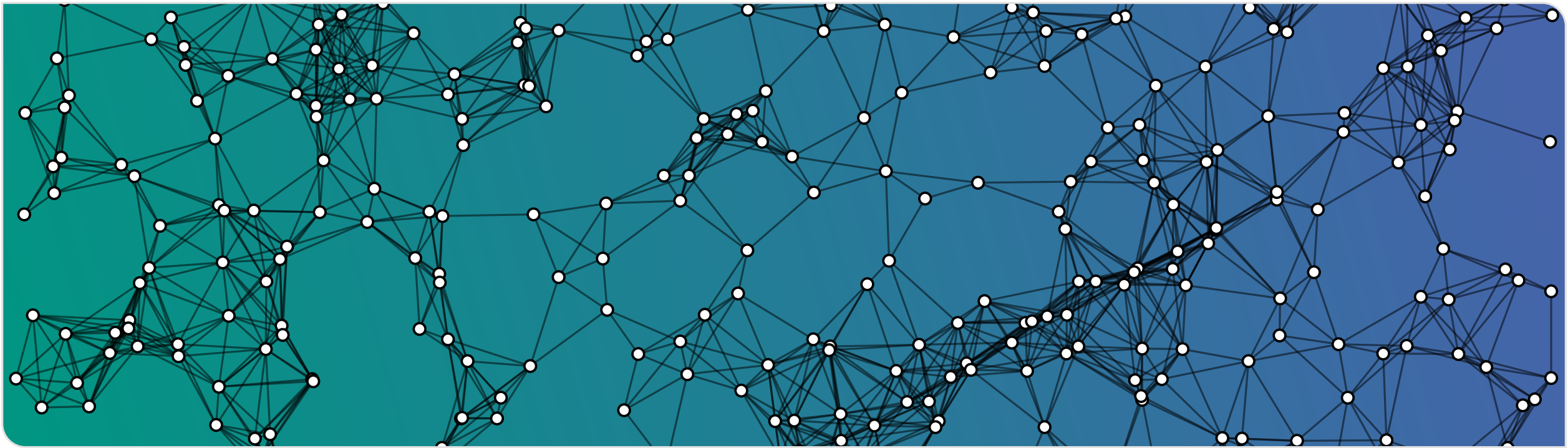
Karlsruher Institut für Technologie

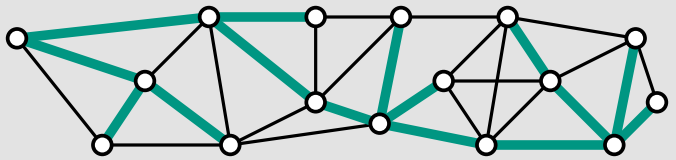
Algorithmen 1

Minimale Spannbäume



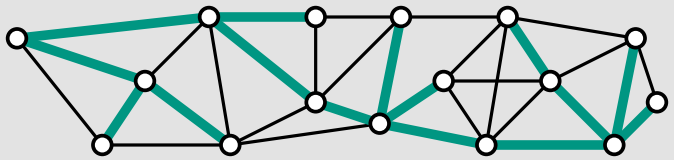
Problemstellung MST

Definition
Sei $G = (V, E)$ ein Graph. Ein Baum auf der selben Knotenmenge $T = (V, E_T)$ mit $E_T \subseteq E$ heißt **Spannbaum** von G.

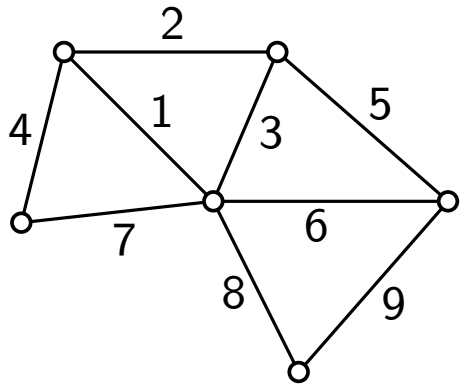


Problemstellung MST

Definition
Sei $G = (V, E)$ ein Graph. Ein Baum auf der selben Knotenmenge $T = (V, E_T)$ mit $E_T \subseteq E$ heißt **Spannbaum** von G.



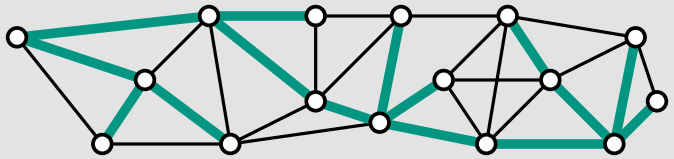
Problem: Minimaler Spannbaum (MST)
Sei $G = (V, E)$ ein Graph mit Kantengewichten $w\colon E \to \mathbb{Z}$. Finde einen Spannbaum $T = (V, E_T)$, sodass die Summe der Gewichte in E_T minimal ist.



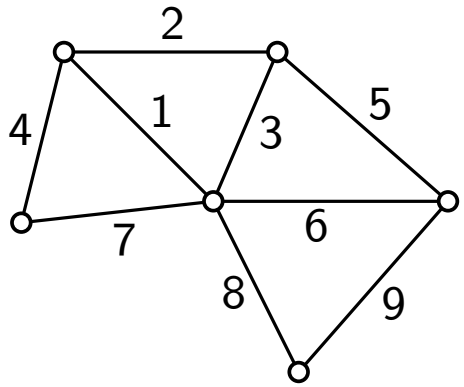
app.klicker.uzh.ch/join/algo1

Problemstellung MST

Definition
Sei $G = (V, E)$ ein Graph. Ein Baum auf der selben Knotenmenge $T = (V, E_T)$ mit $E_T \subseteq E$ heißt **Spannbaum** von G.



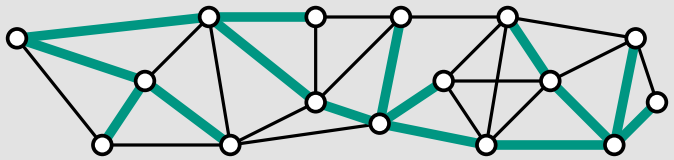
Problem: Minimaler Spannbaum (MST)
Sei $G = (V, E)$ ein Graph mit Kantengewichten $w\colon E \to \mathbb{Z}$. Finde einen Spannbaum $T = (V, E_T)$, sodass die Summe der Gewichte in E_T minimal ist.



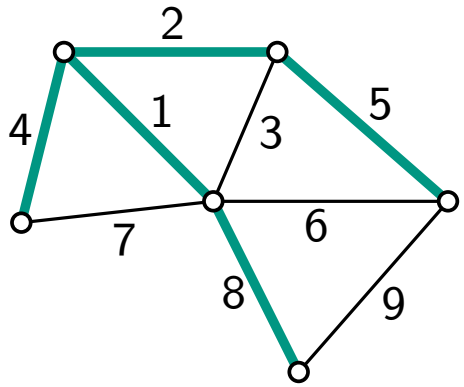
Wie schwer ist der minimale Spannbaum?

Problemstellung MST

Definition
Sei $G = (V, E)$ ein Graph. Ein Baum auf der selben Knotenmenge $T = (V, E_T)$ mit $E_T \subseteq E$ heißt **Spannbaum** von G.



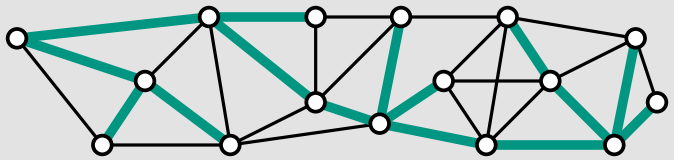
Problem: Minimaler Spannbaum (MST)
Sei $G = (V, E)$ ein Graph mit Kantengewichten $w\colon E \to \mathbb{Z}$. Finde einen Spannbaum $T = (V, E_T)$, sodass die Summe der Gewichte in E_T minimal ist.



Gewicht: $1 + 2 + 4 + 5 + 8 = 20$

Problemstellung MST

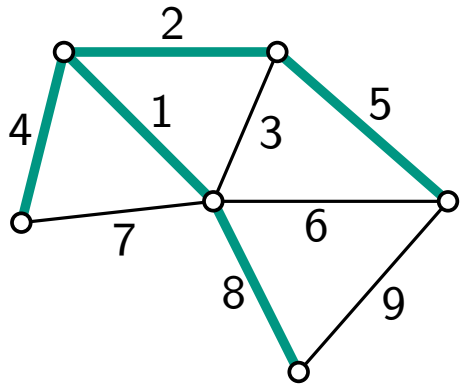
Definition
Sei $G = (V, E)$ ein Graph. Ein Baum auf der selben Knotenmenge $T = (V, E_T)$ mit $E_T \subseteq E$ heißt **Spannbaum** von G.



Problem: Minimaler Spannbaum (MST)
Sei $G = (V, E)$ ein Graph mit Kantengewichten $w\colon E \to \mathbb{Z}$. Finde einen Spannbaum $T = (V, E_T)$, sodass die Summe der Gewichte in E_T minimal ist.

Vereinfachende Annahme

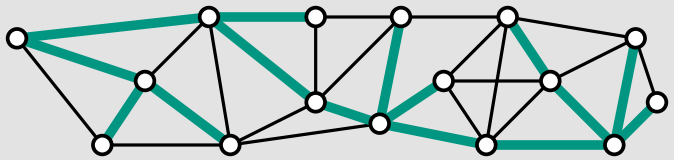
- eindeutige Gewichte, also $w(e_1) \neq w(e_2)$ wenn $e_1 \neq e_2$
- kein echter Unterschied, vereinfacht aber ein paar Dinge



Gewicht: $1 + 2 + 4 + 5 + 8 = 20$

Problemstellung MST

Definition
Sei $G = (V, E)$ ein Graph. Ein Baum auf der selben Knotenmenge $T = (V, E_T)$ mit $E_T \subseteq E$ heißt **Spannbaum** von G.



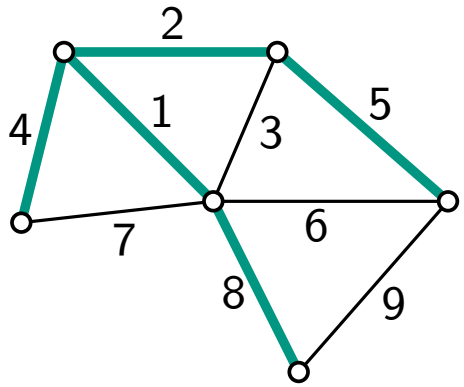
Problem: Minimaler Spannbaum (MST)
Sei $G = (V, E)$ ein Graph mit Kantengewichten $w\colon E \to \mathbb{Z}$. Finde einen Spannbaum $T = (V, E_T)$, sodass die Summe der Gewichte in E_T minimal ist.

Vereinfachende Annahme

- eindeutige Gewichte, also $w(e_1) \neq w(e_2)$ wenn $e_1 \neq e_2$
- kein echter Unterschied, vereinfacht aber ein paar Dinge

Plan im Folgenden

- gewinne ein paar strukturelle Einsichten über MSTs
- leite daraus zwei Algorithmen ab

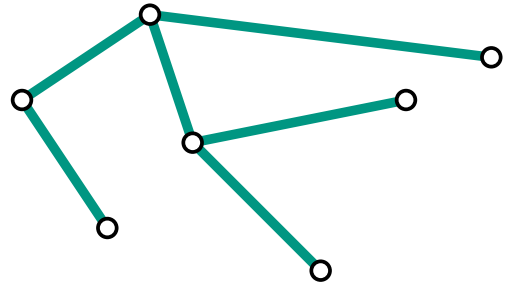


Gewicht: $1 + 2 + 4 + 5 + 8 = 20$

Grundsätzliche Anmerkungen zu Bäumen und Schnitten

Bäume sind charakterisiert durch

- sie sind kreisfrei
- sie sind zusammenhängend
- sie haben $n - 1$ Kanten

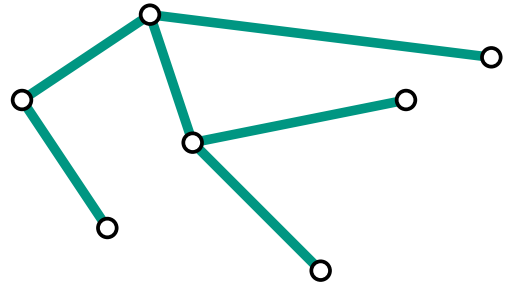


Grundsätzliche Anmerkungen zu Bäumen und Schnitten

Bäume sind charakterisiert durch

- sie sind kreisfrei
- sie sind zusammenhängend
- sie haben $n - 1$ Kanten

} je zwei der Eigenschaften implizieren die dritte

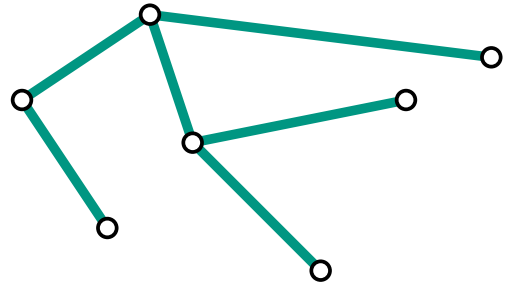


Grundsätzliche Anmerkungen zu Bäumen und Schnitten

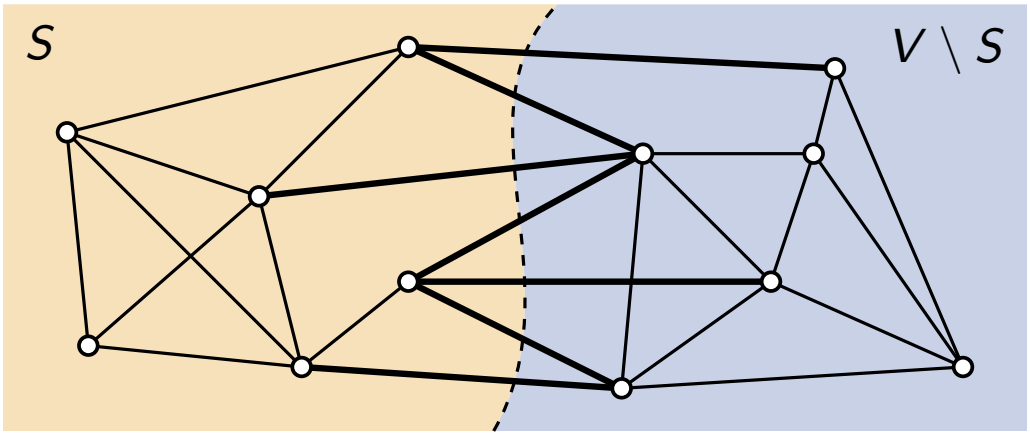
Bäume sind charakterisiert durch

- sie sind kreisfrei
- sie sind zusammenhängend
- sie haben $n - 1$ Kanten

} je zwei der Eigenschaften implizieren die dritte



Definition
Sei $G = (V, E)$ ein Graph. Ein **Schnitt** ist eine Zerlegung von V in zwei nicht-leere Teilmengen S und $V \setminus S$. Eine Kante zwischen einem Knoten aus S und einem aus $V \setminus S$ heißt **Schnittkante**.



Beweisbar essentielle Kanten

Idee

- zeige, dass gewissen Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

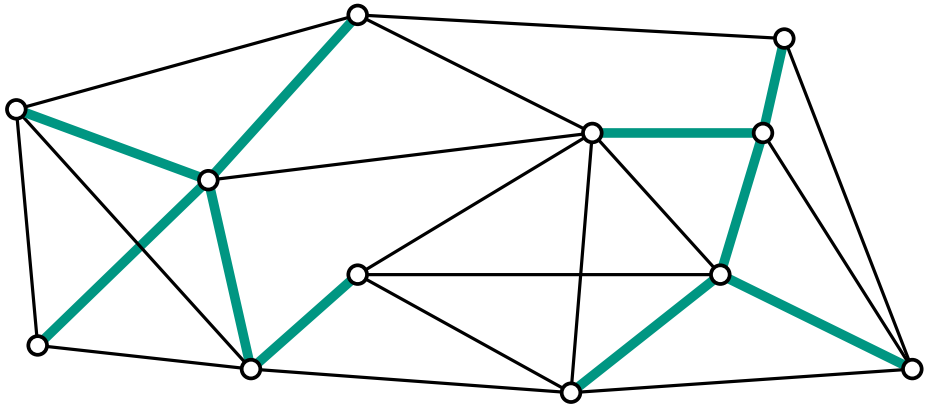
Beweisbar essentielle Kanten

Idee

- zeige, dass gewissen Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Kurz vorm Ziel: Welche Kante ist essentiell?

- Annahme: bereits $n - 2$ Kanten gewählt (grün)



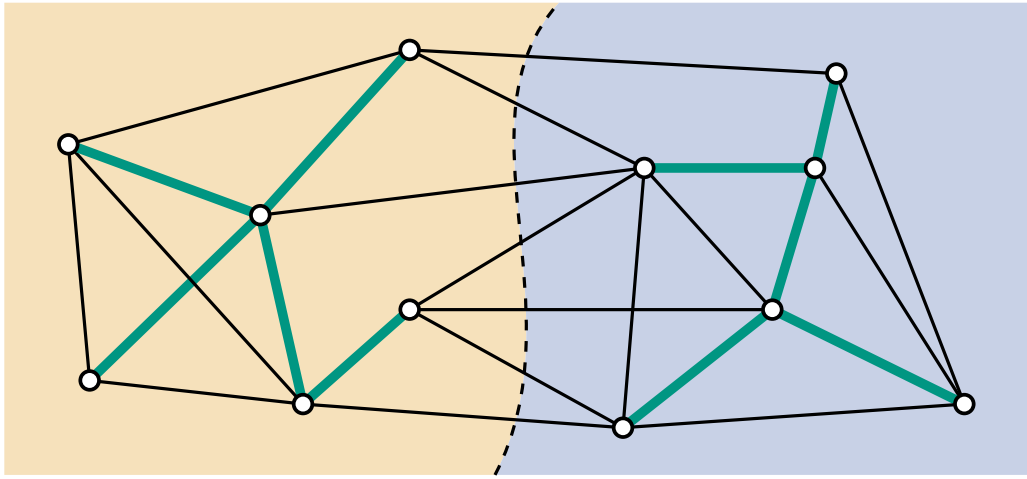
Beweisbar essentielle Kanten

Idee

- zeige, dass gewissen Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Kurz vorm Ziel: Welche Kante ist essentiell?

- Annahme: bereits $n - 2$ Kanten gewählt (grün)
- die zwei Teilbäume induzieren einen Schnitt



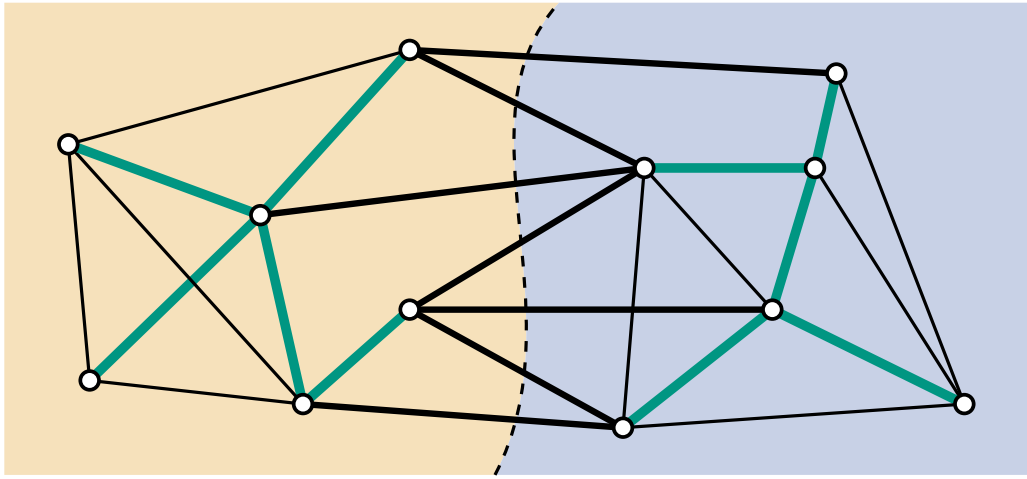
Beweisbar essentielle Kanten

Idee

- zeige, dass gewissen Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n-1$ Schritten bilden die $n-1$ gewählten Kanten den MST

Kurz vorm Ziel: Welche Kante ist essentiell?

- Annahme: bereits $n-2$ Kanten gewählt (grün)
- die zwei Teilbäume induzieren einen Schnitt
- Auswahl der letzten Kante (Schritt $n-1$):
  - wir müssen eine Schnittkante wählen
  - jede Schnittkante liefert einen Baum



Beweisbar essentielle Kanten

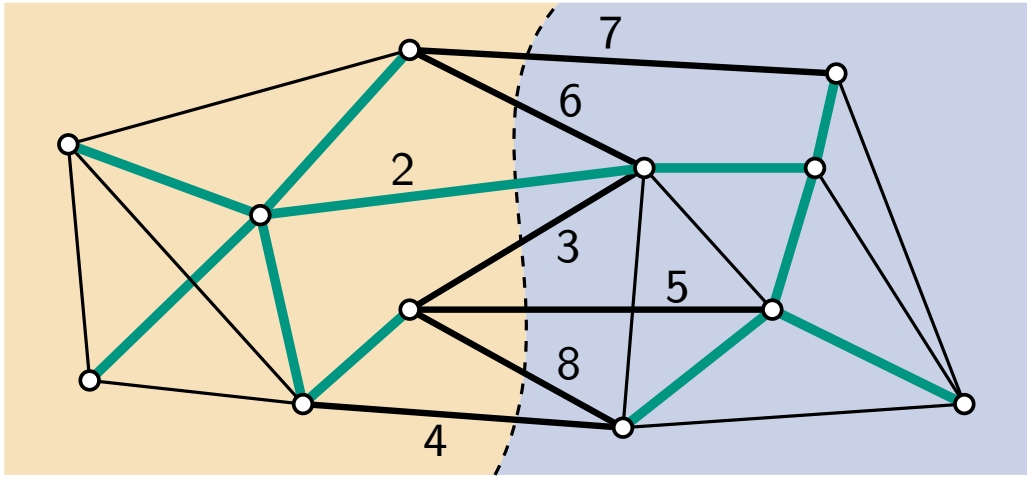
Idee

- zeige, dass gewissen Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Kurz vorm Ziel: Welche Kante ist essentiell?

- Annahme: bereits $n - 2$ Kanten gewählt (grün)
- die zwei Teilbäume induzieren einen Schnitt
- Auswahl der letzten Kante (Schritt $n - 1$):
  - wir müssen eine Schnittkante wählen
  - jede Schnittkante liefert einen Baum

$\Rightarrow$ minimale Schnittkante muss gewählt werden



Beweisbar essentielle Kanten

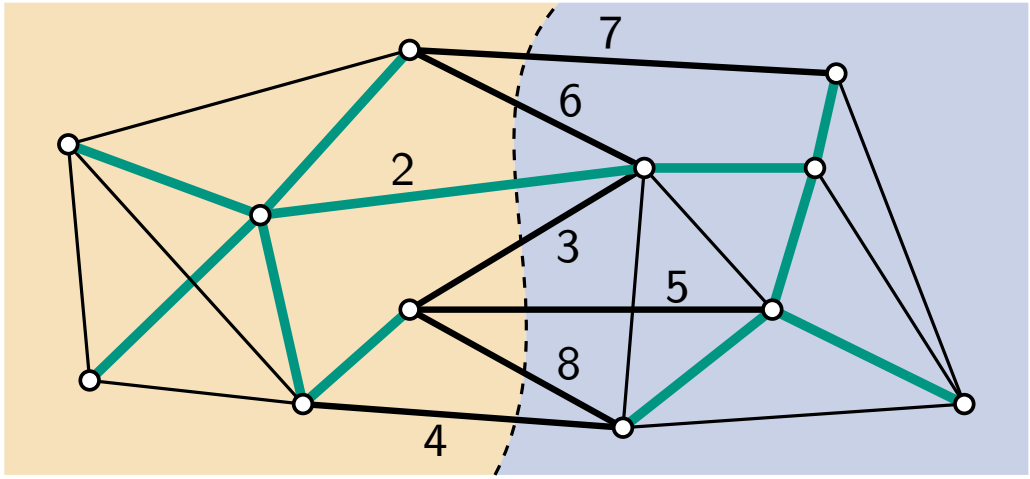
Idee

- zeige, dass gewissen Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Kurz vorm Ziel: Welche Kante ist essentiell?

- Annahme: bereits $n - 2$ Kanten gewählt (grün)
- die zwei Teilbäume induzieren einen Schnitt
- Auswahl der letzten Kante (Schritt $n - 1$):
  - wir müssen eine Schnittkante wählen
  - jede Schnittkante liefert einen Baum

$\Rightarrow$ minimale Schnittkante muss gewählt werden



Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)

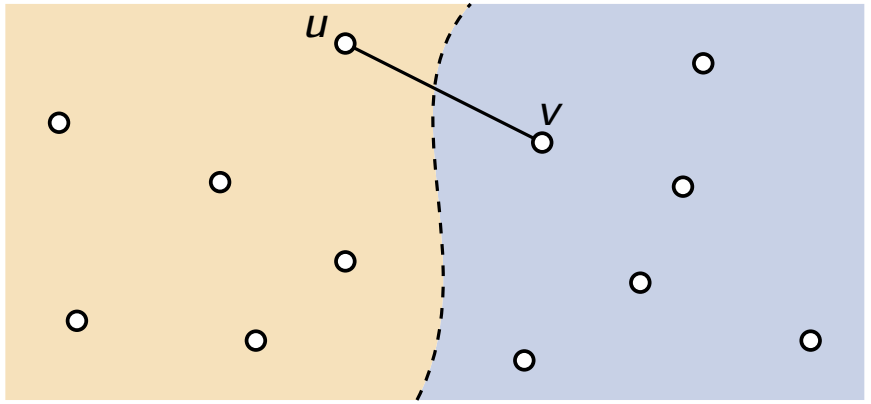
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Minimale Schnittkanten

Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Beweis

- betrachte Schnitt mit minimaler Schnittkante $\{u, v\}$



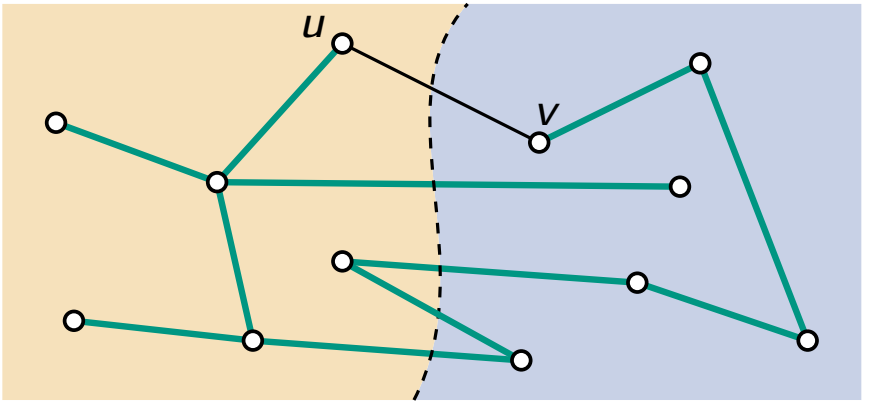
Minimale Schnittkanten

Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Beweis

- betrachte Schnitt mit minimaler Schnittkante $\{u, v\}$
- betrachte einen Spannbaum T ohne $\{u, v\}$

Ziel: zeige, dass T kein MST ist



Minimale Schnittkanten

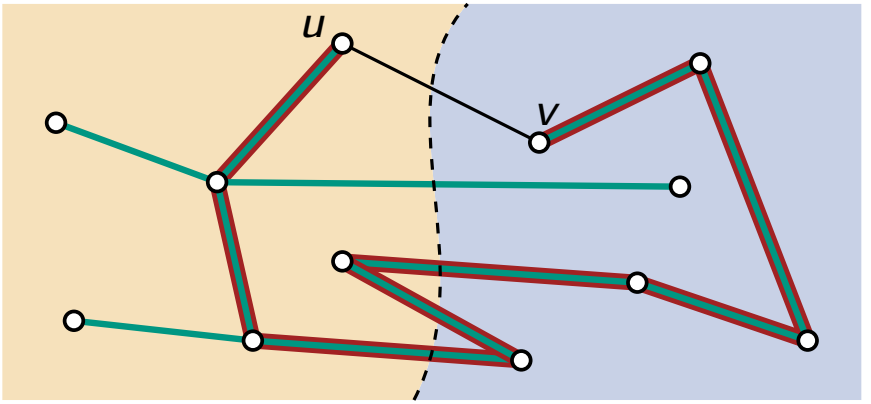
Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Beweis

- betrachte Schnitt mit minimaler Schnittkante $\{u, v\}$
- betrachte einen Spannbaum T ohne $\{u, v\}$

Ziel: zeige, dass T kein MST ist

- betrachte eindeutigen Pfad von u nach v in T

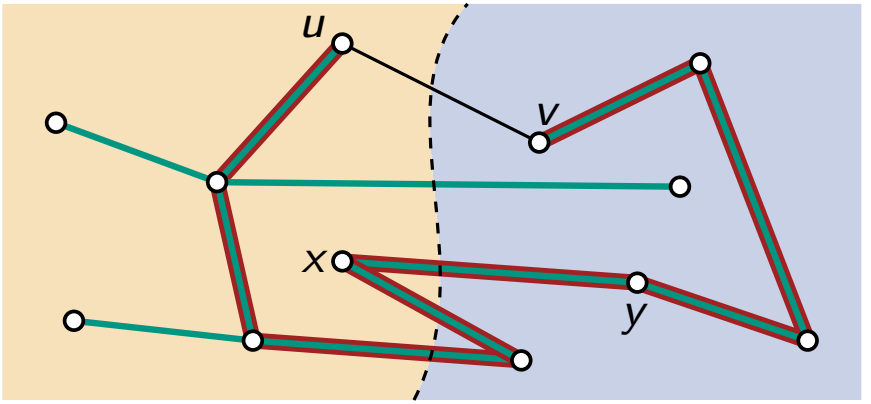


Minimale Schnittkanten

Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Beweis

- betrachte Schnitt mit minimaler Schnittkante $\{u, v\}$
- betrachte einen Spannbaum T ohne $\{u, v\}$



Ziel: zeige, dass T kein MST ist

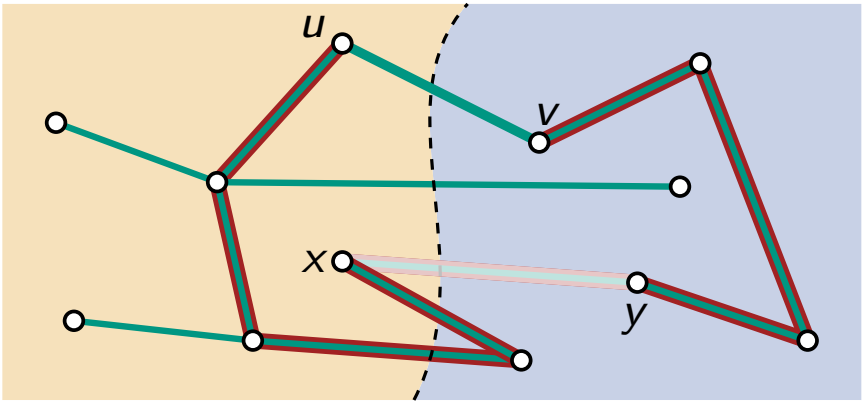
- betrachte eindeutigen Pfad von u nach v in T
- mindestens eine der Pfadkanten $\{x, y\}$ ist auch eine Schnittkante

Minimale Schnittkanten

Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Beweis

- betrachte Schnitt mit minimaler Schnittkante $\{u, v\}$
- betrachte einen Spannbaum T ohne $\{u, v\}$



Ziel: zeige, dass T kein MST ist

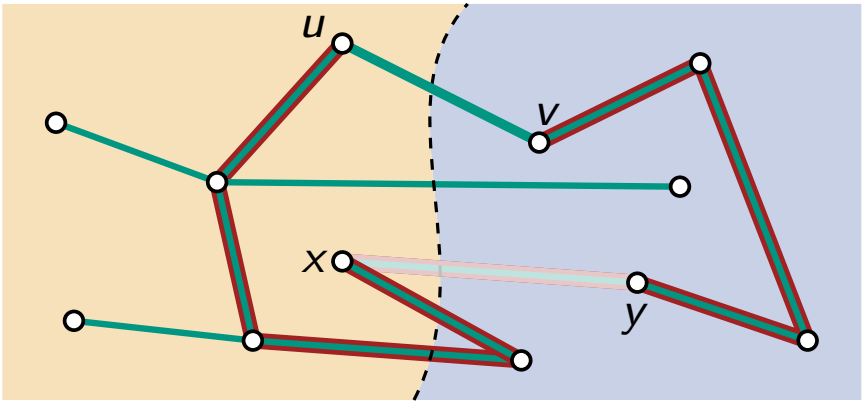
- betrachte eindeutigen Pfad von u nach v in T
- mindestens eine der Pfadkanten $\{x, y\}$ ist auch eine Schnittkante
- neuer Teilgraph T': lösche $\{x, y\}$ und füge $\{u, v\}$ zu T hinzu

Minimale Schnittkanten

Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Beweis

- betrachte Schnitt mit minimaler Schnittkante $\{u, v\}$
- betrachte einen Spannbaum T ohne $\{u, v\}$



Ziel: zeige, dass T kein MST ist

- betrachte eindeutigen Pfad von u nach v in T
- mindestens eine der Pfadkanten $\{x, y\}$ ist auch eine Schnittkante
- neuer Teilgraph T': lösche $\{x, y\}$ und füge $\{u, v\}$ zu T hinzu

Beobachtung

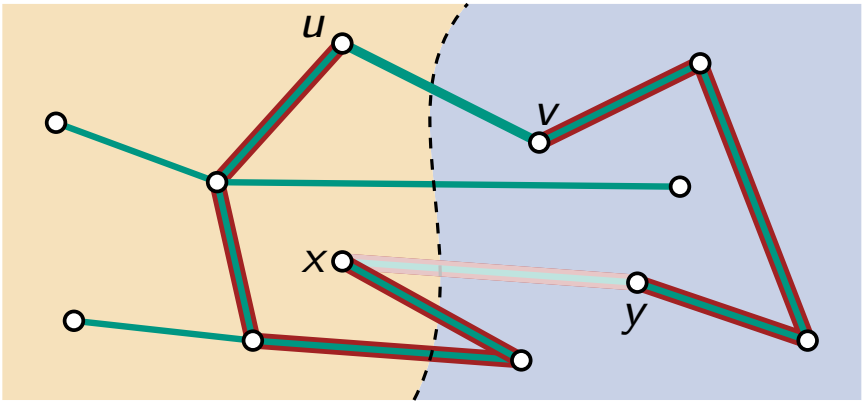
- T' ist ein Baum (kreisfrei, $n - 1$ Kanten)
- T' hat kleineres Gewicht als T ($w(\{u, v\}) < w(\{x, y\})$)

Minimale Schnittkanten

Lemma (Erinnerung: wir nehmen an, dass das Minimum eindeutig ist)
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Beweis

- betrachte Schnitt mit minimaler Schnittkante $\{u, v\}$
- betrachte einen Spannbaum T ohne $\{u, v\}$



Ziel: zeige, dass T kein MST ist

- betrachte eindeutigen Pfad von u nach v in T
- mindestens eine der Pfadkanten $\{x, y\}$ ist auch eine Schnittkante
- neuer Teilgraph T': lösche $\{x, y\}$ und füge $\{u, v\}$ zu T hinzu

Beobachtung

- T' ist ein Baum (kreisfrei, $n - 1$ Kanten)
- T' hat kleineres Gewicht als T $(w(\{u, v\}) < w(\{x, y\}))$

$\Rightarrow T$ ist kein MST

Strukturelle Einsicht $\rightarrow$ Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n-1$ Schritten bilden die $n-1$ gewählten Kanten den MST

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

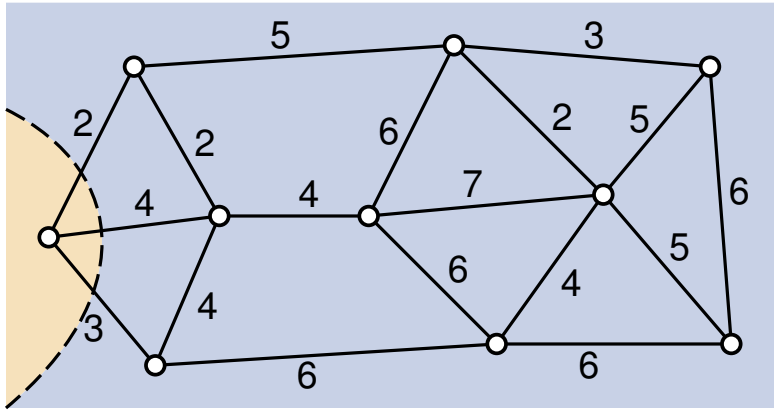
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

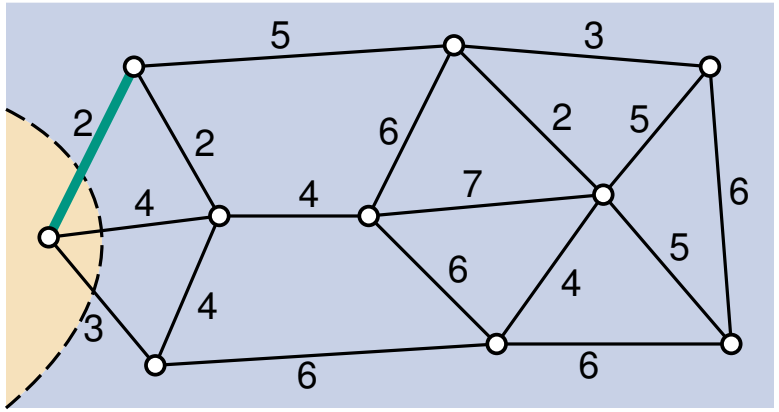
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

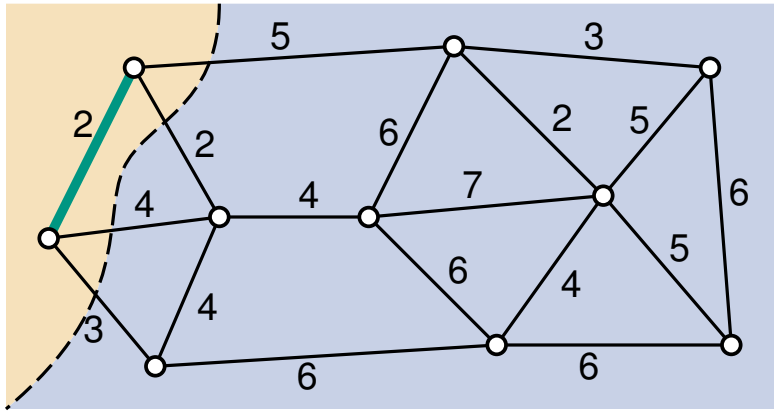
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

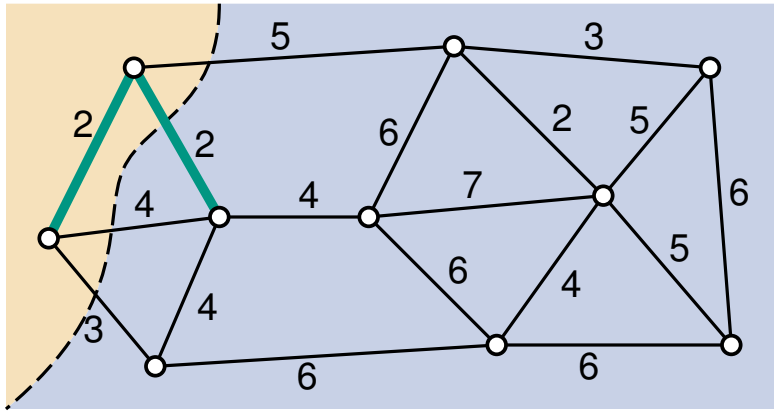
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

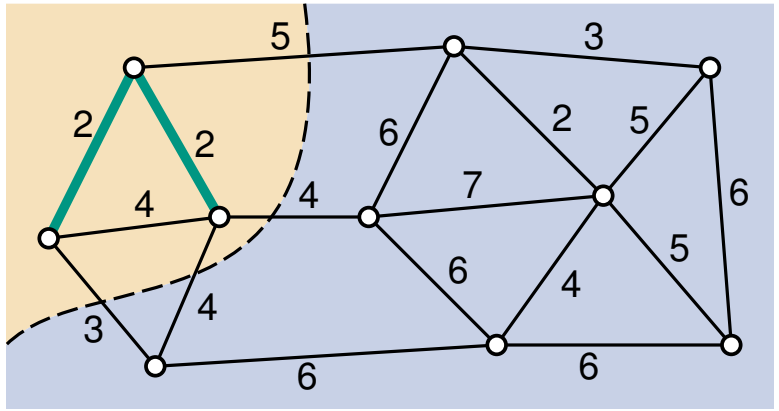
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

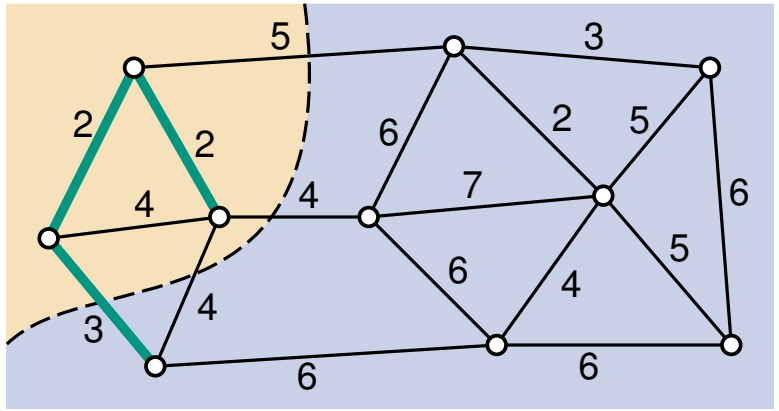
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

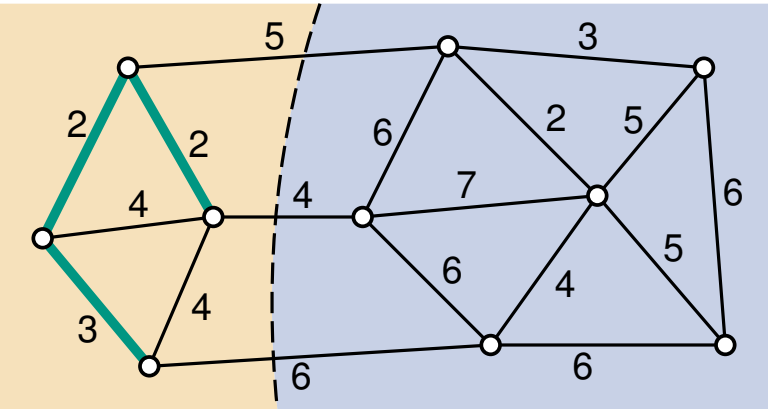
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

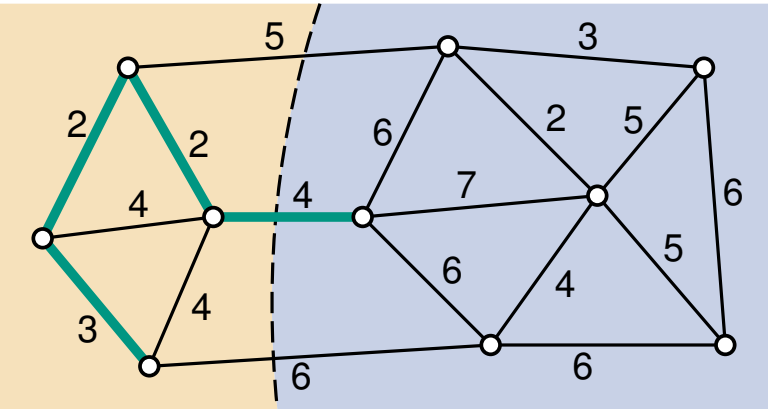
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

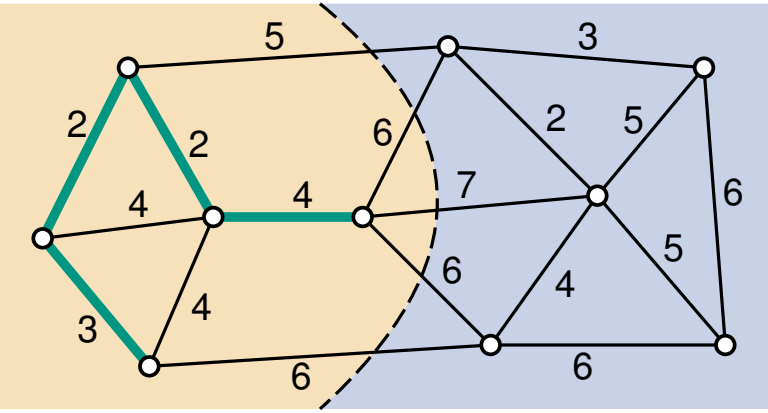
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

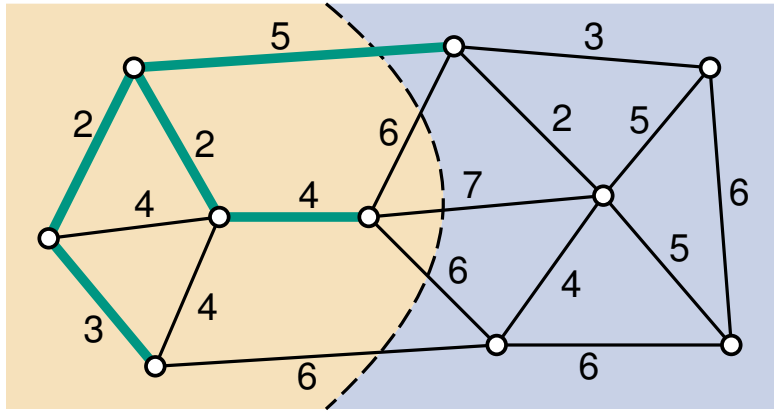
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

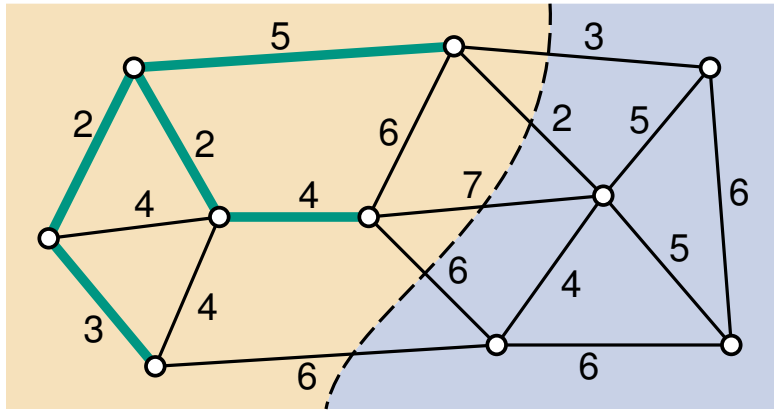
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

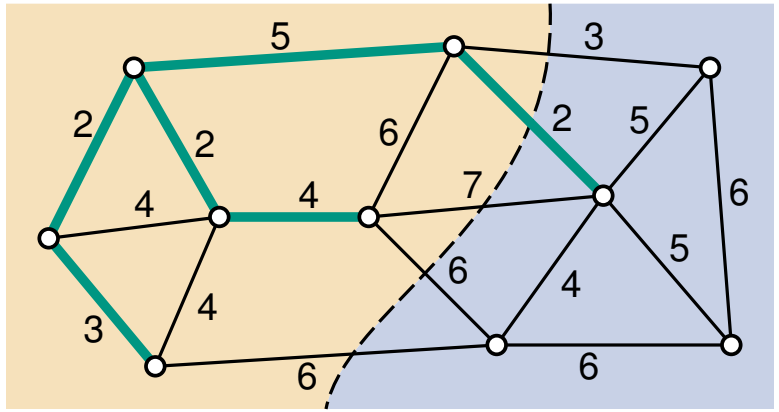
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

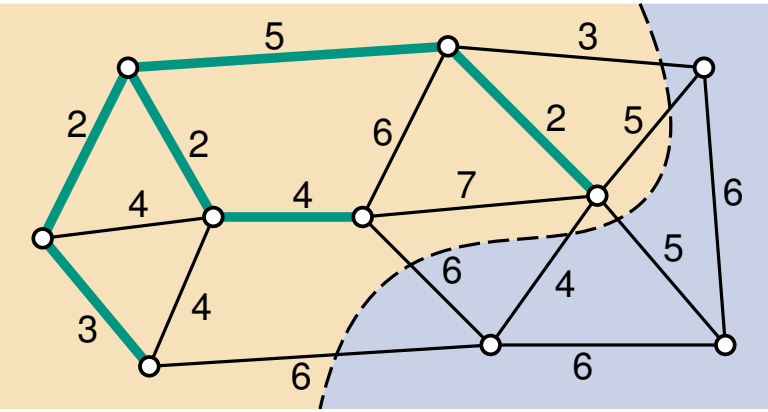
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

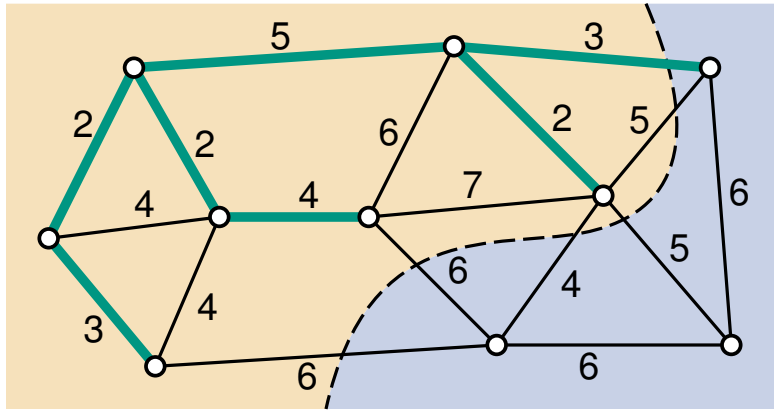
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

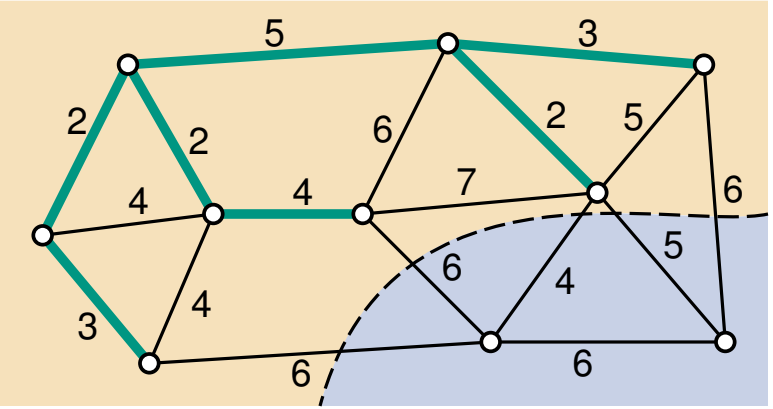
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

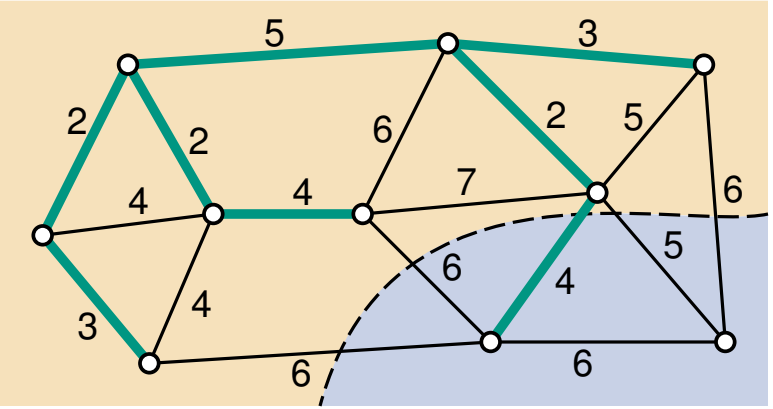
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

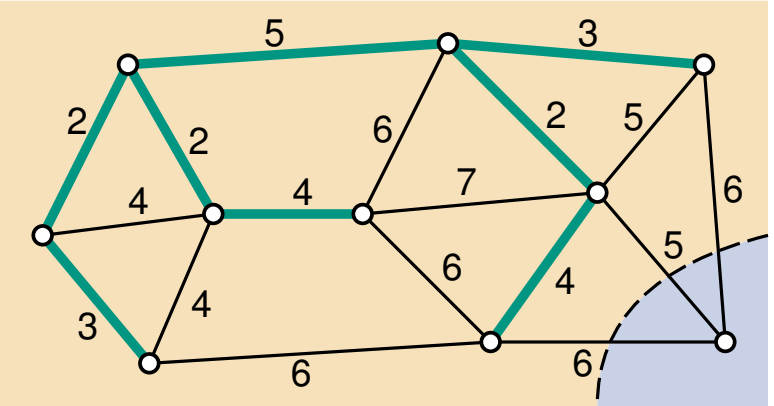
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

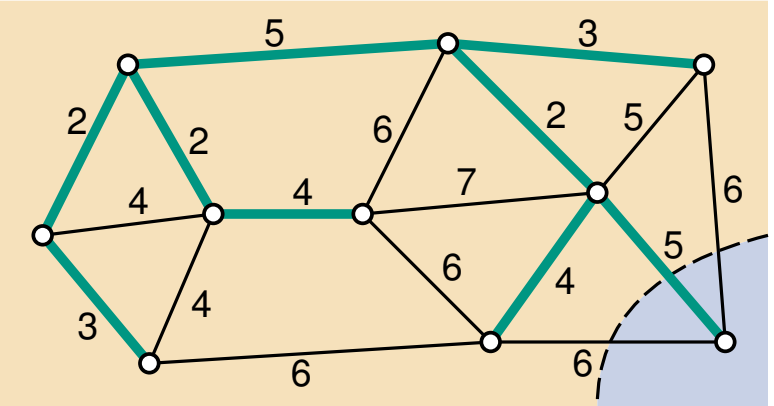
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

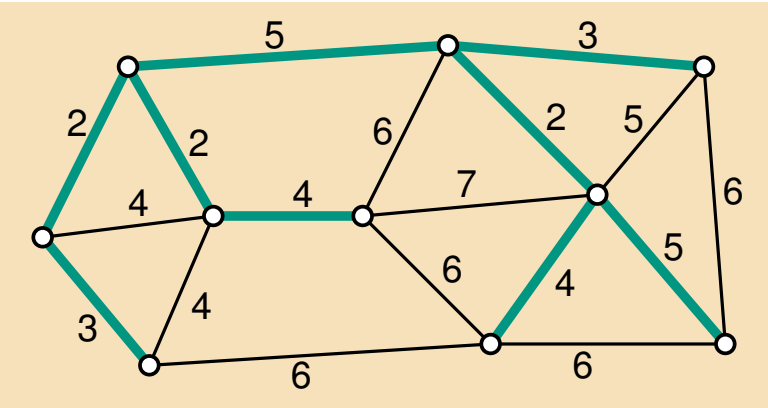
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

app.klicker.uzh.ch/join/algo1

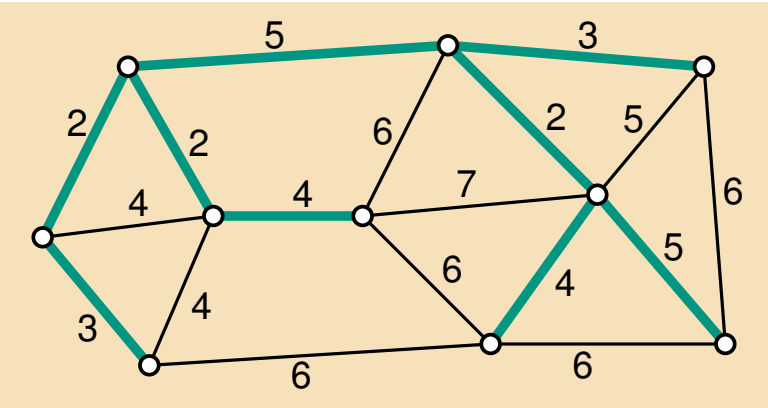
Strukturelle Einsicht → Algorithmus von Prim

Idee

- zeige, dass gewisse Kanten in jedem MST enthalten sind
- finde iterativ solche Kanten und wähle sie aus
- nach $n - 1$ Schritten bilden die $n - 1$ gewählten Kanten den MST

Finde essentielle Kanten

- betrachte Schnitt zwischen abgearbeiteten und nicht-abgearbeiteten Knoten
- starte mit einem abgearbeiteten Knoten
- pro Schritt: markiere Endpunkt der hinzugefügten Kante als abgearbeitet



abgearbeitet

nicht-abgearbeitet

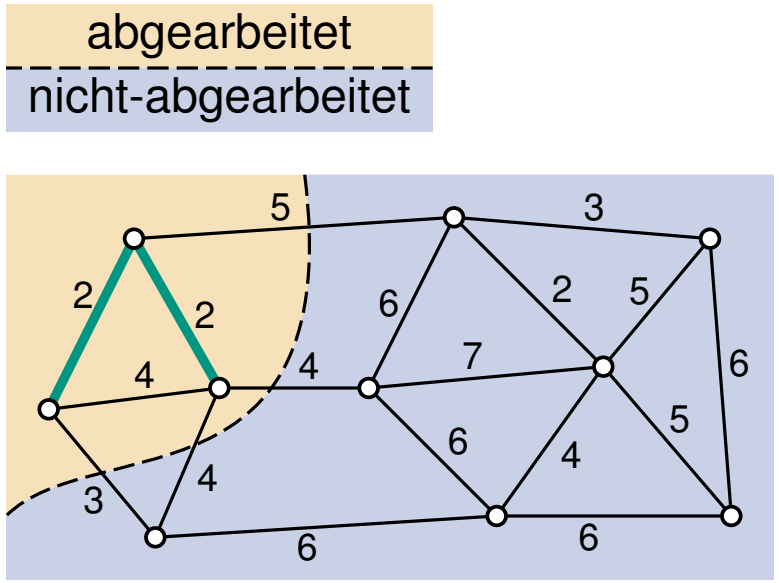
Welche Laufzeit hat der Algorithmus?

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n - 1$ Schritte tun?

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen
  $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$



Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

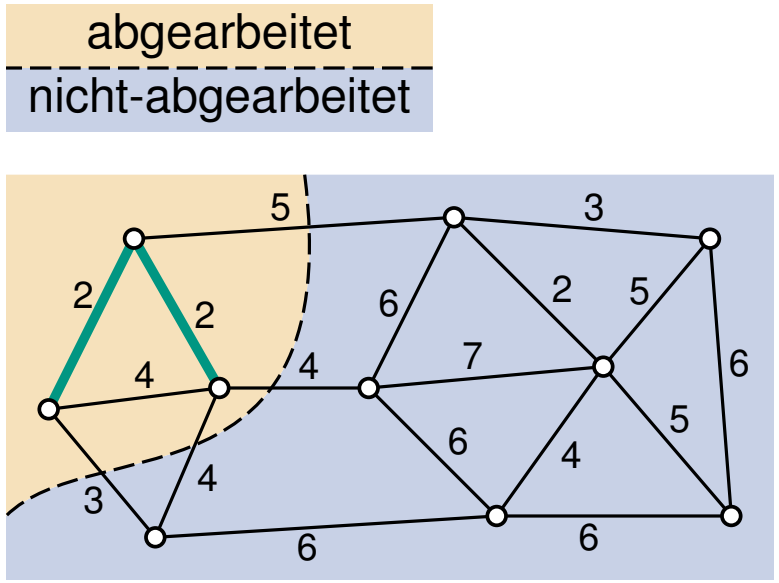
- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin()	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



app.klicker.uzh.ch/join/algo1

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n - 1$ Schritte tun?

(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

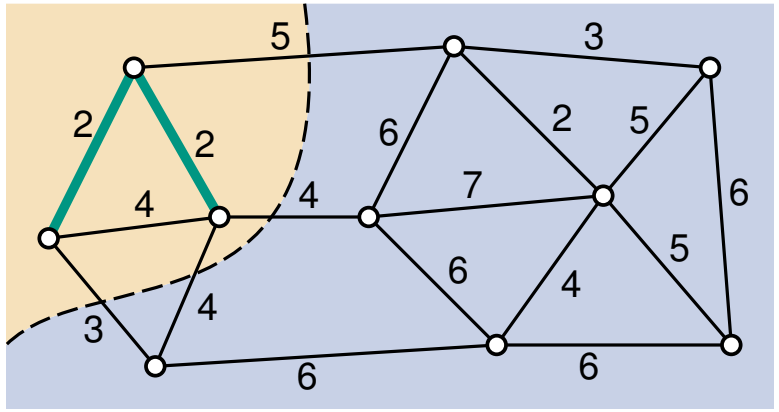
Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin()	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)

abgearbeitet

nicht-abgearbeitet



Welche Laufzeit hat der Algorithmus?

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

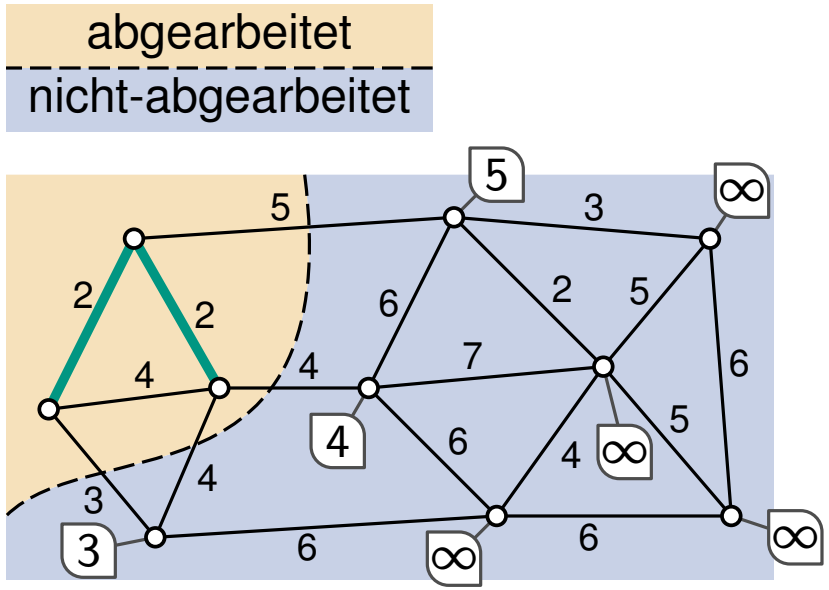
- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.$**popMin**$()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

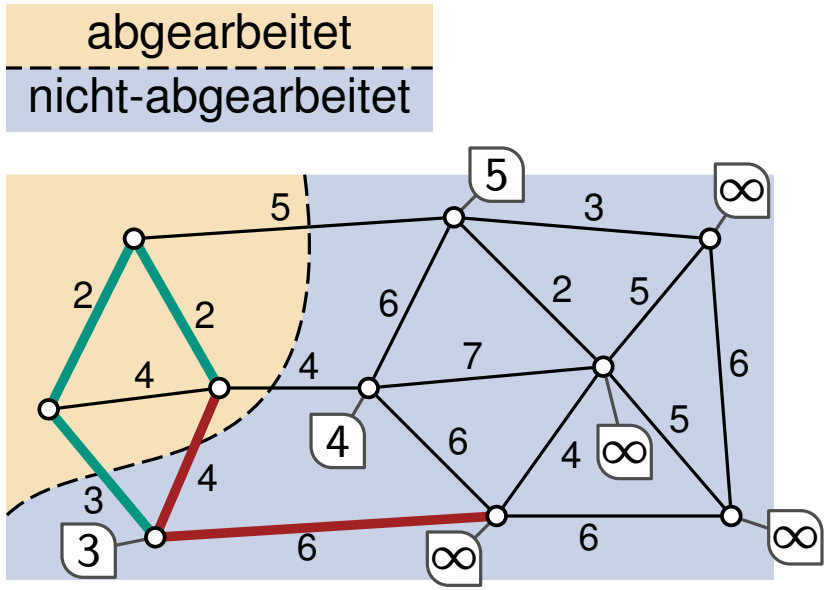
- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

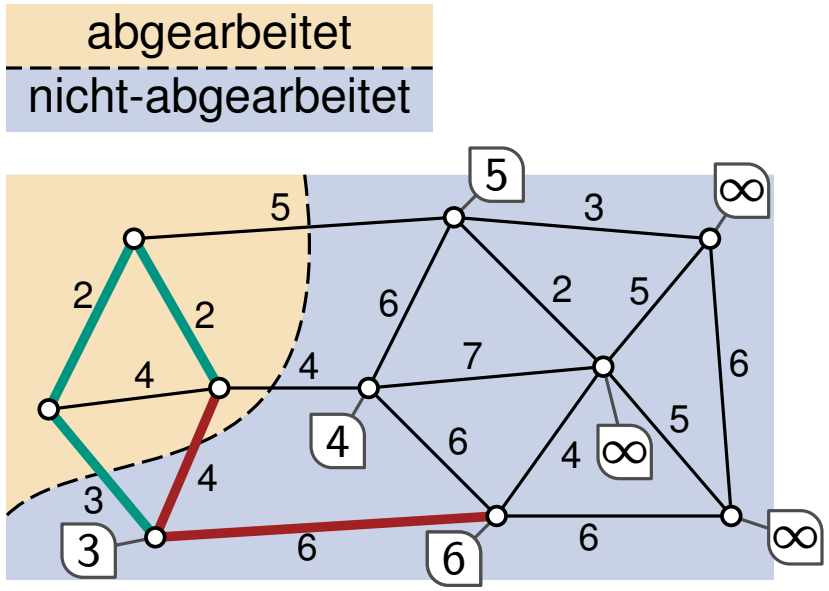
(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen
  $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.$**popMin**$()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

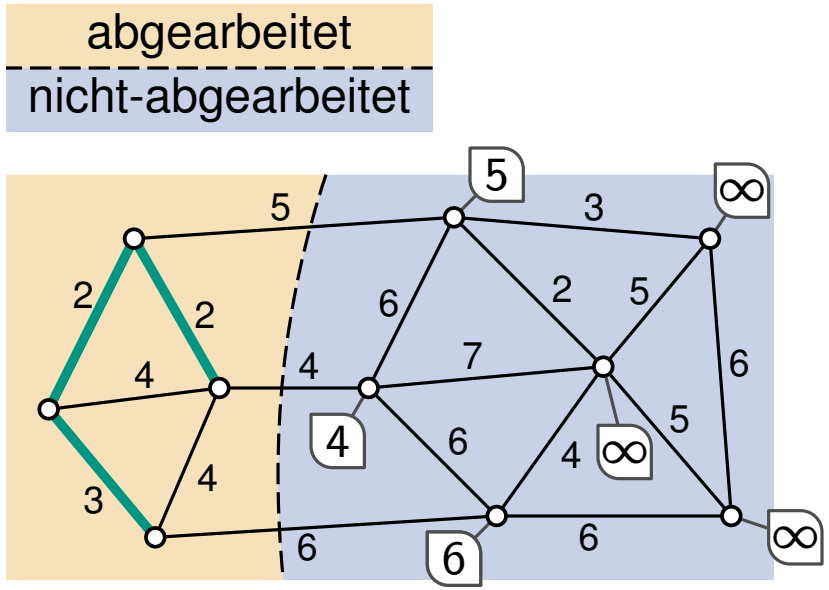
(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

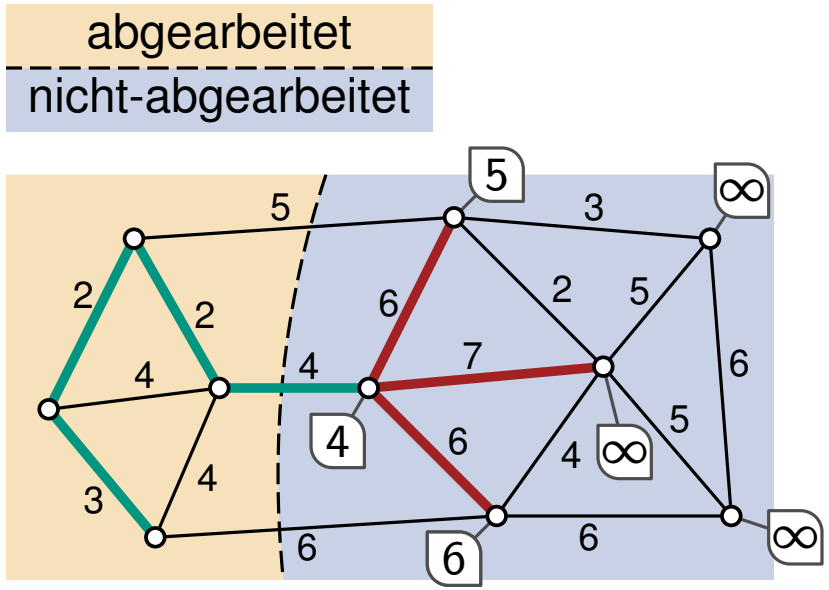
(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen
  $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

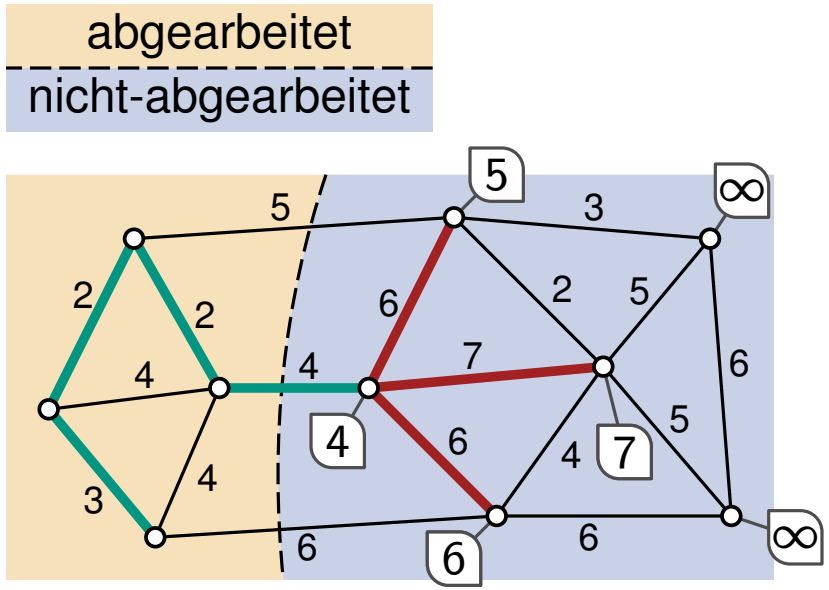
(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen
  $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.$**popMin**$()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

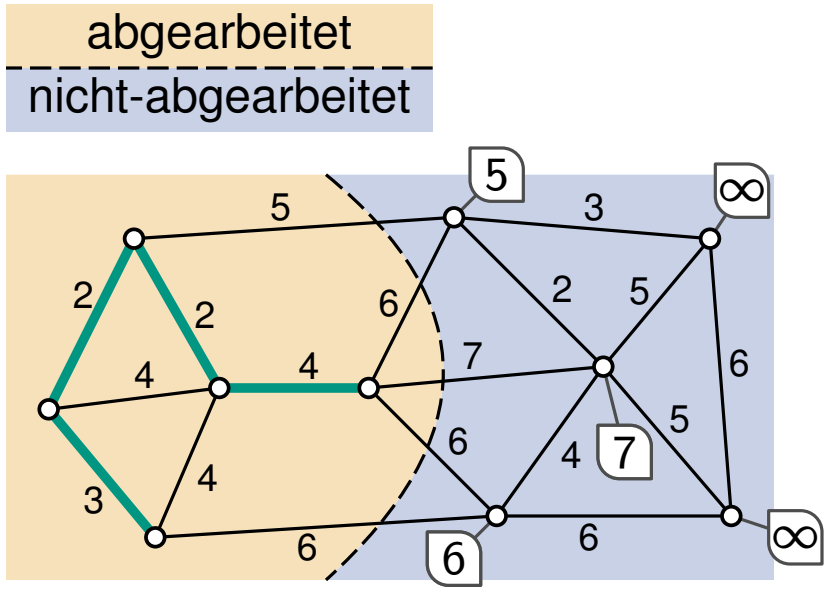
(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen
  $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

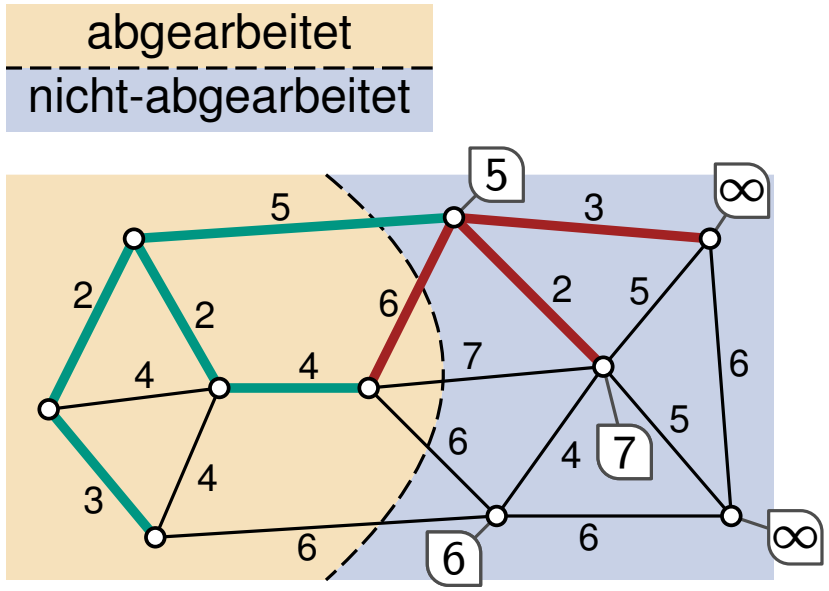
- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

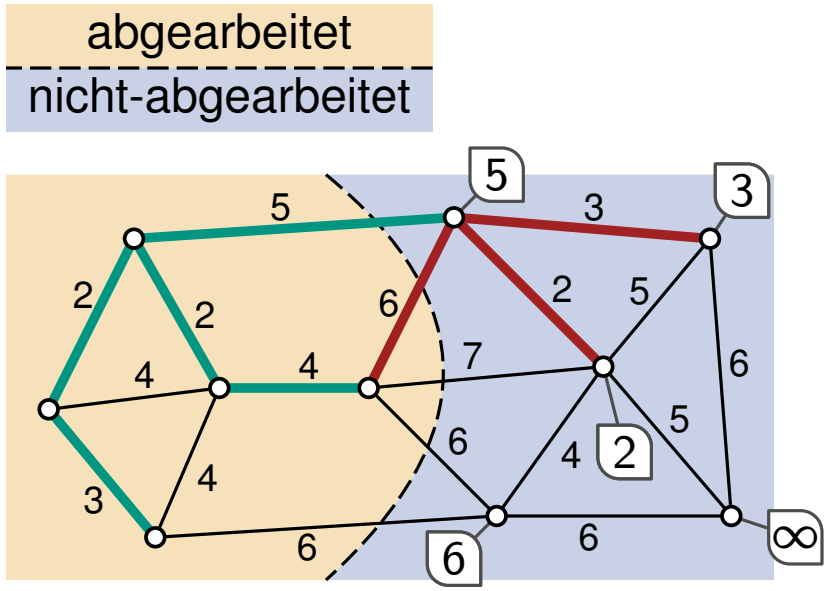
(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Effiziente Umsetzung

Was müssen wir in jedem der $n-1$ Schritte tun?

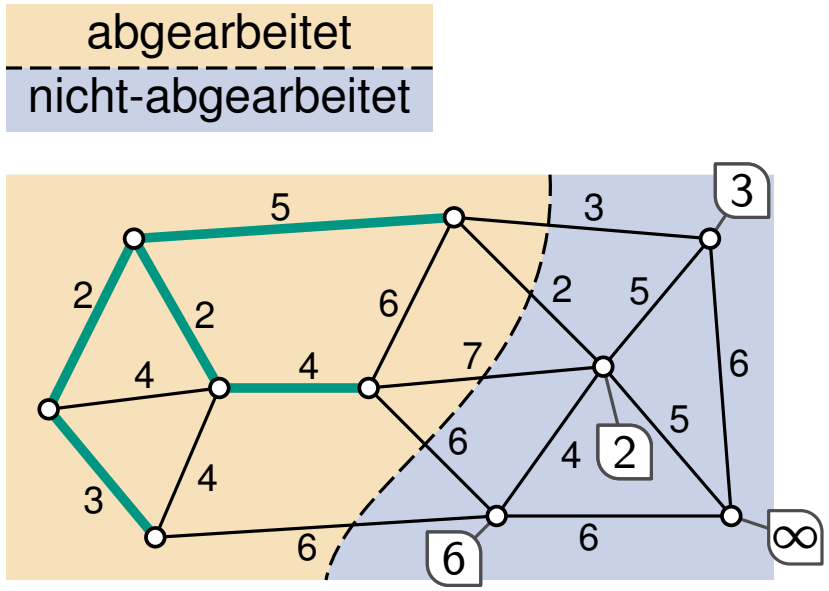
- bestimme Schnittkante mit minimalem Gewicht
- naive Lösung: jede Kante einmal anschauen $\rightarrow \Theta(m) \rightarrow$ insgesamt $\Theta(n \cdot m)$
- Idee: benutze eine Priority-Queue

(z.B. binärer Heap: $\Theta(\log n)$ pro Operation)

Prioritätswarteschlange (Heap)	
push(x, p)	Element x mit Priorität p einfügen
popMin$()$	Element mit minimaler Priorität extrahieren
decPrio(a, p)	Priorität von Knoten a auf p verkleinern

C++: priority_queue **Java:** PriorityQueue

(Prioritätsänderungen muss man sich ggf. selbst bauen: Lazy Evaluation)



Eine mögliche Umsetzung mit Priority-Queue Q

- Ziel für den Zustand von Q zu jedem Zeitpunkt
  - Q enthält die noch nicht abgearbeiteten Knoten
  - Priorität: minimale Kante zu abgearbeiteten Knoten
- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$

Algorithmus von Prim – Laufzeit & Anmerkungen

Eine mögliche Umsetzung mit Priority-Queue Q

- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$
- Initialisierung: jeden Knoten in Q einfügen

Algorithmus von Prim – Laufzeit & Anmerkungen

Eine mögliche Umsetzung mit Priority-Queue Q

- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$
- Initialisierung: jeden Knoten in Q einfügen

$\Theta(n)$ mal **popMin**

$\Theta(m)$ mal **decPrio**

$\Theta(n)$ mal **push**

Algorithmus von Prim – Laufzeit & Anmerkungen

Eine mögliche Umsetzung mit Priority-Queue Q

- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$ — $\Theta(n)$ mal **popMin**
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$ — $\Theta(m)$ mal **decPrio**
- Initialisierung: jeden Knoten in Q einfügen — $\Theta(n)$ mal **push**

$\Rightarrow \Theta((n+m) \cdot \log n)$ mit einem binären und $\Theta(n \log n + m)$ mit einem Fibonacci Heap

Algorithmus von Prim – Laufzeit & Anmerkungen

Eine mögliche Umsetzung mit Priority-Queue Q

- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$ — $\Theta(n)$ mal **popMin**
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$ — $\Theta(m)$ mal **decPrio**
- Initialisierung: jeden Knoten in Q einfügen — $\Theta(n)$ mal **push**

$\Rightarrow \Theta((n+m) \cdot \log n)$ mit einem binären und $\Theta(n \log n + m)$ mit einem Fibonacci Heap

Anmerkung – Vergleich zu Dijkstra

- der Algorithmus hat sehr große Ähnlichkeit zu Dijkstras Algorithmus (kürzeste Wege)
- nur die Prioritäten werden leicht anders aktualisiert

Algorithmus von Prim – Laufzeit & Anmerkungen

Eine mögliche Umsetzung mit Priority-Queue Q

- pro Schritt: abarbeiten von $u = Q.\textbf{popMin}()$ — $\Theta(n)$ mal **popMin**
  - betrachte jeden Nachbarn $v \in N(u)$ von u
  - verringere ggf. die Priorität von v zu $w(\{u, v\})$ — $\Theta(m)$ mal **decPrio**
- Initialisierung: jeden Knoten in Q einfügen — $\Theta(n)$ mal **push**

$\Rightarrow \Theta((n+m) \cdot \log n)$ mit einem binären und $\Theta(n \log n + m)$ mit einem Fibonacci Heap

Anmerkung – Vergleich zu Dijkstra

- der Algorithmus hat sehr große Ähnlichkeit zu Dijkstras Algorithmus (kürzeste Wege)
- nur die Prioritäten werden leicht anders aktualisiert

Anmerkung – Greedy-Algorithmus

- wir wählen in jedem Schritt die minimale Kante, die den bisherigen Teilbaum vergrößert
- wir nennen ein solches Vorgehen auch **Greedy-Algorithmus** (gieriger Algorithmus)

Evaluation

https://onlineumfrage.kit.edu/evasys/online.php?p=MVWNM

Alternative Greedy-Strategie: Kruskal

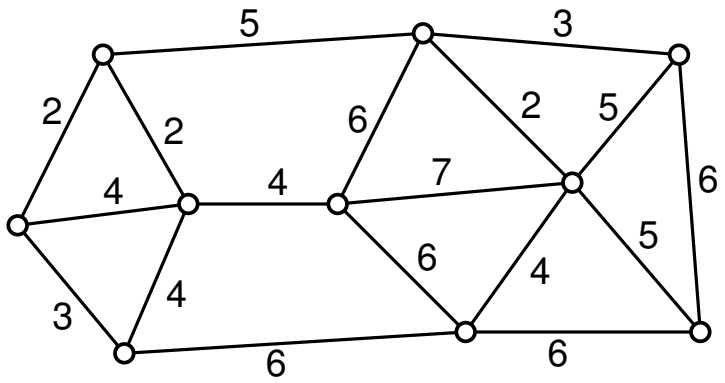
Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden

Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

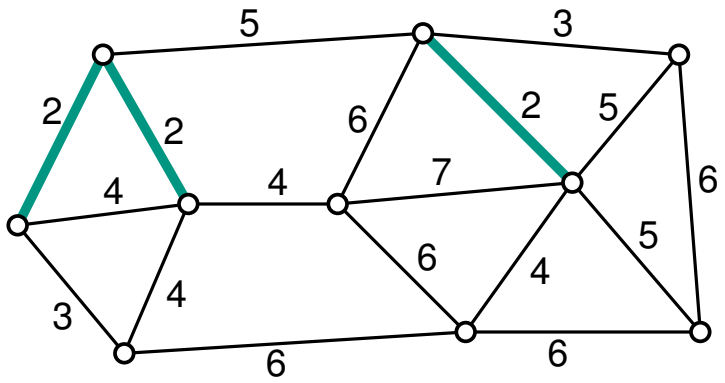
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

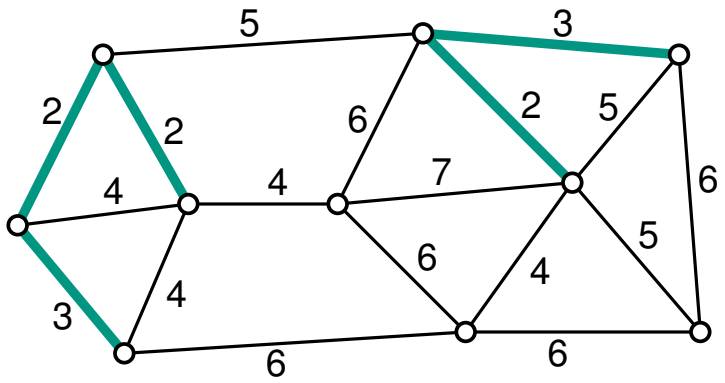
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

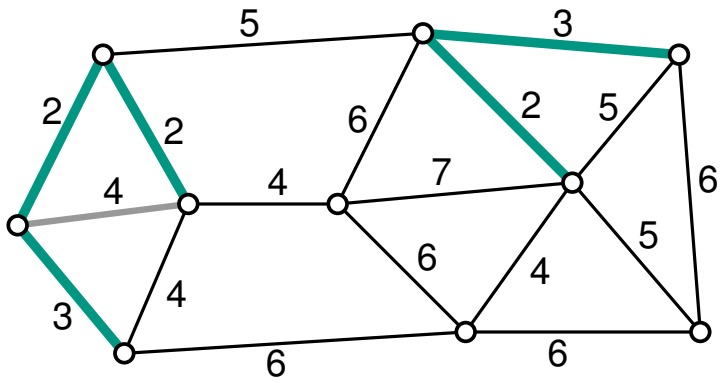
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

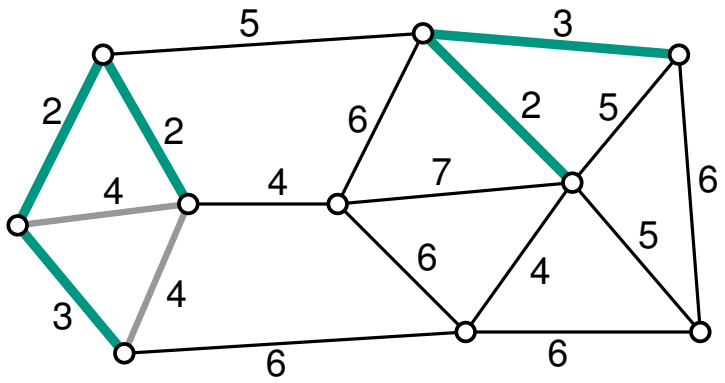
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

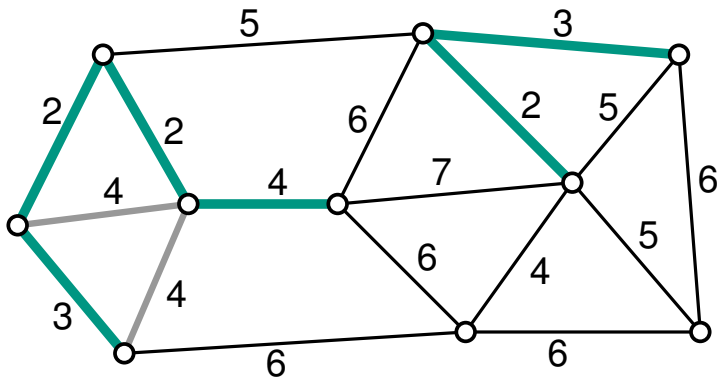
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

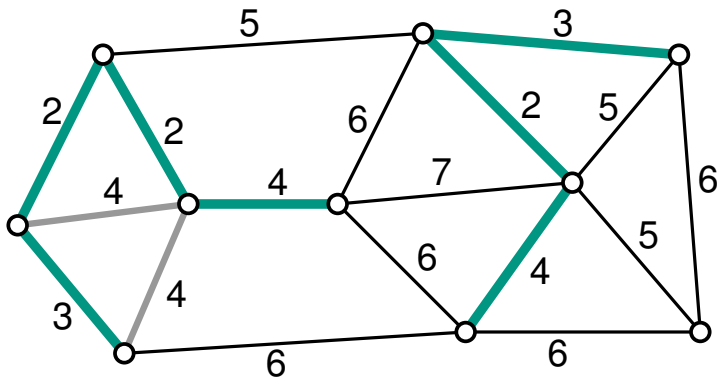
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

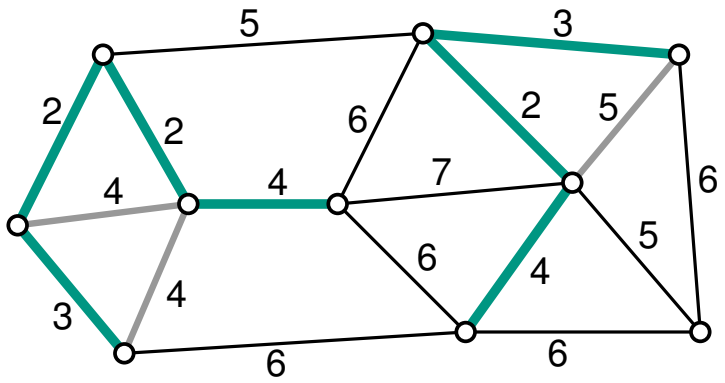
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

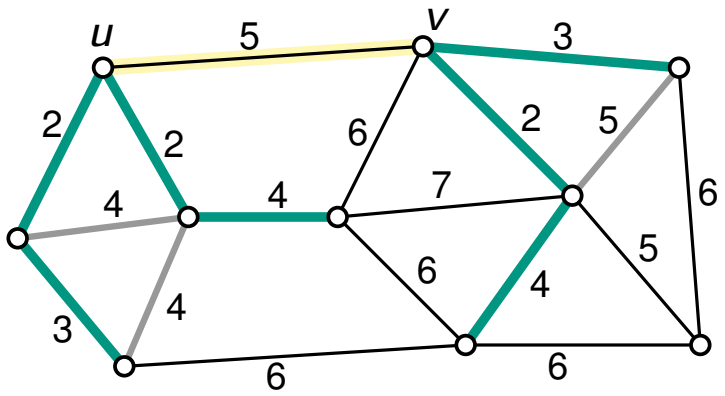
- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Korrektheit

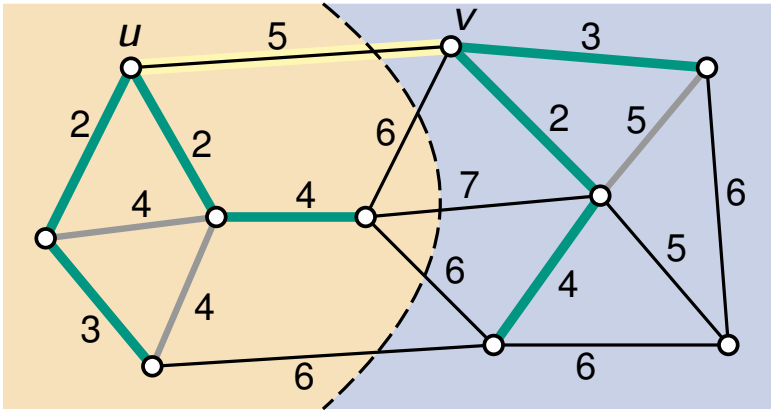
- sei $\{u, v\}$ kleinste Kante, die keinen Kreis schließt

Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

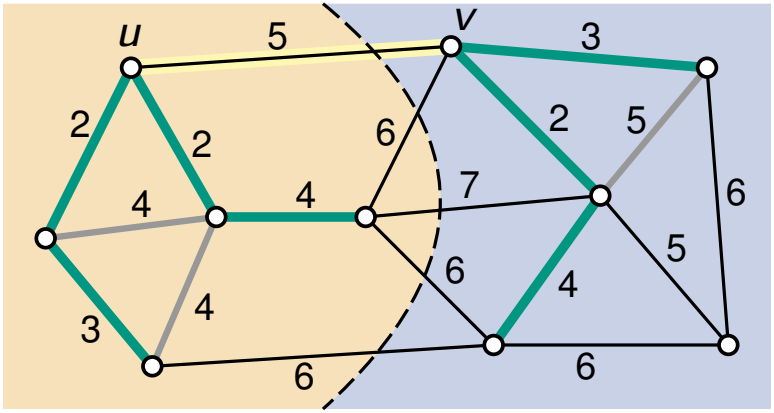
Korrektheit

- sei $\{u, v\}$ kleinste Kante, die keinen Kreis schließt
- beobachte: es gibt einen Schnitt sodass
  - $\{u, v\}$ ist Schnittkante
  - keine Schnittkante ist schon gewählt

Alternative Greedy-Strategie: Kruskal

Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- **Prim:** unter allen Kanten, die den bisherigen Teilbaum vergrößern
- **Kruskal:** unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden



Lemma
Für jeden Schnitt gilt, dass die minimale Schnittkante in jedem MST enthalten ist.

Korrektheit

- sei $\{u, v\}$ kleinste Kante, die keinen Kreis schließt
- beobachte: es gibt einen Schnitt sodass
  - $\{u, v\}$ ist Schnittkante
  - keine Schnittkante ist schon gewählt
- beobachte: Schnittkanten schließen keinen Kreis
- daher: $\{u, v\}$ ist minimale Schnittkante
- Korrektheit folgt aus dem Lemma

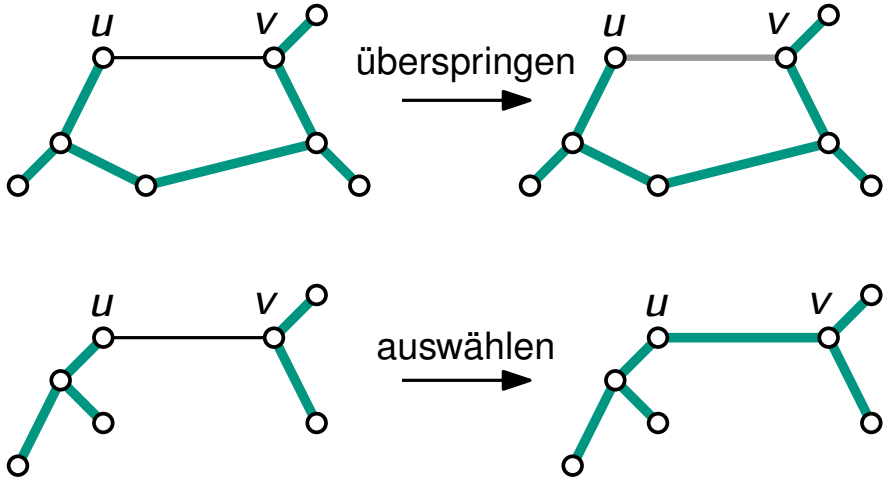
Algorithmus von Kruskal: Umsetzung

Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden

Algorithmische Umsetzung

- sortiere Kanten aufsteigend und iteriere über sie
- zwei Optionen für jede so betrachtete Kante $\{u, v\}$:
  - $\{u, v\}$ schließt Kreis $\rightarrow$ $\{u, v\}$ überspringen
  - $\{u, v\}$ schließt keinen Kreis $\rightarrow$ $\{u, v\}$ auswählen



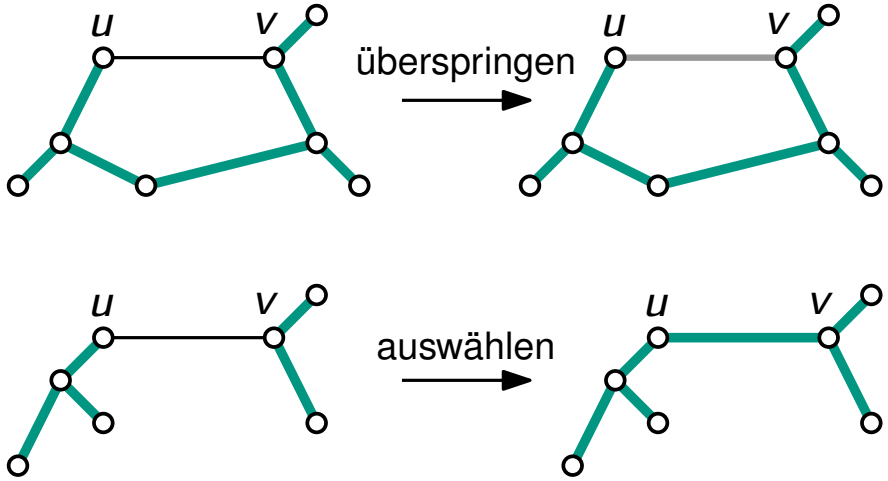
Algorithmus von Kruskal: Umsetzung

Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden

Algorithmische Umsetzung

- sortiere Kanten aufsteigend und iteriere über sie
- zwei Optionen für jede so betrachtete Kante $\{u, v\}$:
  - $\{u, v\}$ schließt Kreis $\rightarrow$ $\{u, v\}$ überspringen
  - $\{u, v\}$ schließt keinen Kreis $\rightarrow$ $\{u, v\}$ auswählen



Wie testen wir schnell, ob $\{u, v\}$ Kreis schließt?

- $\{u, v\}$ schließt Kreis $\Leftrightarrow$ u und v liegen bzgl. gewählter Kanten in der selben Komponente

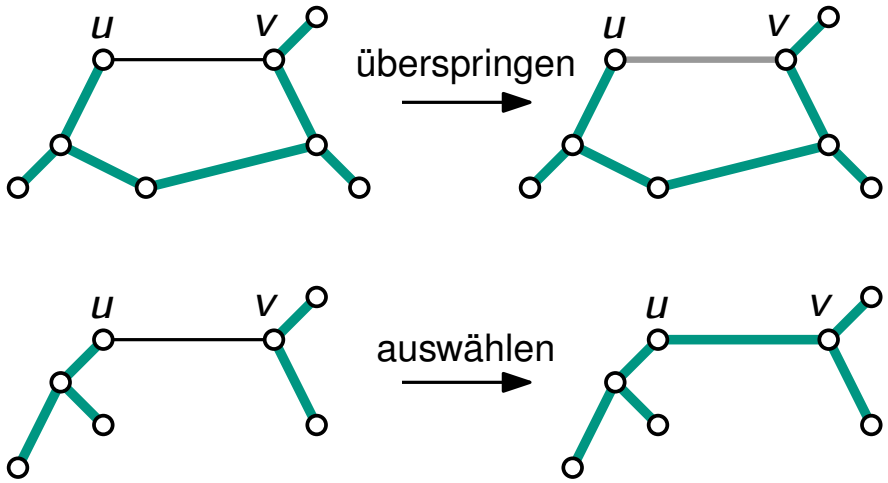
Algorithmus von Kruskal: Umsetzung

Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden

Algorithmische Umsetzung

- sortiere Kanten aufsteigend und iteriere über sie
- zwei Optionen für jede so betrachtete Kante $\{u, v\}$:
  - $\{u, v\}$ schließt Kreis $\rightarrow$ $\{u, v\}$ überspringen
  - $\{u, v\}$ schließt keinen Kreis $\rightarrow$ $\{u, v\}$ auswählen



Wie testen wir schnell, ob $\{u, v\}$ Kreis schließt?

- $\{u, v\}$ schließt Kreis $\Leftrightarrow$ u und v liegen bzgl. gewählter Kanten in der selben Komponente
- jedes Mal BFS oder DFS ist zu teuer ($\Theta(m)$)

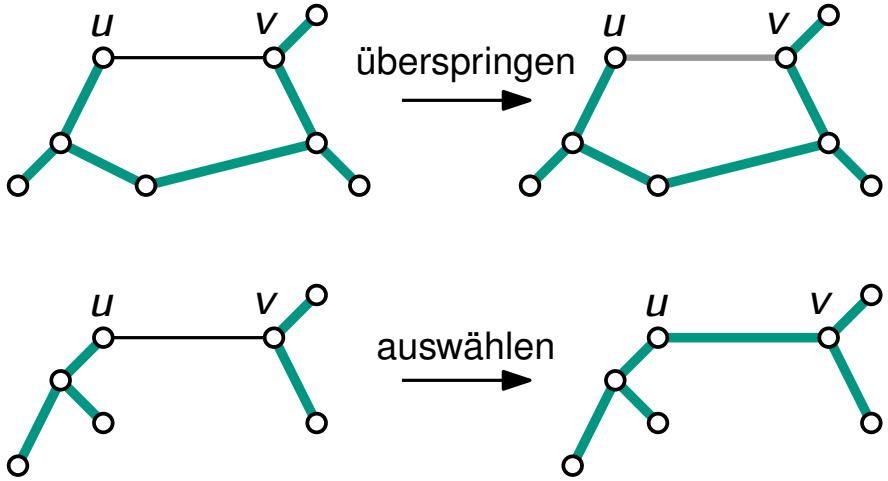
Algorithmus von Kruskal: Umsetzung

Greedy Auswahl

- wähle in jedem Schritt minimale Kante
- unter allen Kanten, die keinen Kreis mit schon gewählten Kanten bilden

Algorithmische Umsetzung

- sortiere Kanten aufsteigend und iteriere über sie
- zwei Optionen für jede so betrachtete Kante $\{u, v\}$:
  - $\{u, v\}$ schließt Kreis $\rightarrow$ $\{u, v\}$ überspringen
  - $\{u, v\}$ schließt keinen Kreis $\rightarrow$ $\{u, v\}$ auswählen



Wie testen wir schnell, ob $\{u, v\}$ Kreis schließt?

- $\{u, v\}$ schließt Kreis $\Leftrightarrow$ u und v liegen bzgl. gewählter Kanten in der selben Komponente
- jedes Mal BFS oder DFS ist zu teuer ($\Theta(m)$)
- nächstes Mal: Datenstruktur zur effizienten Verwaltung der Komponenten

Zusammenfassung

Greedy Algorithmen im Allgemeinen

- Vorgehen: triff iterativ Entscheidungen, die in dem Moment am besten aussehen
- oft nicht optimal (und ggf. sehr schlecht im Worst Case), aber gute Heuristik für die Praxis
- wenn optimal: Beweis mittels Austauschargument
  (für jede Lösung ohne das greedy gewählte Element gibt es eine mindestens genauso gute mit dem Element)

Zusammenfassung

Greedy Algorithmen im Allgemeinen

- Vorgehen: triff iterativ Entscheidungen, die in dem Moment am besten aussehen
- oft nicht optimal (und ggf. sehr schlecht im Worst Case), aber gute Heuristik für die Praxis
- wenn optimal: Beweis mittels Austauschargument
  (für jede Lösung ohne das greedy gewählte Element gibt es eine mindestens genauso gute mit dem Element)

Minimaler Spannbaum (MST)

- zwei Greedy-Varianten, die optimale Lösung liefern (Prim, Kruskal)
- tatsächlich funktioniert fast jegliche Greedy-Variante korrekt
  (z.B. lösche iterativ die größte Kante auf einem Kreis)
- effiziente Implementierung von Prim: Priority-Queue
- effiziente Implementierung von Kruskal: Union-Find (nächstes Mal)