Algorithmen & Datenstrukturen

Herbst 2018

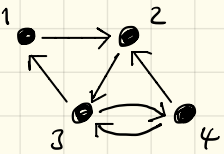
Vorlesung 12

Graphen: Kurze Wiederholung

Graphen $G = (V, E)$, heute meist gerichtet

$|V| = n$, $|E| = m$, $m \leq n^2$

$V = \{1, 2, \ldots, n\}$ (Knoten nummeriert)



G

Datenstrukturen für Graphen?

1.) Adjazenzmatrix $A_G = A = [a_{ij}]_{1 \leq i,j \leq n}$

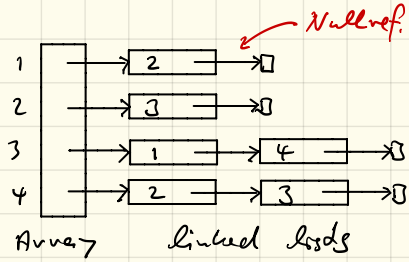
$$a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{sonst} \end{cases}$$

Platz $O(n^2)$

$$A_G = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

2.) Adjazenzliste (viele Varianten möglich)

zu G oben:



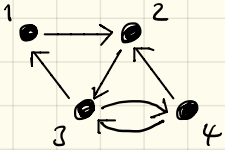
Platz $O(n+m)$

zu jedem Knoten Liste der Nachbarn (Reihenfolge egal)

Varianten: linked list statt Array, doubly-linked list...

Operationen	Adj. matrix	Adj. liste
alle Nachfolger von $v \in V$	$O(n)$	$O(\deg^+(v))$
$(i,j) \in E$?	$O(1)$	$O(\deg^+(v))$
Kante einfügen	$O(1)$	$O(1)$
? Knoten ohne Nachfolger?	$O(n^2)$	$O(n)$
etc.		

Graph $\leftrightarrow$ Matrix ist interessant



$G \longleftrightarrow A_G = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$

$A_G^2 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$, $A_G^3 = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 2 & 1 & 0 \\ 1 & 0 & 2 & 1 \\ 1 & 2 & 1 & 1 \end{pmatrix}$

Bedeutung?

$A_G^2 = B = [s_{ij}]$ $\quad s_{ij} = \sum_{k=1}^{n} \underbrace{a_{ik} a_{kj}}_{=1 \Leftrightarrow a_{ik}=1 \text{ und } a_{kj}=1}$

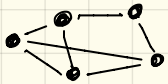
$\Leftrightarrow$ 

zählt Wege der Länge 2!

Satz: Das Element (i,j) in A_G^k ist die Anzahl der Wege $i \to j$ der Länge k.

Beweis: Induktion nach k.

Anwendung 1: Anzahl der Dreiecke in ungerichteten Graph ohne Schleifen: $\text{Spur}(A_G^3)/6$



(2 Dreiecke)

$=$ Summe Diagonalelemente

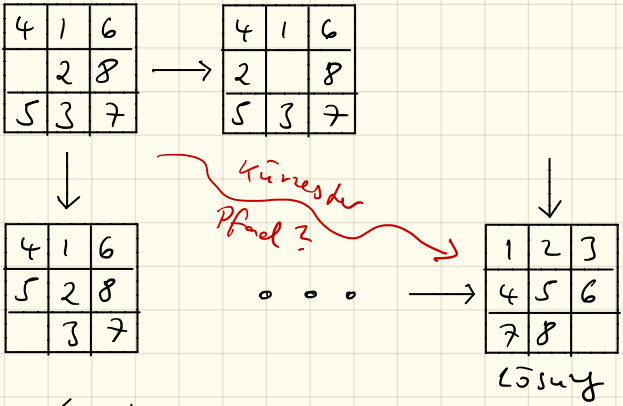
Anwendung 2: Kürzester Weg von i nach j?
Potenziere A_G bis Eintrag $(i,j) \neq 0$
Potenzieren bis $n-1$ reicht $= O(n)$ Matrixmults
$= O(n^4)$

alle kürzesten Wege (all-to-all): auch $O(n^4)$

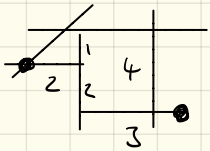
Kürzeste Wege in Graphen

Schiebepuzzle:

Knoten: Zustände
Kante wenn überführbar



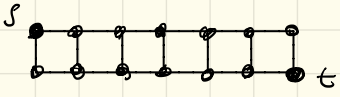
Google Maps:



Distanzgraph
(Kanten haben Gewichte)
$G = (V, E, c)$
$c: E \to \mathbb{R}^+$

SBB: Asfaltstraßen berücksichtigen, ansonsten gleiches Problem

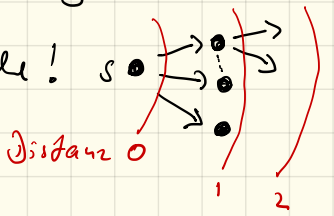
Es kann exponentiell viele Wege geben:



(⇒ aufwendiger zu lösen)

1.) $G = (V, E)$ ohne Gewichte

Kürzester Weg von s nach t?

Breitensuche! 

teilt Graph in Levels ein ⇒ wenn man t findet ist man fertig

Kosten $O(n+m)$, gilt alle kürzesten Wege $s \to t$, $t \in V$ (one-to-all)

all-to-all? Breitensuche von jedem Knoten $O(n^2 + mn)$

2.) $G = (V, E, c)$, $c: E \to \mathbb{R}^+$ (Distanzgraph)

Weg von s nach t: $(s = v_0, \dots, v_k = t)$ (source, target)

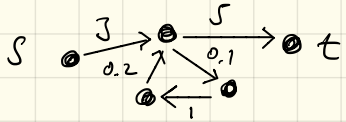
$(v_i, v_{i+1}) \in E$, $i = 0 \dots k-1$

Kosten: $\sum_{i=0}^{k-1} c((v_i, v_{i+1}))$

Bemerkung: Kann man physisch lösen. Baue Graph mit Schnüren, ziehe s und t so straff.

Existenz des kürzesten Wegs?

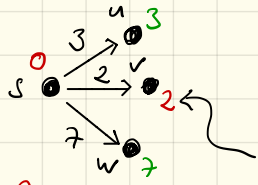
Es kann unendlich viele Wege geben



aber Zyklen machen den Weg nur länger ($c: E \to \mathbb{R}^+$)
$\Rightarrow$ kürzester Weg zyklenfrei $\Rightarrow \leq n$ Knoten
(davon gibt es nur endlich viele)

Kürzester Weg: Algorithmus

Anfang:

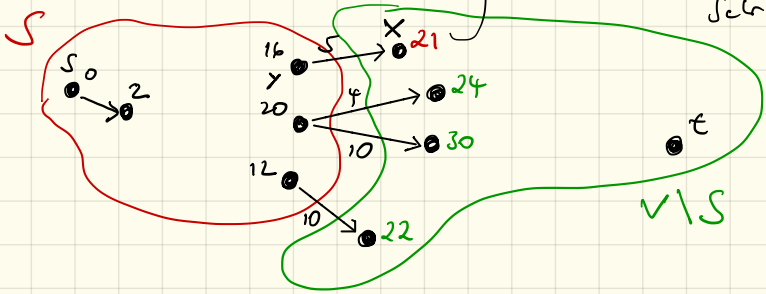


t

ist not weil kleinste obere Schranke

Minimale Distanz von s
Obere Schranke für Distanz von s

Erweiterung der Idee:



not weil kleinste obere Schranke

alle oberen Schranken $\geq$ alle d in S!

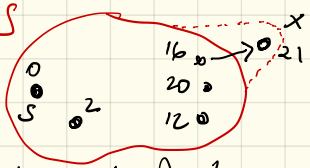
Invariante

- $S \subseteq V$: Knoten bei denen man min. Distanz weiß
- Induktion über $|S|$
- Wähle $x \in V \setminus S$ mit kleinster oberer Schranke und füge zu S
- Merke Vorgänger von x in Array: $a[x] = y$
(ermöglicht Auslesen des kürzesten Weges)

$S = \{s\}$, $d(s) = 0$
while $|S| < n$ do
  wähle (u, v), $u \in S$, $v \in V \setminus S$
    mit minimalem $d(u) + c(u, v)$
  $S = S \cup \{v\}$, $d(v) = d(u) + c(u, v)$

(berechnet alle kürzesten Wege und Distanzen von s nach x, $x \in V$: "one-to-all")

Korrektheit: Induktion über $|S|$. $|S| = 1$ ✓



Es kann keinen kürzeren Weg von s nach $V \setminus S$ geben.
$\Rightarrow$ Distanz ist minimal

Anm. kürzeste Distanzen bekannt

Sozusagen eine wellenförmige Breitensuche "eine Welle gleicher Distanzen wird über G geschoben"

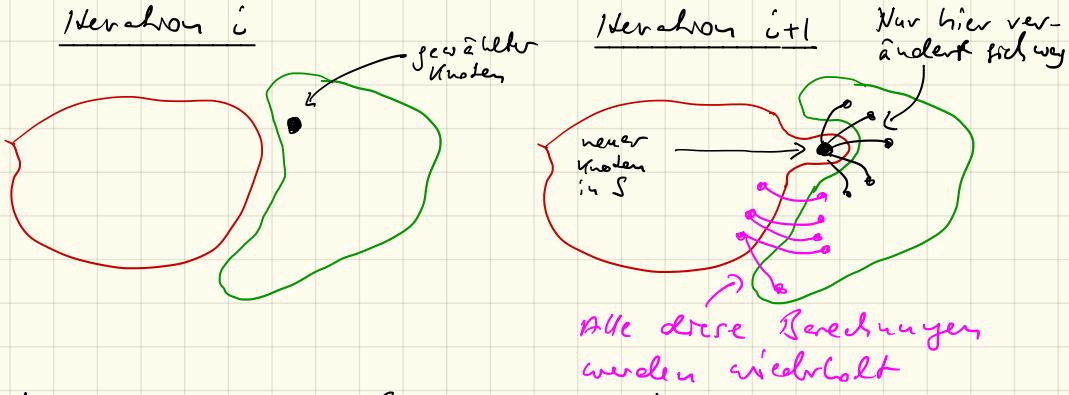
Präzisierung des Algorithmus

"wähle (u, v) ..." — trivial: scanne alle Pfeile durch, prüfe $u \in S$, $v \notin S$, merke min. oder scanne

Laufzeit: $O(n)$ je Iteration, also $O(mn)$ gesamt

"wähle" nach Dijkstra:

Beobachtung: redundante Berechnungen zwischen Iterationen

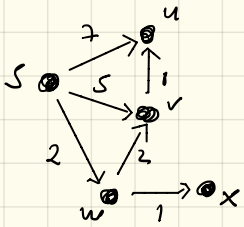


Idee:
- merke ösene Schranken für jeden Knoten (am Anfang: ∞)
- update ösene Schranken in jeder Iteration für Nachfolger von neuem Knoten in S

Algorithmus:

$d(s) = 0$
$d(v) = \infty$ für alle $v \in V \setminus \{s\}$
$S = \emptyset$
finde $v \in V \setminus S$ mit min $d(v)$, füge zu S
for each $(v,w) \in E$ // $\deg^+(v)$ viele
  if $w \in V \setminus S$ then
    $d(w) = \min(d(w), d(v) + c(v,w))$ // kürzester Weg nach w über v?

Beispiel:



init: $d(s) = 0$, $d(u) = d(v) = d(w) = d(x) = \infty$
$S = \emptyset$

1 Iteration 1: $S = \{s\}$
$d(u) = 7$
$d(v) = 5$
$d(w) = 2$ ← d(w) ist min

Distanz von s: 2 — 1 Iteration 2: $S = \{s, w\}$
$d(v) = 4$ ← // senke d(v)
$d(x) = 3$

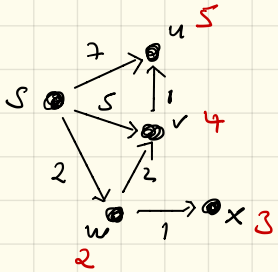
Distanz von s: 3 — 1 Iteration 3: $S = \{s, w, x\}$

Distanz von s: 4 — 1 Iteration 4: $S = \{s, w, x, v\}$
$d(u) = 5$

Distanz von s: 5 — 1 Iteration 5: $S = \{s, w, x, v, u\}$ fertig

wächst

Distanzen d nach Ende:



Analyse Algorithmus:

init	$O(n)$
n extract min	$O(n \log n)$ mit Heap
$\leq m$ decrease key	$O(m \log n)$
	$O((m+n) \log n)$

Heap: extract min $O(\log n)$
decrease key $O(\log n)$ wenn man weiß wo d.key steht → speichere Link in Adjazenzliste

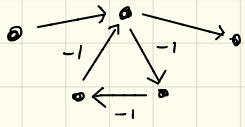
Geht noch besser mit Fibonacci-Heaps → $O(m + n \log n)$
(ohne Erklärung)

3.) $G = (V, E, c)$, $c : E \to \mathbb{R}$

Grundidee "relaxieren" geht immer noch

$d(v) = \min(d(v), d(u) + c(u,v))$
(relax Pfeil (u,v))

Zyklen:



negativer Zyklus
$\Leftrightarrow$ es gibt keinen kürzesten Weg

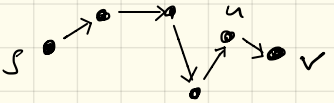
<u>Idee 1</u>: relaxiere bis sich nichts mehr ändert

Ford's Algorithmus:

```
init wie zuvor (d(s)=0, d(v)=∞ für v≠s)
repeat
    for all (u,v) ∈ E: relax(u,v)
until keine Änderung   kann niemals enden
```

es gibt kürzesten Weg $\Rightarrow$ dieser ist einfach (kein Zyklus)
$\Rightarrow$ hat $\leq n$ Knoten

Idee 2: DP



kürzester Weg $s \to v$

$\Rightarrow$ Teilweg $s \to u$ ist auch kürzest

Induktion über Pfeilzahl i in Weg:

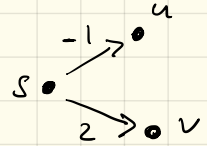
d_w^i = kürzeste Weglänge $s \to w$ mit $\leq i$ Pfeilen

$$d_w^i = \min\left(d_w^{i-1}, \min_{(v,w)\in E}\left(d_v^{i-1} + c(v,w)\right)\right)$$

Init: $d_s^0 = 0$, $d_v^0 = \infty$, $v \neq s$

Tableau		0	1		n-1
	s	0	0		
		∞			
Bellman's	u	∞	-1		
Algorithmus		⋮			
(erstes DP)	v	⋮	2		
		∞			

← mehr Pfeile nicht möglich wenn kürzester existiert



Laufzeit: $O(n^3)$ (n^2 Tableau + $O(n)$ je Eintrag)

Idee 3: Kombination: Bellman-Ford Algorithmus

```
for i = 0 .. n-1
    for (u,v) ∈ E
        relax (u,v)
```
} $O(mn)$

$O(m)$ { for $(u,v) \in E$
  teste ob relax(u,v) etwas ändert → ja keine Lsg
  → nein ✓

alle Algorithmen sind "one-to-all"

Zusammenfassung:

(V, E)	Breitensuche	$O(n+m)$
(V, E, c), $c: E \to \mathbb{R}^+$	Dijkstra	$O((n+m)\log n)$ $(O(m+n\log n))$
(V, E, c), $c: E \to \mathbb{R}$	Bellman-Ford	$O(nm)$