Grundlagen: Algorithmen und Datenstrukturen

Prof. Dr. Hanjo Täubig

Lehrstuhl für Effiziente Algorithmen
(Prof. Dr. Ernst W. Mayr)
Institut für Informatik
Technische Universität München

Sommersemester 2010

Übersicht

Starke Zhk. und DFS

Prinzip: betrachte Kante $e = (v, w)$

- Kante zu schon bekanntem Knoten w in offener ZHK (Rückwärts-/Querkante):
  falls v und w momentan noch in unterschiedlichen ZHKs liegen, müssen diese zusammen mit allen ZHKs dazwischen zu einer einzigen ZHK verschmolzen werden

  bei Vorwärtskanten ist nichts zu tun
- Kante zu Knoten w in geschlossener ZHK (Querkante):
  von w gibt es keinen Weg zu v, sonst wäre die ZHK von w noch nicht geschlossen, also bleiben die ZHKs unverändert
- Kante zu unbekanntem Knoten w (Baumkante):
  neue ZHK für w

Wenn Knoten keine ausgehenden Kanten mehr hat:

- Knoten fertig
- wenn Knoten Repräsentanten seiner ZHK ist, dann ZHK schließen

Starke Zhk. und DFS / Variante 2

2. Variante:

- Verwaltung der unfertigen Knoten in Stack oNodes
  (in Reihenfolge steigender dfsNum)
- Verwaltung der Repräsentanten der offenen ZHKs in Stack oReps

Starke Zhk. und DFS / Variante 2

- void init() {
  ```
  component = new int[n];
  oReps = ⟨ ⟩;
  oNodes = ⟨ ⟩;
  dfsCount = 1;
  }
  ```

- void root(Node w) / traverseTreeEdge(Node v, Node w) {
  ```
  oReps.push(w);      // Repräsentant einer neuen ZHK
  oNodes.push(w);     // neuer offener Knoten
  dfsNum[w] = dfsCount;   dfsCount++;
  }
  ```

Starke Zhk. und DFS / Variante 2

- void traverseNonTreeEdge(Node v, Node w) {

```
    if (w ∈ oNodes)       // verschmelze ZHKs
        while (dfsNum[w] < dfsNum[oReps.top()])
            oReps.pop();
}
```

- void backtrack(Node u, Node v) {

```
    if (v == oReps.top()) {   // v Repräsentant?
        oReps.pop();   // ja: entferne v
        do {   // und offene Knoten bis v
            w = oNodes.pop();
            component[w] = v;
        } while (w!=v);
    }
}
```

Starke Zhk. und DFS / Variante 2

Zeit: $\mathcal{O}(n+m)$

Begründung:

- init, root: $\mathcal{O}(1)$
- traverseTreeEdge: $(n-1) \times \mathcal{O}(1)$
- backtrack, traverseNonTreeEdge:
  da jeder Knoten höchstens einmal in oReps und oNodes landet, insgesamt $\mathcal{O}(n+m)$
- DFS-Gerüst: $\mathcal{O}(n+m)$
- gesamt: $\mathcal{O}(n+m)$

DFS / Anwendung

Weitere Anwendung für DFS: Weg durchs Labyrinth

- Repräsentation von Kreuzungspunkten und Enden von Sackgassen als Knoten
- Repräsentation von direkt verbundenen Punkten als Kanten

- Tiefensuche (DFS) entspricht der Erkundung des Labyrinths, wobei man sich den bisherigen Weg vom Eingang aus merkt (und welche Abzweigungen man schon erkundet hat)

- Suche solange, bis Ausgang gefunden

Kürzeste Wege

Zentrale Frage: Wie kommt man am schnellsten von A nach B?

Kürzeste Wege

Zentrale Frage: Wie kommt man am schnellsten von A nach B?

Fälle:

- Kantenkosten 1
- DAG, beliebige Kantenkosten
- beliebiger Graph, positive Kantenkosten
- beliebiger Graph, beliebige Kantenkosten

Kürzeste-Wege-Problem

gegeben:

- gerichteter Graph $G = (V, E)$
- Kantenkosten $c : \; E \mapsto \mathbb{R}$

2 Varianten:

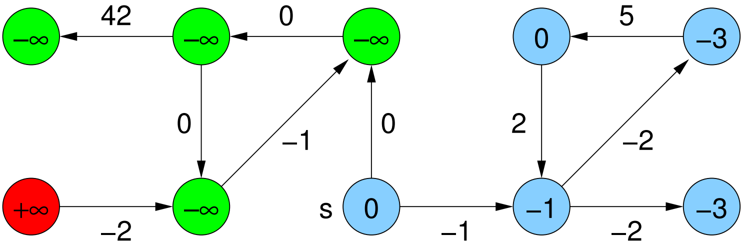
- SSSP (single source shortest paths):

  kürzeste Wege von einer Quelle zu allen anderen Knoten

- APSP (all pairs shorstest paths):

  kürzeste Wege zwischen allen Paaren

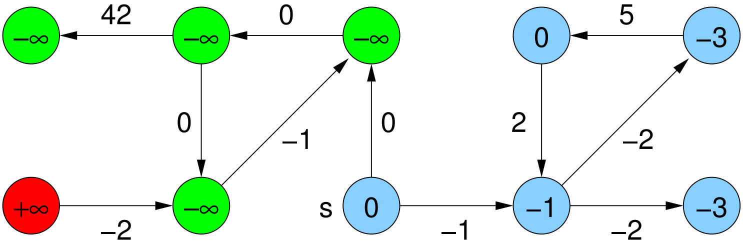
Distanzen



$\mu(s,v)$: Distanz von s nach v

$$\mu(s,v) = \begin{cases} +\infty & \text{kein Weg von } s \text{ nach } v \\ -\infty & \text{Weg beliebig kleiner Kosten von } s \text{ nach } v \\ \min\{c(p) : p \text{ ist Weg von } s \text{ nach } v\} \end{cases}$$

Distanzen



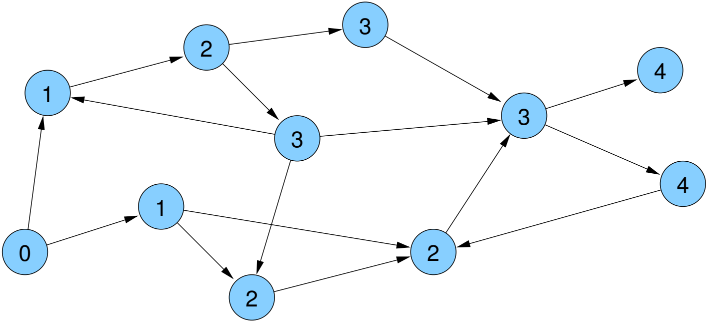
Wann sind die Kosten $-\infty$?

wenn es einen Kreis mit negativer Gewichtssumme gibt
(hinreichende und notwendige Bedingung)

Kürzeste Wege

Graph mit Kantenkosten 1:

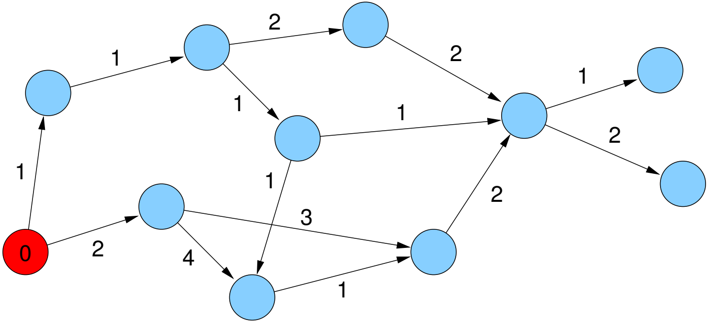
⇒ Breitensuche (BFS)



Kürzeste Wege

Beliebige Kantengewichte in DAGs

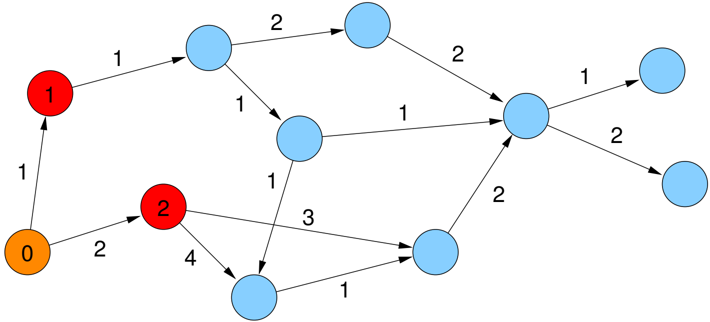
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

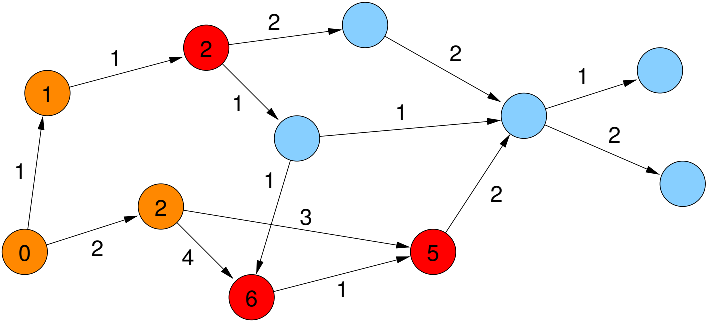
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

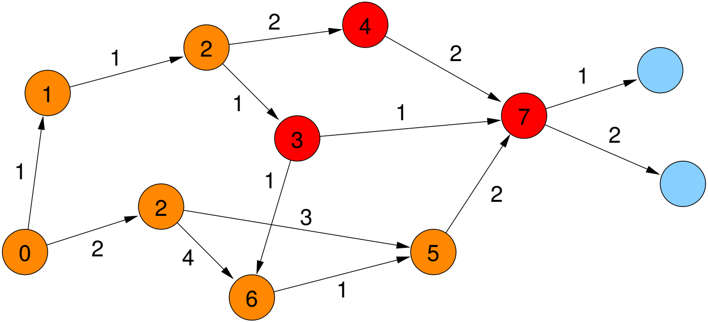
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

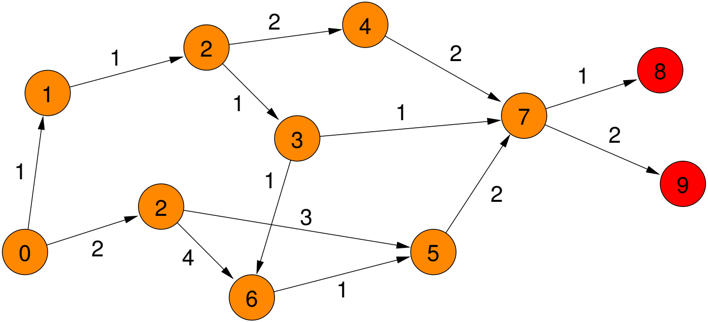
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

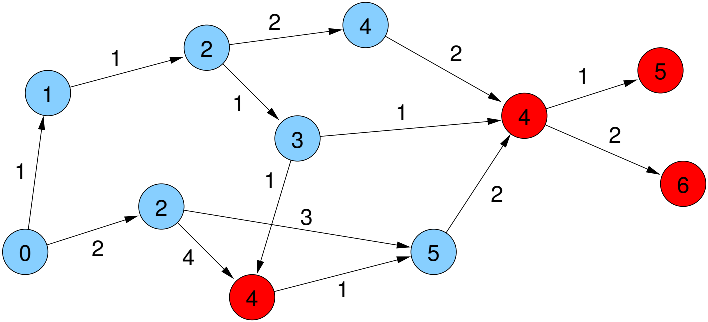
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege

Beliebige Kantengewichte in DAGs

Einfache Breitensuche funktioniert nicht.

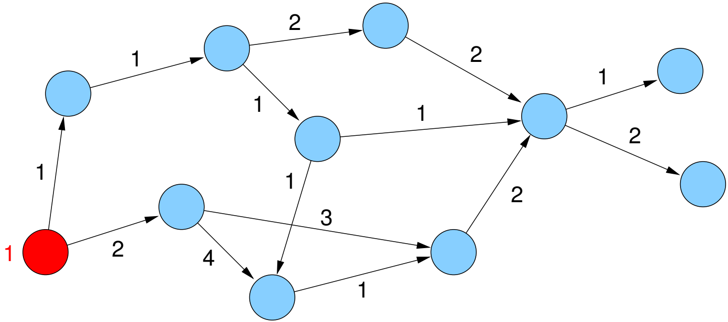


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

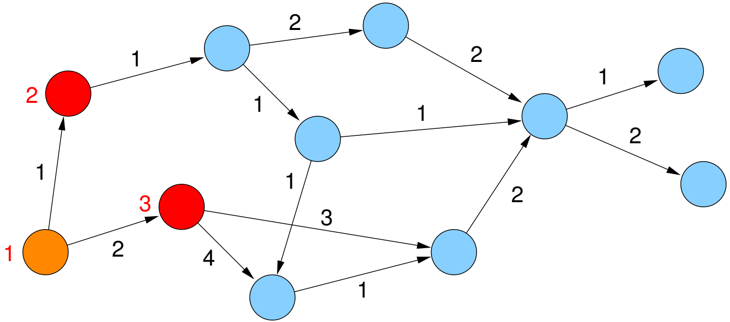


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

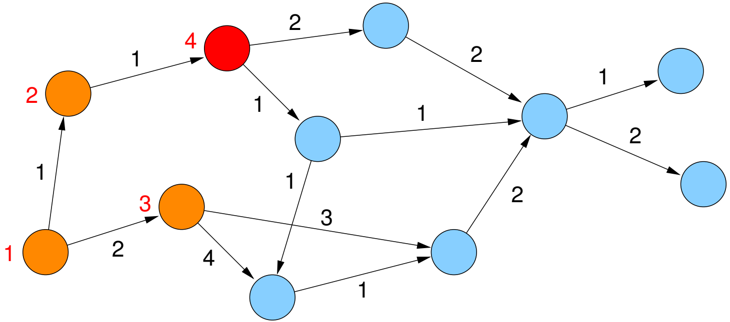


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

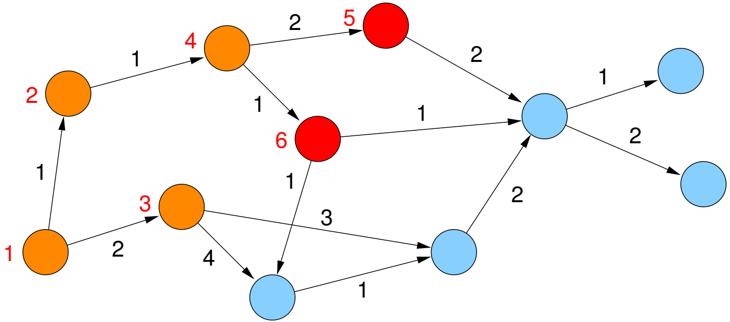


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

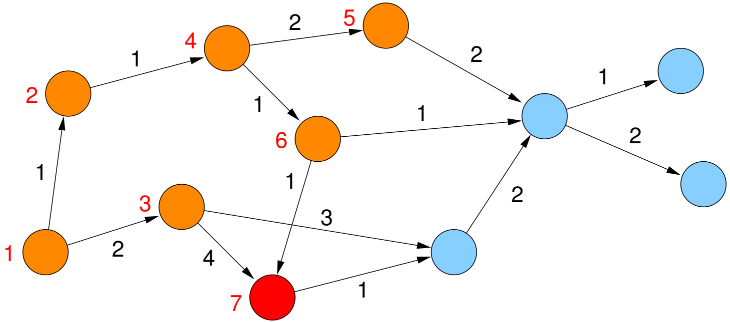


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

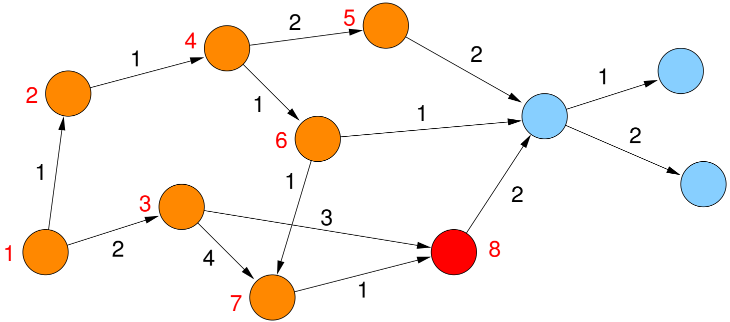


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

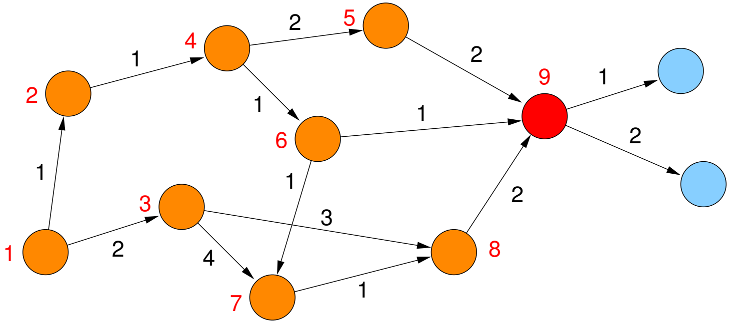


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

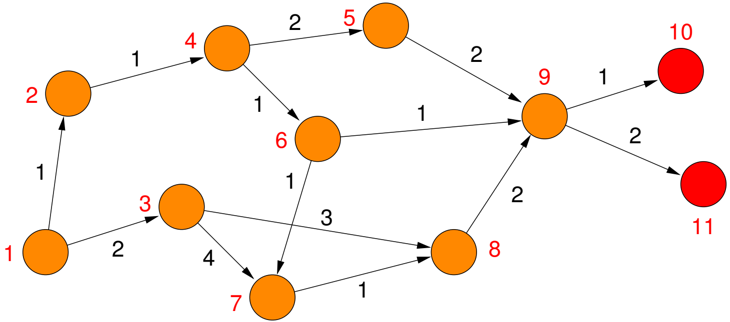


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung

(für alle Kanten e=(v,w) gilt topoNum(v) < topoNum(w))

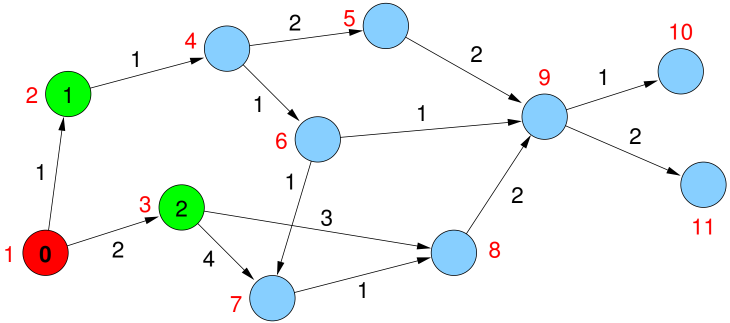


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

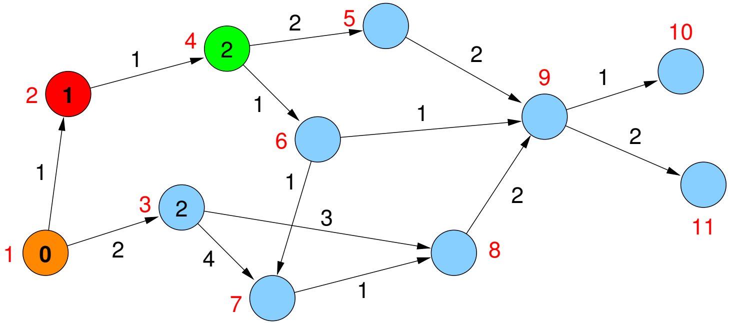


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

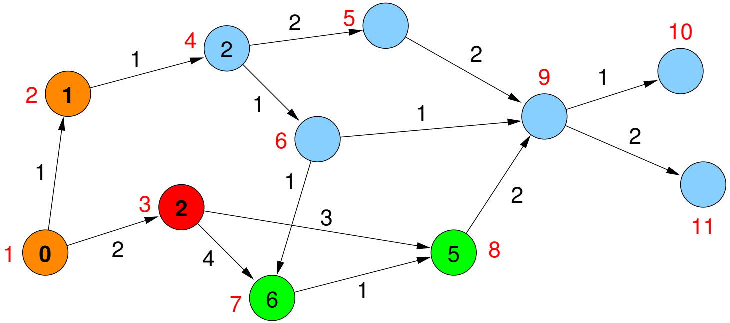


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

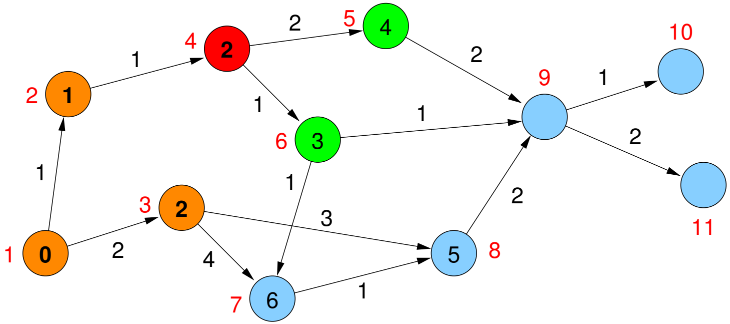


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

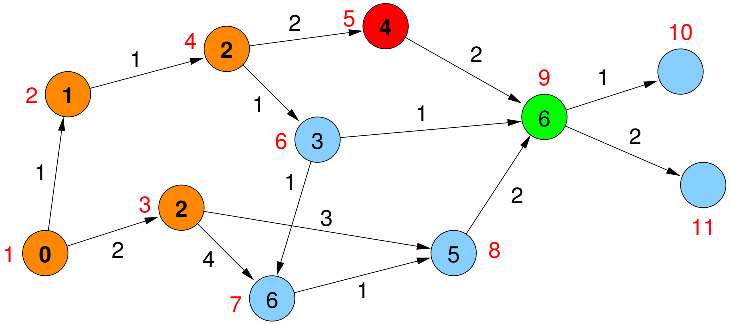


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

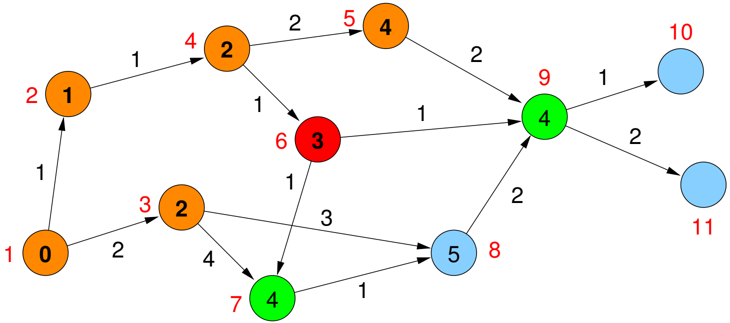


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

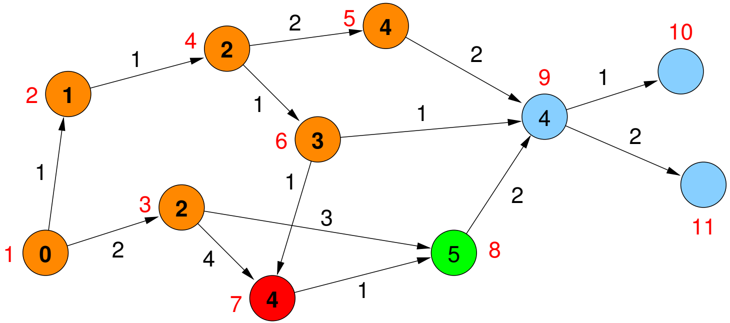


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

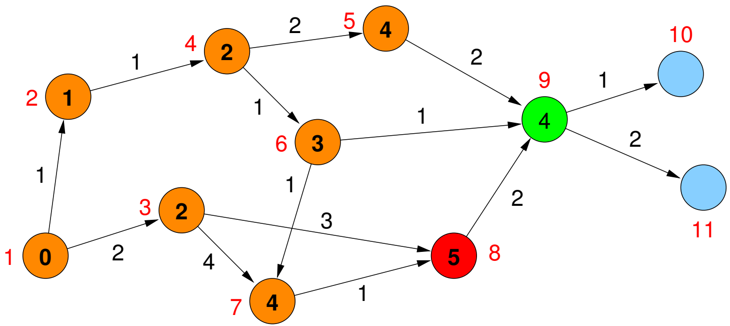


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

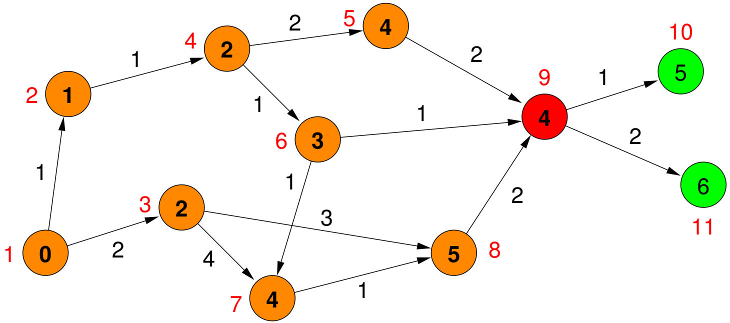


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

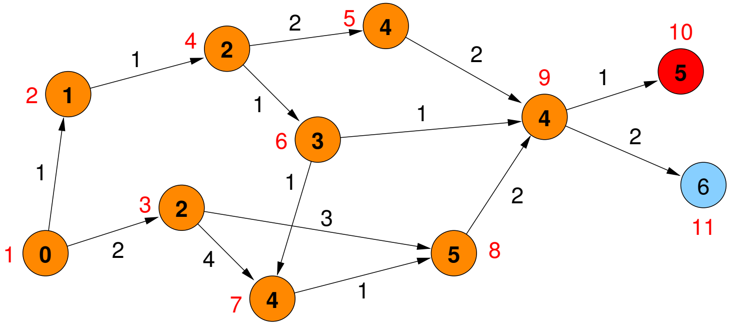


Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte



Kürzeste Wege

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte