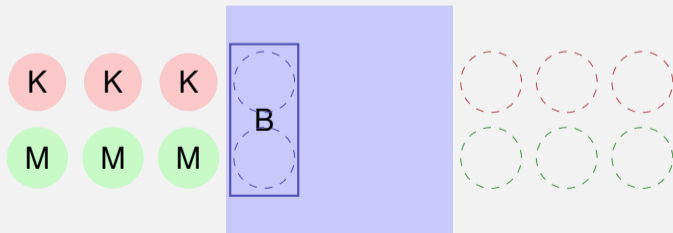


13. Kürzeste Wege

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? ⁹



⁹Es 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

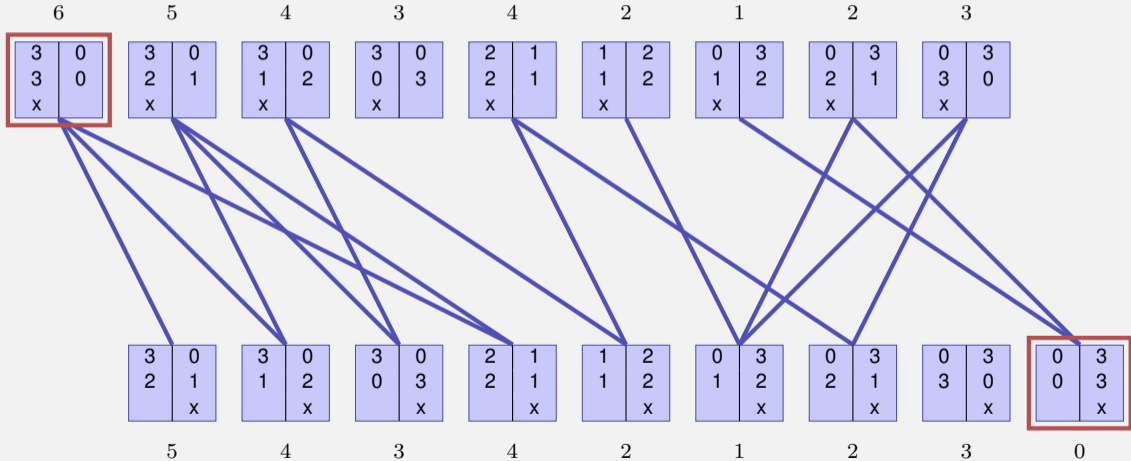
Beispiel

	links	rechts		links	rechts
Missionare	3	0	Überfahrt möglich	Missionare	2
Kannibalen	3	0		Kannibalen	2
Boot	x			Boot	x

6 Personen am linken Ufer

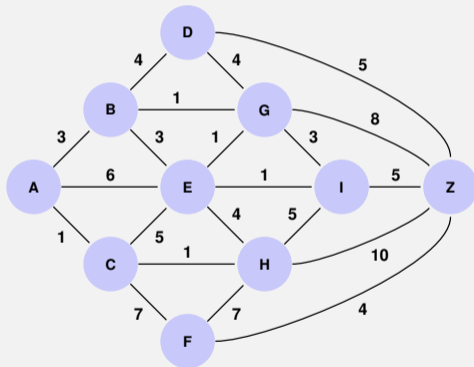
4 Personen am linken Ufer

Das ganze Problem als Graph



Routenfinder

Gegeben Städte A - Z und Distanzen zwischen den Städten.

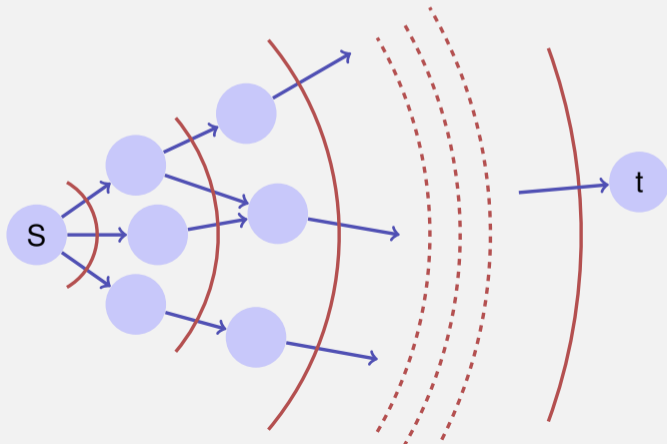


Was ist der kürzeste Weg von A nach Z?

Einfachster Fall

Konstantes Kantengewicht 1 (oBdA)

Lösung: Breitensuche



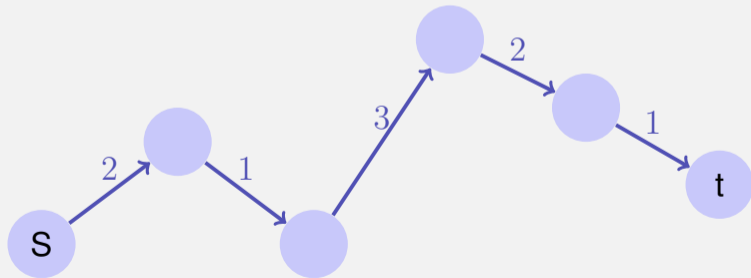
Gewichtete Graphen

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

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

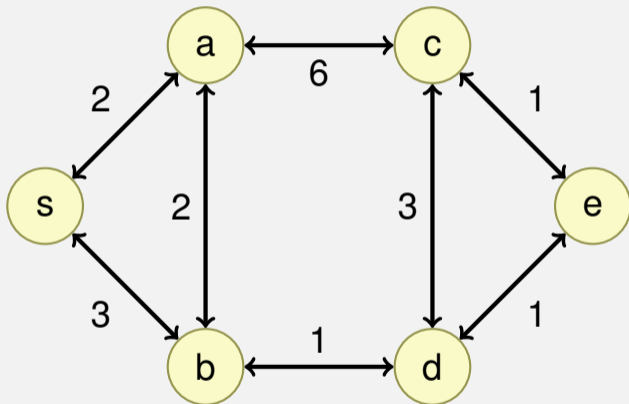
Weg: $p = \langle s = v_0, v_1, \dots, v_k = t \rangle$, $(v_i, v_{i+1}) \in E$ ($0 \leq i < k$)

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



Weg mit Gewicht 9

Annahme (vorerst)



Alle Gewichte von G sind *positiv*.

Kürzeste Wege

Gewicht eines kürzesten Weges von u nach v :

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

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

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

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

Allgemeiner Algorithmus

- 1 Initialisiere d_s und π_s
- 2 Setze $d_s[s] \leftarrow 0$
- 3 Wähle eine Kante $(u, v) \in E$

Relaxiere (u, v) :

if $d_s[v] > d[u] + c(u, v)$ 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 $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$)

Dreiecksungleichung

Für alle $s, u, v \in V$:

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

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 kann nicht länger sein als ein kürzester Weg von s nach v über u .)

Relaxieren ist sicher

Zu jeder Zeit gilt in obigem Algorithmus

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

Relaxieren ist sicher

Zu jeder Zeit gilt in obigem Algorithmus

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

Induktiv:

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

[Dreiecksungleichung].

$$\delta(s, u) \leq d_s[u]$$

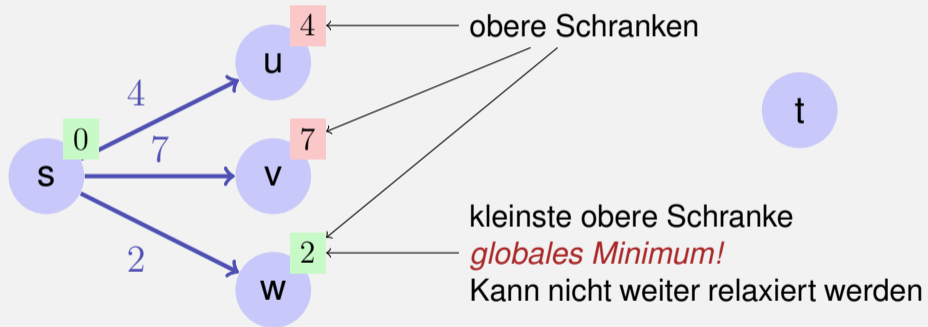
[Induktionsvoraussetzung].

$$\delta(u, v) \leq c(u, v)$$

[Minimalität von δ]

$$\Rightarrow \delta(S, v) \leq d_s[u] + c(u, v)$$

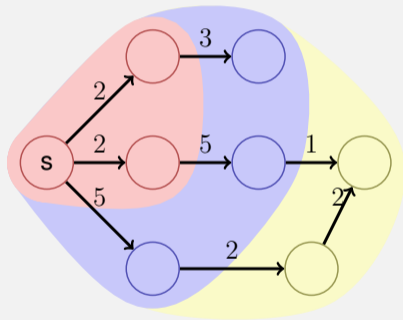
Beobachtung (Dijkstra)



Grundidee

Menge V aller Knoten wird unterteilt in

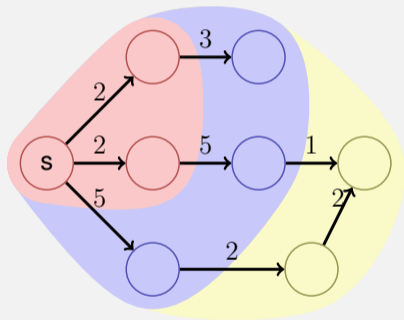
- die Menge M von Knoten, für die schon ein kürzester Weg von s bekannt ist
- die Menge $R = \bigcup_{v \in M} N^+(v) \setminus M$ von Knoten, für die kein kürzester Weg bekannt ist, die jedoch von M direkt erreichbar sind.
- die Menge $U = V \setminus (M \cup R)$ von Knoten, die noch nicht berücksichtigt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



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 gefunden, kann kein Pfad grösseren Gewichts über andere Knoten zu einer Verbesserung führen.

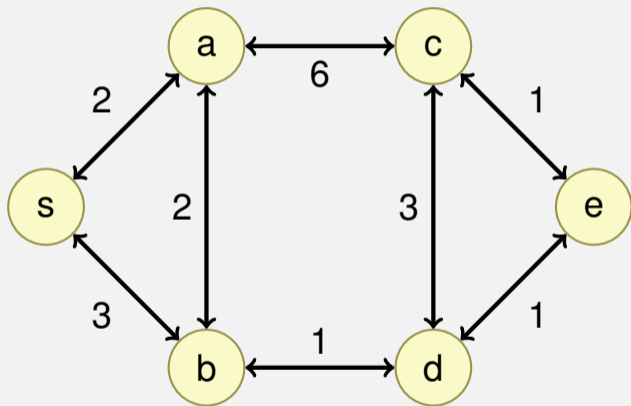


Algorithmus Dijkstra

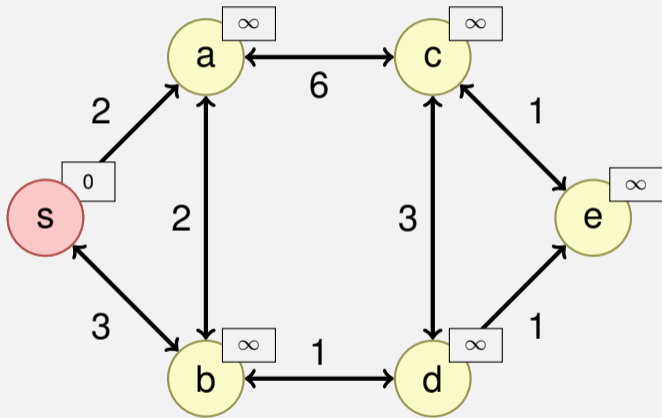
Initial: $PL(n) \leftarrow \infty$ für alle Knoten.

- Setze $PL(s) \leftarrow 0$
- Starte mit $M = \{s\}$. Setze $k \leftarrow s$.
- Solange ein neuer Knoten k hinzukommt und dieser nicht der Zielknoten ist
 - 1 Für jeden Nachbarknoten n von k :
 - Berechne Pfadlänge x nach n über k
 - Wenn $PL(n) = \infty$, so nimm n zu R hinzu
 - Ist $x < PL(n) < \infty$, so setze $PL(n) \leftarrow x$ und passe R an.
 - 2 Wähle als neuen Knoten k den mit kleinster Pfadlänge in R .

Beispiel



Beispiel

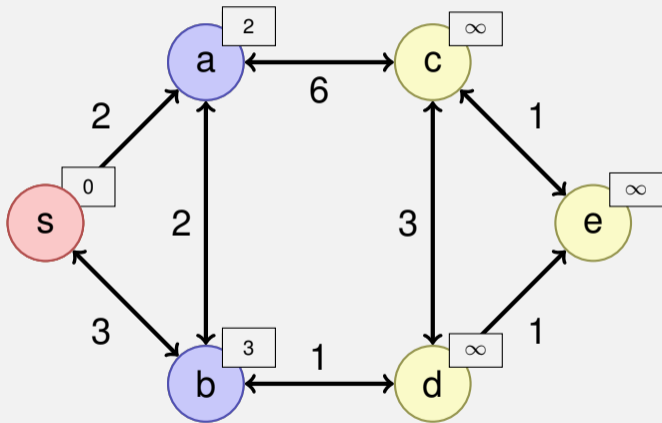


$$M = \{s\}$$

$$R = \{\}$$

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

Beispiel

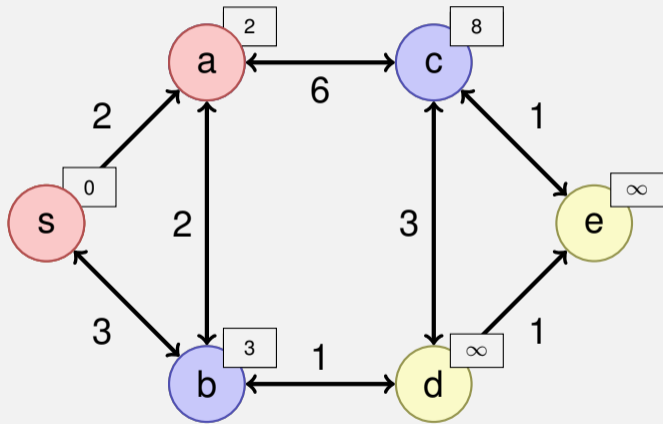


$$M = \{s\}$$

$$R = \{a, b\}$$

$$U = \{c, d, e\}$$

Beispiel

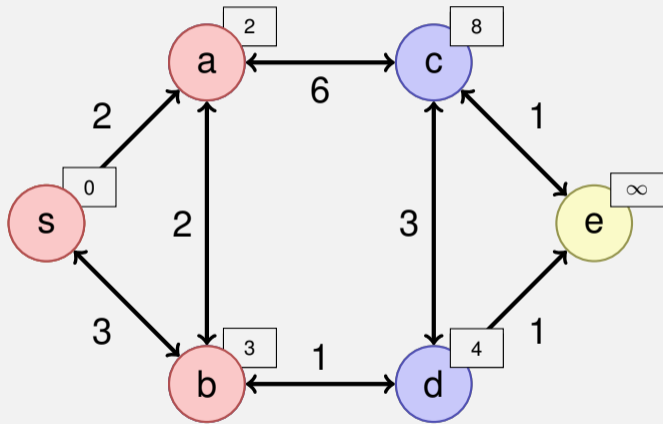


$$M = \{s, a\}$$

$$R = \{b, c\}$$

$$U = \{d, e\}$$

Beispiel

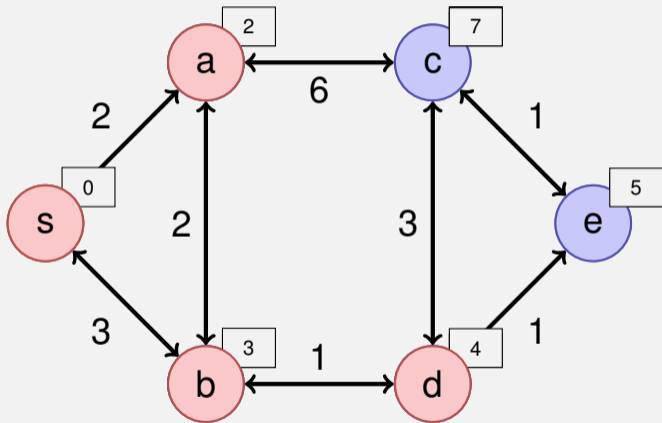


$$M = \{s, a, b\}$$

$$R = \{c, d\}$$

$$U = \{e\}$$

Beispiel

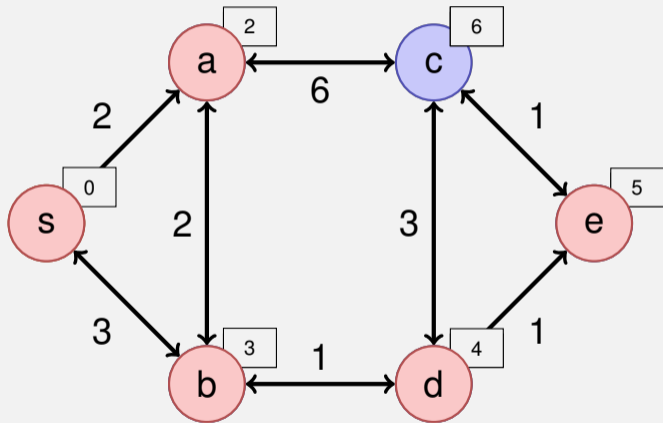


$$M = \{s, a, b, d\}$$

$$R = \{c, e\}$$

$$U = \{\}$$

Beispiel

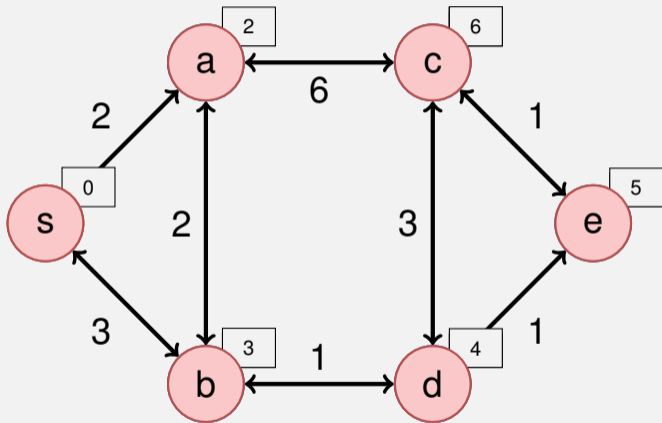


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Beispiel



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

$$R = \{\}$$

$$U = \{\}$$

Zur Implementation: Datenstruktur für R ?

Benötigte Operationen:

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

foreach $v \in N^+(m)$ **do**

if $d(m) + c(m, v) < d(v)$ **then**

$d(v) \leftarrow d(m) + c(m, v)$

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

Zur Implementation: Datenstruktur für R ?

Benötigte Operationen:

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

foreach $v \in N^+(m)$ **do**

if $d(m) + c(m, v) < d(v)$ **then**

$d(v) \leftarrow d(m) + c(m, v)$

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?

DecreaseKey

- DecreaseKey: Aufsteigen im MinHeap in $\mathcal{O}(\log |V|)$
- Position im Heap?
 - Möglichkeit (a): Speichern am Knoten

DecreaseKey

- DecreaseKey: Aufsteigen im MinHeap in $\mathcal{O}(\log |V|)$
- Position im Heap?
 - Möglichkeit (a): Speichern am Knoten
 - Möglichkeit (b): Hashtabelle über Knoten

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 Update-Operation erneut einfügen. Knoten beim Entnehmen als "deleted" kennzeichnen (Lazy Deletion)

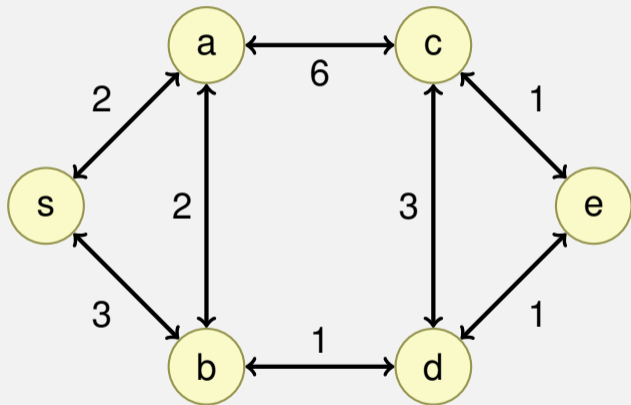
Laufzeit

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

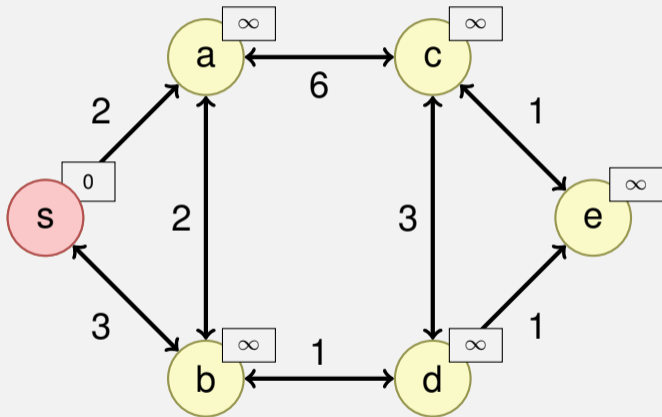
Kürzesten Weg Rekonstruieren

- Beim Updateschritt im obigen Algorithmus jeweils besten Vorgänger merken, an Knoten oder in separater Datenstruktur.
- Besten Pfad rekonstruieren durch Rückwärtslaufen der besten Kanten

Beispiel



Beispiel

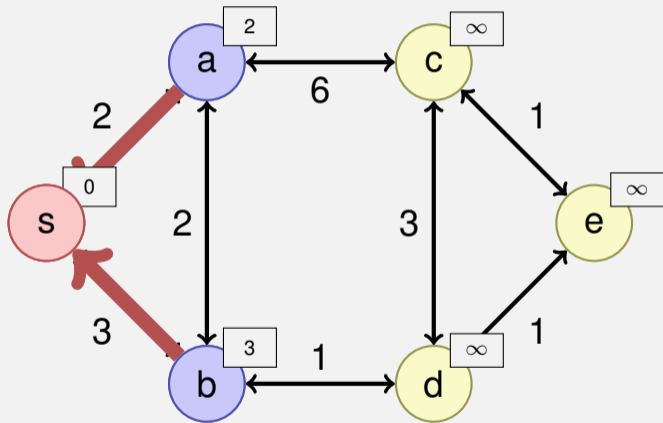


$$M = \{s\}$$

$$R = \{\}$$

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

Beispiel

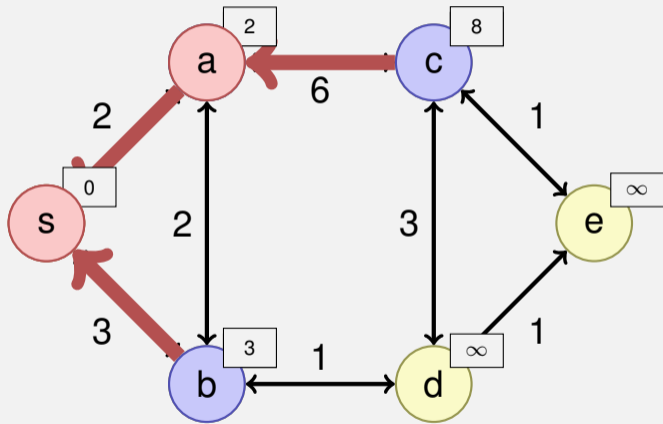


$$M = \{s\}$$

$$R = \{a, b\}$$

$$U = \{c, d, e\}$$

Beispiel

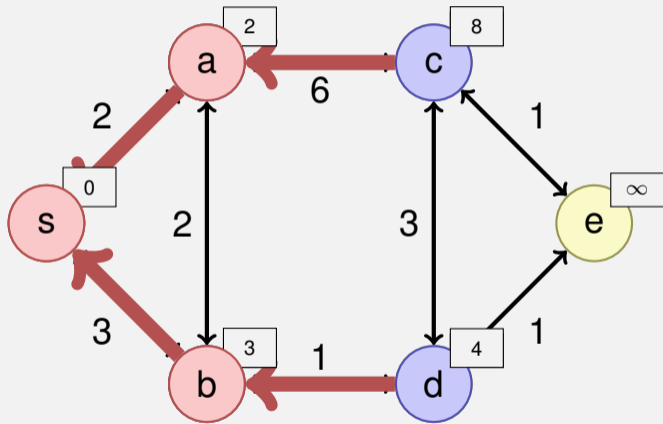


$$M = \{s, a\}$$

$$R = \{b, c\}$$

$$U = \{d, e\}$$

Beispiel

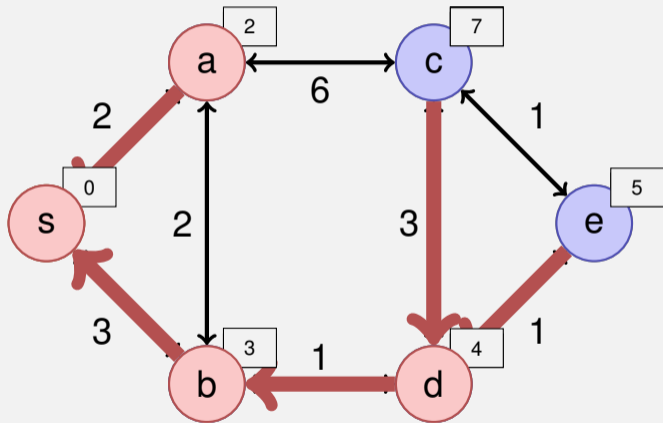


$$M = \{s, a, b\}$$

$$R = \{c, d\}$$

$$U = \{e\}$$

Beispiel

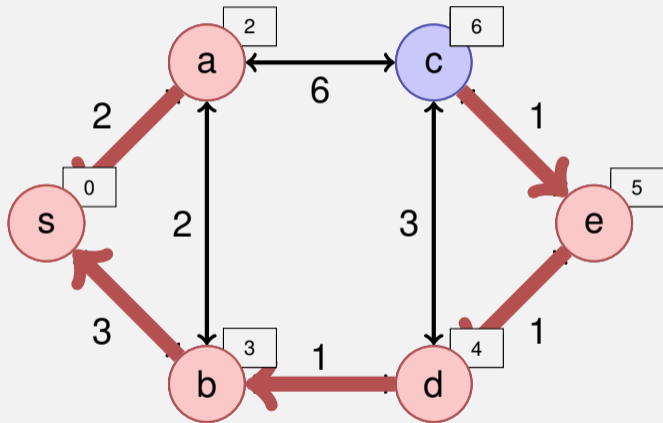


$$M = \{s, a, b, d\}$$

$$R = \{c, e\}$$

$$U = \{\}$$

Beispiel

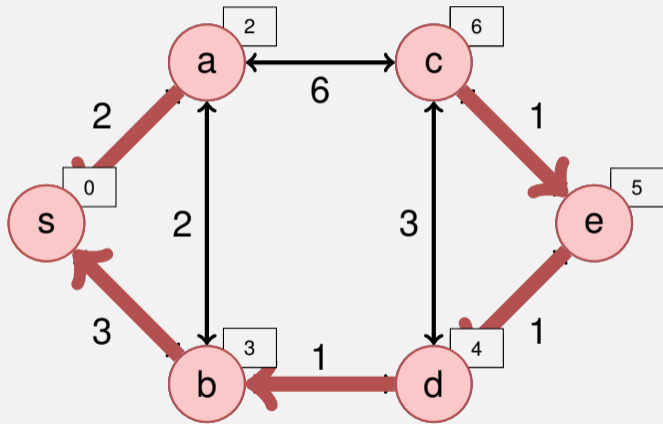


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Beispiel



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

$$R = \{\}$$

$$U = \{\}$$

Allgemeine Bewertete Graphen

Verbesserungsschritt wie bei Dijkstra:

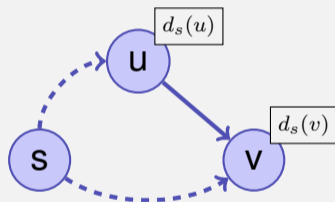
$\text{Relax}(u, v)$ ($u, v \in V, (u, v) \in E$)

if $d_s(v) > d_s(u) + c(u, v)$ **then**

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

return true

return false



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

Beobachtungen

- **Beobachtung 1:** 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 .
Beweis: wäre das nicht so, könnte man einen der Teilpfade kürzen, Widerspruch zur Voraussetzung.
- **Beobachtung 2:** Wenn es einen kürzesten Weg gibt, dann ist dieser einfach, hat also keine doppelten Knoten.
Folgt direkt aus Beobachtung 1.

Dynamic Programming Ansatz (Bellman)

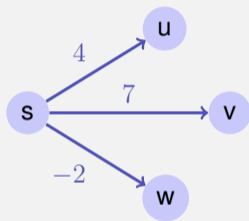
Induktion über Anzahl Kanten. $d_s[i, v]$: Kürzeste Weglänge von s nach v über maximal i Kanten.

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

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

Dynamic Programming Ansatz (Bellman)

	s	\dots	v	\dots	w
0	0	∞	∞	∞	∞
1	0	∞	7	∞	-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.

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.

$d(v) \leftarrow \infty \forall v \in V; d(s) \leftarrow 0$

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;

Laufzeit $\mathcal{O}(|E| \cdot |V|)$.