

# Informatik II - SS 2016

## (Algorithmen & Datenstrukturen)

Vorlesung 17 (22.6.2016)

### Graphenalgorithmen V (Kürzeste Wege)



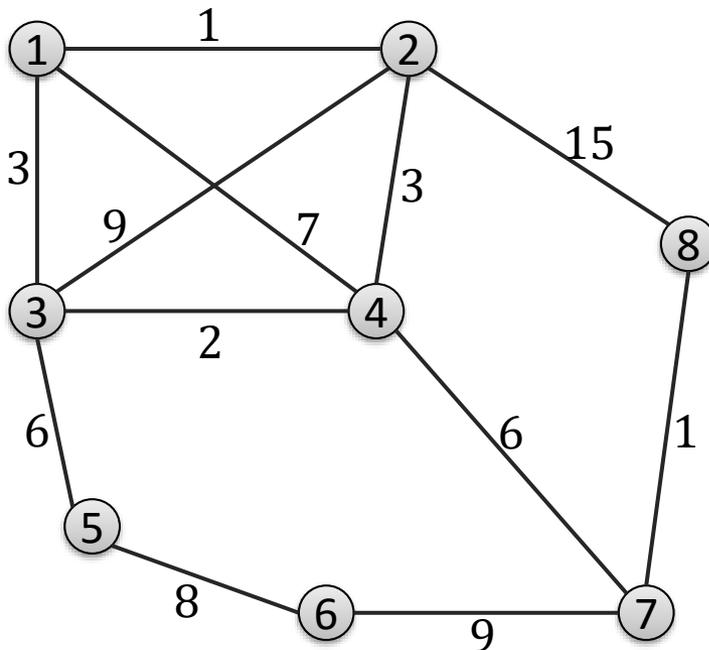
**UNI  
FREIBURG**

Fabian Kuhn

Algorithmen und Komplexität

## Problem

- Gegeben: gewichteter Graph  $G = (V, E, w)$ , Startknoten  $s \in V$ 
  - Wir bezeichnen Gewicht einer Kante  $(u, v)$  als  $w(u, v)$
  - Annahme:  $\forall e \in E: w(e) \geq 0$
- Ziel: Finde kürzeste Pfade / Distanzen von  $s$  zu allen Knoten
  - Distanz von  $s$  zu  $v$ :  $d_G(s, v)$  (Länge eines kürzesten Pfades)

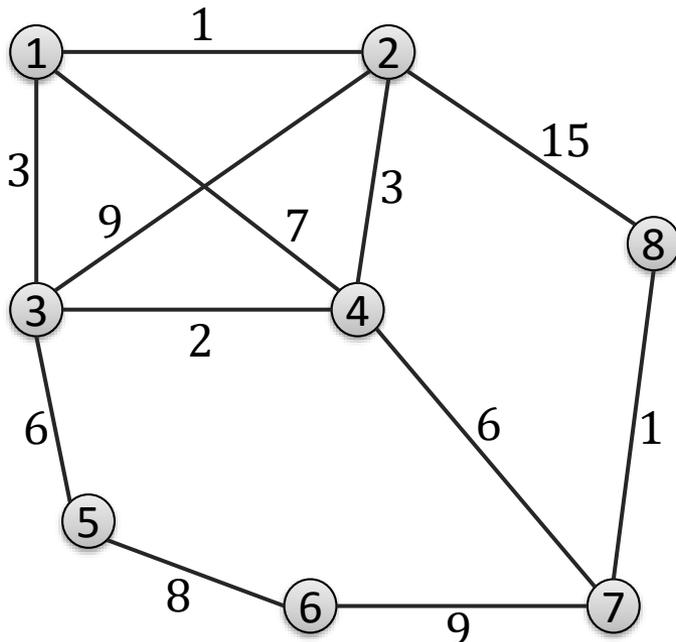


**Lemma:** Falls  $v_0, v_1, \dots, v_k$  ein kürzester Pfad von  $v_0$  nach  $v_k$  ist, dann gilt für alle  $0 \leq i \leq j \leq k$ , dass der Teilpfad  $v_i, v_{i+1}, \dots, v_j$  ein kürzester Pfad von  $v_i$  nach  $v_j$  ist.

- gilt auch bei negativen Kantengewichten...

# Shortest-Path Tree

- Im Knoten  $s$  gewurzelter Spannbaum, welcher kürzeste Pfade von  $s$  zu allen Knoten enthält.
  - Einen solchen Baum gibt es immer (folgt insb. aus der Opt. der Teilpfade)
- Bei ungewichteten Graphen: BFS-Spannbaum
- **Ziel:** Finde einen “Shortest Path Tree”



- Algorithmus von Edsger W. Dijkstra (1959 publiziert)

## Idee:

- Wir starten bei  $s$  und bauen schrittweise den Spannbaum auf

### Invariante:

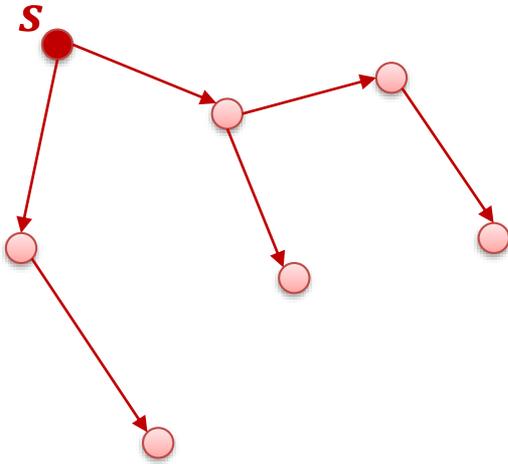
Algorithmus hat zu jeder Zeit einen bei  $s$  gewurzelten Teilbaum eines “Shortest Path Tree”.

- Ziel: In jedem Schritt des Algorithmus einen Knoten hinzufügen
  - Am Anfang: Teilbaum besteht nur aus  $s$  (erfüllt Invariante trivialerweise...)
  - 1. Schritt: Wegen der Optimalität der Teilpfade, muss es einen kürzesten Pfad bestehend aus nur einer Kante geben...
  - Füge Knoten mit kleinstem Abstand zu  $s$  zum Baum hinzu

# Dijkstras Algorithmus: Ein Schritt

**Gegeben:** Einen in  $s$  gewurzelten  $T$ , so dass  $T$  Teilbaum eines “Shortest Path Tree” von  $s$  in  $G$  ist.

Wie können wir  $T$  um einen Knoten erweitern?

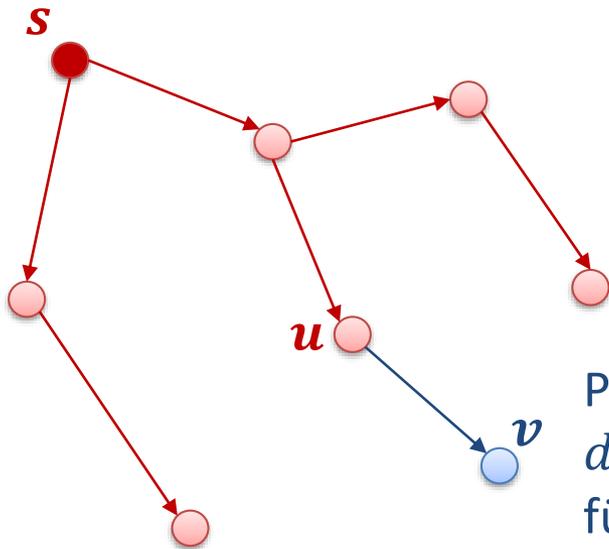


# Dijkstras Algorithmus: Ein Schritt

**Gegeben:**  $T$  ist Teilbaum eines “Shortest Path Tree” von  $s$  in  $G$  ist.

**Lemma:** Für die Kante  $(u, v)$  mit  $u \in T$  und  $v \notin T$ , welche  $d_G(s, u) + w(u, v)$  minimiert, gilt:

$$d_G(s, v) = d_G(s, u) + w(u, v)$$



Paar  $(u, v)$  minimiert  
 $d_G(s, u) + w(u, v)$   
für  $u \in T, v \notin T$

## Invariante:

Algorithmus hat zu jeder Zeit einen bei  $s$  gewurzelten Teilbaum eines “Shortest Path Tree”  $T$ .

- Am Anfang ist  $T = (\{s\}, \emptyset)$
- Für jeden Knoten  $v \notin T$  berechnet man zu jedem Zeitpunkt

$$\delta(s, v) := \min_{u \in T \cap N_{\text{in}}(v)} d_G(s, u) + w(u, v)$$

– sowie den Eingangsnachbar  $u =: \alpha(v)$ , welcher den Ausdruck minimiert...

- Invariante  $\Rightarrow \delta(s, v) \geq d_G(s, v)$
- Lemma auf letzter Folie:

**Für das minimale  $\delta(s, v)$  gilt:  $\delta(s, v) = d_G(s, v)$**

## Initialisierung $T = (\emptyset, \emptyset)$

- $\delta(s, s) = 0$ , sowie  $\delta(s, v) = \infty$  für alle  $v \in V \setminus S$
- $\alpha(v) = \text{NULL}$  für alle  $v \in V \setminus S$  (braucht's nicht für  $s$ )

## Iterationsschritt

- Wähle Knoten  $v$  mit kleinstem

$$\delta(s, v) := \min_{u \in T \cap N_{\text{in}}(v)} d_G(s, u) + w(u, v)$$

- Gehe durch die ausgehenden Nachbarn  $x \in V \setminus T$  und setze

$$\delta(s, x) := \min\{\delta(s, x), \delta(s, v) + w(v, x)\}$$

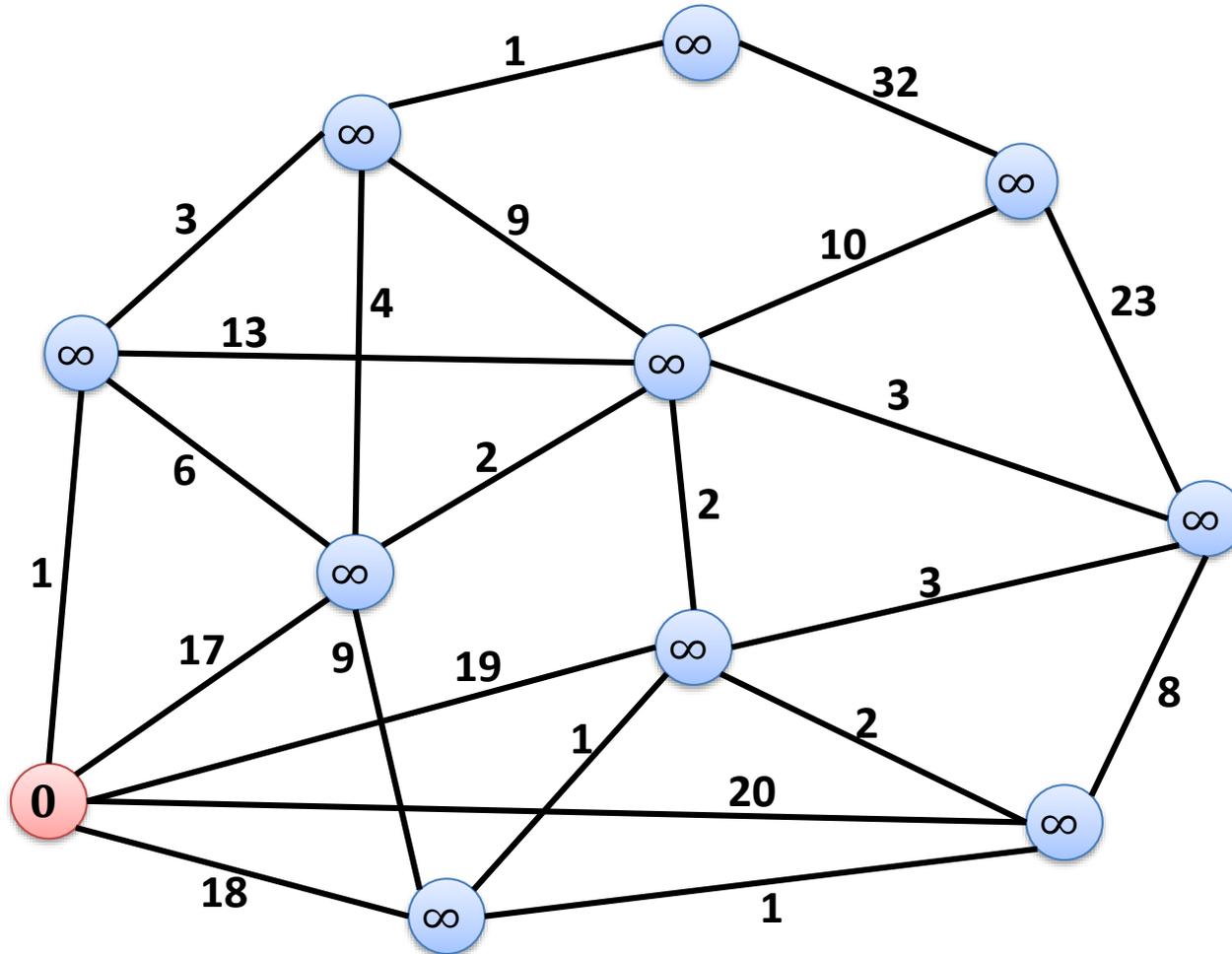
– Falls nötig, setze auch  $\alpha(x) = v$

- **Füge Kante  $(\alpha(v), v)$  zum Baum  $T$  hinzu.**

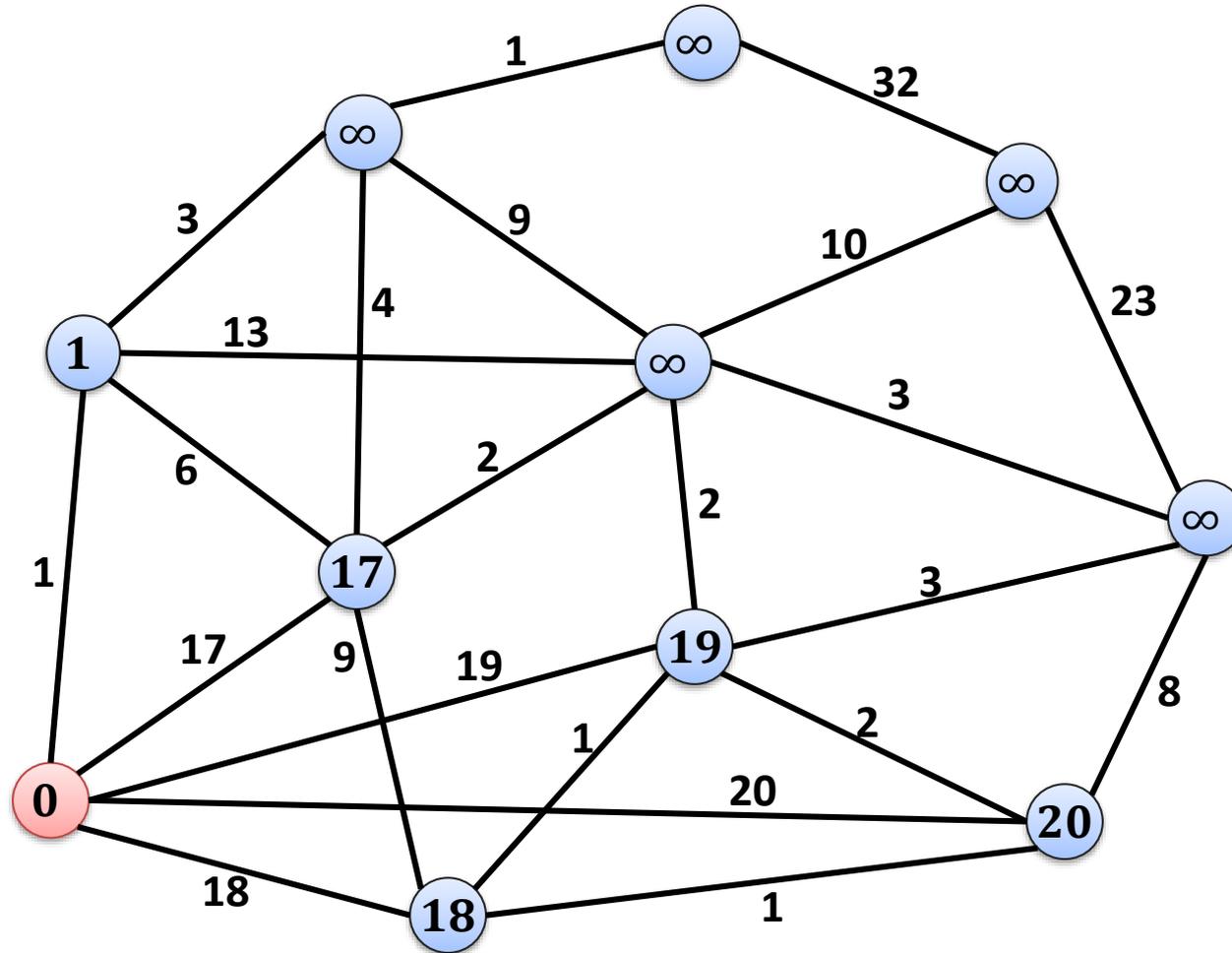
# Dijkstras Algorithmus

---

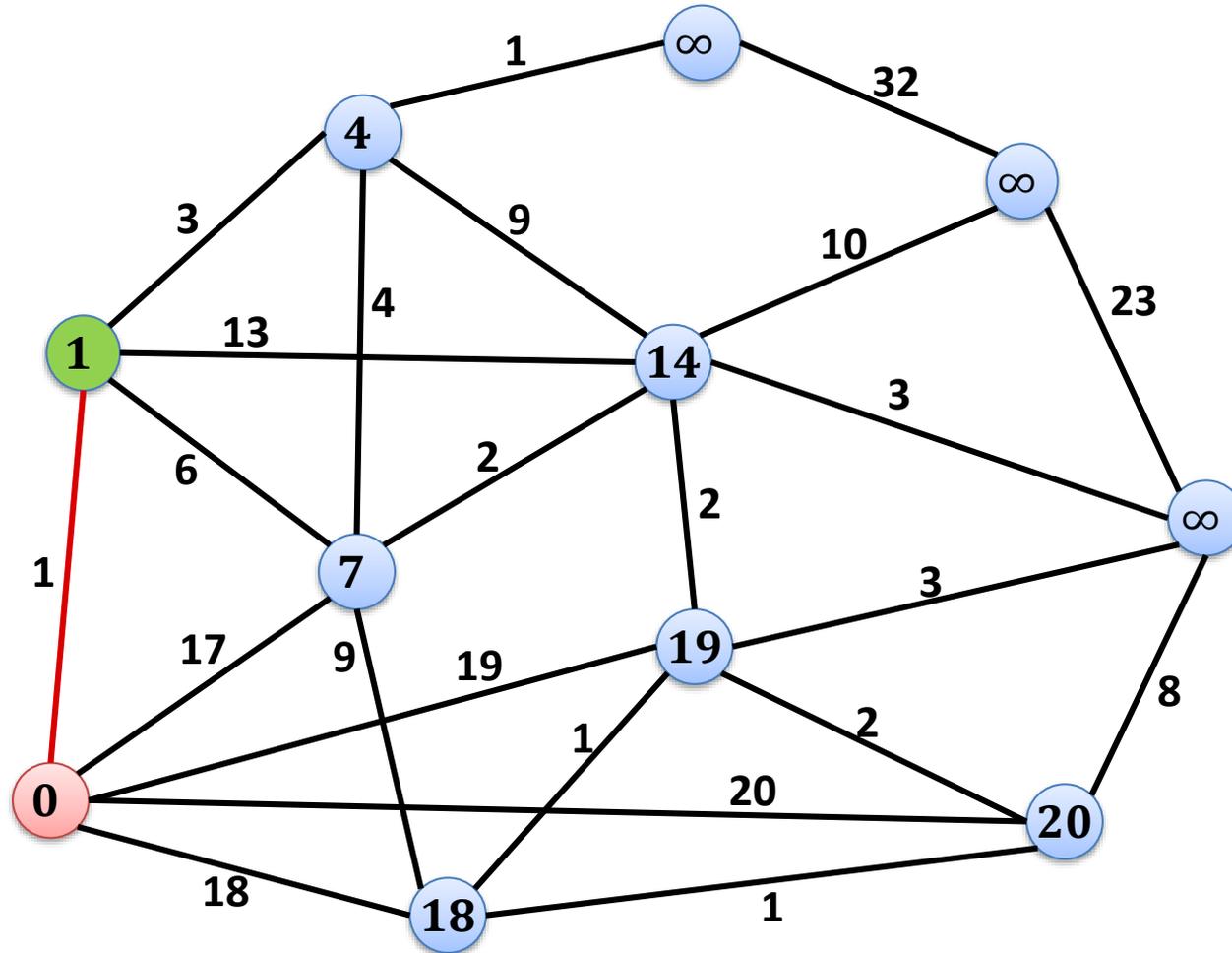
# Dijkstras Algorithmus: Beispiel



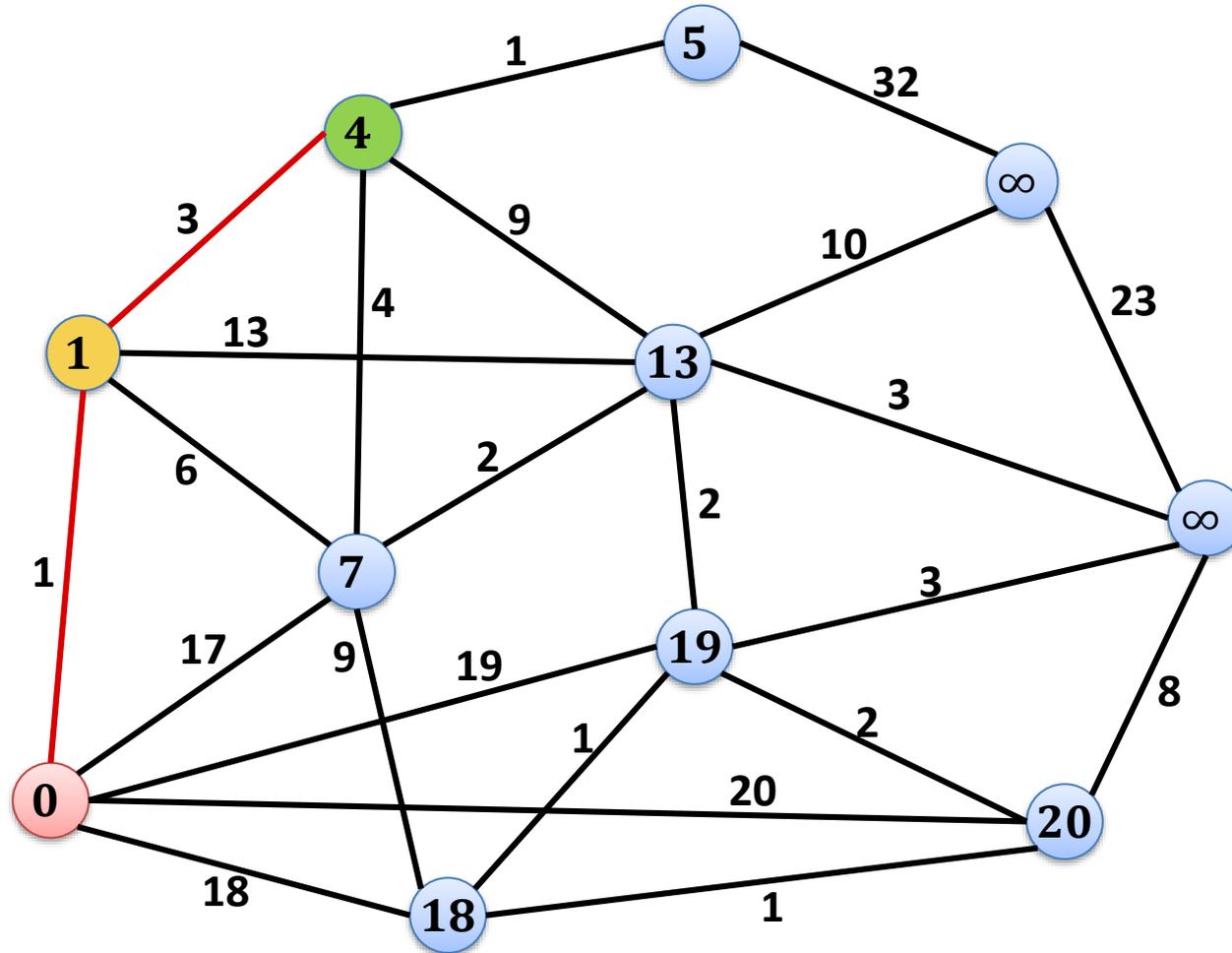
# Dijkstras Algorithmus: Beispiel



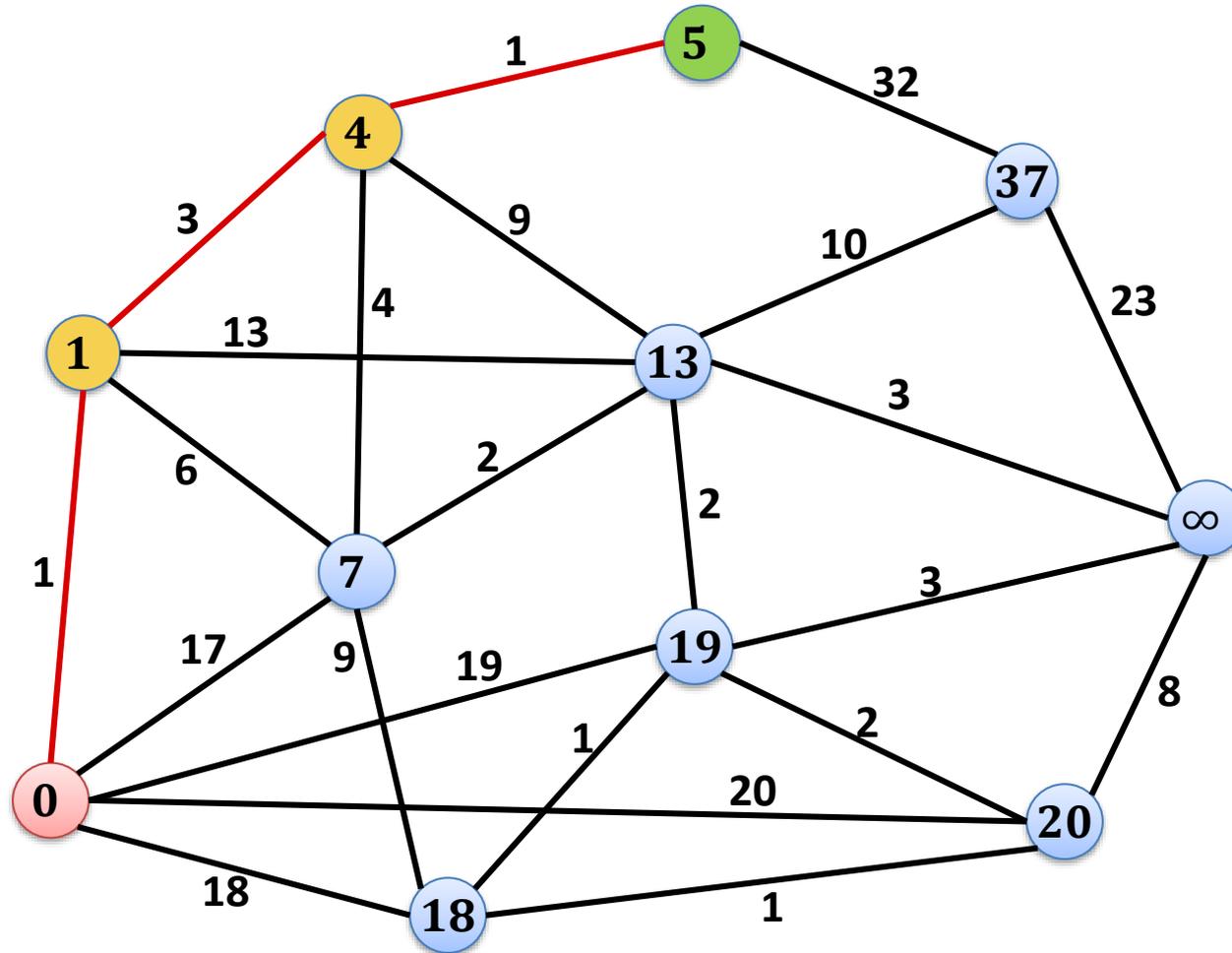
# Dijkstras Algorithmus: Beispiel



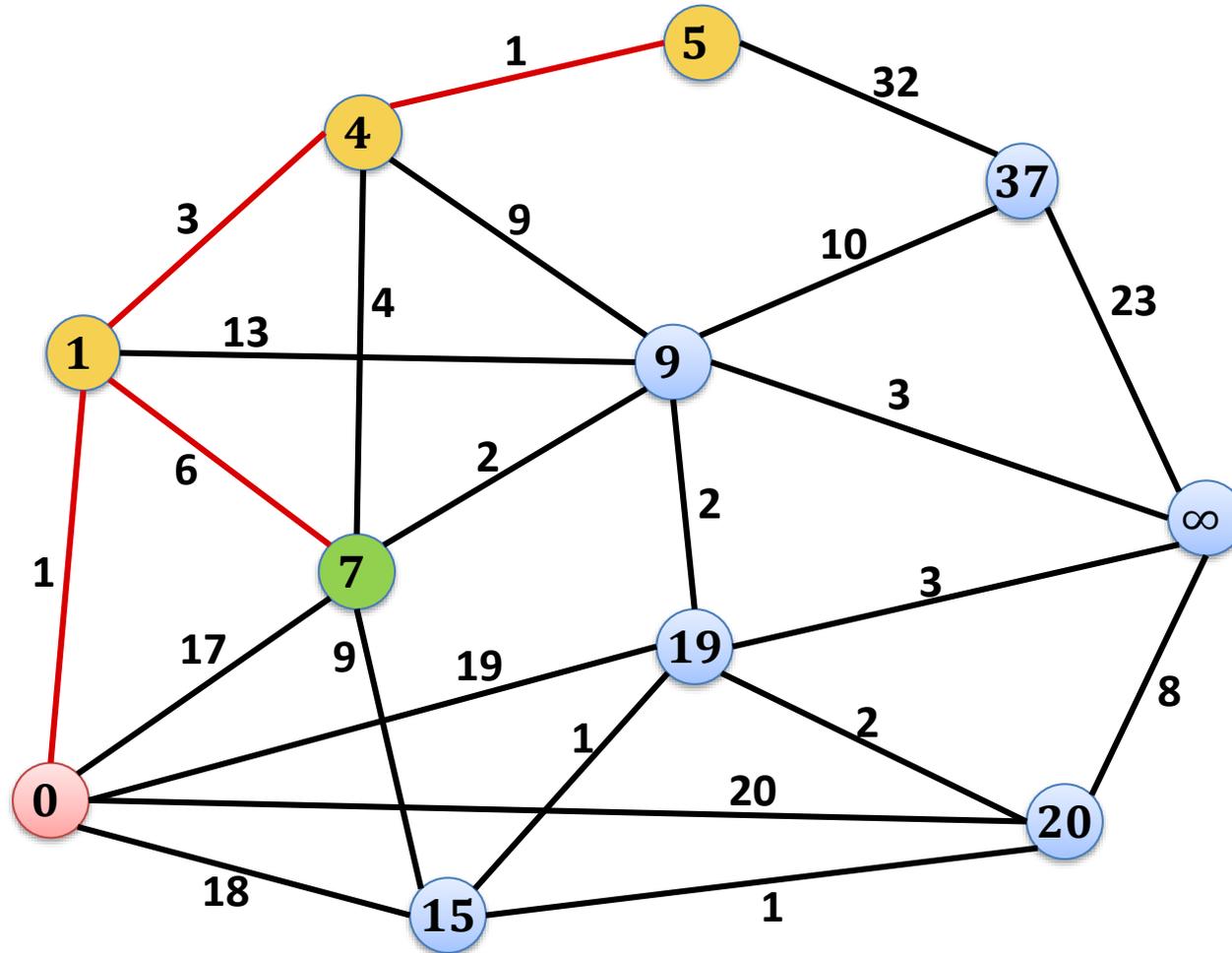
# Dijkstras Algorithmus: Beispiel



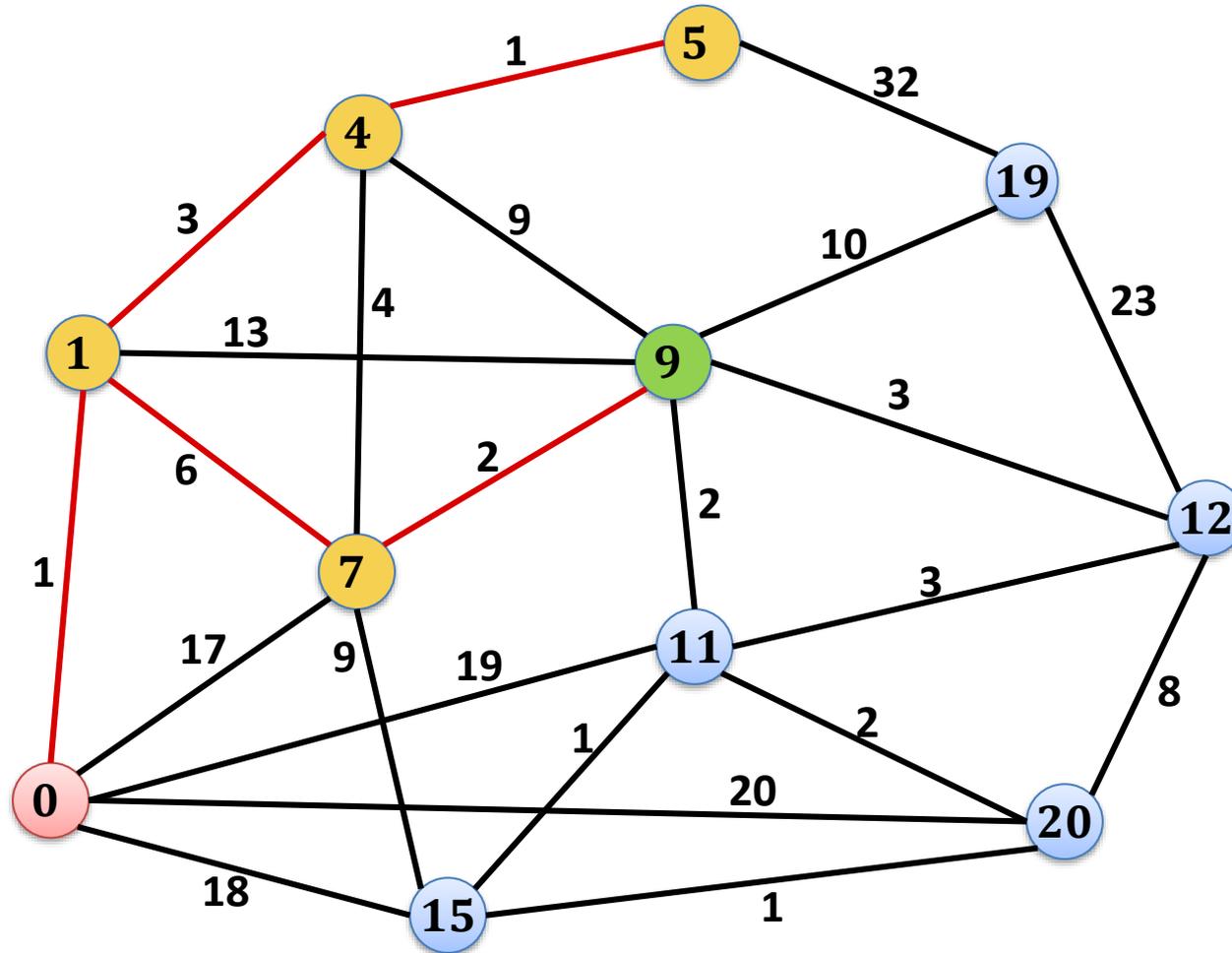
# Dijkstras Algorithmus: Beispiel



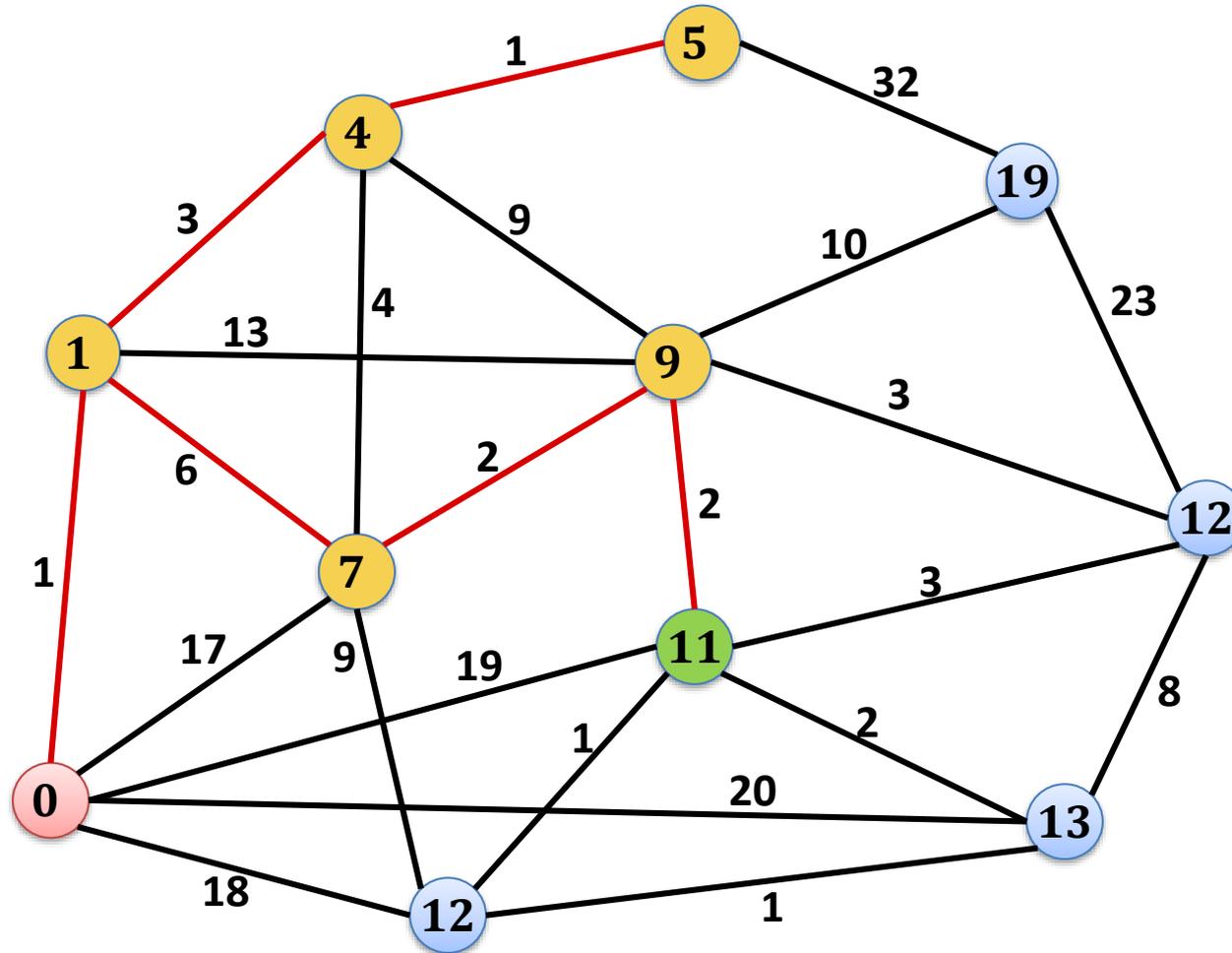
# Dijkstras Algorithmus: Beispiel



# Dijkstras Algorithmus: Beispiel



# Dijkstras Algorithmus: Beispiel



## Initialisierung $T = (\emptyset, \emptyset)$

- $\delta(s, s) = 0$ , sowie  $\delta(s, v) = \infty$  für alle  $v \in V \setminus S$
- $\alpha(v) = \text{NULL}$  für alle  $v \in V \setminus S$  (braucht's nicht für  $s$ )

## Iterationsschritt

- Wähle Knoten  $v$  mit kleinstem

$$\delta(s, v) := \min_{u \in T \cap N_{\text{in}}(v)} d_G(s, u) + w(u, v)$$

- Gehe durch die ausgehenden Nachbarn  $x \in V \setminus T$  und setze

$$\delta(s, x) := \min\{\delta(s, x), \delta(s, v) + w(v, x)\}$$

– Falls nötig, setze auch  $\alpha(x) = v$

- **Füge Kante  $(\alpha(v), v)$  zum Baum  $T$  hinzu.**

## Heap / Priority Queue:

- Verwaltet eine Menge von  $(key, value)$ -Paaren

### Operationen:

- *create* : erzeugt einen leeren Heap
- *H.insert(x, key)* : fügt Element  $x$  mit Schlüssel  $key$  ein
- *H.getMin()* : gibt Element mit kleinstem Schlüssel zurück
- *H.deleteMin()* : löscht Element mit kleinstem Schlüssel  
(gibt Element mit kleinstem Schlüssel zurück)
- *H.decreaseKey(x, newkey)* : Falls  $newkey$  kleiner als der aktuelle Schlüssel von  $x$  ist, wird der Schlüssel von  $x$  auf  $newkey$  gesetzt

# Erinnerung: Prim's MST Algorithmus

```
H = new priority queue;  $A = \emptyset$   
for all  $u \in V \setminus \{s\}$  do  
    H.insert( $u, \infty$ );  $\alpha(u) = \text{NULL}$   
H.insert( $s, 0$ )  
  
while H is not empty do  
     $u = H.\text{deleteMin}()$   
    for all unmarked neighbors  $v$  of  $u$  do  
        if  $w(\{u, v\}) < d(v)$  then  
            H.decreaseKey( $v, w(\{u, v\})$ )  
             $\alpha(v) = u$   
    if  $u \neq s$  then  $A = A \cup \{u, \alpha(u)\}$ 
```

# Dijkstras Algorithmus: Implementierung

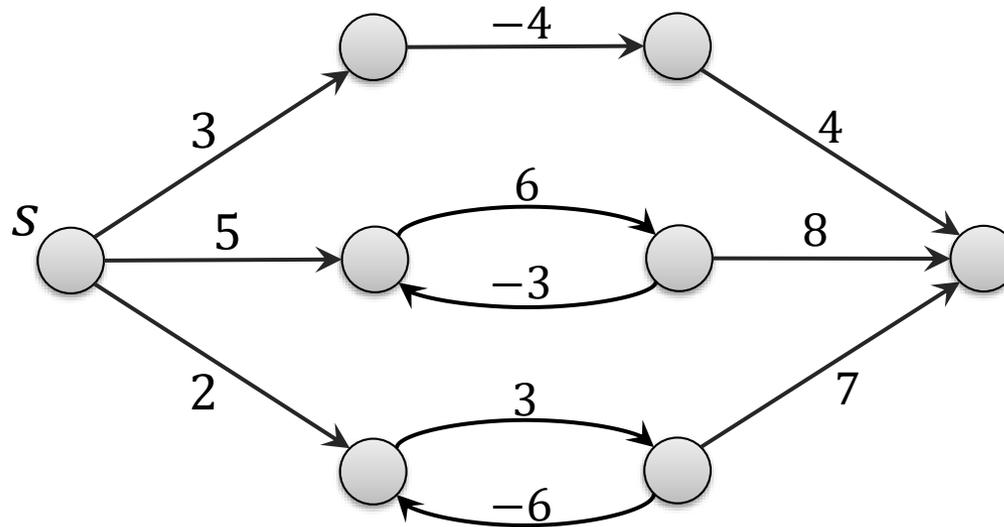
```
H = new priority queue;  $A = \emptyset$   
for all  $u \in V \setminus \{s\}$  do  
    H.insert( $u, \infty$ );  $\delta(s, u) = \infty$ ;  $\alpha(u) = \text{NULL}$   
H.insert( $s, 0$ )  
  
while H is not empty do  
     $u = H.\text{deleteMin}()$   
    for all neighbors  $v$  of  $u$  do  
        if  $\delta(s, u) + w(u, v) < \delta(s, v)$  then  
             $\delta(s, v) = \delta(s, u) + w(u, v)$   
            H.decreaseKey( $v, \delta(s, u)$ )  
             $\alpha(v) = u$   
  
    if  $u \neq s$  then  $A = A \cup \{u, \alpha(u)\}$ 
```

- Algorithmus-Implementierung ist fast identisch, wie diejenige von Prim's MST Algorithmus
- **Anzahl Heap-Operationen:**  
create: 1, insert:  $n$ , deleteMin:  $n$ , decreaseKey:  $\leq m$
- **Laufzeit mit binären Heaps:**  
 $O(m \log n)$
- **Laufzeit mit Fibonacci Heaps:**  
 $O(m + n \log n)$

# Negative Kantengewichte

- Kürzeste Pfade können auch in Graphen mit negativen Kantengewichten definiert werden

## Beispiel



# Negative Kantengewichte

**Satz:** In einem gerichteten, gewichteten Graphen  $G$  hat es genau dann einen kürzesten Pfad von  $u$  nach  $v$ , falls es keinen negativen Kreis gibt, welcher von  $u$  erreichbar ist und von welchem  $v$  erreichbar ist.

- gilt auch für ungerichtete Graphen, falls Kanten  $\{u, v\}$  als 2 gerichtete Kanten  $(u, v)$  und  $(v, u)$  betrachtet werden

**Lemma:** Falls  $v_0, v_1, \dots, v_k$  ein kürzester Pfad von  $v_0$  nach  $v_k$  ist, dann gilt für alle  $0 \leq i \leq j \leq k$ , dass der Teilpfad  $v_i, v_{i+1}, \dots, v_j$  ein kürzester Pfad von  $v_i$  nach  $v_j$  ist.

- gilt auch bei negativen Kantengewichten...

# Zur Erinnerung: Dijkstras Algorithmus

- Algorithmus kennt zu jedem Zeitpunkt einen Teilbaum  $T'$  eines Shortest Path Trees
- Man merkt sich zudem für jeden Knoten, wie er über  $\leq 1$  Kante von  $T'$  am besten erreicht werden kann und nimmt in jedem Schritt den besten solchen Knoten zu  $T'$  hinzu.

## Initialisierung $T = (\emptyset, \emptyset)$

- $\delta(s, s) = 0$ , sowie  $\delta(s, v) = \infty$  für alle  $v \in V \setminus S$
- $\alpha(v) = \text{NULL}$  für alle  $v \in V \setminus S$  (braucht's nicht für  $s$ )

## Iterationsschritt

- Wähle Knoten  $v$  mit kleinstem

$$\delta(s, v) := \min_{u \in T \cap N_{\text{in}}(v)} d_G(s, u) + w(u, v)$$

- Gehe durch die Ausgangsnachbarn  $x \in V \setminus T$  und setze

$$\delta(s, x) := \min\{\delta(s, x), \delta(s, v) + w(v, x)\}$$

– Falls nötig, setze auch  $\alpha(x) = v$

- **Füge Kante  $(\alpha(v), v)$  zum Baum  $T$  hinzu.**

Funktioniert Dijkstras Algorithmus auch bei negativen Kantengewichten?

- Antwort: nein

# Bellman-Ford Algorithmus

- Zur Vereinfachung, berechnen wir nur die Distanzen  $d_G(s, v)$

## Annahme:

- Für alle Knoten  $v$ : Algorithmus hat Wert  $\delta(s, v) \geq d_G(s, v)$
- Initialisierung:  $\delta(s, s) = 0$ ,  $\delta(s, v) = \infty$  für  $v \neq s$

## Beobachtung:

- Falls  $(u, v) \in E$ , so dass  $\delta(s, u) + w(u, v) < \delta(s, v)$ , dann können wir  $\delta(s, v)$  verkleinern, da  $d_G(s, v) \leq \delta(s, u) + w(u, v)$ .

- Betrachte alle Kanten  $(u, v)$  und versuche  $\delta(s, v)$  zu verbessern
  - solange, bis alle Distanzen korrekt sind ( $\forall v \in V: \delta(s, v) = d_G(s, v)$ )

**repeat**

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

**if**  $\delta(s, u) + w(u, v) < \delta(s, v)$  **then**

$\delta(s, v) := \delta(s, u) + w(u, v)$

**until**  $\forall v \in V: \delta(s, v) = d_G(s, v)$

- Wieviele Wiederholungen sind nötig?

- Betrachte alle Kanten  $(u, v)$  und versuche  $\delta(s, v)$  zu verbessern
  - solange, bis alle Distanzen korrekt sind ( $\forall v \in V: \delta(s, v) = d_G(s, v)$ )

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

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

**if**  $\delta(s, u) + w(u, v) < \delta(s, v)$  **then**

$\delta(s, v) := \delta(s, u) + w(u, v)$

Nach  $i$  Wiederholungen ist  $\delta(s, v) \leq d_G^{(i)}(s, v)$ , wobei  $d_G^{(i)}(s, v)$  die Länge des kürzesten Pfades aus höchstens  $i$  Kanten bezeichnet.