

# Algorithmen und Datenstrukturen

## Sommersemester 2021

Stefan Näher

Universität Trier

naeher@uni-trier.de

**Vorlesung 19**

23. Juni 2021

## Ein erster Algorithmus

1. **forall**  $v \in V$  **do**  $\text{DIST}[v] \leftarrow \infty$  **od**
2.  $\text{DIST}[s] \leftarrow 0$ ;
3. **while**  $\exists (u, v) \in E$  mit  $\text{DIST}[v] > \text{DIST}[u] + \text{cost}(u, v)$  **do**
4.      $\text{DIST}[v] \leftarrow \text{DIST}[u] + \text{cost}(u, v)$ ;
5. **od**

## Eigenschaften

- a) Es gilt stets  $\text{DIST}[v] \geq \text{dist}(s, v)$ .
- b) Wenn  $\text{DIST}[v] < \infty$ , dann existiert ein Pfad von  $s$  nach  $v$  mit Kosten  $\text{DIST}[v]$ .
- c) Die kürzesten Pfade bilden einen Baum  $T$  mit Wurzel  $s$ .
- d) Für jede Kante  $(v, w)$  auf einem kürzesten Pfad (Kante in  $T$ ) gilt:  
 $\text{dist}(s, w) = \text{dist}(s, v) + \text{cost}(v, w)$ , d.h.  $\Delta$ -Ungleichung mit Gleichheit.

## Verfeinerter Algorithmus

### Idee:

Verwende eine Kandidatenmenge  $U$  der Knoten  $u$ , aus denen Kanten ausgehen können, die die  $\Delta$ -Ungleich verletzen.

### Beobachtungen:

1. Am Anfang gilt  $U = \{ s \}$
2. Immer wenn  $\text{DIST}[v]$  für einen Knoten  $v$  vermindert wird (Zeile 4), muss  $v$  in die Menge  $U$  aufgenommen werden.
3. Die Hauptschleife wird solange ausgeführt, wie  $U \neq \emptyset$  gilt.

## Verfeinerter Algorithmus

1. **forall**  $v \in V$  **do**  $\text{DIST}[v] \leftarrow \infty$ ; **od**
2.  $\text{DIST}[s] \leftarrow 0$ ;
3.  $U = \{ s \}$ ;
4. **while**  $U \neq \emptyset$  **do**
5.     wähle und entferne einen beliebigen Knoten  $u$  aus  $U$ .
6.     **forall**  $v \in V$  mit  $(u, v) \in E$  **do**
7.          $d \leftarrow \text{DIST}[u] + \text{cost}(u, v)$ ;
8.         **if**  $d < \text{DIST}[v]$  **then**
9.              $\text{DIST}[v] \leftarrow d$ ;
10.          $U \leftarrow U \cup \{ v \}$ ;
11.     **fi**
12. **od**
13. **od**

## Lemma 2

*Falls  $G$  keine negativen Zyklen enthält, dann gilt folgendes:*

- a) Falls  $v \notin U$ , dann gilt für alle ausgehenden Kanten  $(v, w)$ :  
 $DIST[w] \leq DIST[v] + c((v, w))$  (dh.  $\Delta$ -Ungleichung erfüllt)*
- b) Sei  $v_0, \dots, v_k$  ein kürzester Pfad von  $s$  nach  $v$  (dh.  $v_0 = s$ ,  $v_k = v$ ).  
Falls nun  $DIST[v] > dist(s, v)$ , dann existiert ein  $i$  ( $0 \leq i \leq k - 1$ ) mit  
 $DIST[v_i] = dist(s, v_i)$  und  $v_i \in U$  z.B.  $s = 0$*
- c) Es existiert immer ein  $u \in U$  mit  $DIST[u] = dist(s, u)$*
- d) Wenn in Zeile 7 des Algorithmus 2 stets ein  $u \in U$  mit  $DIST[u] = dist(s, u)$  gewählt wird ("perfekte Wahl"), dann wird die while-Schleife für jeden Knoten höchstes einmal ausgeführt.*

a) Induktion über die Schleifendurchläufe (while):

$i = 0$ : Vor dem ersten Lauf  $DIST[s] = 0$ ,  $DIST[v] = \infty$  für  $v \neq s$  und  $U = \{s\}$

$i \rightarrow i + 1$ : Betrachte ein beliebiges  $v \notin U$  nach der  $(i + 1)$ ten Ausführung.

Fall 1:  $v \notin U$  vor  $(i + 1)$ ten Ausführung. Nach I.A. galt:

$DIST[w] \leq DIST[v] + c((v, w))$  für alle ausgehende Kanten  $(v, w)$ .

$DIST[v]$  wurde im  $(i + 1)$ ten Lauf nicht verändert (da  $v \notin U$  danach) und die  $DIST$ -Werte aller Nachbarn  $w$  von  $v$  wurden eventuell vermindert

$\Rightarrow DIST[w] \leq DIST[v] + c((v, w))$  für alle ausgehenden Kanten (nach der  $(i + 1)$ ten Ausführung)

Fall 2:  $v \in U$  vor  $(i + 1)$ Ausführung

$\Rightarrow v$  wurde in Zeile 7 ausgewählt (dh.  $u = v$ )

$\Rightarrow$  Die innere Schleife stellt die  $\Delta$ -Ungleichung für alle ausgehenden Kanten her.

b) Sei  $s = v_0, \dots, v_k = v$  ein kürzester Pfad von  $s$  nach  $v$  mit

$$DIST[v_k] > dist(s, v_k) .$$

Sei  $i$  maximal mit  $DIST[v_i] = dist(s, v_i) \Rightarrow 0 \leq i < k$

Z.Z.  $v_i \in U$

**Widerspruchsbeweis:**

Annahme:  $v_i \notin U$

Teil a)  $\Rightarrow DIST[v_{i+1}] \leq DIST[v_i] + c((v_i, v_{i+1}))$

Außerdem gilt:  $dist(s, v_{i+1}) = dist(s, v_i) + c((v_i, v_{i+1}))$  (auf kürzesten Pfaden ist die  $\Delta$ -Ungleichung mit Gleichheit erfüllt)

Dann folgt:

$$\begin{aligned} dist(s, v_{i+1}) &= dist(s, v_i) + c((v_i, v_{i+1})) \\ &= DIST[v_i] + c((v_i, v_{i+1})) \\ &\geq DIST[v_{i+1}] \text{ (da } v_i \notin U) \end{aligned}$$

Zusammen:  $dist(s, v_{i+1}) \geq DIST[v_{i+1}]$

$\Rightarrow dist(s, v_{i+1}) = DIST[v_{i+1}] \not\Leftarrow$  Widerspruch zur maximalen Wahl von  $i$  da  $DIST$ -Werte nie zu klein werden.

c) Sei  $v \in U$  beliebig.

Fall 1:  $DIST[v] = dist(s, v) \Rightarrow$  fertig.

Fall 2:  $DIST[v] > dist(s, v)$

Betrachte einen kürzesten Pfad von  $s$  nach  $v$ . Dann existiert auf diesem Pfad (nach Teil b)) ein  $v_i \in U$  mit  $DIST[v_i] = dist(s, v_i)$ .

d) Falls in Zeile 7 des Algorithmus 2 immer eine perfekte Wahl getroffen wird, dann kann  $u$  kein zweites Mal in  $U$  eingefügt werden.

Perfekte Wahl

→ Gesamtaufwand:  $\mathcal{O}\left(\underbrace{\sum_{v \in V} (1 + outdeg(v))}_{n+m} + \underbrace{\text{Verwaltung der Menge } \mathcal{U}}_?\right)$

$n = \# \text{Knoten}$

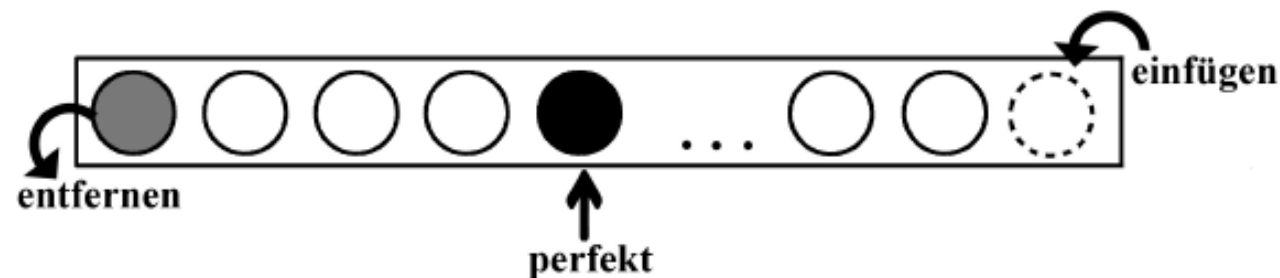
$m = \# \text{Kanten}$

□



Bei einem beliebigen Graph sind Zyklen möglich. Da auch die Kostenfunktion beliebig ist und somit negative Kosten auftreten können, können negative Zyklen entstehen.

Idee: Realisiere  $U$  als Schlange (FIFO)



Beinhaltet der Graph keine negativen Zyklen, so existiert nach Lemma 2 mindestens ein perfekter Knoten ( $DIST[v] = dist(s, v)$ ) in der Schlange.

⇒ Zwischen zwei Entnahmen des selben Knoten  $u$  wurde mindestens ein perfekter Knoten entfernt. Dieser kann nicht wieder in die Schlange eingefügt werden.

⇒ Jeder Knoten wird maximal  $n$ -mal entfernt. Spätestens danach gilt  $U = \emptyset$ .

### Beobachtung:

Wird ein Knoten öfter ( $> n$ ) entfernt, dann muss nach Lemma 2 ein negativer Zyklus existieren.

→ Algorithmus von Bellman/Ford

## Der Algorithmus von Bellman/Ford

1. Die Menge  $U$  wird als Schlange  $Q$  realisiert.
2. Verwende einen Bitvektor  $\text{in\_}Q$   
 $\text{in\_}Q[v] = \text{true}$  genau dann, wenn  $v \in Q$ .
4. Verwende ein Feld von Zählern  $\text{count}$   
 $\text{count}[v] = \text{Anzahl der Entnahmen des Knotens } v \text{ aus } Q$

## Der Algorithmus von Bellman/Ford

1. **forall**  $v \in V$  **do**
2.     $\text{DIST}[v] \leftarrow \infty$ ;
3.     $\text{count}[v] \leftarrow 0$ ;
4.     $\text{in\_Q}[v] = \text{false}$ ;
5. **od**
6.  $\text{DIST}[s] \leftarrow 0$ ;
7.  $Q.\text{append}(s)$ ;
8.  $\text{in\_Q}[s] \leftarrow \text{true}$ ;

```

9. while  $\neg Q.empty()$  do
10.    $u \leftarrow Q.pop()$ ;
11.   if  $++count[u] \geq n$  then ERROR: Negative Cycle fi
12.   forall  $v \in V$  mit  $(u, v) \in E$  do
13.      $d \leftarrow DIST[u] + cost(u, v)$ ;
14.     if  $d < DIST[v]$  then
15.        $DIST[v] \leftarrow d$ ;
16.       if  $\neg in\_Q[v]$  then
17.          $Q.append(v)$ ;
18.          $in\_Q[v] \leftarrow true$ ;
19.       fi
20.     fi
21.   od
22. od

```

## Laufzeit des Bellman/Ford-Algorithmus

Die Hauptschleife wird für jede Knoten  $u$  höchstens  $n$  mal ausgeführt.

Damit ergibt sich eine Gesamtlaufzeit von

$$\mathcal{O} \left( n \cdot \sum_{v \in V} (1 + \text{outdeg}(v)) \right) = \mathcal{O} (n \cdot (n + m)) = \mathcal{O}(n^2 + nm)$$

In zusammenhängenden Graphen gilt  $m \geq n - 1$  und damit ist die Laufzeit  $\mathcal{O}(nm)$ .

## Zusammenfassung

### Satz 1:

Das Single-Source Shortest Paths Problem kann für einen Graphen mit  $n$  Knoten und  $m$  Kanten in Zeit  $\mathcal{O}(n \cdot m)$  gelöst werden.

### Korollar:

Man kann in Zeit  $\mathcal{O}(n \cdot m)$  testen, ob in einem kürzesten Wege Netzwerk ein negativer Zyklus existiert.

**und auch einen negativen Zyklus berechnen !**

**Spezialfälle** (ohne negative Zyklen)

## 1. Azyklische Netzwerke

Es existieren überhaupt keine Kreise.

—→ **Topologisches Sortieren**

*Zeile 5:* wähle  $u \in U$  mit  $\text{topnum}[u]$  minimal

*Laufzeit:*  $\mathcal{O}(n + m)$

## 2. Nicht-negative Netzwerke

$\text{cost}(e) \geq 0$  für alle  $e \in E$ .

—→ **Algorithmus von Dijkstra**

*Zeile 5:* wähle  $u \in U$  mit  $\text{DIST}[u]$  minimal

*Laufzeit:*  $\mathcal{O}(n \log n + m)$