# Algorithmentheorie

# Greedy-Verfahren

Prof. Dr. S. Albers

# 1. Übersicht

**Greedy-Prinzip:**

Triff in jedem Verfahrensschritt diejenige Entscheidung, die im Moment am besten ist.

- Kürzeste (billigste) Wege
- Minimale spannende Bäume

# Algorithmentheorie

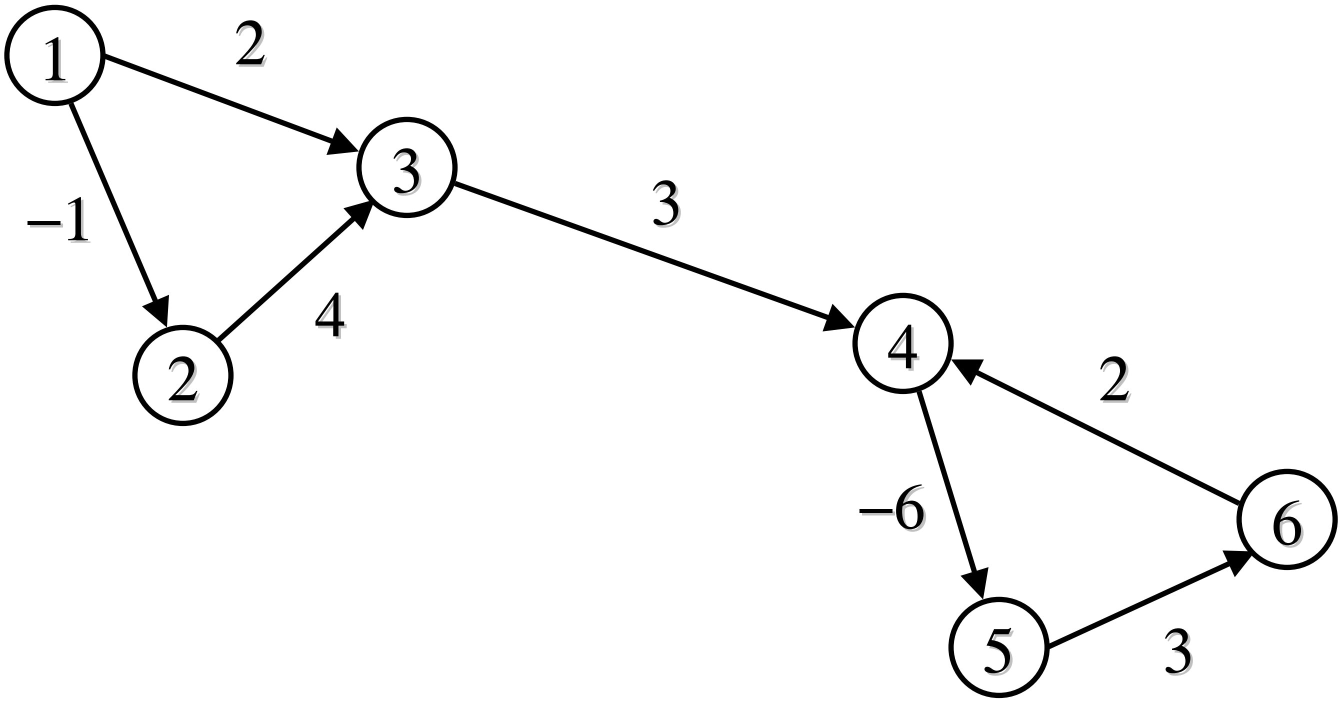
# 11 - Kürzeste (billigste) Wege

Prof. Dr. S. Albers

# 1. Kürzeste (billigste) Wege

Gerichteter Graph $G = (V, E)$

Kostenfunktion $c : E \to R$



# Entfernung zwischen zwei Knoten

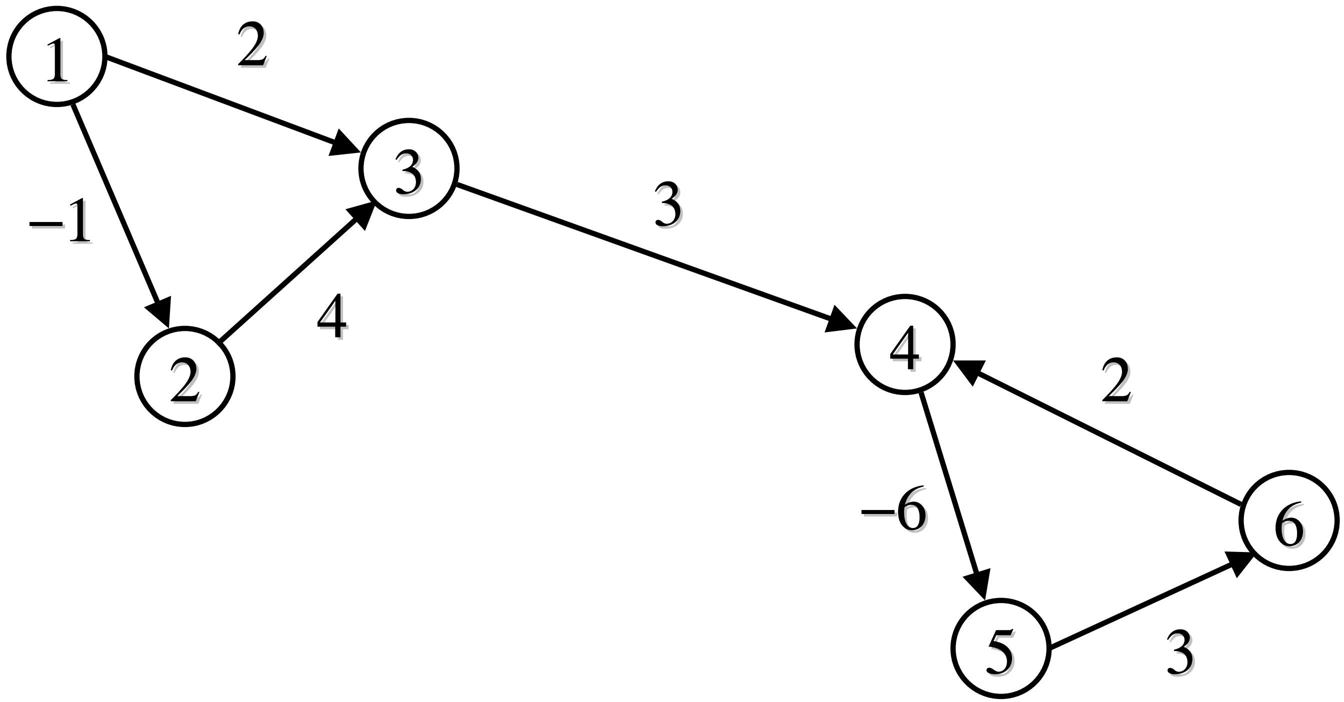
Kosten eines Wegs $P = v_0, v_1, \dots, v_l$ von $v$ nach $w$

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

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)

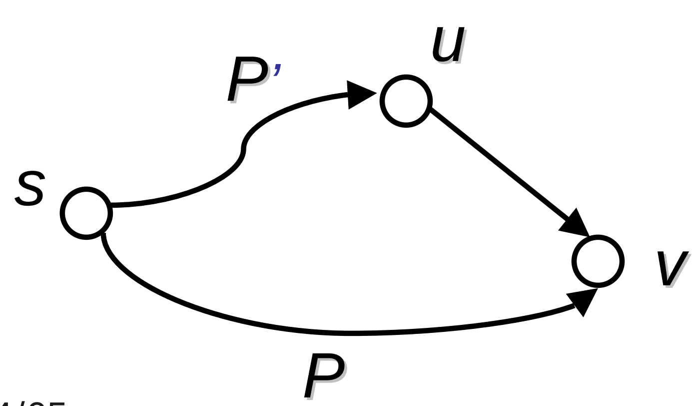
**Eingabe:** Netzwerk $G = (V, E, c)$ $c : E \to R$ Knoten $s$

**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] $\leftarrow \propto$ **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

DIST[$s$] $\leftarrow 0$;

**for all** $v \in V \setminus \{s\}$ **do** DIST[$v$] $\leftarrow \propto$ **endfor;**

$U \leftarrow \{s\}$;

**while** $U \neq \emptyset$ **do**

  Wähle und streiche einen Knoten $u \in U$;

  **for all** $e = (u,v) \in E$ **do**

    **if** DIST[$v$] > DIST[$u$] + $c(u,v)$ **then**

      DIST[$v$] $\leftarrow$ DIST[$u$] + $c(u,v)$;

      $U \leftarrow U \cup \{v\}$;

    **endif;**

  **endfor;**

**endwhile;**

# Invariante für die DIST-Werte

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

**Beweis:** (durch Widerspruch)

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

Dann gilt:

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

# Wichtige Eigenschaften

**Lemma 2:**

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

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

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

d) Wählt man in Zeile 5 immer ein $u \in U$ mit 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

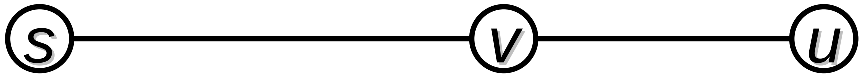
# 3. Nicht-negative Netzwerke

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.

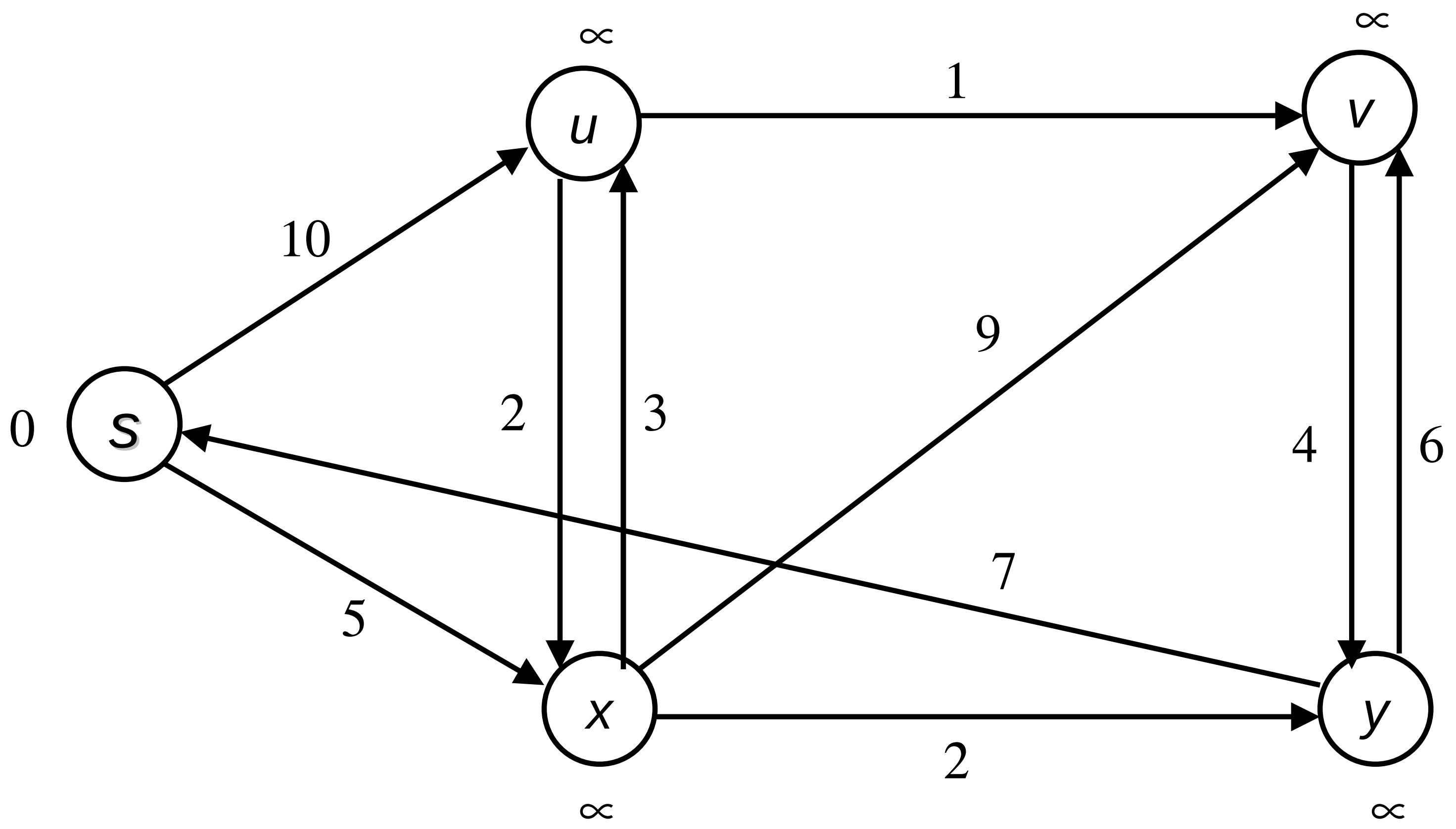
DeleteMin(*Q*): Entfernt das Element mit kleinstem Schlüssel und liefert es zurück.

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

# Algorithmus von Dijkstra

```
DIST[s] ← 0;   Insert(U,0,s);
for all  v ∈ V \ {s}  do DIST[v] ← ∝;  Insert(U, ∝, v);  endfor;
while ¬Empty(U) do
    (d,u) ← DeleteMin(U);
    for all e = (u,v) ∈ E do
        if DIST[v] > DIST[u] + c(u,v) then
            DIST[v] ← DIST[u] + c(u,v);
            DecreaseKey(U, v, DIST[v]);
        endif;
    endfor;
endwhile;
```

# Beispiel



# Laufzeit

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

Fibonacci-Heaps:

$T_{\mathrm{Insert}}$ : O(1)

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

$T_{\mathrm{DecreaseKey}}$ : 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.

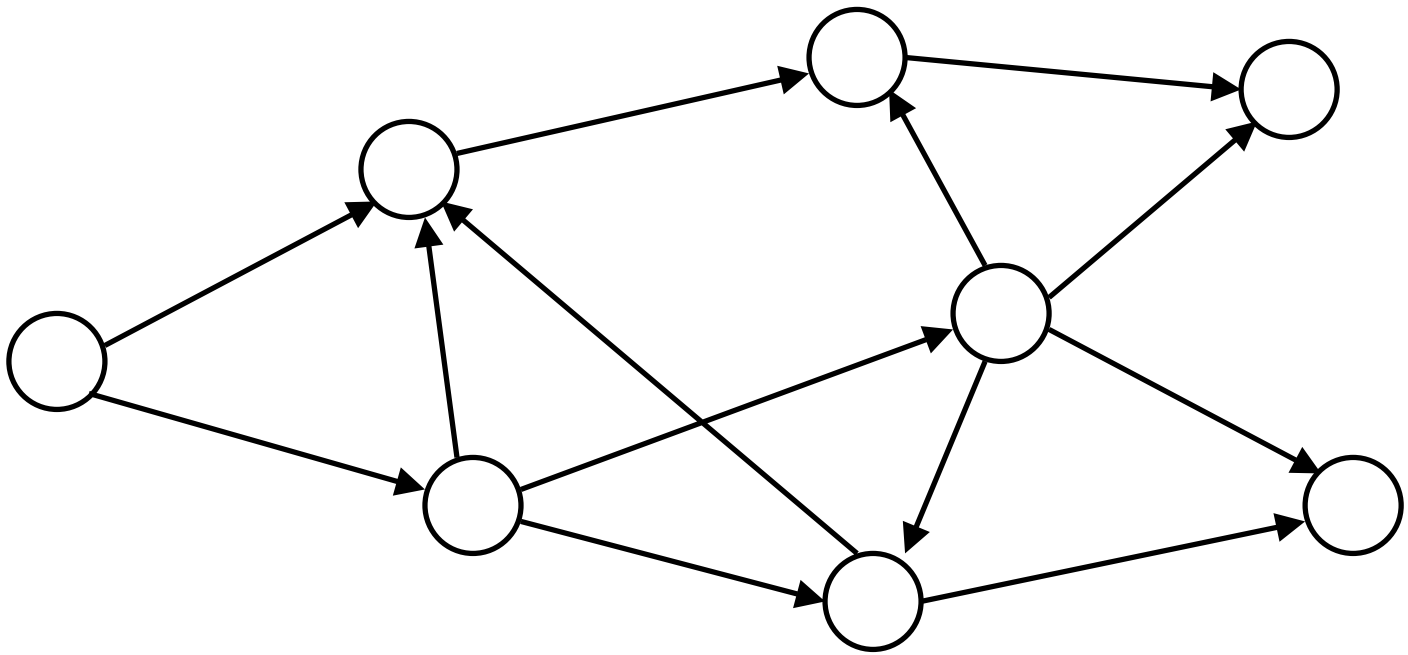
# Bellman-Ford-Algorithmus

```
DIST[s] ← 0;   Z[s] ← 0;
for all  v ∈ V \ {s}  do DIST[v] ← ∞ ;  Z[v] ← 0; endfor;
U ← {s};
while  U ≠ ∅  do
     Wähle und streiche Knoten u am Kopf von U;  Z[u] ← Z[u]+1;
     if  Z[u] > n  then  return „negativer Zyklus“;
     for all e = (u,v) ∈ E do
         if DIST[v] > DIST[u] + c(u,v) then
                DIST[v] ← DIST[u] + c(u,v);
                U ← U ∪ {v};
         endif;
     endfor;
endwhile;
```

# 5. Azyklische Netzwerke

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

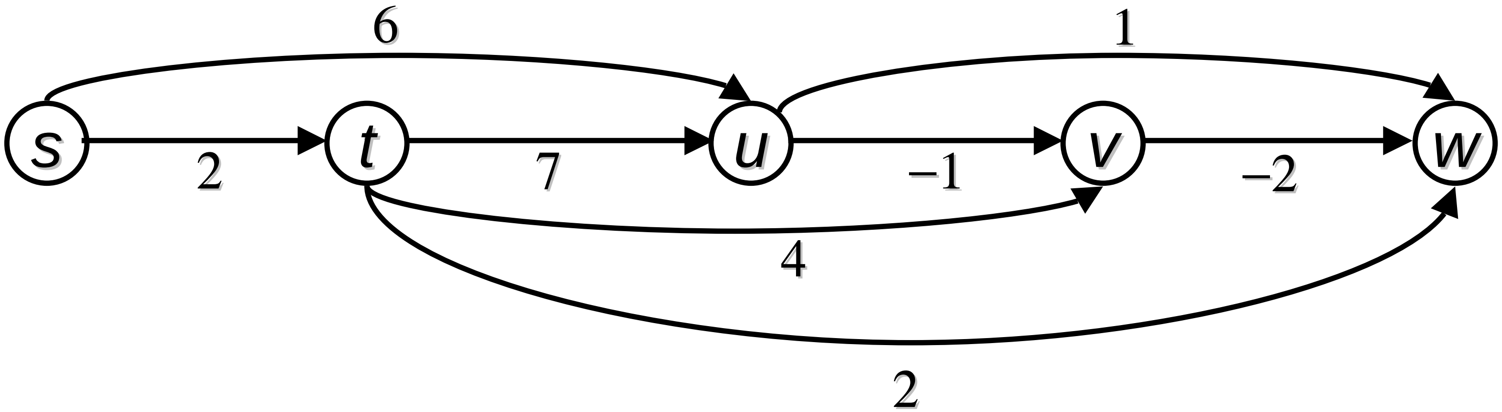
Für $(u,v) \in E$ gilt num($u$) < num($v$)



# Algorithmus für azyklische Graphen

Sortiere $G = (V, E, c)$ topologisch;
DIST[$s$] $\leftarrow 0$;
**for all** $v \in V \setminus \{s\}$ **do** DIST[$v$] $\leftarrow \propto$ ; **endfor;**
$U \leftarrow \{ v \mid v \in V$ mit num($v$) $< n\}$;
**while** $U \neq \emptyset$ **do**
    Wähle und streiche Knoten $u \in U$ mit kleinstem num-Wert ;
    **for all** $e = (u,v) \in E$ **do**
        **if** DIST[$v$] > DIST[$u$] + $c(u,v)$ **then**
            DIST[$v$] $\leftarrow$ DIST[$u$] + $c(u,v)$;
        **endif;**
    **endfor;**
**endwhile;**

# Beispiel



# Korrektheit

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

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

**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 $\mathrm{DIST}[v_l] = dist(s,v_l)$.

Wenn $(v_l, u_i)$ relaxiert ist, gilt:

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