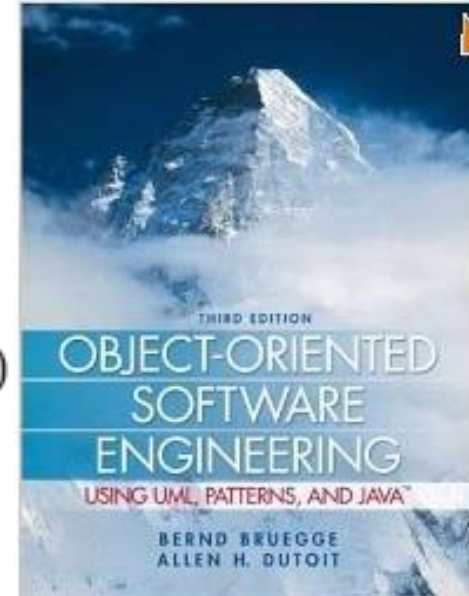


Einführung in die Informatik 2 für Ingenieure (MSE)

Teil 1

Objektorientierte Modellierung (in UML)
und Programmierung (in Java)



Teil 2

Datenbanksysteme eine Einführung
Alfons Kemper und Andre Eickler
Oldenburg Verlag, 8. Auflage, 2011

Oldenburg Verlag

Alfons Kemper, André Eickler
Datenbanksysteme

Eine Einführung

8. Auflage



Prof. Alfons Kemper, Ph. D.
Wolf Rödiger, M.Sc.

Kollektionen in Java

Aufzählungstypen, Generische Typen

- Wiederverwendbare Kollektionsklassen
 - Typparameter
 - Vordefinierte Kollektionen in der Java Collections Bibliothek
 - Insbesondere für die Modellierung von Assoziationen sinnvoll zu nutzen
 - Und für die Indexierung von Objekten
 - Schnelles Auffinden bei der Suche

Aufzählungstypen

- Notlösung:

```
1    public static final int MONTH_JAN = 1;
2    public static final int MONTH_FEB = 2;
3    ...
4    public static final int MONTH_NOV = 11;
5    public static final int MONTH_DEC = 12;
```

Aufzählungstypen

- besser:

```
1 public enum Month { JAN, FEB, MAR, APR, MAY, JUN,  
2     JUL, AUG, SEP, OCT, NOV, DEC };
```

```

1 public class Enums {
2     public enum Month {
3         JAN (31, -2.2), FEB (28, -0.8), MAR (31, 3.1),
4         APR (30, 9.0), MAY (31, 12.7), JUN (30, 15.9),
5         JUL (31, 20.1), AUG (31, 17.1), SEP (30, 15.4),
6         OCT (31, 7.8), NOV (30, 3.1), DEC (31, -0.8);
7
8     private final int    days;
9     private final double avgTemperature;
10
11     Month(int days, double avgTemperature) {
12         this.days = days;
13         this.avgTemperature = avgTemperature;
14     }
15
16     public int    days()          { return days; }
17     public double avgTemperature() { return avgTemperature; }
18
19     public double fractionOfYear() {
20         return days / 365.0;
21     }
22 };
23
24 public static void prettyPrint(Month m) {
25     System.out.println("Monat:_" + m + ",_Anzahl_Tage:_" + m.days() +
26         ",_Anteil_am_Jahr:_" + m.fractionOfYear());
27 }
28
29 public static void main(String[] args) {
30     prettyPrint (Month.JAN);
31
32     for (Month m : EnumSet.range(Month.FEB, Month.MAY))
33         System.out.println(m);
34 }
35 }

```

```
1     Month m = Month.JAN;
2
3     switch (m) {
4         case SEP:
5         case OCT:
6         case NOV:
7         case DEC:
8         case JAN:
9         case FEB:
10        case MAR:
11        case APR:
12            System.out.println("kalt");
13            break;
14        case MAY:
15        case JUN:
16        case JUL:
17        case AUG:
18            System.out.println("warm");
19            break;
20    }
```

Generische Klassen: Motivation

```
public class QuaderStack {  
    private Quader[] elements;  
    private int cardinality = 0;  
  
    public QuaderStack(int capacity) {  
        elements = new Quader[capacity];  
    }  
    public void push(Quader e) {  
        elements[cardinality++] = e;  
    }  
    public Quader pop() {  
        return elements[--cardinality];  
    }  
    public boolean isEmpty() {  
        return (cardinality == 0);  
    }  
}
```

```
1 public class ZylinderStack {  
2     private Zylinder[] elements;  
3     private int cardinality = 0;  
4  
5     public ZylinderStack(int capacity) {  
6         elements = new Zylinder[capacity];  
7     }  
8     public void push(Zylinder e) {  
9         elements[cardinality++] = e;  
10    }  
11    public Zylinder pop() {  
12        return elements[--cardinality];  
13    }  
14    public boolean isEmpty() {  
15        return (cardinality == 0);  
16    }  
17 }
```

Generisch ... aber nicht typsicher

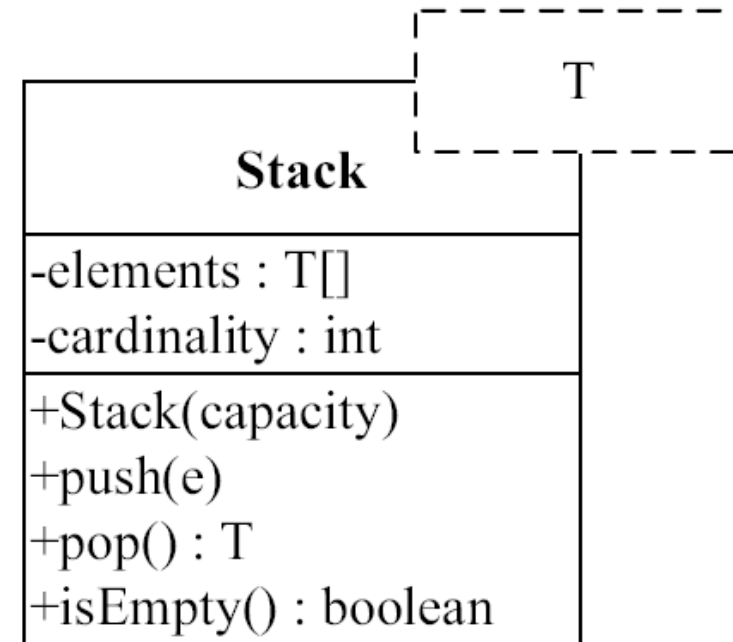
```
1 public class Stack {
2     private Object[] elements;
3     private int cardinality = 0;
4
5     public Stack(int capacity) {
6         elements = new Object[capacity];
7     }
8     public void push(Object e) {
9         elements[cardinality++] = e;
10    }
11    public Object pop() {
12        return elements[--cardinality];
13    }
14    public boolean isEmpty() {
15        return (cardinality == 0);
16    }
17 }
```


Nutzung ... durch type casting

```
1 // Quader q anlegen
2 // ...
3 QuaderStack myQuaderStack = new QuaderStack(5);
4 myQuaderStack.push(q);
5 Quader q2 = myQuaderStack.pop();
6
7 Stack myQuaderStack2 = new Stack(5);
8 myQuaderStack2.push("Dies_führt_später_zu_einem_Fehler!");
9 myQuaderStack2.push(q);
10 Quader q3 = (Quader)myQuaderStack2.pop();
11 Quader q4 = (Quader)myQuaderStack2.pop();
```

Generische Typen in Java und UML

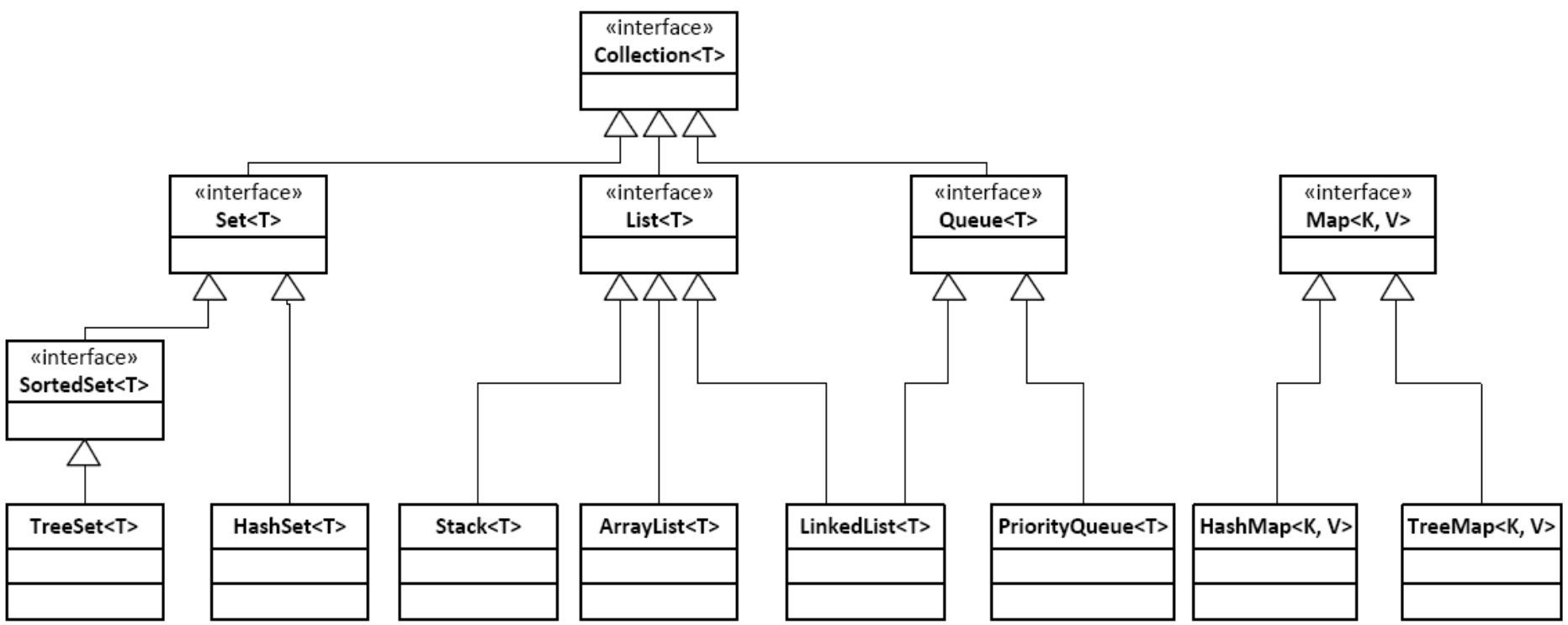
```
1 public class Stack<T> {
2     private T[] elements;
3     private int cardinality = 0;
4
5     public Stack(int capacity) {
6         elements = (T[])new Object[capacity];
7     }
8     public void push(T e) {
9         elements[cardinality++] = e;
10    }
11    public T pop() {
12        return elements[--cardinality];
13    }
14    public boolean isEmpty() {
15        return (cardinality == 0);
16    }
17 }
```



Nutzung

```
1 // Quader q anlegen
2 // ...
3 Stack<Quader> myQuaderStack = new Stack<Quader>(5);
4 myQuaderStack.push(q);
5 // myQuaderStack.push("Dies würde der Compiler beim
6 Quader q2 = myQuaderStack.pop();
```

Das Java Collection Framework



```

7 class PhoneBook {
8     // Map for the standard lookup
9     TreeMap<String, Integer> nameToNumber;
10    // Map for the reverse lookup
11    HashMap<Integer, String> numberToName;
12
13    // Constructor
14    public PhoneBook() {
15        nameToNumber = new TreeMap<String, Integer>();
16        numberToName = new HashMap<Integer, String>();
17    }
18
19    // Add an entry to the phone book
20    void addEntry(String name, Integer phoneNumber) {
21        nameToNumber.put(name, phoneNumber);    // O(log(n))
22        numberToName.put(phoneNumber, name);    // O(1)
23    }
24
25    // Standard lookup: get the phone number for a name
26    Integer lookup(String name) {
27        return nameToNumber.get(name);          // O(log(n))
28    }
29
30    // Reverse lookup: get the name for a phone number
31    String reverseLookup(Integer phoneNumber) {
32        return numberToName.get(phoneNumber);  // O(1)
33    }
34
35    // Get all entries of the phone book whose names lie in the given range
36    Set<Map.Entry<String, Integer>> rangeLookup(String from, String to) {
37        return nameToNumber.subMap(from, to).entrySet();
38    }

```

Nutzungsbeispiele

```
// Executable main method
public static void main(String[] args) {
    // Create the phone book
    PhoneBook phoneBook = new PhoneBook();
    phoneBook.addEntry("Maus, Micky", 4711);
    phoneBook.addEntry("Duck, Donald", 1234);
    phoneBook.addEntry("Maus, Minni", 1704);
    phoneBook.addEntry("Kolumbus, Christoph", 1492);
    // Lookup

    println("Donald's number: " + phoneBook.lookup("Duck, Donald"));
    println("1492 belongs to " + phoneBook.reverseLookup(1492));
    for (Map.Entry<String, Integer> entry
        : phoneBook.rangeLookup("Maier", "Meier")) {
        println(entry.getKey() + ": " + entry.getValue());
    }
}
```

Nutzung für die Modellierung von Assoziationen

```
1  import java.util.Set;
2  import java.util.HashSet;
3
4  public class Student {
5      public String name;
6      public Set<Vorlesung> vorlesungen;
7      // ...
8      vorlesungen = new HashSet<Vorlesung> ();
9      if (!vorlesungen.contains(grundzuege)) {
10         vorlesungen.add(grundzuege);
11     }
```

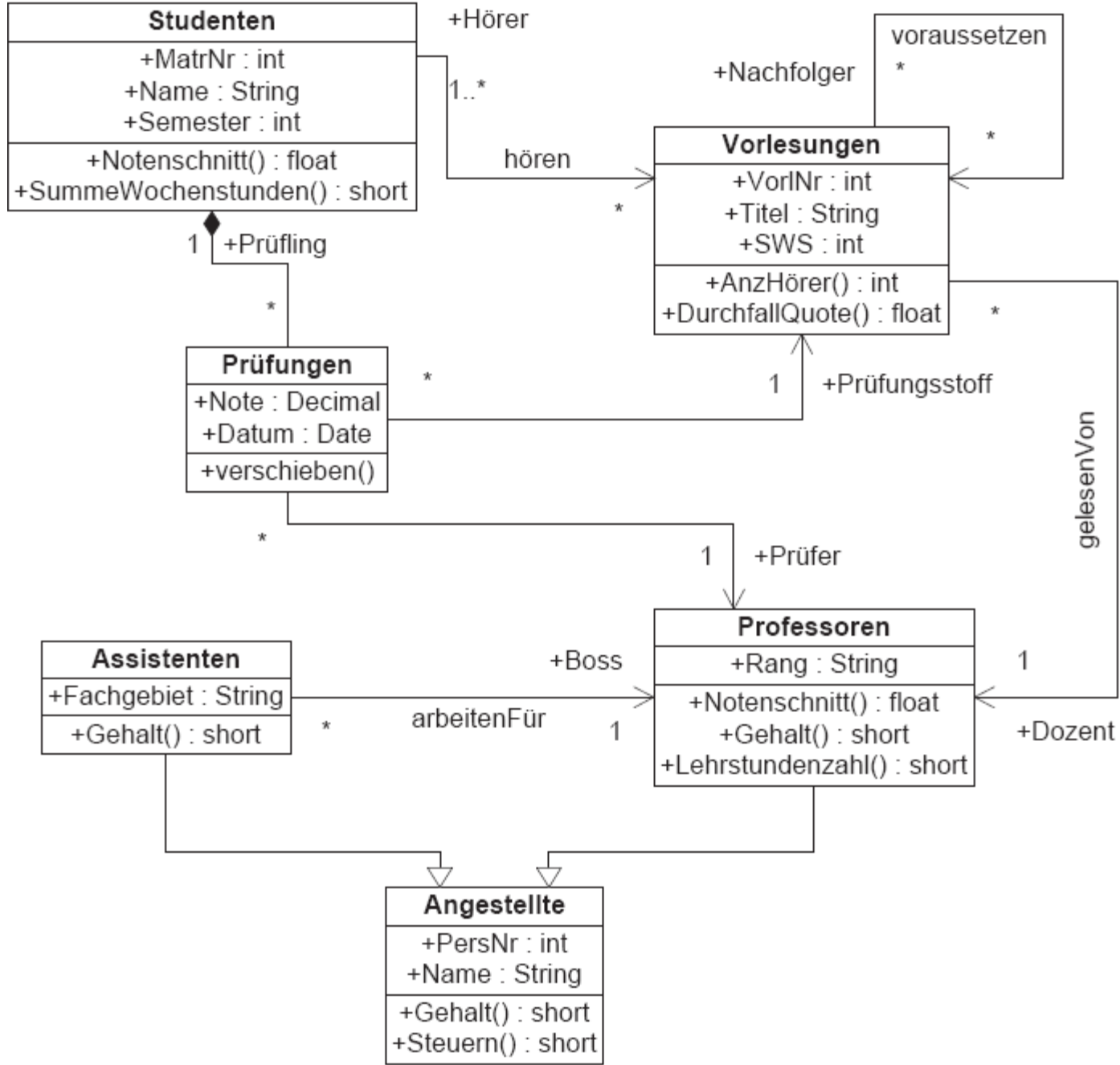
Wrapper für Sorten/Werte

```
1 Stack<Integer> myIntegerStack = new Stack<Integer>(10);
2 int fortyseveneleven = 4711;
3 myIntegerStack.push(new Integer(fortyseveneleven)); // Einp
4 Integer surprise = myIntegerStack.pop();           // Imme
5 int unwrapped = surprise.intValue();              // Ausp
```


Auto-Boxing

```
Stack<Integer> myIntegerStack = new Stack<Integer>(10);  
int fortyseveneleven = 4711;  
myIntegerStack.push(fortyseveneleven); // Automatisches  
int myInt = myIntegerStack.pop();      // Automatisches
```

Modellierungs-Beispiel: Universität



```
1 import java.util.Set;
2 import java.util.HashSet;
3
4 public class Student {
5     public int matrNr;
6     public String name;
7     public int semester;
8     public Set<Vorlesung> vorlesungen;
9     public Set<Pruefung> pruefungen;
10
11     public Student(int matrNr, String name, int semester) {
12         this.matrNr = matrNr;
13         this.name = name;
14         this.semester = semester;
15
16         vorlesungen = new HashSet<Vorlesung>();
17         pruefungen = new HashSet<Pruefung>();
18     }
19
20     public void belegeVorlesung(Vorlesung vorlesung) {
21         vorlesung.erhoeheAnzahlHoerer();
22         vorlesungen.add(vorlesung);
23     }
24
25     public void pruefen(Pruefung pruefung) {
26         pruefungen.add(pruefung);
```

```
20 public void belegeVorlesung (Vorlesung vorlesung) {
21     vorlesung.erhoeheAnzahlHoerer ();
22     vorlesungen.add(vorlesung);
23 }
24
25 public void pruefen (Pruefung pruefung) {
26     pruefungen.add(pruefung);
27 }
28
29 public float notenschnitt () {
30     float durchschnittsNote = 0;
31     for (Pruefung p : pruefungen) {
32         durchschnittsNote += p.note/pruefungen.size ();
33     }
34     return durchschnittsNote;
35 }
36
37 public short summeWochenstunden () {
38     short summeSWS = 0;
39     for (Vorlesung v : vorlesungen) {
40         summeSWS += v.sws;
41     }
42     return summeSWS;
43 }
44 }
```

```

1 import java.util.Calendar;
2
3 public class Pruefung {
4     public double note;
5     public Calendar datum;
6     public Student pruefling;
7     public Vorlesung pruefungsstoff;
8     public Professor pruefer;
9
10    private boolean bewertet;
11
12    public Pruefung(Student student, Vorlesung vorlesung, Professor professor,
13                    Calendar termin) {
14        this.pruefling = student;
15        this.pruefungsstoff = vorlesung;
16
17        this.pruefer = professor;
18        this.datum = termin;
19        bewertet = false;
20    }
21
22    public void bewerten(double note) {
23        if (!bewertet) {
24            bewertet = true;
25            this.note = note;
26            pruefling.pruefen(this);
27            pruefungsstoff.pruefen(this);
28            pruefer.pruefen(this);
29        }
30    }

```

```
1 import java.util.Set;
2 import java.util.HashSet;
3
4 public class Vorlesung {
5     public int vorlNr;
6     public String titel;
7     public int sws;
8     public Professor dozent;
9     public Set<Vorlesung> voraussetzungen;
10
11     private int anzahlHoerer;
12     private int anzahlPruefungen;
13     private int anzahlDurchgefallen;
14
15     public Vorlesung(int vorlNr, String titel, int sws, Professor dozent) {
16         this.vorlNr = vorlNr;
17         this.titel = titel;
18         this.sws = sws;
19         this.dozent = dozent;
20
21         voraussetzungen = new HashSet<Vorlesung>();
22
23         dozent leseVorlesung(this);
24     }
25
```

```
26 public void pruefen(Pruefung pruefung) {
27     if (pruefung.note > 4.0) {
28         anzahlDurchgefallen++;
29     }
30     anzahlPruefungen++;
31 }
32
33 public void erhoeheAnzahlHoerer() {
34     anzahlHoerer++;
35 }
36
37 public int anzahlHoerer() {
38     return anzahlHoerer;
39 }
40
41 public float durchfallQuote() {
42     return (float)anzahlDurchgefallen/anzahlPruefungen;
43 }
44 }
```

```
16     this.pruefer = professor;
17     this.datum = termin;
18     bewertet = false;
19 }
20
21 public void bewerten(double note) {
22     if (!bewertet) {
23         bewertet = true;
24         this.note = note;
25         pruefling.pruefen(this);
26         pruefungsstoff.pruefen(this);
27         pruefer.pruefen(this);
28     }
29 }
30
31 public void verschieben(Calendar neuesDatum) {
32     if (datum.compareTo(Calendar.getInstance()) > 0) {
33         datum = neuesDatum;
34     }
35 }
36 }
```



```
1 public class Angestellter {
2     public int persNr;
3     public String name;
4
5     public Angestellter(int persNr, String name) {
6         this.persNr = persNr;
7         this.name = name;
8     }
9
10    public int gehalt() {
11        return 2000;
12    }
13
14    public int steuern() {
15        return gehalt()/2;
16    }
17 }
```

```

1 public class Professor extends Angestellter {
2     public enum Rang {
3         C1, C2, C3, C4
4     }
5
6     public Rang rang;
7
8     private short lehrstunden;
9     private int notenAnzahl;
10    private int notenSumme;
11
12    public Professor(int persNr, String name, Rang rang) {
13        super(persNr, name);
14        this.rang = rang;
15    }
16
17    public void leseVorlesung(Vorlesung vorlesung) {
18        lehrstunden += vorlesung.sws;
19    }
20
21    public short lehrstundenzahl() {
22        return lehrstunden;
23    }
24
25    public void pruefen(Pruefung pruefung) {
26        notenAnzahl++;
27        notenSumme += pruefung.note;
28    }
29
30    public float notenschnitt() {
31        return (float)notenSumme/notenAnzahl;
32    }
33
34    public int gehalt() {

```

```
35     int gehalt;
36     switch (rang) {
37         case C1: gehalt = 3000;
38             break;
39         case C2: gehalt = 3200;
40             break;
41         case C3: gehalt = 3400;
42             break;
43         case C4: gehalt = 3600;
44             break;
45         default: gehalt = 3000;
46             break;
47     }
48     return gehalt;
49 }
50 }
```

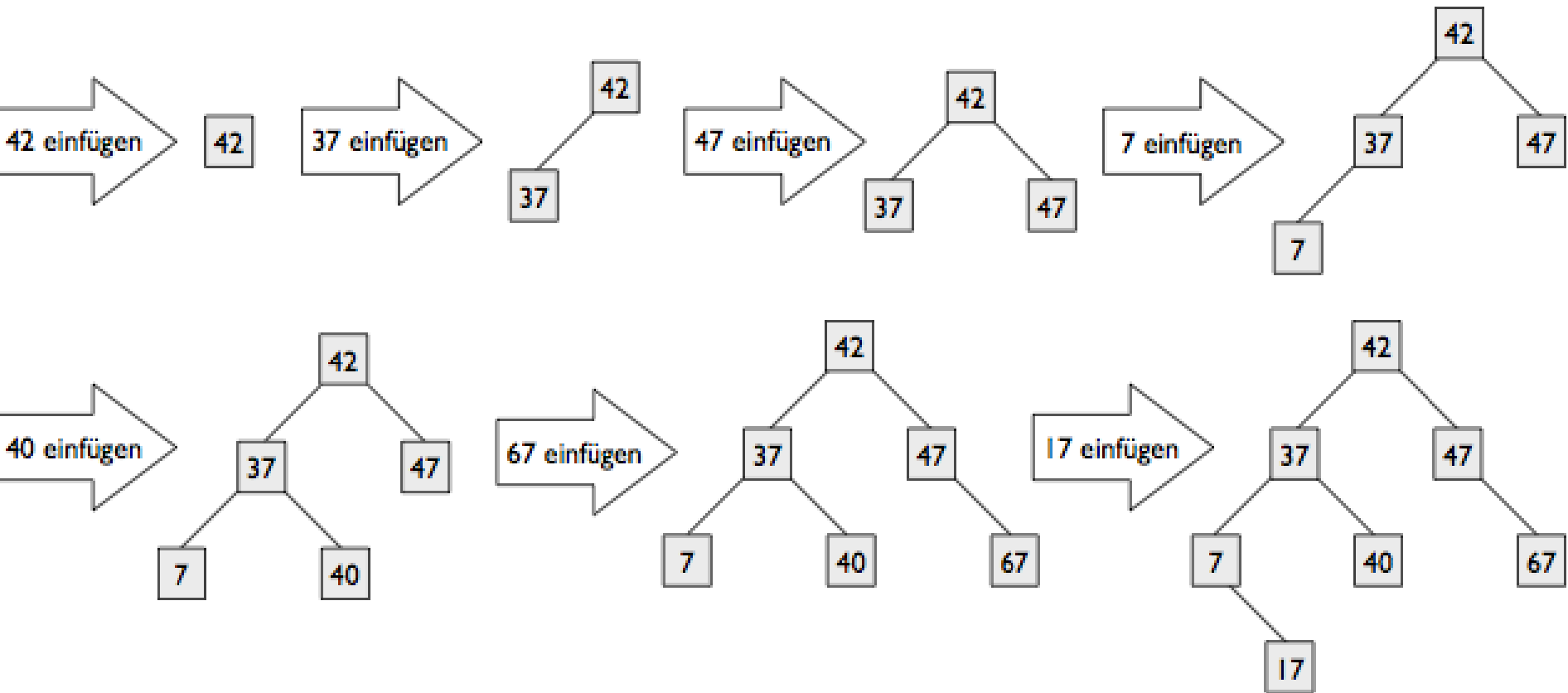
```
1 public class Assistent extends Angestellter {
2     public String fachgebiet;
3     public Professor boss;
4
5     public Assistent(int persNr, String name, String fachgebiet, Professor boss) {
6         super(persNr, name);
7         this.fachgebiet = fachgebiet;
8         this.boss = boss;
9     }
10
11     public int gehalt() {
12         return 2500;
13     }
14 }
```

Datenstrukturen für Kollektionen: Suchbäume und Hashing

- Suchbäume haben logarithmische Höhe
 - Suche kostet dann $O(\log N)$
 - N Elemente im Suchbaum
 - Bei 10.000.000.000 Einträge nicht zu vernachlässigen
 - Unterstützt auch Bereichsanfragen
 - TreeSet und TreeMap

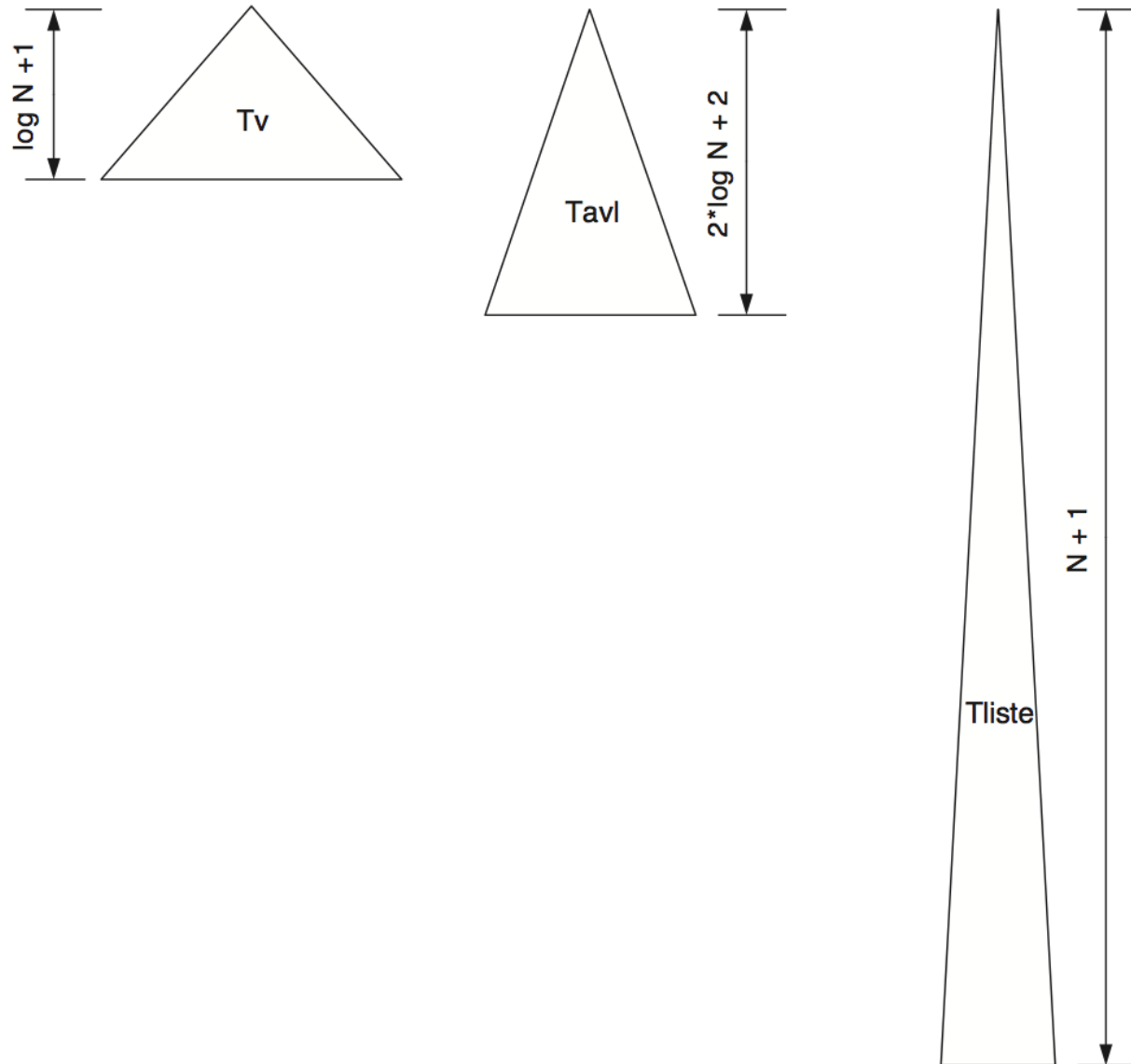
- Hashing ist unabhängig von der Anzahl der Elemente
 - $O(1)$ Suchkosten
 - Egal ob 200 oder 10.000.000.000 Einträge indexiert werden
 - Aber nur Punktanfragen (exact match)
 - HashSet und HashSet

Binäre Suchbäume



Problem: Degenerierter Suchbaum

Lösung: balancierter AVL-Baum



AVL-Baum: Balancierung während des Einfügens

- Höhe des linken Teilbaums unterscheidet sich von der Höhe des rechten Teilbaums um maximal 1

7.3 AVL-Bäume: Balancierte binäre Suchbäume

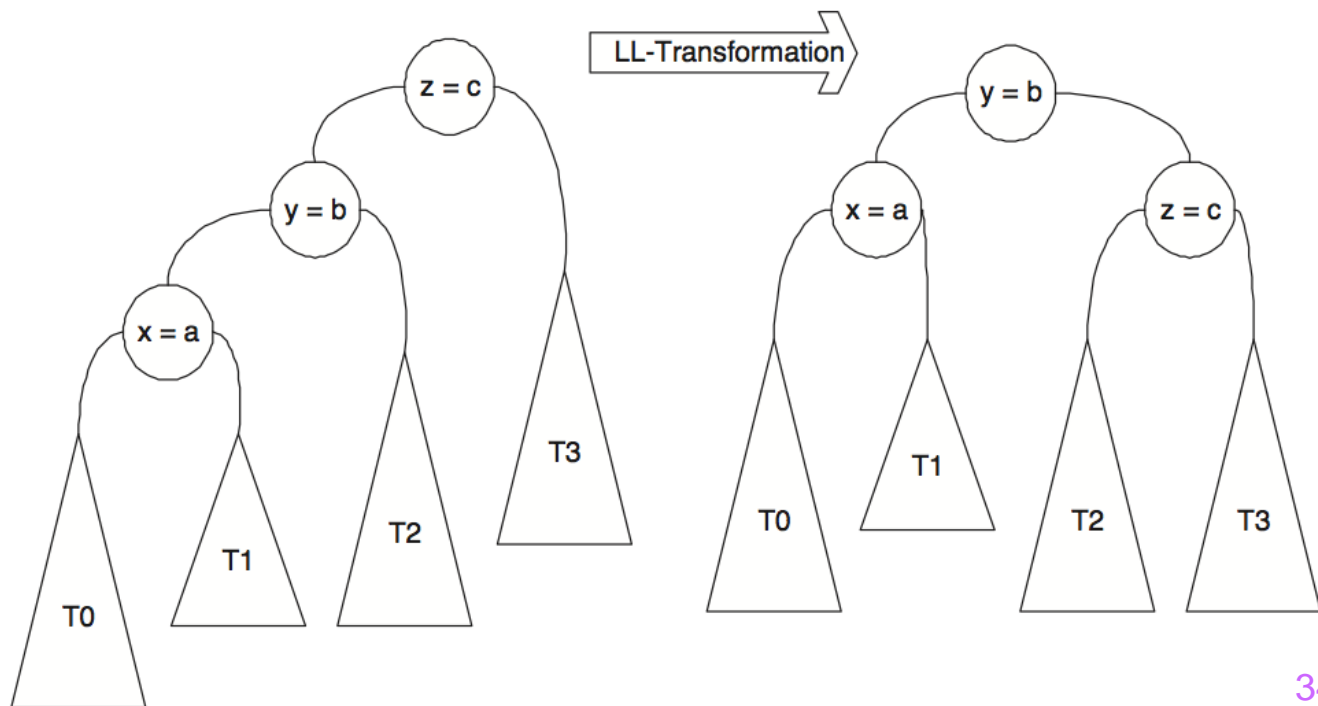
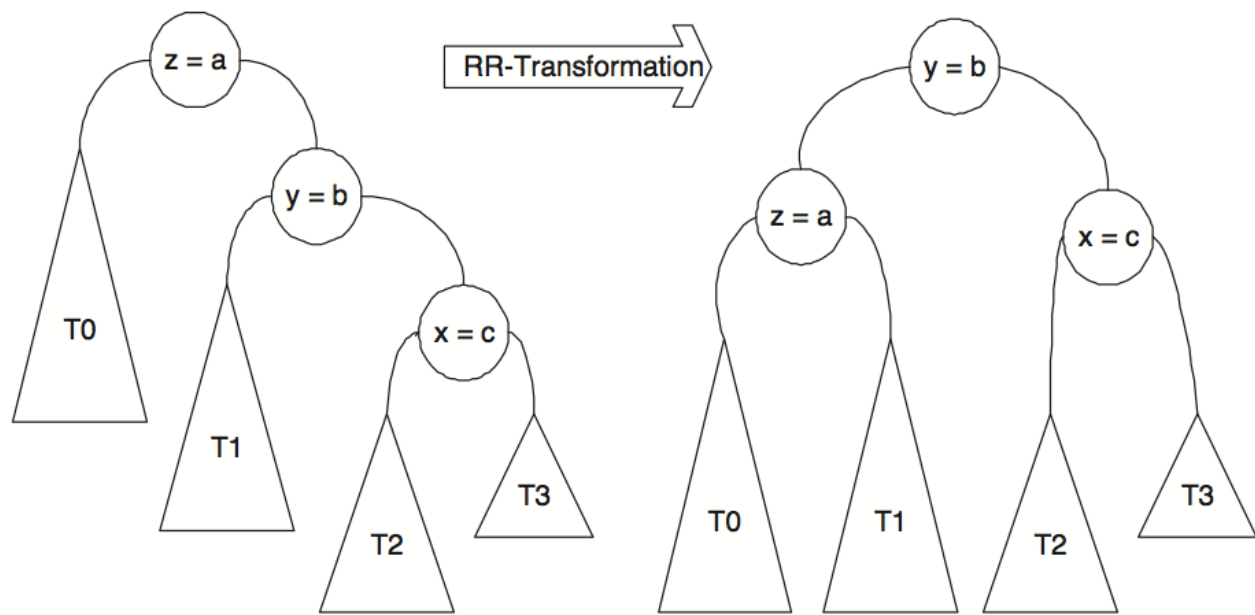
Die Höhe eines Baums mit der Wurzel V ist definiert als $h(v) = 1 + \max(h(T_l), h(T_r))$, wobei T_l und T_r das linke bzw. das rechte Kind (bzw. die Teilbäume) von v sind. Die Höhe eines leeren Teilbaums ist 0.

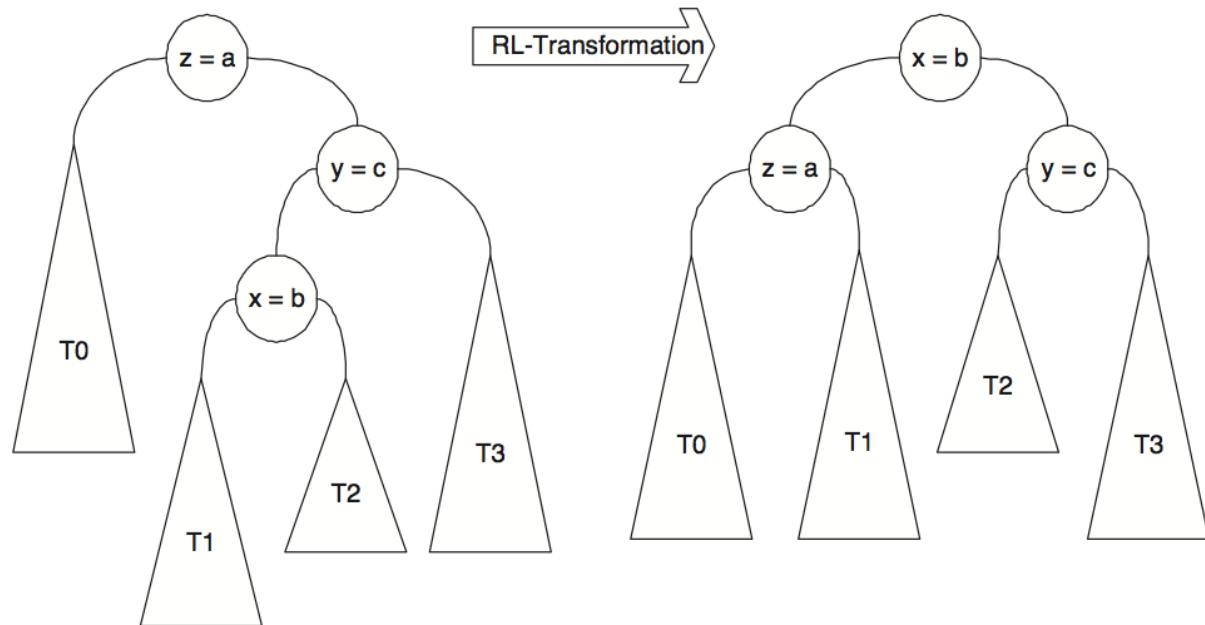
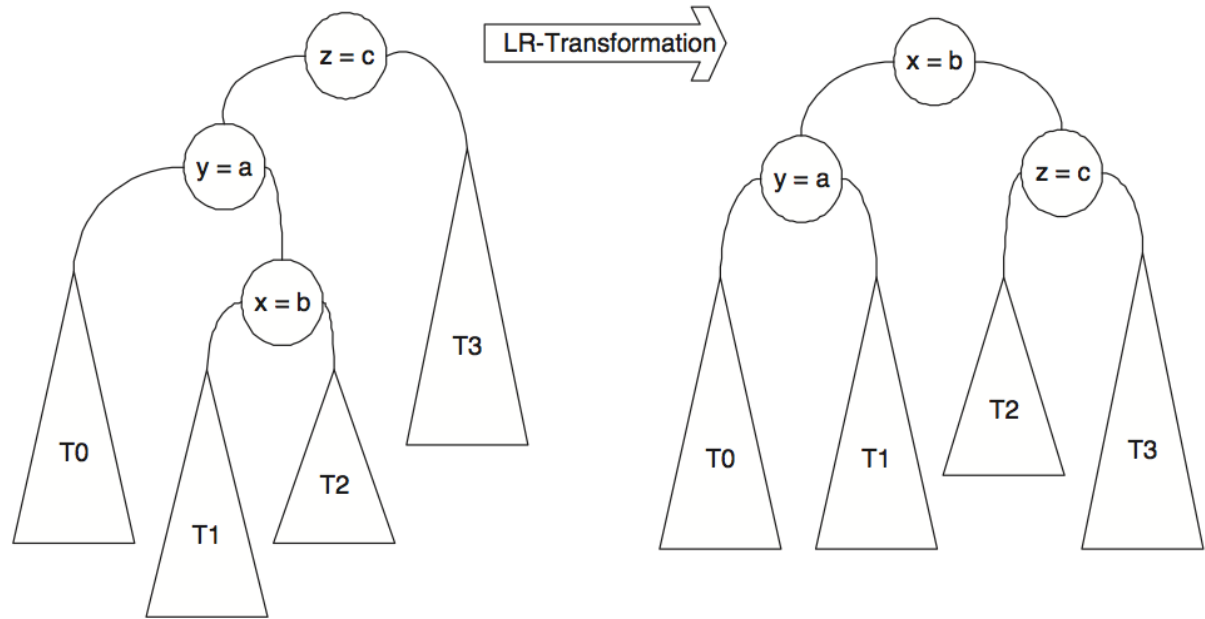
Ein binärer Suchbaum erfüllt die AVL-Eigenschaften, wenn für jeden Knoten v des Baums gilt:

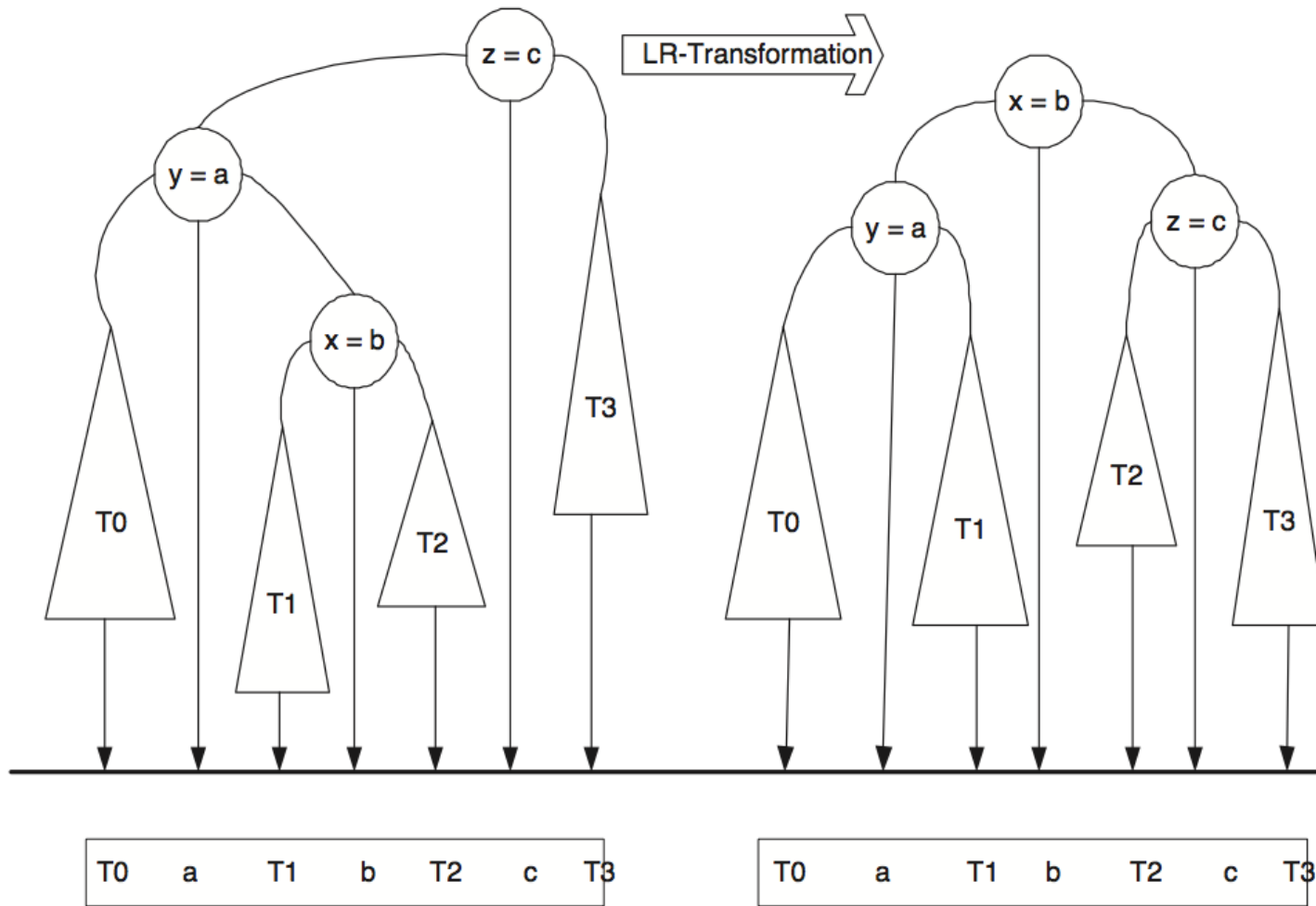
$$|h_l - h_r| \leq 1$$

Der AVL-Baum ist benannt nach Adel'son-Vel'skii und Landis.

Den Wert $h_l - h_r$ nennt man den Balance-Faktor des Knotens. Gültige Werte für diesen Balance-Faktor sind $-1, 0, 1$. Wenn beim Einfügen oder Löschen eine "Unbalanciertheit" auftritt, muß diese durch entsprechende Transformationen revidiert werden.





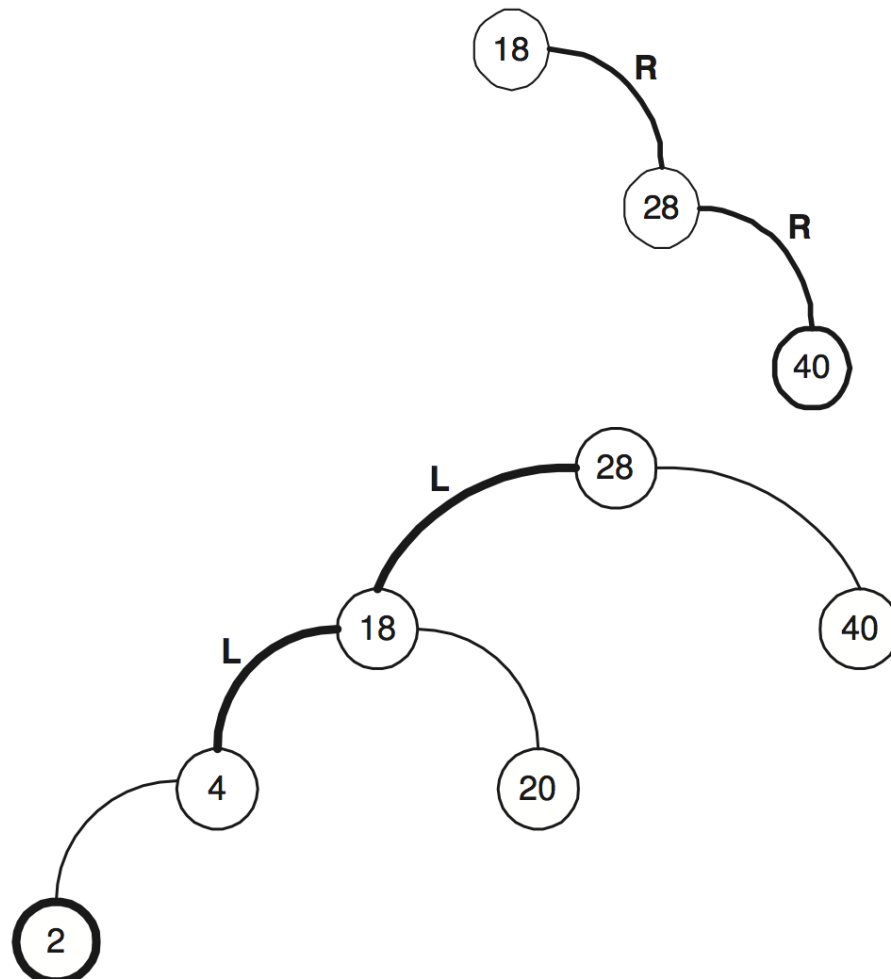


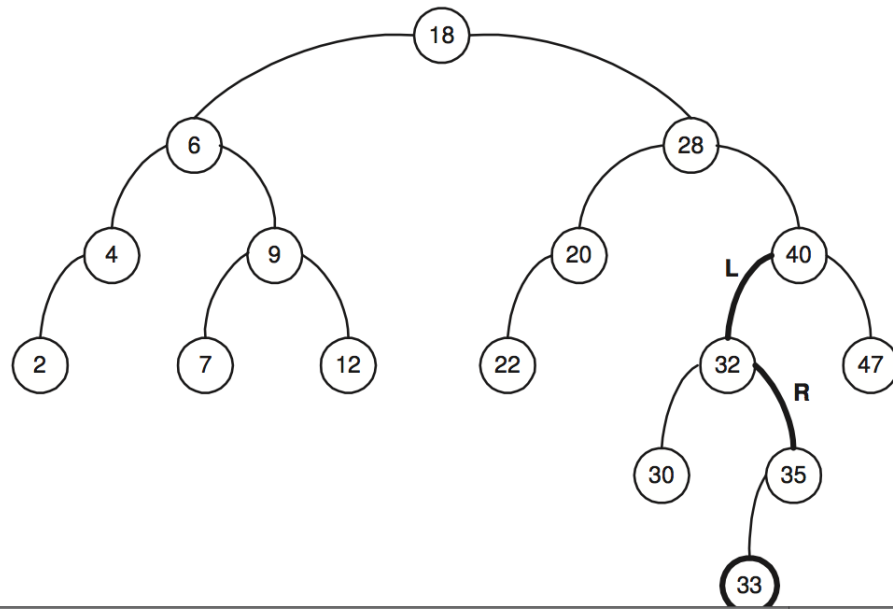
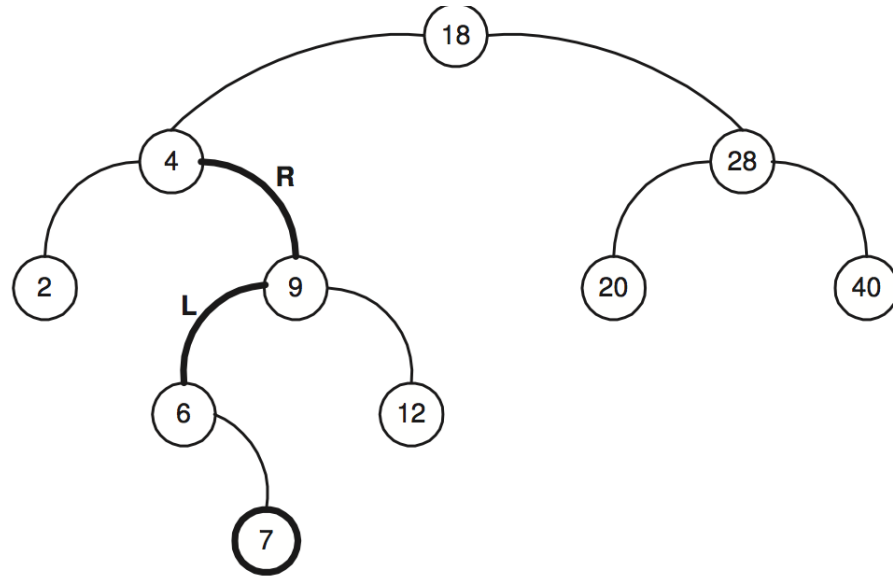
$(T_0 a (T_1 b T_2)) c T_3$

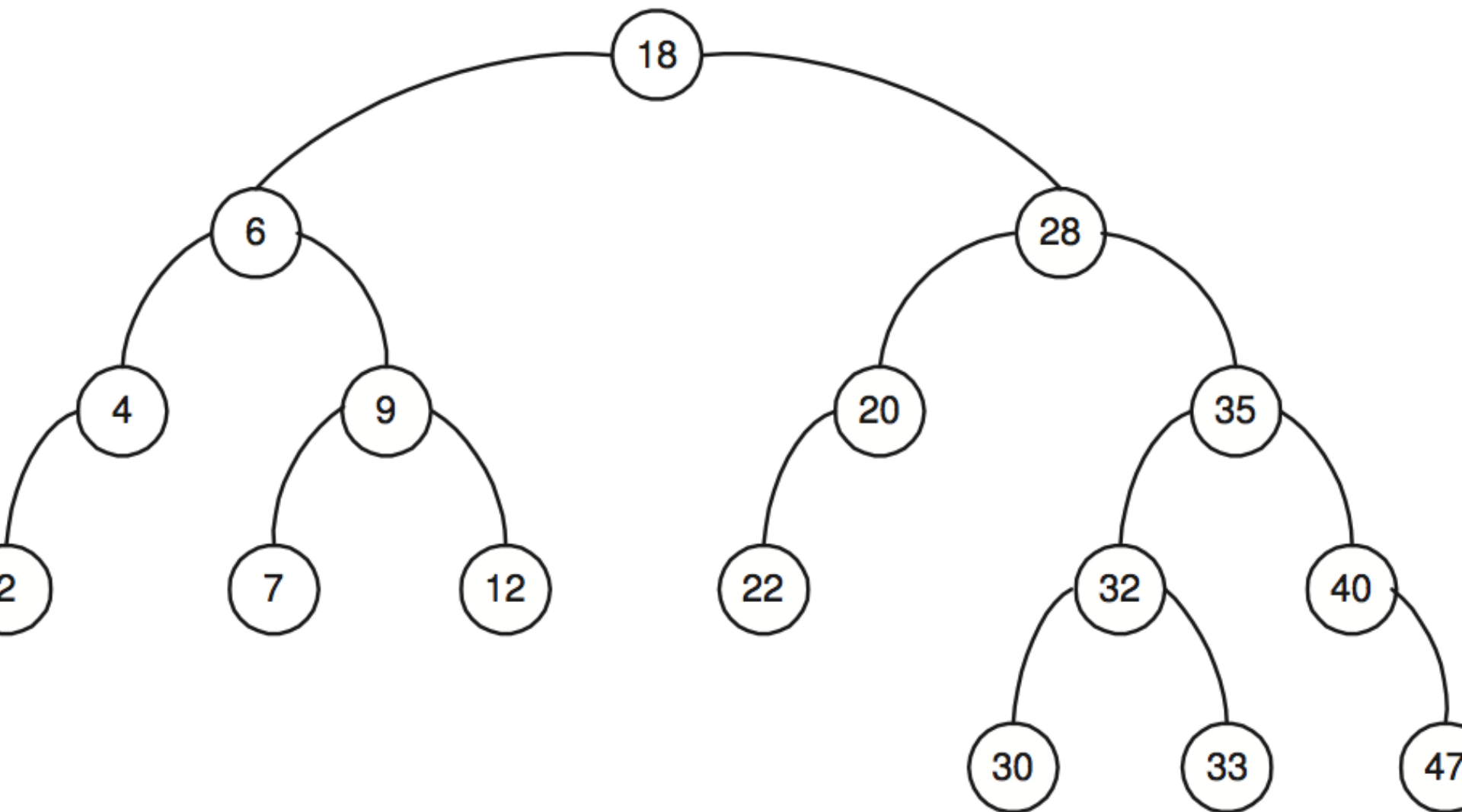
Assoziativ-Gesetz

$(T_0 a T_1) b (T_2 c T_3)$

Einfügen: 18, 28, 40, 20, 4, 2, 9, 6, 12, 7, 22, 32, 47, 30, 35, 33







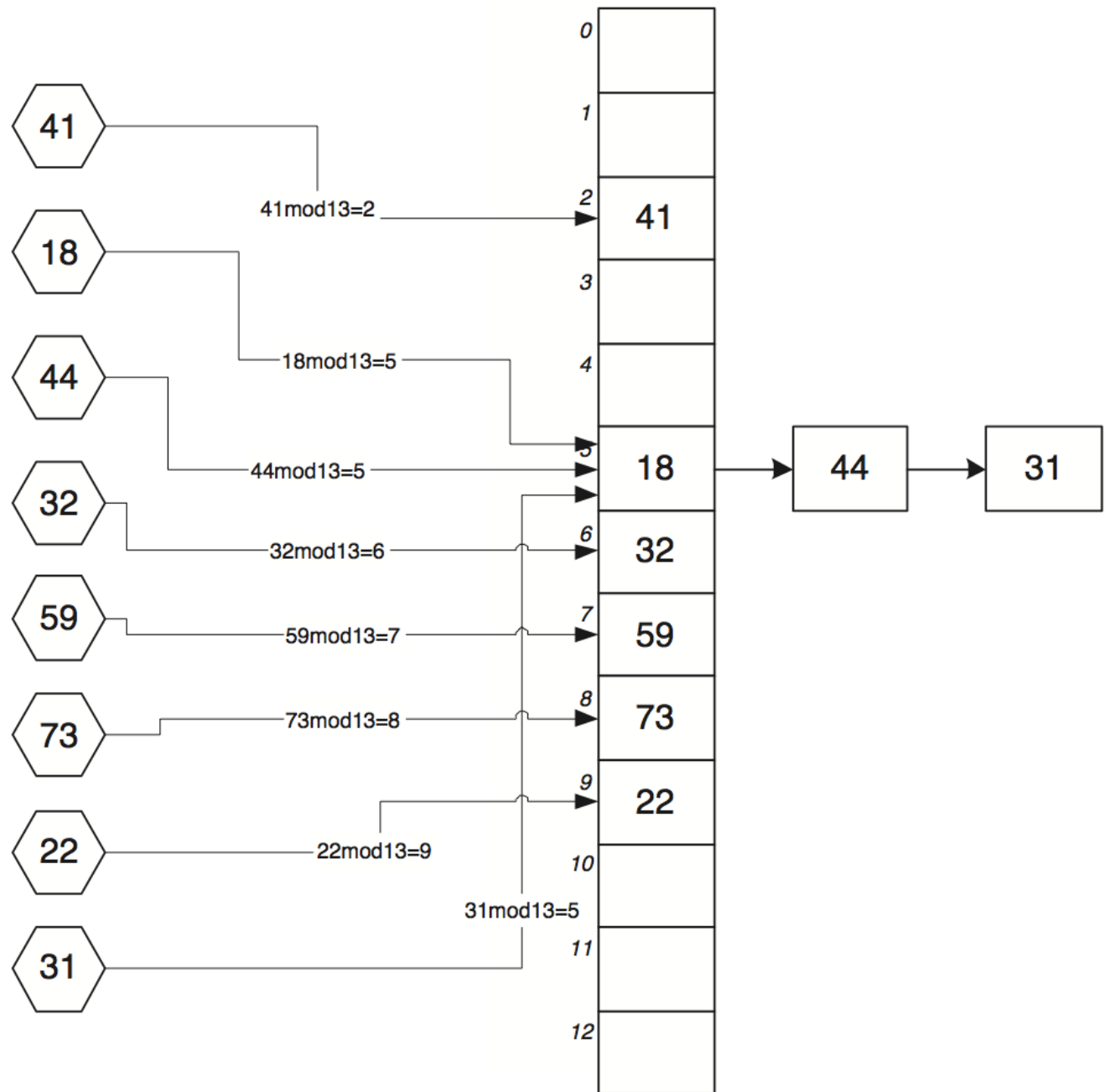


Abbildung 8.3: Hash-Tabelle mit Verkettung als Kollisionsbehandlung

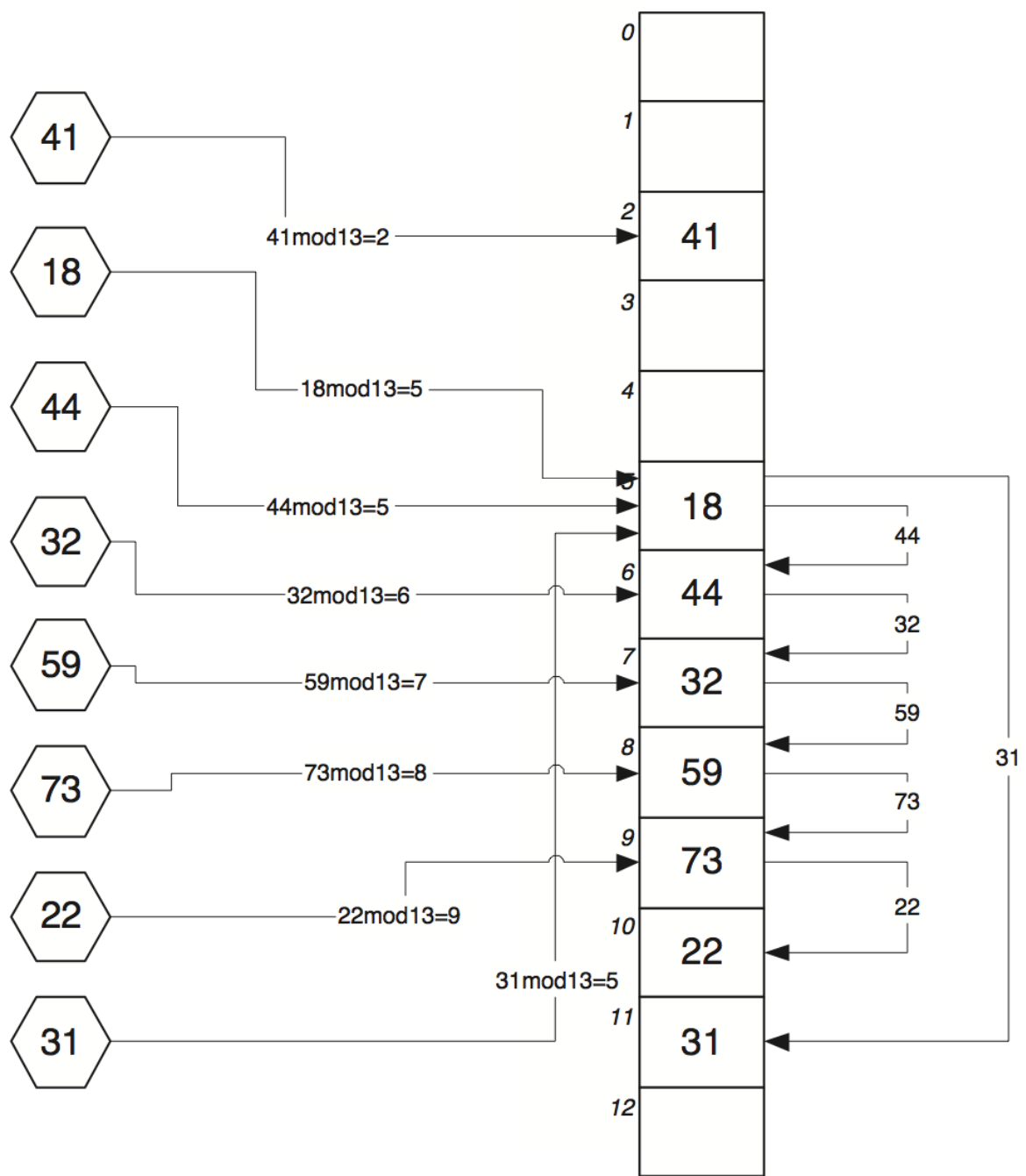


Abbildung 8.4: Hash-Tabelle mit *Linear Probing* als Kollisionsbehandlung

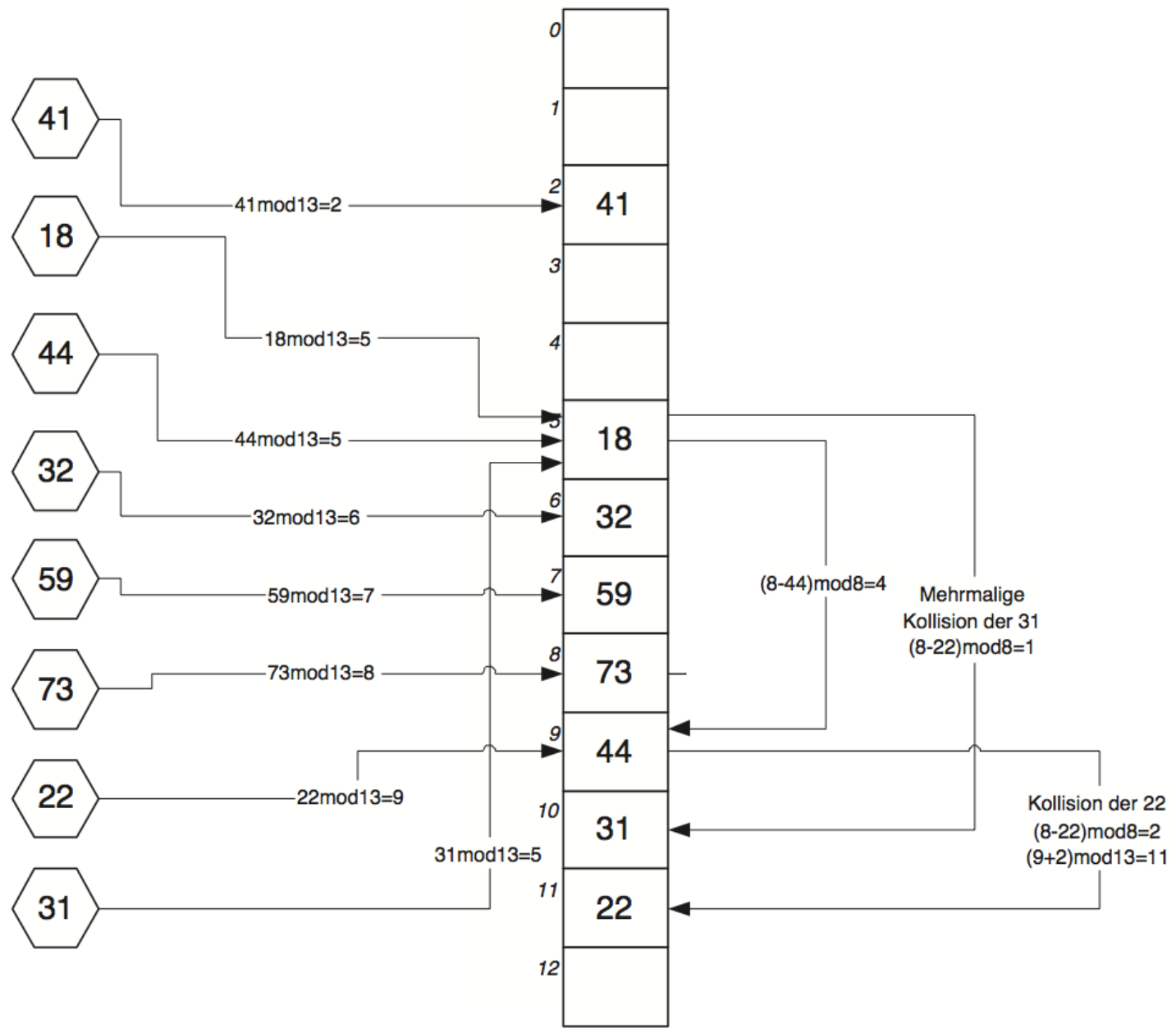


Abbildung 8.5: Hash-Tabelle mit *Double Hashing* als Kollisionsbehandlung: $h_1(K) = K \bmod 13$ und $h_2(K) = (8 - K) \bmod 8$