Vorlesungsmitschrift

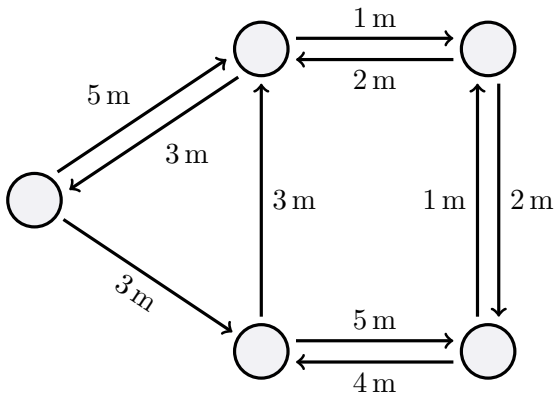
Algorithmen und Berechenbarkeit

Vorlesung 10

Letztes Update: 2018/01/28 - 01:07 Uhr

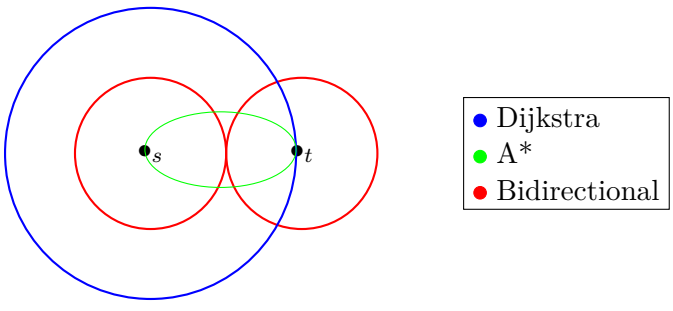
Routenplanung in Straßengraphen

Gegeben ist ein Graph $G(V, E, c)$, bei dem wie gehabt V die Menge der Knoten und E die Menge der Kanten darstellt. Im Gegensatz zum herkömmlichen Graph kommt hier noch die Reisezeit c hinzu. Ein Straßengraph kann wie im Folgenden dargestellt werden



Eine Anfrage beinhaltet einen Startknoten s und einen Targetknoten t wobei $s, t \in V$. Das Ziel ist der kürzeste/schnellste Pfad von s nach t. Für dieses Problem gibt es verschiedene Algorithmen, das Standardverfahren wäre *Dijkstra*. Bei einem Straßengraphen von Deutschland (Größenordnung: $|V| \approx 20$ Millionen, $|E| \approx 40$ Millionen) dauert eine Anfrage etwa 5 s, wenn der Dijkstra-Algorithmus ordentlich implementiert wurde.

Im Nachfolgenden ist skizzenhaft dargestellt, welche Ausdehnung ausgewählte Graphalgorithmen erreichen, die dieses Problem lösen können.



Contraction Hierarchies

Anreiz

Für einen relativ statischen Graphen (wie das Straßennetz) ist es nicht sinnvoll und vor allem ineffizient, für jede Suchanfrage eines kürzesten Pfads einen Dijkstra-Algorithmus laufen zu lassen. Daher möchte man einen Vorverarbeitungsschritt einfügen, der anfangs einige Minuten Zeit benötigt, hinterher aber jede Anfrage enorm (um den Faktor 100.000) beschleunigt.
Ein naiver Ansatz hierfür wäre, alle paarweisen kürzesten Distanzen vorzuberechnen. Theoretisch wäre die Antwortzeit auf eine Anfrage damit sehr schnell, praktisch scheitert es aber daran, die Menge der paarweise kürzesten Pfade im Speicher zu halten. Für $n = 20 \cdot 10^6$ Knoten ergäbe das etwa einen Platzverbrauch von $\approx 400 \cdot 10^{12}$.

Idee: Vorverarbeitung

- Der Graph wird um sogenannte *Shortcuts* erweitert. Das sind zusätzliche Kanten, die *kürzeste Wege* repräsentieren.
- Zusätzlich wird jedem Knoten ein *Level* zugeordnet.

Idee: Anfrage

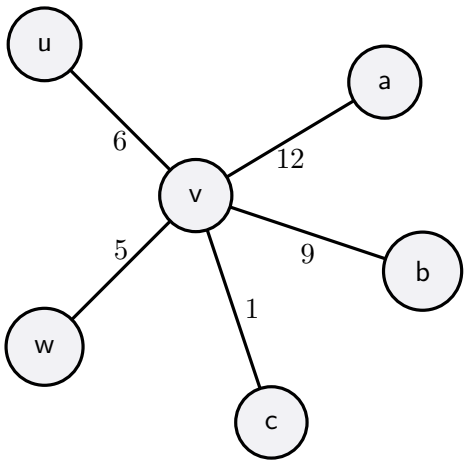
Eine Anfrage vom Startknoten s zum Targetknoten t folgt folgendem Muster:

1. Man lässt den Dijkstra-Algorithmus von s laufen und berücksichtigt nur Kanten, die zu höherleveligen Knoten führen.
2. Man lässt den Dijkstra-Algorithmus von t laufen und berücksichtigt nur Kanten, die zu höherleveligen Knoten führen.
3. Nun betrachtet man alle Knoten, die sowohl von s als auch von t besucht wurden. Die *Kürzeste-Weg-Distanz* d ist bestimmt durch das v, für das

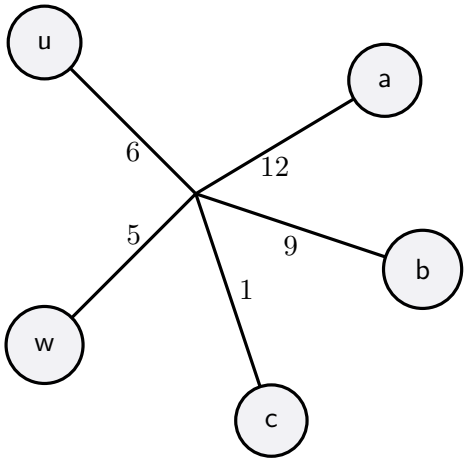
$$d_s(v) + d_t(v) = \text{minimal}$$

Vorverarbeitung: Knotenkontraktion

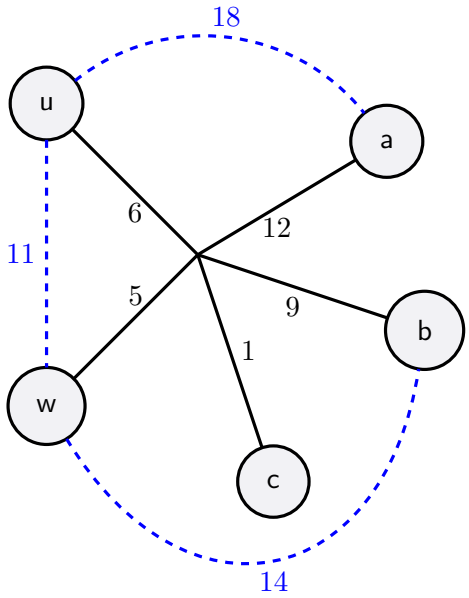
Die zentrale Operation der Contraction Hierarchies ist die Knotenkontraktion. Angenommen, ein kleiner Ausschnitt aus dem Straßengraphen hätte die folgende Struktur:



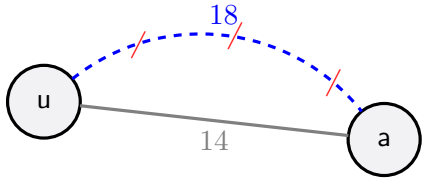
Nun möchte man den Knoten v entfernen, die Kanten zum Platz, an dem v sich befand, aber erhalten.



Um jetzt die Kürzeste-Weg-Distanz zwischen zwei Knoten zu erhalten, muss der Pfad über *ehemals* v gegangen werden. Die Idee ist nun, dass zwischen jedem Nachbarknotenpaar von v eine neue Kante mit den Kosten *Weg über* v eingefügt wird. Ausschnittsweise also



Nun kann aber der Fall auftreten, dass es schon einen Pfad zwischen einem Nachbarknotenpaar von v gibt, der kürzer ist als der neu eingefügte Pfad über v.



Diese falschen *Shortcuts* werden daher nicht eingefügt.

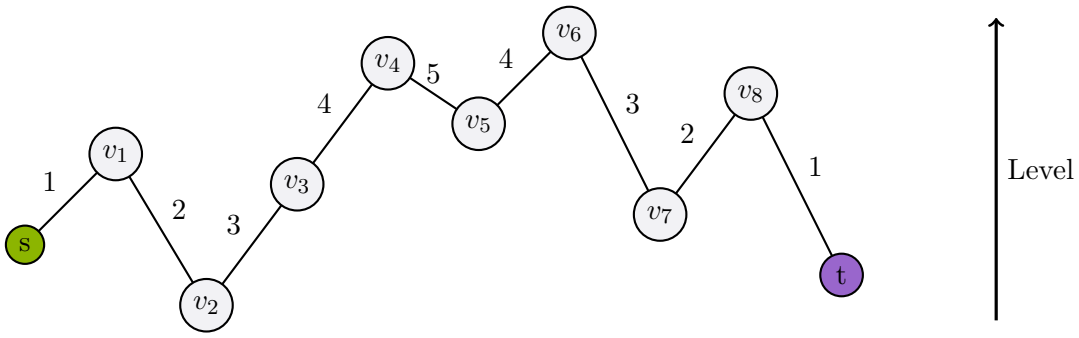
Das Verfahren lässt sich damit so zusammenfassen: **Man fügt Shortcuts zwischen zwei Knoten x und y ein, genau dann wenn ein *kürzester Weg* zwischen x und y existiert**.

```
counter = 0
while (|V| > 1)
    Wähle ein v ∈ V und kontrahiere es
    Füge ein Shortcut zu E hinzu
    level[v] = counter++

return level[] und alle kreierten Shortcuts
```

Vorverarbeitung: Korrektheit der Knotenkontraktion

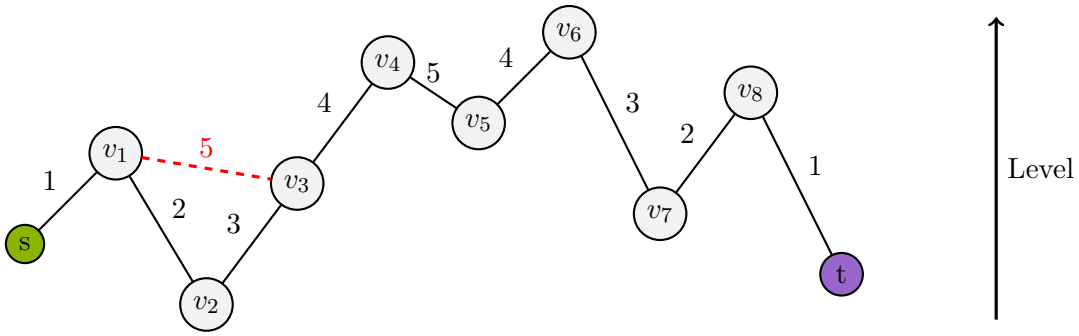
Man betrachtet einen kürzesten Pfad von s nach t, der die Knoten $v_1, v_2, \cdots, v_k$ passiert.



Die Verarbeitungsreihenfolge der Knoten entspricht der Ordnung der Knotenlevel:

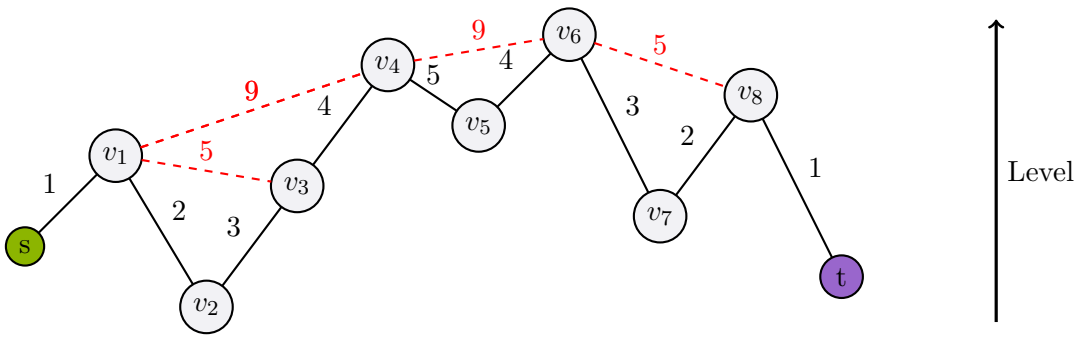
$$v_2 \to t \to s \to v_7 \to v_3 \to v_1 \to v_5 \to v_8 \to v_4 \to v_6$$

Als Erstes wird also der Knoten v_2 kontrahiert. Das ergibt



Danach werden die Knoten s und t kontrahiert. Da beide Randknoten sind, ändert sich jedoch nichts. Es folgt v_7, v_3 usw.

Am Ende ergibt das folgendes Bild:



□

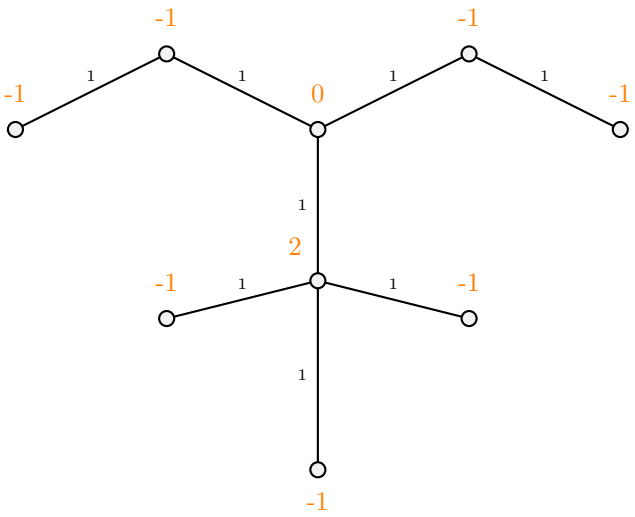
Anmerkungen zur Kontraktionsreihenfolge

Man möchte möglichst wenige Shortcuts neu einfügen. Es gibt daher verschiedene Ansätze, die Anzahl der neu eingefügten Shortcuts durch die Kontraktionsreihenfolge zu beeinflussen.

Ansatz 1: Edge-Difference

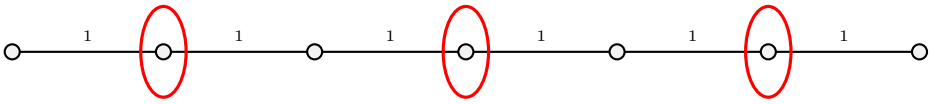
Man berechnet für jeden Knoten die sogenannte *Edge Difference*:

$$\underbrace{\text{Edge Difference}}_{\bullet} = \text{Anzahl neu einzufügender Shortcuts} - \text{Anzahl wegfallender Kanten}$$



Nun kontrahiert man immer einen Knoten mit der minimalen *Edge Difference*.

Ansatz 2: Jeden zweiten Knoten kontrahieren



Ansatz 3: Zufällige Reihenfolge

Man kann die Knoten auch in zufälliger Reihenfolge kontrahieren.

Ansatz 4: In der Praxis sehr gut

Man erlaubt Knoten, die nicht benachbart sind, dasselbe Level. Eine *Kontraktionsrunde* läuft dann nach folgendem Schema ab:

1. Man berechnet eine unabhängige Menge I (also eine Teilmenge von Knoten, die nicht zueinander benachbart sind) von G.
2. Nun berechnet man wieder die *Edge Differences* für alle $v \in I$.
3. Man kontrahiert alle $v \in I$ mit den 75 % kleinsten *Edge Differences*.
4. Alle Knoten in dieser Runde erhalten dasselbe Level.

Anmerkungenen und Notizen

- Für Deutschland benötigt die Vorverarbeitung etwa 4 Minuten.
- Am Anfang laufen viele kleine Dijkstra-Algorithmen, später wenige längere.
- Die Kontraktionsstrategien eliminieren in aller Regel zuerst kleine Dörfer und Dorfstraßen, danach Landstraßen, Bundesstraßen und erst zum Schluss Autobahnen. Die Strategie liefert also eine Art **Level of Detail** gleich mit.
- Der ermittelte kürzeste Pfad enthält Shortcuts, die sehr einfach *entpackt* werden können.
- Anfragen benötigen nach der Vorverarbeitung etwa 5 ms. Diese Zeit kann durch diverse Tricks auf < 1 ms reduziert werden.
- Es gibt noch einige andere sehr schnelle Algorithmen für das Ermitteln des kürzesten Pfads in einem Graphen mit Vorverarbeitung:
  - Transit Nodes benötigt etwa 10 µs für eine Anfrage
  - Hublabels benötigt etwa 10 µs für eine Anfrage
  - CAP benötigt etwa 1 ms für eine Anfrage
- Für den Öffentlichen Verkehr gibt es spezielle Anforderungen. Hier arbeitet man mit Transit Patterns.