Algorithmen und Programmierung 3, WS 2019/2020 — 2. Übungsblatt

Abgabe bis Freitag, 1. November 2019, 12:00 Uhr, in die Fächer der Tutoren

9. Topologisches Sortieren, 0 Punkte

   Geben Sie alle möglichen topologischen Sortierungen für folgende Eingabe an: Es gibt 6 Knoten, und die Kanten sind $\{(6,4),(3,5),(4,2),(3,4),(5,4),(1,2),(1,6),(3,6)\}$.

10. Topologisches Sortieren, 0 Punkte

    (a) Was passiert beim in der Vorlesung angegebenen Algorithmus für topologisches Sortieren, wenn ein Paar (i,j) in der Eingabe mehrfach auftritt? Was passiert, wenn ein Paar (i,i) auftritt?

    (b) Untersuchen Sie verschiedene Möglichkeiten, wie man die Liste der freien Elemente beim topologischen Sortieren verwalten kann, im Hinblick auf ihre Effizienz. Kann man auf diese Liste auch gänzlich verzichten?

    (c) Welche Variablen oder Attribute im Programm aus der Vorlesung[1] könnte man ohne Nachteil einsparen?

11. Zirkuläre Abhängigkeit, 7 Punkte

    Erweitern Sie den Algorithmus zum topologischen Sortieren aus der Vorlesung, sodass bei der Existenz eines gerichteten Kreises nicht einfach mit einer Meldung abgebrochen wird, sondern auch ein Kreis (als „Beweis“) ausgegeben wird.

    Beschreiben Sie Ihren erweiterten Algorithmus in Pseudo-Code. (Zum Beispiel können Sie eine Schleife “für alle Elemente x …” schreiben.)

    Erklären Sie Ihren Algorithmus *in Worten*, ohne notwendigerweise auf Details der Implementierung einzugehen. (Sie müssen nur das erklären, was gegenüber dem ursprünglichen Algorithms neu ist. Ein Korrektheitsbeweis ist nicht verlangt.)

12. Programmieraufgabe, 6 Punkte, hochzuladen auf der KVV-Seite der Veranstaltung

    Schreiben Sie ein Java-Programm für die vorige Aufgabe. Sie können das Programm aus der Vorlesung[1] erweitern. Ihr Programm soll für einen Graphen mit n Knoten und m Kanten nicht mehr als $O(m+n)$ zusätzliche Zeit und nicht mehr als $O(n)$ zusätzlichen Speicher brauchen.

13. algebraischen Spezifikation für Mengen, 7 Punkte

    Leiten Sie aus der algebraischen Spezifikation für Mengen

$$
\begin{aligned}
\mathit{istenthalten}(x, \mathit{leereMenge}) &= \mathit{False} \\
\mathit{istenthalten}(x, \mathit{einfüge}(x,m)) &= \mathit{True} \\
\mathit{istenthalten}(x, \mathit{einfüge}(y,m)) &= \mathit{istenthalten}(x,m), && \text{für } x \neq y \\
\mathit{istenthalten}(x, \mathit{lösche}(x,m)) &= \mathit{False} \\
\mathit{istenthalten}(x, \mathit{lösche}(y,m)) &= \mathit{istenthalten}(x,m), && \text{für } x \neq y
\end{aligned}
$$

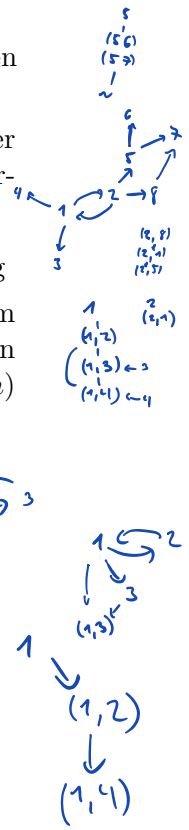
    durch Umformungen folgende Identität her: Für $x \neq y$ gilt

$$\mathit{istenthalten}(u, \mathit{lösche}(x, \mathit{einfüge}(y,m))) = \mathit{istenthalten}(u, \mathit{einfüge}(y, \mathit{lösche}(x,m))).$$

14. Vereinigung und Durchschnitt, 0 Punkte

    Erweitern Sie die algebraische Spezifikation von Mengenoperationen aus der vorigen Aufgabe um die Funktionen *Vereinigung* und *Durchschnitt* (von je zwei Mengen).

[1] https://mycampus.imp.fu-berlin.de/access/content/group/fdc1e04e-407d-4d35-ac90-d6f03f853bc2/Programme/TopoSort.java



13. algebraischen Spezifikation für Mengen, 7 Punkte

Leiten Sie aus der algebraischen Spezifikation für Mengen

$$
\begin{aligned}
&1 \quad istenthalten(x, leereMenge) = False \\
&2 \quad istenthalten(x, einfüge(x, m)) = True \\
&3 \quad istenthalten(x, einfüge(y, m)) = istenthalten(x, m), \qquad \text{für } x \neq y \\
&4 \quad istenthalten(x, lösche(x, m)) = False \\
&5 \quad istenthalten(x, lösche(y, m)) = istenthalten(x, m), \qquad \text{für } x \neq y
\end{aligned}
$$

durch Umformungen folgende Identität her: Für $x \neq y$ gilt

$$istenthalten(u, lösche(x, einfüge(y, m))) = istenthalten(u, einfüge(y, lösche(x, m))).$$

Fall 1. $u = x$

L.S.:		R.S.:	
	ie (u, l (x, ei (y, m)))		ie (u, ei (y, l (x, m)))
$u=x$ ⇒	ie (u, l (u, ei (y, m)))	$u=x$ ⇒	ie (u, ei (y, l (u, m)))
4. ⇒	False	3. ⇒	ie (u, l (u, m))
		4. ⇒	False

Fall 2. $u = y$

L.S.:		R.S.:	
	ie (u, l (x, ei (y, m)))		ie (u, ei (y, l (x, m)))
$u=y$ ⇒	ie (u, l (x, ei (u, m)))	$u=y$ ⇒	ie (u, ei (u, l (x, m)))
5. ⇒	ie (u, ei (u, m))	2. ⇒	True
2. ⇒	True		

Fall 3. $u \neq y \wedge u \neq x$

L.S.:		R.S.:	
	ie (u, l (x, ei (y, m)))		ie (u, ei (y, l (x, m)))
5. ⇒	ie (u, ei (y, m))	3. ⇒	ie (u, ei (y, m))
3. ⇒	ie (u, m)	5. ⇒	ie (u, m)

11. Zirkuläre Abhängigkeit, 7 Punkte

Erweitern Sie den Algorithmus zum topologischen Sortieren aus der Vorlesung, sodass bei der Existenz eines gerichteten Kreises nicht einfach mit einer Meldung abgebrochen wird, sondern auch ein Kreis (als „Beweis“) ausgegeben wird.

Beschreiben Sie Ihren erweiterten Algorithmus in Pseudo-Code. (Zum Beispiel können Sie eine Schleife “für alle Elemente x . . . ” schreiben.)

Erklären Sie Ihren Algorithmus *in Worten*, ohne notwendigerweise auf Details der Implementierung einzugehen. (Sie müssen nur das erklären, was gegenüber dem ursprünglichen Algorithms neu ist. Ein Korrektheitsbeweis ist nicht verlangt.)

Sofern nur ein Kreis vorliegt:

1. O(n-1) Zeit O(1) Speicher
   Geh die Knotenliste durch, bis ein Knoten k des Kreises gefunden wurde (bzw. Anzahl der Vorgänger von k ≠ 0)

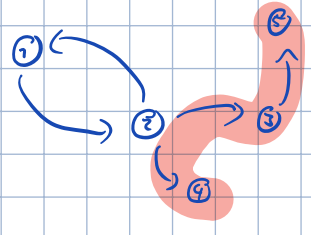
2. O(1) Zeit und Speicher
   Merke den Knoten knext, welcher der Knoten ist, auf den k "zeigt". (Nachfolger von k)

(print) Gebe k und knext auf der Konsole aus

3. O(n) Zeit O(1) Speicher
   Solange k ≠ knext ist: (bzw. Einmal den Kreis durchlaufen)
   a) (print) knext und Nachfolger von knext
   b) knext = Nachfolger von knext

(print) Letztes mal knext und Nachfolger von knext auf Konsole ausgeben.
Nun sollte der Kreis einmal Durchlaufen und ausgegeben worden sein.

Sonst: Auszugebende Knoten von Kreisen entfernen!



1. O(n) Zeit O(n) Speicher
   Alle Knoten die keine Vorgänger haben und der Nachfolger == null in eine Liste speichern (Freie Knoten)

2.
   Wiederhole n-mal:
   Finde einen Knoten V mit Nachfolger == 0 und Vorgänger ≠ 0 und Speichere in einer Liste L
   - Wenn es keinen gibt: Tue "Es liegt nur ein Kreis vor"
   Für alle Knoten in L:
   Lösche die Knoten und die Abhängigkeiten zu den Knoten
   if (Knoten.Vorgänger == null)
   Füge Vorgänger in L hinzu