

# Algorithmen für Routenplanung

1. Sitzung, Sommersemester 2013

Thomas Pajor | 15. April 2013

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK · PROF. DR. DOROTHEA WAGNER



## Dozent



### Julian Dibbelt

- Karlsruher Institut für Technologie
- Mail: `julian.dibbelt@kit.edu`
- Raum 318

## Dozent



### Thomas Pajor

- Karlsruher Institut für Technologie
- Mail: `pajor@kit.edu`
- Raum 246

## 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://illwww.itl.uni-karlsruhe.de/teaching/sommer2013/routenplanung/index>

# Vorläufige Termine

#	Tag	Datum	Dozent	#	Tag	Datum	Dozent
1	Montag	15. April	Thomas	11	Montag	03. Juni	
2	Montag	22. April		12	Montag	10. Juni	Julian
3	Mittwoch	24. April		13	Mittwoch	12. Juni	Julian
4	Montag	29. April		14	Montag	17. Juni	
5	Montag	06. Mai		15	Montag	24. Juni	
6	Mittwoch	08. Mai		16	Mittwoch	26. Juni	
7	Montag	13. Mai		17	Montag	08. Juli	
8	Mittwoch	22. Mai		18	Mittwoch	10. Juli	
9	Montag	27. Mai		19	Montag	15. Juli	
10	Mittwoch	29. Mai					

Siehe auch Vorlesungswebseite!

## Algorithmische Kartografie

- Dozenten: Dr. Martin Nöllenburg und Benjamin Niedermann
- Vorlesung: donnerstags 9:45–11:15 Uhr (SR 301)
- Übung: donnerstags 10:15–11:00 (SR 301)

## Algorithmen für planare Graphen

- Dozent: Dr. Ignaz Rutter
- Vorlesung: dienstags 14:00–15:30 Uhr (SR 301)
- Vorlesung: donnerstags 14:00–15:30 Uhr (SR 301)
- Übungs: zusätzlich an einigen Vorlesungsterminen

# 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:

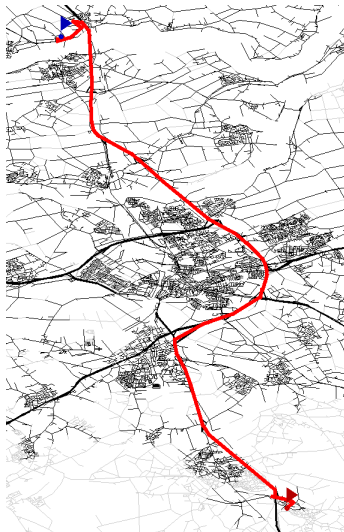
- finde die **beste** Verbindung in einem Transportnetzwerk

## 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:

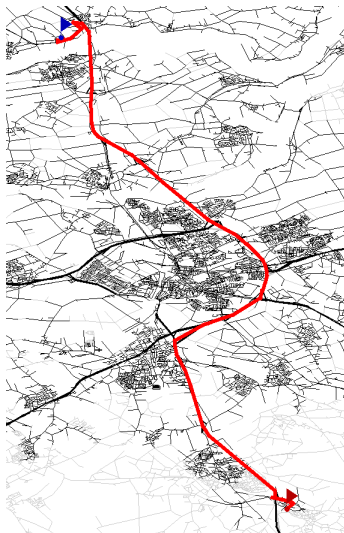
- finde die **beste** Verbindung in einem Transportnetzwerk

## 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)

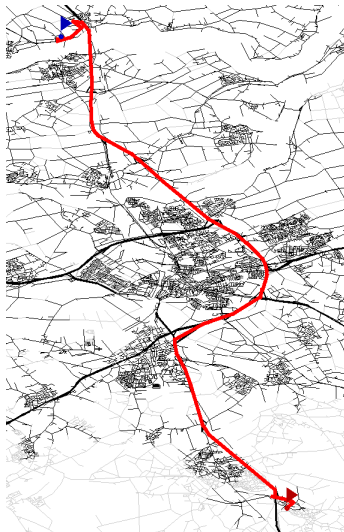


## Beobachtungen:

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

## Idee:

- Zwei-Phasen Algorithmus:
  - offline: berechne Zusatzinformation während **Vorbereitung**
  - online: **beschleunige** Berechnung mit diesen Zusatzinformationen
- drei Kriterien:
  - wenig Zusatzinformation  $O(n)$
  - kurze Vorbereitung (im Bereich Stunden/Minuten)
  - hohe Beschleunigung

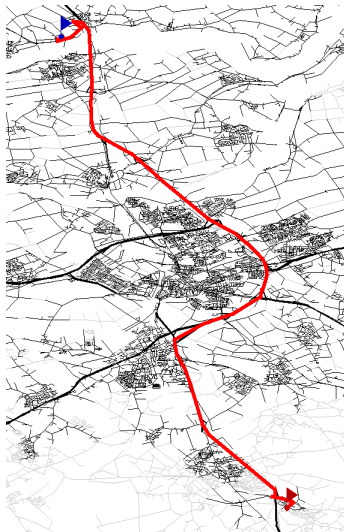


## Beobachtungen:

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

## Idee:

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

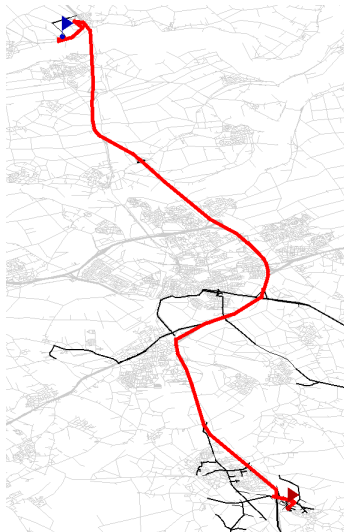


## Beobachtungen:

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

## Idee:

- Zwei-Phasen Algorithmus:
  - offline: berechne Zusatzinformation während **Vorbereitung**
  - online: **beschleunige** Berechnung mit diesen Zusatzinformationen
- drei Kriterien:
  - wenig Zusatzinformation  $\mathcal{O}(n)$
  - kurze Vorbereitung (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



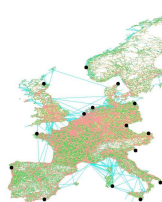
## Unser Anspruch:

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

# Ergebnisse

## Eingabe: Straßennetzwerk von Westeuropa

- 18 Mio. Knoten
- 42 Mio. Kanten



	Jahr	VORBERECHNUNG		ANFRAGE	
		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

- Algorithm Engineering + ein bisschen Theorie
- Beschleunigungstechniken
- Implementierungsdetails
- Ergebnisse auf Real-Welt Daten
- aktuellster Stand der Forschung (Veröffentlichungen bis 2013)
- ideale Grundlage für Bachelor, Master und Diplomarbeiten

## keine Algorithmen III

- Vertiefung von kürzesten Wegen (Dijkstra)
  - Grundlagen sind Stoff von Algo I/II;  
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



## 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 Distanztabelle
- Kombinationen

## 3. Theorie

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

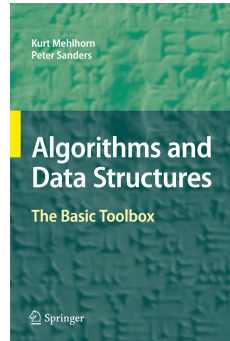
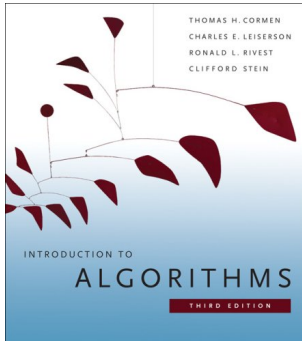
## 4. Fortgeschrittene Szenarien

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

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

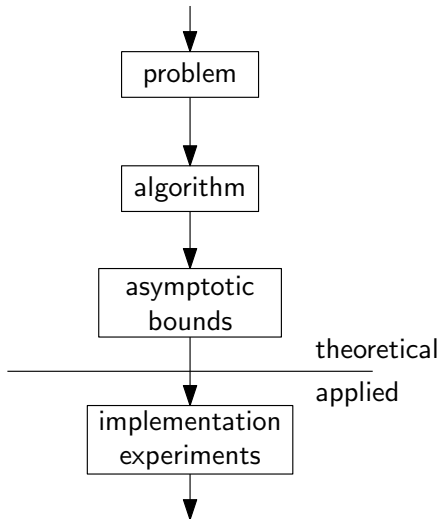
**Vertiefungsgebiet:** Algorithmentechnik, (Theoretische Grundlagen)

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



# 1. Grundlagen





# Lücke Theorie vs. Praxis

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

hier:

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

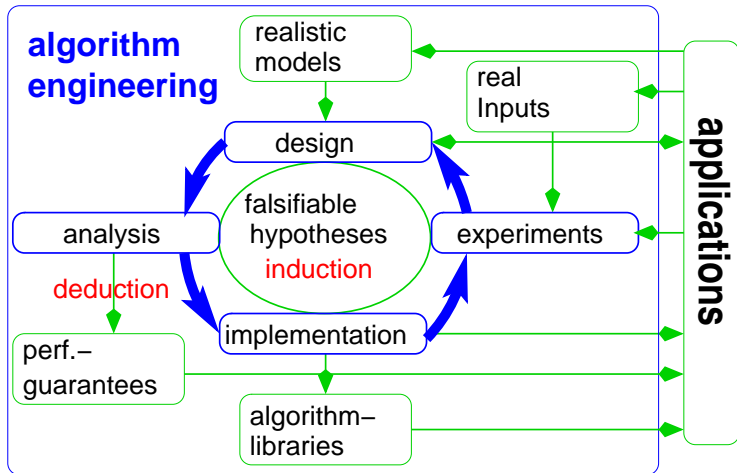
# Lücke Theorie vs. Praxis

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

## hier:

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





- **Graph:** Tupel  $G := (V, E)$

- endliche Knotenmenge  $V$
- endliche Kantenmenge  $E$
- $n := |V|, m := |E|$

- **ungerichtet:** Kanten sind Knotenpaar, d.h.

$$E \subseteq \binom{V}{2} = \{\{u, v\} \mid u \neq v, u, v \in V\}$$

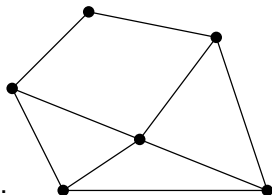
- Grad:  $\deg(u) = \sum_{\{u,v\} \in E}$

- **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)



- **Graph:** Tupel  $G := (V, E)$

- endliche Knotenmenge  $V$
- endliche Kantenmenge  $E$
- $n := |V|, m := |E|$

- **ungerichtet:** Kanten sind Knotenpaar, d.h.

$$E \subseteq \binom{V}{2} = \{\{u, v\} \mid u \neq v, u, v \in V\}$$

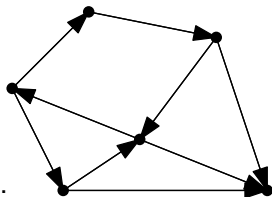
- Grad:  $\deg(u) = \sum_{\{u,v\} \in E} 1$

- **gerichtet:** Kanten sind geordnete Paare, d.h.

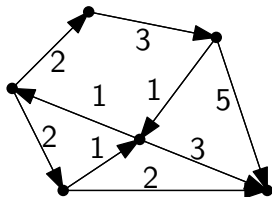
$$E \subseteq \{(u, v) \mid u \neq v, u, v \in V\}$$

- Ausgangsgrad:  $\deg_{\text{out}}(u) = \sum_{(u,v) \in E} 1$
- Eingangsgrad:  $\deg_{\text{in}}(u) = \sum_{(v,u) \in E} 1$

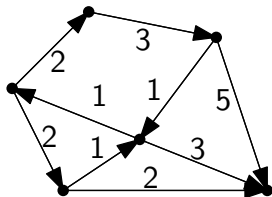
- **einfach:** keine Multi-Kanten ( $E$  ist normale Menge)



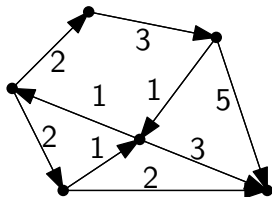
- **gewichtet:** Kantengewichtsfunktion
  - erstmalig  $l: E \rightarrow \mathbb{R}^+$
- **dünn:**  $m \in \mathcal{O}(n)$
- **planar:** kreuzungsfrei einbettbar



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



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

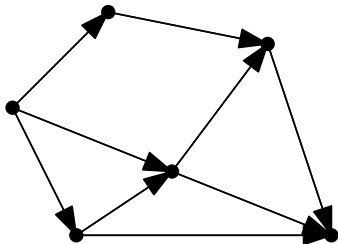


## DAG:

- directed acyclic graph
- gerichtet, zyklensfrei

## Baum:

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

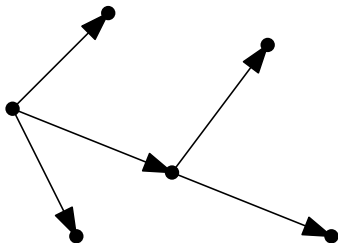


## DAG:

- directed acyclic graph
- gerichtet, zyklonfrei

## Baum:

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





## Weg:

Zu gegebenen Knoten  $s, t \in V$  ist  $P := (v_1 = s, v_2, \dots, v_k = t)$  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  $P := (s, \dots, t)$  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.

## Weg:

Zu gegebenen Knoten  $s, t \in V$  ist  $P := (v_1 = s, v_2, \dots, v_k = t)$  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  $P := (s, \dots, t)$  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.

## Weg:

Zu gegebenen Knoten  $s, t \in V$  ist  $P := (v_1 = s, v_2, \dots, v_k = t)$  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  $P := (s, \dots, t)$  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.

## Weg:

Zu gegebenen Knoten  $s, t \in V$  ist  $P := (v_1 = s, v_2, \dots, v_k = t)$  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  $P := (s, \dots, t)$  mit minimaler Länge  $|P|$ .

## Distanz:

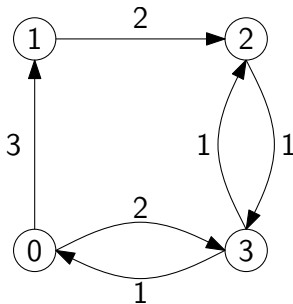
Die Distanz  $d(s, t)$  für zwei Knoten  $s, t \in V$  ist definiert als

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

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

## Drei klassische Ansätze:

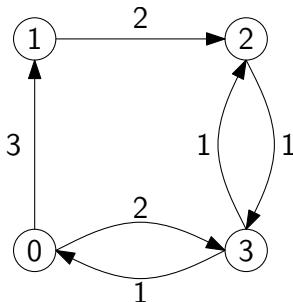
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



## Drei klassische Ansätze:

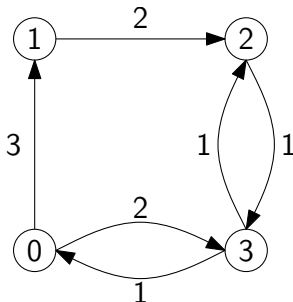
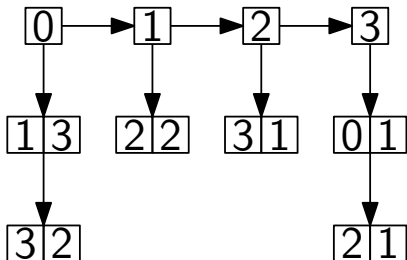
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

	0	1	2	3
0	—	3	—	2
1	—	—	2	—
2	—	—	—	1
3	1	—	1	—



## Drei klassische Ansätze:

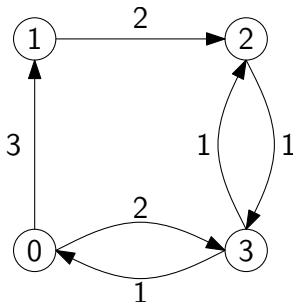
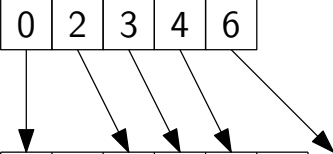
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



## Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

firstEdge	0	2	3	4	6
targetNode	1	3	2	3	2
weight	3	2	2	1	1





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 (Speicherlayout)	+	-	+
Updates (topologisch)	+	+	-
Updates (Gewicht)	+	+	+

## Fragen:

- Was brauchen wir?
- Was muss nicht super effizient sein?
- erstmal Modelle anschauen!

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 (Speicherlayout)	+	-	+
Updates (topologisch)	+	+	-
Updates (Gewicht)	+	+	+

## Fragen:

- Was brauchen wir?
- Was muss nicht super effizient sein?
- erstmal Modelle anschauen!

# Modellierung (Straßengraphen)

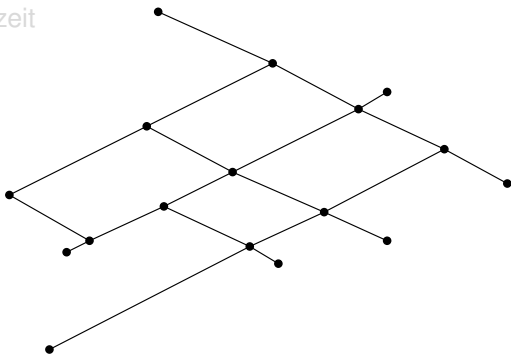


# Modellierung (Straßengraphen)



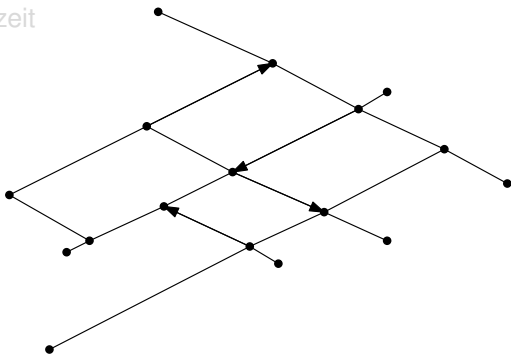
# Modellierung (Straßengraphen)

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



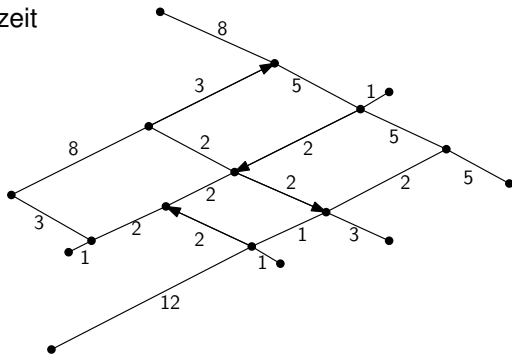
# Modellierung (Straßengraphen)

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



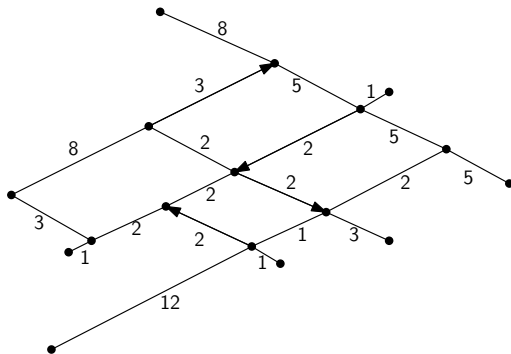
# Modellierung (Straßengraphen)

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



# Modellierung (Straßengraphen)

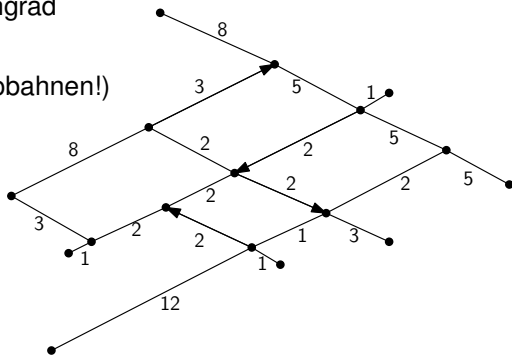
Eigenschaften (sammeln):





## Eigenschaften:

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



- 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

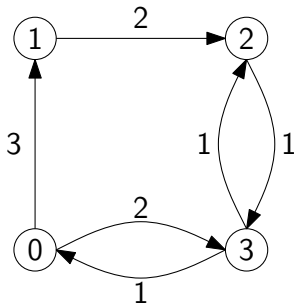
- dünn (!)
- gerichtet
- geringer Knotengrad
- meist verborgene Hierarchie (Autobahnen, ICE)
- Einbettung vorhanden (fast planar?)
- teilweise zeitabhängig
- Kantengewichte nicht-negativ
- zusammenhängend

## Diskussion:

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

## Problem:

- ein- und ausgehende Kanten
- (fast) ungerichtet



## 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

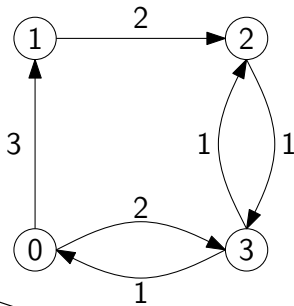
3	2	1	3	2	2	1	1	2	1
---	---	---	---	---	---	---	---	---	---

isIncoming

-	-	✓	✓	-	✓	✓	-	✓	✓
---	---	---	---	---	---	---	---	---	---

isOutgoing

✓	✓	-	-	✓	-	✓	✓	-	✓
---	---	---	---	---	---	---	---	---	---



# 2. Kürzeste Wege

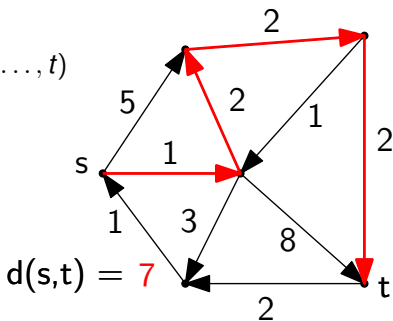


## Gegeben:

Graph  $G = (V, E, \text{len})$  mit positiver Kantenfunktion  $\text{len} : E \rightarrow \mathbb{R}_{\geq 0}$ ,  
Knoten  $s, t \in V$

## Mögliche Aufgaben

- Berechne Distanz  $d(s, t)$
- Finde kürzesten  $s$ - $t$ -Pfad  $P := (s, \dots, t)$



## Azyklichkeit:

- Kürzeste Wege sind zyklenfrei

## Aufspannungseigenschaft:

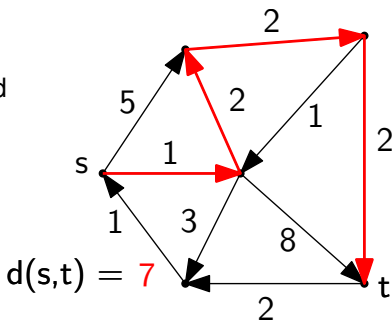
- Alle kürzesten Wege von  $s$  aus bilden DAG bzw. Baum

## Vererbungseigenschaft:

- Subwege von kürzesten Wegen sind kürzeste Wege

## Abstand:

- Steigender Abstand von Wurzel zu Blättern





**Priority Queue:** Datenstruktur die folgende Operationen erfüllt.

Operation	Beschreibung
Empty()	Prüft ob die Queue leer ist
Clear()	Löscht alle Elemente aus der Queue
Insert( $e, k$ )	Fügt Element $e$ mit Key $k$ ein
DecreaseKey( $e, k$ )	Senkt den Key von Element $e$ auf den Wert $k$
DeleteMin()	Löscht das Element mit kleinstem Key und gibt es zurück
MinKey()	Gibt den Key des kleinsten Elements zurück

Näheres in jedem Algorithmik-Standardlehrbuch.

---

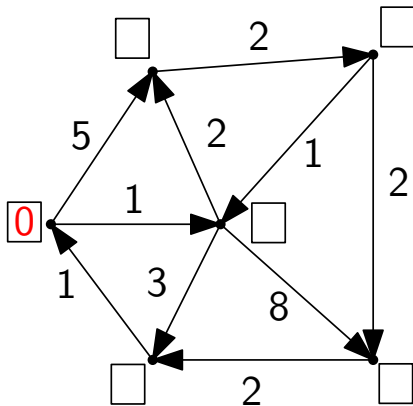
DIJKSTRA( $G = (V, E), s$ )

---

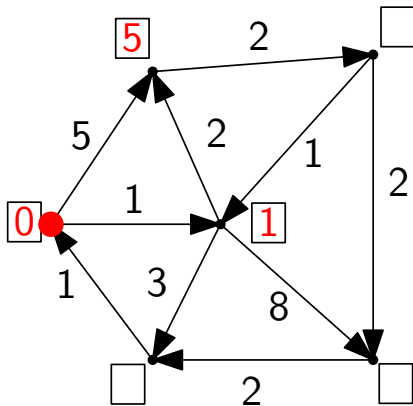
```
1 forall the nodes  $v \in V$  do
2    $d[v] = \infty, p[v] = \mathbf{NULL}$            // distances, parents
3  $d[s] = 0$ 
4  $Q.clear(), Q.insert(s, 0)$                // container
5 while  $!Q.empty()$  do
6    $u \leftarrow Q.deleteMin()$            // settling node u
7   forall the edges  $e = (u, v) \in E$  do
8     // relaxing edges
9     if  $d[u] + \text{len}(e) < d[v]$  then
10       $d[v] \leftarrow d[u] + \text{len}(e)$ 
11       $p[v] \leftarrow u$ 
12      if  $v \in Q$  then  $Q.decreaseKey(v, d[v])$ 
13      else  $Q.insert(v, d[v])$ 
```

---

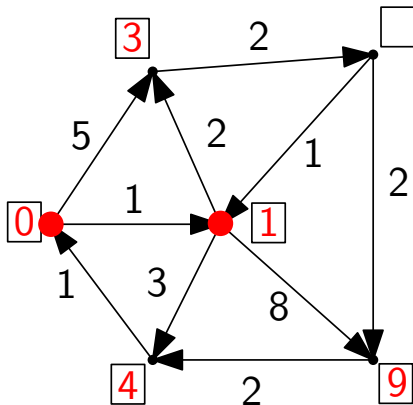
# Beispiel



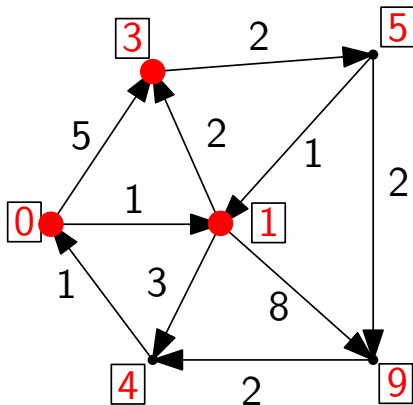
# Beispiel



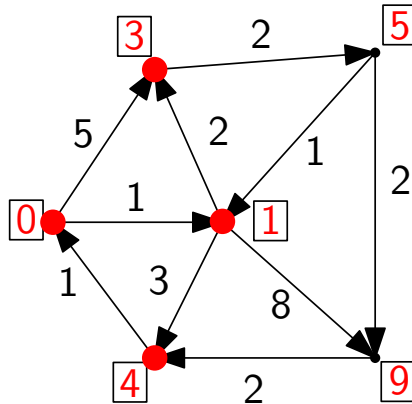
# Beispiel



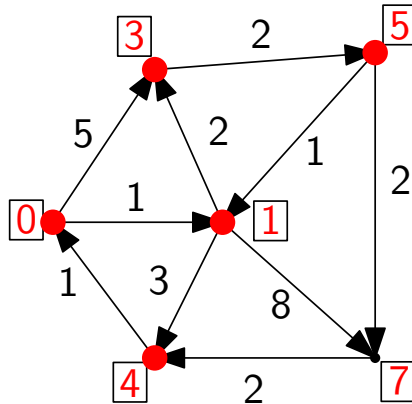
# Beispiel



# Beispiel

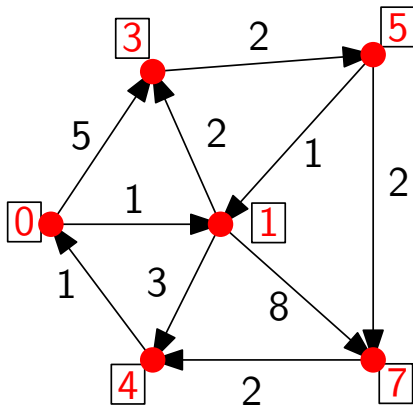


# Beispiel





# Beispiel



# Beispiel

