Algorithmentheorie

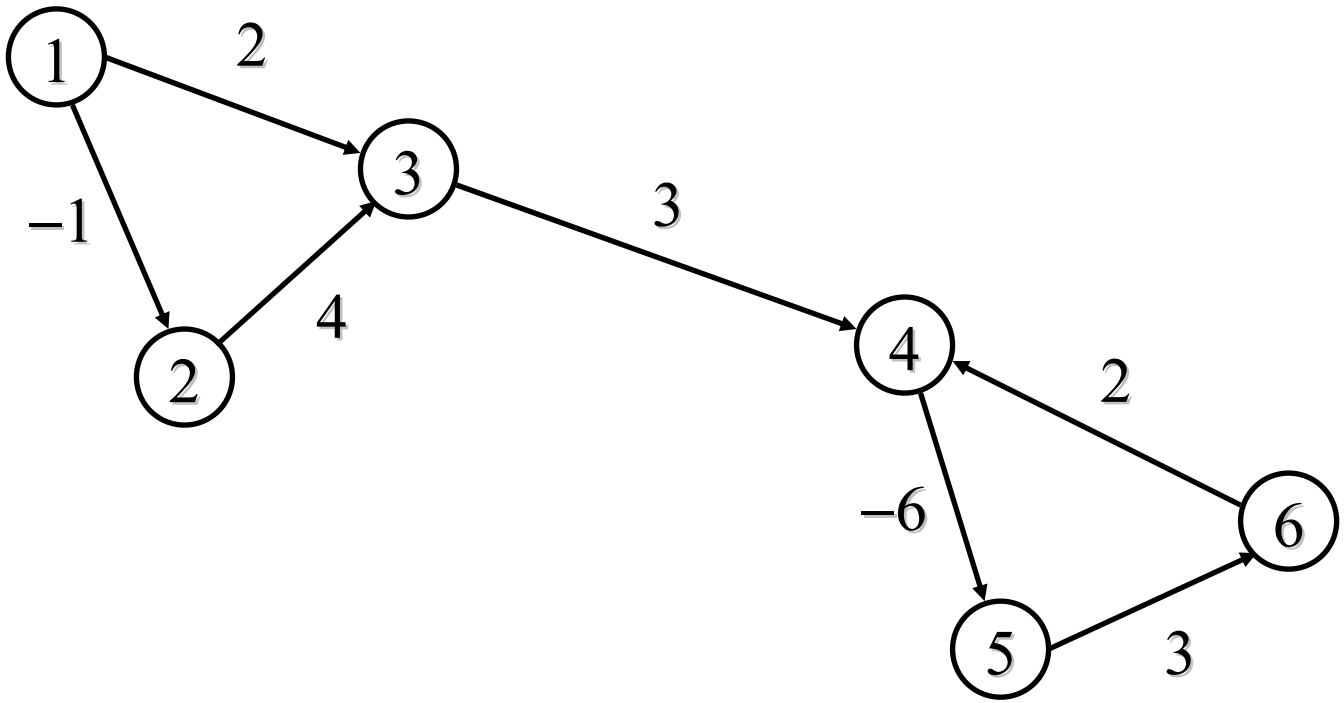
11 - Kürzeste (billigste) Wege

Prof. Dr. S. Albers

1. Kürzeste (billigste) Wege

Gerichteter Graph $G = (V, E)$

Kostenfunktion $c : E \rightarrow R$



Entfernung zwischen zwei Knoten

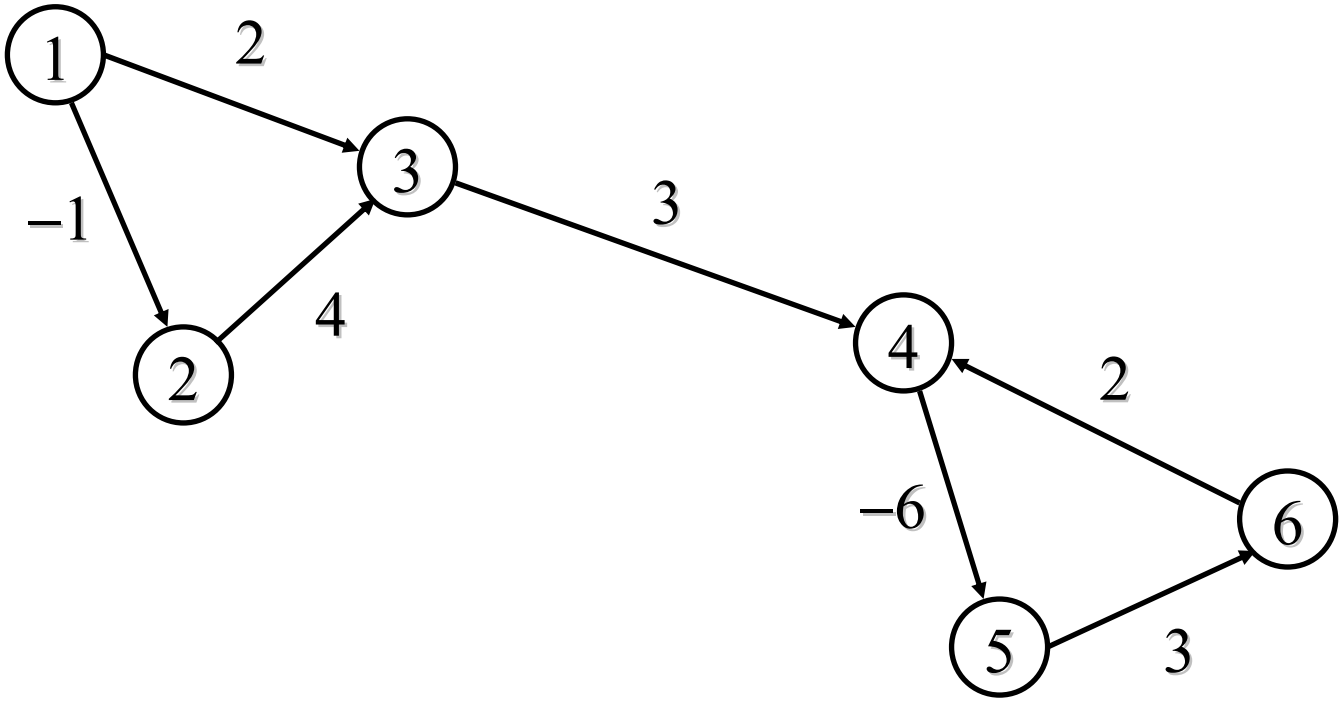
Kosten eines Wegs $P = v_0, v_1, \ldots, v_l$ von v nach w

$$c(P) = \sum_{i=0}^{l-1} c(v_i, v_{i+1})$$

Entfernung von v nach w (nicht immer definiert)

$dist(v,w) = \inf \{ c(P) \mid P \text{ ist Weg von } v \text{ nach } w \}$

Beispiel



dist(1,2) =

dist(1,3) =

dist(3,1) =

dist(3,4) =

2. Kürzeste Wege von einem Knoten *s*

(Single Source Shortest Paths)

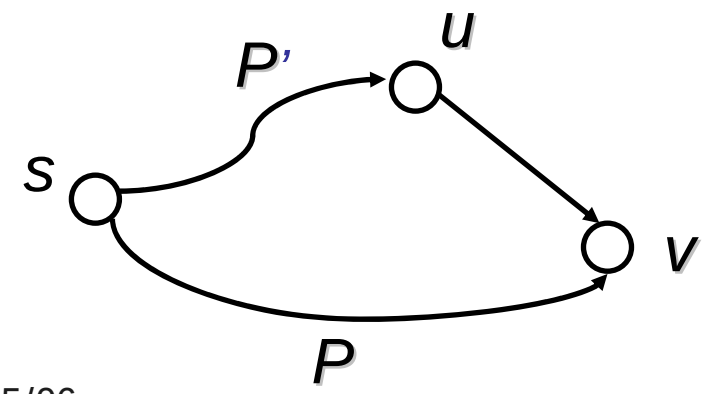
Eingabe: Netzwerk $G = (V, E, c)$ $c : E \to R$ Knoten s

Ausgabe: $dist(s,v)$ für alle $v \in V$

Beobachtung: *dist*-Funktion erfüllt eine Dreiecksungleichung

Sei $(u,v) \in E$ eine beliebige Kante

$$dist(s,v) \leq dist(s,u) + c(u,v)$$



P = kürzester Weg von s nach v

P' = kürzester Weg von s nach u

Greedy-Ansatz für einen Algorithmus

1. Überschätze die *dist*-Funktion

$$dist(s,v) = \begin{cases} 0 & \text{falls } v = s \\ \infty & \text{falls } v \neq s \end{cases}$$

2. Solange eine Kante $e = (u,v)$ existiert mit

   $dist(s,v) > dist(s,u) + c(u,v)$

   setze $dist(s,v) \leftarrow dist(s,u) + c(u,v)$

Grundalgorithmus

1. DIST[s] $\leftarrow 0$;
2. **for all** $v \in V \setminus \{s\}$ **do** DIST[v] $\leftarrow \infty$ **endfor;**
3. **while** $\exists\ e = (u,v) \in E$ mit DIST[v] > DIST[u] + $c(u,v)$ **do**
4. Wähle eine solche Kante $e = (u,v)$;
5. DIST[v] $\leftarrow$ DIST[u] + $c(u,v)$;
6. **endwhile;**

Fragen:

1. Wie testet man in Zeile 3, ob eine Dreiecksungleichung verletzt ist?
2. Welche Kante wählt man in Zeile 4?

Lösung

Speichere eine Menge U aller Knoten, aus denen Kanten ausgehen könnten, die eine Dreiecksungleichung verletzen.

- Initialisierung $U = \{s\}$
- Ein Knoten v wird in U aufgenommen, wenn DIST[v] vermindert wird.

1. Test, ob Dreiecksungleichung verletzt: $U \neq \emptyset$?
2. Wähle einen Knoten aus U aus und stelle für die ausgehenden Kanten die Dreiecksungleichung her.

Verfeinerter Algorithmus

1. DIST[s] $\leftarrow 0$;
2. **for all** $v \in V \setminus \{s\}$ **do** DIST[v] $\leftarrow \infty$ **endfor;**
3. $U \leftarrow \{s\}$;
4. **while** $U \neq \emptyset$ **do**
5. Wähle und streiche einen Knoten $u \in U$;
6. **for all** $e = (u,v) \in E$ **do**
7. **if** DIST[v] > DIST[u] + $c(u,v)$ **then**
8. DIST[v] $\leftarrow$ DIST[u] + $c(u,v)$;
9. $U \leftarrow U \cup \{v\}$;
10. **endif;**
11. **endfor;**
12. **endwhile;**

Invariante für die DIST-Werte

Lemma 1: Für alle Knoten $v \in V$ gilt stets $\mathrm{DIST}[v] \geq dist(s,v)$.

Beweis: (durch Widerspruch)

Sei v der erste Knoten, für den sich bei d. Relaxierung einer Kante (u,v) $\mathrm{DIST}[v] < dist(s,v)$ ergibt.

Dann gilt:

$$\mathrm{DIST}[u] + c(u,v) = \mathrm{DIST}[v] < dist(s,v) \leq dist(s,u) + c(u,v)$$

Wichtige Eigenschaften

Lemma 2:

a) Falls $v \notin U$, dann gilt für alle $(v,w) \in E$: $\text{DIST}[w] \leq \text{DIST}[v] + c(v,w)$

b) Sei $s=v_0, v_1, ..., v_l=v$ ein kürzester Weg von s nach v.
Falls $\text{DIST}[v] > dist(s,v)$, dann gibt es ein v_i, $0 \leq i \leq l-1$, mit
$v_i \in U$ und $\text{DIST}[v_i] = dist(s,v_i)$.

c) Falls G keine negativen Zyklen hat und $\text{DIST}[v] > dist(s,v)$ für ein
$v \in V$, dann gibt es ein $u \in U$ und $\text{DIST}[u] = dist(s,u)$.

d) Wählt man in Zeile 5 immer ein $u \in U$ mit $\text{DIST}[u] = dist(s,u)$,
dann wird die while-Schleife für jeden Knoten nur einmal ausgeführt.

Effiziente Implementierungen

Wie findet man in Zeile 5 immer einen Knoten $u \in U$ mit DIST[u] = $dist(s,u)$?

Im Allgemeinen ist dies nicht bekannt, jedoch für wichtige Spezialfälle.

- Nicht-negative Netzwerke (keine negativen Kantengewichte)
  Algorithmus von Dijkstra
- Netzwerke ohne negative Zyklen
  Bellman-Ford-Algorithmus
- Azyklische Netzwerke

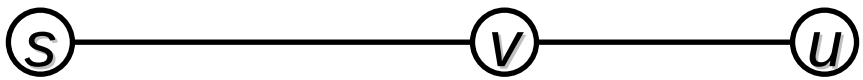
3. Nicht-negative Netzwerke

5’. Wähle und streiche einen Knoten $u \in U$ mit DIST[u] minimal.

Lemma 3: Mit 5’ gilt DIST[u]=$dist(s,u)$.

Beweis: Wegen Lemma 2b) gibt es auf dem kürzesten Weg von s nach u einen Knoten $v \in U$ mit DIST[v]=$dist(s,v)$.

$$\text{DIST}[u] \leq \text{DIST}[v] = dist(s,v) \leq dist(s,u)$$



U als Prioritätswarteschlange

Elemente der Form (*key*, *inf*) sind (DIST[*v*], *v*).

Empty(*Q*): Ist *Q* leer?

Insert(*Q*, *key*, *inf*): Fügt (*key*,*inf*) in *Q* ein.

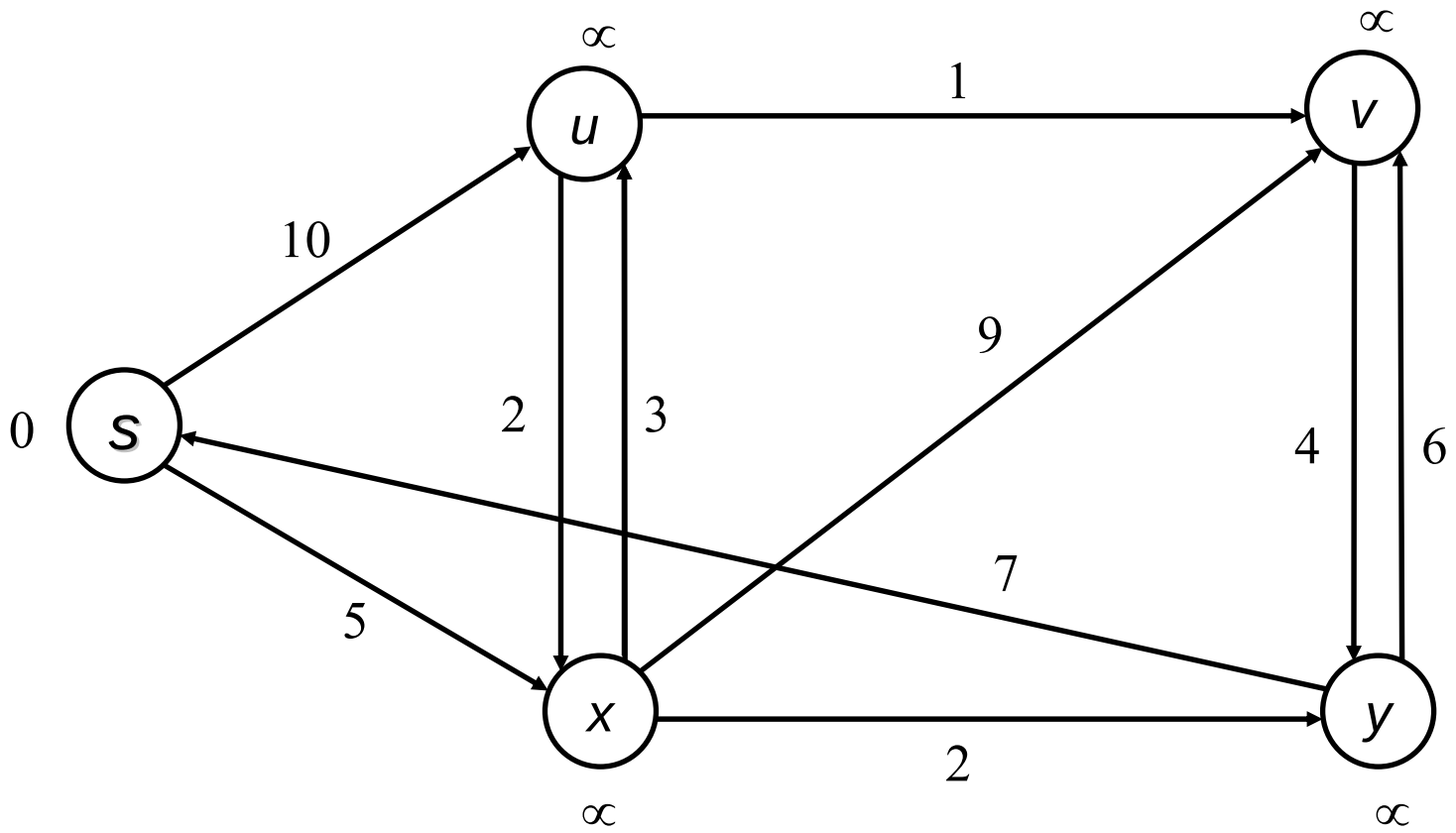
DeleteMin(*Q*): Entfernt das Element mit kleinstem Schlüssel und liefert es zurück.

DecreaseKey(*Q*, *element*, *j*): Vermindert den Schlüssel von *element* auf *j*, sofern *j* kleiner als der alte Schlüssel ist.

Algorithmus von Dijkstra

1. DIST[s] $\leftarrow 0$; Insert(U,0,s);
2. **for all** $v \in V \setminus \{s\}$ **do** DIST[v] $\leftarrow \infty$; Insert(U, ∞, v); **endfor;**
3. **while** $\neg$Empty(U) **do**
4. (d,u) $\leftarrow$ DeleteMin(U);
5. **for all** $e = (u,v) \in E$ **do**
6. **if** DIST[v] > DIST[u] + c(u,v) **then**
7. DIST[v] $\leftarrow$ DIST[u] + c(u,v);
8. DecreaseKey(U, v, DIST[v]);
9. **endif;**
10. **endfor;**
11. **endwhile;**

Beispiel



Laufzeit

$$O(n (T_{Insert} + T_{Empty} + T_{DeleteMin}) + m\, T_{DecreaseKey} + m + n)$$

Fibonacci-Heaps:

T_{Insert} : O(1)

$T_{DeleteMin}$: O(log n) amortisiert

$T_{DecreaseKey}$: O(1) amortisiert

$O(n \log n + m)$

4. Netzwerke ohne negative Zyklen

Organisiere *U* als Schlange.

Lemma 4: Jeder Knoten *v* wird maximal *n*-mal an *U* angehängt.

Beweis: Wird *v* zum *i*-ten Mal an *U* angehängt, wenn DIST[*v*] > *dist*(*s*,*v*) gilt, so existiert wegen Lemma 2c) ein $u_i \in U$ mit DIST[u_i] = *dist*(*s*,u_i)

Knoten u_i wird vor *v* aus *U* entfernt und nie wieder angehängt.

Die Knoten u_1, u_2, u_3, ... sind paarweise verschieden.

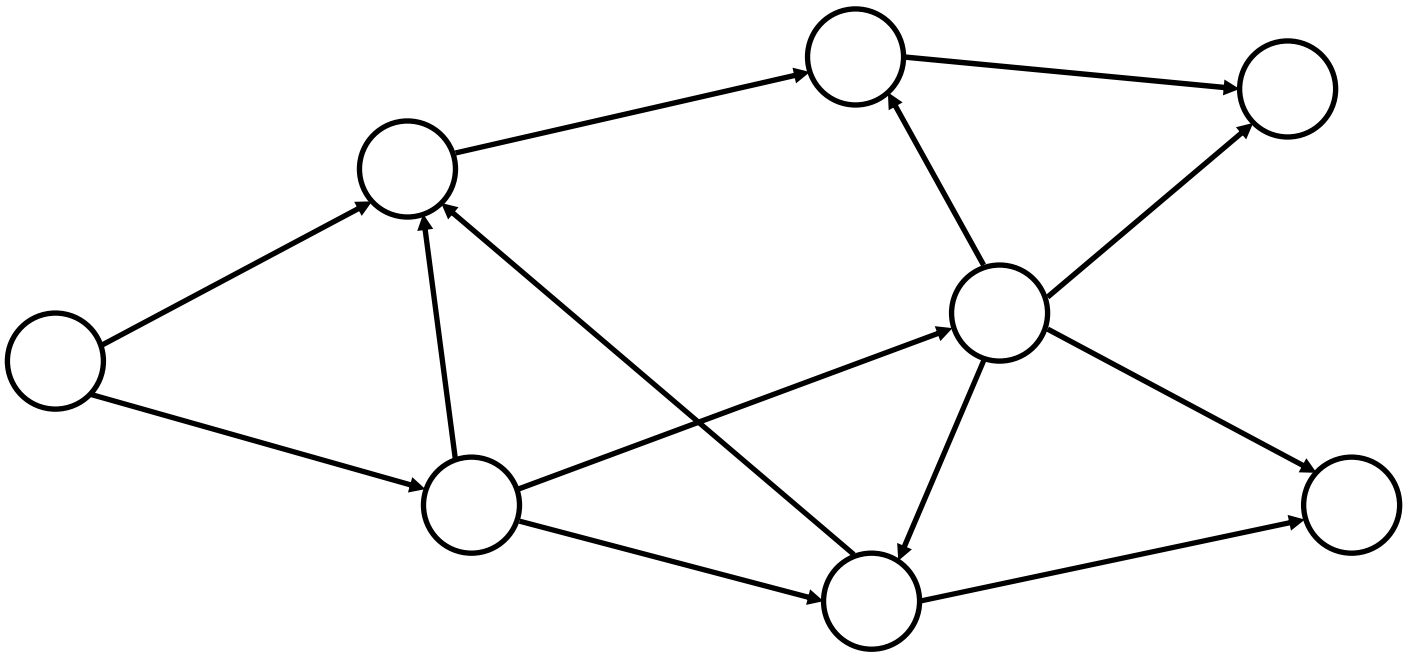
Bellman-Ford-Algorithmus

1. DIST[s] $\leftarrow 0$; Z[s] $\leftarrow$ 0;
2. **for all** $v \in V \setminus \{s\}$ **do** DIST[v] $\leftarrow \infty$; Z[v] $\leftarrow$ 0; **endfor;**
3. $U \leftarrow \{s\}$;
4. **while** $U \neq \emptyset$ **do**
5. Wähle und streiche Knoten u am Kopf von U; Z[u] $\leftarrow$ Z[u]+1;
6. **if** Z[u] > n **then** return „negativer Zyklus“;
7. **for all** $e = (u,v) \in E$ **do**
8. **if** DIST[v] > DIST[u] + $c(u,v)$ **then**
9. DIST[v] $\leftarrow$ DIST[u] + $c(u,v)$;
10. $U \leftarrow U \cup \{v\}$;
11. **endif;**
12. **endfor;**
13. **endwhile;**

5. Azyklische Netzwerke

Topologisches Sortieren: num: $V \rightarrow \{1 , ... , n\}$

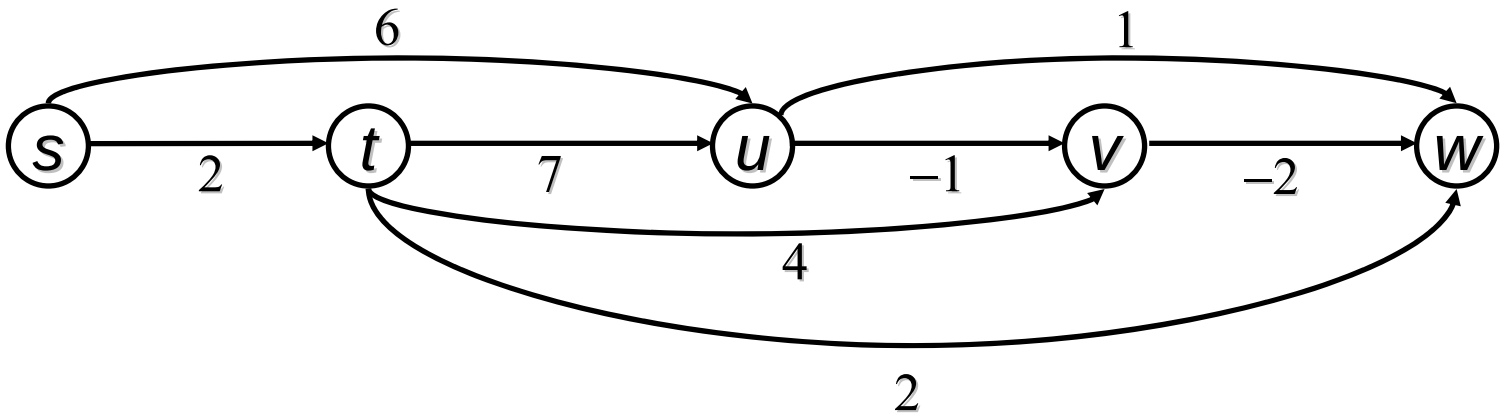
Für $(u,v) \in E$ gilt num(u) < num(v)



Algorithmus für azyklische Graphen

1. Sortiere $G = (V, E, c)$ topologisch;
2. DIST[s] $\leftarrow 0$;
3. **for all** $v \in V \setminus \{s\}$ **do** DIST[v] $\leftarrow \infty$; **endfor;**
4. $U \leftarrow \{ v \mid v \in V$ mit num(v) < $n\}$;
5. **while** $U \neq \emptyset$ **do**
6. Wähle und streiche Knoten $u \in U$ mit kleinstem num-Wert ;
7. **for all** $e = (u,v) \in E$ **do**
8. **if** DIST[v] > DIST[u] + $c(u,v)$ **then**
9. DIST[v] $\leftarrow$ DIST[u] + $c(u,v)$;
10. **endif;**
11. **endfor;**
12. **endwhile;**

Beispiel



Korrektheit

Lemma 5: Wenn der *i*-te Knoten u_i aus U entfernt wird, gilt

$\text{DIST}[u_i] = dist(s,u_i)$.

Beweis: Induktion nach *i*.

i = 1: ok

i > 1: Sei $s=v_1, v_2, \ldots, v_l, v_{l+1}=u_i$ kürzester Weg von *s* nach u_i.

v_l wird vor u_i aus U entfernt. Nach I.V. gilt $\text{DIST}[v_l] = dist(s,v_l)$.

Wenn (v_l, u_i) relaxiert ist, gilt:

$$\text{DIST}[u_i] \leq \text{DIST}[v_l] + c(v_l, u_i) = dist(s,v_l) + c(v_l, u_i) = dist(s,u_i)$$