Nicole Kaiser fl Tutorium: Do. 12-14 Uhr fl Tutor: Martin Klossek

23.01.2002

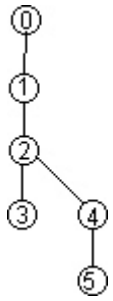
Theoretische Informatik 1, WS 2001/2002

Übungsblatt 11

Keine Garantie für die Korrektheit der hier vorgestellten Lösungen.

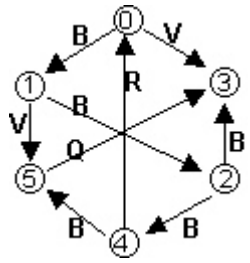
Aufgabe 11.1

(a)
Der Baum der Tiefensuche:



Er lässt aus der Adjazenzliste durch folgende Vorgehensweise ermitteln:

Von 0 nach 1, von 1 nach 2, von 2 nach 3. 3 zeigt auf nil, deshalb zurück zu 2.
2 zeigt auf 3, der Knoten ist aber schon besucht. Der kleinste Knoten, auf den 2 zeigt und der noch nicht besucht wurde, ist 4. Also von 2 nach 4, dann von 4 nach 5, weil 0 schon besucht wurde.
In den Baum können keine weiteren Knoten eingefügt werden, weil alle schon besucht wurden.



Baumkante **B**
Rückwärtskante **R**
Vorwärtskante **V**
Querkante **Q**

(b)
V = {s,x,y,z}
S = {s}
distanz[z] = 1, distanz[x] = 4, distanz[y] = ∞

kleinste Distanz: distanz[z] = 1
z in S einfügen: S = {s,z}
Distanzen der Nachfolger von z neu berechnen:
distanz[x] = 4 > [[distanz[z] = 1] + [Länge(z,x) = 1]] = 2 ?
Ja, also gilt für x: distanz[x] = 2.
distanz[y] = ∞ > [[distanz[z] = 1] + [Länge(z,y) = 3]] = 4 ?
Ja, also gilt für y: distanz[y] = 4.

kleinste Distanz: distanz[x] = 2
x in S einfügen: S = {s,z,x}
Distanzen der Nachfolger von x neu berechnen:
distanz[y] = 4 > [[distanz[x] = 2] + [Länge(x,y) = 1]] = 3 ?
Ja, also gilt für y: distanz[y] = 3.

kleinste Distanz: distanz[y] = 3
y in S einfügen: S = {s,z,x,y}
S = V und damit bricht der Algorithmus ab und liefert folgendes Ergebnis:
distanz[x] = 2, distanz[y] = 3, distanz[z] = 1.

Aufgabe 11.2

Idee:

Wenn die Knoten im ungerichteten Graphen nicht nummeriert sind, werden sie zuerst mit Nummern versehen, dabei darf keine Nummer mehrfach vorkommen.
Der Graph wird nach dem Algorithmus für Tiefensuche durchgegangen. Beginnend beim kleinsten Knoten *s* wähle einen benachbarten Knoten *v* mit dem kleinsten Wert, mache aus der ungerichteten Kante eine gerichtete Kante, und zwar von *s* nach *v*. Das gleiche mit *v*: suche den kleinsten Nachbarn, mache ungerichtete Kante zu gerichteter Kante usw.
Dabei sind folgende Abweichungen zum Algorithmus der Tiefensuche zu beachten:

- Für jeden Knoten wird eine Variable *weitereKanten* definiert, die die Anzahl der Kanten erhält, die an den Knoten anschließen.
- Wenn eine Kante von *v* nach *w* "gerichtet" wird, werden die Variablen *v.weitereKanten* und *w. weitereKanten* um eins verringert. Sobald diese Variable eines Knotens 0 ist, gibt es keine Kanten mehr, die gerichtet werden können. Der Knoten gilt also nach dem Algorithmus der Tiefensuche erst als "besucht", wenn es keine zu richtenden Kanten mehr gibt.

Da jeder Knoten mindestens mit zwei Kanten verbunden sein muss (da sonst die keine-Brücken-Eigenschaft in der Aufgabenstellung nicht erfüllt ist), kann es nicht passieren, dass für einen Knoten nur Inkanten definiert werden. Ein Knoten wird besucht, seine Inkante wird definiert, dann wird er verlassen und seine Outkante wird definiert. Wegen der keine-Brücken-Eigenschaft, kann es auch nicht vorkommen, dass in dem gerichteten Graphen eine Zyklus entsteht, den man nicht mehr verlassen kann.

Algorithmus:

```
n ist Anzahl der Knoten im Graphen;

for (int k=0; k<n; k++) k.weitereKanten = Anzahl der anliegenden Kanten;
for (k=0; k<n; k++)
        if(k.weitereKanten > 0) then tiefenSuche(k);

tiefenSuche(k) {
        k.weitereKanten = k.weitereKanten - 1
        setze gerichtete Kante von k zu seinem kleinsten Nachbarn k.next, fuer den
                k.next.weitereKanten > 0 gilt;
        for (alle Nachbarn von k denen noch keine Kante von k aus gesetzt wurde, in aufsteigender
                Reihenfolge, beginnend bei k.next)
                if (k.next.weitereKanten > 0) then tiefenSuche(k.next);
}
```

Aufgabe 11.3

Idee:

Verwendet wird eine Abwandlung des Dijkstra's Algorithmus. Dabei wird die maximale Steigung einer Strecke von *s* über *w* nach *v* aus dem Maximum der maximalen Steigung von *w* und der Länge der Kante von *w* nach *v* berechnet.

Algorithmus:

```
Startknoten s;
Menge V beinhaltet alle Knoten;
E ist Menge der Kanten;
laenge(x,y) ist Länge der Kante zwischen x und y
maxSteigung[v] ist maximale Steigung auf dem Weg von s nach v;


S = {s}
for (alle Knoten v aus V)
        if ( (s,v) ∈ E)
                maxSteigung[v] = laenge(s,v);
        else
                maxSteigung[v] = ∞;
```

```
while (S ≠ V) {
        waehle Knoten w ∈ V \ S mit kleinster maxSteigung;
        S = S ∪ w;
        for (alle Nachfolger v von w)
                if (maxSteigung[v] > max(maxSteigung[w], laenge(w,v))) then
                        maxSteigung[v] = max(maxSteigung[w], laenge(w,v))
}
```