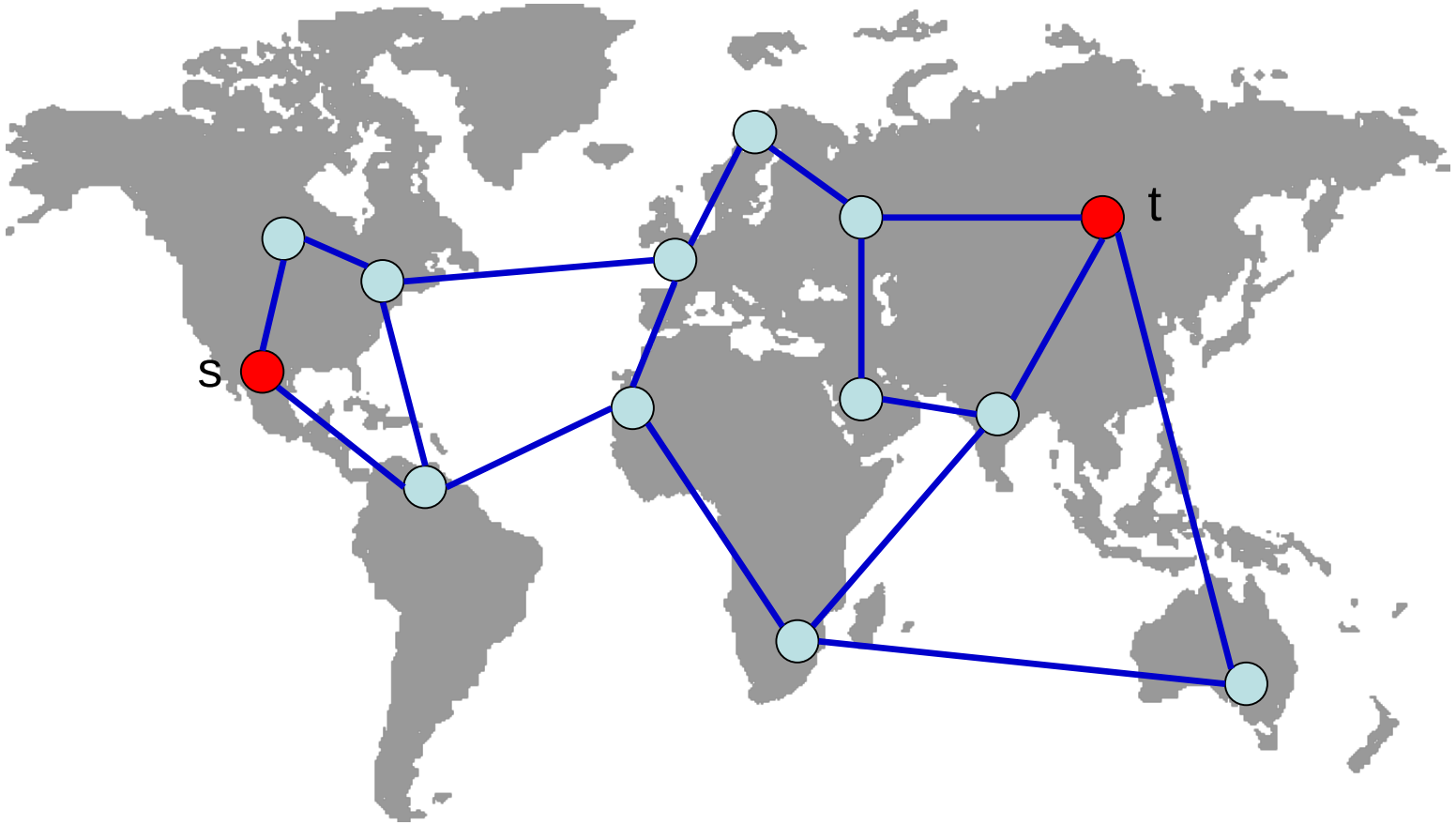
15. Kürzeste Wege



Gewichtete Graphen

- Ein *gewichteter Graph* G ist ein Paar (V,E) zusammen mit einer Gewichtsfunktion w, wobei $E \subseteq V \times V$ und $w: E \rightarrow \mathbb{R}$.

- Für $e \in E$ heißt w(e) das Gewicht von e. Für einen Weg p mit Kanten $e_1,..e_k$ heißt $w(p)=w(e_1)+...+w(e_k)$ das Gewicht von p.

- Ein *kürzester Weg* von Knoten u zum Knoten v ist der Weg mit dem kleinsten Gewicht von u nach v.

Berechnung kürzester Wege

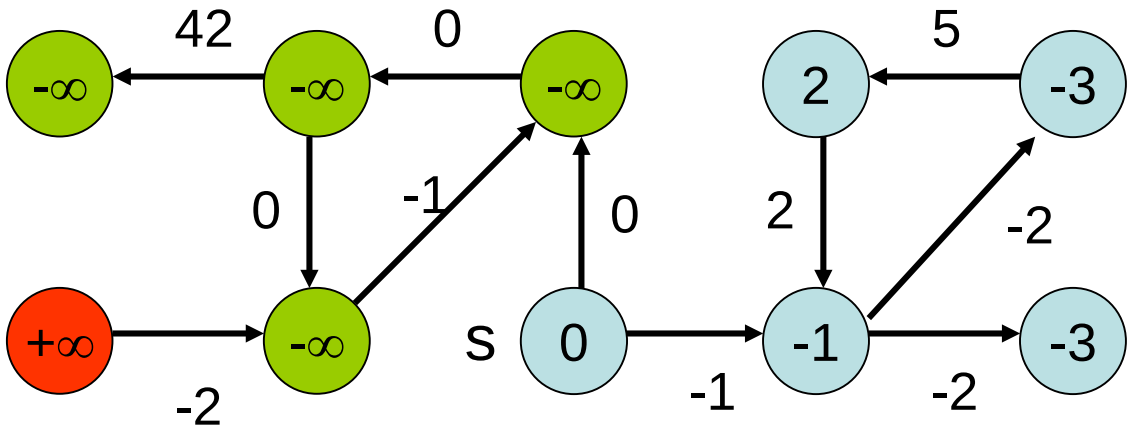
Probleme:

- Kürzeste Wege von einem Startknoten s
  (Single Source Shortest Path)

- Kürzeste Wege zwischen allen Knotenpaaren
  (All Pairs Shortest Path)

- Single Source Shortest Path (SSSP):
  - Gegeben: Gewichteter Graph G und Startknoten s
  - Gesucht: Für alle Knoten v die Distanz $\delta(s,v)$ sowie ein kürzester Weg.

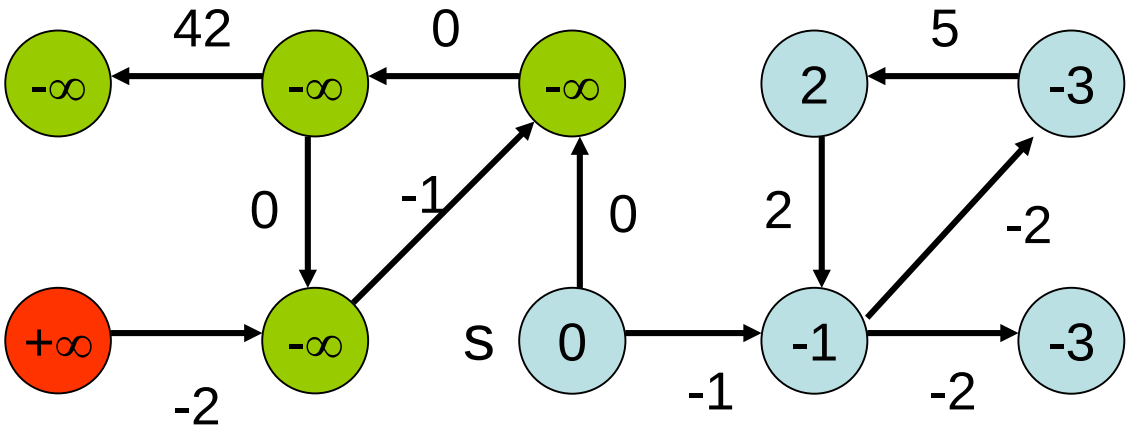
Kürzeste Wege



$\delta(s,v)$: Distanz zwischen s und v

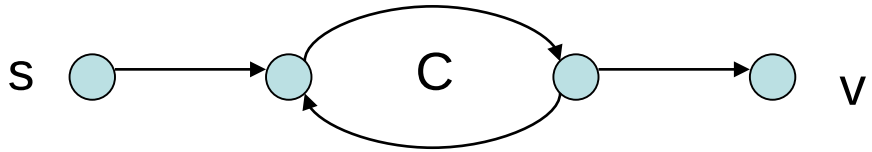
$$\delta(s,v) = \begin{cases} \infty & \text{kein Weg von } s \text{ nach } v \\ -\infty & \text{Weg bel. kleiner Kosten von } s \text{ nach } v \\ \min\{ w(p) \mid p \text{ ist Weg von } s \text{ nach } v\} \end{cases}$$

Kürzeste Wege



Wann sind die Kosten -∞?

Wenn es einen negativen Kreis gibt:

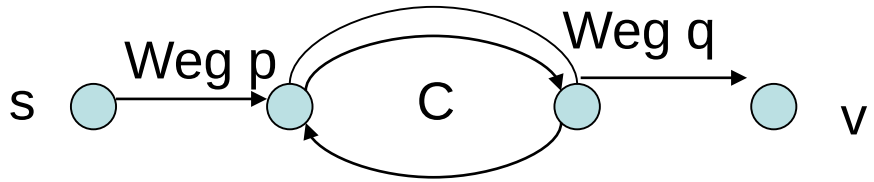


w(C)<0

Kürzeste Wege

Negativer Kreis hinreichend und notwendig für Wegekosten $-\infty$.

Negativer Kreis hinreichend:



$w(C)<0$

Kosten für i-fachen Durchlauf von C:

$$w(p) + i \cdot w(C) + w(q)$$

Für $i \to \infty$ geht Ausdruck gegen $-\infty$.

Kürzeste Wege

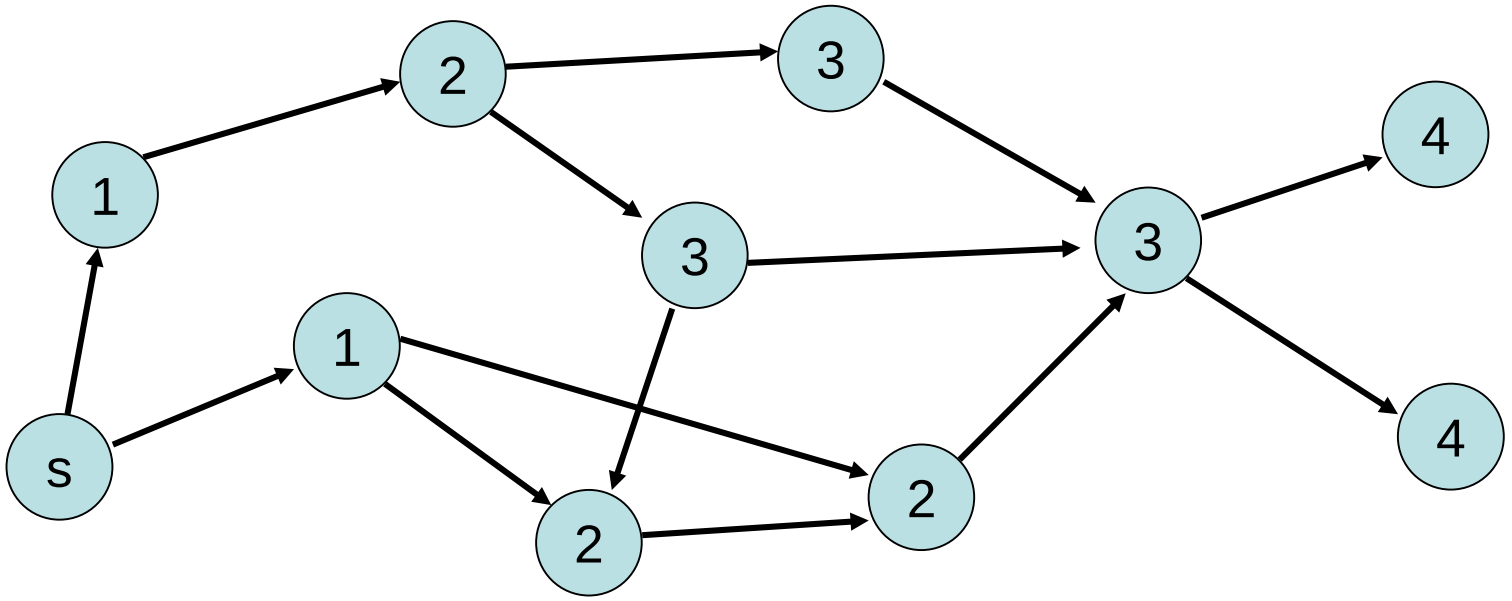
Negativer Kreis hinreichend und notwendig für Wegekosten $-\infty$.

Negativer Kreis notwendig:

- l: minimale Kosten eines einfachen Weges von s nach v
- es gibt nichteinfachen Weg r von s nach v mit Kosten $w(r)<l$
- r nicht einfach: Zerlegung in pCq, wobei C ein Kreis ist und pq ein einfacher Pfad
- da $w(r) < l \leq w(pq)$ ist, gilt $w(C)<0$

Kürzeste Wege in DAGs

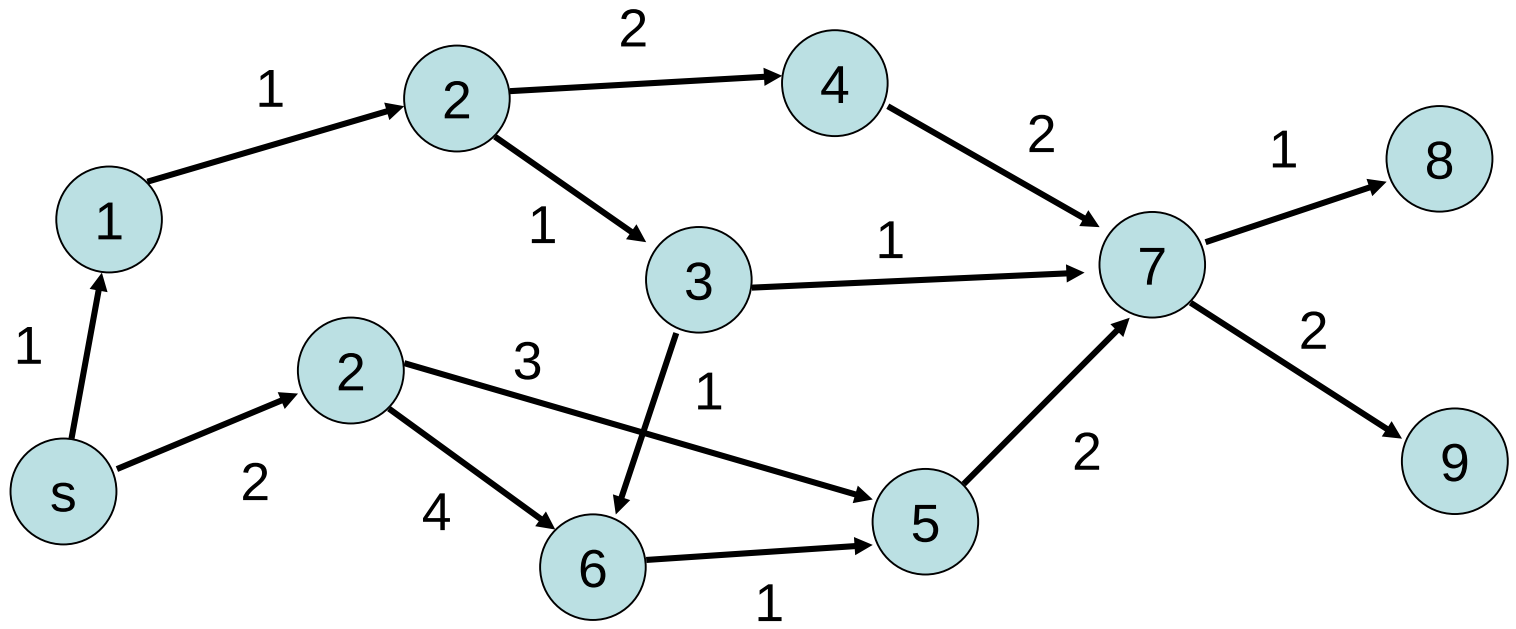
- Gerichteter azyklischer Graph (DAG): Graph ohne gerichtete Kreise
- DAG mit Kantenkosten 1: Breitensuche



Kürzeste Wege in DAGs

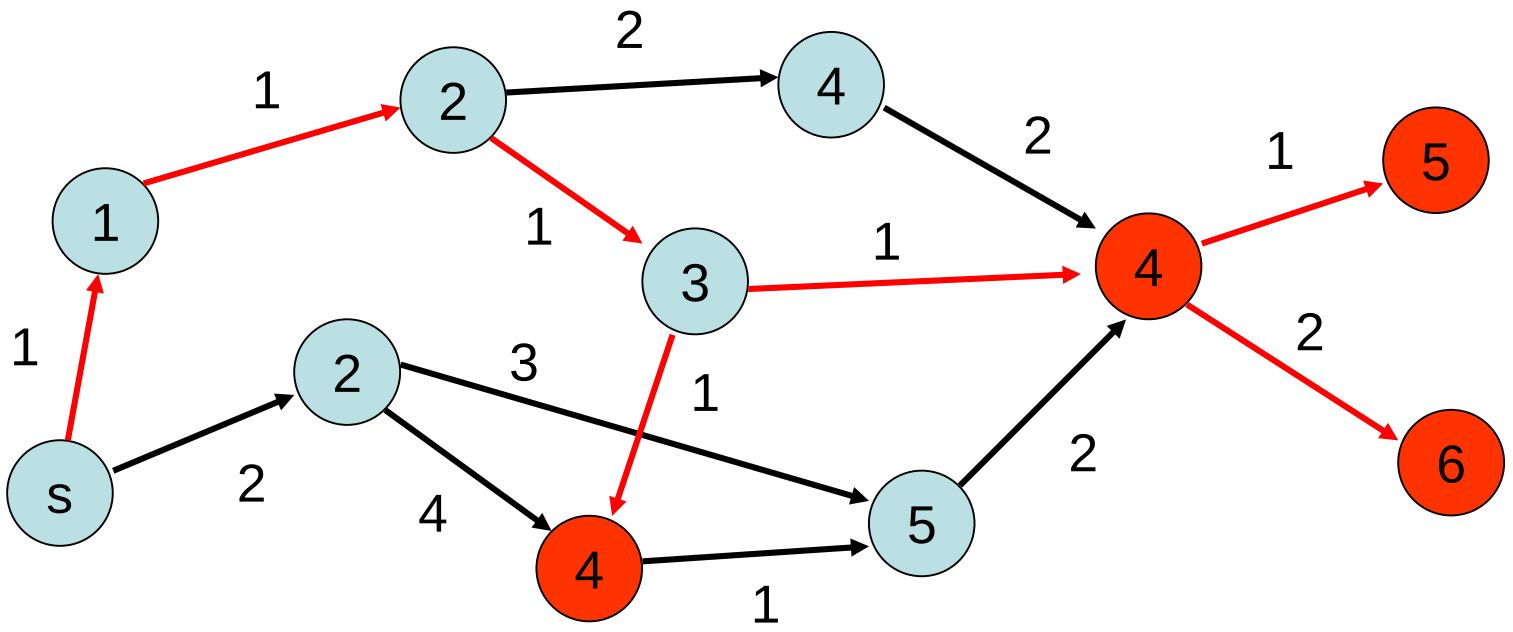
DAG mit beliebigen Kantenkosten:

Reine Breitensuche funktioniert nicht.

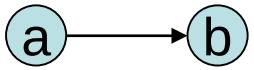


Kürzeste Wege in DAGs

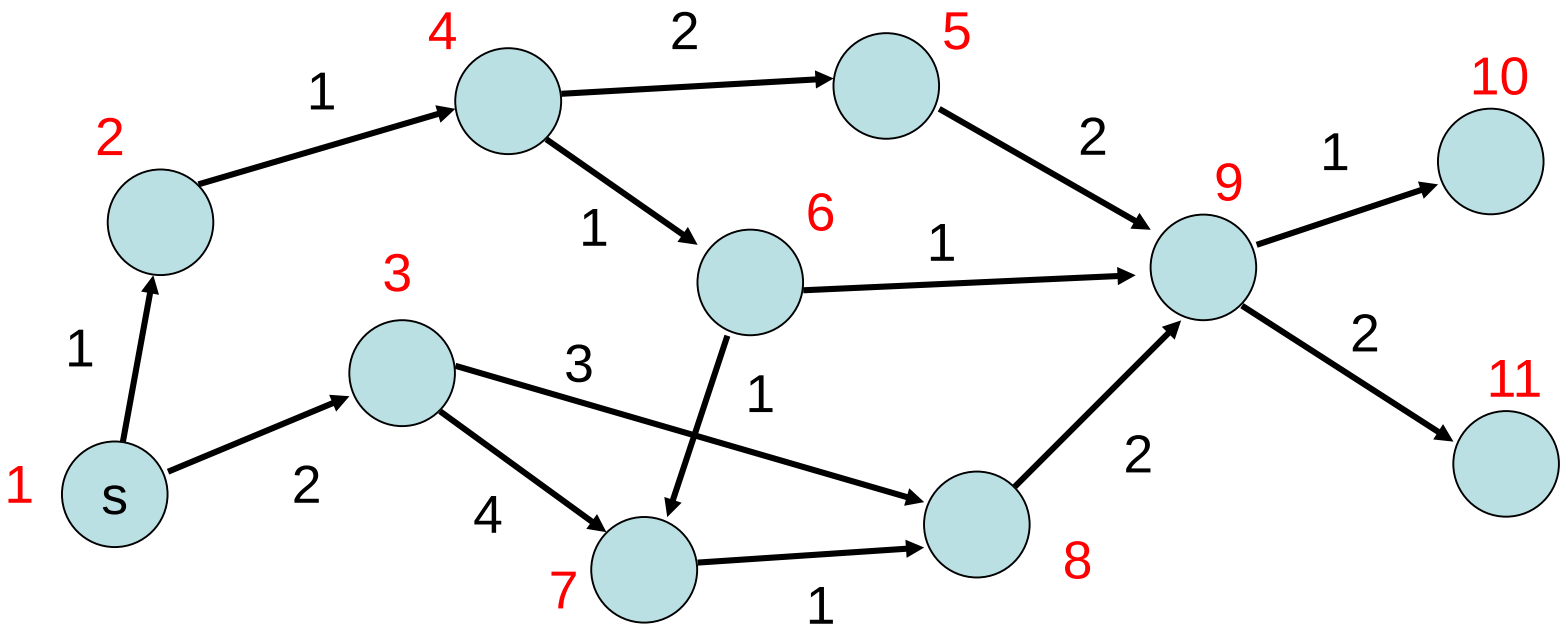
Korrekte Distanzen:



Kürzeste Wege in DAGs

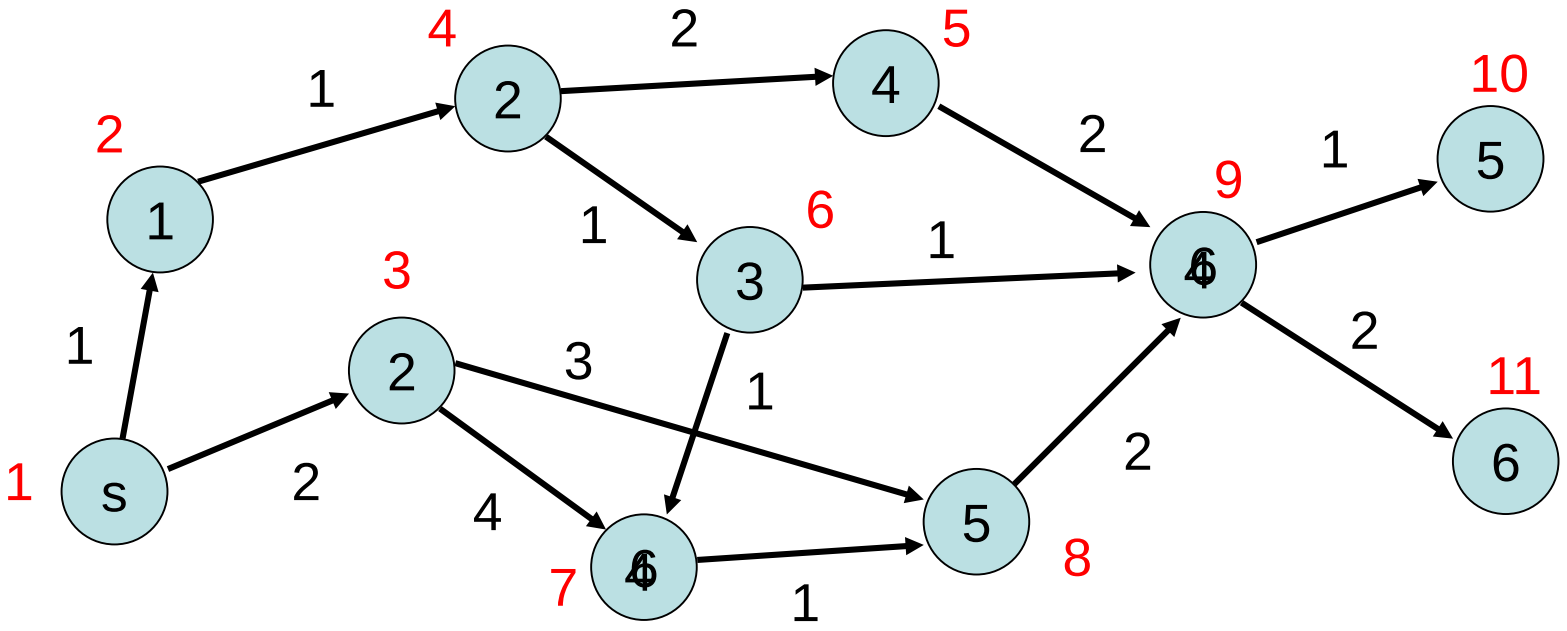
Strategie: nutze aus, dass Knoten in DAGs topologisch sortiert werden können (alle Kanten 

erfüllen a<b)



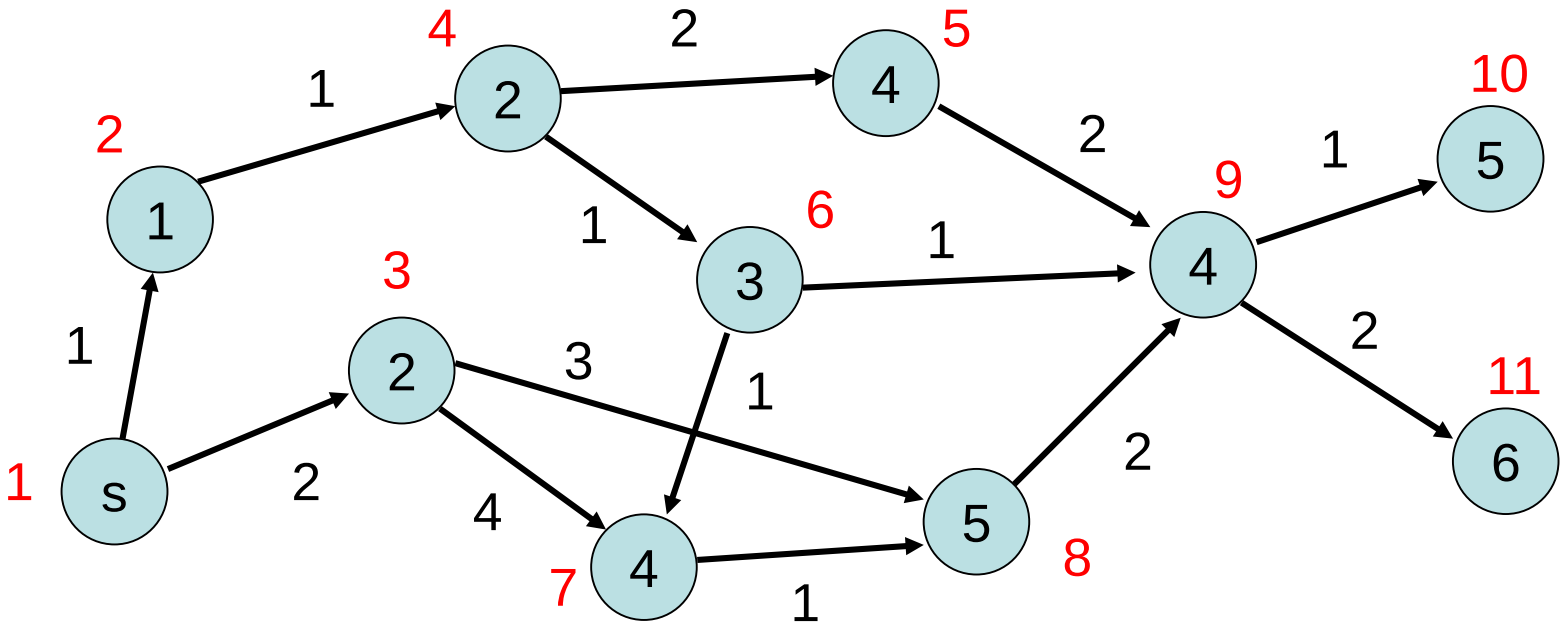
Kürzeste Wege in DAGs

Strategie: betrachte dann Knoten in der Reihenfolge ihrer topologischen Sortierung und aktualisiere Distanzen zu s



Kürzeste Wege in DAGs

Konkret: für jeden besuchten Knoten u:
für alle Kanten (u,v)∈E:
setze Distanz d[v] auf min{d[v],d[u]+w(u,v)}



Kürzeste Wege in DAGs

Strategie:

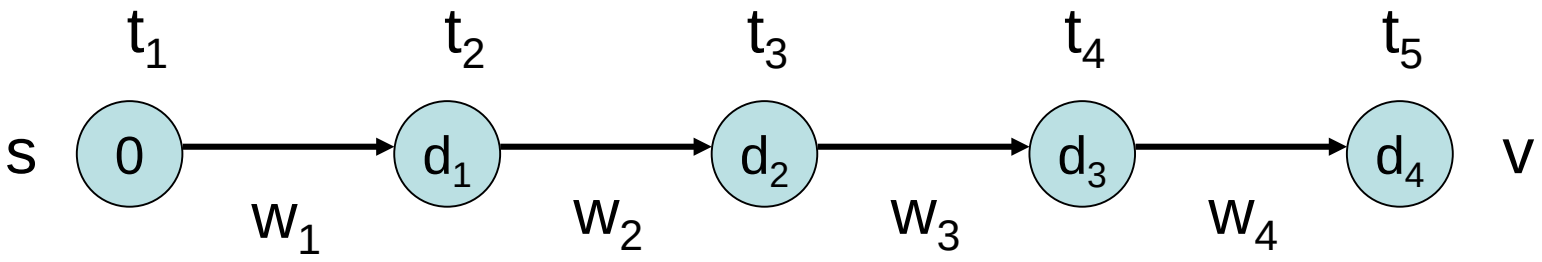
1. Topologische Sortierung der Knoten
2. Aktualisierung der Distanzen gemäß der topologischen Sortierung

Warum funktioniert das??

Kürzeste Wege in DAGs

Betrachte kürzesten Weg von s nach v.

Dieser hat topologische Sortierung $(t_i)_i$ mit $t_i<t_{i+1}$ for alle i.

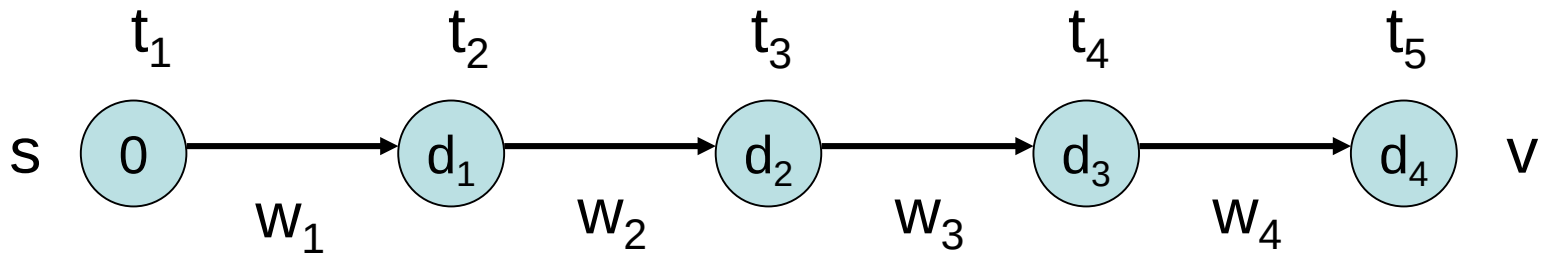


Besuch in topologischer Reihenfolge führt zu richtigen Distanzen ($d_i = \sum_{j \le i} w_i$).

Kürzeste Wege in DAGs

Betrachte kürzesten Weg von s nach v.

Dieser hat topologische Sortierung $(t_i)_i$ mit $t_i<t_{i+1}$ für alle i.



Bemerkung: kein Knoten auf dem Weg zu v kann Distanz $< d_i$ zu s haben, da sonst kürzerer Weg zu v möglich wäre.

Allgemeine Strategie:

- Am Anfang, setze $d(s):=0$ und $d(v):=\infty$ für alle Knoten
- besuche Knoten in einer Reihenfolge, die sicherstellt, dass mindestens ein kürzester Weg von s zu jedem v in der Reihenfolge seiner Knoten besucht wird
- für jeden besuchten Knoten u, aktualisiere die Distanzen der Knoten v mit $(u,v) \in E$, d.h. setze $d(v) := \min\{d(v), d(u)+w(u,v)\}$

Kürzeste Wege in DAGs

Zurück zur Strategie:

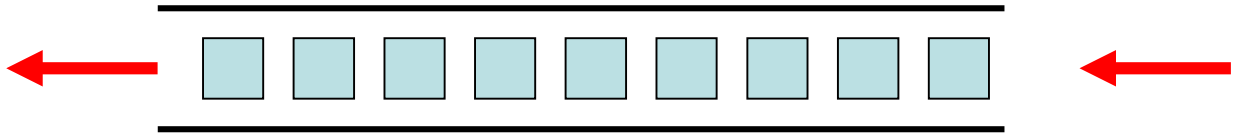
1. Topologische Sortierung der Knoten
2. Aktualisierung der Distanzen gemäß der topologischen Sortierung

Wie führe ich eine topologische Sortierung durch?

Kürzeste Wege in DAGs

Topologische Sortierung:

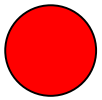
- Verwende eine FIFO Queue Q

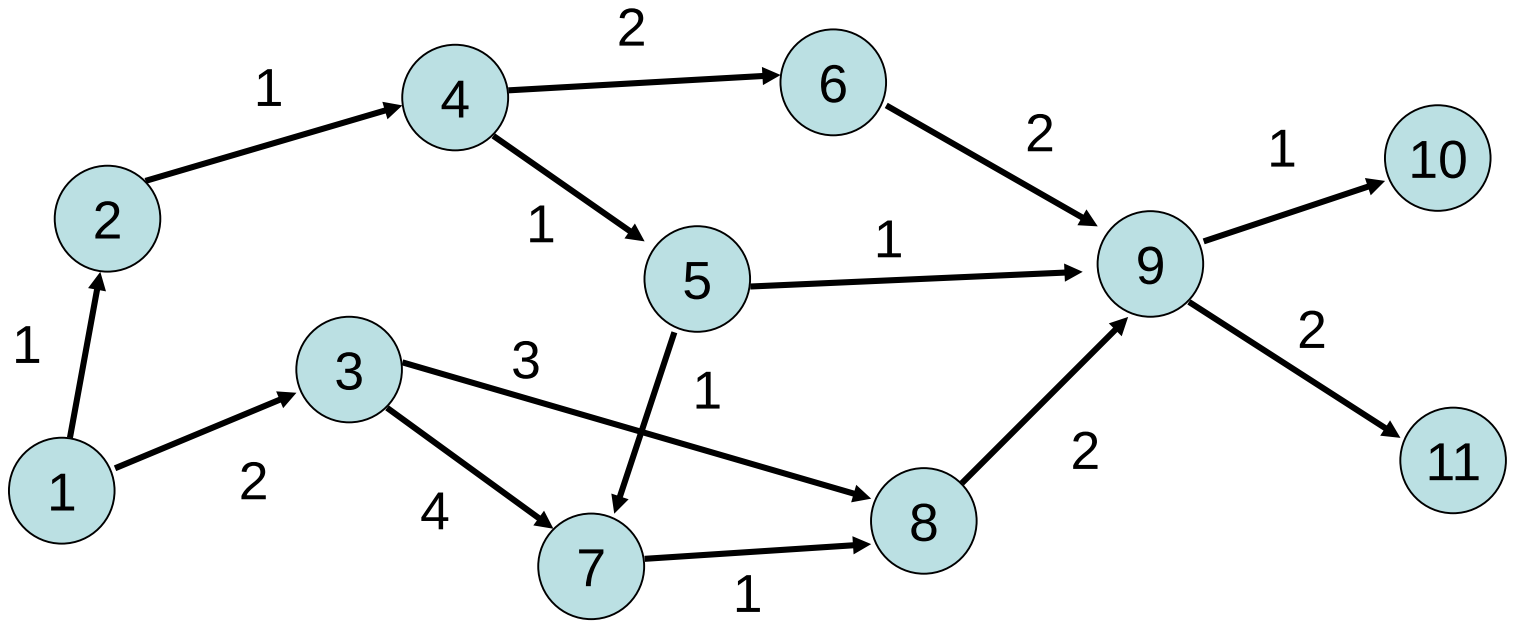


- Anfangs enthält Q alle Knoten, die keine eingehende Kante haben (Quellen).
- Solange Q nicht leer ist, entnehme v aus Q und markiere alle $(v,w) \in E$. Falls alle Kanten nach w markiert sind und w noch nicht in Q war, füge w in Q ein.
- Reihenfolge, in der Knoten aus Q entnommen werden, ergibt topologische Sortierung.

Kürzeste Wege in DAGs

Beispiel:

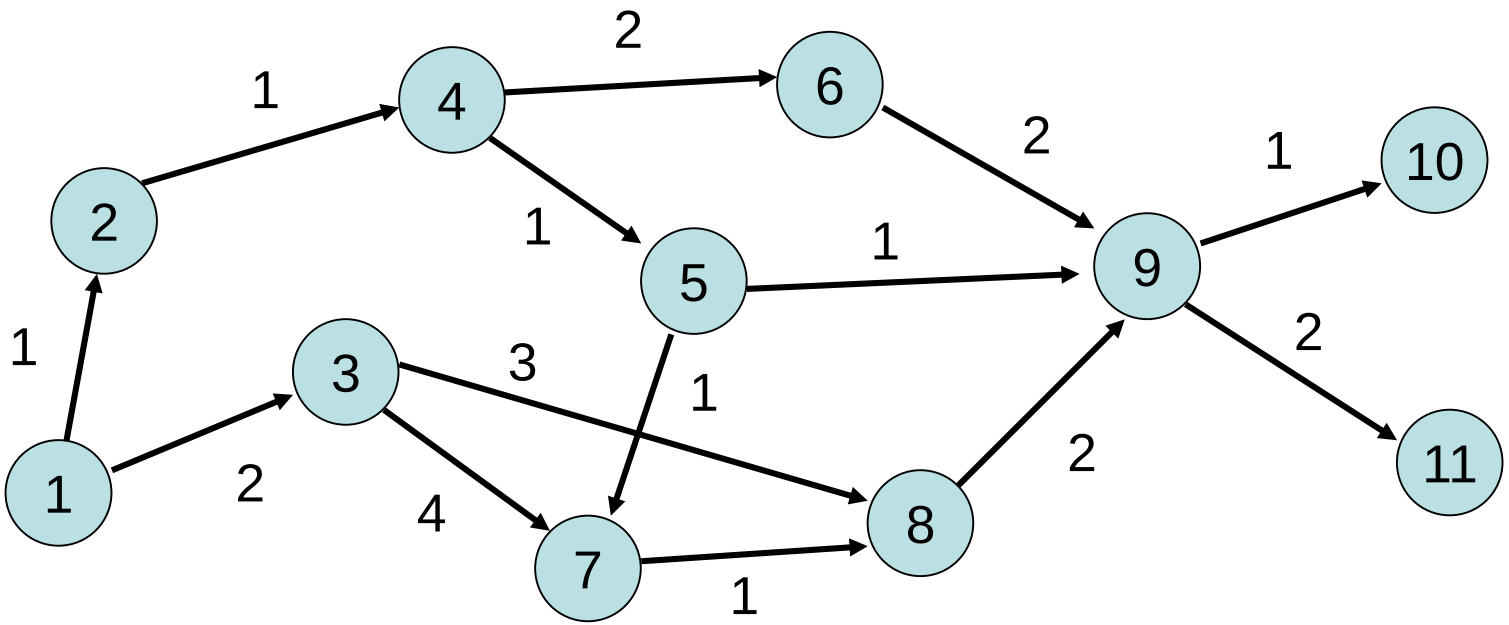
-  : Knoten momentan in Queue Q
- Nummerierung nach Einfügereihenfolge



Kürzeste Wege in DAGs

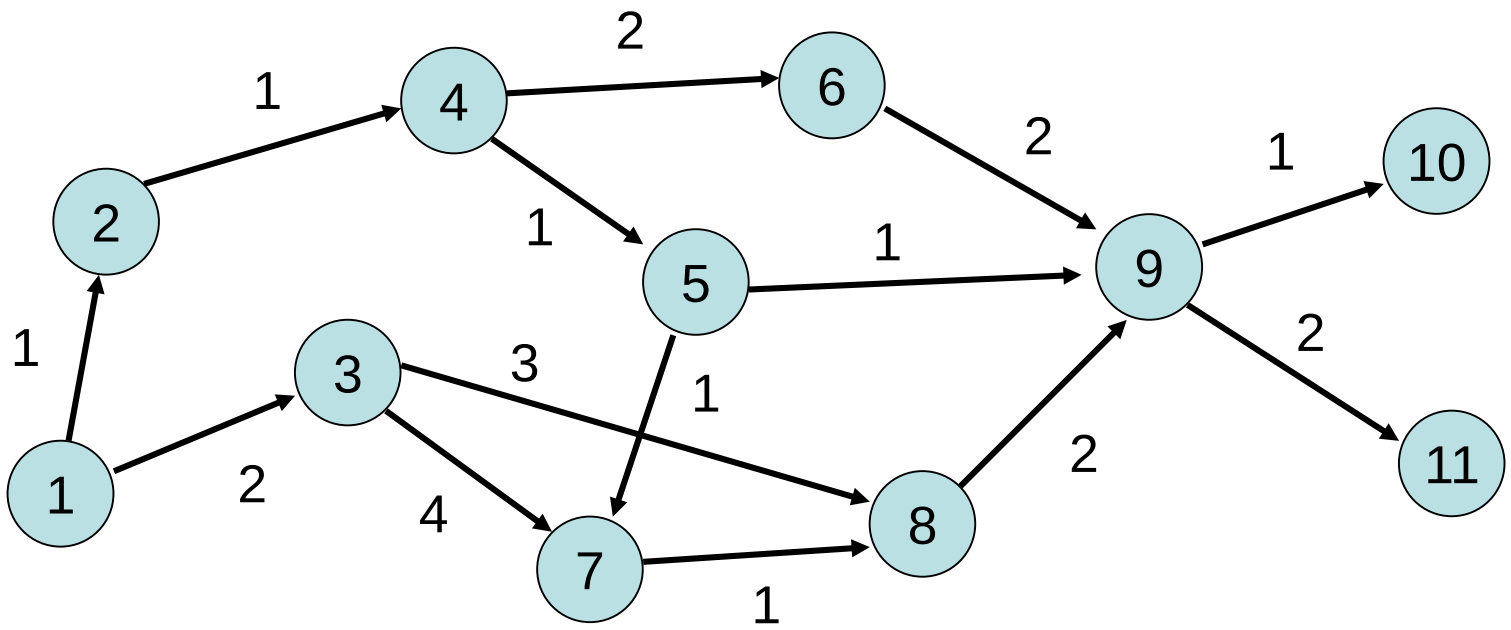
Korrektheit der topologischen Nummerierung:

Knoten wird erst dann nummeriert, wenn alle Vorgänger nummeriert sind.



Kürzeste Wege in DAGs

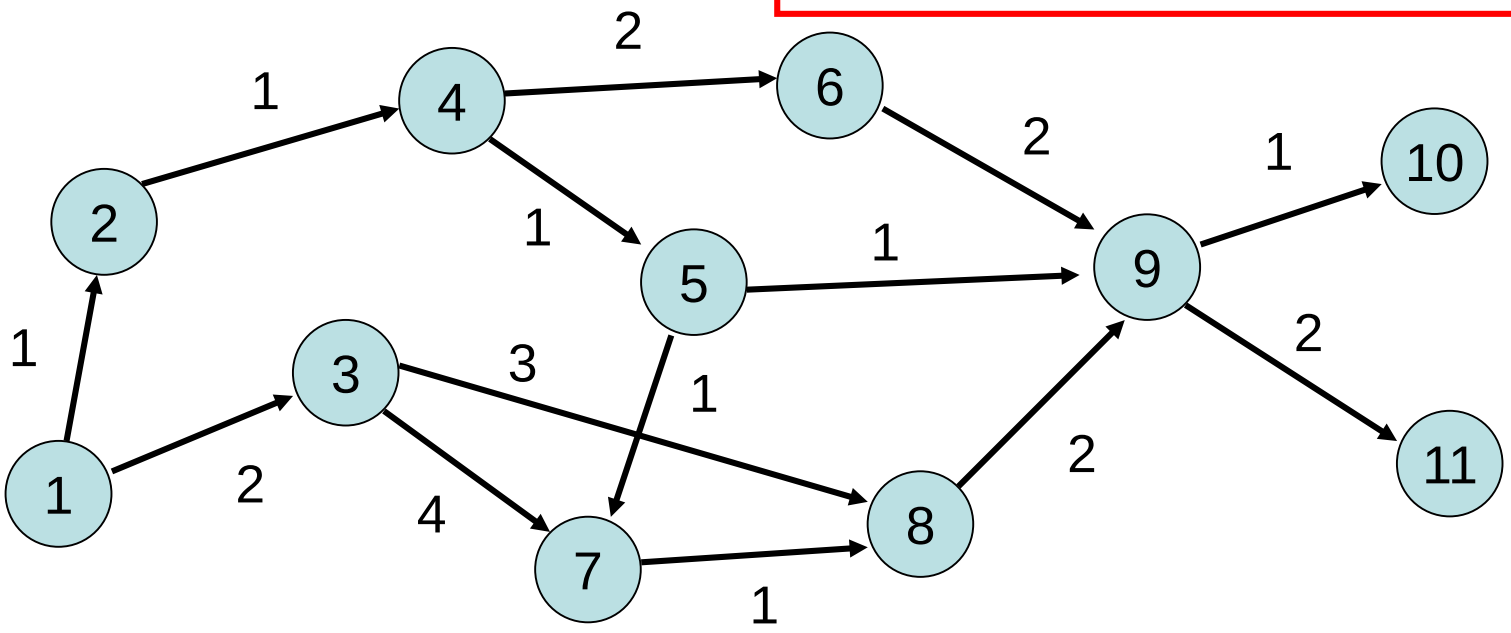
Laufzeit: Zur Bestimmung aller Knoten ohne eingehende Kante muss Graph einmal durchlaufen werden. Danach wird jeder Knoten und jede Kante genau einmal betrachtet, also Zeit O(n+m).



Kürzeste Wege in DAGs

Bemerkung: topologische Sortierung kann nicht alle Knoten nummerieren genau dann, wenn Graph gerichteten Kreis enthält (kein DAG ist)

Test auf DAG-Eigenschaft



Kürzeste Wege in DAGs

DAG-Strategie:

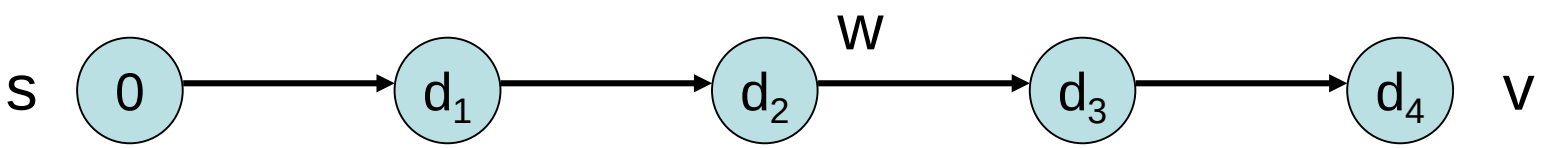
1. Topologische Sortierung der Knoten
   Laufzeit $O(n+m)$
2. Aktualisierung der Distanzen gemäß der topologischen Sortierung
   Laufzeit $O(n+m)$

Insgesamt Laufzeit $O(n+m)$.

Kürzeste Wege

Nächster Schritt: Kürzeste Wege für beliebige Graphen mit positiven Kantengewichten.

Problem: besuche Knoten eines kürzesten Weges in richtiger Reihenfolge



Lösung: besuche Knoten in der Reihenfolge der Distanz zur Quelle s

Kürzeste Wege

Annahme: Alle Kantengewichte sind positiv

Algorithmus von Dijkstra:

1. Es sei S die Menge der entdeckten Knoten
2. Zu Beginn: S={s} und d[s]=0
3. **while** $V \neq S$ **do**
4. Wähle Knoten $v \in V \setminus S$ mit mindestens einer Kante aus S und für den $d[v] := \min_{(u,v) \in (S, V \setminus S)} (d[u] + w(u,v))$ minimal unter allen $v \in V \setminus S$ ist
5. Füge v zu S hinzu

Kürzeste Wege

Wie kann man Pfade berechnen?

- Wie bei BFS/DFS über Feld π
- Wenn (u,v) die Kante ist, für die das Minimum in Zeile 4 erreicht wird, dann setze $\pi[v] \leftarrow u$

Kürzester s-u-Weg $P(u)$ ist implizit gespeichert:

- Für $u=s$ haben wir den leeren Weg als kürzesten Weg von s nach s
- Für $u \neq s$ gilt:
  $P(u)$ besteht aus Weg $P(\pi(u))$ gefolgt von Kante $(\pi[u],u)$

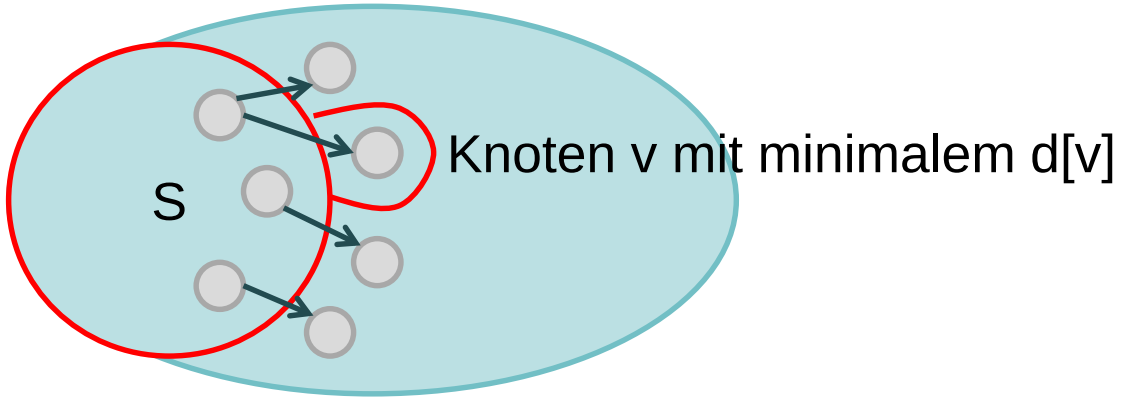
Satz 15.1 (Korrektheit):

- $\delta(s,u)$: Länge des kürzesten Weges von s nach u
- $\Gamma(S)$: Menge der Knoten in $V \setminus S$ mit Kante aus S

Invariante:

1. Für jedes $u \in S$: $d[u]=\delta(s,u)$ und $d[u] \leq \min_{v \in \Gamma(S)} \delta(s,v)$
2. Für jedes $v \in \Gamma(S)$: $d[v] \geq \delta(s,v)$

Beweisskizze:



Kürzeste Wege

Wie kann man Dijkstras Algorithmus effizient implementieren?

- Naiver Ansatz: Überprüfe für jeden Knoten aus V\S alle Kanten (Laufzeit $O(|V| \cdot |E|)$)

Besser:

- Halte d[v]-Werte für alle $v \in V \setminus S$ aufrecht
- Speichere alle Knoten aus $V \setminus S$ in Heap ab mit Schlüssel d[v]

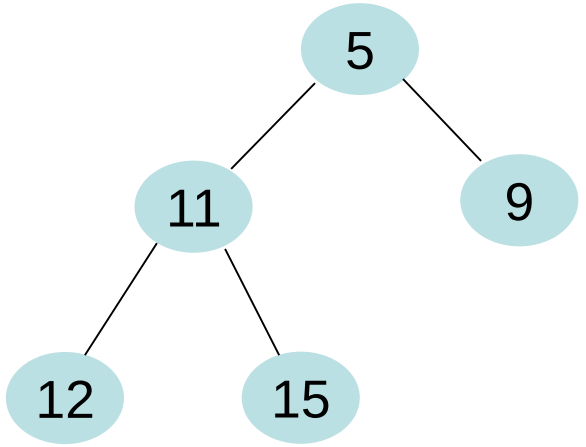
Problem: Was ist, wenn sich Schlüssel verändern

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements des Heaps, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]

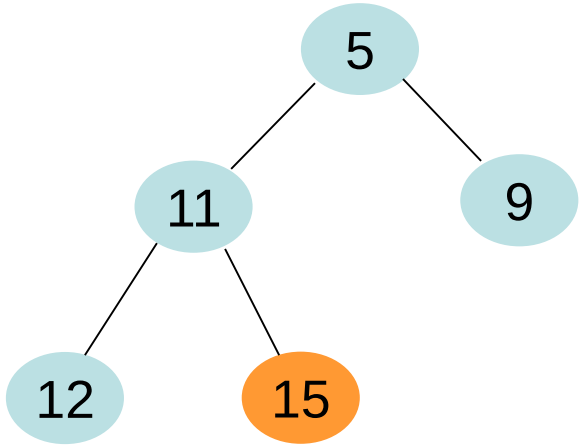


Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements des Heaps, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



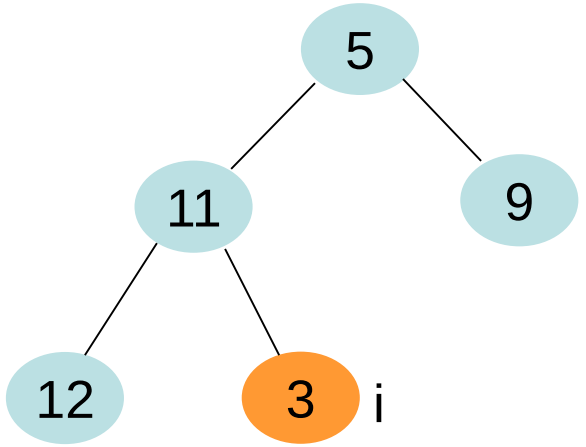
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



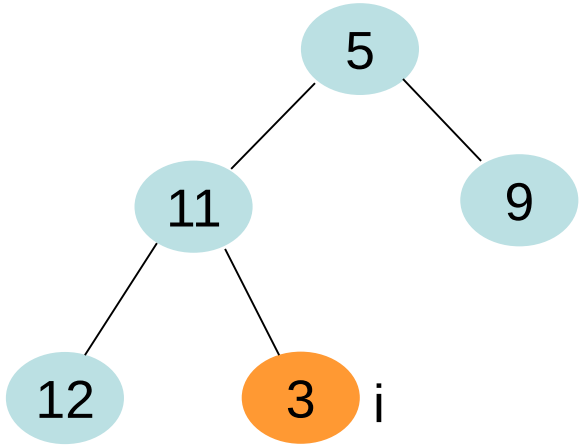
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



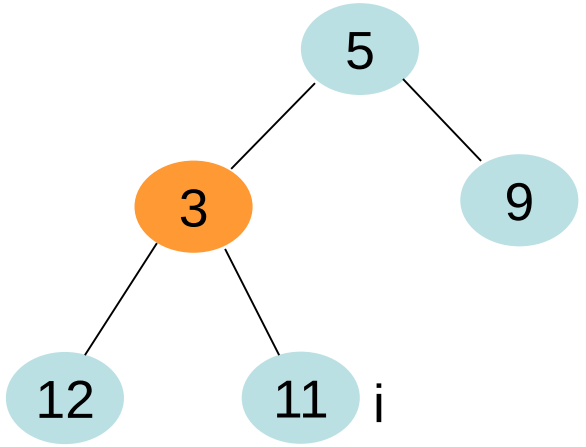
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



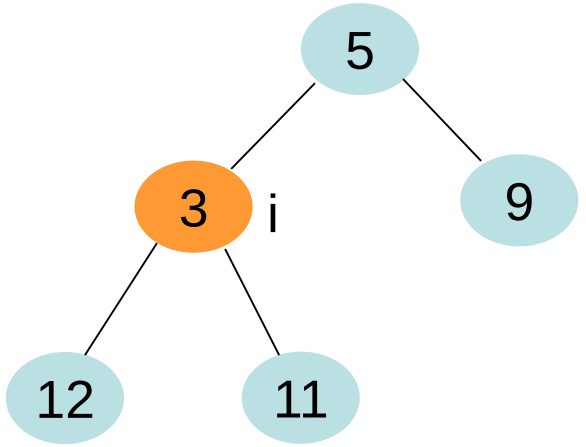
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



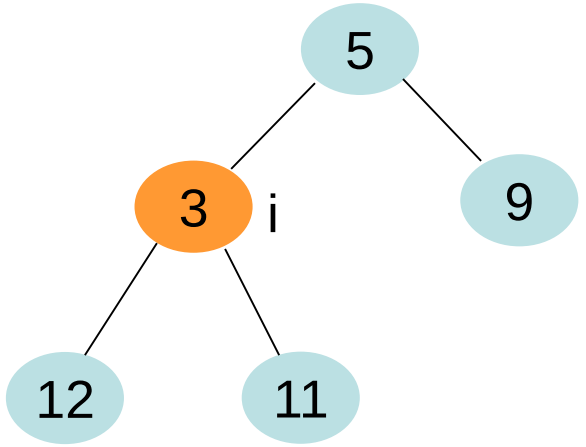
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



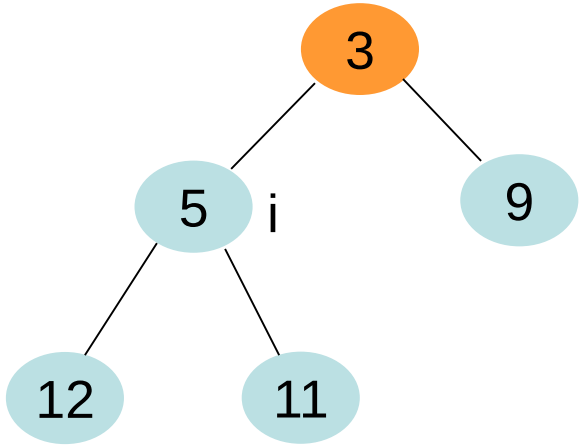
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



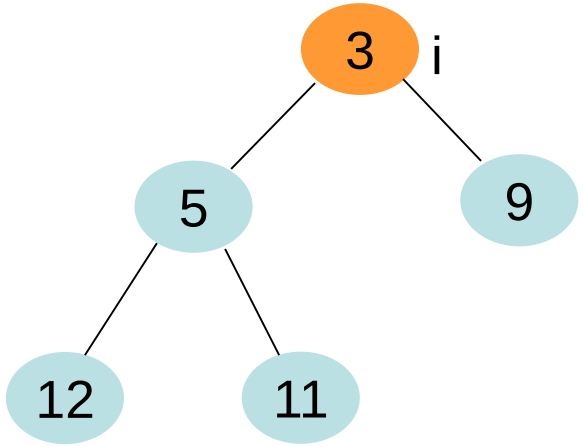
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



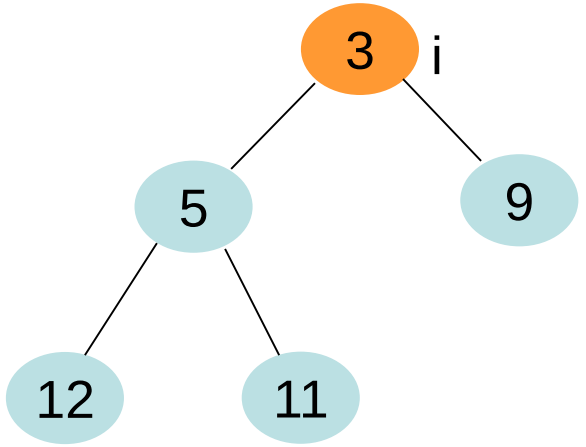
DecreaseKey(A,5,3)

Heaps mit Decrease-Key:

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



DecreaseKey(A,5,3)

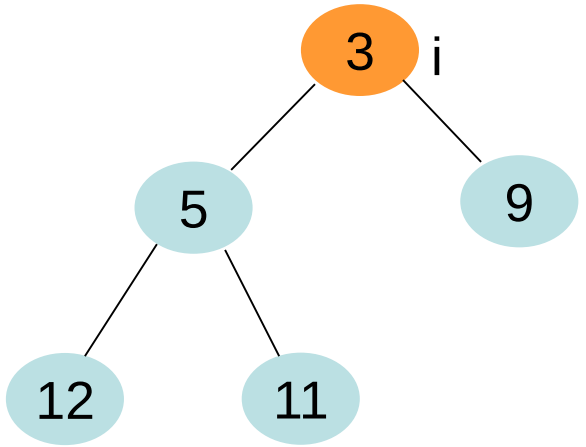
Heaps mit Decrease-Key:

Laufzeit: O(log n)

- Datenstruktur Min-Heap A
- Neue Funktion DecreaseKey(A, i, newkey):
  Wir bekommen Index i des Elements der Heap, dessen Wert (Schlüssel) auf newkey gesetzt wird
- newkey ist kleiner als der alte Schlüssel

Decrease-Key(A,i,newkey)

1. A[i] ← newkey
2. **while** i>1 and A[parent[i]]>A[i] **do**
3. A[parent[i]] ↔ A[i]
4. i ← parent[i]



DecreaseKey(A,5,3)

Wie kann man Dijkstras Algorithmus effizient implementieren?

- Halte d[v] Werte für alle $v \in V \setminus S$ aufrecht
- Speichere alle Knoten aus $V \setminus S$ in Prioritätenschlange ab mit Schlüssel d[v]
- Für jeden Knoten v speichere Zeiger auf sein Vorkommen im Heap
- d[v]-Werte vergrößern sich nie
- Benutze Decrease-Key, wenn sich Wert d[v] verringert

Kürzeste Wege

Dijkstra(G,w,s)

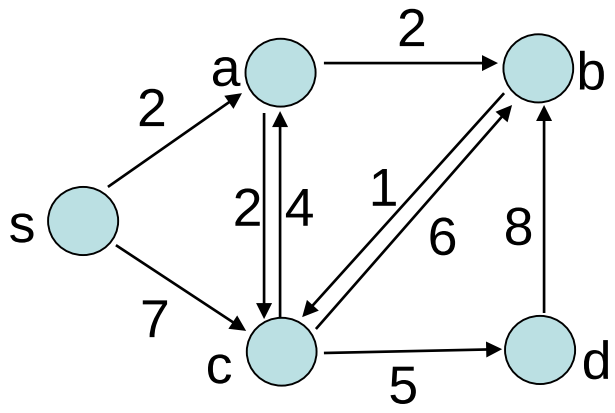
1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow$ BuildHeap(V)
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

Abkürzung für Decrease-Key(Q,v,d[u]+w(u,v)) Wir nehmen an, dass in v Pos. von v in Q vermerkt ist.

Kürzeste Wege

Dijkstra(G,w,s)

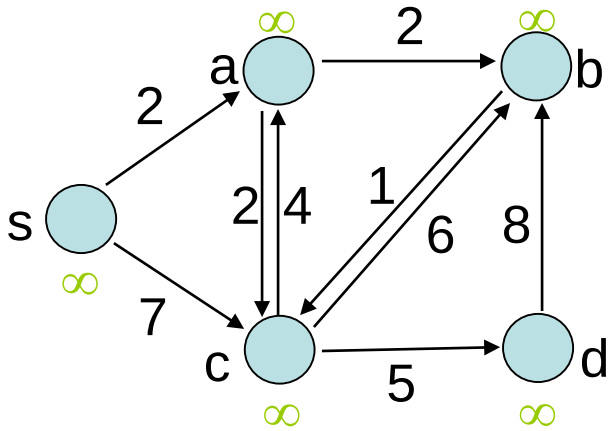
1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



Kürzeste Wege

Dijkstra(G,w,s)

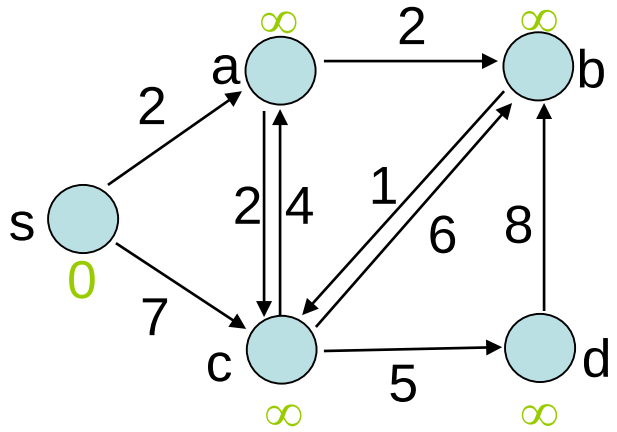
1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



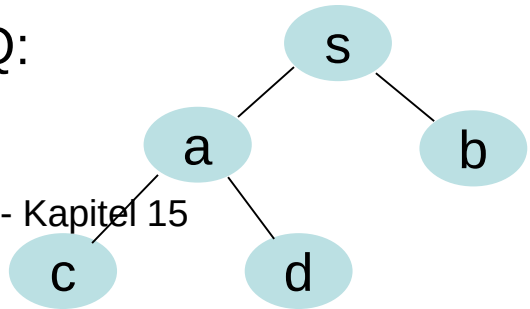
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



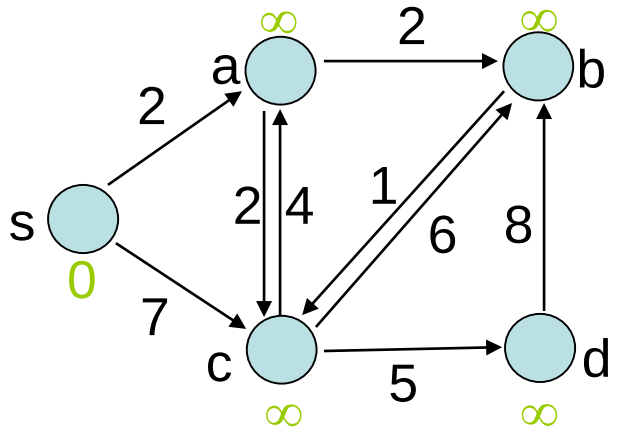
Q:



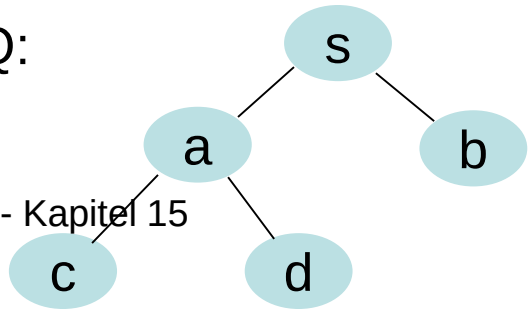
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



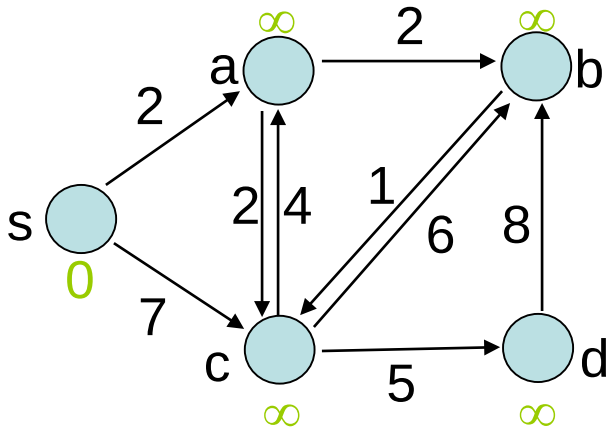
Q:



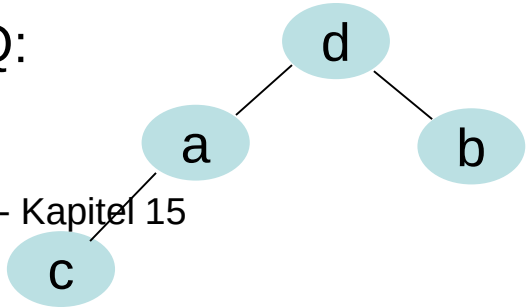
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $u \leftarrow$ deleteMin(Q)
5. $S \leftarrow S \cup \{u\}$
6. **for each** vertex $v \in$ Adj[u] **do**
7. **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



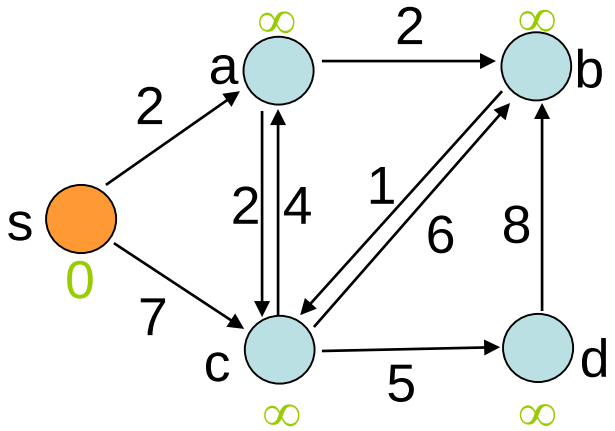
Q:



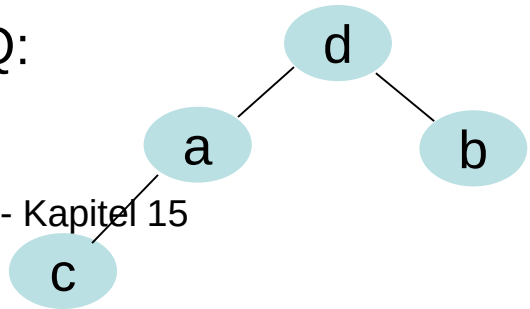
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



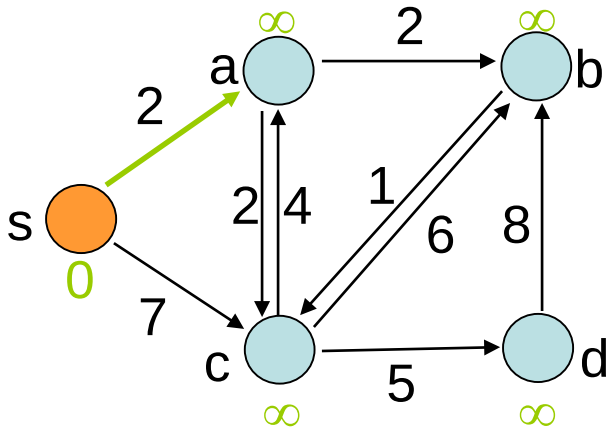
Q:



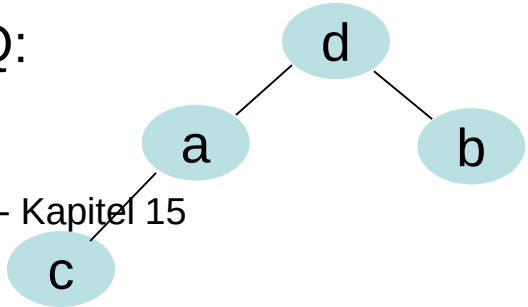
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex v∈V **do** d[v]←∞; π[v]←**nil**
2. d[s]←0; S←∅; Q←V
3. **while** Q≠∅ **do**
4. u←deleteMin(Q)
5. S←S∪{u}
6. **for each** vertex v∈Adj[u] **do**
7. **if** d[v]>d[u]+w(u,v) **then** DK(Q,v,d[u]+w(u,v)); π[v]←u



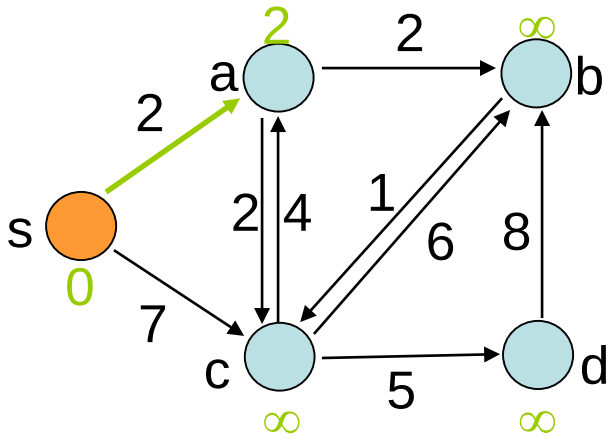
Q:



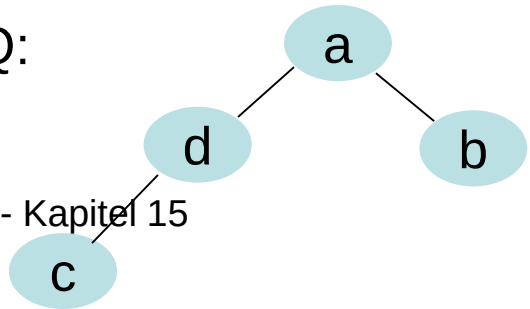
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



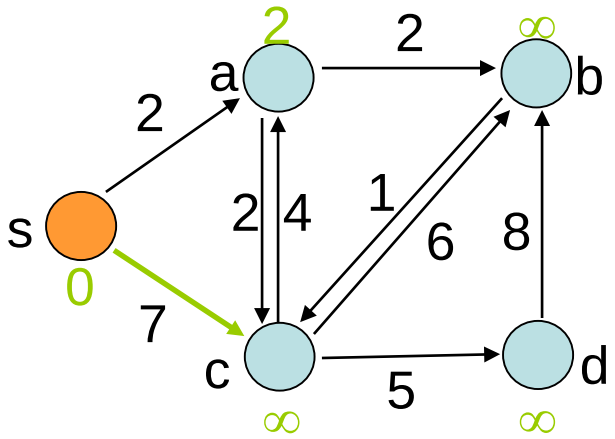
Q:



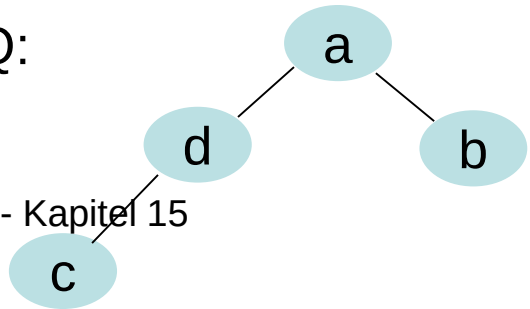
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



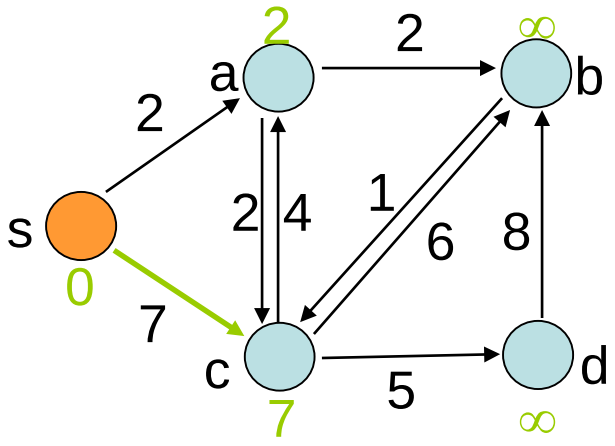
Q:



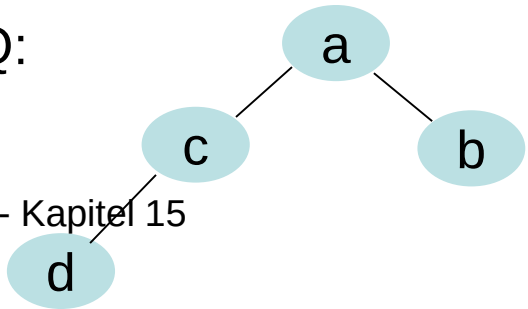
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



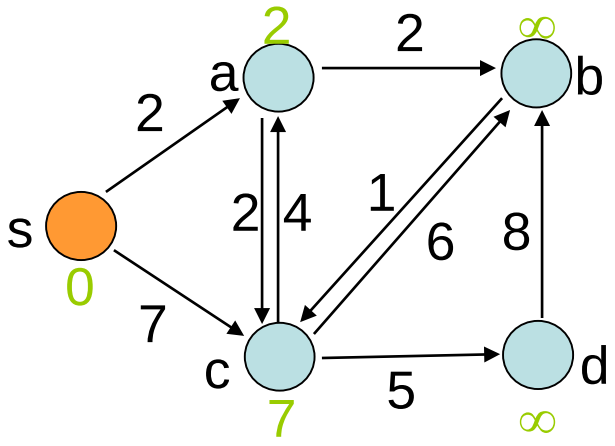
Q:



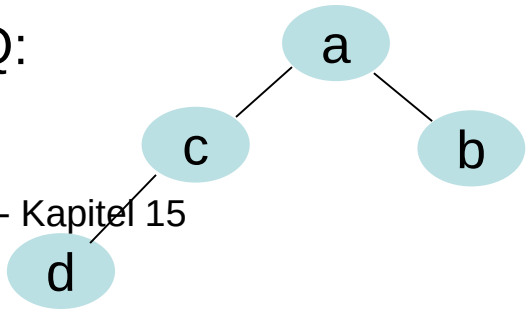
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



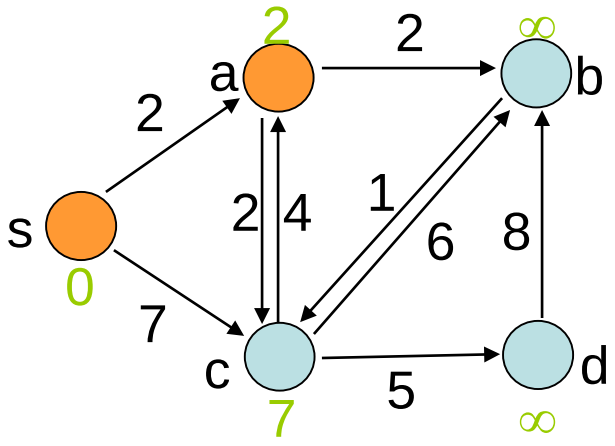
Q:



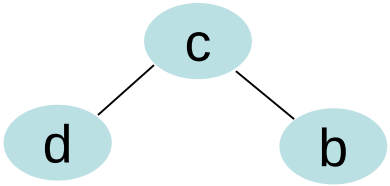
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



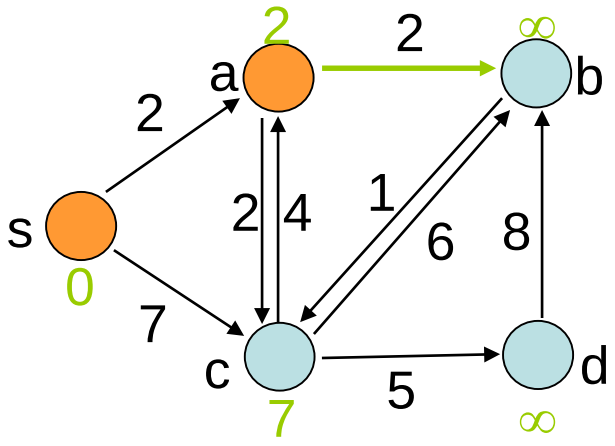
Q:



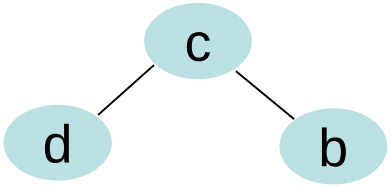
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



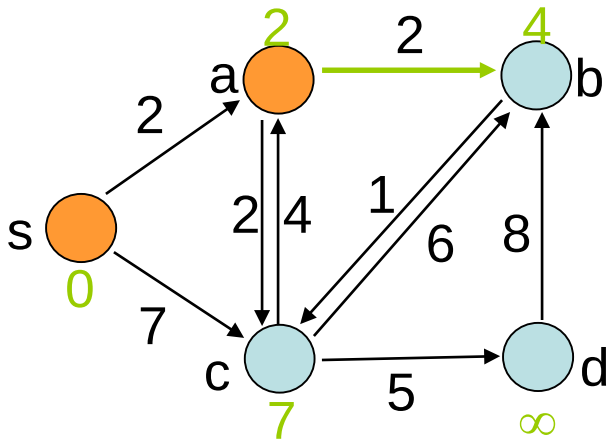
Q:



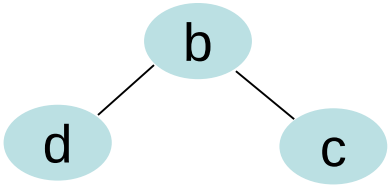
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



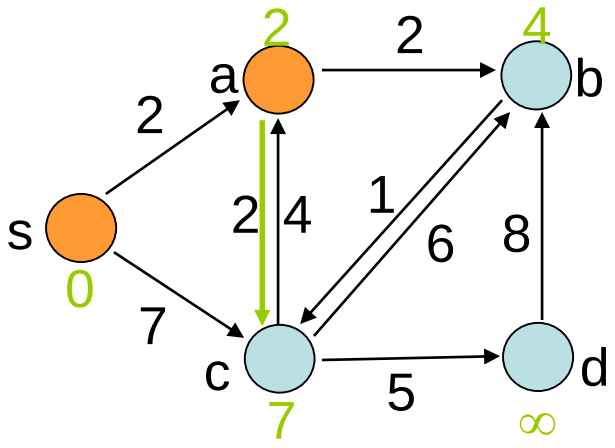
Q:



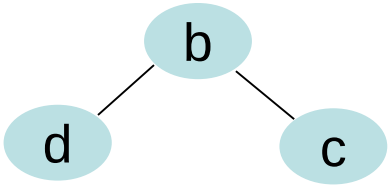
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



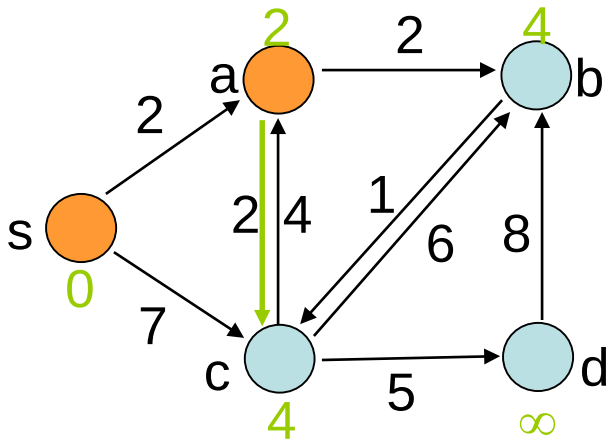
Q:



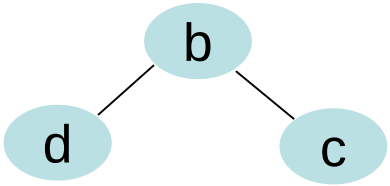
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow$ u



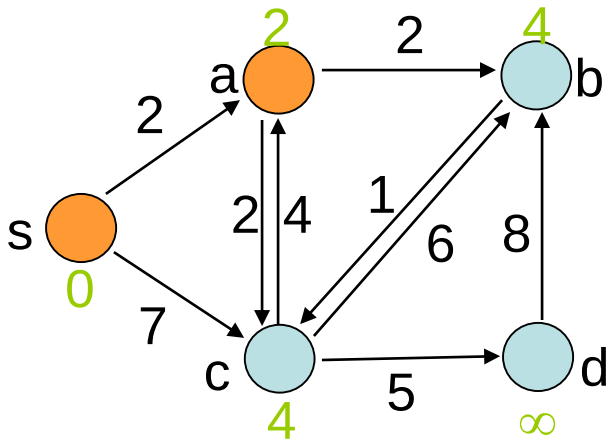
Q:



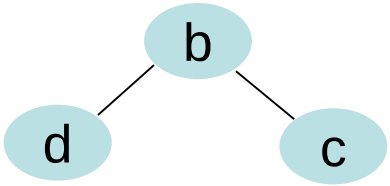
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



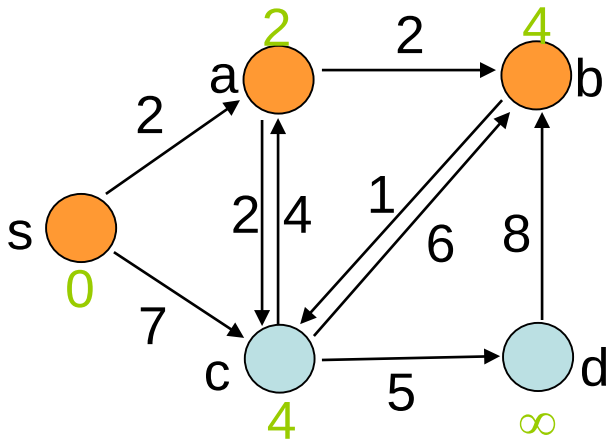
Q:



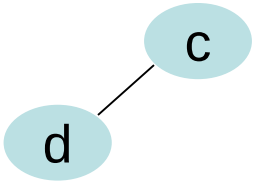
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



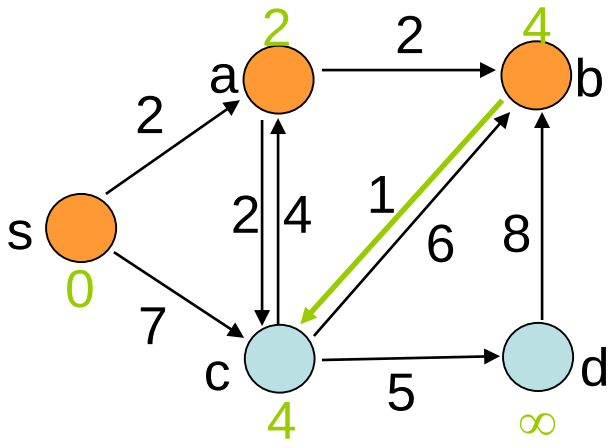
Q:



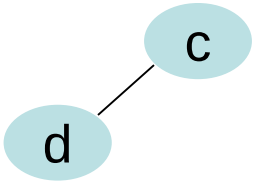
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



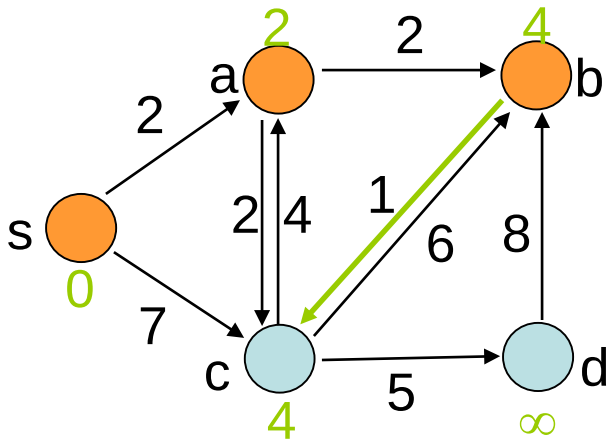
Q:



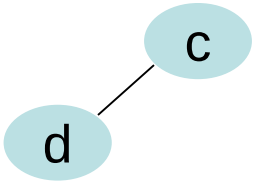
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



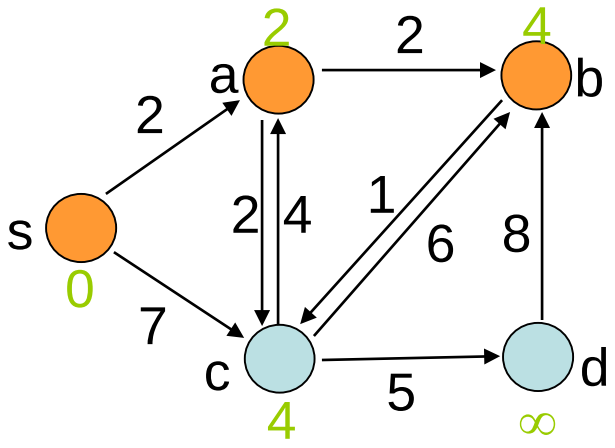
Q:



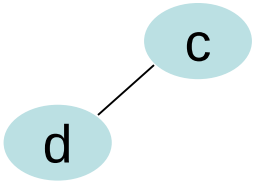
Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



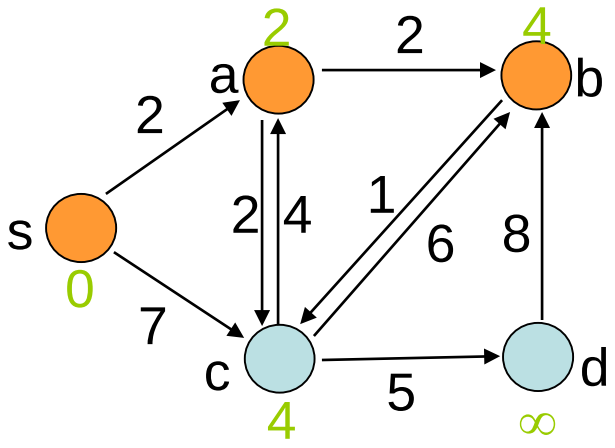
Q:



Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $u \leftarrow$ deleteMin(Q)
5. $S \leftarrow S \cup \{u\}$
6. **for each** vertex $v \in Adj[u]$ **do**
7. **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

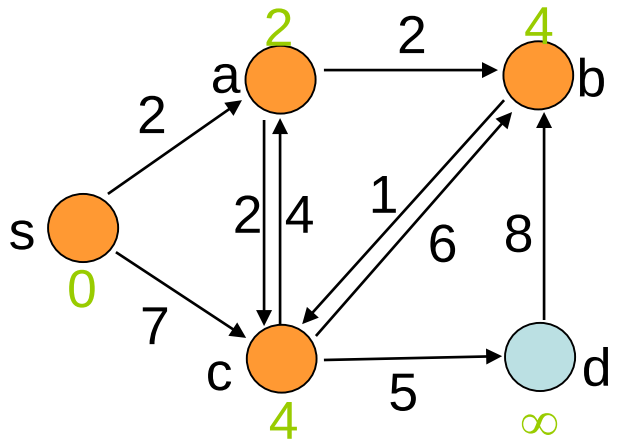


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

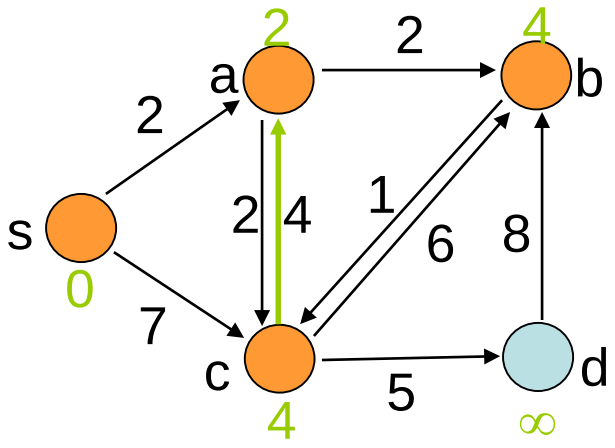


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex v∈V **do** d[v]←∞; π[v]←**nil**
2. d[s]←0; S←∅; Q←V
3. **while** Q≠∅ **do**
4. u←deleteMin(Q)
5. S←S∪{u}
6. **for each** vertex v∈Adj[u] **do**
7. **if** d[v]>d[u]+w(u,v) **then** DK(Q,v,d[u]+w(u,v)); π[v]←u

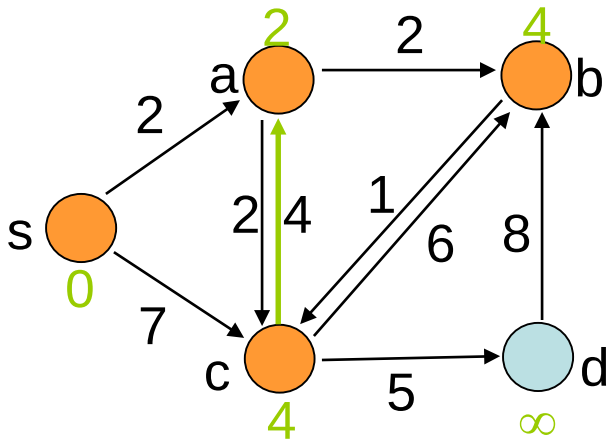


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

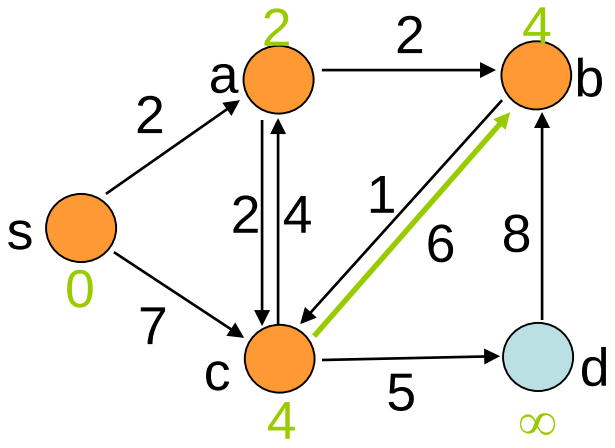


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex v∈V **do** d[v]←∞; π[v]←**nil**
2. d[s]←0; S←∅; Q←V
3. **while** Q≠∅ **do**
4. u←deleteMin(Q)
5. S←S∪{u}
6. **for each** vertex v∈Adj[u] **do**
7. **if** d[v]>d[u]+w(u,v) **then** DK(Q,v,d[u]+w(u,v)); π[v]←u

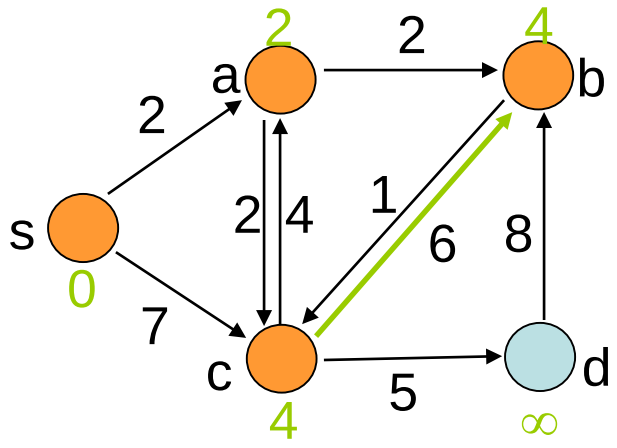


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

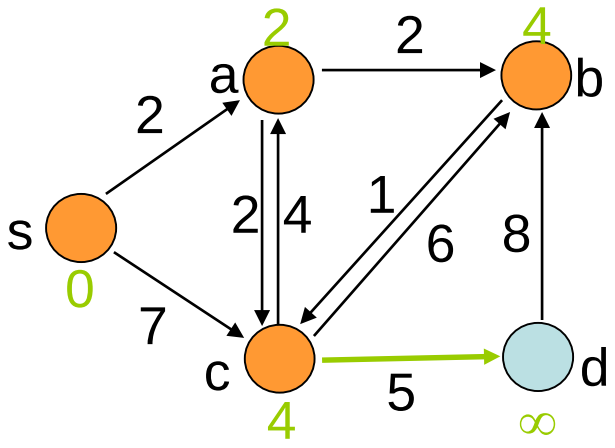


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex v∈V **do** d[v]←∞; π[v]←**nil**
2. d[s]←0; S←∅; Q←V
3. **while** Q≠∅ **do**
4. u←deleteMin(Q)
5. S←S∪{u}
6. **for each** vertex v∈Adj[u] **do**
7. **if** d[v]>d[u]+w(u,v) **then** DK(Q,v,d[u]+w(u,v)); π[v]←u

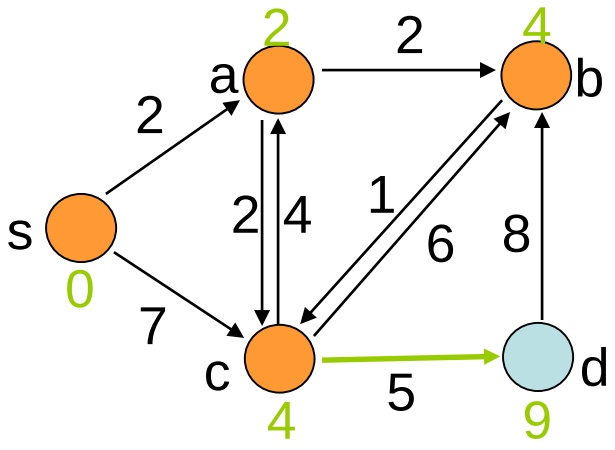


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

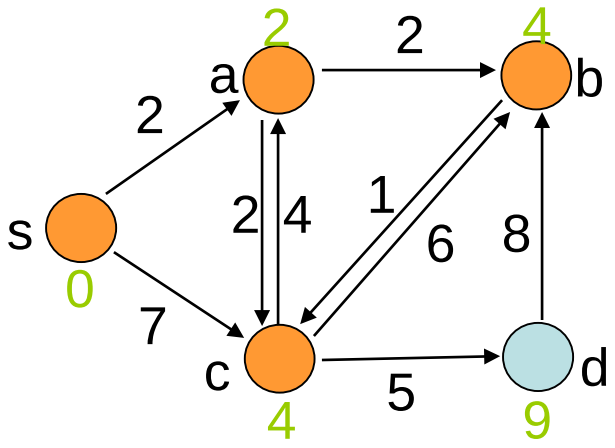


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

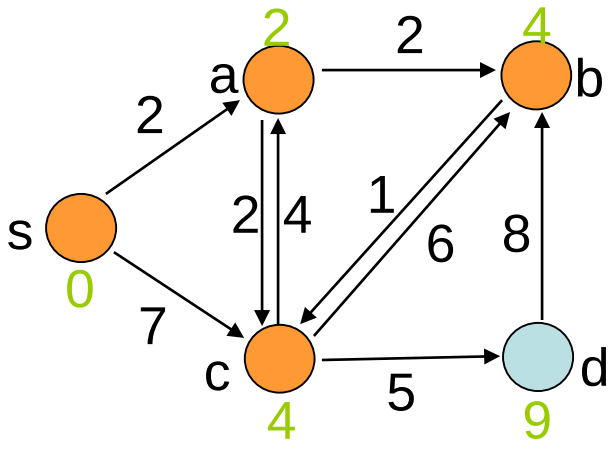


Q: d

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $u \leftarrow$ deleteMin(Q)
5. $S \leftarrow S \cup \{u\}$
6. **for each** vertex $v \in$ Adj[u] **do**
7. **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

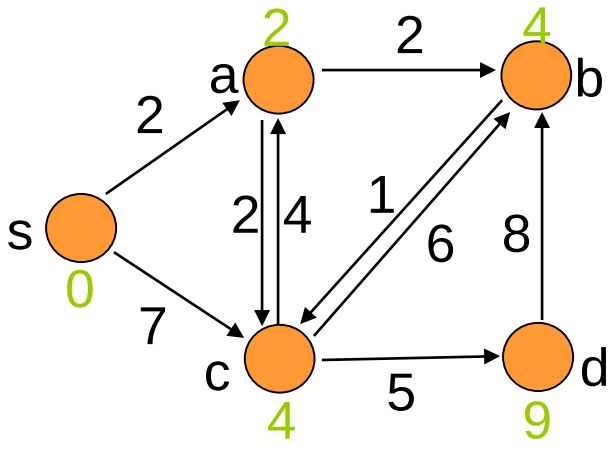


Q:

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

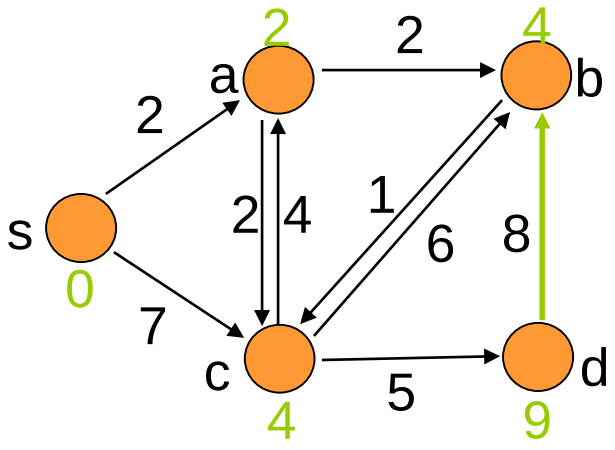


Q:

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex v∈V **do** d[v]←∞; π[v]←**nil**
2. d[s]←0; S←∅; Q←V
3. **while** Q≠∅ **do**
4. u←deleteMin(Q)
5. S←S∪{u}
6. **for each** vertex v∈Adj[u] **do**
7. **if** d[v]>d[u]+w(u,v) **then** DK(Q,v,d[u]+w(u,v)); π[v]←u

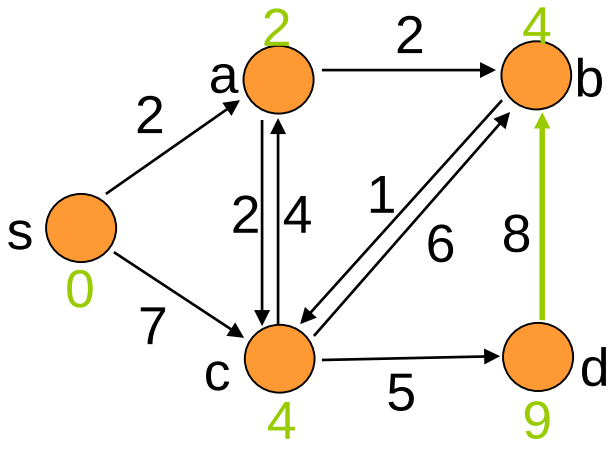


Q:

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$

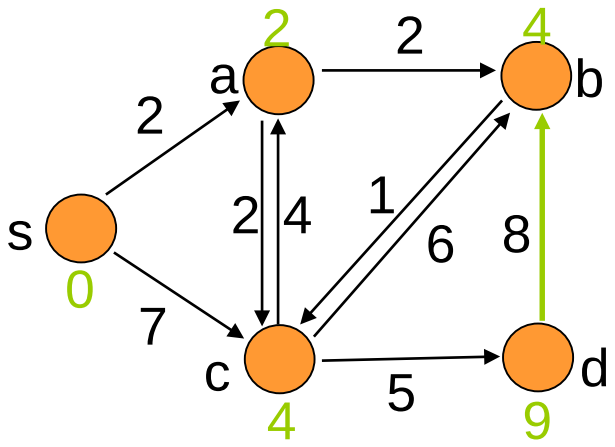


Q:

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



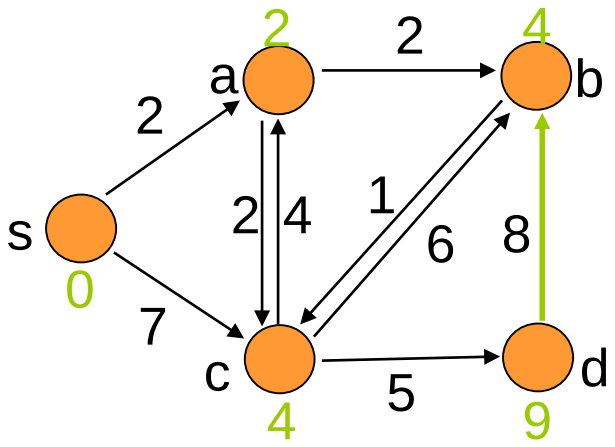
Laufzeit:

- $Q \leftarrow V$ (Aufbau des Heaps): $O(|V|)$

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



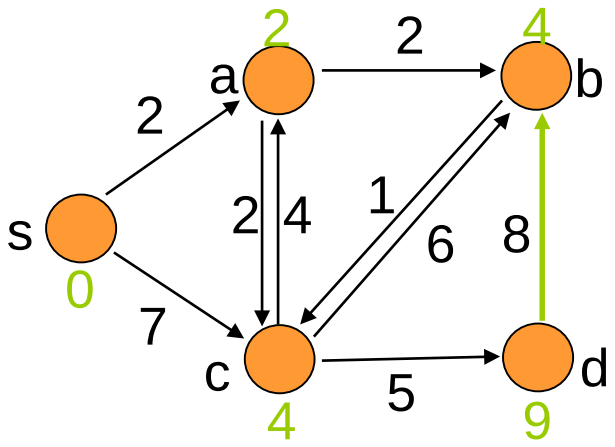
Laufzeit:

- deleteMin und Decrease-Key: O(log |V|)

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



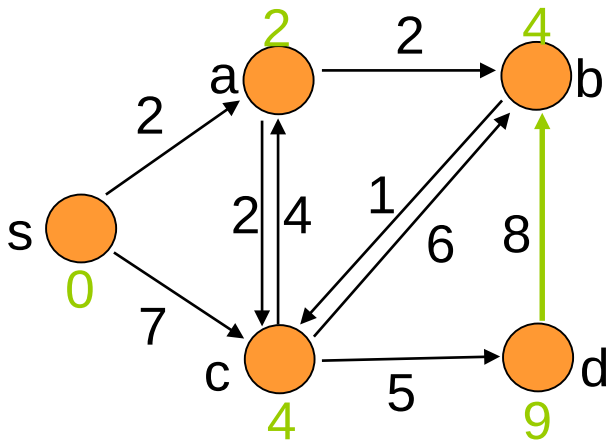
Laufzeit:

- deleteMin: |V|-mal, Decrease-Key: |E|-mal

Kürzeste Wege

Dijkstra(G,w,s)

1. **for each** vertex $v \in V$ **do** $d[v] \leftarrow \infty$; $\pi[v] \leftarrow$ **nil**
2. $d[s] \leftarrow 0$; $S \leftarrow \emptyset$; $Q \leftarrow V$
3. **while** $Q \neq \emptyset$ **do**
4. $\quad u \leftarrow$ deleteMin(Q)
5. $\quad S \leftarrow S \cup \{u\}$
6. $\quad$ **for each** vertex $v \in$ Adj[u] **do**
7. $\quad\quad$ **if** $d[v] > d[u]+w(u,v)$ **then** DK(Q,v,d[u]+w(u,v)); $\pi[v] \leftarrow u$



Laufzeit:

- insgesamt $O((|V|+|E|) \log |V|)$

Kürzeste Wege

Dijkstras Algorithmus

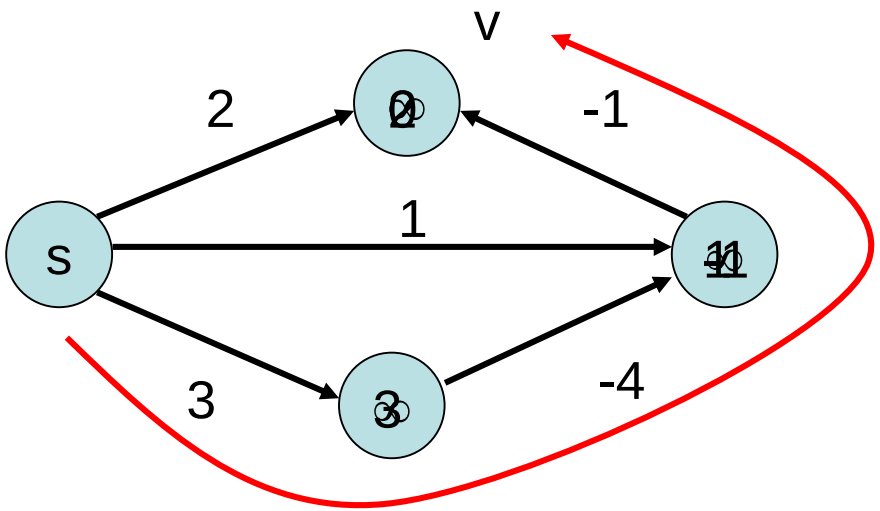
- Kürzeste Wege von einem Knoten und mit positiven Kantengewichten
- Laufzeit $O((|V|+|E|) \log |V|)$
- Bessere Laufzeit von $O(|V| \log |V| + |E|)$ möglich mit besseren Heaps (z.B. Fibonacci Heap)

Fragen:

- Was passiert bei negativen Kantengewichten?
- Wie kann man das kürzeste Wege Problem für alle Paare von Knoten effizient lösen?

Kürzeste Wege

Beispiel für Problem mit Dijkstra Algo:



Knoten v hat falschen Distanzwert!

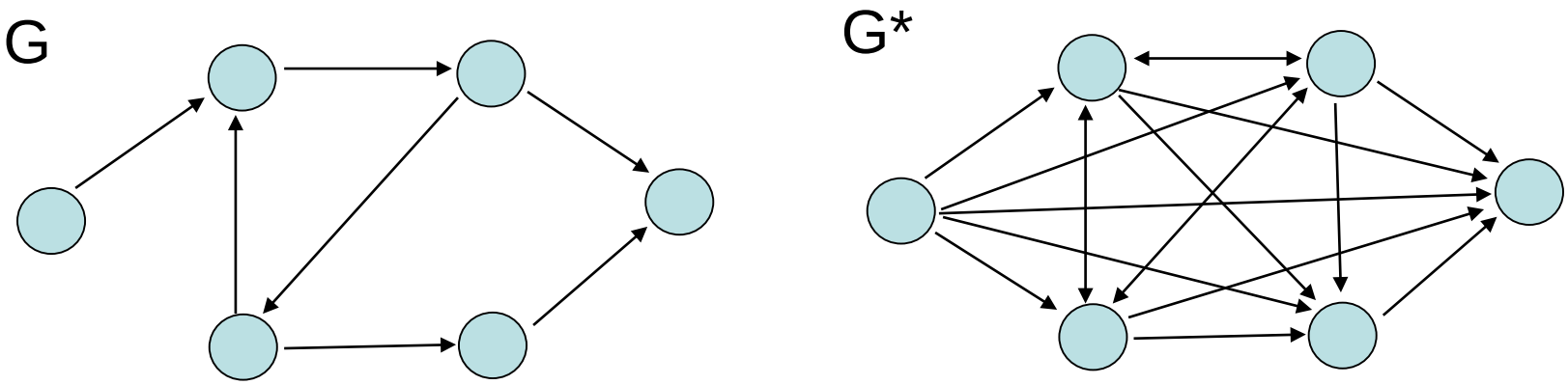
Alternative Verfahren bekannt (siehe Kapitel 17).

Changelog

12.06.16: Folien 28, 42-80 (DK eingesetzt)

Das transitive Hülle Problem:

- Gegeben sei ein gerichteter, ungewichteter Graph G=(V,E)
- Gesucht: Die transitive Hülle G*=(V,E*) von G, wobei E*={(u,v): es gibt Weg von u nach v in G}



Kürzeste Wege

Das transitive Hülle Problem:

- Gegeben sei ein gerichteter, ungewichteter Graph G=(V,E)
- Gesucht: Die transitive Hülle G*=(V,E*) von G, wobei E*={(u,v): es gibt Weg von u nach v in G}

Transitive Hülle:

- In $O(|V|^2+|V|\ |E|)$ Zeit mit Breiten- oder Tiefensuche von jedem Knoten
- Geht das auch schneller?

Graphen und Matrixmultiplikation:

- Sei A die $n \times n$-Adjazenzmatrix von Graph G mit Knotenmenge $\{1,\ldots,n\}$
- Was ist $A \cdot A$?

Behauptung 15.2:

Sei $Z = A \cdot A$. Dann gilt $z_{ij} > 0$, g.d.w. es in G einen Pfad der Länge 2 von Knoten i zu Knoten j gibt.

Behauptung 15.2:

Sei $Z' = A \cdot A + A$. Dann gilt, dass $z'_{ij} > 0$, g.d.w. es einen Weg der Länge 1 oder 2 von Knoten i zu Knoten j gibt.

Konstruiere Matrix B mit:

- $b_{ij} = 1 \Leftrightarrow z'_{ij} > 0$

Behauptung 15.3:

Matrix B hat einen Weg von Knoten i nach j, g.d.w. Matrix A einen solchen Weg hat.

Behauptung 15.4:

Sei P ein Weg der Länge k>1 in G. Dann hat P maximal Länge (2/3)k in dem von Matrix B beschriebenen Graph G‘.

Konsquenz aus Beh. 15.3 und 15.4:

- Wenn wir die Berechnung von B $\log_{3/2} n$ mal iterieren, haben wir die transitive Hülle berechnet

Kürzeste Wege

TransitiveHülle(A)

1. **for** i$\leftarrow$1 **to** $\log_{3/2}$n **do**
2. Z‘ $\leftarrow$ A A+A
3. **for** i$\leftarrow$1 **to** n **do**
4. **for** j$\leftarrow$1 **to** n **do**
5. **if** z'_{ij} >0 **then** $b_{ij} \leftarrow 1$ **else** $b_{ij} \leftarrow 0$
6. A$\leftarrow$B
7. **return** A

Satz 15.5:

Der Algorithmus TransitiveHülle berechnet die transitive Hülle eines Graphen G in $O(M(n) \log n)$ Zeit, wobei $M(n)$ die Laufzeit zur Matrixmultiplikation bezeichnet.