Algorithmen und Programmierung 3, WS 2003/2004 — 10. Übungsblatt

Joscha Krutzki, Georgy Sklyarenko

57. (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.

   (a) durch Fallunterscheidung: die Kante (u,v) sei
   1) Baumkante: sowas kann nicht eintreten;
   2) Vorwärtskante: auch nicht;
   3) Querkante: auch nicht, da v bereits traversiert, wenn u betrachtet wird;
   4) Rückwärtskante: da haben wir - offensichtlich - einen Kreis;

   (b) Sobald eine Kante (u,v) mit T2nummer[u]<T2nummer[v] enthält der Graph einen Kreis, weil dann wurde u später besucht als v und gibt es einen Weg von v nach u.

```
findeKreis (Graph g){
   für alle Kanten (u,v) vom g{
      if (T2nummer[u] < T2nummer[v]) then{
         Ausgabe von u;        // Kreisanfang
         w := v;               // w ist Läufer
         while (w != u){
            Ausgabe von w;
            w := Vorgänger[w]; // Weg von v nach u
         }
      }
   }
}
```

   (c) 1) Prüfe alle Kanten (u,v), ob T2nummer[u]<T2nummer[v] ist. Falls ja, Abbruch: der Graph enthält einen Kreis, topologische Sortierung nicht möglich.
   2) Prüfe alle Knoten u, ob Tnummer[u]>0. Falls nein, Abbruch: der Algorithmus funktioniert nicht.
   3) Ausgabe der Knoten nach absteigender T2nummer. Begründung: der Knoten mit dem größten T2nummer hat keine Vorgänger, sonst hätte sein Vorgänger eine größere T2nummer.

58. (9 Punkte) 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) (1 Punkt) Beschreiben Sie in Worten, was diese beiden Funktionen berechnen.

(b) (4 Punkte) Spezifizieren Sie die beiden Funktionen mathematisch (modellierend).

(c) (4 Punkte) Wie viele Vergleiche der Form `x==y` werden bei den folgenden Eingaben durchgeführt?

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

(a) Beschreibung:

camba(x,y) = "x ist Präfix von y"

klungo(x,y) = "x ist Teilwort von y"

(Bemerkung: ein leeres Wort ist Präfix und Teilwort von jedem Wort).

(b) Spezifikation:

Wort ist eine Zeichenkette $w = x_1x_2...x_n, n \geq 0$;

leeresWort = [ ], $(n = 0)$;

klungo :: Wort $\rightarrow$ Wort $\rightarrow$ Bool

camba :: Wort $\rightarrow$ Wort $\rightarrow$ Bool

$\text{camba}(x, y) = (x = [\,]) \bigvee (n \leq m \bigwedge x_i == y_i, \forall i = 1...n)$

$\text{klungo}(x, y) = (x = [\,]) \bigvee (n \leq m \bigwedge \exists k, 1 \leq k \leq m - n + 1 : x_i = y_{i+k-1}, \forall i = 1...n)$

wobei $x = x_1x_2...x_n$, $y = y_1y_2...y_n$.

(c) `klungo "aaaab" "aaaaaaaaaa"`

Beide Argumente nicht leer $\Rightarrow$ Zeile(6) wird ausgeführt.

Als erstes wird camba(x,y) ausgewertet, was uns zur Zeile (3) führt. Diese Zeile ruft sich rekursiv auf, solange die beiden Argumente nicht leer sind und gleiche Zeichen auf denselben Stellen von vorne stehen, also 4 mal erfolgreich, auf 5-er Stelle wird b==a als false ausgewertet, d.h. 5 Vergleiche.

Zurück zur Zeile (6), zum 2.Teil von oder-Verknüpfung: Das 1.Zeichen von y wird gestrichen und das ganze wird wiederholt. Wieviel mal? Solange $|x| \leq |y|$, d.h. 6 mal.

6 * 5 Vergleiche = 30 Vergleiche

Danach ändert sich nich viel. Wenn $|y| = 4$ wird schon nach 4 Vergleiche das 2.Argument leer $\Rightarrow$ camba wird als false ausgewertet und terminiert, wenn $|y| = 3$ geschieht das nach 3 Vergleiche usw.

Insgesamt: 30 + 4 + 3 + 2 + 1 + 0 = 40 Vergleiche.

`klungo "abababc" "bababababa"`

Analog, nur muss man nun aufpassen, wann ungleiche Zeichen auftauchen: passiert dies schon auf der 1.Stelle, wird nur 1 Vergleich durchgeführt.

Also, zählen wir:

1 + 7 + 1 + 7 + 1 + 5 + 1 + 3 + 1 + 1 = 28 Vergleiche.