TU München, Fakultät für Informatik
Lehrstuhl III: Datenbanksysteme
Prof. Alfons Kemper, Ph.D.

Übung zur Vorlesung *Einführung in die Informatik 2 für Ingenieure (MSE)*

Christoph Anneser (anneser@in.tum.de)
http://db.in.tum.de/teaching/ss21/ei2/

Lösungen zu Blatt 7

Aufgabe 1: Hashtabellen

Fügen Sie in eine anfangs leere Hashtabelle mit Größe 8 nacheinander die folgenden Elemente ein: 4, 8, 16, 12, 14, 3, 2, 6, 5. Zur Kollisionsbehandlung soll lineares probing verwendet werden und als Hashfunktion soll die Identitätsfunction ($h(x) = x$) verwendet werden.

Lösung

Abbildung 1 zeigt die Hashtabelle nach dem einfügen der Werte.

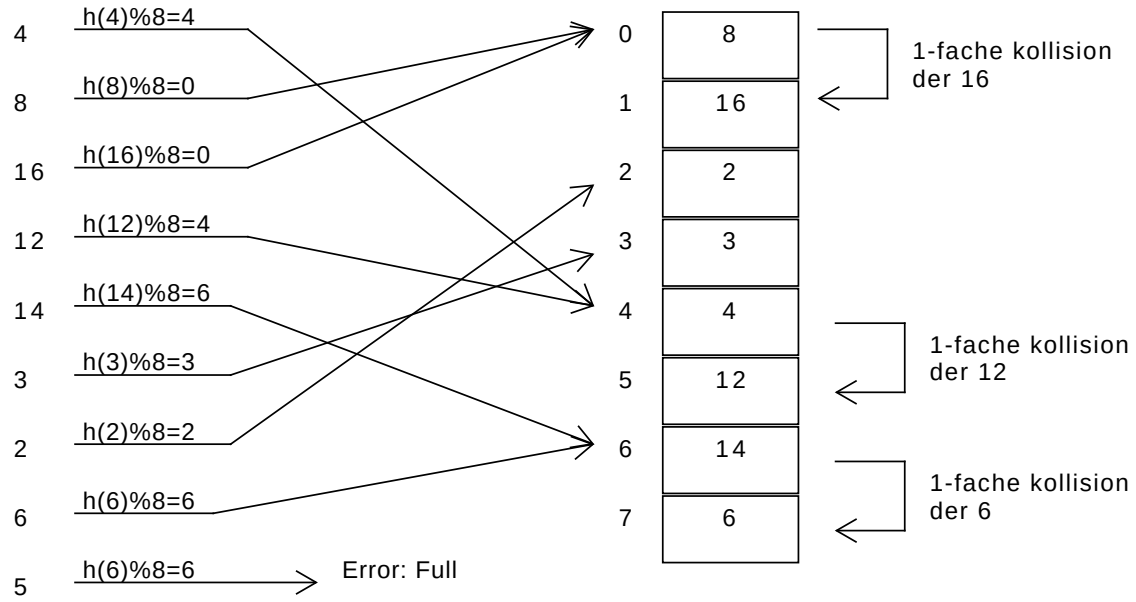

Abbildung 1: Hashtabelle nach dem Einfügen.

Aufgabe 2: Hashing in Java

Warum sollte man in Java, wenn man `equals()` überschreibt, auch `hashCode()` überschreiben?

Lösung

Überschreibt man nur eine der beiden Methoden kommt es zu unerwarteten Ergebnissen. Ein Beispiel sind Hashtabellen, die davon ausgehen, dass wenn zwei Objekte gleich sind (im Sinne von `equals()`), sie auch den gleichen Hashwert haben (`hashCode()`). Sonst kann es passieren, dass zwei „gleiche“ Objekte an verschiedene Stellen in der Hashtabelle landen.

```
1  import java.util.HashMap;
2
3  class Student {
4    String name;
5    Integer matrNr;
6
7    Student(String name, Integer matrNr) {
8      this.name = name;
9      this.matrNr = matrNr;
10   }
11
12   public int hashCode() {
13     final int prime = 31;            // Berechne den hashCode
           basierend
14     int result = 1;                  // auf den gleichen Attributen!
15     int result = 31 * result + name.hashCode();
16     return 31 * result + matrNr.hashCode();
17     return result;
18   }
19
20   public boolean equals(Object obj) {
21     if (obj == this)                 // Teste auf Referenzgleichheit
22       return true;
23     if (!(obj instanceof Student))   // Teste Klasse
24       return false;
25     Student other = (Student)obj;    // Teste Attribute
26     return name.equals(other.name) && matrNr.equals(other.matrNr);
27   }
28 }
29
30 class Equality {
31   public static void main(String[] args) {
32     HashMap<Student, Integer> map = new HashMap<Student, Integer>();
33     Student uliOriginal = new Student("Uli", 123456789);
34     Student uliKlon = new Student("Uli", 123456789);
35     map.put(uliOriginal, 11);
36     System.out.println(map.containsKey(uliKlon));  // Wertet zu "true
           " aus
37   }
38 }
```

Aufgabe 3: Rekursion mit Boxen

Gegeben sei eine Menge an Boxen. Jede Box hat eine Länge, eine Höhe und eine Breite. Sie sollen nun den höchst-möglichen Stapel an Boxen bauen bzw. herausfinden, wie hoch maximal gestapelt werden kann. Dabei muss jedoch beachtet werden, dass eine Box b_1 nur dann auf eine Box b_2 gestellt werden kann, wenn Länge, Höhe und Breite von b_1 kleiner sind als die von b_2.

Wir berechnen damit jedoch für manche Boxen die maximale Höhe mehrmals, wie können wir dies umgehen?

```
class Box implements Comparable<Box> {
    public Box(int id, int width, int heigth, int length) {
        this.id = id;
        this.width = width;
        this.height = heigth;
        this.length = length;
    }

    protected int id;
    protected int width;
    protected int height;
    protected int length;

    @Override
    public int compareTo(Box o) {
        if (this.width < o.width && this.length < o.length && this.height <
        ↪  o.height) return -1;
        return 1;
    }
}


public class Main {
    static int getMaxHeigth(Set<Box> remainingBoxes, Box lastBox) {
        if (remainingBoxes.isEmpty())
            return lastBox != null ? lastBox.height : 0;
                // create deep copy of hash set
        Set<Box> remainingBoxesCopy = new HashSet<>();
        remainingBoxesCopy.addAll(remainingBoxes);

        int maximalSubStackHeight = 0;
        for (Box box : remainingBoxes) {
            if (lastBox == null || box.compareTo(lastBox) < 0) {
                remainingBoxesCopy.remove(box);
                maximalSubStackHeight = max(getMaxHeigth(remainingBoxesCopy,
                ↪  box), maximalSubStackHeight);
                remainingBoxesCopy.add(box);
            }
        }
```

```
        return (lastBox != null ? lastBox.height : 0) + maximalSubStackHeight;
    }

    public static void main(String[] args) {
        Set<Box> boxes = new HashSet<>();
        boxes.add(new Box(1, 1, 2, 3));
        boxes.add(new Box(2, 4, 3, 3));
        boxes.add(new Box(3, 2, 2, 5));
        boxes.add(new Box(4, 1, 2, 1));
        boxes.add(new Box(5, 5, 5, 4));

        int maxHeigth = getMaxHeigth(boxes, null);
        System.out.println(maxHeigth);
    }
}
```

Aufgabe 4: Motivation von DBMS

Nennen Sie drei typische Probleme, die bei dem Verzicht auf ein Datenbankverwaltungssystem eintreten können. Überlegen Sie sich jeweils ein Beispiel bei dem das Problem auftritt.

Lösung

Im folgenden eine Liste von möglichen Problemen:

Redundanz. Ohne DBMS kommt es leicht zu Redundanz in den Daten und diese kann bei Veränderungen zu Inkonsistenzen führen. Ein DMBS garantiert die **Konsistenz** der Daten.

Beschränkte Zugriffsmöglichkeit. Verknüpfung der Daten ist ohne DBMS sehr schwer, da diese dann oft nicht konsistent modelliert und abgespeichert sind.

Probleme durch Mehrbenutzerbetrieb. Gleichzeitige Änderung der gleichen Daten durch verschiedene Benutzer führt schnell zu Anomalien. Ein DMBS garantiert die logische **Isolation** der Anfragen verschiedener Benutzer.

Datenverlust. Bei Speicherung der Daten in Dateien sind meist höchstens regelmäßige Backups möglich. Ein DBMS garantiert hingegen die **Dauerhaftigkeit** jeder durchgeführten Operation. Weiterhin werden Änderungen ganz oder gar nicht durchgeführt (**Atomarität**).

Integritätsverletzung. In der realen Welt gibt es komplexe Integritätsbedingungen, die sich bei Speicherung in Dateien leicht verletzen lassen.

Sicherheitsprobleme. Ein Beispiel ist der Datenschutz, da nicht alle Benutzer Zugriff auf alle Daten haben sollten. Ein DBMS bietet hierfür Rollen und Zugriffsrechte.

Hohe Entwicklungskosten. Bei der Eigenentwicklung der Datenspeicherung in einem Anwendungsprogramm erfindet man das Rad neu und muss alle oben genannten Probleme lösen. Dies führt zu erheblichen Entwicklungskosten. DBMS hingegen sind getestete Standardkomponenten, die man sehr einfach einsetzen kann.

Die vier wichtigsten Eigenschaften der Datenverarbeitung in einem DBMS haben eine leicht zu merkende Abkürzung: **ACID**. Dies steht für **A**tomicity, **C**onsistency, **I**solation und **D**urability.

Aufgabe 5: Terminologie

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

Abbildung 2: Professoren in der relationalen Modellierung

Beschreiben Sie die folgenden Begriffe der relationalen Modellierung. Verwenden Sie die Relation Professoren aus Abbildung 2 um Beispiele für die einzelnen Konzepte anzugeben.

Lösung

Hier eine kurze Beschreibung der Konzepte mit Beispielen:

Attribut. Ein Attribut ist eine Eigenschaft der Entitäten der Relation (entspricht einer Spalte, bei der Relation Professoren z.B. Name).

Schlüssel. Ein Schlüssel identifiziert ein Tupel eindeutig (z.B. die PersNr des Professors).

Relation. Eine Relation besteht aus Schema und Ausprägung (die gesamte Tabelle).

Domäne. Der Wertebereich eines Attributs (bei der PersNr z.B. alle `int` Zahlen).

Tupel. Eine Entität einer Relation (entspricht einer Zeile, z.B. Professor Russel).

Schema. Die Attribute einer Relation mit deren Typen (in diesem Fall `{[PersNr: integer, Name: varchar(30), Rang: varchar(2), Raum: integer]}`).

Ausprägung. Die Gesamtheit aller Tupel einer Relation (alle Zeilen bis auf die Kopfzeilen).