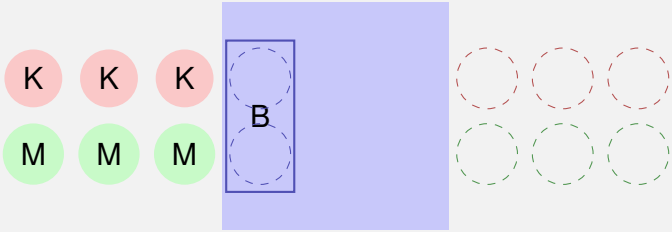
23. Kürzeste Wege

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

[Ottman/Widmayer, Kap. 9.5 Cormen et al, Kap. 24.1-24.3, 25.2-25.3]

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? [32]



[32] 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

	links	rechts
Missionare	3	0
Kannibalen	3	0
Boot	x	

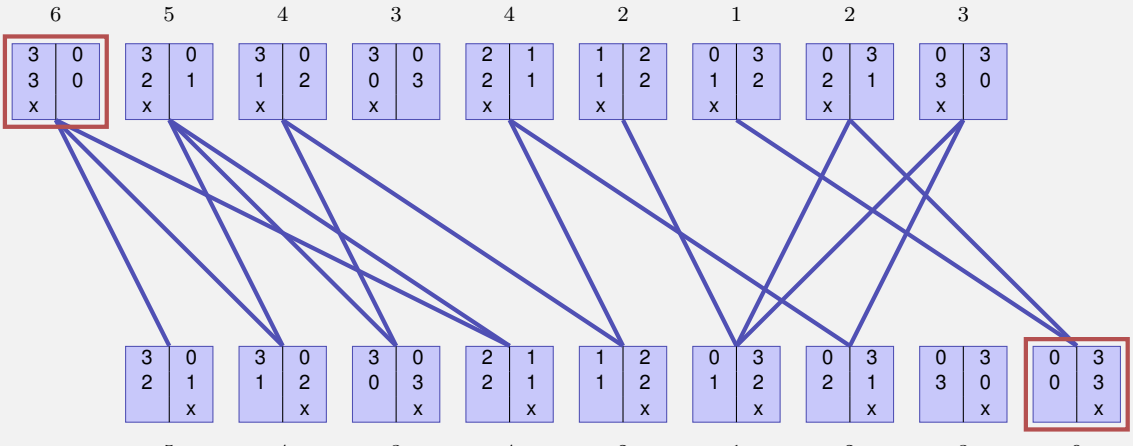
6 Personen am linken Ufer

Überfahrt möglich

	links	rechts
Missionare	2	1
Kannibalen	2	1
Boot		x

4 Personen am linken Ufer

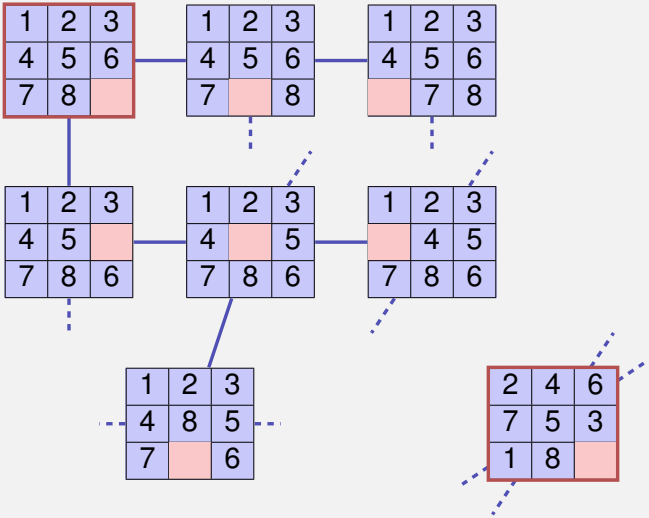
Das ganze Problem als Graph



Beispiel Schiebepuzzle

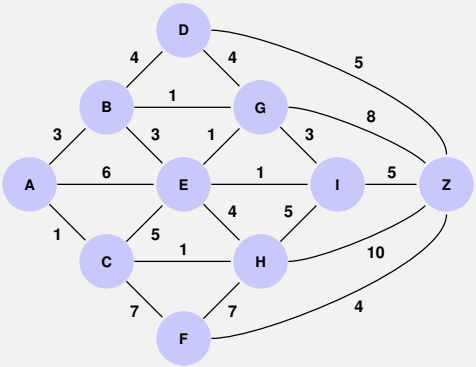
Wollen die schnelleste Lösung finden für

Problem als Graph



Routenfinder

Gegeben Städte A - Z und Distanzen zwischen den Städten.

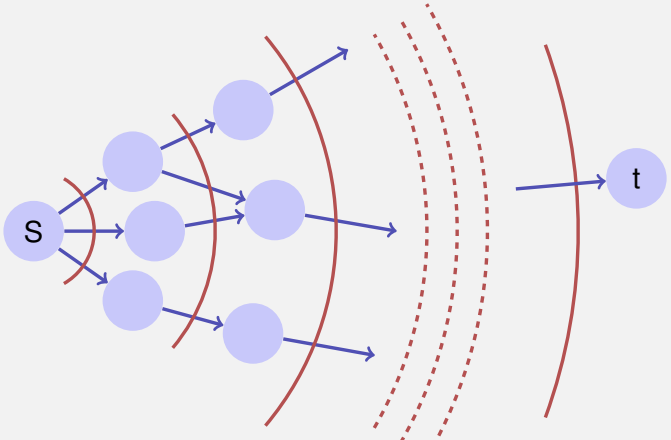


Was ist der kürzeste Weg von A nach Z?

Einfachster Fall

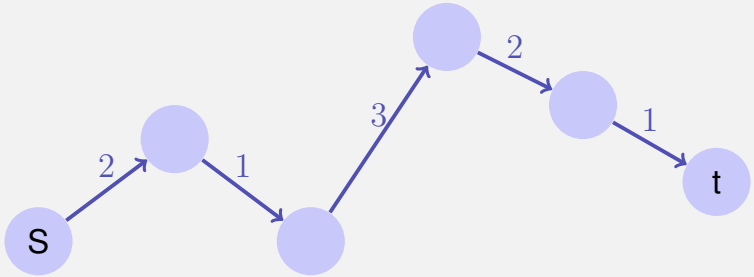
Konstantes Kantengewicht 1 (oBdA)

Lösung: Breitensuche



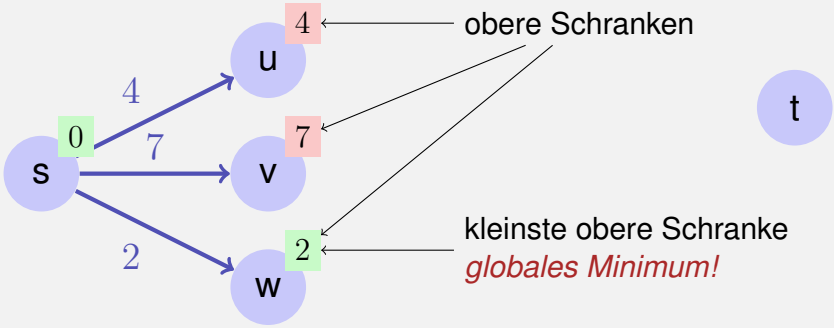
Positiv gewichtete Graphen

Gegeben: $G = (V, E, c)$, $c : E \to \mathbb{R}^+$, $s, t \in V$.
Gesucht: Länge eines kürzesten Weges (Gewicht) von s nach t.
Weg: $\langle s = v_0, v_1, \ldots, v_k = t \rangle$, $(v_i, v_{i+1}) \in E$ $(0 \leq i < k)$
Gewicht: $\sum_{i=0}^{k-1} c((v_i, v_{i+1}))$.



Weg mit Gewicht 9

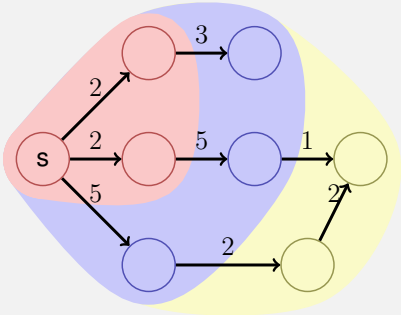
Beobachtung



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.

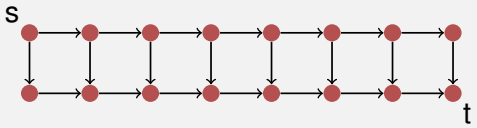


Existenz eines kürzesten Weges

Annahme: Es existiert ein Weg von s nach t in G
Behauptung: Es gibt einen kürzesten Weg s nach t in G

Beweis: Es kann zwar unendlich viele Wege von s nach t geben. Da aber c positiv ist, ist ein kürzester Weg zyklusfrei. Damit ist die Maximallänge eines kürzesten Weges durch ein $n \in N$ beschränkt und es gibt nur endlich viele Kandidaten für kürzeste Wege.

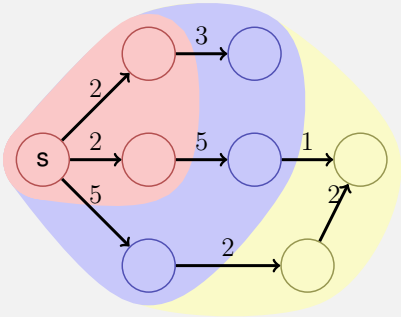
Bemerkung: es kann exponentiell viele Wege geben. Beispiel



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.

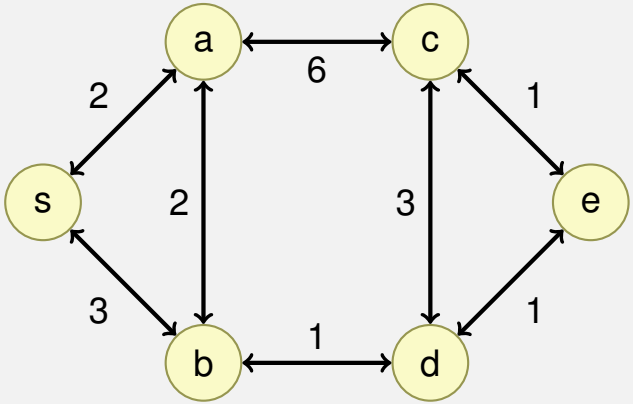


Algorithmus Dijkstra(G, s)

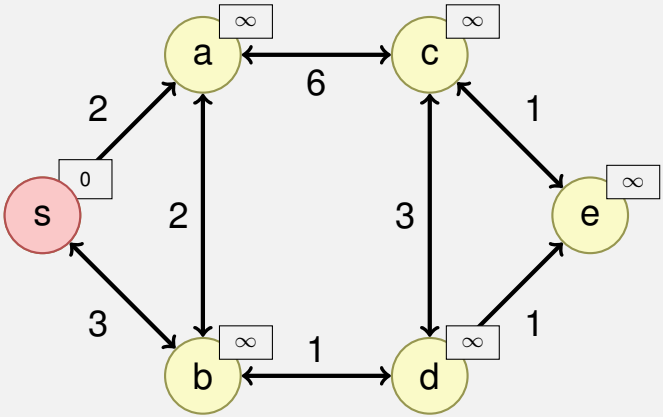
Input : Positiv gewichteter Graph $G = (V, E, c)$, Startpunkt $s \in V$
Output : Minimale Gewichte d der kürzesten Pfade.

$M = \{s\}$; $R = N^+(s)$, $U = V \setminus R$
$d(s) \leftarrow 0$; $d(u) \leftarrow \infty \ \forall u \neq s$
while $R \neq \emptyset$ **do**
    $r \leftarrow \arg\min_{r \in R} \min_{m \in N^-(r) \cap M} d(m) + c(m, r)$
    $d(r) \leftarrow \min_{m \in N^-(r) \cap M} d(m) + c(m, r)$
    $M \leftarrow M \cup \{r\}$
    $R \leftarrow R - \{r\} \cup N^+(r) \setminus M$
return d

Beispiel



Beispiel

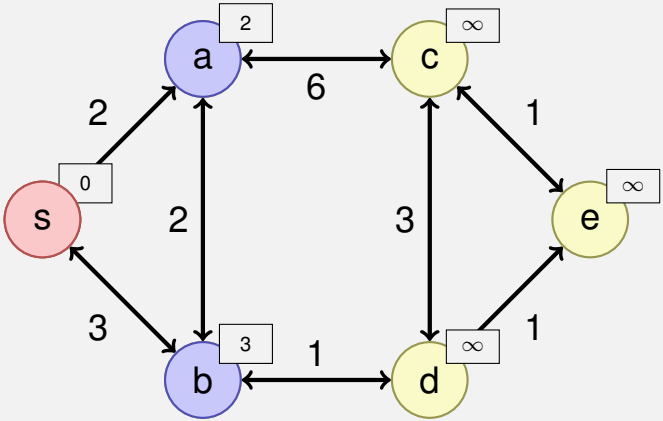


$$M = \{s\}$$

$$R = \{\}$$

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

Beispiel

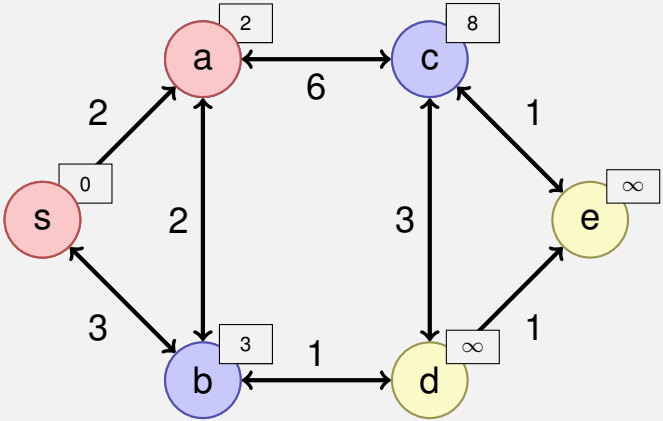


$$M = \{s\}$$

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

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

Beispiel

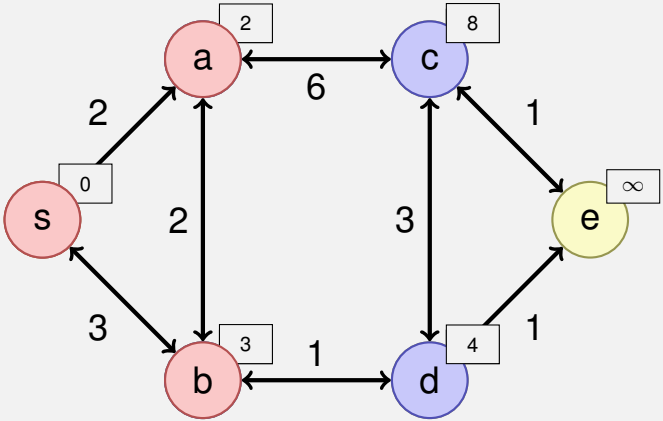


$$M = \{s, a\}$$

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

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

Beispiel

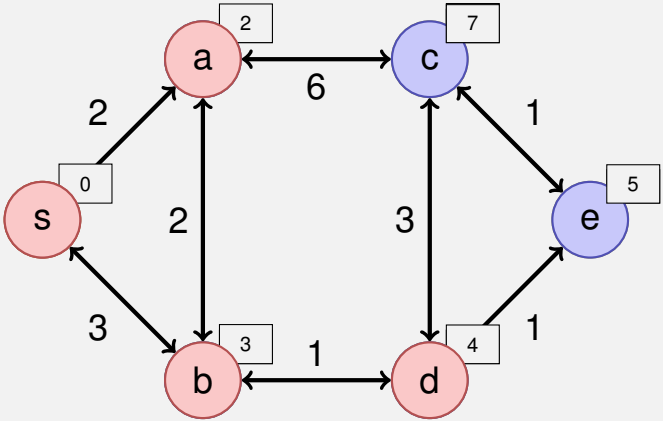


$$M = \{s, a, b\}$$

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

$$U = \{e\}$$

Beispiel

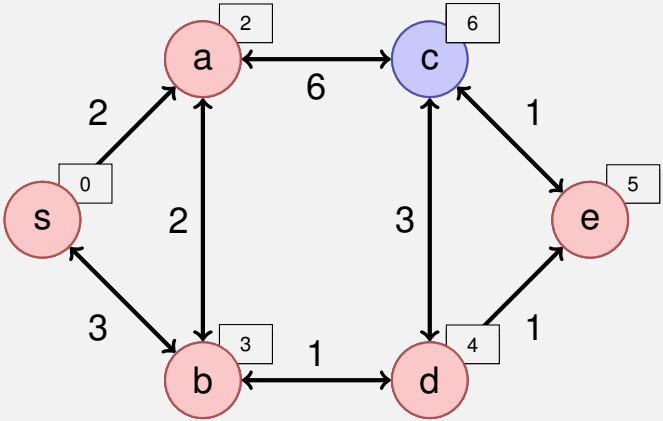


$$M = \{s, a, b, d\}$$

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

$$U = \{\}$$

Beispiel

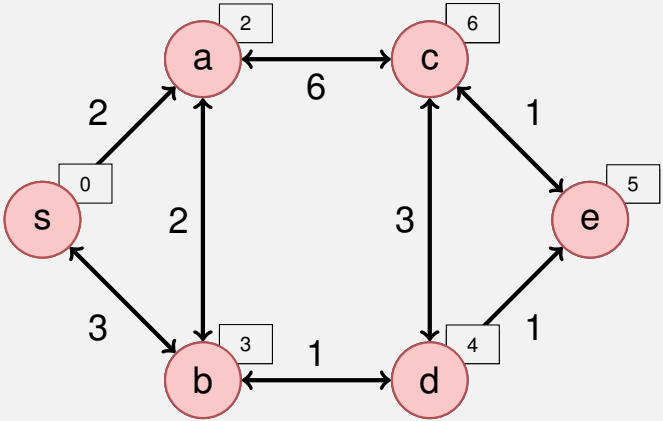


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Beispiel



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

$$R = \{\}$$

$$U = \{\}$$

Zur Implementation: Naive Variante

- Minimum finden: Alle Kanten (u, v) für $u \in M, v \in R$ durchlaufen.
- Gesamtkosten: $\mathcal{O}(|V| \cdot |E|)$

Zur Implementation: Bessere Variante

- Update aller ausgehenden Kanten beim Einfügen eines neuen w in M:

  foreach $(w, v) \in E$ **do**
  - **if** $d(w) + c(w, v) < d(v)$ **then**
    - $d(v) \leftarrow d(w) + c(w, v)$

- Updatekosten: $\mathcal{O}(|E|)$, Minima finden: $\mathcal{O}(|V|^2)$, also Gesamtkosten $\mathcal{O}(|V|^2)$

Zur Implementation: Datenstruktur für R?

Benötigte Operationen:

- ExtractMin (über R)
- DecreaseKey (Update in R)

```
foreach (m, v) ∈ E do
    if d(m) + c(m, v) < d(v) then
        d(v) ← d(m) + c(m, v)
        if v ∈ R then
            DecreaseKey(R, v)          // Update eines d(v) im Heap zu R
        else
            R ← R ∪ {v}                // Einfügen eines neuen d(v) im Heap zu R
```

- Heap Datenstruktur bietet sich an. Problem: Unklar, wo v in R steht (für DecreaseKey).

DecreaseKey

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

Laufzeit

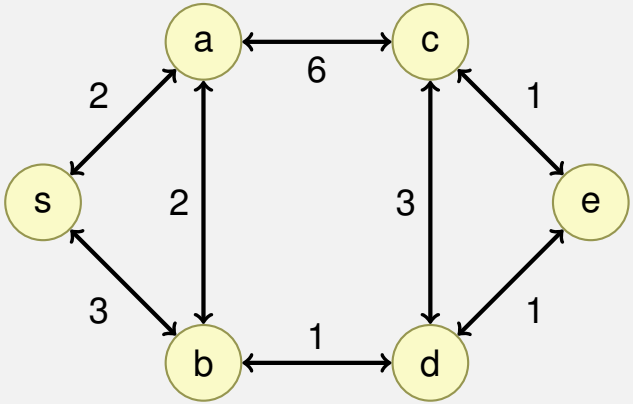
- $|V|\times$ ExtractMin: $\mathcal{O}(|V| \log |V|)$
- $|E|\times$ Insert oder DecreaseKey: $\mathcal{O}(|E| \log |V|)$
- $1\times$ Init: $\mathcal{O}(|V|)$
- Insgesamt: $\mathcal{O}(|E| \log |V|)$.

Kann verbessert werden unter Verwendung einer für ExtractMin und DecreaseKey optimierten Datenstruktur (Fibonacci Heap), dann Laufzeit $\mathcal{O}(|E| + |V| \log |V|)$.

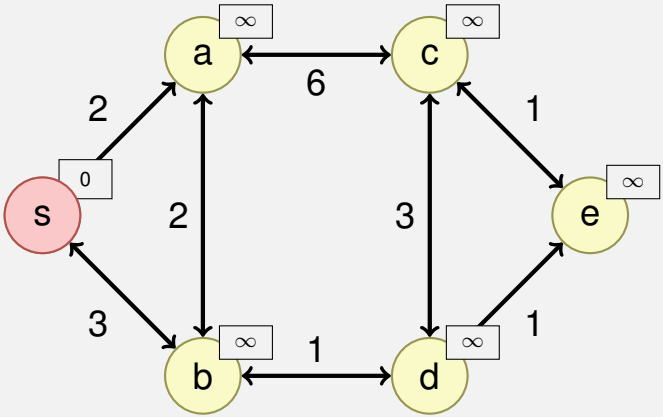
Kürzesten Weg Rekonstruieren

- 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

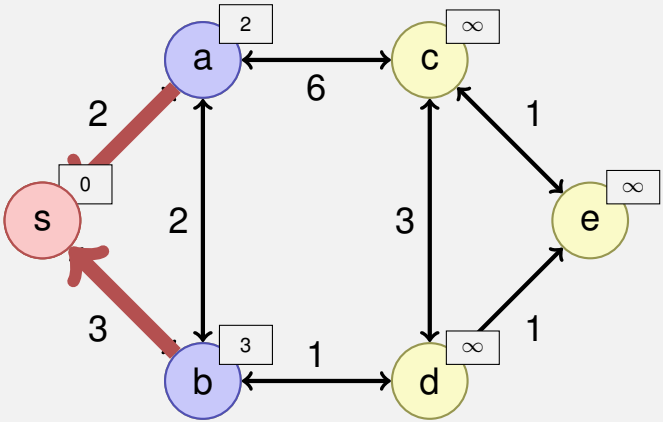


$$M = \{s\}$$

$$R = \{\}$$

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

Beispiel

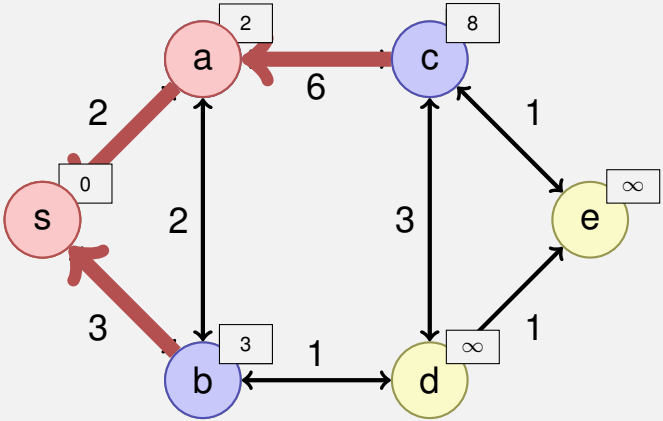


$$M = \{s\}$$

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

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

Beispiel

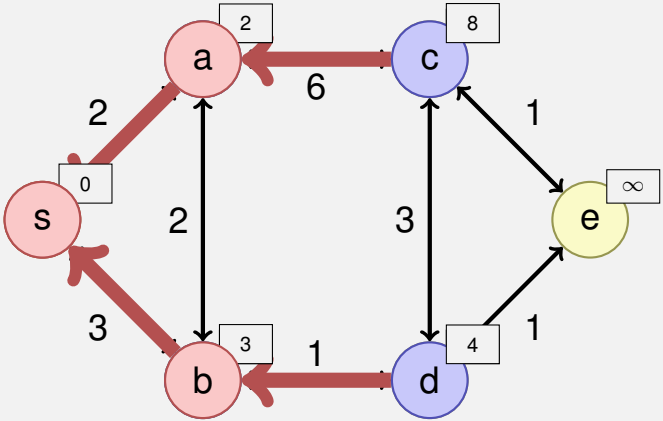


$$M = \{s, a\}$$

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

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

Beispiel

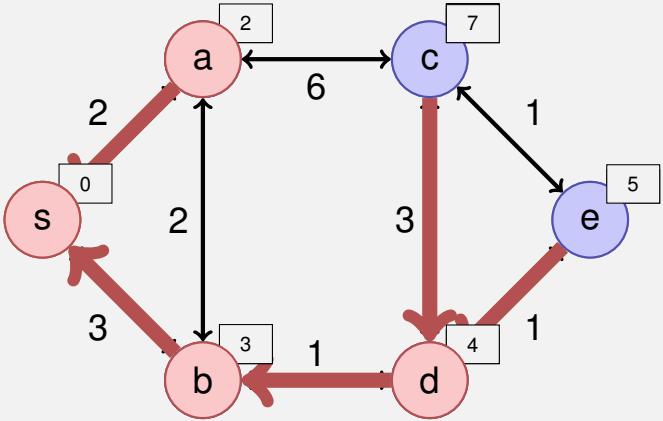


$$M = \{s, a, b\}$$

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

$$U = \{e\}$$

Beispiel

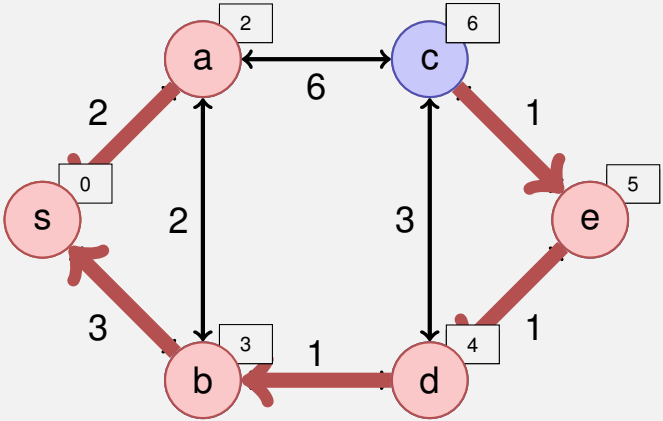


$$M = \{s, a, b, d\}$$

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

$$U = \{\}$$

Beispiel

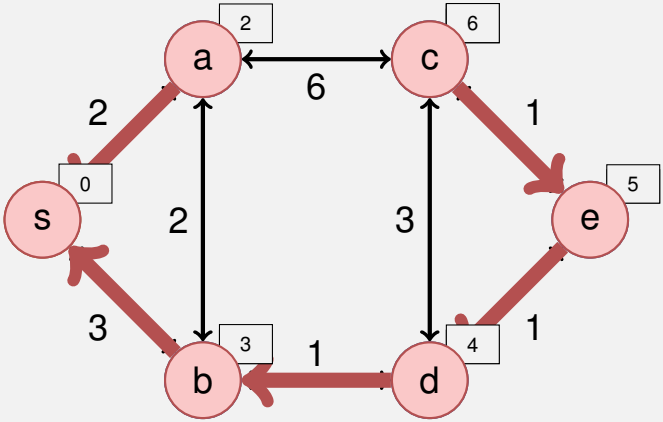


$$M = \{s, a, b, d, e\}$$

$$R = \{c\}$$

$$U = \{\}$$

Beispiel



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

$$R = \{\}$$

$$U = \{\}$$

Allgemeine Bewertete Graphen

Relaxieren geht genauso:

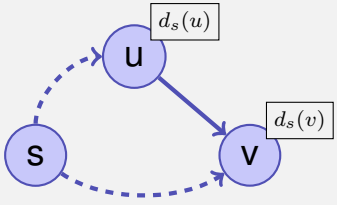
Relax(u, v) $(u, v \in V, (u, v) \in E)$

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 $p = \langle v_0, \ldots, v_k \rangle$ ein kürzester Pfad von v_0 nach v_k. Dann ist jeder der Teilpfade $p_{ij} = \langle v_i, \ldots, v_j \rangle$ $(0 \leq i < j \leq k)$ 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)

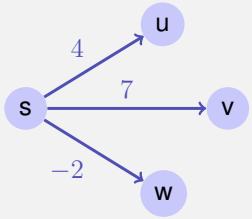
Induktion über Anzahl Kanten. $d_s[i, v]$: Kürzeste Weglänge von s nach v über maximal i Kanten.

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

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

Dynamic Programming Ansatz (Bellman)

	s	$\cdots$	v	$\cdots$	w
0	0	∞	∞	∞	∞
1	0	∞	7	∞	-2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n-1$	0	$\cdots$	$\cdots$	$\cdots$	$\cdots$



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**
    $f \leftarrow$ false
    foreach $(u, v) \in E$ **do**
        $f \leftarrow f \vee \mathrm{Relax}(u, v)$
    if $f =$ false **then return** true
return false;

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

Alle kürzesten Pfade

Ziel: Berechne das Gewicht eines kürzesten Pfades für jedes Knotenpaar.

- $|V|\times$ Anwendung von Dijkstras ShortestPath: $\mathcal{O}(|V| \cdot |E| \cdot \log |V|)$ (Mit Fibonacci-Heap:: $\mathcal{O}(|V|^2 \log |V| + |V| \cdot |E|)$)
- $|V|\times$ Anwendung von Bellman-Ford: $\mathcal{O}(|E| \cdot |V|^2)$
- Es geht besser!

Induktion über Knotennummer.[33]

Betrachte die Gewichte aller kürzesten Wege S^k mit Zwischenknoten in $V^k := \{v_1, \ldots, v_k\}$, wenn Gewichte zu allen kürzesten Wegen S^{k-1} mit Zwischenknoten in V^{k-1} gegeben sind.

- v_k kein Zwischenknoten eines kürzesten Pfades von $v_i \rightsquigarrow v_j$ in V^k: Gewicht eines kürzesten Pfades $v_i \rightsquigarrow v_j$ in S^{k-1} dann auch das Gewicht eines kürzesten Pfades in S^k.
- v_k Zwischenknoten eines kürzesten Pfades $v_i \rightsquigarrow v_j$ in V^k: Teilpfade $v_i \rightsquigarrow v_k$ und $v_k \rightsquigarrow v_j$ enthalten nur Zwischenknoten aus S^{k-1}.

[33] wie beim Algorithmus für die reflexive transitive Hülle von Warshall

DP Induktion

$d^k(u, v)$ = Minimales Gewicht eines Pfades $u \rightsquigarrow v$ mit Zwischenknoten aus V^k

Induktion

$$d^k(u, v) = \min\{d^{k-1}(u, v), d^{k-1}(u, k) + d^{k-1}(k, v)\} (k \geq 1)$$
$$d^0(u, v) = c(u, v)$$

DP Algorithmus Floyd-Warshall(G)

Input : Azyklischer Graph $G = (V, E, c)$
Output : Minimale Gewichte aller Pfade d

$d^0 \leftarrow c$
for $k \leftarrow 1$ **to** $|V|$ **do**
  for $i \leftarrow 1$ **to** $|V|$ **do**
    for $j \leftarrow 1$ **to** $|V|$ **do**
      $d^k(v_i, v_j) = \min\{d^{k-1}(v_i, v_j), d^{k-1}(v_i, v_k) + d^{k-1}(v_k, v_j)\}$

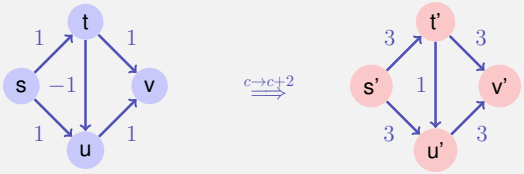
Laufzeit: $\Theta(|V|^3)$

Bemerkung: Der Algorithmus kann auf einer einzigen Matrix d (in place) ausgeführt werden.

Umgewichtung

Idee: Anwendung von Dijkstras Algorithmus auf Graphen mit negativen Gewichten durch Umgewichtung

Das folgende geht *nicht*. Die Graphen sind nicht äquivalent im Sinne der kürzesten Pfade.



Umgewichtung

Andere Idee: “Potentialfunktion” (Höhe) auf den Knoten

- $G = (V, E, c)$ ein gewichteter Graph.
- Funktion $h : V \to \mathbb{R}$
- Neue Gewichte

$$\tilde{c}(u, v) = c(u, v) + h(u) - h(v),\ (u, v \in V)$$

Umgewichtung

Beobachtung: Ein Pfad p ist genau dann kürzester Pfad in $G = (V, E, c)$, wenn er in $\tilde{G} = (V, E, \tilde{c})$ kürzester Pfad ist.

$$\tilde{c}(p) = \sum_{i=1}^{k} \tilde{c}(v_{i-1}, v_i) = \sum_{i=1}^{k} c(v_{i-1}, v_i) + h(v_{i-1}) - h(v_i)$$

$$= h(v_0) - h(v_k) + \sum_{i=1}^{k} c(v_{i-1}, v_i) = c(p) + h(v_0) - h(v_k)$$

Also $\tilde{c}(p)$ minimal unter allen $v_0 \rightsquigarrow v_k \iff c(p)$ minimal unter allen $v_0 \rightsquigarrow v_k$.

Zyklengewichte sind invariant: $\tilde{c}(v_0, \ldots, v_k = v_0) = c(v_0, \ldots, v_k = v_0)$

Johnsons Algorithmus

Hinzunahme eines neuen Knotens $s \notin V$:

$$
\begin{aligned}
G' &= (V', E', c') \\
V' &= V \cup \{s\} \\
E' &= E \cup \{(s, v) : v \in V\} \\
c'(u, v) &= c(u, v),\ u \neq s \\
c'(s, v) &= 0 (v \in V)
\end{aligned}
$$

Johnsons Algorithmus

Falls keine negativen Zyklen: wähle für Höhenfunktion Gewicht der kürzesten Pfade von s,

$$h(v) = d(s, v).$$

Für minimales Gewicht d eines Pfades gilt generell folgende Dreiecksungleichung:

$$d(s, v) \leq d(s, u) + c(u, v).$$

Einsetzen ergibt $h(v) \leq h(u) + c(u, v)$. Damit

$$\tilde{c}(u, v) = c(u, v) + h(u) - h(v) \geq 0.$$

Algorithmus Johnson(G)

Input : Gewichteter Graph $G = (V, E, c)$
Output : Minimale Gewichte aller Pfade D.

Neuer Knoten s. Berechne $G' = (V', E', c')$
if BellmanFord(G', s) = false **then** return "graph has negative cycles"
foreach $v \in V'$ **do**
    $h(v) \leftarrow d(s, v)$ // d aus BellmanFord Algorithmus
foreach $(u, v) \in E'$ **do**
    $\tilde{c}(u, v) \leftarrow c(u, v) + h(u) - h(v)$
foreach $u \in V$ **do**
    $\tilde{d}(u, \cdot) \leftarrow$ Dijkstra$(\tilde{G}', u)$
    foreach $v \in V$ **do**
        $D(u, v) \leftarrow \tilde{d}(u, v) + h(v) - h(u)$

Analyse

Laufzeiten

- Berechnung von G': $\mathcal{O}(|V|)$
- Bellman Ford G': $\mathcal{O}(|V| \cdot |E|)$
- $|V|\times$ Dijkstra $\mathcal{O}(|V| \cdot |E| \cdot \log |V|)$
  (Mit Fibonacci-Heap:: $\mathcal{O}(|V|^2 \log |V| + |V| \cdot |E|)$)

Insgesamt $\mathcal{O}(|V| \cdot |E| \cdot \log |V|)$
($\mathcal{O}(|V|^2 \log |V| + |V| \cdot |E|)$)