

Übungen zu Algorithmen

Wintersemester 2003/2004

Blatt 13

Aufgabe 13.1 (10 Punkte)

Implementieren Sie das Interface `java.lang.Comparable` (siehe Javadokumentation) in der Klasse `StudentMatrComparable`, so daß die Methode `compareTo` folgende Funktionalität hat:

```
/**
 * Vergleicht StudentMatrComparable Objekte.
 * Liefert einen Wert <0, =0 oder >0 je nachdem, ob die
 * Matrikelnummer dieses StudentMatrComparable kleiner, gleich
 * oder groesser der von a ist.
 * @param a Objekt mit dem verglichen werden soll.
 * @throws RuntimeException wenn a kein StudentMatrComparable ist.
 */
public int compareTo(Object a)
```

Leiten Sie dazu die Klasse `StudentMatrComparable` von `Student` aus `/home/ainf/Vorlesung` ab.

Musterlösung vom 28.01.2004:

```
/* ***** StudentMatrComparable.java ***** */
import AlgoTools.IO;

/**
 * Klasse StudentMatrComparable
 * Implementiert Methode aus Comparable, in der sich
 * dies Objekt mit dem Uebergebenen auf die fuer
 * Strings spezifische Weise vergleicht.
 *
 * @version 19.01.2004
 * @author Ralf Kunze, Olaf Mueller
 */

public class StudentMatrComparable extends Student implements Comparable {

    /**
     * Liefert ein StudentMatrComparable-Geruest, das ausser der uebergebenen
     * Matrikelnummer nichts enthaelt.
```

```

*
* @param Matrikelnummer, die das Student-Objekt erhalten soll.
*/
public StudentMatrComparable(int mat_nr) {
    super(null, null, 0, 0, 0, null, 0);
    next_mat_nr--; // Luecke in Matrikelnummern-
                  // Folge vermeiden
    // identisch zu: super.next_mat_nr--; da Klassenvariable vererbt wird!

    this.mat_nr = mat_nr; // Suchnummer eintragen
}

/**
* Liefert ein StudentMatrComparable-Objekt, das mit den uebergebenen Werten
* gefuellt ist und eine automatisch vergebene Matrikelnummer hat.
*
* @param vn Vorname des Studenten
* @param nn Nachname des Studenten
* @param t Tag des Geburtsdatums
* @param m Monat des Geburtsdatums
* @param j Jahr des Geburtsdatums
* @param fach Studienfach des Studenten
* @param jsb Jahr des Studienbeginns des Studenten
*/
public StudentMatrComparable(String vn, String nn, int t, int m, int j,
                             String fach, int jsb) {
    super(vn, nn, t, m, j, fach, jsb);
}

/**
* Vergleicht StudentMatrComparable Objecte.
* Liefert einen Wert <0, =0 oder >0 je nachdem, ob die
* Matrikelnummer dieses StudentMatrComparable kleiner, gleich
* oder groesser der von a ist.
*
* @param a Objekt mit dem verglichen werden soll.
* @throws RuntimeException wenn a kein StudentMatrComparable ist.
*/
public int compareTo(Object a) {

    if(! (a instanceof StudentMatrComparable)) // Ist a ein Objekt dieser Klasse?
        throw new RuntimeException ("StudentMatrComparable.compareTo:a("+a+") ist kein Stu

        // Differenz liefern
    return (mat_nr - ((StudentMatrComparable)a).mat_nr);
}

/**
* Liefert eine Stringrepraesentation dieses Objekts, die alle Datenfelder
* enthaelt.
*
* @return Stringrepraesentation eines Studenten-Objektes
*/

```

```

public String toString() {
    StringBuffer sb = new StringBuffer("Ich bin " + vorname + " ");
    sb.append(nachname + ", geb. am " + geb_datum + ", und habe ");
    sb.append(jsb + " mein " + fach + "-Studium begonnen! Matr.-Nr.:" + mat_nr);
    return sb.toString();
}
}

```

Aufgabe 13.2 (20 Punkte)

Benutzen Sie `/home/ainf/Vorlesung/SuchBaum.java`, um Studenten zu verwalten:

Erweitern Sie Ihre Klasse `StudentMatrComparable` um

- einen Konstruktor, der als Parameter lediglich eine Matrikelnummer erhält
Achtung: Bei der Klasse `Student` wird beim Erzeugen eines `Student` Objektes automatisch die Matrikelnummer erhöht. Dieser Effekt ist bei diesem Konstruktor nicht erwünscht. Ändern Sie die Klasse `Student` nicht.
- einen Konstruktor, der bis auf die Matrikelnummer (welche automatisch vergeben wird) alle Daten des Studenten einliest
- eine Methode `toString()`, die alle Daten des jeweiligen Studenten ausgibt.

Schreiben Sie eine Java-Klasse `Aufg13_2.java`, welche `StudentMatrComparables` in einem *Suchbaum* verwaltet. Es sollen `StudentMatrComparables` eingegeben und nach Instanzen der eingefügten Klasse gesucht werden können, wobei zwei Instanzen genau dann gleich sind, wenn ihre Matrikelnummern identisch sind. Um nach einer bestimmten Matrikelnummer zu suchen, soll ein `StudentMatrComparable` Objekt angelegt werden, welches lediglich die gesuchte Matrikelnummer enthält.

Instanzen sollen gelöscht und in aufsteigender Matrikelnummernreihenfolge, also *inorder*, ausgegeben werden können.

Zum Testen verwenden Sie eine Datei `Studenten.data`:

```
cat Studenten.data | java Aufg13_2
```

Musterlösung vom 28.01.2004:

```

/***** Aufg13_2.java *****/
import AlgoTools.IO;

/**
 * Studenten werden in einem Suchbaum verwaltet.
 * Es koennen Studenten eingegeben, angezeigt und
 * geloescht werden.
 *
 * @version 19.01.2004
 * @author Ralf Kunze, Olaf Mueller

```

```

*/

class Aufg13_2 {

    /**
     * liest einen Studenten ein
     * und fuegt ihn dann in den Baum ein.
     *
     * @param daten Suchbaum, in den ein Student eingefuegt wird.
     */
    private static void eingabe(SuchBaum daten) {
        boolean erfolg;
        String vn = IO.readString("Vorname      : ");
        String nn = IO.readString("Nachname     : ");
        int j     = IO.readInt  ("Geburtsjahr  : ");
        int m     = IO.readInt  ("Geburtsmonat: ");
        int t     = IO.readInt  ("Geburtstag   : ");
        String f  = IO.readString("Fach         : ");
        int jsb   = IO.readInt  ("Jahrgang     : ");

        StudentMatrComparable s = new StudentMatrComparable(vn, nn, t, m, j, f, jsb);

        erfolg = daten.insert(s);
        IO.print("Der Student wurde ");
        if(!erfolg) IO.print("nicht ");
        IO.println("aufgenommen.");
    }

    /**
     * gibt den Inhalt des Baumes inorder aus.
     *
     * @param daten Baum, dessen Inhalt ausgegeben werden soll.
     */
    private static void ausgabe(Baum daten) {
        if (!daten.empty()) {
            ausgabe(daten.left()); // falls Baum nicht leer,
            IO.println(daten.value()); // steige links ab
            ausgabe(daten.right()); // gib Student aus und
            // steige rechts ab
        }
    }

    /**
     * Loescht den Studenten, falls er schon eingegeben wurde.
     *
     * @param daten Loescht den Studenten in diesem Suchbaum
     */
    private static void loeschen(SuchBaum daten) {

        int mat_nr = IO.readInt("Matrikelnummer? ");
        StudentMatrComparable s = new StudentMatrComparable(mat_nr);
        // zu loeschender Studi
    }
}

```

```

    if (daten.delete(s))
        IO.println("Der Student wurde geloescht.");
    else
        IO.println("Der Student ist nicht vorhanden.");
}

/**
 * Sucht den Studenten und gibt gegebenenfalls den kompletten Datensatz aus.
 *
 * @param daten Sucht den Studenten in diesem Suchbaum.
 */
private static void suchen(SuchBaum daten) {

    int mat_nr = IO.readInt("Matrikelnummer? ");
    StudentMatrComparable s = new StudentMatrComparable(mat_nr);
    Comparable c = daten.lookup(s); // zu suchender Studi
    // Suchergebnis sichern

    if (c != null) { // wenn im Baum vorhanden
        IO.println("Der Student ist vorhanden: ");
        IO.println(c); // Kein Cast: Alle Objekte
        // koennen toString
    } // komplett ausgeben.
    else // wenn nicht vorhanden
        IO.println("Der Student ist nicht vorhanden.");
}

/***** Main *****/

public static void main(String[] argv) {

    SuchBaum daten = new SuchBaum(); // nimmt Studenten auf

    char wahl;

    do { // Menue ausgeben
        IO.println();
        IO.println("e : eingeben von Personendaten");
        IO.println("s : suchen von Personendaten");
        IO.println("l : loeschen von Personendaten");
        IO.println("a : ausgeben der Daten in aufsteigender Matrikelnummernreihenfolge");
        IO.println("x : exit");
        wahl = IO.readChar("Eingabe: ");

        switch (wahl){ // entsprechendes tun
            case 'e' : eingabe(daten);
                break;
            case 's' : suchen(daten);
                break;
            case 'l' : loeschen(daten);
                break;
            case 'a' : ausgabe(daten);
        }
    }
}

```

```

        break;
    case 'x' : break;
    default : IO.println("Falsche Eingabe.");
}
} while (wahl != 'x');
}
}

```

Aufgabe 13.3 (30 Punkte)

Erweitern Sie die Klasse BaumIO vom Aufgabenblatt 12 um die Möglichkeit die Knoten eines Baumes ebenenweise zu zählen.

Implementieren Sie dazu die Methode `knotenProEbene(Baum b)` und nutzen Sie das Konzept der Breitensuche (Skript 9.5).

```

/**
 * Gibt die Zahl der Knoten in den einzelnen Ebenen des Baumes
 * mit IO.println() aus.
 *
 * @param b Baum, von dem die Knotenanzahl pro Ebene
 *         bestimmt werden soll
 * @return Array mit den Knoten je Ebene (= Arrayindex)
 */
public static int [] knotenProEbene(Baum b)

```

Implementieren Sie zusätzlich eine Klasse BaumIOtest, in der mit der Methode `baueZufallsbaum()` ein Baum erzeugt wird. Die Klasse soll eine variable Eingabe der Knotenanzahl erlauben.

Weiterhin geben Sie den Baum mit der Methode `breitenSuche()` und liegend mit der Methode `printBaum(Baum b, int tiefe)` aus und führen dann die Methode `knotenProEbene()` aus.

Musterlösung vom 28.01.2004:

```

/***** BaumIO.java *****/

import AlgoTools.IO;

/**
 * Diese Klasse stellt Methoden zum Aufbau und zur Ausgabe
 * von Baeumen zur Verfuegung.
 *
 * @version 19.01.2004
 * @author Ralf Kunze, Olaf Mueller
 */
public class BaumIO {

```

```

/**
 * Die rekursive Klassenmethode printBaum gibt einen Binaerbaum liegend
 * aus, d.h. unten rechts beginnend und nach links fortfahrend.
 *
 * @param tiefe bestimmt die Einruecktiefe vom linken Rand.
 * @param b ist der auszugebende Baum
 */
public static void printBaum(Baum b, int tiefe) {

    if (! b.empty()) { // Baum eventuell leer ?
        printBaum(b.right(), tiefe+1); // rechten Teilbaum ausgeben

        for (int i=0; i<tiefe; i++) // entsprechend der Rekursions-
            IO.print("    "); // tiefe einruecken

        IO.println(b.value()); // Wurzelelement ausgeben

        printBaum(b.left(),tiefe+1); // linken Teilbaum ausgeben
    }
}

/**
 * Die rekursive Klassenmethode baueZufallsbaum konstruiert einen
 * binaeren Baum mit n Knoten, die jeweils zufaellige Kleinbuchstaben
 * enthalten. Jeder Knoten hat dabei zufaellig 0,1 oder 2 Soehne.
 *
 * @param n Anzahl der Knoten in dem Baum
 * @return der zufaellig zusammengebaute Baum
 */
public static VerweisBaum baueZufallsbaum(int n) {
    int l; // Knotenzahl im linken
            // Teilbaum

    if (n==0) // Rekursionsverankerung
        return new VerweisBaum(); // Vater ist Blatt: leeren
            // Baum liefern.

    else { // Falls noch was zu tun ist:
        l = Zufall.zufallsZahl(n); // Knotenzahl zufaellig
            // zwischen 0 und noch
            // verbleibender Knotenzahl
            // waehlen.

        // neuen Baum liefern mit
        // neuer Knoten
        VerweisBaum tmp = new VerweisBaum(); // Zufallswert setzen
        tmp.setValue(Zufall.zufallsBuchstabe()); // linken Teil erzeugen
        tmp.setLeft(baueZufallsbaum(l)); // rechten Teil erzeugen
        tmp.setRight(baueZufallsbaum(n-l-1)); // Knoten zurueckliefern
        return tmp;
    }
}

/**
 * Liefert die Hoehe des Baumes in Ebenen.
 */

```

```

* @param b Baum, von dem die Hoehe bestimmt werden soll
* @return Hoehe des Baumes
*/
public static int hoehe(Baum b) {
    if (b.empty()) // leerer Baum hat Hoehe 0
        return 0;
    else { // wenn nicht leer
        int l = hoehe(b.left()); // linke Hoehe und
        int r = hoehe(b.right()); // rechte Hoehe bestimmen
        return l > r ? l + 1 : 1 + r; // groessere ermitteln und
        // akt. Ebene aufaddieren
    }
}

/**
* Ermittelt die Anzahl der Knoten in den einzelnen Ebenen des Baumes.
*
* @param b Baum, von dem die Knotenanzahl pro Ebene bestimmt werden soll
* @return Array mit den Knoten je Ebene (= Arrayindex)
*/
public static int [] knotenProEbene(Baum b) {

    int [] ebenen = new int[hoehe(b)];
    Baum h; // Hilfsbaum
    int ebene = 0; // akt. Ebene
    int counter = 0; // Anzahl der Knoten in akt. Ebene

    Object marker = new Object();

    Schlange s = new VerweisSchlange(); // konstruiere eine Schlange

    if (!b.empty()) {
        s.enq(b); // lege uebergebenen Baum in Schlange
        s.enq(marker); // 1. Ebene fertig
    }
    else { // falls leerer Baum
        return ebenen;
    }

    while (!s.empty()) { // solange Schlange nicht leer

        if (s.front() == marker) { // Wenn Marker am "Kopf"
            // ist die Ebene
            // abgearbeitet

            if (counter!=0) { // Falls Knoten in der Ebene
                // waren, Anzahl ausgeben

                ebenen[ebene]=counter;
                s.enq(marker); // Neue Ebene Markieren
            }
            s.deq(); // alten Marker entfernen
            ebene++; // neue Ebene
        }
    }
}

```

```

        counter=0; // mit 0 Knoten bisher
    }
    else { // Ebene noch nicht komplett
        counter++; // neuer Knoten gefunden
        h = (Baum)s.front(); // besorge Baum aus Schlange
        s.deq(); // und entferne vordersten Eintrag
        if (!h.left().empty()) // die ev. vorhandenen Soehne
            s.enq(h.left()); // in die Schlange einfuegen
        if (!h.right().empty())
            s.enq(h.right());
    }
}
return ebenen;
}
}

```

```

/***** BaumIOTest.java *****/
import AlgoTools.IO;

```

```

/**
 * Klasse BaumIOTest testet verschiedene Operationen mit Baeumen:
 *
 * 1.) Aufbau eines zufaelligen binaeren Baumes
 *     mit der Suchbaumeigenschaft
 * 2.) Breitensuche
 * 3.) Ausgabe des Baumes liegend mit printBaum
 * 4.) Hoehe des Baumes
 * 5.) Knoten pro Ebene
 *
 * @author Ralf Kunze, Olaf Mueller
 * @version 19.01.2004
 */

```

```

public class BaumIOTest{

    public static void main (String args[]) {

        int n=0;

        do {
            n = IO.readInt("Wieviele Knoten soll der Baum haben (>=0)? ");
        }while (n<0);

        Baum b = BaumIO.baueZufallsbaum(n); // Aufbau binaeren Baumes

        IO.println("Breitensuche:");
        BreitenSuche.breitenSuche(b); // Breitensuche
        IO.println();

        IO.println("Baum liegend mit printBaum ausgegeben:");
        BaumIO.printBaum(b, 0); // Baum liegend ausgeben
    }
}

```

```

// Hoehe des Baumes
IO.println("Der Baum hat die Hoehe "+ BaumIO.hoehe(b) + ".");

IO.println("Zahl der Knoten in den einzelnen Ebenen:");
int [] ebenen = BaumIO.knotenProEbene(b); // Knoten pro Ebene

for (int i=0; i<ebenen.length;i++)
    IO.println("In Ebene "+(i+1)+" sind "+ebenen[i]+" Knoten vorhanden.");
}
}

```

Aufgabe 13.4 (25 Punkte)

Fügen Sie die folgenden Namen in der gegebenen Reihenfolge in einen *AVL-Baum* ein, der Strings lexikografisch vergleicht:

Legolas, Bilbo, Frodo, Arwen, Faramir, Pippin, Sam, Gandalf,
Saruman, Gollum, Gimli, Sauron, Aragorn, Boromir

Zeichnen Sie den AVL-Baum nach jedem Einfügen neu. Geben Sie in jedem Knoten auch die Balance an. Führen Sie ggf. die notwendigen Rotationen durch, und nennen Sie die Art der angewandten Rotation.

Musterlösung vom 28.01.2004:

```

/***** Aufg13_4.txt *****/
/* Datum:          21.01.2002
 * Autoren:        Ralf Kunze Olaf Mueller
 * EMail-Adressen: rkunze@informatik.uni-osnabrueck.de
 *                olaf@informatik.uni-osnabrueck.de
 */

```

```

/*****

```

1. Einfuegen von Legolas

```
Legolas(0)
```

```

/*****

```

2. Einfuegen von Bilbo

```
Legolas(-1)
    Bilbo(0)
```

```

/*****

```

3. Einfuegen von Frodo

LR-Rotation im Teilbaum mit Wurzel Legolas

```
    Legolas(0)
Frodo(0)
    Bilbo(0)
```

/*****/

4. Einfuegen von Arwen

```
    Legolas(0)
Frodo(-1)
    Bilbo(-1)
        Arwen(0)
```

/*****/

5. Einfuegen von Faramir

```
    Legolas(0)
Frodo(-1)
    Faramir(0)
    Bilbo(0)
        Arwen(0)
```

/*****/

6. Einfuegen von Pippin

```
        Pippin(0)
    Legolas(1)
Frodo(0)
    Faramir(0)
    Bilbo(0)
        Arwen(0)
```

/*****/

7. Einfuegen von Sam

RR-Rotation im Teilbaum mit Wurzel Legolas

```
        Sam(0)
    Pippin(0)
    Legolas(0)
Frodo(0)
    Faramir(0)
    Bilbo(0)
        Arwen(0)
```

/*****/

8. Einfuegen von Gandalf

```
        Sam(0)
      Pippin(-1)
        Legolas(-1)
          Gandalf(0)
Frodo(1)
      Faramir(0)
      Bilbo(0)
        Arwen(0)
```

/*****/

9. Einfuegen von Saruman

```
        Saruman(0)
      Sam(1)
      Pippin(0)
        Legolas(-1)
          Gandalf(0)
Frodo(1)
      Faramir(0)
      Bilbo(0)
        Arwen(0)
```

/*****/

10. Einfuegen von Gollum

LR-Rotation im Teilbaum mit Wurzel Legolas

```
        Saruman(0)
      Sam(1)
      Pippin(0)
        Legolas(0)
          Gollum(0)
            Gandalf(0)
Frodo(1)
      Faramir(0)
      Bilbo(0)
        Arwen(0)
```

/*****/

11. Einfuegen von Gimli

RL-Rotation im Teilbaum mit Wurzel Frodo

```
        Saruman(0)
      Sam(1)
```

```
    Pippin(1)
      Legolas(0)
Gollum(0)
      Gimli(0)
      Gandalf(1)
    Frodo(0)
      Faramir(0)
      Bilbo(0)
      Arwen(0)
```

/*****/

12. Einfuegen von Sauron
RR-Rotation im Teilbaum mit Wurzel Sam

```
      Sauron(0)
      Saruman(0)
      Sam(0)
    Pippin(1)
      Legolas(0)
Gollum(0)
      Gimli(0)
      Gandalf(1)
    Frodo(0)
      Faramir(0)
      Bilbo(0)
      Arwen(0)
```

/*****/

13. Einfuegen von Aragorn

```
      Sauron(0)
      Saruman(0)
      Sam(0)
    Pippin(1)
      Legolas(0)
Gollum(-1)
      Gimli(0)
      Gandalf(1)
    Frodo(-1)
      Faramir(0)
      Bilbo(-1)
      Arwen(-1)
      Aragorn(0)
```

/*****/

14. Einfuegen von Boromir

```

        Sauron(0)
    Saruman(0)
        Sam(0)
Pippin(1)
    Legolas(0)
Gollum(-1)
        Gimli(0)
        Gandalf(1)
    Frodo(-1)
        Faramir(-1)
            Boromir(0)
    Bilbo(0)
        Arwen(-1)
            Aragorn(0)

```

Aufgabe 13.5 (15 Punkte)

Schreiben Sie eine Klasse Aufg13_5 mit der Sie Strings einlesen und in einen AVLBaum einfügen.

Der AVLBaum soll nach jeder Veränderung (Einfügen oder Löschen) auf dem Bildschirm liegend ausgegeben werden (Analog zur Klasse AVLBaumTest aus Skript 9.7).

Können String-Objekte problemlos in einen AVLBaum eingefügt werden?

Musterlösung vom 28.01.2004:

```

/***** Aufg13_5.java *****/
import AlgoTools.IO;

/**
 * Klasse zum Testen des AVLBaums: Einfuegen
 * und Loeschen von Strings
 *
 * @version 19.01.2004
 * @author Ralf Kunze, Olaf Mueller
 */

public class Aufg13_5 {

    public static void main(String[] argv) {

        AVLBaum b = new AVLBaum();
        String s = IO.readString("String in AVL-Baum einfuegen (Loeschen: \\n): ");

        while (!s.equals("")) {
            // Wenn nur "return" zum
            // loeschen uebergehen
            if (b.insert(new String(s))) IO.println(s + " eingefuegt");
            else IO.println(s + " nicht eingefuegt");
            IO.println("AVL-Baum mit Balancen:");
            printAVLBaum(b, 0);
            s = IO.readString("String in AVL-Baum einfuegen (Loeschen: \\n): ");
        }
    }
}

```

```

IO.println();
s = IO.readString("String im AVL-Baum loeschen (Abbruch: \\n): ");

while (!s.equals("") ) {
    // loeschen bis nur
    // "return"
    if (b.delete(new String(s)) IO.println(s + " geloescht");
    else IO.println(s + " nicht geloescht");
    IO.println("AVL-Baum mit Balancen:");
    printAVLBaum(b, 0);
    s = IO.readString("String im AVL-Baum loeschen (Abbruch: \\n): ");
}

printAVLBaum(b, 0); // AVLBaum mit Balance
// ausgeben
}

/** Der AVL-Baum wird liegend mit Werten und Balancen ausgegeben. */
public static void printAVLBaum(Baum b, int tiefe) {
    if (! b.empty()) { // Wenn Baum nicht leer:
        printAVLBaum(b.right(), tiefe+1); // rechten Teilbaum ausgeben
        for (int i=0; i<tiefe; i++) // entsprechend der Rekursions-
            IO.print(" "); // tiefe einruecken
        IO.println((AVLBaum)b); // Wurzel und Balance ausgeben
        printAVLBaum(b.left(), tiefe+1); // linken Teilbaum ausgeben
    }
}
}

```