Datenstrukturen & Algorithmen

Peppo Brambilla
Universität Bern
Frühling 2018

Prüfung

- Donnerstag, 07.06.2018, 16:15-18 Uhr
- ExWi, A6
- Unterlagen: zwei Blätter (4 Seiten) handschriftliche Notizen
  - Keine weiteren Unterlagen oder Hilfsmittel
- Wichtig: Anmeldung auf KSL zwingend

Graphenalgorithmen

- Minimale Spannbäume
- Kürzeste Pfade

Minimale Spannbäume

Beispielproblem

- Stadt modelliert durch Menge von Häusern und Strassen
- Jede Strasse verbindet genau 2 Häuser
- Kosten, um Strasse zwischen Haus u und v zu reparieren ist $w(u, v)$
- Ziel: repariere Strassen, so dass
  - alle Häuser verbunden
  - totale Reparaturkosten minimal

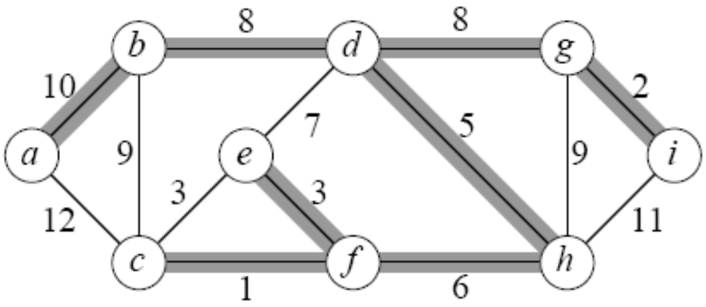
Minimale Spannbäume

Modellierung als Graph

- Zusammenhängender, ungerichteter Graph $G = (V, E)$
- Kantengewichte $w(u, v)$ für Kanten $(u, v) \in E$
- Finde Teilmenge T der Kanten so dass
  - T alle Knoten verbindet (T ist ein Spannbaum)
  - Kosten $w(T) = \sum_{(u,v) \in T} w(u, v)$ minimiert werden
- Spannbaum mit minimalen Kosten über alle Spannbäume ist minimaler Spannbaum (minimal spanning tree, MST)

Minimale Spannbäume

Example of such a graph *[edges in MST are shaded]* :



In this example, there is more than one MST. Replace edge (e, f) by (c, e). Get a different spanning tree with the same weight.

- Eigenschaften
  - $|V| - 1$ Kanten
  - Keine Zyklen
  - Nicht unbedingt eindeutig

Generischer Algorithmus

- Konstruiere Teilmenge A von Kanten
- Starte mit leerer Menge
- Füge eine Kante nach der anderen in A ein
- Invariante: A ist eine Teilmenge eines MST
  - Füge nur Kanten ein, die Invariante erhalten
  - Kante (u, v) heisst sicher für A, wenn $A \cup \{(u, v)\}$ auch Teilmenge eines MST ist
  - D.h., füge nur sichere Kanten hinzu
- Knackpunkt: sichere Kanten für A finden

Generischer Algorithmus

GENERIC-MST(G, w)

$A = \emptyset$
while A does not form a spanning tree
    find an edge (u, v) that is safe for A
    $A = A \cup \{(u, v)\}$
return A

Korrektheit des Algorithmus

- **Initialisierung**: Leere Menge erfüllt Invariante
- **Fortsetzung**: Da nur sichere Kanten hinzugefügt werden, bleibt A Teilmenge eines MST
- **Terminierung**: Alle zu A hinzugefügten Kanten sind im MST. Bei Terminierung der Schleife ist A ein Spannbaum, der auch minimal ist.

Auffinden von sicheren Kanten

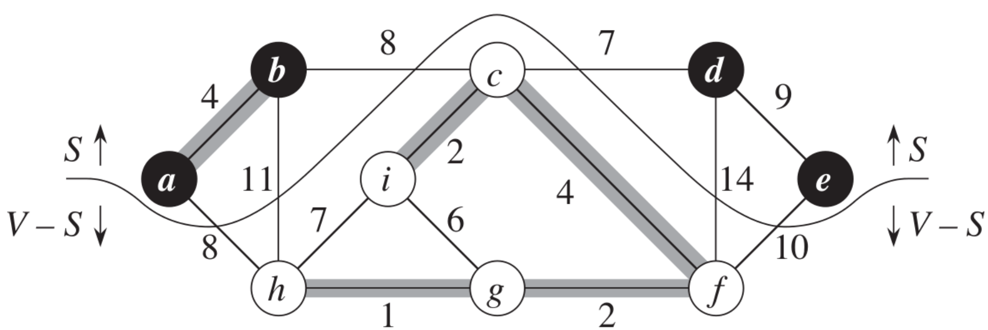
Definitionen: $G = (V, E)$ ein Graph, $S \subseteq V$, $A \subseteq E$

- Schnitt $(S, V - S)$: Partitionierung der Knoten V in disjunkte Mengen S und $V - S$
- Kante $(u, v) \in E$ kreuzt den Schnitt, falls ein Knoten in S und der andere in $V - S$
- Schnitt respektiert Kantenmenge A, wenn keine Kante in A den Schnitt kreuzt
- Kante ist eine leichte Kante, wenn sie einen Schnitt kreuzt und das kleinste Gewicht aller Kanten hat, die den Schnitt kreuzen

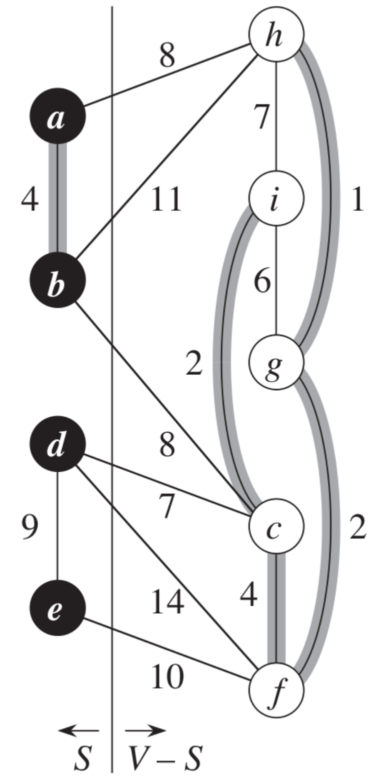
Schnitte

2 Darstellungen eines Schnittes S, der die markierten Kanten respektiert

- schwarze Knoten: Elemente von S
- weisse Knoten: Elemente von $V - S$
- Leichte Kante: (c, d)
  kann auch mehrere geben



(a)



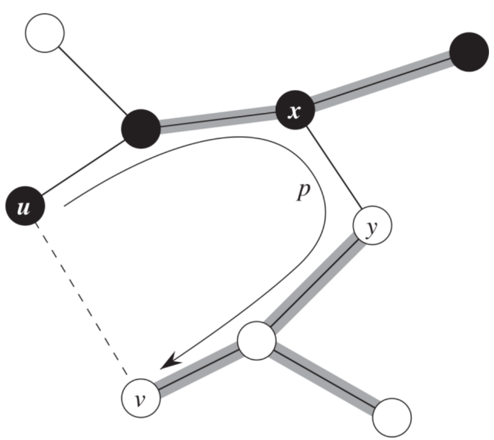
(b)

Auffinden von sicheren Kanten

Theorem: Sei A Teilmenge eines MST, $(S, V - S)$ Schnitt, der A respektiert, (u, v) leichte Kante, die $(S, V - S)$ kreuzt. Dann ist (u, v) sicher für A.

Beweis

- Sei T MST der A beinhaltet
- Wenn (u, v) nicht in T, zeige dass wir MST T' mit (u, v) konstruieren können
  - T': Wähle eine Kante (x, y) in T, die Schnitt kreuzt, ersetze mit (u, v)
  - Weil (u, v) eine leichte Kante ist, gilt $w(u, v) \leq w(x, y)$
- T' ist Spannbaum, Gewicht ist kleiner gleich T, also ist auch T' ein MST
- (u, v) ist sicher für A, weil A vereinigt mit (u, v) in T' ist



Generischer Algorithmus

Während Ablauf des Algorithmus gilt:

- $G_A = (V, A)$ ist Wald, d.h. Zusammenhangs-komponenten sind Bäume
  - Keine Zyklen in Zusammenhangskomponenten!
  - Bäume können aus einzelnen Knoten bestehen
- Jede für A sichere Kante verbindet neu zwei Komponenten
- Zu Beginn hat jeder Baum nur einen Knoten, d.h. es hat $|V|$ Bäume
- In jedem Schritt wird die Anzahl Bäume um eins reduziert, d.h. nach $|V| - 1$ Schritten bleibt ein Baum übrig, der MST

Algorithmus von Kruskal

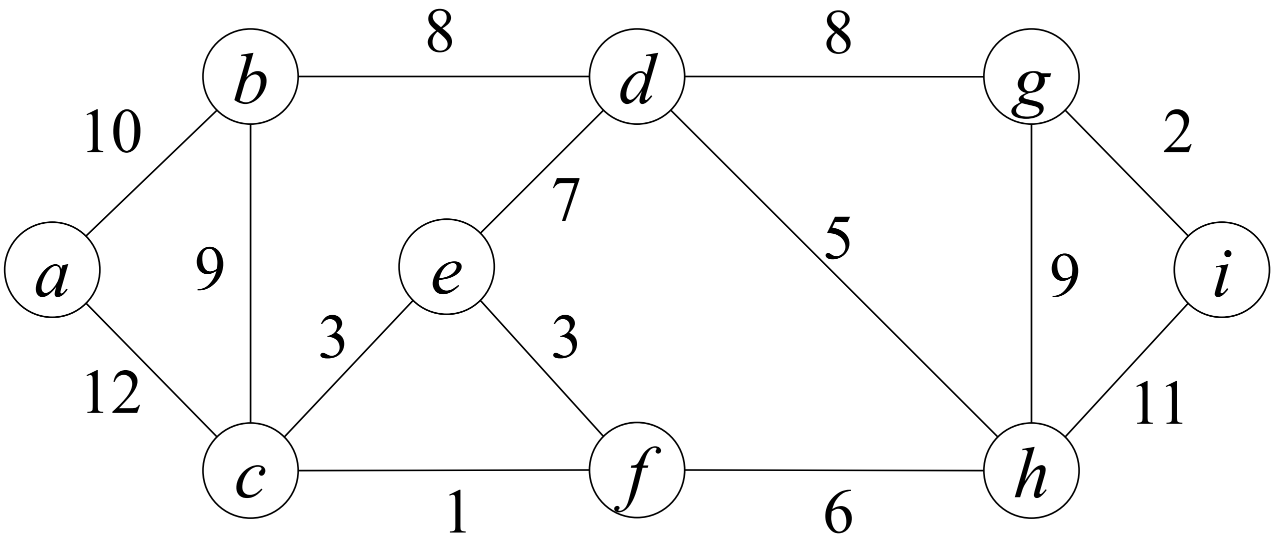
- Folgt direkt der Idee des generischen Algorithmus
- Finde in jedem Schritt sichere Kante, die dem Wald hinzugefügt werden kann
  - Wähle verbleibende Kante mit kleinstem Gewicht, die keinen Zyklus verursacht
    - Kanten werden nach Gewicht sortiert
    - Knoten der einzelnen Zusammenhangskomponenten werden in Mengen verwaltet.
      Kante (u, v) verursacht Zyklus, falls u in gleicher Menge wie v.

Algorithmus von Kruskal

MST-KRUSKAL(G, w)

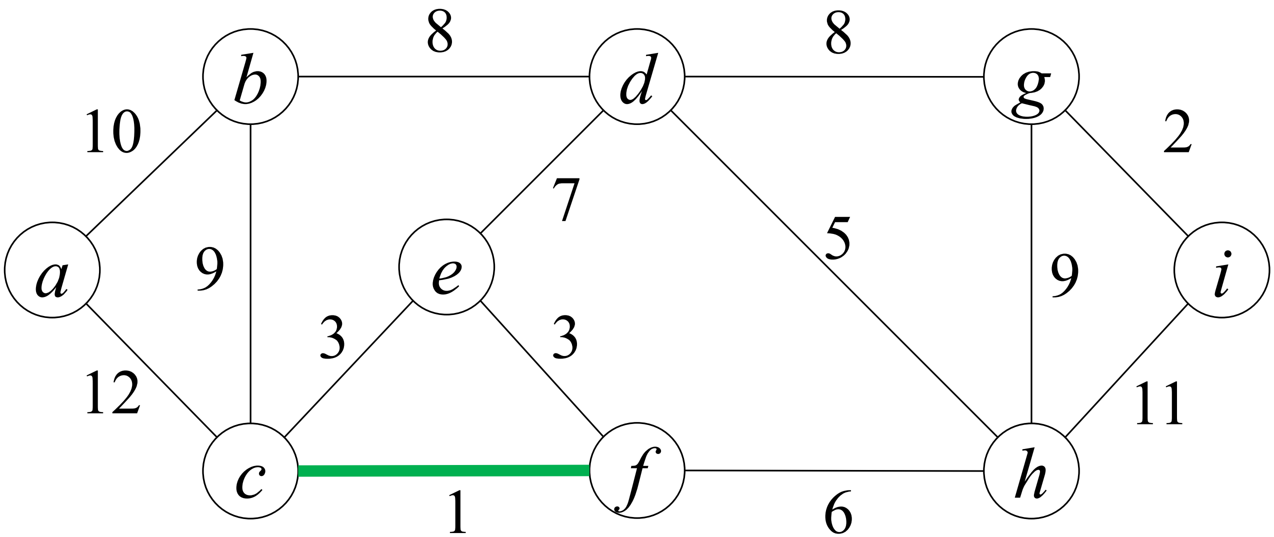
$A = \emptyset$
for each vertex $v \in G.V$
    MAKE-SET(v)
sort the edges of $G.E$ into nondecreasing order by weight w.
for each edge $(u, v) \in G.E$, taken in nondecreasing order by weight
    if FIND-SET$(u) \neq$ FIND-SET(v)
        $A = A \cup \{(u, v)\}$
        UNION(u, v)
return A

Algorithmus von Kruskal



Mengen: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (d, g), (b, c), (g, h), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal



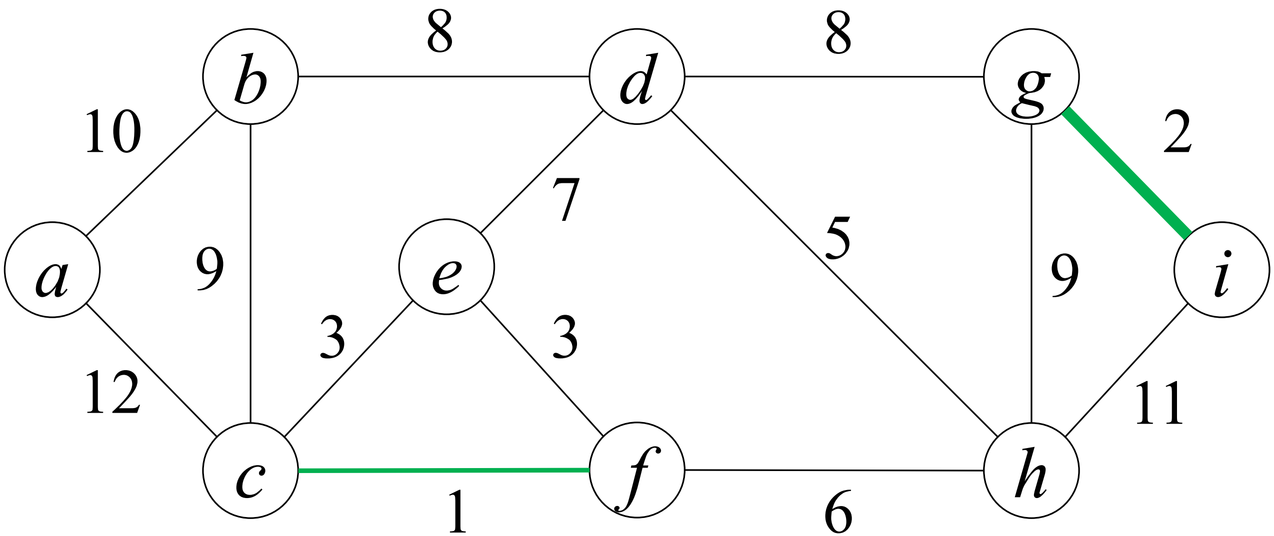
c und f liegen nicht in der gleichen Menge, also zu A hinzu und vereinigen.

Mengen: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{f\}, \{g\}, \{h\}, \{i\} \rightarrow \{a\}, \{b\}, \{c, f\}, \{d\}, \{e\}, \{g\}, \{h\}, \{i\}$

Sortierte Kanten:

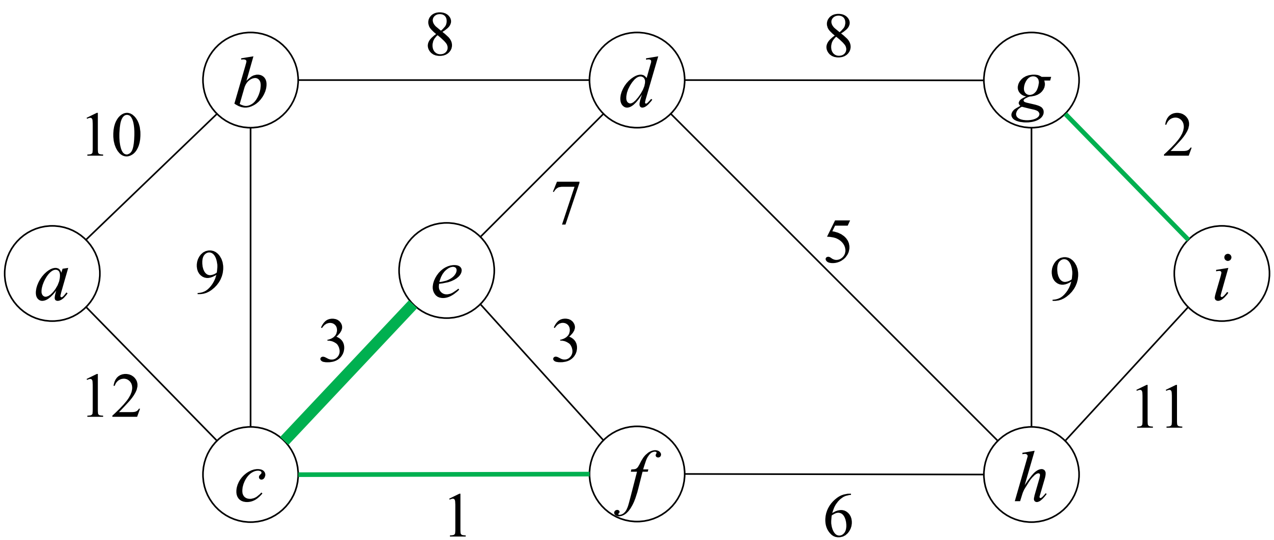
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (d, g), (b, c), (g, h), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal



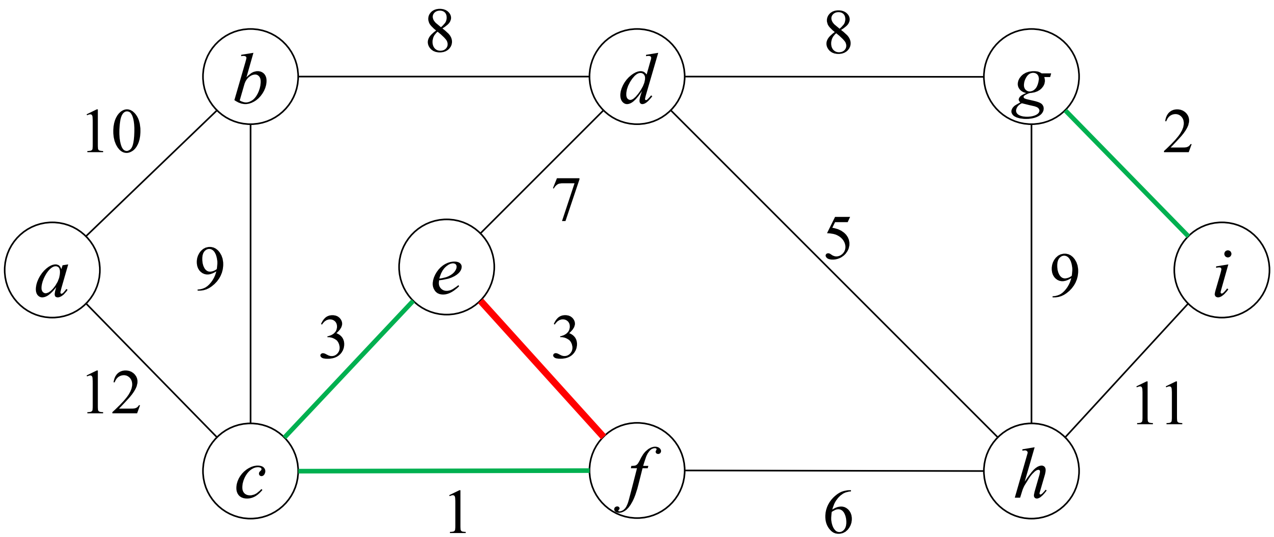
Mengen: $\{a\},\{b\},\{c,f\},\{d\},\{e\},\{g\},\{h\},\{i\} \rightarrow \{a\},\{b\},\{c,f\},\{d\},\{e\},\{g,i\},\{h\}$
Sortierte Kanten:
$(c,f),(g,i),(c,e),(e,f),(d,h),(f,h),(d,e),(b,d),(d,g),(b,c),(g,h),(a,b),(h,i),(a,c)$

Algorithmus von Kruskal



Mengen: $\{a\}, \{b\}, \{c, f\}, \{d\}, \{e\}, \{g, i\}, \{h\}$ → $\{a\}, \{b\}, \{c, e, f\}, \{d\}, \{g, i\}, \{h\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (d, g), (b, c), (g, h), (a, b), (h, i), (a, c)$

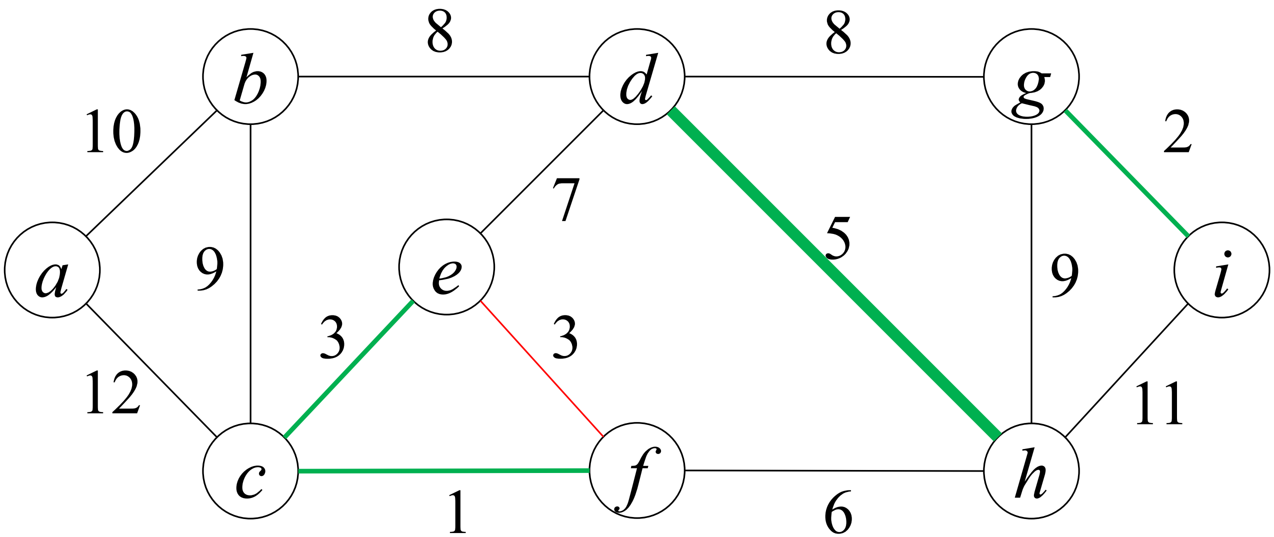
Algorithmus von Kruskal



überspringen

Mengen: $\{a\}, \{b\}, \{c, \boldsymbol{e}, \boldsymbol{f}\}, \{d\}, \{g, i\}, \{h\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (\boldsymbol{e}, \boldsymbol{f}), (d, h), (f, h), (d, e), (b, d), (d, g), (b, c), (g, h), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal

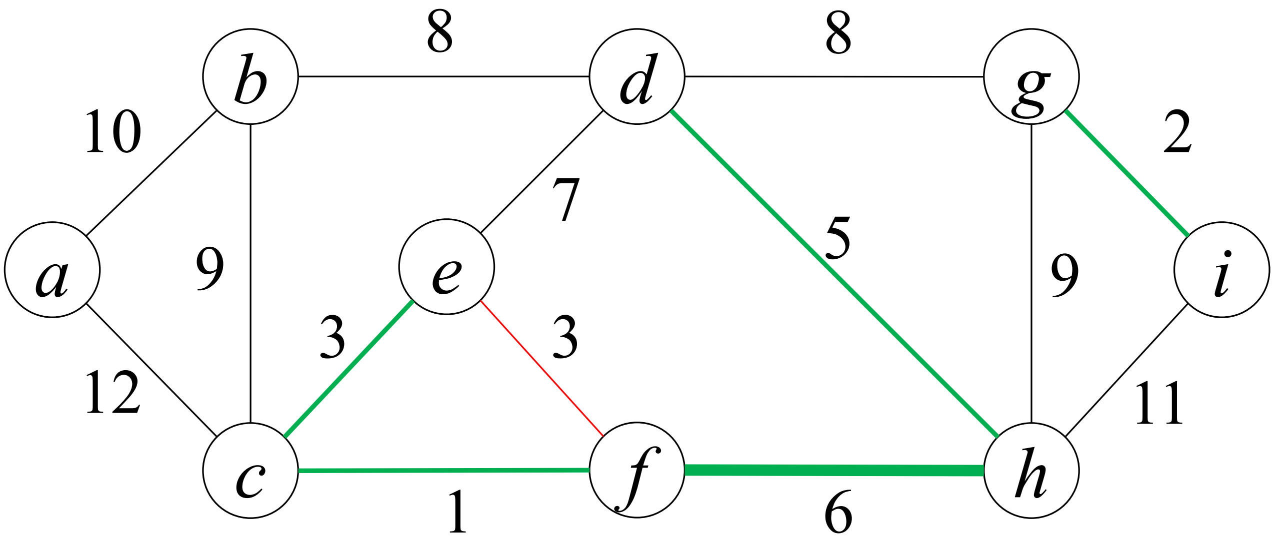


Mengen: $\{a\},\{b\},\{c,e,f\},\{\mathbf{d}\},\{g,i\},\{\mathbf{h}\} \rightarrow \{a\},\{b\},\{c,e,f\},\{\mathbf{d},\mathbf{h}\},\{g,i\}$

Sortierte Kanten:

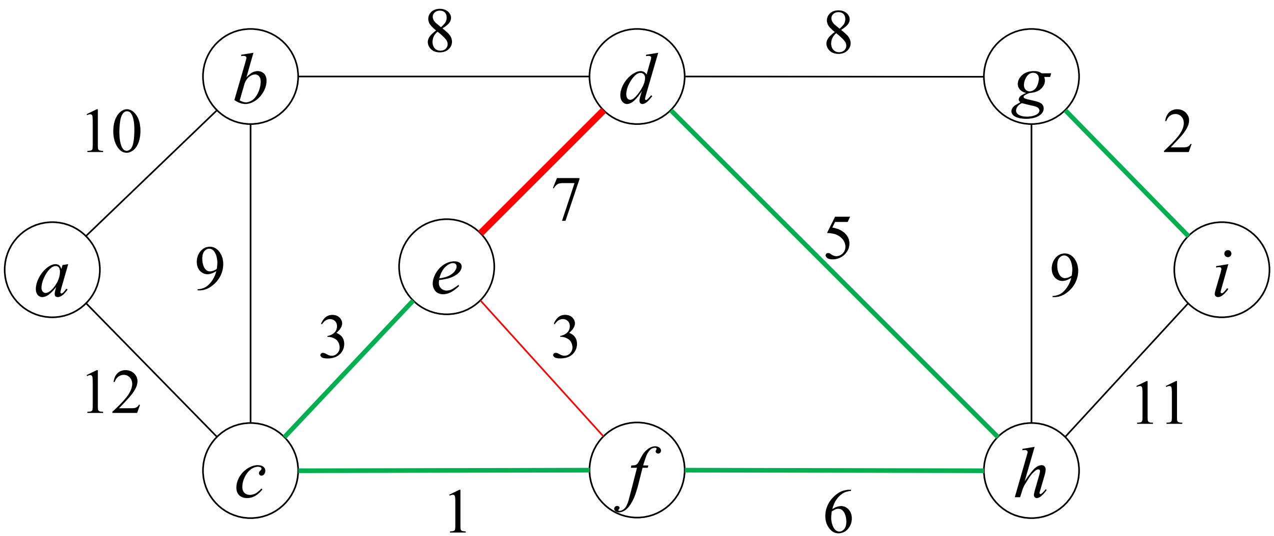
$(c,f),(g,i),(c,e),(e,f),(\mathbf{d},\mathbf{h}),(f,h),(d,e),(b,d),(d,g),(b,c),(g,h),(a,b),(h,i),(a,c)$

Algorithmus von Kruskal



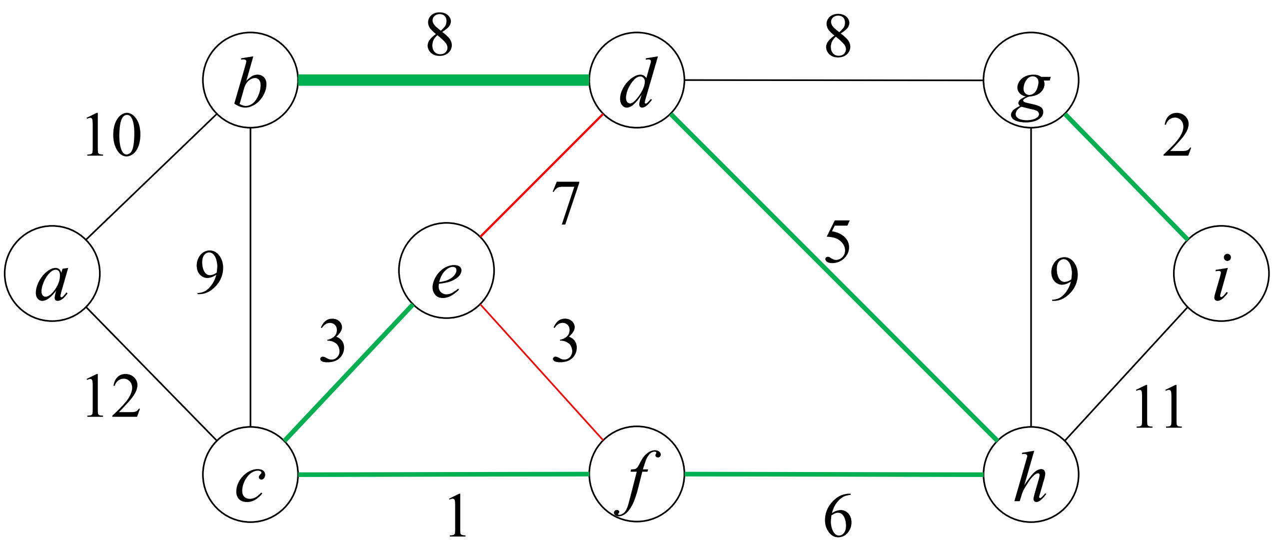
Mengen: $\{a\},\{b\},\{c,e,\boldsymbol{f}\},\{d,\boldsymbol{h}\},\{g,i\}$ → $\{a\},\{b\},\{c,e,\boldsymbol{f},d,\boldsymbol{h}\},\{g,i\}$
Sortierte Kanten:
$(c,f),(g,i),(c,e),(e,f),(d,h),(\boldsymbol{f},\boldsymbol{h}),(d,e),(b,d),(d,g),(b,c),(g,h),(a,b),(h,i),(a,c)$

Algorithmus von Kruskal



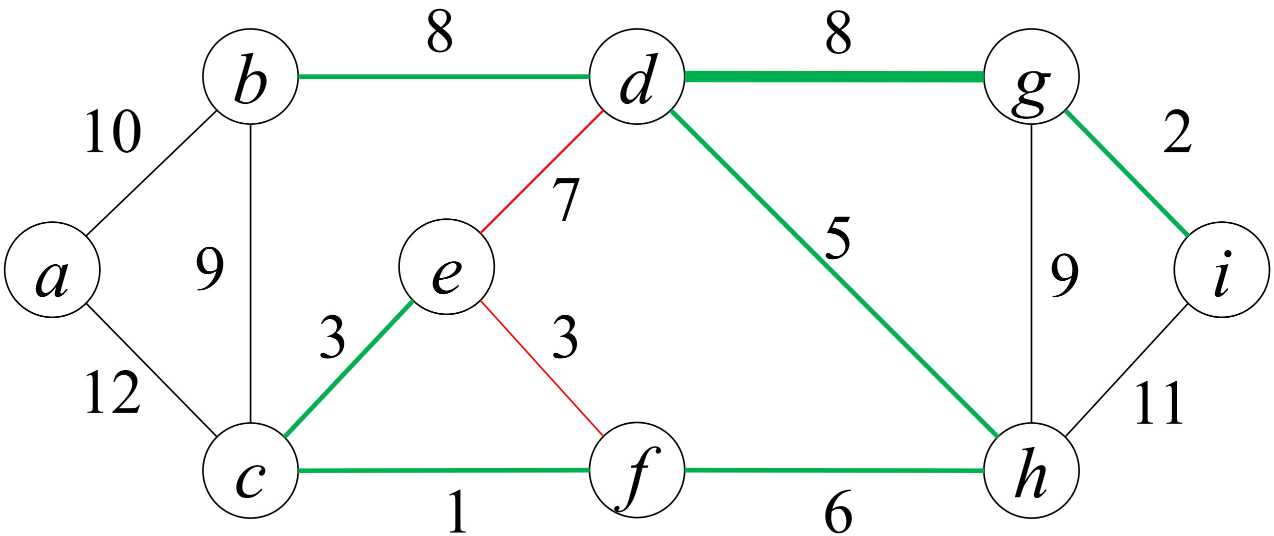
Mengen: $\{a\}, \{b\}, \{c, \mathbf{d}, \mathbf{e}, f, h\}, \{g, i\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), \mathbf{(d, e)}, (b, d), (d, g), (b, c), (g, h), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal



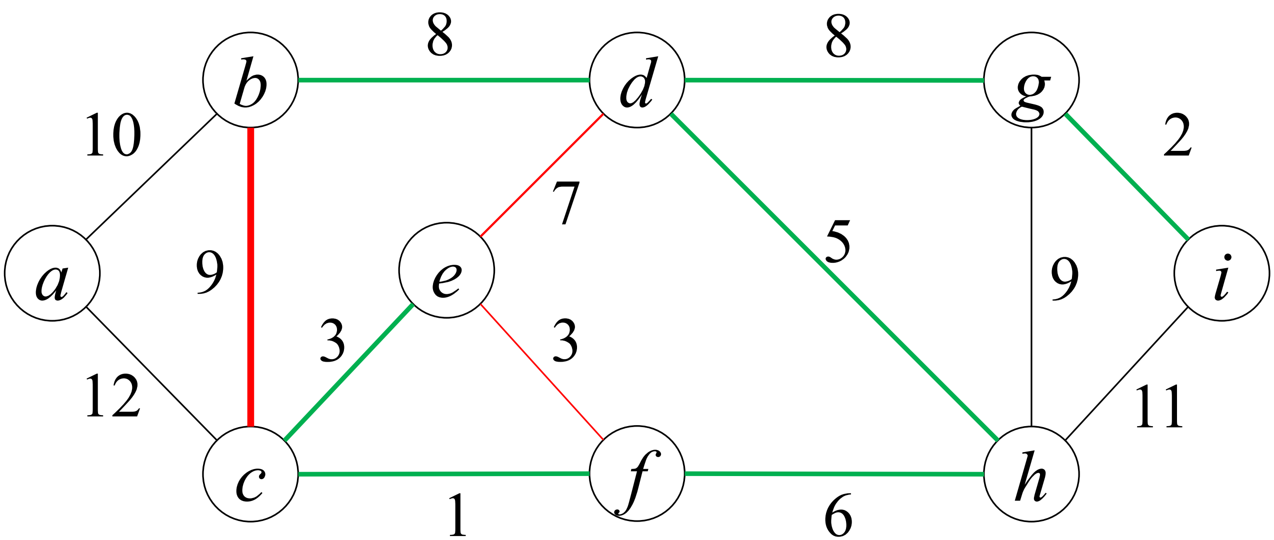
Mengen: $\{a\}, \{\boldsymbol{b}\}, \{c, \boldsymbol{d}, e, f, h\}, \{g, i\}$ → $\{a\}, \{\boldsymbol{b}, c, \boldsymbol{d}, e, f, h\}, \{g, i\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (\boldsymbol{b}, \boldsymbol{d}), (d, g), (b, c), (g, h), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal



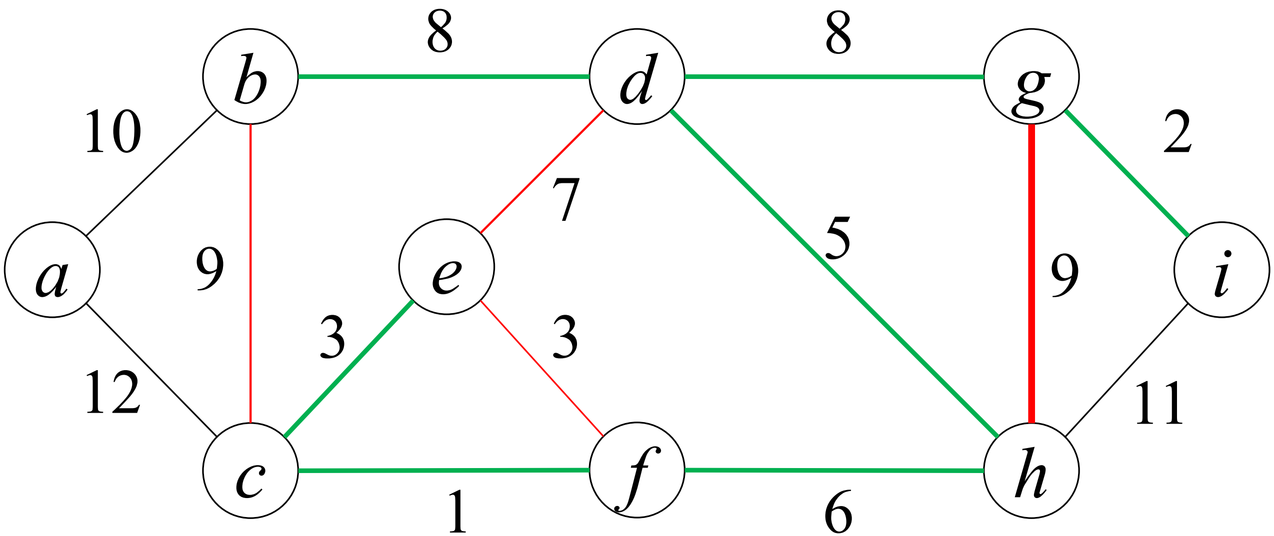
Mengen: $\{a\}, \{b, c, \mathbf{d}, e, f, h\}, \{\mathbf{g}, i\}$ → $\{a\}, \{b, c, \mathbf{d}, e, f, \mathbf{g}, h, i\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (\mathbf{d}, \mathbf{g}), (b, c), (g, h), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal



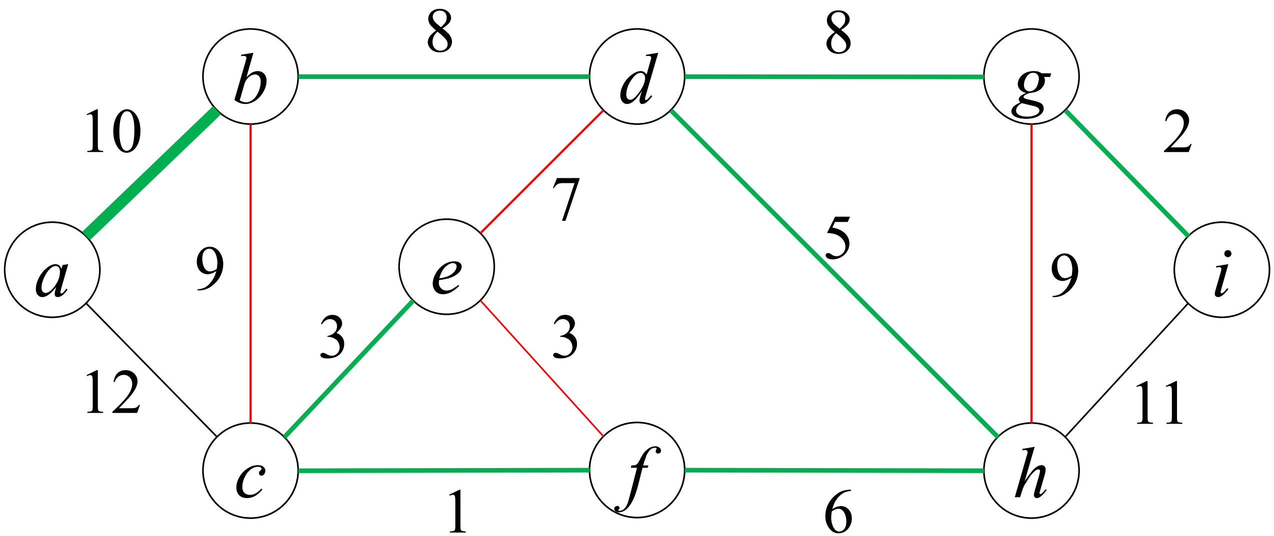
Mengen: $\{a\}, \{\mathbf{b}, \mathbf{c}, d, e, f, g, h, i\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (d, g), (\mathbf{b}, \mathbf{c}), (g, h), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal



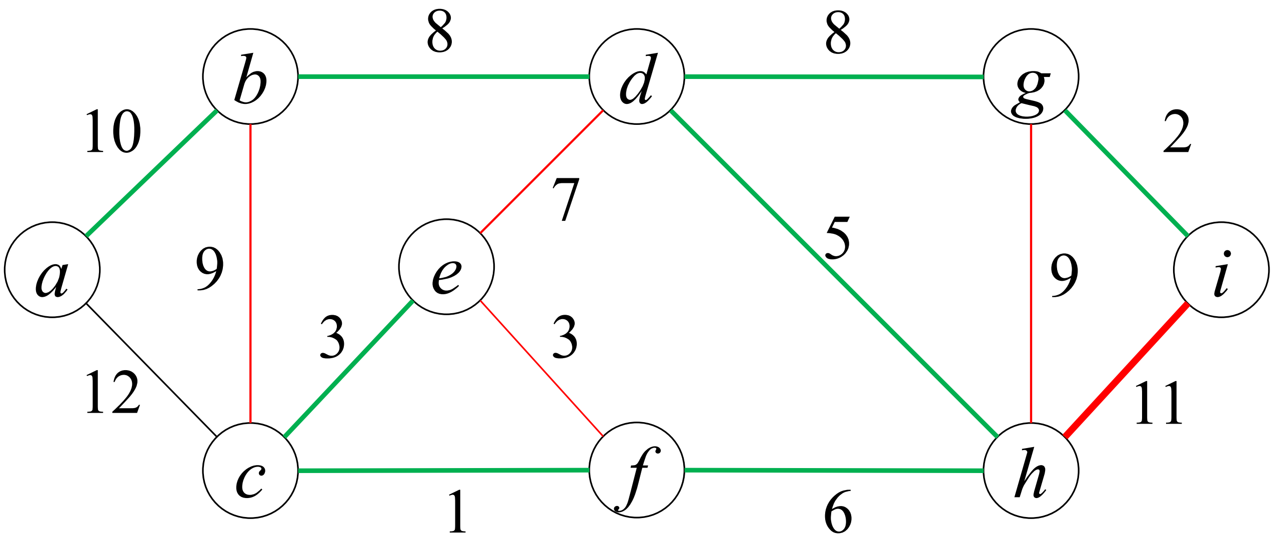
Mengen: $\{a\}, \{b, c, d, e, f, \boldsymbol{g}, \boldsymbol{h}, i\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (d, g), (b, c), (\boldsymbol{g}, \boldsymbol{h}), (a, b), (h, i), (a, c)$

Algorithmus von Kruskal



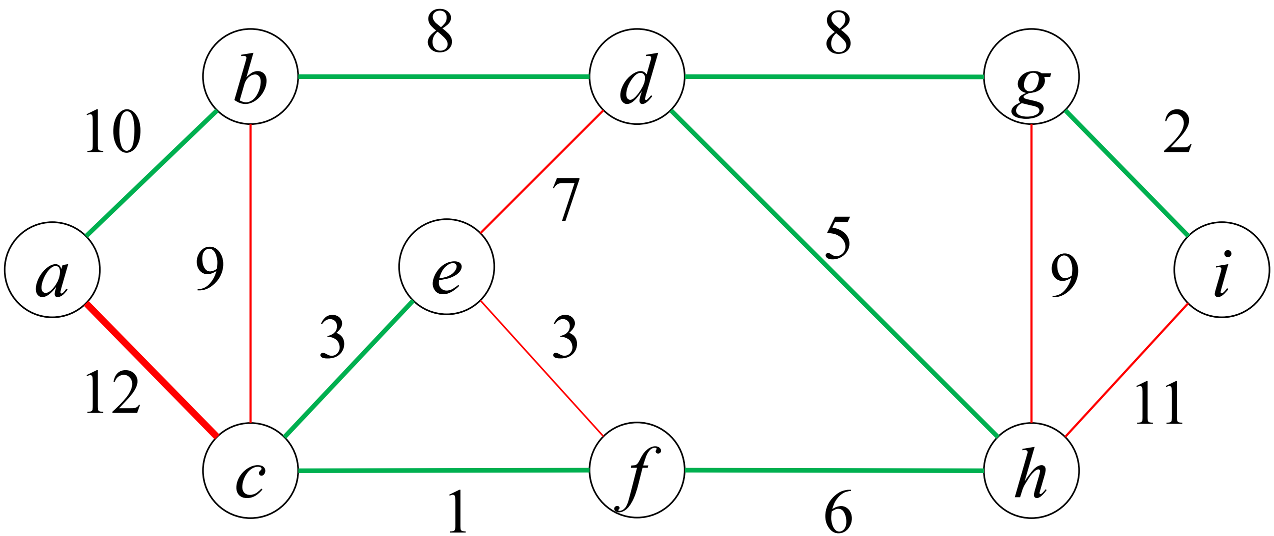
Mengen: $\{\boldsymbol{a}\}, \{\boldsymbol{b}, c, d, e, f, g, h, i\} \rightarrow \{\boldsymbol{a}, \boldsymbol{b}, c, d, e, f, g, h, i\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (d, g), (b, c), (g, h), (\boldsymbol{a}, \boldsymbol{b}), (h, i), (a, c)$

Algorithmus von Kruskal



Mengen: $\{a, b, c, d, e, f, g, \boldsymbol{h}, \boldsymbol{i}\}$
Sortierte Kanten:
$(c, f), (g, i), (c, e), (e, f), (d, h), (f, h), (d, e), (b, d), (d, g), (b, c), (g, h), (a, b), (\boldsymbol{h}, \boldsymbol{i}), (a, c)$

Algorithmus von Kruskal

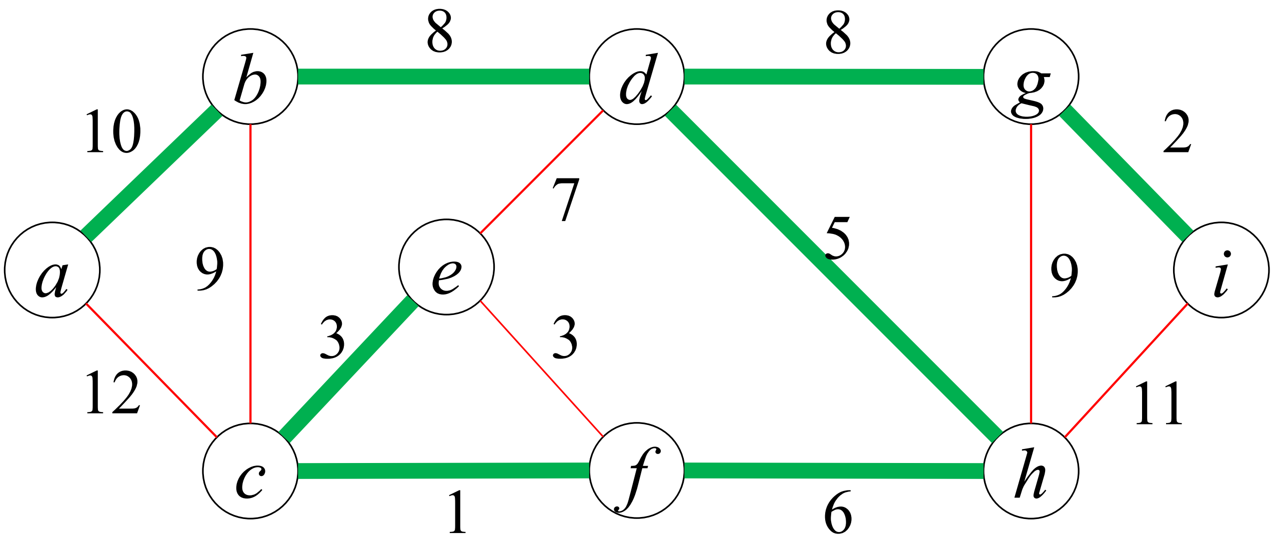


Mengen: $\{\boldsymbol{a}, b, \boldsymbol{c}, d, e, f, g, h, i\}$

Sortierte Kanten:

$(c,f), (g,i), (c,e), (e,f), (d,h), (f,h), (d,e), (b,d), (d,g), (b,c), (g,h), (a,b), (h,i), (\boldsymbol{a}, \boldsymbol{c})$

Algorithmus von Kruskal



Algorithmus von Kruskal

MST-KRUSKAL(G, w)

```
A = ∅
for each vertex v ∈ G.V
    MAKE-SET(v)
sort the edges of G.E into nondecreasing order by weight w.
for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
    if FIND-SET(u) ≠ FIND-SET(v)
        A = A ∪ {(u, v)}
        UNION(u, v)
return A
```

- Analyse

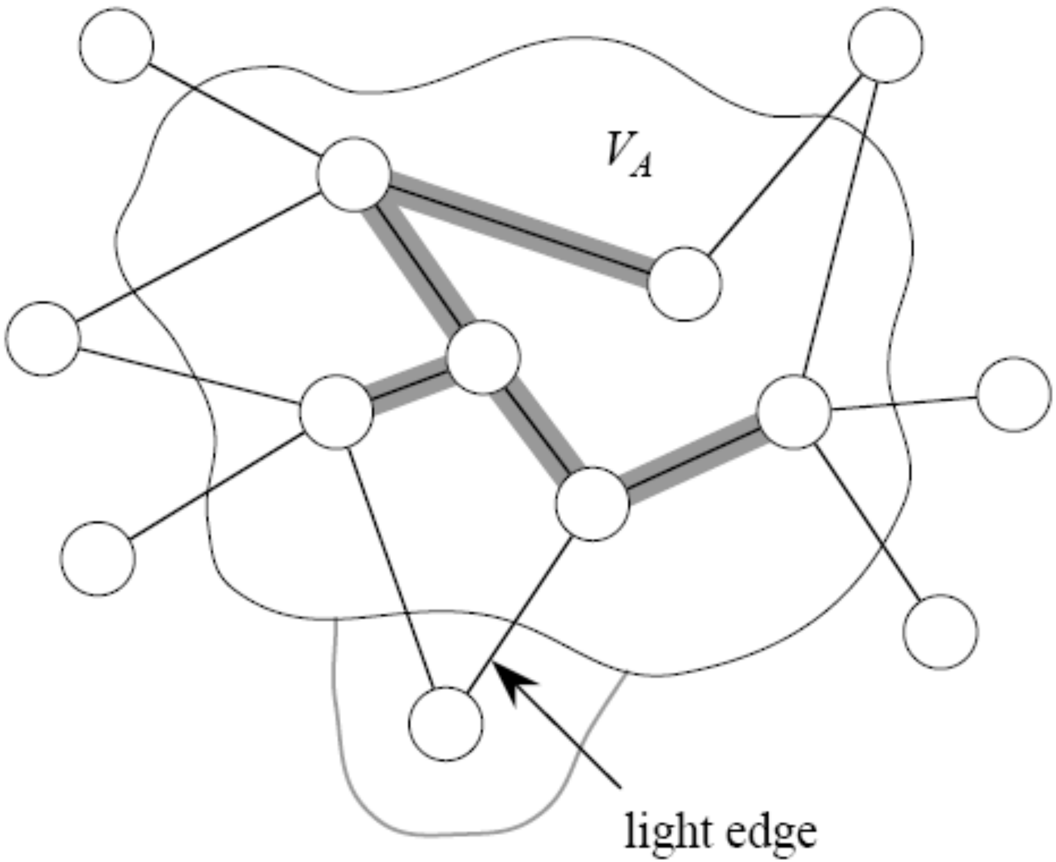
  (diese Operationen nicht besprochen in Vorlesung!)

  - A initialisieren: $O(1)$
  - Erste **for**-Schleife: $|V|$ MAKE-SETs
  - E sortieren: $O(E \lg E)$
  - Zweite **for**-Schleife: $O(E)$ FIND-SETs und UNIONs
- Total: $O(E \lg E)$

Algorithmus von Prim

- Teilmenge von Kanten A des MST ist immer ein einziger Baum
- In jedem Schritt, finde leichte Kante, die Baum mit einem neuem Knoten verbindet
  - Sei V_A Menge der Knoten, die in A erreichbar
  - Finde leichte Kante über Schnitt $(V_A, V - V_A)$

Algorithmus von Prim



Algorithmus von Prim

- Verwende Prioritätswarteschlange, um leichte Kante zu finden
  - Prioritätswarteschlange enthält Knoten, die in A nicht erreichbar sind, d.h. Knotenmenge $V - V_A$
  - Schlüssel ist Kante mit geringstem Gewicht zu einem Knoten in V_A
  - Schlüssel ist unendlich falls keine Kante zu V_A existiert
- Jeder Knoten v, der hinzugenommen wird, speichert seinen Vater $v.\pi$ im Baum

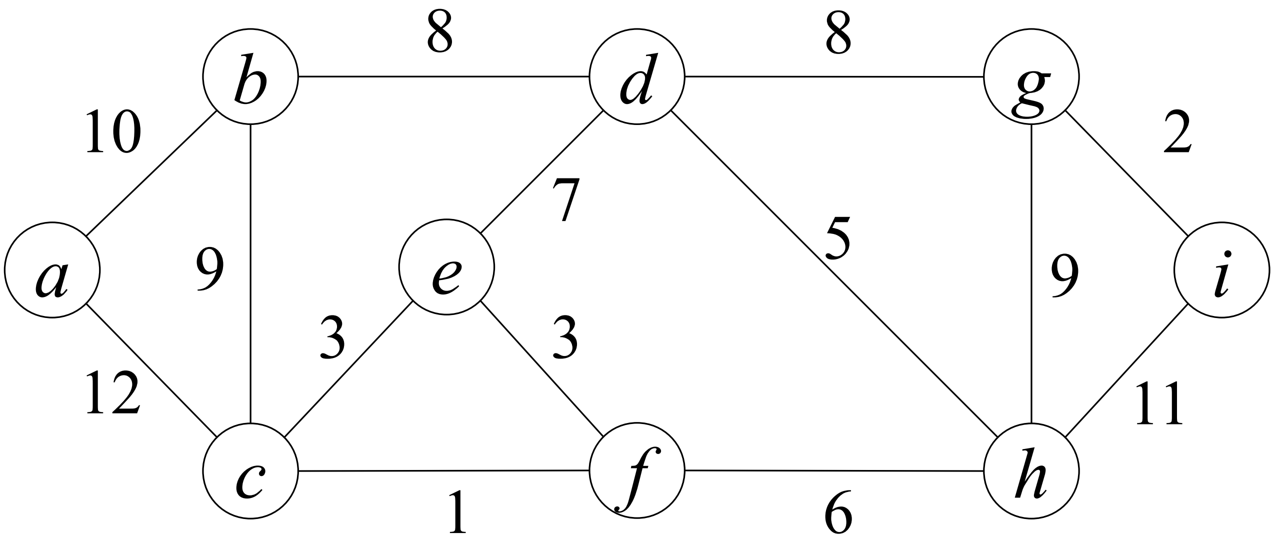
Algorithmus von Prim

- Starte mit beliebiger Wurzel r

MST-PRIM(G, w, r)

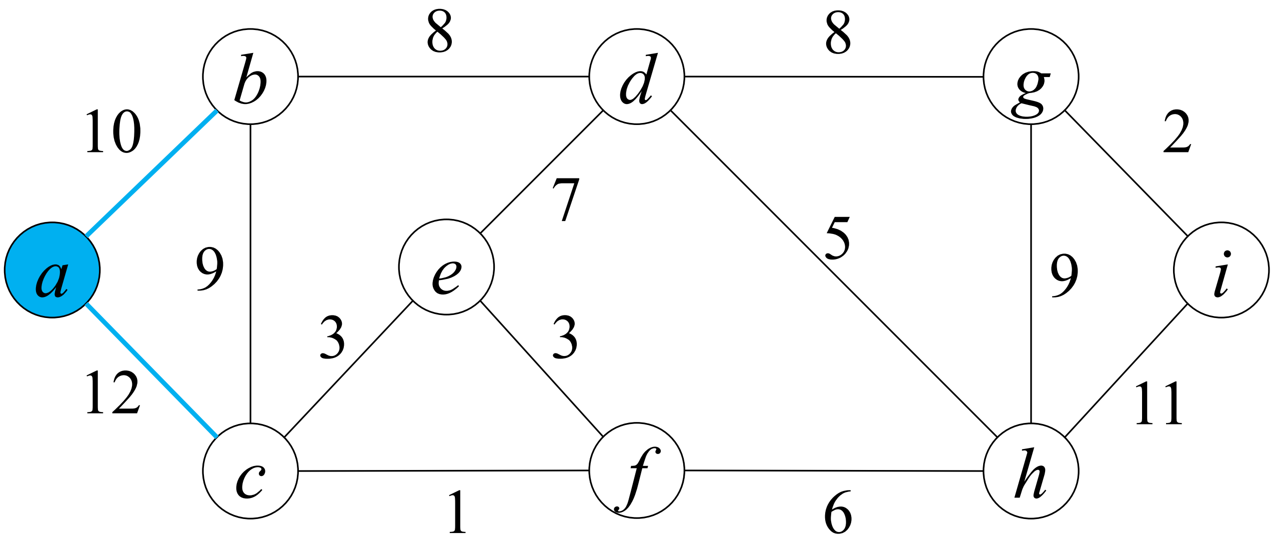
for each $u \in G.V$
    $u.key = \infty$
    $u.\pi = \text{NIL}$
$r.key = 0$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \text{EXTRACT-MIN}(Q)$
    for each $v \in G.Adj[u]$
        if $v \in Q$ and $w(u, v) < v.key$
            $v.\pi = u$
            $v.key = w(u, v)$

Algorithmus von Prim



Q: $a(0), b(\infty), c(\infty), d(\infty), e(\infty), f(\infty), g(\infty), h(\infty), i(\infty)$

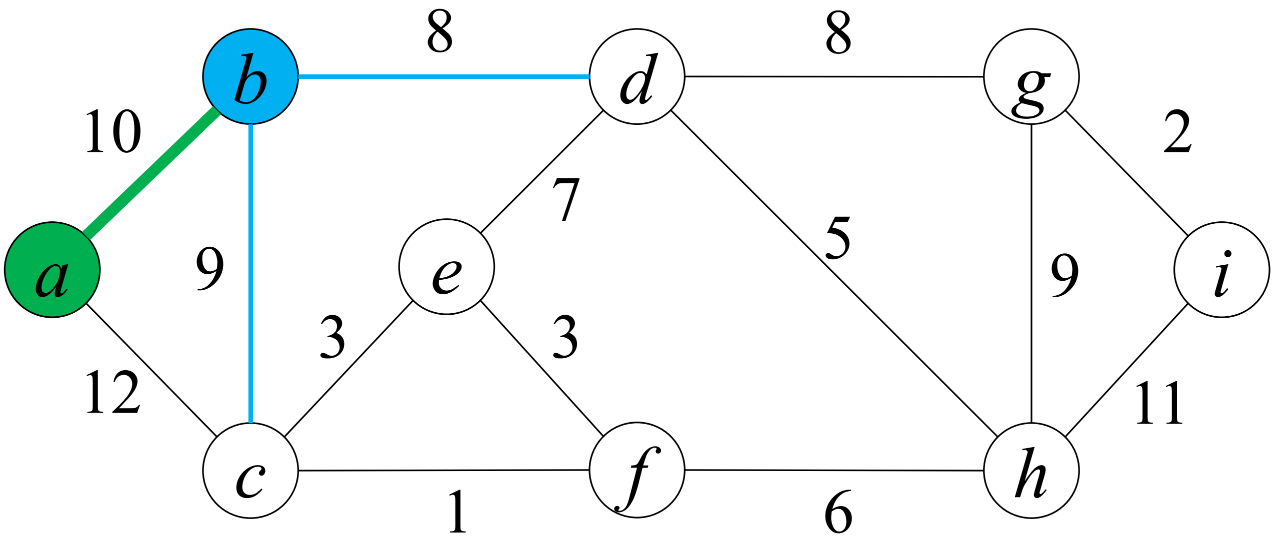
Algorithmus von Prim



Q: $a(0), b(\infty), c(\infty), d(\infty), e(\infty), f(\infty), g(\infty), h(\infty), i(\infty)$

Q': $b(10), c(12), d(\infty), e(\infty), f(\infty), g(\infty), h(\infty), i(\infty)$

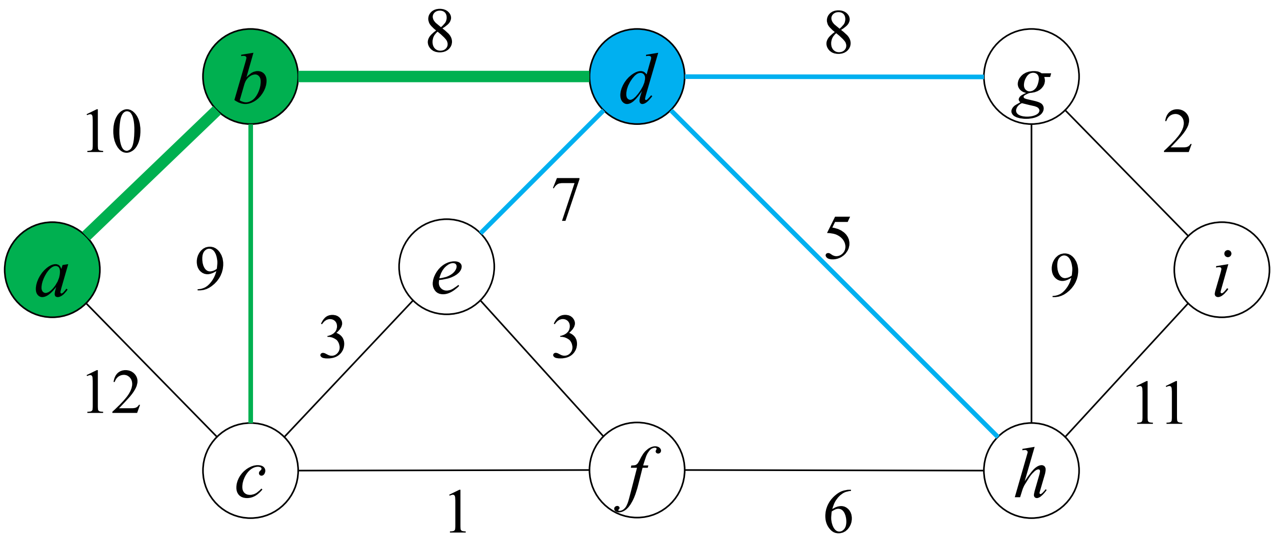
Algorithmus von Prim



Q: $b(10), c(12), d(\infty), e(\infty), f(\infty), g(\infty), h(\infty), i(\infty)$

Q': $d(8), c(9), e(\infty), f(\infty), g(\infty), h(\infty), i(\infty)$

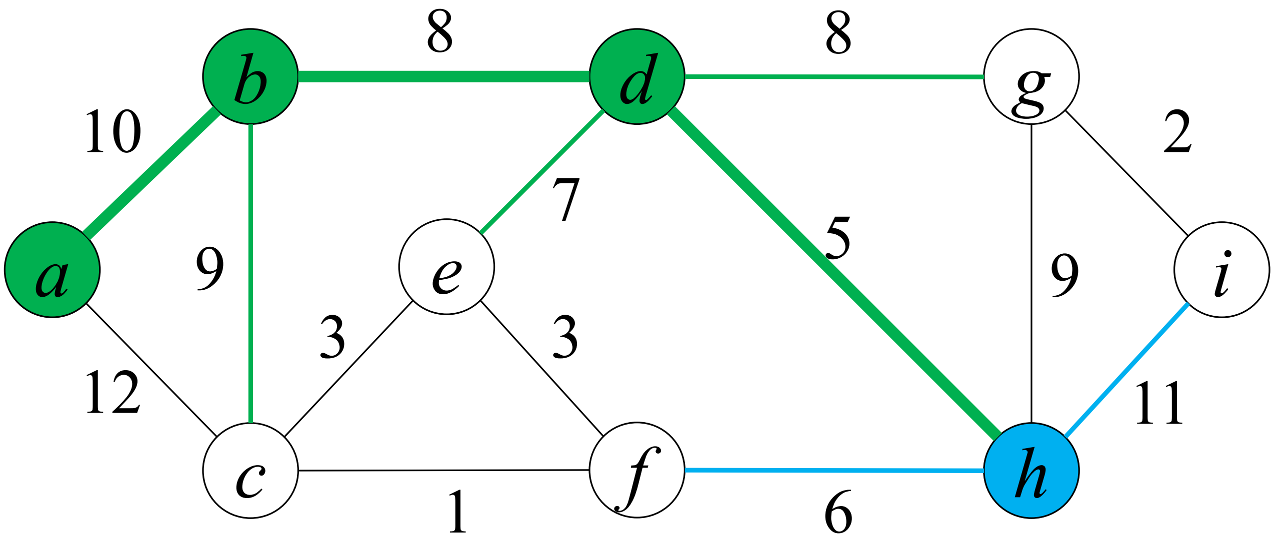
Algorithmus von Prim



Q: $d(8), c(9), e(\infty), f(\infty), g(\infty), h(\infty), i(\infty)$

Q': $h(5), e(7), g(8), c(9), f(\infty), i(\infty)$

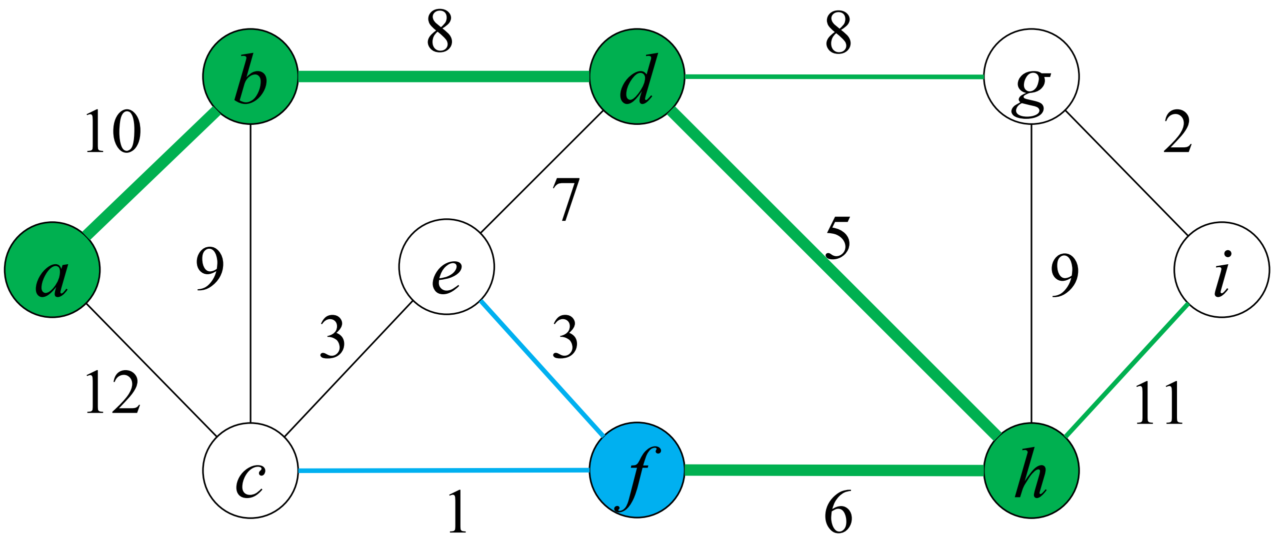
Algorithmus von Prim



Q: $h(5), e(7), g(8), c(9), f(\infty), i(\infty)$

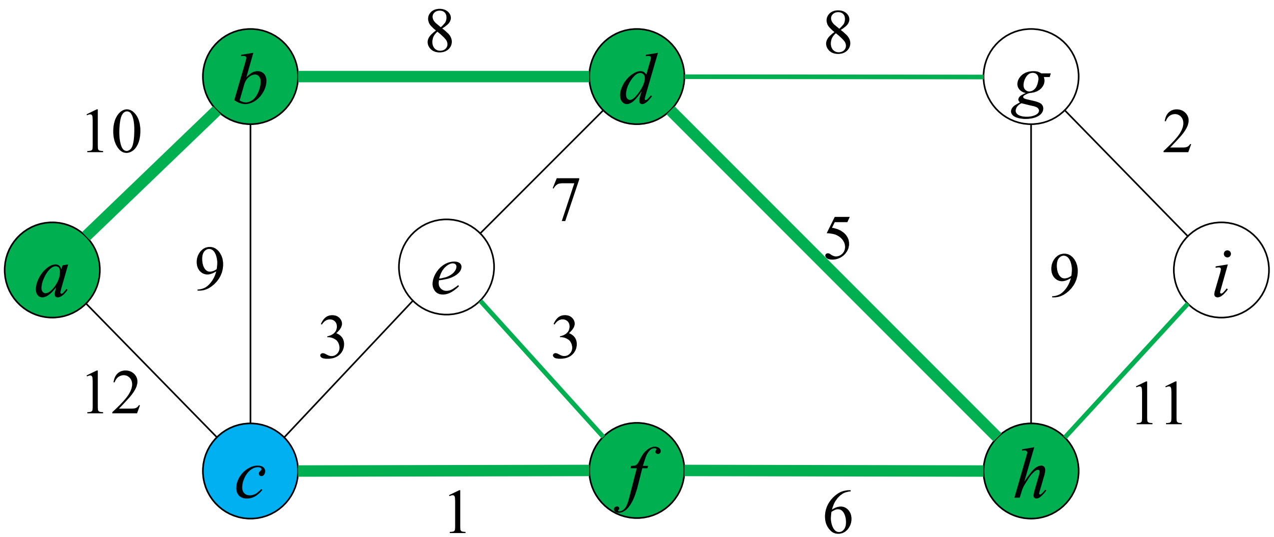
Q': $f(6), e(7), g(8), c(9), i(11)$

Algorithmus von Prim



Q: $f(6), e(7), g(8), c(9), i(11)$
Q': $c(1), e(3), g(8), i(11)$

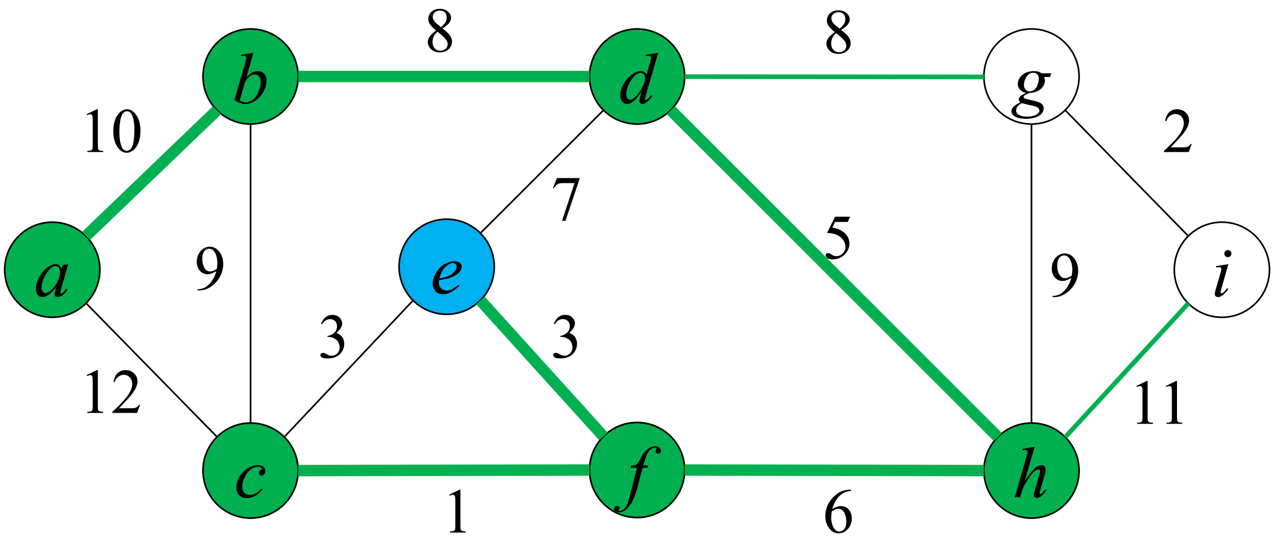
Algorithmus von Prim



Q: $c(1), e(3), g(8), i(11)$

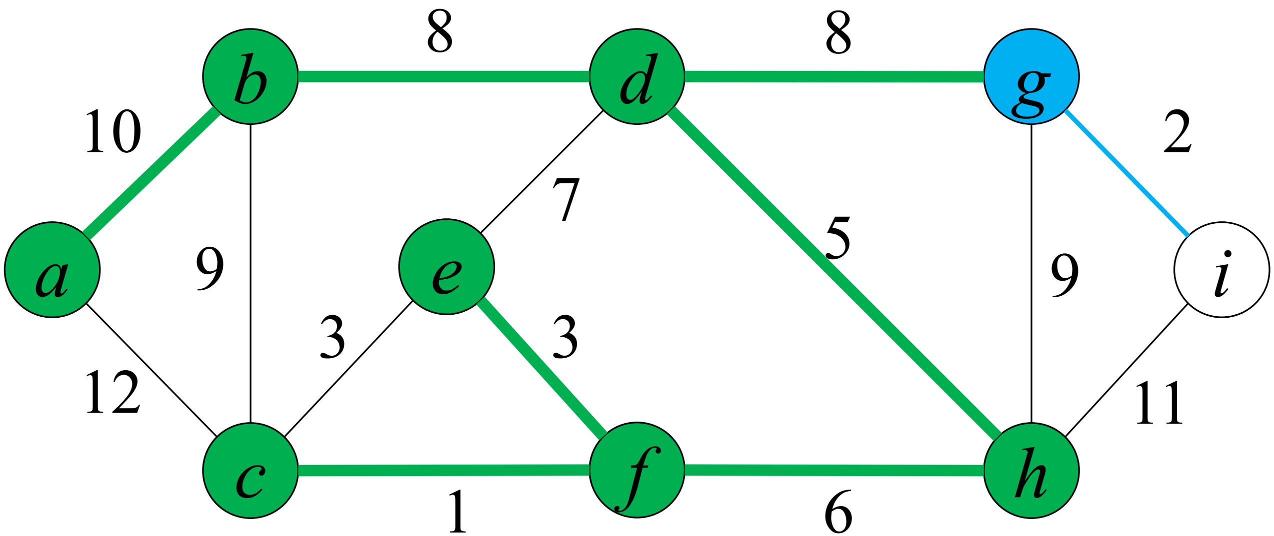
Q': $e(3), g(8), i(11)$

Algorithmus von Prim



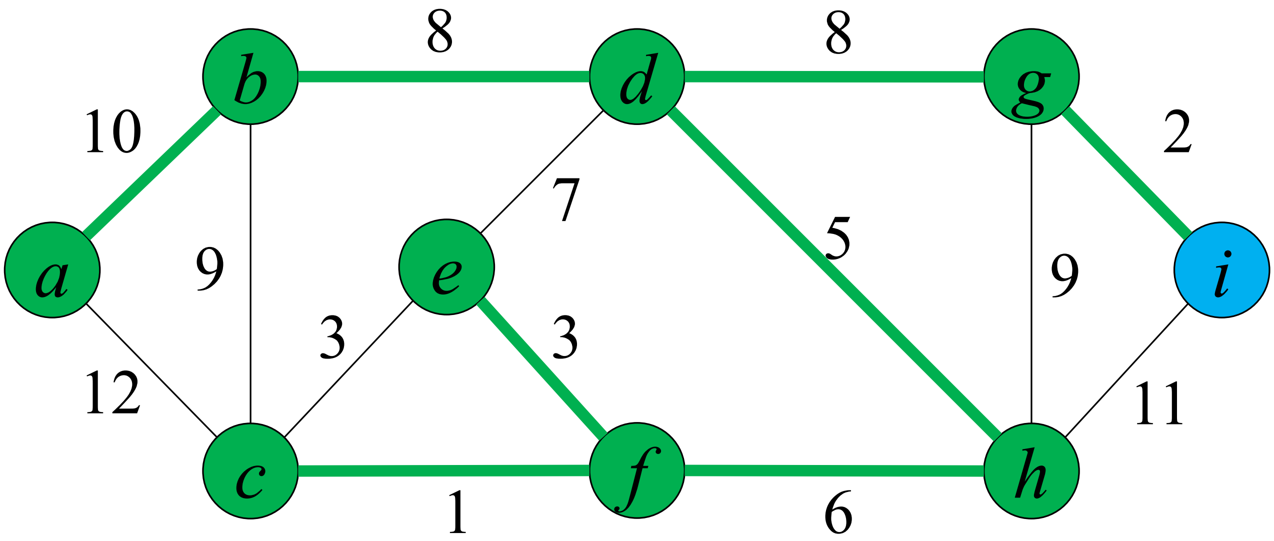
Q: $e(3), g(8), i(11)$
Q': $g(8), i(11)$

Algorithmus von Prim



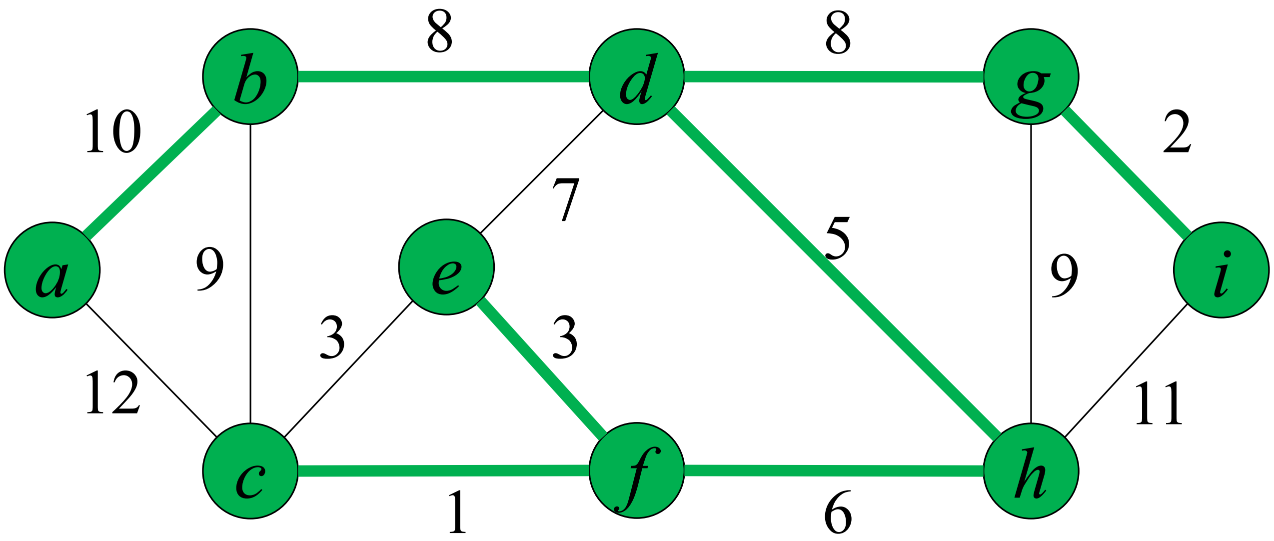
Q: $g(8), i(11)$
Q': $i(2)$

Algorithmus von Prim



Q: $i(2)$
Q':

Algorithmus von Prim



Analyse

- Mit Min-Heap als Prioritätswarteschlange
- Initialisierung der Gewichte: $\theta(V)$
- BUILD-MIN-HEAP: $O(V)$
- while-Schleife:
  - $|V|$ Aufrufe von EXTRACT-MIN: $O(V \lg V)$
  - $\leq |E|$ implizite Aufrufe von DECREASE-KEY: $O(E \lg V)$
- Total $O(E \lg V) + O(V \lg V) = O(E \lg V)$

Graphenalgorithmen

- Minimale Spannbäume
- Kürzeste Pfade

Kürzeste Pfade

- Intuitiv: Finde kürzesten Pfad zwischen zwei Punkten auf einer Karte
- Eingabe: gerichteter Graph mit Kantengewichten
  - Verallgemeinerung von Breitensuche auf gewichtete Bäume
- Gewicht eines Pfades $p = \langle v_0, v_1, \ldots, v_k \rangle$ ist Summe der Kantengewichte

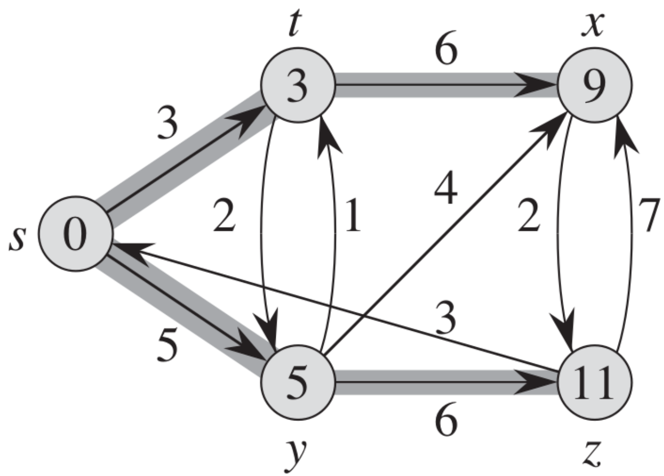
$$\sum_{i=1}^{k} w(v_{i-1}, v_i)$$

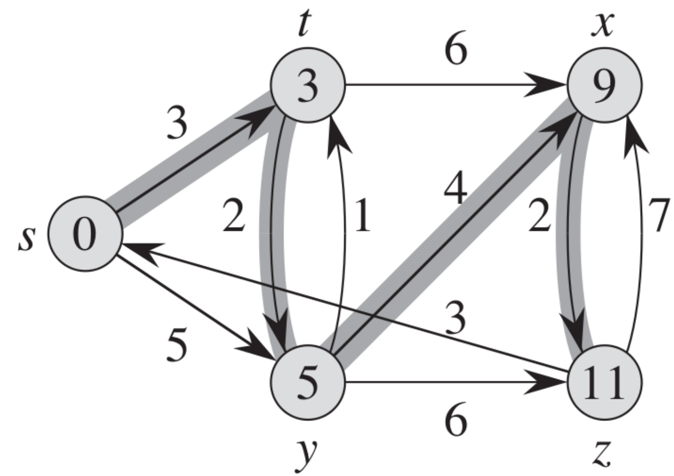
- Gewicht des kürzesten Pfades $\delta(u, v)$

$$\delta(u,v) = \begin{cases} \min\{w(p) : u \overset{p}{\leadsto} v\} & \text{if there is a path from } u \text{ to } v \text{ ,} \\ \infty & \text{otherwise .} \end{cases}$$

Kürzeste Pfade

- Beispiel: kürzester Pfad von s aus





- Nicht eindeutig
- Kürzeste Pfade von einem Startknoten bilden Baum
- Gewichte können für beliebige Grössen stehen
  - Zeit, Kosten, Verlust, etc.

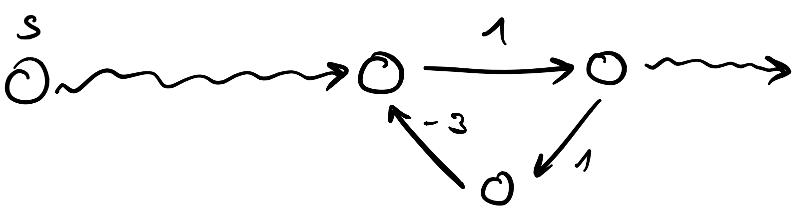
Kürzeste Pfade

Varianten

- Einziger Startknoten (single-source)
  - Alle kürzesten Pfade von einem Startknoten zu allen anderen Knoten
- Einziger Zielknoten (single-destination)
  - Alle kürzesten Pfade von allen Startknoten zu einem Zielknoten
  - Drehe Richtung aller Kanten → single-source Problem
- Festes Knotenpaar (single-pair)
  - Kürzester Pfad zwischen zwei Knoten
  - Kein asymptotisch schnellerer Algorithmus bekannt als für single-source
- Alle Paare (all-pairs)
  - Naiv: löse single-source für alle Startknoten
  - Geht besser, Kapitel 25 im Buch

Negative Kantengewichte

- Ok, falls keine Zyklen mit negativem Gewicht, erreichbar von Startknoten
  - Könnten sonst unendlich lang im Kreis gehen um Gewicht zu reduzieren
  - Kein Problem falls Zyklus vom Startknoten nicht erreichbar



Zyklen

- Annahme: finde kürzeste Pfade ohne Zyklen
- Zyklen mit negativem Gewicht
  - Nicht erlaubt in Eingabe
- Zyklen mit positivem Gewicht
  - Vermeidung von Zyklen führt zu kürzerem Pfad
  - Kommen nicht vor in Lösung
- Zyklen mit Gewicht 0
  - Führt zu nicht-eindeutiger Lösung
  - Annahme: finde Lösung ohne Zyklen mit Gewicht 0

Optimale Teilstruktur

Lemma: Jeder Teilpfad eines kürzesten Pfades ist ein kürzester Pfad

Beweis: Durch Widerspruch

- Sei p kürzester Pfad von u nach v (1)

$$u \overset{p_{ux}}{\rightsquigarrow} x \overset{p_{xy}}{\rightsquigarrow} y \overset{p_{yv}}{\rightsquigarrow} v$$

- Länge $\delta(u, v) = w(p) = w(p_{ux}) + w\left(p_{xy}\right) + w(p_{yv})$
- Annahme: existiere kürzerer Pfad $x \overset{p'_{xy}}{\rightsquigarrow} y$
- Also $w\left(p'_{xy}\right) < w(p_{xy})$
- Neuer Pfad p' $u \overset{p_{ux}}{\rightsquigarrow} x \overset{p'_{xy}}{\rightsquigarrow} y \overset{p_{yv}}{\rightsquigarrow} v$
- Also $w(p') = w(p_{ux}) + w(p'_{xy}) + w(p_{yv})$

$$< w(p_{ux}) + w(p_{xy}) + w(p_{yv}) = w(p).$$

Widerspruch zu (1)!

Kürzeste Pfade, ein Startknoten

Eingabe: Graph, Startknoten s

Ausgabe: für jeden Knoten

- Länge des kürzesten Pfades: $\delta(s, v)$
  - $v.d$ ist Schätzung des kürzesten Pfades
  - Nach Initialisierung $v.d = \infty$
  - Während Algorithmus reduziert bis $v.d = \delta(s, v)$
- Vorgängerknoten auf kürzestem Pfad: $v.\pi$
  - Nach Initialisierung $v.\pi = \text{NIL}$
  - Aus $v.\pi$ wird Vorgängerteilgraph abgeleitet, bildet Baum kürzester Pfade

Verwendete Prozeduren

- Initialisierung

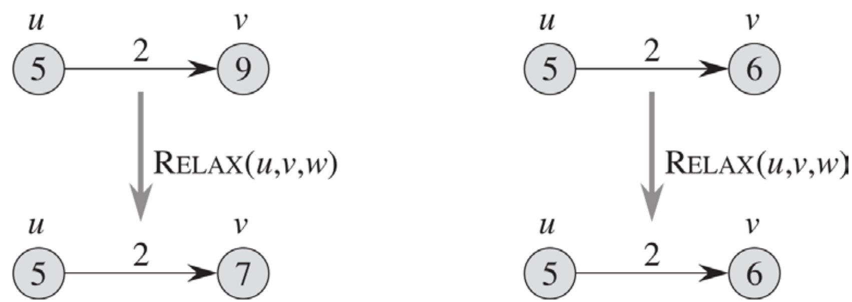
INITIALIZE-SINGLE-SOURCE(G, s)

```
for each vertex v ∈ G.V
    v.d = ∞
    v.π = NIL
s.d = 0
```

- Relaxation

RELAX(u, v, w)

```
if v.d > u.d + w(u,v)
    v.d = u.d + w(u,v)
    v.π = u
```



- Unterschiede konkreter Algorithmen
  - Reihenfolge der Relaxationen
  - Anzahl Relaxationen auf jeder Kante

Eigenschaften kürzester Pfade

- Dreiecksungleichung
  - Für alle Kanten (u, v) gilt
  $$\delta(s, v) \leq \delta(s, u) + w(u, v)$$
- Eigenschaft der oberen Schranke
  - Für alle Knoten ist stets $v.d \geq \delta(s, v)$
  - Sobald $v.d = \delta(s, v)$ bleibt $v.d$ konstant
- Kein-Pfad-Eigenschaft
  - Wenn es keinen Pfad von s nach v gibt, d.h. $\delta(s, v) = \infty$, dann ist stets $v.d = \infty$

Eigenschaften kürzester Pfade

- Konvergenzeigenschaft
  - Falls kürzester Pfad $s \rightsquigarrow u \rightarrow v$ und $u.d = \delta(s,u)$, dann ist nach $\text{Relax}(u,v,w)$ $v.d = \delta(s,v)$
- Pfadrelaxationseigenschaft
  - Wenn $p = \langle v_0, v_1, \dots, v_k \rangle$ kürzester Pfad von $s = v_0$ nach v_k und Kanten in Reihenfolge $(v_0, v_1), (v_1, v_2), \dots (v_{k-1}, v_k)$ relaxiert werden, dann gilt $v_k.d = \delta(s, v_k)$
  - Ungeachtet anderer Relaxationsschritte

Bellman-Ford-Algorithmus

- Negative Kantengewichte erlaubt
- Gibt FALSE zurück bei negativen Zyklen

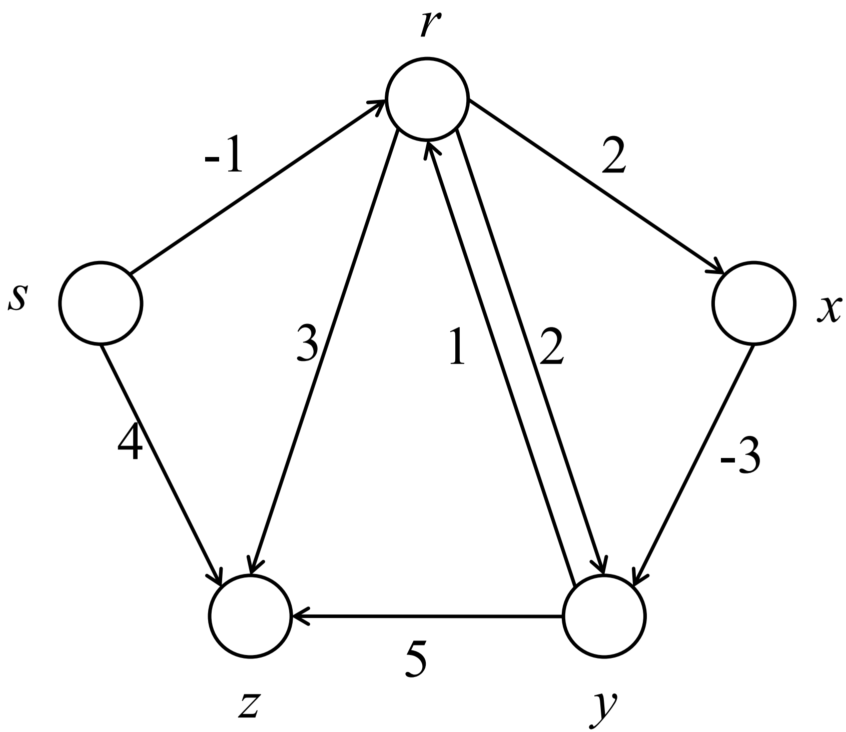
```
BELLMAN-FORD(G, w, s)
INITIALIZE-SINGLE-SOURCE(G, s)
for i = 1 to |G.V| − 1
    for each edge (u, v) ∈ G.E
        RELAX(u, v, w)
// check for negative weight cycle
for each edge (u, v) ∈ G.E
    if v.d > u.d + w(u, v)
        return FALSE
return TRUE
```

```
RELAX(u, v, w)
if v.d > u.d + w(u, v)
    v.d = u.d + w(u, v)
    v.π = u
```

- Laufzeit $\Theta(V\,E)$

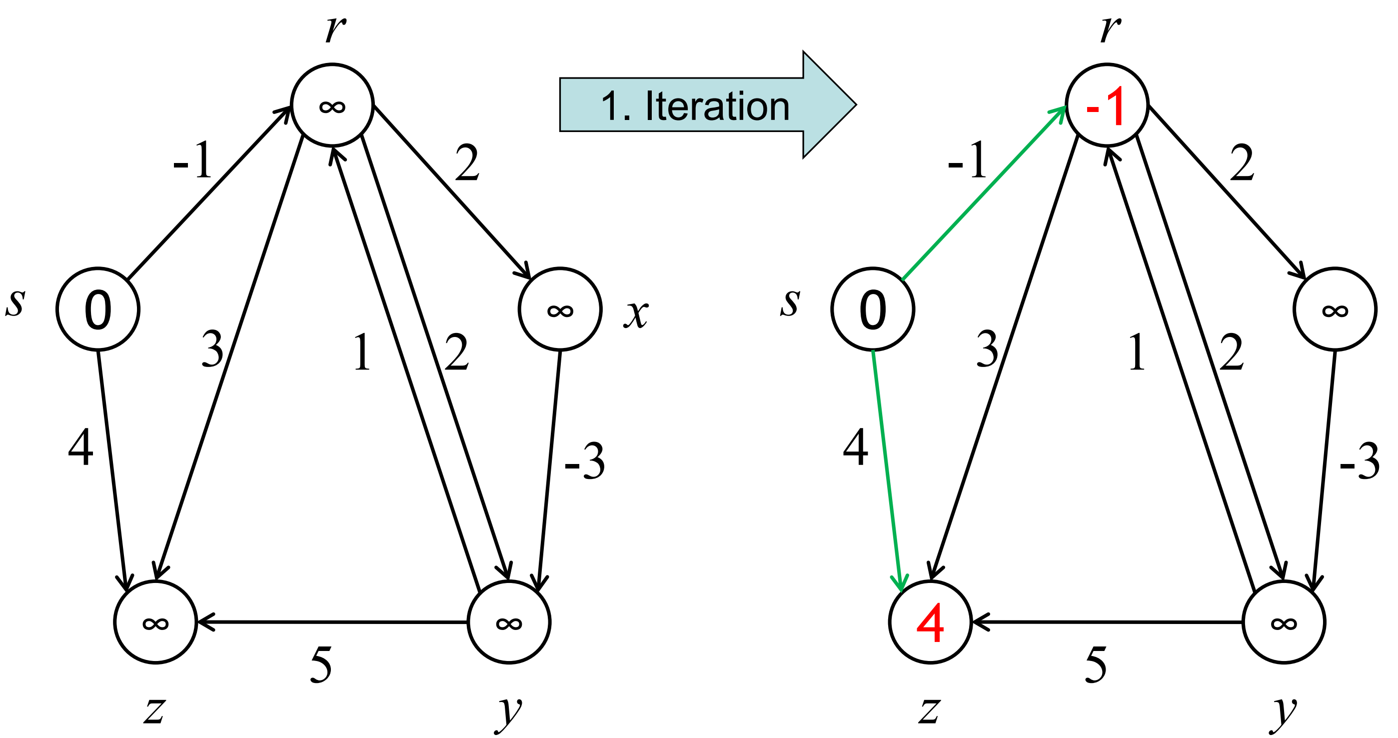
Bellman-Ford-Algorithmus

- Beispiel



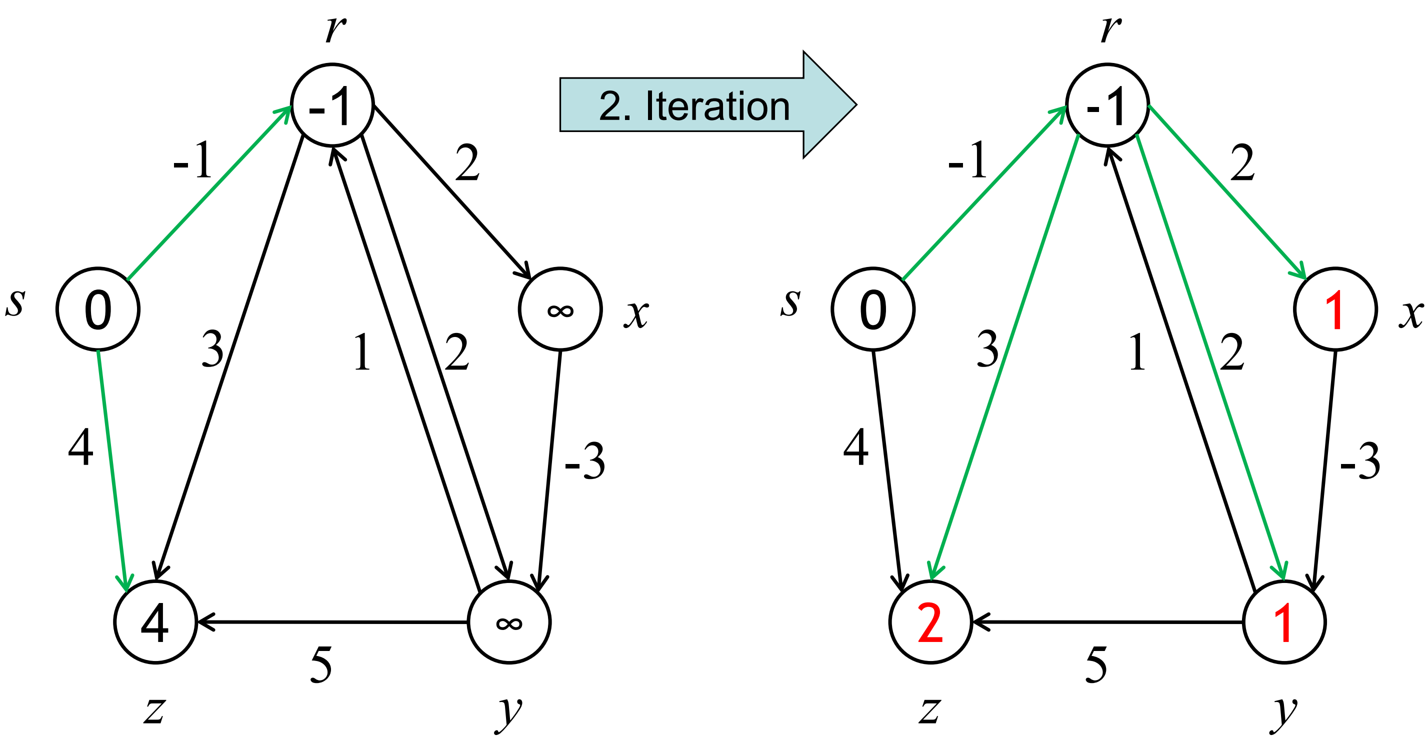
Bellman-Ford-Algorithmus

Reihenfolge: $(x,y),(y,z),(y,r),(r,x),(r,y),(r,z),(s,z),(s,r)$



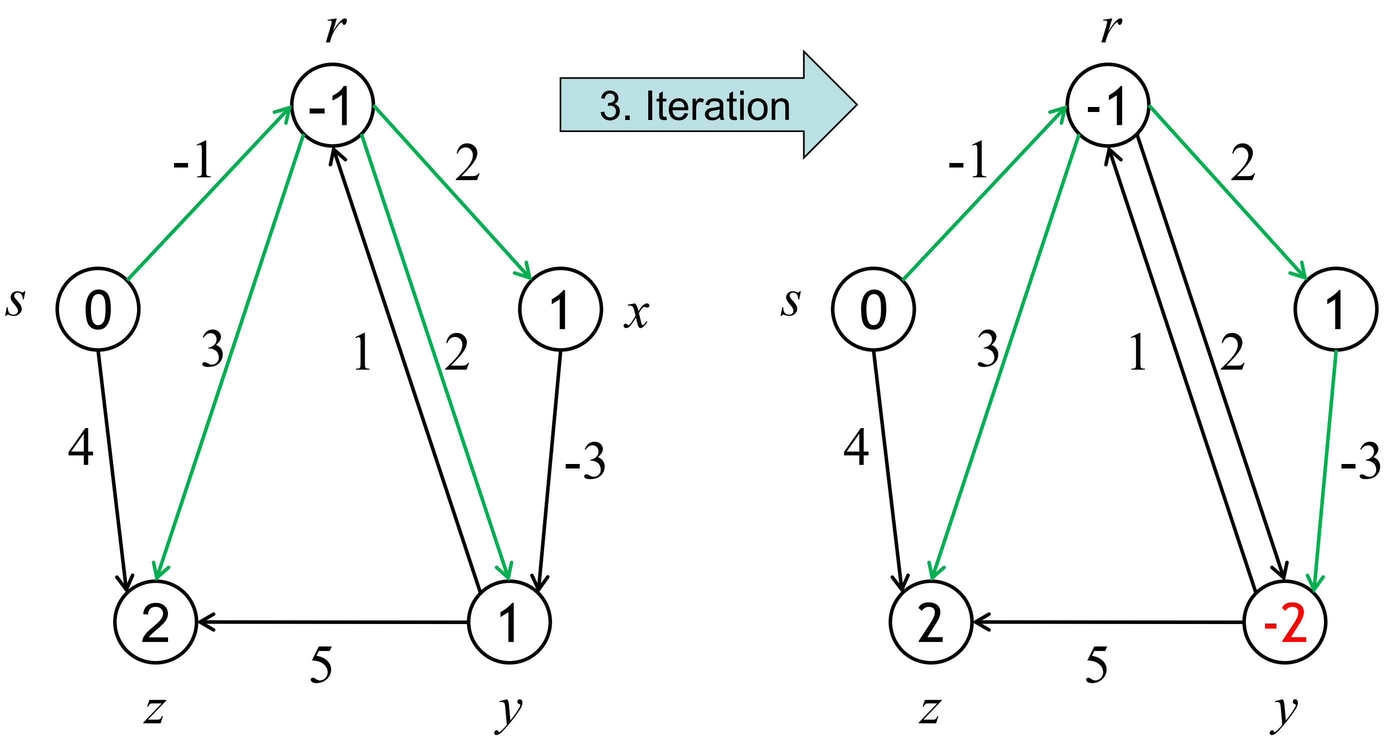
Bellman-Ford-Algorithmus

Reihenfolge: $(x,y),(y,z),(y,r),(r,x),(r,y),(r,z),(s,z),(s,r)$



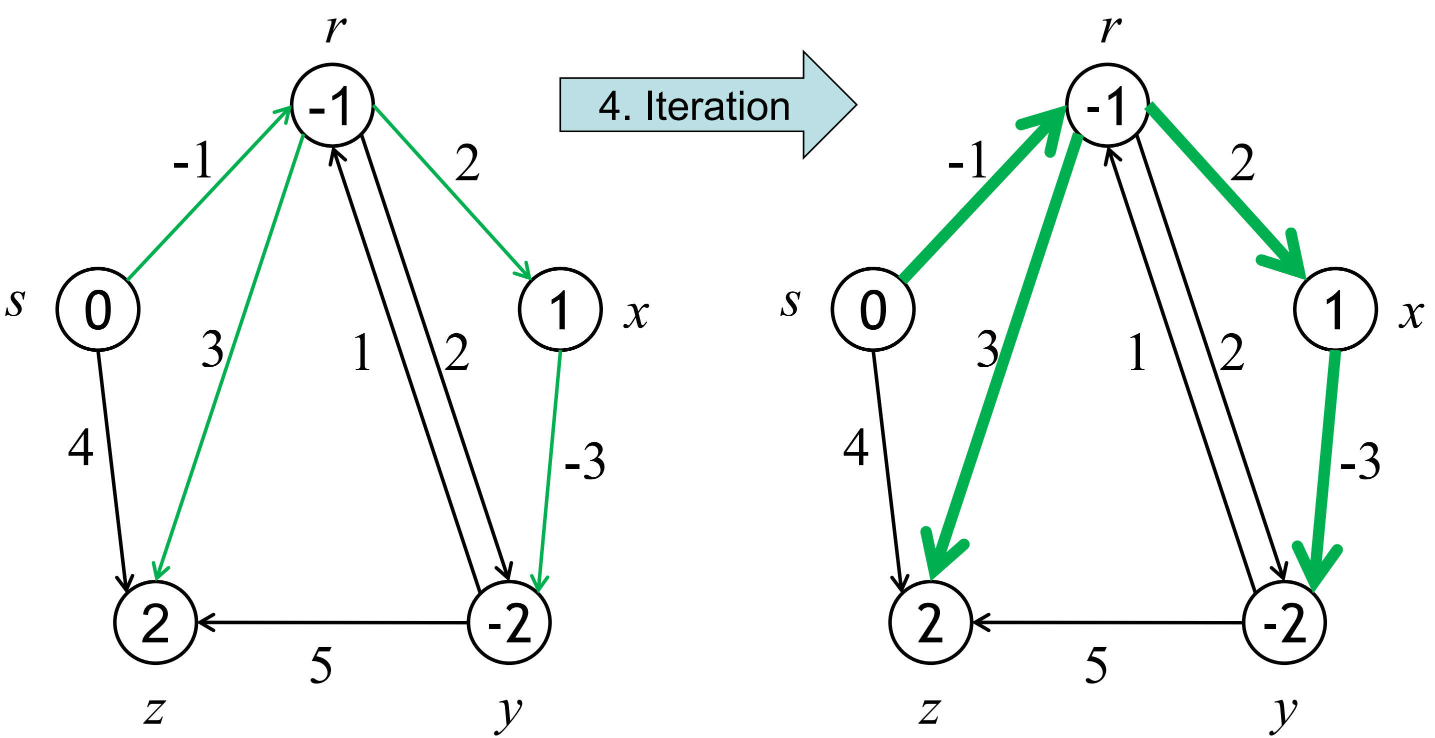
Bellman-Ford-Algorithmus

Reihenfolge: $(x,y),(y,z),(y,r),(r,x),(r,y),(r,z),(s,z),(s,r)$



Bellman-Ford-Algorithmus

Reihenfolge: $(x,y),(y,z),(y,r),(r,x),(r,y),(r,z),(s,z),(s,r)$



Beweis (1. Teil des Algorithmus)

Mit Pfadrelaxationseigenschaft

Sei v erreichbar von s und sei $p = \langle v_0, v_1, \dots, v_k \rangle$ ein kürzester Pfad von s nach v, d.h. $v_0 = s$, $v_k = v$

- Da p azyklisch ist, hat $p \leq |V| - 1$ Kanten, also $k \leq |V| - 1$
- Jede Iteration der for-Schleife relaxiert alle Kanten
  - Erste Iteration relaxiert (v_0, v_1)
  - Zweite Iteration relaxiert (v_1, v_2)
  - etc.
- Wegen Pfadrelaxationseigenschaft gilt
  - nach Iteration i: $v_i.d = \delta(s, v_i)$,
  - also nach Iteration k: $v.d = v_k.d = \delta(s, v_k) = \delta(s, v)$

Beweis (Rückgabwert)

a) Es gibt keinen von s erreichbaren Zyklus mit negativem Gewicht

Dann gilt bei der Terminierung für alle $(u, v) \in G.E$

$$\begin{aligned} v.d &= \delta(s, v) \\ &\leq \delta(s, u) + w(u, v) \quad \text{(Dreiecksungleichung)} \\ &= u.d + w(u, v) \end{aligned}$$

Also gibt der Algorithmus TRUE zurück.

Beweis (Rückgabwert)

b) Es gibt einen von s erreichbaren Zyklus mit negativem Gewicht $c = \langle v_0, \dots, v_k \rangle$ mit $v_0 = v_k$, also $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$

- Annahme: Algorithmus gibt TRUE zurück
- Dann gilt $v_i.d \le v_{i-1}.d + w(v_{i-1}, v_i)$ für $i = 1, \dots, k$
- Betrachte Summe über Zyklus c

$$\sum_{i=1}^{k} v_i.d \le \sum_{i=1}^{k} \left(v_{i-1}.d + w(v_{i-1}, v_i)\right) = \sum_{i=1}^{k} v_{i-1}.d + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Da $v_0 = v_k$ gilt

$$\sum_{i=1}^{k} v_i.d = \sum_{i=1}^{k} v_{i-1}.d, \qquad \text{also } 0 \le \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

- Widerspruch zu $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$

Kürzeste Pfade in DAGs

- Kanten mit negativem Gewicht erlaubt
- Keine (negativen) Zyklen, da DAG

DAG-SHORTEST-PATH(G, w, s)

topologically sort the vertices of G
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex u taken in topologically sorted order
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w)

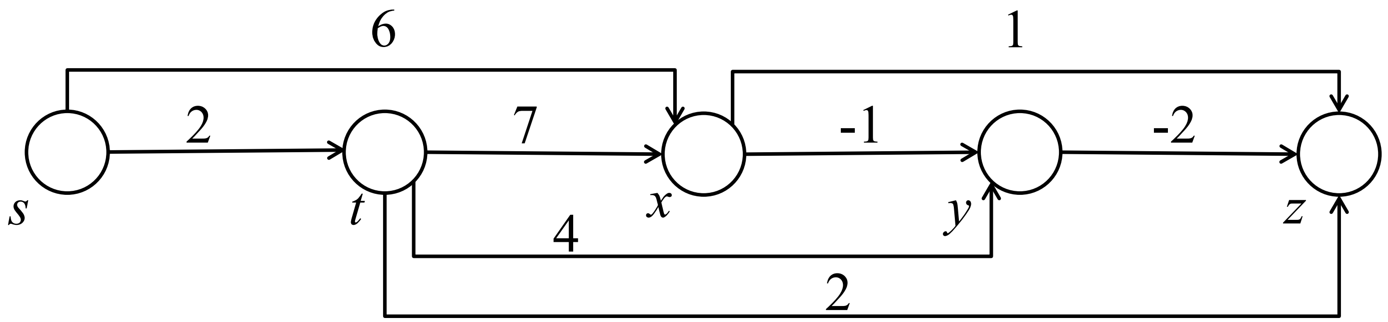
INITIALIZE-SINGLE-SOURCE(G, s)

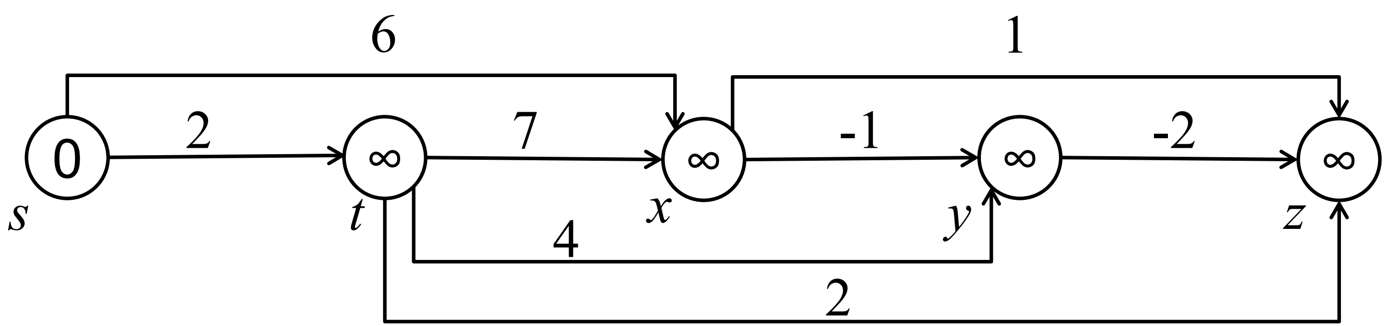
for each vertex $v \in G.V$
    $v.d = \infty$
    $v.\pi = \text{NIL}$
$s.d = 0$

- Laufzeit $\Theta(V + E)$

Kürzeste Pfade in DAGs

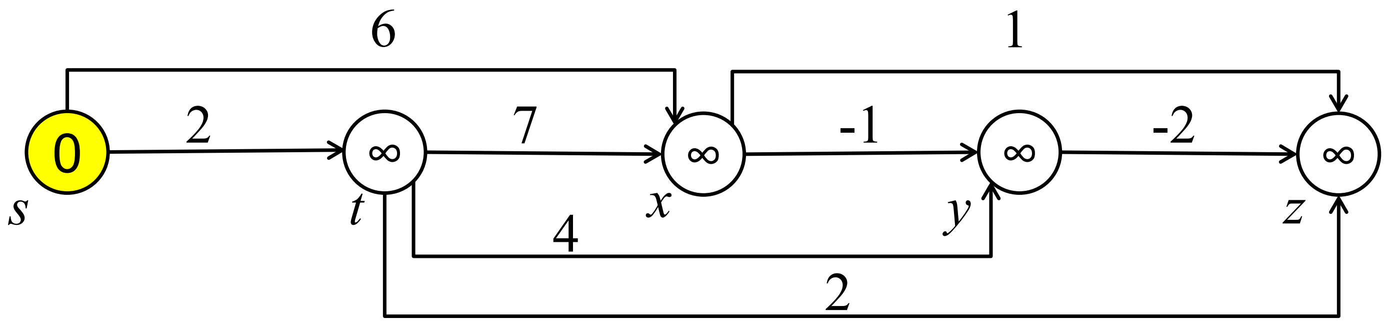
- Beispiel

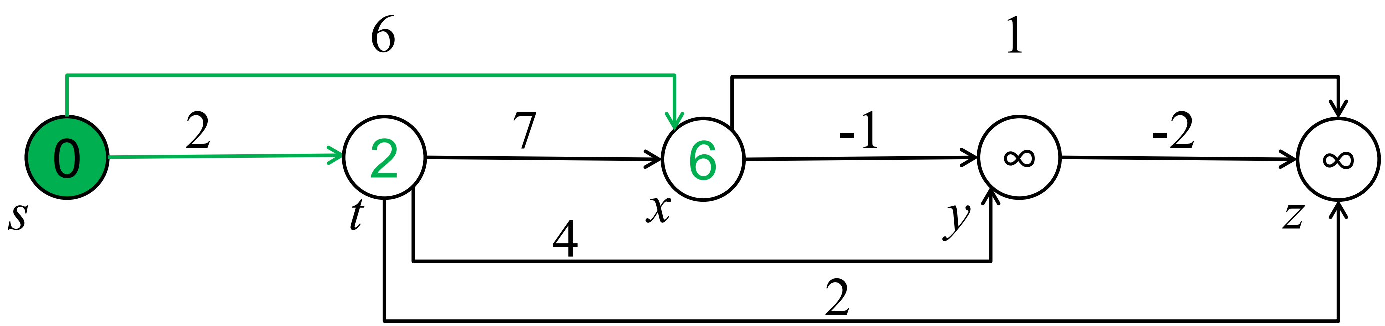




Kürzeste Pfade in DAGs

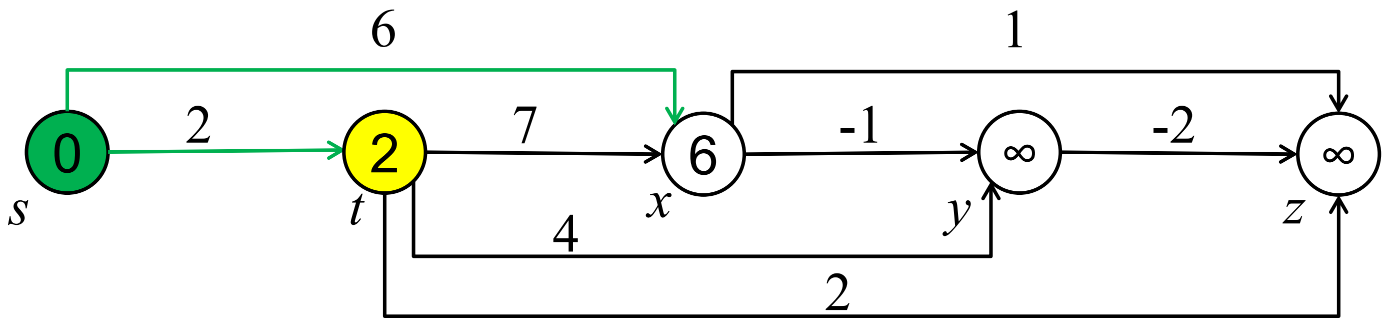
$u = s$, $\textsc{Relax}(s, t, w)$, $\textsc{Relax}(s, x, w)$

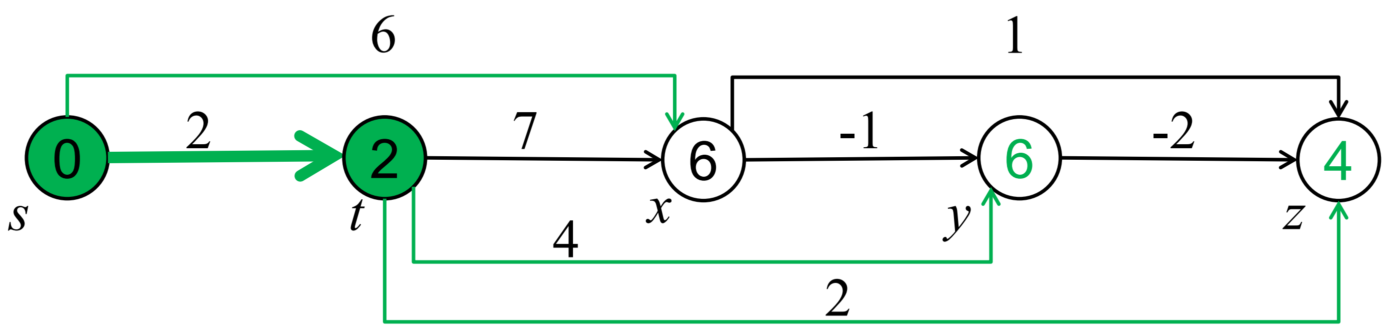




Kürzeste Pfade in DAGs

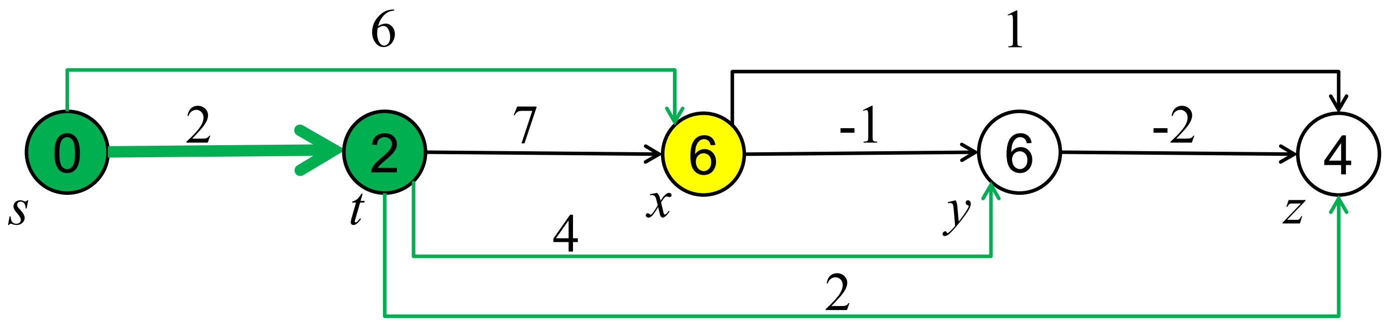
$u = t$, $\textsc{Relax}(t, x, w)$, $\textsc{Relax}(t, y, w)$, $\textsc{Relax}(t, z, w)$

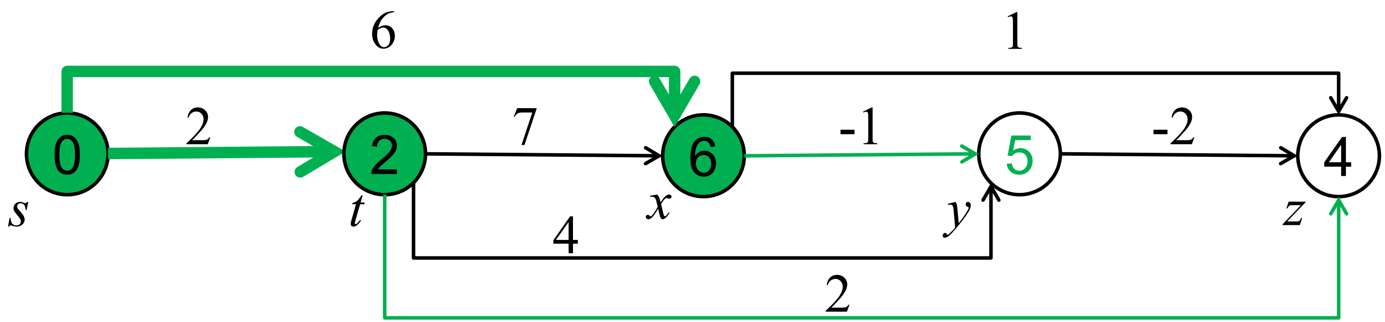




Kürzeste Pfade in DAGs

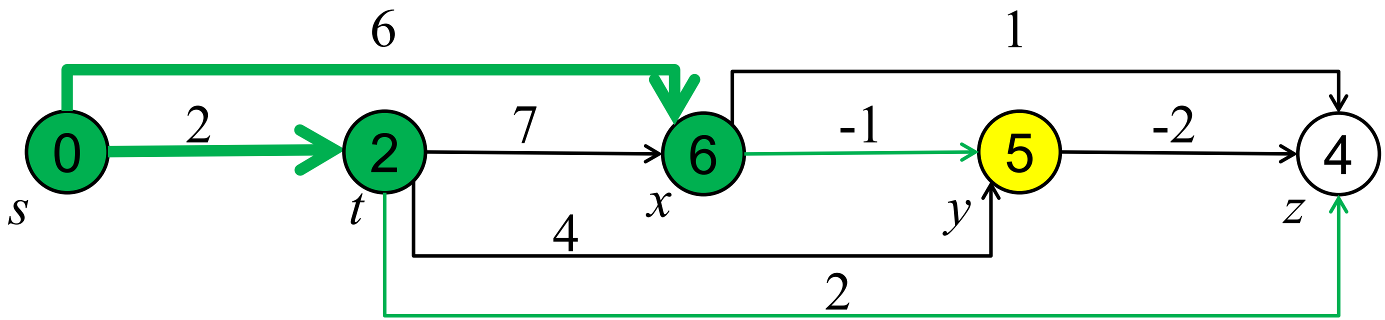
$u = x$, $\textsc{Relax}(x, y, w)$, $\textsc{Relax}(x, z, w)$

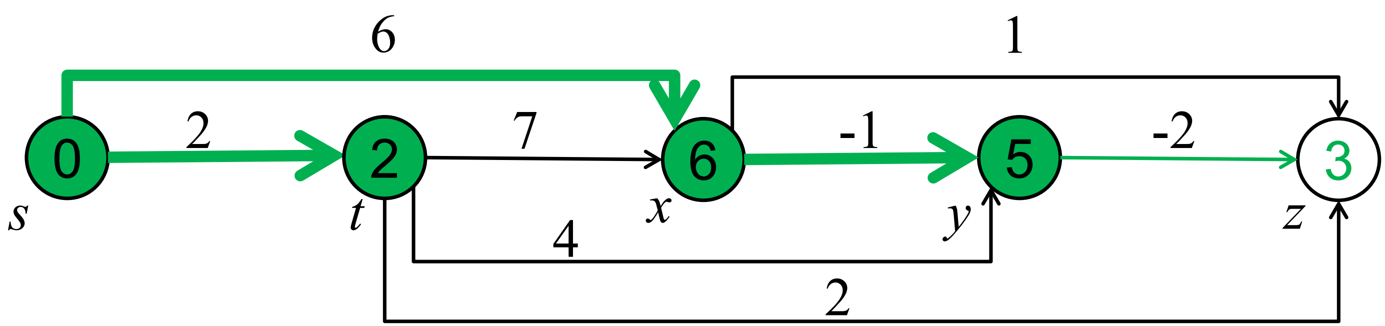




Kürzeste Pfade in DAGs

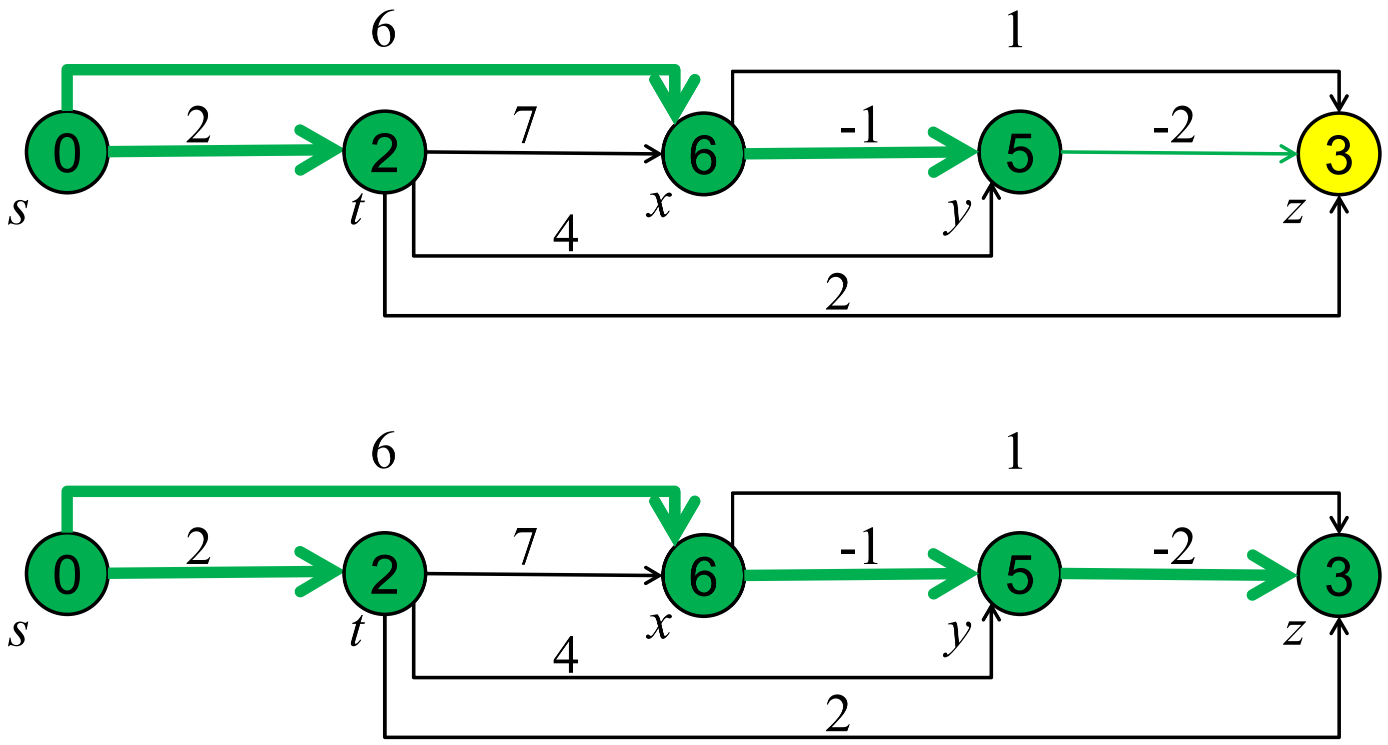
$u = x$, $\textsc{Relax}(y, z, w)$





Kürzeste Pfade in DAGs

$u = z$



Dijkstra-Algorithmus

- Keine negativen Kantengewichte erlaubt in Eingabe
- Modifikation der Breitensuche, sehr ähnlich wie Algorithmus von Prim
  - Prioritätswarteschlange statt FIFO
  - Schlüssel sind kürzeste Pfad Schätzungen
- Knoten aufgeteilt in zwei Teilmengen
  - S: endgültiger kürzester Pfad bestimmt
  - Q: Prioritätswarteschlange, $Q = V - S$

Dijkstra-Algorithmus

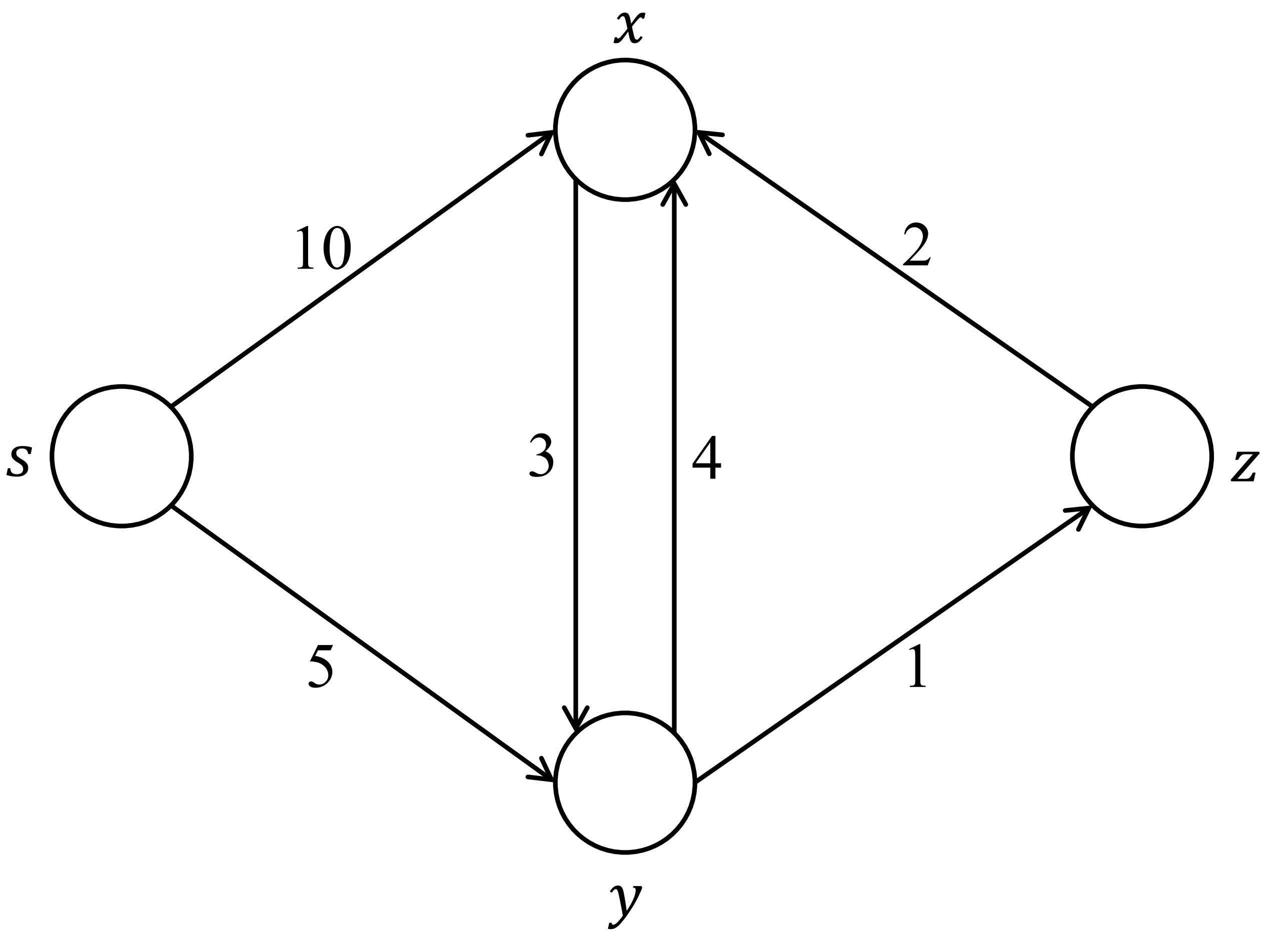
$\textsc{Dijkstra}(G, w, s)$

$\textsc{Initialize-Single-Source}(G, s)$
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    $u = \textsc{Extract-Min}(Q)$
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        $\textsc{Relax}(u, v, w)$

- Ähnlich wie Algorithmus von Prim, aber Schlüssel sind Schätzung des kürzesten Pfades
- Greedy Algorithmus
- Analyse: mit Min-Heap $O(E \lg V)$

Dijkstra-Algorithmus

- Beispiel

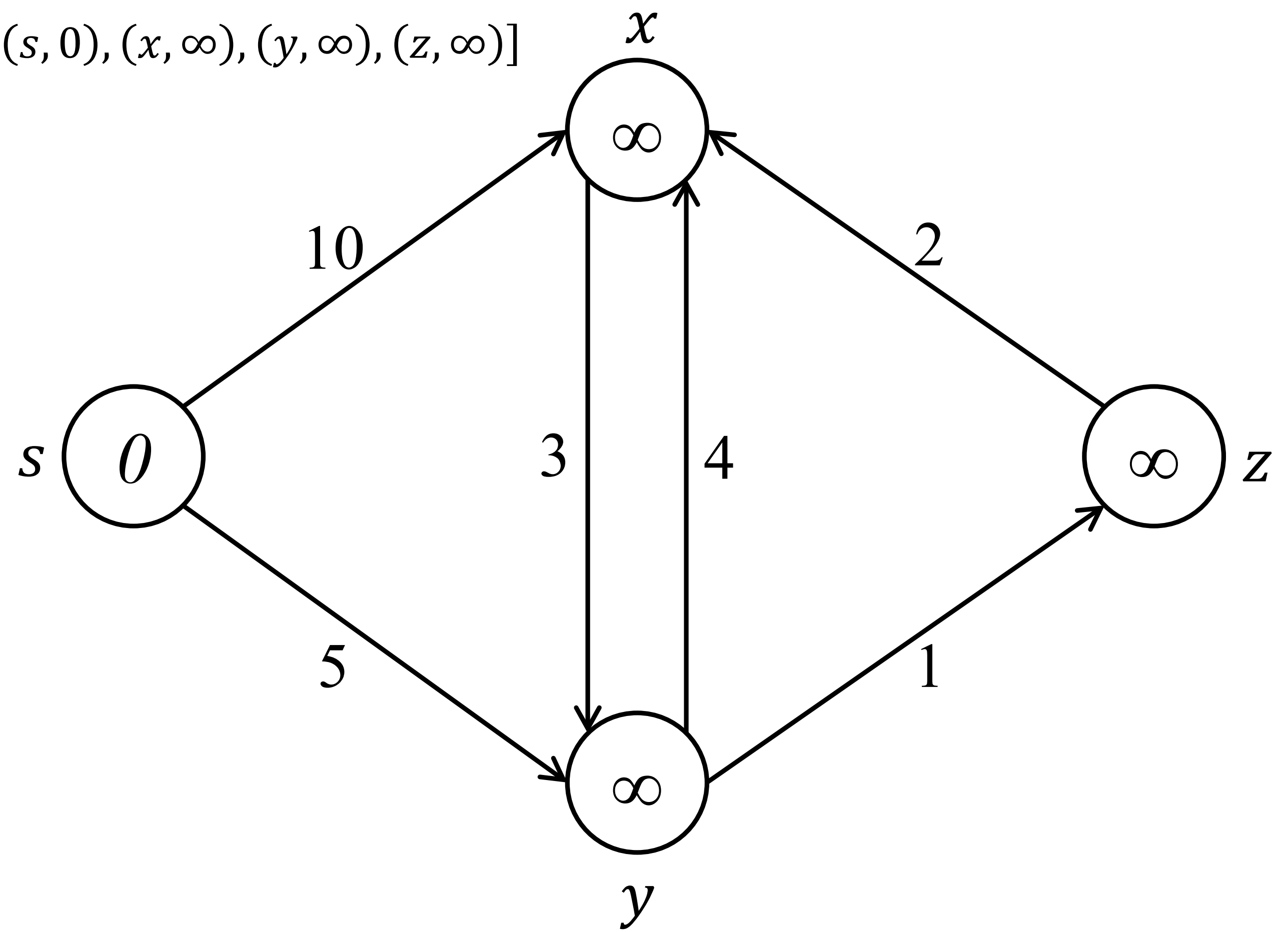


Dijkstra-Algorithmus

$\text{INITIALIZE-SINGLE-SOURCE}(G, s)$

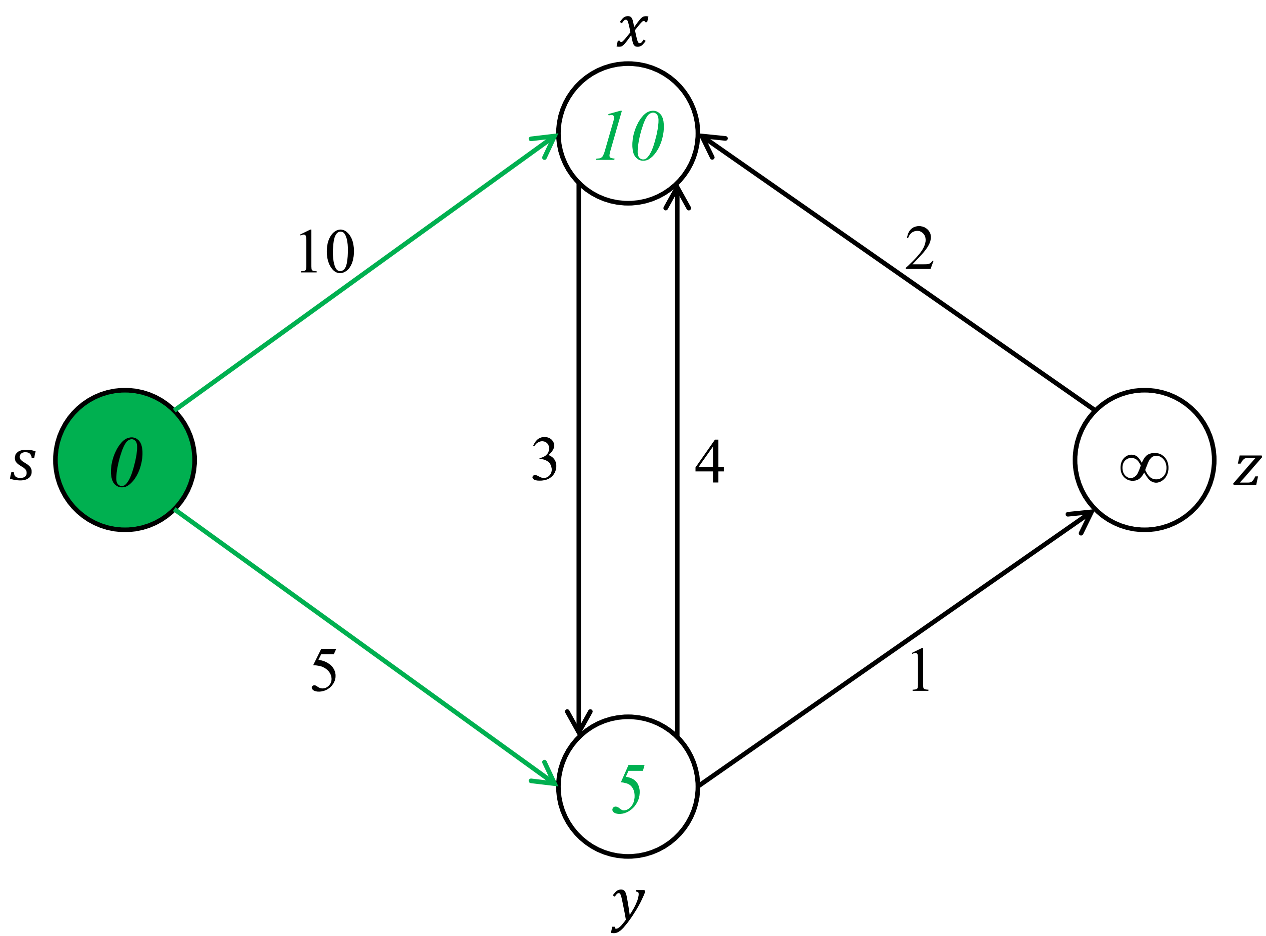
$S = \emptyset$

$Q = [(s, 0), (x, \infty), (y, \infty), (z, \infty)]$



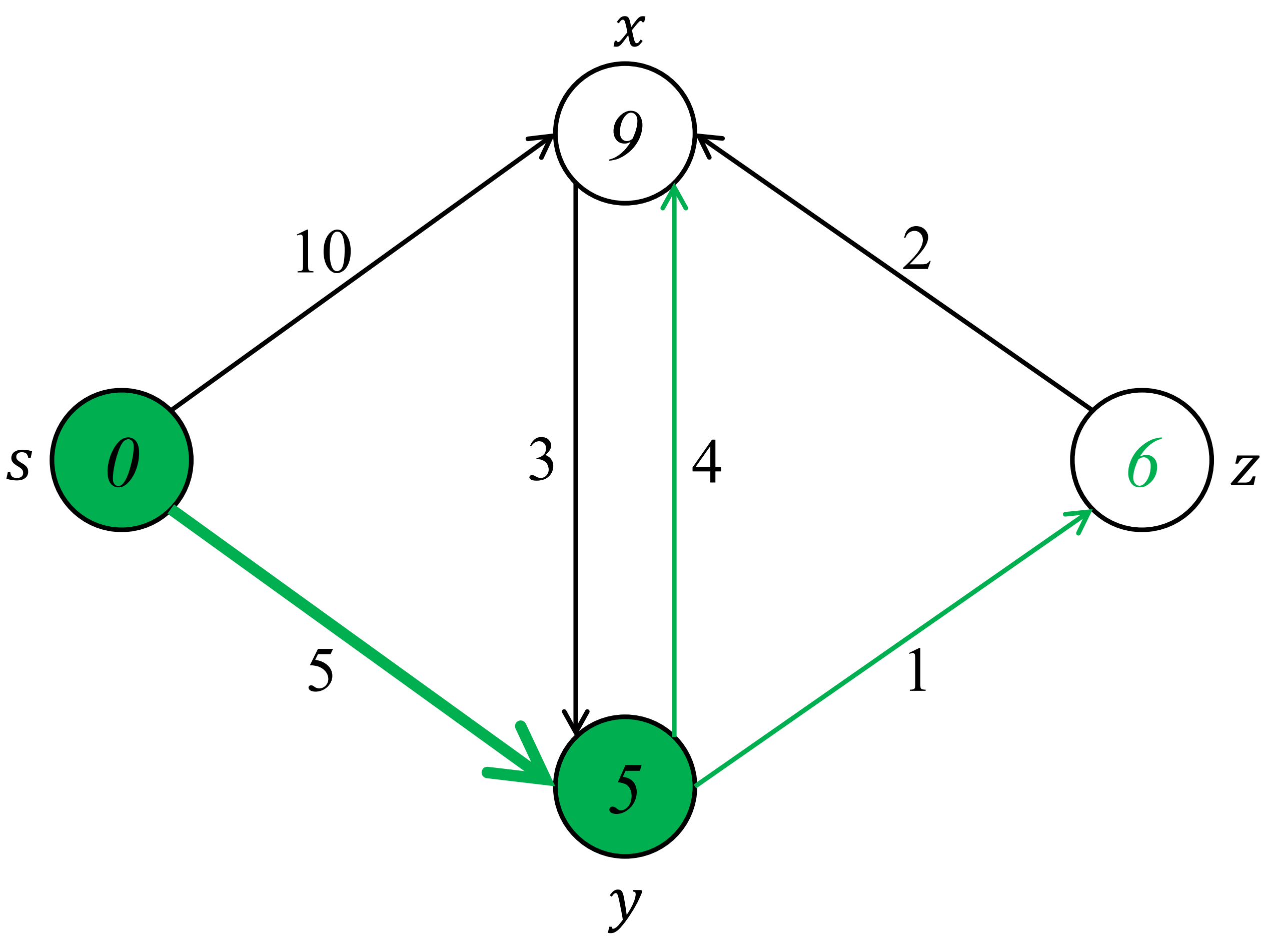
Dijkstra-Algorithmus

$S = \{s\}$, relaxiere $(s, x), (s, y)$ → $Q = [(y, 5), (x, 10), (z, \infty)]$



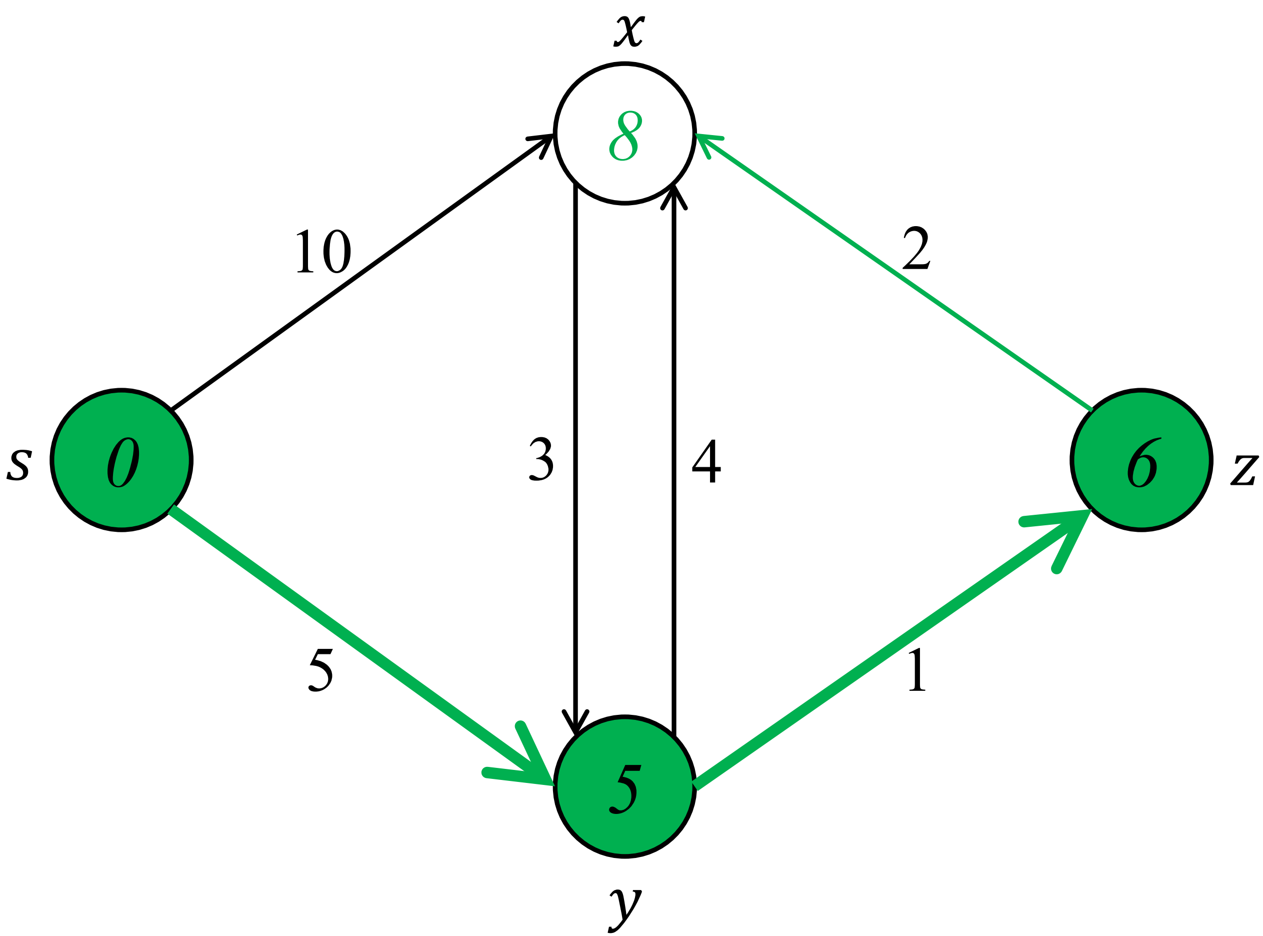
Dijkstra-Algorithmus

$S = \{s, y\}$, relaxiere $(y, x), (y, z)$ → $Q = [(z, 6), (x, 9)]$



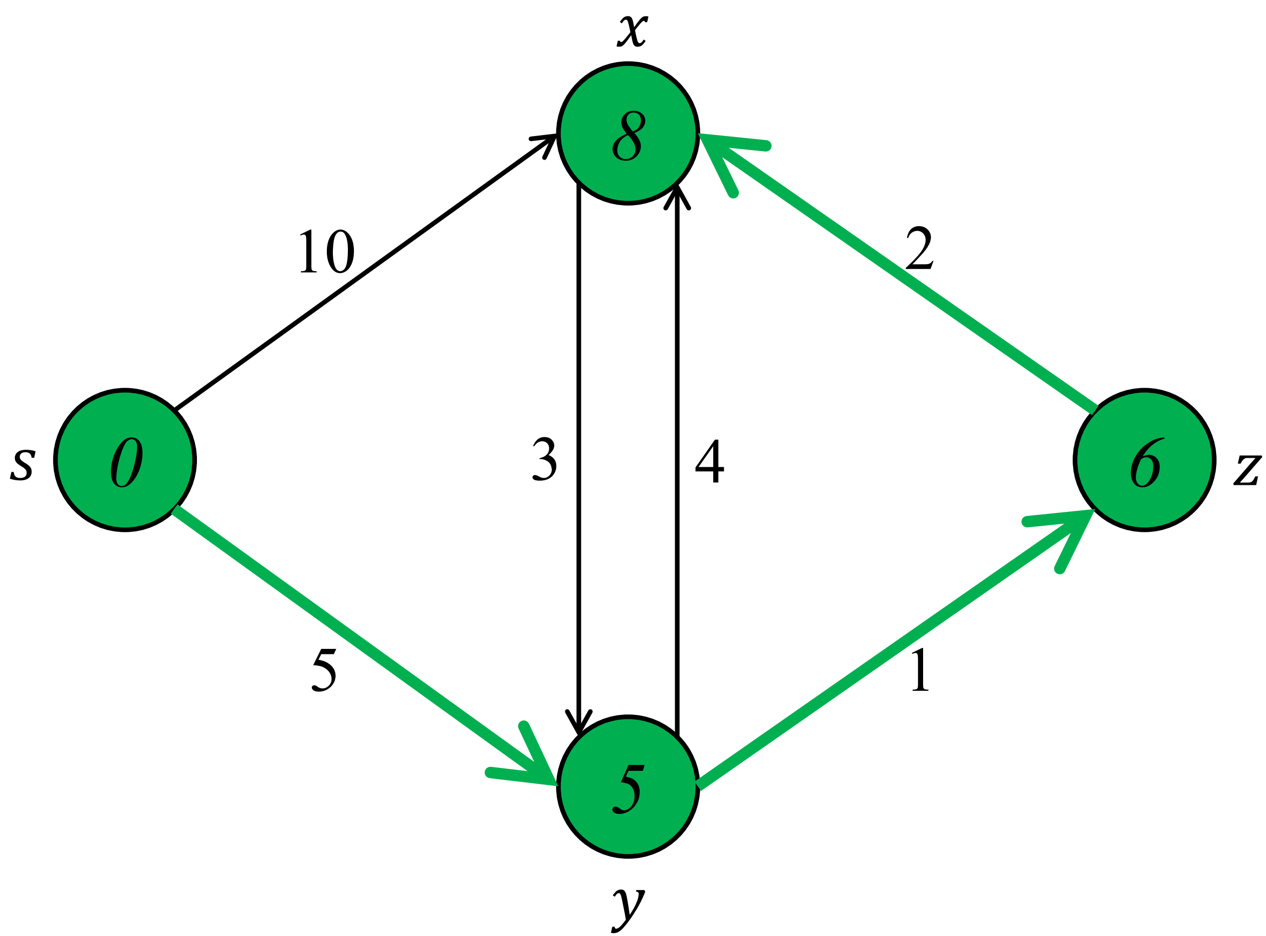
Dijkstra-Algorithmus

$S = \{s, y, z\}$, relaxiere (z, x) → $Q = [(x, 8)]$



Dijkstra-Algorithmus

$S = \{s, x, y, z\}$, relaxiere (x, y) → $Q = [\,]$



Beweis

- Mittels Schleifeninvariante
  „Zu Beginn jeder Iteration der while-Schleife gilt $v.d = \delta(s, v)$ für alle Knoten d in S“

- Beweis: zeige Initialisierung, Fortsetzung, Terminierung der Schleifeninvariante
  - Details siehe Buch

Nächstes Mal

- Noch mehr Graphenalgorithmen