Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Tobias Lieber

Sommersemester 2011
Übungsblatt 5
16. September 2011

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: 16. September 2011

Hausaufgabe 1

Implementieren Sie in der Klasse `UIbiHeap` einen Binomial-Heap. Hierzu muss auch ein BinomialBaum in der Klasse `binomialTree` implementiert werden.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIbiHeap` und `binomialTree`.

Hausaufgabe 2

Implementieren Sie in der Klasse `IbinaryTree` einen binären Baum, der neben den Operationen `insert, remove, find` auch die Operationen `preOrder` und `postOrder` implementiert.
Die Funktion `preOrder` druckt ein Element (anfangs die Wurzel) und steigt dann rekursiv zuerst in den linken und dann in den rechten Teilbaum ab (In einer ersten Version des Blattes war links und rechts vertauscht). Die Funktion `postOrder` arbeitet umgekehrt. Sie führt zuerst die rekursiven Aufrufe aus und druckt dann das Element (anfangs die Wurzel).
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die bereitgestellten Klassen aber nicht deren Interfaces.

Hausaufgabe 3

Implementieren Sie in der Klasse `Graph` eine Repräsentation eines Graphen durch eine Adjazenzliste. Stellen Sie die Funktionen wie im Interface angegeben bereit.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die bereitgestellten Klassen aber nicht deren Interfaces.

Hausaufgabe 4

Implementieren Sie in der Klasse `DFS` eine Funktion, die auf der eben implementierten `Graph`-Klasse zu einem angegebenen Knoten s eine Tiefensuche durchführt und die Knoten ausgibt nachdem alle Kinder abgearbeitet sind. Achten Sie darauf, dass Ihre Implementierung im Gegensatz zu dem in der Vorlesung vorgestellten Ansatz nicht rekursiv ist.
Verwenden Sie für Ihre Implementierung die auf der Übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die bereitgestellten Klassen aber nicht deren Interfaces.

Aufgabe 1

Geben Sie einen Algorithmus für eine Build-Funktion an, die n Elemente $e_1, \ldots, e_n$ als Eingabe erhält und diese in einem Binomial-Heap speichert und die in $\mathcal{O}(n)$ Zeit läuft. Beweisen Sie die Laufzeit Ihres Algorithmus.

Aufgabe 2

Fügen Sie die Schlüssel $a, b, \ldots, i$ in der Reihenfolge $(a, i, e, g, h, f, c, b, d)$ in einen anfänglich leeren AVL-Baum ein. Entfernen Sie anschließend den Schlüssel i. Zeichnen Sie jeweils den resultierenden Baum (einschließlich notwendiger Rotationen).

Aufgabe 3

Führen Sie auf einem anfangs leeren $(2, 4)$-Baum folgende Operationen aus und zeichnen Sie die Zwischenergebnisse: `insert(23)`, `insert(30)`, `insert(13)`, `insert(6)`, `insert(40)`, `insert(80)`, `insert(62)`, `insert(75)`, `insert(28)`, `insert(21)`, `insert(29)`, `remove(62)`, `remove(75)`, `remove(13)`.

Aufgabe 4

Beweisen Sie folgende Aussage:
Für einen $(2, 3)$-Baum gibt es eine Folge von n `insert` bzw. `remove`-Operationen, so dass die Anzahl der nötigen Aufspaltungen und Vereinigungen von internen Knoten in $\Omega(n \log n)$ ist.

Aufgabe 5

In dieser Aufgabe modifizieren wir die gestellten Bedingungen für einen (a, b)-Baum so dass wir einen B^*-Baum erhalten.
Wir ändern die Grad-Invariante wie folgt: Ein Knoten darf höchstens Grad b haben. Jeder Knoten außer der Wurzel hat mindestens $\frac{2b-1}{3}$ Kinder. Die Wurzel hat mindestens 2 Kinder und höchstens $2\lfloor\frac{2b-2}{3}\rfloor + 1$ Kinder.

- Wie müssen die `insert` und `delete`-Operationen des (a, b)-Baumes modifiziert werden, so dass die Grad-Invariante immer gegeben ist?
- Welche Vorteile und Nachteile ergeben sich aus praktischer und theoretischer Sicht für die Speicherausnutzung und die Laufzeit der `insert`-Operationen?

Aufgabe 6

Geben Sie Algorithmen `postNext(v)` und `preNext(v)` an, die zu einem Knoten v in einem Binärbaum den in der PreOrder bzw. PostOrder folgenden Knoten w berechnet. Analysieren Sie die asymptotische Worst-Case Laufzeit Ihres Pseudocodes.
Berechnen Sie außerdem die asymptotische Laufzeit wenn mittels der Operationen `postNext(v)` und `preNext(v)` die vollständige PreOrder bzw. PostOrder berechnet wird (also n-maliges Anwenden der Funktion).