Eidgenössische
Technische Hochschule
Zürich

Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Federal Institute of Technology at Zurich

Departement Informatik
Markus Püschel David Steurer Peter Widmayer
Chih-Hung Liu

17. Dezember 2017

Algorithmen & Datenstrukturen Blatt 12 HS 17

Lösung 12.1 *Algorithmus von Dijkstra ausführen.*

a) Die folgende Tabelle gibt für jeden Schritt von Dijkstras Algorithmus die aktuellen oberen Schranken der Distanzen $d[\cdot]$ zu den einzelnen Knoten sowie die Menge S der Knoten an, deren Distanzwerte bereits korrekt berechnet wurden. Für diese Knoten ist auch der jeweilige Vorgänger auf einem kürzesten Weg angegeben. Man beachte, dass u.U. auch schon vorläufige Vorgänger $p[\cdot]$ berechnet wurden, die hier der Übersicht halber nicht dargestellt sind. Fett gedruckte Distanzwerte bedeuten eine Veränderung gegenüber dem vorigen Schritt.

	S	$d[s]$	$p[s]$	$d[v]$	$p[v]$	$d[w]$	$p[w]$	$d[x]$	$p[x]$	$d[y]$	$p[y]$	$d[t]$	$p[t]$
1.	$\emptyset$	0		∞		∞		∞		∞		∞	
2.	$\{s\}$	0	null	**20**		∞		**2**		∞		∞	
3.	$\{s,x\}$	0	null	**10**		∞		2	s	**3**		∞	
4.	$\{s,x,y\}$	0	null	10		**20**		2	s	3	x	**6**	
5.	$\{s,x,y,t\}$	0	null	10		20		2	s	3	x	6	y
6.	$\{s,x,y,t,v\}$	0	null	10	x	**15**		2	s	3	x	6	y
7.	$\{s,x,y,t,v,w\}$	0	null	10	x	15	v	2	s	3	x	6	y

b) Nun ergibt sich der folgende Ablauf:

	S	$d[s]$	$p[s]$	$d[v]$	$p[v]$	$d[w]$	$p[w]$	$d[x]$	$p[x]$	$d[y]$	$p[y]$	$d[t]$	$p[t]$
1.	$\emptyset$	0		∞		∞		∞		∞		∞	
2.	$\{s\}$	0	null	**20**		∞		**2**		∞		∞	
3.	$\{s,x\}$	0	null	**10**		∞		2	s	**1**		∞	
4.	$\{s,x,y\}$	0	null	10		**18**		2	s	1	x	**4**	
5.	$\{s,x,y,t\}$	0	null	10		18		2	s	1	x	4	y
6.	$\{s,x,y,t,v\}$	0	null	10	x	**15**		2	s	1	x	4	y
7.	$\{s,x,y,t,v,w\}$	0	null	10	x	15	v	2	s	1	x	4	y

Wie man sieht, sind die endgültigen Distanzwerte und entsprechenden Vorgängerknoten korrekt berechnet. Es gibt also Distanzgraphen mit negativen Kantengewichten, auf denen der Algorithmus von Dijkstra korrekt arbeitet.

c) Der Algorithmus läuft jetzt wie folgt ab:

	S	$d[s]$	$p[s]$	$d[v]$	$p[v]$	$d[w]$	$p[w]$	$d[x]$	$p[x]$	$d[y]$	$p[y]$	$d[t]$	$p[t]$
1.	$\emptyset$	0		∞		∞		∞		∞		∞	
2.	$\{s\}$	0	null	**20**		∞		**2**		∞		∞	
3.	$\{s,x\}$	0	null	**10**		∞		2	s	**1**		∞	
4.	$\{s,x,y\}$	0	null	10		**18**		2	s	1	x	**4**	
5.	$\{s,x,y,t\}$	0	null	10		18		2	s	1	x	4	y
6.	$\{s,x,y,t,v\}$	0	null	10	x	**15**		2	s	**0**	v	4	y
7.	$\{s,x,y,t,v,w\}$	0	null	10	x	15	v	2	s	0	v	4	y

Jetzt arbeitet Dijkstras Algorithmus nicht mehr korrekt. Der erste Fehler passiert in Schritt 4, da dort der Knoten y gewählt wird. Folglich wird auch die Distanz zu t falsch berechnet.

Lösung 12.2 *Kanten auf kürzesten Pfaden.*

a) Die Aussage ist falsch. Als Gegenbeispiel betrachte man einen Graphen $G = (V, E)$ mit $V = \{s, t, z\}$ und $E = \{(s, t), (s, z), (z, t)\}$, in dem alle Kanten Gewicht 1 haben. Dann haben die kürzesten Pfade von s nach z bzw. von z nach t Länge 1, also hat ihre Konkatenation Länge 2 und verläuft von s nach t. Aber ein kürzester Pfad von s nach t hat Länge 1, da es eine Kante mit diesem Gewicht gibt.

b) Die Aussage ist korrekt. Wenn irgendein Teilpfad von P nicht kürzestmöglich wäre, dann könnte man diesen Teilpfad in P durch einen kürzeren Pfad ersetzen. Das bedeutete dann aber, dass auch P nicht kürzestmöglich wäre, was ein Widerspruch ist.

c) Sei $d(x, y)$ die Länge eines kürzesten x-y-Pfades in G und sei $e = \{u, v\}$ eine beliebige Kante des Graphen. Wir analysieren den Term $T(e) = \min\{d(s, u)+w(e)+d(v, t),\ d(s, v)+w(e)+d(u, t)\}$.

Da G ungerichtet ist, ist $T(e)$ die Länge eines kürzesten *s-t-Weges* in welchem e benützt werden *muss* (entweder indem zuerst u und dann – via e – v besucht wird, oder umgekehrt). Deshalb gilt $T(e) \geq d(s, t)$ mit Gleichheit genau dann wenn e Kante eines kürzesten *s-t-Pfades* ist.

d) Um zu bestimmen, ob e Kante eines kürzesten s-t-Pfades ist, genügt es also, die fünf Abstände $d(s, u), d(s, v), d(u, t), d(v, t)$ und $d(s, t)$ zu kennen. Ein völlig naiver Algorithmus könnte also über alle Kanten e iterieren, alle Distanzwerte mit drei Aufrufen von Dijkstras Algorithmus (jeweils von s, u und v ausgehend) berechnen und die Kante genau dann ausgeben, wenn $T(e) = d(s, t)$ gilt.

Es geht aber viel schneller. Dazu starten wir Dijkstras Algorithmus einmal ab dem Startknoten s und einmal ab dem Knoten t. Da der Graph ungerichtet ist, gelten $d(u, t) = d(t, u)$ und $d(v, t) = d(t, v)$. Das heisst, nach dieser Vorberechnung kann in konstanter Zeit entschieden werden, ob eine gegebene Kante auf einem kürzesten Pfad von s nach t liegt. Wir iterieren also über alle Kanten e, testen für jede Kante e, ob $T(e) = d(s, t)$ gilt, und geben bei Gleichheit die Kante aus.

e) Ein zweimaliges Anwenden von Dijkstras Algorithmus dauert $\mathcal{O}((|V|+|E|)\log|V|)$, unter Verwendung von Fibonacci-Heaps sogar nur $\mathcal{O}(|V|\log|V|+|E|)$. Danach kann jede der $|E|$ Kanten in Zeit $\mathcal{O}(1)$ getestet werden. Insgesamt ergibt sich damit eine Laufzeit von $\mathcal{O}(|V|\log|V|+|E|)$.

f) Jetzt ist $T(e) = d(s,u)+w(e)+d(v,t)$, da die Richtung von e berücksichtigt werden muss. Zur Berechnung von $d(v,t)$ für alle Knoten $v \in V$ drehen wir einmalig temporär alle Kantenrichtungen um, d.h., wir ersetzen jede Kante (x,y) durch ihre *Rückwärtskante* (y,x). Führt man auf diesem modifizierten Graphen Dijkstras Algorithmus ausgehend vom Startknoten t aus, dann entsprechen die berechneten Distanzwerte zu jedem Knoten v exakt den Distanzen von v nach t im Ausgangsgraphen.

Lösung 12.3 *Minimaler Teilgraph.*

Das Problem, einen minimalen Teilgraphen G' zu bestimmen, welcher eine Menge von Terminalen $T \subset V$ verbindet, ist bekannt als das Steinerbaumproblem und ist eines der 21 klassischen NP-vollständigen Probleme. Hier betrachten wir einen Spezialfall mit $|T| = 3$ und definieren $d_G(u,v)$ als die Länge eines kürzesten u-v-Pfades in G.

a) Zuerst halten wir fest, dass G' keinen Kreis enthalten kann: Andernfalls erhalten wir durch Löschen einer der Kreiskanten einen Teilgraphen $G' \subsetneq G$ kleineren Gewichtes, welcher nach wie vor a, b und c verbindet. Mit dem gleichen Argument kann G' kein Blatt $u \notin \{a,b,c\}$ enthalten, da wir sonst u samt inzidenter Kante löschen können.

   Die einzig verbleibenden Blattkandidaten sind also a, b und c. Da G' ein Baum ist, gilt $|V'|-1 = |E'|$ und $2|E'| = \sum_{v' \in V'} \deg(v')$. Deshalb gibt es in G' entweder genau einen Knoten $v \notin \{a,b,c\}$ vom Grad $\deg(v) = 3$, welcher via drei kantendisjunkten Pfade mit den drei Blättern $\{a,b,c\}$ verbunden ist, oder einen Knoten $v \in \{a,b,c\}$ vom Grad $\deg(v) = 2$, welcher via zwei kantendisjunkten Pfade mit den zwei Blättern $\{a,b,c\} \setminus \{v\}$ verbunden ist.

b) Um das Gewicht eines minimalen Teilgraphen G' zu bestimmen, genügt es nach a), den zugehörigen Knoten v zu finden und die Summe $d(v,a)+d(v,b)+d(v,c)$ zu berechnen. Insbesondere reicht es, alle Knoten $v' \in V$ zu testen, und sich laufend die minimale Summe $d(v',a)+d(v',b)+d(v',c)$ zu merken.

   Naiverweise könnte man dazu von jedem Knoten v' einmal Dijkstra laufen zu lassen. Wie in der vorigen Aufgabe gibt es aber einen effizienteren Weg: Um für alle $v' \in V$ die Terme $d(v',a), d(v',b)$ und $d(v',c)$ zu bestimmen, reicht es, Dijkstra je einmal von a, von b und von c laufen zu lassen.

   Laufzeit: Unser Ausgangsgraph $G = (V,E)$ besitzt $|V| = k+\ldots+2+1 = \frac{k(k+1)}{2} \in \mathcal{O}(k^2)$ Knoten. Da jeder Knoten v Grad $\deg(V) \in \{2,4,6\}$ hat, gilt auch $|E| \in \mathcal{O}(k^2)$. Damit dauert ein dreimaliges Anwenden von Dijkstra's Algorithmus $\mathcal{O}(k^2 \log k)$. Danach muss für jeden der $|V|$ Knoten die Summe der (durch Dijkstra vorberechneten) Distanzen zu a, b und c bestimmt werden. Insgesamt ergibt sich eine Laufzeit von $\mathcal{O}(k^2 \log k)$.

Anmerkung: Das Steinerbaumproblem für drei Terminale ist auch in der euklidischen Ebene lösbar. In diesem Falle besteht V aus der unendlichen Menge aller Punkte der Ebene $\mathbb{R}^2$. Unser Punkt v aus Teilaufgabe a) entspricht in diesem Falle dem sogenannten *Fermatpunkt*.