Karlsruher Institut für Technologie

Algorithmen für Routenplanung

8. Vorlesung, Sommersemester 2016

Ben Strasser | 11. Mai 2016

Institut für Theoretische Informatik · Algorithmik · Prof. Dr. Dorothea Wagner

Kürzeste Wege in Straßennetzwerken

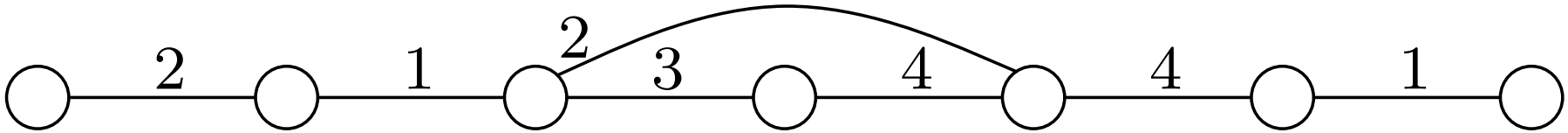
Beschleunigungstechniken (Fortsetzung)

- Wiederholung: CH
- Hub-Labels
- HLDB

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

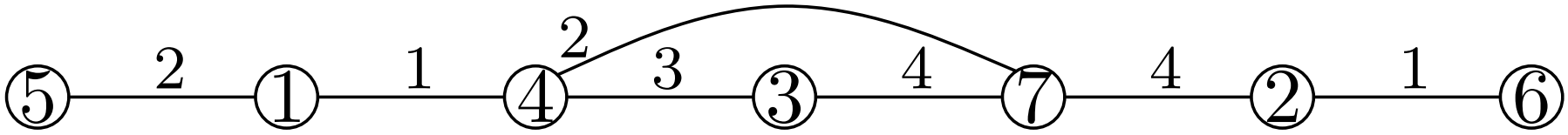
preprocessing:



Wiederholung: CH

preprocessing:

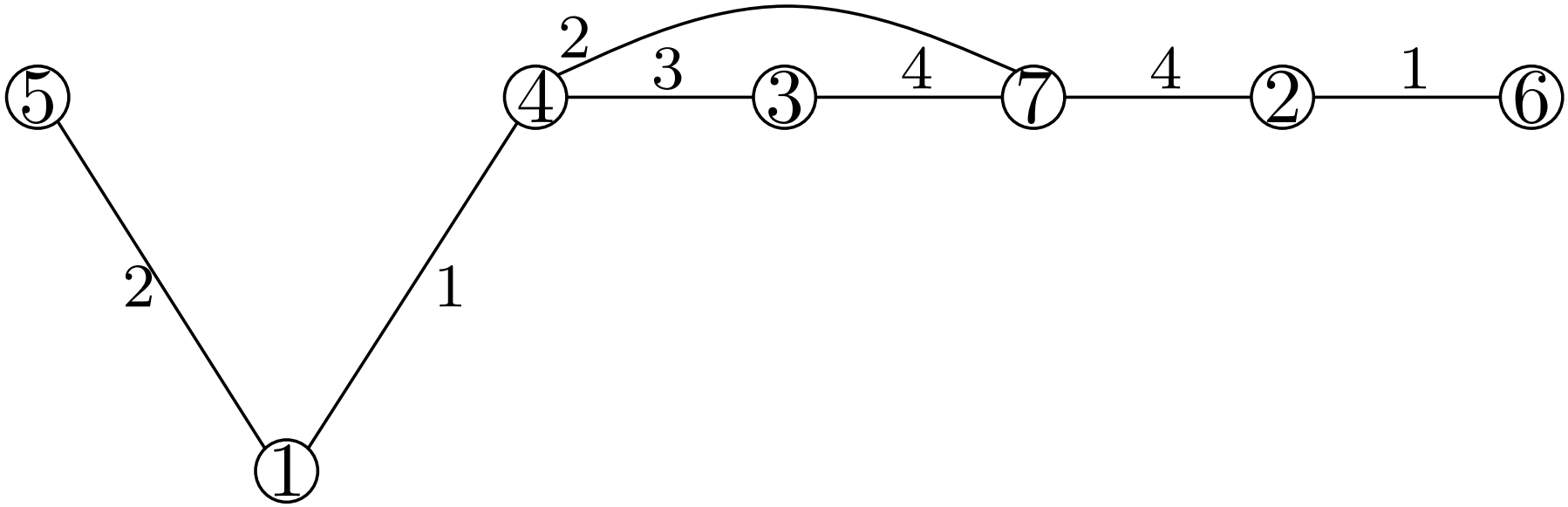
- ordne Knoten nach Wichtigkeit



Wiederholung: CH

preprocessing:

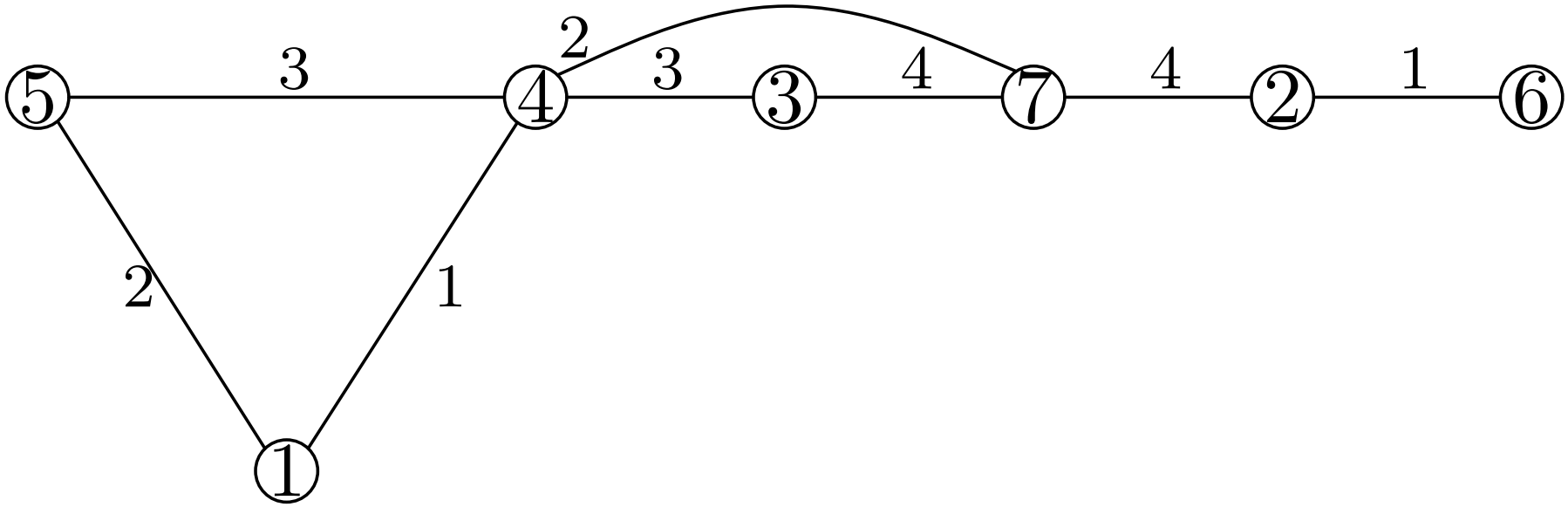
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

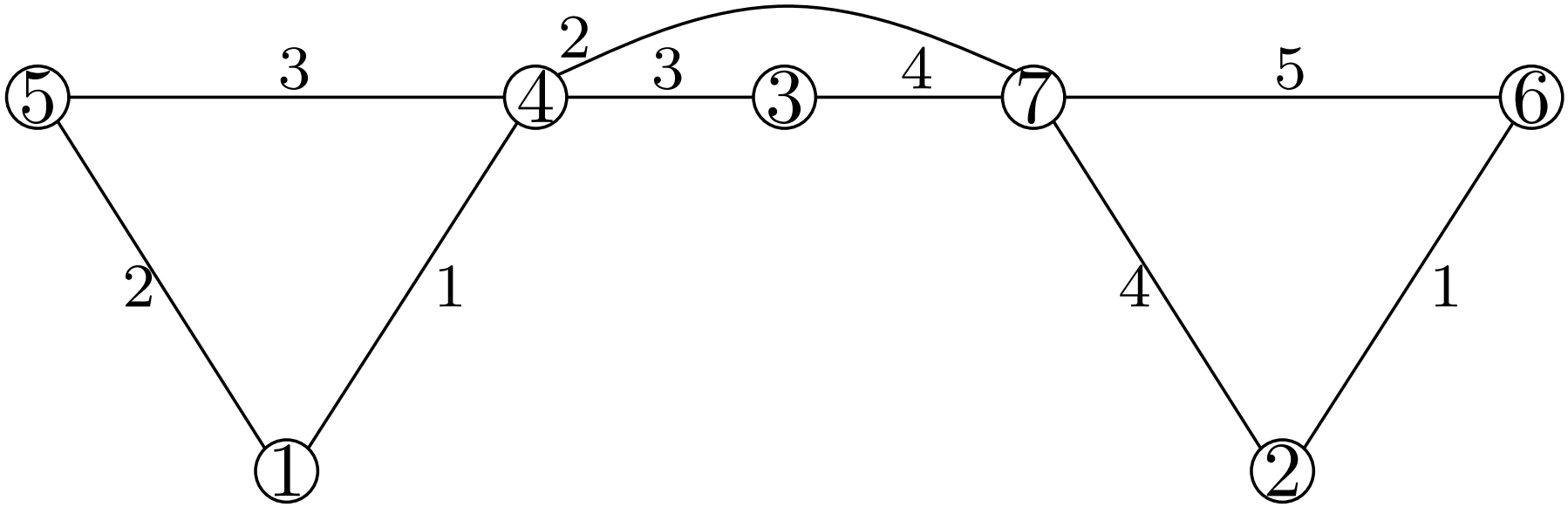
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

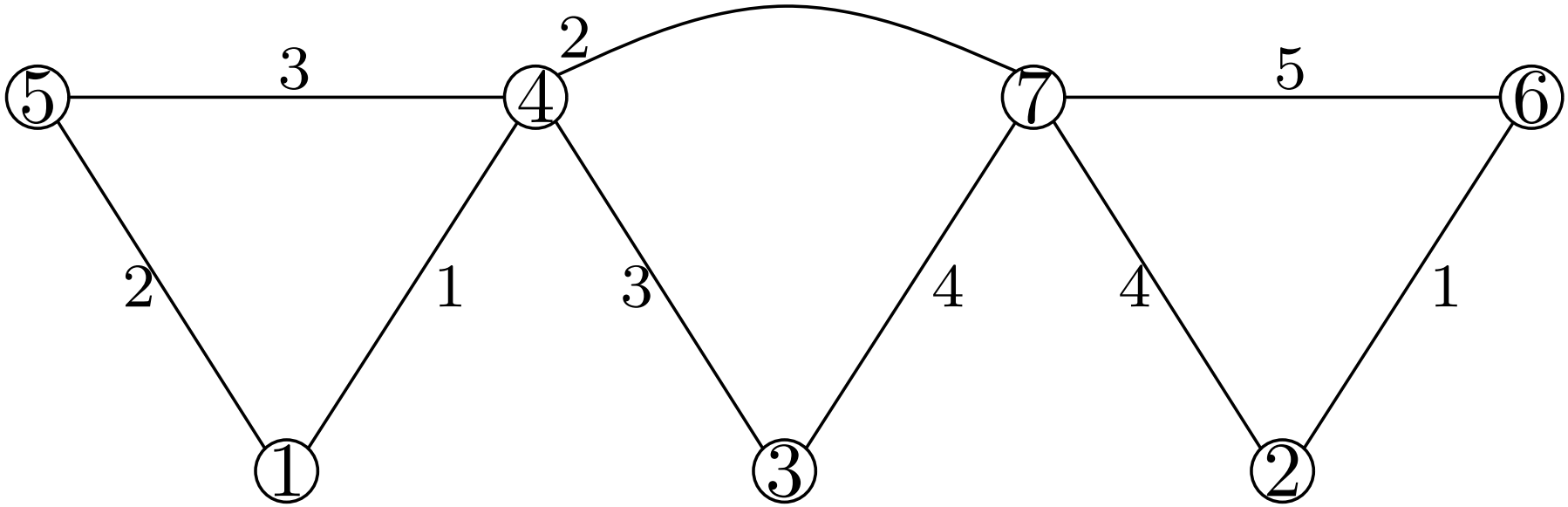
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

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

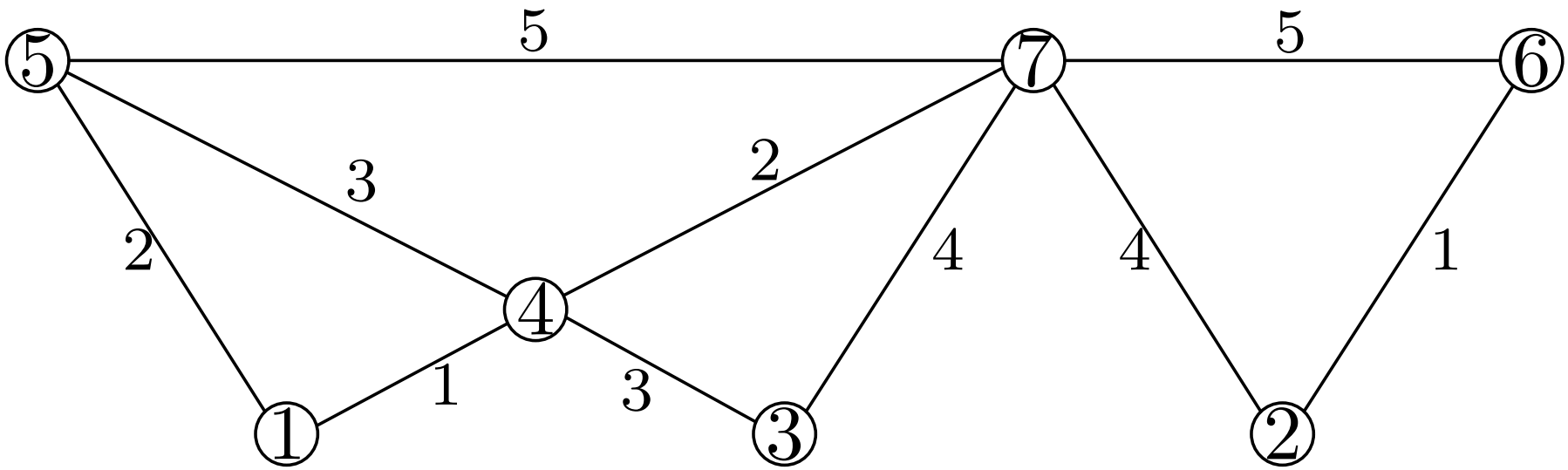


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Wiederholung: CH

preprocessing:

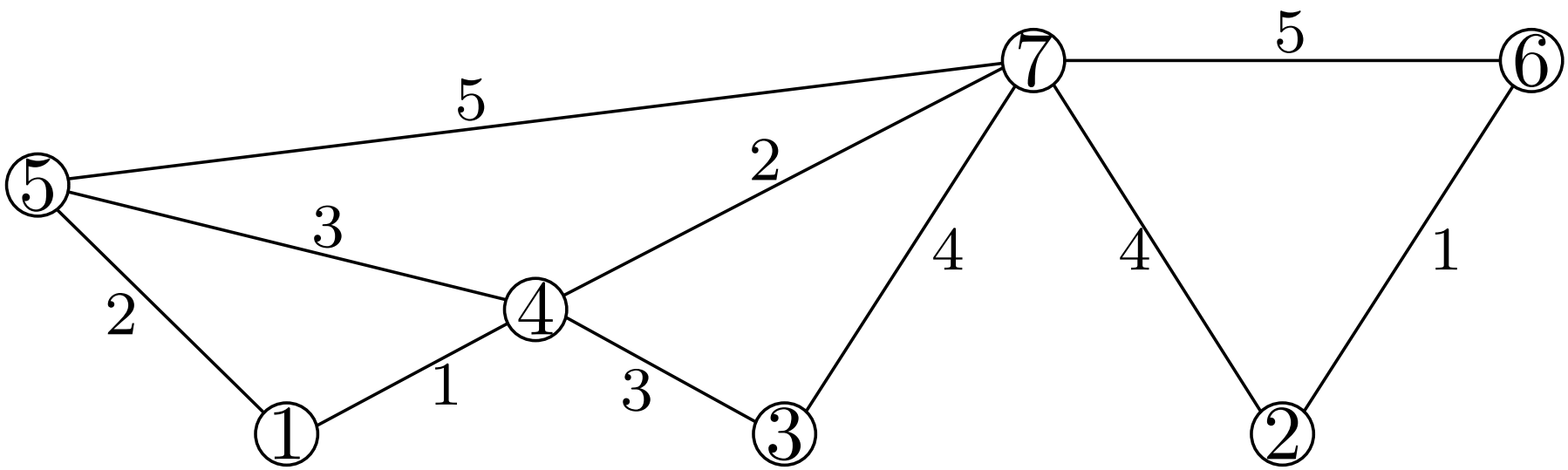
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

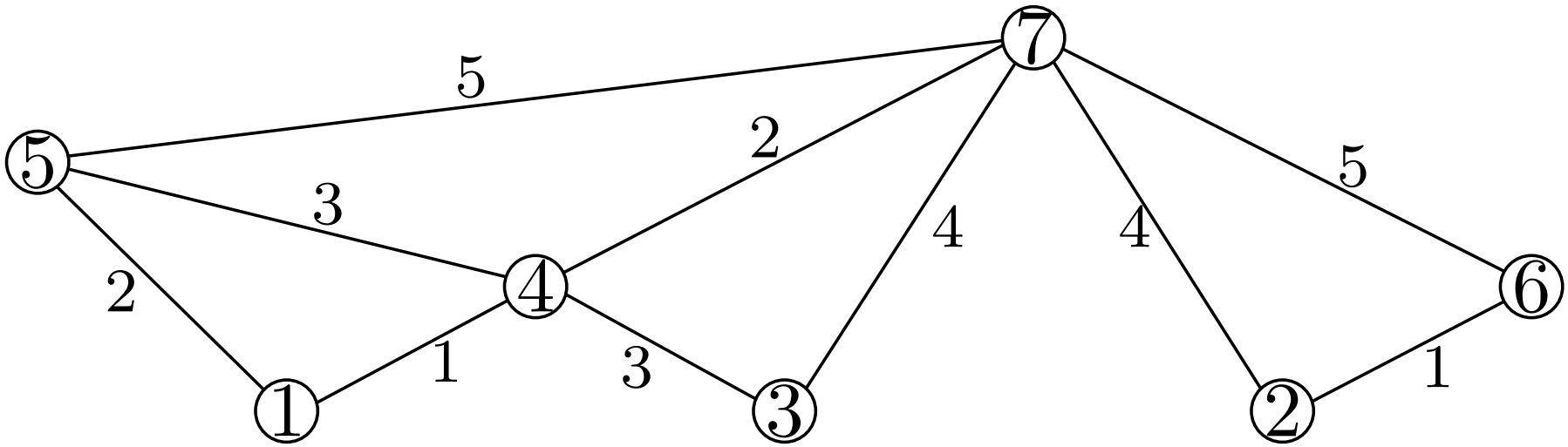
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

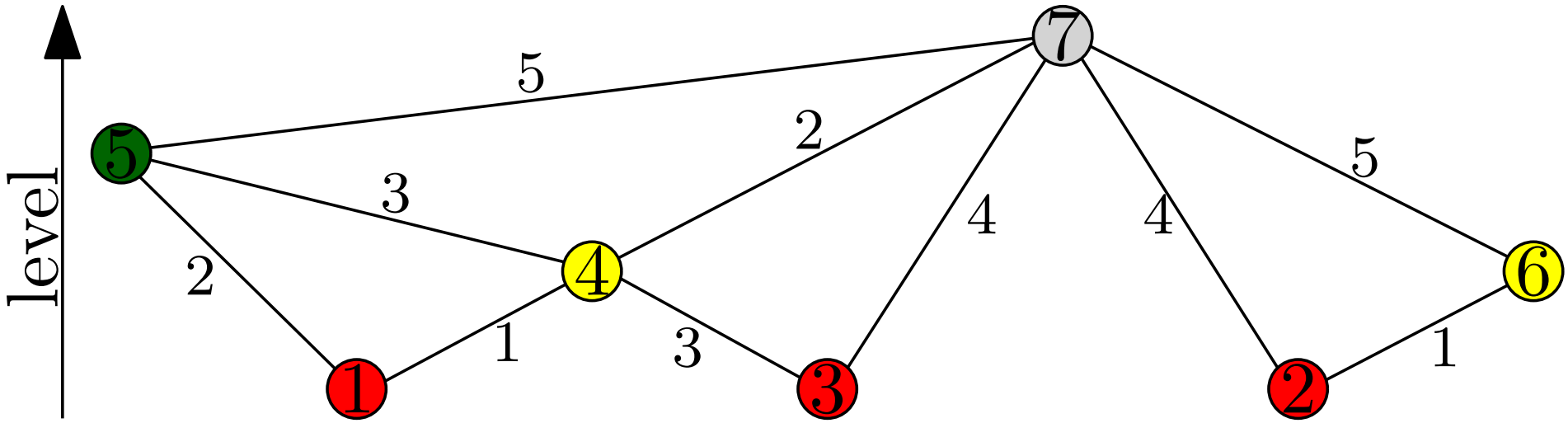
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu



Wiederholung: CH

preprocessing:

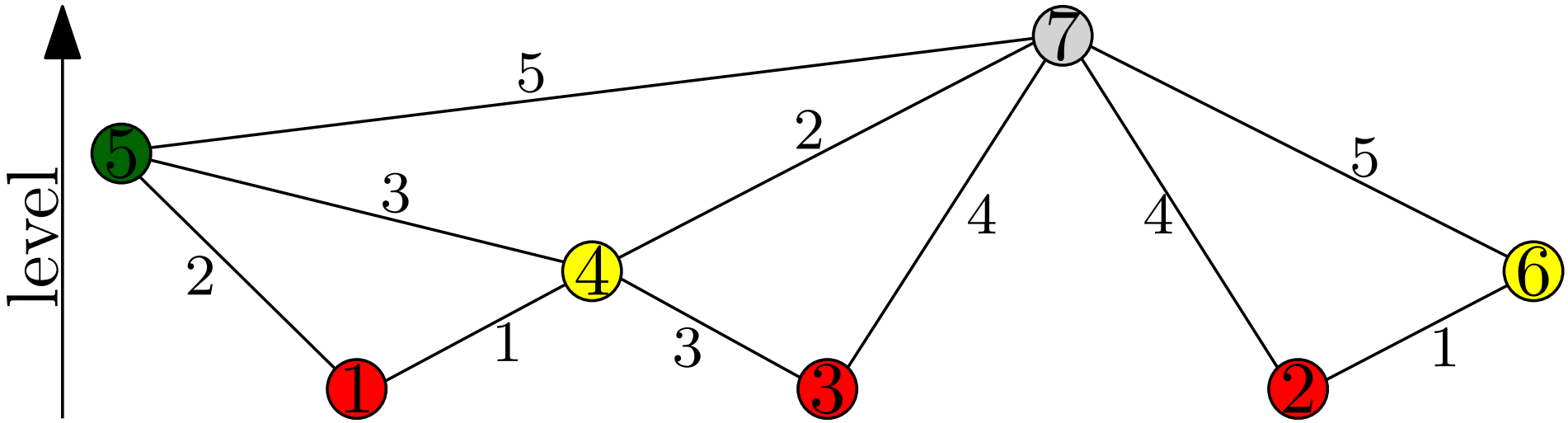
- ordne Knoten nach Wichtigkeit
- bearbeite in der Reihenfolge
- füge Shortcuts hinzu
- Levelzuordnung



Wiederholung: CH

Punkt-zu-Punkt Anfragen

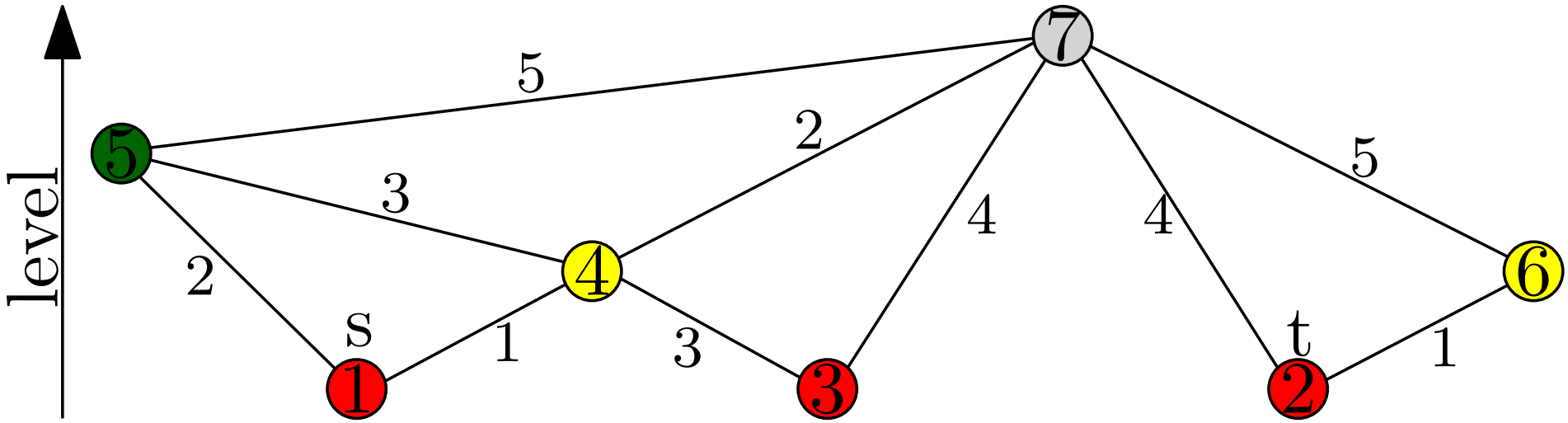
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

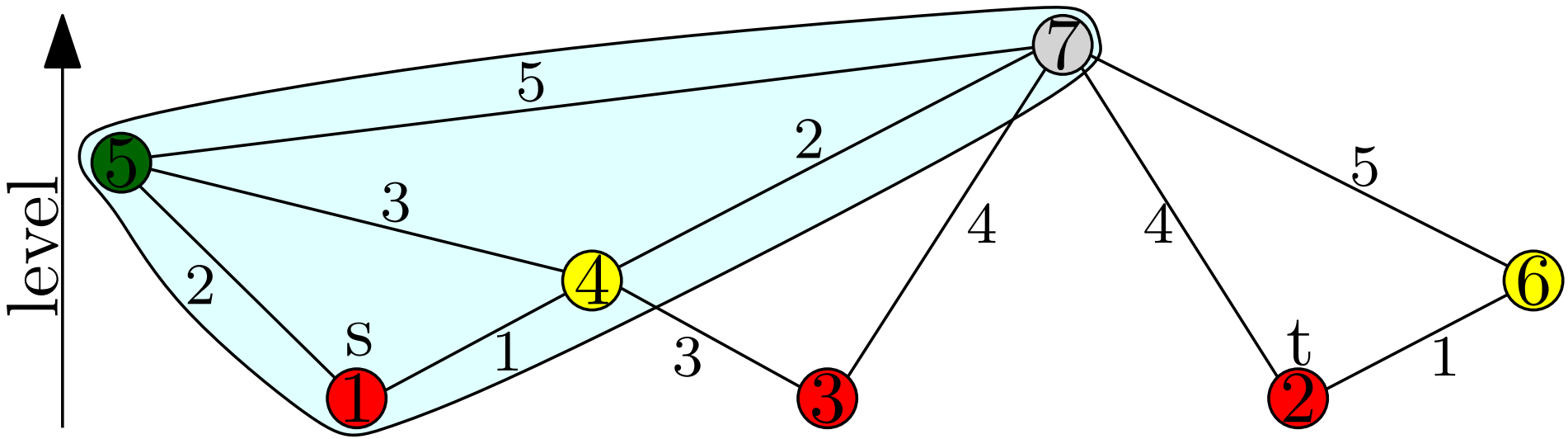
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

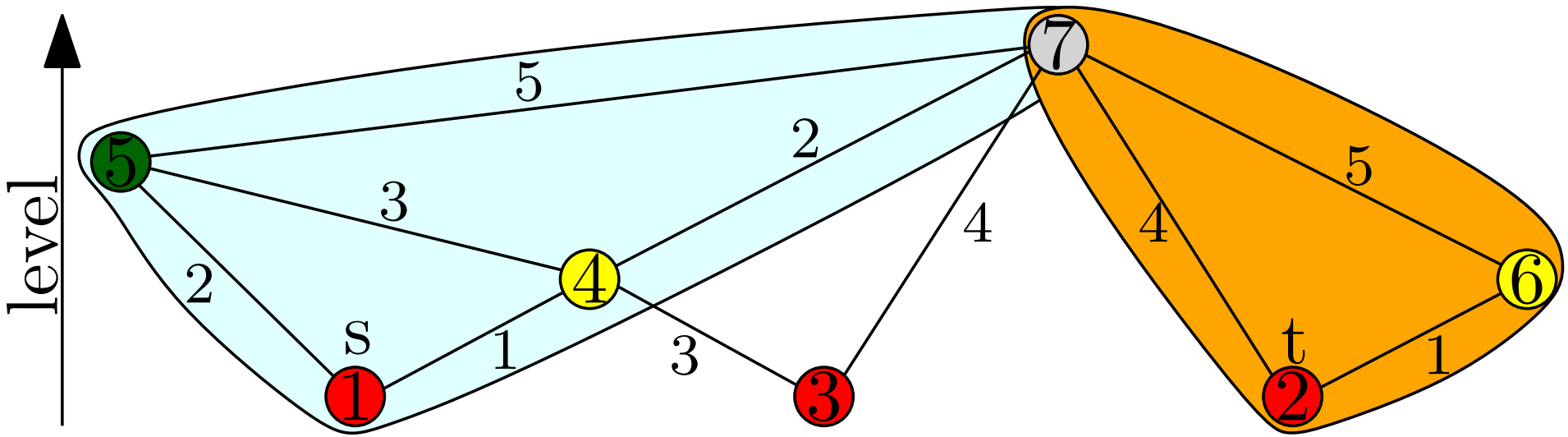
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

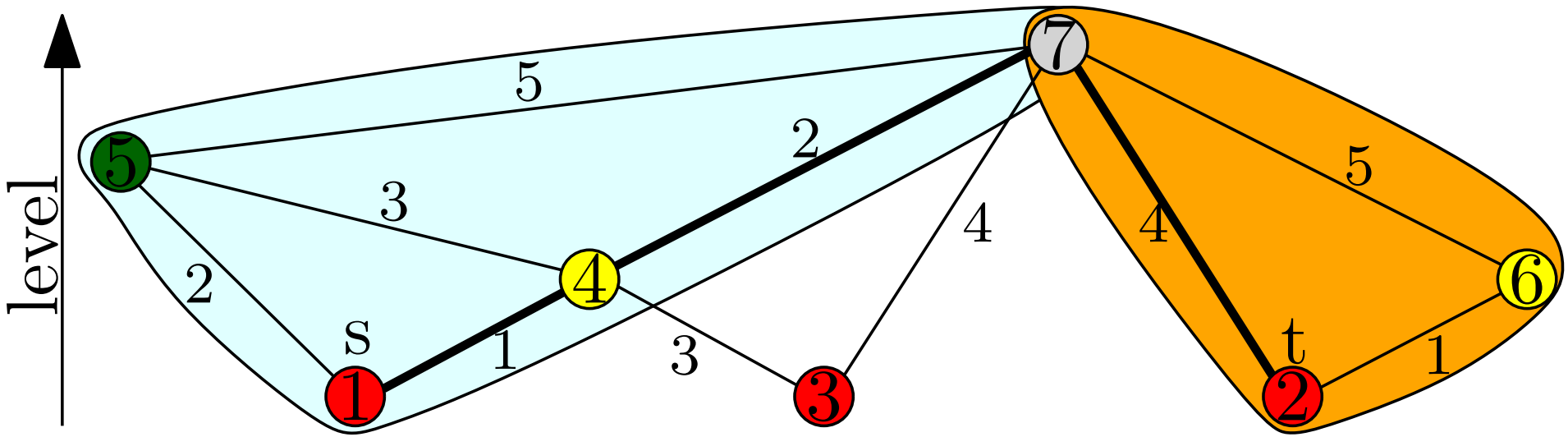
- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



Wiederholung: CH

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten



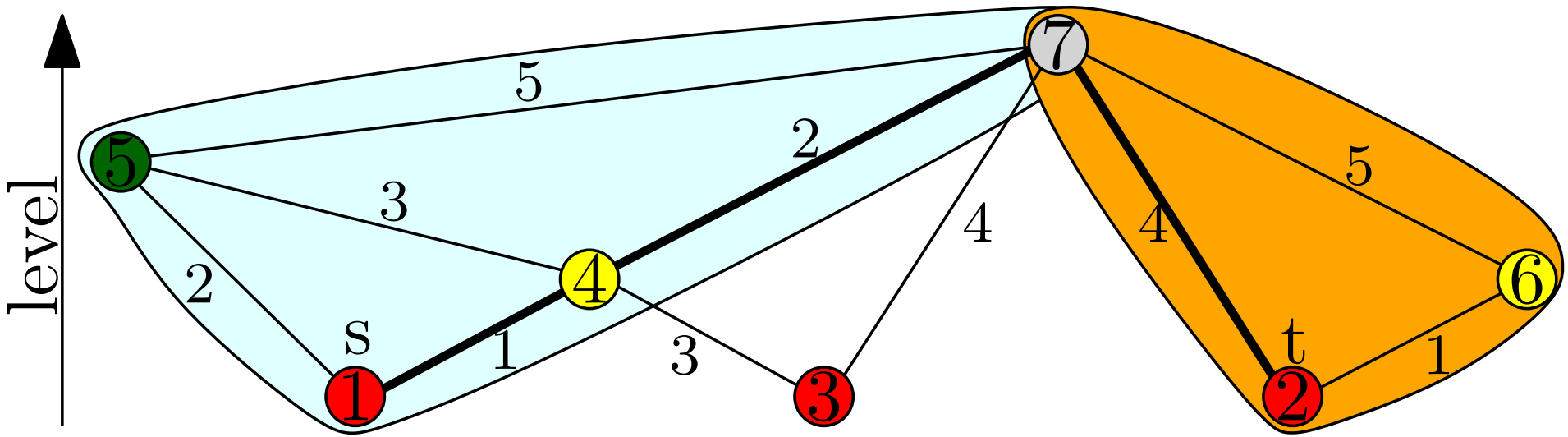
Wiederholung: CH

Punkt-zu-Punkt Anfragen

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten

Korrektheit:

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



HubLabels

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Hublabels

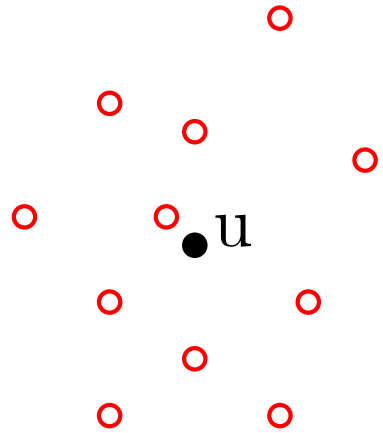
Vorberechung:

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

Hublabels

Vorberechung:

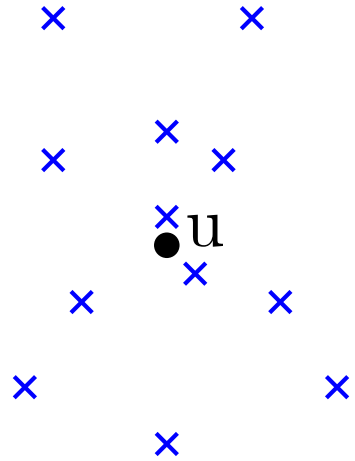
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$



Hublabels

Vorberechung:

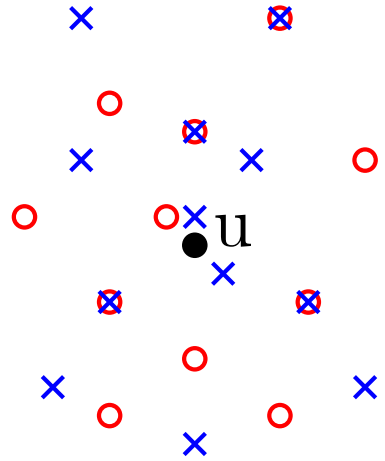
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$



Hublabels

Vorberechung:

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



Hublabels

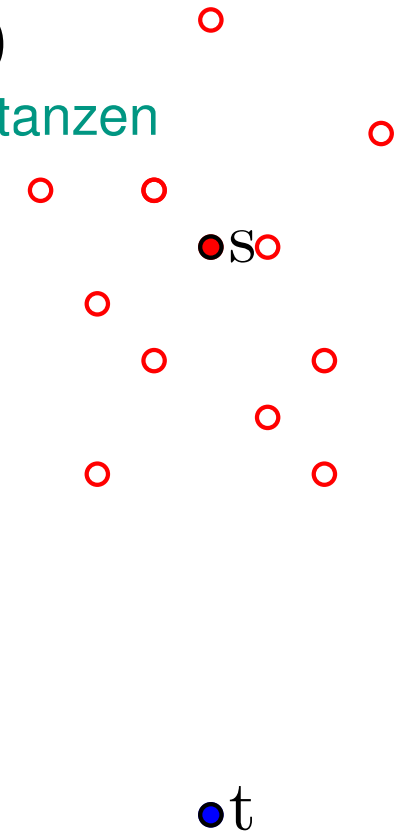
Vorberechung:

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

Hublabels

Vorberechung:

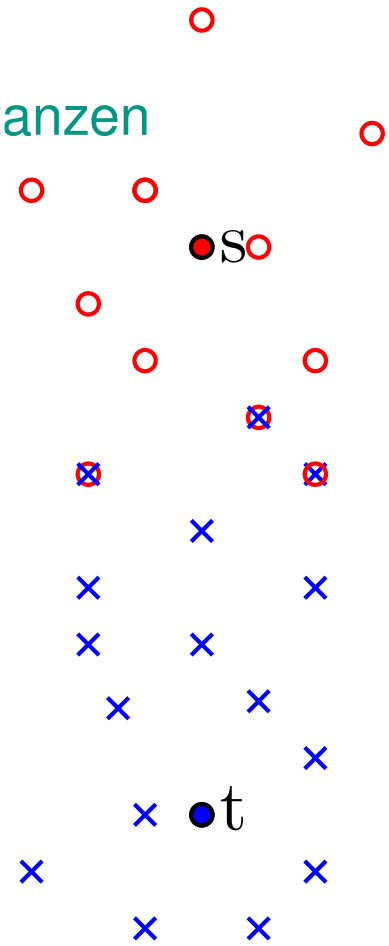
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt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



Hublabels

Vorberechung:

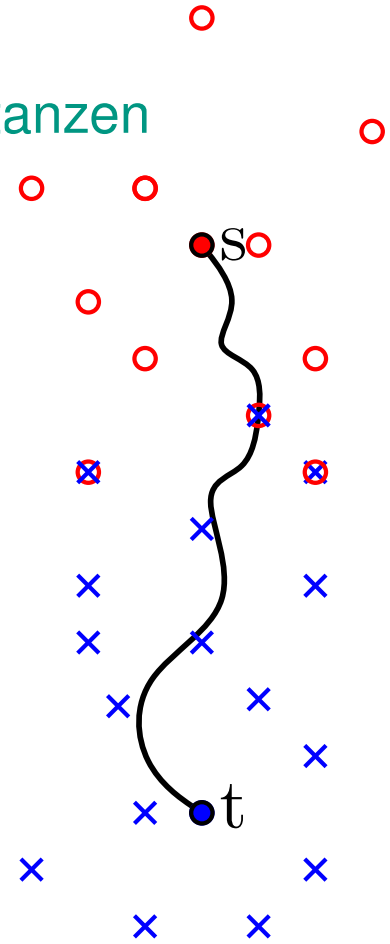
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt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



Hublabels

Vorberechung:

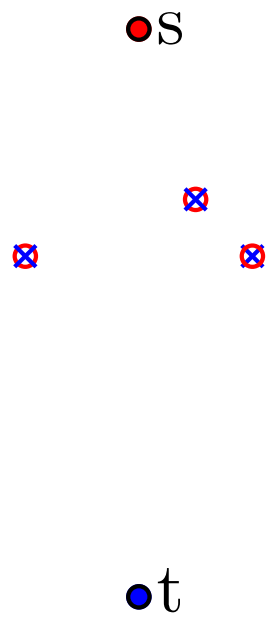
- für jeden Knoten u, berechne zwei Label $L_f(u), L_b(u)$
- ein Label ist eine Menge von Knoten (Hubs) und Distanzen
  - dist(u, v) für jeden Hub $v \in L_f(u)$
  - dist(v, u) für jeden Hub $v \in L_b(u)$
- die Label müssen die cover property einhalten:
  $\forall s, t$, $L_f(s) \cap L_b(t)$ überdeckt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



Hublabels

Vorberechung:

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



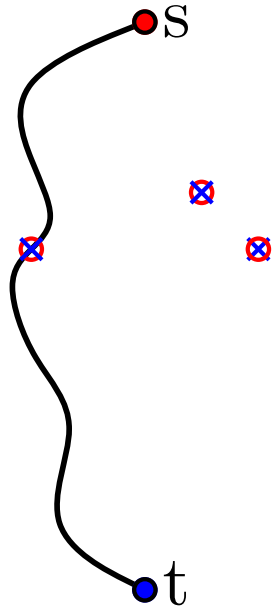
s–t Anfrage:

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

Hublabels

Vorberechung:

- 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



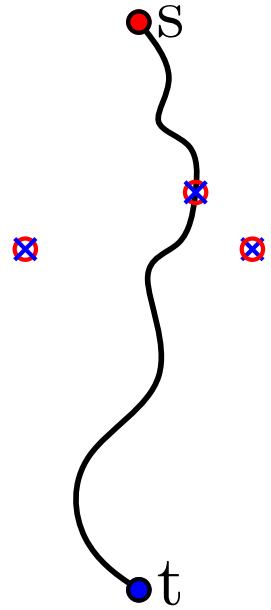
s–t Anfrage:

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

Hublabels

Vorberechung:

- 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



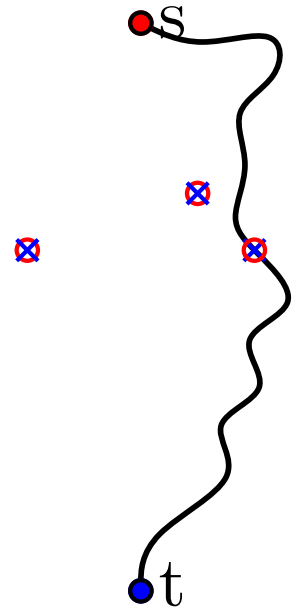
s–t Anfrage:

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

Hublabels

Vorberechung:

- 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



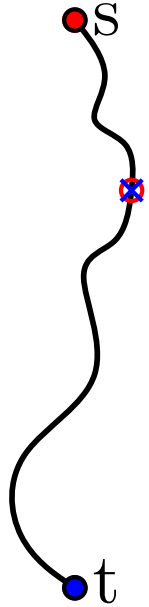
s–t Anfrage:

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

Hublabels

Vorberechung:

- 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



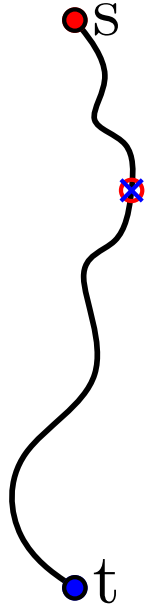
s–t Anfrage:

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

Hublabels

Vorberechung:

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

Beobachtungen:

- Laufzeit hängt von Labelgrösse ab
- wie effizient berechnen?

Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

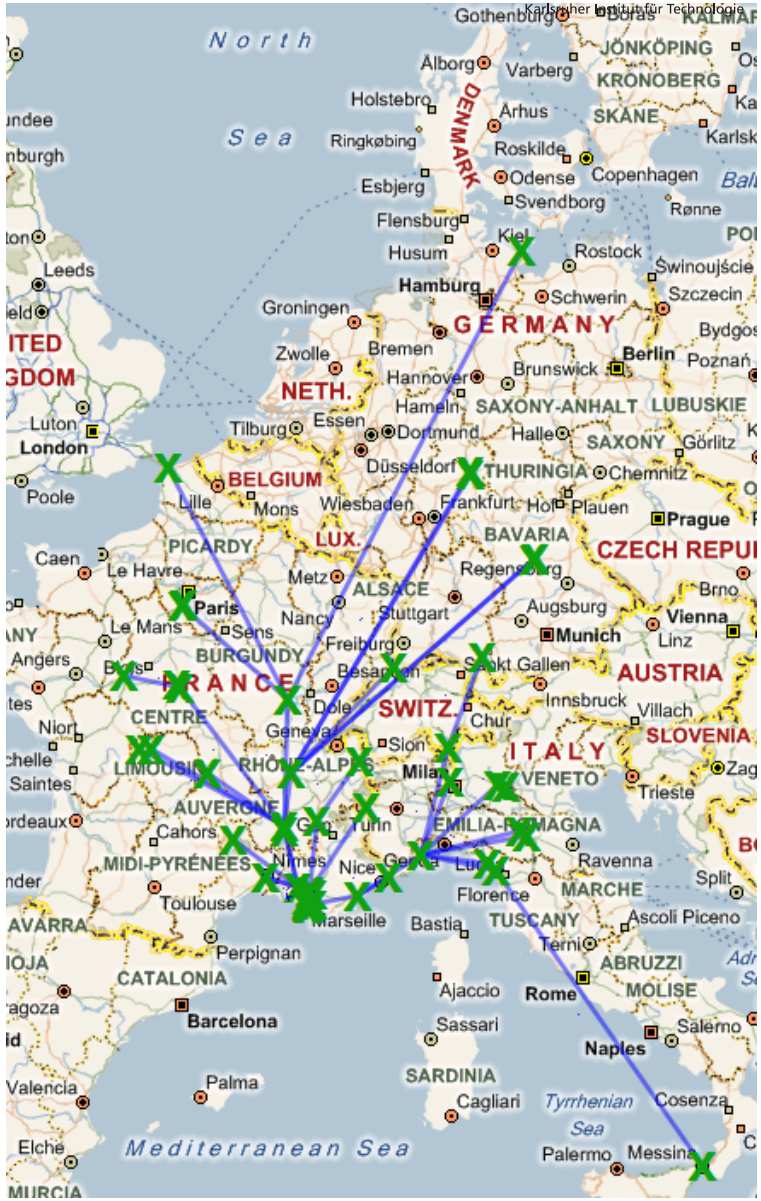
Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

$L_f(s)$

1,0	4,1	5,2	7,3



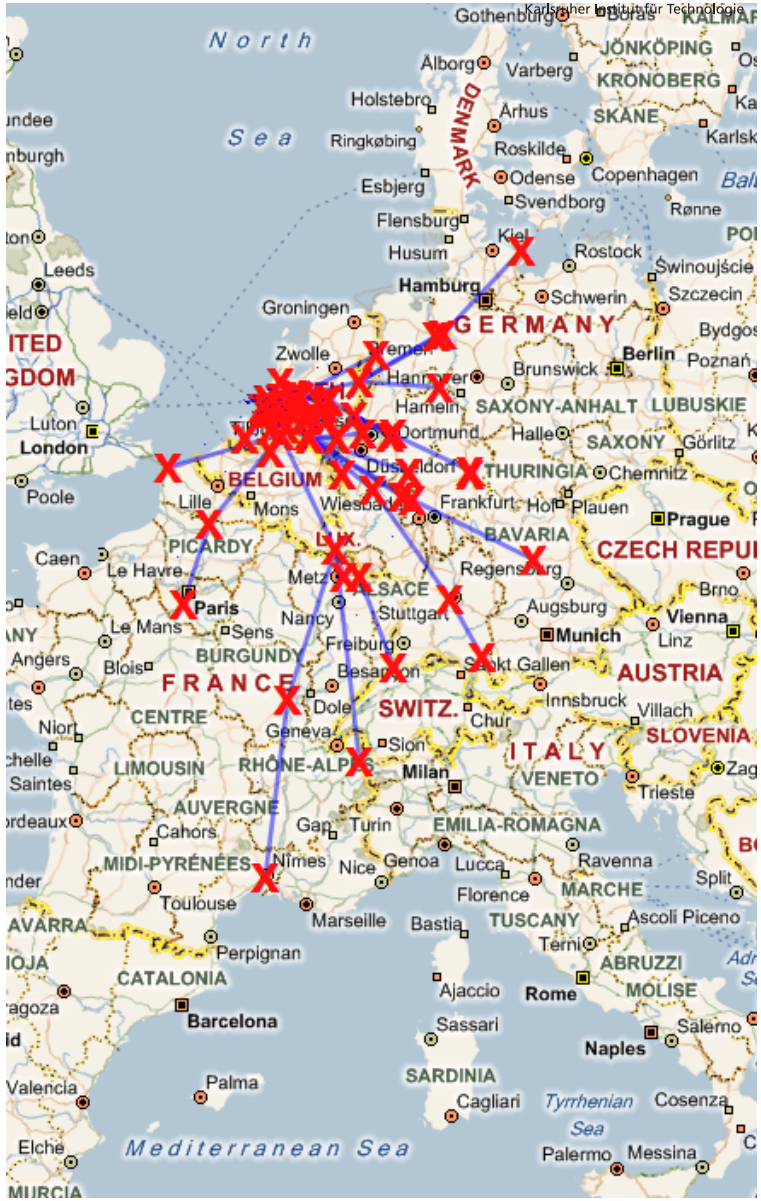
Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

$L_f(s)$ | 1,0 | 4,1 | 5,2 | 7,3 |

$L_b(t)$ | 2,0 | 6,1 | 7,4 | 8,1 | 9,3 |



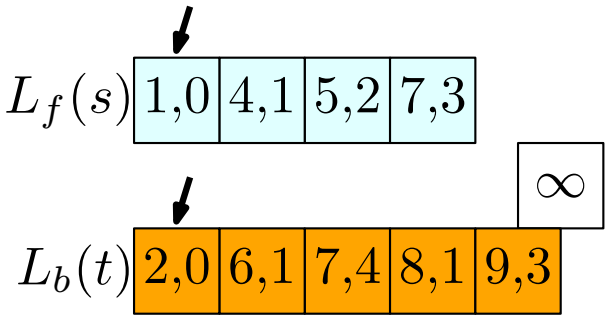
Hublabels

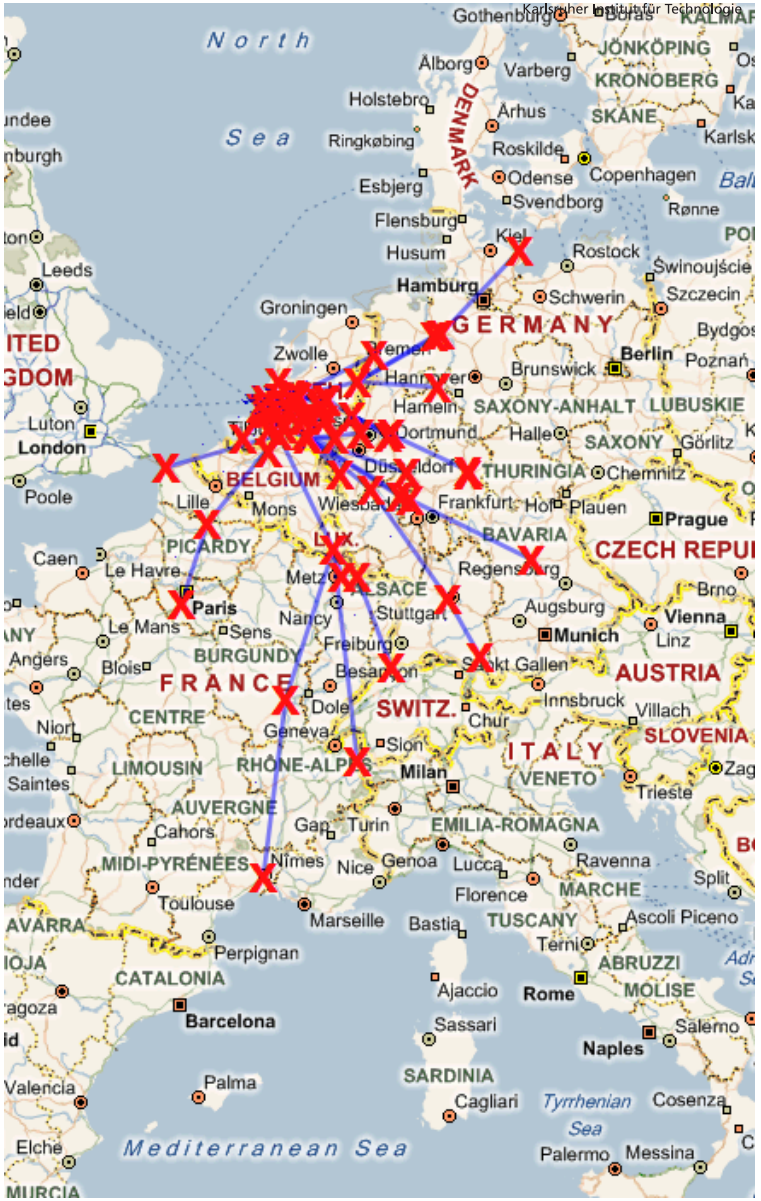
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

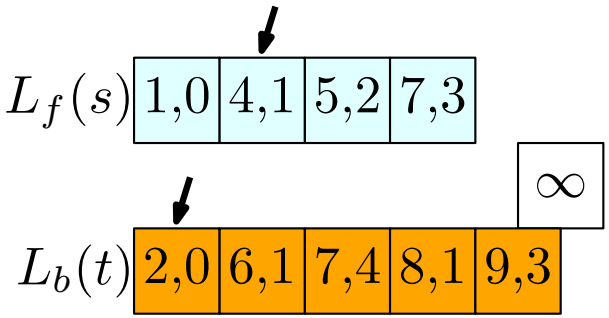
Hublabels

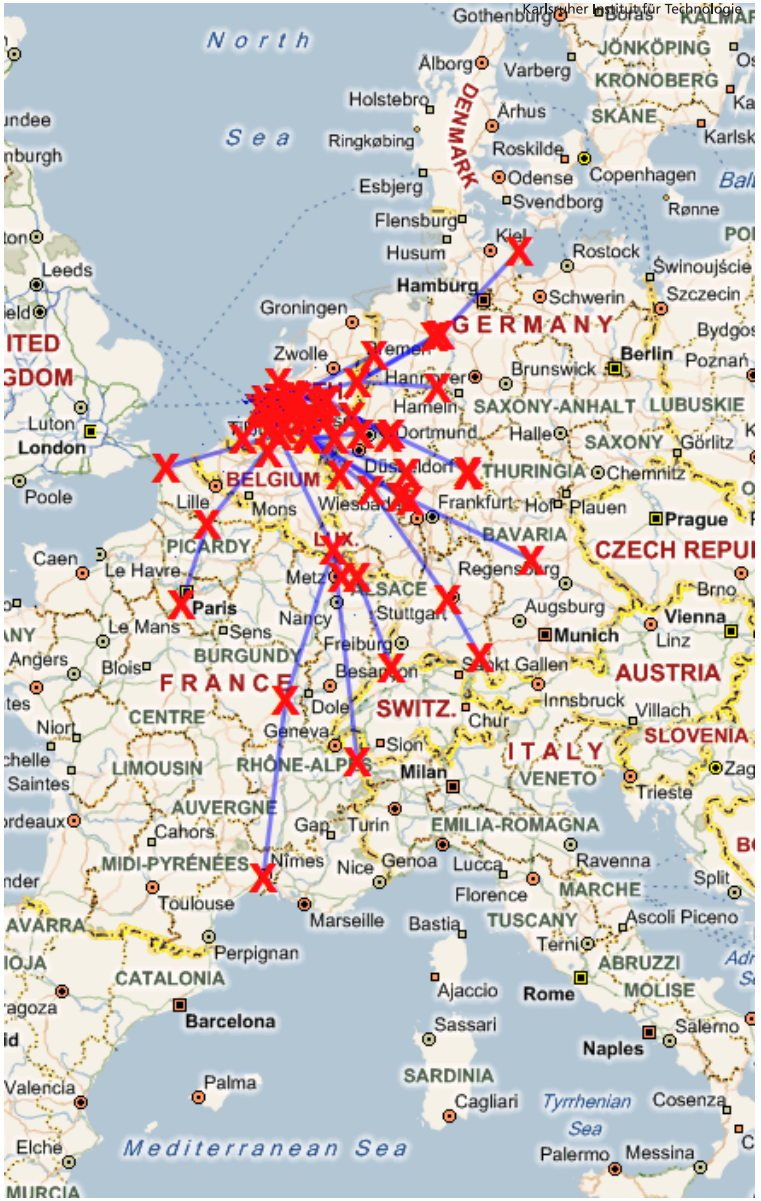
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

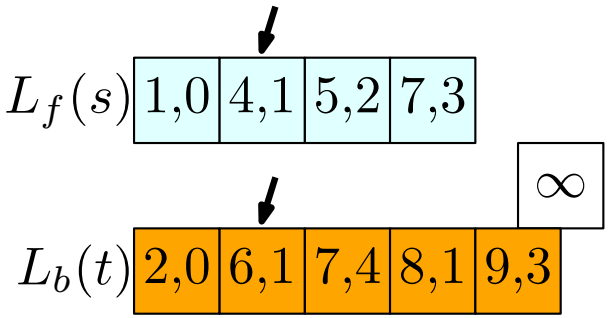
Hublabels

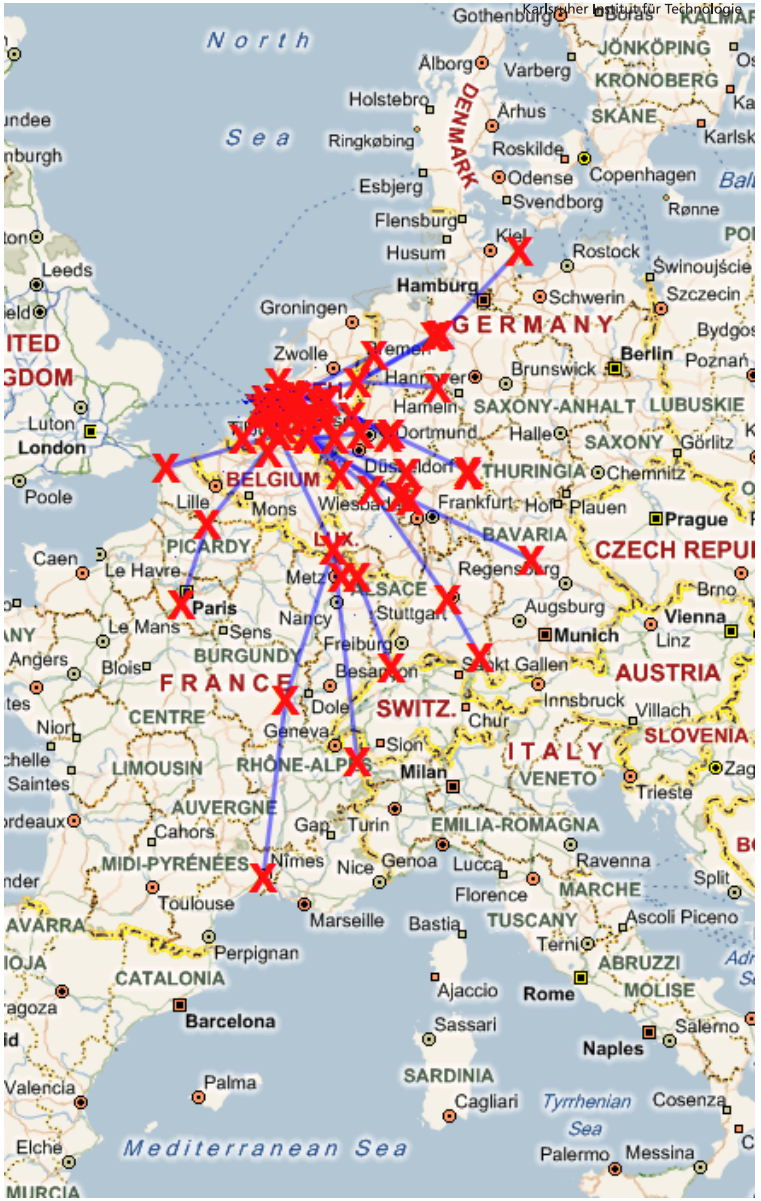
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

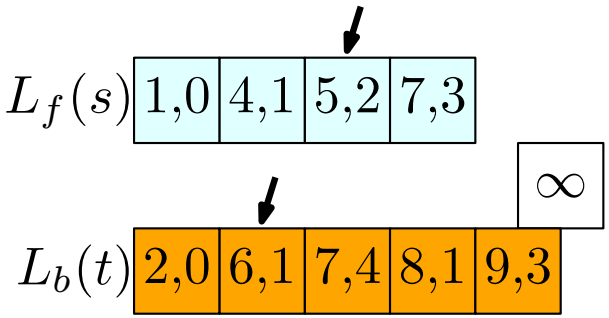
Hublabels

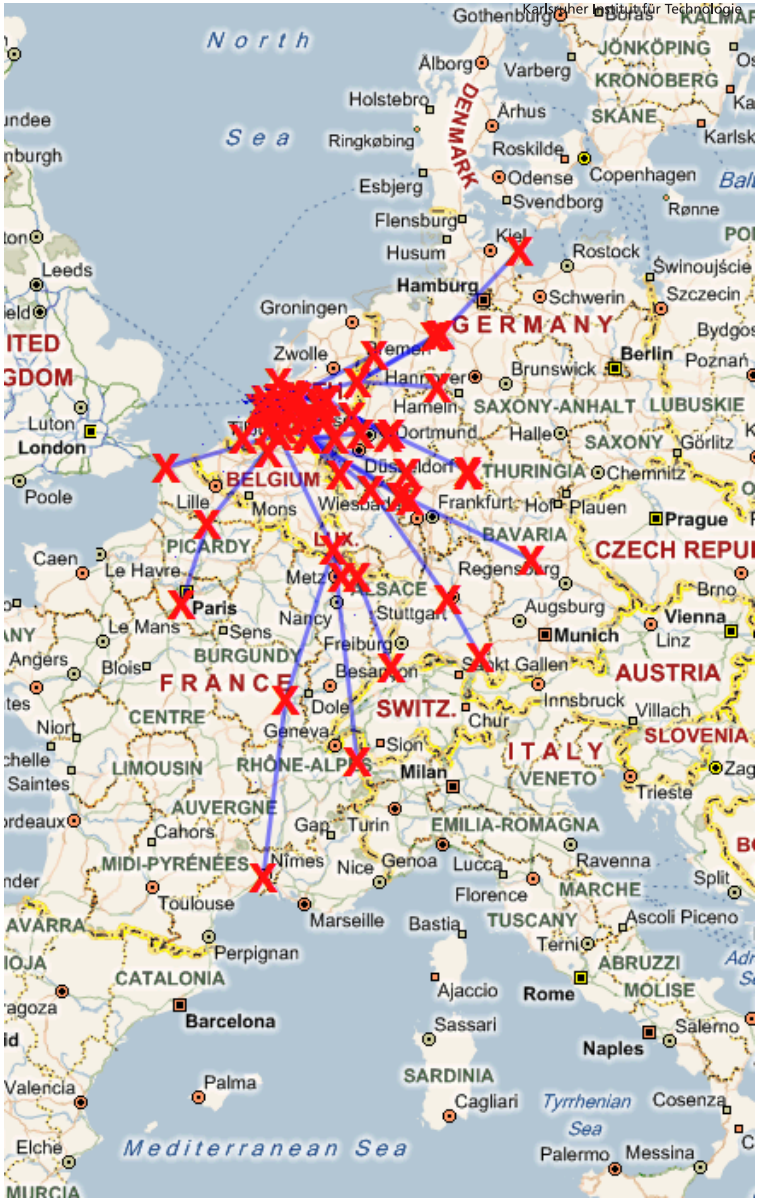
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

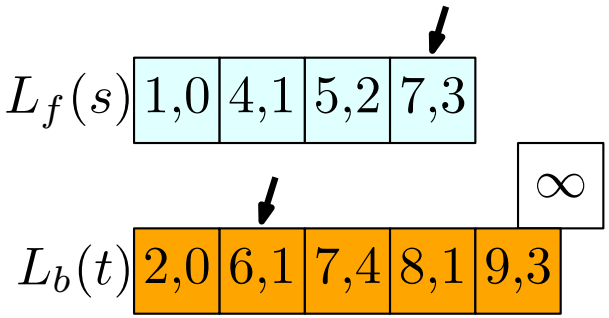
Hublabels

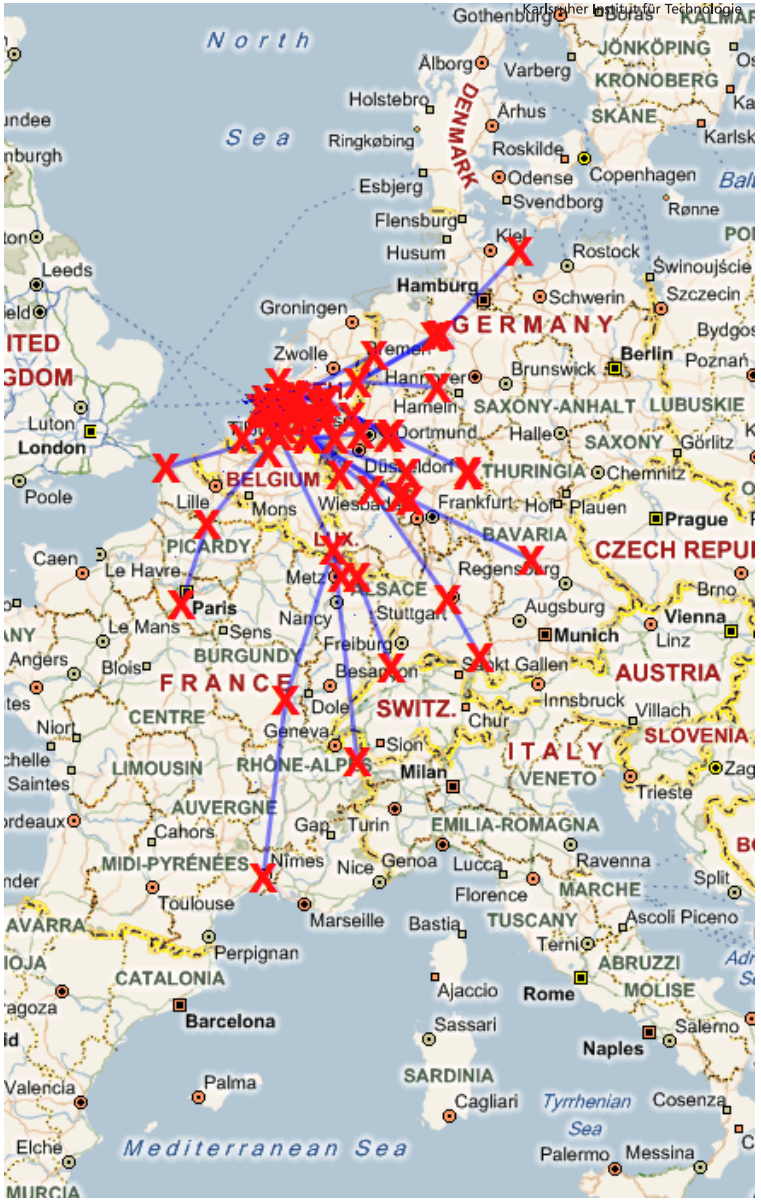
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





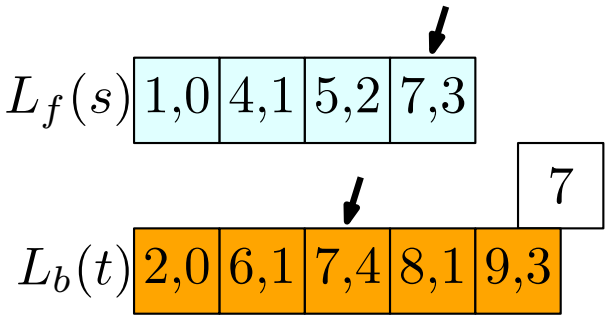
Hublabels

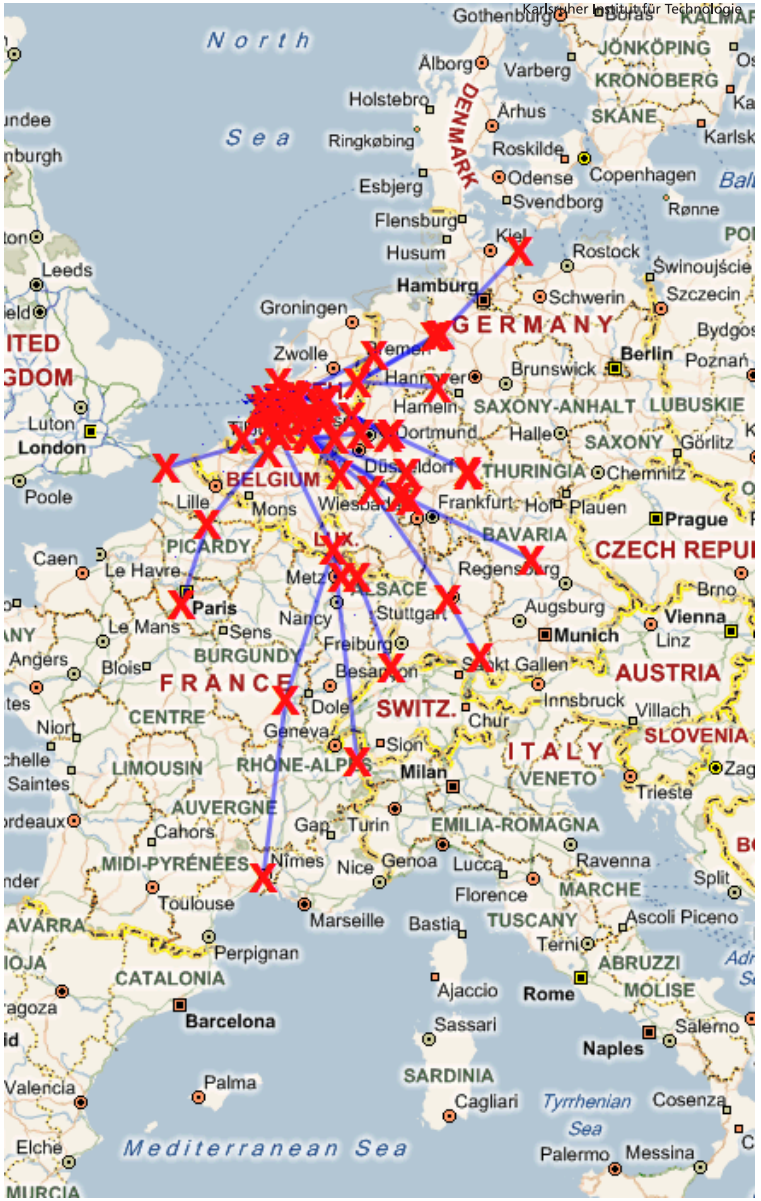
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Institut für Theoretische Informatik
Lehrstuhl Algorithmik

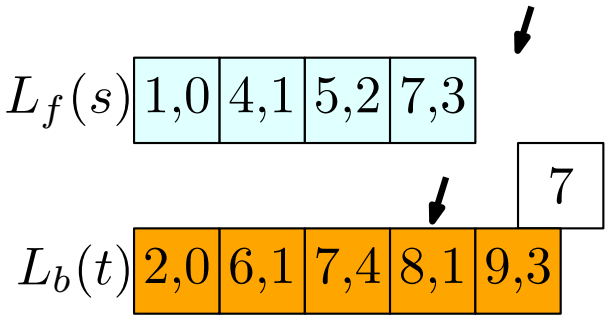
Hublabels

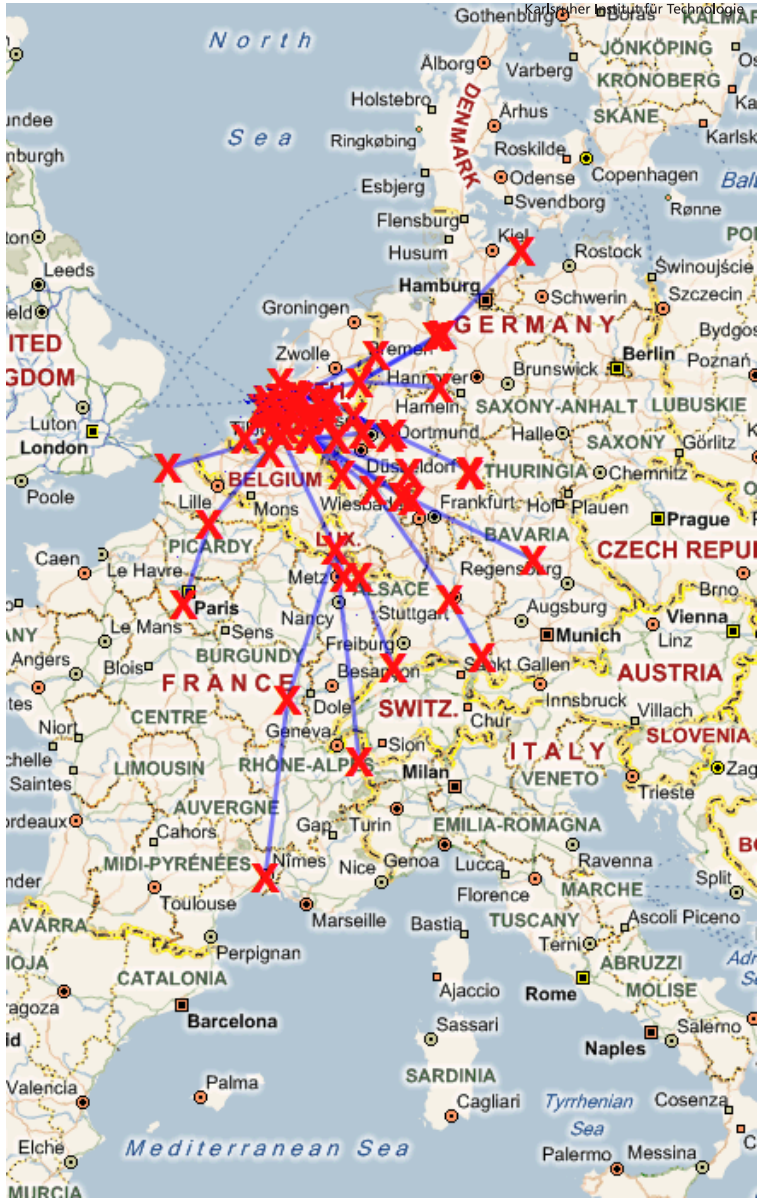
Speichern der Labels:

- als Menge von Hub,Distanz Paaren

Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig





Hublabels

Speichern der Labels:

- als Menge von Hub,Distanz Paaren

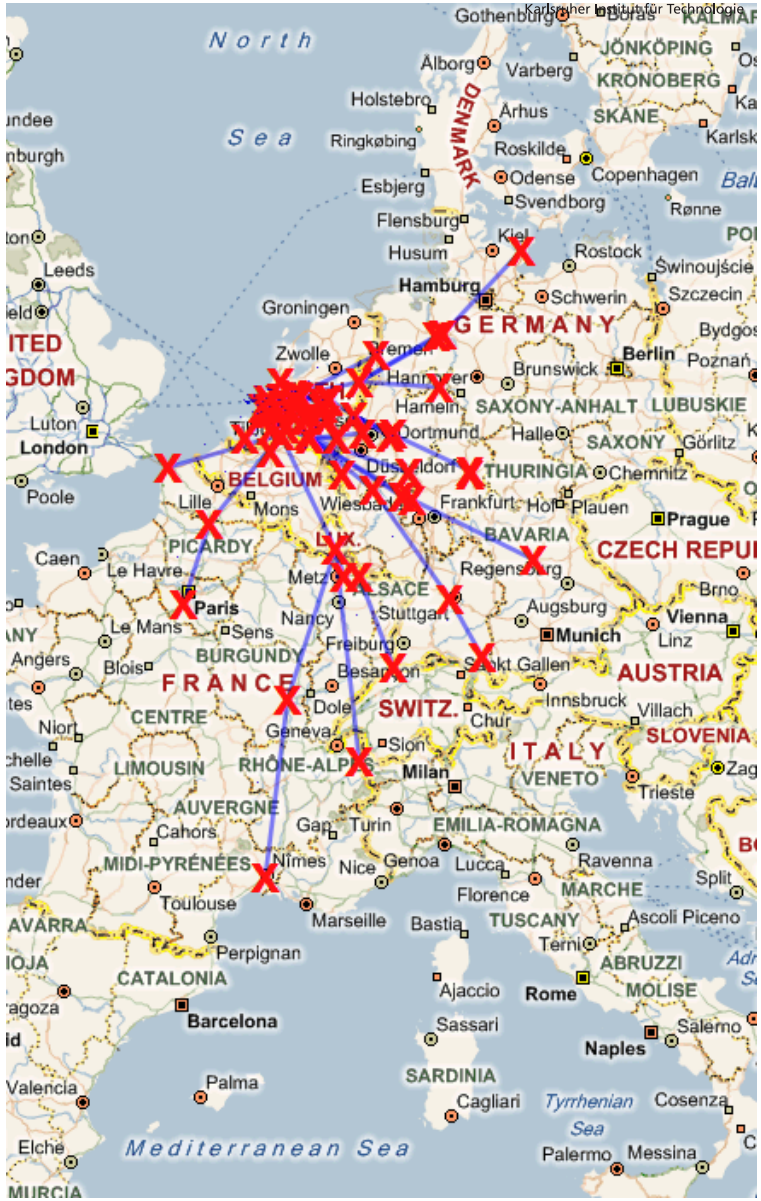
Anfrage:

- scannen von zwei Arrays
- nur einige Speicherzugriffe nötig
- sehr hohe Lokalität

$L_f(s)$ | 1,0 | 4,1 | 5,2 | 7,3 |

7

$L_b(t)$ | 2,0 | 6,1 | 7,4 | 8,1 | 9,3 |



Hublabels

Komplexität:

- Maximale Labellänge soll klein sein
- Optimale Hublabels zu berechnen ist NP-schwer [BGK+15]
- Es gibt $O(\log n)$-Approximation [GPPR04]
  - Ursprüngliche Laufzeit in $O(n^5)$
  - Wurde auf $O(n^3 \log n)$ verbessert [DGSW14]

Hierarchische Hublabels

Hierarchische Hublabels

- Jedes Labeling definiert eine Relation $\preceq$ auf den Labels wie folgt:

$$u \preceq v \iff u \in L_f(v) \cup L_b(v)$$

- Ein Labeling ist hierarchisch wenn $\preceq$ eine partielle Ordnung ist.
- Optimale Hierarchische Hublabels zu berechnen ist NP-schwer [BGK+15]

Kanonische Hublabels

- Ein kanonische Labeling bezüglich einer Knotenordnung O ist
  - hierarchisch
  - $\preceq$ muss mit O konsistent sein
  - aus keinem Label kann man ein Hub löschen
- Das kanonische Labeling ist eindeutig für eine feste Ordnung O

Verbindung zu CHs

- $\preceq$ ordnet die Knoten nach “Wichtigkeit” wie bei der CH
- CH Suchräume sind gültige hierarchisch Labels.
- aber sie sind größer als nötig (siehe stall-on-demand)
- und in der Regel sind sie nicht kanonisch
- $\rightarrow$ Überflüssige Knoten filtern

Verbindung zu CHs

- $\preceq$ ordnet die Knoten nach “Wichtigkeit” wie bei der CH
- CH Suchräume sind gültige hierarchisch Labels.
- aber sie sind größer als nötig (siehe stall-on-demand)
- und in der Regel sind sie nicht kanonisch
- $\rightarrow$ Überflüssige Knoten filtern

- Im folgenden betrachten wir nur noch hierarchisch Hublabels
- Für Beweise nehmen wir ferner an, dass kürzeste Wege eindeutig sind und Graphen ungerichtet sind

Labels Kanonisch Machen

- Sei $m(s,t)$ der Knoten mit höchsten Rank auf dem kürzesten st-Pfad
- $m(s,t)$ ist der gemeinsame Hub von s und t über den der kürzeste Pfad geht.

Satz

Wir können einen Hub h aus dem Label $L(v)$ von v löschen genau dann wenn $h \neq m(v,h)$.

Labels Kanonisch Machen

- Sei $m(s,t)$ der Knoten mit höchsten Rank auf dem kürzesten st-Pfad
- $m(s,t)$ ist der gemeinsame Hub von s und t über den der kürzeste Pfad geht.

Satz

Wir können einen Hub h aus dem Label $L(v)$ von v löschen genau dann wenn $h \neq m(v,h)$.

- Zwei Richtungen:
- Wenn $h = m(v,h)$ dann dürfen wir h nicht aus $L(v)$ löschen.
- Wenn $h \neq m(v,h)$ dann dürfen wir h aus $L(v)$ löschen.

Labels Kanonisch Machen

Übersicht:

- Erste Richtung: Wenn $h = m(v, h)$ dann dürfen wir h nicht aus $L(v)$ löschen.
- Wir müssen zeigen, dass es eine Anfrage gibt die nach der Herausnahme von h aus dem Label v inkorrekt wird.
- Wir zeigen, dass wenn h löschen, dann wird die vh-Anfrage falsch beantwortet

Beweis:

- Der gemeinsame Hub von h und v darf nicht niedriger sein als h und v
  (folgt direkt aus der Definition von kanoischem Labeling)
- Der höchste Knoten auf dem kürzesten vh-Pfad ist h (Voraussetzung)
- Da ferner jeder Knoten eine eindeutige Position hat können sich $L(v)$ und $L(h)$ nur in h schneiden
- $\Longrightarrow$ h darf nicht gelöscht werden

Labels Kanonisch Machen

Übersicht:

- Zweite Richtung: Wenn $h \neq m(v, h)$ dann dürfen wir h aus $L(v)$ löschen
- Wir müssen zeigen, dass alle Anfragen nach der Herausnahme von h aus dem Label v noch korrekt sind

Beweis:

- $L(v)$ wird nur bei vt- oder sv-Anfragen angeschaut, nur diese können also inkorrekt werden
  $\rightarrow$ Betrachte ohne Beschränkung der Allgemeinheit vt-Anfragen
- Eine vt-Anfrage kann nur inkorrekt werden, wenn h auf dem kürzesten vt-Pfad liegt
- Es reicht also zu zeigen, dass alle vt-Anfragen die durch h gehen korrekt sind
- **Ziel:** Wir zeigen, dass diese vt-Anfragen sich nicht nur in h sondern auch in $m(v, h)$ oder in $m(h, t)$ treffen

Labels Kanonisch Machen

Übersicht:

- Zweite Richtung: Wenn $h \neq m(v,h)$ dann dürfen wir h aus $L(v)$ löschen
- Wir müssen zeigen, dass alle Anfragen nach der Herausnahme von h aus dem Label v noch korrekt sind

Beweis:

- **Ziel:** Wir zeigen, dass diese vt-Anfragen sich nicht nur in h sondern auch in $m(v,h)$ oder in $m(h,t)$ treffen
- Fall 1:
  - $m(v,h)$ höher als in $m(h,t)$
  - $m(v,h)$ höchster Knoten auf vt-Pfad
  - Nach Argument von letzter Folie: $m(v,h) \in L(v)$ und $m(v,h) \in L(v)$
  - vt-Anfrage trifft sich nicht nur in h sondern auch in $m(v,h)$
  - $\rightarrow$ wir können h löschen

Labels Kanonisch Machen

Übersicht:

- Zweite Richtung: Wenn $h \neq m(v, h)$ dann dürfen wir h aus $L(v)$ löschen
- Wir müssen zeigen, dass alle Anfragen nach der Herausnahme von h aus dem Label v noch korrekt sind

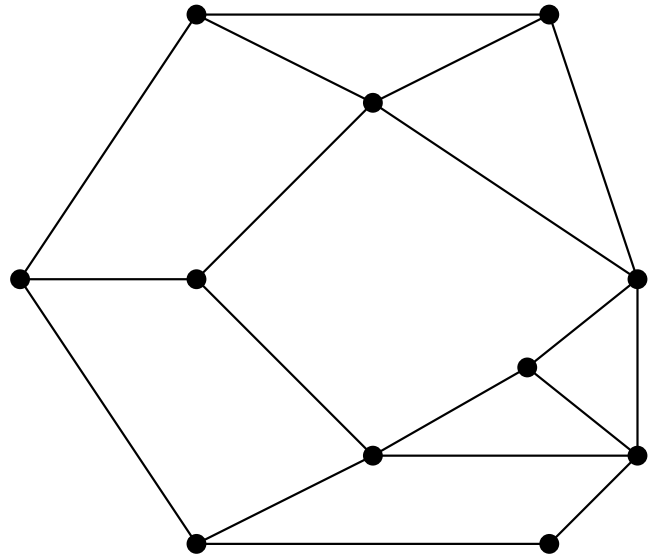
Beweis:

- **Ziel:** Wir zeigen, dass diese vt-Anfragen sich nicht nur in h sondern auch in $m(v, h)$ oder in $m(h, t)$ treffen
- Fall 2:
  - $m(h, t)$ höher als in $m(v, h)$
  - $m(h, t)$ höchster Knoten auf vt-Pfad
  - Nach Argument von letzter Folie: $m(h, t) \in L(v)$ und $m(h, t) \in L(v)$
  - vt-Anfrage trifft sich nicht nur in h sondern auch in $m(h, t)$
  - $\rightarrow$ wir können h löschen

Vorberechnung

Idee:

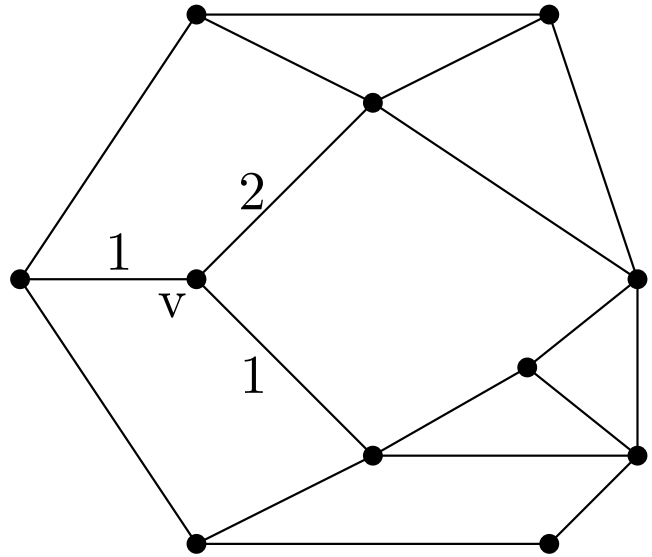
- benutze Knotenordnung



Vorberechnung

Idee:

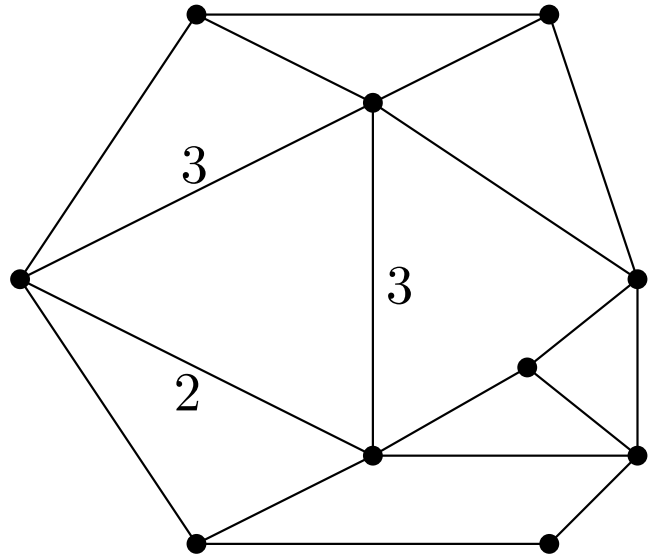
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

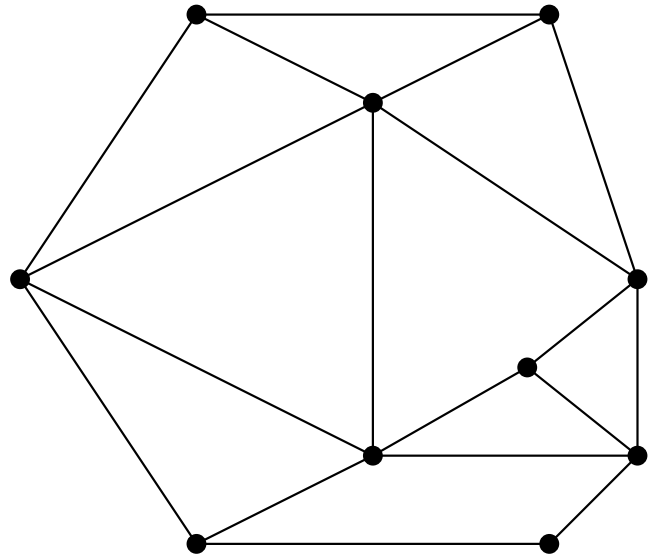
- benutze Knotenordnung
- kontrahiere Knoten v



Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v

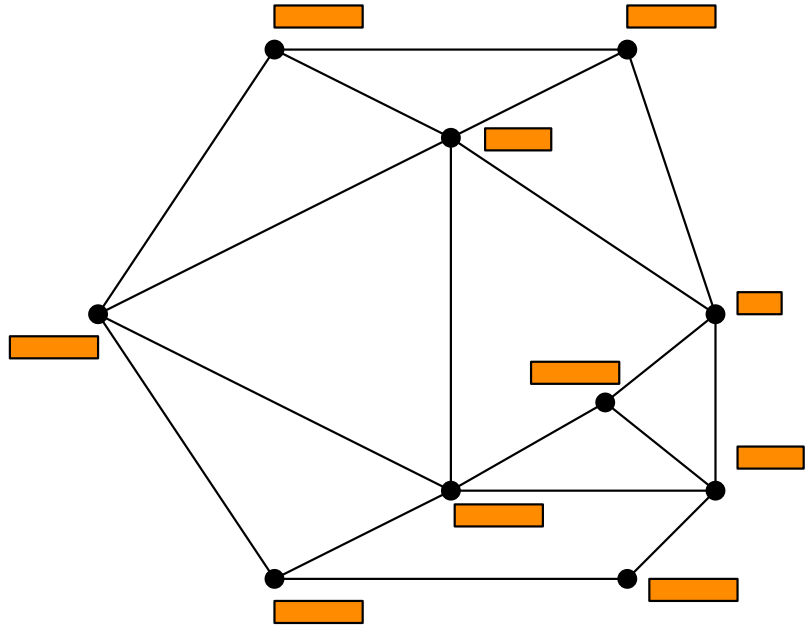


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv

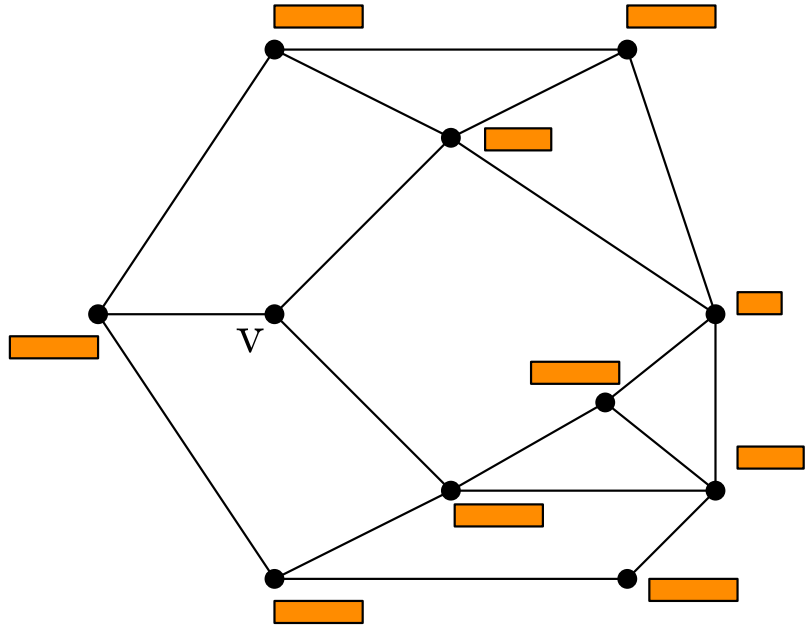


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Vorberechnung

Idee:

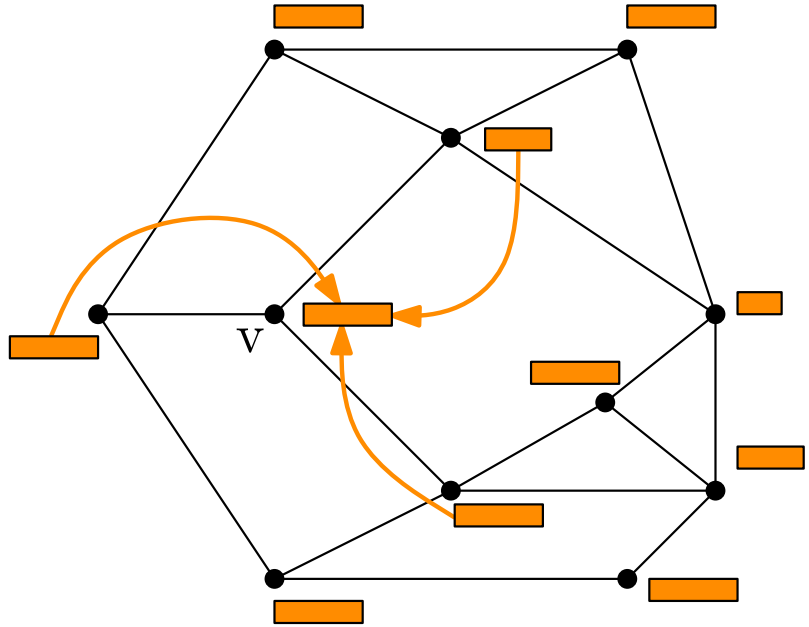
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv



Vorberechnung

Idee:

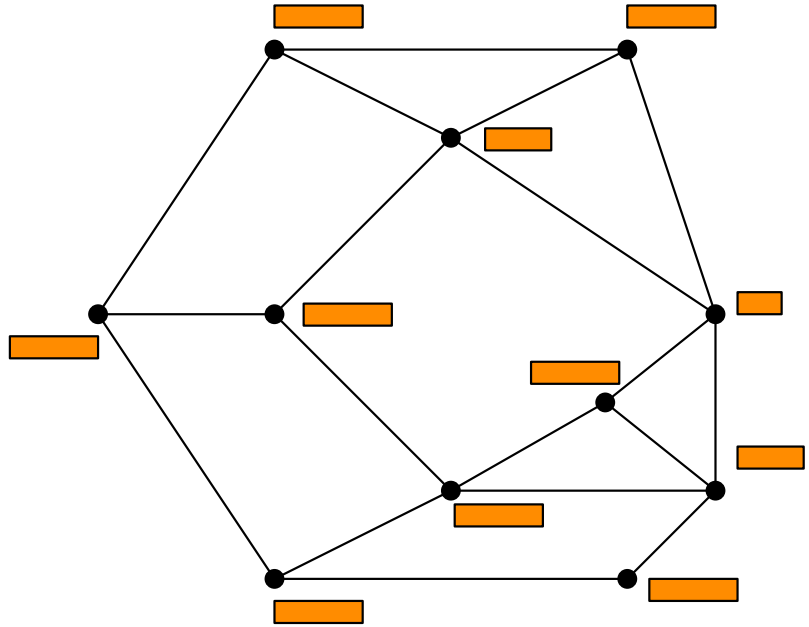
- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- merge Labels der Aufwärts-Nachbarn von v
- dünne Label aus



Vorberechnung

Idee:

- benutze Knotenordnung
- kontrahiere Knoten v
- berechne Labels rekursiv
- merge Labels der Aufwärts-Nachbarn von v
- dünne Label aus



Korrektheit:

- analog zu Korrektheit von CH
- Argumentation über den wichtigsten Knoten auf dem Pfad
- dieser ist im Vorwärtslabel von s und im Rückwärtslabel von t

Mergen von Labels

Generell:

- $L_f(v) = \bigcup_{(v,u) \in G^+} (L_f(u) + w(v,u))$
- wenn ein Hub mehrfach im resultierendem Label, behalte nur den mit minimaler Distanz

Mergen von Labels

Generell:

- $L_f(v) = \bigcup_{(v,u)\in G^+}(L_f(u) + w(v,u))$
- wenn ein Hub mehrfach im resultierendem Label, behalte nur den mit minimaler Distanz

Ausdünnen:

- manche Knoten im Label sind nicht notwendig
- **Ziel**: Für jeden st-Pfad reicht es wenn der Zwischenknoten h mit dem höchsten Rank in $L_f(h)$ und $L_b(h)$ liegt
- Es sei h ein Knoten in $L_f(s)$. Es sei h' der höchste Knoten have dem kürzesten sh-Pfad.
- Wenn $h \neq h'$ dann ist h nie der höchste Knoten auf einem kürzesten st-Pfad durch h
  $\Rightarrow$ Wir können h rauslöschen.

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

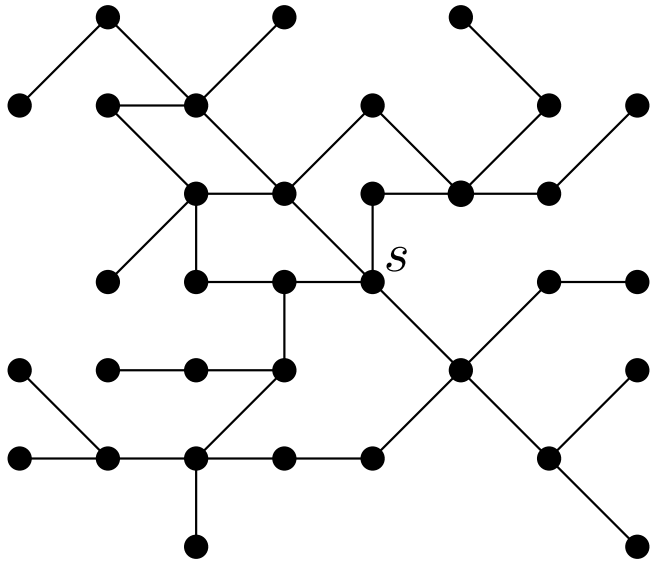
Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen

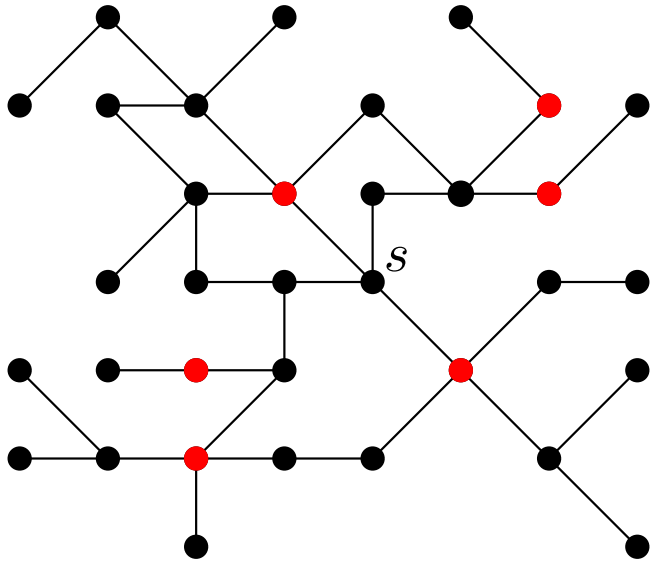


Ziel: s in alle Labels verteilen

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen

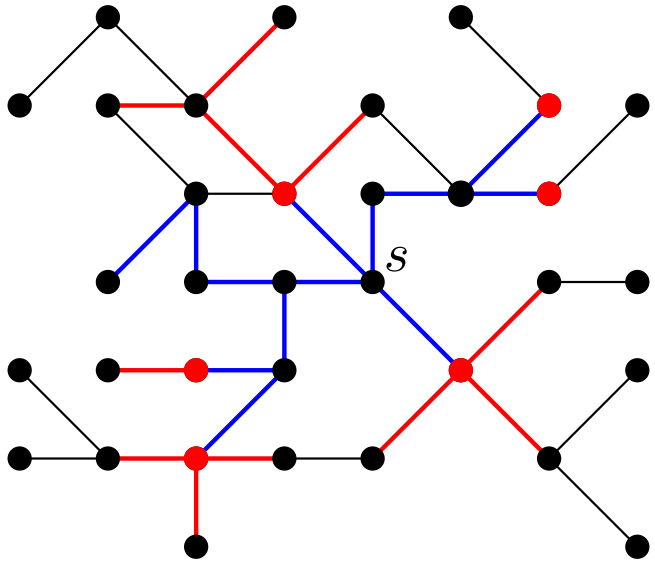


Rote Knoten haben höheren Rank

Pruned Labeling:[DGPW14, AIY13] Alternative Labelkonstruktion

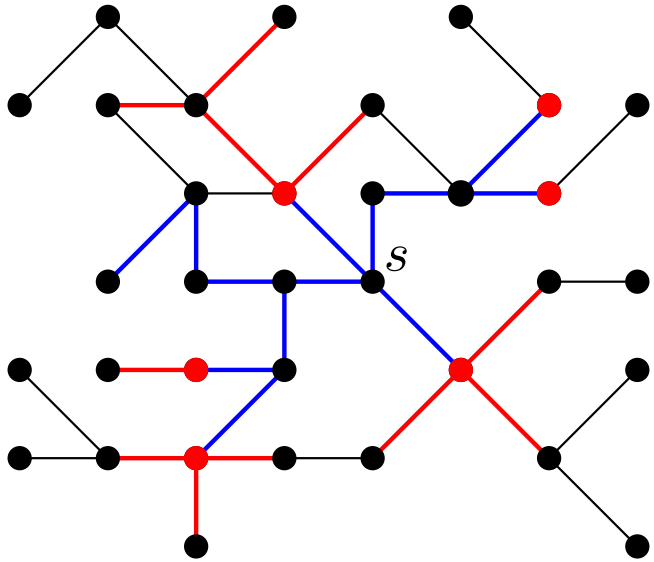
Idee:

- Verteile Hubs auf Labels absteigend nach Knotenordnung
- h ist Hub von $v \iff$ es auf dem hv-Pfad keinen höheren Knoten gibt
- Starte Dijkstra von h und besuche alle v in denen h liegt, breche alle Pfade über höhere Knoten führen



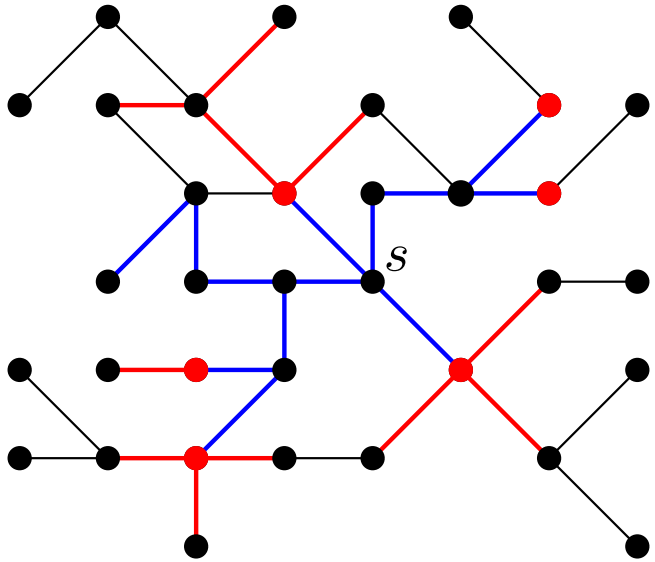
s kommt in über blaue Pfade erreichbaren Label

Pruned Labeling:[DGPW14, AIY13]



- Knoten x ist rot $\iff$ $\exists$ xs-Pfad durch einen roten Knoten
- Bereits aufgebauten partiellen Labels reichen um dies zu testen
- $\rightarrow$ HL-Query auf partiellen Labels kann als Dijkstra-Pruning Regel benutzt werden
- $\rightarrow$ Nur noch der blaue Teilbaum wird besucht
- Pruning macht den Algorithmus leicht schneller

Pruned Labeling:[DGPW14, AIY13]



- Knoten x ist rot $\iff \exists$ xs-Pfad durch einen roten Knoten
- Bereits aufgebauten partiellen Labels reichen um dies zu testen
- $\rightarrow$ HL-Query auf partiellen Labels kann als Dijkstra-Pruning Regel benutzt werden
- $\rightarrow$ Nur noch der blaue Teilbaum wird besucht
- Pruning macht den Algorithmus leicht schneller
- Pruned Labeling funktioniert besser auf Graphen mit hohen Knotengrad, als der Grundalgorithmus.

Label Greedy - Ordnung [ADGW12]

Sei U_v die Menge der überdeckten Pfaden auf denen v liegt.

Zur Erinnerung:

- Knoten werden bei Path-Greedy nach Anzahl überdeckter Pfade geordnet
- d.h. wähle v, so dass so viele Pfade wir möglich überdeckt werden
- d.h. wähle v, so dass $|U_v|$ groß ist

Erweiterung:

- Wir wollen auch wenig Labeleinträge erzeugen
- Wie viele Labeleinträge werden bei der Wahl von v erzeugt?
  - Sei S_v die Menge der Startknoten aus U_v
  - Sei T_v die Menge der Zielknoten aus U_v
  - Es werden $|S_v| + |T_v|$ Knoten erzeugt
- Wir wollen also, dass $|S_v| + |T_v|$ klein wird
- $\rightarrow$ maximiere $|U_v|/(|S_v| + |T_v|)$

Ergebnisse

	preprocessing		query
method	time [h:m]	space [GB]	time [μs]
MLD-3	$<$ 0:01	0.4	912
CH	0:02	0.4	96.3
HL-0	0:03	22.5	0.700
HL-15	0:05	18.8	0.556
HL-17	0:25	18.4	0.545
HL-∞	5:43	16.8	0.508
HL-∞ + Oracle	6:12	17.7	0.254
Table Lookup	???	1 208 358.7	0.056

- Vorberechnung mit 12 Cores parallelisiert
- Table Lookup nimmt an, dass Speicher nicht im Cache liegt

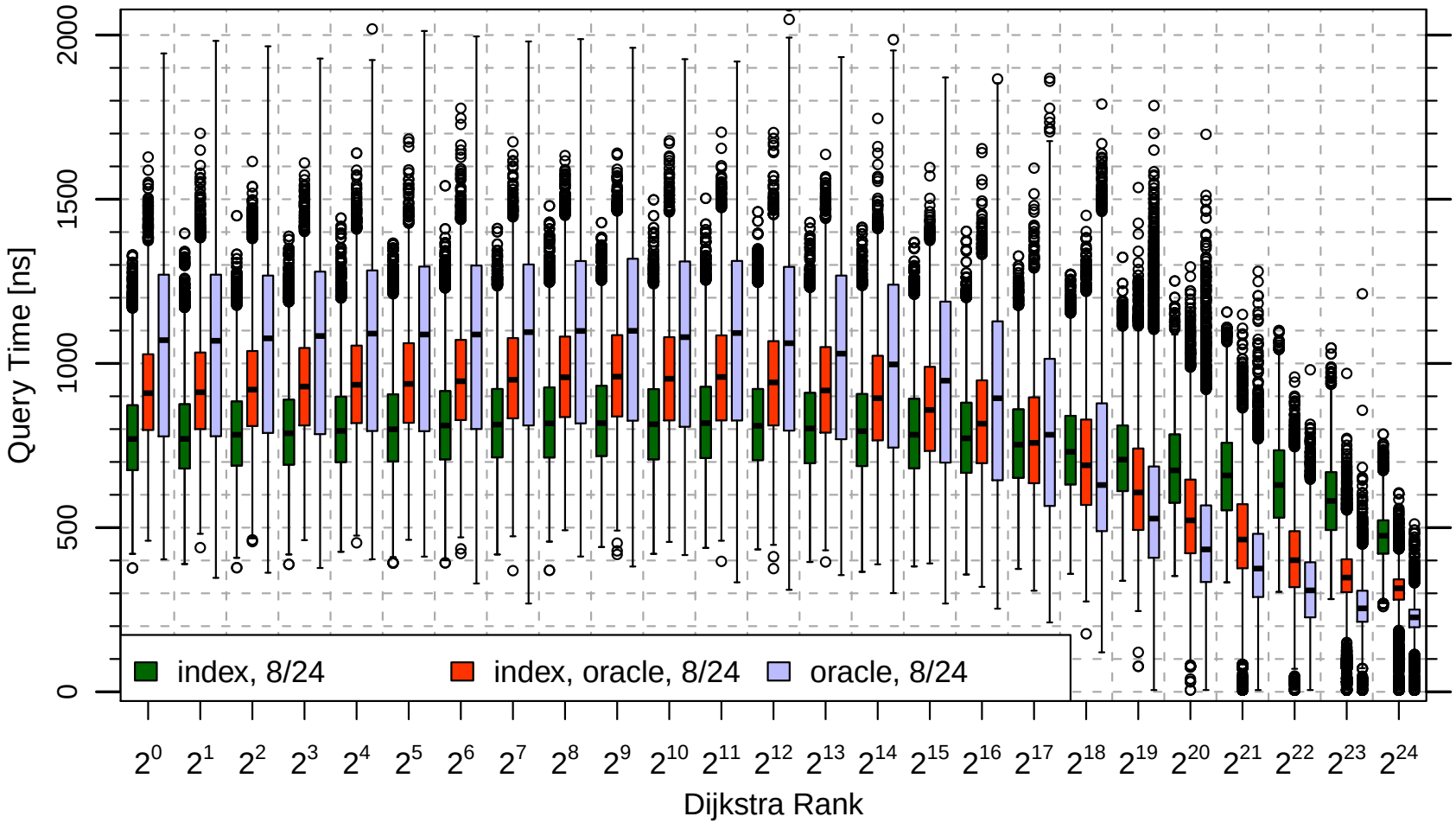
Ergebnisse

	preprocessing		query
method	time [h:m]	space [GB]	time [μs]
MLD-3	$<$ 0:01	0.4	912
CH	0:02	0.4	96.3
HL-0	0:03	22.5	0.700
HL-15	0:05	18.8	0.556
HL-17	0:25	18.4	0.545
HL-∞	5:43	16.8	0.508
HL-∞ + Oracle	6:12	17.7	0.254
Table Lookup	???	1 208 358.7	0.056

- Vorberechnung mit 12 Cores parallelisiert
- Table Lookup nimmt an, dass Speicher nicht im Cache liegt

- HL ist Faktor 100 schneller als CH (Speedup 10 Mio)
- hoher Speicherverbrauch (durch Kompression reduzierbar)

Lokale Queries



Zusammenfassung

- Knotenordnung definiert Labeling
- Beschleunigung gegenüber CH von Faktor mehr als 100
- durch bessere Lokalität
- nur 5 mal langsamer als ein Speicherzugriff
- schnellster Algorithmus momentan
- beschleunigt lokale und globale Anfragen
- aber Speicherverbrauch sehr hoch
- wird zu einem späterem Zeitpunkt noch einmal wichtig

HLDB

Inner Join

a	b
	1
	2
	2
	3
	4

INNER JOIN

b	c
0	
1	
2	
4	
5	

USING(b)

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	

HLDB

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
);
```


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

HLDB - Anfrage

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;
```


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

HLDB - Warum?

- Warum HL in SQL?
- Grund 1: Weil wir es können
  - Es ist interessant, dass es geht
  - Dijkstras Algorithmus geht nicht in reinem SQL
- Grund 2: Man darf nicht auf jedem Server Programme starten
  - z.B. PHP/SQL - Sharedhoster

HLDB - Pfadextraktion

Ansatz 1:

- Speichere für jeden Hub zwei Shortcuts und entpacke den Pfad rekustiv
- Analog zur CH
- **Problem**: verglichen mit Distanzanfrage langsam $\rightarrow$ nicht mehr viel schneller als eine CH

HLDB - Pfadextraktion

Ansatz 1:

- Speichere für jeden Hub zwei Shortcuts und entpacke den Pfad rekustiv
- Analog zur CH
- **Problem**: verglichen mit Distanzanfrage langsam $\rightarrow$ nicht mehr viel schneller als eine CH

Ansatz 2:

- Speichere für jeden Shortcut den vollständigen entpackten Pfad
- **Problem**: braucht viel Speicher, aber HL braucht eh viel Speicher
- wer HL einsetzen
  - einen nicht allzugroßen Graph
  - ist bereit viel in RAM-Riegel zu investieren
- (Die Stärke von HL liegt nicht in der Pfadextraktion)

HLDB - Pfadextraktion

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.

HLDB - Pfadextraktion

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.

HLDB - Pfadextraktion

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
);
```

HLDB - Pfadextraktion

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.

HLDB - Pfadextraktion

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;
```

Literatur I

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

Hierarchical hub labelings for shortest paths.

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

Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida.

Fast exact shortest-path distance queries on large networks by pruned landmark labeling.

In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD'13)*, pages 349–360. ACM Press, 2013.

Maxim Babenko, Andrew V. Goldberg, Haim Kaplan, Ruslan Savchenko, and Mathias Weller.

On the complexity of hub labeling.

Technical report, ArXiv, 2015.

Literatur II

Daniel Delling, Andrew V. Goldberg, Thomas Pajor, and Renato F. Werneck.
Robust distance queries on massive networks.
In *Proceedings of the 22nd Annual European Symposium on Algorithms (ESA'14)*, volume 8737 of *Lecture Notes in Computer Science*, pages 321–333. Springer, September 2014.

Daniel Delling, Andrew V. Goldberg, Ruslan Savchenko, and Renato F. Werneck.
Hub labels: Theory and practice.
In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 259–270. Springer, 2014.

Cyril Gavoille, David Peleg, Stéphane Pérennes, and Ran Raz.
Distance labeling in graphs.
Journal of Algorithms, 53:85–112, 2004.