Informatik II - SS 2018

(Algorithmen & Datenstrukturen)

Vorlesung 17a (20.6.2018)

Graphenalgorithmen VI

UNI FREIBURG

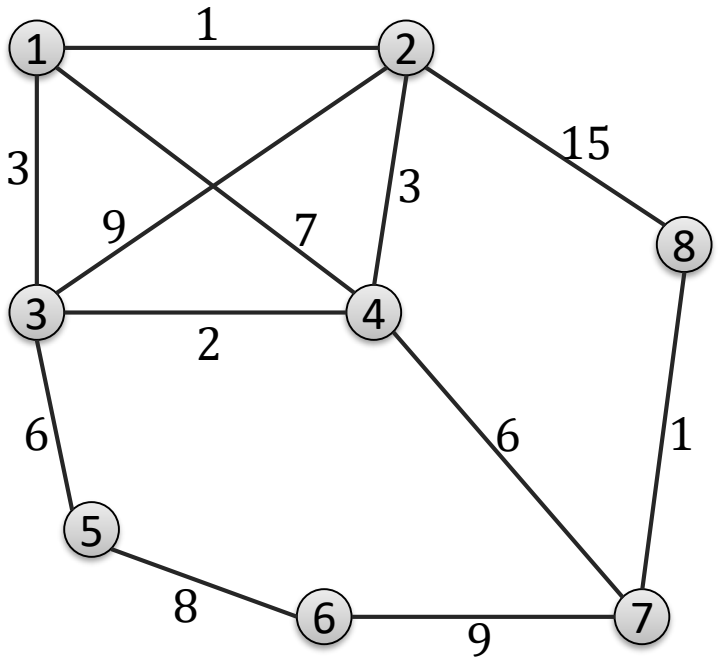
Fabian Kuhn

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)



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.

Negative Kreise erkennen

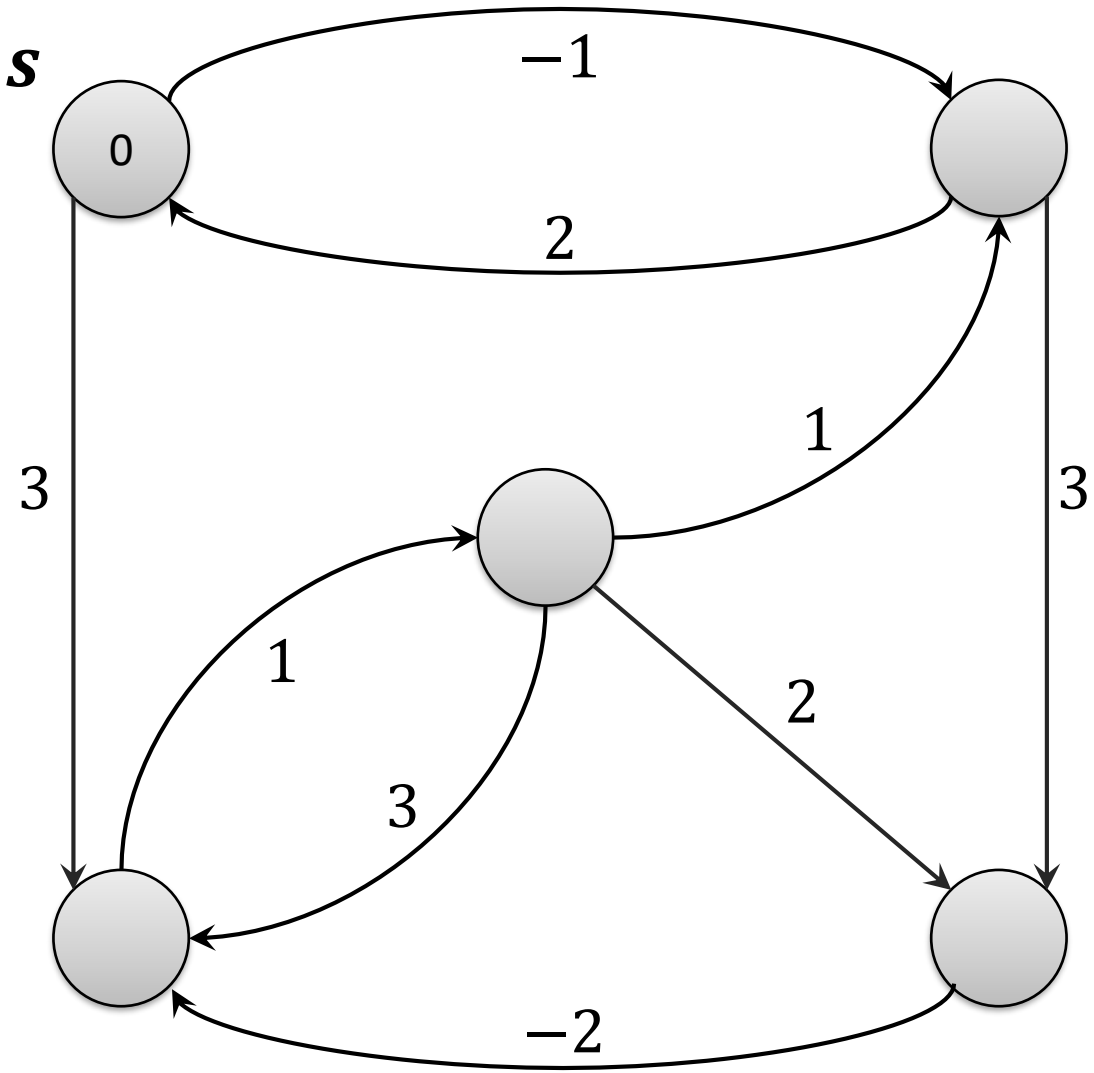
- Wir werden sehen: Falls es einen (von s erreichbaren) negativen Kreis hat, dann gibt es für irgendeine Kante eine Verbesserung.

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

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
```

Bellman-Ford Algorithmus: Beispiel



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

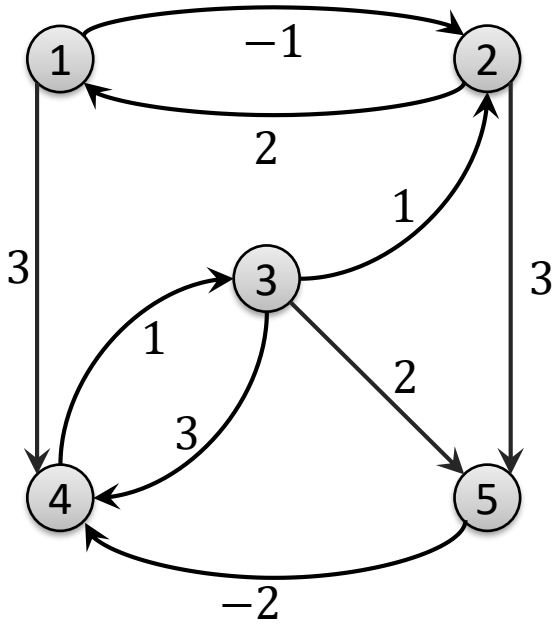
- Bellman-Ford Algorithmus mit allen Knoten:

  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:**



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

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

- Matrixmultiplikation in der Min-Plus-Algebra hat die gleichen Grundeigenschaften, wie die normale Matrixmultiplikation
- Insbesondere erfüllt sie das Assoziativgesetz:
- Daher gilt $L_{x+y} = L_x \odot L_y$
- Und damit auch $L_{2t} = L_t \odot L_t$

Distanzmatrix schneller berechnen

UNI FREIBURG

Algorithmus zum Berechnen der Distanzmatrix

$L := W$

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

$L' := L \odot L$

$L := L'$

- 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 $d_k(i,j)$

- 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 $k > 0$:

Distanzen noch schneller berechnen?

UNI FREIBURG

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 $k > 0$:

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

Floyd-Warshall Algorithmus

// Initialization

for $i \coloneqq 1$ 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

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

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