

# Algorithmen und Datenstrukturen 2

## Lerneinheit 7: Kürzeste Pfade in Graphen

Prof. Dr. Christoph Karg

Studiengang Informatik  
Hochschule Aalen



Sommersemester 2016



# Einleitung

Diese Lerneinheit beschäftigt sich mit der Berechnung von kürzesten Wegen in gewichteten Graphen.

Sie gliedert sich in folgende Abschnitte:

- Single Source Shortest Paths
  - ▷ Algorithmus von Bellman und Ford
  - ▷ Algorithmus von Dijkstra
- All-Pairs Shortest Paths
  - ▷ Algorithmus von Floyd & Warshall

# Kürzeste Pfade in gewichteten Graphen

Gegeben ist ein Graph  $G = (V, E)$  mit einer Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ .

- Das **Gewicht des Pfades**  $p = \langle v_0, v_1, \dots, v_k \rangle$  ist definiert als

$$w(p) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

- $u \overset{p}{\rightsquigarrow} v$  bezeichnet, dass  $p$  ein Pfad von  $u$  nach  $v$  ist
- Das **Gewicht eines kürzesten Pfades** von  $u$  nach  $v$  ist

$$\delta(u, v) = \begin{cases} \min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\} & \text{falls ein Pfad von } u \text{ nach } v \text{ existiert} \\ \infty & \text{sonst} \end{cases}$$

# Optimale Teilstruktur von kürzesten Pfaden

**Lemma 1.** Betrachte einen gerichteten Graphen  $G = (V, E)$  mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $p = \langle v_1, v_2, \dots, v_k \rangle$  ein Pfad von  $v_1$  nach  $v_k$ . Sei  $p_{ij} = \langle v_i, \dots, v_j \rangle$  ein Teilpfad von  $p$ , wobei  $1 \leq i \leq j \leq k$ . Dann ist  $p_{ij}$  ein kürzester Pfad von  $v_i$  nach  $v_j$ .

**Beweis.** Der Pfad  $p$  lässt sich zerlegen in die Pfade

$$v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p_{ij}} v_j \xrightarrow{p_{jk}} v_k$$

Es gilt:  $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$ . Angenommen es gibt einen Pfad  $p'_{ij}$  mit  $w(p'_{ij}) < w(p_{ij})$ . Dann hat der Pfad

$$v_1 \xrightarrow{p_{1i}} v_i \xrightarrow{p'_{ij}} v_j \xrightarrow{p_{jk}} v_k$$

ein geringeres Gewicht als der Pfad  $p$ . Widerspruch!

# Algorithmische Problemstellungen

Betrachte einen Graphen  $G = (V, E)$

- **Single Source Shortest Paths Problem:** Finde für einen Startknoten  $s$  die kürzesten Pfade zu jedem Knoten  $v \in V$
- **Single Destination Shortest Paths Problem:** Finde für jeden Knoten  $v \in V$  einen kürzesten Pfad zu einem Zielknoten  $t$
- **Single Pair Shortest Path Problem:** Finde einen kürzesten Pfad von einem Startknoten  $s$  zu einem Zielknoten  $t$
- **All Pairs Shortest Paths:** Finde für jedes Knotenpaar  $u$  und  $v$  einen kürzesten Pfad von  $u$  nach  $v$

# Kanten mit negativem Gewicht

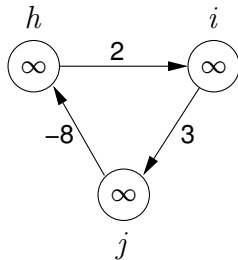
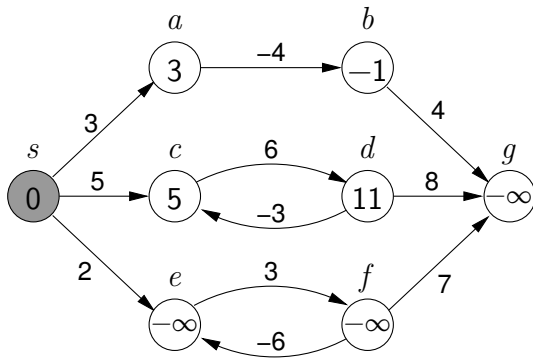
**Gegeben:** Graph  $G = (V, E)$ , der Kanten mit negativen Gewichten enthält

**Frage:** Wie wirken sich diese “negativen” Kanten auf die Berechnung der kürzesten Pfade aus?

**Antwort:**

- Falls in  $G$  vom Startknoten  $s$  aus keine negativen Zyklen erreichbar sind, dann ist für jeden Knoten  $v$  der kürzeste Pfad  $\delta(s, v)$  wohldefiniert
- Andernfalls kann ein Pfad von  $s$  zu einem Knoten  $v$ , der über einen negativen Zyklus führt, kein kürzester Pfad sein. In einem solchen Fall ist  $\delta(s, v) = -\infty$

# Kanten mit negativem Gewicht (Beispiel)



# Arbeitsweise der Single Source Algorithmen

**Annahme:** Die Suche beginnt beim Knoten  $s$

**Berechnete Informationen:**

- $d[v] \rightsquigarrow$  Länge des bisher gefundenen kürzesten Pfads von  $s$  nach  $v$ . Falls  $d[v] = \infty$ , dann wurde (noch) kein Pfad gefunden
- $\pi[v] \rightsquigarrow$  Vorgänger auf dem kürzesten Pfad von  $s$  nach  $v$ . Falls  $\pi[v] = \text{NIL}$ , dann wurde (noch) kein Pfad gefunden

**Bemerkungen**

- Während des Ablauf des Algorithmus enthalten obige Werte Informationen über die aktuell kürzesten Pfade
- Zur Berechnung einer Verbesserung kommt die Relaxation Technik zum Einsatz



# Algorithmus INITIALIZESINGLESOURCE

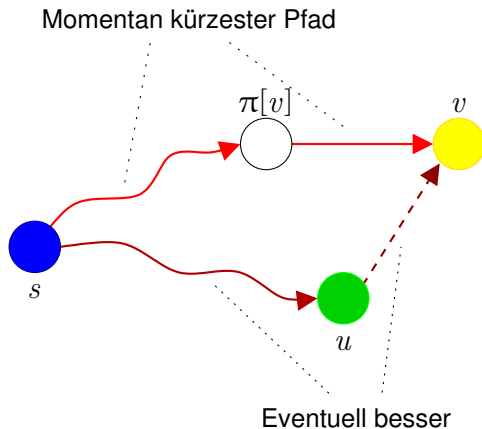
INITIALIZESINGLESOURCE( $G, s$ )

```
1  for jeden Knoten  $v \in V$  do  
2       $d[v] := \infty$   
3       $\pi[v] := \text{NIL}$   
4       $d[s] := 0$ 
```

## Bemerkungen

- Diese Funktion initialisiert die  $d$ - und  $\pi$ -Tabellen
- Der Werte  $d[s] = 0$  und  $\pi[s] = \text{NIL}$  sind korrekt und werden nicht mehr verändert

# Relaxation



Verbesserung, falls  $d[u] + w(u, v) < d[v]$

# Algorithmus RELAX

RELAX( $u, v, w$ )

```
1 if  $d[v] > d[u] + w(u, v)$  then  
2    $d[v] := d[u] + w(u, v)$   
3    $\pi[v] := u$   
4   return true  
5 else  
6   return false
```

## Bemerkungen

- Das Relaxing einer Kante  $(u, v)$  besteht darin, zu überprüfen, ob man den kürzesten Pfad von  $s$  nach  $v$  dadurch verbessern kann, indem man über die Kante  $(u, v)$  läuft
- Anhand des Rückgabewertes kann man ermitteln, ob eine Relaxation durchgeführt wurde

# Eigenschaft von kürzesten Pfaden

## **Eigenschaft 1.** (Dreiecksungleichung)

Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Für alle Kanten  $(u, v) \in E$  gilt  $\delta(s, v) \leq \delta(s, u) + w(u, v)$ .

**Beweis.** Angenommen es gibt einen kürzesten Pfad von  $s$  nach  $v$ . Dann ist das Gewicht von  $p$  kleiner-gleich dem Gewicht von jedem Pfad von  $s$  nach  $v$ , insbesondere von einem Pfad von  $s$  nach  $u$ , der dann über die Kante  $(u, v)$  nach  $v$  führt.

Wenn es keinen Pfad von  $s$  nach  $v$  gibt, dann ist  $\delta(s, v) = \infty$ . In diesem Fall gibt es keinen Pfad von  $s$  nach  $u$ . Somit gilt:

$$\infty = \delta(s, v) \leq \delta(s, u) + w(u, v) = \infty$$

# Eigenschaften der Relaxation

## **Eigenschaft 2.** (Obere Schranke)

Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Angenommen, die  $d$ - und  $\pi$ -Tabellen wurden mittels der Funktion `INITIALIZESINGLESOURCE( $G, s$ )` initialisiert.

Dann gilt:

- $d[v] \geq \delta(s, v)$  für alle Knoten  $v \in V$ . Diese Eigenschaft bleibt erhalten unabhängig von der Anzahl der Relaxationsschritte auf Kanten in  $G$ .
- Erreicht  $d[v]$  die untere Schranke  $\delta(s, v)$ , dann wird der Wert von  $d[v]$  nicht mehr verändert.

# Eigenschaften der Relaxation (Forts.)

**Beweis** mittels Induktion über die Anzahl der Relaxationsschritte.

*Induktionsanfang:* Nach der Initialisierung gilt:  $d[v] = \infty \geq \delta(s, v)$  für alle  $v \in V$ . Insbesondere gilt  $d[s] = 0 \geq \delta(s, s)$ .

*Induktionsschritt:* Laut Induktionsannahme gilt  $d[v] \geq \delta(s, v)$  für alle  $x \in V$ . Betrachte die Relaxation der Kante  $(u, v)$ . Der einzige  $d$ -Wert, der sich ändert ist,  $d[v]$ .

# Eigenschaften der Relaxation (Forts.)

Bei einer Änderung gilt:

$$\begin{aligned}d[v] &= d[u] + w(u, v) \\&\geq \delta(s, u) + w(u, v) \quad (\text{laut Induktionsannahme}) \\&\geq \delta(s, v) \quad (\text{Dreiecksungleichung})\end{aligned}$$

Falls  $d[v] = \delta(s, v)$ , dann kann sich dieser Wert nicht mehr ändern, denn

- es muss  $d[v] \geq \delta(s, v)$  gelten, und
- während der Relaxation werden die  $d$ -Werte nicht vergrößert.

# Eigenschaften der Relaxation (Forts.)

## **Eigenschaft 3.** (Kein Pfad)

Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Angenommen, es führt kein Pfad von  $s$  zu einem Knoten  $v \in V$ .

Nach der Initialisierung der  $d$ - und  $\pi$ -Tabellen gilt  $d[v] = \delta(s, v) = \infty$ . Dieser Wert wird durch die Relaxation nicht mehr verändert.

**Beweis.** Direkte Konsequenz der Obere Schranke Eigenschaft.



# Eigenschaften der Relaxation (Forts.)

## **Eigenschaft 4.** (Konvergenz)

Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Sei  $s \rightsquigarrow u \rightarrow v$  ein kürzester Pfad von  $s$  nach  $v$  in  $G$ . Angenommen, die  $d$ - und  $\pi$ -Tabellen wurden initialisiert und es wurde  $\text{RELAX}(u, v, w)$  für die Kante  $(u, v)$  aufgerufen. Falls vor dem Aufruf  $d[u] = \delta(s, u)$  war, dann gilt nach dem Aufruf  $d[v] = \delta(s, v)$ .

**Beobachtung:** Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $(u, v) \in E$  eine beliebige Kante. Unmittelbar nach der Relaxation mittels  $\text{RELAX}(u, v, w)$  gilt  $d[v] \leq d[u] + w(u, v)$ .

# Eigenschaften der Relaxation (Forts.)

**Beweis.** Obere Schranke Eigenschaft: Nimmt  $d[u]$  den Wert  $\delta(s, u)$  an, dann wird dieser Wert nicht mehr verändert.

Nach Relaxation der Kante  $(u, v)$  gilt:

$$\begin{aligned}d[v] &\leq d[u] + w(u, v) \quad (\text{obige Beobachtung}) \\&= \delta(s, u) + w(u, v) \\&= \delta(s, v) \quad (\text{Lemma 1})\end{aligned}$$

Obere Schranke Eigenschaft:  $d[v] \geq \delta(s, v)$ .

Somit  $d[v] = \delta(s, v)$ .

# Eigenschaften der Relaxation (Forts.)

## **Eigenschaft 5.** (Pfad Relaxation)

Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Betrachte einen beliebigen kürzesten Pfad  $p = \langle v_0, v_1, \dots, v_k \rangle$  von  $s = v_0$  nach  $v_k$ . Angenommen, die  $d$ - und  $\pi$ -Tabellen wurden initialisiert und es wurde eine Folge von Relaxationen durchgeführt.

Falls die Kanten  $(v_0, v_1), (v_1, v_2), \dots, (v_{k-1}, v_k)$  in genau dieser Reihenfolge relaxiert wurden, dann gilt danach  $d[v_k] = \delta(s, v_k)$ . Dieser Wert wird nicht mehr verändert.

Die Relaxation anderer Kanten vor, mitten und nach der Relaxation der obigen Kantenfolge wirkt sich nicht auf den Wert von  $d[v_k]$  aus.

# Eigenschaften der Relaxation (Forts.)

**Beweis** mittels Induktion über die Länge des Pfades  $p$ .

*Induktionsbehauptung:* Nach der Relaxation der Kante  $(v_{i-1}, v_i)$  gilt  $d[v_i] = \delta(s, v_i)$ .

*Induktionsanfang:*  $i = 0$ .  $d[s] = 0 = \delta(s, s) = \delta(v_0, v_0)$  ✓

*Induktionsschritt:*  $i - 1 \rightsquigarrow i$ . Angenommen,  $d[v_{i-1}] = \delta(s, v_{i-1})$ . Betrachte den Aufruf  $\text{RELAX}(v_{i-1}, v_i, w)$ . Wegen der Konvergenzeigenschaft gilt nach dem Aufruf, das  $d[v_i] = \delta(s, v_i)$ .

# Bellman-Ford Algorithmus

- Erfunden von Richard Bellman und Lester Ford in den 1960er Jahren
- Berechnung von kürzesten Pfaden ausgehend von einem Startknoten  $s$
- Für Graphen mit negativen Kantengewichten geeignet
- Zyklen mit negativem Gewicht werden erkannt
- Einsatz von Relaxation zur schrittweisen Verkleinerung der Einträge in der  $d$ -Tabelle
- Der Algorithmus wird unter anderem in Distance Vector Routing Protokollen (z.B. RIP) eingesetzt
- Laufzeit  $O(\|V\| \cdot \|E\|)$

# Bellman-Ford Algorithmus

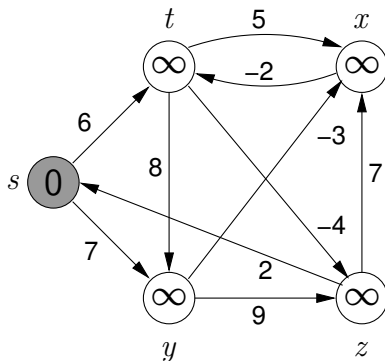
BELLMANFORD( $G, w, s$ )

**Input:** Gerichteter Graph  $G = (V, E)$ ,  
Gewichtsfunktion  $w$ , Startknoten  $s$

**Output:** true genau dann, wenn es in  $G$  keine Zyklen mit  
negativen Gewicht gibt, die von  $s$  aus erreichbar sind

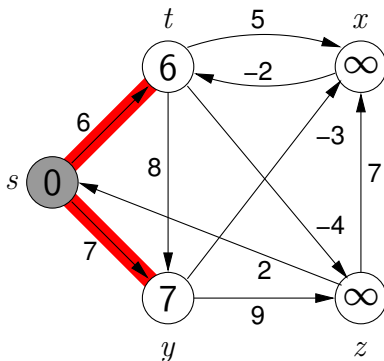
```
1  INITIALIZESINGLESOURCE( $G, s$ )
2  for  $i := 1$  to  $\|V\| - 1$  do
3      for jede Kante  $(u, v) \in E$  do
4          RELAX( $u, v$ )
5  for jede Kante  $(u, v) \in E$  do
6      if  $d[v] > d[u] + w(u, v)$  then
7          return false
8  return true
```

# Bellman-Ford Beispiel



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	$\infty$	$\infty$	$\infty$	$\infty$
$\pi[v]$	—	—	—	—	—

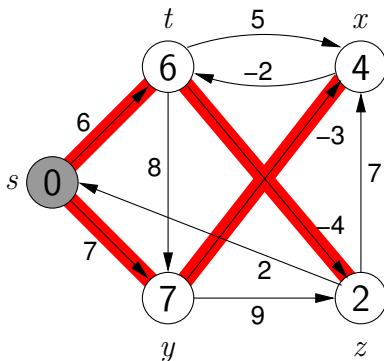
# Bellman-Ford Beispiel (Forts.)



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	6	$\infty$	7	$\infty$
$\pi[v]$	—	$s$	—	$s$	—

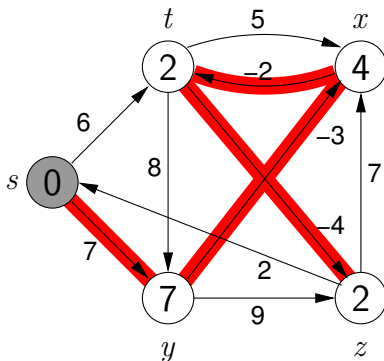


# Bellman-Ford Beispiel (Forts.)



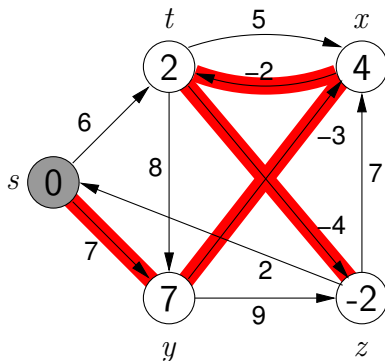
$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	6	4	7	2
$\pi[v]$	—	$s$	$y$	$s$	$t$

# Bellman-Ford Beispiel (Forts.)



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	2	4	7	2
$\pi[v]$	—	$x$	$y$	$s$	$t$

# Bellman-Ford Beispiel (Forts.)



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	2	4	7	-2
$\pi[v]$	—	$x$	$y$	$s$	$t$

# Korrektheit Bellman-Ford Algorithmus

**Lemma 2.** Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Angenommen, von  $s$  aus sind keine Zyklen mit negativem Gewicht erreichbar. Dann gilt nach  $\|V\| - 1$  Iterationen der for Schleife in Zeile 2–4 von `BELLMANFORD`( $G, w, s$ ), dass  $d[v] = \delta(s, v)$  für alle Knoten  $v \in V$ , die von  $s$  aus erreichbar sind.

**Korollar.** Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Für jeden Knoten  $v \in V$  gilt: Es gibt einen Pfad von  $s$  nach  $v$  genau dann, wenn nach Ausführung von `BELLMANFORD`( $G, s$ )  $d[v] < \infty$  gilt.

# Korrektheit Bellman-Ford Algorithmus (Forts.)

**Beweis.** Sei  $v$  ein Knoten, der von  $s$  aus erreichbar ist. Dann gibt es einen Pfad  $p = \langle v_0, v_1, \dots, v_k \rangle$  mit  $v_0 = s$  und  $v_k = v$ .

$p$  hat höchstens  $\|V\| - 1$  Kanten, also ist  $k \leq \|V\| - 1$ .

In jeder der  $\|V\| - 1$  Iterationen der for Schleife werden alle Kanten in  $E$  relaxiert.

In der  $i$ -ten Iteration wird insbesondere auch die Kante  $(v_{i-1}, v_i)$  relaxiert, wobei  $i = 1, 2, \dots, k$ .

Pfad Relaxation Eigenschaft:  $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$

# Korrektheit Bellman-Ford Algorithmus (Forts.)

**Satz.** Sei  $G = (V, E)$  ein gerichteter Graph mit Gewichtsfunktion  $w : E \mapsto \mathbb{R}$ . Sei  $s \in V$  ein beliebiger Knoten. Betrachte die Ausführung von  $\text{BELLMANFORD}(G, s)$ .

Es gilt:

- Falls  $G$  keine von  $s$  aus erreichbaren negativen Zyklen enthält, dann liefert der Algorithmus `true` zurück. Ferner gilt  $d[v] = \delta(s, v)$  für alle  $v \in V$ .
- Falls  $G$  einen von  $s$  aus erreichbaren negativen Zyklus enthält, liefert der Algorithmus `false` zurück.

# Korrektheit Bellman-Ford Algorithmus (Forts.)

## Beweis.

**Fall 1:**  $G$  besitzt keinen negativen Zyklus, der von  $s$  aus erreichbar ist.

- Wegen Lemma 2 gilt für alle von  $s$  aus erreichbaren Knoten  $v$ , dass  $d[v] = \delta(s, v)$ . Der Pfad von  $s$  nach  $v$  ist (in umgekehrter Richtung) in der  $\pi$ -Tabelle gespeichert.
- Ist  $v$  von  $s$  aus nicht erreichbar, dann greift die Kein-Pfad-Eigenschaft, d.h., es gilt  $d[u] = \delta(s, v) = \infty$ .

# Korrektheit Bellman-Ford Algorithmus (Forts.)

**Fall 2:** in  $G$  gibt es einen negativen Zyklus  $c = \langle v_0, v_1, \dots, v_k \rangle$  mit  $v_0 = v_k$ , der von  $s$  aus erreichbar ist. Es gilt:

$$\sum_{i=1}^k w(v_{i-1}, v_i) < 0.$$

Angenommen,  $\text{BELLMANFORD}(G, s)$  gibt `true` zurück. Dann muss für alle Kanten  $(u, v)$  die Ungleichung

$$d[v] \leq d[u] + w(u, v)$$

gelten.



# Korrektheit Bellman-Ford Algorithmus (Forts.)

Hieraus folgt:

$$\begin{aligned}\sum_{i=1}^k d[v_i] &\leq \sum_{i=1}^k (d[v_{i-1}] + w(v_{i-1}, v_i)) \\ &= \sum_{i=1}^k d[v_{i-1}] + \sum_{i=1}^k w(v_{i-1}, v_i)\end{aligned}$$

Da  $v_0 = v_k$ , kommt jeder Knoten des Zyklus genau einmal in den Summen  $\sum_{i=1}^k d[v_i]$  und  $\sum_{i=1}^k d[v_{i-1}]$  vor. Wegen des obigen Korollars gilt  $d[v_i] < \infty$  für alle  $i = 1, \dots, k$ . Somit:

$$\sum_{i=1}^k w(v_{i-1}, v_i) \geq 0$$

Widerspruch!

# Dijkstra Algorithmus

- Entwicklung von Edsger W. Dijkstra aus dem Jahre 1959
- Berechnung der kürzesten Pfade ausgehend von einem Startknoten  $s$  zu allen erreichbaren Knoten
- Voraussetzung: es gibt keine negativen Kantengewichte
- Greedy Algorithmus
- Einsatz einer Min Priority Queue
- Standardalgorithmus für die Berechnung von kürzesten Pfaden
- Laufzeit:  $O((\|V\| + \|E\|) \log_2 \|V\|)$

# DIJKSTRA( $G, s$ )

DIJKSTRA( $G, w, s$ )

**Input:** Gerichteter Graph  $G = (V, E)$ , Gewichtsfunktion  $w$   
mit  $w(u, v) \geq 0$  für alle  $(u, v) \in E$ , Startknoten  $s$

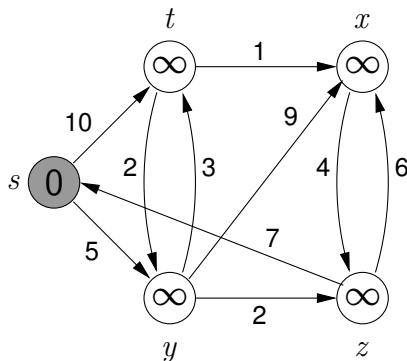
- 1 INITIALIZESINGLESOURCE( $G, s$ )
- 2  $S := \emptyset$
- 3 *Füge die Knoten in  $V$  in die Min Priority Queue  $Q$  ein*
- 4 **while**  $Q$  ist nicht leer **do**
- 5      $u := \text{EXTRACTMIN}(Q)$
- 6      $S := S \cup \{u\}$
- 7     **for** jeden Knoten  $v \in \text{Adj}[u]$  **do**
- 8         **if** ( $\text{RELAX}(u, v, w) = \text{true}$ ) **and** ( $v \notin S$ ) **then**
- 9             DECREASEKEY( $Q, v, d[v]$ )

# Laufzeit des Dijkstra Algorithmus

- Das Einfügen der Elemente in die Queue hat den Aufwand  $O(\|V\| \log_2 \|V\|)$
- Der RELAX-Aufruf in Zeile 8 wirkt sich auf die Reihenfolge der Knoten in der Queue aus. Die Umordnung der Elemente in der Queue hat den Aufwand  $O(\log_2 \|V\|)$
- Sind alle Knoten von  $s$  erreichbar, dann wird jede Kante relaxt. Aufwand:  $O(\|E\| \log_2 \|V\|)$ .
- Gesamtlaufzeit:

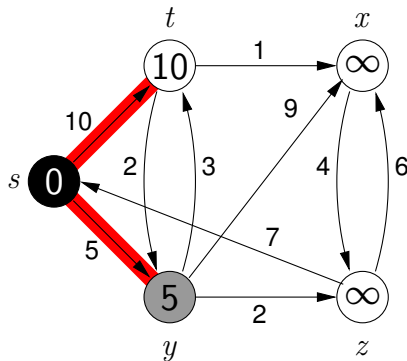
$$\begin{aligned} O(\|V\| \log_2 \|V\|) + O(\|E\| \log_2 \|V\|) \\ = O((\|V\| + \|E\|) \log_2 \|V\|) \end{aligned}$$

# Dijkstra Beispiel



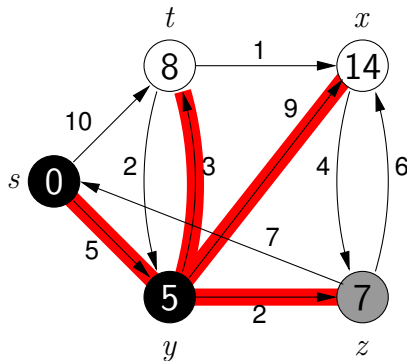
$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	$\infty$	$\infty$	$\infty$	$\infty$
$\pi[v]$	—	—	—	—	—

# Dijkstra Beispiel (Forts.)



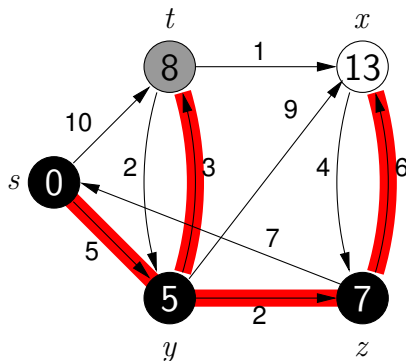
$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	10	$\infty$	5	$\infty$
$\pi[v]$	—	$s$	—	$s$	—

# Dijkstra Beispiel (Forts.)



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	8	14	5	7
$\pi[v]$	—	$y$	$y$	$s$	$y$

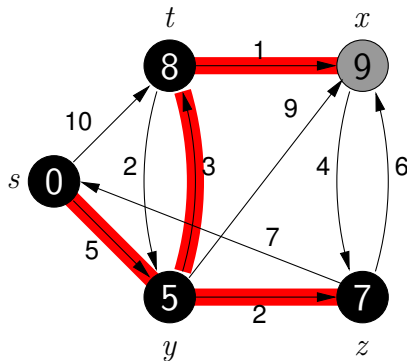
# Dijkstra Beispiel (Forts.)



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	8	13	5	7
$\pi[v]$	—	$y$	$z$	$s$	$y$

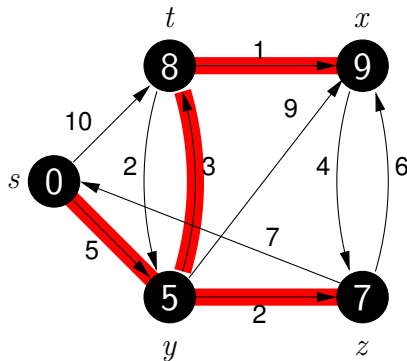


# Dijkstra Beispiel (Forts.)



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	8	9	5	7
$\pi[v]$	—	$y$	$t$	$s$	$y$

# Dijkstra Beispiel (Forts.)



$v$	$s$	$t$	$x$	$y$	$z$
$d[v]$	0	8	9	5	7
$\pi[v]$	—	$y$	$t$	$s$	$y$

# Korrektheit des Dijkstra Algorithmus

**Satz.** Sei  $G = (V, E)$  ein gerichteter Graph mit nicht-negativen Kantengewichten und sei  $s \in V$ . Dann gilt nach dem Aufruf  $\text{DIJKSTRA}(G, s)$   $d[v] = \delta(s, v)$  für alle Knoten  $v \in V$ .

**Beweis.** Zu zeigen ist die folgende **Schleifeninvariante**:

*Zu Beginn jedes Durchlaufs der while Schleife in Zeile 4–8 gilt  $d[v] = \delta(s, v)$  für alle Knoten  $v \in S$ .*

**Beobachtung:** Es genügt zu zeigen, dass  $d[u] = \delta(s, u)$  gilt, wenn  $u$  zu  $S$  hinzugefügt wird. Wegen der Obere Schranke Eigenschaft kann sich dieser Wert dann nicht mehr ändern.

# Korrektheit des Dijkstra Algorithmus (Forts.)

**Initialisierung:** Zu Beginn ist  $S$  leer und die Invariante trivialerweise korrekt ✓

**Aufrechterhaltung:** Angenommen,  $u$  ist der erste Knoten, der zu  $S$  hinzugefügt wird, für den  $d[u] \neq \delta(s, u)$  gilt.

Es muss  $s \neq u$  gelten, da  $d[s] = \delta(s, s) = 0$  zum Zeitpunkt des Einfügens von  $s$  in  $S$  gilt. Folglich war  $S$  nicht leer, bevor  $u$  hinzugefügt wurde.

Ferner existiert ein Pfad von  $s$  nach  $u$ . Andernfalls wäre  $d[u] = \delta(s, u) = \infty$  (Kein-Pfad-Eigenschaft) und somit  $d[u] = \delta(s, u)$ .

# Korrektheit des Dijkstra Algorithmus (Forts.)

Sei  $p$  ein kürzester Pfad von  $s$  nach  $u$ . Vor dem Hinzufügen von  $u$  zu  $S$  verbindet  $p$  den Knoten  $s \in S$  mit dem Knoten  $u \in V \setminus S$ .

Betrachte nun den Zeitpunkt, an dem  $u$  aus der Queue entfernt wird.

Sei  $y$  der erste Knoten auf dem Pfad, der in  $V \setminus S$  enthalten ist. Sei  $x \in S$  der Vorgänger von  $y$  auf dem Pfad.

Der Pfad  $p$  ist wie folgt aufgebaut:

$$s \xrightarrow{p_1} x \rightarrow y \xrightarrow{p_2} u$$

Laut Invariante gilt zum Zeitpunkt, zu dem  $x$  zu  $S$  hinzugefügt wird,  $d[x] = \delta(s, x)$ .

Danach wird die Kante  $(x, y)$  relaxt. Wegen der Konvergenzeigenschaft folgt, dass  $d[y] = \delta(s, y)$  sein muss.

# Korrektheit des Dijkstra Algorithmus (Forts.)

Da  $y$  vor  $u$  auf  $p$  liegt und es keine negativen Kantengewichte gibt, folgt:

$$\begin{aligned}d[y] &= \delta(s, y) \\ &\leq \delta(s, u) \\ &\leq d[u] \quad (\text{wegen Obere-Schranke-Eigenschaft})\end{aligned}$$

Sowohl  $u$  als auch  $y$  sind in  $V \setminus S$ . Da  $u$  vor  $y$  aus der Queue genommen wird, muss  $d[u] \leq d[y]$  gelten. Hieraus folgt:

$$d[y] = \delta(s, y) = \delta(s, u) = d[u].$$

Widerspruch! Also muss  $d[u] = \delta(s, u)$  sein.

# Korrektheit des Dijkstra Algorithmus (Forts.)

**Beendigung:** Nach Beendigung der while Schleife ist die Queue  $Q$  leer. Da sie zu Beginn alle Knoten enthielt, muss  $S = V$  sein. Gemäß Invariante gibt für alle  $v \in V$ , dass  $d[v] = \delta(s, v)$ .

# All-Pairs Shortest Paths

**Gegeben:** Gerichteter Graph  $G = (V, E)$

**Aufgabe:** Berechne für jedes Knotenpaar  $u, v$  einen kürzesten Pfad  $u \xrightarrow{p} v$ .

**Alternativen:**

- Führe den Bellman-Ford Algorithmus aus für jeden Knoten  $s \in V \rightsquigarrow$  Laufzeit  $O(\|V\|^2 \|E\|)$
- Führe den Dijkstra Algorithmus aus für jeden Knoten  $s \in V \rightsquigarrow$  Laufzeit  $O(\|V\|(\|V\| + \|E\|) \log_2 \|V\|)$
- Benutze den Algorithmus von Floyd Warshall  $\rightsquigarrow$  Laufzeit  $O(\|V\|^3)$



# Floyd Warshall Algorithmus

- Berechnung eines kürzesten Pfads für alle Knotenpaare  $(u, v)$  in  $V$
- Technik: Dynamisches Programmieren
- Annahme:  $V = \{1, 2, \dots, n\}$
- Idee: Berechne die  $n \times n$  Matrizen  $D^{(0)}, D^{(1)}, \dots, D^{(n)}$
- $D^k[u, v]$  steht für die Länge des kürzesten Pfads von  $u$  nach  $v$ , der ausschließlich Zwischenknoten aus  $\{1, \dots, k\}$  enthält
- Pfadinformationen werden in den Matrizen  $\Pi^{(k)}$ ,  $k = 0, 1, \dots, n$  gespeichert
- Laufzeit:  $\Theta(\|V\|^3)$

# Einfache Pfade und Zwischenknoten

**Definition.** Sei  $G = (V, E)$  ein Graph.

- Ein Pfad  $p = \langle v_1, v_2, \dots, v_l \rangle$  in  $G$  nennt man **einfach**, falls er keine Zyklen enthält.
- Sei  $p$  ein einfacher Pfad. Ein Knoten  $v_i$  auf  $p$  heißt **Zwischenknoten**, falls  $v_i \neq v_1$  und  $v_i \neq v_l$ .

# Idee hinter dem Floyd Warshall Algorithmus

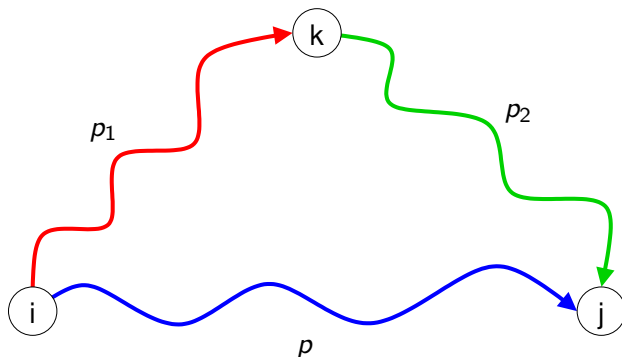
**Annahme:**  $V = \{1, 2, \dots, n\}$

**Gegeben:** Sei  $k \in \{1, \dots, n\}$ . Betrachte alle für ein beliebiges Paar  $i, j$  von Knoten alle einfachen Pfade von  $i$  nach  $j$ , die ausschließlich Zwischenknoten in  $\{1, \dots, k\}$  enthalten. Sei  $p$  ein solcher Pfad mit minimalem Gewicht.

**Beobachtung:**

- **Fall 1:**  $k$  ist kein Zwischenknoten auf  $p$ , d.h., alle Zwischenknoten auf  $p$  sind in  $\{1, \dots, k-1\}$ .
- **Fall 2:**  $k$  ist ein Zwischenknoten auf  $p$ . Dann kann man  $p$  zerlegen in  $i \xrightarrow{p_1} k \xrightarrow{p_2} j$ , wobei die Zwischenknoten auf  $p_1$  und  $p_2$  in  $\{1, \dots, k-1\}$  sind.

# Idee Floyd Warshall Algorithmus (Forts.)



**Gegeben:** Pfade  $p$ ,  $p_1$  und  $p_2$  von  $i$  nach  $j$  mit minimalem Gewichten und Zwischenknoten in  $\{1, \dots, k-1\}$

**Frage:** Bringt der Knoten  $k$  eine Verbesserung?

# Rekursionsgleichungen

Abstandsmatrix  $D^{(k)}$ :

- $k = 0$ :  $D^{(k)}[i, j] = w(i, j)$
- $k \geq 1$ :  $D^{(k)}[i, j] = \min(D^{(k-1)}[i, j], D^{(k-1)}[i, k] + D^{(k-1)}[k, j])$

Vorgängermatrix  $\Pi^{(0)}$ :

- $i \neq j$  und  $w(i, j) < \infty$ :  $\Pi^{(0)}[i, j] = i$
- Ansonsten:  $\Pi^{(0)}[i, j] = \text{NIL}$

Vorgängermatrix  $\Pi^{(k)}$ ,  $k = 1, 2, \dots, n$ :

- $D^{(k-1)}[i, j] \leq D^{(k-1)}[i, k] + D^{(k-1)}[k, j]$ :  $\Pi^{(k)}[i, j] = \Pi^{(k-1)}[i, j]$
- Ansonsten:  $\Pi^{(k)}[i, j] = \Pi^{(k-1)}[k, j]$

# Algorithmus FLOYDWARSHALL( $W$ )

FLOYDWARSHALL( $W$ )

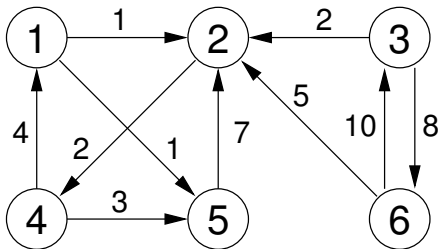
**Input:** Abstandsmatrix  $W$

```
1  $n := \text{rows}(W)$ 
2  $D^{(0)} := W$ 
3 for  $i := 1$  to  $n$  do
4   for  $j := 1$  to  $n$  do
5     if  $i \neq j$  and  $w(i, j) < \infty$  then
6        $\Pi^{(0)}[i, j] := i$ 
7     else
8        $\Pi^{(0)}[i, j] := \text{NIL}$ 
```

# Algorithmus FLOYDWARSHALL( $W$ ) (Forts.)

```
9  for  $k := 1$  to  $n$  do
10  for  $i := 1$  to  $n$  do
11    for  $j := 1$  to  $n$  do
12      if  $D^{(k-1)}[i, j] \leq D^{(k-1)}[i, k] + D^{(k-1)}[k, j]$  then
13         $D^{(k)}[i, j] := D^{(k-1)}[i, j]$ 
14         $\Pi^{(k)}[i, j] := \Pi^{(k-1)}[i, j]$ 
15      else
16         $D^{(k)}[i, j] := D^{(k-1)}[i, k] + D^{(k-1)}[k, j]$ 
17         $\Pi^{(k)}[i, j] := \Pi^{(k-1)}[k, j]$ 
```

# Beispiel FLOYDWARSHALL( $W$ )





# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)

$k = 0$ :

$$D^{(0)} = \begin{pmatrix} 0 & 1 & \infty & \infty & 1 & \infty \\ \infty & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & 8 \\ 4 & \infty & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$\Pi^{(0)} = \begin{pmatrix} - & 1 & - & - & 1 & - \\ - & - & - & 2 & - & - \\ - & 3 & - & - & - & 3 \\ 4 & - & - & - & 4 & - \\ - & 5 & - & - & - & - \\ - & 6 & 6 & - & - & - \end{pmatrix}$$

# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)

$k = 1$ :

$$D^{(1)} = \begin{pmatrix} 0 & 1 & \infty & \infty & 1 & \infty \\ \infty & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & \infty & \infty & 8 \\ 4 & 5 & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & \infty & 0 & \infty \\ \infty & 5 & 10 & \infty & \infty & 0 \end{pmatrix}$$

$$\Pi^{(1)} = \begin{pmatrix} - & 1 & - & - & 1 & - \\ - & - & - & 2 & - & - \\ - & 3 & - & - & - & 3 \\ 4 & 1 & - & - & 4 & - \\ - & 5 & - & - & - & - \\ - & 6 & 6 & - & - & - \end{pmatrix}$$

# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)

$k = 2$ :

$$D^{(2)} = \begin{pmatrix} 0 & 1 & \infty & 3 & 1 & \infty \\ \infty & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & 4 & \infty & 8 \\ 4 & 5 & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & 9 & 0 & \infty \\ \infty & 5 & 10 & 7 & \infty & 0 \end{pmatrix}$$

$$\Pi^{(2)} = \begin{pmatrix} - & 1 & - & 2 & 1 & - \\ - & - & - & 2 & - & - \\ - & 3 & - & 2 & - & 3 \\ 4 & 1 & - & - & 4 & - \\ - & 5 & - & 2 & - & - \\ - & 6 & 6 & 2 & - & - \end{pmatrix}$$

# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)

$k = 3$ :

$$D^{(3)} = \begin{pmatrix} 0 & 1 & \infty & 3 & 1 & \infty \\ \infty & 0 & \infty & 2 & \infty & \infty \\ \infty & 2 & 0 & 4 & \infty & 8 \\ 4 & 5 & \infty & 0 & 3 & \infty \\ \infty & 7 & \infty & 9 & 0 & \infty \\ \infty & 5 & 10 & 7 & \infty & 0 \end{pmatrix}$$

$$\Pi^{(3)} = \begin{pmatrix} - & 1 & - & 2 & 1 & - \\ - & - & - & 2 & - & - \\ - & 3 & - & 2 & - & 3 \\ 4 & 1 & - & - & 4 & - \\ - & 5 & - & 2 & - & - \\ - & 6 & 6 & 2 & - & - \end{pmatrix}$$

# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)

$k = 4$ :

$$D^{(4)} = \begin{pmatrix} 0 & 1 & \infty & 3 & 1 & \infty \\ 6 & 0 & \infty & 2 & 5 & \infty \\ 8 & 2 & 0 & 4 & 7 & 8 \\ 4 & 5 & \infty & 0 & 3 & \infty \\ 13 & 7 & \infty & 9 & 0 & \infty \\ 11 & 5 & 10 & 7 & 10 & 0 \end{pmatrix}$$

$$\Pi^{(4)} = \begin{pmatrix} - & 1 & - & 2 & 1 & - \\ 4 & - & - & 2 & 4 & - \\ 4 & 3 & - & 2 & 4 & 3 \\ 4 & 1 & - & - & 4 & - \\ 4 & 5 & - & 2 & - & - \\ 4 & 6 & 6 & 2 & 4 & - \end{pmatrix}$$

# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)

$k = 5$ :

$$D^{(5)} = \begin{pmatrix} 0 & 1 & \infty & 3 & 1 & \infty \\ 6 & 0 & \infty & 2 & 5 & \infty \\ 8 & 2 & 0 & 4 & 7 & 8 \\ 4 & 5 & \infty & 0 & 3 & \infty \\ 13 & 7 & \infty & 9 & 0 & \infty \\ 11 & 5 & 10 & 7 & 10 & 0 \end{pmatrix}$$

$$\Pi^{(5)} = \begin{pmatrix} - & 1 & - & 2 & 1 & - \\ 4 & - & - & 2 & 4 & - \\ 4 & 3 & - & 2 & 4 & 3 \\ 4 & 1 & - & - & 4 & - \\ 4 & 5 & - & 2 & - & - \\ 4 & 6 & 6 & 2 & 4 & - \end{pmatrix}$$

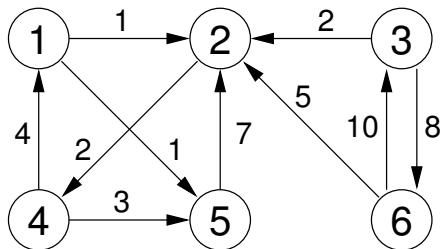
# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)

$k = 6$ :

$$D^{(6)} = \begin{pmatrix} 0 & 1 & \infty & 3 & 1 & \infty \\ 6 & 0 & \infty & 2 & 5 & \infty \\ 8 & 2 & 0 & 4 & 7 & 8 \\ 4 & 5 & \infty & 0 & 3 & \infty \\ 13 & 7 & \infty & 9 & 0 & \infty \\ 11 & 5 & 10 & 7 & 10 & 0 \end{pmatrix}$$

$$\Pi^{(6)} = \begin{pmatrix} - & 1 & - & 2 & 1 & - \\ 4 & - & - & 2 & 4 & - \\ 4 & 3 & - & 2 & 4 & 3 \\ 4 & 1 & - & - & 4 & - \\ 4 & 5 & - & 2 & - & - \\ 4 & 6 & 6 & 2 & 4 & - \end{pmatrix}$$

# Beispiel FLOYDWARSHALL( $W$ ) (Forts.)



$$D^{(6)} = \begin{pmatrix} 0 & 1 & \infty & 3 & 1 & \infty \\ 6 & 0 & \infty & 2 & 5 & \infty \\ 8 & 2 & 0 & 4 & 7 & 8 \\ 4 & 5 & \infty & 0 & 3 & \infty \\ 13 & 7 & \infty & 9 & 0 & \infty \\ 11 & 5 & 10 & 7 & 10 & 0 \end{pmatrix}$$

$$\Pi^{(6)} = \begin{pmatrix} - & 1 & - & 2 & 1 & - \\ 4 & - & - & 2 & 4 & - \\ 4 & 3 & - & 2 & 4 & 3 \\ 4 & 1 & - & - & 4 & - \\ 4 & 5 & - & 2 & - & - \\ 4 & 6 & 6 & 2 & 4 & - \end{pmatrix}$$



# Zusammenfassung

- Die Berechnung von kürzesten Pfaden in Graphen ist die Grundlage für viele Anwendungen
- Enthält der Graph negative Zyklen, dann ist Vorsicht geboten
- Single Source Shortest Path Problem
  - ▷ Algorithmus von Bellman-Ford
  - ▷ Algorithmus von Dijkstra
- All Pairs Shortest Paths Problem
  - ▷ Algorithmus von Floyd-Warshall
  - ▷ Algorithmus von Johnson (für dünne Graphen)