Karlsruher Institut für Technologie

Algorithmen für Routenplanung

1. Vorlesung, Sommersemester 2021

Tim Zeitz | 12. April 2021

Organisatorisches

Vorlesung

- Montags 14:00–15:30 Uhr
- Mittwochs 12:00–13:30 Uhr

Prüfung

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

Vorlesungsaufzeichnungen

- Nach der Vorlesung im ILIAS
- Nur Folien und Dozent werden aufgezeichnet

Dozenten

Torsten Ueckerdt

Valentin Buchhold

Adrian Feilhauer

Jonas Sauer

Tim Zeitz

Vorlesungswebseite

i11www.iti.kit.edu/teaching/sommer2021/routenplanung/index

Organisatorisches

Karlsruher Institut für Technologie

Terminübersicht:

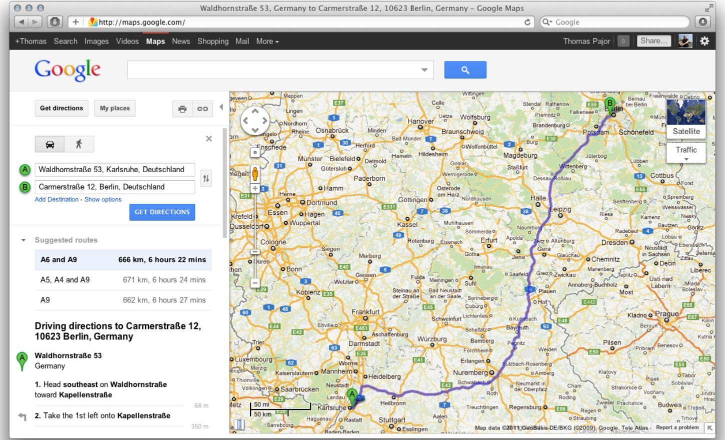
Montag	12.4.	Tim	Einführung, Grundlagen, Dijkstras Algorithmus
Mittwoch	14.4.	Tim	Kontraktion & TopoCore
Montag	19.4.	Tim	A*, ALT, CALT
Mittwoch	21.4.	Tim	Arc-Flags, SHARC
Montag	26.4.	Valentin	Reach, MLO/CRP

...

Routenplanung in der Anwendung

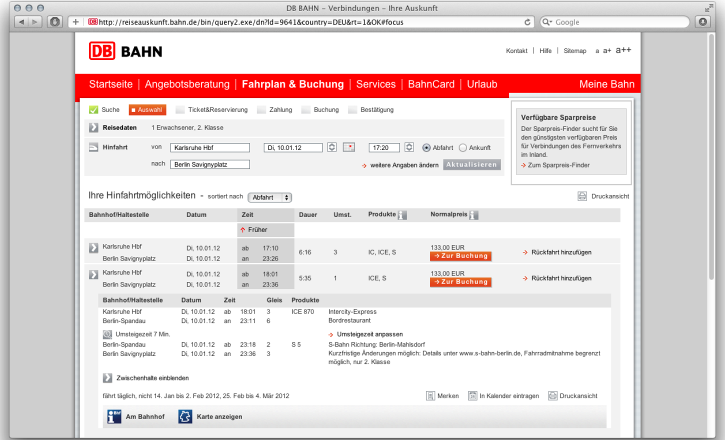
Routenplanung in der Anwendung

Karlsruher Institut für Technologie



Routenplanung in der Anwendung

Karlsruher Institut für Technologie



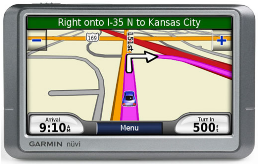
Routenplanung in der Anwendung

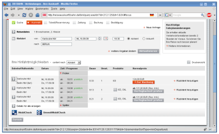
Motivation Routenplanung

Karlsruher Institut für Technologie

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

Karlsruher Institut für Technologie

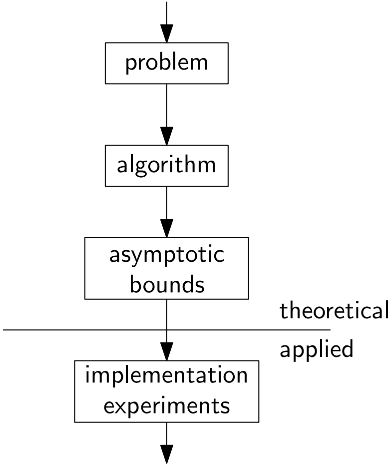
Anfrage:

- Finde beste Verbindung in Transportnetz

Modellierung:

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

Klassischer Algorithmenentwurf



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)

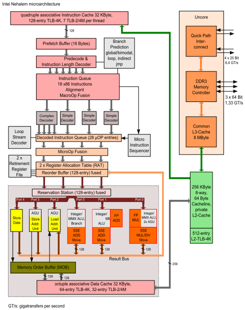
Moderne CPU Architektur

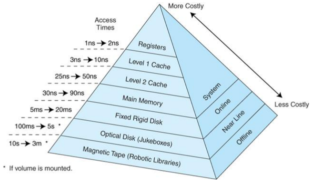
Einige Fakten:

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

Hauptherausforderungen:

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





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

Lücke Theorie vs. Praxis

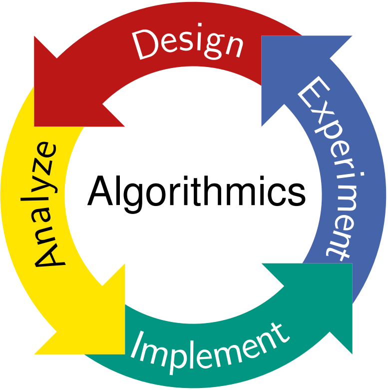
Karlsruher Institut für Technologie

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

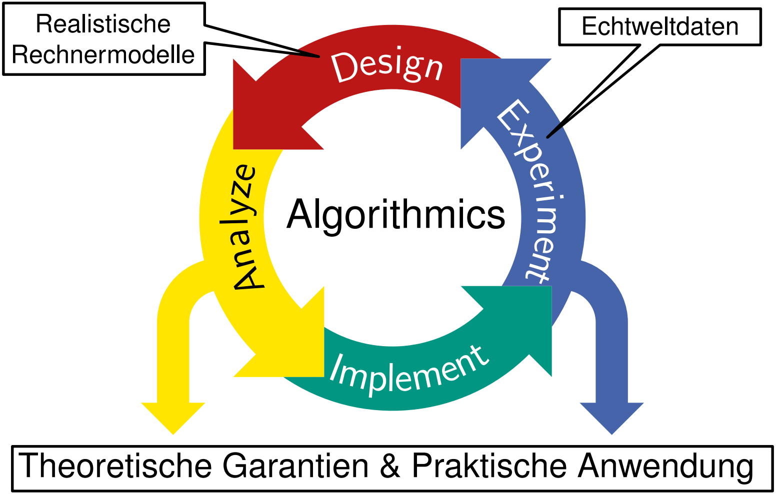
Routenplanung:

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

Algorithm Engineering



Algorithm Engineering



Beschleunigungstechniken

Beobachtung:

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

Beschleunigungstechniken

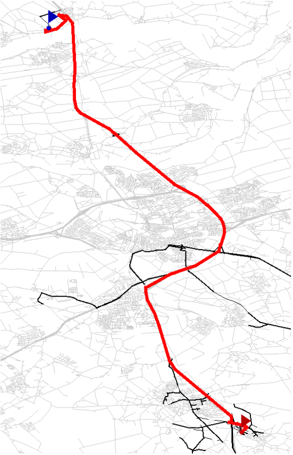
Karlsruher Institut für Technologie

Beobachtung:

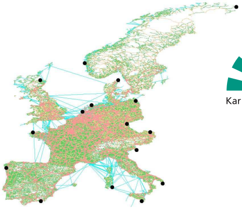
- Viele Anfragen in (statischem) Netzwerk
- „Unnötige“ Berechnungen

Idee:

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



Experimentelle Evaluation



Karlsruher Institut für Technologie

Eingabe: Straßennetzwerk von Europa

- 18 Mio. Knoten, 42 Mio. Kanten

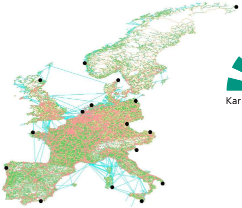
	Vorberechnung		Anfragen	
Algorithmus	Zeit [h:m]	Platz [GiB]	Zeit [μs]	Beschleun.
Dijkstra [Dij59]	—	—	2 550 000	—
ALT [GH05, GW05]	0:42	2.2	24 521	104
CRP [DGPW11]	$\ll$ 0:01	$<$ 0.1	1 650	1 545
Arc-Flags [Lau04]	0:20	0.3	408	6 250
CH [GSSD08]	0:05	0.2	110	23 181
TNR [ALS13]	0:20	2.1	1.25	2 040 000
HL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

Experimentelle Evaluation

Karlsruher Institut für Technologie

Eingabe: Straßennetzwerk von Europa

- 18 Mio. Knoten, 42 Mio. Kanten



	Vorberechnung		Anfragen	
Algorithmus	Zeit [h:m]	Platz [GiB]	Zeit [μs]	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, ...

Schwerpunkte der Vorlesung

Karlsruher Institut für Technologie

- Algorithm Engineering + ein bisschen Theorie
- Beschleunigungstechniken
- Implementierungsdetails
- Ergebnisse auf Real-Welt Daten
- Aktuellster Stand der Forschung (Veröffentlichungen bis 2019)
- Ideale Grundlage für Masterarbeiten

Was diese Vorlesung nicht ist

Karlsruher Institut für Technologie

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

Inhalt der Vorlesung

Karlsruher Institut für Technologie

1. **Grundlagen**
   - Algorithm Engineering
   - Graphen, Modelle, usw.
   - Kürzeste Wege
   - Dijkstra's Algorithmus
2. **Beschleunigung von (statischen) Punkt-zu-Punkt Anfragen**
   - Zielgerichtete Verfahren
   - Hierarchische Techniken
   - Many-to-many-Anfragen und Distanztabellen
   - Kombinationen

Inhalt der Vorlesung

Karlsruher Institut für Technologie

3. Theorie

- 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

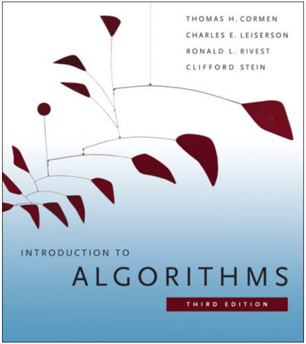
Nützliche Vorkenntnisse

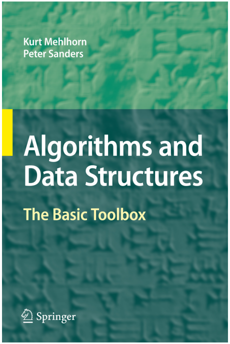
- Informatik I/II oder Algorithmen I
- Algorithmentechnik oder Algorithmen II (muss aber nicht sein)
- ein bisschen Rechnerarchitektur
- passive Kenntnisse von C++/Java

Vertiefungsgebiet: Algorithmentechnik, ~~Theoretische Grundlagen~~

Material

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



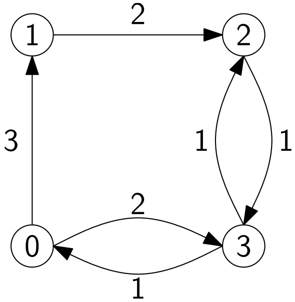


1. Grundlagen

Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

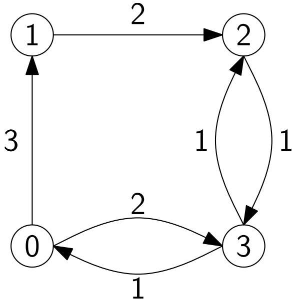


Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

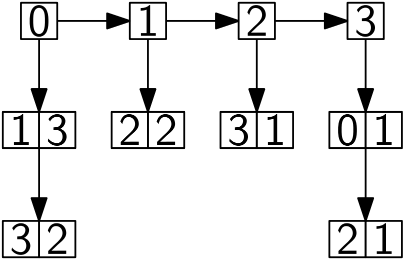
	0	1	2	3
0	–	3	–	2
1	–	–	2	–
2	–	–	–	1
3	1	–	1	–

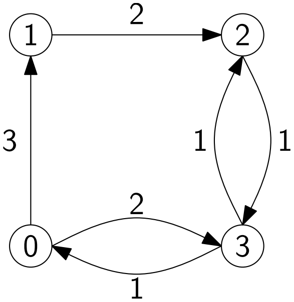


Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray

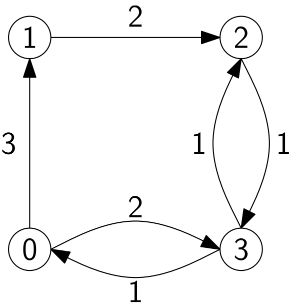




Graph-Repräsentationen

Drei klassische Ansätze:

- Adjazenzmatrix
- Adjazenzlisten
- Adjazenzarray



`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)	$+$	$+$	$+$

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)	$+$	$+$	$+$

Fragen:

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

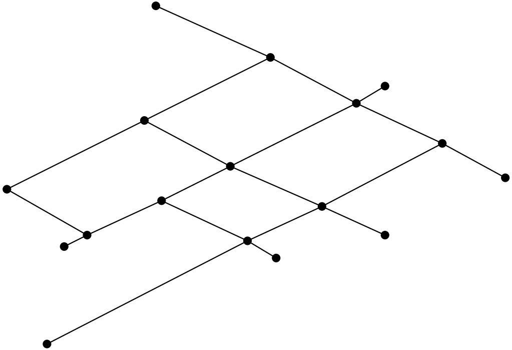
Modellierung (Straßengraphen)

Karlsruher Institut für Technologie

Modellierung (Straßengraphen)

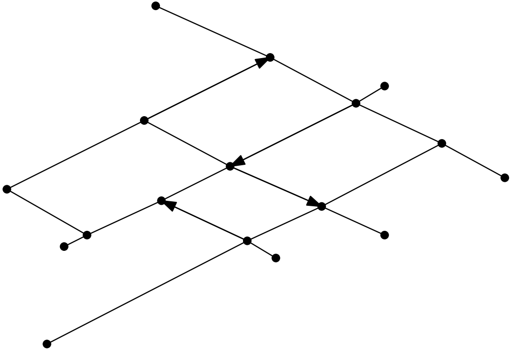
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen



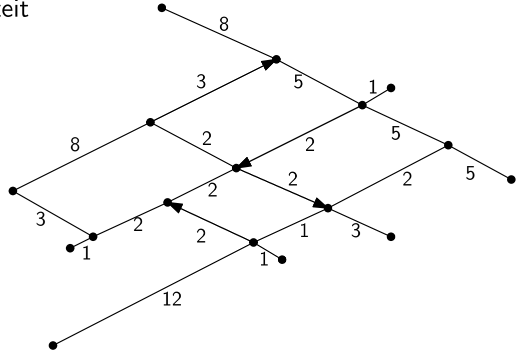
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen



Modellierung (Straßengraphen)

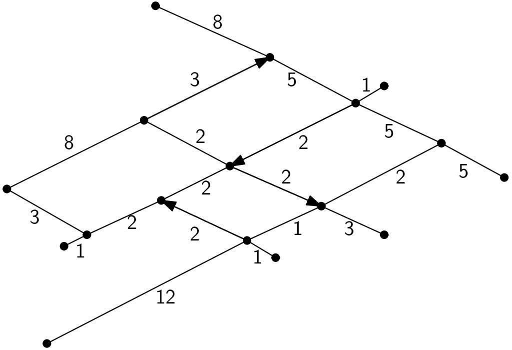
- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



Modellierung (Straßengraphen)

Karlsruher Institut für Technologie

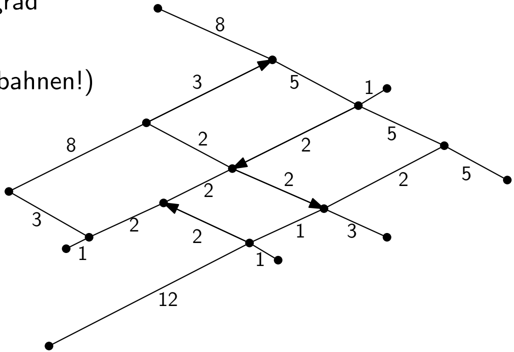
Eigenschaften (sammeln):



Modellierung (Straßengraphen)

Eigenschaften:

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



Modellierung (Eisenbahngraphen)

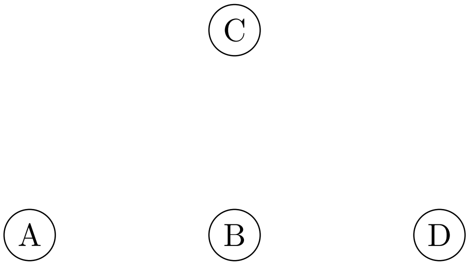
Beispiel:

- 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

Modellierung (Eisenbahngraphen)

Beispiel:

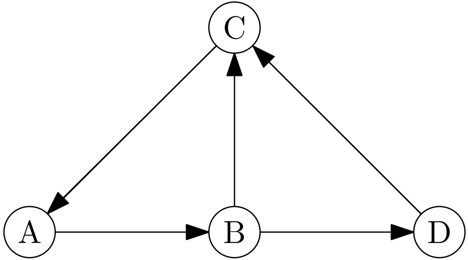
- 4 Stationen (A,B,C,D)
- Zug 1: Station A → B → C → A
- Zug 2: Station A → B → D → C → A
- Züge operieren alle 10 Minuten



Modellierung (Eisenbahngraphen)

Beispiel:

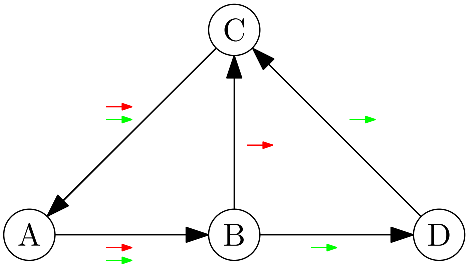
- 4 Stationen (A,B,C,D)
- Zug 1: Station A $\rightarrow$ B $\rightarrow$ C $\rightarrow$ A
- Zug 2: Station A $\rightarrow$ B $\rightarrow$ D $\rightarrow$ C $\rightarrow$ A
- Züge operieren alle 10 Minuten



Modellierung (Eisenbahngraphen)

Beispiel:

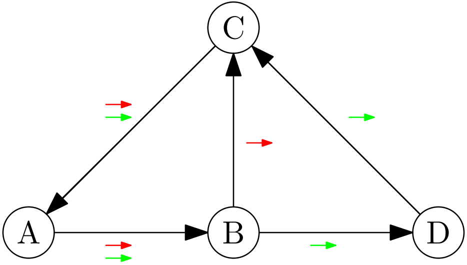
- 4 Stationen (A,B,C,D)
- Zug 1: Station A → B → C → A
- Zug 2: Station A → B → D → C → A
- Züge operieren alle 10 Minuten



Modellierung (Eisenbahngraphen)

Beispiel:

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



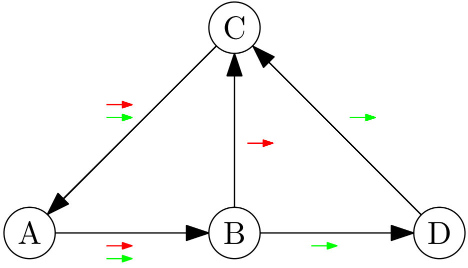
Kanten sind zeitabhängig!

(wann fährt ein Zug wie lange?)

Modellierung (Eisenbahngraphen)

Beispiel:

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



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

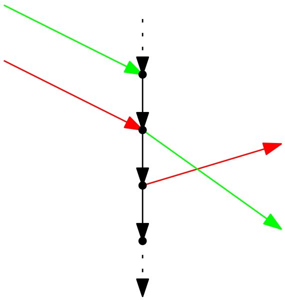
oder roll die Zeit aus

Zeitexpandiertes Modell

Vorgehen:

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

Station B:

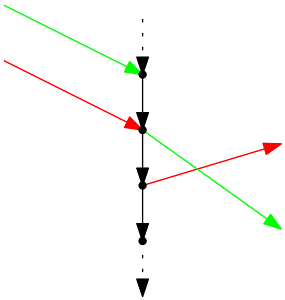


Zeitexpandiertes Modell

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

Eigenschaften der Graphen

Karlsruher Institut für Technologie

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

Eigenschaften der Graphen

Karlsruher Institut für Technologie

- 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

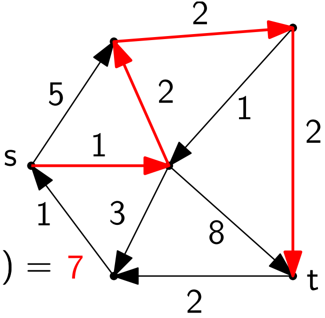
2. Kürzeste Wege

Problemstellung

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

Mögliche Aufgaben

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



d(s,t) = 7

Eigenschaften von kürzesten Wegen

Karlsruher Institut für Technologie

Azyklität:

- Kürzeste Wege sind zyklenfrei

Aufspannungseigenschaft:

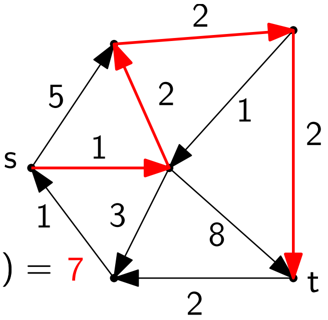
- Alle kürzesten Wege von s aus bilden DAG bzw. Baum

Vererbungseigenschaft:

- Teilwege von kürzesten Wegen sind kürzeste Wege

Abstand:

- Steigender Abstand von Wurzel zu Blättern



$d(s,t) = 7$

Berechnung kürzester Wege

Karlsruher Institut für Technologie

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 beliebig, suche kürzesten einfachen Pfad:
  $\mathcal{NP}$-schwer, Reduktion von „Problem Longest Path“

Der Bellman-Ford Algorithmus

Karlsruher Institut für Technologie

Bellman-Ford(G, s)

```
for v ∈ V do d[v] ← ∞
d[s] ← 0

for i = 1 to |V| − 1 do
    forall edges (u, v) ∈ E do
        if d[u] + len(u, v) < d[v] then
            d[v] ← d[u] + len(u, v)
```

Der Bellman-Ford Algorithmus

```
Bellman-Ford(G, s)

for v ∈ V do d[v] ← ∞
d[s] ← 0

for i = 1 to |V| − 1 do
    forall edges (u, v) ∈ E do
        if d[u] + len(u, v) < d[v] then
            d[v] ← d[u] + len(u, v)

forall edges (u, v) ∈ E do
    if d[v] > d[u] + len(u, v) then
        negative cycle found
```

Problem Longest Path

Karlsruher Institut für Technologie

Problem Longest Path

Gegeben:

- Gerichteter, gewichteter Graph $G = (V, E)$ mit gewicht $\text{len}\colon E \to \mathbb{N}$
- 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])

Dijkstras Algorithmus

Karlsruher Institut für Technologie

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL                          // distances, parents
d[s] = 0
Q.clear(), Q.insert(s, 0)                           // container
while !Q.empty() do
    u ← Q.deleteMin()                               // settling node u
    forall edges e = (u, v) ∈ E do
        // relaxing edges
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])
            else Q.insert(v, d[v])
```

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

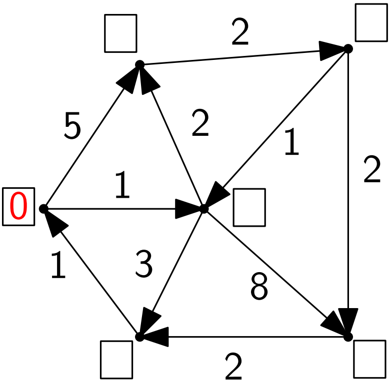
Idee: Suche in G mit zunehmender Distanz von s.

Dijkstras Algorithmus

Karlsruher Institut für Technologie

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.

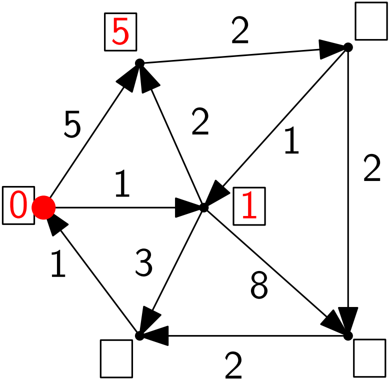


Invariante: Pfade zu roten Knoten sind optimal.

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.

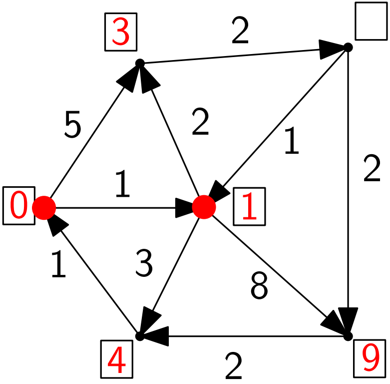


Invariante: Pfade zu roten Knoten sind optimal.

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.

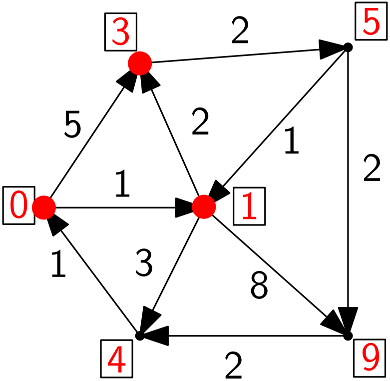


Invariante: Pfade zu roten Knoten sind optimal.

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.

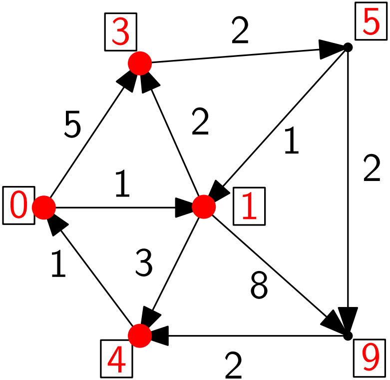


Invariante: Pfade zu roten Knoten sind optimal.

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.

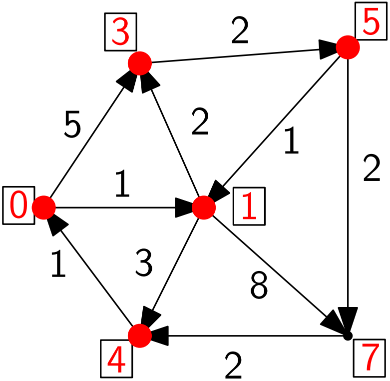


Invariante: Pfade zu roten Knoten sind optimal.

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.

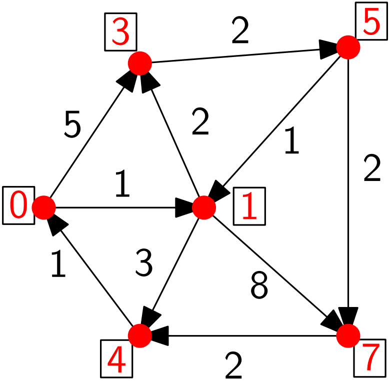


Invariante: Pfade zu roten Knoten sind optimal.

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.

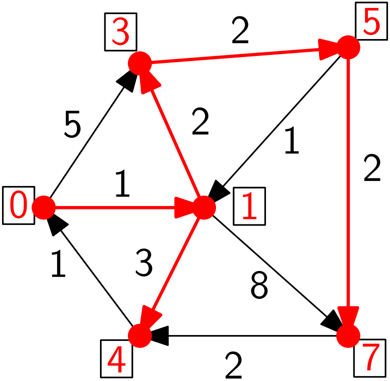


Invariante: Pfade zu roten Knoten sind optimal.

Dijkstras Algorithmus

Gegeben: Graph G, Startknoten.

Idee: Suche in G mit zunehmender Distanz von s.



Invariante: Pfade zu roten Knoten sind optimal.

Kantenrelaxierung

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL                          // distances, parents
d[s] = 0
Q.clear(), Q.insert(s, 0)                           // container
while !Q.empty() do
    u ← Q.deleteMin()                               // settling node u
    forall edges e = (u, v) ∈ E do
        // relaxing edges
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])
            else Q.insert(v, d[v])
```

Definitionen

Kantenrelaxierung: Der Vorgang

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

heißt Kantenrelaxierung.

Definitionen

Kantenrelaxierung: Der Vorgang

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

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).

Definitionen

Kantenrelaxierung: Der Vorgang

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.

Komplexität von Dijkstra

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL
d[s] = 0, Q.clear(), Q.insert(s, 0)
while !Q.empty() do
    u ← Q.deleteMin()
    forall edges e = (u, v) ∈ E do
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])
            else Q.insert(v, d[v])
```

Komplexität von Dijkstra

Karlsruher Institut für Technologie

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL                                  // n Mal
d[s] = 0, Q.clear(), Q.insert(s, 0)
while !Q.empty() do
    u ← Q.deleteMin()
    forall edges e = (u, v) ∈ E do
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])
            else Q.insert(v, d[v])
```

Komplexität von Dijkstra

Karlsruher Institut für Technologie

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL                         // n Mal
d[s] = 0, Q.clear(), Q.insert(s, 0)               // 1 Mal
while !Q.empty() do
    u ← Q.deleteMin()
    forall edges e = (u, v) ∈ E do
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])
            else Q.insert(v, d[v])
```

Komplexität von Dijkstra

Karlsruher Institut für Technologie

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL                          // n Mal
d[s] = 0, Q.clear(), Q.insert(s, 0)               // 1 Mal
while !Q.empty() do
    u ← Q.deleteMin()                              // n Mal
    forall edges e = (u, v) ∈ E do
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])
            else Q.insert(v, d[v])
```

Komplexität von Dijkstra

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL                          // n Mal
d[s] = 0, Q.clear(), Q.insert(s, 0)                // 1 Mal
while !Q.empty() do
    u ← Q.deleteMin()                              // n Mal
    forall edges e = (u, v) ∈ E do
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])   // m Mal
            else Q.insert(v, d[v])
```

Komplexität von Dijkstra

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

forall nodes $v \in V$ **do**
    $d[v] = \infty$, $p[v] = \texttt{NULL}$ // n Mal
$d[s] = 0$, $Q.clear()$, $Q.insert(s, 0)$ // 1 Mal
while $!Q.empty()$ **do**
    $u \leftarrow Q.deleteMin()$ // n Mal
    forall edges $e = (u, v) \in E$ **do**
        if $d[u] + \text{len}(e) < d[v]$ **then**
            $d[v] \leftarrow d[u] + \text{len}(e)$
            $p[v] \leftarrow u$
            if $v \in Q$ **then** $Q.decreaseKey(v, d[v])$ // m Mal
            else $Q.insert(v, d[v])$ // n Mal

Komplexität von Dijkstra

Karlsruher Institut für Technologie

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

```
forall nodes v ∈ V do
    d[v] = ∞, p[v] = NULL                         // n Mal
d[s] = 0, Q.clear(), Q.insert(s, 0)               // 1 Mal
while !Q.empty() do
    u ← Q.deleteMin()                             // n Mal
    forall edges e = (u, v) ∈ E do
        if d[u] + len(e) < d[v] then
            d[v] ← d[u] + len(e)
            p[v] ← u
            if v ∈ Q then Q.decreaseKey(v, d[v])  // m Mal
            else Q.insert(v, d[v])                // n Mal
```

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

Laufzeit Dijkstra

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

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	$\Theta(1)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$	$\Theta(1)$
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	$\mathcal{O}(n^2 + m)$	$\mathcal{O}((n+m)\log n)$	$\mathcal{O}((n+m)\log n)$	$\mathcal{O}(m + n\log n)$

Laufzeit Dijkstra

Karlsruher Institut für Technologie

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

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	$\Theta(1)$	$\Theta(\log k)$	$\mathcal{O}(\log k)$	$\Theta(1)$
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

Laufzeit Dijkstra Experimente

Karlsruher Institut für Technologie

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

Laufzeit Dijkstra Experimente

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: $\approx 1.5\,\text{s}$ $\Rightarrow$ nicht interaktiv

Laufzeit Dijkstra Experimente

Karlsruher Institut für Technologie

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:	$\approx 1.5\,\mathrm{s}$	$\Rightarrow$ nicht interaktiv
$n + m$ CPU clock cycles:	$\approx 20\,\mathrm{ms}$	$\Rightarrow$ viel schneller

Laufzeit Dijkstra Experimente

Karlsruher Institut für Technologie

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: $\approx 1.5\,\text{s}$ $\Rightarrow$ nicht interaktiv
$n + m$ CPU clock cycles: $\approx 20\,\text{ms}$ $\Rightarrow$ viel schneller
BFS: $\approx 1.2\,\text{s}$ $\Rightarrow$ an der Queue liegt's nicht

Laufzeit Dijkstra Experimente

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:	$\approx 1.5\,\text{s}$	$\Rightarrow$ nicht interaktiv
$n + m$ CPU clock cycles:	$\approx 20\,\text{ms}$	$\Rightarrow$ viel schneller
BFS:	$\approx 1.2\,\text{s}$	$\Rightarrow$ an der Queue liegt's nicht

Performanz von Graphsuchen ist speicher-begrenzt

Nächster Termin

Karlsruher Institut für Technologie

Mittwoch, 14. April 2021

Bibliographie I

Karlsruher Institut für Technologie

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.

Bibliographie II

Karlsruher Institut für Technologie

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.