

## 26. Kürzeste Wege

---

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

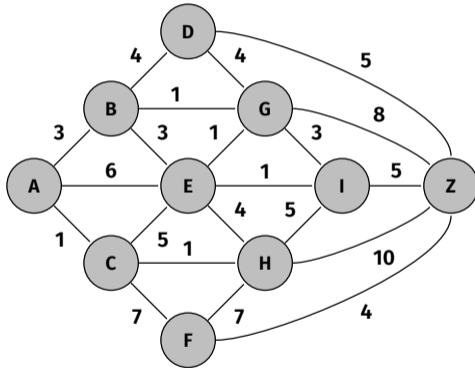
[Ottman/Widmayer, Kap. 9.5 Cormen et al, Kap. 24.1-24.3, 25.2-25.3]

# Routenfinder

**Gegeben:** Städte A - Z und Distanzen zwischen den Städten

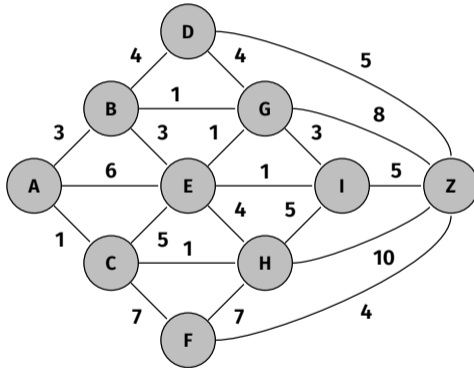
# Routenfinder

**Gegeben:** Städte A - Z und Distanzen zwischen den Städten



# Routenfinder

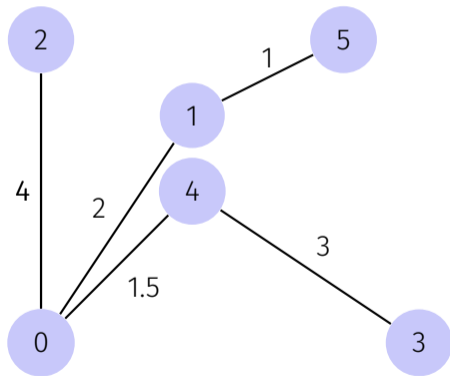
**Gegeben:** Städte A - Z und Distanzen zwischen den Städten



**Gesucht:** Was ist der kürzeste Weg von A nach Z?

# Notation

Ein **gewichteter Graph**  $G = (V, E, c)$  ist ein Graph  $G = (V, E)$  mit einer **Kantengewichtsfunktion**  $c : E \rightarrow \mathbb{R}$ .  $c(e)$  heisst **Gewicht** der Kante  $e$ .

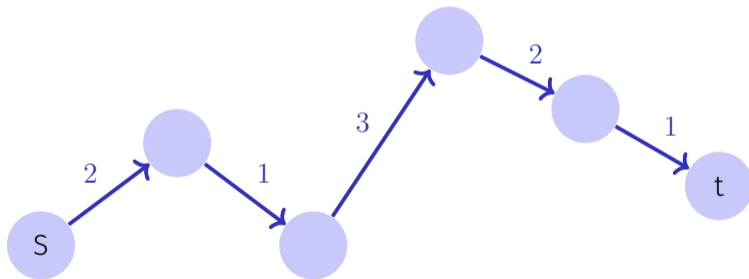


# Gewicht eines Weges

**Gegeben:**  $G = (V, E, c)$ ,  $c : E \rightarrow \mathbb{R}$ ,  $s, t \in V$ .

**Weg:**  $p = \langle s = v_0, v_1, \dots, v_k = t \rangle$ ,  $(v_i, v_{i+1}) \in E$  ( $0 \leq i < k$ )

**Gewicht:**  $c(p) := \sum_{i=0}^{k-1} c((v_i, v_{i+1}))$ .



Weg mit Gewicht 9

# Kürzeste Wege

**Notation:** Wir schreiben

$$u \overset{p}{\rightsquigarrow} v \quad \text{oder} \quad p : u \rightsquigarrow v$$

und meinen einen Weg  $p$  von  $u$  nach  $v$

**Gesucht:**  $\delta(u, v)$  = minimales Gewicht eines Weges von  $u$  nach  $v$ :

$$\delta(u, v) = \begin{cases} \infty & \text{kein Weg von } u \text{ nach } v \\ \min\{c(p) : u \overset{p}{\rightsquigarrow} v\} & \text{sonst} \end{cases}$$

Einen Weg minimalen Gewichts nennen wir im folgenden oft einfach **kürzesten Weg**.

# Trivialer Algorithmus?

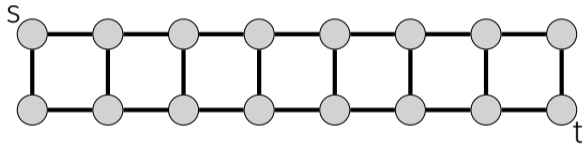


# Trivialer Algorithmus?

Alle Wege ausprobieren?

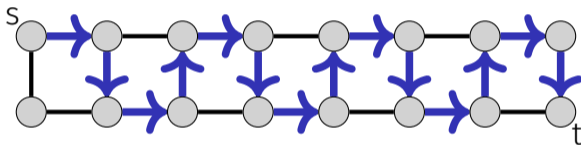
# Trivialer Algorithmus?

Alle Wege ausprobieren?



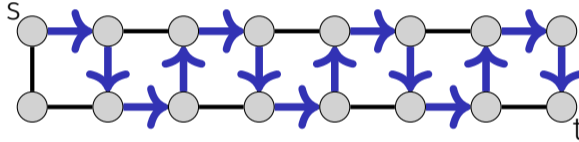
# Trivialer Algorithmus?

Alle Wege ausprobieren?



# Trivialer Algorithmus?

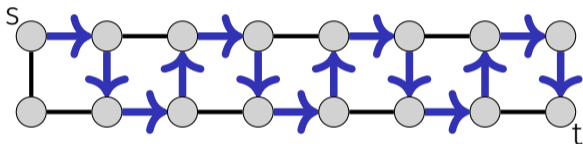
Alle Wege ausprobieren?



(mindestens  $2^{|V|/2}$  Wege von  $s$  nach  $t$ )

# Trivialer Algorithmus?

Alle Wege ausprobieren?



(mindestens  $2^{|V|/2}$  Wege von  $s$  nach  $t$ )

⇒ Ineffizient. Es kann exponentiell viele Wege geben.

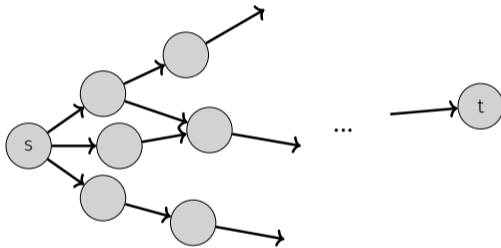
# Einfachster Fall

# Einfachster Fall

Konstantes Kantengewicht (jede Kante hat Gewicht 1)

# Einfachster Fall

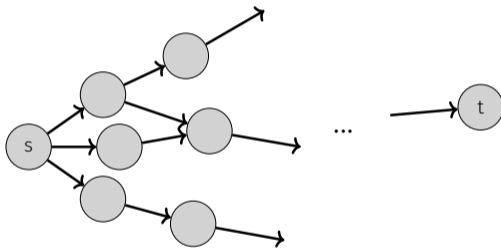
Konstantes Kantengewicht (jede Kante hat Gewicht 1)





# Einfachster Fall

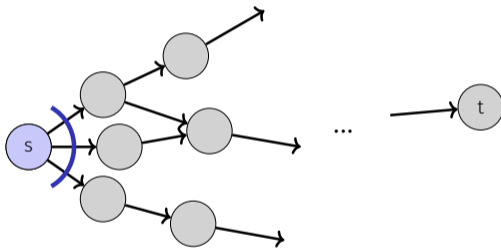
Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche

# Einfachster Fall

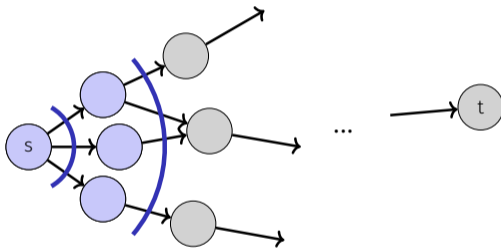
Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche

# Einfachster Fall

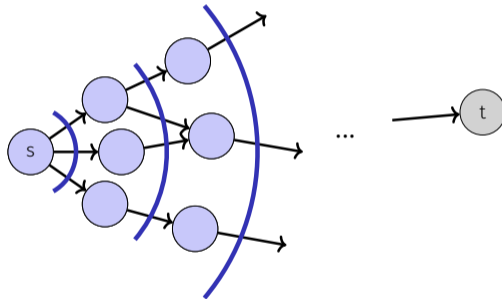
Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche

# Einfachster Fall

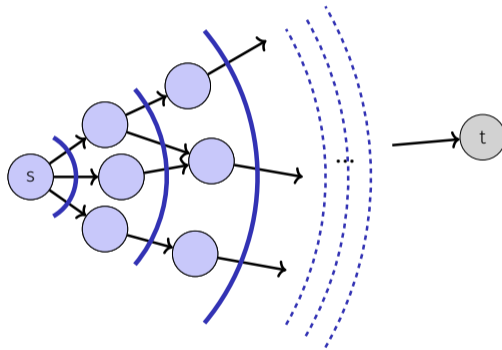
Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche

# Einfachster Fall

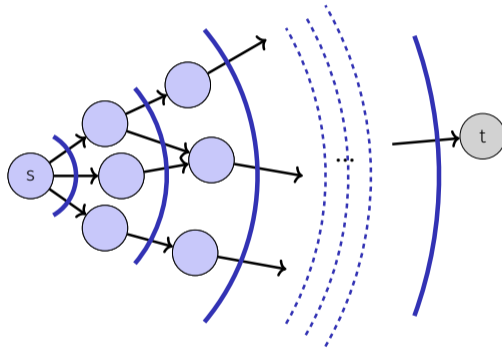
Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche

# Einfachster Fall

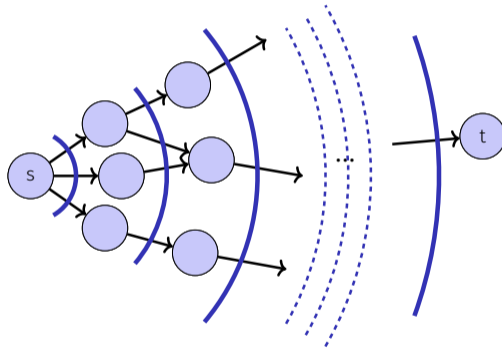
Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche

# Einfachster Fall

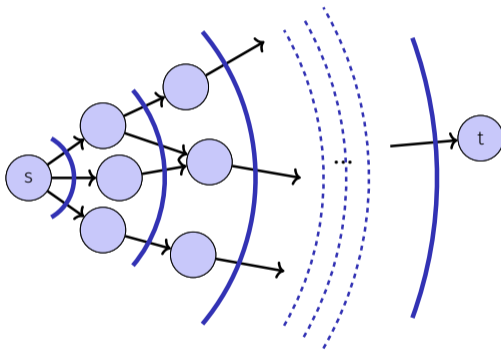
Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche

# Einfachster Fall

Konstantes Kantengewicht (jede Kante hat Gewicht 1)



⇒ **Lösung:** Breitensuche  $\mathcal{O}(|V| + |E|)$



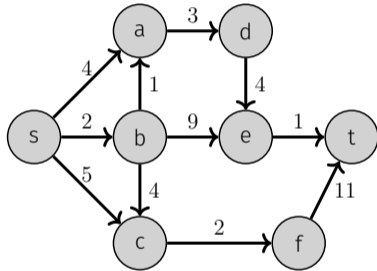
# Dijkstras Algorithmus: Beobachtung

# Dijkstras Algorithmus: Beobachtung

**Wichtige Annahme:** alle Gewichte sind positiv.

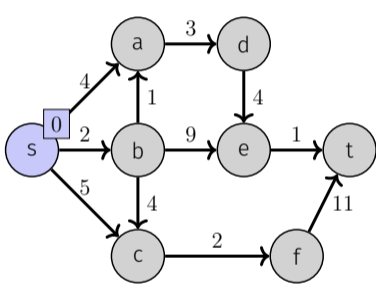
# Dijkstras Algorithmus: Beobachtung

**Wichtige Annahme:** alle Gewichte sind positiv.



# Dijkstras Algorithmus: Beobachtung

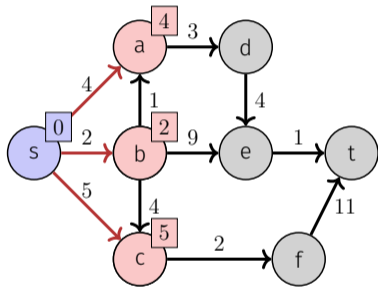
**Wichtige Annahme:** alle Gewichte sind positiv.



Kürzester Weg  $s \rightsquigarrow u$  hat Länge  $l$  (exakt).

# Dijkstras Algorithmus: Beobachtung

**Wichtige Annahme:** alle Gewichte sind positiv.



Kürzester Weg  $s \rightsquigarrow u$  hat Länge  $l$  (exakt).

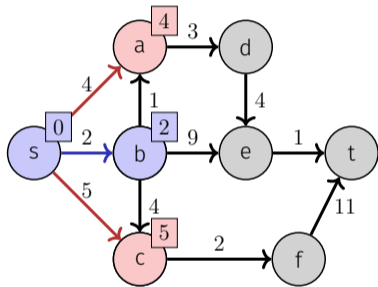


**Obere Schranke:**

Kürzester Weg  $s \rightsquigarrow u$  hat Länge höchstens  $l$ .

# Dijkstras Algorithmus: Beobachtung

**Wichtige Annahme:** alle Gewichte sind positiv.



Kürzester Weg  $s \rightsquigarrow u$  hat Länge  $l$  (exakt).



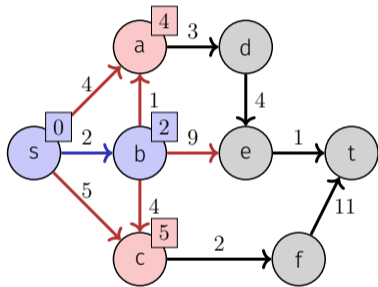
**Obere Schranke:**

Kürzester Weg  $s \rightsquigarrow u$  hat Länge höchstens  $l$ .

**Beobachtung:** Kürzeste ausgehende Kante  $(s, u)$  ist der kürzeste Weg von  $s$  zu diesem Knoten  $u$ .

# Dijkstras Algorithmus: Beobachtung

**Wichtige Annahme:** alle Gewichte sind positiv.



Kürzester Weg  $s \rightsquigarrow u$  hat Länge  $l$  (exakt).



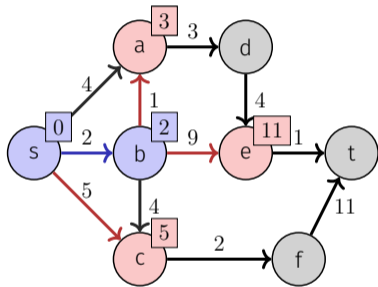
**Obere Schranke:**

Kürzester Weg  $s \rightsquigarrow u$  hat Länge höchstens  $l$ .

**Beobachtung:** Kürzeste ausgehende Kante  $(s, u)$  ist der kürzeste Weg von  $s$  zu diesem Knoten  $u$ .

# Dijkstras Algorithmus: Beobachtung

**Wichtige Annahme:** alle Gewichte sind positiv.



Kürzester Weg  $s \rightsquigarrow u$  hat Länge  $l$  (exakt).



**Obere Schranke:**

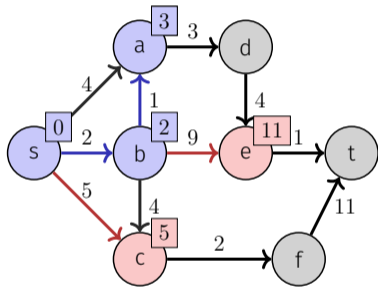
Kürzester Weg  $s \rightsquigarrow u$  hat Länge höchstens  $l$ .

**Beobachtung:** Kürzeste ausgehende Kante  $(s, u)$  ist der kürzeste Weg von  $s$  zu diesem Knoten  $u$ .



# Dijkstras Algorithmus: Beobachtung

**Wichtige Annahme:** alle Gewichte sind positiv.



Kürzester Weg  $s \rightsquigarrow u$  hat Länge  $l$  (exakt).

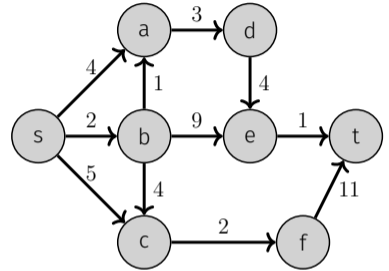


**Obere Schranke:**

Kürzester Weg  $s \rightsquigarrow u$  hat Länge höchstens  $l$ .

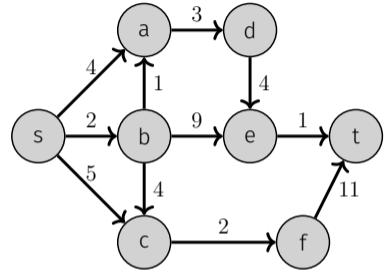
**Allgemeine Beobachtung:** Die kleinste obere Schranke eines (orangenen) Knotens  $u$  ist die exakte Länge des kürzesten Weges von  $s$  zu  $u$ .

# Dijkstras Algorithmus: Grundidee (Greedy)



# Dijkstras Algorithmus: Grundidee (Greedy)

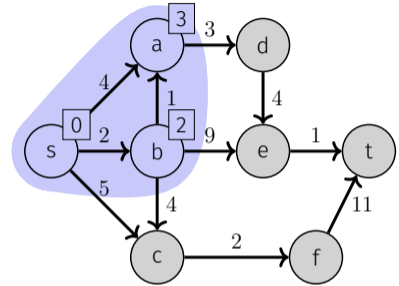
$V$  aufgeteilt in:



# Dijkstras Algorithmus: Grundidee (Greedy)

$V$  aufgeteilt in:

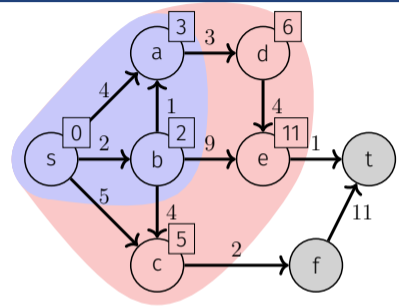
- **K**: Knoten mit bekanntem kürzestem Weg



# Dijkstras Algorithmus: Grundidee (Greedy)

$V$  aufgeteilt in:

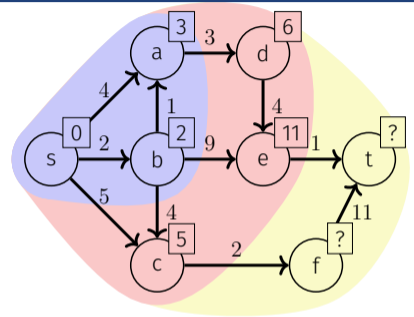
- $K$ : Knoten mit bekanntem kürzestem Weg
- $N = \bigcup_{v \in K} N^+(v) \setminus K$ : Nachfolger von  $K$   
 $\Rightarrow$  eine obere Schranke ist bekannt



# Dijkstras Algorithmus: Grundidee (Greedy)

$V$  aufgeteilt in:

- **K**: Knoten mit bekanntem kürzestem Weg
- **N** =  $\bigcup_{v \in K} N^+(v) \setminus K$ : Nachfolger von  $K$   
⇒ eine obere Schranke ist bekannt
- **R** =  $V \setminus (K \cup N)$ : restliche Knoten  
⇒ noch nichts bekannt



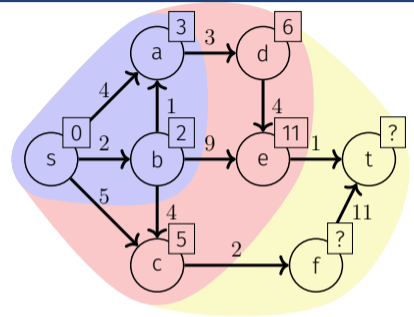
# Dijkstras Algorithmus: Grundidee (Greedy)

$V$  aufgeteilt in:

- **K**: Knoten mit bekanntem kürzestem Weg
- **N** =  $\bigcup_{v \in K} N^+(v) \setminus K$ : Nachfolger von  $K$   
⇒ eine obere Schranke ist bekannt
- **R** =  $V \setminus (K \cup N)$ : restliche Knoten  
⇒ noch nichts bekannt

## Greedy:

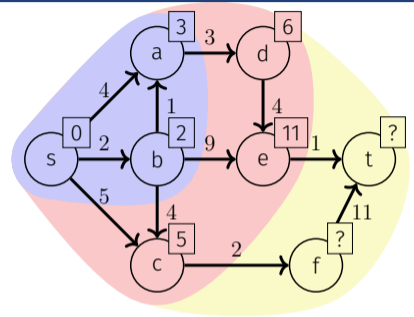
Startend mit **N** =  $\{s\}$ , bis **N** =  $\{\}$ : Knoten aus **N** mit kleinster oberer Schranke wird **K** und seine Nachbarn **N** hinzugefügt.



# Dijkstras Algorithmus: Grundidee (Greedy)

$V$  aufgeteilt in:

- **K**: Knoten mit bekanntem kürzestem Weg
- **N** =  $\bigcup_{v \in K} N^+(v) \setminus K$ : Nachfolger von  $K$   
⇒ eine obere Schranke ist bekannt
- **R** =  $V \setminus (K \cup N)$ : restliche Knoten  
⇒ noch nichts bekannt



## Greedy:

Startend mit  $\mathbf{N} = \{s\}$ , bis  $\mathbf{N} = \{\}$ : Knoten aus  $\mathbf{N}$  mit kleinster oberer Schranke wird  $\mathbf{K}$  und seine Nachbarn  $\mathbf{N}$  hinzugefügt.

## Invarianten:

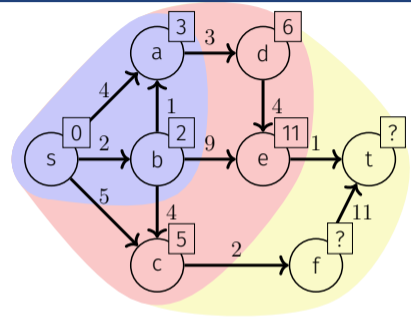
- nach  $i$  Schritten: kürzeste Wege zu  $i$  Knoten bekannt ( $|K| = i$ ).



# Dijkstras Algorithmus: Grundidee (Greedy)

$V$  aufgeteilt in:

- **K**: Knoten mit bekanntem kürzestem Weg
- **N** =  $\bigcup_{v \in K} N^+(v) \setminus K$ : Nachfolger von  $K$   
 $\Rightarrow$  eine obere Schranke ist bekannt
- **R** =  $V \setminus (K \cup N)$ : restliche Knoten  
 $\Rightarrow$  noch nichts bekannt



## Greedy:

Startend mit  $\mathbf{N} = \{s\}$ , bis  $\mathbf{N} = \{\}$ : Knoten aus  $\mathbf{N}$  mit kleinster oberer Schranke wird  $\mathbf{K}$  und seine Nachbarn  $\mathbf{N}$  hinzugefügt.

## Invarianten:

- nach  $i$  Schritten: kürzeste Wege zu  $i$  Knoten bekannt ( $|K| = i$ ).
- für alle Knoten  $v \in N$ : obere Schranke ist (exakte) Länge des kürzesten Weges  $s \rightsquigarrow \bullet \rightarrow v$  von  $s$  nach  $v$  mit Knoten nur aus  $K \cup \{v\}$ .

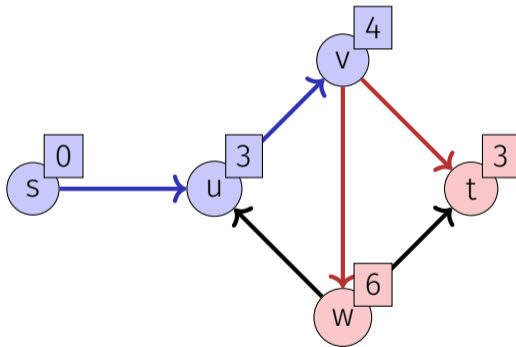
# Quiz

# Quiz

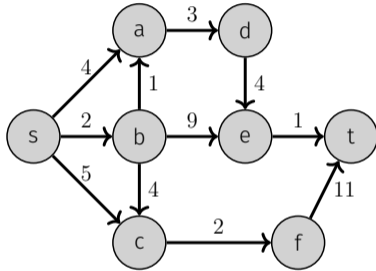
Ist die folgende Konstellation von oberen Schranken möglich?

# Quiz

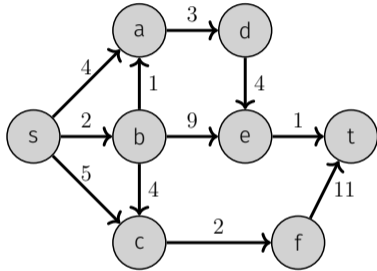
Ist die folgende Konstellation von oberen Schranken möglich?



# Beispiel

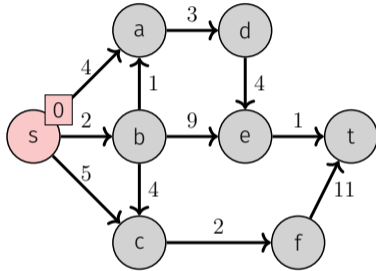


# Beispiel



K  
N  
R

# Beispiel

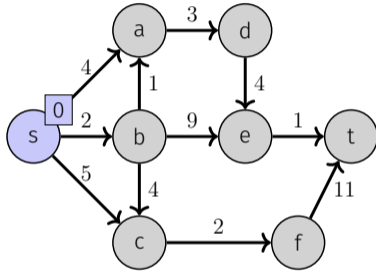


$$\mathbf{K} = \{\}$$

$$\mathbf{N} = \{s\}$$

$$\mathbf{R} = \{a, b, c, d, e, f, t\}$$

# Beispiel



Bekannte kürzeste Wege von  $s$ :

$s \rightsquigarrow s: 0$

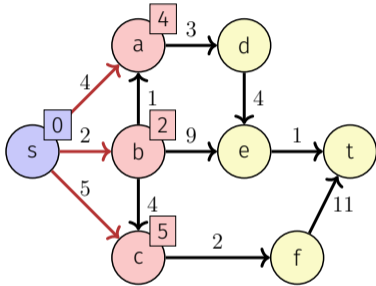
$$\mathbf{K} = \{s\}$$

$$\mathbf{N} = \{\}$$

$$\mathbf{R} = \{a, b, c, d, e, f, t\}$$



# Beispiel



$$\mathbf{K} = \{s\}$$

$$\mathbf{N} = \{a, b, c\}$$

$$\mathbf{R} = \{d, e, f, t\}$$

**Bekannte kürzeste Wege von  $s$ :**

$$s \rightsquigarrow s: 0$$

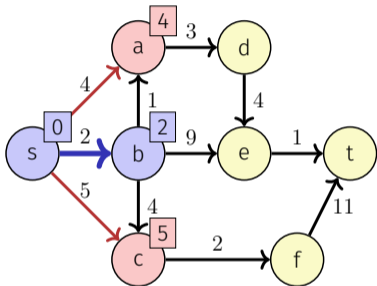
**Ausgehende Kanten:**

$$s \rightarrow a: 4$$

$$s \rightarrow b: 2$$

$$s \rightarrow c: 5$$

# Beispiel



$$\mathbf{K} = \{s, b\}$$

$$\mathbf{N} = \{a, c\}$$

$$\mathbf{R} = \{d, e, f, t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

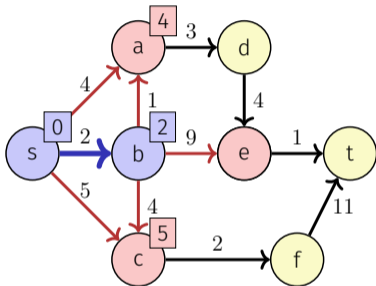
## Ausgehende Kanten:

$$s \rightarrow a: 4$$

$$s \rightarrow b: 2$$

$$s \rightarrow c: 5$$

# Beispiel



$$\mathbf{K} = \{s, b\}$$

$$\mathbf{N} = \{a, c, e\}$$

$$\mathbf{R} = \{d, f, t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

## Ausgehende Kanten:

$$s \rightarrow a: 4$$

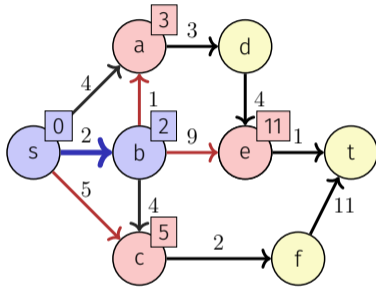
$$s \rightarrow c: 5$$

$$s \rightarrow b \rightarrow a: 3$$

$$s \rightarrow b \rightarrow e: 11$$

$$s \rightarrow b \rightarrow c: 6$$

# Beispiel



$$\mathbf{K} = \{s, b\}$$

$$\mathbf{N} = \{a, c, e\}$$

$$\mathbf{R} = \{d, f, t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

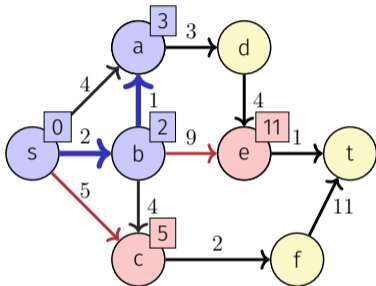
## Ausgehende Kanten:

$$s \rightarrow c: 5$$

$$s \rightarrow b \rightarrow a: 3$$

$$s \rightarrow b \rightarrow e: 11$$

# Beispiel



$$\mathbf{K} = \{s, b, a\}$$

$$\mathbf{N} = \{c, e\}$$

$$\mathbf{R} = \{d, f, t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

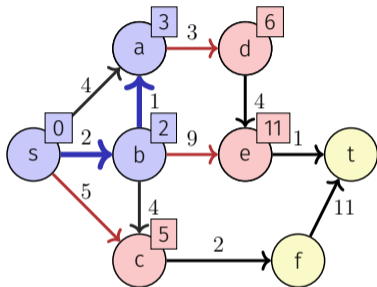
## Ausgehende Kanten:

$$s \rightarrow c: 5$$

$$s \rightarrow b \rightarrow a: 3$$

$$s \rightarrow b \rightarrow e: 11$$

# Beispiel



$$\mathbf{K} = \{s, b, a\}$$

$$\mathbf{N} = \{c, e, d\}$$

$$\mathbf{R} = \{f, t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

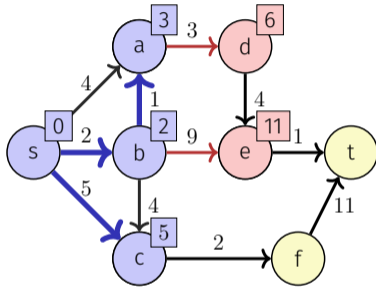
## Ausgehende Kanten:

$$s \rightarrow c: 5$$

$$s \rightarrow b \rightarrow a \rightarrow d: 6$$

$$s \rightarrow b \rightarrow e: 11$$

# Beispiel



$$\mathbf{K} = \{s, b, a, c\}$$

$$\mathbf{N} = \{e, d, f\}$$

$$\mathbf{R} = \{f, t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow c: 5$$

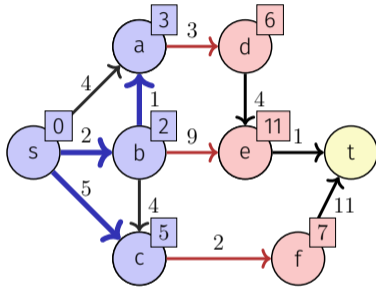
## Ausgehende Kanten:

$$s \rightarrow c: 5$$

$$s \rightarrow b \rightarrow a \rightarrow d: 6$$

$$s \rightarrow b \rightarrow e: 11$$

# Beispiel



$$\mathbf{K} = \{s, b, a, c\}$$

$$\mathbf{N} = \{e, d, f\}$$

$$\mathbf{R} = \{t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow c: 5$$

## Ausgehende Kanten:

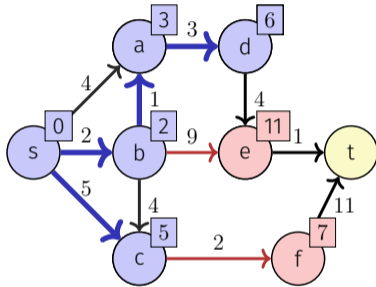
$$s \rightarrow b \rightarrow a \rightarrow d: 6$$

$$s \rightarrow b \rightarrow e: 11$$

$$s \rightarrow c \rightarrow f: 7$$



# Beispiel



$$\mathbf{K} = \{s, b, a, c, d\}$$

$$\mathbf{N} = \{e, f\}$$

$$\mathbf{R} = \{t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow d: 6$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow c: 5$$

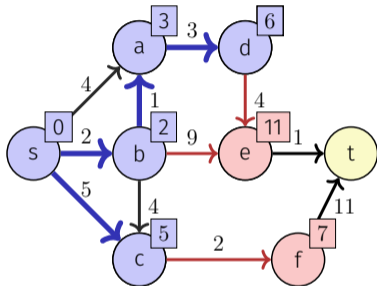
## Ausgehende Kanten:

$$s \rightarrow b \rightarrow a \rightarrow d: 6$$

$$s \rightarrow b \rightarrow e: 11$$

$$s \rightarrow c \rightarrow f: 7$$

# Beispiel



$$\mathbf{K} = \{s, b, a, c, d\}$$

$$\mathbf{N} = \{e, f\}$$

$$\mathbf{R} = \{t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0 \qquad s \rightsquigarrow d: 6$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow c: 5$$

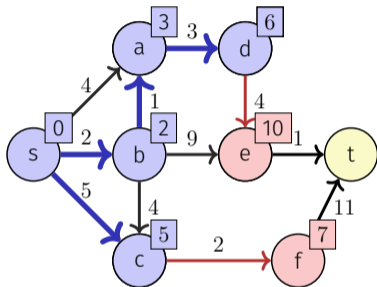
## Ausgehende Kanten:

$$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e: 10$$

$$s \rightarrow b \rightarrow e: 11$$

$$s \rightarrow c \rightarrow f: 7$$

# Beispiel



$$\mathbf{K} = \{s, b, a, c, d\}$$

$$\mathbf{N} = \{e, f\}$$

$$\mathbf{R} = \{t\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow d: 6$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

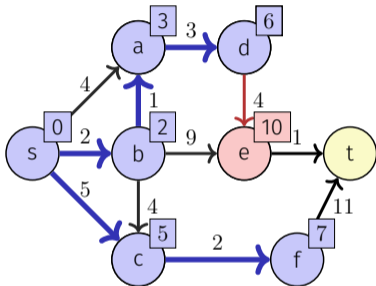
$$s \rightsquigarrow c: 5$$

## Ausgehende Kanten:

$$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e: 10$$

$$s \rightarrow c \rightarrow f: 7$$

# Beispiel



**K** = {s, b, a, c, d, f}

**N** = {e}

**R** = {t}

## Bekannte kürzeste Wege von s:

$s \rightsquigarrow s: 0$

$s \rightsquigarrow d: 6$

$s \rightsquigarrow b: 2$

$s \rightsquigarrow f: 7$

$s \rightsquigarrow a: 3$

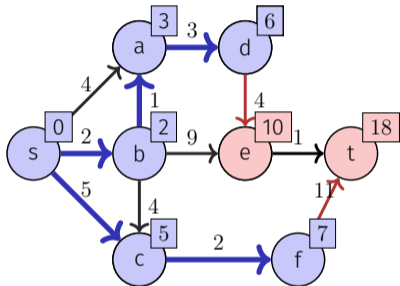
$s \rightsquigarrow c: 5$

## Ausgehende Kanten:

$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e: 10$

$s \rightarrow c \rightarrow f: 7$

# Beispiel



$\mathbf{K} = \{s, b, a, c, d, f\}$

$\mathbf{N} = \{e, t\}$

$\mathbf{R} = \{\}$

## Bekannte kürzeste Wege von $s$ :

$s \rightsquigarrow s: 0$

$s \rightsquigarrow d: 6$

$s \rightsquigarrow b: 2$

$s \rightsquigarrow f: 7$

$s \rightsquigarrow a: 3$

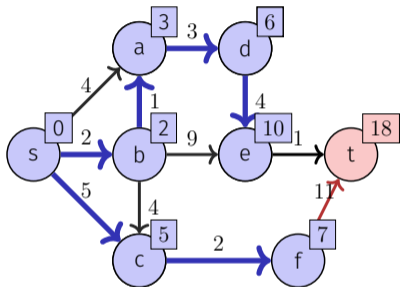
$s \rightsquigarrow c: 5$

## Ausgehende Kanten:

$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e: 10$

$s \rightarrow c \rightarrow f \rightarrow t: 18$

# Beispiel



## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow d: 6$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow f: 7$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow e: 10$$

$$s \rightsquigarrow c: 5$$

## Ausgehende Kanten:

$$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e: 10$$

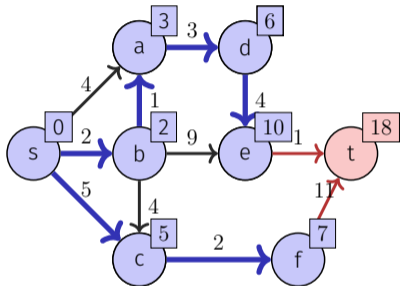
$$s \rightarrow c \rightarrow f \rightarrow t: 18$$

$$\mathbf{K} = \{s, b, a, c, d, f, e\}$$

$$\mathbf{N} = \{t\}$$

$$\mathbf{R} = \{\}$$

# Beispiel



## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow d: 6$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow f: 7$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow e: 10$$

$$s \rightsquigarrow c: 5$$

## Ausgehende Kanten:

$$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e \rightarrow t: 11$$

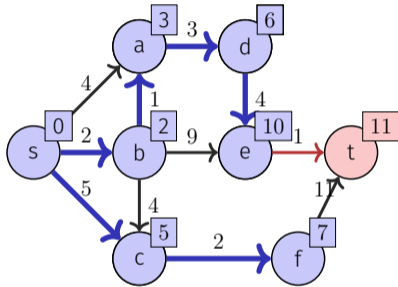
$$s \rightarrow c \rightarrow f \rightarrow t: 18$$

$$\mathbf{K} = \{s, b, a, c, d, f, e\}$$

$$\mathbf{N} = \{t\}$$

$$\mathbf{R} = \{\}$$

# Beispiel



$$\mathbf{K} = \{s, b, a, c, d, f, e\}$$

$$\mathbf{N} = \{t\}$$

$$\mathbf{R} = \{\}$$

## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow d: 6$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow f: 7$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow e: 10$$

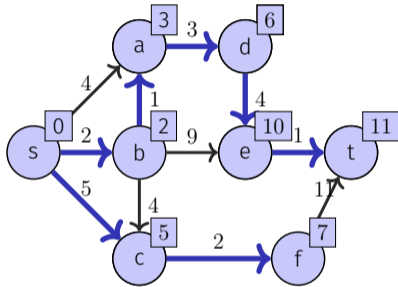
$$s \rightsquigarrow c: 5$$

## Ausgehende Kanten:

$$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e \rightarrow t: 11$$



# Beispiel



## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow d: 6$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow f: 7$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow e: 10$$

$$s \rightsquigarrow c: 5$$

$$s \rightsquigarrow t: 11$$

## Ausgehende Kanten:

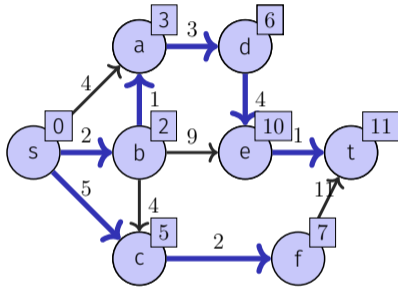
$$s \rightarrow b \rightarrow a \rightarrow d \rightarrow e \rightarrow t: 11$$

$$\mathbf{K} = \{s, b, a, c, d, f, e, t\}$$

$$\mathbf{N} = \{\}$$

$$\mathbf{R} = \{\}$$

# Beispiel



## Bekannte kürzeste Wege von $s$ :

$$s \rightsquigarrow s: 0$$

$$s \rightsquigarrow b: 2$$

$$s \rightsquigarrow a: 3$$

$$s \rightsquigarrow c: 5$$

$$s \rightsquigarrow d: 6$$

$$s \rightsquigarrow f: 7$$

$$s \rightsquigarrow e: 10$$

$$s \rightsquigarrow t: 11$$

## Ausgehende Kanten:

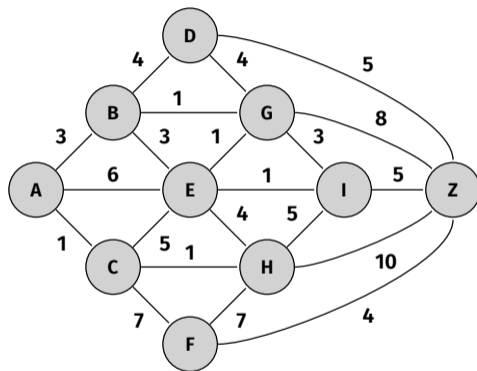
$$\mathbf{K} = \{s, b, a, c, d, f, e, t\}$$

$$\mathbf{N} = \{\}$$

$$\mathbf{R} = \{\}$$

# Quiz

# Quiz



Welche Knoten sind in  $K$  (bekannte kürzeste Wege) nach sechs Schritten von Dijkstras Algorithmus mit Startknoten A?

# Zutaten für einen Algorithmus

Gesucht: Kürzeste Wege von einem Startknoten  $s$  aus.

- Gewicht des kürzesten bisher gefundenen Pfades

$$d_s : V \rightarrow \mathbb{R}$$

**Zu Beginn:**  $d_s[v] = \infty$  für alle Knoten  $v \in V$ .

**Ziel:**  $d_s[v] = \delta(s, v)$  für alle  $v \in V$ .

- Vorgänger eines Knotens

$$\pi_s : V \rightarrow V$$

Zu Beginn  $\pi_s[v]$  undefiniert für jeden Knoten  $v \in V$

# Algorithmus: Dijkstra( $G, s$ )

# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$

# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0; N \leftarrow \{s\}$

# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

[

# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

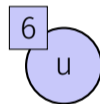
**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0; N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]; N \leftarrow N \setminus \{u\}$



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

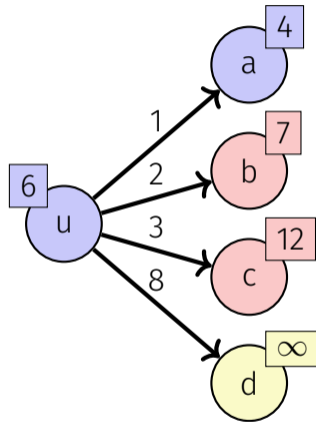
$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

**foreach**  $v \in N^+(u)$  **do**



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

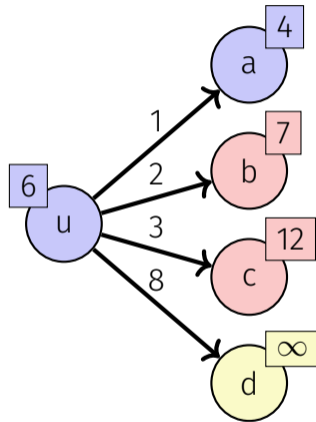
$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

**foreach**  $v \in N^+(u)$  **do**

**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

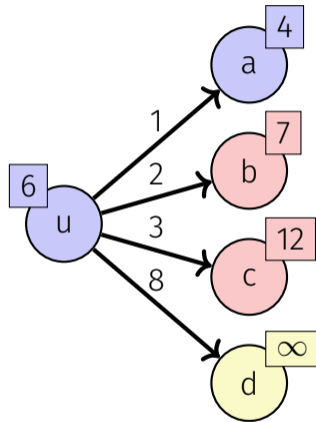
$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

**foreach**  $v \in N^+(u)$  **do**

**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**

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

$\pi_s[v] \leftarrow u$



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

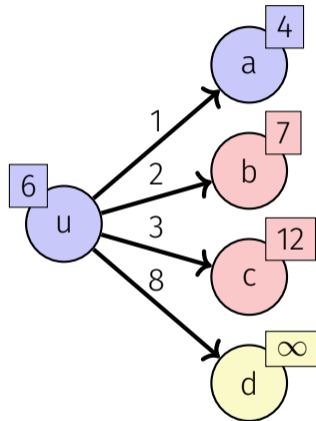
**foreach**  $v \in N^+(u)$  **do**

**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**

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

$\pi_s[v] \leftarrow u$

$N \leftarrow N \cup \{v\}$





# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

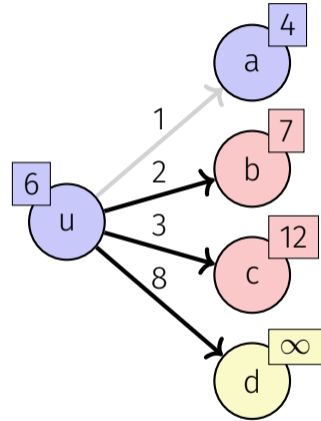
**foreach**  $v \in N^+(u)$  **do**

**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**

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

$\pi_s[v] \leftarrow u$

$N \leftarrow N \cup \{v\}$



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

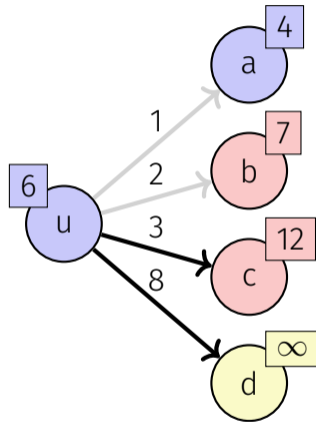
**foreach**  $v \in N^+(u)$  **do**

**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**

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

$\pi_s[v] \leftarrow u$

$N \leftarrow N \cup \{v\}$



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

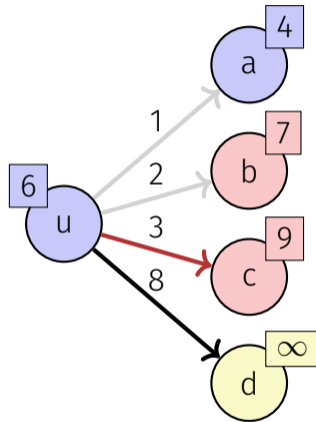
**foreach**  $v \in N^+(u)$  **do**

**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**

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

$\pi_s[v] \leftarrow u$

$N \leftarrow N \cup \{v\}$



# Algorithmus: Dijkstra( $G, s$ )

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ ,  
Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und  
Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$u \leftarrow \arg \min_{u \in N} d_s[u]$ ;  $N \leftarrow N \setminus \{u\}$

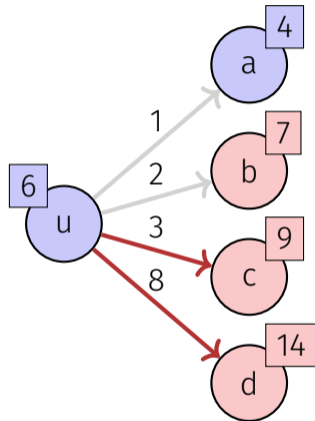
**foreach**  $v \in N^+(u)$  **do**

**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**

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

$\pi_s[v] \leftarrow u$

$N \leftarrow N \cup \{v\}$



# Zur Implementation: Datenstruktur für $N$ ?

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert( $(p, k)$ ):**  
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert((p, k)):**  
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin():**  
Entfernen des Elements mit kleinstem  
Wert



# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

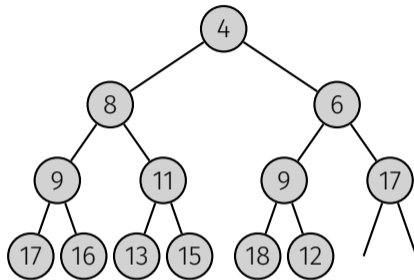
- **Insert( $(p, k)$ ):**  
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin():**  
Entfernen des Elements mit kleinstem  
Wert

⇒ MinHeap

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert((p, k)):**  
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin():**  
Entfernen des Elements mit kleinstem  
Wert

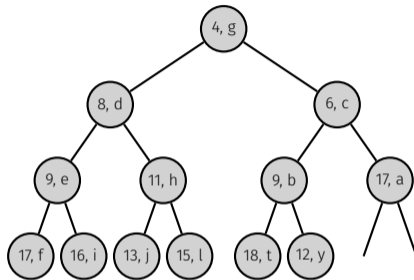


⇒ MinHeap

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert((p, k)):**  
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin():**  
Entfernen des Elements mit kleinstem  
Wert

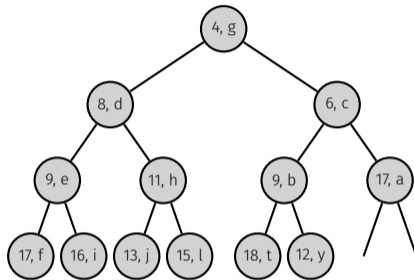


⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert((p, k)):**  
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin():**  
Entfernen des Elements mit kleinstem  
Wert

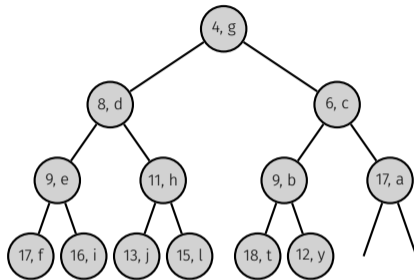


⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin**():  
Entfernen des Elements mit kleinstem  
Wert

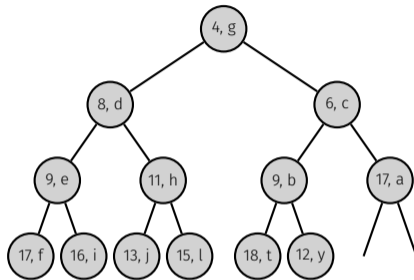


⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin**():  $\mathcal{O}(\log |V|)$   
Entfernen des Elements mit kleinstem  
Wert

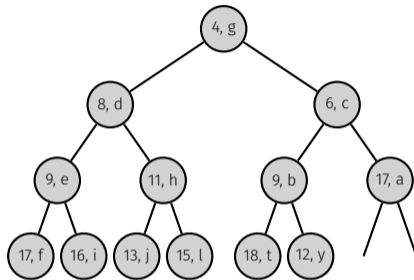


⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin**():  $\mathcal{O}(\log |V|)$   
Entfernen des Elements mit kleinstem  
Wert
- **DecreaseKey**(( $p$ ,  $k$ )):  
Updaten des Werts von Schlüssel  $k$  zu  $p$

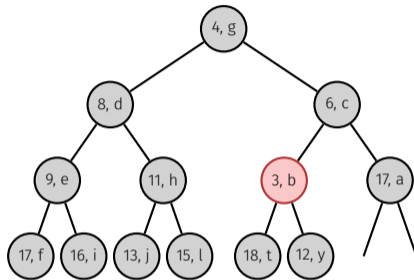


⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin**():  $\mathcal{O}(\log |V|)$   
Entfernen des Elements mit kleinstem  
Wert
- **DecreaseKey**(( $p$ ,  $k$ )):  
Updaten des Werts von Schlüssel  $k$  zu  $p$



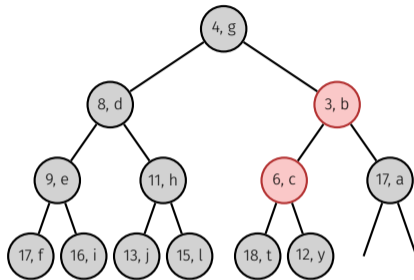
⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert



# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin**():  $\mathcal{O}(\log |V|)$   
Entfernen des Elements mit kleinstem  
Wert
- **DecreaseKey**(( $p$ ,  $k$ )):  
Updaten des Werts von Schlüssel  $k$  zu  $p$

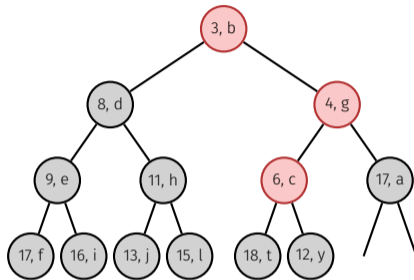


⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin**():  $\mathcal{O}(\log |V|)$   
Entfernen des Elements mit kleinstem  
Wert
- **DecreaseKey**(( $p$ ,  $k$ )):  
Updaten des Werts von Schlüssel  $k$  zu  $p$

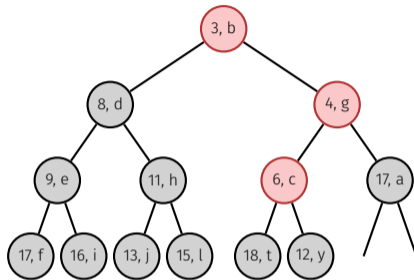


⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# Zur Implementation: Datenstruktur für $N$ ?

Benötigte Operationen:

- **Insert**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Hinzufügen eines Schlüssels (Knoten)  $k$   
mit Wert (obere Schranke)  $p$
- **ExtractMin**():  $\mathcal{O}(\log |V|)$   
Entfernen des Elements mit kleinstem  
Wert
- **DecreaseKey**(( $p$ ,  $k$ )):  $\mathcal{O}(\log |V|)$   
Updaten des Werts von Schlüssel  $k$  zu  $p$



⇒ MinHeap mit Knoten in  $N$  als Schlüssel und mit oberer Schranke als Wert

# DecreaseKey

# DecreaseKey

Zwei Möglichkeiten:

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion

**Lazy Deletion:**



# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion

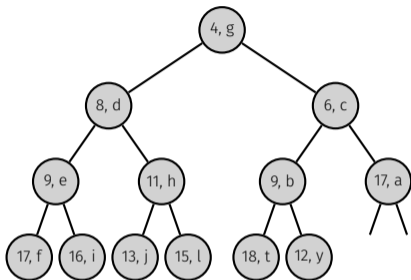
## **Lazy Deletion:**

- Knoten mit kleinerer oberer Schranke erneut einfügen

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



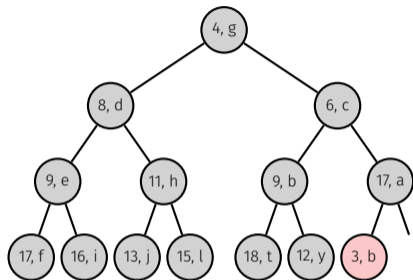
## Lazy Deletion:

- Knoten mit kleinerer oberer Schranke erneut einfügen

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



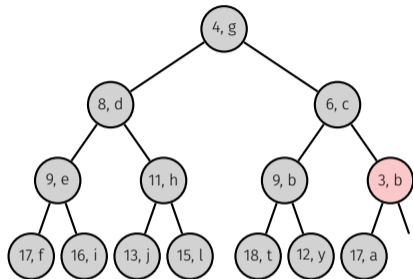
## Lazy Deletion:

- Knoten mit kleinerer oberer Schranke erneut einfügen

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



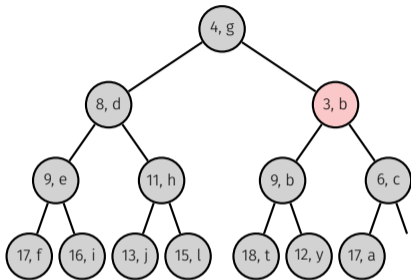
## Lazy Deletion:

- Knoten mit kleinerer oberer Schranke erneut einfügen

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



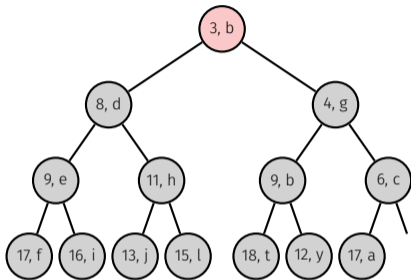
## Lazy Deletion:

- Knoten mit kleinerer oberer Schranke erneut einfügen

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



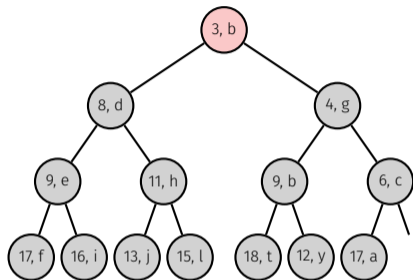
## Lazy Deletion:

- Knoten mit kleinerer oberer Schranke erneut einfügen

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



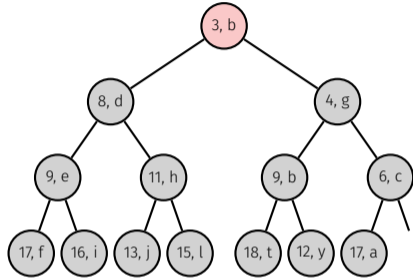
## Lazy Deletion:

- Knoten mit kleinerer oberer Schranke erneut einfügen

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



## Lazy Deletion:

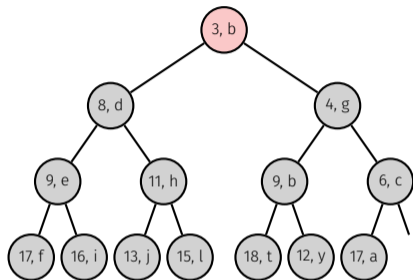
- Knoten mit kleinerer oberer Schranke erneut einfügen
- Knoten beim Entnehmen als "deleted" kennzeichnen



# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



## Lazy Deletion:

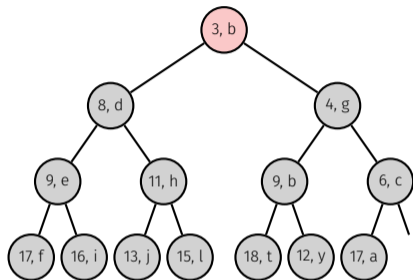
- Knoten mit kleinerer oberer Schranke erneut einfügen
- Knoten beim Entnehmen als "deleted" kennzeichnen

⇒ Speicherbedarf vom Heap wächst bis zu  $\Theta(|E|)$  statt  $\Theta(|V|)$

# DecreaseKey

Zwei Möglichkeiten:

- Merken der Position:  
Speichern am Knoten oder extern
- oder DecreaseKey vermeiden:  
mit Lazy Deletion



## Lazy Deletion:

- Knoten mit kleinerer oberer Schranke erneut einfügen
- Knoten beim Entnehmen als "deleted" kennzeichnen

⇒ Speicherbedarf vom Heap wächst bis zu  $\Theta(|E|)$  statt  $\Theta(|V|)$

⇒ Da  $|E| \leq |V|^2$ : Insert und ExtractMin in  $\mathcal{O}(\log |V|^2) = \mathcal{O}(\log |V|)$

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

  |

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$

$K = \{ \}$ ;  $d_s[s] \leftarrow 0; N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

    [ ]



# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

$K \leftarrow K \cup \{u\}$

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

$K \leftarrow K \cup \{u\}$

**foreach**  $v \in N^+(u)$  **do**

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

$K \leftarrow K \cup \{u\}$

**foreach**  $v \in N^+(u)$  **do**

**if**  $d + c(u, v) < d_s[v]$  **then**

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

$K \leftarrow K \cup \{u\}$

**foreach**  $v \in N^+(u)$  **do**

**if**  $d + c(u, v) < d_s[v]$  **then**

$d_s[v] \leftarrow d + c(u, v)$ ;  $\pi_s[v] \leftarrow u$

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

$K \leftarrow K \cup \{u\}$

**foreach**  $v \in N^+(u)$  **do**

**if**  $d + c(u, v) < d_s[v]$  **then**

$d_s[v] \leftarrow d + c(u, v)$ ;  $\pi_s[v] \leftarrow u$

                Insert( $(d + c(u, v), v)$ )

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

$K \leftarrow K \cup \{u\}$

**foreach**  $v \in N^+(u)$  **do**

**if**  $d + c(u, v) < d_s[v]$  **then**

$d_s[v] \leftarrow d + c(u, v)$ ;  $\pi_s[v] \leftarrow u$

                Insert( $(d + c(u, v), v)$ )

# Algorithmus: Dijkstra( $G, s$ ) mit Lazy Deletion

**Input:** Positiv gewichteter Graph  $G = (V, E, c)$ , Startpunkt  $s \in V$

**Output:** Länge  $d_s$  der kürzesten Pfade von  $s$  und Vorgänger  $\pi_s$  für jeden Knoten

**foreach**  $u \in V$  **do**

$d_s[u] \leftarrow \infty$ ;  $\pi_s[u] \leftarrow \text{null}$

$K = \{\}$ ;  $d_s[s] \leftarrow 0$ ;  $N \leftarrow \{s\}$

**while**  $N \neq \emptyset$  **do**

$d, u \leftarrow \text{ExtractMin}(N)$

**if**  $u \notin K$  **then**

$K \leftarrow K \cup \{u\}$

**foreach**  $v \in N^+(u)$  **do**

**if**  $d + c(u, v) < d_s[v]$  **then**

$d_s[v] \leftarrow d + c(u, v)$ ;  $\pi_s[v] \leftarrow u$

                Insert( $(d + c(u, v), v)$ )

**Laufzeit:**

Initialisierung:  $\mathcal{O}(|V|)$

$(|V| + |E|)$ -mal ExtractMin:  $\mathcal{O}((|V| + |E|) \cdot \log |V|)$ ;

$(|E| + 1)$ -mal Insert:  $\mathcal{O}(|E| \cdot \log |V|)$ ;

$\Rightarrow$  Insgesamt:  $\mathcal{O}((|V| + |E|) \cdot \log |V|)$

# Laufzeit von Dijkstra (ohne Lazy Deletion)

- $|V| \times \text{ExtractMin}$ :  $\mathcal{O}(|V| \log |V|)$
- $|E| \times \text{Insert oder DecreaseKey}$ :  $\mathcal{O}(|E| \log |V|)$
- $1 \times \text{Init}$ :  $\mathcal{O}(|V|)$
- Insgesamt<sup>41 42</sup> :  
$$\mathcal{O}((|V| + |E|) \log |V|)$$

---

<sup>41</sup>Für zusammenhängende Graphen:  $\mathcal{O}(|E| \log |V|)$

<sup>42</sup>Kann verbessert werden unter Verwendung einer für ExtractMin und DecreaseKey optimierten Datenstruktur (Fibonacci Heap), dann Laufzeit  $\mathcal{O}(|E| + |V| \log |V|)$ .



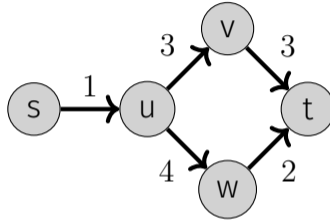
# Beobachtungen

# Beobachtungen

- Ist der kürzeste Weg immer eindeutig?

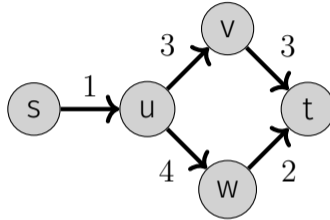
# Beobachtungen

- Ist der kürzeste Weg immer eindeutig? Nein!



# Beobachtungen

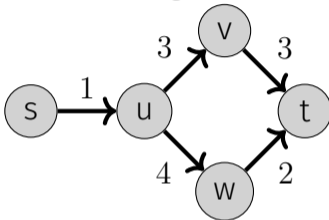
- Ist der kürzeste Weg immer eindeutig? Nein!



Dijkstras Algorithmus findet (irgend)einen kürzesten Weg.

# Beobachtungen

- Ist der kürzeste Weg immer eindeutig? Nein!

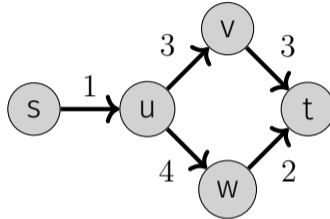


Dijkstras Algorithmus findet (irgend)einen kürzesten Weg.

- Existiert immer (mindestens) ein kürzester Weg?

# Beobachtungen

- Ist der kürzeste Weg immer eindeutig? Nein!

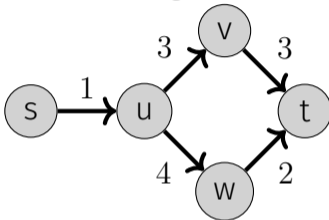


Dijkstras Algorithmus findet (irgend)einen kürzesten Weg.

- Existiert immer (mindestens) ein kürzester Weg? Nein! Negative Zyklen!

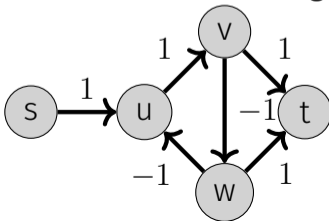
# Beobachtungen

- Ist der kürzeste Weg immer eindeutig? Nein!



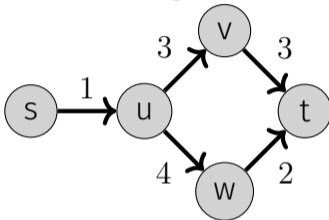
Dijkstras Algorithmus findet (irgend)einen kürzesten Weg.

- Existiert immer (mindestens) ein kürzester Weg? Nein! Negative Zyklen!



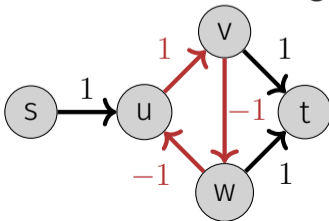
# Beobachtungen

- Ist der kürzeste Weg immer eindeutig? Nein!



Dijkstras Algorithmus findet (irgend)einen kürzesten Weg.

- Existiert immer (mindestens) ein kürzester Weg? Nein! Negative Zyklen!





## 26.3 Allgemeiner Algorithmus

---

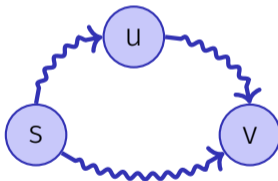
Warum Dijkstra korrekt ist und wie man verallgemeinert.

# Beobachtungen (1)

## Dreiecksungleichung

Für alle  $s, u, v \in V$ :

$$\delta(s, v) \leq \delta(s, u) + \delta(u, v)$$

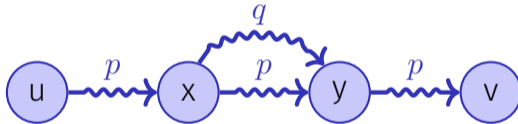


Ein kürzester Weg von  $s$  nach  $v$  (ohne weitere Einschränkungen) kann nicht länger sein als ein kürzester Weg von  $s$  nach  $v$ , der  $u$  enthalten muss.

# Beobachtungen (2)

## Optimale Substruktur

Teilpfade von kürzesten Pfaden sind kürzeste Pfade: Sei  $p = \langle v_0, \dots, v_k \rangle$  ein kürzester Pfad von  $v_0$  nach  $v_k$ . Dann ist jeder der Teilpfade  $p_{ij} = \langle v_i, \dots, v_j \rangle$  ( $0 \leq i < j \leq k$ ) ein kürzester Pfad von  $v_i$  nach  $v_j$ .



Wäre das nicht so, könnte man einen der Teilpfade kürzen, Widerspruch zur Voraussetzung.

# Beobachtungen (3)

Kürzeste Wege enthalten keine Zyklen

1. Kürzester Weg enthält negativen Zyklus: es existiert kein kürzester Weg. Widerspruch.
2. Weg enthält positiven Zyklus: Weglassen des positiven Zyklus kann den Weg verkürzen: Widerspruch
3. Weg enthält Zyklus vom Gewicht 0: Weglassen des Zyklus verändert das Pfadgewicht nicht. Weglassen (Konvention).

# Allgemeiner Algorithmus

1. Initialisiere  $d_s$  und  $\pi_s$ :  $d_s[v] = \infty$ ,  $\pi_s[v] = \text{null}$  für alle  $v \in V$
2. Setze  $d_s[s] \leftarrow 0$
3. Wähle eine Kante  $(u, v) \in E$

**Relax**( $u, v$ ):

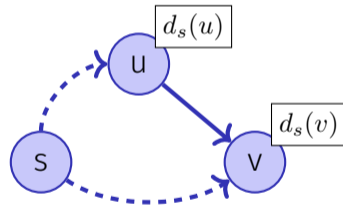
**if**  $d_s[u] + c(u, v) < d_s[v]$  **then**

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

$\pi_s[v] \leftarrow u$

**return true**

**return false**



4. Wiederhole 3 bis nichts mehr relaxiert werden kann.  
(bis  $d_s[v] \leq d_s[u] + c(u, v) \quad \forall (u, v) \in E$ )

# Relaxieren ist sicher

Zu jeder Zeit gilt in obigem Algorithmus

$$d_s[v] \geq \delta(s, v) \quad \forall v \in V$$

# Relaxieren ist sicher

Zu jeder Zeit gilt in obigem Algorithmus

$$d_s[v] \geq \delta(s, v) \quad \forall v \in V$$

Im Relaxierschritt:

$$\delta(s, v) \leq \delta(s, u) + \delta(u, v) \quad \text{[Dreiecksungleichung].}$$

$$\delta(s, u) \leq d_s[u] \quad \text{[Induktionsvoraussetzung].}$$

$$\delta(u, v) \leq c(u, v) \quad \text{[Minimalität von } \delta \text{]}$$

$$\Rightarrow d_s[u] + c(u, v) \geq \delta(s, v)$$

$$\Rightarrow \min\{d_s[v], d_s[u] + c(u, v)\} \geq \delta(s, v)$$

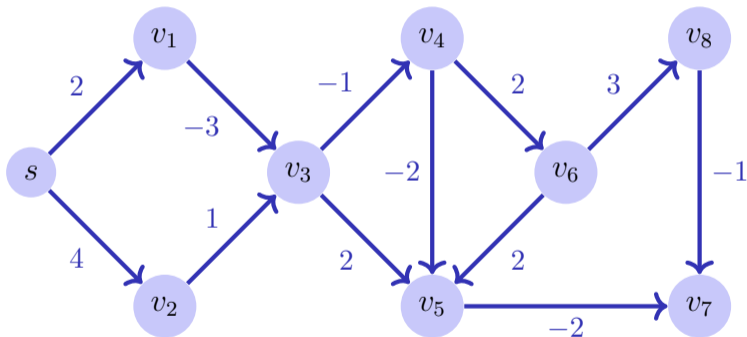
# Zentrale Frage

Wie / in welcher Reihenfolge wählt man die Kanten in obigem Algorithmus?



# Spezialfall: Gerichteter Azyklischer Graph (DAG)

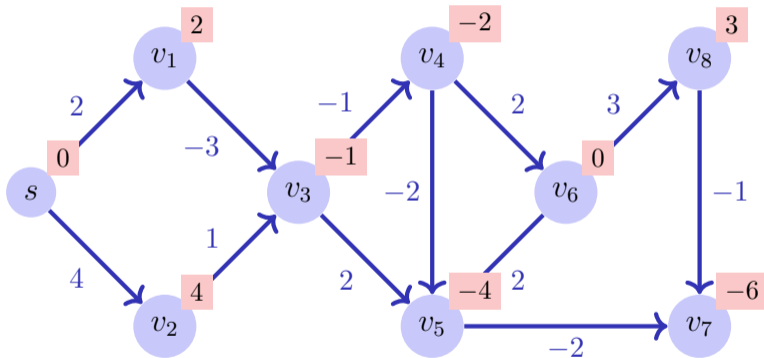
DAG  $\Rightarrow$  Topologische Sortierung liefert optimale Besuchsreihenfolge



Top. Sortieren:  $\Rightarrow$  Reihenfolge  $s, v_1, v_2, v_3, v_4, v_6, v_5, v_8, v_7$ .

# Spezialfall: Gerichteter Azyklischer Graph (DAG)

DAG  $\Rightarrow$  Topologische Sortierung liefert optimale Besuchsreihenfolge



Top. Sortieren:  $\Rightarrow$  Reihenfolge  $s, v_1, v_2, v_3, v_4, v_6, v_5, v_8, v_7$ .

# Andere Fälle

- Spezialfall:  $c \equiv 1 \Rightarrow$  BFS
- Spezialfall: Positive Kantengewichte  $\Rightarrow$  Dijkstra 😊.
- Allgemeine Bewertete Graphen: Zyklen mit negativen Gewichten können Weg verkürzen: es muss keinen kürzesten Weg mehr geben

# Dynamic Programming Ansatz (Bellman)

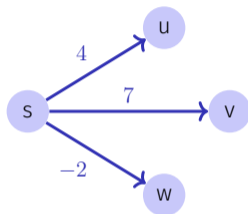
Induktion über Anzahl Kanten.  $d_s[i, v]$ : Kürzeste Weglänge von  $s$  nach  $v$  über maximal  $i$  Kanten.

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

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

# Dynamic Programming Ansatz (Bellman)

	$s$	$\dots$	$v$	$\dots$	$w$
0	0	$\infty$	$\infty$	$\infty$	$\infty$
1	0	$\infty$	7	$\infty$	-2
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
$n - 1$	0	$\dots$	$\dots$	$\dots$	$\dots$



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.

**foreach**  $u \in V$  **do**

┌  $d_s[u] \leftarrow \infty; \pi_s[u] \leftarrow \text{null}$

$d_s[s] \leftarrow 0;$

**for**  $i \leftarrow 1$  **to**  $|V|$  **do**

┌  $f \leftarrow \text{false}$

┌ **foreach**  $(u, v) \in E$  **do**

┌┌  $f \leftarrow f \vee \text{Relax}(u, v)$

┌┌ **if**  $f = \text{false}$  **then return true**

**return false;**

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