Datenstrukturen (DS)

Sommersemester 2017

Prof Dr. Martin Hoefer
Niklas Hahn

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Institut für Informatik
Algorithmen und Komplexität

Übung 3

Ausgabe: 16.05.2017
Abgabe: 30.05.2017

Aufgabe 3.1. *Elementare Datenstrukturen* (9 Punkte)

Wir kennen aus der Vorlesung die Datenstrukturen *Stack* und *Queue*. Es ist möglich, diese Datenstrukturen durch jeweils zwei andere zu ersetzen, ohne ihre Implementierungen zu verändern.

a) Zeigen Sie, wie eine Queue durch zwei Stacks ersetzt werden kann. Dabei sollen n *enqueue*- und m *dequeue*-Operationen insgesamt Zeit $\mathcal{O}(n+m)$ benötigen.

b) Zeigen Sie, wie ein Stack durch zwei Queues ersetzt werden kann und analysieren Sie die Laufzeiten der Operationen *push* und *pop*.

c) Entwickeln Sie eine Datenstruktur, die die Funktionen *push*, *pop* und *avg* unterstützt. Die Funktion *avg* soll hierbei den durchschnittlichen Wert aller in der Datenstruktur enthaltenen Elemente zurückgeben. Alle Operationen sollen in Zeit $\mathcal{O}(1)$ laufen.

Aufgabe 3.2. *Einfach-verkettete Listen* (3 Punkte)

Gegeben ist eine einfach-verkettete Liste `list`, deren n Elemente einen Pointer *next* und einen numerischen Wert *data* enthalten. Geben Sie einen Algorithmus in Pseudo-Code an, analysieren Sie seine Laufzeit und begründen sie seine Korrektheit. Es steht nur konstant viel zusätzlicher Speicher zur Verfügung. Nehmen Sie an, dass *current* auf die Kopfzelle zeigt. Verwenden Sie hierfür die in der Vorlesung vorgestellten Listenoperationen sowie Lese- und Schreibzugriff auf *current*.

Teilen Sie die Liste `list` in zwei Teile, sodass alle Elemente mit $data < k$ für ein gegebenes $k \in \mathbb{N}$ am Anfang der Liste stehen und alle Elemente mit $data \geq k$ am Ende der Liste sind. Die Laufzeit sollte $\mathcal{O}(n)$ sein.

Aufgabe 3.3. *Baum-Traversierungen* (6 Punkte)

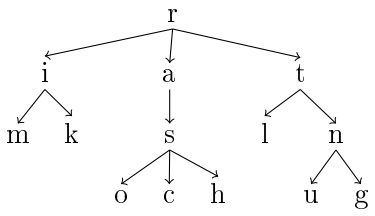
Betrachten Sie den rechts dargestellten geordneten Baum B.
Geben Sie an, in welcher Reihenfolge die Knoten in B in einem

a) Präorder-Durchlauf

b) Inorder-Durchlauf

c) Postorder-Durchlauf

besucht werden.



Bitte wenden!

Aufgabe 3.4. *Mischen mit Stacks, Queues und Deques* (12 Punkte)

Die Eingabe für die folgenden Teilaufgaben ist stets die Folge 1,2,3,...,n für ein gerades $n \in \mathbb{N}_{>4}$. Als Ausgabe erhalten Sie:

a) 1,2,3,...,n

b) $n, n-1, n-2, ..., 1$

c) 1,3,5,...,$n-1$, 2,4,...,n

d) $n-1$,$n-3$,...,1,2,4,...,n

Sie wissen, dass die Daten nur von einem Stack, einer Queue oder einem Deque mit den aus der Vorlesung bekannten Standardoperationen verarbeitet wurden. Sie kennen jedoch weder die verwendete Datenstruktur noch die Reihenfolge der Operationen.
Bestimmen Sie für alle Ausgabefolgen a)-d), mit welchen der Datenstrukturen diese Ausgabe möglich ist und geben Sie jeweils eine mögliche Operationsfolge an. Sollte eine Datenstruktur nicht in Frage kommen, begründen Sie kurz, warum dies so ist. Jedes Element $1, 2, 3, ..., n$ wird hierbei genau einmal in der Datenstruktur gespeichert und irgendwann danach wieder ausgegeben, sodass nach der Verarbeitung kein Element mehr in der Datenstruktur gespeichert ist. Es gibt somit auch genau n Speicher-Operationen und genau n Ausgabe-Operationen.

Beispielsweise gilt für $n = 4$:
Die Operationsfolge „*push, push, pop, push, pop, push, pop, pop*“ auf einem Stack hat als Ausgabe 2,3,4,1.

Aufgabe 3.5. *Ahnenkunde* (6+2* Punkte)

Ein Baum B sei in Eltern-Array-Darstellung gegeben: Für jeden Knoten u ist der Elternknoten von u in Zelle Baum[u] gespeichert.

Entwerfen Sie einen Algorithmus, der für zwei gegebene Knoten u, v ihren letzten gemeinsamen Vorfahren bestimmt, d.h. den *tiefsten* Knoten w, in dessen Teilbaum sich sowohl u als auch v befinden. Ihr Algorithmus soll höchstens die Laufzeit $\mathcal{O}(\text{Tiefe}(u) + \text{Tiefe}(v))$ benötigen.

Begründen Sie die Korrektheit Ihres Algorithmus und zeigen Sie, dass er die Laufzeit einhält.

*Durch das Angeben eines Algorithmus mit Laufzeit $\mathcal{O}(\text{Tiefe}(u) + \text{Tiefe}(v) - 2\text{Tiefe}(w))$ können Sie bis zu 2 Bonuspunkte erhalten.

Hinweis: Sie können das Array “Baum” verwenden, um einen Knoten u zu markieren, indem Sie den Eintrag Baum[u] überschreiben.