Algorithmen & Datenstrukturen

Herbst 2017

Vorlesung 13

Letztes Mal: Kürzeste Wege, one-to-all:

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

Fibonacci Heaps:

Betweenness Centrality: Auf wieviielen kürzesten Wegen ist ein Knoten im Graph?

Brauchen "all-to-all" shortest path

Kürzeste Wege, all-to-all $G = (V, E, c)$, $V = \{1, \dots, n\}$, $|E| = m$

1.) $c: E \to \mathbb{R}^+$ n Mal Dijkstra $O(mn + n^2 \log n)$

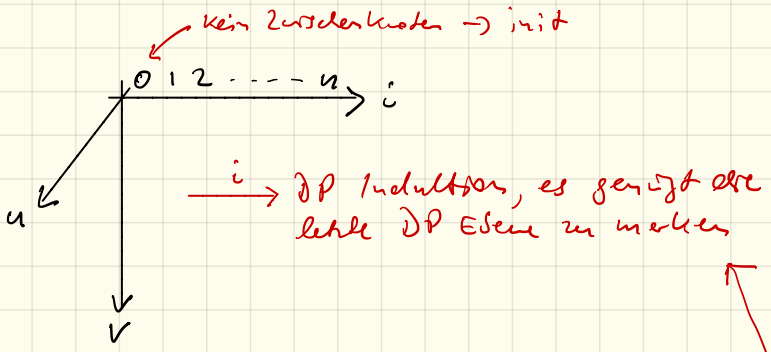
2.) $c: E \to \mathbb{R}$ n Mal Bellman-Ford $O(mn^2)$

3.) Floyd-Warshall Algorithmus: DP!

Idee: d_{uv}^i = kürzeste Weglänge $u \to v$ mit Zwischenknotennummern $\le i$

Induktion über i: $d_{uv}^{i} = \min(d_{uv}^{i-1}, d_{ui}^{i-1} + d_{iv}^{i-1})$

3D-Tableau




```
// init
for u ∈ V: d(u,u) = 0
for (u,v) ∈ E: d(u,v) = c(u,v), else d(u,v) = ∞
// DP
for i = 1..n
    for u = 1..n
        for v = 1..n
            d_uv^i = min(d_uv^(i-1), d_ui^(i-1) + d_iv^(i-1))   // kann man inplace machen
```

Funktioniert wenn keine negativen Zyklen

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

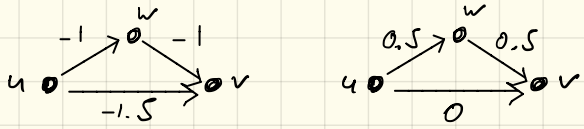
Kürzeste Wege merken: $O(n^2)$ Extraplatz (ähnlich wie bei Dijkstra)

$\Rightarrow O(n^3)$

4.) Johnson's Algorithmus:

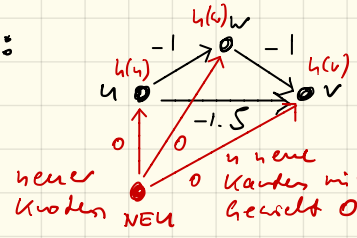
Idee: Mache alle Kantengewichte c positiv

Ansatz 1: Subtrahiere kürzestes Gewicht



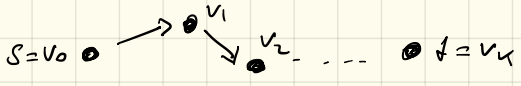
ändert den kürzesten Weg (hier $u \to v$) der Subtraktion von Anzahl der Pfeile abhängt

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.) Bleiben kürzeste Wege kürzest:



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

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

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

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

hängt nur von s und t ab

2.) Zyklenfreiheit stellen sicher: $\hat{c}(s \rightsquigarrow s) = c(s \rightsquigarrow s)$

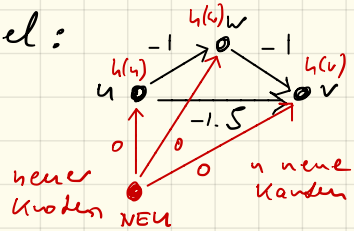
Wie definieren wir h so daß $\hat{c}$ positiv?

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

Denn dann für jede $(u,v) \in E$:

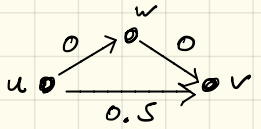
$$h(v) \leq h(u) + c(u,v)$$
$$\Leftrightarrow \quad c(u,v) + h(u) - h(v) \geq 0$$
$$\Leftrightarrow \quad \hat{c}(u,v) \geq 0$$

Im Beispiel:



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

$\Rightarrow$



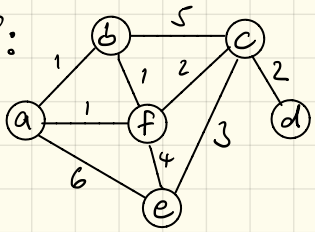
Analyse:

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)

Minimale Spannbäume $|V| = n$, $|E| = m$

G:

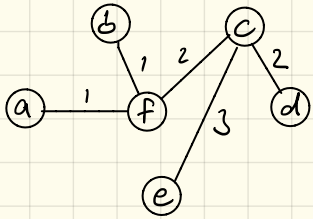


Gegeben: ungerichteter, zus. hängender, gewichteter Graph
$G = (V, E, c)$, $c: E \to \mathbb{R}$

Gesucht: minimaler Spannbaum, also
$T = (V, E')$, $E' \subseteq E$, Baum
so daß $\sum_{e \in E'} c(e)$ minimal

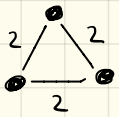
MST = minimal spanning tree

T: (eine Lösung)



Lösung existiert immer? Es reicht zu zeigen dass es immer einen Spannbaum gibt (warum?).
Beweis dazu: Induktion nach n (Lösung) ✓

Lösung eindeutig? Nein, z.B. G:

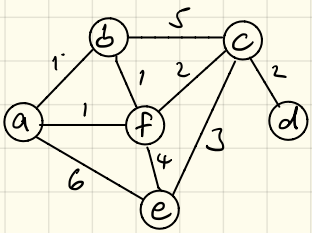


Algorithmus?

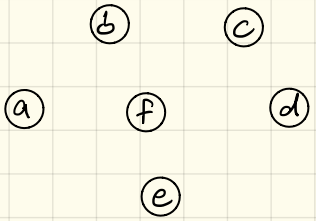
Idee: Greedy

Fange an mit $E' = \emptyset$ und wähle immer die billigste Kante die keinen Kreis verursacht.

Graph:



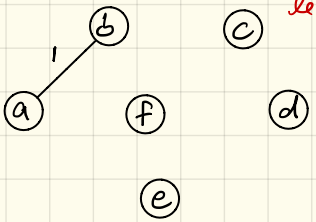
Schritt 0:



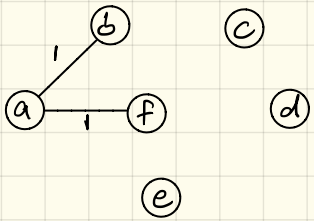
Anfang: 6 Bäume mit je einem Knoten

Schritt 1:

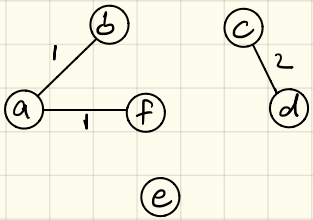
wenn es Auswahl gibt, wähle z.B. lexikographisch



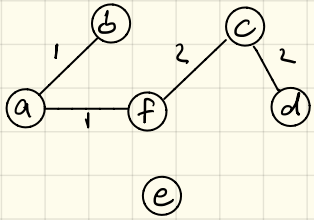
Schritt 2:



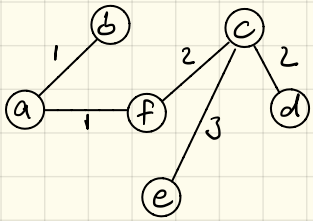
Schritt 3:



Schritt 4:



Schritt 5:



Bemerkung: am Anfang gab es Auswahl
→ MST ist hier nicht eindeutig

MST-Kruskal(G) (Kruskal, 1956)

sortiere Kanten nach Gewicht: $c(e_1) \leq \dots \leq c(e_m)$

$E' = \emptyset$

for $k = 1 \dots m$

if $(V, E' \cup \{e_k\})$ ist kreisfrei // wenn nicht, verwirf Kante

$E' = E' \cup \{e_k\}$ // wähle Kante

return (V, E')

Zu jedem Zeitpunkt während des Algorithmus ist (V, E') ein Wald:

Zusammenhangskomponenten = Bäume des Waldes

Ist MST-Kruskal korrekt?

MST-Kruskal betrachtet alle Kanten e und

a.) entweder wählt e, oder b.) verwirft e (wenn e nicht gewählt wird, wird sie nicht mehr in Betracht gezogen).

Wir zeigen jetzt dass ein Algorithmus der das Wählen und Verwerfen richtig macht eine korrekte Lösung liefert.

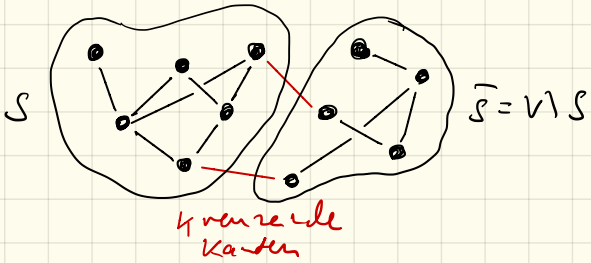
Dazu definieren wir: (während des Algorithmus)

Gew = Menge der gewählten Kanten

Ver = Menge der verworfenen Kanten

Un = Menge der unentschiedenen Kanten

Schnitt von G: Eine Partition $V = S \cup V \setminus S = S \cup \bar{S}$ von V. Eine Kante $e \in E$ kreuzt den Schnitt wenn ein Endpunkt in S, und einer in $\bar{S}$:

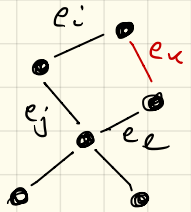


Auswahlregel: Wähle einen Schnitt den keine gewählte Kante kreuzt. Unter allen unentschiedenen Kanten die kreuzen, wähle die mit min. Gewicht.

Verwerfregel: Wähle einen Kreis ohne verworfene Kanten. Unter allen unentschiedenen Kanten im Kreis verwerfe die mit max. Gewicht.

MST-Kruskal wendet beide Regeln an!

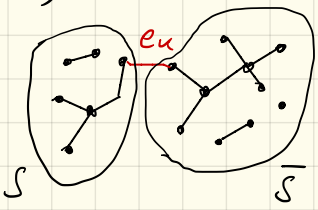
1.) e_k wird verworfen $\Rightarrow$ e_k ist in Kreis und hat höchstes Gewicht dort



$i, j, \ell < k \Rightarrow c(e_i), c(e_j), c(e_\ell) \leq c(e_k)$

2.) e_H wird gewählt $\Rightarrow$ e_H verbindet zwei Zusammenhangskomponenten

$\Rightarrow$ wir können Schnitt wählen den e_H kreuzt und dort minimal ist (da alle anderen kreuzenden später kommen)



Satz: Jeder Algorithmus der, ausgehend von $U_H = V$, solche Regeln anwendet bis $U_H = \emptyset$ ist korrekt.

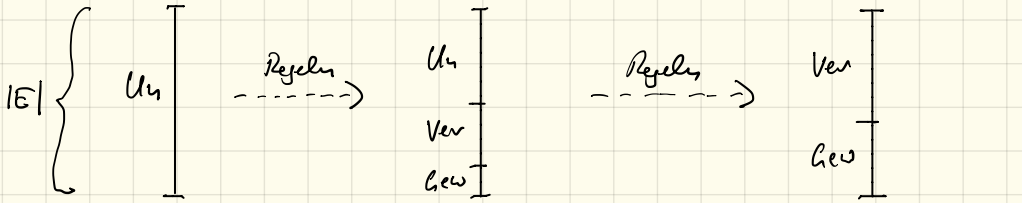
Beweis: über Auswahlinvariante

Auswahlinvariante: Es gibt stets einen MST der alle gewählten und keine verworfenen Kanten enthält.

Wir zeigen: Beide Regeln erhalten diese Invariante.

Denn dann:

Anfang: $E = U_H$ Mitte: $E = U_H \cup Ver \cup Gew$ Ende: $E = Ver \cup Gew$



Invariante gilt (offensichtlich)

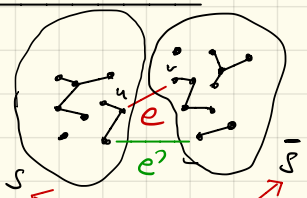
Invariante gilt

Invariante gilt

$\Rightarrow (V, Gew)$ ist Spannbaum

Beweis Auswahlregel erhält Invariante:

Situation: Invariante gilt für Gew (in schwarz) und Ver (nicht im Bild) mit MST T.
e gewählt mit Auswahlregel



eingehender Schnitt der e minimal kreuzt

Fall 1: $e \in T$ ✓

Fall 2: $e \notin T$, $e = (u,v)$

$\Rightarrow$ es gibt einen Weg von u nach v in T, dieser muss den Schnitt kreuzen:
sagen wir sei e', e' ist in T

$e' \notin$ Ver (da in T)
$e' \notin$ Gew (da Auswahlregel angewendet)
daher $e' \in$ Un

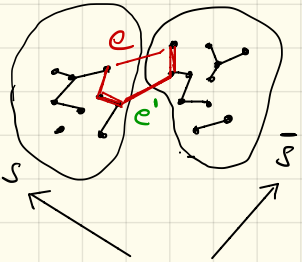
$\Rightarrow c(e') \geq c(e)$ (Auswahlregel)

$\Rightarrow T' = T \setminus \{e'\} \cup \{e\}$ ist MST
und erfüllt Invariante nach Wahl von e

(impliziert auch $c(e') = c(e)$ da sonst T' billiger wäre als T)

Beweis Verwerfregel erhält Invariante:

Situation: Invariante gilt für Gew (in schwarz) und Ver (nicht im Bild) mit MST T.
e verworfen mit Verwerfregel (zugehöriger Kreis in rot)



Fall 1: $e \notin T$ ✓

Fall 2: $e \in T$

$\Rightarrow$ $T\setminus\{e\}$ hat 2 Zus. Komponenten und macht daher einen Schnitt $V = S \cup \bar{S}$ den e kreuzt

$\Rightarrow$ der Kreis kreuzt den Schnitt ein zweites Mal, sagen wir sei e'

$e' \notin Ver$ (da Kreis mit Verwerfregel gewählt)
$e' \notin Gew$ (da sonst $e' \in T$ und T hätte Kreis)
daher $e' \in Un$

$\Rightarrow$ $c(e') \leq c(e)$ (Verwerfregel)

$\Rightarrow$ $T' = T\setminus\{e\} \cup \{e'\}$ ist MST und erfüllt Invariante nach Verwurf von e

(impliziert auch $c(e') = c(e)$ da sonst T' billiger wäre als T) ∎

Damit ist insbesondere MST-Kruskal korrekt.