

Algorithmen für Routenplanung

1. Vorlesung, Sommersemester 2019

Tim Zeitz | 24. April 2019

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



Vorlesung

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

Prüfung

- Prüfbar im Master
- Im Master: 5 ECTS
- VF: Algorithmentechnik

Vorlesungswebseite:

<https://i11www.itl.kit.de/teaching/sommer2019/routenplanung/index>



Dorothea
Wagner



Valentin
Buchhold



Jonas
Sauer



Tim
Zeitz



Tobias
Zündorf

Vorlesungswebseite:

[http://i11www.itl.kit.edu/teaching/sommer2019/
routenplanung/index](http://i11www.itl.kit.edu/teaching/sommer2019/routenplanung/index)

Terminübersicht:

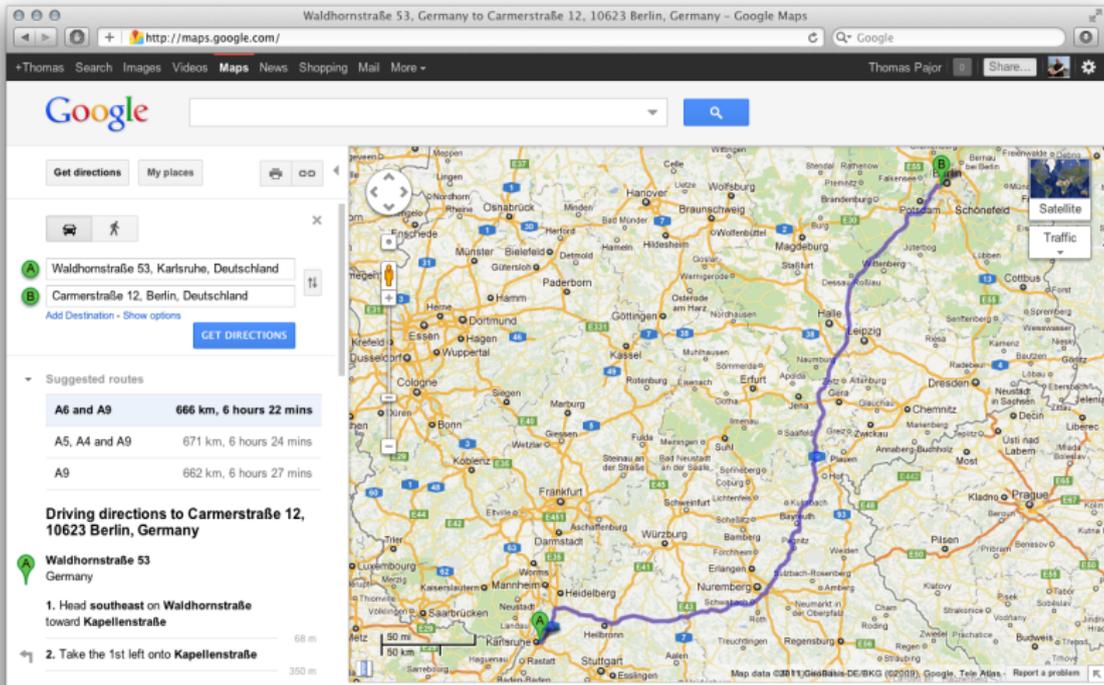
Montag	22.4.	–	<u>Keine Vorlesung</u>
Mittwoch	24.4.	Tim	Einführung, Grundlagen, Dijkstras Algorithmus
Montag	29.4.	Tim	Kontraktion & TopoCore
Mittwoch	1.5.	–	<u>Keine Vorlesung</u>
Montag	6.5.	Tim	A*, ALT, CALT
Mittwoch	8.5.	Tim	Arc-Flags, SHARC
Montag	13.5.	Valentin	Reach, MLO/CRP

...

Routenplanung in der Anwendung



Routenplanung in der Anwendung



Waldhornstraße 53, Germany to Carmerstraße 12, 10623 Berlin, Germany - Google Maps

http://maps.google.com/

Google

Get directions My places

Waldhornstraße 53, Karlsruhe, Deutschland

Carmerstraße 12, Berlin, Deutschland

GET DIRECTIONS

Suggested routes

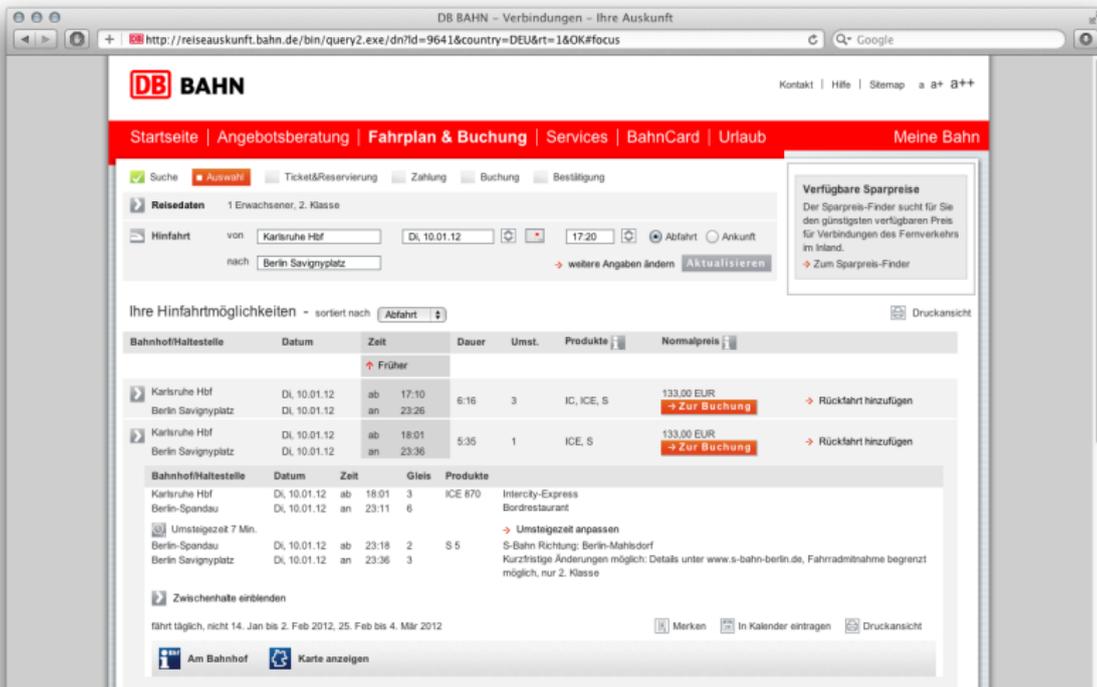
Route	Distance	Time
A6 and A9	666 km	6 hours 22 mins
A5, A4 and A9	671 km	6 hours 24 mins
A9	662 km	6 hours 27 mins

Driving directions to Carmerstraße 12, 10623 Berlin, Germany

- Head southeast on Waldhornstraße toward Kapellenstraße 68 m
- Take the 1st left onto Kapellenstraße 350 m

Map data ©2011, GeoBasis-DE/BKG (©2009), Google, Tele Atlas. Report a problem

Routenplanung in der Anwendung



DB BAHN

Kontakt | Hilfe | Stempal a+ B++

Startseite | Angebotsberatung | Fahrplan & Buchung | Services | BahnCard | Urlaub Meine Bahn

Suche **Auswahl** Ticket&Reservierung Zahlung Buchung Bestätigung

Reisedaten 1 Erwachsener, 2. Klasse

Hinfahrt von Karlsruhe Hbf Di, 10.01.12 17:20 Abfahrt Ankunft
nach Berlin Savignyplatz weitere Angaben ändern Aktualisieren

Verfügbare Sparpreise
Der Sparpreis-Finder sucht für Sie den günstigsten verfügbaren Preis für Verbindungen des Fernverkehrs im Inland.
Zum Sparpreis-Finder

Ihre Hinfahrtmöglichkeiten - sortiert nach Abfahrt

Bahnhof/Haltestelle	Datum	Zeit	Dauer	Umsst.	Produkte	Normalpreis
Karlsruhe Hbf	Di, 10.01.12	ab 17:10				
Berlin Savignyplatz	Di, 10.01.12	an 23:26	6:16	3	IC, ICE, S	133,00 EUR
						Zur Buchung Rückfahrt hinzufügen
Karlsruhe Hbf	Di, 10.01.12	ab 18:01	5:35	1	ICE, S	133,00 EUR
Berlin Savignyplatz	Di, 10.01.12	an 23:36				Zur Buchung Rückfahrt hinzufügen

Bahnhof/Haltestelle	Datum	Zeit	Gleis	Produkte
Karlsruhe Hbf	Di, 10.01.12	ab 18:01	3	ICE 670
Berlin-Spandau	Di, 10.01.12	an 23:11	6	Intercity-Express Bordrestaurant

Umsteigezeit 7 Min.
Umsteigezeit anpassen
S-Bahn Richtung: Berlin-Mahlsdorf
Kurzfristige Änderungen möglich: Details unter www.s-bahn-berlin.de, Fahrradmitnahme begrenzt möglich, nur 2. Klasse

Zwischenhalte einblenden
fährt täglich, nicht 14. Jan bis 2. Feb 2012, 25. Feb bis 4. Mär 2012

Merken In Kalender eintragen Drucksicht

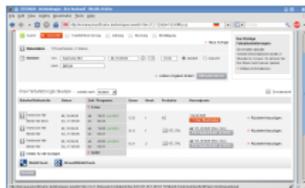
Am Bahnhof Karte anzeigen

Routenplanung in der Anwendung



Wichtiger Anwendungsbereich

- Navigationssysteme
- Kartendienste: Google Maps, Bing Maps, ...
- Fahrplanauskunftssysteme



Viele kommerzielle Systeme

- Nutzen **heuristische** Methoden
- Betrachten nur Teile des Transportnetzwerkes
- Geben keine **Qualitätsgarantien**

Wir untersuchen Methoden zur **Routenplanung** in Transportnetzwerken mit **beweisbar optimalen** Lösungen

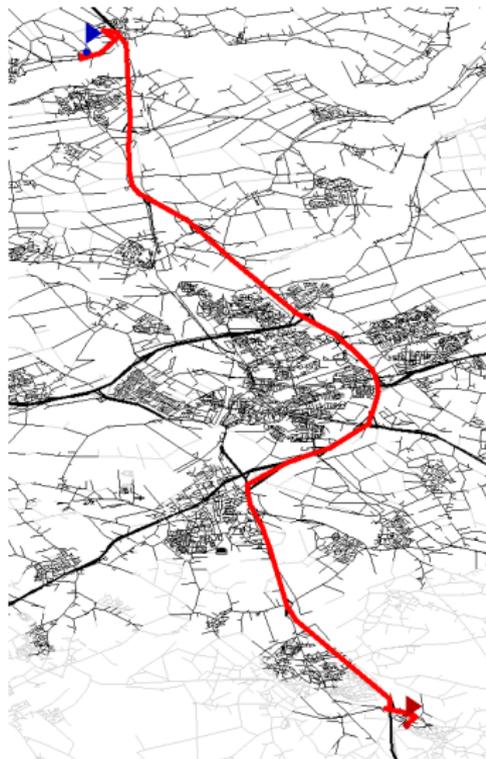
Problemstellung

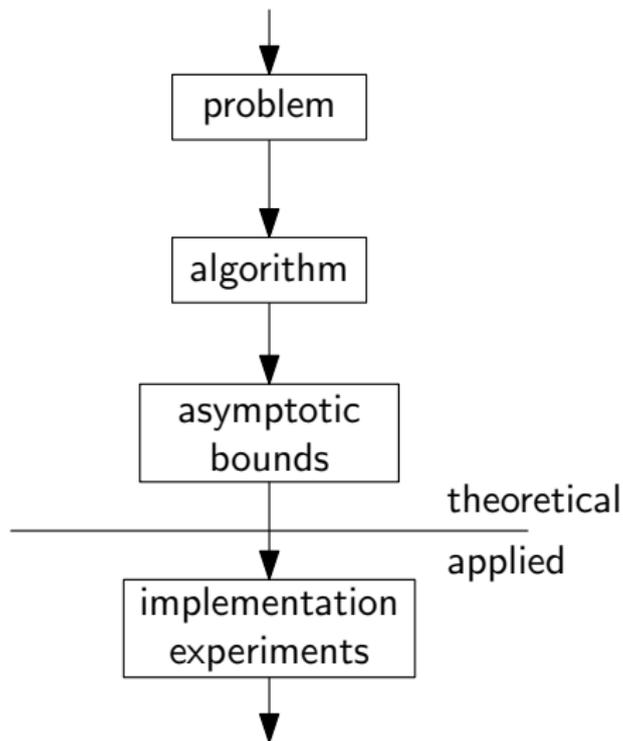
Anfrage:

- Finde **beste** Verbindung in Transportnetz

Modellierung:

- Netzwerk als Graph $G = (V, E)$
- Kantengewichte sind **Reisezeit**
- **Kürzeste** Wege in G entsprechen **schnellsten** Verbindungen





Problemstellung

Anfrage:

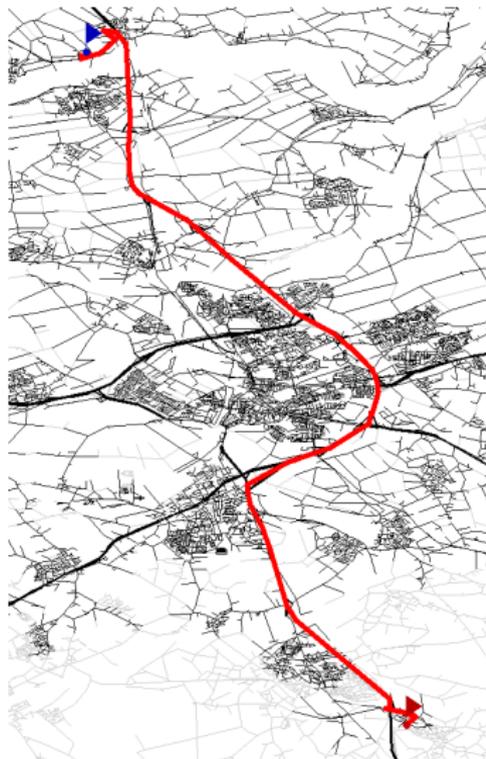
- Finde **beste** Verbindung in Transportnetz

Modellierung:

- Netzwerk als Graph $G = (V, E)$
- Kantengewichte sind **Reisezeit**
- **Kürzeste** Wege in G entsprechen **schnellsten** Verbindungen
- Klassisches Problem (Dijkstra [Dij59])

Probleme:

- Transportnetzwerke sind **riesig**
- Dijkstras Algorithmus zu **langsam**
(> 1 Sekunde)

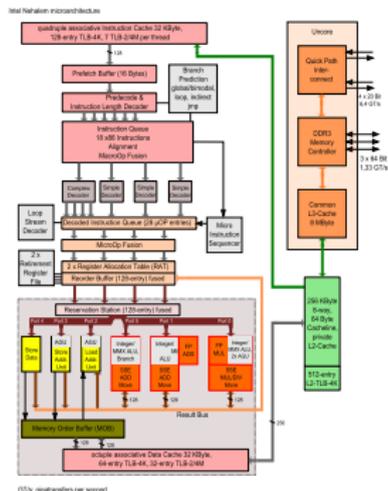


Einige Fakten:

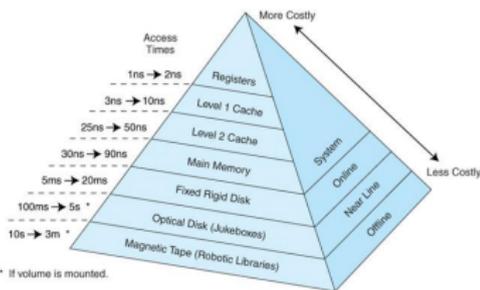
- steile Speicherhierarchie
- viele Kerne
- mehr Kerne als Speichercontroller
- Cache-Kohärenz
- SIMD (SSE, AVX)

Hauptausforderungen:

- Speicherzugriff/Datenlokalität
- Parallelisierung
 - Nicht nur race-conditions, etc.
 - auch Speicherzugriff z.B. "false-sharing" (Cache-Kohärenz)



0.7ns, 1.8ns, 3ns, 5ns, 10ns, 20ns, 30ns, 50ns, 100ns, 100µs



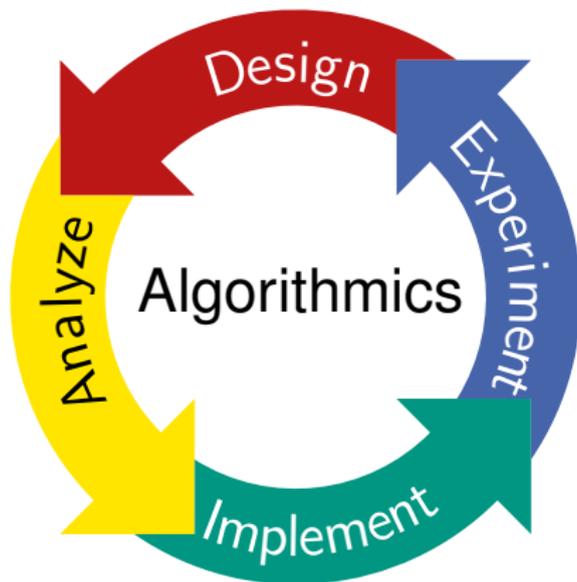
Lücke Theorie vs. Praxis

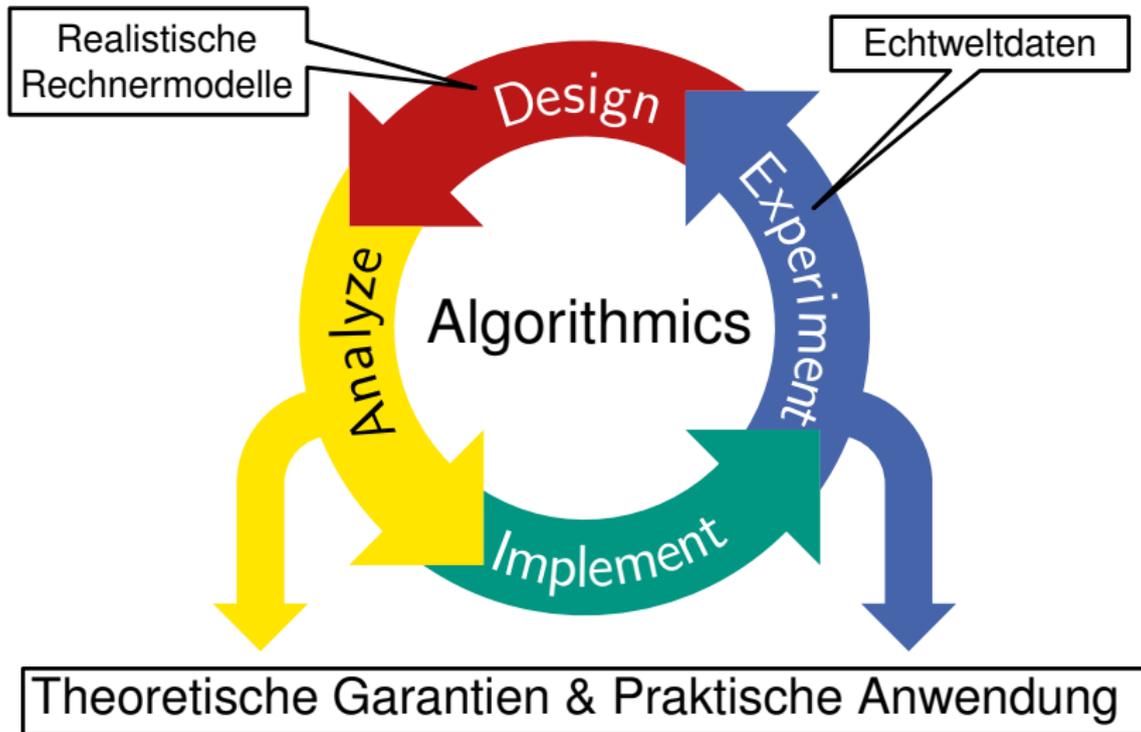
Theorie	vs.	Praxis
einfach einfach	Problem-Modell Maschinenmodell	komplex komplex
komplex fortgeschritten	Algorithmen Datenstrukturen	einfach einfach
worst-case asymptotisch	Komplexitäts-Messung Effizienz	typische Eingaben konstante Faktoren

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

Routenplanung:

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





Beschleunigungstechniken

Beobachtung:

- Viele Anfragen in (statischem) Netzwerk
- „Unnötige“ Berechnungen

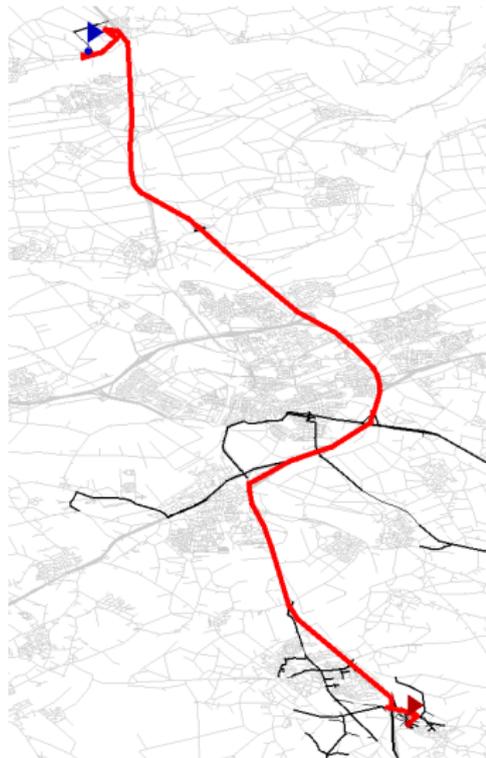


Beobachtung:

- Viele Anfragen in (statischem) Netzwerk
- „Unnötige“ Berechnungen

Idee:

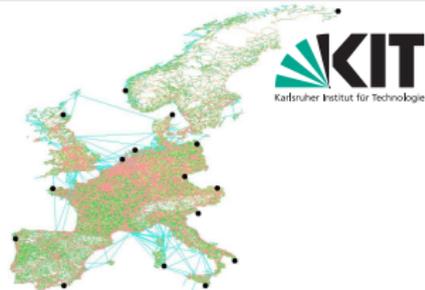
- Zwei Phasen
 - Offline: Generiere Zusatzinformation in **Vorbereitung**
 - Online: **Beschleunige** Anfrage mit dieser Information
- Drei Kriterien: Vorberechnungsplatz, Vorberechnungszeit, Beschleunigung



Experimentelle Evaluation

Eingabe: Straßennetzwerk von Europa

- ca. 18 Mio. Knoten
- ca. 42 Mio. Kanten



Algorithmus	VORBERECHNUNG		ANFRAGEN	
	Zeit [h:m]	Platz [GiB]	Zeit [μ s]	Beschleun.
Dijkstra [Dij59]	—	—	2 550 000	—
ALT [GH05, GW05]	0:42	2.2	24 521	104
CRP [DGPW11]	≪ 0:01	< 0.1	1 650	1 545
Arc-Flags [Lau04]	0:20	0.3	408	6 250
CH [GSSD08]	0:05	0.2	110	23 181
TNR [ALS13]	0:20	2.1	1.25	2 040 000
HL [ADGW12]	0:37	18.4	0.56	4 553 571

Experimentelle Evaluation

Eingabe: Straßennetzwerk von Europa

- ca. 18 Mio. Knoten
- ca. 42 Mio. Kanten



Algorithmus	VORBERECHNUNG		ANFRAGEN	
	Zeit [h:m]	Platz [GiB]	Zeit [μ s]	Beschleun.
Dijkstra [Dij59]	—	—	2 550 000	—
ALT [GH05, GW05]	0:42	2.2	24 521	104
CRP [DGPW11]	\ll 0:01	$<$ 0.1	1 650	1 545
Arc-Flags [Lau04]	0:20	0.3	408	6 250
CH [GSSD08]	0:05	0.2	110	23 181
TNR [ALS13]	0:20	2.1	1.25	2 040 000
HL [ADGW12]	0:37	18.4	0.56	4 553 571

Mittlerweile im Einsatz bei Bing, Google, Apple, Tomtom, ...

- Algorithm Engineering + ein bisschen Theorie
- Beschleunigungstechniken
- Implementierungsdetails
- Ergebnisse auf Real-Welt Daten
- Aktuellster Stand der Forschung (Veröffentlichungen bis 2017)
- Ideale Grundlage für Masterarbeiten

keine Algorithmen III

- Vertiefung von kürzesten Wegen (Dijkstra)
 - Grundlage ist der Stoff von Algo I;
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

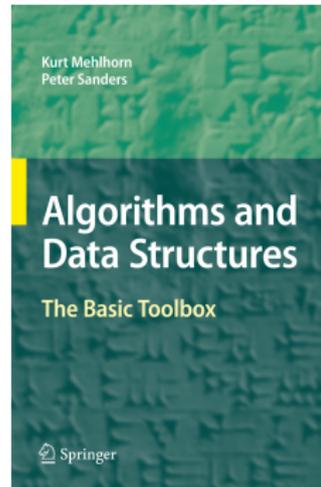
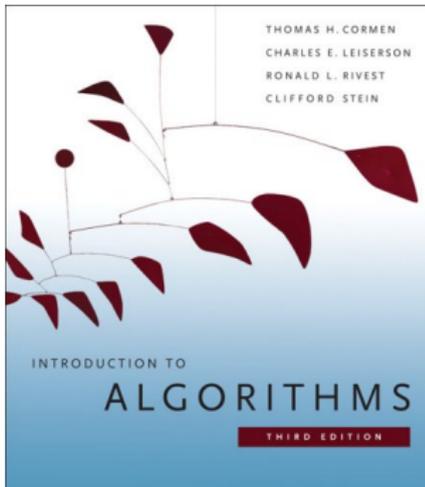
4. Fortgeschrittene Szenarien

- Schnelle many-to-many und all-pairs shortest-paths
- Alternativrouten
- Zeitabhängige Routenplanung
- Fahrplanauskunft
- Routenplanung für Elektroautos
- 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:

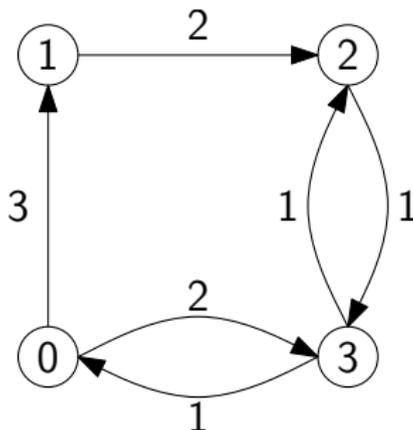


1. Grundlagen



Drei klassische Ansätze:

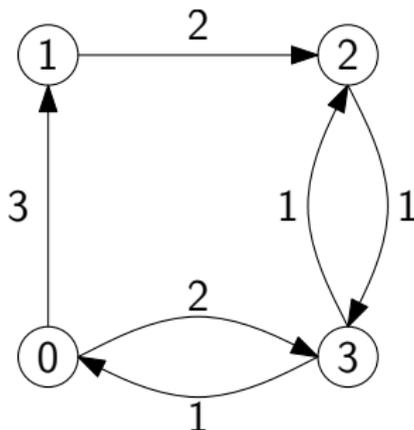
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



Drei klassische Ansätze:

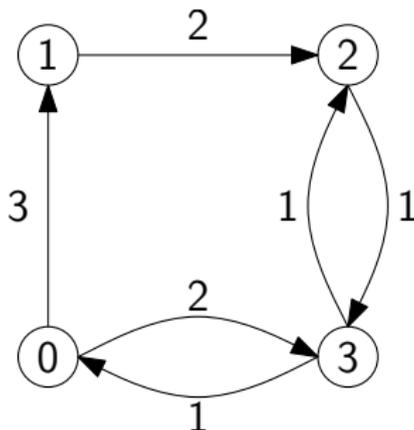
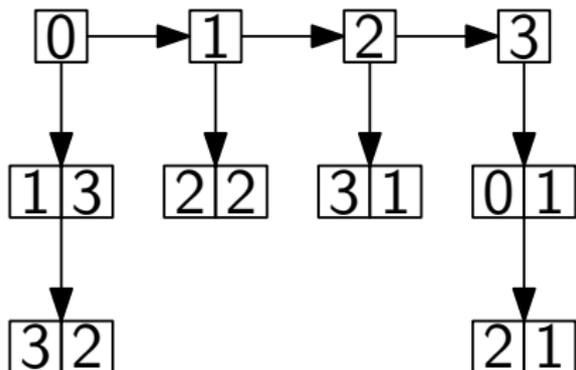
- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

	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

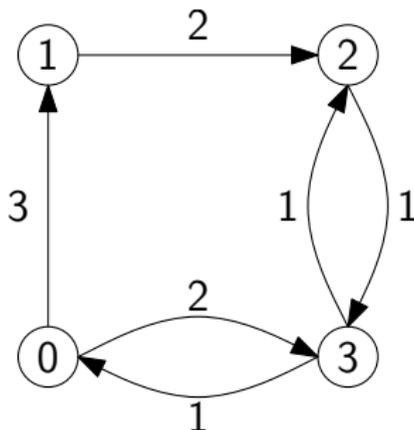
0	2	3	4	6
---	---	---	---	---

targetNode

1	3	2	3	2	0
---	---	---	---	---	---

weight

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



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 (Speicherlayout)	+	-	+
Updates (neue Kante)	+	+	-
Updates (Gewicht)	+	+	+

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 (neue Kante)	+	+	-
Updates (Gewicht)	+	+	+

Fragen:

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

Modellierung (Straßengraphen)

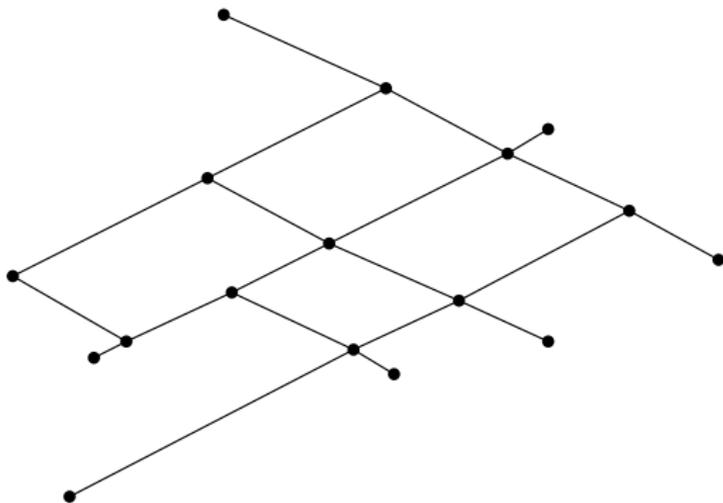


Modellierung (Straßengraphen)



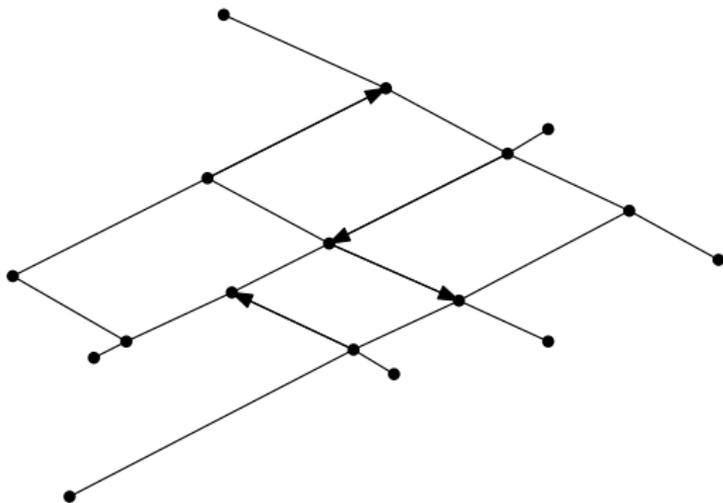
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen



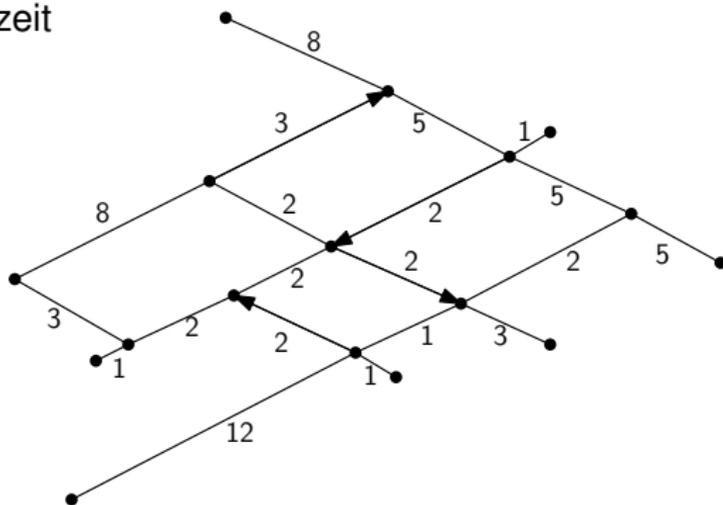
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen

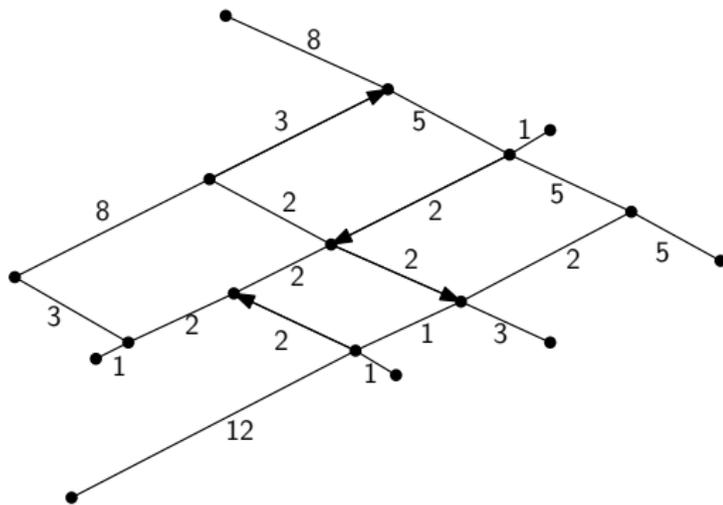


Modellierung (Straßengraphen)

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

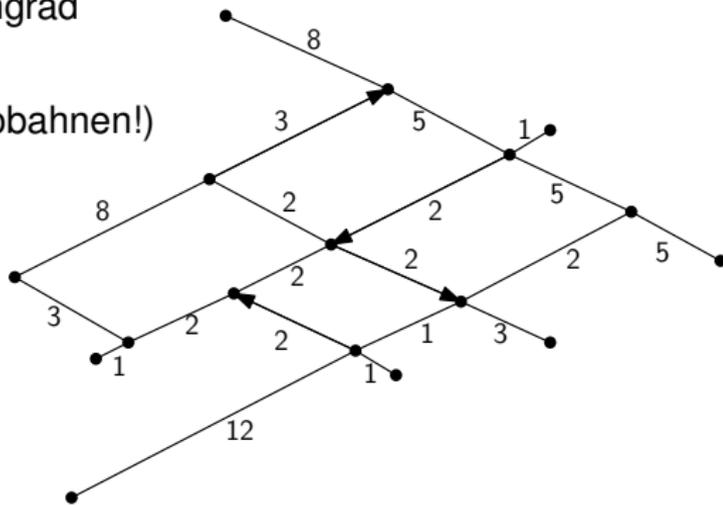


Eigenschaften (sammeln):



Eigenschaften:

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

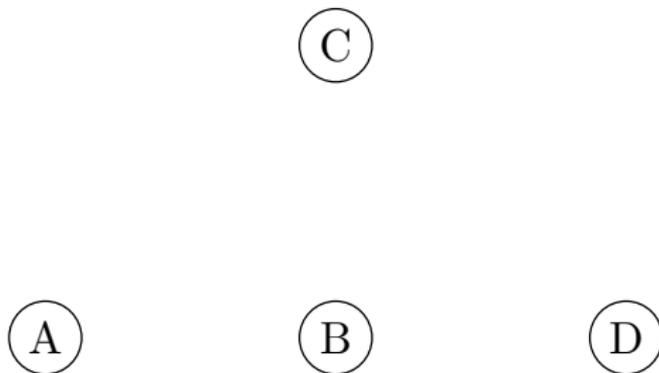


Beispiel:

- 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

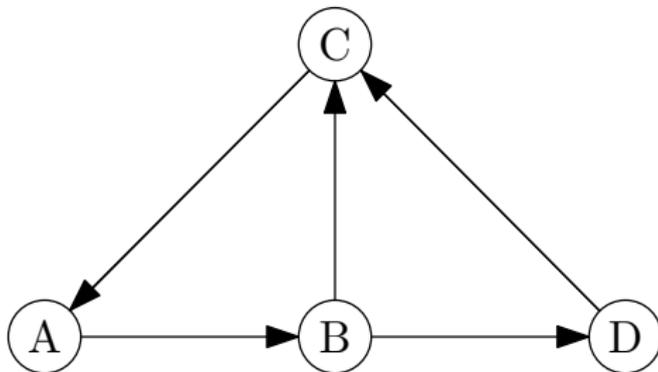
Beispiel:

- 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



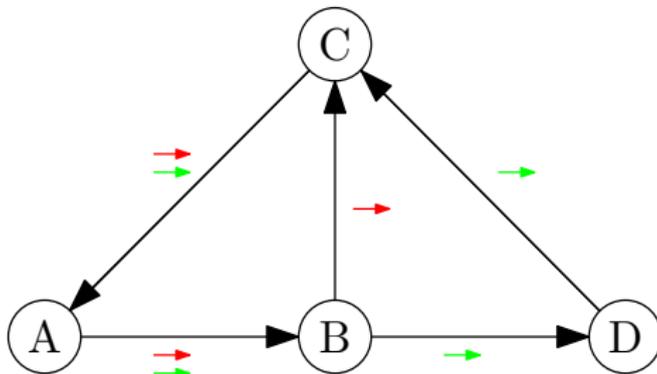
Beispiel:

- 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



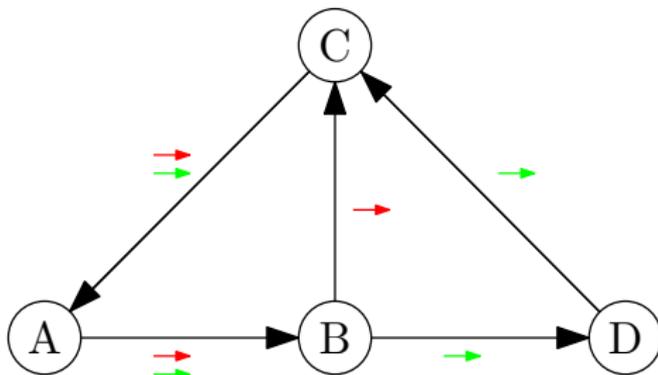
Beispiel:

- 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



Beispiel:

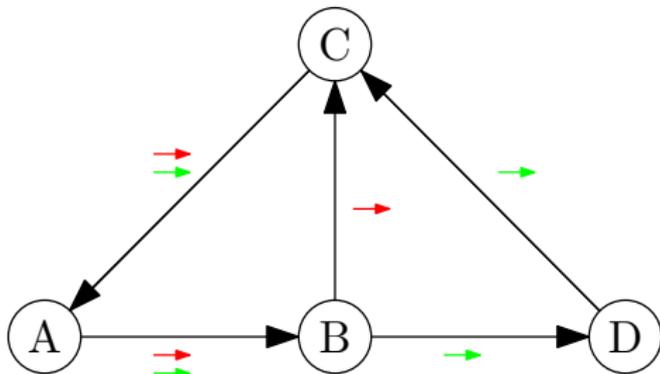
- 4 Stationen (A,B,C,D)
- Zug 1: Station A \rightarrow B \rightarrow C \rightarrow A
- Zug 2: Station A \rightarrow B \rightarrow D \rightarrow C \rightarrow A
- Züge operieren alle 10 Minuten



Kanten sind zeitabhängig!
(wann fährt ein Zug wie lange?)

Beispiel:

- 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



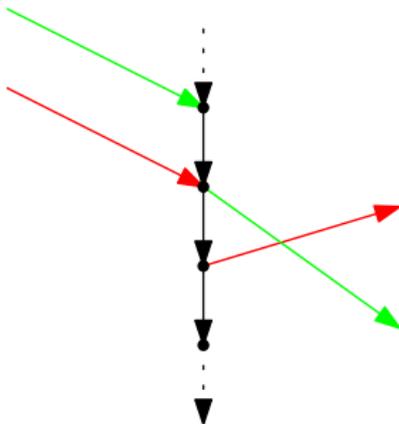
Kanten sind zeitabhängig!
(wann fährt ein Zug wie lange?)

oder roll die Zeit aus

Vorgehen:

- Knoten sind Abfahrts-/Ankunftsereignisse
- Kanten für die Fahrt von Station zu Station
- Wartekanten an den Stationen für Umstiege

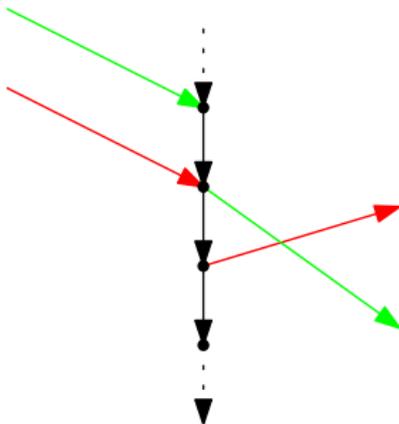
Station B:



Vorgehen:

- Knoten sind Abfahrts-/Ankunftsereignisse
- Kanten für die Fahrt von Station zu Station
- Wartekanten an den Stationen für Umstiege

Station B:



Diskussion:

- + keine zeitabhängigen Kanten
- Graph größer

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

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

Diskussion:

- berechne beste Verbindungen in solchen Netzwerken
- Adjazenzarray als Graphdatenstruktur

2. Kürzeste Wege

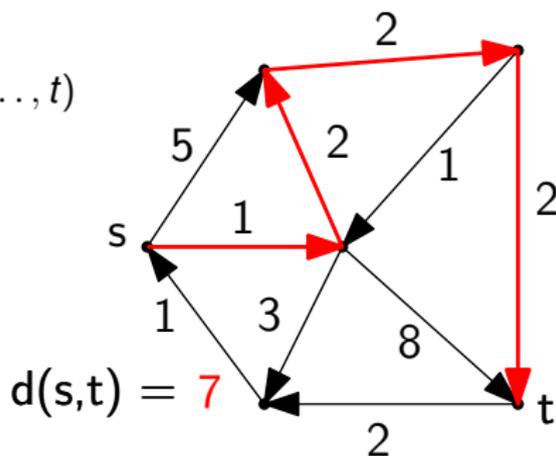


Gegeben:

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 $P := (s, \dots, t)$



Azyklizität:

- Kürzeste Wege sind zyklenfrei

Aufspannungseigenschaft:

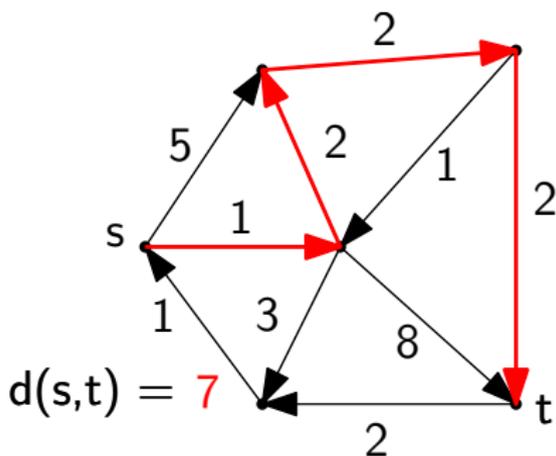
- Alle kürzesten Wege von s aus bilden DAG bzw. Baum

Vererbungseigenschaft:

- Teilwege von kürzesten Wegen sind kürzeste Wege

Abstand:

- Steigender Abstand von Wurzel zu Blättern



Komplexität von **Single-Source Shortest Paths** abhängig vom Eingabegraphen.

- In dieser Vorlesung: Kantengewichte (*fast*) immer nicht-negativ.
- Ein negativer Zyklus ist ein Kreis mit negativem Gesamtgewicht.
- Ein einfacher Pfad ist ein Pfad bei dem sich kein Knoten wiederholt.

Problemvarianten

- Kantengewichte alle positiv:
Dijkstra's Algorithmus anwendbar (Laufzeit $|V| \log |V| + |E|$)
- Kantengewichte auch negativ, aber kein negativer Zyklus:
Algorithmus von Bellmann-Ford anwendbar (Laufzeit $|V| \cdot |E|$)
- Kantengewichte auch negativ, suche kürzesten einfachen Pfad:
 \mathcal{NP} -schwer, Reduktion von „Problem Longest Path“

Bellman-Ford(G, s)

```
1 for  $v \in V$  do  $d[v] \leftarrow \infty$ 
2  $d[s] \leftarrow 0$ 
3 for  $i = 1$  to  $|V| - 1$  do
4   forall edges  $(u, v) \in E$  do
5     if  $d[u] + \text{len}(u, v) < d[v]$  then
6        $d[v] \leftarrow d[u] + \text{len}(u, v)$ 
```

Bellman-Ford(G, s)

```
1 for  $v \in V$  do  $d[v] \leftarrow \infty$ 
2  $d[s] \leftarrow 0$ 
3 for  $i = 1$  to  $|V| - 1$  do
4   forall edges  $(u, v) \in E$  do
5     if  $d[u] + \text{len}(u, v) < d[v]$  then
6        $d[v] \leftarrow d[u] + \text{len}(u, v)$ 
7 forall edges  $(u, v) \in E$  do
8   if  $d[v] > d[u] + \text{len}(u, v)$  then
9     negative cycle found
```

Problem Longest Path

Gegeben:

- Gerichteter, gewichteter Graph $G = (V, E)$ mit gewicht $len: E \rightarrow \mathbb{N}$
- Zahl $K \in \mathbb{N}$
- Knoten $s, t \in V$

Frage:

- Gibt es einen einfachen s - t -Pfad der Länge mindestens K ?

Problem Longest Path ist \mathcal{NP} -schwer (siehe [Garey & Johnson 79])

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

```
1 forall nodes  $v \in V$  do
2    $d[v] = \infty, p[v] = \text{NULL}$            // distances, parents
3  $d[s] = 0$ 
4  $Q.\text{clear}(), Q.\text{insert}(s, 0)$            // container
5 while  $!Q.\text{empty}()$  do
6    $u \leftarrow Q.\text{deleteMin}()$            // settling node u
7   forall 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  $Q.\text{decreaseKey}(v, d[v])$ 
13      else  $Q.\text{insert}(v, d[v])$ 
```

Dijkstras Algorithmus

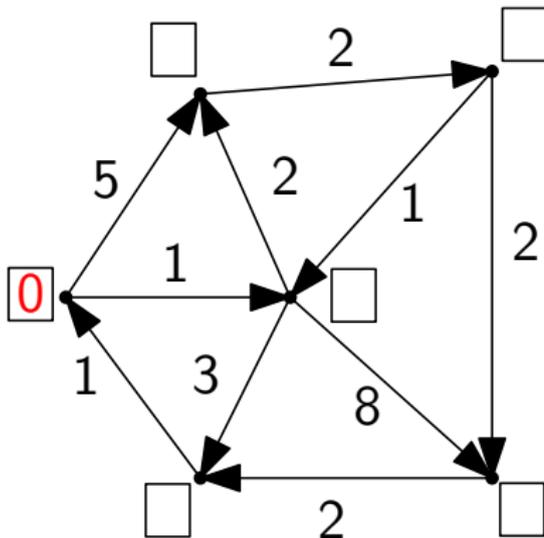
Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

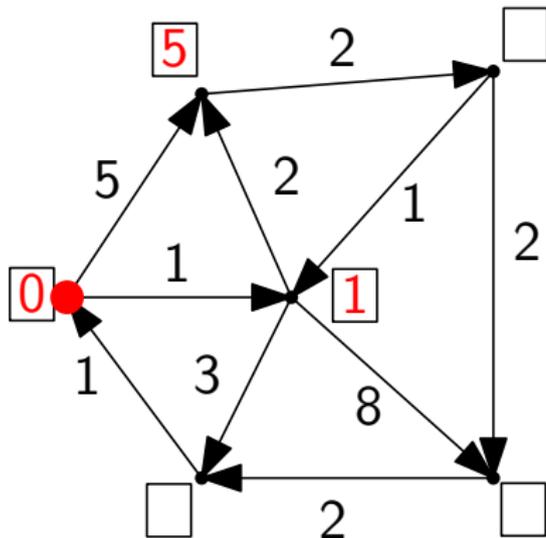


Invariante: Pfade zu roten Knoten sind **optimal**.

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

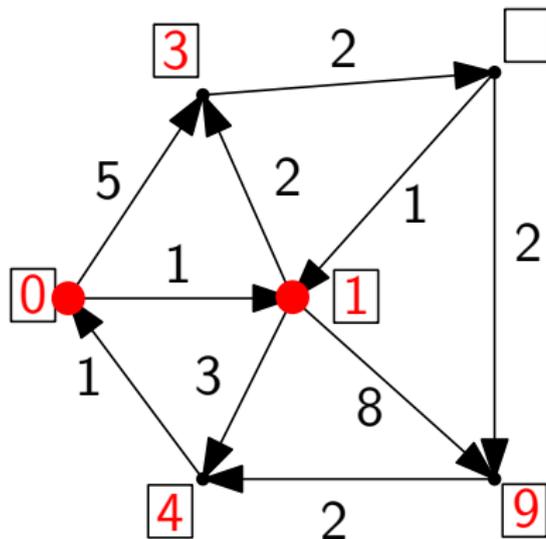


Invariante: Pfade zu roten Knoten sind **optimal**.

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

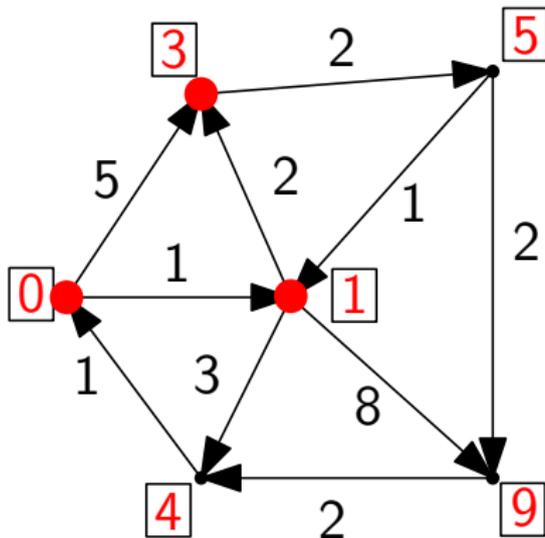


Invariante: Pfade zu roten Knoten sind **optimal**.

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

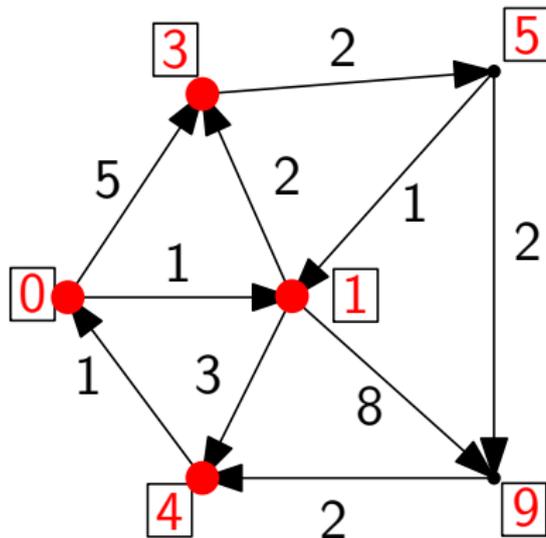


Invariante: Pfade zu roten Knoten sind **optimal**.

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

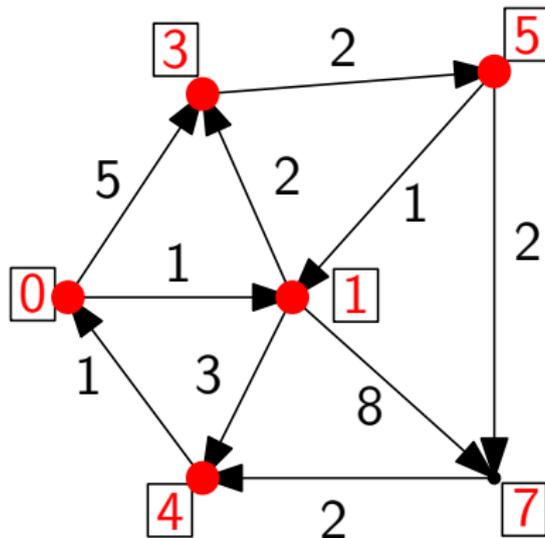


Invariante: Pfade zu roten Knoten sind **optimal**.

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

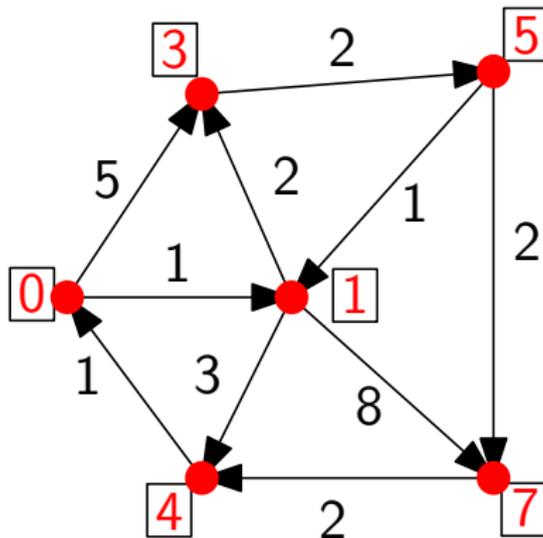


Invariante: Pfade zu roten Knoten sind **optimal**.

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .

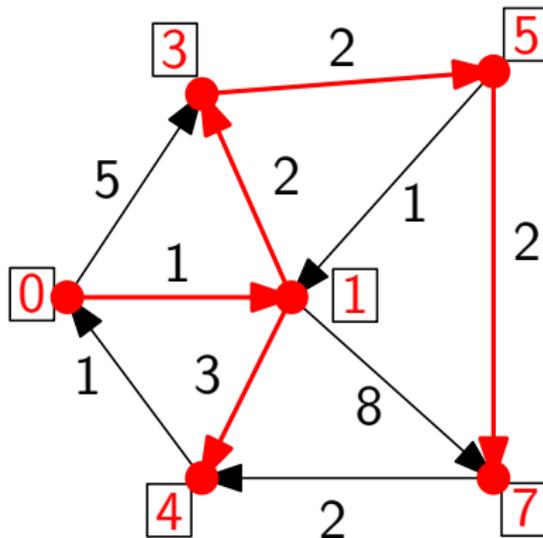


Invariante: Pfade zu roten Knoten sind **optimal**.

Dijkstras Algorithmus

Gegeben: Graph G , Startknoten.

Idee: Suche in G mit zunehmender Distanz von s .



Invariante: Pfade zu roten Knoten sind **optimal**.

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

```
1 forall nodes  $v \in V$  do
2    $d[v] = \infty, p[v] = \text{NULL}$            // distances, parents
3  $d[s] = 0$ 
4  $Q.\text{clear}(), Q.\text{insert}(s, 0)$            // container
5 while  $!Q.\text{empty}()$  do
6    $u \leftarrow Q.\text{deleteMin}()$            // settling node u
7   forall 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  $Q.\text{decreaseKey}(v, d[v])$ 
13      else  $Q.\text{insert}(v, d[v])$ 
```

Kantenrelaxierung: Der Vorgang

1 **if** $d[u] + \text{len}(u, v) < d[v]$ **then** $d[v] \leftarrow d[u] + \text{len}(u, v)$

heißt Kantenrelaxierung.

Kantenrelaxierung: Der Vorgang

1 **if** $d[u] + \text{len}(u, v) < d[v]$ **then** $d[v] \leftarrow d[u] + \text{len}(u, v)$

heißt Kantenrelaxierung.

Besuchte Knoten: Ein Knoten heißt (zu einem Zeitpunkt) besucht (visited) wenn er (zu diesem Zeitpunkt) schon in die Queue eingefügt wurde (unabhängig davon, ob er noch in der Queue ist).

Kantenrelaxierung: Der Vorgang

1 **if** $d[u] + \text{len}(u, v) < d[v]$ **then** $d[v] \leftarrow d[u] + \text{len}(u, v)$

heißt Kantenrelaxierung.

Besuchte Knoten: Ein Knoten heißt (zu einem Zeitpunkt) besucht (visited) wenn er (zu diesem Zeitpunkt) schon in die Queue eingefügt wurde (unabhängig davon, ob er noch in der Queue ist).

Abgearbeitete Knoten: Ein Knoten heißt (zu einem Zeitpunkt) abgearbeitet (settled) wenn er (zu diesem Zeitpunkt) schon in die Queue eingefügt und wieder extrahiert wurde.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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	$\Theta(1)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$	$\Theta(1)$
Minimum	$\Theta(n)$	$\Theta(1)$	$\mathcal{O}(\log k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\log k)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$
Union	$\Theta(1)$	$\Theta(k)$	$\mathcal{O}(\log k)$	$\Theta(1)$
DecreaseKey	$\Theta(1)$	$\Theta(\log k)$	$\Theta(\log k)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(\log k)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$
Dijkstra	$\mathcal{O}(n^2 + m)$	$\mathcal{O}((n + m) \log n)$	$\mathcal{O}((n + m) \log n)$	$\mathcal{O}(m + n \log n)$

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

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	$\Theta(1)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$	$\Theta(1)$
Minimum	$\Theta(n)$	$\Theta(1)$	$\mathcal{O}(\log k)$	$\Theta(1)$
DeleteMin	$\Theta(n)$	$\Theta(\log k)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$
Union	$\Theta(1)$	$\Theta(k)$	$\mathcal{O}(\log k)$	$\Theta(1)$
DecreaseKey	$\Theta(1)$	$\Theta(\log k)$	$\Theta(\log k)$	$\Theta(1)$
Delete	$\Theta(1)$	$\Theta(\log k)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$
Dijkstra	$\mathcal{O}(n^2 + m)$	$\mathcal{O}((n + m) \log n)$	$\mathcal{O}((n + m) \log n)$	$\mathcal{O}(m + n \log n)$
$m \in \mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$	$\mathcal{O}(n \log n)$

Transportnetzwerke sind dünn \Rightarrow Binary Heaps

k-ary Heap: Baum mit (max.) k Kindern je Vorgängerknoten

k	Query [sec]
2	1.834
3	1.595
4	1.507
5	1.525
8	1.561

Graph: 18M Knoten, 42M Kanten

k-ary Heap: Baum mit (max.) k Kindern je Vorgängerknoten

k	Query [sec]
2	1.834
3	1.595
4	1.507
5	1.525
8	1.561

Graph: 18M Knoten, 42M Kanten

Dijkstra: ≈ 1.5 s \Rightarrow nicht interaktiv

k-ary Heap: Baum mit (max.) k Kindern je Vorgängerknoten

k	Query [sec]
2	1.834
3	1.595
4	1.507
5	1.525
8	1.561

Graph: 18M Knoten, 42M Kanten

Dijkstra: ≈ 1.5 s \Rightarrow nicht interaktiv
 $n + m$ CPU clock cycles: ≈ 20 ms \Rightarrow viel schneller

k-ary Heap: Baum mit (max.) k Kindern je Vorgängerknoten

k	Query [sec]
2	1.834
3	1.595
4	1.507
5	1.525
8	1.561

Graph: 18M Knoten, 42M Kanten

Dijkstra:	≈ 1.5 s	\Rightarrow nicht interaktiv
$n + m$ CPU clock cycles:	≈ 20 ms	\Rightarrow viel schneller
BFS:	≈ 1.2 s	\Rightarrow an der Queue liegt's nicht

k-ary Heap: Baum mit (max.) k Kindern je Vorgängerknoten

k	Query [sec]
2	1.834
3	1.595
4	1.507
5	1.525
8	1.561

Graph: 18M Knoten, 42M Kanten

Dijkstra: ≈ 1.5 s \Rightarrow nicht interaktiv
 $n + m$ CPU clock cycles: ≈ 20 ms \Rightarrow viel schneller
BFS: ≈ 1.2 s \Rightarrow an der Queue liegt's nicht

Performanz von Graphsuchen ist speicher-begrenzt

Montag, 29. April 2018



Ittai Abraham, Daniel Delling, Andrew V. Goldberg, and Renato F. Werneck.

Hierarchical Hub Labelings for Shortest Paths.

In Leah Epstein and Paolo Ferragina, editors, Proceedings of the 20th Annual European Symposium on Algorithms (ESA'12), volume 7501 of Lecture Notes in Computer Science, pages 24–35. Springer, 2012.



Julian Arz, Dennis Luxen, and Peter Sanders.

Transit Node Routing Reconsidered.

In Proceedings of the 12th International Symposium on Experimental Algorithms (SEA'13), volume 7933 of Lecture Notes in Computer Science, pages 55–66. Springer, 2013.



Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck.

Customizable Route Planning.

In Panos M. Pardalos and Steffen Rebennack, editors, Proceedings of the 10th International Symposium on Experimental Algorithms (SEA'11), volume 6630 of Lecture Notes in Computer Science, pages 376–387. Springer, 2011.



Edsger W. Dijkstra.

A Note on Two Problems in Connexion with Graphs.

Numerische Mathematik, 1(1):269–271, 1959.



Andrew V. Goldberg and Chris Harrelson.

Computing the Shortest Path: A* Search Meets Graph Theory.

In Proceedings of the 16th Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'05), pages 156–165. SIAM, 2005.



Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling.

Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.

In Catherine C. McGeoch, editor, Proceedings of the 7th Workshop on Experimental Algorithms (WEA'08), volume 5038 of Lecture Notes in Computer Science, pages 319–333. Springer, June 2008.



Andrew V. Goldberg and Renato F. Werneck.

Computing Point-to-Point Shortest Paths from External Memory.

In Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX'05), pages 26–40. SIAM, 2005.



Ulrich Lauther.

An Extremely Fast, Exact Algorithm for Finding Shortest Paths in Static Networks with Geographical Background.

In Geoinformation und Mobilität - von der Forschung zur praktischen Anwendung, volume 22, pages 219–230. IfGI prints, 2004.