# Datenstrukturen, Algorithmen und Programmierung 2

Amin Coja-Oghlan

**May 30, 2023**

Lehrstuhl Informatik 2
Fakultät für Informatik

# Kürzstes Pfade

## Worum geht es?

- wir lernen den Dijkstra-Algorithmus für kürzeste Pfade kennen
- mittels priority queues kann Dijkstra effizient implementiert werden
- wir lernen einen exakten Algorithmus für das Travelling Salesman-Problem kennen

## Kürzstes Pfade

### Das kürzeste-Pfade-Problem

- gegeben ist ein *gewichteter* Graph $G, c$
- das bedeutet, daß $c : E(G) \to \mathbb{Q}_{\geq 0}$ ["Kosten" oder "Länge" einer Kante]
- ferner sind zwei Knoten $s, t$ gegeben
- das Ziel ist, einen kürzesten Pfad von $s$ nach $t$ zu finden
- die Länge des Pfades $p = (v_0, \ldots, v_\ell)$ ist dabei definiert als

$$c(p) = \sum_{i=1}^{\ell} c(v_{i-1} v_i)$$

# Kürzstes Pfade

## Der Dijkstra-Algorithmus

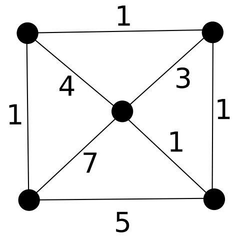
- der Dijkstra-Algorithmus basiert auf dem Paradigma der *dynamischen Programmierung*
- der Algorithmus baut aus Lösungen kleinerer Teilprobleme die Lösungen immer größerer Teilprobleme zusammen, bis schließlich das Gesamtproblem gelöst ist
- im Fall des kürzesten-Pfade-Problems liegt die Beobachtung zugrunde, daß ein kürzester Pfad von $s$ nach $t$ aus kürzesten Teilpfaden besteht

# Kürzstes Pfade

## Algorithmus Dijkstra

1. Für alle $v \in V(G)$ setze $\delta(v) = \infty$ und $p(v) = \emptyset$.
2. Setze $\delta(s) = 0$, $p(s) = s$, $S = \emptyset$ und $U = \{s\}$
3. Solange $U \neq \emptyset$
4.     finde $u \in U$ mit $\delta(u) = \min_{v \in U} \delta(v)$
5.     entferne $u$ aus $U$ und füge $u$ zu $S$ hinzu
6.     für alle $w \in \partial u \setminus S$
7.         füge $w$ zu $U$ hinzu
8.         falls $\delta(w) > \delta(u) + c(uw)$
9.             setze $\delta(w) = \delta(u) + c(uw)$ und $p(w) = u$
10. Gib $p, \delta$ aus

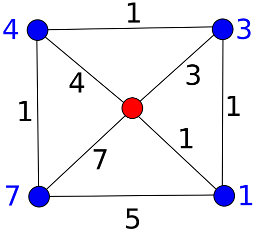
## Kürzstes Pfade




### Beispiel

- in jeder Iteration der Hauptschleife wird der Abstand eines Knotens von $s$ bestimmt
- die Abstände der Nachbarn werden ggf. verringert

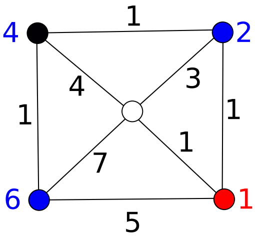
## Kürzstes Pfade




### Beispiel

- in jeder Iteration der Hauptschleife wird der Abstand eines Knotens von $s$ bestimmt
- die Abstände der Nachbarn werden ggf. verringert

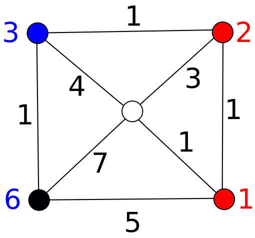
## Kürzstes Pfade




### Beispiel

- in jeder Iteration der Hauptschleife wird der Abstand eines Knotens von $s$ bestimmt
- die Abstände der Nachbarn werden ggf. verringert

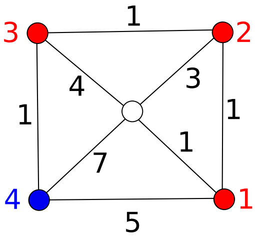
## Kürzstes Pfade




### Beispiel

- in jeder Iteration der Hauptschleife wird der Abstand eines Knotens von $s$ bestimmt
- die Abstände der Nachbarn werden ggf. verringert

## Kürzstes Pfade




### Beispiel

- in jeder Iteration der Hauptschleife wird der Abstand eines Knotens von $s$ bestimmt
- die Abstände der Nachbarn werden ggf. verringert

## Kürzstes Pfade

### Satz

Angenommen $G, c$ ist ein zusammenhängender gewichteter Graph, $s \in V(G)$ und $\delta, p$ ist die Ausgabe von Dijkstra.

- die Laufzeit beträgt $O(|V|^2)$
- für jeden Knoten $v \in V(G)$ ist $\delta(v)$ der gewichtete Abstand von $s$ nach $v$ und

$$v, p(v), \ldots, p^{\ell}(v) \text{ mit } \ell = \min_{j \geq 0} p^j(v) = s$$

  ist ein kürzester Pfad

# Kürzstes Pfade

## Beweis: Laufzeit

- die Hauptschleife wird $|V|$ mal durchlaufen
- die Berechnung von $u$ benötigt Zeit $O(|V|)$
- anschließend werden die $O(|V|)$ Nachbarn von $u$ bearbeitet

# Kürzstes Pfade

## Beweis: Korrektheit

- nach dem ersten Durchlauf der Hauptschleife gilt

$$U = \partial s \qquad S = \{s\}$$

$$\delta(s) = 0 \qquad \delta(v) = c(sv) \quad (v \in \partial s)$$

## Kürzstes Pfade

### Beweis: Korrektheit (fortgesetzt)

- wir beweisen, daß forthin die folgenden drei Bedingungen erfüllt bleiben:
  - **(i)** $U = \partial S \setminus S$
  - **(ii)** für alle $v \in S$ ist $\delta(v)$ der Abstand von $s$ nach $v$ und
    $$v, p(v), \ldots, p^{\ell}(v) \text{ mit } \ell = \min_{j \geq 0} p^j(v) = s$$
    ist ein entsprechender kürzester Pfad
  - **(iii)** für alle $v \in U$ ist $\delta(v)$ der Abstand von $s$ nach $v$ in $G[S \cup \{v\}]$ und
    $$v, p(v), \ldots, p^{\ell}(v) \text{ mit } \ell = \min_{j \geq 0} p^j(v) = s$$
    ein entsprechender Pfad.

# Kürzstes Pfade

## Beweis: Korrektheit (fortgesetzt)

- dazu führen wir Induktion nach der Anzahl der Iterationen der Hauptschleife
- für die erste Iteration ist nichts zu zeigen
- betrachte nun den nächsten Knoten $u$
- dieser erfüllt $\delta(u) = \min_{v \in U} \delta(v)$
- aus der Induktionsannahme und der Konstruktion in Schritten 5–10 folgt die Aussage (i) unmittelbar

# Kürzstes Pfade

## Beweis: Korrektheit (fortgesetzt)

- zu Aussage (ii): nach Induktion ist $P = u, p(u), \ldots, p^{\ell}(u)$ mit $\ell$ minimal, so daß $p^{\ell}(u) = s$, ein kürzester Pfad von $s$ nach $u$ in $G[S \cup \{u\}]$
- angenommen es gäbe in $G$ einen kürzeren Pfad $Q$ von $s$ nach $u$
- sei $z$ der erste Knoten von $Q$, der nicht in $S \cup \{u\}$ liegt
- dann gilt $\delta(z) \geq \delta(u)$
- aus der Wahl von $z$ und der Induktionsannahme folgt also

$$c(Q) \geq \delta(z) \geq \delta(u) = c(P)$$

- Widerspruch zur Annahme

# Kürzstes Pfade

## Beweis: Korrektheit (fortgesetzt)

- die Aussage (iii) folgt aus der Aussage (ii) und der Tatsache, daß für $w \in S$ ein kürzester Pfad von $s$ nach $w$ in $G$ auch ein kürzester Pfad von $s$ nach $w$ in $G[S]$ ist
- da für einen zusammenhängenden Graphen jeder Knoten in $U$ und in der Folge in $S$ eingefügt wird, folgt die Behauptung

# Kürzstes Pfade

## Implementierung mit priority queues

- in Anwendungen treten häufig *dünne* Graphen mit $o(|V(G)|^2)$ Kanten auf
- nicht selten ist sogar $|E(G)| = O(|V(G)|)$
- für dünne Graphen ist Dijkstra relativ langsam
- zeitkritisch ist die Berechnung des Minimums in Schritt 4

# Kürzstes Pfade

## Erinnerung: min priority queues

- wir haben priority queues bereits im Zusammenhang mit Heapsort kennengelernt
- Operationen: `Insert`, `ExtractMin`, `DecreaseKey`
- jede der Operationen hat Laufzeit $O(\log n)$
- *dies sind genau die Operationen, die wir für Dijkstra benötigen!*

## Kürzstes Pfade

### Korollar

Unter Verwendung einer min-priority-queue hat `Dijkstra` eine Laufzeit von

$$O(|E(G)| \log |V(G)|)$$

### Bemerkung

Eine noch bessere Laufzeit kann mit einer ausgefeilten Datenstruktur, den *Fibonacci-Heaps*, erzielt werden. Man kommt dann auf $O(|V(G)| \log |V(G)| + |E(G)|)$.

## Kürzstes Pfade

### Travelling salesman

- gegeben ist ein gewichteter *vollständiger* Graph $G, w$ auf $n$ Knoten
- *Ziel:* eine kürzeste Tour, die jeden Knoten genau einmal besucht
- eine Tour ist eine bijektive Abbildung $\sigma : [n] \to V(G)$
- die Knoten werden also in der Reihenfolge $\sigma(1), \sigma(2), \ldots, \sigma(n), \sigma(1)$ besucht
- die Länge einer Tour ist

$$w(\sigma) = w(\{\sigma(1), \sigma(n)\}) + \sum_{i=1}^{n-1} w(\{\sigma(i), \sigma(\{i+1\}).$$

- *keine Dreiecksungleichung!*

# Kürzstes Pfade

## Travelling salesman

- derzeit ist kein effizienter Algorithmus für TSP bekannt [NP-schwer]
- “naiver” Algorithmus: alle Permutationen durchprobieren…
- *Laufzeit:*

$$\Omega(n!) = \Omega((n/\mathrm{e})^n) = \Omega(n^{n+o(n)}).$$

- *polynomieller Platzbedarf!*

# Kürzstes Pfade

## Bellmann-Held-Karp-Algorithmus

- Prinzip dynamische Programmierung
- wähle einen beliebigen Startknoten $s \in V(G)$
- baue eine Tabelle auf mit einem Eintrag für jede Teilmenge $U \subseteq V(G) \setminus \{s\}$ und jeden Knoten $u \neq s$, $u \notin U$
- Eintrag $T(u, U)$ ist die Länge eines kürzesten Pfades, der in $s$ startet, alle Knoten aus $U$ besucht und in $u$ endet
- die Länge einer optimalen TSP-Tour kann leicht bestimmt werden, wenn die Tabelle bekannt ist

## Kürzstes Pfade

### Lemma

Für alle $s \in V(G)$, $\emptyset \neq U \subseteq V(G) \setminus \{s\}$, $u \in V(G) \setminus \{s\} \cup U$ gilt

$$T(u, U) = \min_{v \in U} T(v, U \setminus \{v\}) + w(\{u, v\}), \qquad T(u, \emptyset) = w(\{s, u\}).$$

### Beweis

Sei $v$ der letzte Knoten aus $U$, der auf einem kürzesten Pfad von $s$ durch alle Knoten aus $U$ nach $u$ liegt. Dann ist die Länge dieses Pfades genau $w(\{u, v\}) + T(v, U \setminus \{v\})$.

## Kürzstes Pfade

$\mathtt{BHK}(G, w)$

1. Wähle einen Startknoten $s$.
2. Für alle $u \in V(G) \setminus \{s\}$ setze $T(u, \emptyset) = w(\{s, u\})$.
3. Für $N = 1, \ldots, n-2$
4. für alle $u \in V(G) \setminus \{s\}$
5. für alle Teilmengen $U \subseteq V(G) \setminus \{u, s\}$ mit $|U| = N$
6. setze

$$T(u, U) = \min_{v \in U} T(v, U \setminus \{v\}) + w(\{u, v\}).$$

7. Gib $\min_{u \in V(G) \setminus \{s\}} T(u, V(G) \setminus \{u, s\}) + w(\{u, s\})$ aus.

## Kürzstes Pfade

**Satz**

BHK berechnet in Zeit $O(2^{n+o(n)})$ die Länge einer optimalen TSP-Tour.

**Beweis**

- Korrektheit folgt aus dem Lemma
- Laufzeit wird dominiert durch die Schleifen (3)–(6)
- dort wird über alle Teilmengen $U \subseteq V(G) \setminus \{s\}$ iteriert
- die Zahl dieser Teilmengen ist $O(2^n)$
- für jede Teilmenge wird wiederum über $O(n^2)$ Knoten $u, v$ iteriert

# Kürzstes Pfade

## Zusammenfassung

- Dijkstra berechnet kürzeste gewichtete Pfade
- wichtig ist dabei, daß die Gewichte *nicht negativ* sind! (Warum?)
- die “naive” Laufzeit ist quadratisch
- mit min priority queues läßt sich die Laufzeit deutlich verbessern
- der Algorithmus `BHK` für TSP