



Datenstrukturen, Algorithmen und Programmierung 2 (DAP2)

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. $\text{DecreaseKey}(v, d[v])$

Graphalgorithmen

Dijkstra's Algorithmus

- Fällt unter die Kategorie gierige Algorithmen
- Laufzeit -> Präsenzübung

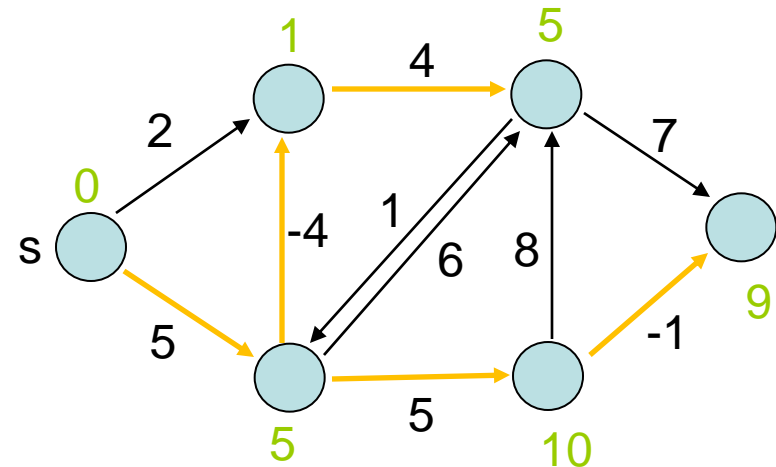
Graphalgorithmen

Single Source Shortest Path (SSSP)

- Eingabe: Gewichteter Graph $G=(V,E)$ und Startknoten s
- Ausgabe: Für jeden Knoten $u \in V$ seine Distanz zu s sowie einen kürzesten Weg

Heute

- Negative Kantengewichte

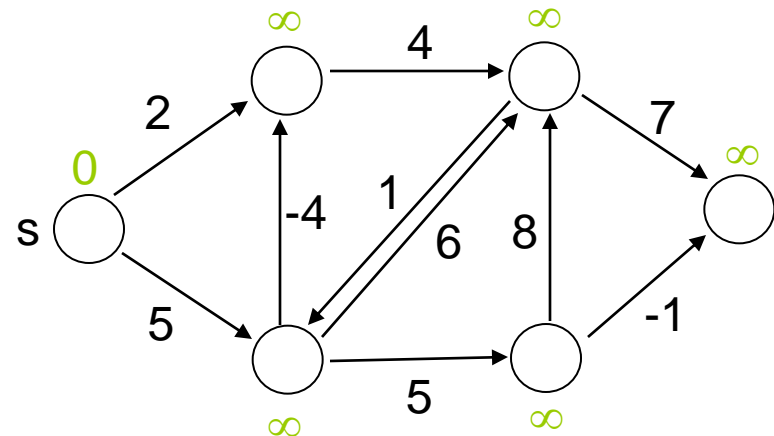


Graphalgorithmen (siehe Vollversion)

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. DecreaseKey($v, d[v]$)

Was berechnet Dijkstra's Algorithmus?

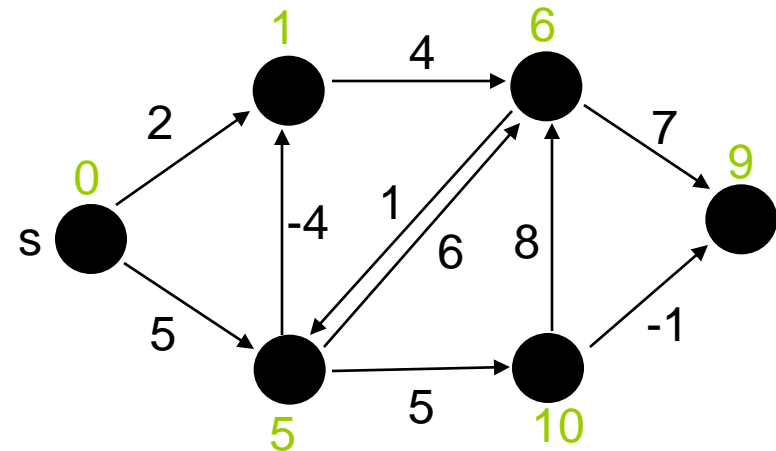


Graphalgorithmen (siehe Vollversion)

Dijkstra's Algorithmus(G, w, s)

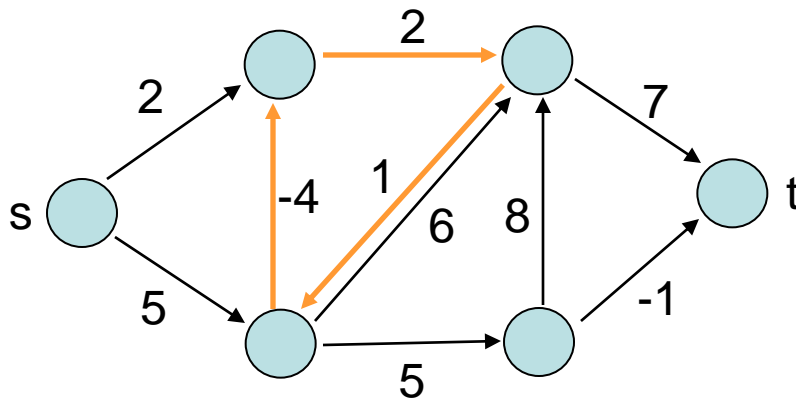
1. Initialisiere SSSP
2. $Q \leftarrow V[G]$
3. **while** $Q \neq \emptyset$ **do**
4. $u \leftarrow \text{ExtractMin}(Q)$
5. **if** $\text{color}[u] = \text{weiß}$ **then**
6. $\text{color}[u] \leftarrow \text{schwarz}$
7. **for each** $v \in \text{Adj}[u]$ **do**
8. **if** $d[u] + w(u, v) < d[v]$ **then**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

Problem: Die Veränderung durch die negative Kante wird nicht mehr durch den Graphen propagiert.



Graphalgorithmen

Negative Zyklen



Problem

- Kann Weg mit beliebig kleinen Kosten finden
- Hier z.B. von s nach t

Graphalgorithmen

Unser Ansatz

- Betrachte zunächst nur Eingaben ohne negative Zyklen
- Dynamische Programmierung
- Frage: Wie formuliert man das Problem rekursiv?

Graphalgorithmen

Lemma 56

Wenn G keine negativen Zyklen hat und t von s aus erreichbar ist, dann gibt es einen kürzesten s - t -Weg in G , in dem kein Knoten doppelt vorkommt.

Beweis

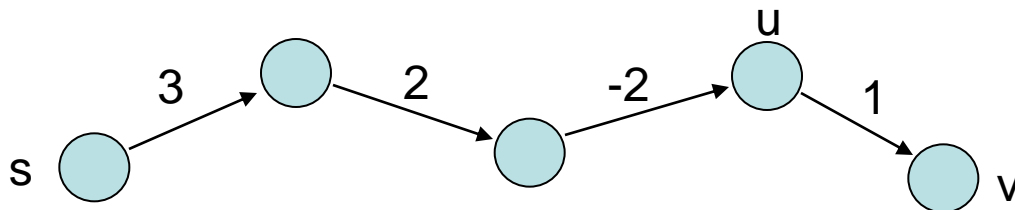
- Annahme: Es gibt keinen kürzesten s - t -Weg in G , in dem kein Knoten doppelt vorkommt.
- Dann kommt in jedem kürzesten s - t -Weg ein Knoten zweimal vor.
- Betrachte den kürzesten s - t -Weg P mit der geringsten Kantenanzahl.
- In P kommt mindestens ein Knoten zweimal vor. Sei dies Knoten v .
- Wir können den Teil von v nach v entfernen, da jeder Kreis nichtnegative Länge hat und erhalten einen kürzesten s - t -Weg mit weniger Kanten.
- Widerspruch zur Wahl von P !

Graphalgorithmen

Eine rekursive Problemformulierung

- $\text{Opt}(i,v)$ sei Länge eines optimalen s-v-Wegs, der maximal i Kanten benutzt
- Sei P ein optimaler s-v-Weg mit max. i Kanten

- $$\text{Opt}(i,v) = \begin{cases} \text{Opt}(i-1,v), & \text{falls } P \text{ weniger als } i \text{ Kanten hat} \\ \text{Opt}(i-1,u) + w(u,v), & \text{falls } P \text{ genau } i \text{ Kanten} \\ & \text{benutzt und } (u,v) \text{ die letzte Kante bezeichnet} \end{cases}$$



Graphalgorithmen

Die Rekursion

- Für $i > 0$ gilt

$$\text{Opt}(i,v) = \min \{ \text{Opt}(i-1,v), \min_{(u,v) \in E} (\text{Opt}(i-1,u) + w(u,v)) \}$$

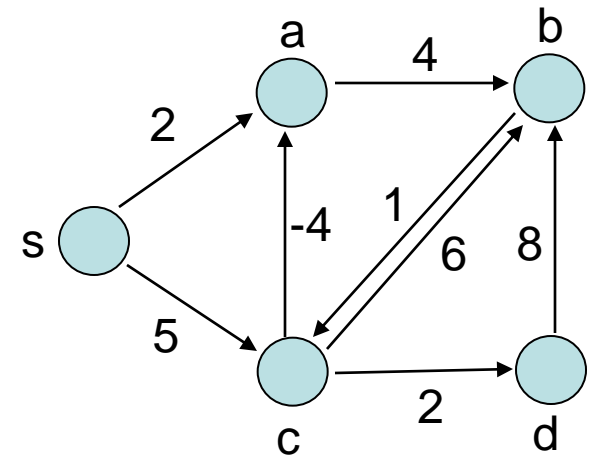
- Für $i=0$ gilt

$$\text{Opt}(0,s)=0 \text{ und } \text{Opt}(0,v)=\infty \text{ für } v \neq s$$

Graphalgorithmen (siehe Vollversion)

Bellman-Ford(G,s)

1. **for each** $v \in V$ **do** $M[0][v] = \infty$
2. $M[0][s] \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** $|V|-1$ **do**
4. **for each** $v \in V$ **do**
5. Berechne $M[i,v]$ nach Rekursionsgleichung
6. **return** M

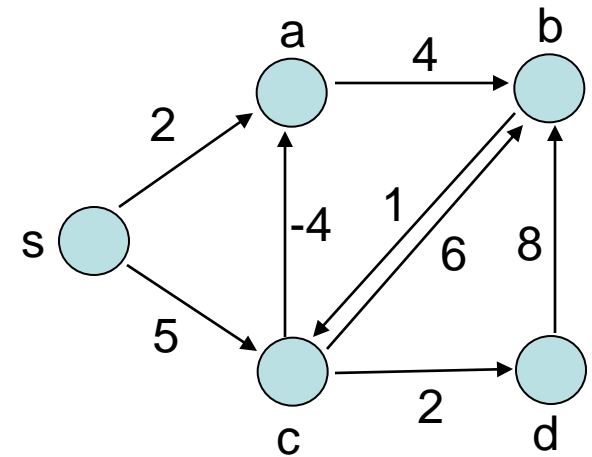


	s	a	b	c	d
0					
1					
2					
3					
4					

Graphalgorithmen (siehe Vollversion)

Bellman-Ford(G,s)

1. **for each** $v \in V$ **do** $M[0][v] = \infty$
2. $M[0][s] \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** $|V|-1$ **do**
4. **for each** $v \in V$ **do**
5. Berechne $M[i,v]$ nach Rekursionsgleichung
6. **return** M



	s	a	b	c	d
0	0	∞	∞	∞	∞
1	0	2	∞	5	∞
2	0	1	6	5	7
3	0	1	5	5	7
4	0	1	5	5	7

Graphalgorithmen

Bellman-Ford(G,s)

1. **for each** $v \in V$ **do** $M[0][v] = \infty$
2. $M[0][s] \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** $|V|-1$ **do**
4. **for each** $v \in V$ **do**
5. Berechne $M[i,v]$ nach Rekursionsgleichung
6. **return** M

Laufzeit: $O(|V|^2)$ für Init. von M

$O(|V|)$

$O(|V|)$

$O(|V|)$

$O(|V|^2)$

$O(|V|^3)$

$O(1)$

$O(|V|^2 |E|)$

Graphalgorithmen

Satz 57

Sei G ein Graph ohne negative Zyklen. Algorithmus Bellman-Ford berechnet für jeden Knoten v aus G die Kosten eines kürzesten s - v -Pfads. Laufzeit der ersten Version des Algorithmus ist $O(|V|^3)$ und Speicherbedarf ist $O(|V|^2)$.

Beweis

- Korrektheit per Induktion. Sei $\text{Opt}(i,v)$ die Länge eines kürzesten Weges mit i Kanten von s nach v . Wir zeigen: $M(i,v)=\text{Opt}(i,v)$.
- (I.A.) $M(0,v)=\text{Opt}(0,v)$, da $M(0,s)=0=\text{Opt}(0,s)$ und $M(0,v)=\infty=\text{Opt}(0,v)$ für $v \neq s$.
- (I.V.) Die Aussage gilt für $i-1$.

Graphalgorithmen

Satz 57

Sei G ein Graph ohne negative Zyklen. Algorithmus Bellman-Ford berechnet für jeden Knoten v aus G die Kosten eines kürzesten s - v -Pfads. Laufzeit der ersten Version des Algorithmus ist $O(|V|^3)$ und Speicherbedarf ist $O(|V|^2)$.

Beweis

- (I.S.) Wir zeigen $\text{Opt}(i,v) \leq M(i,v)$ und $\text{Opt}(i,v) \geq M(i,v)$. Damit folgt die Gleichheit.
- (1) $\text{Opt}(i,v) \leq M(i,v)$:
- Nach (I.V.) enthält $M(i-1,v)$ die Länge eines kürzesten Weges mit maximal $i-1$ Kanten von s nach v . $M(i-1,u)+w(u,v)$ gibt nach (I.V.) die Länge eines Weges mit maximal $i-1$ Kanten von s nach u gefolgt von der Kante (u,v) an. Damit handelt es sich um die Länge eines Weges mit maximal i Kanten von s nach v . Da $M(i,v)$ als die Länge eines kürzesten Weges aus einer Menge von Wegen mit maximal i Kanten gewählt wird, gilt (1).

Graphalgorithmen

Satz 57

Sei G ein Graph ohne negative Zyklen. Algorithmus Bellman-Ford berechnet für jeden Knoten v aus G die Kosten eines kürzesten s - v -Pfads. Laufzeit der ersten Version des Algorithmus ist $O(|V|^3)$ und Speicherbedarf ist $O(|V|^2)$.

Beweis

- (2) $\text{Opt}(i,v) \geq M(i,v)$:
- Betrachte den kürzesten Weg mit i Kanten von s nach v . Hat dieser weniger als i Kanten, so folgt nach (I.V.) $M(i,v) \leq M(i-1,v) = \text{Opt}(i-1,v) = \text{Opt}(i,v)$ ist.
- Hat der Weg genau i Kanten, so betrachte die letzte Kante des Weges, sagen wir (u,v) . Wegen $M(i,v) \leq M(i-1,u) + w(u,v)$ gilt (2) auch in diesem Fall.
- Es folgt: $\text{Opt}(i,v) = M(i,v)$.

Graphalgorithmen

Satz 57

Sei G ein Graph ohne negative Zyklen. Algorithmus Bellman-Ford berechnet für jeden Knoten v aus G die Kosten eines kürzesten s - v -Pfads. Laufzeit der ersten Version des Algorithmus ist $O(|V|^3)$ und Speicherbedarf ist $O(|V|^2)$.

Beweis

- Da es nach Lemma 56 in einem Graph ohne negativen Zyklen immer einen kürzesten Weg mit maximal $n-1$ Kanten gibt, ist $\text{Opt}(n-1, v)$ die Länge eines kürzesten Weges von s nach v .
- Wie bereits gezeigt, ist die Laufzeit $O(|V|^3)$. Der Speicherbedarf für das $|V| \times |V|$ Feld ist $O(|V|^2)$.

Graphalgorithmen

Lemma 58

$M(n,v)=M(n-1,v)$ gilt für alle Knoten $v \in V$ eines Graphen $G=(V,E)$ genau dann, wenn G keinen negativen Zyklus.

Beweis

- „ \Rightarrow “ Gilt $M(n,v)=M(n-1,v)$ so folgt wegen der Rekursion auch $M(m,v)=M(n,v)$ für jedes $m>n$. Außerdem gilt nach dem Beweis von Satz 57 $M(m,v)=\text{Opt}(m,v)$.
- Annahme: Es gibt negativen Zyklus C und $M(m,v)=M(n,v)$ für alle $m>n$. Sei c die Anzahl Kanten in C . Dann ist sicher, dass für jeden Knoten v aus C gilt: $\text{Opt}(n,v) < \text{Opt}(n+c,v)$. Dies ist richtig, da man zunächst einmal um C laufen kann und dann den Weg mit maximal n Kanten und Kosten $\text{Opt}(n,v)$ nimmt. Widerspruch, denn $M(n+c,v)=\text{Opt}(n+c,v) < \text{Opt}(n,v)=M(n,v)$.

Graphalgorithmen

Lemma 58

$M(n,v)=M(n-1,v)$ gilt für alle Knoten $v \in V$ eines Graphen $G=(V,E)$ genau dann, wenn G keinen negativen Zyklus.

Beweis

- „ \Leftarrow “ Hat G keinen negativen Zyklus so gibt es nach Lemma 56 für jeden Knoten v einen kürzesten s - v -Weg mit maximal $n-1$ Kanten. Daher enthält $\text{Opt}(n-1,v)$ bereits die Längen der kürzesten Wege und somit dieselben Werte wie $\text{Opt}(n,v)$. Es folgt nach dem Beweis von Satz 57: $M(n,v)=\text{Opt}(n,v)=\text{Opt}(n-1,v)=M(n-1,v)$.

Graphalgorithmen

Verbesserung der Laufzeit

- Vor Beginn des Algorithmus berechne in $O(|V|+|E|)$ Zeit eine „umgedrehte Adjazenzliste“
- Jeder Knoten v hat eine Liste $In[v]$ der eingehenden Kanten

Graphalgorithmen

Bellman-Ford(G,s)

1. **for each** $v \in V$ **do** $M[0][v] = \infty$
2. $M[0][s] \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** $|V|-1$ **do**
4. **for each** $v \in V$ **do**
5. Berechne $M[i,v]$ nach Rekursionsgleichung
6. **return** M

Graphalgorithmen

Bellman-Ford(G, s)

1. **for each** $v \in V$ **do** $M[0][v] = \infty$
2. $M[0][s] \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** $n-1$ **do**
4. **for each** $v \in V$ **do**
5. $\text{min} \leftarrow M[i-1][v]$
6. **for each** $(u, v) \in \text{In}[v]$ **do**
7. **if** $M[i-1][u] + w(u, v) < \text{min}$ **then** $\text{min} \leftarrow M[i-1][u] + w(u, v)$
8. $M[i][v] \leftarrow \text{min}$
9. **return** M

Laufzeit $O(|V|^2 + |V| \cdot |E|)$, da in Zeile 6 und 7 jede Kante einmal durchlaufen wird.

Graphalgorithmen

Verbesserung des Speicherbedarfs

- Wir speichern nur einen Wert $M[v]$ ab
- Dieser speichert die Länge des kürzesten Wegs nach v , den wir bisher gefunden haben
- Wir führen jetzt nur das Update $M[v] = \min (M[v], \min (M[u]+w(u,v)))$ durch

Beobachtung 59

- (1) $M[v]$ ist immer die Länge irgendeines Wegs von v nach t und
- (2) nach i Runden ist $M[v]$ höchstens so groß wie die Länge des kürzesten Wegs mit i Kanten.

Graphalgorithmen

Speicherbedarf: $O(|V|)$

Bellman-Ford(G,s)

1. **for each** $v \in V$ **do** $M[v] = \infty$
2. $M[s] \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** $n-1$ **do**
4. **for each** $v \in V$ **do**
5. $\text{min} \leftarrow M[v]$
6. **for each** $(u,v) \in \text{In}[v]$ **do**
7. **if** $M[u] + w(u,v) < \text{min}$ **then** $\text{min} \leftarrow M[u] + w(u,v)$
8. $M[v] \leftarrow \text{min}$
9. **return** M

Graphalgorithmen

Speicherbedarf: $O(|V|)$

Bellman-Ford(G,s)

1. **for each** $v \in V$ **do** $M[v] = \infty$
2. $M[s] \leftarrow 0$
3. **for** $i \leftarrow 1$ **to** $n-1$ **do**
4. **for each** $v \in V$ **do**
5. **for each** $(u,v) \in \text{In}[v]$ **do**
6. **if** $M[u] + w(u,v) < M[v]$ **then** $M[v] \leftarrow M[u] + w(u,v)$
7. **return** M

Graphalgorithmen

Satz 60

Sei G ein Graph ohne negative Zyklen. Die verbesserte Implementierung des Algorithmus Bellman-Ford berechnet für jeden Knoten v die Kosten eines kürzesten s - v -Pfads. Laufzeit der verbesserten Implementierung des Algorithmus ist $O(|V|^2 + |V| \cdot |E|)$ und Speicherbedarf ist $O(|V|)$.

Beweis

- Die Korrektheit folgt aus Satz 57 zusammen mit Beobachtung 59.
- Die Laufzeit haben wir im Wesentlichen analysiert.

Graphalgorithmen

Negative Zyklen

- Wir können negative Zyklen daran erkennen, dass sich für mindestens einen Knoten v der Wert $M[v]$ noch nach n Iterationen ändert
- Wir können uns die Wege über ein Feld π merken, ähnlich wie bei der Breitensuche

Graphalgorithmen

Zusammenfassung

- Bellman-Ford für allgemeine Kantengewichte; Laufzeit $O(|V|^2 + |V| \cdot |E|)$
- Negative Zyklen können erkannt werden