Informatik II - SS 2014

(Algorithmen & Datenstrukturen)

Vorlesung 19 (22.7.2014)

Kürzeste Wege:
Negative Gewichte, alle Paare

UNI FREIBURG

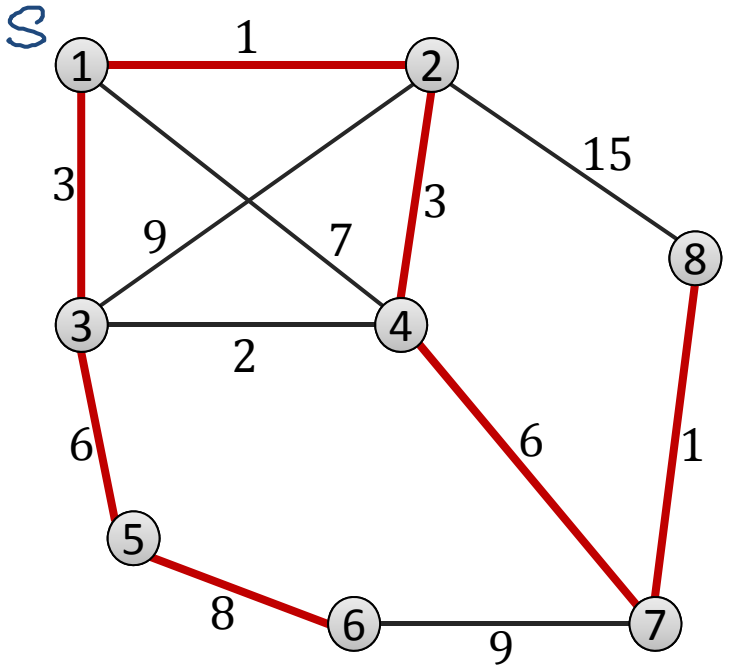
Fabian Kuhn

Algorithmen und Komplexität

Kürzeste Wege

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)

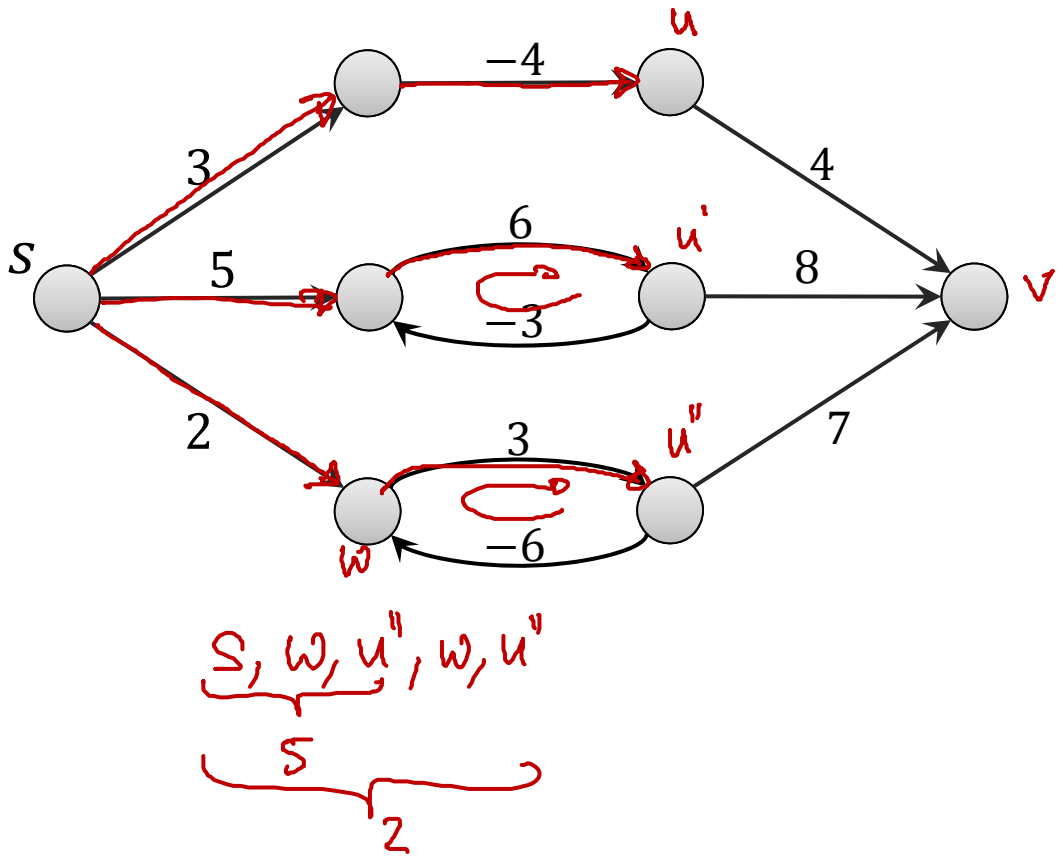


shortest path tree

Negative Kantengewichte

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

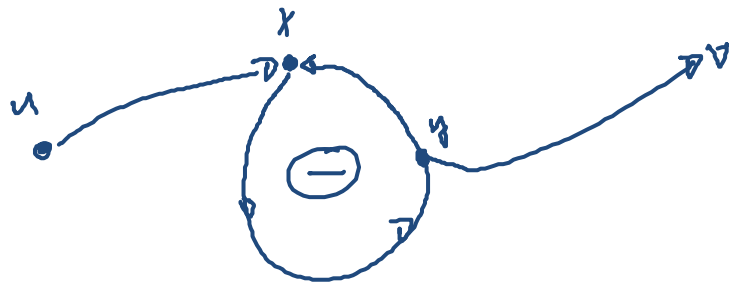
Beispiel



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

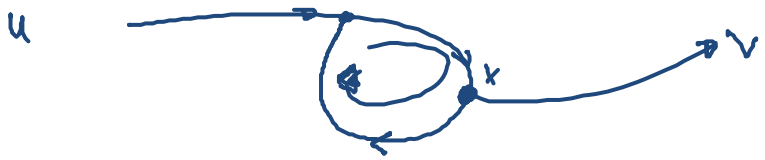


kein kürzester Pfad von u nach v

kein erreichbarer neg. Kreis

- können uns auf einfache Pfade beschränken (es hilft nie 2x den gleichen Knoten zu besuchen)

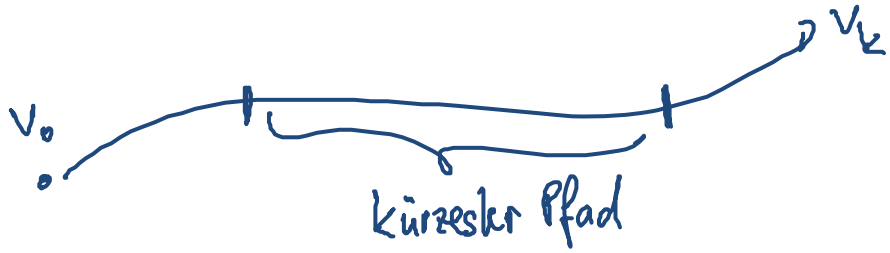
davon gibt's nur endlich viele



Optimalität von Teilpfaden

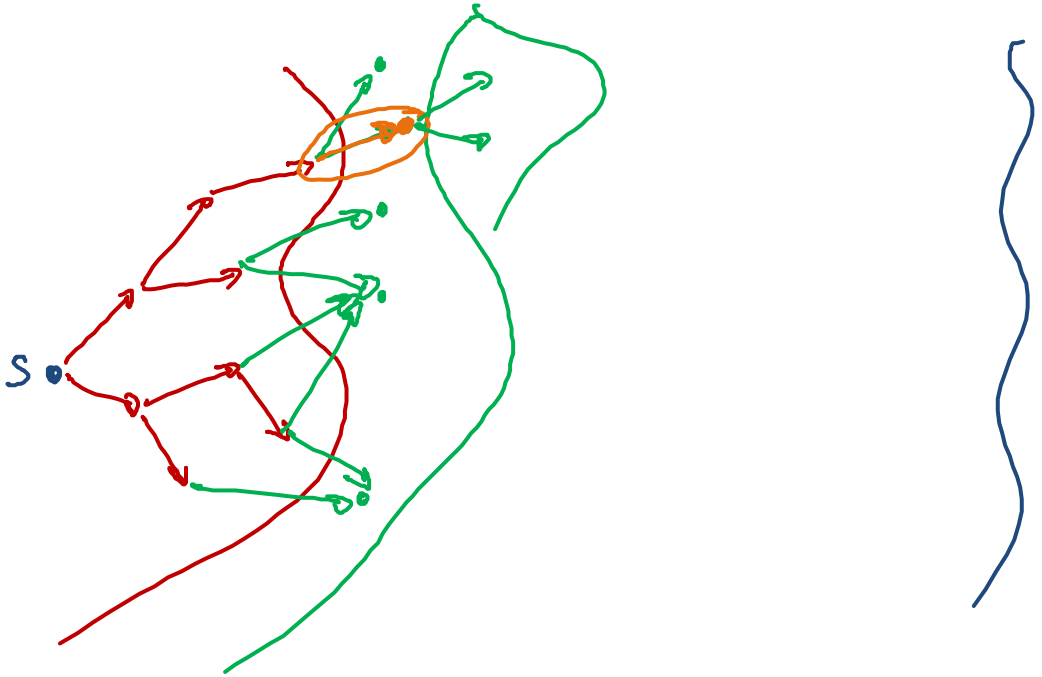
Lemma: Falls $v_0, v_1, \dots, v_k$ ein kürzester Pfad von v_0 nach v_k ist, dann gilt für alle $0 \leq i \leq j \leq k$, dass der Teilpfad $v_i, v_{i+1}, \dots, v_j$ ein kürzester Pfad von v_i nach v_j ist.

- **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

$\delta(s,v) \geq d_G(s,v)$

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

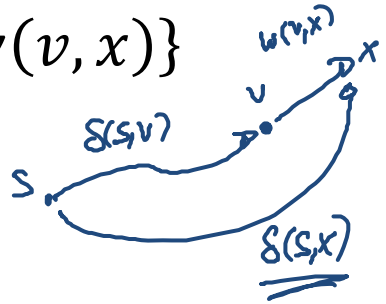
- Wähle Knoten v mit kleinstem

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

- 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 $\alpha(x) = v$

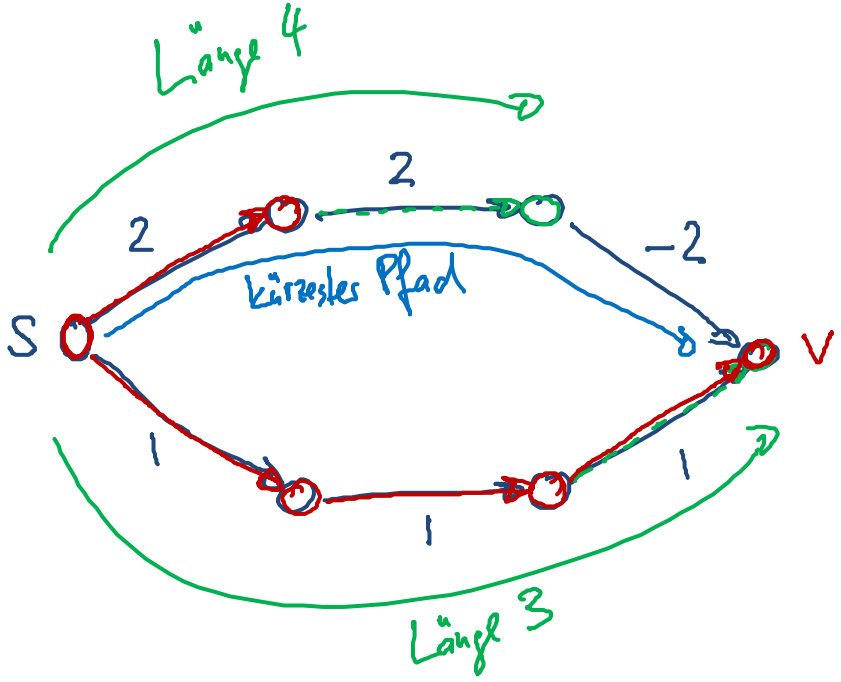


- Füge Kante $(\alpha(v), v)$ zum Baum T hinzu.

Dijkstras Algorithmus und negative Gewichte

Funktioniert Dijkstras Algorithmus auch bei negativen Kantengewichten?

- Antwort: nein



Bellman-Ford Algorithmus

- Zur Vereinfachung beschränken wir uns darauf, die Distanzen zu berechnen

Annahme:

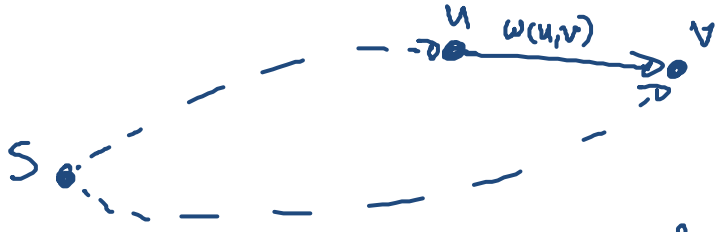
- Für alle Knoten v: Algorithmus hat Wert $\boldsymbol{\delta(s,v) \geq d_G(s,v)}$
- Initialisierung: $\delta(s,s) = 0, \quad \delta(s,v) = \infty$ für $v \neq s$

Beobachtung:

- Falls $(u,v) \in E$, so dass $\delta(s,u) + w(u,v) < \delta(s,v)$, dann gilt:

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

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



$d_G(s,u) \leq \delta(s,u)$

$d_G(s,v) \leq \delta(s,v)$

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

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

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**

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

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

Fall kürzestes Pfad von s nach v ≤2 Kanten hat
→ $\delta(s,v) = d_G(s,v)$ nach 2 Iterationen

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) ∈ E do
        if δ(s,u) + w(u,v) < δ(s,v) then
            δ(s,v) := δ(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 ($i=0$, $i=1$ ✓)

$i \geq 1$, $i-1 \to i$

Pfad der Länge $d_G^{(i)}(s,v)$ ist, welcher $\leq i$ Kanten hat

Ind.-Annahme vor Iter. i: $\delta(s,u) = d_G^{(i-1)}(s,u)$ ←

$d_G^{(i)}(s,v) = d_G^{(i-1)}(s,u) + w(u,v)$

betr. Kante (u,v): $\delta(s,v) \leq \delta(s,u) + w(u,v)$

$= d_G^{(i-1)}(s,u) + w(u,v)$

Bellman-Ford

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

keine neg. Zyklen $\Rightarrow d_G^{(n-1)}(s,v) = d_G(s,v)$

neg. Zyklen: von s erreichbar

$d_G(s,v)$: können uns auf einfache Pfade beschränken

$\Rightarrow$ nach $n-1$ Wiederholungen

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

Negative Kreise erkennen

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

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

Bellman-Ford Algorithmus

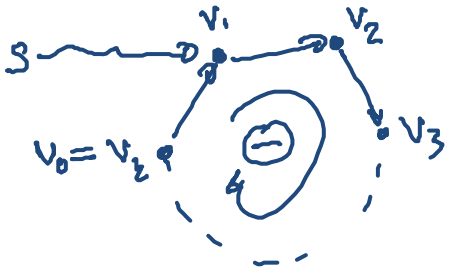
```
for i := 1 to n-1 do
    for all (u,v) ∈ E do
        if δ(s,u) + w(u,v) < δ(s,v) then
            δ(s,v) := δ(s,u) + w(u,v)
for all (u,v) ∈ E do
    if δ(s,u) + w(u,v) < δ(s,v) then
        return false
return true
```

wie vorher

keine neg. Kreise $\Rightarrow \delta(s,v) = d_G(s,v)$

Negative Kreise erkennen

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



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

Widerspruchsbeweis

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

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

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

$$< 0$$

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:

...

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 erkannt. Falls kein solcher negativer Kreis existiert, berechnet der Bellman-Ford Algorithmus in Zeit $O(|V| \cdot |E|)$ einen Shortest Path Tree.

Laufzeit

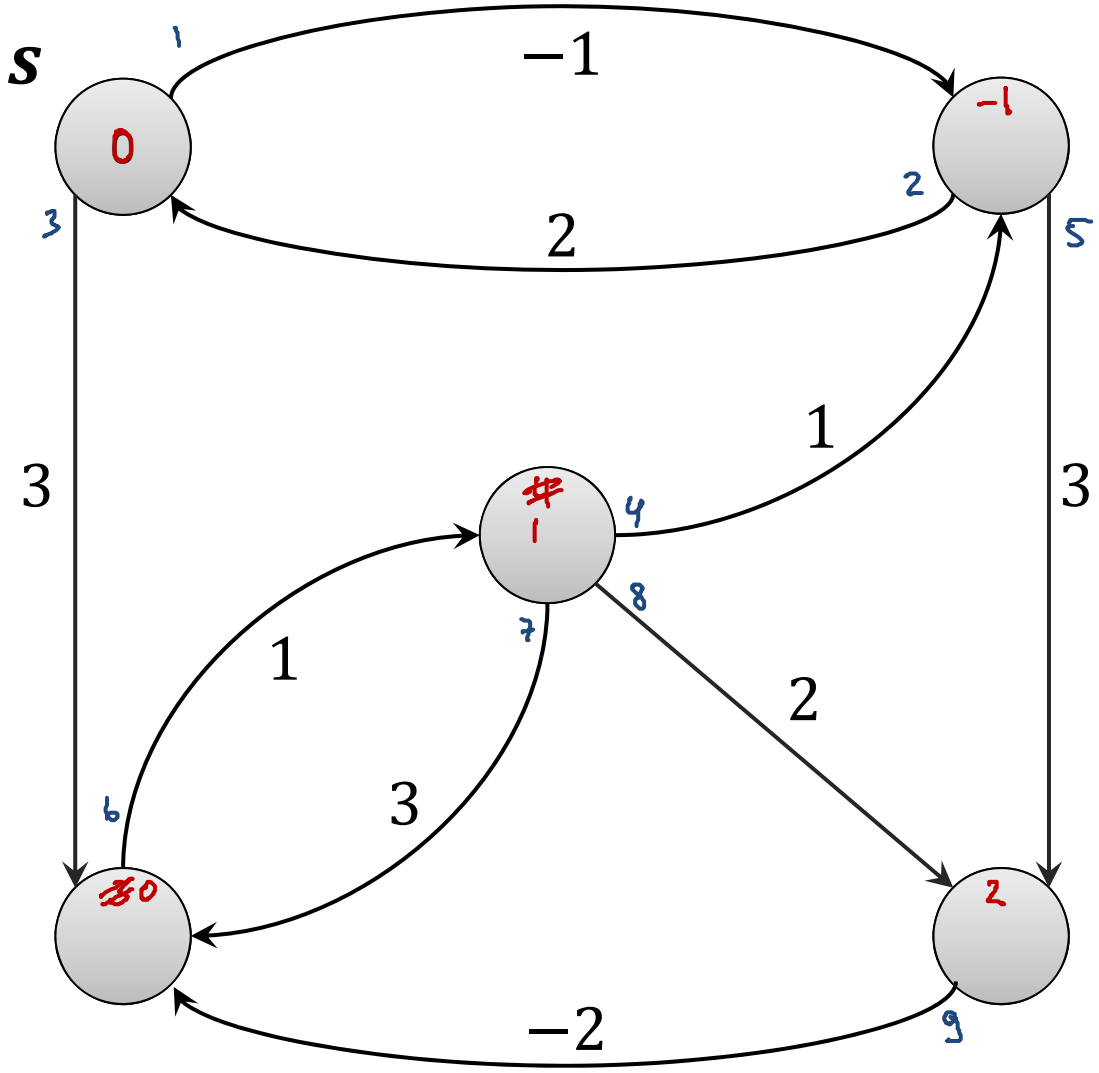
for i := 1 to n-1
  for all edges (u,v)
    δ(u,v) = ... } O(1)

n = |V|
m = |E|

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

Kürzeste Wege zw. allen Knotenpaaren

UNI FREIBURG

- all pairs shortest paths problem

Berechne single-source shortest paths für alle Knoten

- Dijkstras Algorithmus mit allen Knoten:

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

  – Problem: funktioniert nur bei nichtnegativen Kantengewichten

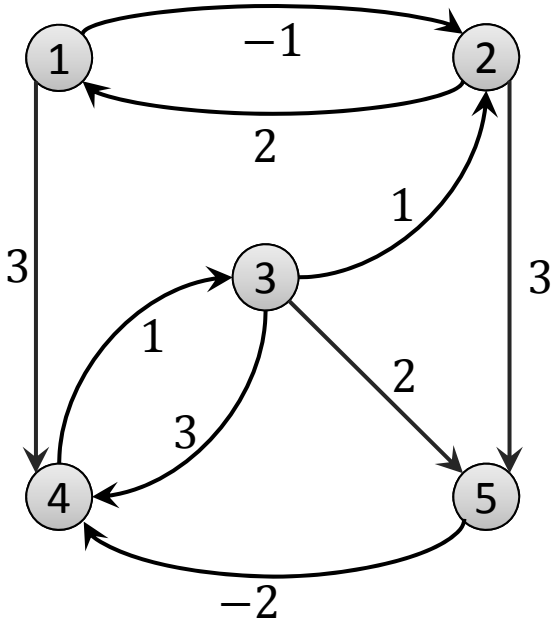
- Bellman-Ford Algorithmus mit allen Knoten:

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

  – Problem: langsam...

Distanzen zw. allen Knotenpaaren

- Wir beschränken uns zur Einfachheit auf Distanzen
- Anstatt mit Adjazenzlisten werden wir dieses Mal mit der Adjazenzmatrix arbeiten
- Oder etwas genauer, mit einer Distanzmatrix
- **Initialisierung:**



$$W = \begin{pmatrix} & 1 & 2 & 3 & 4 & 5 \\ 1 & 0 & -1 & \infty & 3 & \infty \\ 2 & 2 & 0 & \infty & \infty & 3 \\ 3 & \infty & 1 & 0 & 3 & 2 \\ 4 & \infty & \infty & 1 & 0 & \infty \\ 5 & \infty & \infty & \infty & -2 & 0 \end{pmatrix}$$

$W(i,j)$

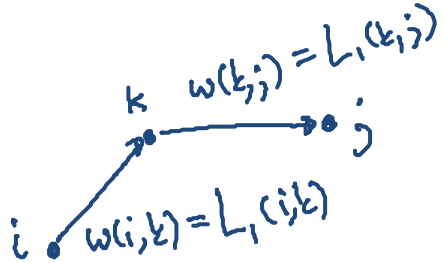
Distanzmatrix

Nach Initialisierung:

- Matrix $W = L_1$: Distanzen, falls man nur Pfade bestehend aus ≤ 1 Kante verwenden darf

Pfade aus ≤ 2 Kanten?

- Ziel: Matrix L_2: Distanzen durch Pfade aus ≤ 2 Kanten



$$L_2(i,j) \leq L_1(i,k) + L_1(k,j)$$

$$\Downarrow$$

$$L_2(i,j) = \min_{k \in \{1,\dots,n\}} \{L_1(i,k) + L_1(k,j)\}$$

(Pfade der Länge) bekommen wir bei $k = i$ / $k = j$

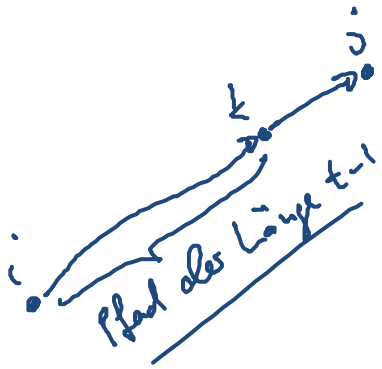
Distanzmatrix

Pfade aus $\leq t$ Kanten?

- Matrix L_t: Distanzen, falls nur Pfade aus $\leq t$ Kanten benutzt werden dürfen

Rekursive Berechnung:

Pfad der Länge t:



$$L_t(i,j) \leq L_{t-1}(i,k) + L_1(k,j)$$

$$L_t(i,j) = \min_{k \in \{1,\dots,n\}} \{L_{t-1}(i,k) + L_1(k,j)\}$$

beste Pfad Länge $< t$ hat, dann $L_t(i,j) = L_{t-1}(i,j)$
(bekommen wir für $k=j$)

Distanzmatrix und Matrixmultiplikation

Berechnung von L_t (aus L_{t-1} und W):

$$L_t(i,j) = \min_{1 \le k \le n} \{ L_{t-1}(i,k) + L_1(k,j) \}$$

$$(L_{t-1} \odot W)(i,j)$$

Matrixmultiplikation von L_{t-1} und W: $L' = L_{t-1} \cdot W$ (Matrixprodukt)

$$L'(i,j) = \sum_{k=1}^{n} L_{t-1}(i,k) \cdot L_1(k,j)$$

$i \to (L_{t-1}\text{-Zeile}) \cdot (W\text{-Spalte } j)$

Definition: $L_t = L_{t-1} \odot W$

- Matrixmultiplikation in der sogenannten Min-Plus-Algebra

Distanzmatrix berechnen

UNI FREIBURG

Algorithmus zum Berechnen der Distanzmatrix

$L_1 := W$
for $t := 2$ **to** $n-1$ **do**
    $L_t := L_{t-1} \odot W$

Laufzeit: $O(n^4)$

oder ausgeschrieben...

$L_1 := W$
for $t := 2$ **to** $n-1$ **do**
    for $i := 1$ **to** n **do**
        for $j := 1$ **to** n **do**
            $L_t(i,j) := L_{t-1}(i,j)$
            for $k := 1$ **to** n **do**
                if $L_{t-1}(i,k) + W(k,j) < L_t(i,j)$ **then**
                $L_t(i,k) := L_{t-1}(i,k) + W(k,j)$

$\forall i,j$

"Skalarprod."