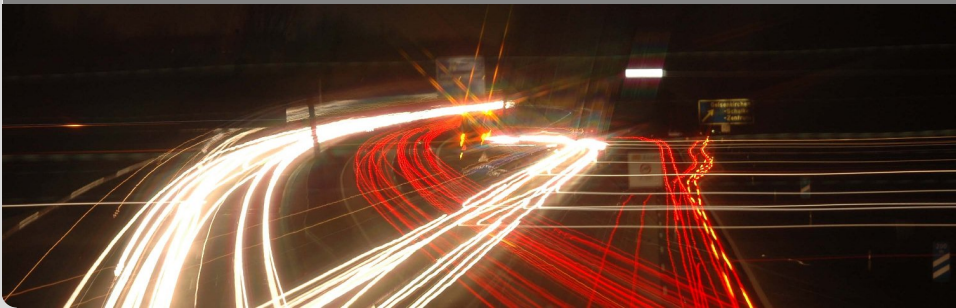


Algorithmen für Routenplanung

7. Vorlesung, Sommersemester 2019

Valentin Buchhold | 20. Mai 2019

INSTITUT FÜR THEORETISCHE INFORMATIK · ALGORITHMIK



Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

- Wiederholung: 3 Phasen
- Wiederholung: CH-Basisvariante
- Customizable Contraction Hierarchies (CCH)

Wiederholung: 3 Phasen

Eingabe:

- Knoten
- Kanten
- Positive Kantengewichte
- Quellknoten s
- Zielknoten t

Ausgabe:

- Ein kürzester Pfad

Beispiel Algorithmus:

- Dijkstras Algorithmus

Phase 1 (Vorbereitung) Eingabe:

- Knoten
- Kanten
- Positive Kantengewichte

Phase 2 (Anfrage) Eingabe:

- Quellknoten s
- Zielknoten t

Phase 2 (Anfrage) Ausgabe:

- Ein kürzester Pfad

Beispiel Algorithmus:

- CH, Arc-Flags, ALT ...

Phase 1 (Vorbereitung) Eingabe:

- Knoten
- Kanten

Phase 2 (Customization) Eingabe:

- Positive Kantengewichte

Phase 3 (Anfrage) Eingabe:

- Quellknoten s
- Zielknoten t

Phase 3 (Anfrage) Ausgabe:

- Ein kürzester Pfad

Beispiel Algorithmus:

- MLD, CCH

Erwünschte Laufzeiten

Phase:	Vorbereitung	Customization	Anfrage
Geschwindigkeit:	Stunden	Sekunde	Millisekunde

Idee Vorbereitung

- Führt man durch, wenn man das Kartenmaterial austauscht
- Kommt selten vor
- Darf eine Nacht rechnen

Erwünschte Laufzeiten

Phase:	Vorberechnung	Customization	Anfrage
Geschwindigkeit:	Stunden	Sekunde	Millisekunde

Idee Customization

- Führt man durch, wenn sich die Verkehrslage ändert
- Die Verbreitung von Verkehrsmeldungen braucht mehrere Minuten
- Routing sollte das Einpflegen der Daten nicht verzögern
→ Sekunden sind schnell genug

Erwünschte Laufzeiten

Phase:	Vorberechnung	Customization	Anfrage
Geschwindigkeit:	Stunden	Sekunde	Millisekunde

Idee Customization (Alternative)

- Pro Benutzeraccount eigene Gewichte
 - Individualisierte Gewichte
 - Namensgebend für Phase
- Customization wird bei Benutzerlogin durchgeführt
 - Sekunde ist ok, schneller besser
 - Braucht viele Ressourcen pro Benutzer

Erwünschte Laufzeiten

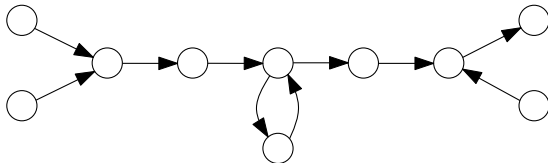
Phase:	Vorberechnung	Customization	Anfrage
Geschwindigkeit:	Stunden	Sekunde	Millisekunde

Idee Anfrage

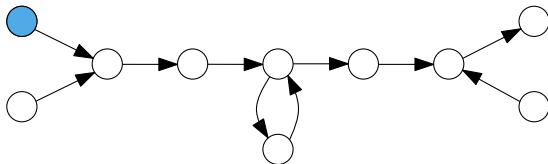
- Benutzer fragt nach Pfad
- Muss schnell sein, da
- Viele Anfragen pro Sekunde (Durchsatz)
- Drag & Drop smooth wirken soll (Reaktivität)
 - Bei erwünschter Bildwiederholrate von 20 Hz:
 - max. 50 ms für Netzwerk und Pfadberechnung

Wiederholung: CH

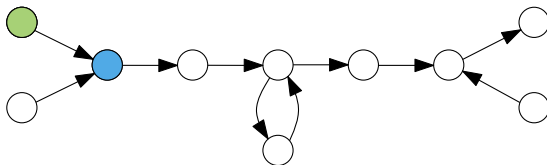




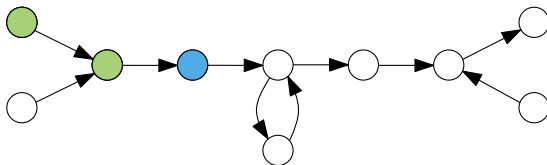
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



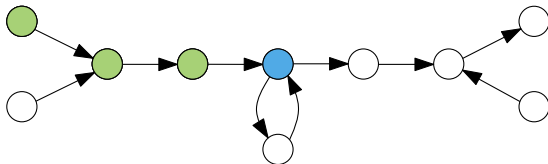
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



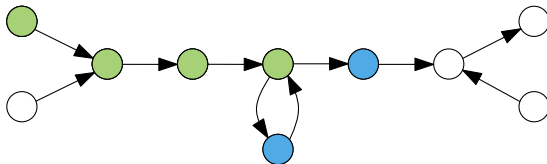
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



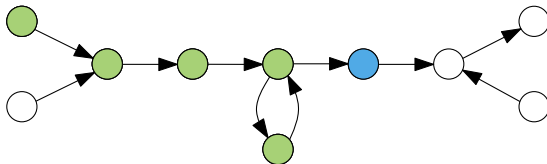
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



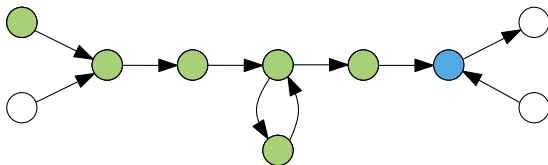
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



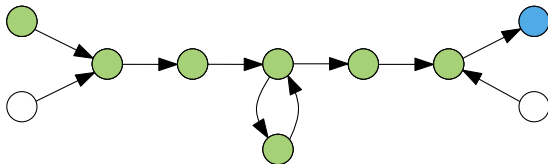
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



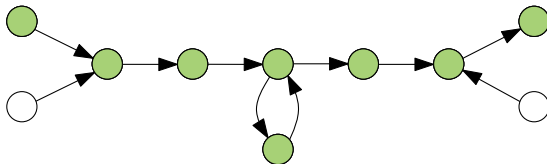
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



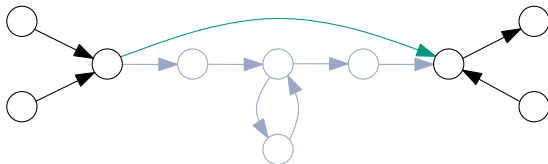
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



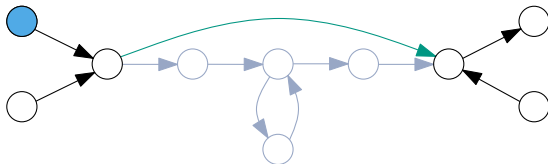
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



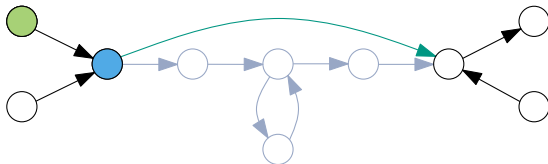
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



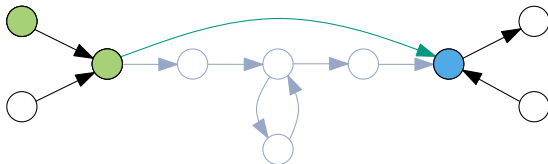
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



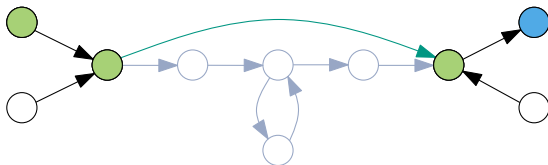
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



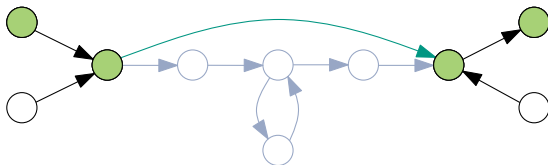
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

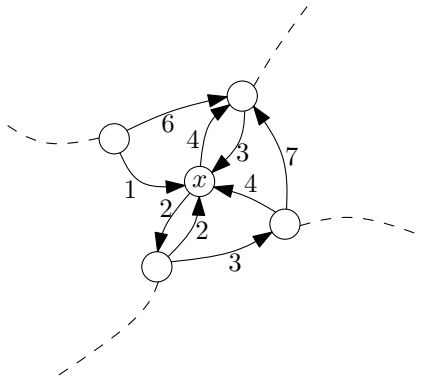


Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



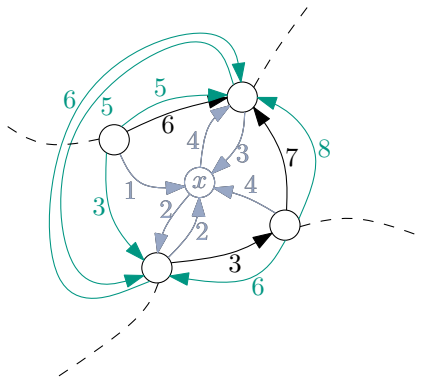
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

Knotenkontraktion von x



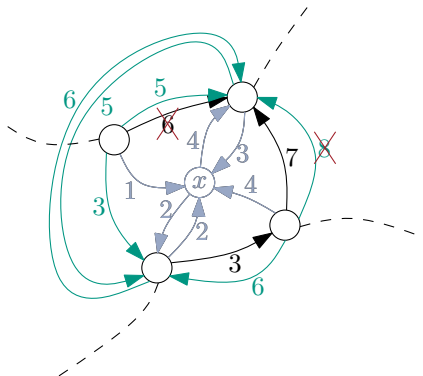
Kontraktion von x : Lösche x und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

Knotenkontraktion von x



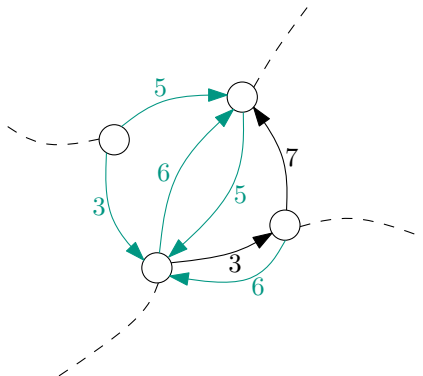
Kontraktion von x : Lösche x und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

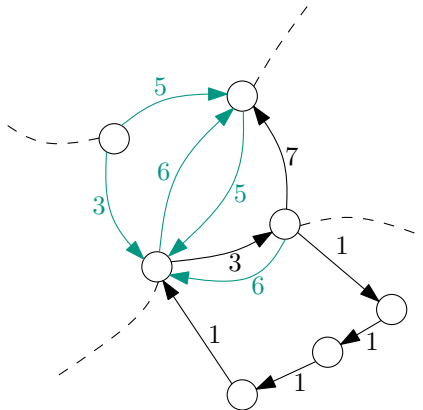
Knotenkontraktion von x



Bei Mehrfachkanten: Längere Kanten verwerfen

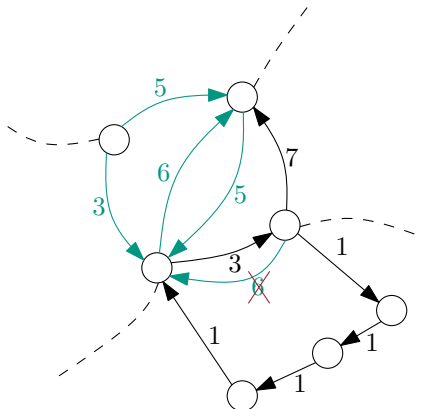
Knotenkontraktion von x





Falls es einen kürzeren Pfad durch den Restgraphen gibt, dann kann man einen Shortcut auch verwerfen.

Suche nach solchem Pfad heißt Zeugensuche/Witness Search



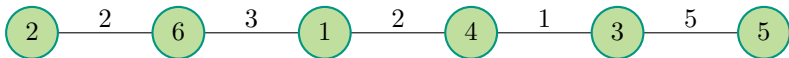
Falls es einen kürzeren Pfad durch den Restgraphen gibt, dann kann man einen Shortcut auch verwerfen.

Suche nach solchem Pfad heißt Zeugensuche/Witness Search

Grundidee

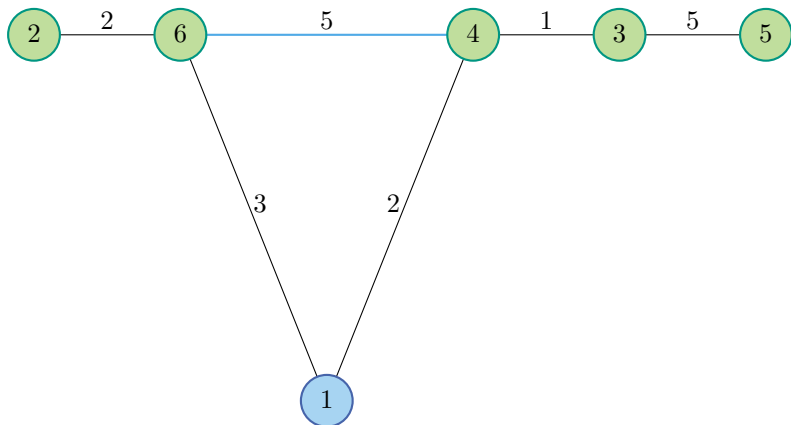
- Eingabe Graph G
- Ordne Knoten von G nach „Wichtigkeit“: $v_1 \dots v_n$
- Kontrahiere Knoten iterativ aus G raus
 - zuerst den „unwichtigsten“ Knoten v_1
 - den „wichtigsten“ Knoten v_n als letztes
- Graph mit Shortcuts heißt augmentierter Graph

Contraction Hierarchy



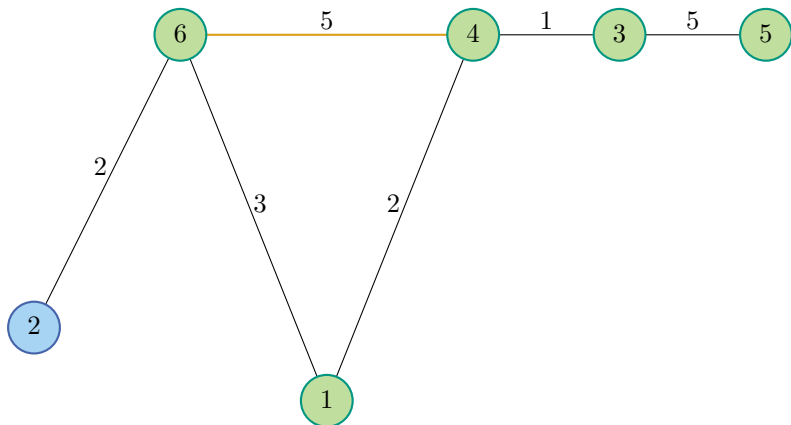
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



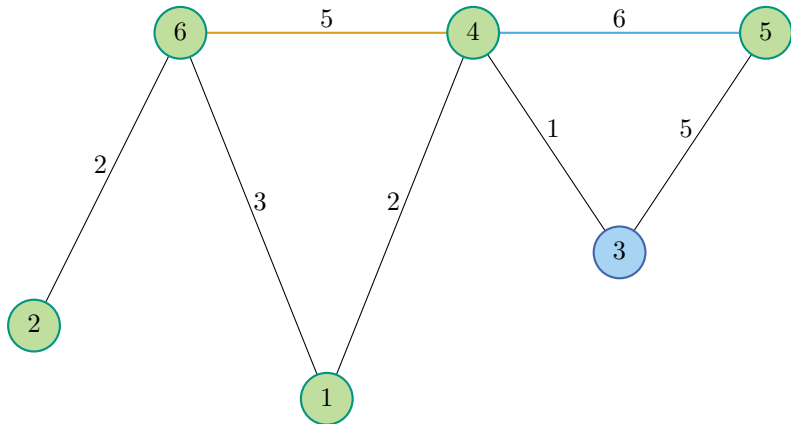
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



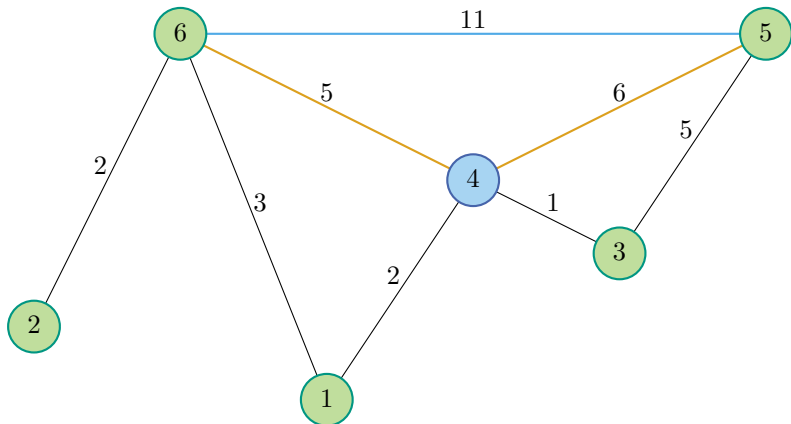
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



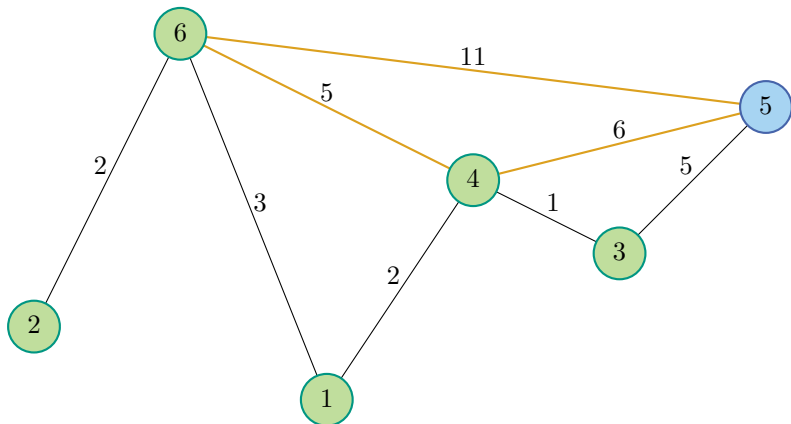
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



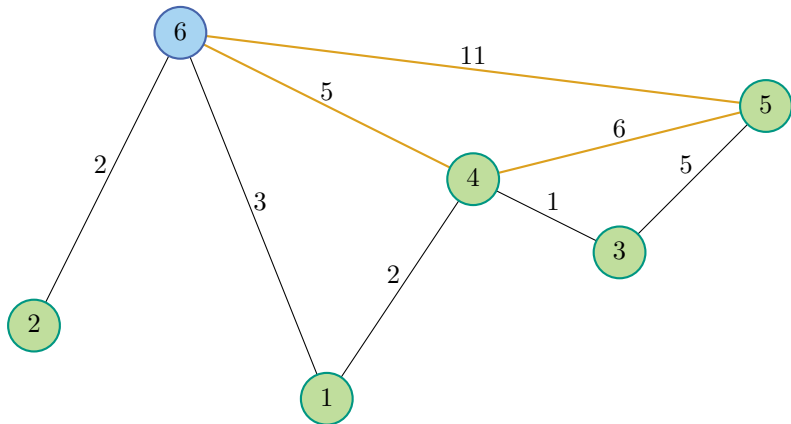
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



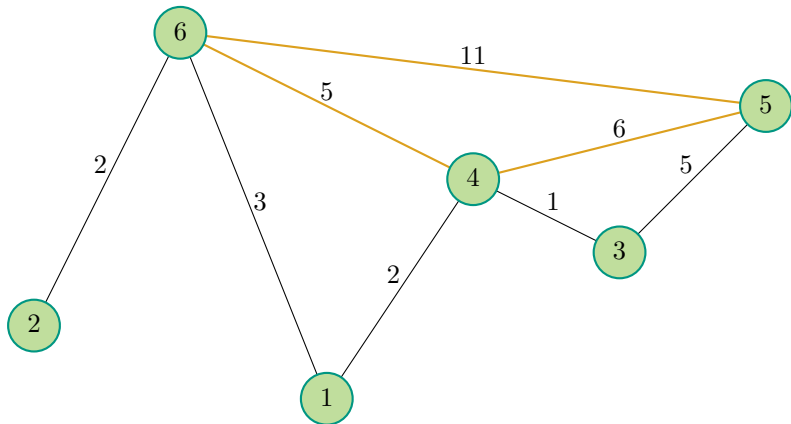
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



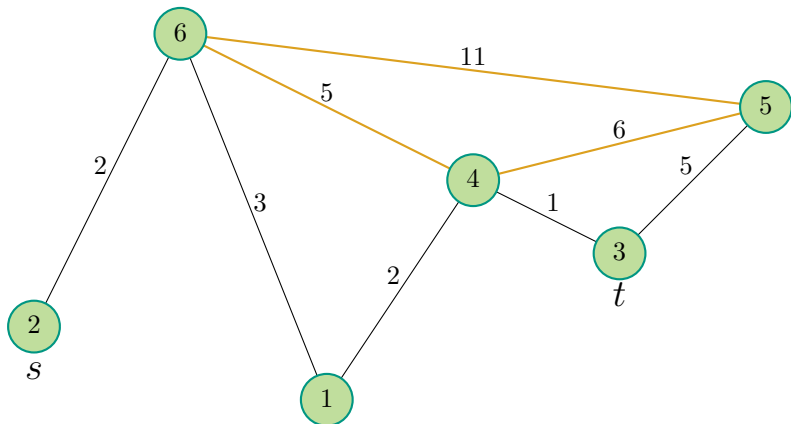
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



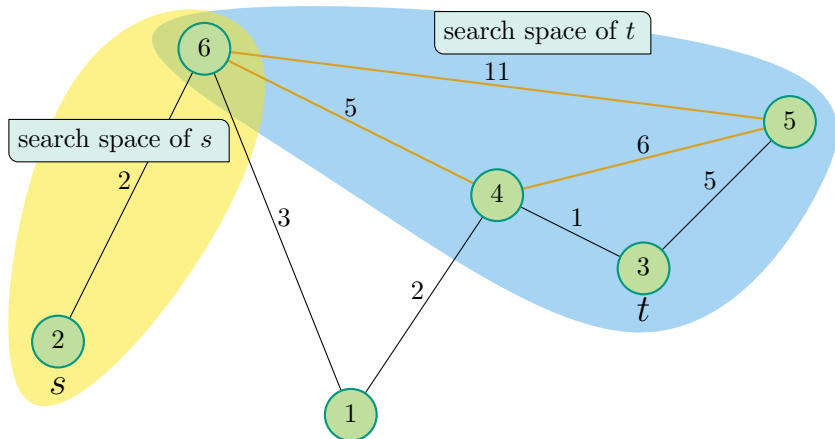
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



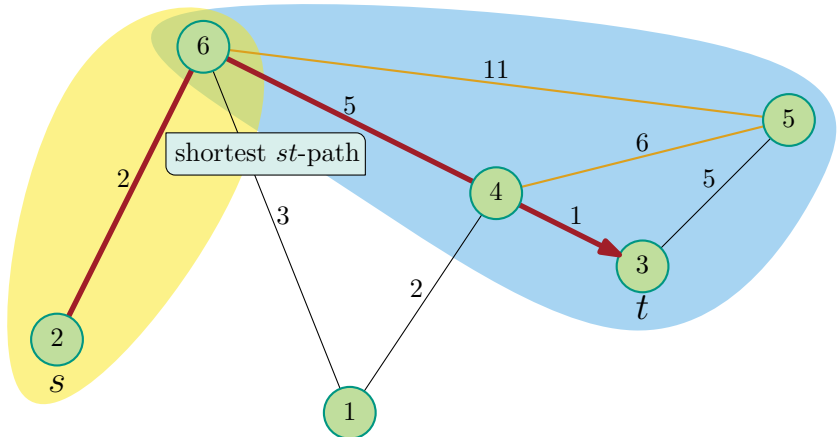
Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



Knoten nummeriert nach „Wichtigkeit“

Contraction Hierarchy



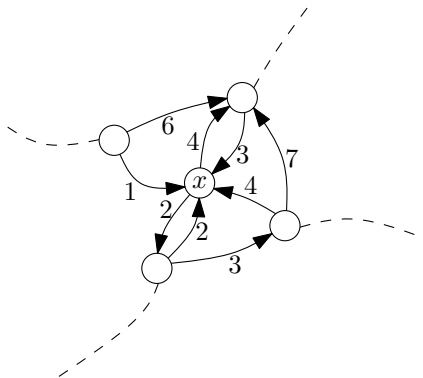
Für jeden ursprünglichen kürzesten Weg gibt es einen hoch-runter-Pfad

- Bidirektionale Variante von Dijkstras Algorithmus
- Verfolge nur Kanten zu wichtigeren Knoten
- Vorwärtssuche findet den „hoch“-Teil des Pfads
- Rückwärtssuche findet den „runter“-Teil des Pfads

Customizable Contraction Hierarchies

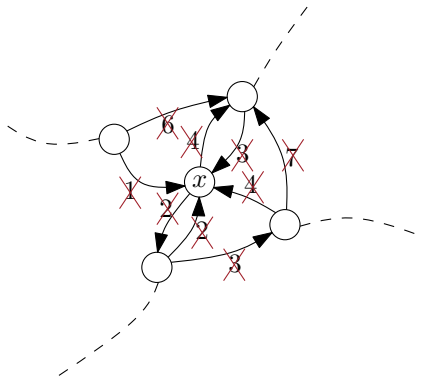
[DSW14, BCRW13, DSW16]

Knotenkontraktion von x



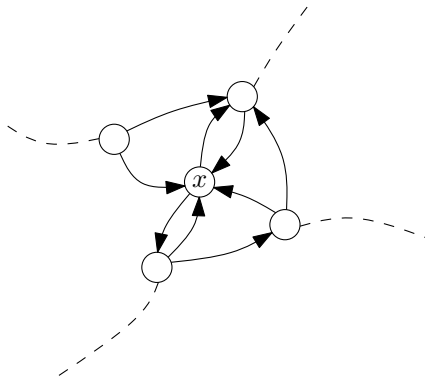
Kontraktion von x

Knotenkontraktion von x



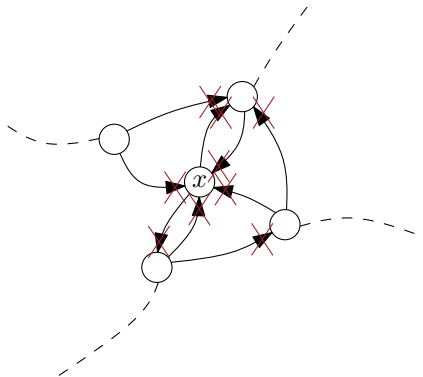
Es gibt keine Kantengewichte

Knotenkontraktion von x



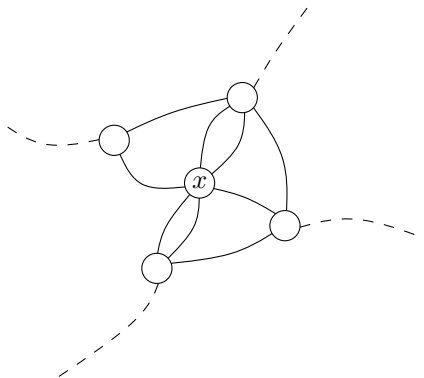
Es gibt keine Kantengewichte

Knotenkontraktion von x



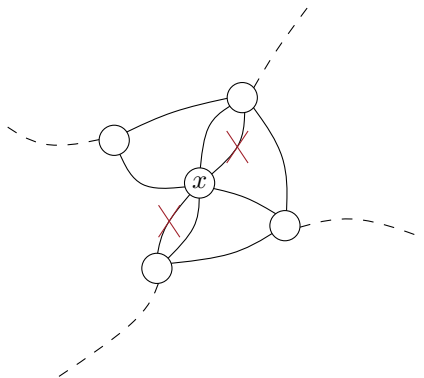
Kantenrichtung vergessen wir auch

Knotenkontraktion von x



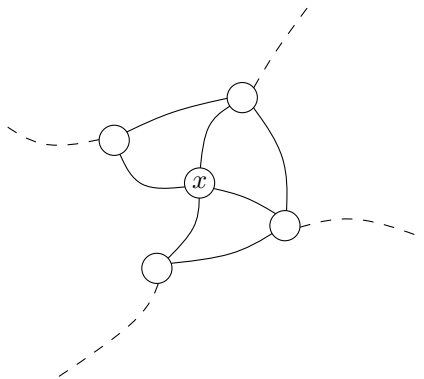
Kantenrichtung vergessen wir auch

Knotenkontraktion von x



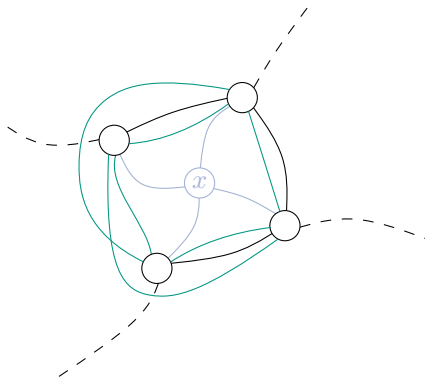
Multikanten brauchen wir nicht

Knotenkontraktion von x



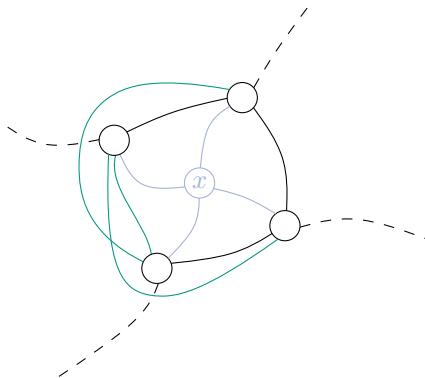
Multikanten brauchen wir nicht

Knotenkontraktion von x



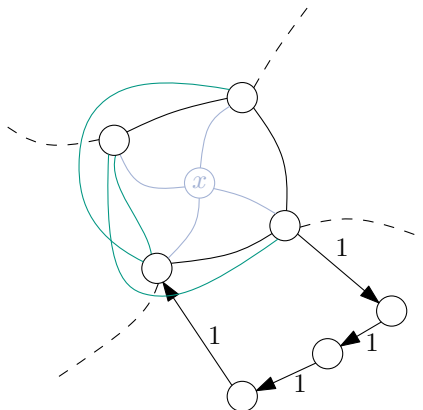
Kanten zwischen allen Nachbarn (Clique)

Knotenkontraktion von x

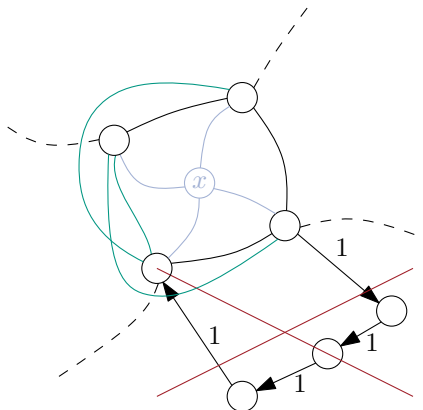


Multikanten wollen wir nicht

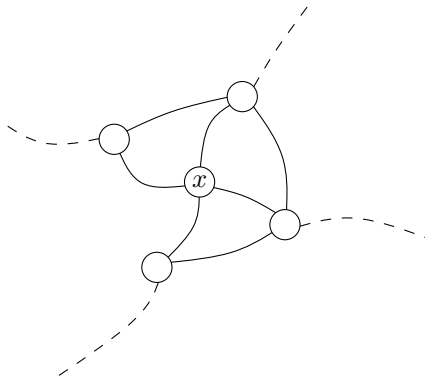
Knotenkontraktion von x



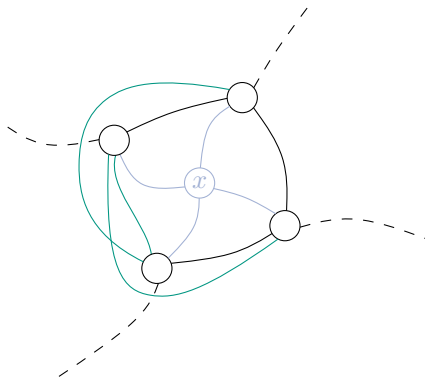
Zeugensuche geht nicht ohne Kantengewichte



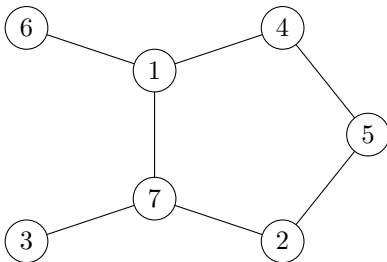
Zeugensuche geht nicht ohne Kantengewichte



Zusammenfassung: Wir fügen in diesem Beispiel drei Kanten ein

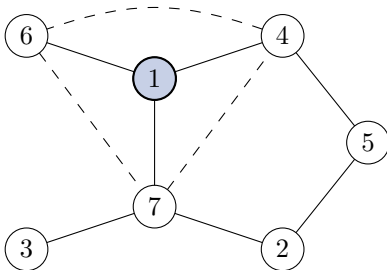


Zusammenfassung: Wir fügen in diesem Beispiel drei Kanten ein



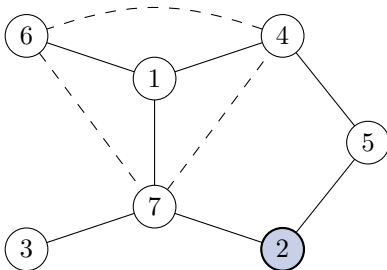
- Kontrahiere Knoten iterativ

Ungewichtete Contraction Hierarchy



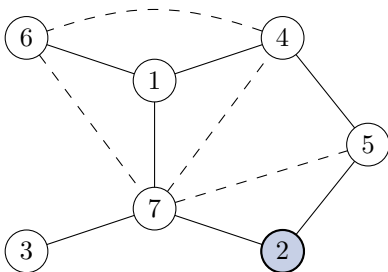
- Kontrahiere Knoten iterativ

Ungewichtete Contraction Hierarchy



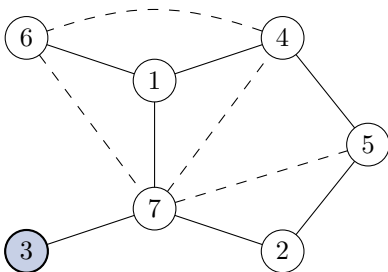
- Kontrahiere Knoten iterativ

Ungewichtete Contraction Hierarchy



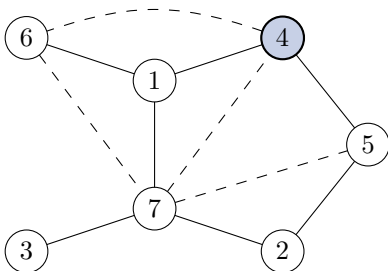
- Kontrahiere Knoten iterativ

Ungewichtete Contraction Hierarchy



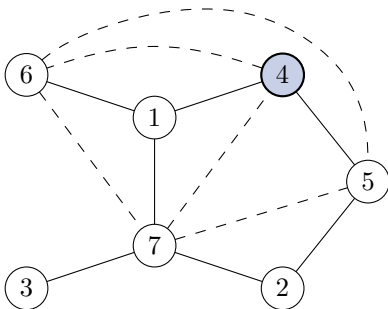
- Kontrahiere Knoten iterativ

Ungewichtete Contraction Hierarchy



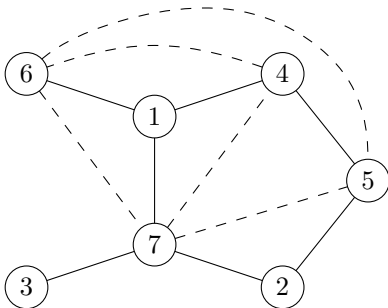
- Kontrahiere Knoten iterativ

Ungewichtete Contraction Hierarchy

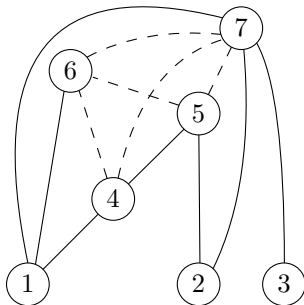


- Kontrahiere Knoten iterativ

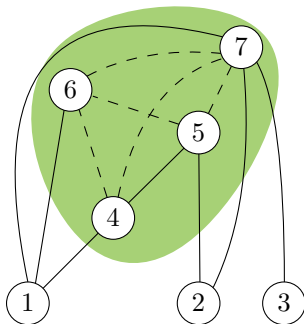
Ungewichtete Contraction Hierarchy



- Kontrahiere Knoten iterativ
- CCH-Graph



- Kontrahiere Knoten iterativ
- CCH-Graph
- Ordne Knoten nach Kontraktionsreihenfolge an



- Kontrahiere Knoten iterativ
- CCH-Graph
- Ordne Knoten nach Kontraktionsreihenfolge an
- Suchraum von einem Knoten sind höhere erreichbare Knoten (im Beispiel der Suchraum von Knoten 4)

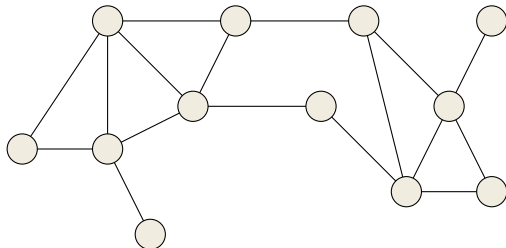
CCH: Knotenordnung



Knotenseparator

Ein **Knotenseparator** S von $G = (V, E)$

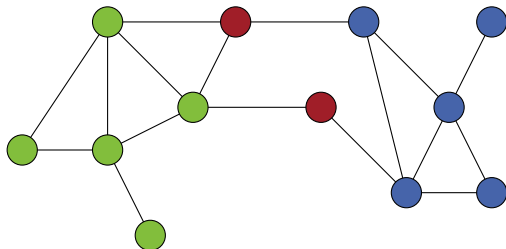
- ist eine Knotenmenge
- $G \setminus S$ zerfällt in zwei Teile G_1 und G_2
- es gibt keine direkte Kante zwischen G_1 und G_2



Knotenseparator

Ein **Knotenseparator** S von $G = (V, E)$

- ist eine Knotenmenge
- $G \setminus S$ zerfällt in zwei Teile G_1 und G_2
- es gibt keine direkte Kante zwischen G_1 und G_2

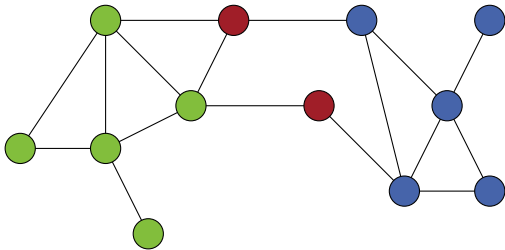


G_1 ist grün, G_2 ist blau, S ist rot

Balancierter Knotenseparator

Ein Knotenseparator S ist α -balanciert wenn

- $|V_1| \leq \alpha |V|$
- $|V_2| \leq \alpha |V|$

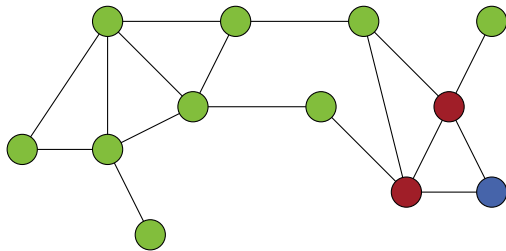


ist $\frac{2}{3}$ -balanciert

Balancierter Knotenseparator

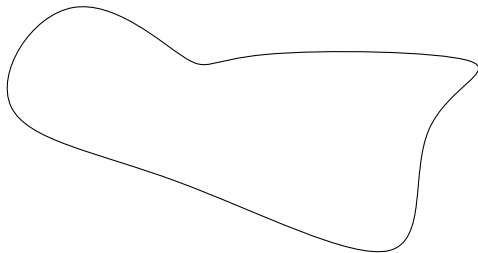
Ein Knotenseparator S ist α -balanciert wenn

- $|V_1| \leq \alpha |V|$
- $|V_2| \leq \alpha |V|$



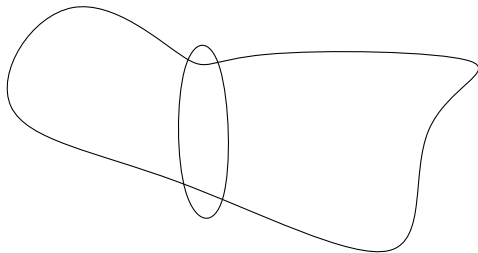
ist nicht $\frac{2}{3}$ -balanciert

Nested Dissection (ND)



Order:

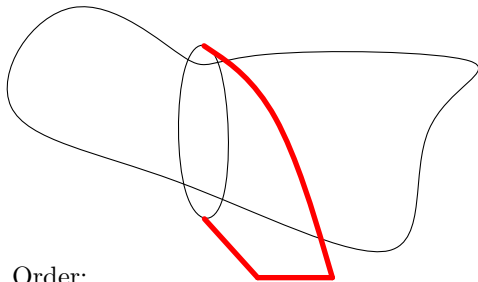
Nested Dissection (ND)



Order:

Finde kleinen balancierten Separator

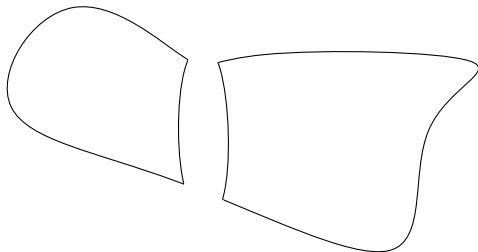
Nested Dissection (ND)



Order:

Füge die Knoten hinten in die Ordnung

Nested Dissection (ND)

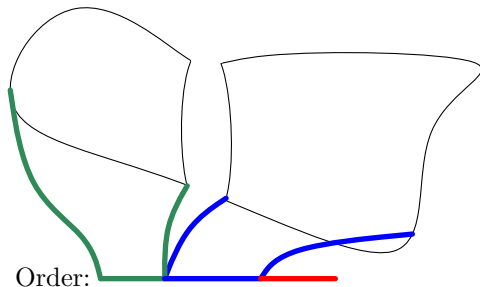


Order:



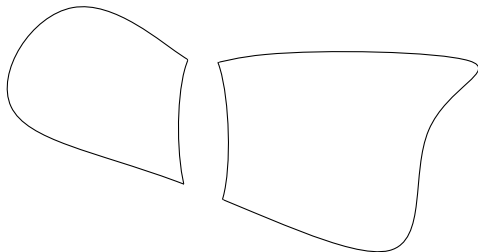
Lösche den Separator

Nested Dissection (ND)



Rekursion auf beiden Teilen

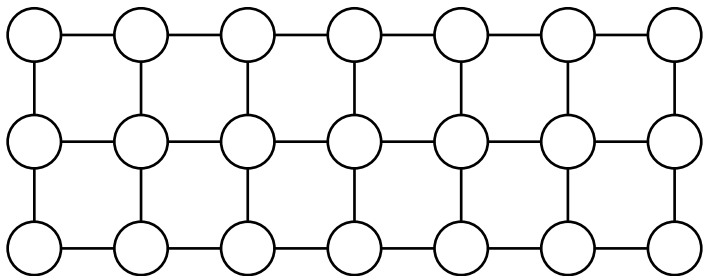
Nested Dissection (ND)



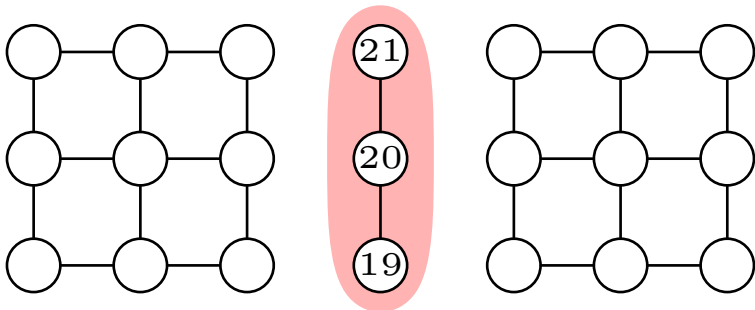
Order: 

Die finale Ordnung

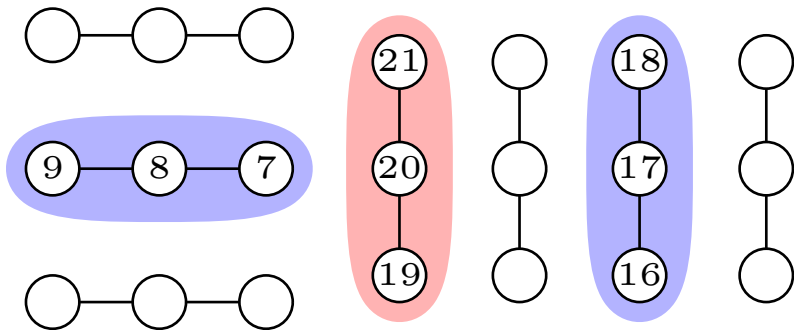
Nested Dissection (Beispiel)



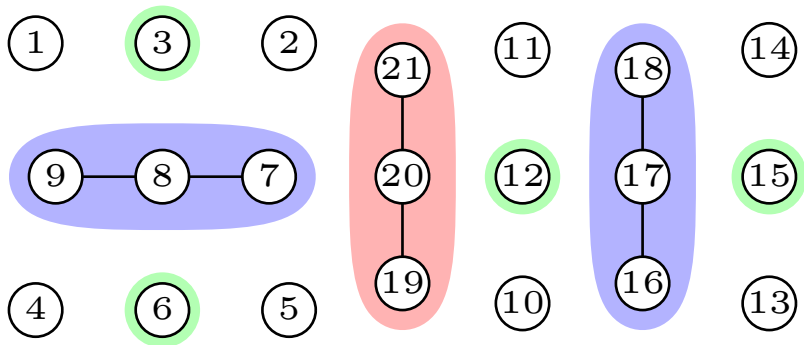
Nested Dissection (Beispiel)



Nested Dissection (Beispiel)



Nested Dissection (Beispiel)



Definition

Eine Familie F von Graphen hat rekursive α -balancierte Separatoren der Größe $O(n^\beta)$, wenn jeder Graph $G \in F$

- einen α -balancierten Separator S hat,
- so dass G_1 und G_2 in der Familie F sind.

Definition

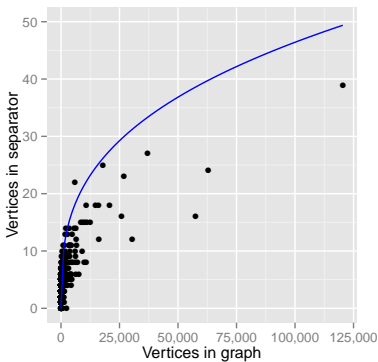
Eine Familie F von Graphen hat rekursive α -balancierte Separatoren der Größe $O(n^\beta)$, wenn jeder Graph $G \in F$

- einen α -balancierten Separator S hat,
- so dass G_1 und G_2 in der Familie F sind.

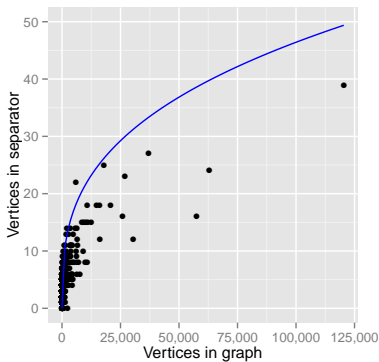
Beispiel

Alle planare Graphen haben rekursive $\frac{2}{3}$ -balancierte Separatoren der Größe $O(\sqrt{n})$.

Beweis: Siehe Vorlesung zu planaren Graphen



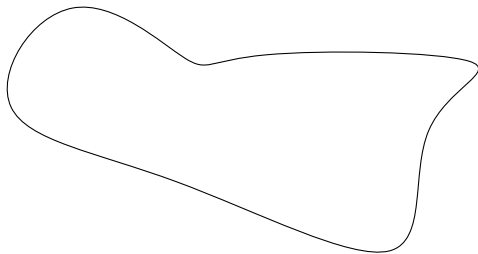
Separatoren für Straßengraph (in der weiteren Umgebung) von Karlsruhe. Blaue Funktion ist $y = \sqrt[3]{x}$.



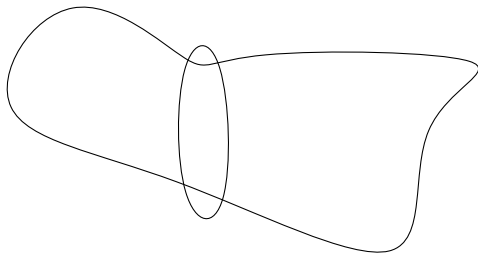
Separatoren für Straßengraph (in der weiteren Umgebung) von Karlsruhe. Blaue Funktion ist $y = \sqrt[3]{x}$.

Annahme: Straßengraphen haben $O(\sqrt[3]{x})$ rekursive $\frac{2}{3}$ -balancierte Separatoren.

- Wie groß kann der Suchraum werden?

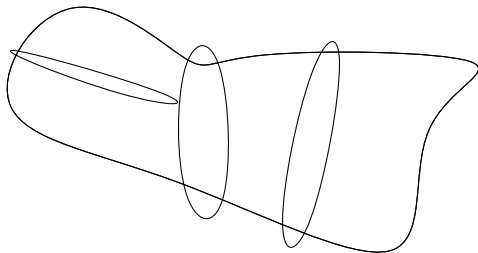


- Wie groß kann der Suchraum werden?



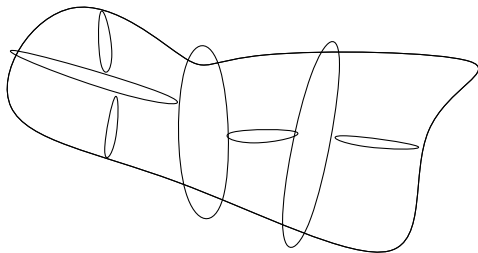
Level 0 Separator

- Wie groß kann der Suchraum werden?



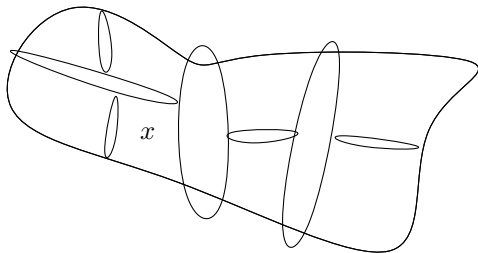
Level 1 Separatoren

- Wie groß kann der Suchraum werden?



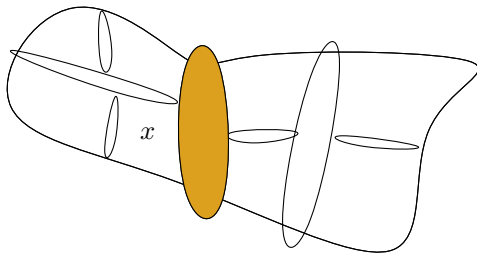
Level 2 Separatoren

- Wie groß kann der Suchraum werden?



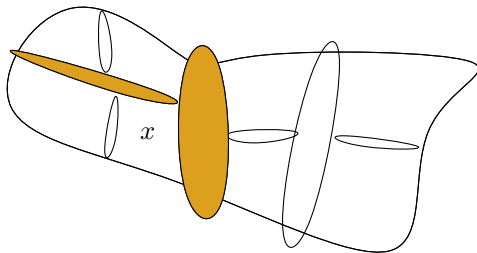
Wie viele Knoten sind im Suchraum von x ?

- Wie groß kann der Suchraum werden?



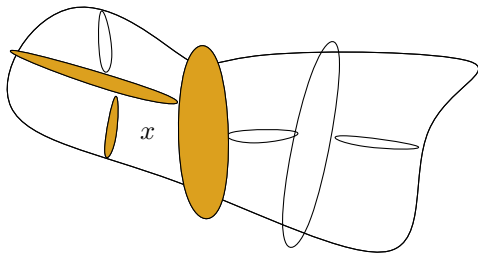
Auf Level 0 höchstens n^{β} viele

- Wie groß kann der Suchraum werden?



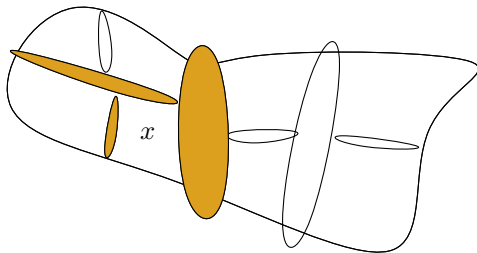
Auf Level 1 höchstens $(\alpha \cdot n)^\beta$ viele

- Wie groß kann der Suchraum werden?



Auf Level 2 höchstens $(\alpha^2 \cdot n)^\beta$ viele

- Wie groß kann der Suchraum werden?



Auf Level i höchstens $(\alpha^i \cdot n)^\beta$ viele

- Insgesamt höchstens $\sum_{i=0}^{\infty} (\alpha^i \cdot n)^\beta$ viele Knoten

$$\begin{aligned} & \sum_{i=0}^{\infty} (\alpha^i \cdot n)^\beta \\ &= n^\beta \sum_{i=0}^{\infty} (\alpha^\beta)^i \\ &= \frac{n^\beta}{1 - \alpha^\beta} \\ &\in O(n^\beta) \end{aligned}$$

- Konvergiert wegen $\alpha^\beta < 1$ und geometrischer Reihe
- $O(n^\beta)$ viele Knoten im Suchraum
- $O(n^{2\beta})$ viele Kanten im Suchraum

CCH: Vorberechnung



Schritt 1

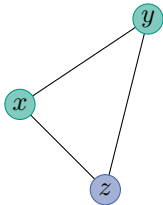
- Berechne ND-Ordnung
- Subroutine Graph-Bisection
- Über das Thema kann man Vorlesungen füllen
- Klammern wir hier aus

Schritt 2

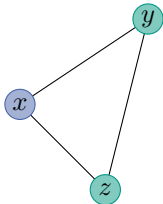
- Berechne CCH-Graph aus Ordnung und Eingabegraph
- Dies schnell machen ist nicht trivial
- Klammern wir hier auch aus

CCH: Customization

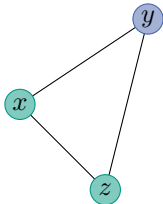




- **Unteres** Dreieck von $\{x, y\}$
- z wird als erstes kontrahiert



- Mittleres Dreieck von $\{z, y\}$
- z wird als erstes kontrahiert



- Oberes Dreieck von $\{z, x\}$
- z wird als erstes kontrahiert

- Anfragen sind korrekt, wenn untere Dreiecksungleichung gilt.
- Für jedes untere Dreieck $\{x, y, z\}$ von $\{x, y\}$ muss gelten

$$w(x, z) + w(z, y) \geq w(x, y)$$

- Ziel der Customization: Stelle untere Dreiecksungleichung her

Schritt 1

- Kanten, die im Eingabegraphen sind, kriegen ihr Gewicht zugewiesen
- Andere Kanten im CCH-Graph bekommen Gewicht ∞

Schritt 2

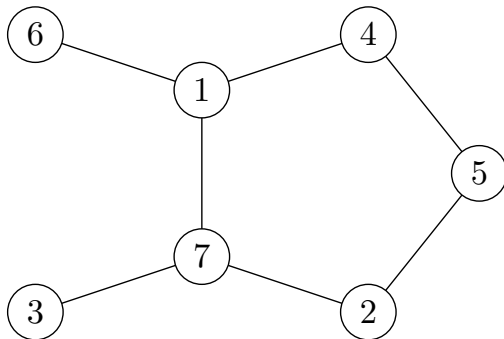
for *alle Knoten x von unten nach oben* **do**

for *alle Aufwärtskanten $\{x, y\}$ im CCH-Graph* **do**

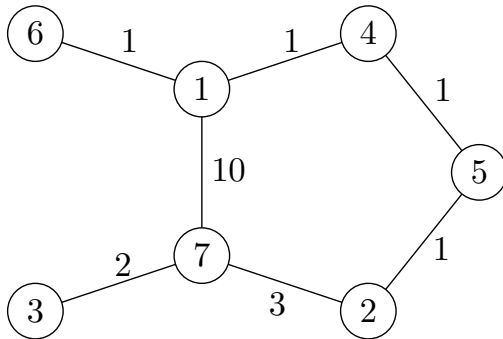
for *alle unteren Dreiecke $\{x, y, z\}$ von $\{x, y\}$* **do**

$w(x, y) := \min\{w(x, y), w(x, z) + w(z, y)\};$

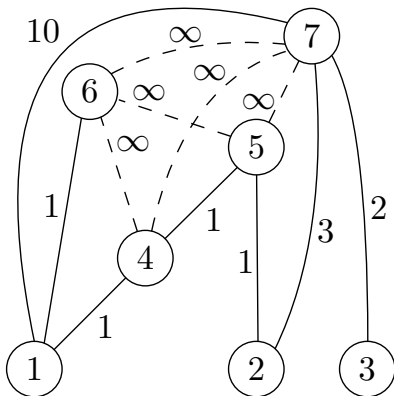
(Basic) Customization



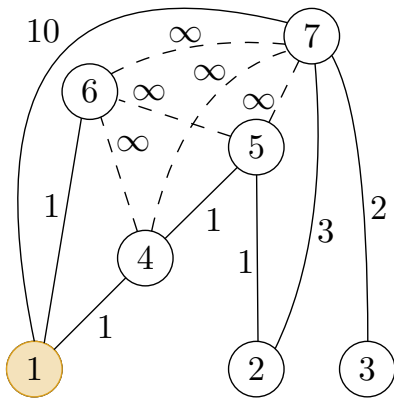
(Basic) Customization



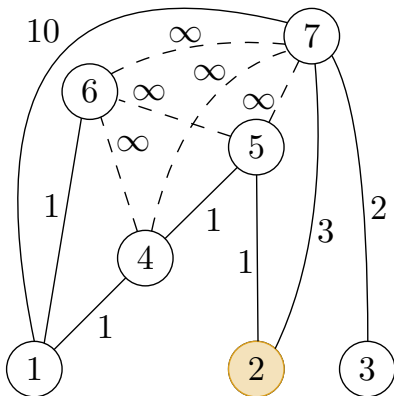
(Basic) Customization



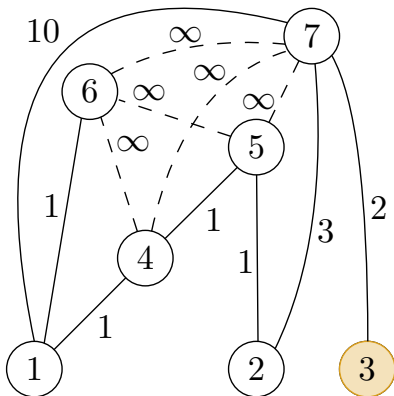
(Basic) Customization



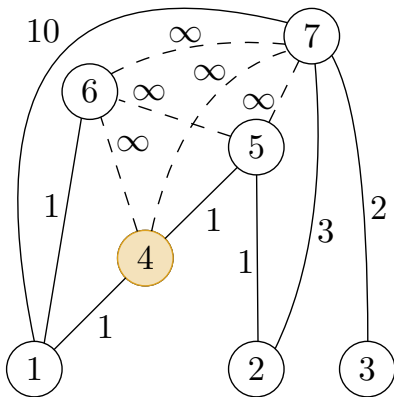
(Basic) Customization



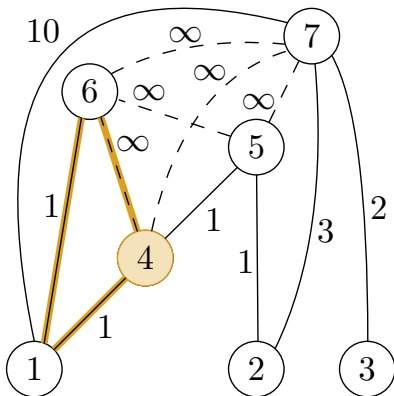
(Basic) Customization



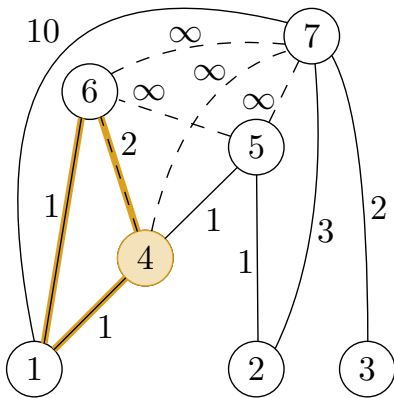
(Basic) Customization



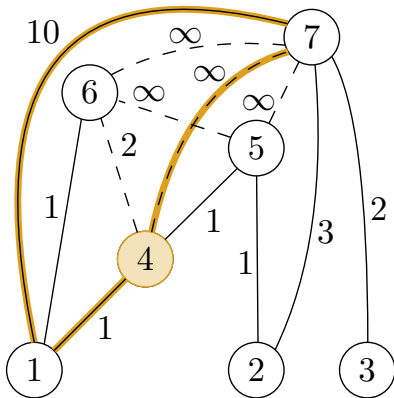
(Basic) Customization



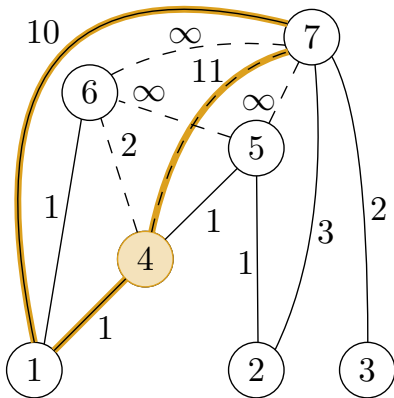
(Basic) Customization



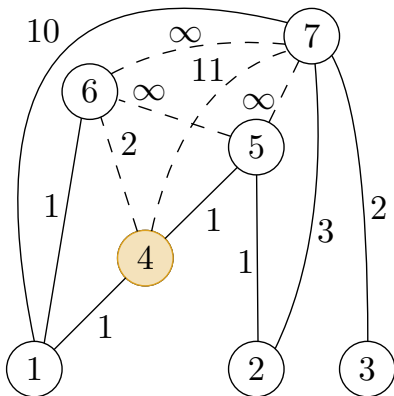
(Basic) Customization



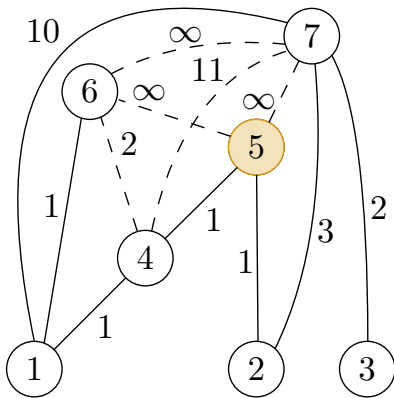
(Basic) Customization



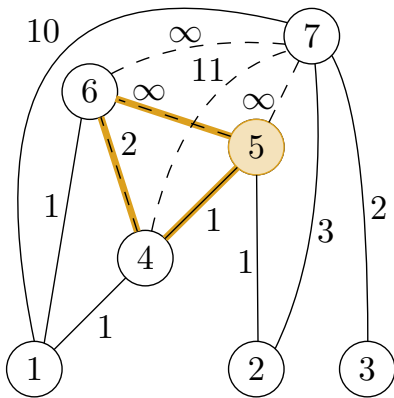
(Basic) Customization



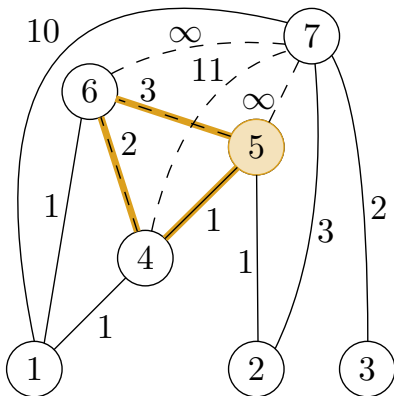
(Basic) Customization



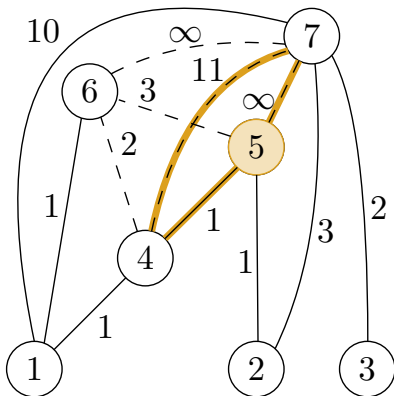
(Basic) Customization



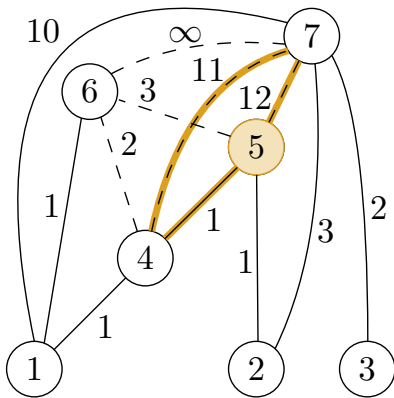
(Basic) Customization



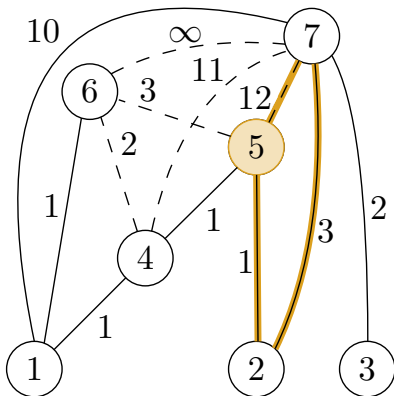
(Basic) Customization



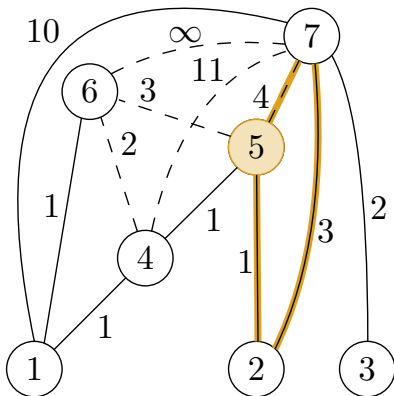
(Basic) Customization



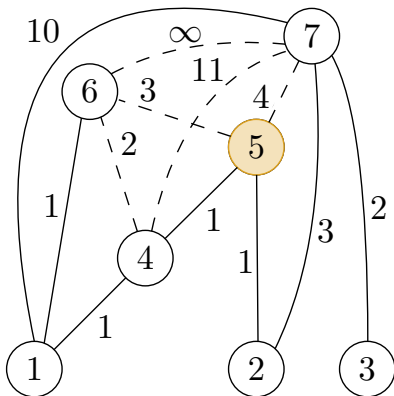
(Basic) Customization



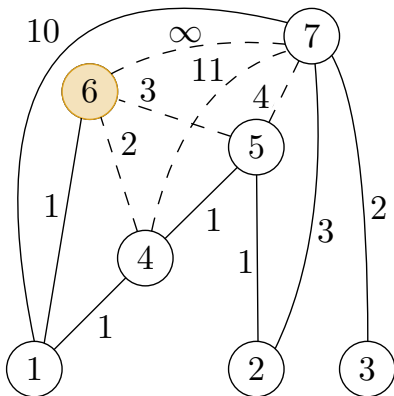
(Basic) Customization



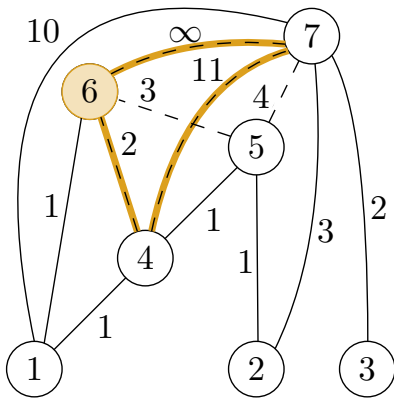
(Basic) Customization



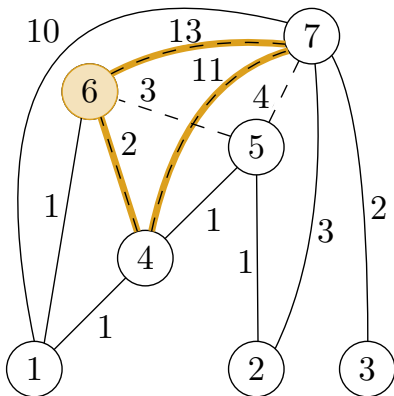
(Basic) Customization



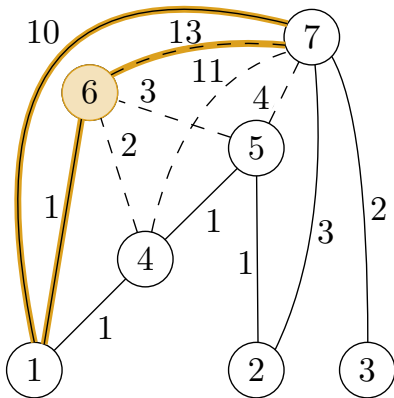
(Basic) Customization



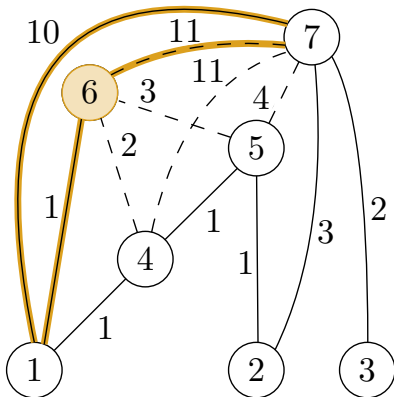
(Basic) Customization



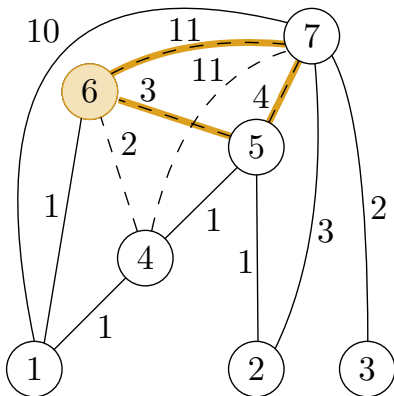
(Basic) Customization



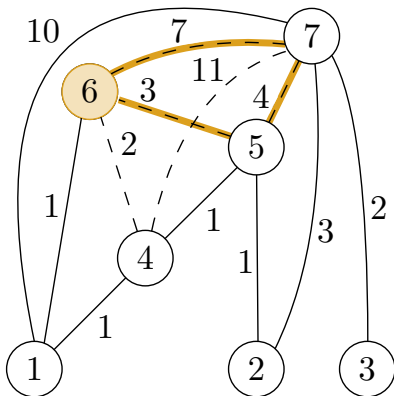
(Basic) Customization



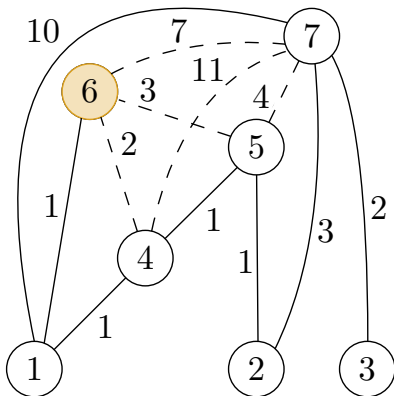
(Basic) Customization



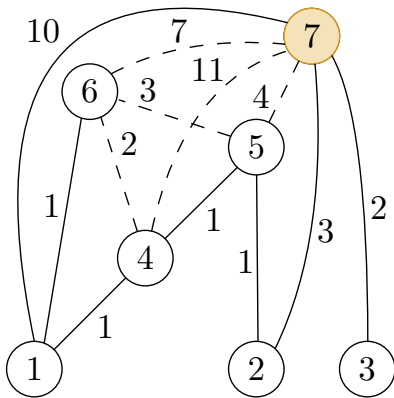
(Basic) Customization



(Basic) Customization



(Basic) Customization



Option 1: Alle Dreiecke einer Kante vorberechnen und speichern

Problem: Viel Speicher

Option 2:

- Speichere die Abwärtsnachbarschaften $N_d(x)$ jedes Knoten x als sortiertes Array
- Genau für jeden Knoten $z \in N_d(x) \cap N_d(y)$ gibt es ein unteres Dreieck $\{x, y, z\}$ von (x, y)
- $N_d(x) \cap N_d(y)$ kann man durch einen simultanen Scan über $N_d(x)$ und $N_d(y)$ berechnen

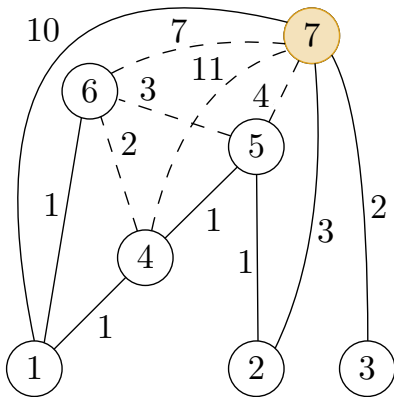
Problem: Langsamer als Option 1

- Customization mit perfekter Zeugensuche möglich
- Idee:
 - Iteriere über mittlere und obere Dreiecke
 - Von oben nach unten
 - Kanten deren Gewicht sich ändert können gelöscht werden

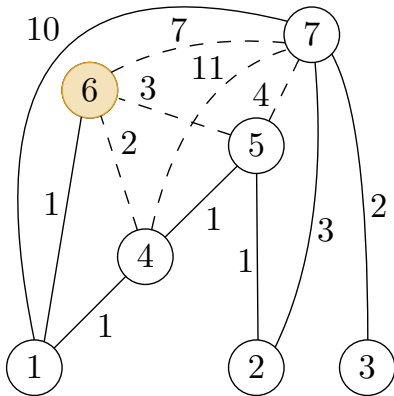
Algorithmus:

```
for alle Knoten  $x$  von oben nach unten do  
  for alle Aufwärtskanten  $\{x, y\}$  im CCH-Graph do  
    for alle oberen und mittleren Dreiecke  $\{x, y, z\}$  von  $\{x, y\}$  do  
       $w(x, y) := \min\{w(x, y), w(x, z) + w(z, y)\};$   
Löschen Kanten deren Gewicht sich verändert hat;
```

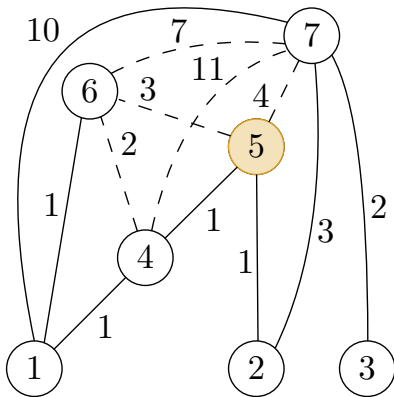
Perfekte Customization

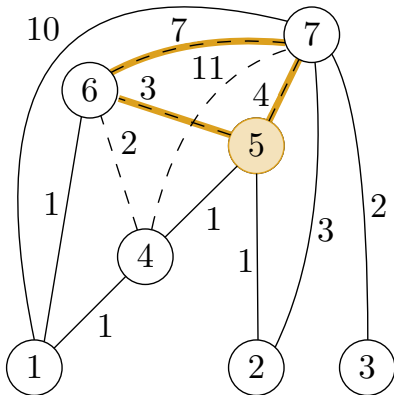


Perfekte Customization



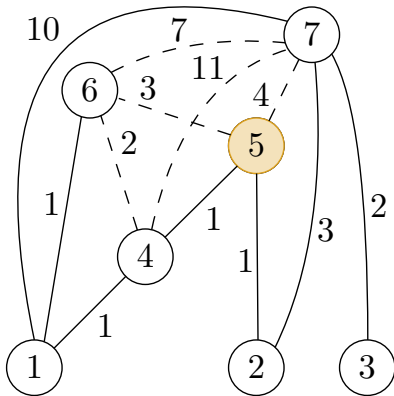
Perfekte Customization



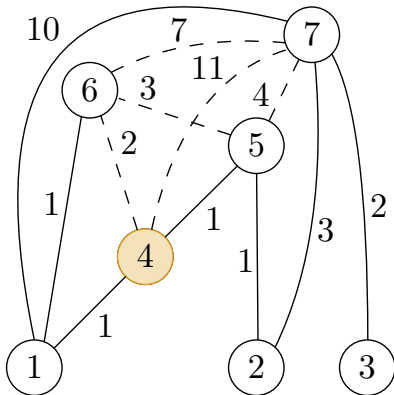


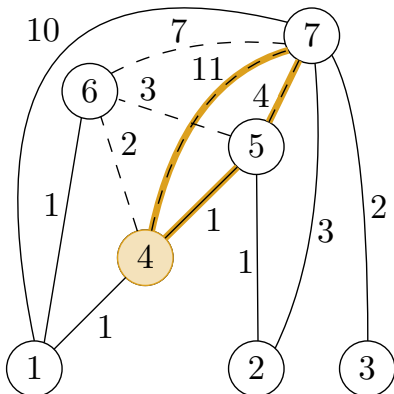
$\{5,6,7\}$ ist ein oberes Dreieck von $\{5,6\}$
 $\{5,6,7\}$ ist ein mittleres Dreieck von $\{5,7\}$
Gewichte unverändert

Perfekte Customization

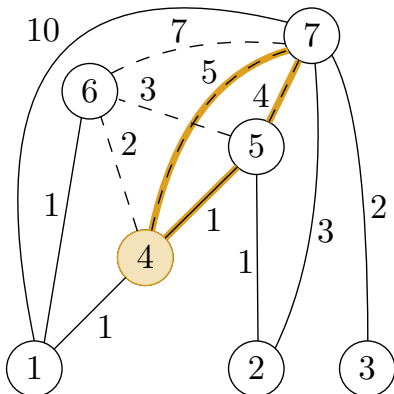


Perfekte Customization

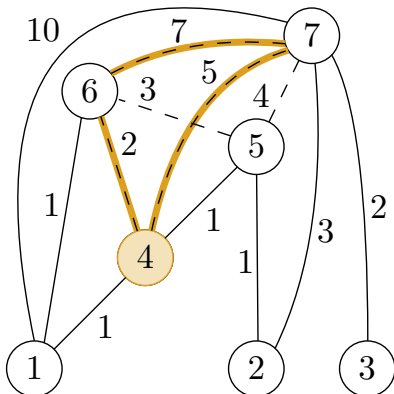




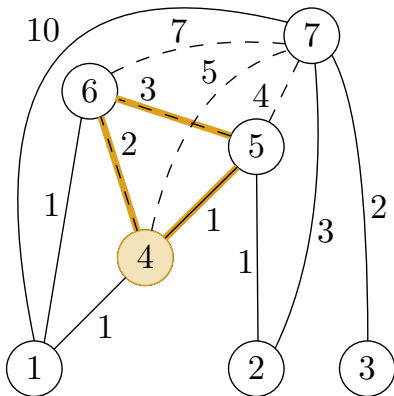
$\{4,5,7\}$ ist ein oberes Dreieck von $\{4,5\}$
 $\{4,5,7\}$ ist ein mittleres Dreieck von $\{4,7\}$
Gewichte von $\{4,7\}$ verändert sich



$\{4,5,7\}$ ist ein oberes Dreieck von $\{4,5\}$
 $\{4,5,7\}$ ist ein mittleres Dreieck von $\{4,7\}$
Gewichte von $\{4,7\}$ verändert sich

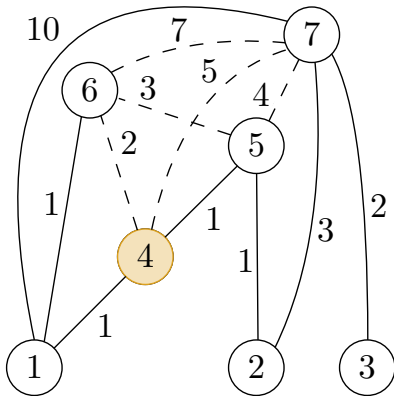


$\{4,6,7\}$ ist ein oberes Dreieck von $\{4,6\}$
 $\{4,6,7\}$ ist ein mittleres Dreieck von $\{4,7\}$
Gewichte unverändert

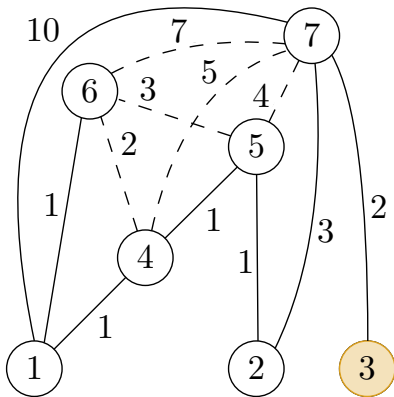


$\{4,5,6\}$ ist ein oberes Dreieck von $\{4,5\}$
 $\{4,5,6\}$ ist ein mittleres Dreieck von $\{4,6\}$
Gewichte unverändert

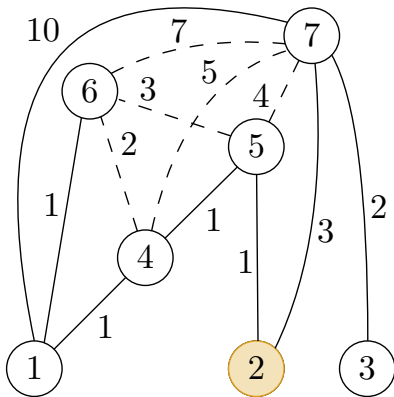
Perfekte Customization

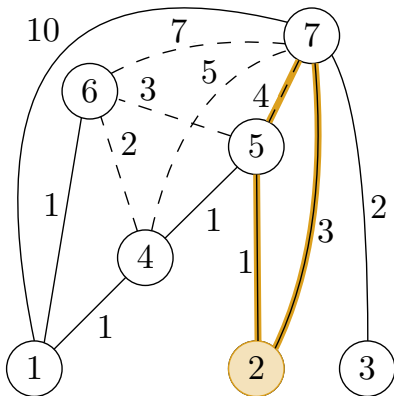


Perfekte Customization



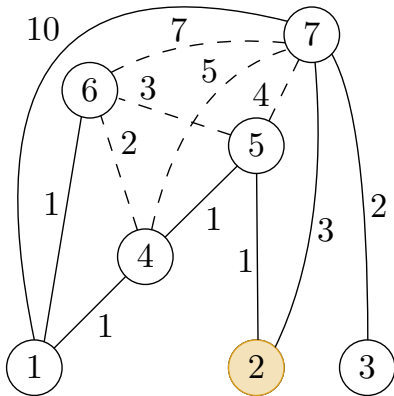
Perfekte Customization



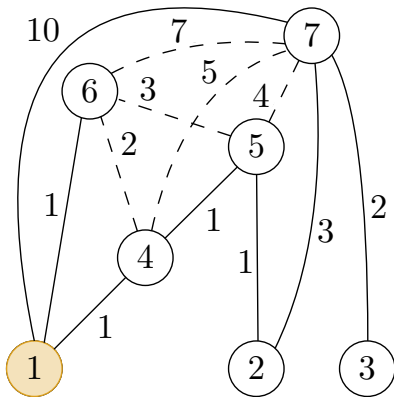


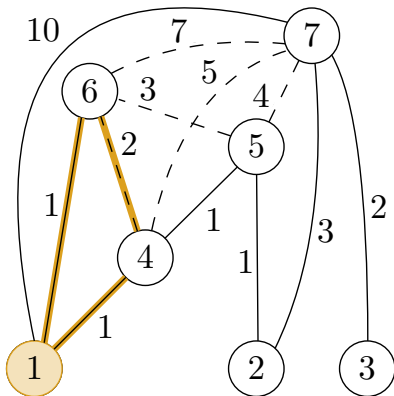
$\{2,5,7\}$ ist ein oberes Dreieck von $\{2,5\}$
 $\{2,5,7\}$ ist ein mittleres Dreieck von $\{2,7\}$
Gewichte unverändert

Perfekte Customization

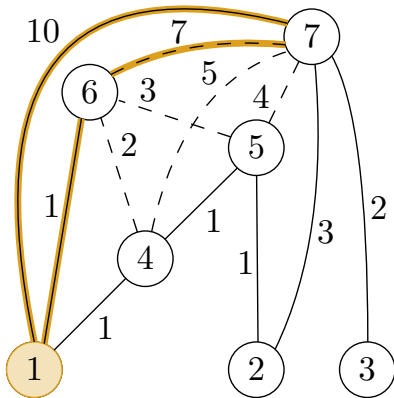


Perfekte Customization

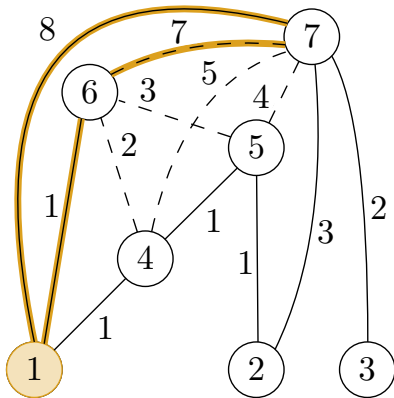




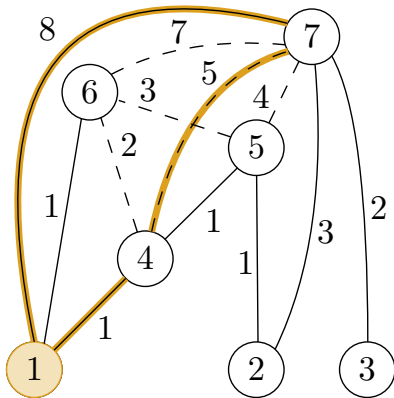
$\{1,4,6\}$ ist ein oberes Dreieck von $\{1,4\}$
 $\{1,4,6\}$ ist ein mittleres Dreieck von $\{1,6\}$
Gewichte unverändert



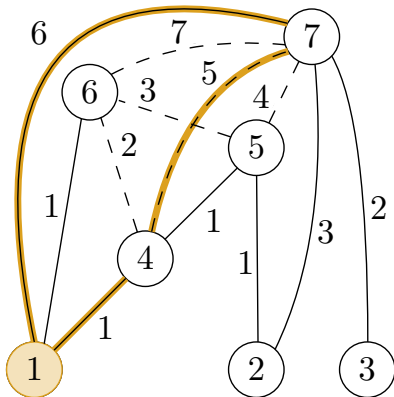
$\{1,6,7\}$ ist ein oberes Dreieck von $\{1,6\}$
 $\{1,6,7\}$ ist ein mittleres Dreieck von $\{1,7\}$
Gewichte von $\{1,7\}$ verändert sich



$\{1, 6, 7\}$ ist ein oberes Dreieck von $\{1, 6\}$
 $\{1, 6, 7\}$ ist ein mittleres Dreieck von $\{1, 7\}$
Gewichte von $\{1, 7\}$ verändert sich

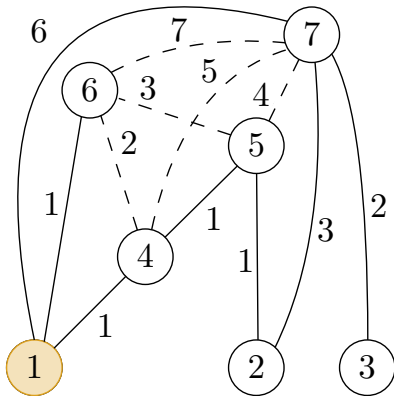


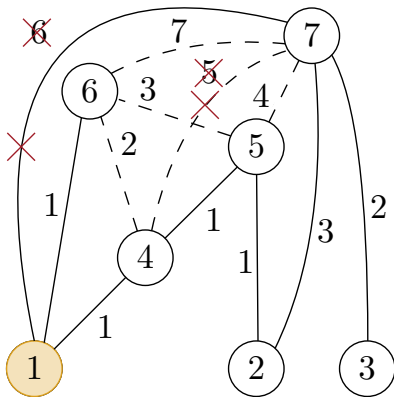
$\{1,4,7\}$ ist ein oberes Dreieck von $\{1,4\}$
 $\{1,4,7\}$ ist ein mittleres Dreieck von $\{1,7\}$
Gewichte von $\{1,7\}$ verändert sich



$\{1,4,7\}$ ist ein oberes Dreieck von $\{1,4\}$
 $\{1,4,7\}$ ist ein mittleres Dreieck von $\{1,7\}$
Gewichte von $\{1,7\}$ verändert sich

Perfekte Customization





Kanten $\{1, 7\}$ und $\{4, 7\}$ können gelöscht werden

Beobachtung

- Gelöschte Kanten waren keine kürzesten Pfade
- Teilpfade kürzester Hochrunterpfade sind kürzeste Pfade

Schlussfolgerung

- Kein kürzester Hochrunterpfade wurde zerstört

Perfekte Zeugensuche (Beweis nicht in der Vorlesung)

- Falls kürzeste Wege im Originalgraph eindeutig sind:
 - Keine weiteren Kanten können gelöscht werden
- Falls kürzeste Wege im Originalgraph nicht eindeutig sind:
 - Leicht komplizierter Algorithmus
 - Alle löschbaren Kanten werden gefunden

CCH: Query



Gewurzelter Baum

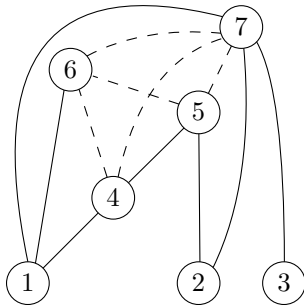
- Kann über parent-Funktion definiert werden
- Elimination-Tree wird durch Angabe von p definiert

Gewurzelter Baum

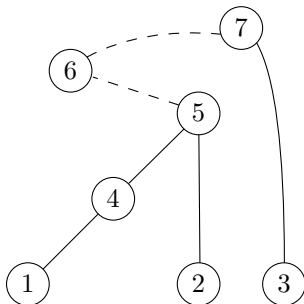
- Kann über parent-Funktion definiert werden
- Elimination-Tree wird durch Angabe von p definiert

Elimination-Tree

- Parent von Knoten v ist
 - Ein Nachbar von v im CCH-Graph
 - Höher als v
 - Unter allen verbleibenden Kandidaten der niedrigste
- (Randfall: bei nicht zusammenhängenden Graphen ist es ein Wald)



■ CCH-Graph



- CCH-Graph
- Elimination-Tree

Theorem

Vorfahren von x im Elimination-Tree = Knoten im Suchraum von x

Definition:

- $V(x)$ = Vorfahren von x inklusive x
- $S(x)$ = Knoten im Suchraum von x inklusive x
- $p(x)$ der Parent von x

zu zeigen:

- $V(x) \subseteq S(x)$
- $S(x) \subseteq V(x)$

$$V(x) \subseteq S(x)$$

- Sei $y \in V(x)$
- Zu zeigen: $y \in S(x)$
- Zu zeigen: Es gibt xy -Pfad im CCH-Graph, der nur hoch geht

$$V(x) \subseteq S(x)$$

- Sei $y \in V(x)$
- Zu zeigen: $y \in S(x)$
- Zu zeigen: Es gibt xy -Pfad im CCH-Graph, der nur hoch geht

- Da $y \in V(x)$, gibt es Pfad:

$$x \rightarrow p(x) \rightarrow p(p(x)) \rightarrow p(p(p(x))) \rightarrow \dots \rightarrow y$$

- Damit gibt es gesuchten xy -Pfad
- Damit gilt $y \in S(x)$

$$S(x) \subseteq V(x)$$

- Beweis per Induktion und Widerspruch

Induktion

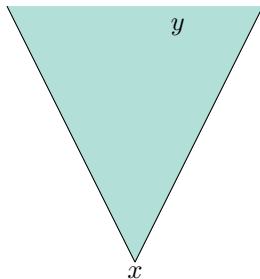
- Induktion über die Höhe eines Knoten x
- Induktionshypothese:
 - $S(x) = V(x)$
- Induktionsanfang:
 - Für die Wurzel r gilt $S(r) = \{r\} = V(r)$
- Induktionsvoraussetzung:
 - $S(p(x)) = V(p(x))$
- Induktionsschluss:
 - Zu zeigen: $S(x) = V(x)$
 - Wir wissen, dass $V(x) \subseteq S(x)$
 - Es reicht $S(x) \subseteq V(x)$ zu zeigen

$$S(x) \subseteq V(x)$$

- $S(x) \subseteq V(x)$ gilt genau dann, wenn $S(x) \setminus V(x) = \emptyset$
- Sei $y \in S(x) \setminus V(x)$, d. h. $y \in S(x)$ und $y \notin V(x)$
- Wir zeigen, per Widerspruch, dass es y nicht geben kann

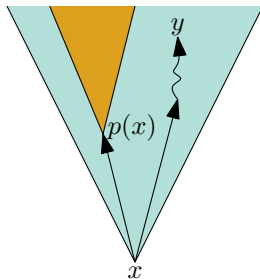
$$S(x) \subseteq V(x)$$

$$S(x) \subseteq V(x)$$



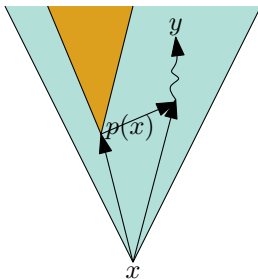
- Der Suchraum $S(x)$ von x in türkis
- Da $y \in S(x)$, ist y im türkisen Bereich

$$S(x) \subseteq V(x)$$



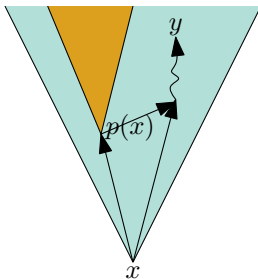
- Der Suchraum $S(p(x))$ von $p(x)$ in orange
- Da $y \notin V(x)$
- und $V(x) = \{x\} \cup V(p(x))$
- und nach IV $V(p(x)) = S(p(x))$
- ist y nicht im orangenen Bereich

$$S(x) \subseteq V(x)$$



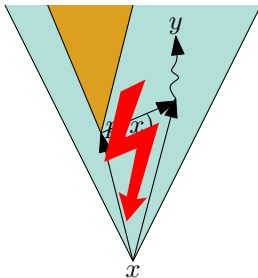
- Nach Kontraktion von x muss es diese Kante geben

$$S(x) \subseteq V(x)$$



- Es gibt Pfad $p(x) \rightarrow \dots \rightarrow y$
- y ist damit im Suchraum $S(p(x))$ von $p(x)$
- Nach IV ist $S(p(x)) = V(p(x))$, d. h. y ist Vorgänger von $p(x)$
- und wegen $V(x) = \{x\} \cup V(p(x))$, d. h. damit ist y auch Vorgänger von x
- Widerspruch zu $y \notin V(x)$

$$S(x) \subseteq V(x)$$



Solange man nicht an der Wurzel ist:

- Wenn s kleineren Rank hat als t :
 - Relaxiere ausgehende Kanten von s im Suchgraph
 - $s \leftarrow \text{parent}(s)$
- Else:
 - Relaxiere ausgehende Kanten von t im Suchgraph
 - $t \leftarrow \text{parent}(t)$

Vorteile:

- Keine Queue
- Funktioniert mit negativen Gewichten

Nachteile:

- Funktioniert nur mit metrikunabhängiger Ordnung
- Sucht immer den gesamten Suchraum ab

Algo:

- suche hoch-runter-Pfad
- für jeden Shortcut $x \rightarrow y$ im Pfad zähle alle unteren Dreiecke $\{x, y, z\}$ auf
- falls $w(x, y) = w(x, z) + w(z, y)$, dann ersetze $x \rightarrow y$ durch $x \rightarrow z \rightarrow y$
- wiederhole bis keine Shortcuts mehr im Pfad sind

- Anders als bei der CH gibt es nur einen Suchgraph bei der CCH
- Die CCH hat zwei Gewichte: ein Aufwärts- und ein Abwärtsgewicht
- Einbahnstraßen haben Gewicht ∞ in eine Richtung

Durchschnittliche Laufzeit von Reisezeitanfragen

■ CCH-Dijkstra :	0.81 ms
■ CCH-Stall :	0.85 ms
■ CCH-Tree :	0.41 ms
■ CCH-perfekt-Tree :	0.16 ms
■ CH-Dijkstra :	0.28 ms
■ CH-Stall :	0.11 ms

Durchschnittliche Laufzeit von Geodistanzanfragen

■ CCH-Dijkstra :	0.87 ms
■ CCH-Stall :	1.00 ms
■ CCH-Tree :	0.42 ms
■ CCH-perfekt-Tree :	0.21 ms
■ CH-Dijkstra :	2.66 ms
■ CH-Stall :	0.54 ms

(Basic) Customization

- Laufzeit: 0.4s
- Parallel mit 16 cores und SIMD/SSE
- Details in [DSW14]

Perfekte Customization

- Etwa 3 mal die Basic Customization

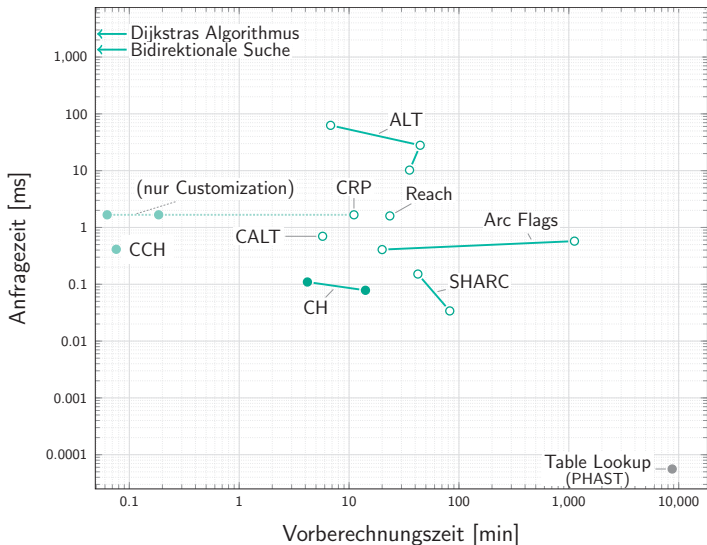
Nested Dissection Ordnung

- Hängt vom verwendeten Partitionierer ab
- KaHip gute Qualität: <2.8 days (nicht auf Schnelligkeit optimiert)
- FlowCutter gute Qualität: 4.6h
- InertialFlow brauchbare Qualität: 17min
- Metis Quick&Dirty: 2.2 min

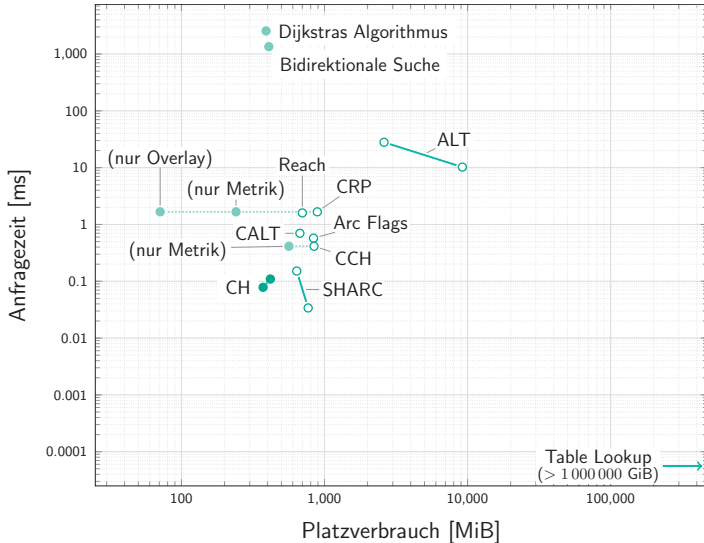
Metricabhängige Ordnungen

- Bottom-Up: 8-100 min (Variiert mit Gewichten und Details)
Parallel mit Reisezeit geht auch 2min
- Top-Down: 29.75 h (mit zusätzlichen Tricks aus [ADGW12])
- Sampling Path Greedy: 139.4 min (mit extra Tricks aus [DGPW14])

Overview



Overview



Mittwoch, 22.5.2019

Montag, 27.5.2019

Montag, 3.6.2019



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.



Reinhard Bauer, Tobias Columbus, Ignaz Rutter, and Dorothea Wagner.
Search-space size in contraction hierarchies.

In *Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP'13)*, volume 7965 of *Lecture Notes in Computer Science*, pages 93–104. Springer, 2013.



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.



Julian Dibbelt, Ben Strasser, and Dorothea Wagner.

Customizable contraction hierarchies.

In *Proceedings of the 13th International Symposium on Experimental Algorithms (SEA'14)*, volume 8504 of *Lecture Notes in Computer Science*, pages 271–282. Springer, 2014.



Julian Dibbelt, Ben Strasser, and Dorothea Wagner.

Customizable contraction hierarchies.

ACM Journal of Experimental Algorithmics, 21(1):1.5:1–1.5:49, April 2016.