

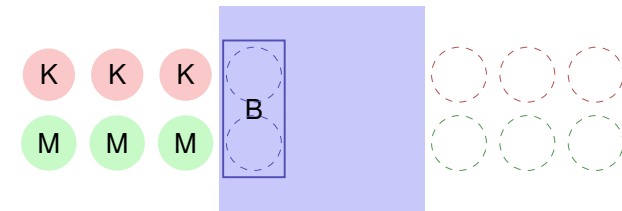
# 14. Kürzeste Wege

Motivation, Dijkstras Algorithmus auf Distanzgraphen, Algorithmus von Bellman-Ford

[Ottman/Widmayer, Kap. 9.5.1-9.5.2 Cormen et al, Kap. 24.1-24.3]

## Flussüberquerung (Missionare und Kannibalen)

Problem: Drei Kannibalen und drei Missionare stehen an einem Ufer eines Flusses. Ein dort bereitstehendes Boot fasst maximal zwei Personen. Zu keiner Zeit dürfen an einem Ort (Ufer oder Boot) mehr Kannibalen als Missionare sein. Wie kommen die Missionare und Kannibalen möglichst schnell über den Fluss? <sup>23</sup>



<sup>23</sup>Es gibt leichte Variationen dieses Problems, es ist auch äquivalent zum Problem der eifersüchtigen Ehemänner

## Formulierung als Graph

Zähle alle erlaubten Konfigurationen als Knoten auf und verbinde diese mit einer Kante, wenn Überfahrt möglich ist. Das Problem ist dann ein Problem des kürzesten Pfades

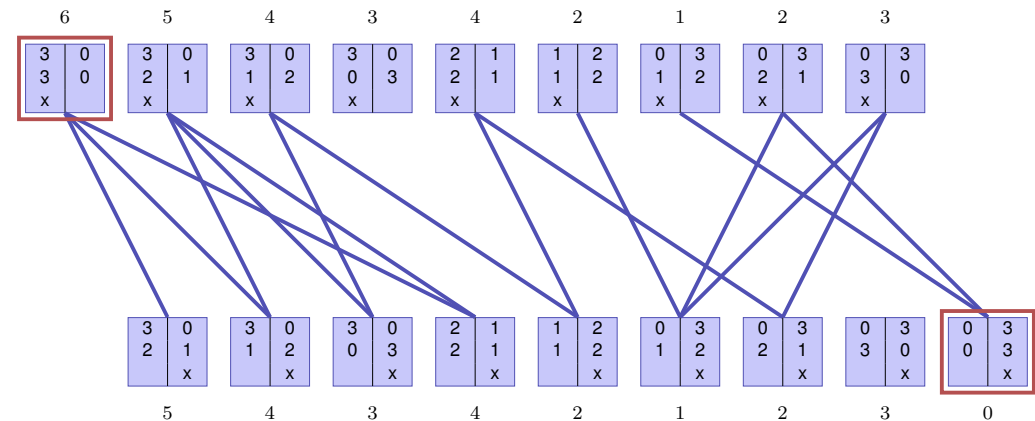
Beispiel



6 Personen am linken Ufer

4 Personen am linken Ufer

## Das ganze Problem als Graph

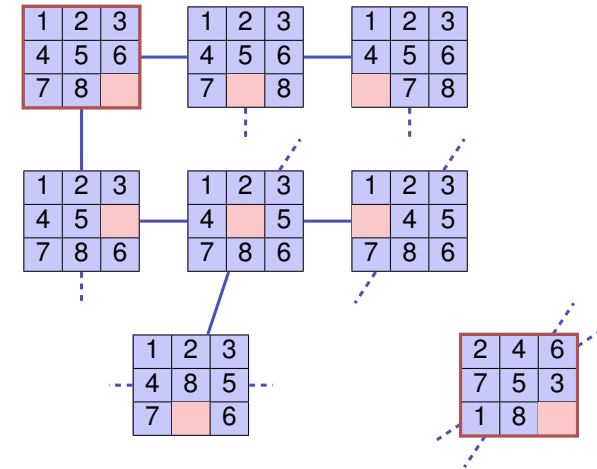


## Anderes Beispiel: Schiebepuzzle

Wollen die schnellste Lösung finden für



## Problem als Graph

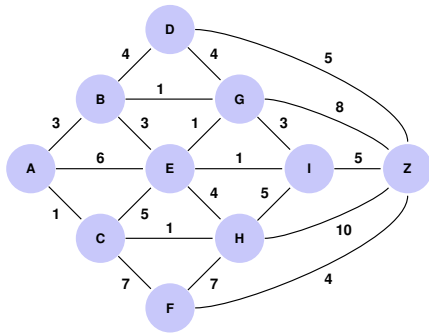


352

353

## Routenfinder

Gegeben Städte A - Z und Distanzen zwischen den Städten.

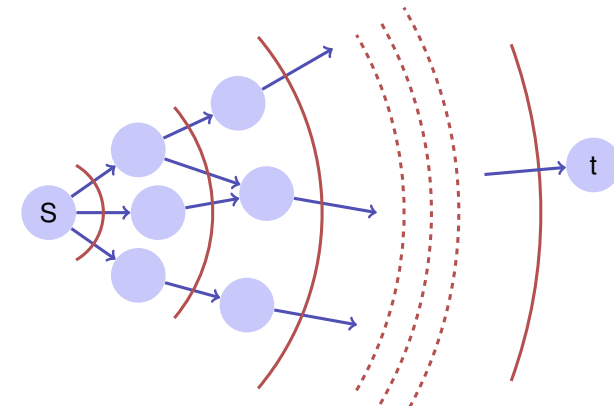


Was ist der kürzeste Weg von A nach Z?

## Einfachster Fall

Konstantes Kantengewicht 1 (oBdA)

Lösung: Breitensuche



354

355

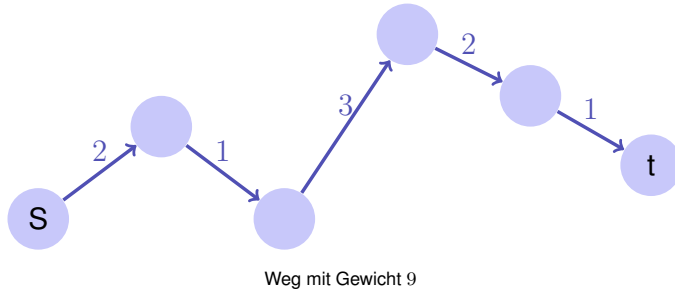
## Gewichtete Graphen

**Gegeben:**  $G = (V, E, c)$ ,  $c : E \rightarrow \mathbb{R}$ ,  $s, t \in V$ .

**Gesucht:** Länge (Gewicht) eines kürzesten Weges von  $s$  nach  $t$ .

**Weg:**  $p = \langle s = v_0, v_1, \dots, v_k = t \rangle$ ,  $(v_i, v_{i+1}) \in E$  ( $0 \leq i < k$ )

**Gewicht:**  $c(p) := \sum_{i=0}^{k-1} c((v_i, v_{i+1}))$ .



356

## Kürzeste Wege

**Notation:** Wir schreiben

$$u \xrightarrow{p} v \quad \text{oder} \quad p : u \rightsquigarrow v$$

und meinen einen Weg  $p$  von  $u$  nach  $v$

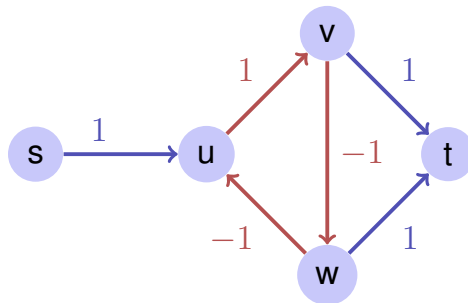
**Notation:**  $\delta(u, v)$  = Gewicht eines kürzesten Weges von  $u$  nach  $v$ :

$$\delta(u, v) = \begin{cases} \infty & \text{kein Weg von } u \text{ nach } v \\ \min\{c(p) : u \xrightarrow{p} v\} & \text{sonst} \end{cases}$$

357

## Beobachtungen (1)

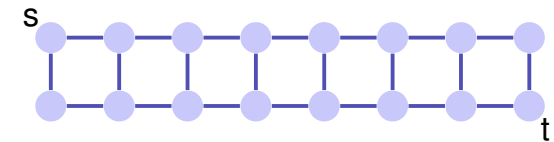
Es gibt Situationen, in denen kein kürzester Weg existiert: negative Zyklen könnten auftreten.



358

## Beobachtungen (2)

Es kann exponentiell viele Wege geben.



(mindestens  $2^{|V|/2}$  Wege von  $s$  nach  $t$ )

$\Rightarrow$  Alle Wege probieren ist zu ineffizient.

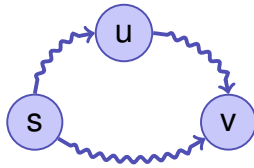
359

## Beobachtungen (3)

### Dreiecksungleichung

Für alle  $s, u, v \in V$ :

$$\delta(s, v) \leq \delta(s, u) + \delta(u, v)$$



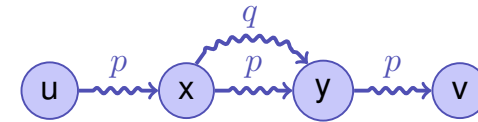
Ein kürzester Weg von  $s$  nach  $v$  (ohne weitere Einschränkungen) kann nicht länger sein als ein kürzester Weg von  $s$  nach  $v$ , der  $u$  enthalten muss.

360

## Beobachtungen (4)

### Optimale Substruktur

Teilpfade von kürzesten Pfaden sind kürzeste Pfade: Sei  $p = \langle v_0, \dots, v_k \rangle$  ein kürzester Pfad von  $v_0$  nach  $v_k$ . Dann ist jeder der Teilpfade  $p_{ij} = \langle v_i, \dots, v_j \rangle$  ( $0 \leq i < j \leq k$ ) ein kürzester Pfad von  $v_i$  nach  $v_j$ .



Wäre das nicht so, könnte man einen der Teilpfade kürzen, Widerspruch zur Voraussetzung.

361

## Beobachtungen (5)

Kürzeste Wege enthalten keine Zyklen

- 1 Kürzester Weg enthält negativen Zyklus: es existiert kein kürzester Weg. Widerspruch.
- 2 Weg enthält positiven Zyklus: Weglassen des positiven Zyklus kann den Weg verkürzen: Widerspruch
- 3 Weg enthält Zyklus vom Gewicht 0: Weglassen des Zyklus verändert das Pfadgewicht nicht. Weglassen (Konvention).

362

## Zutaten für einen Algorithmus

Gesucht: Kürzeste Wege von einem Startknoten  $s$  aus.

- Gewicht des kürzesten bisher gefundenen Pfades

$$d_s : V \rightarrow \mathbb{R}$$

**Zu Beginn:**  $d_s[v] = \infty$  für alle Knoten  $v \in V$ .

**Ziel:**  $d_s[v] = \delta(s, v)$  für alle  $v \in V$ .

- Vorgänger eines Knotens

$$\pi_s : V \rightarrow V$$

Zu Beginn  $\pi_s[v]$  undefiniert für jeden Knoten  $v \in V$

363

## Allgemeiner Algorithmus

- 1 Initialisiere  $d_s$  und  $\pi_s$ :  $d_s[v] = \infty$ ,  $\pi_s[v] = \text{null}$  für alle  $v \in V$
- 2 Setze  $d_s[s] \leftarrow 0$
- 3 Wähle eine Kante  $(u, v) \in E$ 

Relaxiere  $(u, v)$ :  
 if  $d_s[v] > d_s[u] + c(u, v)$  then  
      $d_s[v] \leftarrow d_s[u] + c(u, v)$   
      $\pi_s[v] \leftarrow u$
- 4 Wiederhole 3 bis nichts mehr relaxiert werden kann.  
 (bis  $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$ )

## Relaxieren ist sicher

Zu jeder Zeit gilt in obigem Algorithmus

$$d_s[v] \geq \delta(s, v) \quad \forall v \in V$$

Im Relaxierschritt:

$$\delta(s, v) \leq \delta(s, u) + \delta(u, v) \quad \text{[Dreiecksungleichung]}$$

$$\delta(s, u) \leq d_s[u] \quad \text{[Induktionsvoraussetzung]}$$

$$\delta(u, v) \leq c(u, v) \quad \text{[Minimalität von } \delta \text{]}$$

$$\Rightarrow d_s[u] + c(u, v) \geq \delta(s, v)$$

$$\Rightarrow \min\{d_s[v], d_s[u] + c(u, v)\} \geq \delta(s, v)$$

364

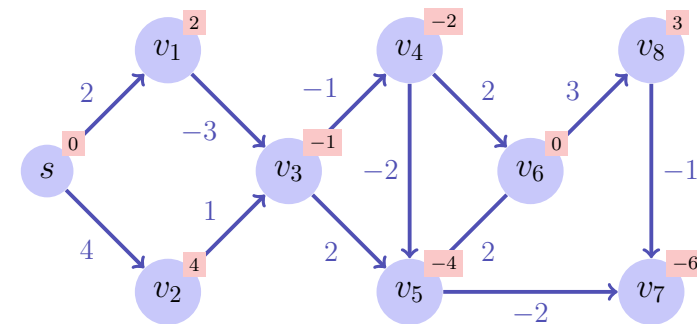
365

## Zentrale Frage

Wie / in welcher Reihenfolge wählt man die Kanten in obigem Algorithmus?

## Spezialfall: Gerichteter Azyklischer Graph (DAG)

DAG  $\Rightarrow$  Topologische Sortierung liefert optimale Besuchsreihenfolge

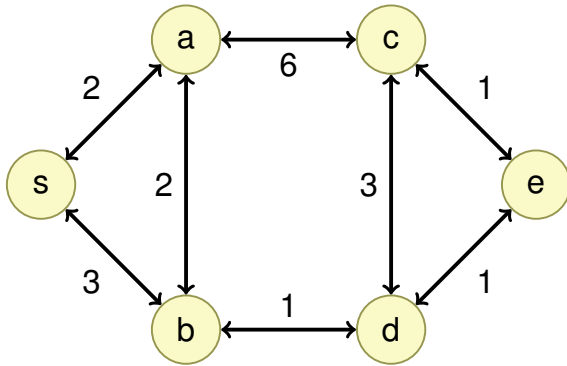


Top. Sortieren:  $\Rightarrow$  Reihenfolge  $s, v_1, v_2, v_3, v_4, v_6, v_5, v_8, v_7$ .

366

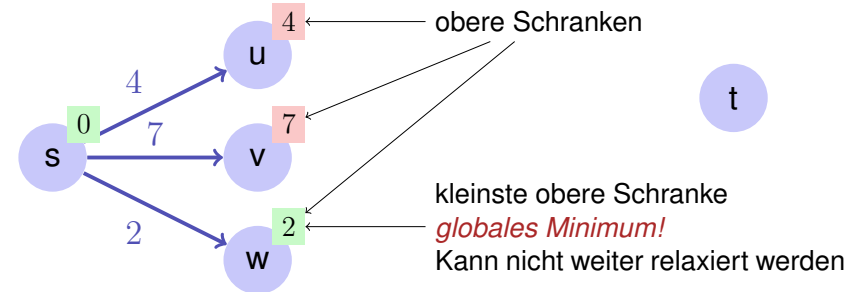
367

## Annahme (vorübergehend)



Alle Gewichte von  $G$  sind *positiv*.

## Beobachtung (Dijkstra)



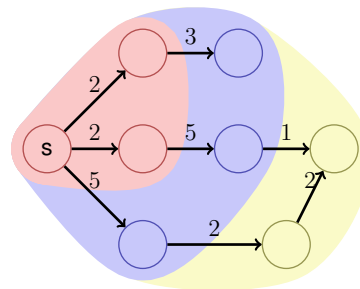
368

369

## Grundidee

Menge  $V$  aller Knoten wird unterteilt in

- die Menge  $M$  von Knoten, für die schon ein kürzester Weg von  $s$  bekannt ist
- die Menge  $R = \bigcup_{v \in M} N^+(v) \setminus M$  von Knoten, für die kein kürzester Weg bekannt ist, die jedoch von  $M$  direkt erreichbar sind.
- die Menge  $U = V \setminus (M \cup R)$  von Knoten, die noch nicht berücksichtigt wurden.

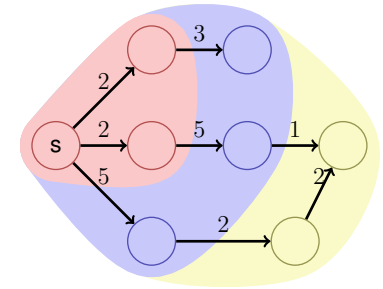


370

## Induktion

Induktion über  $|M|$ : Wähle Knoten aus  $R$  mit kleinster oberer Schranke. Nimm  $r$  zu  $M$  hinzu, und update  $R$  und  $U$ .

Korrektheit: Ist innerhalb einer "Wellenfront" einmal ein Knoten mit minimalem Pfadgewicht  $w$  gefunden, kann kein Pfad über später gefundene Knoten (mit Gewicht  $\geq w$ ) zu einer Verbesserung führen.



371

## Algorithmus Dijkstra( $G, s$ )

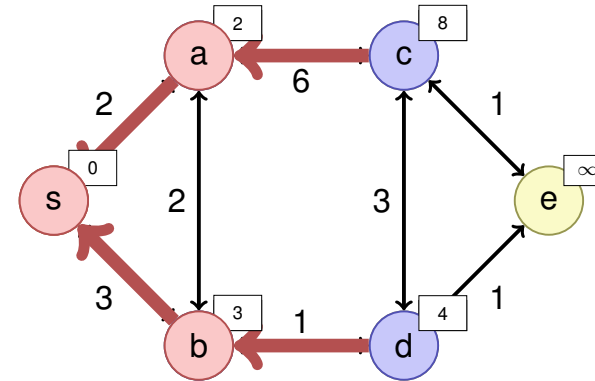
**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Minimale Gewichte  $d$  der kürzesten Pfade und Vorgängerknoten für jeden Knoten.

```

foreach  $u \in V$  do
   $d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$ 
 $d_s[s] \leftarrow 0; R \leftarrow \{s\}$ 
while  $R \neq \emptyset$  do
   $u \leftarrow \text{ExtractMin}(R)$ 
  foreach  $v \in N^+(u)$  do
    if  $d_s[u] + c(u, v) < d_s[v]$  then
       $d_s[v] \leftarrow d_s[u] + c(u, v)$ 
       $\pi_s[v] \leftarrow u$ 
       $R \leftarrow R \cup \{v\}$ 
  
```

## Beispiel



$M = \{s, a, b\}$   
 $R = \{c, d\}$   
 $U = \{e\}$

372

373

## Zur Implementation: Datenstruktur für $R$ ?

Benötigte Operationen:

- Insert (Hinzunehmen zu  $R$ )
- ExtractMin (über  $R$ ) und DecreaseKey (Update in  $R$ )

```

foreach  $v \in N^+(u)$  do
  if  $d_s[u] + c(u, v) < d_s[v]$  then
     $d_s[v] \leftarrow d_s[u] + c(u, v)$ 
     $\pi_s[v] \leftarrow u$ 
    if  $v \in R$  then
      DecreaseKey( $R, v$ ) // Update eines  $d(v)$  im Heap zu  $R$ 
    else
       $R \leftarrow R \cup \{v\}$  // Einfügen eines neuen  $d(v)$  im Heap zu  $R$ 
  
```

MinHeap!

## DecreaseKey

- DecreaseKey: Aufsteigen im MinHeap in  $\mathcal{O}(\log |V|)$
- Position im Heap?
  - Möglichkeit (a): Speichern am Knoten
  - Möglichkeit (b): Hashtabelle über Knoten
  - Möglichkeit (c): Knoten nach erfolgreichem Relaxieren erneut einfügen. Knoten beim Entnehmen als "deleted" kennzeichnen (Lazy Deletion).<sup>24</sup>

<sup>24</sup>Für die lazy deletion benötigt man ein Paar von Kante (oder Zielknoten) und Distanz

374

375

## Laufzeit

- $|V| \times \text{ExtractMin}$ :  $\mathcal{O}(|V| \log |V|)$
- $|E| \times \text{Insert oder DecreaseKey}$ :  $\mathcal{O}(|E| \log |V|)$
- $1 \times \text{Init}$ :  $\mathcal{O}(|V|)$
- Insgesamt:  $\mathcal{O}(|E| \log |V|)$ .

## Allgemeine Bewertete Graphen

Verbesserungsschritt wie bisher, aber mit Rückgabewert:

$\text{Relax}(u, v)$  ( $u, v \in V, (u, v) \in E$ )

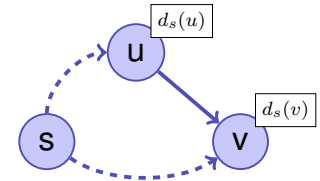
if  $d_s[u] + c(u, v) < d_s[v]$  then

$d_s[v] \leftarrow d_s[u] + c(u, v)$

$\pi_s[v] \leftarrow u$

    return true

return false



Problem: Zyklen mit negativen Gewichten können Weg verkürzen: es muss keinen kürzesten Weg mehr geben

376

377

## Dynamic Programming Ansatz (Bellman)

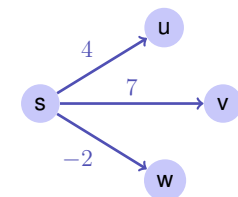
Induktion über Anzahl Kanten.  $d_s[i, v]$ : Kürzeste Weglänge von  $s$  nach  $v$  über maximal  $i$  Kanten.

$$d_s[i, v] = \min\{d_s[i-1, v], \min_{(u,v) \in E} (d_s[i-1, u] + c(u, v))\}$$

$$d_s[0, s] = 0, d_s[0, v] = \infty \forall v \neq s.$$

## Dynamic Programming Ansatz (Bellman)

	$s$	$\dots$	$v$	$\dots$	$w$
0	0	$\infty$	$\infty$	$\infty$	$\infty$
1	0	$\infty$	7	$\infty$	-2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	0	$\dots$	$\dots$	$\dots$	$\dots$



Algorithmus: Iteriere über letzte Zeile bis die Relaxationsschritte keine Änderung mehr ergeben, maximal aber  $n-1$  mal. Wenn dann noch Änderungen, dann gibt es keinen kürzesten Pfad.

378

379



## Algorithmus Bellman-Ford( $G, s$ )

**Input:** Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Wenn Rückgabe true, Minimale Gewichte  $d$  der kürzesten Pfade zu jedem Knoten, sonst kein kürzester Pfad.

```
foreach  $u \in V$  do
   $d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$ 
 $d_s[s] \leftarrow 0;$ 
for  $i \leftarrow 1$  to  $|V|$  do
   $f \leftarrow \text{false}$ 
  foreach  $(u, v) \in E$  do
     $f \leftarrow f \vee \text{Relax}(u, v)$ 
  if  $f = \text{false}$  then return true
return false;
```