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

Sommersemester 2013
Übungsblatt 10
24. Juni 2013

Grundlagen: Algorithmen und Datenstrukturen

Abgabetermin: Jeweilige Tutorübung in der Woche vom 1. bis 5. Juli

In der Vorlesung wurden Suchbäume als sortierte Liste zusammen mit einer Navigationsstruktur (also dem eigentlichen Baum) eingeführt. Im Rahmen der Übung besteht der Einfachheit halber jeder AVL-Baum nur aus der Navigationsstruktur. Alle Ergebnisse können aber auch auf die Darstellung aus der Vorlesung (d.h. inklusive der Elementliste) übertragen werden.

Tutoraufgabe 1

Führen Sie auf einem anfangs leeren Binomialheap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({15, 20, 9, 1, 11, 4, 13, 17, 42, 7}),
- delMin(),
- delMin(),
- delMin(),
- delMin()

Die build-Operation ist dabei durch iterative Ausführung von der insert-Operation auf den Elementen realisiert. Die insert-Operation wiederum ist eine merge-Operation auf dem zu erstellenden Binomialheap und einem einelementigen Binomialheap, wobei letzterer das einzufügende Element enthält.

Tutoraufgabe 2

Diese Aufgabe ist ein Kurztutorial, das die `insert`- und `remove`-Operationen der in der Vorlesung eingeführten AVL-Bäume spezifiziert.

In der Praxis werden AVL-Bäume wie folgt implementiert: An jedem Knoten v des Baumes befindet sich ein Flag b_v. Diese Flags können Werte aus $\{-2, -1, 0, 1, 2\}$ annehmen, wobei der Wert b_v den Balancierungsgrad der beiden an den Kindern von v gewurzelten Teilbäumen repräsentiert (ist ein Teilbaum leer, interpretieren wir dessen Tiefe als -1). Beispielsweise bedeutet $b_v = -1$, dass die Baumtiefe des linken Teilbaum von v um 1 größer als die des rechten Teilbaum von v ist. Diese Flags stellen *lokale* Informationen dar, die dazu verwendet werden können, um den Baum nach einer Modifikation zu rebalancieren und dabei nur eine logarithmische Anzahl an Knoten zu betrachten. Bei einem korrekten AVL-Baum muss der Balancierungsgrad jedes Knotens ein Wert aus $\{-1, 0, 1\}$ sein. Die Werte -2 und 2 treten nur während der Durchführung einer Operation auf.

Die `insert`-Operation

Die `insert`-Operation sucht zunächst denjenigen Knoten v_k, an den der einzufügende Knoten w als neues Blatt angehängt wird, wobei $(v_1, v_2, \ldots, v_k)$ der Pfad von der Wurzel v_1 zu v_k ist. Wir hängen w als Blatt an v_k. Falls w das linke bzw. rechte Kind von v_k ist, ist die Tiefe des linken bzw. rechten Teilbaums von v_k um genau 1 größer geworden. Entsprechend modifizieren wir den Wert b_{v_k}. Anschließend bearbeiten wir die Knoten v_i in der Reihenfolge $v_k, v_{k-1}, \ldots, v_1$, wobei wir gegebenenfalls auch früher aufhören können, wenn wir in einen Fall geraten, der sicherstellt, dass der Baum ein korrekter AVL-Baum ist.

Wir beschreiben nun ganz allgemein, was wir tun müssen, wenn wir einen Knoten $v := v_i$ bearbeiten. Hierbei können wir aufgrund unserer Vorgehensweise stets drei Eigenschaften voraussetzen.

- Der Wert von b_v ist korrekt.
- Die Tiefe entweder des linken oder des rechten Teilbaums von v hat sich vergrößert.
- Der linke und der rechte Teilbaum von v sind AVL-Bäume.

Fall $b_v = 0$: Dann war vor der Einfügung $b_v = \pm 1$. Die Tiefe des Teilbaums von v ist also gleich geblieben, und wir können aufhören, da der Baum insgesamt ein AVL-Baum ist.

Fall $b_v = \pm 1$: Dann war vor der Einfügung $b_v = 0$. Die Tiefe des Teilbaums von v ist also größer geworden. Falls v das linke Kind von v_{i-1} ist, muss $b_{v_{i-1}}$ dekrementiert werden. Andernfalls wird der Wert inkrementiert. Anschließend fährt man bei v_{i-1} fort.

Fall $b_v = \pm 2$: Dann ist der Teilbaum von v kein AVL-Baum. Vor der Einfügung von w war $b_v = \pm 1$. In Abhängigkeit von den Balancierungen der Kinder von v muss eine Einfach- oder eine Doppelrotation durchgeführt werden, wobei auch die Knotenflags entsprechend modifiziert werden. Die Abbildungen 1 und 2 stellen die Situationen und die Rotationen dar, bei denen zu Beginn $b_v = +2$ ist (hierbei entspricht v dem Knoten A). Die entsprechenden Rotationen im Falle $b_v = -2$ sind dazu spiegelverkehrt. Man mache sich klar, dass nach der Rotation der resultierende Baum in jedem Fall ein AVL-Baum ist, sodass wir aufhören können.

Wir sehen also, dass wir nach der Bearbeitung von v_i nur dann bei v_{i-1} weitermachen müssen, wenn $b_{v_i} = \pm 1$ ist.

Die `remove`-Operation

Die Operation sucht zunächst den Knoten v, der entfernt werden soll. Jetzt sind zwei Fälle zu unterscheiden.

Fall 1: v ist ein Blatt oder hat genau ein Kind. Sei $(v_1, \ldots, v_{k-1}, v)$ der Pfad von der Wurzel zu v. Nun wird v gelöscht, wobei gegebenenfalls das Kind von v zu einem Kind von v_{k-1} gemacht wird. Falls v das linke Kind von v_{k-1} war, wird $b_{v_{k-1}}$ inkrementiert, und andernfalls dekrementiert. Anschließend fährt man bei v_{k-1} fort.

Fall 2: v hat zwei Kinder. Sei $(v_1, \ldots, v_d = v, \ldots, v_k)$ der Pfad von der Wurzel über v zum rechtesten Knoten im linken Teilbaum von v (dieser Knoten v_k enthält das nächst kleinere

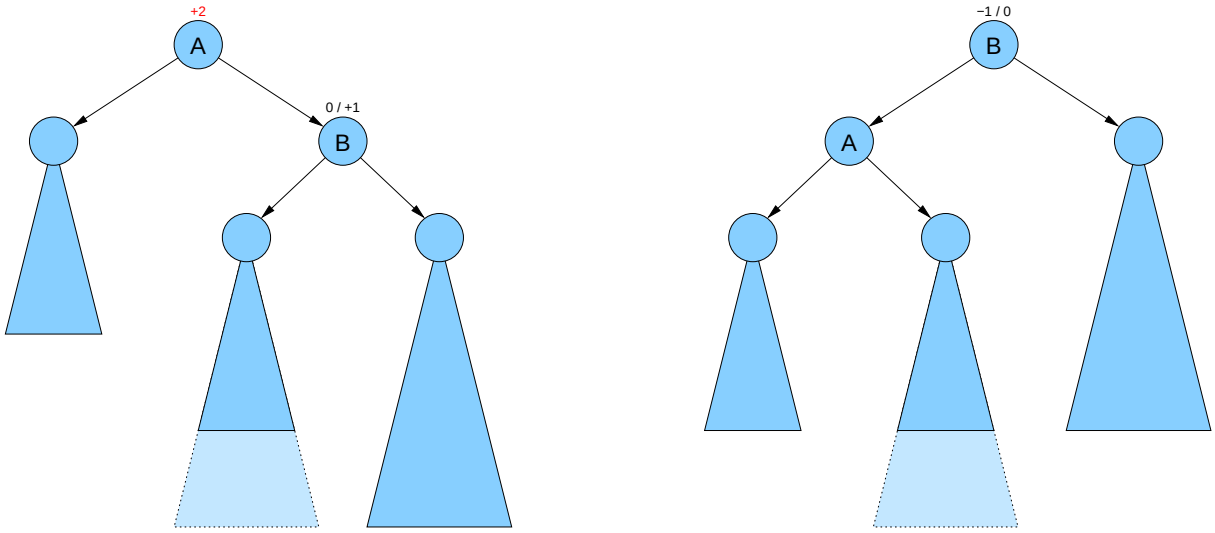

Abbildung 1: Linksrotation am Knoten A

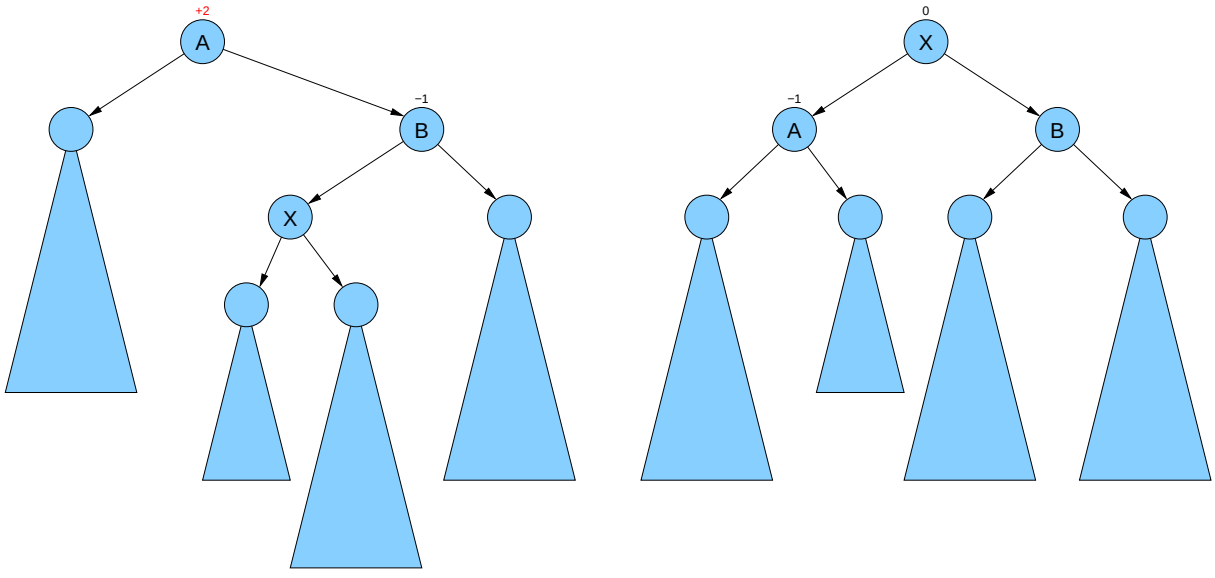

Abbildung 2: Doppelrotation am Knoten A bestehend aus einer Rechtsrotation am Knoten B gefolgt von einer Linksrotation am Knoten A

Element bezüglich v, und hat maximal ein Kind). Zunächst vertauscht man die Knoten v und v_k, d.h., der neue Pfad von der Wurzel zu v ist jetzt $(v_1, \ldots, v_{d-1}, v_k, v_{d+1}, \ldots, v_{k-1}, v)$. Hierdurch verändert sich die Balancierung des Baums zunächst nicht. Nun hat v maximal ein Kind und wird wie bei Fall 1 gelöscht.

Wir beschreiben nun ganz allgemein, was wir tun müssen, wenn wir einen Knoten $v := v_i$ bearbeiten. Hierbei können wir aufgrund unserer Vorgehensweise stets drei Eigenschaften voraussetzen.

- Der Wert von b_v ist korrekt.
- Die Tiefe entweder des linken oder des rechten Teilbaums von v hat sich verkleinert.
- Der linke und der rechte Teilbaum von v sind AVL-Bäume.

Fall $b_v = 0$: Dann war vor der Elementlöschung $b_v = \pm 1$. Die Tiefe des Teilbaums von v ist also kleiner geworden. Falls v das linke Kind von v_{i-1} ist, muss $b_{v_{i-1}}$ inkrementiert werden. Andernfalls wird der Wert dekrementiert. Anschließend fährt man bei v_{i-1} fort.

Fall $b_v = \pm 1$: Dann war vor der Elementlöschung $b_v = 0$. Die Tiefe des Teilbaums von v ist also gleich geblieben, und wir können aufhören, da der Baum insgesamt ein AVL-Baum ist.

Fall $b_v = \pm 2$: Dann ist der Teilbaum von v kein AVL-Baum. Vor der Elementlöschung war $b_v = \pm 1$. In Abhängigkeit von den Balancierungen der Kinder von v muss eine Einfach- oder eine Doppelrotation durchgeführt werden, wobei auch die Knotenflags entsprechend modifiziert werden. Man orientiere sich erneut die Abbildungen 1 und 2. Bei einer Einfachrotation kann der Fall auftreten, dass sich die Tiefe des betroffenen Teilbaums nicht ändert, sodass die Balancierung von v_{i-1} sich nicht ändert und wir aufhören können. Falls hingegen durch die Einfach- oder Doppelrotation die Tiefe des betroffenen Teilbaums geringer wird, muss $b_{v_{i-1}}$ entsprechend angepasst werden: Ist die Tiefe des am linken Kind von v_{i-1} gewurzelten Teilbaums kleiner geworden, wird $b_{v_{i-1}}$ inkrementiert, andernfalls dekrementiert. Anschließend fahren wir beim Knoten v_{i-1} fort.

Intuitiv ausgedrückt: Nach Einfügen oder Löschen eines Elements arbeiten wir uns von der betroffenen Position des Baumes nach oben vor, wobei wir stets am tiefsten unbalancierten Knoten des Baumes Einfach- bzw. Doppelrotationen durchführen.

Tutoraufgabe 3

Gegeben sei ein AVL-Baum der nur aus einem Knoten mit Schlüssel 10 besteht. Fügen Sie nacheinander die Schlüssel 5, 17, 3, 1, 4 ein. Löschen Sie dann den Schlüssel 4, und fügen Sie dann die Schlüssel 8, 2, 7, 6, 9 ein. Löschen Sie dann die Knoten mit den Schlüsseln 2, 1, 8. Zeichnen Sie den AVL-Baum für jede Einfüge- bzw. Löschoperation und geben Sie an, ob Sie keine, eine Einfach- oder Doppelrotation durchgeführt haben.

Hausaufgabe 1

Implementieren Sie in der Klasse `UIaHeap` einen adressierbaren binären Heap, der neben den Basisoperationen auch die folgenden Operationen bereitstellt:

- `insertH(`e`)`: Fügt ein Element e in den Heap ein und gibt ein Integer als Handle zurück

- `removeH(`h`)`: Entfernt das Element mit Handle h
- `decreaseKeyH(`h, k`)`: Verringert das Element mit Handle h auf den Wert k.

Verwenden Sie für Ihre Implementierung die auf der übungswebseite bereitgestellten Klassen und verändern Sie für Ihre Implementierung *ausschließlich* die Klasse `UIaHeap`.

Achten Sie bei der Abgabe Ihrer Aufgabe darauf, dass Ihre Klasse `UIaHeap` heißt und auf den Rechnern der Linuxhalle (`lxhalle.informatik.tu-muenchen.de`) mit der bereitgestellten Datei `main_ah` 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.

Hinweis: Hier ist die Verwendung eines Dynamischen Arrays sinnvoll. Falls Sie die früher benutzte Klasse `UIsArray` benutzen wollen, so fügen Sie den Quellcode Ihrer Abgabe hinzu. Sie können aber auch das in Java implemntierte Dynamische Array benutzen, welches deutlich flexibler ist.

Anmerkung: Handles müssen im Allgemeinen keine Integers sein. Oft ist es sinnvoller, Objekte als Handles zu verwenden, die Referenzen auf die tatsächlichen Daten in der Datenstruktur bzw. deren Wrapper besitzen.

Hausaufgabe 2

Führen Sie auf zwei anfangs leeren Binomialheap folgende Operationen aus und stellen Sie die Zwischenergebnisse graphisch dar:

- build({99,98,97,96,95}) des ersten Binomialheaps,
- build({1,2,3}) des zweiten Binomialheaps,
- merge der beiden Binomialheaps,
- delMin(),
- delMin(),
- delMin()

Die build-Operation ist dabei durch iterative Ausführung von der insert-Operation auf den Elementen realisiert. Die insert-Operation wiederum ist eine merge-Operation auf dem zu erstellenden Binomialheap und einem einelementigen Binomialheap, wobei letzterer das einzufügende Element enthält.

Hausaufgabe 3

Gegeben sei ein AVL-Baum der nur aus einem Knoten mit Schlüssel a besteht. Fügen Sie nacheinander die Schlüssel i, e, g, h, f, c, b, d ein. Löschen Sie dann die Knoten mit den Schlüsseln e, g, d, h, i. Die Sortierung der Schlüssel dabei lexikographisch. Zeichnen Sie den AVL-Baum für jede Einfüge- bzw. Löschoperation und geben Sie an, ob Sie keine, eine Einfach- oder Doppelrotation durchgeführt haben.