Karlsruher Institut für Technologie

Algorithmen für Routenplanung

7. Vorlesung, Sommersemester 2016

Ben Strasser | 9. Mai 2014

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

Kürzeste Wege in Straßennetzwerken

Beschleunigungstechniken (Fortsetzung)

Thema: Contraction Hierarchies (CH)

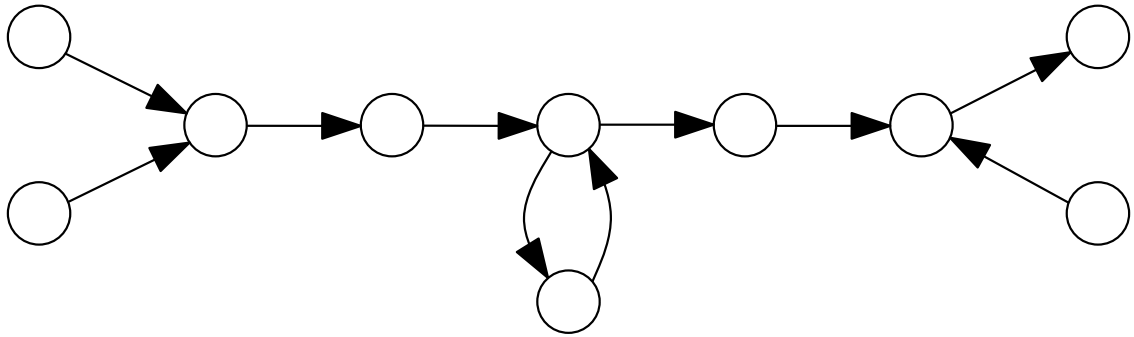
- CH Basisvariante
- CH im Detail
- Customizable Contraction Hierarchies (CCH)
- Exkurs: Dünnbesetzte Gleichungssysteme

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

CH Basisvariante

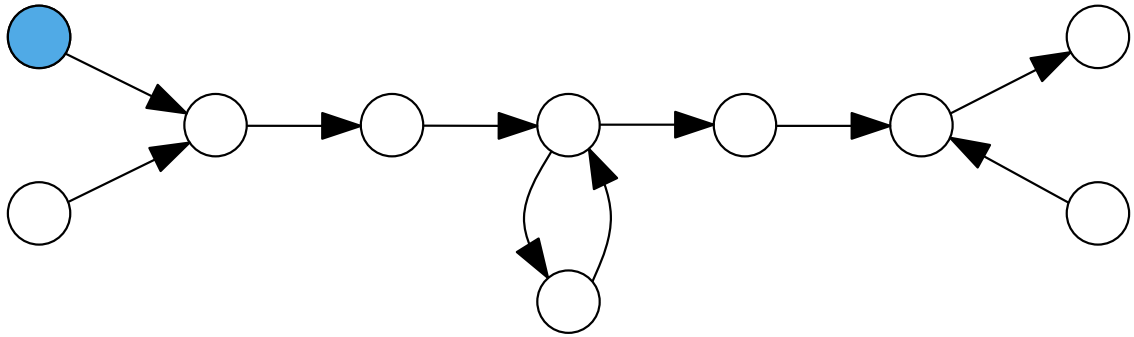
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Shortcut



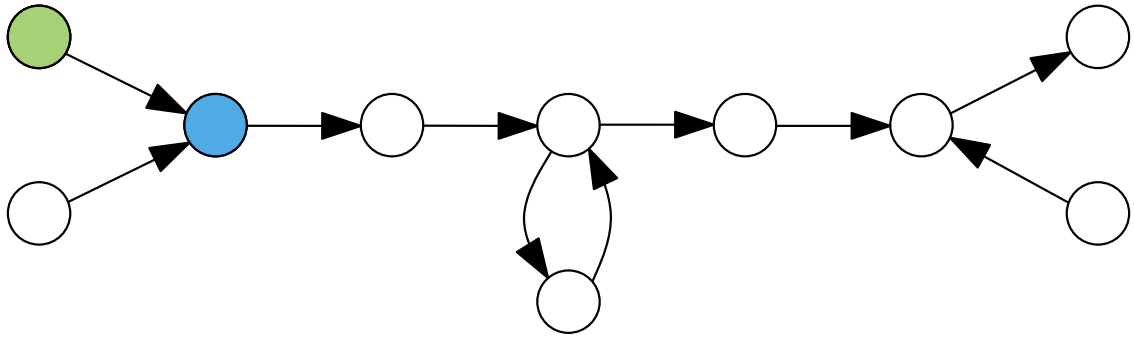
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



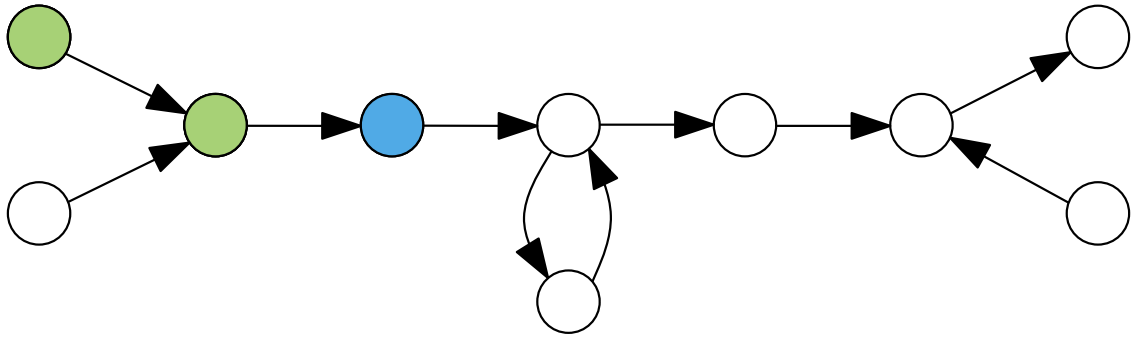
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



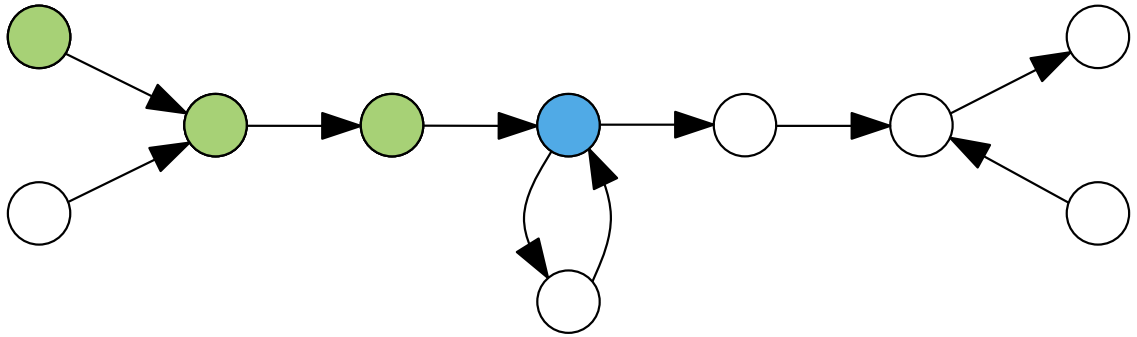
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



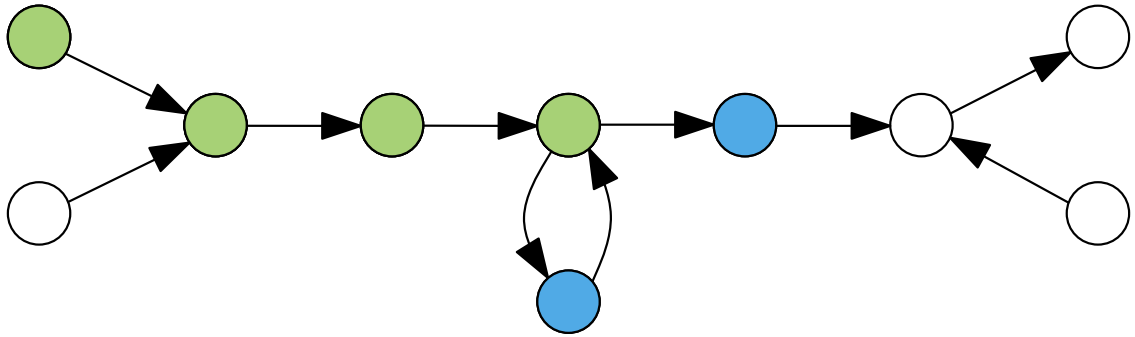
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



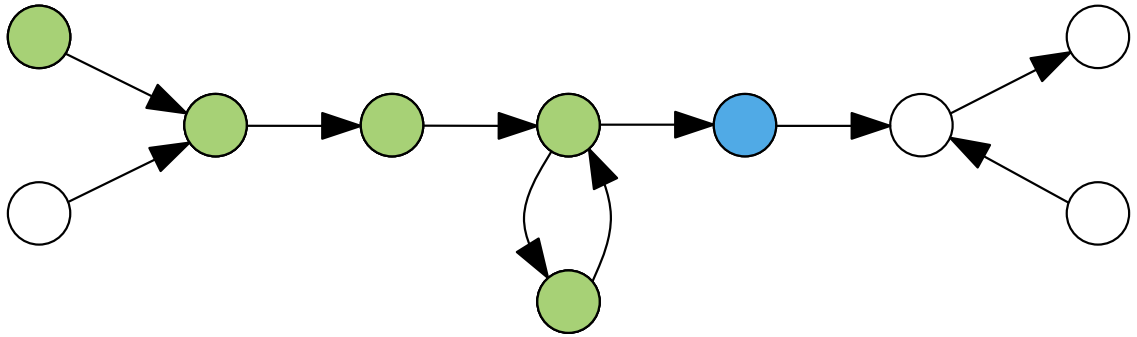
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



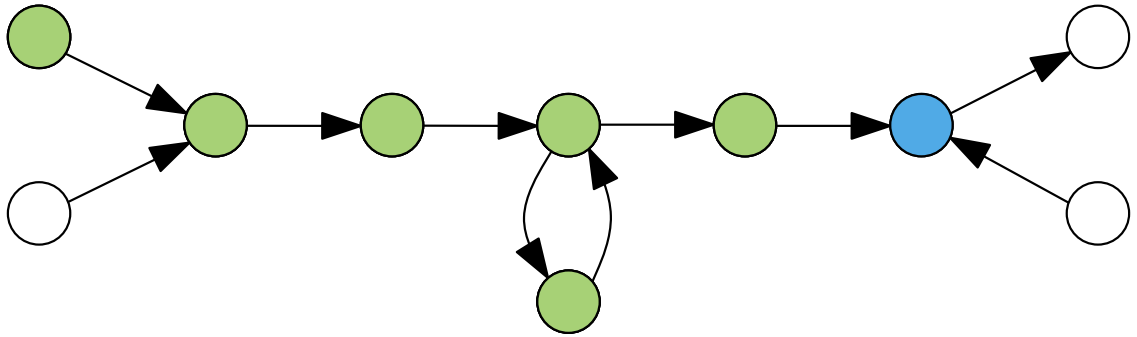
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



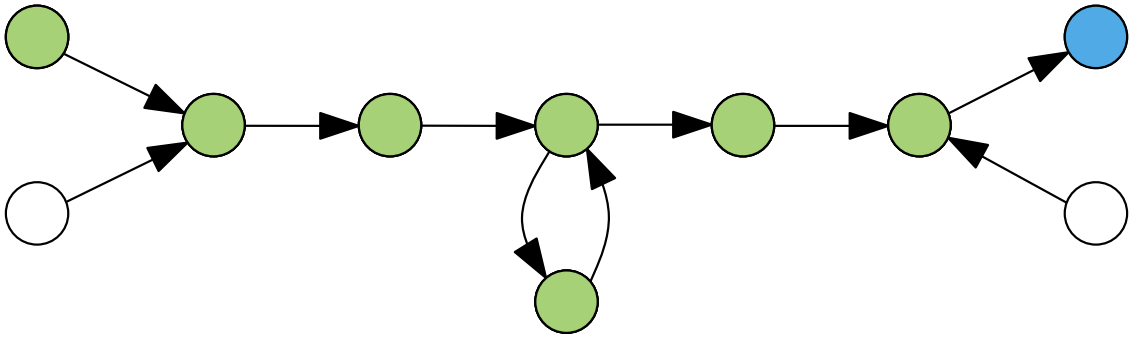
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



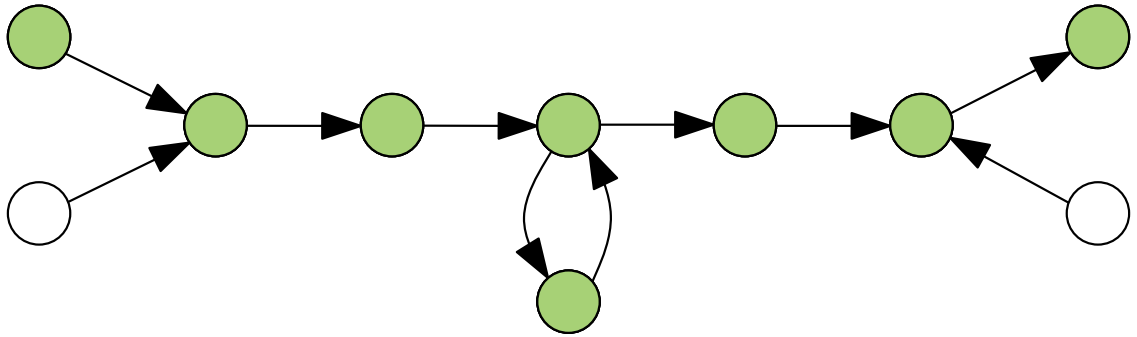
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



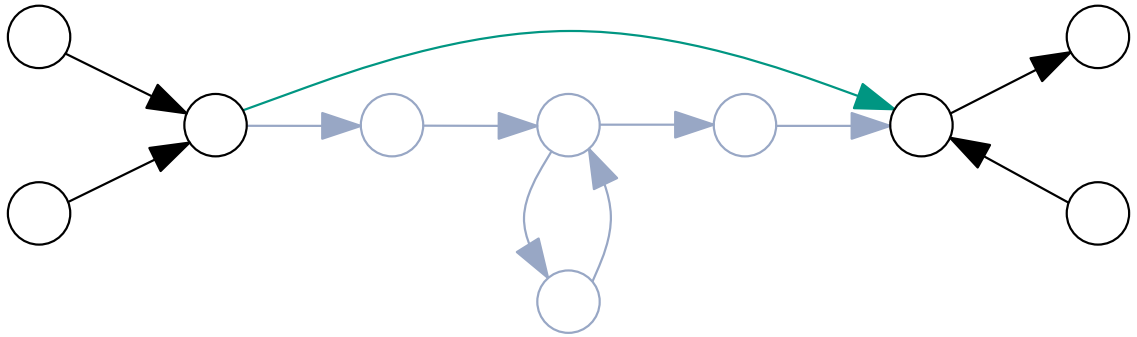
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



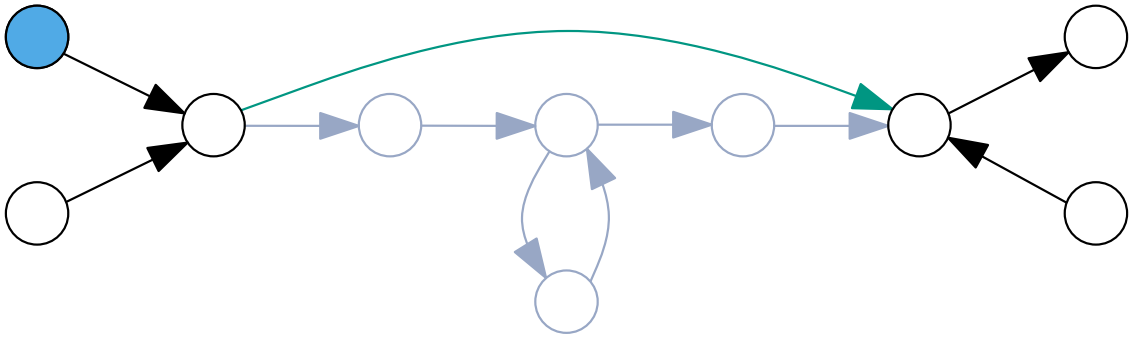
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.

Shortcut



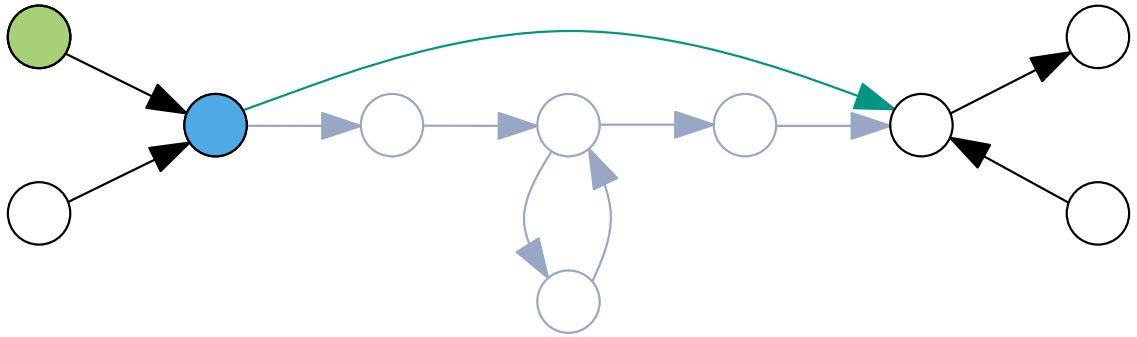
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



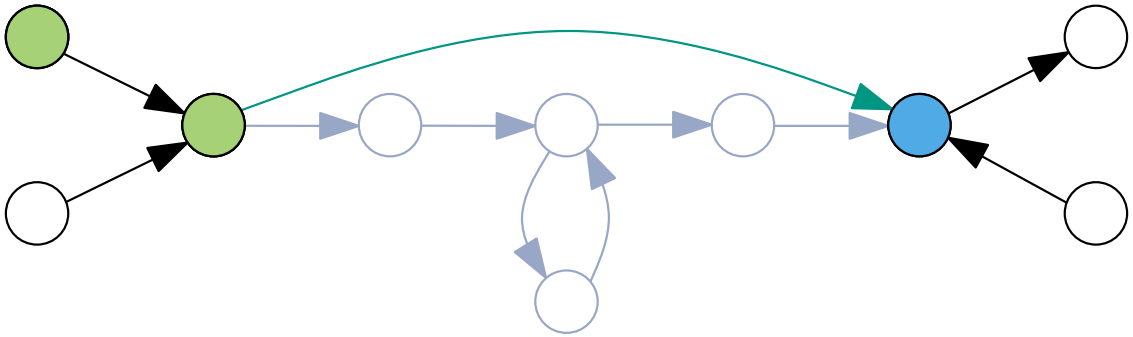
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



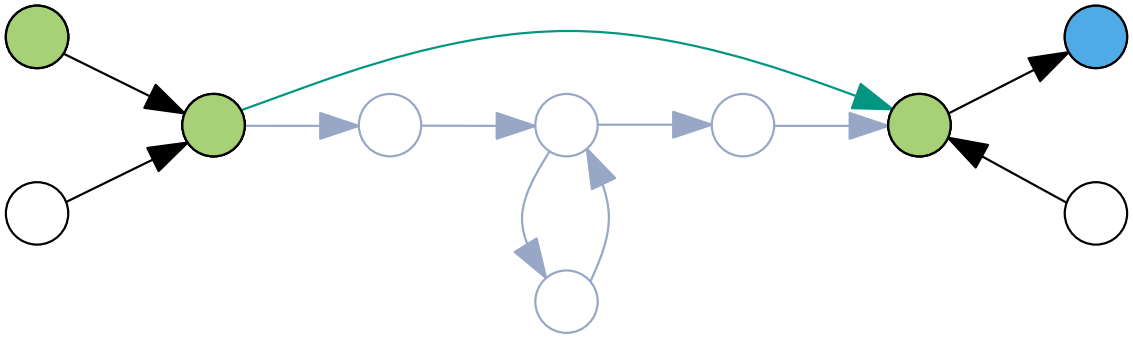
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



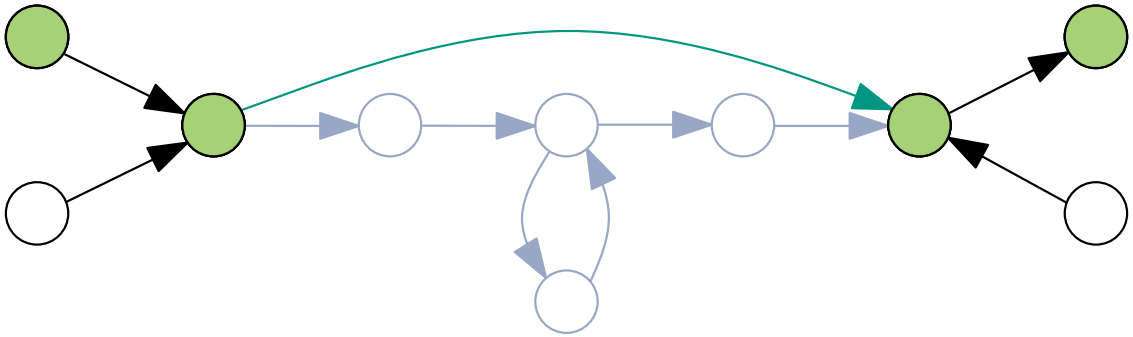
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



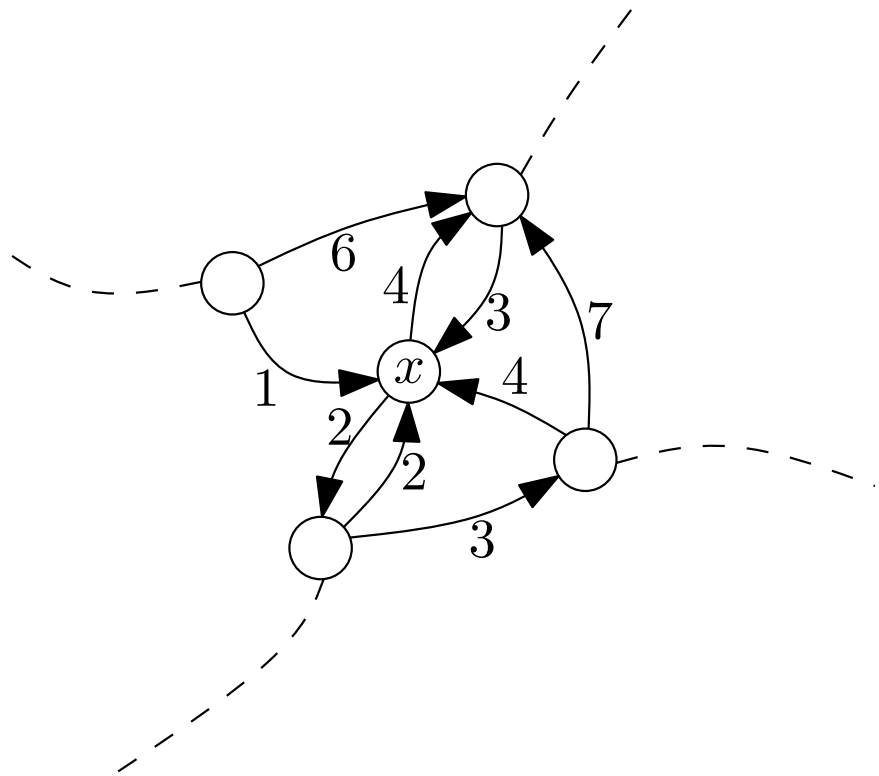
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Shortcut



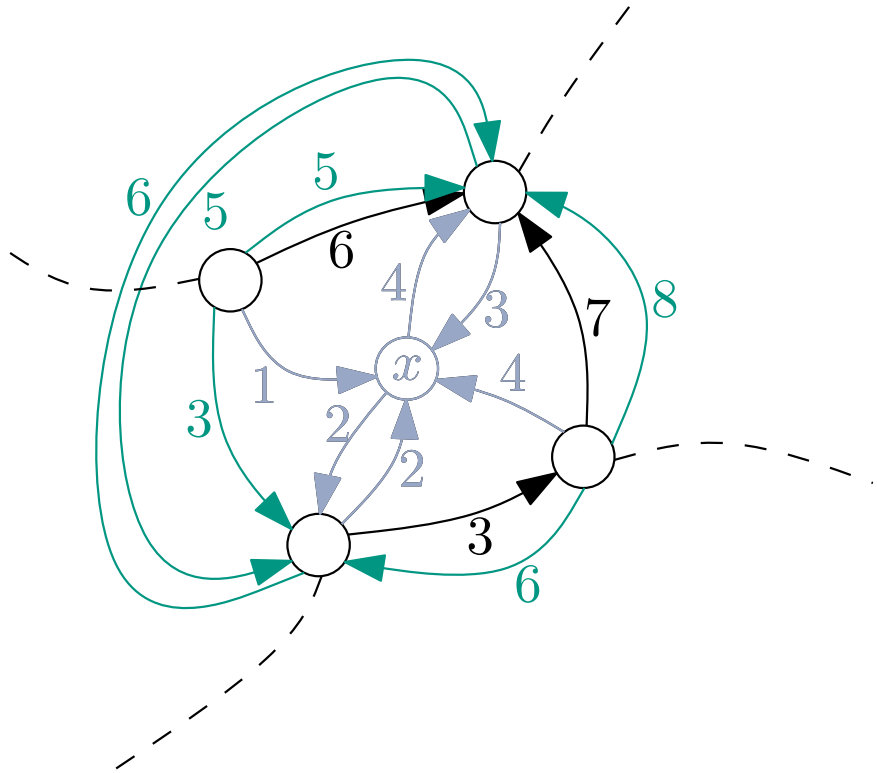
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn *s* oder *t* drin liegt.

Knotenkontraktion von x



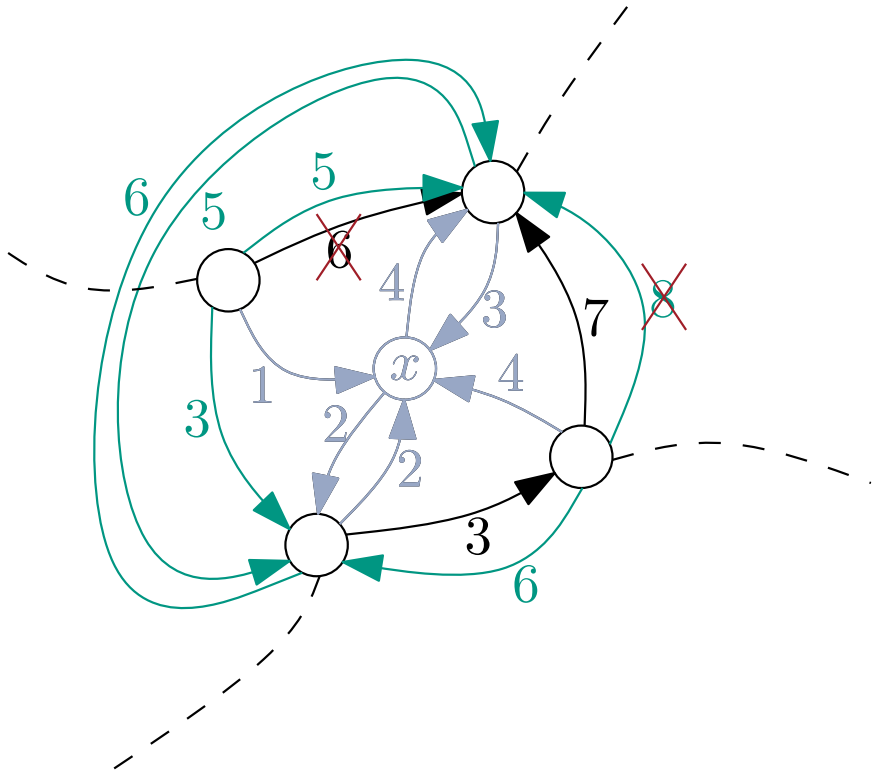
Kontraktion von x: Lösche x und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

Knotenkontraktion von x



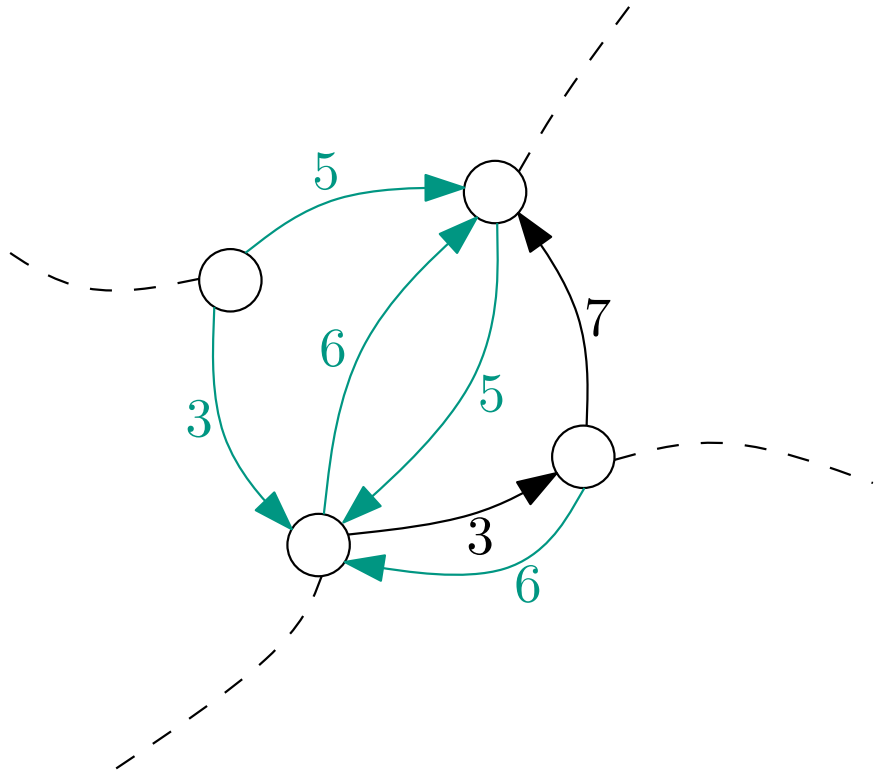
Kontraktion von x: Lösche x und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

Knotenkontraktion von x

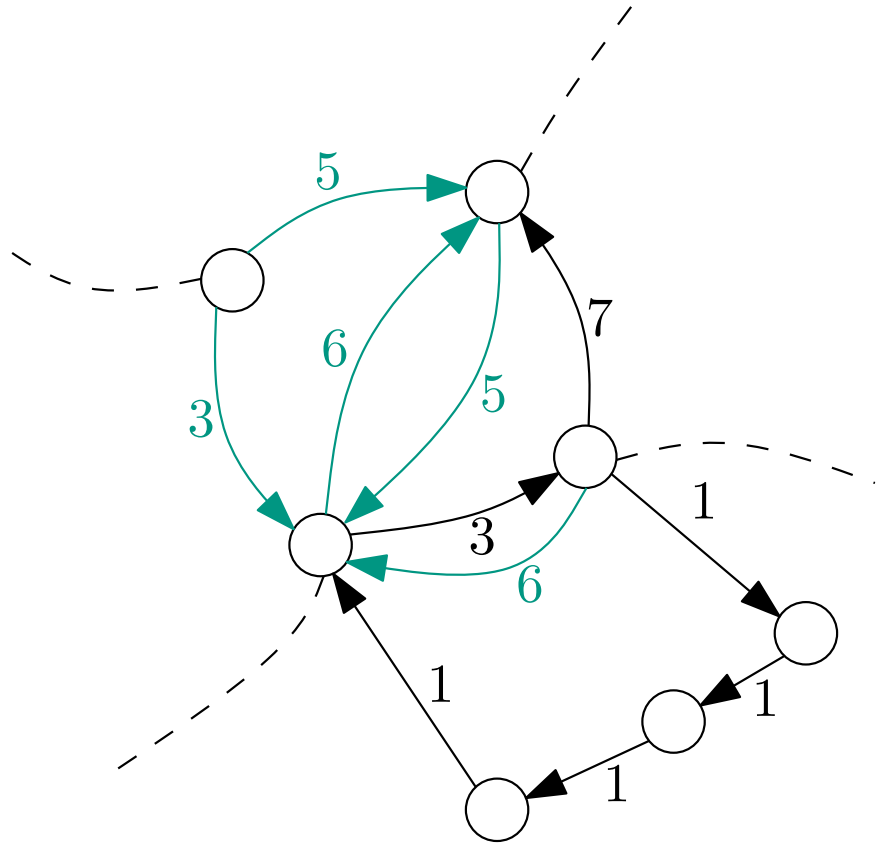


Bei Mehrfachkanten: Längere Kanten verwerfen

Knotenkontraktion von x

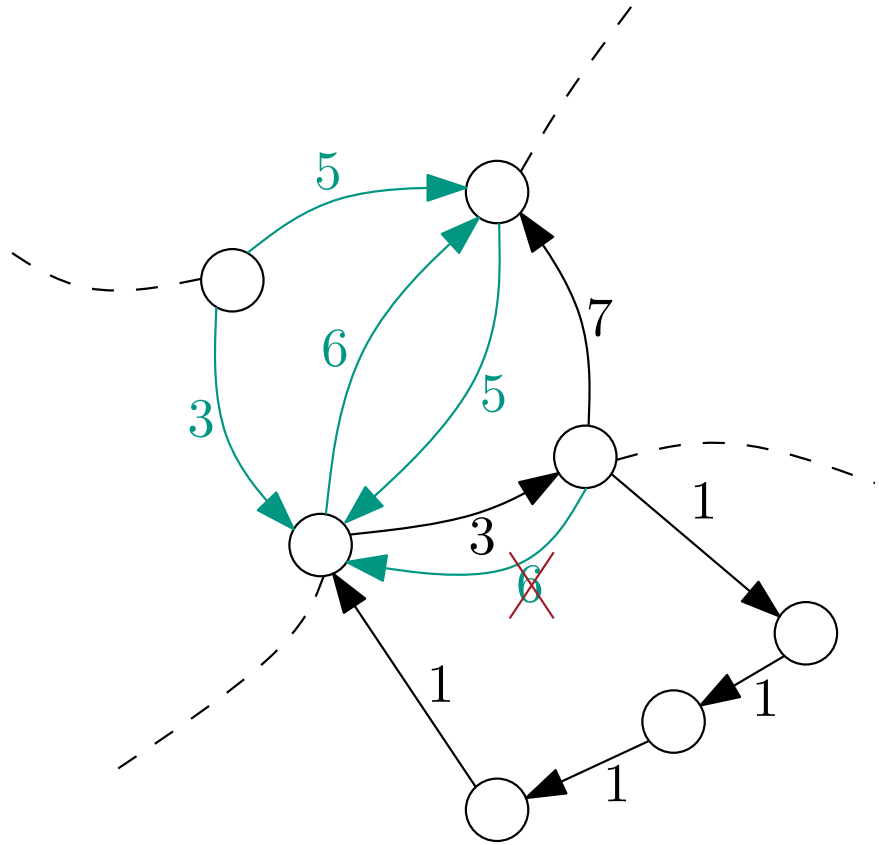


Knotenkontraktion von x



Falls es einen kürzeren Pfad durch den Restgraphen gibt, dann kann man einen Shortcut auch verwerfen.
Suche nach solchem Pfad heißt Zeugensuche/Witness Search

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

Zeugensuche

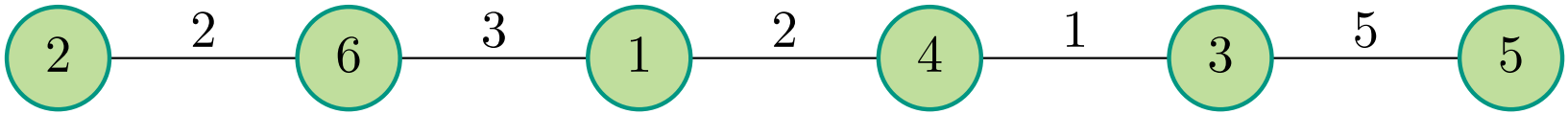
- Es seien y und z zwei Nachbarn des kontrahierten Knoten x
- Wir fügen ein Shortcut (y, z) mit Gewicht $\text{len}(y, x) + \text{len}(x, z)$ ein wenn $y \to x \to z$ ein eindeutiger kürzester Weg ist
- Zum Überprüfen ob es ein kürzer Weg gibt startet man einen Dijkstra von y aus nach z. Diese Suche kann teuer sein. Mögliche Optimierungen:
  - Suche darf nicht über den Knoten x gehen
  - Bidirektionale Variante von Dijkstras Algorithmus
  - Wenn die Suchen sich treffen kann man abbrechen
  - Wenn die Suchfront größer wird als $\text{len}(y, x) + \text{len}(x, z)$ kann man abbrechen
- Wenn das immer noch zu langsam ist: Suche nach k Schritten abbrechen. Eventuell gibt es einen Pfad den wir nicht finden. Das führt zu zusätzlichen Shortcuts, aber das ist kein Problem mit der Korrektheit.

Contraction Hierarchy

Grundidee

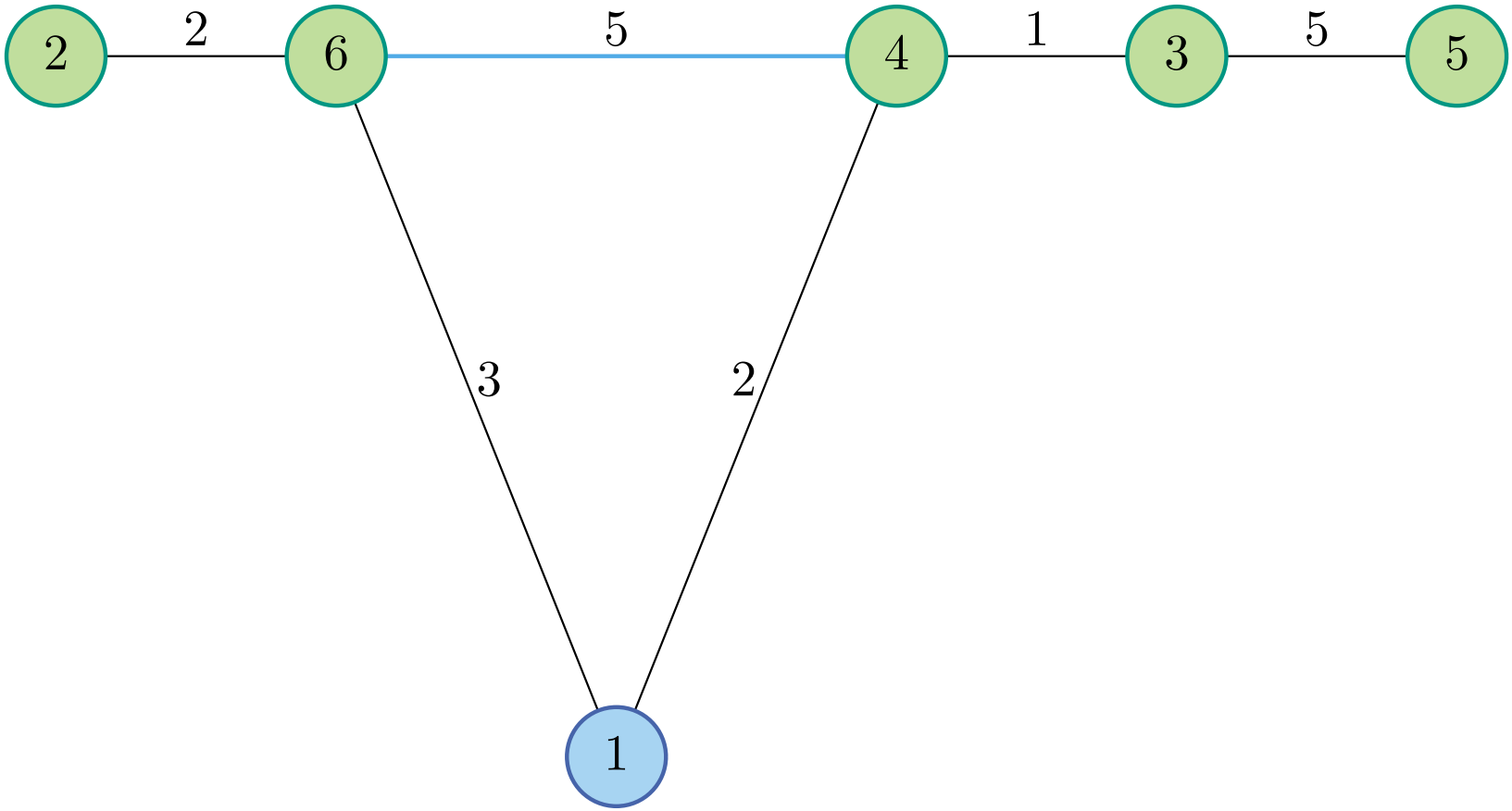
- Eingabe Graph G
- Ordne Knoten von G nach “Wichtigkeit”: $v_1 \ldots v_n$
- Kontrahiere Knoten iterative aus G raus
  - zuerst den “unwichtigsten” Knoten v_1
  - den “wichtigsten” Knoten v_n als letztes
- Graph mit Shortcuts heißt augmentierter Graph

Contraction Hierarchy



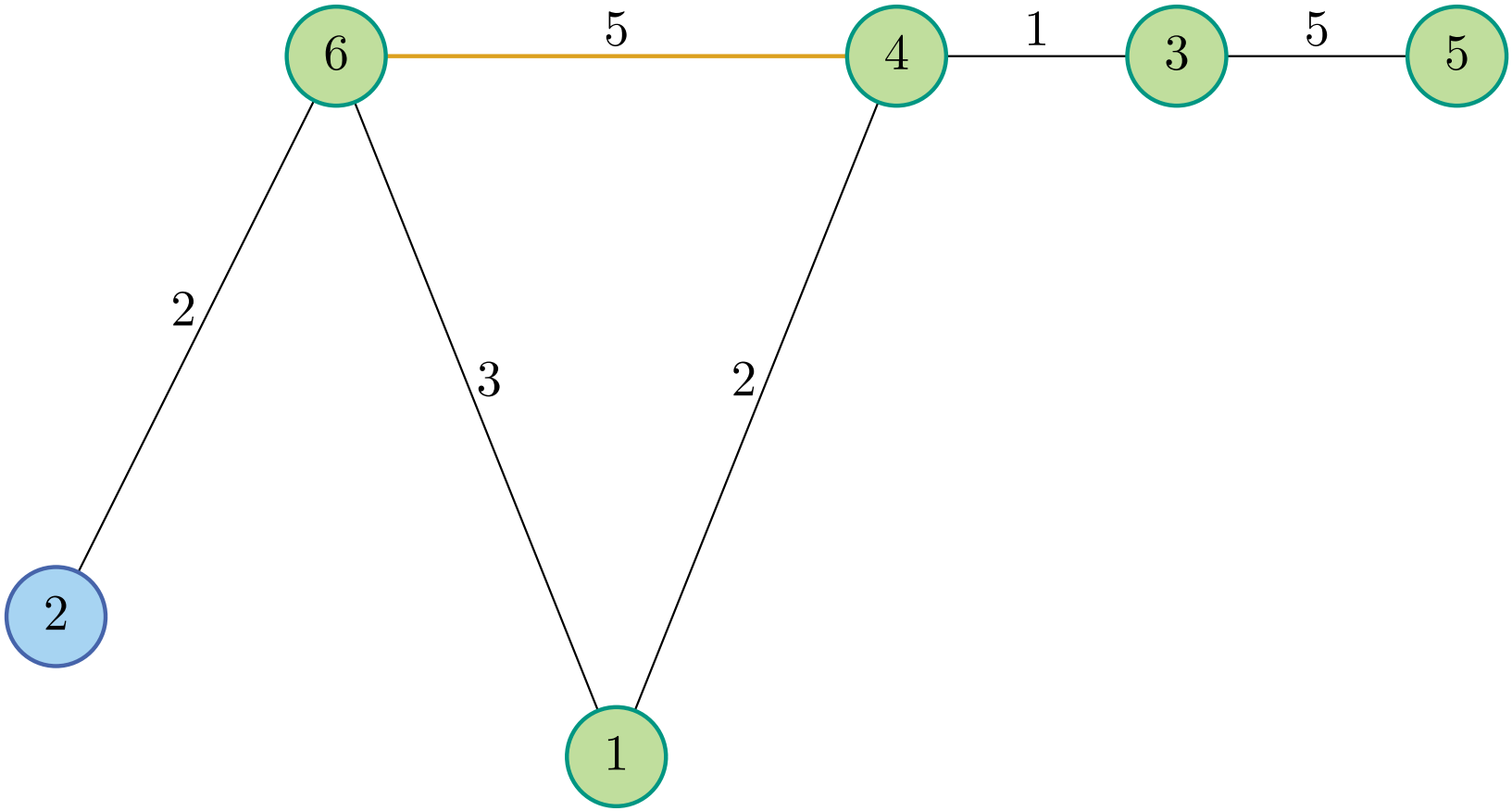
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



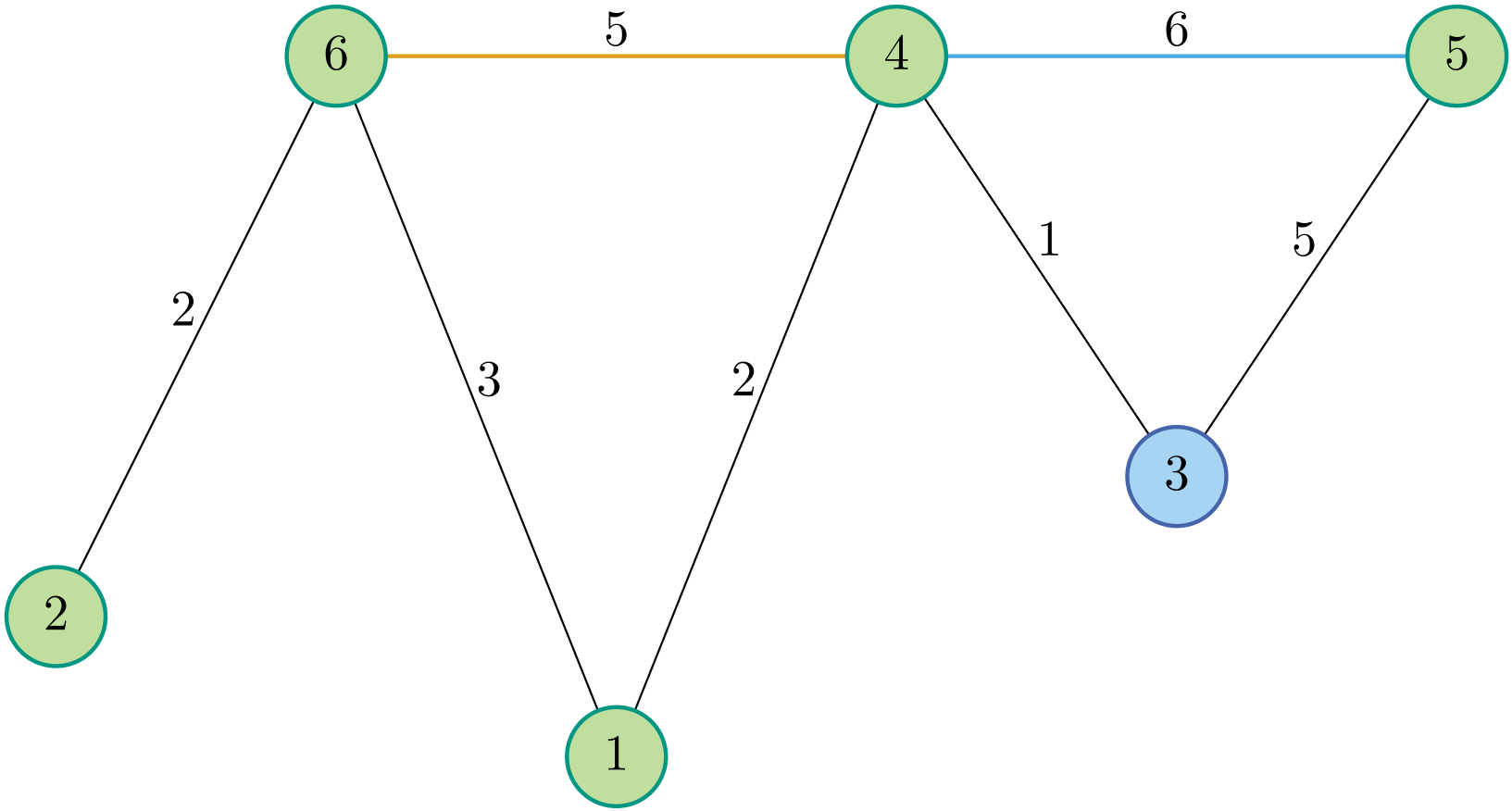
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



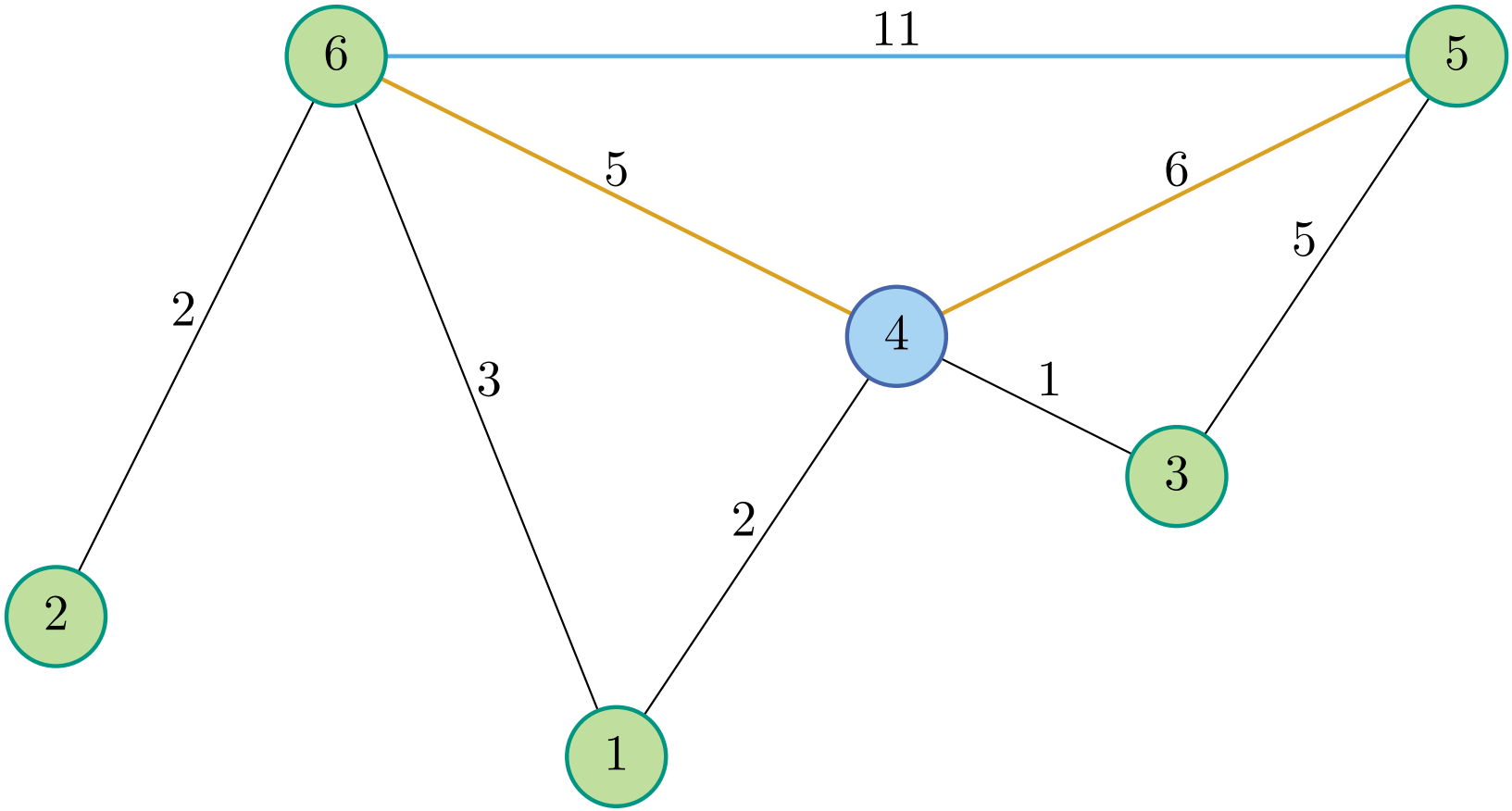
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



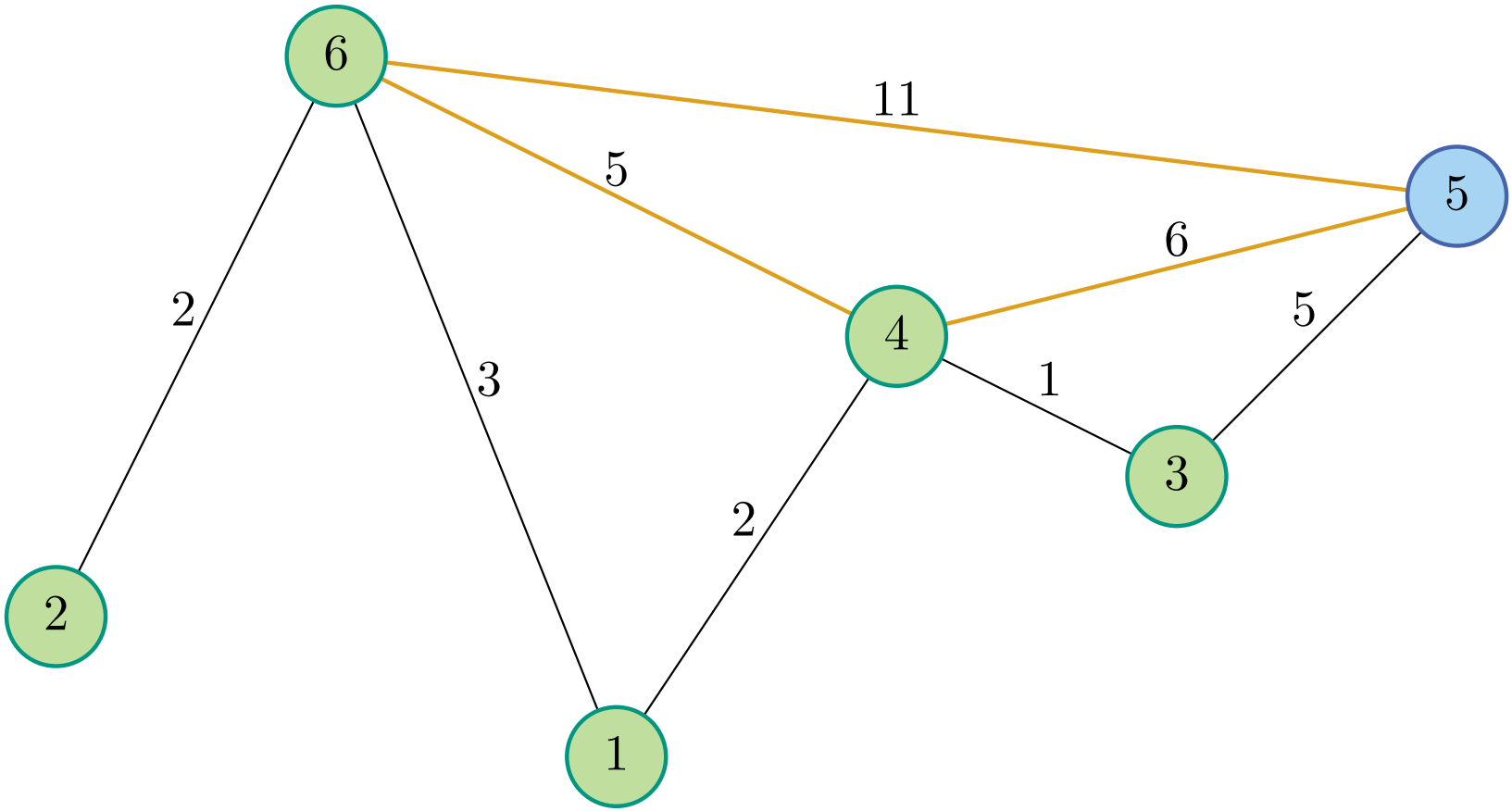
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



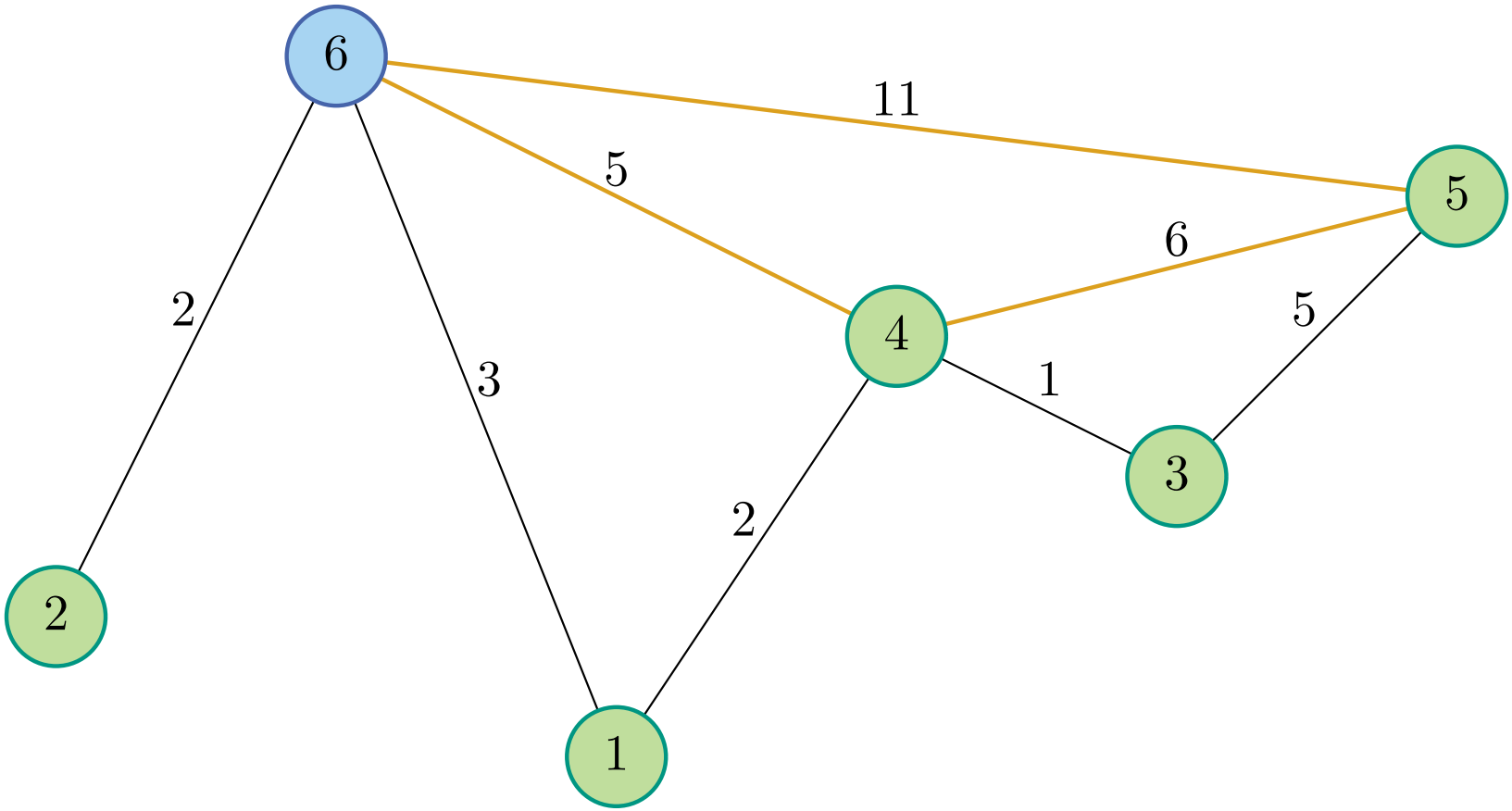
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



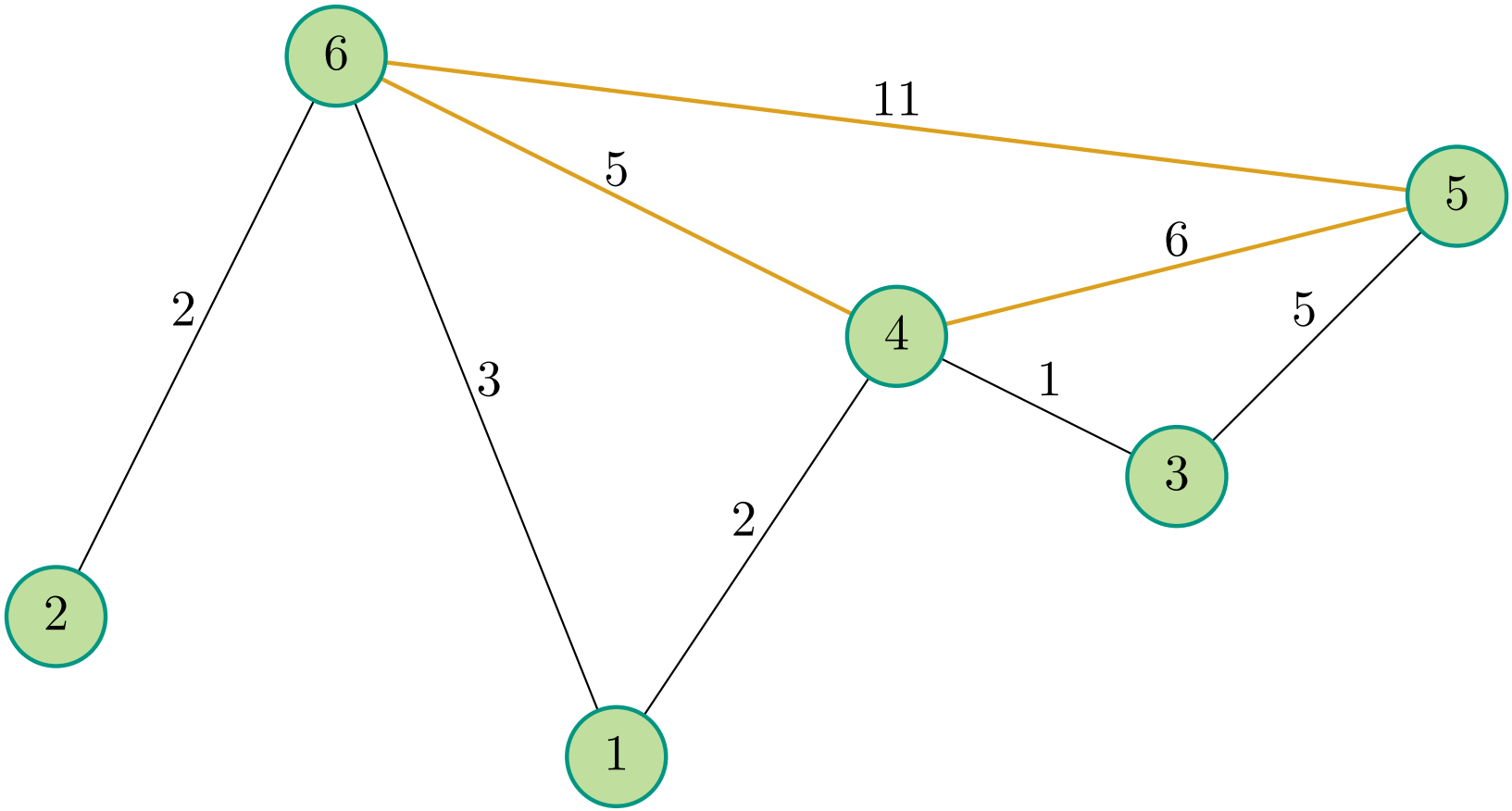
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



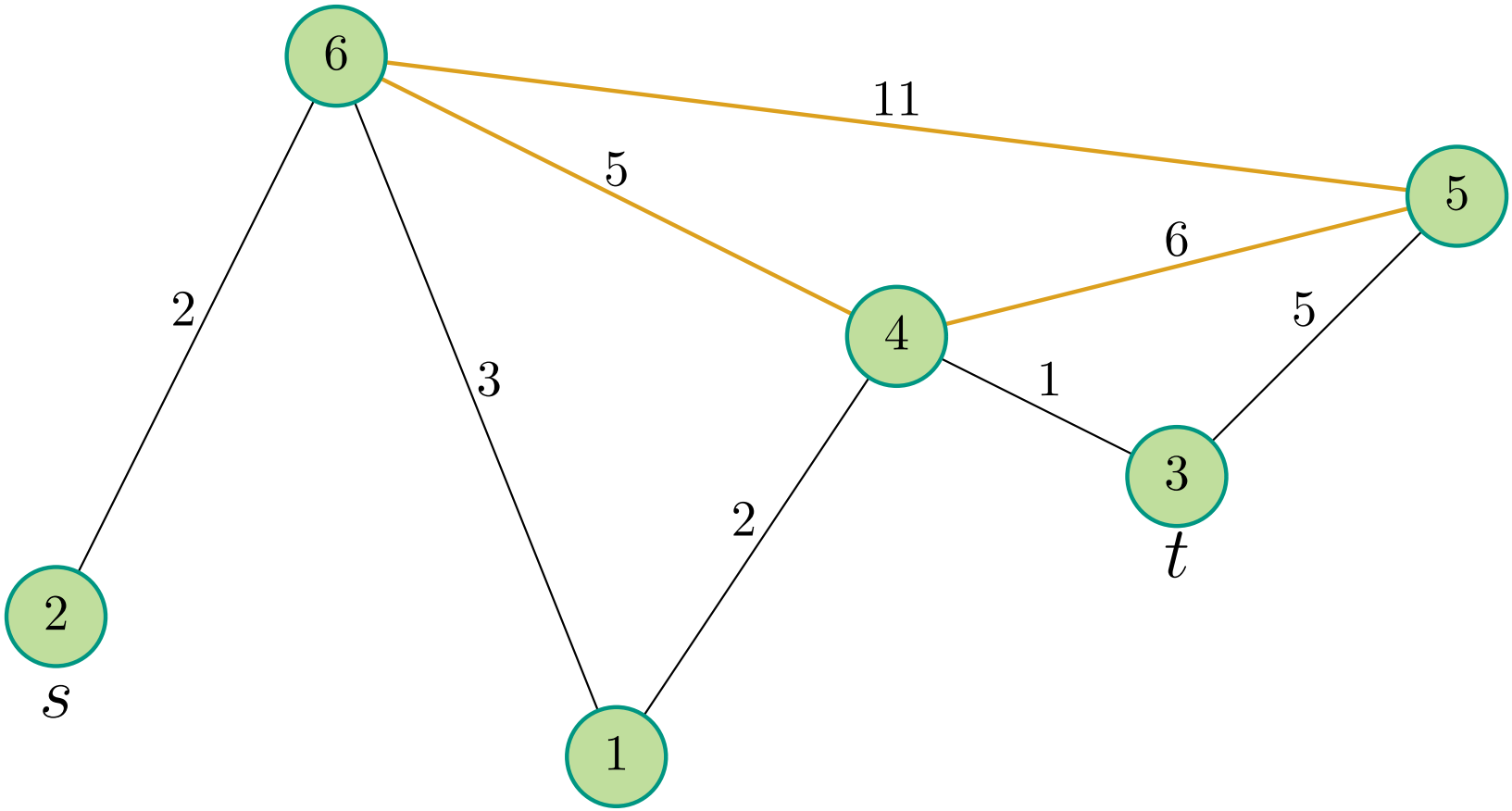
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



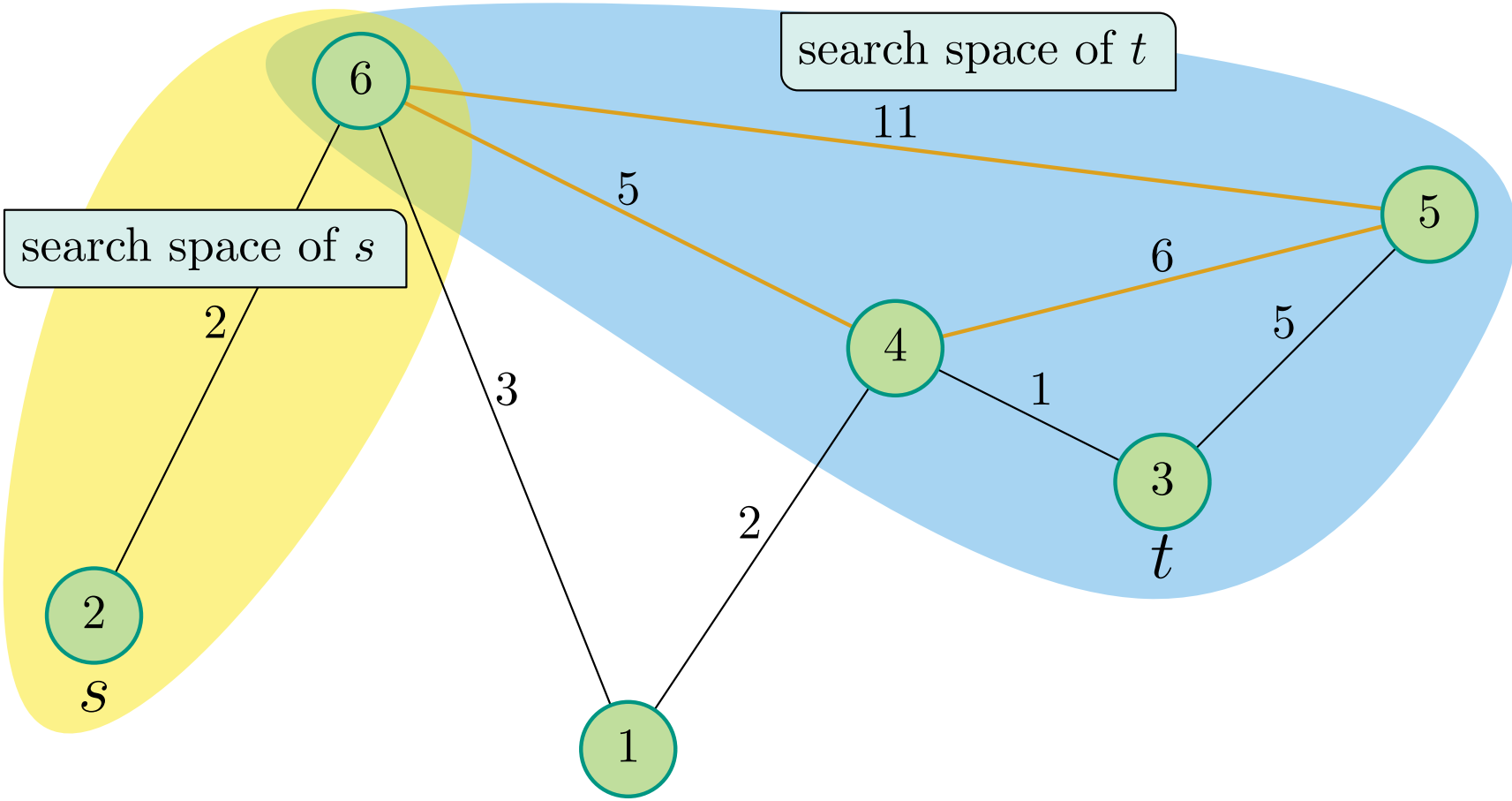
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



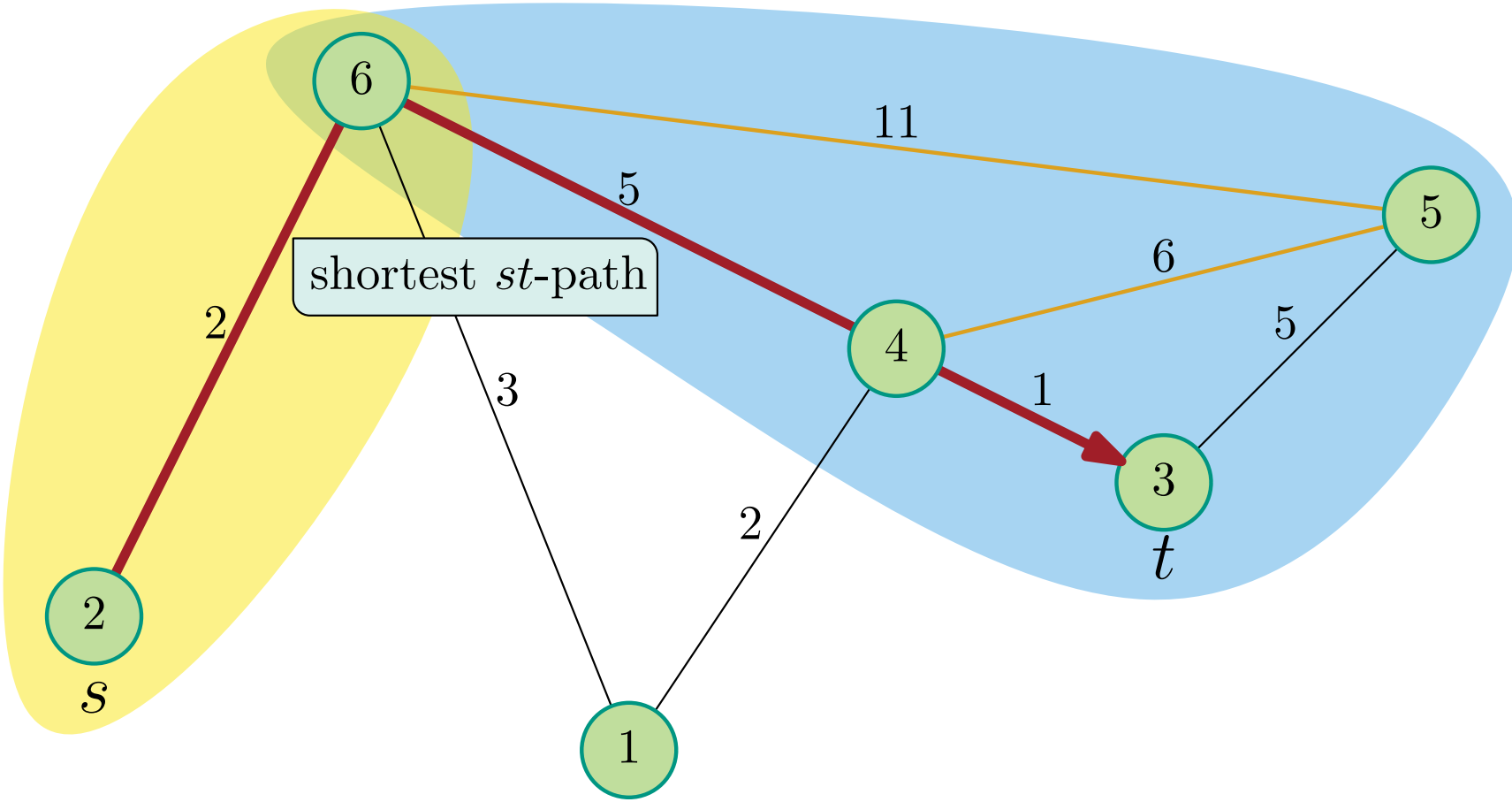
Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



Knoten nummeriert nach “Wichtigkeit”

Contraction Hierarchy



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

Anfrage

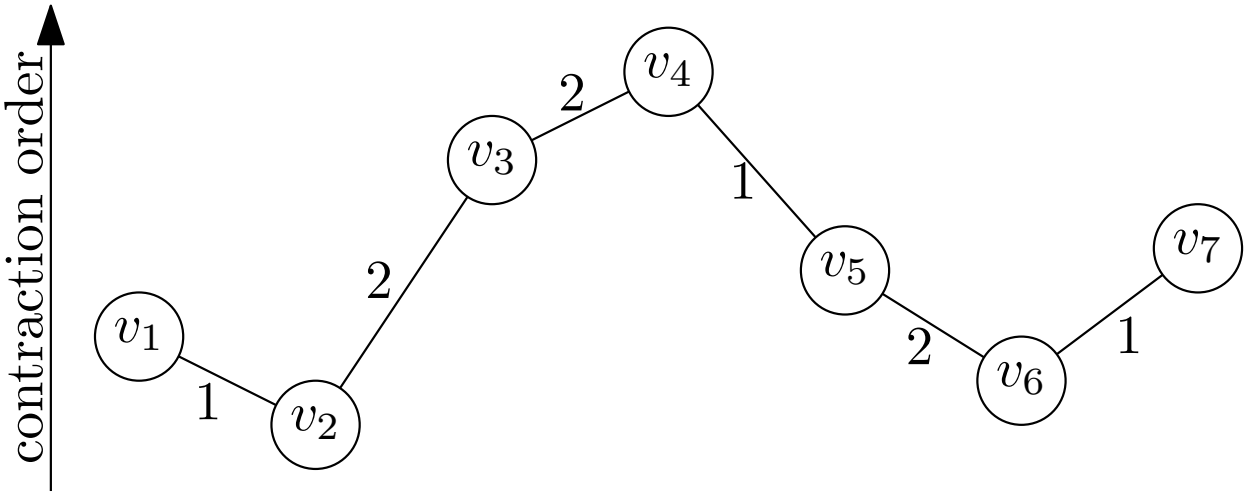
- Bidirektionale Variante von Dijkstras Algorithmus
- Verfolge nur Kanten zu wichtigeren Knoten
- Vorwärtssuche findet den “hoch”-Teil des Pfads
- Rückwärtssuche findet den “runter”-Teil des Pfads
- Abbruch wenn der min-key beider Queues größer ist als der bisher kürzeste gefundene Pfad

Korrektheit

- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.

Korrektheit

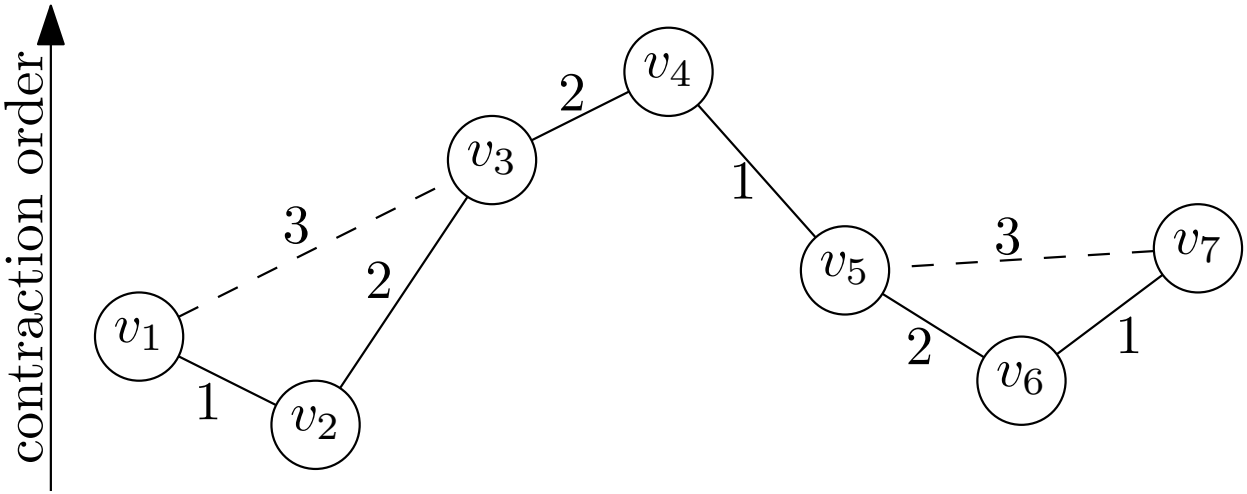
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.



- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch einen hoch-runter Pfad

Korrektheit

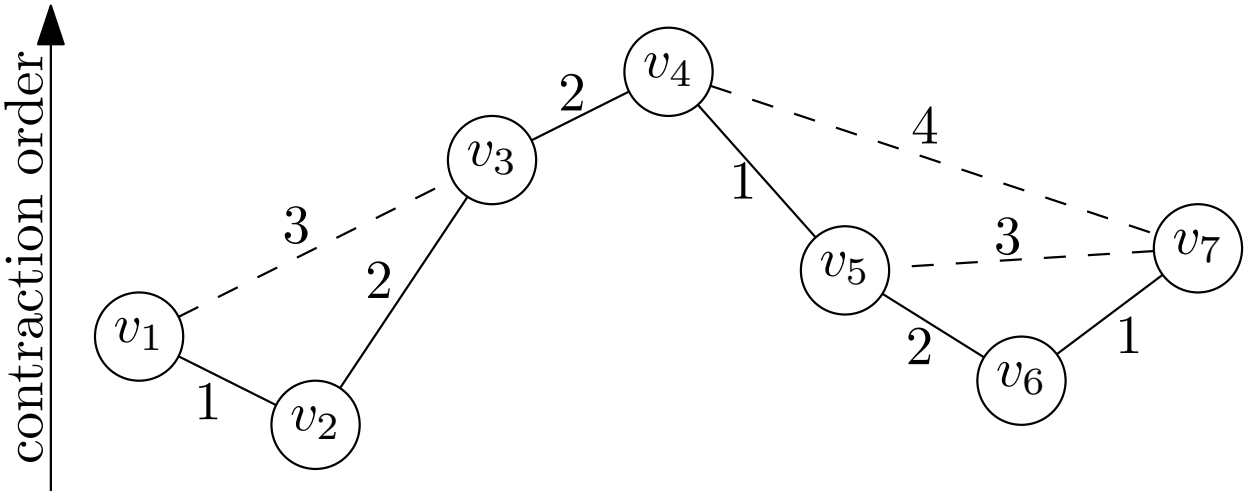
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.



- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch einen hoch-runter Pfad

Korrektheit

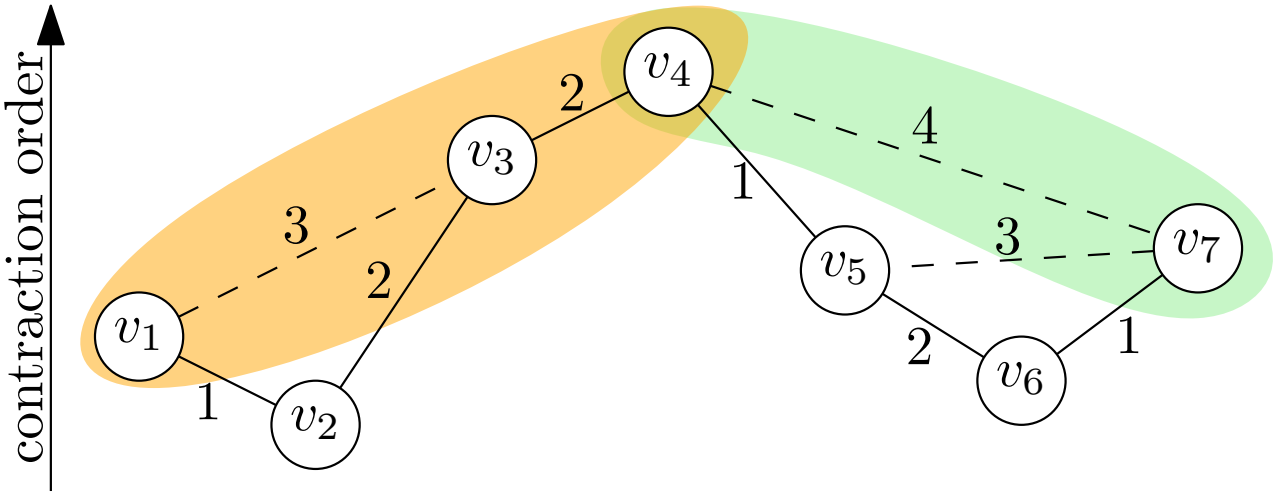
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.



- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch einen hoch-runter Pfad

Korrektheit

- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in G^+ immer einen kürzesten hoch-runter Pfad gibt.



- Es gibt einen kürzesten Weg P
- Wenn P kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch einen hoch-runter Pfad

Nach “Wichtigkeit” Ordnen

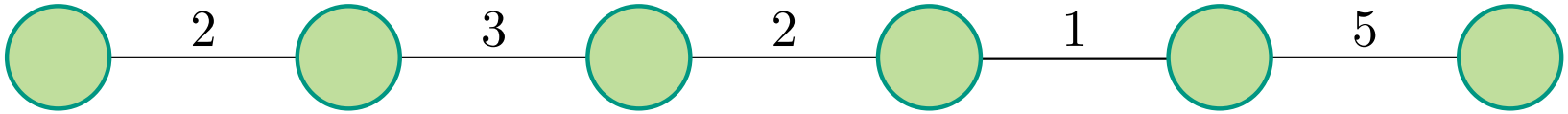
Grund-Idee:

- Wir wollen wenig Shortcuts
- Ein Knoten ist “unwichtig” wenn er wenig Shortcuts erzeugt
- $\rightarrow$ simuliere Knotenkontraktion um Knoten zu gewichten

Algorithmus:

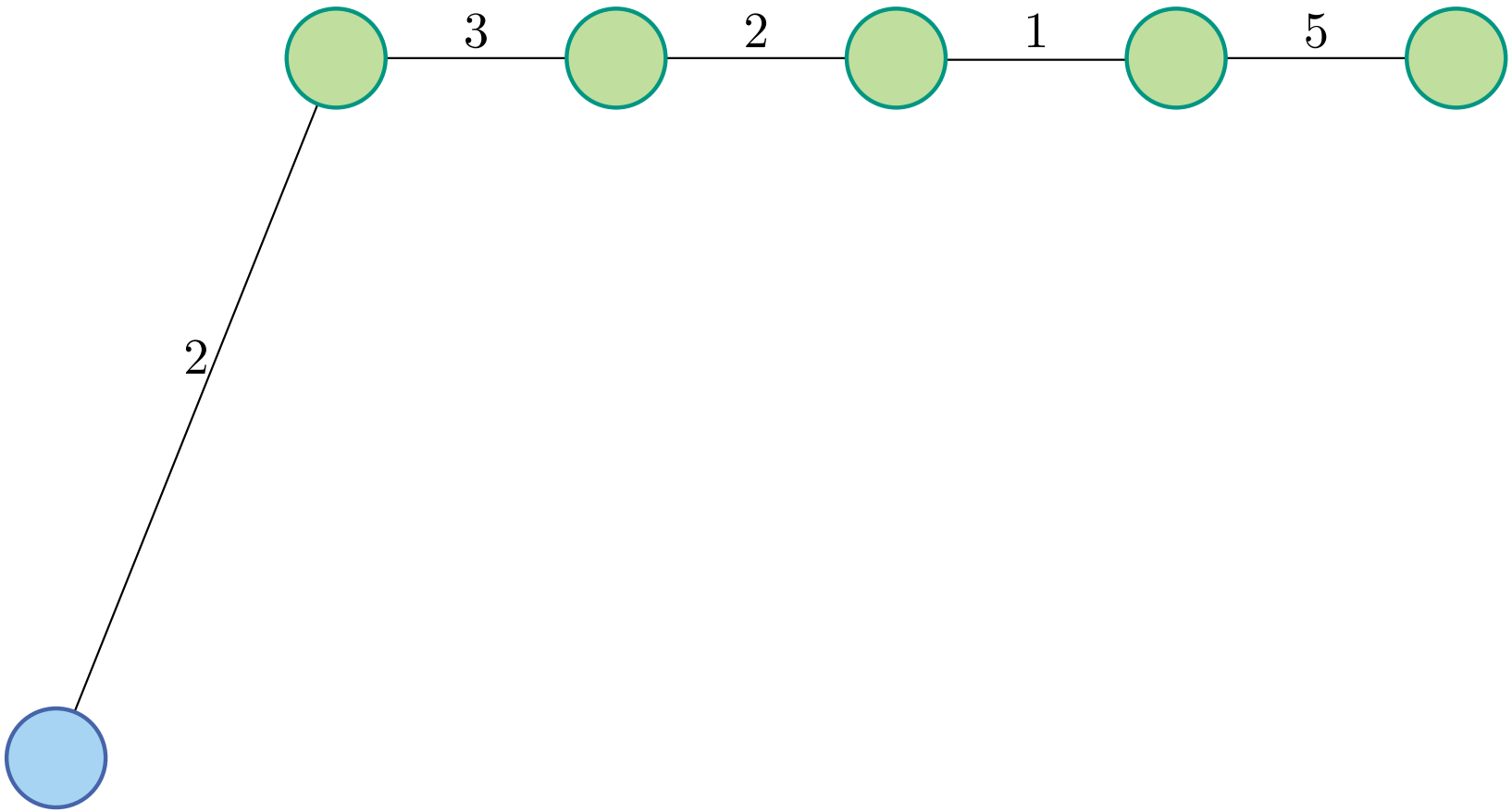
- Baue eine große Warteschlange mit allen Knoten gewichtet nach ihrer “Wichtigkeit”
- Kontrahiere iterative unwichtigsten Knoten
- Kontraktion eines Knoten kann “Wichtigkeit” der Nachbarn beeinflussen
- $\rightarrow$ “Wichtigkeit” der Nachbarn neu berechnen

Problemfall: Pfad



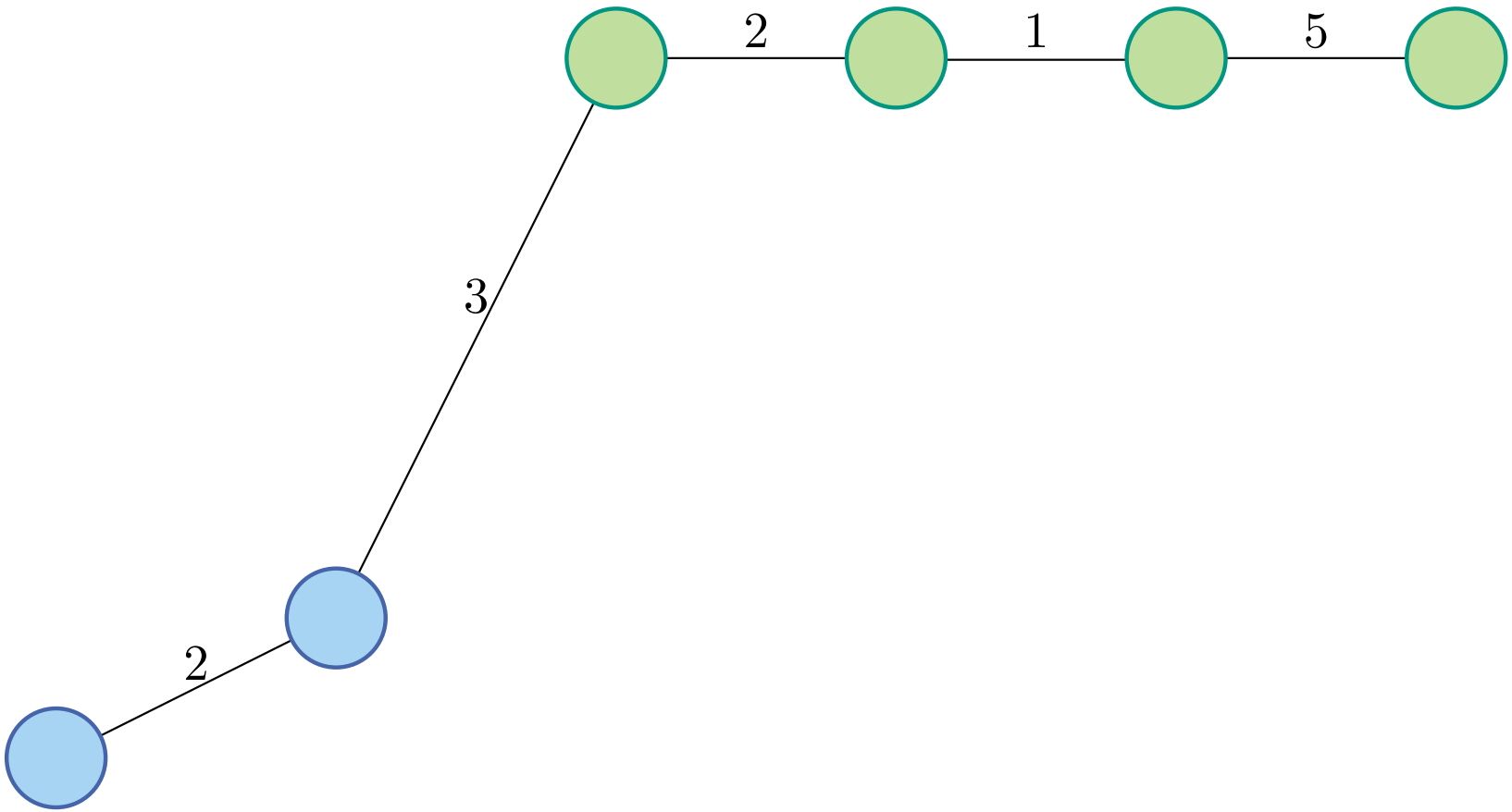
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



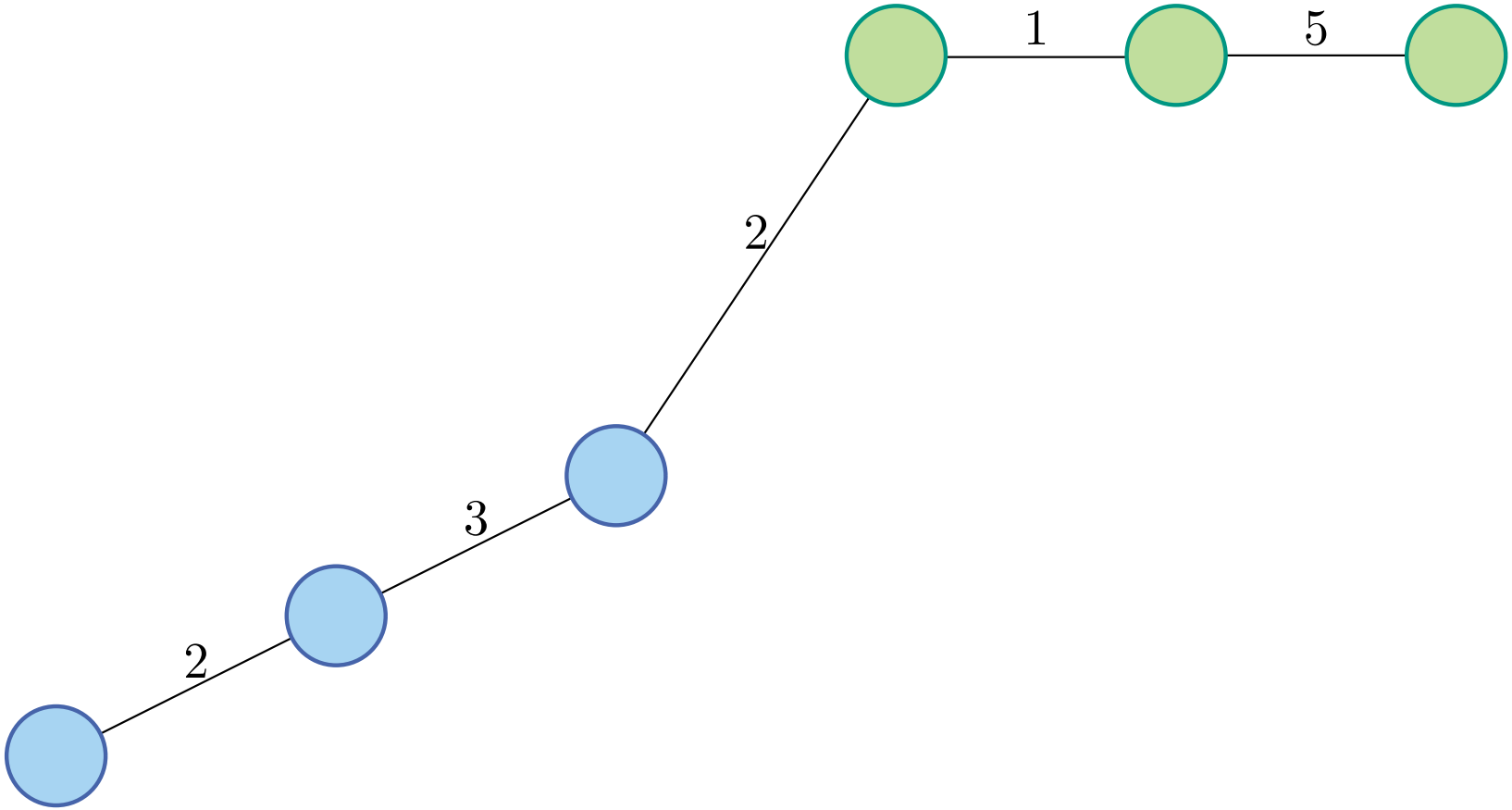
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



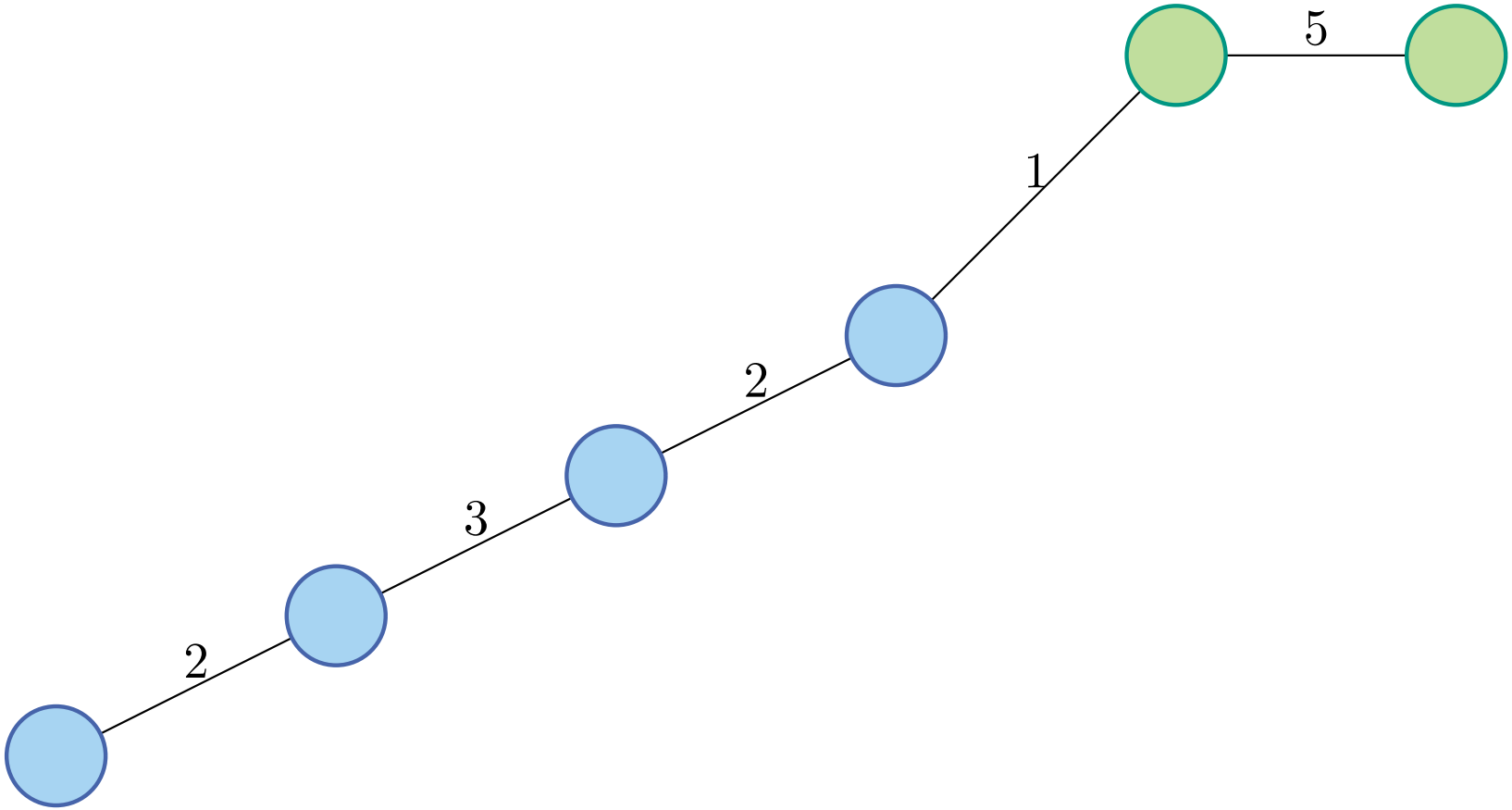
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



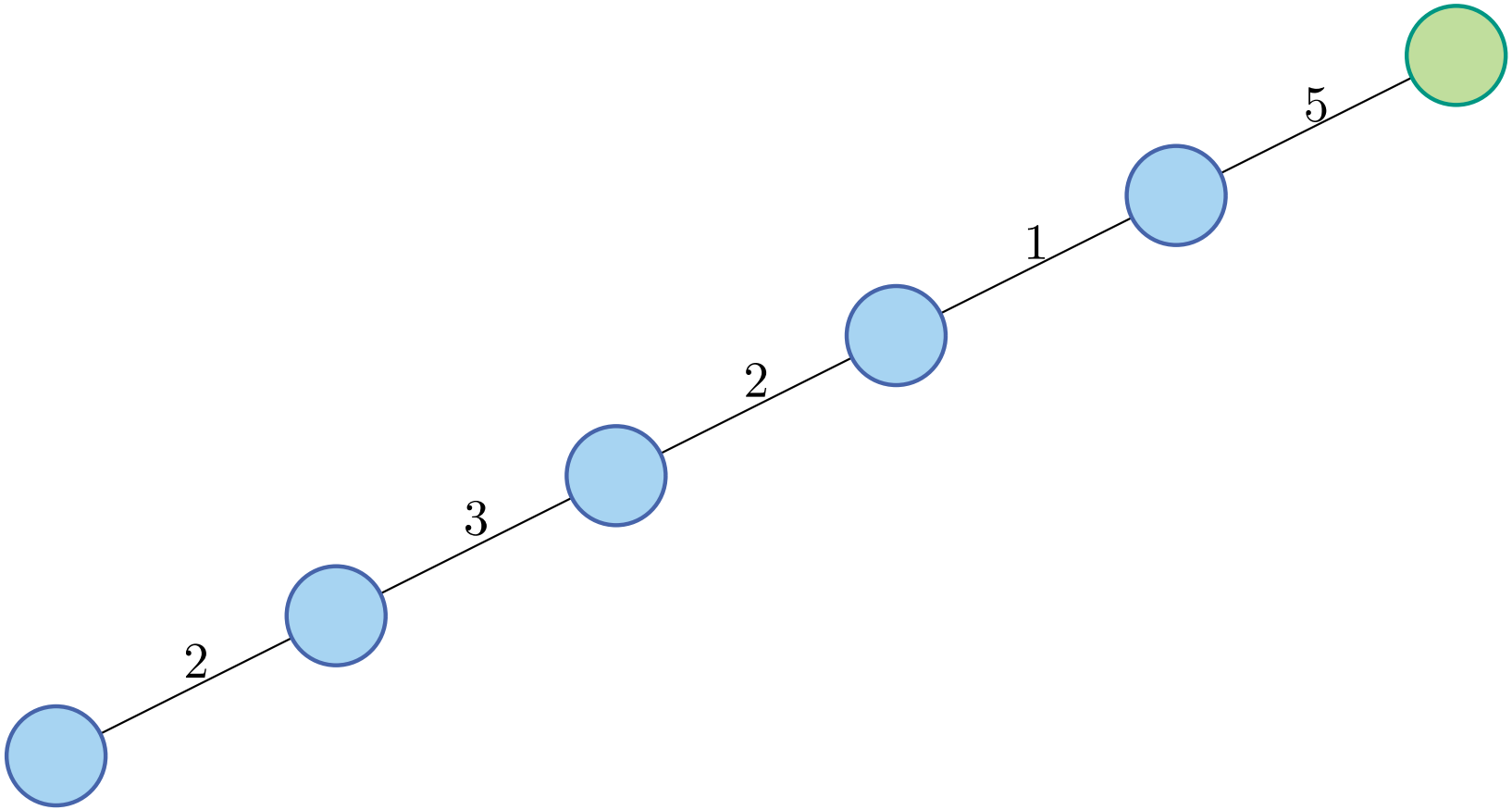
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



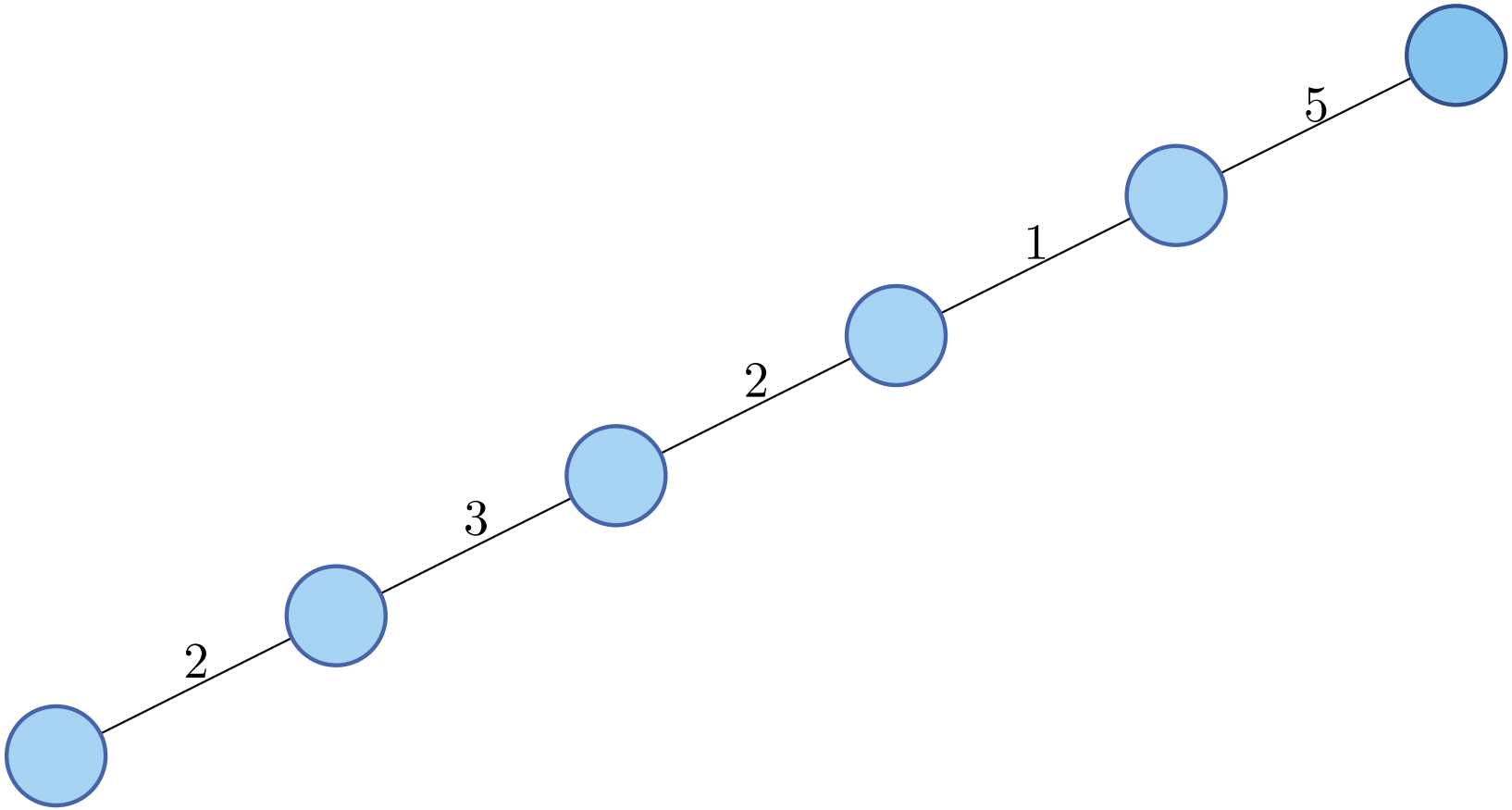
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



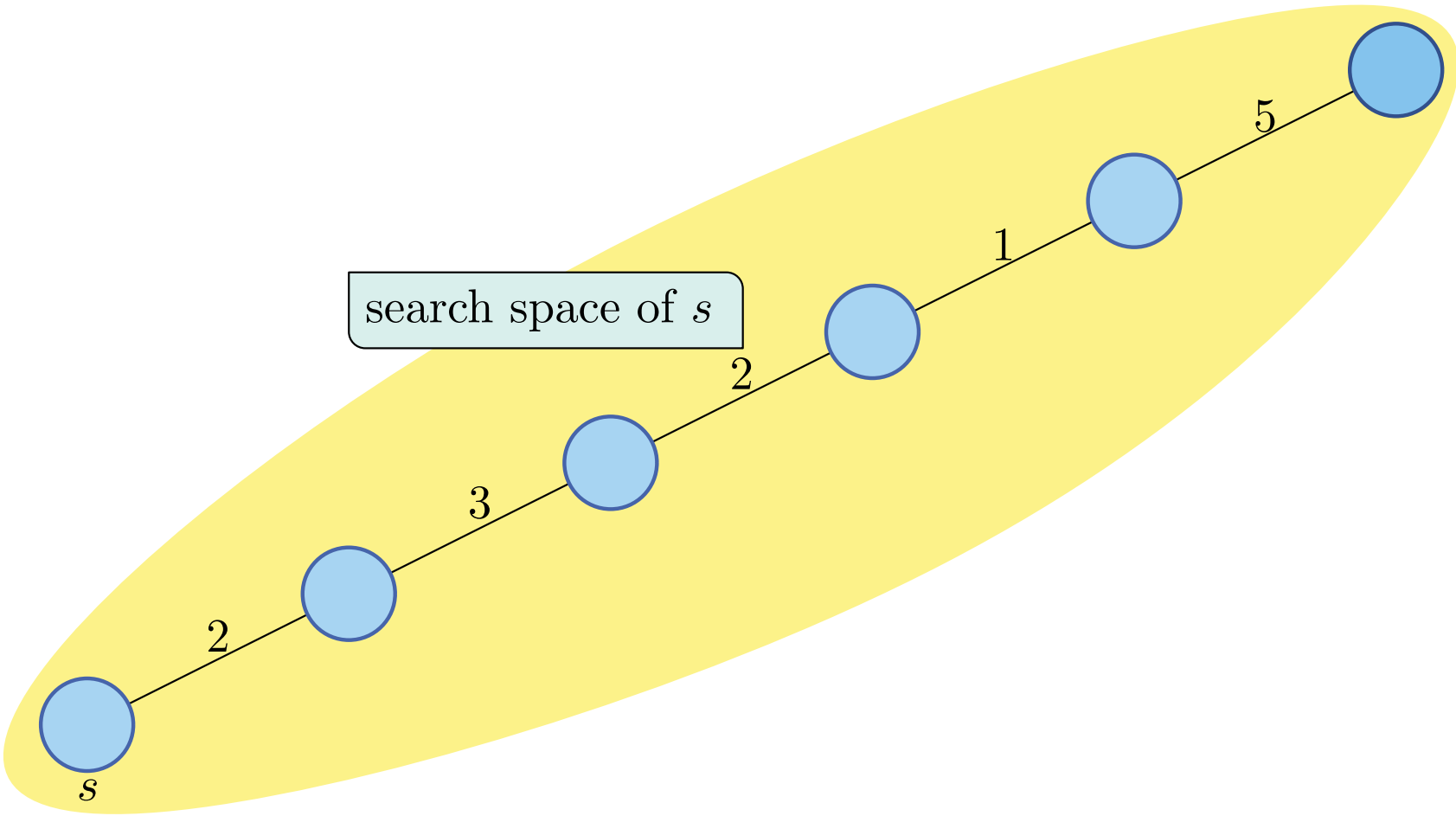
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



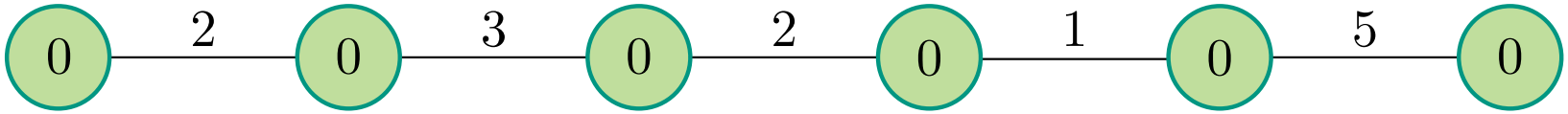
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



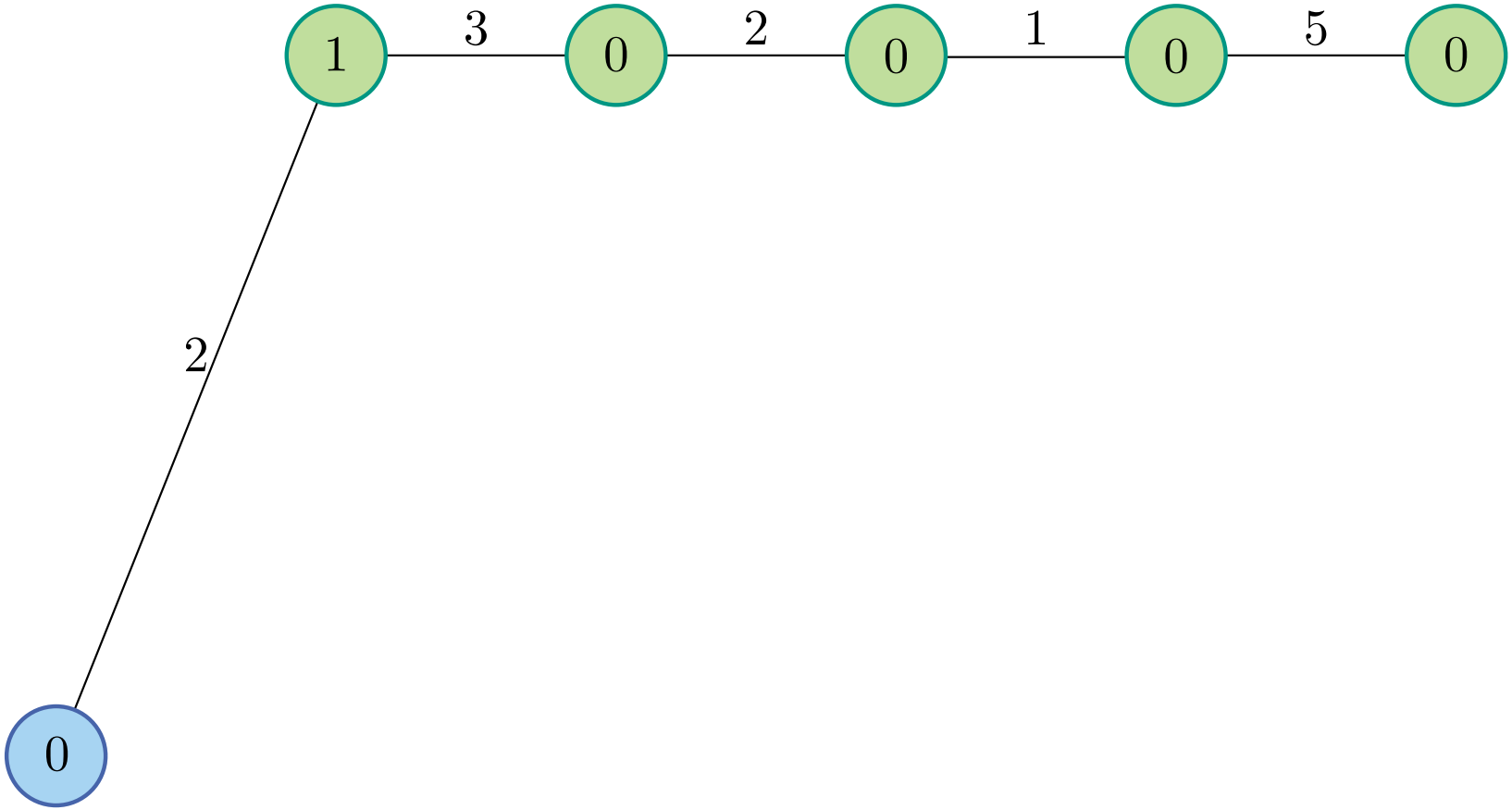
Suchraum von s ist der ganze Graph $\rightarrow$ keine Beschleunigung

Problemfall: Pfad



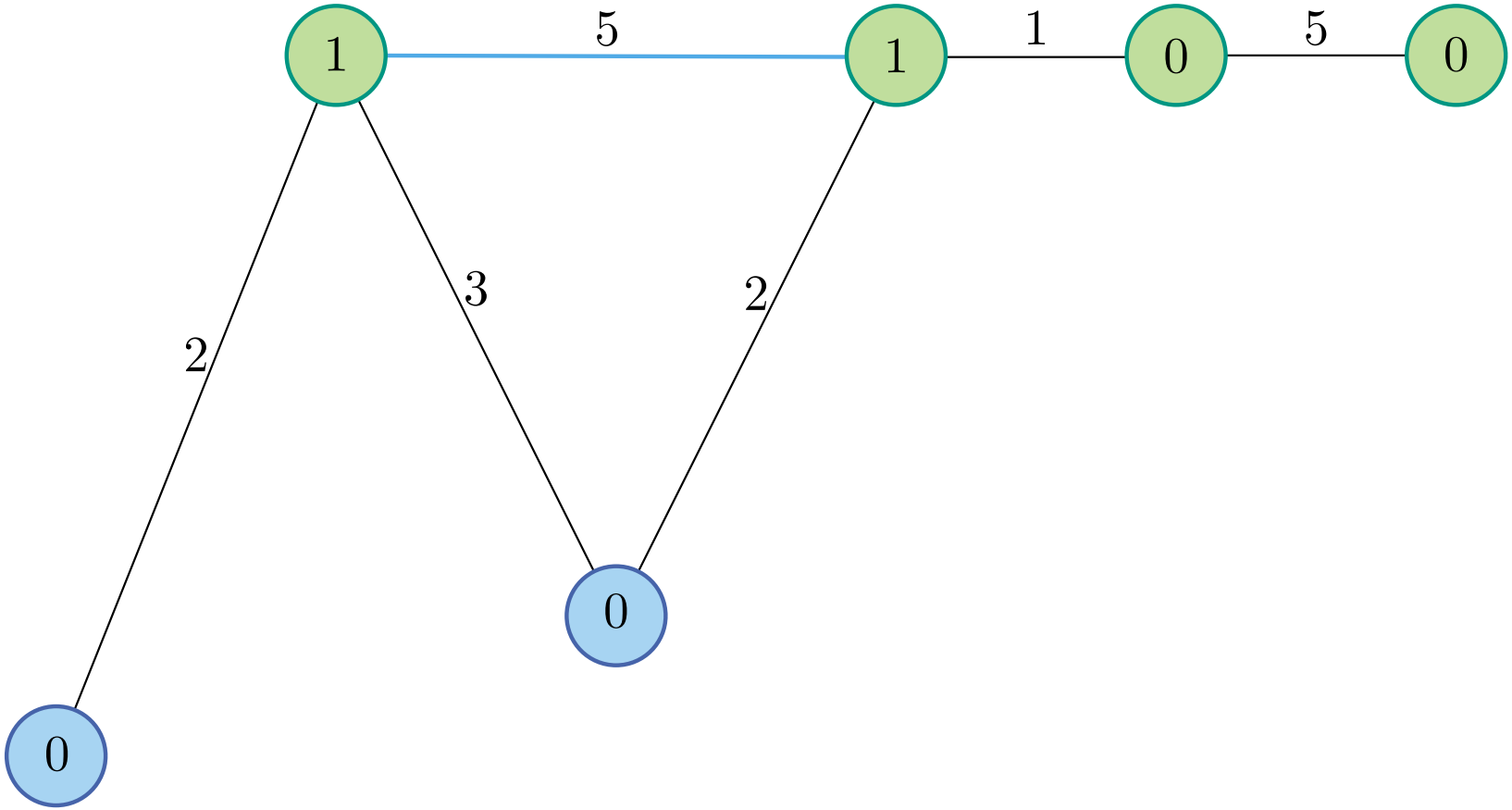
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



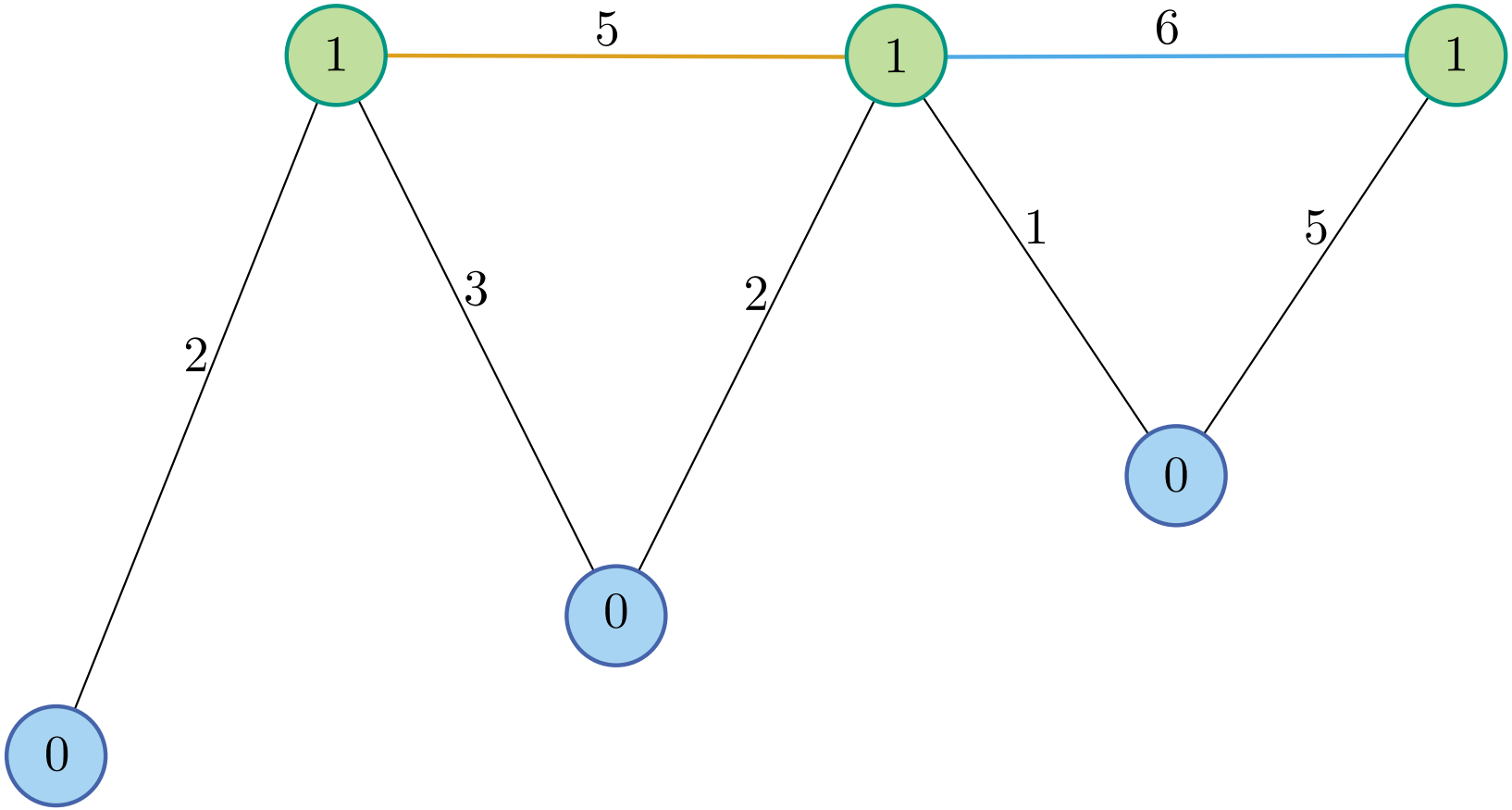
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

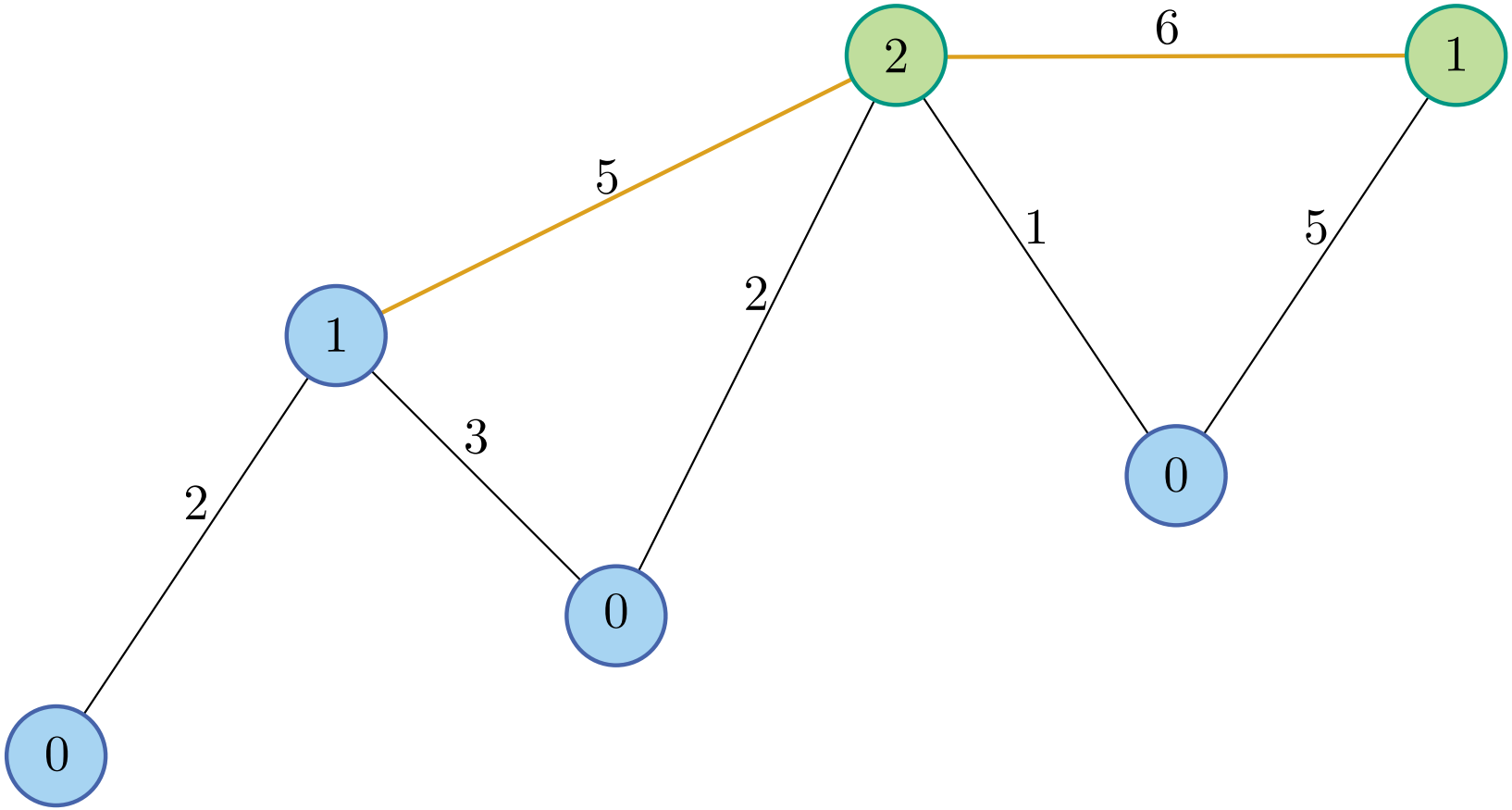
Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

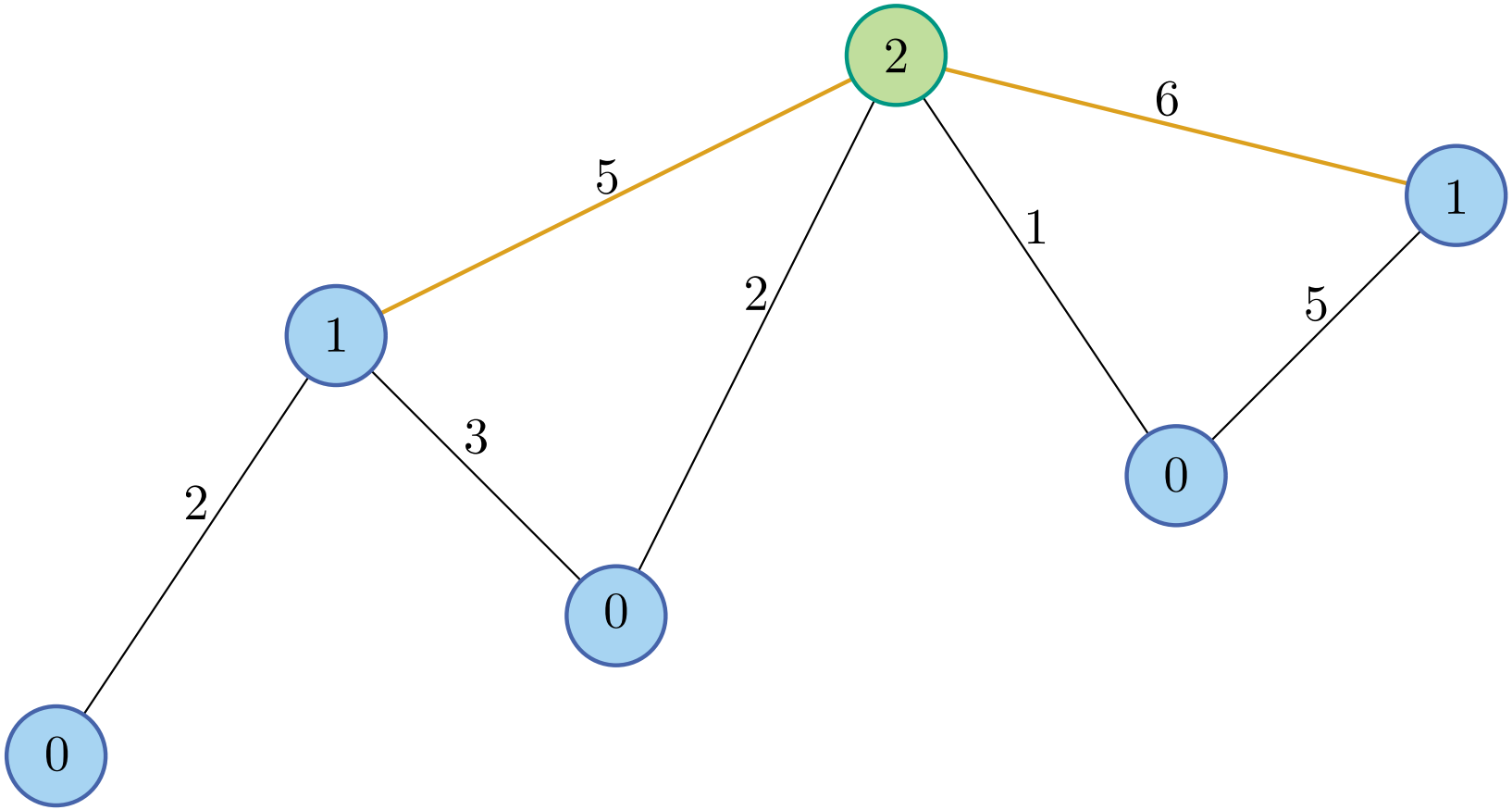
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Problemfall: Pfad



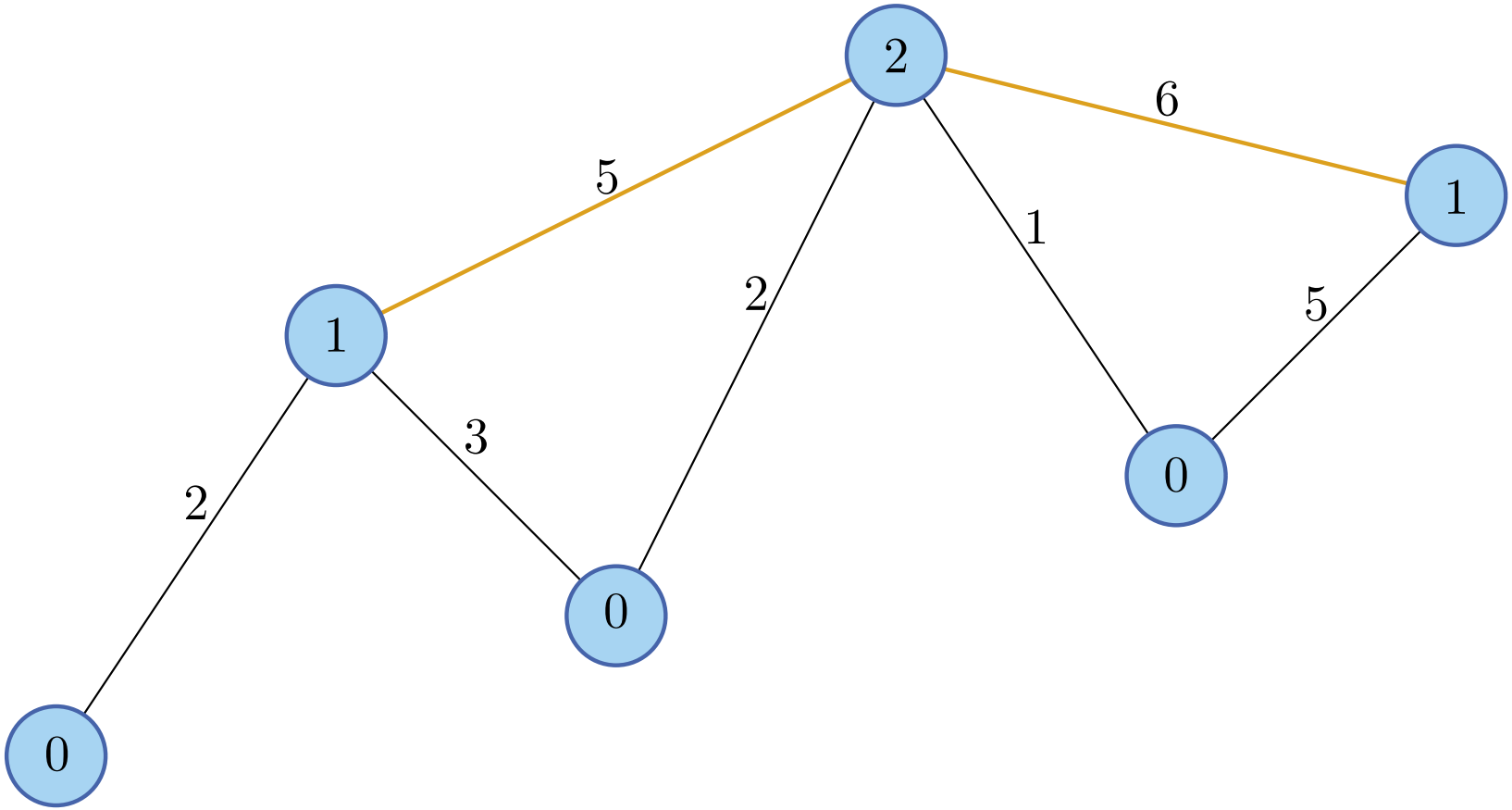
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



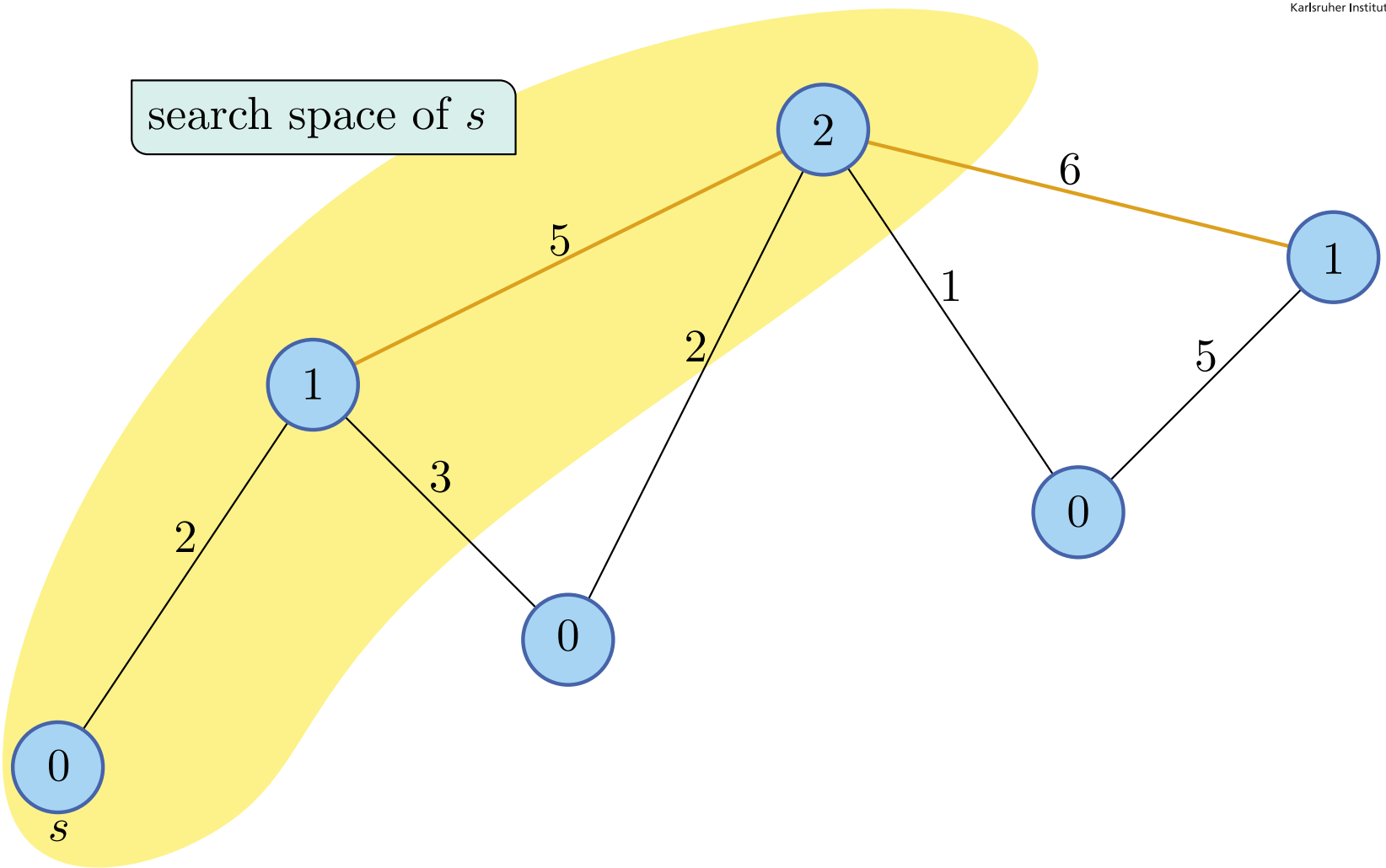
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Problemfall: Pfad



2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert $\rightarrow$ für alle Nachbarn y: $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$

Kombination mehrerer Kriterien

- Speichere für jede Kante e die Anzahl $h(e)$ der original Kanten aus der sie besteht
- Es sei $A(x)$ die Menge der eingefügten Shortcuts wenn x kontrahiert werden würde
- Analog: $D(x)$ die Menge der gelöschten Kanten
- Es sei $I(x)$ die "Wichtigkeit" von x

Eine funktionierende Definition von $I(x)$ ist

$$I(x) := \ell(x) + \frac{|A(x)|}{|D(x)|} + \frac{\sum_{e \in A(x)} h(e)}{\sum_{e \in D(x)} h(e)}$$

Hinweis: Es gibt sehr viele unterschiedliche Definitionen für I. Das ist nur ein Kochrezept, das sich bewährt hat und jeder würzt leicht anders

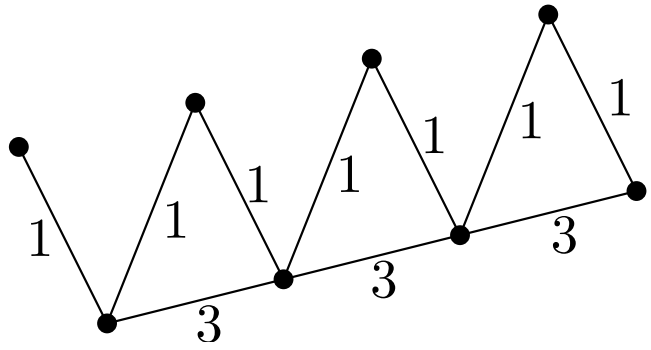
CH im Detail

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

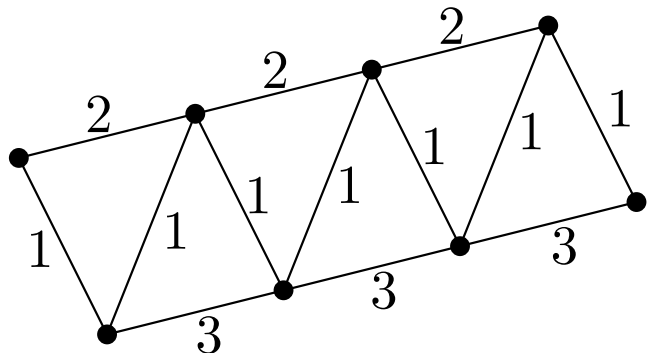


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

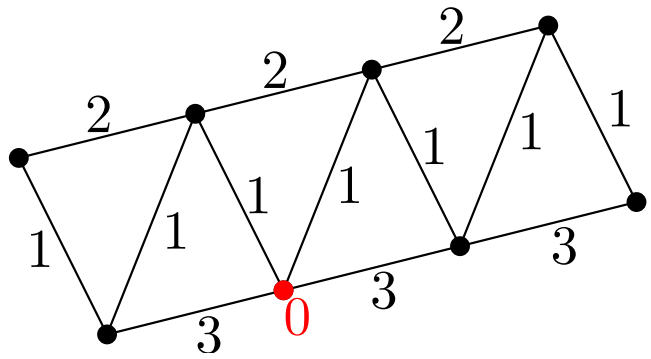


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Stall-On-Demand

Beobachtung:

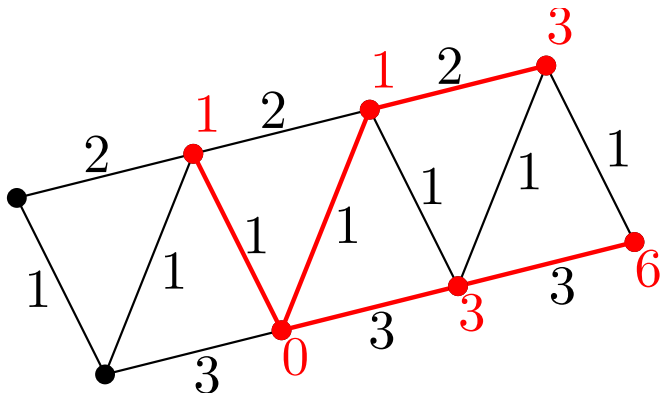
- Suchen können Knoten mit zu großer Distanz besuchen



Stall-On-Demand

Beobachtung:

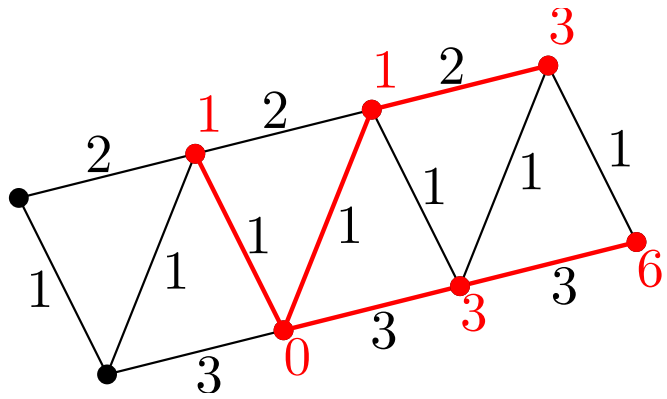
- Suchen können Knoten mit zu großer Distanz besuchen



Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

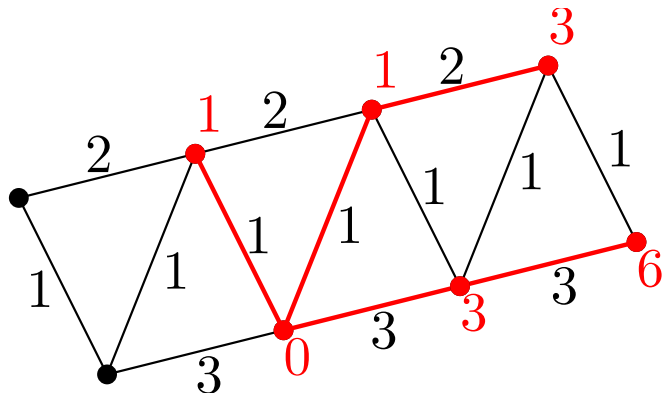


- Kann man zum prunen verwenden

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

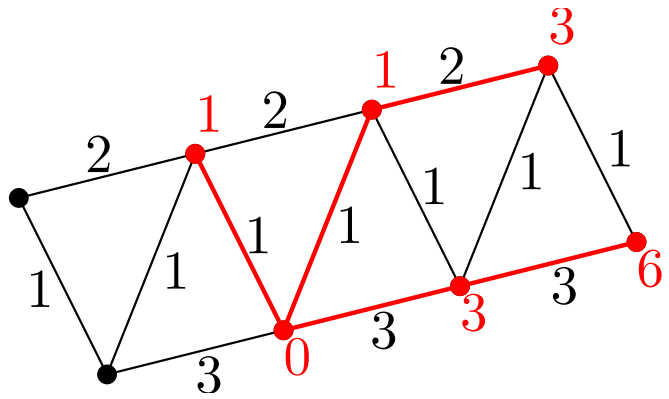


- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settled sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

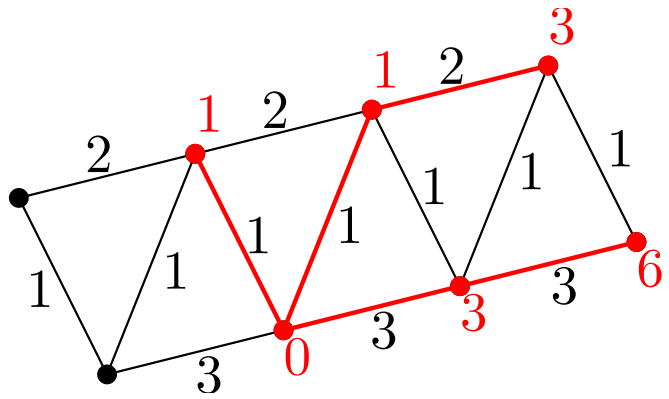


- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settled sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen
- Knoten v wird geprunt, wenn es Aufwärtsnachbar u von v gibt, so dass $d(u) + w(u, v) < d(v)$

Stall-On-Demand

Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

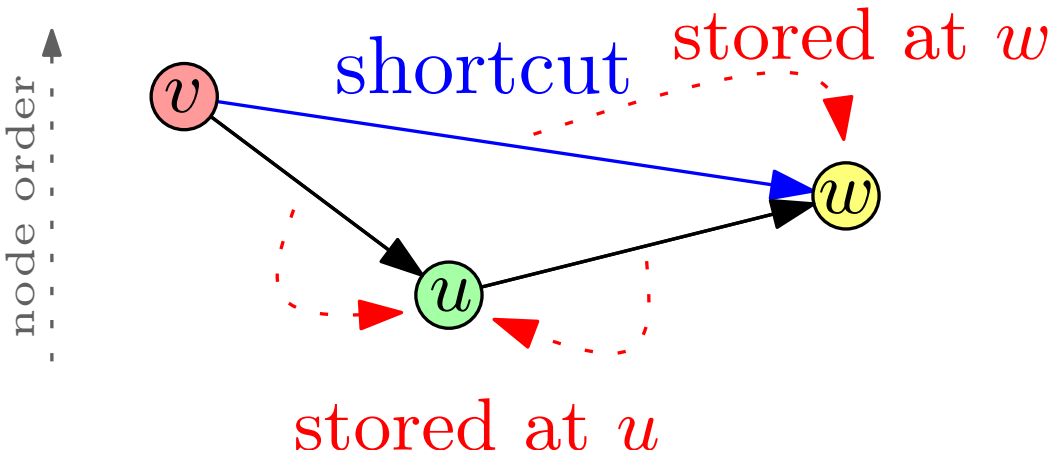


- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settled sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen
- Knoten v wird geprunt, wenn es Aufwärtsnachbar u von v gibt, so dass $d(u) + w(u, v) < d(v)$
- (Dies ist eine vereinfachte Version des ursprünglichen “Stall-On-Demand”.)

Datenstruktur

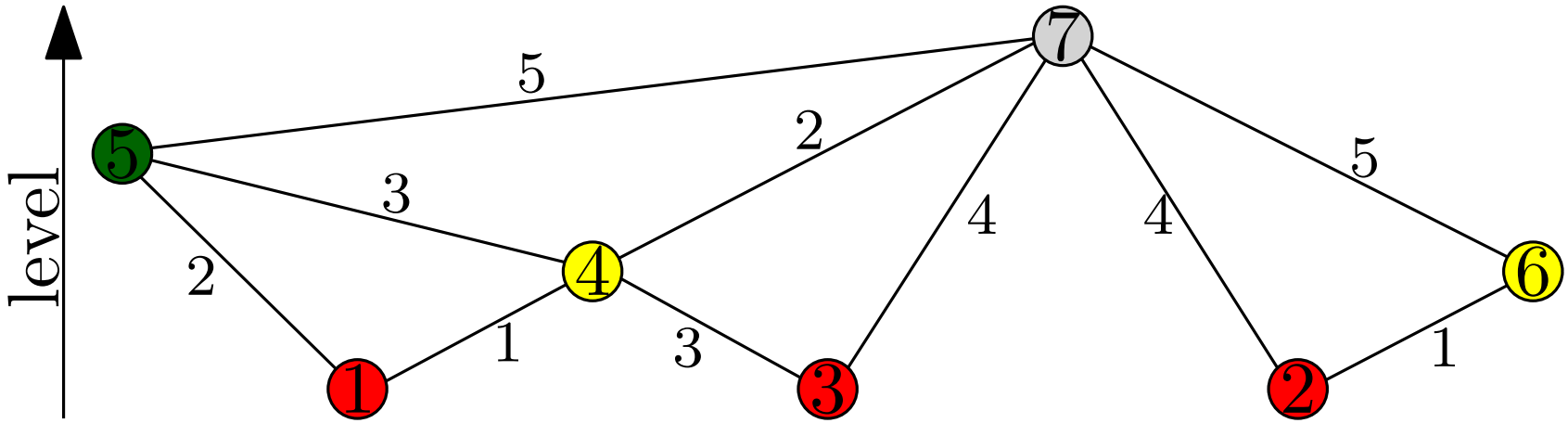
Suchgraph:

- normalerweise: speichere Kanten (v, w) in den Adjacenz-Arrays von v und w um bidirektionale Suche zu erlauben
- für die CH-Suche reicht es aus, die Kante nur an den Knoten $\min\{r(v), r(w)\}$ zu speichern



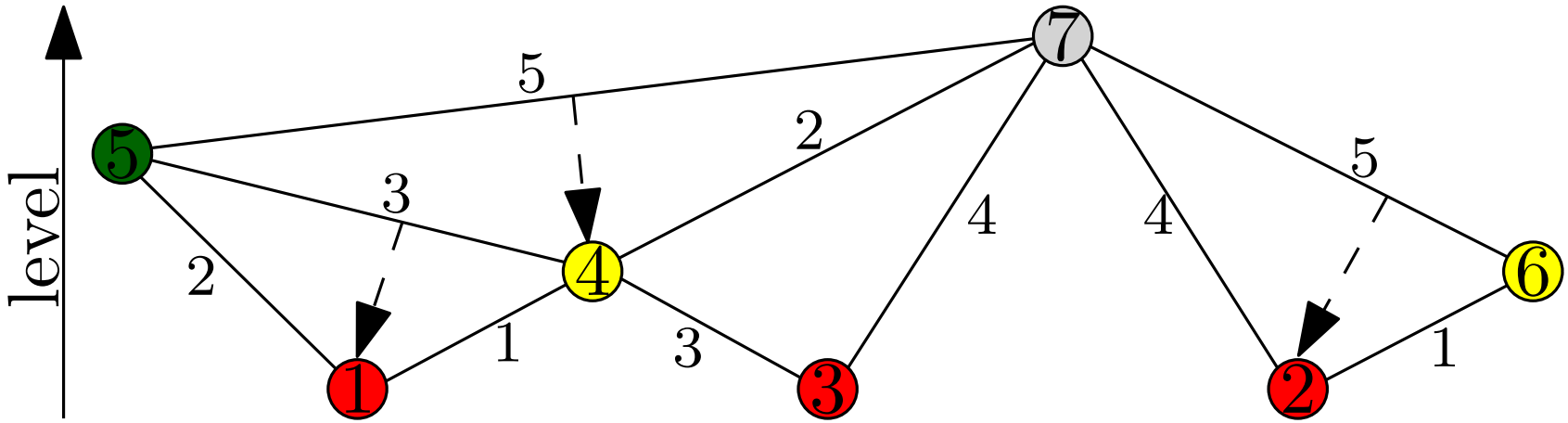
Pfadentpackung

- für jeden Shortcut (u, w) eines Pfades (u, v, w), speichere Mittelknoten v an der Kante
- expandiere Pfade mittels Rekursion



Pfadentpackung

- für jeden Shortcut (u, w) eines Pfades (u, v, w), speichere Mittelknoten v an der Kante
- expandiere Pfade mittels Rekursion



Knotenordnung

Wie Knoten ordnen?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Knotenordnung

Wie Knoten ordnen?

- Bottom-Up, suche nach unwichtigen Knoten [GSSV12], haben wir schon gemacht
- Top-Down, suche nach wichtigen Knoten [ADGW12]
- Sampling Path-Greedy, Variante von Top-Down [DGPW14]

Top-Down

Path-Greedy

- Idee: Baue Ordnung von wichtig nach unwichtig

- initial ist die Ordnung leer
- Pfad ist überdeckt wenn er einen Knoten in der Ordnung enthält
- sei U die Menge der nicht überdeckten kürzesten Pfade
- initial ist U die Menge aller kürzesten Pfade

- Algo:
  - Solange die Ordnung nicht voll:
  - Packe Knoten v oben in die Ordnung, so dass v möglichst viele Pfade aus U überdeckt
  - Entferne diese Pfade aus U

Top-Down

Path-Greedy: Diskussion

- n mal All-Pair-Shortest-Path
  $\rightarrow n^3 \log n$ mit Dijkstra auf dünnen Graphen
- linear Speicherverbrauch
- Kann auf $n^2 \log n$ gedrückt werden mit $O(n^2)$ Speicher
  $O(n^2)$ kommt daher, dass n kürzeste Wege Bäume verwaltet werden
- gute Qualität der Ordnung aber langsam

Sampling Path-Greedy [DGPW14]

- Problem von Top-Down: $O(n^2)$ Speicherverbrauch, da n kürzeste Wege Bäume verwaltet werden
- Ansatz von Sampling Path-Greedy: Speichere nur ein “paar” kürzeste Wege Bäume
- d.h., der Algorithmus sampled Bäume

Sampling Path-Greedy [DGPW14]

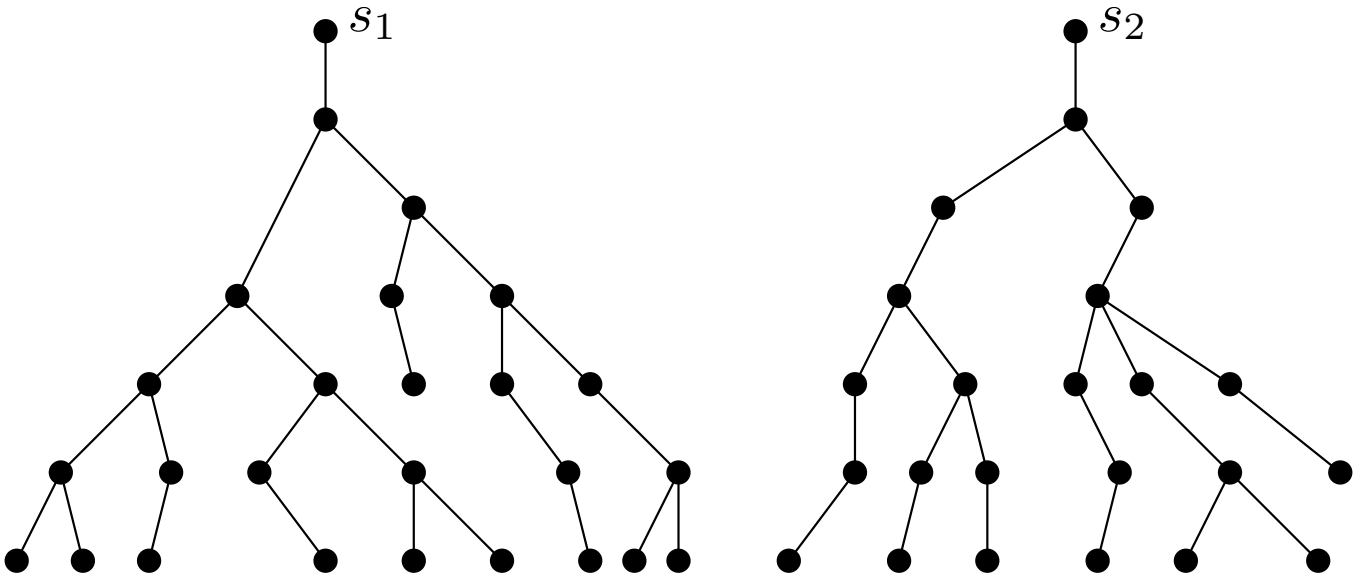
Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
  - Quellknoten: s_1, s_2 ...
- Wähle v das auf den meisten Pfaden in diesen Bäumen liegt

Sampling Path-Greedy [DGPW14]

Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
  - Quellknoten: s_1, s_2 ...
- Wähle v das auf den meisten Pfaden in diesen Bäumen liegt

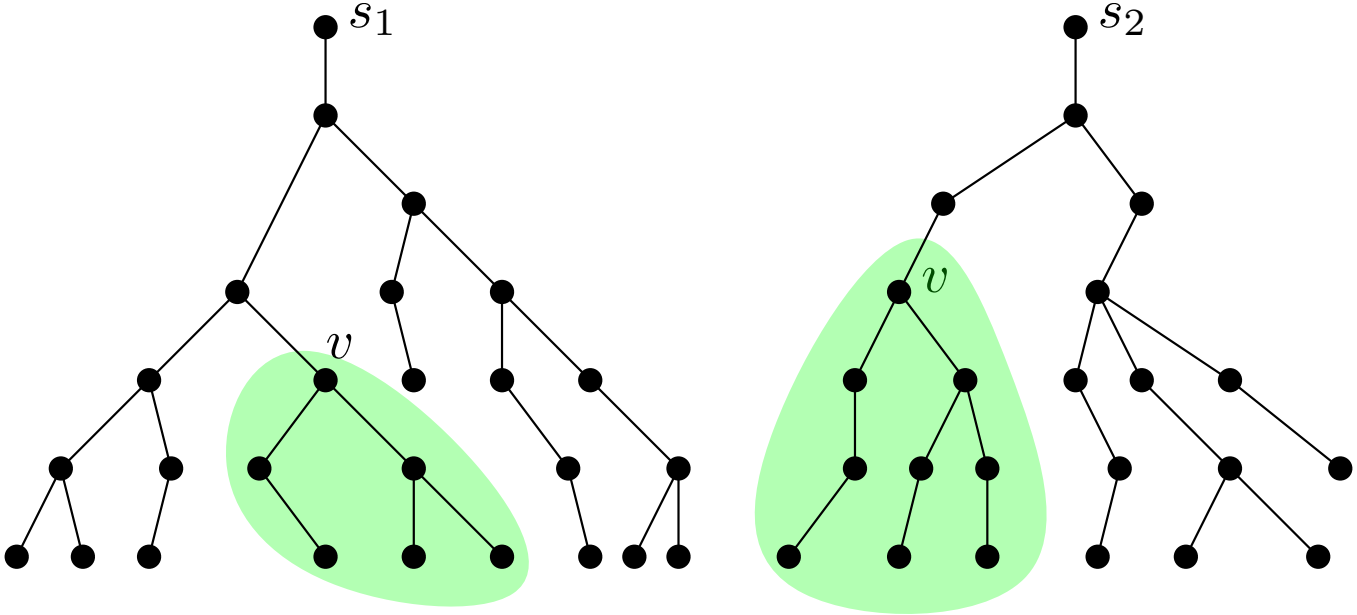


- Die kürzeste Wege Bäum von s_1 und s_2

Sampling Path-Greedy [DGPW14]

Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
  - Quellknoten: s_1, s_2 ...
- Wähle v das auf den meisten Pfaden in diesen Bäumen liegt

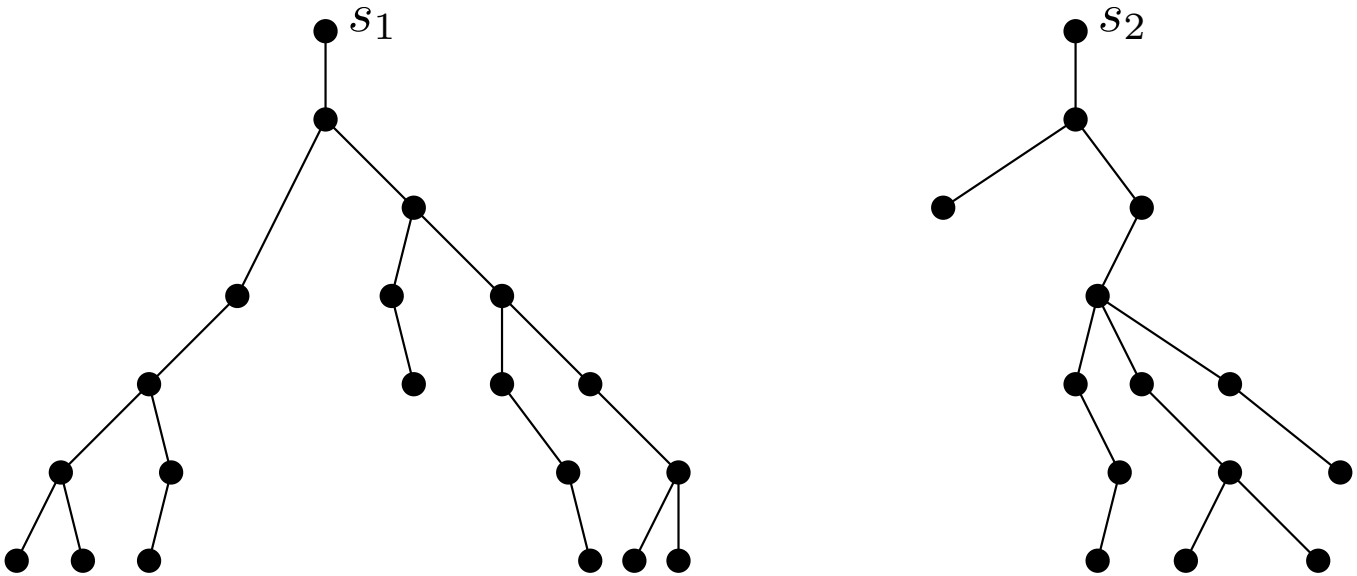


- Die kürzeste Wege Bäum von s_1 und s_2
- v liegt auf so vielen Pfaden die bei s_1 beginnen wie sein s_1-Teilbaum groß ist (hier 6)

Sampling Path-Greedy [DGPW14]

Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
  - Quellknoten: s_1, s_2 ...
- Wähle v das auf den meisten Pfaden in diesen Bäumen liegt



- Die kürzeste Wege Bäum von s_1 und s_2
- v liegt auf so vielen Pfaden die bei s_1 beginnen wie sein s_1-Teilbaum groß ist (hier 6)
- Entferne Teilbäume in allen verwalteten Bäumen

Sampling Path-Greedy [DGPW14]

Experimente zeigen:

- Nur wenige Bäume notwendig um die wichtigsten Knoten zu finden
- Für das Mittelfeld und die unwichtigen Knoten werden mehr Bäume gebraucht

Beobachtung:

- Durch löschen der Teilbäume wird Speicher frei

Idee:

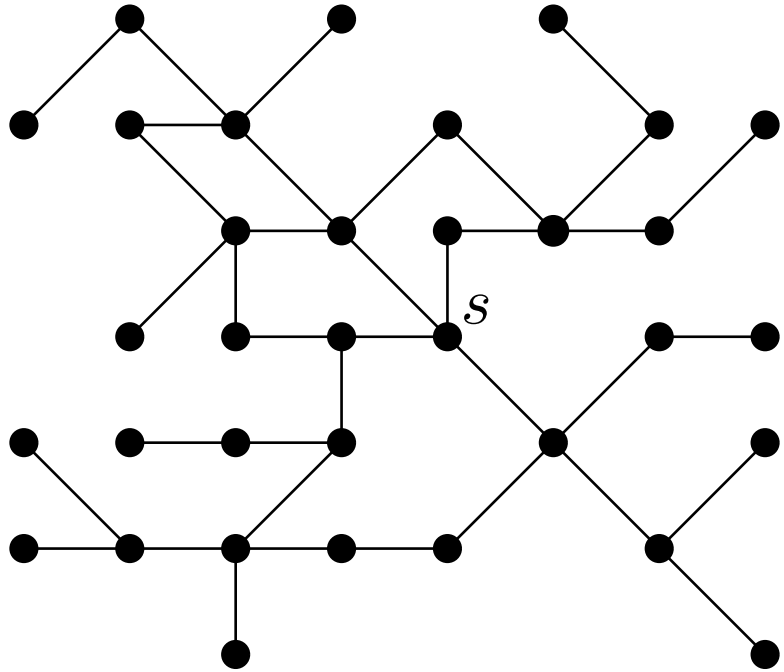
- Fülle frei gewordenen Speicher mit neuen Bäumen

Neue Bäume Aufbauen [DGPW14]

- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?

Neue Bäume Aufbauen [DGPW14]

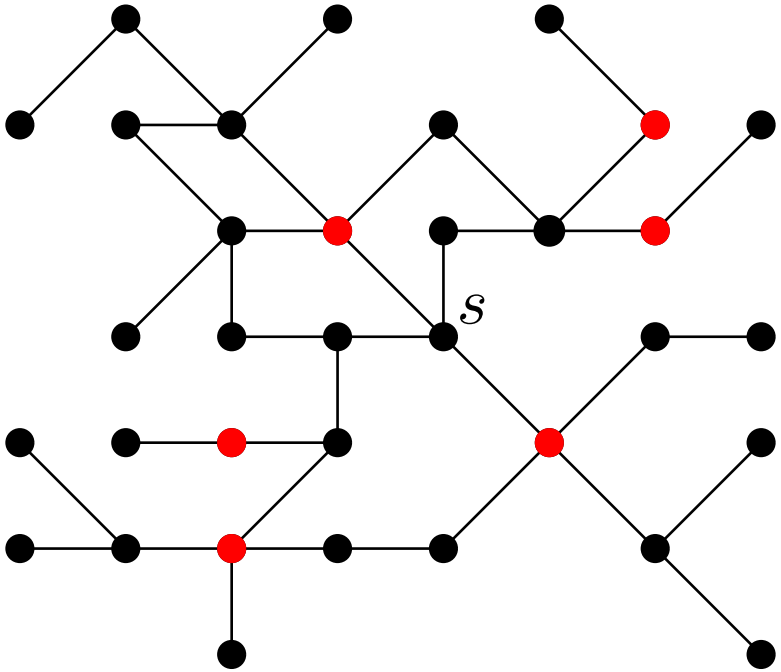
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



s ist neue Baumwurzel

Neue Bäume Aufbauen [DGPW14]

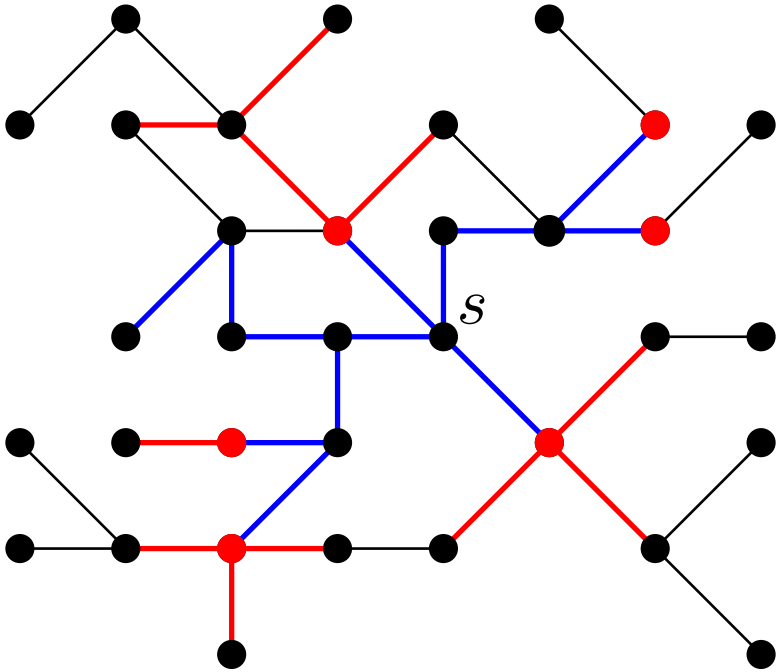
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



rote Knoten sind bereits in der Ordnung

Neue Bäume Aufbauen [DGPW14]

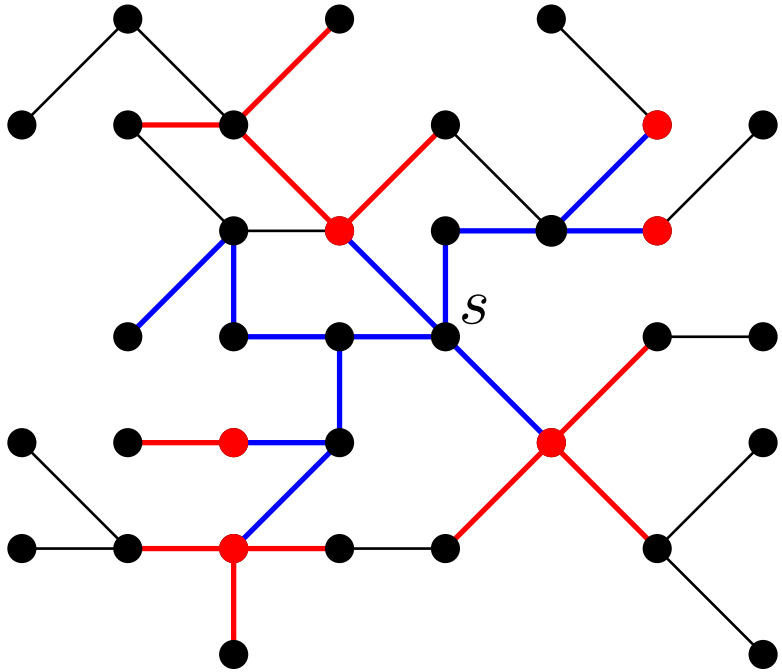
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



lasse Dijkstra laufen

Neue Bäume Aufbauen [DGPW14]

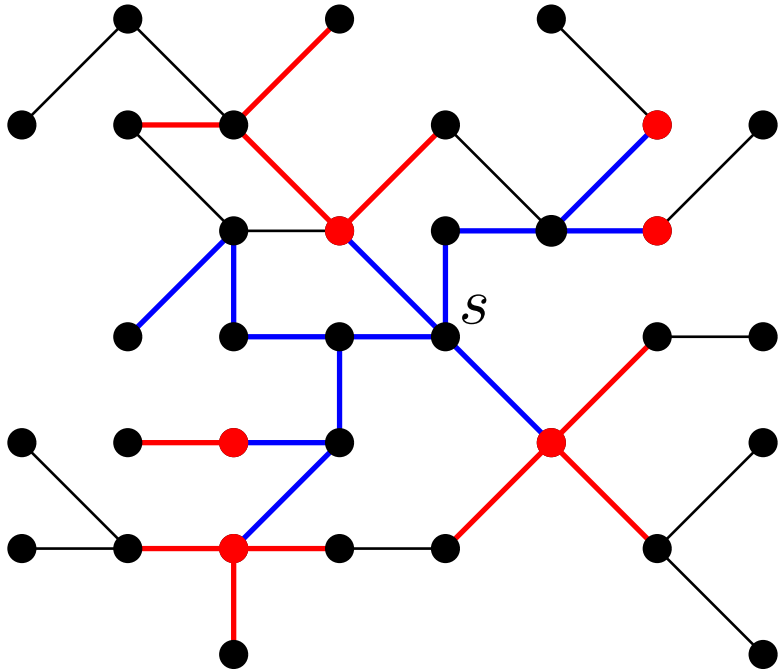
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



speichere welche Knoten über rote Knoten erreicht wurden

Neue Bäume Aufbauen [DGPW14]

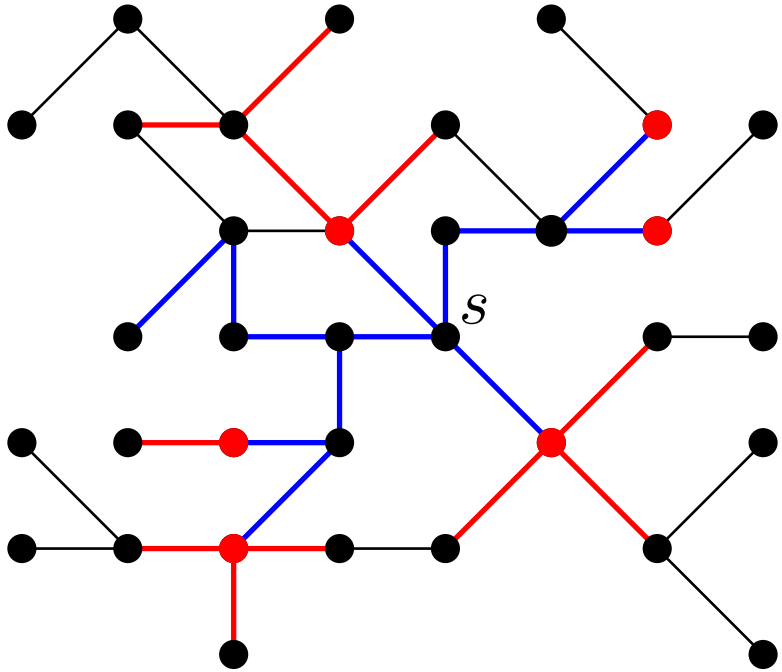
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



breche ab, wenn die Queue nur noch Knoten enthält die über rote Knoten erreicht wurden

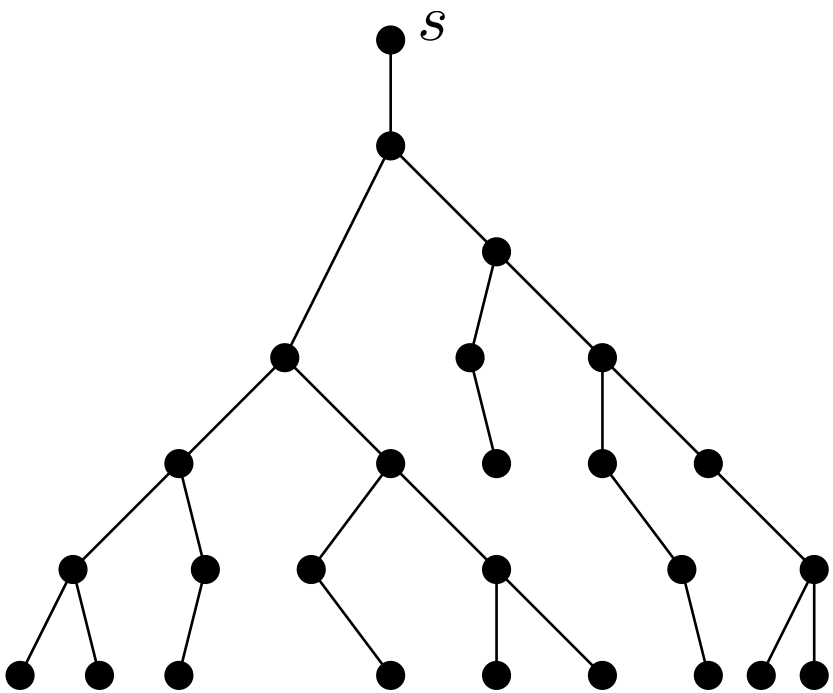
Neue Bäume Aufbauen [DGPW14]

- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf
- Wie machen wir das ohne uns den ganzen Graph anzuschauen?



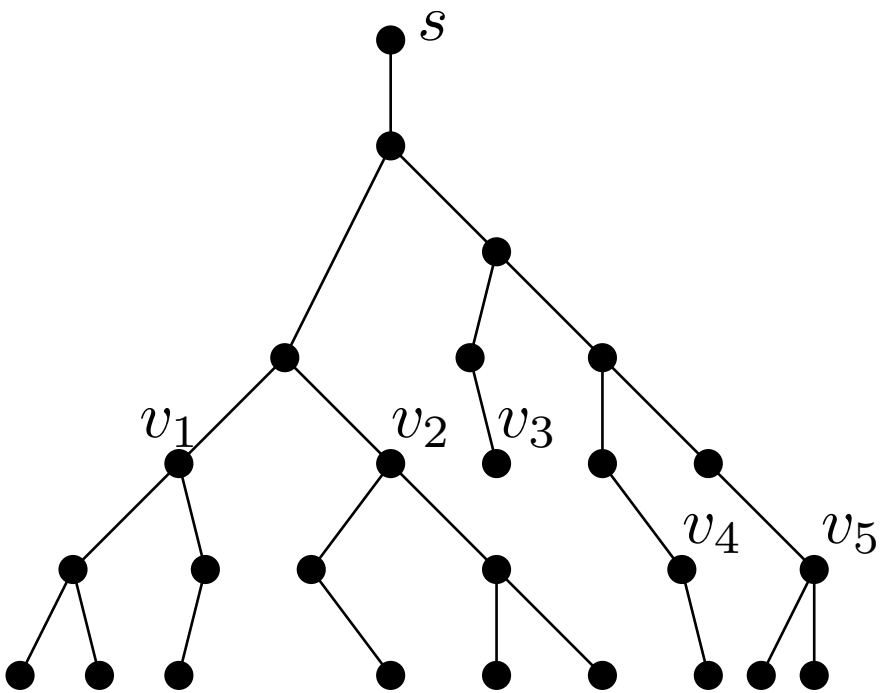
der blaue Teilgraph ist der neue Baum

Neue Bäume Aufbauen (alternaitve Sichtweise) [DGPW14]



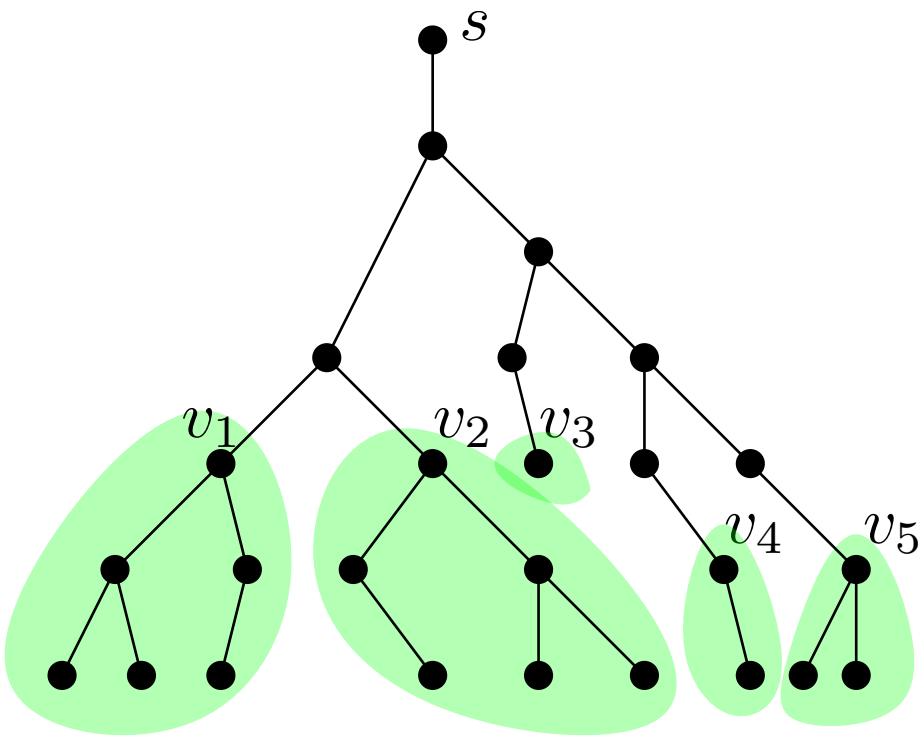
s ist neue Baumwurzel
Oben abgebildet ist der vollständige Kürzeste-Wege Baum

Neue Bäume Aufbauen (alternaitve Sichtweise) [DGPW14]



Die Knoten v_i wurden bereits ausgewählt

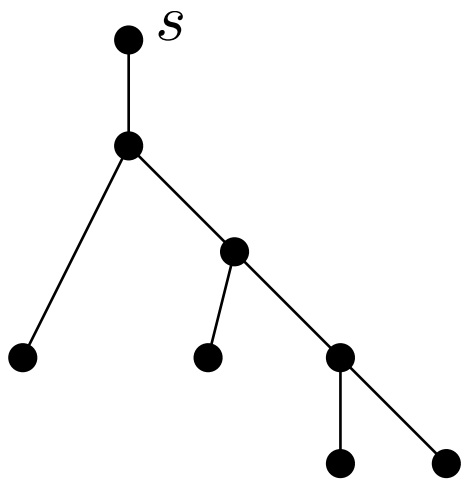
Neue Bäume Aufbauen (alternaitve Sichtweise) [DGPW14]



Ihre Teilbäume werden nicht gebraucht

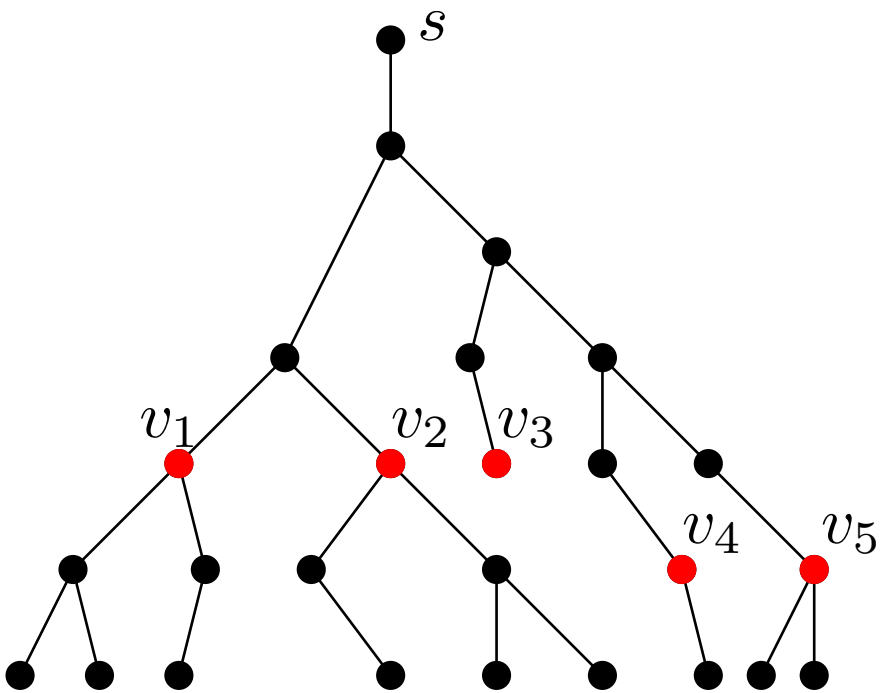
$\rightarrow$ Wir wollen diese Teile gar nicht erst aufbauen

Neue Bäume Aufbauen (alternaitve Sichtweise) [DGPW14]



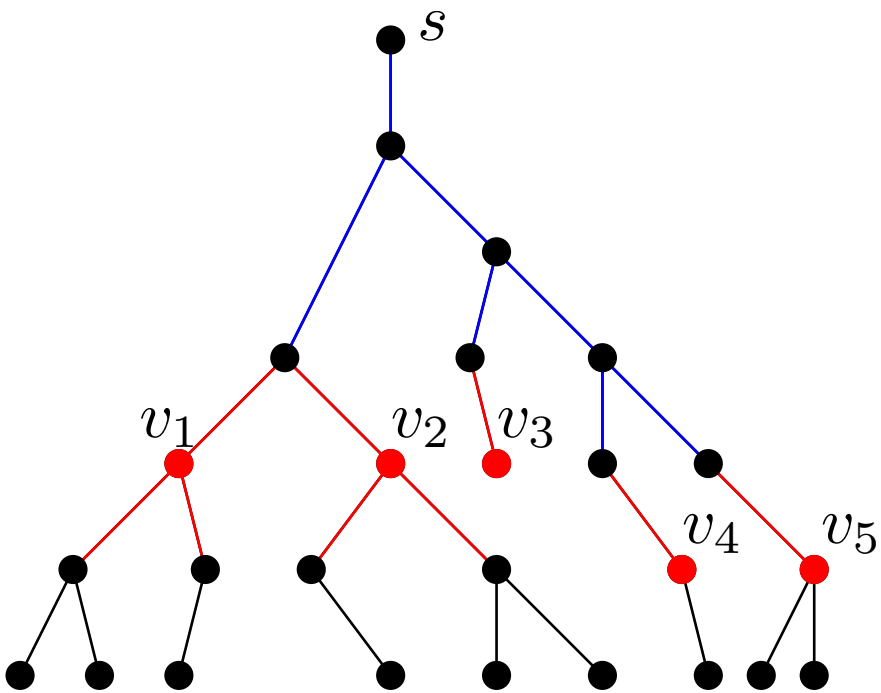
Dies ist der Baum den wir haben wollen

Neue Bäume Aufbauen (alternaitve Sichtweise) [DGPW14]



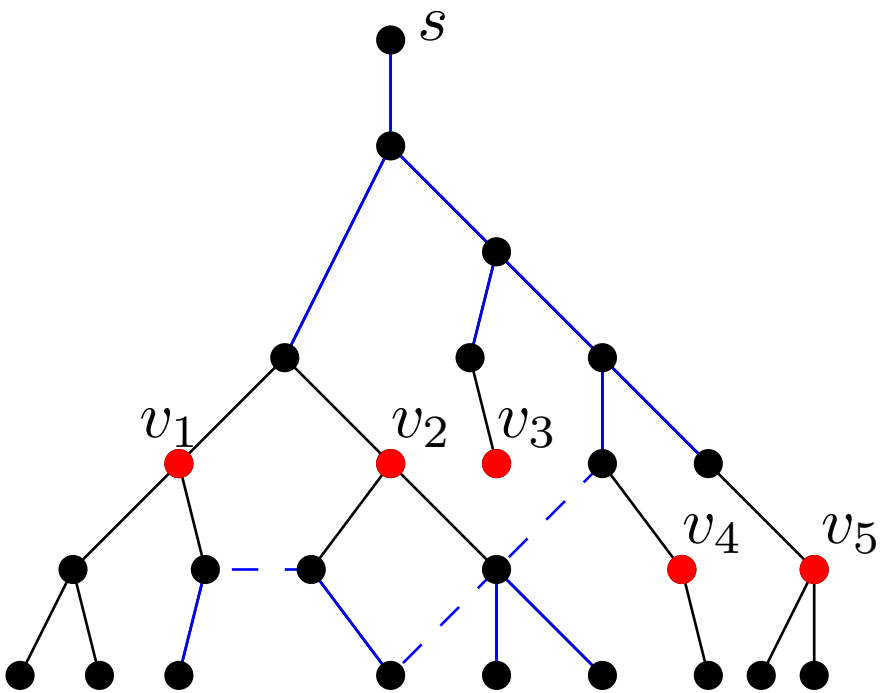
Die Knoten v_i sind bereits in der Ordnung und rot markiert

Neue Bäume Aufbauen (alternaitve Sichtweise) [DGPW14]



Um den Baum aufzubauen lassen wir Dijkstras Algorithmus von s aus laufen und brechen ab sobald alle Äste über rote Knoten gehen

Neue Bäume Aufbauen (alternaitve Sichtweise) [DGPW14]



Dies ist nicht das selbe wie an allen v_1 . . . zu prunen!
Wenn wir prunen findet die Suche neue Wege die gar nicht entlang des Kürzeste-Wege Baum von s entlang gehen.

Sampling Path-Greedy [DGPW14]

Algo:

1. Baue Bäume auf mit zufälliger Wurzel, bis $k \cdot n$ Knoten in allen Bäumen sind
2. Wähle v aus
3. Lösche Teilbäume unter v
4. Gehe zu 1

- k ist ein Parameter der die Qualität steuert
- um v schnell auszuwählen muss man an jedem Knoten speichern in welchen Bäumen er enthalten ist

Sampling Path-Greedy [DGPW14]

Vorteile von Sampling Path-Greedy:

- Funktioniert auf den meisten Graphen
  Auch auf Graphen mit hohen Knotengraden, wo Bottom-Up sich schwer tut

Auf Straße

- Ordnungsqualität vergleichbar mit Bottom-Up
- Aber langsamer als Bottom-Up
- $\rightarrow$ Nehmt Bottom-Up

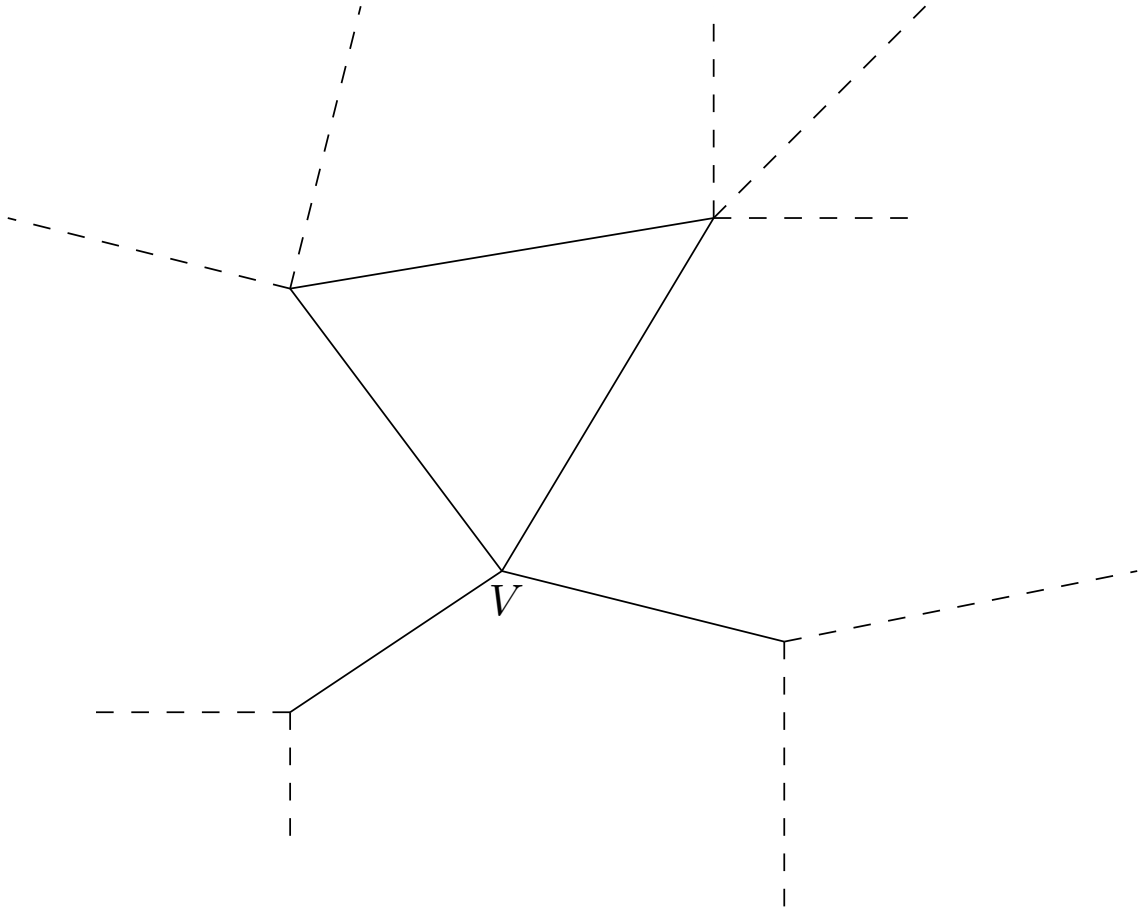
Customizable Contraction Hierarchies

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Customizable Contraction Hierarchies [DSW14, BCRW13, DSW16]

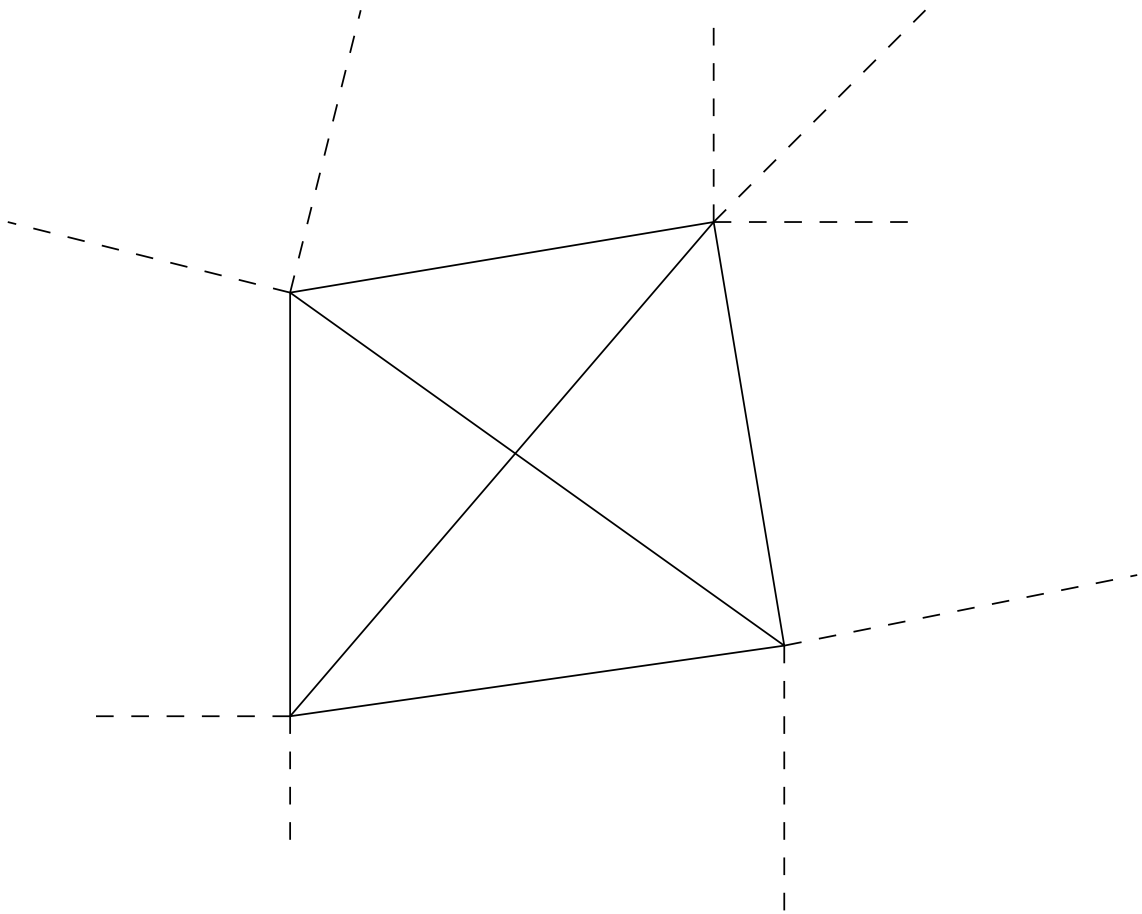
- 3-Phasen Ansatz
- Preprocessing-, Customization- und Query-Phase
- Das jetzt mit CHs

Ungewichtete Knoten-Contraction



Kontraktion von v: Ersetze v mit vollständiger Clique.

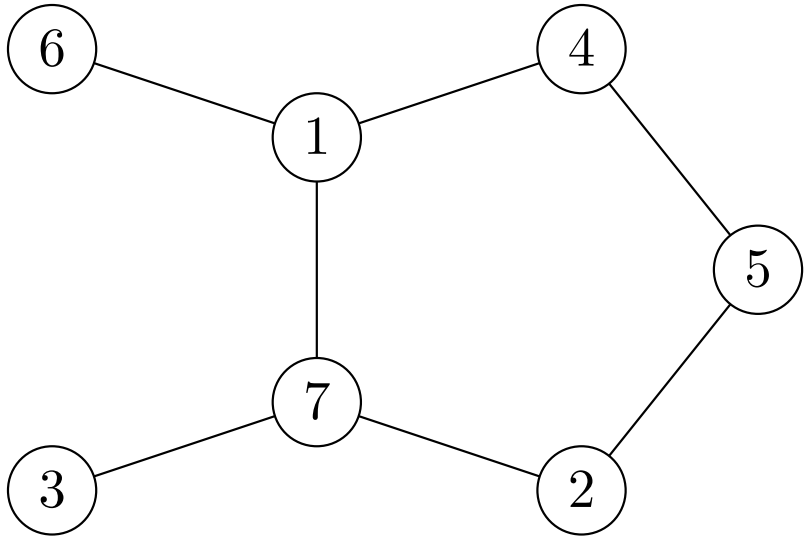
Ungewichtete Knoten-Contraction



Kontraktion von v: Ersetze v mit vollständiger Clique.

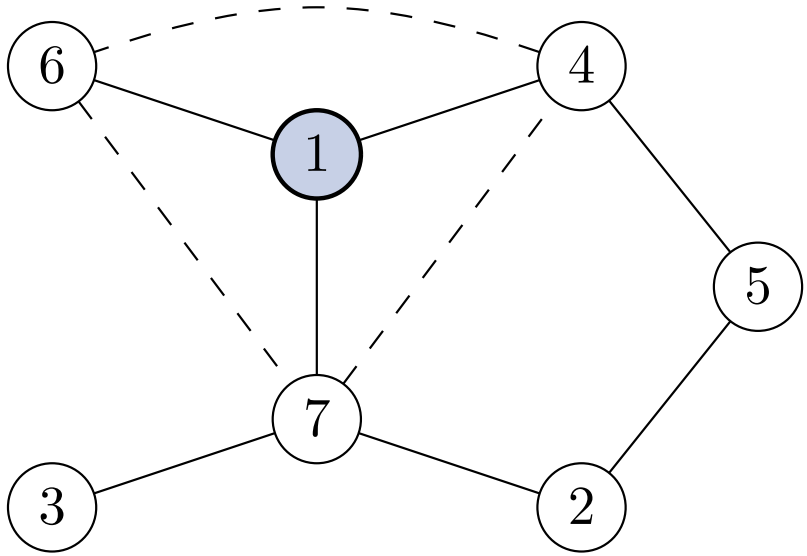
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Graph Fill-In / Ungewichtete Contraction Hierarchy



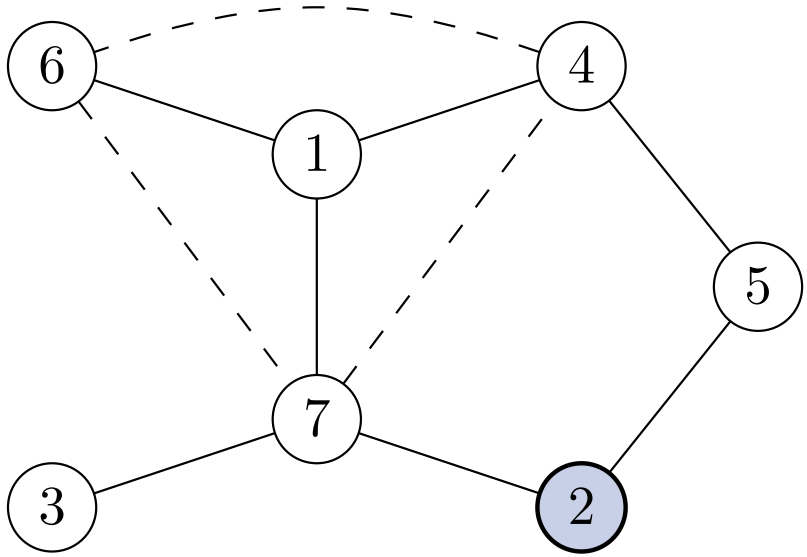
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



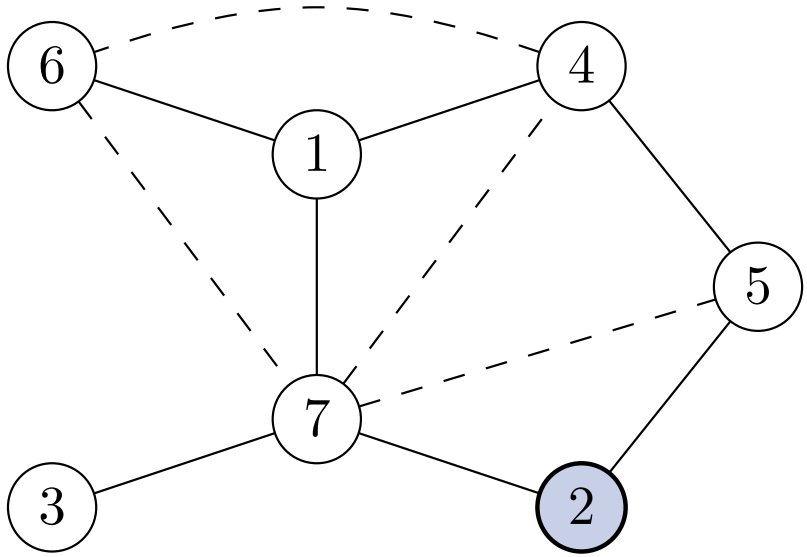
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



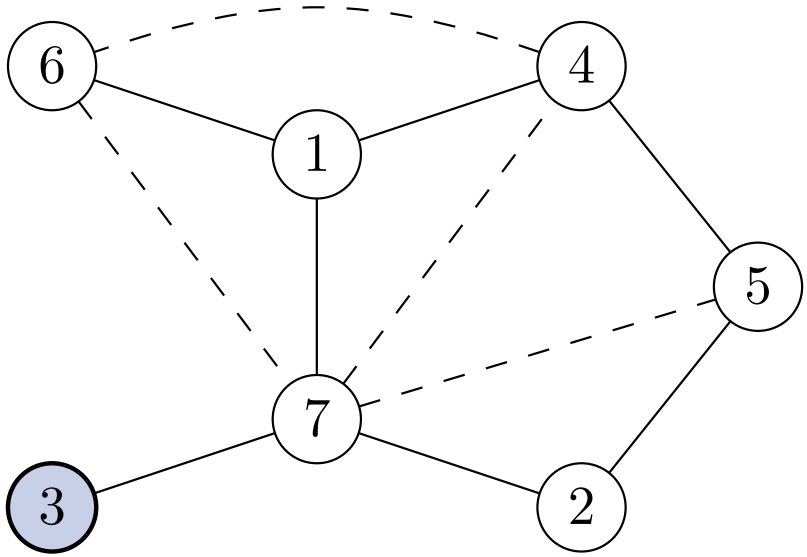
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



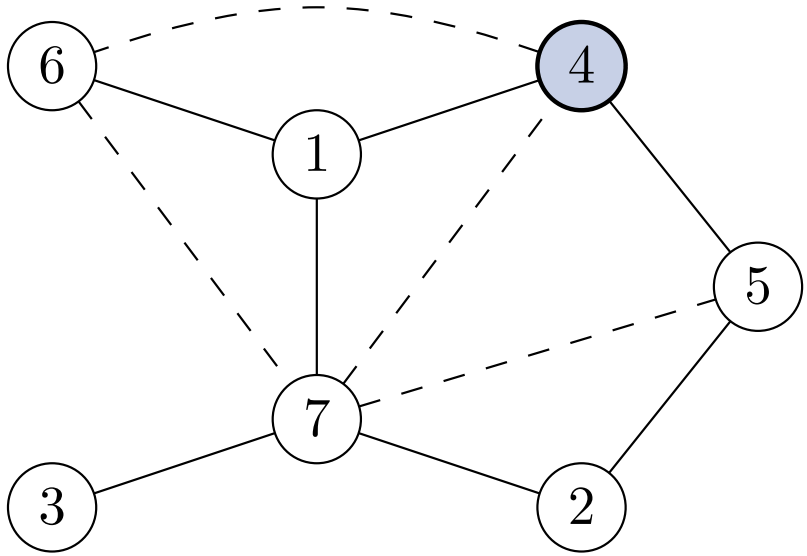
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



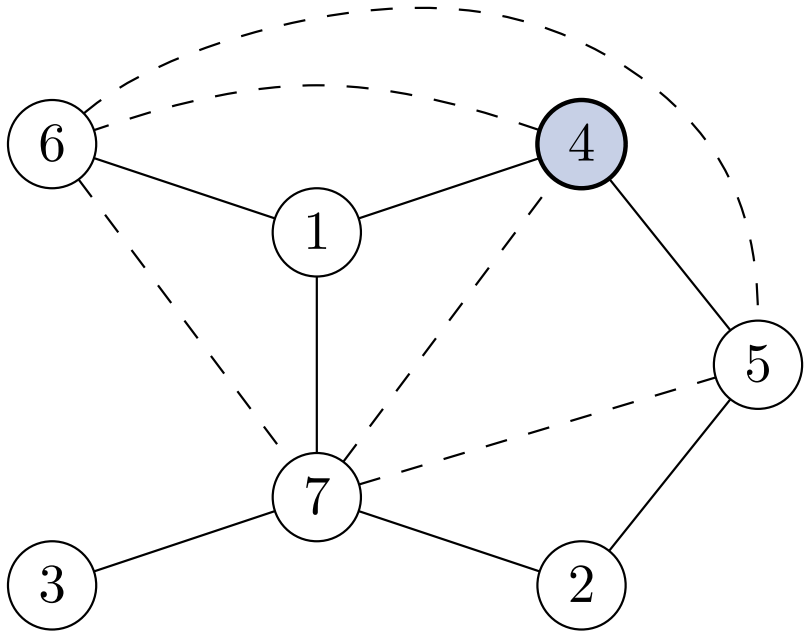
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



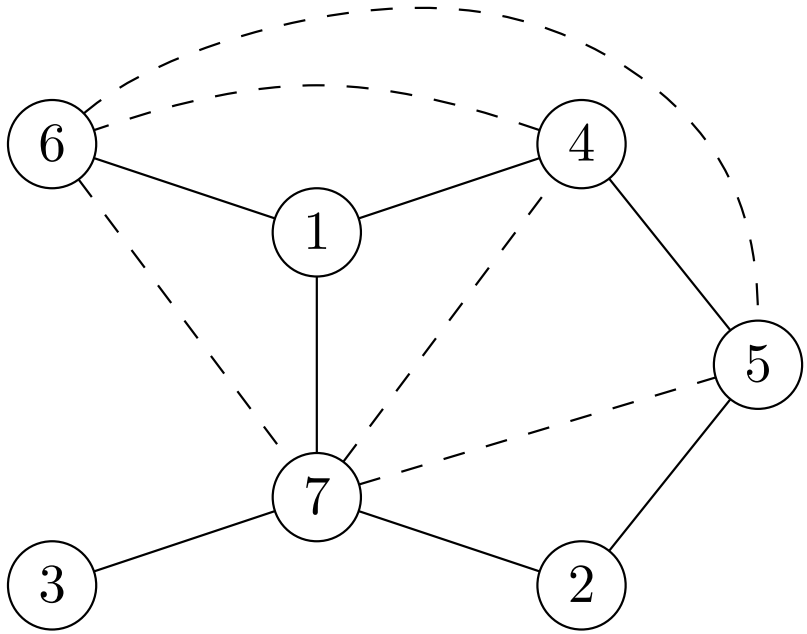
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



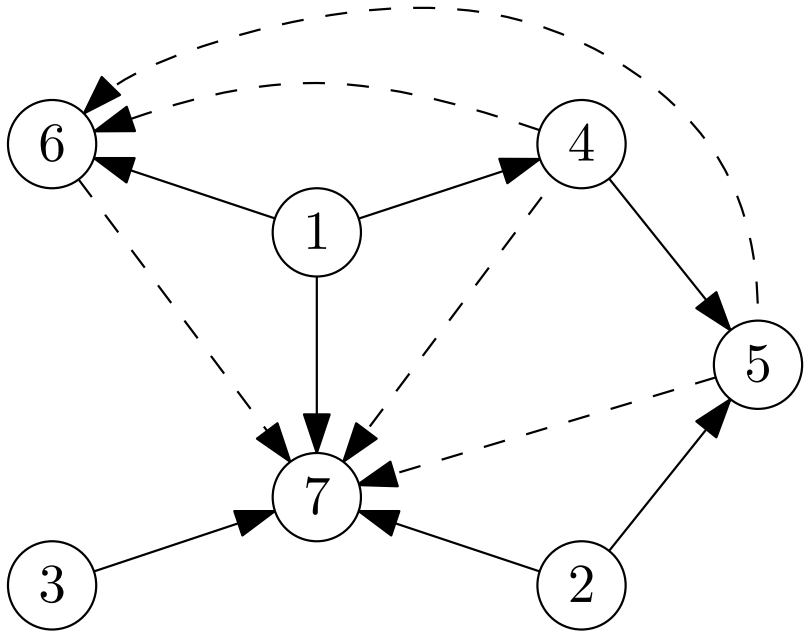
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



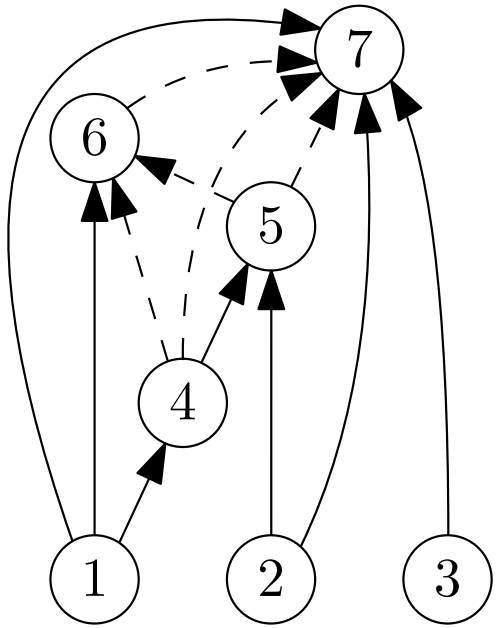
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}

Graph Fill-In / Ungewichtete Contraction Hierarchy



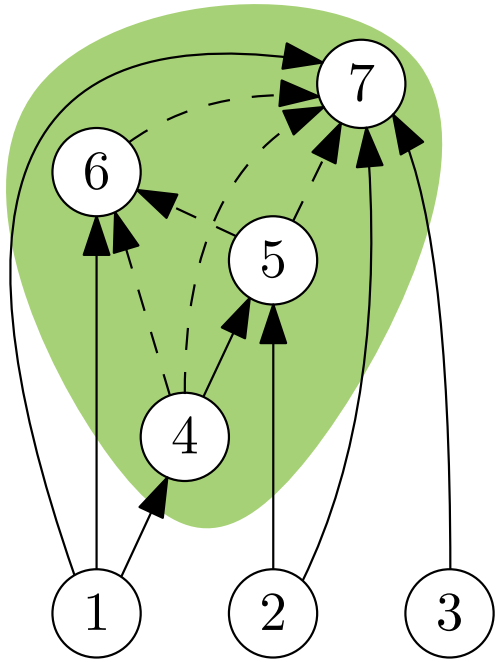
- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}
- Um den Suchgraph zu erhalten: Kanten nach Rank richten, d.h., aufwärts

Graph Fill-In / Ungewichtete Contraction Hierarchy



- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}
- Um den Suchgraph zu erhalten: Kanten nach Rank richten, d.h., aufwärts

Graph Fill-In / Ungewichtete Contraction Hierarchy



- Kontrahiere Knoten iterative
  Reihenfolge heißt π, Rank eines Knoten heißt π^{-1}
- Um den Suchgraph zu erhalten: Kanten nach Rank richten, d.h., aufwärts
- Suchraum von einem Knoten ist der erreichbare Subgraph
  (im Beispiel der Suchraum von Knoten 4)

Knoten Separatoren

Knoten Separator

Ein Knoten Separator S von $G = (V, E)$ ist eine Knotenmenge, so dass das Löschen von S den Graph G in zwei unzusammenhängende Teile $G_1 = (V_1, E_1)$ und $G_2 = (V_2, E_2)$ zerlegt.

Balancierter Knoten Separator

Zusätzlich muss gelten, dass $|V_1| \leq \frac{2}{3}n$ and $|V_2| \leq \frac{2}{3}n$.

Rekursive Balancierter Knoten Separator

Ein Graph G hat rekursive balancierte Knoten Separatoren der Größe $O(|V|^\alpha)$ wenn G einen balancierten Separator der Größe $O(|V|^\alpha)$ hat und G_1 und G_2 rekursive balancierte Separatoren der Größen $O(|V_1|^\alpha)$ und $O(|V_2|^\alpha)$ haben.

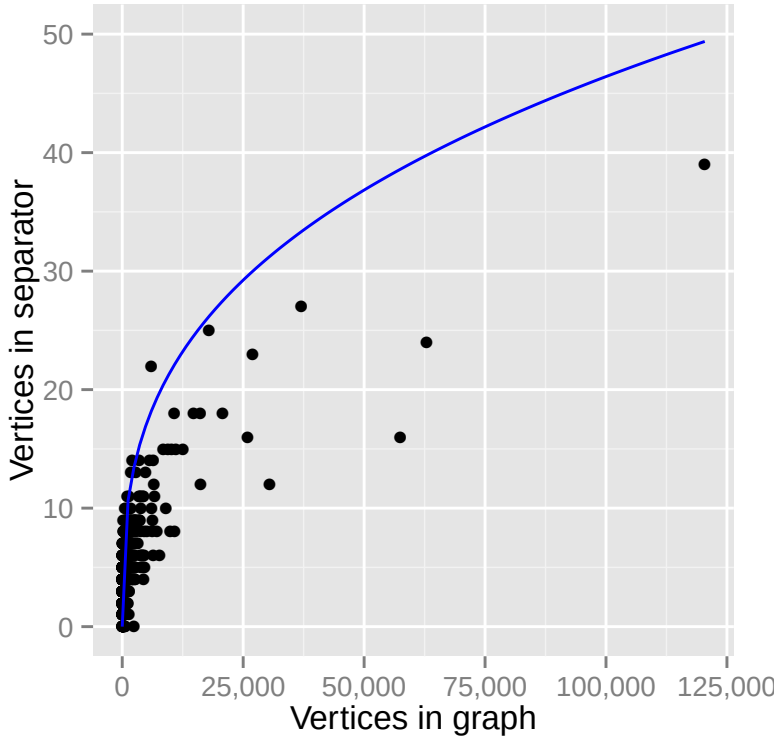
Knoten Separatoren

Beispiel

Alle planare Graphen haben $O(n^{\frac{1}{2}})$ rekursive balancierte Knoten Separatoren.

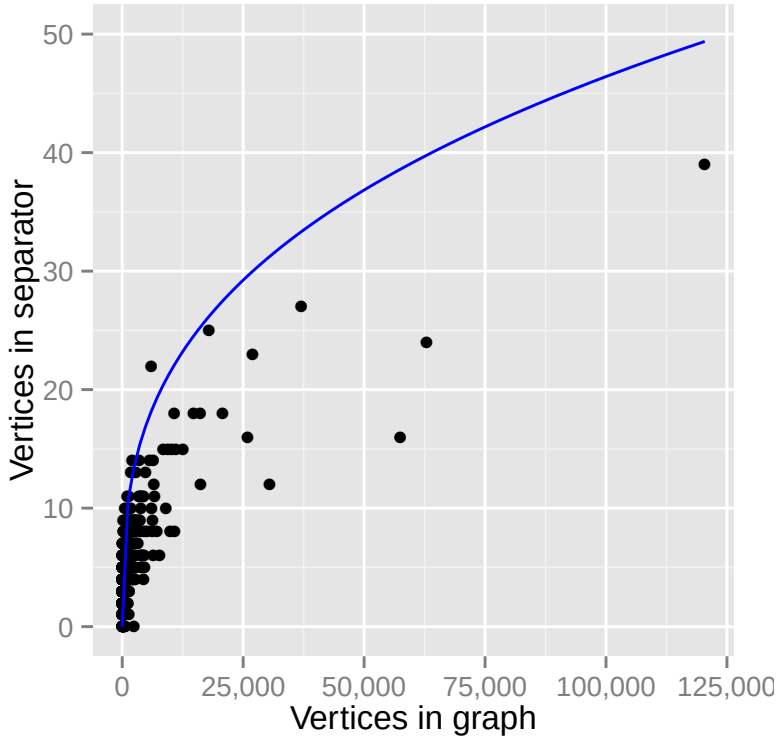
Beweis: Siehe Vorlesung zu planaren Graphen

Knoten Separatoren



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

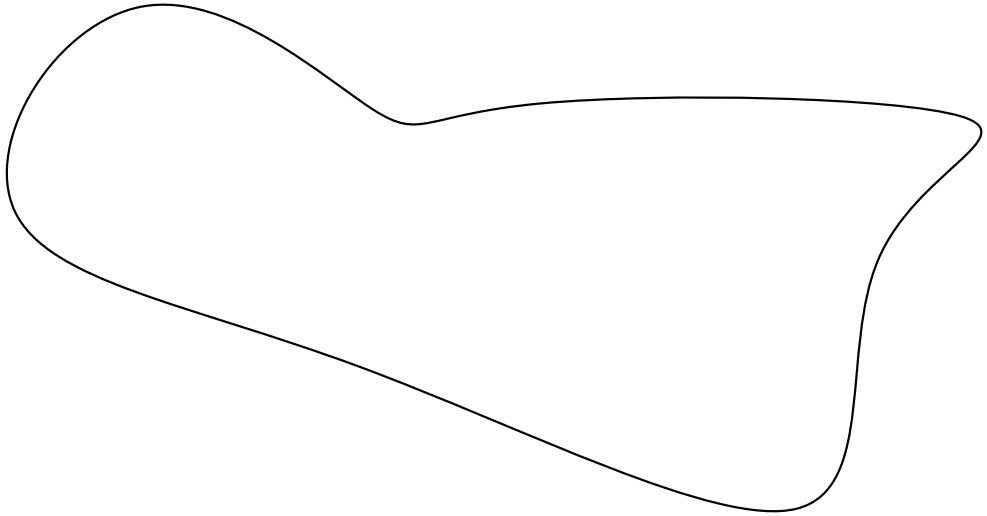
Knoten Separatoren



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

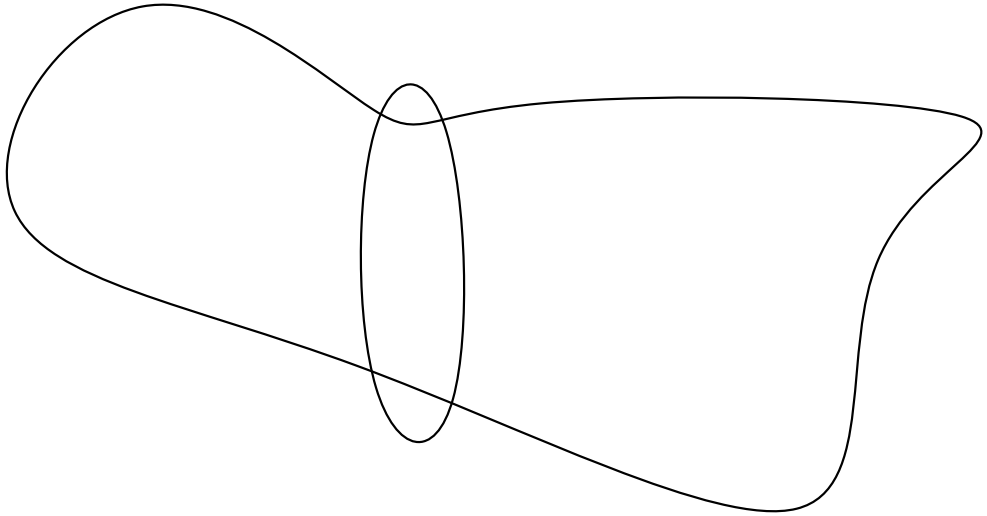
Annahme: Straßengraphen haben $O(\sqrt[3]{x})$ rekursive balancierte Separatoren.

Nested Dissection (ND)



Order:

Nested Dissection (ND)

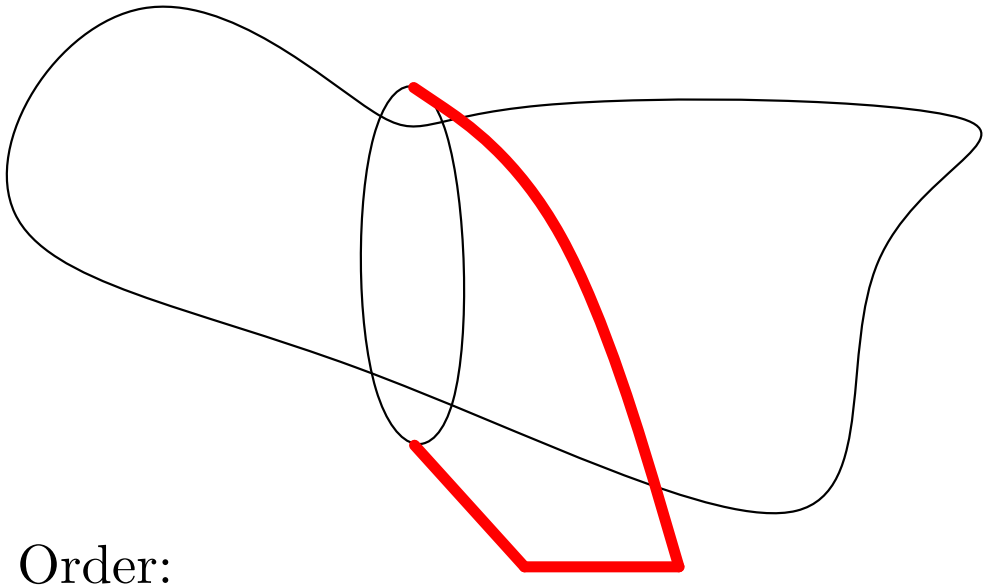


Order:

Finde kleinen Separator

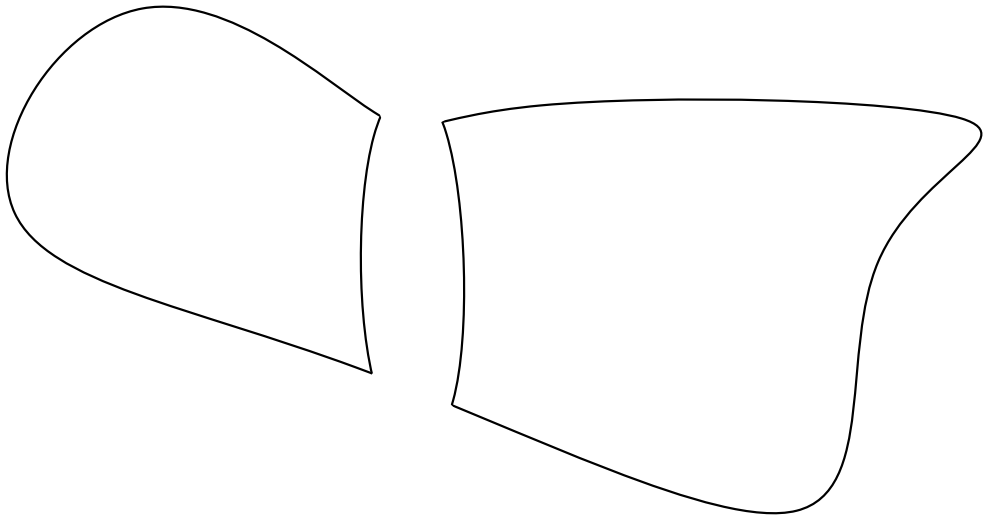
Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Nested Dissection (ND)



Pack die Knoten hinten in die Ordnung

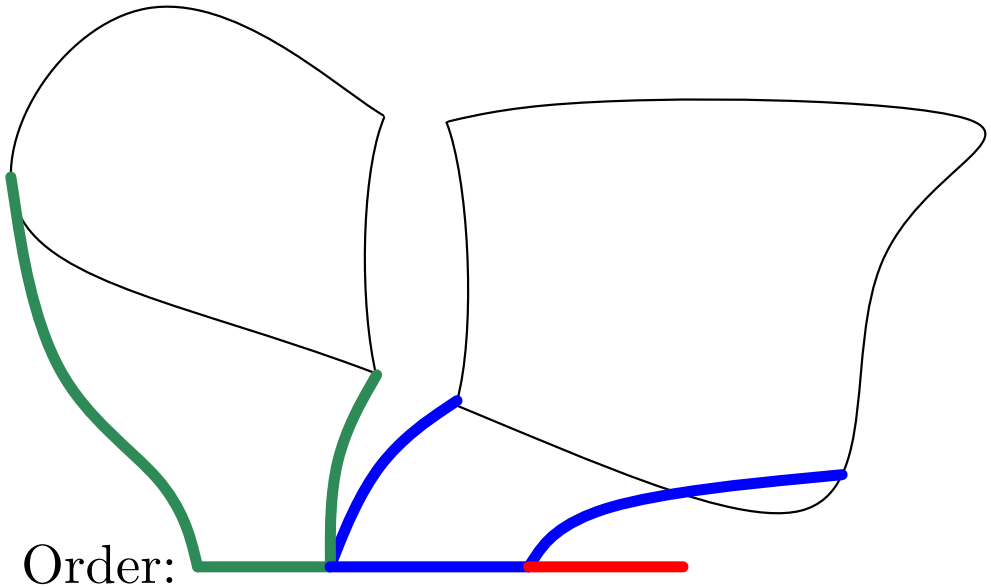
Nested Dissection (ND)



Order: 

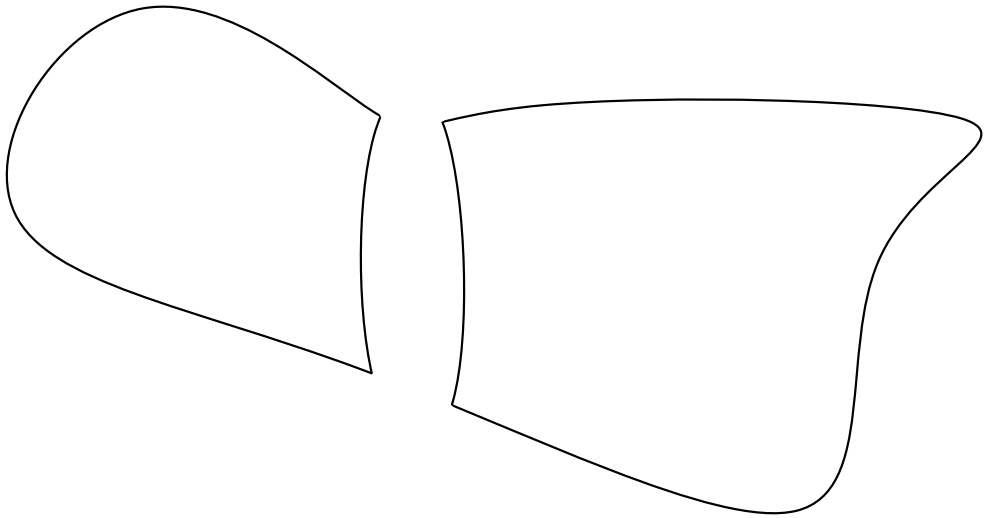
Lösche Separator

Nested Dissection (ND)



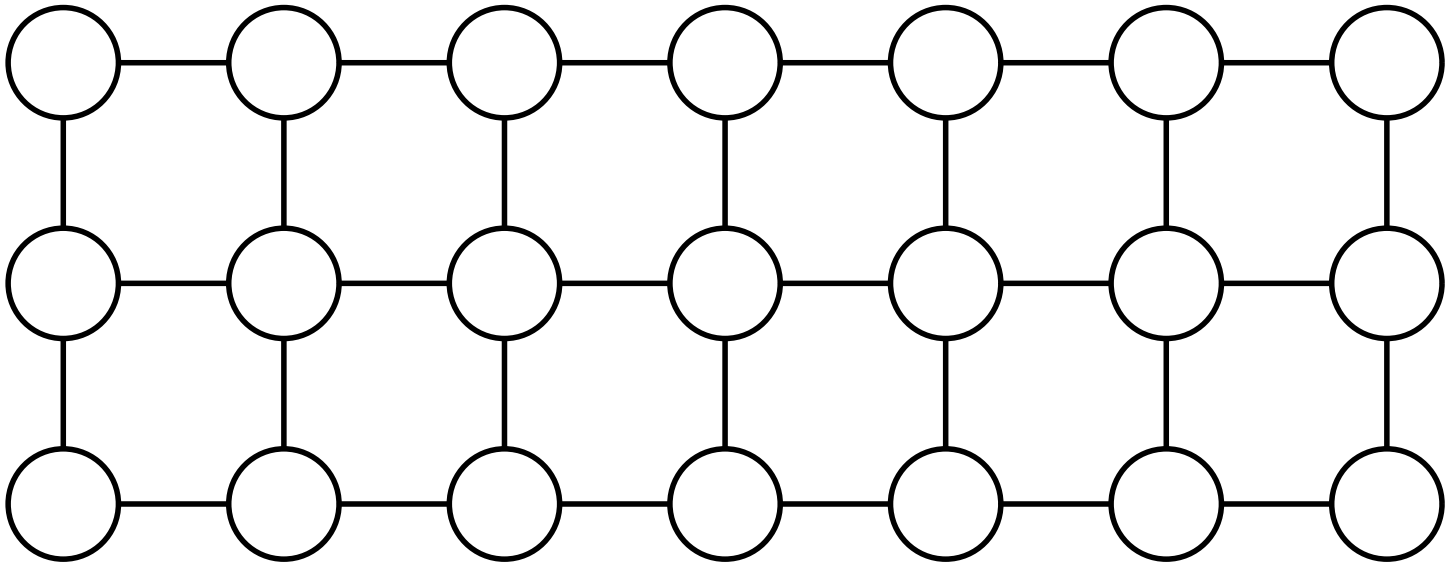
Rekursion auf beiden Teilen

Nested Dissection (ND)



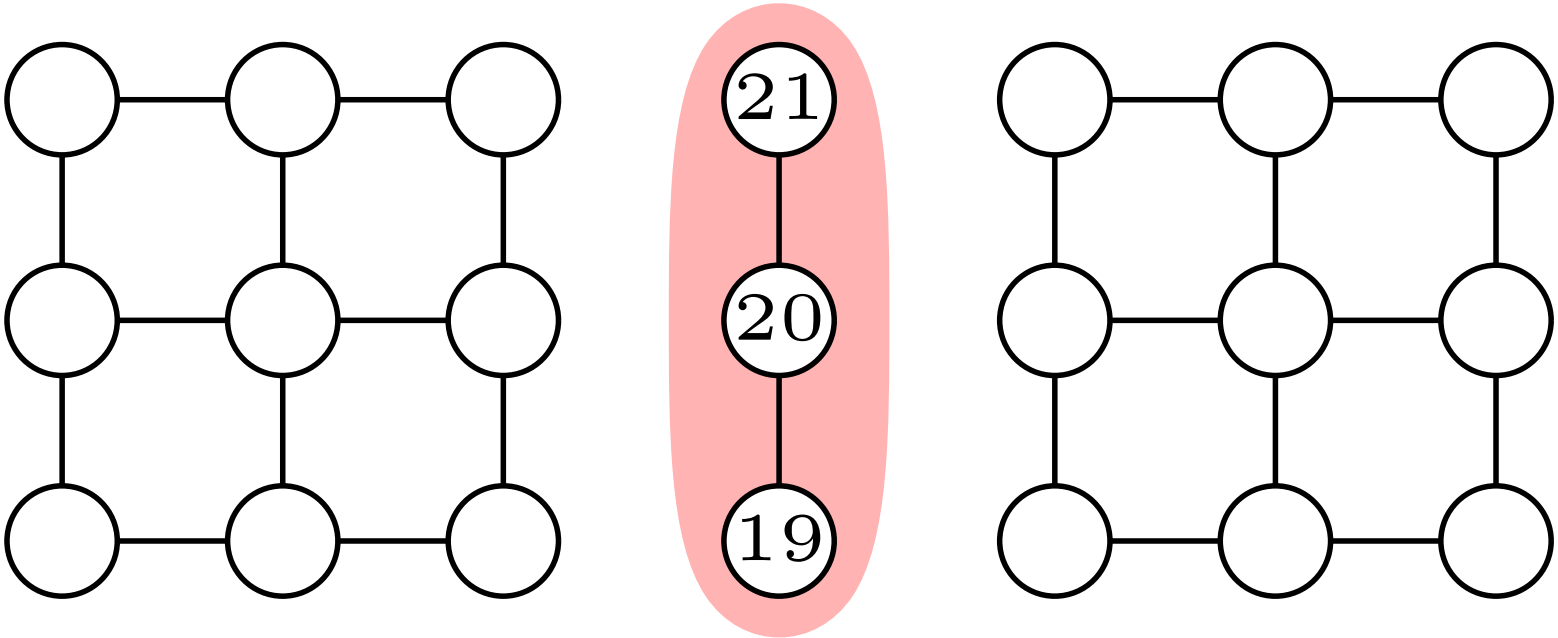
Die Ordnung

Nested Dissection (Beispiel)

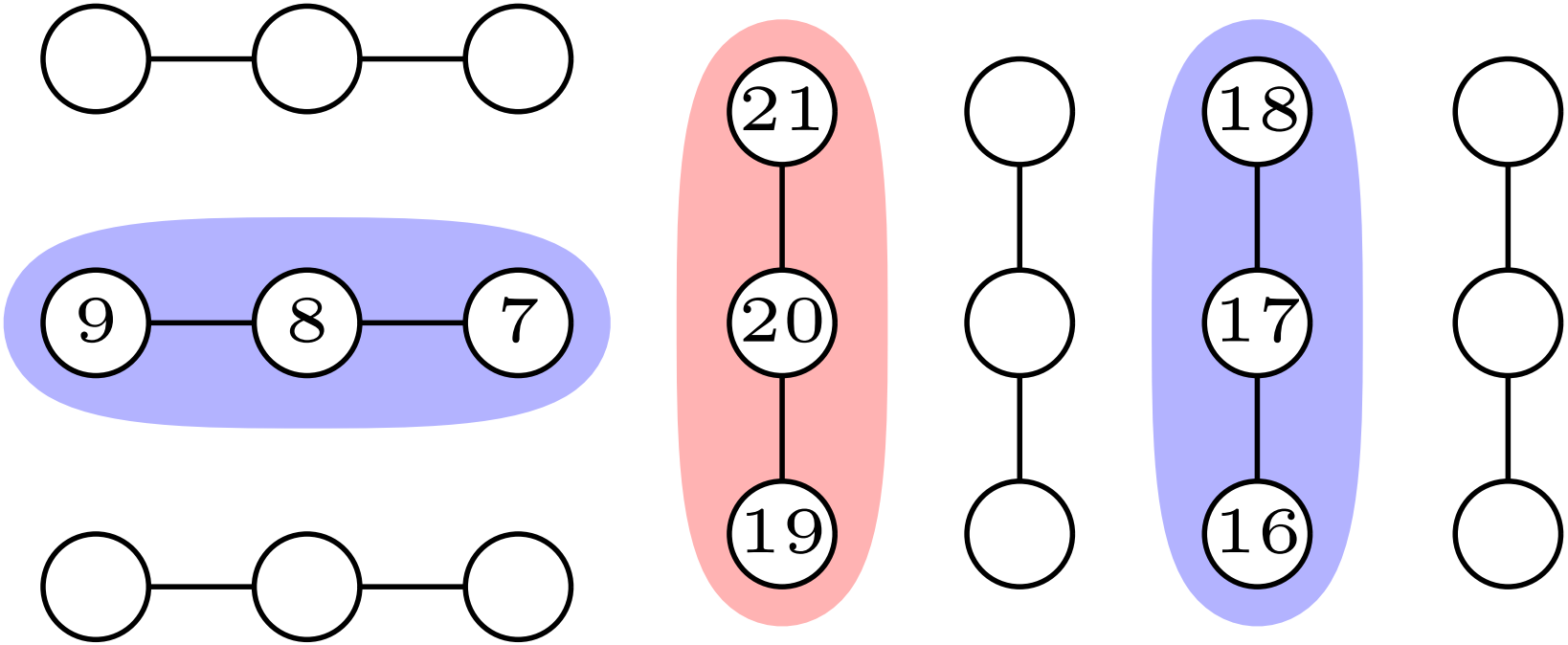


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

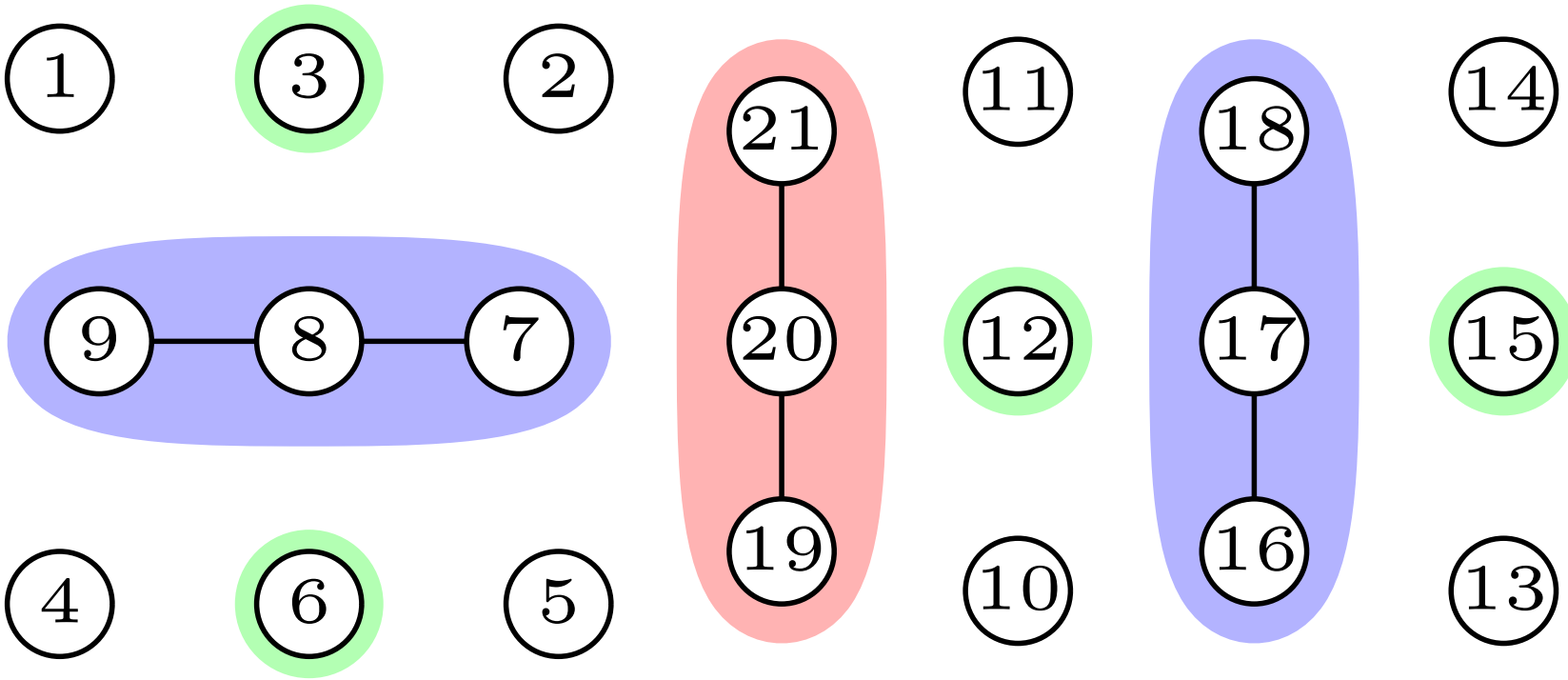
Nested Dissection (Beispiel)



Nested Dissection (Beispiel)



Nested Dissection (Beispiel)



Suchraum Größe

- **Ziel**: Zeige obere Schranke für Knoten im Suchraum

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Suchraum Größe

- **Ziel**: Zeige obere Schranke für Knoten im Suchraum
- Jeder Knoten v liegt in einem Separator
  (in der unteresten Ebende packen wir alles in den Separator)

Suchraum Größe

- **Ziel**: Zeige obere Schranke für Knoten im Suchraum
- Jeder Knoten v liegt in einem Separator
  (in der unteresten Ebende packen wir alles in den Separator)
- Es sei L die Tiefe dieses Separators
  Tiefe 0 = höchstes Level, Tiefe 1 = zweithöchstes. . .

Suchraum Größe

- **Ziel**: Zeige obere Schranke für Knoten im Suchraum
- Jeder Knoten v liegt in einem Separator
  (in der unteresten Ebende packen wir alles in den Separator)
- Es sei L die Tiefe dieses Separators
  Tiefe 0 = höchstes Level, Tiefe 1 = zweithöchstes. . .
- Auf jedem höheren Level liegt v auf einer Seite des Separators

Suchraum Größe

- **Ziel**: Zeige obere Schranke für Knoten im Suchraum
- Jeder Knoten v liegt in einem Separator
  (in der unteresten Ebende packen wir alles in den Separator)
- Es sei L die Tiefe dieses Separators
  Tiefe 0 = höchstes Level, Tiefe 1 = zweithöchstes...
- Auf jedem höheren Level liegt v auf einer Seite des Separators
- Nach Konstruktion gibt es keine aufwärts gerichteten Kanten zwischen den Seiten
  $\Rightarrow$ Auf Level i mit $0 \leq i < L$ liegen höchstens alle Separatorknoten von v im Suchraum

Suchraum Größe

- **Ziel**: Zeige obere Schranke für Knoten im Suchraum
- Jeder Knoten v liegt in einem Separator
  (in der unteresten Ebende packen wir alles in den Separator)
- Es sei L die Tiefe dieses Separators
  Tiefe 0 = höchstes Level, Tiefe 1 = zweithöchstes...
- Auf jedem höheren Level liegt v auf einer Seite des Separators
- Nach Konstruktion gibt es keine aufwärts gerichteten Kanten zwischen den Seiten
  $\Rightarrow$ Auf Level i mit $0 \leq i < L$ liegen höchstens alle Separatorknoten von v im Suchraum
- Auf Level L kommen nochmal höchstens alle Knoten vom Separator von v dazu

Suchraum Größe

- **Ziel**: Zeige obere Schranke für Knoten im Suchraum
- Jeder Knoten v liegt in einem Separator
  (in der unteresten Ebende packen wir alles in den Separator)
- Es sei L die Tiefe dieses Separators
  Tiefe 0 = höchstes Level, Tiefe 1 = zweithöchstes...
- Auf jedem höheren Level liegt v auf einer Seite des Separators
- Nach Konstruktion gibt es keine aufwärts gerichteten Kanten zwischen den Seiten
  $\Rightarrow$ Auf Level i mit $0 \leq i < L$ liegen höchstens alle Separatorknoten von v im Suchraum
- Auf Level L kommen nochmal höchstens alle Knoten vom Separator von v dazu
- Nach Voraussetzung ist die Größe eines Separators auf Tiefe j höchstens $O(((\frac{2}{3})^j n)^\alpha)$

Suchraum Größe

- Wir können nun die Anzahl an Knoten im Suchraum beschränken:

$$
\begin{aligned}
& \sum_{i=0}^{L} O\left(\left(\left(\frac{2}{3}\right)^{i} \cdot n\right)^{\alpha}\right) \\
= \quad & O\left(n^{\alpha} \sum_{i=0}^{L}\left(\left(\frac{2}{3}\right)^{\alpha}\right)^{i}\right) \\
\leq \quad & O\left(n^{\alpha} \sum_{i=0}^{\infty}\left(\left(\frac{2}{3}\right)^{\alpha}\right)^{i}\right) \\
= \quad & O\left(n^{\alpha}\right)
\end{aligned}
$$

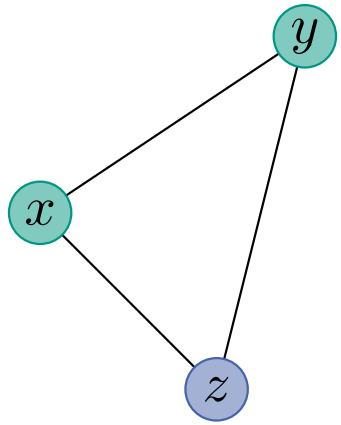
- Die Anzahl an Kanten im Suchraum ist demnach durch $O(n^{2\alpha})$ beschränkt

Nested Dissection

- Der Hauptvorteil von Nested Dissection ist das die Ordnung nicht von den Kantengewichten abhängt
- Customization muss weder die Ordnung noch den Suchgraph ändern

- Aber: mehr Kanten im Suchraum verglichen mit metrikabhängigen Ordnungen

Dreiecke



Ein unteres Dreieck von (x, y) ist ein Dreieck $\{x, y, z\}$ so dass z vor x und y kontrahiert werden.

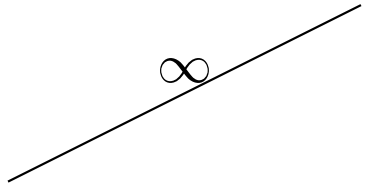
Customization

- Eine CH-Anfrage ist korrekt wenn für jedes untere Dreieck die untere Dreiecksungleichung gilt, d.h., für alle Kanten (x, y) im Suchgraph muss gelten, dass für alle unteren Dreiecke $\{x, y, z\}$

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

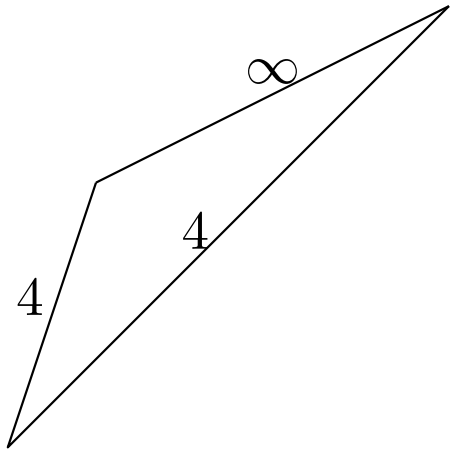
  gilt
- Initial bekommt jede original Kante ihr original Gewicht und alle Shortcuts kriegen Gewicht ∞
- Ziel der Customization ist es die untere Dreiecksungleichung herzustellen

Basic Customization



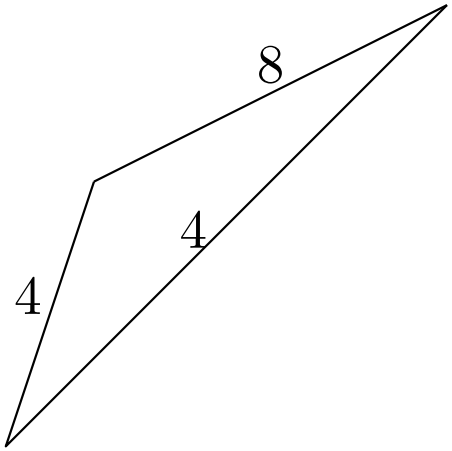
Eine Kante im Suchgraph

Basic Customization



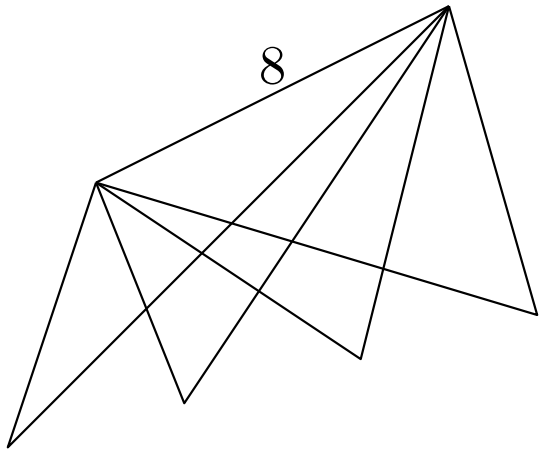
Für jedes untere Dreieck muss die Ungleichung gelten

Basic Customization



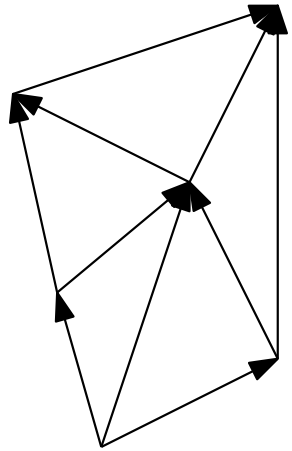
Verringere das Gewicht bis die Ungleichung gilt

Basic Customization



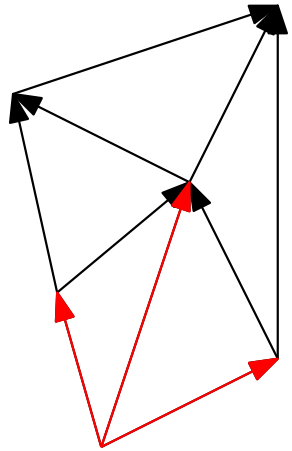
Dies muss für alle unteren Dreiecke gemacht werden

Basic Customization



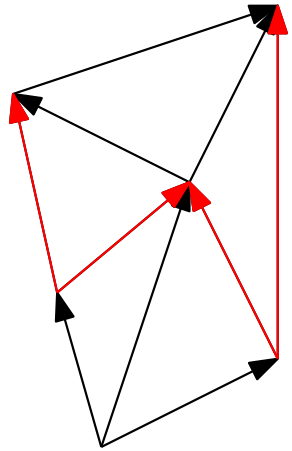
Bleibt die Frage in welcher Reihenfolge wir über die Kanten iterieren

Basic Customization



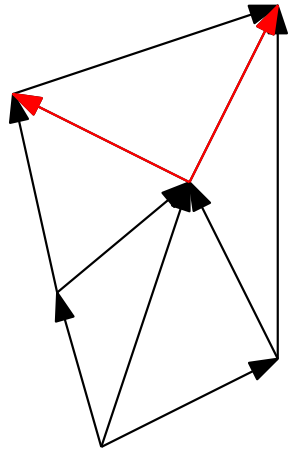
Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

Basic Customization



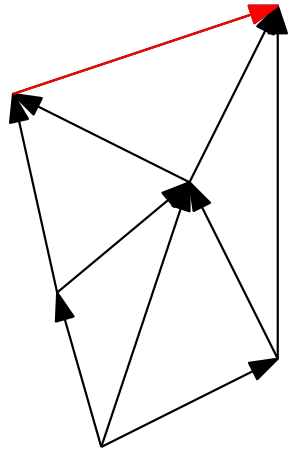
Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

Basic Customization



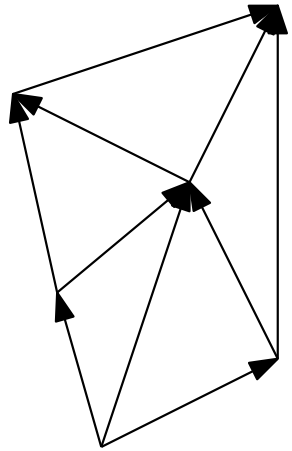
Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

Basic Customization



Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

Basic Customization



Die untere Dreiecksungleichung wurde hergestellt und damit sind die CH-Anfragen korrekt

Aufzählen aller unteren Dreiecke

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.

Elimination-Tree

Der Elimination-Tree ist wie folgt definiert:

$$\text{parent}(x) = \underset{(x,y) \text{ is in search graph}}{\arg\min} \left(\pi^{-1}(y)\right)$$

Satz: Vorfahren von x = Knoten im Suchraum von x

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Elimination-Tree

Der Elimination-Tree ist wie folgt definiert:

$$\text{parent}(x) = \underset{(x,y) \text{ is in search graph}}{\arg\min} \left(\pi^{-1}(y)\right)$$

Satz: Vorfahren von x = Knoten im Suchraum von x

- $V(x)$ = Vorfahren von x
- $S(x)$ = Knoten im Suchraum von x

Beweis:

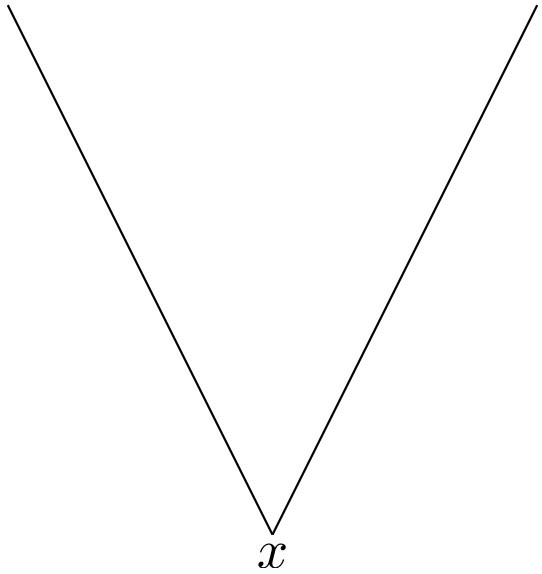
- $V(x) \subseteq S(x)$
- Klar, der Pfad entlang des Baums im Suchraum ist
- Bleibt zu zeigen, dass $V(x) \supseteq S(x)$
- Beweis per Widerspruch

Elimination-Tree

Der Elimination-Tree ist wie folgt definiert:

$$\text{parent}(x) = \underset{(x,y) \text{ is in search graph}}{\arg\min} \left(\pi^{-1}(y)\right)$$

Satz: Vorfahren von x = Knoten im Suchraum von x



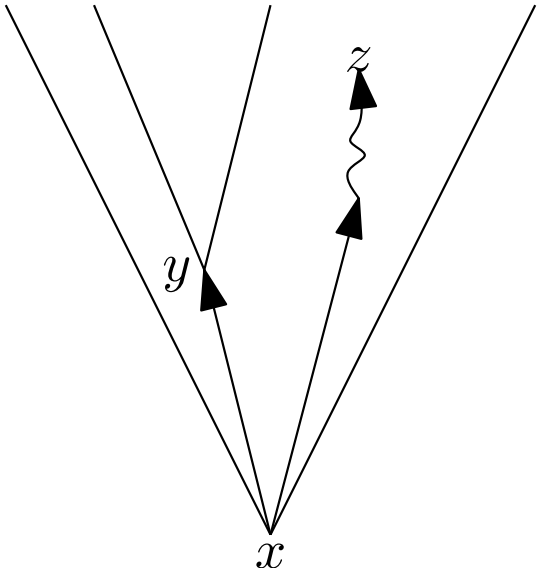
Der Suchraum von x.

Elimination-Tree

Der Elimination-Tree ist wie folgt definiert:

$$\text{parent}(x) = \underset{(x,y) \text{ is in search graph}}{\arg\min} \left(\pi^{-1}(y)\right)$$

Satz: Vorfahren von x = Knoten im Suchraum von x



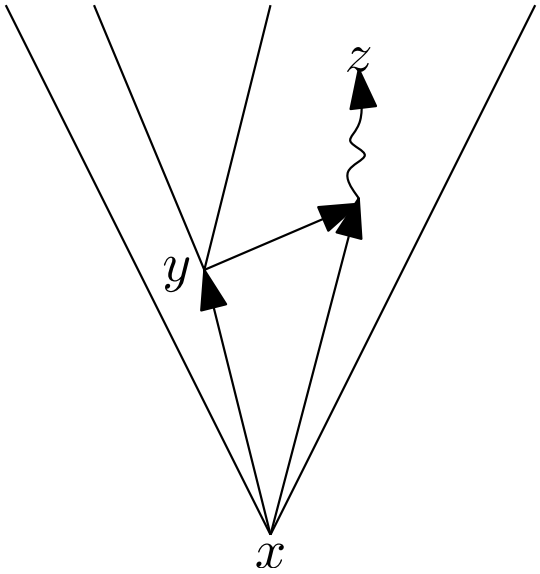
Sei $\text{parent}(x) = y$.
Annahme: z im Suchraum von x aber nicht im Suchraum von y ist.

Elimination-Tree

Der Elimination-Tree ist wie folgt definiert:

$$\text{parent}(x) = \underset{(x,y) \text{ is in search graph}}{\arg\min} \left(\pi^{-1}(y)\right)$$

Satz: Vorfahren von x = Knoten im Suchraum von x



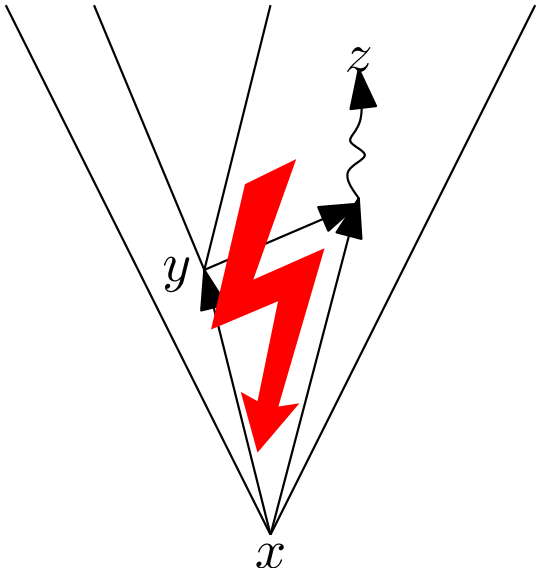
Diese Kante muss es per Konstruktion geben.

Elimination-Tree

Der Elimination-Tree ist wie folgt definiert:

$$\text{parent}(x) = \underset{(x,y) \text{ is in search graph}}{\arg\min} \left(\pi^{-1}(y)\right)$$

Satz: Vorfahren von x = Knoten im Suchraum von x



z muss im Suchraum von y sein. Widerpruch.

Elimination-Tree-Query

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
- Such immer den gesamten Suchraum ab

Pfad Entpacken

Algo:

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

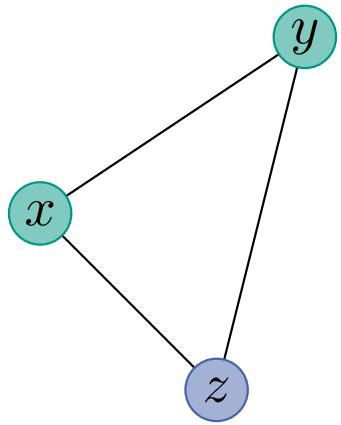
Gerichtete Graphen

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

Perfekte Customization

- Ziel der perfekte Customization ist es jede Kante (x, y) in der CH mit der Länge eines kürzestens xy-Pfads zu gewichten
- Dies erlaubt eine perfekte Zeugensuche

Dreiecke



- Ein mittleres Dreieck von (z, y) ist ein Dreieck $\{x, y, z\}$ so dass z vor x und x vor y kontrahiert wird
- Ein oberes Dreieck von (z, x) ist ein Dreieck $\{x, y, z\}$ so dass y nach z und x kontrahiert wird

Perfekte Customization

Algorithmus

- Führe Basic Customization durch
- Für alle Kanten (x, y) absteigend nach Level:
  - Zähle alle obere und mittleren Dreiecke $\{x, y, z\}$ auf (geht analog zu unteren Dreiecken)
  - Wenn $w(x, z) + w(z, y)$ kleiner ist als $w(x, y)$ dann verbessere $w(x, y)$
- **Eigenschaft**: Nun ist die volle Dreiecksungleichung erfüllt, d.h., für jedes Dreieck $\{x, y, z\}$ gilt $w(x, y) \leq w(x, z) + w(z, y)$
- Lösche alle Kanten deren Gewicht sich seit der Basic Customization verändert hat

Perfekte Customization

Korrektheit

- Zwei Teile:
  - Warum ist $w(x, y)$ die Distanz eines kürzesten xy-Pfads?
  - Warum dürfen wir (x, y) löschen falls sich $w(x, y)$ ändert?

Korrektheit Teil 1

- zu zeigen: $w(x, y)$ die Distanz eines kürzesten xy-Pfads
- Da wir erst eine Basic Customization durchführen gibt es einen kürzesten hoch-runter Pfad P
- Fall 1: P hat nur die Kante $(x, y) \rightarrow$ nix zu tun
- Fall 2: P hat mehere Kanten.
  - Dann gibt es den Knoten z der direkt nach x auf P kommt.
  - Wegen Knotenkontraktion muss es eine Kante zwischen z und y geben.
  - Der Rest des Beweis geht per Induktion
  - Die Kante zwischen z und y muss vor (x, y) abgearbeitet worden sein da sie höher ist. Wir können also die Induktion auf die anwenden.
  - Sie hat also schon das kürzeste Wege Gewicht (nach Induktionsvoraussetzung)
  - Der Pfad $x \rightarrow z \rightarrow y$ ist also ein kürzester
  - Je nachdem ob $z < y$ oder nicht ist er ein mittleres oder oberes Dreieck.

Korrektheit Teil 2

- zu zeigen: wir (x, y) löschen falls sich $w(x, y)$ ändert
- Nach der Basic Customization muss zwischen jedem st-Paar einen kürzesten hoch-runter Pfad Q gegeben haben
- Da jeder Teilpfad auch ein kürzester Pfad ist, kann sich kein Gewicht auf Q verändert haben
- $\Rightarrow$ Q ist immer noch da wenn wir (x, y) löschen

Korrektheit Teil 2

- zu zeigen: wir (x, y) löschen falls sich $w(x, y)$ ändert
- Nach der Basic Customization muss zwischen jedem *st*-Paar einen kürzesten hoch-runter Pfad Q gegeben haben
- Da jeder Teilpfad auch ein kürzester Pfad ist, kann sich kein Gewicht auf Q verändert haben
- $\Rightarrow$ Q ist immer noch da wenn wir (x, y) löschen

Bei Graphen mit eindeutigen kürzesten Wegen:

- Wir dürfen auch keine weiteren Kanten löschen
- Beweis nicht in der Vorlesung

Experimente

Durchschnittliche Laufzeit von Reisezeitanfragen

- CCH-Dijkstra : 0.81 ms
- CCH-Stall : 0.85 ms
- CCH-Tree : 0.41 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
- CH-Dijkstra : 2.66 ms
- CH-Stall : 0.54 ms

as always: instance is DIMACS Europe
sequential unless mentioned

Experimente

Customization

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

as always: instance is DIMACS Europe
sequential unless mentioned

Experimente

Nested Dissection Ordnung

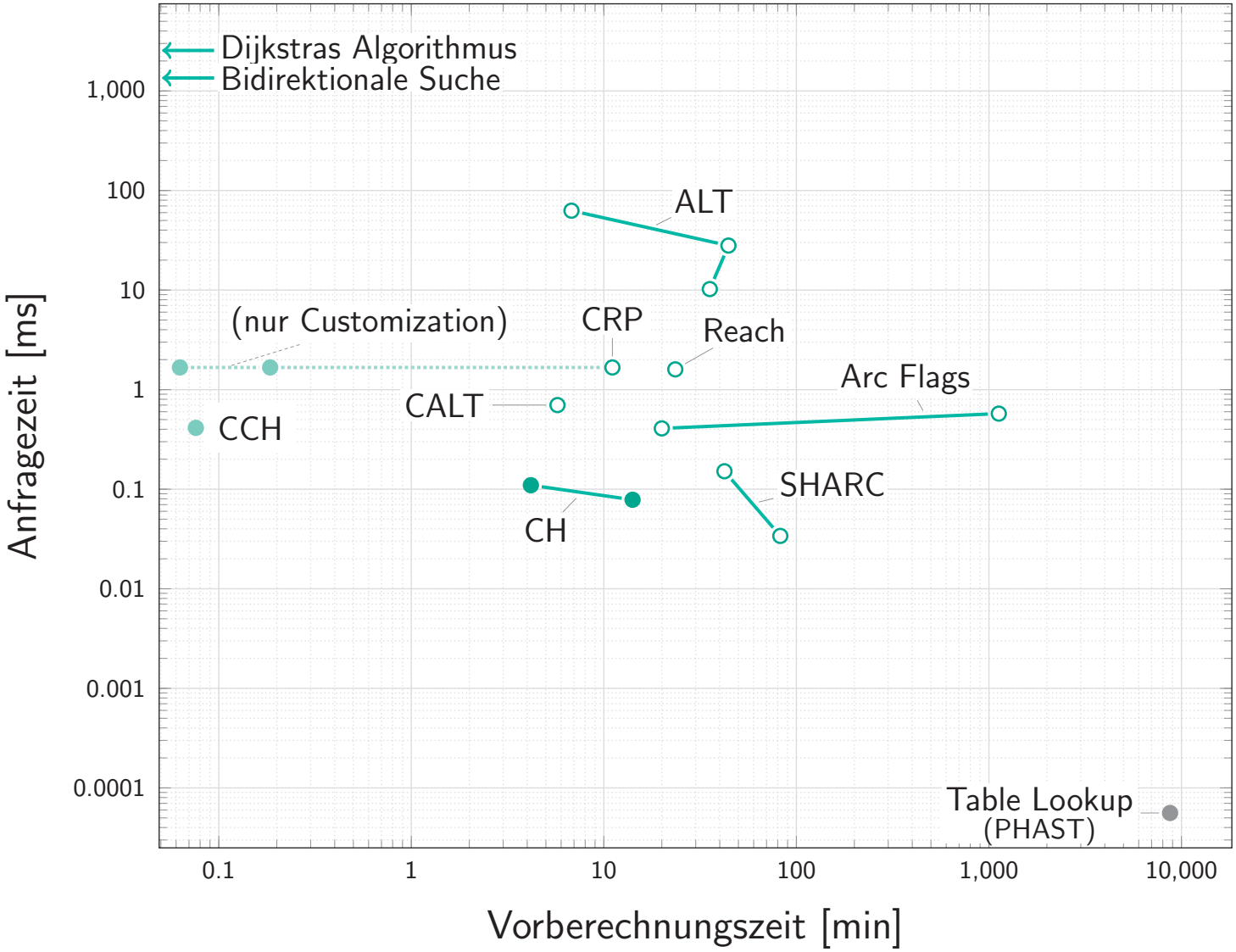
- Hängt vom verwendeten Partitionierer ab
- KaHip gute Qualität: <2.8 days (nicht auf Schnelligkeit optimiert)
- FlowCutter gute Qualität: 4.6h
- InertialFlow brauchbare Qualität: 17min
- Metis Quick&Dirty: 2.2 min

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])

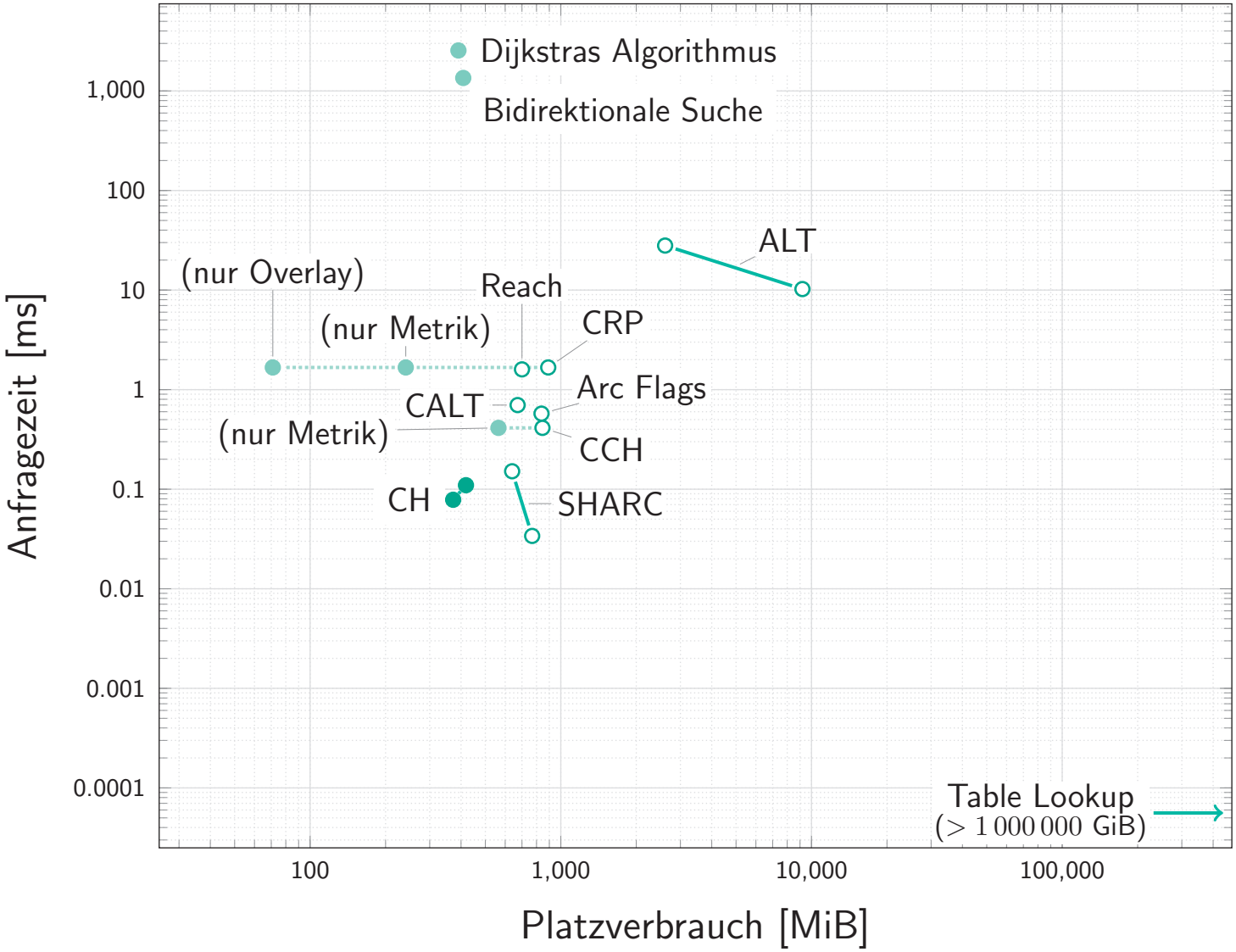
as always: instance is DIMACS Europe
sequential unless mentioned

Overview



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Overview


Dijkstras Algorithmus
Bidirektionale Suche
ALT
(nur Overlay)
Reach
(nur Metrik)
CRP
Arc Flags
CALT
(nur Metrik)
CCH
CH
SHARC
Table Lookup
(> 1 000 000 GiB)
Anfragezeit [ms]
1,000
100
10
1
0.1
0.01
0.001
0.0001
100
1,000
10,000
100,000
Platzverbrauch [MiB]

Dünnbesetzte Gleichungssysteme

Zu einem scheinbar ganz anderen Thema:
Lösen linear Gleichungssysteme

Dünnbesetzte Gleichungssysteme

Ziel: Obere Dreiecksmatrix per Gauselimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -2 \\ 1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -2 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$

4 Nullen im oberen Dreieck

Dünnbesetzte Gleichungssysteme

Ziel: Obere Dreiecksmatrix per Gauselimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 1 & -2 & -2 & -1 \\ 0 & -1 & 0 & -1 & 1 \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

Dünnbesetzte Gleichungssysteme

Ziel: Obere Dreiecksmatrix per Gauselimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{1}{2} \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

Dünnbesetzte Gleichungssysteme

Ziel: Obere Dreiecksmatrix per Gauselimination

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & 0 & -1 & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{3} \end{bmatrix}$$

Keine Nullen übrig $\rightarrow$:(

Dünnbesetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -2 \\ 1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -2 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$

Dünnbesetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ -2 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

Dünnbesetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Dünnbesetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Dünnbesetzte Gleichungssysteme

Idee: Spalten und Zeilen umsortieren

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Alle 4 Nullen erhalten $\rightarrow$:)

Dünnbesetzte Gleichungssysteme

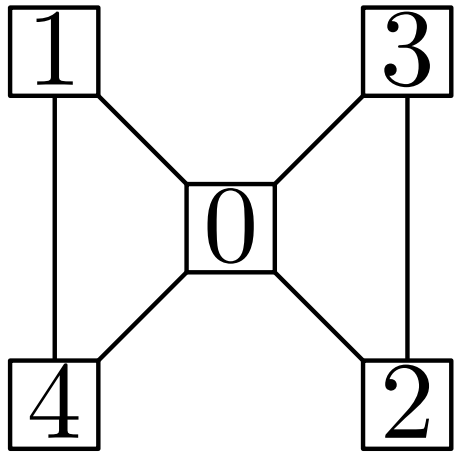
Warum funktioniert das?

Institut für Theoretische Informatik
Lehrstuhl Algorithmik

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

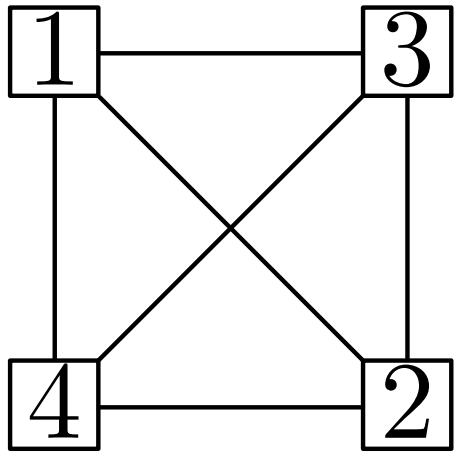
$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -2 \\ 1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -2 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$



Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 1 & -2 & -2 & -1 \\ 0 & -1 & 0 & -1 & 1 \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

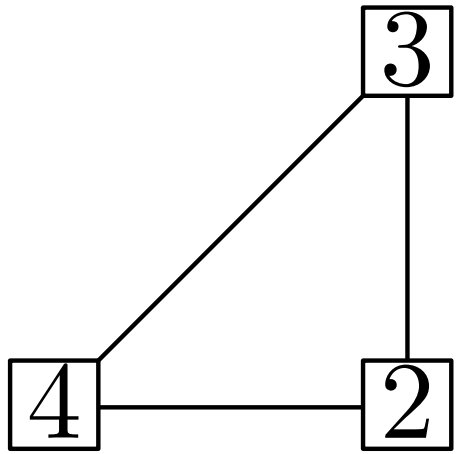


Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & -\frac{1}{2} & -\frac{3}{2} & -\frac{1}{2} \\ 0 & 0 & -1 & -1 & 0 \end{bmatrix}$$

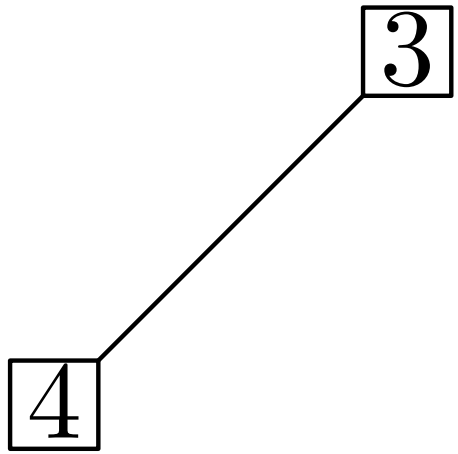


Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & 0 & -1 & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{3} \end{bmatrix}$$



Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 0 & 2 & -1 & -1 & -3 \\ 0 & 0 & -\frac{3}{2} & -\frac{3}{2} & \frac{1}{2} \\ 0 & 0 & 0 & -1 & -\frac{2}{3} \\ 0 & 0 & 0 & 0 & -\frac{1}{3} \end{bmatrix}$$

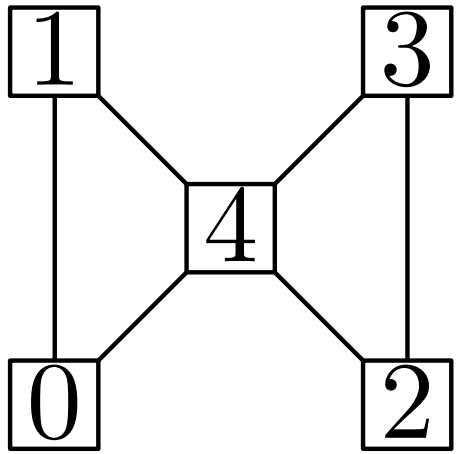
$\boxed{4}$

Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ -2 & 1 & 0 & 0 & 1 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 1 & -1 & 1 & 1 & 1 \end{bmatrix}$$

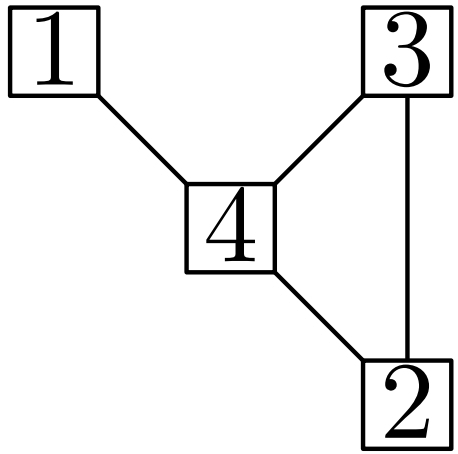


Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

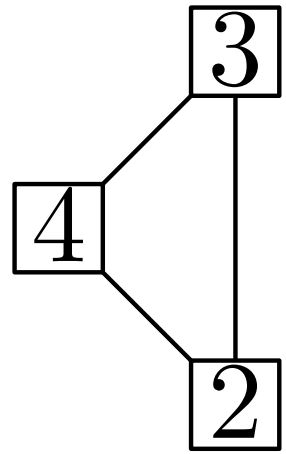


Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & -1 & -2 & -1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

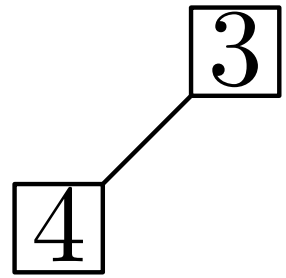


Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$



Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

$$\begin{bmatrix} 1 & -1 & 0 & 0 & 1 \\ 0 & -1 & 0 & 0 & 3 \\ 0 & 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -1 & -2 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

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.

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.

Literatur II

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.
accepted.

Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter.
Exact routing in large road networks using contraction hierarchies.
Transportation Science, 46(3):388–404, August 2012.