

Praktikum Routenplanung

Vorbesprechung, Wintersemester 2017/2018

Moritz Baum, Valentin Buchhold, Ben Strasser, Tim Zeitz, Tobias Zündorf | 18. Oktober 2017

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



Praktikum

- Erste Phase: 1 Übungsblatt mit 4 Aufgaben lösen
- Zweite Phase: Große Aufgabe in **Gruppen** à 3 Studenten
- Betreuer: Moritz Baum, Valentin Buchhold, Ben Strasser, Tim Zeitz, Tobias Zündorf
- Email: {moritz.baum, buchhold, strasser, tim.zeitz, tobias.zuendorf}@kit.edu
- 6 LP/ECTS
- Bei Fragen einfach vorbei kommen

Homepage: <http://illwww.itl.kit.edu/teaching/winter2017/algorithmengineeringpraktikum/>

Voraussetzungen

- Ihr seid im Master Informatik, **alle anderen bitte melden**
- Ihr habt Interesse an algorithmischen Fragestellungen
- Ihr mögt Algorithmen implementieren
- Ihr könnt C++

Übersicht

- Übungsaufgabe
- Anfangsvortrag
- Gruppenaufgabe
- Abschlussvortrag
- Ausarbeitung

Voraussetzungen

- Ihr seid im Master Informatik, **alle anderen bitte melden**
- Ihr habt Interesse an algorithmischen Fragestellungen
- Ihr mögt Algorithmen implementieren
- Ihr könnt C++

Übersicht

- Übungsaufgabe
- Anfangsvortrag
- Gruppenaufgabe
- Abschlussvortrag
- Ausarbeitung

Voraussetzungen

- Ihr seid im Master Informatik, **alle anderen bitte melden**
- Ihr habt Interesse an algorithmischen Fragestellungen
- Ihr mögt Algorithmen implementieren
- Ihr könnt C++

Übersicht

- Übungsaufgabe
- Anfangsvortrag
- Gruppenaufgabe
- Abschlussvortrag
- Ausarbeitung

Übungsblatt

- Es werden Punkte vergeben
 - Punkte gehen nicht in die Endnote ein
 - 15% der Punkte müssen erreicht werden um zu bestehen
- Gruppen nach Punktzahl gebildet
- Gruppe mit den meisten Punkten darf sich Gruppenarbeitsthema zuerst aussuchen
- Nach Übungsblatt: formale Prüfungsanmeldung d.h. ab da: nichts gemacht → durchgefallen
- Gruppenarbeit ist schwerer als Übungsblatt

Gruppenarbeit

- Bearbeitung in 3er Gruppe
- Aufgabe:
 - Reimplementieren eines Forschungspapers
 - Nicht jede Gruppe hat das selbe Paper
- Visualisierung der Ergebnisse
- Einige Experimente aus dem Paper wiederholen
- Einige neue Experimente entwerfen und durchführen

Einteilung und Themen

- Gruppeneinteilung nach Übungsblatt
- Themenvorstellung bei Gruppeneinteilung

Anfangsvortrag

- 10 min
- Problemstellung und den Kernansatz erklären

Ausarbeitung

- Alles was ihr implementiert habt, in eigenen Worten beschreiben
- Experimente und Ergebnisse dokumentieren

Abschlussvortrag

- 20min-30min
- Inhalte der Ausarbeitung vorstellen

Aufwand

- 6 ETCS/LP
- $6 \cdot 30h = 180h$
- Bei 20 Wochen: 9h pro Woche,
also leicht mehr als 1 Tag Vollzeit pro Woche

Grobe Verteilung

- 35h Übungsblatt
- 95h Gruppenaufgabe
inklusive
 - Einarbeitung ins Thema
 - Implementierung
- 5h Kurzvortrag
- 20h Abschlussvortrag
- 20h Ausarbeitung
- 5h Anwesenheit

Problemstellung

Gesucht:

- Finde die **beste** Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Pfad durch Graph entspricht Route
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind **groß**
- Dijkstra zu **langsam** (> 1 Sekunde)



Problemstellung

Gesucht:

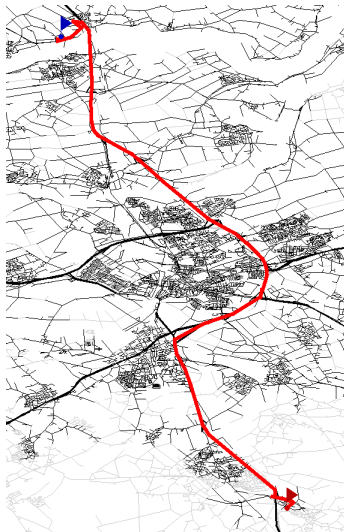
- Finde die **beste** Verbindung in einem Transportnetzwerk

Idee:

- Netzwerk als Graphen $G = (V, E)$
- Pfad durch Graph entspricht Route
- klassisches Problem (Dijkstra)

Probleme:

- Transportnetzwerke sind **groß**
- Dijkstra zu **langsam** (> 1 Sekunde)

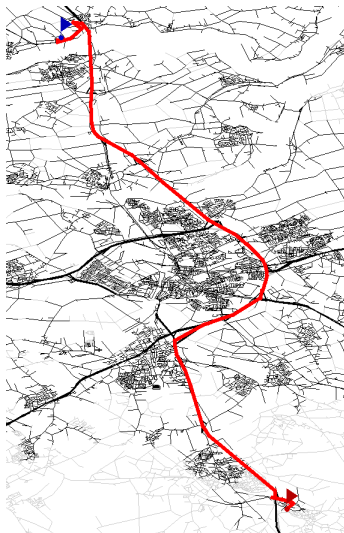


Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen **unnötig**

Idee:

- Zwei-Phasen Algorithmus:
 - offline: berechne Zusatzinformation während **Vorbereitung**
 - online: **beschleunige** Berechnung mit diesen Zusatzinformationen
- drei Kriterien:
 - wenig Zusatzinformation
 - kurze Vorbereitung (im Bereich Stunden/Minuten)
 - hohe Beschleunigung

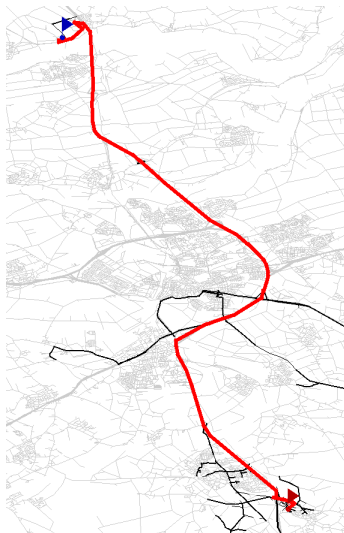


Beobachtungen:

- viele Anfragen in (statischem) Netzwerk
- manche Berechnungen scheinen **unnötig**

Idee:

- Zwei-Phasen Algorithmus:
 - offline: berechne Zusatzinformation während **Vorbereitung**
 - online: **beschleunige** Berechnung mit diesen Zusatzinformationen
- drei Kriterien:
 - wenig Zusatzinformation
 - kurze Vorbereitung (im Bereich Stunden/Minuten)
 - hohe Beschleunigung



Modellierung (Straßengraphen)

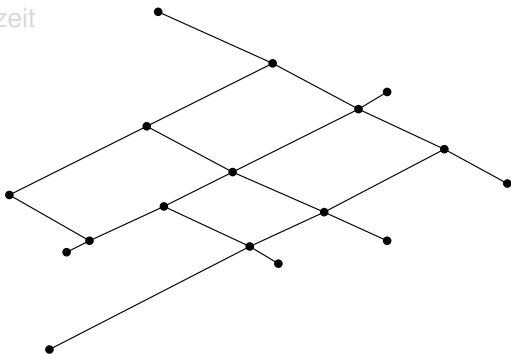


Modellierung (Straßengraphen)



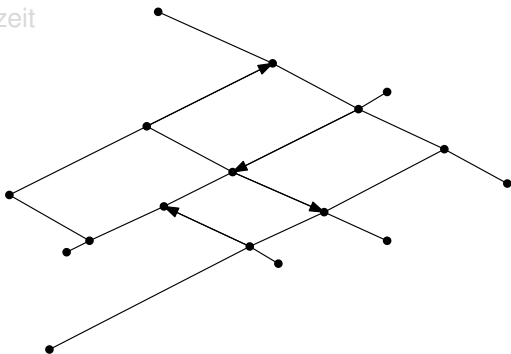
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



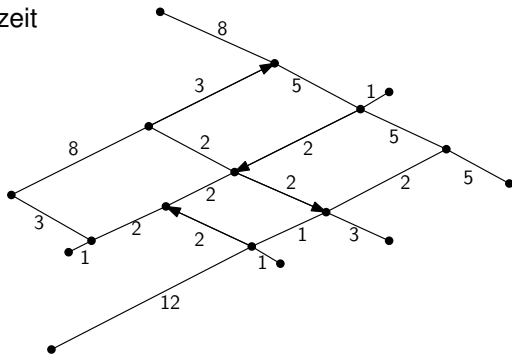
Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



Modellierung (Straßengraphen)

- Knoten sind Kreuzungen
- Kanten sind Straßen
- Einbahnstraßen
- Metrik ist Reisezeit



- Pro Aufgabe: Liste an Start- und Zielknotenpaaren
- Ihr soll die Pfadlänge berechnen
- Punkte einer Aufgabe = #Korrekt berechnete Pfadlängen

Bestehen

Es müssen 15% der Punkte erreicht werden um zu bestehen!

- Bei Fragen oder Problemen könnt ihr euch gerne an uns wenden
- Falls danach gefragt wird, dann können wir auch gerne Feedback zu eurem Code geben

- Allerdings: **Eigeninitiative erwünscht**
- Es ist eure Aufgabe bei Problemen auf einen der Betreuer zuzugehen
- Wer nicht fragt, der kriegt keine Hilfe

- Bei Fragen oder Problemen könnt ihr euch gerne an uns wenden
- Falls danach gefragt wird, dann können wir auch gerne Feedback zu eurem Code geben

- Allerdings: **Eigeninitiative erwünscht**
- Es ist eure Aufgabe bei Problemen auf einen der Betreuer zuzugehen
- Wer nicht fragt, der kriegt keine Hilfe

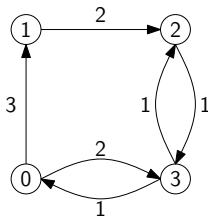
	Wann?	Wo?	Was?
Heute	18.10. um 14:00	SR -120	Vorbesprechung
	16.11., 8:00 ¹ morgens	—	Abgabe Übungsblatt
	16.11.-17.11.	—	Punktevergabe per E-Mail
	22.11. um 14:00	SR -120	Themen & Gruppeneinteilung
	13.12. um 14:00	SR -120	Anfangsvorträge
	16.2.	—	Draft-Version von Ausarbeitung
	1.3. um 14:00	SR -120	Abschlussvorträge
	31.3.	—	Abgabe Ausarbeitung

Es gilt Anwesenheitspflicht. Wer nicht kommen kann muss sich mit Begründung abmelden.

¹Deutsche Zeit

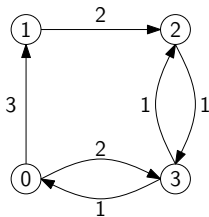
Drei klassische Ansätze:

- Adjazenzmatrix
- (statisches) Adjazenzarray
- Kantenarray



Drei klassische Ansätze:

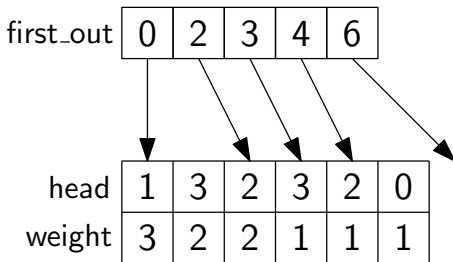
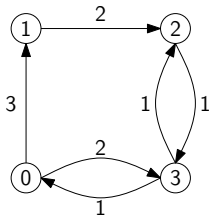
- Adjazenzmatrix
- (statisches) Adjazenzarray
- Kantenarray



	0	1	2	3
0	—	3	—	2
1	—	—	2	—
2	—	—	—	1
3	1	—	1	—

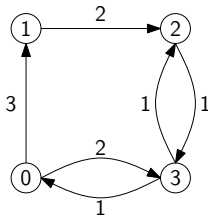
Drei klassische Ansätze:

- Adjazenzmatrix
- (statisches) Adjazenzarray
- Kantenarray



Drei klassische Ansätze:

- Adjazenzmatrix
- (statisches) Adjazenzarray
- Kantenarray



tail	1	0	3	0	2	3
head	2	1	2	3	3	0
weight	2	3	1	2	1	1

Was benutzen wir?

Adjazenzmatrix:

- Braucht $O(n^2)$ Speicher
- $n = 18 \cdot 10^6$
- Speicher $\geq 1/4$ Terrabyte
- Impraktikabel

Kantenarray:

- Perfekt für einfache Transformationen (z.B. Graph umdrehen)
- Traversieren (i.e. Pfadsuche) geht nicht

Adjazenzarray:

- Gut wenn man Pfade suchen will

Adjazenzmatrix:

- Braucht $O(n^2)$ Speicher
- $n = 18 \cdot 10^6$
- Speicher $\geq 1/4$ Terrabyte
- Impraktikabel

Kantenarray:

- Perfekt für einfache Transformationen (z.B. Graph umdrehen)
- Traversieren (i.e. Pfadsuche) geht nicht

Adjazenzarray:

- Gut wenn man Pfade suchen will

Adjazenzmatrix:

- Braucht $O(n^2)$ Speicher
- $n = 18 \cdot 10^6$
- Speicher $\geq 1/4$ Terrabyte
- Impraktikabel

Kantenarray:

- Perfekt für einfache Transformationen (z.B. Graph umdrehen)
- Traversieren (i.e. Pfadsuche) geht nicht

Adjazenzarray:

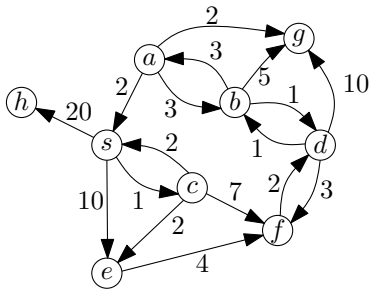
- Gut wenn man Pfade suchen will

Konvertierung Kantenarray \rightarrow Adjazenzarray

- Nach tail sortieren
- Ausgangsgrad jedes Knoten berechnen
- `first_out` = Präfixsumme über Array der Ausgangsgrade

```
1 forall nodes  $v \in V$  do
2   |  $d[v] = \infty$ ;
3  $d[s] = 0$ ;
4  $q.clear()$ ;
5  $q.insert(s, 0)$ ;
6 while  $!q.empty()$  do
7   |  $x \leftarrow q.pop()$ ;
8   | forall edges  $(x, y) \in E$  do
9     | if  $d[x] + \text{len}(x, y) < d[y]$  then
10      |  $d[y] \leftarrow d[x] + \text{len}(x, y)$ ;
11      | if  $y \in q$  then
12        |  $q.decreaseKey(y, d[y])$ 
13      | else
14        |  $q.insert(y, d[y])$ 
```

Dijkstras Algorithmus



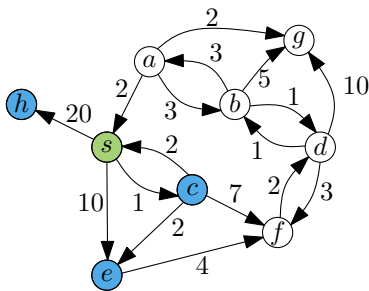
tentative distance d :

ID	Dist.
s	0
a	∞
b	∞
c	∞
d	∞
e	∞
f	∞
g	∞
h	∞

queue q :

ID	Key
s	0

Dijkstras Algorithmus



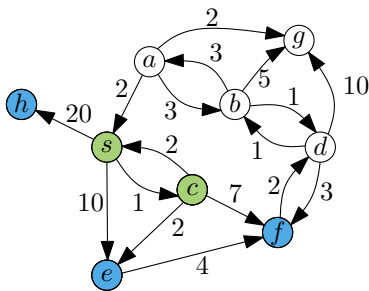
tentative distance d :

ID	Dist.
s	0
a	∞
b	∞
c	1
d	∞
e	10
f	∞
g	∞
h	20

queue q :

ID	Key
c	1
e	10
h	20

Dijkstras Algorithmus



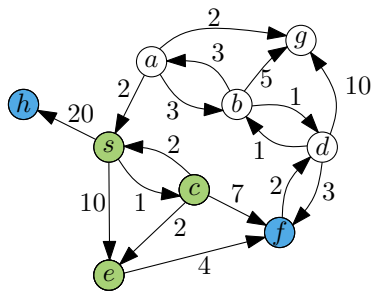
tentative distance d :

ID	Dist.
s	0
a	∞
b	∞
c	1
d	∞
e	3
f	8
g	∞
h	20

queue q :

ID	Key
e	3
f	8
h	20

Dijkstras Algorithmus



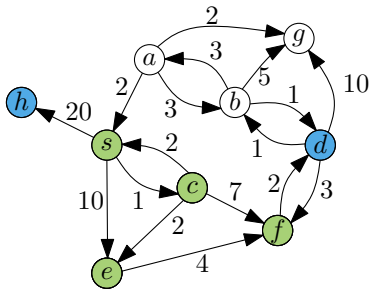
tentative distance d :

ID	Dist.
s	0
a	∞
b	∞
c	1
d	∞
e	3
f	7
g	∞
h	20

queue q :

ID	Key
f	7
h	20

Dijkstras Algorithmus



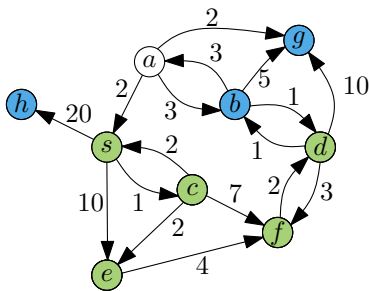
tentative distance d :

ID	Dist.
s	0
a	∞
b	∞
c	1
d	9
e	3
f	7
g	∞
h	20

queue q :

ID	Key
d	9
h	20

Dijkstras Algorithmus



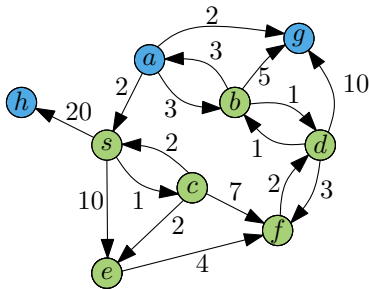
tentative distance d :

ID	Dist.
s	0
a	∞
b	10
c	1
d	9
e	3
f	7
g	19
h	20

queue q :

ID	Key
b	10
g	19
h	20

Dijkstras Algorithmus



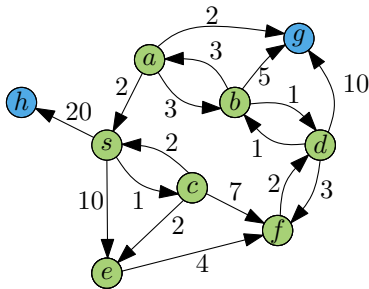
tentative distance d :

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q :

ID	Key
a	13
g	15
h	20

Dijkstras Algorithmus



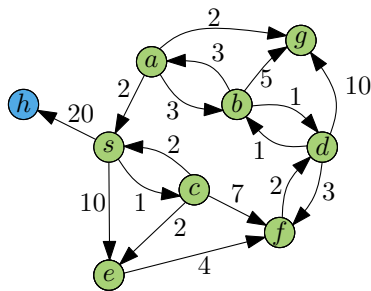
tentative distance d :

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q :

ID	Key
g	15
h	20

Dijkstras Algorithmus



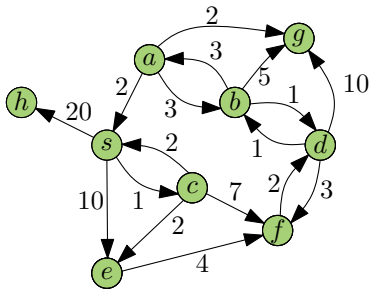
tentative distance d :

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q :

ID	Key
h	20

Dijkstras Algorithmus



tentative distance d :

ID	Dist.
s	0
a	13
b	10
c	1
d	9
e	3
f	7
g	15
h	20

queue q :

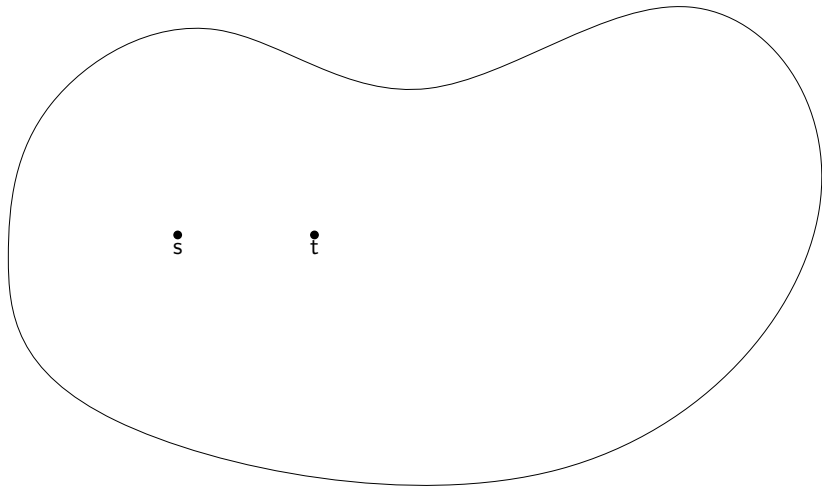
ID	Key

Resultat

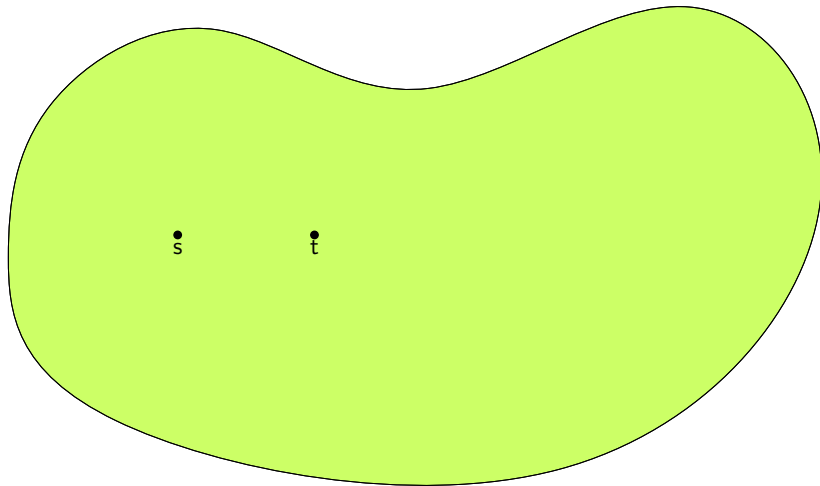
- Nach der Ausführung gilt: $\forall v : d[v] = \text{dist}_G(s, v)$

Stopkriterium

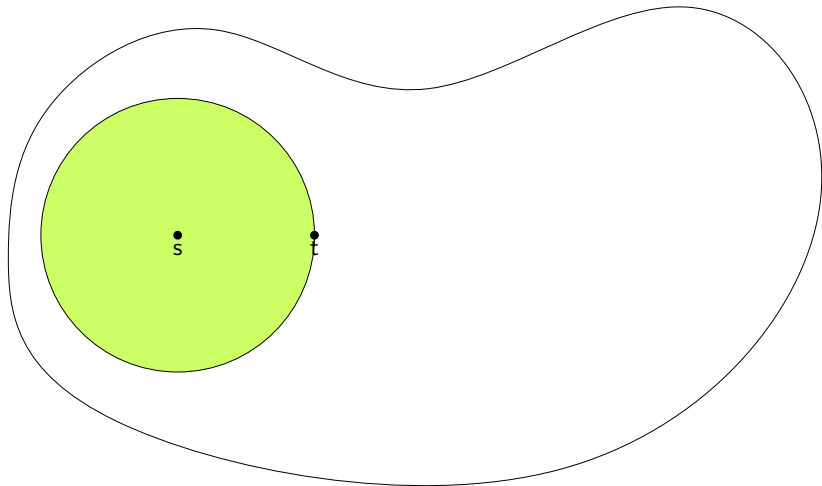
- Geht es schneller, wenn wir $\text{dist}_G(s, t)$ nur für ein t bestimmen müssen?
- Ja: Breche Schleife ab, sobald t aus der Queue genommen wird



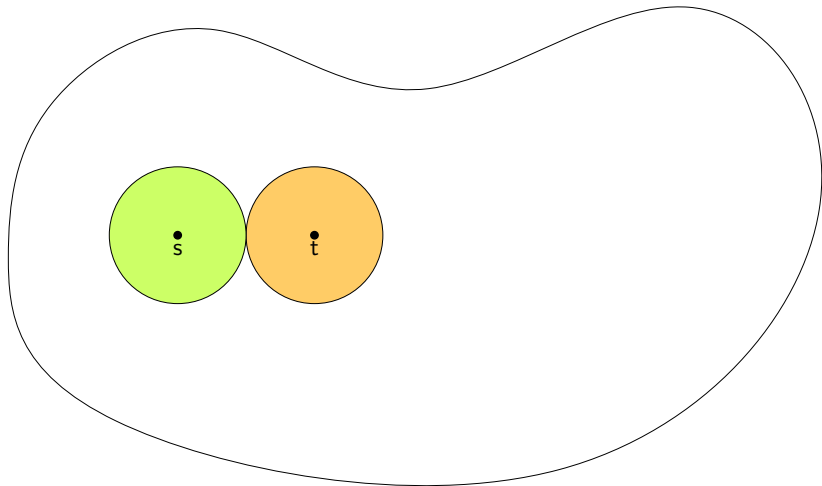
Ein Graph



Suchraum ohne Stopkriterium



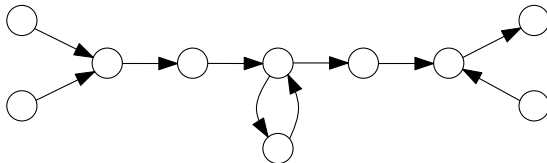
Suchraum mit Stopkriterium



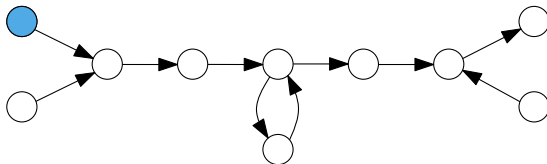
Bidirektionale Variante von Dijkstras Algorithmus

Bidirektionale Variante von Dijkstras Algorithmus

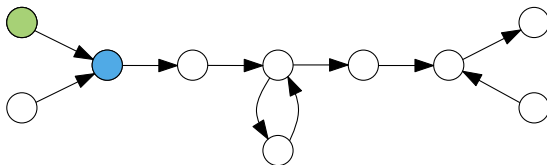
- Bidirektionale Variante von Dijkstras Algorithmus
- Mit zwei Queues und zwei tentative Distanzarrays
- Arbeite die Seite mit den wenigsten Elementen in der Queue als nächstes ab
- Abbruch wenn die Summe der min-keys beider Queues größer ist als der kürzeste bisher gefundene Pfad



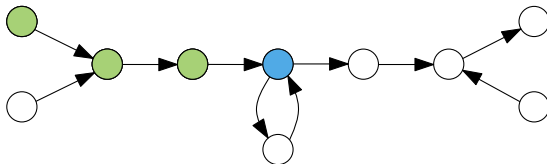
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



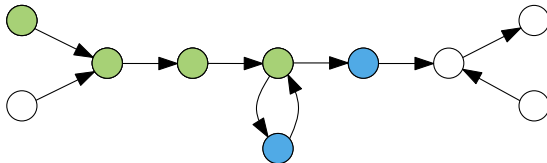
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



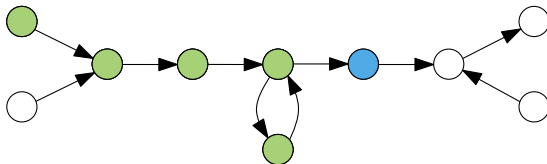
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



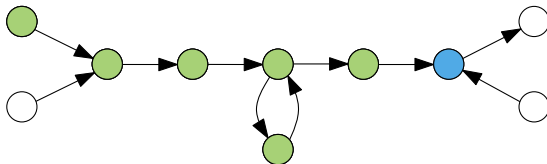
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



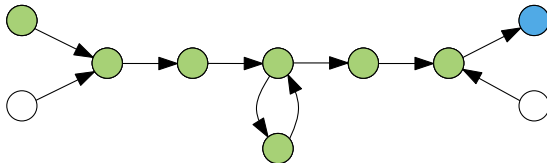
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



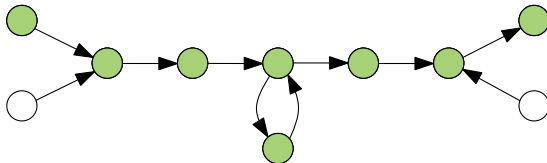
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



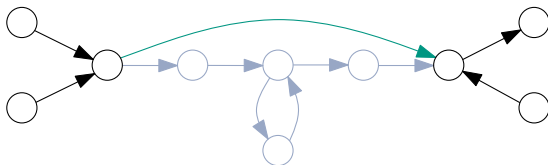
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



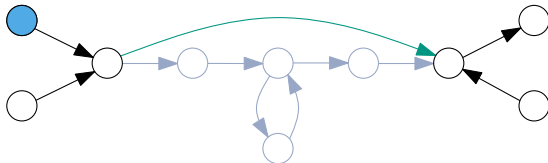
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



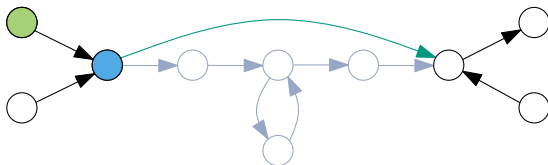
Dijkstra's Algorithmus schaut sich alle Zwischenknoten an.
Das dauert.



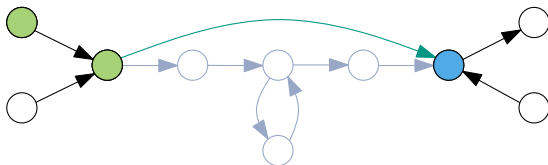
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



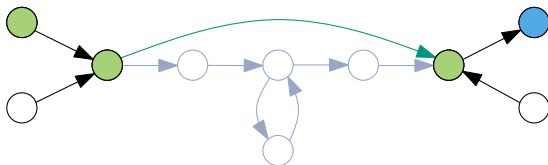
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



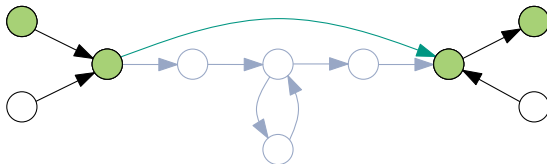
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



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

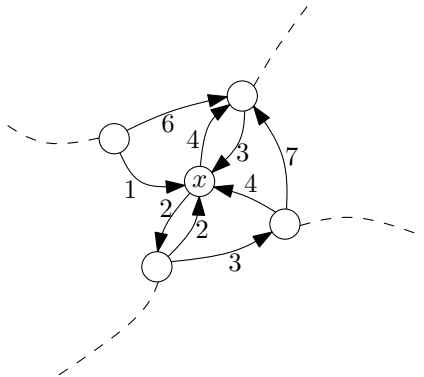


Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.



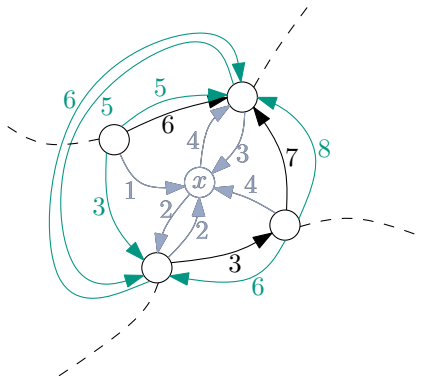
Idee: Füge Shortcut-Kante ein. Grauer Teilgraph muss nur angeschaut werden, wenn s oder t drin liegt.

Knotenkontraktion von x



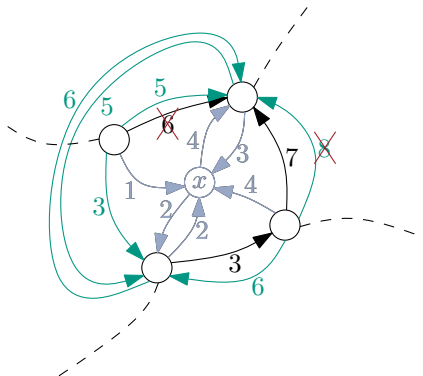
Kontraktion von x : Lösche x und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

Knotenkontraktion von x



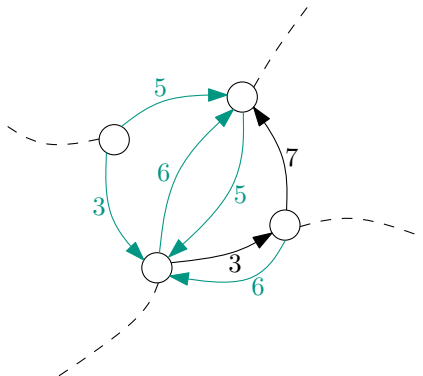
Kontraktion von x : Lösche x und füge Shortcuts zwischen Nachbarn ein, um die Distanzen zwischen allen Knoten zu erhalten

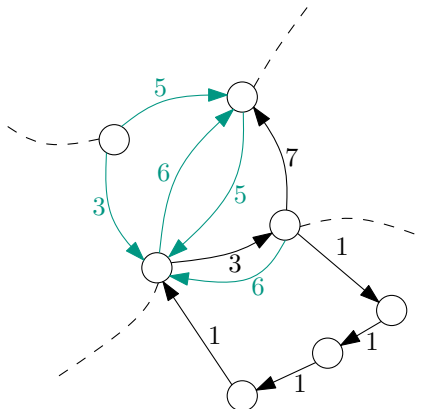
Knotenkontraktion von x



Bei Mehrfachkanten: Längere Kanten verwerfen

Knotenkontraktion von x

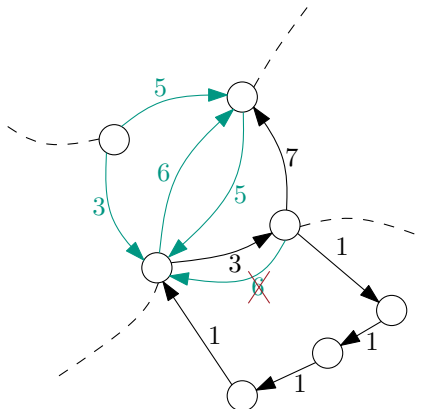




Falls es einen kürzeren Pfad durch den Restgraphen gibt, dann kann man einen Shortcut auch verwerfen.

Suche nach solchem Pfad heißt Zeugensuche/Witness Search

Knotenkontraktion von x



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

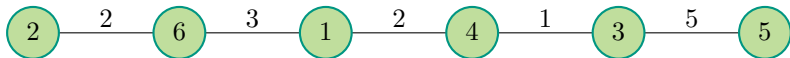
Suche nach solchem Pfad heißt Zeugensuche/Witness Search

- 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 $y \rightarrow x \rightarrow z$ 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.

Grundidee

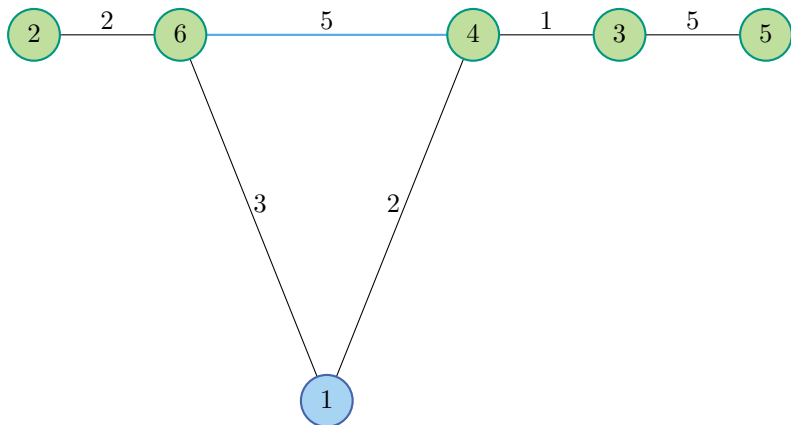
- Eingabe Graph G
- Ordne Knoten von G nach “Wichtigkeit”: $v_1 \dots v_n$
- Kontrahiere Knoten iterative aus G raus
 - zuerst den “unwichtigsten” Knoten v_1
 - den “wichtigsten” Knoten v_n als letztes

Contraction Hierarchy



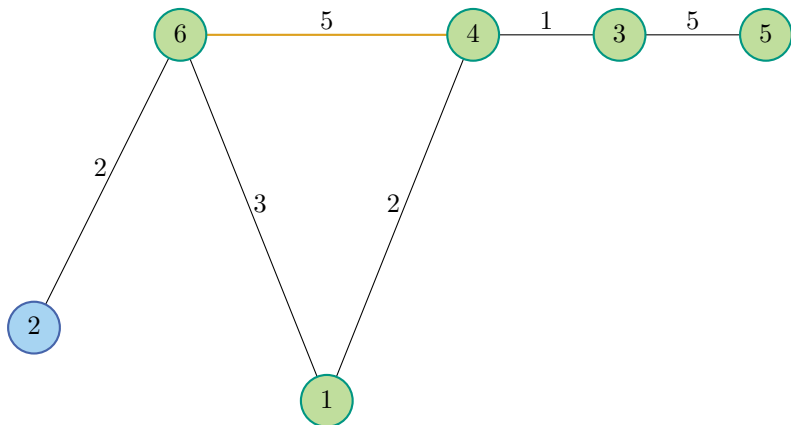
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



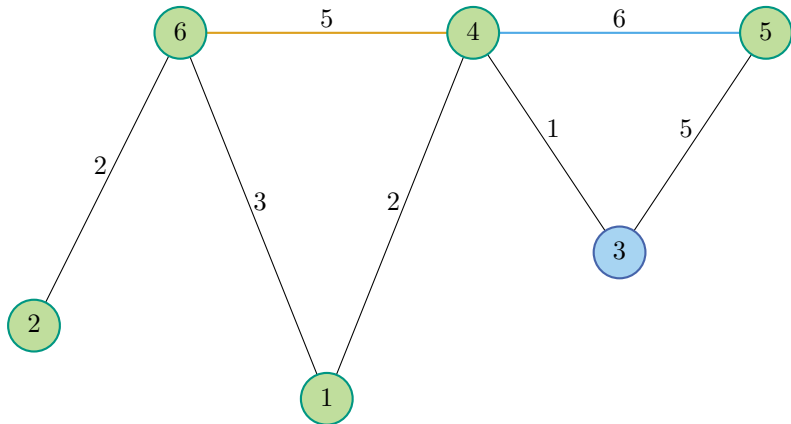
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



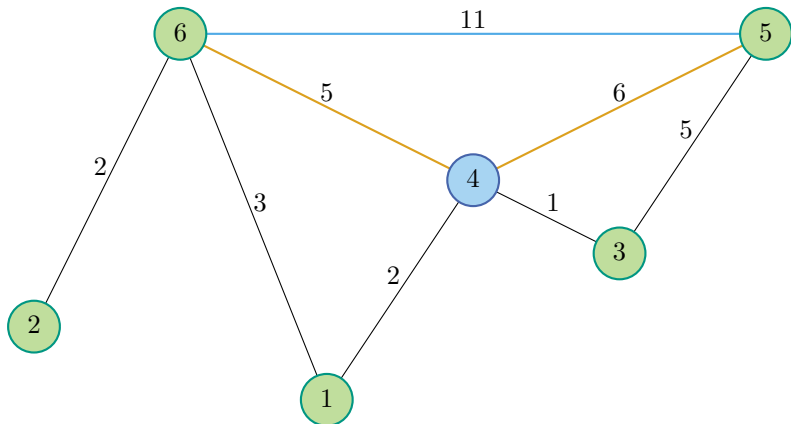
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



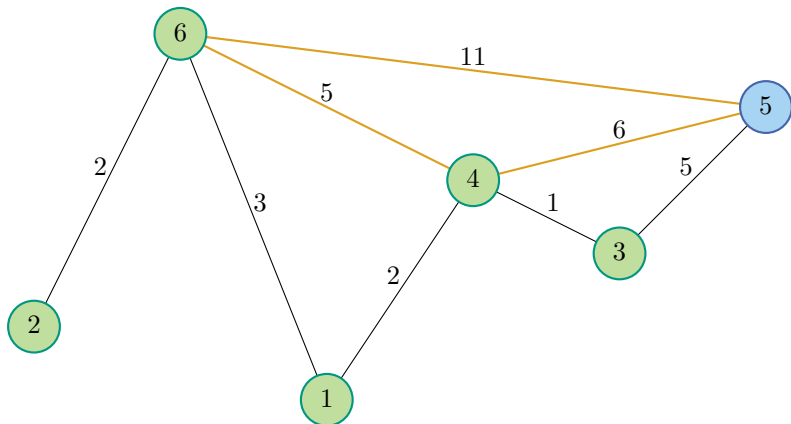
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



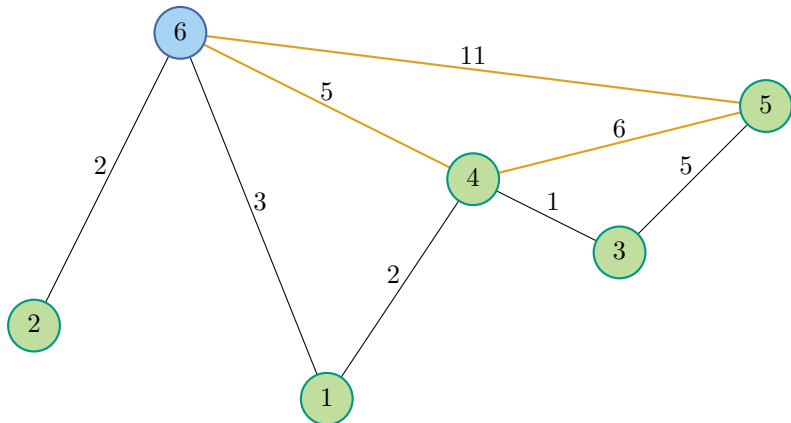
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



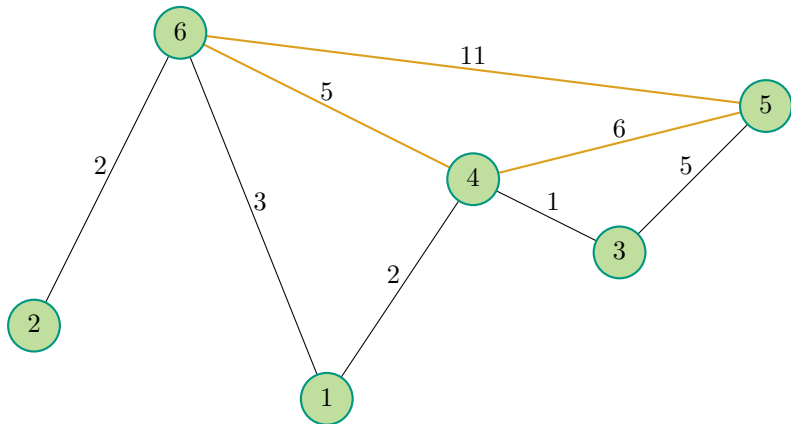
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



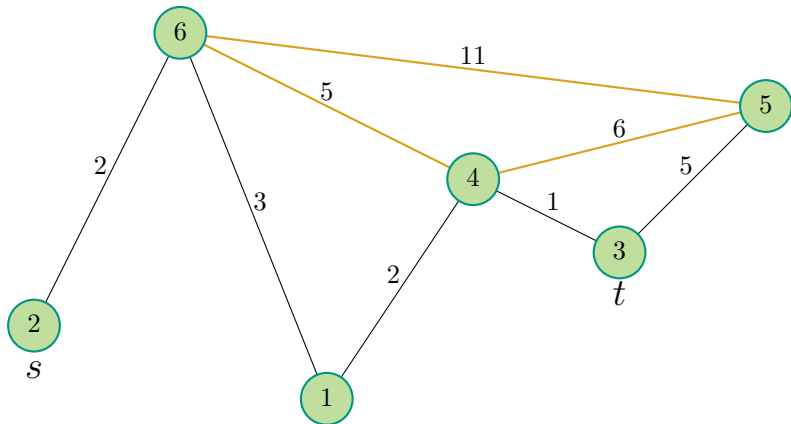
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



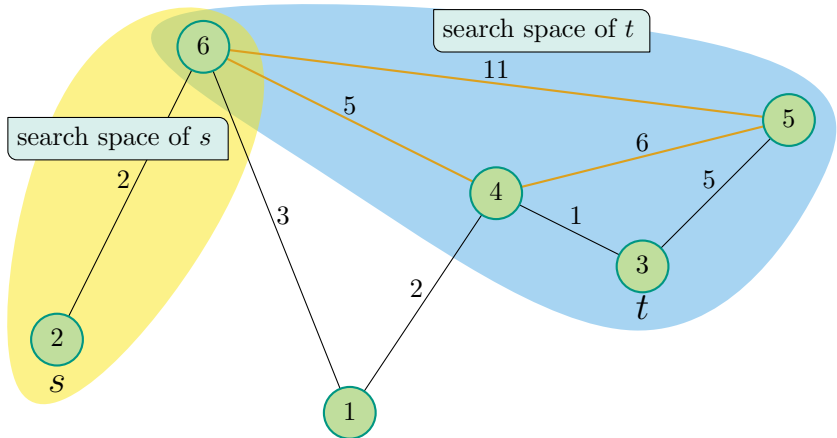
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



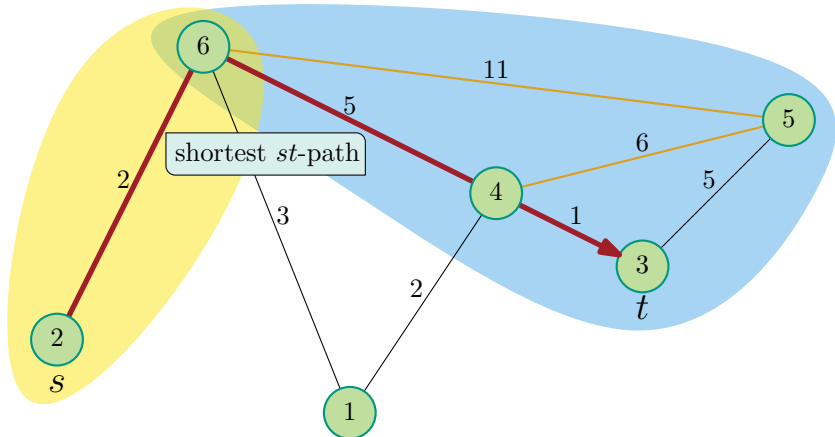
Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



Knoten nummeriert nach "Wichtigkeit"

Contraction Hierarchy



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

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

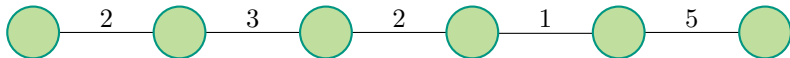
Grund-Idee:

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

Algorithmus:

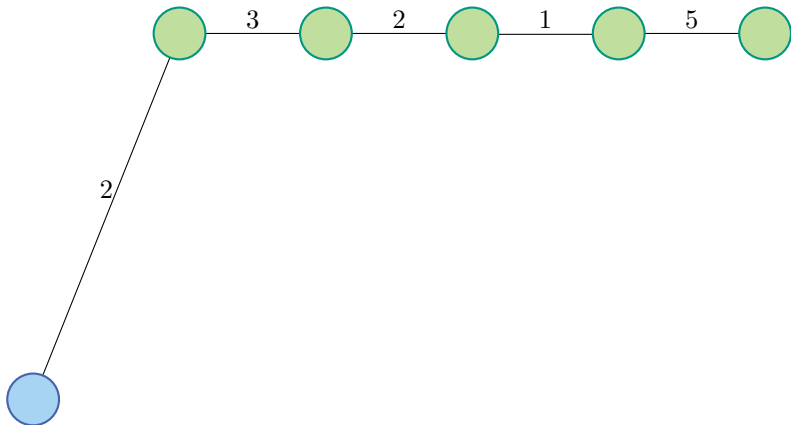
- Baue eine große Warteschlange mit allen Knoten gewichtet nach ihrer “Wichtigkeit”
- Kontrahiere iterative unwichtigsten Knoten
- Kontraktion eines Knoten kann “Wichtigkeit” der Nachbarn beeinflussen
- → “Wichtigkeit” der Nachbarn neu berechnen

Problemfall: Pfad



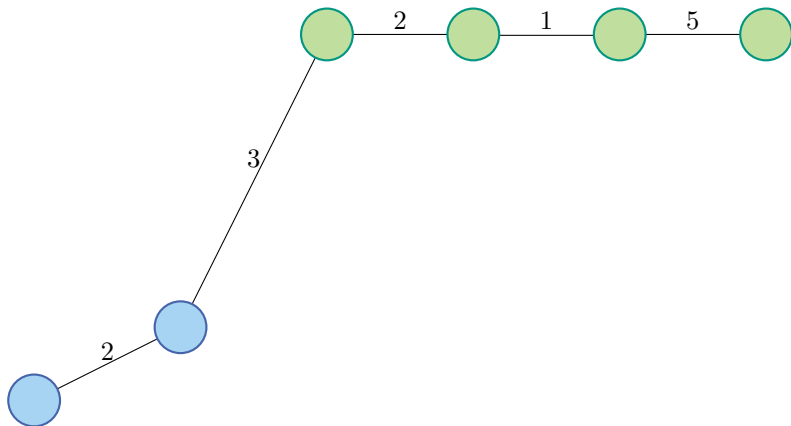
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



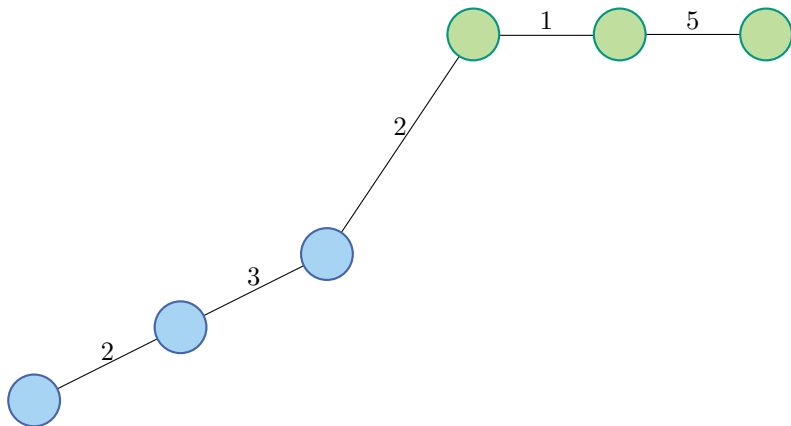
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



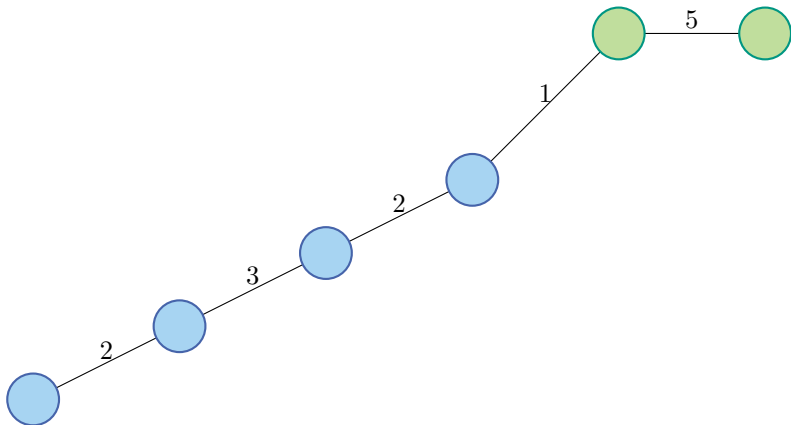
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



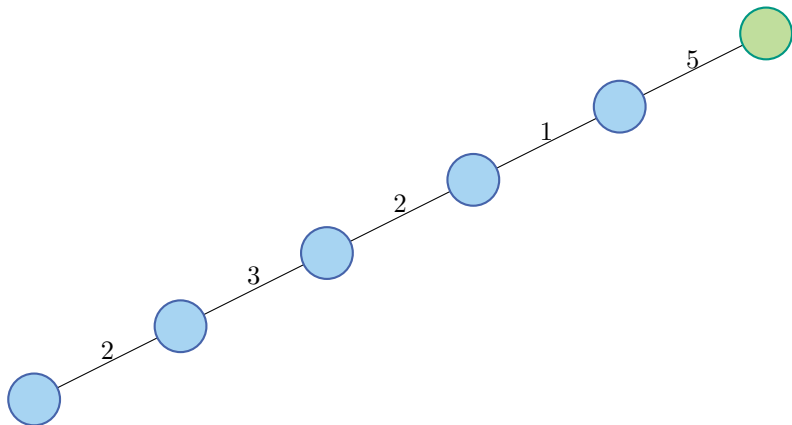
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



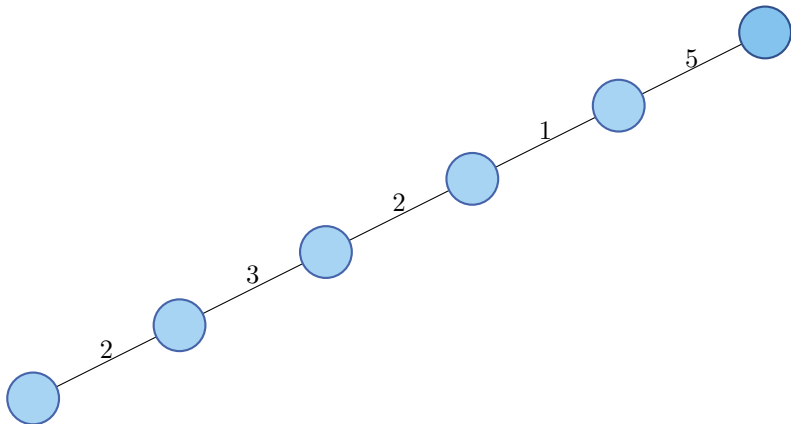
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



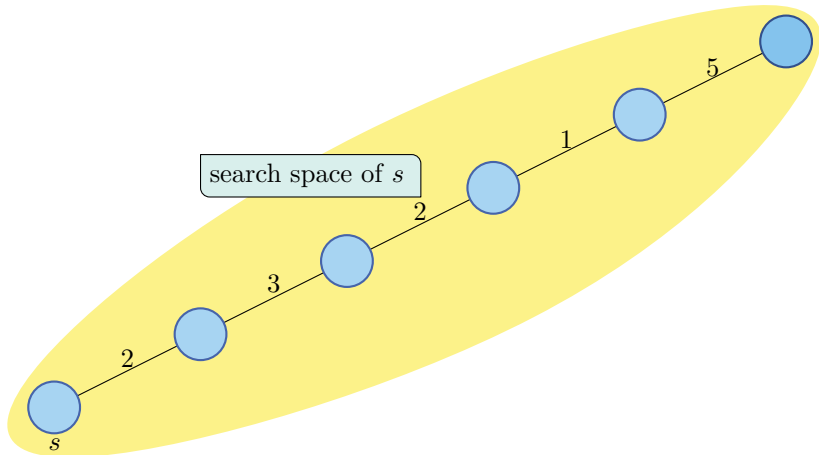
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad



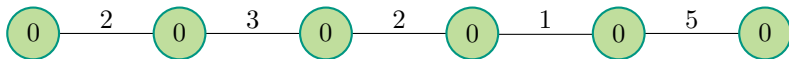
Linker Knoten kontrahieren erzeugt keine Shortcuts

Problemfall: Pfad

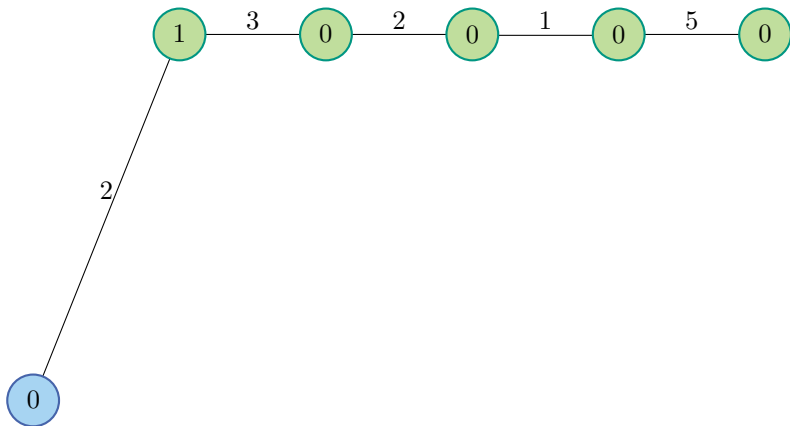


Suchraum von s ist der ganze Graph \rightarrow keine Beschleunigung

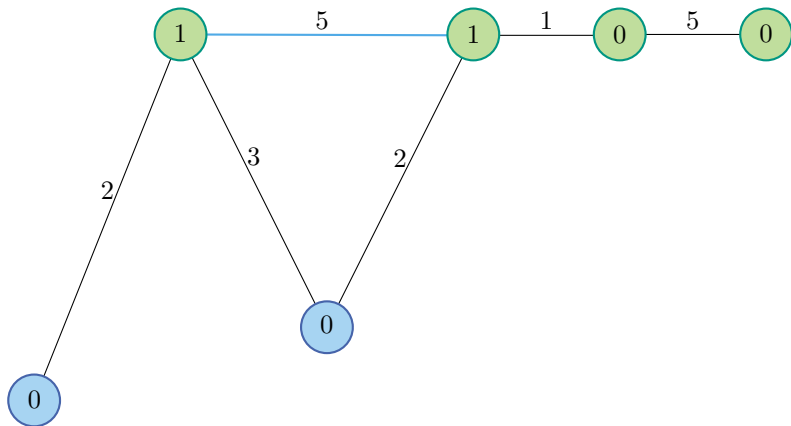
Problemfall: Pfad



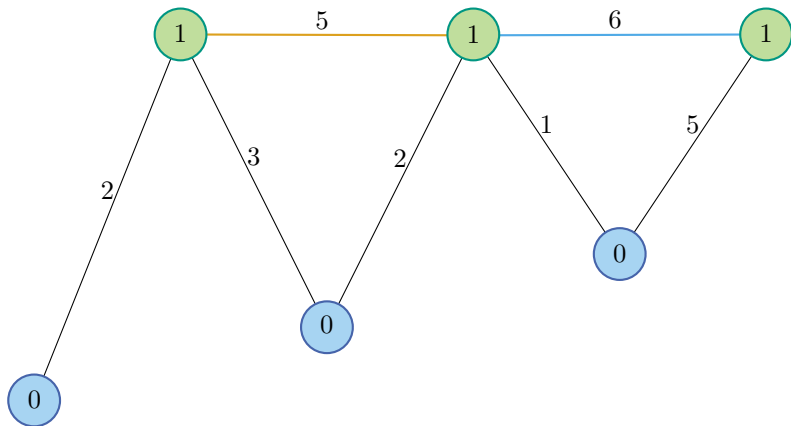
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert \rightarrow für alle Nachbarn y : $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$



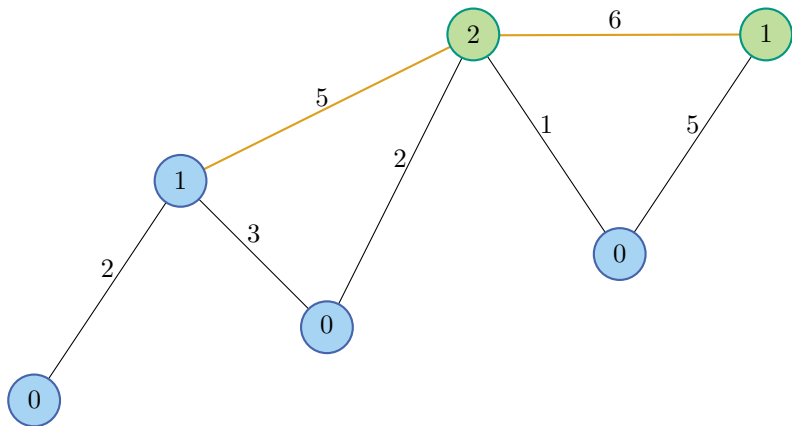
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert \rightarrow für alle Nachbarn y : $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$



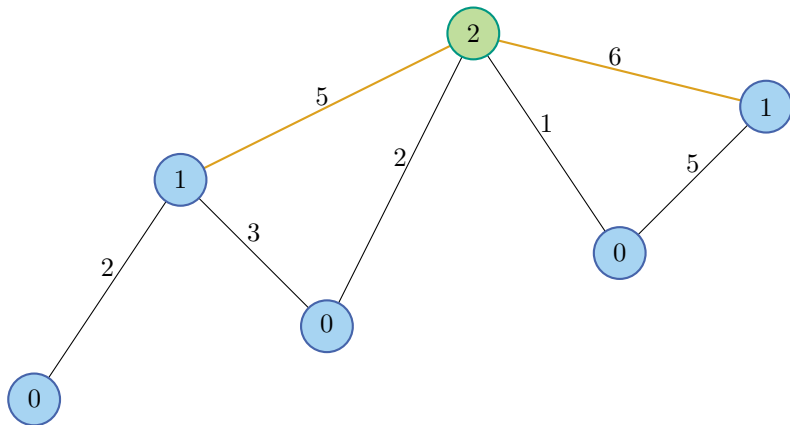
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert \rightarrow für alle Nachbarn y : $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$



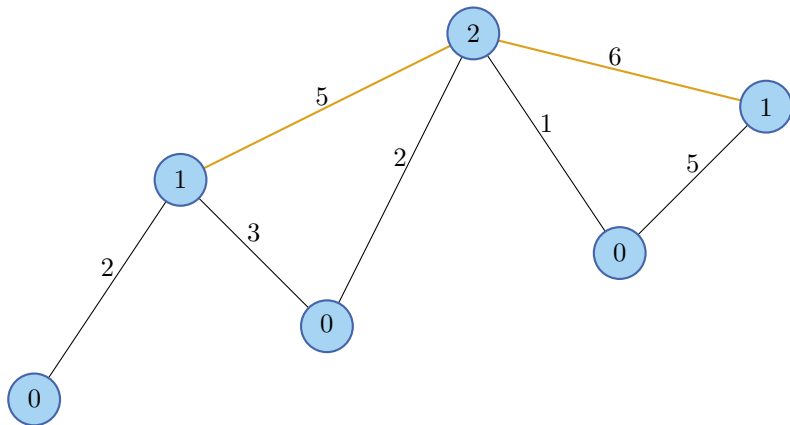
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert \rightarrow für alle Nachbarn y : $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$



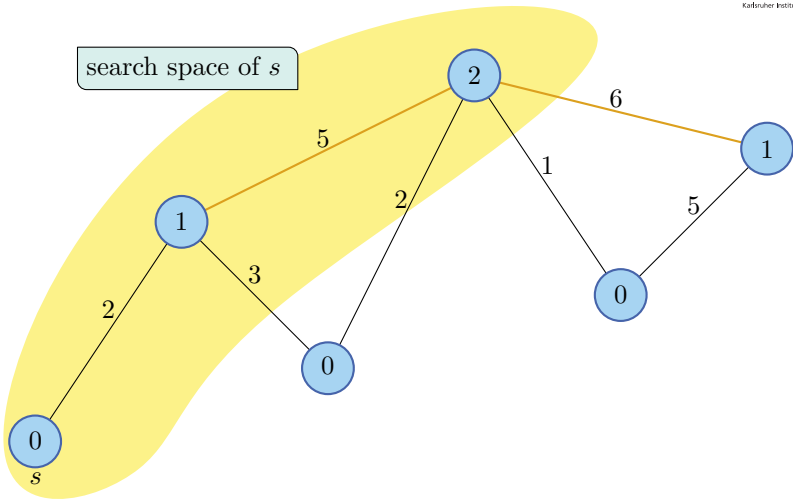
2-tes Kriterium: Das geschätzte Level $\ell(x)$ eines Knoten x kontrahiert \rightarrow für alle Nachbarn y : $\ell(y) \leftarrow \max\{\ell(y), \ell(x) + 1\}$



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



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



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

- 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.

- Graph mit Shortcuts heißt augmentierter Graph
- Eine Ordnung π ist eine Permutation der Knoten, so dass die Knoten in der Reihe $\pi(0), \pi(1) \dots \pi(n-1)$ kontrahiert werden.
- Die inverse Permutation π^{-1} heißt Rank. Der Rank entspricht der “Höhe” eines Knoten in der CH.