

Graphentheorie

für Wiederholer Bachelor Informatik und Wirtschaftsinformatik

Prof. Dr. Peter Becker

Fachbereich Informatik
Hochschule Bonn-Rhein-Sieg

Wintersemester 2018/19



**Hochschule
Bonn-Rhein-Sieg**
University of Applied Sciences

Allgemeines zur Vorlesung

- Es gibt eine **Homepage** zur Vorlesung:
<http://www2.inf.h-brs.de/~pbecke2m/graphentheorie/>
- Die Vorlesung wird **folienbasiert** gehalten,
- aber die Folien enthalten **nur die wichtigsten Aspekte** (Definitionen, Sätze, knappe Beispiele, wichtige Bemerkung).
- Alles was sonst eine Vorlesung ausmacht (Erläuterungen, ausführliche Beispiele, Beweise von Sätzen, Anwendungen, Querverweise auf andere Gebiete der Informatik, etc.) gibt es nur in der Vorlesung selbst.
- Die Folien zur Vorlesung (Skript) stehen auf der Homepage **vor der Vorlesung** zur Verfügung.
- Format: PDF

Übungen

Verfahrensweise für Übungen in Graphentheorie im WS 2018/19: Es gibt

- Hörsaalübungen
- Übungen mit ACAT

Es finden **keine Übungen in Gruppen** statt!

Hörsaalübungen

- vierzehntäglich (uKW), jeweils Donnerstags im HS 3 von 13:30 bis 15:00 Uhr
- Beginn der Hörsaalübungen: KW 43, 25. Oktober 2018.
- Übungsblatt 1 morgen!
- Übungsblätter werden nicht bewertet, es gibt keine Zulassungsvoraussetzung zur Prüfung.
- Üben, üben, üben!

ACAT-Übungen

- über das Semester verstreut
- elektronische Abgabe mit automatischer Auswertung
- ACAT-System
- freiwillig, zum besseren Verständnis und Klausurvorbereitung
- teilweise Programmieraufgaben

Prüfung

- Gemeinsame Prüfung [Graphentheorie/Stochastik](#)
- [6 ECTS-Punkte](#) für Graphentheorie/Stochastik
- Prüfungstermin: [siehe Prüfungsplan](#)
- Prüfungsform: schriftlich ([Klausur](#))
- Hilfsmittel für die Gesamtprüfung: [DIN A4 Blatt](#), zweiseitig beschrieben
- [keine Zwischenklausur](#)

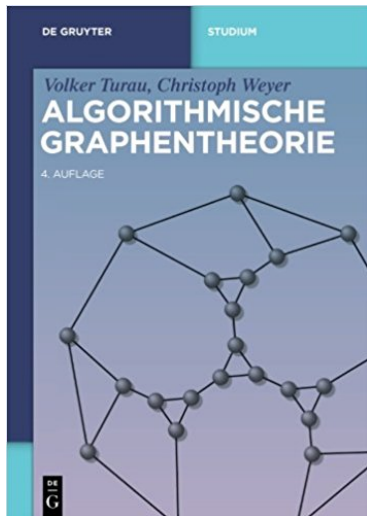
Zulassungsvoraussetzung zur Klausur

Keine!

Inhalt

- 1 Einführung
- 2 Repräsentation von Graphen in Computern
- 3 Durchsuchen von Graphen
- 4 Kreis- und Wegeprobleme
- 5 Bäume und Minimalgerüste
- 6 Planare Graphen und Färbungen
- 7 Flüsse und Zuordnungen

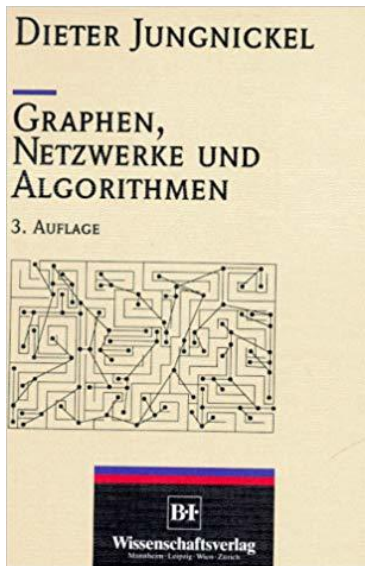
Literatur



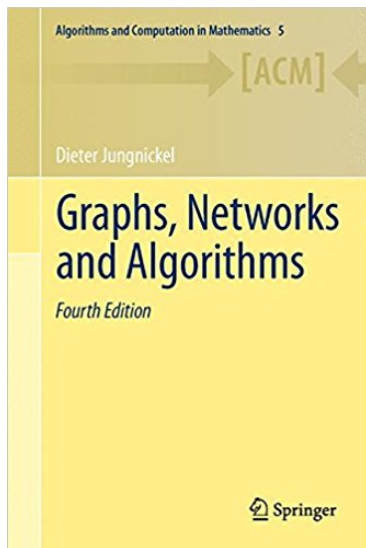
- passt am besten zur Vorlesung
- sehr anwendungsorientiert
- Schwerpunkt liegt auf Algorithmen



- anspruchsvoller als das Buch von Turau,
- aber auch umfassender
- exaktere Darstellungen, stärker mathematisch orientiert



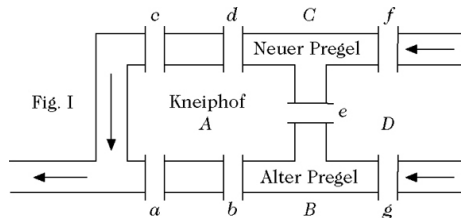
- Sehr zu empfehlen, auch für noch kommende Vorlesungen
- Ausgabe von 1994
- Im Buchhandel nicht mehr erhältlich
- Tipp: gebraucht kaufen



- aktuelle englische Ausgabe des Buchs von Jungnickel
- umfassend und tiefgehend
- in der Bibliothek ausleihbar

Kapitel 1

Einführung



Inhalt

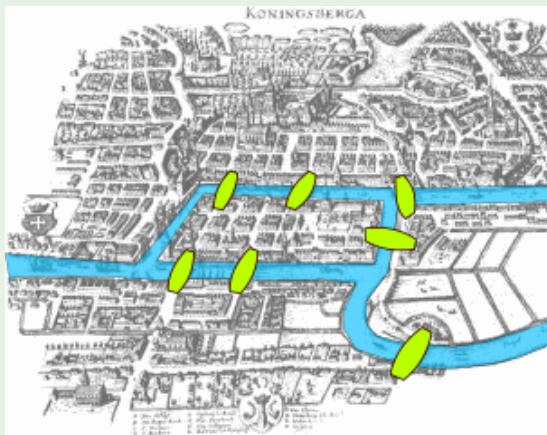
1 Einführung

- Grundbegriffe und Bezeichnungen
- Bäume
- Gerichtete Graphen

Das Königsberger Brückenproblem

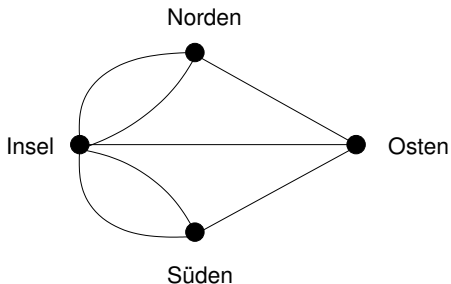
Beispiel 1.1 (Euler, 1736)

Gibt es einen Rundweg durch Königsberg, der jede der sieben Brücken genau einmal überquert?



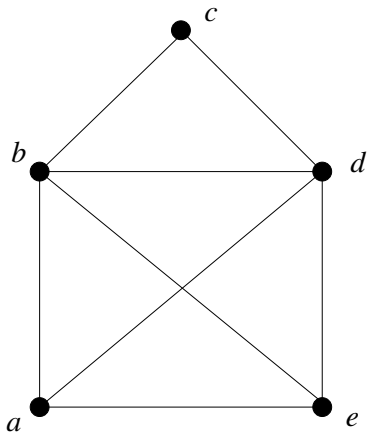
Das Königsberger Brückenproblem (2)

Die **Abstraktion des Problems**:



Gibt es einen Rundweg, der jede Linie (Kante) genau einmal enthält?

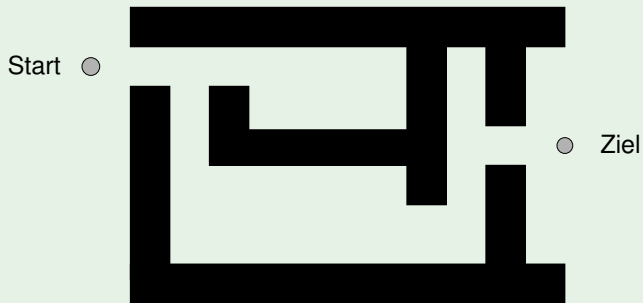
Das Haus vom Nikolaus



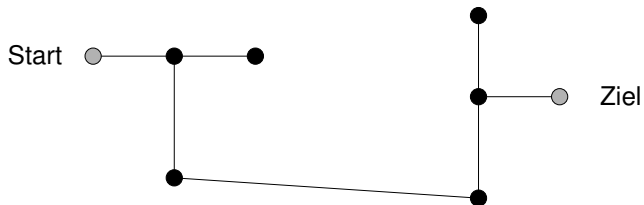
Labyrinth

Beispiel 1.2

Finde einen Weg vom *Start* zum *Ziel* durch das Labyrinth!

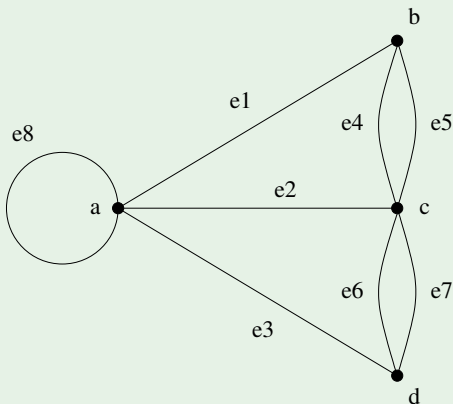


Repräsentation als Graph:



Beispielgraph

Beispiel 1.3



- Ein Graph besteht aus Knoten und Kanten.
 - a, b, c, d sind Knoten.
 - Diese Knoten werden durch die Kanten e_1 bis e_8 miteinander verbunden.
- ☞ Ein Graph symbolisiert die max. zweistellige Beziehungen zwischen Elementen einer Menge.

Graph

Definition 1.4

Ein **Graph (graph)** $G = (V, E, \gamma)$ ist ein Tripel bestehend aus:

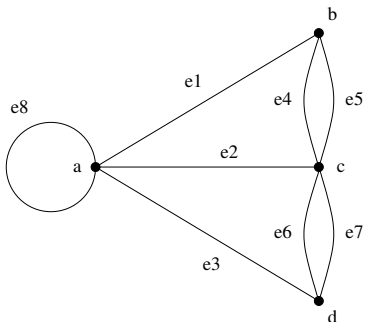
- V , einer nicht leeren Menge von **Knoten (vertices)**,
- E , einer Menge von **Kanten (edges)** und
- γ , einer **Inzidenzabbildung (incidence relation)**, mit $\gamma : E \rightarrow \{X \mid X \subseteq V, 1 \leq |X| \leq 2\}$.

Zwei Knoten $a, b \in V$ heißen **adjazent (adjacent)** gdw.

$$\exists e \in E : \gamma(e) = \{a, b\}.$$

Ein Knoten $a \in V$ und eine Kante $e \in E$ heißen **inzident (incident)** gdw.
 $a \in \gamma(e)$.

Beispielgraph (2)



$$V = \{a, b, c, d\}$$

$$E = \{e_1, e_2, \dots, e_8\}$$

$$\gamma = \{(e_1, \{a, b\}), (e_2, \{a, c\}), (e_3, \{a, d\}), (e_4, \{b, c\}), (e_5, \{b, c\}), (e_6, \{c, d\}), (e_7, \{c, d\}), (e_8, \{a\})\}$$

Endliche Graphen

Definition 1.5

Ein Graph $G = (V, E, \gamma)$ heißt **endlich (finite)** gdw. die Knotenmenge V und die Kantenmenge E endlich sind.

Wir treffen folgende Vereinbarung:

- Im weiteren betrachten wir **nur endliche Graphen**.
- Der Zusatz “endlich” lassen wir dabei weg.

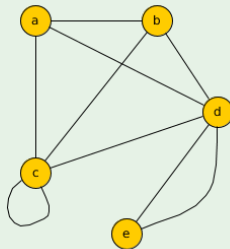
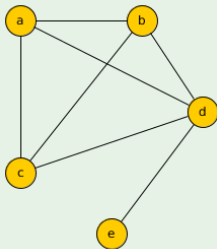
Schlichte Graphen

Definition 1.6

- Eine Kante $e \in E$ heißt **Schlinge (loop)** gdw. e nur zu einem Knoten inzident ist.
- Zwei Kanten $e_1, e_2 \in E$ heißen **parallel (parallel)** gdw. sie zu den selben Knoten inzident sind.
- Ein Graph heißt **schlicht (simple)** gdw. G keine Schlingen und keine parallelen Kanten enthält.

Schlichte Graphen (2)

Beispiel 1.7



Der linke Graph ist schlicht, der rechte nicht.

Schlichte Graphen (3)

Ein schlichter Graph $G = (V, E)$ wird beschrieben durch

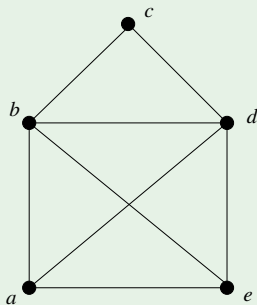
- eine Knotenmenge V und
- eine Kantenmenge E , wobei E eine Menge zweielementiger Teilmengen von V ist, also

$$E \subseteq \{\{v, w\} \mid v, w \in V, v \neq w\}.$$

☞ Wir betrachten im folgenden fast ausschließlich schlichte Graphen.

Schlichte Graphen (4)

Beispiel 1.8



$$V = \{a, b, c, d, e\}$$

$$E = \{\{a, b\}, \{a, d\}, \{a, e\}, \{b, c\}, \\ \{b, d\}, \{b, e\}, \{c, d\}, \{d, e\}\}$$

Diagramme

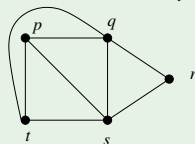
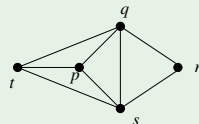
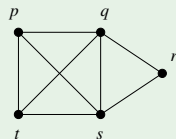
- Graphen können durch **Diagramme** veranschaulicht werden.
- Der selbe Graph kann viele verschiedene Diagramme haben.

Beispiel 1.9

$G = (V, E)$ mit

$$V = \{p, q, r, s, t\}$$

$$E = \{\{p, q\}, \{p, s\}, \{p, t\}, \{q, r\}, \\ \{q, s\}, \{q, t\}, \{r, s\}, \{s, t\}\}$$



Grad

Definition 1.10

Der **Grad (degree)** $\deg(v)$ eines Knotens $v \in V$ ist die Zahl der zu v inzidenten Kanten. Hierbei zählen Schlingen doppelt.

Der **Maximalgrad** $\Delta(G)$ eines Graphen G ist definiert durch

$$\Delta(G) = \max\{\deg(v) \mid v \in V\}$$

Der **Minimalgrad** $\delta(G)$ eines Graphen G ist definiert durch

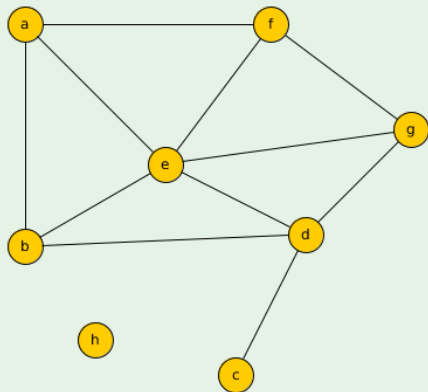
$$\delta(G) = \min\{\deg(v) \mid v \in V\}$$

Ein Knoten $v \in V$ mit $\deg(v) = 0$ heißt **isoliert**.

Ein Knoten $v \in V$ mit $\deg(v) = 1$ heißt **Blatt**.

Grad (2)

Beispiel 1.11



- $\delta(G) = 0$ (für Knoten h)
- $\Delta(G) = 5$ (für Knoten e)
- Knoten c ist ein Blatt.
- Knoten h ist ein isolierter Knoten.

Handschlaglemma

- Jede Kante liefert **genau zweimal** einen Beitrag zu der Summe der Grade über alle Knoten.
- Dies gilt **auch für Schlingen**.

Lemma 1.12

Für jeden Graphen $G = (V, E, \gamma)$ gilt

$$\sum_{v \in V} \deg(v) = 2|E|$$

Handschlaglemma (2)

Folgerung 1.13

Jeder Graph hat eine gerade Anzahl an Knoten mit ungeradem Grad.

Beweis.

$$\begin{array}{l}
 V_g := \{v \in V \mid \deg(v) \text{ ist gerade}\} \\
 V_u := \{v \in V \mid \deg(v) \text{ ist ungerade}\} \\
 2|E| = \sum_{v \in V} \deg(v) \\
 = \sum_{v \in V_g} \deg(v) + \sum_{v \in V_u} \deg(v)
 \end{array}
 \left| \begin{array}{l}
 \text{Definition} \\
 \text{Definition} \\
 \text{Handschlaglemma} \\
 \text{weil } V = V_g + V_u
 \end{array} \right.$$

$$\begin{array}{l}
 \Rightarrow \sum_{v \in V_u} \deg(v) \text{ ist gerade} \\
 \Rightarrow |V_u| \text{ ist gerade}
 \end{array}
 \left| \begin{array}{l}
 \text{weil } 2|E| \text{ und } \sum_{v \in V_g} \deg(v) \text{ gerade} \\
 \text{weil alle Summanden ungerade}
 \end{array} \right.$$

Grad (3)

Satz 1.14

Jeder Graph $G = (V, E)$ mit mindestens zwei Knoten enthält zwei Knoten, die den gleichen Grad haben.

Der Beweis dieses Satzes erfolgt mit Hilfe eines wichtigen kombinatorischen Prinzips, dem **Schubfachprinzip**.

Schubfachprinzip

Satz 1.15

Es seien n Elemente auf m (paarweise disjunkte) Mengen verteilt und es gelte $n > m$.

Dann gibt es mindestens eine Menge, die mindestens zwei Elemente enthält.

Beweis.

Wenn jede der m Mengen höchstens ein Element enthalten würde, dann gäbe es insgesamt höchstens m Elemente. Widerspruch zu $n > m$.

Andere Bezeichnungen für das Schubfachprinzip: **Taubenschlagprinzip**,
engl.: **pigeonhole principle**

Schubfachprinzip (2)

Beispiel 1.16

Herr Müller hat in seiner Sockenbox weiße, schwarze und grüne Socken.

Wenn er vier Socken aus der Box nimmt, hat er mindestens zwei Socken mit der gleichen Farbe.

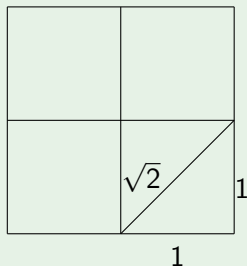
$n = 4$ Elemente verteilt auf $m = 3$ Mengen.

Schubfachprinzip (3)

Beispiel 1.17

Unter je fünf Punkten, die in einem Quadrat der Seitenlänge 2 liegen, gibt es stets zwei, die einen Abstand $\leq \sqrt{2}$ haben.

- Wir unterteilen das Quadrat durch Halbieren der Seitenlänge in vier Unterquadrate mit Seitenlänge 1.
- $n = 5$ Punkte verteilen sich auf $m = 4$ Unterquadrate.
- Dann muss mindestens ein Unterquadrat zwei Punkte enthalten.



Beweis von Satz 1.14

Beweis.

$$\begin{array}{l} n := |V| \\ V_i := \{v \in V \mid \deg(v) = i\} \text{ für } i = 0, \dots, n-1 \end{array} \quad \left| \begin{array}{l} \text{Definition} \\ \text{Definition} \end{array} \right.$$

Damit haben wir aber genauso viele Knoten wie Mengen, das Schubfachprinzip ist noch nicht anwendbar. Deshalb Fallunterscheidung:

G hat keinen isolierten Knoten

$$\Rightarrow V_0 = \emptyset$$

$\Rightarrow n$ Knoten verteilen sich auf die $m = n - 1$ Mengen V_1, \dots, V_{n-1}

$$\Rightarrow \exists i : |V_i| \geq 2$$

Fortsetzung Beweis.

G hat einen isolierten Knoten

⇒ Es existiert kein Knoten, der zu allen anderen Knoten adjazent ist

⇒ $V_{n-1} = \emptyset$

⇒ n Knoten verteilen sich auf die $m = n - 1$ Mengen V_0, \dots, V_{n-2}

⇒ $\exists i : |V_i| \geq 2$

Vollständiger Graph

Definition 1.18

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

Gilt $\{v, w\} \in E$ für alle $v, w \in V, v \neq w$, dann heißt G **vollständig (complete)**.

Mit K_n wird der **vollständige Graph mit n Knoten** bezeichnet.

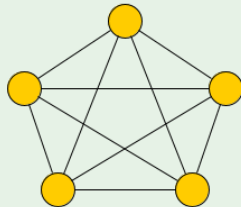
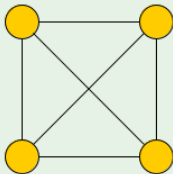
Bemerkungen:

- In einem vollständigen Graphen sind **je zwei Knoten adjazent**.
- Der K_n hat $\binom{n}{2} = \frac{n(n-1)}{2}$ Kanten.

Vollständiger Graph (2)

Beispiel 1.19

Die vollständigen Graphen K_4 und K_5 .



Komplementgraph

Definition 1.20

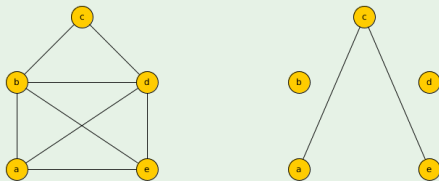
Es sei $G = (V, E)$ ein Graph. Dann heißt der Graph $\bar{G} = (V, \bar{E})$ mit

$$\bar{E} = \{\{v, w\} \mid v, w \in V, v \neq w\} \setminus E$$

Komplementgraph (complementary graph) von G .

Beispiel 1.21

Das “Haus vom Nikolaus” und sein Komplementgraph.



Untergraph

Definition 1.22

Sei $G = (V, E)$ ein Graph. Ein Graph $H = (W, F)$ mit $W \subseteq V$ und $F \subseteq E$ heißt **Untergraph (subgraph)** von G .

Gilt $W = V$, dann heißt H **aufspannender Untergraph (spanning subgraph)** von G .

Gilt

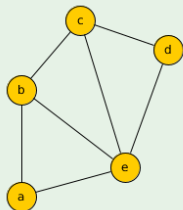
$$F = \{\{v, w\} \mid \{v, w\} \in E, v, w \in W\},$$

dann heißt H **induzierter Untergraph (induced subgraph)** von G .

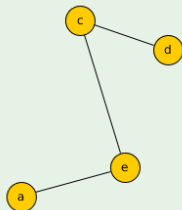
Für solch einen induzierten Untergraphen schreiben wir auch $G(W)$.

Untergraph (2)

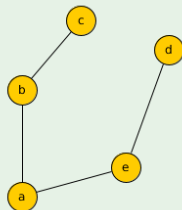
Beispiel 1.23



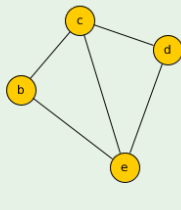
G



Untergraph
nicht aufspannend
nicht induziert



aufspannender Untergraph



induzierter Untergraph

Clique

Definition 1.24

Es sei $G = (V, E)$ ein Graph.

Eine Knotenmenge $U \subseteq V$ (bzw. der von U induzierte Untergraph $G(U)$) heißt **Clique (clique)** gdw. $G(U)$ ein vollständiger Graph ist.

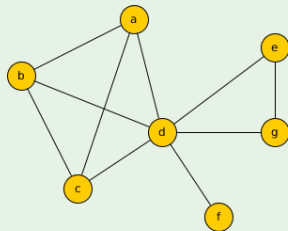
Die **maximale Größe einer Clique** in G wird mit $\omega(G)$ bezeichnet, d.h.

$$\omega(G) := \max\{|U| \mid U \text{ ist Clique in } G\}$$

Clique (2)

Beispiel 1.25

- $\{a, b, c, d\}$ bildet eine Clique der Größe 4.
- $\{d, e, g\}$ bildet eine Clique der Größe 3.
- $\{d, f\}$ bildet eine Clique der Größe 2.
- $\omega(G) = 4$



Wege

Definition 1.26

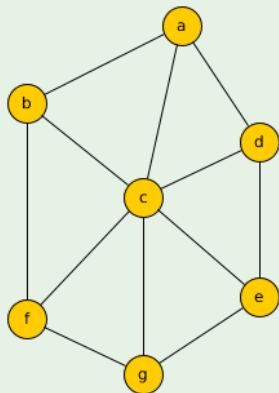
Es sei $G = (V, E)$ ein Graph.

- Eine Folge (v_0, v_1, \dots, v_n) von Knoten mit $e_i := \{v_{i-1}, v_i\} \in E$ für $i = 1, 2, \dots, n$ heißt **Kantenzug (walk)**.
- Ein Kantenzug, bei dem die Kanten e_i alle verschieden sind, heißt **Weg (trail)**. Die **Länge** des Weges ist n .
- Ein Weg heißt **einfacher Weg (path)** gdw. die Knoten v_j paarweise verschieden sind.

Wege (2)

Beispiel 1.27

- (a, b, c, a, b, f) ist ein Kantenzug, aber kein Weg.
- (c, b, f, c, d) ist ein Weg, aber kein einfacher Weg.
- (a, b, f, c, d) ist ein einfacher Weg.



Kreise

Definition 1.28

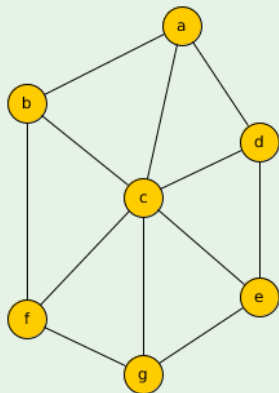
Die folgenden Bezeichnungen beziehen sich auf Definition 1.26.

- Gilt in einem Kantenzug $v_0 = v_n$, so sprechen wir von einem **geschlossenen Kantenzug (closed walk)**.
- Ein Weg für den $v_0 = v_n$ gilt heißt **Kreis (closed trail)**.
- Ein Kreis, bei dem die Knoten v_j mit Ausnahme von $v_0 = v_n$ paarweise verschieden sind, heißt **einfacher Kreis (cycle)**.

Kreise (2)

Beispiel 1.29

- (a, b, c, a, d, c, a) ist ein geschlossener Kantenzug, aber kein Kreis.
- (b, c, e, d, c, f, b) ist ein Kreis, aber kein einfacher Kreis.
- (a, b, f, c, a) ist ein einfacher Kreis.



Bemerkungen zu Wegen und Kreisen

- Ein Knoten alleine stellt einen Kreis der Länge 0 dar.
 - Im folgenden ist mit **“Kreis”** immer ein nichttrivialer Kreis gemeint, d.h. ein Kreis mit Länge > 0 .
 - Nur in schlichten Graphen ist durch die Knotenfolge der Weg bzw. Kreis eindeutig bestimmt.
 - In schlichten Graphen existieren keine Kreise der Länge 1 und 2.
- ☞ Die Begriffe werden in der Literatur nicht einheitlich verwendet.

Hilfssätze für Wege und Kreise

Lemma 1.30

Es sei $G = (V, E)$ ein Graph und es seien $a, b \in V$, $a \neq b$ zwei verschiedene Knoten von G . Dann gilt:

Wenn ein Kantenzug von a nach b existiert, dann existiert auch ein einfacher Weg von a nach b .

Lemma 1.31

Wenn ein Graph G einen geschlossenen Kantenzug K enthält, in dem eine Kante von K nicht mehrfach vorkommt, dann enthält G auch einen einfachen Kreis.

Zusammenhang

Definition 1.32

Es sei $G = (V, E)$ ein Graph.

Zwei Knoten $v, w \in V$ heißen **verbindbar** gdw. ein Weg von v nach w existiert.

G heißt **zusammenhängend (connected)** gdw. je zwei Knoten von G verbindbar sind.

Eine **Zusammenhangskomponente (connected component)** von G ist

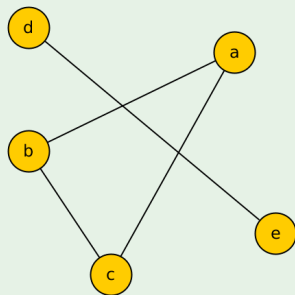
- ein durch eine Knotenmenge $U \subseteq V$ induzierter Untergraph $G(U)$, der zusammenhängend und
- bezüglich der Knotenmenge maximal ist, d.h. $G(W)$ ist nicht zusammenhängend für alle $W \supsetneq U$.

Zusammenhang (2)

Beispiel 1.33

Ein nicht zusammenhängender Graph mit
Zusammenhangskomponenten induziert durch

- $\{a, b, c\}$ und
- $\{d, e\}$.



Zusammenhang (3)

Satz 1.34

Jeder zusammenhängende Graph mit n Knoten hat mindestens $n - 1$ Kanten.

Beweis.

Mittels vollständiger Induktion über die Anzahl der Knoten, also über n .

Induktionsanfang: $n = 1$: Ein Graph mit genau einem Knoten ist zusammenhängend und hat keine Kanten.

Induktionsschritt: $n \rightarrow n + 1$:

Induktionsvoraussetzung: Jeder zusammenhängende Graph mit $n' \leq n$ Knoten hat mindestens $n' - 1$ Kanten.

Induktionsbehauptung: Jeder zusammenhängende Graph mit $n + 1$ Knoten hat mindestens n Kanten.

Fortsetzung Beweis.

Es sei $G = (V, E)$ ein Graph mit $n + 1$ Knoten.

Wähle beliebigen Knoten $v \in V$.

$k := \deg(v)$.

$\Rightarrow k \geq 1$

Definition

weil G z.h.

Es sei G' der Graph der entsteht, wenn wir aus G den Knoten v und alle mit v inzidenten Kanten entfernen.

G' besteht aus höchstens $l \leq k$ ZHKs, ZHK_1, \dots, ZHK_l .

Jede ZHK_i enthält höchstens n Knoten.

Definition

wegen $\deg(v) = k$

weil G' insgesamt nur n Knoten hat

\Rightarrow Wir können für jede ZHK_i die Induktionsvoraussetzung anwenden.

Es sei n_i die Anzahl der Knoten in ZHK_i .

\Rightarrow Jede ZHK_i hat mindestens $n_i - 1$ Kanten.

Definition

Induktionsvoraussetzung

Fortsetzung Beweis.

Weiterhin gilt $n_1 + n_2 + \dots + n_l = n$.

Alle Knoten von G ausgenommen v sind in den ZHKs von G' .

$$|E| \geq (n_1 - 1) + \dots + (n_l - 1) + k$$

Kanten in ZHKs von G' plus die mit v inzidenten Kanten

$$= n_1 + \dots + n_l - l + k$$

$$\geq n - k + k$$

$$= n$$

s.o., und weil $l \leq k$

q.e.d.

Zusammenhang (4)

Lemma 1.35

Ein Graph $G = (V, E)$ ist genau dann zusammenhängend, wenn für jede disjunkte Zerlegung $V = V_1 + V_2$ mit $V_1, V_2 \neq \emptyset$ eine Kante $e = \{v, w\}$ existiert mit $v \in V_1$ und $w \in V_2$.

Beweis.

Übungsaufgabe.

Isomorphie

Definition 1.36

Zwei Graphen $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ heißen **isomorph** (**isomorphic**) gdw. es eine bijektive Abbildung $\varphi : V_1 \rightarrow V_2$ gibt, so dass folgendes gilt:

$$\forall v, w \in V_1 : \{v, w\} \in E_1 \iff \{\varphi(v), \varphi(w)\} \in E_2$$

Wir nennen φ dann einen **Isomorphismus** von G_1 auf G_2 und schreiben $G_1 \cong G_2$.

Isomorphie (2)

- Zwei Graphen sind genau dann isomorph, wenn der eine Graph aus dem anderen Graphen **durch Umbenennung der Knoten hervorgeht**.
 - Isomorphe Graphen haben die **gleichen graphentheoretischen Eigenschaften**.
- ☞ Für den Nachweis, dass zwei Graphen G_1 und G_2 nicht isomorph sind, genügt es, eine **graphentheoretische Eigenschaft zu finden, in der sich G_1 und G_2 unterscheiden**.

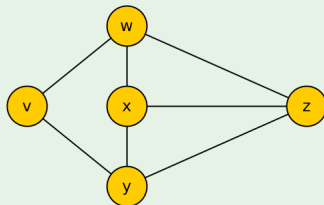
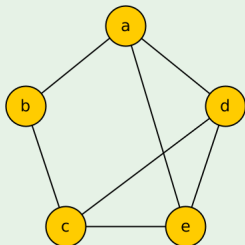
Isomorphie (3)

Beispiel 1.37

Die Abbildung

$$\varphi = \{a \rightarrow w, b \rightarrow v, c \rightarrow y, d \rightarrow x, e \rightarrow z\}$$

ist ein Isomorphismus für die folgenden Graphen.



Isomorphie (4)

Definition 1.38

Wenn in Definition 1.36 $G_1 = G_2$ gilt, dann ist φ ein **Automorphismus**.

Bemerkung: Ein Automorphismus für einen Graphen G ist eine strukturerhaltende Abbildung von G auf sich selbst.

Beispiel 1.39

Für den linken Graphen aus Beispiel 1.37 ist

$$\{a \rightarrow c, b \rightarrow b, c \rightarrow a, d \rightarrow e, e \rightarrow d\}$$

ein Automorphismus.

Bäume

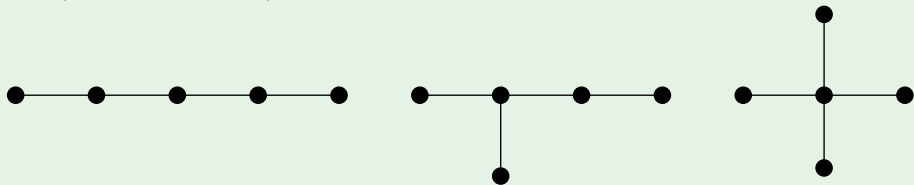
Definition 1.40

Es sei $G = (V, E)$ ein Graph. G heißt **Wald (forest)** gdw. G keinen Kreis enthält.

G heißt **Baum (tree)** gdw. G ein Wald und zusammenhängend ist.

Beispiel 1.41

Die (nichtisomorphen) Bäume mit fünf Knoten.



Charakterisierung von Bäume

Satz 1.42

Für einen Graphen $G = (V, E)$ mit $|V| = n$ sind die folgenden Aussagen äquivalent:

- 1 G ist ein Baum.
- 2 Je zwei Knoten von G sind durch genau einen Weg verbunden.
- 3 G ist zusammenhängend, aber für jede Kante $e \in E$ ist $G' = (V, E \setminus \{e\})$ nicht zusammenhängend.
- 4 G ist zusammenhängend und hat genau $n - 1$ Kanten.
- 5 G ist kreisfrei und hat genau $n - 1$ Kanten.
- 6 G ist kreisfrei, aber für je zwei nicht adjazente Knoten v, w von G enthält $G'' = (V, E \cup \{\{v, w\}\})$ genau einen Kreis.

Beweis.

Der Beweis zur Äquivalenz von (1) bis (3) ist Übungsaufgabe.

Gerichtete Graphen

Für viele Anwendungen ist es sinnvoll, die Kanten mit einer Richtung zu versehen.

Definition 1.43

Ein **gerichteter Graph (directed graph)** ist ein Paar $G = (V, A)$ bestehend aus den Mengen

- V , der Menge der Knoten und
- A , der Menge der **gerichteten Kanten (arcs)**, die aus geordneten Paaren (v, w) mit $v, w \in V, v \neq w$ besteht.

Für eine gerichtete Kante $a = (v, w)$ heißt v der **Anfangsknoten (initial vertex)** und w der **Endknoten (terminal vertex)** von a .

Bemerkung: Man kann ungerichtete Graphen als gerichtete Graphen betrachten, bei denen die Relation A symmetrisch ist.

Definition 1.44

Es sei $G = (V, A)$ ein gerichteter Graph.

- $\text{indeg}(v) := |\{(x, v) | (x, v) \in A\}|$ heißt der **Eingangsgrad** von $v \in V$.
- $\text{outdeg}(v) := |\{(v, y) | (v, y) \in A\}|$ heißt der **Ausgangsgrad** von $v \in V$.
- Ein **gerichteter Kantenzug** ist eine Folge (v_0, \dots, v_n) von Knoten mit $e_i := (v_{i-1}, v_i) \in A$ für $i = 1, \dots, n$.
- Die Begriffe aus Definition 1.26 und Definition 1.28 werden analog auf gerichtete Graphen übertragen.
- Der einem gerichteten Graph $G = (V, A)$ **zugeordnete ungerichtete Graph** $G' = (V, A')$ ist definiert durch: $\{v, w\} \in A'$ gdw. $(v, w) \in A$ oder $(w, v) \in A$.
- G heißt **zusammenhängend** gdw. der zugeordnete ungerichtete Graph G' zusammenhängend ist.
- G heißt **stark zusammenhängend** gdw. es für je zwei Knoten $v, w \in V$ einen gerichteten Weg von v nach w gibt.

Handschlaglemma für gerichtete Graphen

Lemma 1.45

Für einen gerichteten Graphen $G = (V, A)$ gilt:

$$\sum_{v \in V} \text{indeg}(v) = \sum_{v \in V} \text{outdeg}(v)$$

Beispiel 1.46

Tafel .

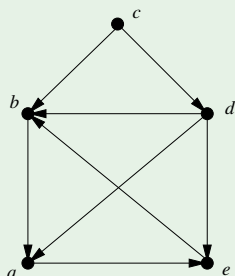
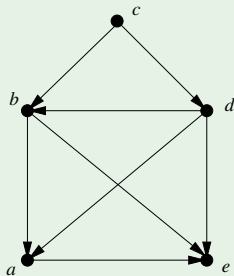
DAGs

Definition 1.47

Ein gerichteter Graph $G = (V, A)$ heißt **DAG (dag, directed acyclic graph)** gdw. G keinen einfachen gerichteten Kreis der Länge ≥ 2 enthält.

Beispiel 1.48

Der linke Graph ist ein DAG, der rechte nicht.



Zusammenfassung

- Ein Graph $G = (V, E, \gamma)$ repräsentiert die zweistelligen Beziehungen zwischen den Elementen einer Menge V .
- Ein **schlichter Graph** $G = (V, E)$ enthält weder Schlingen noch parallele Kanten.
- **Schubfachprinzip**
- wichtige Grundbegriffe
- Ein **Baum** ist kreisfrei und zusammenhängend.
- **gerichtete Graphen**

Kapitel 2

Repräsentationen von Graphen in Computern

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

Inhalt

- 2 Repräsentationen von Graphen in Computern
 - Matrizen- und Listendarstellung von Graphen
 - Anzahl der Kantenzüge zwischen zwei Knoten
 - Eigenwertprobleme
 - Lineare Differenzgleichungen

Adjazenzmatrix

Definition 2.1

Gegeben sei ein Graph $G = (V, E)$ mit $V = \{v_1, \dots, v_n\}$, $n \geq 1$. Dann kann G in Form einer $n \times n$ -Matrix repräsentiert werden. Es sei

$$a_{ij} = \begin{cases} 1 & \text{falls } \{v_i, v_j\} \in E \\ 0 & \text{sonst.} \end{cases}$$

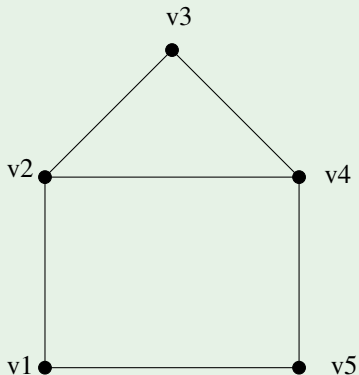
$\mathbf{A}_G = (a_{ij})_{i,j \in \{1, \dots, n\}}$ heißt die **Adjazenzmatrix (adjacency matrix)** von G .

Bemerkungen:

- \mathbf{A}_G ist symmetrisch und $a_{ij} = 0$, $1 \leq i \leq n$.
- Analog kann die Adjazenzmatrix für die Darstellung gerichteter Graphen verwendet werden. Sie ist dann i.d.R. nicht symmetrisch.

Adjazenzmatrix (2)

Beispiel 2.2



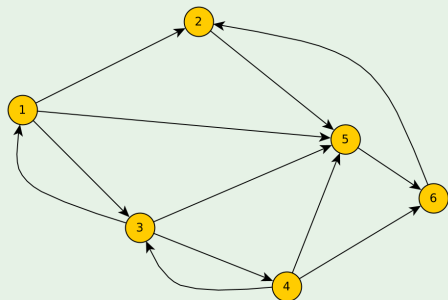
$$\mathbf{A}_G = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Adjazenzmatrix (3)

- Es kann in Zeit $O(1)$ überprüft werden, ob zwei Knoten v_i und v_j **adjazent** sind.
- $\deg(v_i)$ ist gleich der Zeilensumme der i -ten Zeile (bzw. der Spaltensumme der i -Spalte).
Aufwand: $O(|V|)$
- Ermittlung der **Nachbarn zu einem Knoten** v_i : Suche in der i -ten Zeile/Spalte
- notwendiger **Speicherplatz**: $O(|V|^2)$
- Platzverbrauch **ineffizient für bestimmte Graphklassen**, z.B. Bäume, planare Graphen (siehe Kapitel 6)

Adjazenzmatrix für gerichtete Graphen

Beispiel 2.3



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

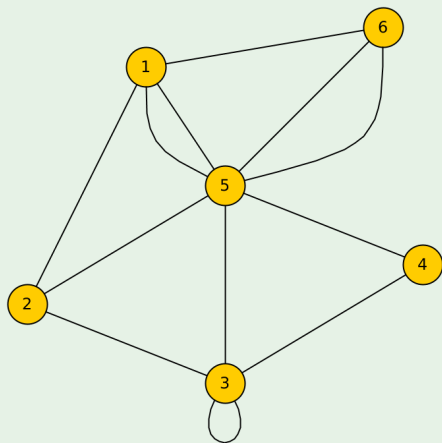
Adjazenzmatrix für nicht schlichte Graphen

- Für nicht schlichte Graphen gibt a_{ij} die Anzahl der Kanten zwischen v_i und v_j an.
- Wenn Schlingen vorliegen, sind die Diagonalelemente der entsprechenden Knoten ungleich 0. Das Element a_{ii} gibt dann die Anzahl der Schlingen am Knoten v_i an.
- Bei der Gradermittlung müssen die Diagonalelemente doppelt gezählt werden:

$$\deg(v_i) = 2 \cdot a_{ii} + \sum_{k=1, k \neq i}^n a_{ik}.$$

Adjazenzmatrix für nicht schlichte Graphen

Beispiel 2.4



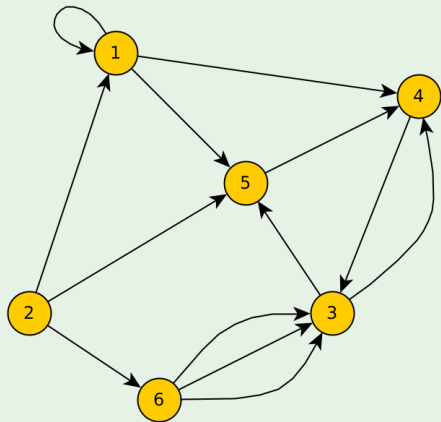
$$\mathbf{A} = \begin{pmatrix} 0 & 1 & 0 & 0 & 2 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 \\ 2 & 1 & 1 & 1 & 0 & 2 \\ 1 & 0 & 0 & 0 & 2 & 0 \end{pmatrix}$$

Adjazenzmatrix: gerichtet und nicht schlicht

- Prinzipiell können natürlich auch **gerichtete Graphen nicht schlicht** sein,
- d.h. an Knoten existieren Schlingen oder
- zwischen zwei Knoten a und b gibt es mehrere Kanten **mit der gleichen Richtung** (von a nach b).

Gerichteter und nicht schlichter Graph

Beispiel 2.5



$$\mathbf{A} = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 & 0 \end{pmatrix}$$

Adjazenzliste

Definition 2.6

Gegeben sei ein Graph $G = (V, E)$ mit $V = \{v_1, \dots, v_n\}$, $n \geq 1$. Dann kann G in Form einer Liste von n -Listen A_i repräsentiert werden. Für $1 \leq i \leq n$ seien $v_{i_1}, v_{i_2}, \dots, v_{i_{n_i}}$ die mit $v_i \in V$ adjazenten Knoten. Die Liste

$$A_i = (v_{i_1}, v_{i_2}, \dots, v_{i_{n_i}})$$

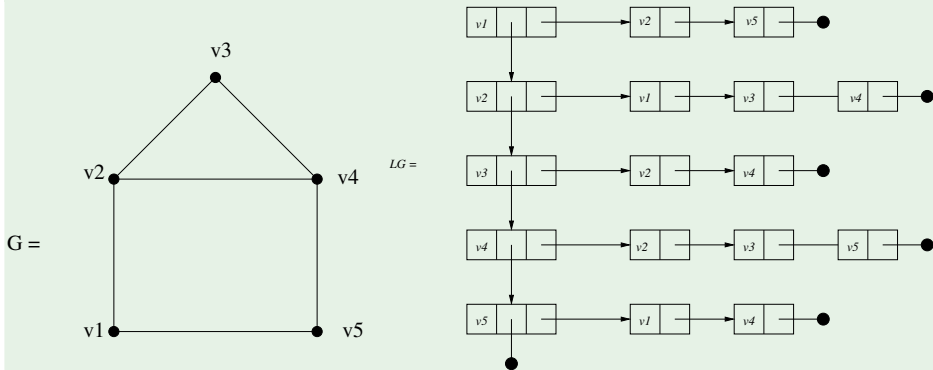
heißt die **Adjazenzliste** von $v_i \in V$.

Die Liste $L_G = (A_1, \dots, A_n)$ ist die **Adjazenzlistendarstellung** von G .

Für einen gerichteten Graphen $G = (V, A)$ enthält die Adjazenzliste A_i die Knoten $w \in V$, für die $(v_i, w) \in A$ gilt.

Adjazenzliste (2)

Beispiel 2.7

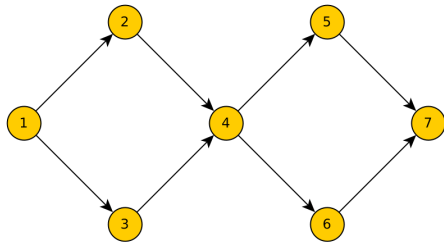


Adjazenzliste (3)

- Um zu überprüfen, ob zwei Knoten v_i und v_j adjazent sind, muss die Adjazenzliste von v_i durchsucht werden.
- Dies ist nicht in $O(1)$ möglich. Der genaue Aufwand hängt von der Implementierung der Adjazenzliste ab.
- Der Knotengrad entspricht der Länge der Adjazenzliste.
- Die Nachbarn zu einem Knoten liegen direkt in der Adjazenzliste vor.
- notwendiger **Speicherplatz**: $O(|V| + |E|)$

Anzahl Wege zwischen zwei Knoten (1)

Wir betrachten als Beispiel den folgenden gerichteten Graphen G mit seiner Adjazenzmatrix:



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

Anzahl Wege zwischen zwei Knoten (2)

Wir bilden die Potenzen der Adjazenzmatrix \mathbf{A} :

$$\mathbf{A}^2 = \begin{pmatrix} 0 & 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{A}^3 = \begin{pmatrix} 0 & 0 & 0 & 0 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{A}^4 = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \mathbf{A}^k = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{für } k \geq 5$$

- Das Element $a_{i,j}$ der Matrizen \mathbf{A}^k gibt hier die **Anzahl der (einfachen) Wege der Länge k von i nach j** an.

Anzahl der Kantenzüge zwischen zwei Knoten

Satz 2.8

Es sei $G = (V, E)$ ein Graph mit der Adjazenzmatrix $\mathbf{A} = (a_{ij})$.

Dann gibt das Element $a_{ij}^{(r)}$ der Matrix \mathbf{A}^r die **Anzahl der Kantenzüge der Länge r von v_i nach v_j** an.

Beweis.

Induktion über r .

$r = 1$: Damit gilt $\mathbf{A}^r = \mathbf{A}$. Die Adjazenzmatrix gibt genau die Kantenzüge der Länge 1 an.

$r \rightarrow r + 1$:

- Jeder Kantenzug der Länge $r + 1$ zwischen zwei Knoten v_i und v_j besteht aus einem Kantenzug der Länge r zwischen v_i und einem Knoten v_k sowie der Kante $\{v_k, v_j\}$.
- Nach I.V. gibt \mathbf{A}^r die Anzahl der Kantenzüge der Länge r zwischen zwei Knoten an.

Fortsetzung Beweis.

- Es gilt

$$a_{ij}^{(r+1)} = \sum_{k=1}^{|V|} a_{ik}^{(r)} \cdot a_{kj}.$$

- Da $a_{kj} = 1$ gdw. zwischen v_i und v_j eine Kante ist, beschreibt diese Formel die Anzahl der Möglichkeiten, einen Kantenzug der Länge $r + 1$ zwischen v_i und v_j aus einem Kantenzug der Länge r zwischen v_i und einem Knoten v_k sowie der Kante $\{v_k, v_j\}$ zu bilden.

Folgerung 2.9

Es sei $G = (V, E)$ ein Graph mit der Adjazenzmatrix $\mathbf{A} = (a_{ij})$.
Dann gibt das Element b_{ij} der Matrix

$$\mathbf{B} = \mathbf{A} + \mathbf{A}^2 + \dots + \mathbf{A}^p$$

die Anzahl der Kantenzüge mit einer Länge $\leq p$ von v_i nach v_j an.

Folgerung 2.10

Es sei $G = (V, E)$ ein Graph mit der Adjazenzmatrix $\mathbf{A} = (a_{ij})$ und es sei

$$\mathbf{B} = \mathbf{A} + \mathbf{A}^2 + \dots + \mathbf{A}^{|V|-1}.$$

Dann gilt: G ist *genau dann zusammenhängend*, wenn $b_{ij} > 0$ für alle $i \neq j$ gilt.

Beweis.

Wenn G zusammenhängend ist,

- gibt es zwischen zwei beliebigen Knoten v_i und v_j mindestens einen Weg,
- damit auch mindestens einen einfachen Weg.
- Ein einfacher Weg hat eine Länge $\leq |V| - 1$.
- Damit liefert der einfache Weg (als Kantenzug) einen Beitrag zu b_{ij} .
- Also folgt $b_{ij} > 0$.

Fortsetzung Beweis.

Andererseits folgt aus $b_{ij} > 0$,

- dass es mindestens einen Kantenzug und damit auch einen Weg von v_i nach v_j gibt.
- Somit folgt aus $b_{ij} > 0$ für alle $i \neq j$, dass es zwischen je zwei Knoten von G einen Weg gibt.
- Damit ist G zusammenhängend.

Bemerkungen

- Weil in Dags **jeder Kantenzug ein gerichteter einfacher Weg** ist, liefert \mathbf{A}^r dort sogar die Anzahl der einfachen Wege der Länge r .
- Auch können wir mit diesem Ansatz prinzipiell testen, **ob ein gerichteter Graph kreisfrei ist**: für $p = |V|$ müssen die b_{ii} alle ungleich 0 sein.
- Sowohl für die Kreisfreiheit als auch für den Zusammenhang sind diese Berechnungsansätze aber **ineffizient**.
- Im nächsten Kapitel werden wir effizientere Algorithmen für diese Probleme kennenlernen.

Eigenwerte und Eigenvektoren

Definition 2.11

Ein Vektor $\mathbf{x} \in \mathbb{R}^n$ mit $\mathbf{x} \neq \mathbf{0}$ heißt **Eigenvektor** der quadratischen $n \times n$ -Matrix \mathbf{A} zum **Eigenwert** $\lambda \in \mathbb{R}$, wenn gilt

$$\mathbf{A}\mathbf{x} = \lambda\mathbf{x}.$$

Die Eigenwerte der Matrix \mathbf{A} sind alle Werte λ , für die ein Eigenvektor existiert.

Bemerkungen zu Eigenwerten und Eigenvektoren

Da der Nullvektor natürlich immer auf sich selbst abgebildet wird, verlangen wir von einem Eigenvektor stets $\mathbf{x} \neq \mathbf{0}$.

In der Ebene (2D) beschreiben Eigenwerte und Eigenvektoren Fixgeraden von linearen Abbildungen.

Beispiel 2.12

Sei

$$\mathbf{A} = \begin{pmatrix} 0 & 4 \\ 1 & 0 \end{pmatrix}.$$

Dann hat \mathbf{A} die Eigenvektoren $\mathbf{u} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ und $\mathbf{v} = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$ mit den Eigenwerten $\lambda = 2$ und $\mu = -2$.

Die durch die Eigenvektoren definierten Geraden werden auf sich selbst abgebildet (als Menge, nicht punktweise).

Eine punktweise Abbildung wäre eine **Fixpunktgerade**. Dies sind Geraden definiert durch Eigenvektoren zum Eigenwert 1.

Eigenwertprobleme betrachtet man **üblicherweise nicht in \mathbb{R} , sondern in \mathbb{C}** , weil für eine allgemeine reelle Matrix die Existenz von reellen Eigenwerten nicht garantiert ist.

Beispiel 2.13

I. A. liefert **eine Drehung keine Fixgeraden**. So hat die Drehmatrix

$$\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$$

nur komplexe Eigenwerte (Drehung um 90° bzw. $\frac{\pi}{2}$).

Eigenwerte symmetrischer Matrizen

Satz 2.14

Es sei \mathbf{A} eine reelle symmetrische $n \times n$ -Matrix.

- Dann sind *alle Eigenwerte von \mathbf{A} reell*.
- *Es gibt n linear unabhängige Eigenvektoren von \mathbf{A} .*
- *Linear unabhängige Eigenvektoren von \mathbf{A} sind zueinander orthogonal (stehen senkrecht aufeinander).*

Berechnung von Eigenwerten und -vektoren

Die Eigenwerte ergeben sich aus den Nullstellen des **charakteristischen Polynoms**

$$P(\lambda) = \det(\mathbf{A} - \lambda\mathbf{E}) = |\mathbf{A} - \lambda\mathbf{E}|.$$

Hierbei ist \mathbf{E} die n -dimensionale Einheitsmatrix.

Beispiel 2.15

Für $\mathbf{A} = \begin{pmatrix} 1 & 6 \\ 1 & 0 \end{pmatrix}$ ergibt sich

$$P(\lambda) = (1 - \lambda)(-\lambda) - 6 = \lambda^2 - \lambda - 6 \stackrel{!}{=} 0.$$

Mit der **p-q-Formel** erhalten wir

$$\lambda_{1,2} = \frac{1}{2} \pm \sqrt{\frac{1}{4} + 6} = \frac{1}{2} \pm \frac{5}{2} = 3 \text{ bzw. } -2.$$

Fortsetzung Beispiel.

Für $\lambda = 3$ ergibt sich der Eigenvektor $\mathbf{u} = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$ aus $u_1 = 3u_2$ (zweite Zeile von \mathbf{A}). Somit ist $\mathbf{u} = \begin{pmatrix} 3 \\ 1 \end{pmatrix}$ ein Eigenvektor zum Eigenwert 3.

Analog ergibt sich, dass $\mathbf{v} = \begin{pmatrix} -2 \\ 1 \end{pmatrix}$ ein Eigenvektor zum Eigenwert -2 ist.

Beachten Sie: Die Eigenvektoren sind hier nicht orthogonal, weil \mathbf{A} nicht symmetrisch ist.

Was bringen uns Eigenwerte?

Wir können bspw. die Berechnung einer n -fachen Anwendung einer linearen Abbildung **deutlich beschleunigen**.

Beispiel 2.16

Wo landet der Punkt $\mathbf{p} = (11, -3)$, wenn wir ihn n -mal mit der linearen Abbildung bzw. der Matrix \mathbf{A} aus Beispiel 2.15 transformieren?

Bei $\mathbf{p}' = \mathbf{A}^n \mathbf{p}$

Naive Berechnung: n -fach Multiplikation von \mathbf{A} mit einem Vektor, beginnend mit dem Vektor \mathbf{p} .

$$\begin{aligned}\mathbf{p}^{(0)} &= \mathbf{p} \\ \mathbf{p}^{(i+1)} &= \mathbf{A}\mathbf{p}^{(i)} \text{ für } i = 0, 1, \dots\end{aligned}$$

Es gilt dann $\mathbf{p}' = \mathbf{p}^{(n)}$.

Aufwand: $4n$ Multiplikationen und $2n$ Additionen, also $O(n)$.

Fortsetzung Beispiel.

Angenommen wir könnten \mathbf{p} als Linearkombination der Eigenvektoren \mathbf{u} und \mathbf{v} darstellen, also

$$\mathbf{p} = \alpha\mathbf{u} + \beta\mathbf{v}.$$

Dann ergibt sich

$$\begin{aligned}\mathbf{p}' &= \mathbf{A}^n\mathbf{p} = \mathbf{A}^n(\alpha\mathbf{u} + \beta\mathbf{v}) = \alpha\mathbf{A}^n\mathbf{u} + \beta\mathbf{A}^n\mathbf{v} \\ &= \alpha\lambda_1^n\mathbf{u} + \beta\lambda_2^n\mathbf{v} \\ &= \alpha 3^n\mathbf{u} + \beta(-2)^n\mathbf{v}.\end{aligned}$$

Aufwand: 2 Exponentenberechnungen, 6 Multiplikationen, 2 Additionen, also $O(1)$

Fortsetzung Beispiel.

Aber wie bestimmen wir α und β ?

Prinzipiell kein Problem, sie ergeben sich aus dem linearen GLS

$$\alpha \mathbf{u} + \beta \mathbf{v} = \begin{pmatrix} 11 \\ -3 \end{pmatrix},$$

also

$$\begin{aligned} 3\alpha - 2\beta &= 11 \\ \alpha + \beta &= -3. \end{aligned}$$

Lösung hier: $\alpha = 1, \beta = -4$, also

$$\mathbf{p}' = \mathbf{A}^n \mathbf{p} = \begin{pmatrix} 3^{n+1} + (-1)^n \cdot 2^{n+3} \\ 3^n + (-1)^{n+1} \cdot 2^{n+2} \end{pmatrix}.$$

Fortsetzung Beispiel.

Und für einen beliebigen Punkt $\mathbf{p} = \begin{pmatrix} x \\ y \end{pmatrix}$?

Dann lautet das GLS

$$\begin{aligned} 3\alpha - 2\beta &= x \\ \alpha + \beta &= y. \end{aligned}$$

Allgemeine Lösung:

$$\alpha = \frac{x + 2y}{5}, \beta = \frac{3y - x}{5}.$$

Ebenfalls in $O(1)$ berechenbar.

Fazit: Für jedes $\mathbf{p} \in \mathbb{R}^2$ können wir $\mathbf{A}^n \mathbf{p}$ in $O(1)$ berechnen.

Berechnung der Potenzen einer Adjazenzmatrix

- Zur Berechnung der Anzahl an Kantenzügen zwischen Knoten benötigen wir die Potenzen der Adjazenzmatrix.
- Können wir auch \mathbf{A}^k mit der Hilfe von Eigenwerten und -vektoren explizit berechnen?
- Ansatz: Wir betrachten die einzelnen Spaltenvektoren der Adjazenzmatrix.

$$\mathbf{A} = \left(\mathbf{a}^{(1)} \mathbf{a}^{(2)} \dots \mathbf{a}^{(n)} \right)$$

wobei $\mathbf{a}^{(i)}$ der i -te Spaltenvektor von \mathbf{A} ist.

- Dann gilt:

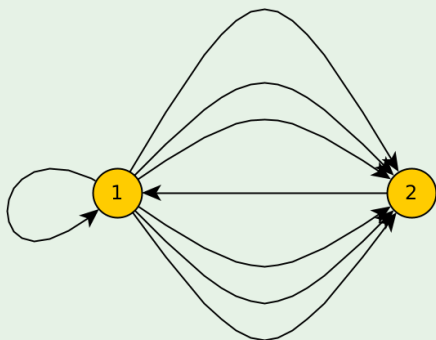
$$\begin{aligned} \mathbf{A}^k &= \mathbf{A}^{k-1} \left(\mathbf{a}^{(1)} \mathbf{a}^{(2)} \dots \mathbf{a}^{(n)} \right) \\ &= \left(\mathbf{A}^{k-1} \mathbf{a}^{(1)} \mathbf{A}^{k-1} \mathbf{a}^{(2)} \dots \mathbf{A}^{k-1} \mathbf{a}^{(n)} \right) \end{aligned}$$

- Damit können wir \mathbf{A}^k mit Hilfe der Techniken aus Beispiel 2.16 berechnen.

Berechnungsbeispiel

Beispiel 2.17

Graph zur Adjazenzmatrix \mathbf{A} aus
Beispiel 2.15.



Darstellung der Spalten von \mathbf{A} als
Linearkombination von
Eigenvektoren:

$$\begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{3}{5} \begin{pmatrix} 3 \\ 1 \end{pmatrix} + \frac{2}{5} \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 6 \\ 0 \end{pmatrix} = \frac{6}{5} \begin{pmatrix} 3 \\ 1 \end{pmatrix} - \frac{6}{5} \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

Fortsetzung Beispiel.

$$\mathbf{A}^n = \left(\mathbf{A}^{n-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \mathbf{A}^{n-1} \begin{pmatrix} 6 \\ 0 \end{pmatrix} \right)$$

Es gilt

$$\mathbf{A}^{n-1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \frac{3}{5} 3^{n-1} \begin{pmatrix} 3 \\ 1 \end{pmatrix} + \frac{2}{5} (-2)^{n-1} \begin{pmatrix} -2 \\ 1 \end{pmatrix}$$

und

$$\mathbf{A}^{n-1} \begin{pmatrix} 6 \\ 0 \end{pmatrix} = \frac{6}{5} 3^{n-1} \begin{pmatrix} 3 \\ 1 \end{pmatrix} - \frac{6}{5} (-2)^{n-1} \begin{pmatrix} -2 \\ 1 \end{pmatrix}.$$

Damit erhalten wir

$$\mathbf{A}^n = \frac{1}{5} \begin{pmatrix} 3^{n+1} + (-1)^n \cdot 2^{n+1} & 2 \cdot 3^{n+1} - 3 \cdot (-1)^n \cdot 2^{n+1} \\ 3^n + (-1)^{n-1} \cdot 2^n & 2 \cdot 3^n - 3 \cdot (-1)^{n-1} \cdot 2^n \end{pmatrix}.$$

Somit haben wir für alle Kantenzüge der Länge n eine explizite Formel.

Lineare Differenzgleichungen

Die **Fibonacci-Zahlen** F_n sind definiert durch

$$F_0 = 0$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad \text{für } n \geq 2.$$

Die letzte Zeile ist ein Beispiel für eine **homogene lineare Differenzgleichung** zweiter Ordnung mit konstanten Koeffizienten.

Allgemein k -ter Ordnung:

$$F_n = a_1 \cdot F_{n-1} + \cdots + a_k \cdot F_{n-k}$$

Lösungsansatz für homogene lineare Differenzgleichungen

Wir stellen die homogene lineare Differenzgleichung in **Matrixform** dar.
Beispiel für die Fibonacci-Zahlen:

$$\begin{aligned} \begin{pmatrix} F_n \\ F_{n-1} \end{pmatrix} &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-2} \\ F_{n-3} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^{n-1} \begin{pmatrix} F_1 \\ F_0 \end{pmatrix} \end{aligned}$$

Jetzt können wir vorgehen wie in Beispiel 2.16:

- Wir stellen das **charakteristische Polynom** von $\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ auf,
- berechnen die **Eigenwerte** der Matrix,
- anschließend die **Eigenvektoren**,
- stellen $\begin{pmatrix} F_1 \\ F_0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ als **Linearkombination der Eigenvektoren** dar und
- erhalten so eine **explizite Formel** für F_n .

Fibonacci-Zahlen

Formel von **Moivre-Binet**:

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right]$$

Charakteristisches Polynom

$$P(\lambda) = \lambda^2 - \lambda - 1$$

mit den Eigenwerten

$$\lambda = \frac{1 + \sqrt{5}}{2} \quad \text{und} \quad \mu = \frac{1 - \sqrt{5}}{2}$$

Homogene lineare Differenzgleichung

Definition 2.18

Für $a_i \in \mathbb{R}, i = 1, \dots, k$ heißt die Gleichung

$$F_n = a_1 \cdot F_{n-1} + \dots + a_k \cdot F_{n-k}$$

homogene lineare Differenzgleichung k -ter Ordnung mit konstanten Koeffizienten.

In Matrixdarstellung können wir solch eine Differenzgleichung schreiben als

$$\begin{pmatrix} F_n \\ F_{n-1} \\ \vdots \\ \vdots \\ F_{n-k+1} \end{pmatrix} = \begin{pmatrix} a_1 & a_2 & \cdots & \cdots & a_k \\ 1 & 0 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ & & & 1 & 0 \end{pmatrix} \begin{pmatrix} F_{n-1} \\ F_{n-2} \\ \vdots \\ \vdots \\ F_{n-k} \end{pmatrix}.$$

Charakteristisches Polynom für eine homogen lineare Differenzgleichung

Satz 2.19

Das charakteristische Polynom $P(\lambda)$ einer homogenen linearen Differenzgleichung k -ter Ordnung mit konstanten Koeffizienten hat die Form

$$P(\lambda) = (-1)^k \left(\lambda^k - a_1 \cdot \lambda^{k-1} - a_2 \cdot \lambda^{k-2} - \dots - a_k \right).$$

Beweis.

Vollständige Induktion und **Entwicklung der Matrix $\mathbf{A} - \lambda \mathbf{E}$ nach der $(k + 1)$ -ten Spalte**. Übungsaufgabe.

Bemerkungen zum charakteristischen Polynom

- Damit können wir das charakteristische Polynom **direkt an der Gleichung “ablesen”**, die Berechnung von $\det(\mathbf{A} - \lambda\mathbf{E})$ ist nicht notwendig.
- Da wir nur an den Nullstellen von $P(\lambda)$ interessiert sind, **spielt der Faktor $(-1)^k$ keine Rolle.**

Lösungen für eine homogene lineare Differenzgleichung

Satz 2.20

Es sei λ eine Nullstelle mit Vielfachheit m des charakteristischen Polynoms. Dann sind die Folgen

$$F_n = n^i \lambda^n$$

für $i = 0, \dots, m - 1$ Lösungen der homogenen linearen Differenzgleichung.

Beispiel 2.21

Aus der Differenzgleichung

$$F_n = 7 F_{n-1} - 16 F_{n-2} + 12 F_{n-3}$$

ergibt sich das charakteristische Polynom

$$\begin{aligned} P(\lambda) &= \lambda^3 - 7\lambda^2 + 16\lambda - 12 \\ &= (\lambda - 3)(\lambda - 2)^2. \end{aligned}$$

Also ist 3 eine einfache Nullstelle und 2 ist eine zweifache Nullstelle. Damit sind

$$F_n = 3^n$$

$$F_n = 2^n$$

$$F_n = n 2^n$$

Lösungen der homogenen linearen Differenzgleichung.

Anfangswertprobleme

- In der Praxis hat man neben der Differenzgleichung häufig Anfangsbedingungen für die ersten k Folgenglieder.
- Dieses Problem nennt man **Anfangswertproblem**.
- Beispiel Fibonacci-Zahlen: Neben $F_n = F_{n-1} + F_{n-2}$ wird zusätzlich $F_0 = 0$ und $F_1 = 1$ verlangt.
- Zur Lösung des Anfangswertproblems **müssen wir eine Linearkombination der homogenen Lösungen finden**.
- Dies resultiert in einem linearen Gleichungssystem.

Beispiel 2.22

Zu der Differenzgleichung von Beispiel 2.21 wollen wir das Anfangswertproblem

$$F_0 = 2, F_1 = 7, F_2 = 21$$

lösen. Es muss gelten

$$F_n = \alpha \cdot 3^n + \beta \cdot 2^n + \gamma \cdot n 2^n.$$

Daraus ergibt sich das lineare GLS

$$\text{für } n = 0 : \quad \alpha + \beta = 2$$

$$\text{für } n = 1 : \quad 3\alpha + 2\beta + 2\gamma = 7$$

$$\text{für } n = 2 : \quad 9\alpha + 4\beta + 8\gamma = 21$$

mit der Lösung $\alpha = \beta = \gamma = 1$. Also wird das Anfangswertproblem gelöst durch:

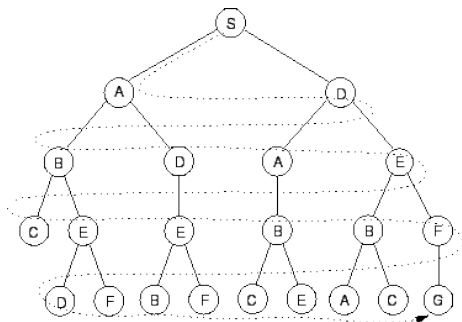
$$F_n = 3^n + 2^n + n 2^n$$

Zusammenfassung

- **Adjazenzmatrix** und **Adjazenzliste** zur Repräsentation von Graphen
- Berechnung der Anzahl an Kantenfolgen zwischen Knoten mit Hilfe der Potenzen der Adjazenzmatrix
- **Eigenwerte** und **Eigenvektoren** zur expliziten Berechnung der **Potenzen einer Adjazenzmatrix**
- Eigenwerte zur Lösung von **Anfangswertproblemen** homogener linearer Differenzgleichungen mit konstanten Koeffizienten

Kapitel 3

Durchsuchen von Graphen



Inhalt

- 3 Durchsuchen von Graphen
 - Tiefensuche
 - Breitensuche
 - Topologisches Sortieren

Tiefensuche

- Verallgemeinerung des preorder-Durchlaufprinzips für binäre Bäume
- Kanten werden ausgehend von dem zuletzt entdeckten Knoten v , der mit noch unerforschten Kanten inzident ist, erforscht.
- Erreicht man von v aus einen noch nicht erforschten Knoten w , so verfährt man mit w genauso wie mit v .
- Wenn alle mit w inzidenten Kanten erforscht sind, erfolgt ein **Backtracking** zu v .
- Die Knoten $v \in V$ erhalten in der Reihenfolge ihres Erreichens eine **DFS-Nummer** $t(v)$.
Zu Beginn setzen wir $t(v) = \infty$ für alle $v \in V$.

Algorithmus Tiefensuche

Algorithmus 3.1

Für einen Knoten v sei die Prozedur $DFSEARCH(v)$ wie folgt definiert:

```
 $i := i + 1; t(v) := i; N(v) := \{w | \{v, w\} \in E\};$   
while  $\exists w \in N(v)$  mit  $t(w) = \infty$  do  
     $N(v) := N(v) \setminus \{w\};$   
     $B := B \cup \{\{v, w\}\};$   
     $DFSEARCH(w);$   
end
```

Algorithmus 3.2 (Tiefensuche, Depth-First Search)

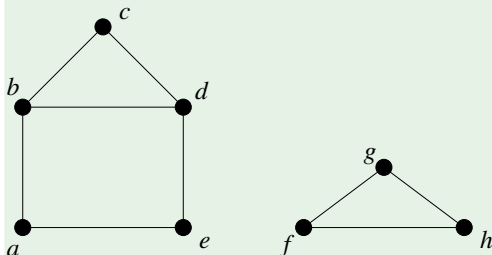
Es sei $G = (V, E)$ ein Graph.

```
 $B := \emptyset; i := 0;$   
for all  $v \in V$  do  $t(v) := \infty;$   
while  $\exists v \in V$  mit  $t(v) = \infty$  do  $DFSEARCH(v);$ 
```

Das Ergebnis für $t(v)$ und B hängt i.d.R. von der Repräsentation des Graphen (z.B. als Adjazenzliste) ab.


Beispiel 3.3

Wir betrachten den folgenden Graphen:



Adjazenzlisten:

| | |
|-----|---------------------------------|
| a | $b \rightarrow e$ |
| b | $a \rightarrow c \rightarrow d$ |
| c | $b \rightarrow d$ |
| d | $b \rightarrow c \rightarrow e$ |
| e | $a \rightarrow d$ |
| f | $g \rightarrow h$ |
| g | $f \rightarrow h$ |
| h | $f \rightarrow g$ |

Durchlauf und weitere Beispiele: Tafel 

Eigenschaften der Tiefensuche

- Durch DFS wird jeder Knoten besucht und jede Kante wird untersucht.
- Ist G nicht zusammenhängend, dann werden durch DFS nacheinander die Zusammenhangskomponenten ermittelt.
- DFS unterteilt die Kantenmenge E in die Menge der **Baumkanten** (die Menge B) und die Menge der **Rückwärtskanten** ($E \setminus B$).
- (V, B) bildet einen aufspannenden Untergraphen von G , der ein Wald ist.
- Ist G zusammenhängend, dann ist (V, B) ein sogenannter **aufspannender Baum** von G .

Satz 3.4

Der Zeitaufwand für DFS ist $O(|V| + |E|)$.

Definition 3.5

Es sei $G = (V, E)$. Ein Knoten $w \in V$ heißt **Nachfolger** von $v \in V$ gdw. $\text{DFSEARCH}(w)$ wird nach Eröffnung und vor Beendigung von $\text{DFSEARCH}(v)$ aufgerufen.

Lemma 3.6

Wenn $\{v, w\}$ eine Rückwärtskante ist, dann ist v Vorfahr von w oder umgekehrt.

D.h., durch DFS werden keine sogenannten Querkanten konstruiert.

Breitensuche

- Man geht an einem Knoten nicht sofort in die Tiefe, sondern bearbeitet zunächst alle noch nicht erreichten Nachbarn dieses Knotens.
- Anschließend fügt man die Knoten in eine **Warteschlange** ein.
- Ein Knoten gilt als erreicht, wenn er in die Warteschlange aufgenommen wurde.
- Solange die Warteschlange nicht leer ist, selektiert man einen Knoten aus der Warteschlange und geht wie oben beschrieben vor.
- Die Knoten $v \in V$ erhalten in der Reihenfolge ihres Herausnehmens aus der Warteschlange eine **BFS-Nummer** $b(v)$.

Algorithmus Breitensuche

Algorithmus 3.7

Für einen Knoten v sei die Prozedur $BFSEARCH(v)$ wie folgt definiert:

```
 $i := i + 1; b(v) := i; N(v) := \{w | \{v, w\} \in E\}$   
while  $\exists w \in N(v)$  mit  $r(w) = false$  do  
     $N(v) := N(v) \setminus \{w\};$   
     $B := B \cup \{\{v, w\}\};$   
    füge  $w$  an  $W$  an;  
     $r(w) := true;$   
end
```

Die Prozedur $BFSEARCH$ wird nun für den jeweils ersten Knoten der Warteschlange W ausgeführt.

Algorithmus 3.8 (Breitensuche, Breadth-First Search)

Es sei $G = (V, E)$ ein Graph.

```
B :=  $\emptyset$ ; i := 0; W := ();  
for all  $v \in V$  do  $b(v) := \infty$ ;  $r(v) := \text{false}$ ; end  
while  $W = ()$  and  $\exists v \in V : b(v) = \infty$  do  
  W := ( $v$ );  
   $r(v) := \text{true}$ ;  
  while  $W \neq ()$  do  
    wähle ersten Knoten  $w$  aus W;  
    entferne  $w$  aus W;  
    BFSEARCH( $w$ );  
  end  
end
```

Beispiel 3.9

Tafel .

Eigenschaften der Breitensuche

- Jeder Knoten wird besucht und jede Kante wird untersucht.
- Es werden nacheinander die Zusammenhangskomponenten ermittelt.
- (V, B) bildet einen aufspannenden Untergraphen, der ein Wald ist.

Satz 3.10

Der Zeitaufwand für BFS ist $O(|V| + |E|)$.

Halbordnung

Definition 3.11

Eine binäre Relation $R \subseteq V \times V$ heißt **Halbordnung** auf V gdw. folgendes gilt:

- 1 $\forall v \in V : (v, v) \in R$ (Reflexivität),
- 2 $\forall v, w \in V : (v, w) \in R \wedge (w, v) \in R \Rightarrow v = w$ (Antisymmetrie),
- 3 $\forall u, v, w \in V : (u, v) \in R \wedge (v, w) \in R \Rightarrow (u, w) \in R$ (Transitivität).

Beispiel 3.12

Die Teilbarkeitseigenschaft definiert eine Halbordnung auf den natürlichen Zahlen.

Topologische Ordnung

Lemma 3.13

Es sei $G = (V, A)$ ein DAG. Die Relation $R \subseteq V \times V$ mit

$$(v, w) \in R \iff \exists \text{ gerichteter Weg von } v \text{ nach } w$$

definiert eine Halbordnung auf V .

Definition 3.14

Es sei $G = (V, A)$ ein DAG mit $V = \{v_1, \dots, v_n\}$. Eine Knotenreihenfolge $L = (v_{i_1}, \dots, v_{i_n})$ aller Knoten aus V heißt **topologische Ordnung** von G gdw.

$$(v_{i_j}, v_{i_k}) \in A \implies i_j < i_k.$$

Beispiel 3.15

Tafel .

Topologisches Sortieren

Algorithmus 3.16

Gegeben sei ein DAG $G = (V, A)$ beschrieben durch seine Adjazenzlisten. Berechnet wird eine *topologische Sortierung*, die durch die Numerierung $t(v)$ gegeben ist.

```
 $k := 0; Q := \{v \in V \mid \text{indeg}(v) = 0\}$   
while  $Q \neq \emptyset$  do  
  wähle ein  $v$  aus  $Q$ ;  
   $k := k + 1$ ;  
   $t(v) := k$ ;  
   $Q := Q \setminus \{v\}$ ;  
  for all  $(v, w) \in A$  do  
     $\text{indeg}(w) := \text{indeg}(w) - 1$ ;  
    if  $\text{indeg}(w) = 0$  then  $Q := Q \cup \{w\}$ ;  
  end  
end
```


Beispiel 3.17

Tafel .

Satz 3.18

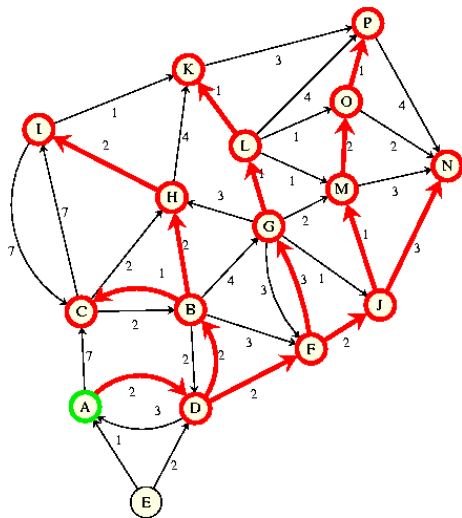
Algorithmus 3.16 liefert in $O(|V| + |E|)$ eine topologische Sortierung $L = (v_{i_1}, \dots, v_{i_n})$, wobei v_i der Knoten mit $t(v) = i$ ist.

Zusammenfassung des Kapitels

- Tiefensuche: Gehe in die Tiefe solange wie möglich.
- Knotennumerierung und aufspannender, kreisfreier Untergraph
- Breitensuche: Prinzip der Warteschlange
- Anwendungen:
 - ▶ systematisches Durchsuchen von Graphen, z.B. Labyrinth
 - ▶ Ermittlung von Eigenschaften eines Graphen
- Berechnung von zulässigen Reihenfolgen durch topologische Sortierung der Knoten

Kapitel 4

Kreis- und Wegeprobleme

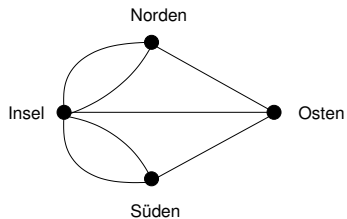


Inhalt

4 Kreis- und Wegeprobleme

- Eulersche Graphen
- Hamiltonsche Graphen
- Abstände in Graphen
- Abstände in Netzwerken

Das Königsberger Brückenproblem

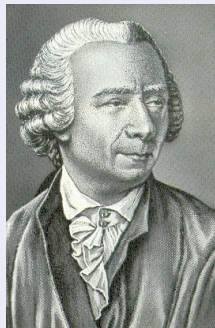


Eulerweg, Eulerkreis

Definition 4.1

Es sei $G = (V, E)$ ein Graph.

- Ein Weg, der jede Kante von G genau einmal enthält, heißt **eulerscher Weg** von G .
- Ein Kreis, der jede Kante von G genau einmal enthält, heißt **eulerscher Kreis** von G .
- G heißt **eulersch** gdw. G einen eulerschen Kreis enthält.



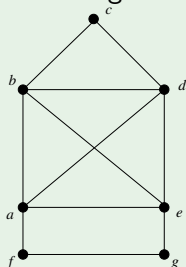
Leonhard Euler

Beispiel 4.2

Der Graph des Königsberger Brückenproblems scheint nicht eulersch zu sein, er scheint auch keinen eulerschen Weg zu enthalten.

Das “Haus des Nikolaus” enthält einen eulerschen Weg.

Das “Haus des Nikolaus mit Keller” ist eulersch:



Weitere Graphen: 

Charakterisierung von eulerschen Graphen

Satz 4.3 (Euler 1736)

Es sei $G = (V, E)$ ein Graph.

G hat einen eulerschen Weg gdw.

- G bis auf isolierte Knoten zusammenhängend ist und*
- für die Zahl u der Knoten mit ungeradem Grad gilt: $u = 0$ oder $u = 2$.*

Die Existenz eines Eulerkreises ist äquivalent mit $u = 0$.

Beweis zu Satz 4.3

Beweis.

1. " \implies ": G habe einen eulerschen Kreis $K = (v_0, v_1, \dots, v_{m-1}, v_0)$.
 - Dann ist G bis auf isolierte Knoten zusammenhängend und
 - tritt der Knoten v in der Folge v_0, v_1, \dots, v_{m-1} genau t -mal auf, so gilt $\deg(v) = 2t$, d.h. v hat geraden Grad.
2. " \impliedby ": Beweis durch vollständige Induktion über die Zahl der Knoten.

Induktionsanfang: Der Graph $G = (\{v_0\}, \{\})$ ist eulersch, denn (v_0) ist ein eulerscher Kreis.

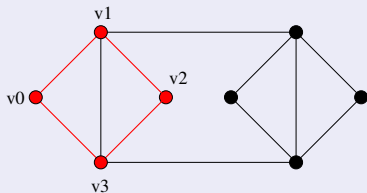
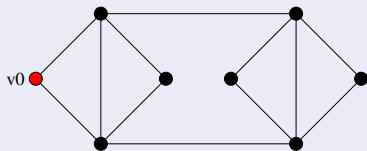
Induktionsannahme: Für Graphen mit höchstens n Knoten gelte die Behauptung.

Fortsetzung Beweis.

Induktionsschritt: Sei $G = (V, E)$ ein zusammenhängender Graph mit $n + 1$ Knoten und alle Knoten haben geraden Grad.

Wähle einen beliebigen Knoten $v_0 \in V$.

Wähle solange dies möglich ist Knoten v_1, v_2, \dots , so dass (v_0, \dots, v_i) jeweils ein Weg in G ist.



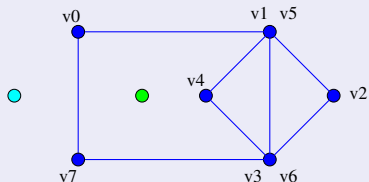
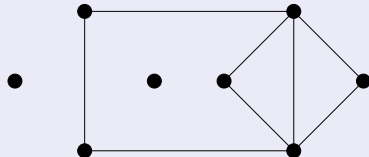
Unter den gegebenen Voraussetzungen entsteht so automatisch ein Kreis K . Sei E_k die Menge der Kanten in K .

Fortsetzung Beweis.

Wenn K alle Kanten aus E enthält, so ist K ein Eulerkreis.

Ansonsten bildet man den Restgraphen $G' = (V, E \setminus E_K)$.

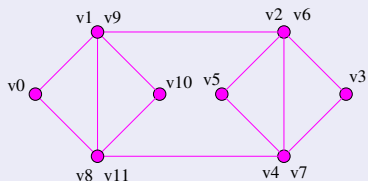
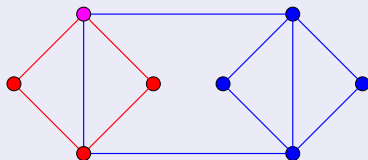
Für die Komponenten des Restgraphen gilt die Induktionsannahme.



Fortsetzung Beweis.

Nun fügt man die Kreise an Knoten zusammen, die in K und einer Komponente des Restgraphen enthalten sind.

Man läuft entlang K bis zu solch einem Knoten, dann entlang des Kreises des Restgraphen und anschließend wieder entlang K .



Berechnung eines Eulerkreises

Algorithmus 4.4 (Hierholzer 1873)

Es sei $G = (V, E)$ ein bis auf isolierte Knoten zusammenhängender Graph, der nur Knoten mit geradem Grad aufweist.

- 1 *Wähle einen beliebigen Knoten $v_0 \in V$.*

Wähle solange dies möglich ist Knoten v_1, v_2, \dots , so dass (v_0, \dots, v_i) jeweils ein Weg in G ist.

Unter den gegebenen Voraussetzungen entsteht so automatisch ein Kreis K . Setze $w' := v_0$.

- 2 *Prüfe, ob K ein eulerscher Kreis ist. Wenn ja, dann STOP, ansonsten gehe zu 3.*

Fortsetzung Algorithmus.

③ Setze $K' := K$.

Laufe ab w' entlang K' und wähle einen in K' enthaltenen Knoten w , der mit einer nicht in K' enthaltenen Kante inzident ist.

Konstruiere wie unter 1. ausgehend von w einen Kreis K'' , der keine Kanten von K' enthält.

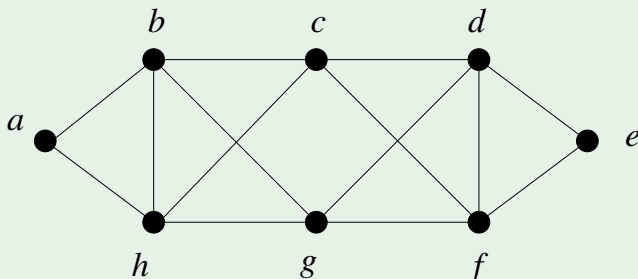
Füge K'' in den Kreis K' an der Stelle w ein. Setze $w' := w$ und $K := K'$.

Gehe zu 2.

Beispiel zum Hierholzer-Algorithmus

Beispiel 4.5

Wir demonstrieren den Algorithmus von Hierholzer an dem folgenden Graphen:



Tafel .

Eigenschaften des Algorithmus von Hierholzer

Satz 4.6

Algorithmus 4.4 ist korrekt, d.h. bei erfüllten Voraussetzungen wird ein eulerscher Kreis konstruiert.

Bemerkung: Wir können Algorithmus 4.4 auch zur Berechnung eines Eulerweges benutzen.

Satz 4.7

Die Zeitkomplexität von Algorithmus 4.4 beträgt $O(|V| + |E|)$.

Beweis.

Siehe Übungen.

Anwendung von Eulerwegen

- Dominospiel: Gegeben sind eine Menge S von Spielsteinen, die auf jeder Seite mit einem Symbol markiert sind.
- Einen Spielstein $[A : B]$ kann man sowohl in dieser Form als auch als $[B : A]$ verwenden.
- Man darf zwei Spielsteine aneinander legen, wenn die sich berührenden Hälften das gleiche Symbol aufweisen.
- Kann man die Steine einer Menge S zu einer ununterbrochenen Kette zusammenlegen?

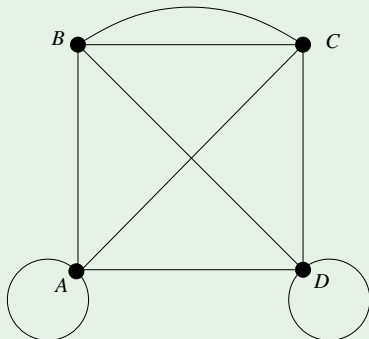
Beispiel 4.8

Gegeben ist die folgende Menge an Spielsteinen:

$$\begin{array}{ccccc}
 [A : B] & [A : D] & [B : C] & [C : D] & [A : A] \\
 [D : D] & [B : C] & [A : C] & [B : D] &
 \end{array}$$

Zugehöriger Graph:

- Jede Kante entspricht einem Spielstein.
- Eulerweg entspricht einer ununterbrochenen Dominokette.



Anwendung in der Praxis: Berechnung von Prozessketten mit möglichst wenigen Unterbrechungen.

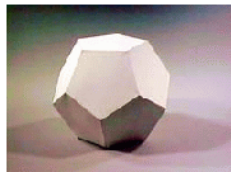
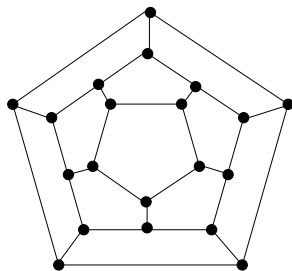
Hamiltonsche Graphen

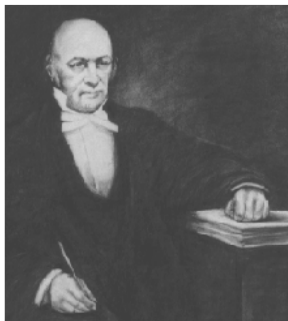
Definition 4.9

Es sei $G = (V, E)$ ein Graph.

- Ein Weg, der jeden Knoten von G genau einmal enthält, heißt **hamiltonscher Weg**.
- Ein Kreis, der jeden Knoten von G genau einmal enthält, heißt **hamiltonscher Kreis**.
- G heißt **hamiltonsch** gdw. G einen hamiltonschen Kreis enthält.

- Die Bezeichnung “hamiltonsch” geht auf [Sir William Rowan Hamilton \(1805 – 1865\)](#) zurück, der 1859 das Spiel “[around the world](#)” erfand.
- Die Punkte eines [Dodekaeders](#) stellen Städte dar.
- Die Aufgabe des Spiels bestand darin, entlang der Kanten des Dodekaeders eine Rundreise zu unternehmen, bei der man jede Stadt genau einmal besucht.





Sir William Rowan Hamilton



Around the World

Charakterisierung von hamiltonschen Graphen

Satz 4.10

Es sei $G = (V, E)$ ein Graph mit $n := |V| \geq 3$. Gilt:

$$\forall v \in V : \deg(v) \geq n/2,$$

dann ist G hamiltonsch.

Beweis.

Wir erzeugen den Graph G' aus G , indem wir

- k neue Knoten in G einfügen und
- jeden neuen Knoten mit allen Knoten von G über neue Kanten verbinden.

Wie viele solcher neuer Knoten brauchen wir mindestens, damit G' hamiltonsch wird?

Fortsetzung Beweis.

Mit $k = n$ gelingt es uns immer, G' hamiltonsch werden zu lassen.

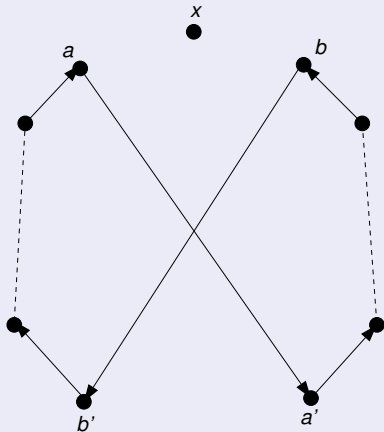
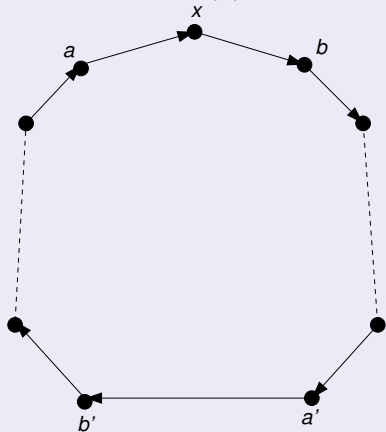
Wir wählen k so klein wie möglich ist und nehmen an, dass $k > 0$ gilt. Hieraus konstruieren wir einen Widerspruch.

Es sei $K = (a, x, b, \dots, a)$ ein hamiltonscher Kreis von G' , a, b seien Knoten von G , x sei ein neuer Knoten. Dann gilt:

- 1 a und b sind nicht benachbart, denn sonst wäre x für den hamiltonschen Kreis überflüssig.
- 2 In K kann ein zu b benachbarter Knoten b' nie direkt auf einen zu a benachbarten Knoten a' folgen.

Fortsetzung Beweis.

Beweisskizze zu (2):



Fortsetzung Beweis.

Wir definieren:

$A :=$ Menge der Nachbarn von a in G'

$B :=$ Menge der Nachbarn von b in G'

$F :=$ Menge der Knoten von G' , die in K direkt auf einen Knoten aus A f

Wegen (i) und (ii) gilt: $B \cap F = \emptyset$.

Weiterhin gilt: $|B| \geq \frac{n}{2} + k$ und $|F| = |A| = \frac{n}{2} + k$.

Konsequenz: $|B \cup F| = |B| + |F| \geq n + 2k$.

Da G' nur $n + k$ Knoten hat, ist dies ein Widerspruch für $k > 0$.

Bemerkung: Das Entscheidungsproblem, ob ein Graph G hamiltonsch ist (HC), ist \mathcal{NP} -vollständig.

Crashkurs \mathcal{NP} -Vollständigkeit

Die Klasse \mathcal{P} :

- Ein Entscheidungsproblem heißt **polynomiell lösbar**, wenn es einen Algorithmus zur Lösung des Problems gibt, dessen Worst-Case-Laufzeit durch ein Polynom in der Länge der Eingabe beschränkt ist.
- Die Klasse \mathcal{P} ist die Klasse der polynomiell lösbaren Probleme.
- Beispiele: Ist der Graph G zusammenhängend? Hat der Graph G einen eulerschen Kreis?

Die Klasse \mathcal{NP}

- Die Klasse \mathcal{NP} ist die Klasse der Entscheidungsprobleme, für die ein Lösungsvorschlag in polynomieller Rechenzeit überprüft werden kann.
- Beispiel: Überprüfung, ob eine Knotenfolge eines Graphen einen hamiltonschen Kreis bildet.
- Beispiel: Überprüfung, ob eine TSP-Tour eine Länge $\leq l$ hat.

Beziehung zwischen \mathcal{P} und \mathcal{NP}

Es gilt: $\mathcal{P} \subseteq \mathcal{NP}$

Gilt sogar $\mathcal{P} = \mathcal{NP}$?

Man weiß es nicht! Man vermutet, dass dies nicht der Fall ist. Stand heute kann man aber

weder $\mathcal{P} = \mathcal{NP}$ noch $\mathcal{P} \neq \mathcal{NP}$

beweisen.

- ☞ Die Lösung des $\mathcal{P} = \mathcal{NP}$ Problems gehört zu den wichtigsten offenen Problemen der Mathematik/Informatik.

\mathcal{NP} -Vollständigkeit

- Ein Problem Π aus \mathcal{NP} ist \mathcal{NP} -vollständig, wenn aus seiner polynomiellen Lösbarkeit $\mathcal{P} = \mathcal{NP}$ folgt, d.h. alle Probleme in \mathcal{NP} polynomiell lösbar wären.
- Die \mathcal{NP} -vollständigen Probleme können anschaulich als die “schwersten” Probleme in der Klasse \mathcal{NP} angesehen werden.

Wie kann man zeigen, dass ein Problem Π \mathcal{NP} -vollständig ist?

- Man nimmt sich ein Problem Π' , von dem man weiß, dass es \mathcal{NP} -vollständig ist und zeigt, dass Π “nicht leichter” ist.
- Genauer: Man reduziert Π' auf Π , d.h. man zeigt, dass man Π' in polynomieller Zeit lösen könnte, wenn man Π in polynomieller Zeit lösen kann.
- Schreibweise:

$$\Pi' \propto \Pi$$

Welche \mathcal{NP} -vollständigen Probleme kennt man denn?

Satz 4.11 (Satz von Cook (1971))

Das *Erfüllbarkeitsproblem der Aussagenlogik (SAT)* ist \mathcal{NP} -vollständig.

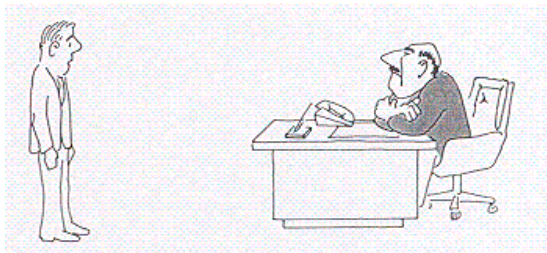
Das Erfüllbarkeitsproblem der Aussagenlogik fragt, ob eine aussagenlogische Formel erfüllbar ist.

Karps 21 \mathcal{NP} -vollständige Probleme [1972]: Richard Karp griff die Idee von Cook auf und zeigte für 21 verschiedene kombinatorische und graphentheoretische Probleme, dass sie \mathcal{NP} -vollständig sind, darunter auch das Hamiltonkreisproblem (HC).

Seitdem konnte für viele weitere Probleme gezeigt werden, dass sie \mathcal{NP} -vollständig sind.

Was bringt einem das in der Praxis?

Stellen Sie sich vor, Ihr Chef gibt Ihnen den Auftrag, für ein wichtiges Problem innerhalb eines Projekts einen effizienten Algorithmus zu programmieren, Ihnen gelingt die Konstruktion solch eines Algorithmus aber nicht.



I can't find an efficient algorithm, I guess I'm just too dumb.

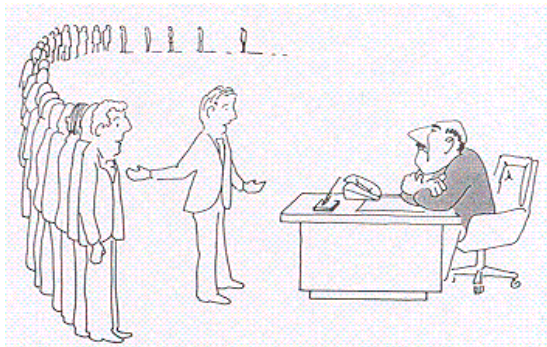
Sie wären natürlich aus dem Schneider, wenn Sie zeigen könnten, dass solch ein Algorithmus nicht existiert.



I can't find an efficient algorithm, because no such algorithm is possible.

Unglücklicherweise gelingt Ihnen dies aber auch nicht.

Wenn Sie aber nachweisen, dass das betreffende Problem \mathcal{NP} -vollständig ist, haben Sie gegenüber Ihrem Chef gute Argumente.



I can't find an efficient algorithm, but neither can all these famous people.

- Gelingt der Nachweis der \mathcal{NP} -Vollständigkeit, so zeigt dies, dass man nicht zu dumm ist, ein polynomielles Lösungsverfahren zu finden.
- Vermutlich existiert keines. Zumindest hat noch nie jemand ein solches gefunden und es macht keinen Sinn, nach einem solchen zu suchen.
- Man kann für “große” Probleme keine (optimalen) Lösungen erwarten.
- Anwendung heuristischer statt exakter Verfahren

Traveling Salesman Problem

Definition 4.12

Gegeben sei ein vollständiger Graph $G = (V, E)$ mit einer Kostenfunktion $c : E \rightarrow \mathbb{N}$ auf den Kanten.

Die **Entscheidungsvariante** des **Traveling Salesman Problems (TSP)** lautet:
Existiert ein hamiltonscher Kreis (eine TSP-Tour) K in G , für den die Summe der Kantengewichte $c(K) \leq k$ ist.

Bemerkungen:

- Die Entscheidungsvariante des TSP ist \mathcal{NP} -vollständig.
- Dies kann durch $HC \propto TSP$ bewiesen werden.

Skizze für $HC \propto TSP$

- Es sei $G = (V, E)$ ein Graph. Wir wollen entscheiden, ob G hamiltonsch ist.
- Es sei $G' = (V, E')$ ein vollständiger Graph für die Knotenmenge von G . Die Kostenfunktion $c : E' \rightarrow \mathbb{N}$ sei

$$c(e) = \begin{cases} 1 & \text{für } e \in E, \text{ also die Kanten aus } G \\ 2 & \text{sonst} \end{cases}$$

- Dann hat G genau dann einen hamiltonschen Kreis, wenn in G' eine TSP-Tour der Länge $|V|$ existiert.
- Konsequenz: Würde es einen polynomiellen Algorithmus für das TSP-Problem geben, wäre auch das HC-Problem polynomiell lösbar.

TSP als Optimierungsproblem

- Die **Optimierungsvariante** des TSP lautet: Finde einen hamiltonschen Kreis mit minimalem Gewicht.
- Durch die \mathcal{NP} -Vollständigkeit des zugehörigen Entscheidungsproblems kann es optimal nur für “kleine” n gelöst werden.
- Für “große” n müssen **Heuristiken** zur Berechnung einer möglichst guten Lösung angewendet werden.
- Theoretisch interessant sind Heuristiken mit Gütegarantie (siehe folgendes Kapitel).

Abstände in Graphen

Definition 4.13

Es sei $G = (V, E)$ ein Graph. Der **Abstand** $d(v, w)$ zweier Knoten $v, w \in V$ ist die minimale Länge eines Weges von v nach w . Falls es keinen solchen Weg gibt, setzen wir $d(v, w) = \infty$.

Bemerkung:

- Die Länge eines Kantenzugs bzw. Weges ist definiert als die Anzahl der Kanten, die in diesem enthalten sind.
- Insbesondere gilt also $d(v, v) = 0$ für alle $v \in V$.

Berechnung von Abständen

Algorithmus 4.14

Es sei $G = (V, E)$ ein durch seine Adjazenzlisten A_v gegebener Graph und v_0 sei ein Knoten von G . Es werden die Abstände $d(v)$ von v_0 zu v für alle $v \in V$ berechnet.

```
 $d(v_0) := 0; V(0) := \{v_0\}; k := 1;$   
for all  $v \in V, v \neq v_0$  do  $d(v) := \infty;$   
while  $V(k-1) \neq \emptyset$  do  
     $V(k) := \emptyset$   
    for all  $v \in \bigcup \{A_u \mid u \in V(k-1)\}$  do  
        if  $d(v) = \infty$  then  
             $d(v) := k; V(k) := V(k) \cup \{v\}$   
        end  
    end  
     $k := k + 1;$   
end
```


Erläuterungen zu Algorithmus 4.14

- Ausgehend von $V(0) = \{v_0\}$ werden sukzessive die Mengen $V(1), V(2), \dots$ berechnet.
- $V(k)$ besteht aus den Knoten, die mit Knoten aus $V(k-1)$ adjazent sind, aber nicht in $V(0) \cup \dots \cup V(k-1)$ liegen.
- Mit Induktion zeigt man leicht, dass jedes $V(k)$ genau aus den Knoten besteht, die von v_0 den Abstand k haben.
- Dieses Vorgehen entspricht einer Breitensuche.
- Die Zeitkomplexität beträgt $O(|V| + |E|)$.

Abstände in Netzwerken

Wir ordnen nun jeder Kante eines Graphen eine Länge zu. Typisches Beispiel sind Straßennetze mit Entfernungsangaben.

Definition 4.15

Es sei $G = (V, E)$ ein Graph sowie $w : E \rightarrow \mathbb{R}$ eine Bewertung der Kanten mit reellen Zahlen.

Das Paar (G, w) heißt **Netzwerk**. Für jede Kante $e \in E$ heißt $w(e)$ die **Länge** oder das **Gewicht** von e .

Für einen Kantenzug $K = (v_0, \dots, v_k)$ ist $w(K) := \sum_{i=1}^k w(\{v_{i-1}, v_i\})$ die **Länge** von K .

Abstände in Netzwerken (2)

- Bei den Längen können auch negative Werte sinnvoll sein (z.B. Ertrag, Aufwand).
- Bei der Definition des Abstandes können solche negativen Längen aber zu Problemen führen.
- Wir setzen daher voraus, dass die **Längen nicht negativ sind**.

Definition 4.16

Es sei (G, w) ein Netzwerk, $G = (V, E)$ sowie $w(e) \geq 0$ für alle $e \in E$.

Der **Abstand** $d(v, w)$ zweier Knoten $v, w \in V$ in einem Netzwerk ist definiert als das Minimum der Längen aller Wege von v nach w . Falls es keinen solchen Weg gibt, setzen wir $d(v, w) = \infty$.

Abstände in Netzwerken (3)

Beispiel 4.17

In einem Netzwerk sei jeder Kante e eine Ausfallwahrscheinlichkeit $p(e)$ zugeordnet.

Setzt man voraus, dass Fehler unabhängig voneinander auftreten, dann ist

$$(1 - p(\{v_0, v_1\})) \cdots (1 - p(\{v_{k-1}, v_k\}))$$

die Wahrscheinlichkeit für das Funktionieren eines Kantenzuges (v_0, \dots, v_k) .

Dieser Wert ist maximal, wenn $\sum_{i=1}^k \log(1 - p(\{v_{i-1}, v_i\}))$ maximal ist.

Setzt man $w(e) := -\log(1 - p(e))$, dann ist die zuverlässigste Verbindung zwischen zwei Knoten v und w äquivalent zu einem kürzesten Weg zwischen v und w .

Kürzeste Wege in Netzwerken

Algorithmus 4.18 (Dijkstra)

Es sei (G, w) ein Netzwerk mit $G = (V, E)$ und einer nichtnegativen Längenfunktion w auf E sowie $v_0 \in V$. Es werden alle Abstände $d(v)$ von v_0 zu einem Knoten $v \in V$ berechnet.

- 1 Setze $d(v_0) := 0$, $d(v) := \infty$ für alle $v \in V \setminus \{v_0\}$, $U := V$.
- 2 Falls $U = \emptyset$, dann STOP. Sonst weiter mit 3.
- 3 Finde ein $u \in U$, für das $d(u)$ minimal ist.
- 4 Für alle $v \in U$ mit $\{u, v\} \in E$ setze

$$d(v) := \min\{d(v), d(u) + w(\{u, v\})\}.$$

- 5 Setze $U := U \setminus \{u\}$. Gehe zu 2.

Eigenschaften des Dijkstra-Algorithmus

Bemerkungen:

- Der Algorithmus von Dijkstra kann analog für gerichtete Graphen verwendet werden.
- Der Algorithmus kann leicht so erweitert werden, dass nicht nur die Abstände $d(v_0, v)$ sondern auch die kürzesten Wege berechnet werden.

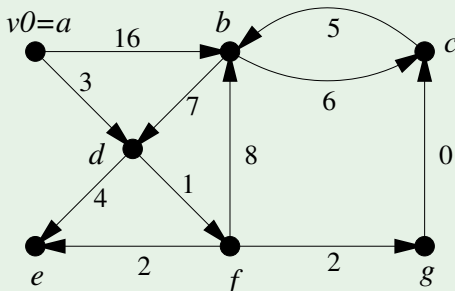
Satz 4.19

Nach der Terminierung von Algorithmus 4.18 gilt $d(v) = d(v_0, v)$ für alle $v \in V$.

Die Zeitkomplexität des Algorithmus ist $O(|V|^2)$.

Beispiel 4.20

Wir betrachten das folgende gerichtete Netzwerk:

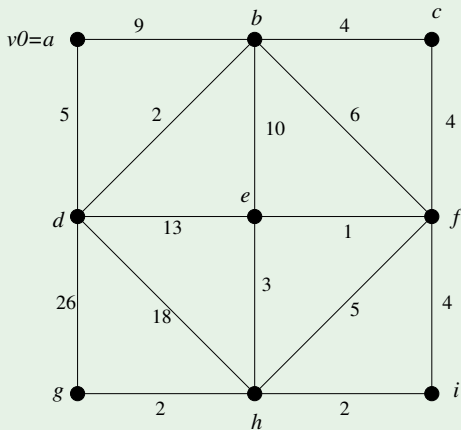


Algorithmus 4.18 liefert hierfür die folgenden Abstände:

| v | a | d | f | e | g | c | b |
|--------|-----|-----|-----|-----|-----|-----|-----|
| $d(v)$ | 0 | 3 | 4 | 6 | 6 | 6 | 11 |

Beispiel 4.21

Wir betrachten das folgende nicht gerichtete Netzwerk:



Bemerkungen zum Dijkstra-Algorithmus

- In einem Netzwerk mit negativen Kantenlängen werden durch den Dijkstra-Algorithmus u. U. falsche Abstände ermittelt
- und zwar selbst dann, wenn keine Kreise negativer Länge existieren.
- Durch die Wahl einer geeigneten Datenstruktur für die Selektion des Knotens u (Schritt 3) kann die Laufzeit erheblich verbessert werden.
- Für die Berechnung von Routen in Straßennetzen benötigt man optimierte Varianten des Dijkstra-Algorithmus.

Kürzeste Wege in DAGs

Da in DAGs keine Kreise auftreten, sind negative Kantenlängen unproblematisch.

Satz 4.22

Gegeben sei ein gerichtetes Netzwerk (G, w) mit $G = (V, A)$. Wir lassen auch $w(e) < 0$ zu. $L = (v_1, \dots, v_n)$ sei eine topologische Sortierung von V . Dann ergibt sich die Länge $d(v_1, v_i)$ eines kürzesten Weges von v_1 zu einem Knoten v_i durch die folgende Rekursion für $d(v_i)$:

① $d(v_1) = 0$

② Für $i > 1$:

$$d(v_i) = \min_{\{1 \leq j < i \mid (v_j, v_i) \in A\}} w(v_j, v_i) + d(v_j)$$


Beispiel 4.23

Tafel .

Kürzeste Wege in DAGs (2)

- Wenn man in Satz 4.22 min durch max ersetzt, dann erhält man die **Länge eines längsten Weges**.
- Das allgemeine Entscheidungsproblem, ob in einem (ungerichteten) Netzwerk ein (einfacher) Weg der Länge $\geq l$ existiert, ist dagegen \mathcal{NP} -vollständig.
- Die in Satz 4.22 angewendete Vorgehensweise, eine optimale Lösung durch eine bestmögliche Kombination der Lösungen zu Subproblemen zu finden, nennt man **dynamisches Programmieren**.
- Satz 4.22 läßt sich in einen Algorithmus mit Zeitkomplexität $O(|V| + |A|)$ umsetzen.

Beweis.

Mit Induktion über i . Tafel .

Kürzeste Wege in DAGs (3)

- Mit dem Verfahren aus Satz 4.22 können nicht nur kürzeste und längste Wege **sondern auch Anzahlen von Wegen** berechnet werden.
- Z.B. die Anzahl $\#(v_i)$ der **Wege von v_1 nach v_i** .
- Hierfür muss der Minimum-Operator durch die **Summenbildung**

$$\#(v_i) = \sum_{\{1 \leq j < i \mid (v_j, v_i) \in A\}} \#(v_j)$$

ersetzt und die **Initialisierung $\#(v_1) = 1$** verwendet werden.

Anwendungen

Für optimale Wege in DAGs gibt es eine Fülle von Anwendungen. Wir betrachten zwei:

- Optimale Einteilung in Blöcke (z.B. in der Textverarbeitung)
- Projektplanung

Blocksatzbildung

- Gegeben ist eine Folge $F = (1, \dots, n)$ von n Gegenständen.
- Jeder Gegenstand hat eine Länge $l(i)$.
- Die Gegenstände sollen in Blöcke eingeteilt werden, wobei die Idealgröße eines Blockes B ist.
- Die Gesamtlänge der Gegenstände in einem Block darf B nicht übersteigen.
- Die Gegenstände dürfen nicht umsortiert werden. Ein Block B_j ergibt sich also durch einen Index i_j mit
 - ▶ $i_{j-1} + 1$ ist das erste Element im Block j und
 - ▶ i_j ist das letzte Element im Block B_j .
- Für die Abweichung der Länge

$$L_j := \sum_{r=i_{j-1}+1}^{i_j} l(r)$$

eines Blockes B_j von B wird eine Bewertung definiert, die monoton in der Größe der Abweichung ist, z.B. $(B - L_j)^2$.

- Gesucht ist eine Blockung, die eine möglichst kleine Bewertung hat, also eine Anzahl k an Blöcken und eine Folge $0 = i_0 < i_1 < i_2 < \dots < i_k = n$ mit

$$\sum_{j=1}^k (B - L_j)^2 \longrightarrow \min$$

unter den Bedingungen

$$L_j = \sum_{r=i_{j-1}+1}^{i_j} l(r) \leq B \text{ für } j = 1, \dots, k$$

Beispiel 4.24

- Länge der Gegenstände: 9, 1, 5, 5, 1, 10
- Blockgröße: 10
- “gierige” Einteilung: [9, 1], [5, 5], [1], [10], Bewertung: 81
- andere Einteilung: [9], [1, 5], [5, 1], [10], Bewertung: 33

Modellierung des Problems als Wegeproblem

- $V := \{0, \dots, n\}$
 - $E := \{(i, j) \mid i < j \text{ und } \sum_{r=i+1}^j l(r) \leq B\}$
 - Gewichte: $w(i, j) := (B - \sum_{r=i+1}^j l(r))^2$
 - Jeder Weg von 0 nach n definiert dann eine zulässige Einteilung in Blöcke, wobei die Bewertung dieser Blockung gleich der Länge des Weges ist.
 - Also: Finde einen Weg minimaler Länge von 0 nach n .
- ☞ Auf diesem Prinzip basiert die Formatierung in TeX.

Projektplanung mit Netzplantechnik

- Bei der Durchführung umfangreicher Projekte ist es erforderlich, dass einzelne Teilaufgaben (Jobs) zeitlich genau aufeinander abgestimmt werden.
- Hierfür werden häufig Methoden der Netzplantechnik eingesetzt.
- Die bekannteste Methode der Netzplantechnik heißt CPM (Critical Path Method).
- J sei die Menge der Jobs eines Projekts. Jeder Job $j \in J$ hat eine zugeordnete Dauer $t(j)$ sowie eine Menge $P(j) \subseteq J$ von Vorgängern.
- Ein Job j kann erst dann gestartet werden, wenn alle Jobs aus $P(j)$ beendet wurden.
- Wir gehen davon aus, dass Jobs parallel bearbeitet werden können.
- Man möchte z.B. wissen:
 - ▶ Ab welchem Zeitpunkt kann mit einem Job begonnen werden?
 - ▶ Wie lange wird ein Projekt dauern?
 - ▶ Welche Jobs sind besonders kritisch in bezug auf die Gesamtdauer eines Projekts?

Beispielprojekt

Beispiel 4.25

Zusammenbau eines Fahrrads:

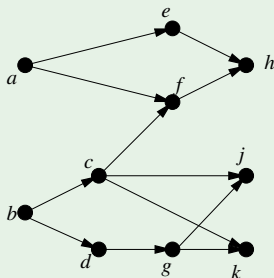
| Job | Beschreibung | Dauer | Vorg. |
|-----|--|-------|-------|
| a | Rahmen vorbereiten (Gabel, Schutzbleche, usw.) | 6 | – |
| b | Kettenführung anbringen | 2 | – |
| c | Gangschaltung anbringen | 3 | b |
| d | Kettenblatt an Kurbel montieren | 4 | b |
| e | Vorderrad montieren und anpassen | 6 | a |
| f | Hinterrad montieren und anpassen | 6 | a,c |
| g | Kurbel am Rahmen anbringen | 3 | d |
| h | Endmontage (Lenker, Sattel, usw.) | 12 | e,f |
| j | Linkes Pedal anbringen | 3 | c,g |
| k | Rechtes Pedal anbringen | 3 | c,g |

Modellierung

- Die Abhängigkeiten der Jobs untereinander können wir in Form eines DAGs repräsentieren.
- Die Knoten stellen hierbei die Jobs dar.
- Die Kanten stellen Vorgängerbeziehungen zwischen den Jobs dar.

Beispiel 4.26

Zusammenbau eines Fahrrads:



Fortsetzung Beispiel.

- Die Jobs a und b können offensichtlich sofort begonnen werden.
- Der Job f kann begonnen werden, wenn a beendet ist und wenn c beendet ist.
- a ist nach einer Dauer von 6 beendet, c nach einer Dauer von 5, da c erst gestartet werden kann, wenn b beendet ist.
- Somit kann f frühestens zum Zeitpunkt 6 begonnen werden.
- Allgemein: Der **frühe Starttermin** eines Jobs ergibt sich durch einen Weg zu diesem Job mit maximaler Gesamtdauer.
- Hier liegen die Bewertungen aber auf den Knoten.

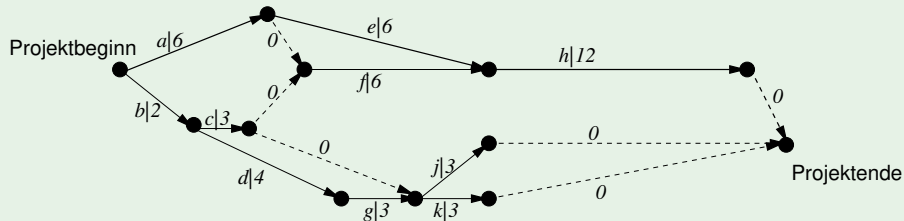
Critical-Path-Method

Die **Critical-Path-Method (CPM)** stellt eine Modellierung dar, bei der die Jobs die Kanten eines DAGs sind.

Die Knoten können als Fertigstellungsereignisse angesehen werden.

Die Kanten a, e, h stellen einen längsten Weg vom Projektbeginn zum -ende dar (**kritische Pfad**).

Beispiel 4.27



Die Gesamtlaufzeit des Beispielprojekts ist also 24.

Projektplanung mit CPM

Für die Projektplanung sind u.a. die folgenden interessant:

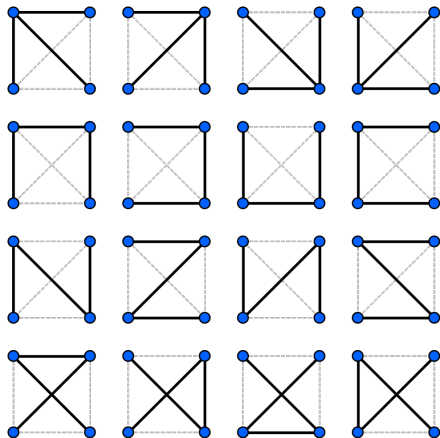
- **Projektdauer D** : Länge eines längsten Weges (kritischer Pfad) vom Projektbeginn bis zum Projektende.
- **Starttermin für Jobs j** : Länge eines längsten Weges bis zum Startknoten von j .
- **Zeitkritische Jobs**: Alle Jobs, die auf einem kritischen Pfad liegen.
- **Pufferzeit für Job j** : D -Länge eines längsten Weges, der Kante j enthält.
- Und warum verzögern sich Projekte trotz ausgefeilter Planungsmethoden? Siehe Übungen.

Zusammenfassung des Kapitels

- Charakterisierung und Berechnung von Eulerwegen (Hierholzer-Algorithmus)
- Hamiltonsche Wege und Kreise (Berechnung ist schwer)
- Dijkstra-Algorithmus zur Ermittlung kürzester Wege
- Dynamische Programmierung zur Berechnung kürzester Wege in DAGs
- Anwendungen: Blocksatzbildung und Netzplantechnik

Kapitel 5

Bäume und Minimalgerüste



Inhalt

5 Bäume und Minimalgerüste

- Charakterisierung von Minimalgerüsten
- Berechnung von Minimalgerüsten
- Minimalgerüste und TSP
- Anwendung von Minimalgerüsten

Gerüst

Definition 5.1

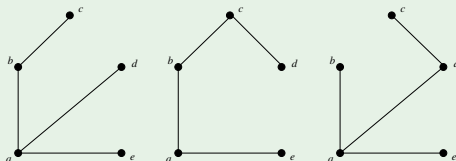
Es ein $G = (V, E)$ ein zusammenhängender Graph.

$H = (V, E')$ heißt **Gerüst** von G gdw. wenn H ein Baum ist und $E' \subseteq E$ gilt.

Ein Gerüst ist also ein **zusammenhängender, kreisfreier, aufspannender Untergraph** von G .

Beispiel 5.2

Einige Gerüste für das Haus vom Nikolaus:



Minimalgerüst

Definition 5.3

Es sei $G = (V, E)$ ein zusammenhängender Graph mit einer Kantengewichtsfunktion $w : E \rightarrow \mathbb{R}$.

- Für $F \subseteq E$ heißt

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

das **Gewicht der Kantenmenge** F .

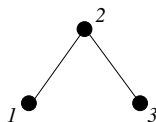
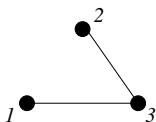
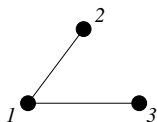
- Es sei $H = (V, F)$ ein Gerüst von G . Dann ist $w(H) := w(F)$ das **Gewicht des Gerüsts** H .
- Ein Gerüst H von G heißt **Minimalgerüst** von G gdw. $w(H) \leq w(H')$ für alle Gerüste H' von G .

Anzahl an Gerüsten

Satz 5.4 (Cayley)

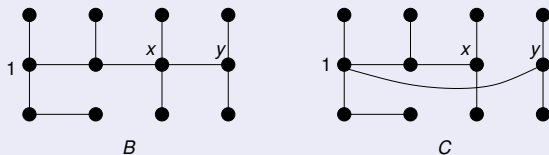
Es gibt n^{n-2} verschiedene Gerüste für den vollständigen Graphen K_n (mit n Knoten).

Bemerkung: Verschieden bedeutet hier wirklich verschieden und nicht "nichtisomorph".



Beweis.

- Idee: **Prinzip der doppelten Abzählung**: In jeder Matrix ist die Summe der Zeilensummen gleich der Summe der Spaltensummen.
- Sei $t(n, k)$ die Anzahl der Bäume auf $V = \{1, \dots, n\}$, in denen der Knoten 1 den Grad k hat.
- Es wird eine Formel für $t(n, k)$ bestimmt. Das Ergebnis erfolgt durch Aufsummieren über alle k .
- B sei ein Baum mit $\deg(1) = k - 1$, C sei ein Baum mit $\deg(1) = k$. (B, C) heißt **verwandtes Paar** gdw. C aus B entsteht, indem eine Kante $\{x, y\}$ entfernt und eine Kante $\{1, y\}$ eingefügt wird.



Fortsetzung Beweis.

- Es seien $B_1, B_2, \dots, B_{t(n, k-1)}$ die Bäume mit $\deg(1) = k - 1$ und $C_1, C_2, \dots, C_{t(n, k)}$ die Bäume mit $\deg(1) = k$.
- Die $t(n, k - 1) \times t(n, k)$ Matrix $A = (a_{ij})$ sei definiert durch:

$$a_{ij} := \begin{cases} 1 & \text{falls } (B_i, C_j) \text{ verwandtes Paar} \\ 0 & \text{sonst} \end{cases}$$

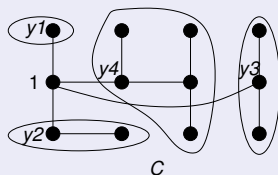
- Aus B_i kann jede der $n - 1$ Kanten entfernt werden, ausgenommen die $k - 1$ mit 1 inzidenten Kanten.

In der i -ten Zeile von A stehen so viele Einsen, wie es verwandte Paare (B_i, C) gibt, also genau $(n - 1) - (k - 1) = n - k$.

Summe der Zeilensummen von $A = t(n, k - 1)(n - k)$.

Fortsetzung Beweis.

- In der j -ten Spalte von A stehen so viele Einsen, wie es verwandte Paare (B, C_j) gibt.
- Die in C_j mit 1 verbundenen Knoten seien y_1, \dots, y_k .



- Entfernt man eine der Kanten $\{1, y_r\} (1 \leq r \leq k)$, so entsteht ein Graph mit zwei ZHKs. V_r sei die ZHK, die y_r enthält und $n_r := |V_r|$.
- Ein verwandtes Paar (B, C_j) entsteht genau dann, wenn $B = (C_j \setminus \{\{1, y_r\}\}) \cup \{\{x, y_r\}\}$ gilt, wobei x einer der $n - 1 - n_r$ Knoten in $V \setminus (\{1\} \cup V_r)$ ist.

Fortsetzung Beweis.

- Es gibt also genau $(n - 1 - n_1) + \dots + (n - 1 - n_k) = (k - 1)(n - 1)$ Einsen in der j -ten Spalte von A .

Summe der Spaltensummen von $A = t(n, k)(k - 1)(n - 1)$.

- Das Prinzip der doppelten Abzählung liefert

$$t(n, k - 1)(n - k) = t(n, k)(k - 1)(n - 1)$$

bzw.

$$t(n, k - 1) = (n - 1) \frac{k - 1}{n - k} t(n, k).$$

- Ausgehend von $t(n, n - 1) = 1$ ergibt sich durch Induktion

$$t(n, n - i) = (n - 1)^{i-1} \binom{n - 2}{i - 1}.$$

Fortsetzung Beweis.

- Für die Anzahl $t(n)$ aller Bäume ergibt sich somit

$$\begin{aligned}t(n) &= \sum_{i=1}^{n-1} t(n, n-i) \\&= \sum_{i=1}^{n-1} (n-1)^{i-1} \binom{n-2}{i-1} \\&= \sum_{i=0}^{n-2} (n-1)^i \binom{n-2}{i} \\&= ((n-1) + 1)^{n-2} = n^{n-2}.\end{aligned}$$

Berechnung von Minimalgerüsten

Lemma 5.5

Es sei $G = (V, E)$ ein zusammenhängender Graph mit einer Kantengewichtsfunktion $w : E \rightarrow \mathbb{R}$. Weiterhin sei $U \subseteq V$ und e_0 eine Kante zwischen U und $V \setminus U$ mit minimalem Gewicht.

Dann existiert ein Minimalgerüst für G , das die Kante e_0 enthält.

Beweis.

Falls ein Minimalgerüst T_0 die Kante e_0 nicht enthält, so nehmen wir e_0 zu T_0 und entfernen eine Kante e_1 , die U und $V \setminus U$ verbindet.

Wegen der Minimalitätseigenschaft von e_0 erhöhen wir damit nicht das Gewicht des Gerüstes.

Der Algorithmus von Prim (1)

- Wir beginnen mit einem beliebigen Knoten v , d.h. $U := \{v\}$.
- In einem Iterationsschritt berechnen wir für alle $v \in V \setminus U$ den Knoten w , der am nächsten zu einem Knoten in U liegt.
- Dieser Knoten w wird selektiert, die entsprechende Kante wird in den Baum aufgenommen und w wird in U aufgenommen.
- Dies setzen wir fort, bis alle Knoten in U sind.

Der Algorithmus von Prim (2)

Algorithmus 5.6 (Prim)

Es sei $G = (V, E)$ ein zusammenhängender Graph mit $V = \{1, \dots, n\}$ und Kantengewichtsfunktion w . Es gelte $w(\{i, j\}) = \infty$, falls $\{i, j\} \notin E$. Der Algorithmus berechnet ein Minimalgerüst in B .

```

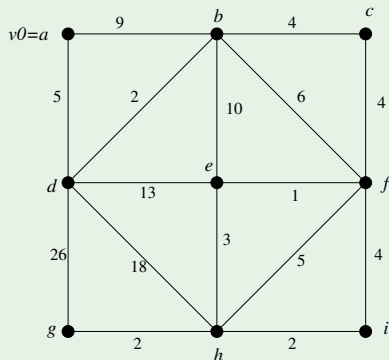
U := {1};
for i := 2 to n do  $d_U(i) := w(\{1, i\})$ ;
while  $U \neq V$  do
  bestimme Knoten  $u$  mit  $d_U(u) = \min\{d_U(v) : v \in V \setminus U\}$ ;      (*)
  bestimme Kante  $e := \{x, u\}$  mit  $w(e) = d_U(u)$  und  $x \in U$ ;      (*)
   $B := B \cup \{e\}$ ;
   $U := U \cup \{u\}$ ;
  for all  $v \in V \setminus U$  do  $d_U(v) := \min\{d_U(v), w(\{u, v\})\}$ ;      (**)
end

```

Der Algorithmus von Prim (3)

Beispiel 5.7

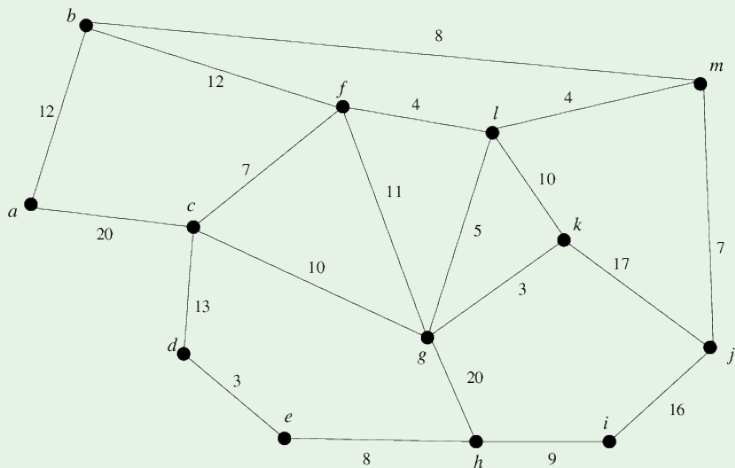
Wir betrachten den Graphen:



Es ergibt sich in dieser Reihenfolge:

$$B = \{\{a, d\}, \{b, d\}, \{b, c\}, \{c, f\}, \{e, f\}, \{e, h\}, \{g, h\}, \{h, i\}\}.$$

Beispiel 5.8



Satz 5.9

Algorithmus 5.6 berechnet ein Minimalgerüst in Zeit $O(|V|^2)$.

Beweis.

Es sind zwei Dinge zu beweisen:

- **Korrektheit des Algorithmus**

Wir zeigen induktiv: Nach jeder Ausführung von (*) (bzw. nach (**)) gibt $d_U(v)$ für alle $v \in V \setminus U$ den Abstand zu einem nächstgelegenen Knoten $u \in U$ an.

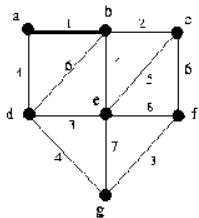
Korrektheit des Algorithmus folgt dann mit Lemma 4.2. Tafel .

- **Laufzeit**

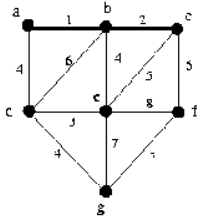
- ▶ Die dominieren Schritte innerhalb der While-Schleife: (*) und (**).
- ▶ Die While-Schleife wird $|V|$ -mal durchlaufen.
- ▶ Schritte (*) und (**) können in $O(|V|)$ ausgeführt werden.

Algorithmus von Kruskal (1)

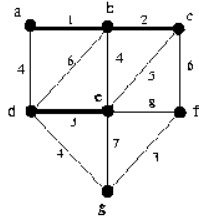
- Beim Algorithmus von Prim baut man ausgehend von einem Knoten den Baum sukzessive auf.
- Beim Algorithmus von Kruskal beginnt man stattdessen mit den einzelnen Knoten. Diese stellen einen Wald dar.
- Man versucht nun, diese Wälder optimal zu einem Baum zusammenzusetzen.
- Hierzu sortiert man zunächst alle Kanten aufsteigend nach ihrer Länge.
- Man beginnt, indem man die kürzeste Kante in den Wald aufnimmt. Der Wald hat jetzt eine ZHK weniger.
- In Iteration i : Falls die i -te Kante zu einem Kreis führen würde, verwirft man sie. Andernfalls nimmt man sie in den Wald auf, wodurch sich wiederum die Anzahl der ZHKs verringert.
- Dies macht man solange, bis ein aufspannender Baum entstanden ist.
- Dies ist genau dann der Fall, wenn man $n - 1$ Kanten aufgenommen hat (vgl. Satz 1.34 und Satz 1.42).



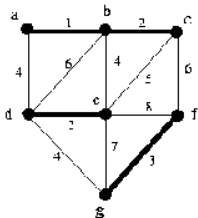
(a)



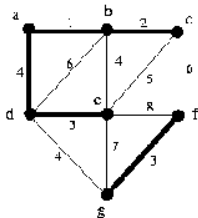
(b)



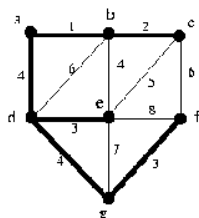
(c)



(d)



(e)



(f)

Algorithmus von Kruskal (2)

Algorithmus 5.10 (Kruskal)

Vorraussetzungen, Ein- und Ausgabe wie beim Algorithmus von Prim.

$B := \emptyset; ZHK := \emptyset; i := 0;$

$L :=$ Liste der Kanten aufsteigend sortiert nach ihrer Länge;

for all $v \in V$ *do* $ZHK := ZHK \cup \{\{v\}\};$

while $|ZHK| > 1$ *do*

$i := i + 1;$

Es sei $\{v, w\}$ *das* i -te Element von L ;

if v *und* w *gehören zu verschiedenen Komponenten* K_1 *und* K_2 *in* ZHK

$ZHK := (ZHK \setminus \{K_1, K_2\}) \cup \{\{K_1 \cup K_2\}\};$

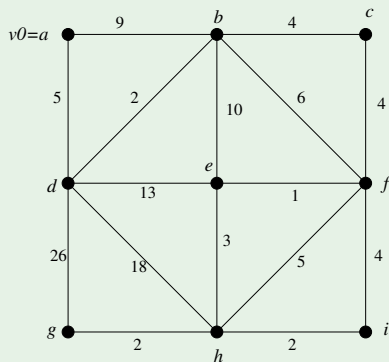
$B := B \cup \{v, w\};$

end

end

Beispiel 5.11

Wir betrachten den Graphen:



Es ergibt sich in dieser Reihenfolge:


$$B = \{\{e, f\}, \{b, d\}, \{g, h\}, \{h, i\}, \{e, h\}, \{b, c\}, \{c, f\}, \{a, d\}\}.$$

Algorithmus von Kruskal (3)

Satz 5.12

Algorithmus 5.10 berechnet ein Minimalgerüst in Zeit $O(|E| \log |V|)$.

Beweis.

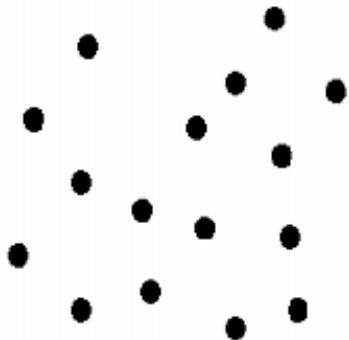
Die Korrektheit des Algorithmus von Kruskal kann induktiv durch wiederholte Anwendung von Lemma 5.5 gezeigt werden. Tafel .

Zeitaufwand: Der dominierende Schritt ist die Sortierung. Der Aufwand zur Sortierung der Kanten ist

$$O(|E| \log |E|) = O(|E| \log |V|^2) = O(|E| 2 \log |V|) = O(|E| \log |V|).$$

Für die effiziente Vereinigung der ZHKs und den vorangehenden Test benutzt man Datenstrukturen für das sogenannte **Union-Find-Problem**.

Minimalgerüste und TSP



Untere Schranke für TSP

Satz 5.13

Es sei $G = (V, E)$ ein vollständiger Graph mit Gewichtsfunktion $w : E \rightarrow \mathbb{N}$. Weiterhin sei TSP eine optimale TSP-Tour und MST sei ein Minimalgerüst.

Dann gilt:

$$w(MST) \leq w(TSP)$$

Beweis.

- Wenn man aus TSP eine beliebige Kante entfernt, erhält man einen hamiltonschen Weg HP .
- Jeder hamiltonsche Weg ist ein Gerüst.
- Also folgt $w(TSP) \geq w(HP) \geq w(MST)$.

Obere Schranke für TSP

Satz 5.14

Es sei $G = (V, E)$ ein vollständiger Graph mit Gewichtsfunktion $w : E \rightarrow \mathbb{N}$, für die die Dreiecksungleichung gilt, d.h.


$$w(\{i, k\}) \leq w(\{i, j\}) + w(\{j, k\}) \quad \forall i, j, k \in V$$

Weiterhin sei TSP eine optimale TSP-Tour und MST sei ein Minimalgerüst. Dann gilt:

$$w(MST) \leq w(TSP) \leq 2 w(MST)$$

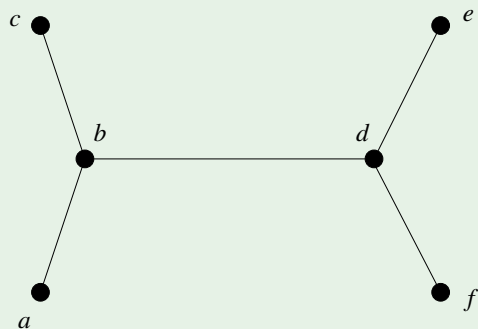
Beweis.

Die linke Ungleichung folgt aus Satz 5.13.

Für den Beweis der rechten Ungleichung konstruieren wir mit Hilfe der Tiefensuche eine Tour aus dem Minimalgerüst MST . Tafel .

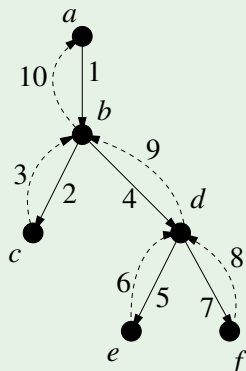
TSP-Tour mittels Tiefensuche auf MST (1)

Beispiel 5.15



| v | a | b | c | d | e | f |
|--------|-----|-----|-----|-----|-----|-----|
| $t(v)$ | 1 | 2 | 3 | 4 | 5 | 6 |

Tour: a, b, c, d, e, f, a

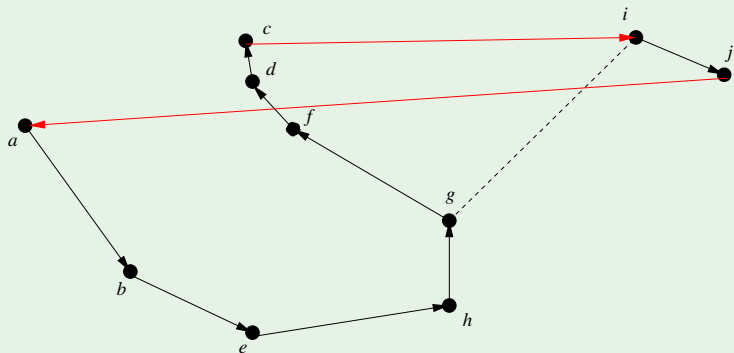


TSP-Tour mittels Tiefensuche auf MST (2)

- Das Verfahren aus Satz 5.14 liefert uns nicht nur eine **obere Schranke** für die Länge einer optimalen Tour,
- sondern auch ein effizientes Konstruktionsverfahren für eine Tour mit Länge $\leq 2 w(TSP)$.
- Die Tour ergibt sich dabei direkt aus der DFS-Numerierung.
- Alternative:
 - ▶ Jede Kante des *MST* wird durch zwei gerichtete Kanten ersetzt,
 - ▶ mit dem Algorithmus von Hierholzer wird ein Eulerkreis berechnet und
 - ▶ die Knoten werden in der Reihenfolge wie im Eulerkreis “angefahren”.

Verbesserung einer Tour (1)

Beispiel 5.16

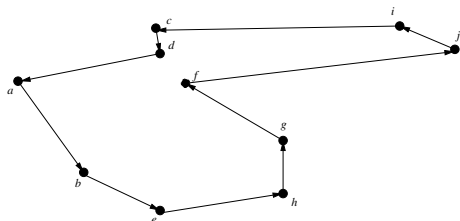
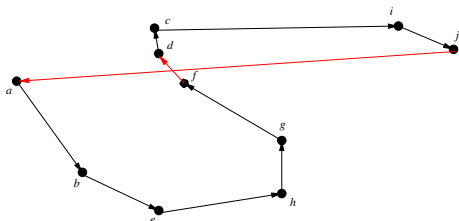


| v | a | b | c | d | e | f | g | h | i | j |
|--------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| $t(v)$ | 1 | 2 | 8 | 7 | 3 | 6 | 5 | 4 | 9 | 10 |

Tour: $a, b, e, h, g, f, d, c, i, j, a$

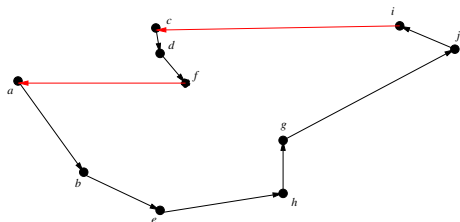
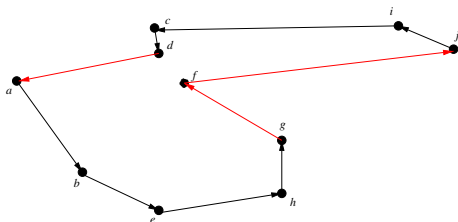
Verbesserung einer Tour (2)

- Anschließend können sogenannte **Verbesserungsverfahren** angewendet werden.
- Der Algorithmus **2-opt** sucht beispielsweise nach zwei Kanten, die gegen zwei andere Kanten ausgetauscht werden können, so dass eine kürzere Tour entsteht.
- Im euklidischen Fall werden damit alle **Kreuzungen** eliminiert.

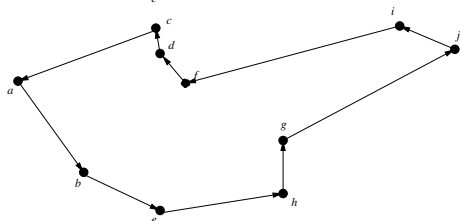


Verbesserung einer Tour (3)

- Analog sucht der Algorithmus **3-opt** nach drei Kanten für einen Austausch.



- Der nächste 2-opt-Tausch liefert eine optimale Tour.

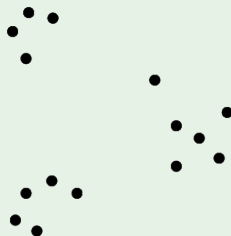


Clusteranalyse mit Minimalgerüsten (1)

Beispiel 5.17 (Clusteranalyse)

Gegeben sei eine Menge von Punkten, z. B. im \mathbb{R}^2 , die gewisse Häufungen aufweist.

Wie kann man diese **Häufungen** algorithmisch erkennen?



Clusteranalyse mit Minimalgerüsten (2)

- In Häufungen liegen Knoten nahe zusammen.
- Ansatz: Wir legen zwischen zwei Knoten eine Kante, wenn sie nicht zu weit voneinander entfernt sind ($\leq \alpha$).

$$E := \{\{v, w\} \mid d(v, w) \leq \alpha\}$$

- Hierbei ist $d(v, w)$ die Entfernung auf Basis einer Metrik $d(\cdot, \cdot)$ und α ist die maximal erlaubte Entfernung.
- Die Zusammenhangskomponenten des entsprechenden Graphen sehen wir dann als Häufungen an.
- Zur Berechnung dieser können wir den Algorithmus von Kruskal verwenden.

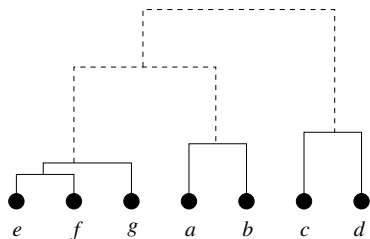
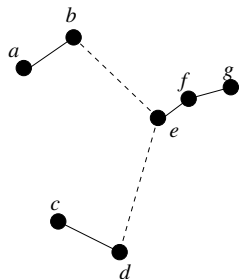
Clusteranalyse mit Minimalgerüsten (3)

- Berechnung eines Minimalgerüstes mit dem Algorithmus von Kruskal und
- Elimination der Kanten mit Länge $> \alpha$.



Clusteranalyse mit Minimalgerüsten (4)

- Die Benutzung des Algorithmus von Kruskal hat den Vorteil, dass die Kanten aufsteigend nach Ihrer Länge selektiert werden.
- Die Aufnahme einer Kante in den MST können wir als die Verschmelzung zweier Cluster ansehen.
- Der Verlauf der Cluster-Verschmelzung wird mit einem sogenannten **Dendrogramm** visualisiert.

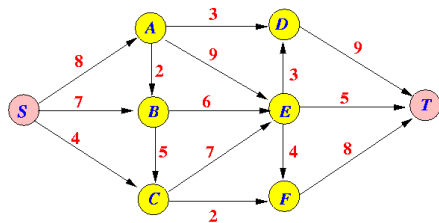


Zusammenfassung des Kapitels

- Berechnung von Minimalgerüsten
- Algorithmen: Prim, Kruskal
- Anwendungen:
 - ▶ Gütegarantie beim Traveling Salesman Problem
 - ▶ Heuristik zur Konstruktion einer TSP-Tour
 - ▶ Clusteranalyse

Kapitel 6

Flüsse und Zuordnungen



Inhalt

6 Flüsse und Zuordnungen

- Flussnetzwerke
- Berechnung maximaler Flüsse
- Max-Flow-Min-Cut
- Matchings

Flussprobleme

- In diesem Kapitel werden Bewertungen von Kanten als **Transportkapazitäten** pro Zeiteinheit interpretiert.
- Wir können uns einen Graphen als **Versorgungsnetzwerk** vorstellen, z.B. als Datennetz.
- Fragen:
 - ▶ Welchen **Durchsatz** können wir erreichen?
 - ▶ Wie viele Einheiten können wir von einem Knoten zu einem anderen pro Zeiteinheit transportieren?
 - ▶ Welche Kanten bilden dabei einen **Engpass**?

Flussnetzwerk

Definition 6.1

Ein **Flussnetzwerk** N ist ein Tupel $N = (G, c, s, t)$ bestehend aus:

- $G = (V, A)$, einem **gerichteten Graphen**,
- $c : A \rightarrow \mathbb{R}_+$, einer **Kapazitätsfunktion** auf den gerichteten Kanten mit nichtnegativen Werten und
- $s, t \in V$, zwei ausgezeichneten Knoten, der **Quelle** s und der **Senke** t mit $s \neq t$.

Fluss

Definition 6.2

Es sei $N = (G, c, s, t)$ ein Flussnetzwerk. Für einen Knoten $v \in V$ sei $A_{in}(v) := \{(u, v) \in A\}$ und $A_{out}(v) := \{(v, u) \in A\}$.

Eine Abbildung $f : A \rightarrow \mathbb{R}$ heißt **Fluss** auf N , wenn die folgenden Bedingungen erfüllt sind:

① $0 \leq f(e) \leq c(e)$ für alle $e \in A$,

d. h. die **Kapazität wird für keine Kante überschritten** und

②
$$\sum_{e \in A_{in}(v)} f(e) = \sum_{e \in A_{out}(v)} f(e) \text{ für alle } v \in V \setminus \{s, t\},$$

d. h. aus **jedem Knoten fließt genausoviel heraus wie hinein**, mit Ausnahme der Quelle s und der Senke t .

Flusswert

Lemma 6.3

Für einen Fluss f eines Flussnetzwerks $N = (G, c, s, t)$ gilt

$$\Phi(f) := \sum_{e \in A_{out}(s)} f(e) - \sum_{e \in A_{in}(s)} f(e) = \sum_{e \in A_{in}(t)} f(e) - \sum_{e \in A_{out}(t)} f(e)$$

Definition 6.4

Der Wert $\Phi(f)$ aus Lemma 6.3 heißt **Wert** des Flusses f auf N .

Ein Fluss f mit $\Phi(f) \geq \Phi(f')$ für alle Flüsse f' auf N heißt **Maximalfluss** auf N .

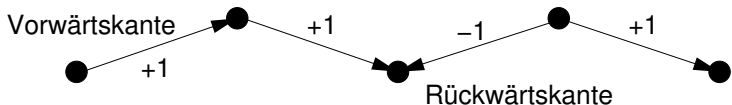
Das **Maximalflussproblem** besteht darin, zu einem gegebenen Flussnetzwerk N einen Maximalfluss zu bestimmen.

Zunehmender Weg

Definition 6.5

Gegeben sei ein Flussnetzwerk $N = (G, c, s, t)$ mit einem Fluss f . Eine Folge (v_0, \dots, v_k) heißt **zunehmender Weg** bzgl. f gdw. für jedes $i = 1, \dots, k$ eine der folgenden Bedingungen erfüllt ist:

- ① $(v_{i-1}, v_i) \in A$ und $f(v_{i-1}, v_i) < c(v_{i-1}, v_i)$ (Vorwärtskante)
- ② $(v_i, v_{i-1}) \in A$ und $f(v_i, v_{i-1}) > 0$ (Rückwärtskante)



Kriterium für Maximalfluss

- Offensichtlich können wir den Fluss erhöhen, wenn wir einen zunehmenden Weg gefunden haben.
- Die Existenz eines zunehmenden Weges ist also **hinreichend** für eine Flusserhöhung.
- Der folgende Satz zeigt, dass dieses Kriterium auch **notwendig** ist.

Satz 6.6

Ein Fluss f in einem Flussnetzwerk N ist genau dann ein Maximalfluss, wenn kein zunehmender Weg von s nach t existiert.

Beweis.

“ \Rightarrow ”: Wenn ein zunehmender Weg W von s nach t existiert, dann kann $\Phi(f)$ um das Minimum der Werte $c(e) - f(e)$ für Vorwärtskanten von W bzw. $f(e)$ für Rückwärtskanten von W erhöht werden.

“ \Leftarrow ”: Es gebe keinen zunehmenden Weg von s nach t .

- Es sei S die Menge der Knoten, die von s aus mit einem zunehmenden Weg erreichbar sind, und sei $T := V \setminus S$.
- Für jede Kante (v, w) , $v \in S$, $w \in T$ gilt: $f(v, w) = c(v, w)$
- Für jede Kante (w, v) , $w \in T$, $v \in S$ gilt: $f(w, v) = 0$
- Anschaulich: Die Kanten zwischen S und T bilden einen Engpass, der eine Flusserhöhung verhindert.

Berechnung eines Maximalflusses

Satz 6.6 liefert die Basis zur Berechnung eines Maximalflusses.

- ① Wir starten mit einem beliebigen Fluss, z.B. $f(e) = 0$ für alle $e \in A$.
- ② Wenn es keinen zunehmenden Weg bzgl. f gibt, dann STOP.
- ③ Sei $W = (s = v_0, v_1, \dots, v_k = t)$ ein zunehmender Weg von s nach t bzgl. f und sei

$$z := \min(\{c(v_{i-1}, v_i) - f(v_{i-1}, v_i) \mid (v_{i-1}, v_i) \text{ Vorwärtskante von } W\} \\ \cup \{f(v_i, v_{i-1}) \mid (v_i, v_{i-1}) \text{ Rückwärtskante von } W\}).$$

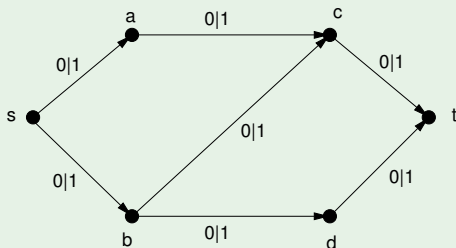
Setze $f(v_{i-1}, v_i) := f(v_{i-1}, v_i) + z$ für jede Vorwärtskante (v_{i-1}, v_i) .

Setze $f(v_i, v_{i-1}) := f(v_i, v_{i-1}) - z$ für jede Rückwärtskante (v_i, v_{i-1}) .

Weiter mit 2.

Beispiel 6.7

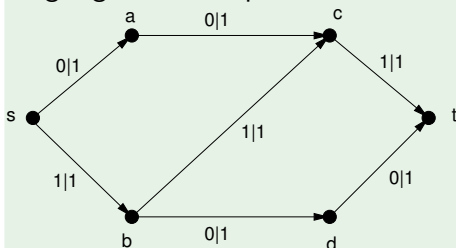
Wir betrachten das folgende Flussnetzwerk. Die Kapazität ist für alle Kanten 1. Der Fluss ist zunächst auf allen Kanten 0.



(s, b, c, t) ist ein zunehmender Weg mit $z = 1$. Alle Kanten des Weges sind Vorwärtskanten.

Fortsetzung Beispiel.

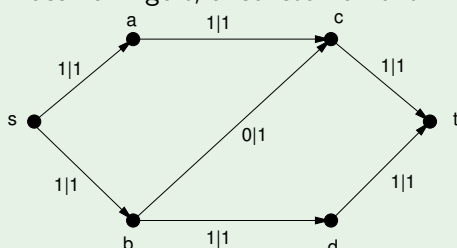
Flusserhöhung auf dem zunehmenden Weg ergibt den Graphen:



mit $\Phi(f) = 1$.

(s, a, c, b, d, t) ist nun ein zunehmender Weg mit $z = 1$, wobei (b, c) eine Rückwärtskante ist.

Auf der Rückwärtskante wird der Fluss verringert, ansonsten erhöht.



Wir haben $\Phi(f) = 2$ und es existiert kein zunehmender Weg. Damit ist der angegebene Fluss ein Maximalfluss.

Markierungsalgorithmus

- Der **Markierungsalgorithmus** von Ford und Fulkerson (1956) konkretisiert das Verfahren zur Berechnung maximaler Flüsse.
- Man markiert sukzessive die Knoten w auf einem zunehmenden Weg mit drei Werten $v(w)$, $r(w)$, $z(w)$.
 - ▶ $v(w)$ ist der **Vorgänger von w** in dem zunehmenden Weg.
 - ▶ $r(w)$ gibt die **Richtung der verwendeten Kante** an
(\rightarrow = Vorwärtskante, \leftarrow = Rückwärtskante).
 - ▶ $z(w)$ ist der mögliche **zusätzliche Fluss** auf dem Weg nach w .

Algorithmus 6.8

Gegeben sei ein Flussnetzwerk $N = (G, c, s, t)$ und ein initialer Fluss $f(e) \equiv 0$.

- 1 Setze $S := \{s\}, R := \{s\}, z(s) := \infty$.
- 2 Wähle einen Knoten $u \in R$. Setze $R := R \setminus \{u\}$.
- 3 Für alle $w \in V \setminus S$ mit $(u, w) \in A$ und $f(u, w) < c(u, w)$:

$$S := S \cup \{w\}, R := R \cup \{w\},$$

$$v(w) := u, r(w) := \rightarrow, z(w) := \min\{z(u), c(u, w) - f(u, w)\}$$

- 4 Für alle $w \in V \setminus S$ mit $(w, u) \in A$ und $f(w, u) > 0$:

$$S := S \cup \{w\}, R := R \cup \{w\},$$

$$v(w) := u, r(w) := \leftarrow, z(w) := \min\{z(u), f(w, u)\}$$

Fortsetzung Algorithmus.

- 5 Falls $R = \emptyset$, dann STOP. Falls $t \in S$, dann weiter mit 6, ansonsten weiter mit 2.
- 6 $z := z(t)$; $w := t$.
- 7 Falls $r(w) = \rightarrow$: $u := v(w)$, $f(u, w) := f(u, w) + z$
Falls $r(w) = \leftarrow$: $u := v(w)$, $f(w, u) := f(w, u) - z$
- 8 $w := v(w)$. Falls $w = s$, dann weiter mit 1, ansonsten weiter mit 7.

Satz 6.9

Sei $N = (G, c, s, t)$ ein Flussnetzwerk mit rationaler Kapazitätsfunktion c . Dann berechnet Algorithmus 6.8 einen maximalen Fluss f auf N .

- Bei irrationalen Kapazitäten kann es vorkommen, dass der Markierungsalgorithmus immer weitere Verbesserungen des Flusswertes findet, ohne jemals zu terminieren.
- Auch bei ganzzahligen Kapazitäten ist die Laufzeit des Markierungsalgorithmus nicht polynomial, da die Anzahl der Schritte von c abhängen kann.
- Eine polynomiale Laufzeit erhält man aber, wenn man für die Suche nach einem zunehmenden Weg die **Breitensuche** einsetzt (Edmonds und Karp, 1972).

Algorithmus von Edmonds und Karp


Satz 6.10

Ersetzt man in Algorithmus 6.8 den Schritt 2 durch

- 2a. *Wähle den Knoten $u \in R$, der zuerst in R eingefügt wurde. Setze $R := R \setminus \{u\}$.*

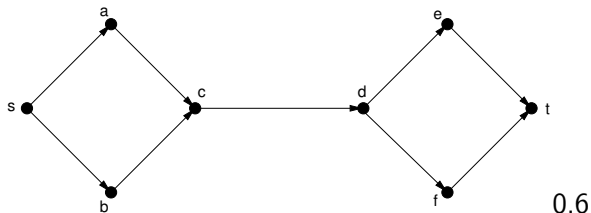
dann berechnet der Markierungsalgorithmus für beliebige Kapazitätsfunktionen in Zeit $O(|V||E|^2)$ einen Maximalfluss.

Beispiel 6.11

Anwendung des Markierungsalgorithmus. Tafel 

Trennender Schnitt

Wie groß kann der Fluss in dem folgenden Flussnetzwerk höchstens sein?



Der Fluss kann nicht größer als die Kapazität der Kante (c, d) sein, da jeder Weg von s nach t diese Kante enthält.

Die Kante (c, d) ist ein sogenannter **trennender Schnitt**.

Minimaler Schnitt

Definition 6.12

Es sei $N = (G, c, s, t)$ ein Flussnetzwerk mit $G = (V, A)$.

Für eine Teilmenge $S \subseteq V$ heißt $A_S := \{(v, w) \in A \mid v \in S, w \in V \setminus S\}$ **Schnitt** von G .

Falls $s \in S, t \in V \setminus S$, so ist A_S ein **trennender Schnitt**.

Ein trennender Schnitt A_S mit minimaler Kapazität

$$c(A_S) := \sum_{e \in A_S} c(e)$$

heißt **minimaler Schnitt**.

Max-Flow-Min-Cut-Theorem

Satz 6.13

In einem Flussnetzwerk $N = (G, c, s, t)$ ist der Wert eines maximalen Flusses gleich der Kapazität eines minimalen Schnittes.

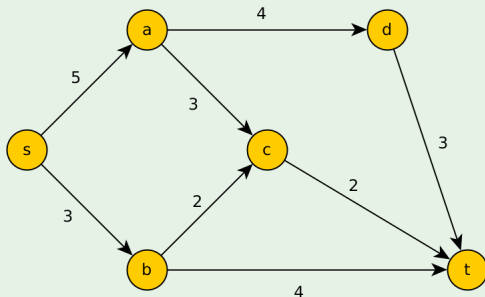
Beweis.

Tafel .

- Kennen wir einen Fluss f und finden wir einen trennenden Schnitt A_S mit $\Phi(f) = c(A_S)$, so ist f ein Maximalfluss und A_S ein minimaler Schnitt.
- Für die Menge S bei Terminierung von Algorithmus 6.8 ist A_S ein minimaler Schnitt (vgl. Beweis zu Satz 6.6).
- Der Markierungsalgorithmus berechnet also nicht nur einen maximalen Fluss sondern auch einen minimalen Schnitt.

Beispiel 6.14

Wie lautet ein Maximalfluss für den folgenden Graphen?

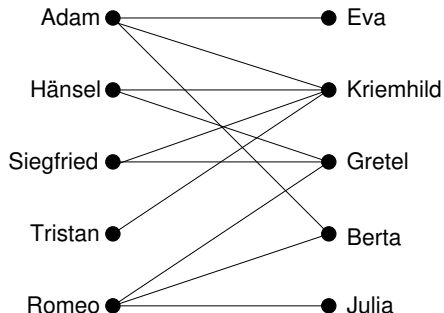


$$\begin{aligned}
 f(s, b) &= f(b, t) = 3, \\
 f(s, a) &= 5, f(a, c) = f(c, t) = 2, \\
 f(a, d) &= f(d, t) = 3 \text{ also } \Phi(f) = 8.
 \end{aligned}$$

Die Kapazität des trennenden Schnittes $\{(s, b), (c, t), (d, t)\}$ ist 8.
Also ist f ein Maximalfluss.

Heiratsproblem

Das **Heiratsproblem** lautet: In einer Gruppe von Männern und Frauen kennen sich einige Männer und Frauen. Ist es möglich, daß jeder mit einer seiner Bekannten verheiratet wird?



Matching

Definition 6.15

Es sei $G = (V, E)$ ein bipartiter Graph.

Ein Menge $M \subseteq E$ von Kanten heißt **Zuordnung (Matching)** gdw.

für alle $e_1, e_2 \in M$ mit $e_1 \neq e_2$ gilt : $e_1 \cap e_2 = \emptyset$,

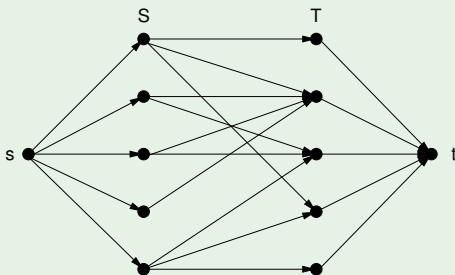
d. h. die Kanten haben keinen gemeinsamen Knoten.

Heiratsproblem als Flussproblem modellieren

- Das Heiratsproblem ist genau dann lösbar, wenn ein Matching maximaler Mächtigkeit existiert.
- Das wesentliche Problem besteht also darin, zu einem bipartiten Graphen ein Matching mit maximaler Mächtigkeit zu berechnen. Hierzu können wir das Problem als Flussproblem modellieren.
- Alle Kanten des Originalgraphen werden von S nach T gerichtet.
- Neuer Knoten s und neue Kanten (s, v) für alle $v \in S$
- Neuer Knoten t und neue Kanten (v, t) für alle $v \in T$
- Alle Kanten erhalten die Kapazität 1.

Beispiel 6.16

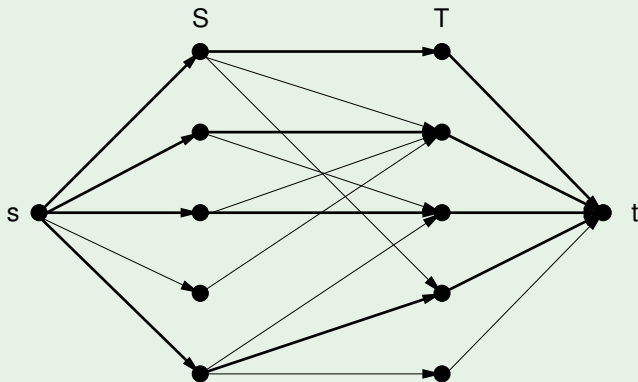
Transformiert man den bipartiten Graphen des gegebenen Heiratsproblem nach obigen Regeln ergibt sich:



- Es sei f ein Maximalfluss, der mit dem Markierungsalgorithmus berechnet wurde. Wegen der Ganzzahligkeit des Problems ist dann auch f ganzzahlig.
- In einen Knoten $v \in T$ kann dann nicht mehr als 1 hineinfließen.
- Aus einem Knoten $v \in S$ kann nicht mehr als 1 herausfließen.
- Daraus folgt, dass die Kanten zwischen S und T mit Fluss 1 ein Matching bilden.
- Die Anzahl dieser Kanten ist gleich dem Flusswert $\Phi(f)$.
- Damit stellen diese Kanten ein Matching maximaler Mächtigkeit dar.

Beispiel 6.17

Für unser spezielles Heiratsproblem erhalten wir als eine Lösung:



Es ist also nicht möglich, alle zu verheiraten. Tristan und Julia bleiben bei dieser Lösung unverheiratet.

Das gewichtete Zuordnungsproblem

Das **gewichtete Zuordnungsproblem** lautet:

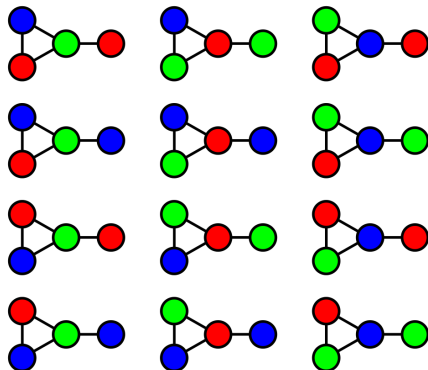
- Gegeben sei ein vollständiger bipartiter Graph, d.h. für alle $s \in S$ und $t \in T$ gibt es eine Kante, die s und t verbindet.
- Auf den Kanten ist eine Gewichtsfunktion definiert.
- Man finde ein Matching maximaler Mächtigkeit mit minimalem Gesamtgewicht.
- Beispiel: Man verbindet im Heiratsproblem alle Männer mit allen Frauen und drückt den Grad der Zuneigung durch ein Antipathiegewicht aus.
- Es sind dann alle zu verheiraten und zwar so, dass die Gesamtantipathie möglichst klein und somit die Gesamtzuneigung möglichst groß ist.
- Zur Lösung von gewichteten Zuordnungsproblemen wendet man ähnliche Verfahren wie bei der Berechnung von Maximalflüssen und wie beim ungewichteten Zuordnungsproblem an.
- Ein sehr bekanntes Verfahren ist der **ungarische Algorithmus**.

Zusammenfassung

- Flussnetzwerk und Maximalfluss
- Zunehmender Weg als Grundlage zur Berechnung von maximalen Flüssen
- Berechnung maximaler Flüsse mit dem Markierungsalgorithmus
- $\max \text{ flow} = \min \text{ cut}$
- Anwendung des Maximalflussproblems zur Berechnung maximaler Matchings

Kapitel 7

Planare Graphen und Färbungen



Inhalt

7 Planare Graphen und Färbungen

- Planarität
- Färbungen

Jordankurve

Zentrale Frage bei der **Planarität**: Welche Graphen lassen sich **zeichnen, ohne dass sich die Kanten schneiden**?

Die Kanten stellen wir dabei als **Jordankurve** dar.

Definition 7.1

Eine **Jordankurve** des \mathbb{R}^n ist eine Menge der Form $\{f(t) \mid t \in [0, 1]\}$, wobei $f : [0, 1] \rightarrow \mathbb{R}^n$ eine injektive stetige Abbildung ist.

Bemerkung: Jordankurven sind stetige und schnittpunktfreie Kurven mit Anfangs- und Endpunkt.

Jordankurven im \mathbb{R}^2 (1)

Beispiel 7.2

Im \mathbb{R}^2 ist eine Funktion f zur Darstellung einer Kurve gegeben durch

$$f(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$$

mit $x : [0, 1] \rightarrow \mathbb{R}$ und $y : [0, 1] \rightarrow \mathbb{R}$.

Die durch f definierte Kurve ist **stetig**, wenn die Funktionen $x(t)$ und $y(t)$ stetig sind.

Jordankurven im \mathbb{R}^2 (1)

Beispiel 7.3

Eine **Strecke** von einem Punkt (x_1, y_1) zu einem Punkt (x_2, y_2) ist gegeben durch

$$f(t) = \begin{pmatrix} x_1 + t(x_2 - x_1) \\ y_1 + t(y_2 - y_1) \end{pmatrix}.$$

Der **Einheitskreis** entspricht der Kurve mit

$$f(t) = \begin{pmatrix} \cos(2\pi t) \\ \sin(2\pi t) \end{pmatrix}.$$

Einbettbarkeit

Definition 7.4

Ein Graph $G = (V, E)$ heißt **einbettbar** in den \mathbb{R}^n gdw.

- seine **Knoten als paarweise verschiedene Punkte des \mathbb{R}^n** und
- seine **Kanten als Jordankurven des \mathbb{R}^n** dargestellt werden können,
- wobei die Jordankurven als Anfangs- bzw. Endpunkt die Punkte der zugeordneten Knoten haben und
- sich die **Kurven paarweise nicht schneiden** (außer in den Knotenpunkten für die Anfangs- und Endpunkte).

Planarität

Definition 7.5

Ein Graph $G = (V, E)$ heißt **planar** gdw. er in den \mathbb{R}^2 einbettbar ist.

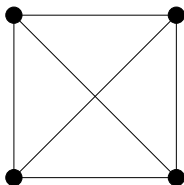
Das Diagramm einer solchen Einbettung heißt **ebenes Diagramm**.

Jedes Ebene Diagramm eines Graphen unterteilt die Ebene in zusammenhängende Gebiete, die man **Flächen** nennt.

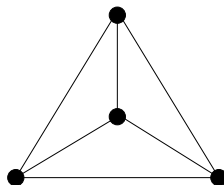
Es existiert genau ein solches Gebiet, das unbeschränkt ist. Es heißt **Außengebiet**.

Diagramme für den K_4

K_4 :



nicht eben



eben

Weitere Beispiele für planare Graphen:

- **Hypercube** für $n = 2, 3$
- **Dodekaedergraph**

Einfache Aussagen zur Planarität

Lemma 7.6

Der K_5 ist nicht planar.

Beweis.

Nach dem Zeichnen des Kreises $(1, 2, 3, 4, 5, 1)$ haben wir genau zwei Flächen: das Außengebiet und die Fläche, die durch den Kreis eingeschlossen wird.

Es fehlen noch die folgenden fünf Kanten:

$\{1, 3\}$, $\{1, 4\}$, $\{2, 4\}$, $\{2, 5\}$, $\{3, 5\}$. Diese können wir **entweder in die Kreisfläche oder das Außengebiet** legen.

Bei **jeder dreielementigen Teilmenge** der fehlenden fünf Kanten **kommt es zu einer Kreuzung**.

Fortsetzung Beweis.

Damit können kreuzungsfrei höchstens zwei Kanten in die Kreisfläche und höchstens zwei Kanten in das Außengebiet gelegt werden.

In der Summe können also höchstens neun Kanten kreuzungsfrei gezeichnet werden, der K_5 hat aber zehn Kanten.

Lemma 7.7

- *Jeder Untergraph eines planaren Graphen ist wieder planar.*
- *Jeder Graph, der einen nicht planaren Untergraphen enthält, ist selbst nicht planar.*

Bipartite Graphen

Definition 7.8

Ein Graph $G = (V, E)$ heißt **bipartit** gdw. V in zwei disjunkte Teilmengen S und T zerlegt werden kann (d.h. $V = S + T$), so dass für alle Kanten $e = \{v, w\} \in E$ gilt: $v \in S, w \in T$.

Ein bipartiter Graph, bei dem jeder Knoten aus S mit jedem Knoten aus T adjazent ist, heißt **vollständig bipartit**.

Mit $K_{n,m}$ wird der vollständige bipartite Graph mit $|S| = n$ und $|T| = m$ bezeichnet.

Lemma 7.9

Der $K_{3,3}$ ist nicht planar.

Beweis.

Analog zu Lemma 7.6.


Eulersche Polyederformel

Satz 7.10

Gegeben sei ein planarer zusammenhängender Graph $G = (V, E)$ mit $n := |V|$ und $m := |E|$ sowie ein ebenes Diagramm von G mit f Flächen. Dann gilt

$$n + f = m + 2$$

Beweis.

Beweis durch Induktion über m . Für $m = 0$ folgt $n = f = 1$. Damit ist die Formel richtig. Für den Induktionsschritt Fallunterscheidung ob G Baum oder nicht. Tafel .

Folgerung 7.11

Für ein ebenes Diagramm eines planaren Graphen $G = (V, E)$ mit k Zusammenhangskomponenten gilt

$$n + f = m + k + 1$$

Knoten- und Kantenanzahlen in planaren Graphen

Satz 7.12

*Es sei $G = (V, E)$ ein planarer Graph mit $n := |V|$ und $m := |E| \geq 2$.
Dann gilt:*

$$m \leq 3n - 6.$$

Wenn G keinen Kreis der Länge 3 enthält, dann gilt:

$$m \leq 2n - 4.$$

Beweis.

Gegeben sei ein ebenes Diagramm eines zusammenhängenden Graphen mit n Knoten, $m \geq 3$ Kanten e_1, \dots, e_m und f Flächen F_1, \dots, F_f .

Die $m \times f$ -Matrix $A = (a_{ij})$ sei definiert durch

$$a_{ij} := \begin{cases} 1 & \text{falls } e_i \text{ Teil des Randes von } F_j \\ 0 & \text{sonst} \end{cases}$$

- Jede Kante gehört zum Rand von höchstens zwei Flächen. Also enthält jede Zeile der Matrix höchstens zwei Einsen, d.h. die Summe der Zeilensummen ist $\leq 2m$.
- Jede Fläche wird von mindestens drei Kanten berandet. Somit gibt es insgesamt mindestens $3f$ Einsen (Summe der Spaltensummen).

Fortsetzung Beweis.

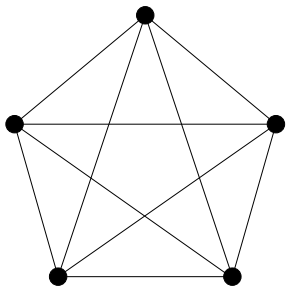
- Es folgt $3f \leq 2m$. Polyederformel: $f = m - n + 2$
- Damit folgt $m \leq 3n - 6$.
- Die Formel gilt auch für $m = 2$.
- Für nicht zusammenhängende Graphen: Formel gilt in jeder ZHK.

Bemerkung:

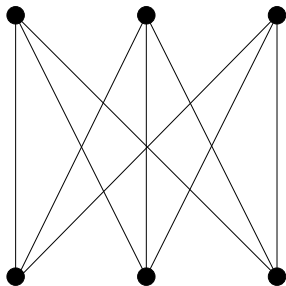
- Satz 7.12 liefert eine andere Begründung dafür, dass der K_5 und der $K_{3,3}$ nicht planar sind.
- Für diese Graphen ist die 1. bzw. 2. Ungleichung in Satz 7.12 nicht erfüllt.

Die kleinsten nicht planaren Graphen

K5



K3,3



Bemerkung: Diese beiden Graphen sind in gewissem Sinne die “kleinsten” nicht planaren Graphen.

Folgerung 7.13

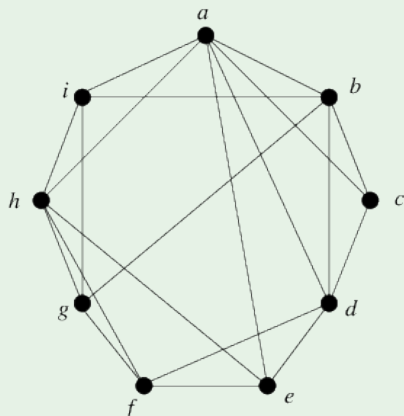
Für jeden planaren Graph $G = (V, E)$ existiert ein $v \in V$ mit $\deg(v) \leq 5$.

Testen auf Planarität

- effizient, aber komplex
- hier: Beispiel für eine heuristische Methode für **hamiltonsche Graphen**

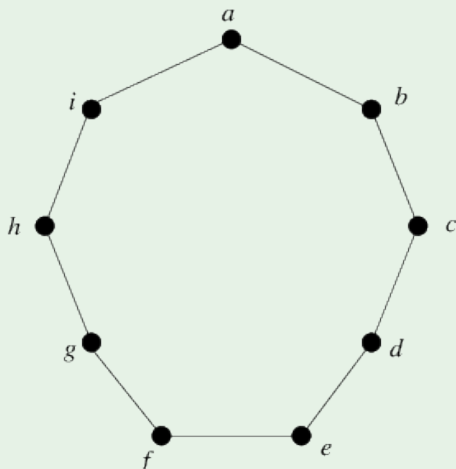
Beispiel 7.14

Ist dieser Graph planar?



Fortsetzung Beispiel.

Der Graph enthält einen hamiltonschen Kreis. Dieser muss auf alle Fälle kreuzungsfrei gezeichnet werden, also fangen wir damit an.



Fortsetzung Beispiel.

Die fehlenden Kanten können wir in die Kreisfläche oder das Außengebiet legen. Wir listen alle fehlenden Kanten auf:

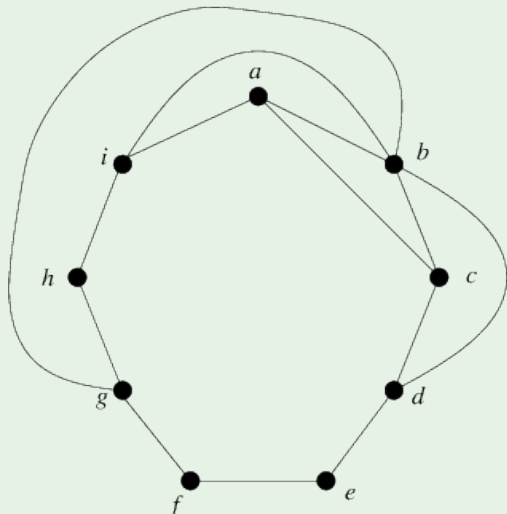
$$\begin{array}{l} \{a, c\} \quad \{b, d\} \quad \{d, f\} \quad \{e, h\} \quad \{f, h\} \quad \{g, i\} \\ \{a, d\} \quad \{b, g\} \\ \{a, e\} \quad \{b, i\} \\ \{a, h\} \end{array}$$

Es sei

- A die Menge der noch fehlenden Kanten, die wir in die Kreisfläche legen und
- B sei die Menge der fehlenden Kanten, die wir ins Außengebiet legen.

Fortsetzung Beispiel.

Wir nehmen die Kante $\{a, c\}$ in die Menge A auf. Diese Kante steht in Konflikt mit den Kanten $\{b, d\}$, $\{b, g\}$, $\{b, i\}$, also nehmen wir diese in B auf. Diese Kanten sind untereinander konfliktfrei.



Jetzt noch fehlende Kanten:

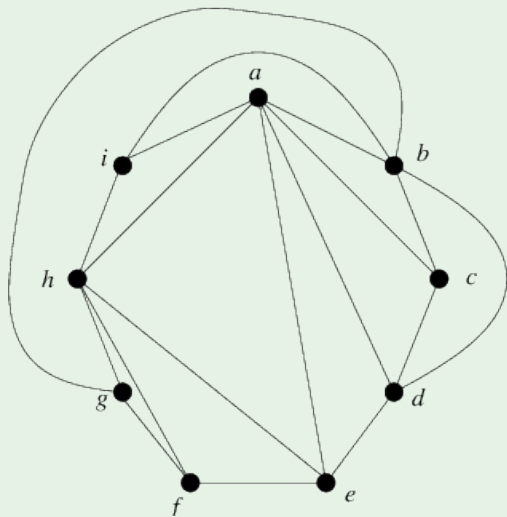
$\{a, d\}$, $\{a, e\}$, $\{a, h\}$, $\{d, f\}$, $\{e, h\}$, $\{f, h\}$, $\{g, i\}$

Fortsetzung Beispiel.

Von den Kanten in B steht $\{b, g\}$ in Konflikt mit den Kanten $\{a, d\}$, $\{a, e\}$, $\{e, h\}$, $\{f, h\}$. Also können wir diese Kanten nur in die Menge A einfügen.

Weiterhin steht $\{b, i\}$ in Konflikt mit $\{a, h\}$, also muss auch diese Kante nach A . Alle jetzt in A befindlichen Kanten sind konfliktfrei.

Jetzt noch fehlende Kanten: $\{d, f\}$, $\{g, i\}$

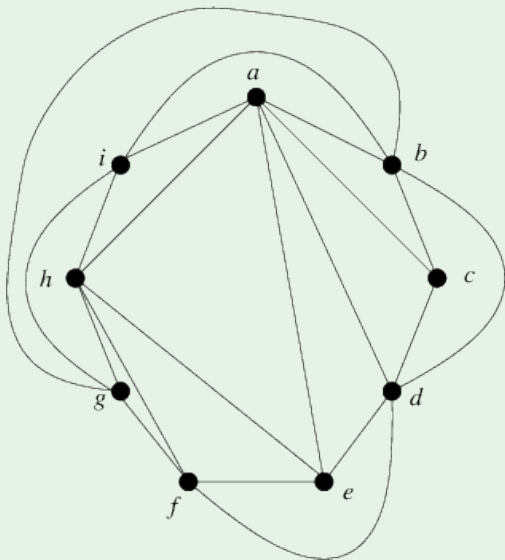


Fortsetzung Beispiel.

Von den Kanten in A steht $\{a, e\}$ in Konflikt mit $\{d, f\}$. Also kommt $\{d, f\}$ nach B .

Weiterhin steht $\{e, h\}$ in Konflikt mit $\{g, i\}$, also nehmen wir auch $\{g, i\}$ in B auf. Alle Kanten in B sind untereinander konfliktfrei.

Damit ist der Graph **planar**.

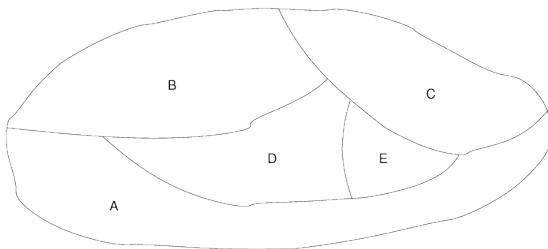


Vier-Farben-Vermutung (1)

Landkarten möchte man so färben, dass **keine benachbarten Länder die gleiche Farbe erhalten**.

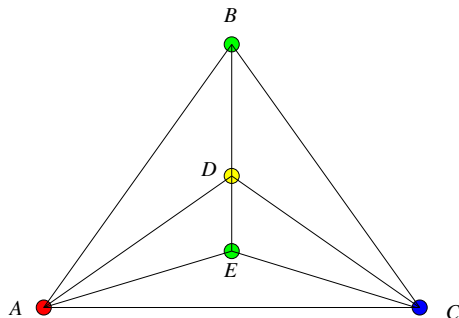
Wie viele Farben braucht man zur Färbung einer Landkarte?

Vier-Farben-Vermutung: Jede Landkarte kann so mit **vier Farben** gefärbt werden, dass **benachbarte Länder stets verschiedene Farben haben**.



Vier-Farben-Vermutung (2)

- Eine Landkarte kann als planarer Graph dargestellt werden.
- Hierzu werden die **Länder durch Knoten repräsentiert** und **benachbarte Länder werden durch eine Kante miteinander verbunden**.
- Die **Knoten** zu einer Kante müssen dann **verschiedene Farben** haben.



Färbungen (1)

Definition 7.15

Ein Graph $G = (V, E)$ heißt **k -färbbar** gdw.

$$\exists f : V \rightarrow \{1, \dots, k\} \text{ mit } \{v, w\} \in E \Rightarrow f(v) \neq f(w),$$

d. h. den Knoten können k Farben zugeordnet werden, so dass adjazente Knoten stets verschiedene Farben haben.

Die Zahl

$$\chi(G) := \min\{k \mid G \text{ ist } k \text{ färbbar}\}$$

heißt **chromatische Zahl** von G .

Färbungen (2)

Bemerkungen:

- Die chromatische Zahl $\chi(G)$ ist die kleinste Anzahl an Farben, die für eine Färbung von G benötigt werden.
- Besteht G aus den ZHKs G_1, \dots, G_n , dann gilt

$$\chi(G) = \max_{i=1}^n \chi(G_i).$$

- Für eine anschauliche Präsentation verwende ich üblicherweise Farben statt natürlicher Zahlen.

Einfache Aussagen zu Färbungen (1)

Lemma 7.16

Ein Graph G ist genau dann 1-färbbar, wenn G nur aus isolierten Knoten besteht.

Lemma 7.17

Bäume sind 2-färbbar.

Lemma 7.18

Ein Graph $G = (V, E)$ ist genau dann 2-färbbar, wenn G bipartit ist.

Lemma 7.19

Es sei G' ein Untergraph von G . Dann gilt

$$\chi(G') \leq \chi(G).$$

Einfache Aussagen zu Färbungen (2)

Das Haus vom Nikolaus ist 4-färbbar. Ist es auch 3-färbbar?

Lemma 7.20

Es sei G ein Graph. Dann gilt

$$\chi(G) \geq \omega(G).$$

Damit kann das Haus vom Nikolaus nicht 3-färbbar sein.

Der Fünf-Farben-Satz

Satz 7.21

Jeder planare Graph $G = (V, E)$ ist 5-färbbar.

Beweis.

Es sei $n := |V|$. Beweis erfolgt mit Induktion über n .

Für $n \leq 5$ ist der Satz richtig.

$n \rightarrow n + 1$: G sei ein planarer Graph mit $n + 1$ Knoten. Nach Korollar 7.13 hat G einen Knoten v mit $\deg(v) \leq 5$.

Gilt $\deg(v) < 5$: fertig.

Gilt $\deg(v) = 5$ und die fünf adjazenten Knoten benutzen weniger als fünf Farben: fertig.

Rest: Tafel .

Der Vier-Farben-Satz

Satz 7.22

Jeder planare Graph $G = (V, E)$ ist 4-färbbar.

- Damit ist die Vier-Farben-Vermutung wahr.
- Beweis 1976 durch [Kenneth Appel](#) und [Wolfgang Haken](#).
- Der Beweis von Appel und Haken enthielt eine so hohe Zahl von Fallunterscheidungen, dass diese nicht manuell sondern nur mit Hilfe eines Computers überprüft werden konnten.
- Zunächst Akzeptanzprobleme, der Satz gilt heute aber (auch durch weitere Forschung) als bewiesen.
- Erstes bedeutendes mathematisches Theorem, das mit Hilfe von Computern bewiesen wurde.

Chromatisches Polynom

Definition 7.23

Es sei $G = (V, E)$ ein Graph. Die Anzahl der Möglichkeiten G mit x Farben zu färben wird mit $f(G, x)$ bezeichnet. Hierbei müssen nicht alle x Farben verwendet werden.

Die Funktion $f(G, x)$ heißt **chromatisches Polynom** von G .

Beispiel 7.24

Der Graph



kann auf 12 verschiedene Arten mit 3 Farben gefärbt werden.

Sein chromatische Polynom lautet

$$f(G, x) = x(x-1)^2 = x^3 - 2x^2 + x.$$

Chromatische Polynome verschiedener Graphen

Für den vollständigen Graphen K_n gilt:

$$f(K_n, x) = x(x-1) \cdots (x-n+1).$$

Für das Komplement (n isolierte Knoten) gilt: $f(\bar{K}_n, x) = x^n$.

Es sei G ein Baum mit n Knoten. Dann gilt

$$f(G, x) = x(x-1)^{n-1}.$$

Besteht G aus den ZHKs G_1, \dots, G_n , dann gilt


$$f(G, x) = \prod_{i=1}^n f(G_i, x).$$

Berechnung des chromatischen Polynoms

Es sei $G = (V, E)$ ein Graph und $a, b \in V$ seien nicht adjazent. Dann sei

$$G_{\{a,b\}} = (V, E \cup \{a, b\})$$

Weiterhin bezeichne $G_{a=b}$ den Graphen, der entsteht, wenn in G die Knoten a und b entfernt und durch einen neuen Knoten c ersetzt werden. Dabei ist $\{v, c\}$ für $v \in V \setminus \{a, b\}$ genau dann eine Kante von $G_{a=b}$, wenn $\{v, a\}$ oder $\{v, b\}$ eine Kante von G war.

Beispiel zu $G_{\{a,b\}}$ und $G_{a=b}$: Tafel .

Satz 7.25

Sind a und b zwei nicht adjazente Knoten eines Graphen G , dann gilt

$$f(G, x) = f(G_{\{a,b\}}, x) + f(G_{a=b}, x)$$

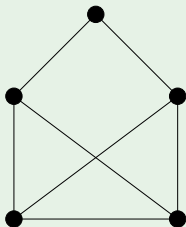

Beweis.

- Es seien a und b zwei nicht adjazente Knoten eines Graphen G .
- a und b sind entweder verschieden oder gleich gefärbt.
- Die Anzahl der Färbungen, bei denen a und b verschiedene Farben haben, ist identisch mit der Anzahl der Färbungen von $G_{\{a,b\}}$.
- Die Anzahl der Färbungen, bei denen a und b die gleiche Farbe haben, ist identisch mit der Anzahl der Färbungen von $G_{a=b}$.
- Die Anzahl aller Färbungen ergibt sich somit aus der Summe.

Bemerkung: Satz 7.25 gibt einem die Möglichkeit, das chromatische Polynom rekursiv zu berechnen.

Beispiel 7.26

Der Graph

hat das chromatische Polynom (Tafel )

$$\begin{aligned} f(G, x) &= f(K_5, x) + 3f(K_4, x) + 2f(K_3, x) \\ &= x^5 - 7x^4 + 19x^3 - 23x^2 + 10x. \end{aligned}$$

Wegen $f(G, 2) = 0$ und $f(G, 3) = 12$ folgt $\chi(G) = 3$.

Anwendungen

Stundenplanprobleme:

- Die Vorlesungen werden als Knoten betrachtet.
- Zwei Knoten sind durch eine Kante verbunden, wenn die Vorlesungen nicht gleichzeitig stattfinden sollen (gleicher Professor, gleiche Studenten, etc.).
- Die Farben entsprechen möglichen Vorlesungszeiten. Eine Färbung des Graphen stellt dann einen möglichen Stundenplan dar.
- Das chromatische Polynom gibt an, wieviele mögliche Stundenpläne es gibt, die chromatische Zahl die Anzahl der benötigten Zeitslots.

Beispiel 7.27

Wir wollen ein Sportfest planen, für das sich Sportler in unterschiedlichen Disziplinen angemeldet haben.

| Disziplin | Klaus | Heinz | Jupp | Peter | Toni | Ralf | Karl |
|-----------------|-------|-------|------|-------|------|------|------|
| 100 Meter Lauf | x | | x | | x | | x |
| Weitsprung | | x | | x | | x | |
| Kugelstoßen | x | x | | | | | x |
| Speerwerfen | | | x | | | x | |
| 5000 Meter Lauf | | | | | x | x | |
| Hochsprung | x | x | | x | x | | x |

Für jede Disziplin wird genau ein Zeitabschnitt benötigt. Die Sportler sollen an allen Disziplinen, zu denen sie sich angemeldet haben, auch teilnehmen können.

Fortsetzung Beispiel.

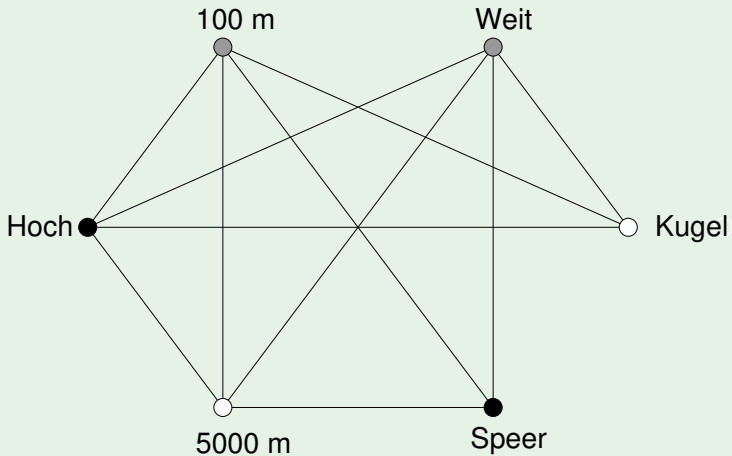
Wieviele Zeitabschnitte werden mindestens zur Planung des Sportfestes benötigt?

Welche Disziplinen sollten zu welchen Zeitabschnitten stattfinden?

Die Knoten des Graphen stellen die zu planenden Disziplinen dar. Die Farben entsprechen den Zeitabschnitten, die den Disziplinen zuzuordnen sind.

Zwei Disziplinen d_1 und d_2 werden genau dann durch eine Kante verbunden, wenn es einen Sportler gibt, der sich zu beiden Disziplinen angemeldet hat.

Fortsetzung Beispiel.



Fortsetzung Beispiel.

Da beispielsweise der induzierte Untergraph zu den Knoten Hoch, 100 m und 5000 m vollständig ist, benötigen wir mindestens drei Farben für die Färbung des gesamten Graphen.

Das Diagramm des Graphen zeigt, dass wir tatsächlich mit drei Farben (schwarz, grau, weiß) auskommen.

| Zeitabschnitt | Farbe | Disziplinen |
|---------------|---------|---------------|
| 1 | schwarz | Hoch, Speer |
| 2 | grau | 100 m, Weit |
| 3 | weiß | 5000 m, Kugel |

Zusammenfassung

- Planare Graphen: Eulersche Polyederformel und Schranken für die Anzahl von Kanten
- Färbungen, chromatische Zahl und chromatisches Polynom
- Planare Graphen sind 4-färbbar.
- Berechnung des chromatischen Polynoms durch Zerlegung in vollständige Graphen
- Anwendungen: Planungsprobleme mit konkurrierenden Ressourcen