Grundzüge der Informatik 1

Vorlesung 22

Überblick Vorlesung

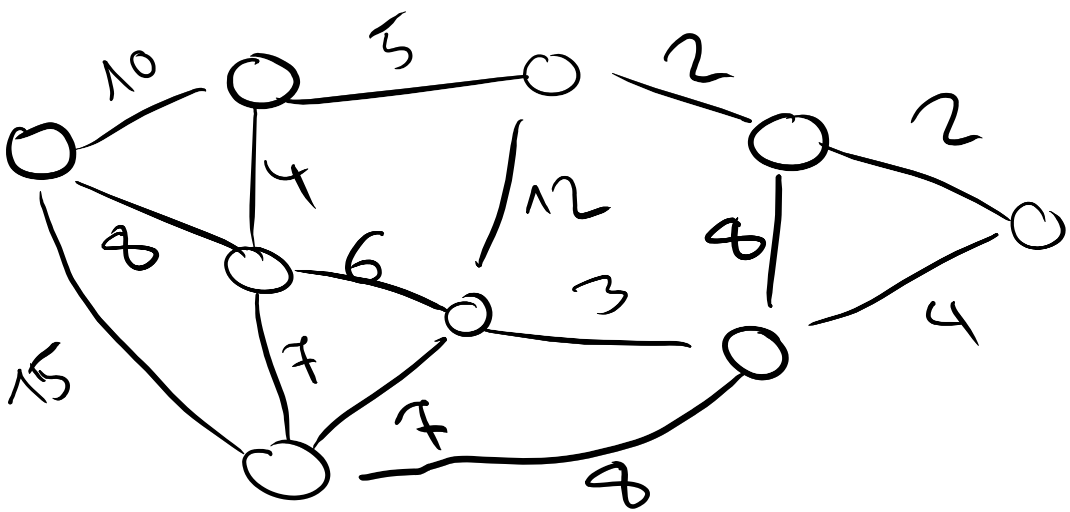
Graphenalgorithmen

- Kürzeste Wege in gewichteten Graphen mit negativen Kantengewichten
- SSSP mit Dynamischer Programmierung
  - Bellman-Ford Algorithmus
- APSP mit Dynamischer Programmierung
  - Floyd-Warshall Algorithmus

Graphenalgorithmen

Kürzeste Wege in Graphen

- Gegeben (möglicherweise gewichteter) Graph G=(V,E)
- Frage: Was ist der kürzeste Weg Knoten v nach Knoten u?
- Länge des Weges: Summe der Kantengewichte (bzw. Anzahl Kanten, wenn ungewichtet)

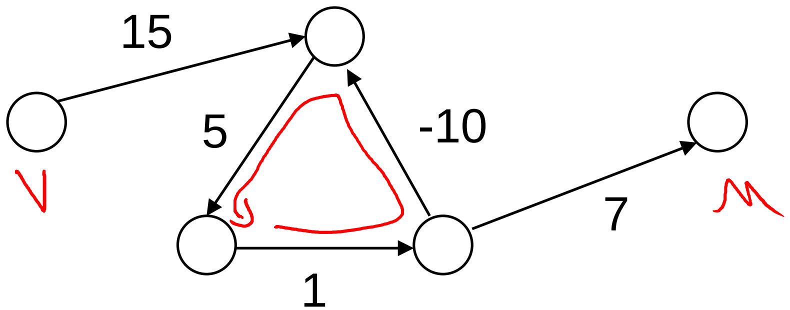


Universität zu Köln

Graphenalgorithmen

Negative Kantengewichte

- Manchmal hat man Instanzen mit negativen Kantenlängen



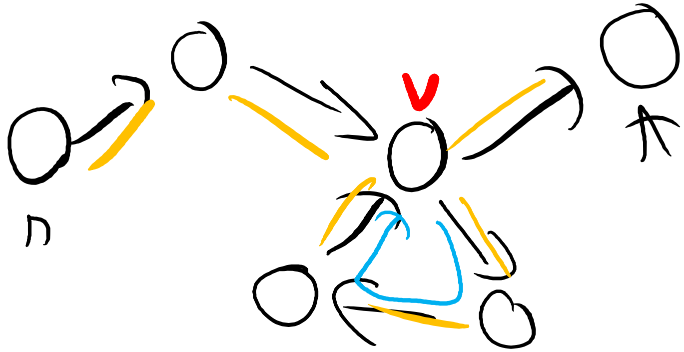
- Bei ungerichteten Graphen kann man Kante immer wieder vorwärts und rückwärts durchlaufen
- Kürzester Weg u.U. nicht wohldefiniert

Graphenalgorithmen

Unser Ansatz

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

Graphenalgorithmen



Lemma 22.1

Wenn G keine negativen Kreise 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

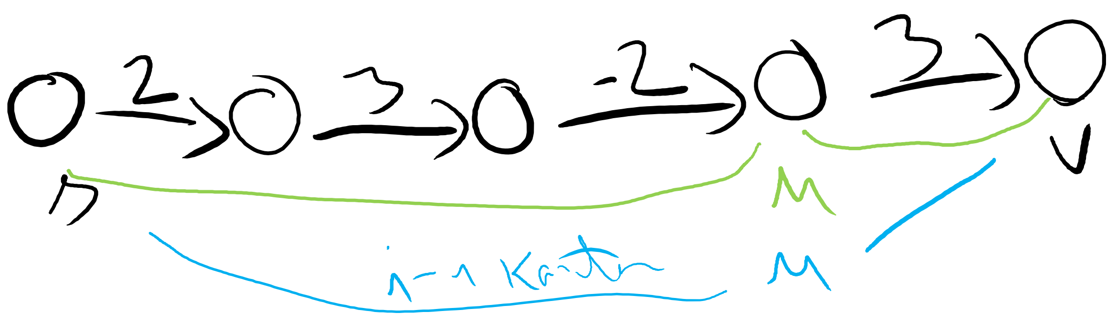
- Annahme: Es gibt keinen kürzesten s-t-Weg in G, in dem kein Knoten doppelt vorkommt.
- Dann kommt in jedem kürzesten s-t-Weg ein Knoten zweimal vor.
- Betrachte den kürzesten s-t-Weg P mit der geringsten Kantenanzahl.
- In P kommt mindestens ein Knoten zweimal vor. Sei dies Knoten v.
- Wir können den 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!

Graphenalgorithmen

Eine rekursive Problemformulierung

- 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

- Opt(i,v) = $\begin{cases} \text{Opt(i-1,v), falls P weniger als i Kanten hat} \\ \text{Opt(i-1,u)+w(u,v), falls P genau i Kanten hat und u der vorletzte} \\ \qquad \text{Knoten von P ist} \end{cases}$



Graphenalgorithmen

Die Rekursion

- $i>0$: $Opt(i,v) = \min_{(u,v)\in E} \{Opt(i-1,v), w(u,v) + Opt(i-1,u)\}$
- $i=0$: $Opt(0,s)= 0$ und $Opt(0,v)=\infty$ für $v\notin s$

Graphenalgorithmen

Lemma 22.2

- Die Rekursion
- i>0: $\mathrm{Opt}(i,v) = \min_{(u,v)\in E} \{\mathrm{Opt}(i-1,v), w(u,v) + \mathrm{Opt}(i-1,u)\}$
- i=0: $\mathrm{Opt}(0,s)= 0$ und $\mathrm{Opt}(0,v)=\infty$ für $v\notin s$
- beschreibt die Länge eines kürzesten Weges von s nach v mit maximal i Kanten.

Beweis

- Ist i=0, so ist die Rekursion korrekt
- Sei also i>0 und P ein kürzester Weg von s nach v mit maximal i Kanten
- Wenn P weniger als i Kanten hat, dann gilt $\mathrm{Opt}(i,v) = \mathrm{Opt}(i-1,v)$. Da für alle $(u,v)\in E$ $w(u,v) + \mathrm{Opt}(i-1,u)$ die Länge eines Weges mit maximal i Kanten von s nach v beschreibt, gilt in diesem Fall auch $\mathrm{Opt}(i,v) \le w(u,v)+\mathrm{Opt}(i-1,u)$
- Es folgt: $\mathrm{Opt}(i,v) = \min_{(u,v)\in E} \{\mathrm{Opt}(i-1,v), w(u,v) + \mathrm{Opt}(i-1,u)\}$

Graphenalgorithmen

Lemma 22.2

- Die Rekursion
- i>0: $Opt(i,v) = \min_{(u,v)\in E} \{Opt(i-1,v), w(u,v) + Opt(i-1,u)\}$
- i=0: $Opt(0,s)= 0$ und $Opt(0,v)=\infty$ für $v\notin s$
- beschreibt die Länge eines kürzesten Weges von s nach v mit maximal i Kanten.

Beweis

- Wenn P genau i Kanten hat, so gilt $Opt(i,v) = w(u,v) + Opt(i-1,u)$, wobei u der vorletzte Knoten von P ist
- Außerdem gilt offensichtlich $Opt(i,v)\leq Opt(i-1,v)$
- Es folgt: $Opt(i,v) = \min_{(u,v)\in E} \{Opt(i-1,v), w(u,v) + Opt(i-1,u)\}$

Graphenalgorithmen

$d[i][v] = \mathrm{OPT}(i,v)$

Bellman-FordVersion1(G,s)

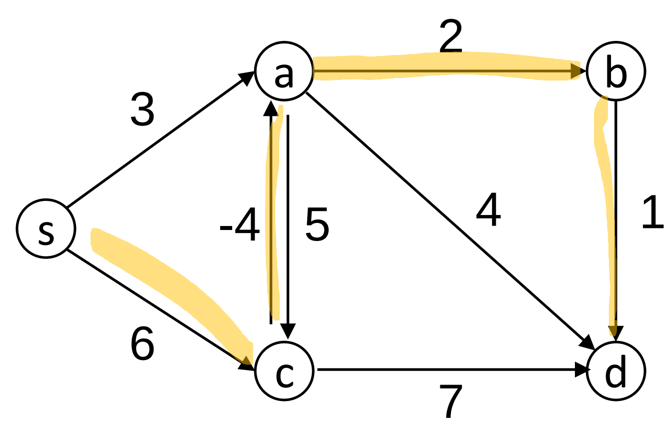
```
1. d = new array [1..|V|-1][1..|V|]
2. for each v∈V do d[0][v]=∞
3. d[0][s]=0
4. for i=1 to |V|-1 do
5.     for each v∈V do
6.         d[i][v] = d[i-1][v]
7.         for each u mit (u,v)∈E do
8.             if d[i][v]>d[i-1][u] + w(u,v) then
                   d[i][v] = d[i-1][u] + w(u,v)
9. return d
```

Graphenalgorithmen

Bellman-FordVersion1(G,s)

1. d = **new array** [1..|V|-1][1..|V|]
2. **for each** v∈V **do** d[0][v]=∞
3. d[0][s]=0
4. **for** i=1 **to** |V|-1 **do**
5. **for each** v∈V **do**
6. d[i][v] = d[i-1][v]
7. **for each** u mit (u,v)∈E **do**
8. **if** d[i][v]>d[i-1][u] + w(u,v) **then**
   d[i][v] = d[i-1][u] + w(u,v)
9. **return** d

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



Graphenalgorithmen

Bellman-FordVersion1(G,s)

1. d = **new array** [1..|V|-1][1..|V|]
2. **for each** v∈V **do** d[0][v]=∞
3. d[0][s]=0
4. **for** i=1 **to** |V|-1 **do**
5.   **for each** v∈V **do**
6.   d[i][v] = d[i-1][v]
7.   **for each** u mit (u,v)∈E **do**
8.   **if** d[i][v]>d[i-1][u] + w(u,v) **then**
                   d[i][v] = d[i-1][u] + w(u,v)
9. **return** d

$O(|V|^2)$

$O(|V|)$

$O(1)$

$O(|V|)$

$O(|V|^2)$

$O(|V|^2)$

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

— " —

$O(1)$

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

Graphenalgorithmen

Satz 22.3

Sei G ein Graph ohne negative Kreise. Algorithmus Bellman-FordVersion1 berechnet für jeden Knoten v aus G die Kosten eines kürzesten s-v-Pfads. Laufzeit der ersten Version des Algorithmus ist $O(|V|^2 |E|)$ und Speicherbedarf ist $O(|V|^2)$.

Beweis

- Nach Lemma 22.2 beschreibt die Rekursion Opt(i,v) = $\min_{(u,v)\in E}$ {Opt(i-1,v), w(u,v) + Opt(i-1,v)} mit Rekursionsabbruch Opt(0,s)= 0 und Opt(0,v)=∞ für v$\notin$s korrekt die Länge eines kürzesten s-v-Weges mit maximal i Kanten
- Die Einträge in Feld d werden nach dieser Rekursion berechnet
- Da es nach Lemma 22.1 in einem Graph ohne negativen Kreise immer einen kürzesten Weg mit mit maximal n-1 Kanten gibt, ist Opt(n-1,v) die Länge eines kürzesten Weges von s nach v.
- Laufzeit haben wir bereits analysiert und Speicherbedarf ist klar

Graphenalgorithmen

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

Graphenalgorithmen

Bellman-Ford(G,s)

```
1.  d = new array [1..|V|-1][1..|V|]
2.  for each v∈V do d[0][v]=∞
3.  d[0][s]=0
4.  for i=1 to |V|-1 do
5.      for each v∈V do
6.          min=d[i-1][v]
7.          for each (u,v)CIn[v] do
8.              if d[i-1][u]+w(u,v)<min then min=d[i-1][u]+w(u,v)
9.          d[i][v]=min
10. return d
```


Universität zu Köln

Graphenalgorithmen

Verbesserung des Speicherbedarfs

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

Beobachtung

- (1) d[v] ist immer die Länge irgendeines Weges von v nach t
- (2) nach i Runden ist d[v] höchstens so groß wie die Länge des kürzesten Weges mit i Kanten

Graphenalgorithmen

Bellman-Ford(G,s)

1. d = **new array** [1..|V|]
2. **for each** v∈V **do** d[v]=∞
3. d[s]=0
4. **for** i=1 **to** |V|-1 **do**
5.   **for each** v∈V **do**
6.   min=d[v]
7.   **for each** (u,v)∈In[v] **do**
8.   **if** d[u]+w(u,v)<min **then** min=d[u]+w(u,v)
9.   d[v]=min
10. **return** d

Graphenalgorithmen

Bellman-Ford(G,s)

1. d = **new array** [1..|V|]
2. **for each** $v \in V$ **do** $d[v]=\infty$
3. d[s]=0
4. **for** i=1 **to** |V|-1 **do**
5.   **for each** $v \in V$ **do**
6.   **for each** $(u,v) \in In[v]$ **do**
7.   **if** d[u]+w(u,v)<d[v] **then** d[v]=d[u]+w(u,v)
8. **return** d

Graphenalgorithmen

Bellman-Ford(G,s)

1. d = **new array** [1..|V|] — $O(|V|)$
2. **for each** v∈V **do** d[v]=∞ — $O(|V|)$
3. d[s]=0 — $O(1)$
4. **for** i=1 **to** |V|-1 **do** — $O(|V|)$
5.   **for each** v∈V **do** — $O(|V|^2)$
6.   **for each** (u,v)∈In[v] **do** — $O(|V| \cdot (|V|+|E|))$
7.   **if** d[u]+w(u,v)<d[v] **then** d[v]=d[u]+w(u,v) — $O(1)$
8. **return** d

$$O\left(|V| \cdot \sum_{v \in V} \left(1 + \mathrm{indeg}(v)\right)\right) = O(|V|^2 + |V| \cdot |E|)$$

$$O(|V| \cdot (|V| + |E|))$$

Graphenalgorithmen

Satz 22.4

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

Beweis

- Die Korrektheit folgt aus Satz 22.3 zusammen mit unserer Beobachtung
- Die Laufzeit haben wir bereits analysiert
- Hinweis: Wenn der Graph mind. $|V|$ Kanten hat, so ist die Laufzeit $O(|V|\ |E|)$

Graphenalgorithmen

Negative Kreise

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

Graphenalgorithmen

Zusammenfassung

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

Graphenalgorithmen

All Pairs Shortest Path (APSP)

- Eingabe: Gewichteter Graph G=(V,E)
- Ausgabe: Für jedes Paar von Knoten $u,v \in V$ die Distanz von u nach v sowie einen kürzesten Weg

Graphenalgorithmen

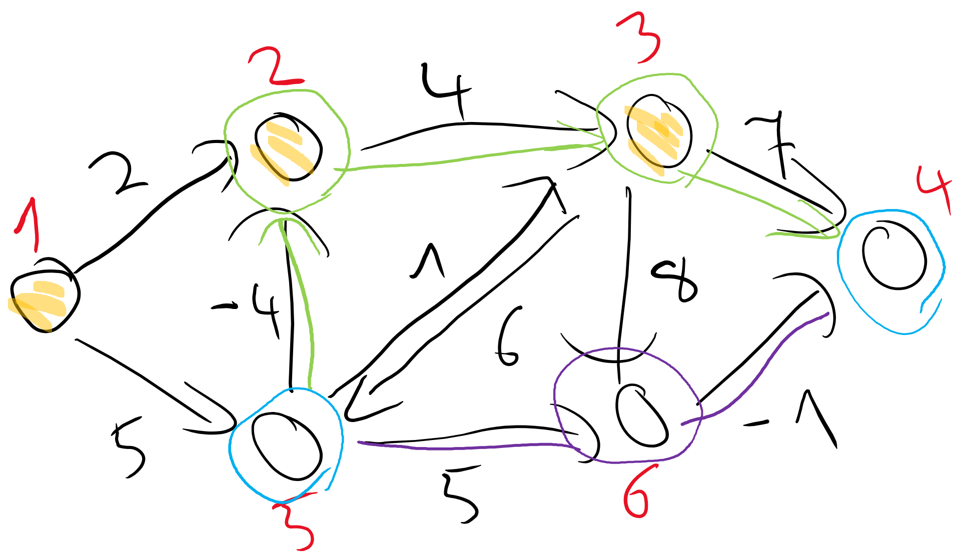
Eingabe APSP

- Matrix $W=(w_{ij})$ mit
- $w_{ij} = 0$, wenn $i=j$
- w_{ij} = Länge der gerichteten Kante (i,j), wenn $i \neq j$ und $(i,j) \in E$
- $w_{ij} = \infty$, wenn $i \neq j$ und $(i,j) \notin E$
- Annahme: Keine negativen Kreise

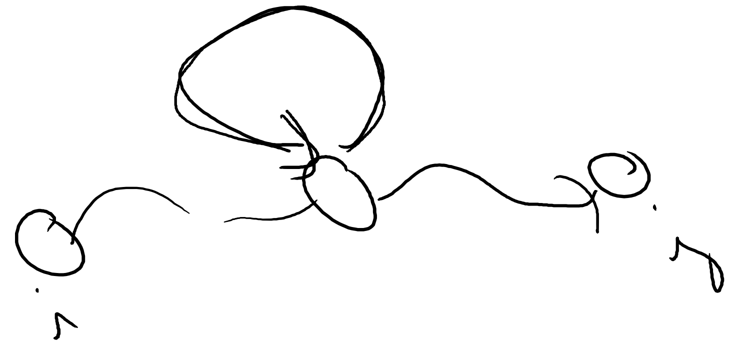
Graphenalgorithmen

Eine neue Rekursion

- Nummeriere Knoten von 1 bis n=|V|
- Betrachte kürzeste i-j-Wege, die nur über Knoten 1 bis k laufen

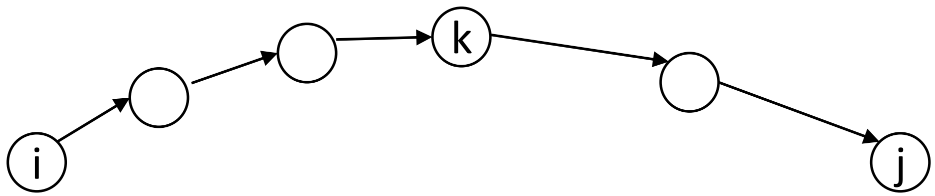


Graphenalgorithmen



Zur Erinnerung

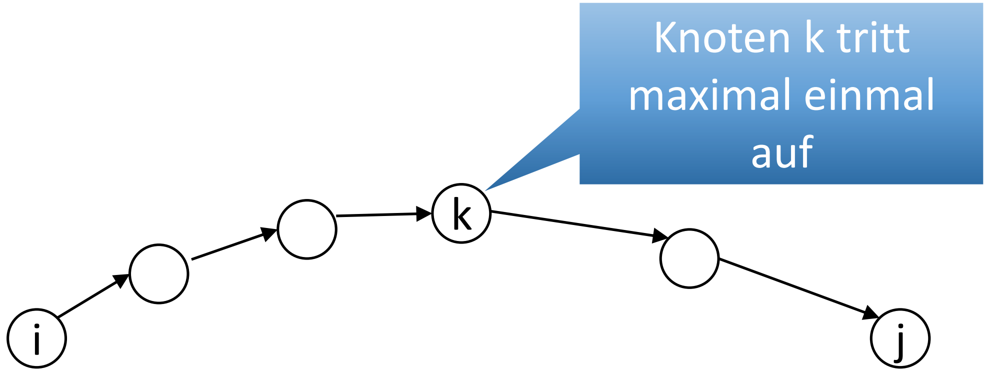
- Sei G ein Graph ohne negative Kreise und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 22.1)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus {1,…,k} läuft:



Graphenalgorithmen

Zur Erinnerung

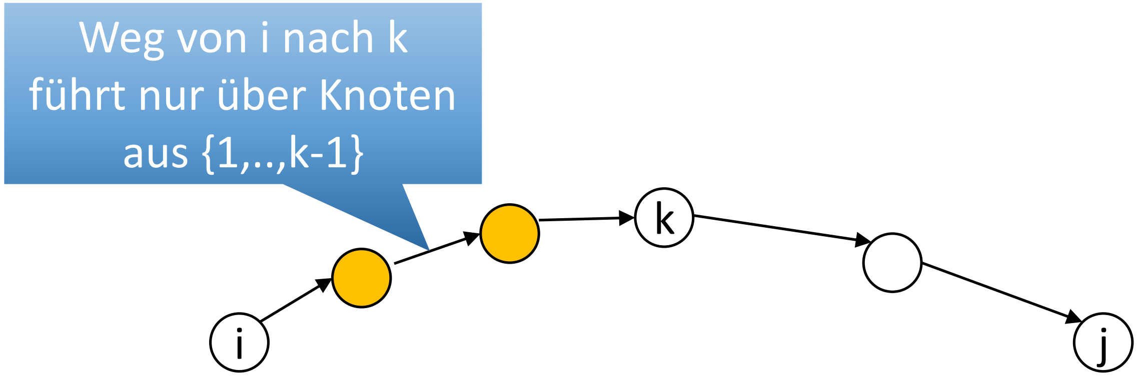
- Sei G ein Graph ohne negative Kreise und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 22.1)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus {1,…,k} läuft:



Graphenalgorithmen

Zur Erinnerung

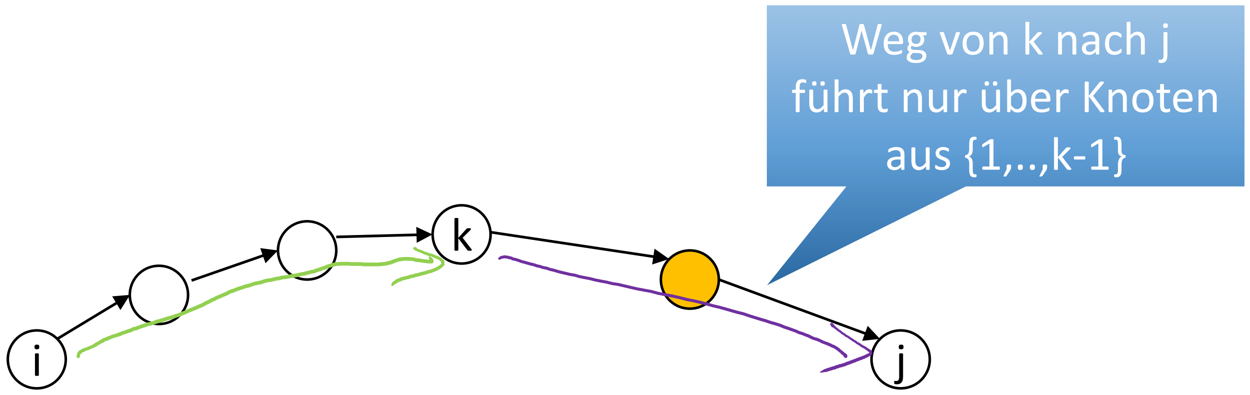
- Sei G ein Graph ohne negative Kreise und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 22.1)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus {1,…,k} läuft:



Graphenalgorithmen

Zur Erinnerung

- Sei G ein Graph ohne negative Kreise und sei j von i aus erreichbar. Dann gibt es einen kürzesten i-j-Weg, der keinen Knoten doppelt benutzt. (Lemma 22.1)
- Wir können also annehmen, dass jeder Knoten in jedem Weg maximal einmal vorkommt
- Betrachte i-j-Weg, der nur über Knoten aus {1,…,k} läuft:

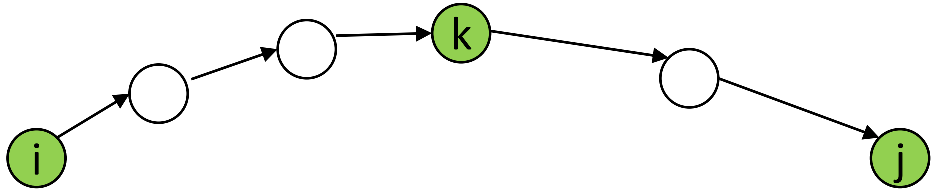


Graphenalgorithmen

Die Rekursion

- Kürzester i-j-Weg über Knoten aus {1,…,k} ist
- (a) kürzester i-j-Weg über Knoten aus {1,…,k-1} oder
- (b) kürzester i-k-Weg über Knoten aus {1,…,k-1} gefolgt von kürzestem k-j-Weg über Knoten aus {1,…,k-1}

Fall b:



Graphenalgorithmen

Die Rekursion

- Sei $d_{ij}^{(k)}$ die Länge eines kürzesten Weges von i nach j, der nur über Knoten aus {1,…,k} verläuft
- $d_{ij}^{(k)} = \begin{cases} w_{ij}, & \text{falls } k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}), & \text{falls } k > 0 \end{cases}$

Graphenalgorithmen

Floyd-Warshall(W,n)

1. Reserviere Speicher für Felder $D^{(k)}$
2. $D^{(0)} = W$
3. **for** k=1 **to** n **do**
4.   **for** i=1 **to** n **do**
5.   **for** j=1 **to** n **do**
6. $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
7. **return** $D^{(n)}$

Graphenalgorithmen

Satz 22.5

Sei G=(V,E) ein Graph ohne negative Kreise. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $O(|V|^3)$ Zeit.

Beweis

- Die Laufzeit folgt sofort, da 3 ineinander geschachtelte Schleifen jeweils von 1 bis $|V|=n$ durchlaufen werden.
- Korrektheit per Induktion über k. Z.z. die Matrix $D^{(k)}$ enthält die kürzeste Entfernung zwischen allen Paaren von Knoten, wenn die Wege nur über die Knoten {1,..,k} verlaufen dürfen.
- Induktionsanfang: Die Matrix $D^{(0)}$ ist gleich der Eingabematrix W und enthält somit alle Wege im Graphen, die keinen Zwischenknoten benutzen.

Graphenalgorithmen

Satz 22.5

Sei G=(V,E) ein Graph ohne negative Kreise. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in $O(|V|^3)$ Zeit.

Beweis

- Induktionsvoraussetzung: Die Matrix $D^{(k-1)}$ erfüllt die Aussage.
- Induktionsschluss: Die Matrix $D^{(k)}$ soll nun die kürzesten Wege zwischen Knotenpaaren enthalten, die nur über die Knoten {1,..,k} verlaufen.
- Da G keine negativen Kreise hat, gibt es immer einen kürzesten Weg, der jeden Knoten nur maximal einmal besucht.
- Verläuft der kürzeste Weg von i nach j nun über Knoten k, so ist seine Länge nach Induktionsvoraussetzung genau $d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

Graphenalgorithmen

Satz 22.5

Sei G=(V,E) ein Graph ohne negative Kreise. Dann berechnet der Algorithmus von Floyd-Warshall die Entfernung zwischen jedem Knotenpaar in O(|V|³) Zeit.

Beweis

- Ansonsten verläuft er nicht über k und somit nur über die Knoten {1,..,k-1}. Damit ist seine Länge $d_{ij}^{(k-1)}$.
- Somit wird die Länge durch die Rekursion korrekt berechnet.

Graphenalgorithmen

Aufrechterhalten der kürzesten Wege:

- Konstruiere Vorgängermatrix Π
- Dazu konstruiere Sequenz $\Pi^{(0)},\ldots,\Pi^{(n)}$ mit $\Pi = \Pi^{(n)}$
- $\Pi^{(k)}$ ist Vorgängermatrix zu $D^{(k)}$
- $\pi_{ij}^{(k)}$ ist Vorgänger von Knoten j auf dem kürzesten Weg von Knoten i über Knoten aus $\{1,\ldots,k\}$
- Die Startmatrix:
- $\pi_{ij}^{(0)} = \begin{cases} \text{NIL}, & \text{falls } i = j \text{ oder } w_{ij} = \infty \\ i, & \text{falls } i \neq j \text{ und } w_{ij} < \infty \end{cases}$

Graphenalgorithmen

Aufrechterhalten der kürzesten Wege:

- Konstruiere Vorgängermatrix Π
- Dazu konstruiere Sequenz $\Pi^{(0)},\ldots,\Pi^{(n)}$ mit $\Pi = \Pi^{(n)}$
- $\Pi^{(k)}$ ist Vorgängermatrix zu $D^{(k)}$
- $\pi_{ij}^{(k)}$ ist Vorgänger von Knoten j auf dem kürzesten Weg von Knoten i über Knoten aus {1,…,k}
- Das Aktualisieren:
- $$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)}, & \text{falls } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)}, & \text{falls } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Graphalgorithmen

SSSP (pos. Kantengewichte)

- Dijkstra; Laufzeit $O((|V|+|E|) \log |V|)$

SSSP (allgemeine Kantengewichte)

- Bellman-Ford; Laufzeit $O(|V|^2+|V|\,|E|)$

APSP (allgemeine Kantengewichte, keine negativen Kreise)

- Floyd-Warshall; Laufzeit $O(|V|^3)$

Referenzen

- T. Cormen, C. Leisserson, R. Rivest, C. Stein. Introduction to Algorithms. The MIT press. Second edition, 2001.