Informatik II - SS 2016

(Algorithmen & Datenstrukturen)

Vorlesung 18 (24.6.2016)

Graphenalgorithmen VI
(Kürzeste Wege II)

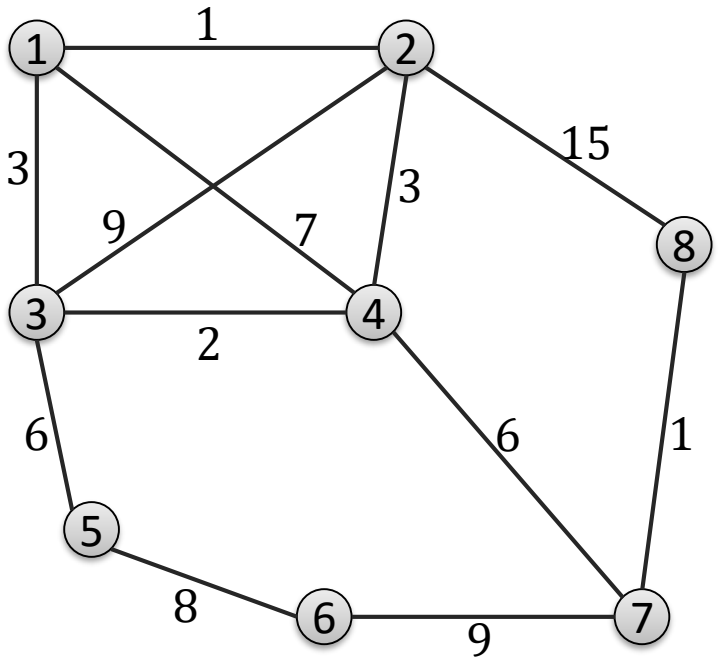
UNI FREIBURG

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)



Dijkstas Algorithmus: Laufzeit

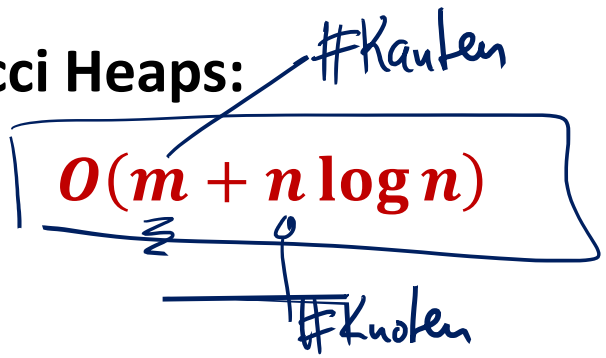
- Algorithmus-Implementierung ist fast identisch, wie diejenige von Prims MST Algorithmus
- **Anzahl Heap-Operationen:**

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

- **Laufzeit mit binären Heaps:**

$$O(m \log n)$$

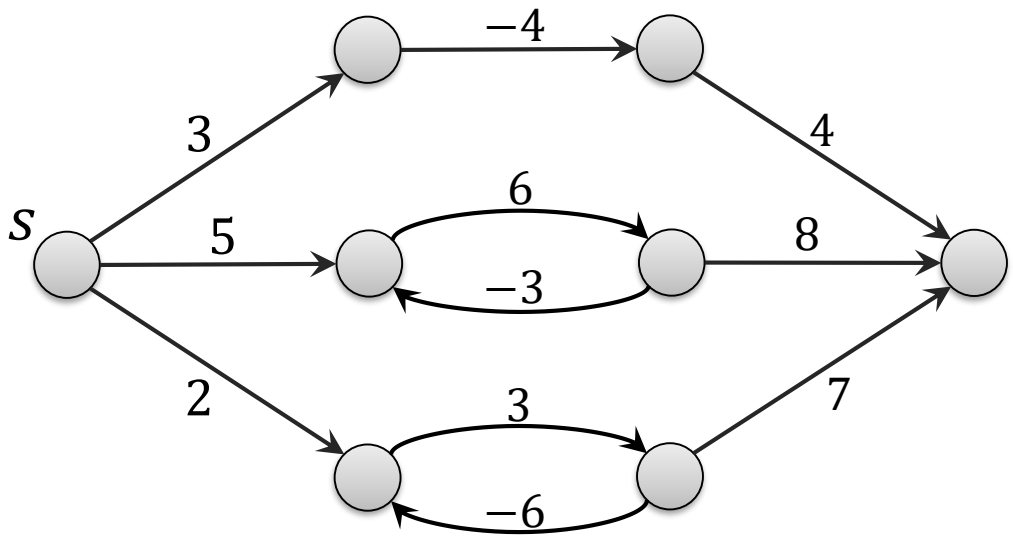
- **Laufzeit mit Fibonacci Heaps:**



Negative Kantengewichte

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

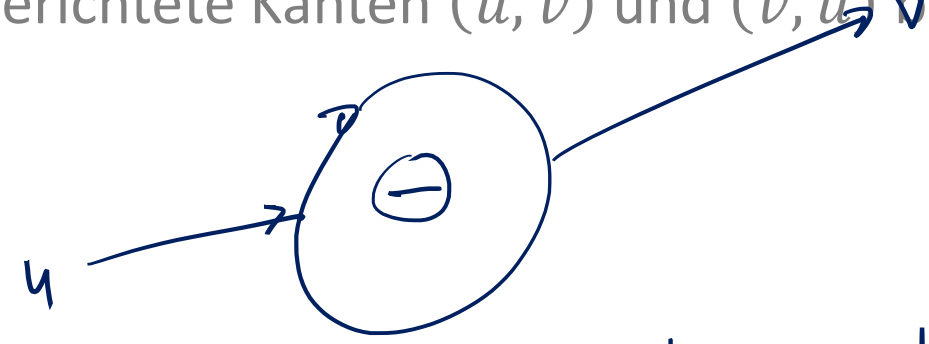
Beispiel



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

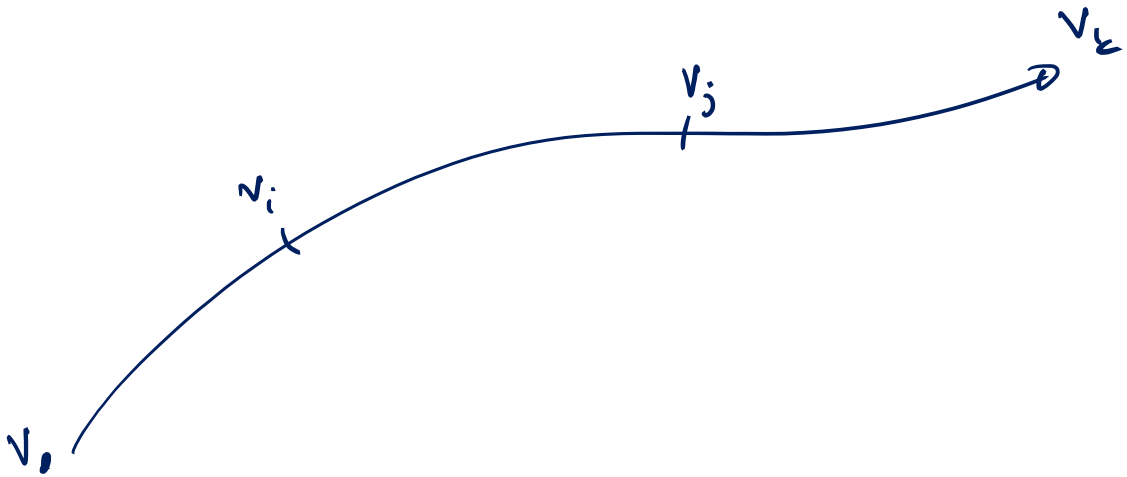


falls es keinen solchen neg. Kreis gibt, können wir uns auf einfache Pfade zw. u und v beschränken

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...**



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.

Bellman-Ford

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

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

neg. Kreise: von s erreichbar

$d_G^{(n-1)}(s,v)$: können uns auf einfache Pfade beschränken

$\Rightarrow$ nach $n-1$ Wiederholungen

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

Negative Kreise erkennen

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

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

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
```

Falls es kürzeste Pfade von s nach v für alle v gibt, haben wir diese gefunden

Negative Kreise erkennen

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

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

Widerspruchsbeweis

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

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

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

$$= \qquad\qquad < 0$$

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 $v \neq s : \alpha(v) = \text{NULL}$

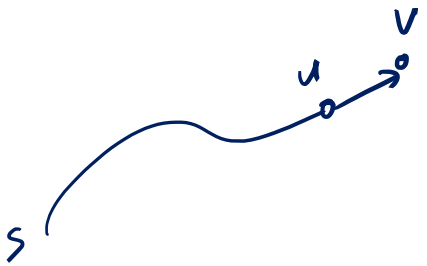
In jedem Schleifendurchlauf:

...

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

Laufzeit

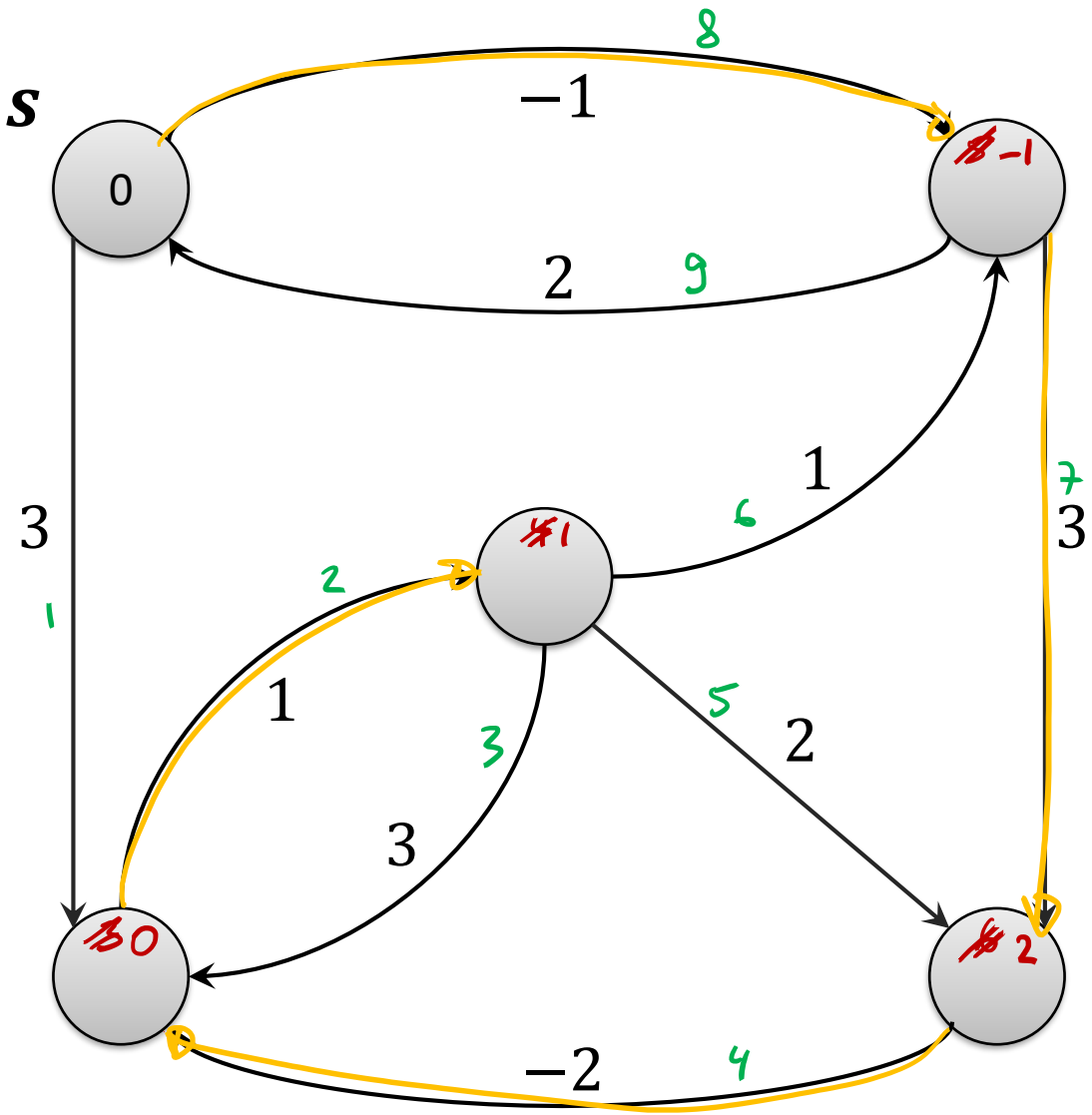
for $i = 1$ to $n-1$
  for all $(u,v) \in E$
    ⋮ } $O(1)$

$O(n \cdot m)$

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

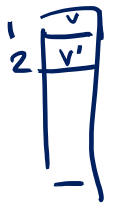


Routing-Pfade in Netzwerken

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

- all pairs shortest paths problem

Berechne single-source shortest paths für alle Knoten

- Dijkstras Algorithmus mit allen Knoten:

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

  – Problem: funktioniert nur bei nichtnegativen Kantengewichten

- Bellman-Ford Algorithmus mit allen Knoten:

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

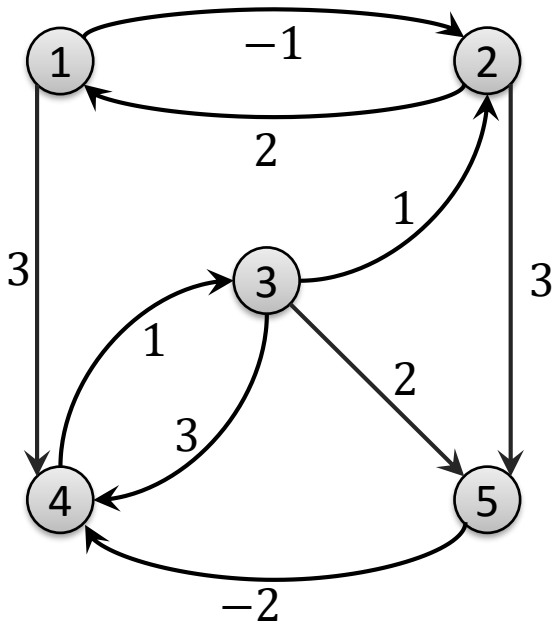
  – Problem: langsam...

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

$\delta(u,v)$

- **Initialisierung:**



W

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 0 & -1 & \infty & 3 & \infty \\ 2 & 0 & \infty & \infty & 3 \\ \infty & 1 & 0 & 3 & 2 \\ \infty & \infty & 1 & 0 & \infty \\ \infty & \infty & \infty & -2 & 0 \end{pmatrix} \end{array}$$

Zeile i, Spalte j : $w(i,j)$

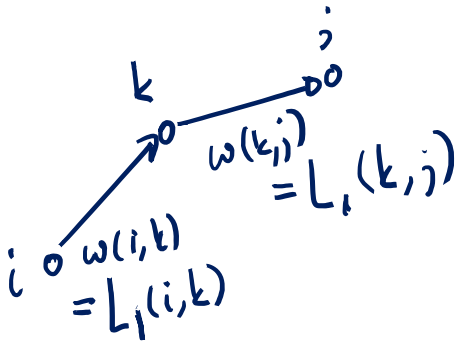
Distanzmatrix

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



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

$\Downarrow$ gilt für alle k

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

Pfade der Länge (# Kanten) 1 bekommen wir mit $k=i$ / $k=j$

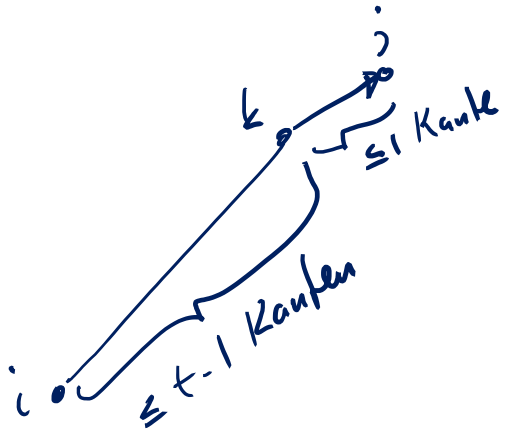
Distanzmatrix

Pfade aus $\leq t$ Kanten?

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

Rekursive Berechnung:

Pfad der Länge $\leq t$:



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

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

falls 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): $\tilde{} = L_1$

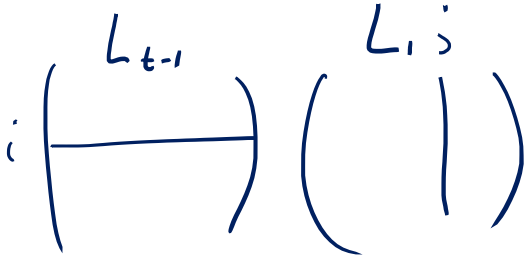
$$L_t(i,j) = \min_{1 \le k \le n} \{ L_{t-1}(i,k) + L_1(k,j) \}$$

$$(L_{t-1} \odot L_1)(i,j)$$

Matrixmultiplikation von L_{t-1} und W:

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

$L' = L_{t-1} \cdot L_1$



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

- Matrixmultiplikation in der sogenannten Min-Plus-Algebra

Distanzmatrix berechnen

Algorithmus zum Berechnen der Distanzmatrix

$L_1 := W$

for $t := 2$ **to** $n-1$ **do**

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

$L_{t-2} \odot L_2$

$L_{t/2} \odot L_{t/2}$

Laufzeit: $O(n^4)$

oder ausgeschrieben...

$L_1 := W$

for $t := 2$ **to** $n-1$ **do**

$\quad$ **for** $i := 1$ **to** n **do**

$\quad\quad$ **for** $j := 1$ **to** n **do**

$\quad\quad\quad L_t(i,j) := L_{t-1}(i,j)$

$\quad\quad\quad$ **for** $k := 1$ **to** n **do**

$\quad\quad\quad\quad$ **if** $L_{t-1}(i,k) + W(k,j) < L_t(i,j)$ **then**

$\quad\quad\quad\quad\quad L_t(i,k) := L_{t-1}(i,k) + W(k,j)$

Distanzmatrix schneller berechnen

- Matrixmultiplikation in der Min-Plus-Algebra hat die gleichen Grundeigenschaften, wie die normale Matrixmultiplikation
- Insbesondere erfüllt sie das ~~Distributivgesetz:~~ Assoziativgesetz

$$(A \odot B) \odot C = A \odot (B \odot C)$$

- Daher gilt $L_{x+y} = L_x \odot L_y$

$$L_{x+y} = (\cdots(((L_1 \odot L_1) \odot L_1) \odot L_1) \cdots) \quad \Big| \quad (\underbrace{L_1 \odot \cdots \odot L_1}_{x \text{ mal}}) \odot (\underbrace{L_1 \odot \cdots \odot L_1}_{y \text{ mal}})$$

- Und damit auch $L_{2t} = L_t \odot L_t$

opt. Pfade mit $\leq 2t$ Kanten bestehen aus 2 opt. Pfaden mit höchstens t Kanten

Distanzmatrix schneller berechnen

Algorithmus zum Berechnen der Distanzmatrix

$L := W$

for $t := 1$ **to** $\lceil \log_2 n \rceil$ **do**

$L' := L \odot L$

$L := L'$

$L_1\ L_2\ L_4\ L_8\ \dots$

$\odot$ → $O(n^3)$

- Am Schluss gilt $L = L_{2^{\lceil \log_2 n \rceil}} = L_{n-1}$ $t \geq n-1$ $L_t = L_{n-1}$

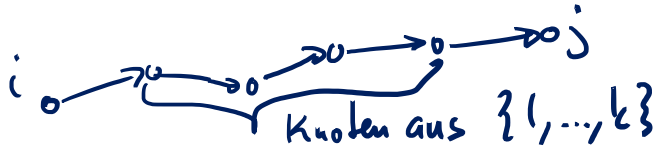
- Laufzeit: $O(n^3 \cdot \log n)$ Bellman-Ford: $O(m \cdot n) \in O(n^3)$

Distanzen noch schneller berechnen?

- Ein anderer Ansatz, die Distanzen rekursiv zu verstehen…
- Annahme: Knoten sind von 1 bis n nummeriert

$d_n(i,j) = d_G(i,j)$

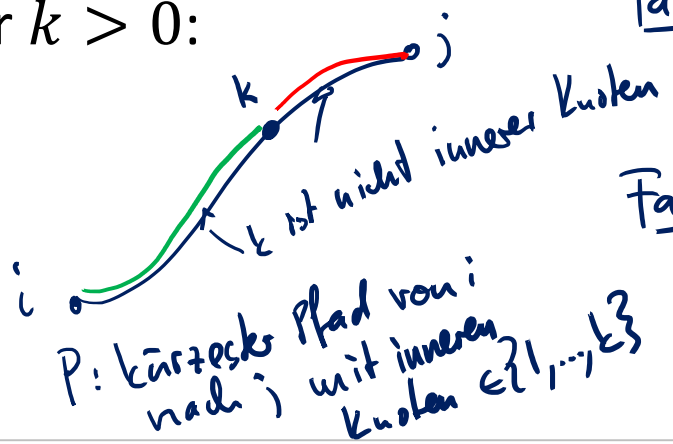
Definition $d_k(i,j)$



- Länge des kürzesten Pfades von i nach j, so dass als innere Knoten nur die Knoten $1, \dots, k$ verwendet werden.

Rekursive Definition von $d_k(i,j)$

- $d_0(i,i) = 0,\ d_0(i,j) = w(i,j)$ falls $(i,j) \in E,\ d_0(i,j) = \infty$ sonst
- Für $k > 0$:



Fall 1: k ist nicht innerer Knoten von P

$d_k(i,j) = d_{k-1}(i,j)$ ←

Fall 2: k ist innerer Knoten von P

$d_k(i,j) = d_{k-1}(i,k) + d_{k-1}(k,j)$ ←

Distanzen noch schneller berechnen?

Definition $d_k(i,j)$

- Länge des kürzesten Pfades von i nach j, so dass als innere Knoten nur die Knoten $1, \dots, k$ verwendet warden.

Rekursive Definition von $d_k(i,j)$

- $d_0(i,i) = 0,\ d_0(i,j) = w(i,j)$ falls $(i,j) \in E,\ d_0(i,j) = \infty$ sonst
- Für $k > 0$:

$$d_k(i,j) := \min\{d_{k-1}(i,j), d_{k-1}(i,k) + d_{k-1}(k,j)\}$$

falls k nicht inneres Knoten von P ist

falls k inneres Knoten von P ist

Floyd-Warshall Algorithmus

// Initialization

for $i \coloneqq 1$ to n do
    for $j \coloneqq 1$ to n do
        if $(i,j) \in E$ then $d_0(i,j) \coloneqq w(i,j)$ else $d_0(i,j) \coloneqq \infty$
    $d_0(i,i) \coloneqq 0$

// Main Loop

for $k \coloneqq 1$ to n do
    for $i \coloneqq 1$ to n do
        for $j \coloneqq 1$ to n do
            $d_k(i,j) \coloneqq \min\{d_{k-1}(i,j), d_{k-1}(i,k) + d_{k-1}(k,j)\}$

- Korrektheit folgt, da $\boldsymbol{d_G(i,j) = d_n(i,j)}$
- Laufzeit: $\boldsymbol{O(n^3)}$

Schnellere Algorithmen

bestes Matrix-Mult.-Alg. zur Zeit hat Laufzeit $O(n^{2.38})$

- Es gibt schnellere Algorithm zum Multiplizieren von Matrizen
  - Die Techniken können zum Teil verwendet werden
  - Allerdings nicht direkt
  - Und insbesondere bei gewichteten Graphen nicht ohne zusätzliche Kosten

- Dünn besetzte Graphen
  - Johnsons Algorithmus (Laufzeit $O(n^2 \log n + mn)$)
  - Idee: Falls die Kantengewichte nichtnegativ sind, kann man einfach n Mal Dijkstra ausführen
  - Falls der Graph negative Kantengewichte hat, werden zuerst nichtnegative Gewichte berechnet, welche die gleichen kürzesten Pfade ergeben
  - Das kann man in Zeit $O(mn)$ tun

n × Dijkstra

Transitive Hülle

- Gegeben ein gerichteter Graph $G = (V, E)$
- Die transitive Hülle von G ist ein Graph $H = (V, E')$ für welchen gilt:

$$(u, v) \in E' \Leftrightarrow \text{gerichteter Pfad von } u \text{ nach } v \text{ in } G$$

- Kann genauso mit Floyd-Warshall in Zeit $O(n^3)$ berechnet warden
  - Man kann etwas optimieren (da man nur eine 1/0-Antwort benötigt)

- Hier kann man die Matrix-Multiplikationstechniken anwenden
  - Beste bekannte Komplexität: $O(n^{2.38})$