CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

Graphentheorie
11 – Minimale Spannbäume und -wälder

Dr. Sven Köhler
Rechnernetze und Telematik
Technische Fakultät
Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

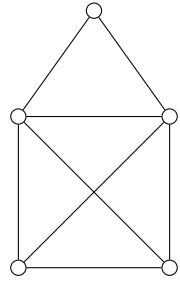
Spannbäume und -wälder

Definition 6.2

Sei $G = (V, E, \gamma)$ ein unger. Graph und $H = (V, E', \gamma')$ ein Partialgraph von G.

Wenn H ein Baum ist, dann heißt H *Spannbaum* (aufspannender Baum) von G. Falls H einen Spannbaum von jeder Zusammenhangskomponente von G enthält, dann heißt H Spannwald (aufspannender Wald) von G.

Konstruktion von Spannbäumen ist durch Breitensuche oder Tiefensuche möglich.



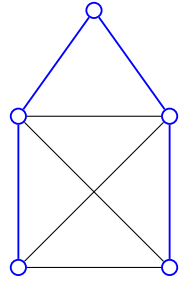
Spannbäume und -wälder

Definition 6.2

Sei $G = (V, E, \gamma)$ ein unger. Graph und $H = (V, E', \gamma')$ ein Partialgraph von G.

Wenn H ein Baum ist, dann heißt H *Spannbaum* (aufspannender Baum) von G. Falls H einen Spannbaum von jeder Zusammenhangskomponente von G enthält, dann heißt H Spannwald (aufspannender Wald) von G.

Konstruktion von Spannbäumen ist durch Breitensuche oder Tiefensuche möglich.



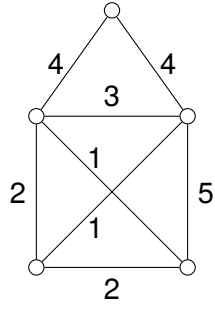
MST – Minimal Spanning Trees

Gegeben:

- ungerichteter zusammenhängender Graph $G = (V, E)$.
- Kostenfunktion $c : E \to \mathbb{R}$

Gesucht:

- Spannbaum T mit minimalen Gesamtkosten $c(T) = \sum_{e \in E(T)} c(e)$



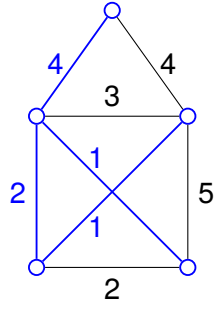
MST – Minimal Spanning Trees

Gegeben:

- ungerichteter zusammenhängender Graph $G = (V, E)$.
- Kostenfunktion $c : E \to \mathbb{R}$

Gesucht:

- Spannbaum T mit minimalen Gesamtkosten $c(T) = \sum_{e \in E(T)} c(e)$



Gesamtkosten: 8

MSF – Minimal Spanning Forest

Gegeben:

- ungerichteter Graph $G = (V, E)$
- Kostenfunktion $c : E \to \mathbb{R}$

Gesucht

- Spannwald $F = (V, E')$ mit minimalen Gesamtkosten $c(F) = \sum_{e \in E(F)} c(e)$

Berechnung MSF:
Berechne MST pro Zusammenhangskomponente

Frage: Wie MST berechnen?

MSF Algorithmen – Vorbereitung

Definition 6.5

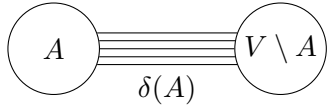
Sei $G = (V, E)$ ein ungerichteter Graph.
Kantenmenge $F \subseteq E$ heißt *fehlerfrei* wenn es MSF F^* von G mit $F \subseteq E(F^*)$ gibt.

Eine Kante e heißt *sicher*, wenn $F \cup \{e\}$ ebenfalls fehlerfrei ist.

Definition 3.18

Sei $G = (V, E)$ ein ungerichteter Graph.
Ein Schnitt (A, B) ist eine Partition $V = A \dot\cup B$, d.h. $B = V \setminus A$.

$\delta_G(A) := \{[a, b] \in E \mid a \in A \ \wedge \ b \in V \setminus A\}$



Grundidee:
Betrachte einen Schnitt, nehme sichere Kante, wiederhole bis MSF fertig.

MSF Algorithmen – Vorbereitung

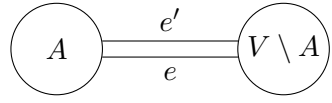
Satz 6.6

Sei (A, B) ein Schnitt von $G = (V, E)$ und $F \subseteq E$ fehlerfrei mit $F \cap \delta(A) = \emptyset$.
Falls $e \in \delta(A)$ geringste Kosten in $\delta(A)$ hat dann ist e sicher.

Beweis.

Es gibt MSF F^* mit $F \subset E(F^*)$ geben, da F fehlerfrei ist.
Sei $e \in \delta(A)$ Kante mit geringsten Kosten.

- Fall 1: $e \in E(F^*)$. Dann ist e sicher.
- Fall 2: $e \notin E(F^*)$.
  Füge e zu F^* hinzu. Es entsteht ein Kreis mit Kanten $e, e' \in \delta(A)$.
  $c(e') \leq c(e)$, sonst wäre F^* kein MSF.
  $c(e) \leq c(e')$, da e geringste Kosten hat.
  Entferne e' aus F^*. Ergebnis ist ein MSF.
  Damit ist e sicher.



□

MSF Algorithmen – Vorbereitung

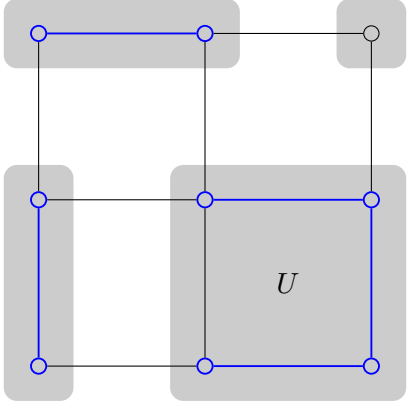
Korollar 6.7

Sei $G = (V, E)$, $F \subseteq E$ fehlerfrei.
Sei $U \subseteq V$ eine Zusammenhangskomponente von $H = (V, F)$.

Falls e eine Kante mit geringsten Kosten aus $\delta_G(U)$ ist, so ist e sicher.

Beweis folgt aus Satz 6.6:

- Schnitt $(U, V \setminus U)$
- Es gilt $\delta(U) \cap F = \emptyset$



MSF – Algorithmus von Kruskal

Algorithmus 6.1 Algorithmus von Kruskal

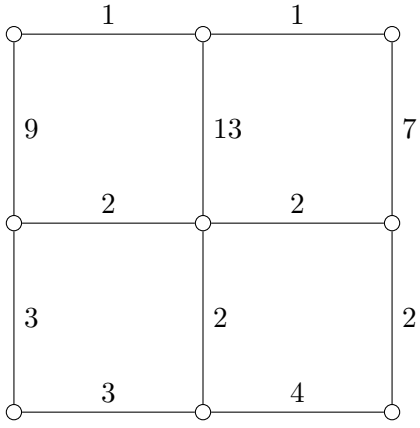
Eingabe: $G = (V, E)$, $c : E \to \mathbb{R}$
Ausgabe: Kantenmenge E_F des MSF

Sortiere Kanten nach Kosten: $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$
$E_F := \emptyset$

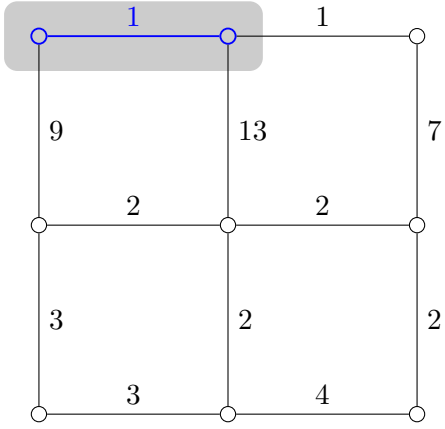
for $i = 1, 2, \ldots, m$ **do**
    $E'_F = E_F \cup \{e_i\}$
    if (V, E'_F) kreisfrei **then**
        $E_F := E'_F$

return E_F

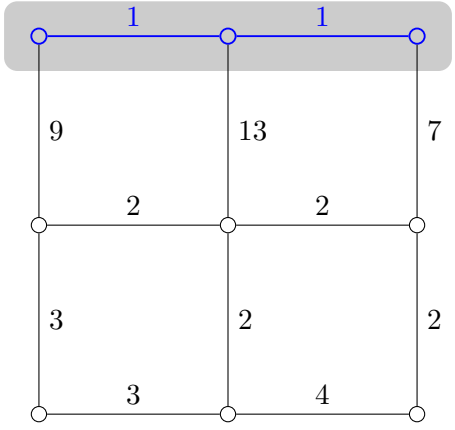
MSF – Beispiel



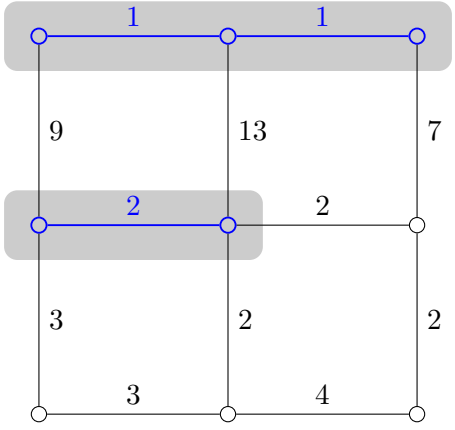
MSF – Beispiel



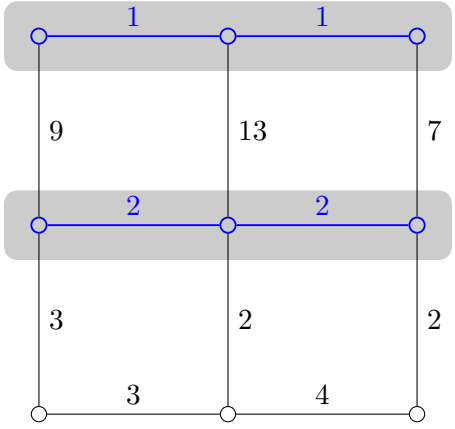
MSF – Beispiel



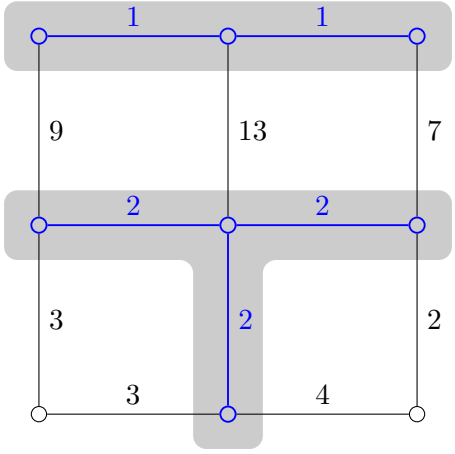
MSF – Beispiel



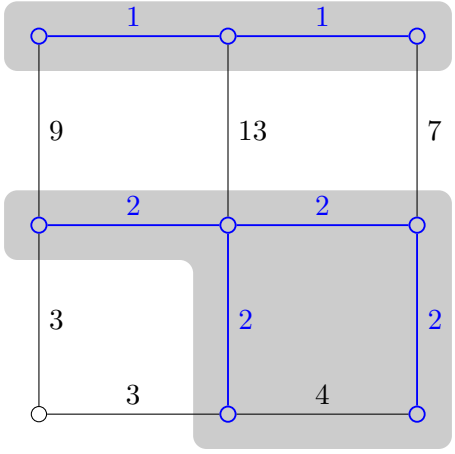
MSF – Beispiel



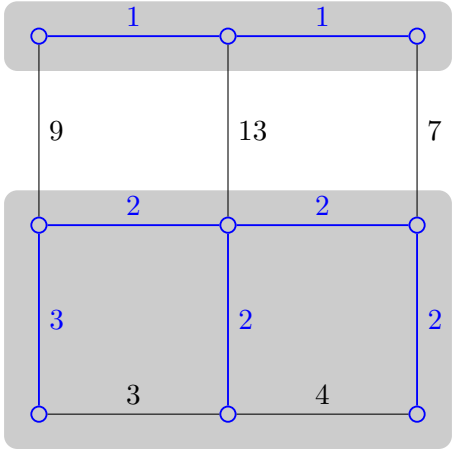
MSF – Beispiel



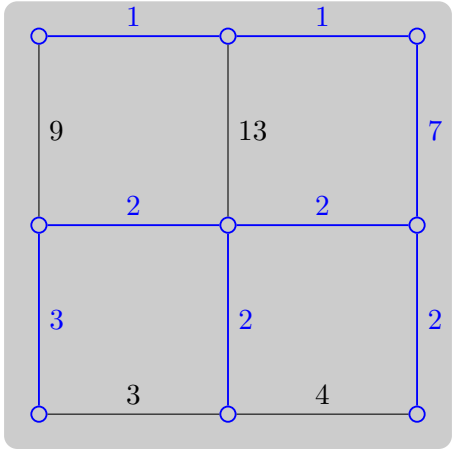
MSF – Beispiel



MSF – Beispiel



MSF – Beispiel



MSF – Algorithmus von Kruskal

Korrektheit:

- Algorithmus sucht Kante e mit geringsten Kosten die keinen Kreis verursacht.
- Wende Korollar 6.7 an auf $H = (V, E_F)$ und $U = ZK_H(v)$ mit $v \in \gamma(e)$
- Daraus folgt, dass die verwendeten Kanten sicher sind

Laufzeit:

- Sortieren der Kanten: $\mathcal{O}(m \log m)$
- Naiver Test auf Kreisfreiheit: sehr teuer!
- Union-Find-Datenstruktur:
  Schneller Test auf Kreisfreiheit
  Gesamtlaufzeit der Schleife: $\mathcal{O}(m \cdot \alpha(n))$

Hinweis: $\alpha(n)$ ist die inverse Ackermann Funktion.
Es gilt $\alpha(n) \leq 4$ für alle $n \leq 10^{684}$.
Anzahl der Atome im Universum: ca. 10^{80}

MST – Algorithmus von Prim

Algorithmus 6.5 Algorithmus von Prim

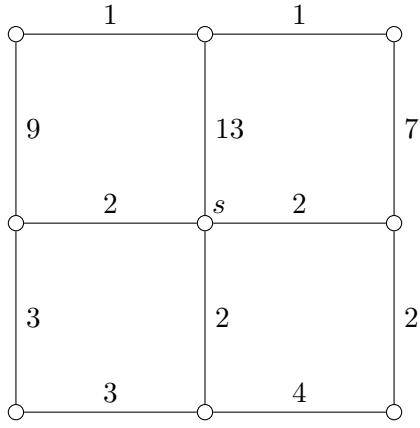
Eingabe: $G = (V, E)$ zusammenhängend, $c : E \to \mathbb{R}$
Ausgabe: Kantenmenge E_T des MST

Wähle $s \in V$ beliebig
$E_T := \emptyset$
$S := \{s\}$

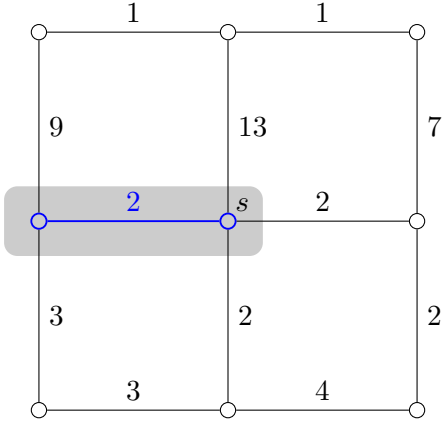
while $S \neq V$ **do**
    Wähle Kante $e = [u, v] \in \delta(S)$ mit $c(e)$ minimal
    $E_T = E_T \cup \{e\}$
    $S = S \cup \{u, v\}$

return E_T

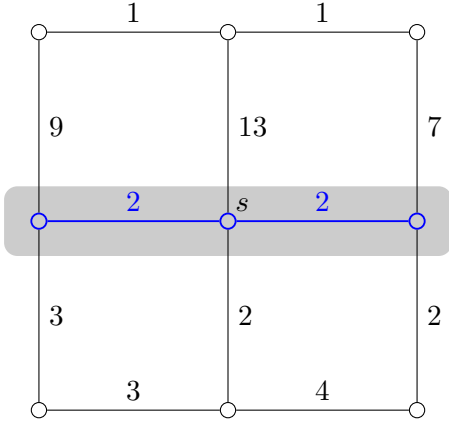
MST – Beispiel



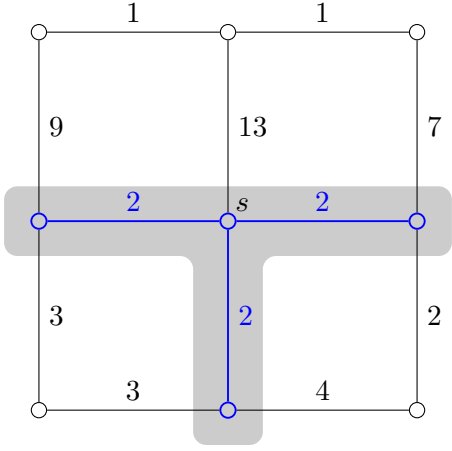
MST – Beispiel



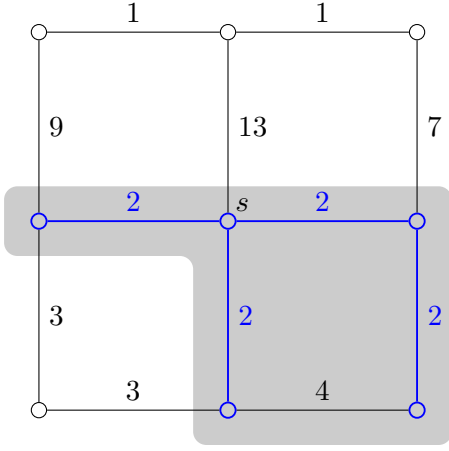
MST – Beispiel



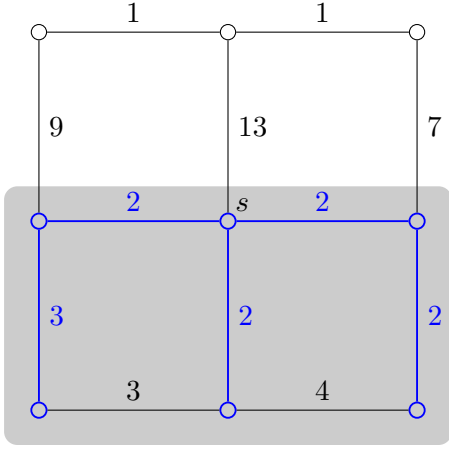
MST – Beispiel



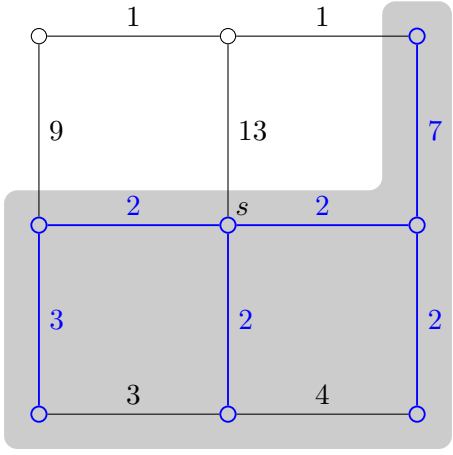
MST – Beispiel



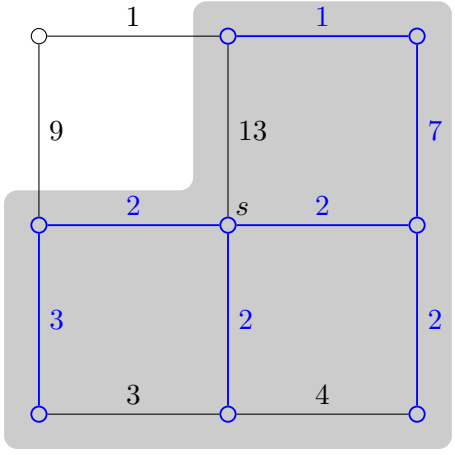
MST – Beispiel



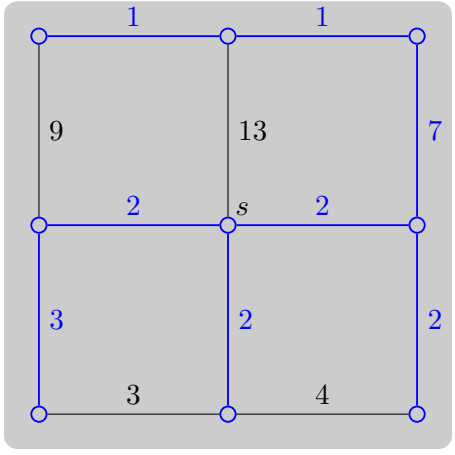
MST – Beispiel



MST – Beispiel



MST – Beispiel



MST – Algorithmus von Prim

Korrektheit:

- $\delta(S)$ enthält keine Kanten die Knoten aus S verbinden.
  $\Rightarrow E_T$ ist stets kreisfrei
- Schleife terminiert erst wenn $S = V$.
  $\Rightarrow E_T$ muss Spannbaum sein
- $\delta(S) \neq \emptyset$ wenn $S \subset V$.
  $\Rightarrow S$ wächst in jedem Schleifendurchlauf
  $\Rightarrow$ Schleife terminiert nach n Durchläufen
- Wende Korollar 6.7 auf Zusammenhangskomponente S an.
  $\Rightarrow E_T$ ist stets fehlerfrei
  $\Rightarrow E_T$ ist am Ende ein MST

MST – Algorithmus von Prim

Laufzeit:

- Schleife wird n mal durchlaufen.
- Problem: Wie Kante aus $\delta(S)$ mit geringsten Kosten finden?
- Lösung: Speichere $\delta(S)$ in einer Priority Queue
  - Kanten hinzufügen: Laufzeit $\mathcal{O}(\log m)$
  - Minimum entfernen: Laufzeit $\mathcal{O}(\log m)$
- In jeder Iteration wird nur ein Knoten v zu S hinzugefügt
  - $\delta(S)$ ändert sich nur minimal
  - Kanten von v zu S fallen weg
  - Kanten von v zu $V \setminus S$ kommen hinzu
  - Markiere Knoten die in S sind
  - Kanten werden maximal zweimal angeschaut
- Gesamtlaufzeit: $\mathcal{O}(m \log m)$

Algorithmus von Borůvka – Vorbereitungen

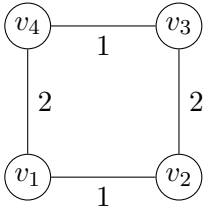
Sei $G = (V, E)$ mit Kostenfunktion $c : E \to \mathbb{R}$

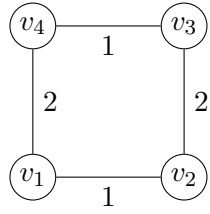
Algorithmus benötigt Annahme:

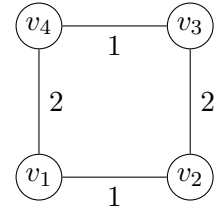
- $\forall K \subseteq E \; \exists! e \in K : c(e) = \min\{c(e') \mid e' \in K\}$
- Für alle $K \subseteq E$ ist die Kante mit geringsten Kosten eindeutig bestimmt

Wir stellen eindeutige Sortierung der Kosten her:

- Kosten leicht verändern
- nach Identitäten der Endknoten sortieren
- nach Identitäten der Kanten sortieren







Algorithmus von Borůvka – Vorbereitungen

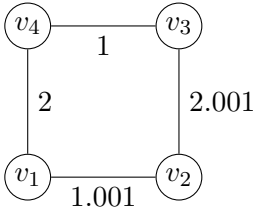
Sei $G = (V, E)$ mit Kostenfunktion $c : E \to \mathbb{R}$

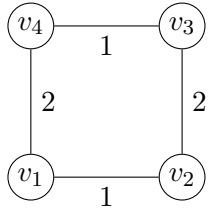
Algorithmus benötigt Annahme:

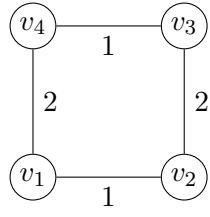
- $\forall K \subseteq E \; \exists! e \in K : c(e) = \min\{c(e') \mid e' \in K\}$
- Für alle $K \subseteq E$ ist die Kante mit geringsten Kosten eindeutig bestimmt

Wir stellen eindeutige Sortierung der Kosten her:

- Kosten leicht verändern
- nach Identitäten der Endknoten sortieren
- nach Identitäten der Kanten sortieren







Algorithmus von Borůvka – Vorbereitungen

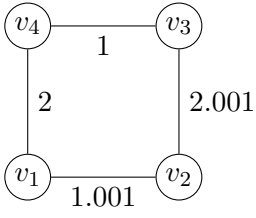
Sei $G = (V, E)$ mit Kostenfunktion $c : E \to \mathbb{R}$

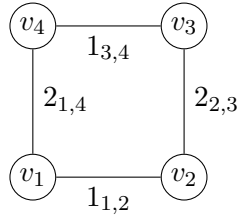
Algorithmus benötigt Annahme:

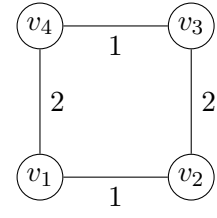
- $\forall K \subseteq E \; \exists! e \in K : c(e) = \min\{c(e') \mid e' \in K\}$
- Für alle $K \subseteq E$ ist die Kante mit geringsten Kosten eindeutig bestimmt

Wir stellen eindeutige Sortierung der Kosten her:

- Kosten leicht verändern
- nach Identitäten der Endknoten sortieren
- nach Identitäten der Kanten sortieren







Algorithmus von Borůvka – Vorbereitungen

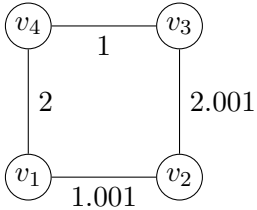
Sei $G = (V, E)$ mit Kostenfunktion $c : E \to \mathbb{R}$

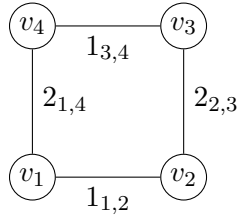
Algorithmus benötigt Annahme:

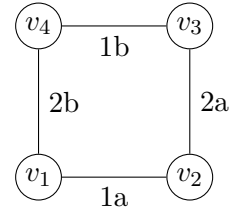
- $\forall K \subseteq E \; \exists! e \in K : c(e) = \min\{c(e') \mid e' \in K\}$
- Für alle $K \subseteq E$ ist die Kante mit geringsten Kosten eindeutig bestimmt

Wir stellen eindeutige Sortierung der Kosten her:

- Kosten leicht verändern
- nach Identitäten der Endknoten sortieren
- nach Identitäten der Kanten sortieren







Algorithmus von Borůvka

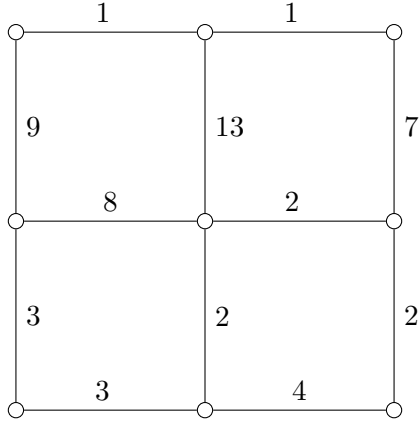
Algorithmus 6.7 Algorithmus von Borůvka

Eingabe: $G = (V, E)$ zusammenhängend, Kostenfunktion $c : E \to \mathbb{R}$
Ausgabe: Kantenmenge E_T des MST

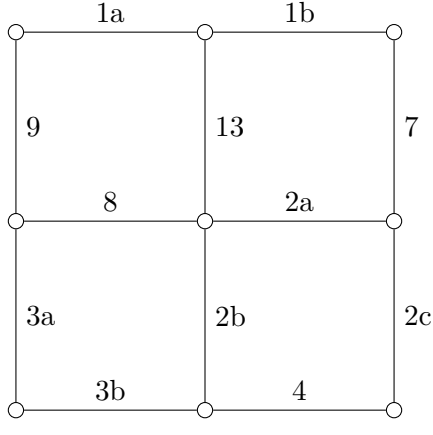
$E_T := \emptyset$
while (V, E_T) hat mehrere Zusammenhangskomponenten $V_1, V_2, \ldots, V_p$ **do**
  for $i := 1, 2, \ldots, p$ **do**
    Bestimme $e_i \in \delta_G(V_i)$, so dass $c(e_i)$ minimal
  $E_T := E_T \cup \{e_1, e_2, \ldots, e_p\}$

Hinweis: Algorithmus kann parallelisiert werden,
ist also für parallele und verteilte Systeme interessant.

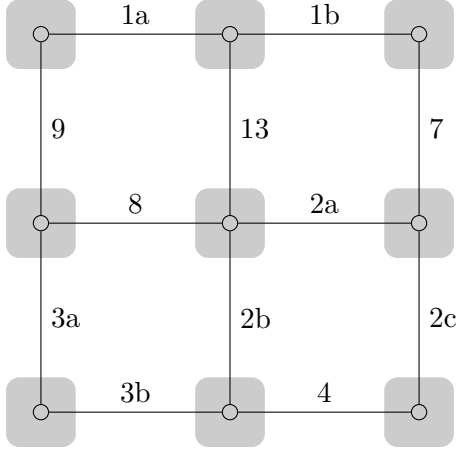
Algorithmus von Borůvka – Beispiel



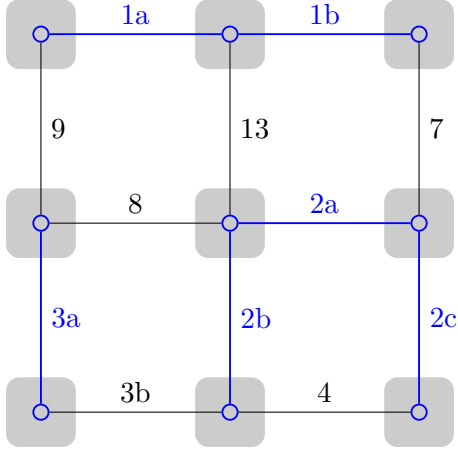
Algorithmus von Borůvka – Beispiel



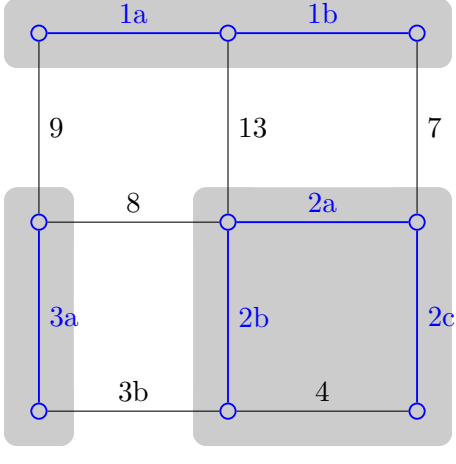
Algorithmus von Borůvka – Beispiel



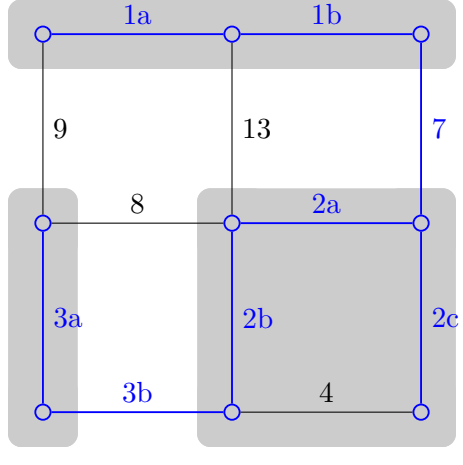
Algorithmus von Borůvka – Beispiel



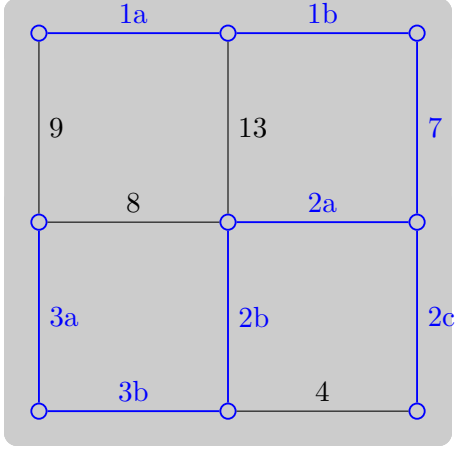
Algorithmus von Borůvka – Beispiel



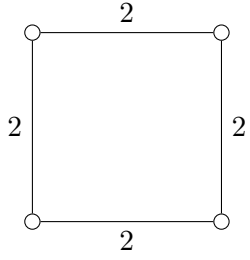
Algorithmus von Borůvka – Beispiel

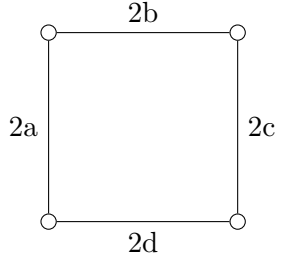


Algorithmus von Borůvka – Beispiel

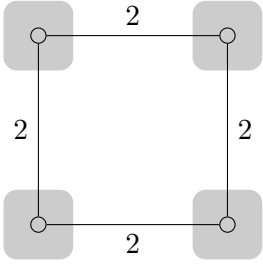


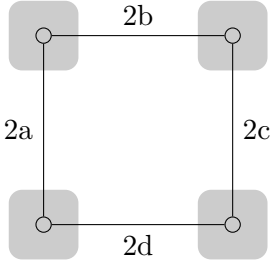
Alg. von Borůvka – Uneindeutige Kantengewichte



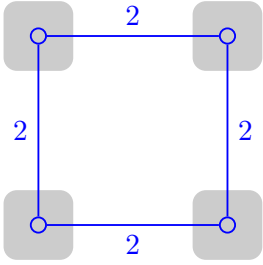


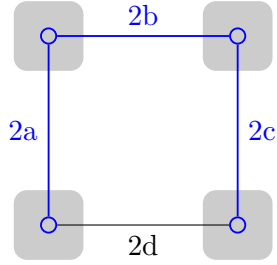
Alg. von Borůvka – Uneindeutige Kantengewichte



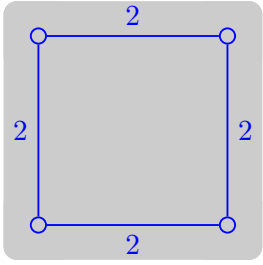


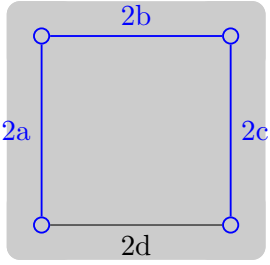
Alg. von Borůvka – Uneindeutige Kantengewichte





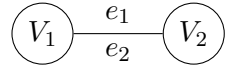
Alg. von Borůvka – Uneindeutige Kantengewichte





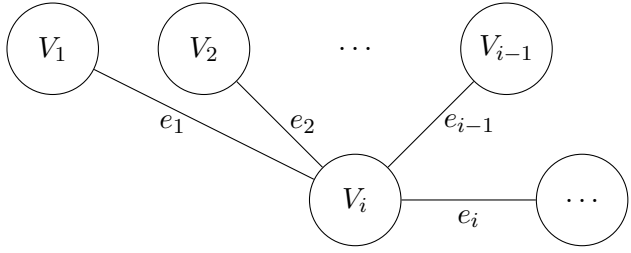
Algorithmus von Borůvka – Korrektheit

Betrachte eine Iteration der while-Schleife.
Entferne doppelte Kanten aus $e_1, \ldots, e_p$ und zugehörige ZK aus $V_1, \ldots, V_p$.
Benenne Kanten um so dass $c(e_1) < c(e_2) < \ldots < c(e_p)$.



Beweis per Induktion über i.
Annahme: $F = E_T \cup \{e_1, e_2, \ldots, e_{i-1}\}$ ist fehlerfrei.

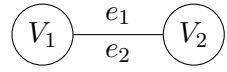
Behauptung: V_i ist Zusammenhangskomponente in (V, F).



Wende Korollar 6.7 an: $e_i \in \delta(V_i)$ ist sicher und $F \cup \{e_i\}$ ist fehlerfrei.

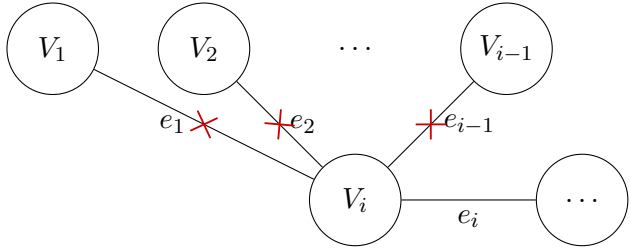
Algorithmus von Borůvka – Korrektheit

Betrachte eine Iteration der while-Schleife.
Entferne doppelte Kanten aus $e_1, \ldots, e_p$ und zugehörige ZK aus $V_1, \ldots, V_p$.
Benenne Kanten um so dass $c(e_1) < c(e_2) < \ldots < c(e_p)$.



Beweis per Induktion über i.
Annahme: $F = E_T \cup \{e_1, e_2, \ldots, e_{i-1}\}$ ist fehlerfrei.

Behauptung: V_i ist Zusammenhangskomponente in (V, F).

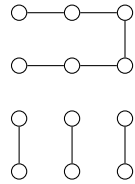


Wende Korollar 6.7 an: $e_i \in \delta(V_i)$ ist sicher und $F \cup \{e_i\}$ ist fehlerfrei.

Algorithmus von Borůvka – Laufzeit

Laufzeit:

- Algo. startet mit n Zusammenahngskomponenten ($E_T = \emptyset$)
- In jeder Iteration wird pro ZK eine Kante gewählt
- Kanten können mehrfach gewählt werden
- Best-Case: alle ZK werden miteinander verbunden
- Worst-Case: immer zwei ZK werden miteinander verbunden



- Am Anfang sind es n Zusammenahngskomponenten
- Die while-Schleife benötigt höchstens $\log n$ Iterationen
- Innere Schleife hat Laufzeit $\mathcal{O}(n + m) \subseteq \mathcal{O}(m)$

Gesamtlaufzeit $\mathcal{O}(m \log n)$