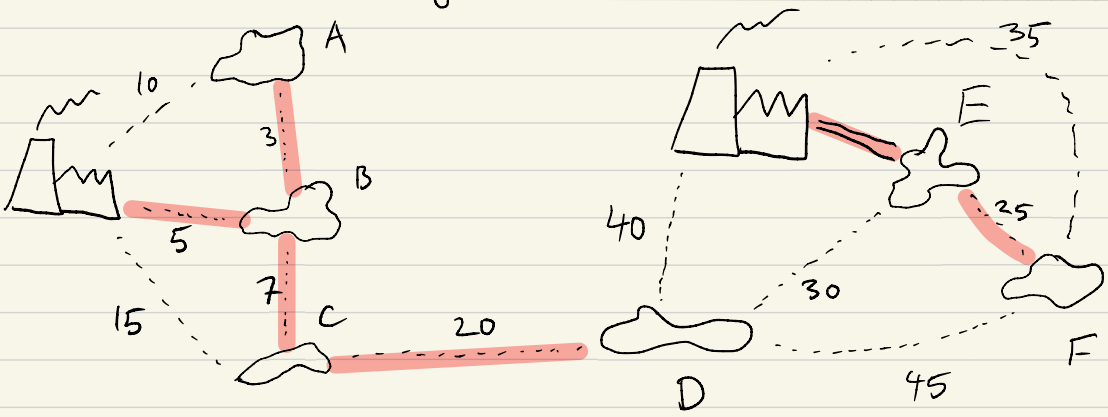
1926: Elektrifizierung Mährens (Region in Tschechien)

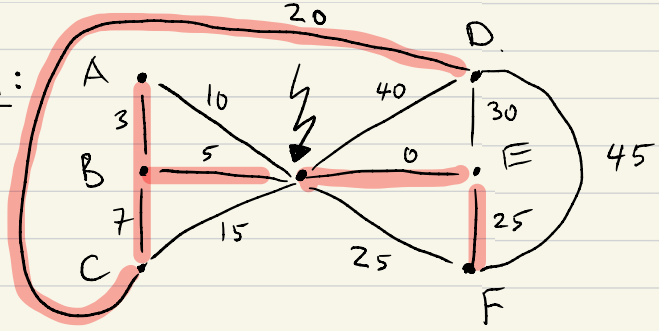


Frage an Otakar Borůvka (Mathematiker):

welche Stromleitungen bauen, so dass

- alle Städte mit Strom versorgt
- Gesamtkosten minimal

gewichteter Graph:



finde zusammenhängenden Teilgraph mit minimalem Gewicht

Formal: Graph $G = (V, E)$ zusammenhängend

Kantengewichte: $w(e) \geq 0$

aufspannende Kanten $A \subseteq E$: Graph (V, A) z.hängend

Spannbaum $T \subseteq E$: aufspannend, kein Kreis enthalten

Gewicht: $w(A) = \sum_{e \in A} w(e)$

Beobachtung: aufspannend & minimales Gewicht

$\Rightarrow$ Spannbaum (entferne Kante auf Kreis solange bis Spannbaum)

minimaler Spannbaum (MST): minimales Gewicht

weitere Anwendungen:

"clustering" zerlege Graph in k "gute" Teile

genauer: finde Teilgraph mit k ZHKs und minimalem Gewicht

Idee: berechne MST und entferne $k-1$ schwerste Kanten

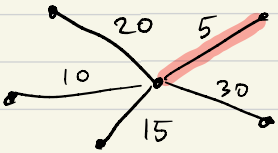
sichere Kante: enthalten in jedem MST

Strategie: finde sichere Kanten solange bis MST

Vereinfachung für Analyse: nehmen an
alle Kantengewichte verschieden

(später: Algorithmen funktionieren auch sonst)

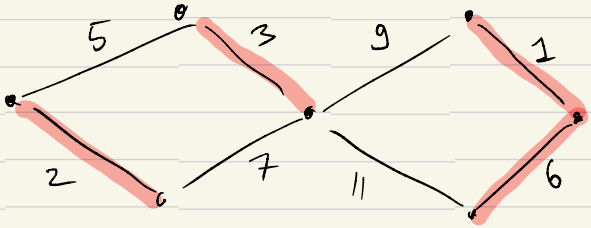
betrachte alle Kanten an einem Knoten:



welche Kante ist sicher?

Vermutung: minimale Kante an Knoten sicher

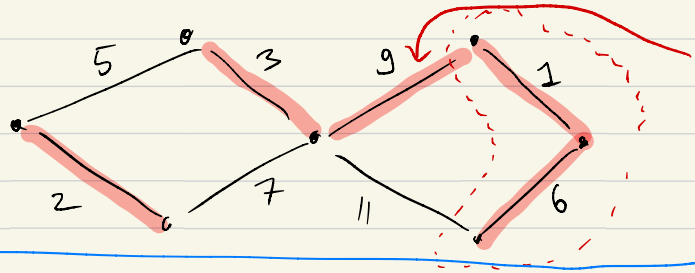
bilden diese Kanten MST?



nicht aufspannend
erhalten Wald

weitere sichere Kanten?

betrachte alle Kanten an ZHK des Walds:



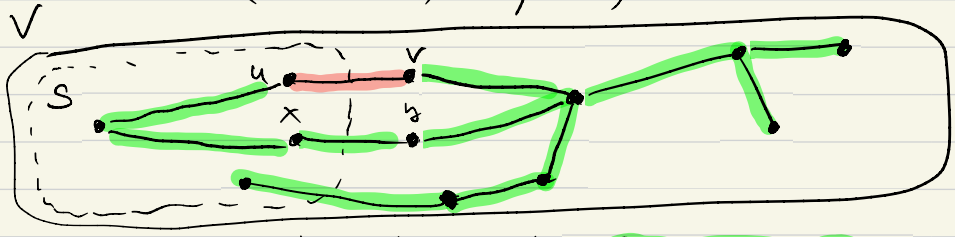
weitere sichere Kante

Vermutung: minimale Kante an ZHK sicher

allgemeine Behauptung (Schnittprinzip):

$\forall S \subseteq V$. minimale Kante uv an S sicher

$(u \in S,\ v \notin S)$



Beweis (indirekt): betrachte Spannbaum T, $uv \notin T$

sei $xy \in T$ an S (damit $w(x,y) > w(u,v)$)

ersetze xy mit uv in T

erhalte besseren Spannbaum ← wirklich Spannbaum?

$\rightsquigarrow$ T nicht MST

wichtig (sonst Beweis falsch):

wähle Kante xy auf Kreis in $T \cup \{uv\}$

Boruvka(G_1):

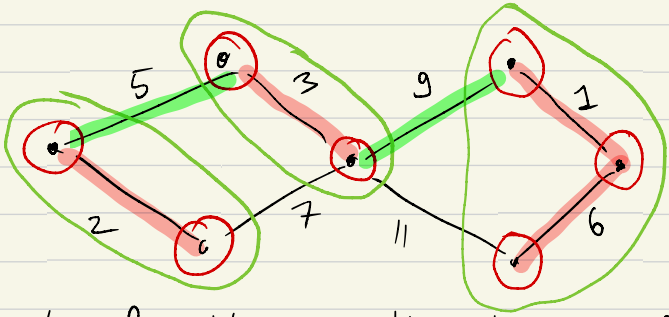
$F \leftarrow \emptyset$ (sichere Kanten)

WHILE F nicht Spannbaum

$(S_1, \dots, S_k) \leftarrow$ ZHKs von F

$(e_1, \dots, e_k) \leftarrow$ minimale Kanten an $S_1, \dots, S_k$

$F \leftarrow F \cup \{e_1, \dots, e_k\}$



Laufzeit pro Iteration: $O(|V| + |E|)$

Anzahl Iterationen: $O(\log |V|)$

(nach jeder Iteration: höchstens halb so viele ZHKs)

total: $O((|V| + |E|) \cdot \log |V|)$

Variation I: auf eine ZHK konzentrieren

Prim(G, s):

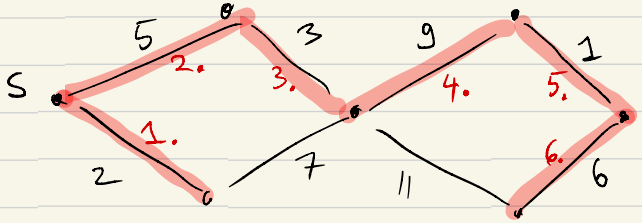
$F \leftarrow \emptyset$

$S \leftarrow \{s\}$ (ZHK von s in F)

WHILE F nicht Spannbaum

$uv^* \leftarrow$ minimale Kante an S ($u \in S, v^* \notin S$)

$F \leftarrow F \cup \{uv^*\}$; $S \leftarrow S \cup \{v^*\}$



Vergleiche Dijkstra!

minimale Kante schnell finden?

Priority Queue (e.g. min-heap):

- für jeden Knoten $v \in V \setminus S$ merke minimales Gewicht um v mit S zu verbinden
- verwende dieses Gewicht als "Schlüssel" (key) um Knoten zu vergleichen

(min-) heap

Operationen:

make-heap (V, ∞) setze alle Schlüssel auf ∞

extract-min (H) entferne Minimum

$O(\log n)$ (versickere Element um heap Eigenschaft wiederherzustellen)

decrease-key (H, v, c) verkleinere Schlüssel von v auf c (falls nötig)

$O(\log n)$ (Element aufsteigen lassen um heap Eigenschaft wiederherstellen)

Prim (G, s)

$H \leftarrow$ make-heap(V, ∞), $S \leftarrow \emptyset$

decrease-key $(H, s, 0)$

WHILE $H \neq \emptyset$

$v^* \leftarrow$ extract-min (H)

$S \leftarrow S \cup \{v^*\}$

FOR $(v^*, v) \in E$, $v \notin S$

decrease-key $(H, v, w(v^*, v))$

Laufzeit

wie Boruvka

gleiches Schema für kürzeste Wege

Dijkstra (G, s):

$H \leftarrow$ make-heap(V, ∞), $S \leftarrow \emptyset$

decrease-key$(H, s, 0)$

$d[s] \leftarrow 0$, $d[v] \leftarrow \infty$ für $v \in V \setminus \{s\}$

WHILE $H \neq \emptyset$:

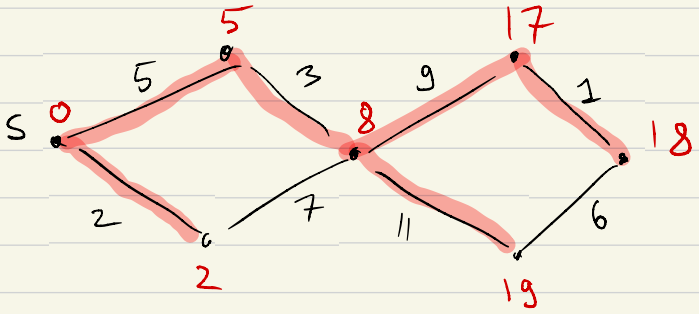
$v^* \leftarrow$ extract-min(H)

$S \leftarrow S \cup \{v^*\}$

FOR $(v^*, v) \in E$, $v \notin S$:

$d[v] \leftarrow \min(d[v], d[v^*] + w(v, v^*))$

decrease-key$(H, v, d[v])$



Shortest Path Tree

$\neq$ MST !

Variation II: berechne sichere Kante sortiert nach aufsteigendem Gewicht

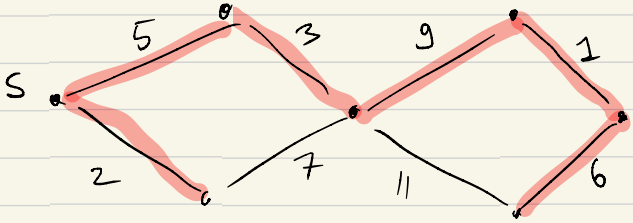
Kruskal (G):

$F \leftarrow \emptyset$ (sichere Kanten)

FOR $uv \in E$, aufsteigend sortier

IF u, v in verschiedenen ZHKs von F

$F \leftarrow F \cup \{uv\}$



Laufzeit (naiv): $O(|E| \cdot |V|)$

geht es besser?

brauchen Datenstruktur für ZHKs von F

Union-find Datenstruktur ($|V| = n$)

Operationen:

make (V): erstelle Datenstruktur (für $F = \emptyset$)

same (u, v): teste ob u, v in selber ZHK von F

union (u, v): vereinige ZHKs von u und v
(füge Kante uv zu F hinzu)

Speicher

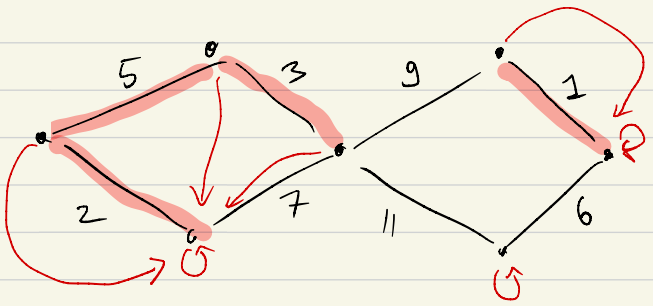
rep[v] : eindeutiger Repräsentant der ZHK von v

Implementierung:

make(V): $rep[v] \leftarrow v$ für alle $v \in V$
$O(n)$

same (u, v): $rep[u] = rep[v]$ $O(1)$

union (u, v): FOR $x \in V$, $rep[x] = rep[u]$
$rep[x] \leftarrow rep[v]$



nach 4 union Schritten

Laufzeit von union : $O(n)$

Idee : verwalte Liste alle Knoten in ZHK in "members" Array

Invariante members[r[u]] ist Liste aller Knoten in ZHK von u

union (u, v):

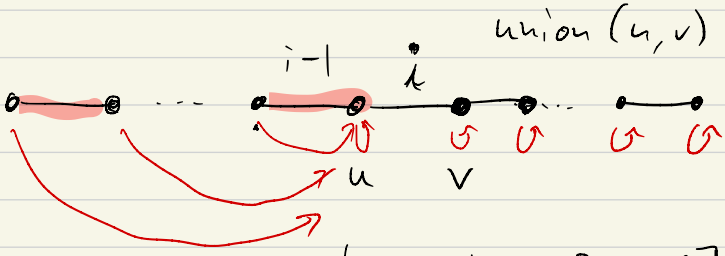
FOR $x \in$ members[r[u]]

r[x] ← r[v]

members[r[v]] ← members[r[v]] ∪ {x}

Laufzeit : $O(|\text{members}[r[u]]|)$

worst case:



$|members[r[u]]| = i$

total: $O(1 + 2 + \cdots + n-1)$

$= O(n^2)$

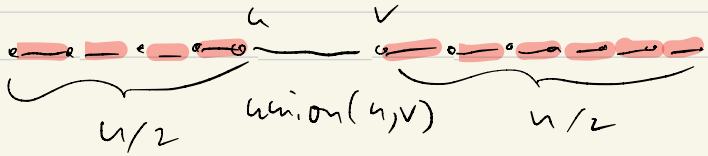
geht es besser?

idee: durchlaufe nur Knoten der kleineren ZHK

Laufzeit $O(\min\{|members[r[u]]|, |members[r[v]]|\})$

einzelne union operationen können Laufzeit $\Theta(n)$ haben

Bsp.:



Amortisierte Analyse:

totale Laufzeit aller union Operation $O(n \cdot \log n)$

Beweis: (nicht in Vorlesung besprochen)

sei N_u = Anzahl Änderungen von rep[u]

totale Laufzeit: $O(\sum_{u \in V} N_u)$

jede Änderung verdoppelt Grösse der ZHK

$\rightsquigarrow$ $N_u \leq \log_2 n$

$\rightsquigarrow$ totale Laufzeit $O(n \cdot \log n)$ ☐

Laufzeit von Kruskal:

$$O(\underbrace{|E| \cdot \log |E|}_{\text{sortieren}} + \underbrace{|V| \cdot \log |V|}_{\text{union find}})$$

nicht in Vorlesung besprochen

wenn nicht alle Kantengewichte verschieden?

Beobachtung: Algorithmen Vergleichs-basiert (Keine andere Berechnungen mit Kantengewichten)

$\rightsquigarrow$ brauchen nur eindeutige Sortierung der Kantengewichte

betrachte beliebige Reihenfolge der Kanten

$e_1, \dots, e_m \quad (m = |E|)$

Definition: $e_i < e_j$ falls $w(e_i) < w(e_j)$

oder $w(e_i) = w(e_j)$

und $i < j$

$\rightsquigarrow$ mit diesen Vergleichen ist Sortierung immer eindeutig