Philipps-Universität Marburg
Fachbereich Mathematik und Informatik
Arbeitsgruppe Datenbanksysteme

Sommersemester 2021
13. 06. 2021

Prof. Dr. Bernhard Seeger
Nikolaus Glombiewski, M.Sc.
Amir El-Shaikh, M.Sc.

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Abgabe: 20. 06. 2021,
bis **spätestens** 23:59 Uhr
über die ILIAS Plattform

Übung 9

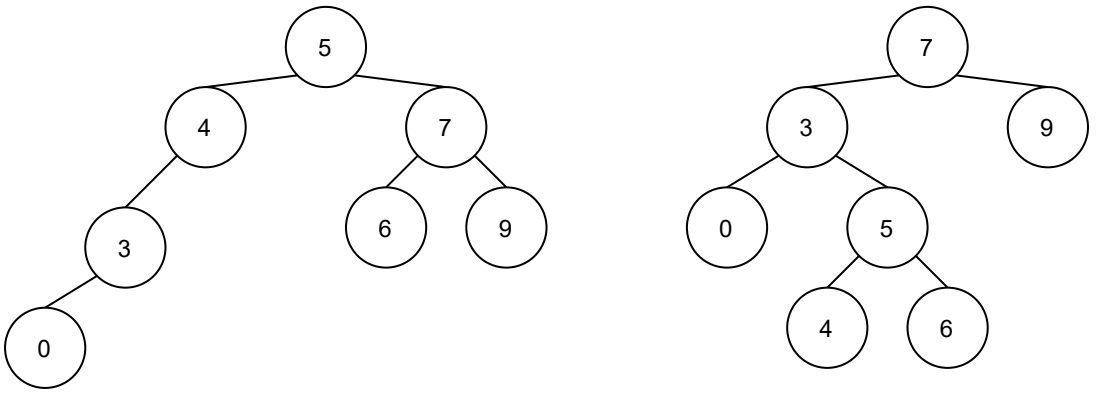
Aufgabe 9.1: Theorie zu AVL- und Rot-Schwarz-Bäumen (1+2+1+1) (5 Punkte)

a) Betrachten Sie die unten abgebildeten Suchbäume. Überführen Sie den linken Suchbaum mit möglichst wenigen Rotationen in den rechten Suchbaum. Geben Sie für jeden Schritt die Art der Rotation an und zeichnen Sie den Suchbaum nach jeder Rotation.

b) Beschreiben Sie einen Algorithmus, der zwei AVL-Bäume in linearer Zeit vereinigt. Begründen Sie die Laufzeit des Algorithmus.

c) Zeigen Sie, dass alle Pfade eines Rot-Schwarz-Baums von der Wurzel zu den Blättern die gleiche Anzahl an schwarze Knoten haben.

d) Wie viele Knoten hat ein Rot-Schwarz-Baum der Höhe h mindestens? Begründen Sie Ihre Antwort.



Aufgabe 9.2: Implementierung von 2-3-4-Bäumen (2+3+2+3) (10 Punkte)

a) Erstellen Sie eine Klasse `TwoThreeFourTree<K>`, für eine generische 2-3-4-Baum Implementierung. Nutzen Sie für die Modellierung der Knoten eine innere Klasse. Weiter sollen die Methoden `K getMin()` und `K getMax()` das kleinste bzw. das größte Element im Baum zurück geben. Beachten Sie, dass `K` das `Comparable` Interface erfüllt.

b) Ergänzen Sie die Klasse `TwoThreeFourTree<K>` um die Methoden `boolean insert(K key)` und `boolean search(K key)`. Diese Methoden sollen entsprechend einen neuen Schlüssel hinzufügen und einen bestehenden suchen.

c) Implementieren Sie die Methode `K getNeighbour(K key)`, um eine Nachbar-Suche durchzuführen. Der zurückgegebene Schlüssel ist der nächst größere Schlüssel nach `key` im Baum.

d) Implementieren Sie die Methode `Iterator<K> rangeQuery(K start, K end)`, um eine Bereichsanfrage zu realisieren. Dazu soll ein Iterator zurückgegeben werden, der über alle Schlüssel k mit $start \leq k \leq end$ in sortierter Reihenfolge läuft.

Aufgabe 9.3: Laufzeiten der Datenstrukturen* (1+1+2+1) **(5 Punkte)**

Erstellen Sie eine Klasse `PerformanceTest`, in der Sie die folgenden Teilaufgaben lösen. Messen und vergleichen Sie die Laufzeiten in jeder Teilaufgabe zwischen den folgenden Datenstrukturen:

- entweder Ihre Implementierung des **2-3-4-Baums**, oder die in der Vorlage bereitgestellte `SkipList` Implementierung.
- die Klasse `java.util.TreeSet<K>`, für einen **Rot-Schwarz-Baum**.
- die Klasse `java.util.HashSet<K>`, für **Hashing mittels Verkettung**.
- die Klasse `java.util.LinkedList<K>`, für **einfache Listen**.

Variieren Sie in Ihren Experimenten die Anzahl n an Daten in der Datenstruktur. Verwenden Sie ausreichend große Zahlen für n (z.B. 100000). **Hinweis:** Lassen Sie jedes Experiment (Teilaufgabe) mindestens 3-Mal durchlaufen und mitteln Sie die gemessene Zeit.

a) Fügen Sie zufällige Integer-Werte (ohne Duplikate) ein und messen Sie die Laufzeiten.

b) Suchen Sie mindestens `10%` vorhandene und mindestens `10%` nicht vorhandene Schlüssel aus dem Datenbestand aus a).

c) Messen Sie die Laufzeit der Nachbar-Suche. Überlegen Sie sich, wie diese auf Hash-Verfahren realisierbar sind. Verwenden Sie mindestens `5%` vorhandene und mindestens `5%` nicht vorhandene Nachbarn aus dem Datenbestand aus a).

d) Plotten (graphisch darstellen) Sie Ihre Ergebnisse in Abhängigkeit von n und geben Sie diese als PDF mit ab.

*** Aufgabe 9.3 ist für Lehramtsstudierende optional.**