Algorithmen und Datenstrukturen

Vorlesung 10

Graphenalgorithmen III:
Kürzeste Wege

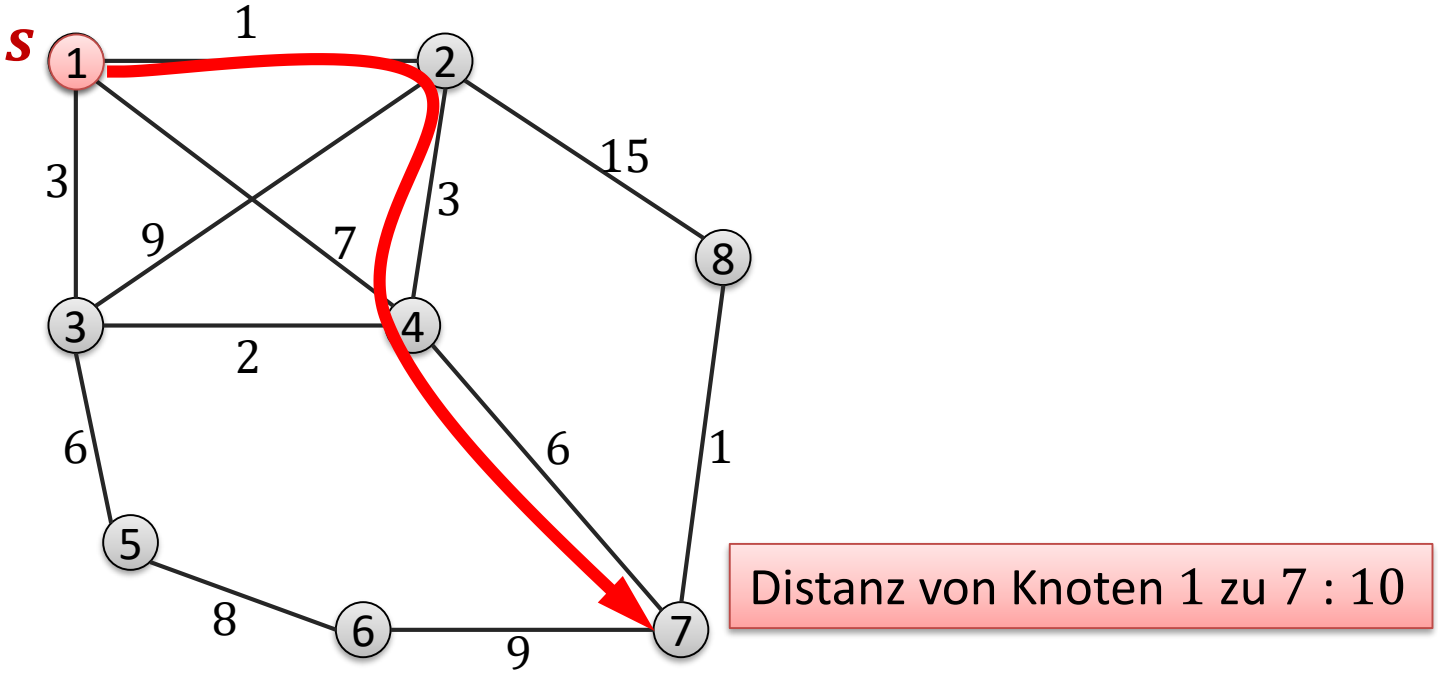
UNI FREIBURG

Fabian Kuhn
Algorithmen und Komplexität

Kürzeste Wege

Single Sourse Shortest Paths Problem

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

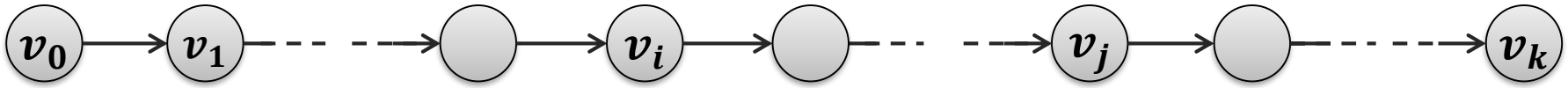


Distanz von Knoten 1 zu 7 : 10

Optimalität von Teilpfaden

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.

Kürzester Pfad von v_0 nach v_k:

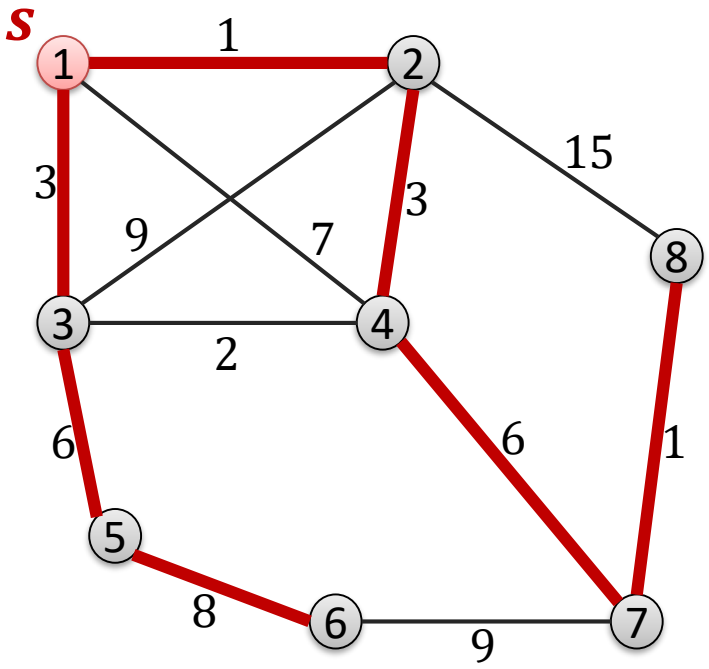


- Teilpfad von v_i nach v_j ist auch kürzester Pfad.
  - Sonst könnte man den Teilpfad von v_i nach v_j durch den kürzesten Pfad von v_i nach v_j ersetzen.
  - Falls dadurch Knoten mehrfach besucht werden, kann man einen Zyklus heraussschneiden und erhält einen noch kürzeren Pfad.

- Lemma gilt auch bei negativen Kantengewichten,
  - solange es im Graph keine negativen Zyklen hat

Shortest-Path Tree

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



Dijkstras Algorithmus: Idee

UNI FREIBURG

- Algorithmus von Edsger W. Dijkstra (1959 publiziert)

Idee:

- Wir starten bei s und bauen schrittweise den Spannbaum auf

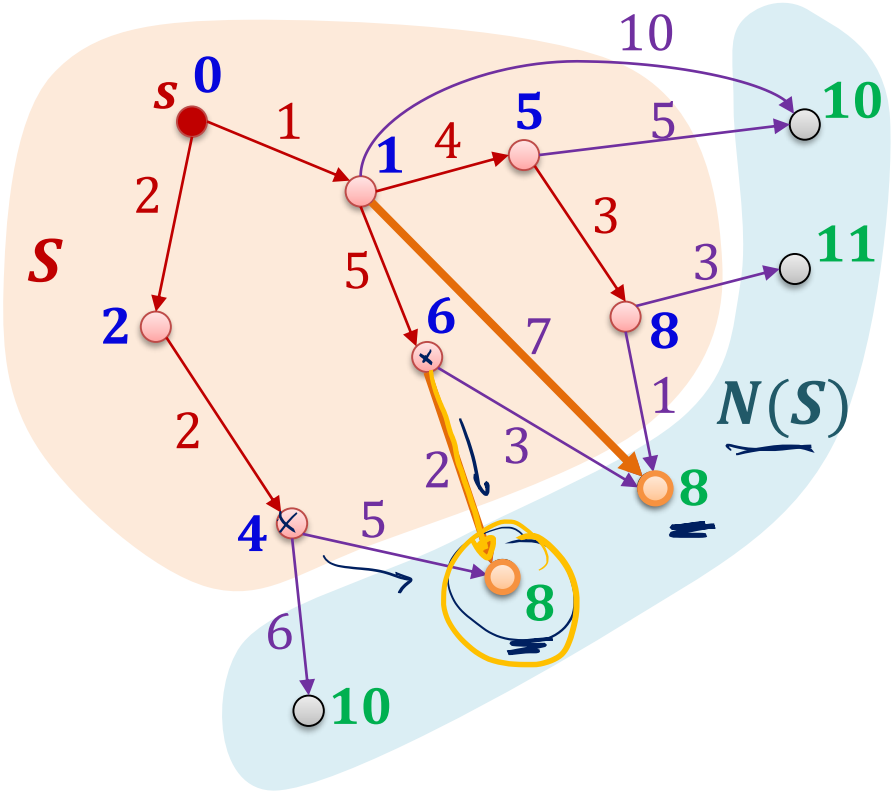
> **Invariante:**
> Algorithmus hat zu jeder Zeit einen bei s gewurzelten Teilbaum eines "Shortest Path Tree".

- 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 von 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. (Knoten von T : S)

Wie können wir T um einen Knoten erweitern?



S : Knoten in Baum T

$N(S)$: Knoten, die direkt zum Baum hinzugefügt werden können.

Um $v \in N(S)$ hinzuzufügen, muss gelten, dass

$$d_G(s,v) = \min_{u \in S}\{d_G(s,u) + w(u,v)\}$$

Wir werden sehen, dass das für $v \in N(S)$ mit minimaler Distanz $d_G(s,v)$ von s gilt.

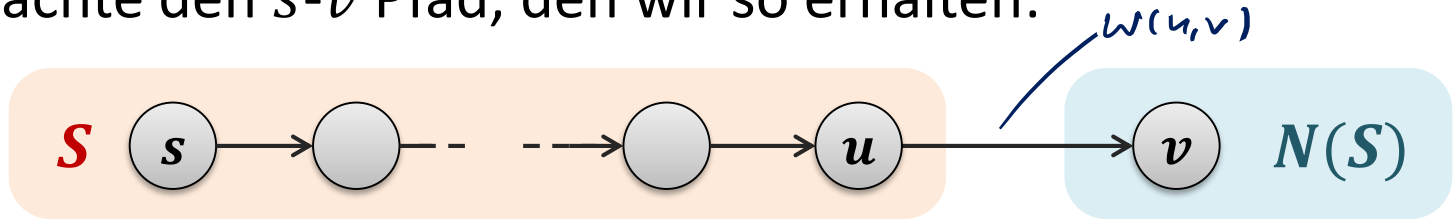
Dijkstras Algorithmus: Ein Schritt

Gegeben: T ist Teilbaum eines "Shortest Path Tree" von s in G ist.

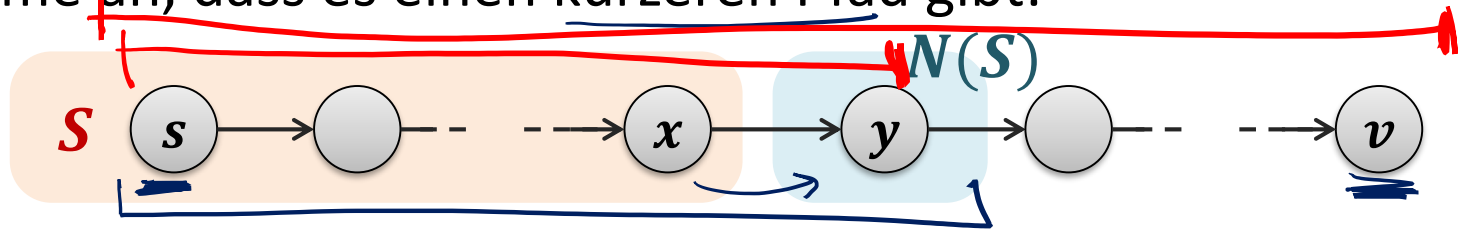
Lemma: Für einen Knoten $v \in N(S)$ und eine Kante (u, v) mit $u \in S$, so dass $d_G(s,u) + w(u,v)$ minimiert wird, gilt:

$$\boldsymbol{d_G(s,v) = d_G(s,u) + w(u,v)}$$

Betrachte den s-v Pfad, den wir so erhalten:



Nehme an, dass es einen kürzeren Pfad gibt:



– Weil es keine negative Kantengewichte gibt, gilt damit

$$d_G(s,x) + w(x,y) \leq d_G(s,v) < d_G(s,u) + w(u,v)$$

Dijkstras Algorithmus

Invariante:
Algorithmus hat zu jeder Zeit einen bei s gewurzelten Teilbaum eines "Shortest Path Tree" $T = (S, A)$.

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

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

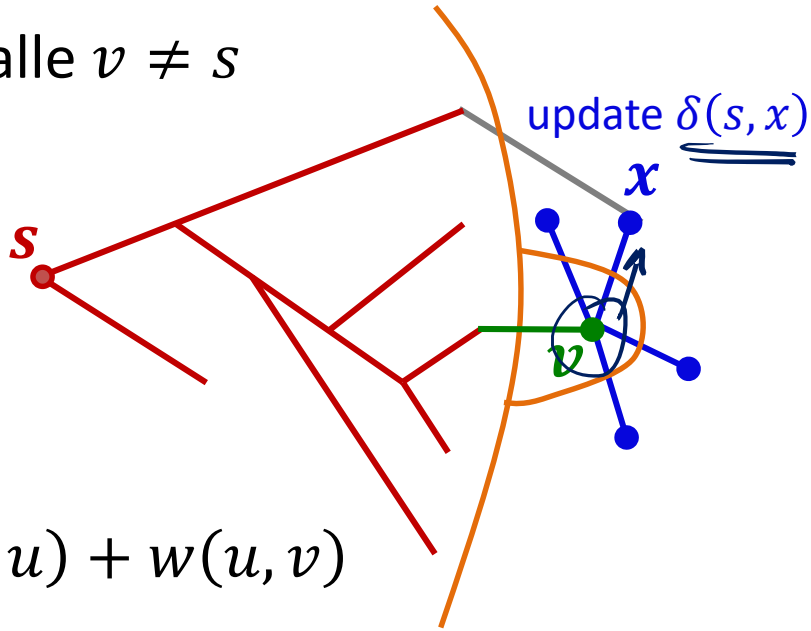
  - sowie den Eingangsnachbar $u =: \alpha(v)$, welcher den Ausdruck minimiert...
- $\delta(s,v)$ entspricht immer einem s-v Pfad $\Longrightarrow \delta(s,v) \geq d_G(s,v)$
- Lemma auf letzter Folie:

  Für das minmale $\delta(s,v)$ gilt: $\boldsymbol{\delta(s,v) = d_G(s,v)}$

Dijkstras Algorithmus

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

- $\delta(s,s) = 0$, sowie $\delta(s,v) = \infty$ für alle $v \neq s$
- $\alpha(v) = \text{NULL}$ für alle $v \in V$



Iterationsschritt

- Wähle Knoten v mit kleinstem

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

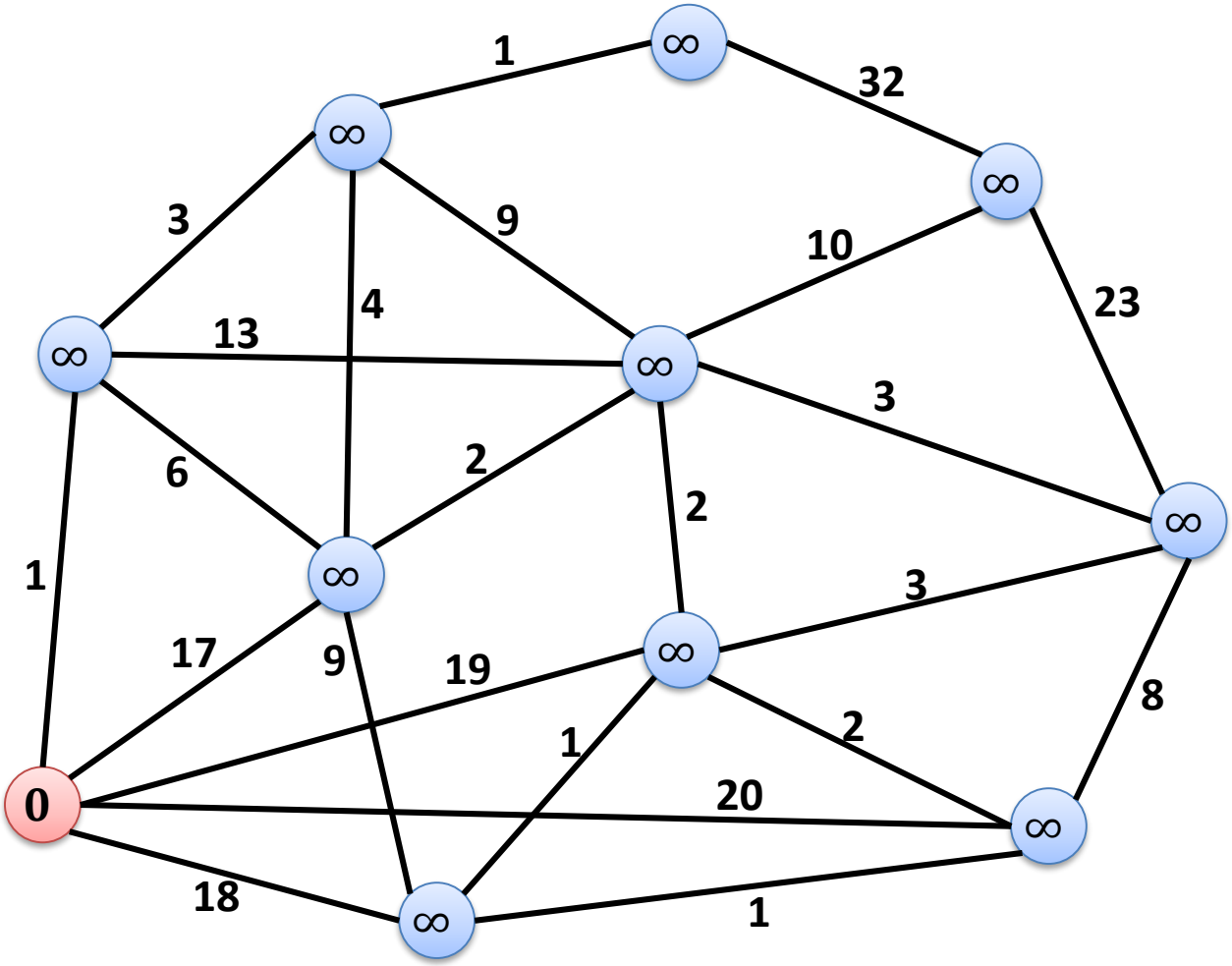
- Gehe durch die Ausgangsnachbarn $x \in V \setminus S$ und setze

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

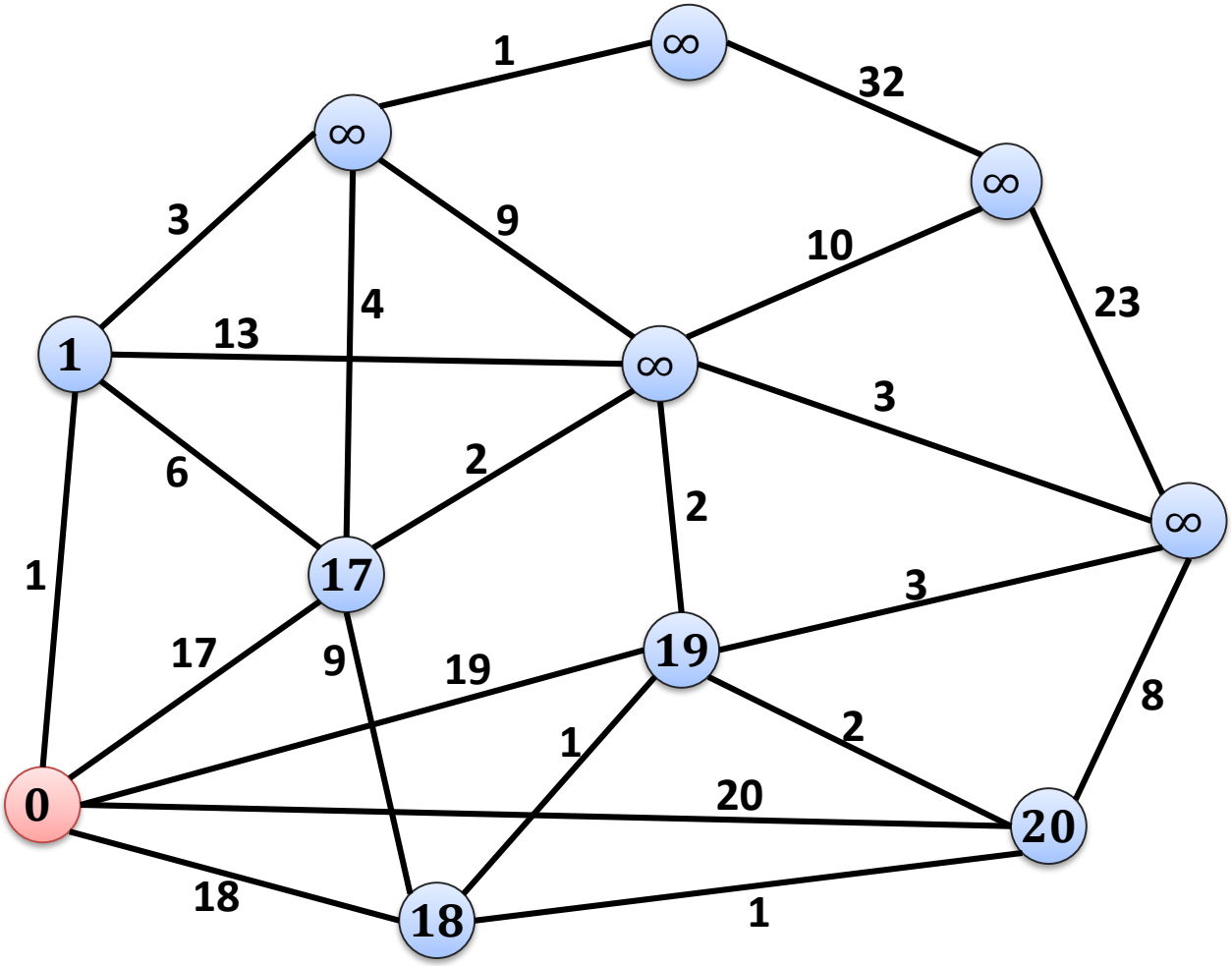
  - Falls $\delta(s,x)$ verkleinert wird, setze $\alpha(x) = v$

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

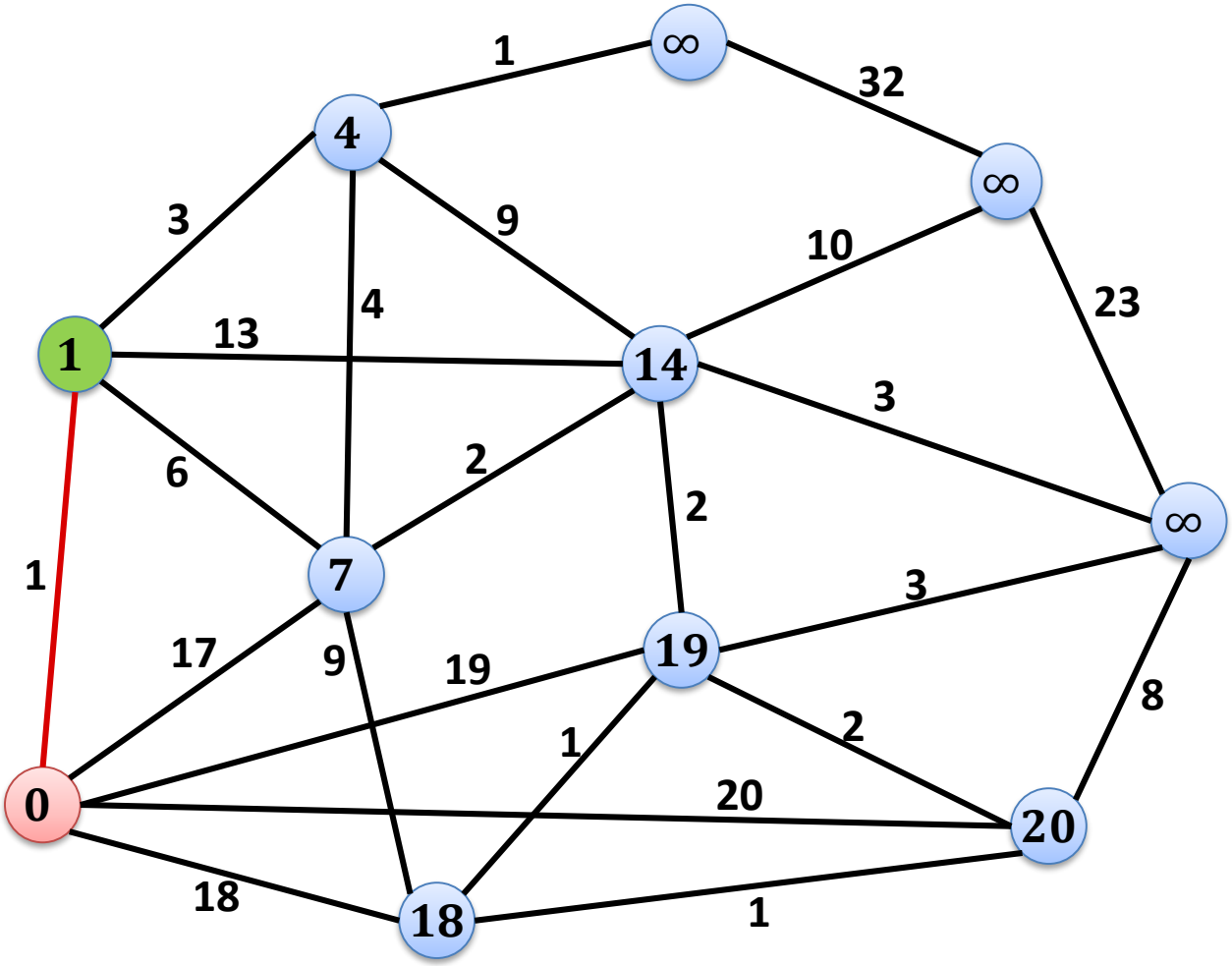
Dijkstras Algorithmus: Beispiel



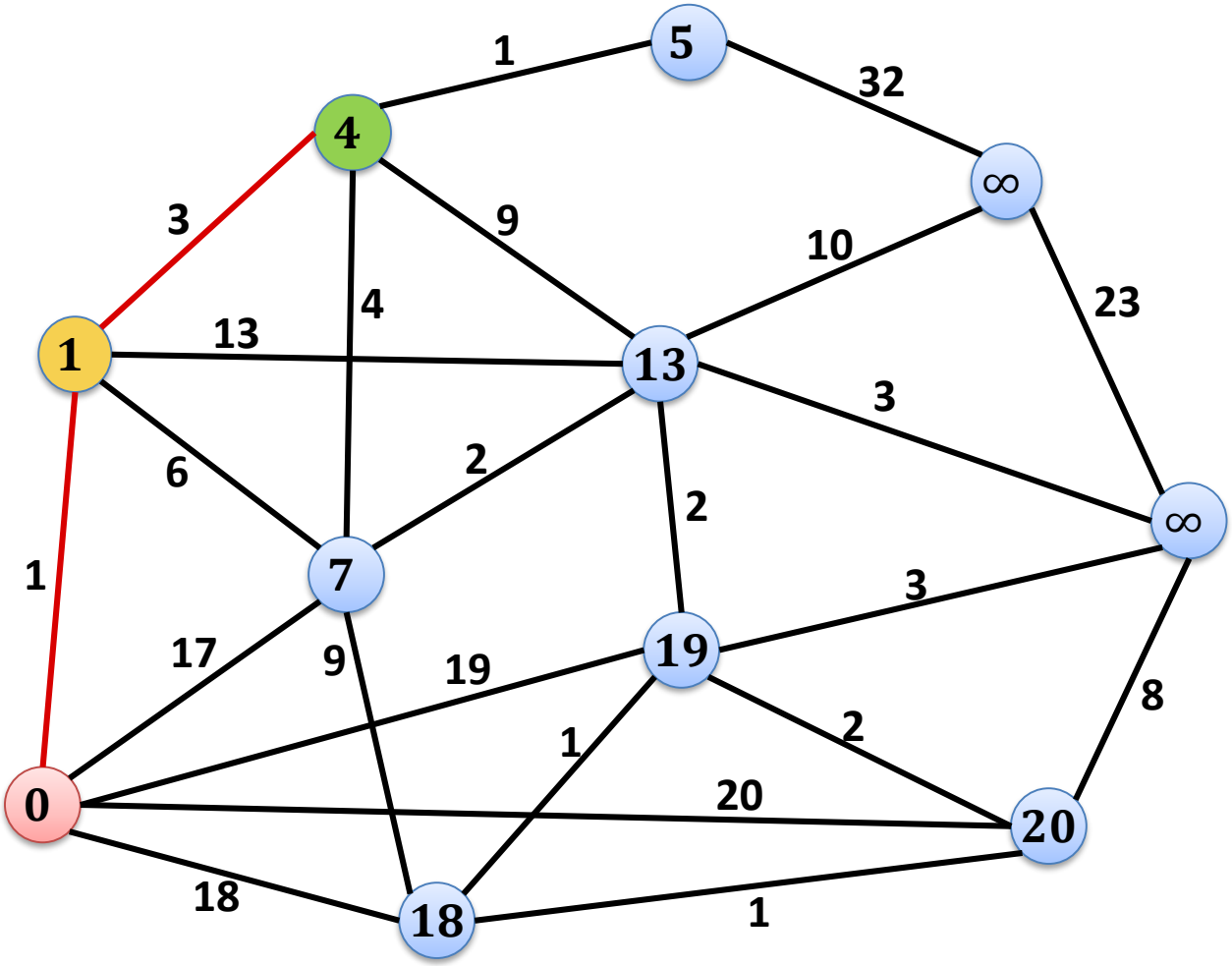
Dijkstras Algorithmus: Beispiel



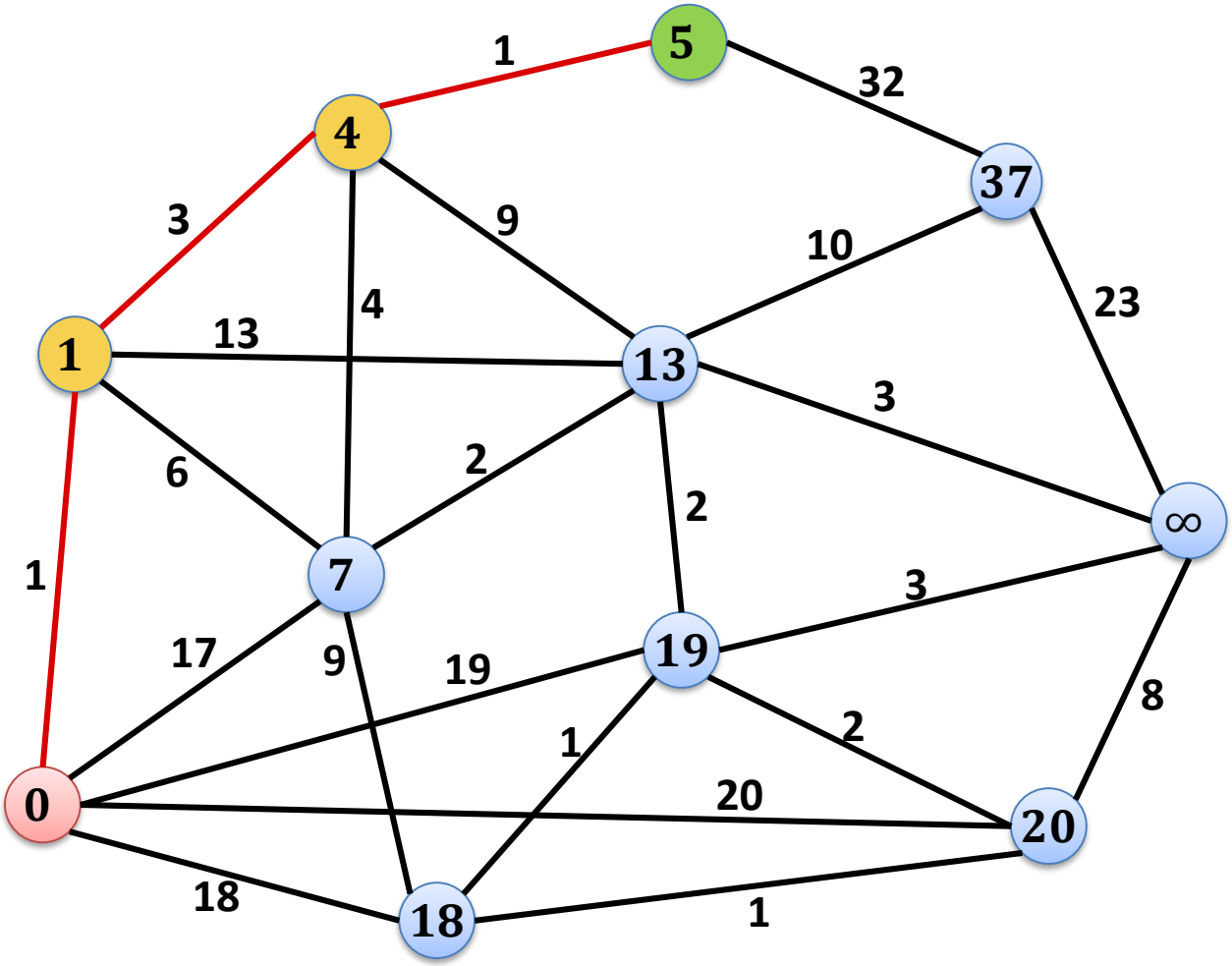
Dijkstras Algorithmus: Beispiel



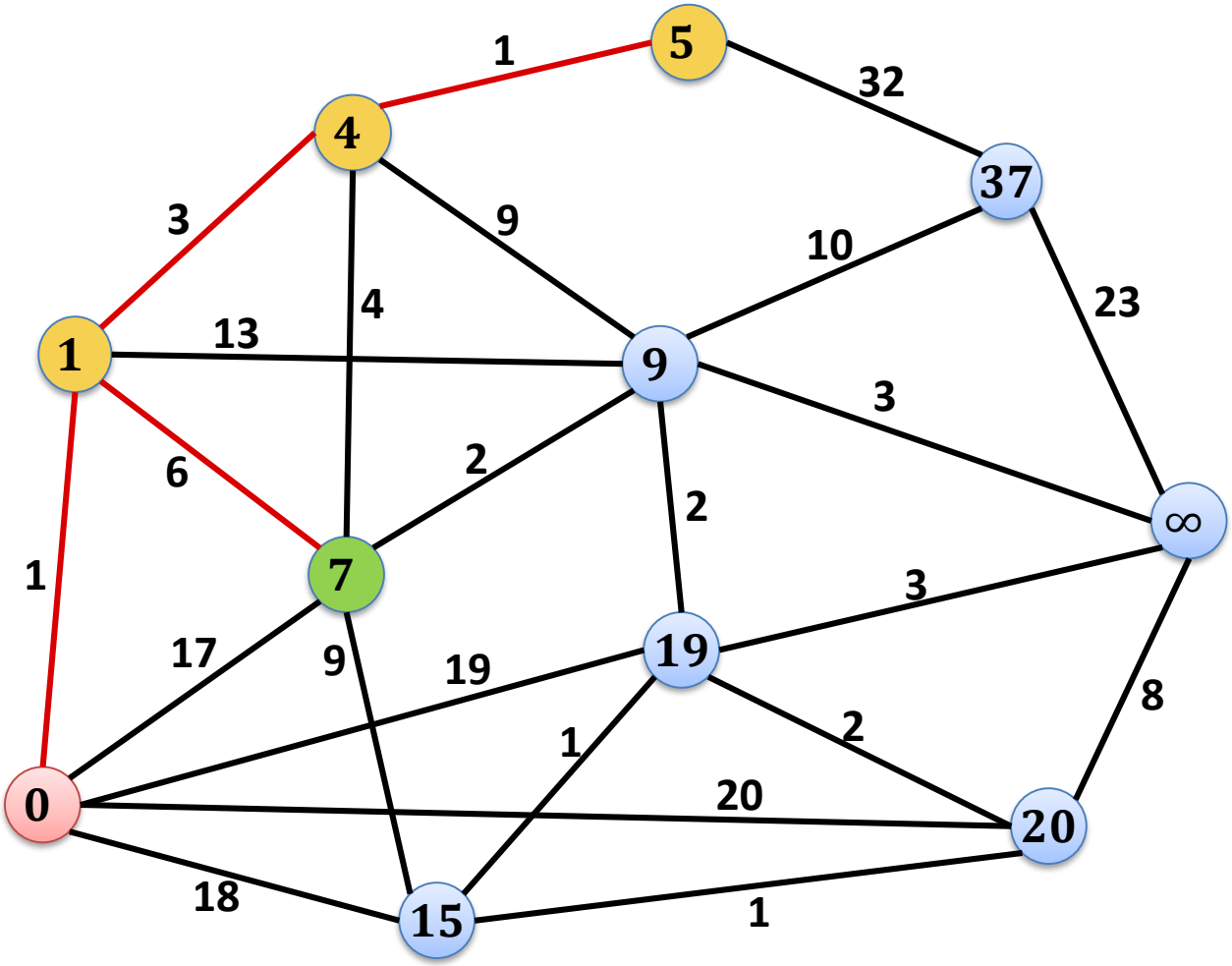
Dijkstras Algorithmus: Beispiel



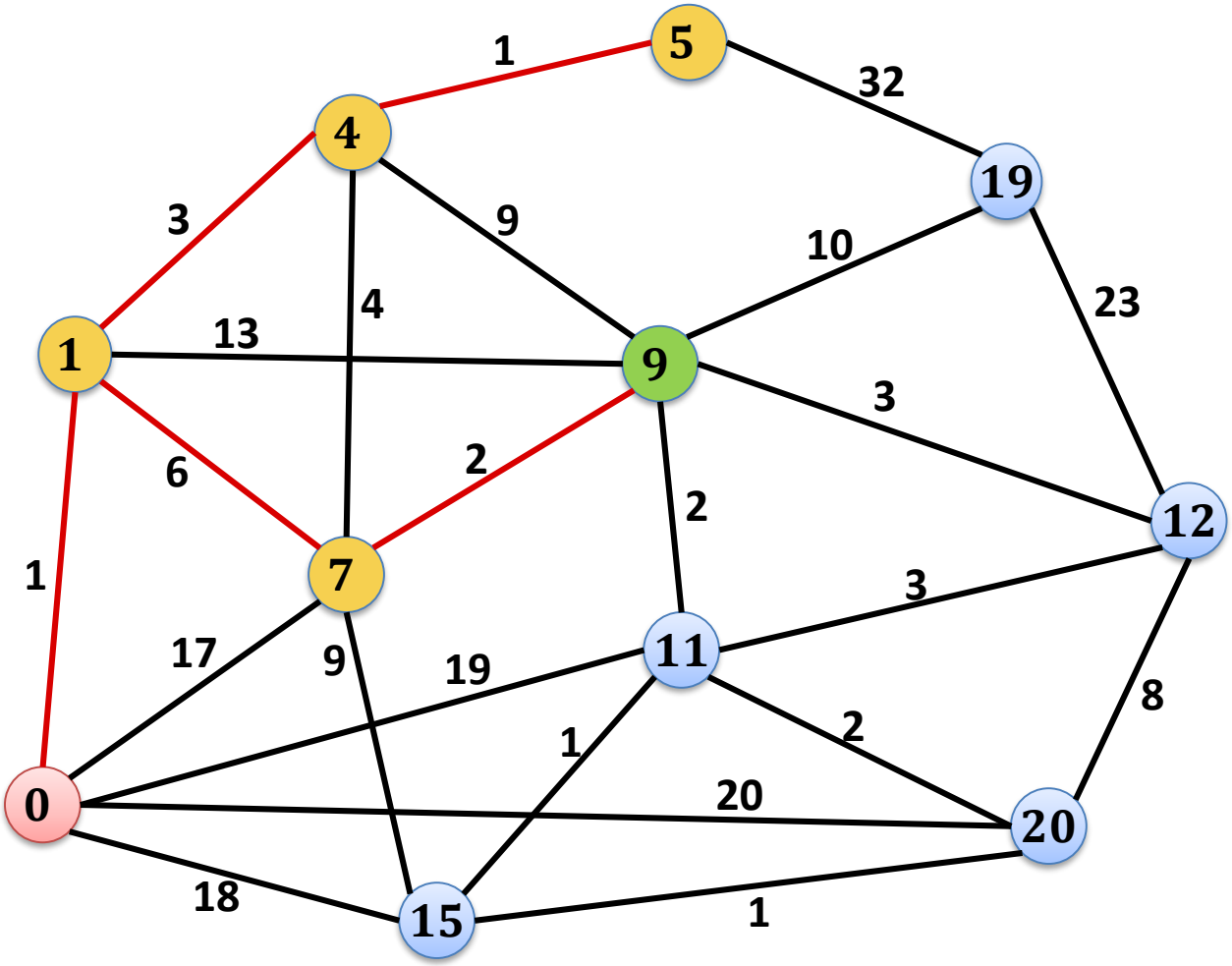
Dijkstras Algorithmus: Beispiel



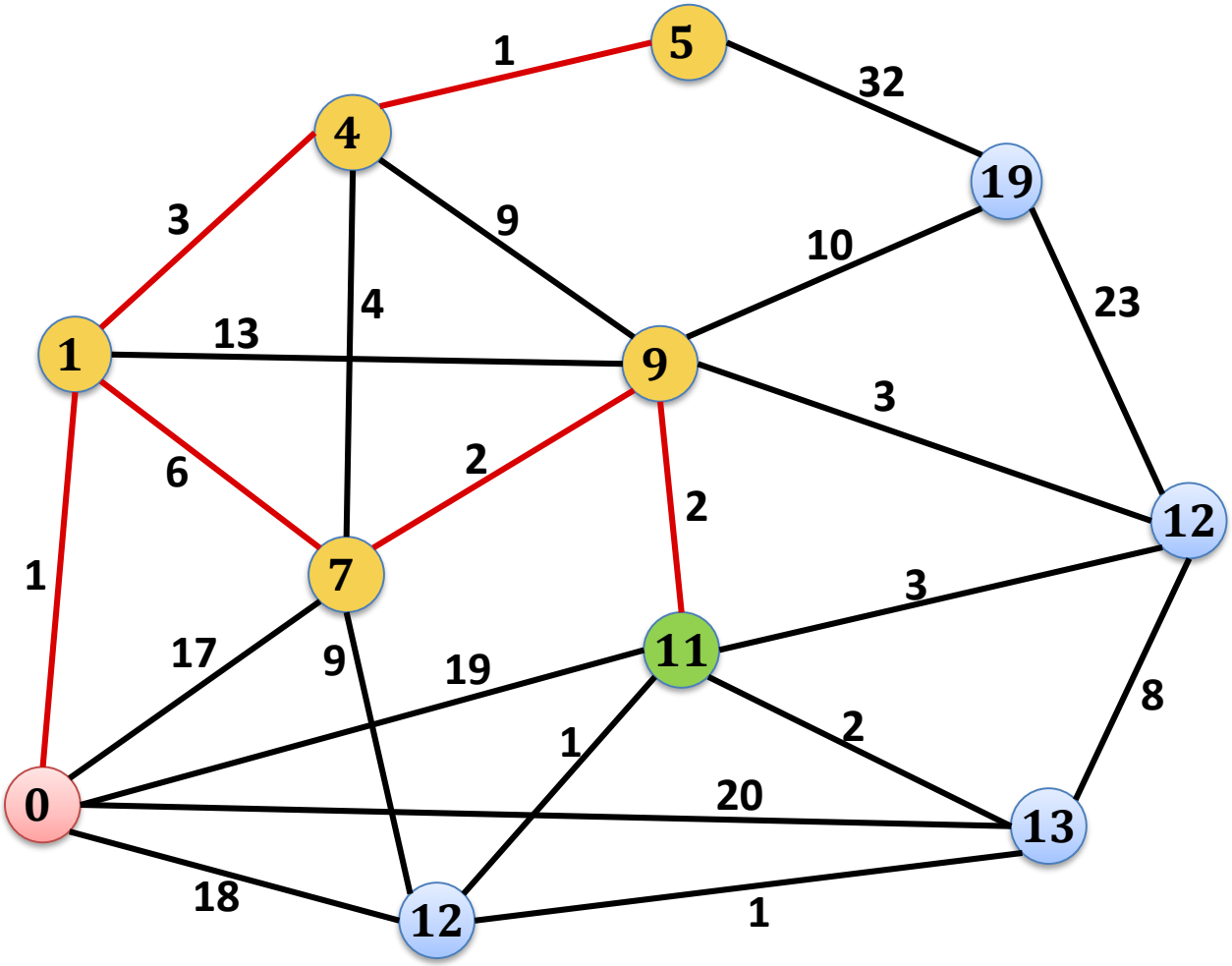
Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Beispiel



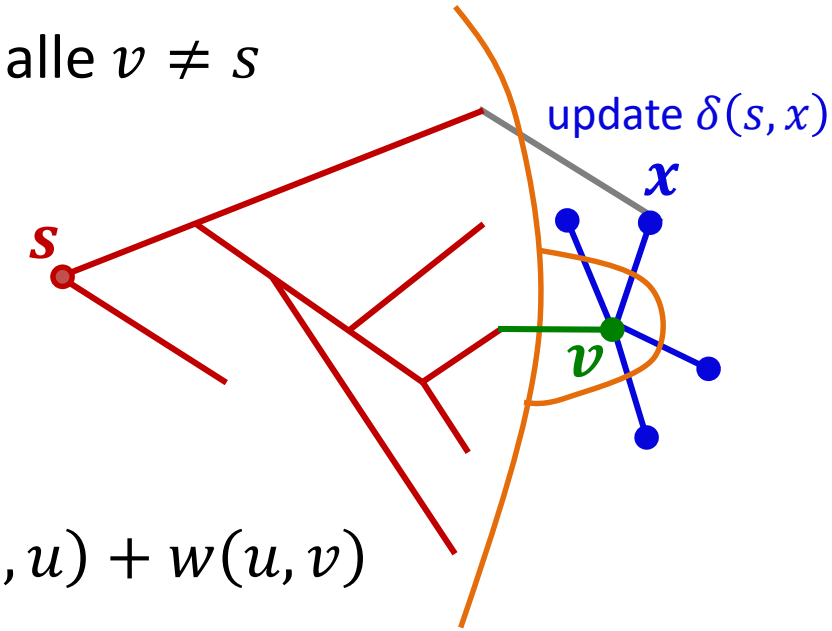
Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus

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

- $\delta(s,s) = 0$, sowie $\delta(s,v) = \infty$ für alle $v \neq s$
- $\alpha(v) = \text{NULL}$ für alle $v \in V$



Interationsschritt

- Wähle Knoten v mit kleinstem

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

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

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

  - Falls $\delta(s,x)$ verkleinert wird, setze $\alpha(x) = v$

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

Ähnlich wie der MST Algorithmus von Prim!

Erinnerung: Prims MST Algorithmus

H = new priority queue; $A = \emptyset$

for all $u \in V \setminus \{s\}$ **do**

    H.insert(u, ∞); $\alpha(u)$ = NULL

H.insert(s, 0)

while H is not empty **do**

    u = H.deleteMin()

    for all unmarked neighbors v of u **do**

        if $w(\{u,v\}) < d(v)$ **then**

            H.decreaseKey(v, $w(\{u,v\})$)

            $\alpha(v) = u$

    u.marked = true

    if $u \neq s$ **then** $A = A \cup \{\{u, \alpha(u)\}\}$

Dijkstras Algorithmus: Implementierung

UNI FREIBURG

```
H = new priority queue; A = ∅
for all u ∈ V \ {s} do
    H.insert(u, ∞); δ(s,u) = ∞; α(u) = NULL
H.insert(s, 0)

while H is not empty do
    u = H.deleteMin()
    for all unmarked out-neighbors v of u do
        if δ(s,u) + w(u,v) < δ(s,v) then
            δ(s,v) = δ(s,u) + w(u,v)
            H.decreaseKey(v, δ(s,v))
            α(v) = u
    u.marked = true
    if u ≠ s then A = A ∪ {(α(u),u)}
```

Dijkstras Algorithmus: Laufzeit

- Algorithmus-Implementierung ist fast identisch, wie diejenige von Prims MST Algorithmus
- **Anzahl Heap-Operationen:**

  create: 1, insert: n, deleteMin: n, decreaseKey: $\leq m$

  – Oder alternativ ohne decrease-key: $O(m)$ insert und deleteMin Op.

- **Laufzeit mit binären Heaps:**

$$O(m \log n)$$

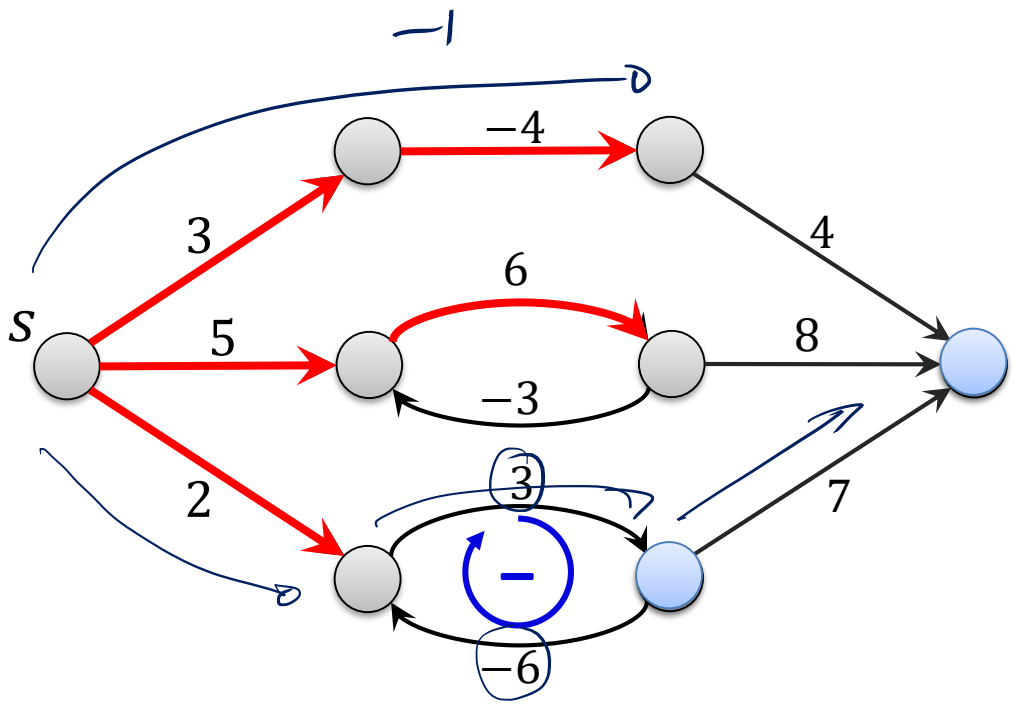
- **Laufzeit mit Fibonacci Heaps:**

$$O(m + n \log n)$$

Negative Kantengewichte

- Kürzeste Pfade können auch in Graphen mit negativen Kantengewichten definiert werden
  - Kürzester Pfad ist definiert, falls es auch keinen kürzeren Weg gibt, bei dem man Knoten mehrfach besuchen kann.

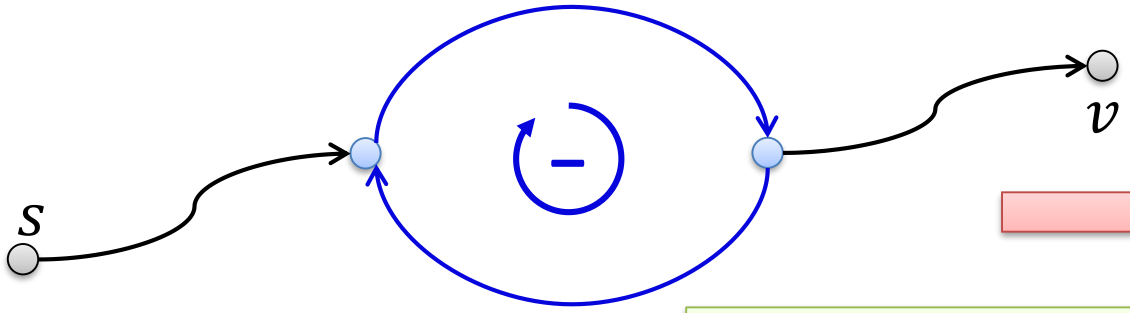
Beispiel



Negative Kantengewichte

Lemma: In einem gerichteten, gewichteten Graphen G hat es genau dann einen kürzesten Pfad von s nach v, falls es keinen negativen Kreis gibt, welcher von s 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



kein kürzester Weg von u nach v

Knoten werden nicht mehrfach besucht.

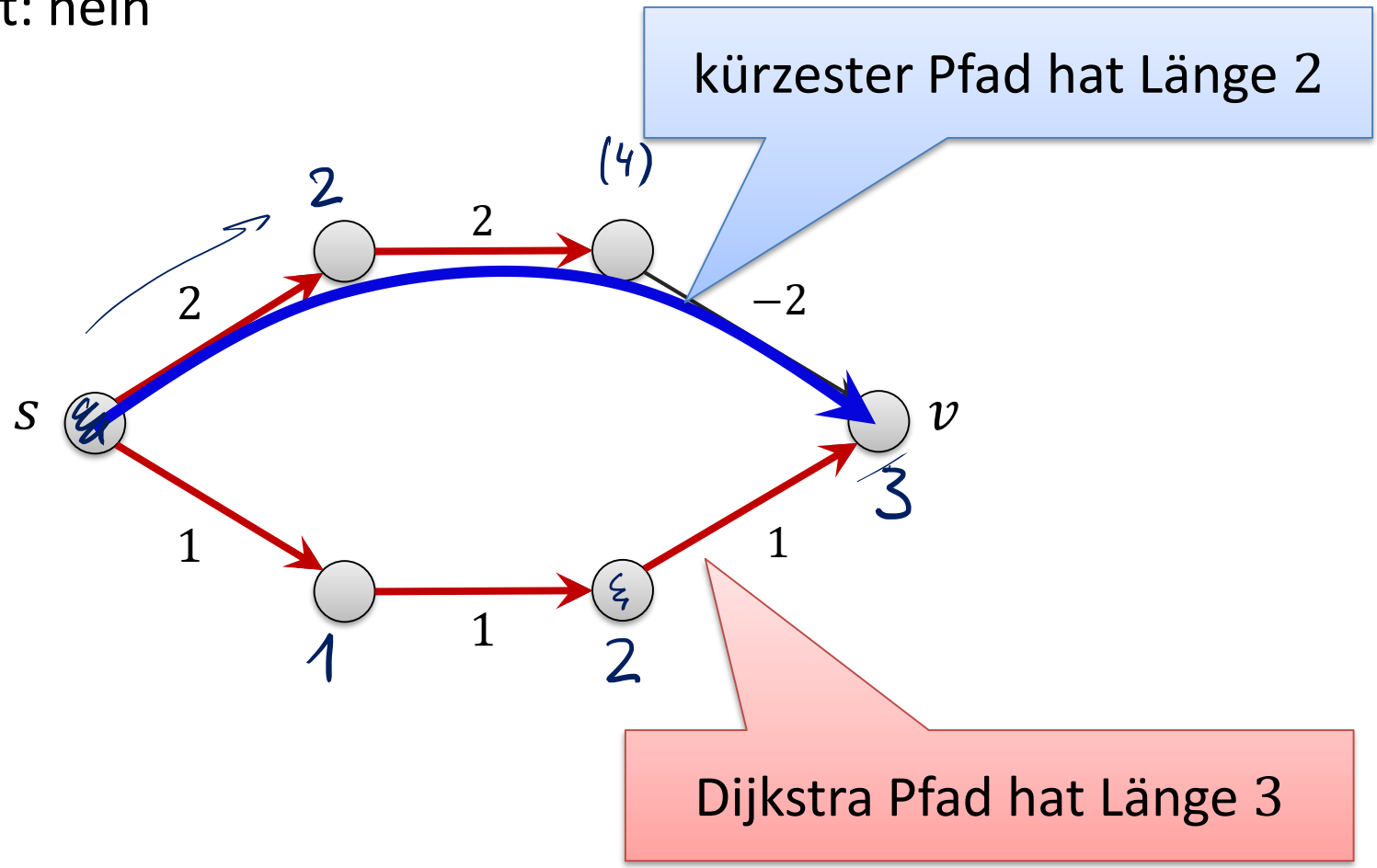
kein erreichbarer negativer Kreis

Wir können uns auf einfache Pfade beschränken. Davon gibt es nur endlich viele.

Dijkstras Algorithmus und negative Gewichte

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 $\boldsymbol{\delta(s,v) \geq d_G(s,v)}$
- 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

$$\boldsymbol{d_G(s,v)} \leq d_G(s,u) + w(u,v)$$
$$\leq \boldsymbol{\delta(s,u) + w(u,v)}$$

Bellman-Ford Algorithmus

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

$\delta(s,s) := 0$; $\forall v \in V \setminus \{s\} : \delta(s,v) := \infty$

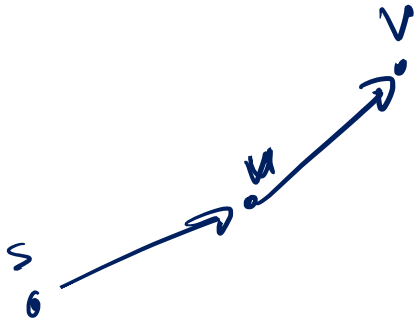
repeat

  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?
  - Kürzeste Pfade mit einer Kante $\Longrightarrow$ 1 Wiederholung
  - Kürzeste Pfade mit zwei Kanten $\Longrightarrow \leq$ 2 Wiederholungen
  - …
  - Kürzeste Pfade mit k Kanten $\Longrightarrow \leq k$ Wiederholungen

Bellman-Ford Algorithmus

$\delta(s,s) := 0;\ \forall v \in V \setminus \{s\} : \delta(s,v) := \infty$

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

- **Folgt per Induktion über i:**
  - $i = 0$: $\delta(s,s) = d_G^{(0)}(s,s) = 0, v \neq s \implies \delta(s,v) = d_G^{(0)}(s,v) = \infty$
  - $i > 0$:

    $$d_G^{(i)}(s,v) = \min\left\{d_G^{(i-1)}(s,v), \min_{u \in N^{in}(v)} d_G^{(i-1)}(s,u) + w(u,v)\right\}$$

    (kürzester Weg besteht aus $\leq i-1$ Kanten oder aus genau i Kanten)

Bellman-Ford Algorithmus

$\delta(s,s) := 0;\ \forall v \in V \setminus \{s\} : \delta(s,v) := \infty$

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**

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

Theorem: Falls der Graph keine negativen Kreise enthält, sind am Schluss alle Distanzen korrekt berechnet.

- Am Schluss haben wir für alle $v \in V$:

$$\delta(s,v) \leq d_G^{(n-1)}(s,v)$$

- Weil jeder kürzeste Pfad $\leq n-1$ Kanten besteht, gilt ausserdem

$$d_G^{(n-1)}(s,v) = d_G(s,v)$$

Negative Kreise erkennen

- Wir werden sehen: Falls es einen (von s erreichbaren) negativen Kreis hat, dann gibt es für irgendeine Kante eine Verbesserung.

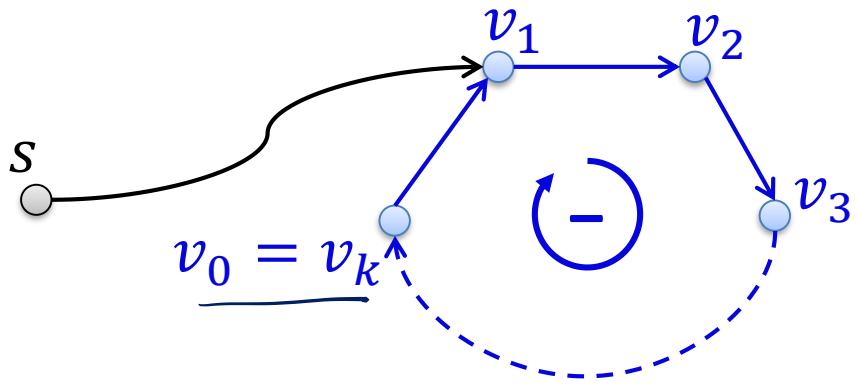
$$\exists (u,v) \in E : \delta(s,u) + w(u,v) < \delta(s,v)$$

Bellman-Ford Algorithmus

```
for i := 1 to n-1 do
    for all (u,v) ∈ E do
        if δ(s,u) + w(u,v) < δ(s,v) then
            δ(s,v) := δ(s,u) + w(u,v)
for all (u,v) ∈ E do
    if δ(s,u) + w(u,v) < δ(s,v) then
        return false
return true
```

Negative Kreise erkennen

Lemma: Falls G einen von s erreichbaren negativen Kreis enthält, dann gibt der Bellman-Ford Algorithmus false zurück.



neg. Kreis $\Rightarrow \sum_{i=1}^{k} w(v_{i-1}, v_i) \overset{!}{<} 0$

von s erreichbar $\Rightarrow \delta(s, v_i) \neq \infty$

Widerpruchsbeweis:

- Annahme : $\forall i \in \{1, \dots, k\} : \delta(s, v_{i-1}) + w(v_{i-1}, v_i) \geq \delta(s, v_i)$

$$\sum_{i=1}^{k} \delta(s, v_i) \leq \sum_{i=1}^{k} \big(\delta(s, v_{i-1}) + w(v_{i-1}, v_i)\big)$$

$$= \sum_{i=1}^{k} \delta(s, v_{i-1}) + \sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

Bellman-Ford : Kürzeste Pfade

Ein Shortest Path Tree kann in der üblichen Art konstruiert werden.

Initialisierung:

- $\delta(s,s) = 0$, für $v \neq s : \delta(s,v) = \text{NULL}$
- $\alpha(s) = s$ (Wurzel zeigt auf sich selbst), für $v \neq s : \alpha(v) = \text{NULL}$

In jedem Schleifendurchlauf:

$\cdots$

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

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

$\alpha(v) := u$

- Am Schluss zeigt $\alpha(v)$ zum Parent in einem Shortest Path Tree
  - falls es keine negativen Kreise hat...

Bellman-Ford Alg.: Zusammenfassung

Theorem: Falls es einen von s erreichbaren negativen Kreis hat, wird dies vom Bellman-Ford Algorithmus erkennt. Falls kein solcher negativer Kreis existiert, berechnet der Bellman-Ford Algorithmus in Zeit $O(|V| \cdot |E|)$ einen Shortest Path Tree.

- **Korrektheit:** haben wir schon gezeigt.
- **Laufzeit:**
  - $n - 1 + 1$ Schleifendurchläufe
  - In jedem Schleifendurchlauf gehen wir einmal durch alle Kanten
- **Bemerkung:** Man kann den Algorithmus einfach so abändern, dass er für alle v, für welche ein kürzester Pfad von s existiert, einen solchen Pfad berechnet.

Routing-Pfade in Netzwerken

UNI FREIBURG

Ziel: Optimale Routing-Pfade zu einer Destination t

- Von jedem Knoten aus wollen wir wissen, zu welchem Nachbar einen Nachricht geschickt werden muss.
- Entspricht einem Shortest Path Tree, falls alle Kanten umgedreht werden (transponierter Graph)

Algorithmus:

- Knoten merken sich aktuelle Distanz $\delta(u,t)$ und den aktuell besten Nachbar
- Alle Knoten schauen gleichzeitig (parallel), ob's bei irgendeinem Nachbar eine Verbesserung gibt

$$\exists (u,v) \in E : w(u,v) + \delta(v,t) < \delta(u,t)$$

- entspricht einer parallelen Version des Bellman-Ford Algorithmus

Kürzeste Wege zw. allen Knotenpaaren

- **all pairs shortest paths problem**

Berechne single-source shortest paths für alle Knoten

$\frac{n^3}{2^{O(\sqrt{\log n})}} = n^{3-o(1)}$

- Dijkstras Algorithmus mit allen Knoten:

  Laufzeit: $n \cdot O(\text{Laufzeit Dijkstra}) \in O(mn + n^2 \log n) = O(n^3)$

  – Problem: funktioniert nur bei nichtnegativen Kantengewichten

- Bellman-Ford Algorithmus mit allen Knoten:

  Laufzeit: $n \cdot O(\text{Laufzeit BF}) \in O(mn^2) \in O(n^4)$

  – Problem: langsam...

  – Wenn man den Bellman-Ford Algorithmus gleichzeitig für alle Knoten ausführt, kann man die Laufzeit auf $O(n^3 \cdot \log n)$ verbessern.

  – Wenn man alle $d_G^{(i)}(u,v)$-Distanzen kennt, kann man in einem Durchlauf direkt die $d_G^{(2i)}(u,v)$-Distanzen berechnen.

    - Details und Diskussion weiterer Verbesserungen, z.B. in der Vorlesung von 2018.