Algorithmen für Routenplanung – Vorlesung 1

Daniel Delling

Lehrstuhl für Algorithmik I
Institut für theoretische Informatik
Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Organisatorisches

Vorlesung

- Daniel Delling
- freitags 9:45 - 11:15, SR 301
- 2+1 SWS
- Achtung: nächste Woche (1.5.) Feiertag
- Achtung: übernächste Woche (4.5.) auf Übungstermin

Übung

- Thomas Pajor
- montags (14 tägig) 15:45 - 17:30, SR 301

Vorlesungsseite:

`http://i11www.iti.uni-karlsruhe.de/teaching/sommer2009/routenplanung/index`

Weitere Vorlesungen SS 09

Algorithmen für planare Graphen (2+1 SWS)

- Dozentin: Prof. Dorothea Wagner
- Vorlesung: dienstags 14:00 - 15:30
- Übung (14-tg): dienstags 14:00 - 15:30

Algorithmen zur Visualisierung von Graphen (2+1 SWS)

- Dozent: Martin Nöllenburg
- Vorlesung: donnerstags 14:00 - 15:30
- Übung (14-tg): dienstags 9:45 - 11:15

Algorithmen für Ad-hoc- und Sensornetze (2 SWS)

- Dozent: Bastian Katz
- Vorlesung: mittwochs 14:00 - 15:30

$\Rightarrow$ **Algorithmentechnikblock SS09 machbar**

Weitere Veranstaltungen SS 09

Seminar: Parametrisierte Algorithmen für NP-schwere Probleme

- Ansprechpartner:
  - Reinhard Bauer
  - Marcus Krug
  - Ignaz Rutter

Praktikum: Werkzeugentwicklung für die Routenplanung

- ITI Sanders
- Vorbesprechung 30.4.
- Ansprechpartner:
  - Veit Batz
  - Dennis Luxen

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 $O(n)$
  - 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 $O(n)$
  - kurze Vorberechnung (im Bereich Stunden/Minuten)
  - hohe Beschleunigung

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 1
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik

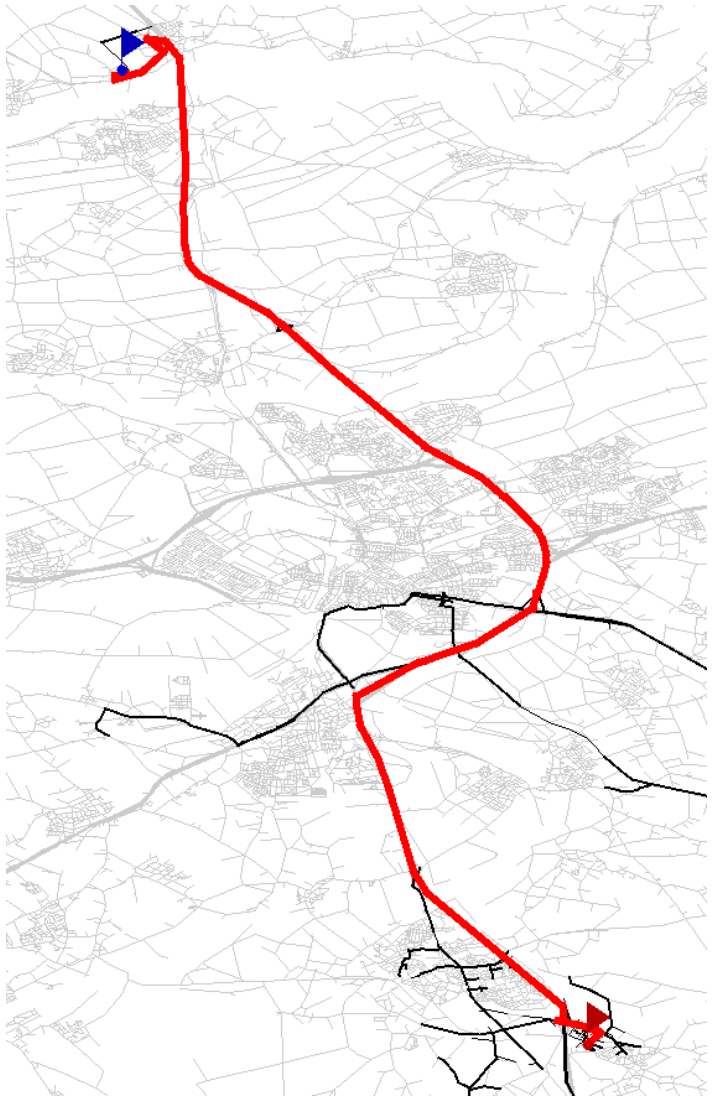
Beschleunigungstechniken

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 $O(n)$
  - kurze Vorberechnung (im Bereich Stunden/Minuten)
  - hohe Beschleunigung



Unterschiede zur Industrie

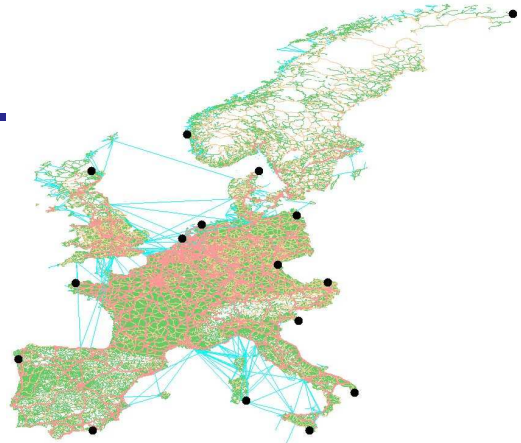
Industrie:

- 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

unser Anspruch:

- Anfragen sollen beweisbar korrekt sein
- ⇒ weniger Ausnahmeregelungen
- ⇒ schneller (!)
- Verfahren sollen nach und nach in der Industrie eingesetzt werden

Ergebnisse



Eingabe: Straßennetzwerk von Westeuropa

- 18 Mio. Knoten
- 42 Mio. Kanten

		Vorberechnung		Anfrage	
		Zeit	Platz	Zeit	
	Jahr	[h:m]	[byte/n]	[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 Variant deutlich langsamer, wir sehen nachher, warum

Schwerpunkte

Vorlesung

- Algorithm Engineering
- Beschleunigungstechniken
- Implementierungsdetails
- Ergebnisse auf Real-Welt Daten
- aktueller Stand der Forschung (Veröffentlichungen bis 2009)
- Ansatz übertragbar auf andere Themen (z.B. Flüsse)
- ideale Grundlage für Studien- und Diplomarbeiten in dem Bereich

Übungen:

- 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 Wege (Dijkstra)
  - Grundlagen sind Stoff von Info 2/Algotech (heute nochmal Crashkurs)
- Grundvorlesung "vereinfachen" Wahrheit oft
- Implementierung
- Betonung auf Messergebnisse

keine Theorievorlesung

- wenig Beweise (wenn doch, sehr kurz)
- Reale Leistung vor Asymptotik
- hinter vielen Optimierungsproblemen stehen (vielleicht ?) NP-schwere Probleme
- wird hier nur kurz erwähnt

Inhalt der Vorlesung

1. Grundlagen
- Algorithm Engineering
- Graphen, Modelle, usw.
- Kürzeste Wege

2. Beschleunigung von (statischen) Punkt-zu-Punkt Anfragen
- zielgerichtete Verfahren
- hierarchische Techniken
- Kombinationen

3. Erweiterungen
- many-to-many
- dynamische Szenarien
- zeitabhängige Routenplanung
- Alternativrouten

4. Offene Probleme

Nützliche Vorkenntnisse

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

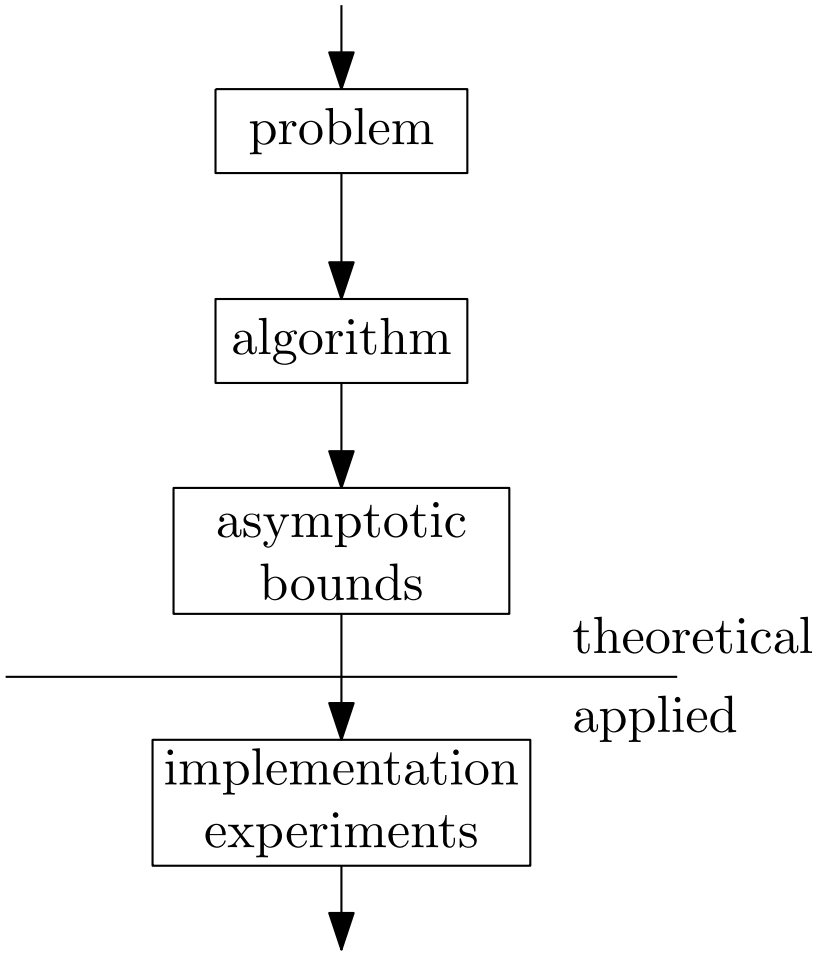
Vertiefungsgebiet: Algorithmentechnik, (Theoretische Grundlagen)

Material

- Folien
- wissenschaftliche Aufsätze (siehe Vorlesunghomepage)
- Basiskenntnisse:
  - Cormen Leiserson, Rivest, Stein: *Introduction to Algorithms*
  - Mehlhorn, Sanders: *Algorithms and Data Structures*

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:

- anwendungsnahes Gebiet
- Eingaben sind "echte" Daten
  - Straßengraphen
  - Eisenbahn
  - Flugplan

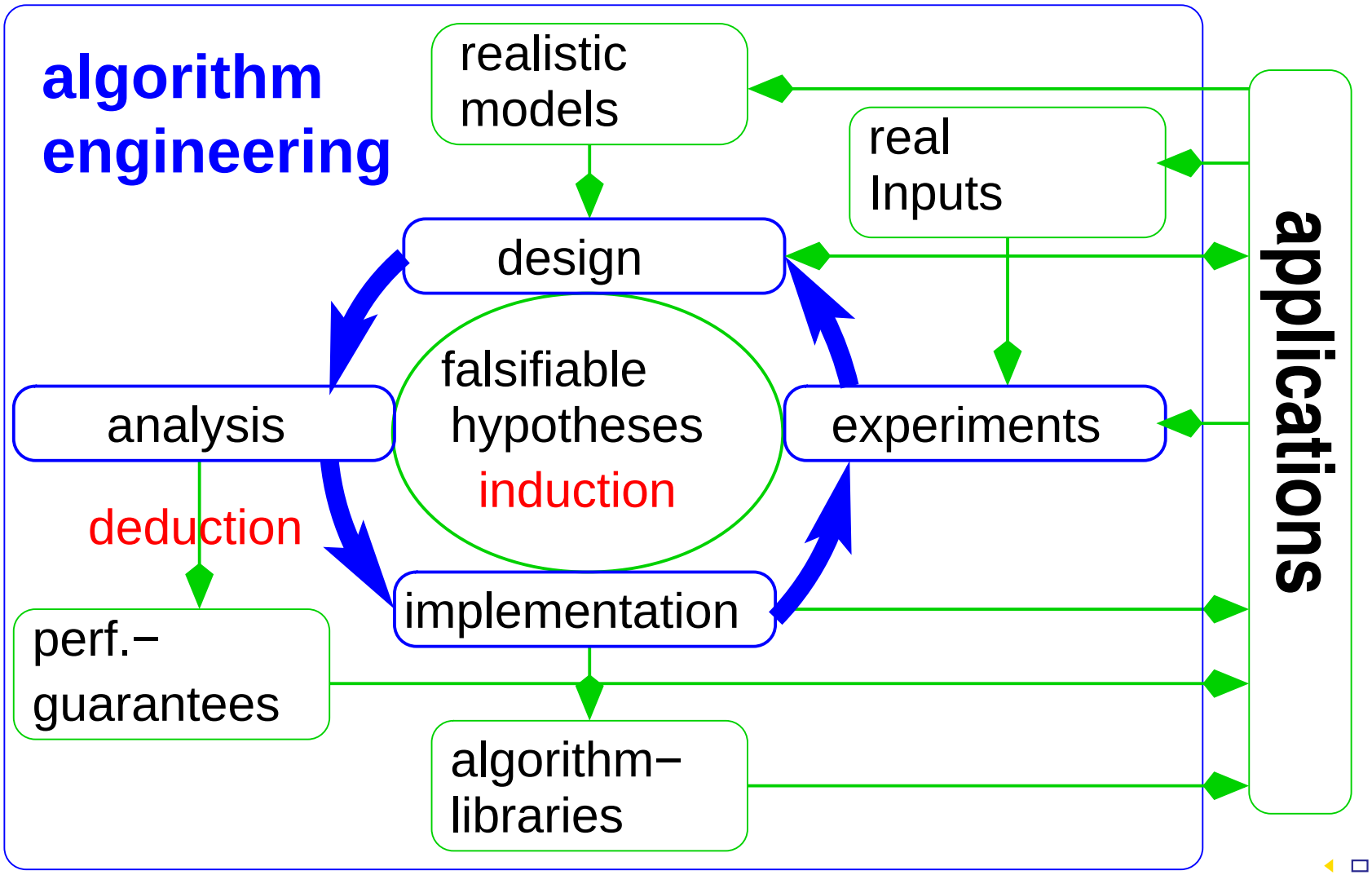
Lücke Theorie vs. Praxis

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:

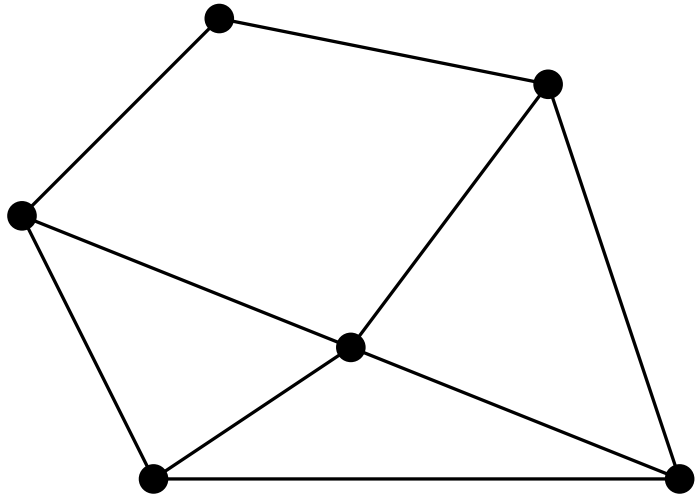
- anwendungsnahes Gebiet
- Eingaben sind "echte" Daten
  - Straßengraphen
  - Eisenbahn
  - Flugplan

Algorithm Engineering

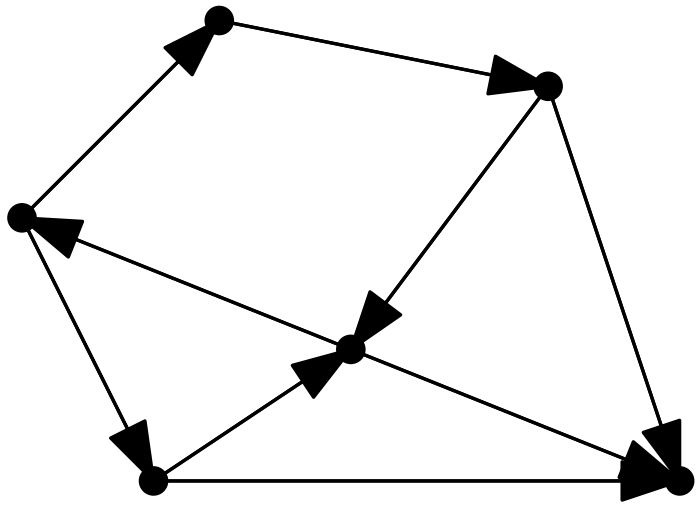


Graphen

- Tupel $(V, E) =: G$
  - endliche Knotenmenge V
  - endliche Kantenmenge E
  - $n := |V|$, $m := |E|$
- **ungerichtet:** Kanten sind Knotenpaar, d.h. $E \subseteq \binom{V}{2} = \{\{u, v\} | u \neq v,\ u, v \in V\}$
  - 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: $\deg_{out}(u) = \sum_{(u,v) \in E}$
  - Eingangsgrad: $\deg_{in}(u) = \sum_{(v,u) \in E}$
- **einfach:** keine Multi-Kanten
- **gewichtet:** Kantengewichtsfunktion
  - erstmal $len : E \to \mathbb{R}^+$
- **dünn:** $m \in O(n)$
- **planar:** kreuzungsfrei einbettbar



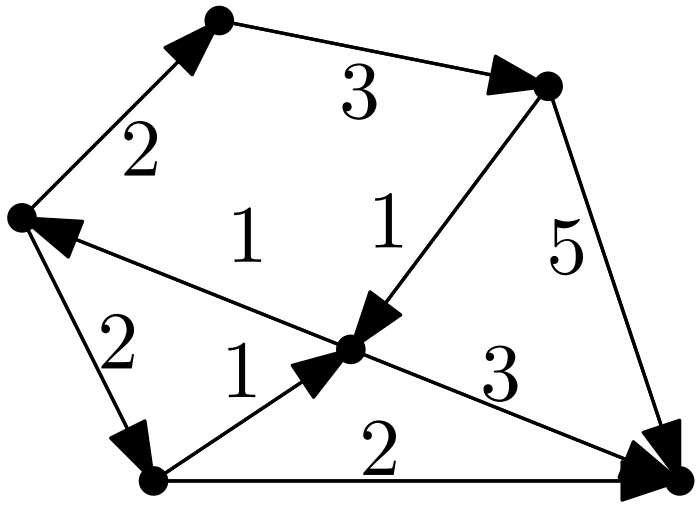
Graphen



- Tupel $(V, E) =: G$
  - endliche Knotenmenge V
  - endliche Kantenmenge E
  - $n := |V|$, $m := |E|$
- **ungerichtet:** Kanten sind Knotenpaar, d.h. $E \subseteq \binom{V}{2} = \{\{u, v\} | u \neq v,\ u, v \in V\}$
  - 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: $\deg_{out}(u) = \sum_{(u,v) \in E}$
  - Eingangsgrad: $\deg_{in}(u) = \sum_{(v,u) \in E}$
- **einfach:** keine Multi-Kanten
- **gewichtet:** Kantengewichtsfunktion
  - erstmal $len : E \to \mathbb{R}^+$
- **dünn:** $m \in O(n)$
- **planar:** kreuzungsfrei einbettbar

Graphen

- Tupel $(V, E) =: G$
  - endliche Knotenmenge V
  - endliche Kantenmenge E
  - $n := |V|$, $m := |E|$
- **ungerichtet:** Kanten sind Knotenpaar, d.h. $E \subseteq \binom{V}{2} = \{\{u, v\} | u \neq v,\ u, v \in V\}$
  - 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: $\deg_{out}(u) = \sum_{(u,v) \in E}$
  - Eingangsgrad: $\deg_{in}(u) = \sum_{(v,u) \in E}$
- **einfach:** keine Multi-Kanten
- **gewichtet:** Kantengewichtsfunktion
  - erstmal $len : E \to \mathbb{R}^+$
- **dünn:** $m \in O(n)$
- **planar:** kreuzungsfrei einbettbar



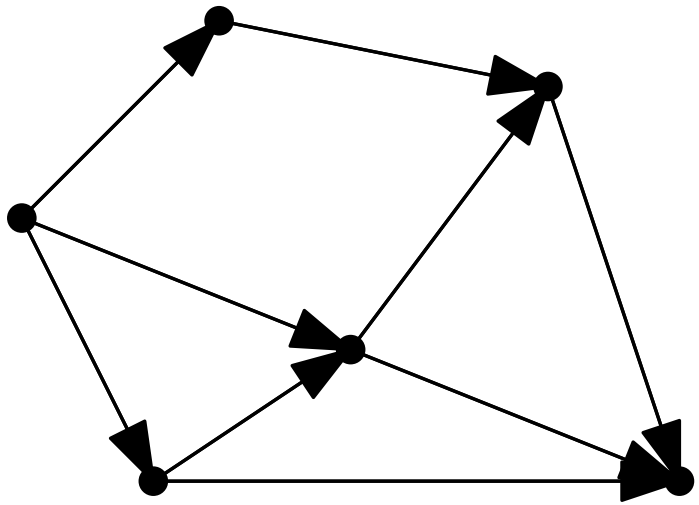
Spezielle Graphen

DAG:

- directed acyclic graph
- zyklenfrei

Baum:

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



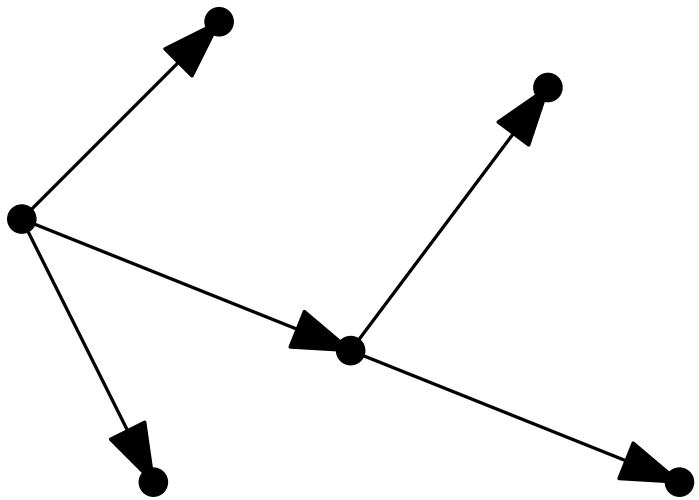
Spezielle Graphen

DAG:

- directed acyclic graph
- zyklenfrei

Baum:

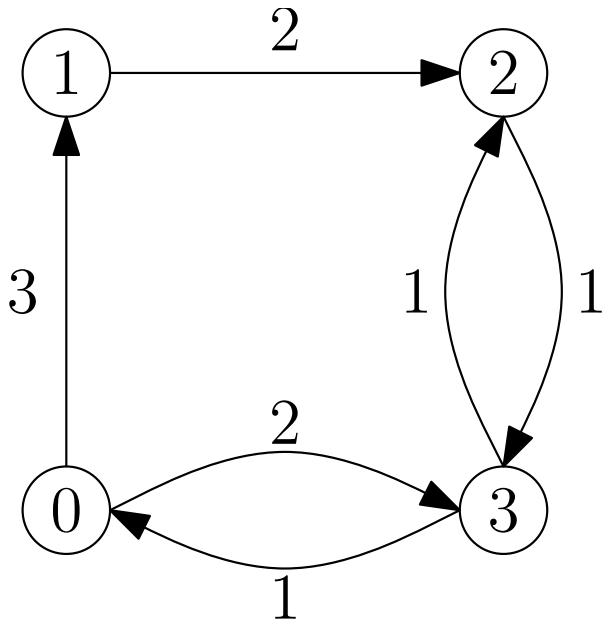
- zyklenfrei
- $m = n - 1$
- also dünn



Datenstruktur

drei klassische Ansätze:

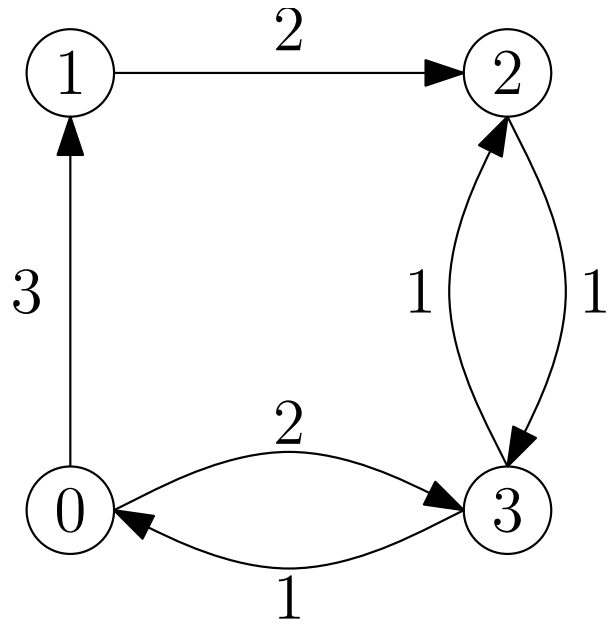
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



Datenstruktur

drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

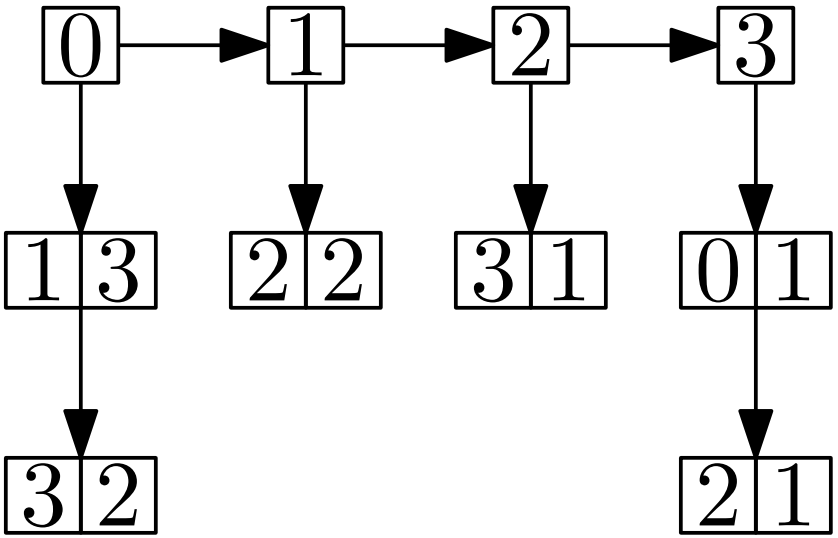


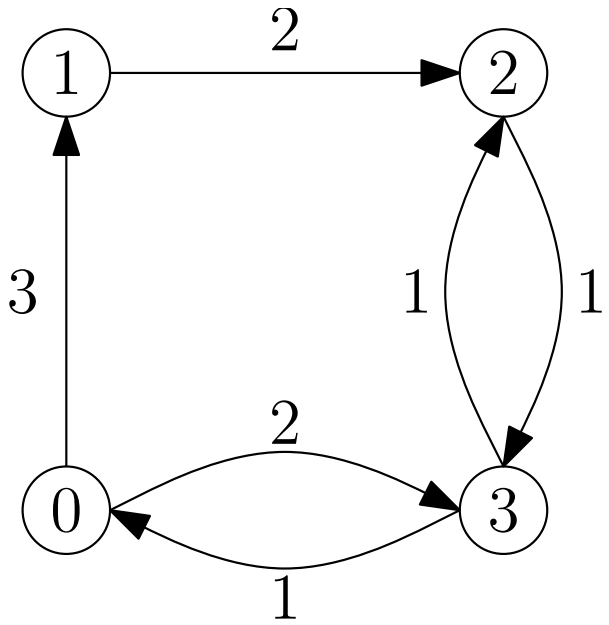
	0	1	2	3
0	–	3	–	2
1	–	–	2	–
2	–	–	–	1
3	1	–	1	–

Datenstruktur

drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

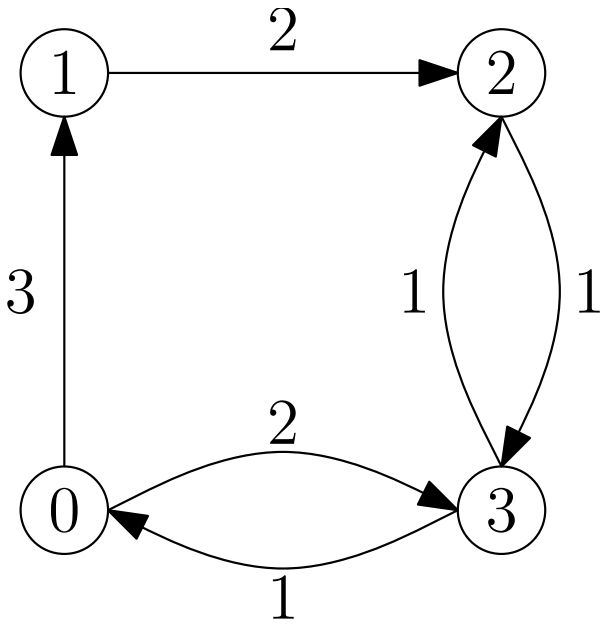


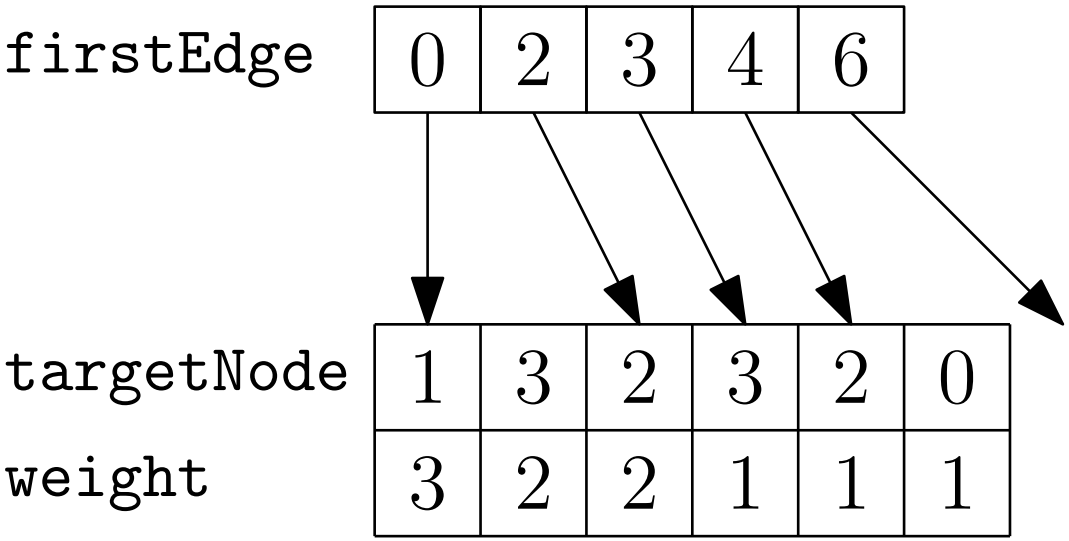


Datenstruktur

drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray





Was brauchen wir?

Eigenschaften:

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

Fragen:

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

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 1
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik


Was brauchen wir?

Eigenschaften:

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

Fragen:

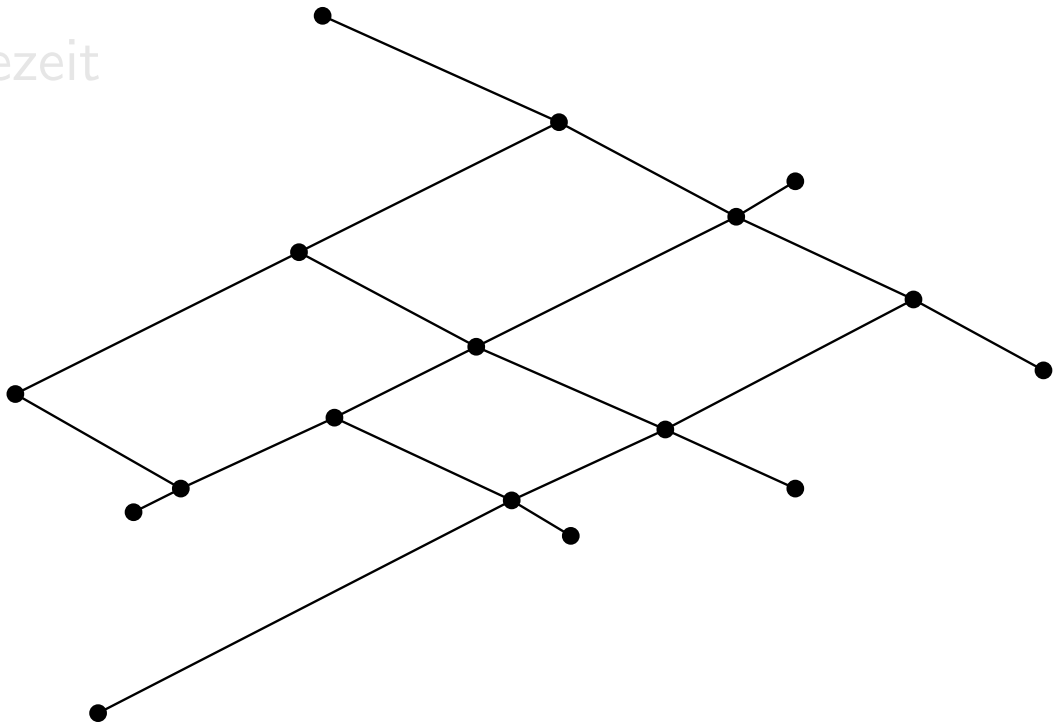
- Was brauchen wir?
- Was muss nich supereffizient sein?
- erstmal Modelle anschauen!

Modellierung (Straßengraphen)

Modellierung (Straßengraphen)

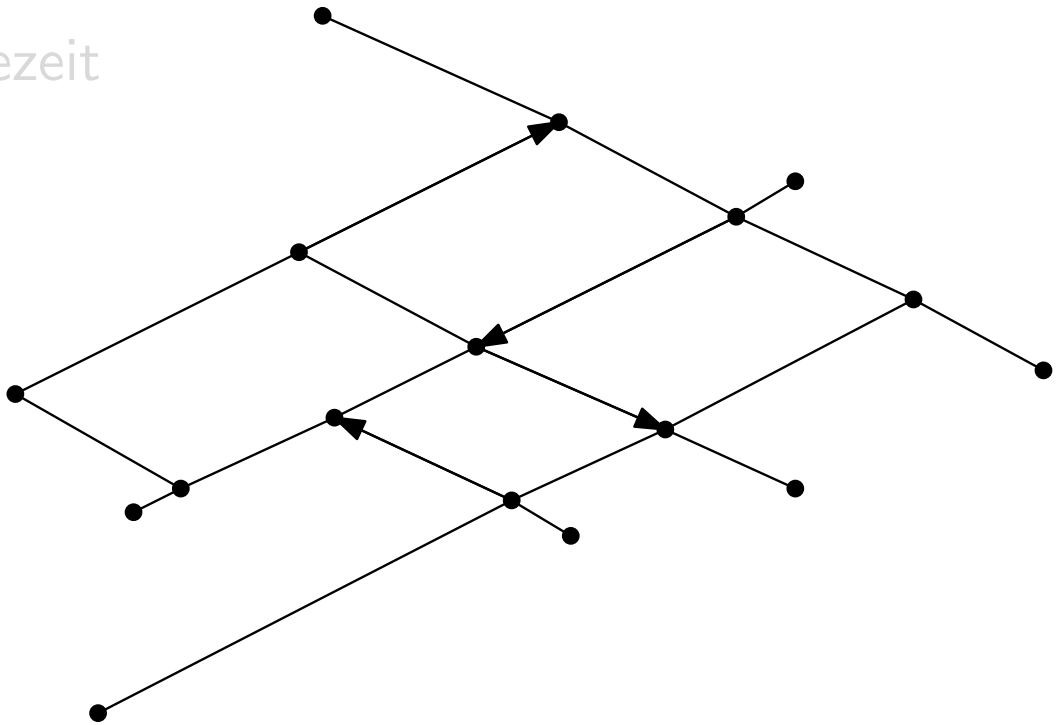
Modellierung (Straßengraphen)

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



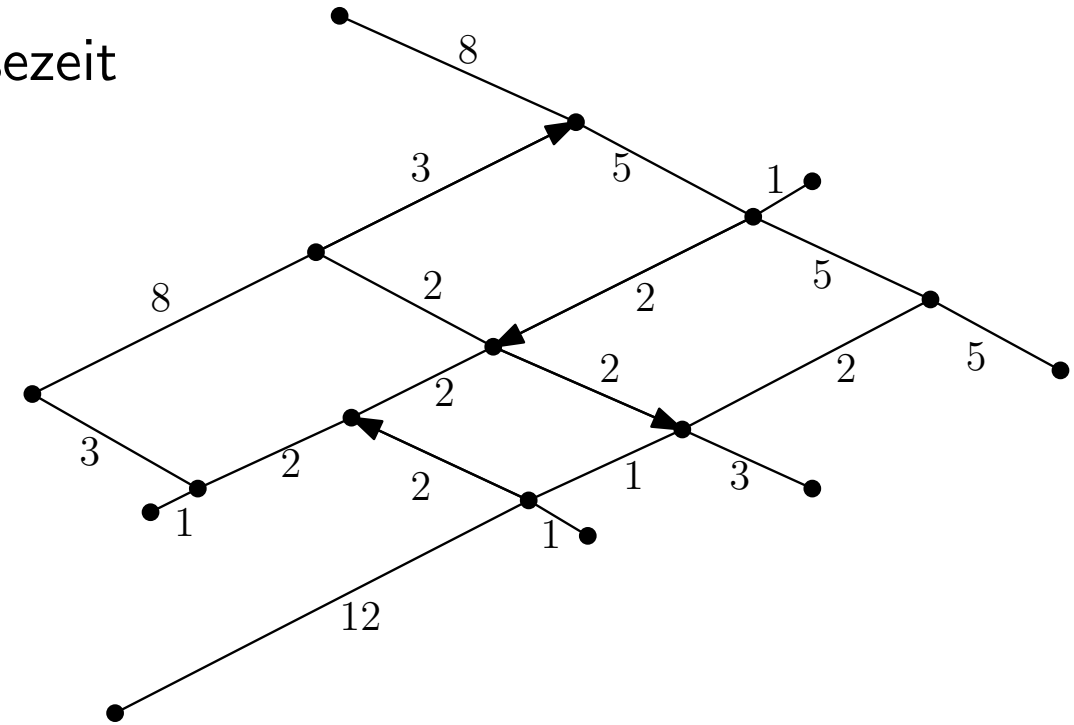
Modellierung (Straßengraphen)

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



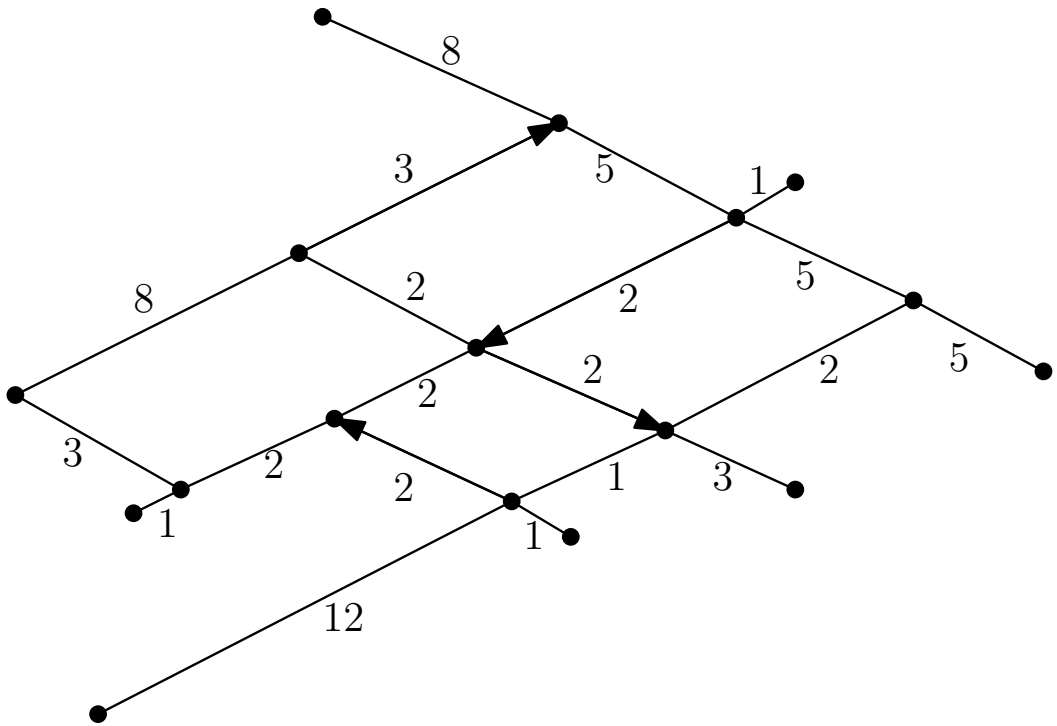
Modellierung (Straßengraphen)

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



Modellierung (Straßengraphen)

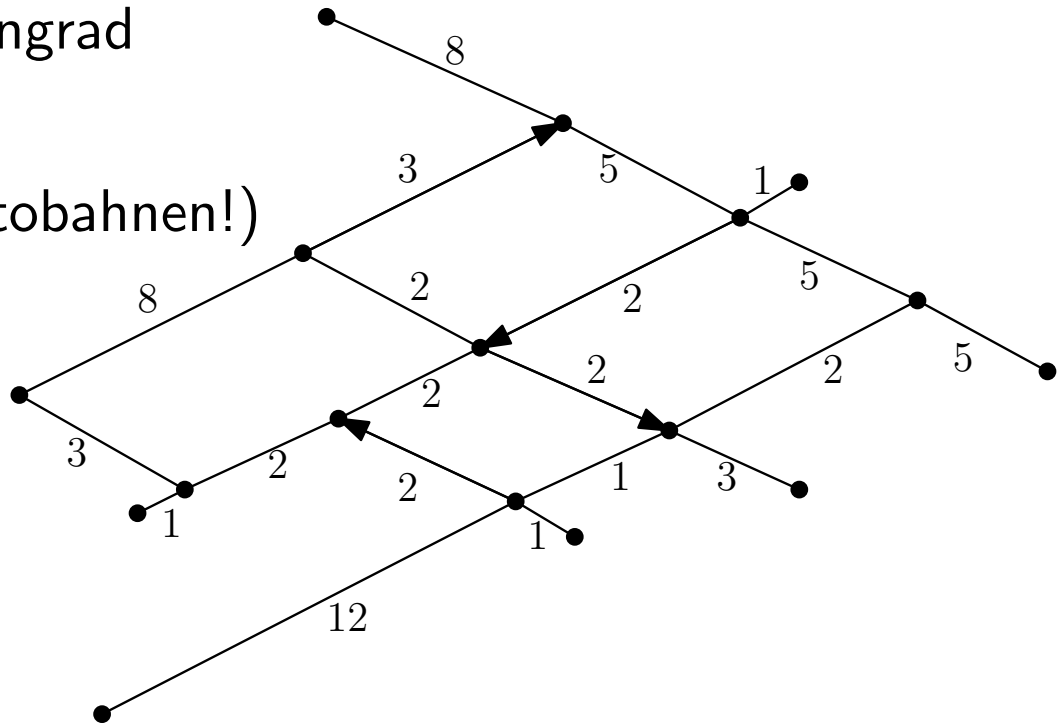
Eigenschaften (sammeln):



Modellierung (Straßengraphen)

Eigenschaften:

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



Modellierung (Eisenbahngraphen)

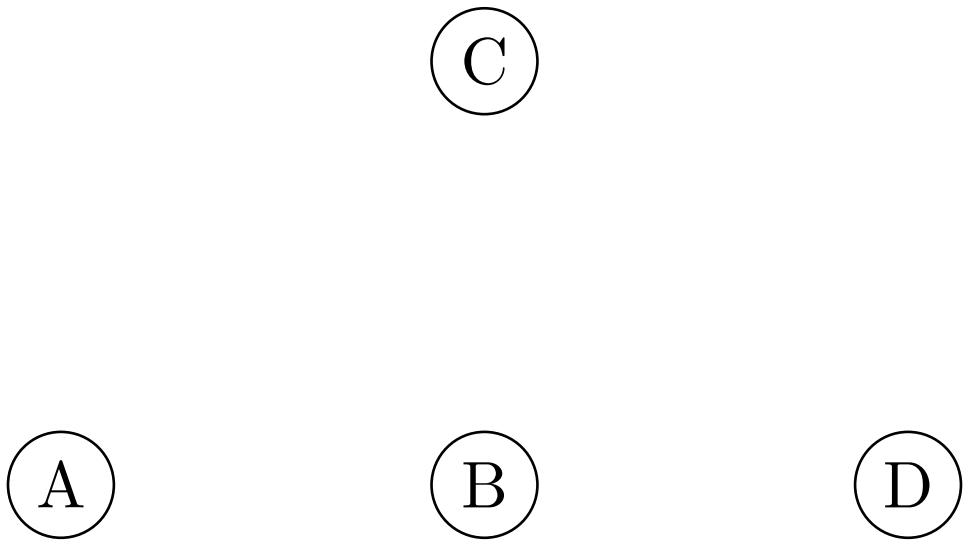
Eingabe:

- 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 aller 10 Minuten

Modellierung (Eisenbahngraphen)

Eingabe:

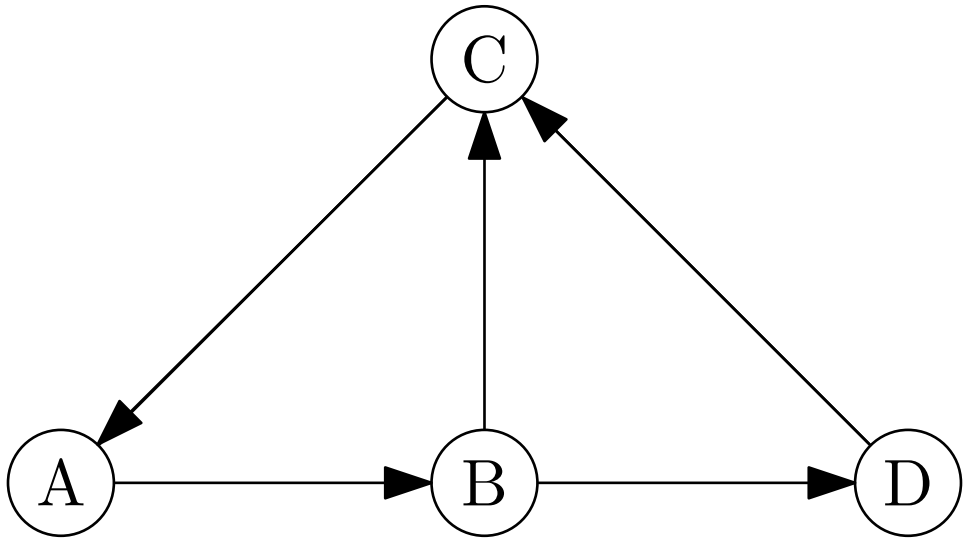
- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren aller 10 Minuten



Modellierung (Eisenbahngraphen)

Eingabe:

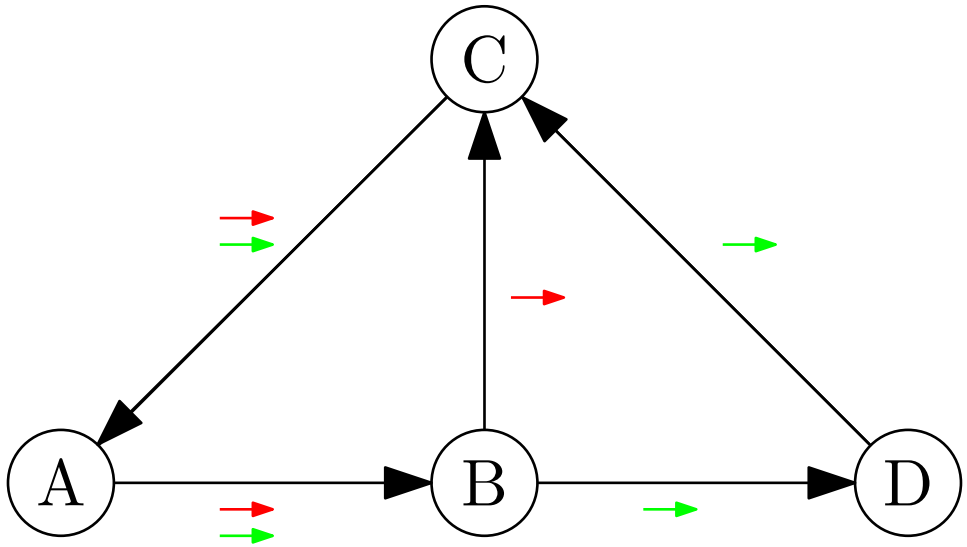
- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren aller 10 Minuten



Modellierung (Eisenbahngraphen)

Eingabe:

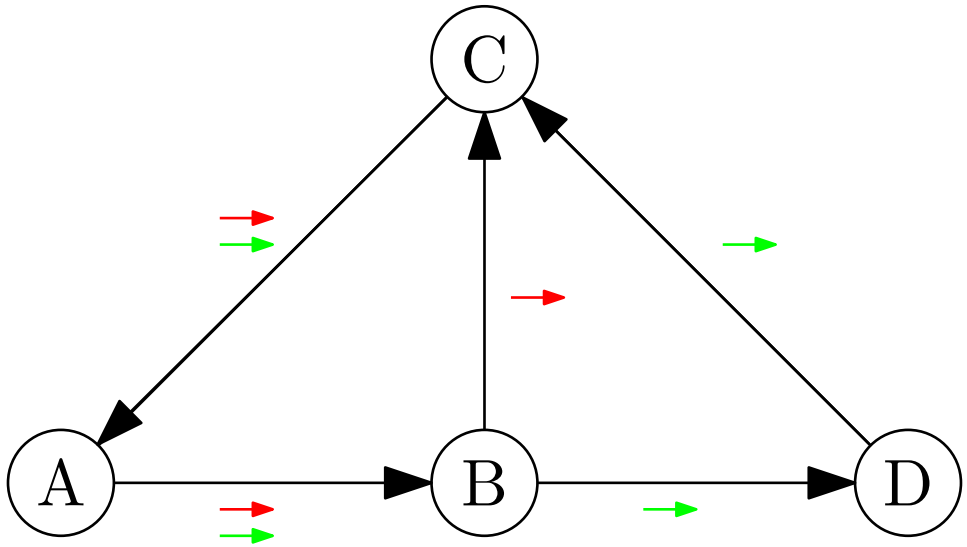
- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren aller 10 Minuten



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 aller 10 Minuten

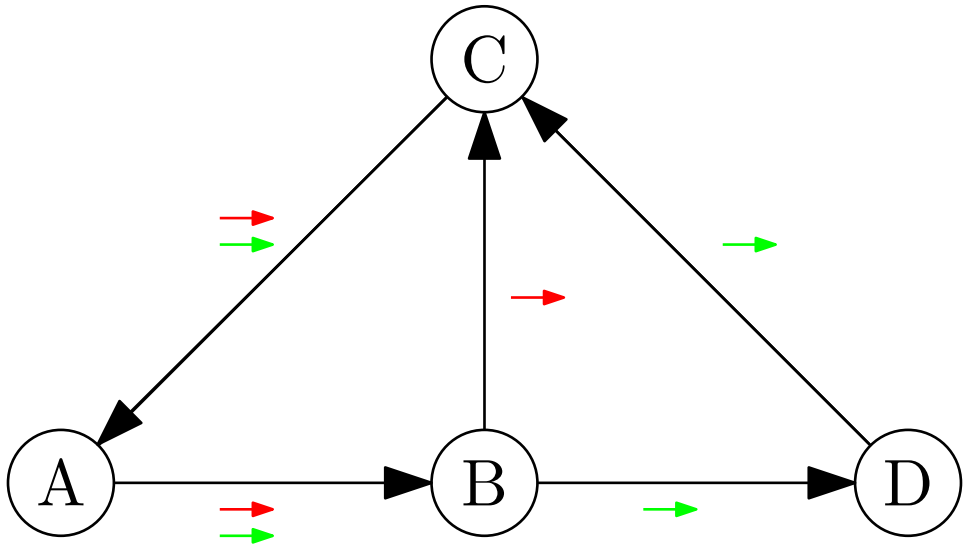


Kanten sind zeitabhängig!

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 aller 10 Minuten



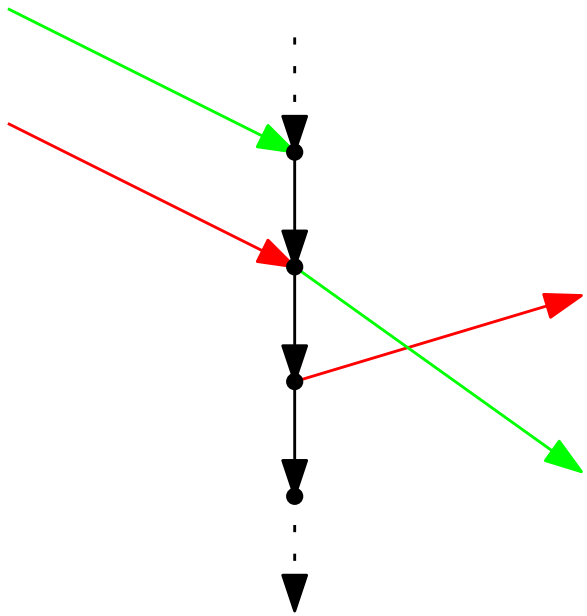
Kanten sind zeitabhängig! oder roll die Zeit aus

Zeitexpandiertes Modell

Vorgehen:

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

Station B:



Diskussion:

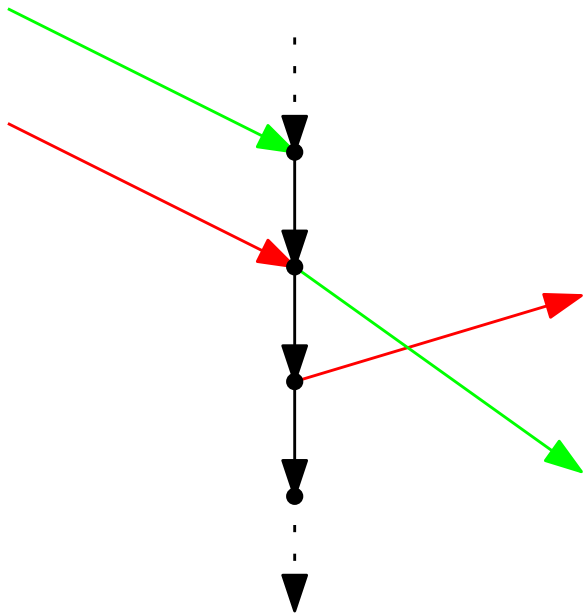
- \+ keine zeitabhängigen Kanten
- – Graph größer

Zeitexpandiertes Modell

Vorgehen:

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

Station B:

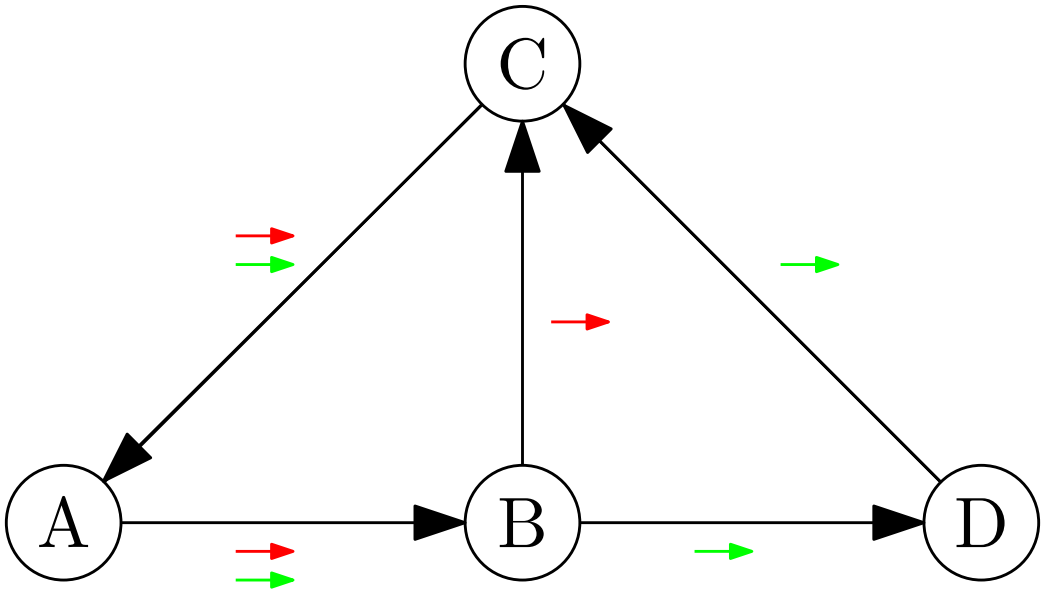


Diskussion:

- \+ keine zeitabhängigen Kanten
- – Graph größer

Problem der Modellierung

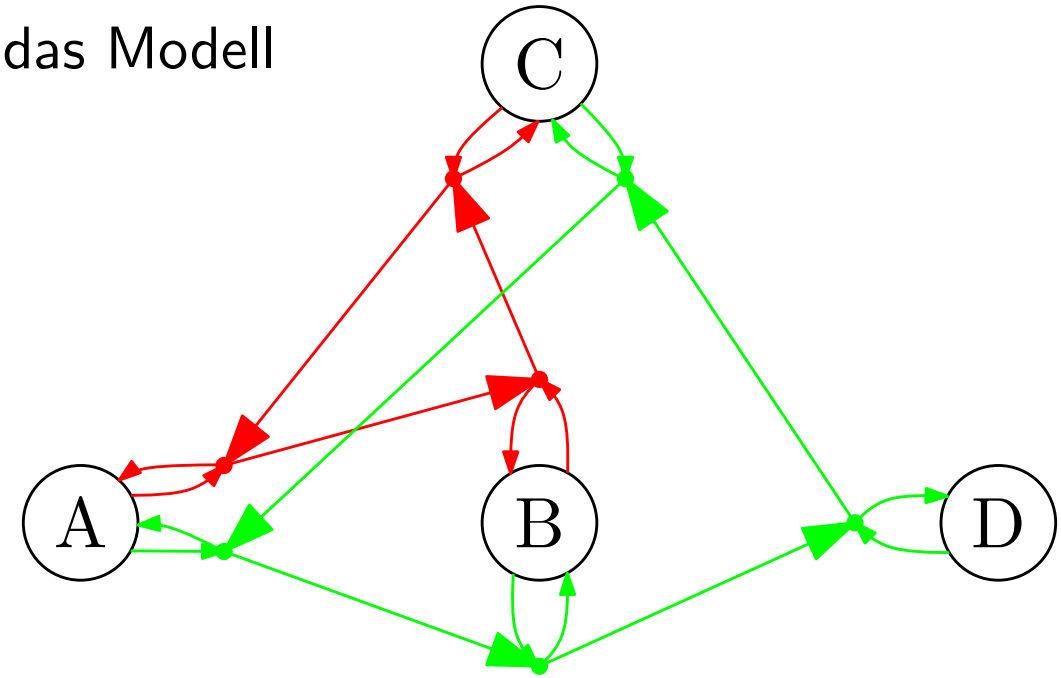
Problem:



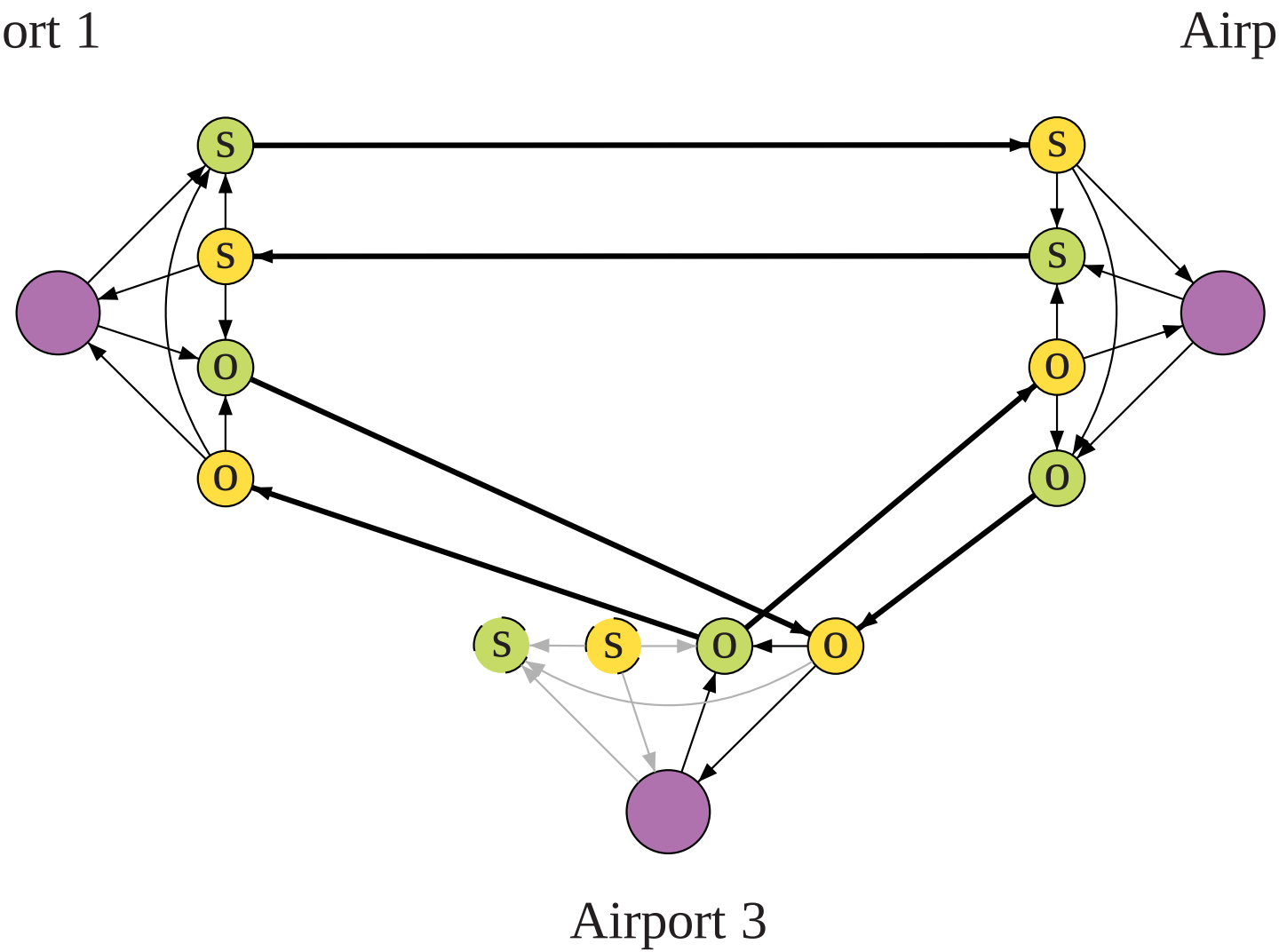
Problem der Modellierung

Problem:

- Umstiegszeiten?
- 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 positiv
- 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	$O(n^2)$	$O(n+m) = O(n)$	$O(n+m) = O(n)$
Ausgehende Kanten iterieren	$O(n)$	$O(\deg u)$	$O(\deg u)$
Kantenzugriff (u, v)	$O(1)$	$O(\deg u)$	$O(\deg u)$
Effizienz	$+$	$-$	$+$
Updates (topologisch)	$+$	$+$	$-$
Updates (Gewicht)	$+$	$+$	$+$

also:

- $\gg$ Adjazenzarray

Datenstruktur

Eigenschaften auf dünnen Graphen:

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

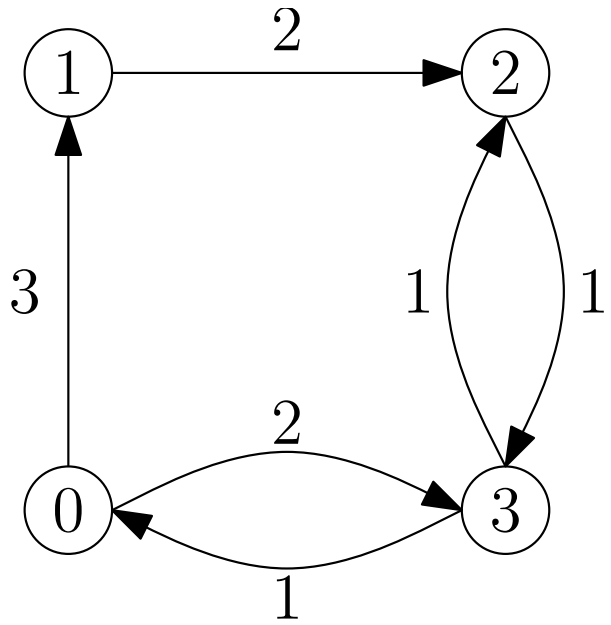
also:

- $\gg$ Adjazenzarray

Datenstruktur

Problem:

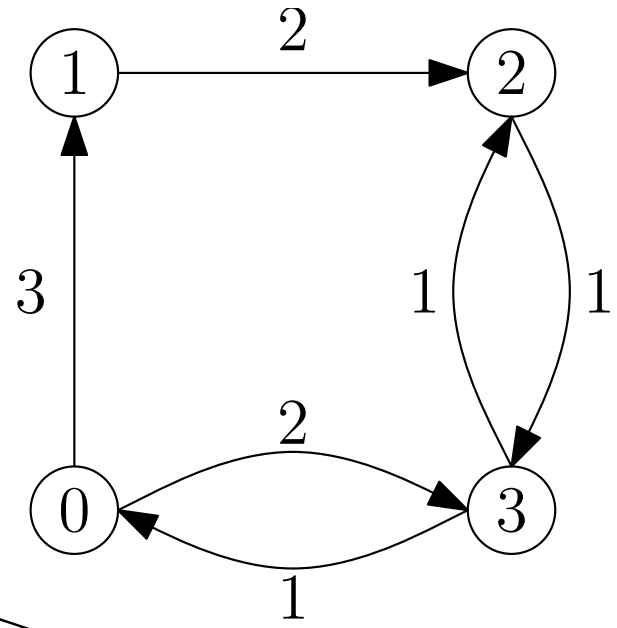
- ein- und ausgehende Kanten
- (fast) ungerichtet



Datenstruktur

Problem:

- ein- und ausgehende Kanten
- (fast) ungerichtet

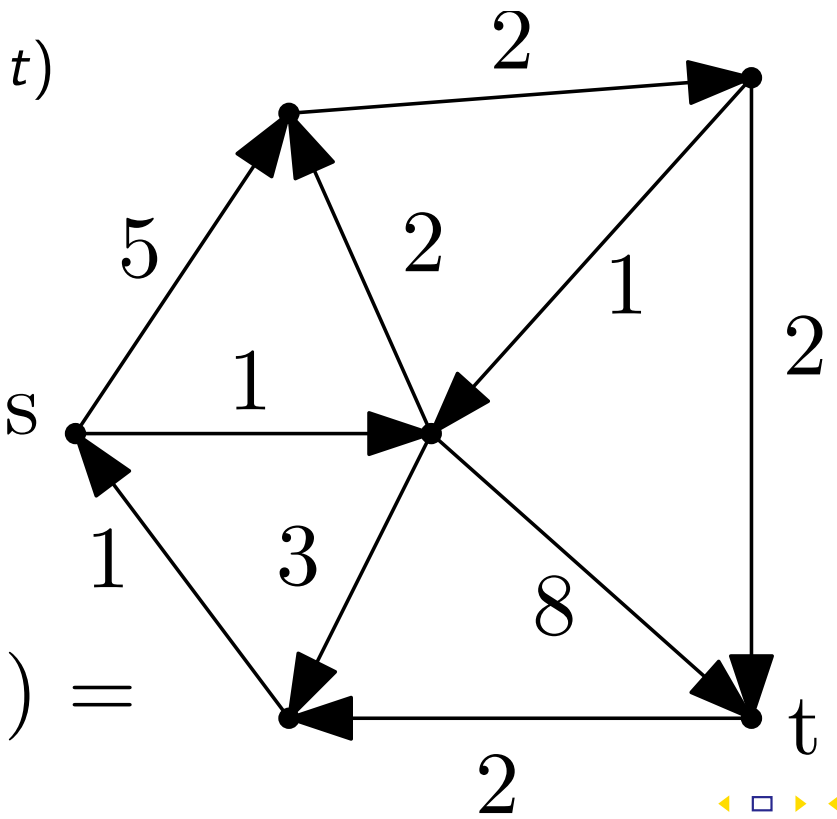


firstEdge	0	3	5	7	10					
targetNode	1	3	3	0	2	1	3	0	0	2
weight	3	2	1	3	2	2	1	1	2	1
isIncoming	–	–	√	√	–	√	√	–	√	√
isOutgoing	√	√	–	–	√	–	√	√	–	√

Kürzeste Wege

Anfrage:

- gegeben Graph $G = (V, E)$ mit positiver Kantenfunktion $len : E \to \mathbb{R}^+$, zwei Knoten s und t
- finde den Pfad $(s, \ldots, t)$ mit $d(s, t)$ minimal

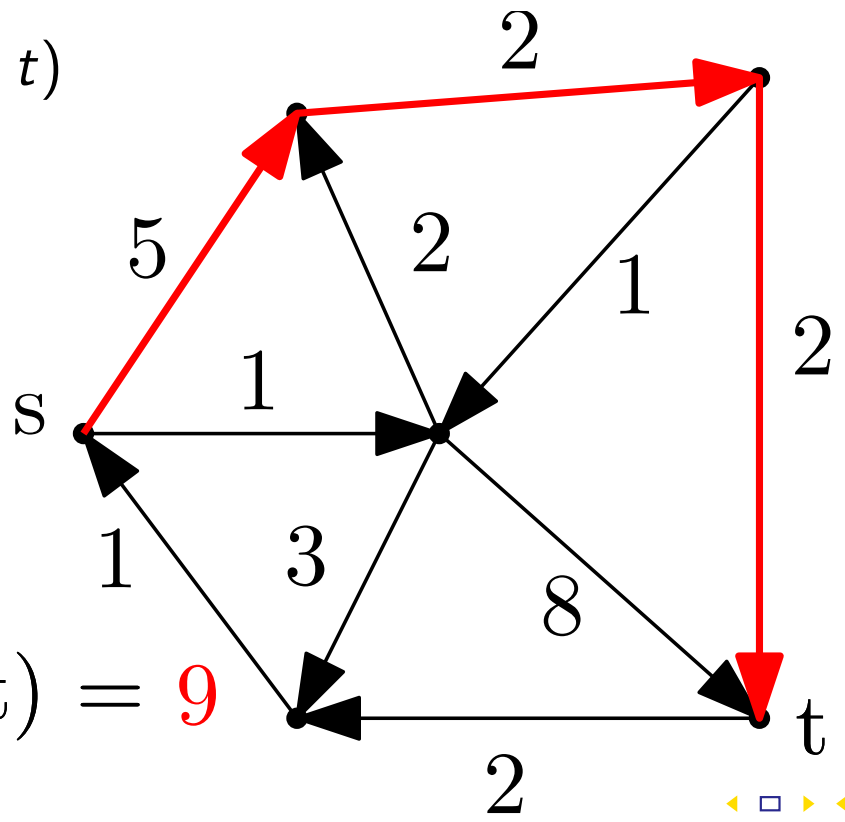


$$d(s,t) =$$

Kürzeste Wege

Anfrage:

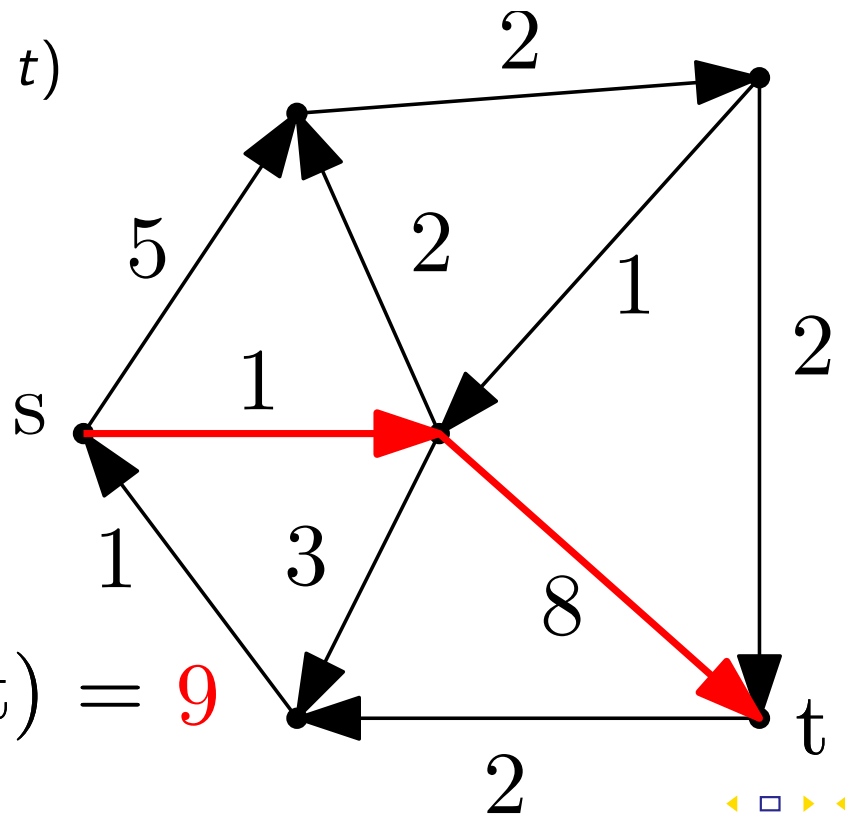
- gegeben Graph $G = (V, E)$ mit positiver Kantenfunktion $len : E \to \mathbb{R}^+$, zwei Knoten s und t
- finde den Pfad $(s, \ldots, t)$ mit $d(s, t)$ minimal



Kürzeste Wege

Anfrage:

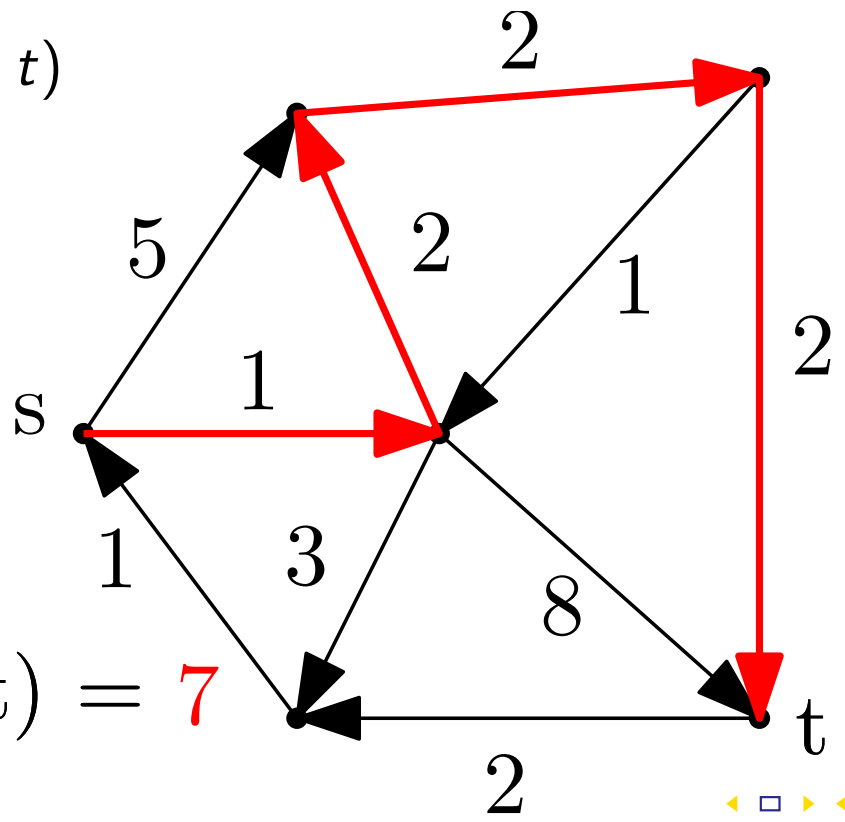
- gegeben Graph $G = (V, E)$ mit positiver Kantenfunktion $len : E \to \mathbb{R}^+$, zwei Knoten s und t
- finde den Pfad $(s, \ldots, t)$ mit $d(s, t)$ minimal



Kürzeste Wege

Anfrage:

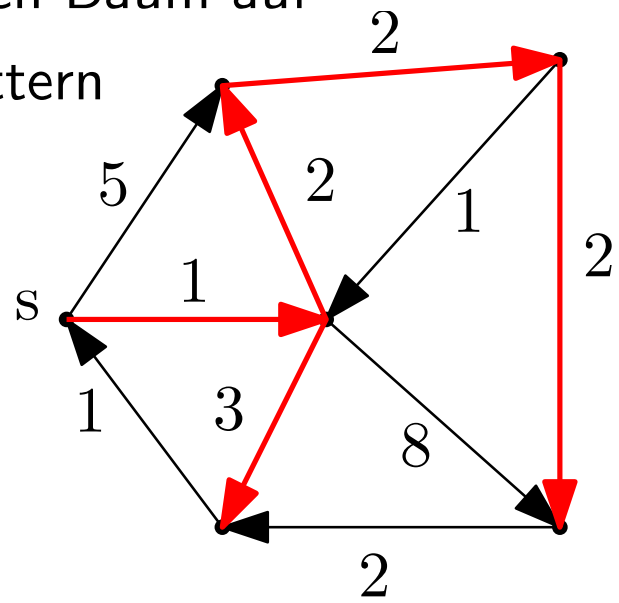
- gegeben Graph $G = (V, E)$ mit positiver Kantenfunktion $len : E \to \mathbb{R}^+$, zwei Knoten s und t
- finde den Pfad $(s, \ldots, t)$ mit $d(s, t)$ minimal



Eigenschaften von kürzesten Wegen

Beobachtungen:

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



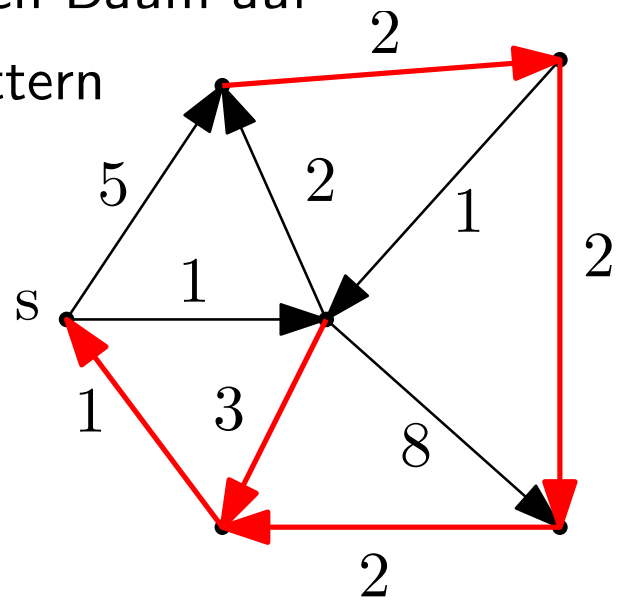
Anmerkungen:

- längste Wege: NP-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: NP-schwer

Eigenschaften von kürzesten Wegen

Beobachtungen:

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



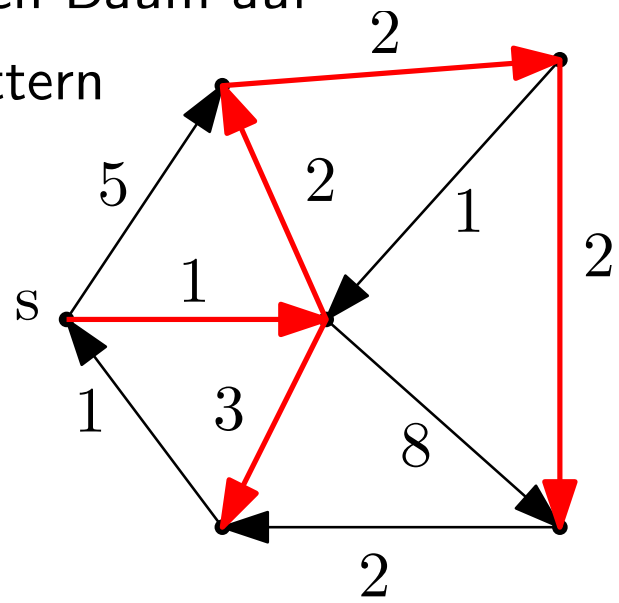
Anmerkungen:

- längste Wege: NP-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: NP-schwer

Eigenschaften von kürzesten Wegen

Beobachtungen:

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



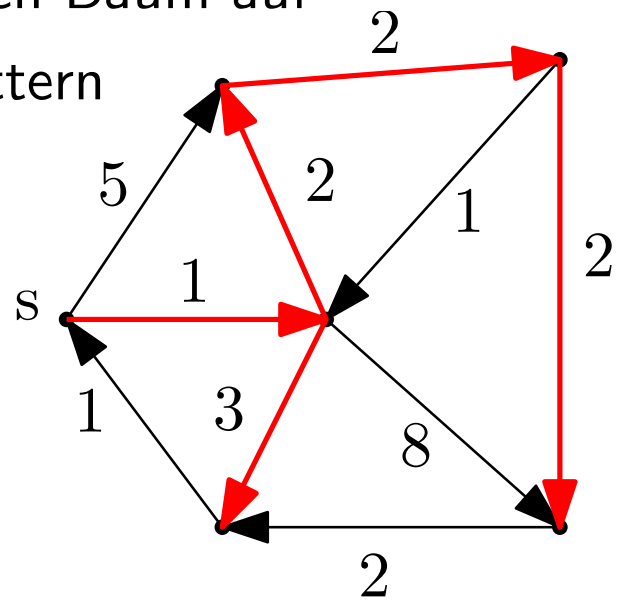
Anmerkungen:

- längste Wege: NP-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: NP-schwer

Eigenschaften von kürzesten Wegen

Beobachtungen:

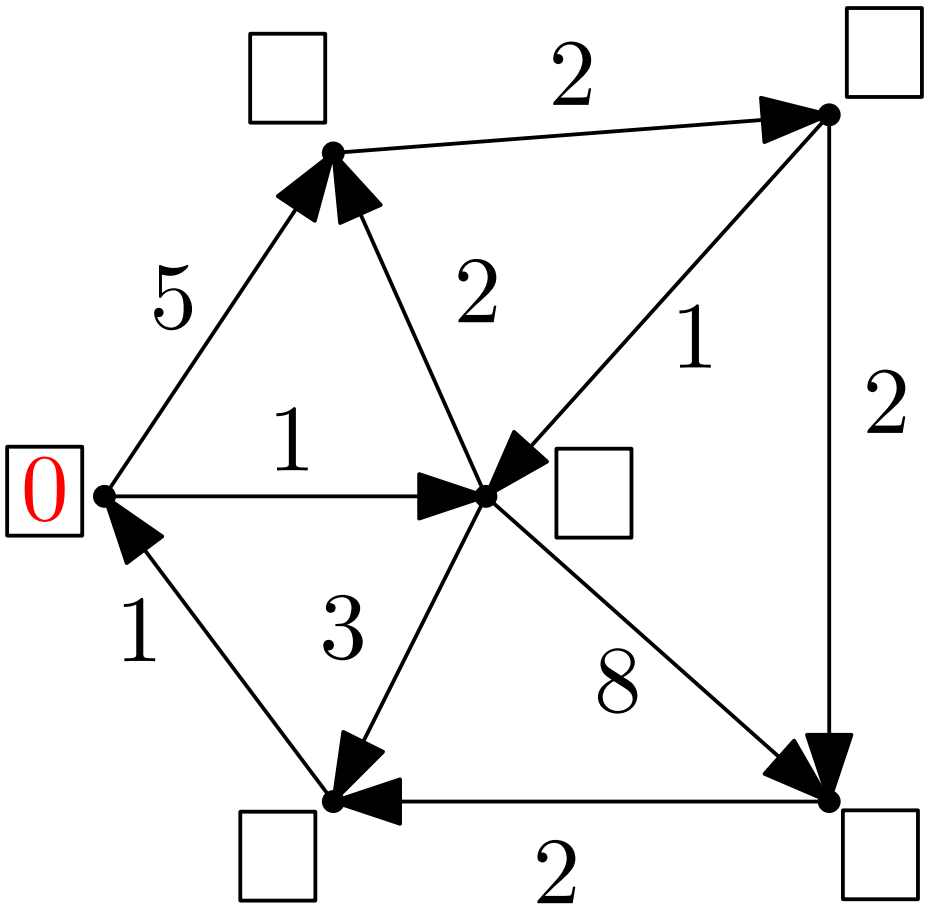
- Subpfade sind auch kürzeste Wege
- alle kürzesten Wege von s aus spannen einen Baum auf
- steigender Abstand von Wurzel zu den Blättern
- sind einfach
- $d(u, s)$ analog über eingehende Kanten
- können mit Dijkstra berechnet werden



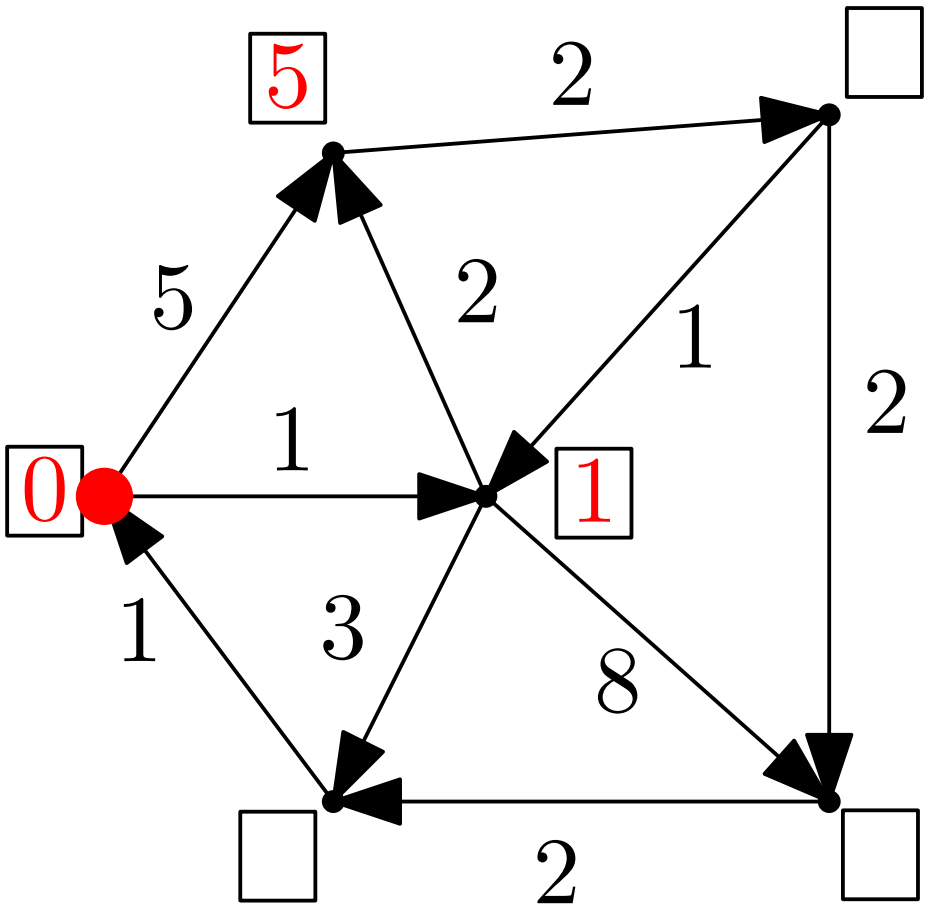
Anmerkungen:

- längste Wege: NP-schwer
- negative Kantengewichte, keine negative Zyklen: Bellmann-Ford
- negative Kantengewichte, negative Zyklen: NP-schwer

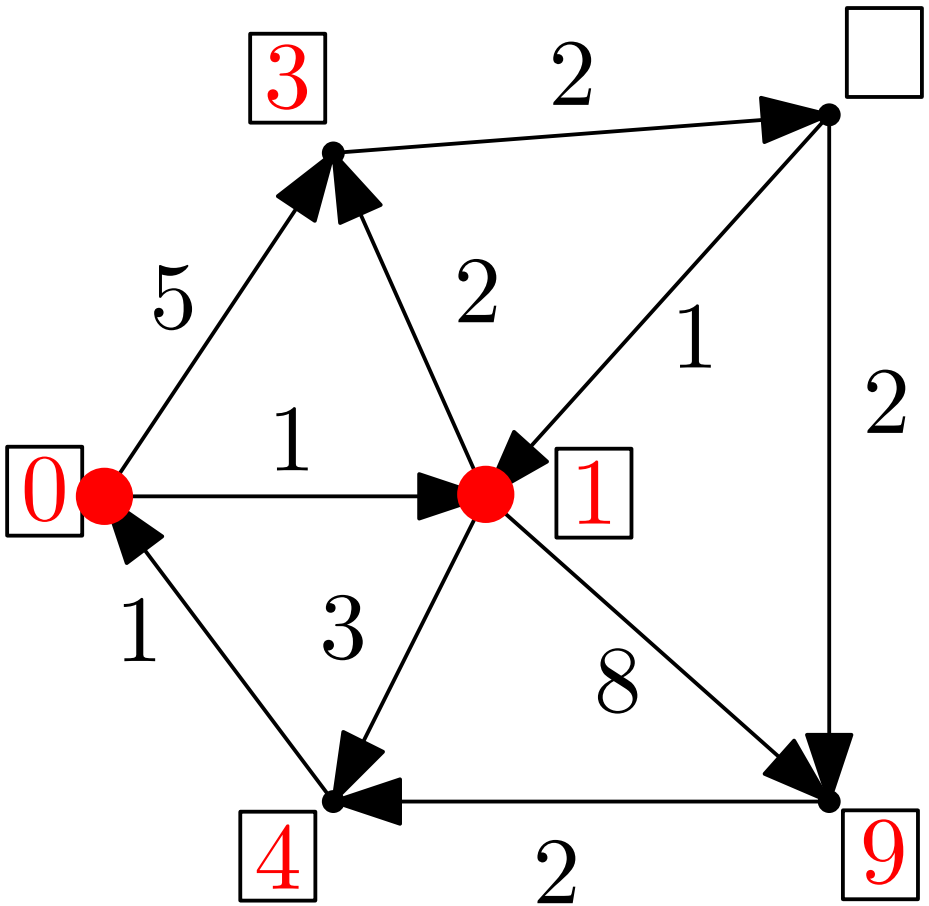
Beispiel



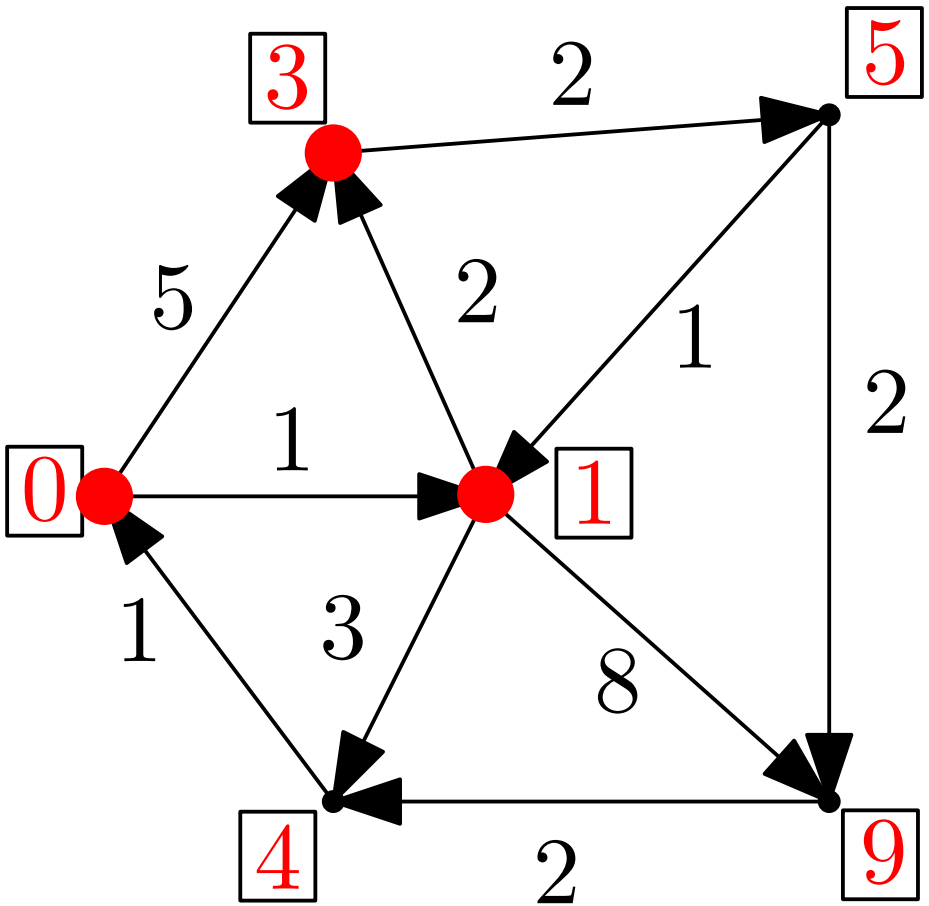
Beispiel



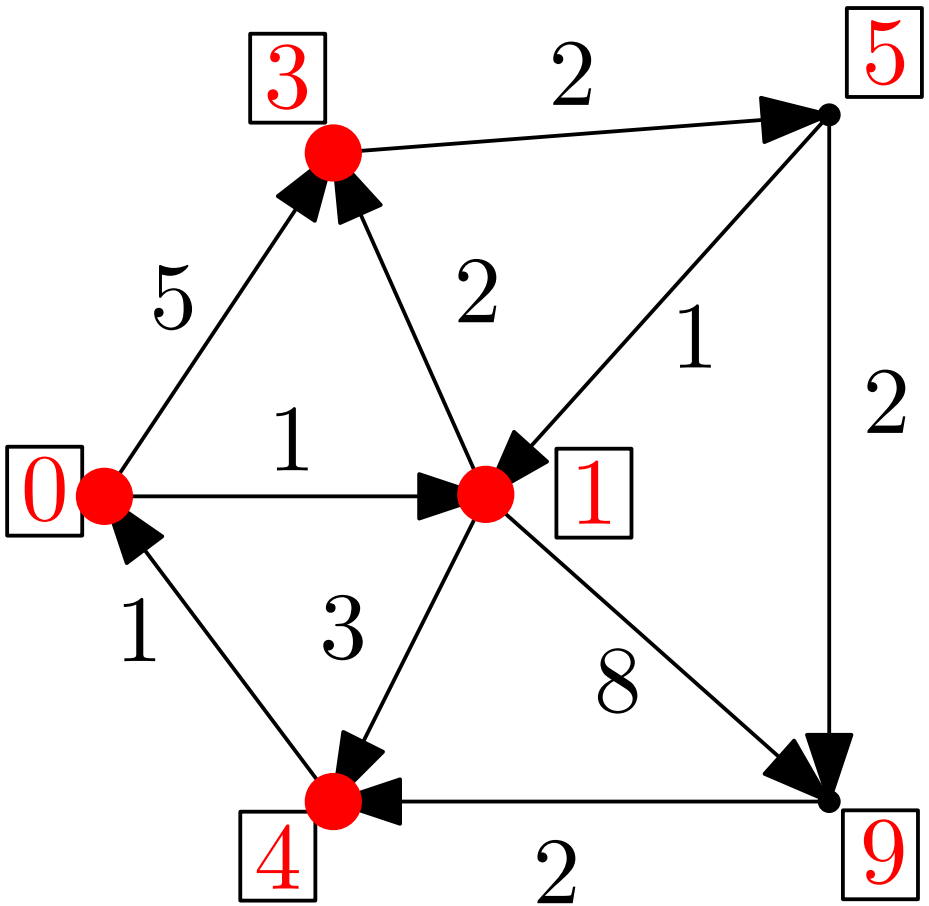
Beispiel



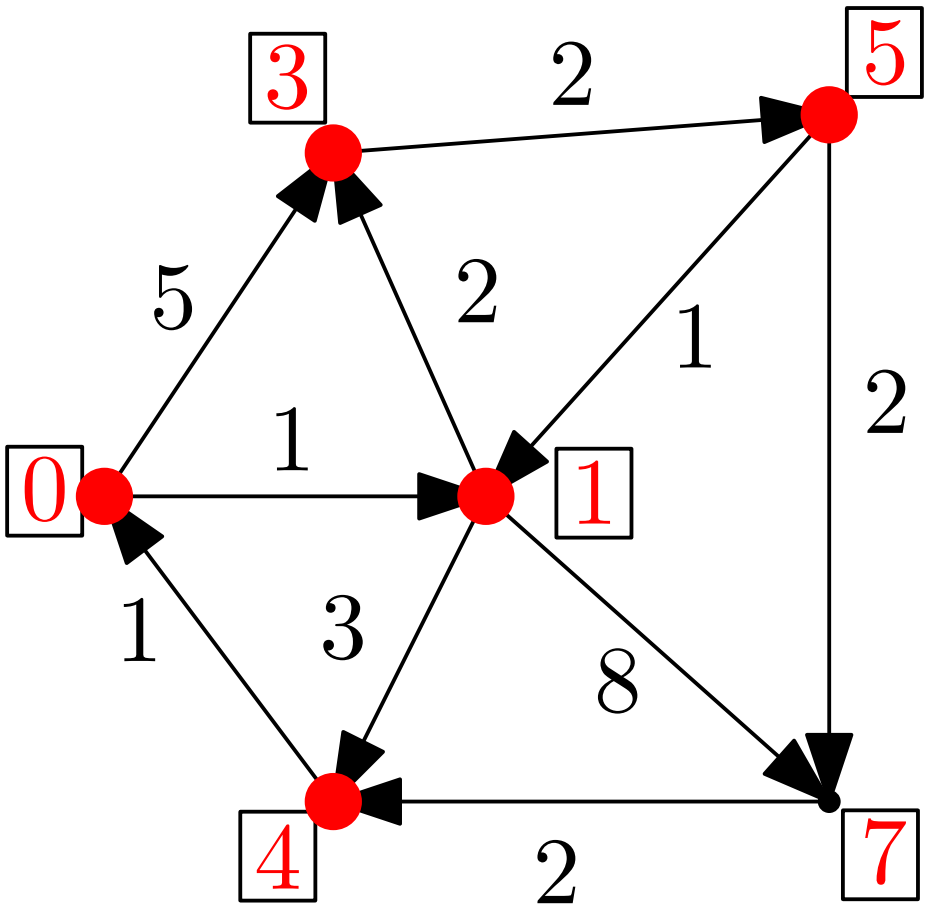
Beispiel



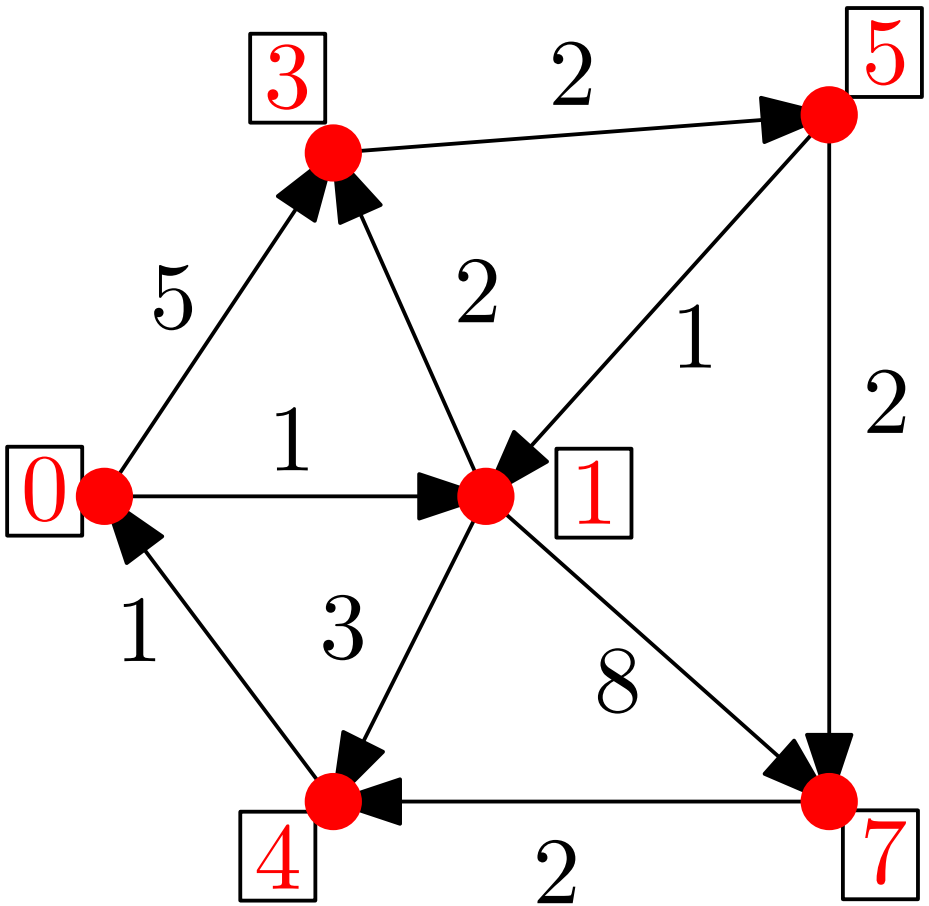
Beispiel



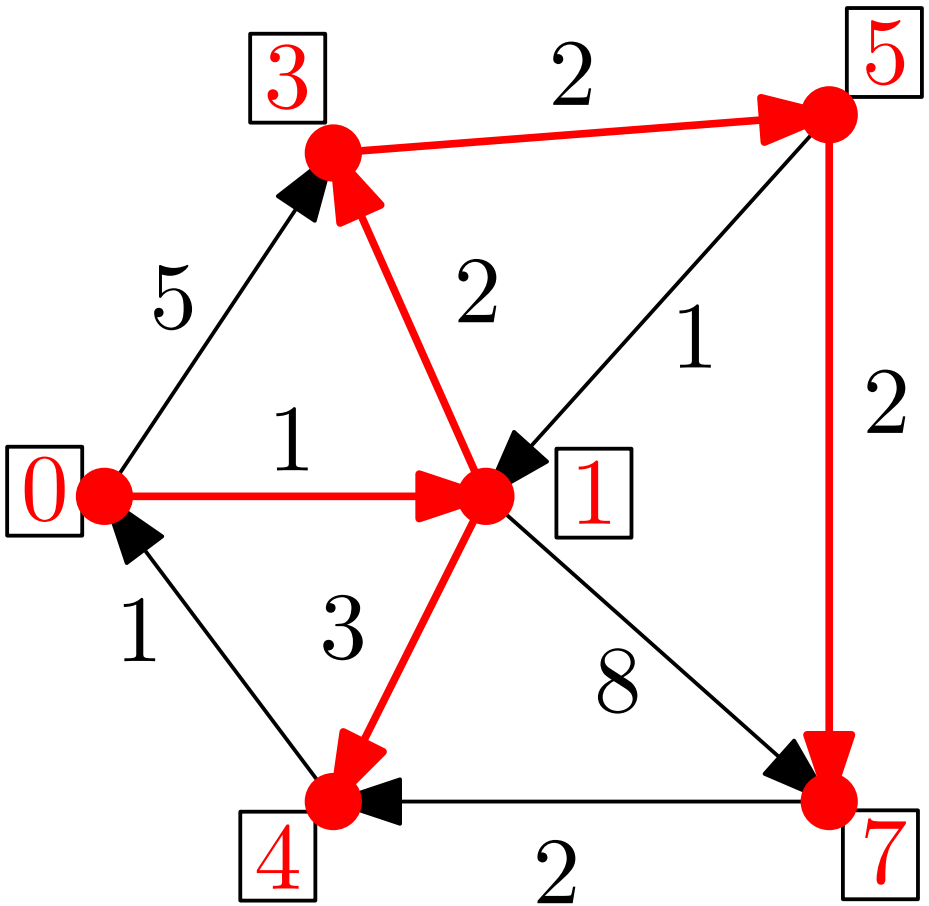
Beispiel



Beispiel



Beispiel



Dijkstra

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

```
1  for all nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL ;        // array for 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      for all 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])
```

Korrektheit

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

zwei Schritte:

- $\gg$ alle erreichbaren Knoten werden abgearbeitet
- $\gg$ wenn u abgearbeitet wird, ist $d[u] = d(s, u)$

Beweis 1:

- $\gg$ Beh.: v wird nicht bearbeitet, ist aber erreichbar
- $\Rightarrow$ es gibt kürzesten Weg $p = (v_1, \ldots, v_k), s = v_1, v = v_k$
- $\Rightarrow$ 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 $d[u] = d(s, u)$ für alle $u \in V$

zwei Schritte:

- $\gg$ alle erreichbaren Knoten werden abgearbeitet
- $\gg$ wenn u abgearbeitet wird, ist $d[u] = d(s, u)$

Beweis 2:

- $\gg$ Beh.: v wird bearbeitet mit $d[v] > d(s, v)$
- $\Rightarrow$ es gibt v_i, dass noch nicht abgearbeitet wurde, und für das gilt $d[v_i] + len(v_i, v) < d[v]$
- $\Rightarrow$ $d[v_i] < d[v]$
- $\Rightarrow$ v_i wird vor v
- $\Rightarrow$ Widerspruch

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 1
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik


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

```
1  for all nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL ;                          // n times
3  Q.clear() Q.add(s, 0);                               // make-heap once
4  while !Q.empty() do
5      u ← Q.deleteMin();                               // n times
6      for all 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 times
11             else Q.insert(v, d[v]);                  // n times
```

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

```
1  for all nodes v ∈ V do
2      d[v] = ∞, p[v] = NULL ;                          // n times
3  Q.clear() Q.add(s, 0);                               // make-heap once
4  while !Q.empty() do
5      u ← Q.deleteMin();                               // n times
6      for all 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 times
11             else Q.insert(v, d[v]);                  // n times
```

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

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	$\Theta(1)$	$\Theta(\lg k)$	$O(\lg k)$	$\Theta(1)$
`Minimum`	$\Theta(n)$	$\Theta(1)$	$O(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
`Union`	$\Theta(1)$	$\Theta(k)$	$O(\lg k)$	$\Theta(1)$
Decrease-Key	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
`Delete`	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
Dijkstra				

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 1
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik

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	$\Theta(1)$	$\Theta(\lg k)$	$O(\lg k)$	$\Theta(1)$
`Minimum`	$\Theta(n)$	$\Theta(1)$	$O(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
`Union`	$\Theta(1)$	$\Theta(k)$	$O(\lg k)$	$\Theta(1)$
Decrease-Key	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
`Delete`	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
Dijkstra	$O(n^2 + m)$	$O((n+m)\lg n)$	$O((n+m)\lg n)$	$O(n + m\lg n)$

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 1
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik

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	$\Theta(1)$	$\Theta(\lg k)$	$O(\lg k)$	$\Theta(1)$
`Minimum`	$\Theta(n)$	$\Theta(1)$	$O(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
`Union`	$\Theta(1)$	$\Theta(k)$	$O(\lg k)$	$\Theta(1)$
Decrease-Key	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
`Delete`	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
Dijkstra	$O(n^2 + m)$	$O((n+m)\lg n)$	$O((n+m)\lg n)$	$O(n + m\lg n)$
Dij $(m \in O(n))$	$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$

Daniel Delling – Algorithmen für Routenplanung – Vorlesung 1
Lehrstuhl für Algorithmik I
Institut für theoretische Informatik


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	$\Theta(1)$	$\Theta(\lg k)$	$O(\lg k)$	$\Theta(1)$
`Minimum`	$\Theta(n)$	$\Theta(1)$	$O(\lg k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
`Union`	$\Theta(1)$	$\Theta(k)$	$O(\lg k)$	$\Theta(1)$
Decrease-Key	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$\Theta(1)$
`Delete`	$\Theta(1)$	$\Theta(\lg k)$	$\Theta(\lg k)$	$O(\lg k)$
Dijkstra	$O(n^2 + m)$	$O((n+m)\lg n)$	$O((n+m)\lg n)$	$O(n + m\lg n)$
Dij ($m \in O(n)$)	$O(n^2)$	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$

Transportnetzwerke sind dünn $\Rightarrow$ binary heaps

2. Beschleunigungstechniken

Bewertungskriterien

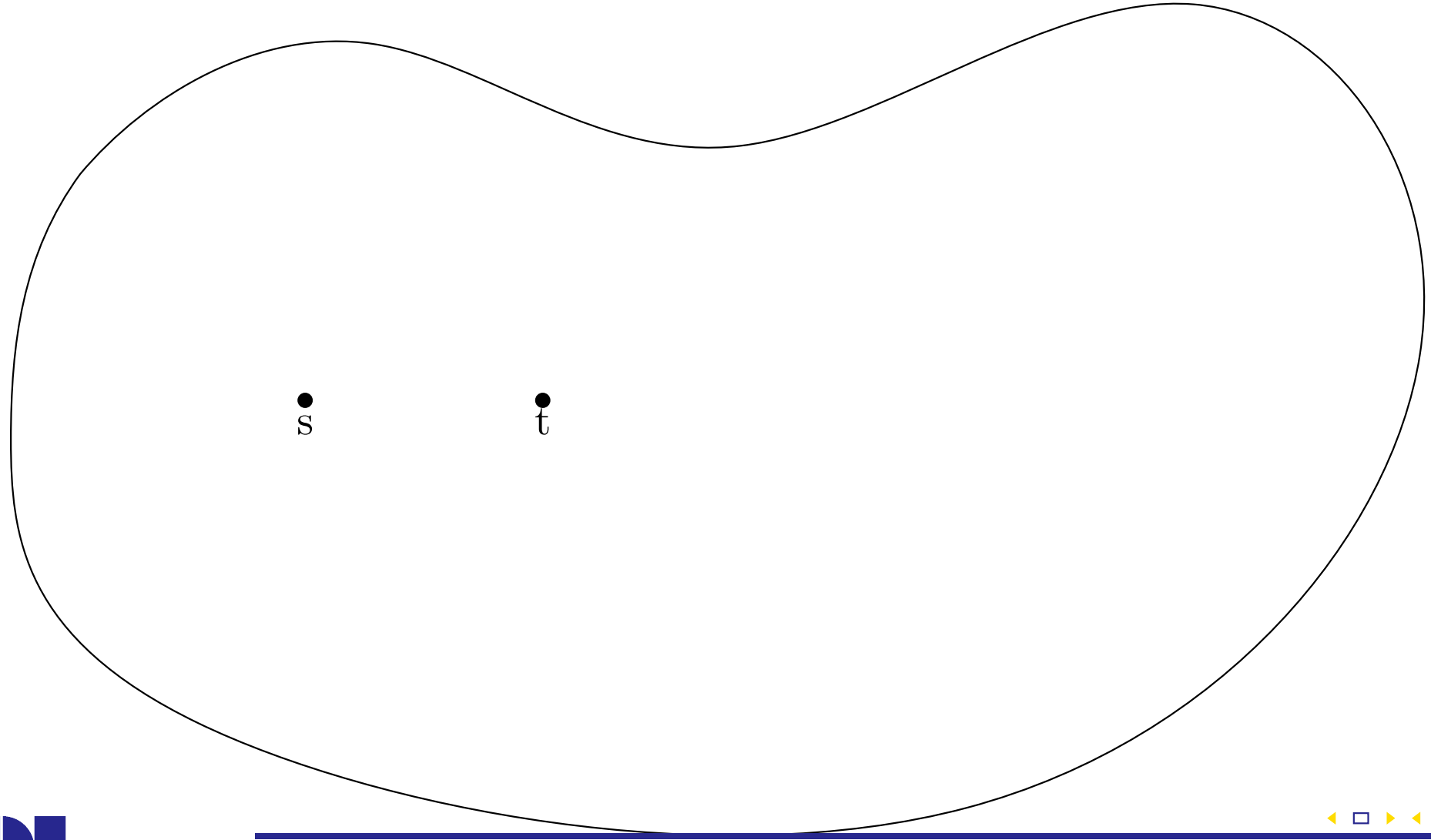
Vorberechnung:

- Platz
- Zeit

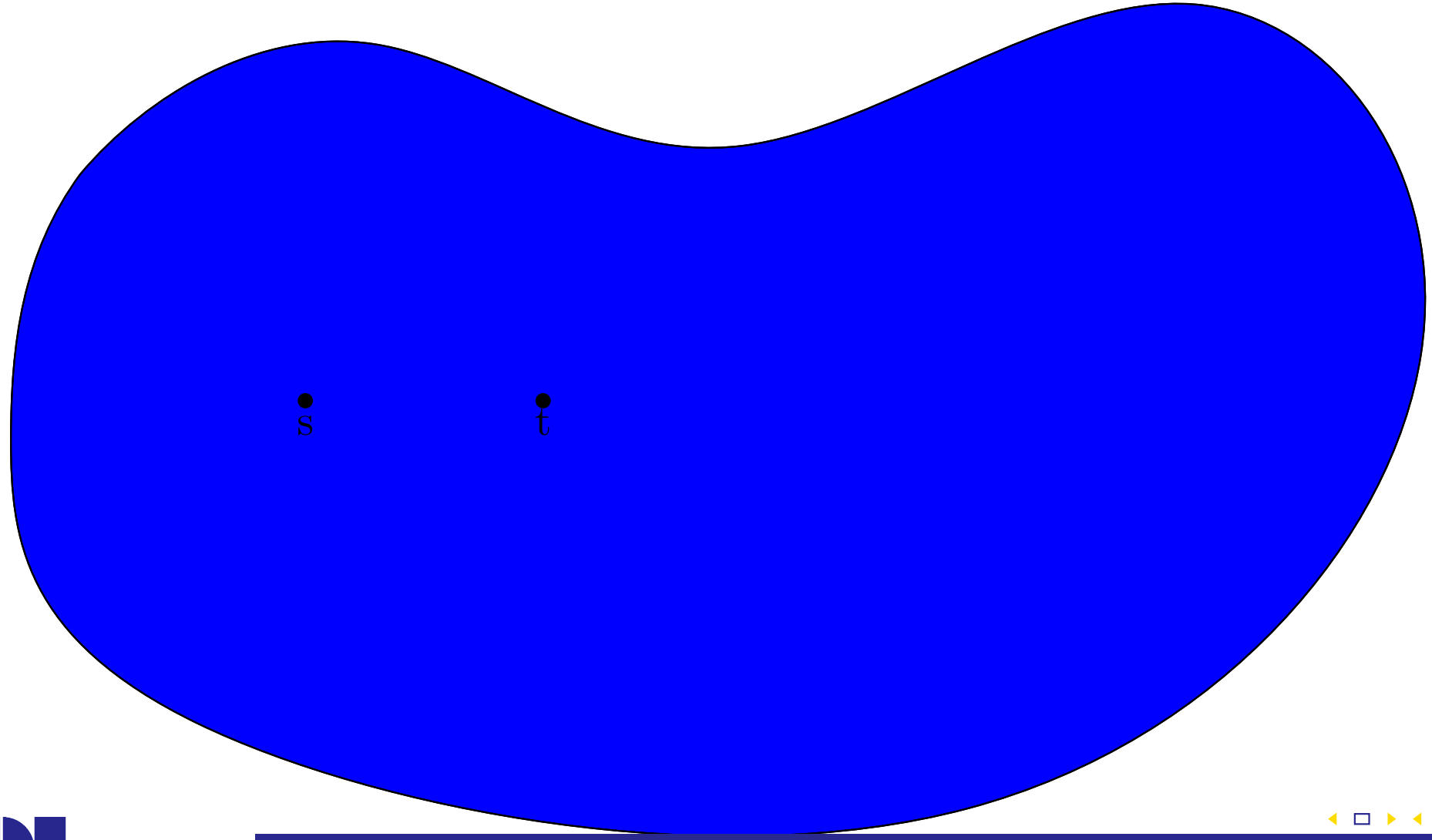
Beschleunigung der Anfragen:

- Suchraum
  - Anzahl abgearbeiteter Knoten
  - Anzahl relaxierter Kanten
  - Problem: Overhead durch Beschleunigung?
- Anfragezeit (Vergleichbarkeit?)
- auf Basis von $k > 1\,000$ Zufallsanfragen, d.h. wähle s und t zufällig (gleichverteilt)

Schematischer Suchraum von Dijkstra



Schematischer Suchraum von Dijkstra



Abbruch

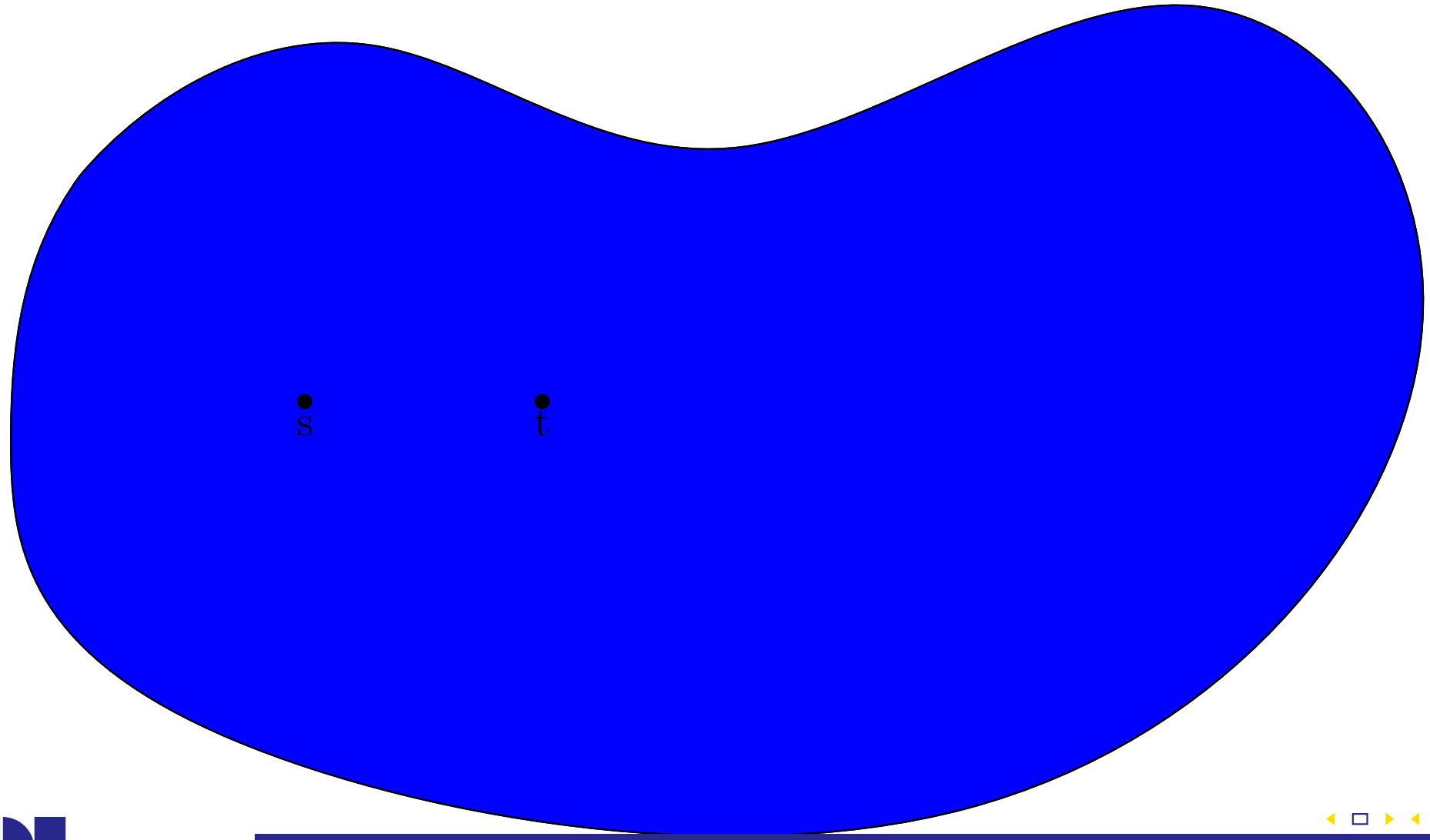
Beobachtung:

- durchsuchen des ganzen Graphen wenig sinnvoll
- vor allem wenn s und t nah beiander sind

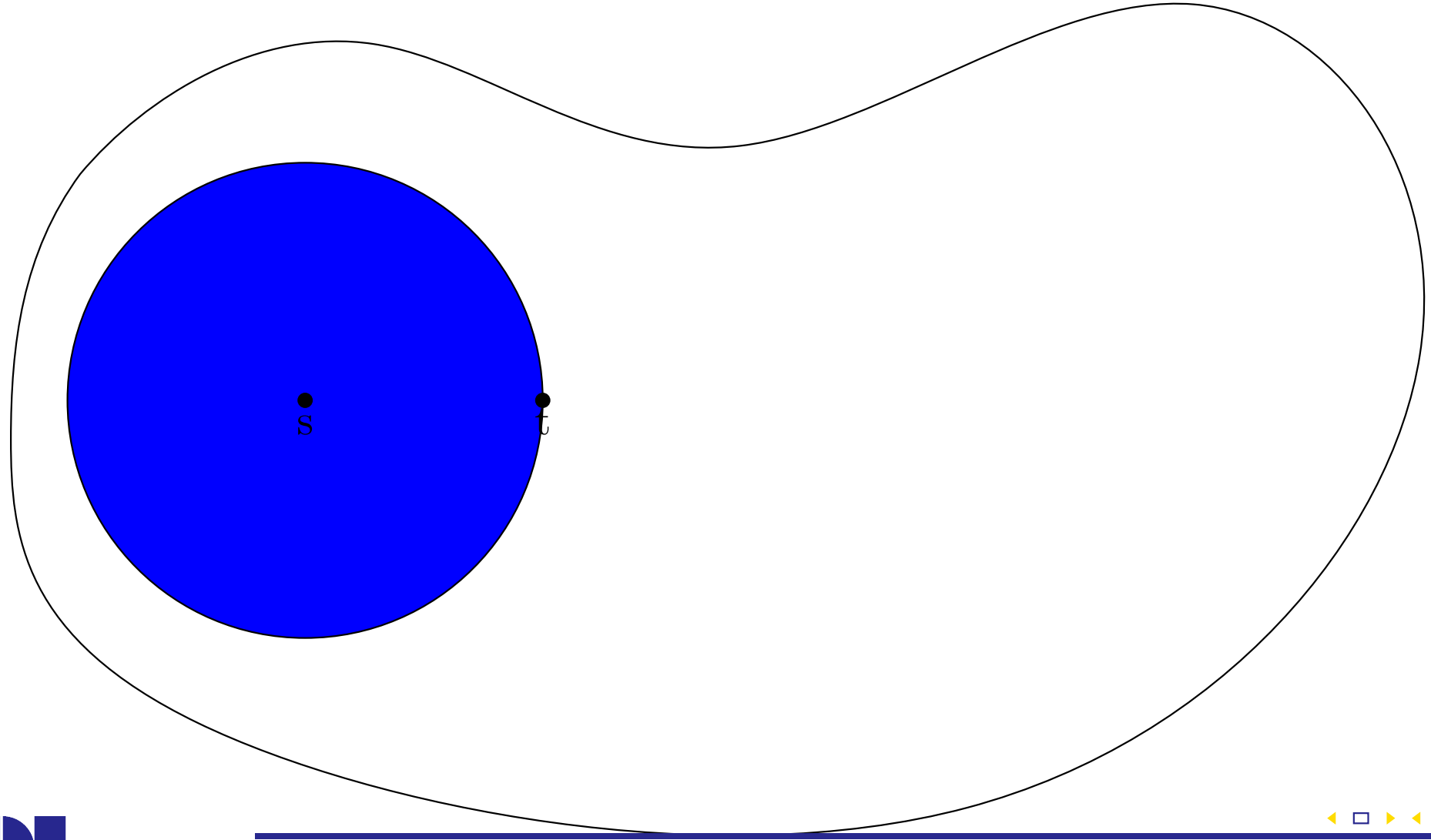
Idee:

- stop die Anfrage, sobald t aus der Queue entfernt wurde
- korrektheit analog zu Dijkstra
- im Schnitt: Beschleunigungsfaktor 2

Schematischer Suchraum von Dijkstra



Schematischer Suchraum von Dijkstra



Initialisierung

1 **for** *all nodes* $v \in V$ **do**
2 $\quad d[v] = \infty$, $p[v] = \texttt{NULL}$

Problem:

- $O(n)$
- Viele Anfragen auf gleichem Netzwerk
- wird auch ausgeführt, wenn s und t im vorigen Lauf nah beinander waren

Idee:

- lasse counter mitlaufen

Initialisierung

1 **for** *all nodes* $v \in V$ **do**
2 $\quad d[v] = \infty$, $p[v] = \texttt{NULL}$

Problem:

- $O(n)$
- Viele Anfragen auf gleichem Netzwerk
- wird auch ausgeführt, wenn s und t im vorigen Lauf nah beinander waren

Idee:

- lasse counter mitlaufen

Initialisierung

1 **for** *all nodes* $v \in V$ **do**
2 $\quad d[v] = \infty$, $p[v] = \texttt{NULL}$

Problem:

- $O(n)$
- Viele Anfragen auf gleichem Netzwerk
- wird auch ausgeführt, wenn s und t im vorigen Lauf nah beinander waren

Idee:

- lasse counter mitlaufen

Dijkstra

```
1  ++count
2  d[s] = 0
3  Q.clear(), Q.add(s, 0)
4  while !Q.empty() do
5      u ← Q.deleteMin()
6      if u = t then RETURN
7      for all edges e = (u, v) ∈ E do
8          if run[v] ≠ count then
9              d[v] ← d[u] + len(e)
10             Q.insert(v, d[v])
11             run[v] ← count
12         else if d[u] + len(e) < d[v] then
13             d[v] ← d[u] + len(e)
14             Q.decreaseKey(v, d[v])
```