12. Aufgabenblatt zur Vorlesung

Algorithmen und Programmierung III

Wolfgang Mulzer WiSe 2013/14

Abgabe am 23. Januar 2014 vor der Vorlesung in die jeweiligen Tutorenfächer

Aufgabe 1 Implementierung von Tries *10 Punkte*

(a) Beschreiben Sie kurz, wie man konkret die Operationen `put`(s, v), `get`(s), `remove`(s) und `succ`(s) auf unkomprimierten Tries implementieren kann. Dabei ist s jeweils eine nichtleere Zeichenkette und v ein Wert aus einer endlichen Wertemenge V. Geben Sie die Laufzeiten an.

(b) Implementieren Sie unkomprimierte Tries mit den Operationen aus (a) in Java.

Aufgabe 2 Collections in Java *10 Punkte*

(a) In der Vorlesung haben Sie verschiedene abstrakte Datentypen kennengelernt, mit denen sich Mengen von Objekten bzw. von Paaren von Objekten speichern lassen. Nennen Sie vier solche ADTs, und nennen Sie jeweils zwei substantiell unterschiedliche Implementierungen.

(b) Informieren Sie sich über das *Java Collections Framework*. Wie ist dieses grob strukturiert? Wo und wie finden sich die ADTs aus (a) wieder? Welche Implementierungen bietet das Framework für die ADTs?

Anmerkung: Falls es Ihnen lieber ist und falls Ihr Tutor nichts dagegen hat, können Sie (b) natürlich auch für eine andere Programmiersprache oder Laufzeitbibliothek Ihrer Wahl bearbeiten, wie zum Beispiel die STL von C++, das .Net Framework oder die Python standard library.

Aufgabe 3 Tiefensuche *10 Punkte*

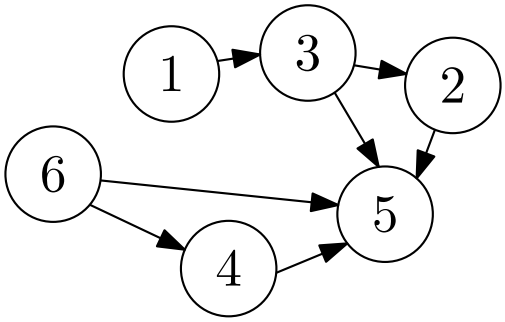
(a) Benutzen Sie Tiefensuche, um in einem zusammenhängenden, ungerichteten Graphen einen Weg zu finden, der jede Kante genau einmal in jeder Richtung durchläuft.

(b) Sei $G = (V, E)$ ein kreisfreier gerichteter Graph. Ein solcher Graph heißt DAG (für **d**irected **a**cyclic **g**raph). Eine *topologische Sortierung* von G ist eine Anordnung der Knoten von G, so dass keine Kante in G von einem späteren zu einem früheren Knoten verläuft. Betrachten Sie den folgenden Algorithmus:

```
dfs(v, topoOrder)
  v.found <- true
  for e in v.outgingEdges() do
    w <- e.opposite(v)
    if !w.found then
      dfs(w, topoOrder)
  topoOrder.push(v)

topoSort(G)
  topoOrder <- new Stack
  for v in G.vertices() do
    v.found <- false
  for v in G.vertices() do
    if !v.found do
      dfs(v, topoOrder)
```

Führen Sie den Algorithmus auf dem folgenden Graphen aus. Gehen Sie dabei davon aus, dass die Knoten in `vertices` und `outgoingEdges` gemäß der Knotennummern angeordnet sind. Zeigen Sie die einzelnen Schritte. Wie erhält man im Anschluss die topologische Sortierung?



(c) (*freiwillig*, 5 Zusatzpunkte) Beweisen Sie, dass der Algorithmus aus (b) funktioniert.

Hinweis: Führen Sie einen Widerspruchsbeweis. Nehmen Sie an, es gäbe eine Kante e von einem späteren zu einem früheren Knoten in `topoSort`, und überlegen Sie sich, was passiert, wenn `dfs` die Kante e untersucht. Beachten Sie dabei, dass es einen Unterschied macht, ob die Tiefensuche für den Endpunkt von e noch aktiv ist.