Ludwig-Maximilians-Universität München
Institut für Informatik
PD Dr. Matthias Renz
Dr. Erich Schubert, Daniel Kaltenthaler

München, 17.6.2015

Algorithmen und Datenstrukturen

SS 2015

Übungsblatt 9: B*-Bäume, Hashing

Besprechung: 29.6.– 3.7.

Abgabe aller mit **Punkten** versehenen Aufgaben bis 28.6.2015 17:35

Aufgabe 9-1 B*-Bäume — Tutoraufgabe

(a) Konstruieren Sie einen (zu Beginn leeren) B*-Baum der Ordnung 3 durch Einfügen der folgenden Schlüssel: 47, 23, 35, 66, 7, 4, 71, 12, 55, 2, 1, 9, 10, 25, 39, 42, 91.

Zeichnen Sie den jeweils resultierenden Baum nach der 9., 12., 14. und letzten Einfügeoperation.

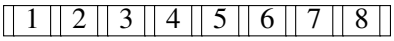
(b) Löschen Sie aus dem entstandenen B*-Baum folgende Schlüssel: 1, 2, 35, 7, 25, 42, 55, 91, 12.

Zeichnen Sie den jeweils resultierenden Baum nach der 1., 3., 5., 7. und letzten Löschung.

(c) Man betrachte den Splitalgorithmus beim Einfügen in einen B*-Baum. Wodurch wird das Ausgleichen mit dem Bruderknoten beim Einfügen notwendig ?

Aufgabe 9-2 B*-Bäume — 7 Punkte

Gegeben sei der folgende B*-Baum der Ordnung m:



(a) Angenommen die Wurzel ist maximal gefüllt.
Was ist dann die Ordnung m des B*-Baumes? Begründen Sie ihre Antwort.

(b) Fügen Sie in diesen Baum die folgenden Schlüssel (in dieser Reihenfolge) ein:
9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19,.
Zeichnen Sie den Baum nach dem Einfügen der Zahlen 9, 12, 14, 17, 19

Prüfen Sie nach jedem Schritt, ob es sich um einen korrekten B*-Baum handelt. Wenn Sie mit einem offensichtlich inkorrekten B*-Baum fortfahren, wird der Rest der Aufgabe mit 0 Punkten gewertet (kein "Folgefehler").

(c) Stellen Sie sich vor, Sie würden nun in aufsteigener Reihenfolge alle weiteren Zahlen aus $\mathbb{N}$ in diesen Baum einfügen. Zeigen Sie, gegen welchen Wert b die durchschnittliche Speicherplatzausnutzung der Blattknoten konvergiert (Sie müssen weder innere Knoten, noch die Wurzel betrachten)?

Aufgabe 9-3 Offenes Hashing **Tutoraufgabe**

Gegeben sei die Hashfunktion $h(k) = k \mod 7$ mit der folgenden Hashtabelle:

0	1	2	3	4	5	6

Sortieren Sie die folgenden numerischen und nicht-numerischen Schlüssel mittels offener Divisionsmethode in je eine solche Hashtabelle ein. Geben Sie jeweils die gefüllten Hashtabellen an.

(a) 1, 7, 13, 15, 14, 5

(b) a, k, v, d, y (mit $ORD(a) = 1$ usw.)

Aufgabe 9-4 Hashing mit geschlossener Kollisionsstrategie **5 Punkte**

Es sei die Hashfunktion $h(k) = k \mod 11$ gegeben. Zur Kollisionsbehandlung verwende man die geschlossene Kollisionsstrategie "Direkte Verkettung mit Verschmelzen".

(a) Welchen Wertebereich hat diese Hashfunktion? Welche Größe muss eine Hashtabelle für diese Kollissionsstrategie daher haben?

(b) Sortieren Sie die folgenden Schlüssel 42, 14, 39, 20, 1, 23, 40, 31, 13 in folgende Hashtabelle ein:

0	1	2	3	4	5	6	7	8	9	10	11

Zeichnen Sie die Hashtabelle dabei nach der 3., 5., 6. und letzten Einfügung.

(c) Suchen Sie in dieser Hashtabelle den Schlüssel 9.
Geben Sie die durchsuchten Speicherzellen an, und das Suchergebnis.

Aufgabe 9-5 Lineares Sondieren **7 Punkte**

Gegeben Sei folgende (unvollständige) Implementierung einer geschlossenen Hashtabelle mit der Kollissionsstrategie “Lineares Sondieren”. Sie sollen diese Klasse um die fehlende Funktionalität ergänzen.

Beachten Sie, dass diese Implementierung von dem in der Vorlesung vorgestellten Code abweicht, um den Speicherverbrauch zu reduzieren. Transferieren Sie die algorithmischen Überlegungen auf diese reduzierte Datenstruktur.

```
public class LinearesSondieren {
  /** Speicher */
  private int[] daten;

  /** Reservierter Wert */
  private static final int FREE = Integer.MIN_VALUE;

  /** Exception, falls die Tabelle voll ist */
  public static class FullException extends RuntimeException {
    /** Constructor */
    public FullException() {
      super("Tabelle ist voll.");
    }
  }

  /** Constructor
   * @param size Tabellengroesse
   */
  public LinearesSondieren(int size) {
    this.daten = new int[size];
    java.util.Arrays.fill(this.daten, FREE);
  }

  /* Aktuellen Inhalt ausgeben (Debug) */
  private void output() {
    for(int i : daten) {
      System.out.print(i == FREE ? "___ " : String.format("%3d ", i));
    }
    System.out.println();
  }

  /* Einfache Hashfunktion. Nicht besonders gut! */
  private int hash(int val) {
    val = val % daten.length;
    // Vorsicht: Java Modulo kann negative Werte liefern!
    return (val < 0) ? (val + daten.length) : val;
  }

  /* Einen Schluessel in die Hashtabelle einfuegen */
  public void insert(int val) {
    if (val == FREE) { // Sonderfall verbieten
      throw new RuntimeException("Der Schluessel "+FREE
        + " darf in dieser Implementierung nicht verwendet werden!");
    }
    final int h = hash(val);
    // TODO: implementieren
    // Falls die Tabelle voll ist: throw new FullException();
  }

  /** Schluessel in der Tabelle suchen */
  public boolean contains(int val) {
```

```
    final int h = hash(val);
    // TODO: implementieren
  }

  // TODO: Funktionstest als main()-Methode implementieren
}
```

Zum Berechnen des Hashcodes verwenden Sie die vorgegebene Methode `hash(int)`.

(a) Wie wird in dieser Datenstruktur gespeichert, welche Zellen belegt sind?
Welche Einschränkung ergibt sich daraus?

(b) Diese Implementierung hat keinen Zähler r, wie viele Elemente aktuell in der Tabelle gespeichert sind. Überlegen Sie sich, wie Sie eine volle Tabelle ohne diesen Zähler erkennen können.

(c) Vervollständigen Sie die `insert(int)` Methode, um Elemente in die Hashtabelle einzufügen.
Falls die Tabelle voll ist, *müssen* Sie eine `FullException` werfen.
Falls der Schlüssel bereits vorhanden ist, soll *keine Änderung* an der Tabelle erfolgen, aber dies ist kein Fehler (die Datenstruktur soll eine Menge repräsentieren).

(d) Vervollständigen Sie die `contains(int)` Methode, um zu testen ob Elemente in der Tabelle vorhanden sind. Verwenden Sie den Algorithmus aus der Vorlesung, es ist nicht zulässig immer die komplette Tabelle linear zu durchsuchen!

(e) Fügen Sie eine `main(String[])` Methode zu Ihrer Klasse hinzu, mit der ihr Programm automatisch getestet werden kann. Diese Tests sollen alle wichtigen Fälle ihres Programmes abdecken. Die Methode `output()` können Sie dazu verwenden, den Inhalt ihrer Tabelle anzuzeigen.

(f) Testen Sie ihr Programm ausführlich. Fehler, die Sie aufgrund von zu einfachen Tests übersehen haben, müssen Sie verantworten.

Sie dürfen diese Klasse dazu nur an den vorgesehenen Stellen ändern.

Geben Sie Fragen (a) und (b) in einer separaten `.txt` Datei. Geben Sie (c)-(e) als `LinearesSondieren.java` ab. Klassen die nicht compilieren, oder nicht wie gefordert bearbeitet wurden, müssen von den Korrektoren mit 0 Punkten bewertet werden.