8. Übung Informatik I

Michael Kramer Marcus Rickert
Version Marcus Rickert

30. Dezember 1995

Aufgabe 1

Erläuterungen

Bei diesem veränderten Algorithmus verwenden wir die Menge U, in der alle Knoten enthalten sind, für die schon ein Weg gefunden worden ist. Das Komplement von U enthält damit automatisch die Menge M und alle die Knoten, für die noch kein Weg gefunden wurde (also mit $dist(v) = \infty$).

1. Der erste Schritt des Algorithmus bleibt im Vergleich zur Vorlesung unverändert bis auf die Ausnahme, daß anstatt M die Menge U aufgebaut wird. Dies geschieht derart, daß in U alle Knoten $v \in V \setminus S$ mit $dist(v) \neq \infty$ eingetragen werden.

2. In der Vorlesung wurde im zweiten Schritt der Knoten $u \in V \setminus M$ mit minimalen Abstand von S bestimmt. Es läßt sich zeigen, daß gilt $u \in U$, d.h. es gibt einen Knoten mit $dist(u) < \infty$: angenommen dies wäre nicht der Fall, dann gäbe es in $V \setminus M$ nur Knoten mit unendlichem Abstand. Dies würde aber bedeuten, daß diese Knoten keinen Vorgänger haben, also nie durch Punkt 3 erreicht worden sind und dabei einen endlichen Weg zugewiesen bekommen haben. Das würde demnach der Definition eines gerichteten Graphen widersprechen.

   Also dürfen wir genauso gut den Knoten $u \in U$ mit minimalen Weg bestimmen. Im Gegensatz zur Vorlesung wird nun nicht M um u erweitert, sondern U um u verringert, denn zu u existiert nun ein kürzester Weg ($u \in M$).

3. Hier werden in der Vorlesung, wie auch bei uns, die Nachfolger von u untersucht. Dabei ist die Bedingung der Vorlesung $v \notin M$ offenbar unwesentlich (denn die Knoten mit kürzesten Wegen, die wir durch diese Abfrage abfangen würden, erfüllen die anschließende Größerrelation sowieso nicht).

   Die Knoten, die die Größerrelation erfüllen, bekommen wie in der Vorlesung einen endlichen Weg zugewiesen und müssen deshalb in U aufgenommen werden (falls sie nicht schon in U sind).

4. Nach der Überlegung von Punkt 2) erreichen wir mit 3) jeden Knoten $v \in V$. Somit wird auch jeder Knoten v irgendwann einmal Element von U, so daß ihm im folgenden durch Punkt 2) ein kürzester Weg zugewiesen wird. Wir sind also genau dann fertig, wenn wir jeden Knoten erreicht haben, also $U = \emptyset$. Dies ist die neue Abbruchbedingung.

Die Prozedur in D-Pidgin-Pascal (d für deutsch)

PROCEDURE dijkstra_u
$U := \emptyset$
Für alle $v \in V$ tue
    Falls $Kante(s, v) \in E$ dann
        $dist(s) := 0$
        $dist(v) := c(s, v)$
        $U := U \cup \{u\}$
        $vor(v) := s$
    sonst
        $dist(v) := \infty$
        $vor(v) := 0$
    Endefalls
Endefür
Solange $U \neq \emptyset$ tue
    Bestimme $u \in U$ mit $dist(u) = min\{dist(u) : v \in U\}$
    $U := U \setminus \{u\}$
    Für alle v mit $Kante(u, v) \in E$ tue
        Falls $dist(v) > dist(u) + c(u, v)$ dann
            $dist(v) := dist(u) + c(u, v)$
            $vor(v) := u$
            $U := U \cup \{u\}$
        Endefalls
    Endefür
Endesolange

Aufgabe 2

Obiger Algorithmus läßt sich mit folgenden Datenstrukturen realisieren. Zur Darstellung des gerichteten Graphen verwenden wir das in der Vorlesung angegebene Prinzip. Wir speichern die Knoten des Graphen in einem Feld, das Einträge folgenden Typs enthält:

```
type knotentyp =record
                nummer:integer
                dist:integer
                vor:integer
                element_von_u:boolean
                nachfolger:pointer
                end
```

Die Nachfolgerliste besteht nun aus Zeigern mit folgenden Einträgen:

```
type nachfolger =record
                nächster:pointer
                nummer:integer
                kosten:integer
                end
```

wobei *nummer* die Nummer des Nachfolgers in obigem Feld angibt und *kosten* die jeweiligen Kosten für diese Kante enthält.

Die Operationen Vereinigung und Löschen auf U geschehen dann durch das Setzen und Löschen der entsprechenden Boolean-Variablen *element_von_u* in den betroffenen Knoten ($O(1)$).

Die Bestimmung des Minimums von $dist(u)$ kann durch Durchsuchen aller Knoten mit gesetztem *element_von_u* erfolgen. Mit obiger Datenstruktur benötigen wir dazu $O(n)$ Schritte.

Durch die Kenntnis der Knotennummmer können wir in $O(1)$ auf $dist(v)$ zugreifen. In Schritt 3) arbeiten wir sukzessiv die Nachfolgerlisten ab und erhalten also dort auch in $O(1)$ die Kosten. Im ungünstigsten Fall hat der Knoten u n Nachfolger, so daß wir für Punkt 3) insgesamt $O(n)$ Schritte benötigen.

Dadurch daß jeder Knoten einmal Element von U war und in jedem Durchlauf bei Punkt 2) ein Element aus U entfernt wird, das aber nie wieder Element von U wird, müssen wir die Punkte 2) und 3) n-mal durchlaufen und benötigen eine Gesamtlaufzeit von $O(n^2)$.

Aufgabe 3

zu b)

In diesem Fall benötigt Punkt 3) immer noch $O(n)$ Schritte, weil im Durchschnitt auf jeden Knoten n Nachfolger entfallen. Selbst eine geeignete Datenstruktur, die die Laufzeit zur Bestimmung des Minimums verringert, würde für die Gesamtlaufzeit keine Besserung ergeben. Es bleibt bei der in Aufgabe 2) bestimmten Laufzeit von $O(n^2)$.

zu a)

Hier hat jeder Knoten im Durchschnitt einen Nachfolger, d.h. Punkt 3) benötigt nur noch $O(1)$. Gelänge es, die Laufzeit zur Minimumbestimmung in $O(\log(n))$ durchzuführen, dann betrüge die Gesamtlaufzeit lediglich $(O(n\log(n))$.

Mit der Datenstruktur eines (A,B)-Baumes für U könnte man das Minimum in $O(\log(n))$ bestimmen, dabei handelt man sich aber eventuell höhere Kosten für die Vereinigung- und Löschoperationen ein. Insgesamt führen wir maximal $2n$ solcher Operationen durch. Laut Vorlesung betragen die amortisierten Kosten für einen debalancierten (A,B)-Baum dabei insgesamt $O(2n) = O(n)$. Verwenden wir einen solchen Datentyp für U, dann brauchen wir bei der Laufzeitbetrachtung für die wiederholte Ausführung von Punkt 2) und 3) diese Kosten nicht zu berücksichtigen, denn damit fallen allein die Kosten für die Minimumsbestimmung mit einer Gesamtlaufzeit von $O(n\log(n))$ ins Gewicht. Mit einer solchen Datenstruktur läßt sich also der kürzeste Weg in $O(n\log(n))$ bestimmen.