



Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

Graphalgorithmen

Single Source Shortest Path (SSSP)

- Startknoten s
- Aufgabe: Berechne kürzeste Wege von s zu allen anderen Knoten

All Pairs Shortest Path (APSP)

- Aufgabe: Berechne kürzeste Wege zwischen allen Knotenpaaren

SSSP in ungerichteten Graphen (Breitensuche)

- Graph in Adjazenzlistendarstellung
- Startknoten s
- Nutze Kanten von G , um alle Knoten zu finden, die von s aus erreichbar sind
- Finde kürzeste Distanz (Anzahl Kanten) zu jedem anderen Knoten

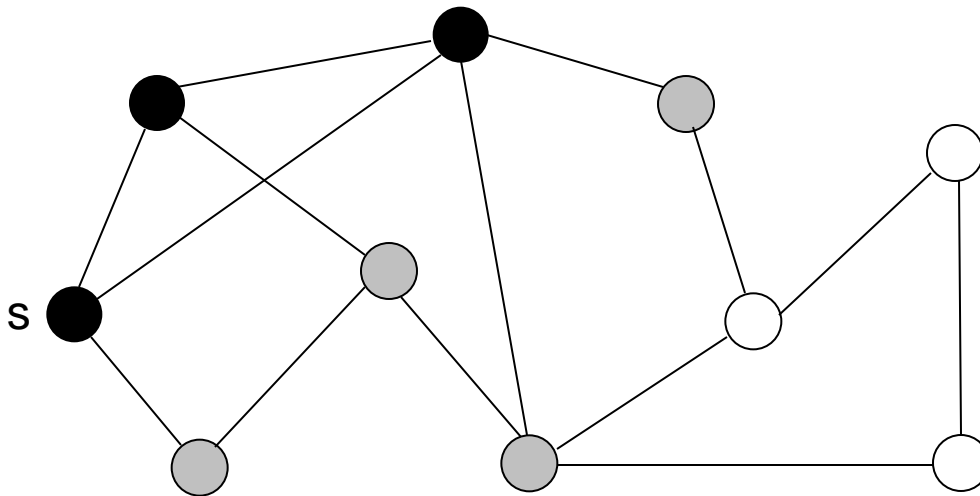
Graphalgorithmen

Invariante (Breitensuche)

- Knoten haben 3 Farben: weiß, grau und schwarz
- Zu Beginn: Alle Knoten sind weiß
- Ein nicht-weißer Knoten heißt „entdeckt“
- Unterscheidung grau-schwarz dient zur Steuerung des Algorithmus
- Wenn $(u,v) \in E$ ist und u ist schwarz, dann sind seine adjazenten Knoten grau oder schwarz
- Graue Knoten können adjazente weiße Knoten haben

Graphalgorithmen

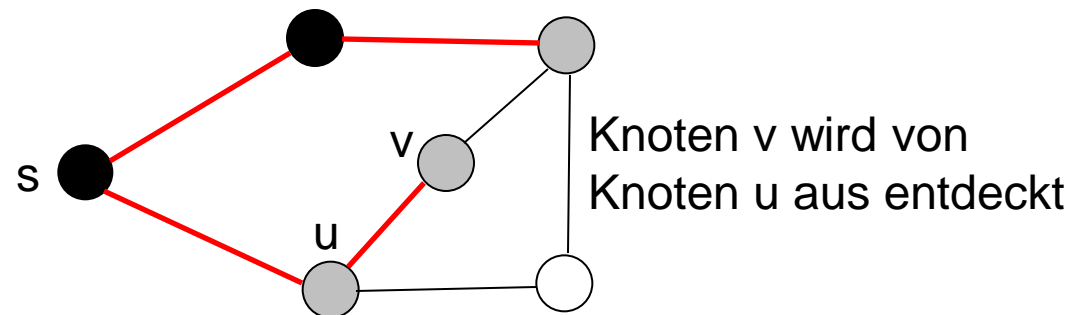
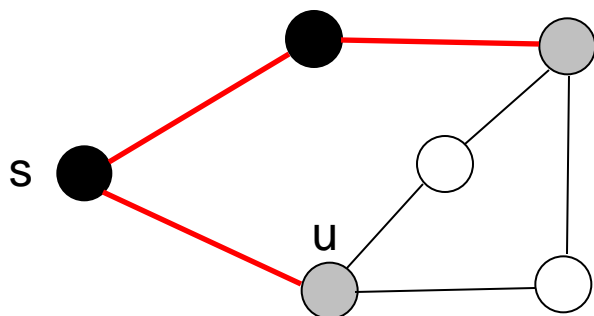
Beispiel (mögl. Zustand bei einer Breitensuche)



Graphalgorithmen

Breitensuche

- Baut Breitensuche Baum (BFS Baum)
- Zu Beginn enthält der Baum nur die Wurzel, nämlich s
- Wenn weißer Knoten beim Durchsuchen der Adjazenzliste eines entdeckten Knotens entdeckt wird, dann werden v und (u,v) dem Baum hinzugefügt
- u ist dann Vaterknoten von v



Graphalgorithmen

Datenstruktur Schlange

- Operationen: head, enqueue, dequeue
- head: Gibt Referenz auf das erste in der Schlange gespeicherte Element zurück
- enqueue: Fügt neues Element ans Ende der Schlage
- dequeue: Entfernt Kopf der Schlange

Wir verwenden

- Doppelt verkettete Liste
- Zusätzlich halten wir Zeiger auf das letzte Element aufrecht
- Alle Operationen in $O(1)$ Zeit

Graphalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** $Q \neq \emptyset$ **do**
3. $u \leftarrow \text{head}[Q]$
4. **for each** $v \in \text{Adj}[u]$ **do**
5. **if** $\text{color}[v] = \text{weiß}$ **then**
6. $\text{color}[v] \leftarrow \text{grau}$
7. $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8. $\text{enqueue}(Q, v)$
9. $\text{dequeue}(Q)$
10. $\text{color}[u] \leftarrow \text{schwarz}$

$d[u]$: Abstand zu s (zu Beginn ∞)

$\pi[u]$: Vaterknoten von u (zu Beginn nil)

Graphalgorithmen

BFS(G,s)

1. „initialisiere BFS“
2. **while** $Q \neq \emptyset$ **do**
3. $u \leftarrow \text{head}[Q]$
4. **for each** $v \in \text{Adj}[u]$ **do**
5. **if** $\text{color}[v] = \text{weiß}$ **then**
6. $\text{color}[v] \leftarrow \text{grau}$
7. $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8. $\text{enqueue}(Q, v)$
9. $\text{dequeue}(Q)$
10. $\text{color}[u] \leftarrow \text{schwarz}$

Für jeden Knoten u:

- $\text{color}[u] = \text{weiß}$
- $d[u] = \infty$
- $\pi[u] = \text{nil}$

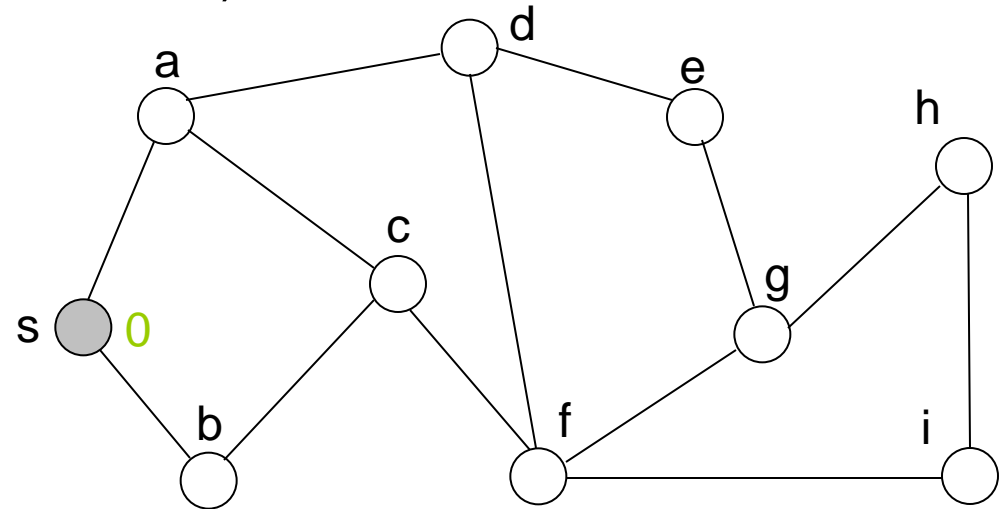
Für Knoten s:

- $\text{color}[s] = \text{grau}$
- $d[s] = 0$
- $\pi[s] = \text{nil}$
- s wird in Schlange Q eingefügt

Graphalgorithmen (siehe Vollversion)

BFS(G,s)

1. „initialisiere BFS“
2. **while** $Q \neq \emptyset$ **do**
3. $u \leftarrow \text{head}[Q]$
4. **for each** $v \in \text{Adj}[u]$ **do**
5. **if** $\text{color}[v] = \text{weiß}$ **then**
6. $\text{color}[v] \leftarrow \text{grau}$
7. $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8. $\text{enqueue}(Q, v)$
9. $\text{dequeue}(Q)$
10. $\text{color}[u] \leftarrow \text{schwarz}$

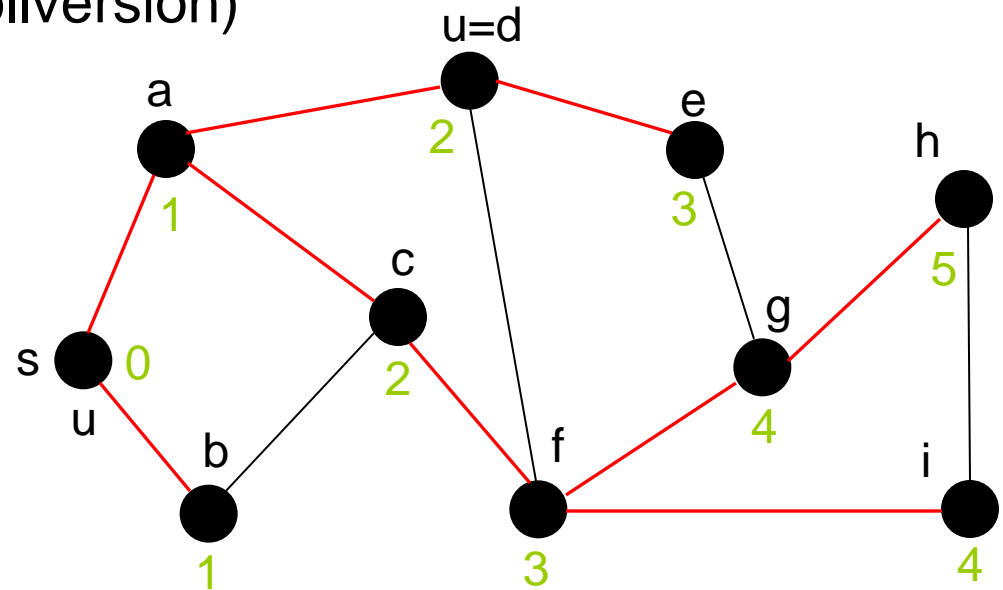


Q: s

Graphalgorithmen (siehe Vollversion)

BFS(G,s)

1. „initialisiere BFS“
2. **while** $Q \neq \emptyset$ **do**
3. $u \leftarrow \text{head}[Q]$
4. **for each** $v \in \text{Adj}[u]$ **do**
5. **if** $\text{color}[v] = \text{weiß}$ **then**
6. $\text{color}[v] \leftarrow \text{grau}$
7. $d[v] \leftarrow d[u] + 1; \pi[v] \leftarrow u$
8. $\text{enqueue}(Q, v)$
9. $\text{dequeue}(Q)$
10. $\text{color}[u] \leftarrow \text{schwarz}$



Graphalgorithmen

Satz 47

Sei $G=(V,E)$ ein Graph. Die Laufzeit des Algorithmus BFS beträgt $O(|V|+|E|)$.

Beweis

- Laufzeit Initialisierung: $O(|V|)$
- Nach der Initialisierung wird kein Knoten weiß gefärbt
- Daher ist jeder Knoten nur einmal in der Schlange
- \Rightarrow Zeit für Schlangenoperationen ist $O(|V|)$
- Adjazenzliste jedes Knotens wird nur durchlaufen, wenn er aus der Schlange entfernt wird
- Damit wird jede Adjazenzliste maximal einmal durchlaufen (d.h. jede Kante maximal zweimal) \Rightarrow Laufzeit für Liste: $O(|E|)$
- Gesamtlaufzeit: $O(|V|+|E|)$

Graphalgorithmen

Kürzeste Wege in ungewichteten Graphen

- Sei $\delta(s,t)$ die minimale Anzahl Kanten in einem s-t-Weg
- Ein s-t-Weg der Länge $\delta(s,t)$ heißt **kürzester Weg**
- Wollen zeigen, dass BFS korrekt kürzeste Wege berechnet

Graphalgorithmen

Lemma 48

Sei $G=(V,E)$ ein gerichteter oder ungerichteter Graph und sei $s \in V$ ein beliebiger Knoten. Dann gilt für jede Kante $(u,v) \in E$:

$$\delta(s,v) \leq \delta(s,u)+1$$

Beweis

- Ist u erreichbar von s , dann ist es auch v
- Der kürzeste Weg von s nach v kann nicht länger sein, als der kürzeste Weg von s nach u gefolgt von der Kante (u,v) . Damit gilt die Ungleichung.
- Ist u nicht erreichbar von s , dann ist $\delta(s,u)=\infty$ und die Ungleichung gilt.

Graphalgorithmen

Lemma 49

Sei $G=(V,E)$ ein gerichteter oder ungerichteter Graph und es laufe die Breitensuche von einem Startknoten $s \in V$. Nach Abschluss der Breitensuche gilt für jeden Knoten v , dass $d[v] \geq \delta(s,v)$ ist.

Beweis

- Induktion über Anzahl von Zeitpunkten, an denen ein Knoten in Q eingefügt wird
- (I.A.) Nach Initialisierung gilt $d[s]=0=\delta(s,s)$ und $d[v]=\infty \geq \delta(s,v)$ für alle $v \in V - \{s\}$
- (I.V.) Aussage gilt nach m Einfügeoperationen
- (I.S.) Betrachte nach m Einfügeoperationen einen weißen Knoten v , der während einer Suche von u entdeckt wird. Nach (I.V.) gilt $d[u] \geq \delta(s,u)$.
- Zeile 7: $d[v]$ wird auf $d[u]+1$ gesetzt
- Es gilt: $d[v] = d[u]+1 \geq \delta(s,u)+1 \geq \delta(s,v)$ nach Lemma 48

Graphalgorithmen

Lemma 49

Sei $G=(V,E)$ ein gerichteter oder ungerichteter Graph und es laufe die Breitensuche von einem Startknoten $s \in V$. Nach Abschluss der Breitensuche gilt für jeden Knoten v , dass $d[v] \geq \delta(s,v)$ ist.

Beweis

- Knoten v wird dann in die Schlange eingefügt und grau gefärbt
- Damit ändert sich $d[v]$ im Laufe des Algorithmus nicht mehr und die Aussage des Lemmas bleibt erhalten

Graphalgorithmen

Lemma 50

Sei $\langle v_1, \dots, v_r \rangle$ der Inhalt der Schlange Q während eines Durchlaufs der Breitensuche auf einem Graph $G=(V,E)$, wobei v_1 Kopf und v_r Ende der Schlange ist. Dann gilt
 $d[v_r] \leq d[v_1] + 1$ und $d[v_i] \leq d[v_{i+1}]$ für $i=1,2,\dots,r-1$.

Beweis

- Induktion über die Anzahl Schlangenoperationen
- (I.A.) Die Schlange enthält nur s , damit gilt das Lemma
- (I.V.) Das Lemma gilt nach m Schlangenoperationen
- (I.S.) Wir müssen zeigen, dass das Lemma immer noch nach $m+1$ Schlangenoperationen gilt. Die $(m+1)$ ste Schlangenoperation ist entweder eine enqueue oder dequeue Operation

Graphalgorithmen

Lemma 50

Sei $\langle v_1, \dots, v_r \rangle$ der Inhalt der Schlange Q während eines Durchlaufs der Breitensuche auf einem Graph $G=(V,E)$, wobei v_1 Kopf und v_r Ende der Schlange ist. Dann gilt
 $d[v_r] \leq d[v_1] + 1$ und $d[v_i] \leq d[v_{i+1}]$ für $i=1,2,\dots,r-1$.

Beweis

- Dequeue:
- Wird v_1 aus der Schlange entfernt, so wird v_2 der neue Kopf
- Dann gilt aber sicherlich $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$
- Alle anderen Ungleichungen sind nicht betroffen, also gilt das Lemma

Graphalgorithmen

Lemma 50

Sei $\langle v_1, \dots, v_r \rangle$ der Inhalt der Schlange Q während eines Durchlaufs der Breitensuche auf einem Graph $G=(V,E)$, wobei v_1 Kopf und v_r Ende der Schlange ist. Dann gilt
 $d[v_r] \leq d[v_1] + 1$ und $d[v_i] \leq d[v_{i+1}]$ für $i=1,2,\dots,r-1$.

Beweis

- Enqueue:
- Wird in Zeile 8 ein Knoten v eingefügt (und damit zu v_{r+1}), so ist v_1 der Knoten u , von dem aus v entdeckt wurde
- Es gilt: $d[v_{r+1}] = d[v] = d[u] + 1 = d[v_1] + 1$
- Außerdem: $d[v_r] \leq d[v_1] + 1 = d[u] + 1 = d[v] = d[v_{r+1}]$
- Die anderen Ungleichungen bleiben erhalten; Also gilt das Lemma

Graphalgorithmen

Satz 51

Sei $G=(V,E)$ ein gerichteter oder ungerichteter Graph und sei $s \in V$ Startknoten der Breitensuche. Dann entdeckt die Breitensuche alle Knoten $v \in V$, die von s aus erreichbar sind und nach Terminierung gilt $d[v]=\delta(s,v)$ für alle $v \in V$. Außerdem gilt für jeden von s erreichbaren Knoten $v \neq s$, dass ein kürzester Weg von s nach $\pi[v]$ gefolgt von der Kante $(\pi[v],v)$ ein kürzester s - v -Weg ist.

Beweis

- Fall 1 (v nicht erreichbar von s):
- Nach Lemma 49 gilt $d[v] \geq \delta(s,v) = \infty$
- Es kann keinen ersten Knoten geben, dessen d -Wert in Zeile 7 auf ∞ gesetzt wird
- Somit wird v nie entdeckt

Graphalgorithmen

Beweis

- Fall 2 (v erreichbar von s):
- Sei $V_k = \{v \in V : \delta(s, v) = k\}$
- Wir zeigen per Induktion über k :
- Es gibt genau einen Zeitpunkt, zu dem jeder Knoten $v \in V_k$
 - (a) grau gefärbt wird
 - (b) $d[v]=k$ gesetzt wird
 - (c) wenn $v \neq s$, dann $\pi[v]$ auf u gesetzt wird für ein $u \in V_{k-1}$
 - (d) v in Schlange Q eingefügt wird
- Da nur zur Initialisierung Knoten weiß gefärbt werden, gibt es maximal einen solchen Zeitpunkt

Graphalgorithmen

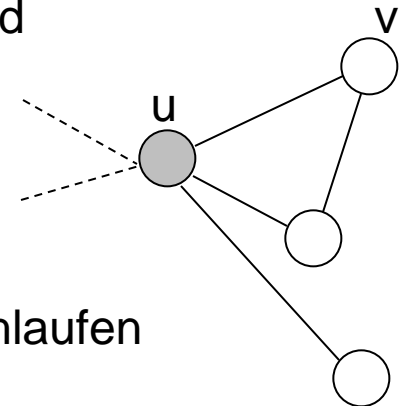
Beweis

- (I.A.) $V_0 = \{s\}$. Während der Initialisierung wird s grau gefärbt, $d[s]$ auf 0 gesetzt, und s in Q eingefügt. Somit gilt die Aussage.
- (I.V.) Aussage gilt für alle Knoten aus V_{k-1}
- (I.S.) Q ist nie leer bis Algorithmus terminiert.
- Nachdem v in Q eingefügt wurde, ändern sich $d[v]$ und $\pi[v]$ nicht mehr
- Nach Lemma 50 sind die d -Werte monoton steigend, wenn Knoten in die Schlange eingefügt werden
- Betrachte nun $v \in V_k$, $k > 0$.
Monotonie mit $d[v] \geq k$ (Lemma 49) und (I.V.): wenn v entdeckt wird, dann erst nachdem alle Knoten aus V_{k-1} in die Schlange eingefügt wurden

Graphalgorithmen

Beweis

- Da $\delta(s,v)=k$ gibt es Pfad mit k Kanten von s nach v und Knoten u mit $(u,v) \in E$ und $u \in V_{k-1}$
- ObdA. Sei u der erste solche Knoten, der grau gefärbt wird (existiert wegen I.V.)
- Wird Knoten grau gefärbt, so wird er auch in Schlange eingefügt und muss irgendwann als Kopf der Schlange auftauchen
- Ist u Kopf der Schlange, so wird seine Adjazenzliste durchlaufen und v in Zeile 4 entdeckt
- Dann wird v in Zeile 6 grau gefärbt und Zeile 7 setzt $d[v]=k$ und $\pi[v]=u$.
- Zeile 8 fügt v in die Schlange ein
- Damit folgt unsere Aussage per Induktion für alle V_k



Graphalgorithmen

Beweis

- Abschließend beobachten wir, dass wenn $v \in V_k$ ist, dann ist $\pi[v]$ in V_{k-1}
- Damit können wir einen kürzesten Weg von s nach v bekommen, indem wir einen kürzesten Weg von s nach $\pi[v]$ nehmen und der Kante $(\pi[v], v)$ folgen

Graphalgorithmen

Zusammenfassung (Breitensuche)

- Die Breitensuche kann dazu genutzt werden, um das SSSP Problem in ungewichteten Graphen zu lösen
- Die Laufzeit der Breitensuche ist $O(|V|+|E|)$

Graphalgorithmen

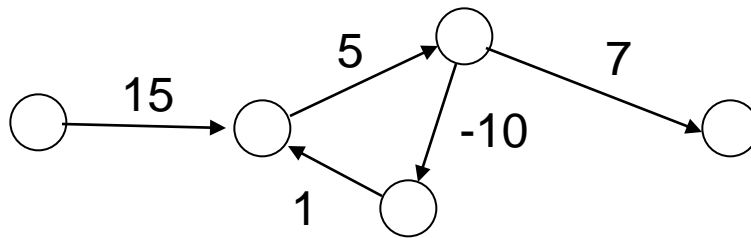
Kürzeste Wege in gewichteten Graphen

- $G=(V,E)$
- $w: E \rightarrow \mathcal{R}$; $w(e)$ ist Länge der Kante e ; $w(u,v)$ ist Länge der Kante (u,v)
- Für Pfad $\langle v_0, v_1, \dots, v_k \rangle$ ist Pfadlänge gegeben durch $\sum_{i=1}^k w(v_{i-1}, v_i)$
- $\delta(u,v) = \min_{\text{Pfade } p \text{ von } u \text{ nach } v} w(p)$, falls es Pfad von u nach v gibt
- $\delta(u,v) = \infty$, sonst

Graphalgorithmen

Negative Kantengewichte

- Manchmal hat man Instanzen mit negativen Kantenlängen



- Bei ungerichteten Graphen kann man Kante immer wieder vorwärts und rückwärts durchlaufen
- Kürzester Pfad u.U. nicht wohldefiniert
- Erstmal nichtnegative Kantengewichte

Graphalgorithmen

Lemma 52

- Sei $G=(V,E)$ ein gewichteter, gerichteter Graph mit Kantengewichten $w(e)$ und sei $\langle v_1, \dots, v_k \rangle$ ein kürzester Weg von v_1 nach v_k . Dann ist für alle $1 \leq i, j \leq k$ der Weg $\langle v_i, \dots, v_j \rangle$ ein kürzester Weg von v_i nach v_j .

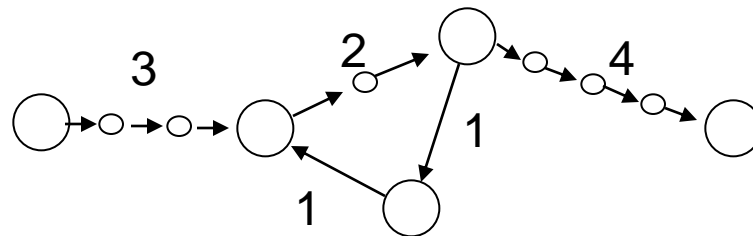
Beweis

- Annahme: Es gäbe einen kürzeren Weg $\langle v_i, u_1, \dots, u_l, v_j \rangle$ von v_i nach v_j . Dann wäre $\langle v_1, \dots, v_i, u_1, \dots, u_l, v_j, \dots, v_k \rangle$ kürzer als $\langle v_1, \dots, v_k \rangle$. Widerspruch.

Graphalgorithmen

Dijkstra's Algorithmus

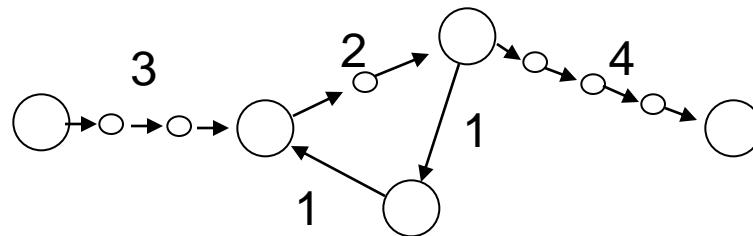
- Graph in Adjazenzlistendarstellung
- Keine negativen Kantenlängen
- Überträgt Idee der Breitensuche auf gewichtete Graphen
- Erster Ansatz: Ersetze Kantenlängen durch mehrfache Kanten



Graphalgorithmen

Dijkstra's Algorithmus

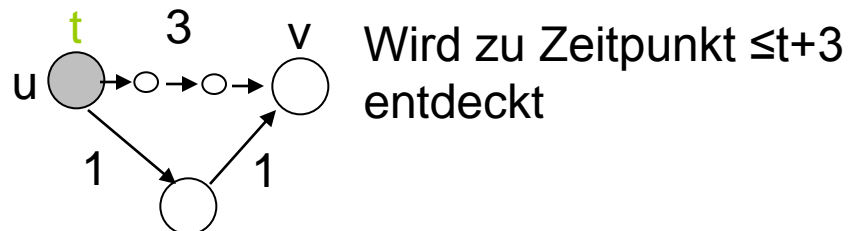
- Erste Idee: Ersetze Kantenlängen durch mehrfache Kanten
- Probleme: Langsam bei großen Kantenlängen; nur ganzzahlige Längen
- Zunächst ganzzahlige Längen. Idee: Simuliere Breitensuche effizient
- Aufgabe: Bestimme für jeden Knoten den Zeitpunkt, zu dem er entdeckt wird



Graphalgorithmen

Dijkstra's Algorithmus

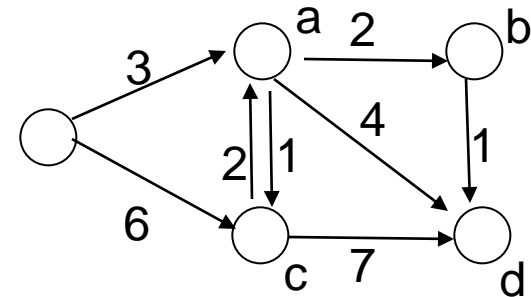
- Betrachte Breitensuche in der expandierten Version von G
- $u, v \in V$
- Wird ein Knoten u zum Zeitpunkt t (d.h. $d[u]=t$) entdeckt und ist Kante (u, v) mit Gewicht $w(u, v)$ in G , so wird v spätestens zum Zeitpunkt $t+w(u, v)$ entdeckt
- Unter Umständen wird v eher über einen anderen Knoten entdeckt



Graphalgorithmen (siehe Vollversion)

BreitensucheSimulation(G, w, s)

1. Initialisiere Simulation
2. Füge $(s, \text{prio}[s])$ mit Priorität $\text{prio}[s]$ in Prioritätenschlange Q ein
3. **while** $Q \neq \emptyset$ **do**
4. $(u, \text{prio}[u]) \leftarrow \text{ExtractMin}(Q)$
5. **if** $\text{color}[u] = \text{weiß}$ **then**
6. $\text{color}[u] \leftarrow \text{schwarz}$
7. $d[u] \leftarrow \text{prio}[u]$
8. **for each** $v \in \text{Adj}[u]$ **do**
9. $\text{prio}[v] \leftarrow d[u] + w(u, v)$
10. Füge $(v, \text{prio}[v])$ mit Priorität $\text{prio}[v]$ in Q ein

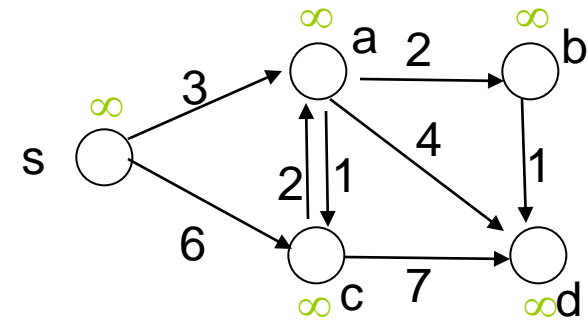


Graphalgorithmen (siehe Vollversion)

BreitensucheSimulation(G, w, s)

1. Initialisiere Simulation
2. Füge $(s, \text{prio}[s])$ mit Priorität $\text{prio}[s]$ in Prioritäten
3. **while** $Q \neq \emptyset$ **do**
4. $(u, \text{prio}[u]) \leftarrow \text{ExtractMin}(Q)$
5. **if** $\text{color}[u] = \text{weiß}$ **then**
6. $\text{color}[u] \leftarrow \text{schwarz}$
7. $d[u] \leftarrow \text{prio}[u]$
8. **for each** $v \in \text{Adj}[u]$ **do**
9. $\text{prio}[v] \leftarrow d[u] + w(u, v)$
10. Füge $(v, \text{prio}[v])$ mit Priorität $\text{prio}[v]$ in Q ein

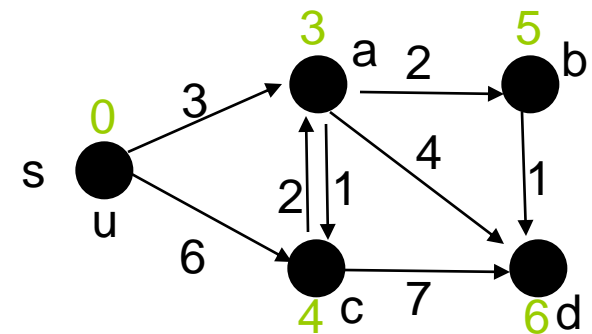
$d[u] = \infty$ für alle $u \in V$
 $\text{prio}[u] = \infty$
 $\text{prio}[s] = 0$
 $\text{color}[u] = \text{weiß}$ für alle $u \in V$



Graphalgorithmen (siehe Vollversion)

BreitensucheSimulation(G, w, s)

1. Initialisiere Simulation
2. Füge $(s, \text{prio}[s])$ mit Priorität $\text{prio}[s]$ in Prioritätenschlange Q ein
3. **while** $Q \neq \emptyset$ **do**
4. $(u, \text{prio}[u]) \leftarrow \text{ExtractMin}(Q)$
5. **if** $\text{color}[u] = \text{weiß}$ **then**
6. $\text{color}[u] \leftarrow \text{schwarz}$
7. $d[u] \leftarrow \text{prio}[u]$
8. **for each** $v \in \text{Adj}[u]$ **do**
9. $\text{prio}[v] \leftarrow d[u] + w(u, v)$
10. Füge $(v, \text{prio}[v])$ mit Priorität $\text{prio}[v]$ in Q ein



Graphalgorithmen

BreitensucheSimulation(G, w, s)

1. Initialisiere Simulation
2. Füge $(s, \text{prio}[s])$ mit Priorität $\text{prio}[s]$ in Prioritätenschlange ein
3. **while** $Q \neq \emptyset$ **do**
4. $(u, \text{prio}[u]) \leftarrow \text{ExtractMin}(Q)$
5. **if** $\text{color}[u] = \text{weiß}$ **then**
6. $\text{color}[u] \leftarrow \text{schwarz}$
7. $d[u] \leftarrow \text{prio}[u]$
8. **for each** $v \in \text{Adj}[u]$ **do**
9. $\text{prio}[v] \leftarrow d[u] + w(u, v)$
10. Füge $(v, \text{prio}[v])$ mit Priorität $\text{prio}[v]$ in Q ein

Beobachtung:

Sind mehrere Paare (u, p) in der Prioritätenschlange, so ist nur das mit der geringsten Priorität relevant

Beobachtung:

d -Werte und Prioritäten fast identisch

Graphalgorithmen

Dijkstra's Algorithmus(G, w, s)

1. Initialisiere SSSP
2. $Q \leftarrow V[G]$
3. **while** $Q \neq \emptyset$ **do**
4. $u \leftarrow \text{ExtractMin}(Q)$
5. **if** $\text{color}[u] = \text{weiß}$ **then**
6. $\text{color}[u] \leftarrow \text{schwarz}$
7. **for each** $v \in \text{Adj}[u]$ **do**
8. **if** $d[u] + w(u, v) < d[v]$ **then**
9. $d[v] \leftarrow d[u] + w(u, v)$
10. $\text{DecreaseKey}(v, d[v])$

$d[u] = \infty$ für alle $u \in V - \{s\}$
 $d[s] = 0$
 $\text{color}[u] = \text{weiß}$ für alle $u \in V$

Invariante:
Für alle schwarzen Knoten wurde
die Distanz korrekt berechnet.

Graphalgorithmen

Prioritätenschlängen mit DecreaseKey

- $\text{DecreaseKey}(v,p)$: Erlaubt das Verringern der Priorität von Schlüssel v auf Wert p

Einfachste Implementation

- Lösche v
- Füge v mit Priorität p in Prioritätenschlange ein

Graphalgorithmen

Lemma 53

Sei $G=(V,E)$ ein gerichteter oder ungerichteter Graph und sei $s \in V$ ein beliebiger Knoten. Dann gilt für jede Kante $(u,v) \in E$:

$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$

Beweis

- (Argumentation identisch zu Lemma 48)
- Ist u erreichbar von s , dann ist es auch v
- Der kürzeste Weg von s nach v kann nicht länger sein, als der kürzeste Weg von s nach u gefolgt von der Kante (u,v) . Damit gilt die Ungleichung.
- Ist u nicht erreichbar von s , dann ist $\delta(s,u) = \infty$ und die Ungleichung gilt.

Graphalgorithmen

Lemma 54

Zu jedem Ausführungszeitpunkt von Dijkstra's Algorithmus gilt für jeden Knoten w :

$$d[w] \geq \delta(s,w).$$

Beweis

- Zeile 9 ist die einzige Zeile, in der d -Werte geändert werden.
- Wir zeigen per Induktion über die Ausführungen von Zeile 9, dass Lemma 54 gilt.
- (I.A.) Vor der ersten Ausführung entsprechen die d -Werte den Werten direkt nach der Initialisierung. Für diese Werte gilt das Lemma.
- (I.V.) Das Lemma gilt nach m Ausführungen von Zeile 9.
- (I.S.) Betrachte $(m+1)$ sten Ausführung. Nach (I.V.) gilt $d[v] \geq \delta(s,v)$ und $d[u] \geq \delta(s,u)$. Wir setzen in Zeile 9 $d[v]$ auf $d[u]+w(u,v)$.

Graphalgorithmen

Lemma 54

Zu jedem Ausführungszeitpunkt von Dijkstra's Algorithmus gilt für jeden Knoten w :

$$d[w] \geq \delta(s,w).$$

Beweis

- (I.S.) Betrachte $(m+1)$ sten Ausführung. Nach (I.V.) gilt $d[v] \geq \delta(s,v)$ und $d[u] \geq \delta(s,u)$. Wir setzen in Zeile 9 $d[v]$ auf $\min\{d[v], d[u]+w(u,v)\}$. Es gilt $d[v] = d[u] + w(u,v) \geq \delta(s,u) + w(u,v) \geq \delta(s,v)$ nach Lemma 53. Somit gilt auch hier das Lemma.

Graphalgorithmen

Satz 55

Wenn wir Dijkstra's Algorithmus auf einem gewichteten, gerichteten Graph $G=(V,E)$ mit nichtnegativen Kantengewichten und Startknoten s ausführen, so gilt nach Terminierung $d[u] = \delta(s,u)$ für alle Knoten $u \in V$.

Beweis

- Jeder Knoten wird im Verlauf des Algorithmus schwarz gefärbt.
- Wir zeigen per Widerspruchsbeweis, dass für jeden Knoten $u \in V$ zum Zeitpunkt des Schwarzfärbens $d[u] = \delta(s,u)$ gilt.
- Annahme: Es gibt einen Knoten u , für den zum Zeitpunkt des Schwarzfärbens $d[u] \neq \delta(s,u)$ gilt. Sei u der erste solche Knoten. Betrachte die Situation zu Beginn des Durchlaufs der while-Schleife, in dem u schwarz gefärbt wird. Es gilt $u \neq s$, da s als erster Knoten schwarz gefärbt wird und zu diesem Zeitpunkt $d[s]=0 = \delta(s,s)$ gilt. (Widerspruch!)

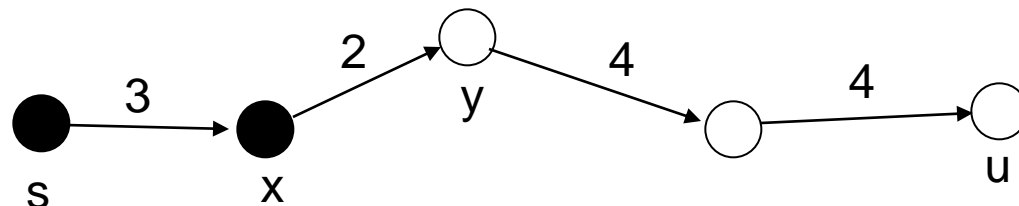
Graphalgorithmen

Satz 55

Wenn wir Dijkstra's Algorithmus auf einem gewichteten, gerichteten Graph $G=(V,E)$ mit nichtnegativen Kantengewichten und Startknoten s ausführen, so gilt nach Terminierung $d[u] = \delta(s,u)$ für alle Knoten $u \in V$.

Beweis

- Es gibt Weg von s nach u , da ansonsten wegen Lemma 54 $d[u]=\infty=\delta(s,u)$ gilt. (Widerspruch!)
- Sei nun y der erste weiße Knoten auf einem kürzesten Weg von s nach u und x sein Vorgänger.



Graphalgorithmen

Beweis

- Es gilt $d[x] = \delta(s, u)$ nach Wahl von u .
- In Zeile 9 wird nun $d[y]$ auf $\min\{d[y], d[x] + w(x, y)\}$ gesetzt. Nach Lemma 52 ist der Weg von s nach y über x ein kürzester Weg (da er „Teilweg“ des kürzesten Weges von s nach u ist). Somit ist $d[x] + w(x, y) = \delta(s, y)$ und $d[y]$ wird wegen Lemma 53 auf diesen Wert gesetzt.
- Da die Kantengewichte nichtnegativ sind, folgt $\delta(s, y) \leq \delta(s, u)$ und somit nach Lemma 53 $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$.
- Da aber u von ExtractMin aus der Prioritätenschlange entfernt wurde, gilt $d[u] \leq d[y]$ und somit $d[y] = \delta(s, y) = \delta(s, u) = d[u]$. Widerspruch!

