Karlsruher Institut für Technologie

Algorithmen für Routenplanung

1. Sitzung, Sommersemester 2012

Thomas Pajor | 18. April 2012

Institut für Theoretische Informatik · Algorithmik · Prof. Dr. Dorothea Wagner

Organisatorisches

Dozent

Dr. Daniel Delling

- Microsoft Research Silicon Valley (USA)
- Mail: daniel.delling@microsoft.com
- Raum ψ (wird noch bekanntgegeben)

Dozent

Thomas Pajor

- Karlsruher Institut für Technologie
- Mail: pajor@kit.edu
- Raum 322 (kommt einfach vorbei)

Organisatorisches

Vorlesung

- Montags 14:00–15:30 Uhr, SR 301 (hier)
- Mittwochs 11:30–13:00 Uhr, SR 301 (hier)

Prüfung

- Prüfbar im Hauptstudium (Diplom) und *Master*
- Im Master: 5 ECTS Kredite
- VF: 1 (Theoretische Grundlagen), 2 (Algorithmentechnik)

Vorlesungswebseite:
`http://i11www.iti.uni-karlsruhe.de/teaching/sommer2012/routenplanung/index`

Vorläufige Termine

#	Tag	Datum	Dozent	#	Tag	Datum	Dozent
1	Mittwoch	18. April	Thomas	10	Montag	04. Juni	Daniel
2	Montag	23. April	Thomas	11	Mittwoch	06. Juni	Daniel
3	Mittwoch	02. Mai	Daniel	12	Montag	11. Juni	Daniel
4	Montag	07. Mai	Daniel	13	Mittwoch	13. Juni	Daniel
5	Mittwoch	09. Mai	Daniel	14	Montag	18. Juni	Daniel
6	Montag	14. Mai	Daniel	15	Mittwoch	20. Juni	Daniel
7	Mittwoch	16. Mai	Daniel	16	Montag	25. Juni	Daniel
8	Montag	21. Mai	Thomas	17	Montag	02. Juli	Thomas
9	Mittwoch	30. Mai	Thomas	18	Montag	09. Juli	Thomas

Siehe auch Vorlesungswebseite!

Weitere Vorlesungen

Algorithmen für Ad-hoc- und Sensornetze

- Dozent: Markus Völker
- Vorlesung: mittwochs 14:00–15:30 Uhr (SR 301)

Algorithmische Geometrie

- Dozenten: Dr. Martin Nöllenburg und Andreas Gemsa
- Vorlesung: dienstags 9:45–11:15 (SR 301)
- Übung donnerstags 10:15–11:00 (SR 131)

Algorithmen für planare Graphen

- Dozent: Dr. Ignaz Rutter
- Vorlesung: mittwochs 15:45–17:15 Uhr (SR 301)
- Vorlesung: donnerstags 14:00–15:30 Uhr (SR 301)

0. Motivation

Worum geht es bei der Routenplanung?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problemstellung

Gesucht:

- finde die beste Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Kantengewichte sind Reisezeiten
- kürzeste Wege in G entsprechen schnellsten Verbindungen
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind groß
- Dijkstra zu langsam (> 1 Sekunde)

Problemstellung

Gesucht:

- finde die beste Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Kantengewichte sind Reisezeiten
- kürzeste Wege in G entsprechen schnellsten Verbindungen
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind groß
- Dijkstra zu langsam (> 1 Sekunde)

Problemstellung

Gesucht:

- finde die beste Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Kantengewichte sind Reisezeiten
- kürzeste Wege in G entsprechen schnellsten Verbindungen
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind groß
- Dijkstra zu langsam ($>$ 1 Sekunde)

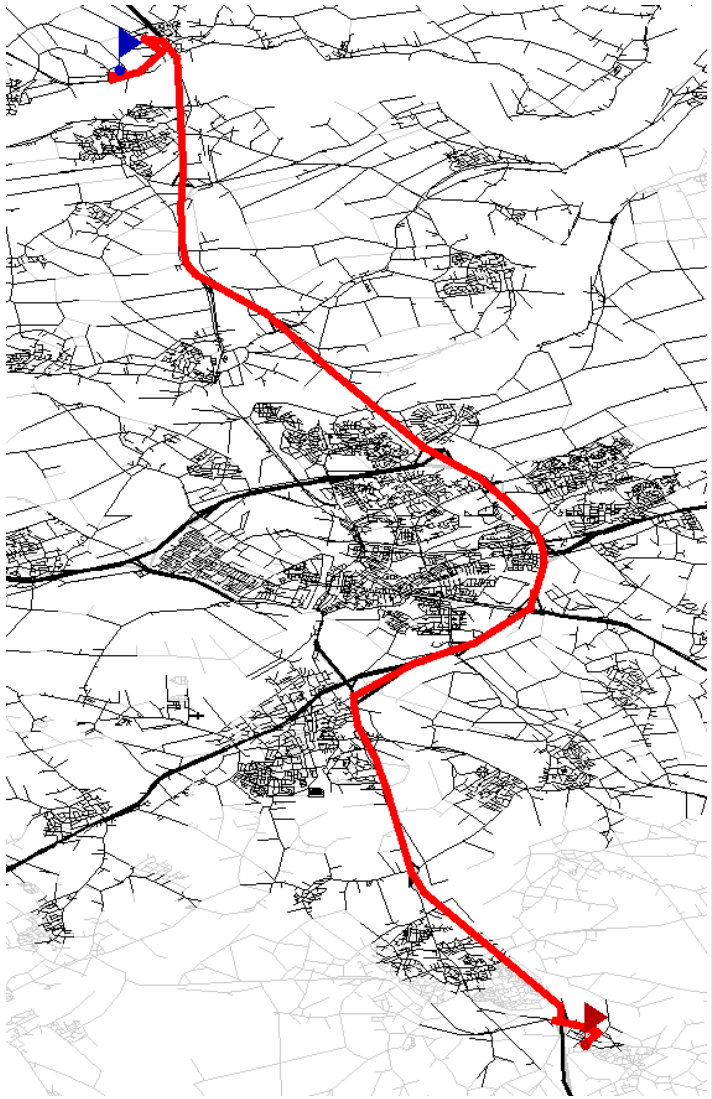
Beschleunigungstechniken

Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen unnötig

Idee:

- Zwei-Phasen Algorithmus:
  - offline: berechne Zusatzinformation während Vorberechnung
  - online: beschleunige Berechung mit diesen Zusatzinformationen
- drei Kriterien:
  - wenig Zusatzinformation $\mathcal{O}(n)$
  - kurze Vorberechnung (im Bereich Stunden/Minuten)
  - hohe Beschleunigung



Beschleunigungstechniken

Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen unnötig

Idee:

- Zwei-Phasen Algorithmus:
  - offline: berechne Zusatzinformation während Vorberechnung
  - online: beschleunige Berechung mit diesen Zusatzinformationen
- drei Kriterien:
  - wenig Zusatzinformation $\mathcal{O}(n)$
  - kurze Vorberechnung (im Bereich Stunden/Minuten)
  - hohe Beschleunigung

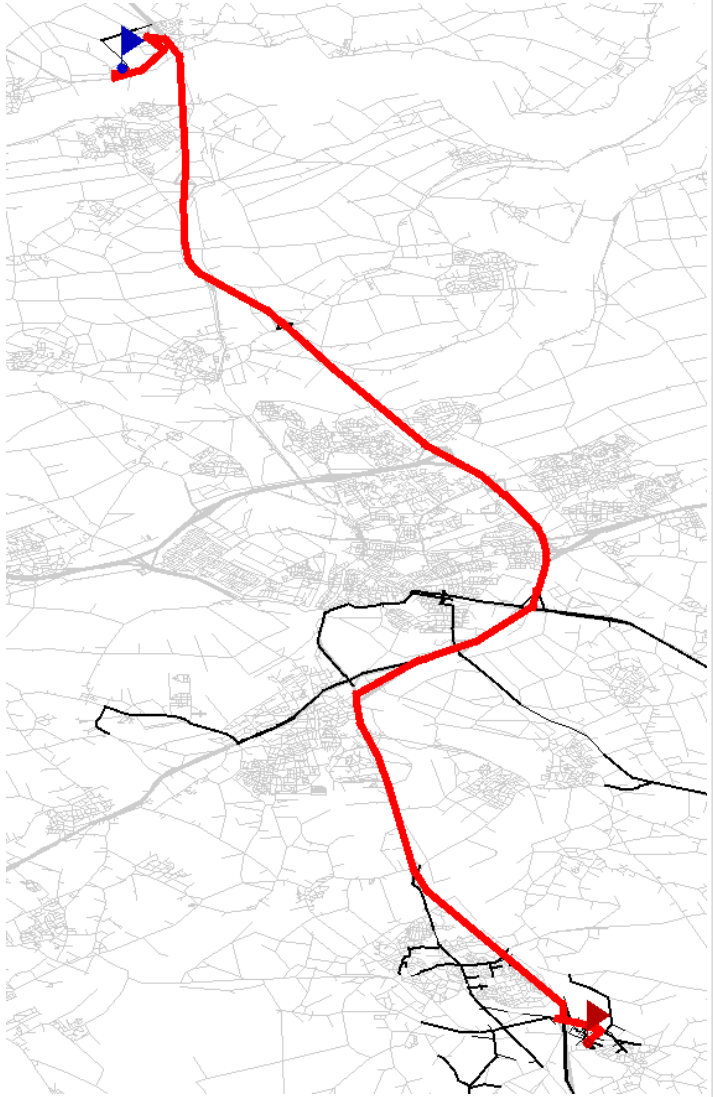
Beschleunigungstechniken

Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen unnötig

Idee:

- Zwei-Phasen Algorithmus:
  - offline: berechne Zusatzinformation während Vorberechnung
  - online: beschleunige Berechung mit diesen Zusatzinformationen
- drei Kriterien:
  - wenig Zusatzinformation $\mathcal{O}(n)$
  - kurze Vorberechnung (im Bereich Stunden/Minuten)
  - hohe Beschleunigung



Unterschiede zur Industrie

Industrie:

- Falk, TomTom, bahn.de, usw.
- alles heuristische Verfahren
  - betrachte nur noch “wichtige” Kanten wenn mehr als x Kilometer von Start weg
  - Kombination mit A*-Suche
  - langsam!
- Ausnahme: Bing Maps, Google Maps

Unser Anspruch:

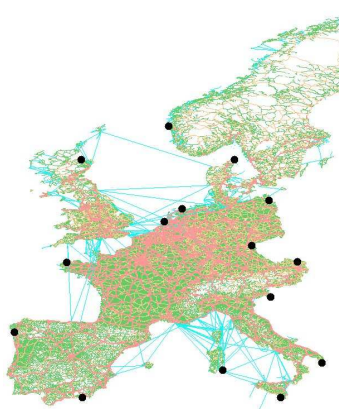
- Anfragen sollen beweisbar korrekt sein

$\Rightarrow$ weniger Ausnahmeregelungen

$\Rightarrow$ schneller (!)

- Verfahren sollen nach und nach in der Industrie eingesetzt werden

Ergebnisse



Eingabe: Straßennetzwerk von Westeuropa

- 18 Mio. Knoten
- 42 Mio. Kanten

		Vorberechnung		Anfrage	
	Jahr	Zeit [h:m]	Platz [byte/n]	Zeit [ms]	Beschl.
Dijkstra	1959*	0:00	0	5 153.0	0
Arc-Flags	2004	17:08	19	1.6	3 221
Transit-Node Routing	2006	1:15	226	0.0043	1.2 Mio.
Contraction Hier.	2008	0:29	0	0.19	27 121
CH + Arc-Flags	2008	1:39	12	0.017	ca. 300 000
TNR + AF	2008	3:49	312	0.0019	ca. 3 Mio.
Hub Labels	2011	≈ 4:30	1 241	≈ 0.0005	ca. 11 Mio.

* Damalige Variante deutlich langsamer, wir sehen nachher, warum

Schwerpunkte

- Algorithm Engineering + ein bisschen Theorie
- Beschleunigungstechniken
- Implementierungsdetails
- Ergebnisse auf Real-Welt Daten
- aktuellster Stand der Forschung (Veröffentlichungen bis 2012)
- ideale Grundlage für Bachelor, Master und Diplomarbeiten

Was diese Vorlesung nicht ist

keine Algorithmentechnik 2

- Vertiefung von kürzesten Wegen (Dijkstra)
  - Grundlagen sind Stoff von Info 2/Algotech; heute nochmal Crashkurs
- Grundvorlesung “vereinfachen” Wahrheit oft
- Implementierung
- Betonung auf Messergebnisse

keine reine Theorievorlesung

- relativ wenig Beweise (wenn doch, eher kurz)
- reale Leistung vor Asymptotik
- Vielen vorkommende Optimierungsproblemen sind $\mathcal{NP}$-schwer

Inhalt der Vorlesung

1. Grundlagen

- Algorithm Engineering
- Graphen, Modelle, usw.
- Kürzeste Wege
- Dijkstra's Algorithmus

2. Beschleunigung von (statischen) Punkt-zu-Punkt Anfragen

- zielgerichtete Verfahren
- hierarchische Techniken
- many-to-many-Anfragen und Distanztabellen
- Kombinationen

Inhalt der Vorlesung

3. Theorie

- Theoretische Charakterisierung von Straßennetzwerken
- Highway-Dimension
- Komplexität von Beschleunigungstechniken

4. Fortgeschrittene Szenarien

- schnelle many-to-many und all-pairs shortest-paths
- Alternativrouten
- dynamische Szenarien
- zeitabhängige Routenplanung
- Fahrplanauskunft
- multi-modale Routenplanung

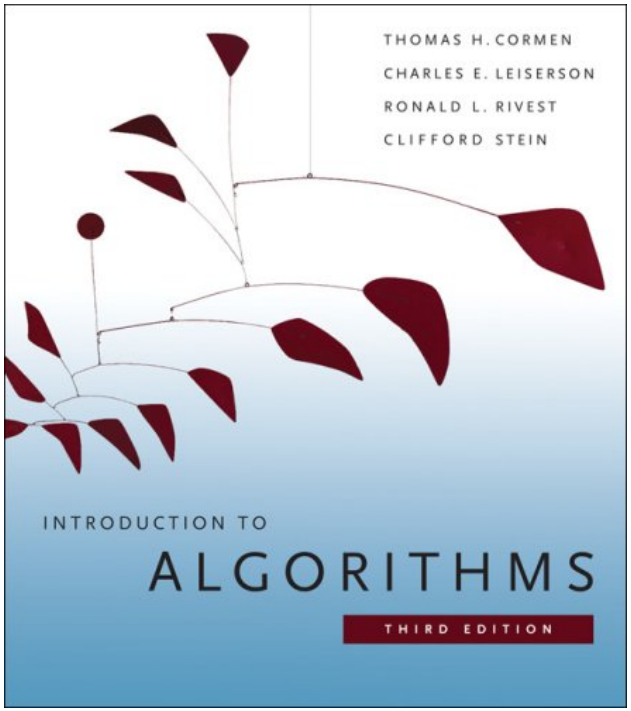
Nützliche Vorkenntnisse

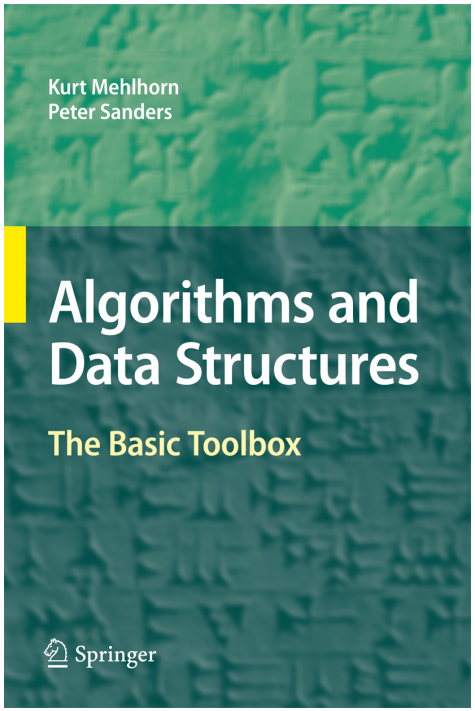
- Informatik I/II oder Algorithmen I
- Algorithmentechnik oder Algorithmen II (muss aber nicht sein)
- ein bisschen Rechnerarchitektur
- passive Kenntnisse von C++/Java

Vertiefungsgebiet: Algorithmentechnik, (Theoretische Grundlagen)

Material

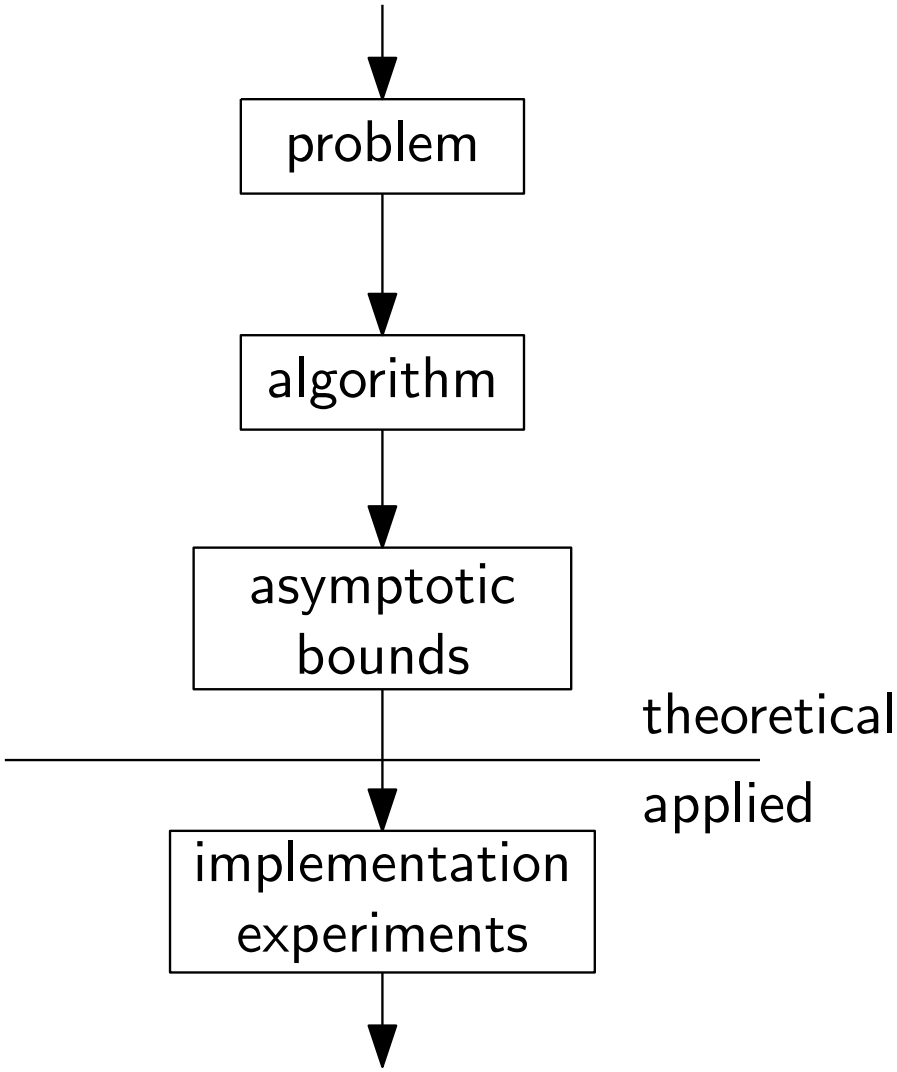
- Folien
- wissenschaftliche Aufsätze (siehe Vorlesunghomepage)
- Basiskenntnisse:





1. Grundlagen

Klassischer Ansatz



Lücke Theorie vs. Praxis

Theorie	vs.	Praxis
einfach	Problem-Modell	komplex
einfach	Maschinenmodell	komplex
komplex	Algorithmen	einfach
fortgeschritten	Datenstrukturen	einfach
worst-case	Komplexitäts-Messung	typische Eingaben
asymptotisch	Effizienz	konstante Faktoren

hier:

- sehr anwendungsnahes Gebiet
- Eingaben sind echte Daten
  - Straßengraphen
  - Eisenbahn (Fahrpläne)
  - Flugpläne

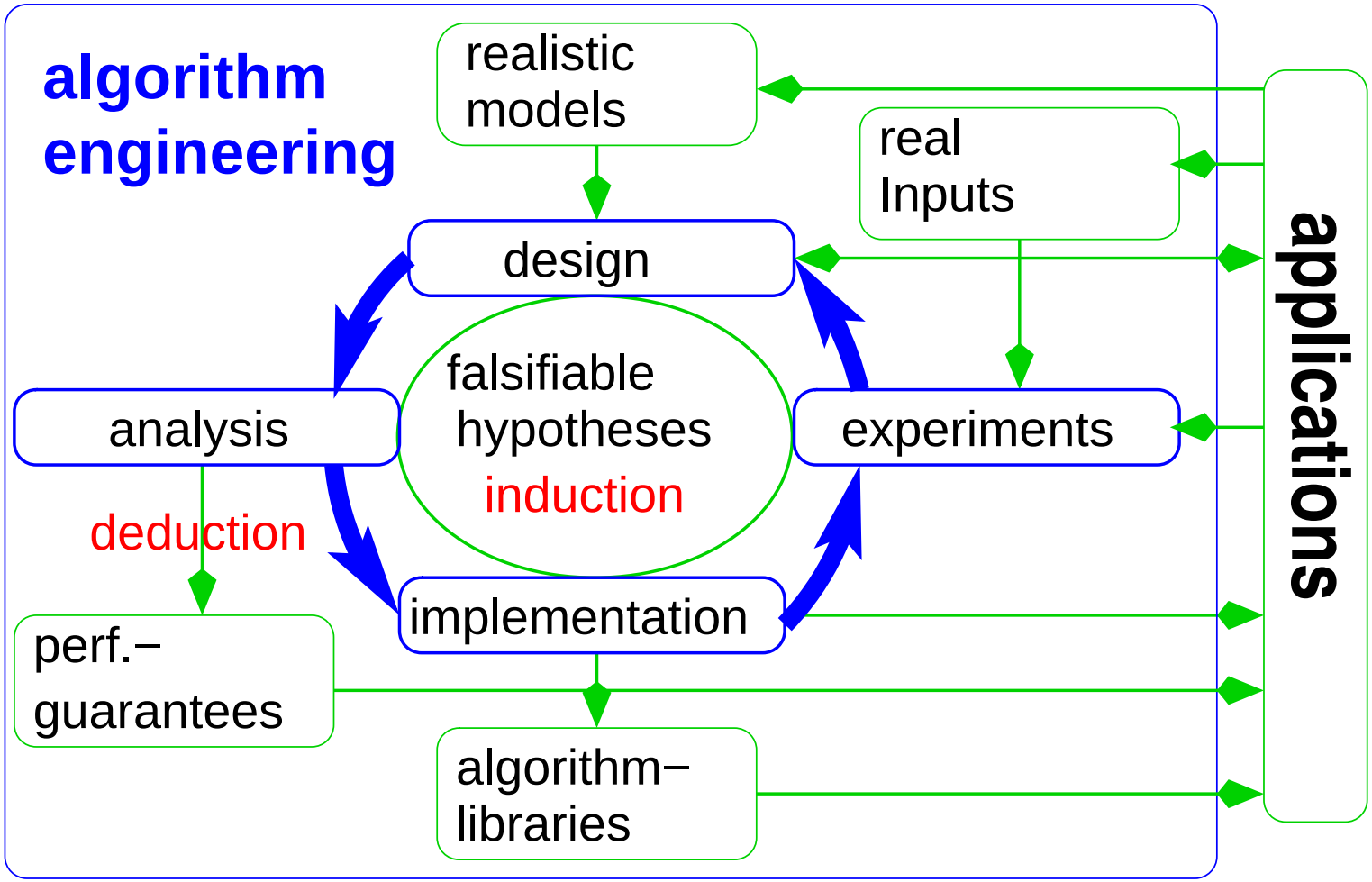
Lücke Theorie vs. Praxis

Theorie	vs.	Praxis
einfach	Problem-Modell	komplex
einfach	Maschinenmodell	komplex
komplex	Algorithmen	einfach
fortgeschritten	Datenstrukturen	einfach
worst-case	Komplexitäts-Messung	typische Eingaben
asymptotisch	Effizienz	konstante Faktoren

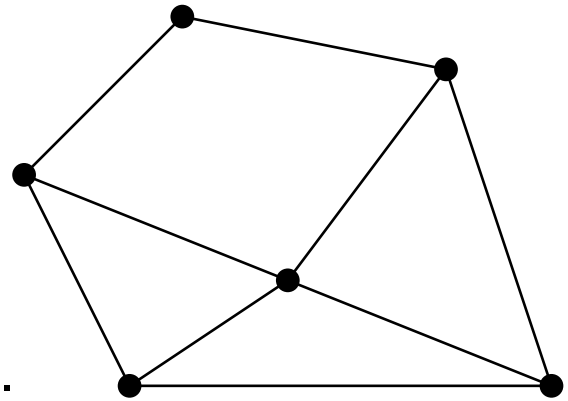
hier:

- sehr anwendungsnahes Gebiet
- Eingaben sind echte Daten
  - Straßengraphen
  - Eisenbahn (Fahrpläne)
  - Flugpläne

Algorithm Engineering

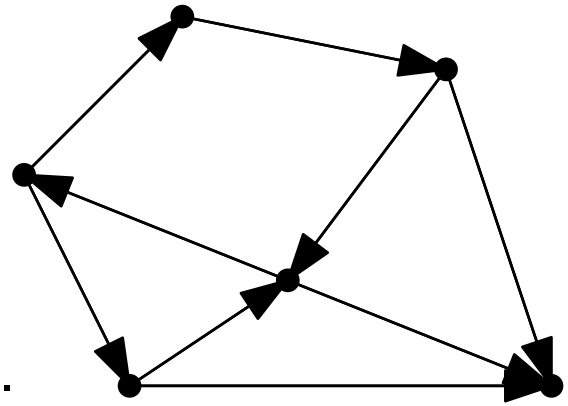


Grundbegriffe: Graphen



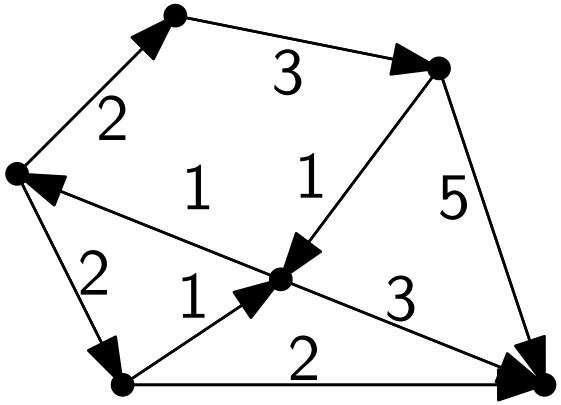
- **Graph:** Tupel $G := (V, E)$
  - endliche Knotenmenge V
  - endliche Kantenmenge E
  - $n := |V|$, $m := |E|$
- **ungerichtet:** Kanten sind Knotenpaar, d.h. $E \subseteq \binom{V}{2} = \{\{u, v\} \mid u \neq v,\ u, v \in V\}$
  - Grad: $\deg(u) = \sum_{\{u,v\} \in E}$
- **gerichtet:** Kanten sind geordnete Paare, d.h. $E \subseteq \{(u, v) \mid u \neq v, u, v \in V\}$
  - Ausgangsgrad: $\deg_{\text{out}}(u) = \sum_{(u,v) \in E}$
  - Eingangsgrad: $\deg_{\text{in}}(u) = \sum_{(v,u) \in E}$
- **einfach:** keine Multi-Kanten (E ist normale Menge)

Grundbegriffe: Graphen



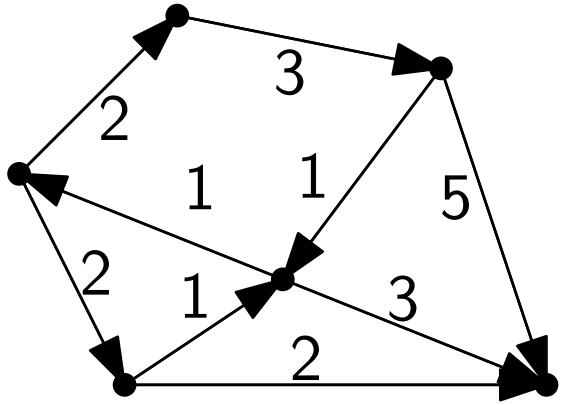
- **Graph:** Tupel $G := (V, E)$
  - endliche Knotenmenge V
  - endliche Kantenmenge E
  - $n := |V|$, $m := |E|$
- **ungerichtet:** Kanten sind Knotenpaar, d.h. $E \subseteq \binom{V}{2} = \{\{u, v\} \mid u \neq v,\ u, v \in V\}$
  - Grad: $\deg(u) = \sum_{\{u,v\} \in E}$
- **gerichtet:** Kanten sind geordnete Paare, d.h. $E \subseteq \{(u, v) \mid u \neq v, u, v \in V\}$
  - Ausgangsgrad: $\deg_{\text{out}}(u) = \sum_{(u,v) \in E}$
  - Eingangsgrad: $\deg_{\text{in}}(u) = \sum_{(v,u) \in E}$
- **einfach:** keine Multi-Kanten (E ist normale Menge)

Grundbegriffe: Graphen



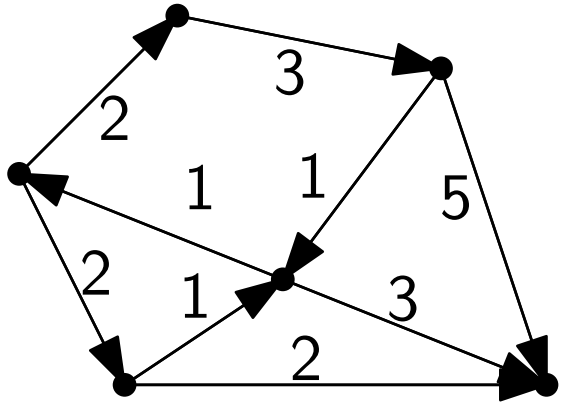
- **gewichtet:** Kantengewichtsfunktion
  - erstmal len : $E \to \mathbb{R}^+$
- **dünn:** $m \in \mathcal{O}(n)$
- **planar:** kreuzungsfrei einbettbar

Grundbegriffe: Graphen



- **gewichtet:** Kantengewichtsfunktion
  - erstmal len : $E \to \mathbb{R}^+$
- **dünn:** $m \in \mathcal{O}(n)$
- **planar:** kreuzungsfrei einbettbar

Grundbegriffe: Graphen



- **gewichtet:** Kantengewichtsfunktion
  - erstmal len : $E \to \mathbb{R}^+$
- **dünn:** $m \in \mathcal{O}(n)$
- **planar:** kreuzungsfrei einbettbar

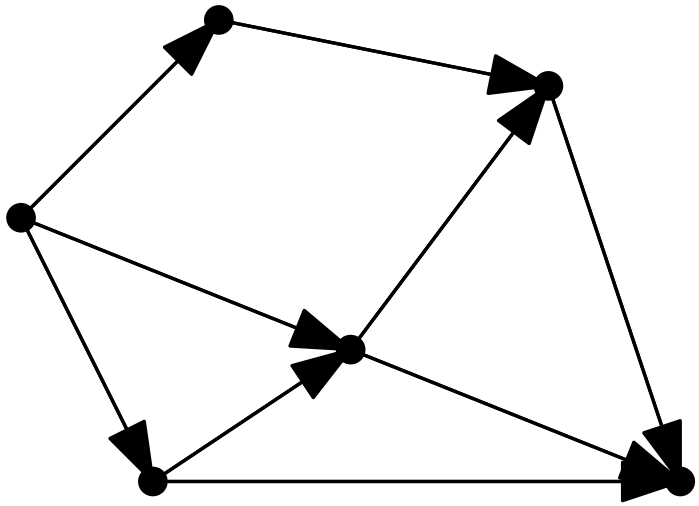
Spezielle Graphen

DAG:

- directed acyclic graph
- gerichtet, zyklenfrei

Baum:

- zyklenfrei
- $m = n - 1$
- alle Knoten von Wurzelknoten erreichbar



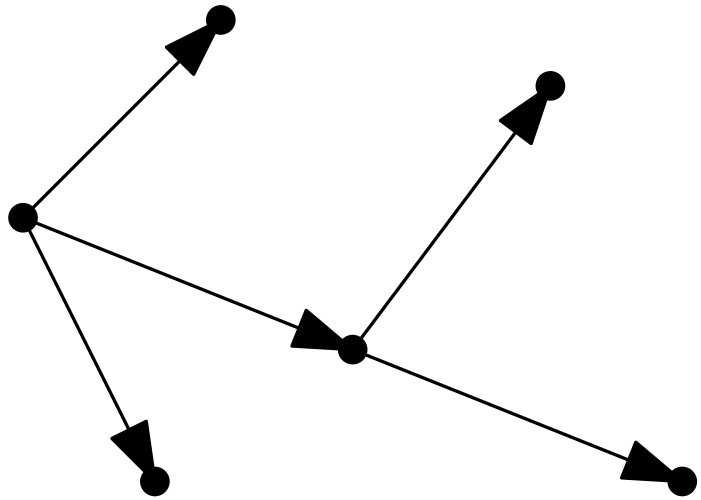
Spezielle Graphen

DAG:

- directed acyclic graph
- gerichtet, zyklenfrei

Baum:

- zyklenfrei
- $m = n - 1$
- alle Knoten von Wurzelknoten erreichbar



Wege und Distanzen

Weg:
Zu gegebenen Knoten $s, t \in V$ ist $P := (v_1 = s, v_2, \ldots, v_k = t)$ ein s-t-Weg, g. d. w. für alle $v_i, v_{i+1} \in P$ gilt $(v_i, v_{i+1}) \in E$.

Länge:
Die Länge eines Weges P ist $|P| := \sum_{(v_i, v_{i+1}) \in P} \text{len}(v_i, v_{i+1})$.

kürzester Weg:
Der kürzeste s-t-Weg $\Pi(s, t)$ ist ein s-t-Weg $P := (s, \ldots, t)$ mit minimaler Länge $|P|$.

Distanz:
Die Distanz $d(s, t)$ für zwei Knoten $s, t \in V$ ist definiert als

$$d(s,t) = \begin{cases} |\Pi(s,t)| & \text{wenn } \exists \text{ Weg von } s \text{ nach } t \text{ in } G \\ \infty & \text{sonst} \end{cases}$$

d ist symmetrisch in ungerichteten, und i. A. nicht symmetrisch in gerichteten Graphen.

Wege und Distanzen

Weg:
Zu gegebenen Knoten $s, t \in V$ ist $P := (v_1 = s, v_2, \ldots, v_k = t)$ ein s-t-Weg, g. d. w. für alle $v_i, v_{i+1} \in P$ gilt $(v_i, v_{i+1}) \in E$.

Länge:
Die Länge eines Weges P ist $|P| := \sum_{(v_i, v_{i+1}) \in P} \text{len}(v_i, v_{i+1})$.

kürzester Weg:
Der kürzeste s-t-Weg $\Pi(s, t)$ ist ein s-t-Weg $P := (s, \ldots, t)$ mit minimaler Länge $|P|$.

Distanz:
Die Distanz $d(s, t)$ für zwei Knoten $s, t \in V$ ist definiert als

$$d(s,t) = \begin{cases} |\Pi(s,t)| & \text{wenn } \exists \text{ Weg von } s \text{ nach } t \text{ in } G \\ \infty & \text{sonst} \end{cases}$$

d ist symmetrisch in ungerichteten, und i. A. nicht symmetrisch in gerichteten Graphen.

Wege und Distanzen

Weg:
Zu gegebenen Knoten $s, t \in V$ ist $P := (v_1 = s, v_2, \ldots, v_k = t)$ ein s-t-Weg, g. d. w. für alle $v_i, v_{i+1} \in P$ gilt $(v_i, v_{i+1}) \in E$.

Länge:
Die Länge eines Weges P ist $|P| := \sum_{(v_i, v_{i+1}) \in P} \text{len}(v_i, v_{i+1})$.

kürzester Weg:
Der kürzeste s-t-Weg $\Pi(s, t)$ ist ein s-t-Weg $P := (s, \ldots, t)$ mit minimaler Länge $|P|$.

Distanz:
Die Distanz $d(s, t)$ für zwei Knoten $s, t \in V$ ist definiert als

$$d(s,t) = \begin{cases} |\Pi(s,t)| & \text{wenn } \exists \text{ Weg von } s \text{ nach } t \text{ in } G \\ \infty & \text{sonst} \end{cases}$$

d ist symmetrisch in ungerichteten, und i. A. nicht symmetrisch in gerichteten Graphen.

Wege und Distanzen

Weg:
Zu gegebenen Knoten $s, t \in V$ ist $P := (v_1 = s, v_2, \ldots, v_k = t)$ ein s-t-Weg, g. d. w. für alle $v_i, v_{i+1} \in P$ gilt $(v_i, v_{i+1}) \in E$.

Länge:
Die Länge eines Weges P ist $|P| := \sum_{(v_i, v_{i+1}) \in P} \text{len}(v_i, v_{i+1})$.

kürzester Weg:
Der kürzeste s-t-Weg $\Pi(s, t)$ ist ein s-t-Weg $P := (s, \ldots, t)$ mit minimaler Länge $|P|$.

Distanz:
Die Distanz $d(s, t)$ für zwei Knoten $s, t \in V$ ist definiert als

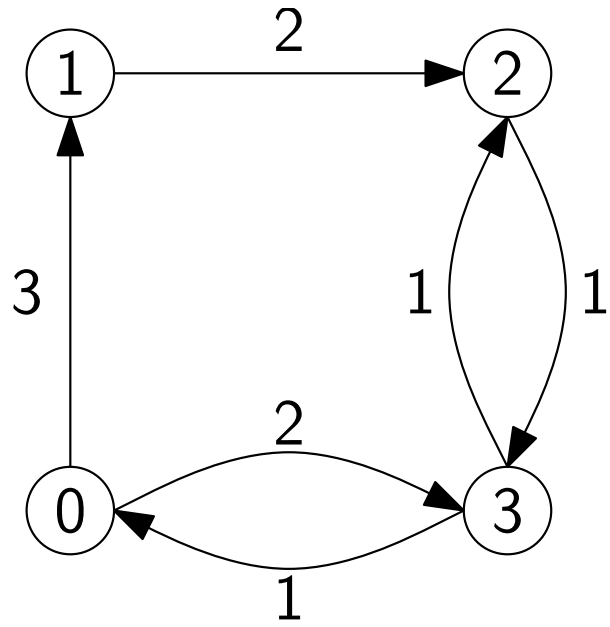
$$d(s,t) = \begin{cases} |\Pi(s,t)| & \text{wenn } \exists \text{ Weg von } s \text{ nach } t \text{ in } G \\ \infty & \text{sonst} \end{cases}$$

d ist symmetrisch in ungerichteten, und i. A. nicht symmetrisch in gerichteten Graphen.

Graph-Repräsentationen

Drei klassische Ansätze:

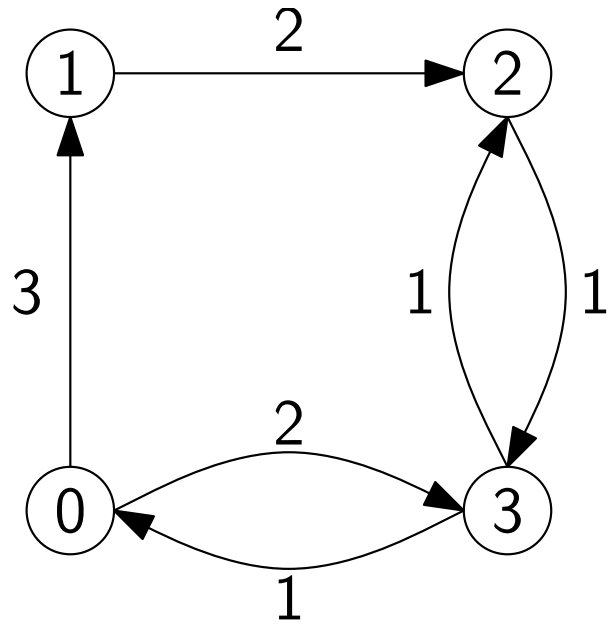
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

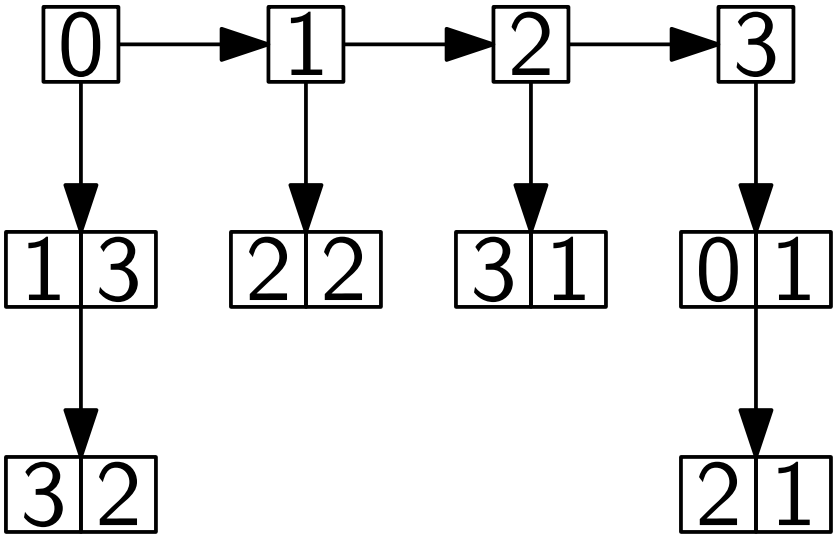


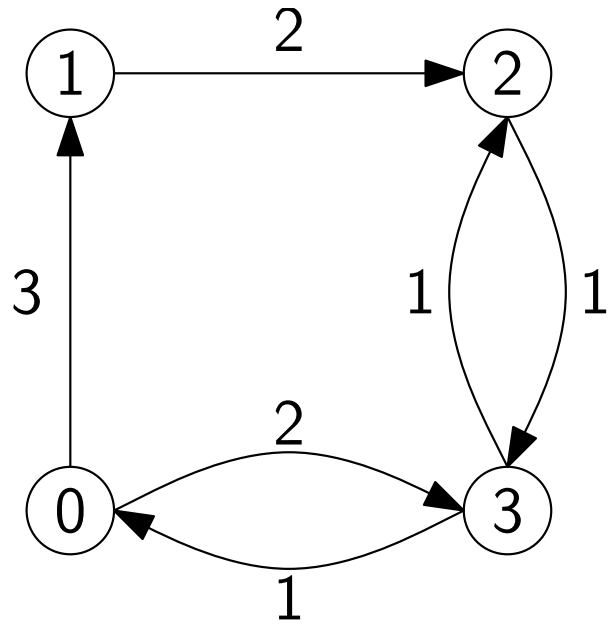
	0	1	2	3
0	–	3	–	2
1	–	–	2	–
2	–	–	–	1
3	1	–	1	–

Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

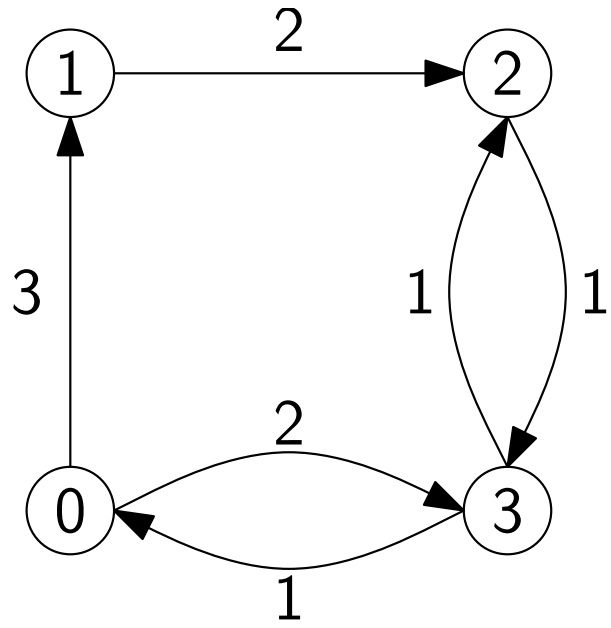


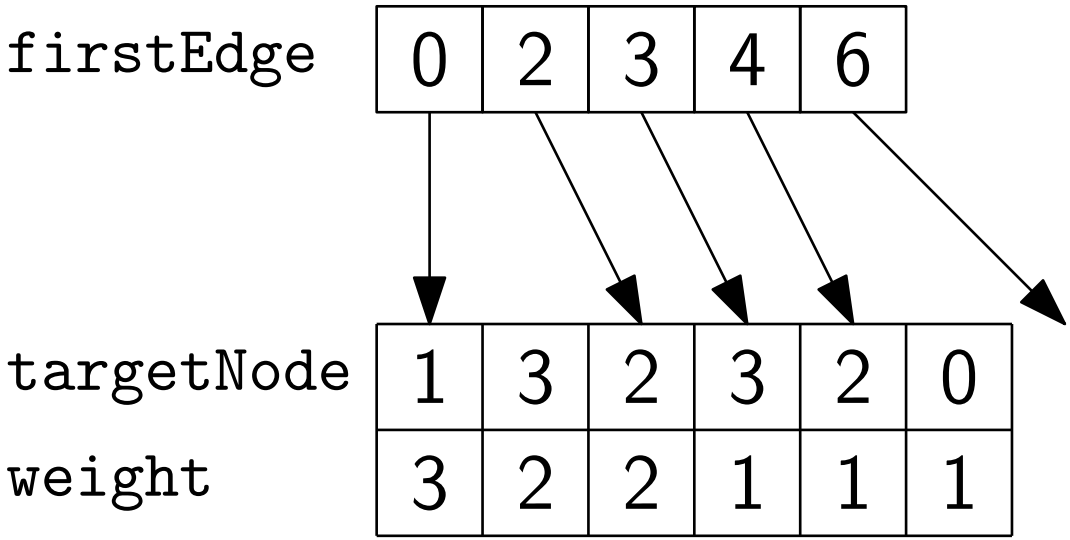


Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray





Was brauchen wir?

Eigenschaften:	Matrix	Liste	Array
Speicher	$\mathcal{O}(n^2)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$
Ausgehende Kanten iterieren	$\mathcal{O}(n)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Kantenzugriff (u, v)	$\mathcal{O}(1)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Effizienz (Speicher)	$+$	$-$	$+$
Updates (topologisch)	$+$	$+$	$-$
Updates (Gewicht)	$+$	$+$	$+$

Fragen:

- Was brauchen wir?
- Was muss nich supereffizient sein?
- erstmal Modelle anschauen!

Was brauchen wir?

Eigenschaften:	Matrix	Liste	Array
Speicher	$\mathcal{O}(n^2)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$
Ausgehende Kanten iterieren	$\mathcal{O}(n)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Kantenzugriff (u, v)	$\mathcal{O}(1)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Effizienz (Speicher)	$+$	$-$	$+$
Updates (topologisch)	$+$	$+$	$-$
Updates (Gewicht)	$+$	$+$	$+$

Fragen:

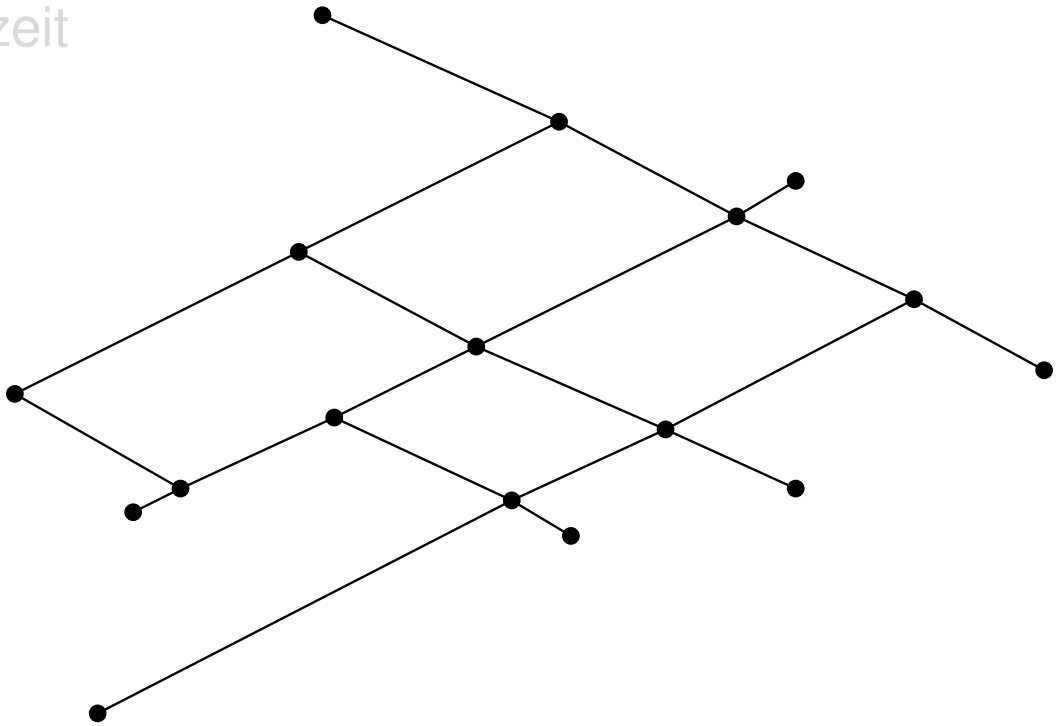
- Was brauchen wir?
- Was muss nich supereffizient sein?
- erstmal Modelle anschauen!

Modellierung (Straßengraphen)

Modellierung (Straßengraphen)

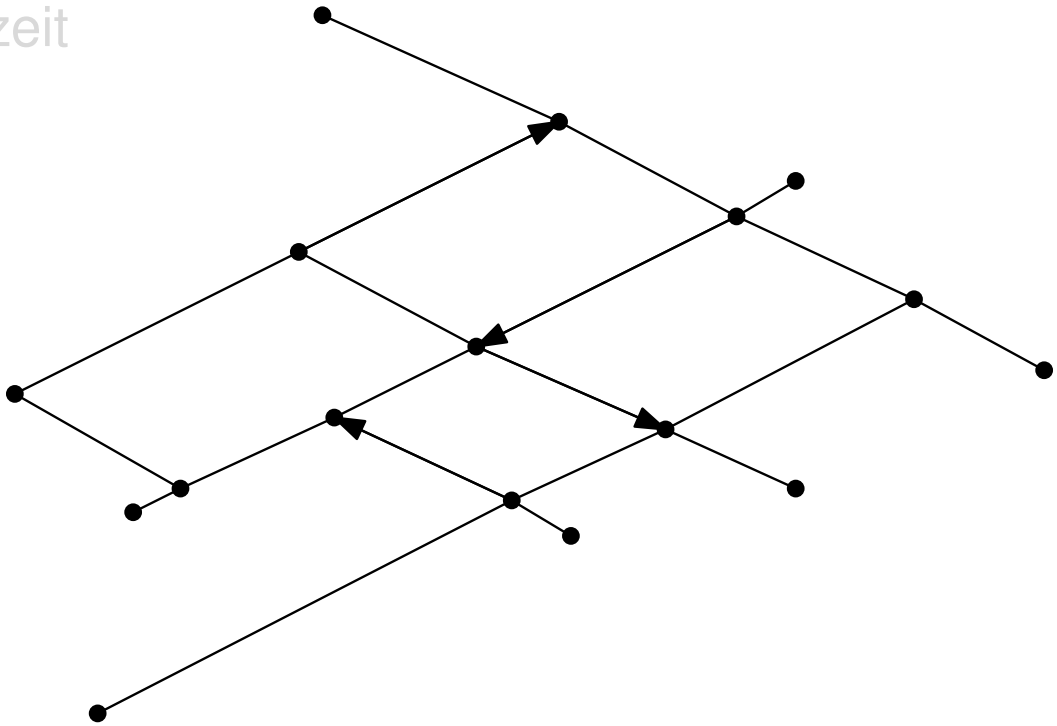
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



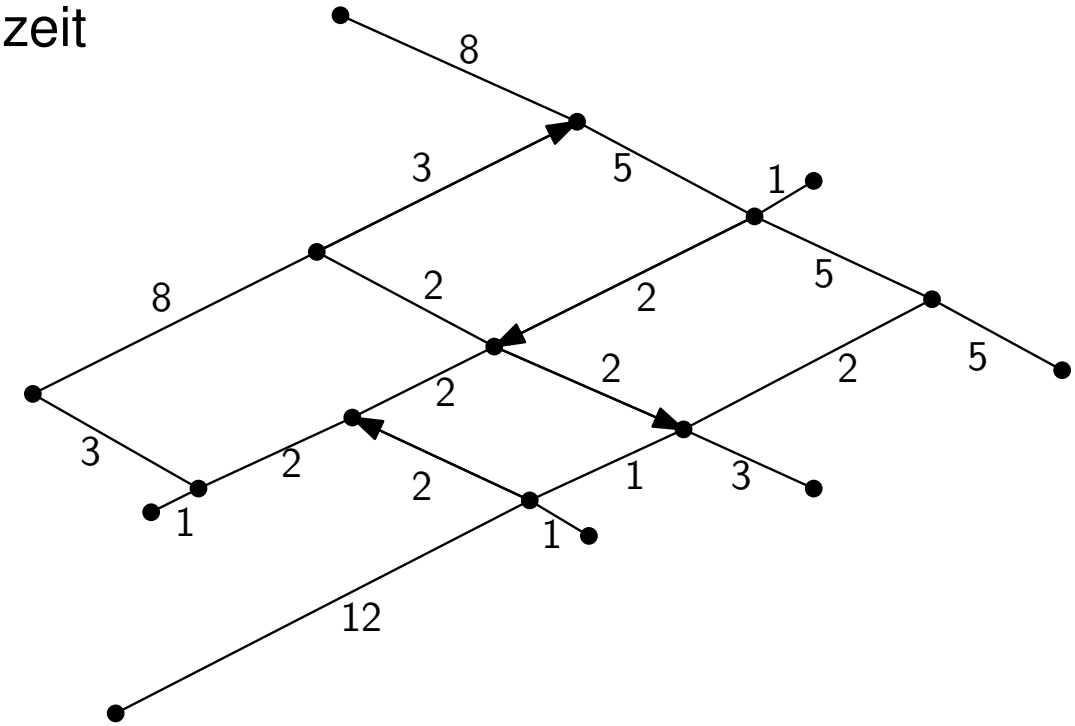
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



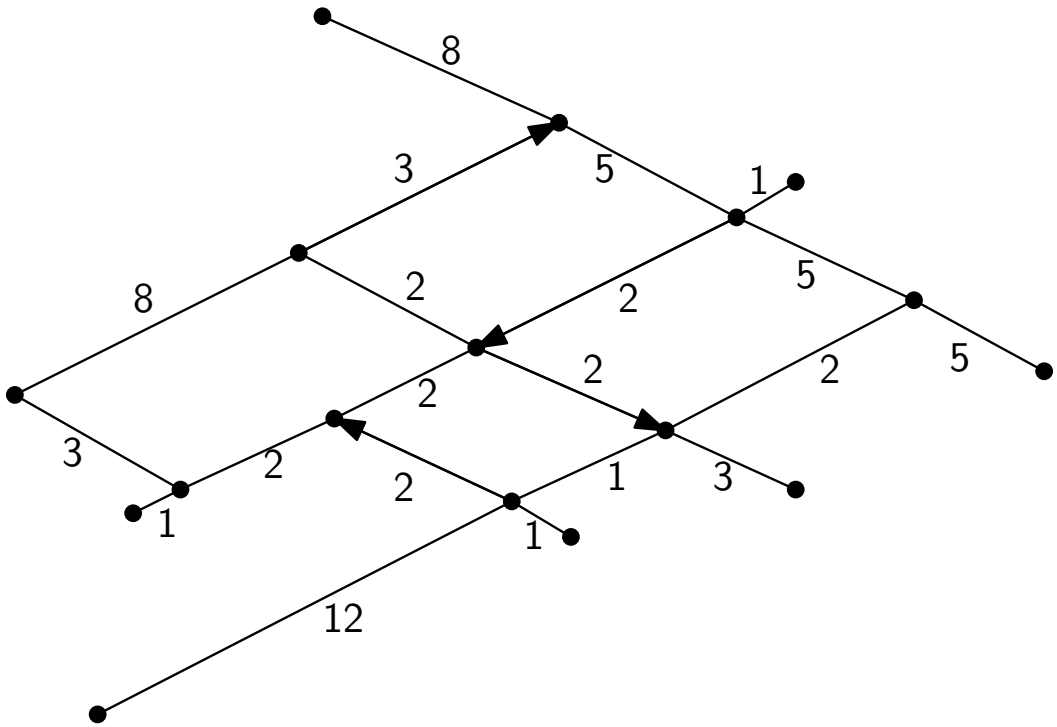
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



Modellierung (Straßengraphen)

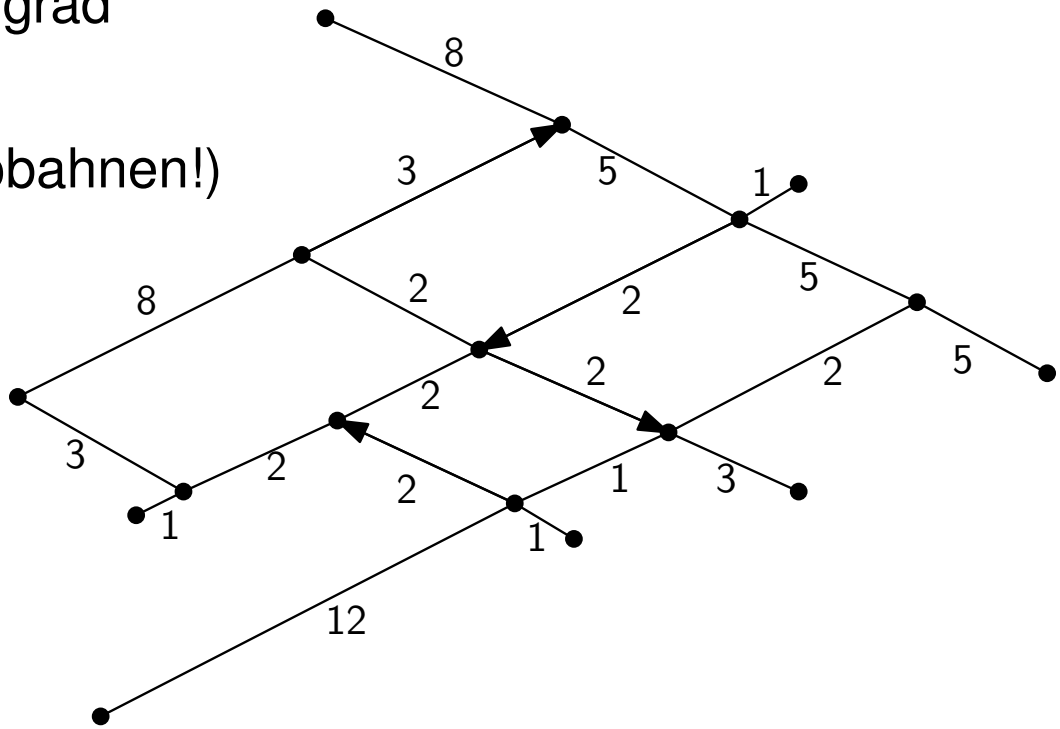
Eigenschaften (sammeln):



Modellierung (Straßengraphen)

Eigenschaften:

- dünn
- (fast) ungerichtet
- geringer Knotengrad
- Kantenzüge
- Hierarchie (Autobahnen!)



Eigenschaften der Graphen

- dünn (!)
- gerichtet
- geringer Knotengrad
- meist verborgene Hierarchie (Autobahnen, ICE)
- Einbettung vorhanden (fast planar?)
- teilweise zeitabhängig
- Kantengewichte nicht-negativ
- zusammenhängend

Diskussion:

- berechnen beste Verbindungen in solchen Netzwerken
- zeitabhängigkeit (war lange) ein Problem
- ab jetzt: zeitunabhängige Netze (Straßen)
- später: zeitabhängig

Eigenschaften der Graphen

- dünn (!)
- gerichtet
- geringer Knotengrad
- meist verborgene Hierarchie (Autobahnen, ICE)
- Einbettung vorhanden (fast planar?)
- teilweise zeitabhängig
- Kantengewichte nicht-negativ
- zusammenhängend

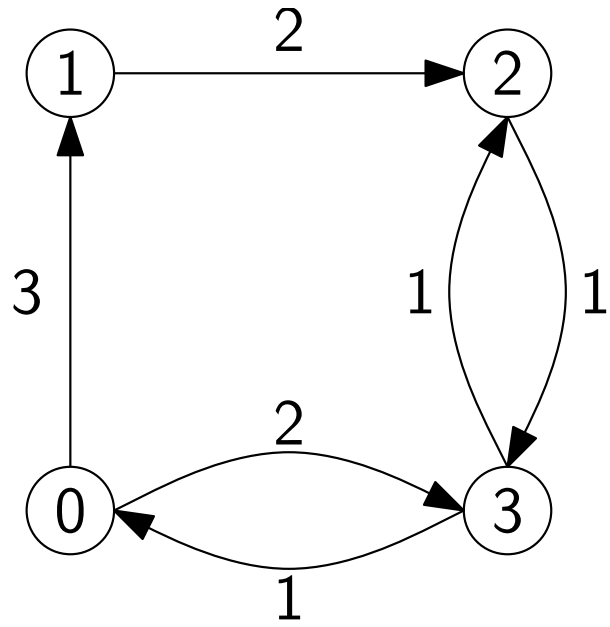
Diskussion:

- berechnen beste Verbindungen in solchen Netzwerken
- zeitabhängigkeit (war lange) ein Problem
- ab jetzt: zeitunabhängige Netze (Straßen)
- später: zeitabhängig

Datenstruktur

Problem:

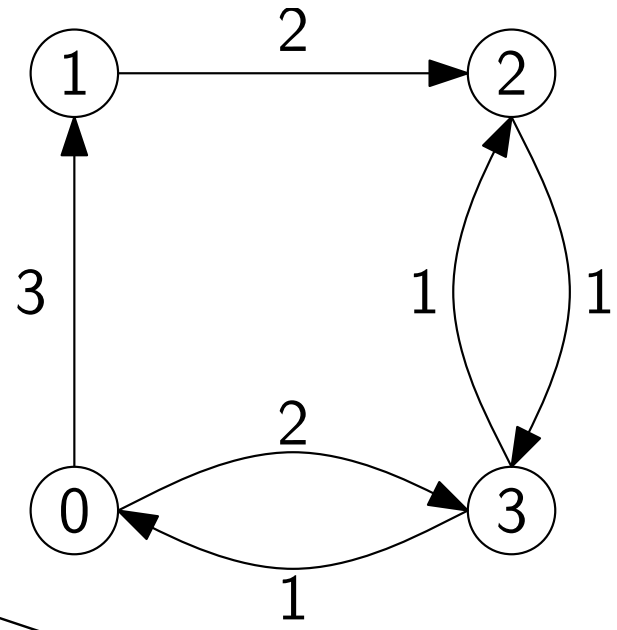
- ein- und ausgehende Kanten
- (fast) ungerichtet



Datenstruktur

Problem:

- ein- und ausgehende Kanten
- (fast) ungerichtet



`firstEdge`	0	3	5	7	10					
`targetNode`	1	3	3	0	2	1	3	0	0	2
`weight`	3	2	1	3	2	2	1	1	2	1
`isIncoming`	—	—	√	√	—	√	√	—	√	√
`isOutgoing`	√	√	—	—	√	—	√	√	—	√