Technische Universität München
Fakultät für Informatik
Lehrstuhl für Effiziente Algorithmen
Dr. Hanjo Täubig
Jeremias Weihmann

Sommersemester 2012
Übungsblatt 12
9. Juli 2012

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 16. bis 22. Juli

Tutoraufgabe 1

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 (bei einem gewurzelten Baum ist unter Grad die Zahl der Kinder zu verstehen). Jeder Knoten außer der Wurzel hat mindestens $\frac{2b-1}{3}$ Kinder. Die Wurzel hat mindestens 2 Kinder (falls mindestens ein Element im Baum vorhanden ist, welches nicht das Dummy-Element ∞ ist) und höchstens $2\lfloor\frac{2b-2}{3}\rfloor + 1$ Kinder.

(a) Wie müssen die `insert` und `delete`-Operationen des (a, b)-Baumes modifiziert werden, so dass die Grad-Invariante immer gegeben ist?

(b) Welche Vorteile und Nachteile ergeben sich aus praktischer und theoretischer Sicht für die Speicherausnutzung und die Laufzeit der `insert`-Operationen?

Tutoraufgabe 2

Ein nichtleerer Binärbaum kann (unter anderem) in PreOrder, InOrder und PostOrder traversiert werden. Diese Traversierungen sind wie folgt rekursiv definiert:

- PreOrder:
  1. Besuche die Wurzel.
  2. Traversiere den linken Teilbaum in PreOrder, falls dieser nichtleer ist.
  3. Traversiere rechten Teilbaum in PreOrder, falls dieser nichtleer ist.
- InOrder:
  1. Traversiere den linken Teilbaum in InOrder, falls dieser nichtleer ist.
  2. Besuche die Wurzel.
  3. Traversiere den rechten Teilbaum in InOrder, falls dieser nichtleer ist.
- PostOrder:
  1. Traversiere den linken Teilbaum in PostOrder, falls dieser nichtleer ist.
  2. Traversiere den rechten Teilbaum in PostOrder, falls dieser nichtleer ist.

3. Besuche die Wurzel.

Wir bemerken, dass die Anordnung nach PreOrder genau der Anordnung nach der dfs-Nummer entspricht, wenn die Tiefensuche einen linken Teilbaum stets vor dem korrespondierenden rechten Teilbaum besucht. Analog dazu entspricht die Anordnung nach PostOrder genau der Anordnung nach der (dfs-)finish-Nummer.

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).

Tutoraufgabe 3

Gegeben Sei die Adjazenzmatrix $A = (a_{ij})_{1 \leq i,j \leq n}$ eines gerichteten, einfachen Graphen G mit n Knoten. Geben Sie einen Algorithmus an, der eine Senke in $\mathcal{O}(n)$ Schritten findet. Eine Senke sei (abweichend von der üblichen Definition) ein Knoten v, der Eingangsgrad $n - 1$ und keine ausgehenen Kanten hat.

Anmerkung zu den Hausaufgaben auf diesem Blatt:

Da der Abgabetermin Ihre individuelle letzte Tutorstunde ist, und daher keine weitere Tutorstunde für die Rückgabe der Hausaufgaben zur Verfügung steht, werden die korrigierten Hausaufgaben spätestens am folgenden Montag, dem 23. Juli, in einem Ordner beschriftet mit **GAD** im Aufenthaltsraum MI 03.09.051 zur Verfügung stehen.

Hausaufgabe 1

Implementieren Sie in der Klasse `IbinaryTree` einen binären Baum, der neben den Operationen `insert, remove, find` auch die Operationen `preOrder` und `postOrder` implementiert. Es empfiehlt sich, unter anderem die Klasse `biNode` zu verwenden.

Die Funktion `preOrder` druckt ein Element (anfangs die Wurzel) und steigt dann rekursiv zuerst in den linken und dann in den rechten Teilbaum ab. 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.

Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse auf den Rechnern der Rayhalle (`rayhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_bt` kompiliert werden kann. Anderenfalls kann eine Korrektur nicht garantiert werden. Achten Sie darauf, dass Ihr Quelltext ausreichend kommentiert ist.

Schicken Sie die Lösung per Email mit dem Betreff `[GAD] Gruppe <Gruppennummer>` an Ihren Tutor.

Hausaufgabe 2

Bestimmen Sie die Laufzeit des BFS-Algorithmus. Betrachten Sie zwei Implementierungen: Zum einen basierend auf einer Adjazenzmatrix und zum anderen basierend auf einer Adjazenzliste.

Hausaufgabe 3

In dieser Aufgabe wollen wir ein alternatives Verfahren zur topologischen Sortierung auf einem DAG G betrachten. Dazu nehmen wir an, dass uns weder die üblichen Graphoperationen zum Traversieren einer Kante oder Abfragen der Nachbarschaft eines Knotens, noch irgendwelche „höheren Datenstrukuren“ wie z.B. Queues, Listen, PriorityQueues oder Stacks zur Verfügung stehen.

Die einzige Operation, die wir benutzen können ist eine Tiefensuche (DFS), die Arrays `dfs_num` und `finish_num` zurückgibt, die zu den jeweiligen Knoten die entsprechenden Werte beeinhalten. Außerdem stehen uns `for`-Schleifen und die übliche Addition bzw. Subtraktion zur Verfügung.

a) Geben Sie einen Algorithmus (Pseudocode reicht) an, der dieselbe asymptotische Laufzeit erreicht, wie der in der Vorlesung vorgestellte Algorithmus.

b) Begründen Sie die Laufzeit Ihres Algorithmus kurz.

c) Beweisen Sie die Korrektheit Ihres Algorithmus.