

Suchbäume

Dr. Philipp Wendler

Zentralübung zur Vorlesung

„Einführung in die Informatik: Programmierung und Softwareentwicklung“

<https://www.sosy-lab.org/Teaching/2018-WS-InfoEinf/>

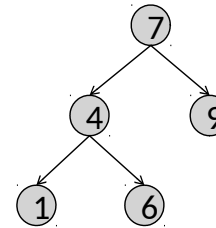
WS18/19

Bäume

Ein Baum besteht aus Knoten, die durch Kanten miteinander verbunden sind.

Spezialfall:

Ein Binärbaum ist ein Baum, in dem jeder Knoten max. zwei Kindknoten hat.



Im Vergleich zu verketteten Listen:

- verkettete Liste: max. **ein** Nachfolger
- Baum: **mehrere** Nachfolger

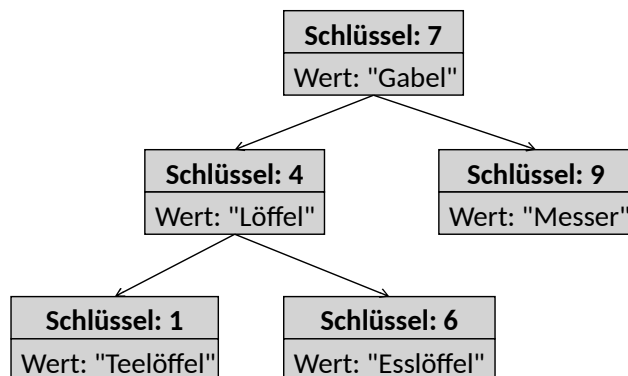
Philipp Wendler:

Bäume

2

Datenspeicherung in Bäumen

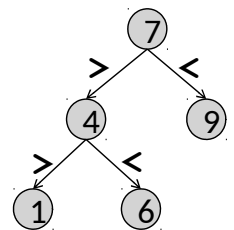
In den Knoten eines Baumes können je nach Anwendung verschiedene Daten gespeichert werden.



Binäre Suchbäume

Ein Binärbaum b heißt **geordnet**, wenn

- b der leere Baum ist, oder
- es gilt für alle Teilbäume t von b (auch b selbst):
Der Schlüssel der Wurzel von t ist
 - **größer** als alle Schlüssel im **linken** Teilbaum
 - **kleiner** als alle Schlüssel im **rechten** Teilbaum

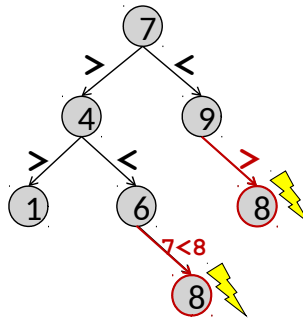


Ein geordneter Binärbaum heißt binärer Suchbaum.

Binäre Suchbäume

Ein Binärbaum b heißt **geordnet**, wenn

- b der leere Baum ist, oder
- es gilt für alle Teilbäume t von b (auch b selbst):
Der Schlüssel der Wurzel von t ist
 - **größer** als alle Schlüssel im **linken** Teilbaum
 - **kleiner** als alle Schlüssel im **rechten** Teilbaum



Wo darf die Zahl 8 im Baum eingehängt werden?

- a) an beiden Positionen
- b) an der linken, aber nicht der rechten Position
- c) an der rechten, aber nicht der linken Position
- d) an beiden Positionen nicht

Aufgabe: zu speichernde Daten

Das Institut für Informatik möchte für alle Büroräume abspeichern, welcher Mitarbeiter dort arbeitet.

- Die Büroräume sind durchnummeriert.
- In jedem Büroraum sitzt nur ein Mitarbeiter.

D.h. wir brauchen eine Datenstruktur, die die **Zimmernummern** zusammen mit dem jeweiligen **Mitarbeiter** abspeichert.

Zimmernummer: 006
Mitarbeiter: "Klarl"

Aufgabe

Das Institut für Informatik möchte für alle Büroräume abspeichern, welcher Mitarbeiter dort arbeitet.

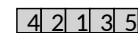
- Die Büroräume sind durchnummeriert.
- In jedem Büroraum sitzt nur ein Mitarbeiter.

Implementieren Sie eine Datenstruktur, die das Einfügen eines neuen Eintrags als auch die Suche nach einem Eintrag möglichst schnell verarbeiten kann.

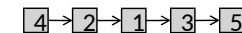
Aufgabe: Datenstrukturen (I)

Einfüge-Operation in verschiedenen Datenstrukturen: **6**

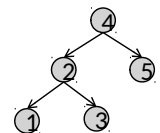
Array



verkettete Liste



binärer Suchbaum



In welcher Datenstruktur ist die Einfügeoperation am schnellsten?

- a) Array
- b) verkettete Liste
- c) binärer Suchbaum

Aufgabe: Datenstrukturen (II)

Einfüge-Operation in verschiedenen Datenstrukturen: **6**

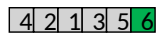
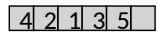
Array

4 2 1 3 5

Erzeugen eines neuen Arrays: $O(n)$



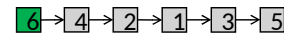
Kopieren der alten Werte: $O(n)$



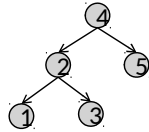
verkettete Liste

4 → 2 → 1 → 3 → 5

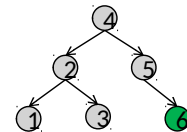
Vorne-Anhängen des neuen Werts: $O(1)$



binärer Suchbaum



Richtig-Einsortieren des neuen Werts: $O(\log n)$



Aufgabe: Datenstrukturen (III)

Element-Zugriff in verschiedenen Datenstrukturen: **3**

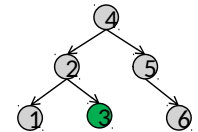
Array

4 2 1 3 5 6

verkettete Liste

4 → 2 → 1 → 3 → 5 → 6

binärer Suchbaum



In welcher Datenstruktur ist der Elementzugriff am schnellsten?

- Array
- verkettete Liste
- binärer Suchbaum

Aufgabe: Datenstrukturen (IV)

Element-Zugriff in verschiedenen Datenstrukturen: **3**

Array

4 2 1 3 5 6

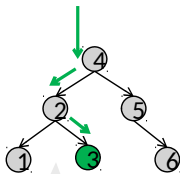
Direkter Zugriff: $O(1)$

verkettete Liste

4 → 2 → 1 → 3 → 5 → 6

Durchlaufen der kompletten Liste (worst-case): $O(n)$

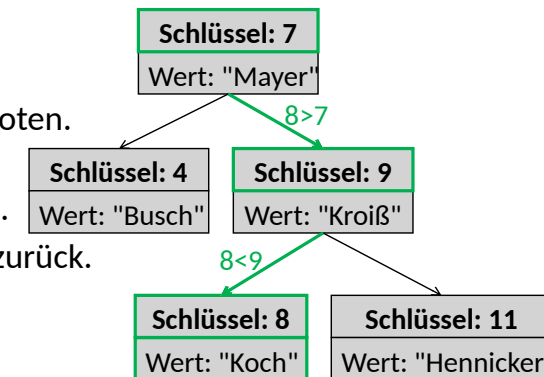
binärer Suchbaum



Binäre Suche im Baum (average-case): $O(\log n)$

Suche im binären Suchbaum: Suche Zimmer 8

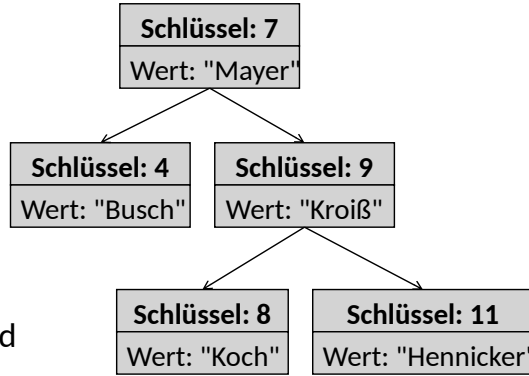
- Vergleiche 8 mit dem Schlüssel der Wurzel.
- Da $8 > 7$, gehe zum rechten Kindknoten.
- Vergleiche 8 mit dem Schlüssel des rechten Kindknoten.
- Da $8 < 9$, gehe zum linken Kindknoten.
- Vergleiche 8 mit dem Schlüssel des linken (zweiten) Kindknoten.
- Da $8 == 8$, gebe Ergebnis "Koch" zurück.



Suche im binären Suchbaum: Allgemein

Vergleiche den gesuchten Schlüssel k_{gesucht} mit dem Schlüssel k_{akt} des aktuellen Knotens

1. Falls $k_{\text{gesucht}} == k_{\text{akt}}$, gib den zugehörigen Wert aus.
2. Falls $k_{\text{gesucht}} < k_{\text{akt}}$, gehe zum linken Kindknoten und wiederhole die Suche dort.
3. Falls $k_{\text{gesucht}} > k_{\text{akt}}$, gehe zum rechten Kindknoten und wiederhole die Suche dort.



Suche im binären Suchbaum: Java (I)

```
public class Node {
    ...
    public Object findElement(int keyGesucht) {
        if (keyGesucht == this.key) return this.value;

        else if (keyGesucht < this.key) {

            return this.left.findElement(keyGesucht);

        }

        else {

            return this.right.findElement(keyGesucht);

        }
    }
}
```

1. Falls $k_{\text{gesucht}} == k_{\text{akt}}$, gib den zugehörigen Wert aus.
2. Falls $k_{\text{gesucht}} < k_{\text{akt}}$, gehe zum linken Kindknoten und wiederhole die Suche dort.
3. Falls $k_{\text{gesucht}} > k_{\text{akt}}$, gehe zum rechten Kindknoten und wiederhole die Suche dort.

Suche im binären Suchbaum: Java (I)

```
public class Node {
    ...
    public Object findElement(int keyGesucht) {
        if (keyGesucht == this.key) return this.value;

        else if (keyGesucht < this.key) {
            if (this.left == null)
                return null;
            else
                return this.left.findElement(keyGesucht);
        }

        else {
            if (this.right == null)
                return null;
            else
                return this.right.findElement(keyGesucht);
        }
    }
}
```

Was bedeutet es, wenn `this.left == null` oder `this.right == null` gilt?

- a) Dieser Fall kann bei einer Suche nie eintreten und kann missachtet werden.
- b) Die Suche hat den falschen Ast des Baums durchsucht.
- c) Das gesuchte Element kommt nicht im Baum vor.

Suche im binären Suchbaum: Java (I)

```
public class Node {
    ...
    public Object findElement(int keyGesucht) throws NoSuchElementException {
        if (keyGesucht == this.key) return this.value;

        else if (keyGesucht < this.key) {
            if (this.left == null)
                throw new NoSuchElementException();
            else
                return this.left.findElement(keyGesucht);
        }

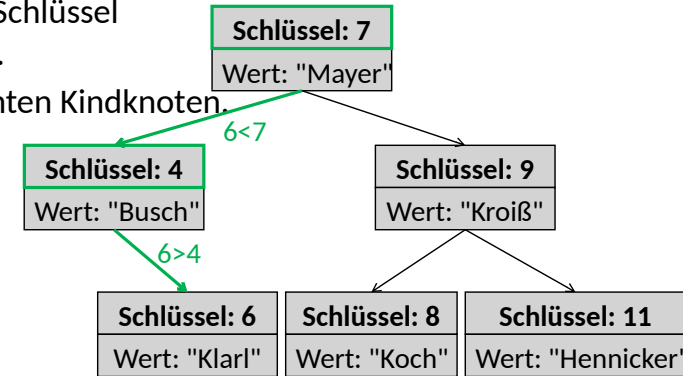
        else {
            if (this.right == null)
                throw new NoSuchElementException();
            else
                return this.right.findElement(keyGesucht);
        }
    }
}
```

Suche im binären Suchbaum: Java (II)

```
public class BinTree {
    ...
    public Object findElement(int keyGesucht) {
        if (this.root == null) throw new NoSuchElementException();
        else return this.root.findElement(keyGesucht);
    }
}
```

Einfügen im binären Suchbaum: Füge Zimmer 6 ein

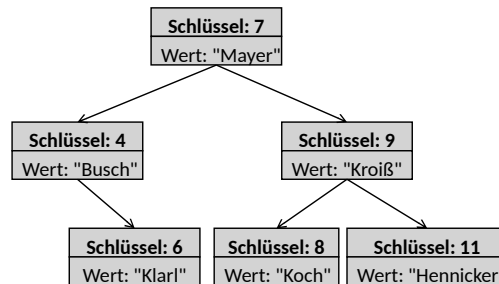
1. Vergleiche 6 mit dem Schlüssel der Wurzel.
2. Da $6 < 7$, gehe zum linken Kindknoten.
3. Vergleiche 6 mit dem Schlüssel des linken Kindknoten.
4. Da $6 > 4$, gehe zum rechten Kindknoten.
5. Da der rechte Kindknoten leer ist, füge einen neuen Kindknoten ein.



Einfügen im binären Suchbaum: Allgemein

Vergleiche den einzufügenden Schlüssel $k_{\text{einfuegen}}$ mit dem Schlüssel k_{akt}

1. Falls $k_{\text{einfuegen}} == k_{\text{akt}}$, überschreibe den bisherigen Wert.
2. Falls $k_{\text{einfuegen}} < k_{\text{akt}}$, gehe zum linken Kindknoten:
 - a. Falls der linke Kindknoten leer ist, füge dort einen neuen Knoten ein.
 - b. Ansonsten wiederhole die Suche dort.
3. Falls $k_{\text{einfuegen}} > k_{\text{akt}}$, gehe zum rechten Kindknoten:
 - a. Falls der rechte Kindknoten leer ist, füge dort einen neuen Knoten ein.
 - b. Ansonsten wiederhole die Suche dort.



Einfügen im binären Suchbaum: Java (I)

```
public class Node {
    ...
    public void insert(int keyEinfuegen, Object value) {
        if (keyEinfuegen == this.key) this.value = value;
        else if (keyEinfuegen < this.key) {
            if (this.left == null) this.left = new Node(keyEinfuegen, value);
            else this.left.insert(keyEinfuegen, value);
        }
        else {
            if (this.right == null) this.right = new Node(keyEinfuegen, value);
            else this.right.insert(keyEinfuegen, value);
        }
    }
}
```

1. Falls $k_{\text{einfuegen}} == k_{\text{akt}}$, überschreibe den bisherigen Wert.

2. Falls $k_{\text{einfuegen}} < k_{\text{akt}}$, gehe zum linken Kindknoten:
 a. Falls der linke Kindknoten leer ist, füge dort einen neuen Knoten ein.
 b. Ansonsten wiederhole die Suche dort.

3. Falls $k_{\text{einfuegen}} > k_{\text{akt}}$, gehe zum rechten Kindknoten:
 a. Falls der rechte Kindknoten leer ist, füge dort einen neuen Knoten ein.
 b. Ansonsten wiederhole die Suche dort.

Einfügen im binären Suchbaum: Java (II)

```
public class BinTree {  
    ...  
    public void insert(int key, Object value) {  
        if (this.root == null) this.root = new Node(key, value);  
        else this.root.insert(key, value);  
    }  
}
```