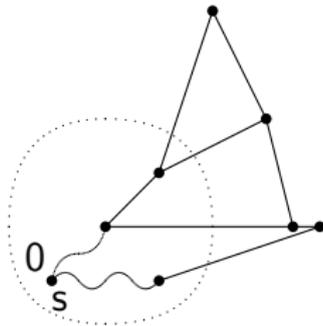


## 2. Das single-source-shortest-path-Problem

Zunächst nehmen wir an, dass  $d \geq 0$  ist. Alle kürzesten Pfade von  $a$  nach  $b$  sind o.B.d.A. einfache Pfade.

### 2.1 Dijkstra's Algorithmus

**Gegeben:**  $G = (V, A)$ , ( $A = V \times V$ ),  
Distanzfunktion  $d : A \rightarrow \mathbb{R}^+ \cup \{+\infty\}$ ,  
Startknoten  $s$ ,  $G$  durch Adjazenzlisten dargestellt.



**algorithm** sssp:=

$S := \{s\}$ ;  $\text{dis}[s] := 0$ ; initialisiere eine Priority Queue  $PQ$ , die alle Knoten  $v \in V \setminus \{s\}$  enthält mit Schlüssel  $\text{dis}[v] := d(s, v)$

**for** alle  $v \in V - \{s\}$  **do**  $\text{from}[v] := s$  **od**

**while**  $S \neq V$  **do**

$v := \text{ExtractMin}(PQ)$

$S := S \cup \{v\}$

**for** alle  $w \in V \setminus S$ ,  $d(v, w) < \infty$  **do**

**if**  $\text{dis}[v] + d(v, w) < \text{dis}[w]$  **then**

$\text{DecreaseKey}(w, \text{dis}[v] + d(v, w))$

**co**  $\text{DecreaseKey}$  aktualisiert  $\text{dis}[w]$  **oc**

$\text{from}[w] := v$

**fi**

**od**

**od**

Seien  $n = |V|$  und  $m =$  die Anzahl der wirklichen Kanten in  $G$ .  
**Laufzeit (mit Fibonacci-Heaps):**

Initialisierung:	$\mathcal{O}(n)$
<i>ExtractMin</i> :	$n \cdot \mathcal{O}(\log n)$
Sonstiger Aufwand:	$m \cdot \mathcal{O}(1)$ (z.B. <i>DecreaseKey</i> )

⇒ Zeitbedarf also:  $\mathcal{O}(m + n \log n)$

**Korrektheit:** Wir behaupten, dass in dem Moment, in dem ein  $v \in V \setminus \{s\}$  Ergebnis der *ExtractMin* Operation ist, der Wert  $\text{dis}[v]$  des Schlüssels von  $v$  gleich der Länge eines kürzesten Pfades von  $s$  nach  $v$  ist.

**Beweis:**

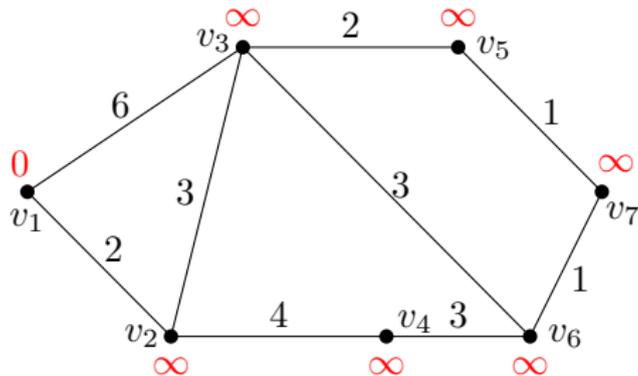
[durch Widerspruch] Sei  $v \in V \setminus \{s\}$  der erste Knoten, für den diese Behauptung nicht stimmt, und sei



ein kürzester Pfad von  $s$  nach  $v$ , mit einer Länge  $< \text{dis}[v]$ . Dabei sind  $s_1, \dots, s_r \in S$ ,  $v_1 \notin S$  [ $r = 0$  und/oder  $q = 0$  ist möglich].

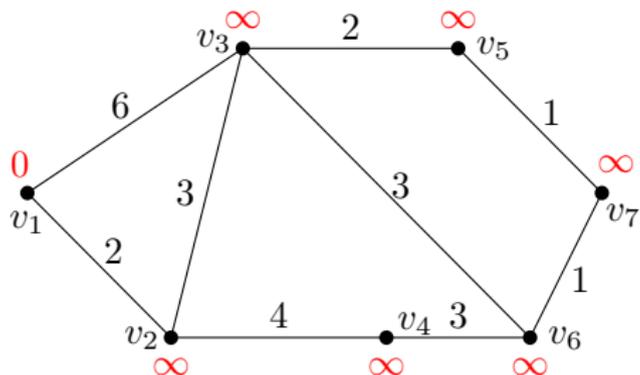
Betrachte den Pfad  $s \xrightarrow{s_1} \dots \xrightarrow{s_r} v_1$ ; seine Länge ist  $< \text{dis}[v]$ , für  $q \geq 1$  (ebenso für  $q = 0$ ) ist also  $\text{dis}[v_1] < \text{dis}[v]$ , im Widerspruch zur Wahl von  $v$ . □

## Beispiel 108 (Dijkstra's Algorithmus)

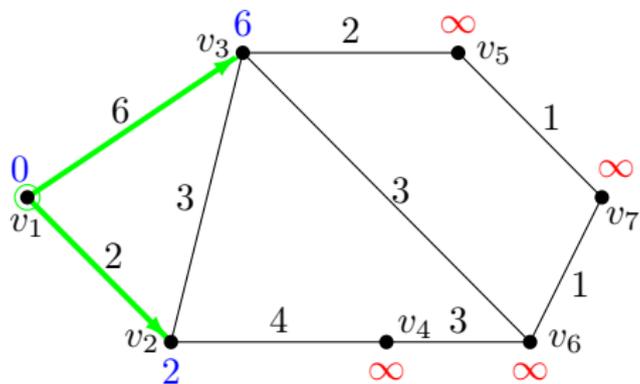


gegeben Graph  $G$ ;  
 $v_1$  ist der Startknoten;  
setze  $v_1$  als Bezugsknoten;  
setze  $\text{dis}[v_1] = 0$ ;  
setze  $\text{dis}[\text{Rest}] = +\infty$ ;

## Beispiel 108 (Dijkstra's Algorithmus)

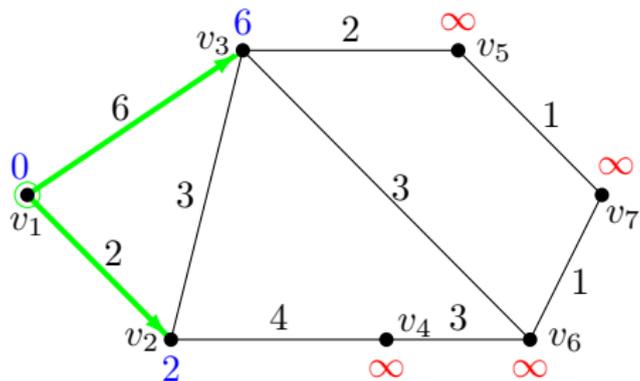


gegeben Graph  $G$ ;  
 $v_1$  ist der Startknoten;  
setze  $v_1$  als Bezugsknoten;  
setze  $dis[v_1] = 0$ ;  
setze  $dis[Rest] = +\infty$ ;

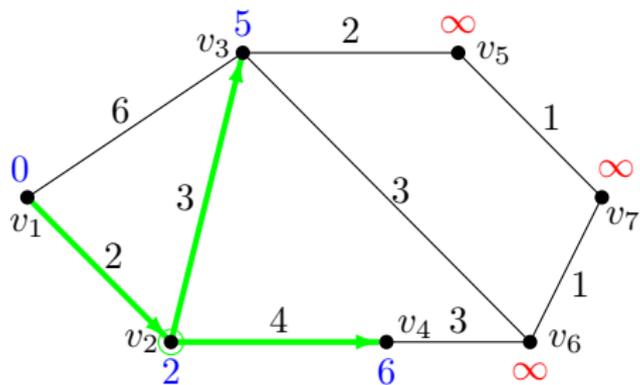


setze  $dis[v_2] = 2$ ;  
markiere  $(v_1, v_2)$ ;  
setze  $dis[v_3] = 6$ ;  
markiere  $(v_1, v_3)$ ;  
setze  $v_2$  als Bezugsknoten,  
da  $dis[v_2]$  minimal;

## Beispiel 108 (Dijkstra's Algorithmus)

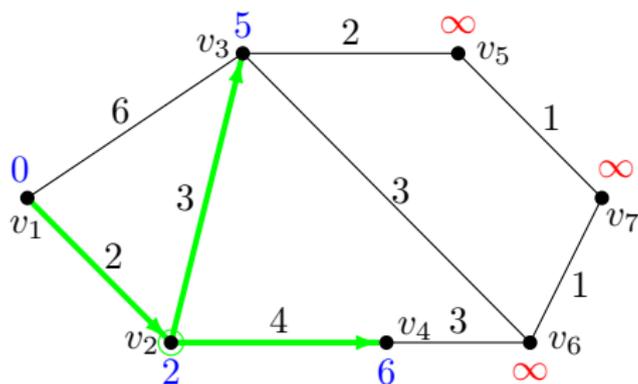


setze  $\text{dis}[v_2] = 2$ ;  
 markiere  $(v_1, v_2)$ ;  
 setze  $\text{dis}[v_3] = 6$ ;  
 markiere  $(v_1, v_3)$ ;  
 setze  $v_2$  als Bezugsknoten,  
 da  $\text{dis}[v_2]$  minimal;

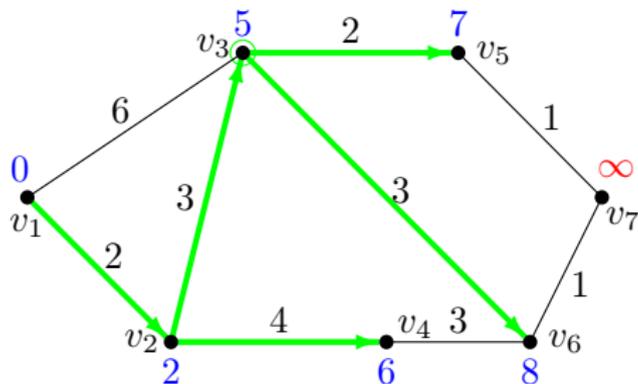


setze  $\text{dis}[v_3] = 2 + 3 = 5$ ;  
 markiere  $(v_2, v_3)$ ;  
 unmarkiere  $(v_1, v_3)$ ;  
 setze  $\text{dis}[v_4] = 2 + 4 = 6$ ;  
 markiere  $(v_2, v_4)$ ;  
 setze  $v_3$  als Bezugsknoten,  
 da  $\text{dis}[v_3]$  minimal;

## Beispiel 108 (Dijkstra's Algorithmus)

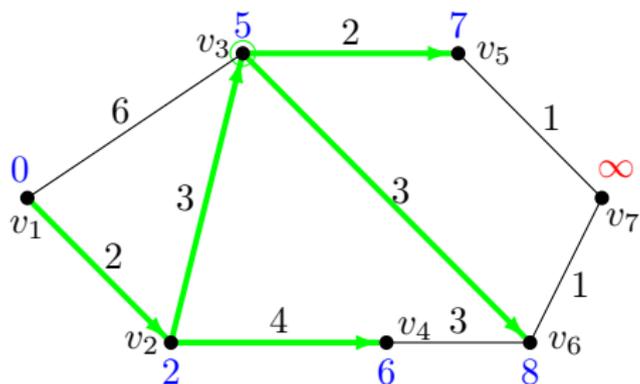


setze  $\text{dis}[v_3] = 2 + 3 = 5$ ;  
markiere  $(v_2, v_3)$ ;  
unmarkiere  $(v_1, v_3)$ ;  
setze  $\text{dis}[v_4] = 2 + 4 = 6$ ;  
markiere  $(v_2, v_4)$ ;  
setze  $v_3$  als Bezugsknoten,  
da  $\text{dis}[v_3]$  minimal;

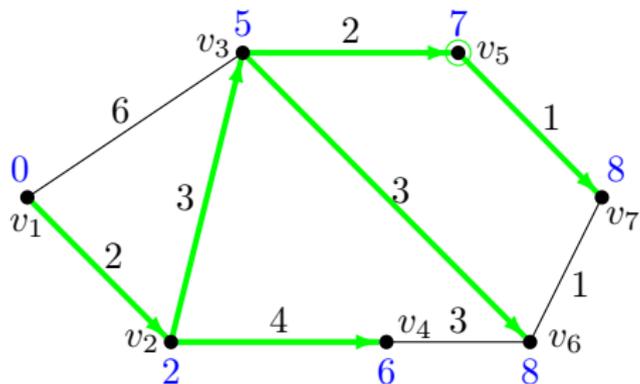


setze  $\text{dis}[v_5] = 5 + 2 = 7$ ;  
markiere  $(v_3, v_5)$ ;  
setze  $\text{dis}[v_6] = 5 + 3 = 8$ ;  
markiere  $(v_3, v_6)$ ;  
setze  $v_4$ , dann  $v_5$   
als Bezugsknoten;

## Beispiel 108 (Dijkstra's Algorithmus)



setze  $\text{dis}[v_5] = 5 + 2 = 7$ ;  
markiere  $(v_3, v_5)$ ;  
setze  $\text{dis}[v_6] = 5 + 3 = 8$ ;  
markiere  $(v_3, v_6)$ ;  
setze  $v_4$ , dann  $v_5$   
als Bezugsknoten;



setze  $\text{dis}[v_7] := 7 + 1 = 8$ ;  
markiere  $(v_5, v_7)$ ;  
alle Knoten wurden erreicht:  
 $\Rightarrow$  Algorithmus zu Ende

## Beobachtung:

- *ExtractMin* liefert eine (schwach) monoton steigende Folge von Schlüsseln  $\text{dis}[\cdot]$ ;
- Die Schlüssel  $\neq \infty$  in  $PQ$  sind stets  $\leq \text{dis}[v] + C$ , wobei  $v$  das Ergebnis der vorangehenden *ExtractMin*-Operation (bzw.  $s$  zu Beginn) und  $C := \max_{(u,w) \in A} \{\text{dis}(u, w)\}$  ist.

## Satz 109

*Dijkstra's Algorithmus (mit Fibonacci-Heaps) löst das single-source-shortest-path-Problem in Zeit  $\mathcal{O}(m + n \log n)$ .*

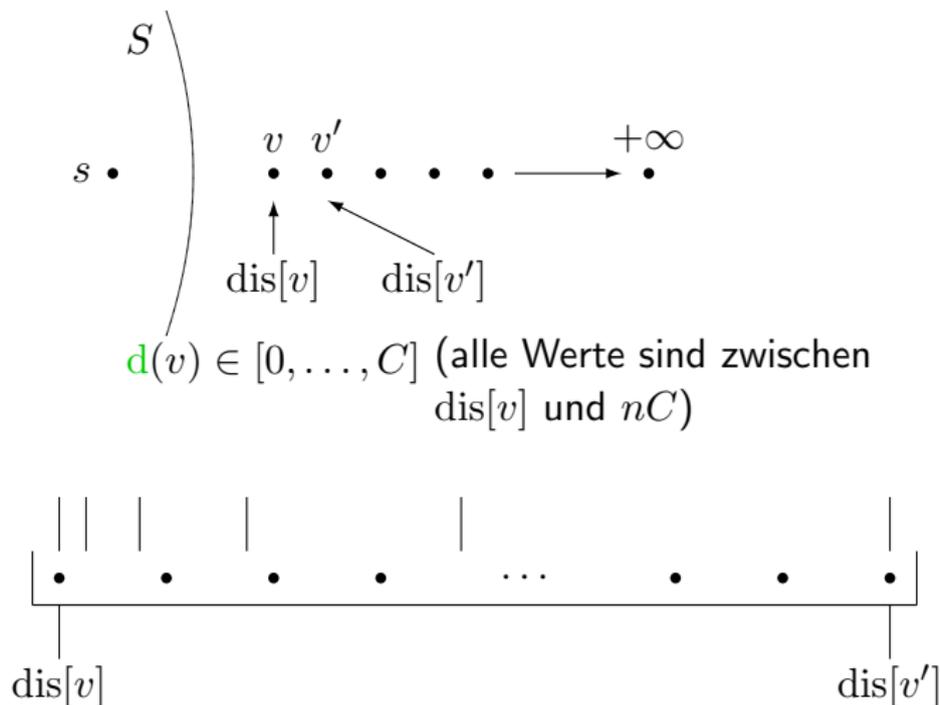
## Bemerkungen:

- 1 Verwendet man Dijkstra's Algorithmus mit  $d$ -Heaps, so erhält man Laufzeit

$$\mathcal{O}(m \log_{2+\frac{m}{n}} n)$$

- 2  Mikkel Thorup:  
*Undirected single-source shortest paths with positive integer weights in linear time*  
J. ACM **46**(3), pp. 362–394 (1999)

## 2.2 Dijkstra's Algorithmus mit Radix-Heaps



Wir verwenden Radix-Heaps (siehe [dort](#)), in der einstufigen Variante wie vorgestellt.

### Satz 110

*Dijkstra's Algorithmus mit einstufigen Radix-Heaps hat Zeitkomplexität  $\mathcal{O}(m + n \log C)$ .*

Beweis:

siehe Satz [72](#).



## Verbesserungen:

① **zweistufige** Heaps:  $\mathcal{O}(m + n \cdot \frac{\log C}{\log \log C})$

② **mehrstufige** Heaps (+Fibonacci-Heaps):  $\mathcal{O}(m + n\sqrt{\log C})$



R.K. Ahuja, Kurt Mehlhorn, J.B. Orlin, R.E. Tarjan:

*Faster Algorithms for the Shortest Path Problem*

J.ACM **37**, pp. 213–223 (1990)

## 2.3 Bellman-Ford-Algorithmus

Wir setzen (zunächst) wiederum voraus:

$$d \geq 0 .$$

Dieser Algorithmus ist ein Beispiel für **dynamische Programmierung**.

Sei  $B_k[i] :=$  Länge eines kürzesten Pfades von  $s$  zum Knoten  $i$ , wobei der Pfad höchstens  $k$  Kanten enthält.

Gesucht ist  $B_{n-1}[i]$  für  $i = 1, \dots, n$  (o.B.d.A.  $V = \{1, \dots, n\}$ ).

## Initialisierung:

$$B_1[i] := \begin{cases} d(s, i) & , \text{ falls } d(s, i) < \infty, i \neq s \\ 0 & , \text{ falls } i = s \\ +\infty & , \text{ sonst} \end{cases}$$

## Iteration:

**for**  $k := 2$  **to**  $n - 1$  **do**

**for**  $i := 1$  **to**  $n$  **do**

$$B_k[i] := \begin{cases} 0 & , \text{ falls } i = s \\ \min_{j \in N^{-1}(i)} \{ B_{k-1}[i], B_{k-1}[j] + d(j, i) \} & , \text{ sonst} \end{cases}$$

**od**

**od**

**Bemerkung:**  $N^{-1}(i)$  ist die Menge der Knoten, von denen aus eine Kante zu Knoten  $i$  führt.

**Korrektheit:**

klar (Beweis durch vollständige Induktion)

**Zeitbedarf:**

Man beachte, dass in jedem Durchlauf der äußeren Schleife jede Halbkante einmal berührt wird.

**Satz 111**

*Der Zeitbedarf des Bellman-Ford-Algorithmus ist  $\mathcal{O}(n \cdot m)$ .*

**Beweis:**

s.o.



### 3. Floyd's Algorithmus für das all-pairs-shortest-path-Problem

Dieser Algorithmus wird auch als „Kleene's Algorithmus“ bezeichnet. Er ist ein weiteres Beispiel für **dynamische Programmierung**.

Sei  $G = (V, A)$  mit Distanzfunktion  $d : A \rightarrow \mathbb{R}_0^+$  gegeben. Sei o.B.d.A.  $V = \{v_1, \dots, v_n\}$ .

Wir setzen nun

$c_{ij}^k :=$  Länge eines kürzesten Pfades von  $v_i$  nach  $v_j$ , der als innere Knoten (alle bis auf ersten und letzten Knoten) nur Knoten aus  $\{v_1, \dots, v_k\}$  enthält.

**algorithm** floyd:=

**for** alle  $(i, j)$  **do**  $c_{ij}^{(0)} := d(i, j)$  **od**      **co**  $1 \leq i, j \leq n$  **oc**

**for**  $k := 1$  **to**  $n$  **do**

**for** alle  $(i, j), 1 \leq i, j \leq n$  **do**

$$c_{ij}^{(k)} := \min \left\{ c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)} \right\}$$

**od**

**od**

**Laufzeit:**  $\mathcal{O}(n^3)$

**Korrektheit:**

Zu zeigen:  $c_{ij}^{(k)}$  des Algorithmus =  $c_{ij}^k$  (damit sind die Längen der kürzesten Pfade durch  $c_{ij}^{(n)}$  gegeben).

**Beweis:**

Richtig für  $k = 0$ . Induktionsschluss: Ein kürzester Pfad von  $v_i$  nach  $v_j$  mit inneren Knoten  $\in \{v_1, \dots, v_{k+1}\}$  enthält entweder  $v_{k+1}$  gar nicht als inneren Knoten, oder er enthält  $v_{k+1}$  genau einmal als inneren Knoten. Im ersten Fall wurde dieser Pfad also bereits für  $c_{ij}^{(k)}$  betrachtet, hat also Länge =  $c_{ij}^{(k)}$ . Im zweiten Fall setzt er sich aus einem kürzesten Pfad  $P_1$  von  $v_i$  nach  $v_{k+1}$  und einem kürzesten Pfad  $P_2$  von  $v_{k+1}$  nach  $v_j$  zusammen, wobei alle inneren Knoten von  $P_1$  und  $P_2 \in \{v_1, \dots, v_k\}$  sind. Also ist die Länge des Pfades =  $c_{i,k+1}^{(k)} + c_{k+1,j}^{(k)}$ . □

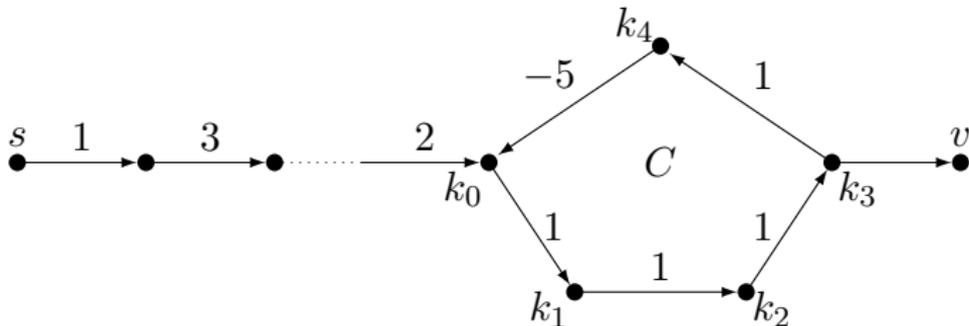
## Satz 112

*Floyd's Algorithmus für das all-pairs-shortest-path-Problem hat Zeitkomplexität  $\mathcal{O}(n^3)$ .*

## 4. Digraphen mit negativen Kantengewichten

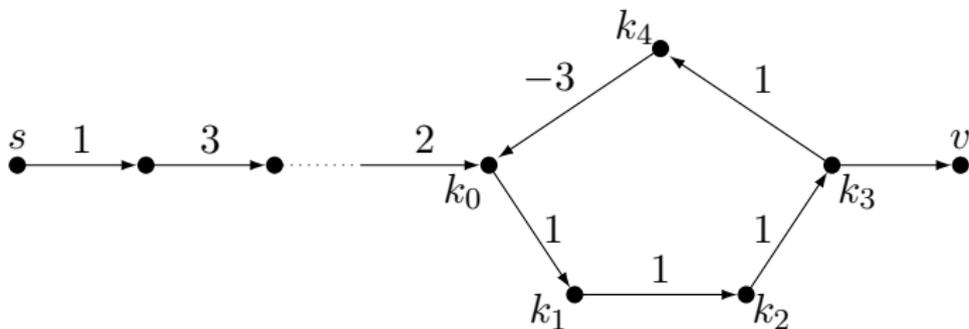
### 4.1 Grundsätzliches

Betrachte Startknoten  $s$  und einen Kreis  $C$  mit Gesamtlänge  $< 0$ .



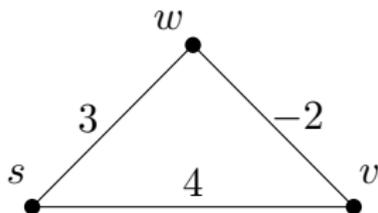
Sollte ein Pfad von  $s$  nach  $C$  und von  $C$  nach  $v$  existieren, so ist ein kürzester Pfad von  $s$  nach  $v$  **nicht definiert**.

Falls aber die Gesamtlänge des Kreises  $C \geq 0$  ist,



dann ist der kürzeste Pfad wohldefiniert. Probleme gibt es also nur dann, wenn  $G$  einen Zyklus negativer Länge enthält.

Dijkstra's Algorithmus funktioniert bei negativen Kantenlängen  
nicht:



Bei diesem Beispielgraphen (der nicht einmal einen negativen Kreis enthält) berechnet der Dijkstra-Algorithmus die minimale Entfernung von  $s$  nach  $w$  fälschlicherweise als 3 (statt 2).