

Script generated by TTT

Title: Täubig: GAD (12.07.2012)

Date: Thu Jul 12 12:31:12 CEST 2012

Duration: 45:23 min

Pages: 37

Graphen Kürzeste Wege

Übersicht

- 9 Graphen
 - Netzwerke und Graphen
 - Graphrepräsentation
 - Graphtraversierung
 - Kürzeste Wege**
 - Minimale Spannbäume

H. Täubig (TUM) GAD SS'12 488 / 638

Graphen Kürzeste Wege

Kürzeste Wege

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

H. Täubig (TUM) GAD SS'12 489 / 638

Graphen Kürzeste Wege

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

H. Täubig (TUM) GAD SS'12 490 / 638

Kürzeste-Wege-Problem

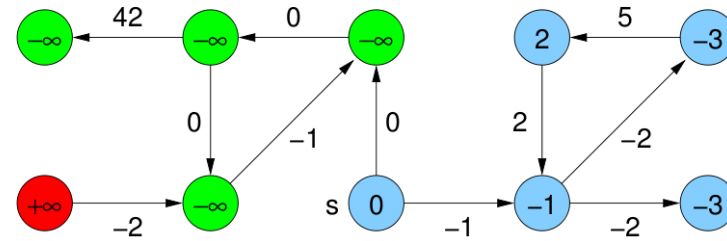
gegeben:

- 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 shortest paths):
kürzeste Wege zwischen allen Paaren

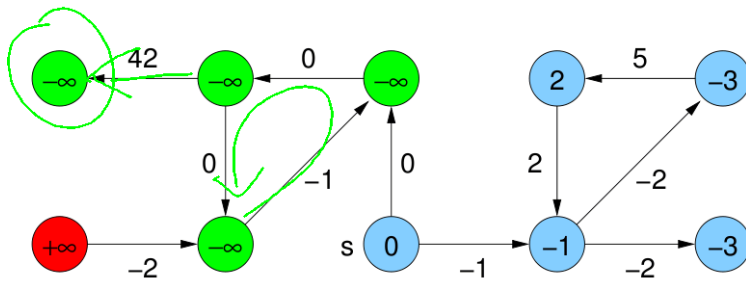
Distanzen



$\mu(s, v)$: 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



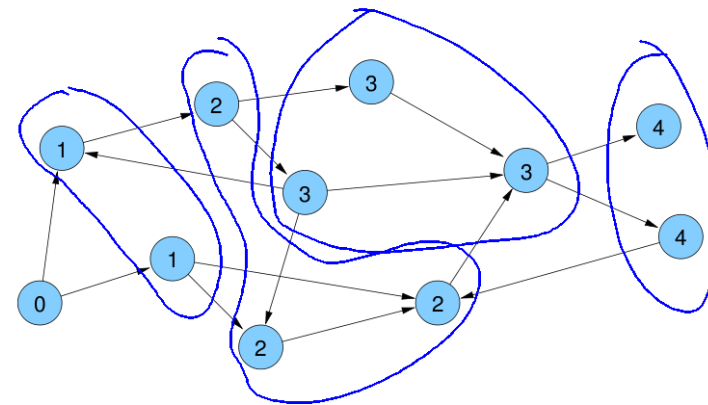
Wann sind die Kosten $-\infty$?

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

Kürzeste Wege bei uniformen Kantenkosten

Graph mit Kantenkosten 1:

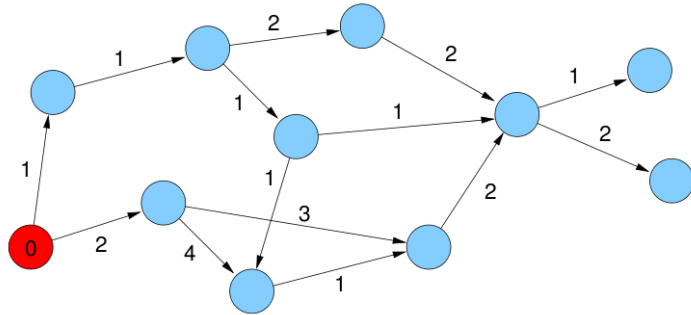
\Rightarrow Breitensuche (BFS)



Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

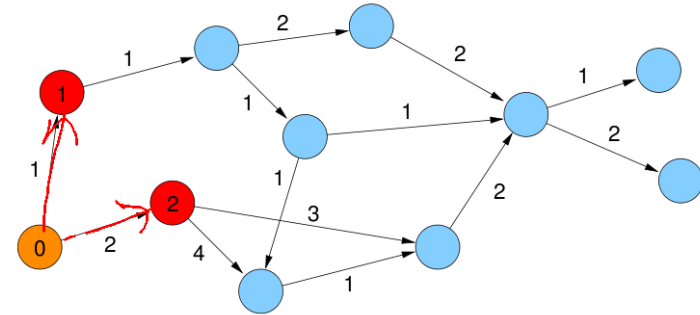
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

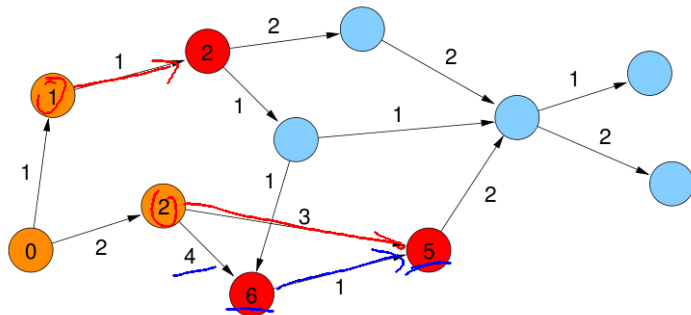
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

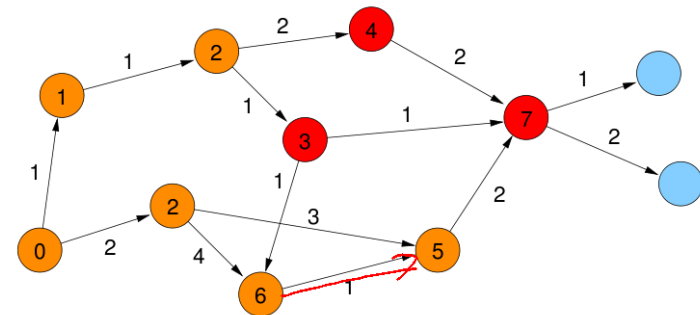
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

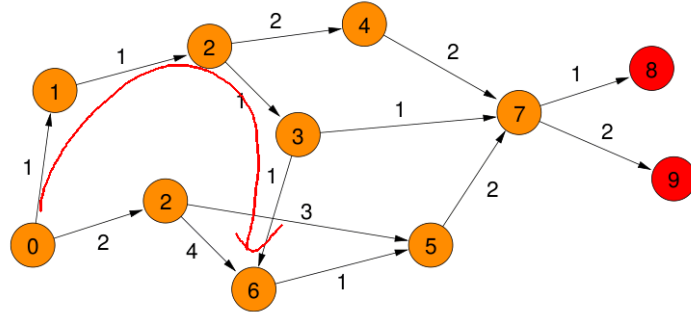
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

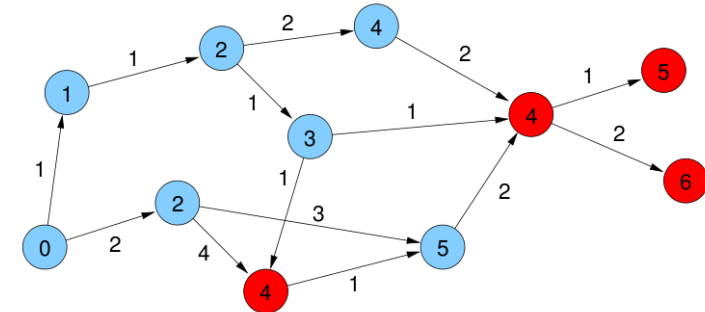
Einfache Breitensuche funktioniert nicht.



Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

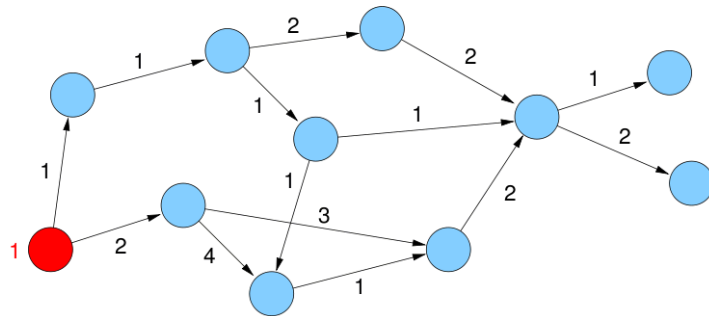
Einfache Breitensuche funktioniert nicht.



Topologische Sortierung in DAGs

Beliebige Kantengewichte in DAGs

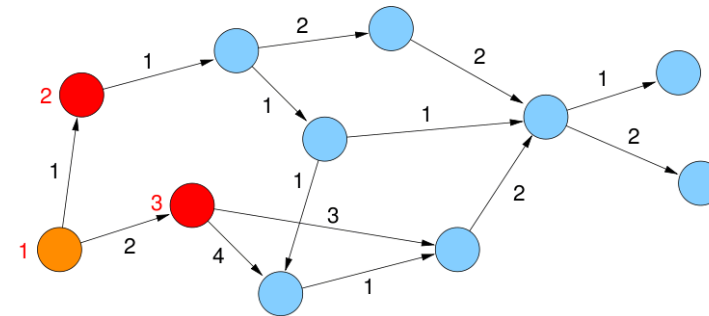
Strategie: in DAGs gibt es **topologische Sortierung**
 (für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)



Topologische Sortierung in DAGs

Beliebige Kantengewichte in DAGs

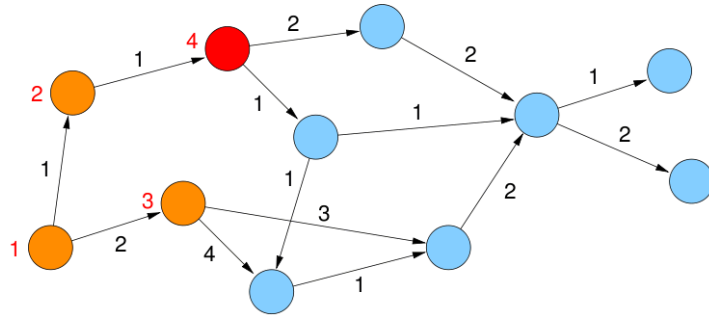
Strategie: in DAGs gibt es topologische Sortierung
 (für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)



Topologische Sortierung in DAGs

Beliebige Kantengewichte in DAGs

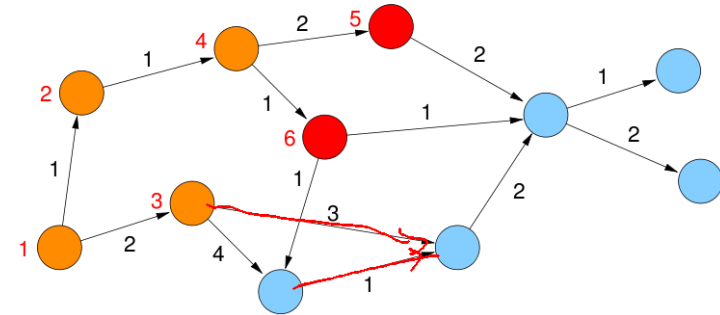
Strategie: in DAGs gibt es topologische Sortierung
 (für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)



Topologische Sortierung in DAGs

Beliebige Kantengewichte in DAGs

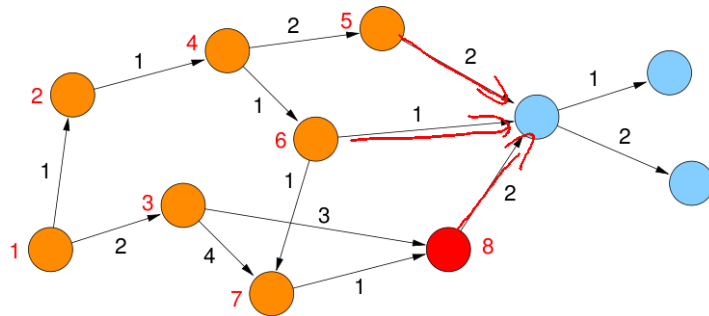
Strategie: in DAGs gibt es topologische Sortierung
 (für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)



Topologische Sortierung in DAGs

Beliebige Kantengewichte in DAGs

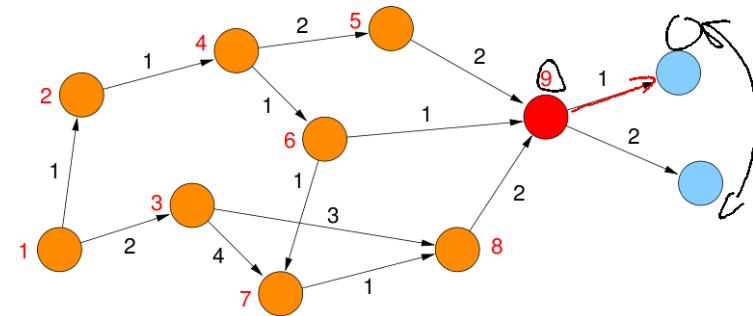
Strategie: in DAGs gibt es topologische Sortierung
 (für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)



Topologische Sortierung in DAGs

Beliebige Kantengewichte in DAGs

Strategie: in DAGs gibt es topologische Sortierung
 (für alle Kanten $e=(v,w)$ gilt $\text{topoNum}(v) < \text{topoNum}(w)$)

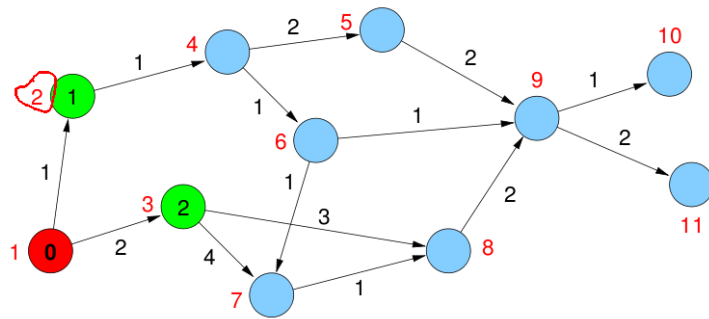


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

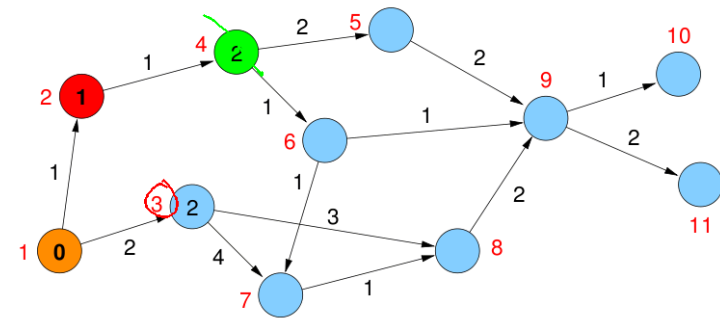


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

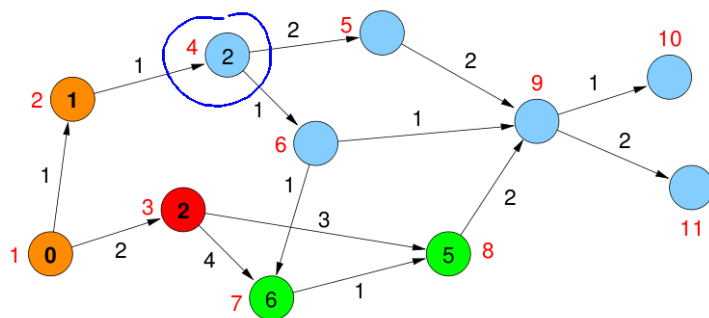


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

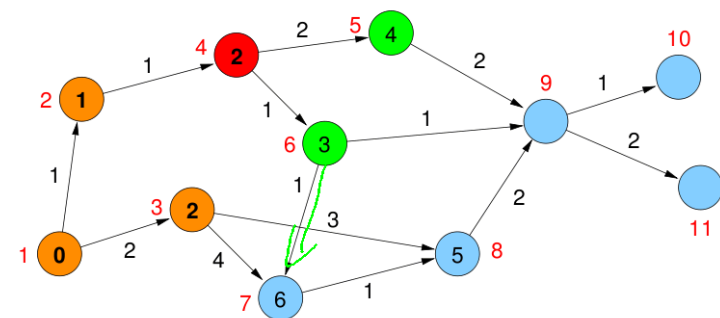


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

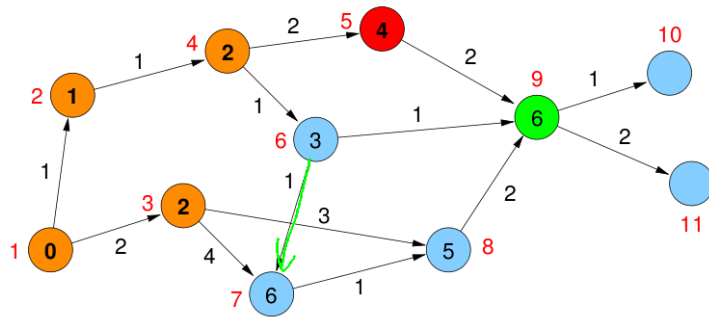


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

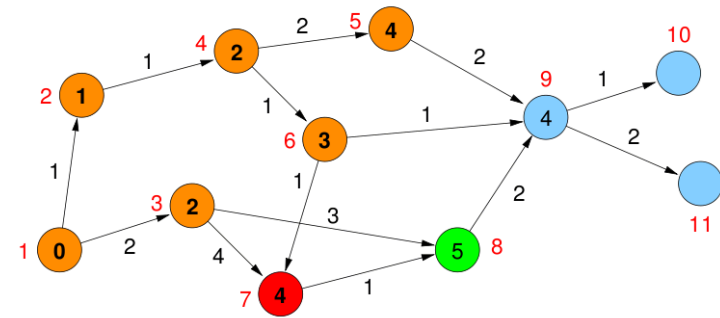


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

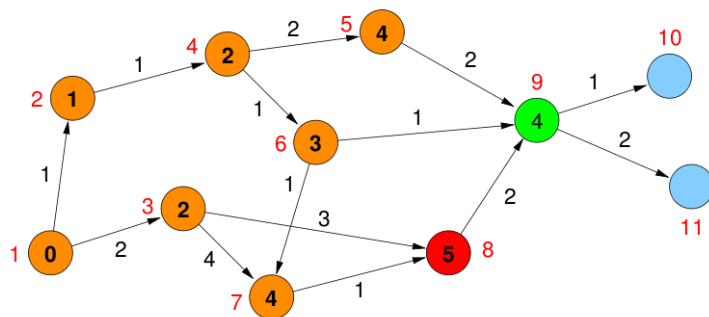


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte

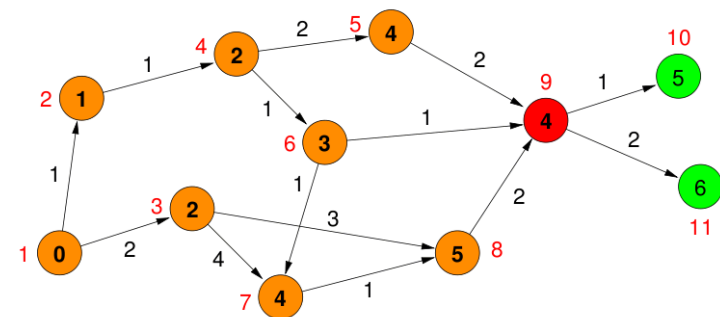


Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Strategie:

- betrachte Knoten in Reihenfolge der topologischen Sortierung
- aktualisiere Distanzwerte



Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs



Topologische Sortierung – warum funktioniert das?

- betrachte einen kürzesten Weg von s nach v
- der ganze Pfad beachtet die topologische Sortierung
- d.h., die Distanzen werden in der Reihenfolge der Knoten vom Anfang des Pfades zum Ende hin betrachtet
- damit ergibt sich für v der richtige Distanzwert
- ein Knoten x kann auch nie einen Wert erhalten, der echt kleiner als seine Distanz zu s ist
- die Kantenfolge von s zu x , die jeweils zu den Distanzwerten an den Knoten geführt hat, wäre dann ein kürzerer Pfad (Widerspruch)

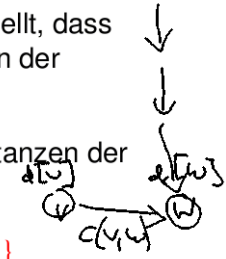
Kürzeste Wege in DAGs

Beliebige Kantengewichte in DAGs

Allgemeine Strategie:

- Anfang: setze $d(s) = 0$ und für alle anderen Knoten v setze $d(v) = \infty$
- 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 v aktualisiere die Distanzen der Knoten w mit $(v, w) \in E$, d.h. setze

$$d(w) = \min\{d(w), d(v) + c(v, w)\}$$



Kürzeste Wege in DAGs

Topologische Sortierung

- verwende FIFO-Queue q
- verwalte für jeden Knoten einen Zähler für die noch nicht markierten eingehenden Kanten
- initialisiere q mit allen Knoten, die keine eingehende Kante haben (Quellen)
- nimm nächsten Knoten v aus q und markiere alle $(v, w) \in E$, d.h. dekrementiere Zähler für w
- falls der Zähler von w dabei Null wird, füge w in q ein
- wiederhole das, bis q leer wird

Kürzeste Wege in DAGs

Topologische Sortierung

Korrektheit

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

Laufzeit

- für die Anfangswerte der Zähler muss der Graph einmal traversiert werden $O(n + m)$
 - danach wird jede Kante genau einmal betrachtet
- ⇒ gesamt: $O(n + m)$

Test auf DAG-Eigenschaft

- topologische Sortierung erfasst genau dann **alle** Knoten, wenn der Graph ein **DAG** ist
- bei gerichteten Kreisen erhalten diese Knoten keine Nummer

Kürzeste Wege in DAGs

DAG-Strategie

- 1 Topologische Sortierung der Knoten
Laufzeit $O(n + m)$
- 2 Aktualisierung der Distanzengemäß der topologischen Sortierung
Laufzeit $O(n + m)$

Gesamtlaufzeit: $O(n + m)$

Beliebige Graphen mit nicht-negativen Gewichten

Gegeben:

- beliebiger Graph
(gerichtet oder ungerichtet, muss diesmal kein DAG sein)
- mit nicht-negativen Kantengewichten
⇒ keine Knoten mit Distanz $-\infty$

Problem:

- besuche Knoten eines kürzesten Weges in der richtigen Reihenfolge
- wie bei Breitensuche, jedoch diesmal auch mit Distanzen $\neq 1$

Lösung:

- besuche Knoten in der Reihenfolge der kürzesten Distanz zum Startknoten s

Kürzeste Pfade: SSSP / Dijkstra

Dijkstra-Algorithmus

Input : $G = (V, E)$, $c : E \rightarrow \mathbb{R}$, $s \in V$

Output : Distanzen $d(s, v)$ zu allen $v \in V$

$P = \emptyset$; $T = V$;

$d(s, v) = \infty$ for all $v \in V \setminus s$;

$d(s, s) = 0$; $pred(s) = 0$;

while $P \neq V$ **do**

$v = \operatorname{argmin}_{v \in T} \{d(s, v)\}$;

$P = P \cup v$; $T = T \setminus v$;

forall the $(v, w) \in E$ **do**

if $d(s, w) > d(s, v) + c(v, w)$ **then**

$d(s, w) = d(s, v) + c(v, w)$;

$pred(w) = v$;

Dijkstra-Algorithmus für SSSP

Input : $G = (V, E)$, $c : E \rightarrow \mathbb{R}_{\geq 0}$, $s \in V$

Output : Distanzen $d[v]$ von s zu allen $v \in V$

$d[v] = \infty$ for all $v \in V \setminus s$;

$d[s] = 0$; $pred[s] = \perp$;

$pq = \langle \rangle$; $pq.insert(s, 0)$;

while $\neg pq.empty()$ **do**

$v = pq.deleteMin()$;

forall the $(v, w) \in E$ **do**

$newDist = d[v] + c(v, w)$;

if $newDist < d[w]$ **then**

$pred[w] = v$;

if $d[w] == \infty$ **then** $pq.insert(w, newDist)$;

else $pq.decreaseKey(w, newDist)$;

$d[w] = newDist$;

Dijkstra-Algorithmus für SSSP

Input : $G = (V, E)$, $c : E \rightarrow \mathbb{R}_{\geq 0}$, $s \in V$

Output : Distanzen $d[v]$ von s zu allen $v \in V$

$d[v] = \infty$ for all $v \in V \setminus s$;

$d[s] = 0$; $pred[s] = \perp$;

$pq = \langle \rangle$; $pq.insert(s, 0)$;

while $\neg pq.empty()$ **do**

$v = pq.deleteMin()$;

forall the $(v, w) \in E$ **do**

$newDist = d[v] + c(v, w)$;

if $newDist < d[w]$ **then**

$pred[w] = v$;

if $d[w] == \infty$ **then** $pq.insert(w, newDist)$;

else $pq.decreaseKey(w, newDist)$;

$d[w] = newDist$;

Dijkstra-Algorithmus

- setze Startwert $d(s, s) = 0$ und zunächst $d(s, v) = \infty$
- verwende **Prioritätswarteschlange**, um die Knoten zusammen mit ihren aktuellen Distanzen zu speichern
- am Anfang nur Startknoten (mit Distanz 0) in Priority Queue
- dann immer nächsten Knoten v (mit kleinster Distanz) entnehmen, endgültige Distanz dieses Knotens v steht nun fest
- betrachte alle Nachbarn von v , füge sie ggf. in die PQ ein bzw. aktualisiere deren Priorität in der PQ