Algorithmen & Datenstrukturen

Herbst 2018

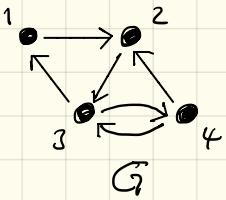
Vorlesung 13

Graphen: Kurze Wiederholung

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

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

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



G

Datenstruktur für Graphen?

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

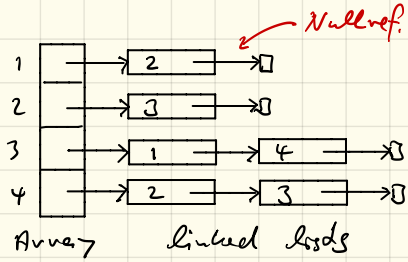
$$a_{ij} = \begin{cases} 1 & (i,j) \in E \\ 0 & \text{sonst} \end{cases}$$

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 $O(n+m)$

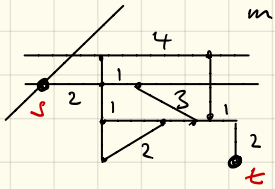
zu jedem Knoten Liste der Nachbarn (Reihenfolge egal)

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

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

Kürzeste Wege

Google Maps:



Straßennetzwerk = Graph mit positiven Gewichten (= Distanzgraph)

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

gesucht: kürzester Weg von s nach t

Bisher in Vorlesung:

Graph	one-to-all shortest path	all-to-all shortest path
(V, E)	Breitensuche $O(m+n)$	$n \times$ Breitensuche $O(mn + n^2)$ ⊗
(V, E, c) $c: E \to \mathbb{R}^+$	?	?
(V, E, c) $c: E \to \mathbb{R}$	Bellman Ford $O(mn)$	Floyd-Warshall $O(n^3)$

allgemeiner ↓

⊗ Heißt: Breitensuche von jedem Knoten $s \in V$

Beachte: one-to-one kommt in der Tabelle nicht vor da diese Algorithmen immer one-to-all berechnen

One-to-all shortest path, Distanzgraph

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

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

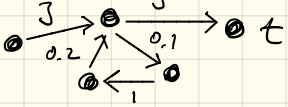
$(v_i, v_{i+1}) \in E$, $i = 0 \dots k-1$

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

Bemerkung: Kann man physisch lösen: Bastle Graph mit Schnüren, ziehe s und t so weit wie möglich auseinander.

Existenz des kürzesten Wegs?

Denn: es kann unendlich viele Wege geben!



Aber Zyklen machen den Weg nur länger (da $c: E \to \mathbb{R}^+$)

$\Rightarrow$ kürzester Weg ist zyklenfrei

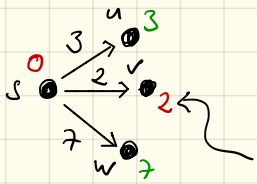
$\Rightarrow$ hat $\leq n$ Knoten

davon gibt es nur endlich viele

Also: wenn es einen Weg gibt, dann gibt es einen kürzesten

Algorithmus:

Anfang:

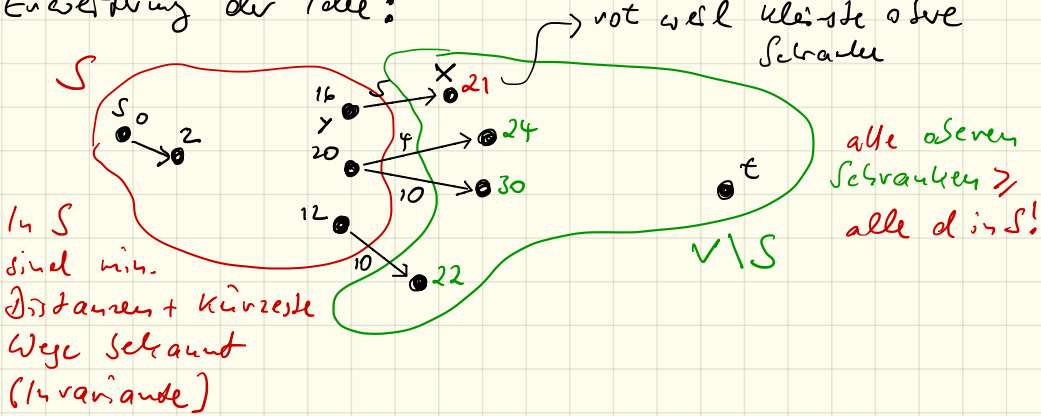


t

ist nat weil kleinste obere Schranke
⇒ ist minimale Distanz von s

Minimale Distanz von s
Obere Schranke für Distanz von s

Erweiterung der Idee:

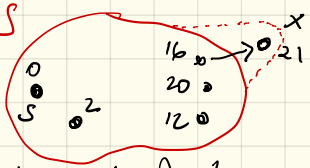


- $S \subseteq V$: Knoten bei denen man min. Distanz weiß
- Induktion über $|S|$
- Wähle $x \in V \setminus S$ mit kleinster oberer Schranke und füge zu S
- Merke Vorgänger von x in Array: $a[x] = y$ (ermöglicht Auslesen des kürzesten Weges)

$S = \{s\}$, $d(s) = 0$
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 $|S|$. $|S| = 1$ ✓



Anm. kürzeste Distanzen bekannt

kleines S

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

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

Präzisierung des Algorithmus

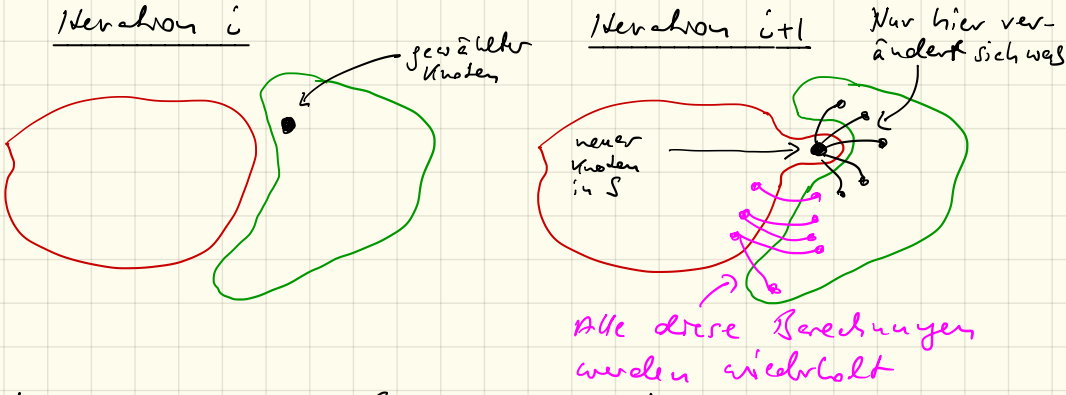
"wähle (u, v) ..." trivial: scanne alle Pfeile durch, prüfe $u \in S$, $v \notin S$, merke min. oder scanne

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

(wie Bellmann-Ford)

"wähle" nach Dijkstra:

Beobachtung: redundante Berechnungen zwischen Iterationen



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

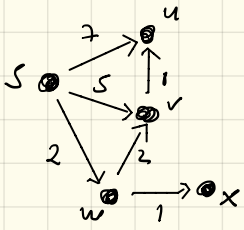
Algorithmus:

$d(s) = 0$
$d(v) = \infty$ für alle $v \in V \setminus \{s\}$
$S = \emptyset$
while $|S| < n$ do
  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
      $d(w) = \min(d(w), d(v) + c(v,w))$ // update obere Schranke

unerreichbare Knoten haben am Ende $d = \infty$

□ : ist min wurde gewählt

Beispiel:



init: $S = \emptyset$, $d(s) = 0$

$d(u) = d(v) = d(w) = d(x) = \infty$

Iteration 1: $S = \{s\}$

$d(u) = 7$ $\quad d(w) = 2$

$d(v) = 5$ $\quad d(x) = \infty$

Distanz von s: 2 — Iteration 2: $S = \{s, w\}$

$d(u) = 7$ $\quad d(x) = 3$

$d(v) = 4$

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

$d(u) = 7$ $\quad d(v) = 4$

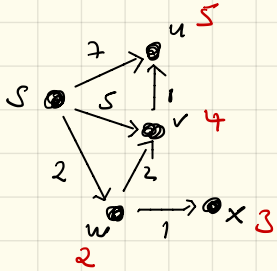
Distanz von s: 4 — Iteration 4: $S = \{s, w, x, v\}$

$d(u) = 5$

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

wächst

Distanzen d nach Ende:



Analyse Algorithmus:

init	$O(n)$
n extractmin	$O(n \log n)$
$\leq m$ decrease key	$O(m \log n)$
	$O((m+n) \log n)$

mit Heap

Heap: extractmin $O(\log n)$

decrease key $O(\log n)$ wenn man weiß wo das Objekt steht → gespeicherte Link in Adjazenzliste

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

Also:

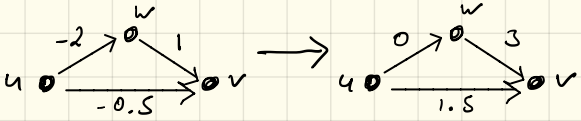
Graph	one-to-all shortest path	all-to-all shortest path
(V, E)	Breitensuche $O(m+n)$	$n \times$ Breitensuche $O(mn + n^2)$
(V, E, c) $c: E \to \mathbb{R}^+$	Dijkstra $O(m + n \log n)$	$n \times$ Dijkstra $O(mn + n^2 \log n)$
(V, E, c) $c: E \to \mathbb{R}$	Bellman Ford $O(mn)$	Floyd-Warshall $O(n^3)$

All-to-all shortest path, $G = (V, E, c)$, $c: E \to \mathbb{R}$

Johnson's Algorithmus:

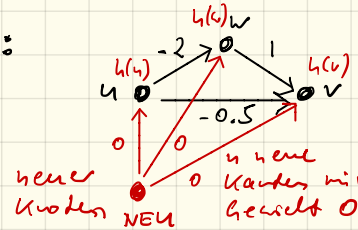
Idee: mache alle Kantengewichte positiv
→ dann kann Dijkstra verwendet werden!

Ansatz 1: Subtrahiere kürzesten Gerade



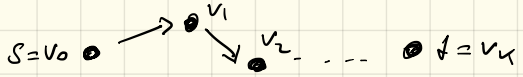
ändert den kürzesten Weg (hier: $u \rightsquigarrow v$) da Subtraktion von Anzahl der Pfeile abhängt

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.) Bleibt 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: $\hat{c}(s \rightsquigarrow s) = c(s \rightsquigarrow s)$

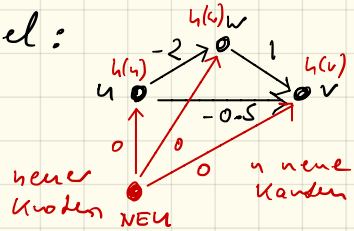
Wie definieren wir h so daß $\hat{c}$ positiv?

$h(u)$ = Länge kürzester Weg NEU $\rightsquigarrow u$

Dann dann für jede $(u,v) \in E$:

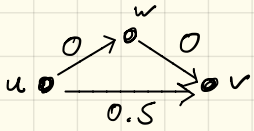
$$h(v) \leq h(u) + c(u,v)$$
$$\Leftrightarrow \quad c(u,v) + h(u) - h(v) \geq 0$$
$$\Leftrightarrow \quad \hat{c}(u,v) \geq 0$$

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 dünn besetzte Graphen)