Algorithmen und Datenstrukturen

C5. Kürzeste Pfade: Grundlagen

Gabriele Röger

Universität Basel

24. Mai 2023

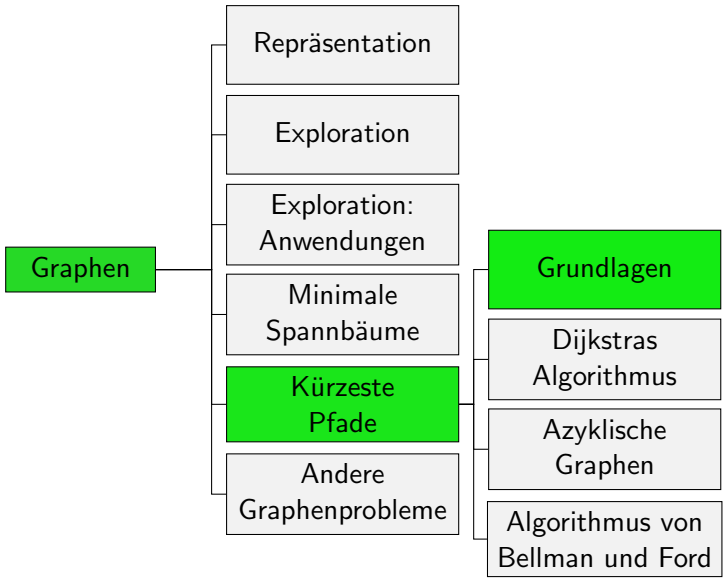
C5.1 Einführung

C5.2 Grundlagen

C5.3 Optimalitätskriterium und Generisches Verfahren

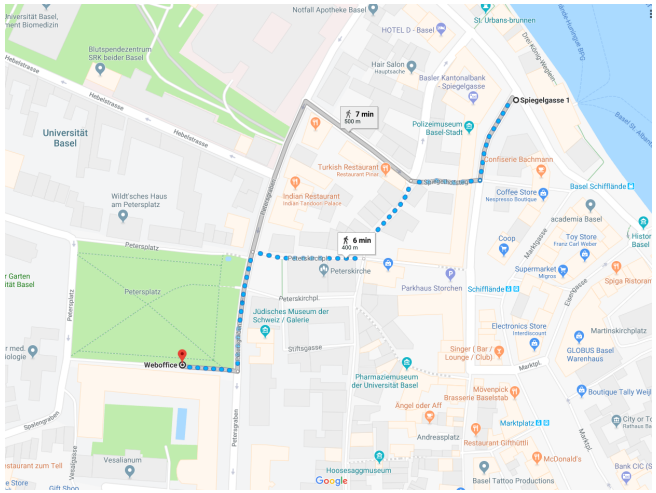
C5.4 Zusammenfassung

Graphen: Übersicht



C5.1 Einführung

Google Maps



Inhaltsabhängige Bildverzerrung (Seam Carving)

Anwendungen

- Routenplanung
- Pfadplanung in Computerspielen
- Roboternavigation
- Seam Carving
- Handlungsplanung
- Typesetting in TeX
- Routingprotokolle in Netzwerken (OSPF, BGP, RIP)
- Routing von Telekommunikationsnachrichten
- Verkehrsplanung
- Ausnutzen von Arbitrage-Möglichkeiten in Wechselkursen

Quelle (teilweise): Network Flows: Theory, Algorithms, and Applications,
R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, Prentice Hall, 1993

Varianten

Was interessiert uns?

- Single source: von einem Knoten s zu allen anderen Knoten
- Single sink: von allen Knoten zu einem Knoten t
- Source-sink: von Knoten s zu Knoten t
- All pairs: von jedem Knoten zu jedem anderen

Grapheigenschaften

- Beliebige / nicht-negative / euklidische Gewichte
- Beliebige / nicht-negative / keine Zyklen

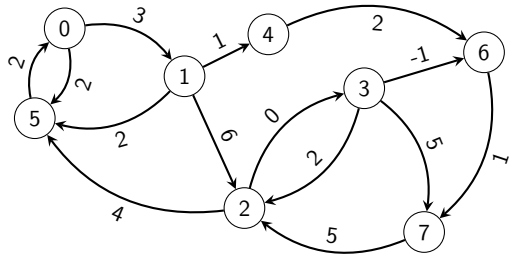
C5.2 Grundlagen

Gewichtete gerichtete Graphen

Die (high-level) Definition gewichteter Graphen bleibt gleich, wir betrachten jetzt aber gerichtete Graphen.

Gewichteter Graph

Bei einem (kanten-)gewichteten Graphen hat jede Kante $e \in E$ ein Gewicht (oder Kosten) $weight(e)$ aus den reellen Zahlen.



Erinnerung: Ein gerichteter Graph heisst auch Digraph.

API für gewichtete, gerichtete Kante

```
1  class DirectedEdge:
2    # Kante von n1 zu n2 mit Gewicht w
3    def __init__(n1: int, n2: int, w: float) -> None
4
5    # Gewicht der Kante
6    def weight() -> float
7
8    # Knoten, von dem Kante ausgeht
9    def from_node() -> int
10
11   # Knoten, zu dem die Kante führt
12   def to_node() -> int
```


API für gewichtete Digraphen

```
1  class EdgeWeightedDigraph:
2    # Graph mit no_nodes Knoten und keinen Kanten
3    def __init__(no_nodes: int) -> None
4
5    # Füge gewichtete Kante hinzu
6    def add_edge(e: DirectedEdge) -> None
7
8    # Anzahl der Knoten
9    def no_nodes() -> int
10
11   # Anzahl der Kanten
12   def no_edges() -> int
13
14   # Alle Kanten, die von n ausgehen
15   def adjacent_edges(n: int) -> Generator[DirectedEdge]
16
17   # Alle Kanten
18   def all_edges() -> Generator[DirectedEdge]
```

Kürzeste-Pfade-Problem

Kürzeste-Pfade-Problem mit einem Startknoten, SSSP

- Gegeben: Graph und Startknoten s
- Anfrage für Knoten v
  - Gibt es Pfad von s nach v?
  - Wenn ja, was ist der kürzeste Pfad?
- In kantengewichteten Graphen:
  Kürzester Pfad ist der mit dem geringstem Gewicht
  (= minimale Summe der Kantenkosten)

Engl. single-source shortest paths problem

API für Kürzeste-Pfade-Implementierungen

Die Algorithmen für kürzeste Pfade sollen folgendes Interface implementieren:

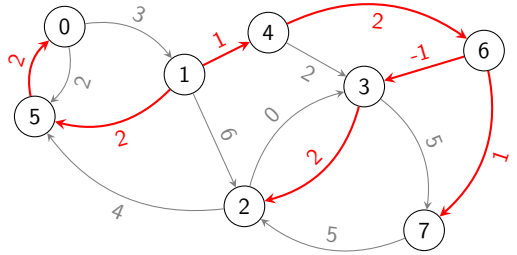
```
1  class ShortestPaths:
2    # Konstruktor mit Startknoten s
3    def __init__(graph: EdgeWeightedDigraph, s: int) -> None
4
5    # Abstand von s zu v; infinity, falls kein Pfad existiert
6    def dist_to(v: int) -> float
7
8    # Gibt es Pfad von s zu v?
9    def has_path_to(v: int) -> bool
10
11   # Pfad von s zu v; None, falls keiner vorhanden
12   def path_to(v: int) -> Generator[DirectedEdge]
```


Kürzeste-Pfade-Baum

Kürzeste-Pfade-Baum

Für einen kantengewichteten Digraphen G und Knoten s ist ein Kürzeste-Pfade-Baum ein Teilgraph, der

- einen gerichteten Baum mit Wurzel s bildet,
- alle von s aus erreichbaren Knoten enthält, und
- bei dem jeder Baumpfad ein kürzester Pfad in G ist.



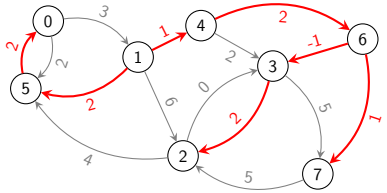
Kürzeste-Pfade-Baum: Repräsentation

Repräsentation: knotenindizierte Arrays

- `parent` mit Elternknotenreferenz
  None für nicht erreichbare und Startknoten
- `distance` mit Abstand vom Startknoten
  ∞ für nicht erreichbare Knoten

	0	1	2	3	4	5	6	7
parent	5	None	3	6	1	1	4	6

	0	1	2	3	4	5	6	7
distance	4	0	4	2	1	2	3	4



Was ist mit parallelen Kanten?

Extraktion der kürzesten Pfade

```
1    def path_to(self, node):
2        if self.distance[node] == float('inf'):
3            yield None
4        elif self.parent[node] is None:
5            yield node
6        else:
7            # output path from start to parent node
8            self.path_to(self.parent[node])
9            # finish with node
10           yield node
```

Diese Implementierung generiert eine Sequenz von Knoten. Was müssten wir ändern, um eine entsprechende Sequenz von Kanten zu generieren?

Kantenrelaxierung

Kantenrelaxierung für Kante (u, v)

- `distance[u]`: Länge des kürzesten bekannten Pfades zu u
- `distance[v]`: Länge des kürzesten bekannten Pfades zu v
- `parent[v]`: Vorgänger in letzter Kante des kürzesten bekannten Weges zu v
- Ermöglicht Kante (u, v) einen kürzeren Weg zu v (durch u)?
- Dann update `distance[v]` und `parent[v]`.

Illustration: Tafel

Kantenrelaxierung

```
1    def relax(self, edge):
2        u = edge.from_node()
3        v = edge.to_node()
4        if self.distance[v] > self.distance[u] + edge.weight():
5            self.parent[v] = u
6            self.distance[v] = self.distance[u] + edge.weight()
```


C5.3 Optimalitätskriterium und Generisches Verfahren

Optimalitätskriterium

Theorem

Sei G ein gewichteter Digraph ohne negative Zyklen. Array `distance[]` enthält die Kosten der kürzesten Pfade von s genau dann, wenn

1. $\texttt{distance[s]} = 0$
2. $\texttt{distance}[w] \leq \texttt{distance}[v] + \mathit{weight}(e)$ *für alle Kanten $e = (v, w)$, und*
3. *für alle Knoten v ist* $\texttt{distance}[v]$ *die Länge irgendeines Pfades von s zu v bzw. ∞, falls kein solcher Pfad existiert.*

Optimalitätskriterium (Forts.)

Beweis

„$\Rightarrow$“

Da der Graph keine Zyklen mit negativen Gesamtkosten enthält, kann kein Pfad von s zu s negative Kosten haben. Die Kosten des leeren Pfades sind damit optimal und `distance[s]` ist 0.

Betrachte beliebige Kante e von u nach v.

Der kürzeste Pfad von s nach u hat Kosten `distance[u]`. Erweitern wir diesen Pfad um Kante e, erhalten wir einen Pfad von s zu v mit Kosten $\texttt{distance[u]} + \mathit{weight}(e)$. Die Kosten eine kürzesten Pfades von s zu v können also nicht grösser sein und es gilt $\texttt{distance[v]} \leq \texttt{distance[u]} + \mathit{weight}(e)$. ...

Optimalitätskriterium (Forts.)

Beweis (Fortsetzung).

„$\Leftarrow$“

Für unerreichbare Knoten ist der Wert per Definition unendlich.

Betrachte beliebigen Knoten v und kürzesten Pfad $p = (v_0, \ldots, v_n)$ von s zu v, d.h. $v_0 = s$, $v_n = v$.
Sei e_i jeweils eine günstigste Kante von v_{i-1} zu v_i.
Da alle Ungleichungen erfüllt sind, gilt

$$\begin{aligned}\texttt{distance}[v_n] &\leq \texttt{distance}[v_{n-1}] + \mathit{weight}(e_n)\\ &\leq \texttt{distance}[v_{n-2}] + \mathit{weight}(e_{n-1}) + \mathit{weight}(e_n)\\ &\leq \ldots \leq \mathit{weight}(e_1) + \cdots + \mathit{weight}(e_n)\\ &= \text{ Kosten des optimalen Pfads}\end{aligned}$$

Wegen Punkt 3 ist $\texttt{distance}[v_n]$ auch nicht echt kleiner als die optimalen Pfadkosten. □

Generischer Algorithmus

Generischer Algorithmus für Startknoten s

- Initialisiere $\texttt{distance}[s] = 0$ und $\texttt{distance}[v] = \infty$ für alle anderen Knoten
- Solange das Optimalitätskriterium nicht erfüllt ist: Relaxiere eine beliebige Kante

Korrekt:

- Endliches `distance[v]` entspricht immer den Kosten eines Pfades von s zu v.
- Jede erfolgreiche Relaxierung reduziert `distance[v]` für ein v.
- Für jeden Knoten kann Distanz nur endlich oft reduziert werden.

C5.4 Zusammenfassung

Zusammenfassung

- single-source shortest paths: Berechne in einem kantengewichteten Digraphen kürzeste Pfade von einem gegebenen Knoten zu allen erreichbaren Knoten.
- Kantenrelaxierung: Ist für Kante (u,v) die beste bekannte Distanz zu v grösser als die zu u plus das Kantengewicht, dann update die Distanz zu v (mit Vorgänger u).
- Generischer Algorithmus
  - Basiert auf Kantenrelaxierung und Optimalitätskriterium
  - Ist in jeder Instanziierung für alle gewichteten Digraphen ohne negative Zyklen korrekt.
  - Konkrete Instanziierungen: nächstes Kapitel