Karlsruher Institut für Technologie

Algorithmen für Routenplanung

1. Sitzung, Sommersemester 2010

Thomas Pajor | 12. April 2010

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

Organisatorisches

Vorlesung

- Thomas Pajor
- Montags 14:00–15:30 Uhr, SR 301 (hier)
- E-Mail: `pajor@kit.edu`
- Sprechstunde: Kommt einfach vorbei (Raum 322)

Übung

- ist in Vorlesung integriert
- Besprechung mancher Aufgaben

Vorlesungsseite:
`http://i11www.iti.uni-karlsruhe.de/teaching/sommer2010/routenplanung/index`

Vorläufige Termine

Woche	Tag	Datum	Woche	Tag	Datum
1	Montag	12. April	9	Montag	7. Juni
2	—	—	10	Montag	14. Juni
3	Montag	26. April	11	Freitag	25. Juni
3	Freitag	30. April	12	Montag	28. Juni
4	Montag	3. Mai	13	Montag	5. Juli
5	Montag	10. Mai	13	Freitag	9. Juli
6	Montag	17. Mai	14	—	—
7	Freitag	28. Mai			
8	Montag	31. Mai			

Siehe auch Vorlesungswebseite!

Weitere Vorlesungen

Karlsruher Institut für Technologie

Algorithmen für planare Graphen

- Dozenten: Robert Görke und Ignaz Rutter
- Vorlesung: dienstags 14:00–15:30 Uhr (SR 301)
- Übung (14-tg): mittwochs 15:45–17:15 Uhr (SR 131)

Algorithmische Spieltheorie

- Dozent: Priv. Doz. Dr. Rob van Stee
- Vorlesung: dienstags 11:30–13:00 (SR -120)

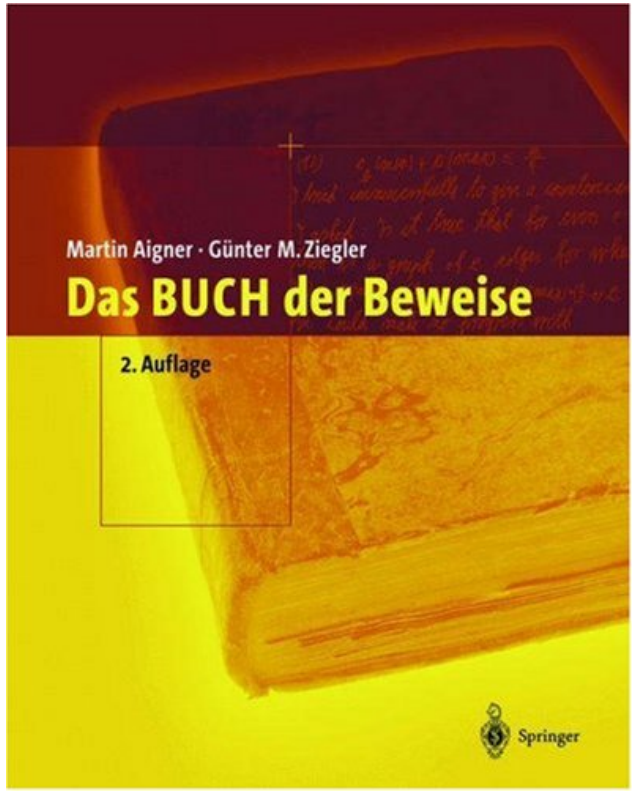
Approximations- und Online-Algorithmen

- Dozent: Priv. Doz. Dr. Rob van Stee
- Vorlesung: mittwochs 11:30–13:00 Uhr (SR 301)

Weitere Veranstaltungen

Seminar: Proofs from The Book

- Betreuer:
  - Priv. Doz. Rob van Stee
  - Markus Krug
  - Ignaz Rutter
- Termin:
  mittwochs 9:45–11:15 Uhr, SR 301
- Vorbesprechung:
  14. April, 9:45 Uhr, SR 301



0. Motivation

Worum geht es bei der Routenplanung?

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

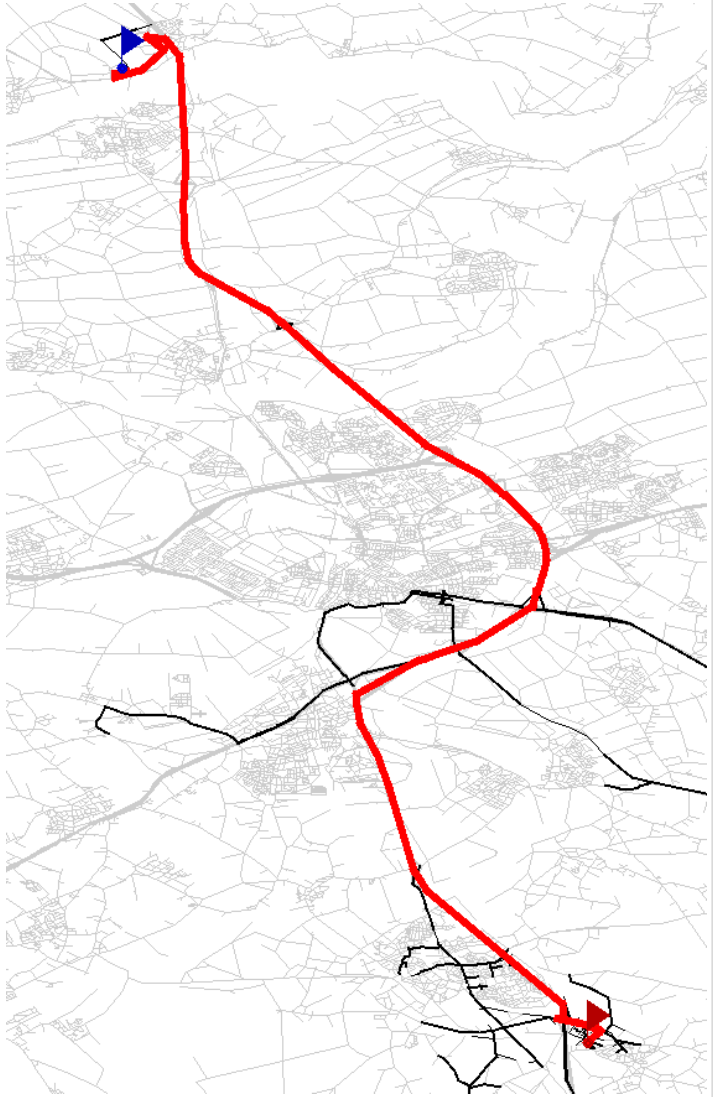
Beschleunigungstechniken

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: Google Maps

Unser Anspruch:

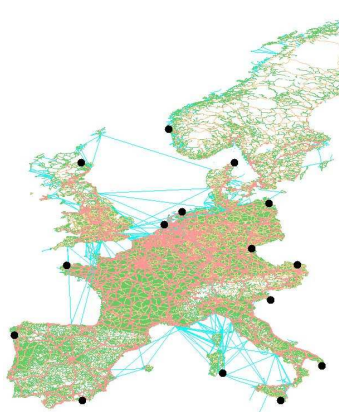
- Anfragen sollen beweisbar korrekt sein

$\Rightarrow$ 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
Highway Hierarchies	2005	0:13	48	0.61	8 448
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.

* Damalige Variante deutlich langsamer, wir sehen nachher, warum

Schwerpunkte

Vorlesung

- Algorithm Engineering + ein bisschen Theorie
- Beschleunigungstechniken
- Implementierungsdetails
- Ergebnisse auf Real-Welt Daten
- aktueller Stand der Forschung (Veröffentlichungen bis 2010)
- Ansatz übertragbar auf andere Themen (z. B. Flüsse)
- ideale Grundlage für Studien- und Diplomarbeiten

Übungen:

- Vertiefende theoretische Aufgaben
- Implementation von (kleinen!) Teilproblemen
- bestehendes Framework
- Straßendaten von Nordamerika
- Implementation hilft beim Verständnis
- aber nicht zwingend nötig für die Vorlesung und Prüfung

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
- hinter vielen Optimierungsproblemen stehen $\mathcal{NP}$-schwere Probleme

Inhalt der Vorlesung

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

3. Theorie

- Komplexität von Beschleunigungstechniken
- theoretische Charakterisierung von Straßennetzwerken
- Optimalität von Beschleunigungstechniken
- Straßengraphengeneratoren (evtl.)

4. Fortgeschrittene Szenarien

- zeitabhängige Routenplanung
- Alternativrouten
- multi-modale Routenplanung
- dynamische Szenarien (evtl.)

5. Offene Probleme

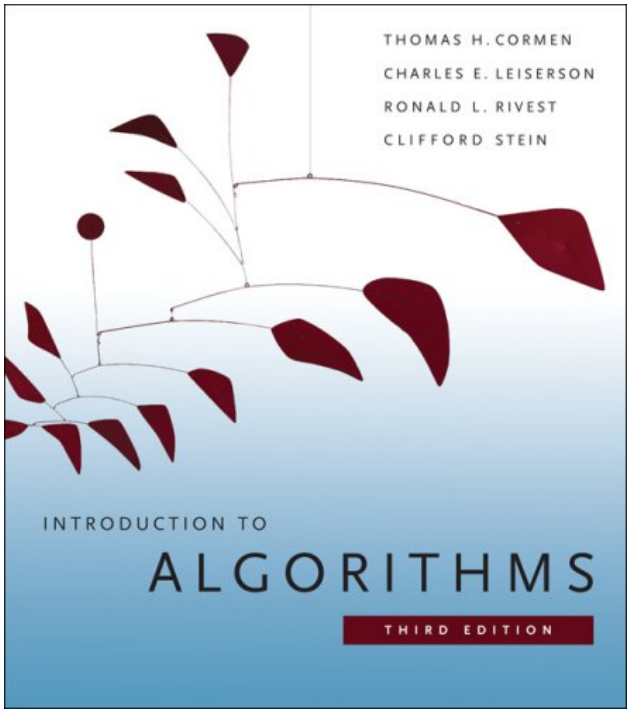
Nützliche Vorkenntnisse

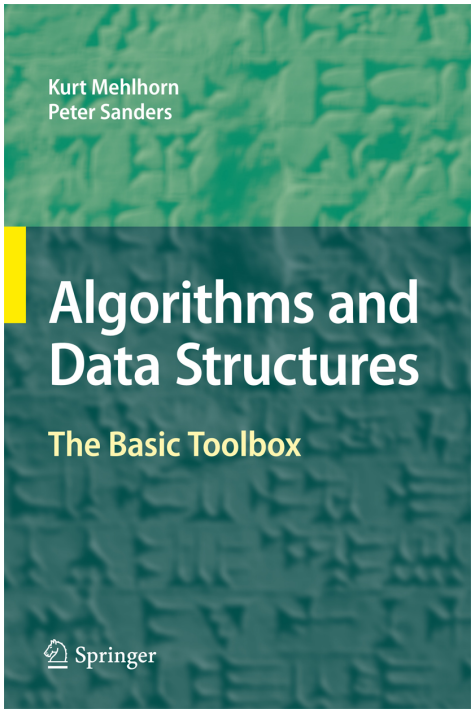
- Informatik I/II oder Algorithmen I
- Algorithmentechnik oder Algorithmen II (muss aber nicht sein)
- ein bisschen Rechnerarchitektur
- passive Kenntnisse von C++/Java

Vertiefungsgebiet: Algorithmentechnik, (Theoretische Grundlagen)

Material

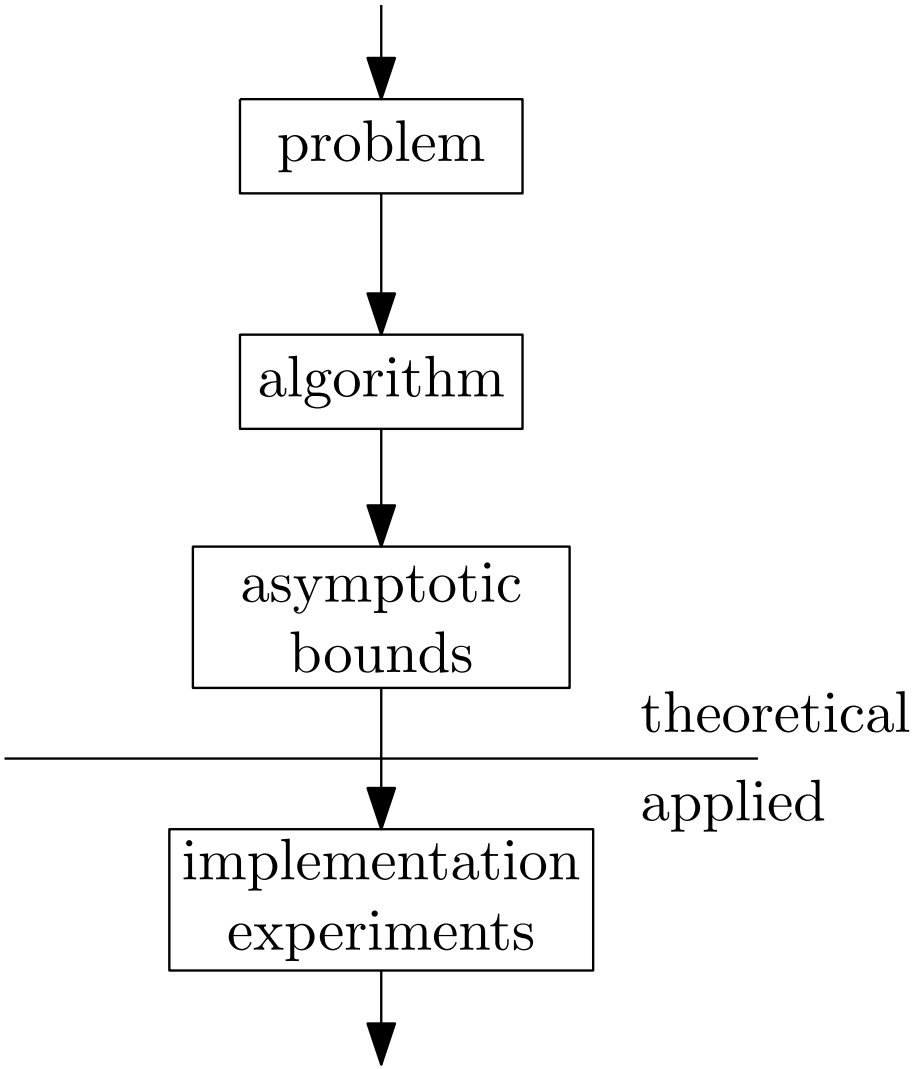
- Folien
- Übungsblätter
- wissenschaftliche Aufsätze (siehe Vorlesunghomepage)
- Basiskenntnisse:





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

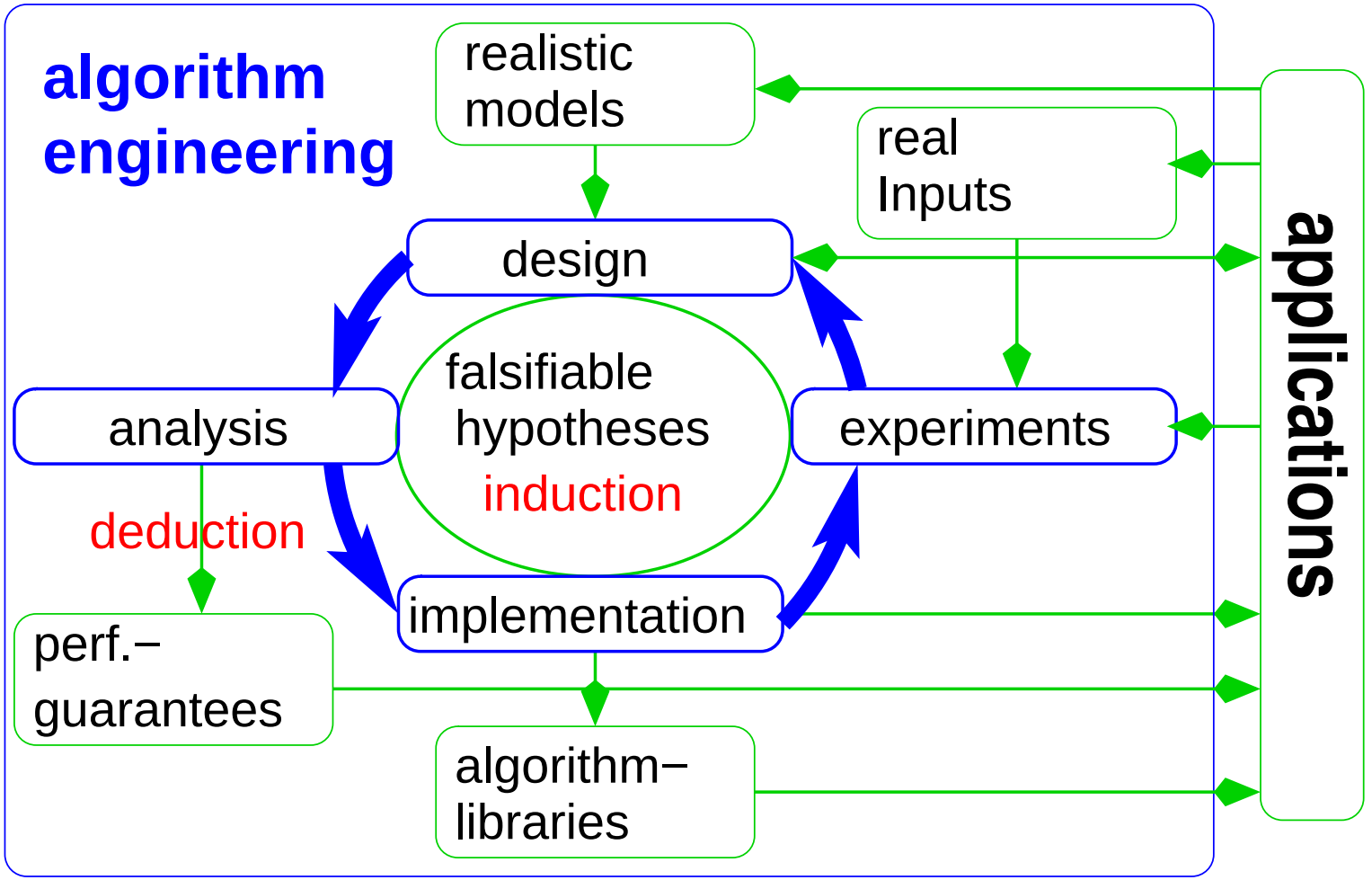
Lücke Theorie vs. Praxis

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

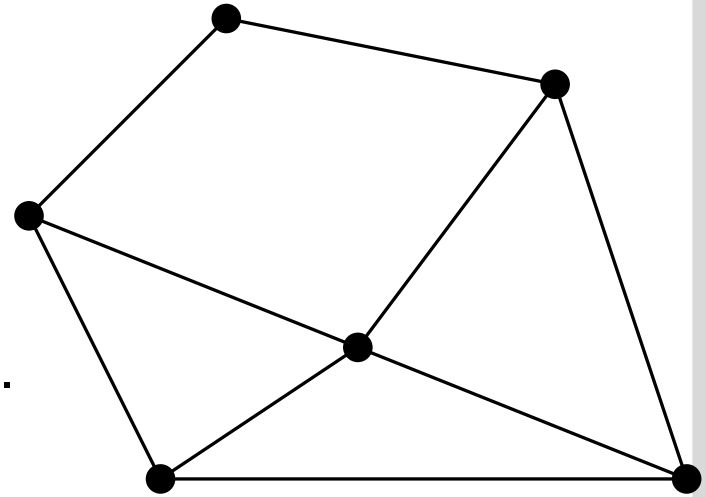
hier:

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

Algorithm Engineering

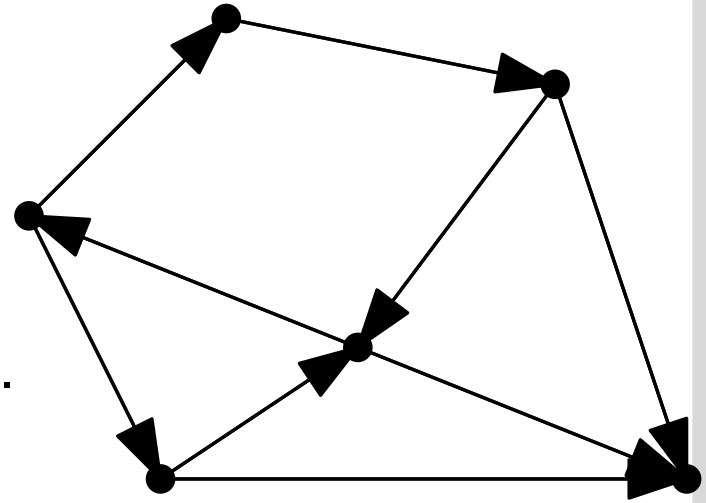


Grundbegriffe: Graphen



- **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)

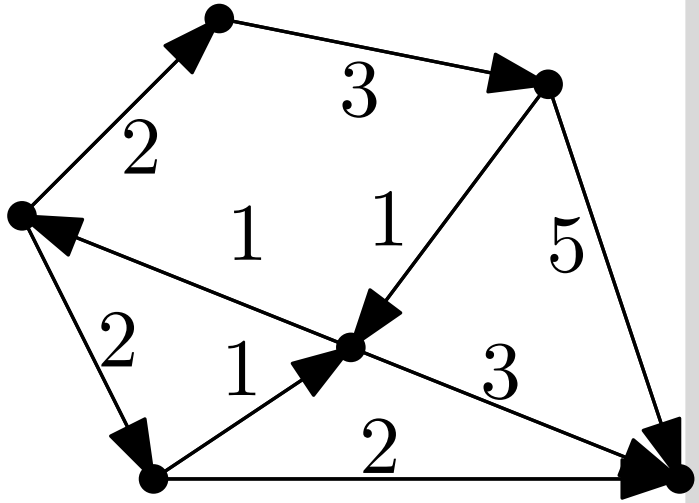
Grundbegriffe: Graphen



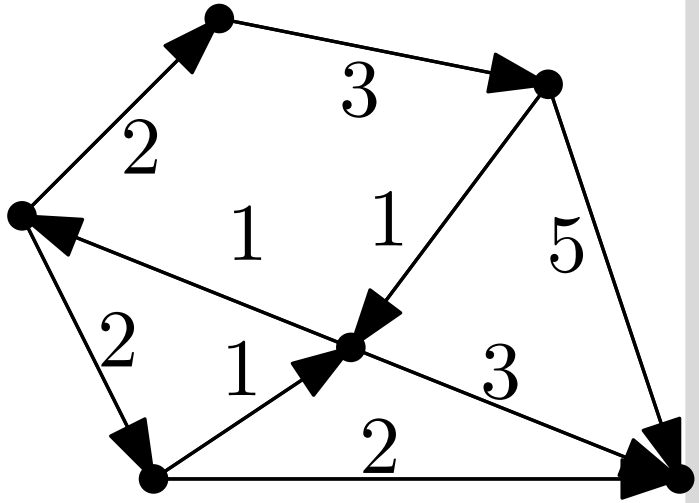
- **Graph:** Tupel $G := (V, E)$
  - endliche Knotenmenge V
  - endliche Kantenmenge E
  - $n := |V|$, $m := |E|$
- **ungerichtet:** Kanten sind Knotenpaar, d.h. $E \subseteq \binom{V}{2} = \{\{u, v\} \mid u \neq v,\ u, v \in V\}$
  - Grad: $\deg(u) = \sum_{\{u,v\} \in E}$
- **gerichtet:** Kanten sind 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

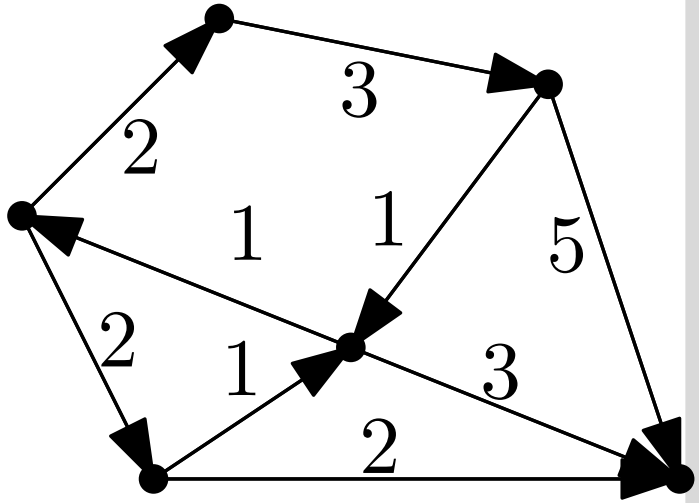


Grundbegriffe: Graphen



- **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

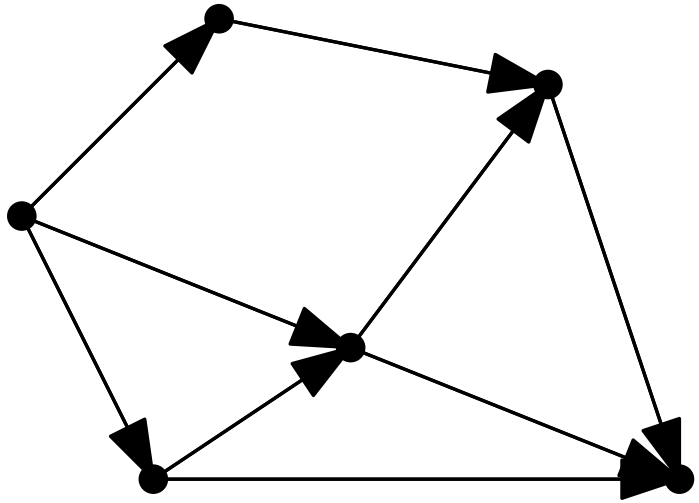
Spezielle Graphen

DAG:

- directed acyclic graph
- zyklenfrei

Baum:

- zyklenfrei
- $m = n - 1$
- also dünn



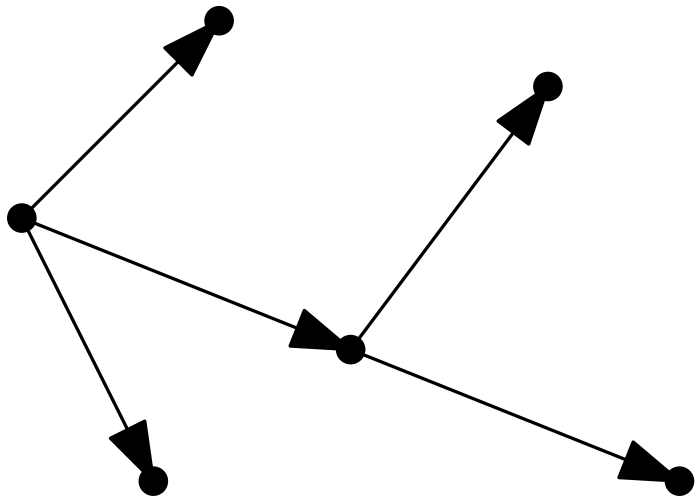
Spezielle Graphen

DAG:

- directed acyclic graph
- zyklenfrei

Baum:

- zyklenfrei
- $m = n - 1$
- also dünn



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

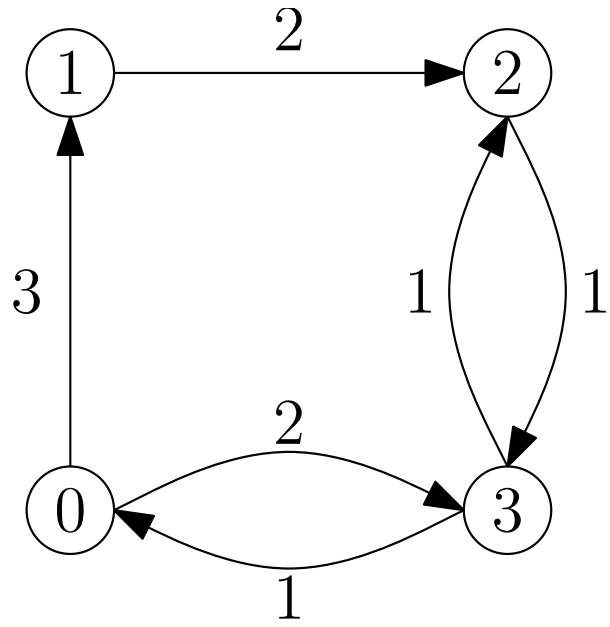
$$d(s,t) = \begin{cases} |\Pi(s,t)| & \text{wenn } \exists \text{ Weg von } s \text{ nach } t \text{ in } G \\ \infty & \text{sonst} \end{cases}$$

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

Graph-Repräsentationen

Drei klassische Ansätze:

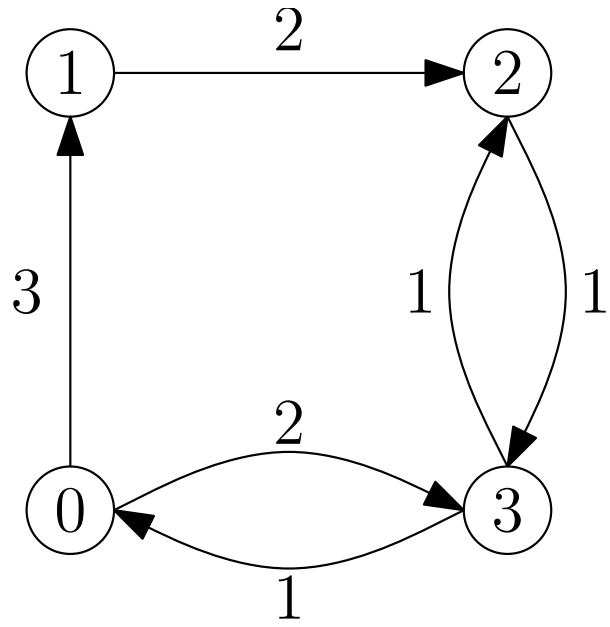
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

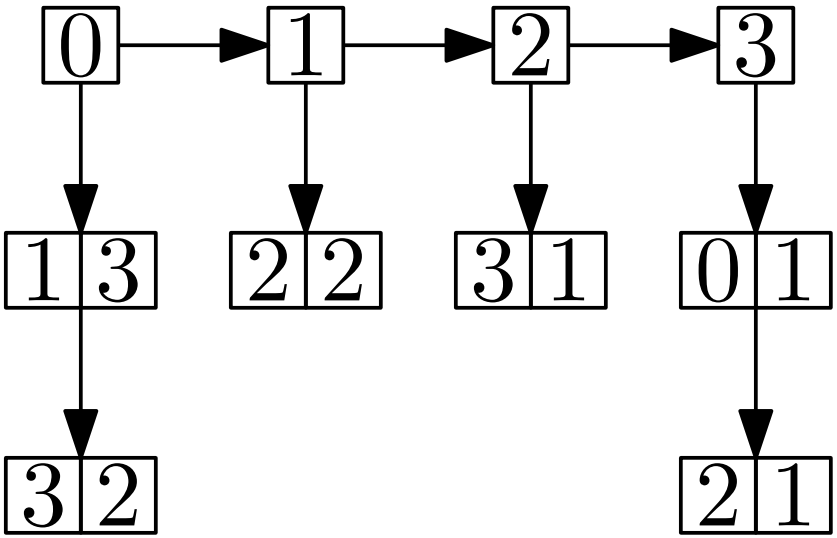


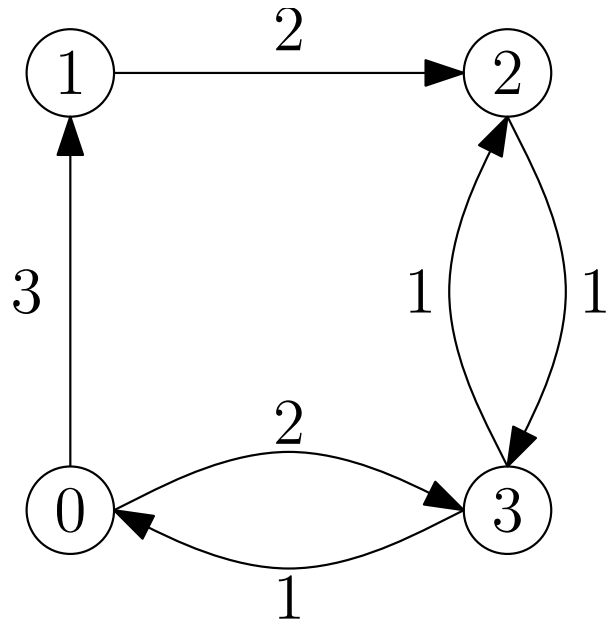
	0	1	2	3
0	–	3	–	2
1	–	–	2	–
2	–	–	–	1
3	1	–	1	–

Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

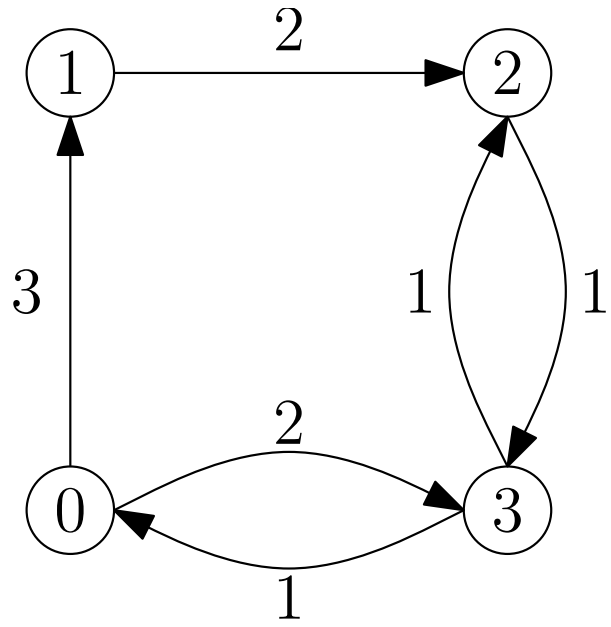


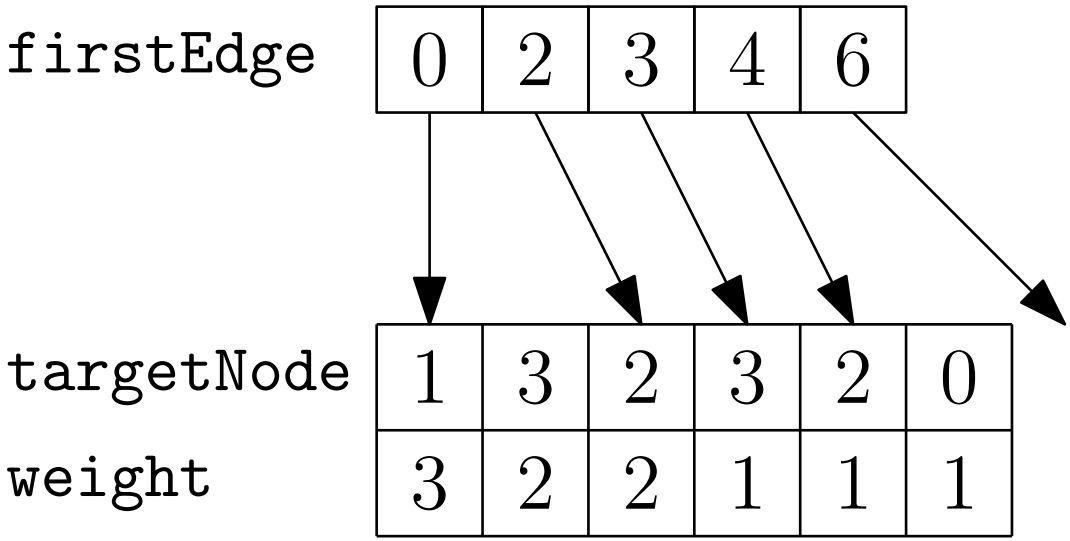


Graph-Repräsentationen

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	$+$	$-$	$+$
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	$+$	$-$	$+$
Updates (topologisch)	$+$	$+$	$-$
Updates (Gewicht)	$+$	$+$	$+$

Fragen:

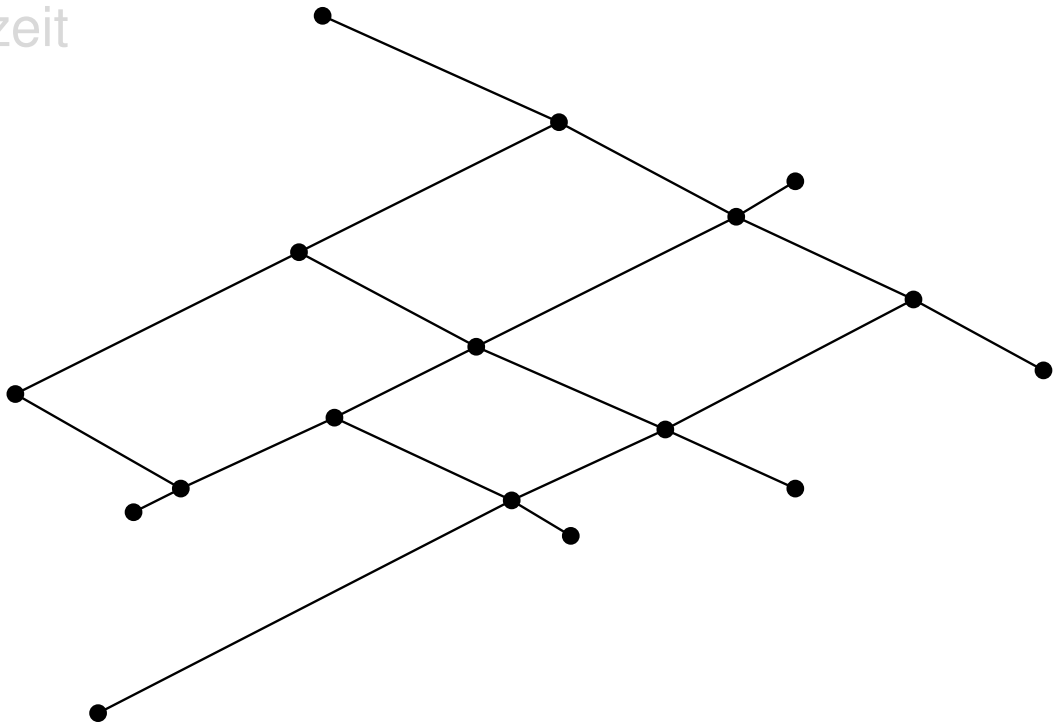
- Was brauchen wir?
- Was muss nich supereffizient sein?
- erstmal Modelle anschauen!

Modellierung (Straßengraphen)

Modellierung (Straßengraphen)

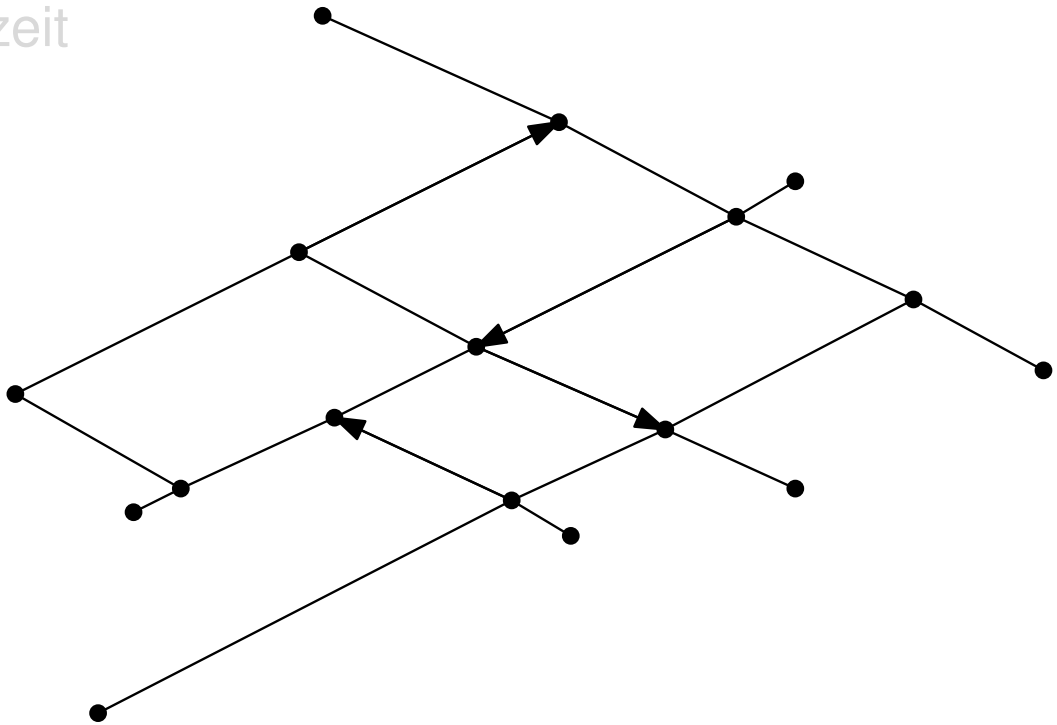
Modellierung (Straßengraphen)

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



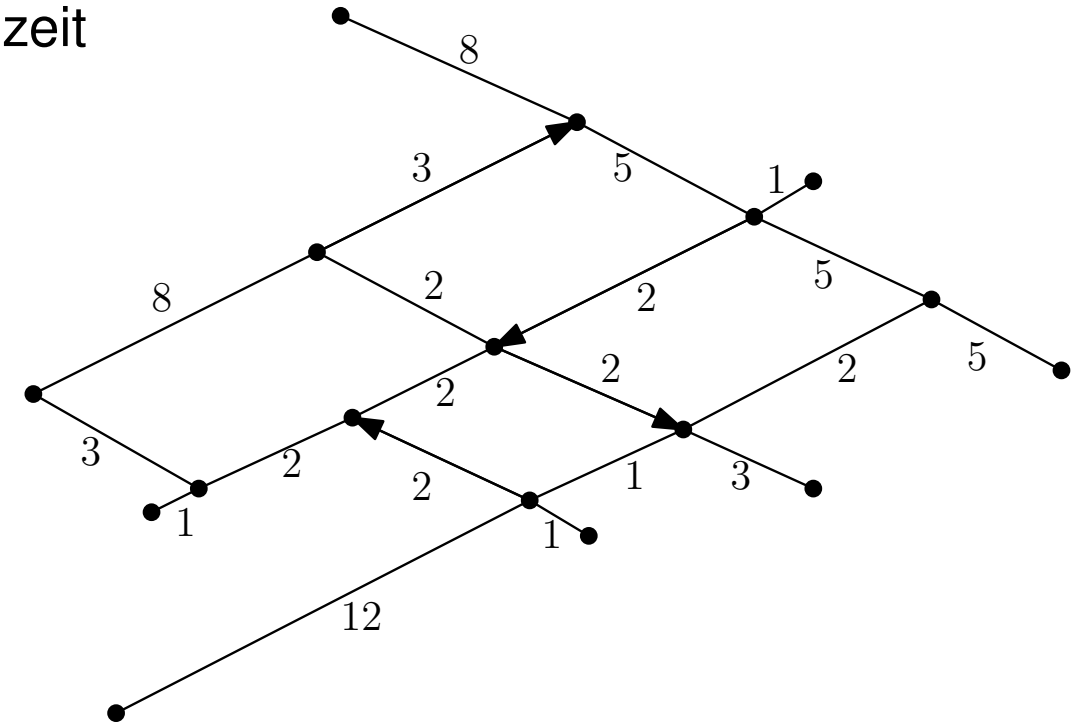
Modellierung (Straßengraphen)

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



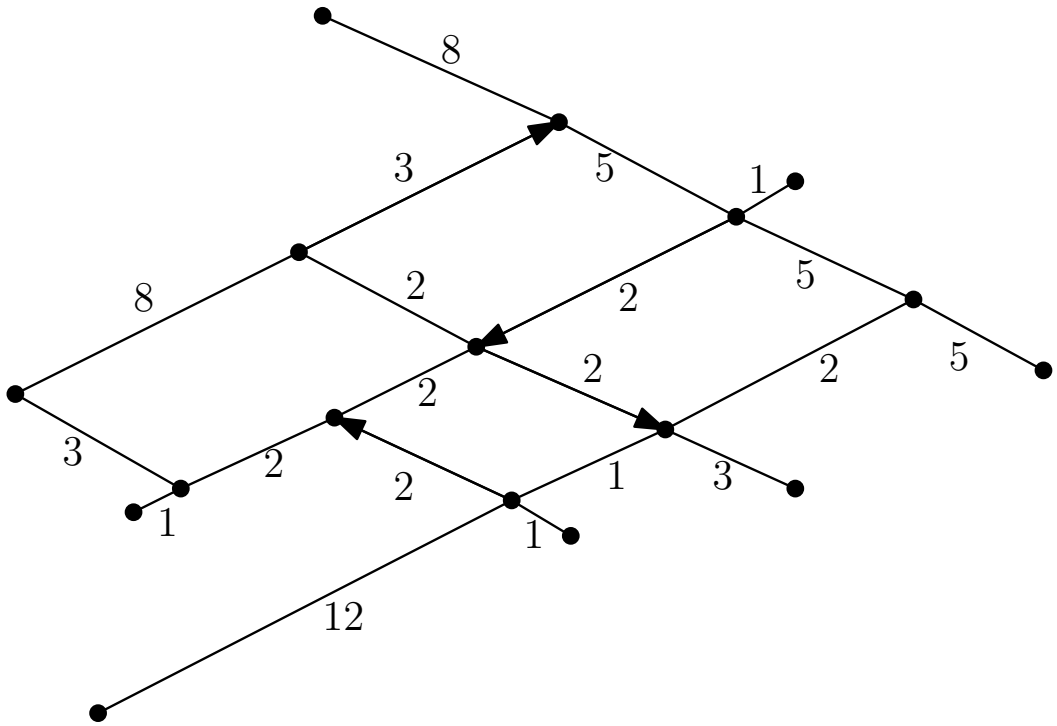
Modellierung (Straßengraphen)

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



Modellierung (Straßengraphen)

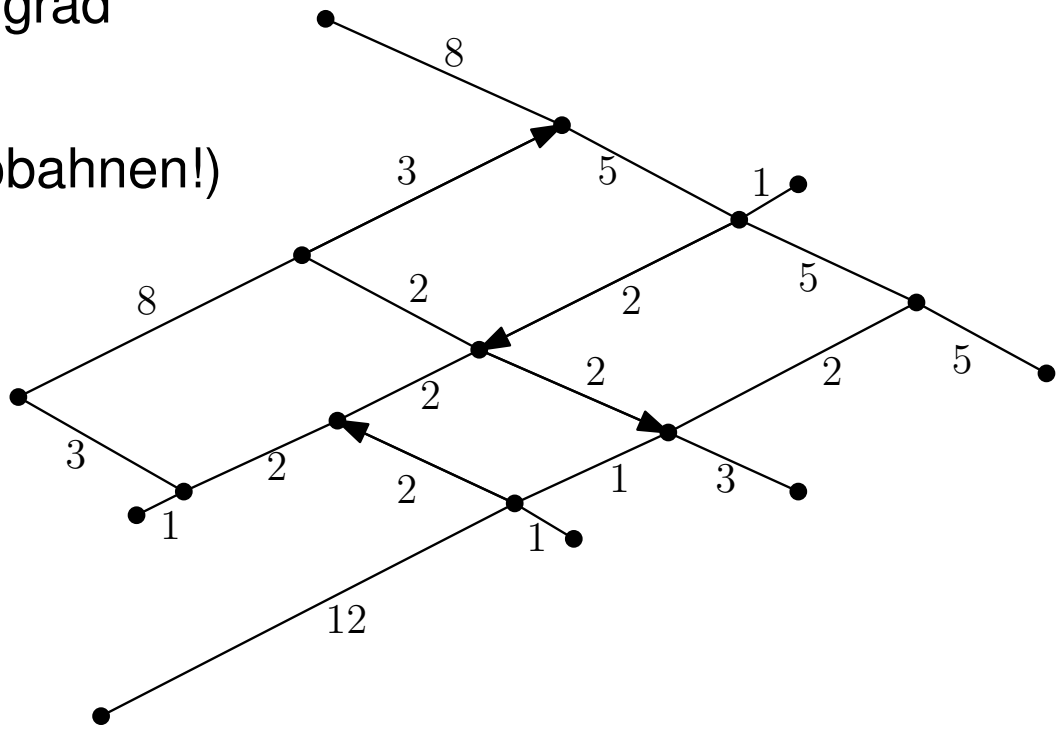
Eigenschaften (sammeln):



Modellierung (Straßengraphen)

Eigenschaften:

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



Modellierung (Eisenbahngraphen)

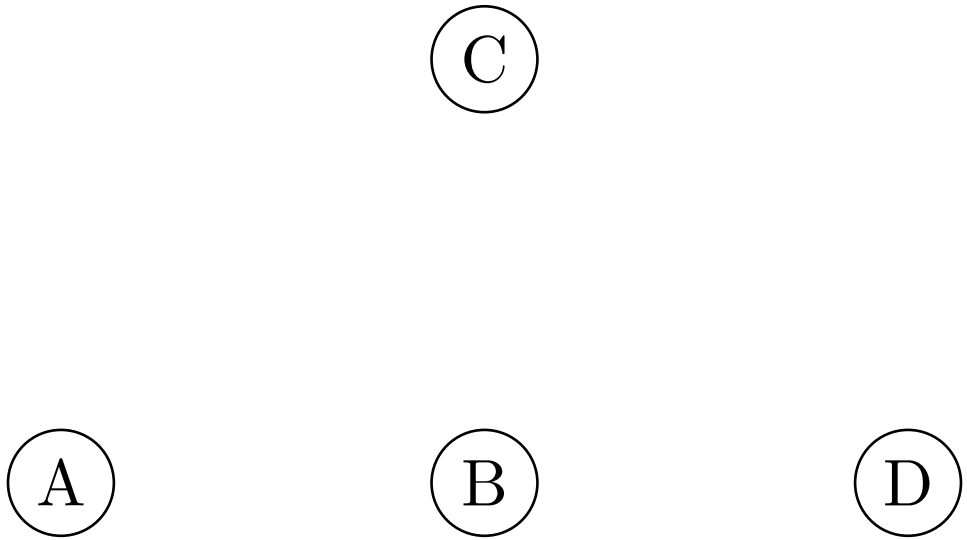
Eingabe:

- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren alle 10 Minuten

Modellierung (Eisenbahngraphen)

Eingabe:

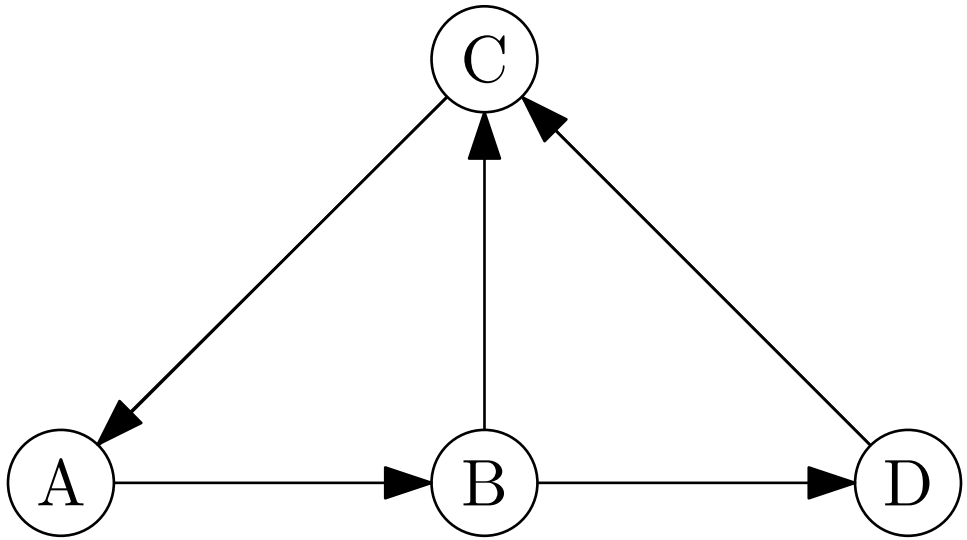
- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren alle 10 Minuten



Modellierung (Eisenbahngraphen)

Eingabe:

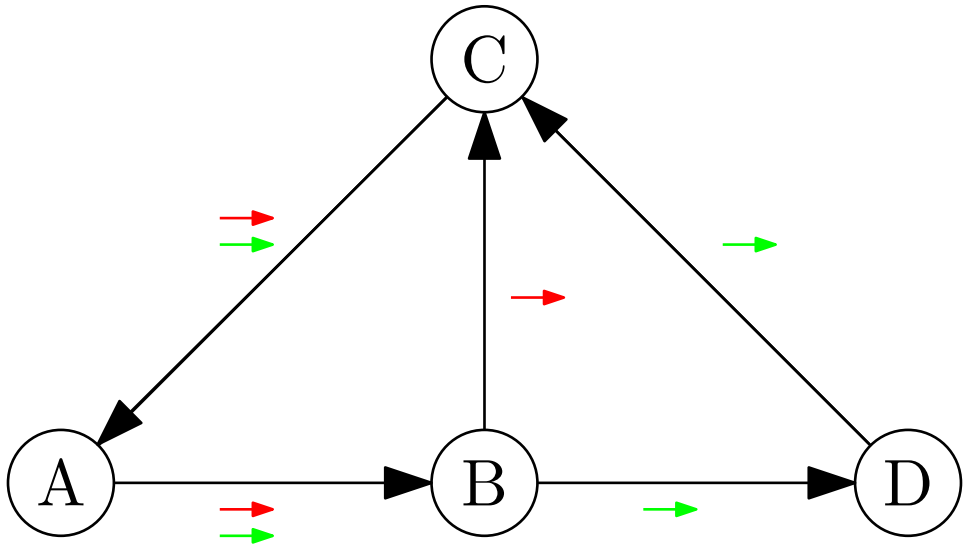
- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren alle 10 Minuten



Modellierung (Eisenbahngraphen)

Eingabe:

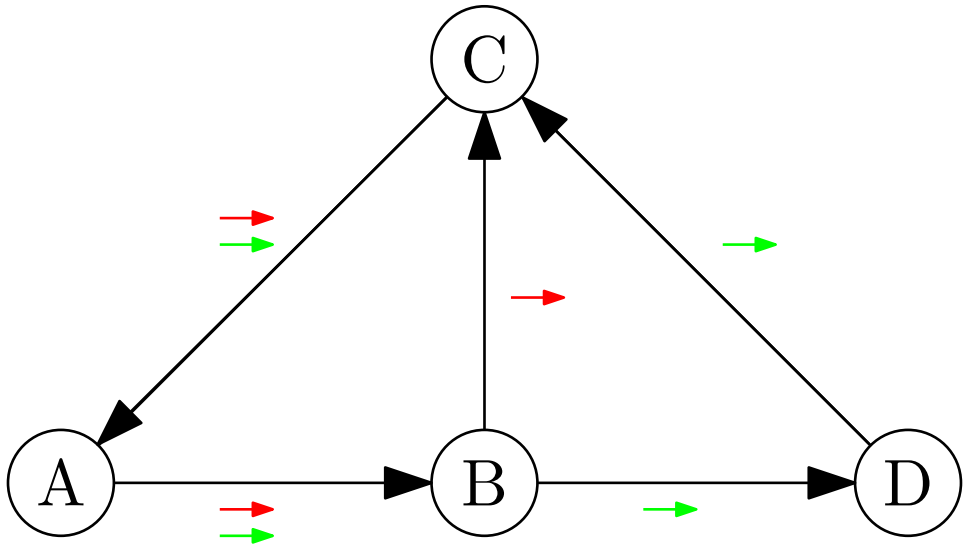
- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren alle 10 Minuten



Modellierung (Eisenbahngraphen)

Eingabe:

- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren alle 10 Minuten

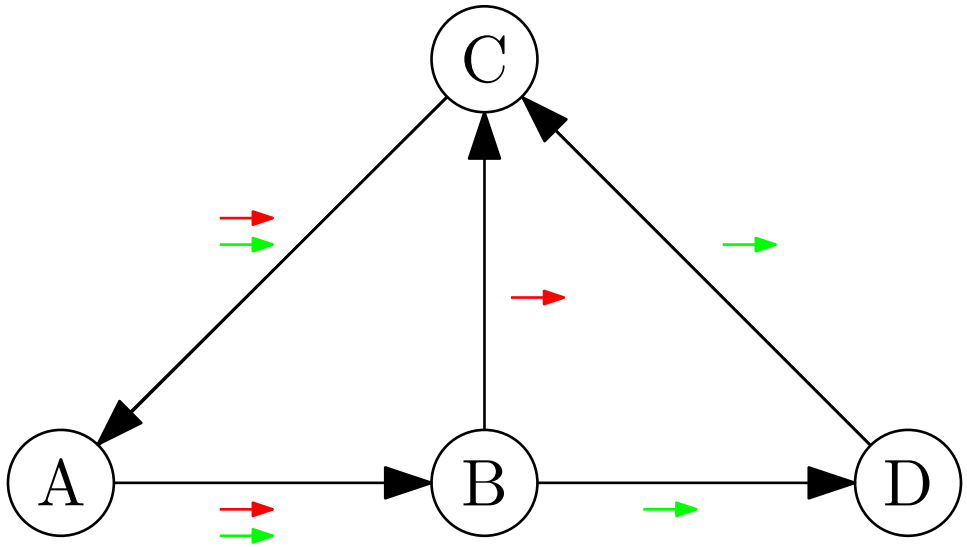


Kanten sind zeitabhängig!

Modellierung (Eisenbahngraphen)

Eingabe:

- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren alle 10 Minuten



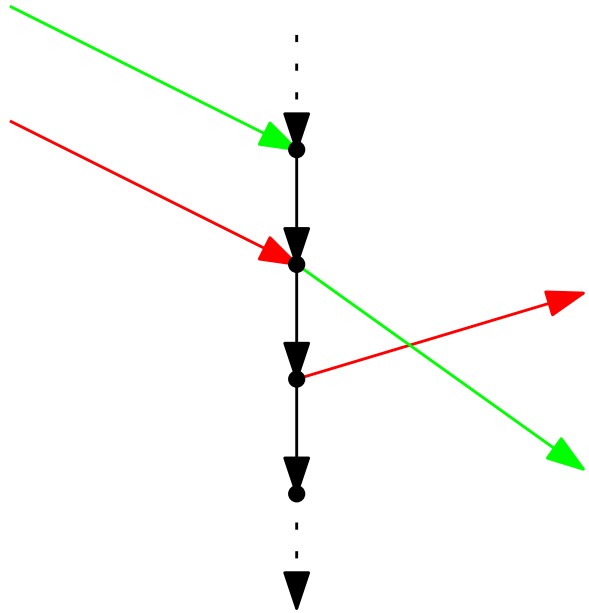
Kanten sind zeitabhängig! oder roll die Zeit aus

Zeitexpandiertes Modell

Vorgehen:

- Knoten sind “Events”
- Kanten Elementarverbindungen
- Wartekanten an den Stationen

Station B:



Diskussion:

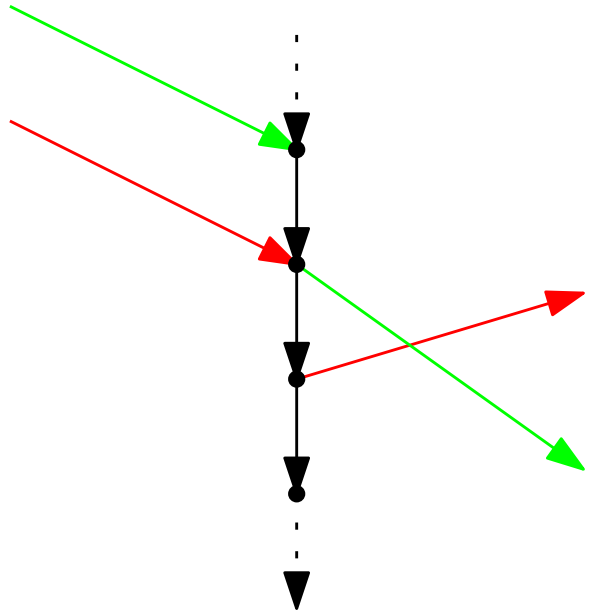
+ keine zeitabhängigen Kanten
– Graph größer

Zeitexpandiertes Modell

Vorgehen:

- Knoten sind “Events”
- Kanten Elementarverbindungen
- Wartekanten an den Stationen

Station B:

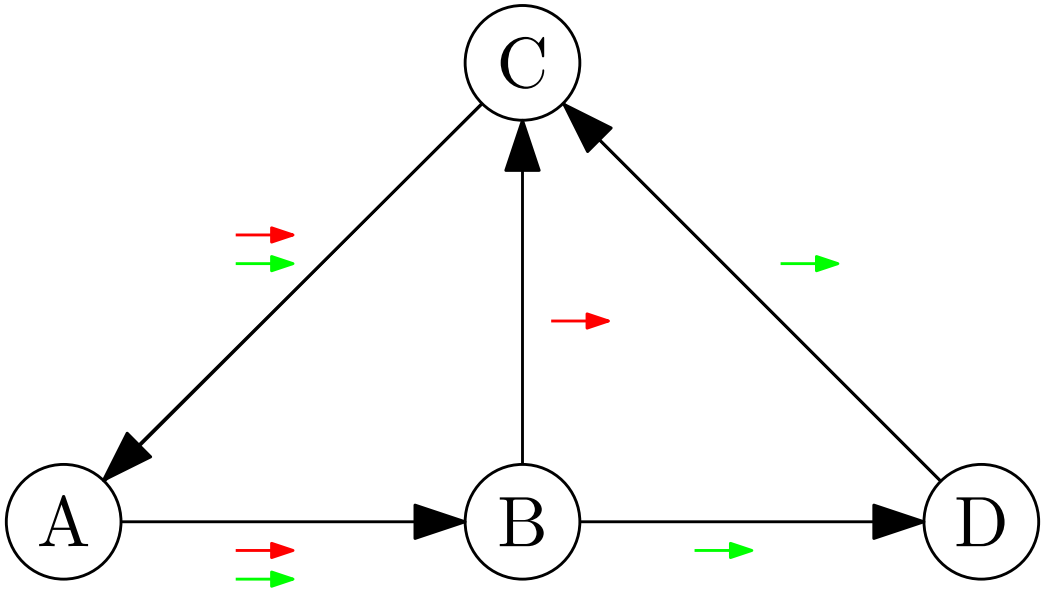


Diskussion:

+ keine zeitabhängigen Kanten
– Graph größer

Problem der Modellierung

Problem:

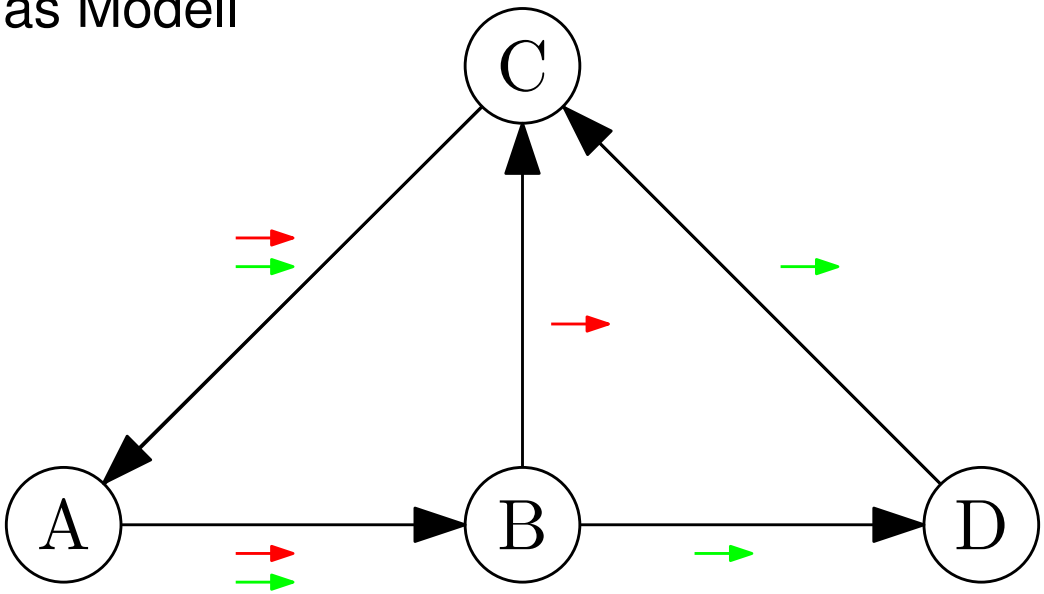


Problem der Modellierung

Problem:

- Umstiegszeiten?

$\Rightarrow$ erweitere das Modell

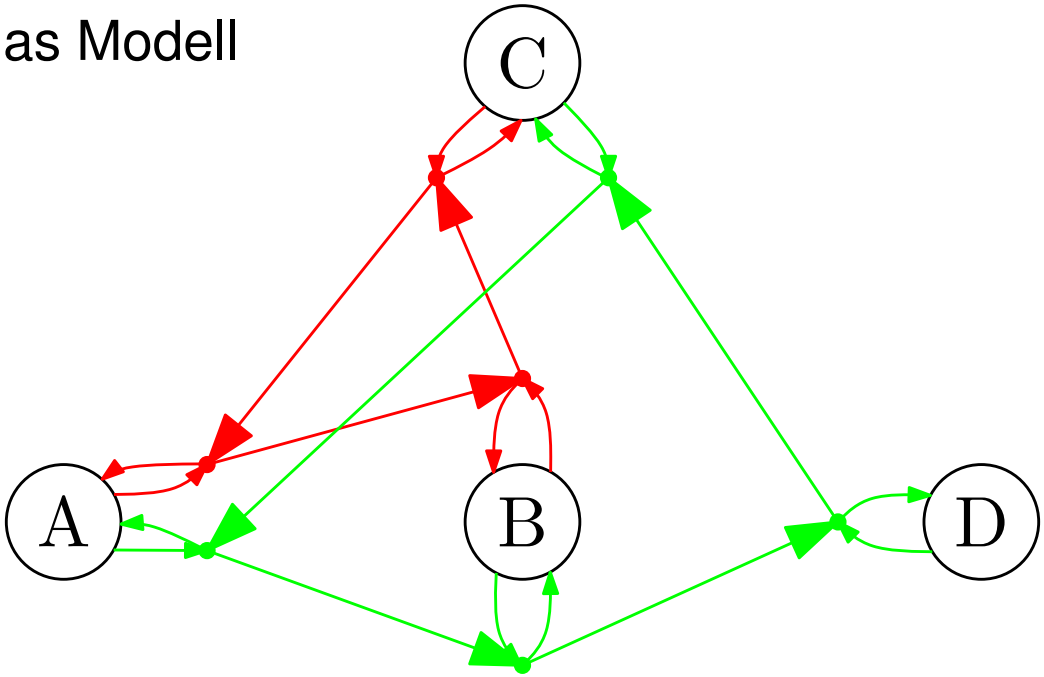


Problem der Modellierung

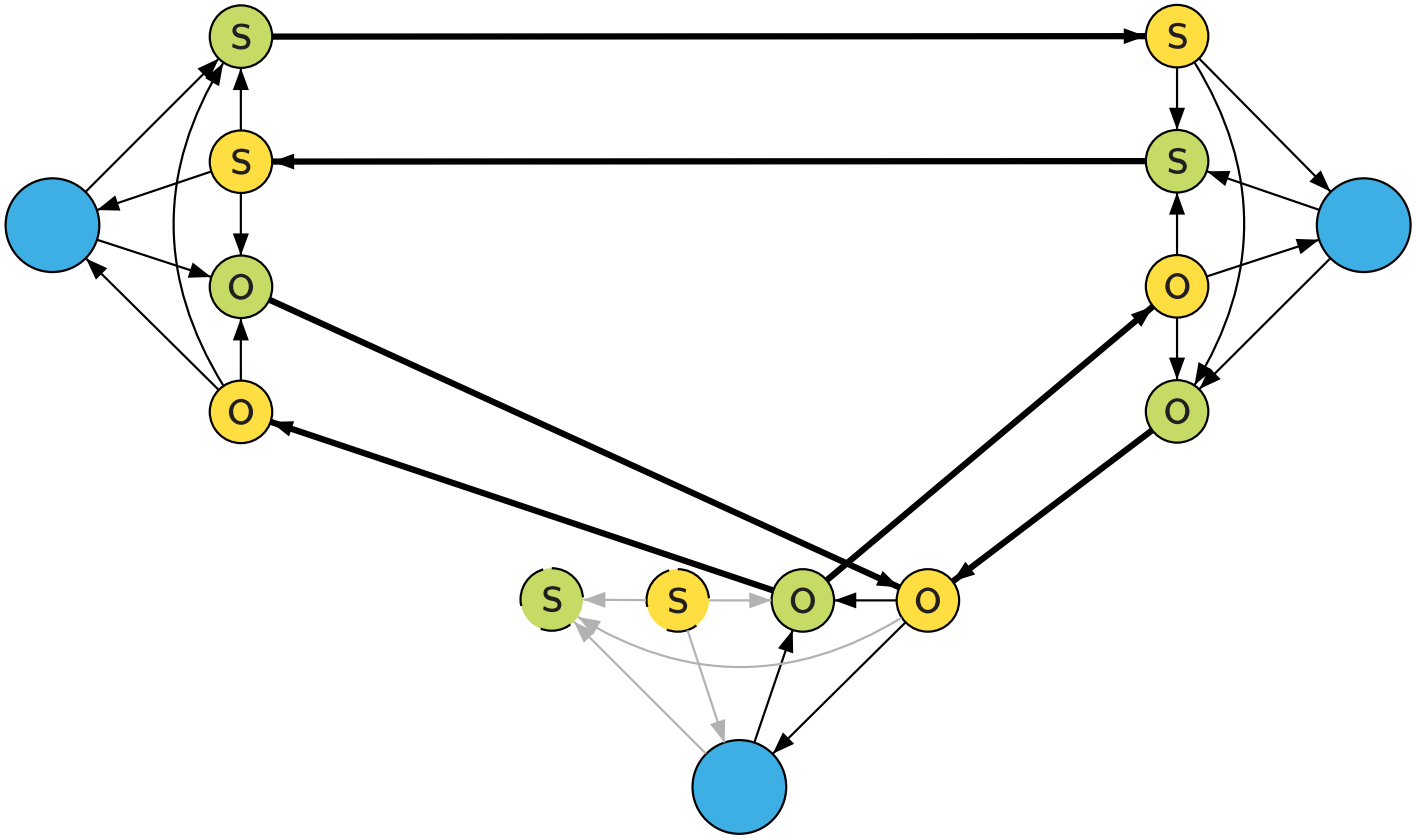
Problem:

- Umstiegszeiten?

$\Rightarrow$ erweitere das Modell



Modellierung (Flugdaten)



Eigenschaften der Graphen

- dünn (!)
- gerichtet
- geringer Knotengrad
- meist verborgene Hierarchie (Autobahnen, ICE, Langstreckenflüge)
- 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, Langstreckenflüge)
- 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

Eigenschaften auf dünnen Graphen:

	Matrix	Liste	Array
Speicher	$\mathcal{O}(n^2)$	$\mathcal{O}(n+m)$ $= \mathcal{O}(n)$	$\mathcal{O}(n+m)$ $= \mathcal{O}(n)$
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	$+$	$-$	$+$
Updates (topologisch)	$+$	$+$	$-$
Updates (Gewicht)	$+$	$+$	$+$

Also: Adjazenzarray!

Datenstruktur

Eigenschaften auf dünnen Graphen:

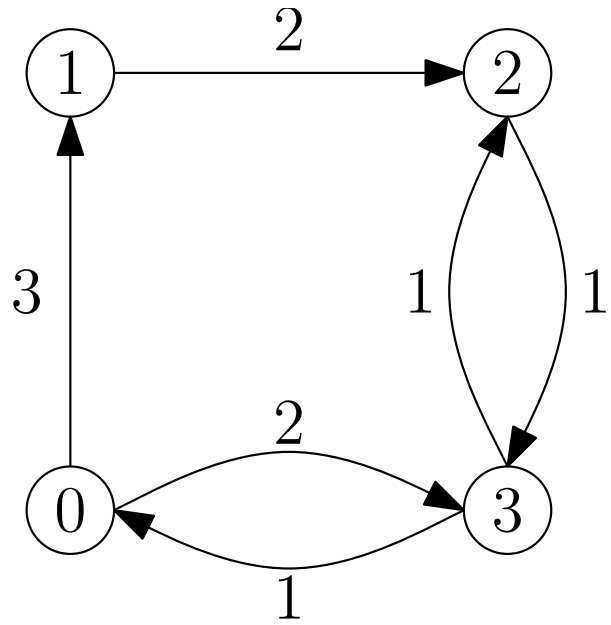
	Matrix	Liste	Array
Speicher	$\mathcal{O}(n^2)$	$\mathcal{O}(n+m)$ $= \mathcal{O}(n)$	$\mathcal{O}(n+m)$ $= \mathcal{O}(n)$
Ausgehende Kanten iterieren	$\mathcal{O}(n)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Kantenzugriff (u, v)	$\mathcal{O}(1)$	$\mathcal{O}(\deg u)$	$\mathcal{O}(\deg u)$
Effizienz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

Also: Adjazenzarray!

Datenstruktur

Problem:

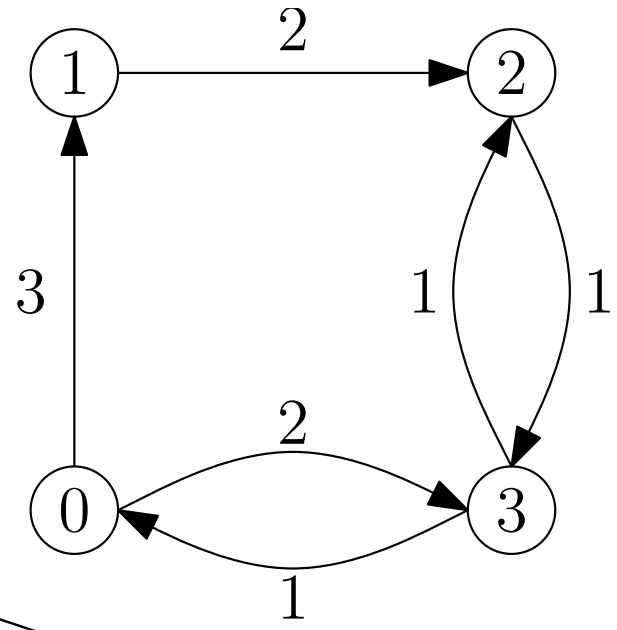
- ein- und ausgehende Kanten
- (fast) ungerichtet



Datenstruktur

Problem:

- ein- und ausgehende Kanten
- (fast) ungerichtet

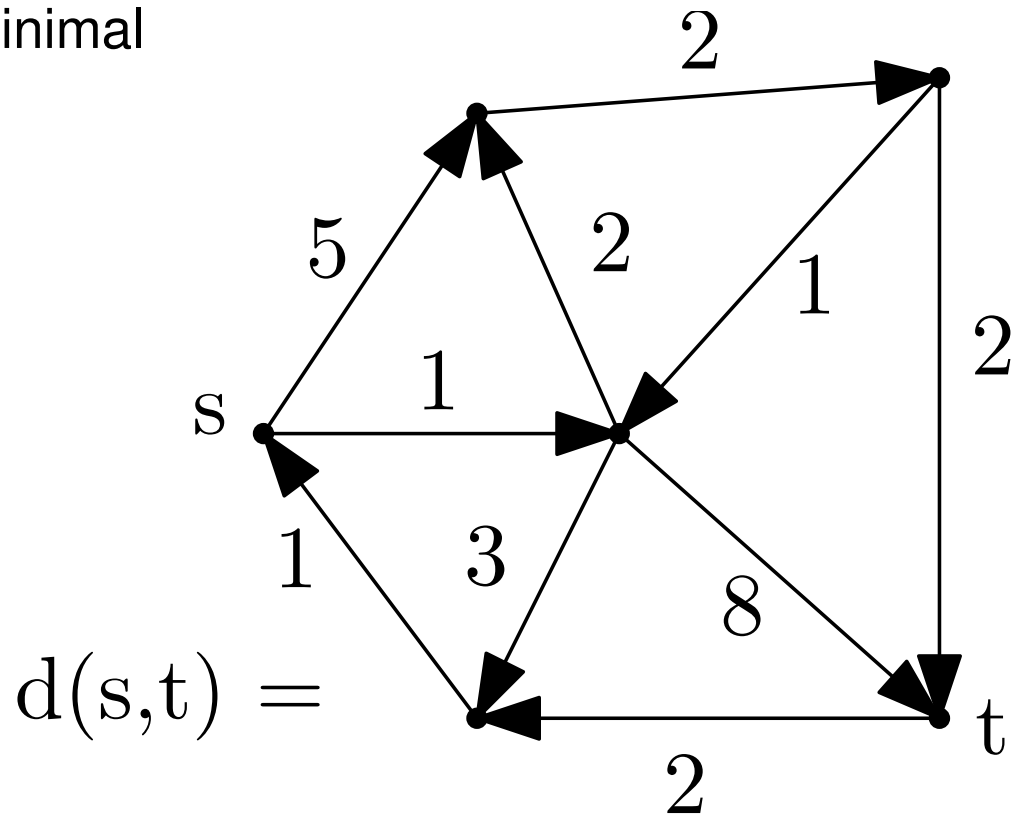


`firstEdge`	0	3	5	7	10					
`targetNode`	1	3	3	0	2	1	3	0	0	2
`weight`	3	2	1	3	2	2	1	1	2	1
`isIncoming`	–	–	√	√	–	√	√	–	√	√
`isOutgoing`	√	√	–	–	√	–	√	√	–	√

Kürzeste Wege

Gegeben:
Graph $G = (V, E)$ mit positiver Kantenfunktion
len : $E \to \mathbb{R}^+$, zwei Knoten s und t

Finde:
Pfad $P := (s, \ldots, t)$ mit $d(s, t)$ minimal

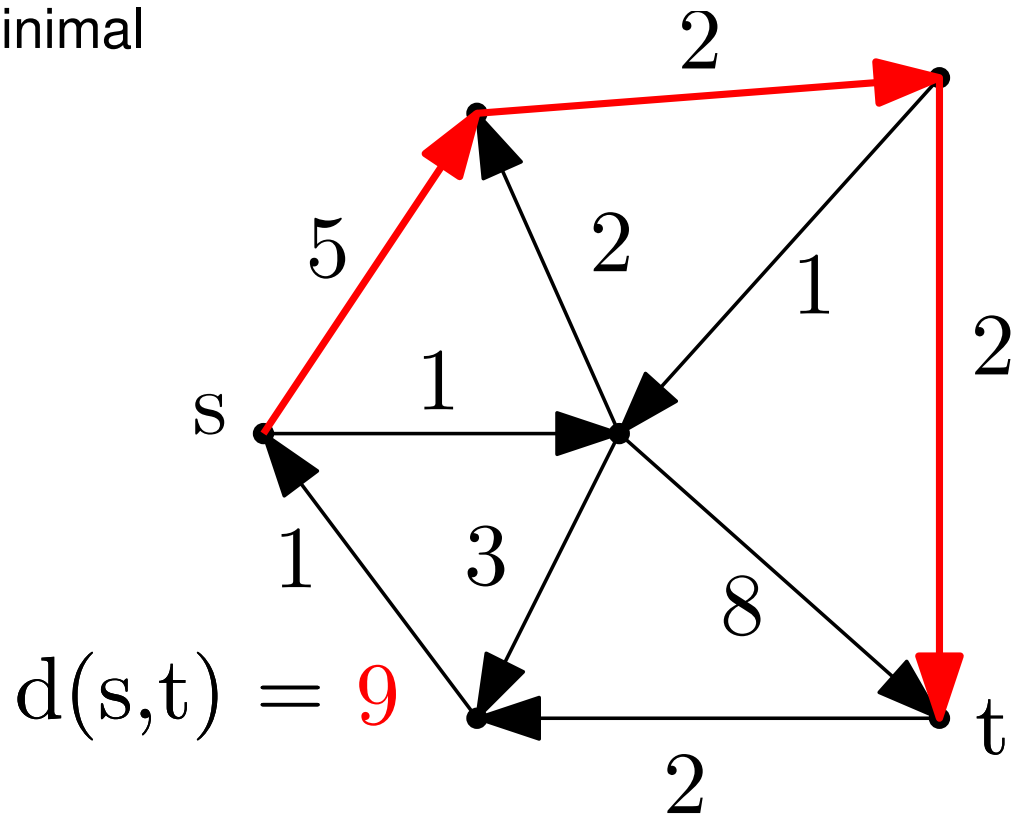


$\mathrm{d(s,t)} =$

Kürzeste Wege

Gegeben:
Graph $G = (V, E)$ mit positiver Kantenfunktion
len : $E \to \mathbb{R}^+$, zwei Knoten s und t

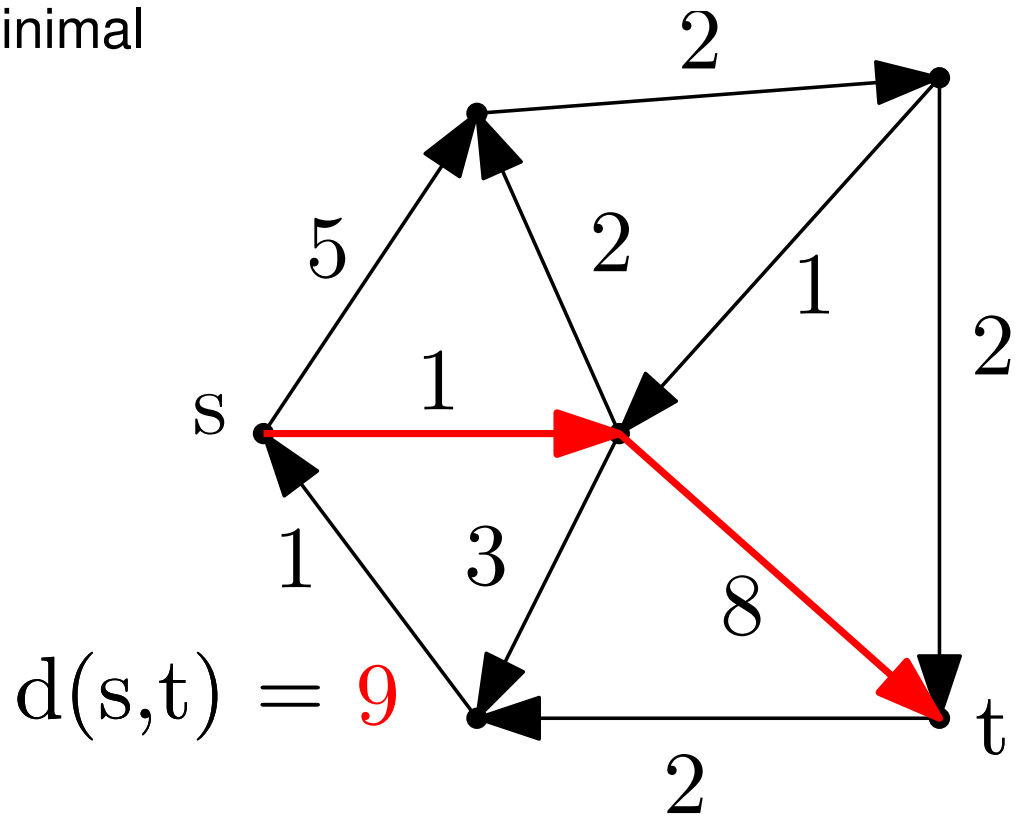
Finde:
Pfad $P := (s, \ldots, t)$ mit $d(s, t)$ minimal



Kürzeste Wege

Gegeben:
Graph $G = (V, E)$ mit positiver Kantenfunktion
len : $E \to \mathbb{R}^+$, zwei Knoten s und t

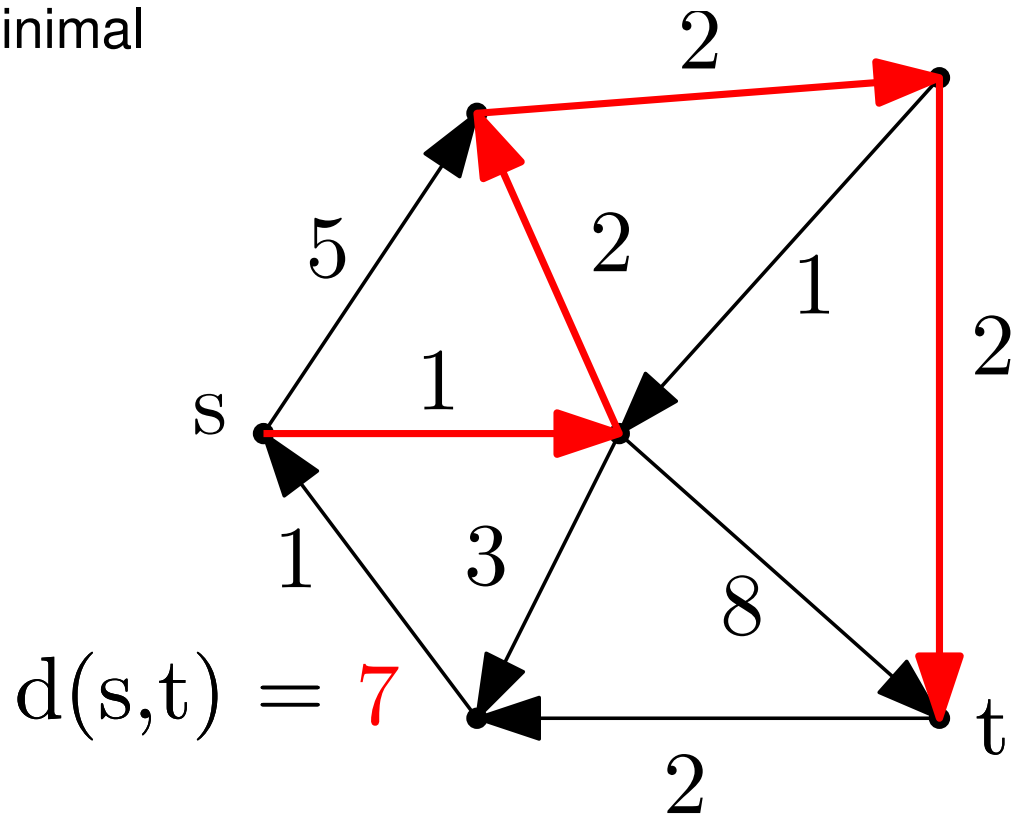
Finde:
Pfad $P := (s, \ldots, t)$ mit $d(s, t)$ minimal



Kürzeste Wege

Gegeben:
Graph $G = (V, E)$ mit positiver Kantenfunktion
len : $E \to \mathbb{R}^+$, zwei Knoten s und t

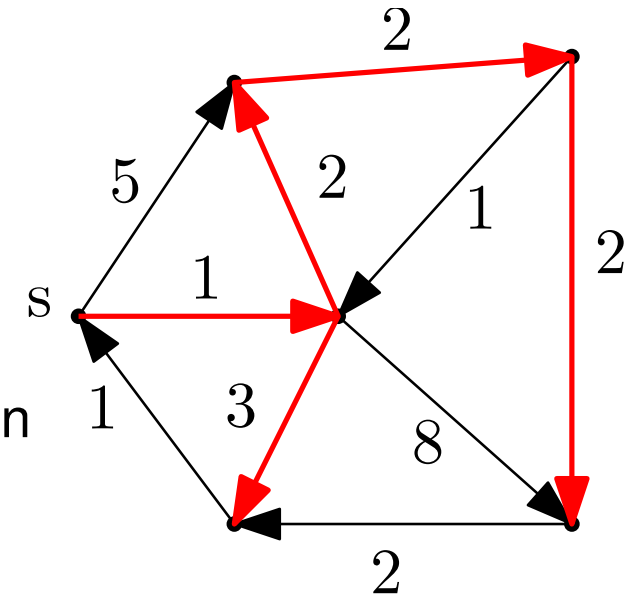
Finde:
Pfad $P := (s, \ldots, t)$ mit $d(s, t)$ minimal



Eigenschaften von kürzesten Wegen

Beobachtungen:

- kürzeste Wege sind zyklenfrei
- Subpfade sind auch kürzeste Wege
- alle kürzesten Wege von s aus spannen einen DAG bzw. Baum auf
- steigender Abstand von Wurzel zu Blättern
- $d(u, s)$ analog über “eingehende” Kanten
- können mit Dijkstra berechnet werden



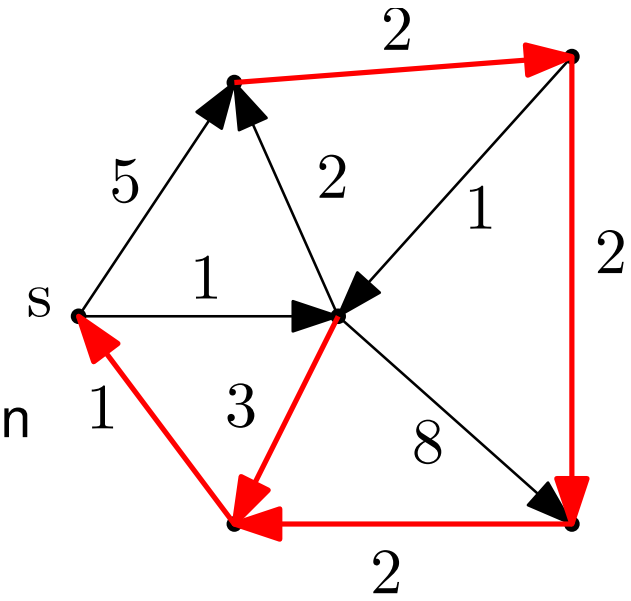
Anmerkungen:

- längste Wege: $\mathcal{NP}$-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: $\mathcal{NP}$-schwer

Eigenschaften von kürzesten Wegen

Beobachtungen:

- kürzeste Wege sind zyklenfrei
- Subpfade sind auch kürzeste Wege
- alle kürzesten Wege von s aus spannen einen DAG bzw. Baum auf
- steigender Abstand von Wurzel zu Blättern
- $d(u, s)$ analog über “eingehende” Kanten
- können mit Dijkstra berechnet werden



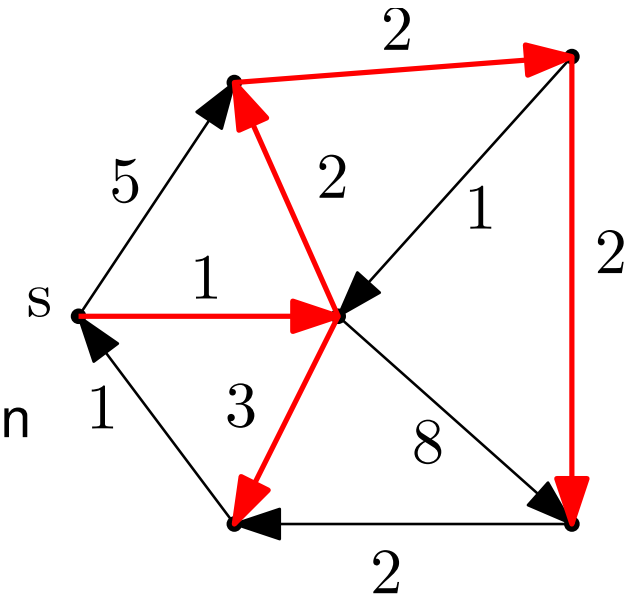
Anmerkungen:

- längste Wege: $\mathcal{NP}$-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: $\mathcal{NP}$-schwer

Eigenschaften von kürzesten Wegen

Beobachtungen:

- kürzeste Wege sind zyklenfrei
- Subpfade sind auch kürzeste Wege
- alle kürzesten Wege von s aus spannen einen DAG bzw. Baum auf
- steigender Abstand von Wurzel zu Blättern
- $d(u, s)$ analog über “eingehende” Kanten
- können mit Dijkstra berechnet werden



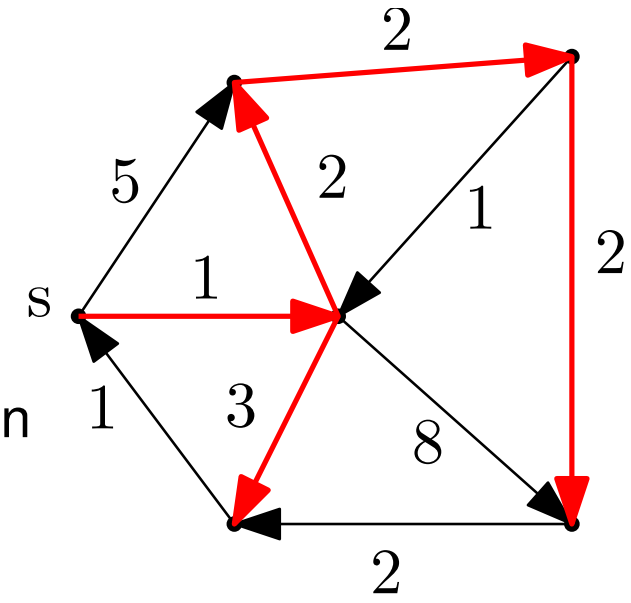
Anmerkungen:

- längste Wege: $\mathcal{NP}$-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: $\mathcal{NP}$-schwer

Eigenschaften von kürzesten Wegen

Beobachtungen:

- kürzeste Wege sind zyklenfrei
- Subpfade sind auch kürzeste Wege
- alle kürzesten Wege von s aus spannen einen DAG bzw. Baum auf
- steigender Abstand von Wurzel zu Blättern
- $d(u, s)$ analog über “eingehende” Kanten
- können mit Dijkstra berechnet werden



Anmerkungen:

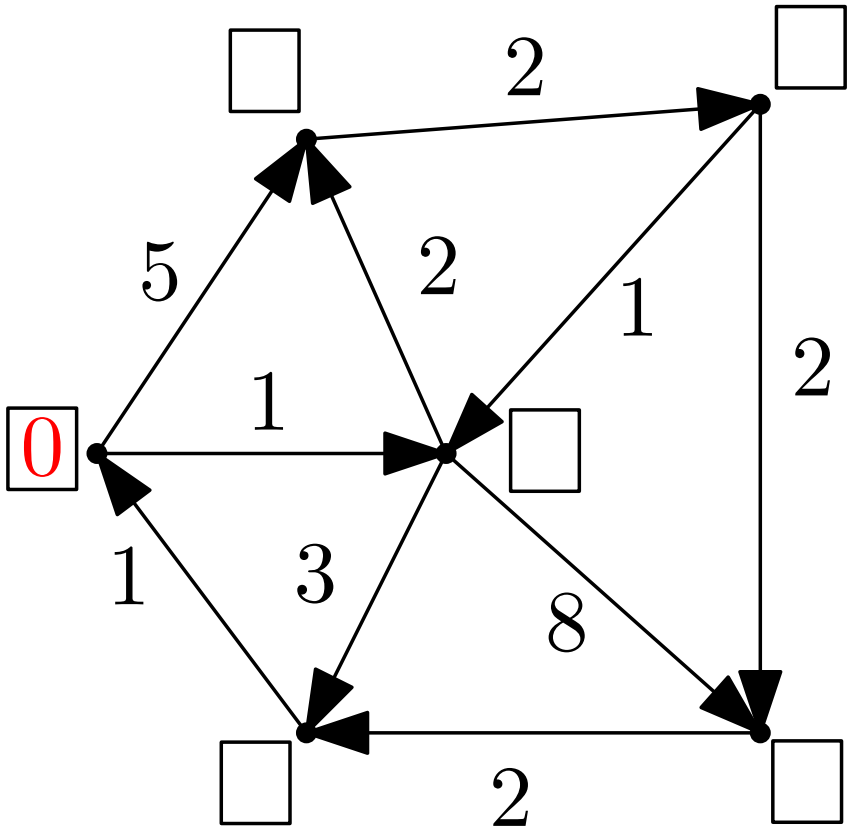
- längste Wege: $\mathcal{NP}$-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: $\mathcal{NP}$-schwer

Dijkstra

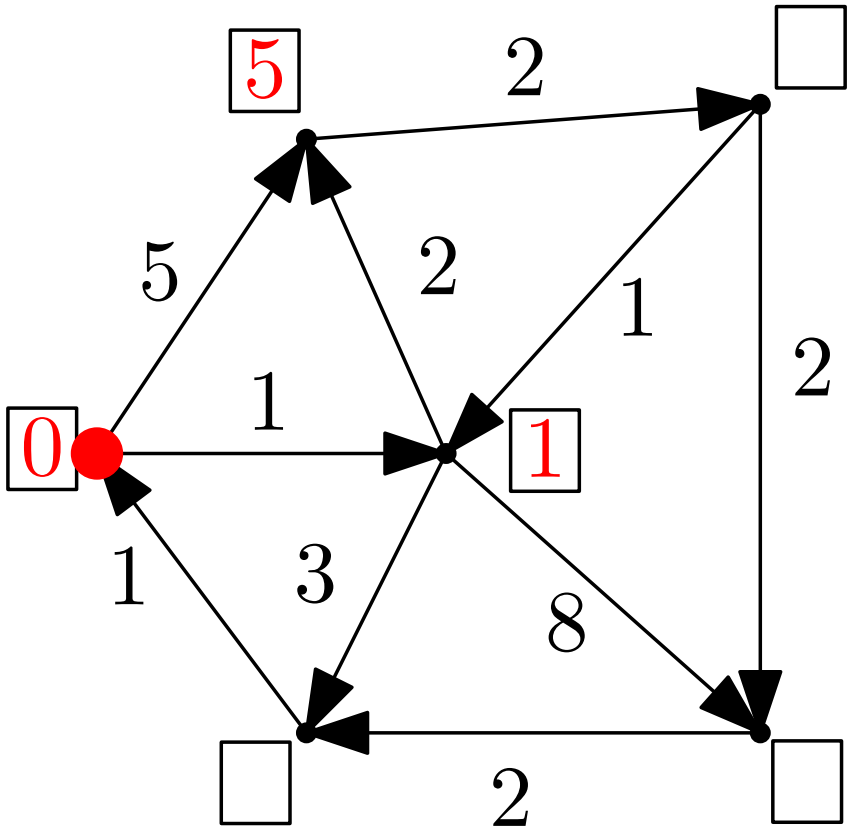
DIJKSTRA($G = (V, E)$, s)

```
1  forall nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL ;                    // distances, parents
3  d[s] = 0
4  Q.clear(), Q.add(s, 0);                        // container
5  while !Q.empty() do
6      u ← Q.deleteMin();                         // settling node u
7      forall edges e = (u, v) ∈ E do
           // relaxing edges
8          if d[u] + len(e) < d[v] then
9              d[v] ← d[u] + len(e)
10             p[v] ← u
11             if v ∈ Q then Q.decreaseKey(v, d[v])
12             else Q.insert(v, d[v])
```

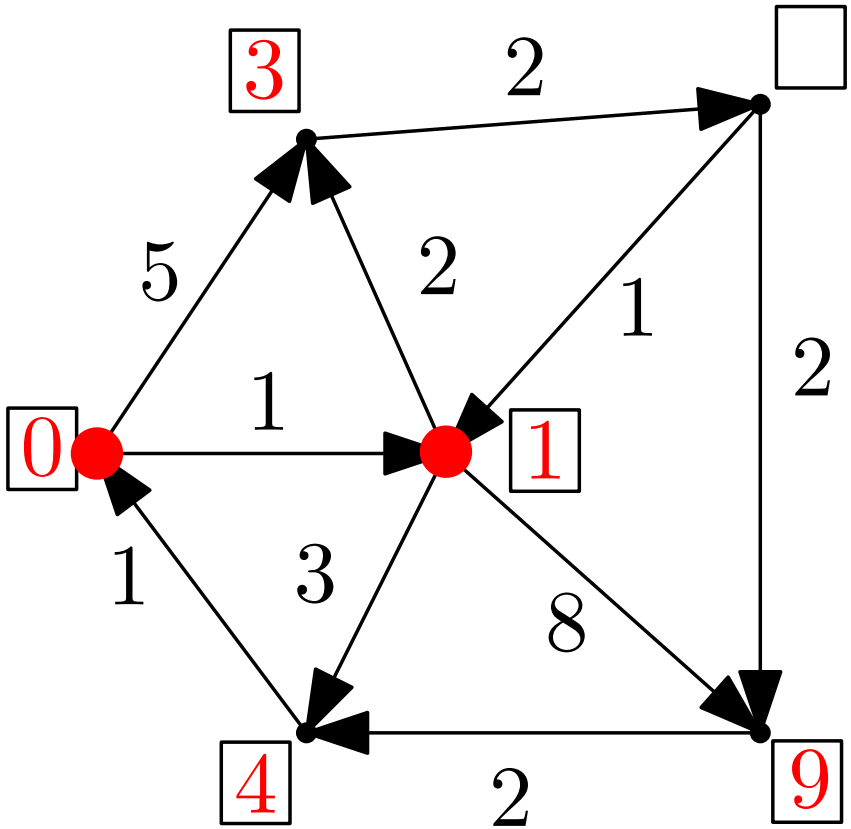
Beispiel



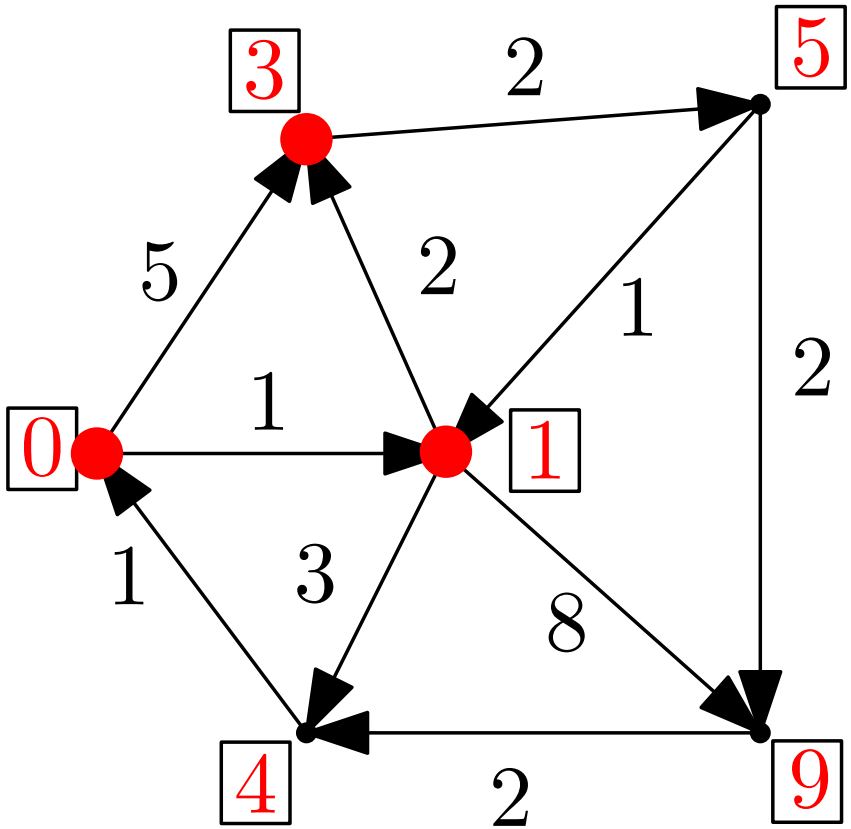
Beispiel



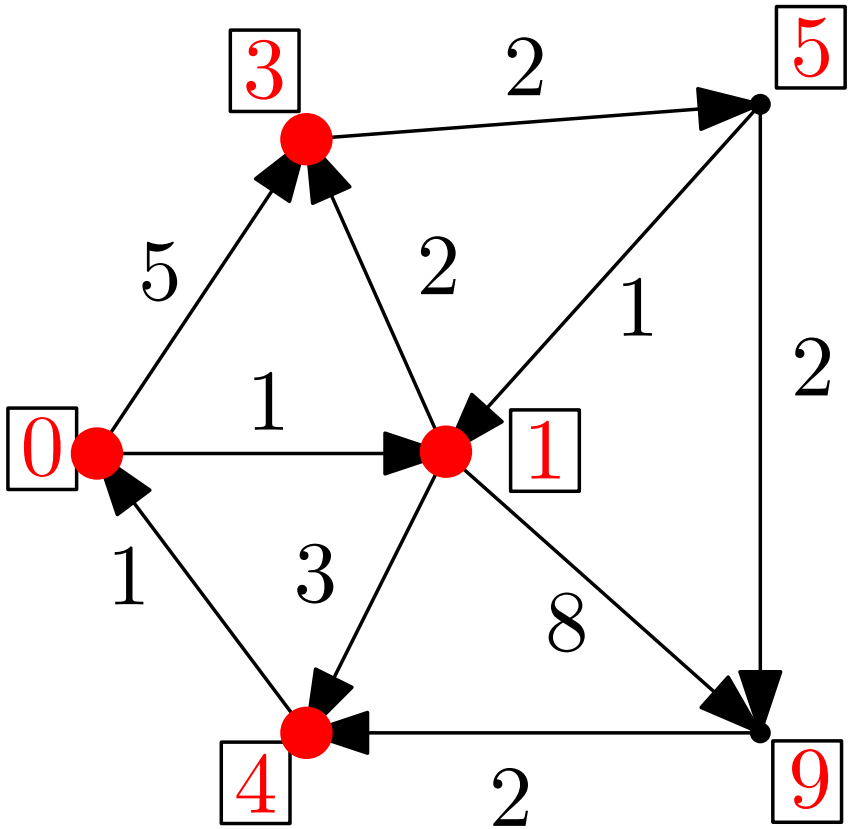
Beispiel



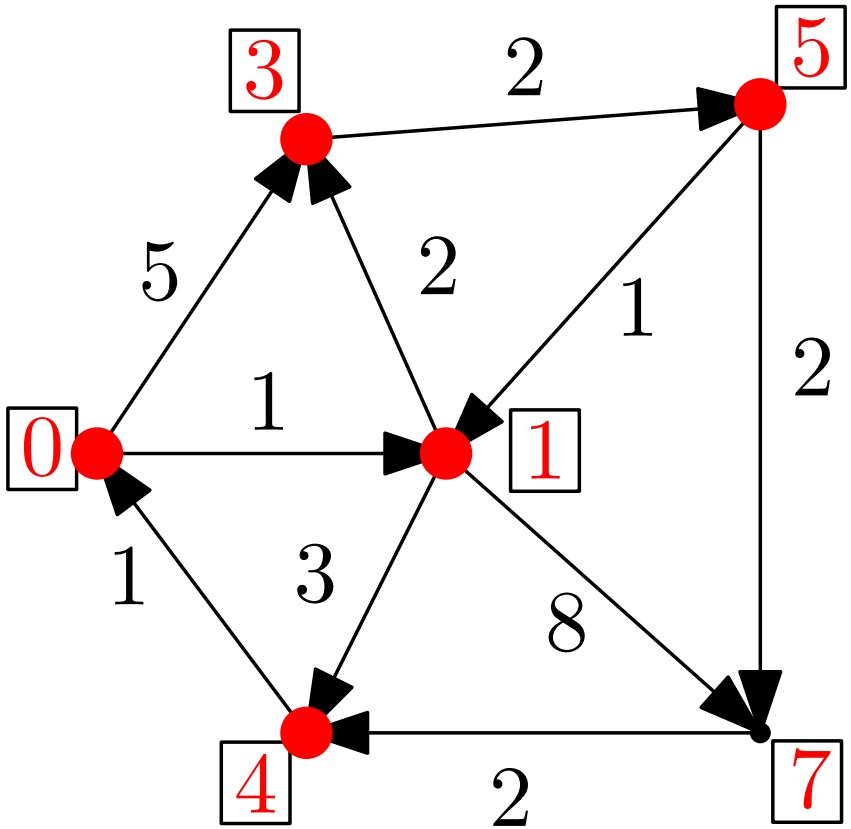
Beispiel



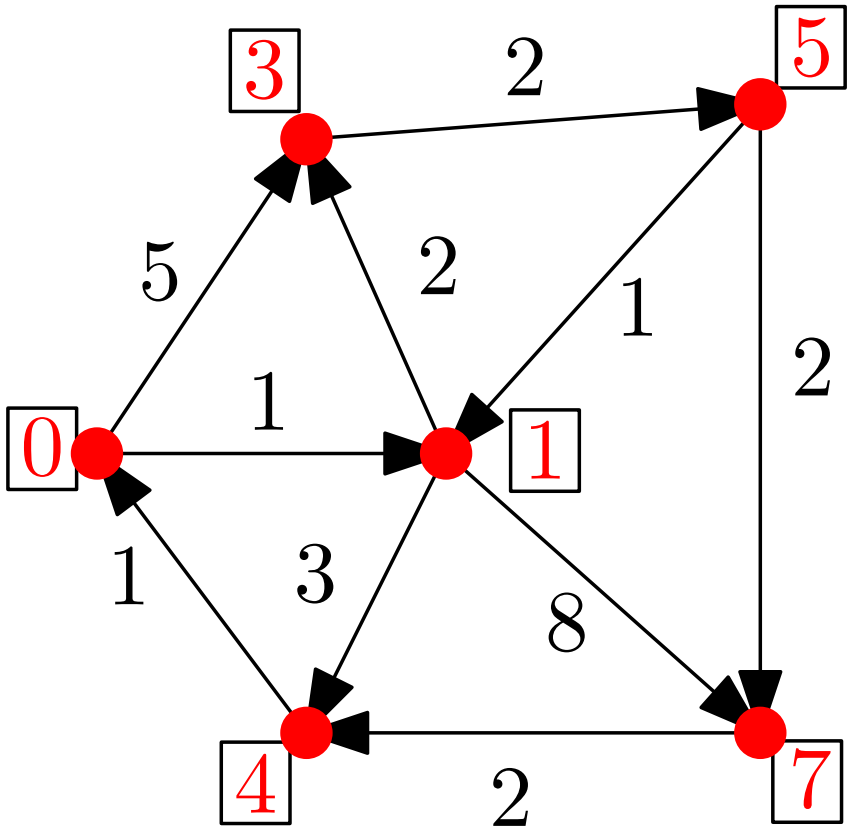
Beispiel



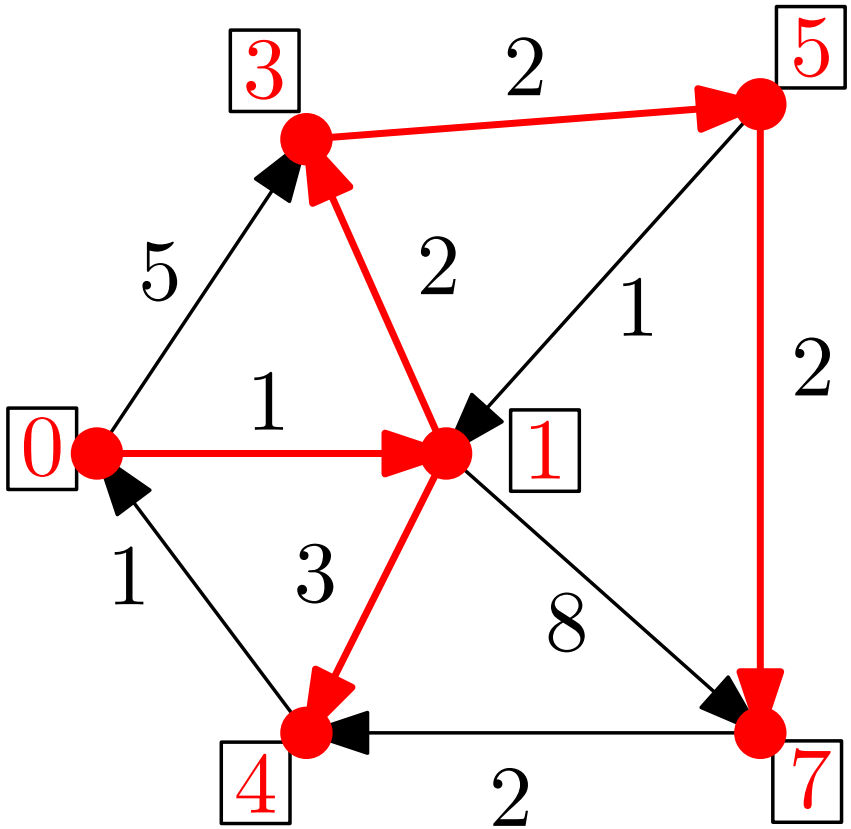
Beispiel



Beispiel



Beispiel



Korrektheit

Beh.: Dijkstra terminiert mit $d[v] = d(s, v)$ für alle $v \in V$

zwei Schritte:

(i) alle erreichbaren Knoten werden abgearbeitet

(ii) wenn v abgearbeitet wird, ist $d[v] = d(s, v)$

Korrektheit

Beh.: Dijkstra terminiert mit $d[v] = d(s, v)$ für alle $v \in V$

zwei Schritte:

(i) alle erreichbaren Knoten werden abgearbeitet

(ii) wenn v abgearbeitet wird, ist $d[v] = d(s, v)$

Beweis (i):

- Beh.: v wird nicht bearbeitet, ist aber erreichbar

$\Rightarrow$ es gibt kürzesten Weg $\Pi(s, t) = (v_1, \ldots, v_k), s = v_1, v = v_k$

$\Rightarrow$ es gibt v_i mit v_{i-1} abgearbeitet und v_i nicht

$\Rightarrow$ v_i wird in die Queue eingefügt

$\Rightarrow$ Widerspruch

Korrektheit

Beh.: Dijkstra terminiert mit $d[v] = d(s, v)$ für alle $v \in V$

zwei Schritte:

(i) alle erreichbaren Knoten werden abgearbeitet

(ii) wenn v abgearbeitet wird, ist $d[v] = d(s, v)$

Beweis (ii):

- Beh.: v wird abgearbeitet mit $d[v] > d(s, v)$

$\Rightarrow$ es gibt v_i, dass noch nicht abgearbeitet wurde, und für das gilt $d[v_i] + \text{len}(v_i, v) < d[v]$

$\Rightarrow$ $d[v_i] < d[v]$

$\Rightarrow$ v_i wird vor v abgearbeitet

$\Rightarrow$ Widerspruch

Dijkstra

```
1  forall nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL ;                          // n Mal
3  d[s] = 0, Q.clear(), Q.add(s, 0);                    // 1 Mal
4  while !Q.empty() do
5      u ← Q.deleteMin();                               // n Mal
6      forall edges e = (u, v) ∈ E do
           // relaxing edges
7          if d[u] + len(e) < d[v] then
8              d[v] ← d[u] + len(e)
9              p[v] ← u
10             if v ∈ Q then Q.decreaseKey(v, d[v]);    // m Mal
11             else Q.insert(v, d[v]);                  // n Mal
```

Dijkstra

```
1  forall nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL ;                          // n Mal
3  d[s] = 0, Q.clear(), Q.add(s, 0);                    // 1 Mal
4  while !Q.empty() do
5      u ← Q.deleteMin();                               // n Mal
6      forall edges e = (u, v) ∈ E do
           // relaxing edges
7          if d[u] + len(e) < d[v] then
8              d[v] ← d[u] + len(e)
9              p[v] ← u
10             if v ∈ Q then Q.decreaseKey(v, d[v]);    // m Mal
11             else Q.insert(v, d[v]);                  // n Mal
```

$$T_{\text{DIJKSTRA}} = T_{init} + n \cdot T_{deleteMin} + m \cdot T_{decreaseKey} + n \cdot T_{insert}$$

Laufzeit

$$T_{\text{DIJKSTRA}} = T_{init} + n \cdot T_{deleteMin} + m \cdot T_{decreaseKey} + n \cdot T_{insert}$$

Operation	Liste (worst-case)	Binary Heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
Init	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(1)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
Minimum	$\Theta(n)$	$\Theta(1)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Union	$\Theta(1)$	$\Theta(k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DecreaseKey	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Dijkstra				

Transportnetzwerke sind dünn $\Rightarrow$ Binary Heaps

Laufzeit

$$T_{\text{DIJKSTRA}} = T_{init} + n \cdot T_{deleteMin} + m \cdot T_{decreaseKey} + n \cdot T_{insert}$$

Operation	Liste (worst-case)	Binary Heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
Init	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(1)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
Minimum	$\Theta(n)$	$\Theta(1)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Union	$\Theta(1)$	$\Theta(k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DecreaseKey	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Dijkstra	$\mathcal{O}(n^2 + m)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}(m + n\lg n)$

Transportnetzwerke sind dünn $\Rightarrow$ Binary Heaps

Laufzeit

$$T_{\text{DIJKSTRA}} = T_{init} + n \cdot T_{deleteMin} + m \cdot T_{decreaseKey} + n \cdot T_{insert}$$

Operation	Liste (worst-case)	Binary Heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
Init	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(1)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
Minimum	$\Theta(n)$	$\Theta(1)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Union	$\Theta(1)$	$\Theta(k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DecreaseKey	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Dijkstra	$\mathcal{O}(n^2 + m)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}(m + n\lg n)$
Dij ($m \in \Theta(n^2)$)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2 \lg n)$	$\mathcal{O}(n^2 \lg n)$	$\mathcal{O}(n^2)$

Transportnetzwerke sind dünn $\Rightarrow$ Binary Heaps

Laufzeit

$$T_{\text{DIJKSTRA}} = T_{init} + n \cdot T_{deleteMin} + m \cdot T_{decreaseKey} + n \cdot T_{insert}$$

Operation	Liste (worst-case)	Binary Heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
Init	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(1)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
Minimum	$\Theta(n)$	$\Theta(1)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Union	$\Theta(1)$	$\Theta(k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DecreaseKey	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Dijkstra	$\mathcal{O}(n^2 + m)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}(m + n\lg n)$
Dij ($m \in \Theta(n^2)$)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2 \lg n)$	$\mathcal{O}(n^2 \lg n)$	$\mathcal{O}(n^2)$
Dij ($m \in \mathcal{O}(n)$)	$\mathcal{O}(n^2)$	$\mathcal{O}(n \lg n)$	$\mathcal{O}(n \lg n)$	$\mathcal{O}(n \lg n)$

Transportnetzwerke sind dünn $\Rightarrow$ Binary Heaps

Laufzeit

$$T_{\text{DIJKSTRA}} = T_{init} + n \cdot T_{deleteMin} + m \cdot T_{decreaseKey} + n \cdot T_{insert}$$

Operation	Liste (worst-case)	Binary Heap (worst-case)	Binomial heap (worst-case)	Fibonacci heap (amortized)
Init	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
Insert	$\Theta(1)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
Minimum	$\Theta(n)$	$\Theta(1)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Union	$\Theta(1)$	$\Theta(k)$	$\mathcal{O}(\lg k)$	$\Theta(1)$
DecreaseKey	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\mathcal{O}(\lg k)$
Dijkstra	$\mathcal{O}(n^2 + m)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}((n+m)\lg n)$	$\mathcal{O}(m + n\lg n)$
Dij ($m \in \Theta(n^2)$)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2 \lg n)$	$\mathcal{O}(n^2 \lg n)$	$\mathcal{O}(n^2)$
Dij ($m \in \mathcal{O}(n)$)	$\mathcal{O}(n^2)$	$\mathcal{O}(n \lg n)$	$\mathcal{O}(n \lg n)$	$\mathcal{O}(n \lg n)$

Transportnetzwerke sind dünn $\Rightarrow$ Binary Heaps