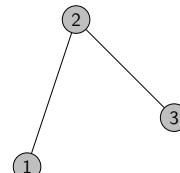
		zum Knoten	1	2	3
vom Knoten		1	1	0	0
	1	2	0	0	1
	2	3	0	1	0



## Knotengrade in der Adjazenzmatrix

Die Zeilen- bzw. Spaltensummen ergeben den **Knotengrad**.

		zum Knoten	1	2	3	
vom Knoten		1	0	1	0	1
	1	2	1	0	1	2
	2	3	0	1	0	1
	3	1	0	1	0	1



**Anmerkung:** Einträge "1" in der Hauptdiagonale müssen für die Berechnung der Knotengrade doppelt gezählt werden. Oftmals wird für Schlingen auch einfach der Wert 2 verwendet.

## Definition (Potenzmatrix)

Unter einer *Potenzmatrix* versteht man das mehrfache Produkt einer (Adjazenz-)Matrix mit sich selbst.

*Beispiel:*  $A^2 = A \cdot A$ , bzw.  $A^3 = A^2 \cdot A = A \cdot A \cdot A$ .

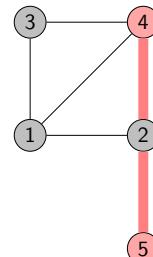
**Anmerkung:** Im Allgemeinen können  $n \times m$  Matrizen nicht mit sich selbst multipliziert werden. Da jedoch Adjazenzmatrizen immer *quadratische* Matrizen sind, ist dies immer möglich!

## Zweck und Verwendung der Potenzmatrix

Die Einträge  $a_{i,j}$  der Potenzmatrix  $A^k(G)$  geben die Anzahl der Kantenfolgen der Länge  $k$  zwischen dem Knoten  $i$  und Knoten  $j$  an ( $i, j \in V$ ).

- Die Potenzmatrix soll in weiterer Folge verwendet werden um die Distanzen im Graphen zu berechnen.
- Grundidee:
  - Es werden nach und nach höhere Potenzmatrizen berechnet.
  - Wenn zwei Knoten  $i$  und  $j$  die Distanz  $k$  haben, dann tritt in  $a_{i,j}$  aus  $A^k(G)$  erstmals ein von 0 verschiedener Wert auf.
  - In allen Potenzmatrizen mit Potenzen kleiner als  $k$  war dieser Eintrag 0 (d.h. es gibt keine Kantenfolgen kürzerer Länge).
  - Das erstmalige Auftreten von  $a_{i,j} \neq 0$  in einer Potenzmatrix wird im nächsten Abschnitt für die Berechnung der Distanzen verwendet.

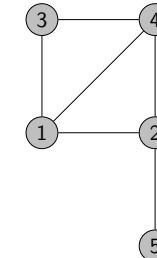
## Beispiel: Potenzmatrix (2)



$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

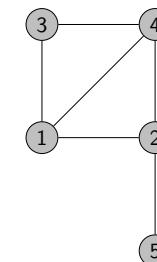
Von  $v_4$  zu  $v_5$  existiert eine Kantenfolge der Länge 2 (über Knoten  $v_2$ ). In der Matrix ist  $a_{4,5} = 0$ , da keine Kante  $[4, 5]$  existiert. In der Matrix sind markiert: die Kante  $[4, 2]$  die von  $v_2$  weg führt, und die Kante  $[2, 5]$  die zu  $v_5$  hin führt.

## Beispiel: Potenzmatrix (1)



$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

## Beispiel: Potenzmatrix (3)

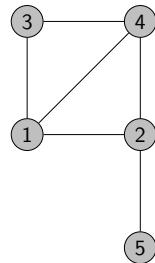


$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$A^2(G) = \begin{pmatrix} 3 & 1 & 1 & 2 & 1 \\ 1 & 3 & 2 & 1 & 0 \\ 1 & 2 & 2 & 1 & 0 \\ 2 & 1 & 1 & 3 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

Wir bezeichnen mit  $a_{i,j}^2$  einen Wert in der Matrix  $A^2(G)$ . Bei der Berechnung von  $a_{4,5}^2 = a_{4,1} \cdot a_{1,5} + a_{4,2} \cdot a_{2,5} + a_{4,3} \cdot a_{3,5} + a_{4,4} \cdot a_{4,5} + a_{4,5} \cdot a_{5,5} = 1 \cdot 0 + 1 \cdot 1 + 1 \cdot 0 + 0 \cdot 0 + 0 \cdot 0 = 1$  wurde der Eintrag der Kantenfolge von  $v_4$  nach  $v_2$  mit jener von  $v_2$  nach  $v_5$  multipliziert, und ergab einen Wert  $a_{4,5}^2 = 1 > 0$ .

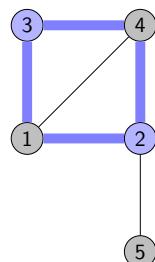
## Beispiel: Potenzmatrix (4)



$$A^2(G) = \begin{pmatrix} 3 & 1 & 1 & 2 & 1 \\ 1 & 3 & 2 & 1 & 0 \\ 1 & 2 & 2 & 1 & 0 \\ 2 & 1 & 1 & 3 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

- Die Einträge in  $A^2(G)$  enthalten in  $a_{ij}^2$  die Anzahl der Kantenfolgen vom Knoten  $i$  zum Knoten  $j$ .
- Die Matrix ist ebenso wie  $A(G)$  symmetrisch (im ungerichteten Fall).
- $a_{3,5}^2 = a_{5,3}^2 = 0$ , weil keine Kantenfolge der Länge 2 von  $v_3$  nach  $v_5$  (und umgekehrt) existiert.

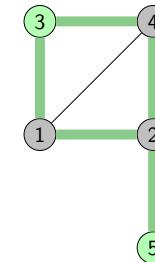
## Beispiel: Potenzmatrix (6)



$$A^2(G) = \begin{pmatrix} 3 & 1 & 1 & 2 & 1 \\ 1 & 3 & 2 & 1 & 0 \\ 1 & 2 & 2 & 1 & 0 \\ 2 & 1 & 1 & 3 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

- In  $A^2(G)$  finden wir zwei Kantenfolgen von  $v_3$  nach  $v_2$
- Somit ist  $a_{3,2}^2 = 2$

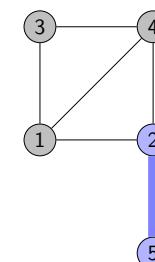
## Beispiel: Potenzmatrix (5)



$$A^2(G) = \begin{pmatrix} 3 & 1 & 1 & 2 & 1 \\ 1 & 3 & 2 & 1 & 0 \\ 1 & 2 & 2 & 1 & 0 \\ 2 & 1 & 1 & 3 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

- Wir betrachten nun einen weiteren Schritt, und dabei die Kantenfolgen von  $v_3$  nach  $v_5$  der Länge 3, die im dritten Schritt (Matrix  $A^3(G)$ ) gefunden werden.
- Die Kantenfolgen der Länge 3 sind im Graphen grün markiert.

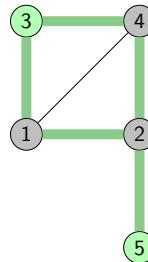
## Beispiel: Potenzmatrix (7)



$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & \textcolor{blue}{1} \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

- In  $A^1(G)$  finden wir eine Kantenfolge von  $v_2$  nach  $v_5$
- Somit ist  $a_{2,5}^1 = 1$

## Beispiel: Potenzmatrix (8)



$$A^2(G) = \begin{pmatrix} 3 & 1 & 1 & 2 & 1 \\ 1 & 3 & 2 & 1 & 0 \\ 1 & 2 & 2 & 1 & 0 \\ 2 & 1 & 1 & 3 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}, A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$A^2(G) \cdot A(G) = A^3(G) = \begin{pmatrix} 4 & 6 & 5 & 5 & 1 \\ 6 & 2 & 2 & 6 & 3 \\ 5 & 2 & 2 & 5 & 2 \\ 5 & 6 & 5 & 4 & 1 \\ 1 & 3 & 2 & 1 & 0 \end{pmatrix}$$

- In  $a_{3,5}^3 = 2$  erhalten wir nun die Information, daß es zwei Kantenfolgen der Länge 3 von  $v_3$  nach  $v_5$  gibt.
- Rechnung:  $a_{3,5}^3 = a_{3,1}^2 \cdot a_{1,5}^1 + a_{3,2}^2 \cdot a_{2,5}^1 + a_{3,3}^2 \cdot a_{3,5}^1 + a_{3,4}^2 \cdot a_{4,5}^1 + a_{3,5}^2 \cdot a_{5,5}^1 = 1 \cdot 0 + 2 \cdot 1 + 2 \cdot 0 + 1 \cdot 0 + 0 \cdot 0 = 2$

## Distanzmatrix

### Definition (Distanzmatrix)

Sei  $G = (V, E)$  ein Graph, und  $n = |V|$ . Eine Matrix  $D \in \mathbb{R}^{n \times n}$  heißt *Distanzmatrix* von  $G$ , wenn alle Einträge  $d_{ij}$  der Distanz zwischen den Knoten  $i$  und  $j$  entsprechen ( $i, j \in V$ ).

### Ergänzende Erklärung zur Berechnung der Anzahl der Kantenfolgen der Länge $k$ :

Bei der Operation der Multiplikation der Matrizen  $A^{k-1}$  mit  $A$  werden die Informationen über (die Anzahl der) Kantenfolgen der Länge  $k - 1$  und der Länge 1 zusammengefügt (nämlich zu jenen der Länge  $k$ ). Dabei werden alle Knoten als Zwischenknoten berücksichtigt; konkret als vorletzten Knoten der Kantenfolge.

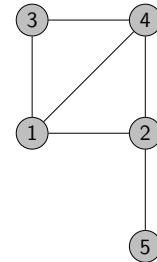
## Distanzmatrix: Berechnung

Die Berechnung der Distanzmatrix  $D(G)$  basiert wiederum auf der Adjazenzmatrix:

- ① Initialisierung:
  - Einträge "1" aus  $A(G)$  werden in  $D^{(1)}(G)$  übernommen
  - Bei Einträgen "0" in  $A(G)$  erhält  $D^{(1)}(G)$  Einträge  $\infty$
  - Nullen in Hauptdiagonale
- ②  $k = 2$
- ③ Für alle Einträge aus der  $A^k(G)$  mit  $a_{ij}^k \neq 0$  und  $d_{ij} = \infty$  setzen wir in  $D^{(k)}(G)$  die Werte  $d_{ij} = k$
- ④  $k = k + 1$
- ⑤ Gehe zu Schritt 3, außer wenn  $\forall i, j, i \neq j : d_{ij} \neq \infty$  oder  $k = n$  oder  $D^{(k-2)} = D^{(k-1)}$

**Anmerkung:**  $D^{(k)}$  steht hier für die Distanzmatrix im Schritt  $k$ .

## Beispiel: Distanzmatrix (1)

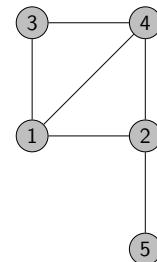


$$A(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$D^{(1)}(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & \infty \\ 1 & 0 & \infty & 1 & 1 \\ 1 & \infty & 0 & 1 & \infty \\ 1 & 1 & 1 & 0 & \infty \\ \infty & 1 & \infty & \infty & 0 \end{pmatrix}$$

Die **Hauptdiagonale** in  $D^{(1)}(G)$  enthält lauter **0en**, alle **1en** werden aus  $A(G)$  übernommen, für alle **0en** in  $A(G)$  die nicht in der Hauptdiagonale liegen, wird in  $D^{(1)}(G)$  der Wert  $\infty$  übernommen.

## Beispiel: Distanzmatrix (3)

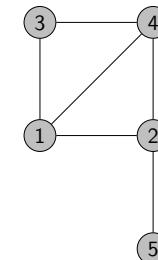


$$A^3(G) = \begin{pmatrix} 4 & 6 & 5 & 5 & 1 \\ 6 & 2 & 2 & 6 & 3 \\ 5 & 2 & 2 & 5 & 2 \\ 5 & 6 & 5 & 4 & 1 \\ 1 & 3 & 2 & 1 & 0 \end{pmatrix}$$

$$D^{(3)}(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 & 1 \\ 1 & 2 & 0 & 1 & 3 \\ 1 & 1 & 1 & 0 & 2 \\ 2 & 1 & 3 & 2 & 0 \end{pmatrix}$$

Im Schritt  $k = 3$  wird für alle neu entstandenen Werte  $a_{ij}^3 \neq 0$  der Wert  $d_{ij} = k$  gesetzt.

## Beispiel: Distanzmatrix (2)



$$A^2(G) = \begin{pmatrix} 3 & 1 & 1 & 2 & 1 \\ 1 & 3 & 2 & 1 & 0 \\ 1 & 2 & 2 & 1 & 0 \\ 2 & 1 & 1 & 3 & 1 \\ 1 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$D^{(2)}(G) = \begin{pmatrix} 0 & 1 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 & 1 \\ 1 & 2 & 0 & 1 & \infty \\ 1 & 1 & 1 & 0 & 2 \\ 2 & 1 & \infty & 2 & 0 \end{pmatrix}$$

Im Schritt  $k = 2$  wird für alle neu entstandenen Werte  $a_{ij}^2 \neq 0$  der Wert  $d_{ij} = k$  gesetzt.

## Distanzmatrix: Anwendungen

- Mit der Distanzmatrix können die **Exzentrizitäten** berechnet werden: Maximum einer Zeile, bzw.

$$ex(i) = \max_k d_{ik}, 1 \leq k \leq n$$

- Durchmesser:**

$$dm(G) = \max_k ex(v_k), 1 \leq k \leq n$$

- Radius:**

$$rad(G) = \min_k ex(v_k), 1 \leq k \leq n$$

## Beispiel: Anwendung Distanzmatrix

$D(G)$ :

	1	2	3	4	5	ex( $v$ )
1	0	1	1	1	2	2
2	1	0	2	1	1	2
3	1	2	0	1	3	3
4	1	1	1	0	2	2
5	2	1	3	2	0	3

Die **Exzentrizitäten** können aus den Zeilen der Matrix ermittelt werden.

## Wegmatrix: Berechnung

Die Berechnung der Wegmatrix basiert auf der Adjazenzmatrix:

- ① Initialisierung:

$$W^{(1)}(G) = A(G) + \mathbb{1}$$

Dabei bezeichnet  $\mathbb{1}$  die Einheitsmatrix, die in der Hauptdiagonale die Werte 1, und sonst nur die Werte 0 enthält.

- ②  $k = 2$   
 ③ Für alle Einträge aus der  $A^k(G)$  mit  $a_{ij}^k \neq 0$  setzen wir in  $W^{(k)}(G)$  die Werte  $w_{ij} = 1$   
 ④  $k = k + 1$   
 ⑤ Gehe zu Schritt 3, außer wenn  $\forall i, j : w_{ij} \neq 0$  oder  $k = n$  oder  $W^{(k-2)} = W^{(k-1)}$

**Anmerkung:**  $W^{(k)}$  steht hier für die Wegmatrix im Schritt  $k$ .

## Wegmatrix

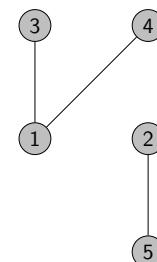
### Definition (Wegmatrix)

Sei  $G = (V, E)$  ein Graph, und  $n = |V|$ . Eine Matrix  $W \in \{0, 1\}^{n \times n}$  heißt **Wegmatrix** oder **Erreichbarkeitsmatrix** von  $G$ , wenn für alle Elemente  $w_{ij}$  gilt:

$$w_{ij} = \begin{cases} 1 & \text{wenn } i \rightsquigarrow j, \\ 0 & \text{sonst.} \end{cases}$$

**Anmerkung:**  $i \rightsquigarrow j$  bedeutet hierbei, dass der Knoten  $j$  von Knoten  $i$  aus erreichbar ist, also dass ein Weg zwischen diesen Knoten existiert. In anderen Worten: die beiden Knoten liegen in der selben (Zusammenhangs-)Komponente.

## Beispiel: Wegmatrix (1)

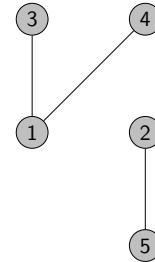


$$A(G) = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

$$W^{(1)}(G) = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Die **Hauptdiagonale** von  $W^{(1)}(G)$  wird mit **1en** initialisiert, der Rest wird von  $A(G)$  übernommen.

## Beispiel: Wegmatrix (2)



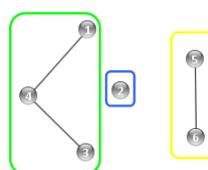
$$A^2(G) = \begin{pmatrix} 2 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$W^{(2)}(G) = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Für die neuen Einträge  $a_{ij}^2 \neq 0$  aus  $A^2(G)$  wird  $w_{ij} = 1$  in  $W^{(2)}(G)$  übernommen.

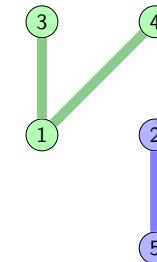
## Wegmatrix: Anwendungen

	1	2	3	4	5	6
1	1	0	1	1	0	0
2	0	1	0	0	0	0
3	1	0	1	1	0	0
4	1	0	1	1	0	0
5	0	0	0	0	1	1
6	0	0	0	0	1	1



- Die Anzahl der unterschiedlichen Zeilen von  $W(G)$  ergibt die Anzahl der Komponenten von  $G$
- Artikulationen können durch Entfernung eines Knoten und Neuberechnung der Matrix ermittelt werden (Anzahl der Komponenten wird größer)
- Brücken können durch Entfernung von Kanten und Neuberechnung der Matrix ermittelt werden (Anzahl der Komponenten wird ebenso wieder größer)

## Beispiel: Wegmatrix (3)

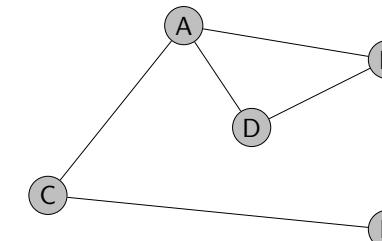


$$W(G) = \begin{pmatrix} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Wir können aus  $W(G)$  die Komponenten  $K_1 = (\{1, 3, 4\}, \{[1, 3], [1, 4]\})$  und  $K_2 = (\{2, 5\}, \{[2, 5]\})$  ablesen.

## Beispiel 7.1.1

Gegeben sei der Graph  $G_1$ :

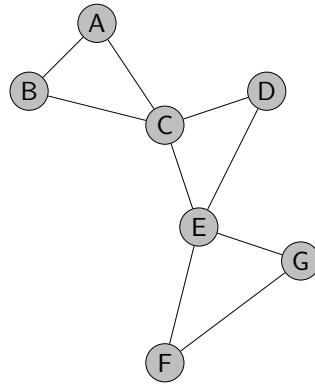


Berechnen Sie die

- Distanzmatrix, und die
  - Wegmatrix
- anhand der Potenz-Matrizen.

### Beispiel 7.1.3

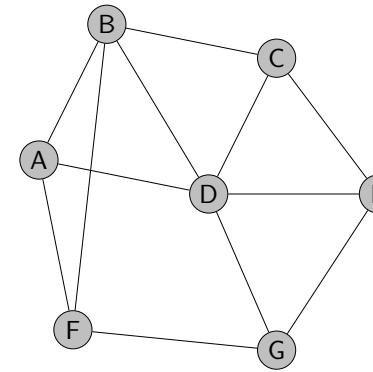
Gegeben sei der Graph  $G_2$ :



Berechnen Sie die Distanzmatrix des Graphen  $G_2$

## Beispiel 7.2.1

Gegeben sei der Graph  $G_3$ :



Berechnen Sie die Anzahl der Kantenfolgen der Länge 5 vom Knoten D zu B in möglichst wenigen Schritten.

## Bäume

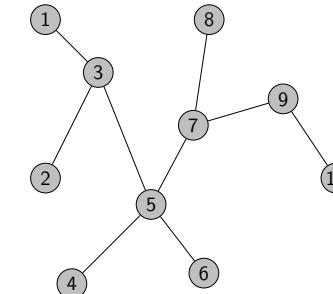
# Graphentheorie: Bäume

## Programmieren und Software-Engineering Theorie

2. September 2025

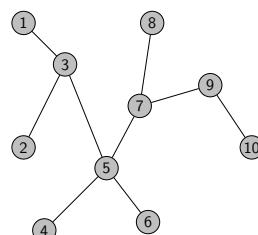
### Definition (Baum)

Ein zusammenhängender Graph der keine Kreise enthält wird *Baum T* genannt.



## Bäume

## Begriffe



Ein Baum  $T$  kann durch folgende äquivalente Aussagen charakterisiert werden:

- $T$  ist ein Baum
- $T$  ist zusammenhängend und kreisfrei
- Zwei beliebige Knoten von  $T$  sind durch genau einen Weg verbunden
- $T$  hat  $n - 1$  Kanten und ist zusammenhängend
- $T$  hat  $n - 1$  Kanten und ist kreisfrei
- $T$  ist zusammenhängend und jede Kante ist eine Brücke

- **Grad eines Baumes:** maximaler Grad eines Knoten des Baumes
- **Blatt:** Knoten mit Grad 1 heißen *Blätter* des Baumes. Blätter haben nur einen Nachbarn. Jeder Baum hat mindestens ein Blatt.
- **Ast:** Die Kanten eines Baumes werden auch als *Äste* bezeichnet.
- **Innerer Knoten:** Ein Knoten heißt *innerer Knoten* wenn er kein Blatt ist.

**Anmerkung:** Bäume haben zahlreiche Anwendungen in der Informatik als Datenstrukturen, z.B. können Objekte in Bäumen so abgespeichert werden, dass sie schnell gesucht (gefunden) werden können!

## Wurzelbäume, Arboreszenzen

### Definition (Wurzelbaum, Arboreszenz)

Ein gerichteter Graph  $G$  heißt *Wurzelbaum* oder *Arboreszenz*, wenn er

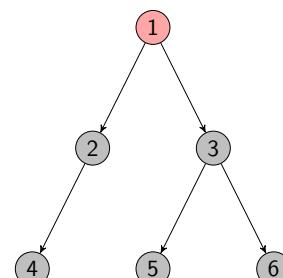
- zusammenhängend ist, und
- es genau einen Knoten  $w \in V(G)$  gibt mit  $d^-(w) = 0$ , und
- für alle anderen Knoten  $v \in V(G)$  gilt  $d^-(v) = 1$ .

Hierbei bezeichnet  $d^-(v)$  den *Eingangsgrad* (engl. in-degree), also die Anzahl der zum Knoten führenden Kanten. Der *Ausgangsgrad*  $d^+(v)$  (engl. out-degree) bezeichnet hingegen die vom Knoten wegführenden Kanten.

**Anmerkung:** Ein Wurzelbaum ist zusammenhängend und hat genau einen Knoten mit Eingangsgrad 0. Jeder andere Knoten hat den Eingangsgrad 1. Eine Arboreszenz ist ein gerichteter Graph dessen Schatten ein Baum ist, sodass ein Knoten  $w$  ausgezeichnet ist (Wurzel), und jede Kante der Arboreszenz von  $w$  weggerichtet ist.

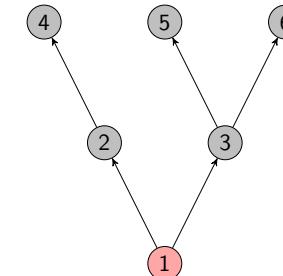
## Wurzelbäume, Arboreszenzen

**Beispiel:** Heute wird die Wurzel immer oben gezeichnet.



## Wurzelbäume, Arboreszenzen

**Beispiel:** Ursprünglich wurden Wurzelbäume so gezeichnet:



## Traversierung von Bäumen

Wir betrachten im Folgenden Algorithmen zur Traversierung von Bäumen. Diese können jedoch auch zur Traversierung von Graphen im Allgemeinen verwendet werden.

**Ziel:** Wir wollen die Knoten eines Baumes systematisch durchlaufen, mit dem Ziel einen bestimmten Knoten zu finden.

- **Breitensuche (Breadth-First-Search (BFS)):** In jedem Schritt werden zunächst alle Nachbarknoten eines Knoten besucht, bevor von dort aus weitere Pfade gebildet werden.
- **Tiefensuche (Depth-First-Search (DFS)):** Ein Pfad wird vollständig in die Tiefe durchlaufen, bevor etwaige Abzweigungen verwendet werden.

## Traversierung von Bäumen

Zur Beschreibung der Suchverfahren werden folgende Begriffe benötigt:

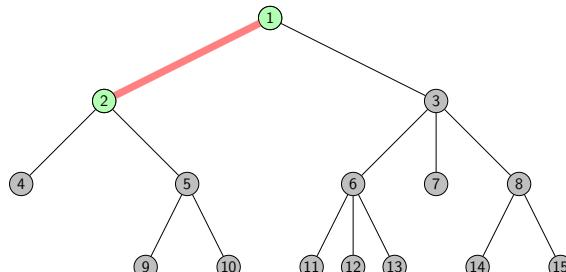
- Ein Knoten wird **entdeckt**, wenn er das erste Mal besucht wird.
- Ein Knoten wird **fertiggestellt/abgeschlossen**, wenn er das letzte Mal verlassen wird.

Für manche Anwendungen ist es wichtig festzuhalten, wann ein Knoten *entdeckt*, bzw. abgeschlossen wurde. Dazu führen wir einen Zähler  $\tau$  mit, der in jedem Schritt um 1 erhöht wird.

- Wird ein Knoten  $v$  das erste mal besucht ("entdeckt"), so setzen wir  $\tau_d(v) = \tau++$
- Wird ein Knoten  $v$  das letzte mal verlassen ("abgeschlossen"), so setzen wir  $\tau_f(v) = \tau++$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

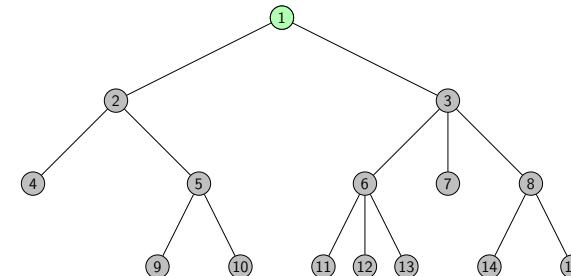


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

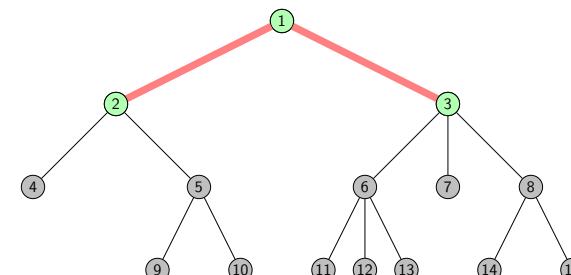


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

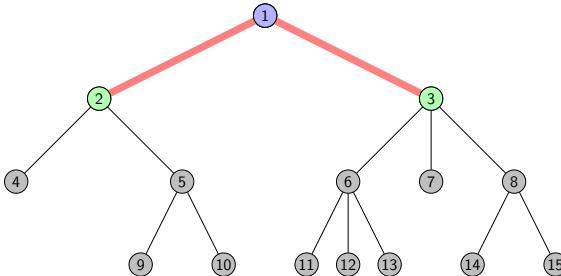


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

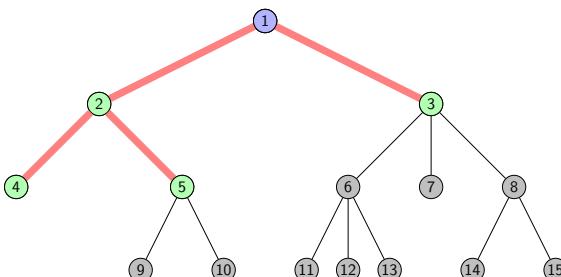


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

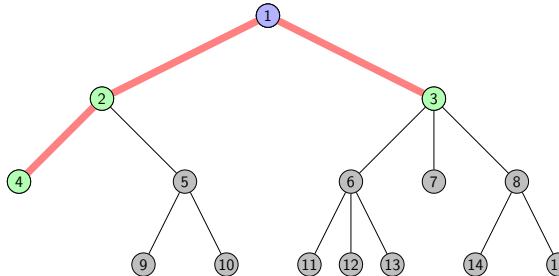


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

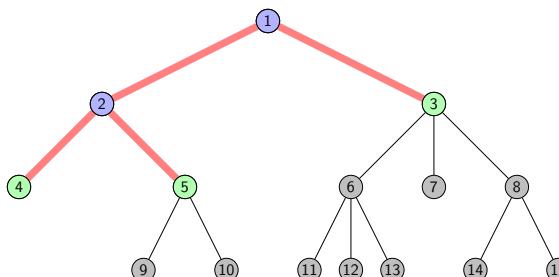


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

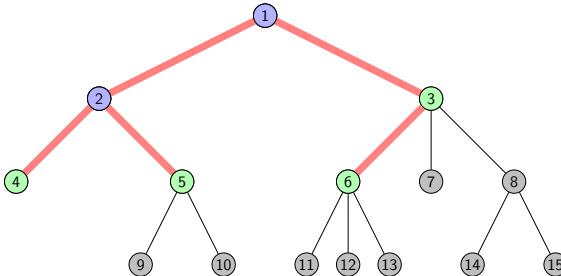


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

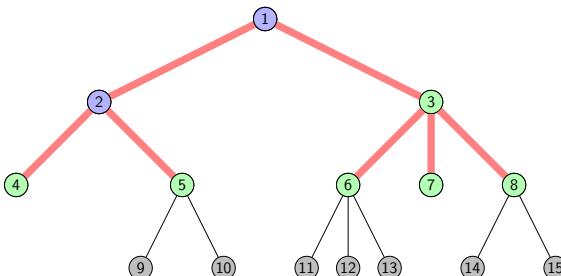


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

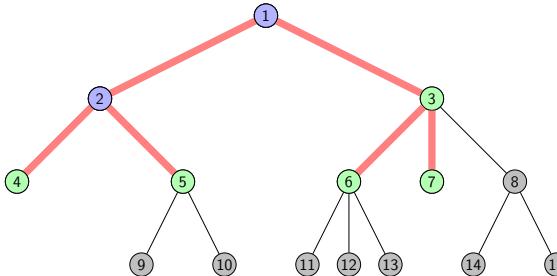


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

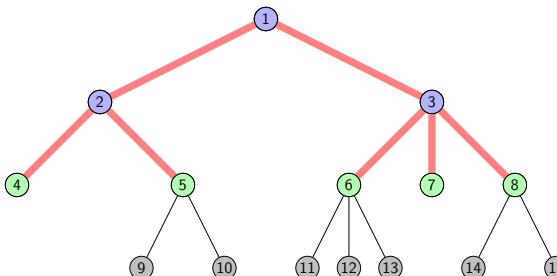


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.

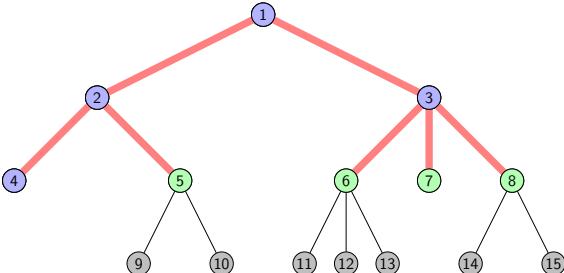


$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

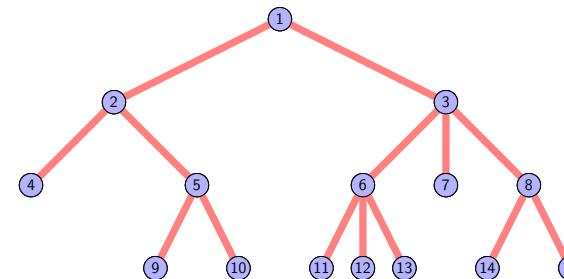
## Breitensuche: Datenstruktur

In nahezu allen Programmiersprachen existiert eine Datenstruktur namens **Queue** (Warteschlange). Elemente können hinzugefügt werden ("hinten anstellen"), und werden geordnet abgespeichert. Das Element das als erstes hinzugefügt wurde, kann entnommen werden ("Nächster!").



## Breitensuche

**Beispiel:** Beispiel einer Breitensuche. Die Suche kann abgebrochen werden, sobald der gesuchte Knoten gefunden ("entdeckt") wurde.



**Achtung: einige Zwischenschritte sind in diesem Handout übersprungen! Alle Schritte sind im einzelnen Foliensatz als Animation verfügbar!**

$$\tau_d(1) = 1, \tau_d(2) = 2, \tau_d(3) = 3, \tau_d(4) = 5, \tau_d(5) = 6, \tau_d(6) = 8, \tau_d(7) = 9, \tau_d(8) = 10, \tau_d(9) = 13, \tau_d(10) = 14, \dots$$

$$\tau_f(1) = 4, \tau_f(2) = 7, \tau_f(3) = 11, \tau_f(4) = 12, \tau_f(5) = 15, \dots$$

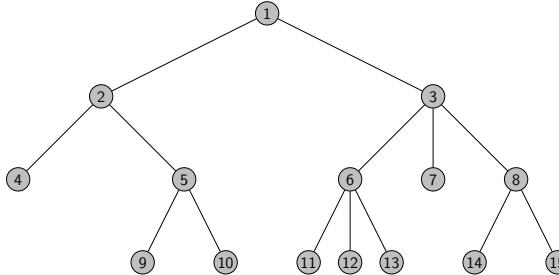
## Breitensuche: Algorithmus

- ❶ Füge Startknoten (Wurzel) in Queue (Warteschlange) ein und markiere ihn als entdeckt
- ❷ Entnimm Knoten am Beginn der Queue
  - Markiere Knoten als entdeckt
  - Wenn dies der gesuchte Knoten ist  $\Rightarrow$  Fertig!
  - Sonst: füge alle unbesuchten<sup>1</sup> Nachbarn dieses Knotens in die Queue ein
- ❸ Wenn die Warteschlange leer ist wurden alle Knoten bereits besucht  $\Rightarrow$  Fertig
- ❹ Gehe zu Schritt (2)

<sup>1</sup>nicht entdeckt, nicht abgeschlossen

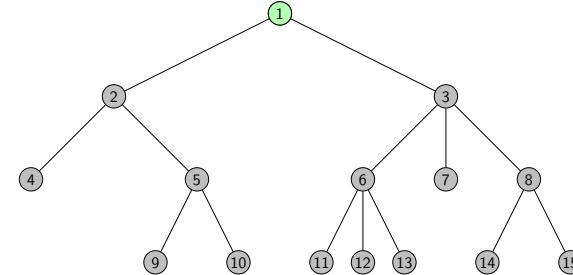
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



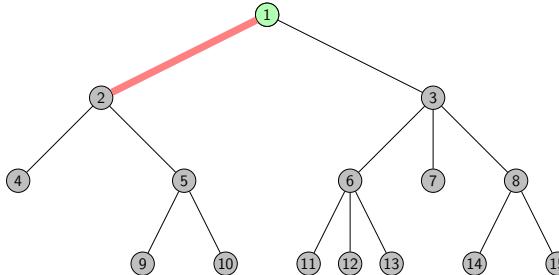
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



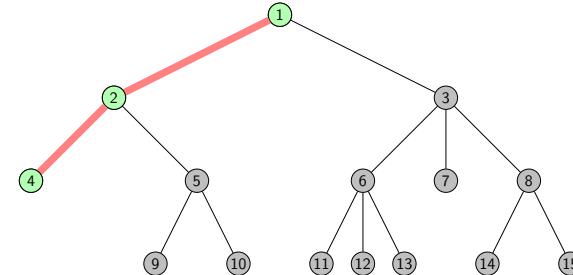
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



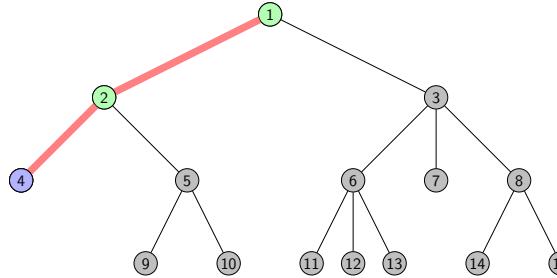
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



## Tiefensuche

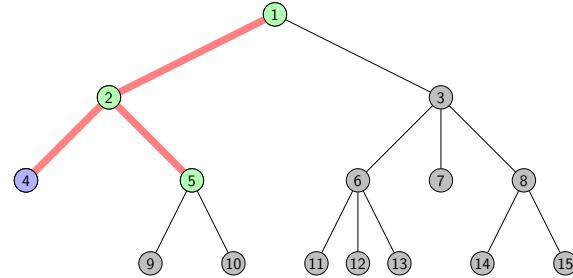
**Beispiel:** Beispiel einer Tiefensuche:



## Tiefensuche

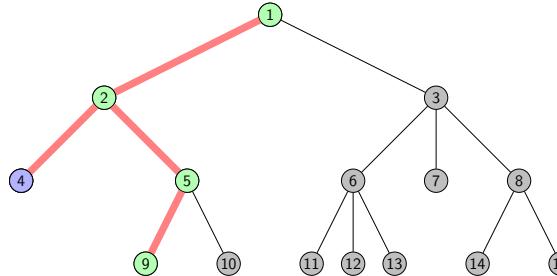
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:

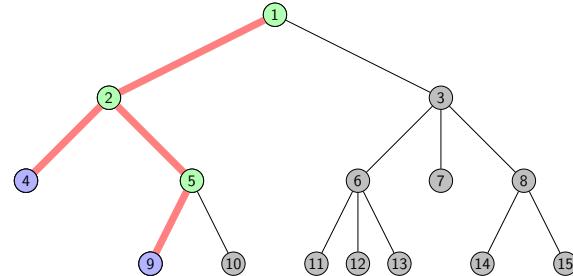


## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:

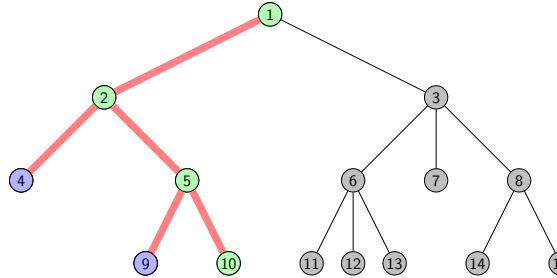


**Beispiel:** Beispiel einer Tiefensuche:



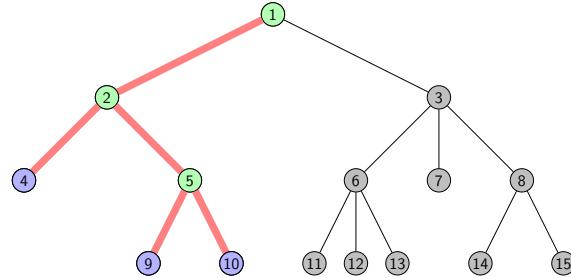
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



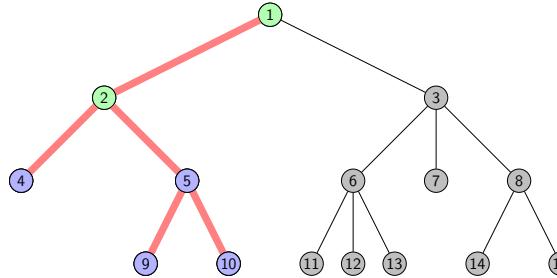
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



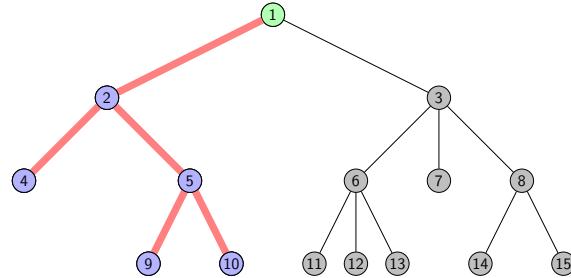
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



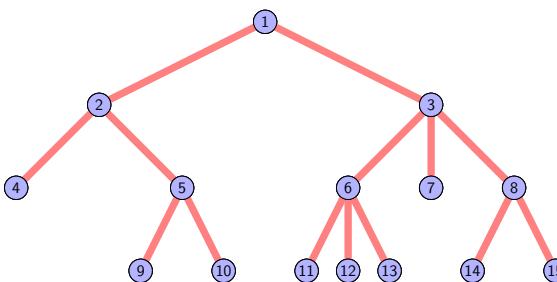
## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



## Tiefensuche

**Beispiel:** Beispiel einer Tiefensuche:



**Achtung:** einige Zwischenschritte sind in diesem Handout übersprungen! Alle Schritte sind im einzelnen Foliensatz als Animation verfügbar!

## Tiefensuche: Algorithmus (\*)

### Algorithm 1: Tiefensuche

```

1 Function DFS( $G = (V, E)$ , Startknoten  $v$ , Gesuchter Knoten  $s$ )
  Result: vertex  $s \in V$ , if exists
2   Stack  $S$ ;
3   markiere  $v$  als entdeckt;
4    $S.push(v)$ ; // Lege  $v$  auf Stapel
5   while  $S$  not empty do
6      $v = S.pop()$ ; // nimm obersten Knoten vom Stapel
7     if  $v$  gesuchter Knoten  $s$  then
8       return  $v$ ;
9     markiere  $v$  als abgeschlossen;
10    for all  $[v, u] \in E(G)$  do
11      if  $u$  noch nicht besucht then
12        markiere  $u$  als entdeckt;
13         $S.push(u)$ ;
  
```

## Tiefensuche: Datenstruktur

In nahezu allen Programmiersprachen existiert eine Datenstruktur namens **Stack** (Stapel).

Elemente können hinzugefügt werden ("oben drauflegen"), und werden geordnet abgespeichert. Das Element das als *letztes* hinzugefügt wurde, kann entnommen werden (oberstes Element vom Stapel nehmen).



## Anmerkungen

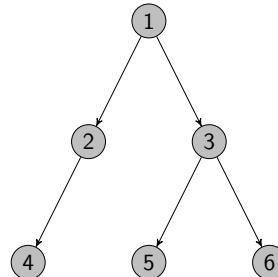
- Die Algorithmen können auch für allgemeine Graphen (und nicht nur Bäume) verwendet werden.
- Dabei werden schon besuchte Knoten *nicht* erneut besucht!
- Anwendungen BFS:
  - 2-färbbarkeit
  - Kürzester Pfad zwischen zwei Knoten
  - Kürzeste-Kreise-Problem
- Anwendungen DFS:
  - Test auf Kreisfreiheit
  - Topologische Sortierung
  - Starke Zusammenhangskomponente

## Binärbaum

In einem Wurzelbaum bezeichnet man alle Knoten  $u$  als Kinder von  $v$  wenn eine Kante  $(v, u)$  existiert.

### Definition (Binärbaum)

Ein Binärbaum ist ein Wurzelbaum in dem jeder Knoten maximal zwei Kinder hat.



POS (Theorie)

Bäume

17 / 28

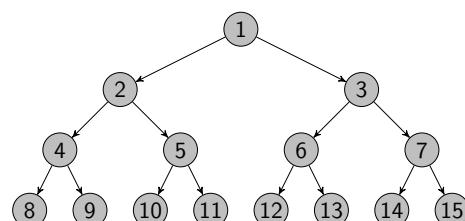
## Vollständiger Binärbaum

### Definition (Tiefe eines Knotens)

Die Tiefe eines Knotens ist die Distanz zur Wurzel.

### Definition (Vollständiger Binärbaum)

Ein vollständiger Binärbaum ist ein voller Binärbaum in dem alle Blätter die gleiche Tiefe haben.



POS (Theorie)

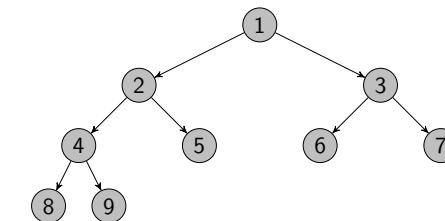
Bäume

19 / 28

## Voller Binärbaum

### Definition (Voller Binärbaum)

Ein Binärbaum ist *voll*, wenn alle Knoten entweder 0 oder 2 Kinder haben.



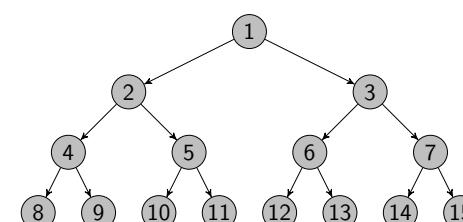
POS (Theorie)

Bäume

18 / 28

## Vollständiger Binärbaum

Tiefe	Anzahl Knoten
0	1
1	2
2	4
3	8



POS (Theorie)

Bäume

20 / 28

## Vollständiger Binärbaum

In jeder Ebene verdoppelt sich die Anzahl der Knoten.

Zusammenhang: Tiefe – Anzahl an Knoten pro Ebene

Die Tiefe  $t$  hängt mit der Anzahl der Knoten  $n$  in dieser Ebene wie folgt zusammen:

$$2^t = n$$

Tiefe eines vollständigen Binärbaumes ist der *Logarithmus Dualis* der Anzahl der Blätter (hier ebenso mit  $n$  bezeichnet):

$$t = \lceil \ln n \rceil$$

Der Logarithmus Dualis hängt mit dem natürlichen Logarithmus wie folgt zusammen:

$$\lceil \ln a \rceil = \log_2 a = \frac{\ln a}{\ln 2}$$

## Algorithmus von Kruskal

### Algorithm 2: Algorithmus von Kruskal

1 **Function** Kruskal(*Graph G = (V, E)*)

**Result:** Minimum Spanning Tree  $T$

2 Ordne alle Kanten  $e \in E(G)$  aufsteigend nach  $w(e)$ ;  
 $\Rightarrow L = (e_1, e_2, \dots, e_n)$  mit  $w(e_1) \leq w(e_2) \leq \dots \leq w(e_n)$

3  $V(T) = V(G)$ ;

4  $E(T) = \emptyset$ ;

5 **for**  $e \in L$  **do**

6     **if**  $T = (V, E(T) \cup \{e\})$  ist kreisfrei **then**  
7          $E(T) = E(T) \cup \{e\}$ ;

- Die Kreisfreiheit kann mit DFS berechnet werden.
- Nach Hinzufügen von  $n - 1$  Kanten kann der Algorithmus beendet werden.

## Minimale Spannbäume

### Definition (Spannbaum)

Ein *Spannbaum*  $T$  zu einem Graphen  $G = (V, E)$  ist ein (auf-)spannender Teilgraph von  $G$  der ein Baum ist.

Wir betrachten nun ungerichtete, schlichte und zusammenhängende Graphen  $G = (V, E)$  mit einer Gewichtsfunktion  $w : E \rightarrow \mathbb{R}^+$  auf den Kanten  $e \in E(G)$ :

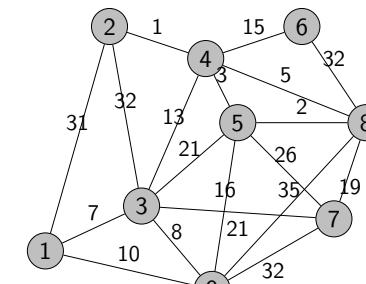
### Definition (Minimaler Spannbaum)

Ein *Minimaler Spannbaum*  $T$  von  $G = (V, E)$  mit  $w : E \rightarrow \mathbb{R}^+$  für alle  $e \in E(G)$  ist ein zusammenhängender Teilgraph mit  $|E(T)| = |V(G)| - 1$  der alle Knoten enthält, und für den gilt:

$$\sum_{e \in E(T)} w(e) = \min$$

Ein Minimaler Spannbaum (Minimum Spanning Tree (MST)) ist also ein Baum mit minimalen Kantengewichten (=Kantenkosten).

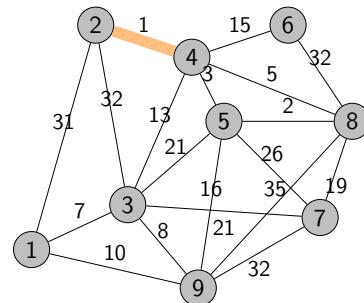
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

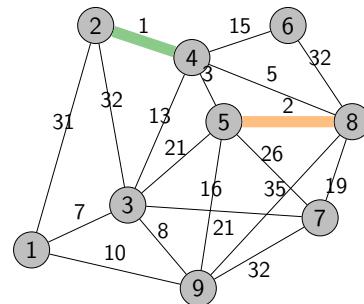
$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

POS (Theorie)

Bäume

24 / 28

## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

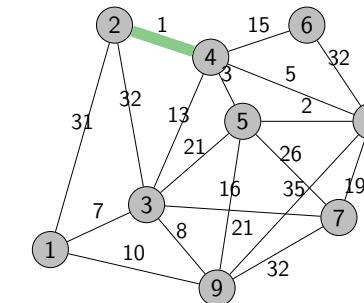
$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

POS (Theorie)

Bäume

24 / 28

## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

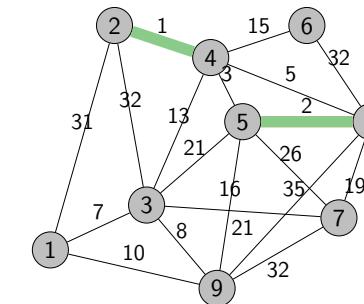
$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

POS (Theorie)

Bäume

24 / 28

## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

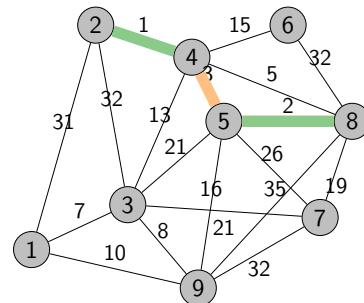
$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

POS (Theorie)

Bäume

24 / 28

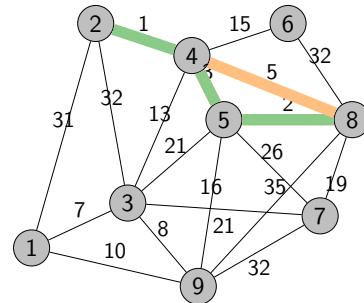
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

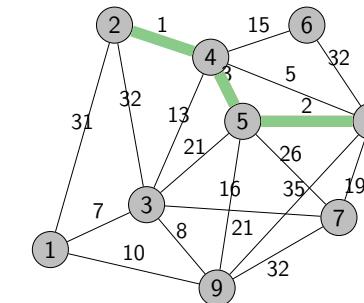
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

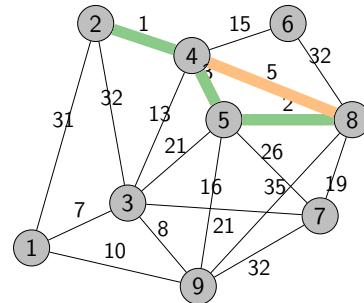
## Algorithmus von Kruskal



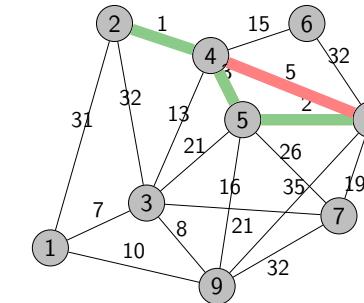
Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

## Algorithmus von Kruskal



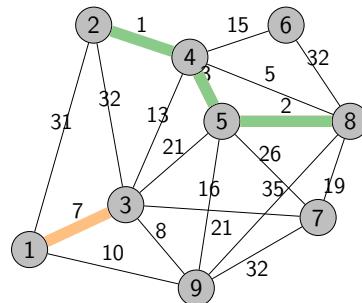
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

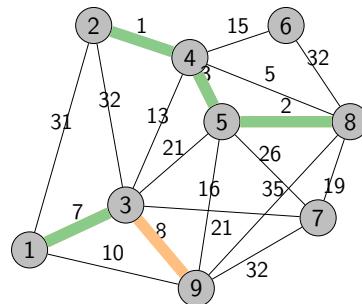
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

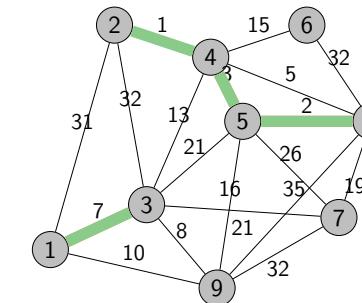
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

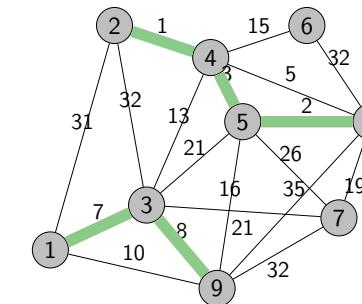
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

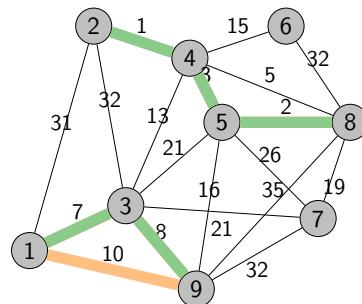
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

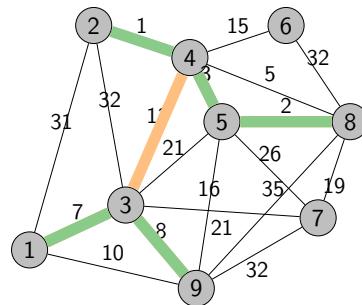
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

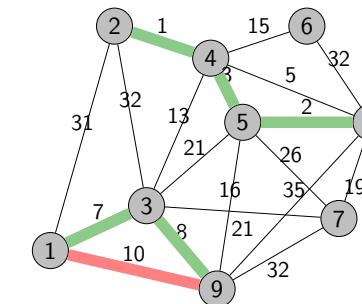
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

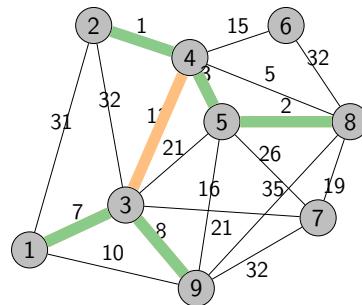
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

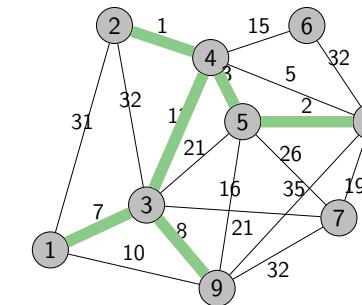
$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

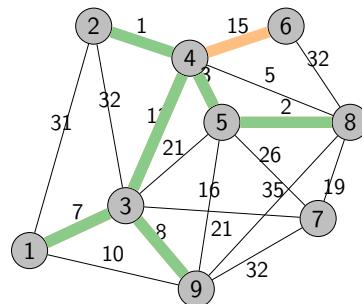
$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

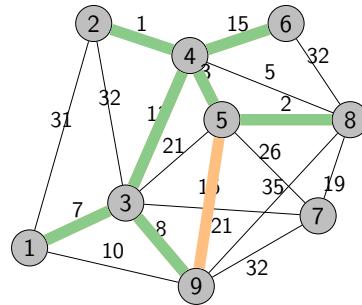
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

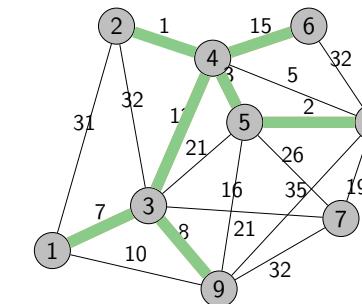
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

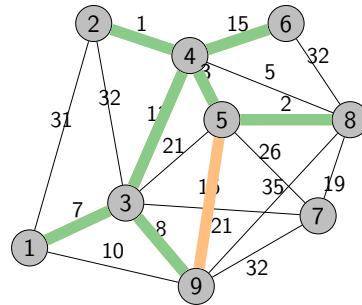
## Algorithmus von Kruskal



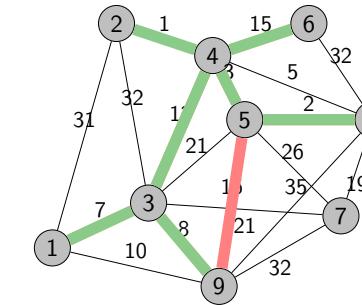
Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

## Algorithmus von Kruskal



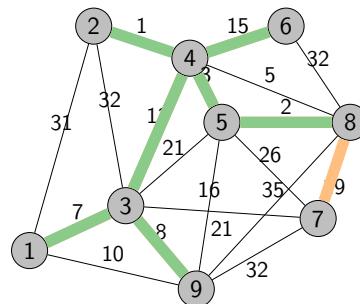
## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

## Algorithmus von Kruskal



Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

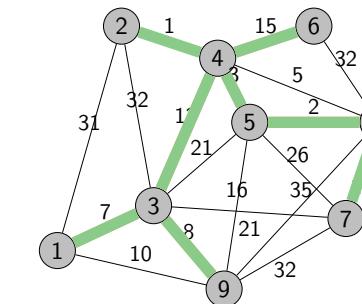
## Algorithmus von Prim

Der Algorithmus von Prim konstruiert ebenfalls einen MST.

### Algorithm 3: Algorithmus von Prim

```

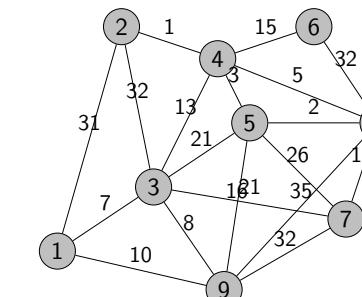
1 Function Prim(Graph G)
  Result: Minimum Spanning Tree T
  E(T) = ∅;
  V(T) = ein zufällig gewählter Startknoten;
  while V(T) ⊂ V(G) do
    Sei E' die Menge aller Kanten zwischen Knoten
    aus V(T) und V(G) \ V(T)
    e' = argmine ∈ E' w(e); // Kante mit kleinstem Gewicht
    E(T) = E(T) ∪ e';
    Füge neuen Knoten von e' zu V(T) hinzu;
```



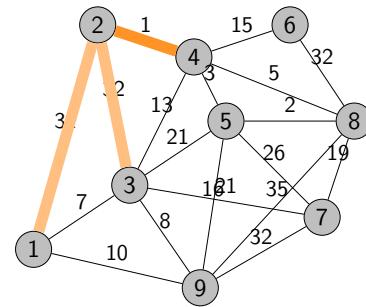
Aufsteigend sortierte Kanten:

$$\begin{aligned} w([2, 4]) &= 1, w([5, 8]) = 2, w([4, 5]) = 3, w([4, 8]) = 5 \\ w([1, 3]) &= 7, w([3, 9]) = 8, w([1, 9]) = 10, w([3, 4]) = 13 \\ w([4, 6]) &= 15, w([5, 9]) = 16, w([7, 8]) = 19, w([3, 5]) = 21 \\ w([3, 7]) &= 21, w([5, 7]) = 26, \dots \end{aligned}$$

## Algorithmus von Prim

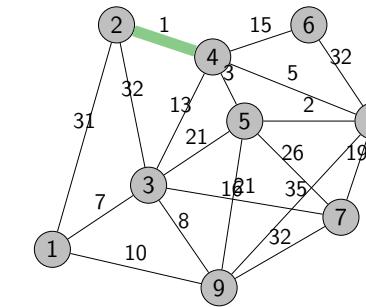


## Algorithmus von Prim



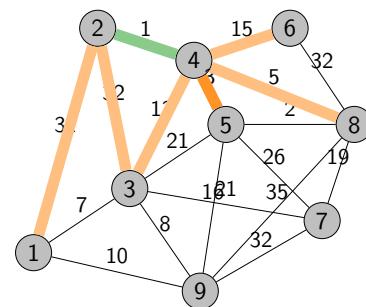
Startknoten: 2

## Algorithmus von Prim



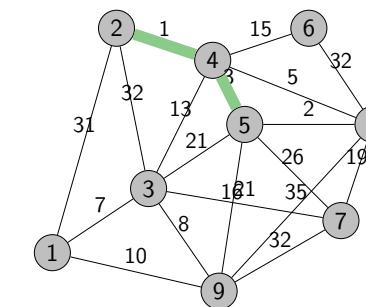
Startknoten: 2

## Algorithmus von Prim



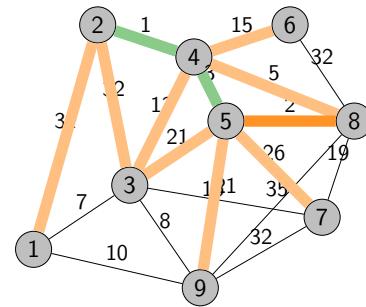
Startknoten: 2

## Algorithmus von Prim



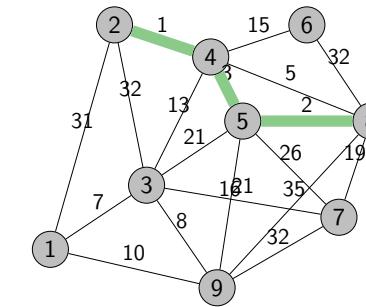
Startknoten: 2

## Algorithmus von Prim



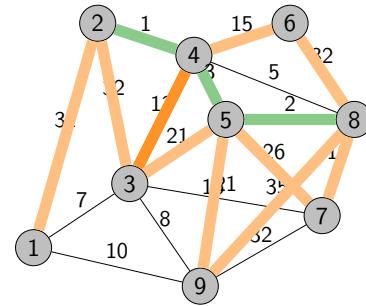
Startknoten: 2

## Algorithmus von Prim



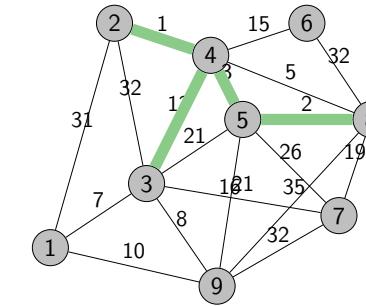
Startknoten: 2

## Algorithmus von Prim



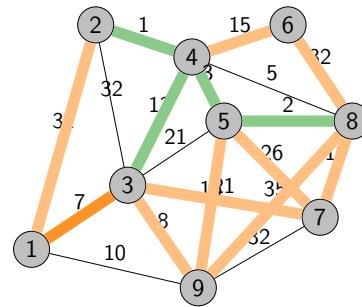
Startknoten: 2

## Algorithmus von Prim



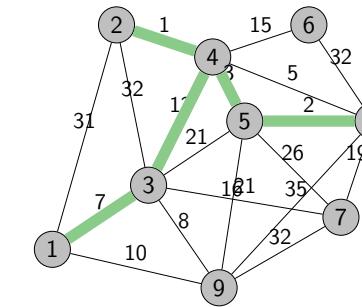
Startknoten: 2

## Algorithmus von Prim



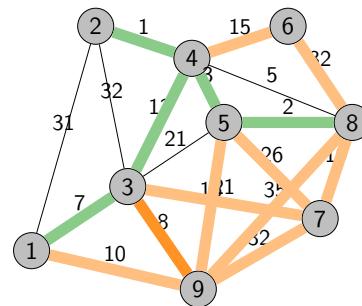
Startknoten: 2

## Algorithmus von Prim



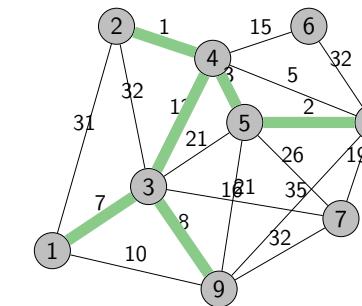
Startknoten: 2

## Algorithmus von Prim



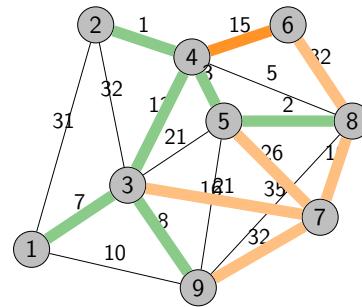
Startknoten: 2

## Algorithmus von Prim



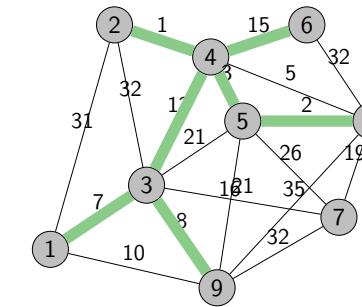
Startknoten: 2

## Algorithmus von Prim



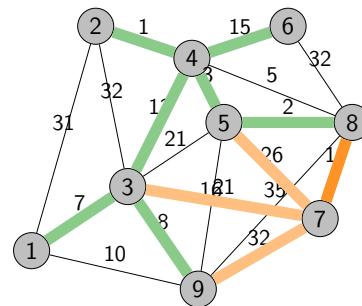
Startknoten: 2

## Algorithmus von Prim



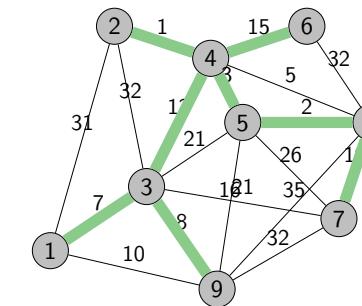
Startknoten: 2

## Algorithmus von Prim



Startknoten: 2

## Algorithmus von Prim



Startknoten: 2

## Vergleich: Kruskal vs. Prim

- Beide Algorithmen (Kruskal und Prim) finden den MST in gewichteten Graphen.
- Bei *dicht*<sup>2</sup> besetzten Graphen ist der Algorithmus von Prim jedoch effizienter.
- Bei *dünn*<sup>3</sup> besetzten Graphen ist Kruskal besser.
- Beim Algorithmus von Kurskal müssen die Kanten vorab sortiert werden, was bei vielen Kanten einen erheblichen Aufwand darstellt (sogar den Hauptaufwand des gesamten Verfahrens).
- Bei vergleichsweise wenigen Kanten überwiegen jedoch die Vorteile der einfacheren darauffolgenden Schritte.

<sup>2</sup>|E| ∈ O(|V|^2), d.h. die Anzahl der Kanten liegt in der Größenordnung der Anzahl der Kanten eines vollständigen Graphen.

<sup>3</sup>|E| ∈ O(|V|), d.h. die Anzahl der Kanten liegt in der Größenordnung der Anzahl der Knoten; der Graph enthält also relativ wenige Kanten.

## Zusammenfassung (MST)

- Beide Algorithmen (Kruskal, Prim) sind sogenannte **Greedy-Algorithmen**
- Sie wählen in jedem Schritt ("gierig") die nächst-besten Erweiterungen der Teillösung
- Normalerweise erreicht man mit einer derartigen Strategie nur **Näherungslösungen** (d.h. nicht die insgesamt beste Lösung)
- In diesem Fall finden jedoch beide Greedy-Algorithmen das **globale Optimum**, d.h. die insgesamt beste Lösung
- ...der Minimale Spannbaum kann also (wie der Euler-Zyklus) leicht gefunden werden.
- Andere Spannbaum-Probleme (Varianten) sind jedoch wesentlich schwieriger!

## Determinante

# Graphentheorie:

Berechnung der Anzahl an Spannbäumen/Gerüsten eines Graphen

## Programmieren und Software-Engineering Theorie

2. September 2025

### Definition (Determinante)

In der linearen Algebra ist eine Determinante eine Zahl die einer quadratischen Matrix zugeordnet werden kann.

**Anmerkung:** Determinanten können beispielsweise als Volumensänderung interpretiert werden, die sich durch die *lineare Abbildung* ergibt, die durch die quadratische Matrix festgelegt wird.

## Berechnung von Determinanten

- Berechnung der Determinante einer 2x2 Matrix:

$$\det A = \begin{vmatrix} a & b \\ c & d \end{vmatrix} = ad - bc$$

- Berechnung einer Determinante einer 3x3 Matrix  
**(Regel von Sarrus):**

$$\begin{vmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{vmatrix} = a_{11}a_{22}a_{33} + a_{12}a_{23}a_{31} + a_{13}a_{21}a_{32} - a_{13}a_{22}a_{31} - a_{12}a_{21}a_{33} - a_{11}a_{23}a_{32}$$

## Laplacescher Entwicklungssatz

### Definition (Laplacescher Entwicklungssatz)

$$\det A = \sum_{i=1}^n (-1)^{i+j} \cdot a_{ij} \cdot \det A_{ij} \quad (\text{Entwicklung nach der } j\text{-ten Spalte})$$

bzw. alternativ:

$$\det A = \sum_{j=1}^n (-1)^{i+j} \cdot a_{ij} \cdot \det A_{ij} \quad (\text{Entwicklung nach der } i\text{-ten Zeile})$$

wobei  $A_{ij}$  die  $(n-1) \times (n-1)$ -Untermatrix (**Minor**) von  $A$  ist, die sich durch Streichen der  $i$ -ten Zeile und  $j$ -ten Spalte in  $A$  ergibt.

**Anmerkung:** Das Produkt  $\tilde{a}_{ij} := (-1)^{i+j} \det A_{ij}$  wird Kofaktor  $\tilde{a}_{ij}$  genannt.

## Beispiel

Wir berechnen die folgende Determinante sowohl mit dem Laplaceschen Entwicklungssatz, als auch mit der Regel von Sarrus:

$$\det A = |A| = \begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix}.$$

Die Entwicklung nach der ersten Zeile ergibt

$$\det A = a \cdot \begin{vmatrix} e & f \\ h & i \end{vmatrix} - b \cdot \begin{vmatrix} d & f \\ g & i \end{vmatrix} + c \cdot \begin{vmatrix} d & e \\ g & h \end{vmatrix},$$

und ausrechnen dieses Terms ergibt schließlich

$$= a \cdot e \cdot i - a \cdot f \cdot h - b \cdot d \cdot i + b \cdot f \cdot g + c \cdot d \cdot h - c \cdot e \cdot g,$$

was wir auch direkt mit der Regel von Sarrus erhalten.

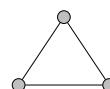
POS (Theorie)

Kirchhoff

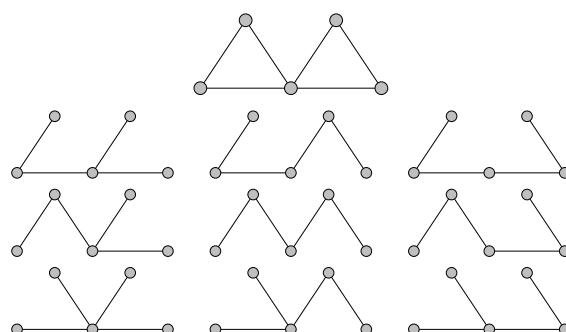
5 / 10

## Anzahl an Spannbäumen

**Beispiel:** Anzahl der Spannbäume dieses Graphen? 3



**Beispiel:** Anzahl der Spannbäume dieses Graphen? 9



POS (Theorie)

Kirchhoff

7 / 10

## Alternativ: Gauß-Verfahren

- Ist eine  $n \times n$  Matrix  $M$  in der oberen Dreiecksform gegeben, so ist  $\det M = \prod_{i=1}^n m_{ii}$
- Mit dem Gauß-Verfahren kann jede quadratische Matrix auf diese Form gebracht werden.
- Bei einer Zeilenumtauschung ändert sich die Determinante um den Faktor  $-1$ .
- Bei Multiplikation einer Zeile mit  $c$  ändert sich der Wert der Determinante ebenfalls um den Faktor  $c$ .
- Die Addition des Vielfachen einer Zeile der Matrix zu einer anderen Zeile ändert den Wert der Determinante nicht!

POS (Theorie)

Kirchhoff

6 / 10

## Anzahl an Spannbäumen

**Lemma (Anzahl an Spannbäumen eines Graphen)**

Die Anzahl der Spannbäume eines zusammenhängenden (schlichten) Graphen erhält man durch Multiplikation der Anzahl der Spannbäume aller Blöcke des Graphen.

Wie berechnet man die Anzahl der Spannbäume in einem Block?  
 ⇒ Satz von Kirchhoff

Kirchhoff

8 / 10

## Anzahl an Spannbäumen

Sei  $A$  die Adjazenzmatrix eines Graphen  $G$ . Sei weiters  $D$  eine Matrix mit  $d_{ii} = d(i)$  für alle  $i \in V$ , und  $d_{ij} = 0$  für  $i \neq j$ . D.h.  $D$  ist eine Matrix die in der Hauptdiagonale die Knotengrade als Einträge enthält, und sonst lauter 0er.

### Definition (Laplace-Matrix)

- $L := D - A$
- Sei nun  $L^*$  die Matrix die aus  $L$  entsteht, indem eine beliebige Zeile und beliebige Spalte gelöscht werden.
- **Anmerkung:**  $L^*$  ist also eine  $(n - 1) \times (n - 1)$ -Matrix

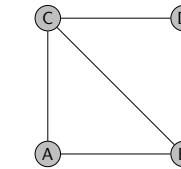
### Satz von Kirchhoff

Die Anzahl der Spannbäume von  $G$  ist gegeben durch

$$\det L^*$$

## Anzahl an Spannbäumen

**Beispiel:** Anzahl der Spannbäume vom Graphen



$$L = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix}$$

Damit erhalten wir  $L^* = \begin{pmatrix} 2 & -1 & 0 \\ -1 & 3 & -1 \\ 0 & -1 & 2 \end{pmatrix}$  und  $\det L^* = 8$ .

## WH: Zusammenhangskomponenten

# Starke Zusammenhangskomponente

## Programmieren und Software-Engineering Theorie

2. September 2025

POS (Theorie)

Starker Zusammenhang

1 / 21

POS (Theorie)

Starker Zusammenhang

2 / 21

Starke Zusammenhangskomponente  
○○○○

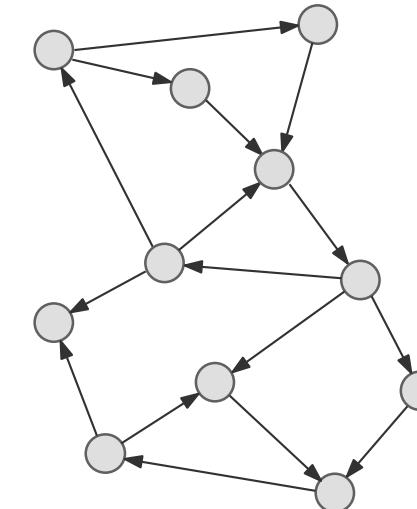
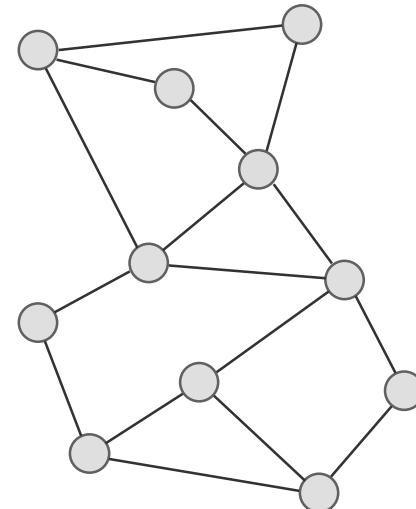
Algorithmus  
○○○○○

Korrektheitsbeweis  
○○○○○○○○

Starke Zusammenhangskomponente  
○○○○

Algorithmus  
○○○○○

Korrektheitsbeweis  
○○○○○○○○



POS (Theorie)

Starker Zusammenhang

3 / 21

POS (Theorie)

Starker Zusammenhang

4 / 21

### Definition 1 (Zusammenhangskomponenten)

In einem ungerichteten Graphen  $G$  ist eine *Zusammenhangskomponente* ein maximaler, zusammenhängender Teilgraph. Zwei Knoten  $u$  und  $v$  sind in der selben *Zusammenhangskomponente*, genau dann wenn  $u \rightsquigarrow v$ . <sup>a</sup>

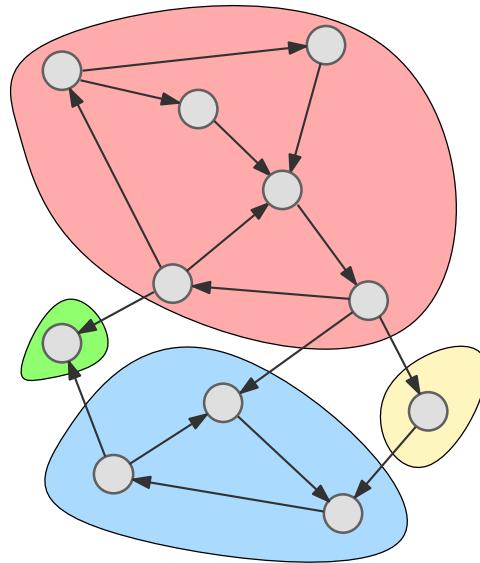
<sup>a</sup> $u \rightsquigarrow v$  bedeutet, dass  $v$  von  $u$  aus erreichbar ist. Im ungerichteten Graphen impliziert dies  $v \rightsquigarrow u$ .

Wie kann der Begriff der *Zusammenhangskomponente* auf gerichtete Graphen übertragen werden?

Starke Zusammenhangskomponente  
○○○○○

Algorithmus  
○○○○○

Korrektheitsbeweis  
○○○○○○○○○



POS (Theorie)

Starker Zusammenhang

5 / 21

Starke Zusammenhangskomponente  
○○○○○

Algorithmus  
●○○○○

Korrektheitsbeweis  
○○○○○○○○○

#### Definition 4 (Transponierter Graph $G^T$ )

Sei  $G^T = (V, A^T)$  der Graph der aus dem gerichteten Graphen  $G = (V, A)$  entsteht indem alle gerichteten Kanten "umgedreht" werden, also  $A^T = \{(j, i) \mid (i, j) \in A\}$ .

Aufruf der Tiefensuche  $\text{DFS}(G)$ :

- Im ersten Aufruf werden nicht unbedingt alle Knoten erreicht
- Die Tiefensuche wird dann so lange für die verbleibenden (unbesuchten) Knoten aufgerufen, bis alle Knoten besucht sind.
- Auch für erneuten Aufruf gilt: besuchte Knoten werden nicht erneut besucht
- Wenn zu jedem Knoten eine gerichtete Kante vom Vorgänger auf diesen Knoten gespeichert wird, entsteht **Tiefensuch-Wald** (mehrere gerichtete Bäume)

Starke Zusammenhangskomponente  
○○○○●

Algorithmus  
○○○○○

Korrektheitsbeweis  
○○○○○○○○○

## Zusammenhang in gerichteten Graphen

### Definition 2 (Schwacher Zusammenhang)

Man spricht von *schwachem Zusammenhang* in einem gerichteten Graphen, wenn der zugrunde liegende ungerichtete Graph ("Schatten") zusammenhängend ist.

### Definition 3 (Starke Zusammenhangskomponente)

Eine *Starke Zusammenhangskomponente* ist eine maximale Teilmenge an Knoten  $U \subseteq V$  mit der Eigenschaft, dass für alle Paare an Knoten  $u, v \in U$  gilt, dass sowohl  $u \rightsquigarrow v$  als auch  $v \rightsquigarrow u$ .

POS (Theorie)

Starker Zusammenhang

7 / 21

POS (Theorie)

Starker Zusammenhang

6 / 21

Starke Zusammenhangskomponente  
○○○○○

Algorithmus  
○●○○○

Korrektheitsbeweis  
○○○○○○○○○

## Algorithmus zur Berechnung der starken Zusammenhangskomponenten

### Algorithmus von Kosaraju-Shahir

- ① Aufruf von  $\text{DFS}(G) \Rightarrow \tau_f(v)$
- ② Berechne  $G^T$
- ③ Aufruf von  $\text{DFS}(G^T)$  für Knoten  $v \in V$  in absteigender Reihenfolge bezüglich  $\tau_f(v)$  aus Schritt (1).
- ④ Die Knotenmengen die in einem Aufruf der DFS in Schritt (3) gefunden werden entsprechen den starken Zusammenhangskomponenten.

POS (Theorie)

Starker Zusammenhang

8 / 21

POS (Theorie)

Starker Zusammenhang

8 / 21

Starke Zusammenhangskomponente  
○○○○○

Algorithmus  
○○●○○○

Korrektheitsbeweis  
○○○○○○○○

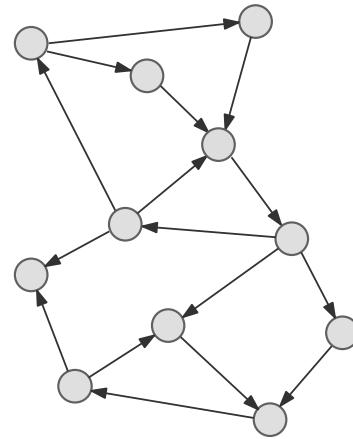


Abbildung: Beispiel: Bestimmung der starken Zusammenhangskomponenten

POS (Theorie)

Starker Zusammenhang

9 / 21

Starke Zusammenhangskomponente  
○○○○○

Algorithmus  
○○○○●○

Korrektheitsbeweis  
○○○○○○○○

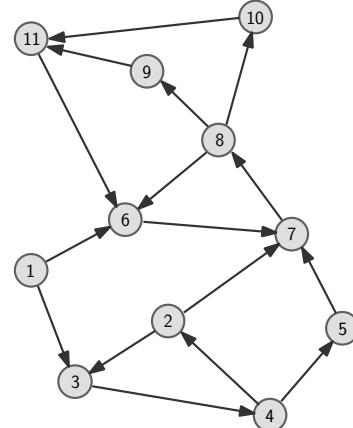


Abbildung: Nach dem ersten Aufruf der DFS wird der transponierte Graph  $G^T$  von  $G$  berechnet.

POS (Theorie)

Starker Zusammenhang

11 / 21

Starke Zusammenhangskomponente  
○○○○○

Algorithmus  
○○○●○○

Korrektheitsbeweis  
○○○○○○○○

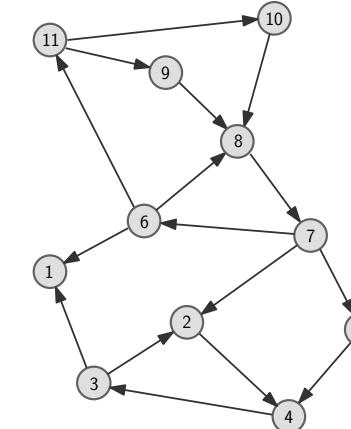


Abbildung: Die Werte in den Knoten entsprechen den Fertigstellungszeiten  $\tau_f$  (Bem.  $\tau_d$ 's werden hier vernachlässigt)

POS (Theorie)

Starker Zusammenhang

10 / 21

Starke Zusammenhangskomponente  
○○○○○

Algorithmus  
○○○○○●

Korrektheitsbeweis  
○○○○○○○○

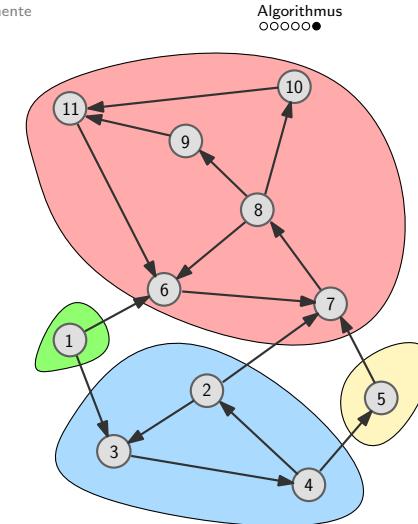


Abbildung: Die DFS Aufrufe in Schritt (3) liefern als Ergebnis die Starken Zusammenhangskomponenten von  $G^T$  (und somit von  $G$ ).

POS (Theorie)

Starker Zusammenhang

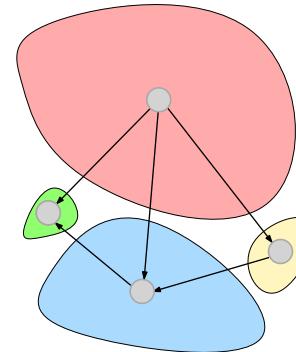
12 / 21

**Lemma 5**

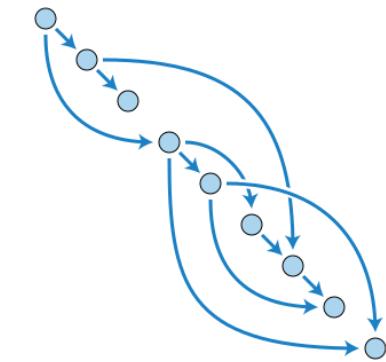
Sei  $G' = (V', E')$  der Graph der daraus entsteht indem in  $G$  jede Starke-Zusammenhangs-Komponente (SZK) in einen einzelnen Knoten kontrahiert wird.  $G'$  ist dann ein gerichteter azyklischer Graph, (engl. directed acyclic graph, DAG).

**Beweis:**

- Angenommen es gibt einen Kreis  $C_1, C_2, \dots, C_n$  mit  $C_i \in V'$
- $\Rightarrow$  Dann existiert eine gerichtete Kante  $a = (i, j)$  mit  $i \in C_k$  und  $j \in C_{k+1}$  (und jeweils  $i \in C_n$  und  $j \in C_1$ ) für  $1 \leq k \leq n - 1$
- Es existiert ein Pfad  $P$  von jedem Knoten in  $C_i$  zu jedem Knoten in  $C_j$  für alle  $i, j \in \{1 \dots n\}$ .
- Alle  $v \in V(\bigcup_{i \in \text{Kreis}} C_i)$  gehören zur selben SZK.  $\square$



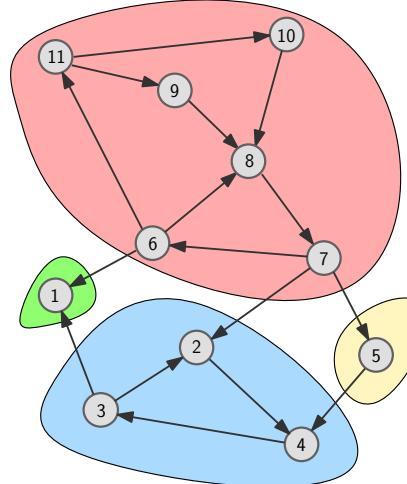
- Ein DAG enthält keinen gerichteten Kreis.
- Die Knoten können so angeordnet werden, dass alle Kanten von links nach rechts verlaufen.
- Diese Anordnung wird auch *topologische Sortierung* genannt.
- $\Rightarrow$  Jeder DAG enthält mindestens einen Knoten  $v$  mit  $d^+(v) = 0$ , und mindestens einen Knoten  $v$  mit  $d^-(v) = 0$ .

**Lemma 6**

Sei  $C$  eine SZK von  $G$  ohne auslaufenden Kanten. Ein DFS-Aufruf für einen Knoten  $v \in C$  besucht genau alle Knoten  $u \in C$ .

**Beweis:**

- Sei  $v$  ein beliebiger Knoten in  $C$ .
- $\forall u \in C : v \rightsquigarrow u$ .
- Da  $C$  keine auslaufenden Kanten hat, sind keine weiteren Knoten erreichbar.  $\square$

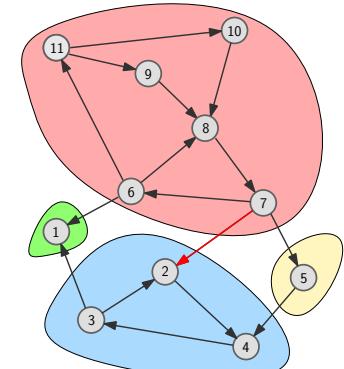
**Lemma 7**

Seien  $C_1$  und  $C_2$  zwei SZK von  $G$ . Weiters sei  $a = (i, j)$  eine gerichtete Kante von einer Komponente in die andere, also  $i \in C_1, j \in C_2$ . Sei  $v^*$  der erste Knoten in  $C_1$  der von DFS besucht wird. Dann gilt:  $\tau_f(v^*) > \tau_f(v_k) \forall v_k \in C_2$ .

**Beweis:**

**case 1:** DFS wird für einen Knoten  $v \in C_1$  aufgerufen bevor sie für irgend einen Knoten in  $C_2$  aufgerufen wird.

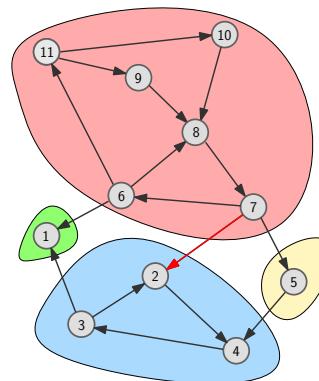
- Aufgrund der Annahme sind alle Knoten in  $C_1$  und  $C_2$  vom Knoten  $v$  aus erreichbar.
- DFS( $v$ ) ist fertig wenn alle Knoten in  $C_1$  und  $C_2$  abgeschlossen wurden.
- $\tau_f(v^*) > \tau_f(v_k) \forall v_k \in C_2$ .



**Proof:**

**case 2:** DFS wird für einen Knoten  $v \in C_2$  bevor es für irgend einen anderen Knoten in  $C_1$  aufgerufen wird.

- Es gibt keinen Weg von einem Knoten in  $C_2$  zu einem in  $C_1$ .
- DFS ist für alle Knoten in  $C_2$  abgeschlossen wenn es für den ersten Knoten in  $C_1$  aufgerufen wird.
- $\tau_f(v^*) > \tau_f(v_k) \quad \forall v_k \in C_2$ .



Die zusammenhängenden Komponenten des resultierenden Tiefensuch-Waldes in  $G^T$  (Schritt 3 des Algorithmus) entsprechen den SZK in  $G$ .

**Beweis:**

- **Teil 1:** Erster Aufruf der DFS in  $G^T$ :

Es folgt aus Lemma 8, dass DFS für die SZK ohne auslaufende Kanten (in  $G^T$ ) aufgerufen wird,  $\Rightarrow$  DFS besucht genau alle Knoten dieser SZK.

- **Teil 2:** weitere DFS-Aufrufe in  $G^T$ :

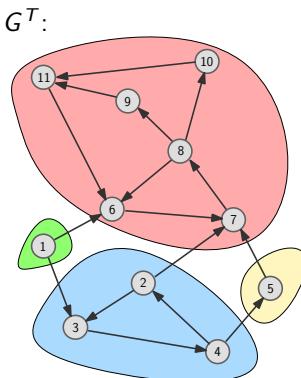
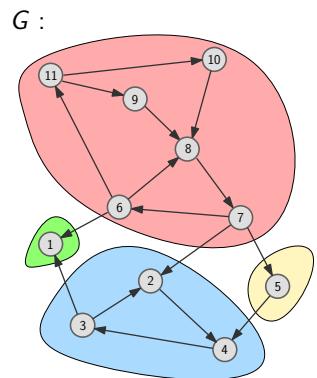
Sei  $v$  der Knoten für den DFS aufgerufen wird, und  $C_v$  die zugehörige SZK. Alle Knoten in  $C_v$  sind erreichbar und werden von diesem DFS-Aufruf besucht. Sei weiters  $w \rightsquigarrow v$ ,  $w \notin C_v$  (sondern in  $C_w$ ). Es wird nun gezeigt, dass  $w$  nicht in diesem DFS-Aufruf besucht wird.

In  $G$  gilt, dass  $w \rightsquigarrow v$ . Wenn  $w$  schon in einem vorangegangenen DFS-Aufruf besucht wurde, wird es nicht erneut besucht. Somit betrachten wir die (kritische) Situation, dass  $w$  noch unbesucht ist.

Es muss somit gelten, dass  $\max_{w' \in C_w}(\tau_f(w')) < \tau_f(v)$ . (Andernfalls wären ja alle Knoten von  $C_w$  schon zuvor besucht worden).

Es folgt somit, dass der DFS-Aufruf in  $G$  schon für alle Knoten  $u \in C_w$  abgeschlossen war, bevor  $v$  abgeschlossen wurde.

Da jedoch  $u \rightsquigarrow v \quad \forall u \in C_w$  in  $G$  folgt, dass ein Knoten  $u \in C_w$  existieren muss, für den  $\tau_f(u) > \tau_f(v)$  gilt, was ein Widerspruch zu der vorigen Aussage ist.



Anmerkung: Wenn also eine Kante in  $G^T$  von  $C_v$  nach  $C_w$  führt, müssen alle Knoten in  $C_w$  nach jenen aus  $C_v$  fertiggestellt worden sein, da ja in  $G$  in diesem Fall eine Kante von  $C_w$  nach  $C_v$  verlaufen ist. Somit müssen diese Knoten im Schritt 3 schon zuvor fertiggestellt worden sein (da ja die Aufrufe in absteigender Reihenfolge der Fertigstellungszeiten aus Schritt 1 erfolgt).

# Algorithmus von Dijkstra

## Programmieren und Software-Engineering Theorie

2. September 2025

## Algorithmus von Dijkstra

Der Algorithmus von Dijkstra berechnet die kürzesten Wege in einem gewichteten Graphen mit  $w_{ij} \geq 0$ , für alle  $[i, j] \in E$ .

### Grundidee:

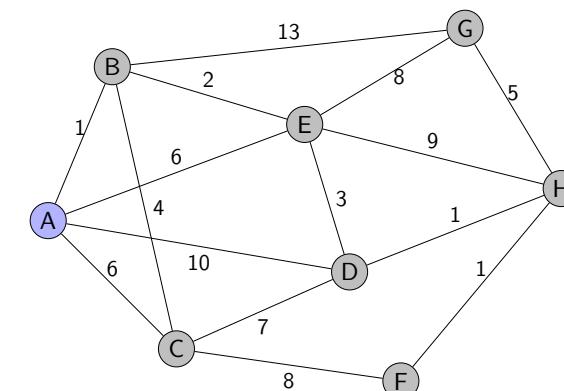
- Ähnlichkeit zu DFS, jedoch andere Regeln für die Auswahl des nächsten Knoten.
- Ein Aufruf von Dijkstra berechnet die kürzesten Wege von einem Startknoten zu allen anderen Knoten des Graphen.
- In jedem Schritt werden **Zwischenergebnisse**  $\delta_k, k \in V$  berechnet, bzw. aktualisiert.
- Diese Zwischenergebnisse sind die Länge des kürzesten *bisher gefundenen* Weges bis zu diesem Knoten.
- Wiederhole (bis alle Knoten abgeschlossen):
  - ① Wähle Knoten  $k$  mit minimalem  $\delta_k$  und schließe diesen ab.
  - ② Speichere **Verweis auf direkten Vorgänger**.
  - ③ Aktualisiere die Werte  $\delta_k$  für noch nicht abgeschlossene Nachbarknoten von  $k$ .

## Algorithmus von Dijkstra

### Algorithm 1: DIJKSTRA

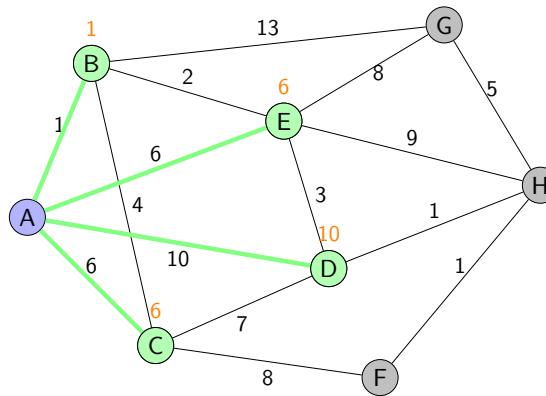
```
Data: Graph G mit  $w_{ij} \geq 0$  für alle  $(i, j) \in E(G)$ 
Data: Startknoten s
1  $\forall v \in V : \delta_v \leftarrow \infty;$ 
2  $\delta_s \leftarrow 0;$ 
3 Prioritätswarteschlange Q gefüllt mit allen Knoten ;
4 while  $Q \neq \emptyset$  do
5    $u \leftarrow Q.getMin();$  // entnimmt Knoten u mit kleinstem  $\delta_u$ 
6   Fertigstellung von Knoten u;
7   Speichere Verweis auf direkten Vorgänger von u;
8   for all  $(u, v) \in E, v$  noch nicht fertiggestellt do
9     if  $\delta_v > \delta_u + w_{uv}$  then
10        $\delta_v \leftarrow \delta_u + w_{uv};$ 
```

## Algorithmus von Dijkstra: Beispiel



**Wir suchen den kürzesten Weg vom Knoten A zum Knoten H.** Im ersten Schritt wird der Startknoten "fertiggestellt".

## Algorithmus von Dijksta: Beispiel



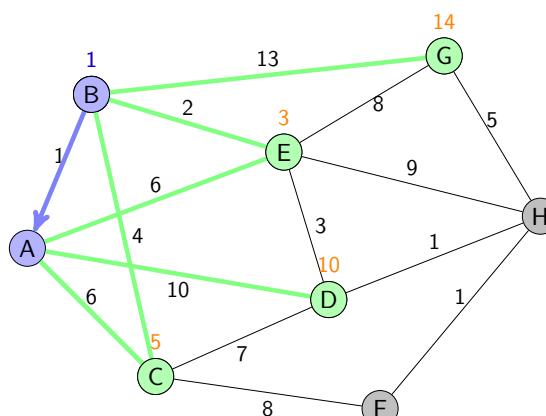
Im nächsten Schritt werden die Nachbarknoten  $B, C, D$  und  $E$  entdeckt. Die Zwischenwerte werden wie folgt berechnet:  
 $\delta_A = 0, \delta_B = \delta_A + 1 = 1, \delta_E = \delta_A + 6 = 6, \delta_D = \delta_A + 10 = 10, \delta_C = \delta_A + 6 = 6.$

POS (Theorie)

Algorithmus von Dijkstra

4 / 6

## Algorithmus von Dijksta: Beispiel



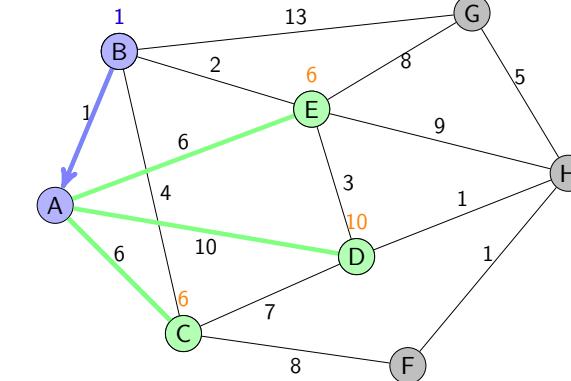
Ausgehend vom letzten fertiggestellten Knoten ( $B$ ) werden nun neue Zwischenergebnisse für  $C, E$  und  $G$  berechnet. Wir erhalten  $\delta_G = 1 + 13 = 14, \delta_E = 1 + 2 = 3, \delta_C = 1 + 4 = 5$ . Für die Knoten  $C$  und  $E$  erhalten wir kleinere Werte als die bisher gefundenen.

POS (Theorie)

Algorithmus von Dijkstra

4 / 6

## Algorithmus von Dijksta: Beispiel



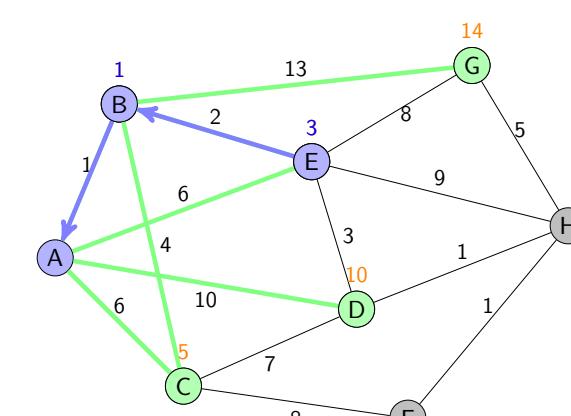
Nun wird der Knoten  $v$  mit kleinstem  $\delta_v$  fertiggestellt. Im konkreten Beispiel ist dies Knoten  $B$ .

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijksta: Beispiel



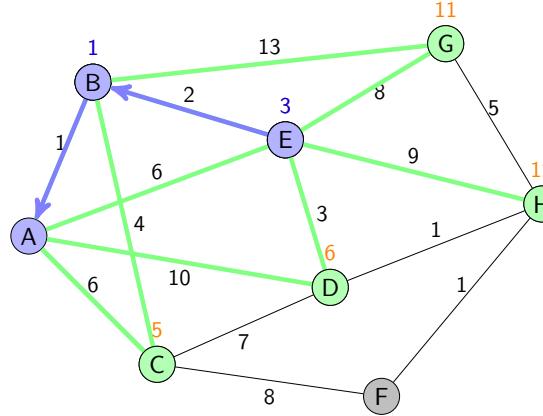
Knoten  $E$  wird fertiggestellt. Bei fertiggestellten Knoten merkt man sich wo man hergekommen ist (daher die blaue Kante).

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijkstra: Beispiel



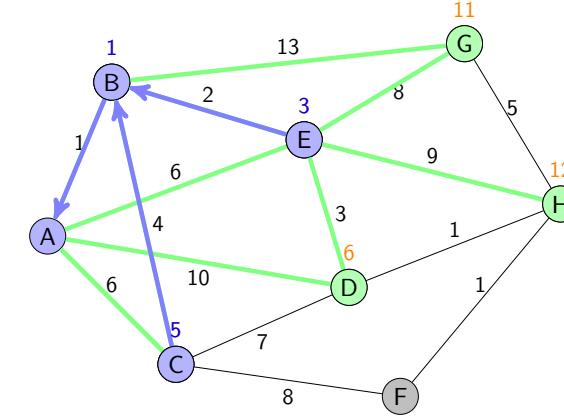
Berechnung neuer Zwischenergebnisse für  $D$ ,  $G$  und  $H$ .

POS (Theorie)

Algorithmus von Dijkstra

4 / 6

## Algorithmus von Dijkstra: Beispiel



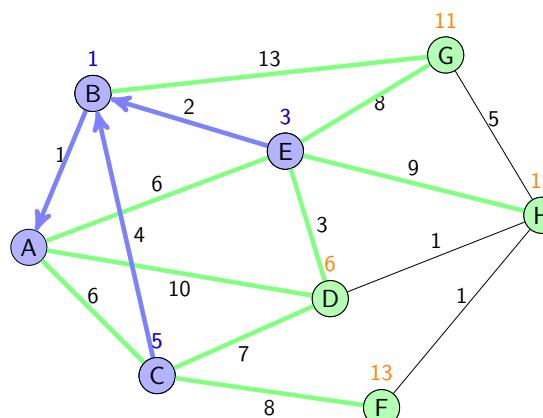
Knoten  $C$  wird fertiggestellt, da er nun den kleinsten Zwischenwert  $\delta$  hat.

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijkstra: Beispiel

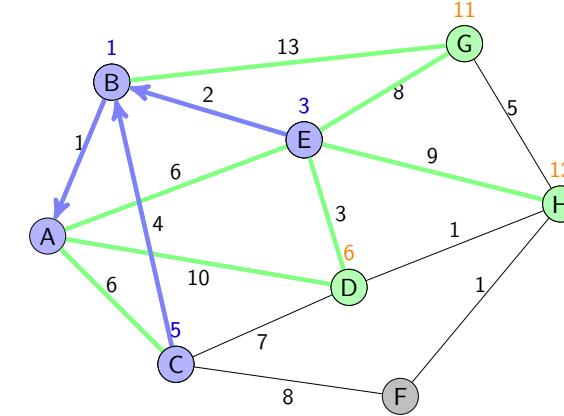


Ausgehend von  $C$  werden die Werte  $\delta_D$  und  $\delta_F$  berechnet. Da  $5 + 7 > 6$  kommt es bei  $\delta_D$  zu keiner Änderung.

POS (Theorie)

Algorithmus von Dijksta

4 / 6



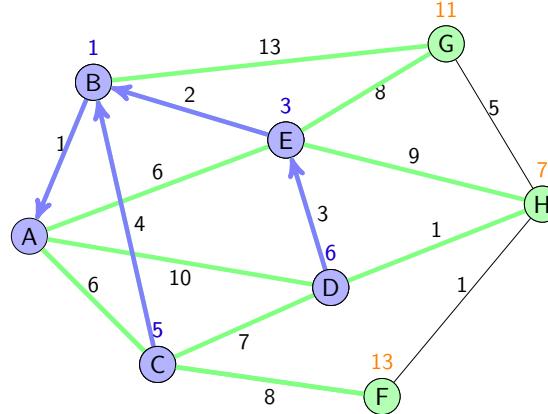
Fortgefahrene wird mit Knoten  $D$ , da kleinstes  $\delta$ .

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijksta: Beispiel



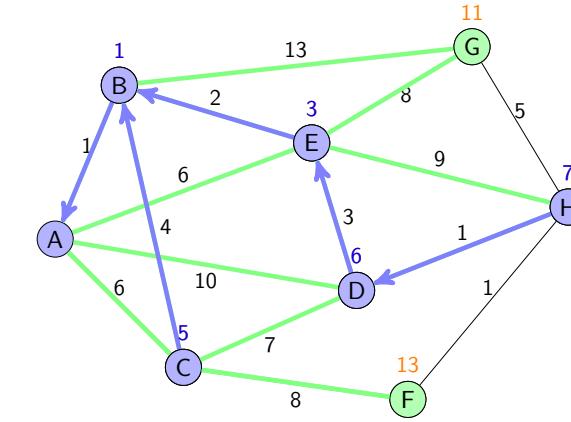
$\delta_H$  wird aktualisiert.

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijksta: Beispiel



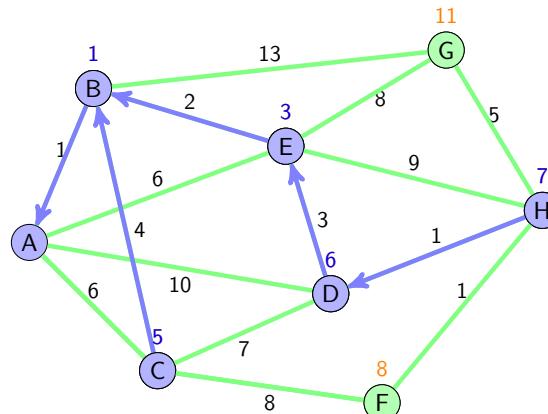
H wird fertiggestellt.

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijksta: Beispiel



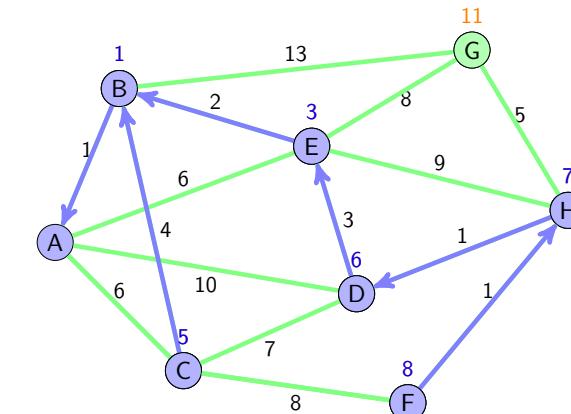
Update von  $\delta_F$  und  $\delta_G$ , wobei nur  $\delta_F$  tatsächlich geändert wird.

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijksta: Beispiel



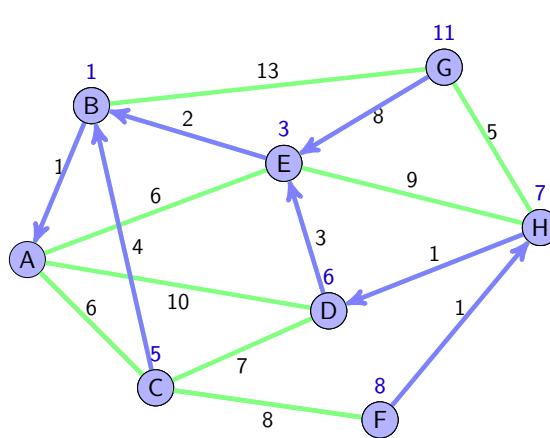
F wird fertiggestellt.

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijkstra: Beispiel



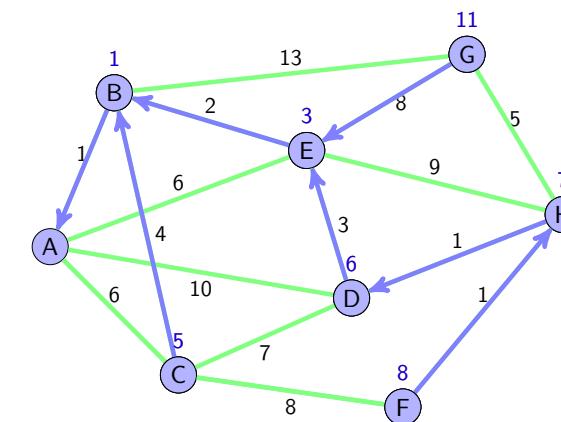
G wird fertiggestellt (Vorgänger E).

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijksta: Beispiel



Die blauen Kanten bilden einen Wurzelbaum, der die kürzesten Wege vom Startknoten zu jedem Knoten enthält.

POS (Theorie)

Algorithmus von Dijksta

4 / 6

## Algorithmus von Dijksta

- Die blauen Kanten bilden einen Wurzelbaum, der die kürzesten Wege vom Startknoten zu jedem Knoten enthält.
- Die Berechnung des kürzesten Weges von einem Startknoten zu einem Zielknoten beinhaltet also die Berechnung der kürzesten Wege vom Startknoten zu *allen* anderen Knoten.

POS (Theorie)

Algorithmus von Dijksta

5 / 6

## Algorithmus von Dijksta – Laufzeitanalyse

Mit  $n = |V|$  und  $m = |E|$  können wir die Laufzeiteigenschaften angeben. Diese hängen von der konkreten Umsetzung der Prioritätswarteschlange  $Q$  ab.

Name	Operation	Queue Implementierung		
		Liste	Heap	Fibonacci Heap <sup>1</sup>
decreaseKey [10]	$m$	$O(1)$	$O(\log n)$	$O(1)$
getMin [5]	$n$	$O(n)$	$O(\log n)$	$O(\log n)$
create [3]	1	$O(n)$	$O(n)$	$O(n)$
Gesamt		$O(n^2 + m)$ $= O(n^2)$	$O((n + m) \log n)$	$O(n \log n + m)$

Anmerkung: In der Spalte ganz links ist in eckigen Klammern auf die Zeile der jeweiligen Operation im Pseudocode verwiesen!

<sup>1</sup>amortisierte Laufzeit

## References I

- [1] Berger, Krieger, Mahr: "Grundlagen der elektronischen Datenverarbeitung", Skriptum
- [2] Reinhard Diestel: "Graph Theory", Springer, 5th edition