

Die Aussagen gelten nach ①.

Für den Induktionsschritt müssen wir also zeigen, dass ③ und ④ die Gültigkeit von (a), (b), (c) nicht beeinträchtigen. Sei v ein in ② ausgewählter Knoten.

Für (a):

Für beliebiges $x \in R$ und $y \in V(G) \setminus R$

gilt $l(x) \leq l(v) \leq l(y)$ wegen (a) und der Wahl von v in ②.

Dann gilt (a) weiter nach ③ und nach ④.

Für (b):

Nur z.z.: Aussage gilt nach ③ auch für v .

Also z.z.: Kein s - v -Pfad in G , ~~welcher~~ der einen Knoten in $V(G) \setminus R$ enthält, kann kürzer sein als $l(v)$.

Betrachte einen derartigen Pfad, der $w \in V(G) \setminus R$ enthält; sei w der erste Knoten auf dem Weg von s nach v . Da vor ③ (c) gilt, ist $l(w) \leq c(P_{[s,w]})$. Da die Kantengewichte nicht negativ sind, gilt

$$c(P_{[s,w]}) \leq c(P) < l(v)$$

also $l(w) < l(v)$, im Widerspruch zur Wahl von v in ②.

Für (c):

Wenn für ein w $p(w)$ auf v
 und $l(w)$ auf $l(v) + c((v,w))$ gesetzt wird,
 dann gibt es einen s-w-Pfad in $G[R \cup \{w\}]$
 der Länge $l(v) + c((v,w))$ mit letzter Kante (v,w) ,
 denn (c) war gültig für v .

Nehmen wir an, dass nach (3) und (4) ein
 s-w-Pfad P in $G[R \cup \{w\}]$ existiert,
 der kürzer ist als $l(w)$ für ein $w \in V(G) \setminus R$.

Dieser muss v enthalten - den einzigen hinzugekommenen Knoten,
 sonst wäre (c) bereits vor (3) und (4) verletzt.

Sei x der Nachbar von w in P . Da $x \in R$ gilt,
 muss wegen (a) $l(x) \leq l(v)$ gelten; wegen der Wahl in (4)
 gilt $l(w) \leq l(x) + c((x,w))$. Also ist

$$l(w) \leq l(x) + c((x,w)) \leq l(v) + c((x,w)) \leq c(P),$$

denn nach (b) ist $l(v)$ die Länge eines kürzesten
 s-v-Pfades, und P enthält einen s-v-Pfad und die Kante (x,w) .
 $l(w) \leq c(P)$ ist ein Widerspruch zur Annahme.

Damit gelten die Aussagen (a), (b), (c) immer nach (3) und (4)
 also auch am Ende.

Laufzeit: n Iteration zu $O(n)$ □

Was braucht man für eine Implementation?

Folgende Operationen:

- find-min (Minimum finden)
- union / insert (Mengen erweitern)
- delete (Mengen verkleinern)
- decrease-key (Distanzen ändern)
- is-empty (Leere bzw. Gleichheit überprüfen)

Bei „einfachen“ Datenstrukturen benötigt eine dieser Operationen Zeit $O(n)$.

Es gibt eine „effizientere“ Datenstruktur, für die der durchschnittliche Aufwand $O(\log n)$ ist:

Fredman und Tarjan (1987): Fibonacci-Heaps

≈ „Heaps“

Liefert Laufzeit $O(n \log n + m)$

Mehr dazu in der nächsten Vorlesung!

3.2.2 Algorithmus von Moore-Bellman-Ford

| | | |
|------|------|------|
| ↑ | ↑ | ↑ |
| 1959 | 1958 | 1956 |

Algorithmus 3.5

Eingabe: Digraph G , konservative Gewichte $c: E(G) \rightarrow \mathbb{R}$
 Knoten $s \in V(G)$.

Ausgabe: ~~Kürzeste Wege~~ $l(v)$ von s zu v .

Für jeden Knoten $v \in V(G)$ die Angaben

$l(v)$: Länge eines kürzesten s - v -Pfad

$p(v)$: Vorgänger von v in einem kürzesten s - v -Pfad.

(Falls v nicht erreichbar ist, dann gilt $l(v) = \infty$, $p(v) = \text{NIL}$.)

(1) Setze $l(s) := 0$
 $l(v) := \infty$ für alle $v \in V(G) \setminus \{s\}$

(2) FOR $i := 1$ TO $n-1$ DO
 FOR (jede Kante $(v, w) \in E(G)$) DO
 IF $(l(w) > l(v) + c(v, w))$ THEN
 $l(w) := l(v) + c(v, w)$,
 $p(w) := v$