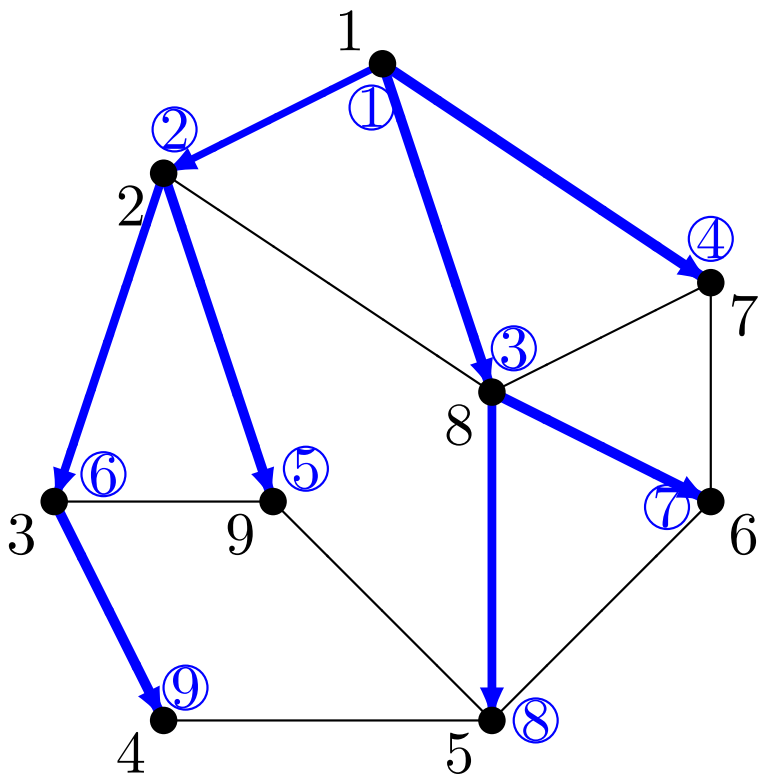
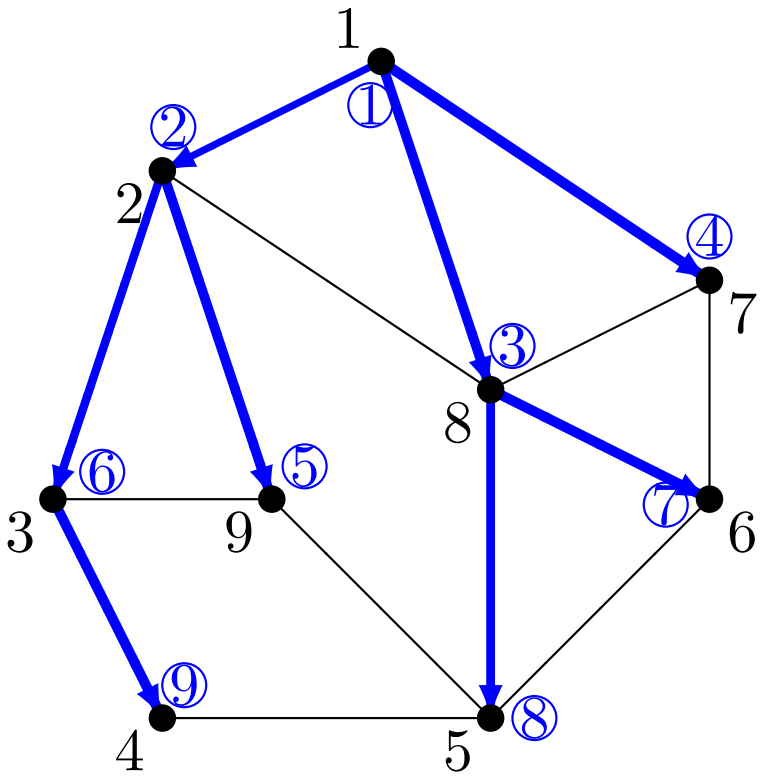
6.2 BFS-Algorithmus

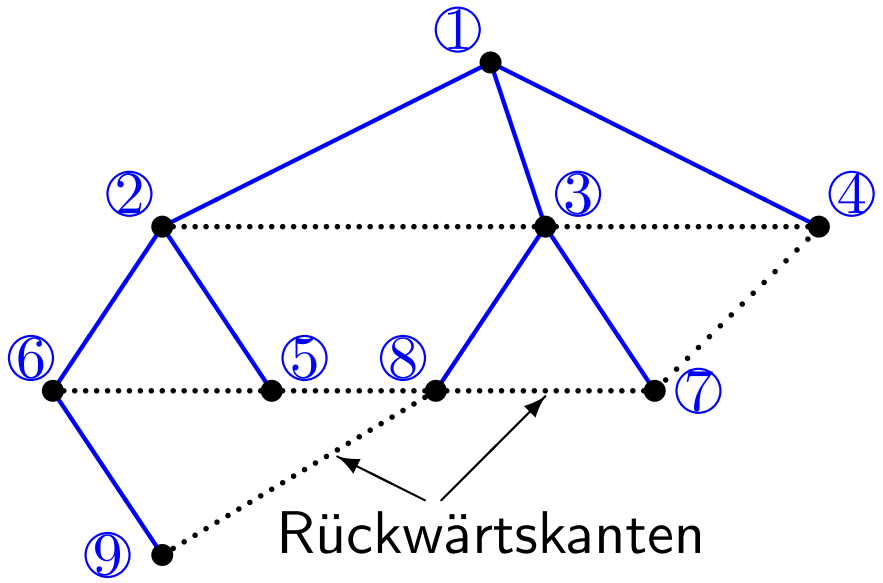
while $\exists$ unvisited v **do**
    $r :=$ pick (random) unvisited node
    push r into (end of) queue
    while queue $\neq \emptyset$ **do**
        $v :=$ remove front element of queue
        if v unvisited **then**
            mark v visited
            append all neighbours of v to end of queue
            perform operations BFS_Ops(v)
        fi
    od
od

Beispiel 184

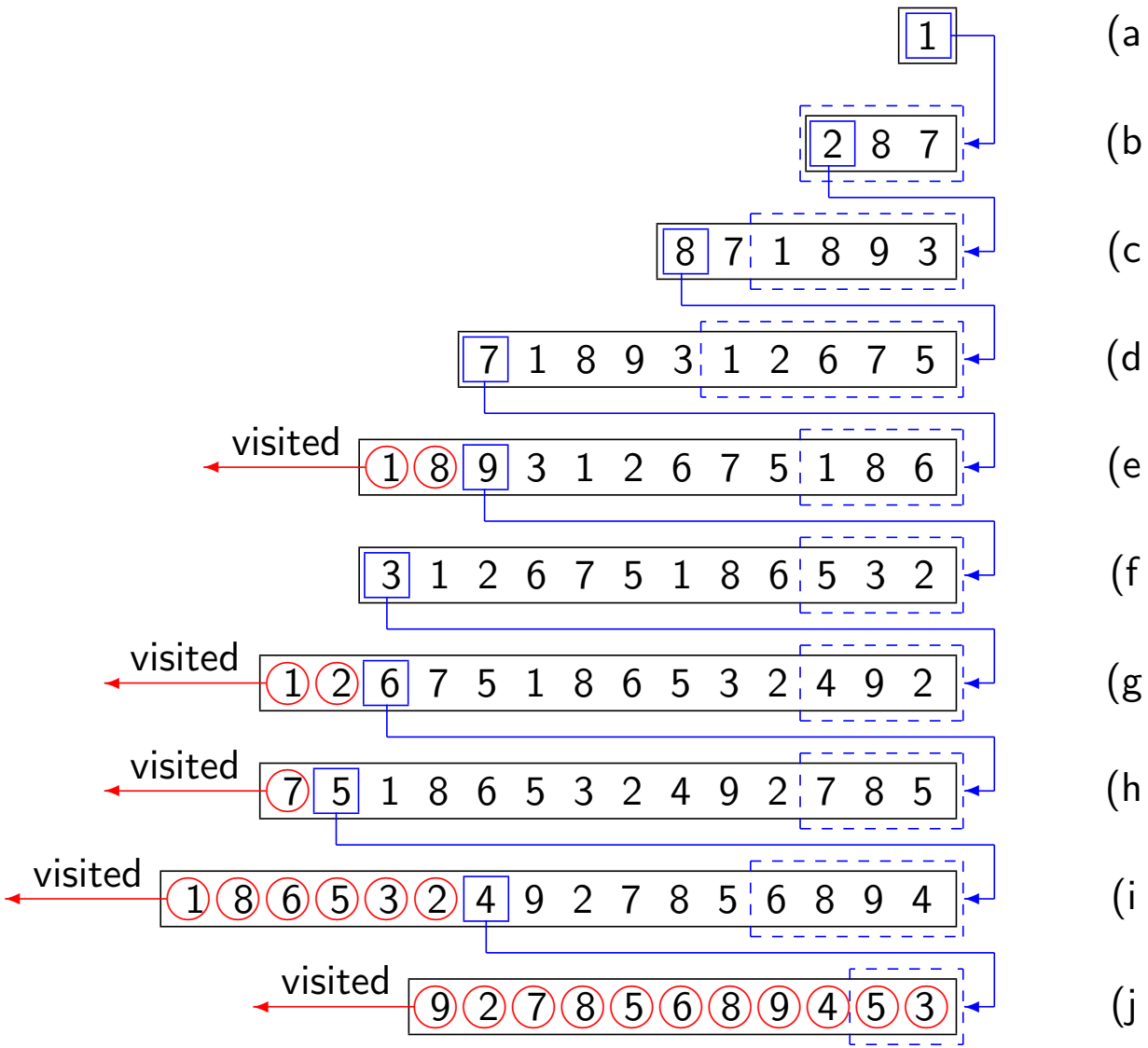


Beobachtung: Die markierten Kanten bilden einen Spannbaum:





Folge der Queuezustände



Wir betrachten die Folge der Queuezustände. Wiederum bedeutet die Notation:

 : vorderstes Queue-Element

 : schon besuchte Knoten

 : Nachbarn

Im Zustand e) sind die Elemente 1 und 8 als visited markiert (siehe Zustände a) und c)). Deswegen werden sie aus der Queue entfernt, und so wird das Element 9 das vorderste Queueelement. Das gleiche passiert in den Zuständen g), h) und i). Im Zustand j) sind alle Elemente markiert, so dass sie eins nach dem anderen aus der Queue entfernt werden.

7. Minimale Spannbäume

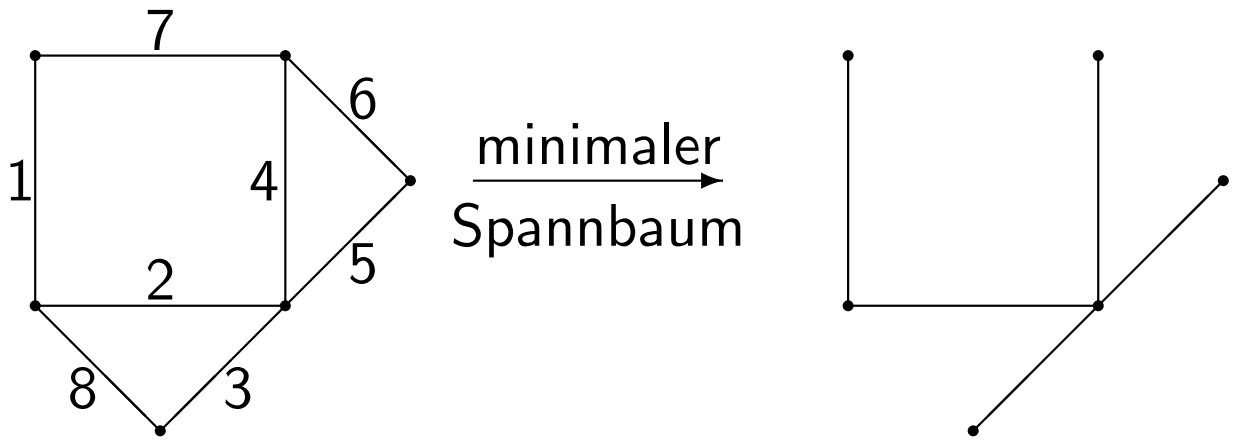
Sei $G = (V, E)$ ein einfacher ungerichteter Graph, der o.B.d.A. zusammenhängend ist. Sei weiter $w : E \to \mathbb{R}$ eine Gewichtsfunktion auf den Kanten von G.
Wir setzen

- $E' \subseteq E$: $w(E') = \sum_{e \in E'} w(e)$,
- $T = (V', E')$ ein Teilgraph von G: $w(T) = w(E')$.

Definition 185
T heißt minimaler Spannbaum (MSB, MST) von G, falls T Spannbaum von G ist und gilt:

$$w(T) \leq w(T') \text{ für alle Spannbäume } T' \text{ von } G.$$

Beispiel 186



Anwendungen:

- Telekom: Verbindungen der Telefonvermittlungen
- Leiterplatinen

7.1 Konstruktion von minimalen Spannbäumen

Es gibt zwei Prinzipien für die Konstruktion von minimalen Spannbäumen (Tarjan):

- „blaue“ Regel
- „rote“ Regel

Satz 187

*Sei $G = (V, E)$ ein zusammenhängender ungerichteter Graph, $w : E \to \mathbb{R}$ eine Gewichtsfunktion, $C = (V_1, V_2)$ ein Schnitt (d.h. $V = V_1 \cup V_2$, $V_1 \cap V_2 = \emptyset$, $V_1 \neq \emptyset \neq V_2$). Sei weiter $E_C = E \cap (V_1 \times V_2)$ die Menge der Kanten „über den Schnitt hinweg“. Dann gilt: (**„blaue“ Regel**)*

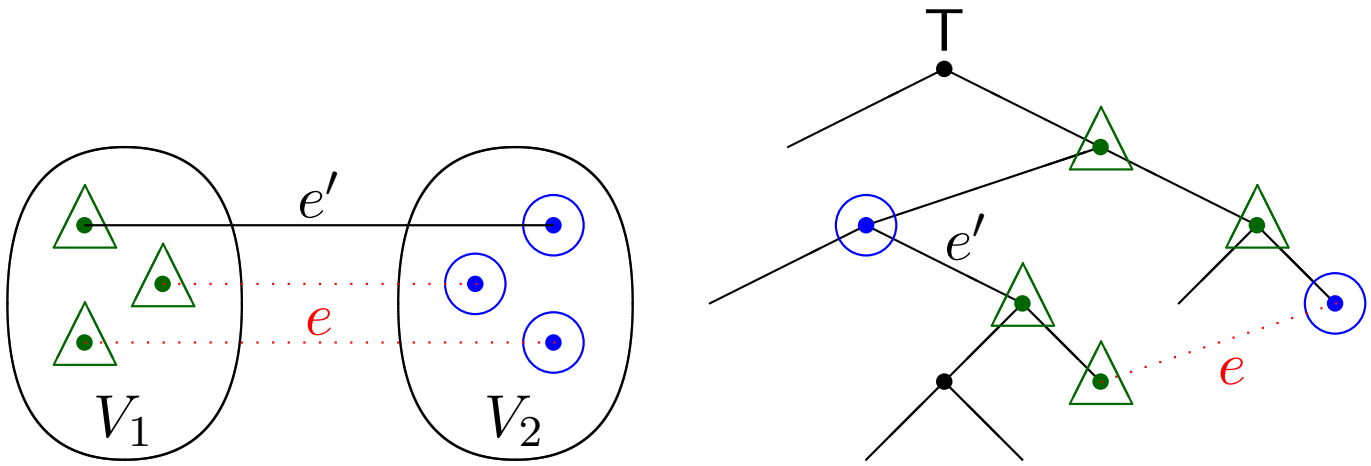
1. *Ist $e \in E_C$ die einzige Kante minimalen Gewichts (über alle Kanten in E_C), dann ist e in jedem minimalen Spannbaum für (G,w) enthalten.*
2. *Hat $e \in E_C$ minimales Gewicht (über alle Kanten in E_C), dann gibt es einen minimalen Spannbaum von (G,w), der e enthält.*

Beweis:

[durch Widerspruch]

1. Sei T ein minimaler Spannbaum von (G, w), sei $e \in E_C$ die minimale Kante. Annahme: $e \notin T$. Da T Spannbaum $\Rightarrow T \cap E_C \neq \emptyset$.
   Sei $T \cap E_C = \{e_1, e_2, \ldots, e_k\}$, $k \geq 1$. Dann enthält $T \cup \{e\}$ einen eindeutig bestimmten Kreis (den sogenannten durch e bzgl. T bestimmten Fundamentalkreis). Dieser Kreis muss mindestens eine Kante $\in E_C \cap T$ enthalten, da die beiden Endpunkte von e auf verschiedenen Seiten des Schnitts C liegen.

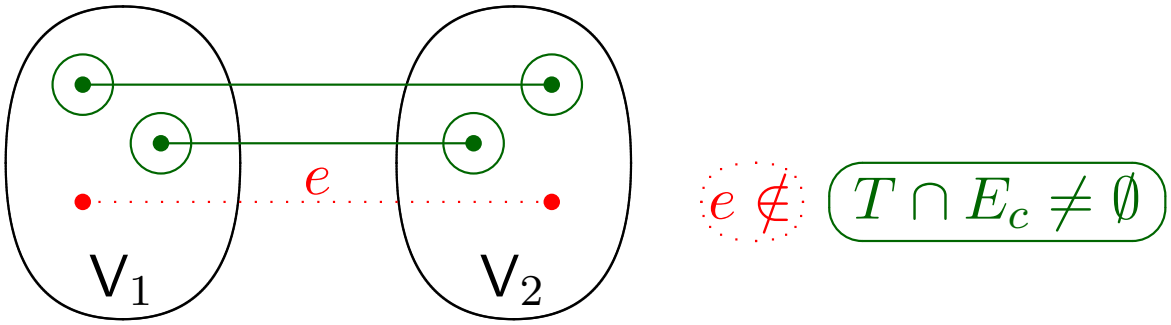
Beweis (Forts.):



Sei $e' \in E_C \cap T$. Dann gilt nach Voraussetzung $w(e') > w(e)$. Also ist $T' := T - \{e'\} \cup \{e\}$ ein Spannbaum von G, der echt kleineres Gewicht als T hat, Widerspruch zu „T ist minimaler Spannbaum“.

Beweis (Forts.):

2. Sei $e \in E_C$ minimal. Annahme: e kommt in keinem minimalen Spannbaum vor. Sei T ein beliebiger minimaler Spannbaum von (G, w).



$e \notin T \cap E_C \neq \emptyset$. Sei $e' \in E_C \cap T$ eine Kante auf dem durch e bezüglich T erzeugten Fundamentalkreis. Dann ist $T' = T - \{e'\} \cup \{e\}$ wieder ein Spannbaum von G, und es ist $w(T') \leq w(T)$. Also ist T' minimaler Spannbaum und $e \in T'$.

Satz 188

*Sei $G = (V, E)$ ein ungerichteter, gewichteter, zusammenhängender Graph mit Gewichtsfunktion $w : E \to \mathbb{R}$. Dann gilt: (**„rote“ Regel**)*

1. *Gibt es zu $e \in E$ einen Kreis C in G, der e enthält und $w(e) > w(e')$ für alle $e' \in C \setminus \{e\}$ erfüllt, dann kommt e in keinem minimalen Spannbaum vor.*
2. *Ist $C_1 = e_1, \ldots, e_k$ ein Kreis in G und $w(e_i) = \max\{w(e_j);\ 1 \leq j \leq k\}$, dann gibt es einen minimalen Spannbaum, der e_i nicht enthält.*

Beweis:

1. Nehmen wir an, dass es einen minimalen Spannbaum T gibt, der $e = \{v_1, v_2\}$ enthält. Wenn wir e aus T entfernen, so zerfällt T in zwei nicht zusammenhängende Teilbäume T_1 und T_2 mit $v_i \in T_i$, $i = 1, 2$. Da aber e auf einem Kreis in G liegt, muss es einen Weg von v_1 nach v_2 geben, der e nicht benützt. Mithin gibt es eine Kante $\hat{e} \neq e$ auf diesem Weg, die einen Knoten in T_1 mit T_2 verbindet. Verbinden wir T_1 und T_2 entlang $\hat{e}$, so erhalten wir einen von T verschiedenen Spannbaum $\hat{T}$. Wegen $w(\hat{e}) < w(e)$ folgt $w(\hat{T}) < w(T)$, im Widerspruch zur Minimalität von T.

Beweis (Forts.):

2. Wir nehmen an, e_i liege in jedem minimalen Spannbaum (MSB) von G, und zeigen die Behauptung durch Widerspruch.

   Sei T ein beliebiger MSB von G. Entfernen wir e_i aus T, so zerfällt T in zwei nicht zusammenhängende Teilbäume T_1 und T_2. Da e_i auf einem Kreis $C_1 = e_1, \ldots, e_k$ in G liegt, können wir wie zuvor e_i durch eine Kante e_j des Kreises C_1 ersetzen, die T_1 und T_2 verbindet. Dadurch erhalten wir einen von T verschiedenen Spannbaum $\tilde{T}$, der e_i nicht enthält. Da nach Voraussetzung $w(e_j) \leq w(e_i)$ gilt, folgt $w(\tilde{T}) \leq w(T)$ (und sogar $w(\tilde{T}) = w(T)$, da T nach Annahme ein MSB ist). Also ist $\tilde{T}$ ein MSB von G, der e_i nicht enthält, im Widerspruch zur Annahme, e_i liege in *jedem* MSB von G.

□

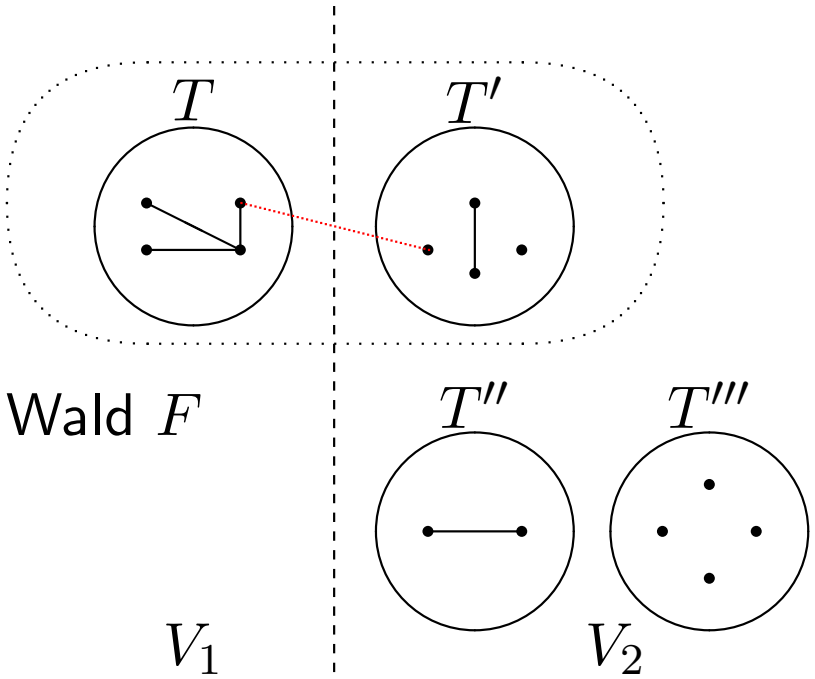
Literatur

Robert E. Tarjan:
Data Structures and Network Algorithms
SIAM CBMS-NSF Regional Conference Series in Applied Mathematics Bd. 44 (1983)

7.2 Generischer minimaler Spannbaum-Algorithmus

Initialisiere Wald F von Bäumen, jeder Baum ist ein singulärer Knoten
(jedes $v \in V$ bildet einen Baum)
while Wald F mehr als einen Baum enthält **do**
    wähle einen Baum $T \in F$ aus
    bestimme eine leichteste Kante $e = \{v, w\}$ aus T heraus
    sei $v \in T$, $w \in T'$
    vereinige T und T', füge e zum minimalen Spannbaum hinzu
od

Generischer MST-Algorithmus



7.3 Kruskal's Algorithmus

algorithm Kruskal $(G, w) :=$
  sortiere die Kanten nach aufsteigendem Gewicht in eine Liste L
  initialisiere Wald $F = \{T_i, i = 1, \ldots, n\}$, mit $T_i = \{v_i\}$
  MSB:=$\emptyset$
  for $i := 1$ **to** length(L) **do**
    $\{v, w\} := L_i$
    $x :=$ Baum $\in F$, der v enthält; **co** $x :=$*Find*(v) **oc**
    $y :=$ Baum $\in F$, der w enthält; **co** $y :=$*Find*(w) **oc**
    if $x \neq y$ **then**
      MSB:=MSB $\cup \{\{v, w\}\}$
      Union(x, y) **co** gewichtete Vereinigung **oc**
    fi
  od

Korrektheit: Falls die Gewichte eindeutig sind ($w(\cdot)$ injektiv), folgt die Korrektheit direkt mit Hilfe der "blauen" und "roten" Regel.
Ansonsten Induktion über die Anzahl $|V|$ der Knoten:
Ind. Anfang: $|V|$ klein: $\surd$

Sei $r \in \mathbb{R}$, $E_r := \{e \in E; w(e) < r\}$.
Es genügt zu zeigen:
Sei $T_1, \ldots, T_k$ ein minimaler Spannwald für $G_r := \{V, E_r\}$ (d.h., wir betrachten nur Kanten mit Gewicht $< r$). Sei weiter T ein MSB von G, dann gilt die

Hilfsbehauptung: Die Knotenmenge eines jeden T_i induziert in T einen zusammenhängenden Teilbaum, dessen Kanten alle Gewicht $< r$ haben.

Beweis der Hilfsbehauptung:

Sei $T_i =: (V_i, E_i)$. Wir müssen zeigen, dass V_i in T einen zusammenhängenden Teilbaum induziert. Seien $u, v \in V_i$ zwei Knoten, die in T_i durch eine Kante e verbunden sind. Falls der Pfad in T zwischen u und v auch Knoten $\notin V_i$ enthält (also der von V_i induzierte Teilgraph von T nicht zusammenhängend ist), dann enthält der in T durch Hinzufügen der Kante e entstehende Fundamentalkreis notwendigerweise auch Kanten aus $E \setminus E_r$ und ist damit gemäß der "roten" Regel nicht minimal! Da T_i zusammenhängend ist, folgt damit, dass je zwei Knoten aus V_i in T immer durch einen Pfad verbunden sind, der nur Kanten aus E_r enthält.

Zeitkomplexität: (mit $n = |V|, m = |E|$)

Sortieren	$m \log m = \mathcal{O}(m \log n)$
$\mathcal{O}(m)$ *Find*-Operationen	$\mathcal{O}(m \log n)$
$n - 1$ *Unions*	$\mathcal{O}(n \log n)$

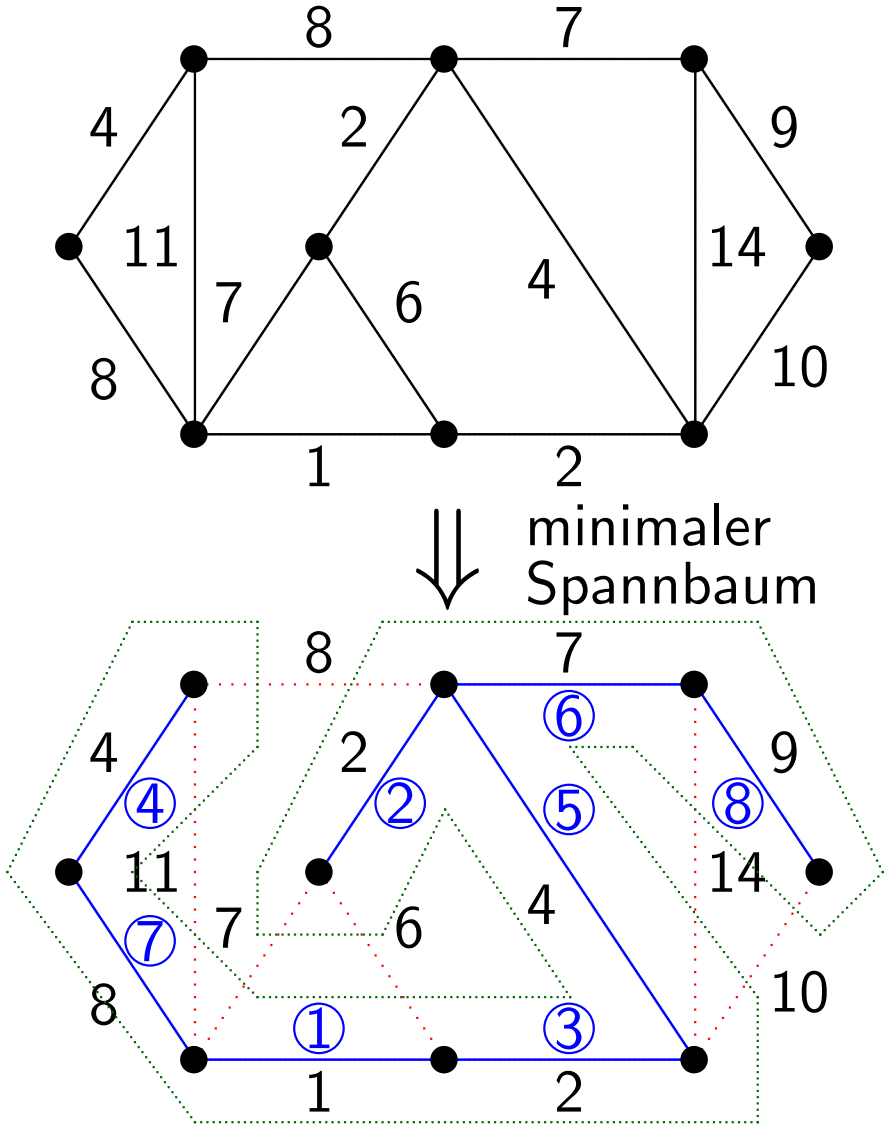
Satz 189

Kruskal's MSB-Algorithmus hat die Zeitkomplexität $\mathcal{O}((m + n) \log n)$.

Beweis:

s.o.

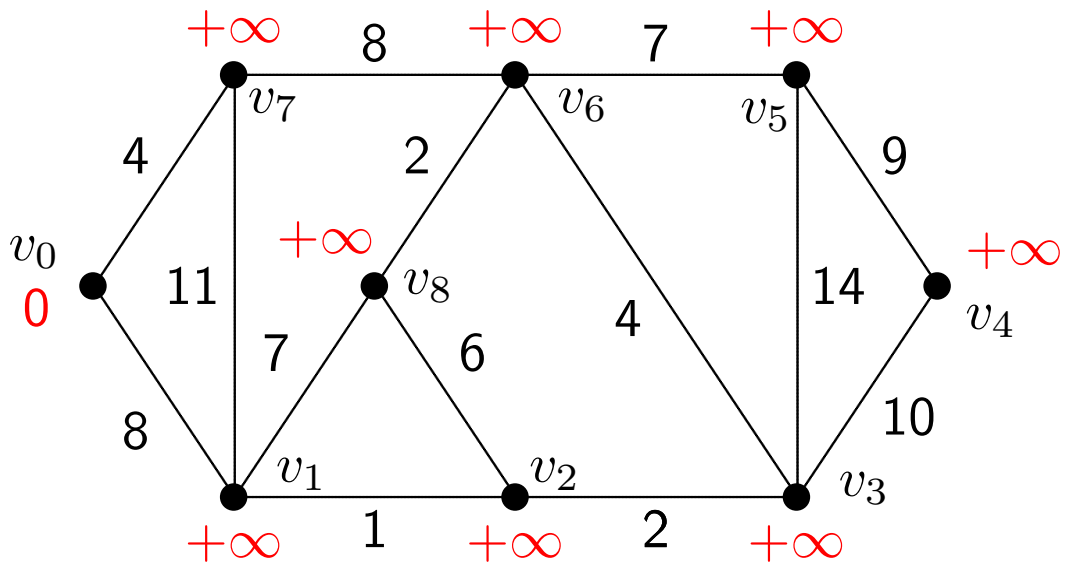
Beispiel 190 (Kruskals Algorithmus)



7.4 Prim's Algorithmus

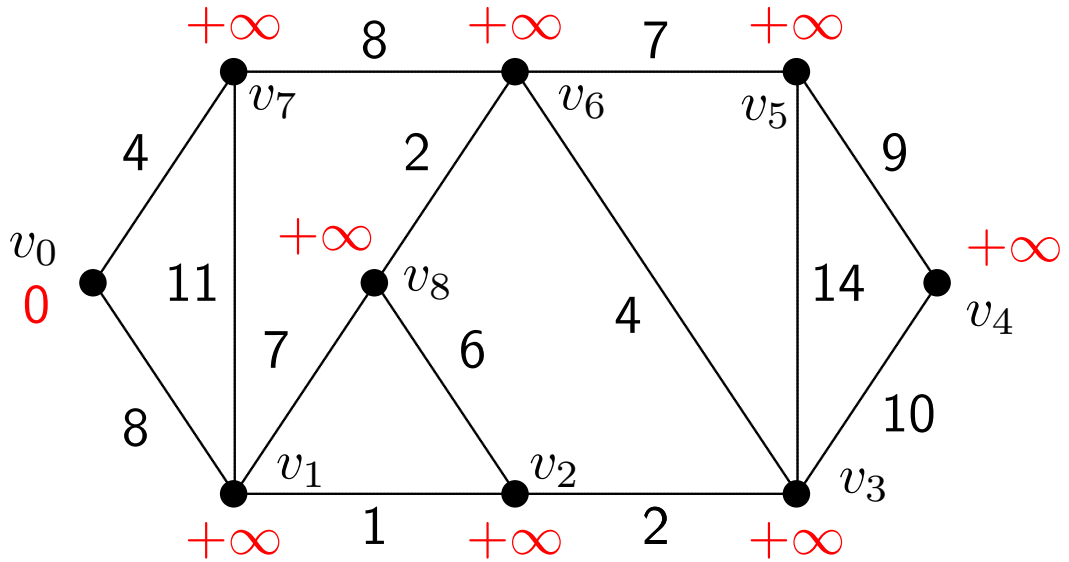
algorithm PRIM-MSB $(G, w) :=$
  initialisiere Priority Queue PQ mit Knotenmenge V und
    Schlüssel $+\infty$, $\forall v \in V$
  wähle Knoten r als Wurzel (beliebig)
  Schlüssel $k[r] := 0$
  Vorgänger$[r] :=$ **nil**
  while $PQ \neq \emptyset$ **do**
    $u :=$ *ExtractMin*(PQ)
    for alle Knoten v, die in G zu u benachbart sind **do**
      if $v \in PQ$ **and** $w(\{u, v\}) < k[v]$ **then**
        Vorgänger$[v] := u$
        DecreaseKey$(PQ, v, w(\{u, v\}))$
      fi
    od
  od

Beispiel 191 (Prim's Algorithmus)

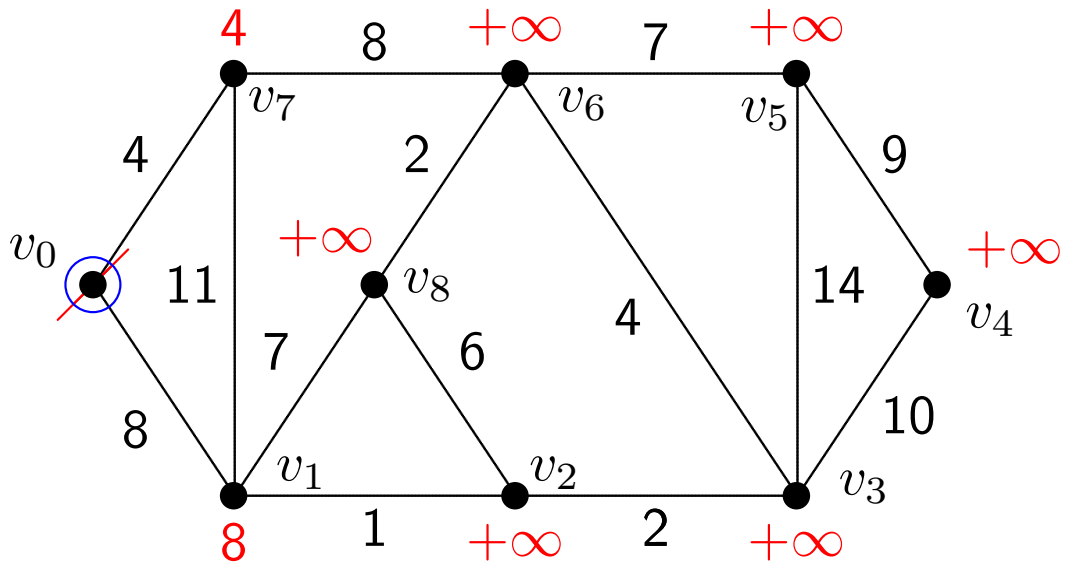


Ausgangszustand:
alle Schlüssel $= +\infty$
aktueller Knoten u: ◉
Startknoten: r $(= v_0)$

Beispiel 191 (Prim's Algorithmus)

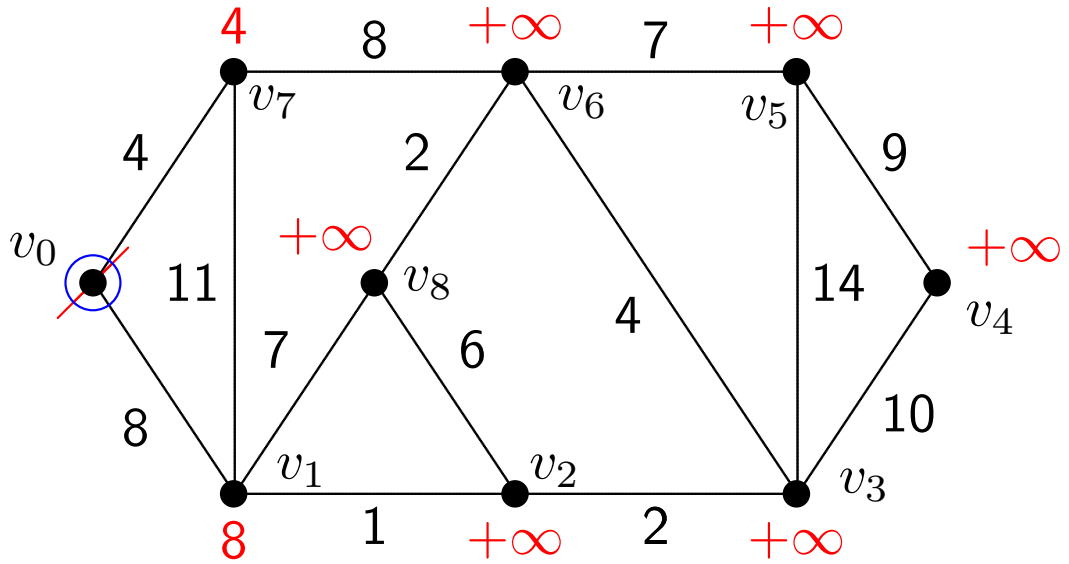


Ausgangszustand:
alle Schlüssel $= +\infty$
aktueller Knoten u: ◉
Startknoten: r $(= v_0)$

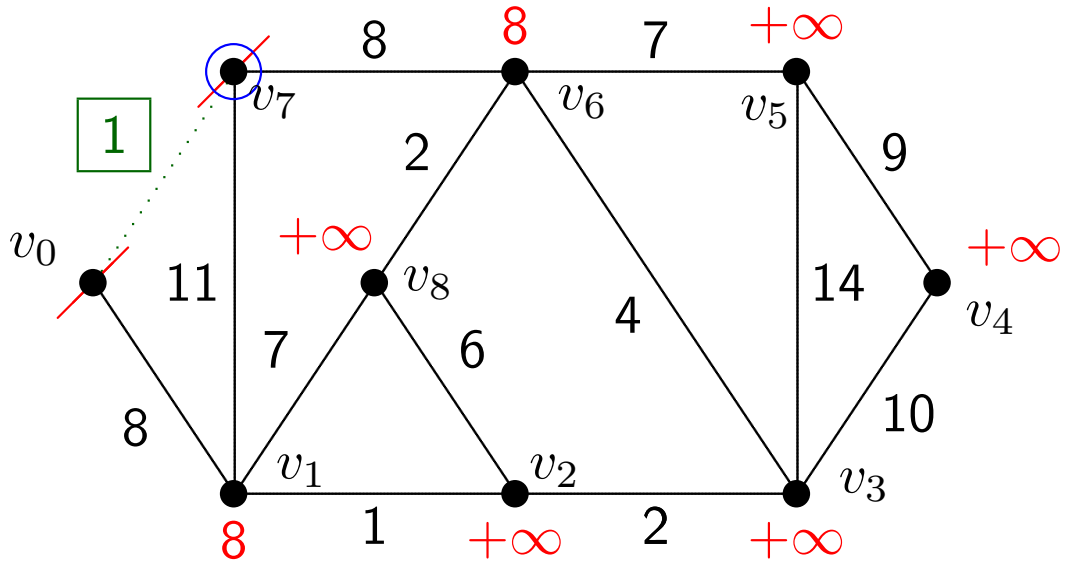


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_1 = 8, v_7 = 4)$

Beispiel 191 (Prim's Algorithmus)

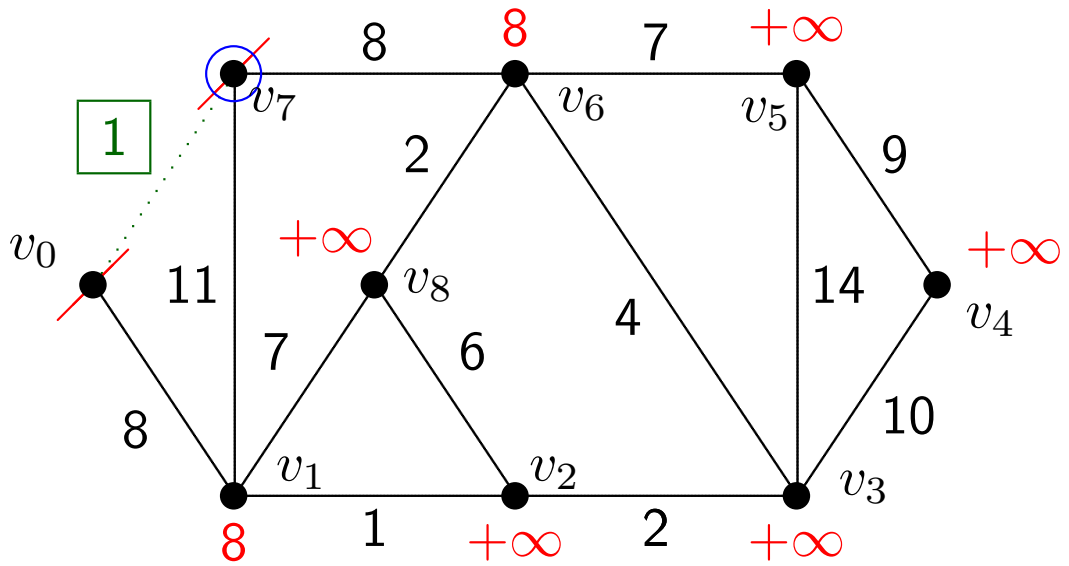


suche $u :=$ FindMin(PQ) und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit $w(\{u,v\}) <$ Schlüssel$[v]$:
$(v_1 = 8, v_7 = 4)$

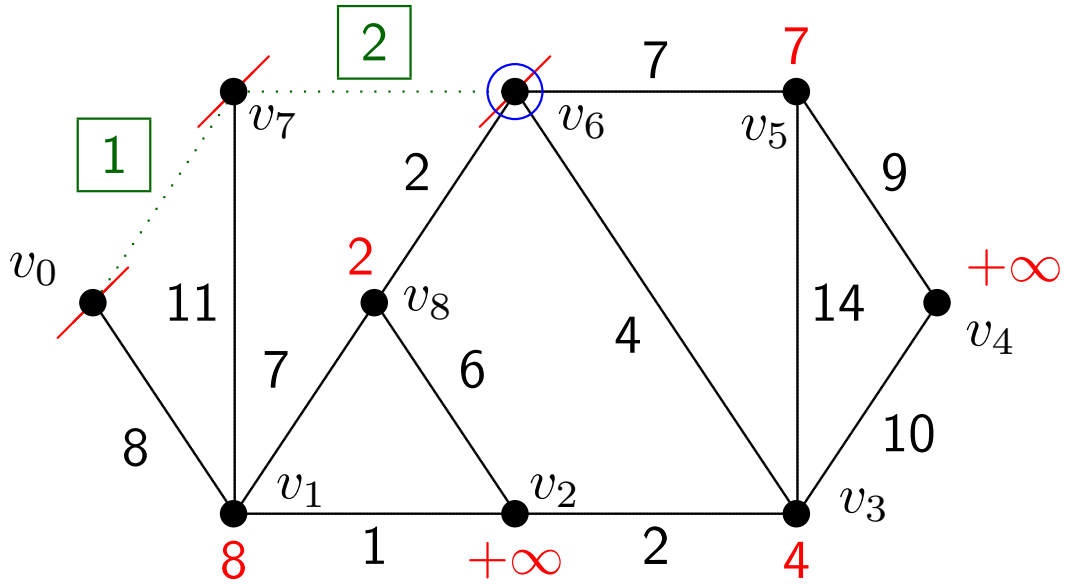


suche $u :=$ FindMin(PQ) und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit $w(\{u,v\}) <$ Schlüssel$[v]$:
$(v_6 = 8)$

Beispiel 191 (Prim's Algorithmus)

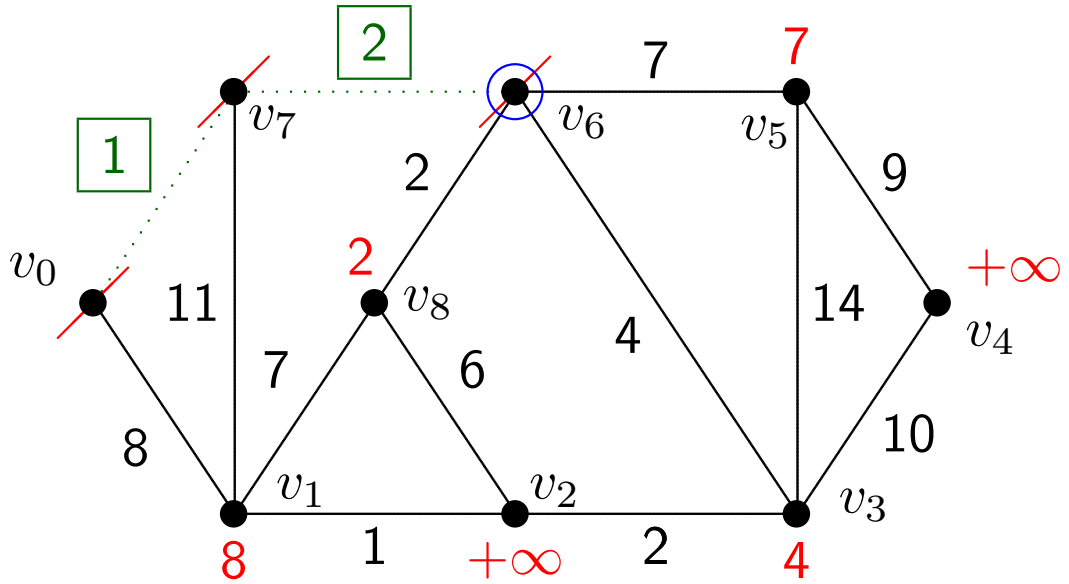


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_6 = 8)$

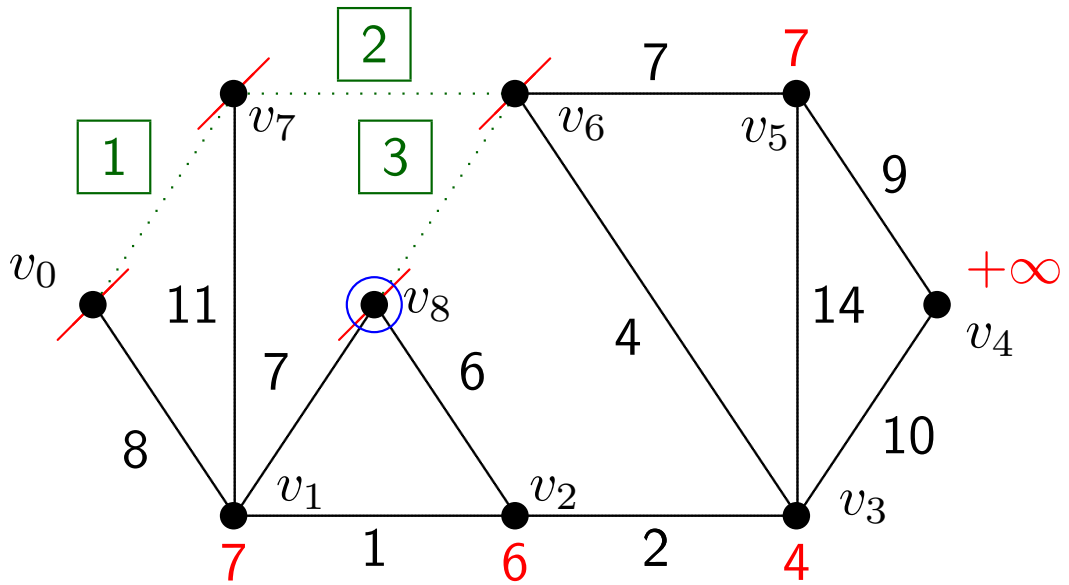


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_3 = 4, v_5 = 7, v_8 = 2)$

Beispiel 191 (Prim's Algorithmus)

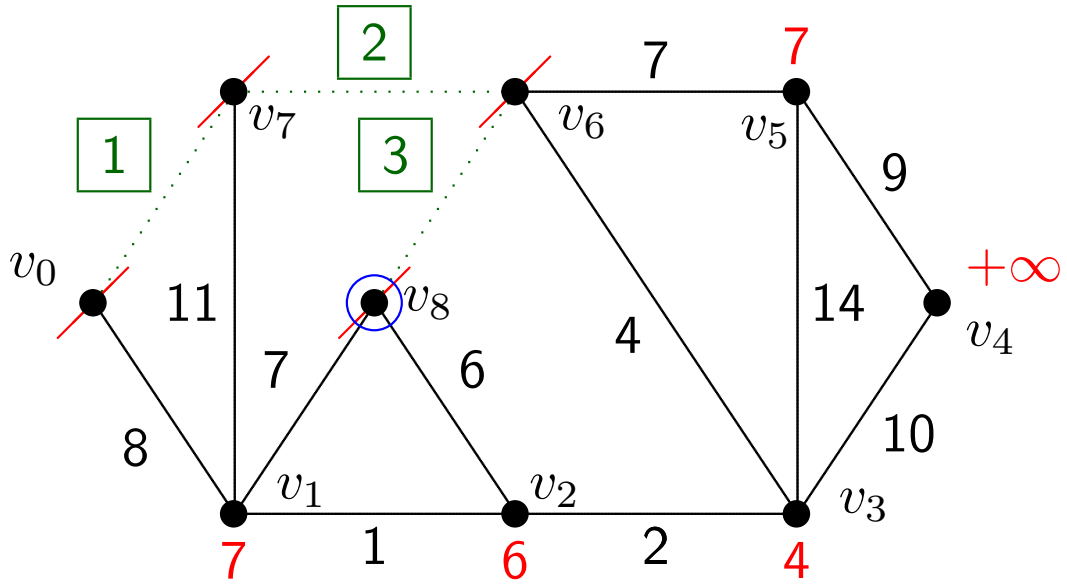


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_3 = 4, v_5 = 7, v_8 = 2)$

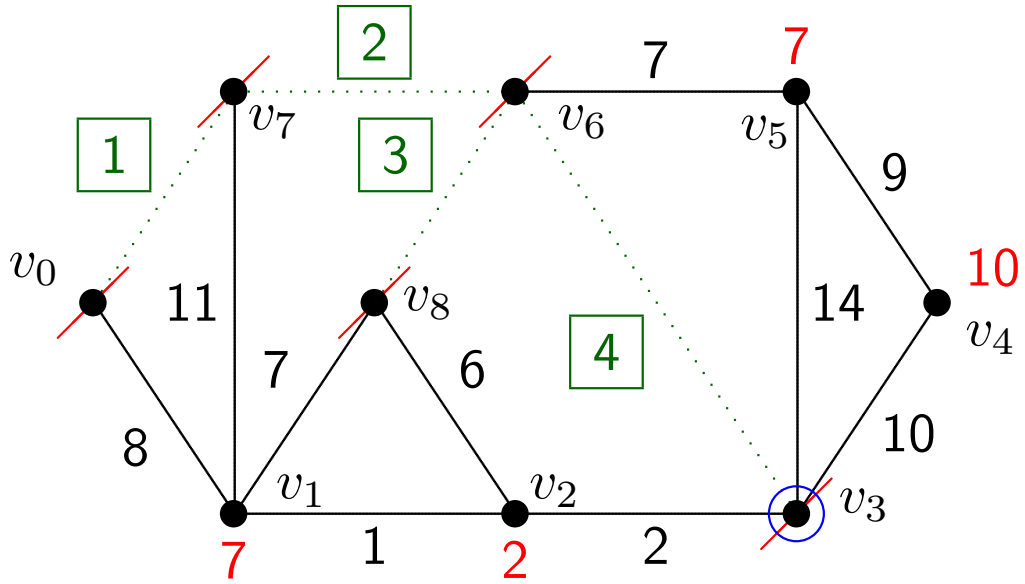


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_1 = 7, v_2 = 6)$

Beispiel 191 (Prim's Algorithmus)

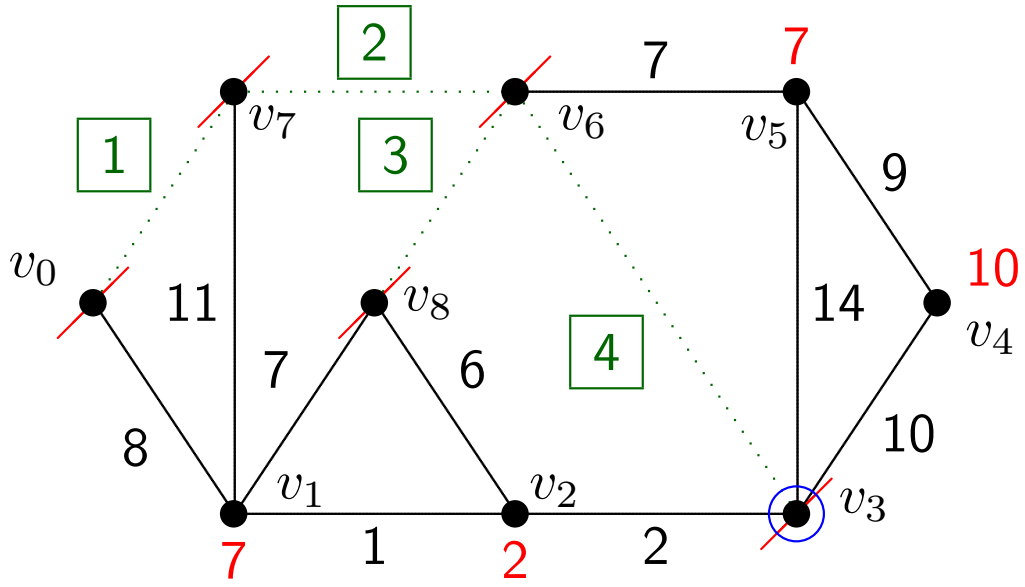


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_1 = 7, v_2 = 6)$

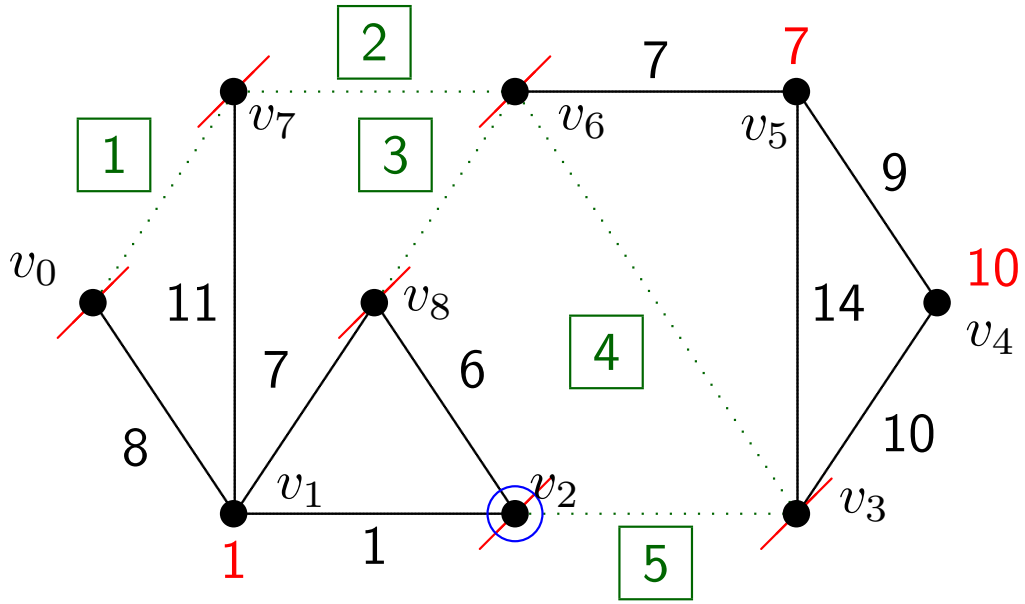


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_2 = 2, v_4 = 10)$

Beispiel 191 (Prim's Algorithmus)

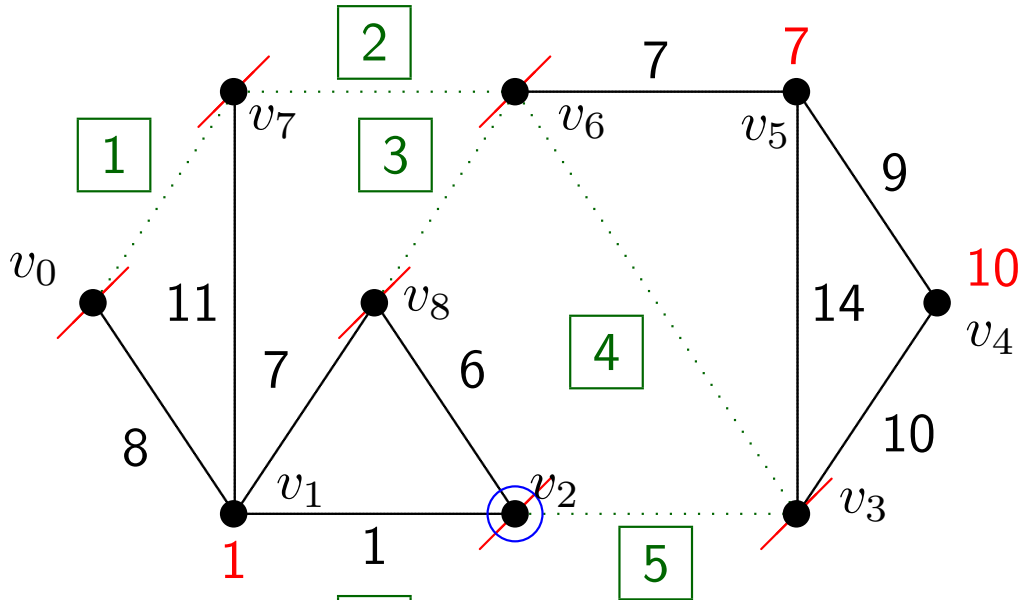


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_2 = 2, v_4 = 10)$

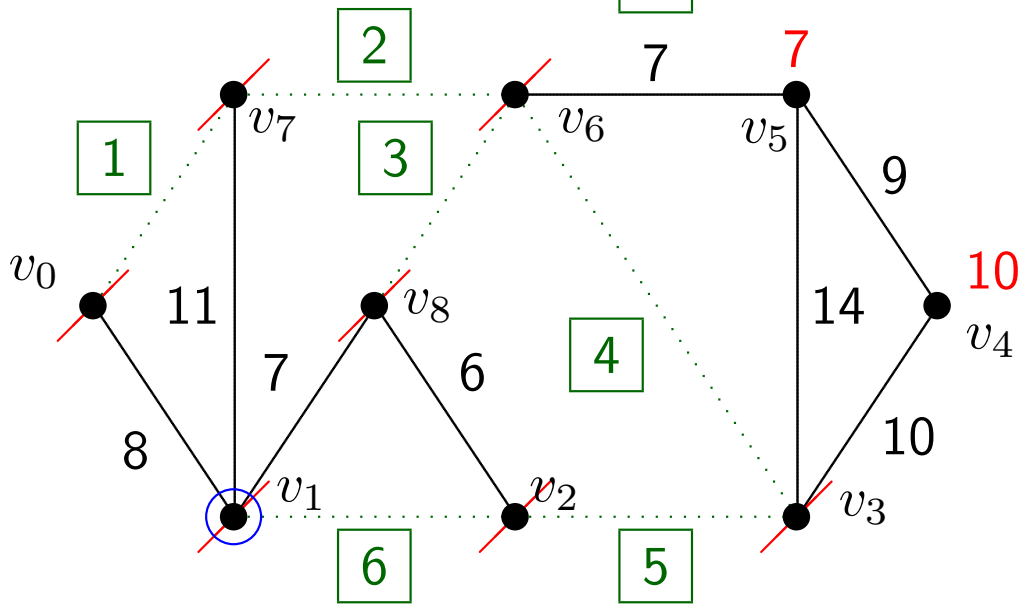


suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_1 = 1)$

Beispiel 191 (Prim's Algorithmus)

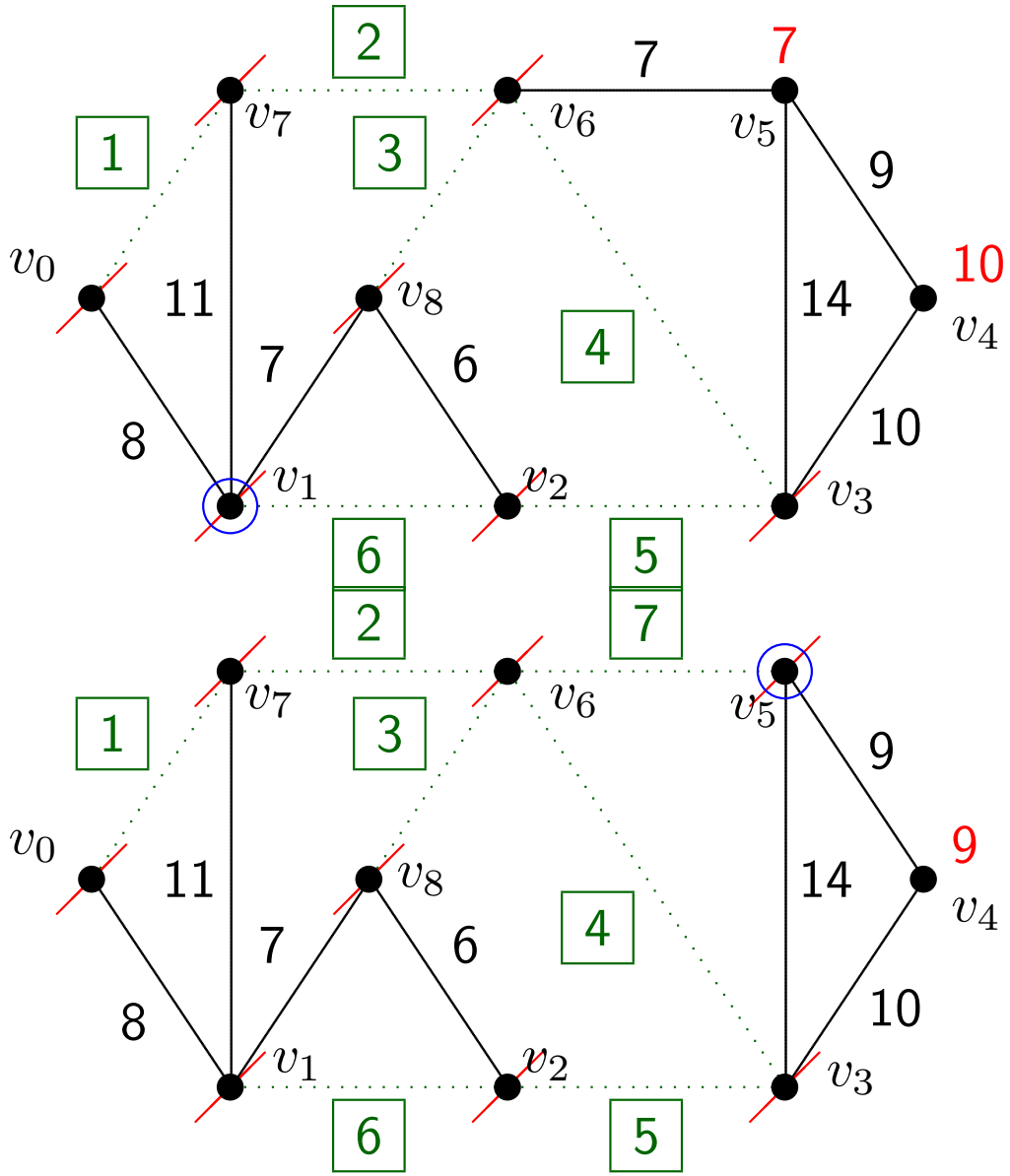


suche $u :=$ FindMin(PQ) und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit $\mathrm{w}(\{u,v\}) <$ Schlüssel$[v]$:
$(v_1 = 1)$



suche $u :=$ FindMin(PQ) und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit $\mathrm{w}(\{u,v\}) <$ Schlüssel$[v]$:
solche Nachbarn existieren nicht

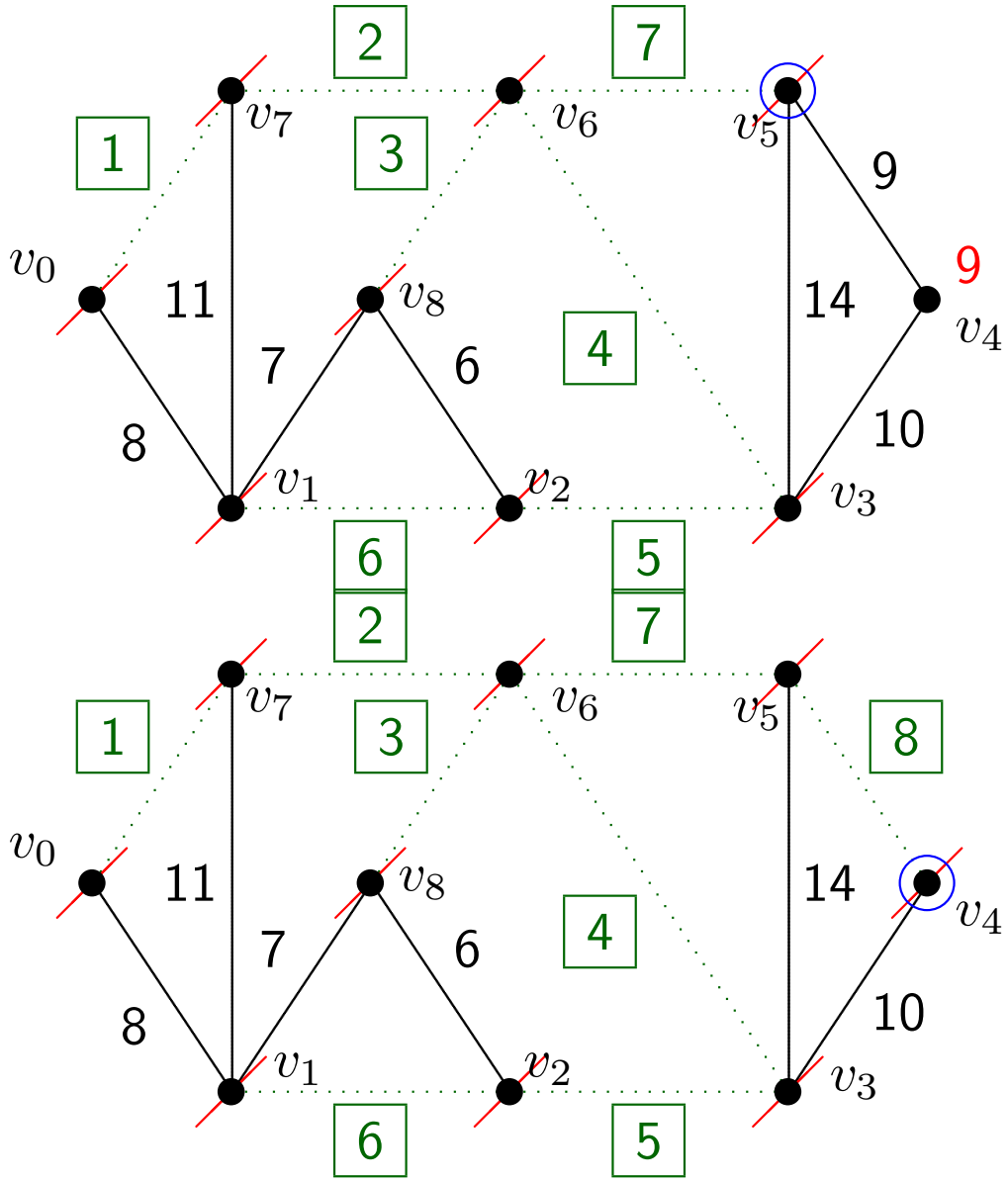
Beispiel 191 (Prim's Algorithmus)



suche $u := \text{FindMin}(PQ)$
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
$w(\{u, v\}) <$ Schlüssel$[v]$:
solche Nachbarn existieren nicht

suche $u := \text{FindMin}(PQ)$
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
$w(\{u, v\}) <$ Schlüssel$[v]$:
$(v_4 = 9)$

Beispiel 191 (Prim's Algorithmus)



suche $u :=$ FindMin(PQ)
und entferne u aus PQ
setze Schlüssel der Nachbarn in PQ mit
w$(\{u, v\}) <$ Schlüssel$[v]$:
$(v_4 = 9)$

Endzustand:
suche $u :=$ FindMin(PQ)
und entferne u aus PQ,
damit ist PQ leer und der Algorithmus beendet