

Algorithmen für Routenplanung

10. Sitzung, Sommersemester 2014

Ben Strasser | 21. Mai 2014

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



Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

- HLDB
- (G)PHAST

Mittwoch, 21.5.2014 fällt aus

Montag, 26.5.2014 (Auf Englisch)

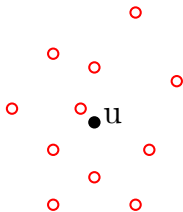
Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$

• u

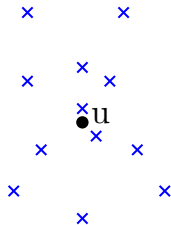
Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$



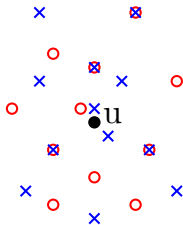
Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$



Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$



Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen $\bullet s$
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad

$\bullet t$

Vorbereitung:

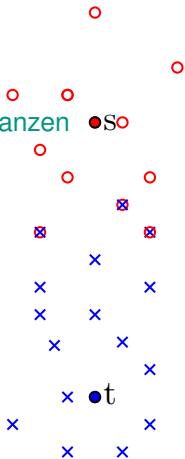
- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad



t

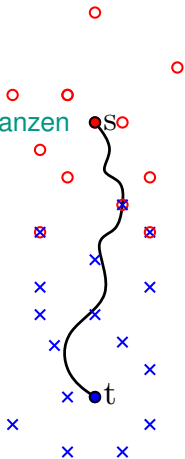
Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad



Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad



Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen \bullet s
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad

$s-t$ Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t) \dots$

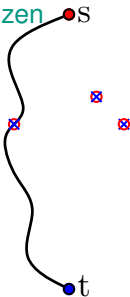


Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad

$s-t$ Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**

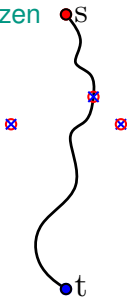


Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad

$s-t$ Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**

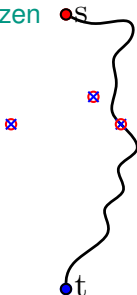


Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad

$s-t$ Anfrage:

- finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**



Vorbereitung:

- für jeden Knoten u , berechne zwei Label $L_f(u)$, $L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
 - $\text{dist}(u, v)$ für jeden Hub $v \in L_f(u)$
 - $\text{dist}(v, u)$ für jeden Hub $v \in L_b(u)$
- die Label müssen die **cover property** einhalten:
 $\forall s, t, L_f(s) \cap L_b(t)$ überdeckt den kürzesten $s-t$ Pfad

$s-t$ Anfrage:






- finde Knoten $v \in L_f(s) \cap L_b(t) \dots$
- \dots der $\text{dist}(s, v) + \text{dist}(v, t)$ **minimiert**



Wiederholung: Inner Join

a	b
	1
	2
	2
	3
	4

INNER JOIN






b	c
0	
1	
2	
4	
5	

USING(b)

Wiederholung: Inner Join









a	b
	1
	2
	2
	3
	4

INNER JOIN

b	c
0	
1	
2	
4	
5	

USING(b)

ergibt

a	b	c
	1	
	2	
	2	
	4	

HL kann man mit einer SQL-Datenbank umsetzen.

Lege zwei Tabellen an:

```
CREATE TABLE forward(  
    vertex_id integer ,  
    hub_id integer ,  
    distance integer  
);
```

```
CREATE TABLE backward(  
    vertex_id integer ,  
    hub_id integer ,  
    distance integer  
);
```

Die Distanz-Anfrage ist ein Join.

```
SELECT
  min(forward.distance+backward.distance)
FROM
  forward INNER JOIN backward USING(hub_id)
WHERE
  forward.vertex_id = source ,
  backward.vertex_id = target ;
```

Zwei zusätzliche Spalten:

```
CREATE TABLE forward(  
  vertex_id integer ,  
  hub_id integer ,  
  distance integer ,  
  previous_hub integer ,  
  shortcut_id integer  
);
```

- shortcut_id ist die ID der letzten Kante auf einem kürzesten up-down Weg (im CH Sinn).
- previous_hub ist Ausgangsknoten von shortcut_id.

Und eine zusätzliche Tabelle:

```
CREATE TABLE shortcut(  
  shortcut_id integer ,  
  shortcut_pos integer ,  
  arc_id integer  
);
```

- Idee: Speichere vorentpackte Shortcuts
- arc_id gibt die original Kante an
- shortcut_pos ist Position von arc_id im entpackten Pfad der shortcut_id entspricht
- Braucht nochmal etwa soviel Speicher wie die Hubs selber.

Schritt 1: Bestimme meeting_hub wie für Distanzanfrage.

Schritt 2: Baue temporäre Tabelle path:

```
CREATE TABLE path(  
    shortcut_id integer ,  
    up_down_path_pos integer  
);
```

Schritt 3: Befülle Tabelle

```
SET sequence = 0;
SET now = meeting_hub;
WHILE now <> source do
    SELECT shortcut_id ,previous_hub
    FROM forward
    WHERE forward.node = source AND forward.hub = now;
    SET now = previous_hub;
    INSERT INTO path VALUES(shortcut_id ,sequence);
    SET sequence = sequence -1;
END WHILE;
```

Achtung: Nicht alle Datenbanken unterstützen WHILE und nicht alle gleich.

Schritt 4: Join mit shortcut-Tabelle

```
SELECT
  arc_id
FROM
  shortcut INNER JOIN path USING(shortcut_id)
ORDERED BY
  up_down_path_pos ,
  shortcut_pos ;
```

Was bisher geschah

bisheriger Stoff:

- Punkt-zu-Punkt Abfragen

bisheriger Stoff:

- Punkt-zu-Punkt Abfragen

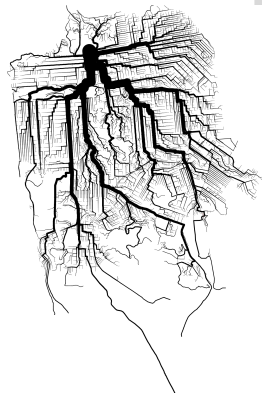
Erweiterte Anfragen

- one-to-many
- many-to-many
- one-to-all?

Kürzeste-Wege Bäume

Anfrage:

- gegeben ein nicht negativ gewichteter gerichteter Graph und Knoten s
- berechne Distanzen von s zu *allen* anderen



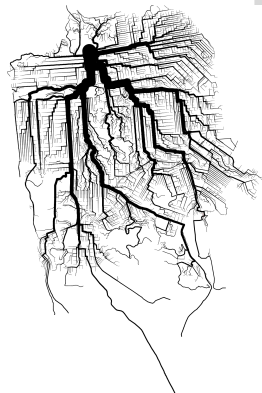
Kürzeste-Wege Bäume

Anfrage:

- gegeben ein nicht negativ gewichteter gerichteter Graph und Knoten s
- berechne Distanzen von s zu *allen* anderen

Lösung:

- Dijkstra [Dij59]



Anfrage:

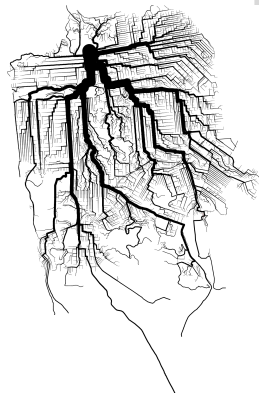
- gegeben ein nicht negativ gewichteter gerichteter Graph und Knoten s
- berechne Distanzen von s zu *allen* anderen

Lösung:

- Dijkstra [Dij59]

Fakten:

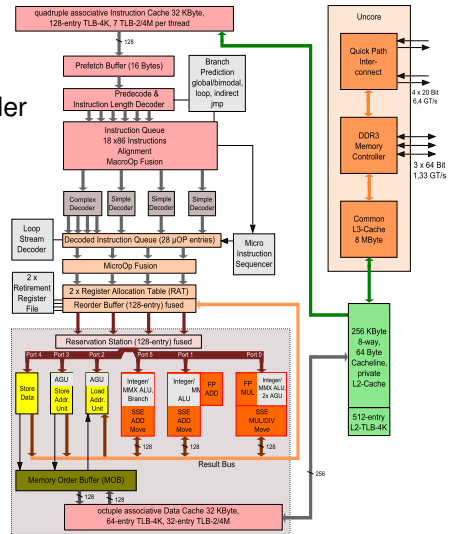
- $O(m + n \log n)$ mit Fibonacci Heaps [FT87]
- **linear** (mit kleiner Konstanten) in Praxis [Go10]
- Ausnutzung von moderner Hardware schwierig



Einige Fakten:

- viele Kerne
- mehr Kerne als Speichercontroller
- Hyperthreading
- Multi-Sockel System
- steile Speicherhierarchie
- Cache coherency

Intel Nehalem microarchitecture



GT/s: gigatransfers per second

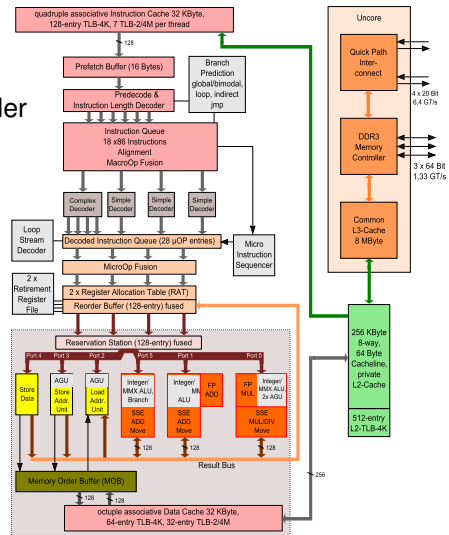
Einige Fakten:

- viele Kerne
- mehr Kerne als Speichercontroller
- Hyperthreading
- Multi-Sockel System
- steile Speicherhierarchie
- Cache coherency

Haupt Herausforderungen:

- Parallelisierung
- Speicherzugriff

Intel Nehalem microarchitecture



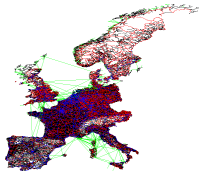
GT/s: gigatransfers per second

Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra: ≈ 3.0 s \Rightarrow nicht real-time



Core-i7 workstation (2.66 GHz)

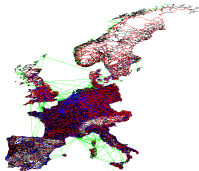
Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

Dijkstra: ≈ 3.0 s \Rightarrow nicht real-time

$n + m$ clock cycles: ≈ 15 ms \Rightarrow viel schneller



Core-i7 workstation (2.66 GHz)

Ausnutzen von Moderner Hardware

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen

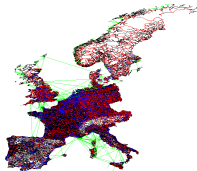
Dijkstra: ≈ 3.0 s \Rightarrow nicht real-time

$n + m$ clock cycles: ≈ 15 ms \Rightarrow viel schneller

BFS: ≈ 2.0 s

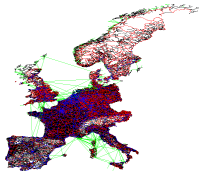
- Verlangsamung kommt nicht durch Priorityqueue allein

Core-i7 workstation (2.66 GHz)



Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen
 - Dijkstra: ≈ 3.0 s \Rightarrow nicht real-time
 - $n + m$ clock cycles: ≈ 15 ms \Rightarrow viel schneller
 - BFS: ≈ 2.0 s
- Verlangsamung kommt nicht durch Priorityqueue allein



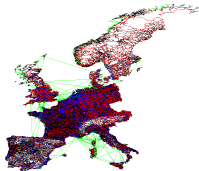
Core-i7 workstation (2.66 GHz)

Parallelisierung:

- Spekulation
- Δ -stepping [MS03],[MBBC09]
- mehr Operationen als Dijkstra
- keine grosse Beschleunigung auf dünnen Graphen

Daten Lokalität

- Eingabe: West Europa
- 18M Knoten, 23M Strassen
 - Dijkstra: ≈ 3.0 s \Rightarrow nicht real-time
 - $n + m$ clock cycles: ≈ 15 ms \Rightarrow viel schneller
 - BFS: ≈ 2.0 s
- Verlangsamung kommt nicht durch Priorityqueue allein



Core-i7 workstation (2.66 GHz)

Parallelisierung:

- Spekulation
- Δ -stepping [MS03],[MBBC09]
- mehr Operationen als Dijkstra
- keine grosse Beschleunigung auf dünnen Graphen
- Berechnen von mehreren Bäumen ist einfach

Ansatz 1

Idee:

- Umordnen der Knoten im Graphen

Idee:

- Umordnen der Knoten im Graphen

algorithm	details	time per tree [ms]		
		random	input	DFS
Dijkstra	binary heap	11159	5859	5180
	Dial	7767	3538	2908
	smart queue	7991	3556	2826
BFS	—	6060	2445	2068

(Achtung: One-to-All & Älterer Rechner)

Idee:

- Umordnen der Knoten im Graphen

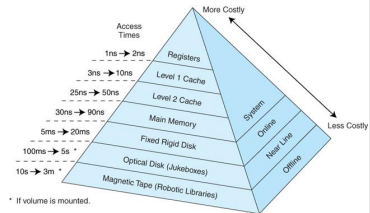
algorithm	details	time per tree [ms]		
		random	input	DFS
Dijkstra	binary heap	11159	5859	5180
	Dial	7767	3538	2908
	smart queue	7991	3556	2826
BFS	—	6060	2445	2068

(**Achtung:** One-to-All & Älterer Rechner)

⇒ keine grosse Beschleunigung

Dijkstra's Algorithmus:

- moderne Hardware nicht voll zu nutzen
- Hauptprobleme:
 - Daten Lokalität
 - Parallelisierung

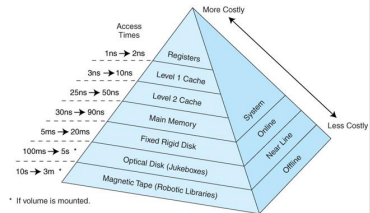


Dijkstra's Algorithmus:

- moderne Hardware nicht voll zu nutzen
- Hauptprobleme:
 - Daten Lokalität
 - Parallelisierung

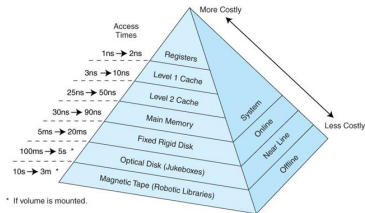
Fragen:

- hilft Vorbereitung?
- wie?
- Ansatzpunkt?



Dijkstra's Algorithmus:

- moderne Hardware nicht voll zu nutzen
- Hauptprobleme:
 - Daten Lokalität
 - Parallelisierung



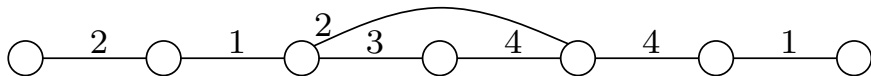
Fragen:

- hilft Vorbereitung?
- wie?
- Ansatzpunkt?

PHAST: Hardware-Accelerated Shortest path Trees

Contraction Hierarchies

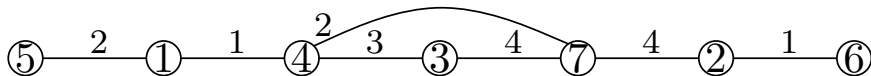
preprocessing:



Contraction Hierarchies

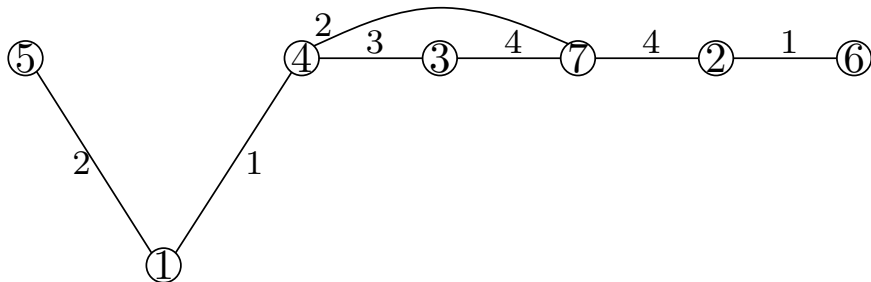
preprocessing:

- ordne Knoten nach Wichtigkeit



preprocessing:

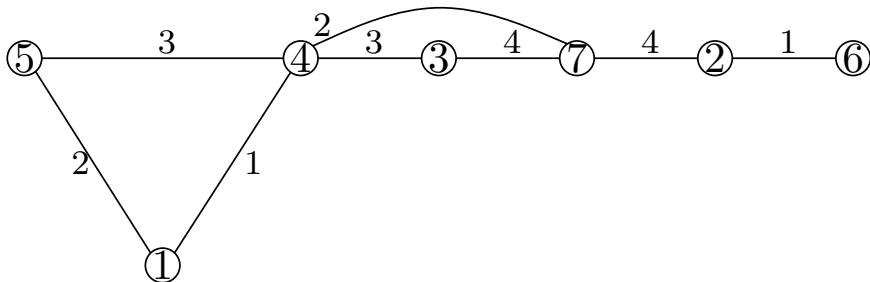
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

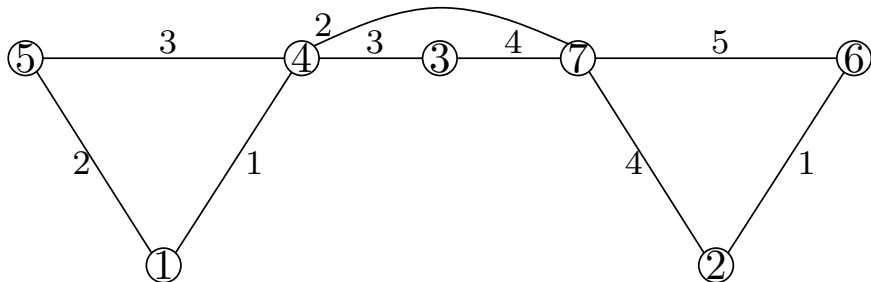
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

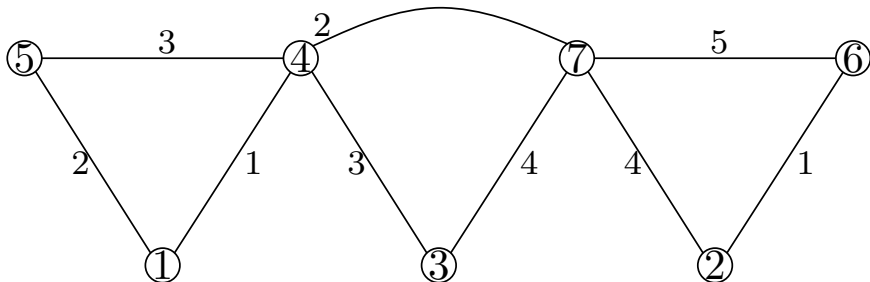
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

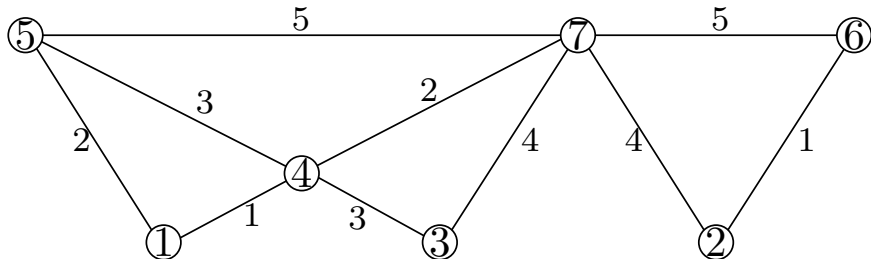
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

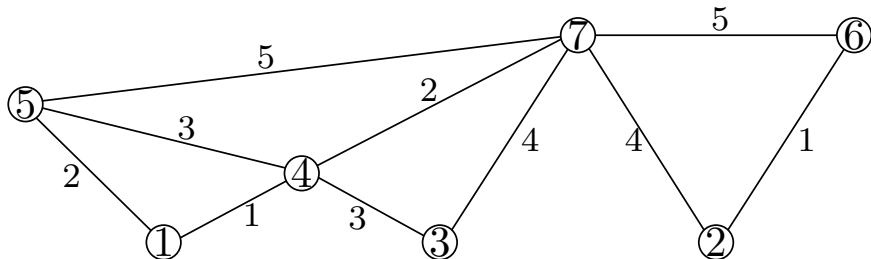
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

preprocessing:

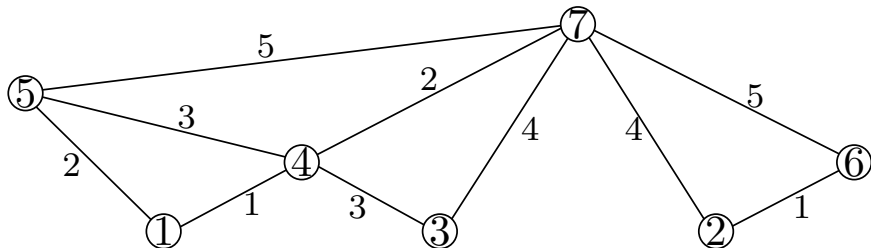
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Contraction Hierarchies

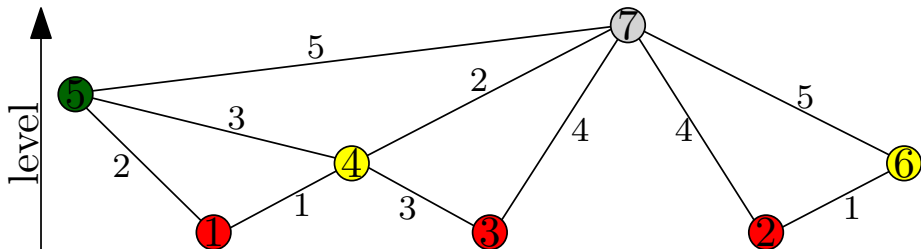
preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



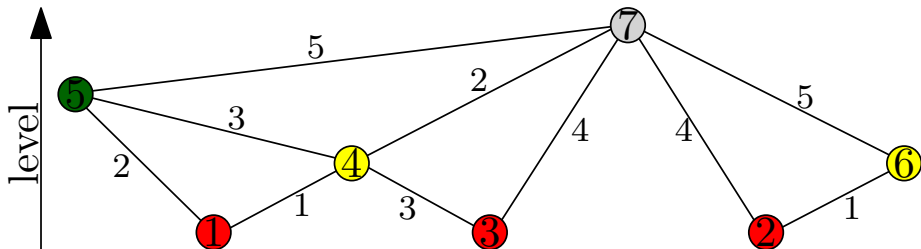
preprocessing:

- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung (ca. 150 in Strassennetzwerken)



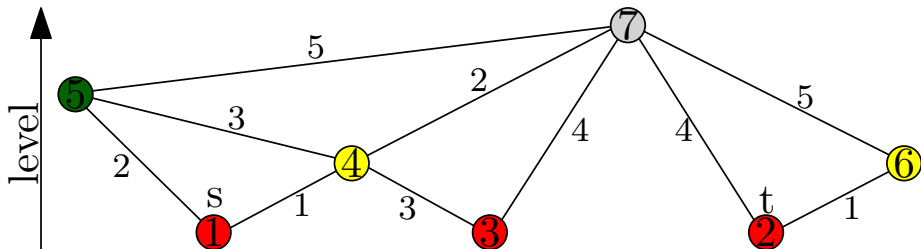
Punkt-zu-Punkt Anfragen

- modifizierter **bidirektionaler** Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



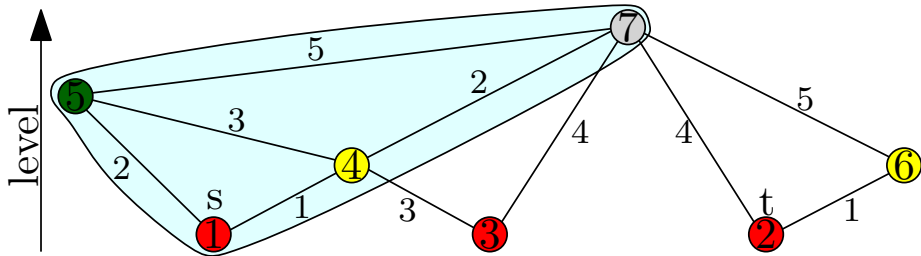
Punkt-zu-Punkt Anfragen

- modifizierter **bidirektionaler** Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



Punkt-zu-Punkt Anfragen

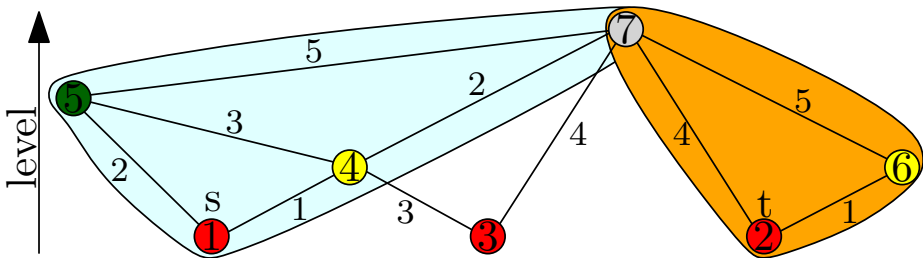
- modifizierter **bidirektionaler** Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



Contraction Hierarchies

Punkt-zu-Punkt Anfragen

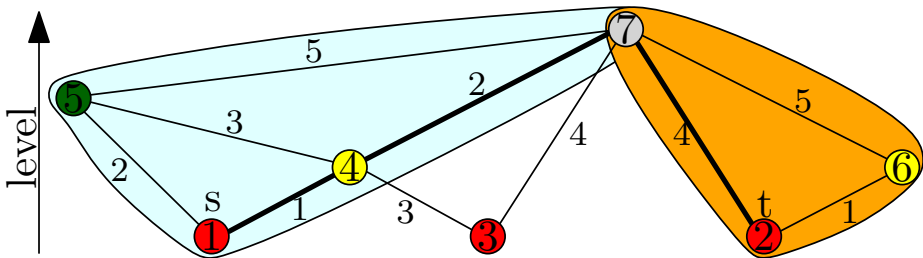
- modifizierter **bidirektionaler** Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter **bidirektionaler** Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten



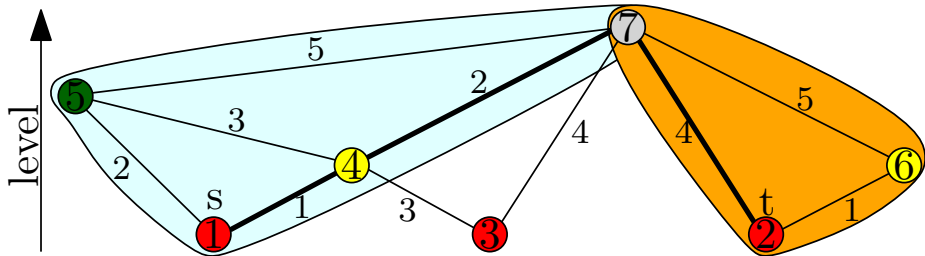
Contraction Hierarchies

Punkt-zu-Punkt Anfragen

- modifizierter **bidirektionaler** Dijkstra
- folge nur Kanten zu wichtigeren Knoten
- besucht nur 500 Knoten

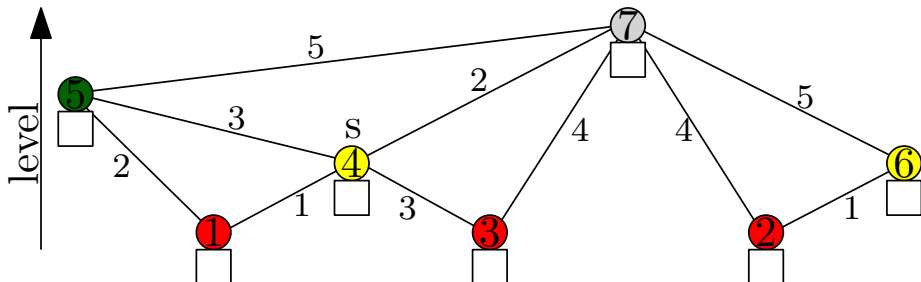
Korrektheit:

- es gibt einen wichtigsten Knoten auf dem Pfad
- dieser wird von Vorwärts- und Rückwärtssuche gescannt



Neuer Anfragealgorithmus

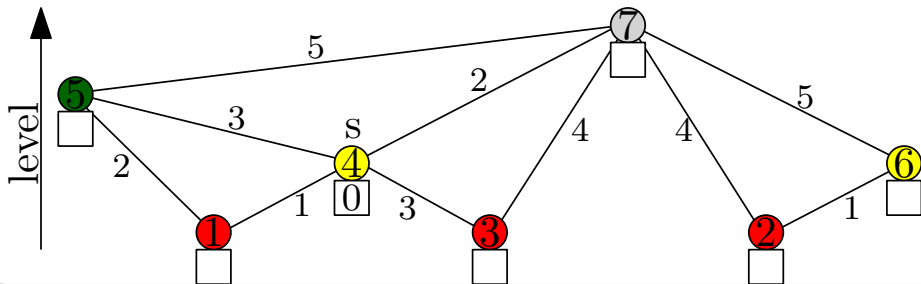
one-to-all Suche von s :



Neuer Anfragealgorithmus

one-to-all Suche von s :

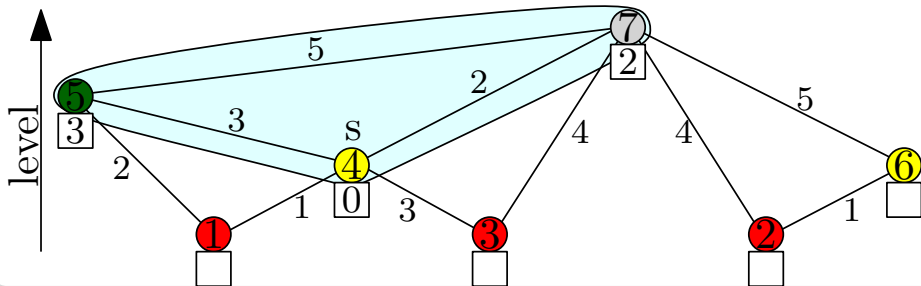
- vorwärts CH Suche von s (≈ 0.05 ms)



Neuer Anfragealgorithmus

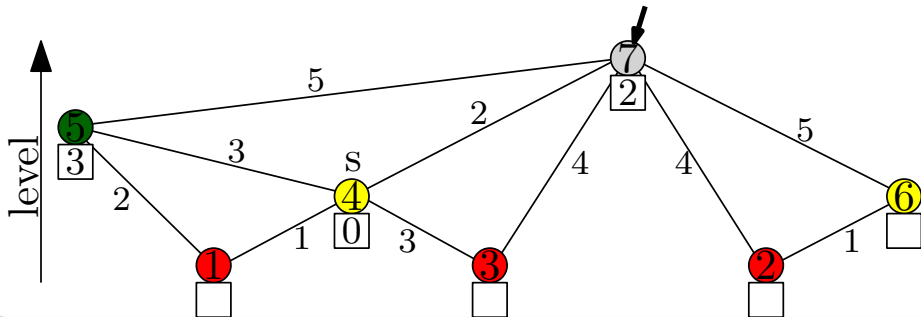
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten



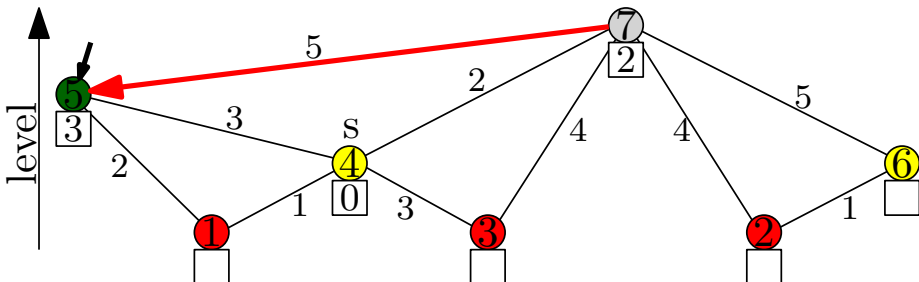
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



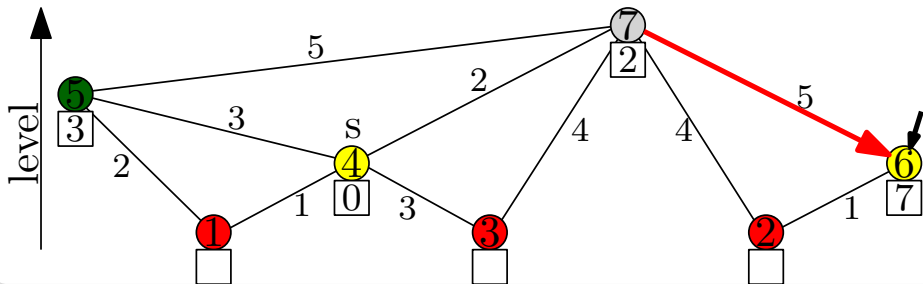
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



one-to-all Suche von s :

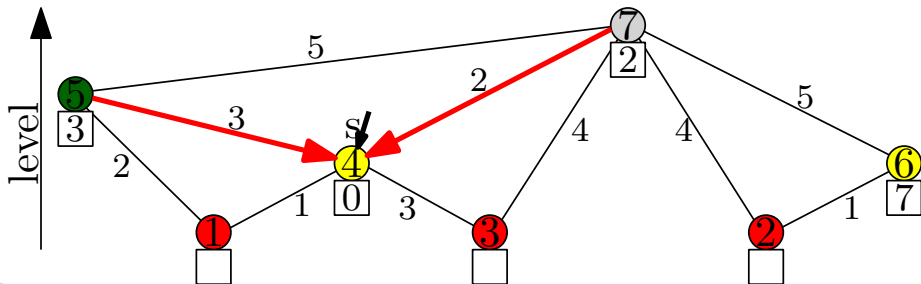
- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



Neuer Anfragealgorithmus

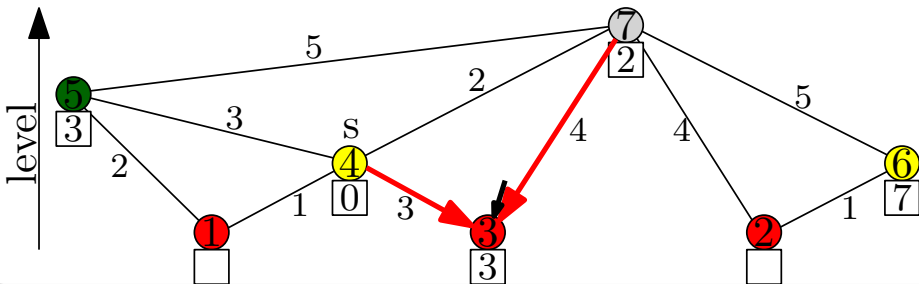
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



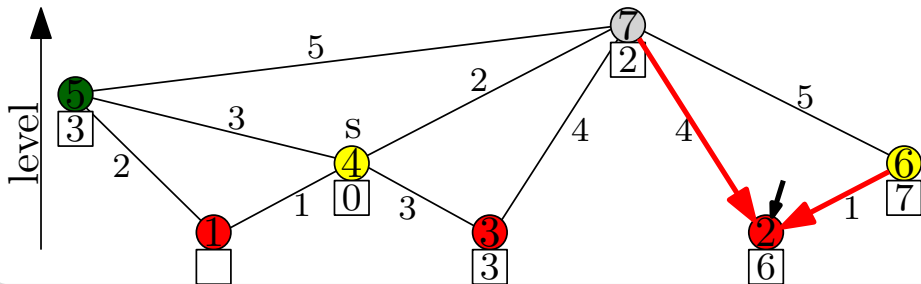
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



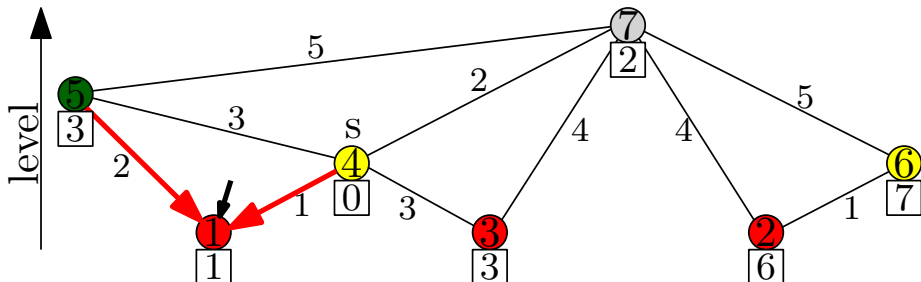
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



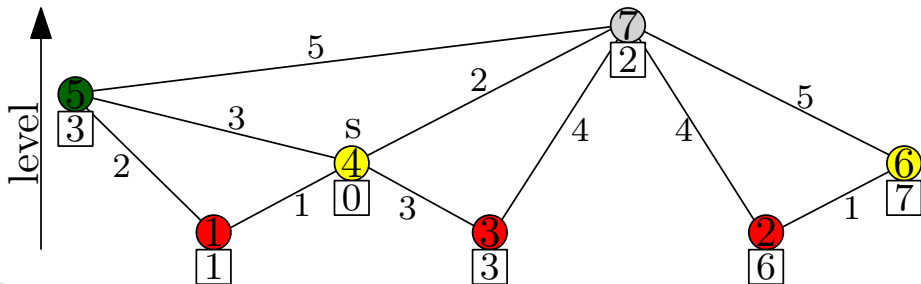
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$



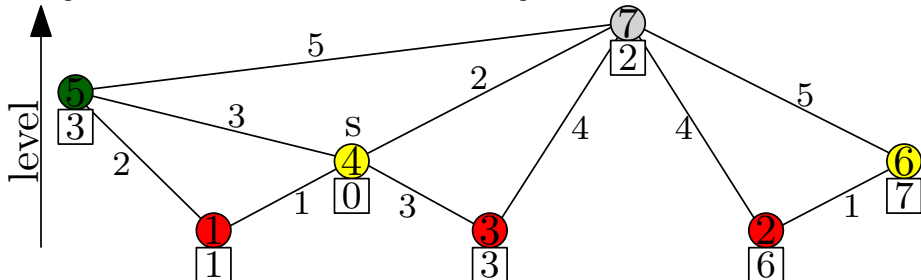
one-to-all Suche von s :

- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- **top-down** Bearbeitung ohne Priority Queue (ca. 2.0 s)



one-to-all Suche von s :

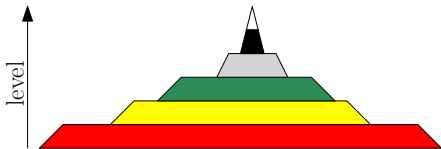
- vorwärts CH Suche von s (≈ 0.05 ms)
- setze Distanzen d für alle erreichten Knoten
- bearbeite alle Knoten u in **inverser** Levelordnung:
 - checke **eingehende** Kanten (v, u) mit $lev(v) > lev(u)$
 - setze $d(u) = \min\{d(u), d(v) + w(v, u)\}$
- **top-down** Bearbeitung ohne Priority Queue (ca. 2.0 s)
- genauso schnell wie BFS. Warum das ganze?



Analyse

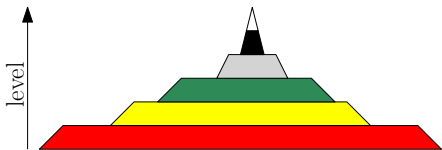
Beobachtung:

- top-down Prozess ist der Flaschenhals



Beobachtung:

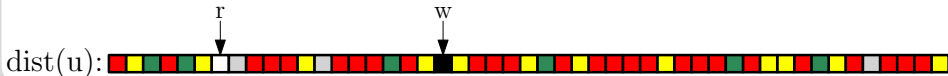
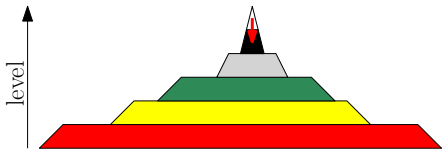
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**



dist(u): 

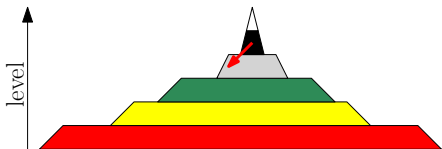
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**



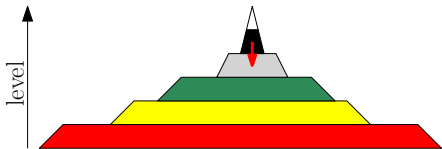
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**



Beobachtung:

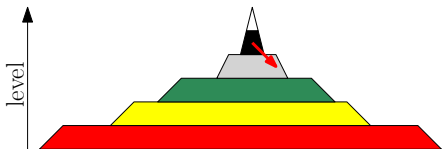
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**



Analyse

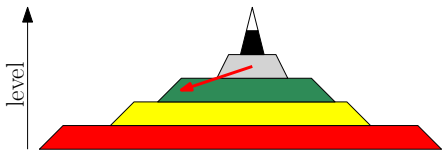
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**



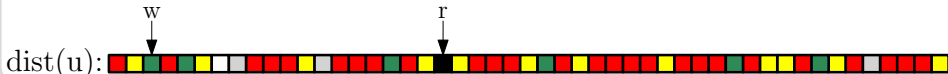
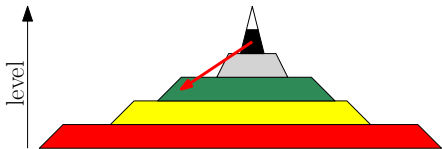
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**



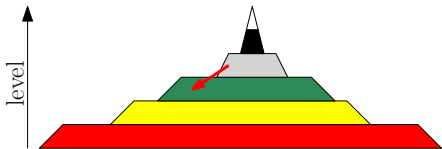
Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**



Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

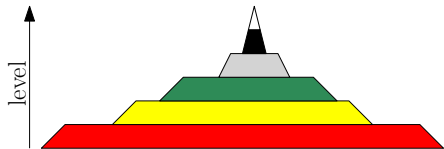


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**

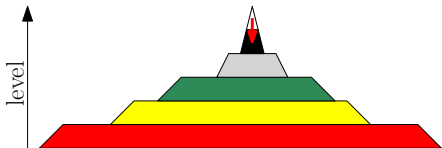


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**

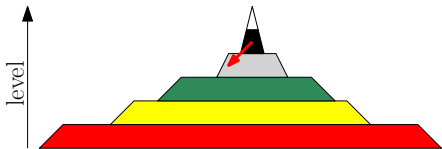


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**

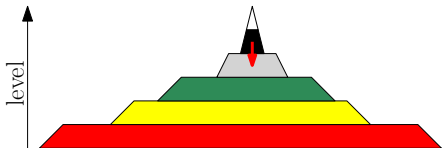


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**

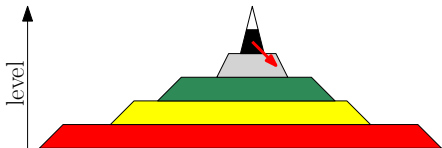


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**

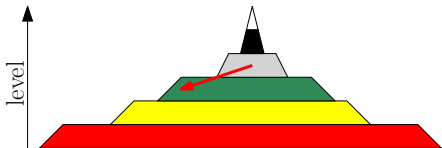


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**

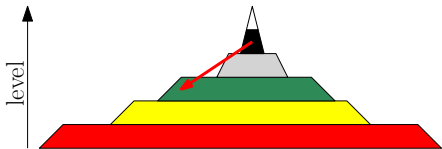


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**

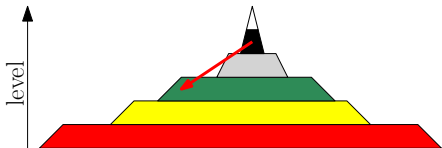


Beobachtung:

- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten
und schreiben der Distanzen
wird zu einem **sequenziellen Sweep**
- ⇒ 172 ms pro Baum

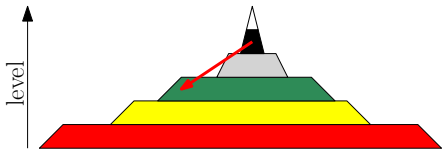


Beobachtung:

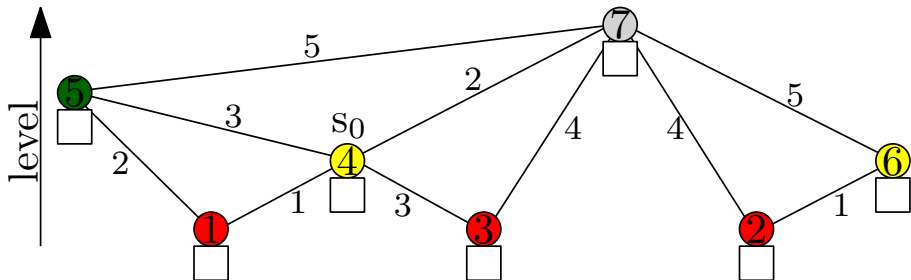
- top-down Prozess ist der Flaschenhals
- Zugriff auf die Daten ist immer noch **ineffizient**
- Zugriffsmuster sind **unabhängig** von s

Idee:

- speicher G_{\uparrow} und G_{\downarrow} separat
 - **Umordnung** der Knoten, Kanten, und Distanzlabel nach Level
- ⇒ lesen der Kanten und schreiben der Distanzen wird zu einem **sequenziellen Sweep**
- ⇒ 172 ms pro Baum
- aber lesen der Distanzen immer noch **ineffizient**

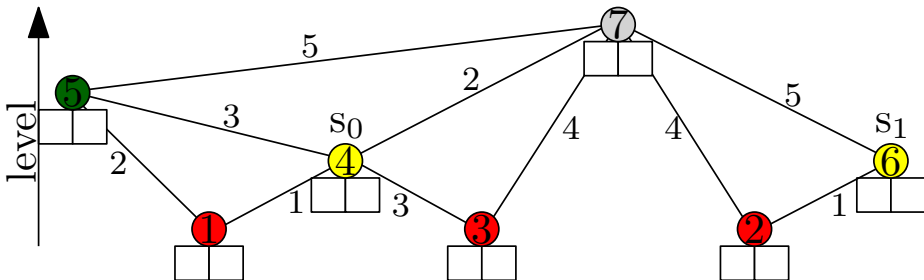


Szenario: Multiple Startknoten



Szenario: Multiple Startknoten

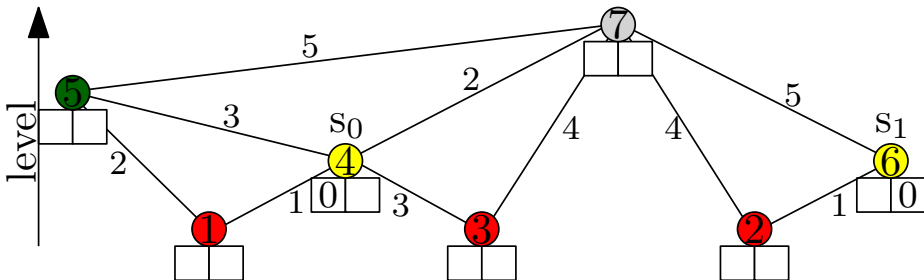
Idee:



Szenario: Multiple Startknoten

Idee:

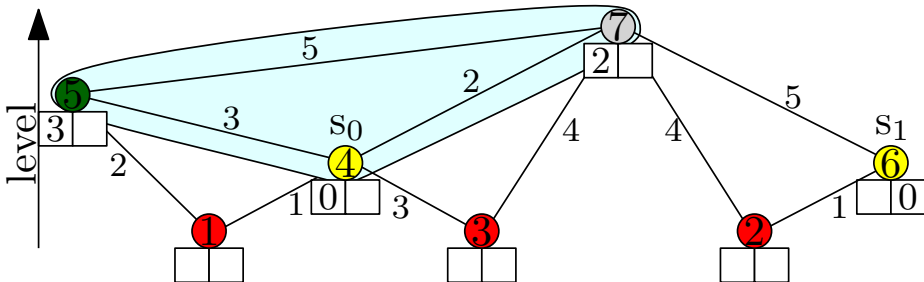
- k Vorwärtssuchen



Szenario: Multiple Startknoten

Idee:

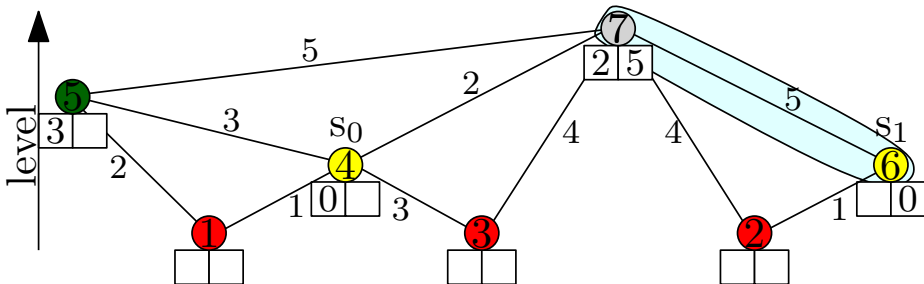
- k Vorwärtssuchen



Szenario: Multiple Startknoten

Idee:

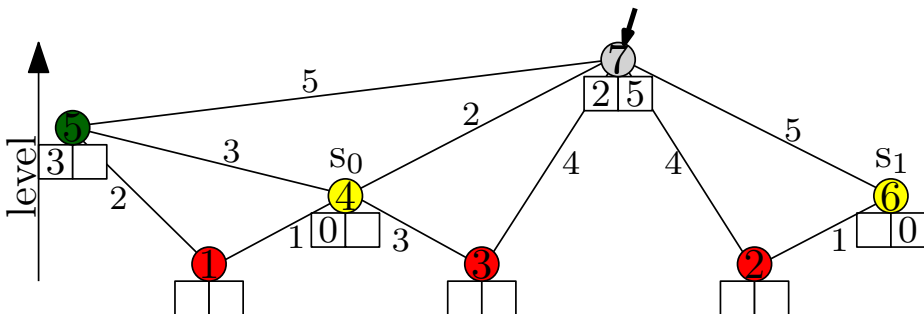
- k Vorwärtssuchen



Szenario: Multiple Startknoten

Idee:

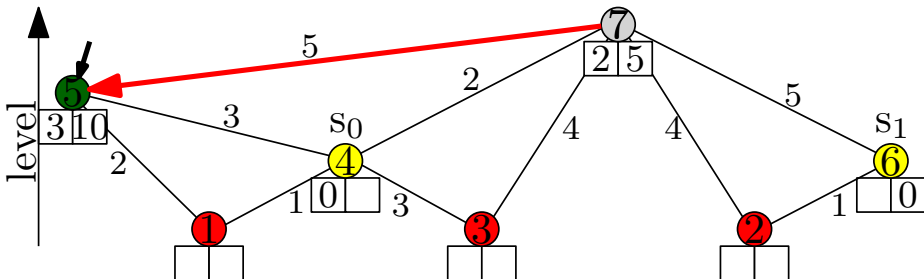
- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

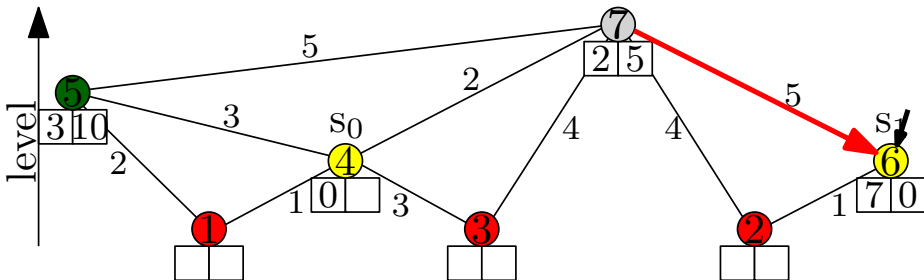
- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

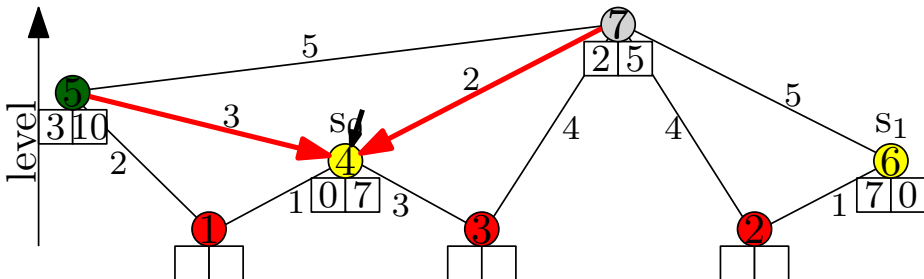
- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

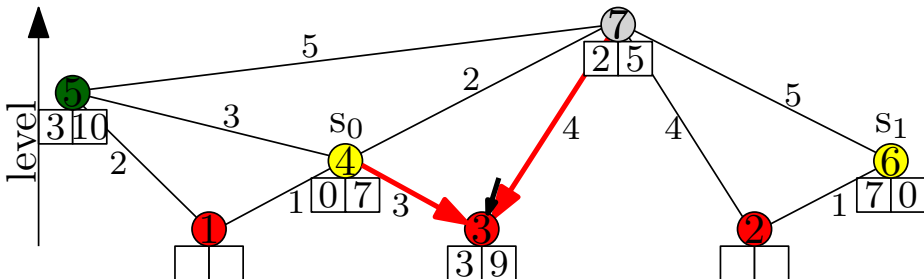
- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

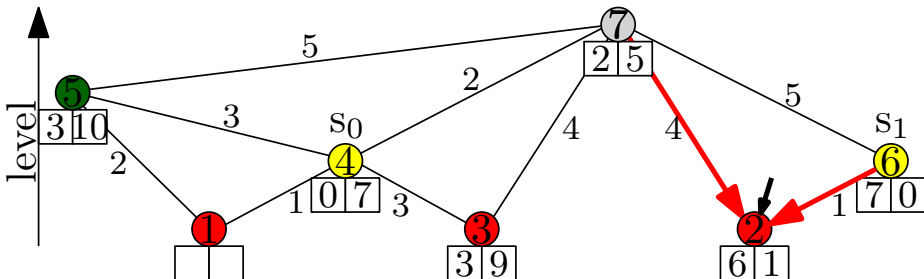
- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

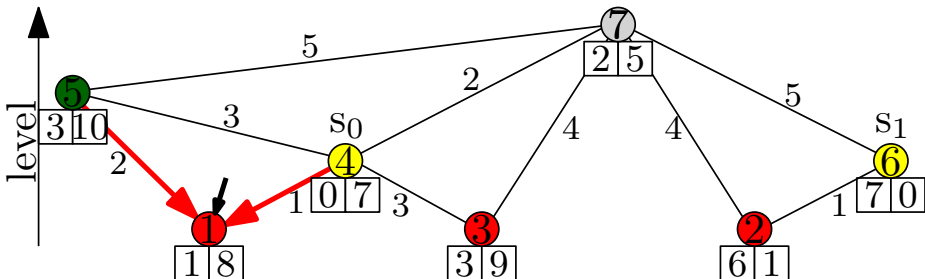
- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

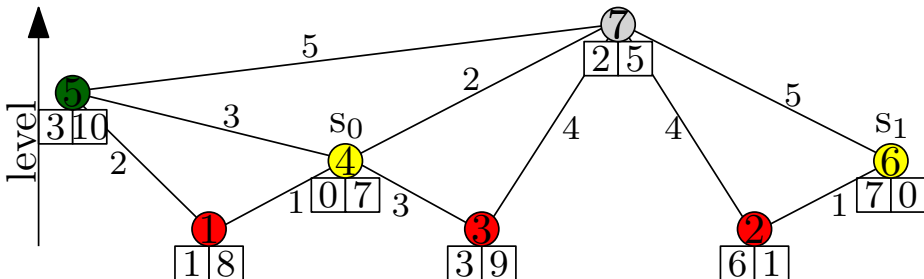
- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten



Szenario: Multiple Startknoten

Idee:

- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)



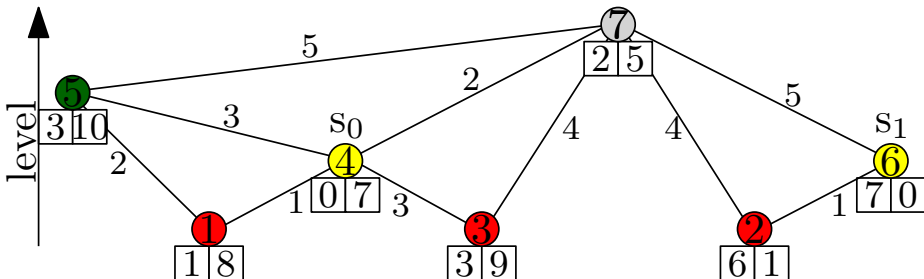
Szenario: Multiple Startknoten

Idee:

- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)

SSE:

- 128-bit Register
- Basisoperationen (min, add) für vier 32-bit Integer parallel
- scanne 4 Distanzlabel auf einmal



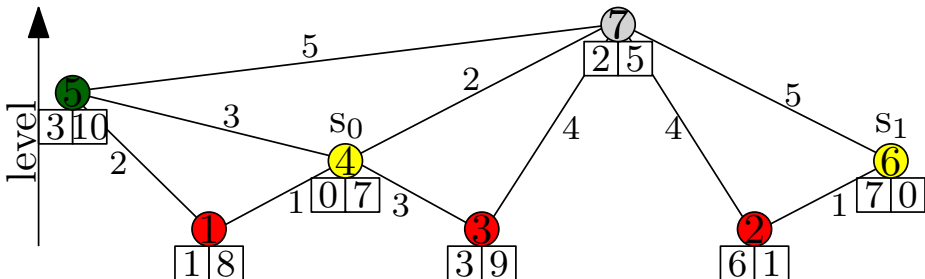
Szenario: Multiple Startknoten

Idee:

- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)

SSE:

- 128-bit Register
- Basisoperationen (min, add) für vier 32-bit Integer parallel
- scanne 4 Distanzlabel auf einmal
- 37.1 ms pro Baum ($k = 16$)



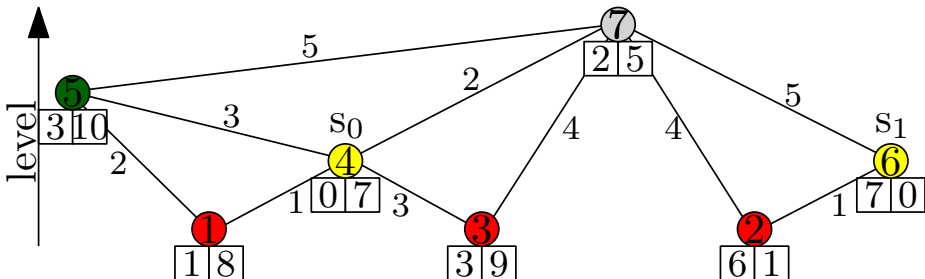
Szenario: Multiple Startknoten

Idee:

- k Vorwärtssuchen
- ein sweep (update aller k Werte)
- speicher Distanzlabel pro Knoten
- 96.8 ms pro Baum ($k = 16$)

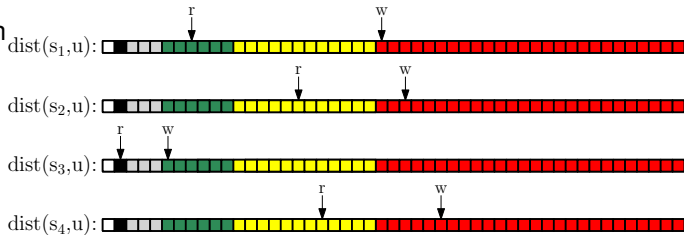
SSE:

- 128-bit Register
- Basisoperationen (min, add) für vier 32-bit Integer parallel
- scanne 4 Distanzlabel auf einmal
- 37.1 ms pro Baum ($k = 16$)
- Sandy Bridge Architektur: 256-bit Register



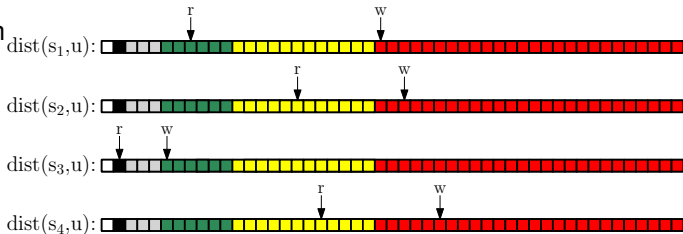
ganz einfach

- nach Startknoten



ganz einfach

- nach Startknoten

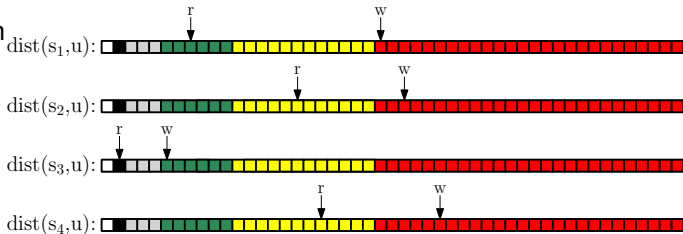


Ergebnisse:

- 16 Startknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten \Rightarrow 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum

ganz einfach

- nach Startknoten

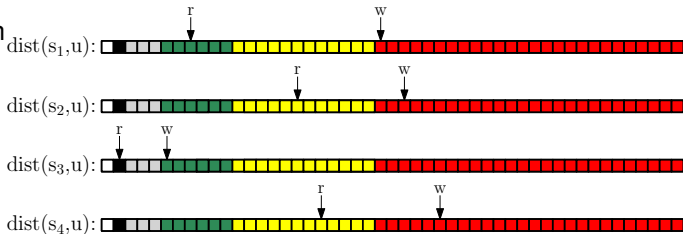


Ergebnisse:

- 16 Startknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten \Rightarrow 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?

ganz einfach

- nach Startknoten

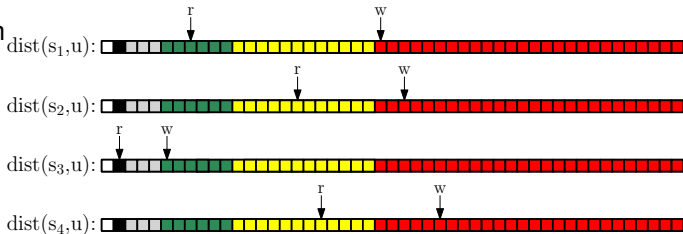


Ergebnisse:

- 16 Startknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten \Rightarrow 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?
- lower bound tests zeigen: nah an Speicherbandbreite

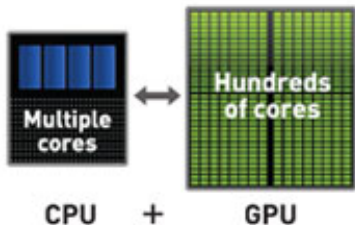
ganz einfach

- nach Startknoten



Ergebnisse:

- 16 Startknoten pro Sweep (updates via SSE)
- multi-core nach Startknoten \Rightarrow 64 Startknoten parallel (4 cores)
- 18.8 ms per Baum
- Warum kein perfekter Speedup?
- lower bound tests zeigen: nah an Speicherbandbreite
- kann eine GPU helfen?



Intel Xeon X5680:

- 3.33 GHz, oft ≥ 10 GB RAM
- 32 GB/s Speicherbandbreite
- 6 Kerne
- SIMD/SSE mit 4 floats/ints per Vektor

NVIDIA GTX 580:

- 772 MHz, 1.5 GB RAM
- 192 GB/s Speicherbandbreite
- 16 Kerne, 32 parallele Threads pro Kern
⇒ 512 parallele Threads
- eingeschränkte Berechnungen

512 Threads?

Gemeinsamer Codezeiger:

- GPU Threads innerhalb eines Kerns haben einen gemeinsamen Codezeiger.
- If-Bedingung wahr für ein Thread → Alle Threads durchlaufen den Wahr-Block.
 - Aber für die falsch-Threads werden die Anweisungen ausmaskiert.
- Schleifen dauern immer so lange wie der längste Thread braucht.
- Stichwort: Single Instruction Multiple Threads

Verwandt mit SIMD:

- Jede Vektorkomponente ist ein "Thread".
- Beim komponentenweisen if-else wird auch eine Seite ausmaskiert.

Gemeinsamer Codezeiger:

- GPU Threads innerhalb eines Kerns haben einen gemeinsamen Codezeiger.
- If-Bedingung wahr für ein Thread → Alle Threads durchlaufen den Wahr-Block.
 - Aber für die falsch-Threads werden die Anweisungen ausmaskiert.
- Schleifen dauern immer so lange wie der längste Thread braucht.
- Stichwort: Single Instruction Multiple Threads

Verwandt mit SIMD:

- Jede Vektorkomponente ist ein "Thread".
- Beim komponentenweisen if-else wird auch eine Seite ausmaskiert.

Fairerer Vergleich:

- $772 \text{ MHz} \times 16 \text{ Kerne} \times 32 \text{ "Threads"} = 395264$
- $3.33 \text{ GHz} \times 6 \text{ Kerne} \times 4 \text{ SIMD-Vektor} = 79992$
- In optimalen Bedingungen bis zu 5-mal mehr Berechnungen

- Der Transfer CPU-Speicher \leftrightarrow GPU-Speicher ist langsam.
- GPU-Threads zwischen Kernen synchronisieren ist sehr teuer.
Die CPU muss warten bis der erste Job fertig ist und dann einen neuen starten.
Dauert schon mal ein paar Mikrosekunden.
- Keine Cache-coherency zwischen Cores:
 - 1 Core A schreibt einen neuen Wert and die Adresse 42.
 - 2 Core B liest die Adresse 42 aus, aber erhält den alten Wert.Erklärung: Der neue Wert steckt noch im Cache von A fest.
- Langsam, wenn GPU-Threads durcheinander auf den Speicher zugreifen.

GPHAST - Ideen

Beobachtungen:

- Auswärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Beobachtungen:

- Auswärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Idee:

- speicher CH und Distanzarray auf der GPU
- Auswärtssuche auf der CPU
- kopiere Suchraum zur GPU (weniger als 2 kB)
- linearen Sweep auf der GPU

Beobachtungen:

- Auswärtsuche ist schnell
- Flaschenhals ist der lineare Sweep
- Speicherbandbreite das Problem

Idee:

- speicher CH und Distanzarray auf der GPU
- Auswärtssuche auf der CPU
- kopiere Suchraum zur GPU (weniger als 2 kB)
- linearen Sweep auf der GPU

Problem:

- nicht genug Speicher auf GPU um tausende von Bäumen parallel zu bearbeiten
- wir müssen eine einzelne Baumberechnung parallelisieren

Beobachtung:

- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten



Beobachtung:

- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten



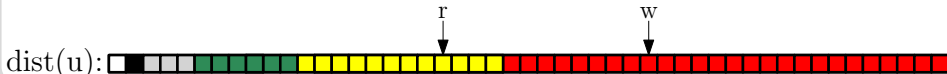
Beobachtung:

- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten



Beobachtung:

- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten



Beobachtung:

- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten

Idee:

- scanne alle Knoten auf Level i **parallel**
- Synchronization nach jedem Level
- ein Thread pro Knoten



Beobachtung:

- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten

Idee:

- scanne alle Knoten auf Level i **parallel**
- Synchronization nach jedem Level
- ein Thread pro Knoten



Beobachtung:

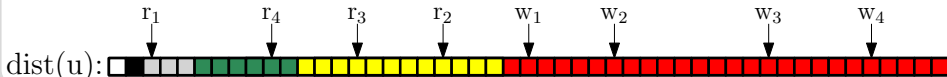
- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten

Idee:

- scanne alle Knoten auf Level i **parallel**
- Synchronization nach jedem Level
- ein Thread pro Knoten

results:

- 5.5 ms auf NVIDIA GTX 580
- Beschleunigung von 511 gegenüber Dijkstra



Beobachtung:

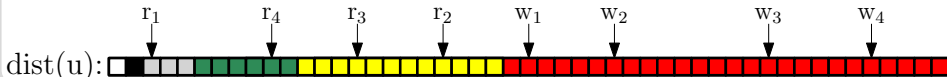
- beim Scannen von Level i :
 - nur eingehende Kanten von level $> i$ wichtig
 - schreiben von Distanzlabeln in Level i , lesen von Level $> i$
 - Distanzlabel für Level $> i$ sind korrekt
- scannen eines Level- i Knoten ist **unabhängig** von anderen Level- i Knoten

Idee:

- scanne alle Knoten auf Level i **parallel**
- Synchronization nach jedem Level
- ein Thread pro Knoten

results:

- 5.5 ms auf NVIDIA GTX 580
- Beschleunigung von 511 gegenüber Dijkstra
- (mehrere Bäume: 2.2 ms)



PHAST auf 4-Kern Workstation (Core-i7 920)

sources/ sweep	time per tree [ms]					
	1 core		2 cores		4 cores	
1	171.9		86.7		47.1	
4	121.8	(67.6)	61.5	(35.5)	32.5	(24.4)
8	105.5	(51.2)	53.5	(28.0)	28.3	(20.8)
16	96.8	(37.1)	49.4	(22.1)	25.9	(18.8)

Werte in Klammern mit SSE aktiviert

PHAST auf Nvidia GTX 580

trees / sweep	memory [MB]	time [ms]
1	395	5.53
2	464	3.93
4	605	3.02
8	886	2.52
16	1448	2.21

algorithm	device	Europe		USA	
		time	distance	time	distance
Dijkstra	4-core workstation	947.72	609.19	1269.12	947.75
	12-core server	288.81	177.58	380.40	280.17
	48-core server	168.49	108.58	229.00	167.77
PHAST	4-core workstation	18.81	22.25	27.11	28.81
	12-core server	7.20	8.27	10.42	10.71
	48-core server	4.03	5.03	6.18	6.58
GPHAST	GTX 580	2.21	3.88	3.41	4.65

Beobachtung:

- Beschleunigung für Distanzmetrik geringer

All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580		

All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580	11h	

All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	
	12-core server	60d	
	48-core server	35d	
PHAST	4-core workstation	94h	
	12-core server	36h	
	48-core server	20h	
GPHAST	GTX 580	11h	

4-core workstation without GPU: 163 watts

4-core workstation with GPU: 375 watts

12-core server: 332 watts

48-core server: 747 watts

All-Pairs Shortest Paths

Eingabe: Europa mit Reisezeiten

algorithm	device	time	energy [MJ]
Dijkstra	4-core workstation	197d	2780.6
	12-core server	60d	1725.9
	48-core server	35d	2265.5
PHAST	4-core workstation	94h	55.2
	12-core server	36h	43.0
	48-core server	20h	54.2
GPHAST	GTX 580	11h	14.9

4-core workstation without GPU: 163 watts

4-core workstation with GPU: 375 watts

12-core server: 332 watts

48-core server: 747 watts

bis jetzt:

- nur Distanzen berechnet, nicht Bäume

bis jetzt:

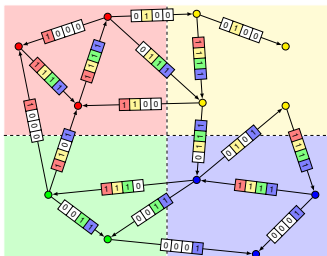
- nur Distanzen berechnet, nicht Bäume

Idee:

- iteration über alle Kanten (1 Thread pro Kante)
- Wenn $d(v) + \text{len}(v, u) = d(u)$ dann ist v der Vorgänger von u (sofern kürzeste Wege eindeutig sind)

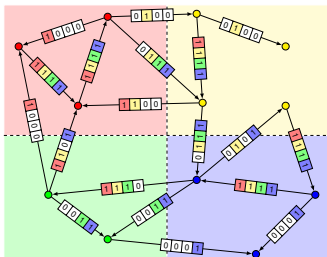
Idee:

- benutze GPHAST zum Berechnen der Bäume von den Randknoten aus



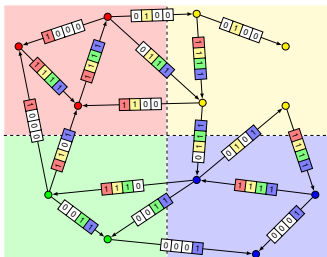
Idee:

- benutze GPHAST zum Berechnen der Bäume von den Randknoten aus
- setze Flags durch zusätzlichen Sweep auf GPU
Wichtig, weil alle Bäume “auf CPU kopieren” teuer ist



Idee:

- benutze GPHAST zum Berechnen der Bäume von den Randknoten aus
- setze Flags durch zusätzlichen Sweep auf GPU
Wichtig, weil alle Bäume “auf CPU kopieren” teuer ist
- Vorberechnung sinkt von 17 Stunden auf 3 Minuten



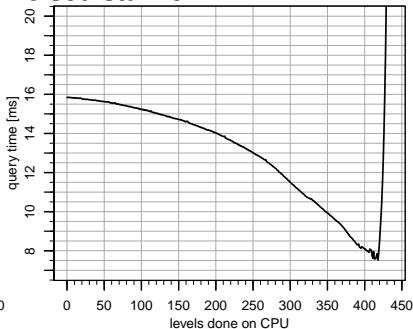
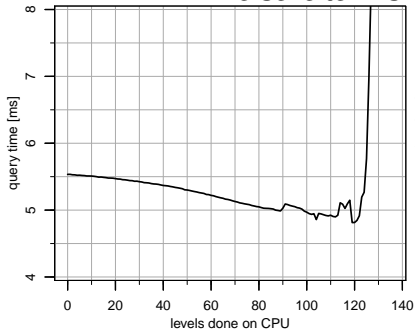
Beobachtung:

- Synchronisation des Level kostet Zeit auf der GPU ($5 \mu\text{s}$ pro Level)
- obere Level sind klein

Idee:

- beginne linearen Sweep auf CPU (bis level k)
- kopiere Suchraum und alle Distanzlabel für Knoten oberhalb k zur GPU
- restlicher Scan auf der GPU

Reisezeiten vs. Reisedistanzen



Es lohnt sich, ein Paar Level auf der CPU zu berechnen.

- neuer Algorithmus für kürzeste Wege Bäume
- **skaliert** auf Modern Architektur
- ein Baum auf GPU: **5.5 ms**
(ungefähr **0.31 ns** pro Eintrag)
- **real-time** Berechnung von kompletten Bäumen
- 16 Bäume auf einer GPU auf einmal: 2.2 ms pro Baum
(ungefähr **0.13 ns** pro Eintrag)
- APSP in **11 Stunden** (auf workstation mit einer GPU),
anstellen von 6 Monaten (auf 4 Kernen)
- erlaub APSP-basierte Berechnungen
- **150** mal Energie-effizienter als Dijkstras Algorithmus
- funktioniert nur, wenn CH funktioniert

Wie kann man folgende Anwendungen mit PHAST beschleunigen?
Welche sind auf der GPU schneller?

- Stau-Updates
- APSP-Matrix berechnen
- Diameter berechnen
- ALT
- Reach

Wie kann man folgende Anwendungen mit PHAST beschleunigen?
Welche sind auf der GPU schneller?

- Stau-Updates
Geht nicht, da 12h Reaktionszeit zu langsam sind
- APSP-Matrix berechnen
Geht nicht, da es nicht genug RAM gibt um die Ergebnisse im Speicher zu halten.
- Diameter berechnen
Geht. **CPU**: Straight-Forward **GPU**: Geht, aber man darf nur eine parallele Maximumbildung über alle n Einträge machen. Sonst ist es langsamer als auf der CPU.
- ALT
Geht auf der CPU, GPU -> CPU Transfer zu teuer
- Reach
Geht. **CPU**: Straight-Forward **GPU**: Baumhöhenberechnung dominiert. Auf der CPU ist es oft schneller.

Mittwoch, 21.5.2014 fällt aus

Montag, 26.5.2014 (Auf Englisch)



Daniel Delling, Andrew V. Goldberg, Andreas Nowatzky, and Renato F. Werneck.
PHAST: Hardware-accelerated shortest path trees.
Journal of Parallel and Distributed Computing, 73(7):940–952, 2013.