Algorithmen & Datenstrukturen 1

Sommersemester 2022

Prof. Dr. Martin Hoefer
Tim Koglin, Marco Schmalhofer

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Institut für Informatik
Algorithmen und Komplexität

Übungsblatt 4

Ausgabe: 10.05.2022
Abgabe: 17.05.2022, **08:00**

Aufgabe 4.1 *AVL-Bäume* (7 + 6 Punkte)

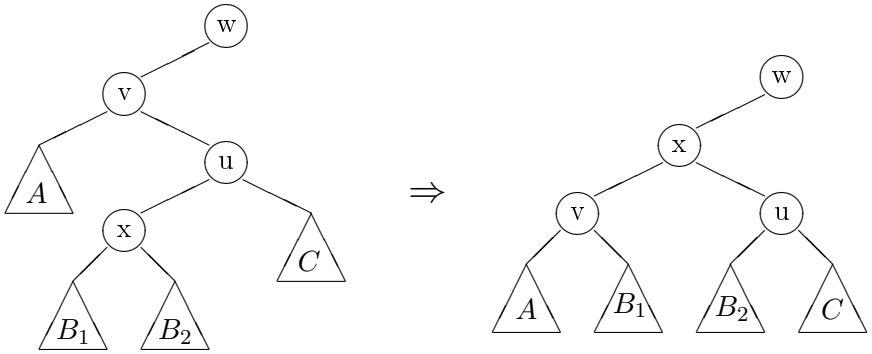
a) Fügen Sie die Schlüssel

34, 18, 5, 61, 40, 42, 50

in dieser Reihenfolge in einen anfangs leeren AVL-Baum ein. Stellen Sie den Baum inklusive der Balancegrade aller Knoten nach jeder Einfüge-Operation dar. Sofern nach einer Einfügung (mindestens) eine Rotation notwendig ist, stellen Sie den Baum zusätzlich vor jeder Rotation dar. Geben Sie in diesem Zusammenhang außerdem an, welcher Rotationsfall (Zick-Zick, Zick-Zack, Zack-Zick oder Zack-Zack) vorliegt und welche Rotation(en) ausgeführt wird bzw. werden.

b) Betrachten Sie die im Folgenden explizit dargestellte Doppelrotation für einen AVL-Baum (Zick-Zack-Fall). Die Bezeichnungen w, v, u und x geben die *Adresse* des jeweiligen Knotens im Speicher an. Wir nehmen an, dass jeder Knoten im Baum zwei Zeiger besitzt, `links` und `rechts`, die auf das linke bzw. rechte Kind des Knotens zeigen.

Geben Sie eine gültige Folge von notwendigen Zeigeraktualisierungen für diese Doppelrotation an. Sie brauchen Ihre Antwort nicht weiter zu begründen.



Aufgabe 4.2 *Ganzzahlige Prioritäten* (8 Punkte)

Betrachten Sie eine Prioritätswarteschlange mit n Elementen. Jedes Element hat eine ganzzahlige Priorität aus der Menge $\{1, 2, \ldots, k\}$, wobei k eine Konstante unabhängig von n ist.

Entwerfen Sie eine Datenstruktur, welche die Operationen `insert(x,p)` und `delete_latest_max()` in Zeit $\mathcal{O}(1)$ implementiert. Die Operation `insert(x,p)` soll ein Element x mit Priorität p einfügen, die Operation `delete_latest_max()` soll das Element mit größter Priorität ausgeben und entfernen. Falls es mehrere Elemente mit maximaler Priorität gibt, soll `delete_latest_max()` das Element ausgeben, welches davon zuletzt in die Datenstruktur eingefügt wurde.

Beschreiben Sie, wie die Elemente in Ihrer Datenstruktur gespeichert werden und wie die Operationen `insert(x,p)` und `delete_latest_max()` ablaufen. Begründen Sie, warum die Laufzeitschranke eingehalten wird.

Aufgabe 4.3 *Erweiterte AVL-Bäume* (12 Punkte)

Neben den Operationen `lookup(x)`, `insert(x)` und `remove(x)` betrachten wir nun zusätzlich die Operationen `select(k)` und `rang(x)`. Hierbei bestimmt die Operation `select(k)` den k-kleinsten Schlüssel für $k \in \mathbb{N}$, und `rang(x)` $= k$ gilt genau dann, wenn der Schlüssel x der k-kleinste Schlüssel ist.

Beschreiben Sie eine Modifikation der AVL-Bäume, die bei n gegenwärtig gespeicherten Schlüsseln *alle* fünf obigen Operationen in worst-case Laufzeit $\mathcal{O}(\log n)$ unterstützt. Beschreiben Sie wie immer zuerst Ihre Ideen und geben Sie dann Pseudocode für die Anpassungen an. Begründen Sie, dass die Laufzeit eingehalten wird, und argumentieren Sie, warum Ihre Prozeduren korrekt sind.

Hinweis: Speichern Sie neben dem Balancegrad weitere Informationen in den Knoten. Wie sieht die Aktualisierung dieser Informationen aus?

Aufgabe 4.4 *k-AVL-Bäume* (5 + 2 Punkte)

In dieser Aufgabe relaxieren wir die AVL-Eigenschaft: Sei $k \geq 2$ eine natürliche Zahl. Ein binärer Suchbaum ist ein *k-AVL-Baum*, wenn er folgende Eigenschaft hat:

Für jeden Knoten v und Teilbäume T_1 und T_2 direkt unterhalb von v gilt

$$\text{Tiefe}(T_1) \leq \text{Tiefe}(T_2) + k.$$

a) Zeigen Sie durch vollständige Induktion nach t, dass ein k-AVL-Baum mit Tiefe t mindestens $2^{t/k}$ viele Knoten enthält.

b) Bezeichne $t(n, k)$ die maximale Tiefe eines k-AVL-Baumes mit n Knoten. Leiten Sie aus der Aussage in a) eine möglichst gute asymptotische obere Schranke für $t(n, k)$ her. Die Schranke soll eine Funktion von n und k sein.