ADS: Algorithmen und Datenstrukturen 2

Teil 2

Prof. Dr. Gerhard Heyer

Institut für Informatik
Abteilung Automatische Sprachverarbeitung
Universität Leipzig

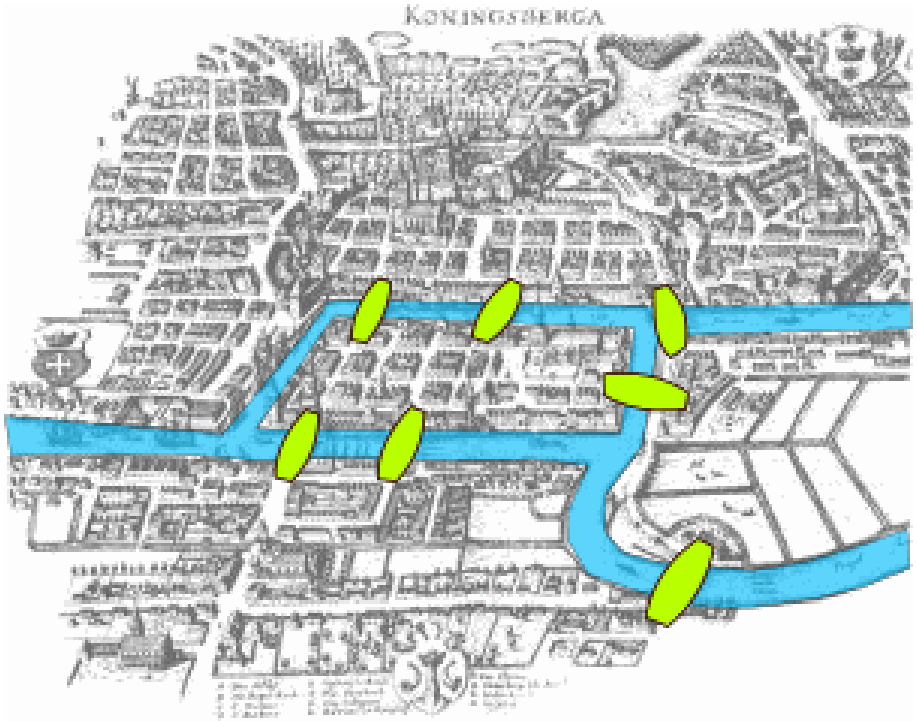
10. April 2019

[Letzte Aktualisierung: 10/04/2019, 12:46]

Graphen – Themenübersicht

1. Ungerichtete Graphen: Grundlegende Definitionen
2. Gewichtete, ungerichtete Graphen, minimale Spannbäume
3. Gerichtete Graphen: Definitionen, Speicherung, topologische Sortierung, transitive Hülle, starke Zusammenhangskomponenten, Zentralität
4. Gerichtete gewichtete Graphen: Kürzeste Pfade, Flußnetzwerke, kürzeste Wege

Königsberger Brückenproblem (Euler, 1736)



Gibt es einen Weg, bei dem man alle sieben Brücken genau einmal überquert? Ist dieser Weg als Rundweg möglich?

Originalartikel: `http://www.math.dartmouth.edu/~euler/pages/E053.html`

Ungerichteter Graph

Ein Tupel (V, E) heißt *(ungerichteter) Graph*, genau dann wenn

- V eine endliche Menge und
- E eine Menge ungeordneter Paare von Elementen in V ist.

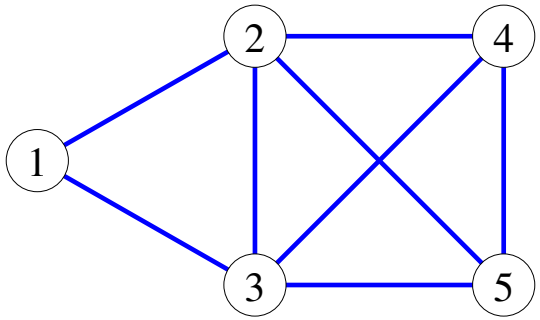
V heißt *Knotenmenge*, die Elemente von V heißen *Knoten*.
E heißt *Kantenmenge*, die Elemente von E heißen *Kanten*.

Ungerichteter Graph - Beispiel

Beispiel: $V = \{1, 2, 3, 4, 5\}$
$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$

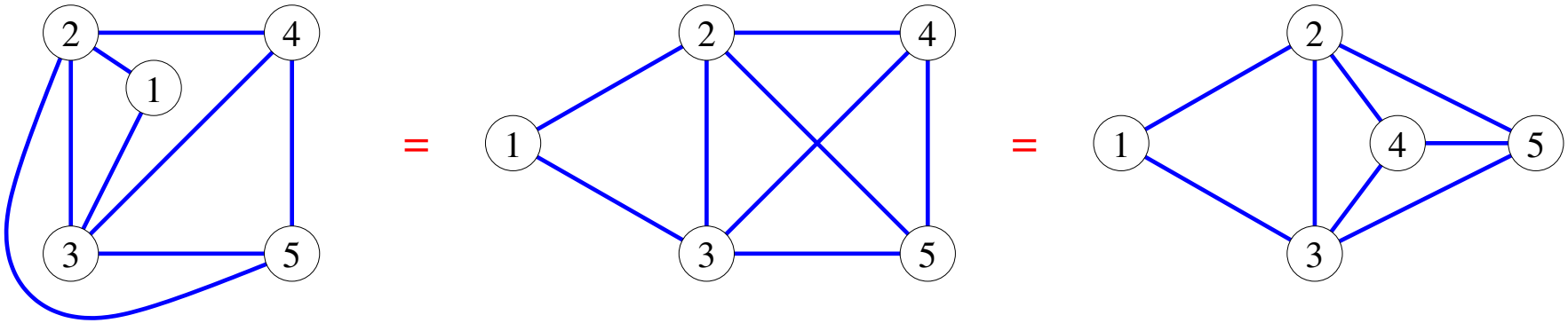
Ungerichteter Graph - Beispiel

Beispiel: $V = \{1, 2, 3, 4, 5\}$
$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$



Ungerichteter Graph - Beispiel

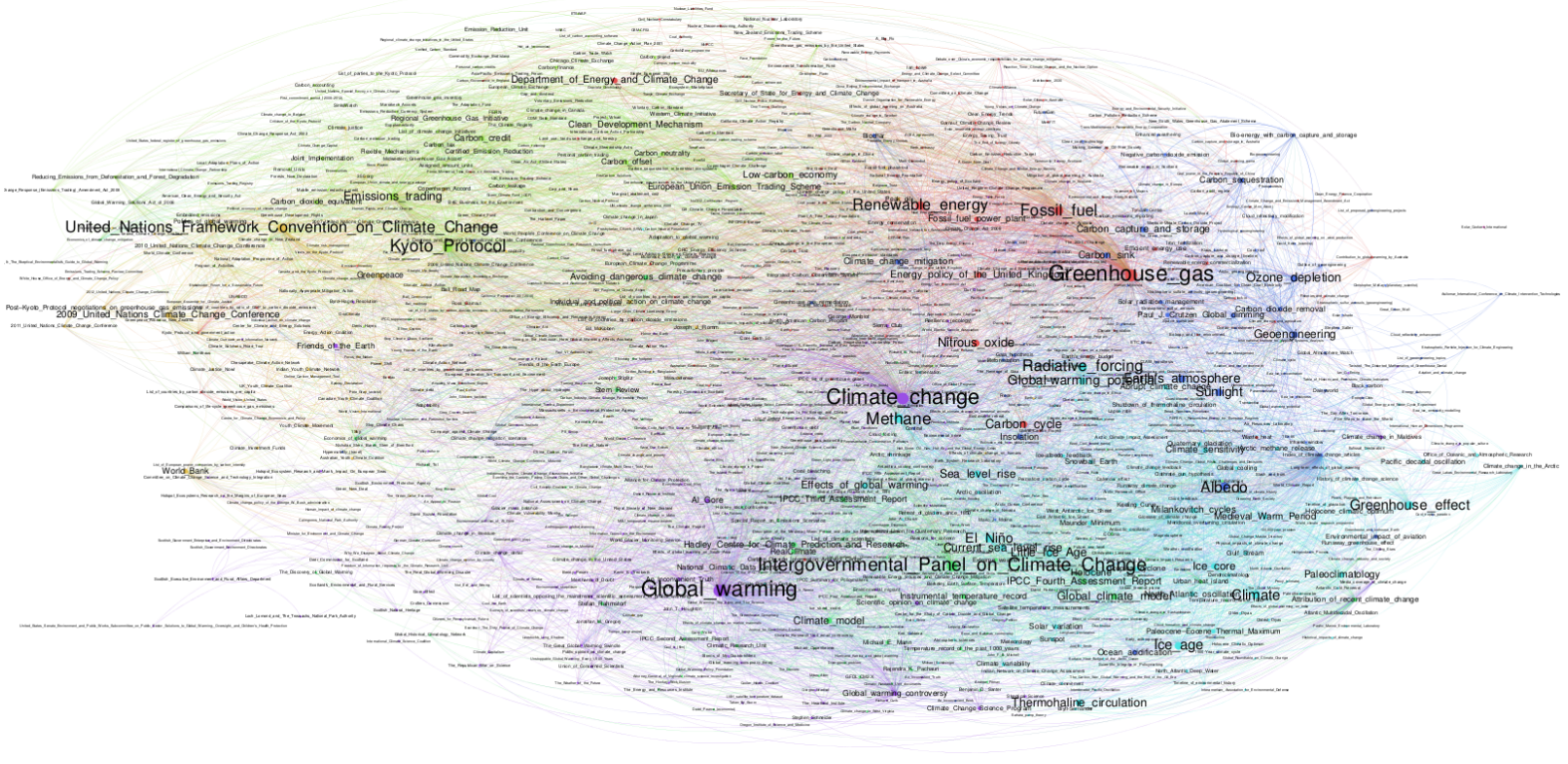
Beispiel: $V = \{1, 2, 3, 4, 5\}$
$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{3, 5\}, \{4, 5\}\}$



Graphen: Beispiele realer Systeme

System	Knoten	Kanten
Internet	Router	Datenleitungen
WWW	Webseiten/-dokumente	Hyperlinks
Gesellschaft	Personen	soziale Kontakte
Sprache	Wörter	gemeinsames Auftreten
Biotop	Spezies	trophische Bez., “Fressen”
Molekül	Atome	chem. Bindungen

Netzwerk von Artikeln über Klimawandel (Okt 2012)

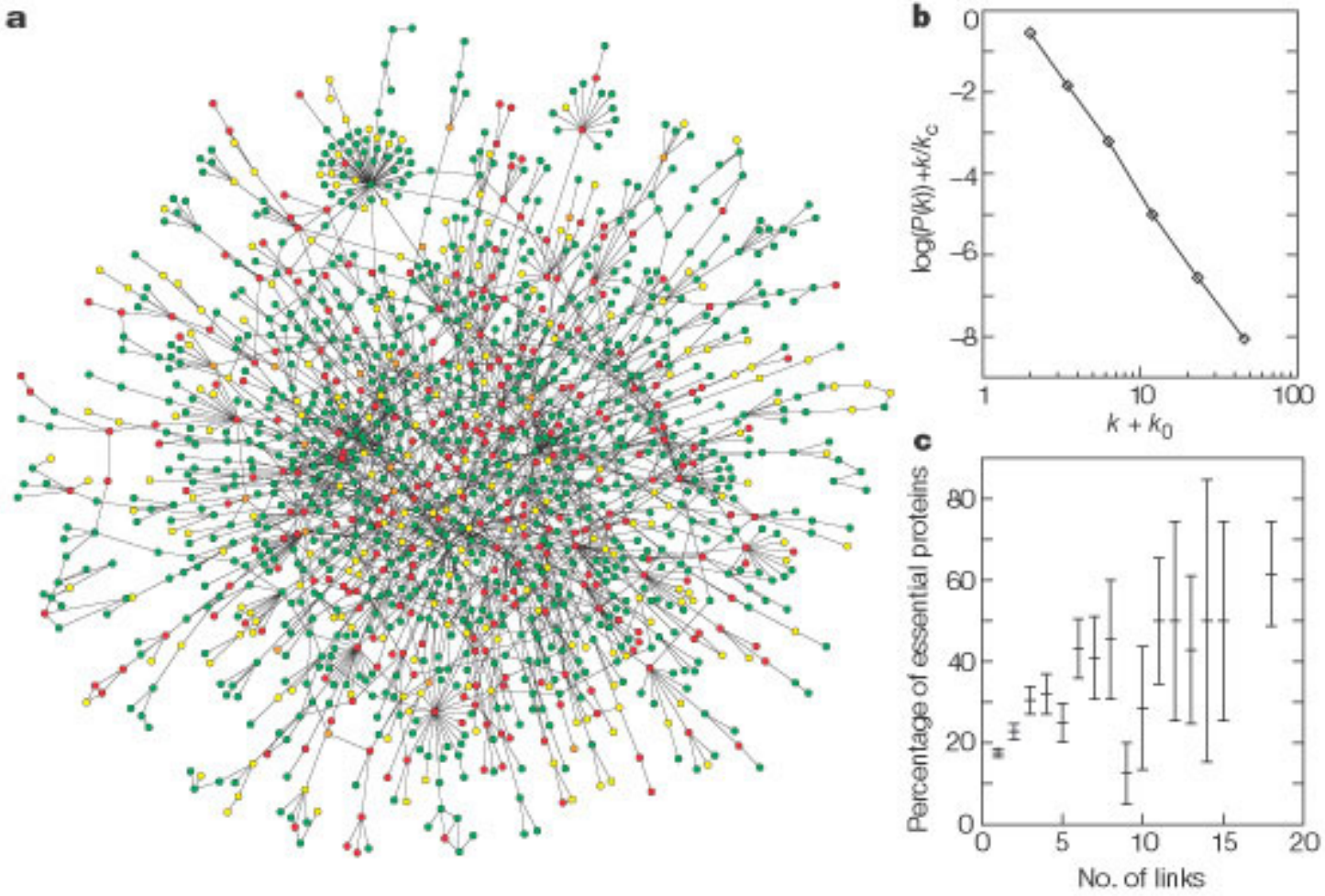


http://www.emapsproject.com/blog/archives/1572

Social Network (Facebook, 2010)

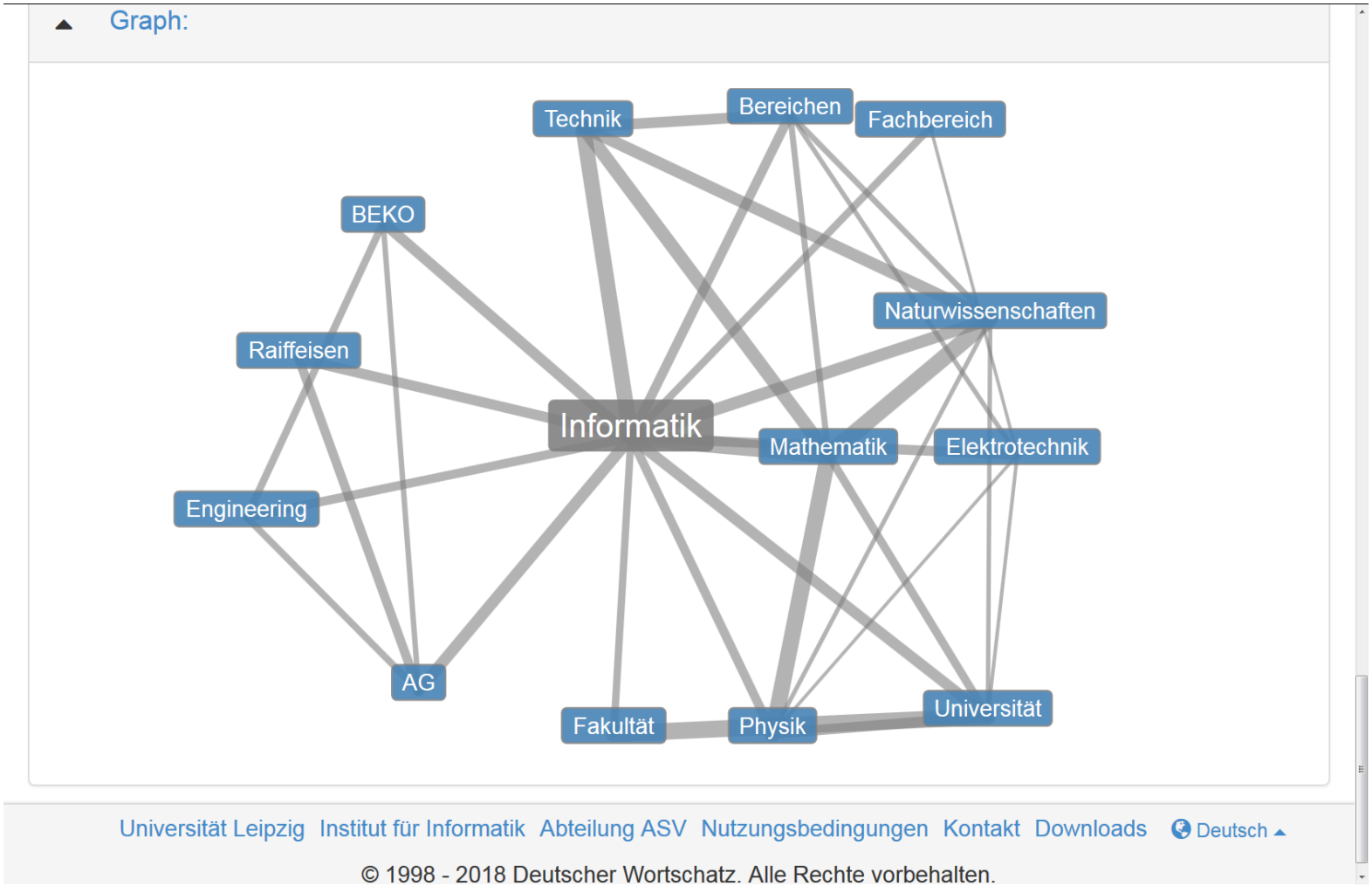
http://blog.revolutionanalytics.com/2010/12/facebooks-social-network-graph.html

Graph von Protein-Wechselwirkungen (Hefe)



Jeong, Mason, Barabási & Oltvai, *Nature* 2001.

Wortschatzgraph



http://corpora.uni-leipzig.de/de?corpusId=deu_newscrawl_2011

Teilgraphen

Seien $G = (V, E)$ und $G' = (V', E')$ Graphen.

- G' heißt *Teilgraph* von G, wenn $V' \subseteq V$ und $E' \subseteq E$ ist.
- G' heißt *aufspannender Teilgraph* von G, wenn G' Teilgraph von G mit $V' = V$ ist (G' enthält dieselben Knoten wie G).
- G' heißt *induzierter Teilgraph* von G, wenn G' Teilgraph von G ist und für alle $e \in E$ gilt: $e \subseteq V' \Rightarrow e \in E'$. (Alle Kanten aus G, deren zwei Knoten in G' liegen, sind auch in G enthalten.)

Pfade, Zyklen, Zusammenhang

Sei $G = (V, E)$ ein Graph, $\ell \in \mathbb{N}$ und $k = (v_0, v_1, \ldots, v_\ell) \in V^{\ell+1}$.

- k heißt *Weg* der Länge ℓ, wenn für alle $i \in \{1, \ldots, \ell\}$ gilt: $\{v_{i-1}, v_i\} \in E$
- k heißt *Pfad* der Länge ℓ, wenn k ein Weg ist und für alle $i, j \in \{0, \ldots, \ell\}$ mit $i \neq j$ gilt: $v_i \neq v_j$.
- k heißt *Zyklus* (oder *Kreis*) der Länge ℓ, wenn $(v_1, \ldots, v_\ell)$ ein Pfad der Länge $\ell - 1$ ist, $v_0 = v_\ell$ und $\{v_0, v_1\} \in E$.
- G heißt *zusammenhängend*, wenn für alle $x, y \in V$ ein Pfad zwischen x und y existiert.

Wälder und Bäume

Sei $G = (V, E)$ ein ungerichteter Graph.

- G heißt *Wald* (oder *zyklenfrei*), wenn kein Weg in G ein Zyklus ist.
- G heißt *Baum*, wenn G ein Wald ist und G zusammenhängend ist.

Satz: Ist G ein Baum, so hat G genau $|V| - 1$ Kanten.

Satz: Ist G zusammenhängend, so hat G einen aufspannenden Teilgraphen T, so daß T ein Baum ist. T heißt dann *Spannbaum* von G.

Spannbaum - Beispiel

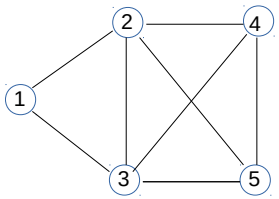
Ein **Spannbaum** ist

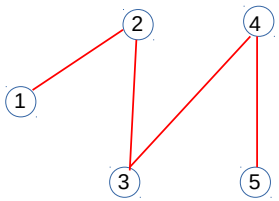
1. Teilgraph eines ungerichteten Graphen
2. der ein Baum ist und
3. alle Knoten dieses Graphen enthält

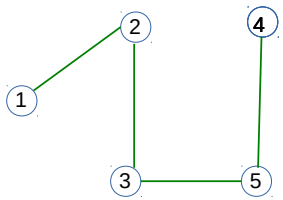
Spannbaum - Beispiel

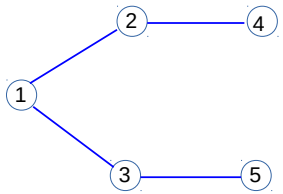
Ein **Spannbaum** ist

1. Teilgraph eines ungerichteten Graphen
2. der ein Baum ist und
3. alle Knoten dieses Graphen enthält







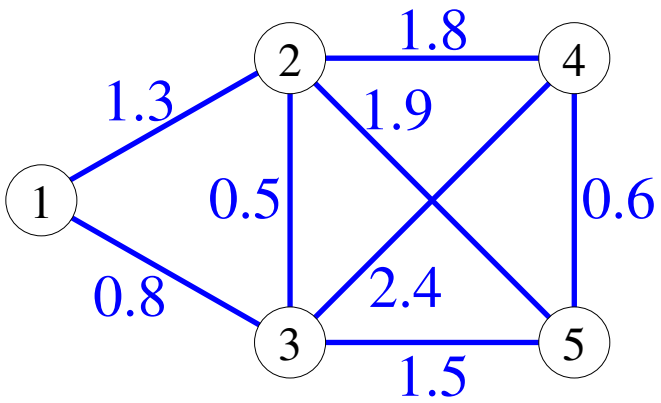


Gewichteter Graph

Sei (V, E) ein Graph und $w : E \to \mathbb{R}$.

- Das Tripel $G = (V, E, w)$ heißt *gewichteter* (oder *kantenbewerteter*) Graph.
- $w(e)$ heißt *Gewicht* (oder Länge) der Kante $e \in E$.

Beispiel:



Minimaler Spannbaum

Gegeben: Gewichteter zusammenhängender Graph $G = (V, E, w)$.

Gesucht: Spannbaum $T = (V, F)$ mit *minimaler Kantensumme*. Wähle also die Kantenmenge $F \subseteq E$ so, daß

$$\sum_{e \in F} w(e)$$

möglichst klein wird.

Kruskal-Algorithmus (1956)

Minimaler Spannbaum T für $G = (V, E, w)$.

Initialisiere F als leere Menge.

Erzeuge Liste L der Kanten in E; Sortiere L aufsteigend nach Gewicht.

Solange L nicht leer ist:

- Entferne Kante $e = \{u, v\}$ mit kleinstem Gewicht aus L.
- Falls (V, F) keinen Pfad zwischen u und v enthält:

$$F := F \cup \{e\}$$

(Sonst: tue nichts.)

Ergebnis: F, bzw. $T = (V, F)$

Wir verwenden dabei einen *minHeap* und eine *UnionFind* Datenstruktur

Union-Find-Datenstruktur

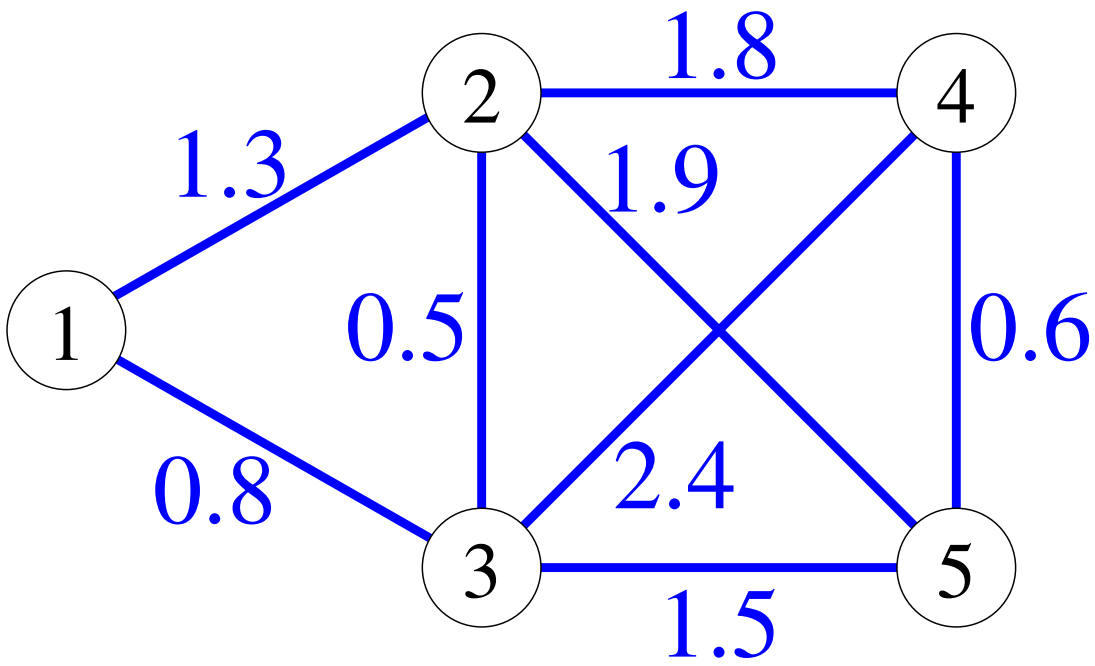
Gegeben:

- endliche Grundmenge E
- Familie S $= \{S_1, S_2, ...S_n\}$ von disjunkten Teilmengen $S_i \subseteq E$
- jede Menge S_i ist durch einen eindeutigen Repräsentanten $x \in S_i$ gekennzeichnet

Die Union-Find-Datenstruktur unterstützt folgende Operationen:

- **erzeuge(x)**
  - erzeugt eine neue Menge $\{x\}$ mit x als Repräsentant von $\{x\}$
- **vereinige(x,y)**
  - vereinigt die Mengen S_x und S_y, die x und y enthalten, zu $S_x \cup S_y$
  - der Repräsentant von $S_x \cup S_y$ ist ein belibiges Element aus $S_x \cup S_y$
- **finde(x)**
  - liefert den Repräsentanten der (eindeutigen) Menge, die x enthält

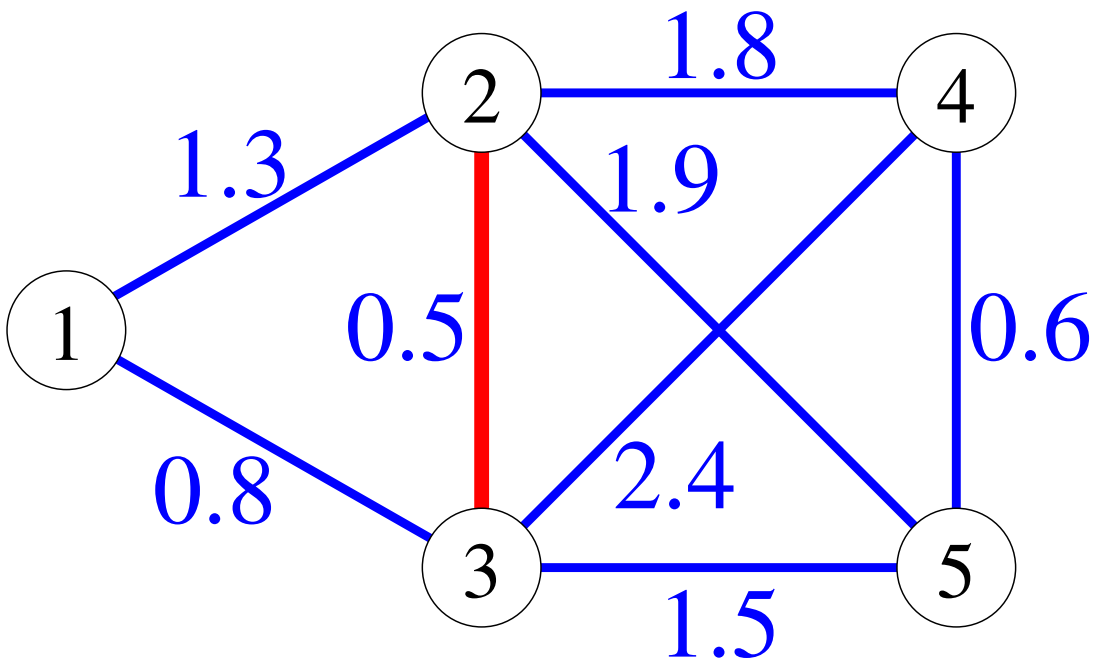
Beispiel-Lauf: Kruskal-Algorithmus



$L = [\{2,3\},\{4,5\},\{1,3\},\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\}$

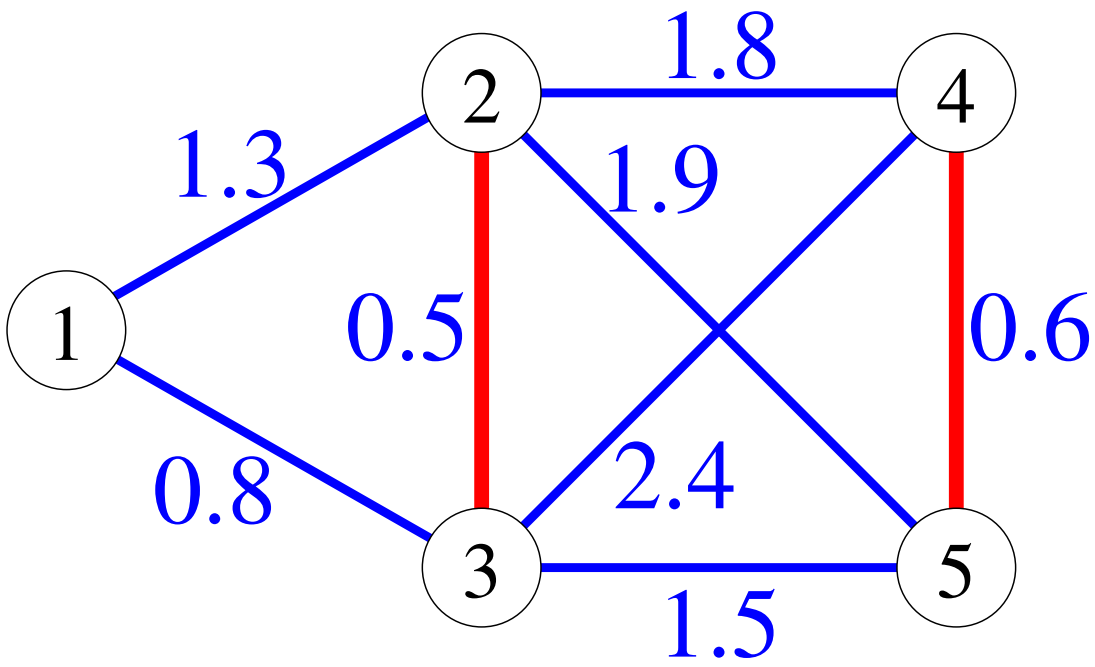
Beispiel-Lauf: Kruskal-Algorithmus



$L = [\{4,5\}, \{1,3\}, \{1,2\}, \{3,5\}, \{2,4\}, \{2,5\}, \{3,4\}]$

$F = \{\{2,3\}\}$

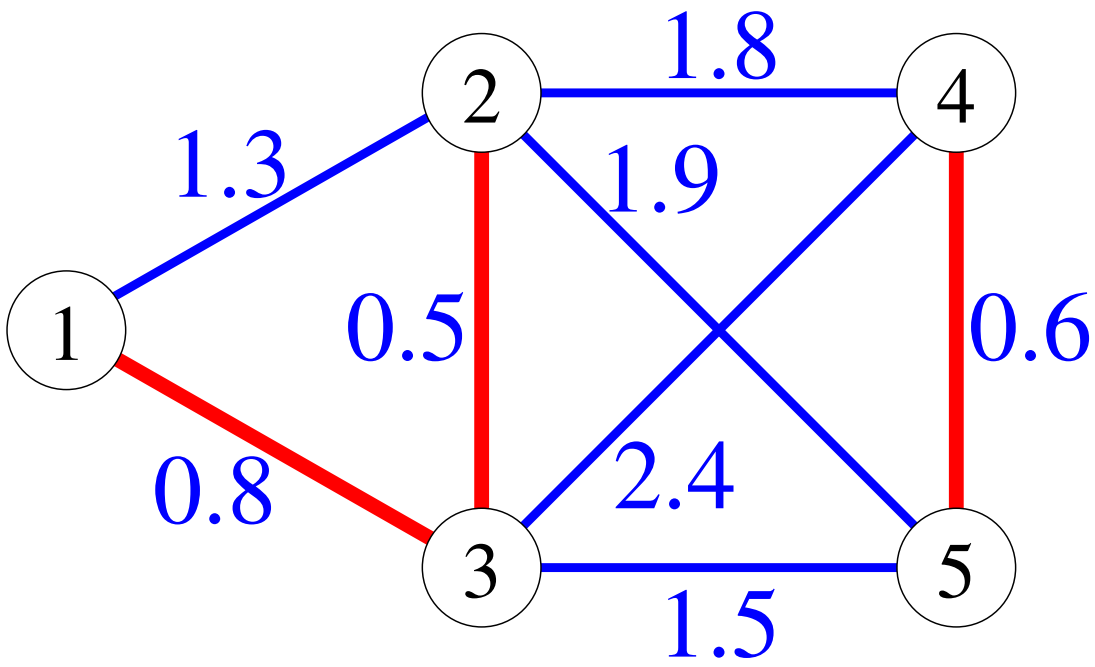
Beispiel-Lauf: Kruskal-Algorithmus



$L = [\{1,3\},\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

$F = \{\{2,3\},\{4,5\}\}$

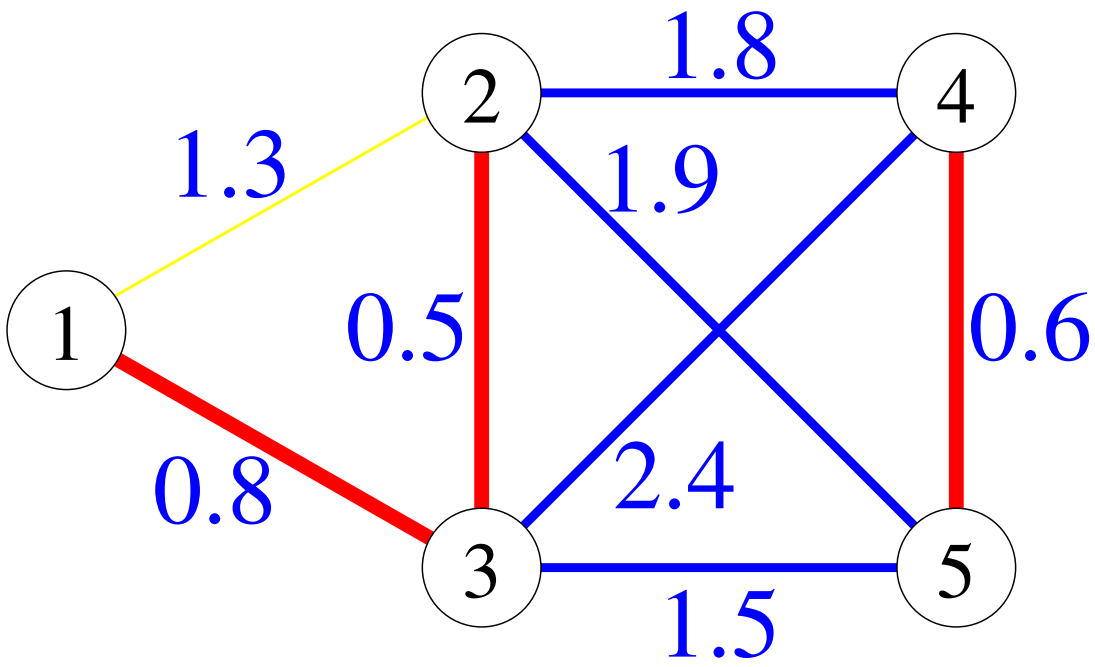
Beispiel-Lauf: Kruskal-Algorithmus



$L = [\{1,2\},\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

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

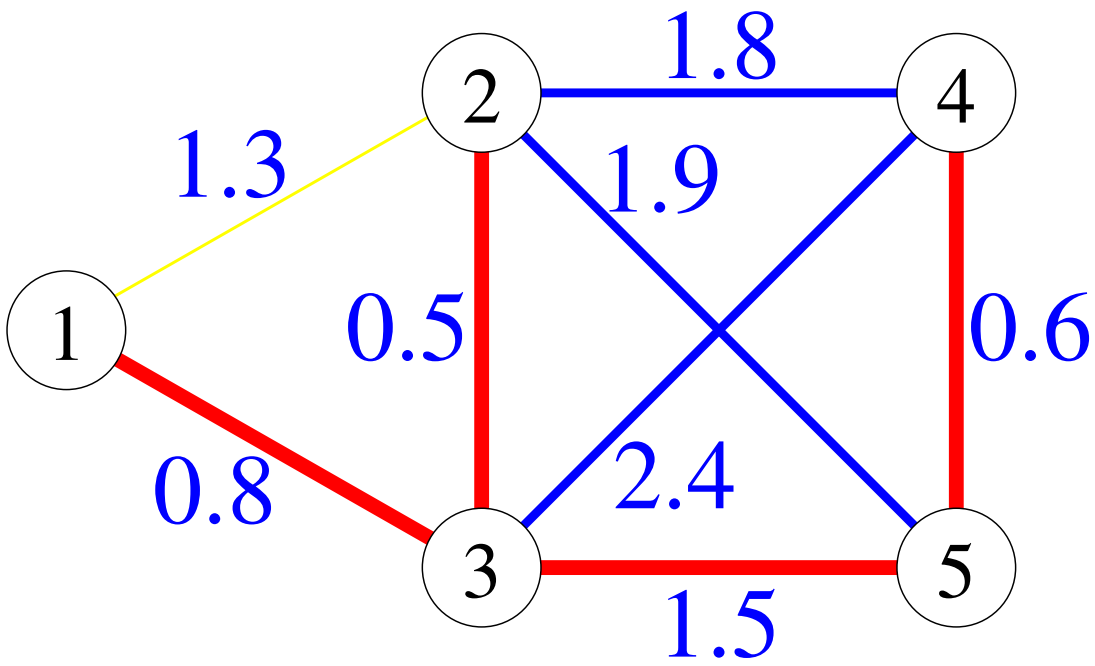
Beispiel-Lauf: Kruskal-Algorithmus



$L = [\{3,5\},\{2,4\},\{2,5\},\{3,4\}]$

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

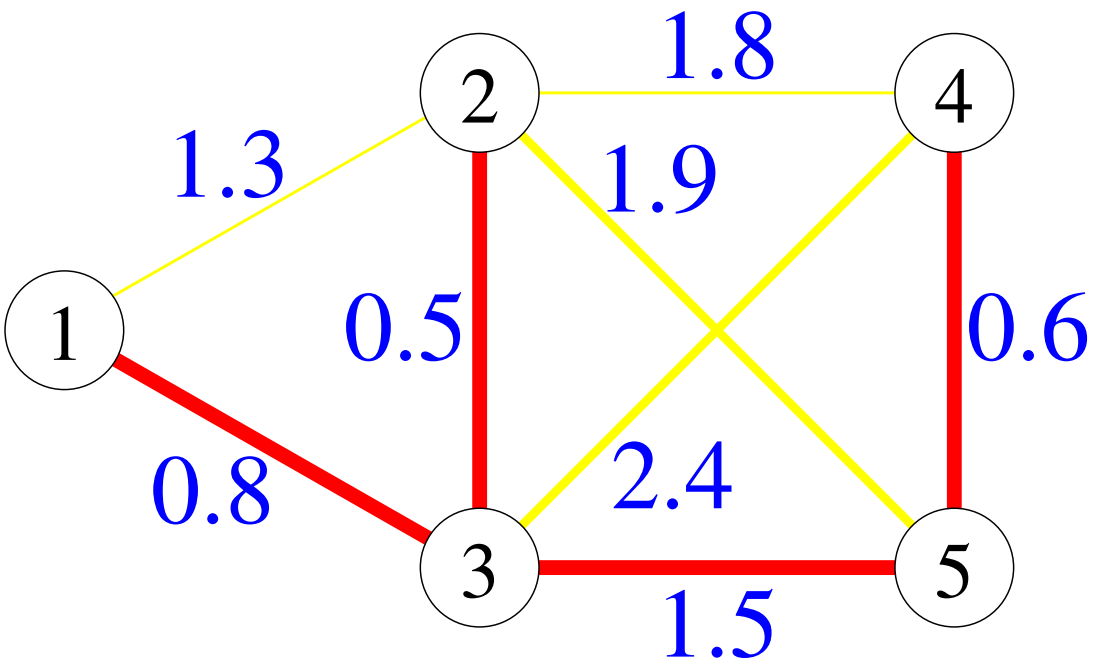
Beispiel-Lauf: Kruskal-Algorithmus



$L = [\{2,4\},\{2,5\},\{3,4\}]$

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

Beispiel-Lauf: Kruskal-Algorithmus



$L = []$

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

Anmerkungen zum Kruskal-Algorithmus

Berechnung eines *minimalen* Spannbaums (daraus folgt: gegeben ein zusammenhängender Graph)

- **Korrektheit:** Findet der Kruskal-Algorithmus garantiert einen minimalem Spannbaum?
  $\rightarrow$ Ja, sehen wir später bei Greedy-Algorithmen.

- **Laufzeit-Komplexität:** $O(|E| \log |E|)$ $[= O(|E| \log |V|)]$.
  Betrachten wir dazu den folgenden Algorithmus

Kruskal Pseudocode

Initialisiere eine lineare Liste T und eine UnionFind-Struktur auf V mit $\{v_1, ..., v_n\}$;
konstruiere minHeap der Kanten E mit Bewertung c;
setze k:= 0;
while *k* $\neq$ *n-1* **do**

```
    {u, v} := Wurzel des minHeaps;
    entferne {u, v} aus Heap;
    stelle Heapstruktur wieder her;
    A = finde (u);
    B = finde (v);
    if A≠B then
        T = T ∪ {u, v};
        k = k+1;
        vereinige (A, B)
    end
end
```

Komplexitätsbetrachtung

- sei $|E| = m, |V| = n, m \geqslant n - 1$
- sei t die Anzahl der Durchläufe in der *While-Schleife*, $t \leqslant m$
- die Laufzeit eines Schleifendurchlaufs wird dominiert von den Heap-Operationen $O(\log |E|)$ $[= O(\log m)]$
- **Laufzeit-Komplexität:** $O(t \log m)$ $[= O(|E| \log |E|)]$

Gerichteter Graph

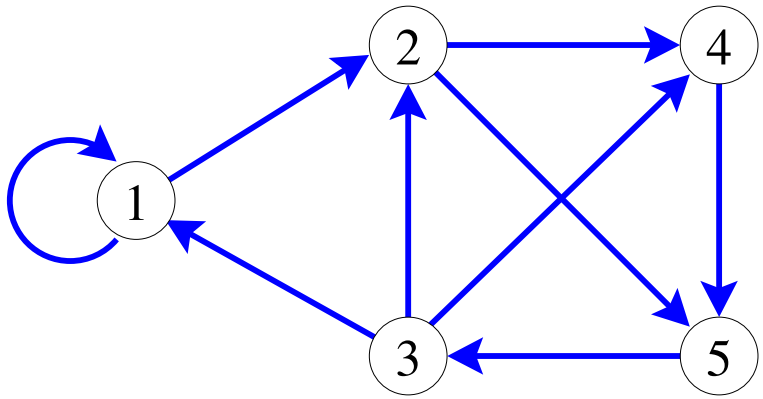
Ein Tupel (V, E) heißt *gerichteter Graph* (Digraph), wenn V eine endliche Menge und E eine Menge geordneter Paare von Elementen in V ist.
V heißt *Knotenmenge*, die Elemente von V heißen *Knoten*.
E heißt *Kantenmenge*, die Elemente von E heißen *Kanten*.
Eine Kante (v, v) heißt *Schleife*.

Beispiel:
$V = \{1, 2, 3, 4, 5\}$,
$E = \{(1,1), (1,2), (2,4), (2,5), (3,1), (3,2), (3,4), (4,5), (5,3)\}$

Gerichteter Graph

Ein Tupel (V, E) heißt *gerichteter Graph* (Digraph), wenn V eine endliche Menge und E eine Menge geordneter Paare von Elementen in V ist.
V heißt *Knotenmenge*, die Elemente von V heißen *Knoten*.
E heißt *Kantenmenge*, die Elemente von E heißen *Kanten*.
Eine Kante (v, v) heißt *Schleife*.

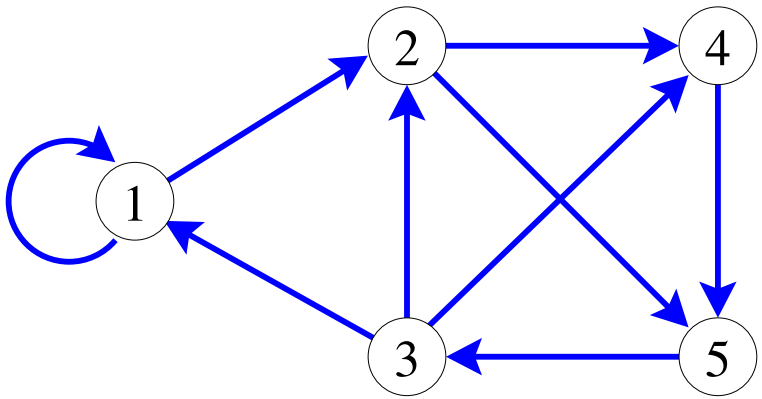
Beispiel:
$V = \{1, 2, 3, 4, 5\}$,
$E = \{(1,1), (1,2), (2,4), (2,5), (3,1), (3,2), (3,4), (4,5), (5,3)\}$



Vorgänger, Nachfolger, Grad

Sei $G = (V, E)$ ein gerichteter Graph und $v \in V$.

- $u \in V$ heißt *Vorgänger* von v, wenn $(u, v) \in E$.
- Mit $\text{pred}(v) := \{u \in V | (u, v) \in E\}$ bezeichnen wir die Menge der Vorgänger von v.
- Der *Eingangsgrad* von v ist $\text{eg}(v) = |\text{pred}(v)|$
- $w \in V$ heißt *Nachfolger* von v, wenn $(v, w) \in E$.
- Mit $\text{succ}(v) := \{w \in V | (v, w) \in E\}$ bezeichnen wir die Menge der Nachfolger von v.
- Der *Ausgangsgrad* von v ist $\text{ag}(v) = |\text{succ}(v)|$



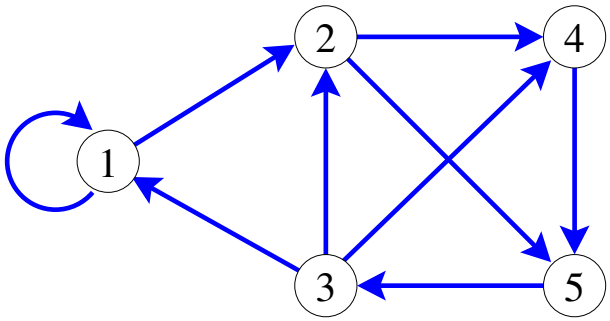
$\text{eg}(3) = 1$, $\text{pred}(3) = \{5\}$
$\text{ag}(3) = 3$, $\text{succ}(3) = \{1, 2, 4\}$

Speicherung von Graphen: Adjazenzmatrix

Ein Graph $G = (V, E)$ mit $|V| = n$ wird in einer Boole'schen $n \times n$-Matrix $A = (a_{ij})$ gespeichert, wobei

$$a_{ij} = \begin{cases} 1 & \text{falls } (i,j) \in E \\ 0 & \text{sonst} \end{cases}$$

Beispiel:



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

Speicherplatzbedarf $O(n^2)$

- 1 Bit pro Position (statt Knoten/Kantennummern)
- unabhängig von Kantenmenge
- für ungerichtete Graphen ergibt sich symmetrische Belegung (Halbierung des Speicherbedarfs möglich)

Speicherung von Graphen in Listen

Knoten- und Kantenlisten

- Speicherung von Graphen als Liste von Zahlen (z.B. in Array oder verketteter Liste)
- Knoten werden von 1 bis n durchnummeriert; Kanten als Paare von Knoten

Kantenliste

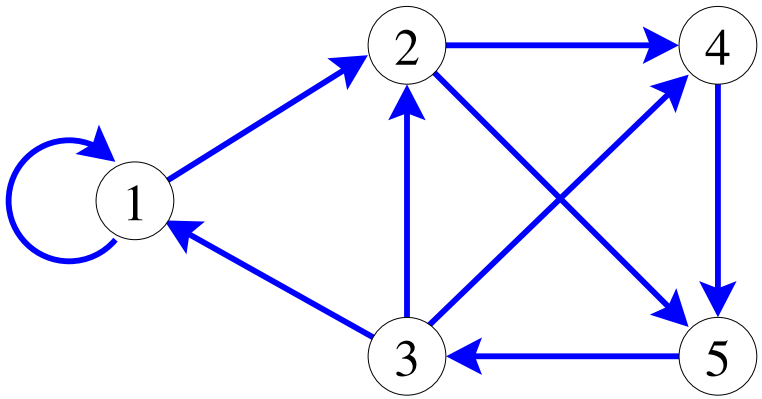
- Liste: Knotenzahl, Kantenzahl, Liste von Kanten (je als 2 Zahlen)
- Speicherbedarf: $2 + 2m$ (m = Anzahl Kanten, bei Digraphs $2 + m$)

Knotenliste

- Liste: Knotenzahl, Kantenzahl, Liste von Knoteninformationen
- Knoteninformation: Ausgangsgrad und Nachfolger $\mathrm{ag}(v), s_1, s_2, \ldots, s_{\mathrm{ag}(v)}$
- Speicherbedarf: $2 + n + m$

Adjazenzlisten

- verkettete Liste der n Knoten (oder Array-Realisierung)
- pro Knoten: verkettete Liste der Nachfolger (repräsentiert die von dem Knoten ausgehenden Kanten)
- Speicherbedarf: $n + m$ Listenelemente



$$
\begin{array}{l}
1 \to 1 \to 2 \\
\downarrow \\
2 \to 4 \to 5 \\
\downarrow \\
3 \to 1 \to 2 \to 4 \\
\downarrow \\
4 \to 5 \\
\downarrow \\
5 \to 3
\end{array}
$$

Speicherung von Graphen: Vergleich

Komplexitätsvergleich

Operation	Adj.-matrix	Kantenliste	Knotenliste	Adjazenzliste
Einfügen Kante	$O(1)$	$O(1)$	$O(n+m)$	$O(n)$
Löschen Kante	$O(1)$	$O(m)$	$O(n+m)$	$O(n)$
Einfügen Knoten	$O(n^2)$	$O(1)$	$O(1)$	$O(1)$
Löschen Knoten	$O(n^2)$	$O(m)$	$O(n+m)$	$O(n+m)$

- Löschen eines Knotens löscht auch zugehörige Kanten
- änderungsaufwand abhängig von Realisierung der Adjazenzmatrix und Adjazenzliste

Welche Repräsentation geeigneter ist, hängt vom Problem ab:

- Frage: Gibt es Kante von a nach b? $\rightarrow$ Matrix
- Durchsuchen von Knoten in durch Nachbarschaft gegebener Reihenfolge $\rightarrow$ Listen

Kantenfolgen, Pfade, Zyklen

Sei $G = (V, E)$ gerichteter Graph, $\ell \in \mathbb{N} \cup \{0\}$ und $k = (v_0, v_1, \ldots, v_\ell) \in V^{\ell+1}$.

- k heißt *Kantenfolge* (oder *Weg*) der Länge ℓ von v_0 nach v_ℓ, wenn für alle $i \in \{1, \ldots, \ell\}$ gilt: $(v_{i-1}, v_i) \in E$
- $v_1, \ldots, v_{\ell-1}$ sind die *inneren* Knoten von k. Ist $v_0 = v_\ell$, so ist die Kantenfolge *geschlossen*.
- k heißt *Kantenzug*, wenn k Kantenfolge ist und für alle $i, j \in \{0, \ldots, \ell - 1\}$ mit $i \neq j$ gilt: $(v_i, v_{i+1}) \neq (v_j, v_{j+1})$.
- k heißt *Pfad* der Länge ℓ, wenn k eine Kantenfolge ist und für alle $i, j \in \{0, \ldots, \ell\}$ mit $i \neq j$ gilt: $v_i \neq v_j$.
- k heißt *Zyklus* (oder *Kreis*), wenn $(v_1, \ldots, v_\ell)$ ein Pfad der Länge $\ell - 1$ ist, $v_0 = v_\ell$, und $(v_0, v_1) \in E$.
- k heißt *Hamiltonscher Zyklus*, wenn k Zyklus ist und $\ell = |V|$ (also eine geschlossene Kantenfolge, die jeden Knoten genau einmal enthält; TSP sucht nach dem kürzesten Hamilton Zyklus) .