Technische Hochschule Rosenheim
Technical University of Applied Sciences

Objektorientierte Programmierung
Kapitel 2b – Bäume

Prof. Dr. Kai Höfig

Menge von Elementen

also ohne Duplikate

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- hatten wir die Liste als sequenzielle Datenstruktur behandelt. Diese speichert Daten in der Reihenfolge ab, in der sie hinzugefügt werden. Dabei können auch Duplikate eingefügt werden.

- Ein *Set,* also eine Menge, soll also jedes Element nur genau einmal enthalten, auch wenn man versucht es wieder einzufügen.

- Das Interface dafür könnte so aussehen:

```
public interface CharSet {
    void add(char c);          // Element hinzufügen
    boolean contains(char c);  // prüfen ob bereits enthalten
    char remove(char c);       // Element entfernen
    int size();                // nicht "length", da keine Sequenz
}
```

- Da ein Set keine Ordnung hat, also alle Elemente "einfach so" darin liegen ohne Index-Nummern, gibt es statt der `length` eine `size` Methode. Daher übergibt man der `remove` Methode auch keinen Index, sondern einen Wert!

Innere Klassen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wenn wir ein Set nun analog zur Liste implementieren, so brauchen wir wieder eine Hilfsklasse, welche die eigentlichen Daten speichert und die Datenstruktur aufspannt. Da die Klasse spezifisch für diese Struktur und Implementierung ist, kann sie als *innere* Klasse angelegt werden:

```
class CharSetImpl1 implements CharSet {

    private Element head;

    class Element {
        char value;
        Element next;
        Element(char c, Element n) {
            value = c;
            next = n;
        }
    }
..
```

- Innere Klassen sind in Java...
  - sinnvoll, wenn sie ausschließlich innerhalb einer Klasse, also lokal verwendet werden.
  - normal oder `static` definiert; normale innere Klassen können auf die Variablen der äußeren Instanz zugreifen, statische innere Klassen können nur auf statische Variablen und Methoden der äußeren Klasse zugreifen.
  - mit Sichtbarkeiten versehen - so wie Variablen und Methoden: `package` (kein Schlüsselwort), `private`, `public`.

Duplikate Vermeiden

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Wir können nun Duplikate vermeiden, indem wir vor dem Einfügen prüfen, ob ein Element schon enthalten ist. Dazu beginnen wir vorne und hangeln uns bis hinten durch, wobei wir jedes Element auf (Wert-)Gleichheit prüfen.

```
public boolean contains(char c) {
    if (head == null)
        return false;

    Element it = head;
    while (it != null) {
        if (it.value == c)
            return true;
        it = it.next;
    }

    return false;
}
```

- Das Einfügen (`add`) kann dann analog zur Liste realisiert werden, ebenso die Methoden `size`, `remove` und `toString`
- die Implementierung des Sets als Liste hochgrading ineffizient ist: bei jedem add bzw. contains muss zuerst die gesamte Liste durchlaufen werden, um zu prüfen ob das Element nicht bereits enthalten ist. Daher verwenden wir jetzt *Bäume*

Ziele

Technische Hochschule Rosenheim
Technical University of Applied Sciences

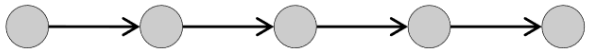
- Den Begriff des *Baums* in der Informatik kennenlernen
- Bäume als verkettete Datenstruktur repräsentieren können
- Rekursive Funktionen auf Bäumen verstehen und schreiben können

Bäume

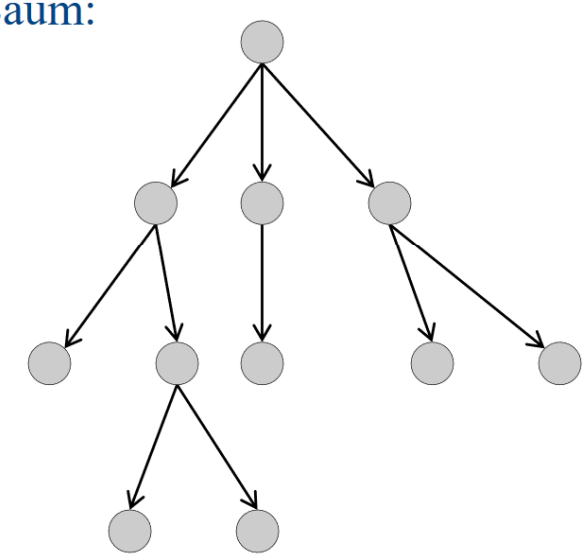
Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Bäume sind eine der am meisten verwendeten Datenstrukturen in der Informatik
- Bäume verallgemeinern Listen:
  - In einer Liste hat jeder Knoten höchstens einen Nachfolger
  - In einem Baum kann ein Knoten mehrere Nachfolger haben.

Liste:



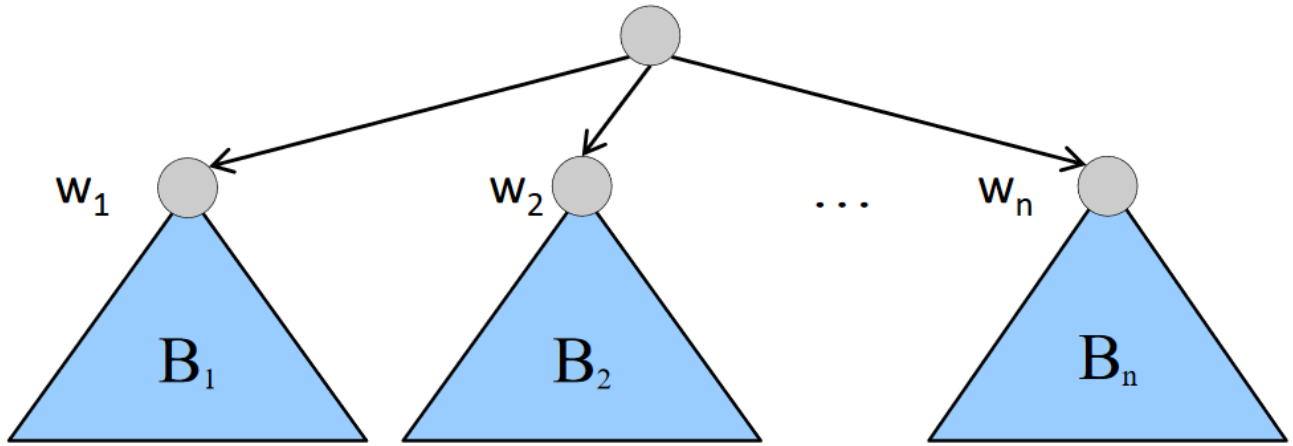
Baum:



Definitionen

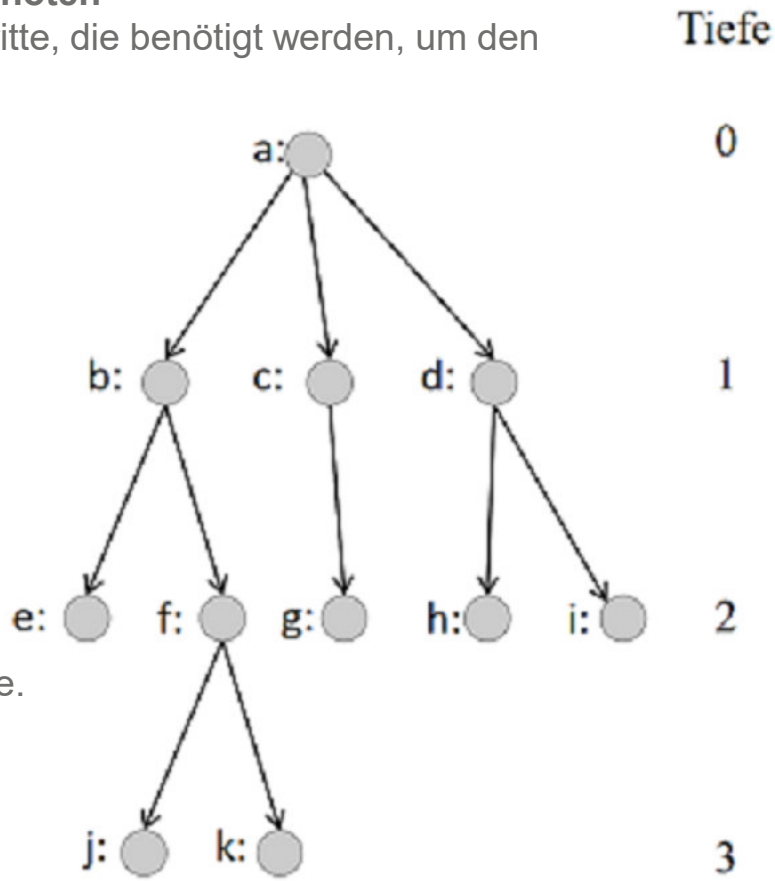
Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Ein **Baum** besteht aus **Knoten**, die durch **Kanten** miteinander verbunden sind.
- Ein **Wurzelknoten** (*Root*) ist ein Knoten, auf den keine Kante zeigt.
- In einem Knoten können je nach Anwendung verschiedene Daten gespeichert sein.
- Die Menge aller Bäume wird durch folgende Regeln konstruiert:
  - Es gibt einen leeren Baum.
  - Ein einzelner Knoten ohne irgendwelche Kanten ist ein Baum (Root).
  - Ist n > 0 und sind *B1,B2, ...Bn* Bäume mit Wurzelknoten *w1,w2, ... wn*, so kann man diese zu einem größeren Baum zusammensetzen, indem man einen neuen Knoten hinzufügt und diesen mit w1,w2, ..., wn verbindet.
  - Der neue Knoten ist dann Wurzelknoten des so aufgebauten Baums.



Terminologie

- a ist der **Wurzelknoten** des Baums.
- h und i sind die **Nachfolger-** oder auch **Kindknoten** des Knotens d.
- d ist **Vorgänger-** oder **Elternknoten** von h.
- Knoten ohne Nachfolger (hier: e, j, k, g, h, i) heißen **Blattknoten**
- Die **Tiefe eines Knotens** im Baum ist die Anzahl der Schritte, die benötigt werden, um den Knoten von der Wurzel zu erreichen.
- Die **Tiefe des Baums** ist das Maximum der Tiefen aller Knoten des Baums. (hier: 3)
- Ein Baum ist ein **Binärbaum**, wenn jeder Knoten darin **höchstens** zwei Nachfolger hat.
- Jeder Knoten in einem Baum ist Wurzel eines **Teilbaums** des gegebenen Baums.
- Der Teilbaum mit Wurzel k besteht aus dem Knoten k, seinen Kindern, Kindeskindern, usw.
- Beispiele: Der Teilbaum mit Wurzel b und der mit Wurzel d. Im Vergleich dazu der Teilbaum mit Wurzel a ist der ganze Baum selbst.
- Die Teilbäume mit Wurzeln b und d sind beide Binärbäume.
- Der gesamte Baum ist kein Binärbaum.



Binärbaum

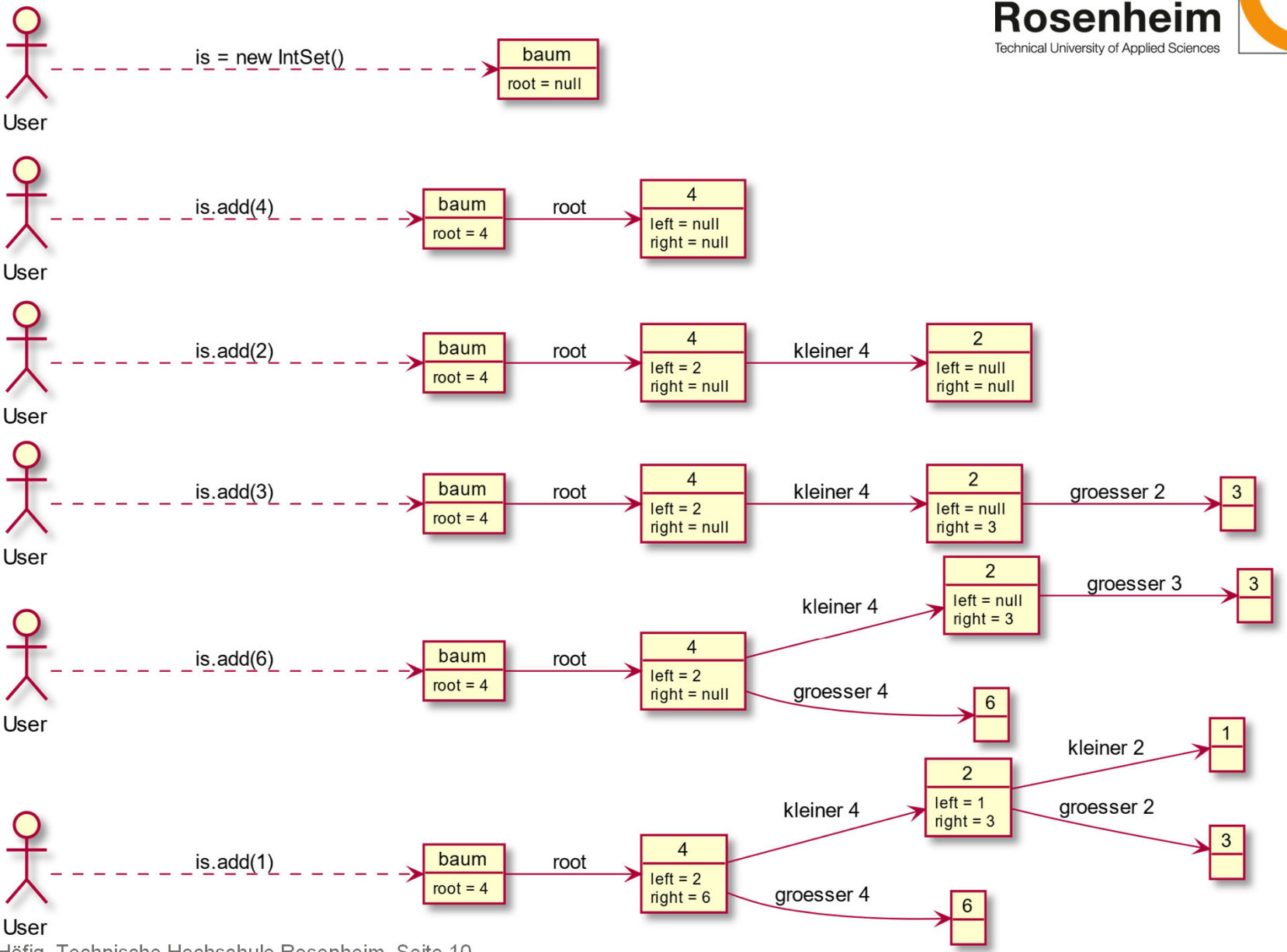
Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Eine fundamentale Datenstruktur in der Informatik ist der *Binärbaum*. Er unterscheidet sich von der Liste dahingehend, dass jedes Element nicht einen, sondern *zwei* Nachfolger hat -- daher der Name: *binär*.

- Ein Element zeigt also nicht *sequenziell* auf das *nächste* Element, sondern unterhält Referenzen auf einen *linken* und *rechten Teilbaum*, welche dann jeweils nur *kleinere* bzw. *größere* Werte als das Element selbst enthalten.

- Fügt man nun ein Element ein, so steigt man vom Wurzelknoten (engl. *root*) so weit nach links oder rechts ab, bis man entweder den Wert gefunden hat, oder an einer Stelle angekommen ist, wo man den neuen Wert einzufügen kann.

Beispiel


User
is = new IntSet()
baum
root = null
User
is.add(4)
baum
root = 4
root
4
left = null
right = null
User
is.add(2)
baum
root = 4
root
4
left = 2
right = null
kleiner 4
2
left = null
right = null
User
is.add(3)
baum
root = 4
root
4
left = 2
right = null
kleiner 4
2
left = null
right = 3
groesser 2
3
User
is.add(6)
baum
root = 4
root
4
left = 2
right = null
kleiner 4
2
left = null
right = 3
groesser 3
3
groesser 4
6
User
is.add(1)
baum
root = 4
root
4
left = 2
right = 6
kleiner 4
2
left = 1
right = 3
kleiner 2
1
groesser 2
3
groesser 4
6

Implementierung Binärbaum

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Möchte man nun ein Element suchen (contains), so beginnt man am Wurzelknoten (root), prüft ob der Wert bereits dort vorhanden ist, und steigt ansonsten nach links bzw. rechts ab, je nachdem ob der gesuchte Wert kleiner oder größer als das angesehene Element ist.

- Wie wird so ein Baum nun implementiert, und wie steigt man nach links oder rechts ab? Beginnen wir mit der Struktur; ein Element hat nun also nicht einen Nachfolger next, sondern zwei: left und right.

```
public class BinaryTreeCharSetImpl implements CharSet{
    class Element {
        char value;
        Element left, right;
        Element(char c, Element le, Element re) {
            value = c;
            left = le;
            right = re;
        }
    }
    Element root;
..
```

Die Methode `contains`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Bei der contains Methode wird zwar ähnlich zur Liste iteriert, aber statt it = it.next muss unterschieden werden, ob man links oder rechts absteigen möchte:

```
public boolean contains(char t) {
    if (root == null)
        return false;

    Element it = root;
    while (it != null) {
        if (t == it.value)
            return true;
        else if (t < it.value) {
            it = it.left;
        } else {
            it = it.right;
        }
    }

    // nicht gefunden!
    return false;
}
```

Die Methode add

- Beim Einfügen wird nun ähnlich vorgegangen, nur dass man hier nach links oder nach rechts schauen muss, statt einfach immer ein Element voraus:

```
public void add(char c) {
    Element e = new Element(c, null, null);

    if (root == null) {
        root = e;
        return;
    }

    Element it = root;  // beginne an der Wurzel
    while (it != null) {
        // schon vorhanden -> fertig!
        if (it.value == e.value)
            return;
            // links absteigen?
        else if (e.value < it.value) {
            // wir sind unten angekommen -> einfügen!
            if (it.left == null) {
                it.left = e;
                return;
            } else
                it = it.left;
        } else {
            // analog
            if (it.right == null) {
                it.right = e;
                return;
            } else
                it = it.right;
        }
    }
}
```

Die Methode `size`

mit Rekursion

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
class Element {
    char value;
    Element left, right;
    int size() {
        int s = 1;  // schon mal mindestens ein Element.

        // gibts einen linken Teilbaum? Dann dessen Größe dazuaddieren.
        if (left != null) s += left.size();

        // analog.
        if (right != null) s += right.size();

        return s;
    }

..
```

```
@Override
public int size() {
    if (root == null) return 0;
    else return root.size();
}
```

Die Methode `toString`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Auch diese Methode ist wieder in einen Teil für `Element`

```
public String toString() {
    String s = "" + value;  // schon mal dieses Element

    // gibts einen linken Teilbaum? Dann dessen String dazu
    if (left != null) s += ", " + left.toString();

    // analog.
    if (right != null) s += ", " + right.toString();

    return s;
}
```

- ..und einen Teil für unsere Tree Set Implementierung aufgeteilt:

```
public String toString() {
    if (root == null) return "[]";
    else return "[" + root.toString() + "]";
}
```

Löschen von Elementen

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Bei der **Liste** war das Löschen eines Elementes einfach: Zunächst schaut man immer ein Element voraus, um das zu löschende Element zu finden; man bleibt also ein Element davor stehen. Das eigentliche Löschen wird dadurch erreicht, in dem man nun die Verlinkung so ändert, dass das Element vor dem zu löschenden auf das Element hinter dem zu löschenden zeigt.

- Möchte man nun ein Element aus dem Set bzw. **Baum** löschen, ist das etwas komplizierter, da man beim Suchen ja rechts und links betrachten muss, analog zu `contains`.

- Wir brechen dieses schwierige Problem in einfachere Teilprobleme auf. Zunächst gibt es drei Fälle zu betrachten:

    - Der Baum ist bereits leer; hier sollte eine NoSuchElementException geworfen werden.
    - Das zu löschende Element ist das Wurzelelement root
    - Das zu löschende Element ist ein innerer Knoten oder Blatt.

Löschen von Elementen

Der Baum ist leer

Technische Hochschule Rosenheim
Technical University of Applied Sciences

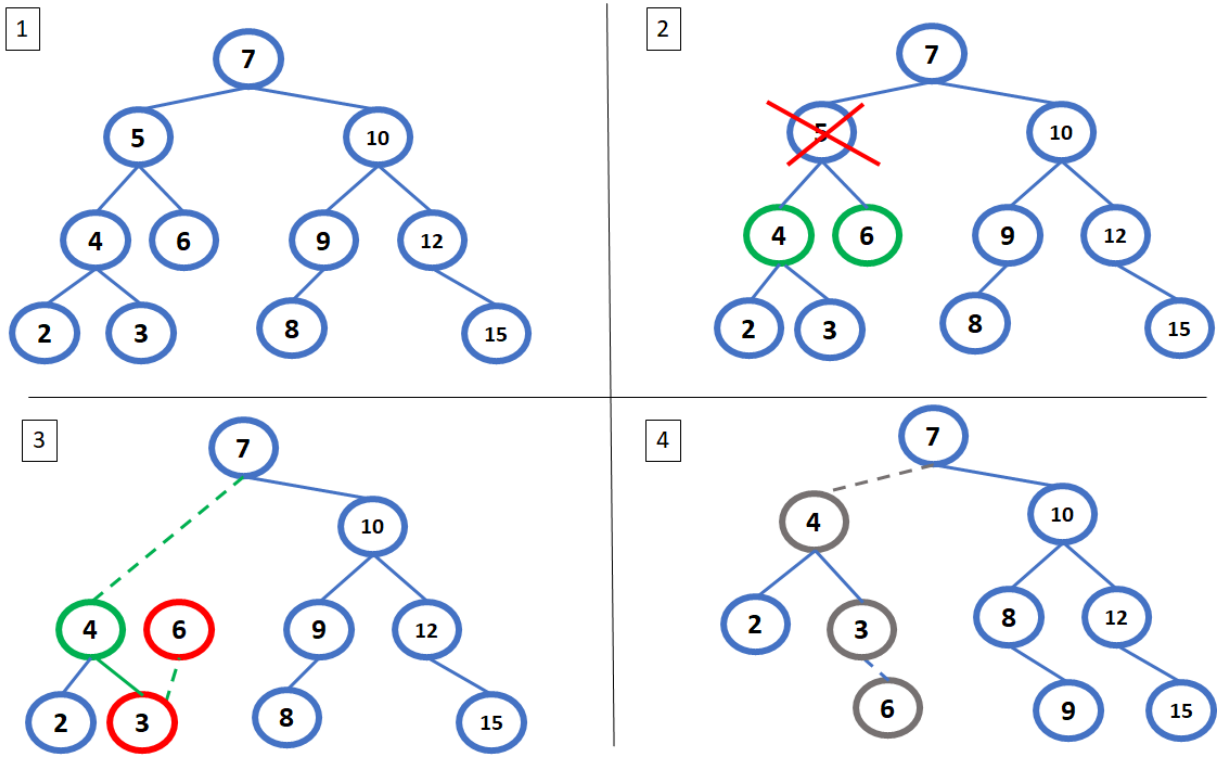
- Der Baum ist bereits leer; hier sollte eine NoSuchElementException geworfen werden.

```
public char remove(char c) {
    // Trivialfall prüfen
    if (root == null)
        throw new NoSuchElementException();
```

Vorgehen beim Löschen

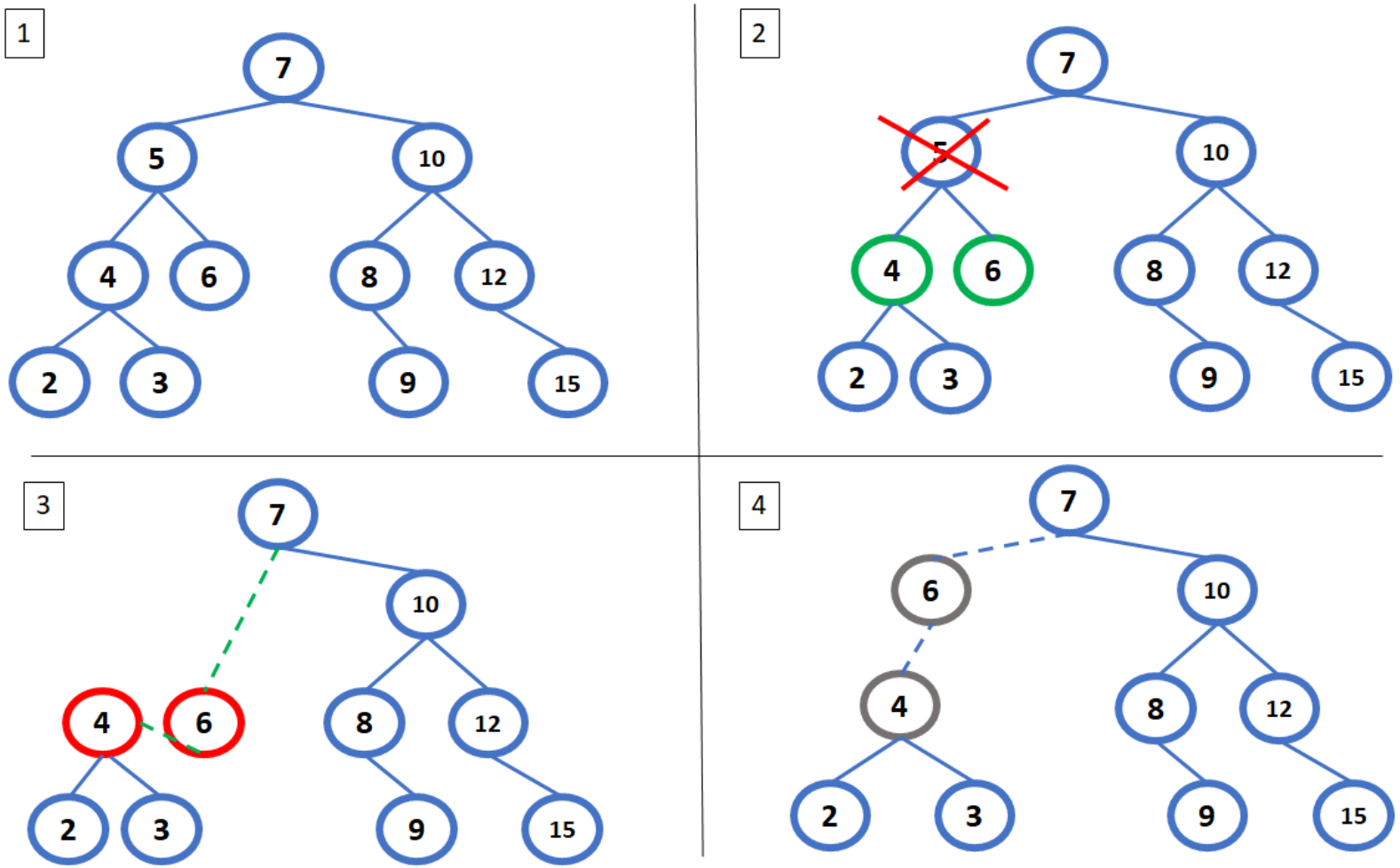
Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Löschen wir ein Element, so müssen wir sicherstellen, dass etwaige Teilbäume (left und right) wieder in den Baum eingefügt werden. Hierzu erstellt man eine Hilfsmethode `addElement`, welche analog zur bestehenden `add` Methode arbeitet, aber eben gleich statt einem Wert ein Element (mit etwaigen Teilbäumen) einfügt.



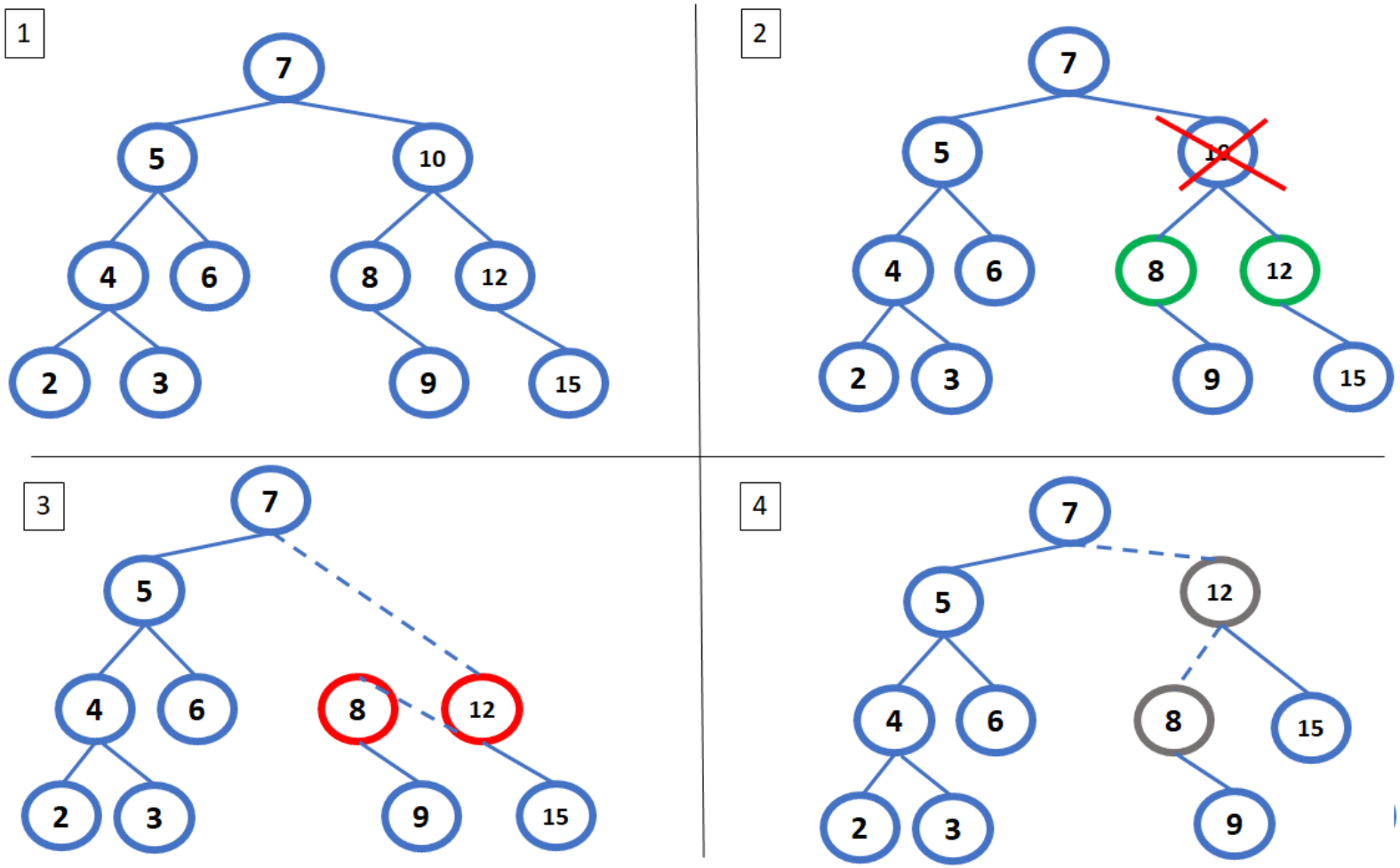
Option 2

Technische Hochschule Rosenheim
Technical University of Applied Sciences



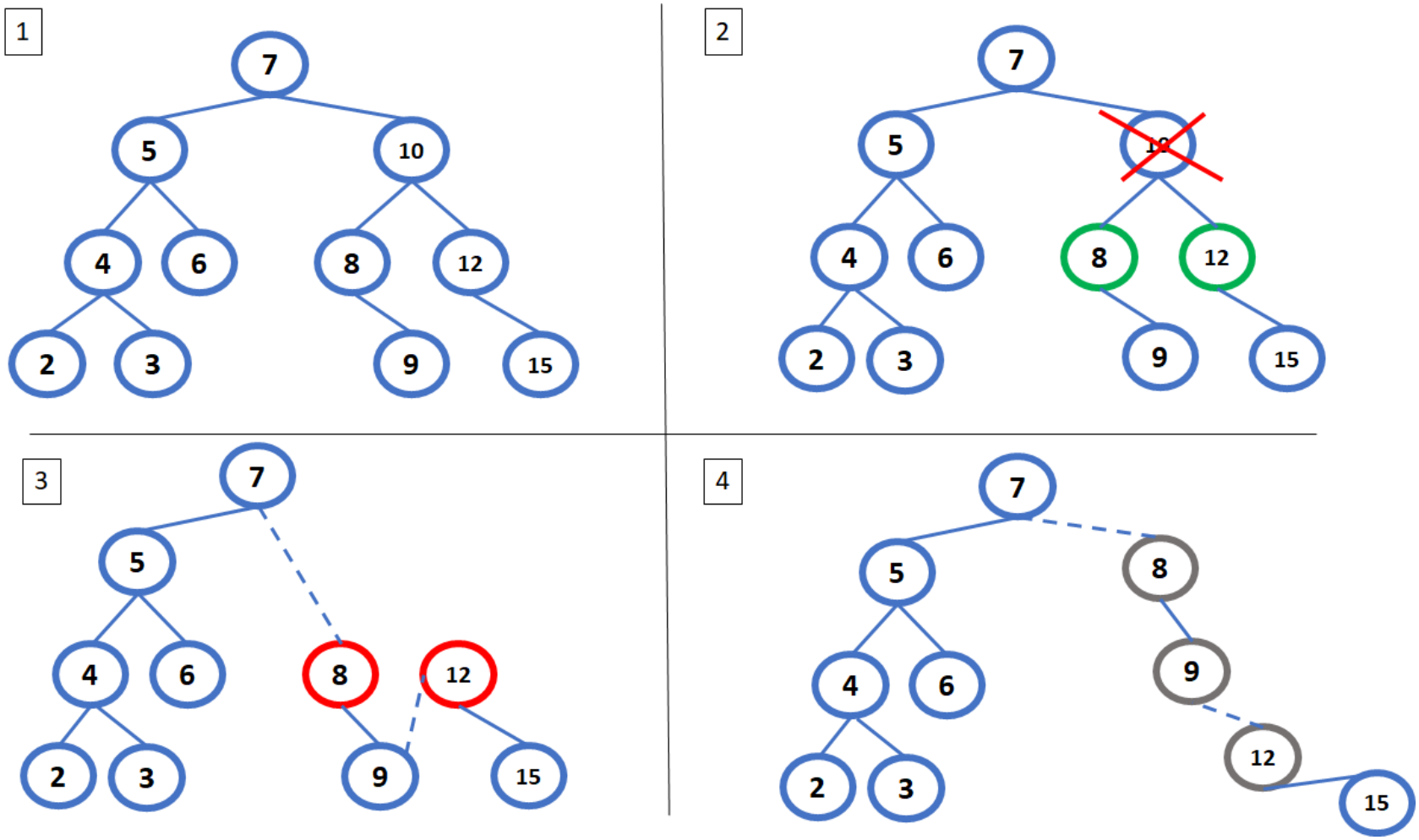
Option 3

Technische Hochschule Rosenheim
Technical University of Applied Sciences



Option 4

Technische Hochschule Rosenheim
Technical University of Applied Sciences



Hilfsmethode `addElement`

Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
private void addElement(Element e) {
    if (e == null)
        return;
    if (root == null) {
        root = e;
        return;
    }
    Element it = root;
    while (it != null) {
        if (it.value == e.value)
            return;
        else if (e.value < it.value) {
            if (it.left == null) {
                it.left = e;
                return;
            } else
                it = it.left;
        } else {
            if (it.right == null) {
                it.right = e;
                return;
            } else
                it = it.right;
        }
    }
}
```

Löschen von Elementen

Löschen der Wurzel

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Das zu löschende Element ist das Wurzelelement root

```
private char removeRoot() {
    Element e = root;
    if (e.left == null && root.right == null) {
        // keine Teilbäume, also ist der Baum nun leer
        root = null;
    } else if (e.left == null) {
        // nur rechter Teilbaum, also diesen als Wurzel setzen
        root = e.right;
    } else if (e.right == null) {
        // dito, fuer links
        root = e.left;
    } else {
        // es gibt beide Teilbäume; links wird neue Wurzel,
        // rechts wird als Element eingefügt.
        root = e.left;
        addElement(e.right);
    }
```

Löschen von Elementen

Löschen innerhalb oder Blatt

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Das zu löschende Element ist ein innerer Knoten oder Blatt.

```
// Am Wurzelknoten beginnen
Element it = root;
while (it != null) {
    if (c < it.value) {
        // ist der gesuchte Wert kleiner, so müssen
        // wir nach links schauen, und ggf. das linke
        // Nachfolgeelement löschen; delegieren
        if (it.left != null && it.left.value == c)
            return removeElement(it, it.left);
        it = it.left;
    } else {
        // rechts analog.
        if (it.right != null && it.right.value == c)
            return removeElement(it, it.right);
        it = it.right;
    }
}
```

Hilfsmethode `removeElement`

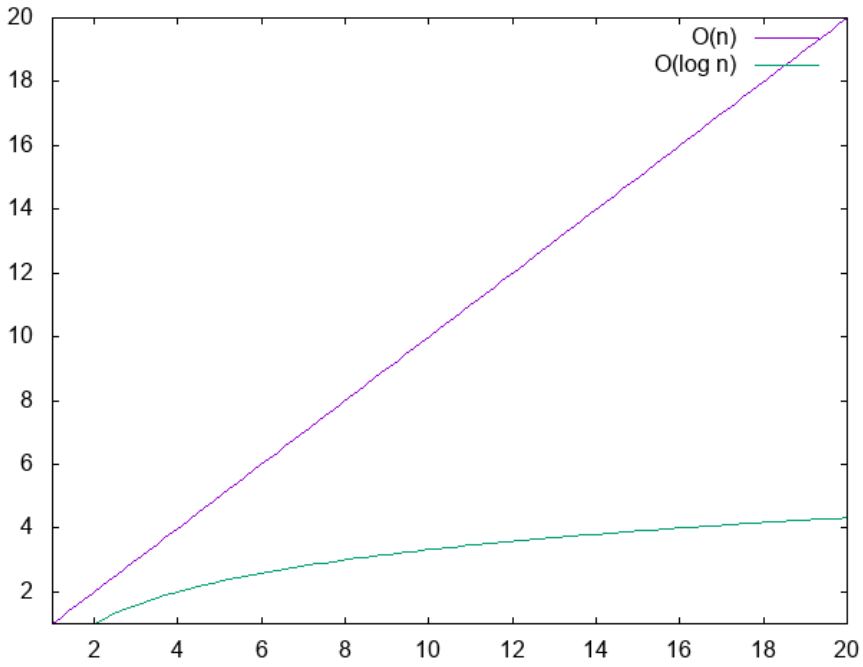
Technische Hochschule Rosenheim
Technical University of Applied Sciences

```
private char removeElement(Element parent, Element element) {
    // wollen wir das linke Element löschen?
    if (element == parent.left) {
        parent.left = null;
    } else {
        parent.right = null;
    }

    // eventuelle Teilbäume neu in den Baum einfügen
    addElement(element.left);
    addElement(element.right);

    return element.value;
}
```

Komplexität

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Warum nun das ganze mit dem Baum, wenn doch eine Liste so viel einfacher wäre?
- Man kann mathematisch-kombinatorisch beweisen, dass der mittlere Aufwand (Komplexität) zum Suchen bzw. Einfügen in einen Binärbaum O(log n) ist; ist der Baum *balanciert* (also die Verzweigung ausgewogen), so ist der Aufwand sogar im *worst case* O(log n).
- Wir erinnern uns an den Eingang dieses Kapitels: Die naive Implementierung eines Sets als Liste hatte die Komplexität O(n). Das scheint beim Lesen nicht arg unterschiedlich, doch auch hier sagt ein Bild mehr als 1000 Worte:
- Man sieht: bei linear steigenden n (x-Achse) bleibt die logarithmische Steigung sehr schnell sehr weit unter der Linearen. Die Baumstruktur ist also wesentlich effizienter!



Zusammenfassung

Technische Hochschule Rosenheim
Technical University of Applied Sciences

- Ein Set ist im Gegensatz zu einer Liste frei von Duplikaten; in der Regel werden `add`, `remove`, `contains` und `size` unterstützt.
- Da ein Set eine Menge ohne Ordnung (Reihenfolge) ist, gibt es keinen Zugriff über einen Index.
- Ein Binärbaum ist eine Datenstruktur, bei der Elemente zwei Nachfolger haben; in diesen Teilbäumen sind dann die kleineren (links) und größeren Elemente (rechts) gespeichert.
- Ein Set kann zwar mit einer Liste implementiert werden, ein Binärbaum ist aber deutlich effizienter.
- Bei komplizierter Struktur kann Rekursion oft eine elegante Lösung sein; hierbei ruft eine Methode sich selbst wieder auf.