Algorithmen und Programmierung 3

Übung 10

Torsten Schmidt und Steffen Richter
Tutor: Till Zoppke

Aufgabe 58

58. (11 Punkte) Betrachten wir die Knoten v eines Graphen in der Reihenfolge, wie der rekursive Aufruf T(v) bei der Tiefensuche beendet wird. Das heißt, wir fügen am Ende des Programms T(v) folgende Zeile ein:
num2++; T2Nummer[v] := num2;
(a) (4 Punkte) Beweisen Sie: Wenn es eine Kante (u, v) mit T2Nummer[v] > T2Nummer[u] gibt, dann enthält der Graph einen Kreis.
(b) (4 Punkte) Wie kann man diesen Kreis bestimmen? Schreiben Sie ein Programmstück für diese Aufgabe.
(c) (3 Punkte) Verwenden Sie die Tatsache aus Aufgabe (a), um aus der T2Nummerierung eines kreisfreien Graphen eine topologische Sortierung zu berechnen, sofern bei der Tiefensuche alle Knoten besucht werden. Beschreiben Sie den Algorithmus in Worten.

Pseudocode für die verwendete Tiefensuche in einem gerichteten Graphen:

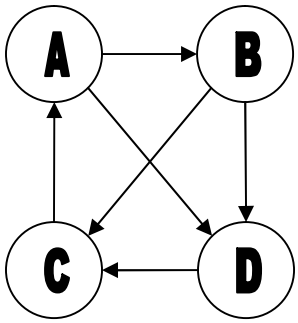
```
//globale Variablen:
int count = 0;
int count2 = 0;
bool Status (besucht / nicht besucht);
int tnum;
Liste der Nachfolger; //Die Adjazenzliste
int t2numm;
Knoten Vorgänger;


T(Knoten v)
{
        Status[v] = besucht;
        tnum[v] = ++count;
        //Nachfolger anschauen und Knoten aufrufen
        für alle Nachfolger w
        {
                if(Status[w]=nicht besucht)
                {
                        T(w);
                        Vorgänger[w]=v;
                }
        }
t2Numm[v] = ++count2;
}
```

Beispielgraph:

Algorithmen und Programmierung 3

Übung 10

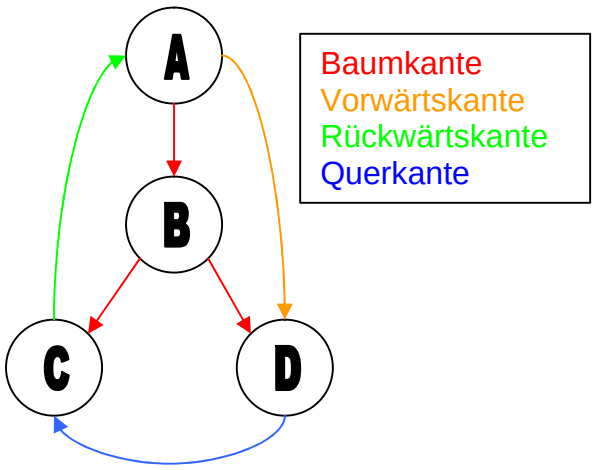
Torsten Schmidt und Steffen Richter
Tutor: Till Zoppke



Variablen bei einem Durchlauf der Tiefensuche:

	tnum	t2num	Status	Vorgänger
A	1	4	B	-
B	2	3	B	A
C	3	2	B	B
D	4	1	B	B

Es ergibt sich folgender Tiefensuchbaum:



a) Beweis:

Zu zeigen:
Sei G=(V,E) gerichteter Graph
Dann gilt nach T(s) für $s \in V$ beliebig:
$\exists (u,v) \in E : t2nummer[v] > t2num[u] \Rightarrow$ G enthält einen Kreis.
Beweis:
Durch Fallunterscheidung:

Algorithmen und Programmierung 3

Übung 10

Torsten Schmidt und Steffen Richter
Tutor: Till Zoppke

1. Baumkante:
Dieser Fall kann nicht eintreten, da für alle Baumkanten (u,v): t2num[u]>t2num[v];
2. Vorwärtskante
Dieser Fall kann nicht eintreten, da zu jeder Vorwärtskante (u,v) ein Weg (u,w1,…,wm,v) über Baumkanten existiert und somit sch t2num[n]>t2num[w1]>…>t2num[v] ergibt.
3. Querkante
Dieser Fall kann nicht eintreten, da zum Zeitpunkt des Aufrufes von T(u) v bereits traversiert und t2num[v] gesetzt wurde, also auch hier: t2num[u]>t2num[v]
4. Rückwärtskante
Dies ist der einzige Fall der noch verbleibt. Eine Rückwärtskante (u,v) schließt mit dem Baum-Kanten einen Kreis. Somit ist die Behauptung erfüllt.

b) Pseudocode für die Kreisausgabe:

Eingabe : Gerichteter Graph G=(V,E) und der Startknoten s
Ausgabe: Ein oder kein Kreis

```
//Tiefensuche durchführen um T2Num zu füllen
T(s)
Für alle Kanten (u,v) ∈ E
{
if(t2num[v] > t2num[u])
{
        //rückwärts
        do {print(u); u=Vorgänger[u ];}
        until (u=v);return;
}
Ausgabe „Kein Kreis“;
return;
}
```

c) Topologische Sortierung:

Speichere Knoten sortiert nach t2num;
Gebe Knoten absteigend sortiert aus.

Algorithmen und Programmierung 3

Übung 10

Torsten Schmidt und Steffen Richter
Tutor: Till Zoppke

Aufgabe 61

Betrachten Sie das folgende einfache Haskell-Programm:

```
camba [ ] _ = True
camba _ [ ] = False
camba (x:xs) (y:ys) = (x==y) && camba xs ys
klungo [ ] _ = True
klungo _ [ ] = False
klungo xs (y:ys) = camba xs (y:ys) || klungo xs ys
```

a) Beschreiben Sie in Worten, was diese beiden Funktionen berechnen.

camba ermittelt, ob das erste Array ein Präfix des zweiten Arrays ist. klungo verwendet camba, um zu ermitteln, ob das erste Array ein Teilwort des zweiten Arrays ist.

b) Spezifizieren Sie die beiden Funktionen mathematisch (modellierend).

camba : (STRING x STRING -> BOOL)

camba $((x_0, x_1, x_2, \ldots, x_m), (y_0, y_1, y_2, \ldots, y_n)) = \begin{cases} true & m \leq n \wedge \forall 0 \leq i \leq m : x_i = y_i \\ false & sonst \end{cases}$

klungo : (STRING x STRING -> BOOL)

klungo $((x_0, x_1, x_2, \ldots, x_m), (y_0, y_1, y_2, \ldots, y_n)) = \begin{cases} true & \exists 0 \leq k \leq m-n \forall 0 \leq i \leq m : x_i = y_{i+k} \\ false & sonst \end{cases}$

c) Wie viele Vergleiche der Form x==y werden bei den folgenden Eingaben durchgeführt?

```
klungo "aaaab" "aaaaaaaaaa"
klungo "abababc" "bababababa"
```

Dieser Vergleich wird nur in camba durchgeführt, wir brauchen uns also nur diese Funktion anzusehen. Für den Fall, dass alle Elemente überprüft werden müssen, wird für jedes Element im ersten Array wird ein Vergleich durchgeführt, es sei denn, das zweite Array ist kürzer; dann wird für jedes Element im zweiten Array ein Vergleich durchgeführt. Wird jedoch ein Element gefunden, für dass x=y falsch ist, wird abgebrochen, da durch die UND-

Verknüpfung der Rückgabewert nicht mehr wahr sein kann. Im ersten Aufruf werden also zuerst sechsmal fünf Vergleiche durchgeführt – für die Fälle, dass das zweite Array die Länge zehn bis fünf hat -, und dann bei jedem Durchlauf ein Vergleich für jeweils ein Element des zweiten Arrays:

AnzVerg = 6*5 + 4 + 3 +2 + 1 = 40

Beim zweiten Aufruf wird beim ersten Durchlauf nur ein Vergleich durchgeführt, da bereits die beiden ersten Elemente falsch liefern. Der zweite Durchlauf bricht erst beim letzten, siebten Element des ersten Arrays ab. Der dritte Durchlauf benötigt wieder nur einen Vergleich usw. Ab dem sechsten Durchlauf greift die Tatsache, dass das zweite Array nur noch fünf Elemente hat, ab dann werden also fünf, ein, drei, ein, ein Vergleich(e) nötig:

AnzVerg = 1 + 7 + 1 + 7 + 1 + 5 + 1 + 3 + 1 + 1 = 28