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

$G = (V, E, c)$ Graph mit Kostenfunktion $c: E \to \mathbb{R}$, $|V| = n$, $|E| = m$

↑ gerichtet oder ungerichtet

Kosten	Algorithmus	Laufzeit
alle Kosten 1	Breitensuche	$O(m+n)$
$c(e) \geq 0$	Dijkstra	$O((m+n) \cdot \log n)$ oder $O(m + n \log n)$
$c(e) \in \mathbb{R}$ auch negativ	Bellman-Ford	$O(n\,m)$

↓ allgemeiner

nicht besprochen, Verbesserung mit Fibonacci-Heaps

- One-to-one ist nicht schneller als one-to-all
- Falls G keine Zyklen hat, kann man $O(m+n)$ für beliebige Kosten erreichen: topologische Sortierung + DP

All Pairs Shortest Path ← Name etwas irreführend: wir suchen Wege, nicht Pfade

Finde für alle Paare $u, v \in V$ einen kürzesten u-v-Weg

Kanten	Algorithmus	Laufzeit
alle Kanten 1	$n \times$ Breitensuche	$O(nm + n^2)$
$c(e) \geq 0$	$n \times$ Dijkstra	$O(n \cdot (m+n) \cdot \log n)$ Fibonacci-Heaps
		oder $O(nm + n^2 \log n)$
$c(e) \in \mathbb{R}$ auch negativ	$n \times$ Bellman-Ford	$O(n^2 m)$
	Floyd-Warshall	$O(n^3)$
	Johnson	$O(n \cdot (m+n) \cdot \log n)$ } selbe Laufzeit wie $n \times$ Dijkstra
		oder $O(nm + n^2 \log n)$ Fibonacci-Heaps

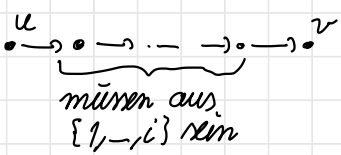
allgemeiner ↓

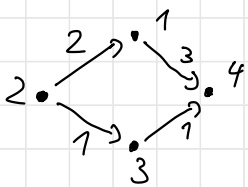
↑ falls es keine negativen Zyklen gibt

Floyd-Warshall-Algorithmus (all pairs shortest path)

DP! Nummerieren Knoten $1 \dots n$

Teilproblem: d_{uv}^{i} = Länge von kürzestem u-v-Weg, der nur Zwischenknoten aus $\{1, \dots, i\}$ benutzen darf

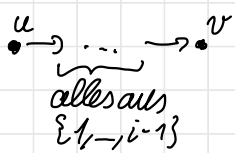




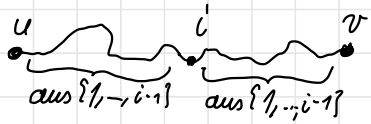
$d_{24}^{2} = 5$

Rekursion: 2 (3) mögliche Wege für d_{uv}^{i}

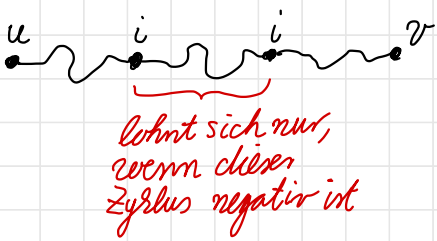
1) i kommt auf dem Weg nicht vor



2) i kommt auf dem Weg genau einmal vor



3) i kommt auf dem Weg mehrfach vor



$$i > 0: \quad d_{uv}^{i} = \min\{ d_{uv}^{i-1}, d_{ui}^{i-1} + d_{iv}^{i-1} \}$$

funktioniert, falls es keine negativen Zyklen gibt

$$i = 0: \quad u = v: \; d_{uu}^{0} = 0$$

oder $c(u,u)$ falls (u,u) Kante von negativem Gewicht ist

$$u \neq v: \; d_{uv}^{0} = \begin{cases} c(u,v), & \text{falls } (u,v) \in E \\ \infty, & \text{sonst} \end{cases}$$

Floyd Warshall (G) // Annahme: $V = \{1, \dots, n\}$

// Initialisierung

for $u \in V$: $d^0_{uu} \leftarrow 0$ ← falls keine negativen Schleifen

for $u \in V, v \in V \setminus \{u\}$: if $(u,v) \in E$ then $d^0_{uv} \leftarrow c(u,v)$; else $d^0_{uv} \leftarrow \infty$

// DP

for $i = 1 \dots n$

  for $u = 1 \dots n$

    for $v = 1 \dots n$

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

return d^n ← $n \times n$-Matrize mit Resultaten

Laufzeit $O(n^3)$

Speicherplatz $O(n^3)$ (kann man auf $O(n^2)$ verbessern)

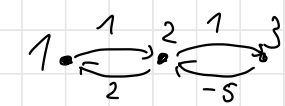
Kürzesten Weg finden: Rückverfolgen

Ist nur korrekt, falls es keine negativen Zyklen gibt!!

OPTIONAL: Was gibt der Algorithmus aus, wenn es negative Zyklen gibt?

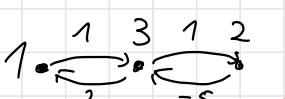
Beobachtung:
- Die Rekursion berücksichtigt alle Pfade. (Fall 1 oder 2) — Weg ohne doppelte Knoten
- Andere Wege werden teilweise berücksichtigt, teilweise nicht

Beispiel:

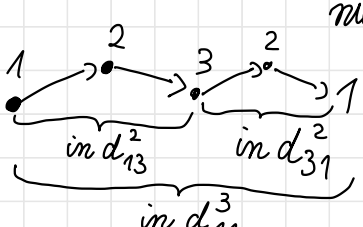
a) 

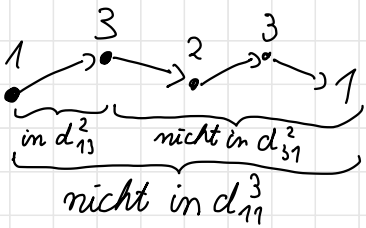
$\min\{0, 3\} = 0$

$d^3_{11} = \min \{ d^2_{11}, d^2_{13} + d^2_{31} \} = -1$ (2, −3)



$d^3_{11} = \min \{ d^2_{11}, d^2_{13} + d^2_{31} \} = 0$ (0, 1, 2)



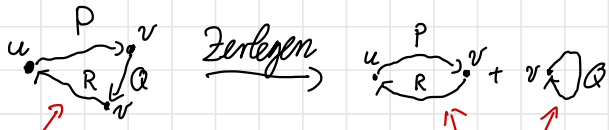


Behauptung: Es gibt negativen Zyklus $\iff$ Es gibt v mit $d^n_{vv} < 0$

Beweis: „$\Rightarrow$": Starte mit einem negativen Zyklus $v \rightsquigarrow v$. Durch iteriertes Zerlegen (Skizze) finden wir auch einen negativen Kreis C, Länge 1 oder 2 erlaubt

Kreis: Zyklus ohne doppelte Knoten ausser Start = Ende

wird oft bei Definition von Kreis ausgeschlossen

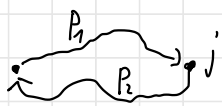


negativer Zyklus mit doppeltem Knoten

mindestens ein Teilzyklus muss negativ sein.

Fall 1: C hat Länge ≥ 2

Sei j der maximale Index auf C und $i \in C$ beliebig

C: 

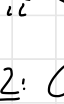
Die Pfade P_1 und P_2 werden in d^{j-1}_{ij} bzw d^{j-1}_{ji} berücksichtigt.

$\rightsquigarrow d^{j-1}_{ij} \leq c(P_1)$, $d^{j-1}_{ji} \leq c(P_2)$

$\rightsquigarrow d^{j}_{ii} \leq d^{j-1}_{ij} + d^{j-1}_{ji} \leq c(P_1) + c(P_2) = c(C) < 0$

$\rightsquigarrow d^n_{ii} < 0$

Bei Initialisierung beachten

Fall 2: C hat Länge 1: 

. Dann $d^0_{uu} < 0$. $\rightsquigarrow d^n_{uu} < 0$.

„$\Leftarrow$": Falls $d^n_{vv} < 0$, finden wir durch Rückverfolgen v-v-Weg der Länge $d^n_{vv} < 0$. ∎

Zusammenfassung Floyd-Warshall:

1) Lasse Floyd-Warshall laufen

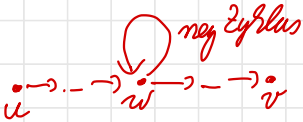
2) Falls es ein v gibt mit $d_{vv}^{n} < 0$ $\rightsquigarrow$ negative Zyklen, Ausgabe falsch

3) sonst: keine negativen Zyklen, Ausgabe korrekt

Allgemeine Antwort für kürzeste u-v-Wege:

Falls es einen Knoten w gibt mit $d_{uw}^{n} < \infty$, $d_{ww}^{n} < \infty$ und $d_{wv}^{n} < 0$,

dann ist die „kürzeste Länge" $-\infty$, sonst d_{uv}^{n}.

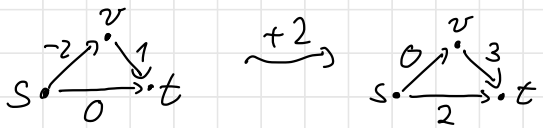


Algorithmus von Johnson

Idee: - mache alle Kantengewichte ≥ 0

- dann $n \times$ Dijkstra

Ansatz 1: Erhöhe alle Kantengewichte um c



Kosten von $s \to v \to t$: $\boxed{-1}$ $\quad$ $s \to v \to t$: 3 ← um 4 erhöht

$s \to t$: 0 $\quad$ $s \to t$: $\boxed{2}$ ← um 2 erhöht

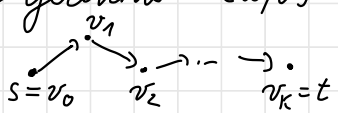
Kürzester Weg hat sich geändert! ⇝ Funktioniert nicht

Wir brauchen eine Methode, bei der sich alle u-v-Pfade um denselben Wert ändern!

Idee: teleskopierende Summen

Ansatz 2: Wähle zu jedem Knoten v eine Höhe $h(v) \in \mathbb{R}$ und setze neue Gewichte $\hat{c}(u,v) := c(u,v) + h(u) - h(v)$

⇝ Länge eines Weges



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

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

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

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

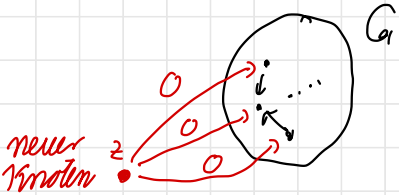
Länge vorher $\quad$ gleicher Term für alle s-t-Wege

⇝ Kürzester Weg bleibt der kürzeste

Aber: Wir wollen, dass alle Kantengewichte $\hat{c} \geq 0$ sind, um Dijkstra anzuwenden

neues Ziel: Finde $h: V \to \mathbb{R}$, sodass $\hat{c}(u,v) = c(u,v) + h(u) - h(v) \geq 0$ für alle $(u,v) \in E$

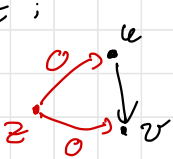
Idee (Johnson):



Füge neuem Knoten z ein und je eine Kante (z,v) mit $c(z,v) = 0$ für alle $v \in V$.

$h(u) :=$ Länge kürzesten Weg $z \rightsquigarrow u$ ← also alle $h(u) \leq 0$ funktioniert nicht mit negativen Zyklen $\rightsquigarrow h(u) = -\infty$

Denn dann gilt für alle $(u,v) \in E$:

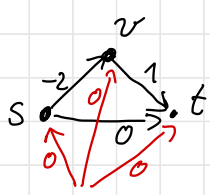


$$h(v) \leq h(u) + c(u,v)$$

$$\rightsquigarrow c(u,v) + h(u) - h(v) \geq 0$$

$$\rightsquigarrow \hat{c}(u,v) \geq 0$$

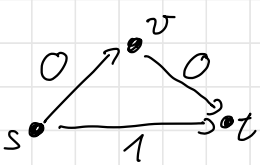
Beispiel:



$h(s) = 0$
$h(v) = -2$
$h(t) = -1$

$\rightsquigarrow$

neue Kosten $\hat{c}$:



$c(s \to v \to t) = \boxed{-1}$
$c(s \to t) = 0$

$\hat{c}(s \to v \to t) : \boxed{0}$
$\hat{c}(s \to t) : 1$

} beide um denselben Wert erhöht: $h(s) - h(t) = 1$

Algorithmus: 1) Erzeuge z und neue Kanten $O(n)$
2) Berechne h mit Bellman-Ford $O(m \cdot n)$
3) n-mal Dijkstra mit $\hat{c}$ $n \times O((m+n)\log n)$ oder $n \times O(m + n\log n)$

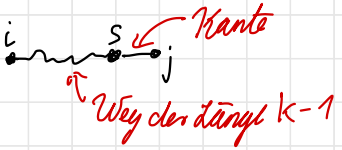
Insgesamt: $O(n \cdot (m+n) \cdot \log n)$ $O(mn + n^2 \log n)$

Besser als Floyd-Warshall für dünnbesetzte Graphen

Matrizen und Graphen

Annahme: $V = \{1, \dots, n\}$. Betrachte i-j-Wege der Länge k.

Jeder solche Weg sieht so aus:



1) $L_{ij}^{(k)} :=$ „Gibt es so einen Weg?"

$$\rightsquigarrow \quad L_{ij}^{(k)} = \bigvee_{s=1}^{n} \left(L_{is}^{(k-1)} \wedge L_{sj}^{(1)} \right)$$

2) $M_{ij}^{(k)} :=$ minimale Kosten von so einem Weg

$$\rightsquigarrow \quad M_{ij}^{(k)} = \min_{s=1,\dots,n} \left\{ M_{is}^{(k-1)} + M_{sj}^{(1)} \right\}$$

„tropischer Halbring, min+plus-Halbring" fast ein Ring, außer dass additive Inverse fehlen

3) $N_{ij}^{(k)} :=$ Anzahl solcher Wege

$$\rightsquigarrow \quad N_{ij}^{(k)} = \sum_{s=1}^{n} N_{is}^{(k-1)} \cdot N_{sj}^{(1)}$$

Matrixmultiplikation

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

Adjazenzmatrix A_G

Satz: Das Element (i,j) in A_G^k (k-te Potenz) ist die Anzahl der i-j-Wege der Länge k.

Beweis (Induktion nach k):

$k = 1$: ein Weg, falls $(i,j) \in E$, sonst kein Weg. ✓

$k > 1$: $N_{ij}^{(k)} = \sum_{s=1}^{n} N_{is}^{(k-1)} \cdot N_{sj}^{(1)}$

$N_{is}^{(k-1)}$: Einträge von A_G^{k-1} nach I.-H.

$N_{sj}^{(1)}$: Einträge von A_G

$\overset{\text{I.-H.}}{=}$ Produkt (i-ter Zeilenvektor von A_G^{k-1}) $\cdot$ (j-ter Spaltenvektor von A_G)

$=$ (i,j)-Eintrag von A_G^k ▨

Anwendung 1: Wie viele Dreiecke gibt es in G? G gerichtet ohne Schleifen (kein (v,v) in E)

↑ Kreise der Länge 3

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

Anwendung 2: Kann man von jedem Knoten aus alle anderen Knoten erreichen?

Trick: Füge alle Schleifen zu E hinzu, also alle (v,v).

$\rightsquigarrow$ Wenn es einen $u \rightsquigarrow v$-Pfad gibt, dann gibt es auch einen $u \sim v$-Weg der Länge n. (Laufe zu v und warte dort.)

$\rightsquigarrow$ Es genügt, A_G^n zu berechnen und zu prüfen, ob alle Einträge >0 sind.

Laufzeit zur Berechnung von A_G^n:

naiv: $n-1$ Matrixmultiplikationen

Übungsaufgabe 3.5 ↘ iteriertes Quadrieren: $A_G^n = \begin{cases} A_G^{n/2} \cdot A_G^{n/2}, & \text{falls } n \text{ gerade} \\ A_G^{(n-1)/2} \cdot A_G^{(n-1)/2} \cdot A, & \text{falls } n \text{ ungerade} \end{cases}$

$\rightsquigarrow O(\log n)$ Matrixmultiplikationen

Es gibt noch schnellere Methoden für Anwendung 2 $\rightsquigarrow$ nächstes Semester

Verbleibende Frage: Wie teuer ist Matrixmultiplikation?

Matrixmultiplikation ($n \times n$-Matrizen $C = A \cdot B$)

naiv: $c_{ij} = \sum_{s=1}^{n} a_{is} \cdot b_{sj}$

$O(n)$ pro Eintrag $\rightsquigarrow$ insgesamt Laufzeit $O(n^3)$

Strassen (bekannt aus Linearer Algebra):

- Zerlege $n \times n$-Matrizen in je 4 Teilmatrizen der Größe $\frac{n}{2} \times \frac{n}{2}$
- Führe 7 Multiplikationen und 18 Additionen der Teilmatrizen durch.
- Iteriere

Laufzeit: $T(n) = 7 \cdot T(\frac{n}{2}) + \Theta(n^2)$

$\rightsquigarrow T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.807\ldots})$ (Master-Theorem)

GEHT ES BESSER?

JA: Ähnliche Ideen mit noch komplizierteren Formeln geben bessere Laufzeit.

Momentaner Rekord: $O(n^{2.3715\ldots})$

Große offene Frage: Kommt man mit dem Exponenten beliebig nahe an 2?