Graphen und Algorithmen

Vorlesung #2: Minimale aufspannende Bäume

Dr. Armin Fügenschuh
Technische Universität Darmstadt
WS 2007/2008

Übersicht

Übersicht

* Definition und Eigenschaften von Bäumen

Übersicht

* Definition und Eigenschaften von Bäumen
* Abzählen von Bäumen

Übersicht

* Definition und Eigenschaften von Bäumen
* Abzählen von Bäumen
* Aufspannende Bäume

Übersicht

* Definition und Eigenschaften von Bäumen
* Abzählen von Bäumen
* Aufspannende Bäume
* Minimale Spannbäume und maximale Wälder

Übersicht

* Definition und Eigenschaften von Bäumen
* Abzählen von Bäumen
* Aufspannende Bäume
* Minimale Spannbäume und maximale Wälder
* Anwendungen von Spannbäumen

Übersicht

* Definition und Eigenschaften von Bäumen
* Abzählen von Bäumen
* Aufspannende Bäume
* Minimale Spannbäume und maximale Wälder
* Anwendungen von Spannbäumen
* Optimalitätsbedingungen für Spannbäume

Übersicht

* Definition und Eigenschaften von Bäumen
* Abzählen von Bäumen
* Aufspannende Bäume
* Minimale Spannbäume und maximale Wälder
* Anwendungen von Spannbäumen
* Optimalitätsbedingungen für Spannbäume
* Algorithmus von Kruskal

Übersicht

* Definition und Eigenschaften von Bäumen
* Abzählen von Bäumen
* Aufspannende Bäume
* Minimale Spannbäume und maximale Wälder
* Anwendungen von Spannbäumen
* Optimalitätsbedingungen für Spannbäume
* Algorithmus von Kruskal
* Algorithmus von Jarnik (Prim)

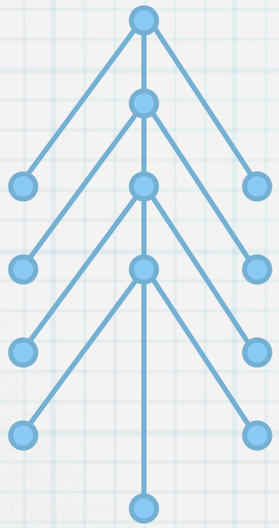
Bäume

Bäume

* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.

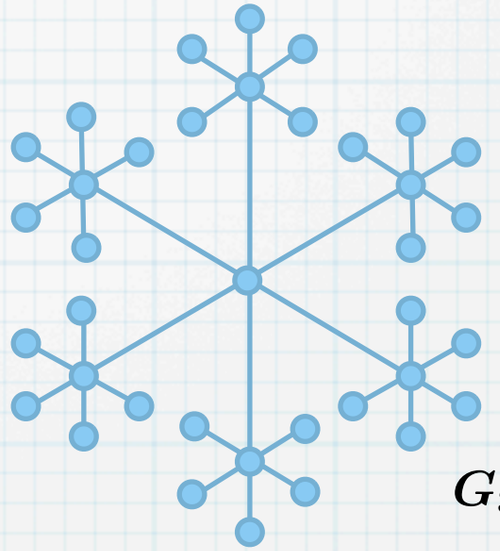
Bäume

* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.
* Beispiele:



G_1

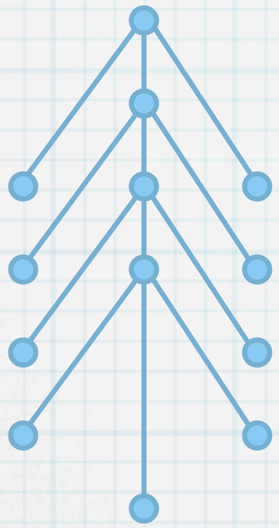
G_2



G_3

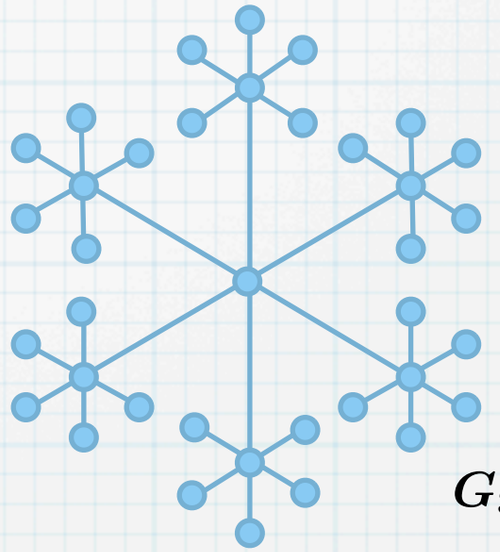
Bäume

* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.
* Beispiele:



G_1

G_2

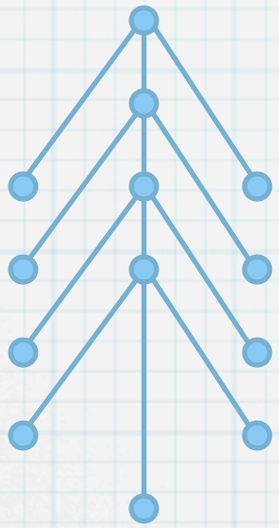


G_3

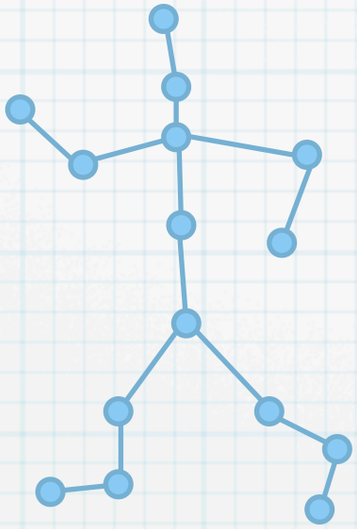
* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G :

Bäume

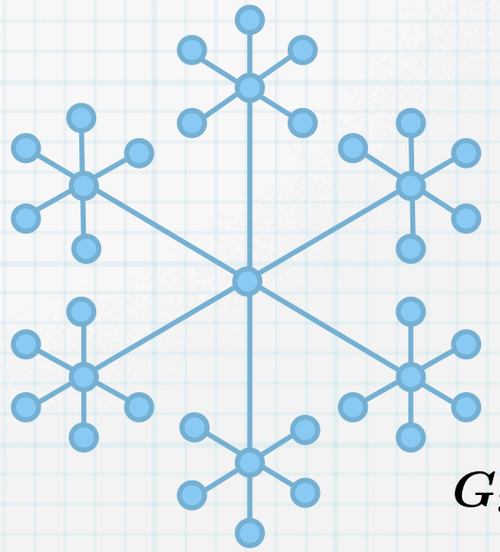
* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.
* Beispiele:



G_1



G_2

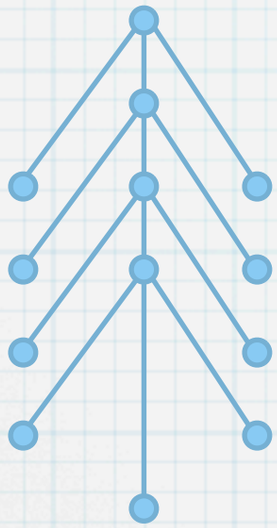


G_3

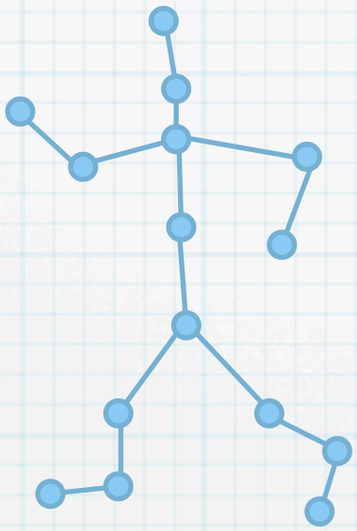
* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G :
  (i) G ist ein Baum.

Bäume

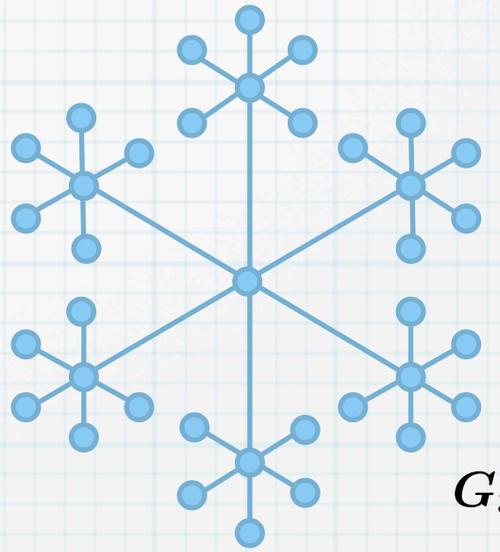
* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.
* Beispiele:



G_1



G_2

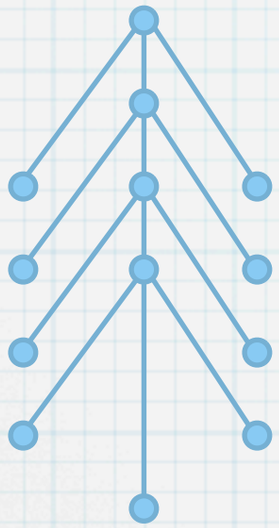


G_3

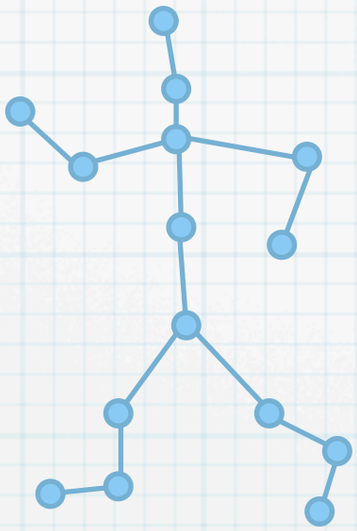
* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.

Bäume

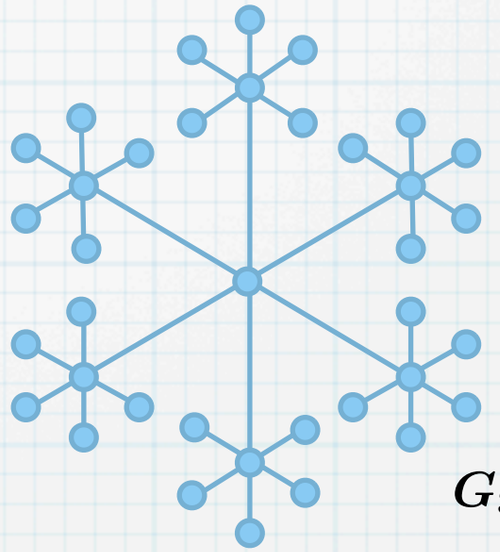
* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.
* Beispiele:



G_1



G_2

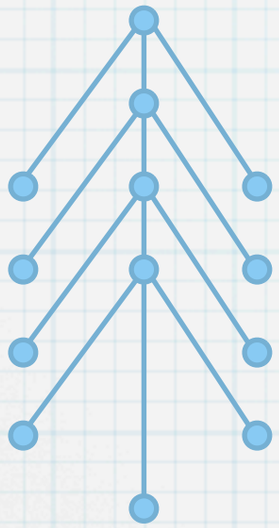


G_3

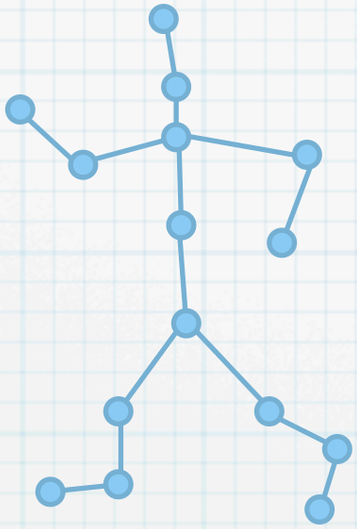
* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.

Bäume

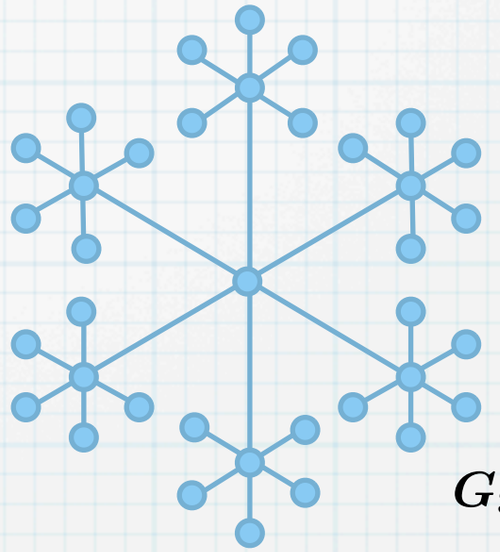
* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.
* Beispiele:



G_1



G_2

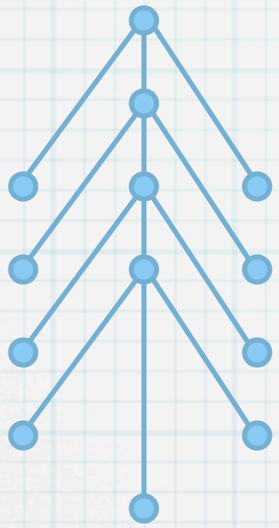


G_3

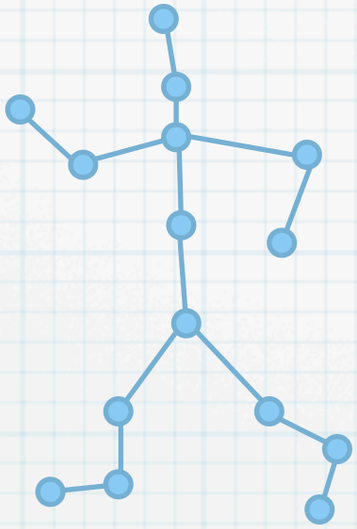
* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.

Bäume

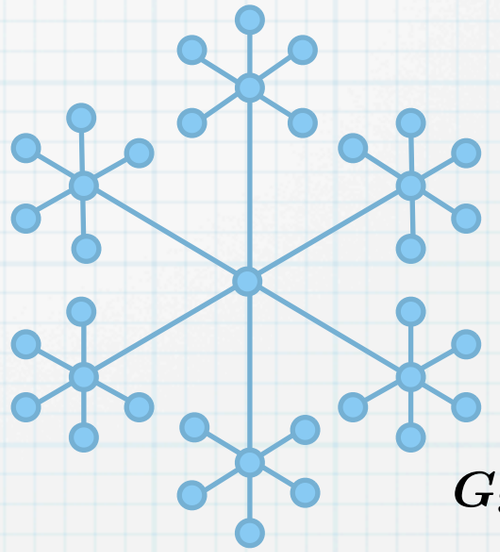
* **Definition 1**:
  Ein **Baum** ist ein zusammenhängender Graph, der keine Kreise enthält.
* Beispiele:



G_1



G_2



G_3

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

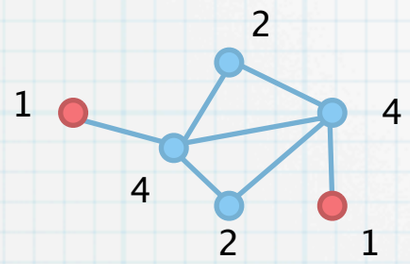
Ein Lemma für den Beweis von Satz 2

Ein Lemma für den Beweis von Satz 2

* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.

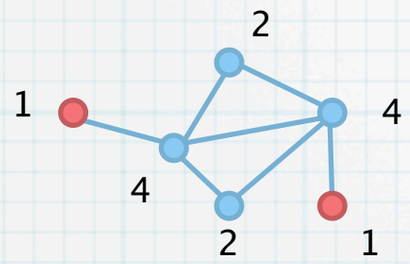
Ein Lemma für den Beweis von Satz 2

* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



Ein Lemma für den Beweis von Satz 2

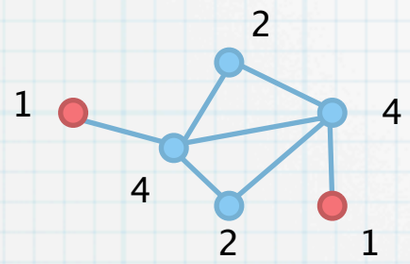
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.

Ein Lemma für den Beweis von Satz 2

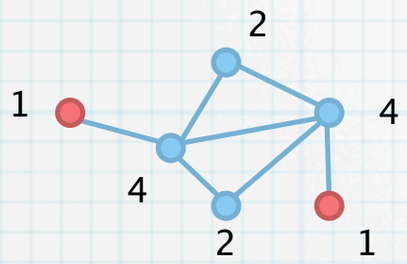
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:

Ein Lemma für den Beweis von Satz 2

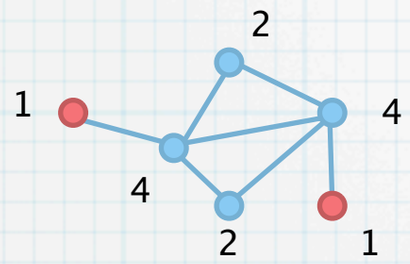
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:
  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.

Ein Lemma für den Beweis von Satz 2

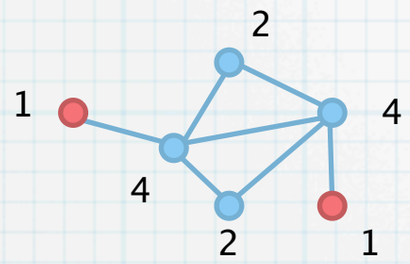
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:
  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.
  Die Länge von P ist mindestens 1, also ist $v_0 \neq v_t$.

Ein Lemma für den Beweis von Satz 2

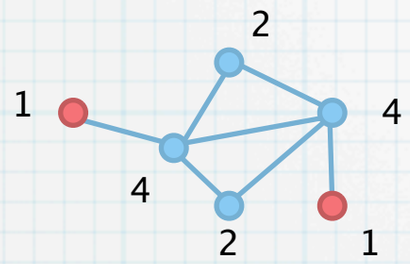
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:
  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.
  Die Länge von P ist mindestens 1, also ist $v_0 \neq v_t$.
  Behauptung: v_0, v_t sind beides Endknoten.

Ein Lemma für den Beweis von Satz 2

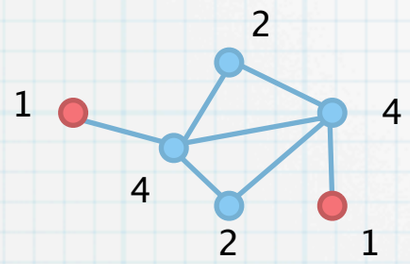
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:
  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.
  Die Länge von P ist mindestens 1, also ist $v_0 \neq v_t$.
  Behauptung: v_0, v_t sind beides Endknoten.
    Sei o.B.d.A. v_0 kein Endknoten.

Ein Lemma für den Beweis von Satz 2

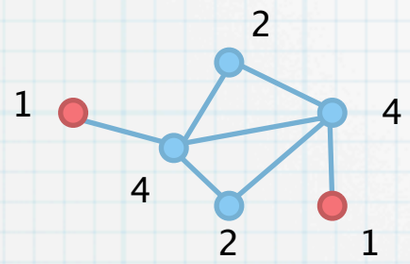
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:
  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.
  Die Länge von P ist mindestens 1, also ist $v_0 \neq v_t$.
  Behauptung: v_0, v_t sind beides Endknoten.
    Sei o.B.d.A. v_0 kein Endknoten.
    Dann gibt es eine Kante $e = \{v_0, v\}$, die v_0 enthält und von $e_1 = \{v_0, v_1\}$verschieden ist.

Ein Lemma für den Beweis von Satz 2

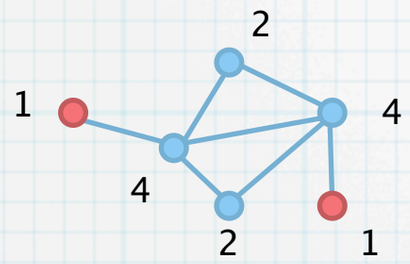
* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:
  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.
  Die Länge von P ist mindestens 1, also ist $v_0 \neq v_t$.
  Behauptung: v_0, v_t sind beides Endknoten.
    Sei o.B.d.A. v_0 kein Endknoten.
    Dann gibt es eine Kante $e = \{v_0, v\}$, die v_0 enthält und von $e_1 = \{v_0, v_1\}$verschieden ist.
    Dann ist <u>entweder</u> v ein Knoten im Pfad P, d.h. $v = v_i, i \geq 2$, und e zusammen mit diesem Teil des Pfades ist ein Kreis. Widerspruch.

Ein Lemma für den Beweis von Satz 2

* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:

  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.

  Die Länge von P ist mindestens 1, also ist $v_0 \neq v_t$.

  Behauptung: v_0, v_t sind beides Endknoten.

  Sei o.B.d.A. v_0 kein Endknoten.

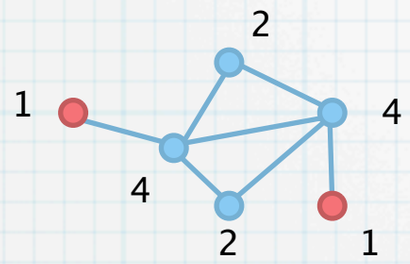
  Dann gibt es eine Kante $e = \{v_0, v\}$, die v_0 enthält und von $e_1 = \{v_0, v_1\}$verschieden ist.

  Dann ist <u>entweder</u> v ein Knoten im Pfad P, d.h. $v = v_i, i \geq 2$, und e zusammen mit diesem Teil des Pfades ist ein Kreis. Widerspruch.

  <u>Oder</u> $v \notin \{v_0, \ldots, v_t\}$, dann kann Pfad P durch Hinzufügen von e verlängert werden. Ebenfalls ein Widerspruch.

Ein Lemma für den Beweis von Satz 2

* **Definition 3**:
  Ein Knoten mit Grad 1 in einem Graphen G heißt **Endknoten** oder **Blatt** von G.
* Beispiel:



* **Lemma 4** (Endknotenlemma):
  Ein Baum $T = (V, E)$ mit mindestens 2 Knoten enthält mindestens 2 Endknoten.
* Beweis:
  Sei $P = (v_0, e_1, v_1, \ldots, e_t, v_t)$ ein Pfad maximaler Länge in T.
  Die Länge von P ist mindestens 1, also ist $v_0 \neq v_t$.
  Behauptung: v_0, v_t sind beides Endknoten.
    Sei o.B.d.A. v_0 kein Endknoten.
    Dann gibt es eine Kante $e = \{v_0, v\}$, die v_0 enthält und von $e_1 = \{v_0, v_1\}$verschieden ist.
    Dann ist <u>entweder</u> v ein Knoten im Pfad P, d.h. $v = v_i, i \geq 2$, und e zusammen mit diesem Teil des Pfades ist ein Kreis. Widerspruch.
    <u>Oder</u> $v \notin \{v_0, \ldots, v_t\}$, dann kann Pfad P durch Hinzufügen von e verlängert werden. Ebenfalls ein Widerspruch.
* Bemerkung: Lemma 4 gilt nicht in unendlichen Graphen.

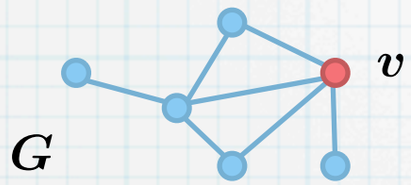
Ein weiteres Lemma für den Beweis von Satz 2

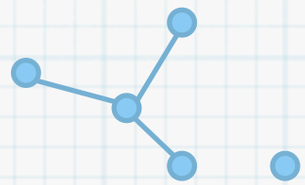
Ein weiteres Lemma für den Beweis von Satz 2

* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.

Ein weiteres Lemma für den Beweis von Satz 2

* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:

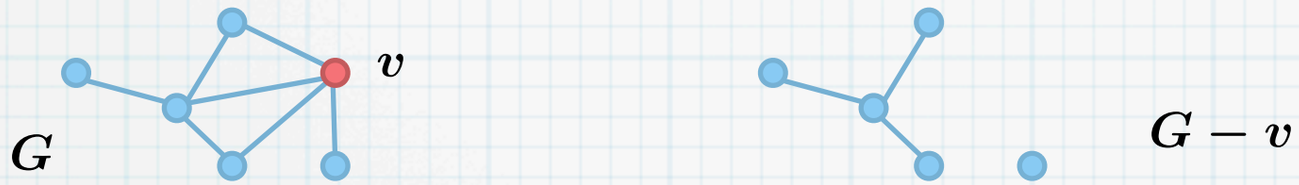




$G - v$

Ein weiteres Lemma für den Beweis von Satz 2

* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:

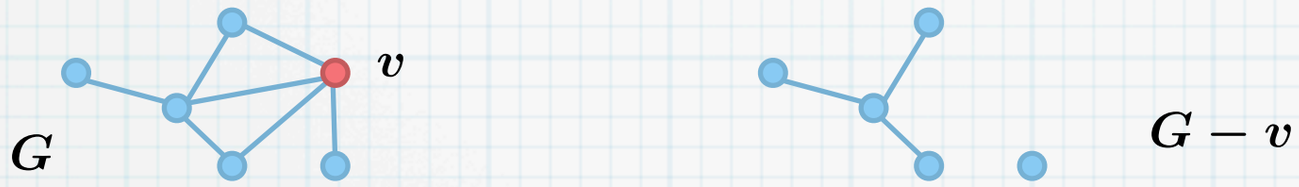
Ein weiteres Lemma für den Beweis von Satz 2

* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:

* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.

Ein weiteres Lemma für den Beweis von Satz 2

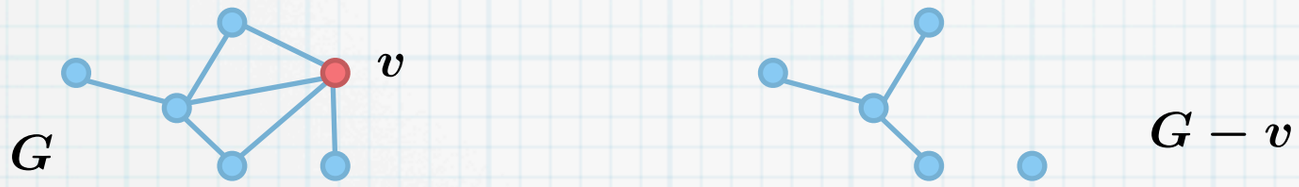
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.

Ein weiteres Lemma für den Beweis von Satz 2

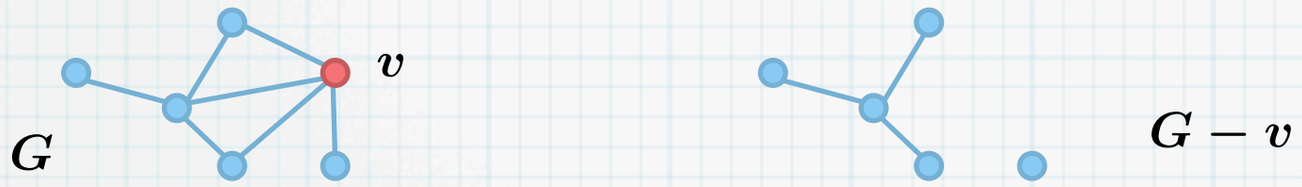
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:

Ein weiteres Lemma für den Beweis von Satz 2

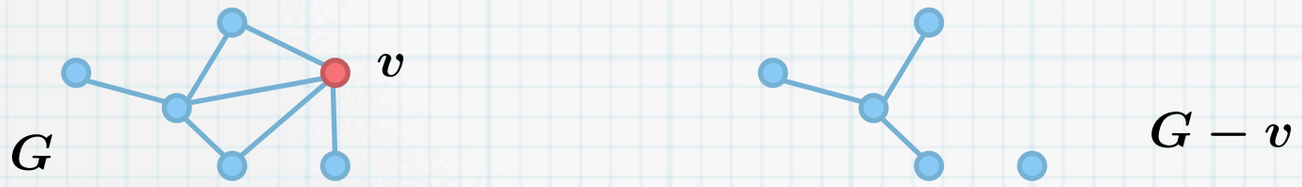
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.

Ein weiteres Lemma für den Beweis von Satz 2

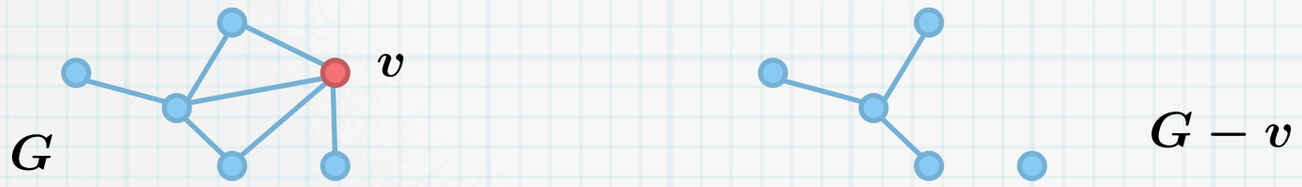
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.

Ein weiteres Lemma für den Beweis von Satz 2

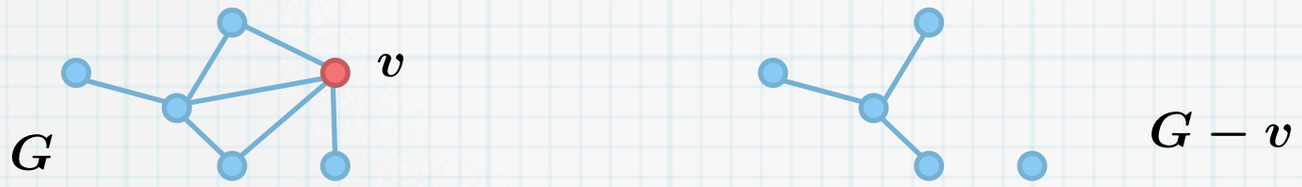
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.
  Dieser Pfad enthält keinen Knoten mit Grad 1 (außer evtl. x, y). Also enthält er nicht v.

Ein weiteres Lemma für den Beweis von Satz 2

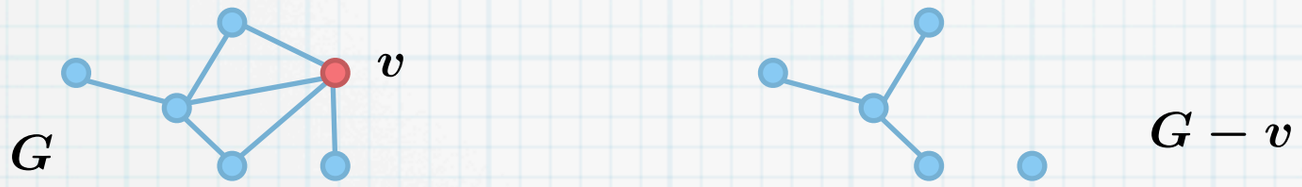
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.
  Dieser Pfad enthält keinen Knoten mit Grad 1 (außer evtl. x, y). Also enthält er nicht v.
  Also ist der Pfad vollständig in $G - v$, und $G - v$ ist somit zusammenhängend.

Ein weiteres Lemma für den Beweis von Satz 2

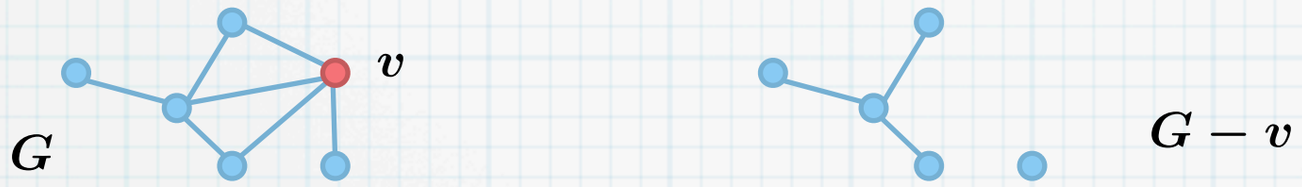
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.
  Dieser Pfad enthält keinen Knoten mit Grad 1 (außer evtl. x, y). Also enthält er nicht v.
  Also ist der Pfad vollständig in $G - v$, und $G - v$ ist somit zusammenhängend.
  Da G keinen Kreis enthält, kann auch $G - v$ keinen Kreis enthalten.

Ein weiteres Lemma für den Beweis von Satz 2

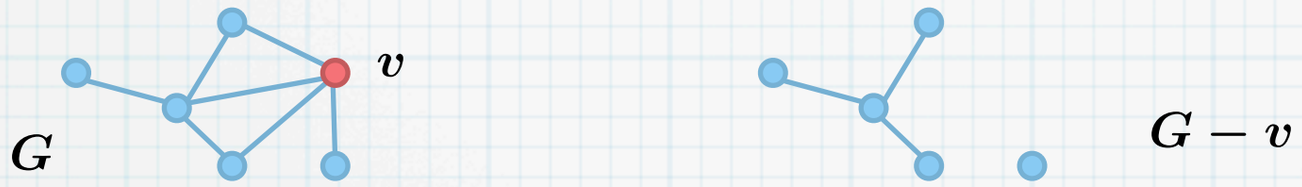
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.
  Dieser Pfad enthält keinen Knoten mit Grad 1 (außer evtl. x, y). Also enthält er nicht v.
  Also ist der Pfad vollständig in $G - v$, und $G - v$ ist somit zusammenhängend.
  Da G keinen Kreis enthält, kann auch $G - v$ keinen Kreis enthalten.
  Damit ist $G - v$ ein Baum.

Ein weiteres Lemma für den Beweis von Satz 2

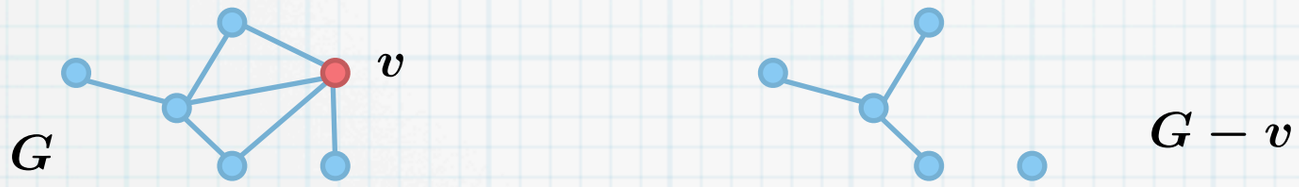
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.
  Dieser Pfad enthält keinen Knoten mit Grad 1 (außer evtl. x, y). Also enthält er nicht v.
  Also ist der Pfad vollständig in $G - v$, und $G - v$ ist somit zusammenhängend.
  Da G keinen Kreis enthält, kann auch $G - v$ keinen Kreis enthalten.
  Damit ist $G - v$ ein Baum.
  (ii) $\Rightarrow$ (i): Durch Hinzufügen eines Endknotens v zu $G - v$ kann kein Kreis entstehen.

Ein weiteres Lemma für den Beweis von Satz 2

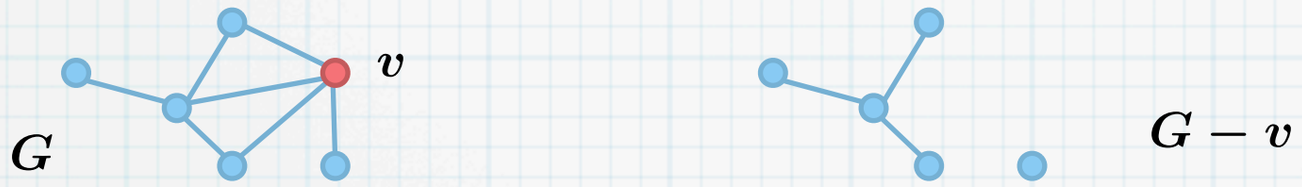
* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.
  Dieser Pfad enthält keinen Knoten mit Grad 1 (außer evtl. x, y). Also enthält er nicht v.
  Also ist der Pfad vollständig in $G - v$, und $G - v$ ist somit zusammenhängend.
  Da G keinen Kreis enthält, kann auch $G - v$ keinen Kreis enthalten.
  Damit ist $G - v$ ein Baum.
  (ii) $\Rightarrow$ (i): Durch Hinzufügen eines Endknotens v zu $G - v$ kann kein Kreis entstehen.
  Es gibt einen Pfad von v über den (eindeutigen) Nachbarn v' von v zu jedem anderen Knoten in G.

Ein weiteres Lemma für den Beweis von Satz 2

* **Definition 5**:
  Sei G ein Graph und v ein Knoten von G. Mit $G - v$ bezeichnen wir den Graphen, der entsteht, wenn man von G Knoten v und alle dazugehörigen Kanten entfernt.
* Beispiel:



* **Lemma 6** (Baumwachstumslemma):
  Sei G ein Graph, der einen Endknoten v enthält. Dann sind folgende Aussagen äquivalent:
  (i) G ist ein Baum.
  (ii) $G - v$ ist ein Baum.
* Beweis:
  (i) $\Rightarrow$ (ii): Betrachte zwei Knoten x, y in $G - v$.
  Da G zusammenhängt, sind x, y durch einen Pfad verbunden.
  Dieser Pfad enthält keinen Knoten mit Grad 1 (außer evtl. x, y). Also enthält er nicht v.
  Also ist der Pfad vollständig in $G - v$, und $G - v$ ist somit zusammenhängend.
  Da G keinen Kreis enthält, kann auch $G - v$ keinen Kreis enthalten.
  Damit ist $G - v$ ein Baum.
  (ii) $\Rightarrow$ (i): Durch Hinzufügen eines Endknotens v zu $G - v$ kann kein Kreis entstehen.
  Es gibt einen Pfad von v über den (eindeutigen) Nachbarn v' von v zu jedem anderen Knoten in G.
  Also ist G ein Baum.

Beweis von Satz 2

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:

  (i) G ist ein Baum.

  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.

  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.

  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.

  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.
* Sei G ein Baum mit mind. 2 Knoten.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.
* Sei G ein Baum mit mind. 2 Knoten.
  G hat mind. 2 Endknoten (Endknotenlemma); sei v einer davon, und sei v' sein Nachbar.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.
* Sei G ein Baum mit mind. 2 Knoten.
  G hat mind. 2 Endknoten (Endknotenlemma); sei v einer davon, und sei v' sein Nachbar.
  Induktionsannahme: Aussage ist wahr für $G - v$.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.
* Sei G ein Baum mit mind. 2 Knoten.
  G hat mind. 2 Endknoten (Endknotenlemma); sei v einer davon, und sei v' sein Nachbar.
  Induktionsannahme: Aussage ist wahr für $G - v$.
* Die Aussagen (i) $\Rightarrow$ (ii), (iii), (iv), (v) sind dann offensichtlich wahr.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.
* Sei G ein Baum mit mind. 2 Knoten.
  G hat mind. 2 Endknoten (Endknotenlemma); sei v einer davon, und sei v' sein Nachbar.
  Induktionsannahme: Aussage ist wahr für $G - v$.
* Die Aussagen (i) $\Rightarrow$ (ii), (iii), (iv), (v) sind dann offensichtlich wahr.
* Anmerkung: Für die Aussage (i) $\Rightarrow$ (iv) brauchen wir die Induktionsannahme für $G - v$ nicht.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.
* Sei G ein Baum mit mind. 2 Knoten.
  G hat mind. 2 Endknoten (Endknotenlemma); sei v einer davon, und sei v' sein Nachbar.
  Induktionsannahme: Aussage ist wahr für $G - v$.
* Die Aussagen (i) $\Rightarrow$ (ii), (iii), (iv), (v) sind dann offensichtlich wahr.
* Anmerkung: Für die Aussage (i) $\Rightarrow$ (iv) brauchen wir die Induktionsannahme für $G - v$ nicht.
  G ist zusammenhängend, d.h. je zwei Knoten $x, y \in V$ sind mit einem Pfad verbunden.

Beweis von Satz 2

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* Strategie: zeige Äquivalenz von (ii), (iii), (iv), (v) zu (i).
* Induktionsbeweis über die Knotenanzahl mittels Baumwachstumslemma.
* Induktionsanfang: bemerke, dass alle Äquivalenzen für Graphen mit 1 Knoten wahr sind.
* Sei G ein Baum mit mind. 2 Knoten.
  G hat mind. 2 Endknoten (Endknotenlemma); sei v einer davon, und sei v' sein Nachbar.
  Induktionsannahme: Aussage ist wahr für $G - v$.
* Die Aussagen (i) $\Rightarrow$ (ii), (iii), (iv), (v) sind dann offensichtlich wahr.
* Anmerkung: Für die Aussage (i) $\Rightarrow$ (iv) brauchen wir die Induktionsannahme für $G - v$ nicht.
  G ist zusammenhängend, d.h. je zwei Knoten $x, y \in V$ sind mit einem Pfad verbunden.
  Falls $\{x, y\} \notin E$, so ist die Kante $\{x, y\}$ zusammen mit diesem Pfad ein Kreis.

Beweis von Satz 2 (Fortsetzung)

Beweis von Satz 2 (Fortsetzung)

* **Satz 2** (Charakterisierung von Bäumen):
Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

Beweis von Satz 2 (Fortsetzung)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (ii)⇒(i): Graph ist zusammenhängend (es existiert ein Weg zwischen allen Knotenpaaren).

Beweis von Satz 2 (Fortsetzung)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (ii)⇒(i): Graph ist zusammenhängend (es existiert ein Weg zwischen allen Knotenpaaren).
  Graph enthält keinen Kreis, da es sonst Knoten x, y mit zwei verschiedenen Wegen gäbe.

Beweis von Satz 2 (Fortsetzung)

* **Satz 2** (Charakterisierung von Bäumen):
Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (ii)⇒(i): Graph ist zusammenhängend (es existiert ein Weg zwischen allen Knotenpaaren).
  Graph enthält keinen Kreis, da es sonst Knoten x, y mit zwei verschiedenen Wegen gäbe.
* (iii) ⇒ (i): Graph ist zusammenhängend (wird in (iii) bereits so gefordert).

Beweis von Satz 2 (Fortsetzung)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (ii)⇒(i): Graph ist zusammenhängend (es existiert ein Weg zwischen allen Knotenpaaren).
  Graph enthält keinen Kreis, da es sonst Knoten x, y mit zwei verschiedenen Wegen gäbe.
* (iii) ⇒ (i): Graph ist zusammenhängend (wird in (iii) bereits so gefordert).
  Graph kreisfrei, da es sonst eine Kante $\{x, y\}$ gibt, ohne die der Graph immer noch zusammenhängend wäre.

Beweis von Satz 2 (Fortsetzung)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (ii)$\Rightarrow$(i): Graph ist zusammenhängend (es existiert ein Weg zwischen allen Knotenpaaren).
  Graph enthält keinen Kreis, da es sonst Knoten x, y mit zwei verschiedenen Wegen gäbe.
* (iii) $\Rightarrow$ (i): Graph ist zusammenhängend (wird in (iii) bereits so gefordert).
  Graph kreisfrei, da es sonst eine Kante $\{x, y\}$ gibt, ohne die der Graph immer noch zusammenhängend wäre.
* (iv) $\Rightarrow$ (i): Graph ist kreisfrei (wird in (iv) bereits so gefordert).

Beweis von Satz 2 (Fortsetzung)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (ii)⇒(i): Graph ist zusammenhängend (es existiert ein Weg zwischen allen Knotenpaaren).
  Graph enthält keinen Kreis, da es sonst Knoten x, y mit zwei verschiedenen Wegen gäbe.
* (iii) ⇒ (i): Graph ist zusammenhängend (wird in (iii) bereits so gefordert).
  Graph kreisfrei, da es sonst eine Kante $\{x, y\}$ gibt, ohne die der Graph immer noch zusammenhängend wäre.
* (iv) ⇒ (i): Graph ist kreisfrei (wird in (iv) bereits so gefordert).
  Graph zusammenhängend, da für alle $\{x, y\} \notin E$ der Graph G mit $\{x, y\}$ einen Kreis enthält. Löscht man diese Kante, bleibt ein Pfad von x nach y.

Beweis von Satz 2 (Schluss)

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:

  (i) G ist ein Baum.

  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.

  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.

  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.

  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)⇒(i): Induktion über Knotenanzahl.

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.
  Die Summe aller Knotengrade ist dann $2|V| - 2$. (Warum?)

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.
  Die Summe aller Knotengrade ist dann $2|V| - 2$. (Warum?)
  Also kann nicht jeder Knoten Grad 2 haben.

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.
  Die Summe aller Knotengrade ist dann $2|V| - 2$. (Warum?)
  Also kann nicht jeder Knoten Grad 2 haben.
  Der Grad jedes Knotens ist aber mindestens 1.

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.
  Die Summe aller Knotengrade ist dann $2|V| - 2$. (Warum?)
  Also kann nicht jeder Knoten Grad 2 haben.
  Der Grad jedes Knotens ist aber mindestens 1.
  Somit gibt es einen Knoten v, dessen Grad genau 1 ist, d.h. v ist ein Endknoten.

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.
  Die Summe aller Knotengrade ist dann $2|V| - 2$. (Warum?)
  Also kann nicht jeder Knoten Grad 2 haben.
  Der Grad jedes Knotens ist aber mindestens 1.
  Somit gibt es einen Knoten v, dessen Grad genau 1 ist, d.h. v ist ein Endknoten.
  Der Graph $G' := G - v$ ist auch zusammenhängend, und erfüllt $|V(G')| = |E(G')| + 1$.

Beweis von Satz 2 (Schluss)

* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.
  Die Summe aller Knotengrade ist dann $2|V| - 2$. (Warum?)
  Also kann nicht jeder Knoten Grad 2 haben.
  Der Grad jedes Knotens ist aber mindestens 1.
  Somit gibt es einen Knoten v, dessen Grad genau 1 ist, d.h. v ist ein Endknoten.
  Der Graph $G' := G - v$ ist auch zusammenhängend, und erfüllt $|V(G')| = |E(G')| + 1$.
  Nach Induktionsannahme ist G' daher ein Baum.

Beweis von Satz 2 (Schluss)

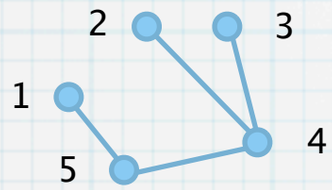
* **Satz 2** (Charakterisierung von Bäumen):
  Sei $G = (V, E)$ ein Graph. Die folgenden Aussagen sind äquivalent für G:
  (i) G ist ein Baum.
  (ii) Pfade in G sind eindeutig, d.h. für je zwei Knoten $x, y \in V$ gibt es genau einen Pfad von x nach y.
  (iii) G ist ein minimaler zusammenhängender Graph, d.h. G ist zusammenhängend, aber die Löschung einer beliebigen Kante führt zu einem unzusammenhängenden Graphen.
  (iv) G ist ein maximaler Graph ohne Kreise, d.h. G enthält keinen Kreis, und jeder Graph, der aus G durch Hinzufügen einer weiteren Kante entsteht, enthält einen Kreis.
  (v) G ist zusammenhängend und es gilt Eulers Formel: $|V| = |E| + 1$.

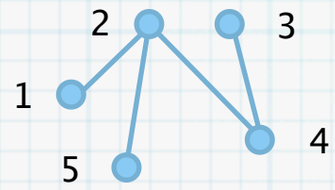
* (v)$\Rightarrow$(i): Induktion über Knotenanzahl.
  Sei G zusammenhängend mit $|V| = |E| + 1 \geq 2$.
  Die Summe aller Knotengrade ist dann $2|V| - 2$. (Warum?)
  Also kann nicht jeder Knoten Grad 2 haben.
  Der Grad jedes Knotens ist aber mindestens 1.
  Somit gibt es einen Knoten v, dessen Grad genau 1 ist, d.h. v ist ein Endknoten.
  Der Graph $G' := G - v$ ist auch zusammenhängend, und erfüllt $|V(G')| = |E(G')| + 1$.
  Nach Induktionsannahme ist G' daher ein Baum.
  Folglich ist auch G ein Baum (Baumwachstumslemma).

Wieviele verschiedene Bäume auf n Knoten gibt es?

Wieviele verschiedene Bäume auf n Knoten gibt es?

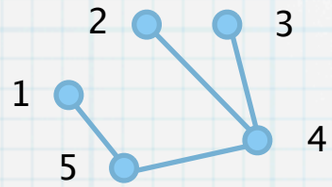
* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

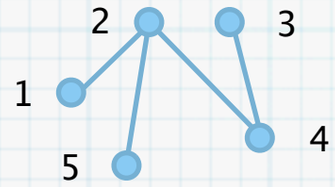




Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

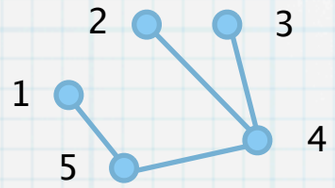


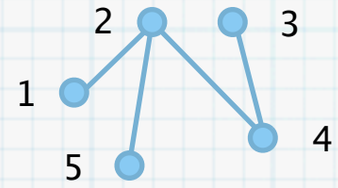


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

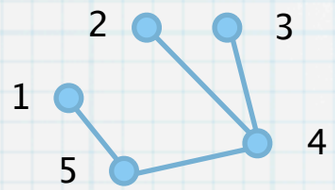


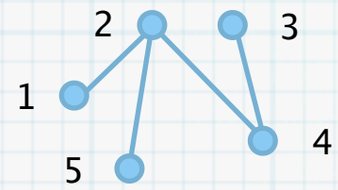


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

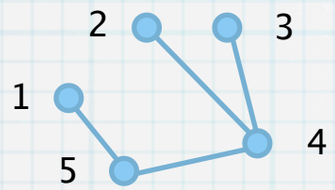


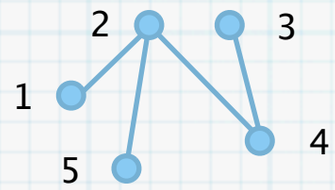


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.
* Bei Möglichkeit A sprechen wir von „markierten (indizierten) Bäumen“, bei B von „unmarkierten (unindizierten) Bäumen“

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

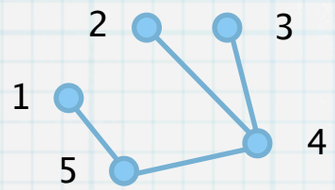


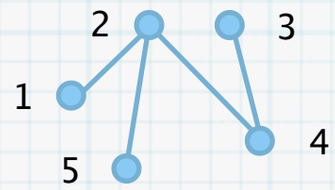


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.
* Bei Möglichkeit A sprechen wir von „markierten (indizierten) Bäumen“, bei B von „unmarkierten (unindizierten) Bäumen“
* **Satz 7** (Satz von Cayley, 1889):
  Für n Knoten gibt es genau n^{n-2} markierte Bäume.

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

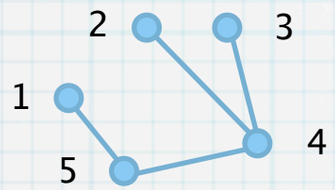


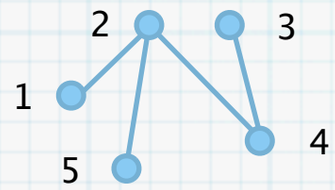


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.
* Bei Möglichkeit A sprechen wir von „markierten (indizierten) Bäumen“, bei B von „unmarkierten (unindizierten) Bäumen“
* **Satz 7** (Satz von Cayley, 1889):
  Für n Knoten gibt es genau n^{n-2} markierte Bäume.
* Im Falle von unmarkierten Bäumen ist keine exakte Formel bekannt, lediglich Abschätzungen.

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

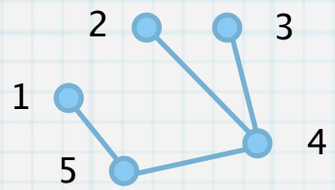


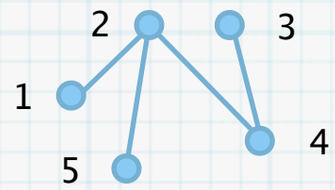


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.
* Bei Möglichkeit A sprechen wir von „markierten (indizierten) Bäumen“, bei B von „unmarkierten (unindizierten) Bäumen“
* **Satz 7** (Satz von Cayley, 1889):
  Für n Knoten gibt es genau n^{n-2} markierte Bäume.
* Im Falle von unmarkierten Bäumen ist keine exakte Formel bekannt, lediglich Abschätzungen.
* Für den Satz von Cayley gibt es verschiedene Beweise.

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

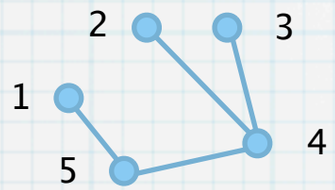


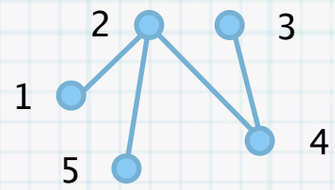


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.
* Bei Möglichkeit A sprechen wir von „markierten (indizierten) Bäumen“, bei B von „unmarkierten (unindizierten) Bäumen“
* **Satz 7** (Satz von Cayley, 1889):
  Für n Knoten gibt es genau n^{n-2} markierte Bäume.
* Im Falle von unmarkierten Bäumen ist keine exakte Formel bekannt, lediglich Abschätzungen.
* Für den Satz von Cayley gibt es verschiedene Beweise.
  Im Folgenden stellen wir einen Beweis von Prüfer (1916) vor.

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:

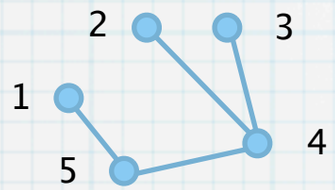


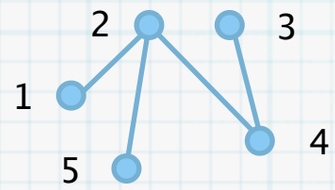


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.
* Bei Möglichkeit A sprechen wir von „markierten (indizierten) Bäumen“, bei B von „unmarkierten (unindizierten) Bäumen“
* **Satz 7** (Satz von Cayley, 1889):
  Für n Knoten gibt es genau n^{n-2} markierte Bäume.
* Im Falle von unmarkierten Bäumen ist keine exakte Formel bekannt, lediglich Abschätzungen.
* Für den Satz von Cayley gibt es verschiedene Beweise.
  Im Folgenden stellen wir einen Beweis von Prüfer (1916) vor.
  Essenziell in seinem Beweis ist es, einen Baum geschickt „abzuspeichern“, zu kodieren.

Wieviele verschiedene Bäume auf n Knoten gibt es?

* Wann bezeichnen wir zwei Bäume als gleich? Beispiel:



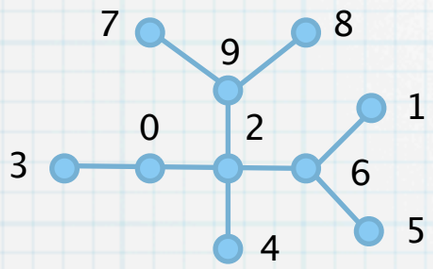


* **Möglichkeit A**: Zwei Bäume sind gleich, wenn in beiden die gleichen Knoten benachbart sind.
* **Möglichkeit B**: Zwei Bäume sind gleich, wenn man eine eineindeutige (bijektive) Abbildung zwischen den Knoten definieren kann, so dass zwei Urbildknoten genau dann benachbart sind, wenn die entsprechenden Bildknoten auch benachbart sind.
* Bei Möglichkeit A sprechen wir von „markierten (indizierten) Bäumen“, bei B von „unmarkierten (unindizierten) Bäumen“
* **Satz 7** (Satz von Cayley, 1889):
  Für n Knoten gibt es genau n^{n-2} markierte Bäume.
* Im Falle von unmarkierten Bäumen ist keine exakte Formel bekannt, lediglich Abschätzungen.
* Für den Satz von Cayley gibt es verschiedene Beweise.
  Im Folgenden stellen wir einen Beweis von Prüfer (1916) vor.
  Essenziell in seinem Beweis ist es, einen Baum geschickt „abzuspeichern“, zu kodieren.
  Man spricht daher auch von der „Prüfer-Kodierung“ eines Baumes.

Wie speichert man einen Baum ab (z.B. im Rechner)?

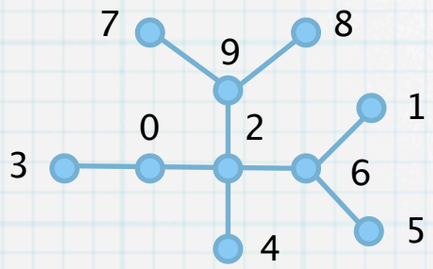
Wie speichert man einen Baum ab (z.B. im Rechner)?

* Beispiel: Baum G mit n Knoten



Wie speichert man einen Baum ab (z.B. im Rechner)?

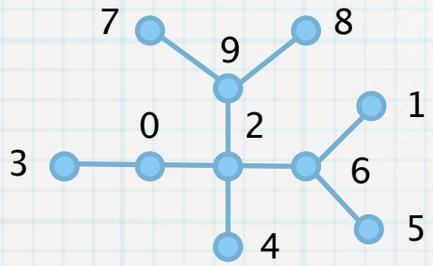
* Beispiel: Baum G mit n Knoten



* **Definition 8**:
Eine **Adjazenzmatrix** eines Graphen G mit n Knoten ist eine $n \times n$ Matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ mit Einträgen $a_{ij} \in \{0, 1\}$, wobei $a_{ij} = 1$ genau dann, wenn Kante $\{i, j\}$ zu G gehört.

Wie speichert man einen Baum ab (z.B. im Rechner)?

* Beispiel: Baum G mit n Knoten

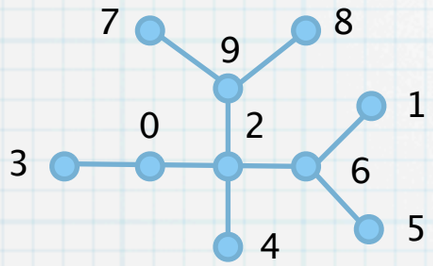


* **Definition 8**:
Eine **Adjazenzmatrix** eines Graphen G mit n Knoten ist eine $n \times n$ Matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ mit Einträgen $a_{ij} \in \{0, 1\}$, wobei $a_{ij} = 1$ genau dann, wenn Kante $\{i, j\}$ zu G gehört.
* Beispiel: Die Adjazenzmatrix zu obigem Baum ist gegeben durch

$$A := \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Wie speichert man einen Baum ab (z.B. im Rechner)?

* Beispiel: Baum G mit n Knoten



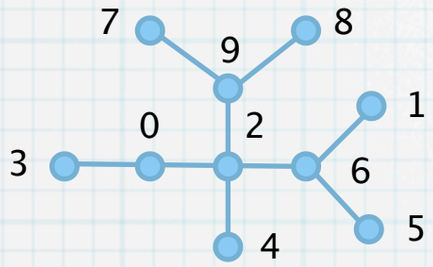
* **Definition 8**:
Eine **Adjazenzmatrix** eines Graphen G mit n Knoten ist eine $n \times n$ Matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ mit Einträgen $a_{ij} \in \{0, 1\}$, wobei $a_{ij} = 1$ genau dann, wenn Kante $\{i, j\}$ zu G gehört.
* Beispiel: Die Adjazenzmatrix zu obigem Baum ist gegeben durch

$$A := \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

* Speicherbedarf:

Wie speichert man einen Baum ab (z.B. im Rechner)?

* Beispiel: Baum G mit n Knoten



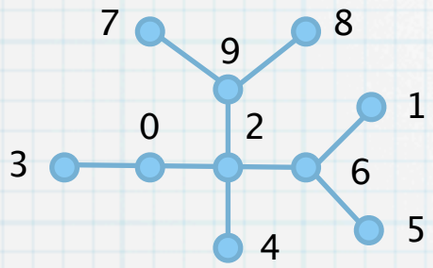
* **Definition 8**:
Eine **Adjazenzmatrix** eines Graphen G mit n Knoten ist eine $n \times n$ Matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ mit Einträgen $a_{ij} \in \{0, 1\}$, wobei $a_{ij} = 1$ genau dann, wenn Kante $\{i, j\}$ zu G gehört.
* Beispiel: Die Adjazenzmatrix zu obigem Baum ist gegeben durch

$$A := \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

* Speicherbedarf:
n^2 Bits für volle Matrix.

Wie speichert man einen Baum ab (z.B. im Rechner)?

* Beispiel: Baum G mit n Knoten



* **Definition 8**:
Eine **Adjazenzmatrix** eines Graphen G mit n Knoten ist eine $n \times n$ Matrix $A = (a_{ij})_{1 \leq i,j \leq n}$ mit Einträgen $a_{ij} \in \{0, 1\}$, wobei $a_{ij} = 1$ genau dann, wenn Kante $\{i, j\}$ zu G gehört.
* Beispiel: Die Adjazenzmatrix zu obigem Baum ist gegeben durch

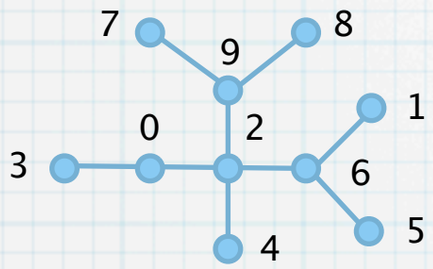
$$A := \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

* Speicherbedarf:
n^2 Bits für volle Matrix.
Es genügen $(n^2 - n)/2$ Bits (da Matrix symmetrisch, und 0 auf Diagonale).

Abspeichern von Bäumen als Kantenliste

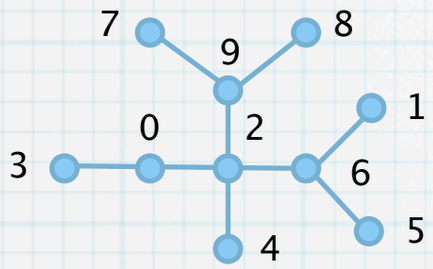
Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten

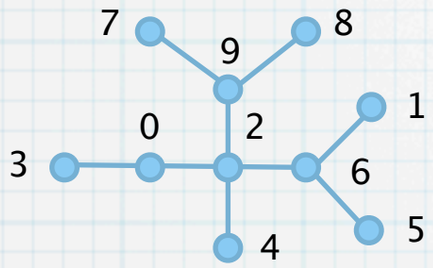


* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



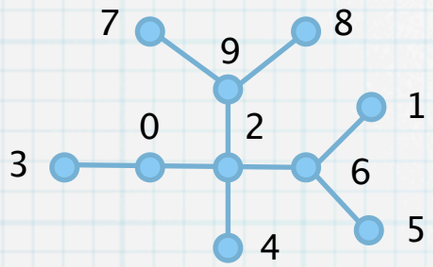
* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

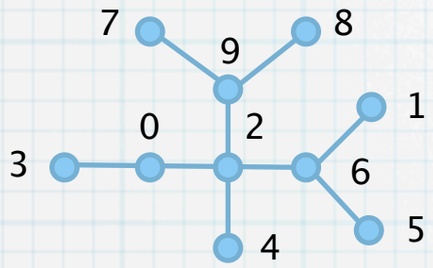
$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:

  Je Eintrag $\lceil \log_2 n \rceil$ Bits.

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

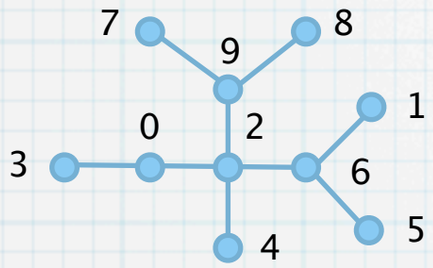
* Speicherbedarf:

  Je Eintrag $\lceil \log_2 n \rceil$ Bits.

  Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:

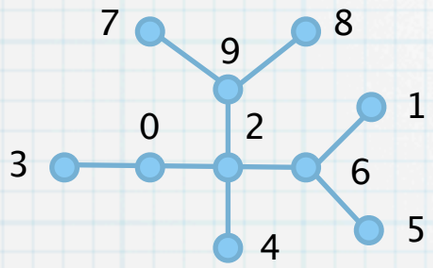
  Je Eintrag $\lceil \log_2 n \rceil$ Bits.

  Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.

  Für große n ist $2(n-1)\lceil \log_2 n \rceil \ll (n^2 - n)/2$.

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:

  Je Eintrag $\lceil \log_2 n \rceil$ Bits.

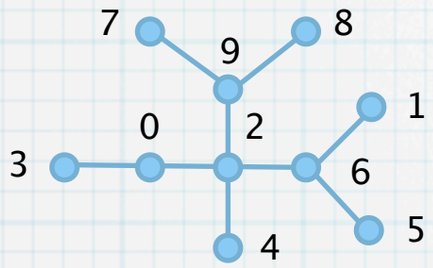
  Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.

  Für große n ist $2(n-1)\lceil \log_2 n \rceil \ll (n^2 - n)/2$.

* Diese Form der Speicherung ist nicht eindeutig:

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:

  Je Eintrag $\lceil \log_2 n \rceil$ Bits.

  Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.

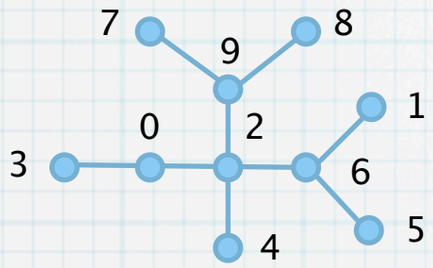
  Für große n ist $2(n-1)\lceil \log_2 n \rceil \ll (n^2 - n)/2$.

* Diese Form der Speicherung ist nicht eindeutig:

  Reihenfolge der Kanten in der Liste.

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



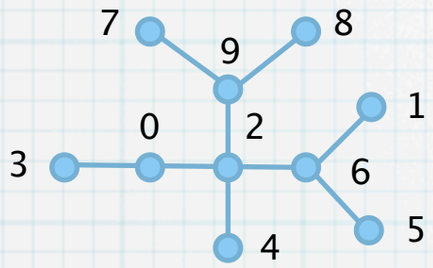
* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:
  - Je Eintrag $\lceil \log_2 n \rceil$ Bits.
  - Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.
  - Für große n ist $2(n-1)\lceil \log_2 n \rceil \ll (n^2 - n)/2$.
* Diese Form der Speicherung ist nicht eindeutig:
  - Reihenfolge der Kanten in der Liste.
  - Reihenfolge der Knoten je Spalte.

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



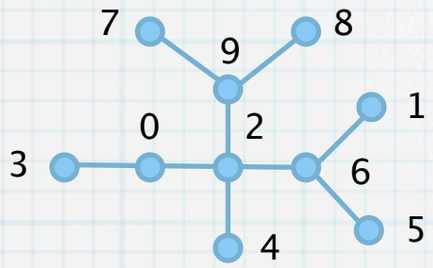
* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:
  - Je Eintrag $\lceil \log_2 n \rceil$ Bits.
  - Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.
  - Für große n ist $2(n-1)\lceil \log_2 n \rceil \ll (n^2 - n)/2$.
* Diese Form der Speicherung ist nicht eindeutig:
  - Reihenfolge der Kanten in der Liste.
  - Reihenfolge der Knoten je Spalte.
* Nächstes Ziel: Modifiziere Speicherung so, dass...

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



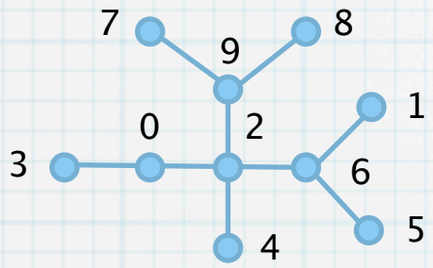
* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:
  - Je Eintrag $\lceil \log_2 n \rceil$ Bits.
  - Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.
  - Für große n ist $2(n-1)\lceil \log_2 n \rceil \ll (n^2 - n)/2$.
* Diese Form der Speicherung ist nicht eindeutig:
  - Reihenfolge der Kanten in der Liste.
  - Reihenfolge der Knoten je Spalte.
* Nächstes Ziel: Modifiziere Speicherung so, dass...
  - ...sie eindeutig ist und

Abspeichern von Bäumen als Kantenliste

* Beispiel: Baum G mit n Knoten



* Die Spalten einer $2 \times (n-1)$ Matrix mit Einträgen in $\{0, 1, \ldots, n-1\}$ entsprechen den Kanten:

$$\begin{pmatrix} 7 & 8 & 9 & 6 & 3 & 0 & 2 & 6 & 6 \\ 9 & 9 & 2 & 2 & 0 & 2 & 4 & 1 & 5 \end{pmatrix}$$

* Speicherbedarf:
  - Je Eintrag $\lceil \log_2 n \rceil$ Bits.
  - Insgesamt also $2(n-1)\lceil \log_2 n \rceil$ Bits.
  - Für große n ist $2(n-1)\lceil \log_2 n \rceil \ll (n^2 - n)/2$.
* Diese Form der Speicherung ist nicht eindeutig:
  - Reihenfolge der Kanten in der Liste.
  - Reihenfolge der Knoten je Spalte.
* Nächstes Ziel: Modifiziere Speicherung so, dass...
  - ...sie eindeutig ist und
  - ...außerdem Speicherplatz spart.

Die „Stammbaum“-Kodierung

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.

  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.

  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.

  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.

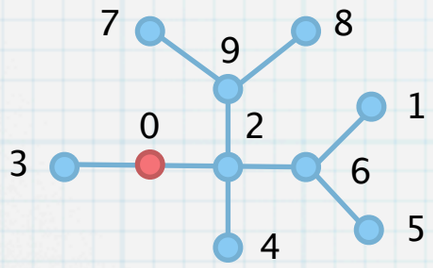
Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.

  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.

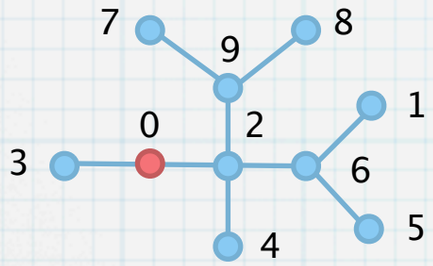
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.

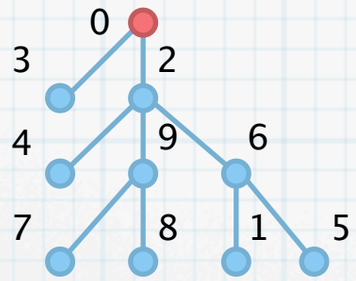
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



Die „Stammbaum“-Kodierung

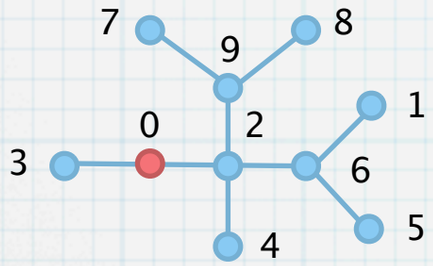
* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

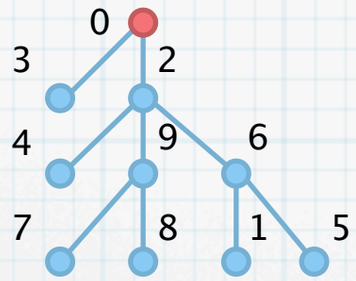




Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

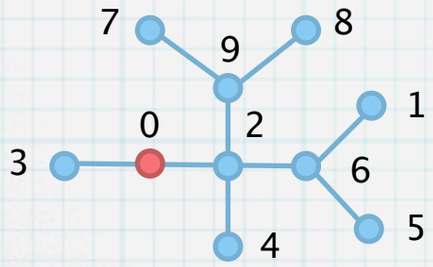


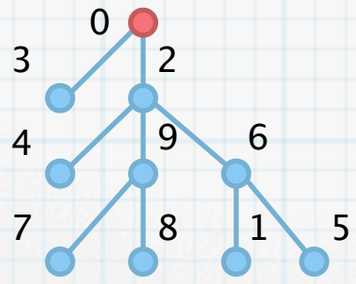


$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



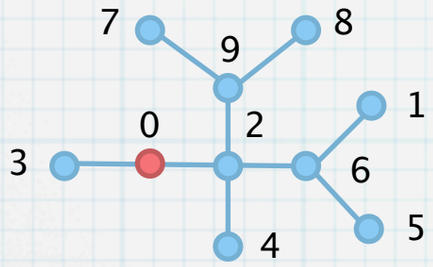


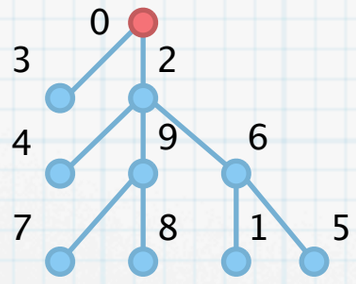
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



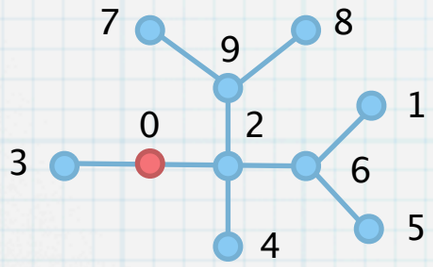


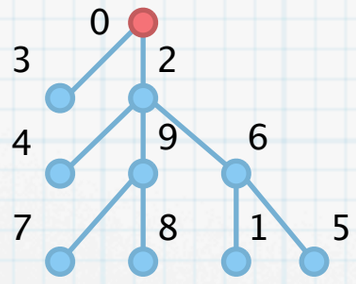
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



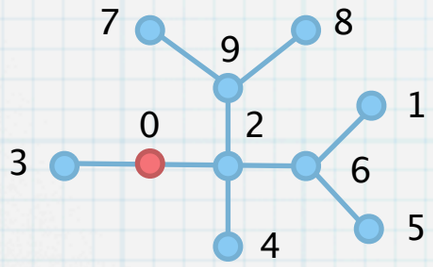


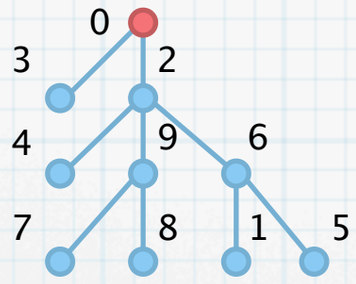
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



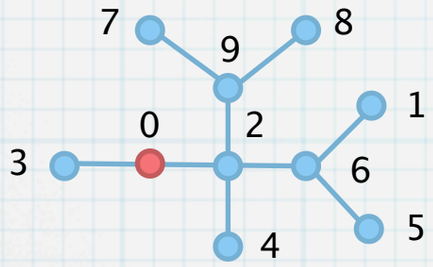


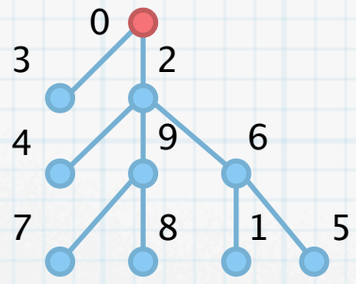
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.
  Keine Zahl kommt mehrfach vor, da jedes Kind von genau einem Elternknoten abstammt.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



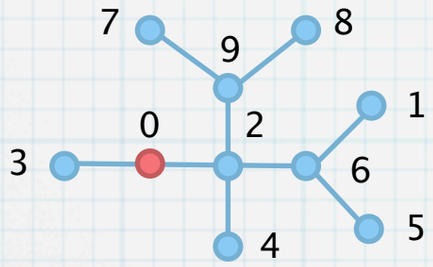


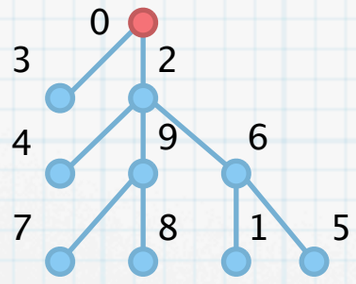
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.
  Keine Zahl kommt mehrfach vor, da jedes Kind von genau einem Elternknoten abstammt.
* Speicherbedarf:

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



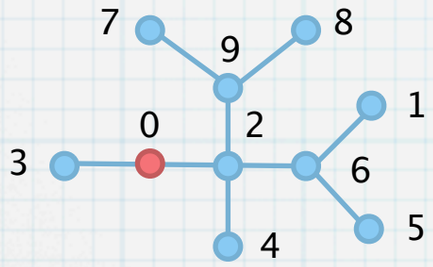


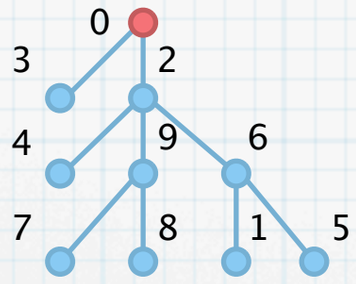
$$\begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} \\ \mathbf{6} & \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{6} & \mathbf{2} & \mathbf{9} & \mathbf{9} & \mathbf{2} \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.
  Keine Zahl kommt mehrfach vor, da jedes Kind von genau einem Elternknoten abstammt.
* Speicherbedarf:
  Erste Zeile enthält die Zahlen von 1 bis n in dieser Reihenfolge. Speicherung nicht nötig.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



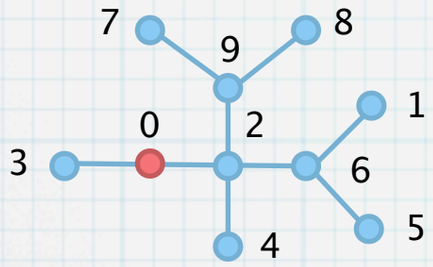


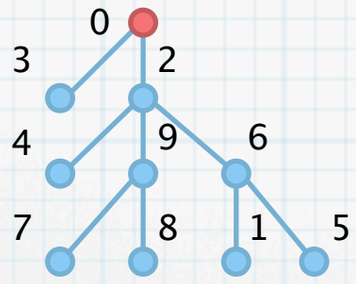
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.
  Keine Zahl kommt mehrfach vor, da jedes Kind von genau einem Elternknoten abstammt.
* Speicherbedarf:
  Erste Zeile enthält die Zahlen von 1 bis n in dieser Reihenfolge. Speicherung nicht nötig.
  Speicherplatz für zweite Zeile = Speicherbedarf insgesamt = $(n-1)\lceil \log_2 n \rceil$.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



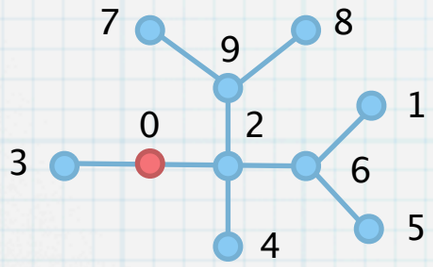


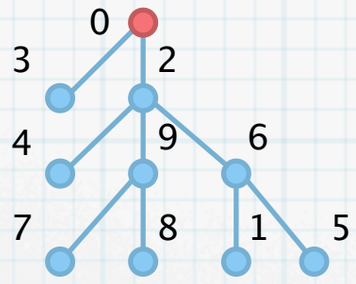
$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.
  Keine Zahl kommt mehrfach vor, da jedes Kind von genau einem Elternknoten abstammt.
* Speicherbedarf:
  Erste Zeile enthält die Zahlen von 1 bis n in dieser Reihenfolge. Speicherung nicht nötig.
  Speicherplatz für zweite Zeile = Speicherbedarf insgesamt = $(n-1)\lceil \log_2 n \rceil$.
* Aber auch die Stammbaum-Kodierung ist noch nicht optimal.

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



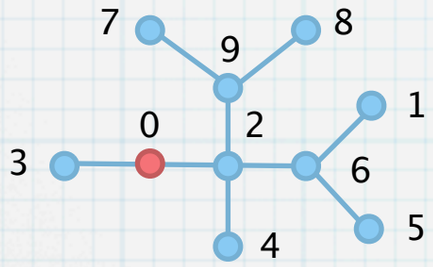


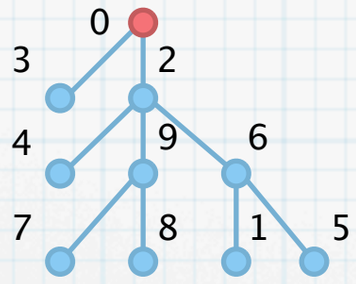
$$\begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{9} \\ \mathbf{6} & \mathbf{0} & \mathbf{0} & \mathbf{2} & \mathbf{6} & \mathbf{2} & \mathbf{9} & \mathbf{9} & \mathbf{2} \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.
  Keine Zahl kommt mehrfach vor, da jedes Kind von genau einem Elternknoten abstammt.
* Speicherbedarf:
  Erste Zeile enthält die Zahlen von 1 bis n in dieser Reihenfolge. Speicherung nicht nötig.
  Speicherplatz für zweite Zeile = Speicherbedarf insgesamt = $(n-1)\lceil \log_2 n \rceil$.
* Aber auch die Stammbaum-Kodierung ist noch nicht optimal.
  Nicht jeder Kode ergibt auch einen (Stamm-) Baum!

Die „Stammbaum“-Kodierung

* Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Daher auch der Name „Stammbaum“-Kodierung: unten stehen die „Eltern“, oben das „Kind“.
  Die Kanten werden der Größe des ersten Knotens nach aufsteigend sortiert.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0





$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

* Die Einträge in der ersten Zeile der Matrix sind kein Zufall, denn:
  Knoten 0 kommt nicht dort vor, da es nicht Kind eines anderen Knoten ist.
  Jede andere Zahl kommt vor, da jedes Kind (oben) Eltern (unten) hat.
  Keine Zahl kommt mehrfach vor, da jedes Kind von genau einem Elternknoten abstammt.
* Speicherbedarf:
  Erste Zeile enthält die Zahlen von 1 bis n in dieser Reihenfolge. Speicherung nicht nötig.
  Speicherplatz für zweite Zeile = Speicherbedarf insgesamt = $(n-1)\lceil \log_2 n \rceil$.
* Aber auch die Stammbaum-Kodierung ist noch nicht optimal.
  Nicht jeder Kode ergibt auch einen (Stamm-) Baum!
  Wir können die Stammbaum-Kodierung aber noch etwas verfeinern...

Die Prüfer-Kodierung

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).

Die Prüfer-Kodierung

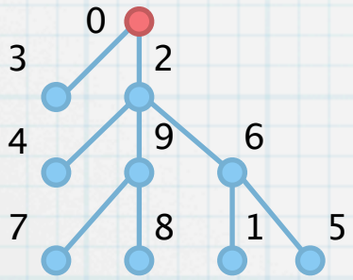
* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.

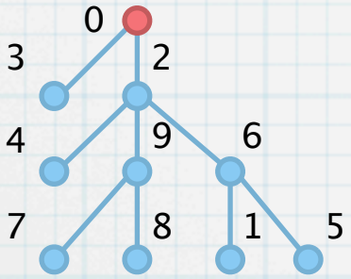
Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



Die Prüfer-Kodierung

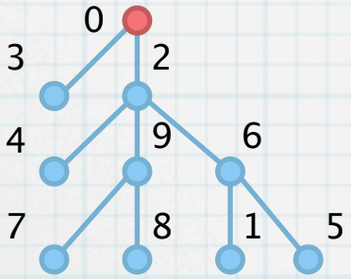
* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



$($ $)$

Die Prüfer-Kodierung

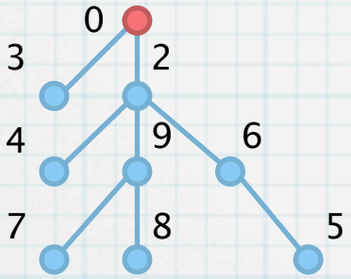
* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



$$\begin{pmatrix} \mathbf{1} & & \\ \mathbf{6} & & \end{pmatrix}$$

Die Prüfer-Kodierung

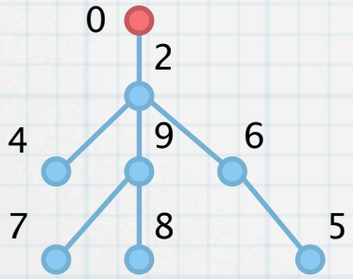
* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



$$\begin{pmatrix} \mathbf{1} & & \\ \mathbf{6} & & \end{pmatrix}$$

Die Prüfer-Kodierung

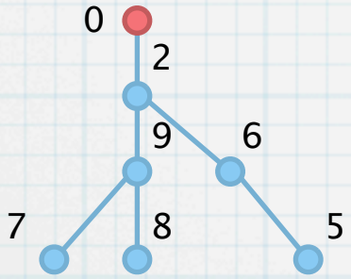
* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



$$\begin{pmatrix} \mathbf{1} & \mathbf{3} \\ \mathbf{6} & \mathbf{0} \end{pmatrix}$$

Die Prüfer-Kodierung

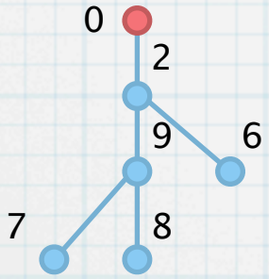
* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



$$\begin{pmatrix} \mathbf{1} & \mathbf{3} & \mathbf{4} \\ \mathbf{6} & \mathbf{0} & \mathbf{2} \end{pmatrix}$$

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0



$$\begin{pmatrix} \mathbf{1} & \mathbf{3} & \mathbf{4} & \mathbf{5} \\ \mathbf{6} & \mathbf{0} & \mathbf{2} & \mathbf{6} \end{pmatrix}$$

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} \mathbf{1} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} \\ \mathbf{6} & \mathbf{0} & \mathbf{2} & \mathbf{6} & \mathbf{2} \end{pmatrix}$$

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} \mathbf{1} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} \\ \mathbf{6} & \mathbf{0} & \mathbf{2} & \mathbf{6} & \mathbf{2} & \mathbf{9} \end{pmatrix}$$

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} \mathbf{1} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} \\ \mathbf{6} & \mathbf{0} & \mathbf{2} & \mathbf{6} & \mathbf{2} & \mathbf{9} & \mathbf{9} \end{pmatrix}$$

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 \end{pmatrix}$$

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer–Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{pmatrix}$$

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer–Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{pmatrix}$$

* Bemerke, dass der rechte, untere Eintrag immer die Wurzel (Knoten 0) ist:

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer–Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{pmatrix}$$

* Bemerke, dass der rechte, untere Eintrag immer die Wurzel (Knoten 0) ist:
  Wurzel wurde bei der Sortierung der Kanten in jeder Runde ausgelassen.

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer–Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{pmatrix}$$

* Bemerke, dass der rechte, untere Eintrag immer die Wurzel (Knoten 0) ist:
    Wurzel wurde bei der Sortierung der Kanten in jeder Runde ausgelassen.
    Letzte Kante muss also mit Wurzel inzident sein.

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer–Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{pmatrix}$$

* Bemerke, dass der rechte, untere Eintrag immer die Wurzel (Knoten 0) ist:
    Wurzel wurde bei der Sortierung der Kanten in jeder Runde ausgelassen.
    Letzte Kante muss also mit Wurzel inzident sein.
* Vorteil der erweiterten Prüfer–Kodierung: brauche rechten, unteren Eintrag nicht speichern.

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer-Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{pmatrix}$$

* Bemerke, dass der rechte, untere Eintrag immer die Wurzel (Knoten 0) ist:
    Wurzel wurde bei der Sortierung der Kanten in jeder Runde ausgelassen.
    Letzte Kante muss also mit Wurzel inzident sein.
* Vorteil der erweiterten Prüfer-Kodierung: brauche rechten, unteren Eintrag nicht speichern.
* Aber: erste Zeile nicht mehr sortiert!

Die Prüfer-Kodierung

* Zunächst die erweiterte Prüfer–Kodierung:
  Knoten 0 wird ausgezeichnet als „Wurzelknoten“.
  Für jede Kante wird der Knoten zuerst genommen, der weiter von der Wurzel weg ist.
  Sortierung der Kanten:
    Suche Knoten mit Grad 1 (Endknoten) und kleinster Knotennummer (verschieden Wurzel).
    Lösche diesen Knoten und die zugehörige Kante.
    Wiederhole diese Schritte, bis alle Knoten weg sind.
* Beispiel: Baum G mit n Knoten und ausgezeichnetem Wurzelknoten 0

$$\begin{pmatrix} 1 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 2 \\ 6 & 0 & 2 & 6 & 2 & 9 & 9 & 2 & 0 \end{pmatrix}$$

* Bemerke, dass der rechte, untere Eintrag immer die Wurzel (Knoten 0) ist:
    Wurzel wurde bei der Sortierung der Kanten in jeder Runde ausgelassen.
    Letzte Kante muss also mit Wurzel inzident sein.
* Vorteil der erweiterten Prüfer–Kodierung: brauche rechten, unteren Eintrag nicht speichern.
* Aber: erste Zeile nicht mehr sortiert!
* War der Preis zu hoch?

Prüfers Lemma

Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.

Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.

( )

Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
* Wer ist das Kind von Knoten 2?

Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
* Wer ist das Kind von Knoten 2?
  * 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.

Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
* Wer ist das Kind von Knoten 2?
  * 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  * 2? Schleifen sind verboten!

Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
* Wer ist das Kind von Knoten 2?
  * 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  * 2? Schleifen sind verboten!
  * 3? Nein, siehe 1.

Prüfers Lemma

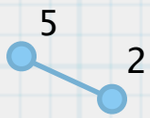
* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
* Wer ist das Kind von Knoten 2?
  * 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  * 2? Schleifen sind verboten!
  * 3? Nein, siehe 1.
  * 4? Nein, siehe 1.

Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
* Wer ist das Kind von Knoten 2?
  * 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  * 2? Schleifen sind verboten!
  * 3? Nein, siehe 1.
  * 4? Nein, siehe 1.
  * 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.

Prüfers Lemma

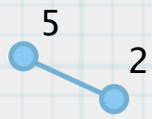
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.



$$\begin{pmatrix} \mathbf{5} & & \\ \mathbf{2} & & \end{pmatrix}$$

Prüfers Lemma

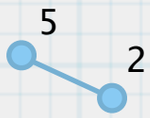
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?



$$\begin{pmatrix} 5 & & \\ 2 & & \end{pmatrix}$$

Prüfers Lemma

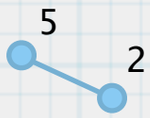
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.



$$\begin{pmatrix} 5 & & \\ 2 & & \end{pmatrix}$$

Prüfers Lemma

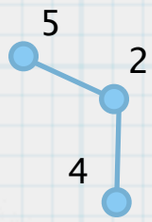
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.
  - 2? Ja, denn jetzt ist Knoten 2 der kleinste, der später nicht mehr vorkommt.



$$\begin{pmatrix} \mathbf{5} & & \\ \mathbf{2} & & \end{pmatrix}$$

Prüfers Lemma

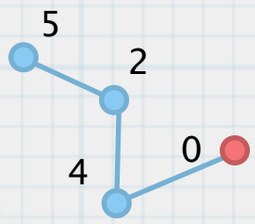
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.
  - 2? Ja, denn jetzt ist Knoten 2 der kleinste, der später nicht mehr vorkommt.
- usw.



$$\begin{pmatrix} \mathbf{5} & \mathbf{2} \\ \mathbf{2} & 4 \end{pmatrix}$$

Prüfers Lemma

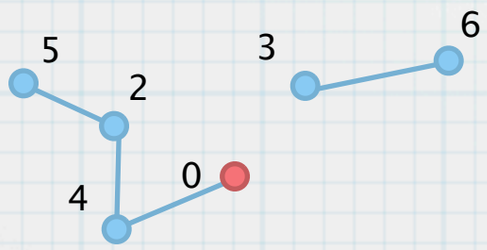
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.
  - 2? Ja, denn jetzt ist Knoten 2 der kleinste, der später nicht mehr vorkommt.
- usw.



$$\begin{pmatrix} 5 & 2 & 4 \\ 2 & 4 & 0 \end{pmatrix}$$

Prüfers Lemma

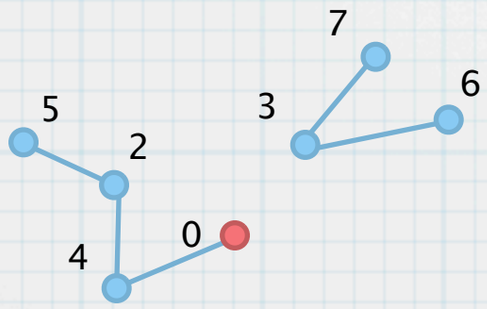
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.
  - 2? Ja, denn jetzt ist Knoten 2 der kleinste, der später nicht mehr vorkommt.
- usw.



$$\begin{pmatrix} 5 & 2 & 4 & 6 \\ 2 & 4 & 0 & 3 \end{pmatrix}$$

Prüfers Lemma

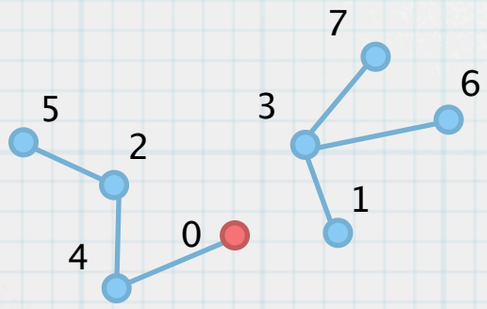
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.
  - 2? Ja, denn jetzt ist Knoten 2 der kleinste, der später nicht mehr vorkommt.
- usw.



$$\begin{pmatrix} 5 & 2 & 4 & 6 & 7 \\ 2 & 4 & 0 & 3 & 3 \end{pmatrix}$$

Prüfers Lemma

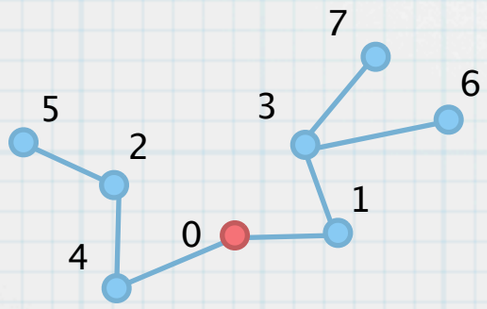
- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.
  - 2? Ja, denn jetzt ist Knoten 2 der kleinste, der später nicht mehr vorkommt.
- usw.



$$\begin{pmatrix} \mathbf{5} & \mathbf{2} & \mathbf{4} & \mathbf{6} & \mathbf{7} & \mathbf{3} \\ \mathbf{2} & \mathbf{4} & \mathbf{0} & \mathbf{3} & \mathbf{3} & \mathbf{1} \end{pmatrix}$$

Prüfers Lemma

- **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
- Beispiel: was ist der Baum zu (2,4,0,3,3,1,0) ? Klar: Baum hat 8 Knoten.
- Wer ist das Kind von Knoten 2?
  - 1? Nein, denn dann wäre Knoten 1 gelöscht und käme nicht mehr als Vater vor.
  - 2? Schleifen sind verboten!
  - 3? Nein, siehe 1.
  - 4? Nein, siehe 1.
  - 5? Ja, denn Knoten 5 ist der kleinste, der später (rechts) nicht mehr vorkommt.
- Nächster Eintrag: wer ist das Kind von Knoten 4, nachdem 5 gelöscht wurde?
  - 1? Nein, siehe oben.
  - 2? Ja, denn jetzt ist Knoten 2 der kleinste, der später nicht mehr vorkommt.
- usw.



$$\begin{pmatrix} 5 & 2 & 4 & 6 & 7 & 3 & 1 \\ 2 & 4 & 0 & 3 & 3 & 1 & 0 \end{pmatrix}$$

Beweis von Prüfers Lemma

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:

  Beobachtung aus vorherigem Beispiel:

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:

  Beobachtung aus vorherigem Beispiel:

  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:

  Beobachtung aus vorherigem Beispiel:

  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,
  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.
  Begründung:

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,
  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.
  Begründung:
  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,
  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.
  Begründung:
  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,
  dann sind alle vorherigen Knoten in der ersten Zeile gelöscht,

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,
  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.
  Begründung:
  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,
  dann sind alle vorherigen Knoten in der ersten Zeile gelöscht,
  ebenso wie die dazugehörigen Kanten der ersten $k-1$ Spalten.

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,
  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.
  Begründung:
  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,
  dann sind alle vorherigen Knoten in der ersten Zeile gelöscht,
  ebenso wie die dazugehörigen Kanten der ersten $k-1$ Spalten.
  Die verbleibenden Einträge der zweiten Zeile sind zu diesem Zeitpunkt Elternknoten,

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:

  Beobachtung aus vorherigem Beispiel:

  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Begründung:

  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,
  dann sind alle vorherigen Knoten in der ersten Zeile gelöscht,
  ebenso wie die dazugehörigen Kanten der ersten $k-1$ Spalten.
  Die verbleibenden Einträge der zweiten Zeile sind zu diesem Zeitpunkt Elternknoten,
  also insbesondere (noch nicht) Endknoten.

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:

  Beobachtung aus vorherigem Beispiel:

  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Begründung:

  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,
  dann sind alle vorherigen Knoten in der ersten Zeile gelöscht,
  ebenso wie die dazugehörigen Kanten der ersten $k-1$ Spalten.
  Die verbleibenden Einträge der zweiten Zeile sind zu diesem Zeitpunkt Elternknoten,
  also insbesondere (noch nicht) Endknoten.

Damit haben wir also einen Algorithmus („finde den kleinsten Knoten, der...“), der aus jeder beliebigen zweiten Zeile stets in eindeutiger Weise eine erste Zeile konstruiert.

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,
  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.
  Begründung:
  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,
  dann sind alle vorherigen Knoten in der ersten Zeile gelöscht,
  ebenso wie die dazugehörigen Kanten der ersten $k-1$ Spalten.
  Die verbleibenden Einträge der zweiten Zeile sind zu diesem Zeitpunkt Elternknoten,
  also insbesondere (noch nicht) Endknoten.
  Damit haben wir also einen Algorithmus („finde den kleinsten Knoten, der...“), der aus jeder beliebigen zweiten Zeile stets in eindeutiger Weise eine erste Zeile konstruiert.
* Wir können also Bäume mit nur noch $(n-2)\lceil \log_2 n \rceil$ Bits speichern.

Beweis von Prüfers Lemma

* **Lemma 9** (Baumrekonstruktionslemma):
  Die zweite Zeile einer erweiterten Prüfer-Kodierung bestimmt eindeutig die erste.
* Beweis:
  Beobachtung aus vorherigem Beispiel:
  Eintrag in erster Zeile der erweiterten Prüfer-Kodierung ist der kleinste Knoten, der
  ... nicht schon vorher in der ersten Zeile vorkam,
  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.
  Begründung:
  Wenn dieser Eintrag (nennen wir ihn k) in der ersten Zeile aufgezeichnet wird,
  dann sind alle vorherigen Knoten in der ersten Zeile gelöscht,
  ebenso wie die dazugehörigen Kanten der ersten $k-1$ Spalten.
  Die verbleibenden Einträge der zweiten Zeile sind zu diesem Zeitpunkt Elternknoten,
  also insbesondere (noch nicht) Endknoten.
  Damit haben wir also einen Algorithmus („finde den kleinsten Knoten, der...“), der aus jeder beliebigen zweiten Zeile stets in eindeutiger Weise eine erste Zeile konstruiert.
* Wir können also Bäume mit nur noch $(n-2)\lceil \log_2 n \rceil$ Bits speichern.
* Das entscheidende an dieser Sache kommt aber erst noch...

Optimalität der Prüfer-Kodierung

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

  Schritt 2: Diese Matrix ist nun die erweiterte Prüfer-Kodierung eines Baumes mit n Knoten.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

  Schritt 2: Diese Matrix ist nun die erweiterte Prüfer-Kodierung eines Baumes mit n Knoten.

  Zusammenhang: von jedem Knoten v kommen wir zur Wurzel. Denn:

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

  Schritt 2: Diese Matrix ist nun die erweiterte Prüfer-Kodierung eines Baumes mit n Knoten.

  Zusammenhang: von jedem Knoten v kommen wir zur Wurzel. Denn:

  Knoten v ist nach Konstruktion irgendwo Kindknoten (also oben).

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

  Schritt 2: Diese Matrix ist nun die erweiterte Prüfer-Kodierung eines Baumes mit n Knoten.

  Zusammenhang: von jedem Knoten v kommen wir zur Wurzel. Denn:

  Knoten v ist nach Konstruktion irgendwo Kindknoten (also oben).

  Sein Elternknoten (unten) wird nach Konstruktion irgendwo rechts wiederum als Kind vorkommen.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

  Schritt 2: Diese Matrix ist nun die erweiterte Prüfer-Kodierung eines Baumes mit n Knoten.

  Zusammenhang: von jedem Knoten v kommen wir zur Wurzel. Denn:

  Knoten v ist nach Konstruktion irgendwo Kindknoten (also oben).

  Sein Elternknoten (unten) wird nach Konstruktion irgendwo rechts wiederum als Kind vorkommen.

  ... und so weiter.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

  Schritt 2: Diese Matrix ist nun die erweiterte Prüfer-Kodierung eines Baumes mit n Knoten.

  Zusammenhang: von jedem Knoten v kommen wir zur Wurzel. Denn:

  Knoten v ist nach Konstruktion irgendwo Kindknoten (also oben).

  Sein Elternknoten (unten) wird nach Konstruktion irgendwo rechts wiederum als Kind vorkommen.

  ... und so weiter.

  Spätestens ganz rechts, unten (Elternknoten) stoßen wir auf die Wurzel.

Optimalität der Prüfer-Kodierung

* **Satz 10** (Prüfers Optimalitätssatz):
  Jede Folge von Zahlen zwischen 0 und $n-1$ der Länge $n-2$ ist eine Prüfer-Kodierung eines Baumes mit n Knoten.
* Beweis: Der Beweis erfolgt in zwei Schritten.

  Schritt 1:

  Erweitere die Folge zunächst um die (abgeschnittene) 0 am Ende.

  Diese erweitere Folge nehmen wir als zweite Zeile der Prüfer-Matrix.

  Rekonstruiere dann die fehlende erste Zeile (analog Baumrekonstruktionslemma):

  Schreibe von links nach rechts diejenige Zahl in die erste Zeile, die

  ... nicht schon vorher in der ersten Zeile vorkam,

  ... und auch nicht in der zweiten Zeile an gleicher Stelle oder danach vorkommt.

  Eine solche Zahl gibt es (insbesondere beim ersten Mal!), da in der Voraussetzung des Satzes höchstens $n-1$ von n Zahlen („zwischen 0 und $n-1$“) vorkommen können.

  Schritt 2: Diese Matrix ist nun die erweiterte Prüfer-Kodierung eines Baumes mit n Knoten.

  Zusammenhang: von jedem Knoten v kommen wir zur Wurzel. Denn:

  Knoten v ist nach Konstruktion irgendwo Kindknoten (also oben).

  Sein Elternknoten (unten) wird nach Konstruktion irgendwo rechts wiederum als Kind vorkommen.

  ... und so weiter.

  Spätestens ganz rechts, unten (Elternknoten) stoßen wir auf die Wurzel.

  Baumeingenschaft folgt damit aus Eulers Formel, da wir $n-1$ Kanten und n Knoten haben.

Folgerungen aus Prüfers Optimalitätsatz

Folgerungen aus Prüfers Optimalitätsatz

* Beweis für den Satz von Cayley:

Folgerungen aus Prüfers Optimalitätsatz

* Beweis für den Satz von Cayley:
  Es gibt n^{n-2} Prüfer-Kodierungen. Also gibt es genau so viele markierte Bäume.

Folgerungen aus Prüfers Optimalitätsatz

* Beweis für den Satz von Cayley:
  Es gibt n^{n-2} Prüfer-Kodierungen. Also gibt es genau so viele markierte Bäume.
* Jeder markierte Baum auf n Knoten kann eineindeutig einer Zahl zwischen 0 und $n^{n-2}-1$ zugeordent werden.

Folgerungen aus Prüfers Optimalitätsatz

* Beweis für den Satz von Cayley:
  Es gibt n^{n-2} Prüfer-Kodierungen. Also gibt es genau so viele markierte Bäume.
* Jeder markierte Baum auf n Knoten kann eineindeutig einer Zahl zwischen 0 und $n^{n-2}-1$ zugeordent werden.
  Damit kann ein Baum mit $\lceil (n-2)\log_2 n \rceil$ Bits kodiert werden.

Folgerungen aus Prüfers Optimalitätsatz

* Beweis für den Satz von Cayley:
  Es gibt n^{n-2} Prüfer-Kodierungen. Also gibt es genau so viele markierte Bäume.
* Jeder markierte Baum auf n Knoten kann eineindeutig einer Zahl zwischen 0 und $n^{n-2}-1$ zugeordent werden.
  Damit kann ein Baum mit $\lceil (n-2)\log_2 n \rceil$ Bits kodiert werden.
* Einfaches Verfahren, um einen zufälligen Baum zu generieren, so dass alle Bäume die gleiche Wahrscheinlichkeit haben:

Folgerungen aus Prüfers Optimalitätsatz

* Beweis für den Satz von Cayley:

  Es gibt n^{n-2} Prüfer-Kodierungen. Also gibt es genau so viele markierte Bäume.
* Jeder markierte Baum auf n Knoten kann eineindeutig einer Zahl zwischen 0 und $n^{n-2}-1$ zugeordent werden.

  Damit kann ein Baum mit $\lceil (n-2)\log_2 n \rceil$ Bits kodiert werden.
* Einfaches Verfahren, um einen zufälligen Baum zu generieren, so dass alle Bäume die gleiche Wahrscheinlichkeit haben:

  Generiere $n-2$ Zufallszahlen zwischen 0 und $n-1$.

Folgerungen aus Prüfers Optimalitätsatz

* Beweis für den Satz von Cayley:

  Es gibt n^{n-2} Prüfer-Kodierungen. Also gibt es genau so viele markierte Bäume.
* Jeder markierte Baum auf n Knoten kann eineindeutig einer Zahl zwischen 0 und $n^{n-2}-1$ zugeordent werden.

  Damit kann ein Baum mit $\lceil (n-2)\log_2 n \rceil$ Bits kodiert werden.
* Einfaches Verfahren, um einen zufälligen Baum zu generieren, so dass alle Bäume die gleiche Wahrscheinlichkeit haben:

  Generiere $n-2$ Zufallszahlen zwischen 0 und $n-1$.

  „Dekodiere“ diese Folge wie beschrieben als Baum.

Aufspannende Bäume

Aufspannende Bäume

* **Definition 11**:
  Sei $G = (V, E)$ ein Graph. Ein beliebiger Baum der Form (V, E') mit $E' \subseteq E$ wird **aufspannender Baum** (**Spannbaum**, **Gerüst**) des Graphen G genannt.

Aufspannende Bäume

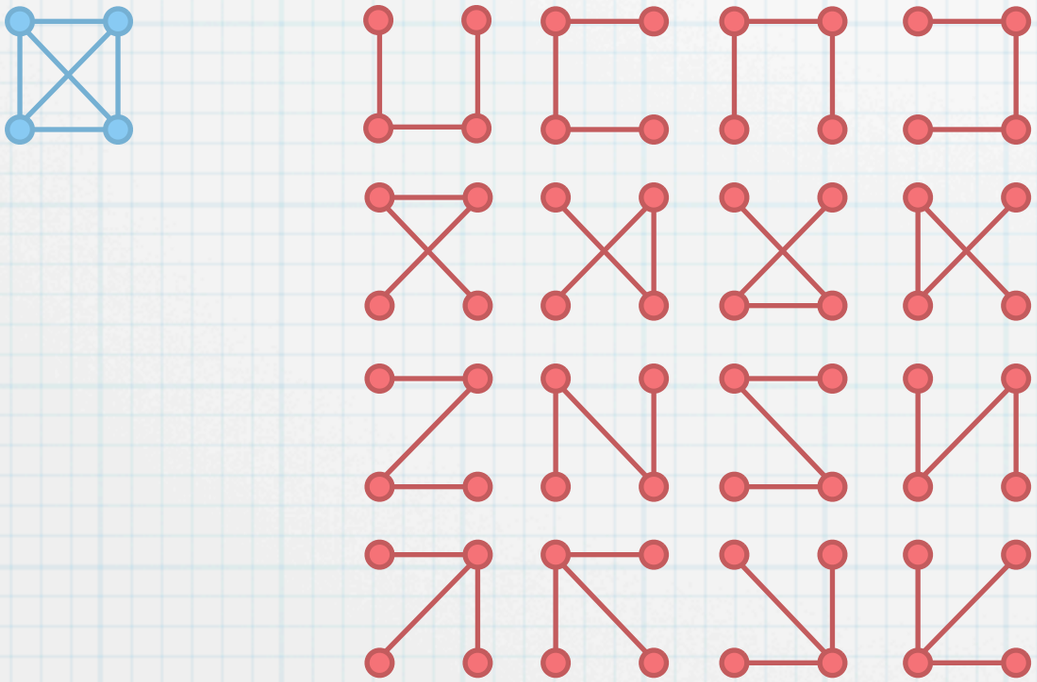
* **Definition 11**:
  Sei $G = (V, E)$ ein Graph. Ein beliebiger Baum der Form (V, E') mit $E' \subseteq E$ wird **aufspannender Baum** (**Spannbaum**, **Gerüst**) des Graphen G genannt.
* Ein aufspannender Baum ist ein Untergraph von G , der ein Baum ist und alle Knoten von G enthält.

Aufspannende Bäume

* **Definition 11**:
  Sei $G = (V, E)$ ein Graph. Ein beliebiger Baum der Form (V, E') mit $E' \subseteq E$ wird **aufspannender Baum** (**Spannbaum**, **Gerüst**) des Graphen G genannt.
* Ein aufspannender Baum ist ein Untergraph von G , der ein Baum ist und alle Knoten von G enthält.
* Bemerkung: Ein vollständiger Graph K_n hat n^{n-2} aufspannende Bäume. (Satz von Cayley)

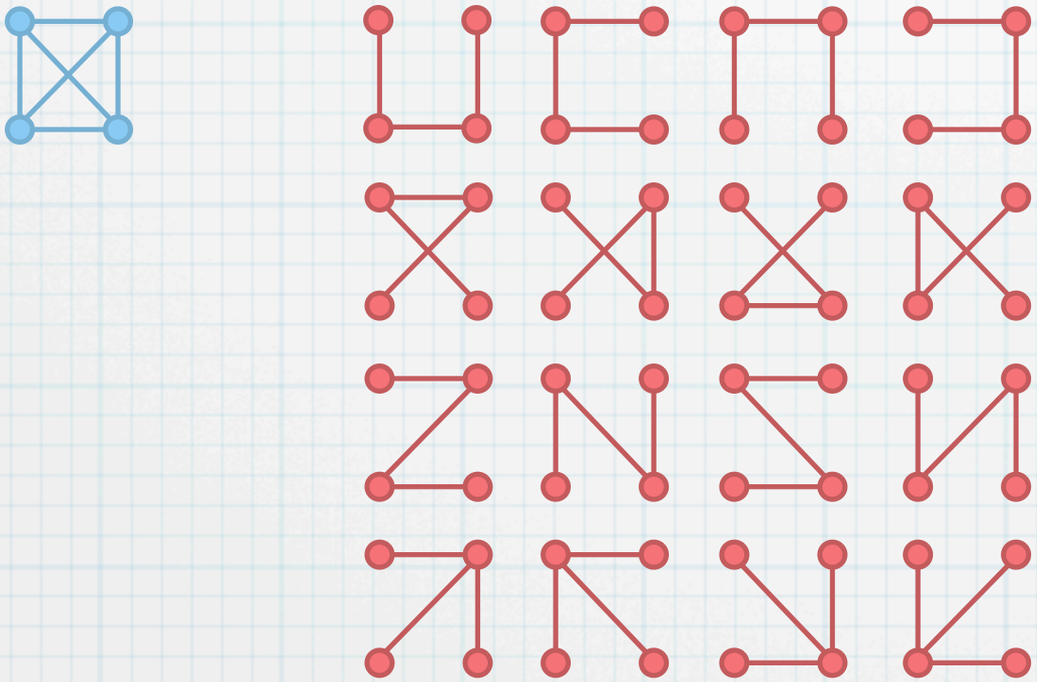
Aufspannende Bäume

* **Definition 11**:
  Sei $G = (V, E)$ ein Graph. Ein beliebiger Baum der Form (V, E') mit $E' \subseteq E$ wird **aufspannender Baum** (**Spannbaum**, **Gerüst**) des Graphen G genannt.
* Ein aufspannender Baum ist ein Untergraph von G , der ein Baum ist und alle Knoten von G enthält.
* Bemerkung: Ein vollständiger Graph K_n hat n^{n-2} aufspannende Bäume. (Satz von Cayley)
* Beispiel: K_4 und seine $4^{4-2} = 16$ unterschiedlichen aufspannenden Bäume.



Aufspannende Bäume

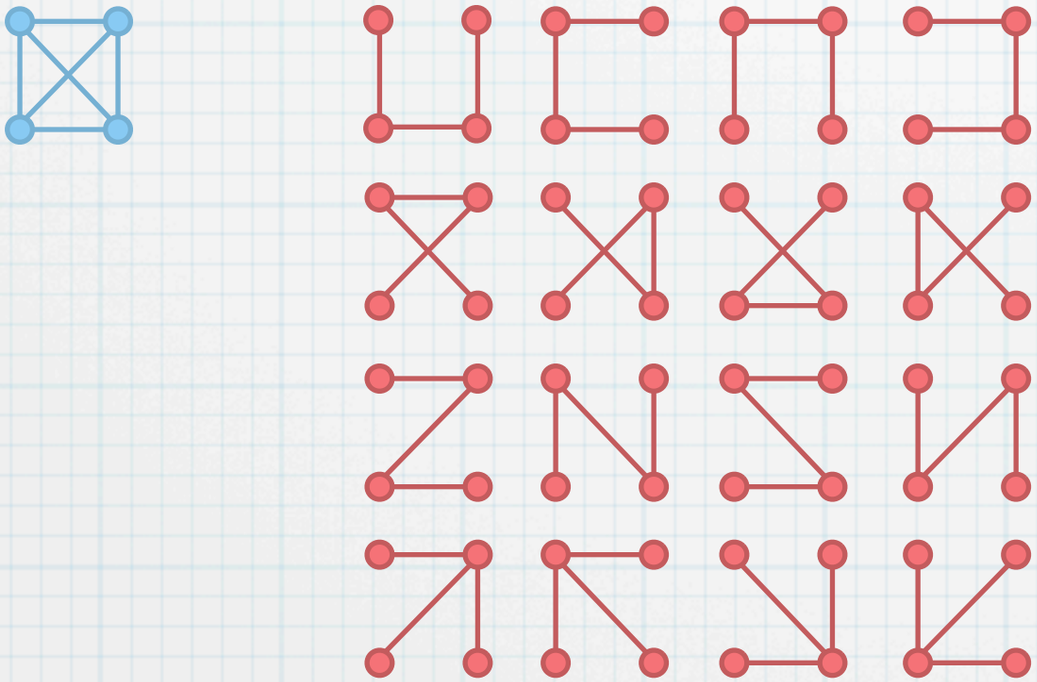
* **Definition 11**:
  Sei $G = (V, E)$ ein Graph. Ein beliebiger Baum der Form (V, E') mit $E' \subseteq E$ wird **aufspannender Baum** (**Spannbaum**, **Gerüst**) des Graphen G genannt.
* Ein aufspannender Baum ist ein Untergraph von G , der ein Baum ist und alle Knoten von G enthält.
* Bemerkung: Ein vollständiger Graph K_n hat n^{n-2} aufspannende Bäume. (Satz von Cayley)
* Beispiel: K_4 und seine $4^{4-2} = 16$ unterschiedlichen aufspannenden Bäume.



* Naheliegende Fragen:

Aufspannende Bäume

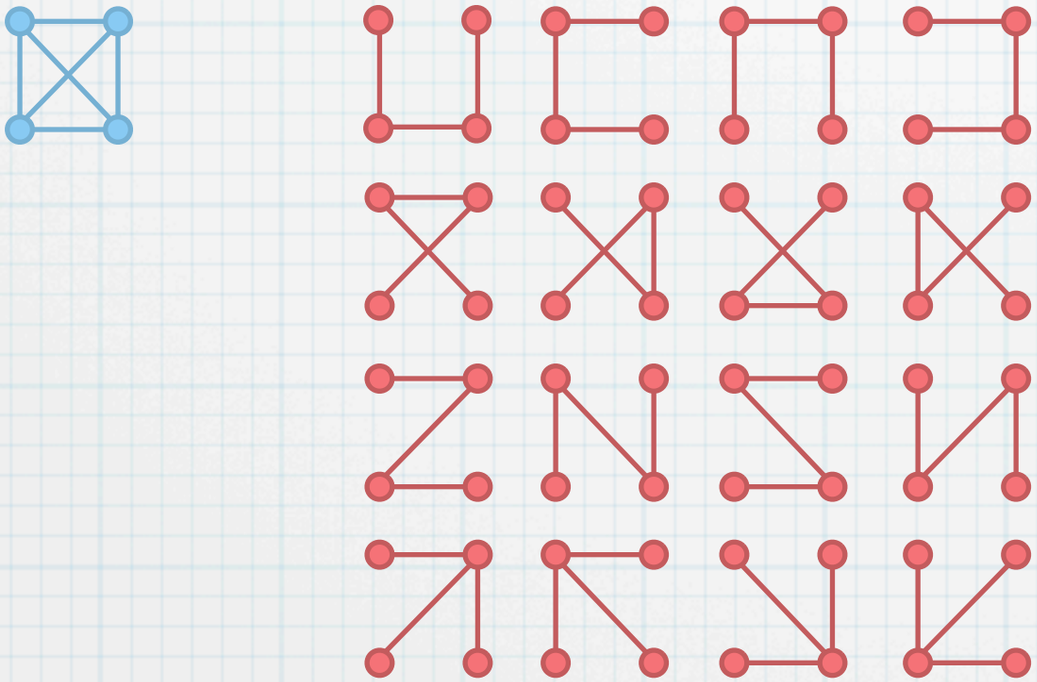
- **Definition 11**:
  Sei $G = (V, E)$ ein Graph. Ein beliebiger Baum der Form (V, E') mit $E' \subseteq E$ wird **aufspannender Baum** (**Spannbaum**, **Gerüst**) des Graphen G genannt.
- Ein aufspannender Baum ist ein Untergraph von G , der ein Baum ist und alle Knoten von G enthält.
- Bemerkung: Ein vollständiger Graph K_n hat n^{n-2} aufspannende Bäume. (Satz von Cayley)
- Beispiel: K_4 und seine $4^{4-2} = 16$ unterschiedlichen aufspannenden Bäume.



- Naheliegende Fragen:
  - Hat jeder Graph einen aufspannenden Baum?

Aufspannende Bäume

- **Definition 11**:
  Sei $G = (V, E)$ ein Graph. Ein beliebiger Baum der Form (V, E') mit $E' \subseteq E$ wird **aufspannender Baum** (**Spannbaum**, **Gerüst**) des Graphen G genannt.
- Ein aufspannender Baum ist ein Untergraph von G , der ein Baum ist und alle Knoten von G enthält.
- Bemerkung: Ein vollständiger Graph K_n hat n^{n-2} aufspannende Bäume. (Satz von Cayley)
- Beispiel: K_4 und seine $4^{4-2} = 16$ unterschiedlichen aufspannenden Bäume.



- Naheliegende Fragen:
  - Hat jeder Graph einen aufspannenden Baum?
  - Wie findet man einen solchen Baum, so es ihn gibt?

Welche Graphen haben Spannbäume?

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:
  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.
  1. Fall, $|V| = |E| + 1$. Dann ist der gesuchte Spannbaum G selbst (vgl. Satz 2).

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.
  1. Fall, $|V| = |E| + 1$. Dann ist der gesuchte Spannbaum G selbst (vgl. Satz 2).
  2. Fall, $|V| > |E| + 1$. Dann kann G nicht zusammenhängend sein, Widerspruch.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.
  1. Fall, $|V| = |E| + 1$. Dann ist der gesuchte Spannbaum G selbst (vgl. Satz 2).
  2. Fall, $|V| > |E| + 1$. Dann kann G nicht zusammenhängend sein, Widerspruch.
  3. Fall, $|V| < |E| + 1$. Dann ist G kein Baum (nach Satz 2), und muss Kreis(e) enthalten.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.
  1. Fall, $|V| = |E| + 1$. Dann ist der gesuchte Spannbaum G selbst (vgl. Satz 2).
  2. Fall, $|V| > |E| + 1$. Dann kann G nicht zusammenhängend sein, Widerspruch.
  3. Fall, $|V| < |E| + 1$. Dann ist G kein Baum (nach Satz 2), und muss Kreis(e) enthalten.
     Entferne eine beliebige Kante e des Kreises.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.
  1. Fall, $|V| = |E| + 1$. Dann ist der gesuchte Spannbaum G selbst (vgl. Satz 2).
  2. Fall, $|V| > |E| + 1$. Dann kann G nicht zusammenhängend sein, Widerspruch.
  3. Fall, $|V| < |E| + 1$. Dann ist G kein Baum (nach Satz 2), und muss Kreis(e) enthalten.
     Entferne eine beliebige Kante e des Kreises.
     Der übriggebliebene Graph $G' = (V, E')$ mit $E' := E \backslash \{e\}$ ist auch zusammenhängend.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.
  1. Fall, $|V| = |E| + 1$. Dann ist der gesuchte Spannbaum G selbst (vgl. Satz 2).
  2. Fall, $|V| > |E| + 1$. Dann kann G nicht zusammenhängend sein, Widerspruch.
  3. Fall, $|V| < |E| + 1$. Dann ist G kein Baum (nach Satz 2), und muss Kreis(e) enthalten.
     Entferne eine beliebige Kante e des Kreises.
     Der übriggebliebene Graph $G' = (V, E')$ mit $E' := E \backslash \{e\}$ ist auch zusammenhängend.
     Entweder gilt nun $|V| = |E'| + 1$ und G' ist der gesuchte Spannbaum.

Welche Graphen haben Spannbäume?

* **Satz 12**:
  Ein Graph G enthält genau dann einen Spannbaum, wenn er zusammenhängend ist.
* Beweis:

  Enthält G einen Spannbaum T, sind je zwei Knoten durch einen Weg in T verbunden.
  Da alle Kanten von T auch in G existieren, ist dieser Weg auch in G.
  Also ist G zusammenhängend.
  Umgekehrt sei $G = (V, E)$ nun zusammenhängend. Wir unterscheiden drei Fälle.
  1. Fall, $|V| = |E| + 1$. Dann ist der gesuchte Spannbaum G selbst (vgl. Satz 2).
  2. Fall, $|V| > |E| + 1$. Dann kann G nicht zusammenhängend sein, Widerspruch.
  3. Fall, $|V| < |E| + 1$. Dann ist G kein Baum (nach Satz 2), und muss Kreis(e) enthalten.
     Entferne eine beliebige Kante e des Kreises.
     Der übriggebliebene Graph $G' = (V, E')$ mit $E' := E \backslash \{e\}$ ist auch zusammenhängend.
     Entweder gilt nun $|V| = |E'| + 1$ und G' ist der gesuchte Spannbaum.
     Oder wir wiederholen diese Reduktion (möglich, da G' weiterhin zusammenhängend).

Minimale aufspannende Bäume

Minimale aufspannende Bäume

* **Definition 13**:
  Ein **bewerteter Graph** (auch: gewichteter Graph) ist ein Graph $G = (V, E)$, in dem jeder Kante e eine reelle Zahl $w(e)$ zugeordnet wird, die als **Bewertung**, **Gewicht** oder **Länge** von e bezeichnet wird. Wir schreiben auch kurz: $G = (V, E, w)$.

Minimale aufspannende Bäume

* **Definition 13**:
  Ein **bewerteter Graph** (auch: gewichteter Graph) ist ein Graph $G = (V, E)$, in dem jeder Kante e eine reelle Zahl $w(e)$ zugeordnet wird, die als **Bewertung**, **Gewicht** oder **Länge** von e bezeichnet wird. Wir schreiben auch kurz: $G = (V, E, w)$.

  Ist H ein Untergraph eines bewerteten Graphen, so ist die **Bewertung** $w(H)$ die Summe der Bewertungen $w(e_1) + \ldots + w(e_k)$, wobei $\{e_1, \ldots, e_k\}$ die Menge der Kanten von H ist.

Minimale aufspannende Bäume

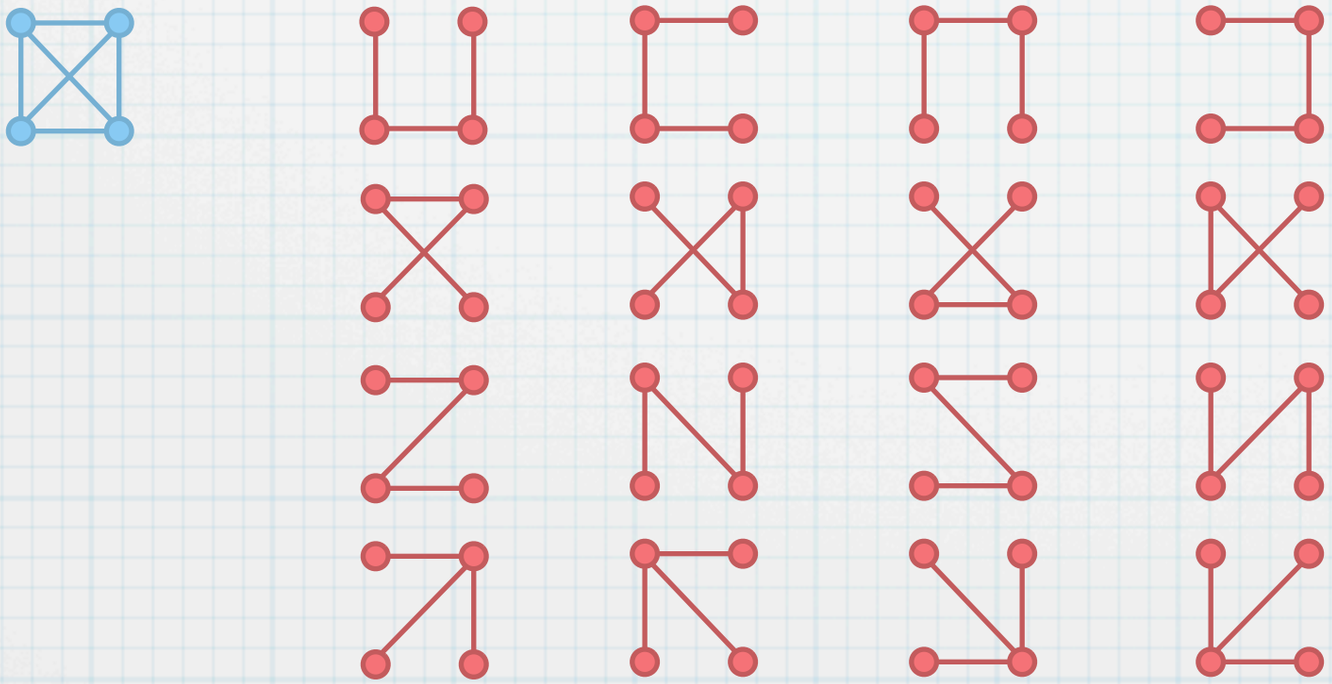
* **Definition 13**:
  Ein **bewerteter Graph** (auch: gewichteter Graph) ist ein Graph $G = (V, E)$, in dem jeder Kante e eine reelle Zahl $w(e)$ zugeordnet wird, die als **Bewertung**, **Gewicht** oder **Länge** von e bezeichnet wird. Wir schreiben auch kurz: $G = (V, E, w)$.

  Ist H ein Untergraph eines bewerteten Graphen, so ist die **Bewertung** $w(H)$ die Summe der Bewertungen $w(e_1) + \ldots + w(e_k)$, wobei $\{e_1, \ldots, e_k\}$ die Menge der Kanten von H ist.
* **Definition 14**:
  Sei $G = (V, E)$ ein bewerteter Graph mit Gewichten $w : E \to \mathbb{R}$. Die Aufgabe, einen Spannbaum von G mit minimalem (maximalem) Gewicht zu finden, oder zu entscheiden, dass G keinen Spannbaum hat, wird als **minimales (maximales) Spannbaumproblem** bezeichnet.

Minimale aufspannende Bäume

* **Definition 13**:
  Ein **bewerteter Graph** (auch: gewichteter Graph) ist ein Graph $G = (V, E)$, in dem jeder Kante e eine reelle Zahl $w(e)$ zugeordnet wird, die als **Bewertung**, **Gewicht** oder **Länge** von e bezeichnet wird. Wir schreiben auch kurz: $G = (V, E, w)$.

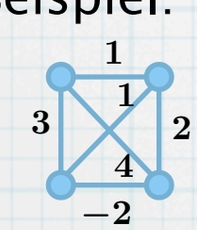
  Ist H ein Untergraph eines bewerteten Graphen, so ist die **Bewertung** $w(H)$ die Summe der Bewertungen $w(e_1) + \ldots + w(e_k)$, wobei $\{e_1, \ldots, e_k\}$ die Menge der Kanten von H ist.
* **Definition 14**:
  Sei $G = (V, E)$ ein bewerteter Graph mit Gewichten $w : E \to \mathbb{R}$. Die Aufgabe, einen Spannbaum von G mit minimalem (maximalem) Gewicht zu finden, oder zu entscheiden, dass G keinen Spannbaum hat, wird als **minimales (maximales) Spannbaumproblem** bezeichnet.
* Beispiel:

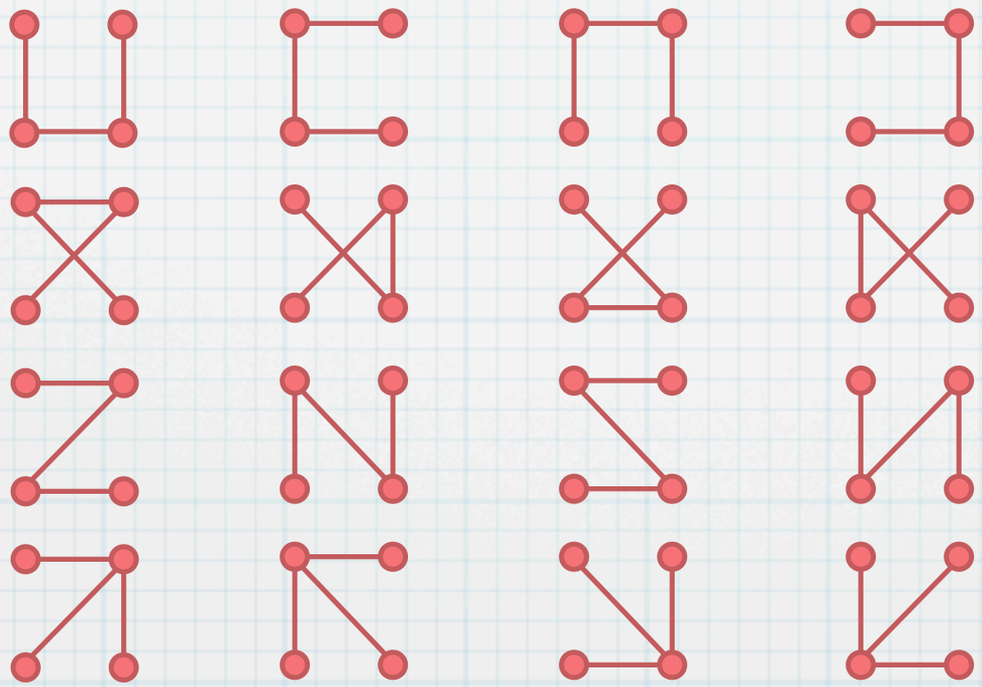


Minimale aufspannende Bäume

* **Definition 13**:
  Ein **bewerteter Graph** (auch: gewichteter Graph) ist ein Graph $G = (V, E)$, in dem jeder Kante e eine reelle Zahl $w(e)$ zugeordnet wird, die als **Bewertung**, **Gewicht** oder **Länge** von e bezeichnet wird. Wir schreiben auch kurz: $G = (V, E, w)$.

  Ist H ein Untergraph eines bewerteten Graphen, so ist die **Bewertung** $w(H)$ die Summe der Bewertungen $w(e_1) + \ldots + w(e_k)$, wobei $\{e_1, \ldots, e_k\}$ die Menge der Kanten von H ist.
* **Definition 14**:
  Sei $G = (V, E)$ ein bewerteter Graph mit Gewichten $w : E \to \mathbb{R}$. Die Aufgabe, einen Spannbaum von G mit minimalem (maximalem) Gewicht zu finden, oder zu entscheiden, dass G keinen Spannbaum hat, wird als **minimales (maximales) Spannbaumproblem** bezeichnet.
* Beispiel:

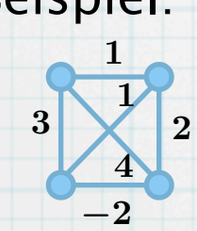


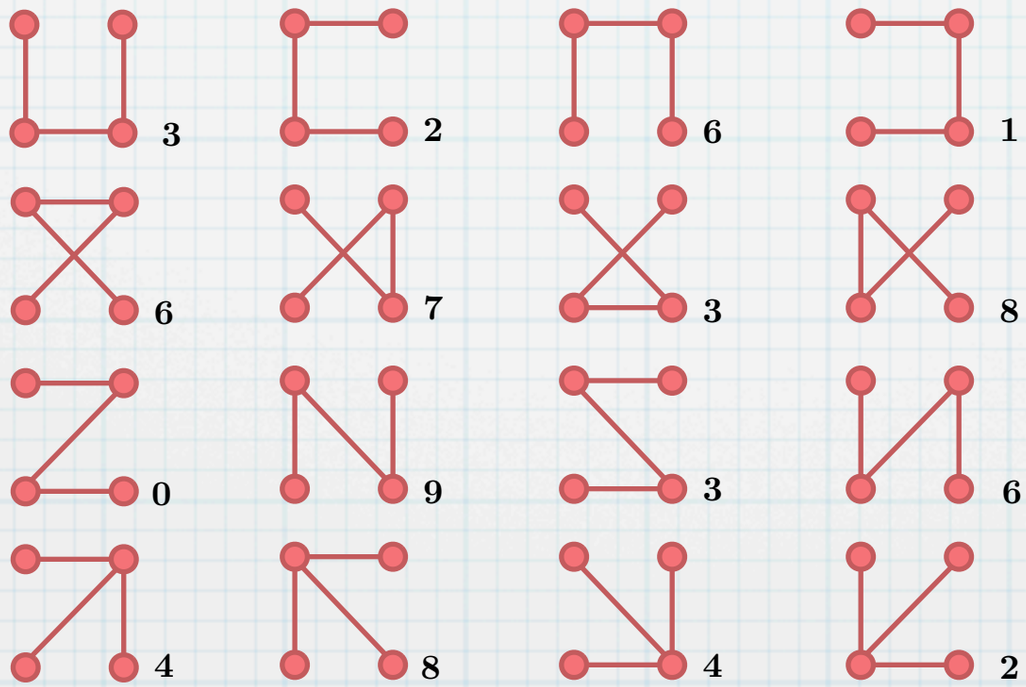


Minimale aufspannende Bäume

* **Definition 13**:
  Ein **bewerteter Graph** (auch: gewichteter Graph) ist ein Graph $G = (V, E)$, in dem jeder Kante e eine reelle Zahl $w(e)$ zugeordnet wird, die als **Bewertung**, **Gewicht** oder **Länge** von e bezeichnet wird. Wir schreiben auch kurz: $G = (V, E, w)$.

  Ist H ein Untergraph eines bewerteten Graphen, so ist die **Bewertung** $w(H)$ die Summe der Bewertungen $w(e_1) + \ldots + w(e_k)$, wobei $\{e_1, \ldots, e_k\}$ die Menge der Kanten von H ist.
* **Definition 14**:
  Sei $G = (V, E)$ ein bewerteter Graph mit Gewichten $w : E \to \mathbb{R}$. Die Aufgabe, einen Spannbaum von G mit minimalem (maximalem) Gewicht zu finden, oder zu entscheiden, dass G keinen Spannbaum hat, wird als **minimales (maximales) Spannbaumproblem** bezeichnet.
* Beispiel:





Wälder und maximale Wälder

Wälder und maximale Wälder

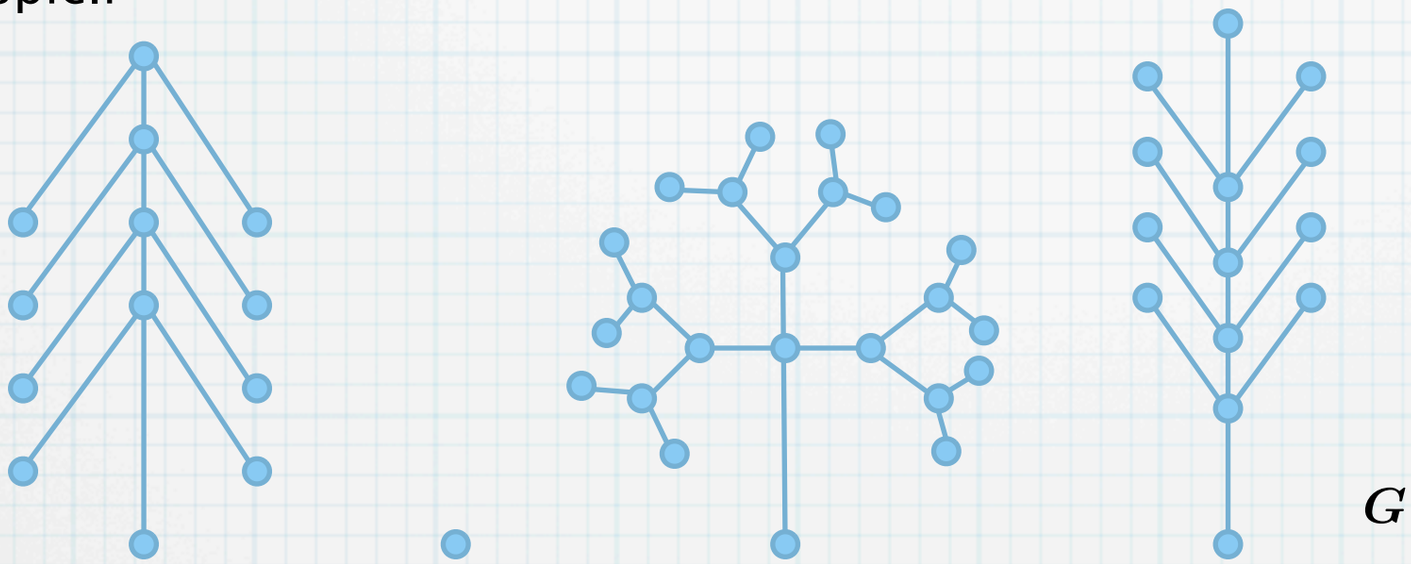
* **Definition 16**:
  Ein Wald ist ein kreisfreier Graph.

Wälder und maximale Wälder

* **Definition 16**:
  Ein Wald ist ein kreisfreier Graph.
* Bemerkung: Ein Baum kann somit als zusammenhängender Wald gesehen werden.

Wälder und maximale Wälder

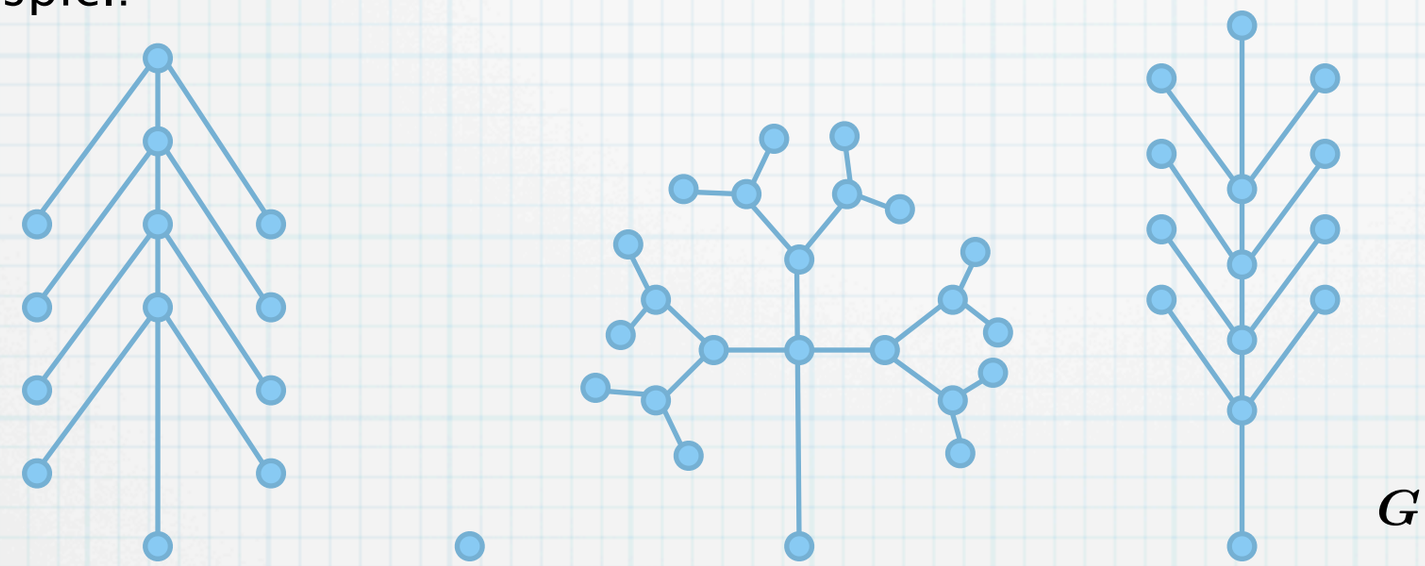
* **Definition 16**:
  Ein Wald ist ein kreisfreier Graph.
* Bemerkung: Ein Baum kann somit als zusammenhängender Wald gesehen werden.
* Beispiel:



G

Wälder und maximale Wälder

* **Definition 16**:
  Ein Wald ist ein kreisfreier Graph.
* Bemerkung: Ein Baum kann somit als zusammenhängender Wald gesehen werden.
* Beispiel:



G

* **Definition 15**:
  Sei G ein bewerteter Graph mit Gewichten w. Die Aufgabe, einen Wald in G mit maximalem Gewicht zu finden, wird als **maximales gewichtetes Waldproblem** bezeichnet.

Etwas Komplexitätstheorie

Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent**.

Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent.**

Problem:

Instanz:

(Algorithmus):

Lösung:

Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent.**

Problem:	$\mathcal{P}$	$\mathcal{Q}$
Instanz:		
(Algorithmus):		
Lösung:		

Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent.**

Problem:	$\mathcal{P}$	$\mathcal{Q}$
Instanz:	x	
(Algorithmus):		
Lösung:		

Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent.**

Problem:	$\mathcal{P}$		$\mathcal{Q}$
Instanz:	x	$\xrightarrow{f}$	$f(x)$

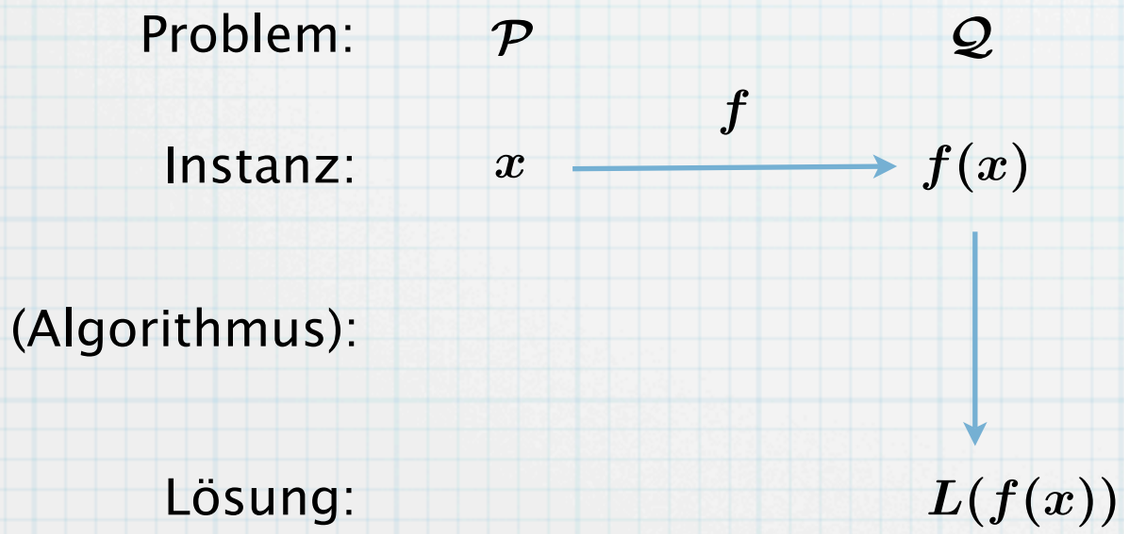
(Algorithmus):

Lösung:

Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

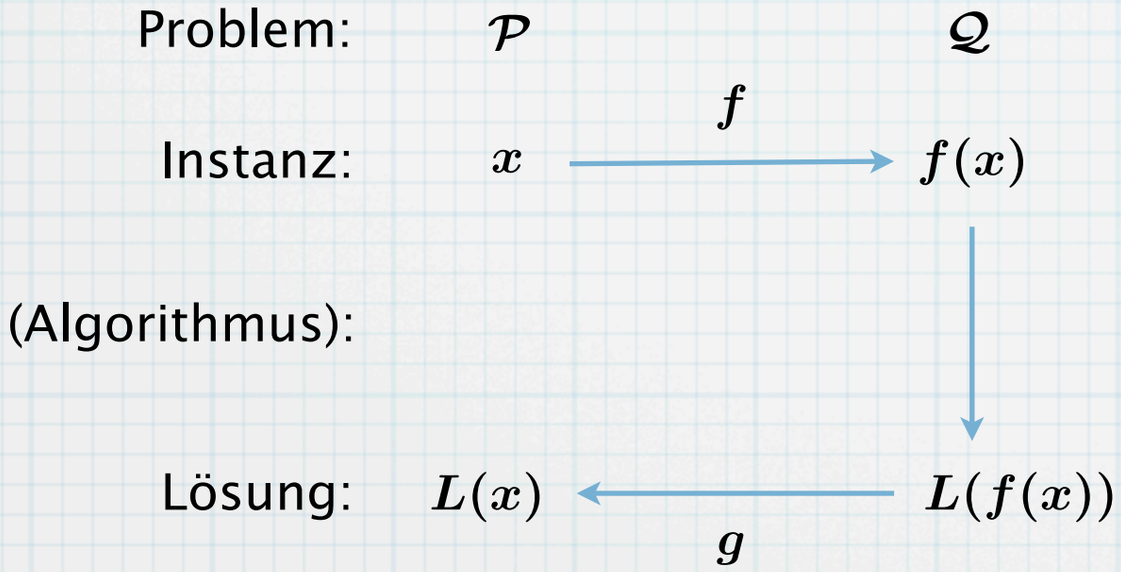
  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent.**



Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

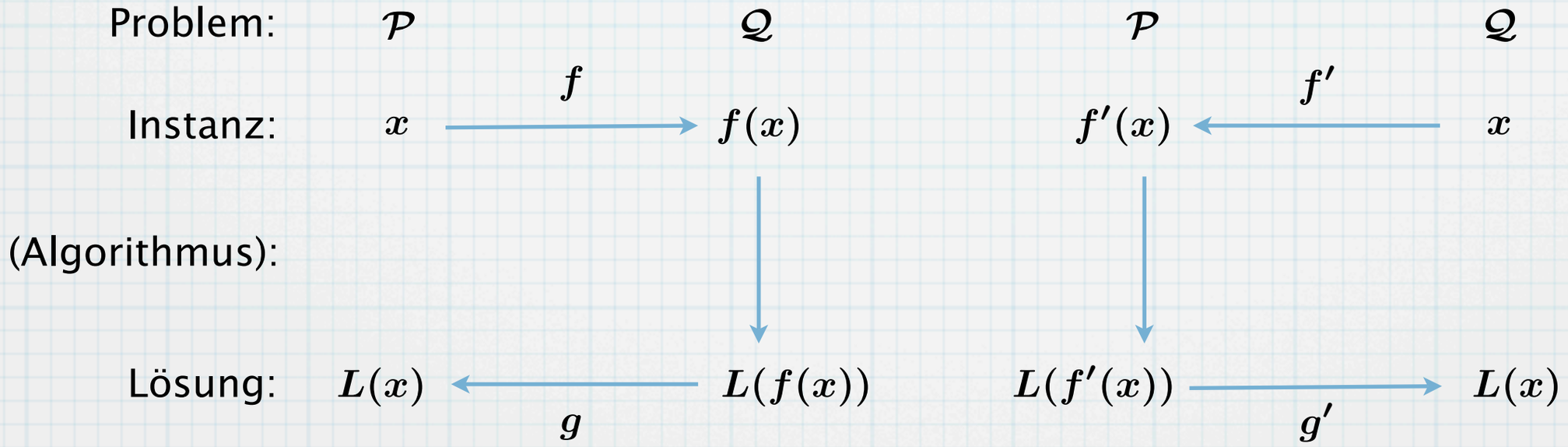
  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent.**



Etwas Komplexitätstheorie

* **Definition 17**:
  Seien $\mathcal{P}$ und $\mathcal{Q}$ Probleme. Problem $\mathcal{P}$ kann **linear** auf Problem $\mathcal{Q}$ **reduziert** werden, falls es Funktionen f und g gibt, die in linearer Zeit ausgerechnet werden können, so dass f aus einer Instanz x von $\mathcal{P}$ eine Instanz $f(x)$ von $\mathcal{Q}$ macht, und g aus einer Lösung von $f(x)$ eine Lösung von x macht.

  Wenn $\mathcal{P}$ linear auf $\mathcal{Q}$ und $\mathcal{Q}$ linear auf $\mathcal{P}$ reduziert werden kann, dann heißen die beiden Probleme **äquivalent.**



Minimale Spannbäume und maximale Wälder

Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.

Minimale Spannbäume und maximale Wälder

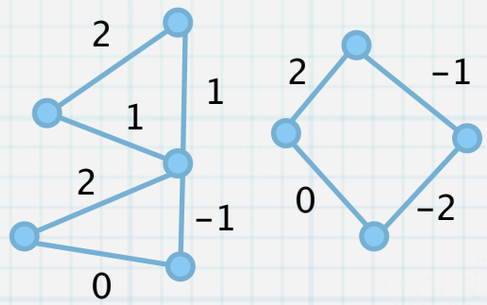
* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

(G, w)

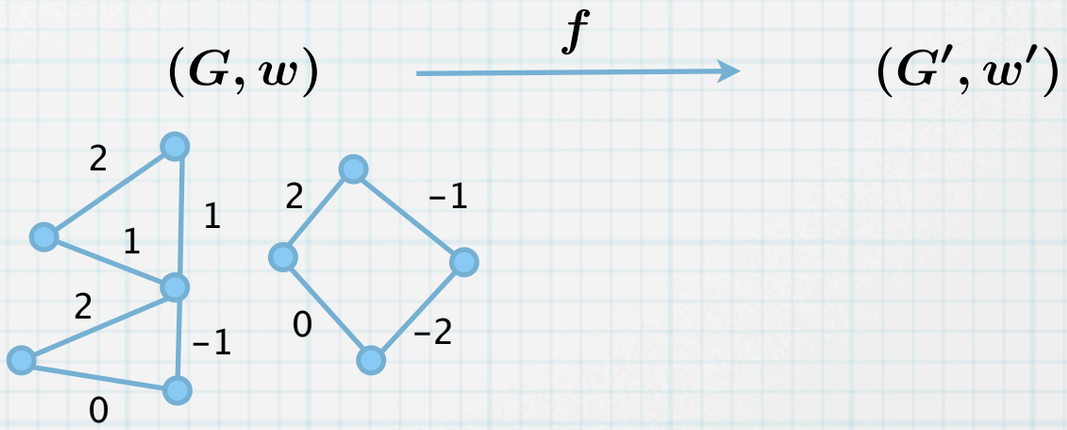


Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

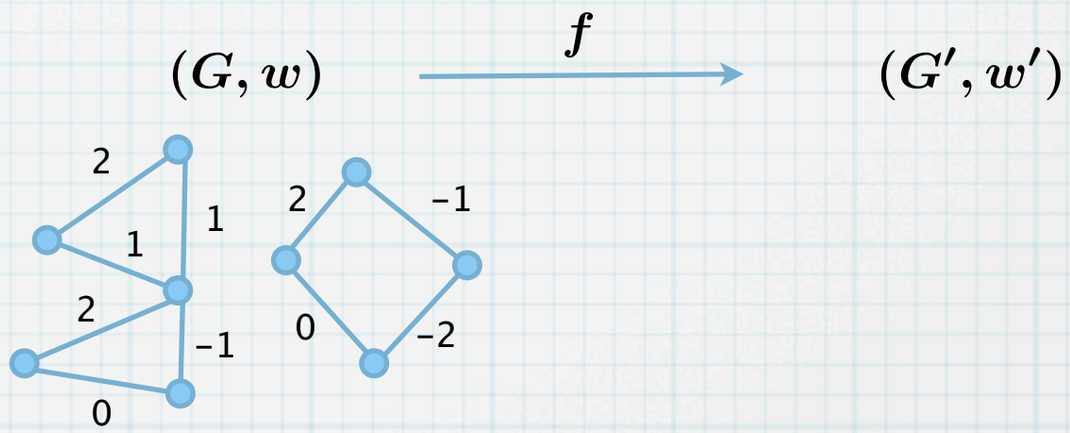
  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.



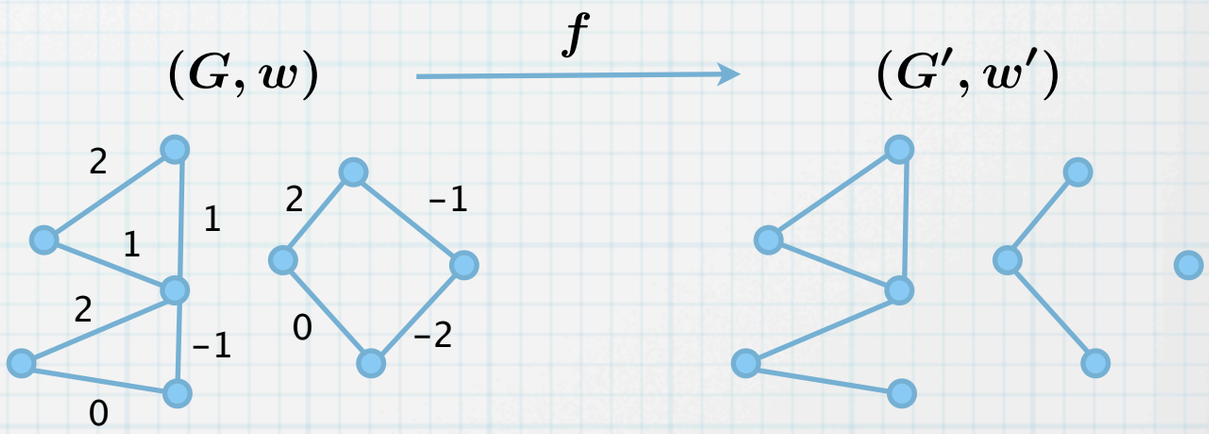
Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):
  Sei (G, w) eine Instanz des max. gew. Waldprob.
  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.
  ...entferne aus G alle Kanten e mit $w(e) < 0$,



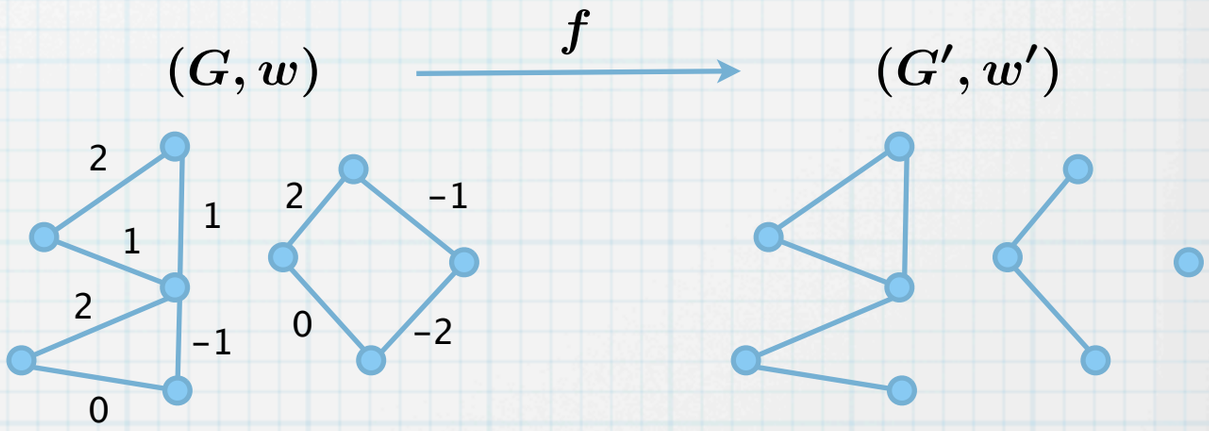
Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):
  Sei (G, w) eine Instanz des max. gew. Waldprob.
  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.
  ...entferne aus G alle Kanten e mit $w(e) < 0$,



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):
  Sei (G, w) eine Instanz des max. gew. Waldprob.
  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.
  ...entferne aus G alle Kanten e mit $w(e) < 0$,
  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,



Minimale Spannbäume und maximale Wälder

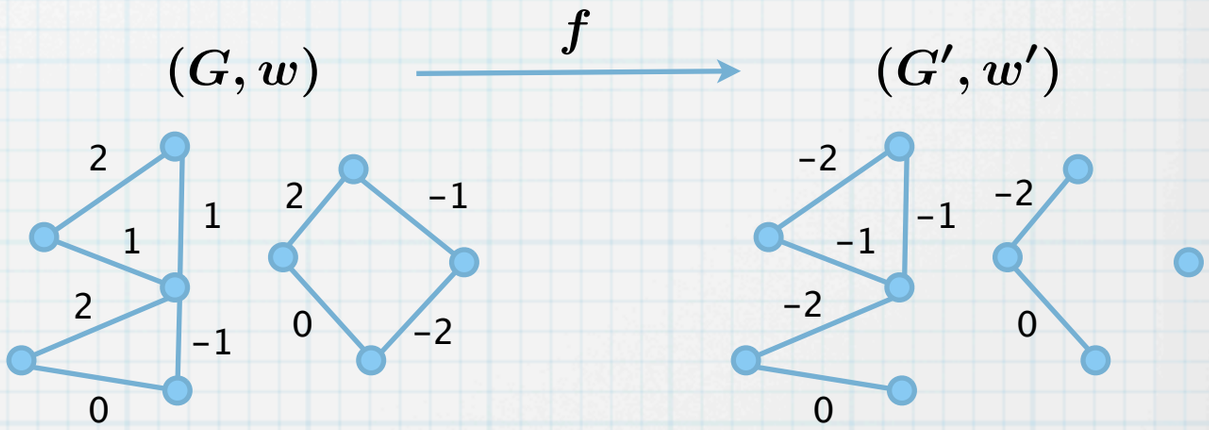
* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

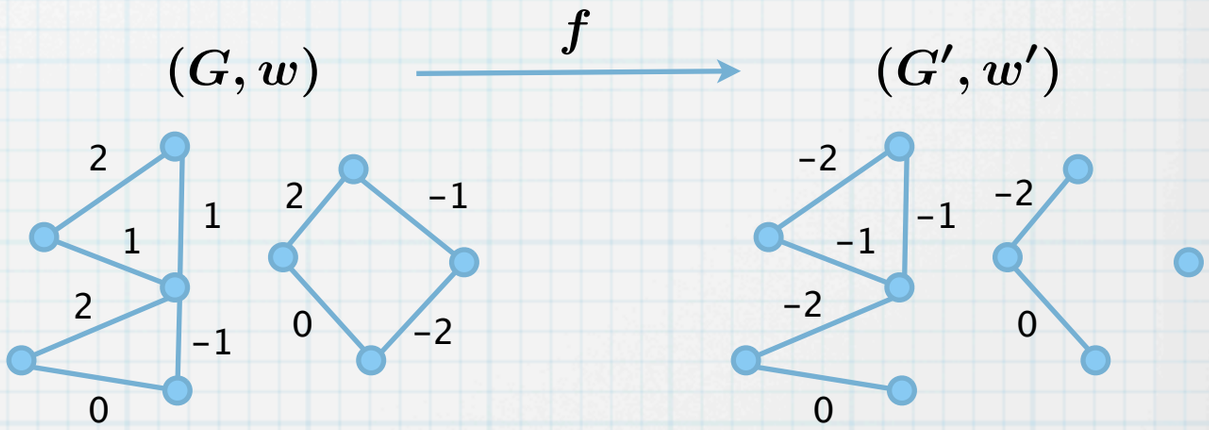
  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

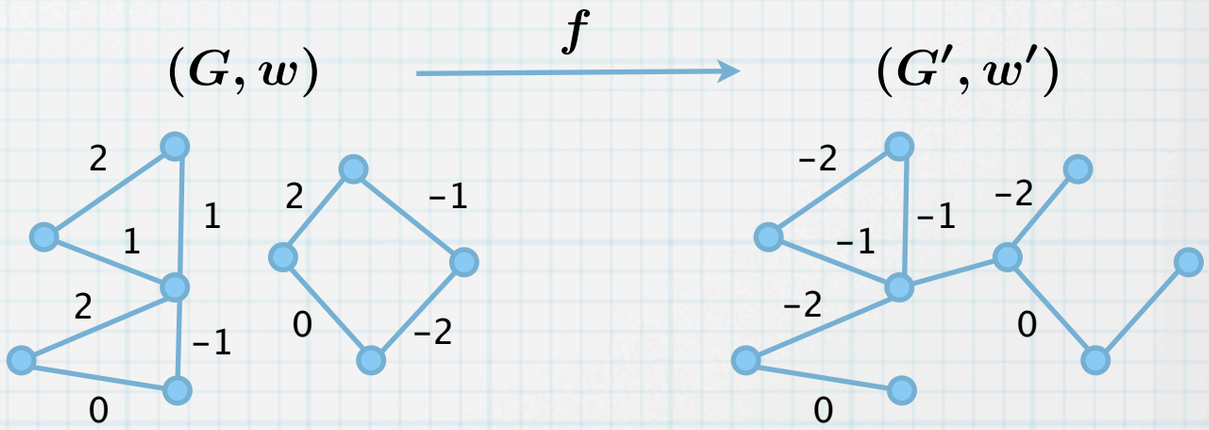
  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

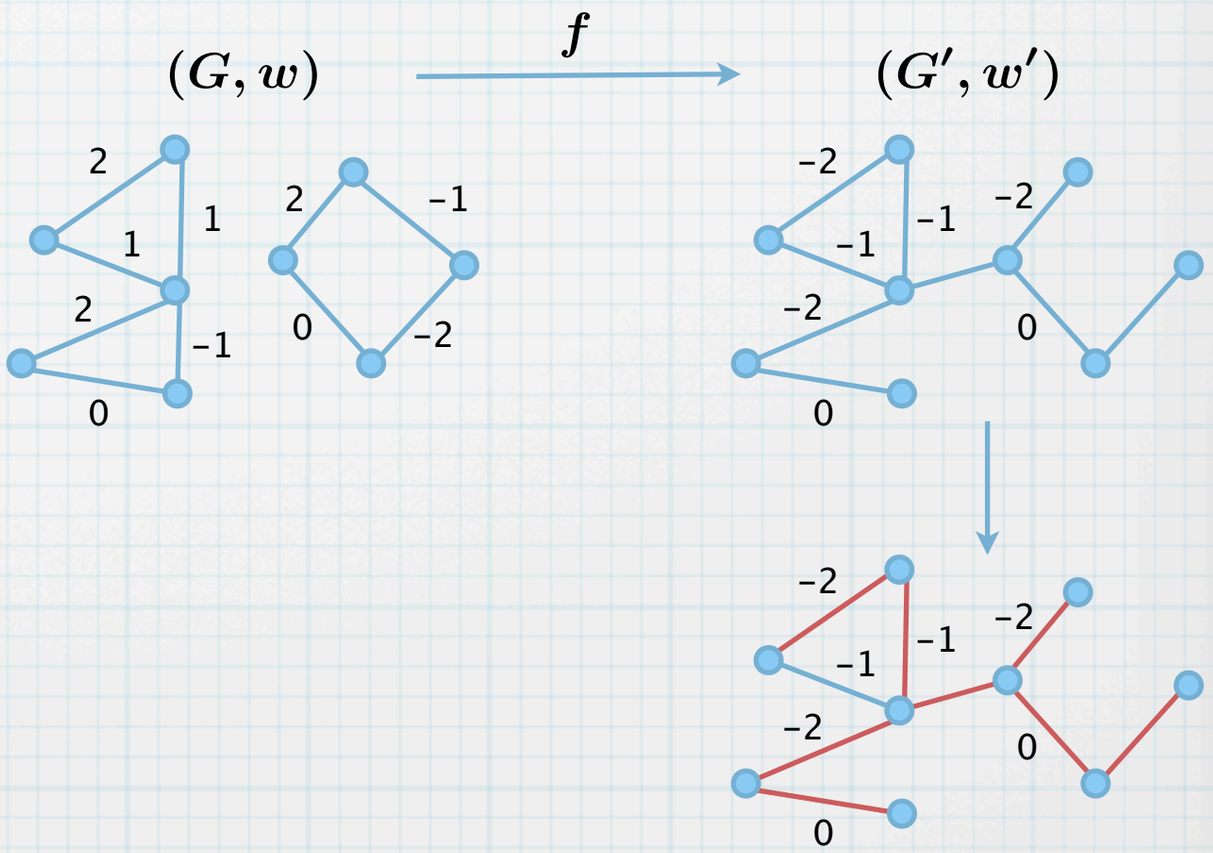
  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

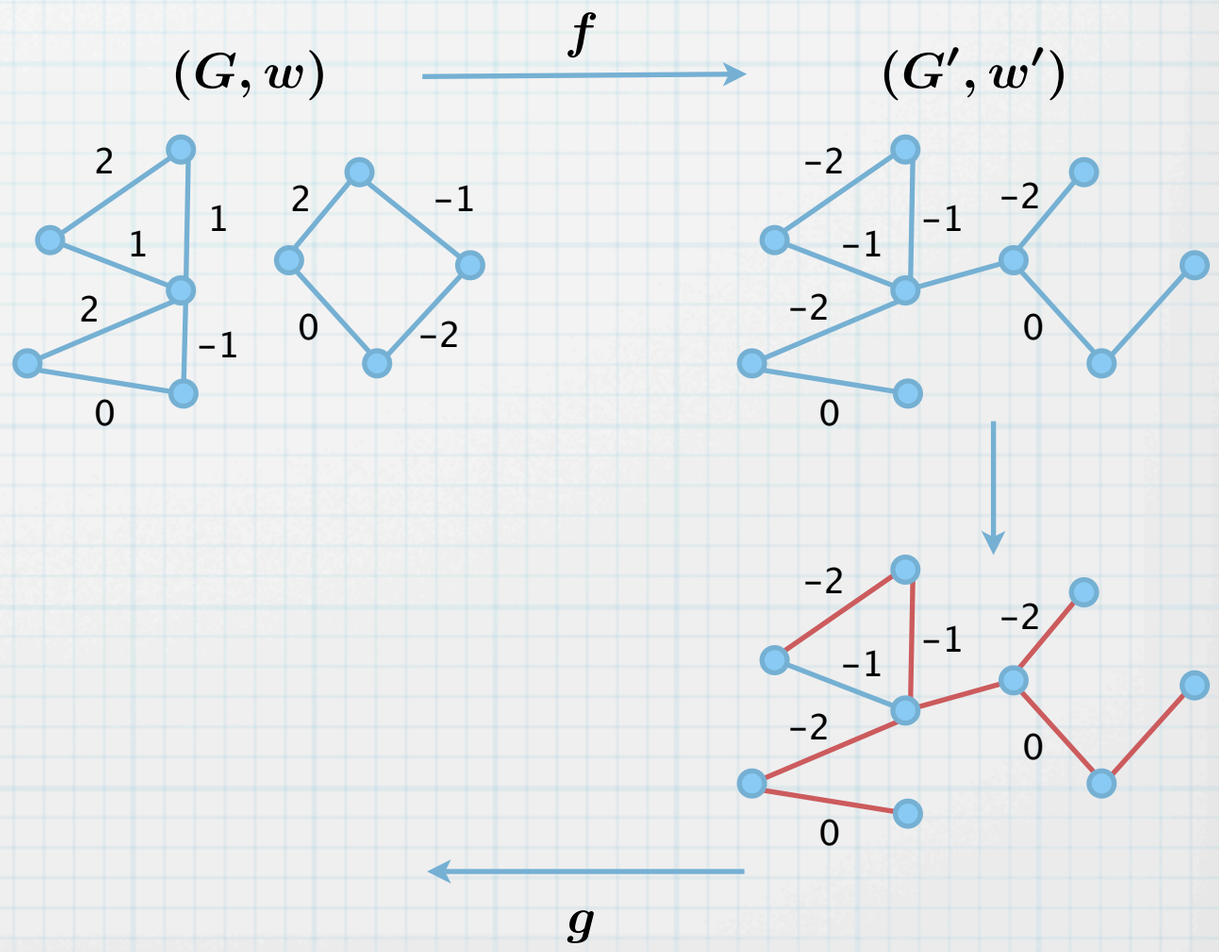
  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

  Funktion g erzeugt aus Lösung von T
  (G', w') eine Lösung F von (G, w)



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

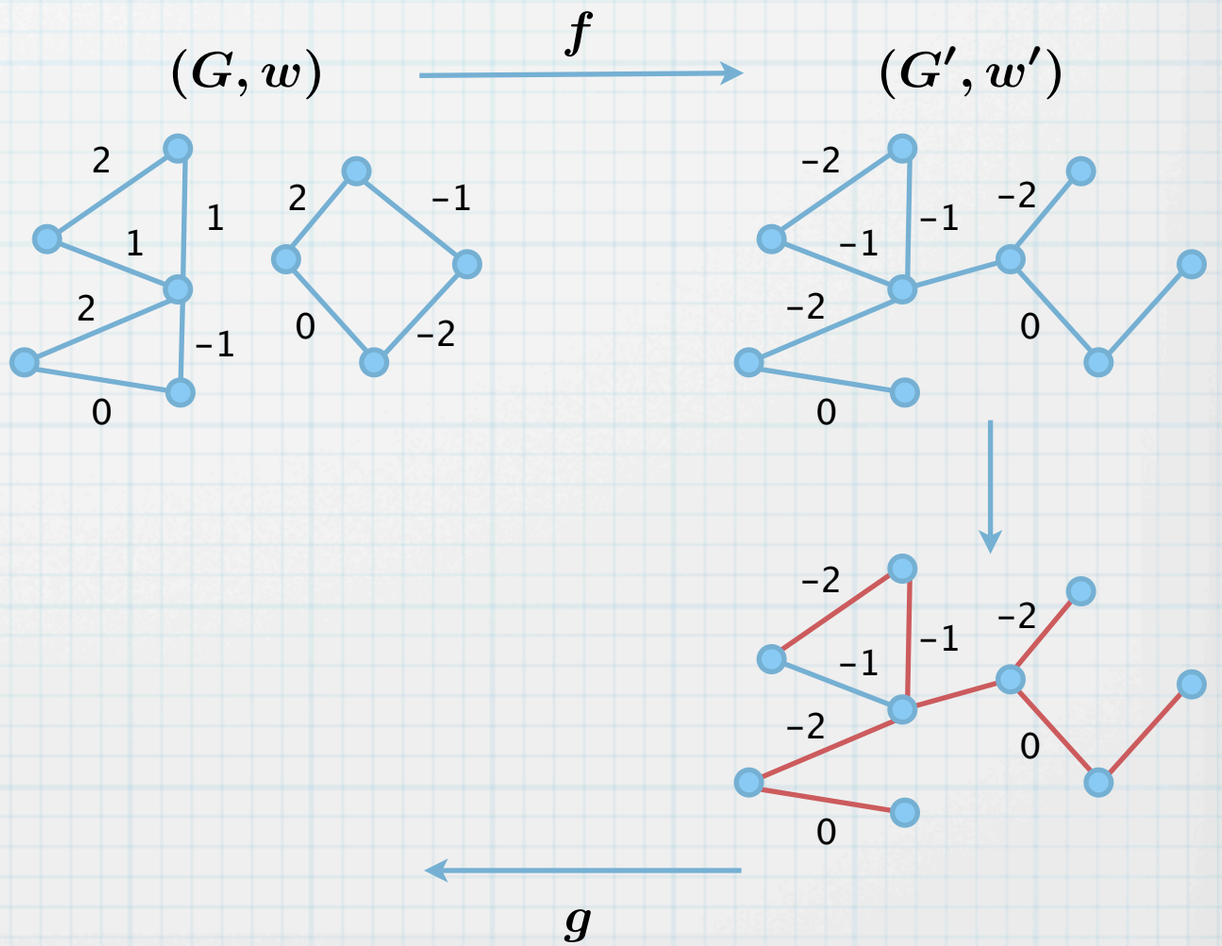
  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

  Funktion g erzeugt aus Lösung von T (G', w') eine Lösung F von (G, w)

  ...lasse alle Kanten aus K weg



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

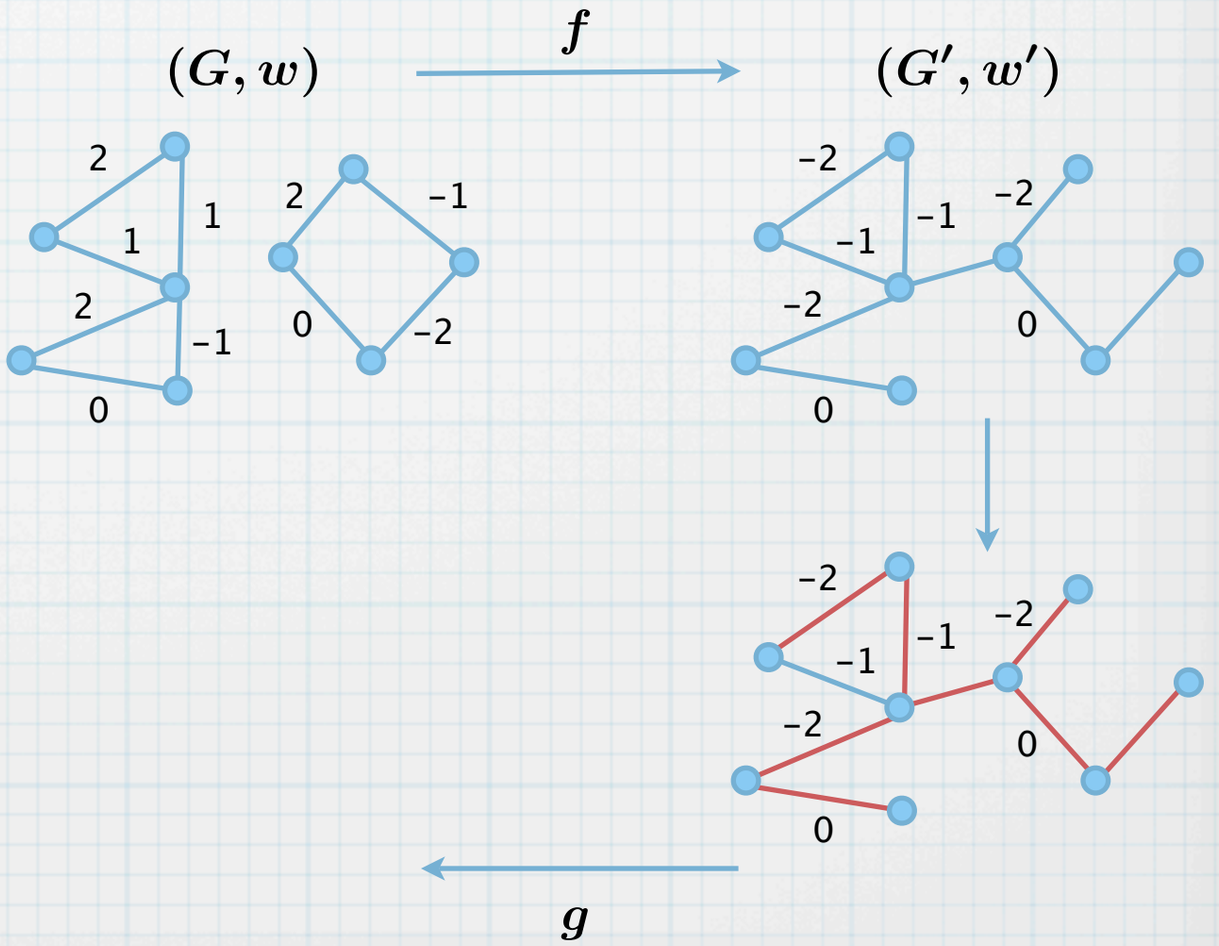
  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

  Funktion g erzeugt aus Lösung von T (G', w') eine Lösung F von (G, w)

  ...lasse alle Kanten aus K weg

  ...übernehme alle anderen 1:1



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

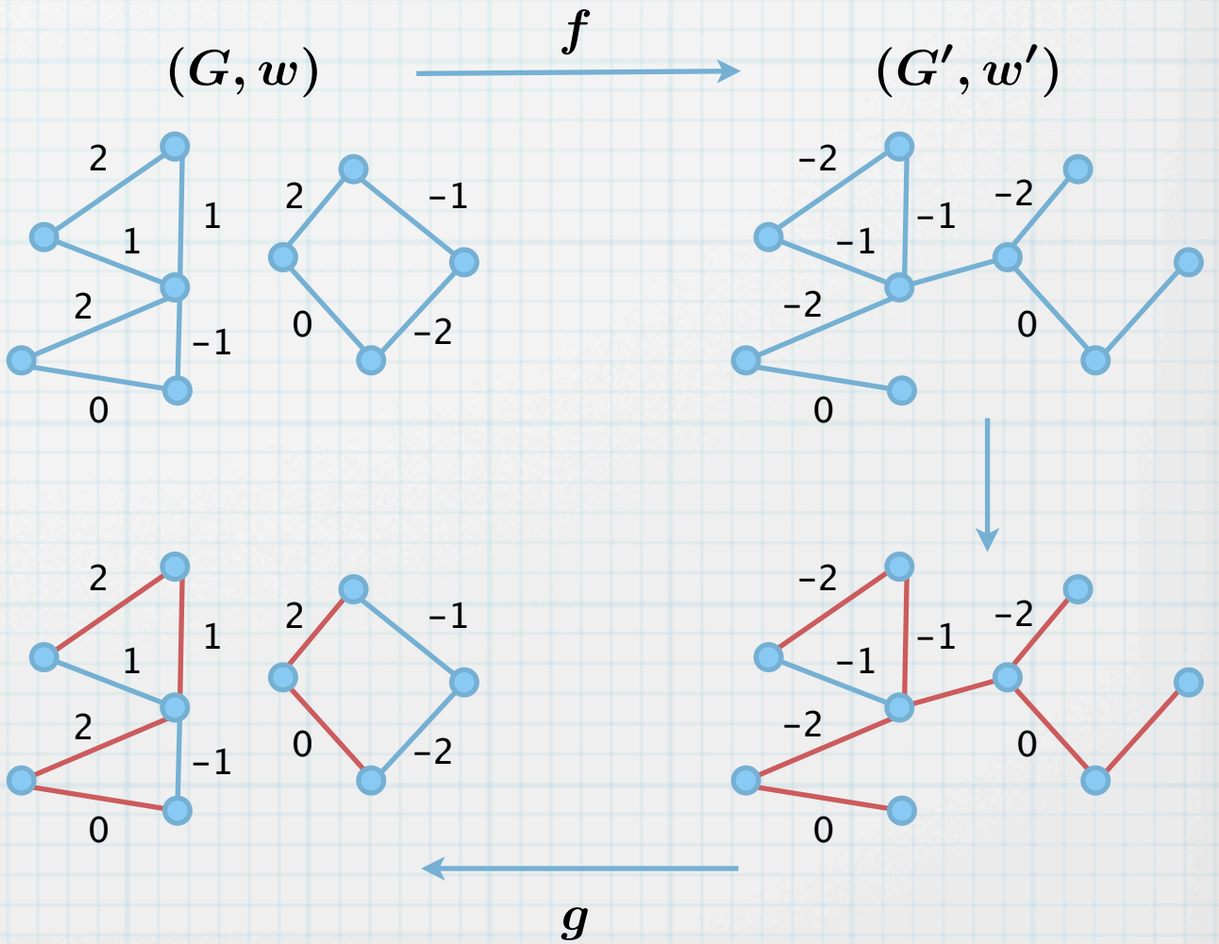
  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

  Funktion g erzeugt aus Lösung von T (G', w') eine Lösung F von (G, w)

  ...lasse alle Kanten aus K weg

  ...übernehme alle anderen 1:1



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

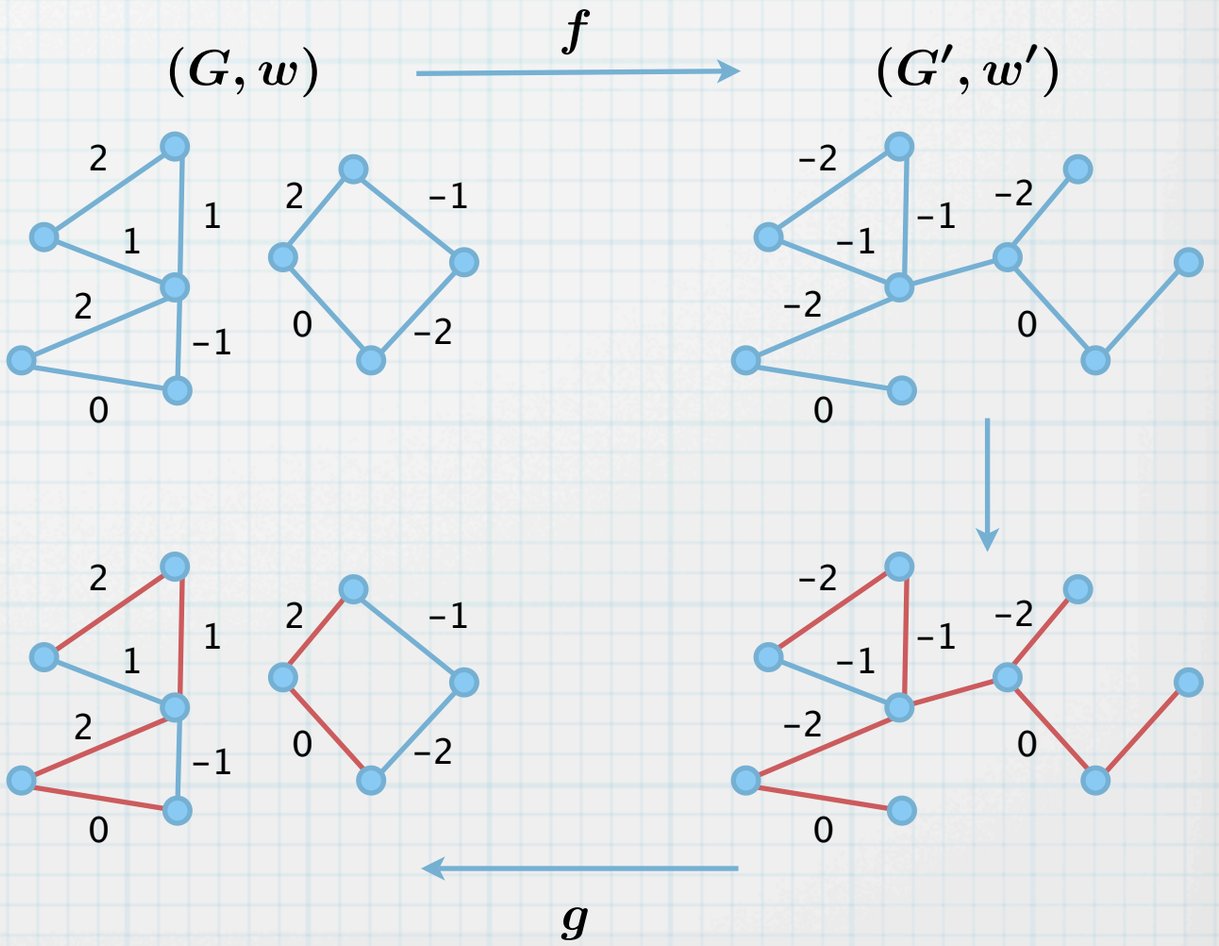
  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

  Funktion g erzeugt aus Lösung von T (G', w') eine Lösung F von (G, w)

  ...lasse alle Kanten aus K weg

  ...übernehme alle anderen 1:1

  F ist Wald, da Untergraph von Baum T.



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

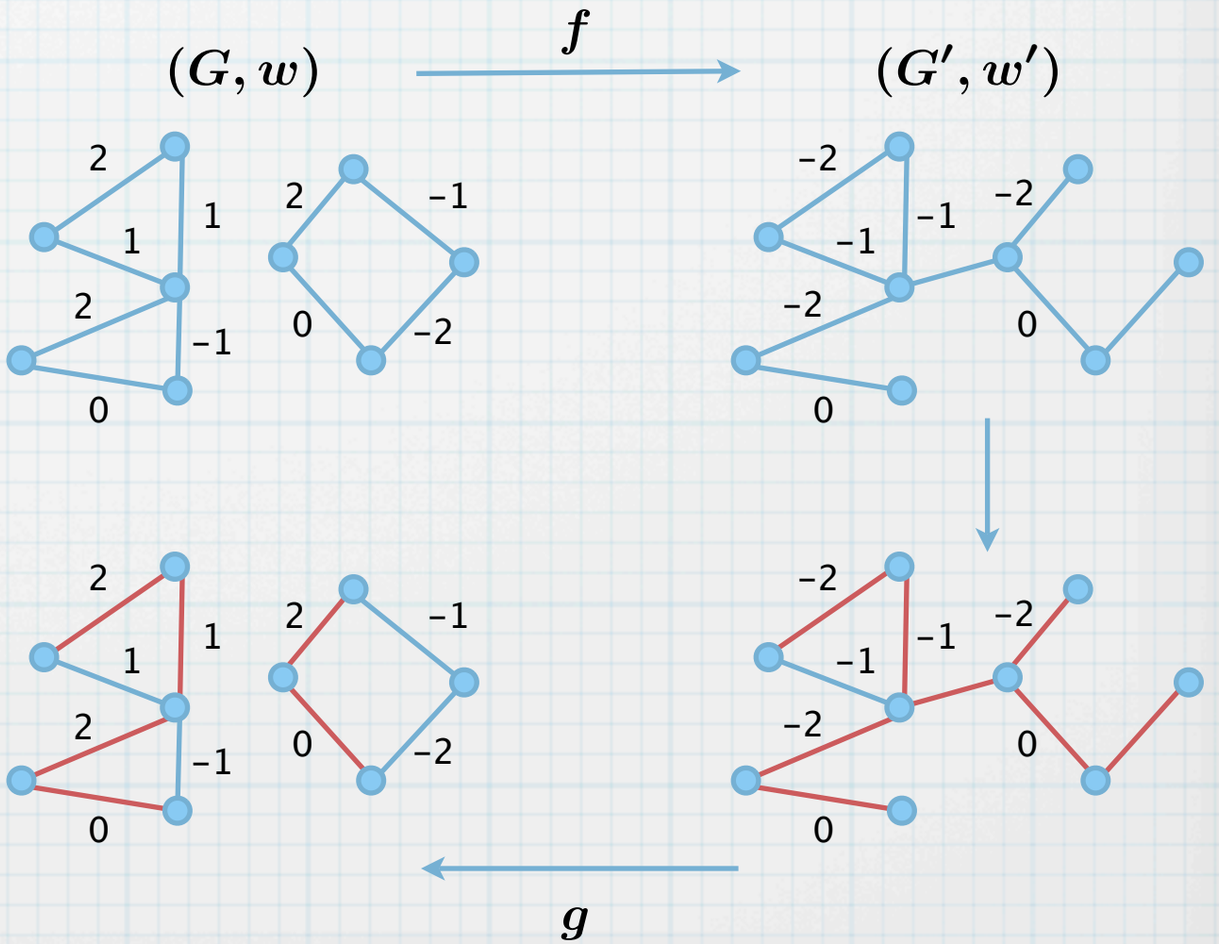
  Funktion g erzeugt aus Lösung von T (G', w') eine Lösung F von (G, w)

  ...lasse alle Kanten aus K weg

  ...übernehme alle anderen 1:1

  F ist Wald, da Untergraph von Baum T.

  Fügt man weitere Kante zu F hinzu, so hätte T einen Kreis. Widerspruch.



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

  Funktion g erzeugt aus Lösung von T (G', w') eine Lösung F von (G, w)

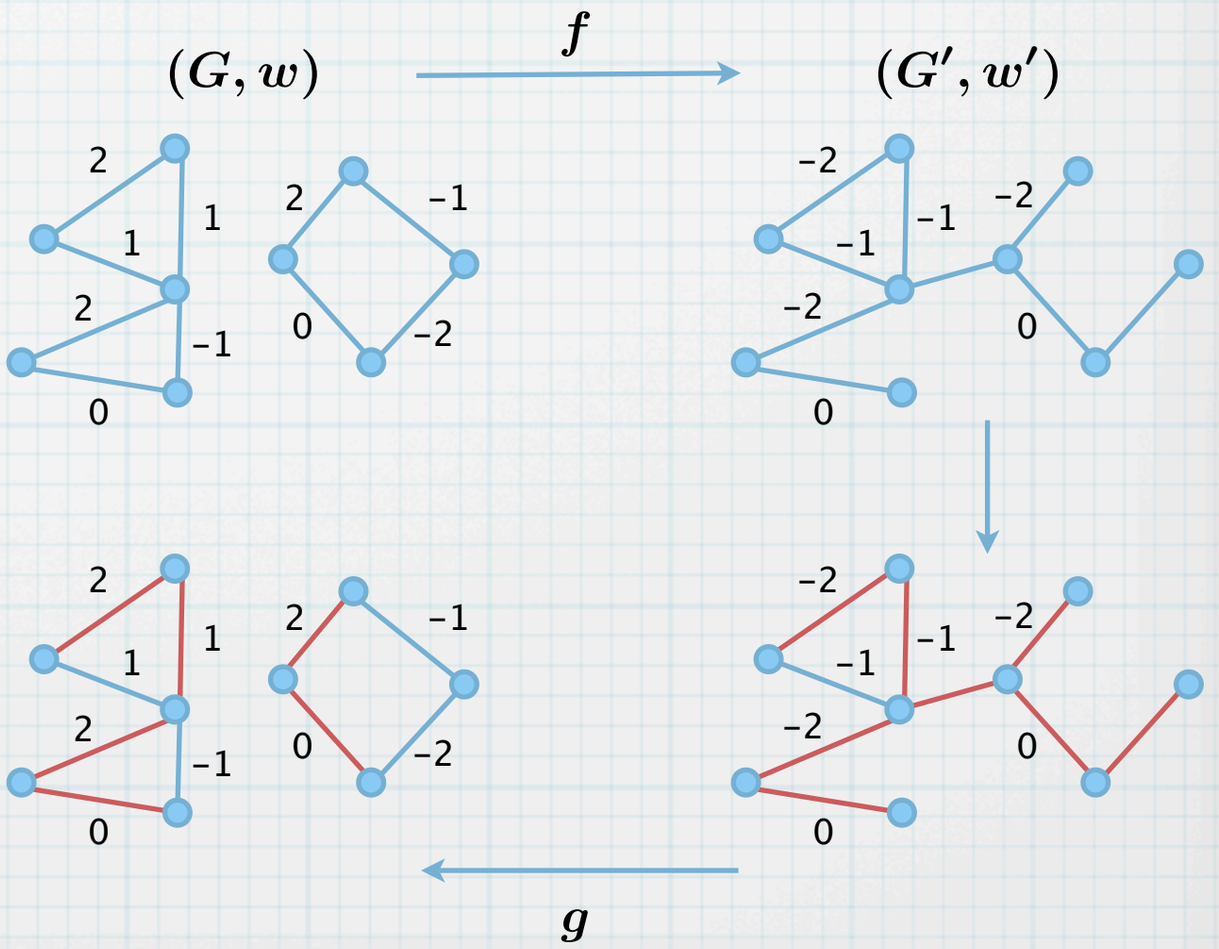
  ...lasse alle Kanten aus K weg

  ...übernehme alle anderen 1:1

  F ist Wald, da Untergraph von Baum T.

  Fügt man weitere Kante zu F hinzu, so hätte T einen Kreis. Widerspruch.

  Tauscht man Kante(n) in F aus, so muss man gleiches auch in T tun. Widerspruch zur Optimalität des Baums.



Minimale Spannbäume und maximale Wälder

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Rightarrow$):

  Sei (G, w) eine Instanz des max. gew. Waldprob.

  Funktion f erzeugt Instanz (G', w') des min. aufsp. Baumprob.

  ...entferne aus G alle Kanten e mit $w(e) < 0$,

  ...setze $w'(e) := -w(e)$ für alle verbleibenden Kanten,

  ...füge minimale Kantenmenge K (Gewicht egal) hinzu, so dass G' zusammenhängend.

  Funktion g erzeugt aus Lösung von T (G', w') eine Lösung F von (G, w)

  ...lasse alle Kanten aus K weg

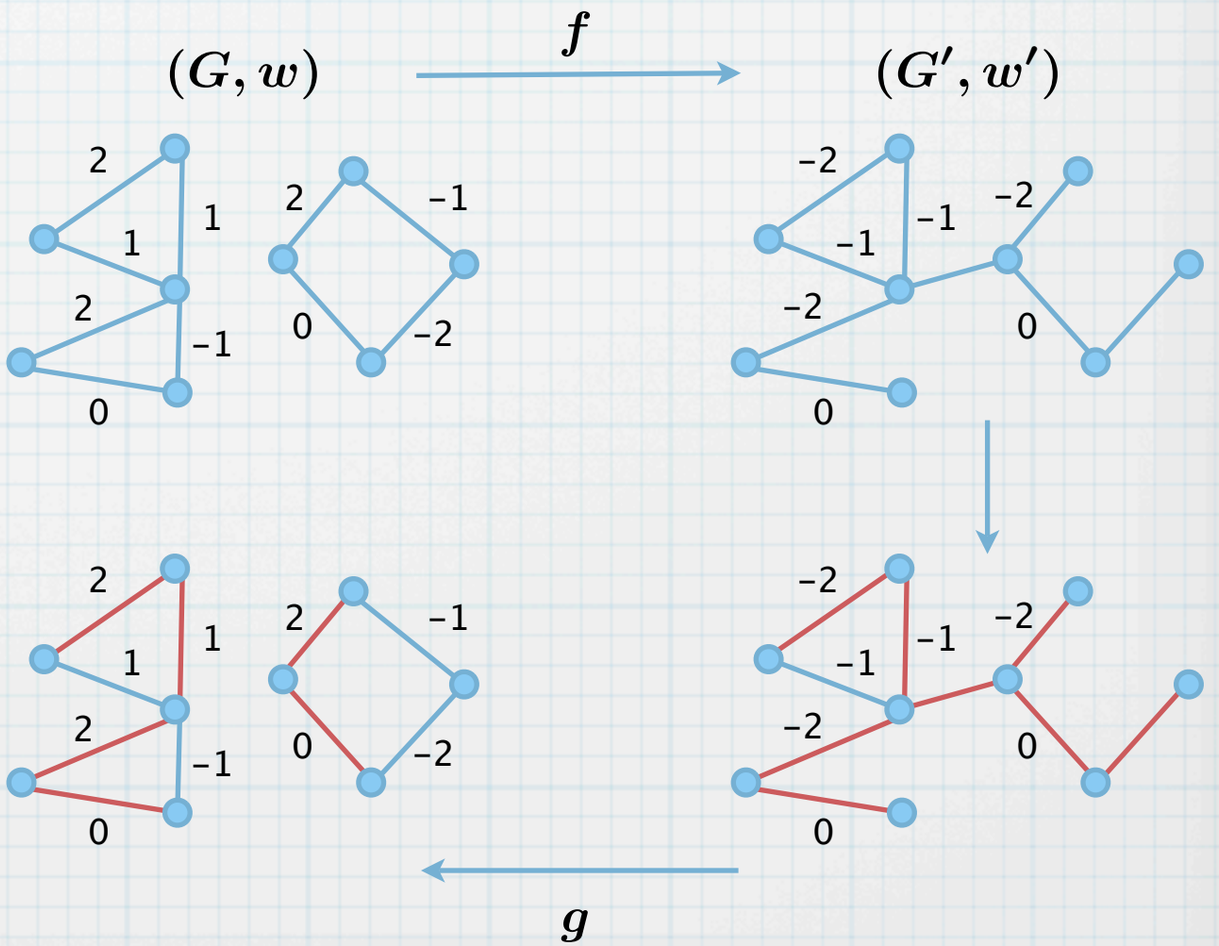
  ...übernehme alle anderen 1:1

  F ist Wald, da Untergraph von Baum T.

  Fügt man weitere Kante zu F hinzu, so hätte T einen Kreis. Widerspruch.

  Tauscht man Kante(n) in F aus, so muss man gleiches auch in T tun. Widerspruch zur Optimalität des Baums.

  Also: max. gew. Waldproblem kann linear auf min. aufsp. Baumproblem reduziert werden.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

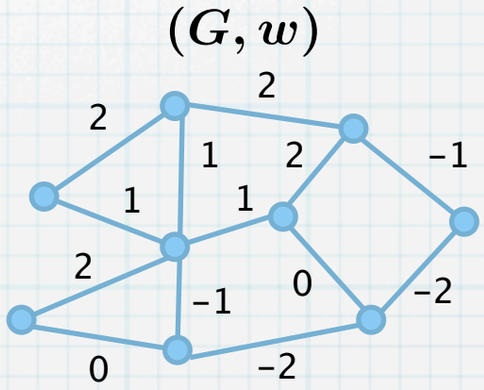
Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

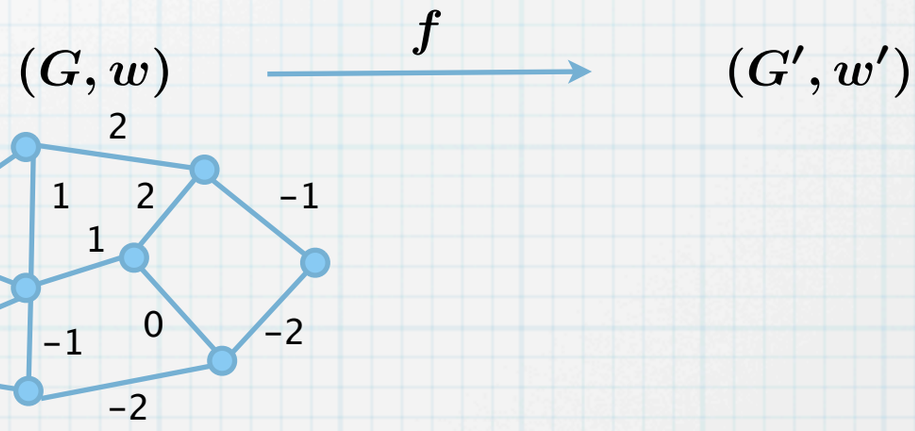


Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.



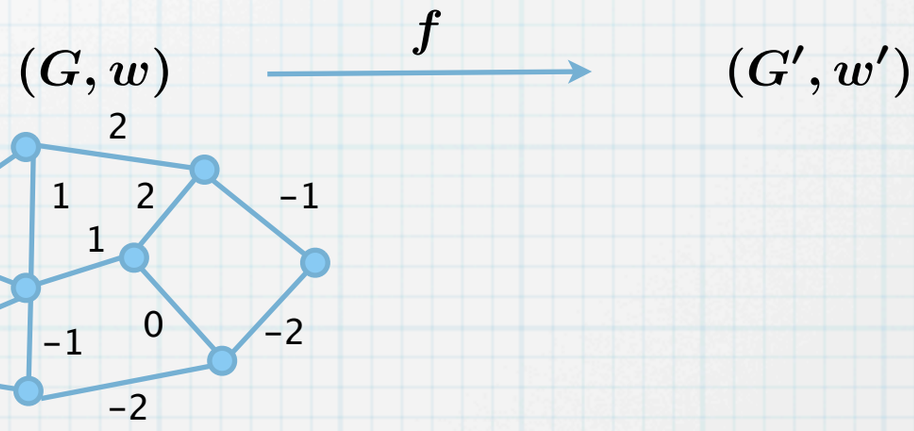
Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.



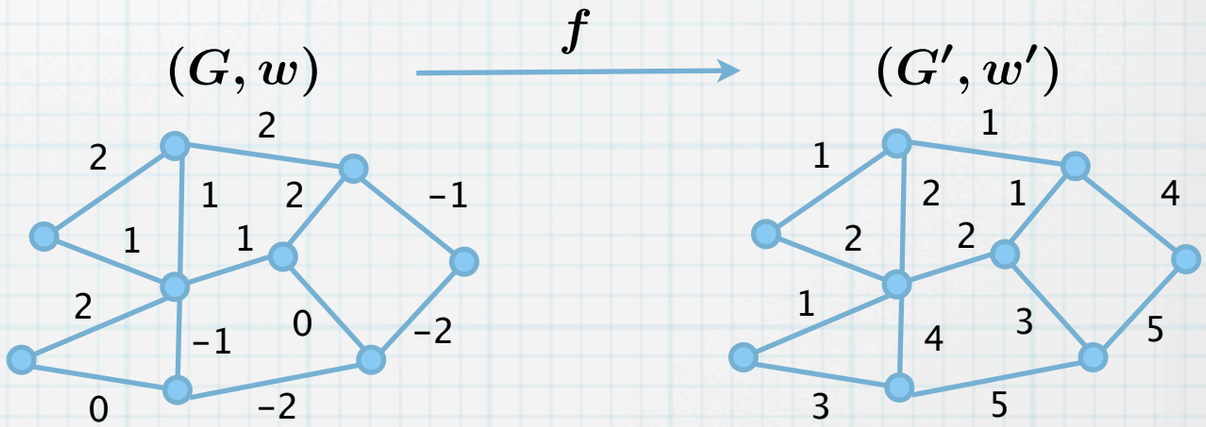
Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.



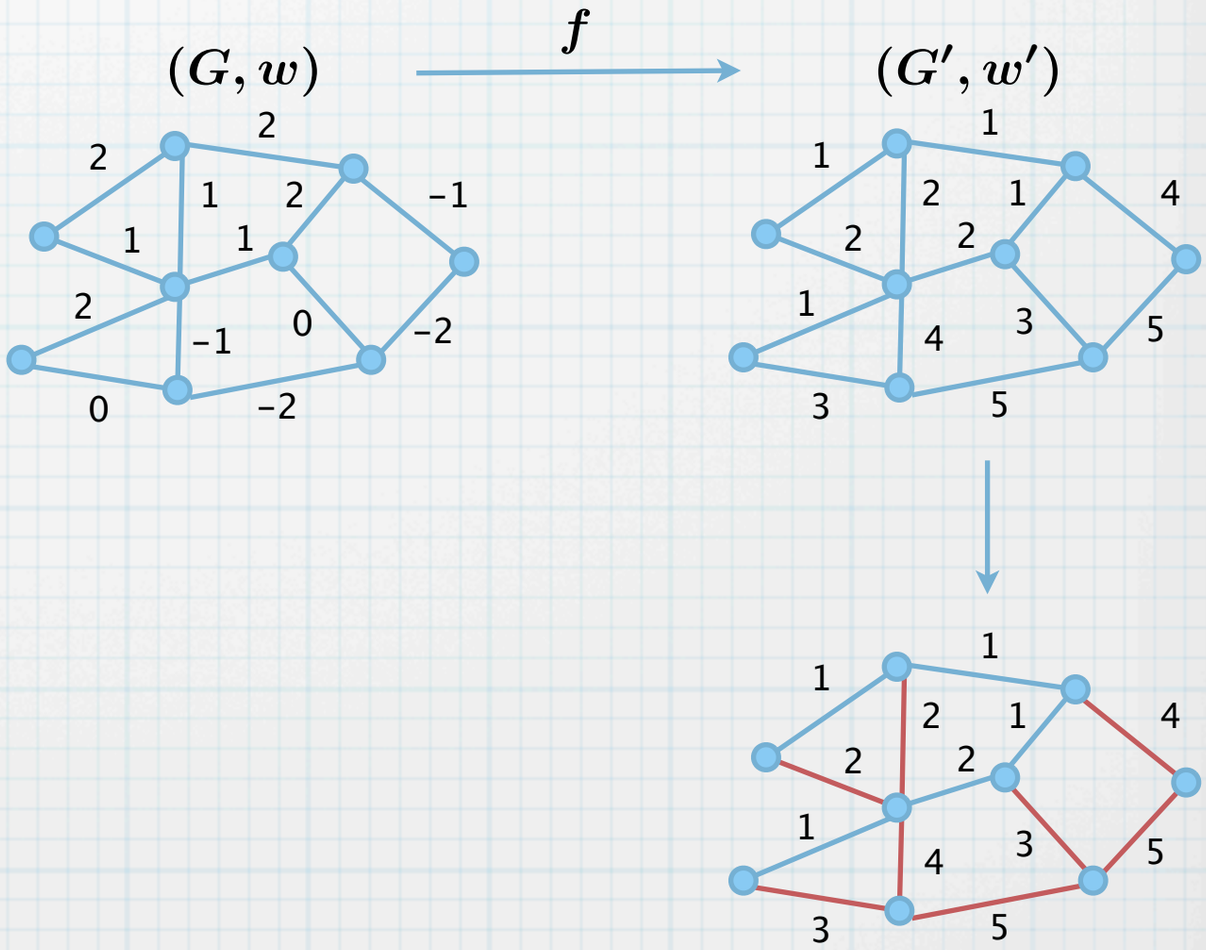
Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

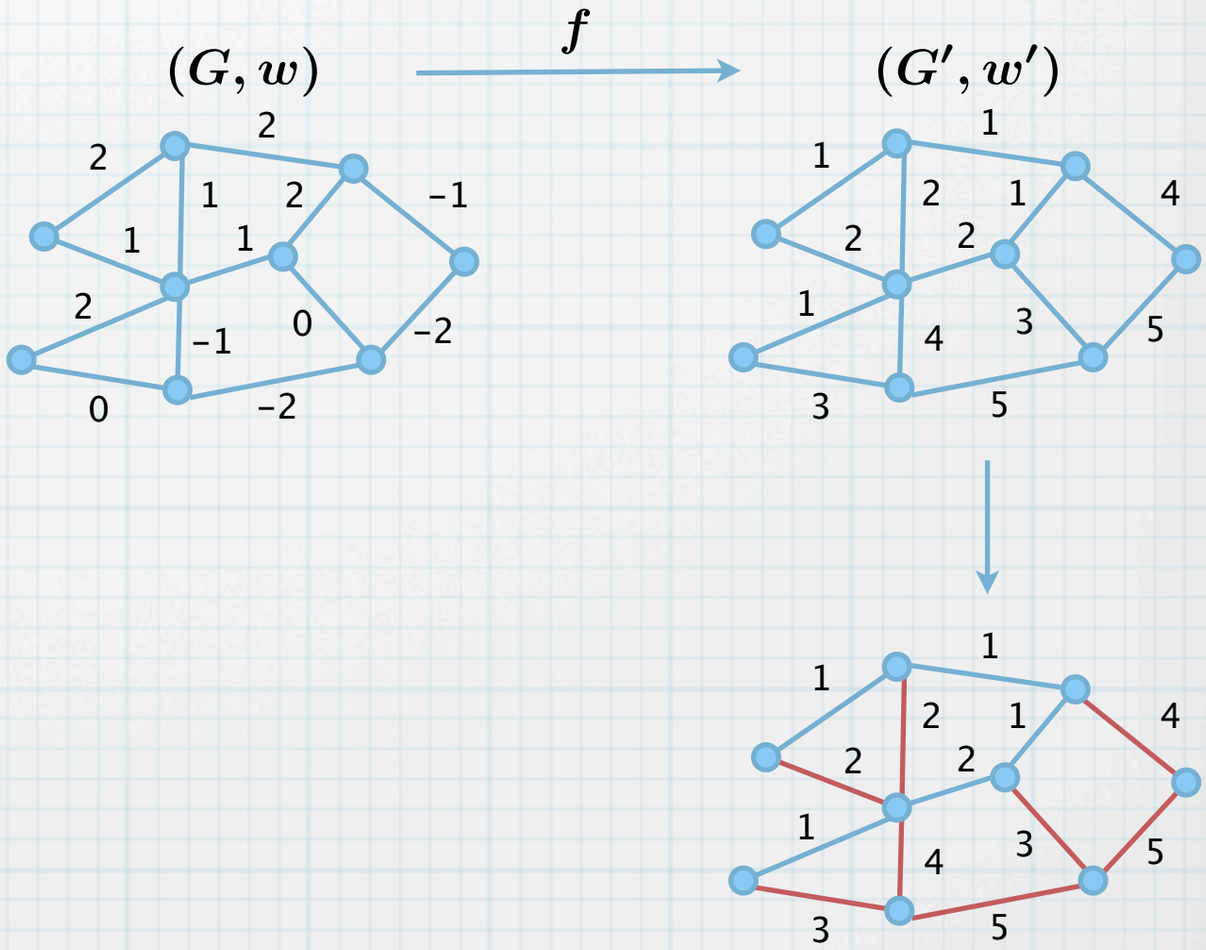
* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')



Minimale Spannbäume und maximale Wälder (Fortsetzung)

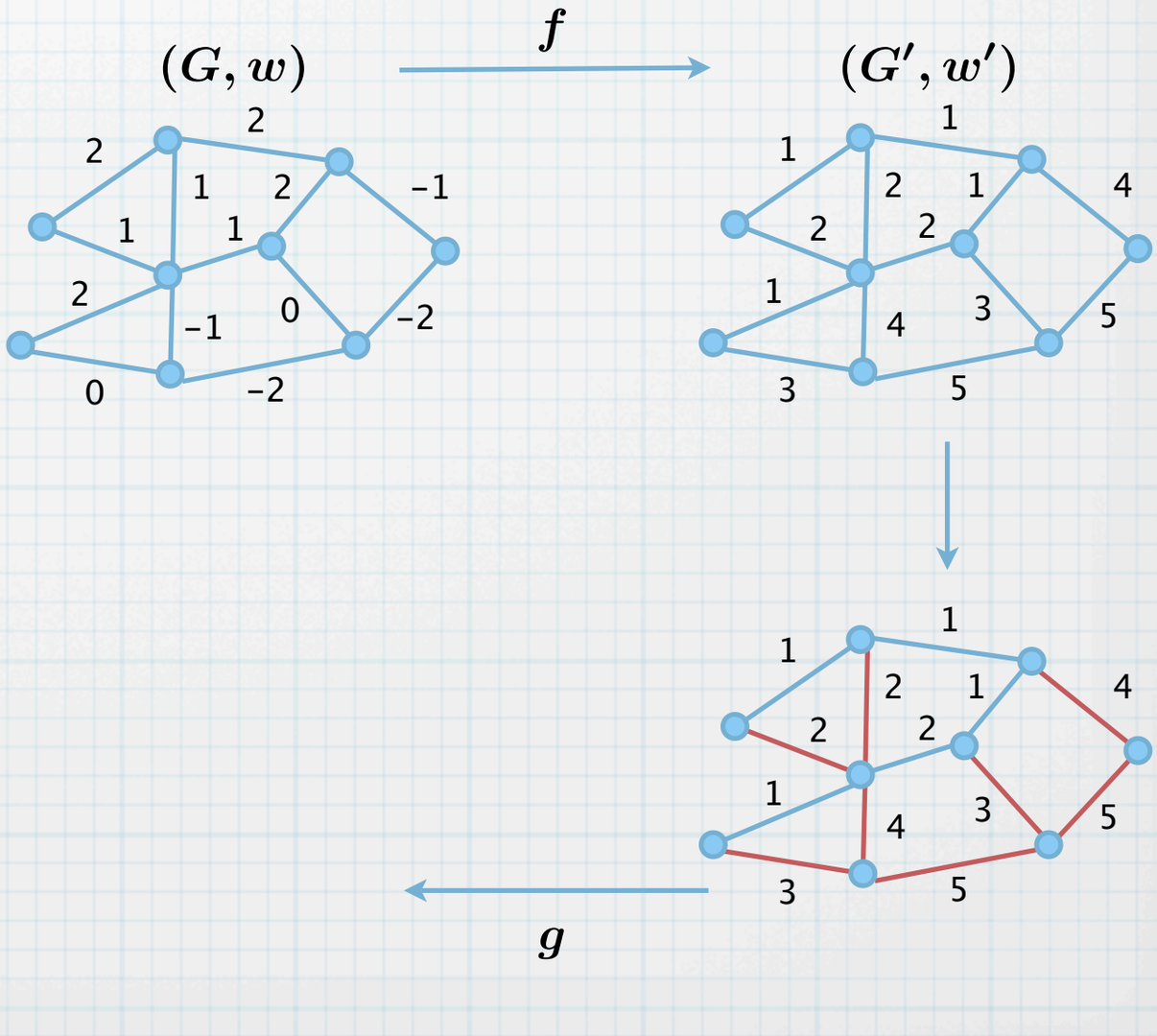
* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

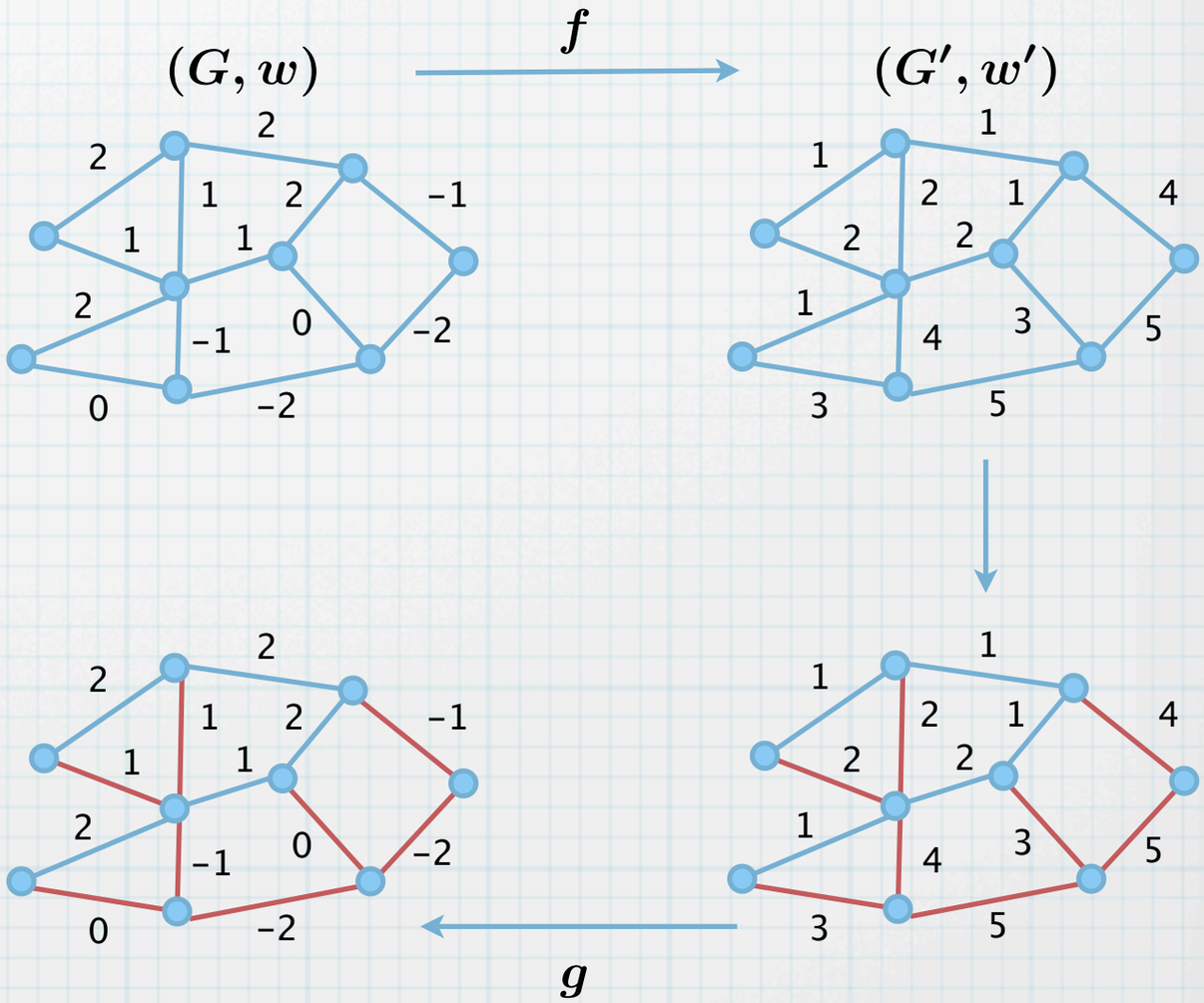
  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')

  ...übernehme alle Kanten 1:1.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

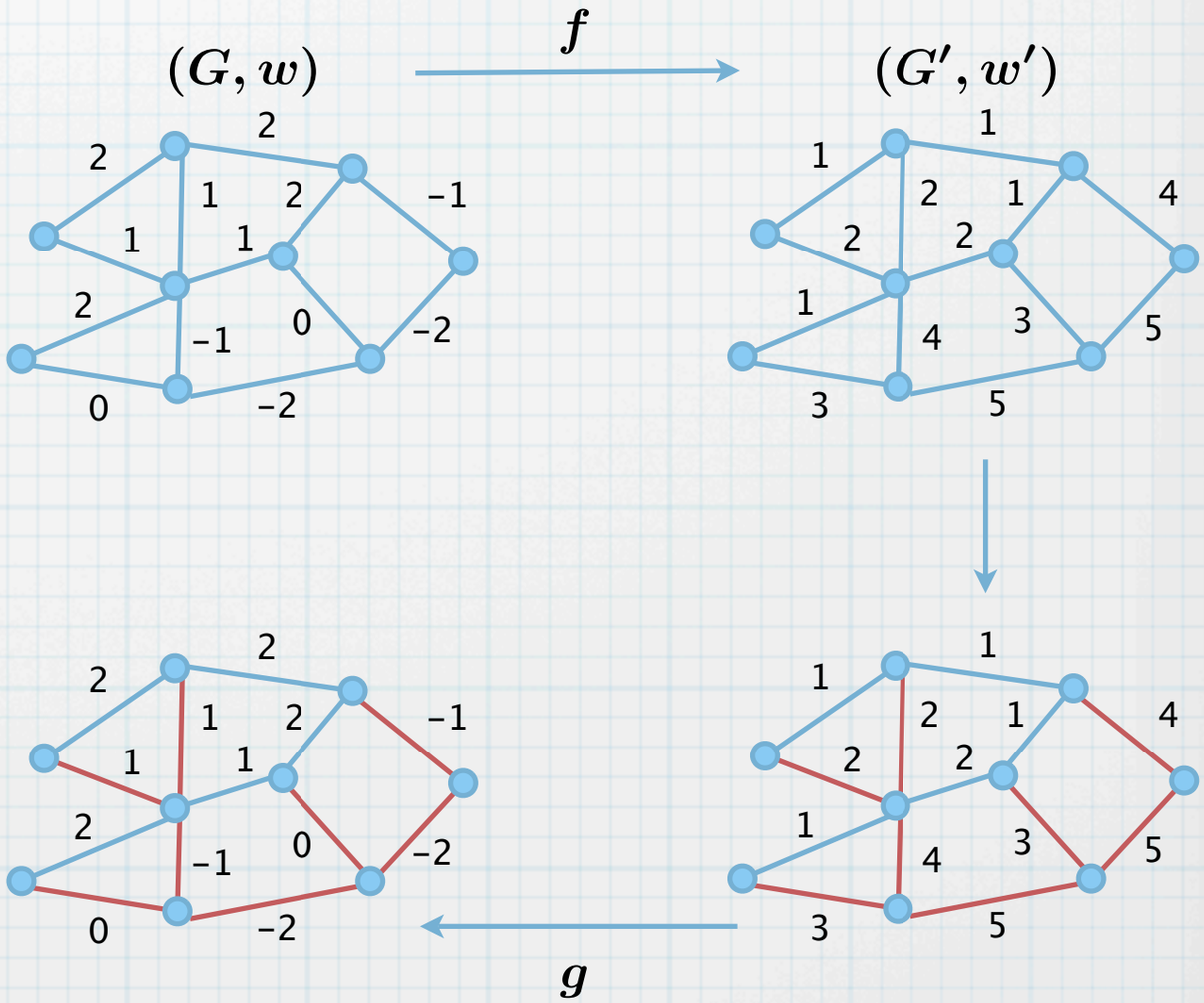
  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')

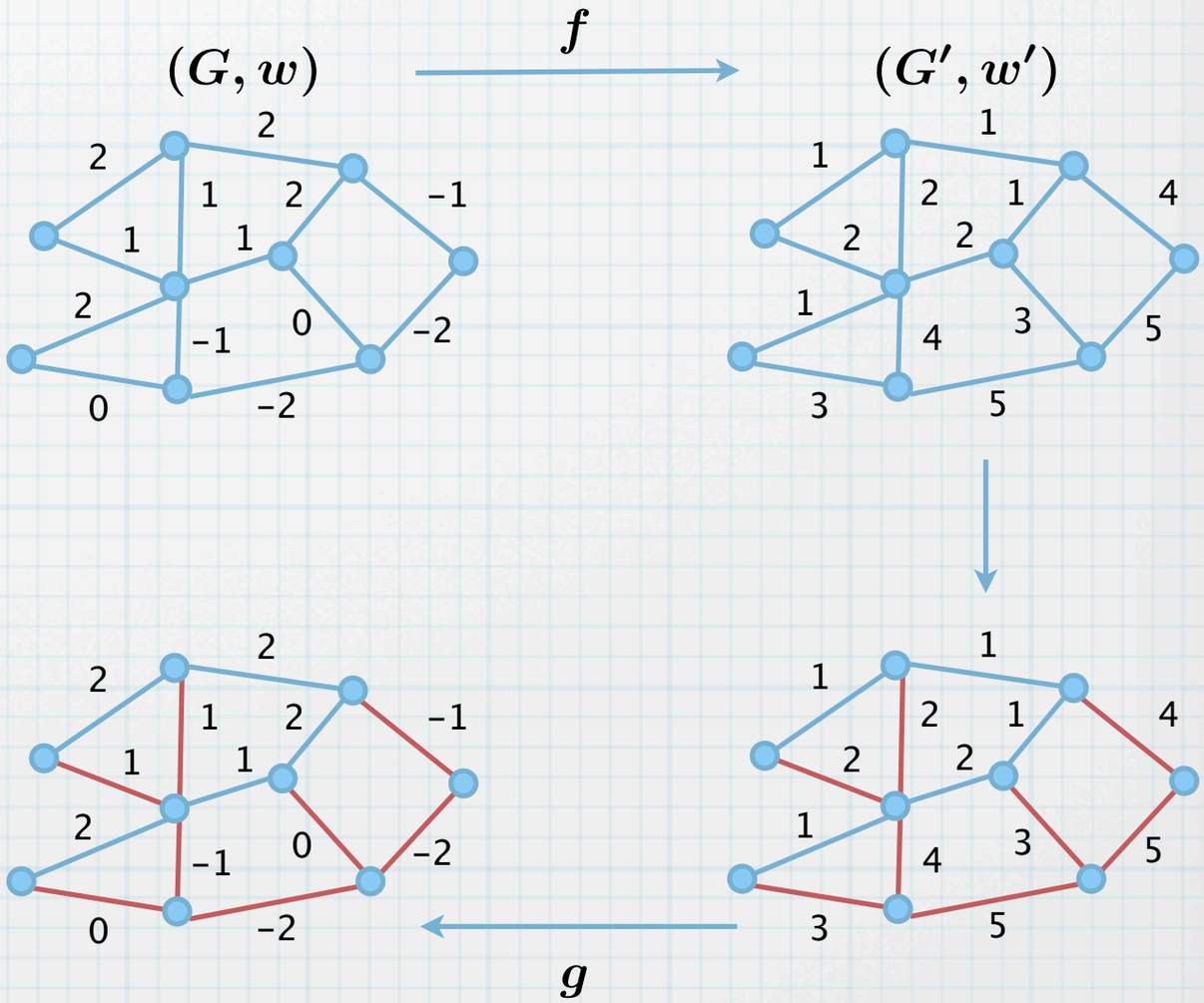
  ...übernehme alle Kanten 1:1.

  Entweder ist F zusammenhängend.



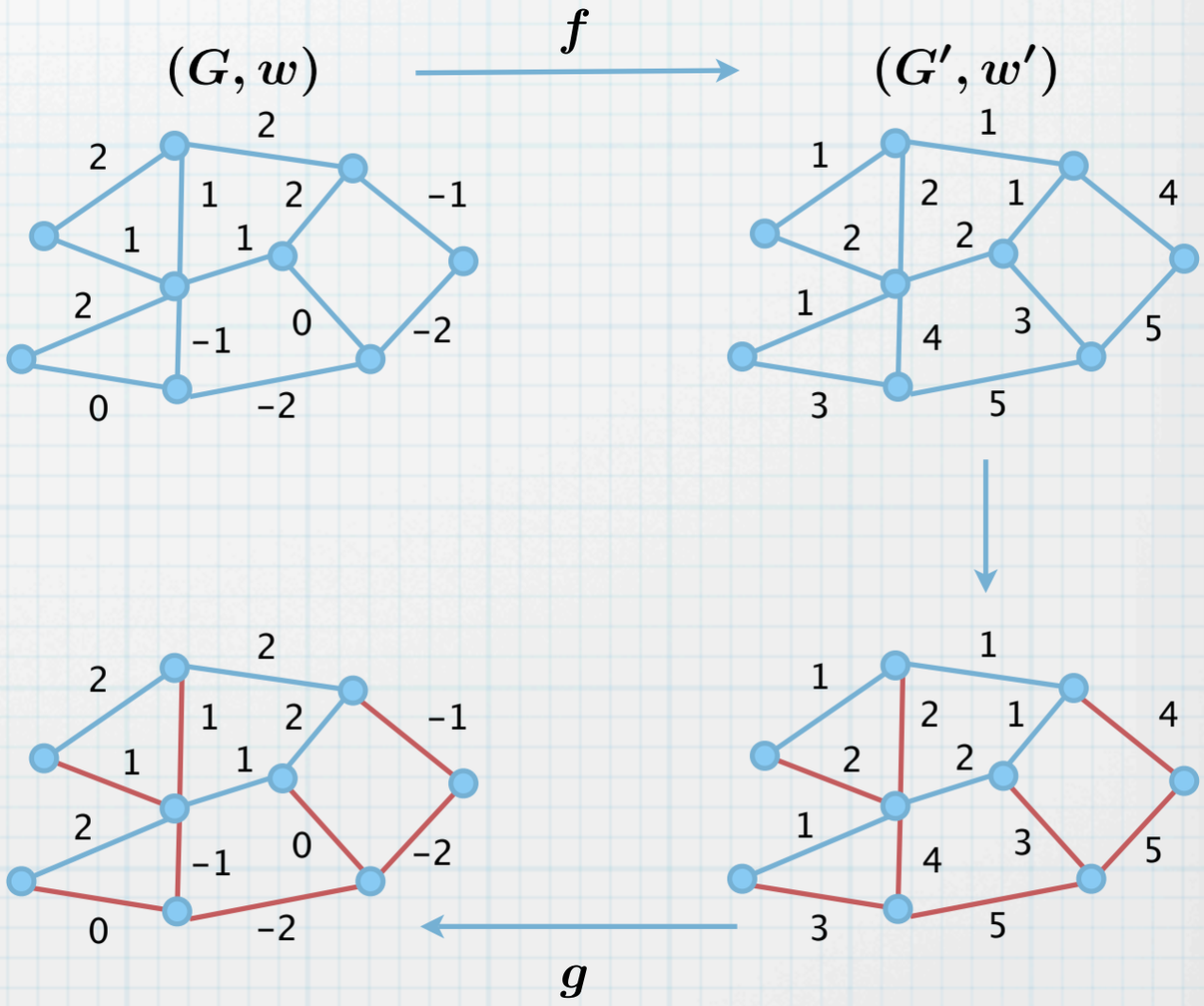
Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):
  Sei (G, w) eine Instanz des min. aufsp. Baumprob.
  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.
  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.
  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')
  ...übernehme alle Kanten 1:1.
  Entweder ist F zusammenhängend.
  Dann ist auch T zusammenhängend.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):
  Sei (G, w) eine Instanz des min. aufsp. Baumprob.
  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.
  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.
  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')
  ...übernehme alle Kanten 1:1.
  Entweder ist F zusammenhängend.
  Dann ist auch T zusammenhängend.
  Zudem ist T kreisfrei (geerbt von F).



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')

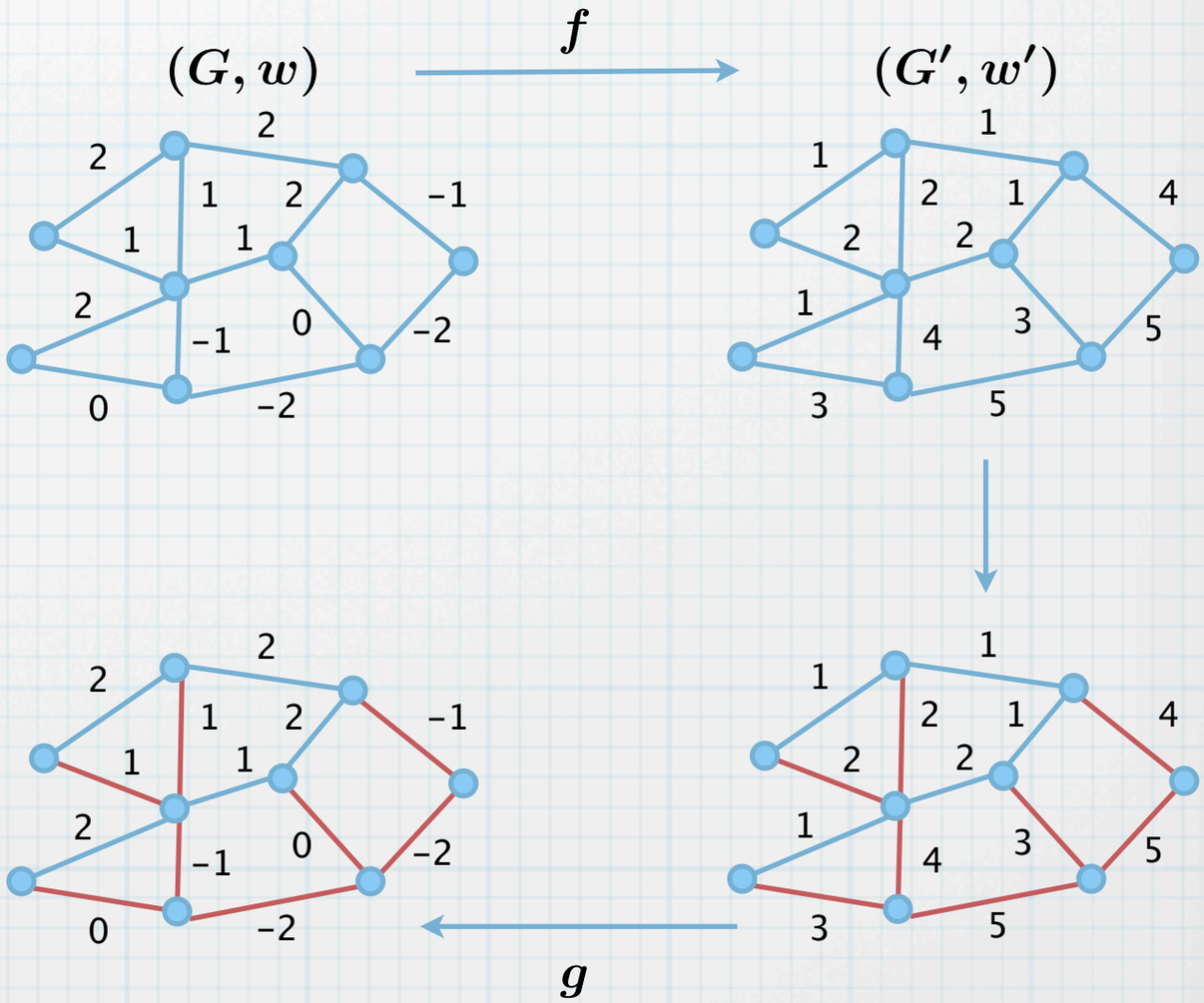
  ...übernehme alle Kanten 1:1.

  Entweder ist F zusammenhängend.

  Dann ist auch T zusammenhängend.

  Zudem ist T kreisfrei (geerbt von F).

  Da $w'(e) > 0$, enthält T alle Knoten von G. Somit ist T ein Spannbaum.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')

  ...übernehme alle Kanten 1:1.

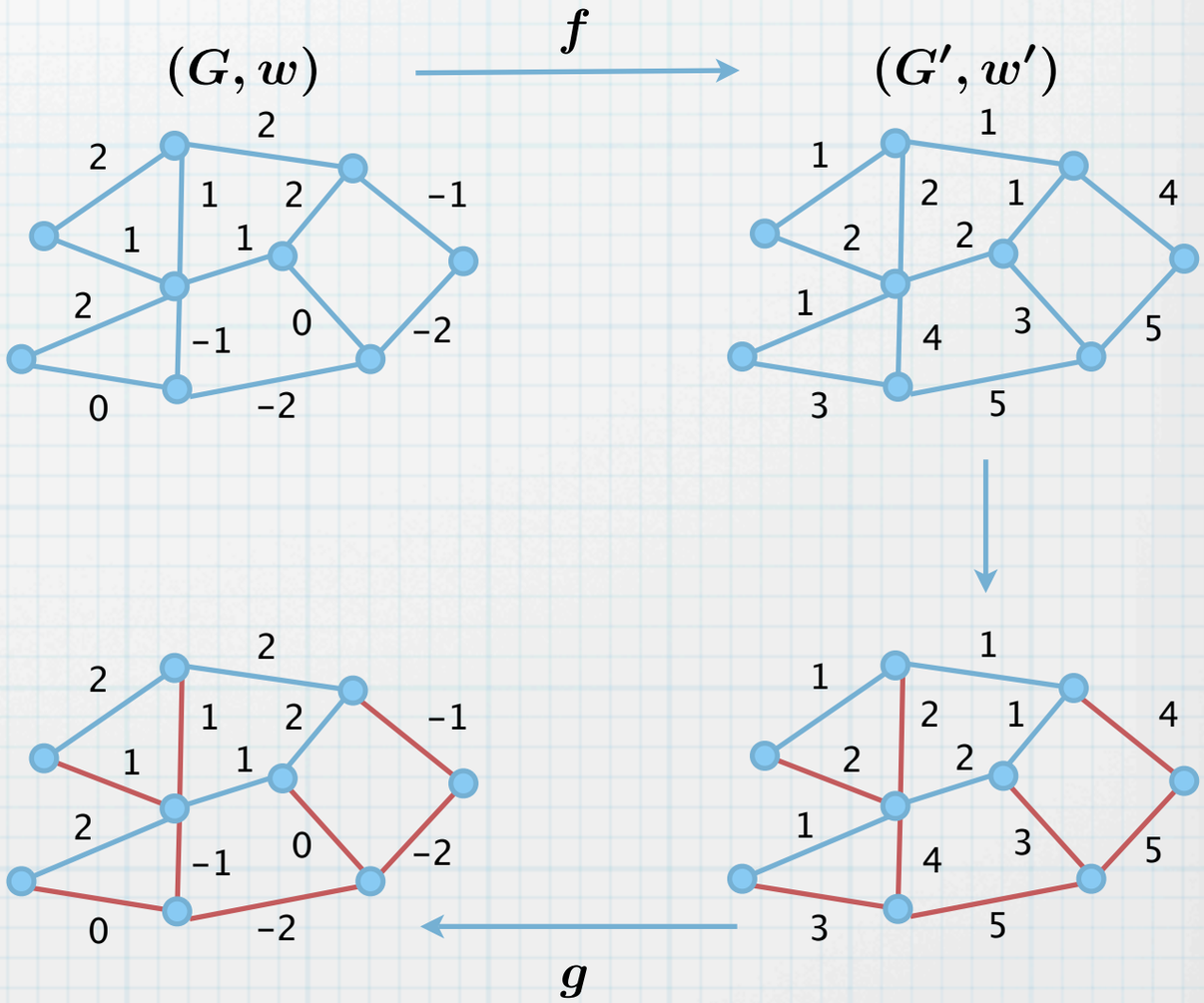
  Entweder ist F zusammenhängend.

  Dann ist auch T zusammenhängend.

  Zudem ist T kreisfrei (geerbt von F).

  Da $w'(e) > 0$, enthält T alle Knoten von G. Somit ist T ein Spannbaum.

  Tauscht man Kanten in T aus, um einen „minimaleren“ Spannbaum zu erhalten, so müsste man dieses auch in F tun. Widerspruch zur Maximalität des Waldes.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')

  ...übernehme alle Kanten 1:1.

  Entweder ist F zusammenhängend.

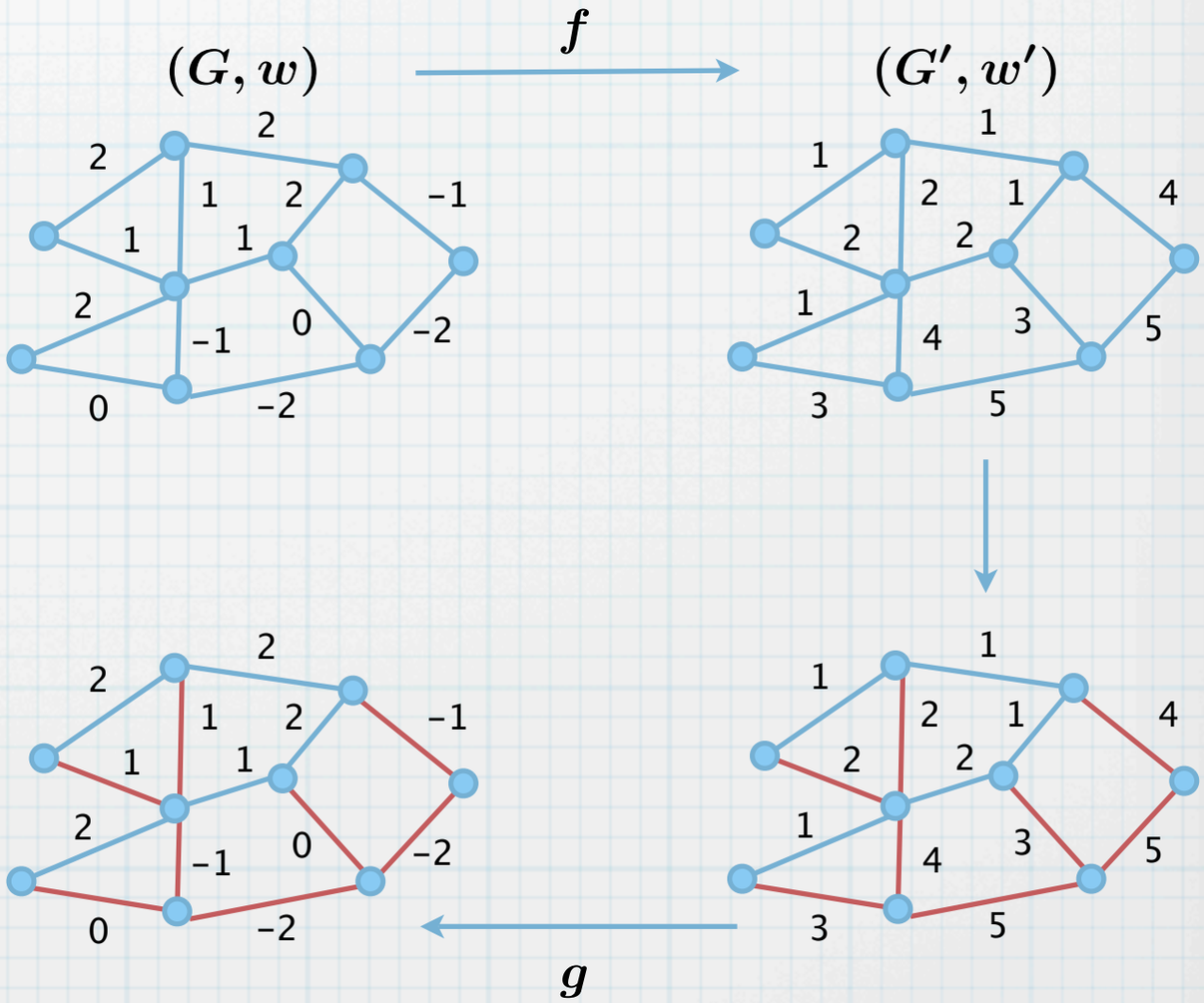
  Dann ist auch T zusammenhängend.

  Zudem ist T kreisfrei (geerbt von F).

  Da $w'(e) > 0$, enthält T alle Knoten von G. Somit ist T ein Spannbaum.

  Tauscht man Kanten in T aus, um einen „minimaleren“ Spannbaum zu erhalten, so müsste man dieses auch in F tun. Widerspruch zur Maximalität des Waldes.

Oder F ist nicht zusammenhängend.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')

  ...übernehme alle Kanten 1:1.

  Entweder ist F zusammenhängend.

  Dann ist auch T zusammenhängend.

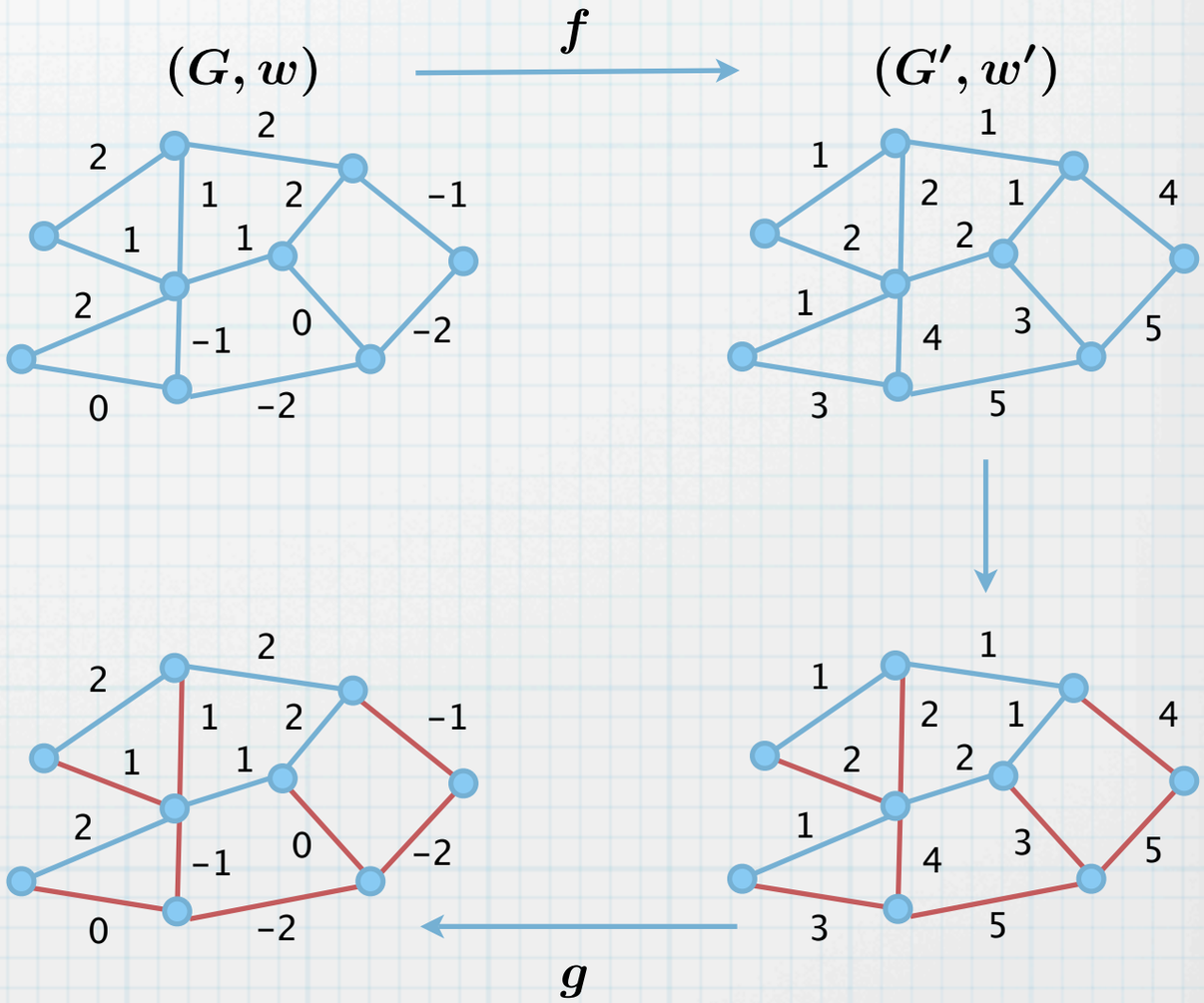
  Zudem ist T kreisfrei (geerbt von F).

  Da $w'(e) > 0$, enthält T alle Knoten von G. Somit ist T ein Spannbaum.

  Tauscht man Kanten in T aus, um einen „minimaleren“ Spannbaum zu erhalten, so müsste man dieses auch in F tun. Widerspruch zur Maximalität des Waldes.

  Oder F ist nicht zusammenhängend.

  Dann ist G nicht zusammenhängend, und es gibt keinen Spannbaum.



Minimale Spannbäume und maximale Wälder (Fortsetzung)

* **Satz 16**:
  Maximales gewichtetes Waldproblem und minimales Spannbaumproblem sind äquivalent.
* Beweis ($\Leftarrow$):

  Sei (G, w) eine Instanz des min. aufsp. Baumprob.

  Funktion f erzeugt Instanz (G', w') des max. gew. Waldprob.

  ...setze $w'(e) := M - w(e)$, mit $M := \max\{-w(e) : e \in E\} + 1$. Dann ist $w'(e) > 0$.

  Funktion g erzeugt Lösung T von (G, w) aus Lösung F von (G', w')

  ...übernehme alle Kanten 1:1.

  Entweder ist F zusammenhängend.

  Dann ist auch T zusammenhängend.

  Zudem ist T kreisfrei (geerbt von F).

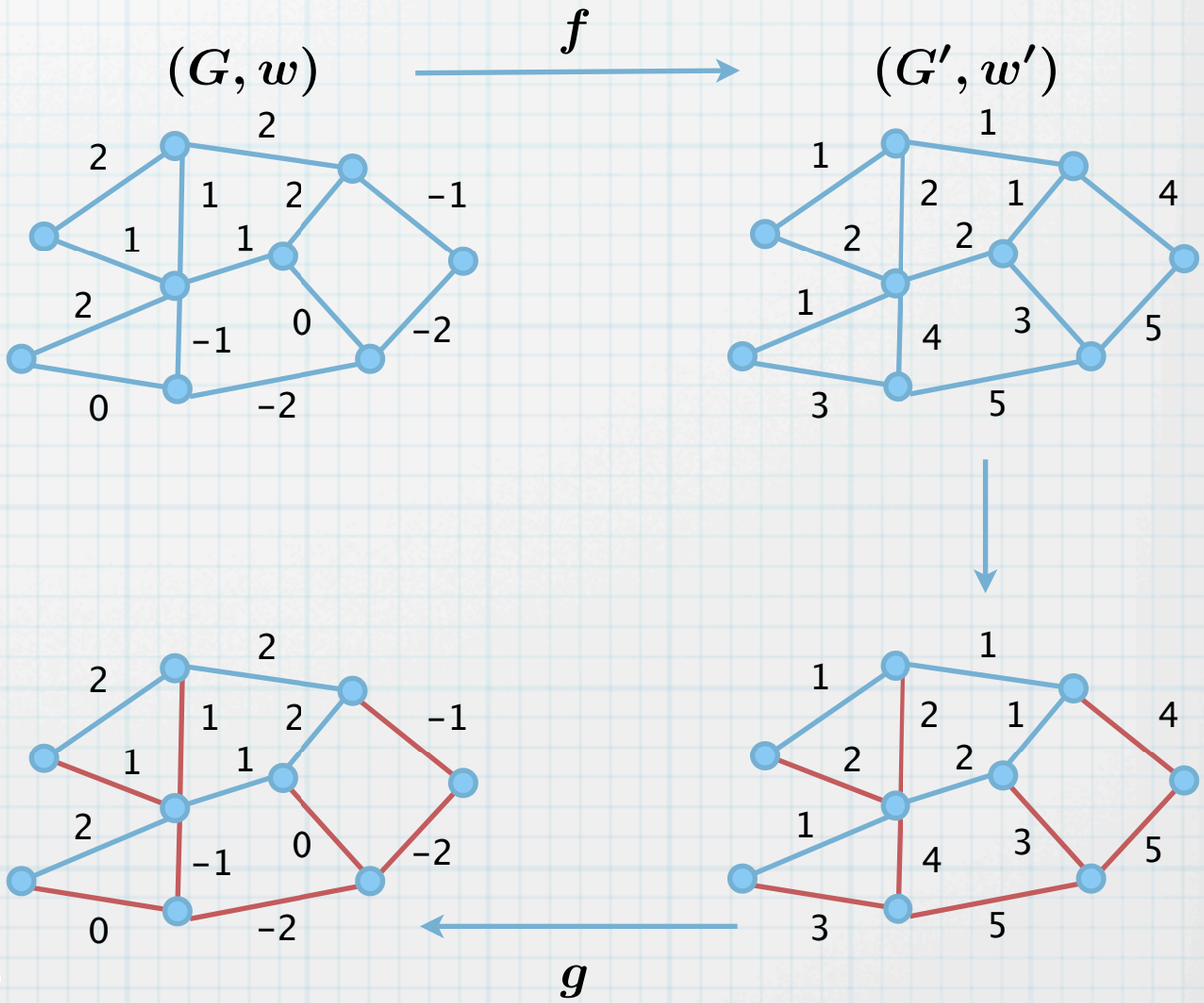
  Da $w'(e) > 0$, enthält T alle Knoten von G. Somit ist T ein Spannbaum.

  Tauscht man Kanten in T aus, um einen „minimaleren“ Spannbaum zu erhalten, so müsste man dieses auch in F tun. Widerspruch zur Maximalität des Waldes.

  Oder F ist nicht zusammenhängend.

  Dann ist G nicht zusammenhängend, und es gibt keinen Spannbaum.

  Also: max. gew. Waldproblem und min. aufsp. Baumproblem sind äquivalent.



Direkte Anwendungen des Spannbaumproblems

Direkte Anwendungen des Spannbaumproblems

* Entwurf physikalischer Netz-Systeme, bei denen Ausfallsicherheit keine Rolle spielt

Direkte Anwendungen des Spannbaumproblems

* Entwurf physikalischer Netz-Systeme, bei denen Ausfallsicherheit keine Rolle spielt
* Beispiele:
  * Straßen
  * Stromnetz
  * Wasser-, Gas- oder Ölrohrleitungsnetze
  * Entwurf von Mikrochips
  * Vernetzung von Rechnern

Direkte Anwendungen des Spannbaumproblems

- Entwurf physikalischer Netz-Systeme, bei denen Ausfallsicherheit keine Rolle spielt
- Beispiele:
  - Straßen
  - Stromnetz
  - Wasser-, Gas- oder Ölrohrleitungsnetze
  - Entwurf von Mikrochips
  - Vernetzung von Rechnern
- Knoten: Teilnehmer / Erzeuger / Abnehmer

Direkte Anwendungen des Spannbaumproblems

- Entwurf physikalischer Netz-Systeme, bei denen Ausfallsicherheit keine Rolle spielt
- Beispiele:
  - Straßen
  - Stromnetz
  - Wasser-, Gas- oder Ölrohrleitungsnetze
  - Entwurf von Mikrochips
  - Vernetzung von Rechnern
- Knoten: Teilnehmer / Erzeuger / Abnehmer
- Kanten: Mögliche Verbindung zwischen Knoten

Direkte Anwendungen des Spannbaumproblems

- Entwurf physikalischer Netz-Systeme, bei denen Ausfallsicherheit keine Rolle spielt
- Beispiele:
  - Straßen
  - Stromnetz
  - Wasser-, Gas- oder Ölrohrleitungsnetze
  - Entwurf von Mikrochips
  - Vernetzung von Rechnern
- Knoten: Teilnehmer / Erzeuger / Abnehmer
- Kanten: Mögliche Verbindung zwischen Knoten
- Gewichte: Verbindungskosten

Indirekte Anwendungen des Spannbaumproblems

Indirekte Anwendungen des Spannbaumproblems

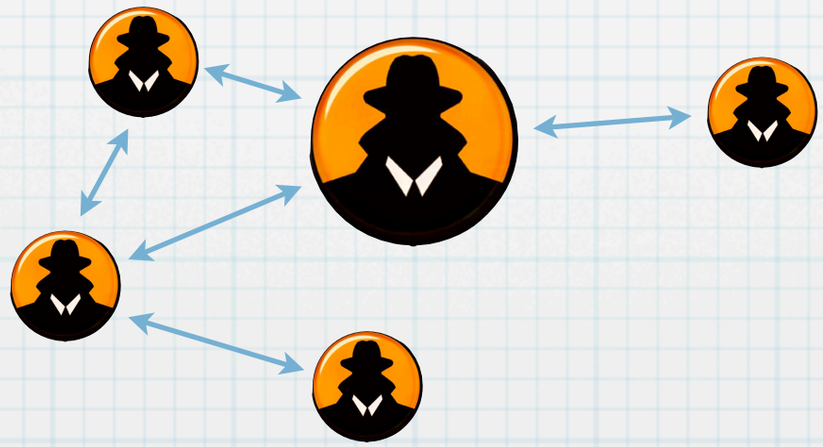
* Optimale Nachrichtenweitergabe

Indirekte Anwendungen des Spannbaumproblems

* Optimale Nachrichtenweitergabe
* Geheimdienst hat n Agenten (Knoten)

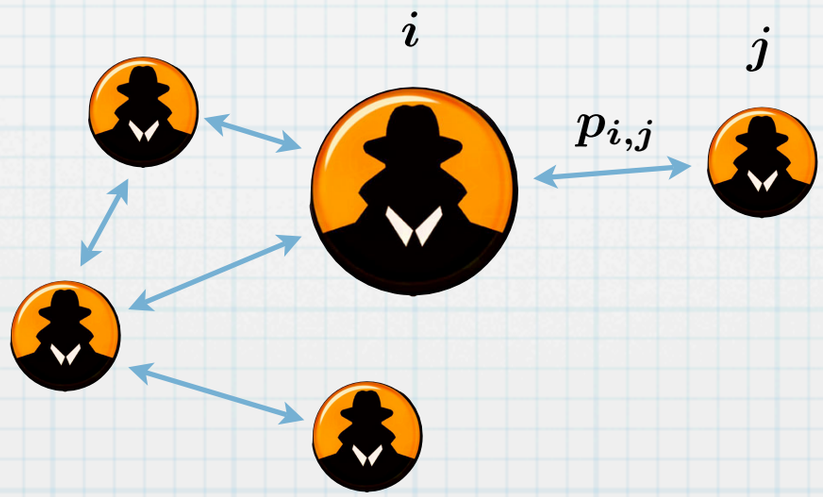
Indirekte Anwendungen des Spannbaumproblems

* Optimale Nachrichtenweitergabe
* Geheimdienst hat n Agenten (Knoten)
* Jeder Agent kennt einige andere Agenten (Kanten)



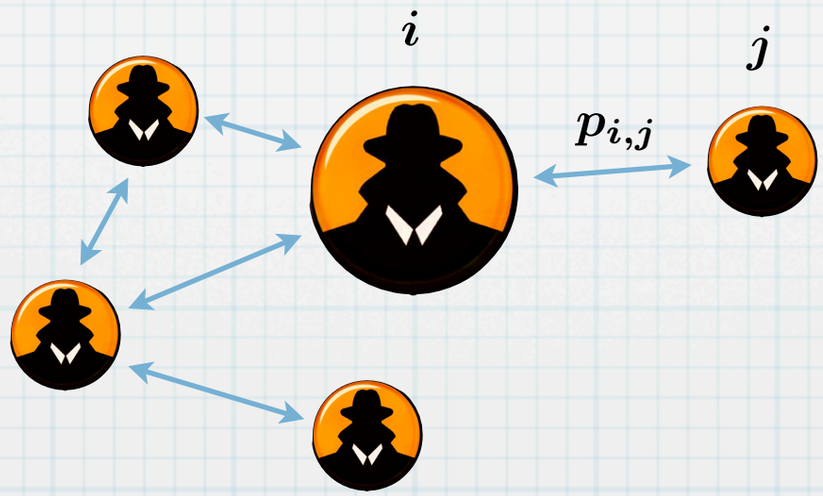
Indirekte Anwendungen des Spannbaumproblems

* Optimale Nachrichtenweitergabe
* Geheimdienst hat n Agenten (Knoten)
* Jeder Agent kennt einige andere Agenten (Kanten)
* Wenn Agent i mit Agent j Nachricht tauscht, wird sie mit Wahrscheinlichkeit $p_{i,j}$ abgefangen



Indirekte Anwendungen des Spannbaumproblems

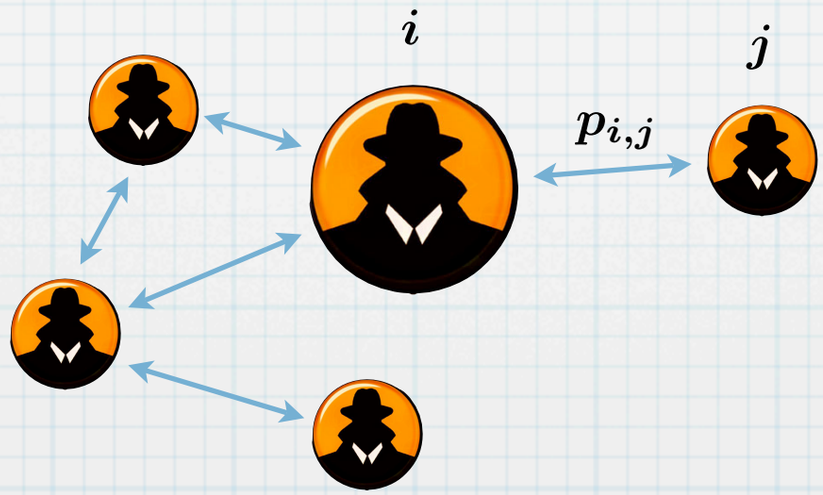
* Optimale Nachrichtenweitergabe
* Geheimdienst hat n Agenten (Knoten)
* Jeder Agent kennt einige andere Agenten (Kanten)
* Wenn Agent i mit Agent j Nachricht tauscht, wird sie mit Wahrscheinlichkeit $p_{i,j}$ abgefangen
* Problem: Wie kann eine Nachricht an alle Agenten übermittelt werden, so dass sie mit minimaler Wahrscheinlichkeit abgefangen wird?



Indirekte Anwendungen des Spannbaumproblems

- Optimale Nachrichtenweitergabe
- Geheimdienst hat n Agenten (Knoten)
- Jeder Agent kennt einige andere Agenten (Kanten)
- Wenn Agent i mit Agent j Nachricht tauscht, wird sie mit Wahrscheinlichkeit $p_{i,j}$ abgefangen
- Problem: Wie kann eine Nachricht an alle Agenten übermittelt werden, so dass sie mit minimaler Wahrscheinlichkeit abgefangen wird?
- Modell: Finde Spannbaum T, der die Gesamtwahrscheinlichkeit eines Abfangens der Nachricht minimiert:

$$\left(1 - \prod_{\{i,j\} \in T} (1 - p_{i,j})\right) \to \min \text{, äquivalent dazu: } \prod_{\{i,j\} \in T} (1 - p_{i,j}) \to \max$$

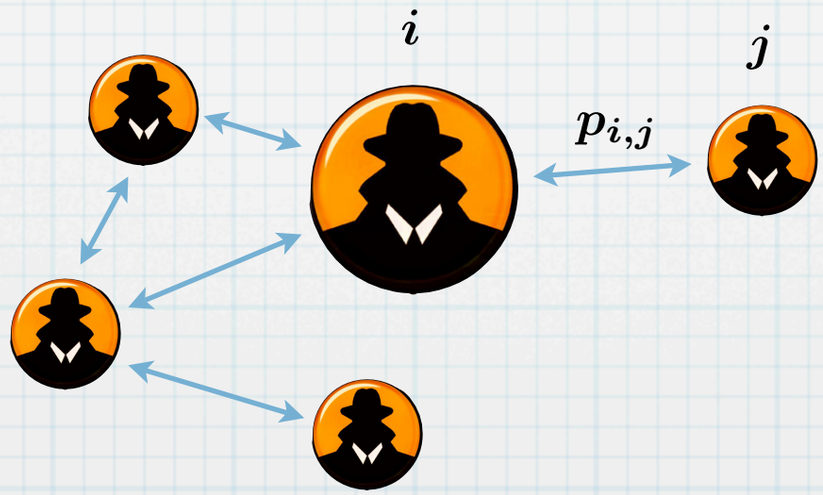


Indirekte Anwendungen des Spannbaumproblems

- Optimale Nachrichtenweitergabe
- Geheimdienst hat n Agenten (Knoten)
- Jeder Agent kennt einige andere Agenten (Kanten)
- Wenn Agent i mit Agent j Nachricht tauscht, wird sie mit Wahrscheinlichkeit $p_{i,j}$ abgefangen
- Problem: Wie kann eine Nachricht an alle Agenten übermittelt werden, so dass sie mit minimaler Wahrscheinlichkeit abgefangen wird?
- Modell: Finde Spannbaum T, der die Gesamtwahrscheinlichkeit eines Abfangens der Nachricht minimiert:

$$\left(1 - \prod_{\{i,j\} \in T} (1 - p_{i,j})\right) \to \min \text{, äquivalent dazu: } \prod_{\{i,j\} \in T} (1 - p_{i,j}) \to \max$$

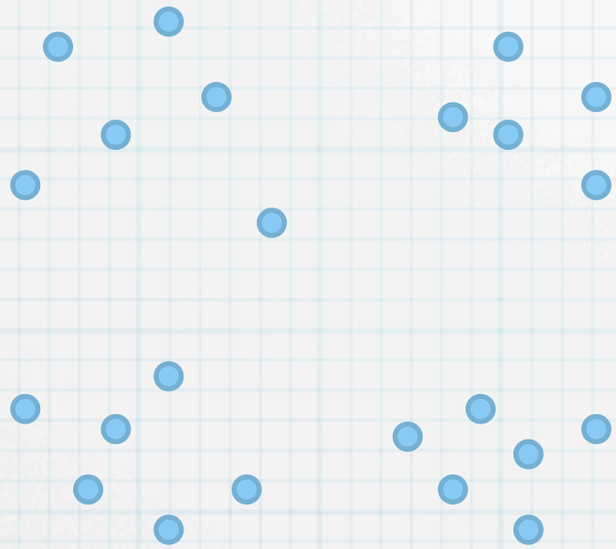
- Lösung: Setze Kantengewichte auf $\ln(1 - p_{i,j})$, und bestimme maximalen Spannbaum



Indirekte Anwendungen des Spannbaumproblems (Forts.)

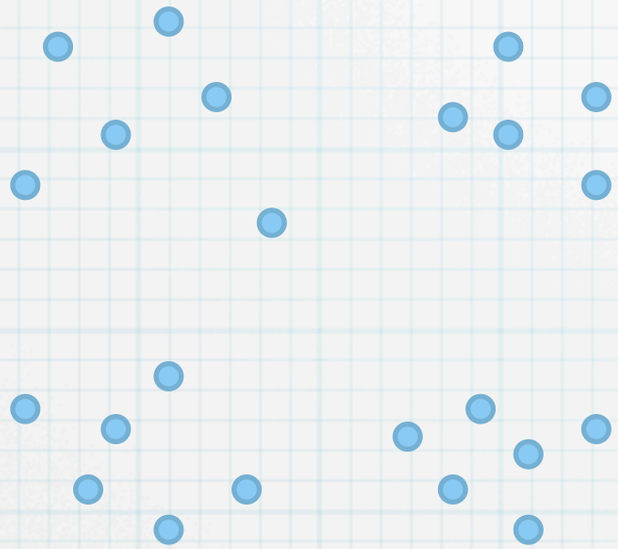
Indirekte Anwendungen des Spannbaumproblems (Forts.)

* Cluster-Analyse (Data-Mining)



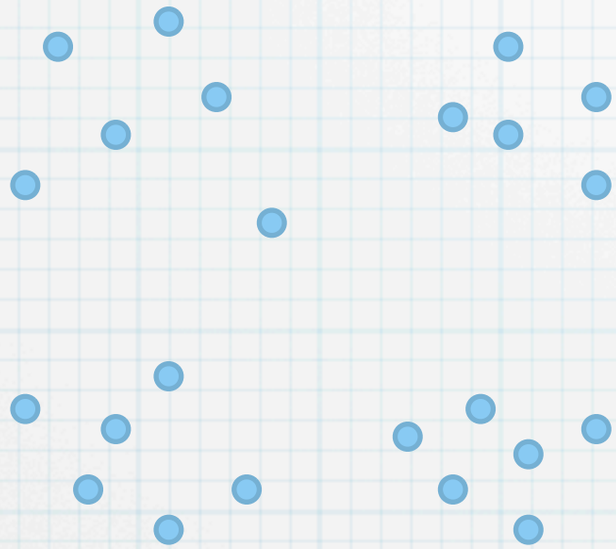
Indirekte Anwendungen des Spannbaumproblems (Forts.)

* Cluster-Analyse (Data-Mining)
* Einteilen einer großen Zahl von einzelnen Datensätzen in wenige charakteristische Gruppen



Indirekte Anwendungen des Spannbaumproblems (Forts.)

* Cluster-Analyse (Data-Mining)
* Einteilen einer großen Zahl von einzelnen Datensätzen in wenige charakteristische Gruppen
* Berechne minimalen aufspannenden Baum



Indirekte Anwendungen des Spannbaumproblems (Forts.)

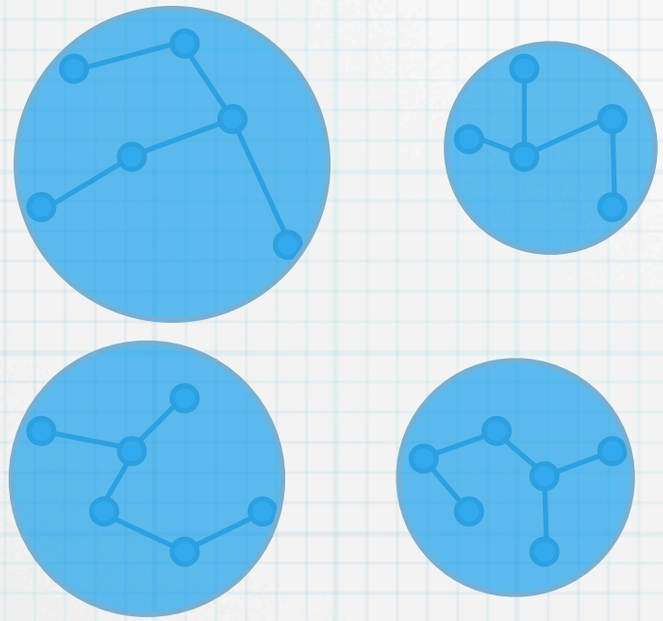
* Cluster-Analyse (Data-Mining)
* Einteilen einer großen Zahl von einzelnen Datensätzen in wenige charakteristische Gruppen
* Berechne minimalen aufspannenden Baum

Indirekte Anwendungen des Spannbaumproblems (Forts.)

- Cluster-Analyse (Data-Mining)
- Einteilen einer großen Zahl von einzelnen Datensätzen in wenige charakteristische Gruppen
- Berechne minimalen aufspannenden Baum
- Weglassen der k längsten Kanten liefert $k+1$ Cluster

Indirekte Anwendungen des Spannbaumproblems (Forts.)

- Cluster-Analyse (Data-Mining)
- Einteilen einer großen Zahl von einzelnen Datensätzen in wenige charakteristische Gruppen
- Berechne minimalen aufspannenden Baum
- Weglassen der k längsten Kanten liefert $k+1$ Cluster



Indirekte Anwendungen des Spannbaumproblems (Forts.)

Indirekte Anwendungen des Spannbaumproblems (Forts.)

* Effizientes Speichern von großen Datenmengen

Indirekte Anwendungen des Spannbaumproblems (Forts.)

* Effizientes Speichern von großen Datenmengen
* Gegeben sei ein zwei-dimensionales Datenfeld (Array)

1:	0	0	1	1	1	1	0	1	1	0
2:	0	1	0	1	0	1	1	1	0	1
3:	1	0	1	0	1	0	1	0	1	1
4:	1	0	1	1	0	1	1	1	1	0
5:	0	1	1	1	1	1	1	0	1	1

Indirekte Anwendungen des Spannbaumproblems (Forts.)

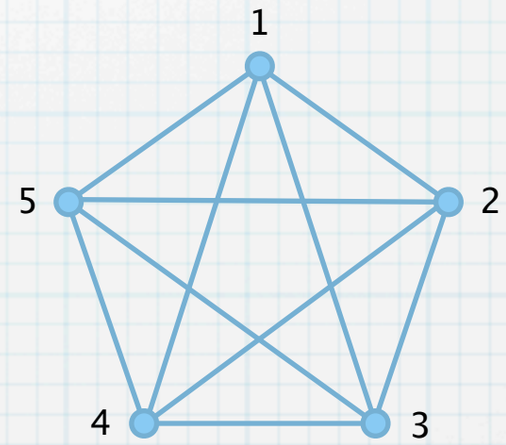
- Effizientes Speichern von großen Datenmengen
- Gegeben sei ein zwei-dimensionales Datenfeld (Array)
- Annahme: Unterschiede zwischen zwei Zeilen i, j nur an $c_{i,j}$ (wenigen) Stellen

1:	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**
2:	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
3:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**
4:	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**
5:	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**

Indirekte Anwendungen des Spannbaumproblems (Forts.)

- Effizientes Speichern von großen Datenmengen
- Gegeben sei ein zwei-dimensionales Datenfeld (Array)
- Annahme: Unterschiede zwischen zwei Zeilen i, j nur an $c_{i,j}$ (wenigen) Stellen
- Modell: Zeilen sind Knoten, Unterschiede sind Kantengewichte

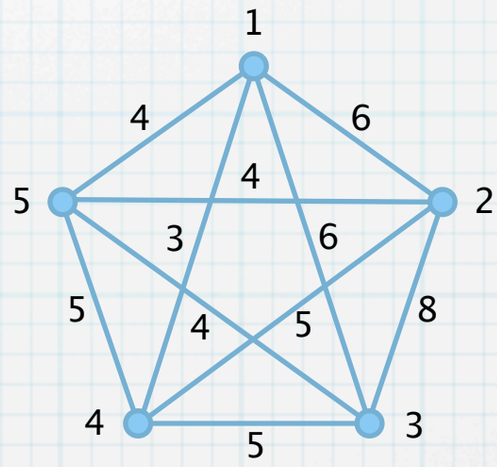
1:	0	0	1	1	1	1	0	1	1	0
2:	0	1	0	1	0	1	1	1	0	1
3:	1	0	1	0	1	0	1	0	1	1
4:	1	0	1	1	0	1	1	1	1	0
5:	0	1	1	1	1	1	1	0	1	1



Indirekte Anwendungen des Spannbaumproblems (Forts.)

- Effizientes Speichern von großen Datenmengen
- Gegeben sei ein zwei-dimensionales Datenfeld (Array)
- Annahme: Unterschiede zwischen zwei Zeilen i, j nur an $c_{i,j}$ (wenigen) Stellen
- Modell: Zeilen sind Knoten, Unterschiede sind Kantengewichte

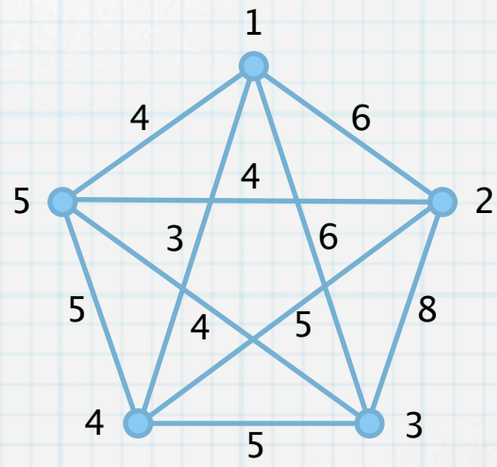
1:	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**
2:	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
3:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**
4:	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**
5:	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**



Indirekte Anwendungen des Spannbaumproblems (Forts.)

- Effizientes Speichern von großen Datenmengen
- Gegeben sei ein zwei-dimensionales Datenfeld (Array)
- Annahme: Unterschiede zwischen zwei Zeilen i, j nur an $c_{i,j}$ (wenigen) Stellen
- Modell: Zeilen sind Knoten, Unterschiede sind Kantengewichte
- Berechne minimalen Spannbaum

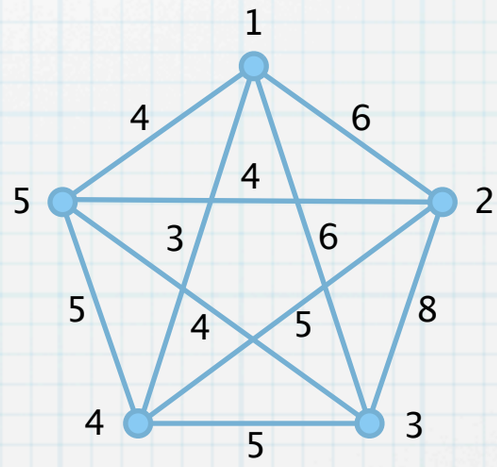
1:	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**
2:	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
3:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**
4:	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**
5:	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**



Indirekte Anwendungen des Spannbaumproblems (Forts.)

- Effizientes Speichern von großen Datenmengen
- Gegeben sei ein zwei-dimensionales Datenfeld (Array)
- Annahme: Unterschiede zwischen zwei Zeilen i, j nur an $c_{i,j}$ (wenigen) Stellen
- Modell: Zeilen sind Knoten, Unterschiede sind Kantengewichte
- Berechne minimalen Spannbaum
- Speichere eine Referenzzeile vollständig, und nur die Unterschiede zwischen den Zeilenpaaren des Spannbaums

1:	**0**	**0**	**1**	**1**	**1**	**1**	**0**	**1**	**1**	**0**
2:	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**1**
3:	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**0**	**1**	**1**
4:	**1**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**1**	**0**
5:	**0**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**1**	**1**



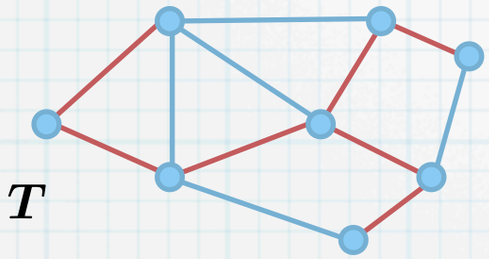
Zwei Kriterien für optimale Spannbäume

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.

Zwei Kriterien für optimale Spannbäume

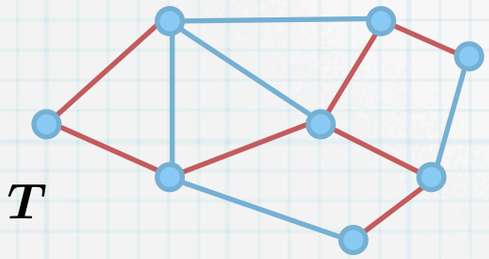
* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.



T

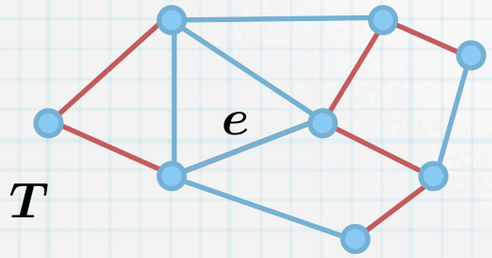
Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
  Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.



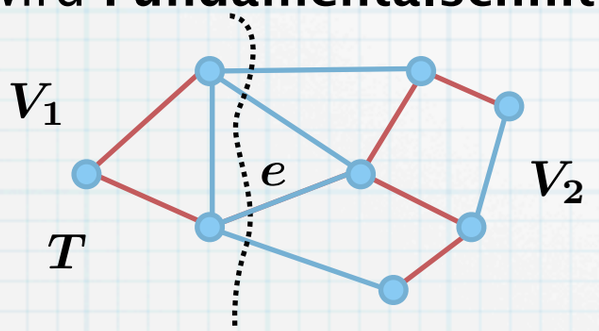
Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
  Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.



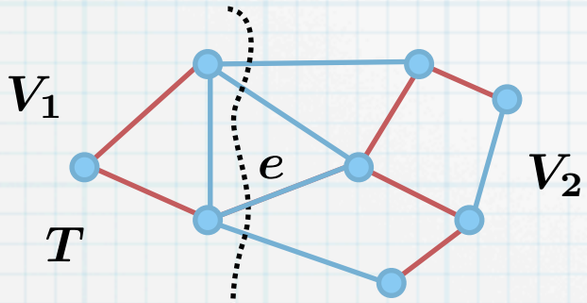
Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
  Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.



Zwei Kriterien für optimale Spannbäume

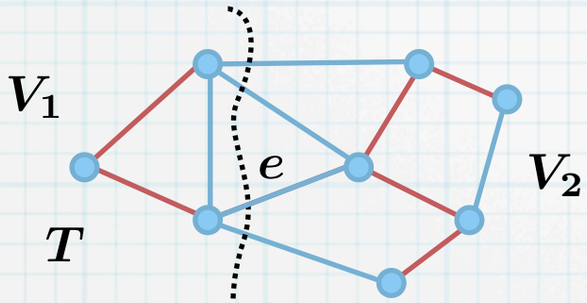
* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

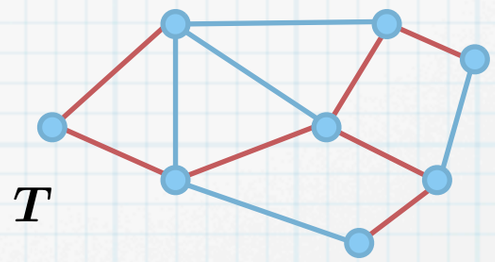


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

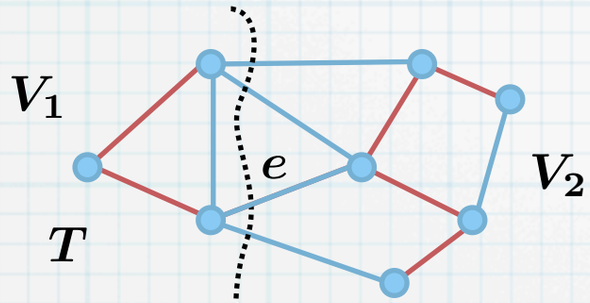


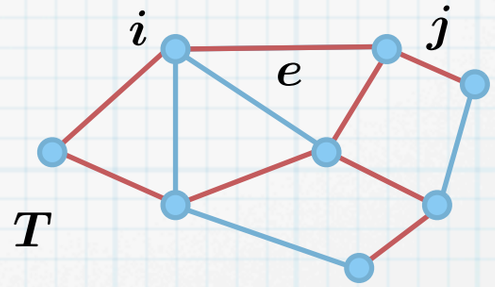


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

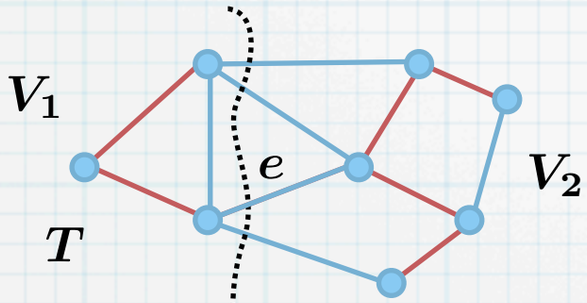


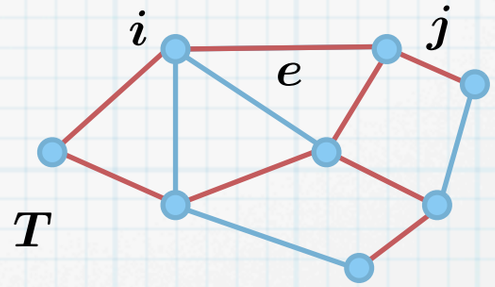


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

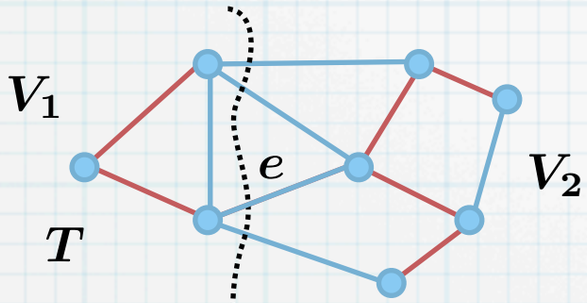


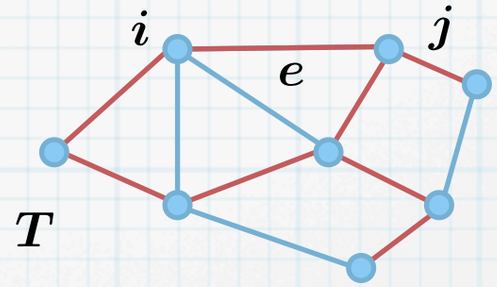


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

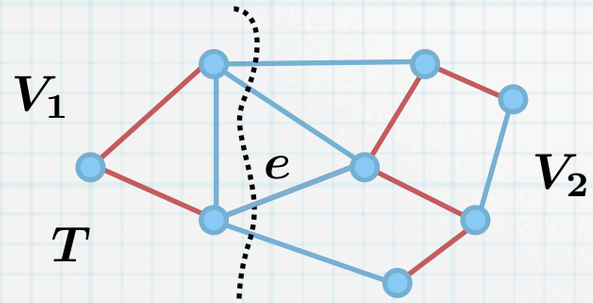


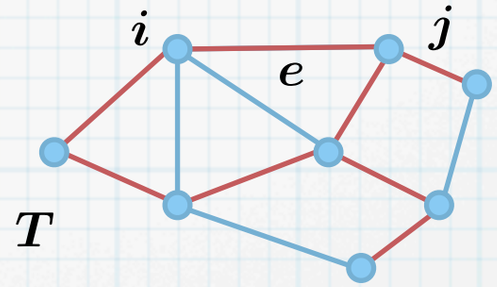


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:
  (i) T ist ein minimaler Spannbaum.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

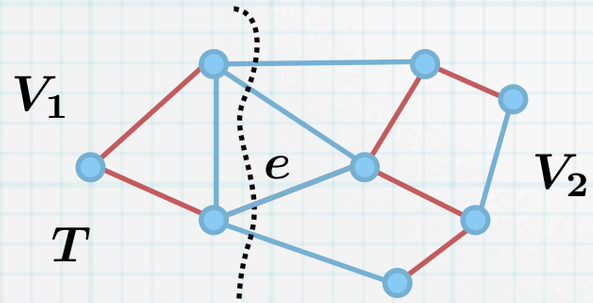


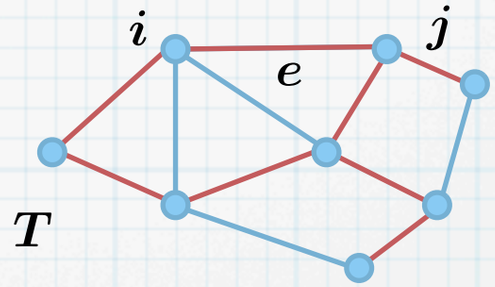


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:
  (i) T ist ein minimaler Spannbaum.
  (ii) (Schnitt-Optimalitätsbedingung) Für jede Baumkante $e \in E(T)$ gilt: e ist die „billigste“ Kante im Fundamentalschitt von e, d.h. für alle Kanten e' im Schnitt ist $w_e \leq w_{e'}$.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

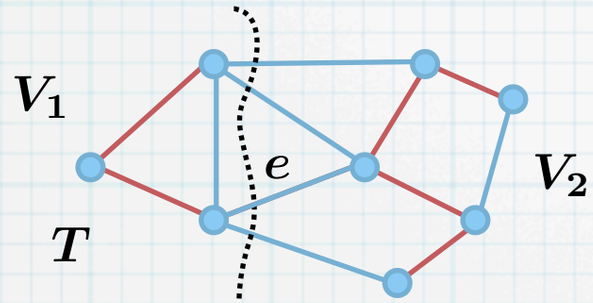


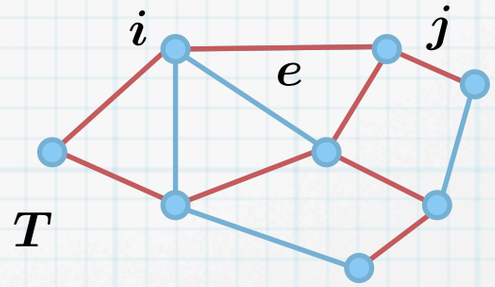


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:
  (i) T ist ein minimaler Spannbaum.
  (ii) (Schnitt-Optimalitätsbedingung) Für jede Baumkante $e \in E(T)$ gilt: e ist die „billigste“ Kante im Fundamentalschitt von e, d.h. für alle Kanten e' im Schnitt ist $w_e \leq w_{e'}$.
  (iii) (Pfad-Optimalitätsbedingung) Für jede Nicht-Baumkante $e \in E \backslash E(T)$ gilt: e ist die „teuerste“ Kante im Fundamentalkreis von e, d.h. für alle Kanten e' im Kreis ist $w_{e'} \leq w_e$.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

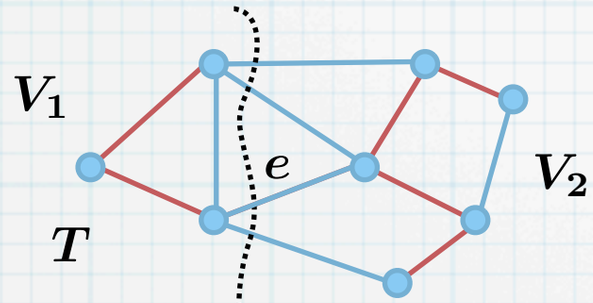


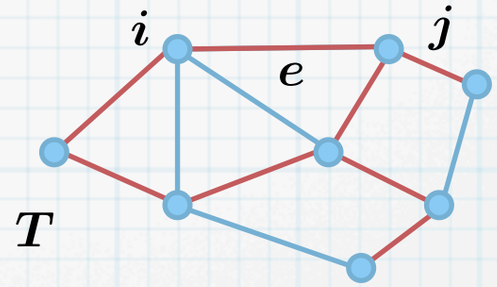


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:
  (i) T ist ein minimaler Spannbaum.
  (ii) (Schnitt-Optimalitätsbedingung) Für jede Baumkante $e \in E(T)$ gilt: e ist die „billigste“ Kante im Fundamentalschitt von e, d.h. für alle Kanten e' im Schnitt ist $w_e \leq w_{e'}$.
  (iii) (Pfad-Optimalitätsbedingung) Für jede Nicht-Baumkante $e \in E \backslash E(T)$ gilt: e ist die „teuerste“ Kante im Fundamentalkreis von e, d.h. für alle Kanten e' im Kreis ist $w_{e'} \leq w_e$.
* Bemerkungen:

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

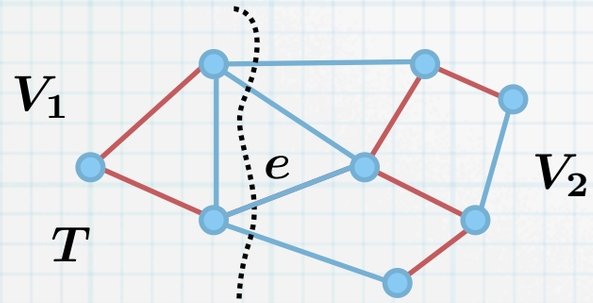


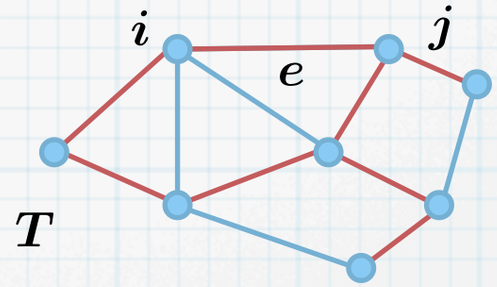


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:
  (i) T ist ein minimaler Spannbaum.
  (ii) (Schnitt-Optimalitätsbedingung) Für jede Baumkante $e \in E(T)$ gilt: e ist die „billigste“ Kante im Fundamentalschitt von e, d.h. für alle Kanten e' im Schnitt ist $w_e \leq w_{e'}$.
  (iii) (Pfad-Optimalitätsbedingung) Für jede Nicht-Baumkante $e \in E \backslash E(T)$ gilt: e ist die „teuerste“ Kante im Fundamentalkreis von e, d.h. für alle Kanten e' im Kreis ist $w_{e'} \leq w_e$.
* Bemerkungen:
  * Schnitt-Optimalitätskriterium ist eine „externe“ Charakterisierung.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.

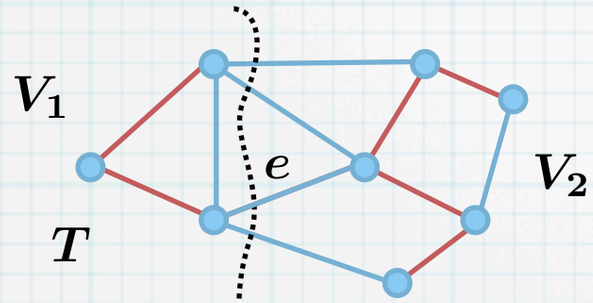


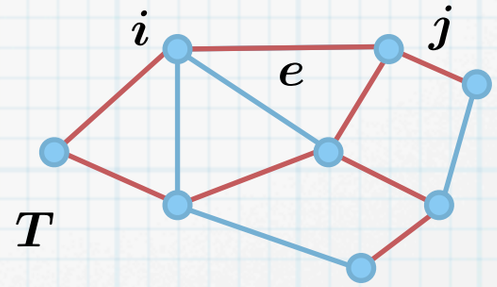


  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:
  (i) T ist ein minimaler Spannbaum.
  (ii) (Schnitt-Optimalitätsbedingung) Für jede Baumkante $e \in E(T)$ gilt: e ist die „billigste“ Kante im Fundamentalschitt von e, d.h. für alle Kanten e' im Schnitt ist $w_e \leq w_{e'}$.
  (iii) (Pfad-Optimalitätsbedingung) Für jede Nicht-Baumkante $e \in E \backslash E(T)$ gilt: e ist die „teuerste“ Kante im Fundamentalkreis von e, d.h. für alle Kanten e' im Kreis ist $w_{e'} \leq w_e$.
* Bemerkungen:
  * Schnitt-Optimalitätskriterium ist eine „externe“ Charakterisierung.
  * Pfad-Optimalitätskriterium ist eine „interne“ Charakterisierung.

Zwei Kriterien für optimale Spannbäume

* **Definition 17**:
Gegeben sein ein aufspannender Baum T eines Graphen $G = (V, E)$.
  * Löscht man eine Kante e des Baums, so zerfällt die Knotenmenge V in zwei disjunkte Teilmengen V_1, V_2. Die Teilmenge von E, die zu je einem Knoten in V_1 und V_2 inzident sind, wird **Fundamentalschnitt** von e genannt.





  * Fügt man eine Nicht-Baumkante $e \in E \backslash E(T)$ dem Baum hinzu, so definiert $e = \{i, j\}$ mit dem eindeutigen Pfad zwischen i und j im Baum einen Kreis, den **Fundamentalkreis** von e.
* **Satz 18** (Optimalitätskriterien für aufspannende Bäume):
Für einen aufspannenden Baum T eines gewichteten Graphen $G = (V, E, w)$ sind äquivalent:
  (i) T ist ein minimaler Spannbaum.
  (ii) (Schnitt-Optimalitätsbedingung) Für jede Baumkante $e \in E(T)$ gilt: e ist die „billigste“ Kante im Fundamentalschitt von e, d.h. für alle Kanten e' im Schnitt ist $w_e \leq w_{e'}$.
  (iii) (Pfad-Optimalitätsbedingung) Für jede Nicht-Baumkante $e \in E \backslash E(T)$ gilt: e ist die „teuerste“ Kante im Fundamentalkreis von e, d.h. für alle Kanten e' im Kreis ist $w_{e'} \leq w_e$.
* Bemerkungen:
  * Schnitt-Optimalitätskriterium ist eine „externe“ Charakterisierung.
  * Pfad-Optimalitätskriterium ist eine „interne“ Charakterisierung.
  * Beide Kriterien gelten analog für maximale Spannbäume (ersetze $\leq$ durch $\geq$ in Satz 18).

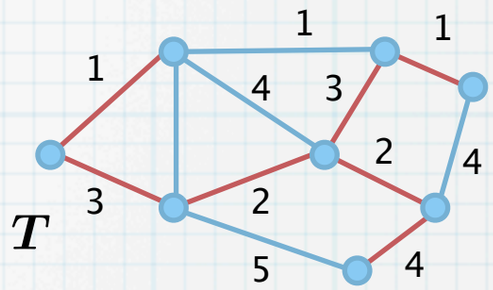
Beweis des Schnitt-Optimalitäts-Kriteriums

Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

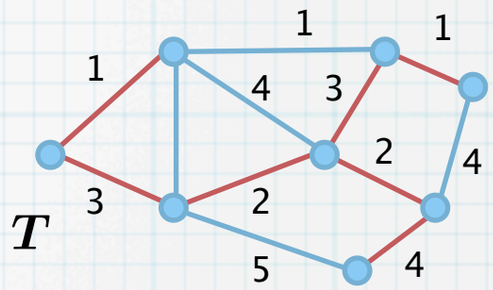
Beweis des Schnitt-Optimalitäts-Kriteriums

* (⇒) Sei T ein minimaler aufspannender Baum.



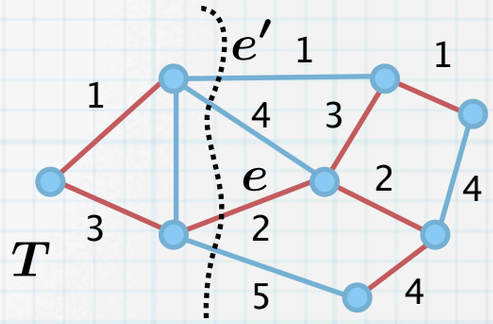
Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.



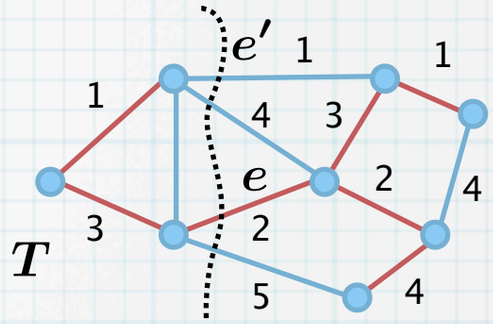
Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.



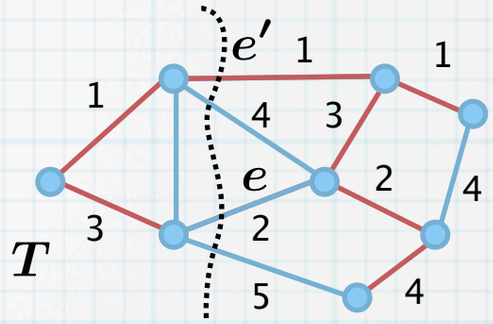
Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.



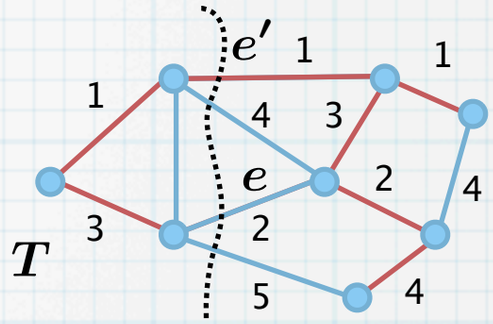
Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.



Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.



Beweis des Schnitt-Optimalitäts-Kriteriums

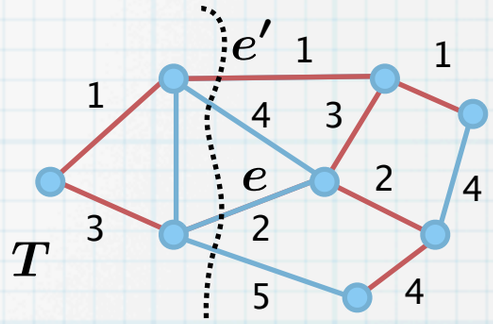
* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.



Beweis des Schnitt-Optimalitäts-Kriteriums

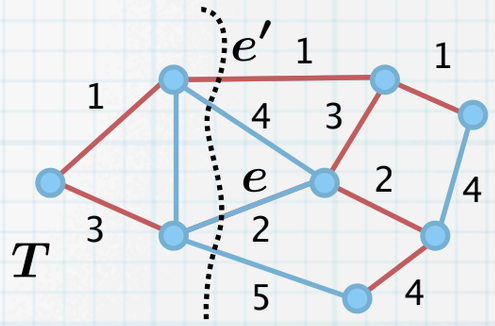
* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

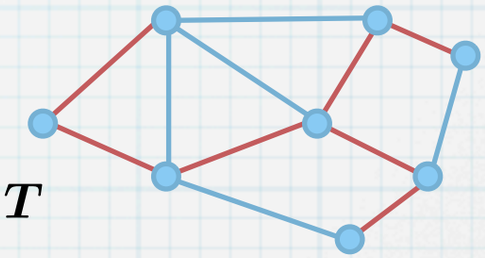
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.





Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

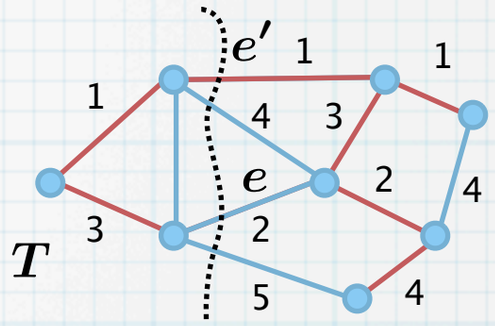
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

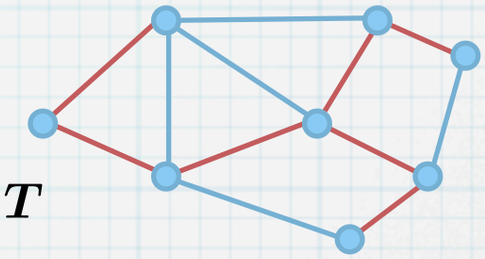
  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.

  Sei T' ein minimaler Spannbaum mit $T' \neq T$.





Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

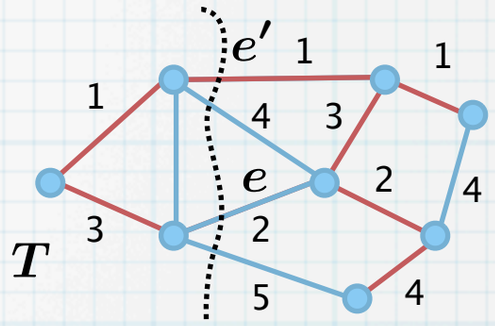
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

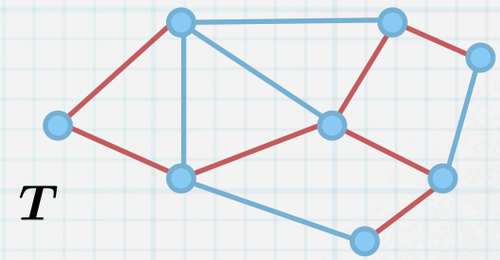
  Dann lösche Kante e und füge e' hinzu.

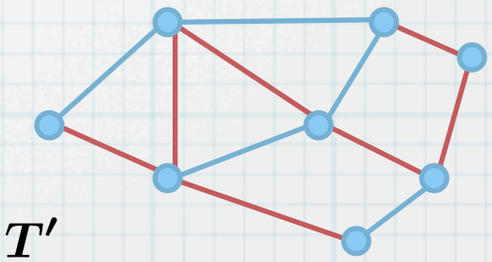
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.

  Sei T' ein minimaler Spannbaum mit $T' \neq T$.







Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

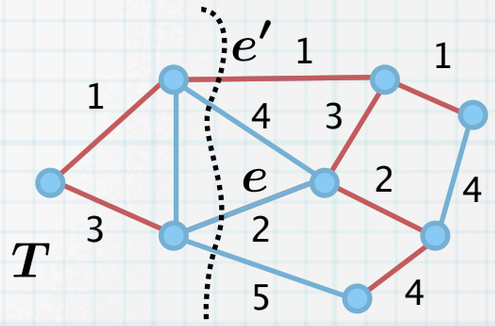
  Dann lösche Kante e und füge e' hinzu.

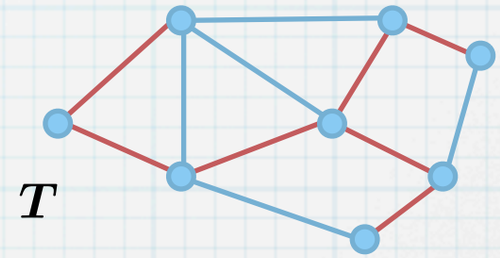
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

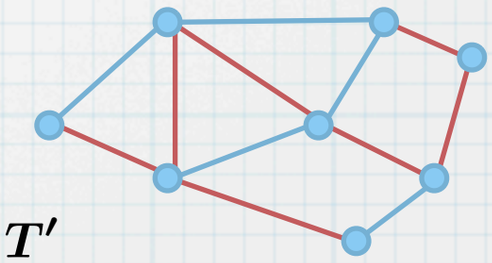
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.

  Sei T' ein minimaler Spannbaum mit $T' \neq T$.

  Dann gibt es eine Kante $e \in E(T) \backslash E(T')$.







Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

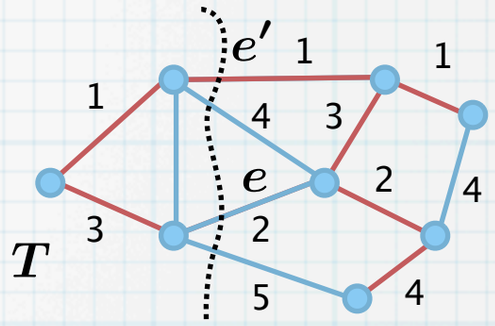
  Dann lösche Kante e und füge e' hinzu.

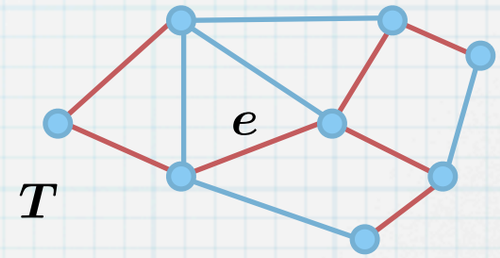
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

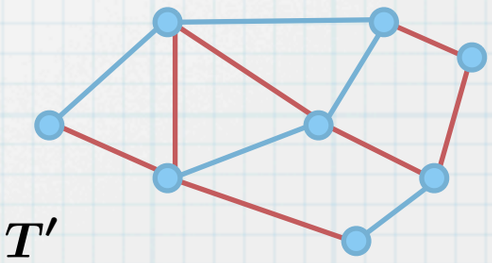
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.

  Sei T' ein minimaler Spannbaum mit $T' \neq T$.

  Dann gibt es eine Kante $e \in E(T) \backslash E(T')$.







Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

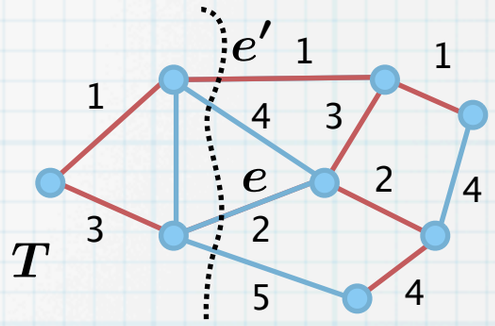
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

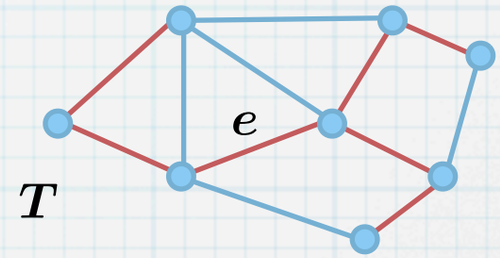
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.

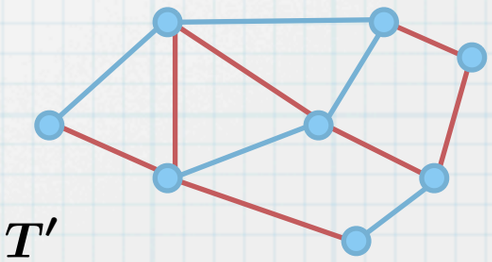
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.

  Dann gibt es eine Kante $e \in E(T) \backslash E(T')$.

  Fügt man e zu T' hinzu, erhält man einen Kreis.







Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

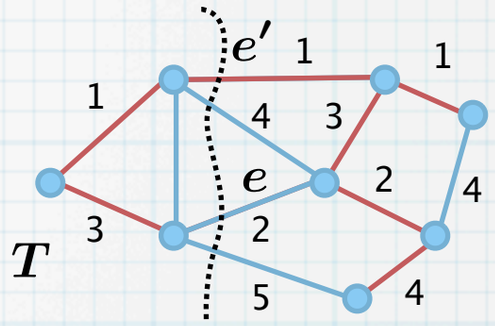
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

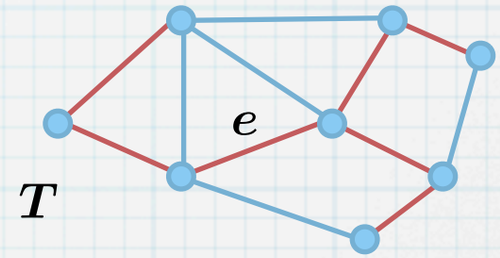
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.

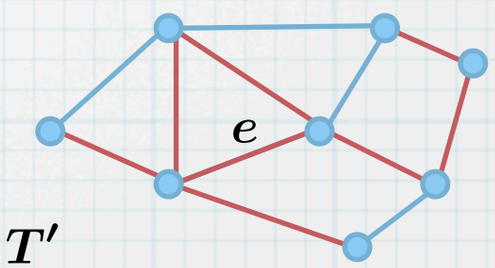
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.

  Dann gibt es eine Kante $e \in E(T) \backslash E(T')$.

  Fügt man e zu T' hinzu, erhält man einen Kreis.

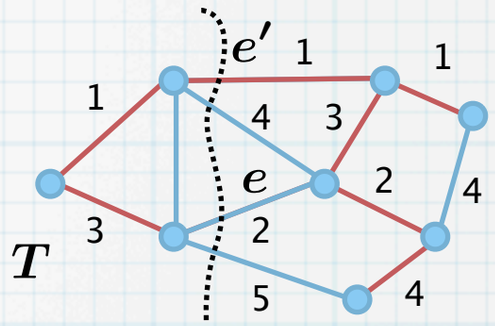


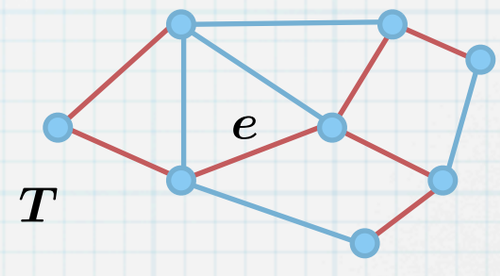


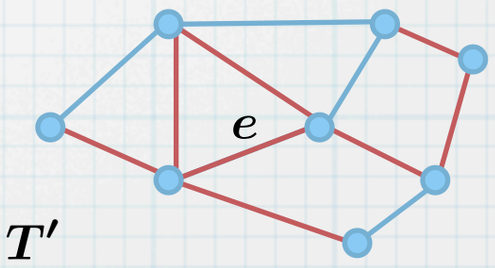


Beweis des Schnitt-Optimalitäts-Kriteriums

* (⇒) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* (⇐) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \setminus E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.

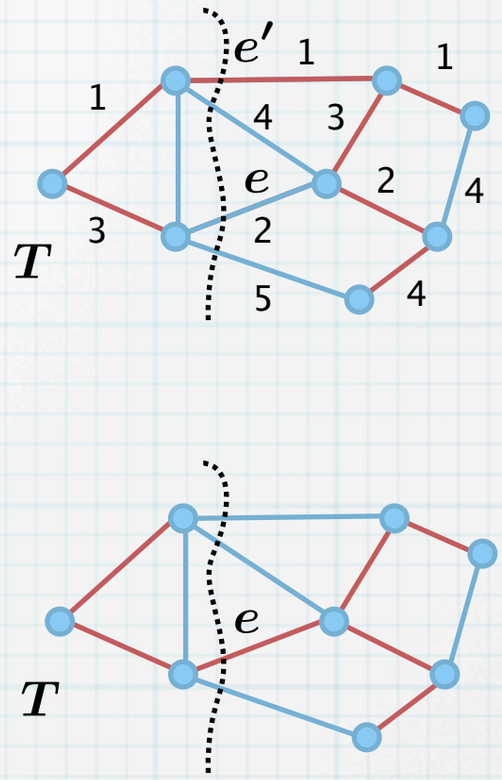


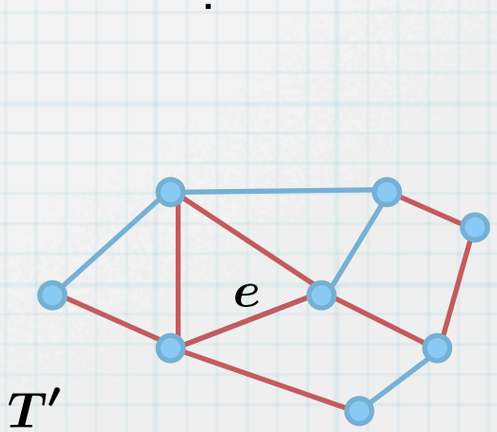




Beweis des Schnitt-Optimalitäts-Kriteriums

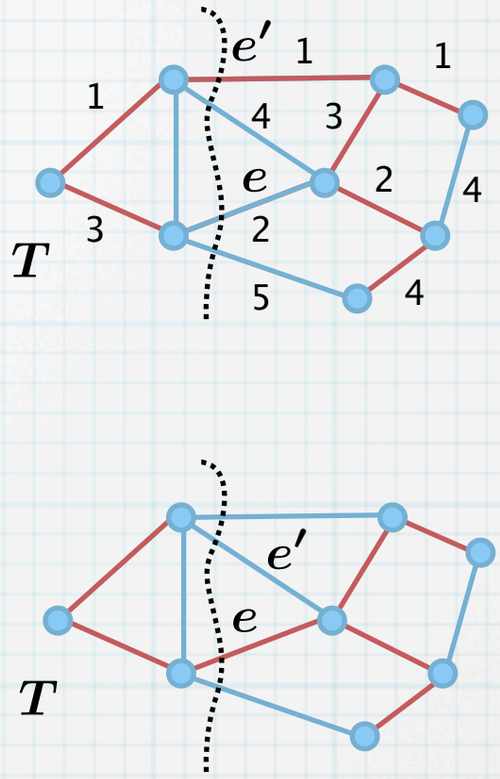
* (⇒) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* (⇐) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \setminus E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.

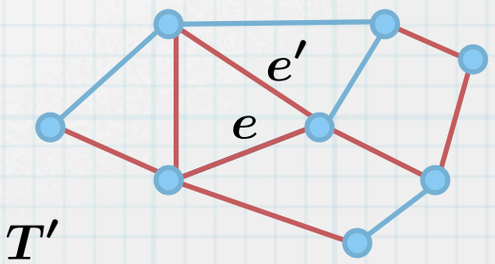




Beweis des Schnitt-Optimalitäts-Kriteriums

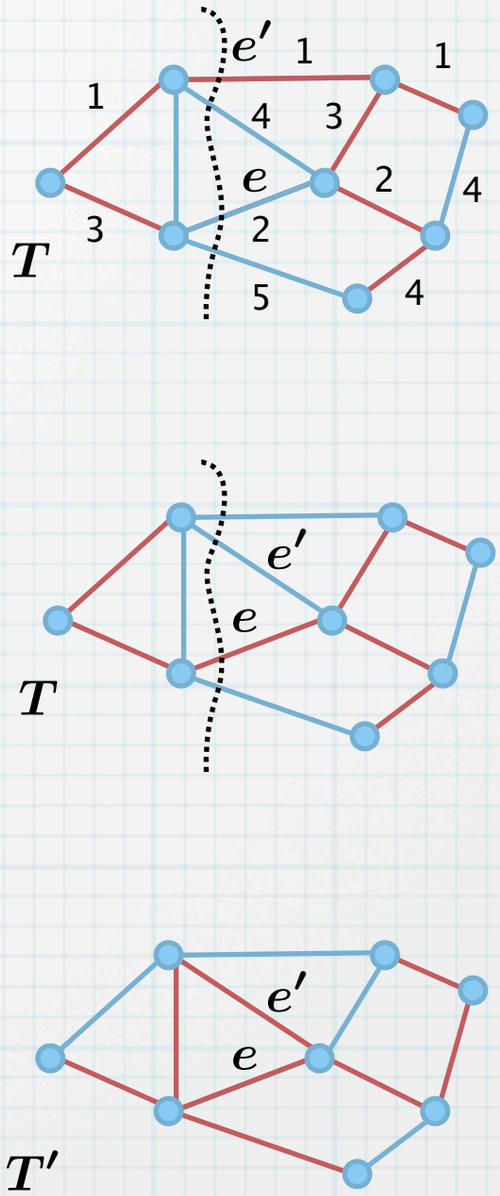
* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \backslash E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.





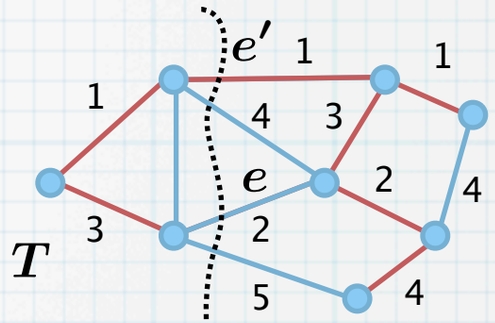
Beweis des Schnitt-Optimalitäts-Kriteriums

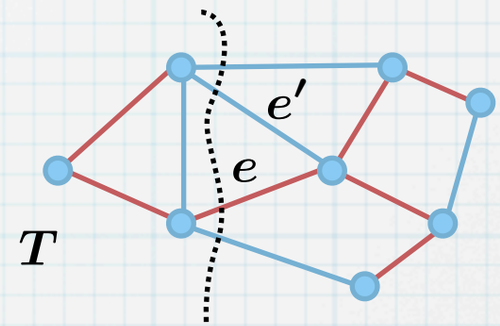
* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \backslash E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.
  Aus der Schnittbedingung in T folgt $w_e \leq w_{e'}$.

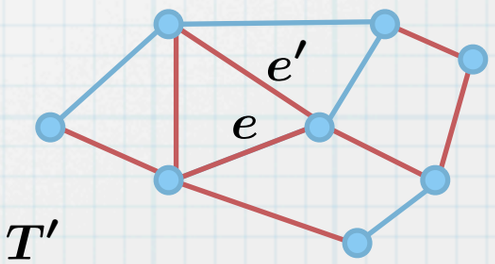


Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \setminus E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.
  Aus der Schnittbedingung in T folgt $w_e \leq w_{e'}$.
  Da T' minimal ist, gilt $w_e \geq w_{e'}$ (sonst könnte man e' durch e ersetzen, und würde einen „minimaleren“ Spannbaum erhalten).

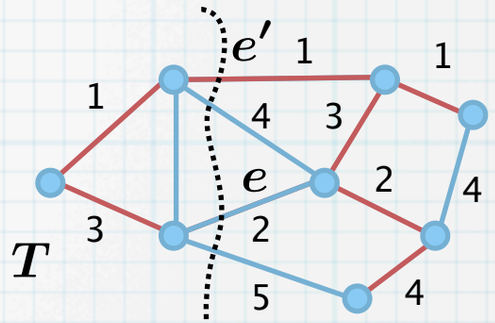


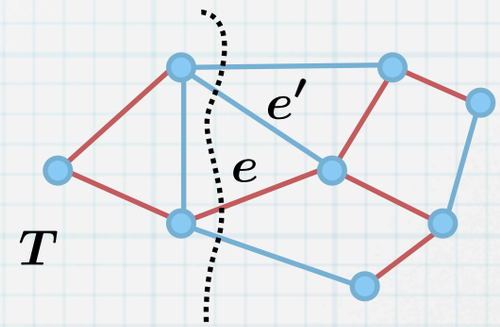


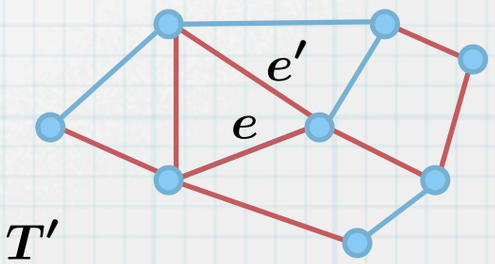


Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \setminus E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.
  Aus der Schnittbedingung in T folgt $w_e \leq w_{e'}$.
  Da T' minimal ist, gilt $w_e \geq w_{e'}$ (sonst könnte man e' durch e ersetzen, und würde einen „minimaleren“ Spannbaum erhalten).
  Zusammen gilt also $w_e = w_{e'}$.

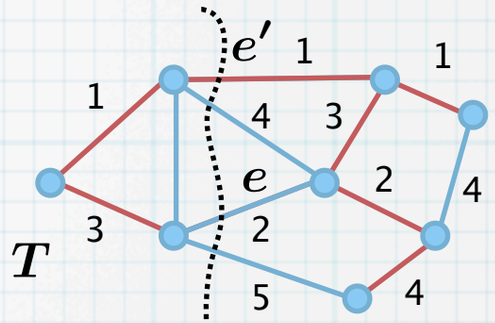


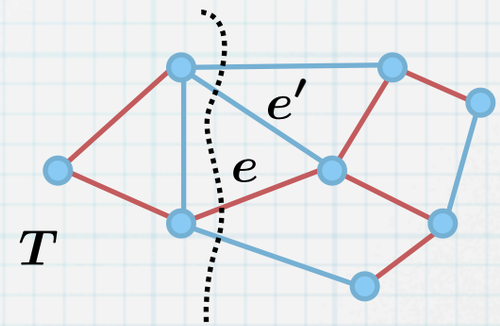


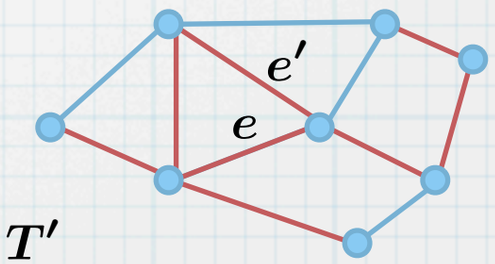


Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \setminus E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.
  Aus der Schnittbedingung in T folgt $w_e \leq w_{e'}$.
  Da T' minimal ist, gilt $w_e \geq w_{e'}$ (sonst könnte man e' durch e ersetzen, und würde einen „minimaleren“ Spannbaum erhalten).
  Zusammen gilt also $w_e = w_{e'}$.
  Ersetze nun e' durch e in T', so ist auch T' minimaler Spannbaum, hat aber eine Kante mehr mit T gemein.

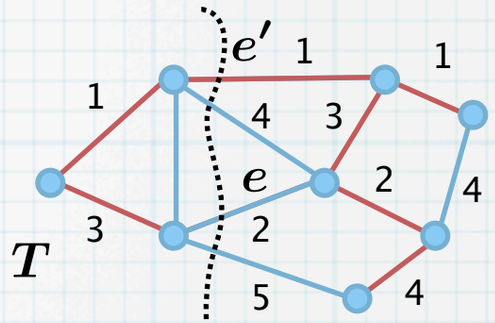


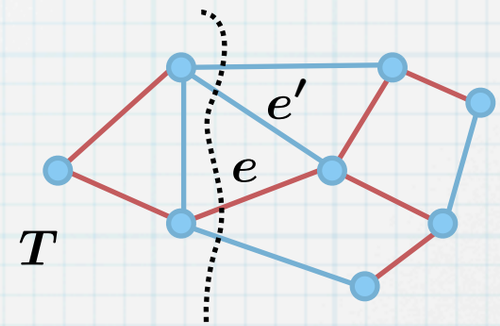


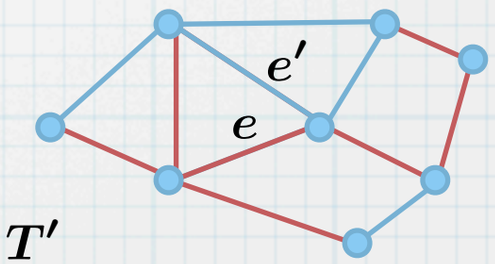


Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \setminus E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.
  Aus der Schnittbedingung in T folgt $w_e \leq w_{e'}$.
  Da T' minimal ist, gilt $w_e \geq w_{e'}$ (sonst könnte man e' durch e ersetzen, und würde einen „minimaleren“ Spannbaum erhalten).
  Zusammen gilt also $w_e = w_{e'}$.
  Ersetze nun e' durch e in T', so ist auch T' minimaler Spannbaum, hat aber eine Kante mehr mit T gemein.

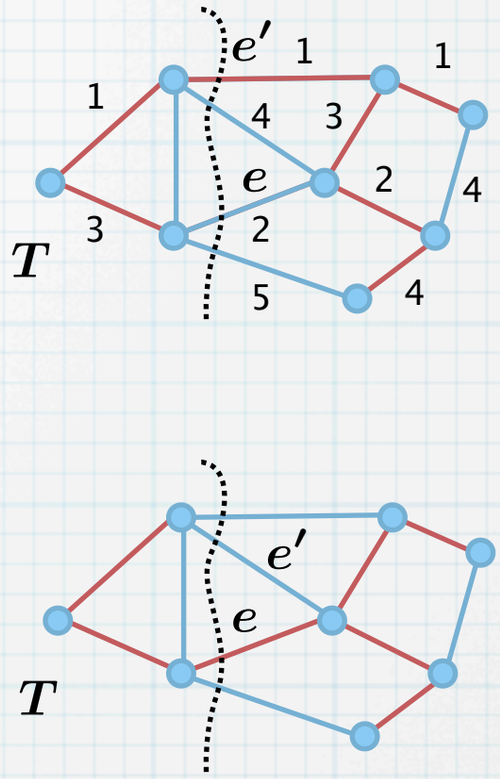


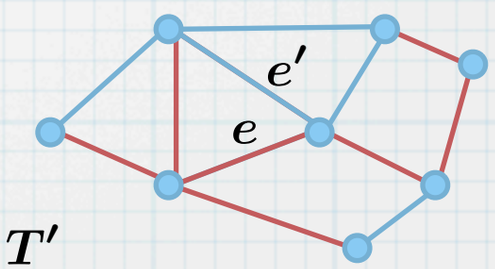




Beweis des Schnitt-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e \in E(T)$, deren Fundamentalschnitt eine Kante e' enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.
* ($\Leftarrow$) Sei T ein Spannbaum mit Schnittbedingung.
  Sei T' ein minimaler Spannbaum mit $T' \neq T$.
  Dann gibt es eine Kante $e \in E(T) \backslash E(T')$.
  Fügt man e zu T' hinzu, erhält man einen Kreis.
  Der Kreis enthält eine Kante $e' \in E(T'), e \neq e'$, die zudem im Fundamentalschnitt von e in T liegt.
  Aus der Schnittbedingung in T folgt $w_e \leq w_{e'}$.
  Da T' minimal ist, gilt $w_e \geq w_{e'}$ (sonst könnte man e' durch e ersetzen, und würde einen „minimaleren“ Spannbaum erhalten).
  Zusammen gilt also $w_e = w_{e'}$.
  Ersetze nun e' durch e in T', so ist auch T' minimaler Spannbaum, hat aber eine Kante mehr mit T gemein.
  Diese Argumentation kann nun so lange wiederholt werden, bis T' vollständig in T umgewandelt wurde.
  Also ist auch T ein minimaler Spannbaum.





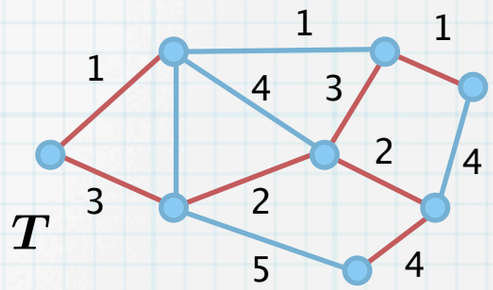
Beweis des Pfad-Optimalitäts-Kriteriums

Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

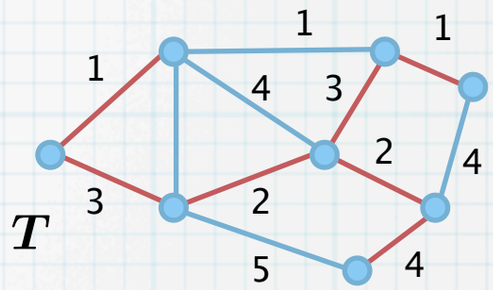
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.



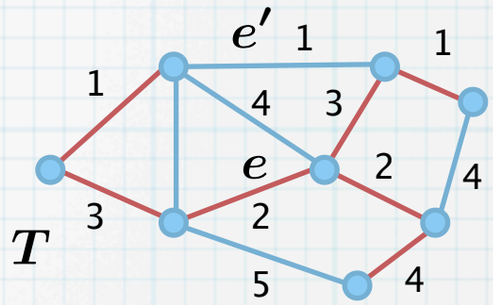
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.



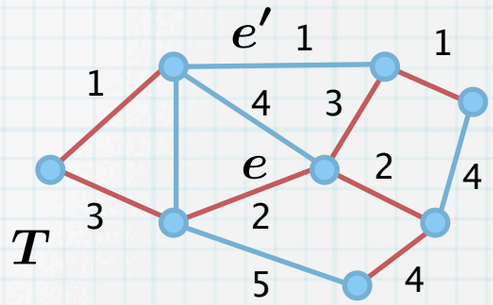
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.



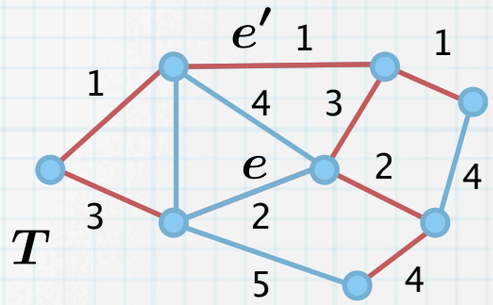
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.



Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.
  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.
  Dann lösche Kante e und füge e' hinzu.



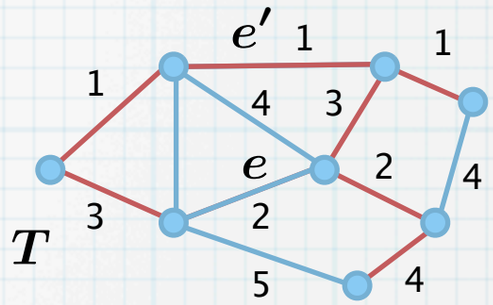
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.



Beweis des Pfad-Optimalitäts-Kriteriums

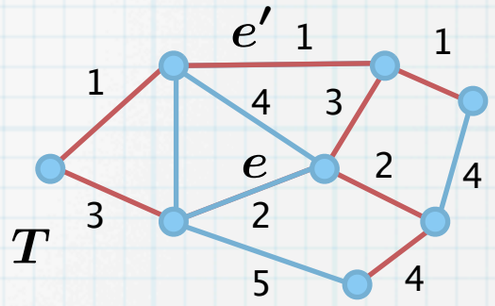
* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

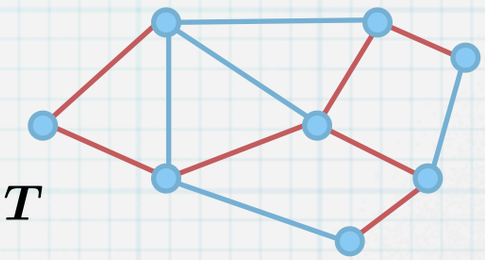
  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.





Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

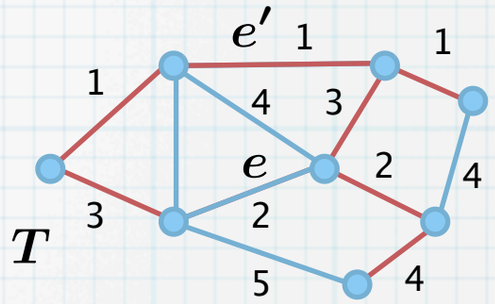
  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

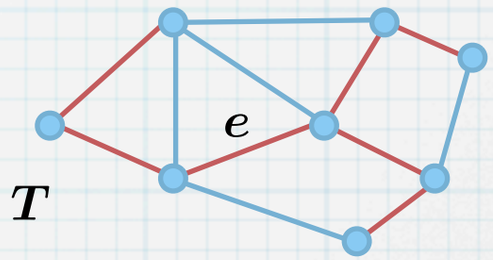
  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.

  Sei $e \in E(T)$.





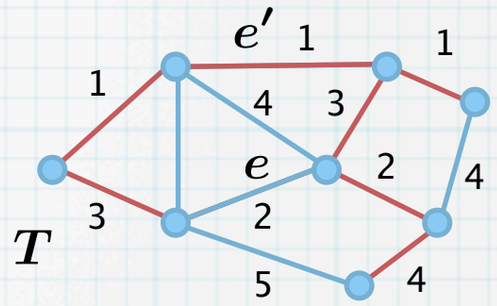
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

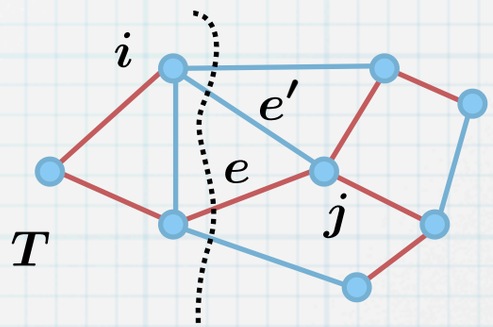
  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.



* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.

  Sei $e \in E(T)$.

  Wähle irgendeine Kante $e' = \{i, j\}$ im Fundamentalschnitt von e.



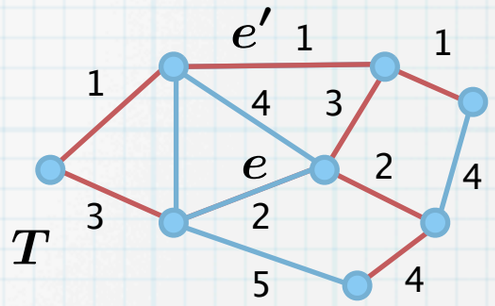
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.

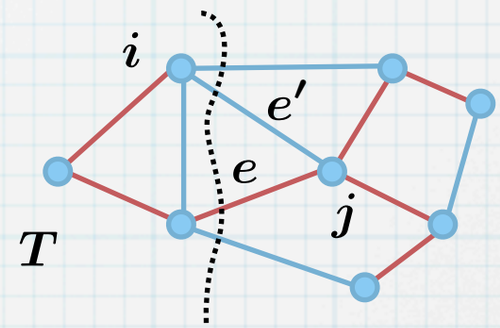


* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.

  Sei $e \in E(T)$.

  Wähle irgendeine Kante $e' = \{i, j\}$ im Fundamentalschnitt von e.

  In T gibt es einen eindeutigen Pfad zwischen i und j.



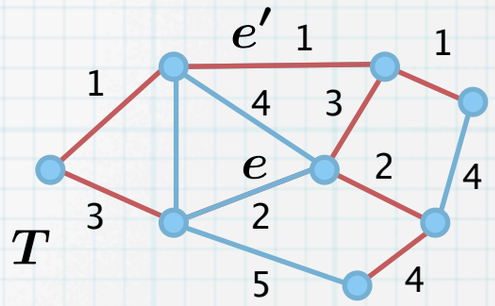
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.



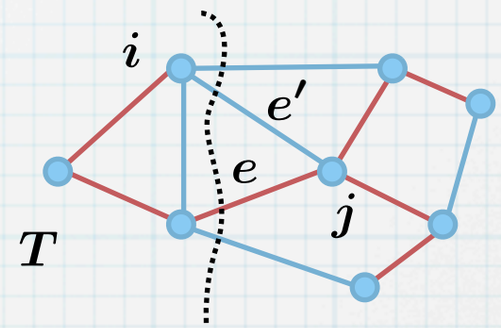
* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.

  Sei $e \in E(T)$.

  Wähle irgendeine Kante $e' = \{i, j\}$ im Fundamentalschnitt von e.

  In T gibt es einen eindeutigen Pfad zwischen i und j.

  Kante e gehört zu diesem Pfad, da nur e sowohl zum Fundamentalschnitt als auch zum Baum T gehört.



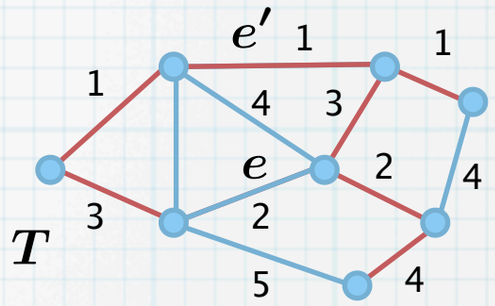
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.



* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.

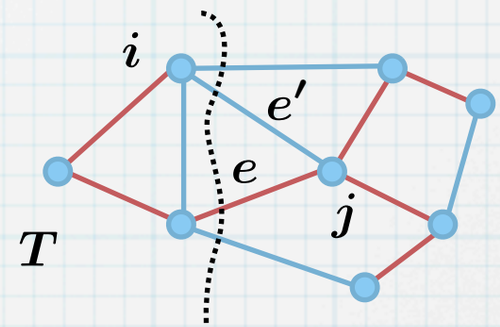
  Sei $e \in E(T)$.

  Wähle irgendeine Kante $e' = \{i, j\}$ im Fundamentalschnitt von e.

  In T gibt es einen eindeutigen Pfad zwischen i und j.

  Kante e gehört zu diesem Pfad, da nur e sowohl zum Fundamentalschnitt als auch zum Baum T gehört.

  Aus der Pfadbedingung folgt $w_e \leq w_{e'}$.



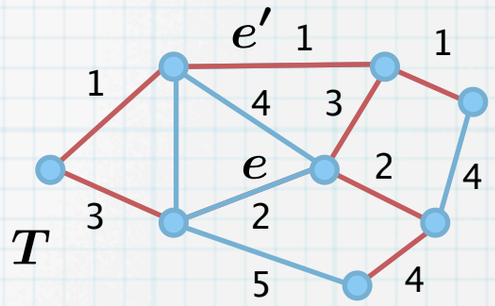
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.



* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.

  Sei $e \in E(T)$.

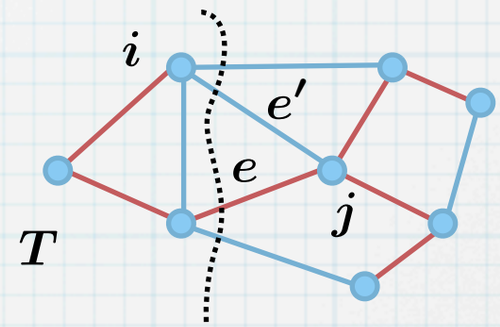
  Wähle irgendeine Kante $e' = \{i, j\}$ im Fundamentalschnitt von e.

  In T gibt es einen eindeutigen Pfad zwischen i und j.

  Kante e gehört zu diesem Pfad, da nur e sowohl zum Fundamentalschnitt als auch zum Baum T gehört.

  Aus der Pfadbedingung folgt $w_e \leq w_{e'}$.

  Da dieses für jede Kante e' im Fundamentalschnitt von e gilt, erfüllt T somit auch die Schnittbedingung.



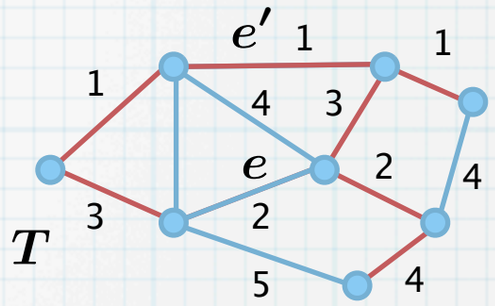
Beweis des Pfad-Optimalitäts-Kriteriums

* ($\Rightarrow$) Sei T ein minimaler aufspannender Baum.

  Angenommen, es gibt eine Kante $e' \notin E(T)$, deren Fundamentalkreis eine Kante $e \in E(T)$ enthält mit $w_e > w_{e'}$.

  Dann lösche Kante e und füge e' hinzu.

  Dieses führt zu einem neuen Spannbaum mit geringem Gewicht als T, im Widerspruch zur Optimalität von T.



* ($\Leftarrow$) Sei T ein Spannbaum mit Pfadbedingung.

  Sei $e \in E(T)$.

  Wähle irgendeine Kante $e' = \{i, j\}$ im Fundamentalschnitt von e.

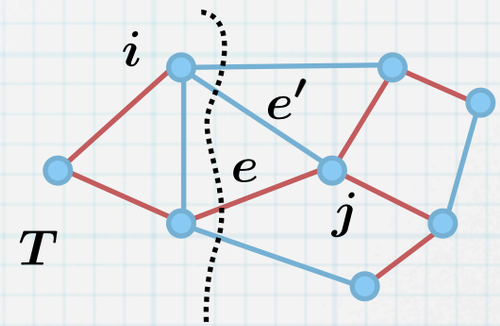
  In T gibt es einen eindeutigen Pfad zwischen i und j.

  Kante e gehört zu diesem Pfad, da nur e sowohl zum Fundamentalschnitt als auch zum Baum T gehört.

  Aus der Pfadbedingung folgt $w_e \leq w_{e'}$.

  Da dieses für jede Kante e' im Fundamentalschnitt von e gilt, erfüllt T somit auch die Schnittbedingung.

  Also ist T ein minimaler Spannbaum.



Folgerung aus der Schnittbedingung

Folgerung aus der Schnittbedingung

* **Lemma 19**:
Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
  Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

Folgerung aus der Schnittbedingung

* **Lemma 19**:
  Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
  Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
  Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

Folgerung aus der Schnittbedingung

* **Lemma 19**:
  Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

  2. Fall, $e \notin T$.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
  Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

  2. Fall, $e \notin T$.

     Da T ein aufspannender Baum ist, enthält $E(T) \cup \{e\}$ einen Kreis.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

  2. Fall, $e \notin T$.

     Da T ein aufspannender Baum ist, enthält $E(T) \cup \{e\}$ einen Kreis.
     Dieser Kreis enthält eine weitere Kante $e' = \{u', v'\}$ mit $e' \neq e, u' \in U, v' \notin U$.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

  2. Fall, $e \notin T$.

     Da T ein aufspannender Baum ist, enthält $E(T) \cup \{e\}$ einen Kreis.
     Dieser Kreis enthält eine weitere Kante $e' = \{u', v'\}$ mit $e' \neq e, u' \in U, v' \notin U$.
     Nach Voraussetzung ist einerseits $w_e \leq w_{e'}$.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

  2. Fall, $e \notin T$.

     Da T ein aufspannender Baum ist, enthält $E(T) \cup \{e\}$ einen Kreis.
     Dieser Kreis enthält eine weitere Kante $e' = \{u', v'\}$ mit $e' \neq e, u' \in U, v' \notin U$.
     Nach Voraussetzung ist einerseits $w_e \leq w_{e'}$.
     Wegen Schnittbedingung ist e' billigste Kante im Fundamentalschnitt, d.h. $w_e \geq w_{e'}$.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

  2. Fall, $e \notin T$.

     Da T ein aufspannender Baum ist, enthält $E(T) \cup \{e\}$ einen Kreis.
     Dieser Kreis enthält eine weitere Kante $e' = \{u', v'\}$ mit $e' \neq e, u' \in U, v' \notin U$.
     Nach Voraussetzung ist einerseits $w_e \leq w_{e'}$.
     Wegen Schnittbedingung ist e' billigste Kante im Fundamentalschnitt, d.h. $w_e \geq w_{e'}$.
     Zusammen also $w_e = w_{e'}$.

Folgerung aus der Schnittbedingung

* **Lemma 19**:
Sei $G = (V, E, w)$ ein gewichteter Graph. Sei U Untergraphen eines minimalen Spannbaums T von G. Sei $e = \{u, v\} \in E, u \in U, v \notin U$ eine Kante mit $w_e \leq w_{e'}$ für alle $e' = \{u', v'\}$ mit $u' \in U, v' \notin U$. Dann gibt es einen minimalen Spannbaum T' von G, der U und e enthält.
* Beweis:

  Betrachte eine Kante $e = \{u, v\} \in E$ mit $u \in U, v \notin U$ und $w_e \leq w_{e'}$.

  1. Fall, $e \in T$.

     Dann ist nichts zu zeigen (bzw. setze $T' := T$).

  2. Fall, $e \notin T$.

     Da T ein aufspannender Baum ist, enthält $E(T) \cup \{e\}$ einen Kreis.
     Dieser Kreis enthält eine weitere Kante $e' = \{u', v'\}$ mit $e' \neq e, u' \in U, v' \notin U$.
     Nach Voraussetzung ist einerseits $w_e \leq w_{e'}$.
     Wegen Schnittbedingung ist e' billigste Kante im Fundamentalschnitt, d.h. $w_e \geq w_{e'}$.
     Zusammen also $w_e = w_{e'}$.
     Somit ist auch T' mit $E(T') := (E(T) \backslash \{e\}) \cup \{e'\}$ ein minimaler Spannbaum.

Der Algorithmus von Jarnik (Prim)

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

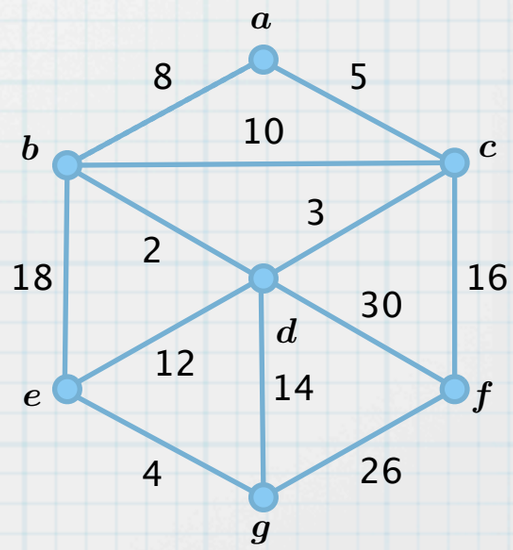
* Beispiel:

Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

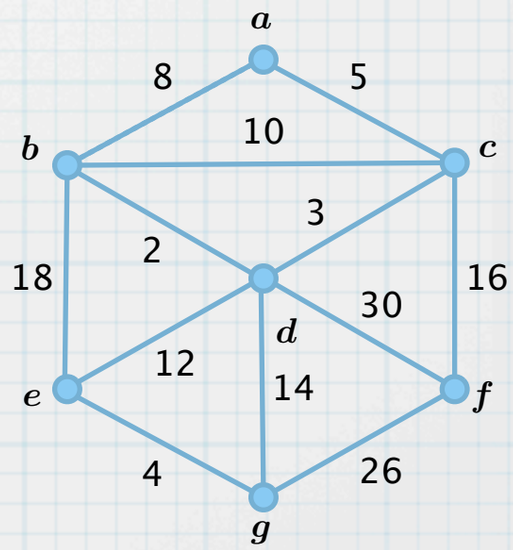


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

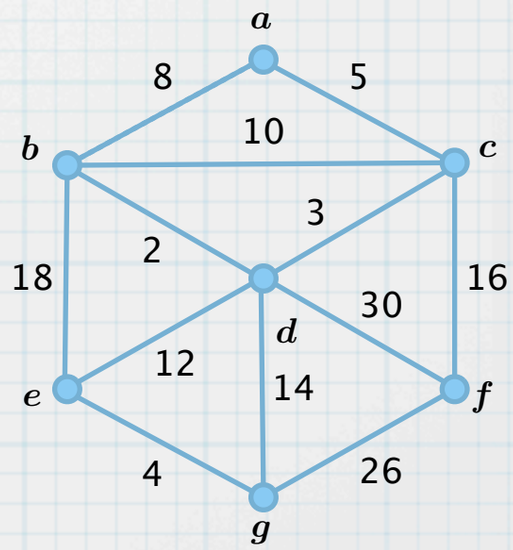


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

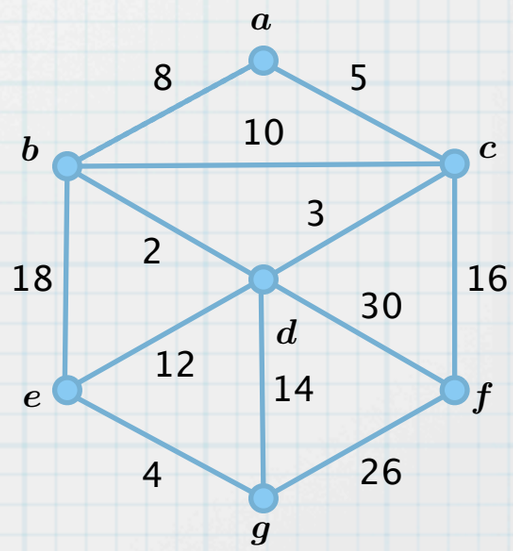


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

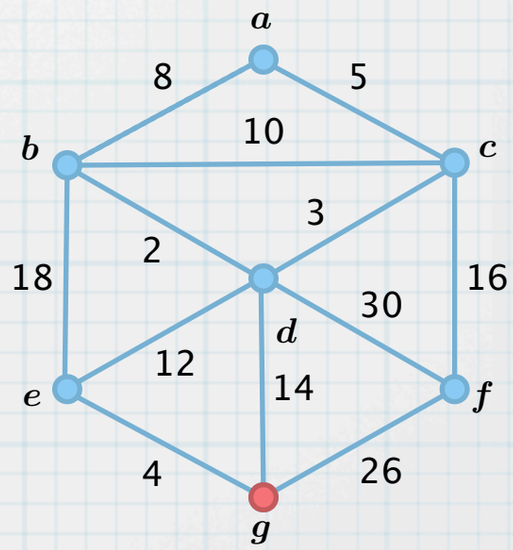


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

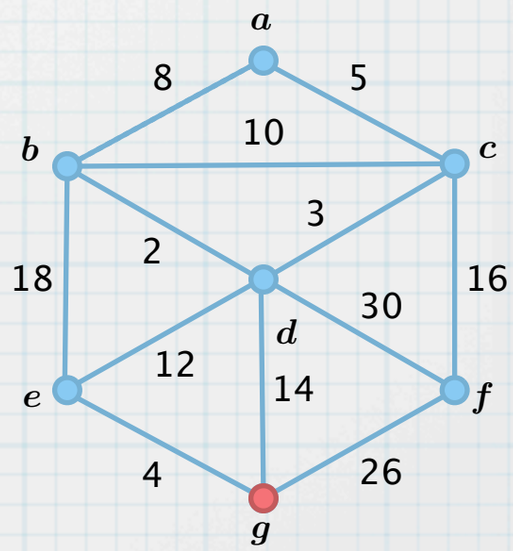


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:



Der Algorithmus von Jarnik (Prim)

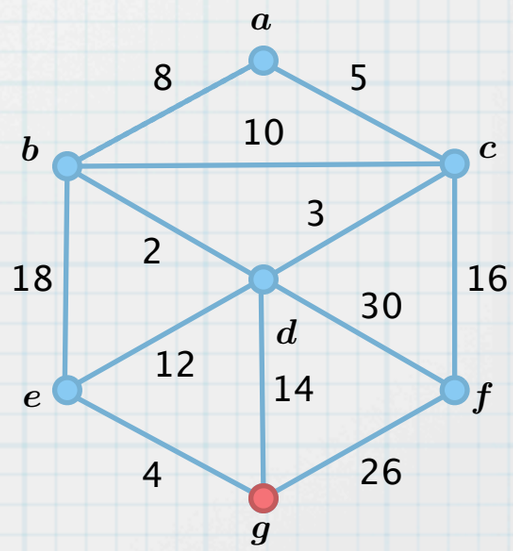
* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

$v = g$

$V(T) = \{g\}$

* Beispiel:

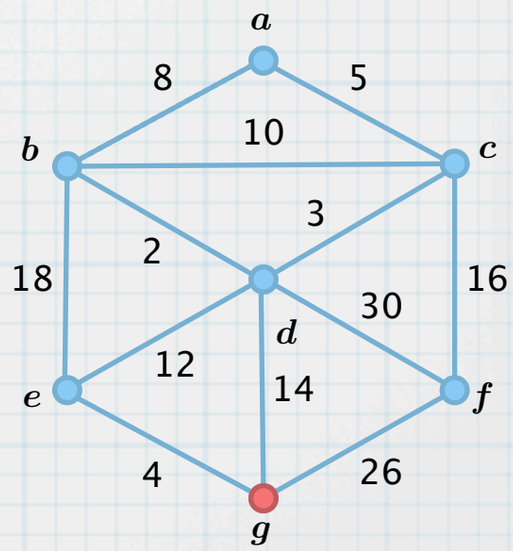


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $\quad v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ $\quad V(T) = \{g\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

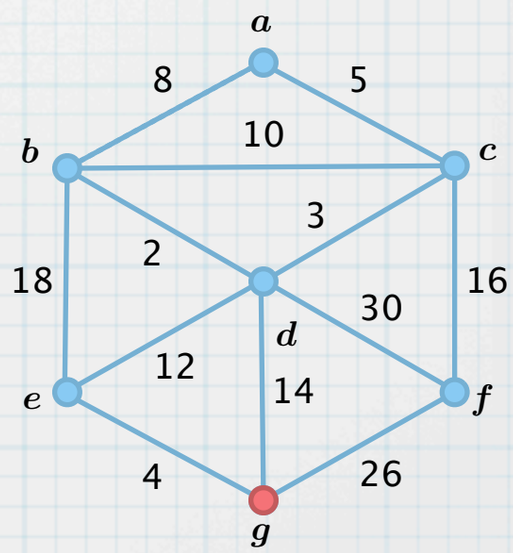


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $\qquad v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ $\qquad V(T) = \{g\}$
(8) $E(T) := E(T) \cup \{e\}$ $\qquad E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:



Der Algorithmus von Jarnik (Prim)

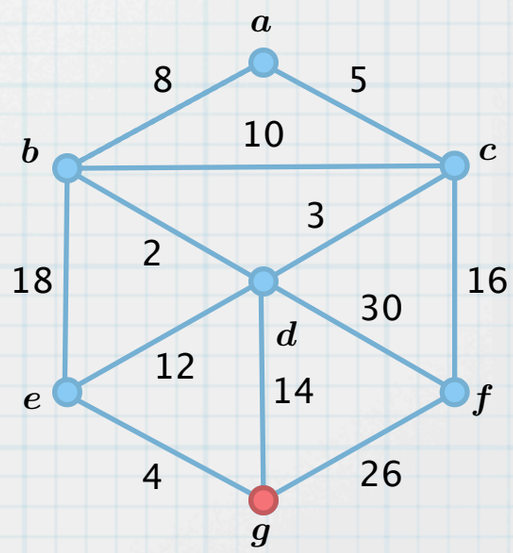
* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do**
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

$v = g$

$V(T) = \{g\}$
$E(T) = \{\}$

* Beispiel:

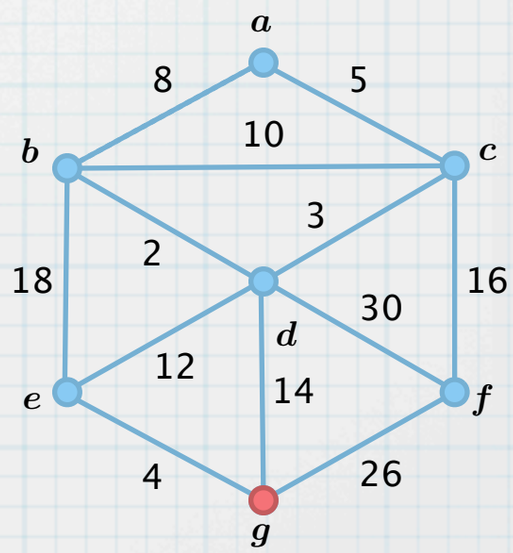


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$ — $V(T) = \{g\}$
(4) $E(T) := \emptyset$ — $E(T) = \{\}$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$
(8) $E(T) := E(T) \cup \{e\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

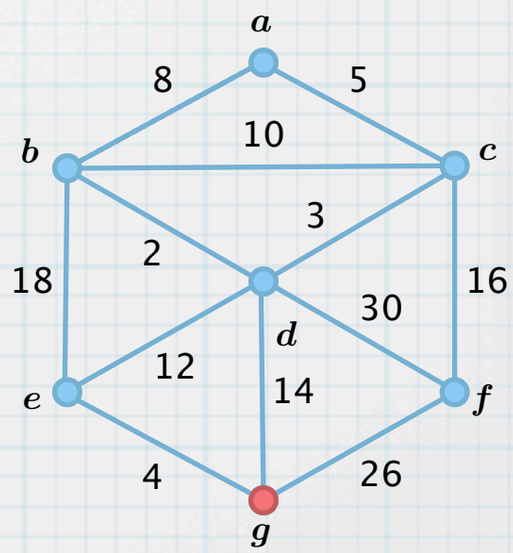


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

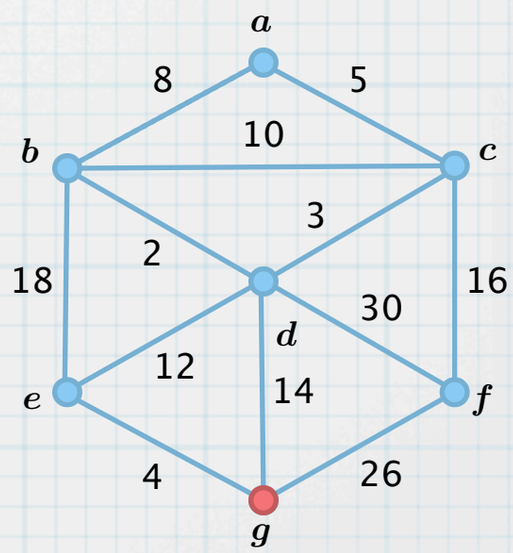


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

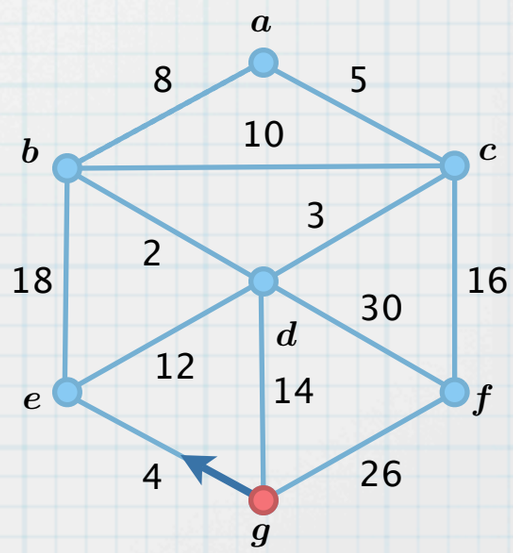


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$ — $e = \{g, e\}$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

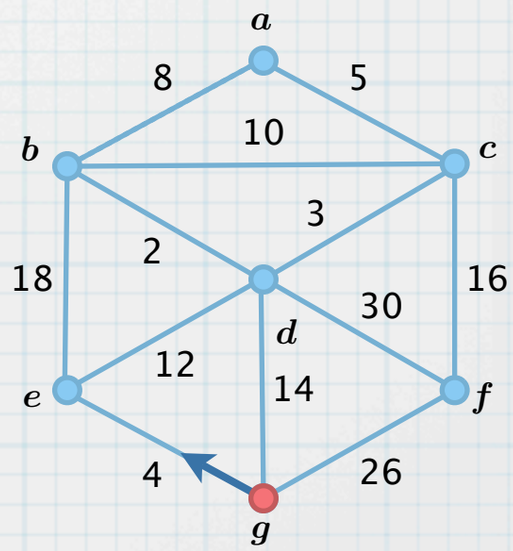


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$ — $e = \{g, e\}$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

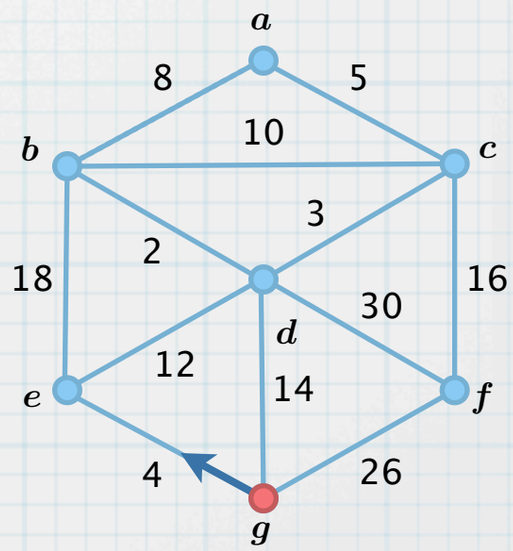


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

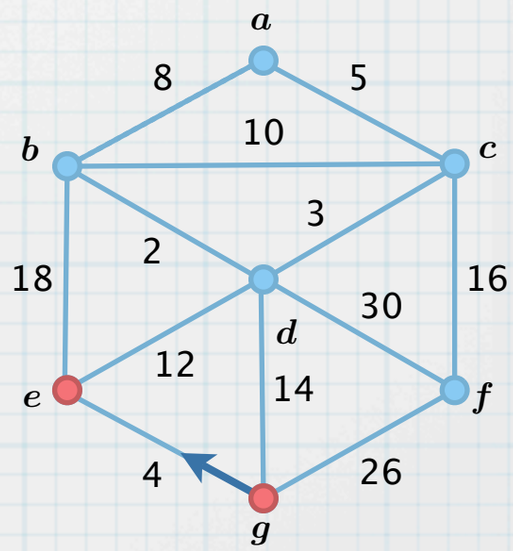


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

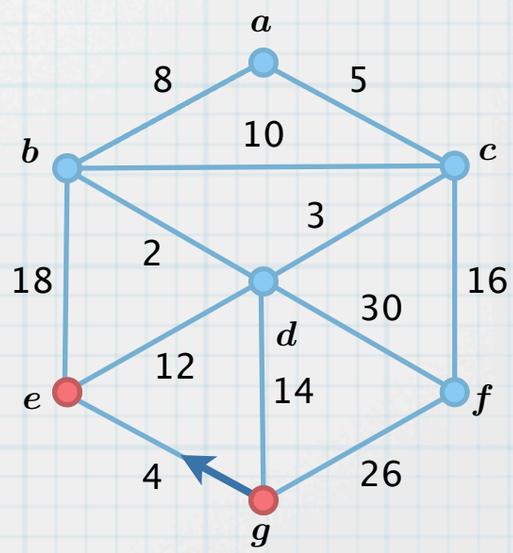


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

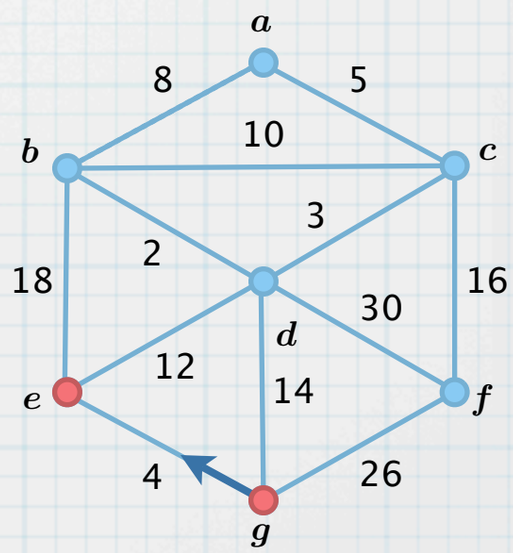


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

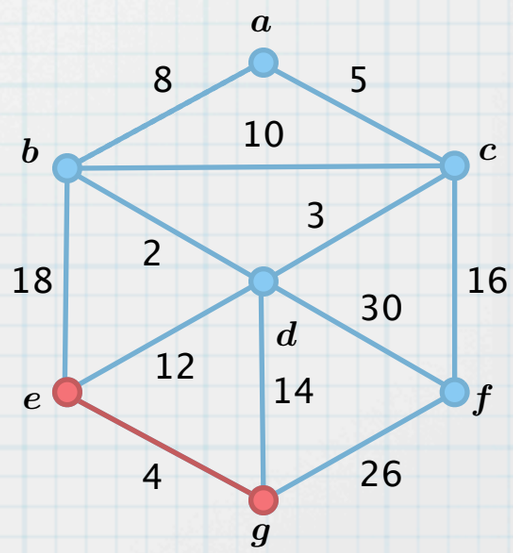


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

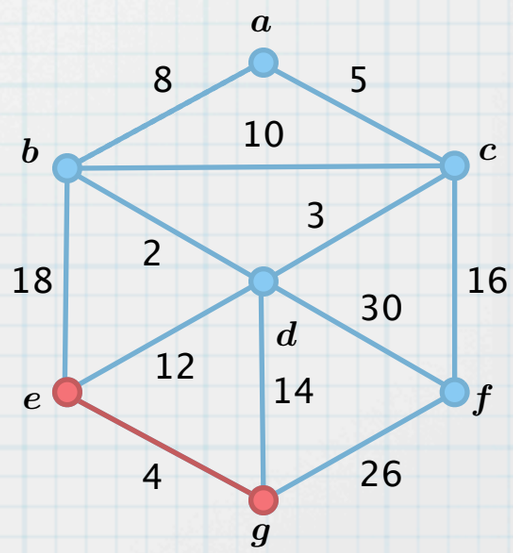


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$ — $e = \{g, e\}$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

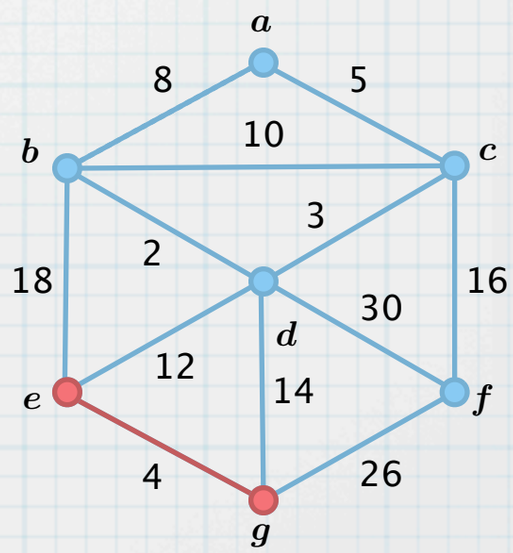


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $1 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

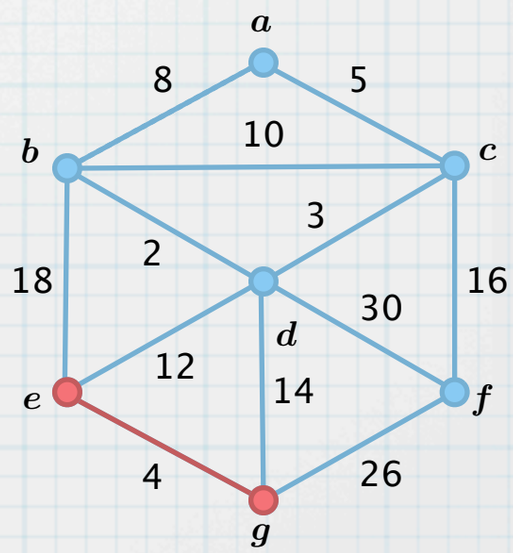


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

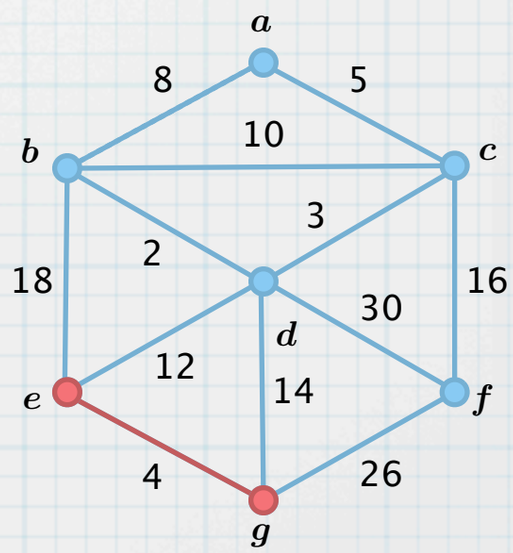


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{g, e\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

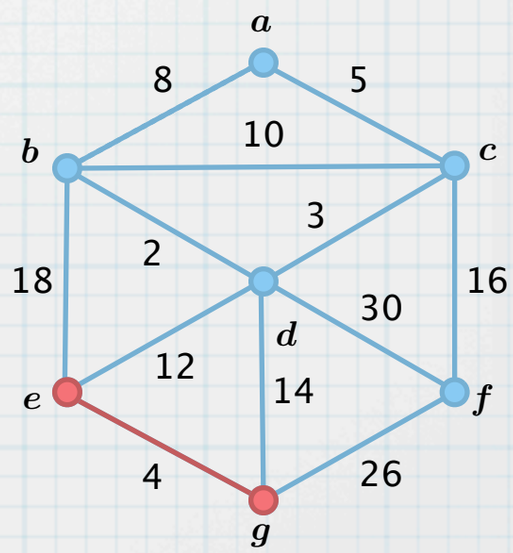


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$ — $e = \{e, d\}$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

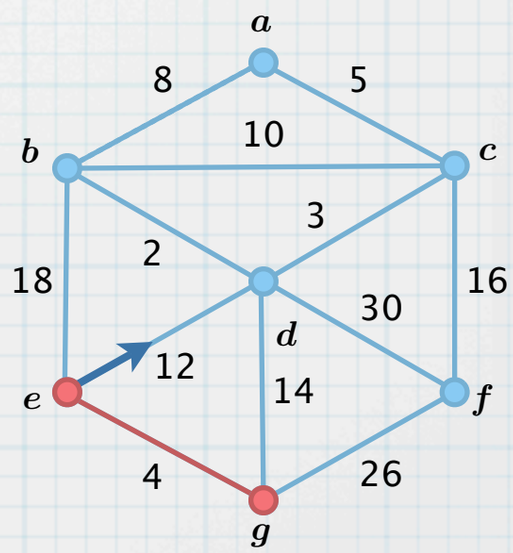


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$ — $e = \{e, d\}$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

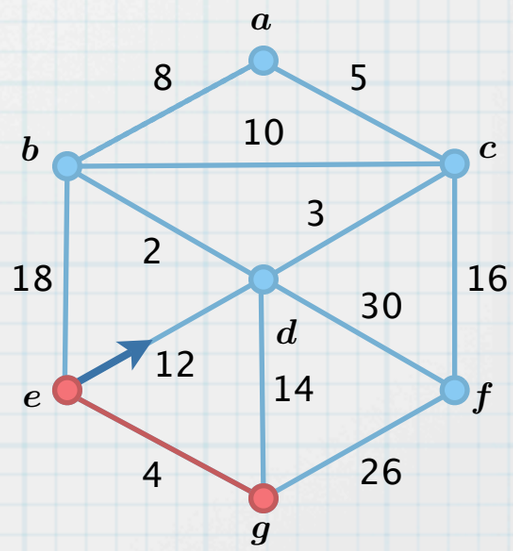


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

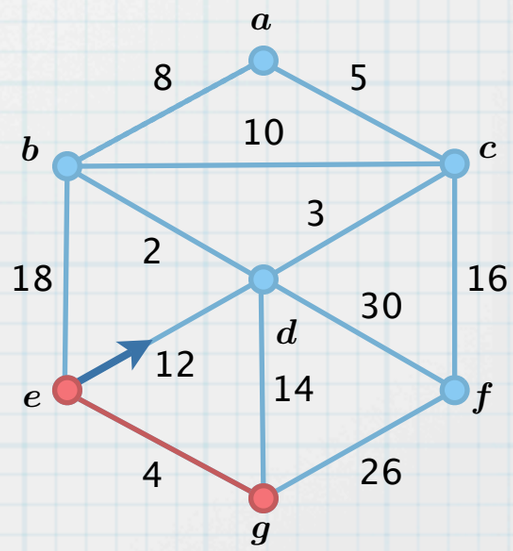


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

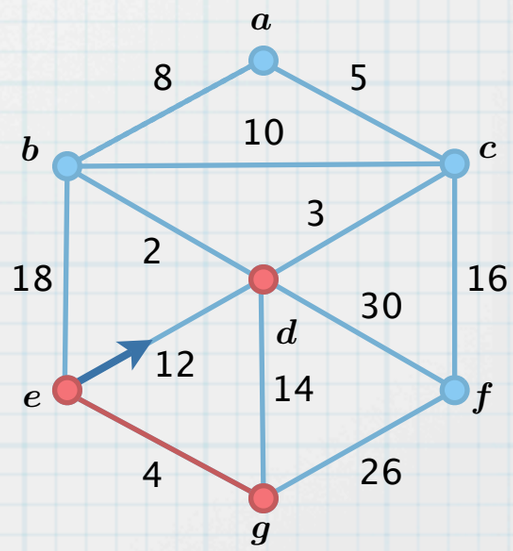


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

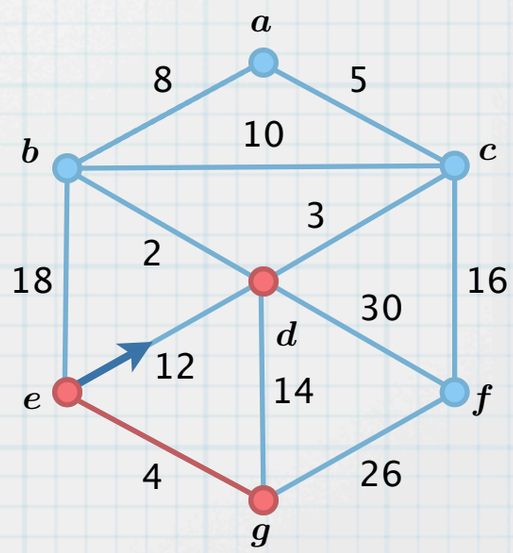


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

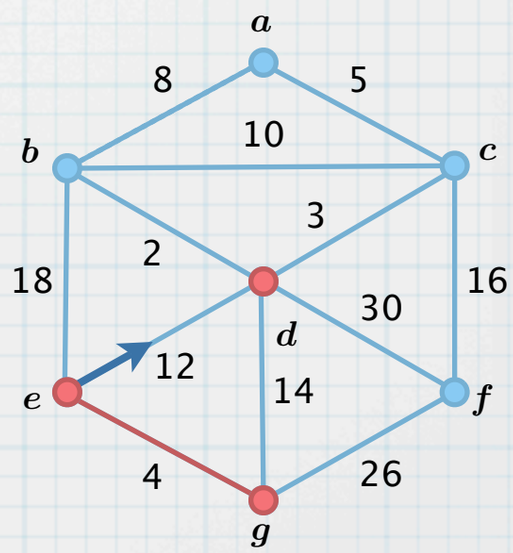


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

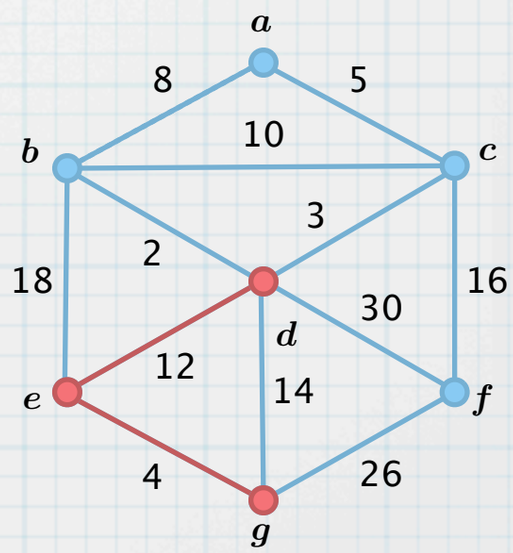


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

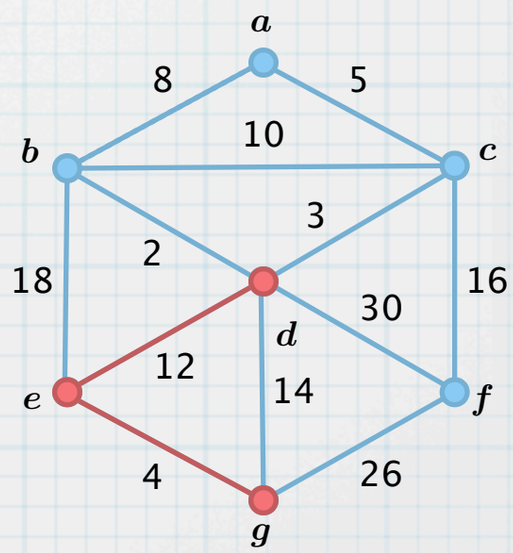


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

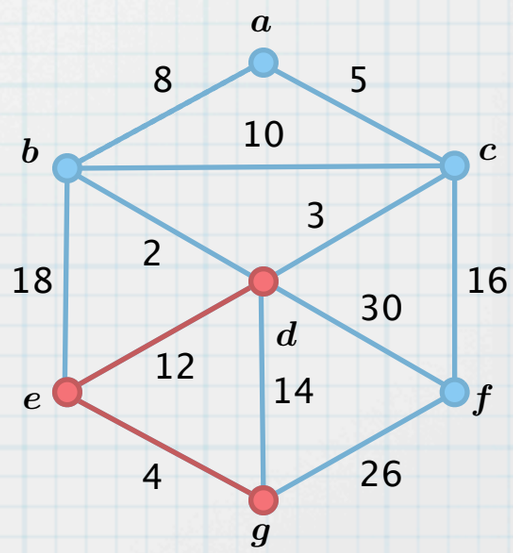


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $2 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

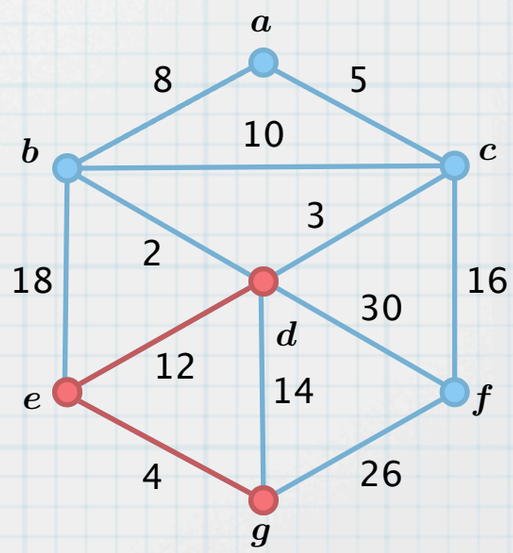


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

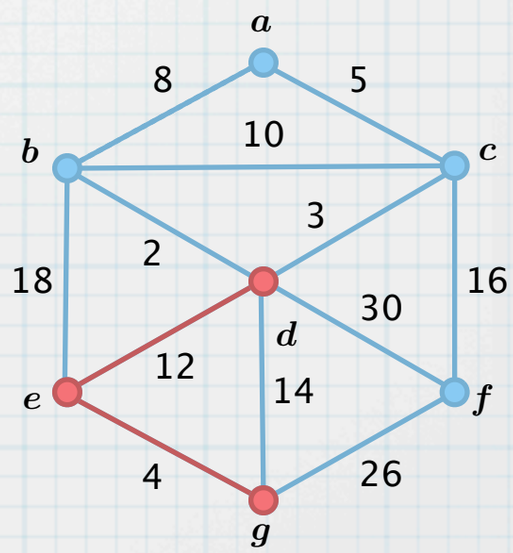


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{e, d\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

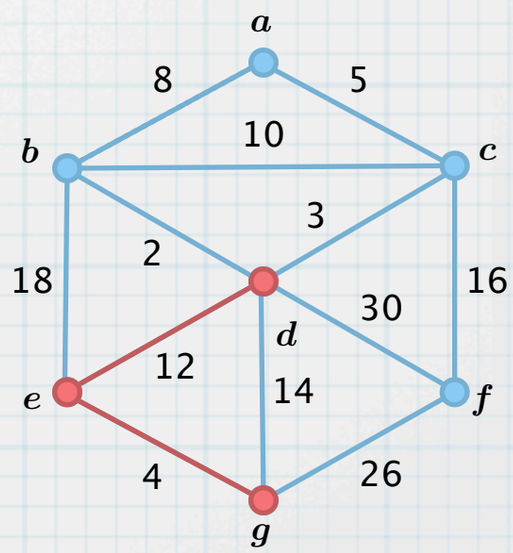


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

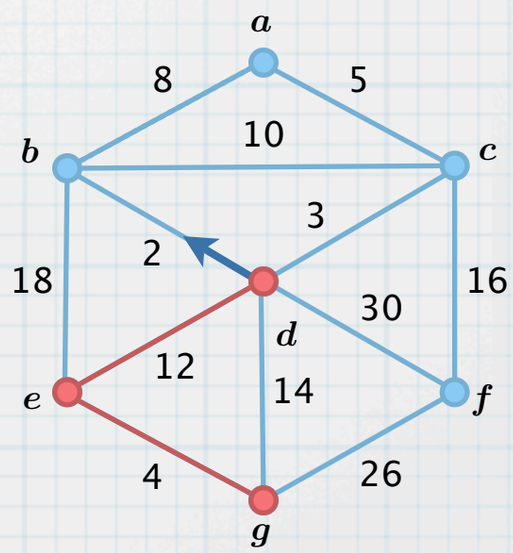


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

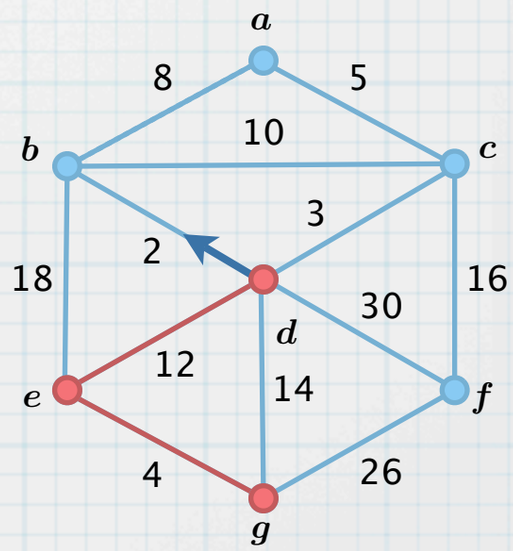


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

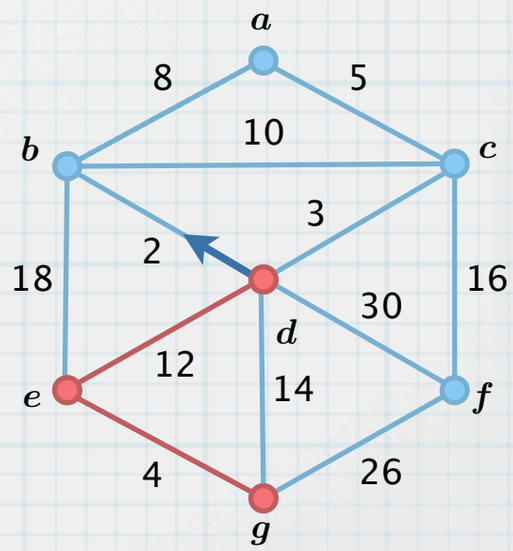


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

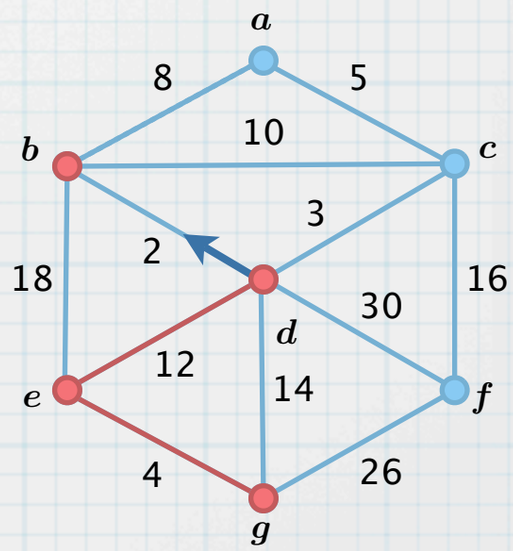


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

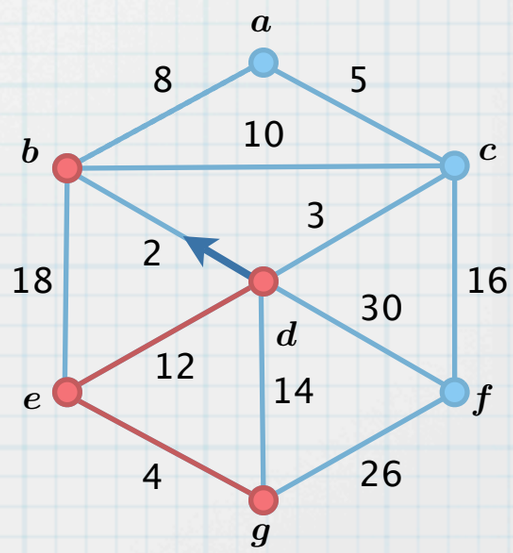


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

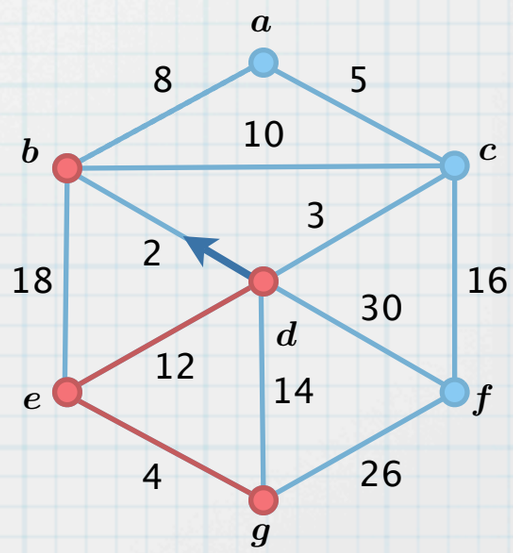


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

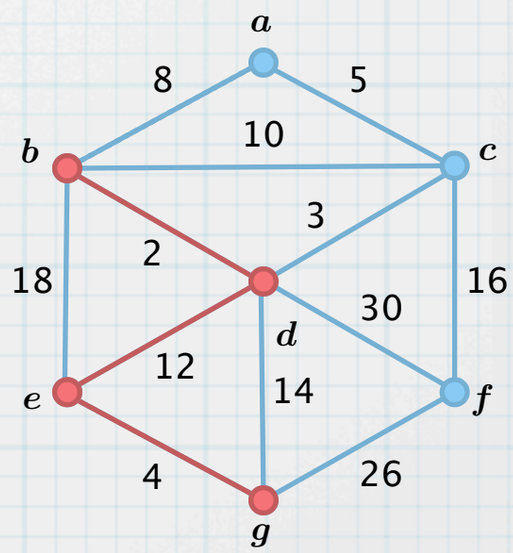


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

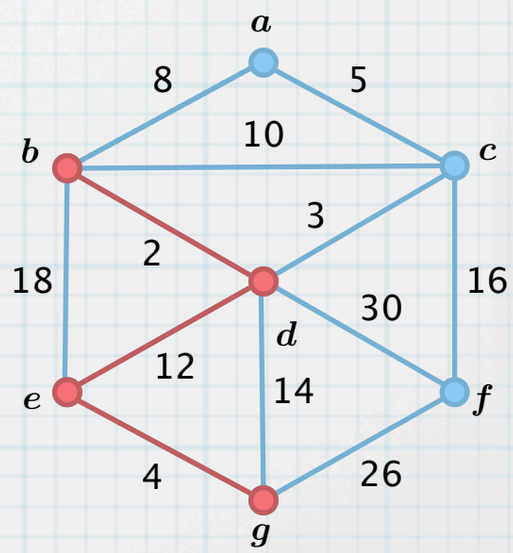


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

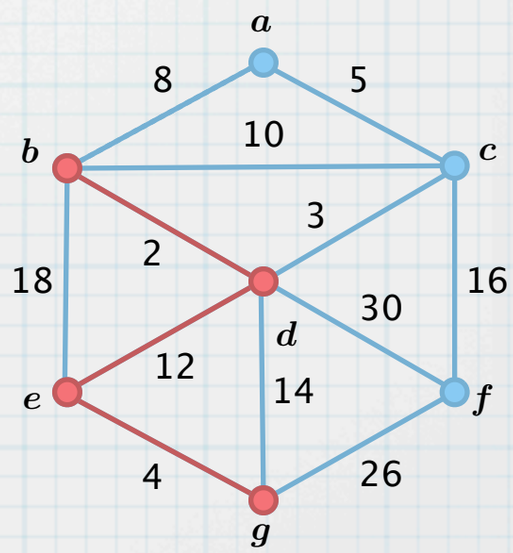


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $3 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

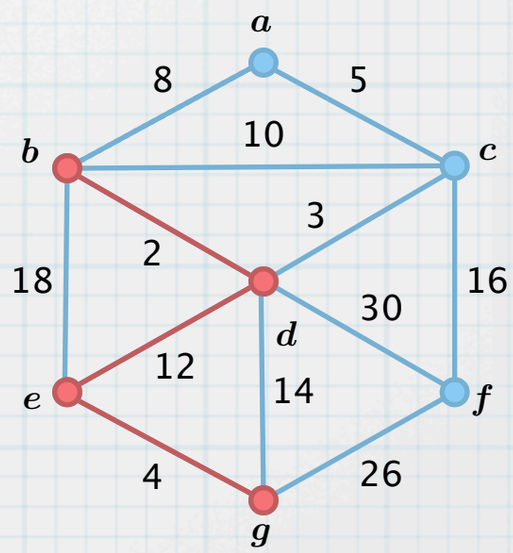


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

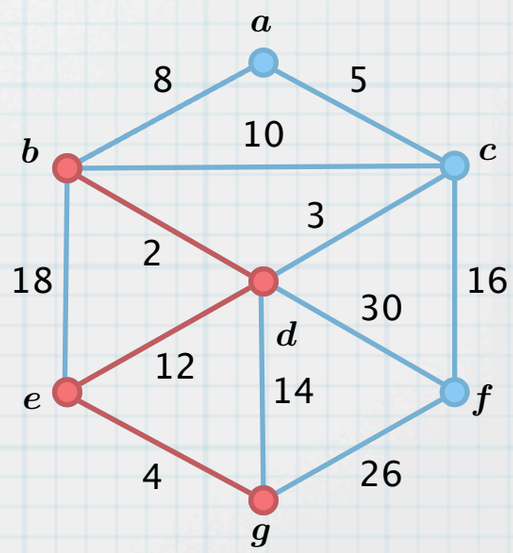


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $\quad v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** $\quad 4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ $\quad e = \{d, b\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ $\quad V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ $\quad E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

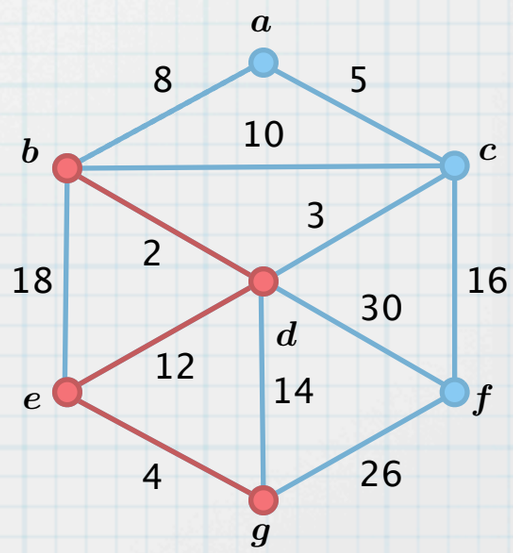


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

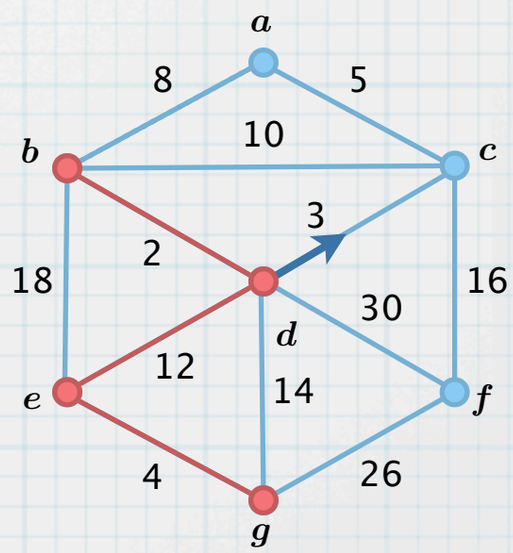


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

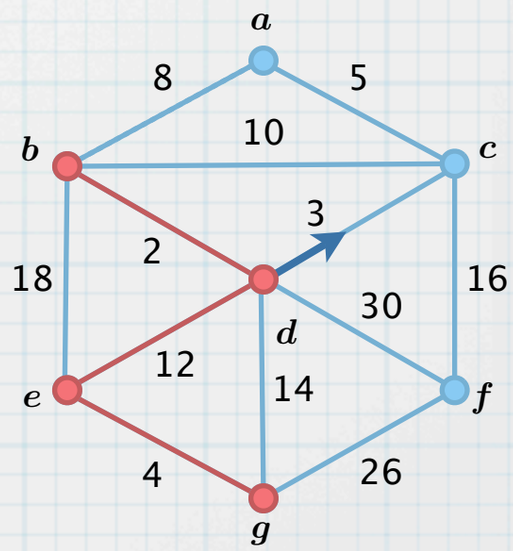


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

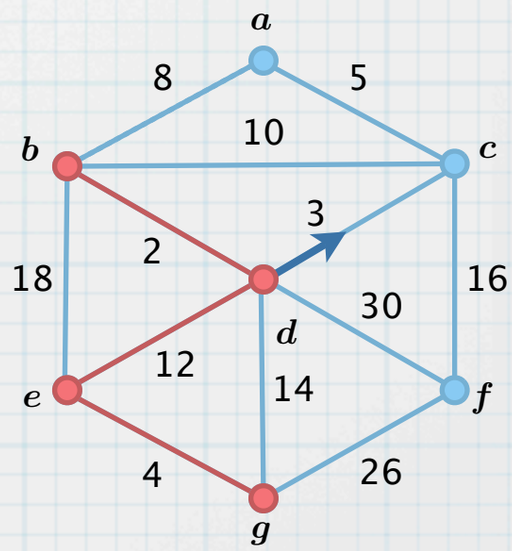


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

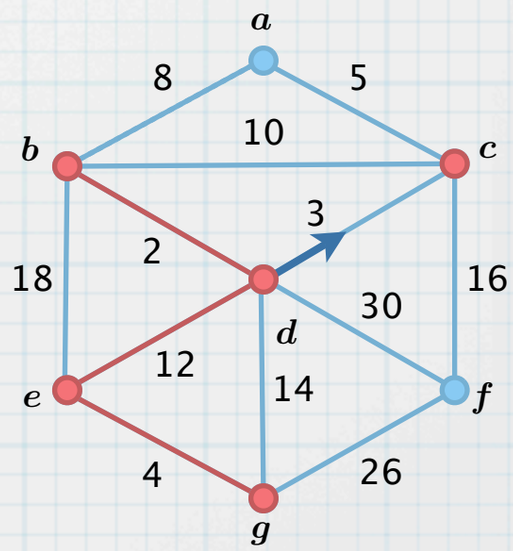


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

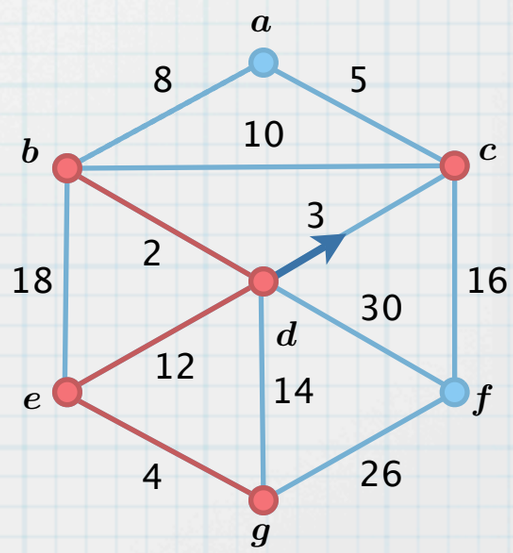


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

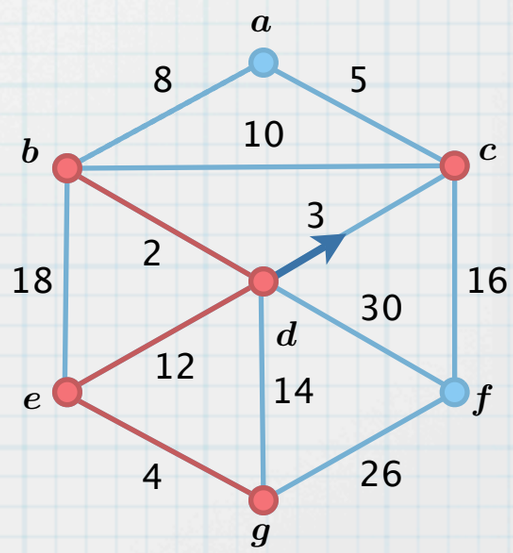


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

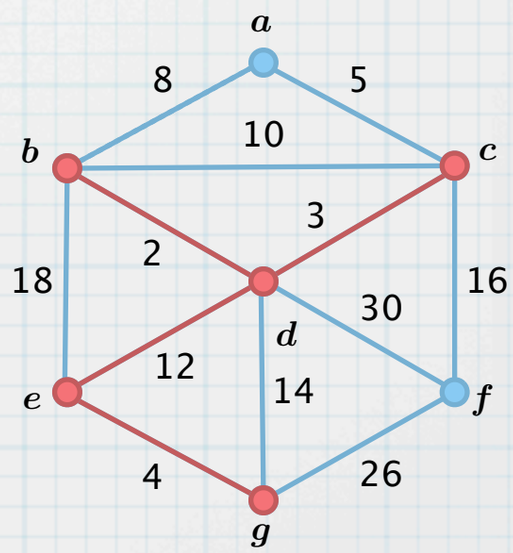


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

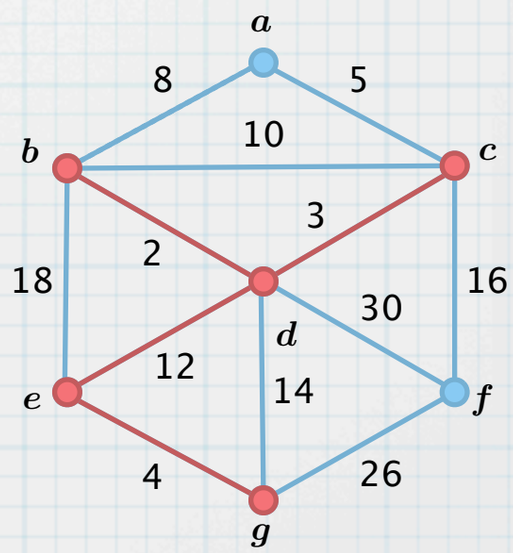


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

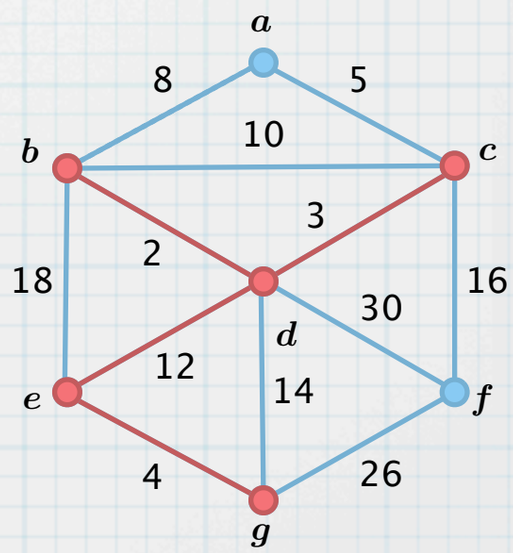


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $4 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

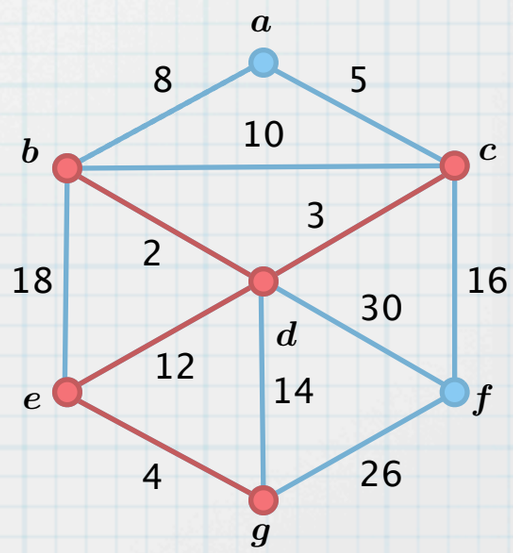


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

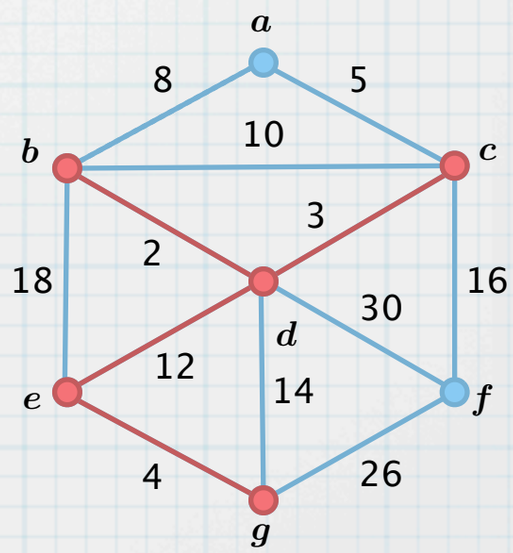


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{d, c\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

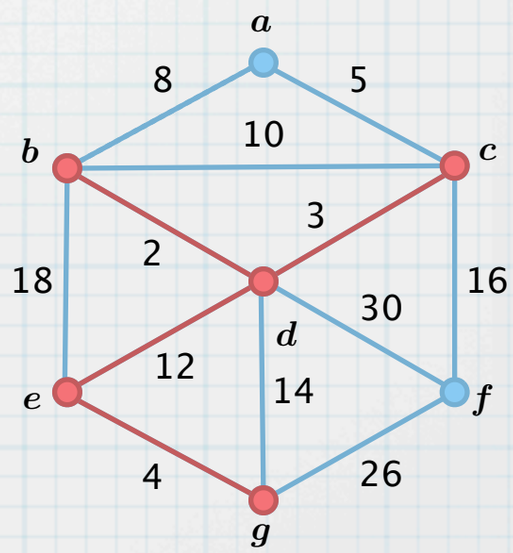


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

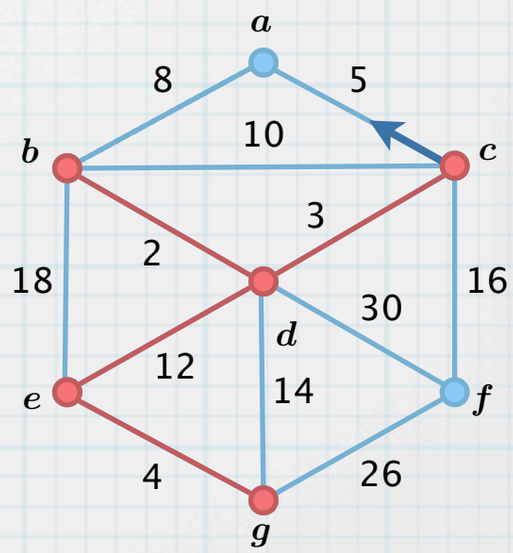


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

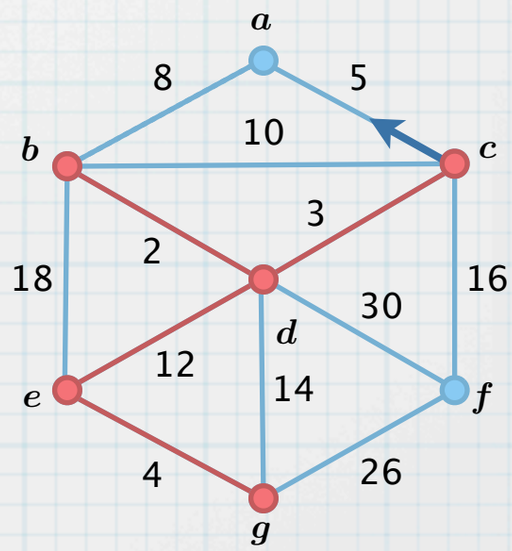


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

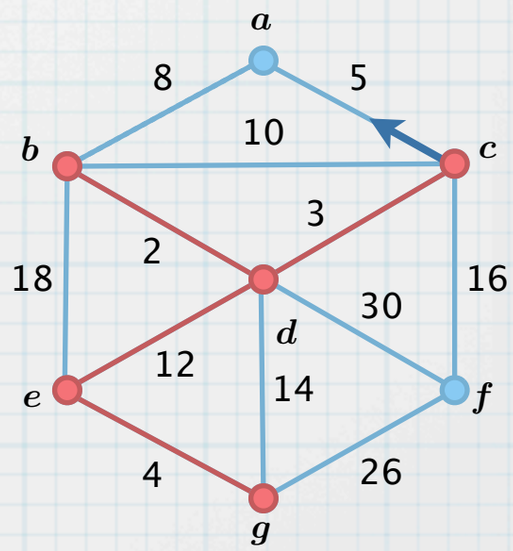


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

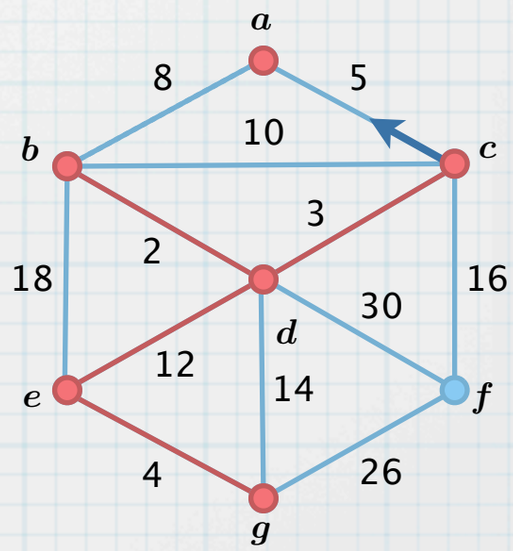


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

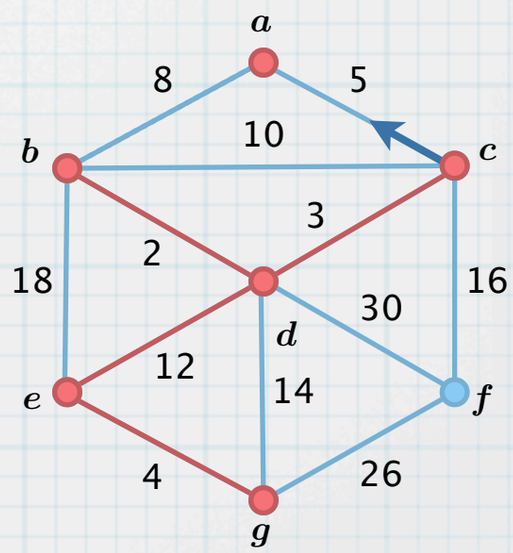


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

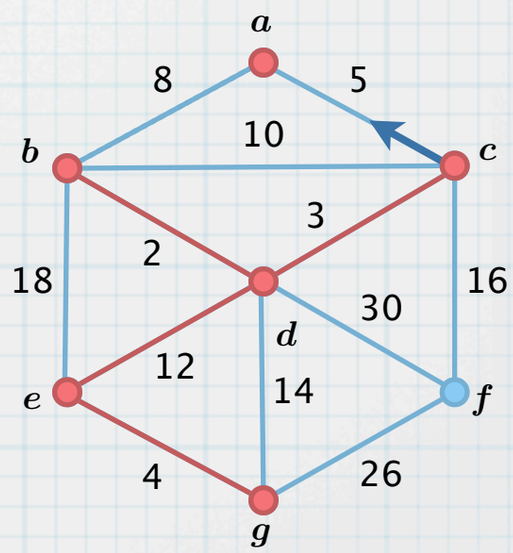


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

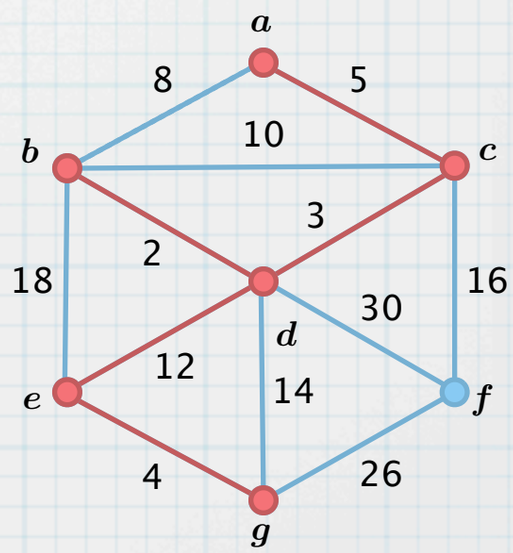


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

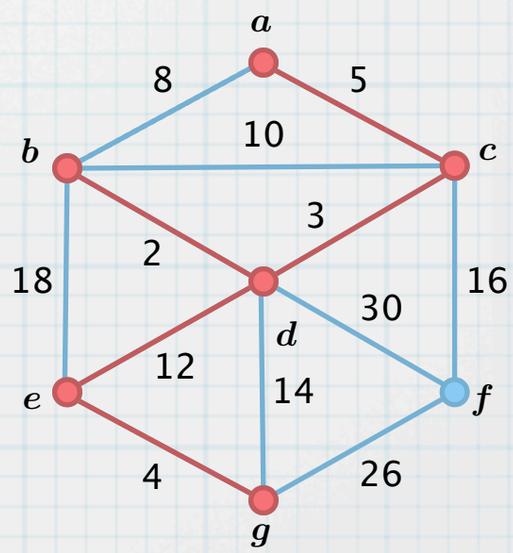


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

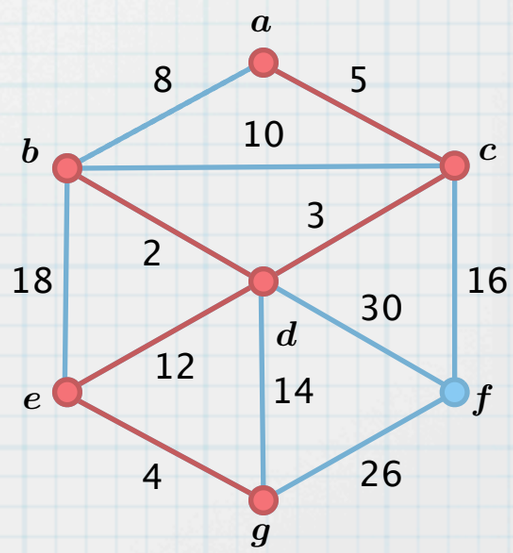


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $5 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

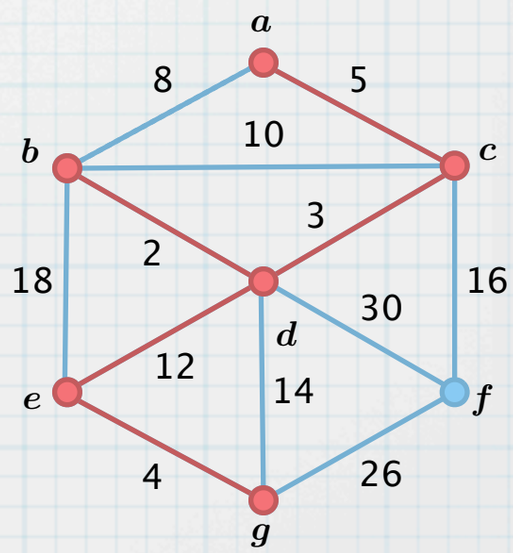


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

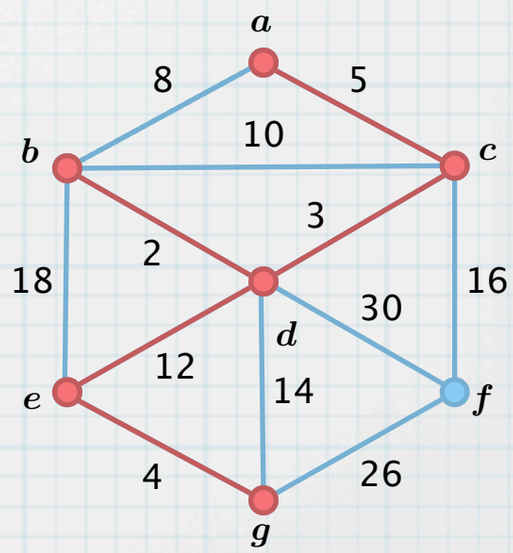


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, a\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

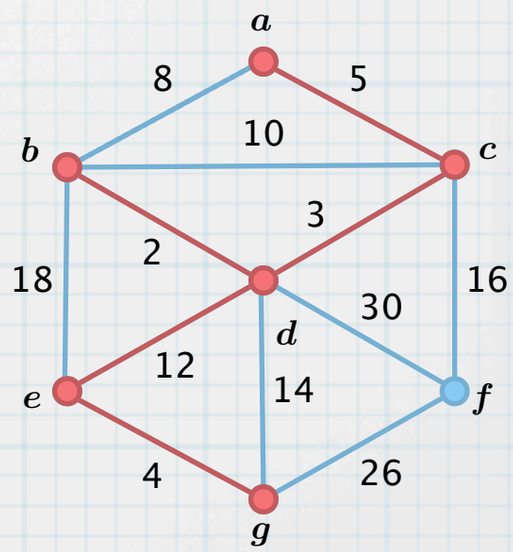


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $\quad v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** $\quad 6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ $\quad e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ $\quad V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ $\quad E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

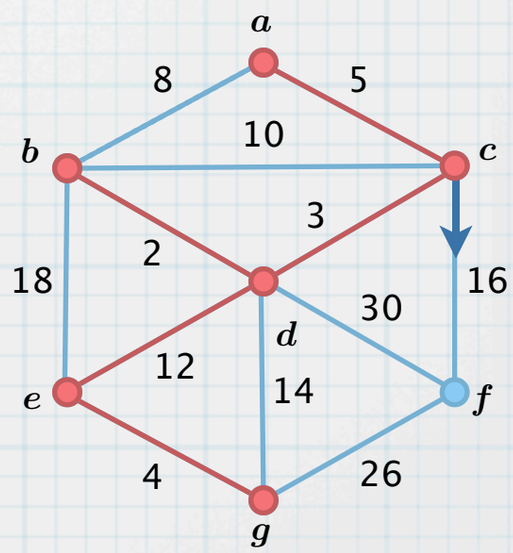


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ $\quad v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** $\quad 6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ $\quad e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ $\quad V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ $\quad E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

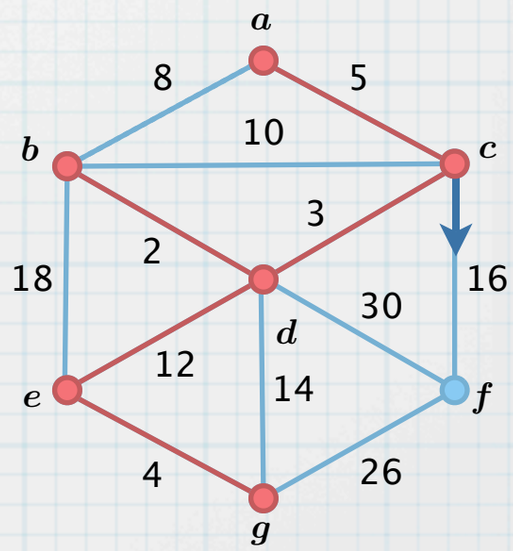


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

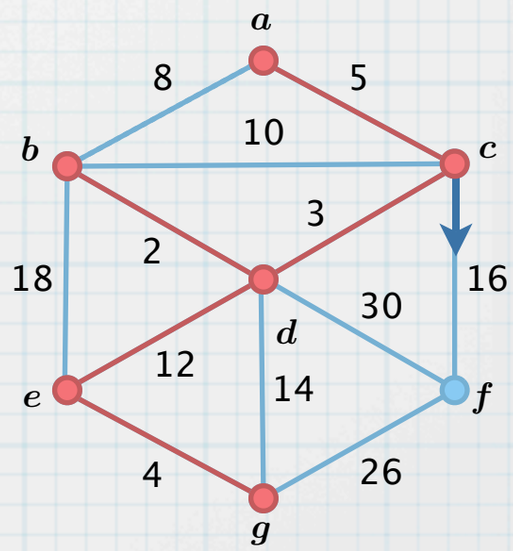


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

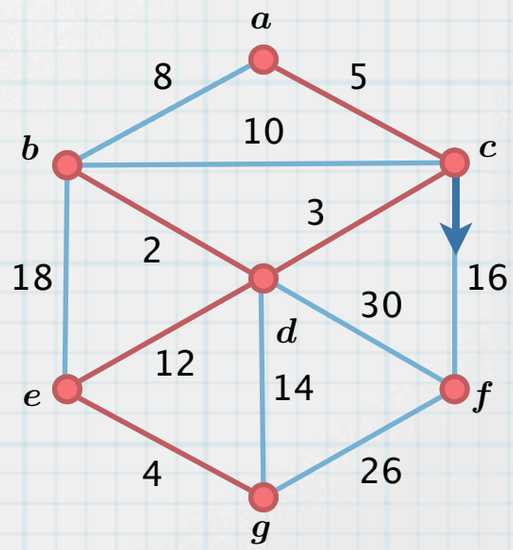


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

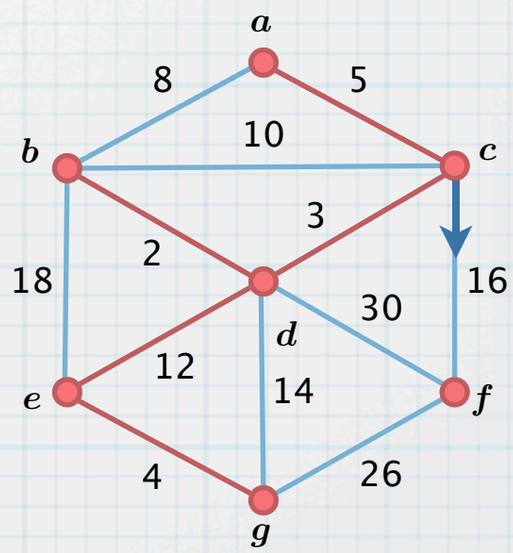


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

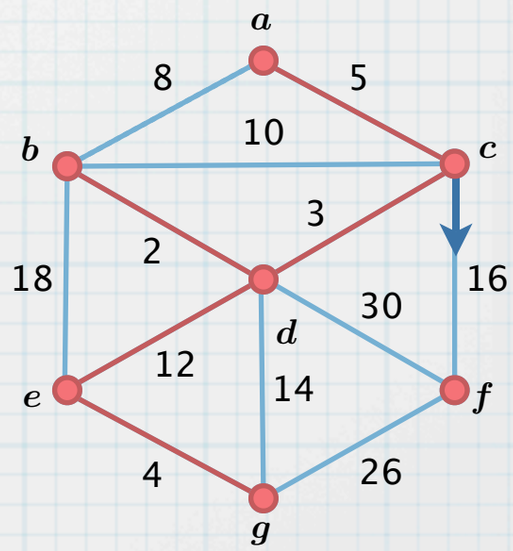


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5, 16\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

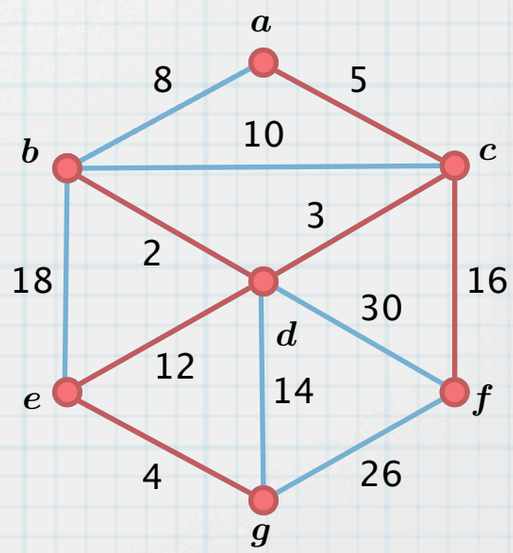


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5, 16\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

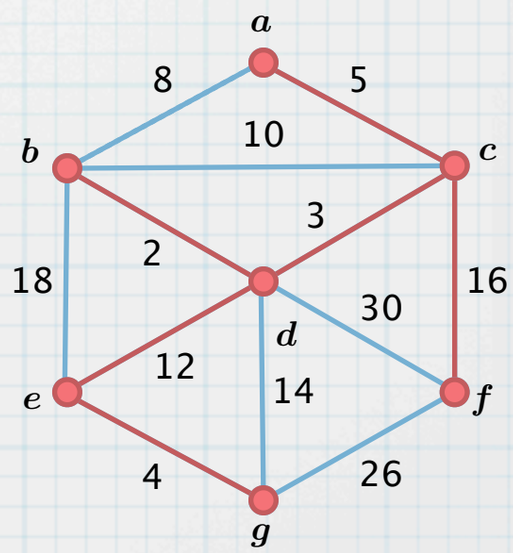


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5, 16\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

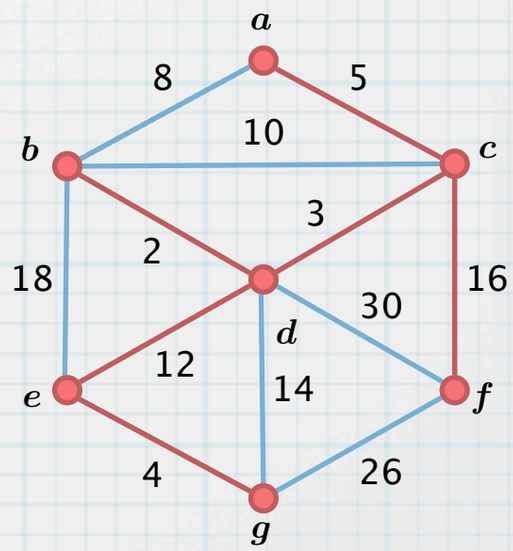


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $6 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5, 16\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

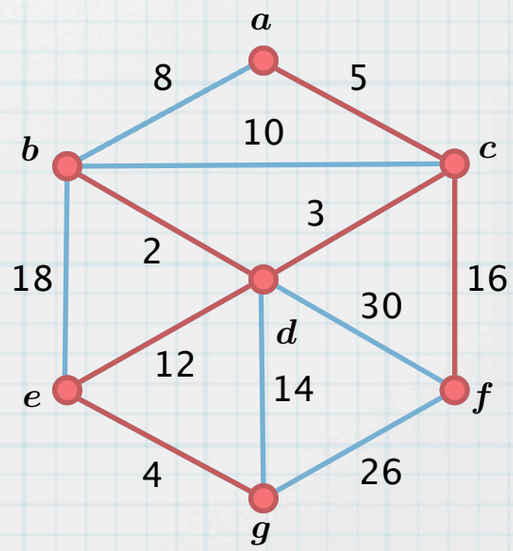


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $7 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5, 16\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:

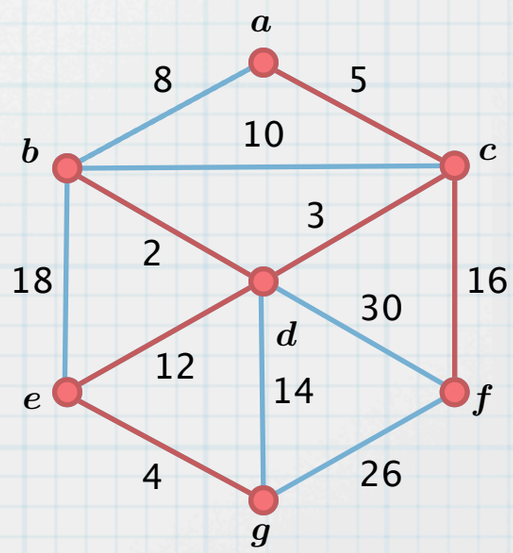


Der Algorithmus von Jarnik (Prim)

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** prim
(2) wähle Knoten $v \in V$ — $v = g$
(3) $V(T) := \{v\}$
(4) $E(T) := \emptyset$
(5) **while** $|V(T)| < |V|$ **do** — $7 < 7$
(6) wähle Kante $e = \{u, v\} \in E, u \in V(T), v \notin V(T), w_e \leq w_{e'}$ — $e = \{c, f\}$
für alle $e' = \{u', v'\} \in E, u' \in V(T), v' \notin V(T)$
(7) $V(T) := V(T) \cup \{v\}$ — $V(T) = \{g, e, d, b, c, a, f\}$
(8) $E(T) := E(T) \cup \{e\}$ — $E(T) = \{4, 12, 2, 3, 5, 16\}$
(9) **end while**
(10) **end algorithm**

* Beispiel:



Jarniks Korrektheit und Laufzeit

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

  Um auf $O(|V|^2)$ Laufzeitkomplexität zu kommen, erzeugen wir eine Liste, in der zu jedem Knoten $u \in V$ die „billigste“ Kante $e = \{u, v\} \in E$ hinterlegt wird.

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

  Um auf $O(|V|^2)$ Laufzeitkomplexität zu kommen, erzeugen wir eine Liste, in der zu jedem Knoten $u \in V$ die „billigste“ Kante $e = \{u, v\} \in E$ hinterlegt wird.

  Die Erzeugung der Liste hat $O(|E|)$ Komplexität.

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

  Um auf $O(|V|^2)$ Laufzeitkomplexität zu kommen, erzeugen wir eine Liste, in der zu jedem Knoten $u \in V$ die „billigste“ Kante $e = \{u, v\} \in E$ hinterlegt wird.

  Die Erzeugung der Liste hat $O(|E|)$ Komplexität.

  Die Suche nach der „billigsten“ Kante der Liste hat $O(|V|)$ Komplexität.

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

  Um auf $O(|V|^2)$ Laufzeitkomplexität zu kommen, erzeugen wir eine Liste, in der zu jedem Knoten $u \in V$ die „billigste“ Kante $e = \{u, v\} \in E$ hinterlegt wird.

  Die Erzeugung der Liste hat $O(|E|)$ Komplexität.

  Die Suche nach der „billigsten“ Kante der Liste hat $O(|V|)$ Komplexität.

  Diese Liste ist nach jeder Runde (while...end while) im Algorithmus zu aktualisieren.

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

  Um auf $O(|V|^2)$ Laufzeitkomplexität zu kommen, erzeugen wir eine Liste, in der zu jedem Knoten $u \in V$ die „billigste“ Kante $e = \{u, v\} \in E$ hinterlegt wird.

  Die Erzeugung der Liste hat $O(|E|)$ Komplexität.

  Die Suche nach der „billigsten“ Kante der Liste hat $O(|V|)$ Komplexität.

  Diese Liste ist nach jeder Runde (while...end while) im Algorithmus zu aktualisieren.

  Für diese Aktualisierung werden alle Kanten geprüft, die mit dem neu hinzugefügten Knoten von T inzidieren. Die billigste Kante wird wiederum gespeichert.

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

  Um auf $O(|V|^2)$ Laufzeitkomplexität zu kommen, erzeugen wir eine Liste, in der zu jedem Knoten $u \in V$ die „billigste“ Kante $e = \{u, v\} \in E$ hinterlegt wird.

  Die Erzeugung der Liste hat $O(|E|)$ Komplexität.

  Die Suche nach der „billigsten“ Kante der Liste hat $O(|V|)$ Komplexität.

  Diese Liste ist nach jeder Runde (while...end while) im Algorithmus zu aktualisieren.

  Für diese Aktualisierung werden alle Kanten geprüft, die mit dem neu hinzugefügten Knoten von T inzidieren. Die billigste Kante wird wiederum gespeichert.

  Dieses hat $O(|V|)$ Komplexität und wird insgesamt $|V| - 1$ wiederholt (while-Schleife).

Jarniks Korrektheit und Laufzeit

* **Satz 20**:
  Der Algorithmus von Jarnik bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V|^2)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die eine Teilmenge der Knoten mit dem übrigen Knoten verbindet.

  Nach Lemma 19 wird daher ein minimaler Spannbaum bestimmt (Induktionsargument).

  Um auf $O(|V|^2)$ Laufzeitkomplexität zu kommen, erzeugen wir eine Liste, in der zu jedem Knoten $u \in V$ die „billigste“ Kante $e = \{u, v\} \in E$ hinterlegt wird.

  Die Erzeugung der Liste hat $O(|E|)$ Komplexität.

  Die Suche nach der „billigsten“ Kante der Liste hat $O(|V|)$ Komplexität.

  Diese Liste ist nach jeder Runde (while...end while) im Algorithmus zu aktualisieren.

  Für diese Aktualisierung werden alle Kanten geprüft, die mit dem neu hinzugefügten Knoten von T inzidieren. Die billigste Kante wird wiederum gespeichert.

  Dieses hat $O(|V|)$ Komplexität und wird insgesamt $|V| - 1$ wiederholt (while-Schleife).

  Komplexität des Algorithmus damit $O(|E| + |V| \cdot (|V| - 1) + |V| \cdot (|V| - 1)) = O(|V|^2)$.

Der Algorithmus von Kruskal

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal

(2) sortiere Kanten in E nach Gewicht

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
```

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
```

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
```

Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

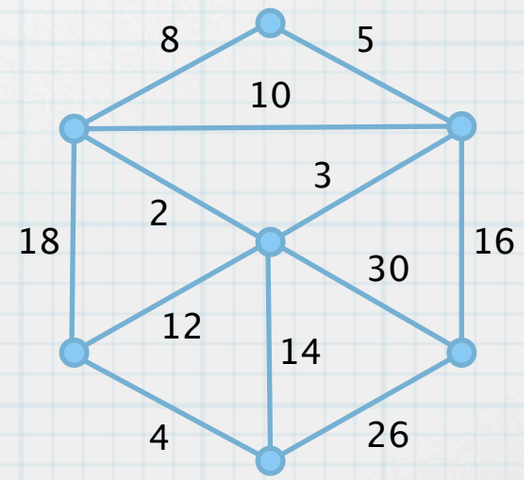
Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$$E = \{8, 5, 10, 18, 2, 3, 16, 12, 14, 30, 4, 26\}$$



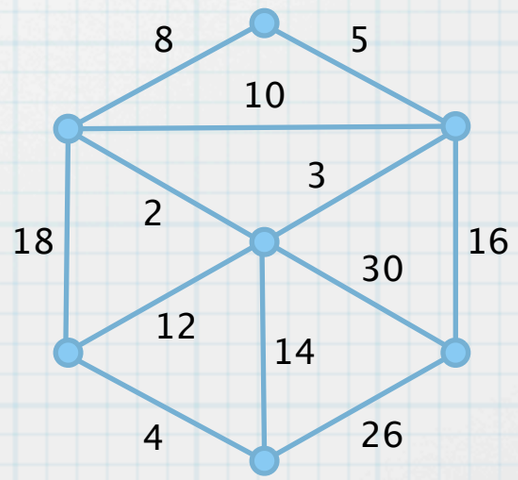
Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$$E = \{8, 5, 10, 18, 2, 3, 16, 12, 14, 30, 4, 26\}$$



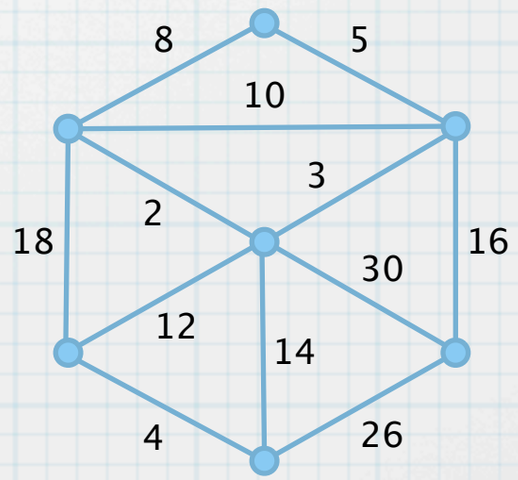
Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$$E = \{8, 5, 10, 18, 2, 3, 16, 12, 14, 30, 4, 26\}$$



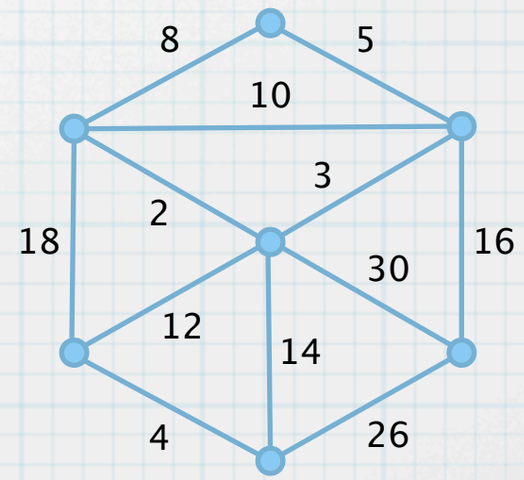
Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



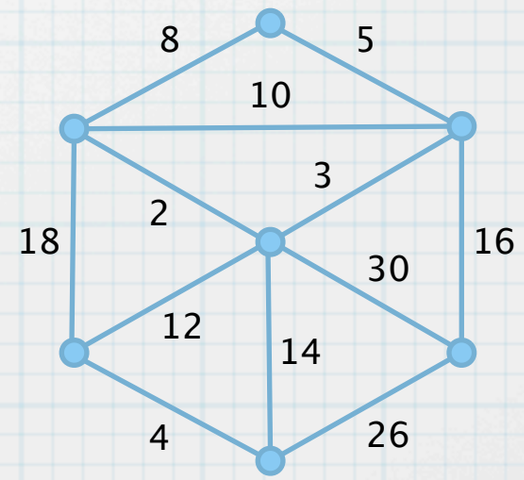
Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

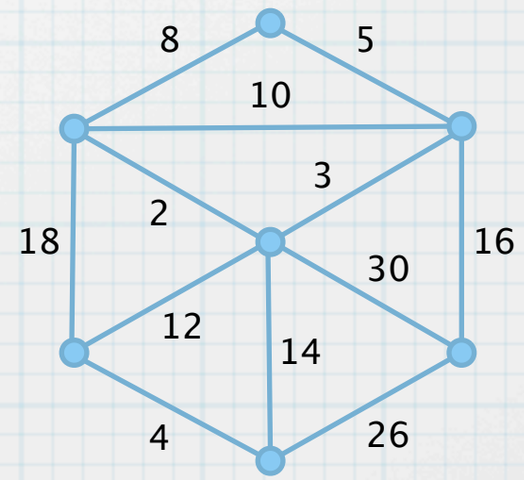
* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

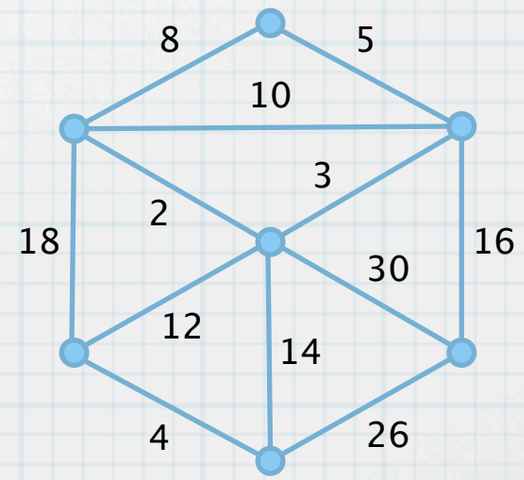
* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

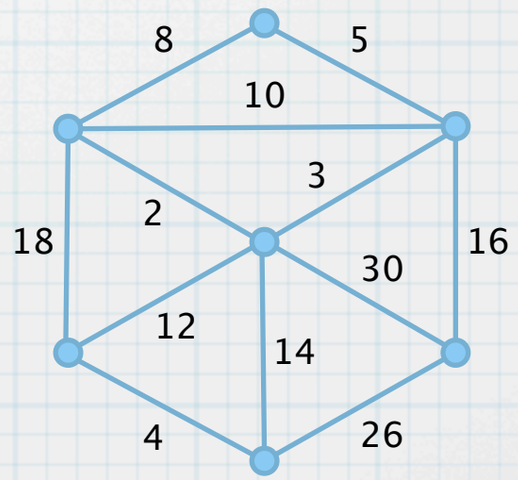
```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$0 < 6$

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

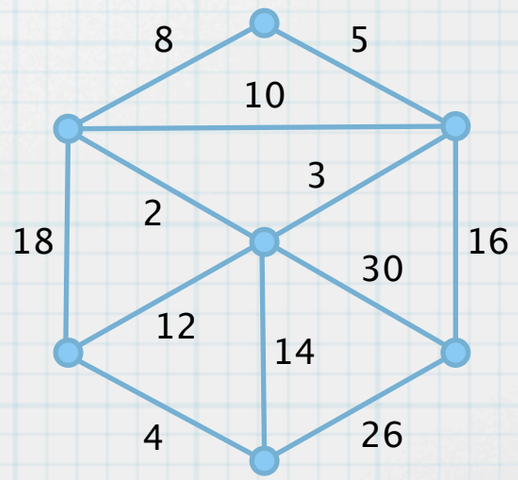
```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

* Beispiel:

$0 < 6$

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

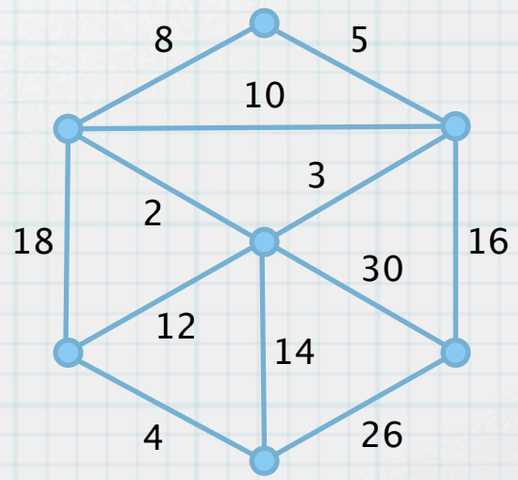
$0 < 6$

$e = 2$

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

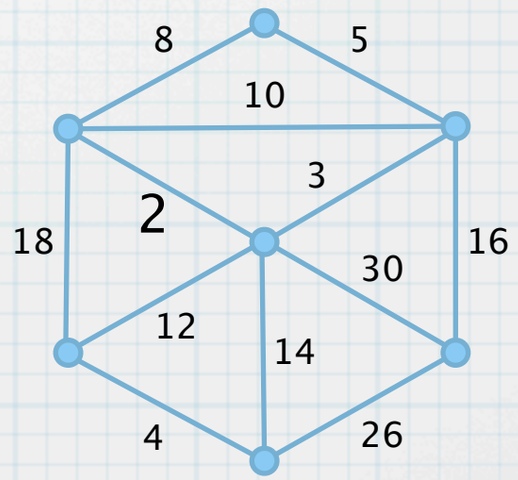
* Beispiel:

$0 < 6$

$e = 2$

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

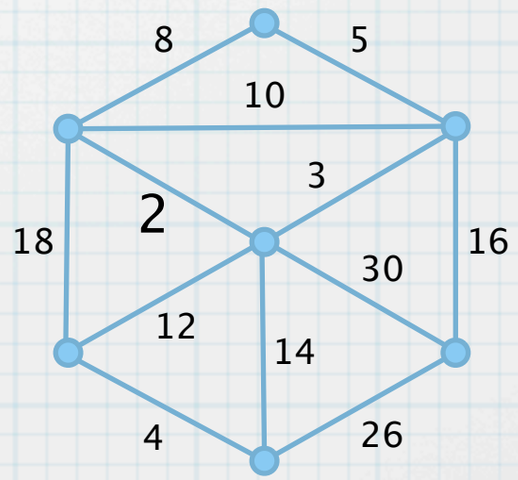
* Beispiel:

$0 < 6$

$e = 2$

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

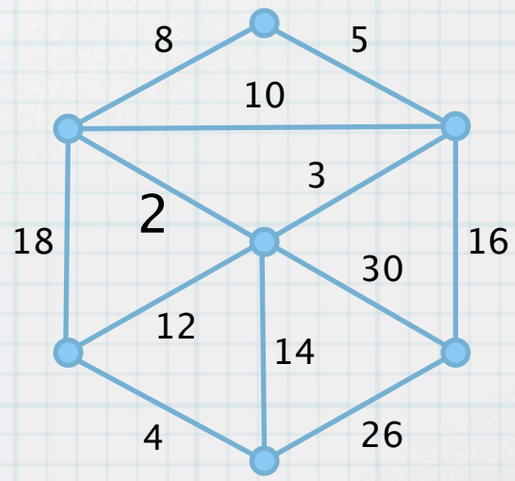
* Beispiel:

$0 < 6$

$e = 2$

$E(T) := \{\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

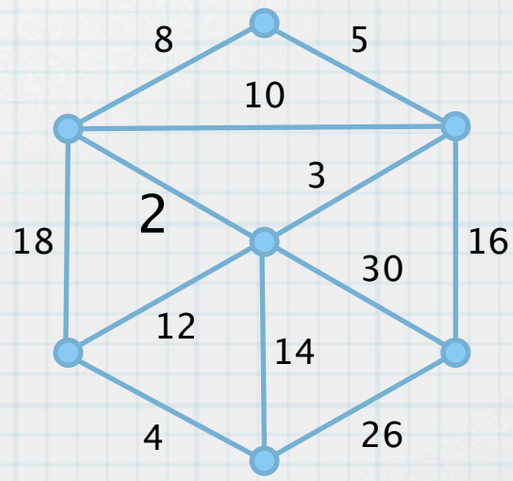
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

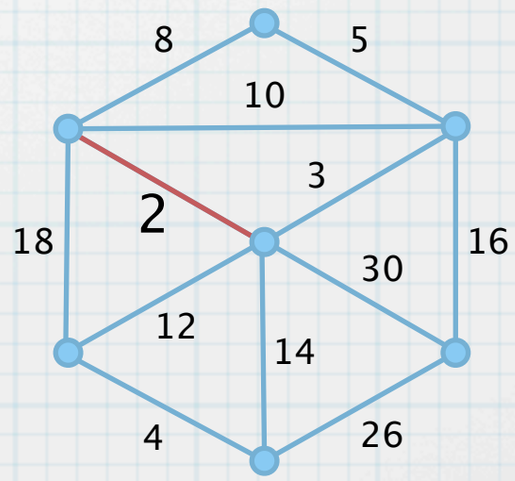
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

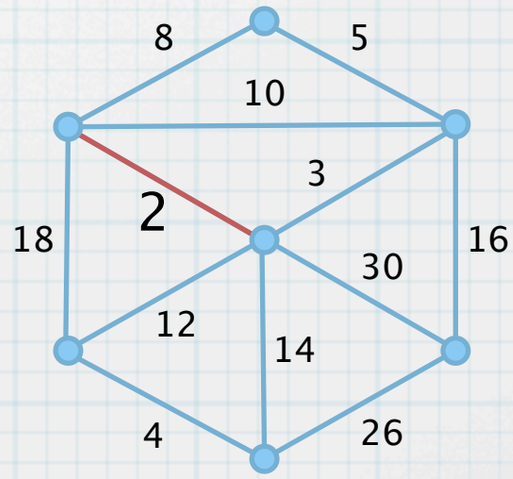
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

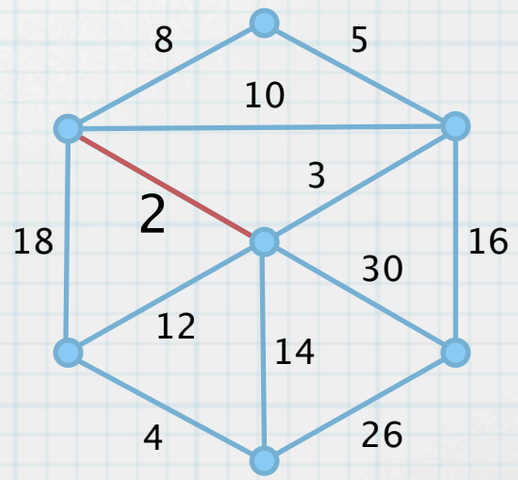
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

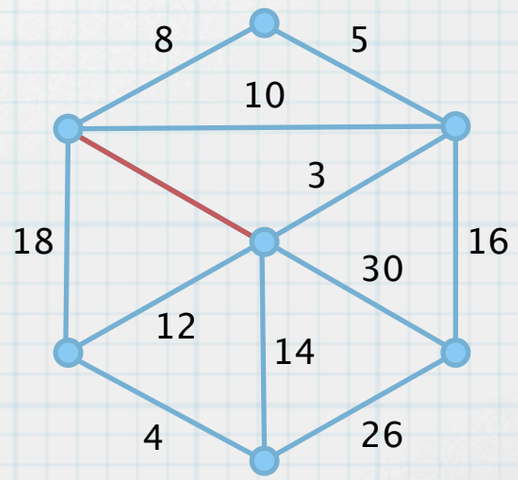
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (2, 3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

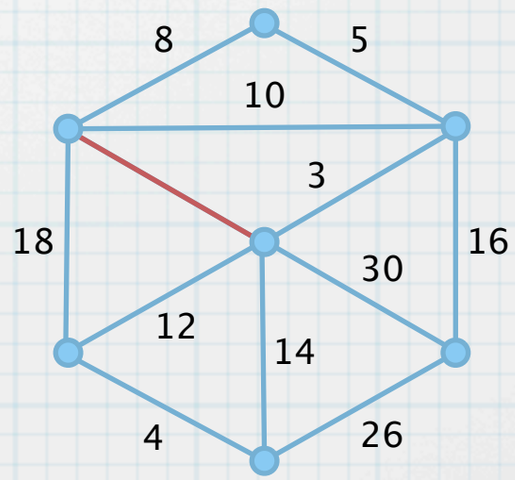
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

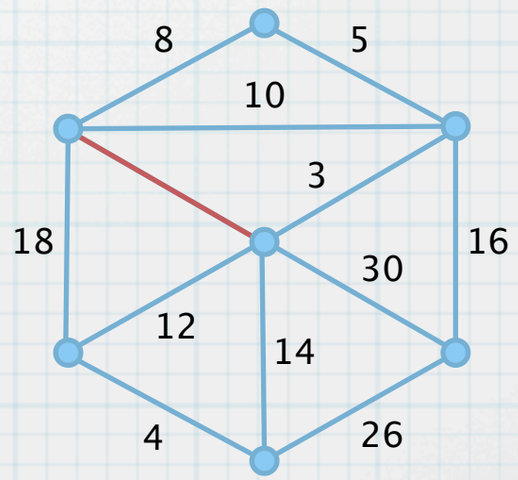
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

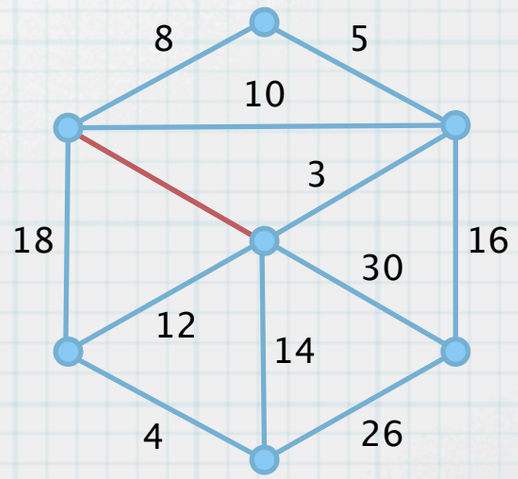
$0 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

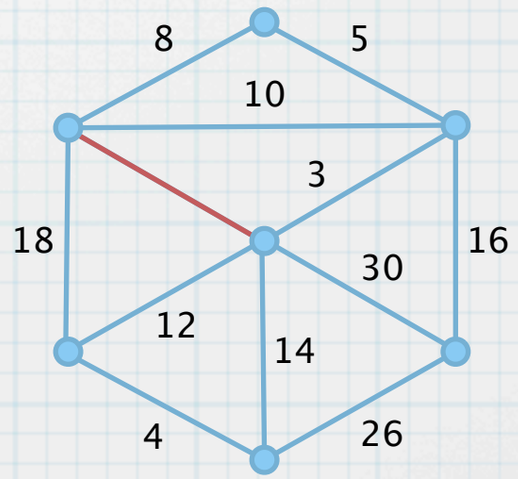
$1 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

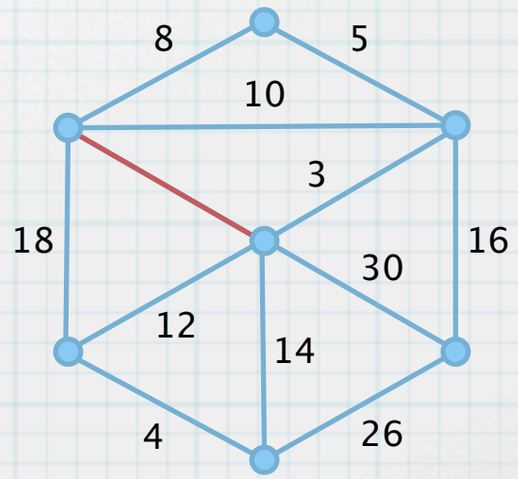
* Beispiel:

$1 < 6$

$e = 2$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

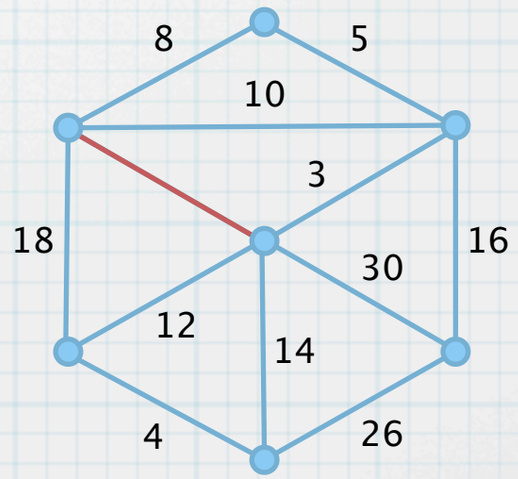
$1 < 6$

$e = 3$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

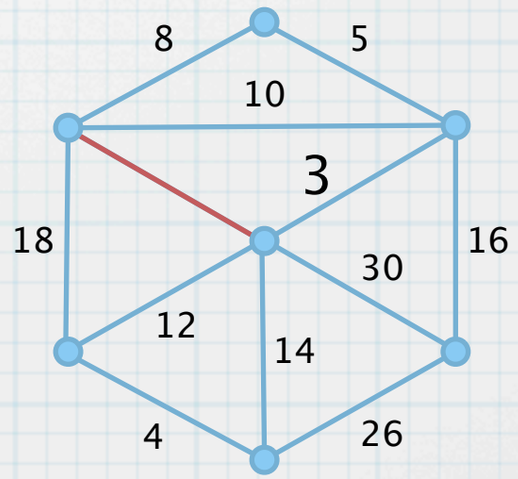
* Beispiel:

$1 < 6$

$e = 3$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

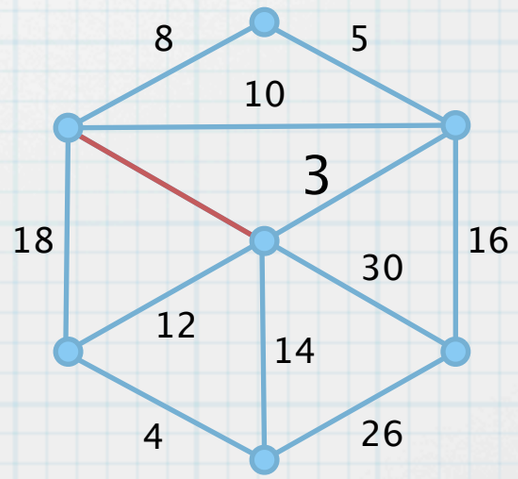
$1 < 6$

$e = 3$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

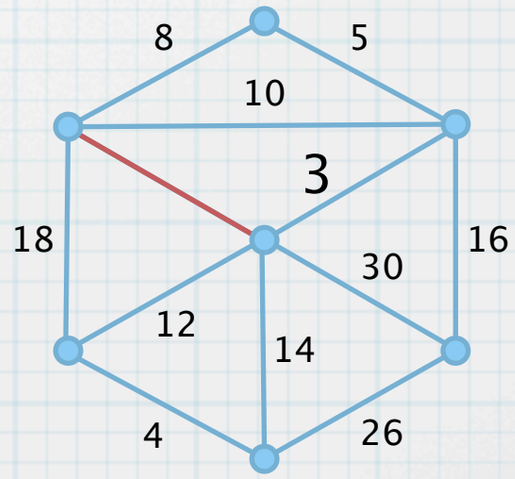
$1 < 6$

$e = 3$

$E(T) := \{2\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

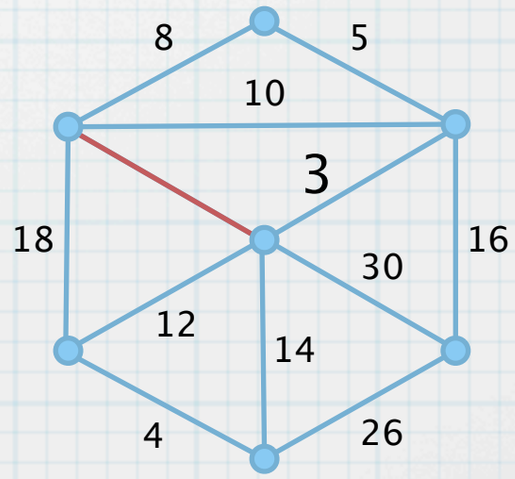
$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

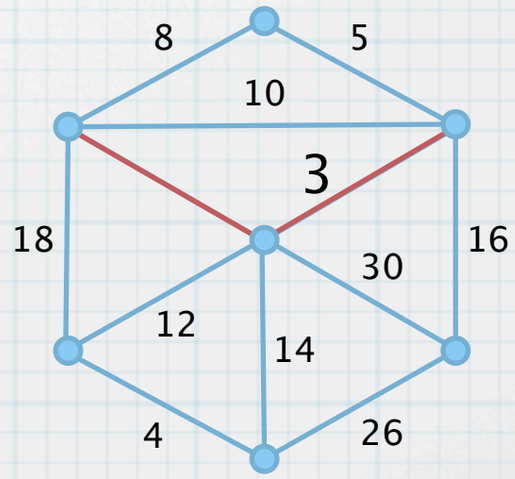
* Beispiel:

$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

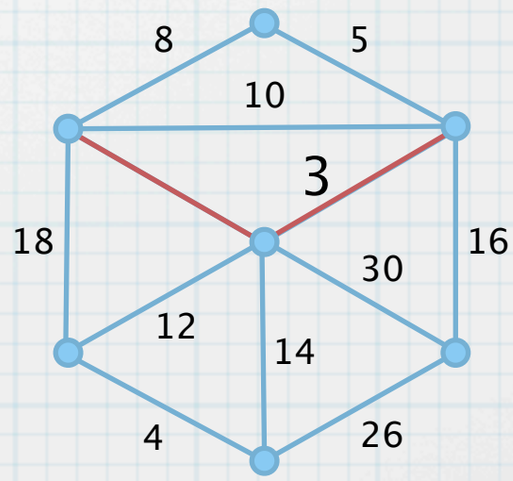
$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

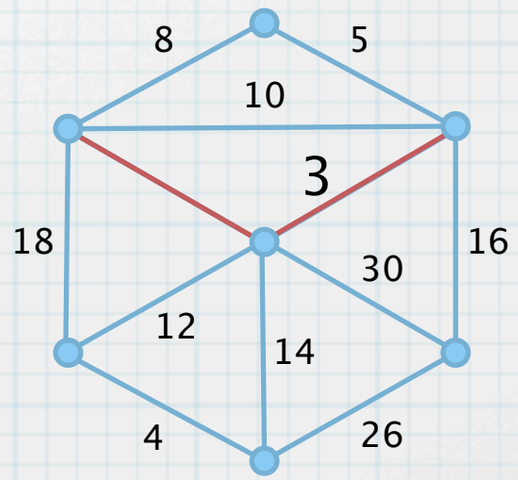
$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

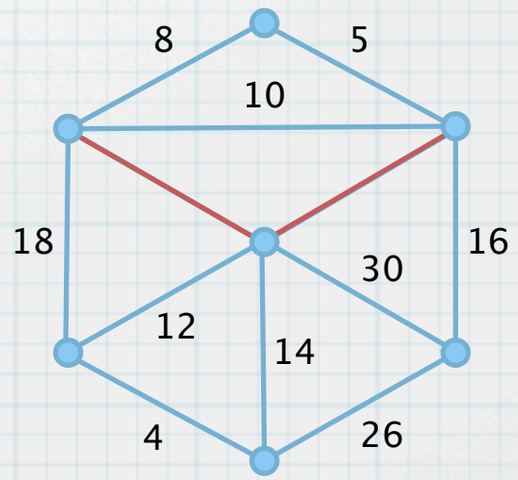
* Beispiel:

$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (3, 4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

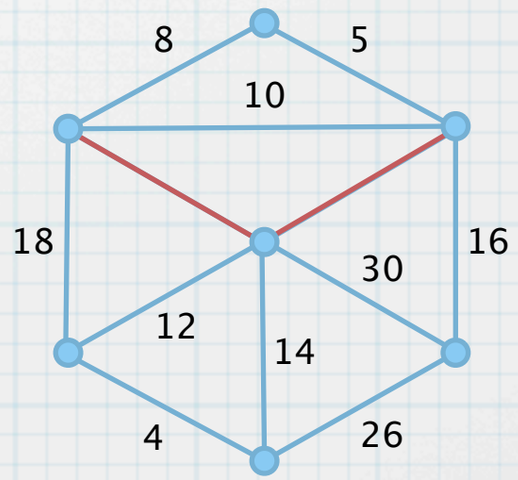
$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

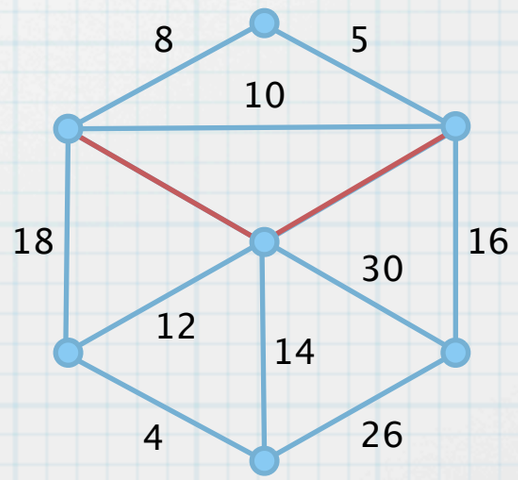
$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

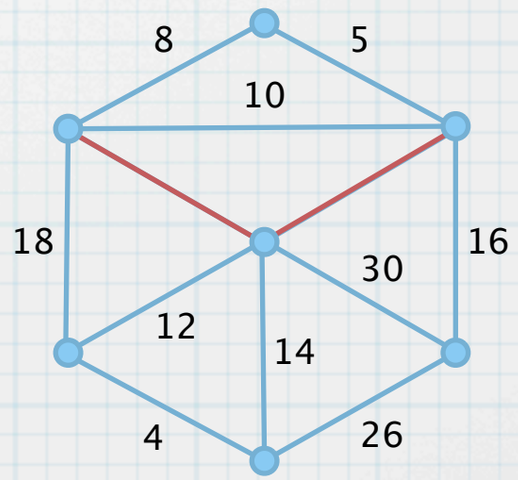
$1 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

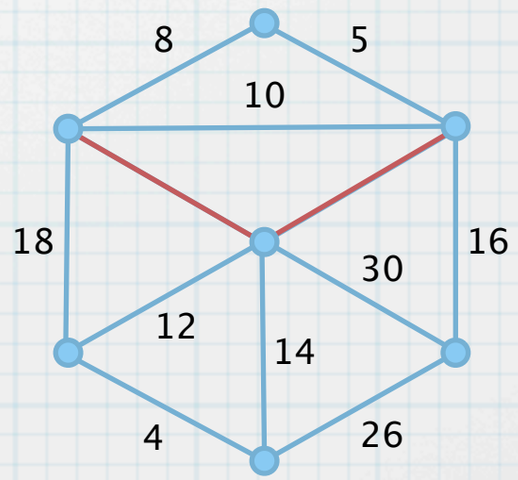
$2 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

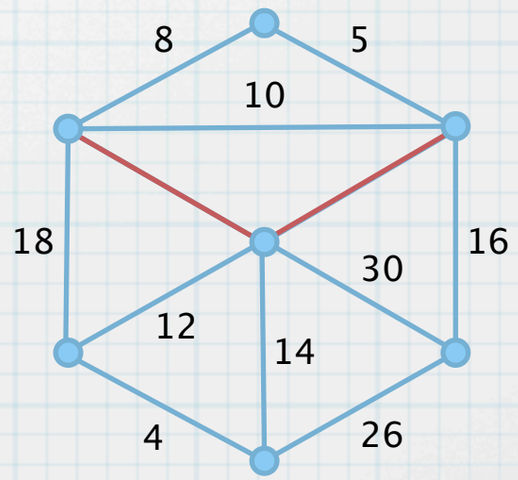
$2 < 6$

$e = 3$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

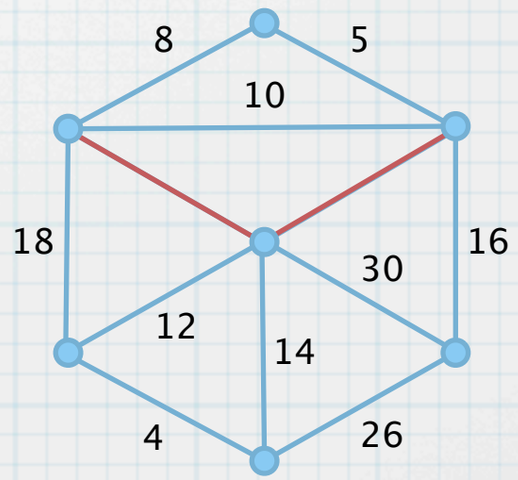
$2 < 6$

$e = 4$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

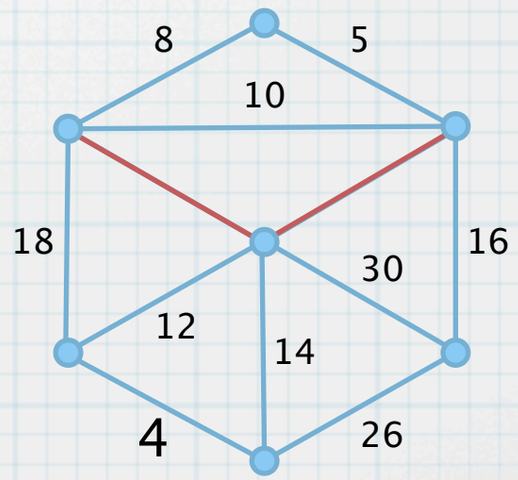
$2 < 6$

$e = 4$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

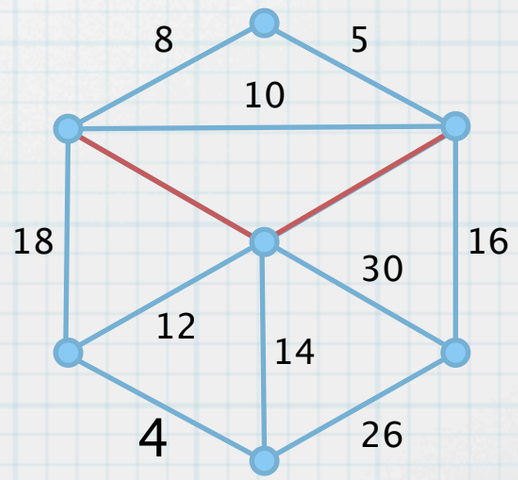
$2 < 6$

$e = 4$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

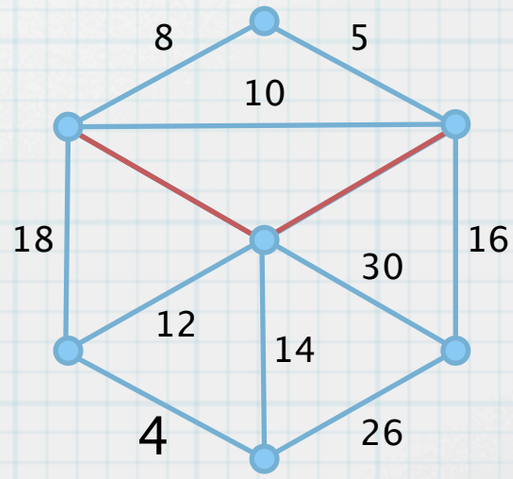
$2 < 6$

$e = 4$

$E(T) := \{2, 3\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

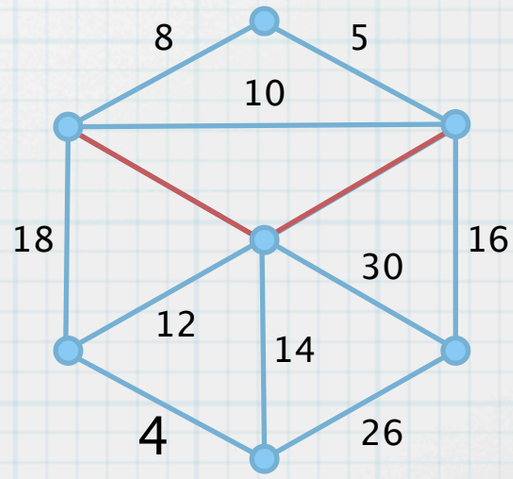
* Beispiel:

$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

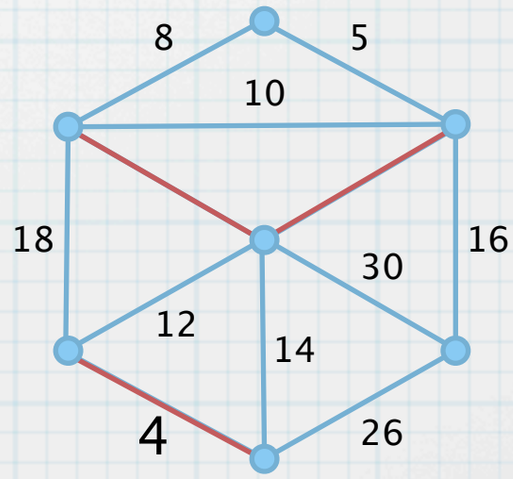
* Beispiel:

$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

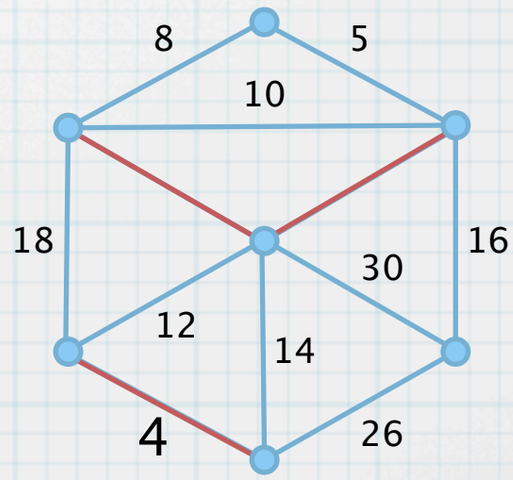
$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

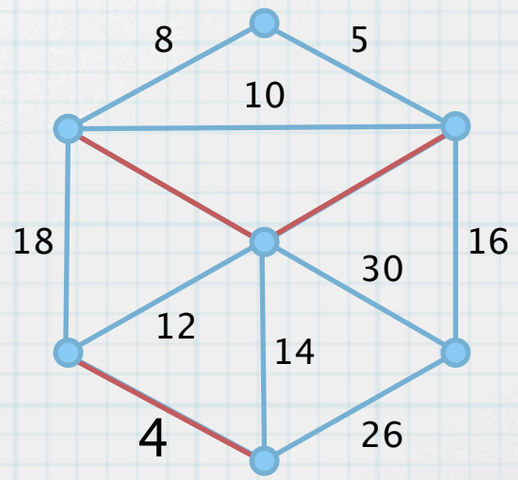
$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

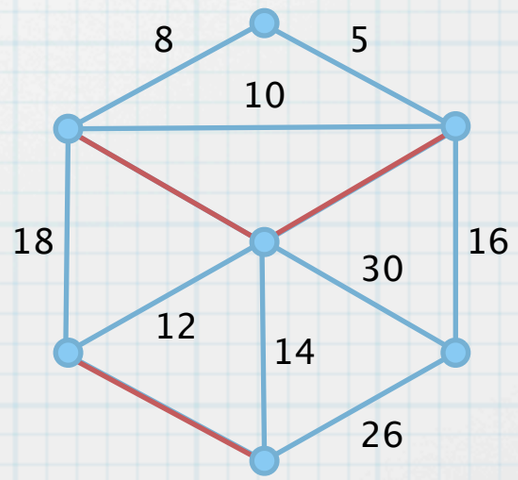
$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (4, 5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

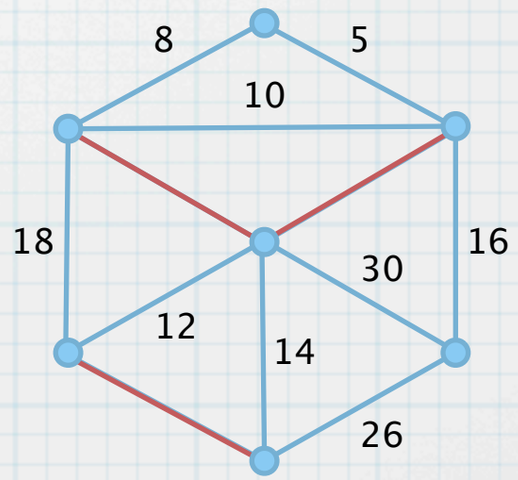
$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

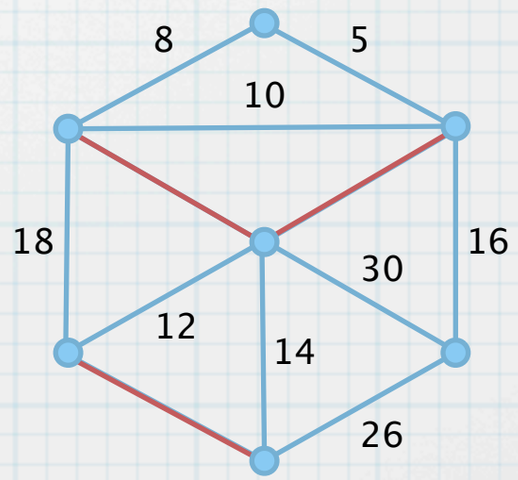
$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

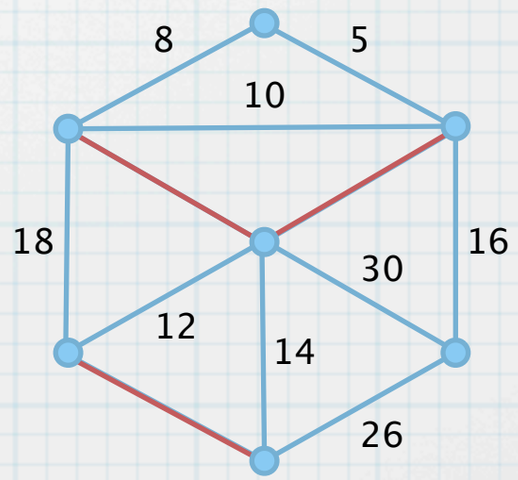
$2 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

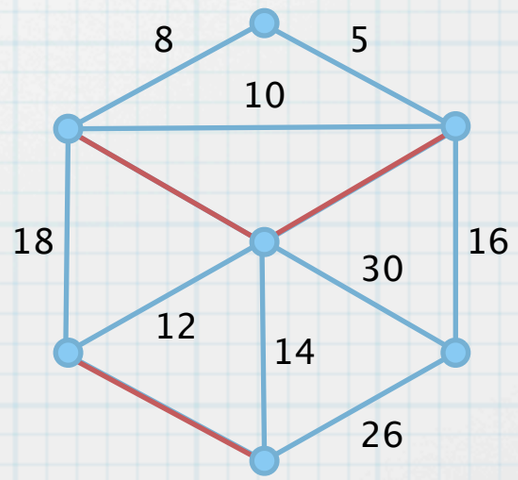
$3 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

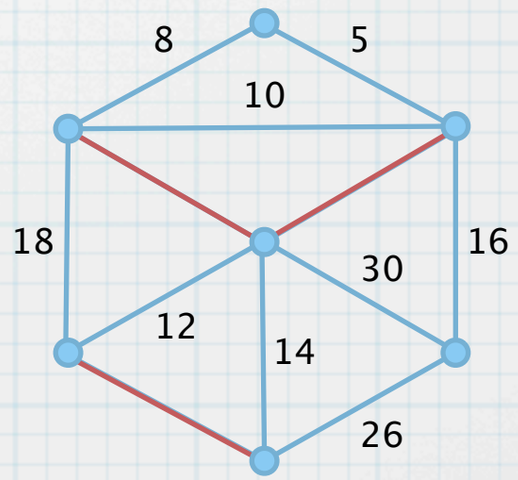
$3 < 6$

$e = 4$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

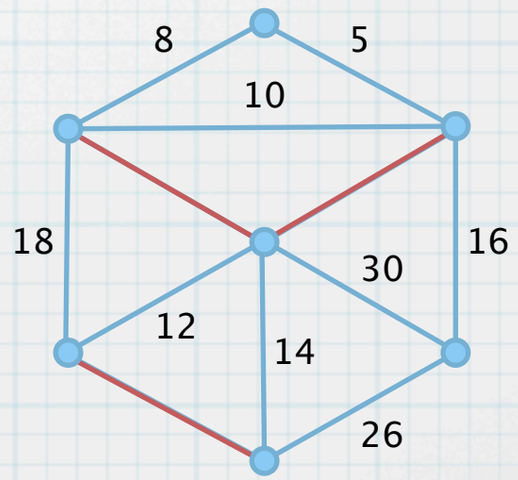
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

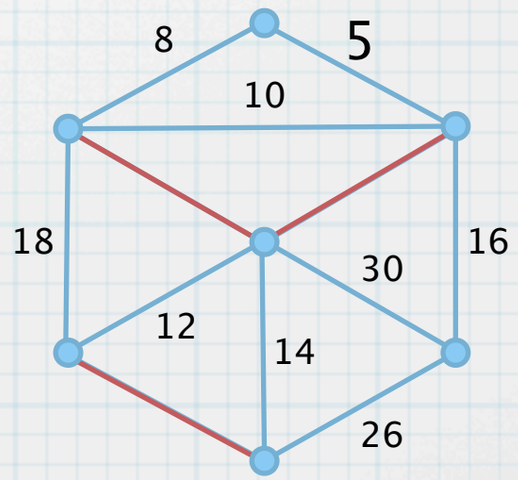
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

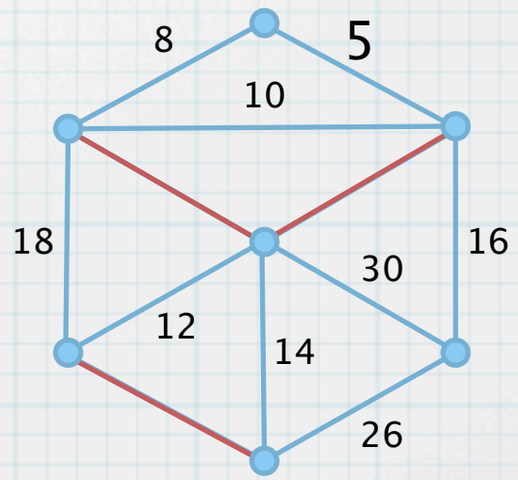
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

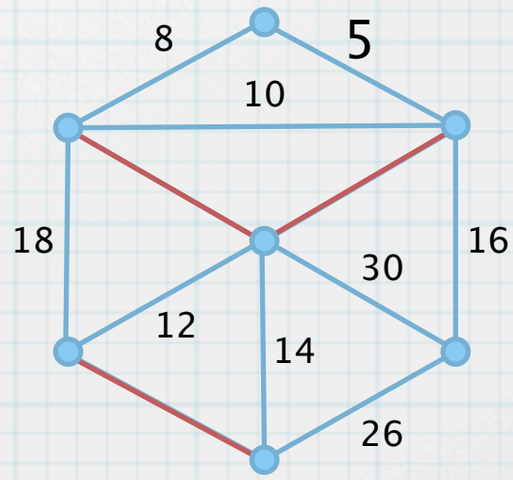
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

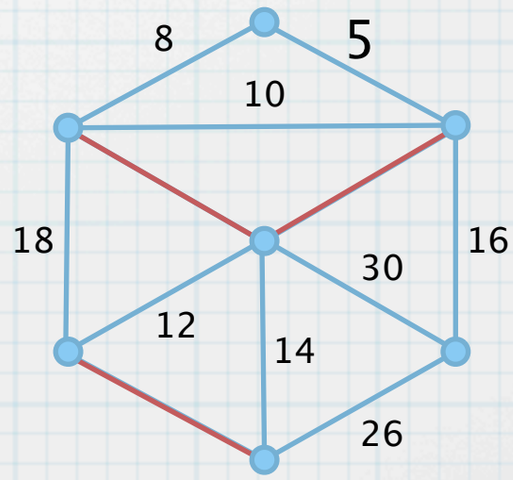
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

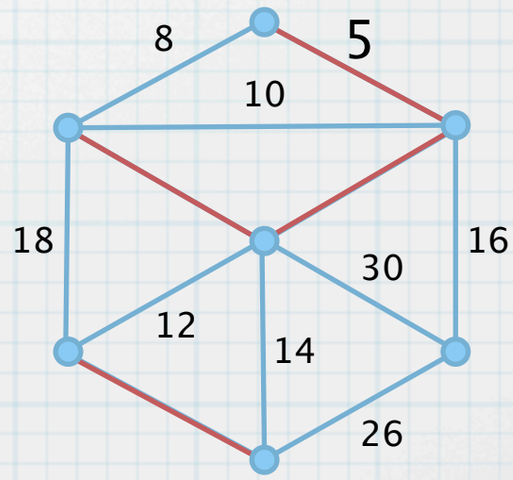
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

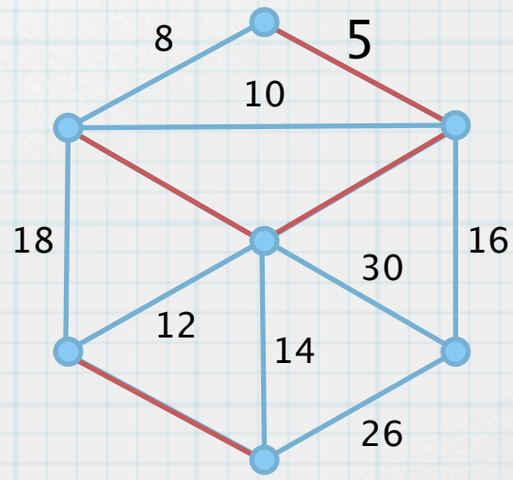
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

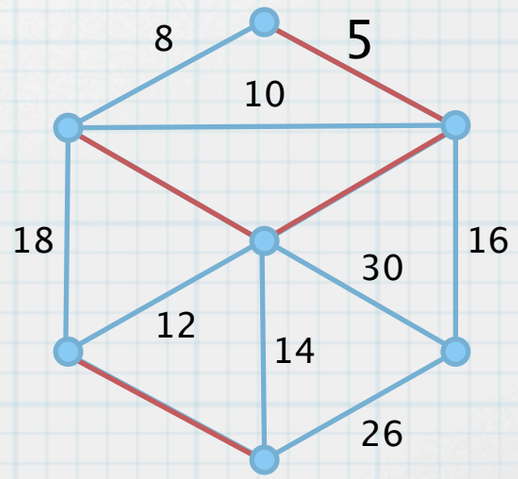
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

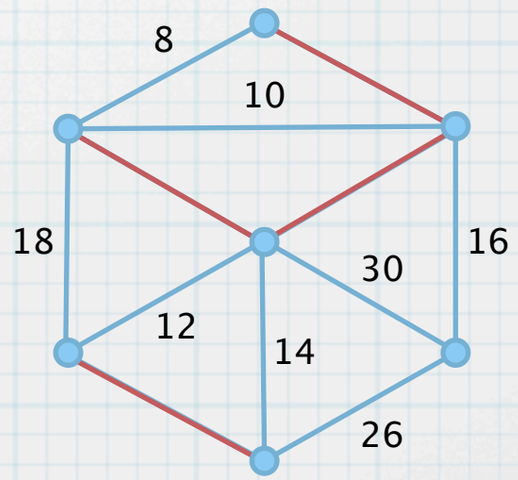
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (5, 8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

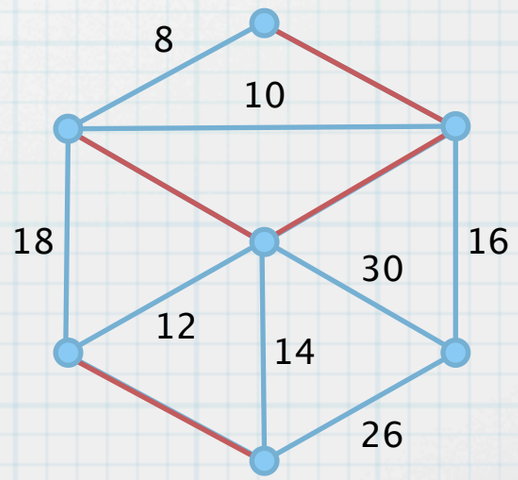
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

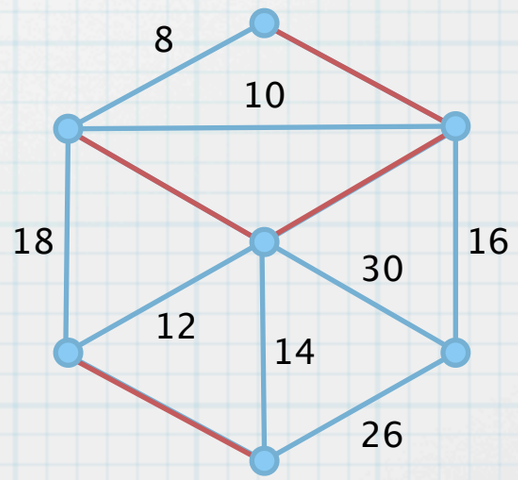
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

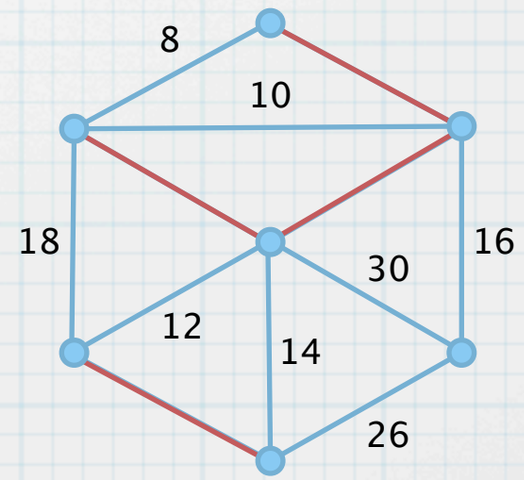
$3 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

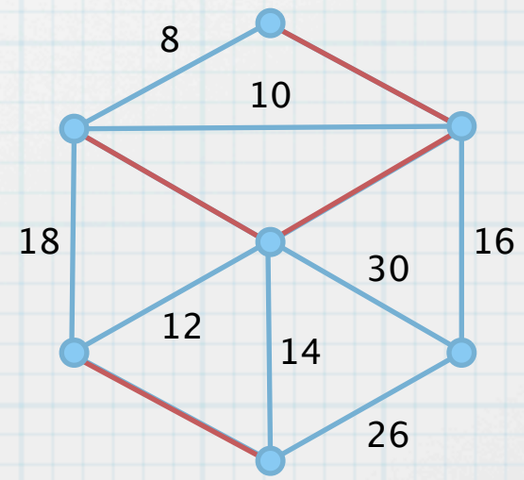
$4 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

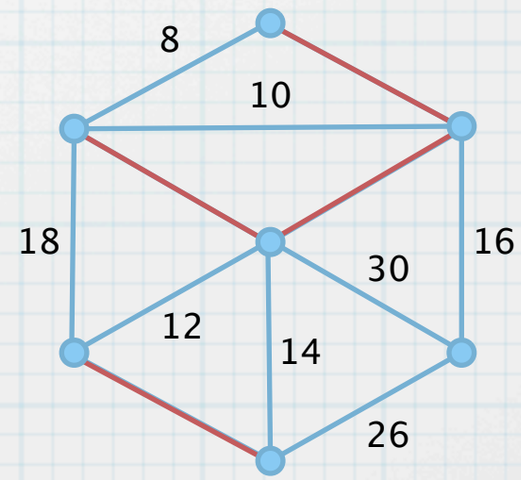
$4 < 6$

$e = 5$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

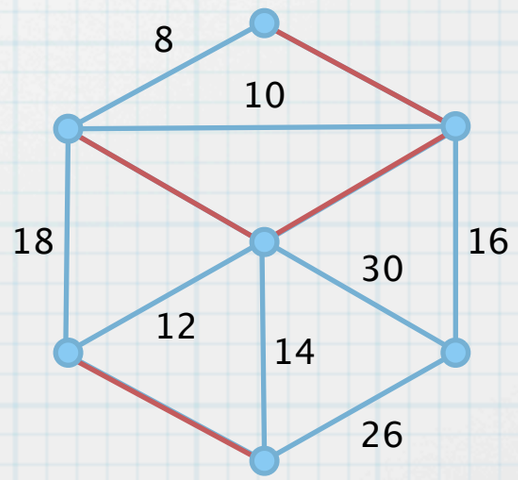
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

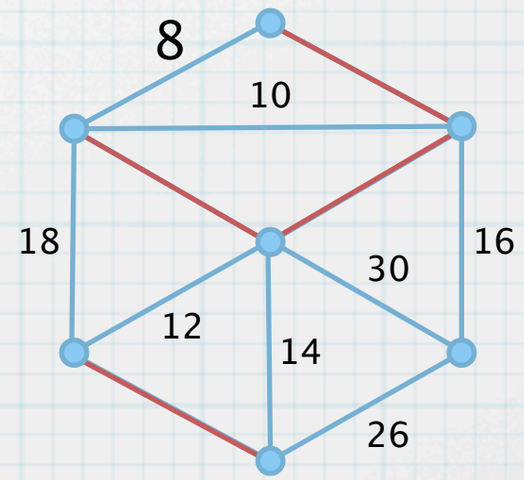
* Beispiel:

$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

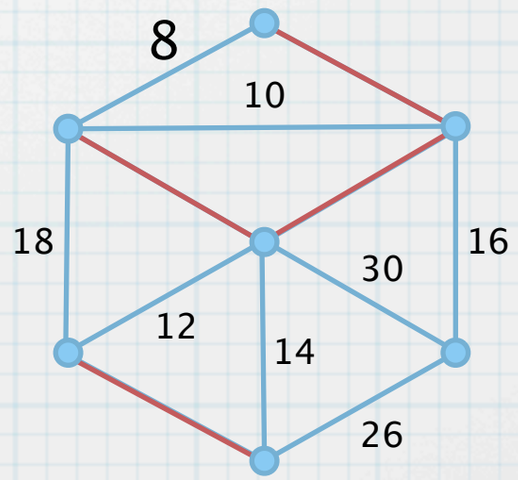
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

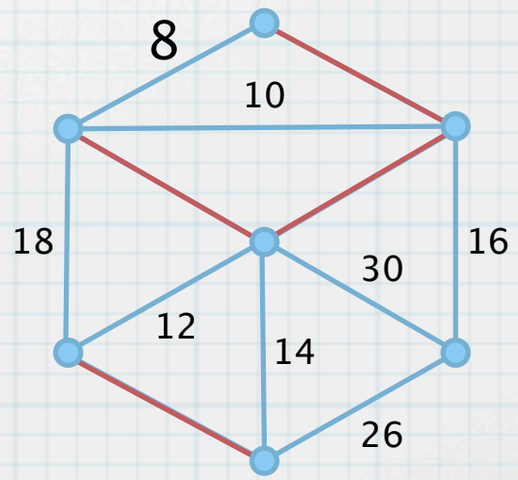
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

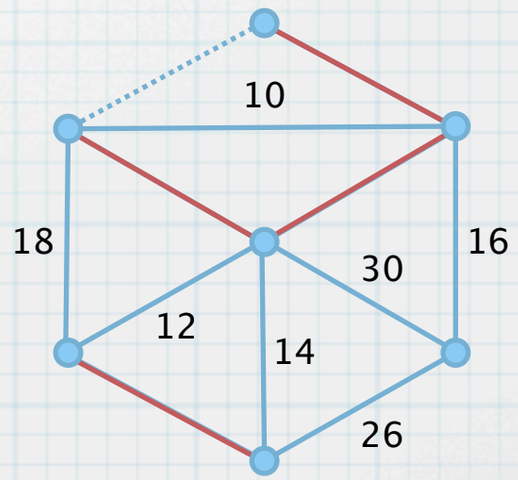
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (8, 10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

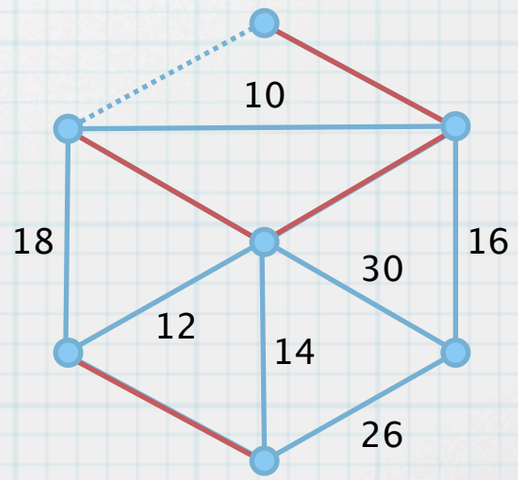
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

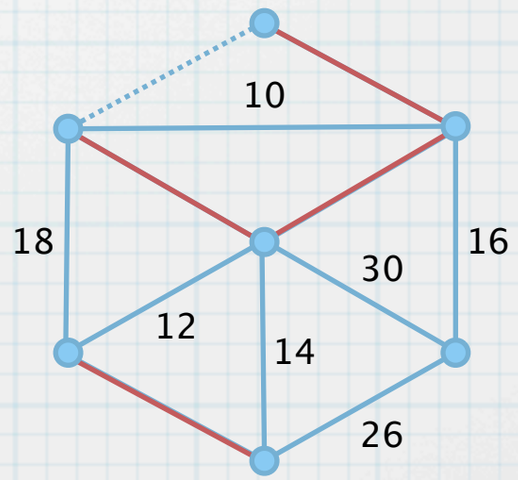
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

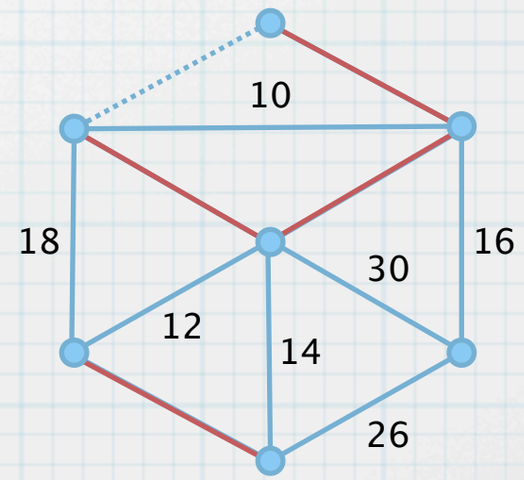
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

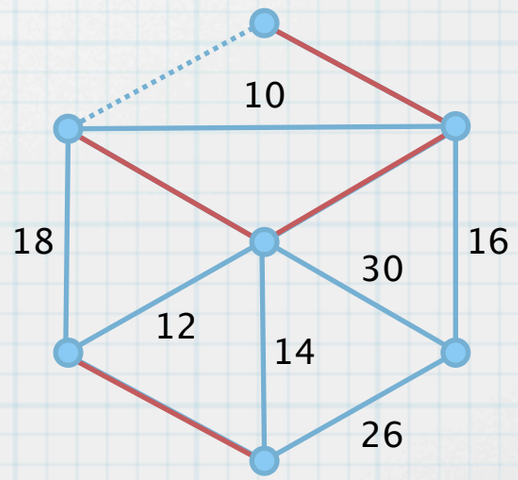
$4 < 6$

$e = 8$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

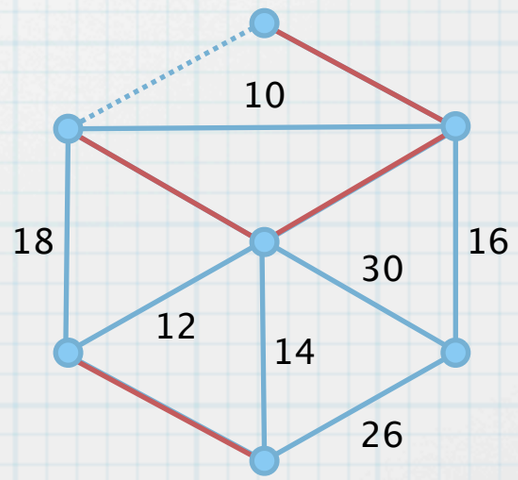
$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

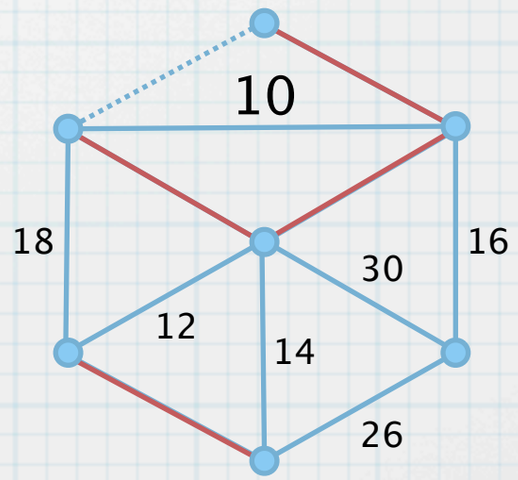
$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

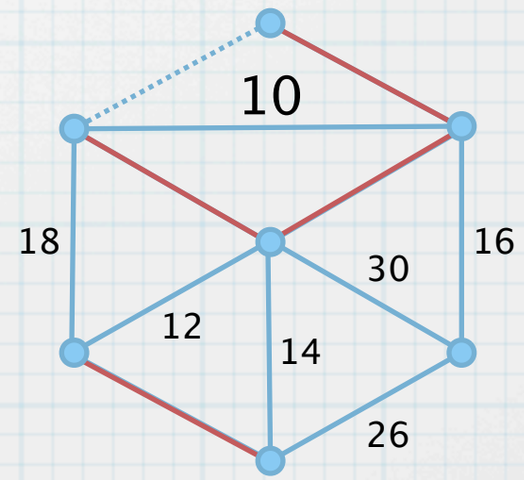
* Beispiel:

$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

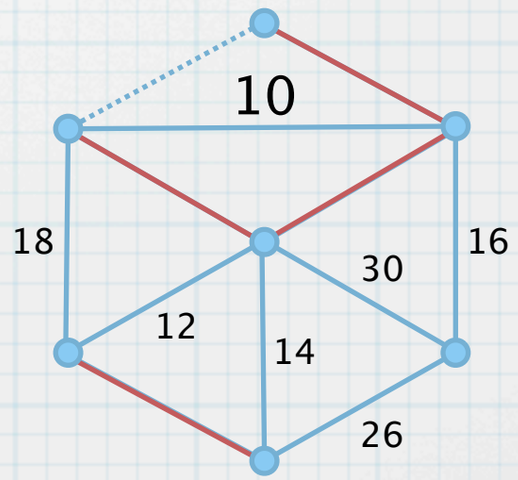
$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

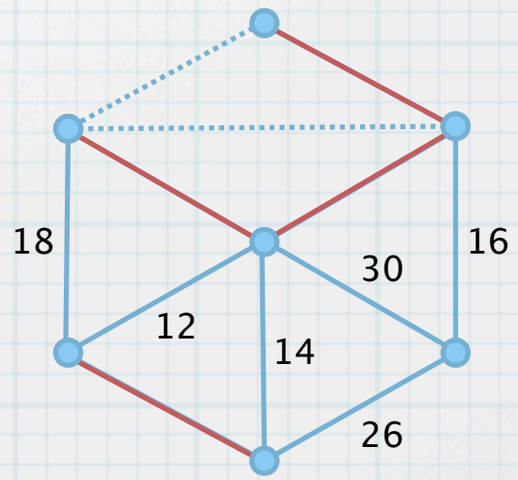
$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (10, 12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

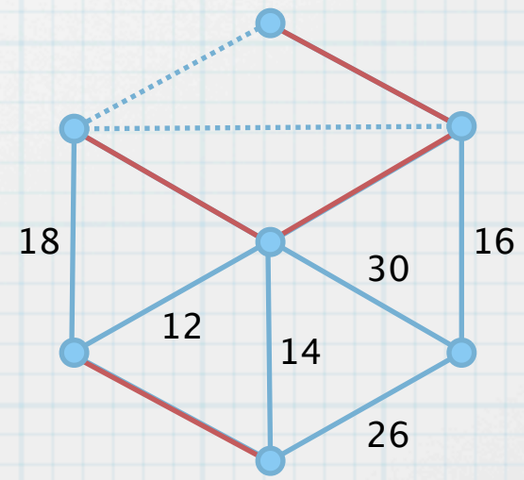
* Beispiel:

$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

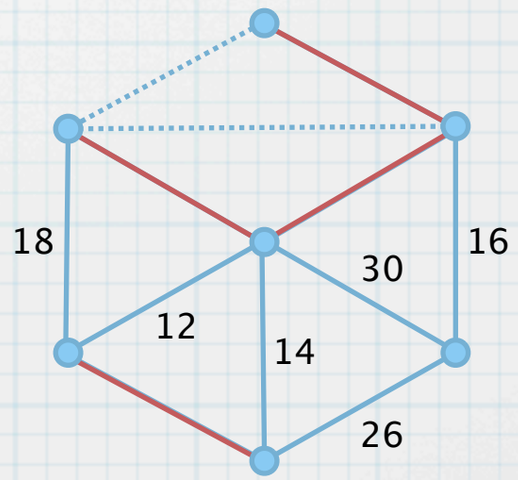
$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

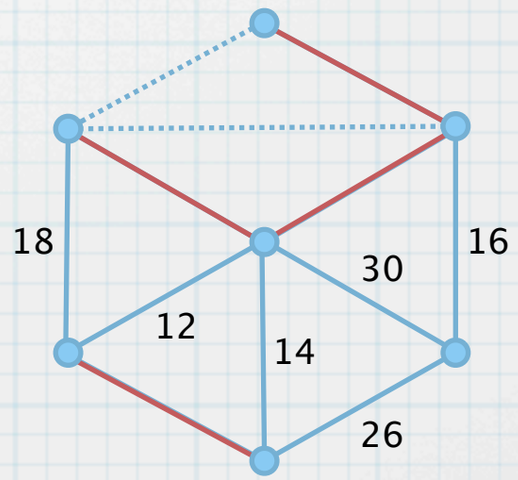
$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

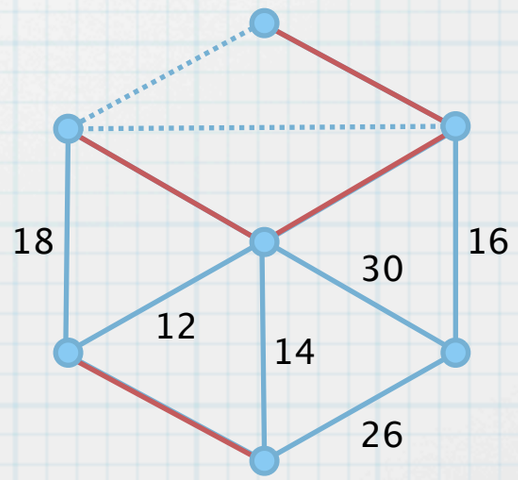
$4 < 6$

$e = 10$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

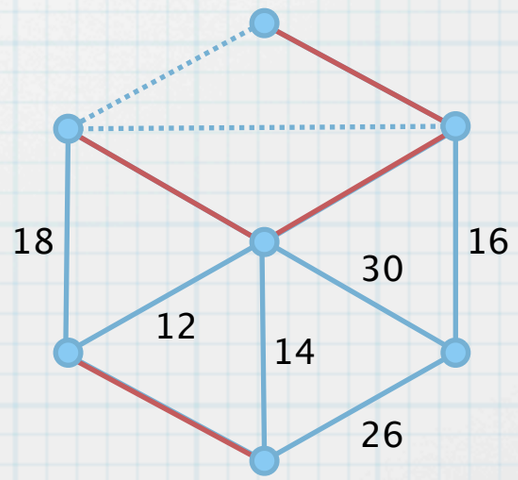
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

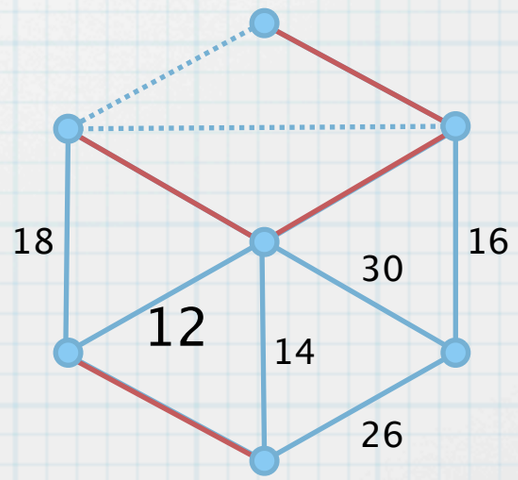
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

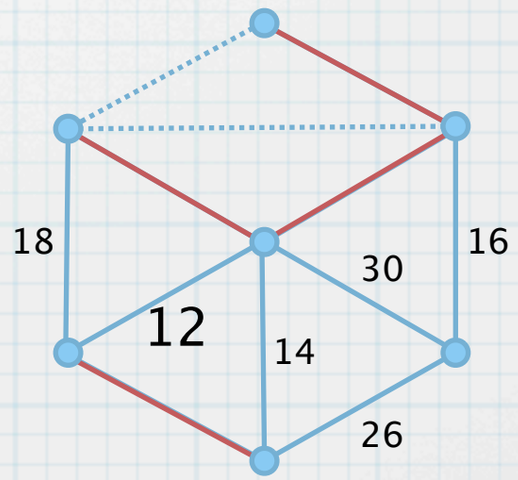
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

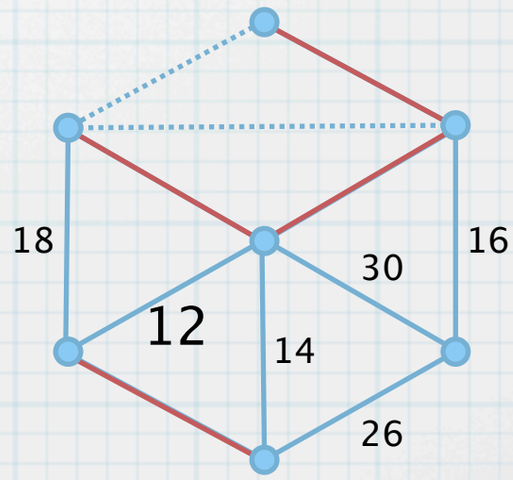
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

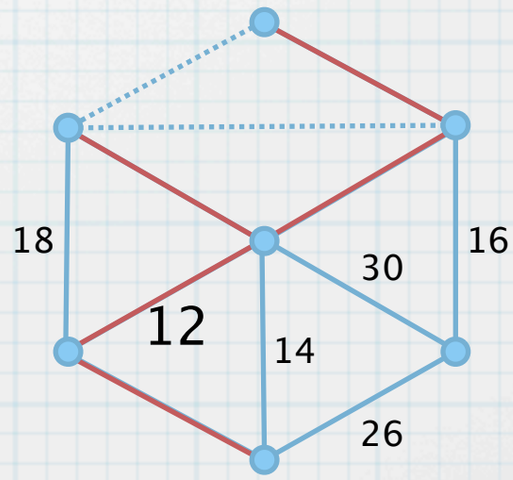
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

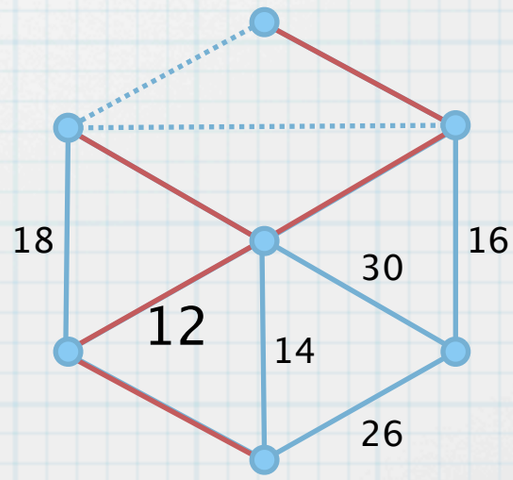
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

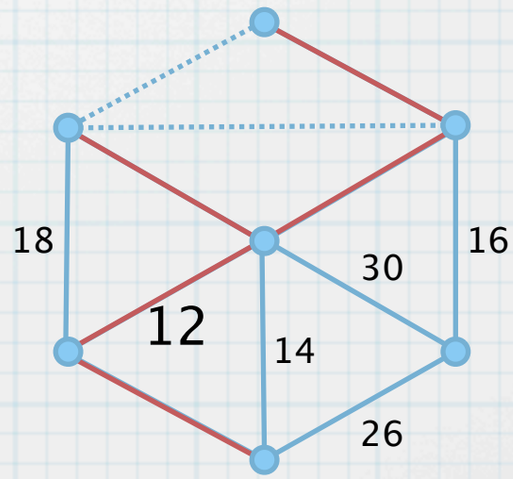
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

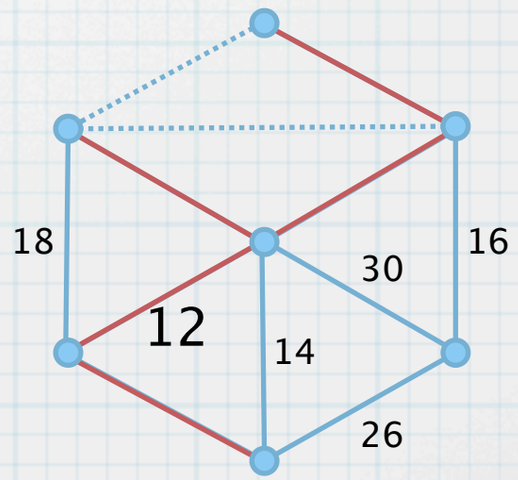
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

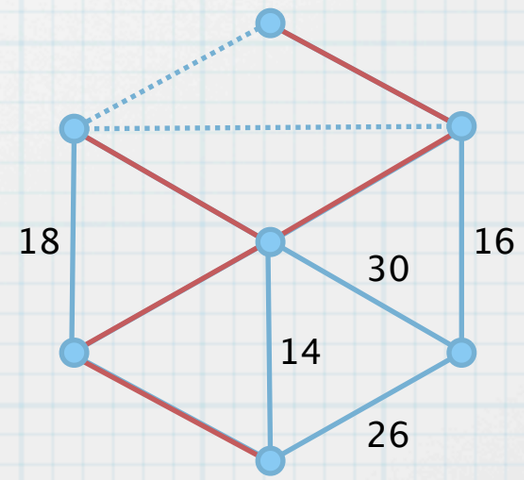
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (12, 14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

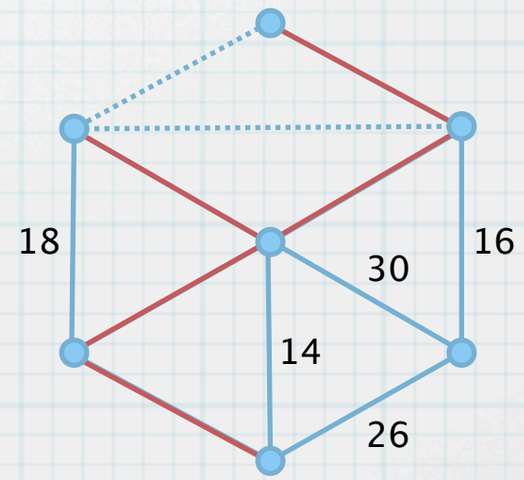
* Beispiel:

$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

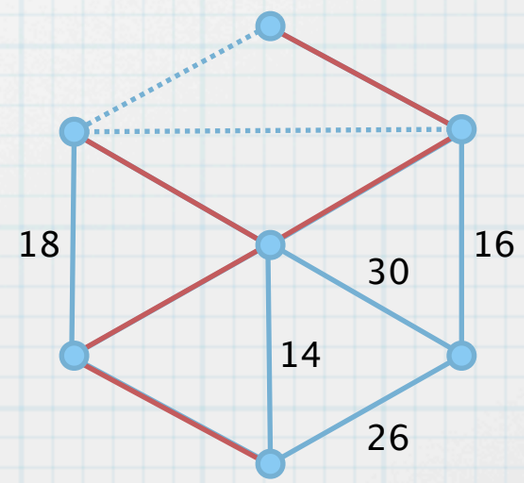
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

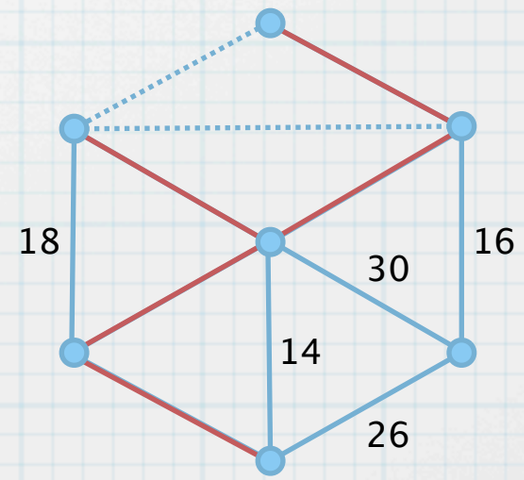
$4 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

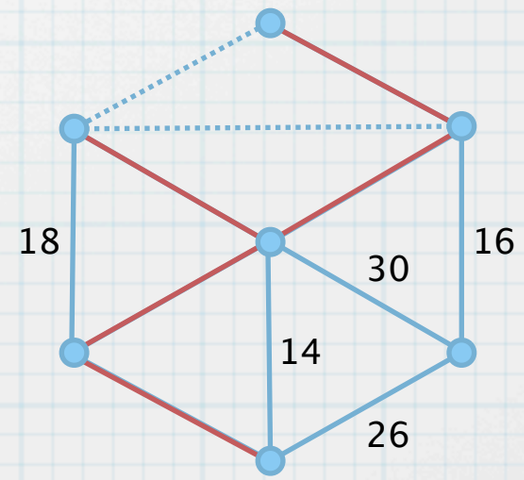
$5 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

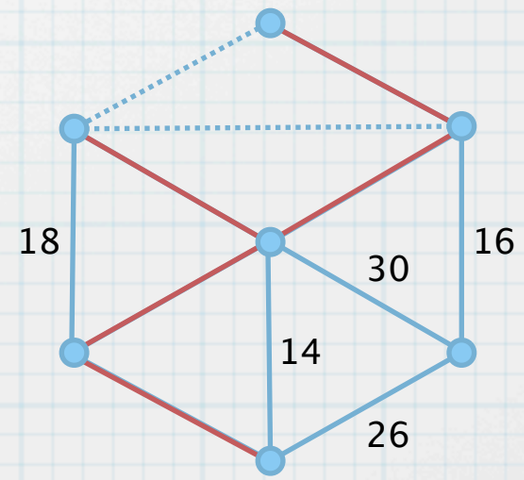
$5 < 6$

$e = 12$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

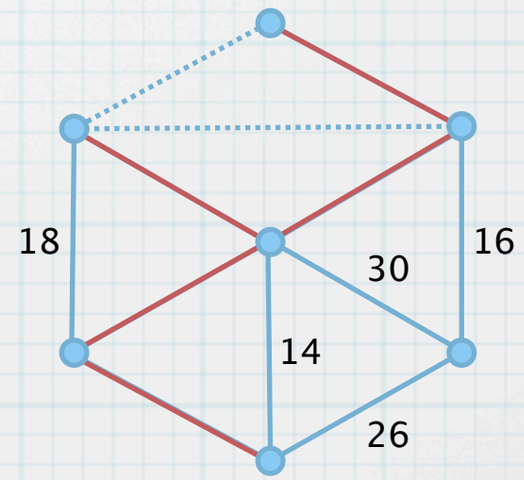
$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

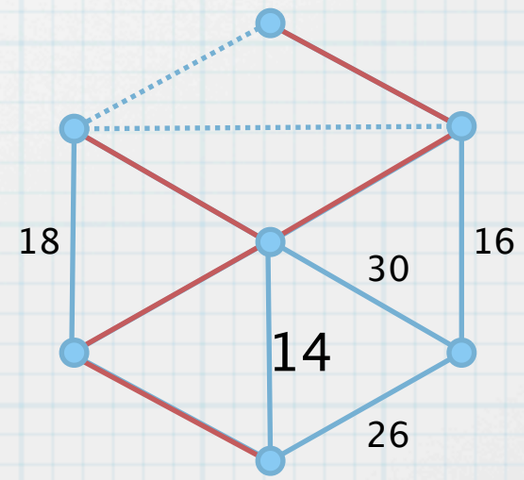
$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

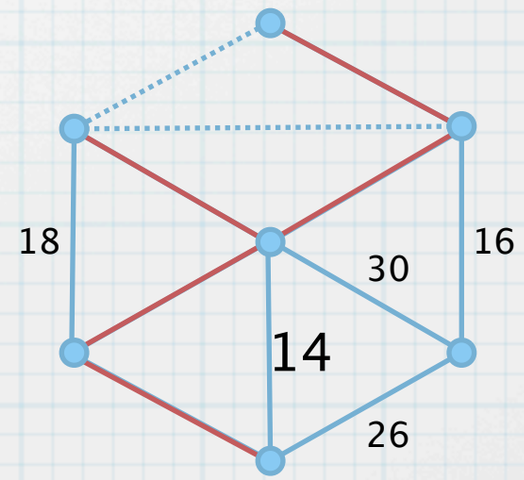
* Beispiel:

$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

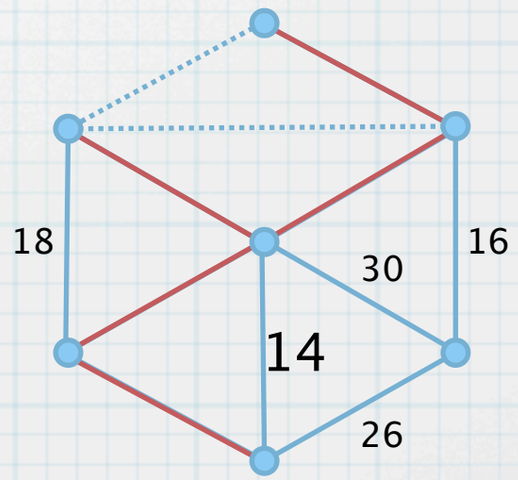
$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (14, 16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

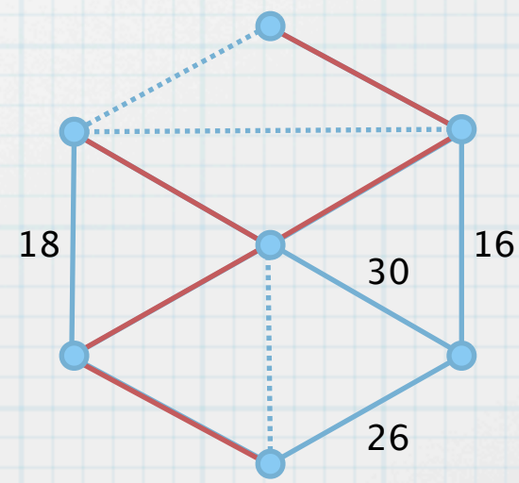
$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

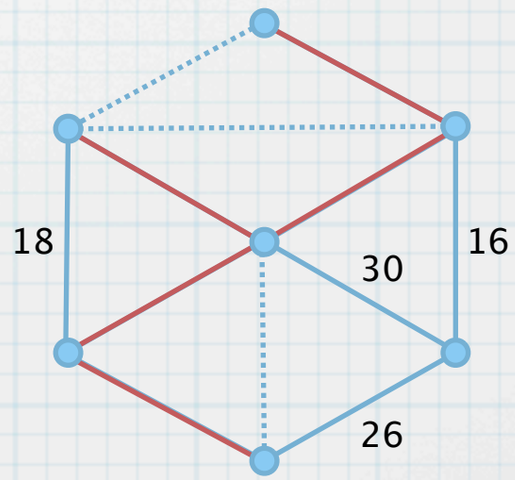
$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

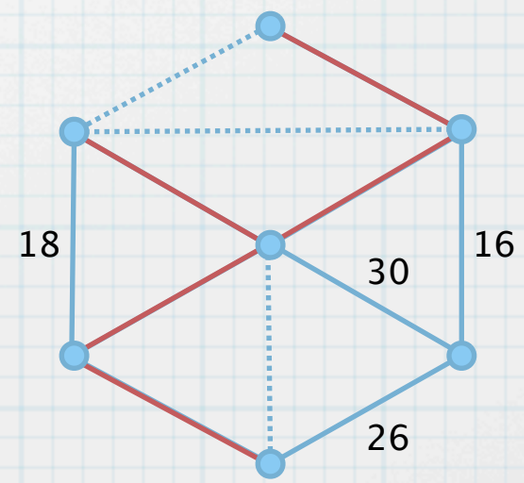
$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

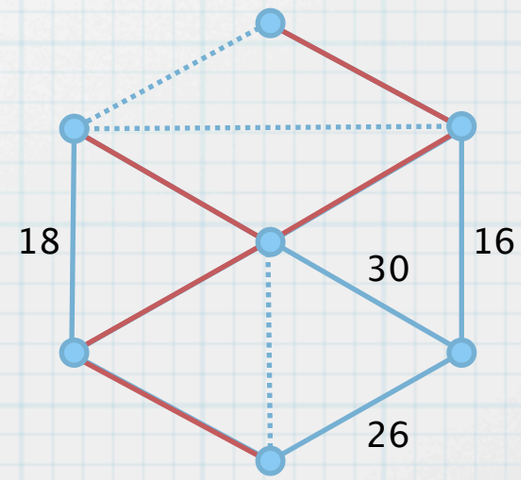
$5 < 6$

$e = 14$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

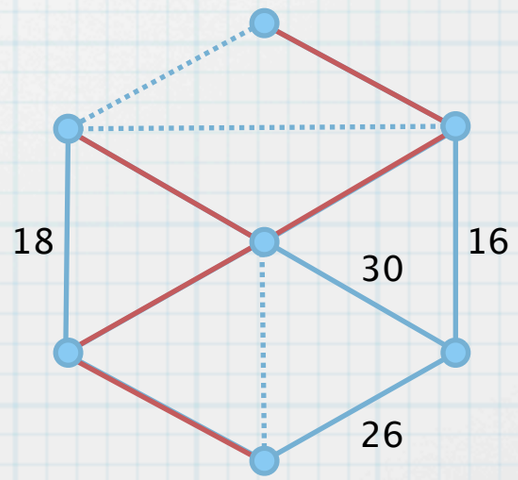
$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

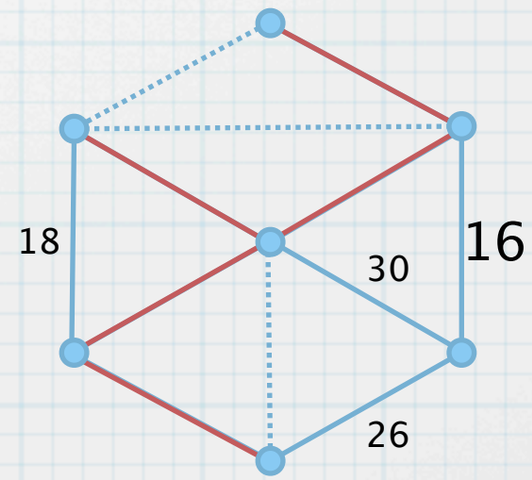
$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

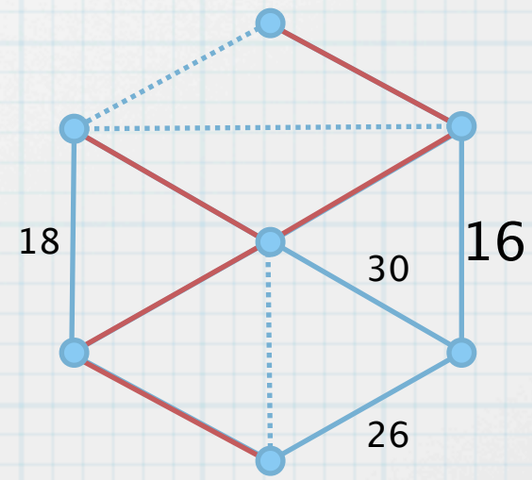
* Beispiel:

$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

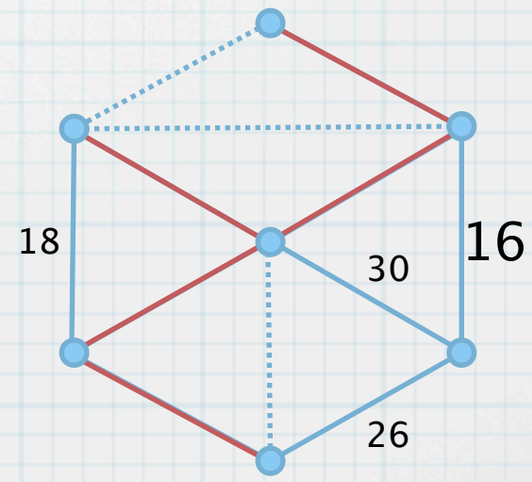
$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

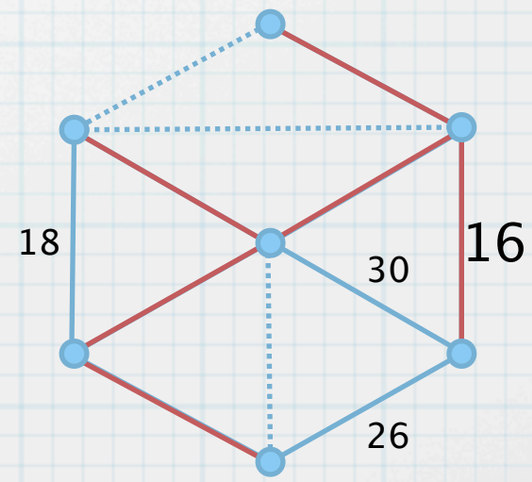
$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

(1) **algorithm** kruskal
(2) sortiere Kanten in E nach Gewicht
(3) $E(T) := \emptyset$
(4) **while** $|E(T)| < |V| - 1$ **do**
(5) wähle $e \in E$ mit kleinstem Gewicht
(6) **if** $E(T) \cup \{e\}$ kreisfrei **then**
(7) $E(T) := E(T) \cup \{e\}$
(8) **end if**
(9) $E := E \backslash \{e\}$
(10) **end while**
(11) **end algorithm**

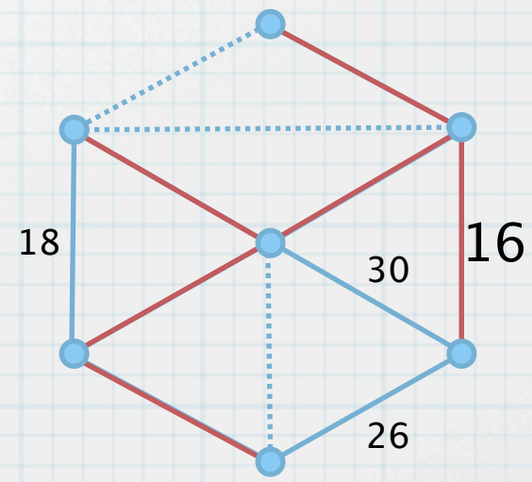
* Beispiel:

$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

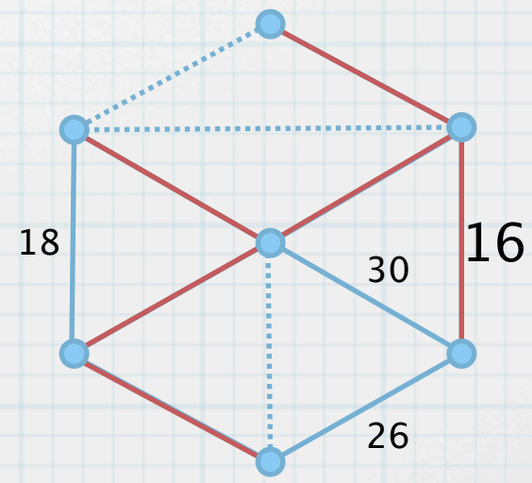
$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (16, 18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

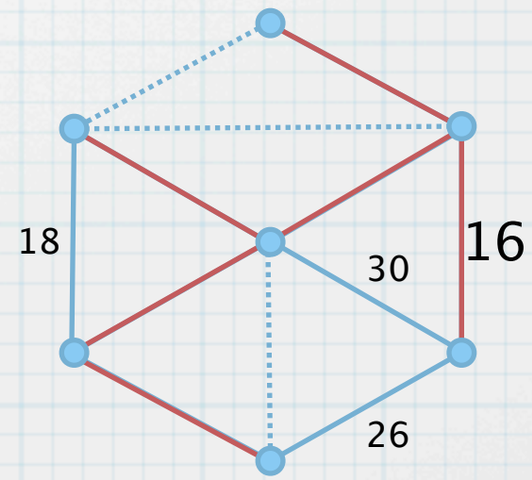
* Beispiel:

$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

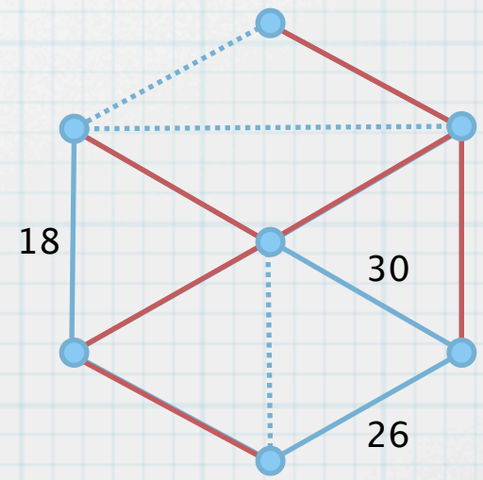
* Beispiel:

$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (16, 18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

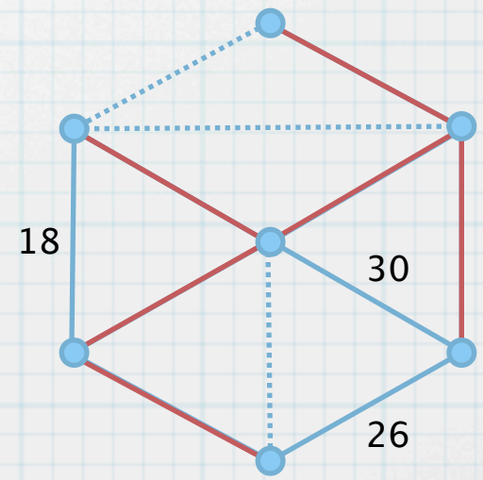
* Beispiel:

$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (18, 26, 30)$



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

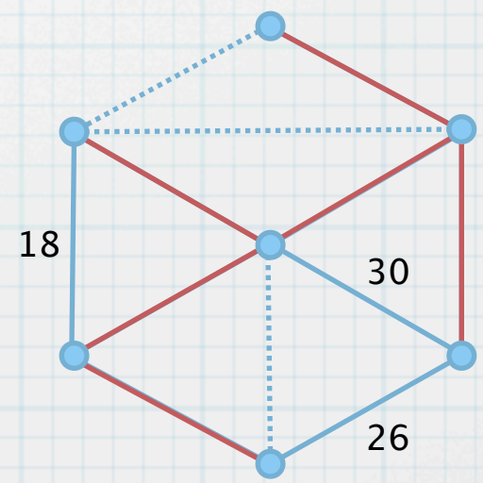
$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

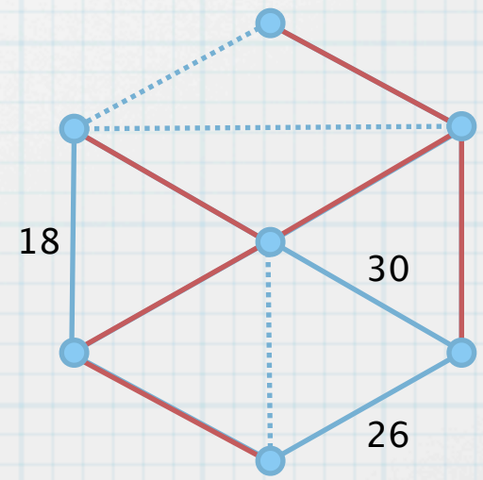
$5 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

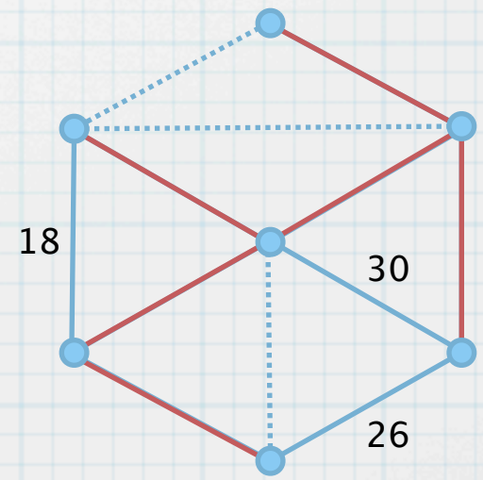
$6 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (18, 26, 30)$

* Beispiel:



Der Algorithmus von Kruskal

* Eingabe: gewichteter, zusammenhängender Graph $G = (V, E, w)$
* Ausgabe: minimaler Spannbaum T von G

```
(1)  algorithm kruskal
(2)      sortiere Kanten in E nach Gewicht
(3)      E(T) := ∅
(4)      while |E(T)| < |V| − 1 do
(5)          wähle e ∈ E mit kleinstem Gewicht
(6)          if E(T) ∪ {e} kreisfrei then
(7)              E(T) := E(T) ∪ {e}
(8)          end if
(9)          E := E\{e}
(10)     end while
(11) end algorithm
```

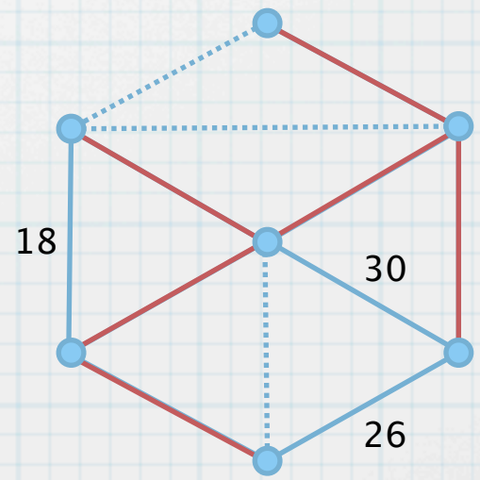
$6 < 6$

$e = 16$

$E(T) := \{2, 3, 4, 5, 12, 16\}$

$E = (18, 26, 30)$

* Beispiel:



Kruskals Korrektheit und Laufzeit

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.
* Beweis:

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die mit den vorhandenen keinen Kreis bildet.

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die mit den vorhandenen keinen Kreis bildet.

  Also erfüllt die so konstruierte Kantenmenge die Pfadbedingung, und ist daher ein minimaler Spannbaum.

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die mit den vorhandenen keinen Kreis bildet.

  Also erfüllt die so konstruierte Kantenmenge die Pfadbedingung, und ist daher ein minimaler Spannbaum.

  Die Sortierung der Kanten nach Gewicht ist in $O(|E| \log |E|)$.

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die mit den vorhandenen keinen Kreis bildet.

  Also erfüllt die so konstruierte Kantenmenge die Pfadbedingung, und ist daher ein minimaler Spannbaum.

  Die Sortierung der Kanten nach Gewicht ist in $O(|E| \log |E|)$.

  Mittels Tiefen- oder Breitensuche kann ein Kreis in einem Graphen mit höchstens $|V|$ Kanten in $O(|V|)$ gefunden werden.

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die mit den vorhandenen keinen Kreis bildet.

  Also erfüllt die so konstruierte Kantenmenge die Pfadbedingung, und ist daher ein minimaler Spannbaum.

  Die Sortierung der Kanten nach Gewicht ist in $O(|E| \log |E|)$.

  Mittels Tiefen- oder Breitensuche kann ein Kreis in einem Graphen mit höchstens $|V|$ Kanten in $O(|V|)$ gefunden werden.

  Dieses wird höchstens $|E|$ mal wiederholt.

Kruskals Korrektheit und Laufzeit

* **Satz 21**:
  Der Algorithmus von Kruskal bestimmt einen minimalen aufspannenden Baum mit Komplexität $O(|V| \cdot |E|)$.
* Beweis:

  In jedem Schritt wird eine Kante geringsten Gewichts gewählt wird, die mit den vorhandenen keinen Kreis bildet.

  Also erfüllt die so konstruierte Kantenmenge die Pfadbedingung, und ist daher ein minimaler Spannbaum.

  Die Sortierung der Kanten nach Gewicht ist in $O(|E| \log |E|)$.

  Mittels Tiefen- oder Breitensuche kann ein Kreis in einem Graphen mit höchstens $|V|$ Kanten in $O(|V|)$ gefunden werden.

  Dieses wird höchstens $|E|$ mal wiederholt.

  Also ist die Komplexität $O(|E| \log |E| + |V| \cdot |E|) = O(|V| \cdot |E|)$.

Literaturquellen

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 2, Seite 9–18)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 2, Seite 9–18)
* T.H. Cormen, C.E. Leiserson, **Introduction to Algorithms**, 2nd Edition, McGraw–Hill Book Company, Boston, 2001. (Kapitel 23, Seite 561–579)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 2, Seite 9–18)
* T.H. Cormen, C.E. Leiserson, **Introduction to Algorithms**, 2nd Edition, McGraw–Hill Book Company, Boston, 2001. (Kapitel 23, Seite 561–579)
* J. Erickson, **Algorithms**, Lecture Notes, University of Illinois at Urbana–Champaign, 2007. (Kapitel 12)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 2, Seite 9–18)
* T.H. Cormen, C.E. Leiserson, **Introduction to Algorithms**, 2nd Edition, McGraw–Hill Book Company, Boston, 2001. (Kapitel 23, Seite 561–579)
* J. Erickson, **Algorithms**, Lecture Notes, University of Illinois at Urbana–Champaign, 2007. (Kapitel 12)
* D. Jungnickel: Graphen, **Netzwerke und Algorithmen**, BI Wissenschaftsverlag, Mannheim, 1994. (Kapitel 4, Seite 133–168)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 2, Seite 9–18)
* T.H. Cormen, C.E. Leiserson, **Introduction to Algorithms**, 2nd Edition, McGraw–Hill Book Company, Boston, 2001. (Kapitel 23, Seite 561–579)
* J. Erickson, **Algorithms**, Lecture Notes, University of Illinois at Urbana–Champaign, 2007. (Kapitel 12)
* D. Jungnickel: Graphen, **Netzwerke und Algorithmen**, BI Wissenschaftsverlag, Mannheim, 1994. (Kapitel 4, Seite 133–168)
* B. Korte, J. Vygen: **Combinatorial Optimization – Theory and Algorithms**, 2nd Edition, Springer Verlag, Berlin, 2001. (Kapitel 6, Seite 117–137)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 2, Seite 9–18)
* T.H. Cormen, C.E. Leiserson, **Introduction to Algorithms**, 2nd Edition, McGraw–Hill Book Company, Boston, 2001. (Kapitel 23, Seite 561–579)
* J. Erickson, **Algorithms**, Lecture Notes, University of Illinois at Urbana–Champaign, 2007. (Kapitel 12)
* D. Jungnickel: Graphen, **Netzwerke und Algorithmen**, BI Wissenschaftsverlag, Mannheim, 1994. (Kapitel 4, Seite 133–168)
* B. Korte, J. Vygen: **Combinatorial Optimization – Theory and Algorithms**, 2nd Edition, Springer Verlag, Berlin, 2001. (Kapitel 6, Seite 117–137)
* L. Lovasz, J. Pelikan, K. Vesztergombi, **Discrete Mathematics – Elementary and Beyond**, Springer Verlag, New York, 2003. (Kapitel 8&9, Seite 141–164)

Literaturquellen

* R.K. Ahuja, T.L. Magnanti, J.B. Orlin, **Network Flows – Theory, Algorithms, and Applications**, Prentice Hall, Upper Saddle River, 1993. (Kapitel 13, Seite 510–542)
* J. Clark, D.A. Holton, **Graphentheorie**, Spektrum Akademischer Verlag, Heidelberg, 1994. (Kapitel 2, Seite 51–92)
* W.J. Cook, W.H. Cunningham, W.R. Pulleyblank, A. Schrijver, **Combinatorial Optimization**, Wiley Interscience, New York, 1998. (Kapitel 2, Seite 9–18)
* T.H. Cormen, C.E. Leiserson, **Introduction to Algorithms**, 2nd Edition, McGraw–Hill Book Company, Boston, 2001. (Kapitel 23, Seite 561–579)
* J. Erickson, **Algorithms**, Lecture Notes, University of Illinois at Urbana–Champaign, 2007. (Kapitel 12)
* D. Jungnickel: Graphen, **Netzwerke und Algorithmen**, BI Wissenschaftsverlag, Mannheim, 1994. (Kapitel 4, Seite 133–168)
* B. Korte, J. Vygen: **Combinatorial Optimization – Theory and Algorithms**, 2nd Edition, Springer Verlag, Berlin, 2001. (Kapitel 6, Seite 117–137)
* L. Lovasz, J. Pelikan, K. Vesztergombi, **Discrete Mathematics – Elementary and Beyond**, Springer Verlag, New York, 2003. (Kapitel 8&9, Seite 141–164)
* J. Matousek, J. Nesetril, **Invitation to Discrete Mathematics**, Clarendon Press, Oxford, 1999. (Kapitel 4, Seite 138–166)