Karlsruher Institut für Technologie

# Algorithmen für Routenplanung

5. Vorlesung, Sommersemester 2015

Ben Strasser | 6. Mai 2014

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

# Hierarchische Techniken

**Ideensammlung:**

- identifiziere wichtige Knoten
- überspringe unwichtige Teile des Graphen



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Bilevel-Overlay Graph

**Gegeben**

- Eingabegraph $G = (V, E, \text{len})$
- Menge an Core Knoten $V_C \subseteq V$

**Berechne**

- Berechne $G_C = (V_C, E_C, \text{len}_C)$, so dass Distanzen in $G_C$ wie in $G$
- Suche zum Core von $s$ und $t$ aus und dann nur noch im Core

# Multilevel-Overlay Graphs

**Gegeben**

- Eingabegraph $G = (V, E, \text{len})$
- Folge $V := V_0 \supseteq V_1 \supseteq \ldots \supseteq V_L$ von Teilmengen von $V$.

**Berechne**

- Folge $G_0 = (V_0, E_0, \text{len}_0), \ldots, G_L = (V_L, E_L, \text{len}_L)$ von Graphen, so dass Distanzen in $G_i$ wie in $G_0$
- Suche von $s$ und $t$ erst zum Level 1 Core, dann zum Level 2 Core ... bis man beim Level $L$ Core ist

# Wahl der Knotenmenge

## Wichtige Knoten

- Brücken
- Autbahnen
- Kreuzungen mit vielen Straßen

## Unwichtige Knoten

- Sackgaßen
- Feldwege
- Nebenstraßen

# Contraction Hierarchies [GSSV12]

**Idee:**

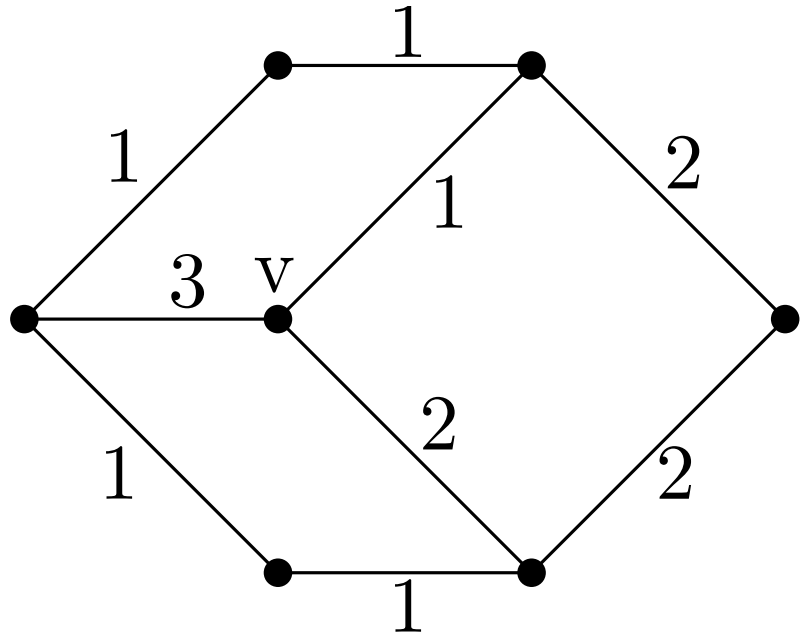
- Pro Knoten ein Level
- Von $G_i$ kommt man zu $G_{i+1}$ durch kontrahieren von einem Knoten $v$

# Contraction Hierarchies [GSSV12]

**Idee:**

- Pro Knoten ein Level
- Von $G_i$ kommt man zu $G_{i+1}$ durch kontrahieren von einem Knoten $v$

**Gewichtete Knotenkontraktion von $v$:**



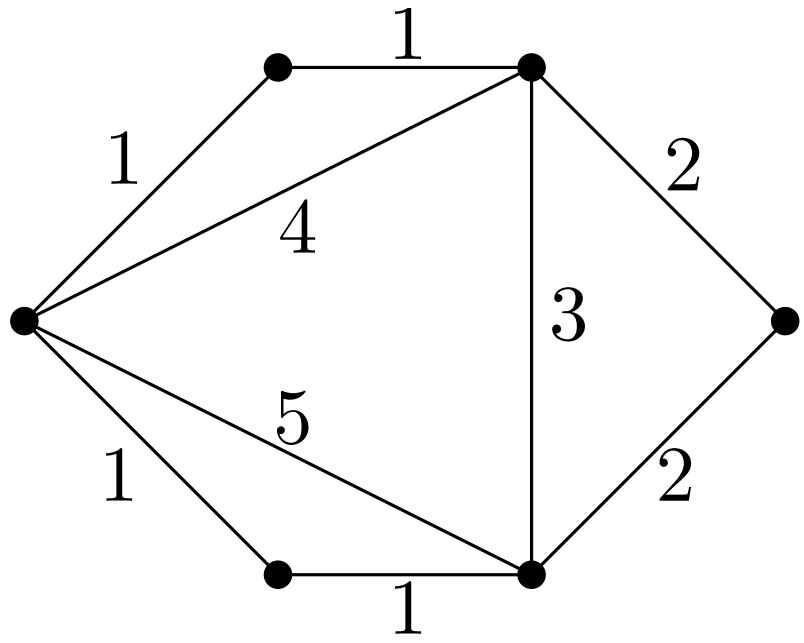
# Contraction Hierarchies [GSSV12]

**Idee:**

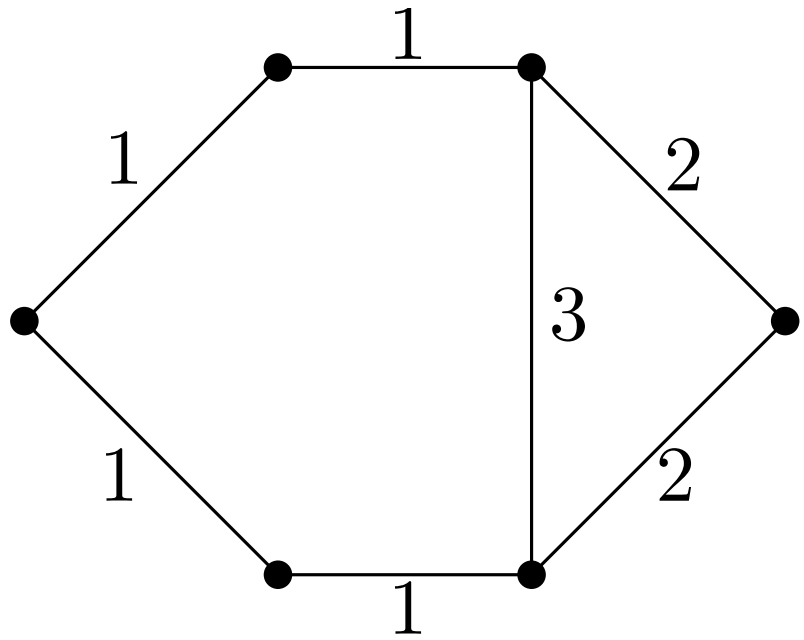
- Pro Knoten ein Level
- Von $G_i$ kommt man zu $G_{i+1}$ durch kontrahieren von einem Knoten $v$

**Gewichtete Knotenkontraktion von $v$:**

- entferne $v$



# Contraction Hierarchies [GSSV12]



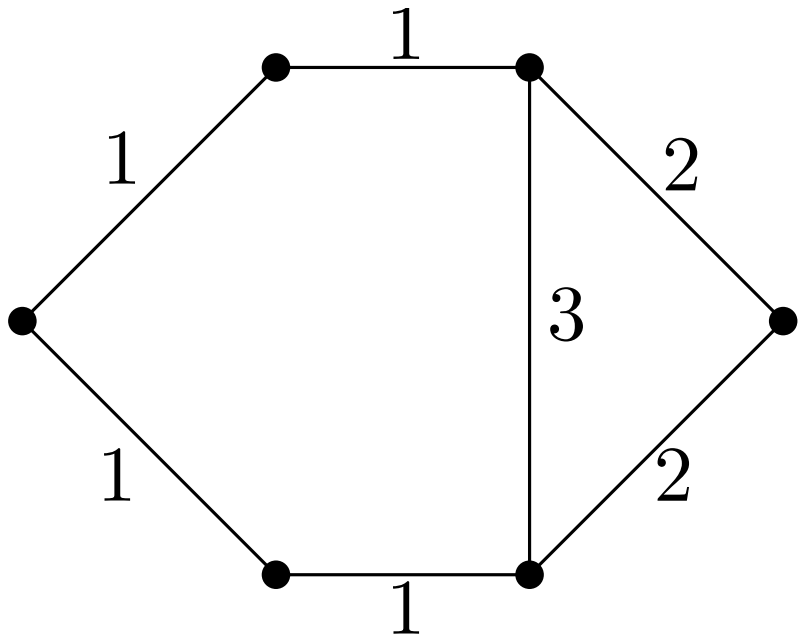

**Idee:**

- Pro Knoten ein Level
- Von $G_i$ kommt man zu $G_{i+1}$ durch kontrahieren von einem Knoten $v$

**Gewichtete Knotenkontraktion von $v$:**

- entferne $v$
- für jedes Paar $u, w$ füge Kanten $(u, w)$ zum Graphen wenn $v$ auf dem einzigen kürzesten Weg von $u$ nach $w$ liegt
- Neue Kanten heißen Shortcuts

# Contraction Hierarchies [GSSV12]




**Idee:**

- Pro Knoten ein Level
- Von $G_i$ kommt man zu $G_{i+1}$ durch kontrahieren von einem Knoten $v$

**Gewichtete Knotenkontraktion von $v$:**

- entferne $v$
- für jedes Paar $u, w$ füge Kanten $(u, w)$ zum Graphen wenn $v$ auf dem einzigen kürzesten Weg von $u$ nach $w$ liegt
- Neue Kanten heißen Shortcuts
- kann durch lokale Dijkstra Suchen von den Nachbarn berechnet werden, genannt Zeugensuche

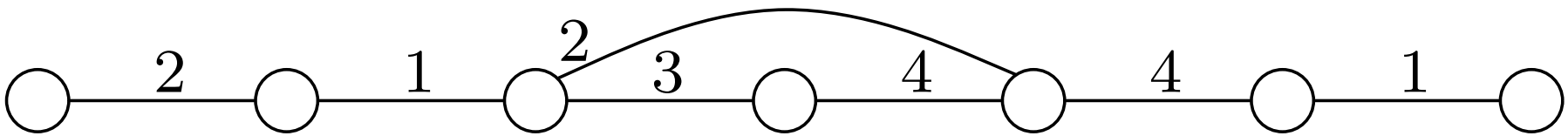
# Contraction Hierarchies

## Zeugensuche

- Zeugensuche kann aufwendig sein
- Lösung: Breche die Zeugensuche nach frühzeitig ab z.B. nach $k$ gesettelten Knoten
- Führt zu zu vielen Kanten
- Bleibt aber korrekt

# Contraction Hierarchies

## Vorberechnung:



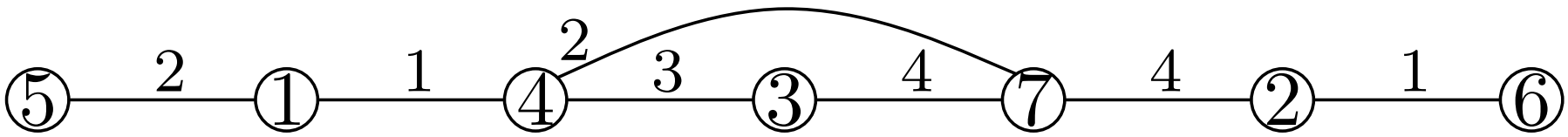



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Contraction Hierarchies

## Vorberechnung:

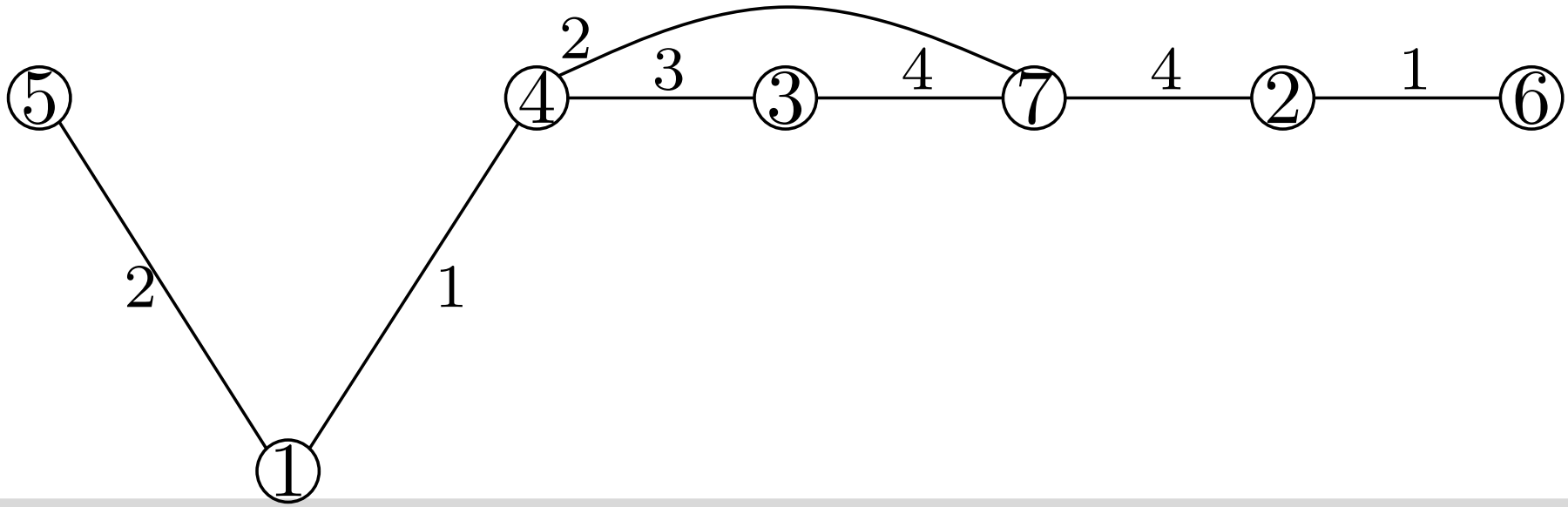
- ordne Knoten



# Contraction Hierarchies

**Vorberechnung:**

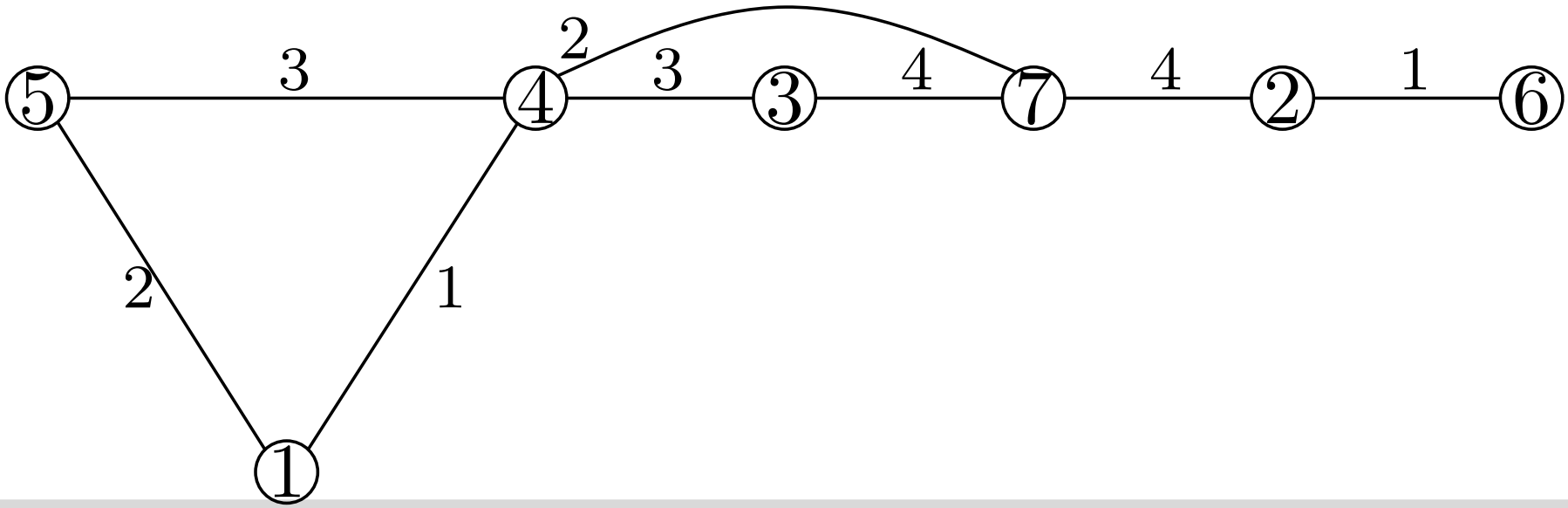
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)



# Contraction Hierarchies

**Vorberechnung:**

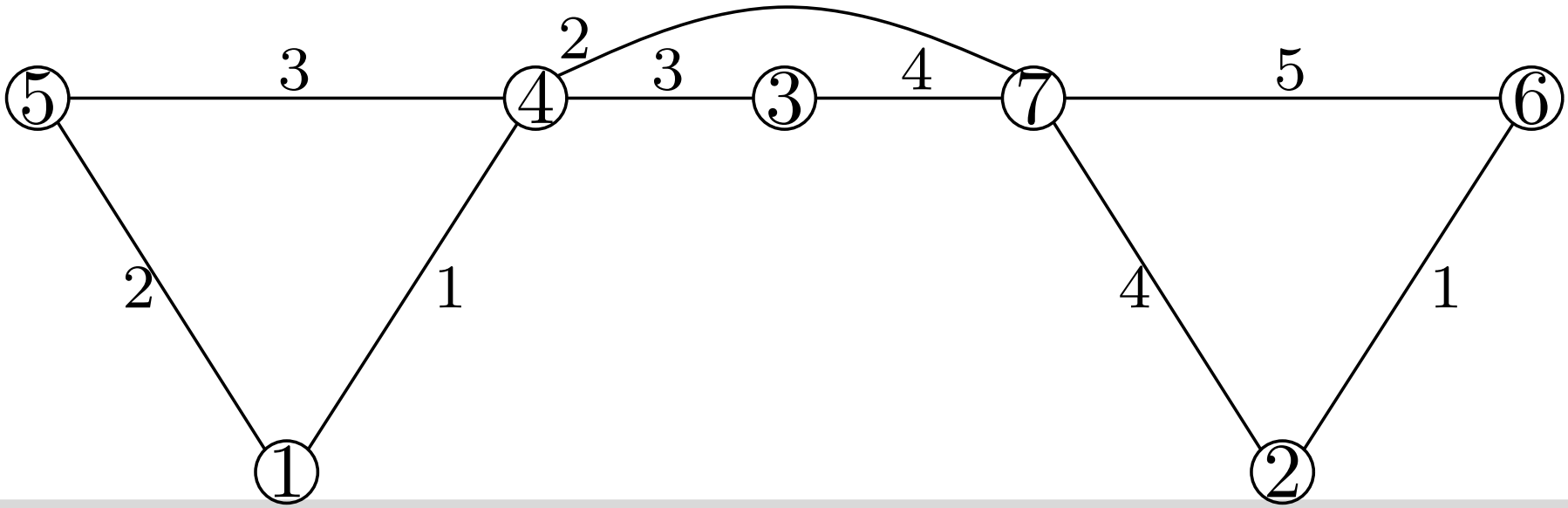
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)



# Contraction Hierarchies

**Vorberechnung:**

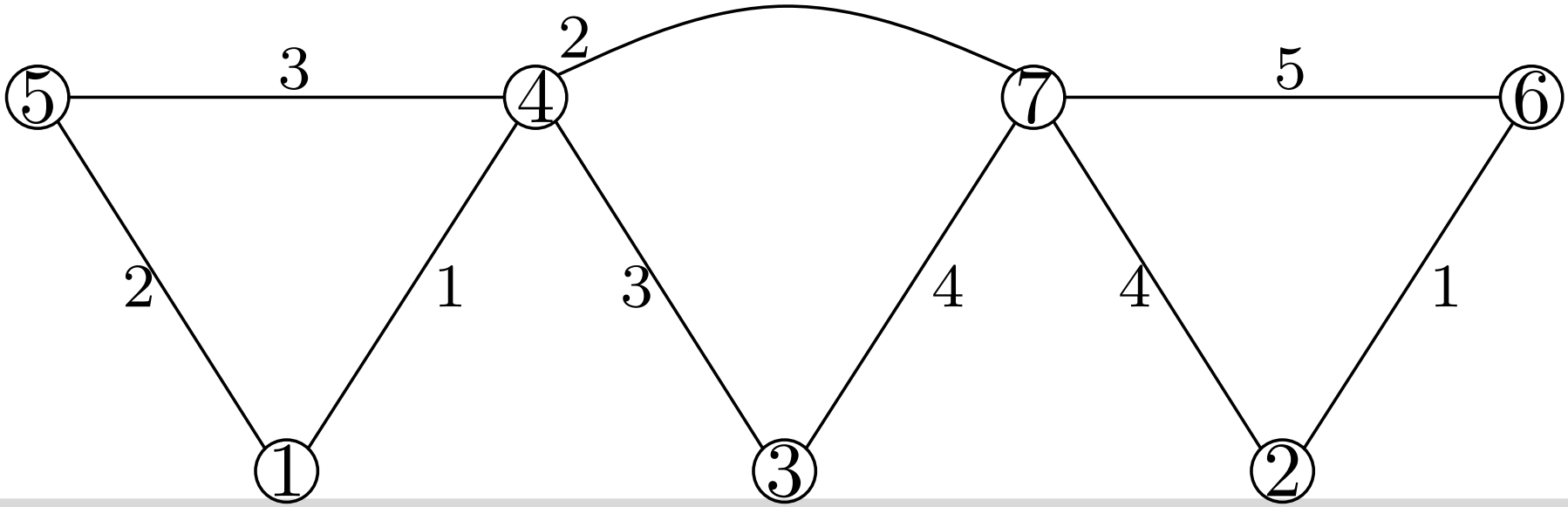
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)



# Contraction Hierarchies

**Vorberechnung:**

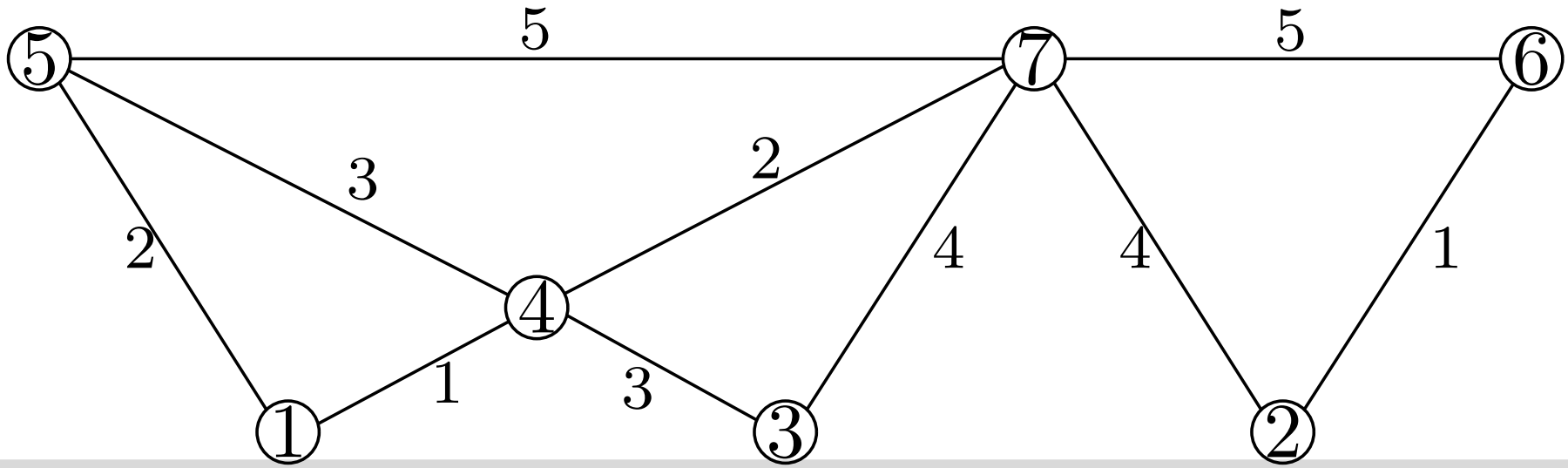
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)



# Contraction Hierarchies

**Vorberechnung:**

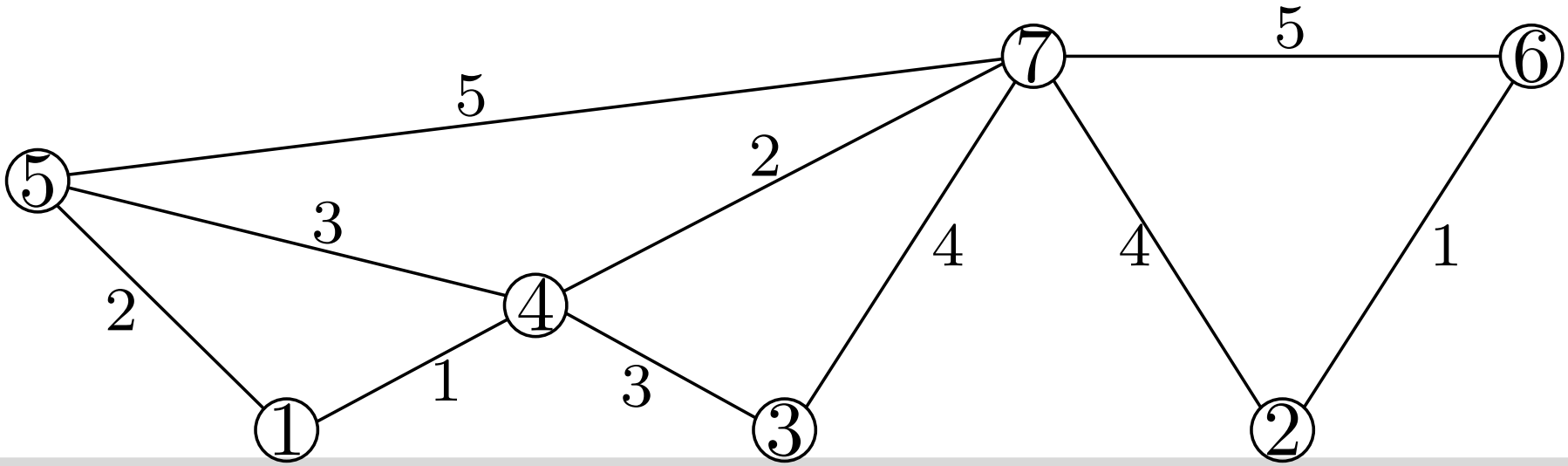
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)



# Contraction Hierarchies

**Vorberechnung:**

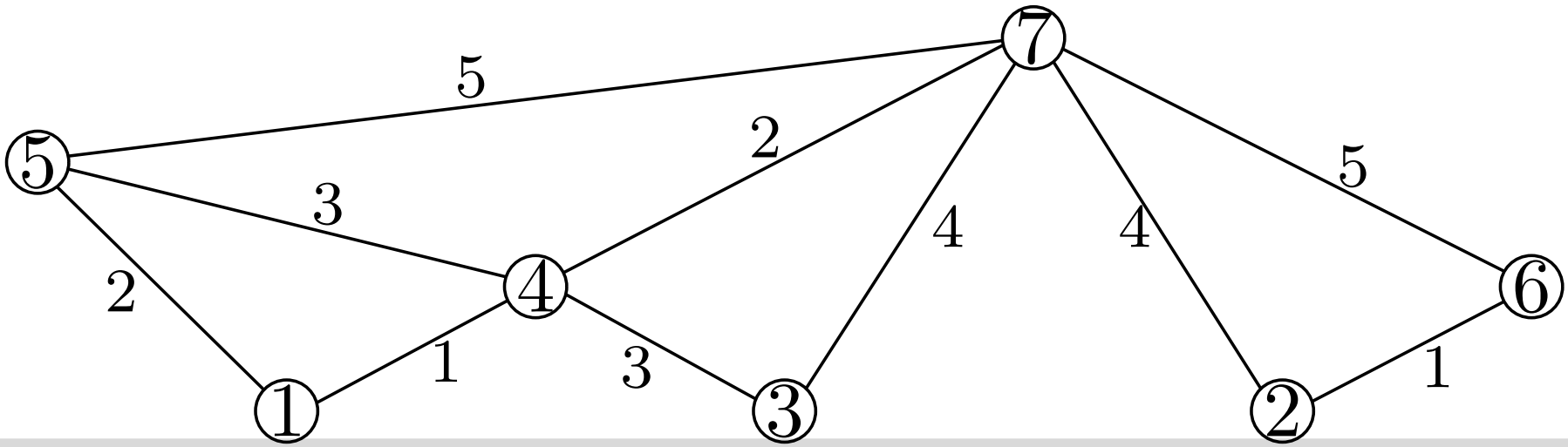
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)



# Contraction Hierarchies

**Vorberechnung:**

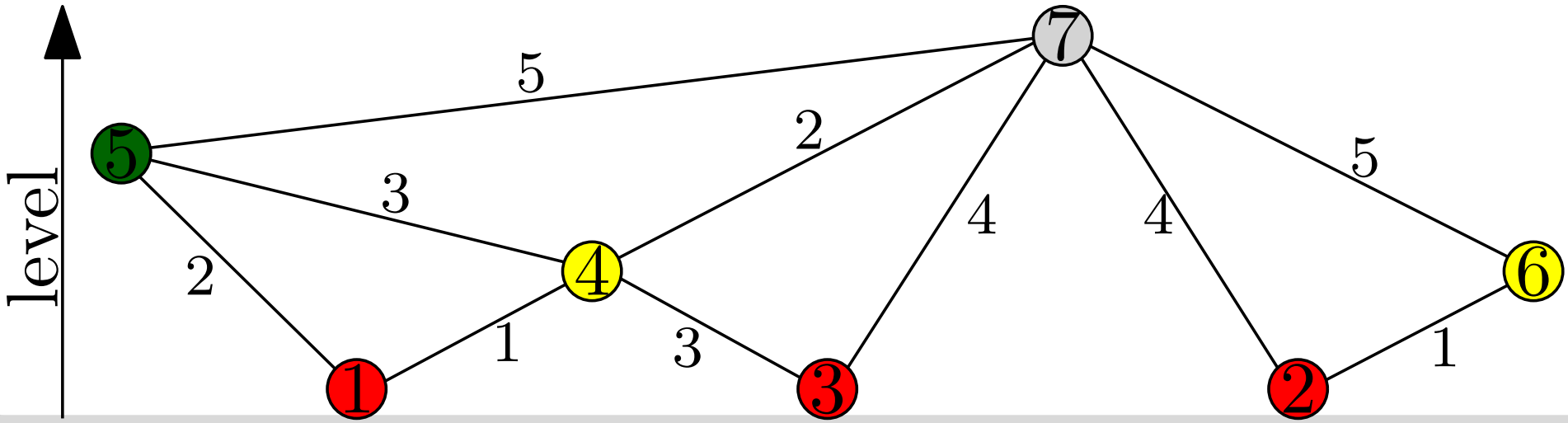
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)



# Contraction Hierarchies

**Vorberechnung:**

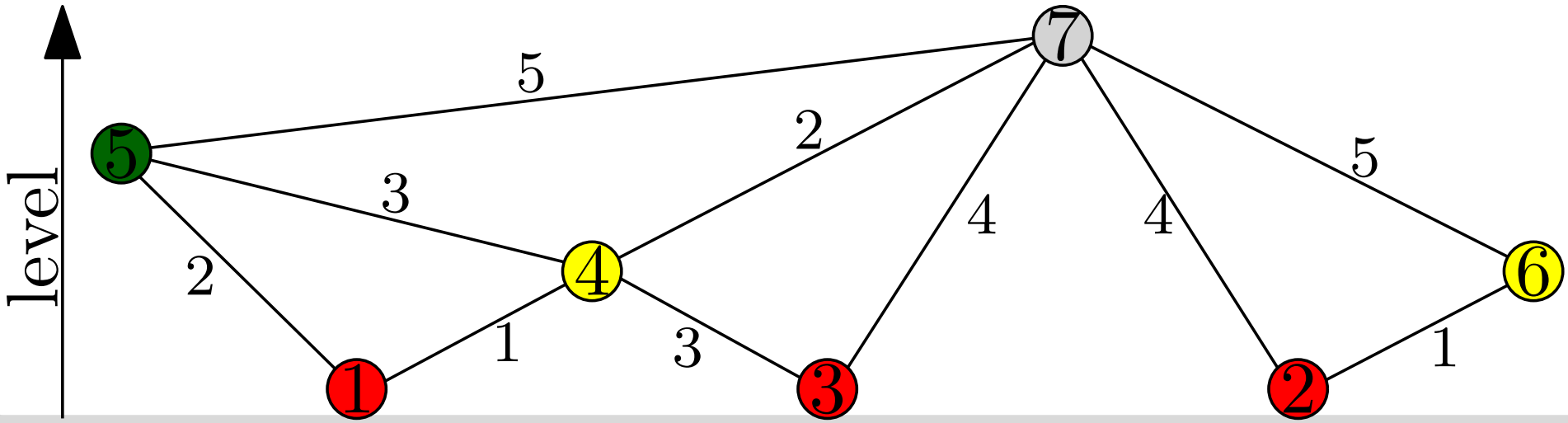
- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)
- bestimme Level des Knoten



# Contraction Hierarchies

**Vorberechnung:**

- ordne Knoten
- bearbeite in der Reihenfolge
- kontrahieren des Knoten, hinzufügen von Shortcuts ($A^+$)
- bestimme Level des Knoten
- erzeugt einen Graph $G^+ = (V, A \cup A^+)$



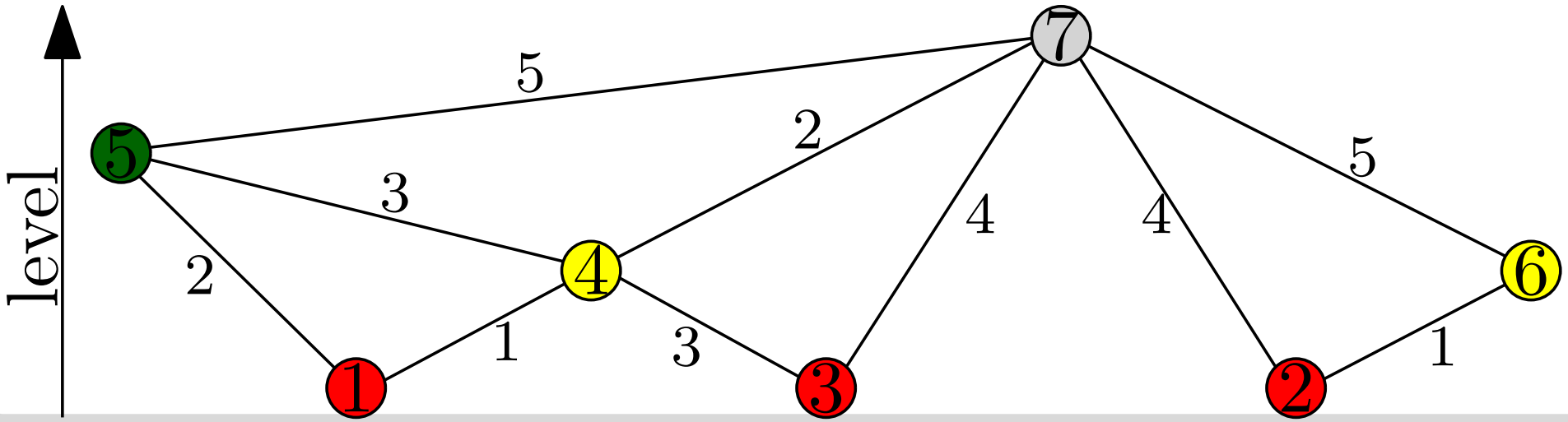
# Contraction Hierarchies

## Anfrage

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten

  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$

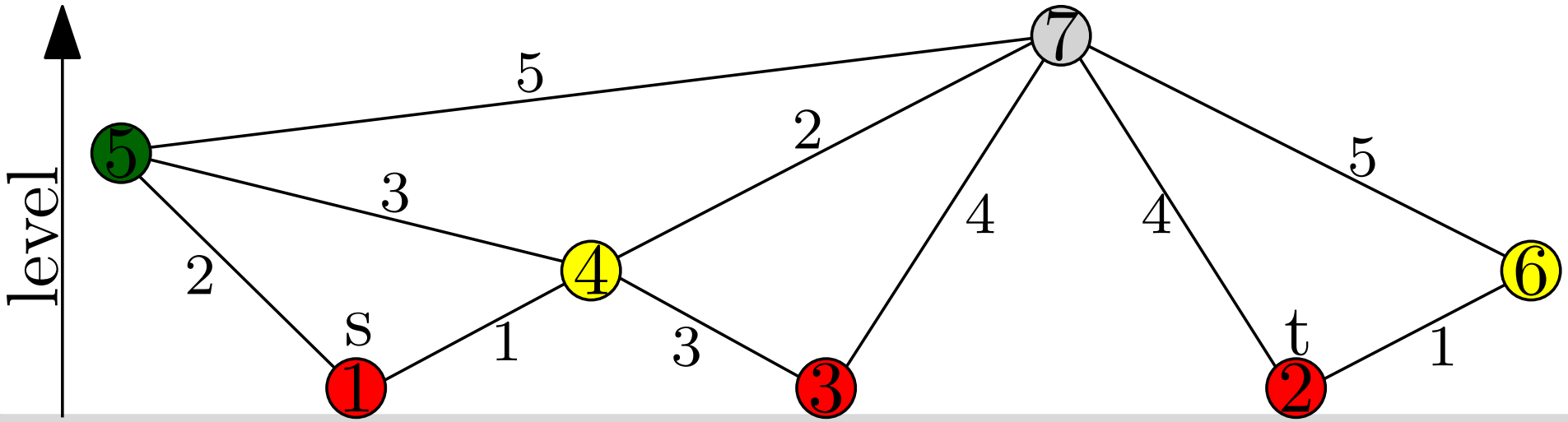
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium:
  - jede Suche stoppt wenn Queue leer oder $minKey(Q) > \mu$



# Contraction Hierarchies

## Anfrage

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium:
  - jede Suche stoppt wenn Queue leer oder $minKey(Q) > \mu$



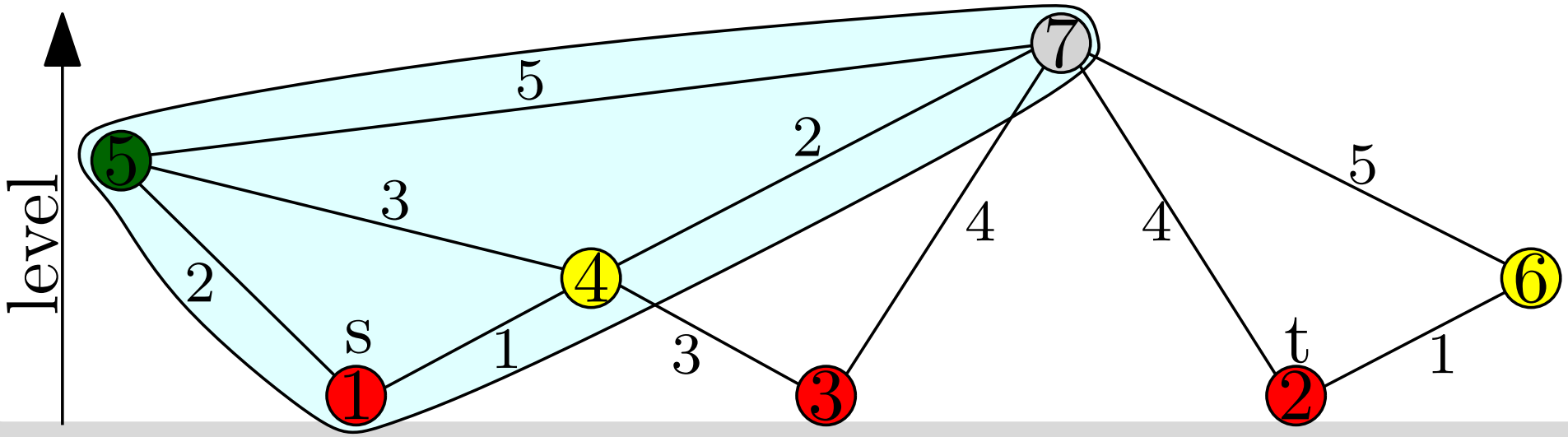
# Contraction Hierarchies

## Anfrage

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten

  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$

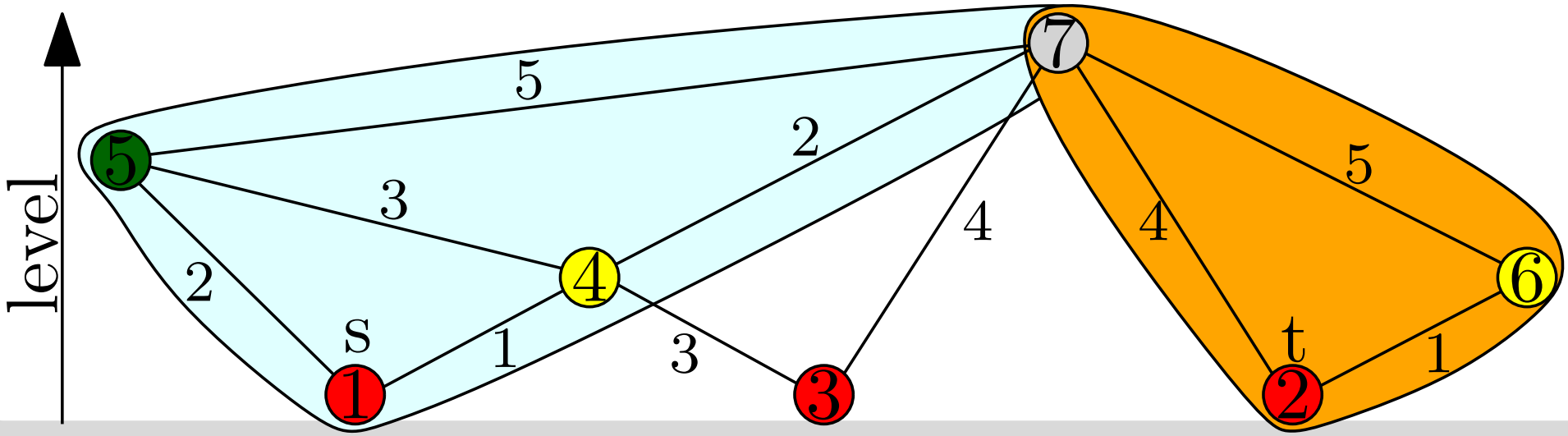
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium:
  - jede Suche stoppt wenn Queue leer oder $\mathit{minKey}(Q) > \mu$



# Contraction Hierarchies

## Anfrage

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten
  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$
  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium:
  - jede Suche stoppt wenn Queue leer oder $minKey(Q) > \mu$



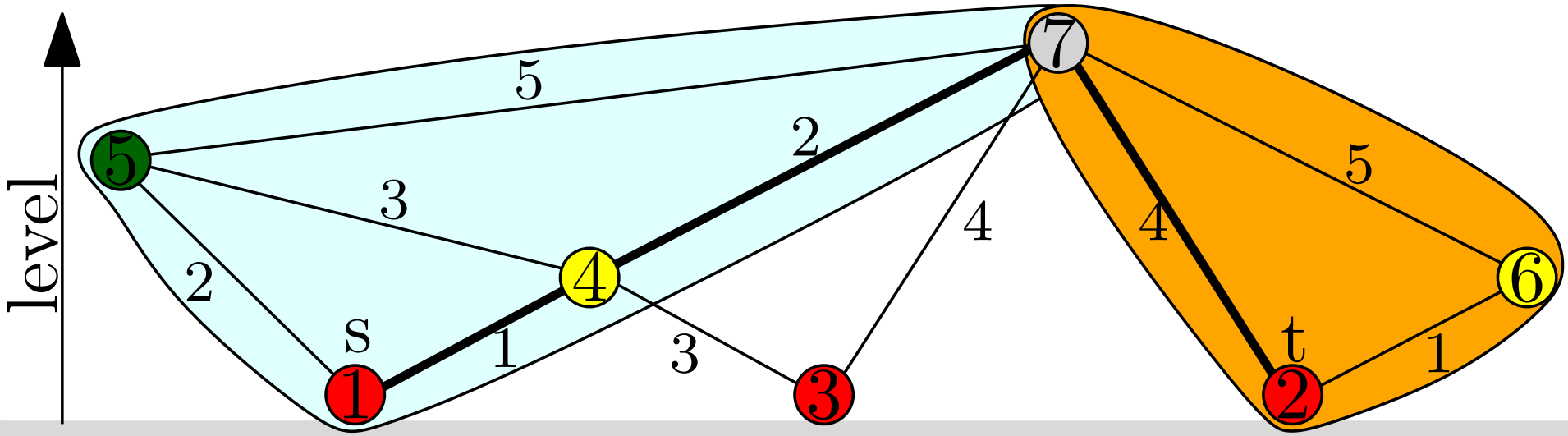
# Contraction Hierarchies

## Anfrage

- modifizierter bidirektionaler Dijkstra
- folge nur Kanten zu wichtigeren Knoten

  Aufwärtsgraph $G_\uparrow := (V, E_\uparrow)$ with $E_\uparrow := \{(u, v) \in E : u < v\}$

  Abwärtsgraph $G_\downarrow := (V, E_\downarrow)$ with $E_\downarrow := \{(u, v) \in E : u > v\}$
- Vorwärtssuche in $G_\uparrow$ und Rückwärtssuche in $G_\downarrow$
- konservatives Stoppkriterium:
  - jede Suche stoppt wenn Queue leer oder $\mathit{minKey}(Q) > \mu$

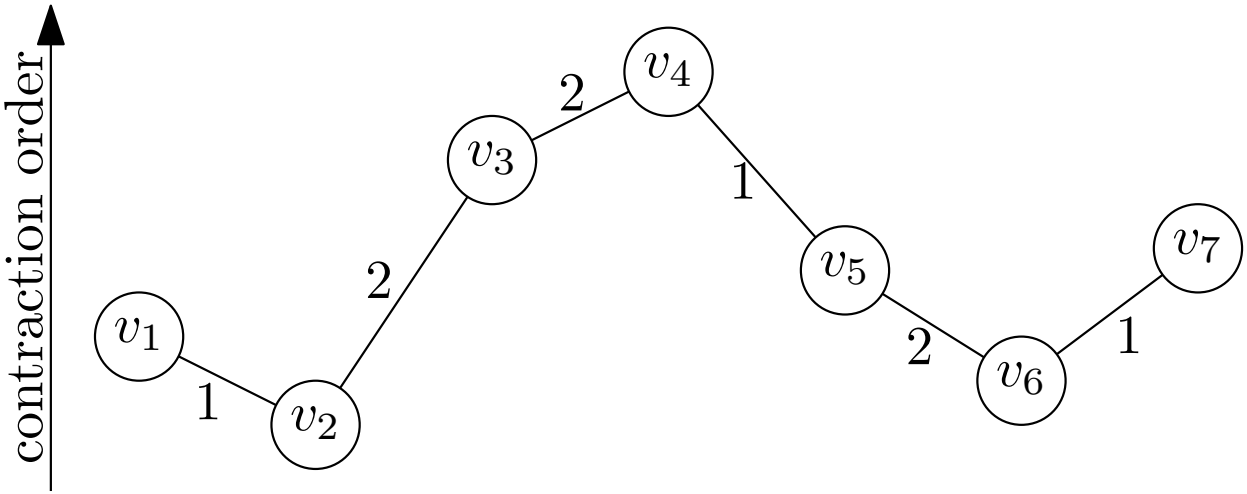


# Korrektheit

- 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

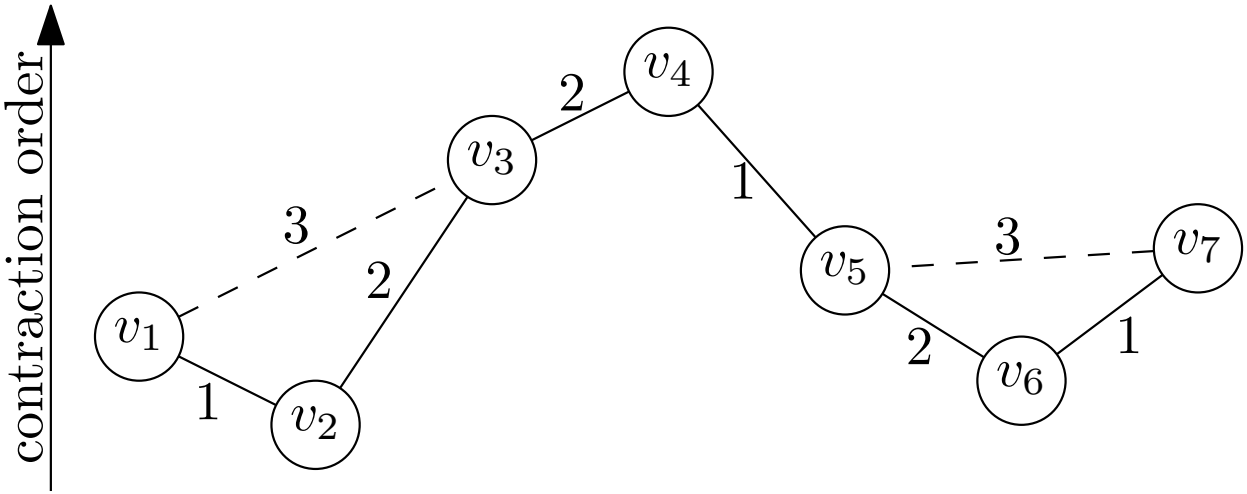
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in $G^+$ immer einen kürzesten hoch-runter Pfad gibt.




- 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

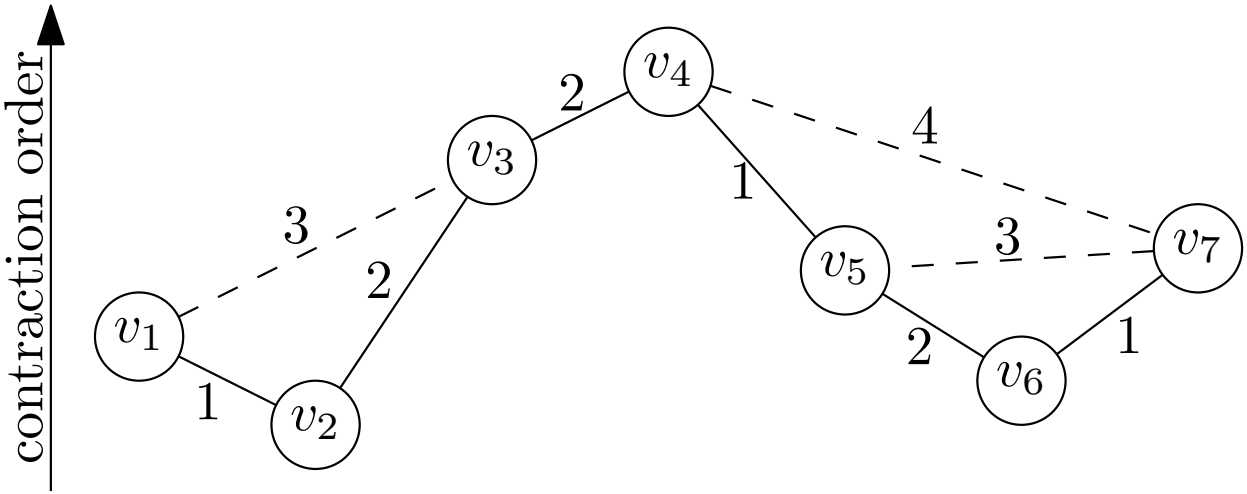
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in $G^+$ immer einen kürzesten hoch-runter Pfad gibt.




- 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

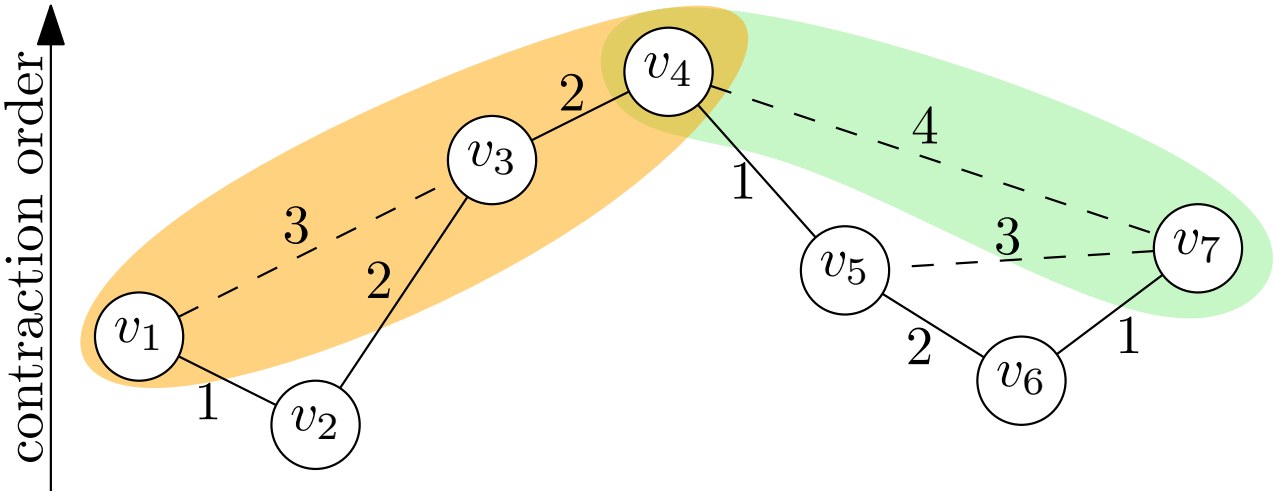
- Die Vorwärtssuche findet nur Aufwärtspfade.
- Die Rückwärtssuche findet nur Abwärtspfade.
- Gemeinsam werden nur Pfade gefunden die hoch und dann wieder runter gehen.
- Wir müssen also beweisen, dass es in $G^+$ immer einen kürzesten hoch-runter Pfad gibt.




- 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.

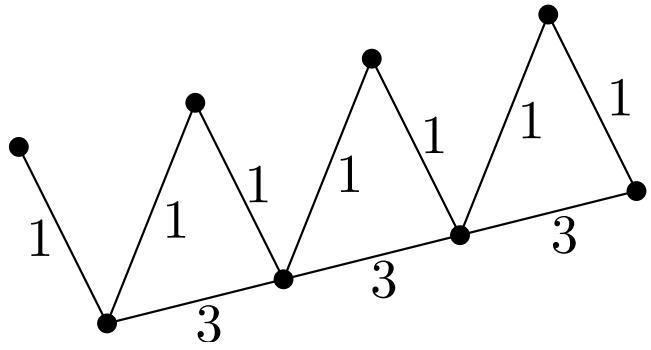



- Es gibt einen kürzesten Weg $P$
- Wenn $P$ kein hoch-runter Pfad ist, dann gibt es einen Knoten der höhere Nachbarn hat
- Dann gibt es auch einen hoch-runter Pfad

# Stall-On-Demand

## Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen



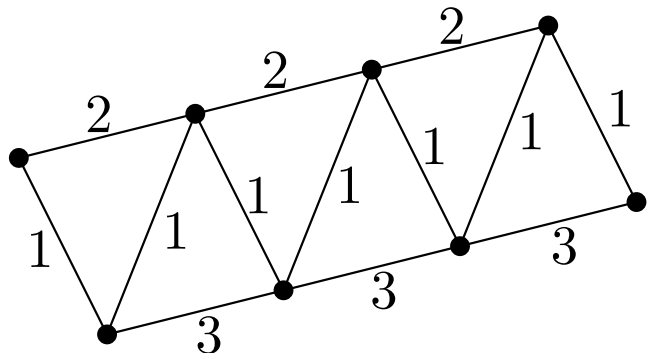



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Stall-On-Demand

## Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen



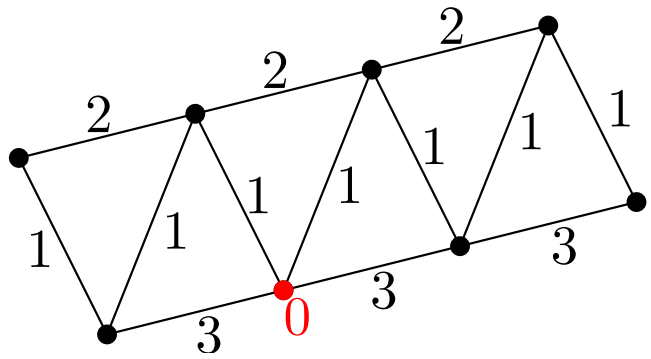



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Stall-On-Demand

## Beobachtung:

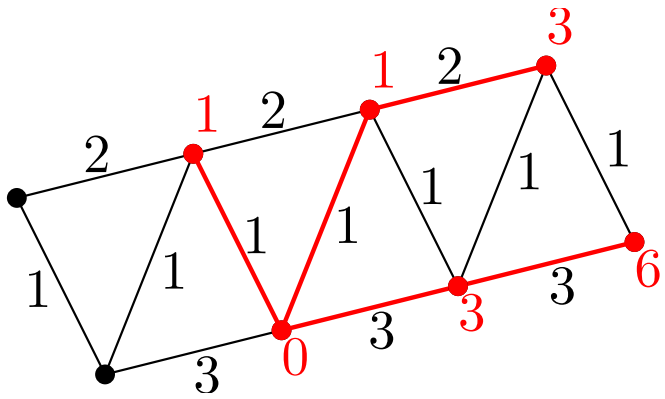
- Suchen können Knoten mit zu großer Distanz besuchen



# Stall-On-Demand

## Beobachtung:

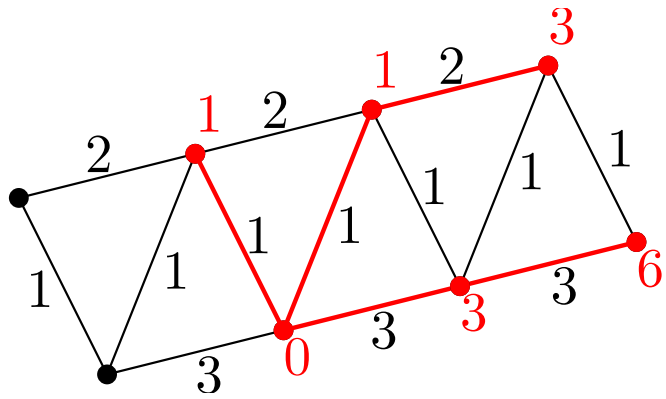
- Suchen können Knoten mit zu großer Distanz besuchen



# Stall-On-Demand

## Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

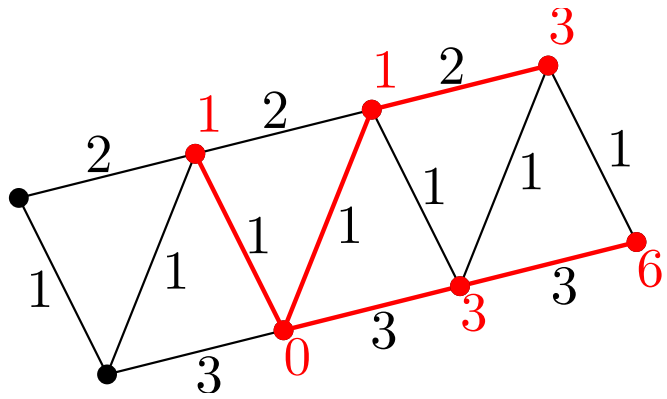



- Kann man zum prunen verwenden

# Stall-On-Demand

## Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

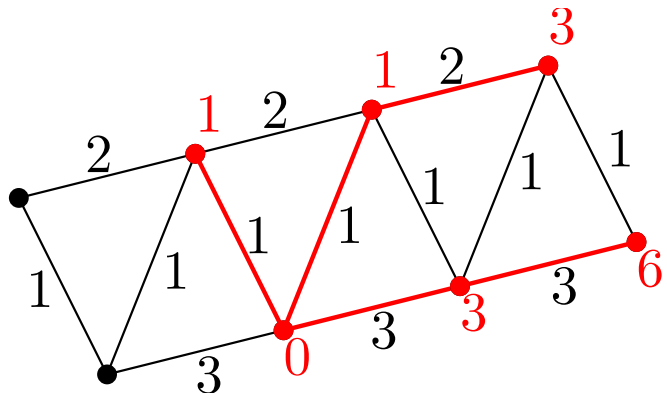



- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settled sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen

# Stall-On-Demand

## Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

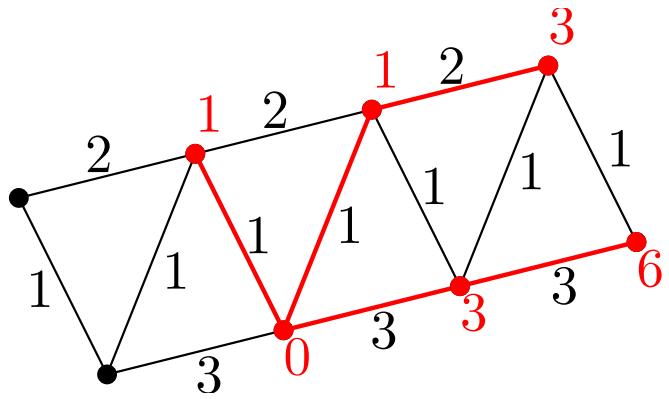



- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settled sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen
- Knoten $v$ wird geprunt, wenn es Aufwärtsnachbar $u$ von $v$ gibt, so dass $d(u) +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

# Stall-On-Demand

## Beobachtung:

- Suchen können Knoten mit zu großer Distanz besuchen

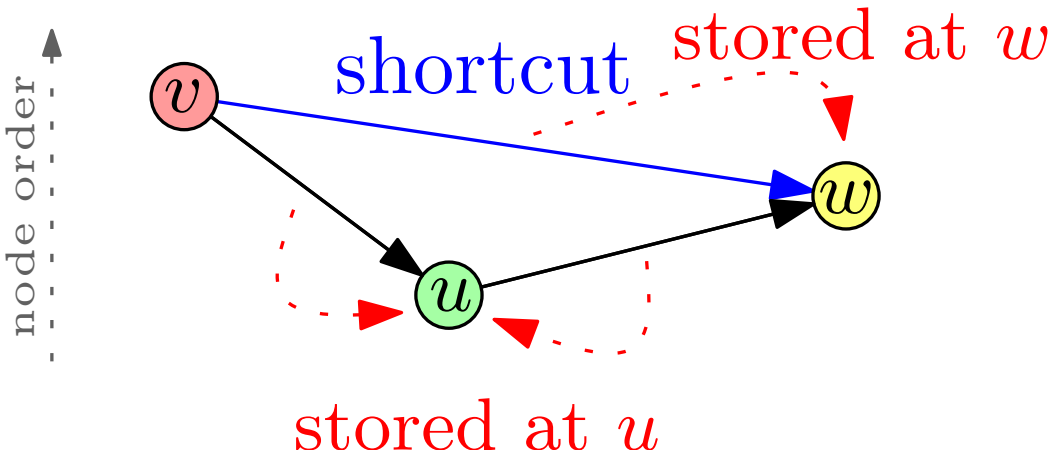



- Kann man zum prunen verwenden
- Jeder Teilpfad eines kürzesten hoch-runter-Pfads muss ein kürzester Pfad sein
- Ehe man einen Knoten settled sucht man nach kürzeren hoch-runter-Pfaden, gibt es diese kann man prunen
- Knoten $v$ wird geprunt, wenn es Aufwärtsnachbar $u$ von $v$ gibt, so dass $d(u) +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

# Datenstruktur

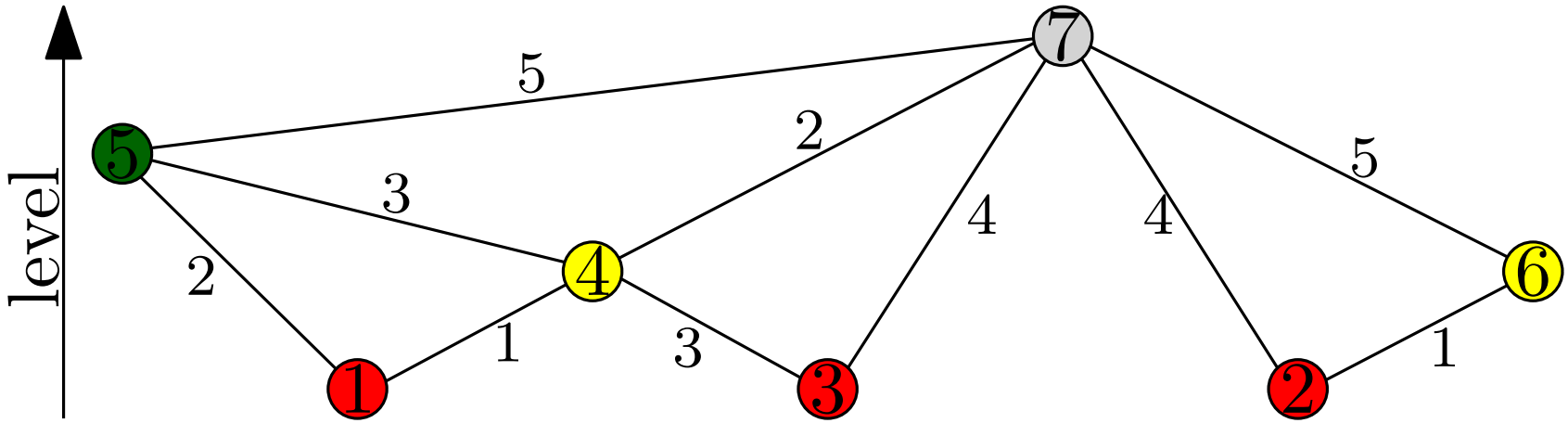
**Suchgraph:**

- 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



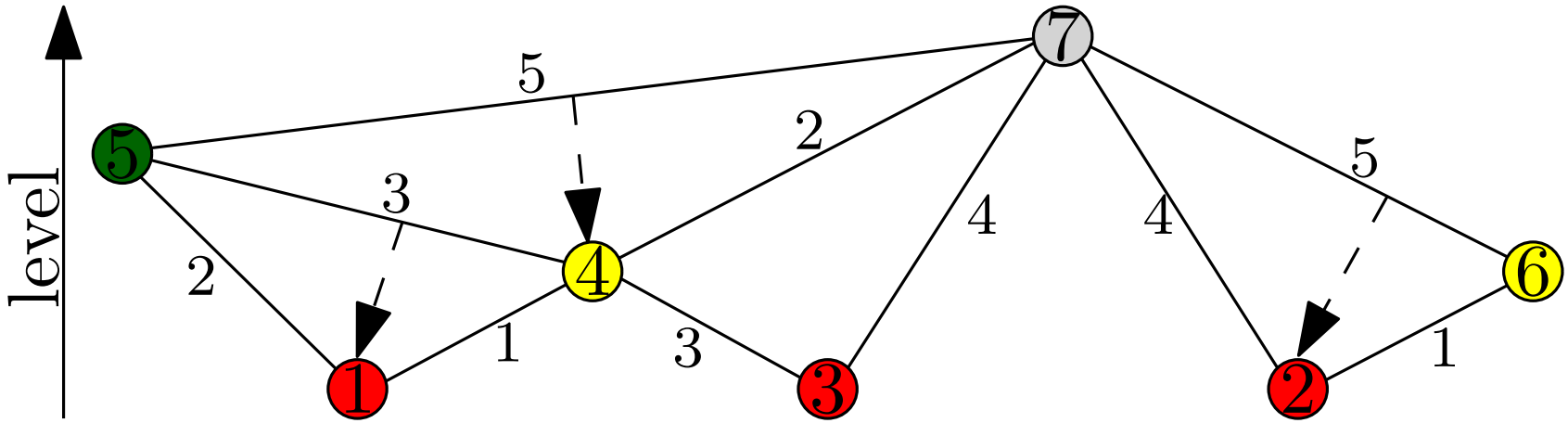
# Pfadentpackung

- 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]
- top-down, suche nach wichtigen Knoten [ADGW12]

# Bottom-Up

## Vorgehen:



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Bottom-Up

## Vorgehen:

- identifiziere aktuell unwichtigsten Knoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Bottom-Up

**Vorgehen:**

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Bottom-Up

**Vorgehen:**

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten
- wiederhole



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Bottom-Up

**Vorgehen:**

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten
- wiederhole

**Kriterien zum Bestimmen dieses Knotens:**

- Differenz Hinzugefügter und Gelöschter Kanten
- Differenz der Originalkanten, die diese Kanten repräsentieren
- Level des Knoten

# Bottom-Up

**Vorgehen:**

- identifiziere aktuell unwichtigsten Knoten
- kontrahiere den Knoten
- wiederhole

**Kriterien zum Bestimmen dieses Knotens:**

- Differenz Hinzugefügter und Gelöschter Kanten
- Differenz der Originalkanten, die diese Kanten repräsentieren
- Level des Knoten

**Diskussion**

- schnell, da alles bis auf die Zeugensuche lokale Informationen sind
- kann on-the-fly berechnet werden
- aber nur, solange der Knotengrad gering bleibt
- Qualität der Ordnung ist schlechter (größere Suchräume)

# 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
  (Details nicht in der Vorlesung)
- gute Qualität der Ordnung aber langsam

# Sampling Path-Greedy [DGPW14]

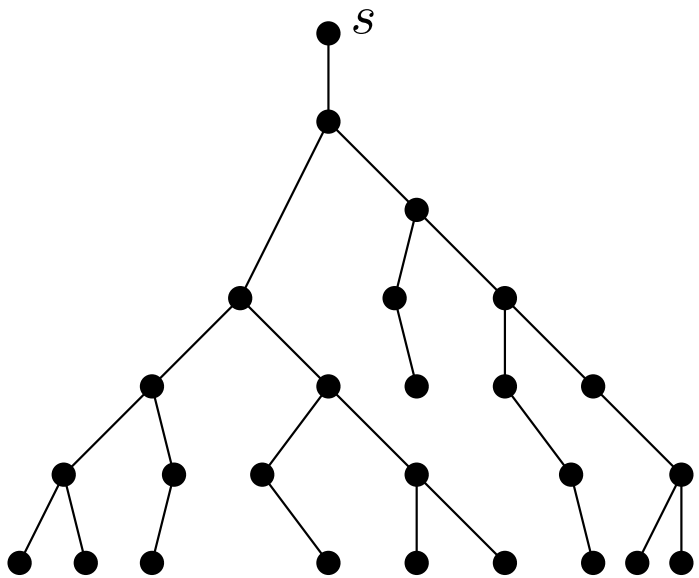
## Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
- Wähle $v$ das auf den meisten Pfaden in diesen Bäumen liegt

# Sampling Path-Greedy [DGPW14]

## Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
- Wähle $v$ das auf den meisten Pfaden in diesen Bäumen liegt

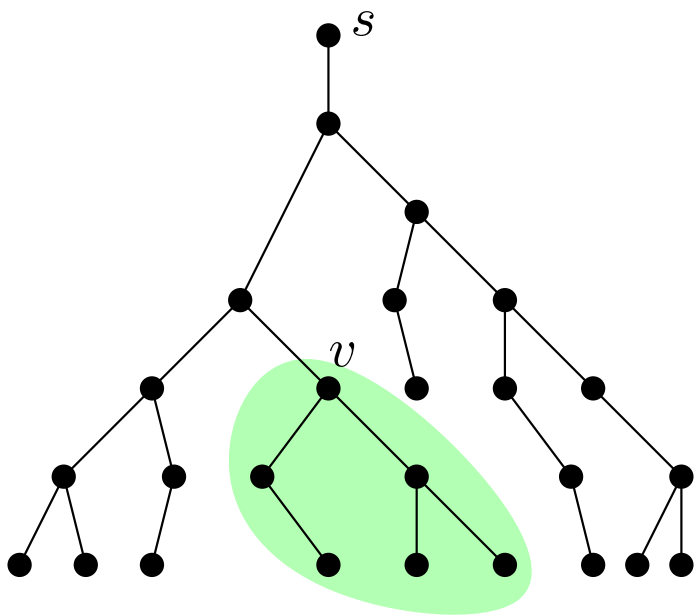



- Der kürzeste Wege Baum von $s$

# Sampling Path-Greedy [DGPW14]

## Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
- Wähle $v$ das auf den meisten Pfaden in diesen Bäumen liegt

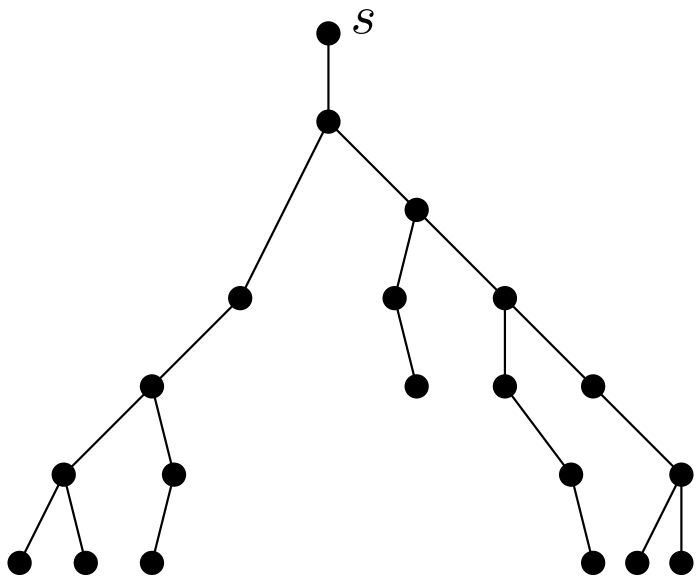



- Der kürzeste Wege Baum von $s$
- $v$ liegt auf so vielen Pfaden die bei $s$ beginnen wie sein Teilbaum groß ist (hier 6)

# Sampling Path-Greedy [DGPW14]

## Idee

- Verwalte nur wenige zufällige kürzeste Wege Bäume
- Wähle $v$ das auf den meisten Pfaden in diesen Bäumen liegt



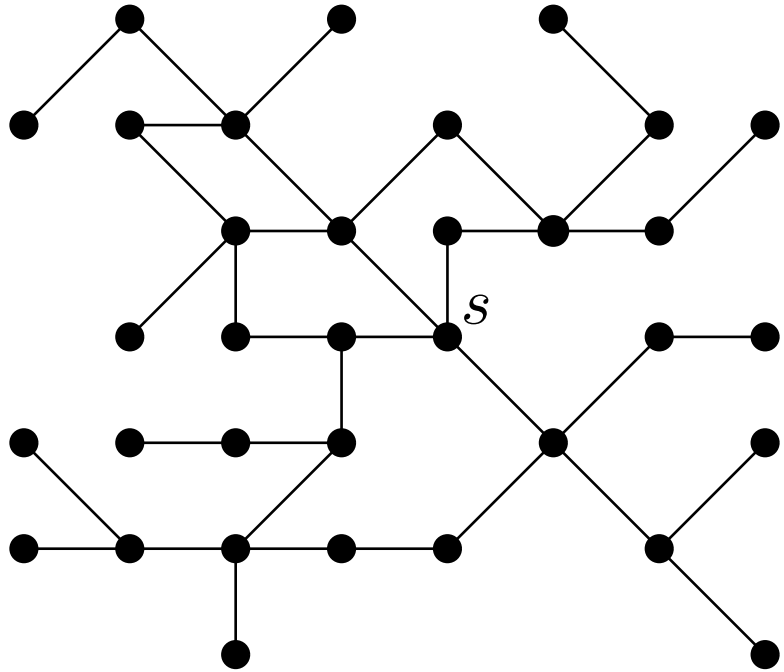

- Der kürzeste Wege Baum von $s$
- $v$ liegt auf so vielen Pfaden die bei $s$ beginnen wie sein Teilbaum groß ist (hier 6)
- Entferne Teilbaum

# Sampling Path-Greedy [DGPW14]

- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf

# Sampling Path-Greedy [DGPW14]

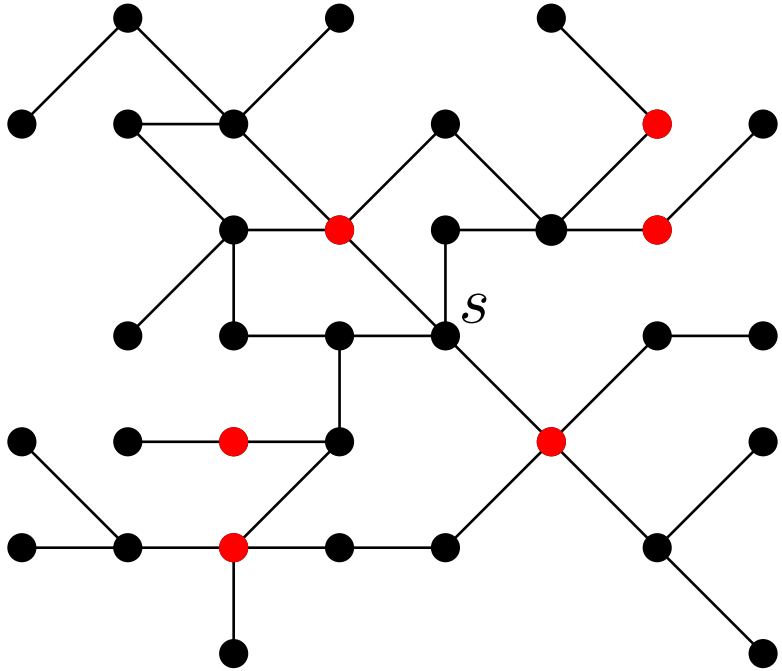
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf




*s* ist neue Baumwurzel

# Sampling Path-Greedy [DGPW14]

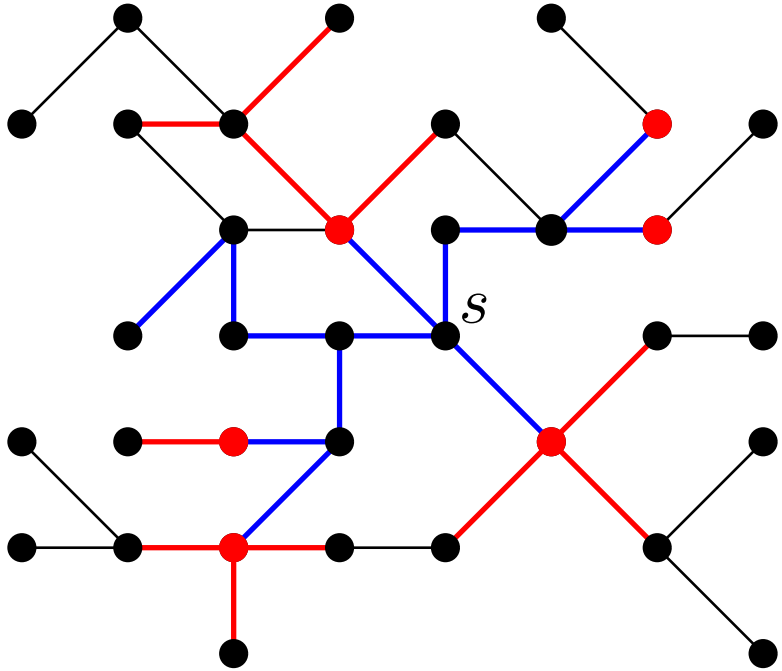
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf




rote Knoten sind bereits in der Ordnung

# Sampling Path-Greedy [DGPW14]

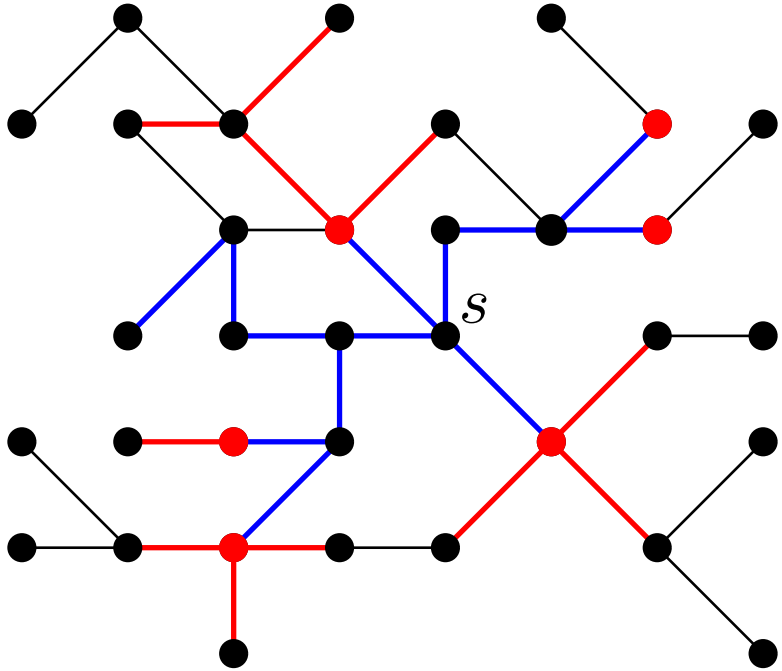
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf




lasse Dijkstra laufen

# Sampling Path-Greedy [DGPW14]

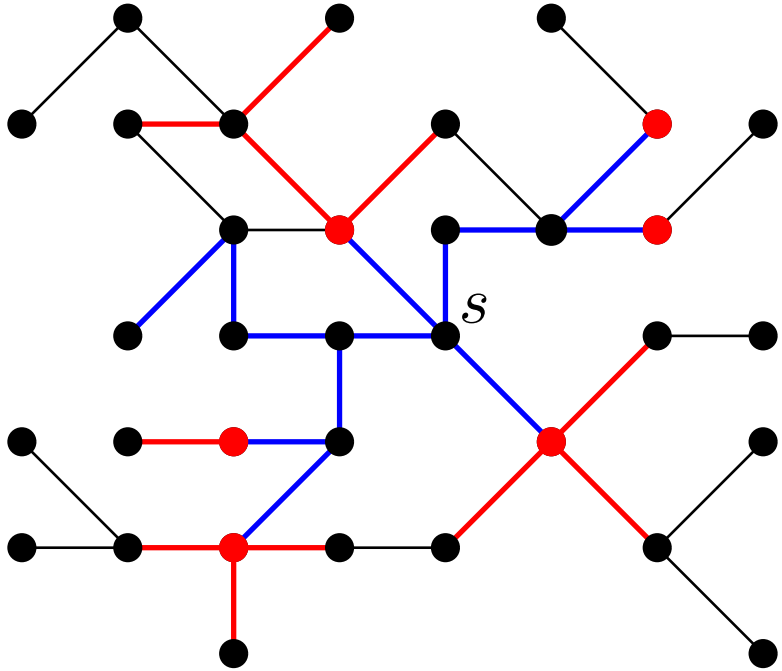
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf




speichere welche Knoten über rote Knoten erreicht wurden

# Sampling Path-Greedy [DGPW14]

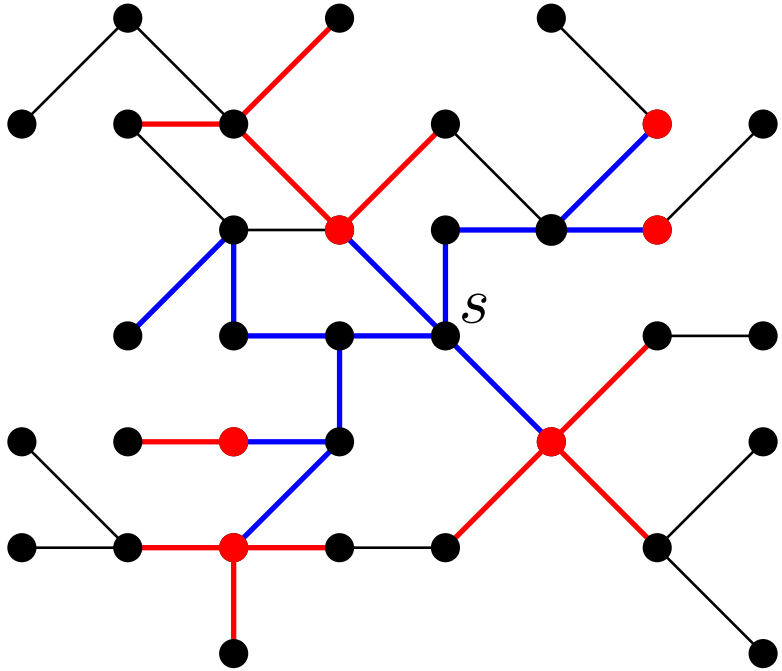
- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf




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

# Sampling Path-Greedy [DGPW14]

- Durch entfernen von Teilbäumen schrumpfen die Bäume
- $\rightarrow$ Baue neue Bäume auf




der blaue Teilgraph ist der neue Baum

# Sampling Path-Greedy [DGPW14]

Algo (vereinfacht):

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
- brauchbare Qualität und Laufzeit

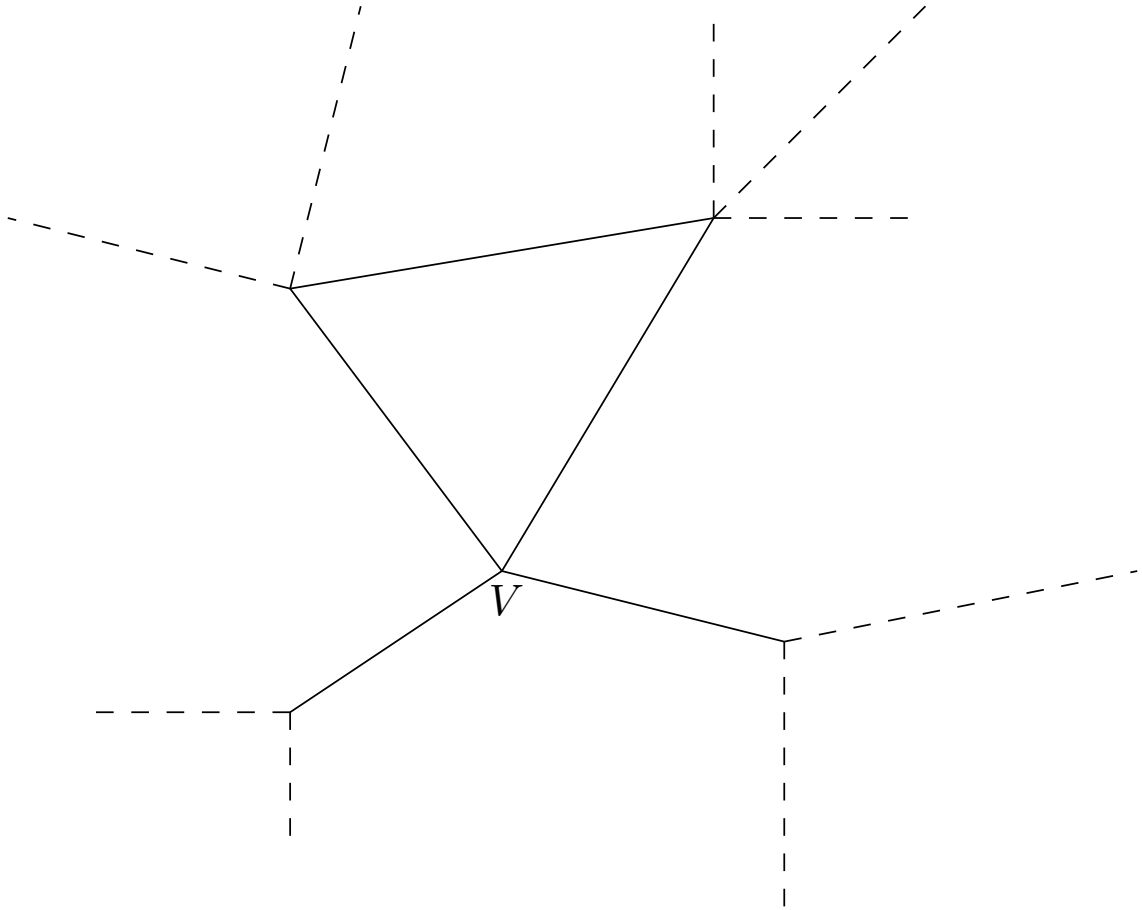
Auf Straße

- leicht besser als bottom-up
- aber langsamer als bottom-up

# Customizable Contraction Hierarchies [DSW14, BCRW13]

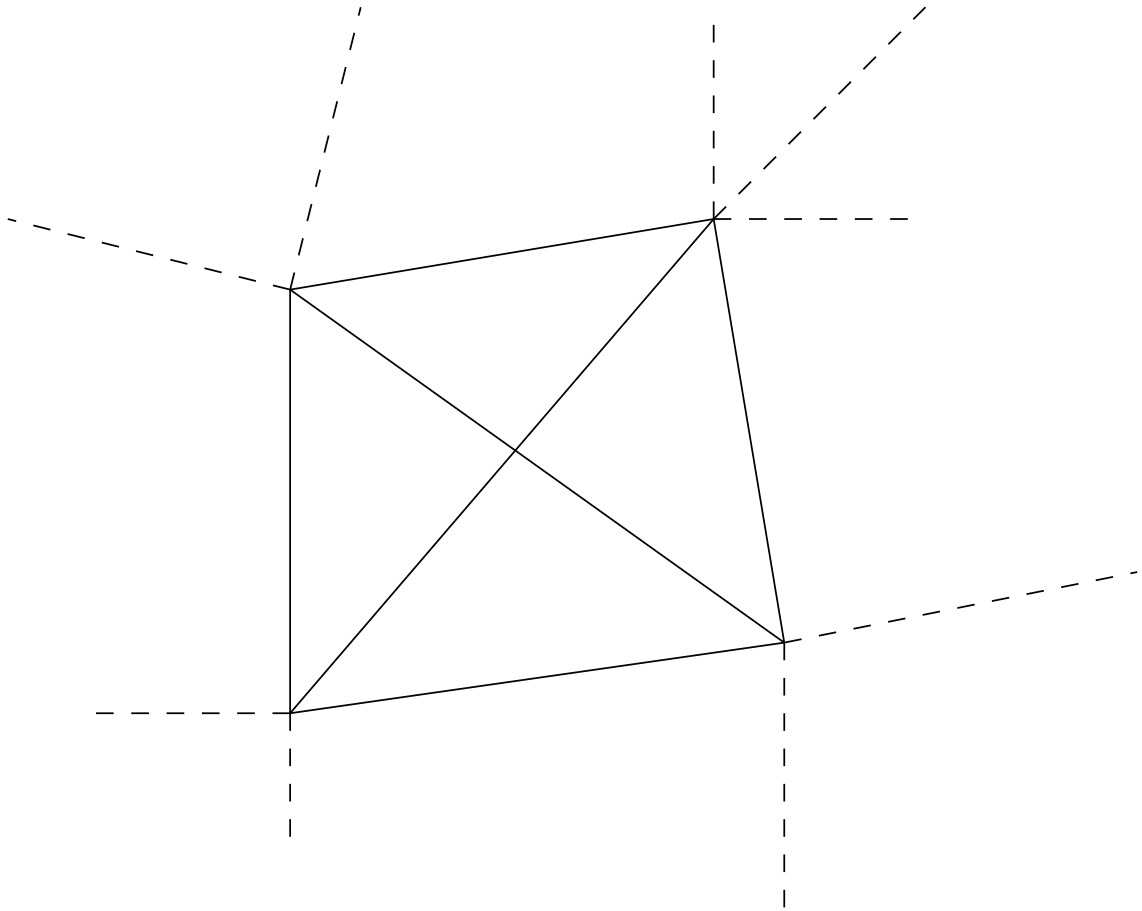
- 3-Phasen Ansatz wie bei CRP
- Preprocessing-, Customization- und Query-Phase
- Das jetzt mit CHs

# Ungewichtete Vertex-Contraction




Kontraktion von $v$: Ersetze $v$ mit vollständiger Clique.

# Ungewichtete Vertex-Contraction

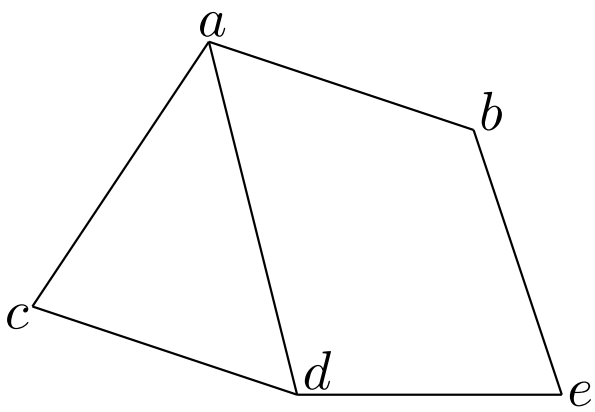


Kontraktion von $v$: Ersetze $v$ mit vollständiger Clique.



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

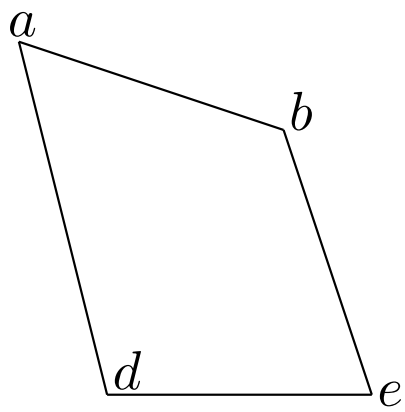
# Contraction-Order




Order $\pi$: $c$, $d$, $a$, $e$, $b$

Kontrahiere Knoten entlang Ordnung $\pi$.

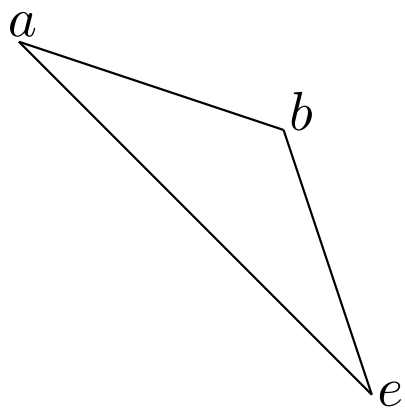
# Contraction-Order




Order $\pi$: $c$, $d$, $a$, $e$, $b$

Kontrahiere Knoten entlang Ordnung $\pi$.

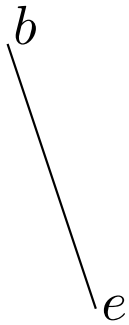
# Contraction-Order




Order $\pi$: $c$, $d$, $a$, $e$, $b$

Kontrahiere Knoten entlang Ordnung $\pi$.

# Contraction-Order




Order $\pi$: $c$, $d$, $a$, $e$, $b$

Kontrahiere Knoten entlang Ordnung $\pi$.

# Contraction-Order

$b$

Order $\pi$: $c$, $d$, $a$, $e$, $b$

Kontrahiere Knoten entlang Ordnung $\pi$.

# Contraction-Order

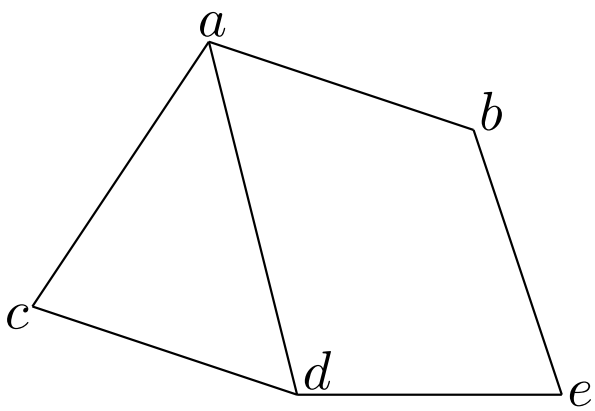
Order $\pi$: $c$, $d$, $a$, $e$, $b$

Kontrahiere Knoten entlang Ordnung $\pi$.

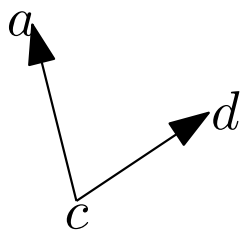


Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Contraction-Order

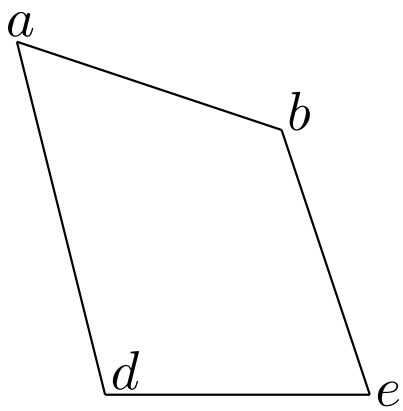


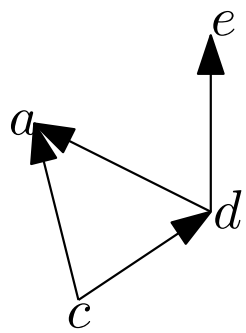

Order $\pi$: $c$, $d$, $a$, $e$, $b$




Baue Suchgraph auf

# Contraction-Order

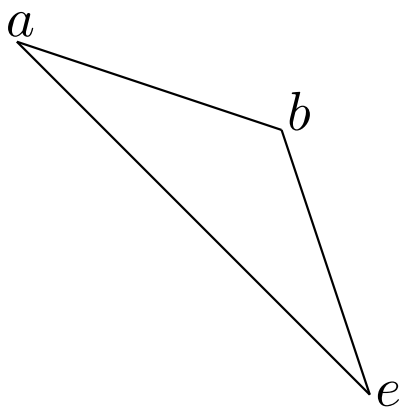


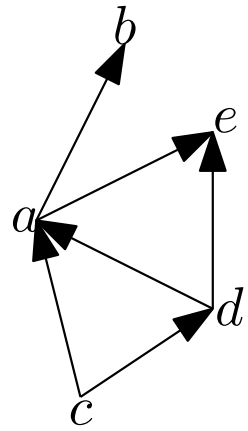




Order $\pi$: $c$, $d$, $a$, $e$, $b$

Baue Suchgraph auf

# Contraction-Order



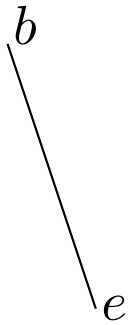

Order $\pi$: $c$, $d$, $a$, $e$, $b$

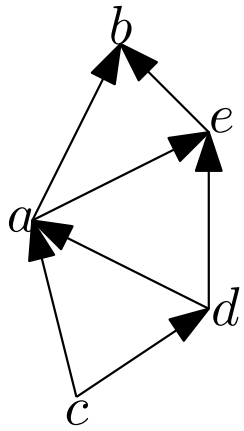
Baue Suchgraph auf



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Contraction-Order




Order $\pi$: $c$, $d$, $a$, $e$, $b$

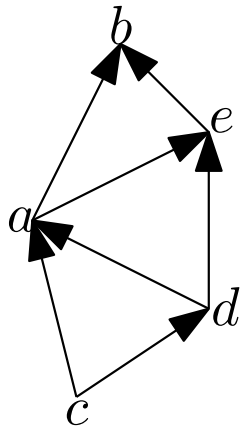
Baue Suchgraph auf



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Contraction-Order

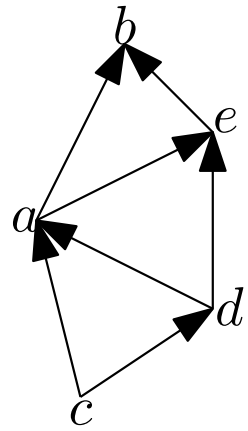
$b$




Order $\pi$: $c$, $d$, $a$, $e$, $b$

Baue Suchgraph auf

# Contraction-Order

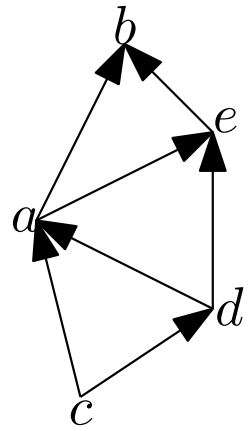



Order $\pi$: $c$, $d$, $a$, $e$, $b$

Baue Suchgraph auf

# Contraction-Order

Order $\pi$: $c$, $d$, $a$, $e$, $b$






Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# Definitionen

- Der Suchraum von $v$ ist der Subgraph vom Suchgraph der induziert wird von den Knoten die von $v$ aus erreicht werden können
- Der noch nicht kontrahierte Graph heißt Core
- $\pi^{-1}$ heißt rank
- Eingefügte Kanten heißen Shortcuts

# 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.

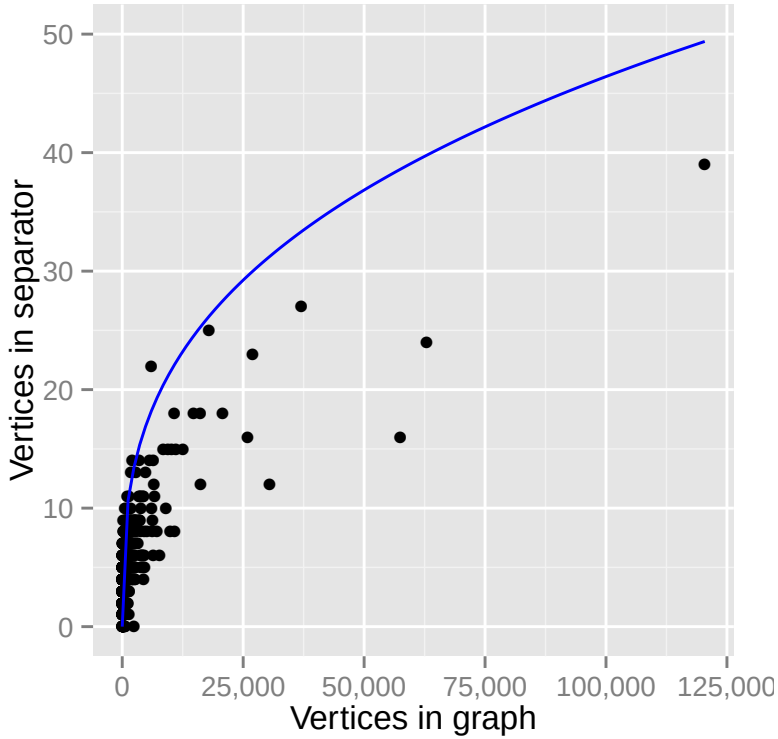
# Knoten Separatoren

## Beispiel

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

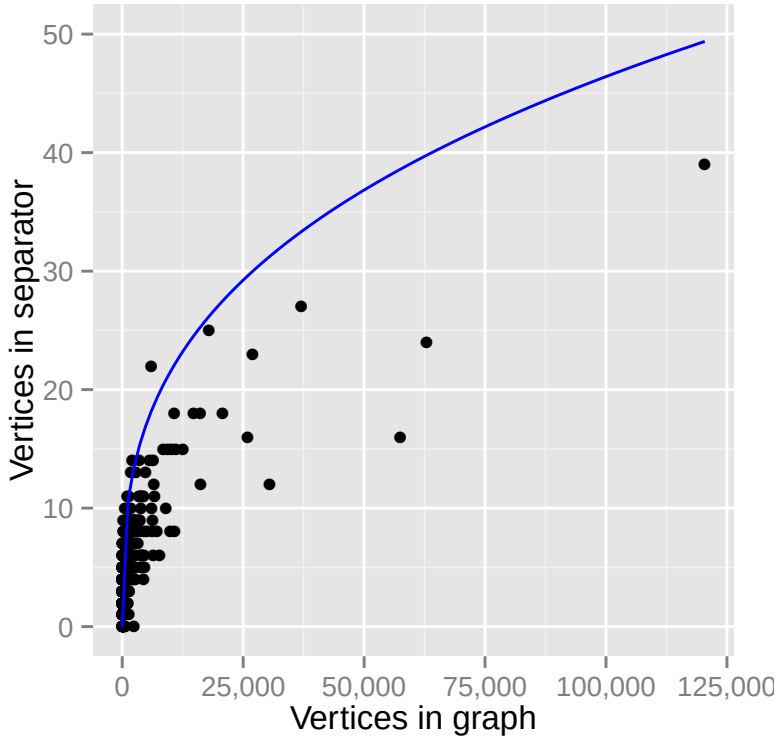
**Beweis**: Siehe Vorlesung zu planaren Graphen

# Knoten Separatoren




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

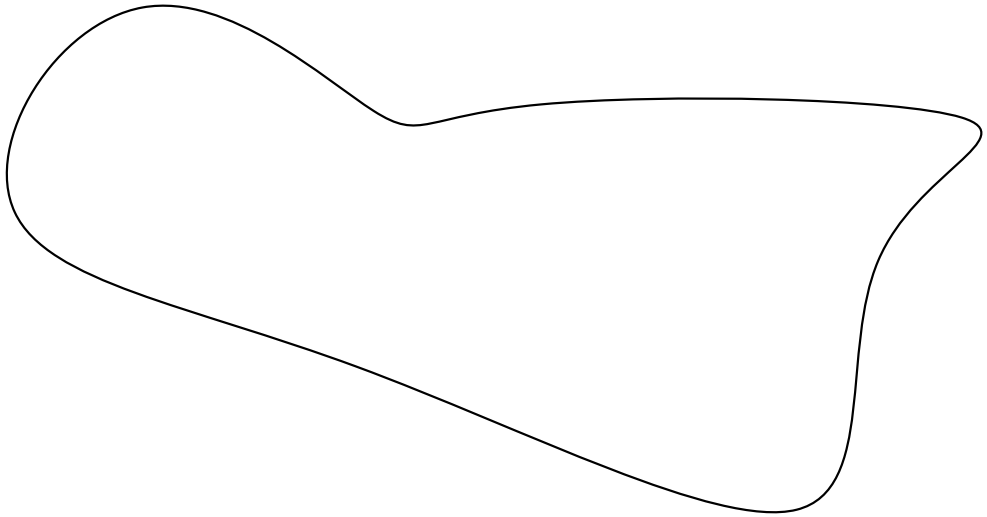
# Knoten Separatoren




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

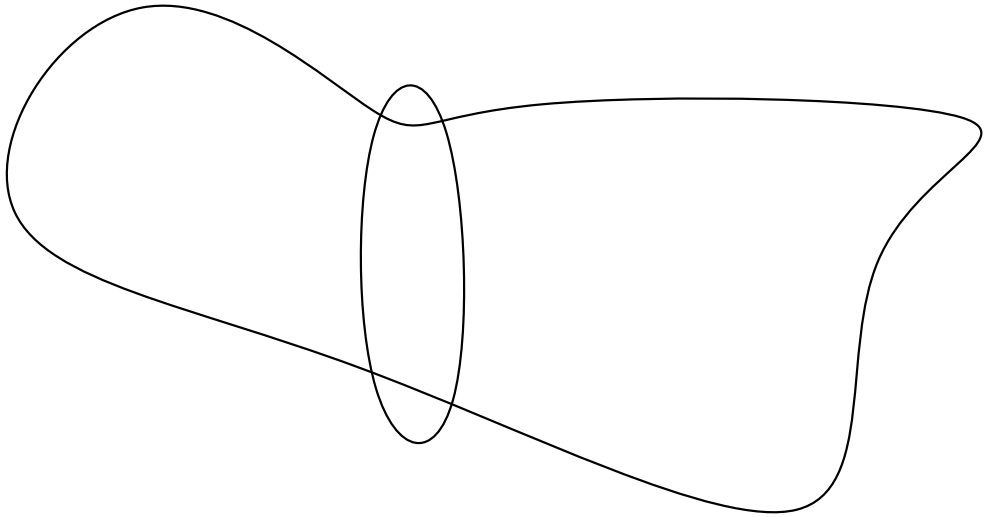


Order:



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

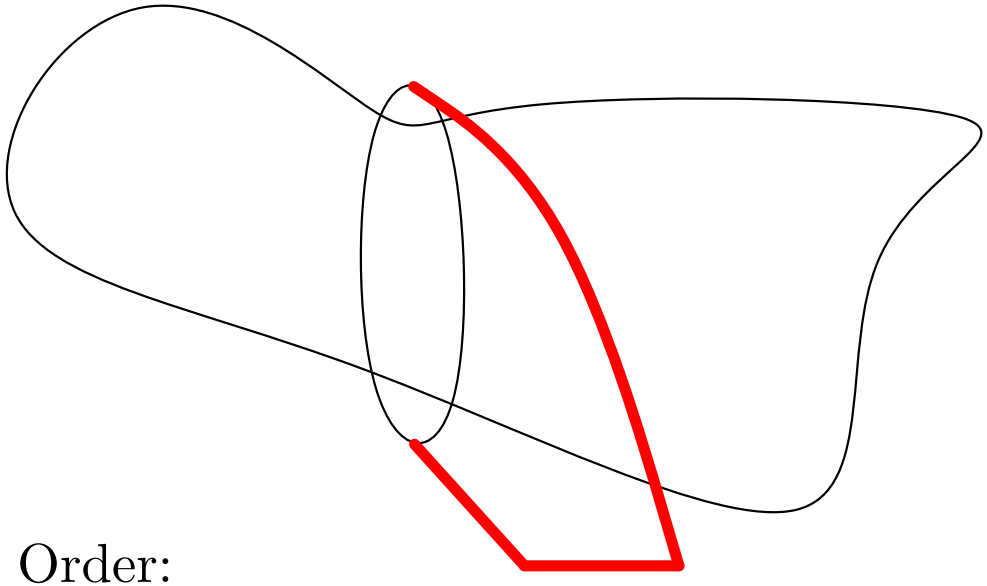
# Nested Dissection



Order:

Finde kleinen Separator

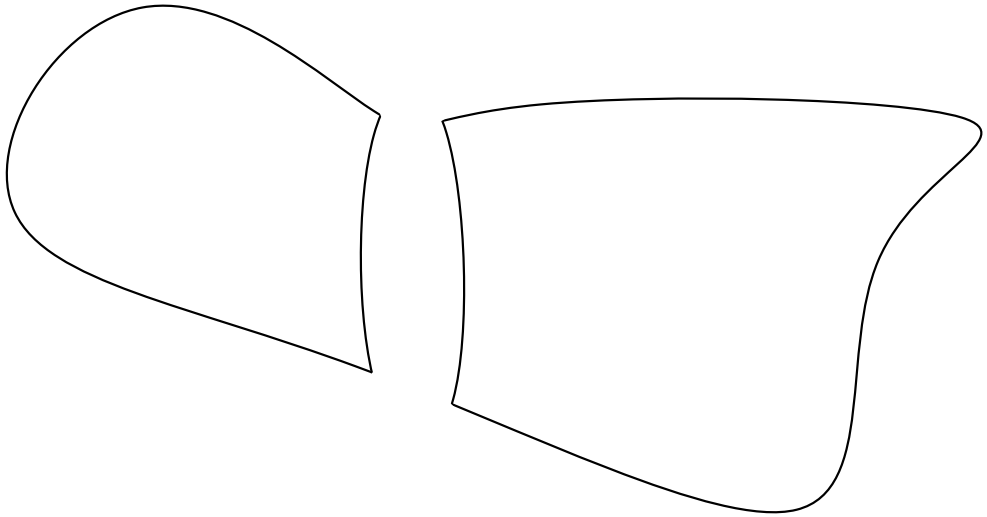
# Nested Dissection



Order:

Pack die Knoten hinten in die Ordnung

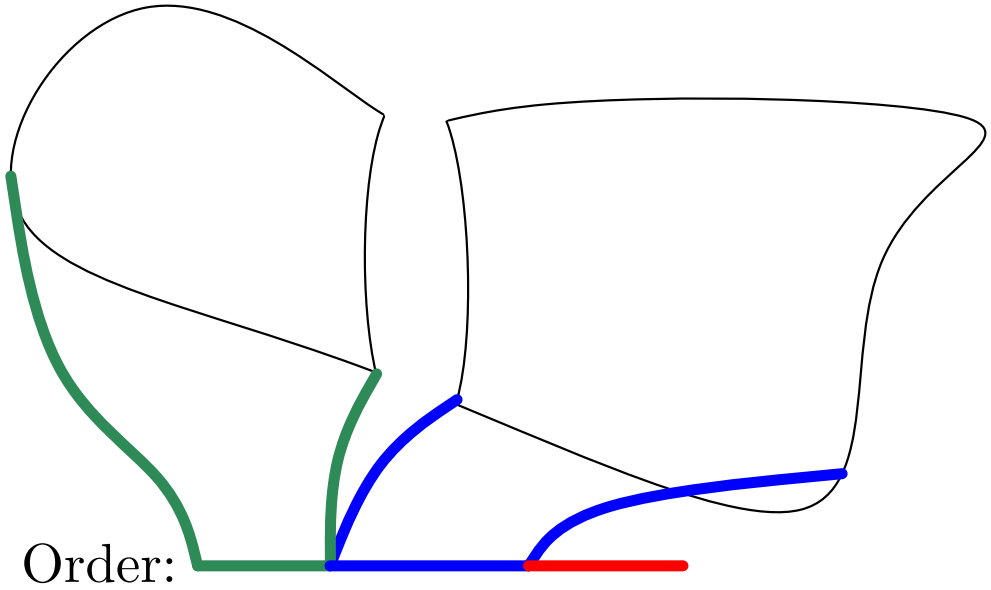
# Nested Dissection



Order:

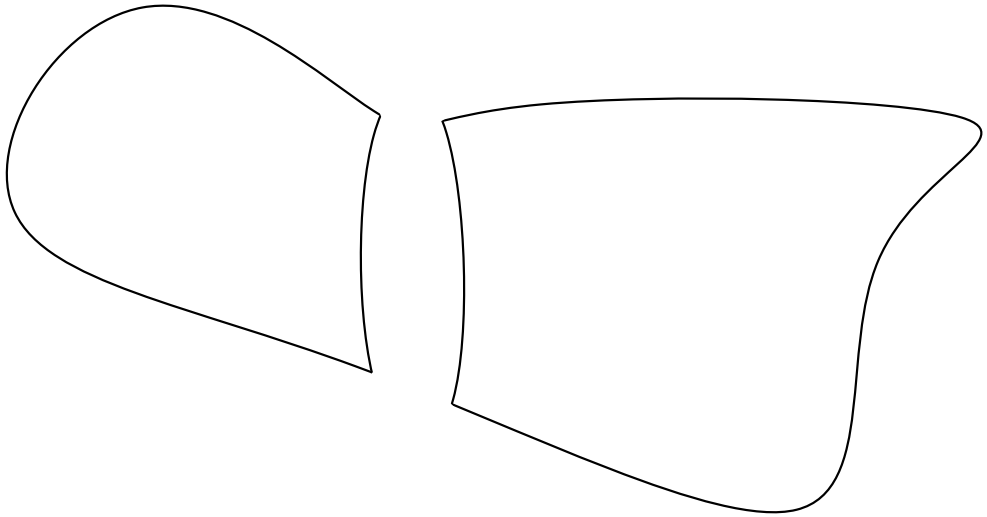
Lösche Separator

# Nested Dissection




Rekursion auf beiden Teilen

# Nested Dissection




Die Ordnung

# Suchraum Größe

- Der Suchraum eines Knoten $v$ enthält auf jeder Ebene höchstens einen Separator, da $v$ nicht auf beiden Seiten sein kann.
- Wir können die Knotenanzahl im Suchraum wie folgt abschätzen:

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

- Es gibt höchstens $O(n^{2/3})$ Kanten im Suchraum

# 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: viel mehr Kanten im Suchraum verglichen mit Metrikabhängigen Ordnungen

# Dreiecke

Ein Dreieck $\{x, y, z\}$ ist

- ein unteres Dreieck von $(x, y)$ wenn $z$ einen kleineren Rank hat, als $x$ und $y$
- ein oberes Dreieck von $(x, y)$ wenn $z$ einen größeren Rank hat, als $x$ und $y$
- ein mittleres Dreieck von $(x, y)$ wenn $z$ einen Rank zwischen denen von $x$ und $y$ hat

# Customization

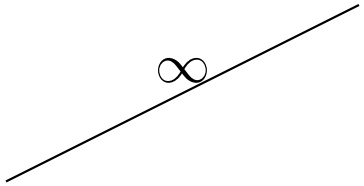
- Eine CH-Anfrage ist korrekt wenn für jedes untere Dreieck die untere Dreiecksungleichung gilt, d.h., für alle Kanten $(x, y)$ im Suchgraph muss gelten, dass für alle unteren Dreiecke $\{x, y, z\}$

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

  gilt
- Initial bekommt jede original Kante ihr original Gewicht, alle Shortcuts kriegen Gewicht $\infty$
- Ziel der Customization ist es die untere Dreiecksungleichung herzustellen

# Customization

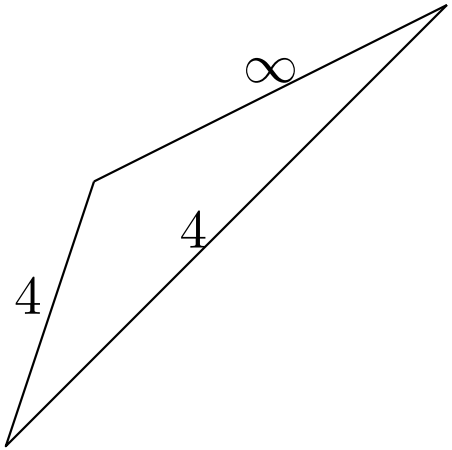
## Normal Customization




Eine Kante im Suchgraph

# Customization

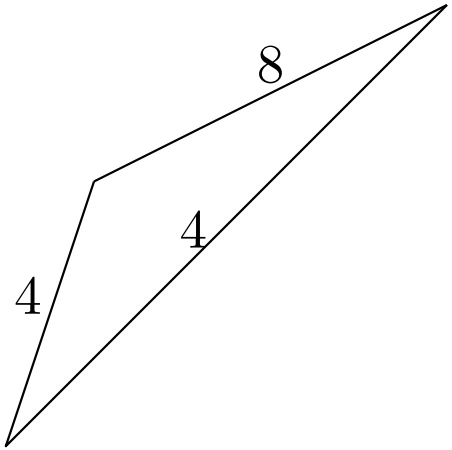
## Normal Customization




Für jedes untere Dreieck muss die Ungleichung gelten

# Customization

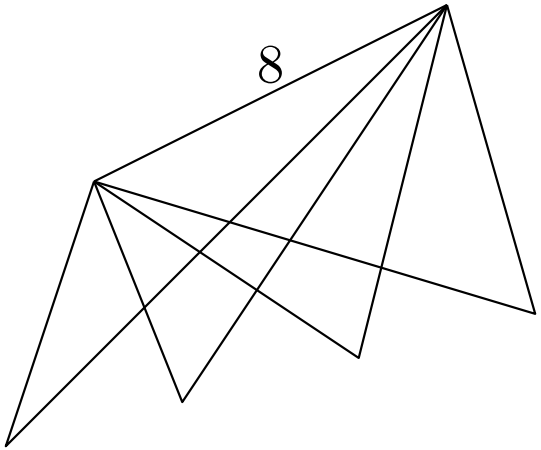
## Normal Customization




Verringere das Gewicht bis die Ungleichung gilt

# Customization

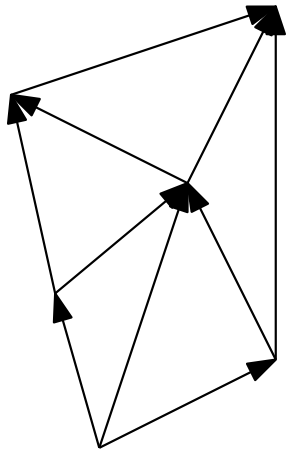
## Normal Customization




Dies muss für alle unteren Dreiecke gemacht werden

# Customization

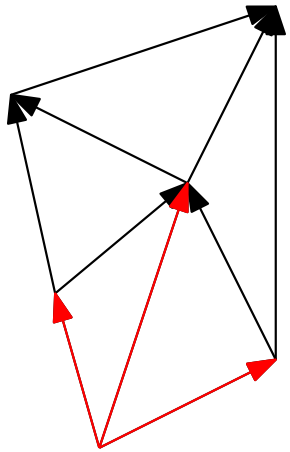
## Normal Customization



Bleibt die Frage in welcher Reihenfolge wir über die Kanten iterieren

# Customization

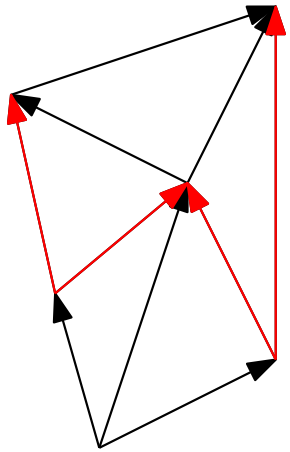
## Normal Customization



Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

# Customization

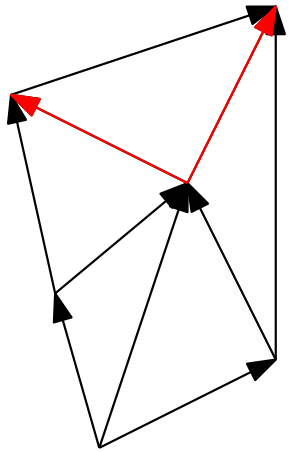
## Normal Customization



Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

# Customization

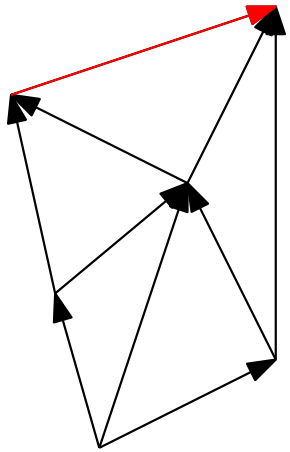
## Normal Customization



Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

# Customization

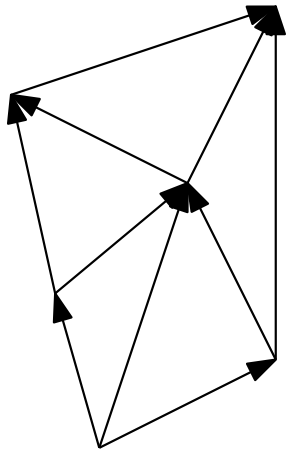
## Normal Customization



Iteriere über alle Kanten im Suchgraph aufsteigen nach Level

# Customization

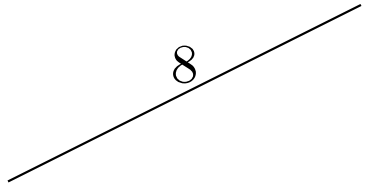
## Normal Customization



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

# Customization

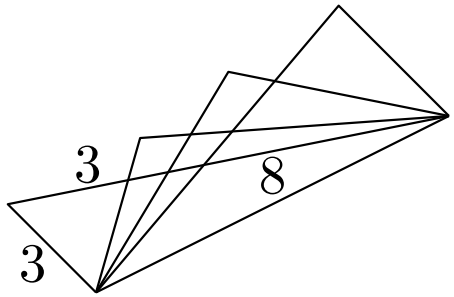
## Perfect Customization




Wir können danach zusätzliche alle mittleren und oberen Dreiecke betrachten. Diesmal gehen wir absteigend nach Level über die Kanten. Das ist eine perfekte Customization.

# Customization

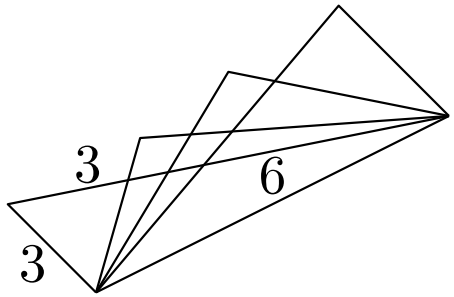
## Perfect Customization




Wir können danach zusätzliche alle mittleren und oberen Dreiecke betrachten. Diesmal gehen wir absteigend nach Level über die Kanten. Das ist eine perfekte Customization.

# Customization

## Perfect Customization




Wir können danach zusätzliche alle mittleren und oberen Dreiecke betrachten. Diesmal gehen wir absteigend nach Level über die Kanten. Das ist eine perfekte Customization.

# Perfekte Customization

- Mit einer perfekten Customization kann man eine perfekte Zeugensuche machen.
- Dies führt zu langsameren Customizationzeiten aber schnelleren Anfragezeiten.
- Achtung: Perfekte Customization geht nur mit metrikunabhängiger Ordnung.
- Mit gewichtsabhängigen Ordnung kriegt man kleinere Suchräume hin.

# 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 $x$

# 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:**

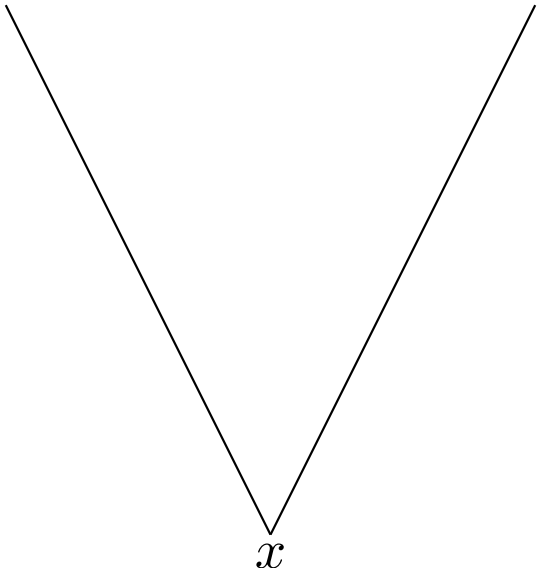
- $V(x) \subseteq S(x)$
- Klar, der Pfad entlang des Baums im Suchraum ist
- Bleibt zu zeigen, dass $V(x) \supseteq S(x)$
- Beweis per Widerspruch

# 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 $x$



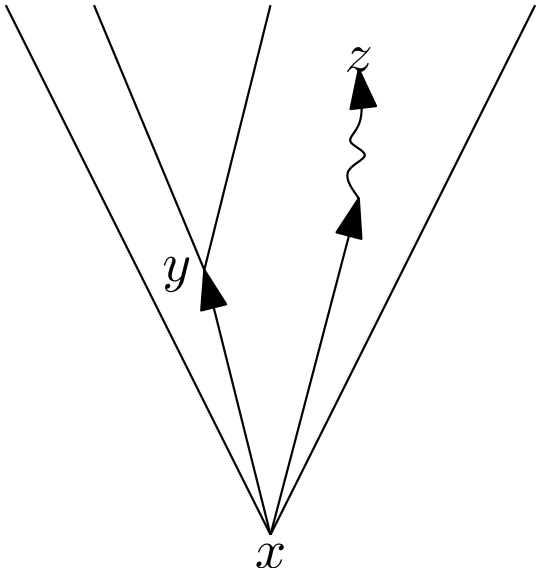

Der Suchraum von $x$.

# 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 $x$



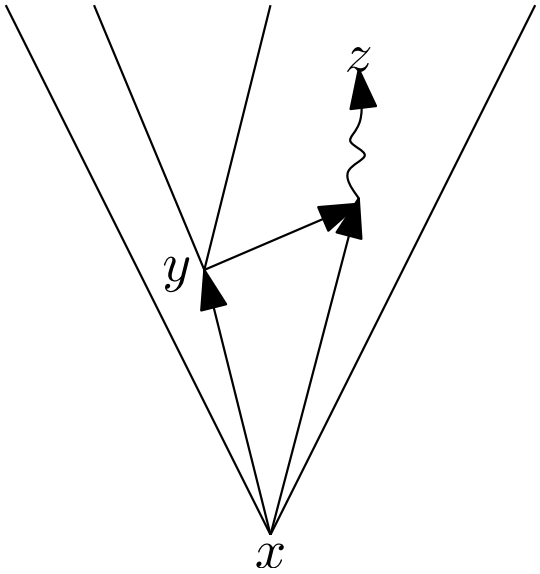

Sei $\text{parent}(x) = y$.
Annahme: $z$ im Suchraum von $x$ aber nicht im Suchraum von $y$ ist.

# 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 $x$



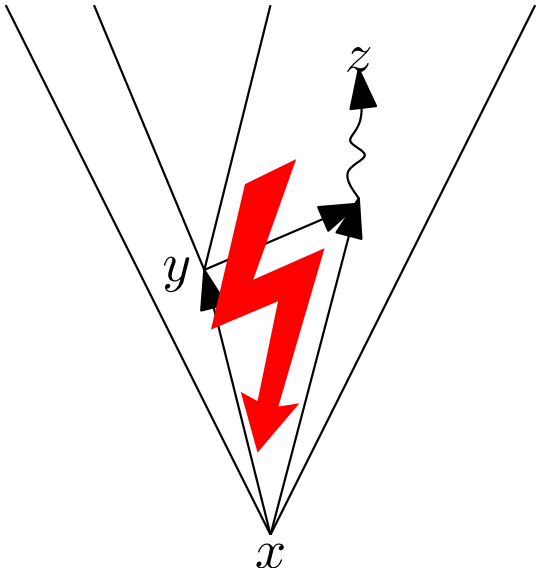

Diese Kante muss es per Konstruktion geben.

# 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

# 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 $\infty$ in eine Richtung

# 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

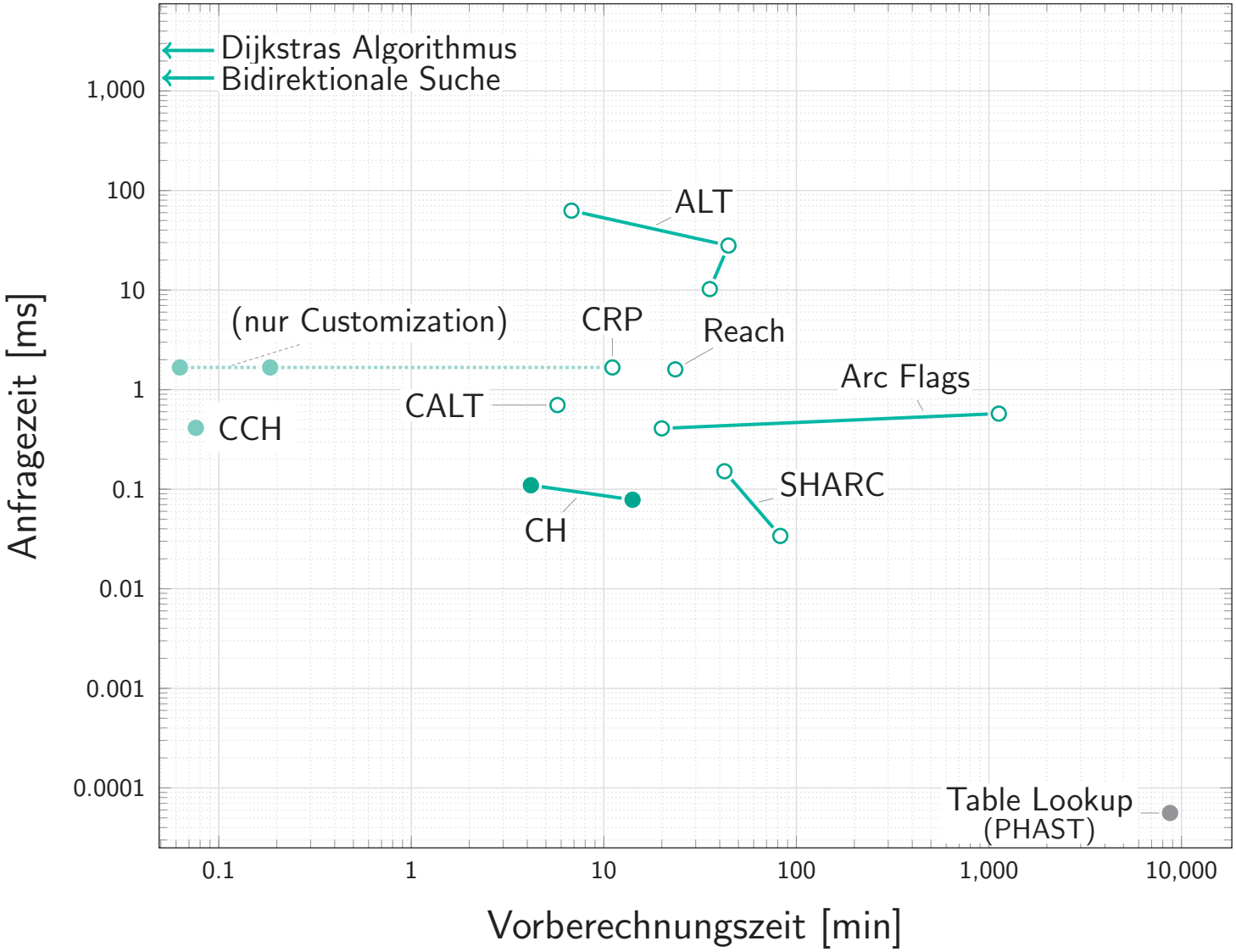
- Hängt vom verwendeten Partitionierer ab
- Metis ist schnell, KaHip hat gute Qualität
- KaHip: 2.8 days (nicht auf Schnelligkeit optimiert)
- Metis: 2.2 min
- Details in [DSW14]

## Metricabhängige Ordnungen

- Bottom-Up: 8-100 min (depending on importance heuristic)
  Parallel mit Reisezeit geht auch 2min
- Top-Down: 29.75 h (mit zusätzlichen Tricks aus [ADGW12])
- Sampling Path Greedy: 139.4 min (mit zusätzlichen Tricks aus [DGPW14])

as always: instance is DIMACS Europe
sequential unless mentioned

# Overview

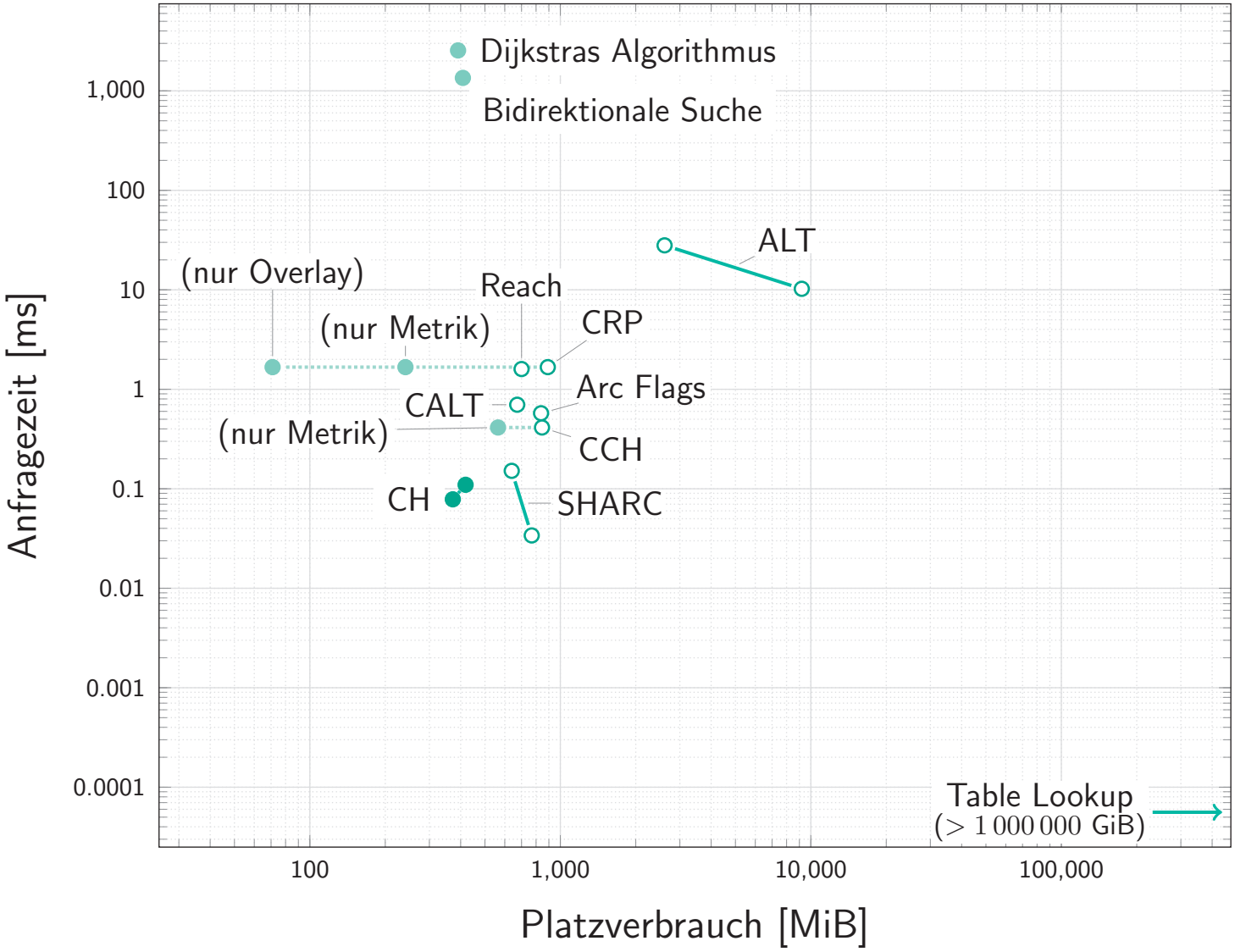






Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# 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



Institut für Theoretische Informatik
Lehrstuhl Algorithmik

# 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 $\rightarrow$ :)

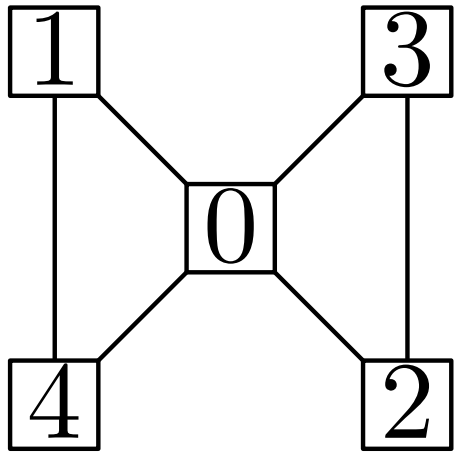
# Dünnbesetzte Gleichungssysteme

Warum funktioniert das?

# Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

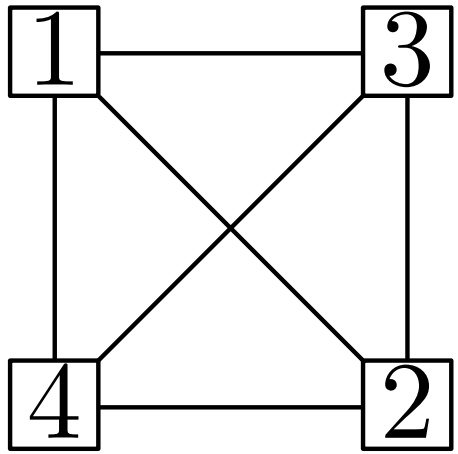
$$\begin{bmatrix} 1 & -1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -2 \\ 1 & 0 & -1 & -1 & 0 \\ -1 & 0 & -1 & -2 & 0 \\ 1 & -1 & 0 & 0 & 1 \end{bmatrix}$$



# Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

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

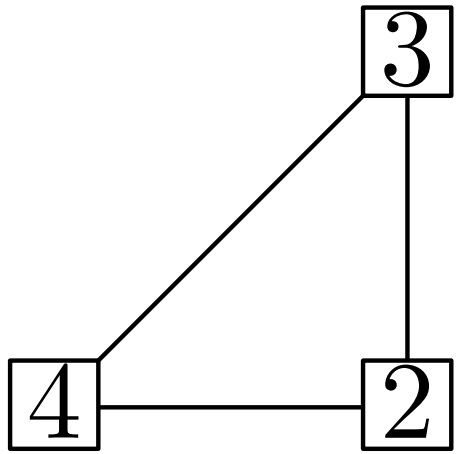



Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

# Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

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

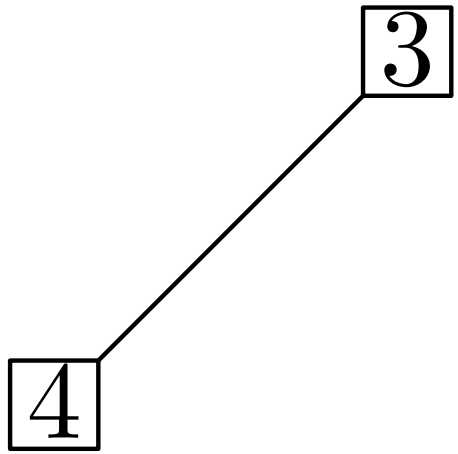



Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

# 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.

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

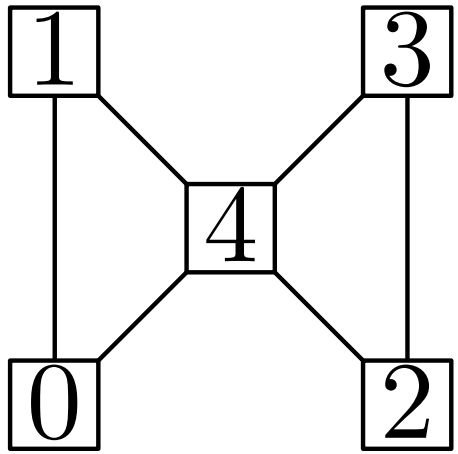
$\boxed{4}$

Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

# Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

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

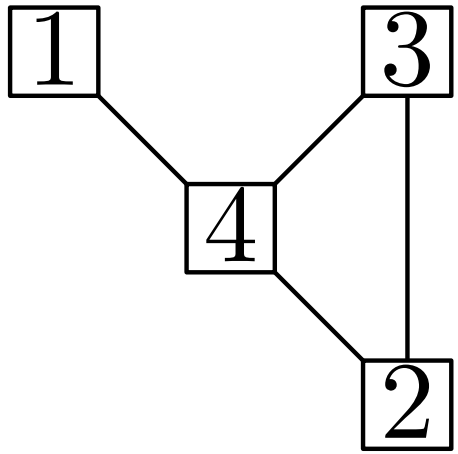



Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

# Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

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

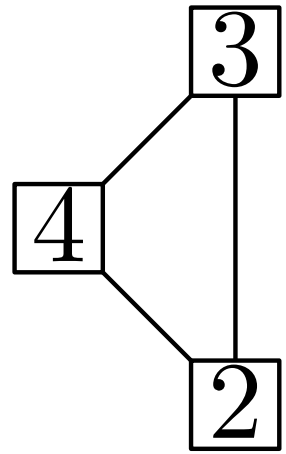



Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

# Dünnbesetzte Gleichungssysteme

Jeder Eintrag der nicht Null ist entspricht einer Kante.

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

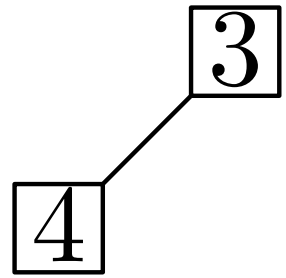



Variablenelimination $\leftrightarrow$ Knotenkontraktion
Jeder Shortcut zerstört eine Null

# 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.

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.