Informatik II - SS 2018

(Algorithmen & Datenstrukturen)

Vorlesung 16 (18.6.2018)

Graphenalgorithmen V (Kürzeste Wege)

UNI FREIBURG

Fabian Kuhn

Algorithmen und Komplexität

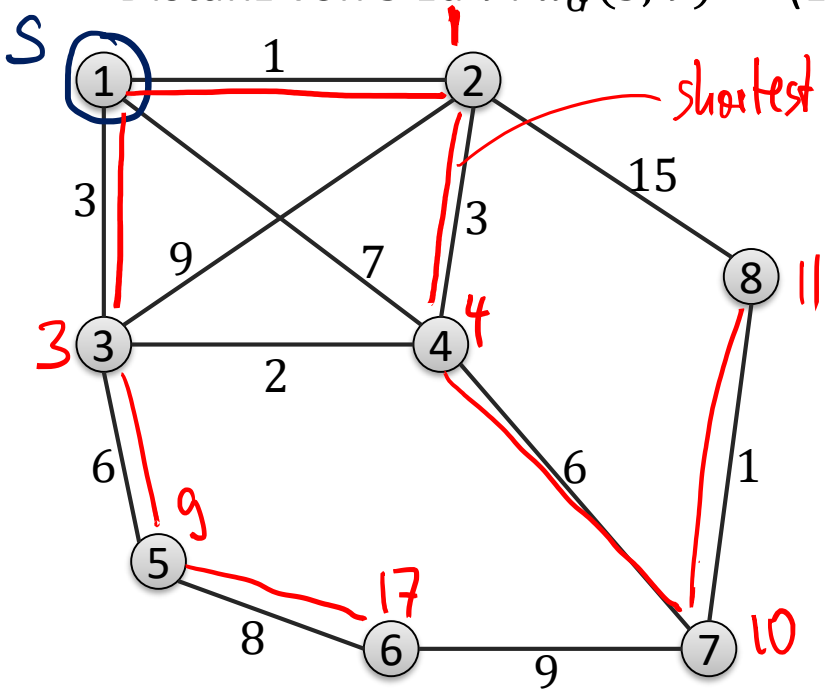
Kürzeste Wege

Annahme: G zusammenhängend

gerichteter Graph

Problem

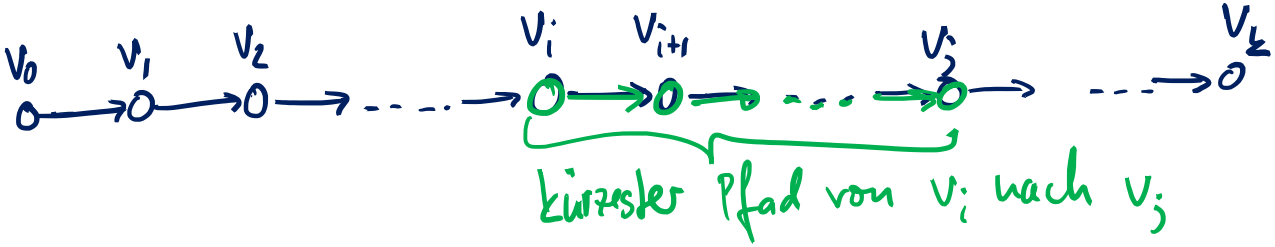
- Gegeben: gewichteter Graph $G = (V, E, w)$, Startknoten $s \in V$
  - Wir bezeichnen Gewicht einer Kante (u, v) als $w(u, v)$
  - Annahme: $\forall e \in E: w(e) \geq 0$
- Ziel: Finde kürzeste Pfade / Distanzen von s zu allen Knoten
  - Distanz von s zu v: $d_G(s, v)$ (Länge eines kürzesten Pfades)

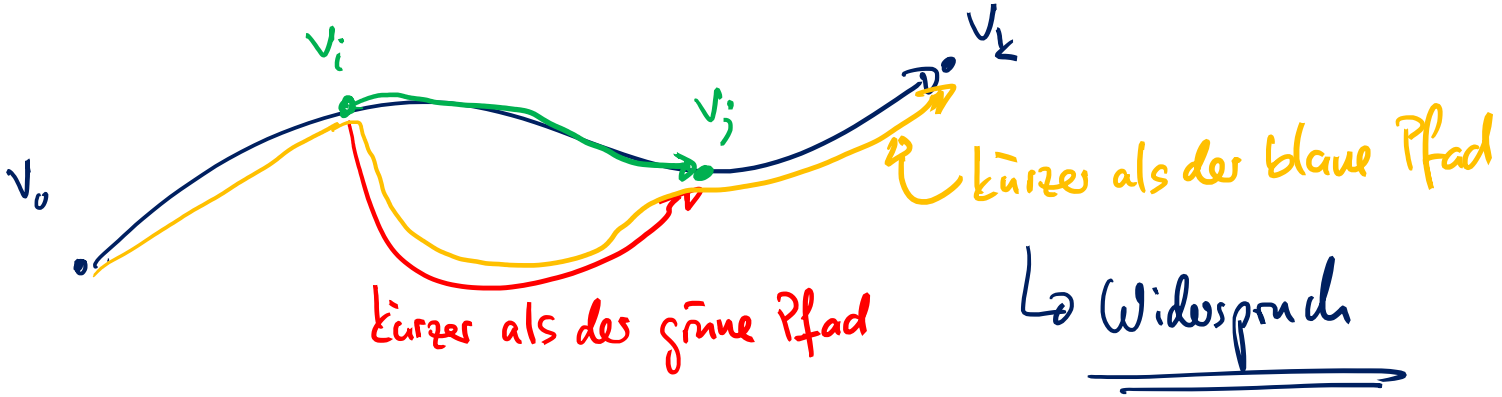


Optimalität von Teilpfaden

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.

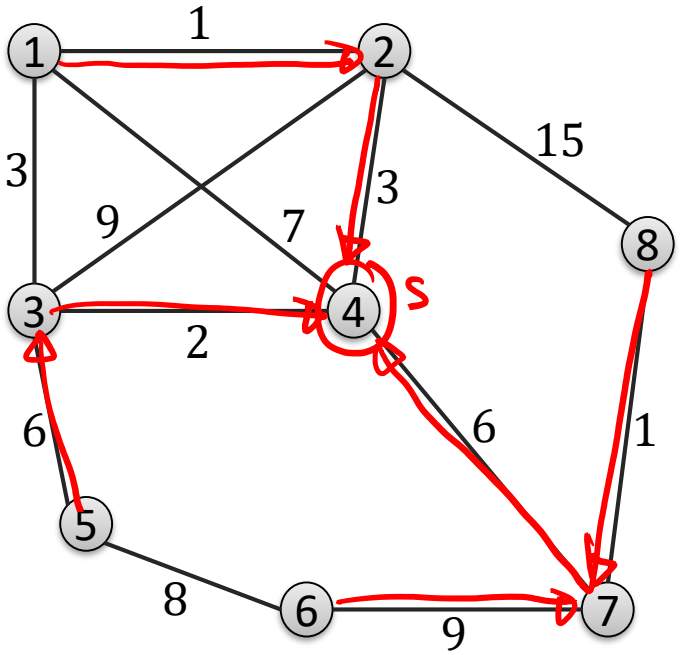
- gilt auch bei negativen Kantengewichten...





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" (SPT)



Wir wollen für alle Knoten v $d_G(s,v)$ und den Parent-Knoten in einem SPT finden

Dijkstras Algorithmus: Idee

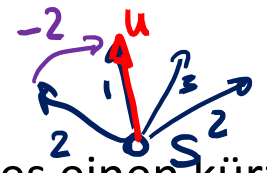
- Algorithmus von Edsger W. Dijkstra (1959 publiziert)

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".

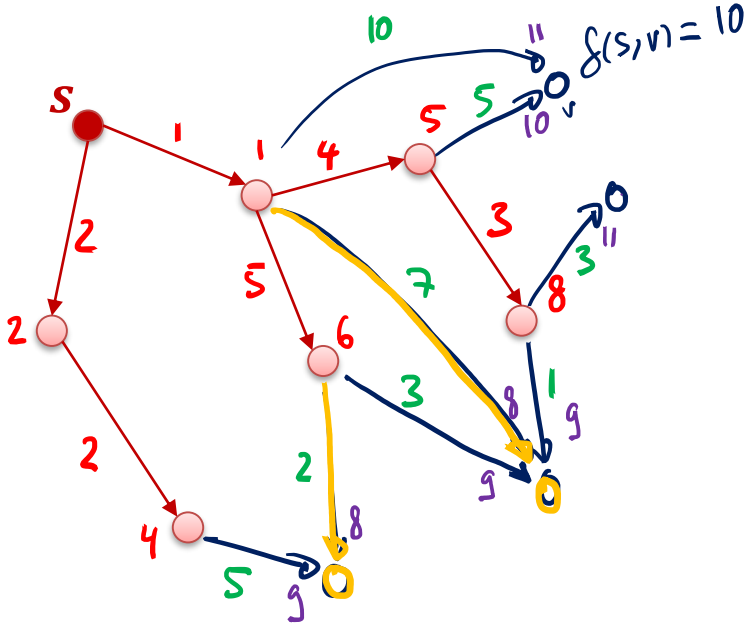
- Ziel: In jedem Schritt des Algorithmus einen Knoten hinzufügen
  - Am Anfang: Teilbaum besteht nur aus s (erfüllt Invariante trivialerweise...)
  - 1. Schritt: Wegen der Optimalität der Teilpfade, muss es einen kürzesten Pfad bestehend aus nur einer Kante geben...
  - Füge Knoten mit kleinstem Abstand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

Wie können wir T um einen Knoten erweitern?



u im Baum T, $v \notin T$, Kante (u,v)

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

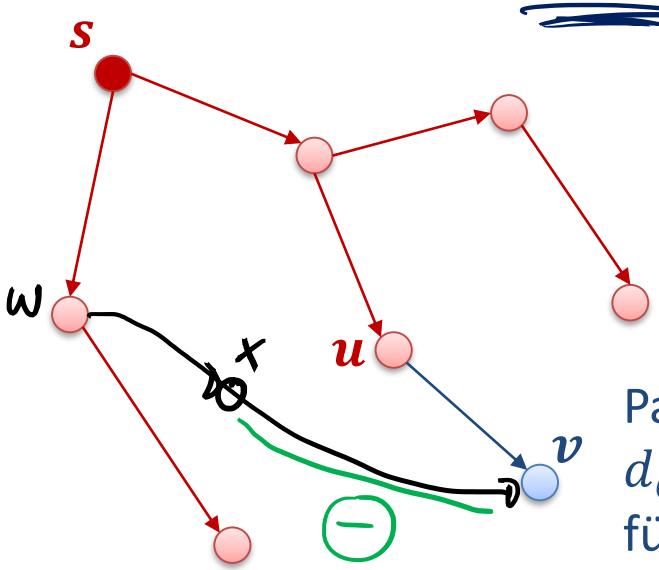
Nehme Knoten $u \in T$, $v \notin T$, sodass $(u,v) \in E$ und so dass $d_G(s,u) + w(u,v)$ minimiert wird

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:

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



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

Länge des schw. Pfades $\geq$ $d_G(s,w) + w(w,x) \geq d_G(s,u) + w(u,v)$

keine neg. Kantengewichte

Dijkstras Algorithmus

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

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

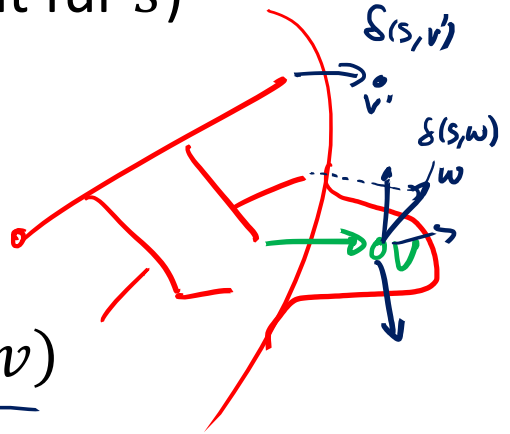
  - sowie den Eingangsnachbar $u =: \alpha(v)$, welcher den Ausdruck minimiert...
- Invariante $\Longrightarrow \delta(s,v) \geq d_G(s,v)$
- Lemma auf letzter Folie:

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

Dijkstras Algorithmus

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

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



Interationsschritt

- Wähle Knoten v mit kleinstem

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

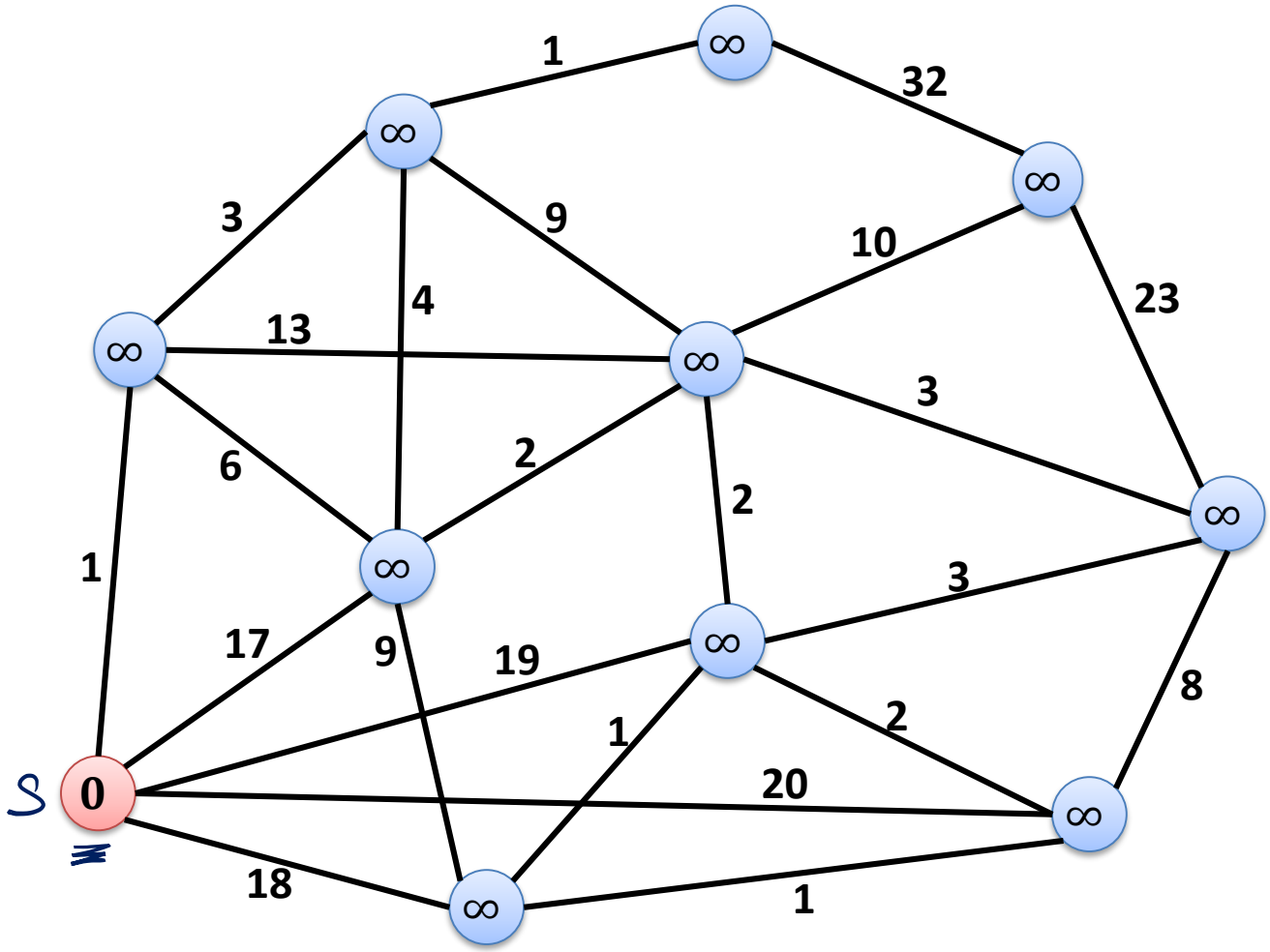
- Gehe durch die Ausgangsnachbarn $x \in V \setminus T$ und setze

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

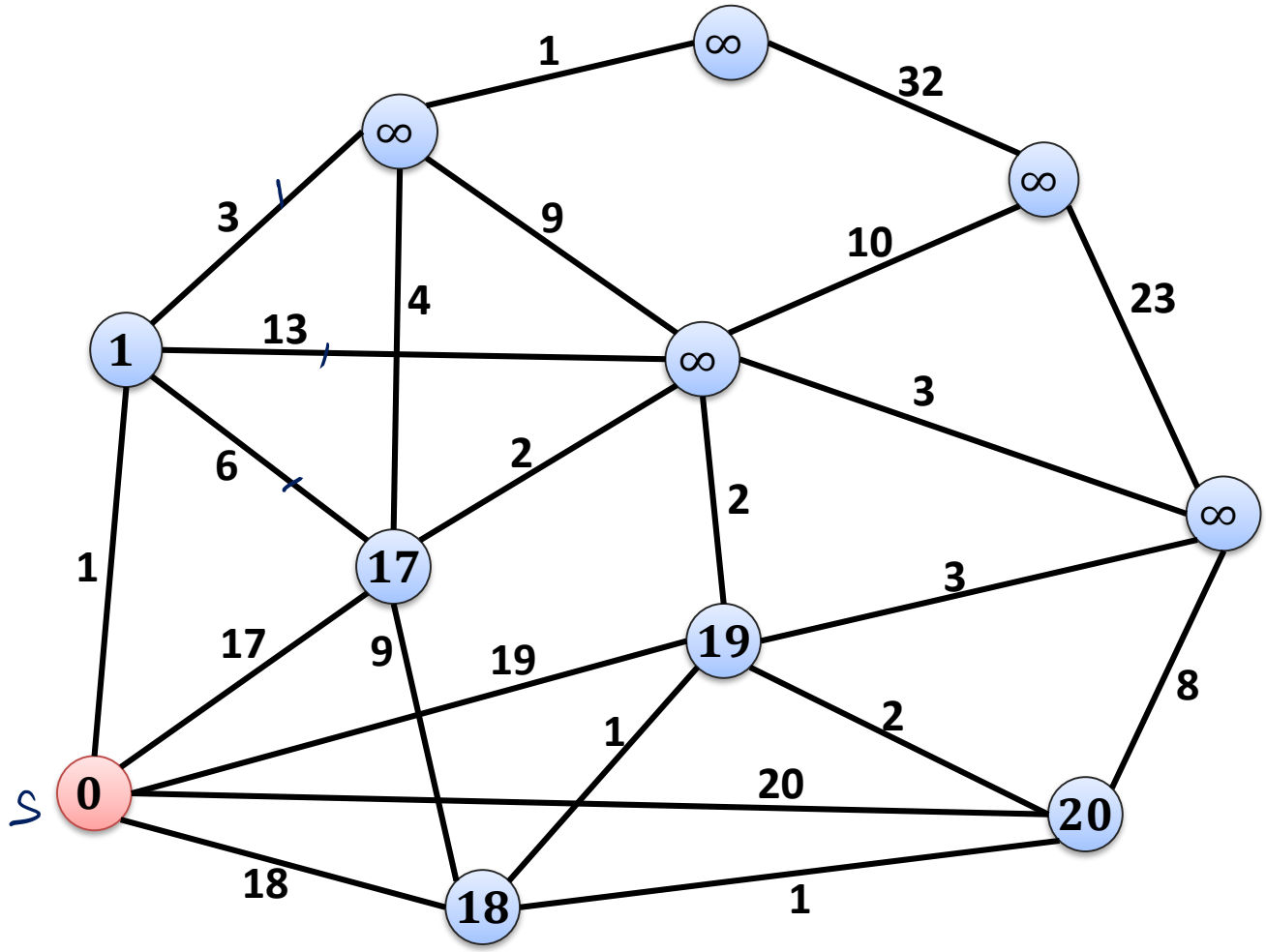
  - Falls nötig, setze auch $\alpha(x) = v$

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

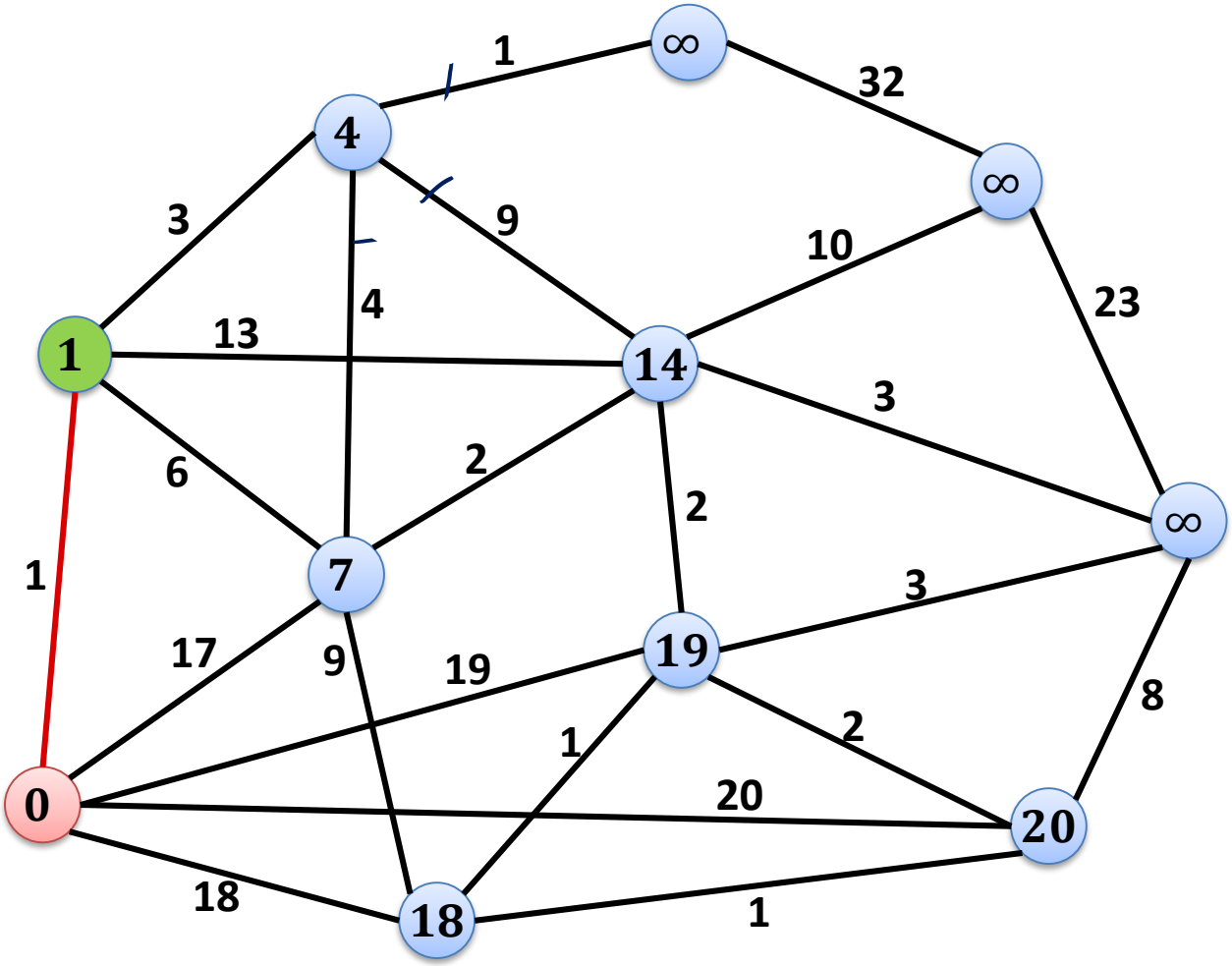
Dijkstras Algorithmus: Beispiel



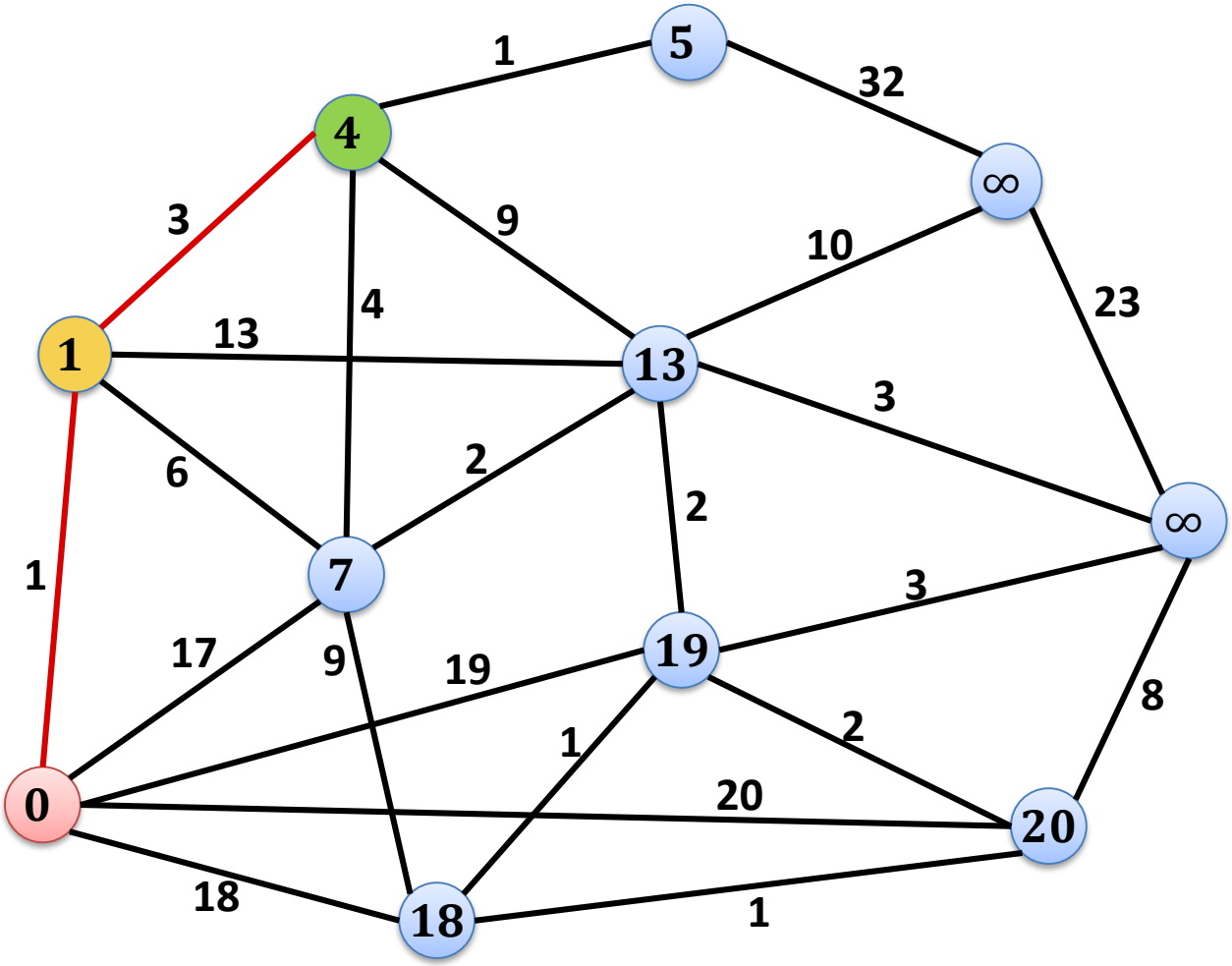
Dijkstras Algorithmus: Beispiel



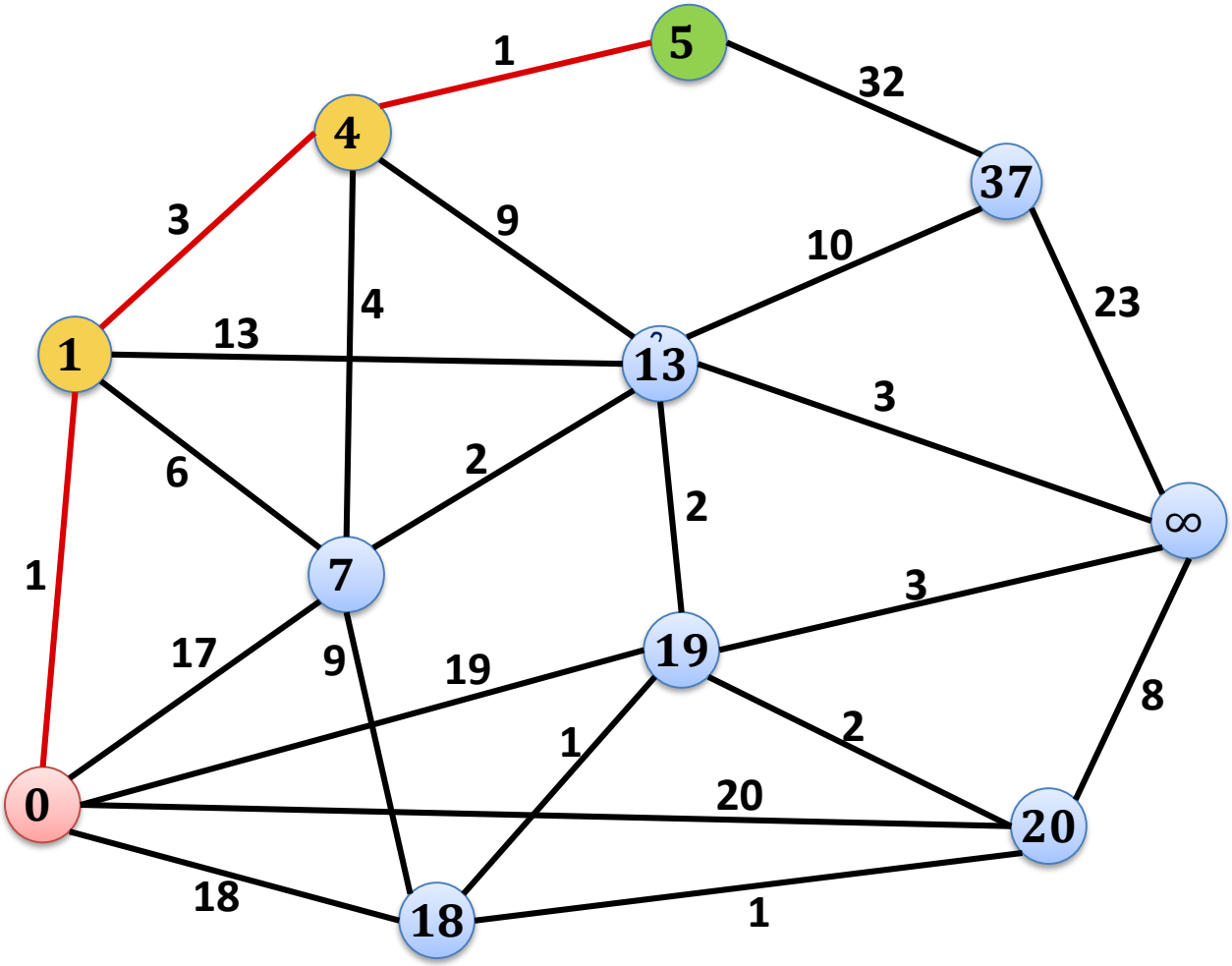
Dijkstras Algorithmus: Beispiel



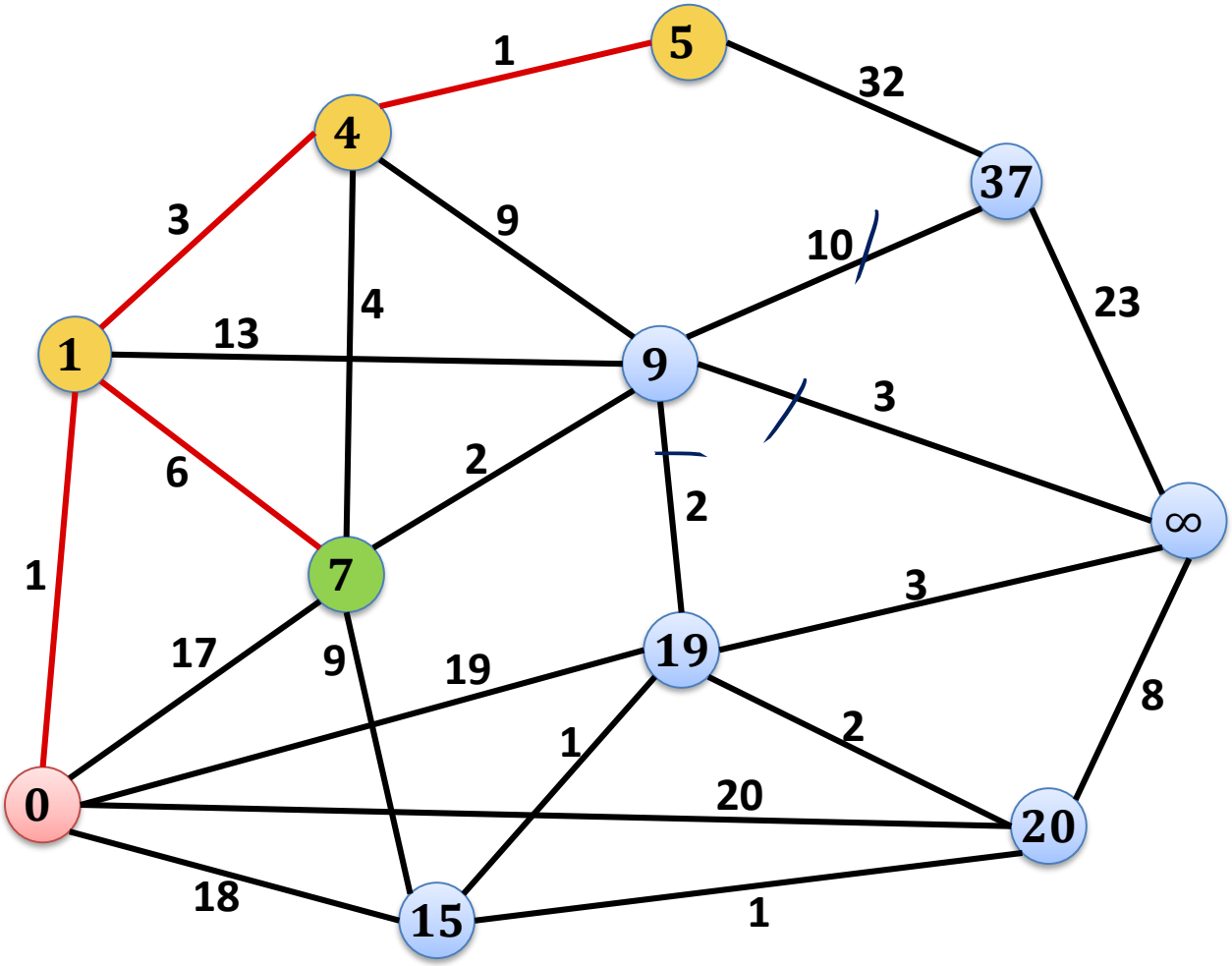
Dijkstras Algorithmus: Beispiel



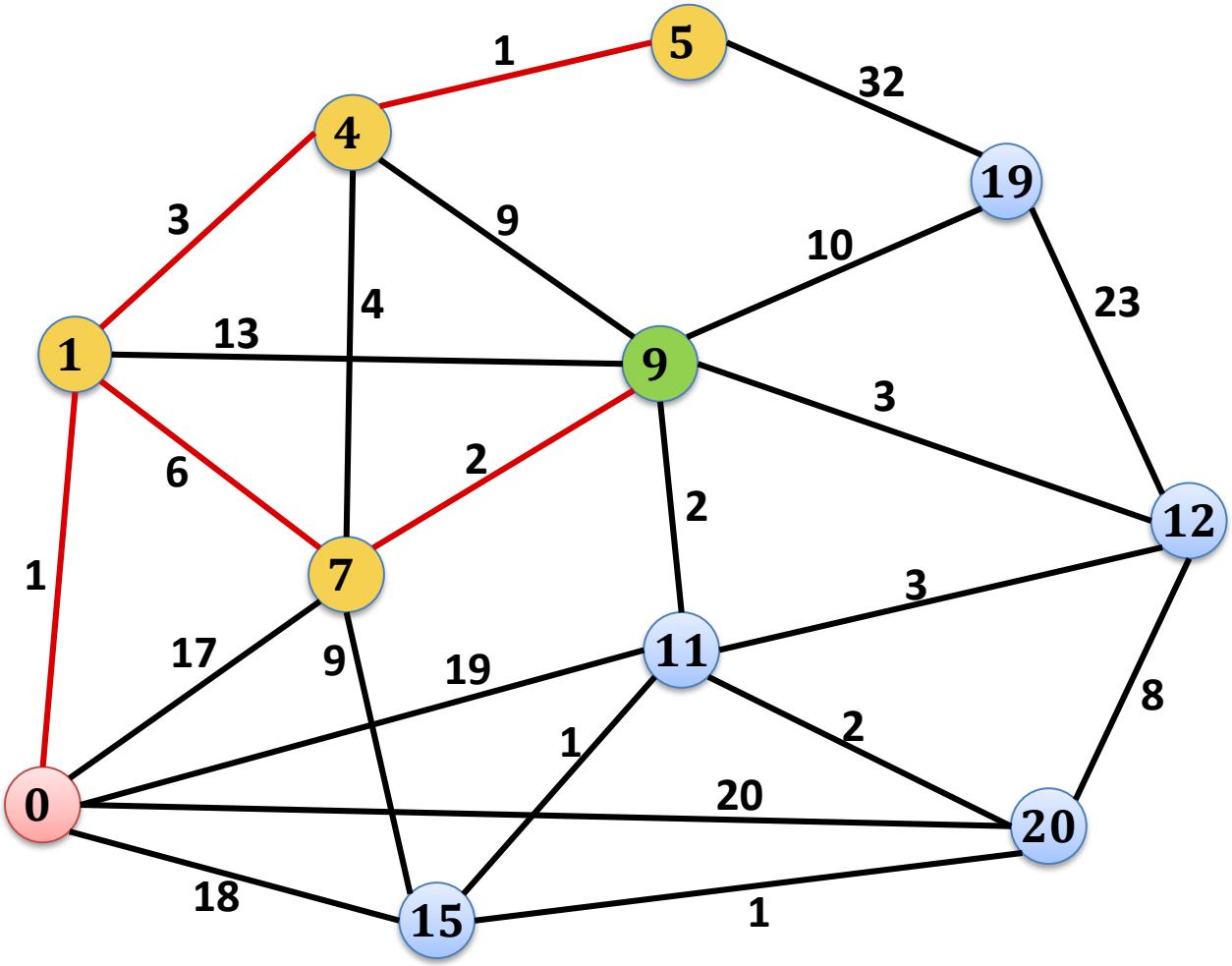
Dijkstras Algorithmus: Beispiel



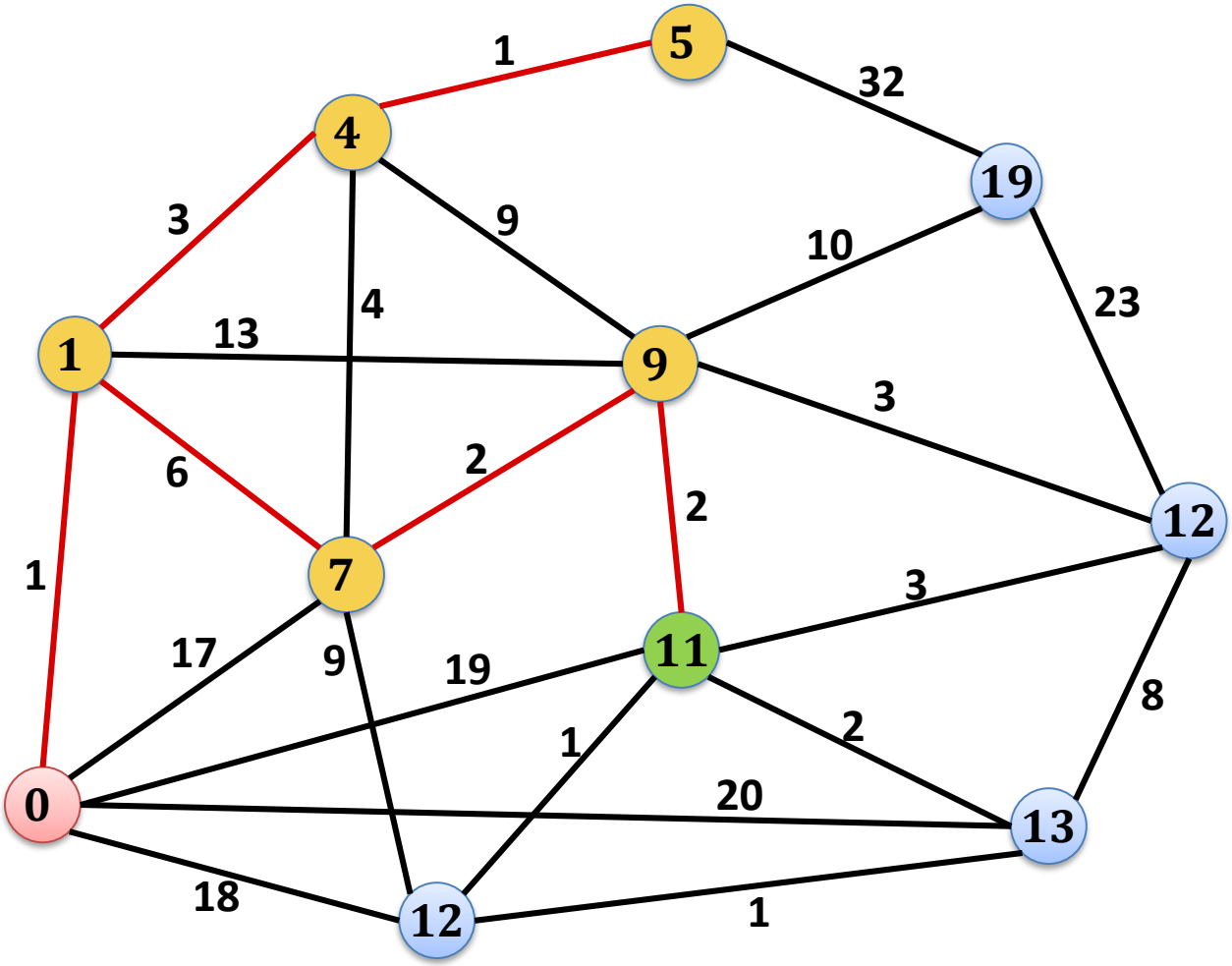
Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Beispiel



Dijkstras Algorithmus: Implementierung

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)

ähnlich wie der MST Alg. von Prim

Interationsschritt

- 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.

Erinnerung: Prioritätswarteschlangen

UNI FREIBURG

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: Prims MST Algorithmus

UNI FREIBURG

```
H = new priority queue; A = ∅
for all u ∈ V \ {s} do
    H.insert(u, ∞); α(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}))
            α(v) = u
    if u ≠ s then A = A ∪ {{u, α(u)}}
```

Dijkstras Algorithmus: Implementierung

UNI FREIBURG

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

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

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

H.insert(s, 0)

while H is not empty **do**

    u = H.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, $\delta(s,v)$)

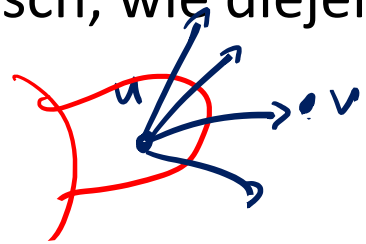
            $\alpha(v) = u$

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

v unmarked (v in priority queue)

Dijkstas Algorithmus: Laufzeit

- Algorithmus-Implementierung ist fast identisch, wie diejenige von Prims MST Algorithmus



- **Anzahl Heap-Operationen:**

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

  $O(\log n)$ $O(\log n)$ $O(\log n)$

- **Laufzeit mit binären Heaps:**

$$O(m \log n)$$

$m \geq n-1$

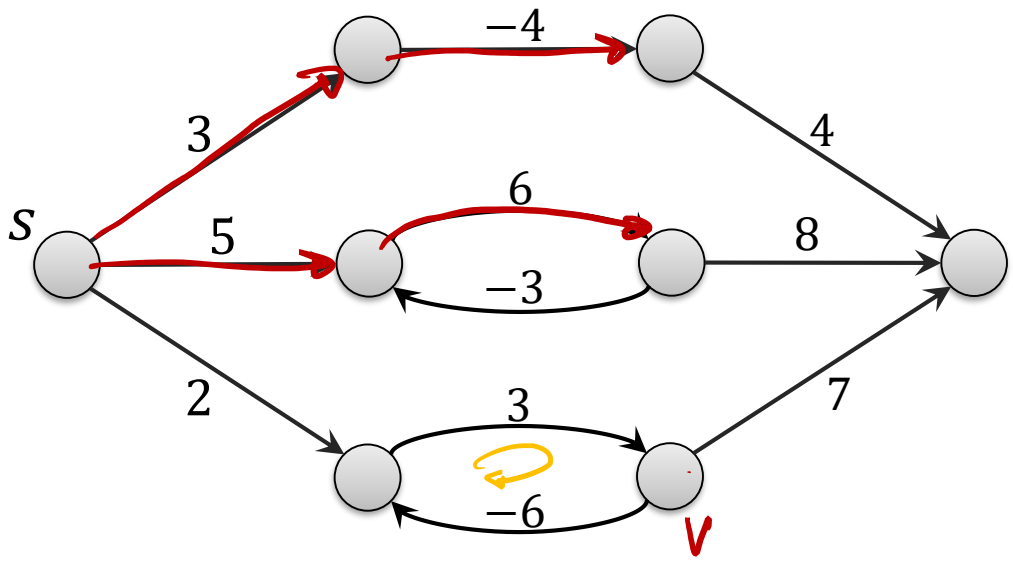
- **Laufzeit mit Fibonacci Heaps:**

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

Negative Kantengewichte

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

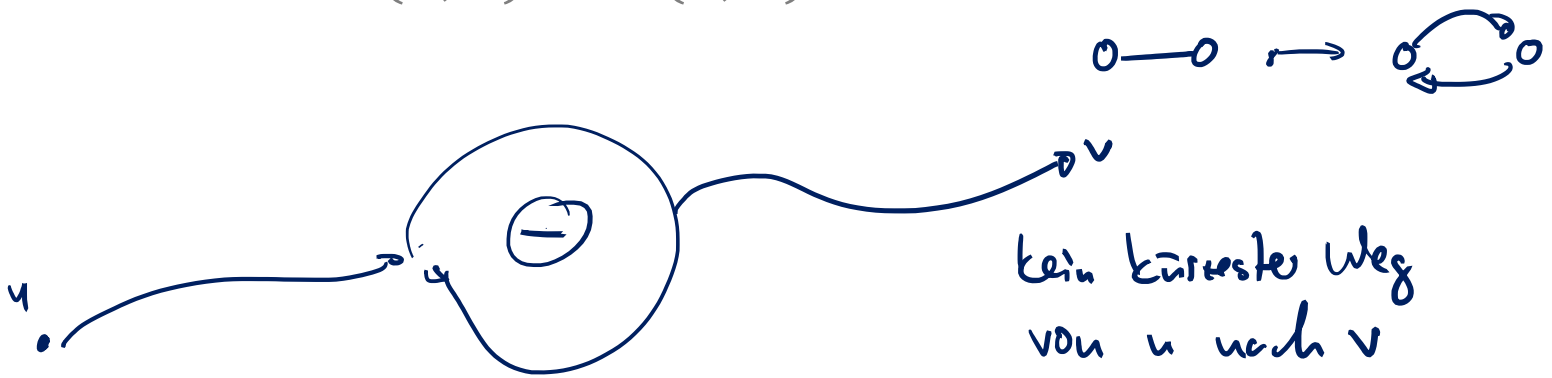
Beispiel



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



kein erreichbarer neg. Kreis

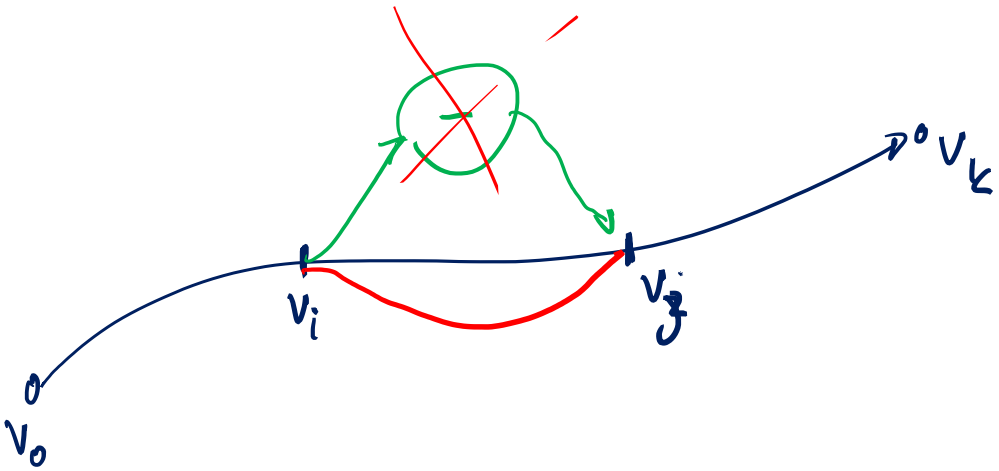
- Wir können uns auf einfache Pfade beschränken (es hilft nichts, den gleichen Knoten 2x zu besuchen

davon gibt es nur endlich viele

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.

- **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 ≤ 1 Kante von T' am besten erreicht werden kann und nimmt in jedem Schritt den besten solchen Knoten zu T' hinzu.

Dijkstras Algorithmus

UNI FREIBURG

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)

Interationsschritt

- 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

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

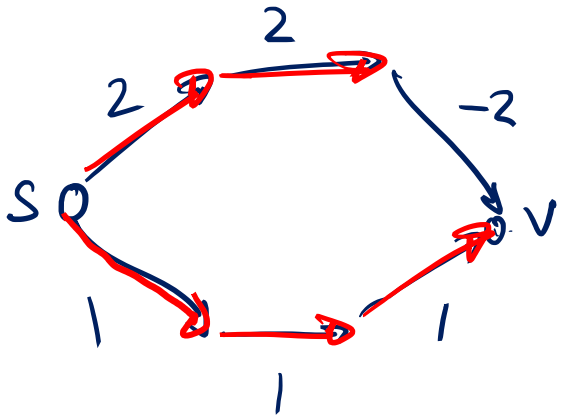
  - Falls nötig, setze auch $\alpha(x) = v$

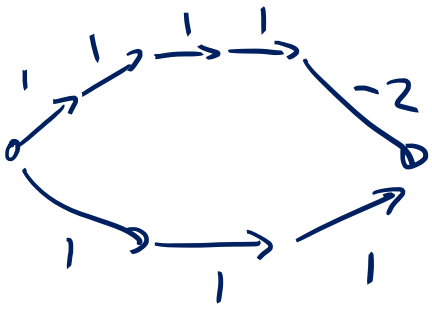
- Füge Kante $(\alpha(v), v)$ zum Baum T hinzu.

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: $\delta(s,s) = 0, \quad \delta(s,v) = \infty$ 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 $\boldsymbol{d_G(s,v) \leq \delta(s,u) + w(u,v)}$.

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

Bellman-Ford

- 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**

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

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

- Wieviele Wiederholungen sind nötig?

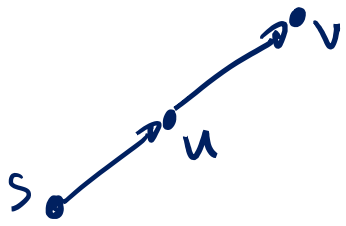
Falls kürzester Pfad von s nach v nur 1 Kante hat

$\longrightarrow \delta(s,v) = d_G(s,v)$ nach 1 Iteration

Falls kürzester Pfad von s nach v ≤ 2 Kanten hat

$\longrightarrow \delta(s,v) = d_G(s,v)$ nach 2 Iterationen

nach 1 Iter: $\delta(s,u) = d_G(s,u)$



Bellman-Ford

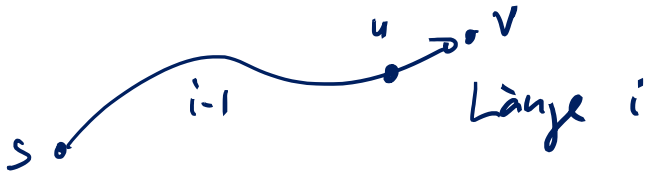
- Betrachte alle Kanten (u,v) und versuche $\delta(s,v)$ zu verbessern
  - solange, bis alle Distanzen korrekt sind ($\forall v \in V: \delta(s,v) = d_G(s,v)$)

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

per Induktion nach i

$i=0$ ✓



Bellman-Ford

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

keine neg. Kreise ⟹ $d_G^{(n-1)}(s,v) = d_G(s,v)$

neg. Kreise: von s erreichbar

können uns auf einfache Pfade beschränken

⟹ nach $n-1$ Wiederholungen

$\forall v: \quad \delta(s,v) = d_G^{(i-1)}(s,v) = d_G(s,v)$

Negative Kreise erkennen

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

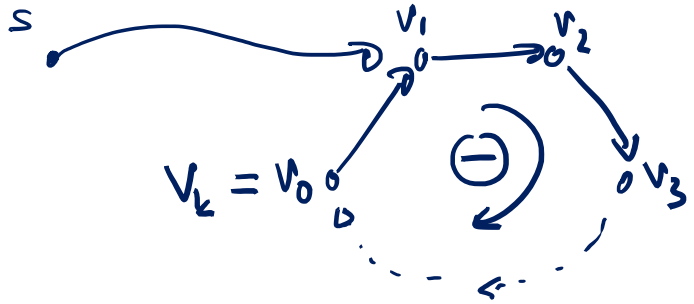
$$\exists (u,v) \in E : \delta(s,u) + w(u,v) < \delta(s,v)$$

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

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

Widerspruchsbeweis

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

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

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

< 0

Widerspruch!

□

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 dieser 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.

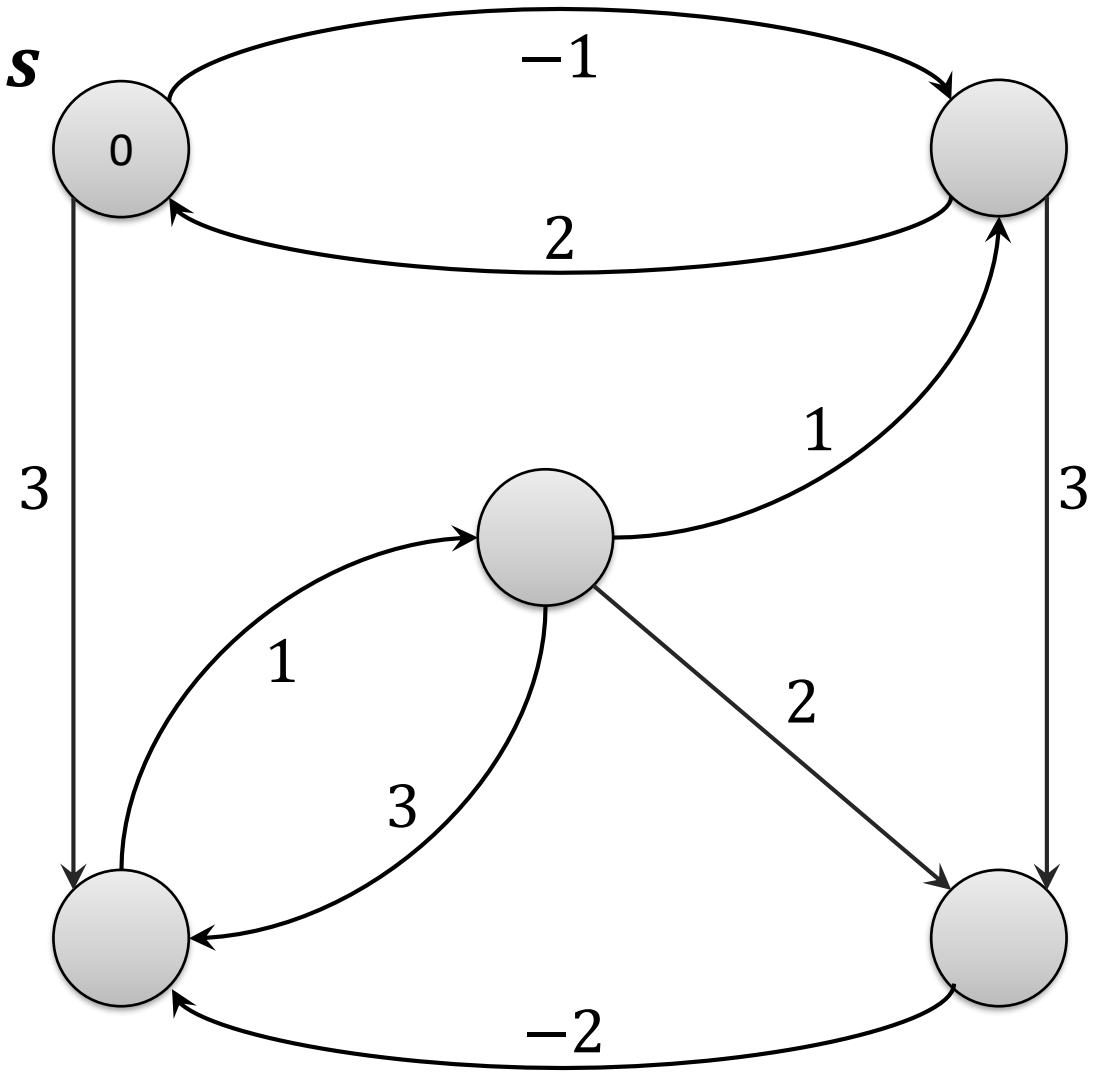
Laufzeit

```
for i = 1 to n-1                          O(m·n)
    for all (u,v) ∈ E
        ⋮                                 } O(1)
```

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

- Man kann den Algorithmus einfach so abändern, dass er für alle v, für welche ein kürzester Pfad von s existiert, einen solchen Pfad berechnet.

Bellman-Ford Algorithmus: Beispiel



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