Zurück zur Breitensuche

Satz

BFS baut einen Breitensuchbaum $G_\pi = (V_\pi, E_\pi)$ *mit*

$$V_\pi = \{v \in V \mid \pi(v) \neq NIL\} \cup \{s\}$$
$$E_\pi = \{\pi(v)v \mid v \in V_\pi \setminus \{s\}\}$$

Beweis: Die Pfade $W(s \to v)$ in G_π sind kürzeste Wege in G, insbesondere gilt jedoch für alle v

$$D(v) = d_G(s, v) = d_{G_\pi}(s, v),$$

also ist G_π zusammenhängend.

Es gilt $|V_\pi| = |E_\pi| + 1$, daher ist G_π ein Baum. □

Ausgabe der kürzesten Wege:

Algorithm PRINT-PATH(G, s, v)

```
1: if v = s then
2:     print s
3: else if π(v) =NIL then
4:     print „Es gibt keinen Pfad von s nach v.“
5: else
6:     PRINT-PATH(G, s, π(v))
7:     print v
8: end if
```

Laufzeit: $\mathcal{O}(|V|)$.

Tiefensuche (Depth-first-search, DFS)

- Untersucht Kanten, die inzident zum letzten Knoten v sind, der noch nicht untersuchte ausgehende Knoten hat.
- Sobald alle mit v inzidenten Kanten untersucht: Suche „kehrt zurück“ und untersucht Kanten des Knotens, von dem aus v entdeckt wurde.
- Fortsetzen, bis alle Knoten der Komponente des Startknotens entdeckt wurden.
- Falls unentdeckte Knoten bleiben: Neuen Startknoten setzen und fortfahren.

Konstruktion eines Vorgängergraph $G_\pi = (V, E_\pi)$ mit $E_\pi = \{(\pi(v), v) | \, v \in V, \pi(v) \neq NIL\}$: DFS-Wald.

Zustände der Knoten wieder mit Farben markieren:

- Weiß: unentdeckt
- Grau: entdeckt
- Schwarz: Adjazenzliste des Knotens vollständig abgearbeitet

Jeder Knoten speichert wieder einige Attribute:

- Farbe $c : V \to \{\text{WEISS, GRAU, SCHWARZ}\}$,
- Zeitpunkt des Grauwerdens $g : V \to \mathbb{N}$,
- Zeitpunkt des Schwarzwerdens $f : V \to \mathbb{N}$.

Bemerkung: $1 \leq g(u) < f(u) \leq 2|V|$.

Algorithm DFS(G)

```
 1: for u ∈ V do
 2:     c(u) :=WEISS
 3:     π(u) :=NIL
 4: end for
 5: t := 0
 6: for u ∈ V do
 7:     if c(u) =WEISS then
 8:         DFS-VISIT(G, u)
 9:     end if
10: end for
```

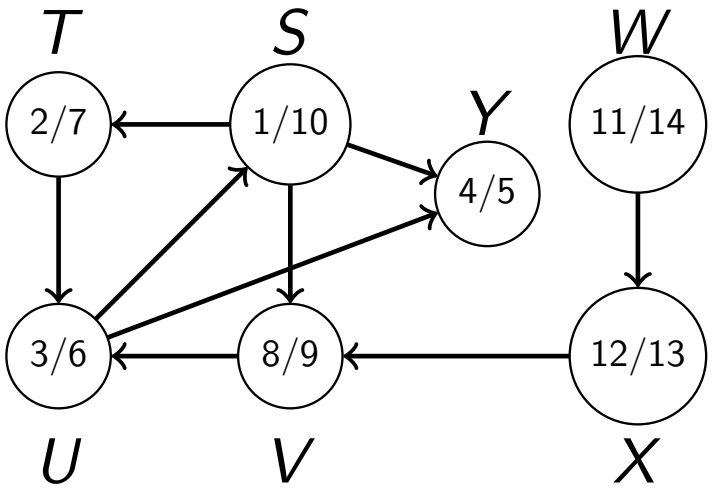
Algorithm DFS-VISIT(G, u)

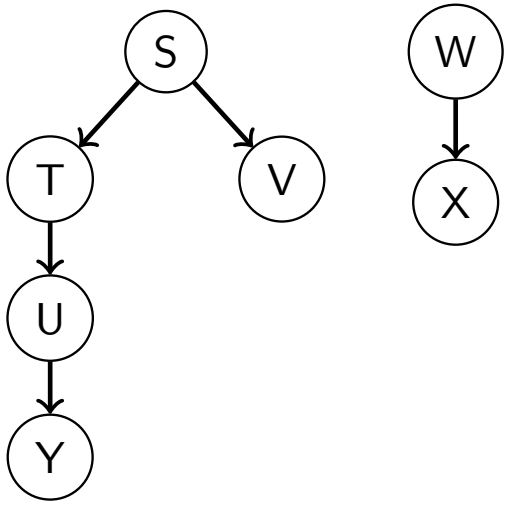
```
 1: t := t+1    % weißer Knoten u gerade entdeckt
 2: g(u) := t
 3: c(u) :=GRAU
 4: for v ∈Adj[u] do    % verfolge die Kante (u, v)
 5:     if c(v) =WEISS then
 6:         π(v) := u
 7:         DFS-VISIT(G, v)
 8:     end if
 9: end for
10: c(u) :=SCHWARZ    % u abgearbeitet
11: t := t + 1
12: f(u) := t
```

Laufzeit von DFS:
$\Theta(|V|) + T(\text{DFS-VISIT})$

DFS-VISIT wird für jeden Knoten v einmal aufgerufen,
Schleife hat Adj[v] Iterationen.
Daraus folgt $T(\text{DFS-VISIT}) = \Theta(|E|)$.

Insgesamt ergibt das für DFS Kosten von $\Theta(|V| + |E|)$.





Klammernstruktur: v wird grau $\rightarrow$ „$(v$“; v wird schwarz $\rightarrow$ „$v)$“.

1	2	3	4	5	6	7	8	9	10	11	12	13	14
(s	(t	(u	(y	y)	u)	t)	(v	v)	s)	(w	(x	x)	w)

Satz (Klammerntheorem)

Bei DFS, *angewendet auf* $G = (V, E)$ *(gerichtet oder ungerichtet), gilt für alle* $u, v \in V$ *genau eine der folgenden Bedingungen:*

- $[g(u), f(u)] \cap [g(v), f(v)] = \varnothing$, *d.h. weder* u *ist Nachfahre von* v *noch umgekehrt.*
- $[g(u), f(u)] \subseteq [g(v), f(v)]$, u *ist Nachfahre von* v.
- $[g(u), f(u)] \supseteq [g(v), f(v)]$, v *ist Nachfahre von* u.

Beweis: O.B.d.A. $g(u) < g(v)$

1. Fall: $g(v) < f(u)$: v entdeckt, während u grau ist
   $\Longrightarrow$ v Nachfahre von u.
   Da v später entdeckt wird als u, werden alle von v ausgehenden Kanten erforscht (v fertig), bevor der Algorithmus zu u zurückkehrt. Folglich gilt $f(v) < f(u)$.
2. Fall: $g(v) > f(u)$: Daraus folgt
   $g(u) < f(u) < g(v) < f(v)$.

□

Folgerung (Satz von der „Intervallschachtelung“)

Wenn v ein echter Nachfahre von u ist, dann gilt $g(u) < g(v) < f(v) < f(u)$.

Satz (Satz der weißen Pfade)

In einem DFS-Wald von $G = (V, E)$ ist v genau dann ein Nachfahre von u, wenn es zum Zeitpunkt $g(u)$ einen Pfad $u \to v$ gibt, der nur aus weißen Knoten besteht.

Beweis: „$\Longrightarrow$“:

1. Fall: $v = u$: Der Weg $u \to v$ besteht nur aus u. u ist weiß, wenn $g(u)$ gesetzt wird.
2. Fall: v ist echter Nachfahre von u: Dann gilt $g(u) < g(v)$.

   Zum Zeitpunkt $g(u)$ ist v daher noch weiß. Das gilt für alle Nachfahren von u auf dem Pfad $u \to v$.

„$\Longleftarrow$“: Zur Zeit $g(u)$ gebe es einen Pfad $u \to v$ aus lauter weißen Knoten.

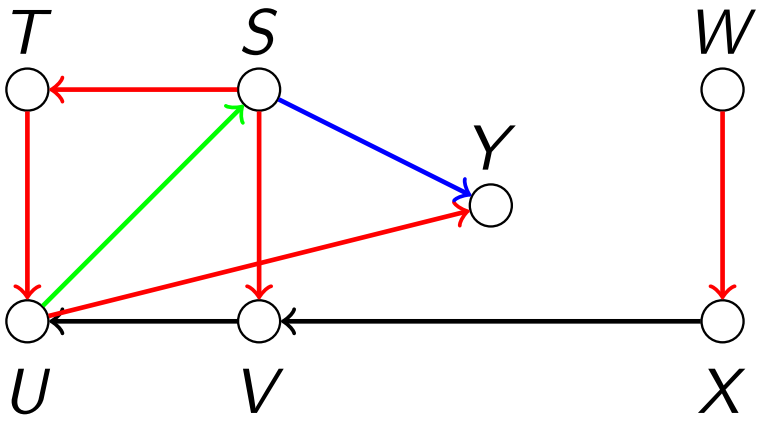
Annahme: Seien alle von v verschiedenen Knoten auf $u \to v$ Nachfahren von u.

Sei w der Vorgänger von v auf $u \to v$. Dann ist w ein Nachfahre von u und somit $f(w) \leq f(u)$.
Also wird v entdeckt, nachdem u entdeckt wird und bevor w fertig ist.
Daher gilt $g(u) < g(v) < f(w) \leq f(u)$.

Aus dem Klammerntheorem folgt schließlich $f(v) < f(u)$ und somit ist v Nachfolger von u □

Klassifikation der Kanten in $G = (V, E)$

- Baumkanten (E_B): Kanten des DFS-Waldes G_π, d.h. $(u, v) \in E_B$, wenn v beim Untersuchen von u entdeckt wird;
- Rückwärtskanten (E_R): Kanten (u, v), die von u zu Vorfahren v im DFS-Wald führen;
- Vorwärtskanten (E_V): Kanten (u, v), die von u zu Nachfahren v im DFS-Wald führen;
- **Querkanten** (E_Q): alle übrigen Kanten.



Während DFS wird (u, v) untersucht:

- Wenn v noch weiß, dann $(u, v) \in E_B$;
- Wenn v bereits grau: graue Knoten bilden Pfade aus lauter Nachfahren eines Knoten v_0 (entspricht dem Stack der Aufrufe von DFS-VISIT). Also $(u, v) \in E_R$;
- Wenn v schwarz, dann $(u, v) \in E_V \cup E_Q$;
  - $(u, v) \in E_V \iff g(u) < g(v)$
  - $(u, v) \in E_Q \iff g(u) > g(v)$

Falls G ungerichtet, wird (u, v) zweimal untersucht und bei der ersten Untersuchung einer Kategorie zugeordnet.

Satz

$G = (V, E)$ ungerichtet, DFS werde angewendet. Dann gibt es nur Baum- und Rückwärtskanten, d.h. $E = E_B \cup E_R$.

Beweis: Sei uv eine beliebige Kante, o.B.d.A. gelte $g(u) < g(v)$.
Daher muss v fertig untersucht werden, bevor u fertig wird, also $f(u) > f(v)$.
Währenddessen gilt: u grau, da $v \in Adj[u]$.

Falls zuerst $uv = (u, v)$ untersucht wird, ist v noch weiß, daher $uv \in E_B$.
Falls zuerst $uv = (v, u)$ untersucht wird, ist u grau und daher $uv \in E_R$. □

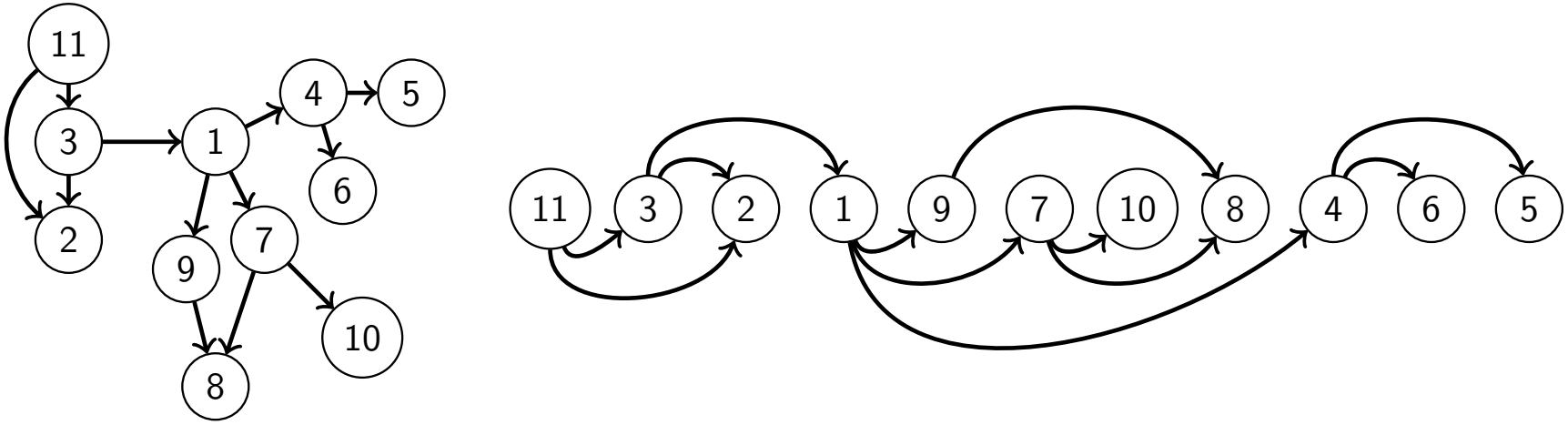
Topologisches Sortieren

Definition

Betrachte gerichteten, azyklischen Graphen $G = (V, E)$. Eine topologische Sortierung von G ist eine lineare Ordnung $(V, \leq)$ derart, dass aus $(u, v) \in E$ immer $u \leq v$ folgt.

Bemerkung: Falls G Zyklen besitzt, kann keine topologisch Sortierung existieren.

Beispiel:



TOP-SORT funktioniert wie DFS, wobei jeder abgearbeitete Knoten an den Kopf einer verketteten Liste eingefügt wird. Diese wird dann zurückgegeben:

Algorithm TOP-SORT(G)

```
 1: for u ∈ V do
 2:     c(u) :=WEISS
 3:     π(u) :=NIL
 4: end for
 5: Sei s eine verkettete Liste
 6: t := 0
 7: for u ∈ V do
 8:     if c(u) =WEISS then
 9:         DFS-VISIT'(G, u)
10:     end if
11: end for
12: return s
```

Algorithm DFS-VISIT'(G, u)

```
 1: t := t+1    % weißer Knoten u gerade entdeckt
 2: g(u) := t
 3: c(u) :=GRAU
 4: for v ∈ Adj[u] do    % (u, v) untersuchen
 5:     if c(v) =WEISS then
 6:         π(v) := u
 7:         DFS-VISIT'(G, v)
 8:     end if
 9: end for
10: c(u) :=SCHWARZ    % u abgearbeitet
11: t := t + 1
12: f(u) := t
13: PUSH(s, u)    % füge u am Kopf von s ein
```

Aufwand: $\Theta(|V| + |E|) + \Theta(|V|) = \Theta(|V| + |E|)$.

Lemma

Ein gerichteter Graph $G = (V, E)$ ist genau dann azyklisch, wenn DFS *keine Rückwärtskanten liefert.*

Beweis: „$\Longrightarrow$“: Es gebe $(u, v) \in E_R$. Dann ist v Vorfahre von u und somit gibt es einen Pfad $P(v \to u)$.
Dann ist aber $P \cup \{(u, v)\}$ ein Zyklus. ↯

„$\Longleftarrow$“: G enthalte Zyklus C. Sei $v \in V(C)$ der erste Knoten von C, der entdeckt wird.
Betrachte $(u, v) \in E(C)$.
Zum Zeitpunkt $g(u)$ bildet $V(C)$ einen Pfad $v \to u$ aus lauter weißen Knoten.
Daher ist u Nachfahre von v im DFS-Wald; somit gilt $(u, v) \in E_R$.
↯ □

Satz

TOP-SORT *ist korrekt.*

Beweis: DFS auf G anwenden, dann ist f bestimmt.

Zu zeigen: $u \neq v, (u,v) \in E \implies f(v) < f(u)$.
Werde (u,v) gerade untersucht. Dann ist v nicht grau, da andernfalls $(u,v) \in E_R$.
Falls v weiß, dann ist v Nachfahre von u im DFS-Wald, also $f(v) < f(u)$.
Falls v schwarz, dann ist $f(v)$ bereits gesetzt, somit $f(v) < f(u)$. □

Bemerkung: Ein Knoten wird erst dann schwarz gefärbt (und in die Liste eingehängt), wenn alle Nachfolger schon schwarz (und bereits in der Liste) sind.

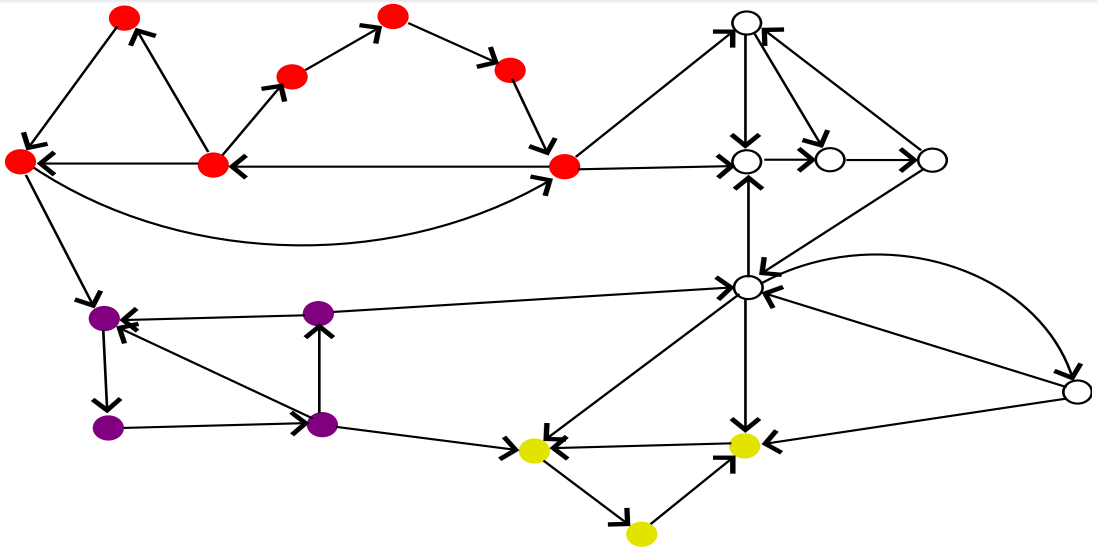
Noch einmal die Erreichbarkeitsrelation

Definition (Symmetrische Erreichbarkeitsrelation)

Sei $G = (V, E)$ ein gerichteter Graph und $v, w \in V$.
Wir definieren: $v \approx w$ genau dann, wenn sowohl eine Kantenfolge von v nach w als auch eine Kantenfolge von w nach v existiert.

Satz

Die Erreichbarkeitsrelation für gerichtete Graphen ist eine Äquivalenzrelation auf V. Ihre Äquivalenzklassen heißen die starken Zusammenhangskomponenten von G.



Ermitteln der starken Zusammenhangskomponenten

Definition (Transponierter Graph)

Der transponierte Graph eines gerichteten Graphen $G = (V, E)$ *ist der Graph* $G^T = (V, E^T)$ *mit* $E^T = \{(u, v) \mid (v, u) \in E\}$.

Definition (Reduktion)

Ein gerichteter Graph $G = (V, E)$ *habe die starken Zusammenhangskomponenten* $K_1, \ldots, K_s$. *Die Reduktion von* G *ist der Graph* $G_R = (V_R, E_R)$ *mit* $V_R = \{K_1, \ldots, K_s\}$ *und* $(K_i, K_j) \in E_R \iff \exists x \in K_i \, \exists y \in K_j : \; (x, y) \in E$.

Algorithm ST-ZU-KO(G)

1: DFS(G)
2: Bestimme den zu G transponierten Graph G^{T}.
3: DFS(G^{T}), wobei in der zweiten Schleife von DFS die Knoten u absteigend bezüglich $f(u)$ abgearbeitet werden.
4: Gib die Komponenten des im vorigen Schritt bestimmten DFS-Waldes aus.

Lemma

Die Reduktion eines gerichteten Graphen ist azyklisch.

Lemma

Sei $G = (V, E)$ gerichtet und $K, L \in V_R$ mit $(K, L) \in E_R$. Wir erweitern g und f wie folgt auf Mengen $U \subseteq V$:

$$g(U) := \min_{x \in U} g(x) \text{ und } f(U) := \max_{x \in U} f(x).$$

Dann gilt $f(K) > f(L)$.

Beweis: 1. Fall: $g(K) < g(L)$, erster in K entdeckter Knoten sei x.

Zum Zeitpunkt $g(x)$ sind alle Knoten von K und L weiß und es gibt weiße Pfade von x zu allen $y \in K$ und – wegen $(K, L) \in E_R$ – auch zu allen $z \in L$.

$\Longrightarrow$ Alle diese Knoten Nachfahren von x im DFS-Wald (Satz d. weißen Pfade),

$\Longrightarrow$ $f(x) = f(K) > f(L)$ (Satz v. d. Intervallschachtelung).

2. Fall: $g(K) > g(L)$, erster in L entdeckter Knoten sei x.
Zum Zeitpunkt $g(x)$ sind alle Knoten von L weiß und es gibt weiße Pfade von x zu allen $y \in L$.
$\Longrightarrow$ Alle diese Knoten Nachfahren von x im DFS-Wald (Satz d. weißen Pfade), $\Longrightarrow$ $f(x) = f(L)$ (Satz v.d. Intervallschachtelung).

Zum Zeitpunkt $g(x)$: Auch die Knoten von K sind weiß, aber von x aus nicht erreichbar.
$\Longrightarrow$ zum Zeitpunkt $f(x)$ gilt: Alle $y \in K$ immer noch weiß, daher gilt $f(K) > f(x) = f(L)$. □

Folgerung

Im Graph G_R^T gilt die umgekehrte Ungleichung: Für $K, L \in V_R$ mit $(K, L) \in E_R^T$ gilt $f(K) < f(L)$.

$$(K, L) \in E_R^{\mathsf{T}} \implies f(K) < f(L) \qquad (*)$$

Satz

ST-ZU-KO ist korrekt.

Beweis: DFS(G^{T}) beginnt mit $x \in K \in V_R$ mit $f(K)$ maximal.
$(*) \implies d^+_{G_R^{\mathsf{T}}}(K) = 0. \implies$ Die Knoten des in x gewurzelten Baums T_1 des DFS-Waldes sind genau jene von K.

Induktion nach # der ausgegebenen Komponenten $T_1, \ldots, T_n$ des DFS-Waldes von G^{T}: $n = 1$: ✓

VS: $T_1, \ldots, T_n$ bilden Knoten von G_R.
T_{n+1} habe Wurzel u; $L = V(T_{n+1})$; DFS(G^{T}) besuche gerade u.

Dann ist $f(u) = f(L) > f(w)$ für alle weißen Knoten w.
(u wird gerade grau)
Laut VS ist $L \setminus \{u\}$ weiß und enthält daher nur Nachfolger von u im DFS-Wald.
$(*) \implies$ alle Kanten aus E^{T} nur zu bereits besuchten Knoten.
$L \setminus \{u\}$ enthält daher alle Nachfolger von u, also $V(T_{n+1}) \in V_R$. □

Anwendungen der Tiefensuche:

- topologische Sortierung
- Ermittlung der starken Zusammenhangskomponenten
- Ermittlung der Zusammenhangskomponenten
- Finden von Zyklen
- Planaritätstests
- Lösen oder Erzeugen von Irrgärten

12) Minimale Spannbäume

Minimale Spannbäume

Allgemeine Problemstellung

$G = (V, E)$ ungerichtet und zusammenhängend, $w : E \to \mathbb{R}$ Gewichtsfunktion,
$F \subseteq E$: Wir setzen $w(F) = \sum_{e \in F} w(e)$.

Definition

$F \subseteq E$, $T = (V, F)$ heißt Spannbaum von G, wenn (V, F) ist ein Baum ist.

Problem:
gegeben: $G = (V, E)$, $w : E \to \mathbb{R}$;
gesucht: Spannbaum mit minimalem Gewicht

Generische Methode: Erzeuge Mengen $A \subseteq E$, sodass vor bzw. nach jeder Iteration gilt: (V, A) ist Teilgraph eines minimalen Spannbaumes $T = (V, F)$.

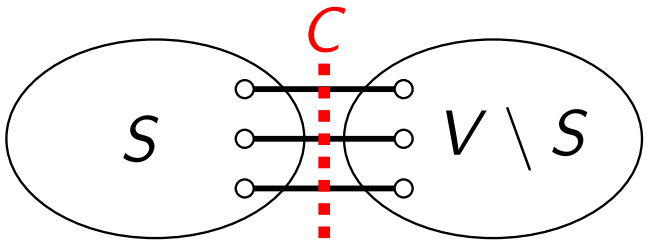
Minimale Spannbäume

Algorithm GENERIC-MST(G, w)

```
1: A := ∅
2: while (V, A) ≠ Spannbaum do
3:     finde e ∈ E mit (V, A ∪ {e}) Teilgraph eines minimalen Spannbaums
4:     A := A ∪ {e}
5: end while
6: return A
```

Definition

Eine Zerlegung $C := (S, V \setminus S)$ mit $S \subseteq V, S \neq \varnothing, S \neq V$ heißt Schnitt von G.
$e = v_1 v_2$ Kante von C, wenn $v_1 \in S$ und $v_2 \in V \setminus S$.
e heißt leichte Kante von C, wenn e Kante von C und $w(e)$ minimal.



Minimale Spannbäume

Satz

$G = (V, E)$ sei zusammenhängend, $w : E \to \mathbb{R}$ und $T := (V, F)$ $(F \subseteq E)$ ein minimaler Spannbaum.
Weiters: $A \subseteq F$, $C = (S, V \setminus S)$ ein Schnitt, der A respektiert, d.h. kein $e \in A$ ist Kante von C. e sei eine leichte Kante von C.
Dann gibt es einen minimalen Spannbaum $T' = (V, F')$ mit $A \cup \{e\} \subseteq F'$.

Beweis:

1. Fall: $e \in F$. Dann ist offensichtlich $T' := T = (V, F)$.
2. Fall: $e = uv \notin F$.
   u, v verbunden in T (Pfad $P \subseteq F$), also ist $P \cup \{e\}$ ein Kreis.
   Da $u \in S$, $v \in V \setminus S$, gibt es $f = xy \in P$ in C $(f \notin A)$.
   $T \setminus \{f\}$ zerfällt daher in zwei Komponenten $K_1, K_2 \subseteq V$ mit $u \in K_1$, $v \in K_2$.
   Folglich ist $T' = (T \setminus \{f\}) \cup \{e\}$ ein Spannbaum.

Minimale Spannbäume

Noch zu zeigen: $T' = (T \setminus \{f\}) \cup \{e\}$ ist minimal:
e ist leicht, somit gilt $w(e) \leq w(f)$ und daher $w(T') \leq w(T)$, also $w(T') = w(T)$.
Außerdem gilt: $A \cup \{e\} \subseteq F' := (F \setminus \{f\}) \cup \{e\}$, weil $f \notin A$. □

Ad GENERIC-MST:

$G_A = (V, A)$ ist immer ein Wald, denn am am Anfang gilt
$G_\varnothing = (V, \varnothing)$ (Wald mit $|V|$ Komponenten)
Die Zuweisung $A := A \cup \{e\}$ verbindet zwei Komponenten von G_A.
Folgerung: Anzahl der Iterationen der **while**-Schleife ist $|V| - 1$.

Minimale Spannbäume

Folgerung

$G = (V, E)$ sei zusammenhängend, $w : E \to \mathbb{R}$, $T := (V, F)$ $(F \subseteq E)$ ein minimaler Spannbaum und $A \subseteq F$.
Sei $K = (V_K, E_K)$ eine Zusammenhangskomponente von $G_A = (V, A)$. Dann gilt für jede bezüglich $(V_K, V \setminus V_K)$ leichte Kante, dass $(V, A \cup \{e\})$ ist kreisfrei ist.

Beweis: e ist leicht und $(V_K, V \setminus V_K)$ respektiert A. □

Der Algorithmus von Kruskal

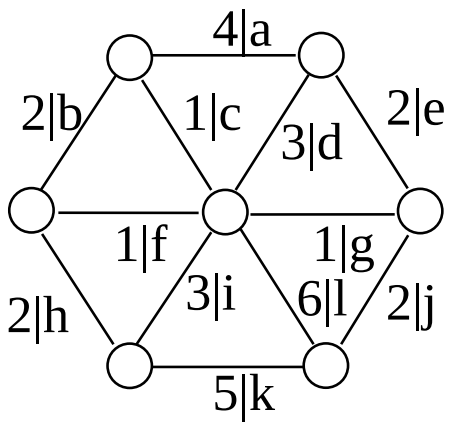
Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach dem Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Der Algorithmus verwendet folgende Hilfsfunktionen:

1. MAKE-SET(v): erzeugt eine neue Menge V mit $v \in V$;
2. FIND-SET(u): gibt die Komponente von u zurück;
3. UNION(u, v): vereinigt die Komponenten von u und v.

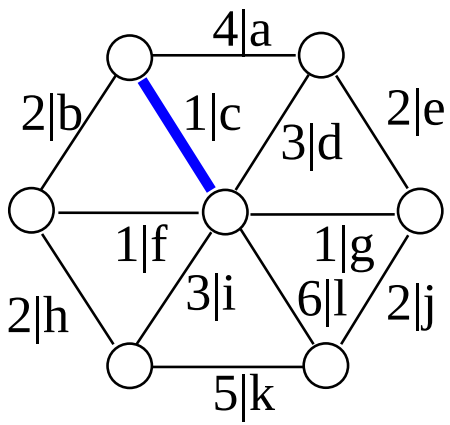
Minimale Spannbäume



Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

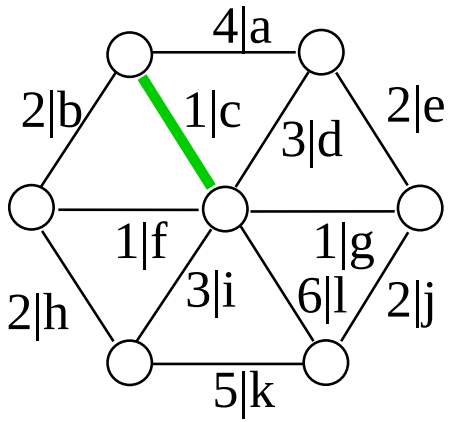


$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$
$A = \varnothing$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

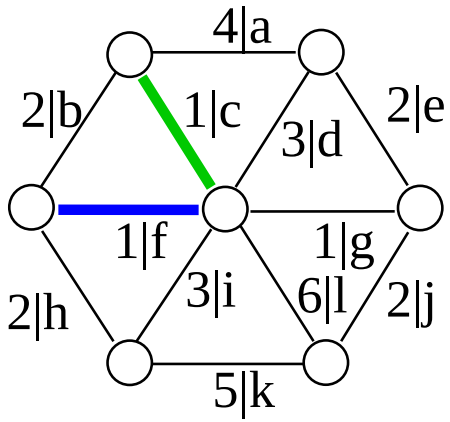


$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$
$A = \{c\}$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

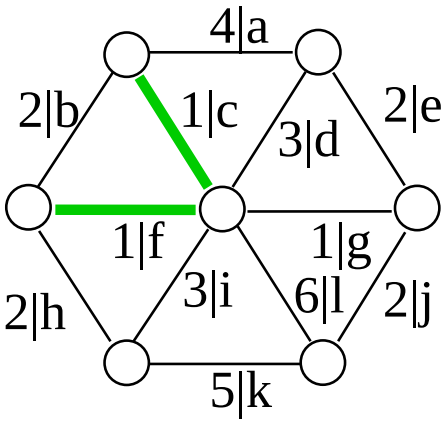


$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$
$A = \{c\}$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

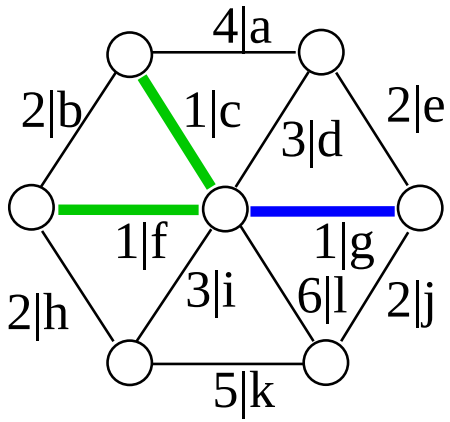


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

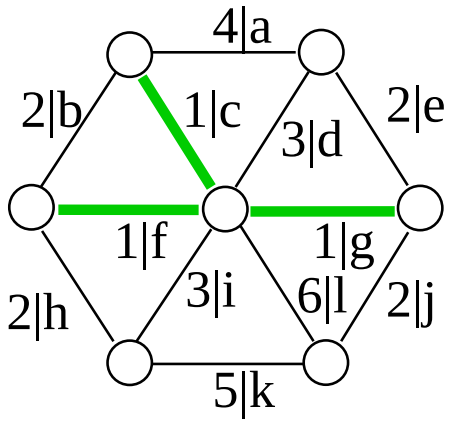


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

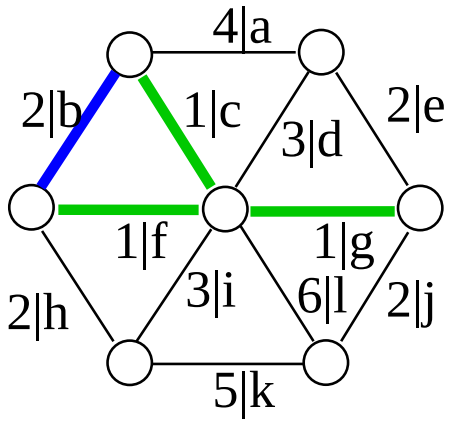


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

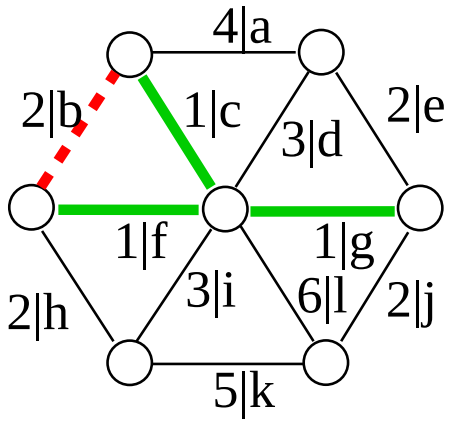


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

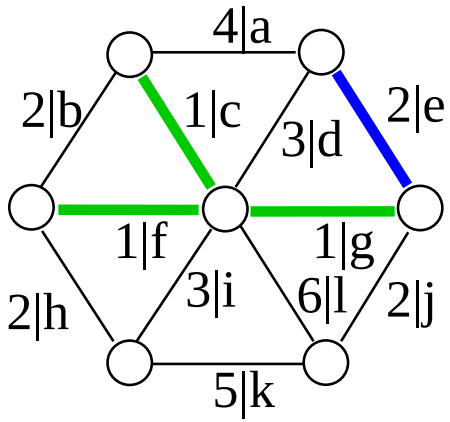


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

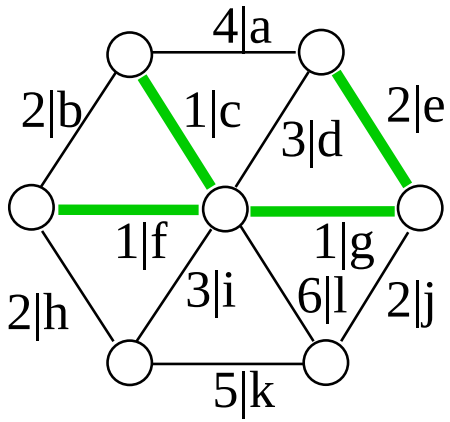


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

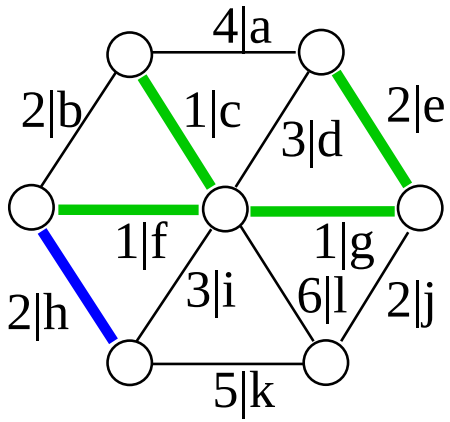


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g, e\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

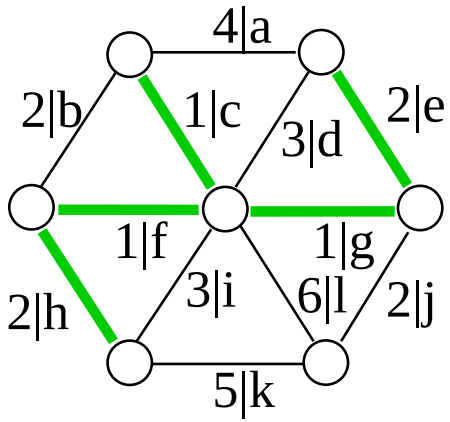


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g, e\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

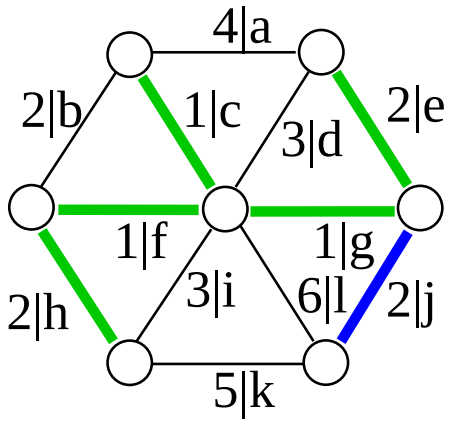


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g, e, h\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

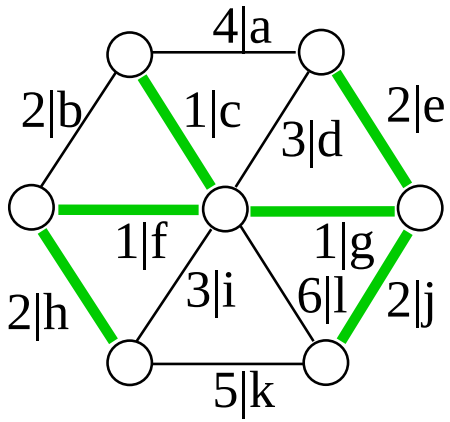


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g, e, h\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume

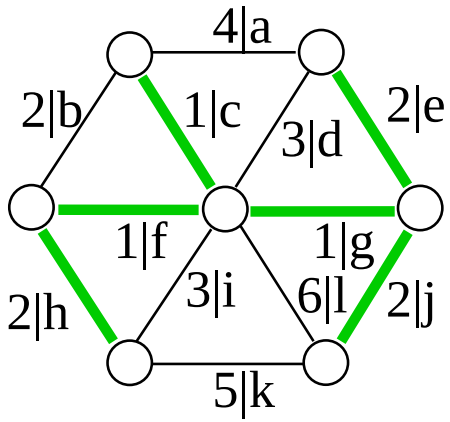


$$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$$
$$A = \{c, f, g, e, h, j\}$$

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Minimale Spannbäume



$E = \{c, f, g, b, e, h, j, d, i, a, k, l\}$
$A = \{c, f, g, e, h, j\}$
$|A| = 6 \rightsquigarrow$ ENDE

Algorithm MST-KRUSKAL(G, w)

```
 1: Sortiere E aufsteigend nach Gewicht w
 2: A := ∅
 3: for v ∈ V do
 4:     MAKE-SET(v)
 5: end for
 6: for uv ∈ E do
 7:     if FIND-SET(u) ≠FIND-SET(v) then
 8:         A := A ∪ {uv}
 9:         UNION(u, v)
10:     end if
11: end for
12: return A
```

Der Algorithmus von Prim

konstruiert MST ausgehend

Algorithm MST-PRIM(G, w, r)

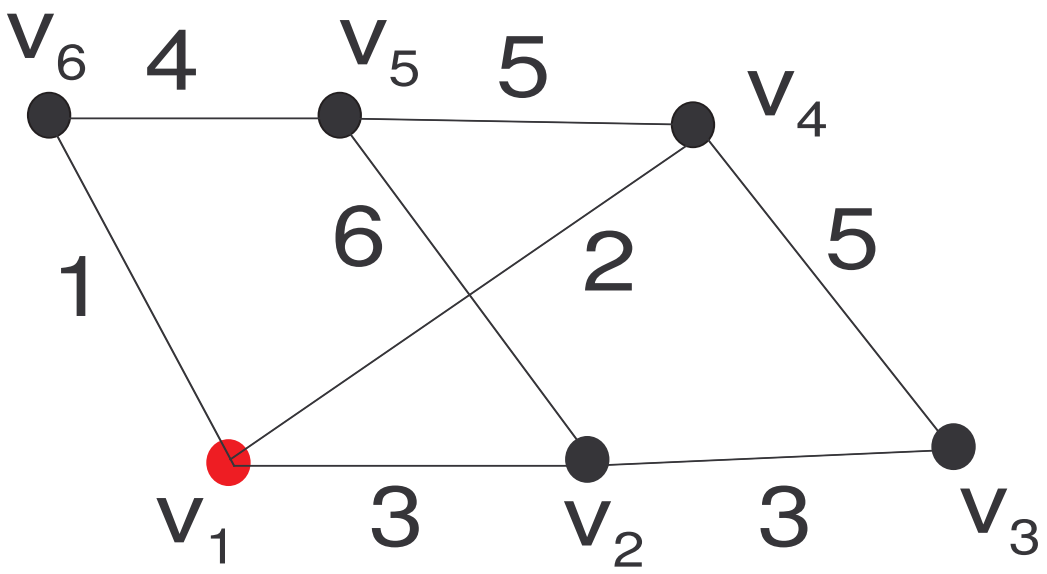
```
 1: for u ∈ V do
 2:     g(u) := ∞
 3:     π(u) :=NIL
 4: end for
 5: g(r) := 0
 6: Q := V        % Prioritätswarteschlange
 7: while Q ≠ ∅ do
 8:     u :=EXTRACT-MIN(Q)
 9:     for v ∈Adj[u] do
10:         if v ∈ Q and w(u, v) < g(v) then
11:             π(v) := u
12:             g(v) := w(u, v)
13:         end if
14:     end for
15: end while
```

Minimale Spannbäume

Anmerkungen:

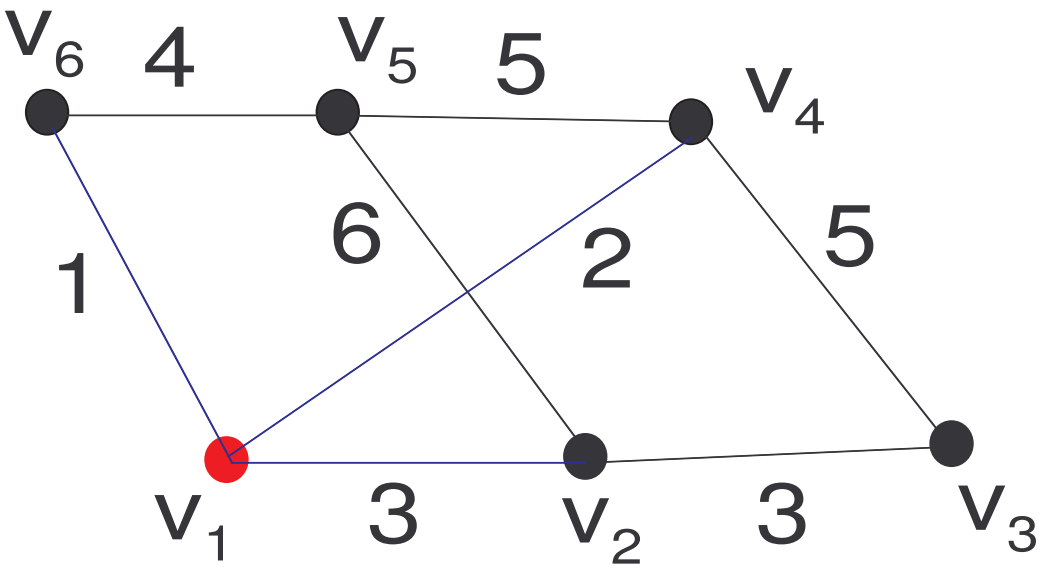
- $A = \{v\pi(v) \mid v \in V \setminus (\{r\} \cup Q)\}$ ist ein Baum. (Schleifeninvariante)
- $V \setminus Q$ ist die Menge der Knoten des aktuellen MST.
- $\forall v \in Q : \pi(v) \neq \text{NIL} \implies g(v) < \infty$.
- $v\pi(v)$ ist leicht bezüglich $(Q, V \setminus Q)$.
- Kosten: $\mathcal{O}(\log |V|)$ für EXTRACT-MIN, insgesamt daher $\mathcal{O}(|V| \log |V| + |E|)$.
- Vergleich mit Kruskal: $\mathcal{O}(|E| \log |E|)$.

Minimale Spannbäume



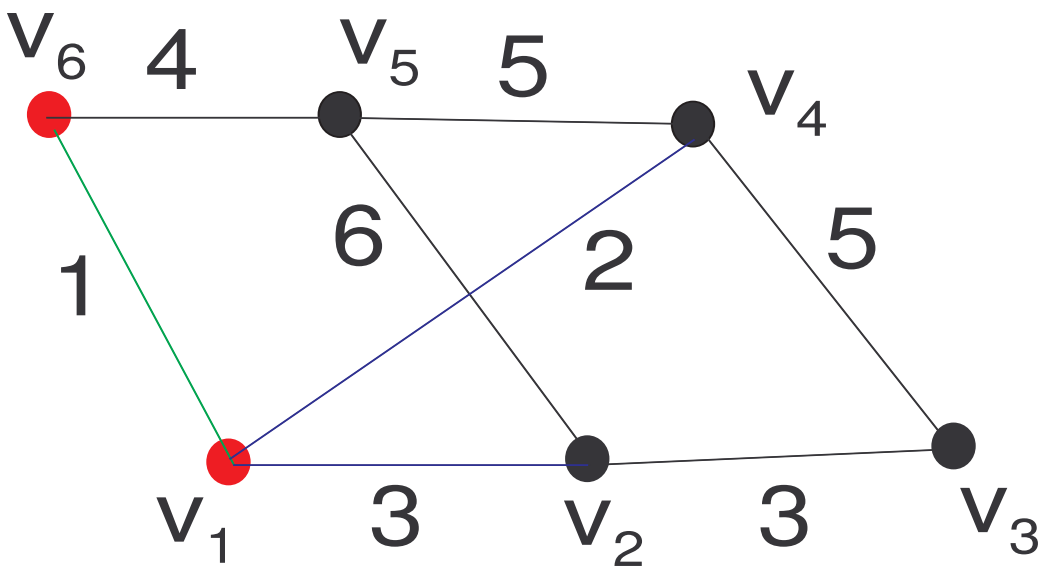
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1					

Minimale Spannbäume



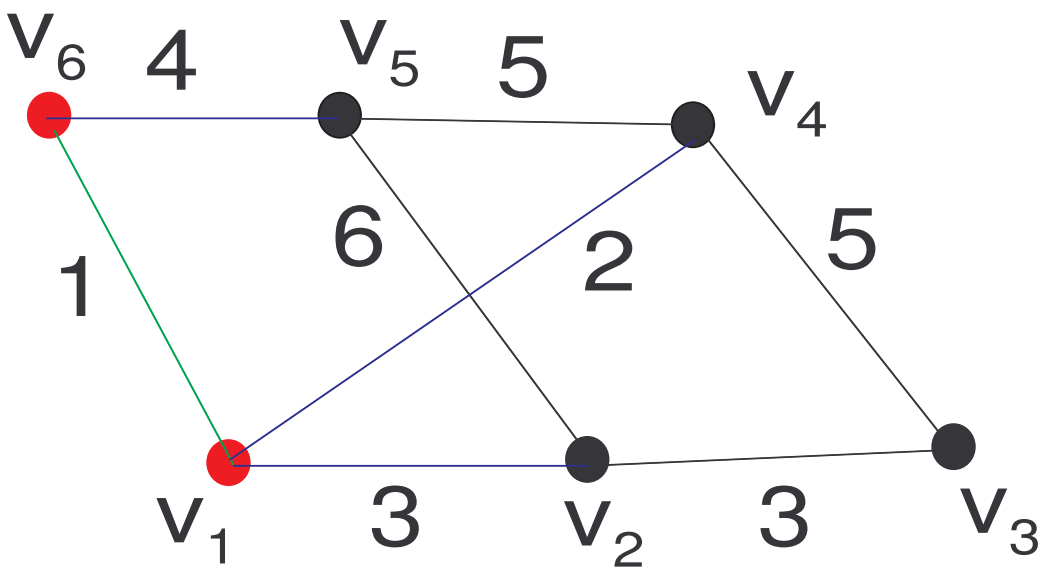
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	**1**

Minimale Spannbäume



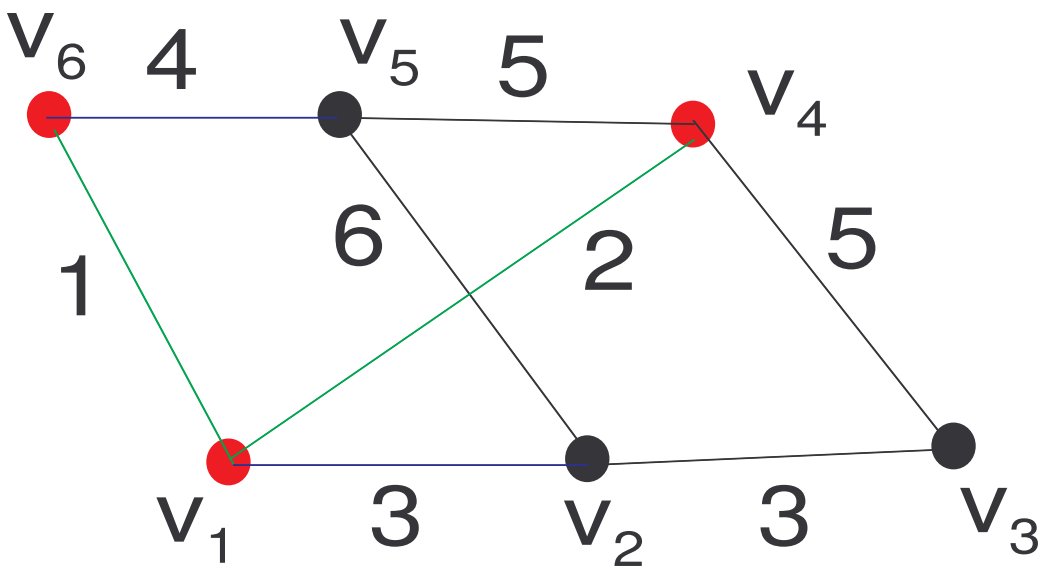
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6					–

Minimale Spannbäume



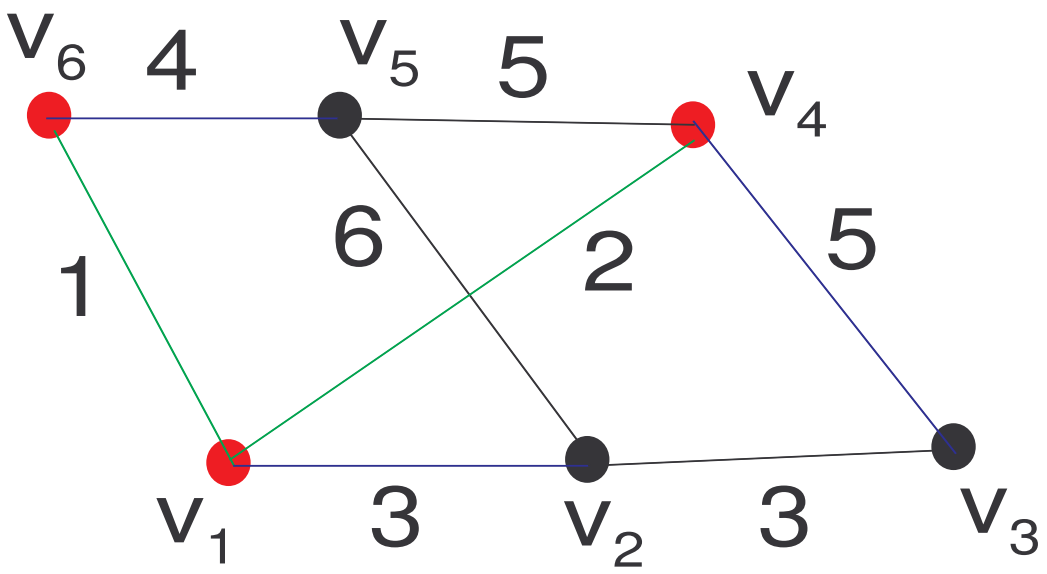
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6	3	∞	**2**	4	–

Minimale Spannbäume



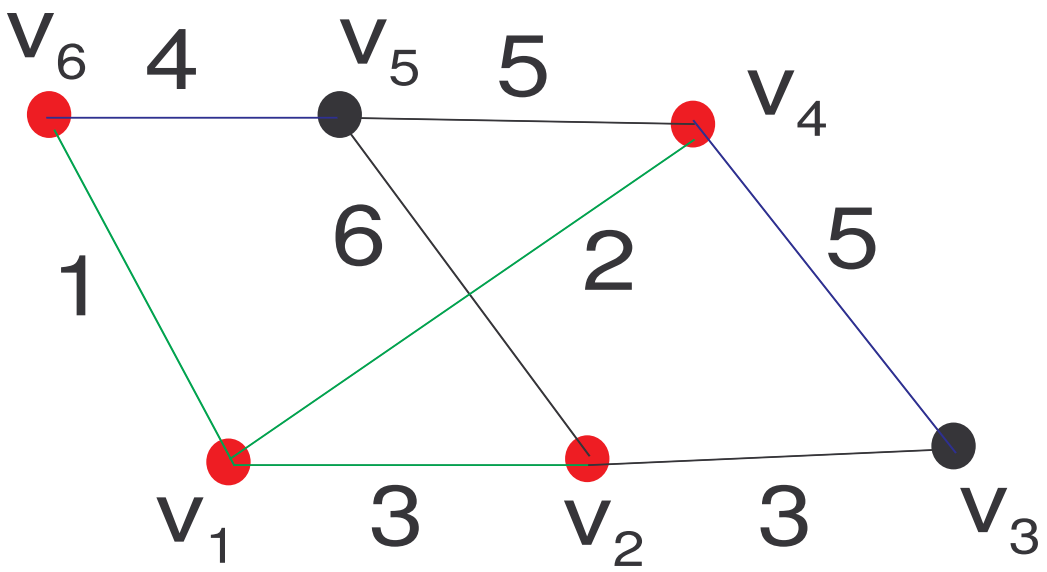
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6	3	∞	2	4	–
v_4			–		–

Minimale Spannbäume



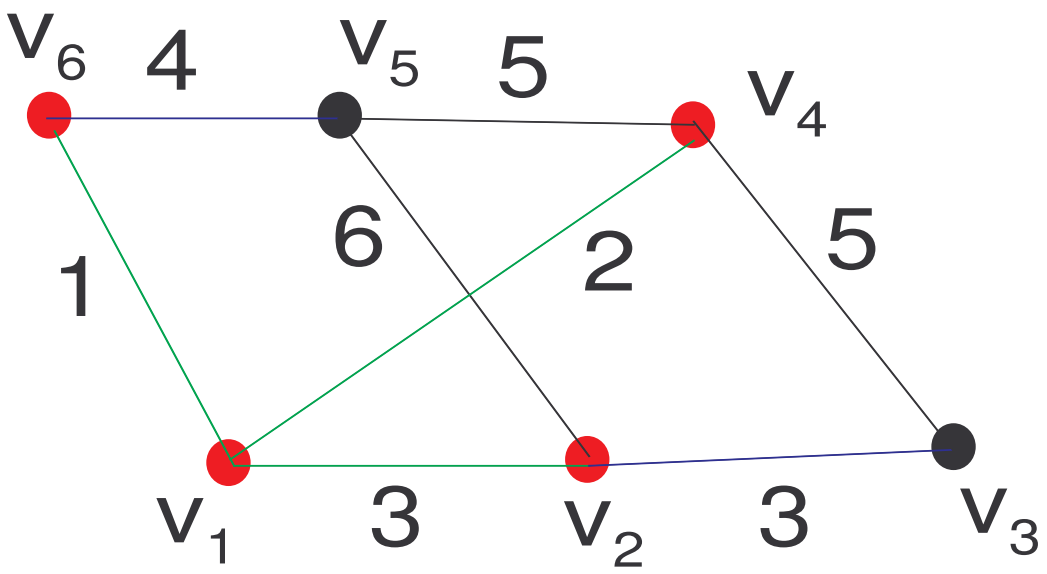
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6	3	∞	2	4	–
v_4	**3**	5	–	4	–

Minimale Spannbäume



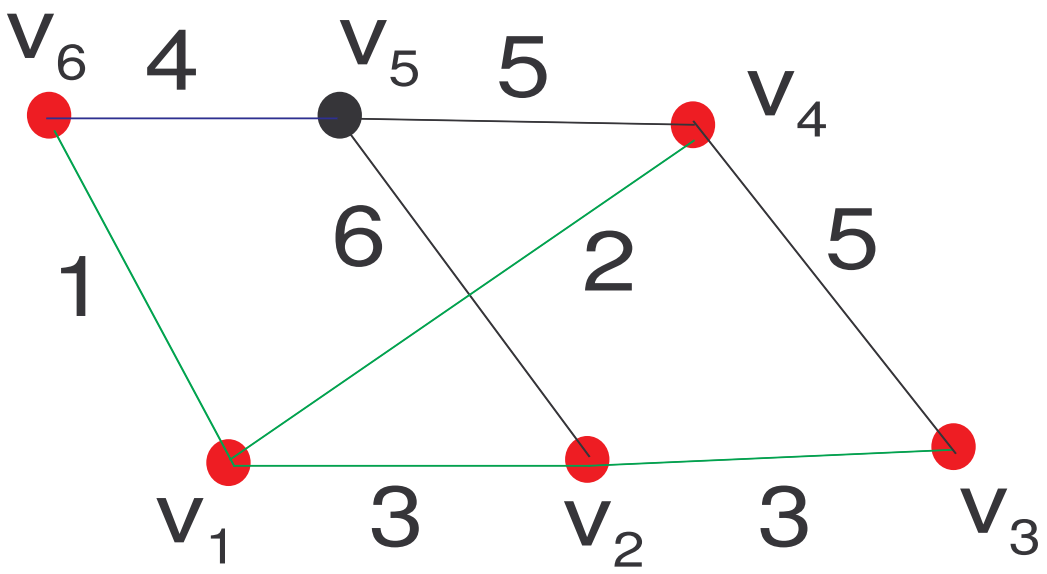
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6	3	∞	2	4	–
v_4	3	5	–	4	–
v_2	–		–		–

Minimale Spannbäume



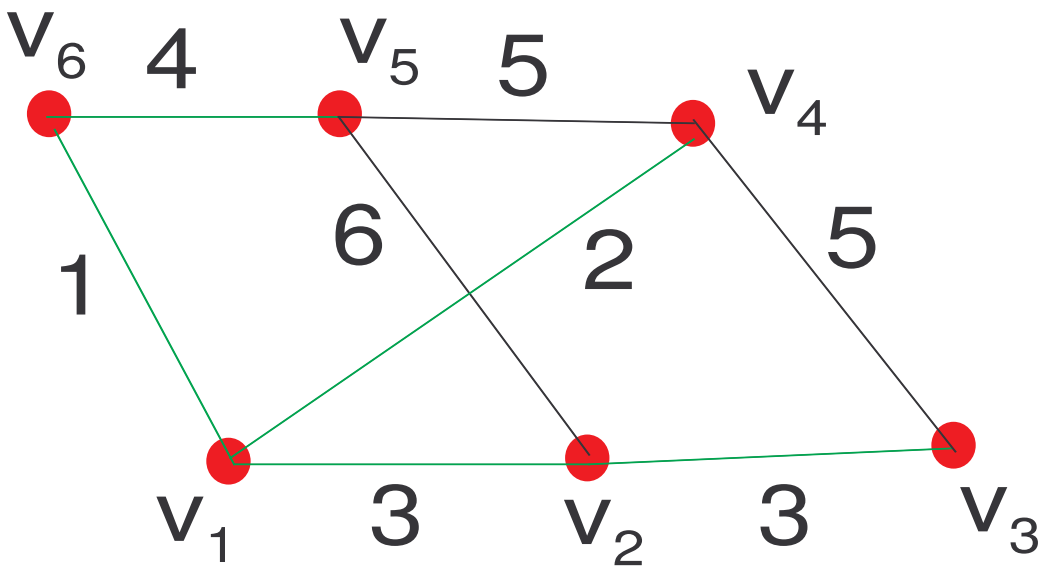
	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6	3	∞	2	4	–
v_4	3	5	–	4	–
v_2	–	$\boxed{3}$	–	4	–

Minimale Spannbäume



	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6	3	∞	2	4	–
v_4	3	5	–	4	–
v_2	–	3	–	4	–
v_3	–	$\boxed{3}$	–	4	–

Minimale Spannbäume



	$g(v_2)$	$g(v_3)$	$g(v_4)$	$g(v_5)$	$g(v_6)$
v_1	3	∞	2	∞	1
v_6	3	∞	2	4	–
v_4	3	5	–	4	–
v_2	–	3	–	4	–
v_3	–	$\boxed{3}$	–	4	–

13) Matroide und Greedy-Algorithmen

Matroide und Greedy-Algorithmen

Definition

Ein Paar (E, S) heißt Unabhängigkeitssystem, wenn E eine Menge ist, $S \subseteq \mathcal{P}(E)$, nicht leer und abgeschlossen bezüglich der Inklusion, d.h., falls $A \in S$ und $B \subseteq A$, dann gilt $B \in S$.
Die Elemente von S werden als unabhängige Mengen bezeichnet, jene von $\mathcal{P}(E) \setminus S$ als abhängige Mengen.

Beispiele:

- Sei $G = (V, E)$ ein Graph, $S := \{F \subseteq E \mid (V, F) \text{ ist Wald } \}$. Dann ist (E, S) ein Unabhängigkeitssystem.
- Sei $E \subseteq \mathbb{R}^n$ und $S := \{F \subseteq E \mid F \text{ ist lin. unabh.}\}$. Dann ist (E, S) ein Unabhängigkeitssystem.

Optimierungsproblem:
Gegeben: Unabhängigkeitssystem (E, S), Gewichtsfunktion $w : E \to \mathbb{R}$; $w(A) := \sum_{x \in A} w(x)$;
Gesucht: $A \in S$, maximal bezüglich der Mengeninklusion, sodass $w(A)$ minimal (oder maximal) ist.

Algorithm GREEDY(E, S, w)

```
1: Sortiere E = {e_1, e_2, ..., e_m} aufsteigend nach Gewicht
2: T := ∅
3: for k = 1 to m do
4:     if T ∪ {e_k} ∈ S then
5:         T := T ∪ {e_k}
6:     end if
7: end for
```