Algorithmen & Datenstrukturen

Herbst 2022

Vorlesung 14

All pairs shortest path: Floyd-Warshall, Johnson

Kürzeste Wege, one-to-all (single source), Überblick

G gerichteter oder ungerichteter Graph, $|V|=n$, $|E|=m$
ich schreibe $G=(V,E,c)$ wenn c Gewichtsfunktion

Graph	Algorithmus	Laufzeit
$G=(V,E)$	Breitensuche	$O(m+n)$
$G=(V,E,c)$, $c: E \to \mathbb{R}^+$	Dijkstra	$O((m+n)\log n)$ $O(m+n\log n)$ mit Fibonacci-Heaps, war nicht in Vorlesung
$G=(V,E,c)$, $c: E \to \mathbb{R}$	Bellman-Ford	$O(mn)$

allgemeiner ↓

- nur one-to-one gibt es nicht da alle one-to-all berechnen
- wenn man weiß das G keine Zyklen hat gelt es in $O(m+n)$ (topologische Sortierung + DP, Vorlesung 12). Das kann man also immer zuerst prüfen:
  1.) DFS → Zyklen? + top. Sortierung
  2.) keine Zyklen → DP

Problem: Finde wichtigste Personen in einem sozialen Netzwerk.

"wichtig"? Ein Ansatz: Person liegt auf vielen kürzesten Wegen zwischen 2 Personen ("betweenness centrality")

Braucht all-to-all (all pairs) shortest path

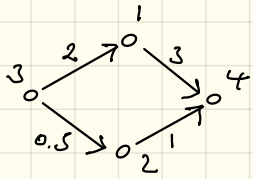
Graph	Algorithmus	Laufzeit
$G = (V, E)$	$n \times$ Breitensuche	$O(mn + n^2)$
$G = (V, E, c)$, $c: E \to \mathbb{R}^+$	$n \times$ Dijkstra	$O(mn + n^2 \log n)$
$G = (V, E, c)$, $c: E \to \mathbb{R}$	$n \times$ Bellman-Ford	$O(mn^2)$
	Floyd-Warshall	$O(n^3)$
	Johnson	$O(mn + n^2 \log n)$

allgemeiner ↓

Floyd-Warshall algorithm: all-to-all (all pairs) shortest path

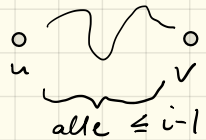
Idee: nummeriere Knoten $1..n$

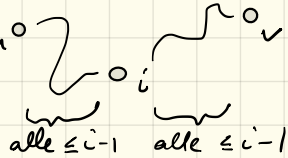
d_{uv}^{i} = Kosten günstigster Weg $u \rightsquigarrow v$
mit Zwischenknoten nummern $\leq i$
(nicht: mit i vielen Zwischenknoten)



$d'_{3,4} = 5$

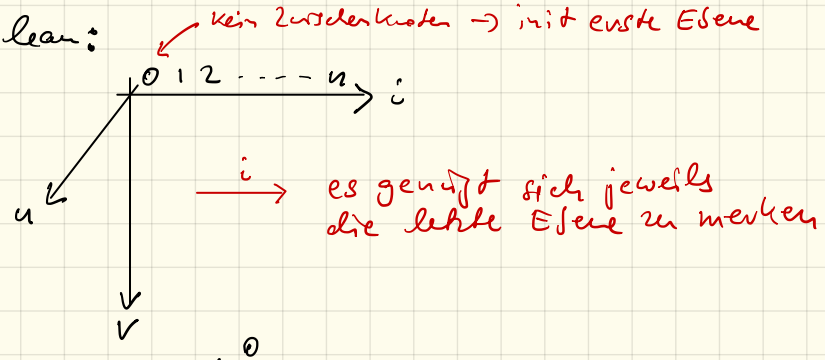
DP! $d_{uv}^{i} = ?$ 2 Möglichkeiten:

1.) 

2.) 

$\Rightarrow$ Rekurrenz: $d_{uv}^{i} = \min\left(d_{uv}^{i-1},\ d_{ui}^{i-1} + d_{iv}^{i-1}\right)$

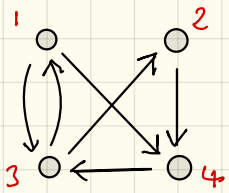
3D-Tableau:



Initialisierung:

$d_{uu}^{0} = 0$
$d_{uv}^{0} = c(u,v)$ falls $(u,v) \in E$
$d_{uv}^{0} = \infty$ sonst

Graph:



Init: $d^0_{uu} = 0$

$d^0_{uv} = 1$ (u,v) Kante

$d^0_{uv} = \infty$ sonst

d^i_{uv} = kürzester Weg $u \rightsquigarrow v$ mit Zwischenknoten $\leq i$

$$d^i_{uv} = \min(d^{i-1}_{uv}, d^{i-1}_{ui} + d^{i-1}_{iv})$$

$i=0$:

u \ v	1	2	3	4
1	0	-	1	1
2	-	0	-	1
3	1	1	0	-
4	-	-	1	0

$i=1$: $\min(d_{12}, d_{11}+d_{12})$

	1	2	3	4
1	0	-	1	1
2	-	0	-	1
3	1	1	0	2
4	-	-	1	0

$d_{31}+d_{14}$

$i=2$:

	1	2	3	4
1	0	-	1	1
2	-	0	-	1
3	1	1	0	2
4	-	-	1	0

$i=3$: $d_{13}+d_{32}$

	1	2	3	4
1	0	2	1	1
2	-	0	-	1
3	1	1	0	2
4	2	2	1	0

$d_{43}+d_{32}$

$i=4$: $d_{24}+d_{41}$

	1	2	3	4
1	0	2	1	1
2	3	0	2	1
3	1	1	0	2
4	2	2	1	0

Beachte: hier hatten wir keine Kantengewichte

FW(G)

// initialisiere Tableau

for $u \in V$: $d^0_{uu} = 0$

for $(u,v) \in E$: $d^0_{uv} = c(u,v)$; else $d^0_{uv} = \infty$

// DP

for $i = 1 \dots n$

for $u = 1 \dots n$

for $v = 1 \dots n$

$d^i_{uv} = \min(d^{i-1}_{uv}, d^{i-1}_{ui} + d^{i-1}_{iv})$

wenn inplace lässt man den Index weg

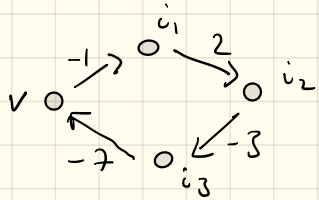
return d

Funktioniert wenn keine negativen Zyklen

Test: v ist in negativem Zyklus $\Leftrightarrow d^n_{vv} < 0$

wenn $i_1 < i_2 < i_3$

hat man $d^{i_3}(v,v) < 0$



Kürzeste Wege merken: $O(n^2)$ Extraplatz

(ähnlich Ford's Algorithmus)

Laufzeit: $O(n^3)$

Variante: Transitive Closure für Relationen

$G = (V, E)$ beschreibt Relation $\leq$ auf V:

$u \leq v \Leftrightarrow (u,v) \in E$

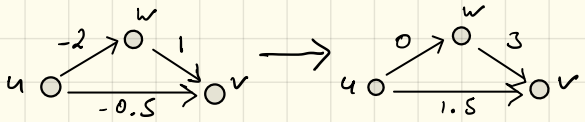
d^i_{uv}: v ist erreichbar von u mit Knoten $\leq i$

Rekurrenz: $d^i_{uv} = d^{i-1}_{uv} \vee (d^{i-1}_{ui} \wedge d^{i-1}_{iv})$

Johnson's Algorithmus

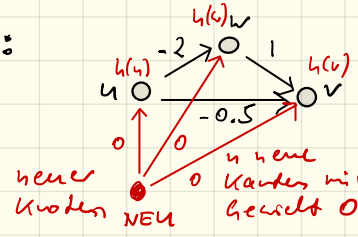
Idee:
- mache alle Kantengewichte positiv
- dann nx Dijkstra

Ansatz 1: Subtrahiere kürzestes Gewicht



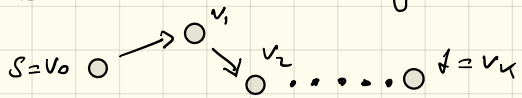
ändert den kürzesten Weg (hier: $u \rightsquigarrow v$) da Subtraktion von Anzahl der Pfade abhängt

Ansatz 2:



- merke zu jedem Knoten v eine "Höhe" $h(v)$
- neue Gewichte: $\hat{c}(u,v) = c(u,v) + h(u) - h(v)$
- Ziel: $\hat{c} : E \to \mathbb{R}^+$

Damit: 1.) Bleiben kürzeste Wege kürzest:



Länge vorher: $c(s \rightsquigarrow t) = \sum_{i=0}^{k-1} c(v_i, v_{i+1})$

Länge jetzt: $\hat{c}(s \rightsquigarrow t) = \sum_{i=0}^{k-1} \hat{c}(v_i, v_{i+1})$

$$= \sum_{i=0}^{k-1} \left(c(v_i, v_{i+1}) + h(v_i) - h(v_{i+1}) \right)$$

$$= c(s \rightsquigarrow t) + h(s) - h(t)$$

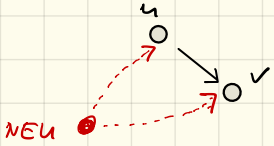
hängt nur von s und t ab

2.) Zyklenkosten bleiben: $\hat{c}(s \rightsquigarrow s) = c(s \rightsquigarrow s)$

Wie definieren wir h so daß $\hat{c}$ positiv?

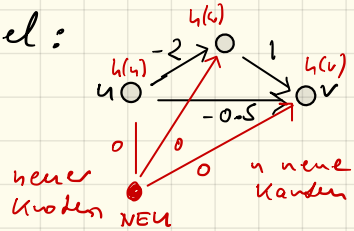
$h(u)$ = Länge kürzester Weg $NEU \rightsquigarrow u$ ($\Rightarrow$ alle $h(u) \leq 0$)

Denn dann für jede $(u,v) \in E$:



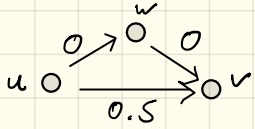
$$h(v) \leq h(u) + c(u,v)$$
$$\Leftrightarrow \quad c(u,v) + h(u) - h(v) \geq 0$$
$$\Leftrightarrow \quad \hat{c}(u,v) \geq 0$$

Im Beispiel:



$h(u) = 0$
$h(v) = -1$
$h(w) = -2$

$\Rightarrow$



Analyse:

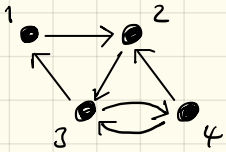
Neuer Knoten und Pfeile	$O(n)$
h-Werte: Bellman-Ford	$O(nm)$
n Mal Dijkstra	$O(nm + n^2 \log n)$
Insgesamt:	$O(nm + n^2 \log n)$

(besser als Floyd-Warshall für dünnbesetzte Graphen)

Floyd-Warshall nochmal:

$d_{uv} = \min(d_{uv}, d_{ui} + d_{iv})$ // all pairs shortest path
$d_{uv} = d_{uv} \vee (d_{ui} \wedge d_{iv})$ // transitive closure
$d_{uv} = d_{uv} + d_{ui} \cdot d_{iv}$ // Matrix multiplikation

interessant! Hat Matrixmultiplikation eine Bedeutung für Graphen?

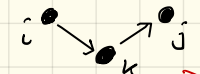


$G \quad \longleftrightarrow \quad A_G = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$

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

Bedeutung?

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

$\Leftrightarrow$ 

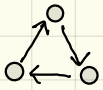
zählt Wege der Länge 2!

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

Beweis: Induktion über k Übung!

Andersherum: Potenzierung von A_G ist ein DP zur Lösung von:
Wieviele Wege gibt es von u nach v, für alle $u, v \in V$

Anwendung 1: Anzahl Dreiecke in gerichtetem Graphen ohne Schleifen ($\Leftrightarrow (u,u) \notin E$)

Dreieck: 

$\text{Spur}(A_G^3)/3$

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

$\Rightarrow$ Laufzeit $O(n^4)$
all-pairs shortest path: $O(n^3)$

nicht kompetitiv

oder geht Matrixmultiplikation vielleicht besser als $\Theta(n^3)$?

Optional:

Matrixmultiplikation nach Strassen (1969)

ähnliche Idee wie Karatsuba

Blocken für Rekursion: alle Matrizen $n \times n$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}_A \cdot \begin{pmatrix} e & f \\ g & h \end{pmatrix}_B = \begin{pmatrix} u & v \\ w & x \end{pmatrix}_C$$

$$\left.\begin{aligned} u &= ae + bg \\ v &= af + bh \\ w &= ce + dg \\ x &= cf + dh \end{aligned}\right\} \text{ 8 Matrixmult. halber Größe}$$

$T(n) = 8\,T(n/2) + c \cdot n^2$
$= \Theta(n^3)$
bringt nichts

Strassen:

$$\begin{aligned} \alpha_1 &= (a+d)(e+h) \\ \alpha_2 &= (c+d)\,e \\ \alpha_3 &= a(f-h) \\ \alpha_4 &= d(g-e) \\ \alpha_5 &= (a+b)\,h \\ \alpha_6 &= (c-a)(e+f) \\ \alpha_7 &= (b-d)(g+h) \end{aligned}$$

7 Matrixmult. halber Größe

$$\begin{aligned} u &= \alpha_1 + \alpha_4 - \alpha_5 + \alpha_7 \\ v &= \alpha_3 + \alpha_5 \\ w &= \alpha_2 + \alpha_4 \\ x &= \alpha_1 - \alpha_2 + \alpha_3 + \alpha_6 \end{aligned}$$

$= a(f-h) + (a+b)h = af + bh$ ✓

$$T(n) = 7\,T(n/2) + c \cdot n^2 = \Theta(n^{\log_2 7}) \approx \Theta(n^{2,8\ldots})$$