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

12. Januar 2018

Algorithmen & Datenstrukturen Blatt 13 HS 17

Lösung 13.1 *Kürzeste Pfade bei negativen Kantengewichten.*

a) Der kürzeste Pfad von Knoten 1 zu Knoten 6 ist $(1, 3, 5, 2, 6)$ und hat Länge $5-4+1+1 = 3$.

b) Dijkstra's Algorithmus findet den Pfad $(1, 2, 6)$ der Länge 4. Der erste Fehler geschieht bereits nachdem Knoten 1 abgearbeitet wurde. Die Mengen sind zu diesem Zeitpunkt $A = \{1\}$, $B = \{2, 3\}$, $C = \{4, 5, 6\}$. Zu Knoten 2 ist zu diesem Zeitpunkt ein Pfad der Länge 3 bekannt, zu Knoten 3 ist ein Pfad der Länge 5 bekannt. Dijkstra's Algorithmus übernimmt daher Knoten 2 in A, was der Annahme entspricht, dass der kürzeste Pfad zu Knoten 2 bereits gefunden wurde. Dies ist ein Fehler, da es über die negative Kante auch einen Pfad der Länge 2 zu Knoten 2 gibt, nämlich $(1, 3, 5, 2)$.

c) Hier bietet sich der Algorithmus von Bellman-Ford an, der eine Laufzeit von $\mathcal{O}(nm)$ besitzt.

d) Jede der folgenden Tabellen gehört zu einem festen Wert von $k \in \{0, 1, 2, 3, 4, 5, 6\}$ und enthält die Pfadlängen aller kürzesten Pfade, die über Knoten aus der Menge $\{0, \ldots, k\}$ führen. Dass es keine negativen Zyklen gibt, erkennt man an den Diagonaleinträgen der (letzten) Tabelle: Gäbe es einen negativen Zyklus, so wäre dort ein Eintrag negativ.

von \ nach	1	2	3	4	5	6
1	0	3	5	∞	∞	∞
2	1	0	4	∞	4	1
3	∞	∞	0	1	-4	∞
4	∞	∞	∞	0	5	∞
5	∞	1	∞	2	0	∞
6	∞	∞	∞	∞	2	0

$k = 0$

von \ nach	1	2	3	4	5	6
1	0	3	5	∞	∞	∞
2	1	0	4	∞	4	1
3	∞	∞	0	1	-4	∞
4	∞	∞	∞	0	5	∞
5	∞	1	∞	2	0	∞
6	∞	∞	∞	∞	2	0

$k = 1$

von \ nach	1	2	3	4	5	6
1	0	3	5	∞	*7*	*4*
2	1	0	4	∞	4	1
3	∞	∞	0	1	-4	∞
4	∞	∞	∞	0	5	∞
5	*2*	1	*5*	2	0	*2*
6	∞	∞	∞	∞	2	0

$k = 2$

von \ nach	1	2	3	4	5	6
1	0	3	5	*6*	*1*	4
2	1	0	4	*5*	*0*	1
3	∞	∞	0	1	-4	∞
4	∞	∞	∞	0	5	∞
5	2	1	5	2	0	2
6	∞	∞	∞	∞	2	0

$k = 3$

von \ nach	1	2	3	4	5	6
1	0	3	5	6	1	4
2	1	0	4	5	0	1
3	∞	∞	0	1	-4	∞
4	∞	∞	∞	0	5	∞
5	2	1	5	2	0	2
6	∞	∞	∞	∞	2	0
$k = 4$						

von \ nach	1	2	3	4	5	6
1	0	*2*	5	*3*	1	*3*
2	1	0	4	*2*	0	1
3	*-2*	*-3*	0	*-2*	-4	*-2*
4	*7*	*6*	*10*	0	5	*7*
5	2	1	5	2	0	2
6	*4*	*3*	*7*	*4*	2	0
$k = 5$						

von \ nach	1	2	3	4	5	6
1	**0**	2	5	3	1	3
2	1	**0**	4	2	0	1
3	-2	-3	**0**	-2	-4	-2
4	7	6	10	**0**	5	7
5	2	1	5	2	**0**	2
6	4	3	7	4	2	**0**
$k = 6$						

Lösung 13.2 *Eigenschaften minimaler Spannbäume.*

a) Wir zeigen, dass jeder gewichtete, ungerichtete, zusammenhängende Graph G mit einer endlichen Anzahl Knoten n und einer endlichen Anzahl Kanten m mindestens einen Spannbaum T enthält. Weil die Anzahl der Spannbäume in jedem endlichen Graph endlich ist, folgt daraus dass es auch einen Minimalen Spannbaum gibt.

Wir zeigen die Existenz von T in G mit Induktion über n.

Verankerung: Wenn $n = 1$, dann ist der Graph bestehend aus genau einem Knoten bereits ein Spannbaum.

Hypothese: Jeder gewichtete, ungerichtete, zusammenhängende Graph mit höchstens $n-1$ Knoten enthält einen Spannbaum.

Schritt ($n - 1 \longmapsto n$)*:* Betrachte den induzierten Graph $G' = G[V \setminus \{v\}]$ ohne einen beliebigen Knoten v. G' besteht aus k Zusammenhangskomponenten $G'_1, \ldots, G'_k$ (d.h., für alle $1 \leq i \leq k$ ist G'_i zusammenhängend und es gibt *keine* Kante (u, w) mit $u \in G'_i$ und $w \in G'_j$ für $i \neq j$). Per Induktions-Hypothese enthält jede Zusammenhangskomponente G'_i einen Spannbaum T'_i. Ausserdem gibt es eine Kante $e_i = (v, u_i)$ mit $u_i \in G'_i$, weil G selbst auch zusammenhängend war. $T = \bigcup_{1 \leq i \leq k} T'_i \cup \{e_i\}$ ist deshalb ein Spannbaum für G. ∎

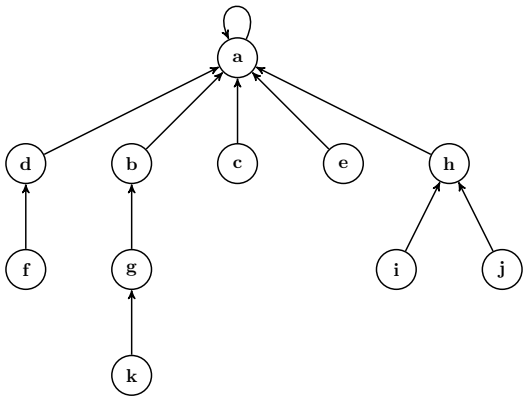
b) Sei $G = (V, E, w)$ ein ungerichteter Graph, bei dem keine zwei Kanten $e, e' \in E$ das gleiche Gewicht haben. Angenommen, G hätte zwei minimale Spannbäume $T_1 = (V, E_1)$ und $T_2 = (V, E_2)$. Die Kanten aus $E_1 \cap E_2$ kommen sowohl in T_1 als auch in T_2 vor. Seien $E'_1 = E_1 \backslash E_2$ und $E'_2 = E_2 \backslash E_1$ die Kanten, die ausschliesslich in T_1 bzw. in T_2 enthalten sind. Da die Gewichte aller Kanten paarweise verschieden ist, enthält die Menge $E'_1 \cup E'_2$ eine eindeutig bestimmte Kante $e^\perp$ minimalen Gewichts. Sei diese o.B.d.A. in T_1 enthalten. Würde sie T_2 hinzugefügt, dann enthielte T_2 einen Kreis. Dieser enthält mindestens eine Kante $e' \in E'_2$ (ansonsten hätte er neben $e^\perp$ ausschliesslich Kanten aus $E_1 \cap E_2$ und wäre auch in T_1 ein Kreis). Wird e' aus T_2 entfernt und stattdessen $e^\perp$ eingefügt, dann erhalten wir eine Teilmenge von Kanten, die noch immer kreisfrei und zusammenhängend (also ein Spannbaum) ist, wegen $w(e^\perp) < w(e')$ aber ein geringeres Gesamtgewicht hat. Also war T_2 kein minimaler Spannbaum, was der Annahme widerspricht. ∎

c) Für jeden Spannbaum T gilt, dass er in G Gewicht $W = \sum_{e \in T} w(e)$ und in G' Gewicht $\sum_{e \in T} w(e) \circ c = W \circ c$ hat. Wir nehmen an, dass T in G kein MST ist, jedoch aber in G'. Das bedeutet, dass es in G einen MST $T' \neq T$ gibt, der ein Gewicht $W' < W$ hat. Nun ist das Gewicht von T' aber auch in G' kleiner als jenes von T, was unserer Annahme widerspricht, dass T ein MST ist in G'. Also ist jeder MST in G' auch ein MST in G. Die andere Richtung wird analog gezeigt oder die Operationen können umgekehrt werden. ∎

Beobachtung: Durch Addition, Subtraktion, Mulitiplikation und Division jedes Kantengewichts $w(e)$ mit einem Parameter $c \in \mathbb{R}^+$ ändert sich die relative Ordnung der Kantengewichte nicht. Wenn ein Kantengewicht $w(e)$ vor der Operation kleiner (oder grösser oder gleich) war als das einer anderen Kante $w(e')$ gilt dies auch nach der Operation. Jede monoton steigende Reihenfolge der Kantengewichte in G ist deshalb auch monoton steigend in G'. Deshalb können die Kantengewichte beliebig skaliert werden bevor zum Beispiel Kruskal ausgeführt wird.

Lösung 13.3 *Union Find von Hand.*

Bei Union wird zuerst h unter a gehängt. Bei Find werden d und e unter a gehängt. Die Datenstruktur sieht dann wie folgt aus:



Lösung 13.4 *AVL Trees.*

a) Es genügt, in jedem Knoten v die Grösse seines Teilbaumes $v.size$ zu speichern: die Anzahl Elemente in seinem linken Teilbaum plus die Anzahl Elemente in seinem rechten Teilbaum plus eins.

b) Sei `root` die Wurzel des AVL-Baums. Für jeden Knoten v, seien $v.left$ und $v.right$ sein linker und sein rechter Nachfolger, falls diese existieren, und `null` sonst.

Wir definieren die folgende FIND$(\cdot, \cdot)$ Operation, so dass FIND$(v, index)$ das $index$-te kleinste Element im Teilbaum mit Wurzel v zurückgibt, falls es ein solches Element gibt, und "*inexistent*" sonst. Dann rufen wir die Funktion für `root` und k auf, also FIND(`root`, k), um das k-te kleinste Element im AVL-Baum zu finden.

FIND$(v, index)$

```
1 if v.size < index then return "inexistent"
2 else if v.left ≠ null
3     if v.left.size ≥ index then return FIND(v.left, index)
4     else if v.left.size + 1 = index then return v
5     else return FIND(v.right, index − 1 − v.left.size)
```

```
6 else
7     if index = 1 then return v
8     else return FIND(v.right, index − 1)
```

Weil die Höhe des AVL-Baumes $\mathcal{O}(\log n)$ ist und jeder Aufruf der Funktion $\textsc{Find}(\cdot,\cdot)$ konstante Zeit benötigt, findet $\textsc{Find}(\texttt{root}, k)$ das k-te kleinste Element in $\mathcal{O}(\log n)$ Zeit.

c) Die Felder *.size* werden bei einer Einfügeoperation wie folgt aktualisiert:

- **Suchen**: In jedem besuchten Knoten v, wird $v.size$ um 1 erhöht. Beim Erstellen des neuen Knotens für das neu eingefügte Element, wird *.size* mit 1 initialisiert.
- **Rebanacieren**: Nach jeder Rotation von v und u werden deren Grössen (*.size*) neuberechnet als die Summe der Grössen ihrer jeweiligen linken und rechten Teilbäume plus 1. Dabei muss natürlich zuerst das (neue) Kind und erst dann der (neue) Vater aktualisiert werden.

Das Rebalancieren kostet pro Rotation konstante Zeit, da sich lediglich die Felder *.size* von zwei Knoten ändern und diese aber in konstanter Zeit neuberechnet werden können. Das Inkrementieren und Initialisieren in allen besuchten und neuen Knoten benötigt ebenfalls konstante Zeit pro Knoten. Weil die Höhe des AVL-Baumes $\mathcal{O}(\log n)$ ist, beträgt die Gesamtzeit also $\mathcal{O}(\log n)$.