Goethe-Universität Frankfurt am Main
Institut für Informatik
Theorie Komplexer Systeme
Dr. Mariano Zelke

15. Mai 2012

Datenstrukturen

Sommersemester 2012

Übungsblatt 3

Abgabe: *bis 29. Mai 2012, 8.15 Uhr (vor der Vorlesung oder in Raum RM 11-15/113)*

Bitte achten Sie darauf, dass Sie auf der Abgabe Ihrer Lösung Ihren ***Namen****, Ihre* ***Matrikelnummer*** *und Ihre* ***Übungsgruppe*** *angeben. Fehlt eine dieser Angaben, müssen Sie mit* ***Punktabzug*** *rechnen. Mehrseitige Abgaben müssen zusammengeheftet werden.*

Eine Aufgabe gilt nur dann als bearbeitet, wenn neben der Lösung auch die notwendigen Begründungen angegeben sind – es sei denn, in der Aufgabenstellung steht, dass eine solche Begründung nicht erforderlich ist.

Aufgabe 1: (10 Punkte)

(a) Es sei eine einfach verkettete Liste `a` der Klasse `liste` aus der Vorlesung gegeben. Die Elemente von `a` seien entsprechend ihrer jeweiligen Werte von `data` der Größe nach aufsteigend in `a` angeordnet.

Beschreiben Sie einen möglichst effizienten Algorithmus in Pseudocode oder C++, der die Liste `a` invertiert, so dass eine Liste entsteht, deren Elemente absteigend sortiert sind. Analysieren Sie die asymptotische Laufzeit Ihres Algorithmus.

(b) Ziel dieser Aufgabe ist die Simulation einer Queue durch eine einfach verkettete Liste.

Dazu sei die Klasse `liste2` gegeben, die genauso wie die Klasse `liste` aus der Vorlesung definiert ist, außer, dass sie zusätzlich eine Funktion `movetoend()` besitzt, welche den current-Zeiger in Laufzeit $\mathcal{O}(1)$ auf das letzte Element der Liste setzt.

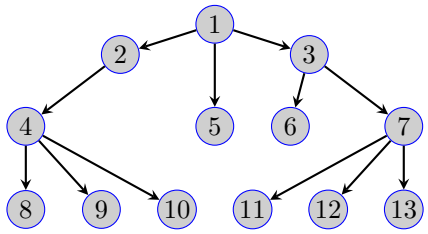
Beschreiben Sie, wie sich eine Queue für die Speicherung von ganzen Zahlen durch eine einfach verkettete Liste der Klasse `liste2` simulieren lässt. Geben Sie dazu in Pseudocode oder C++ an, wie durch die Operationen der Klasse `liste2` eine Implementierung der Queue-Operationen `enqueue(x)` und `dequeue()` in Laufzeit $\mathcal{O}(1)$ gelingt.

Aufgabe 2: (8 Punkte)

Entwerfen Sie eine Datenstruktur, die ganze Zahlen speichert und die Stack-Operationen `pop()` und `push(x)` unterstützt sowie die Operation `findmin()`. Jede dieser Operationen soll eine Laufzeit von $\mathcal{O}(1)$ haben; die Dauer einer jeden Operation darf also nicht von der Anzahl der schon in der Datenstruktur gespeicherten Zahlen abhängen. Dabei liefert die Funktion `findmin()` die kleinste der gespeicherten Zahlen, verändert aber deren Menge nicht.

Aufgabe 3: (11 Punkte)

(a) Es sei der folgende *geordnete* Baum B in graphischer Darstellung gegeben:



Geben Sie die Reihenfolge an, in der ein Postorder-Durchlauf die Knoten von B besucht. Geben Sie außerdem die Implementierung von B mittels Vater-Array, Adjazenzliste sowie Kind-Geschwister-Darstellung an. Eine Begründung ist jeweils nicht erforderlich.

(b) Sei T ein *geordneter binärer* Baum, für den die Besuchsreihenfolge der Knoten in Preorder und Inorder wie folgt bekannt ist:

- Preorder-Reihenfolge: 8, 9, 6, 2, 3, 4, 7, 1, 5
- Inorder-Reihenfolge: 6, 9, 8, 3, 7, 4, 1, 2, 5

Geben Sie T in graphischer Darstellung an, wobei klar erkennbar sein soll, ob es sich bei einem Kind um ein rechtes oder ein linkes Kind handelt. Eine Begründung ist nicht erforderlich.

Aufgabe 4: (13 Punkte)

Es sei ein Baum B mit n Knoten in Kind-Geschwister-Darstellung gegeben.

(a) Geben Sie je einen Algorithmus in C++ oder Pseudocode an, der in Laufzeit $\mathcal{O}(n)$ die Werte der Knoten von B in

(i) Postorder ausgibt. **(ii)** Inorder ausgibt.

Eine Begründung ist jeweils nicht erforderlich.

(b) Geben Sie eine Funktion `AnzahlBlätter(Knoten p)` in C++ oder Pseudocode an, die auf dem Baum B arbeitet und berechnet, wie viele Blätter im Teilbaum von B mit der Wurzel `p` existieren. Dabei bezeichnet `p` den Knoten, der der Funktion beim Start übergeben wird, der Aufruf `AnzahlBlätter(wurzel)` bestimmt also die Gesamtzahl aller Blätter in B. Die Funktion soll in einer Laufzeit von $\mathcal{O}$(Anzahl der Knoten in Teilbaum mit Wurzel `p`) arbeiten und *nicht rekursiv* sein.