Die Relaxierung in kürzeste–Wege–Algorithmen

(Handout)

D. Weber, dweber@htwsaar.de, 12. Juni 2014

1. Vorbemerkungen

Ein gewichteter Graph G besteht aus einer Knotenmenge V, einer Kantenmenge E und einer Gewichtsfunktion $w : E \longrightarrow \mathbb{Z}$

$$G = (V, E, w)$$

Ein kürzeste–Wege–Algorithmus berechnet die Distanz zwischen Knoten eines gewichteten Graphen. Es gibt zwei Klassen von Algorithmen

- Distanzen zwischen einem festen Startknoten s und allen Knoten $v \in V$ des Graphs, dies sind die single–source–shortest-path Algorithmen. Hier werden die Distanzen in einem Array `d[ ]` gespeichert, `d[v]` als Länge eines kürzesten Weges $s \rightsquigarrow v$. Zu dieser Klasse zählt der schon behandelte Dijkstra–Algorithmus.
- Distanzen zwischen allen Knotenpaaren $i, j \in V$ des Graphs, dies sind die all–pairs–shortest-path Algorithmen. Hier werden die Distanzen in einer Matrix (d_{ij}) gespeichert, d_{ij} als Länge eines kürzesten Weges $i \rightsquigarrow j$.

Die Breitensuche kann für einen ungewichteten Graphen als single–source–shortest-path Algorithmus aufgefasst werden.

2. Relaxierungsbedingung

Die Relaxierungsbedingung ist die wesentliche Idee in fast allen kürzeste Wege Algorithmen. Das Objekt einer Relaxierung ist eine Kante des Graphen.

Das Relaxieren eine Kante (u, v) bedeutet:

```
prüfe, ob die Ungleichung d[v] > d[u]+w(u,v) gilt
falls ja, setze d[v] <- d[u]+w(u,v)
```

Interpretation: die bisherigen gefundenen Wege nach v sind länger als der gerade gefundene Weg über u. Daher führt der Zwischenknoten u zu einer Verkürzung der Wegstrecke und die bisher (schlechtere) Abschätzung `d[v]` kann mit Hilfe von u verbessert werden. Deshalb ist u ein Vorgänger von v auf einem kürzesten Weg nach v. Es kann die Logik aus der Breitensuche zur Ausgabe des kürzesten Weges verwendet werden (`pred[ ]`–Array).

3. Eigenschaft der Relaxierung

Die Eigenschaft, die bewirkt, dass das Relaxieren funktioniert, ist folgende:

Falls
(1) die Kante (u, v) auf einem kürzesten Weg $s \rightsquigarrow v$ liegt **und**
(2) die Distanz `d[u]` schon korrekt berechnet ist
dann ist nach der Relaxierung der Kante (u, v) auch `d[v]` korrekt berechnet.

Dies kann man in einem Verfahren ausnutzen, das alle Kanten relaxiert: da die Distanz des Startpunkts richtig berechnet ist, kann man in obigem grauen Kasten $u = s$ setzen und es sind nach der Relaxierung aller Kanten diejenigen `d[v]` richtig, bei denen ein kürzester Weg aus einer Kante besteht.

In einem zweiten Durchlauf des Relaxierens aller Kanten werden die nun richtigen `d[v]` die Rolle von `d[u]` spielen, also sind danach diejenigen `d[v]` richtig, bei denen ein kürzester Weg aus zwei Kanten besteht.

Nach einem $n - 1$–ten Durchlauf des Relaxierens aller Kanten ist man fertig. Das Verfahren heißt *Bellman–Ford–Algorithmus* und wurde 1958 entdeckt.

4. All–Pairs Basisalgorithmus

Der Basisalgorithmus wendet den Bellman–Ford–Algorithmus für alle möglichen Startpunkte i an.

Der Basisalgorithmus besteht aus dem Relaxieren aller Kanten auf allen Wegen $i \rightsquigarrow j$, was nach der Logik aus dem vorherigen Abschnitt nach n Durchläufen zu korrekten Distanzen zwischen allen Paaren (i, j) führt.

```
for r=1 to n-1  // n-1 mal alle Kanten für alle (i,j)-Paare relaxieren
    for i=1 to n   // für alle Startknoten i
        for j=1 to n   // für alle Zielknoten j
            for k=1 to n   // für alle Zwischenknoten k
                d(i,j)=Ergebnis der Relaxierung der Kante (k,j)
```

Der Algorithmus lässt sich sehr einfach notieren, wenn die Struktur der Rechenoperationen klar wird.

Gesetz	$(\mathbb{R}, +, \cdot)$			$(\mathbb{R} \cup \{\infty\}, \min, +)$		
	Op.	in Formeln	OK?	Op.	in Formeln	OK?
assoziativ	$+$			min		
neutral	$+$			min		
invers	$+$			min		
kommutativ	$+$			min		
assoziativ	$\cdot$			$+$		
neutral	$\cdot$			$+$		
invers*	$\cdot$			$+$		
kommutativ	$\cdot$			$+$		
distributiv	$+, \cdot$			min, $+$		
algebraische Struktur						

(*) inverse zweite Operation: ohne neutrales Element der ersten Operation