Informatik II - SS 2016

(Algorithmen & Datenstrukturen)

Vorlesung 18 (24.6.2016)

Graphenalgorithmen VI
(Kürzeste Wege II)

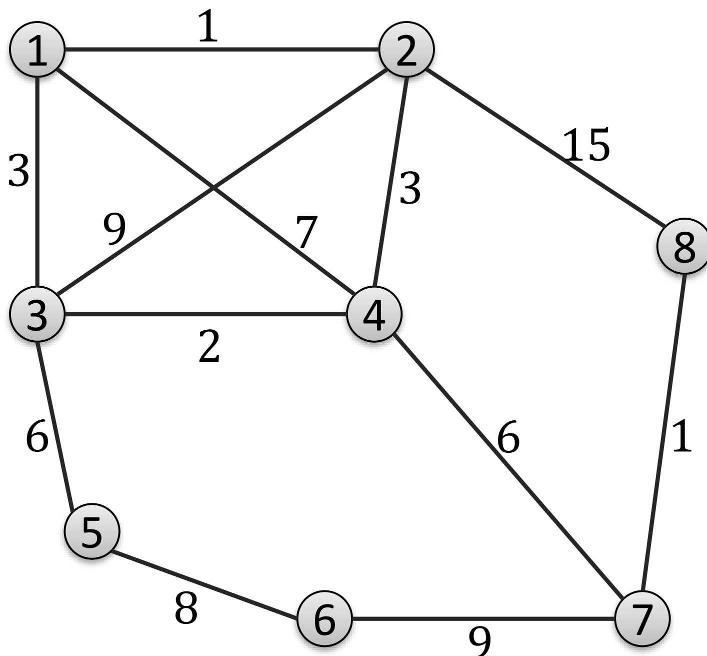
Fabian Kuhn
Algorithmen und Komplexität

Kürzeste Wege

UNI FREIBURG

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:**

$$\boldsymbol{O(m \log n)}$$

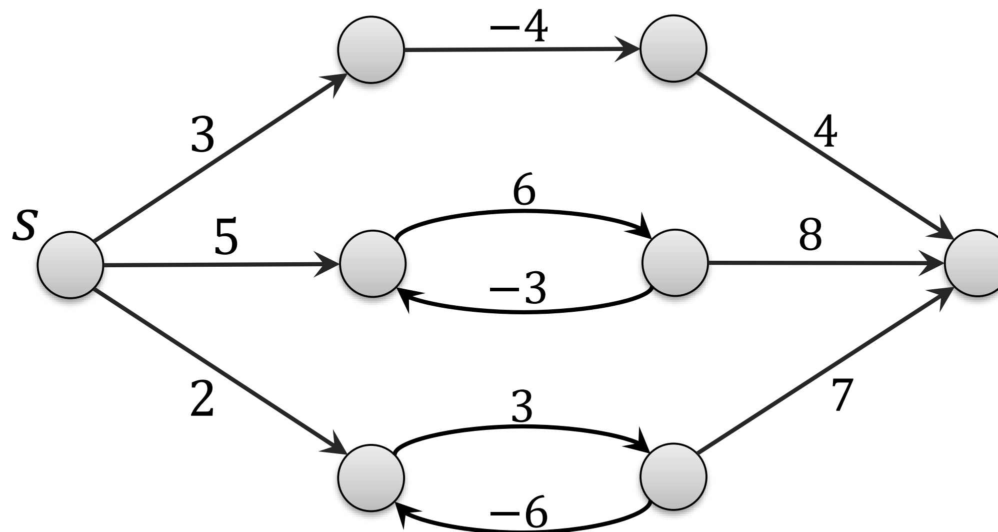
- **Laufzeit mit Fibonacci Heaps:**

$$\boldsymbol{O(m + n \log n)}$$

Negative Kantengewichte

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

Beispiel



Negative Kantengewichte

UNI FREIBURG

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

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

UNI FREIBURG

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

Negative Kreise erkennen

UNI FREIBURG

- 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
```

Negative Kreise erkennen

UNI FREIBURG

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

Kürzeste Pfade

UNI FREIBURG

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**

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

$\alpha(v) \coloneqq u$

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

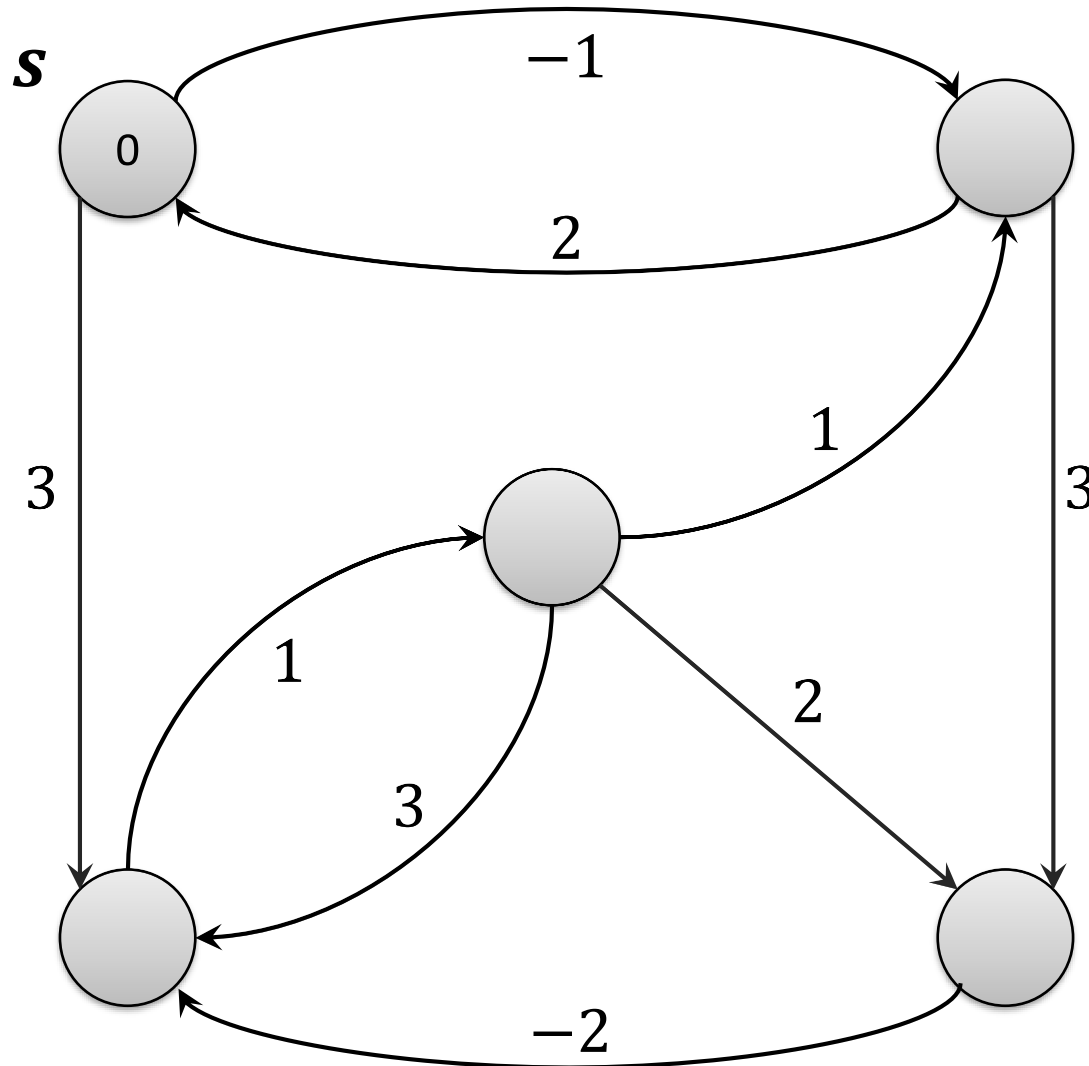
Bellman-Ford Alg.: Zusammenfassung

UNI FREIBURG

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.

- 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: $n \cdot O(\text{Laufzeit Dijkstra}) \in O(mn + n^2 \log n)$
  - Problem: funktioniert nur bei nichtnegativen Kantengewichten

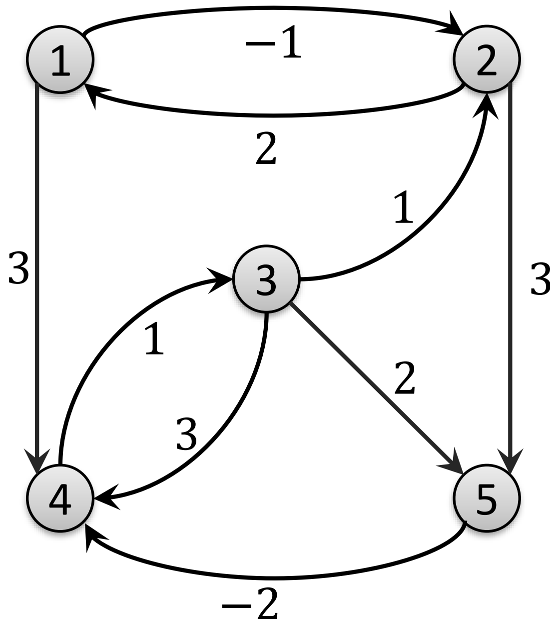
- Bellman-Ford Algorithmus mit allen Knoten:

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

Distanzen zw. allen Knotenpaaren

UNI FREIBURG

- 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:**



Distanzmatrix

UNI FREIBURG

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

Distanzmatrix

UNI FREIBURG

Pfade aus $\leq t$ Kanten?

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

Rekursive Berechnung:

Distanzmatrix und Matrixmultiplikation

UNI FREIBURG

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

Matrixmultiplikation von L_{t-1} und W:

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**

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

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)
```

Distanzmatrix schneller berechnen

UNI FREIBURG

- Matrixmultiplikation in der Min-Plus-Algebra hat die gleichen Grundeigenschaften, wie die normale Matrixmultiplikation

- Insbesondere erfüllt sie das Distributivgesetz:

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

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

Distanzmatrix schneller berechnen

Algorithmus zum Berechnen der Distanzmatrix

$L \coloneqq W$

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

$\quad L' \coloneqq L \odot L$

$\quad L \coloneqq L'$

- Am Schluss gilt $L = L_{2^{\lceil \log_2 n \rceil}} = L_n$

- Laufzeit:

Distanzen noch schneller berechnen?

UNI FREIBURG

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

Definition $\boldsymbol{d_k(i,j)}$

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

Rekursive Definition von $\boldsymbol{d_k(i,j)}$

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

Distanzen noch schneller berechnen?

UNI FREIBURG

Definition $\boldsymbol{d_k(i,j)}$

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

Rekursive Definition von $\boldsymbol{d_k(i,j)}$

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

- Für $k > 0$:

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

Floyd-Warshall Algorithmus

// Initialization

```
for i := 1 to n do
    for j := 1 to n do
        if (i,j) ∈ E then d_0(i,j) := w(i,j) else d_0(i,j) := ∞
    d_0(i,i) := 0
```

// Main Loop

```
for k := 1 to n do
    for i := 1 to n do
        for j := 1 to n do
            d_k(i,j) := 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

UNI FREIBURG

- 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

Transitive Hülle

UNI FREIBURG

- 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: $O(n^{2.38})$