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

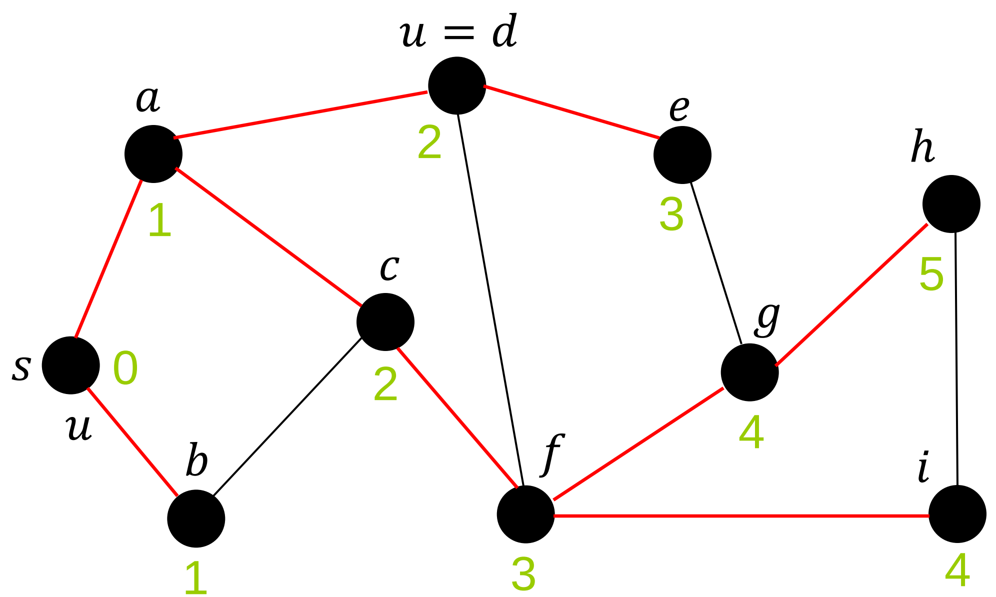
SSSP in ungewichteten Graphen (Breitensuche)

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

Wiederholung: Breitensuche

BFS(G, s)

„initialisiere BFS“
while $Q \neq \emptyset$ **do**
    $u \leftarrow \mathrm{head}[Q]$
    for each $v \in \mathrm{Adj}[u]$ **do**
        if $\mathrm{color}[v] = \mathrm{weiß}$ **then**
            $\mathrm{color}[v] \leftarrow \mathrm{grau}$
            $d[v] \leftarrow d[u] + 1;\ \pi[v] \leftarrow u$
            $\mathrm{enqueue}(Q, v)$
    $\mathrm{dequeue}(Q)$
    $\mathrm{color}[u] \leftarrow \mathrm{schwarz}$



Graphalgorithmen

Kürzeste Wege in gewichteten Graphen

- $G = (V, E)$
- w: $E \to \mathbb{R}$; $w(e)$ ist Länge der Kante e; $w(u, v)$ ist Länge der Kante (u, v)

- Für Pfad $p = < v_0, v_1, \dots, v_k >$ ist Pfadlänge gegeben durch

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

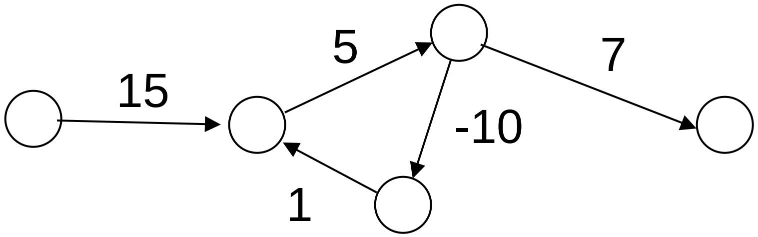
- $\delta(u, v) = \min_{\text{Pfade } p \text{ von } u \text{ nach } v} w(p)$, falls es Pfad von u nach v gibt

- $\delta(u, v) = \infty$, sonst

Graphalgorithmen

Negative Kantengewichte

- Manchmal hat man Instanzen mit negativen Kantenlängen

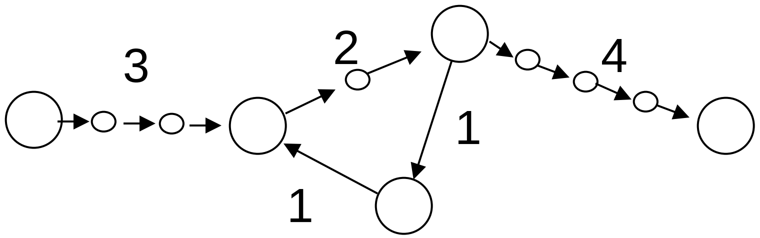


- Bei ungerichteten Graphen kann man Kante immer wieder vorwärts und rückwärts durchlaufen
- Kürzester Pfad u.U. nicht wohldefiniert
- Erstmal nichtnegative Kantengewichte

Graphalgorithmen

Dijkstras Algorithmus

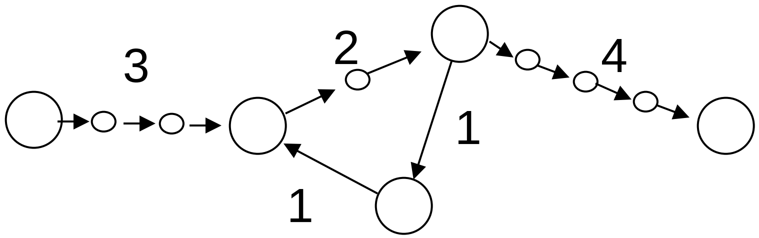
- Graph in Adjazenzlistendarstellung
- Keine negativen Kantenlängen
- Überträgt Idee der Breitensuche auf gewichtete Graphen
- Erster Ansatz: Ersetze Kantenlängen durch mehrfache Kanten



Graphalgorithmen

Dijkstras Algorithmus

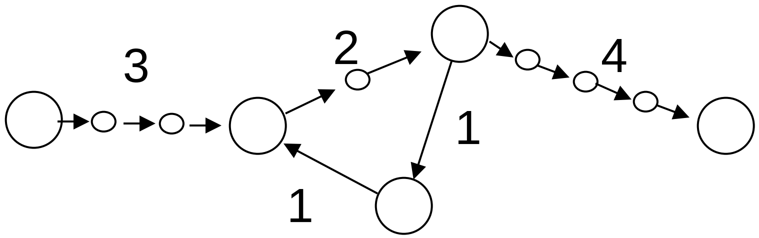
- Erste Idee: Ersetze Kantenlängen durch mehrfache Kanten
- Probleme: Langsam bei großen Kantenlängen; nur ganzzahlige Längen
- Annahme: Zunächst ganzzahlige Längen.
- Idee: Simuliere Breitensuche effizient



Graphalgorithmen

Dijkstras Algorithmus

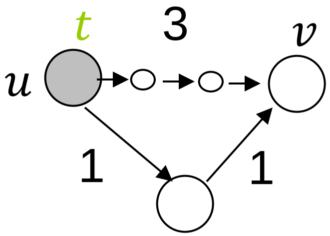
- Erste Idee: Ersetze Kantenlängen durch mehrfache Kanten
- Probleme: Langsam bei großen Kantenlängen; nur ganzzahlige Längen
- Zunächst ganzzahlige Längen. Idee: Simuliere Breitensuche effizient
- Aufgabe: Bestimme für jeden Knoten den Zeitpunkt, zu dem er entdeckt wird



Graphalgorithmen

Dijkstras 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

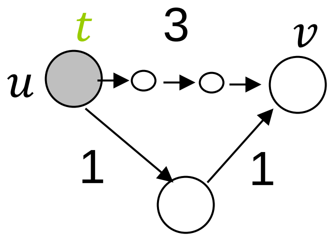


Wird zu Zeitpunkt $\leq t + 3$ entdeckt

Graphalgorithmen

Simulation der Breitensuche

- Simuliere die Breitensuche schrittweise, wobei jeder Zeitschritt der Abarbeitung einer Entfernungsebene entspricht
(alle Knoten mit Distanz k werden bearbeitet)
- Priorität eines Knotens ist der nächste Zeitpunkt, an dem die Breitensuche diesen Knoten erreicht
- Im Laufe des Algorithmus können sich die Prioritäten verändern
- Startet z.B. der Algorithmus die simulierte Breitensuche im Graph unten bei u, so wird die Priorität von v zunächst auf 3 gesetzt. Wird der untere Knoten entdeckt, so wird sie auf 2 reduziert



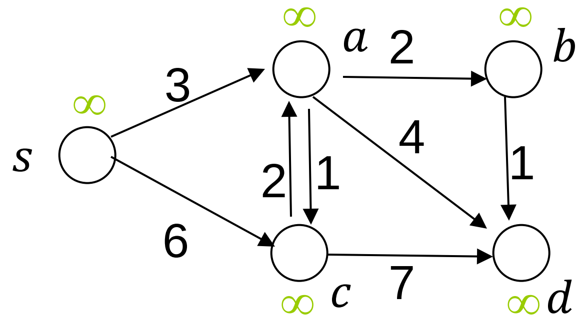
Wird zu Zeitpunkt $\leq t+3$ entdeckt

Graphalgorithmen

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

BreitensucheSimulation(G, w, s)

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



Graphalgorithmen

BreitensucheSimulation(G, w, s)

Initialisiere Simulation

Füge $(s, \mathrm{prio}[s])$ mit Priorität $\mathrm{prio}[s]$ in Prioritätenschlange Q ein

while $Q \neq \emptyset$ **do**

    $(u, \mathrm{prio}[u]) \leftarrow \mathrm{ExtractMin}(Q)$

    if $\mathrm{color}[u] = \mathrm{weiß}$ **then**

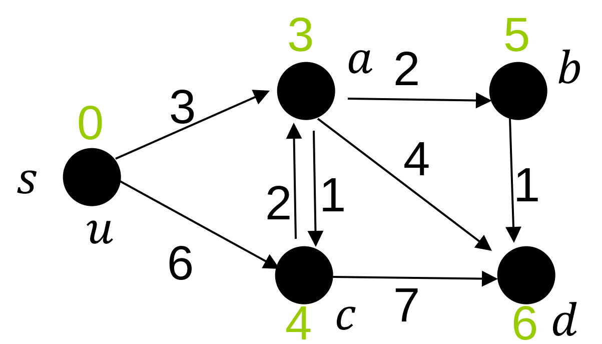
        $\mathrm{color}[u] \leftarrow \mathrm{schwarz}$

        $d[u] \leftarrow \mathrm{prio}[u]$

        for each $v \in \mathrm{Adj}[u]$ **do**

            $\mathrm{prio}[v] \leftarrow d[u] + w(u, v)$

            Füge $(v, \mathrm{prio}[v])$ mit Priorität $\mathrm{prio}[v]$ in Q ein



Graphalgorithmen

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

Dijkstras Algorithmus(G, w, s)

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

$d[u] = \infty$ 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

Prioritätenschlangen mit DecreaseKey

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

Implementierung mit AVL-Bäumen

- Lösche v (zusätzliches Array verwenden)
- Füge v mit Priorität p in Prioritätenschlange ein

Laufzeit

- Zeile 2 und 4 jeweils $\mathbf{O}(|V| \log|V|)$
- Zeile 10 wird höchstens einmal für jede Kante aufgerufen, also $\mathbf{O}((|\boldsymbol{V}| + |E|) \log|V|)$
- Laufzeit insgesamt $\mathbf{O}\big((|V| + |E|) \log|V|\big)$

Graphalgorithmen

Lemma 49

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

Beweis

Annahme: Es gäbe einen kürzeren Weg $< v_i, u_1, \ldots, u_L, v_j >$ von v_i nach v_j.
Dann wäre $< v_1, \ldots, v_i, u_1, \ldots, u_L, v_j, \ldots, v_k >$ kürzer als $< v_1, \ldots, v_k >$. Widerspruch.

Graphalgorithmen

Lemma 50

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 45)
- 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 51

Zu jedem Ausführungszeitpunkt von Dijkstras 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 51 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 $(m+1)$ste 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 $\min\{d[v], d[u]+w(u,v)\}$.

Graphalgorithmen

Lemma 51

Zu jedem Ausführungszeitpunkt von Dijkstras Algorithmus gilt für jeden Knoten w:

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

Beweis

- (I.S.) Betrachte $(m+1)$ste 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 $d[u]+w(u,v) \geq \delta(s,u)+w(u,v) \geq \delta(s,v)$ nach Lemma 50. Somit gilt auch hier das Lemma.

Graphalgorithmen

Satz 52

Wenn wir Dijkstras 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

- Ist ein Knoten nicht erreichbar, so gilt wegen Lemma 51 $d[u] = \delta(s, u)$
- Es genügt also, den Satz für erreichbare Knoten zu zeigen

Graphalgorithmen

Satz 52

Wenn wir Dijkstras 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 erreichbare 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.
- <u>Annahme:</u> 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!)

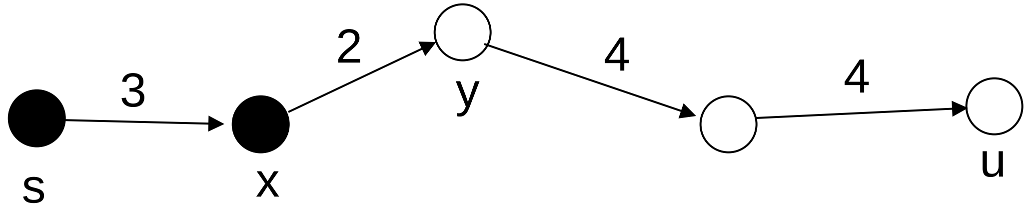
Graphalgorithmen

Satz 52

Wenn wir Dijkstras 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

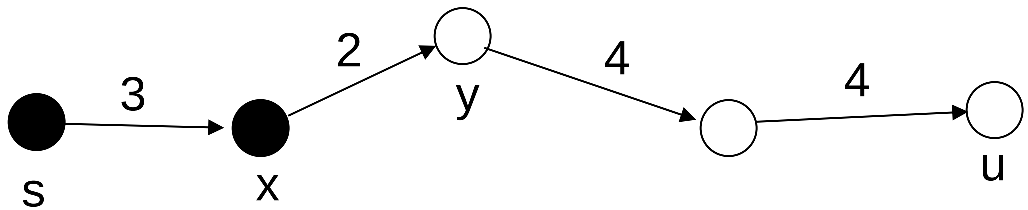
- Sei nun y der erste weiße Knoten auf einem kürzesten Weg von s nach u und x sein Vorgänger.



Graphalgorithmen

Beweis

- Es gilt $d[x] = \delta(s, x)$ nach Wahl von u.
- In Zeile 9 wird nun $d[y]$ auf $\min\{d[y], d[x] + w(x, y)\}$ gesetzt. Nach Lemma 49 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 auf diesen Wert gesetzt (wegen Lemma 51).
- Da die Kantengewichte nichtnegativ sind, folgt $\delta(s, y) \le \delta(s, u)$ und somit nach Lemma 51 $d[y] = \delta(s, y) \le \delta(s, u) \le d[u]$.
- Da aber u von ExtractMin aus der Prioritätenschlange entfernt wurde, gilt $d[u] \le d[y]$ und somit $d[y] = \delta(s, y) = \delta(s, u) = d[u]$. Widerspruch!



Graphalgorithmen

Zusammenfassung (Dijkstras Algorithmus)

- Der Algorithmus von Dijkstra kann dazu genutzt werden, um das SSSP Problem in gewichteten Graphen mit nichtnegativen Kantengewichten zu lösen
- Dijkstras Algorithmus kann als Erweiterung der Breitensuche interpretiert werden
- Die Laufzeit von Dijkstras Algorithmus ist $\mathbf{O}\big((|V| + |E|)\log|V|\big)$, wenn die Prioritätenschlange als AVL-Baum implementiert wird

Graphalgorithmen

Und was macht man bei negativen Kantengewichten?