IIF
INSTITUT FÜR
INFORMATIK
FREIBURG

Vorlesung Informatik 2
Algorithmen und Datenstrukturen

(19 - Bäume: Durchlaufreihenfolgen)

Prof. Dr. Susanne Albers

Binäre Suchbäume

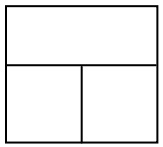
Binärbäume zur Speicherung von Mengen von Schlüsseln (in den inneren Knoten der Bäume), so dass die Operationen

- Suchen (find)
- Einfügen (insert)
- Entfernen (remove, delete)

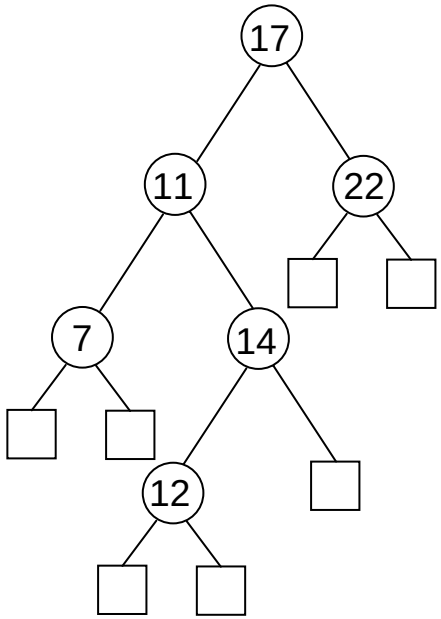
unterstützt werden.

Suchbaumeigenschaft: Die Schlüssel im linken Teilbaum eines Knotens *p* sind alle kleiner als der Schlüssel von *p*, und dieser ist wiederum kleiner als sämtliche Schlüssel im rechten Teilbaum von *p*.

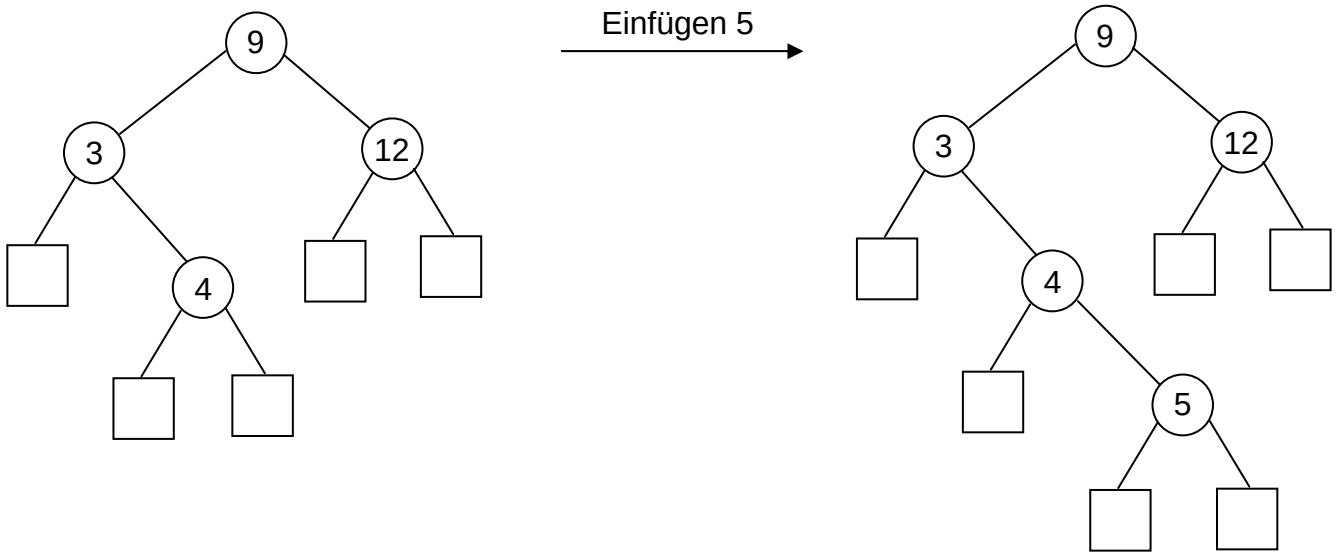
Implementierung:



Beispiel für Suchen, Einfügen, Entfernen



Natürliche Bäume



- Baum-Struktur hängt von Einfügereihenfolge in anfangs leeren Baum ab
- Höhe kann linear zunehmen, sie kann aber auch in O(log *n*) sein, genau $\lceil \log_2(n+1) \rceil$

Durchlaufreihenfolgen in Bäumen

Durchlaufreihenfolgen zum Besuchen der Knoten des Baums

- zur Ausgabe
- zur Berechnung von Summe, Durchschnitt, Anzahl der Schlüssel . . .
- zur Änderung der Struktur

Wichtigste Durchlaufreihenfolgen:

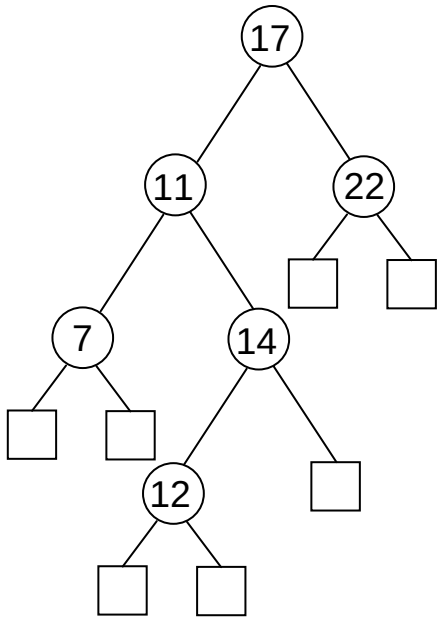
1. Hauptreihenfolge = Preorder = WLR
   besuche erst Wurzel, dann rekursiv linken und rechten Teil-Baum falls vorhanden
2. Nebenreihenfolge = Postorder = LRW
3. Symmetrische Reihenfolge = Inorder = LWR
4. die Spiegelbild-Varianten von 1-3

Hauptreihenfolge (Preorder)

Ausgehend von der Wurzel *p* eines Baums ist die Hauptreihenfolge wie folgt rekursiv definiert:

Durchlaufen aller Knoten eines Binärbaumes mit Wurzel p in Hauptreihenfolge:

Besuche p,
durchlaufe den linken Teilbaum von p in Hauptreihenfolge,
durchlaufe den rechten Teilbaum von p in Hauptreihenfolge.



Hauptreihenfolge Programm

```
// Hauptreihenfolge; WLR
void preOrder (){
    preOrder (root);
    System.out.println ();
}
void preOrder (SearchNode n){
    if (n == null) return;
    System.out.print (n.content+" ");
    preOrder (n.left);
    preOrder (n.right);
}
```

```
// Nebenreihenfolge; LRW
void postOrder (){
    postOrder (root);
    System.out.println ();
}
// ...
```

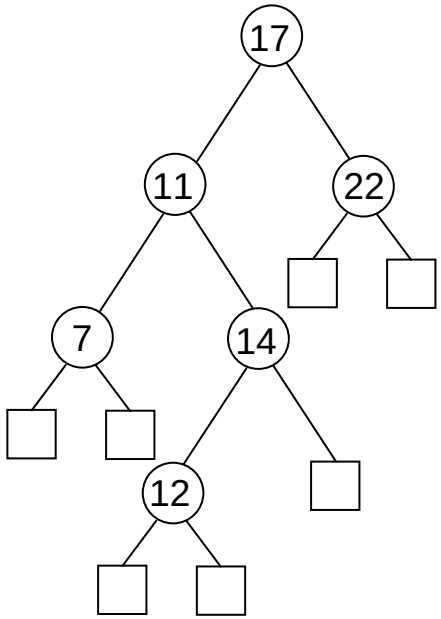
Symmetrische Reihenfolge (Inorder)

Die Durchlaufreihenfolge ist: erst linker Teilbaum, dann Wurzel, dann rechter Teilbaum:

```
// Symmetrische; LWR
void inOrder (){
    inOrder (root);
    System.out.println ();
}
void inOrder (SearchNode n){
    if (n == null) return;

    inOrder (n.left);

    System.out.print (n.content+" ");

    inOrder (n.right);
}
// Nebenreihenfolge; LRW
// ...
```

Die anderen Durchlaufreihenfolgen werden analog implementiert.

Beispiel



Preorder:

17, 11, 7, 14, 12, 22

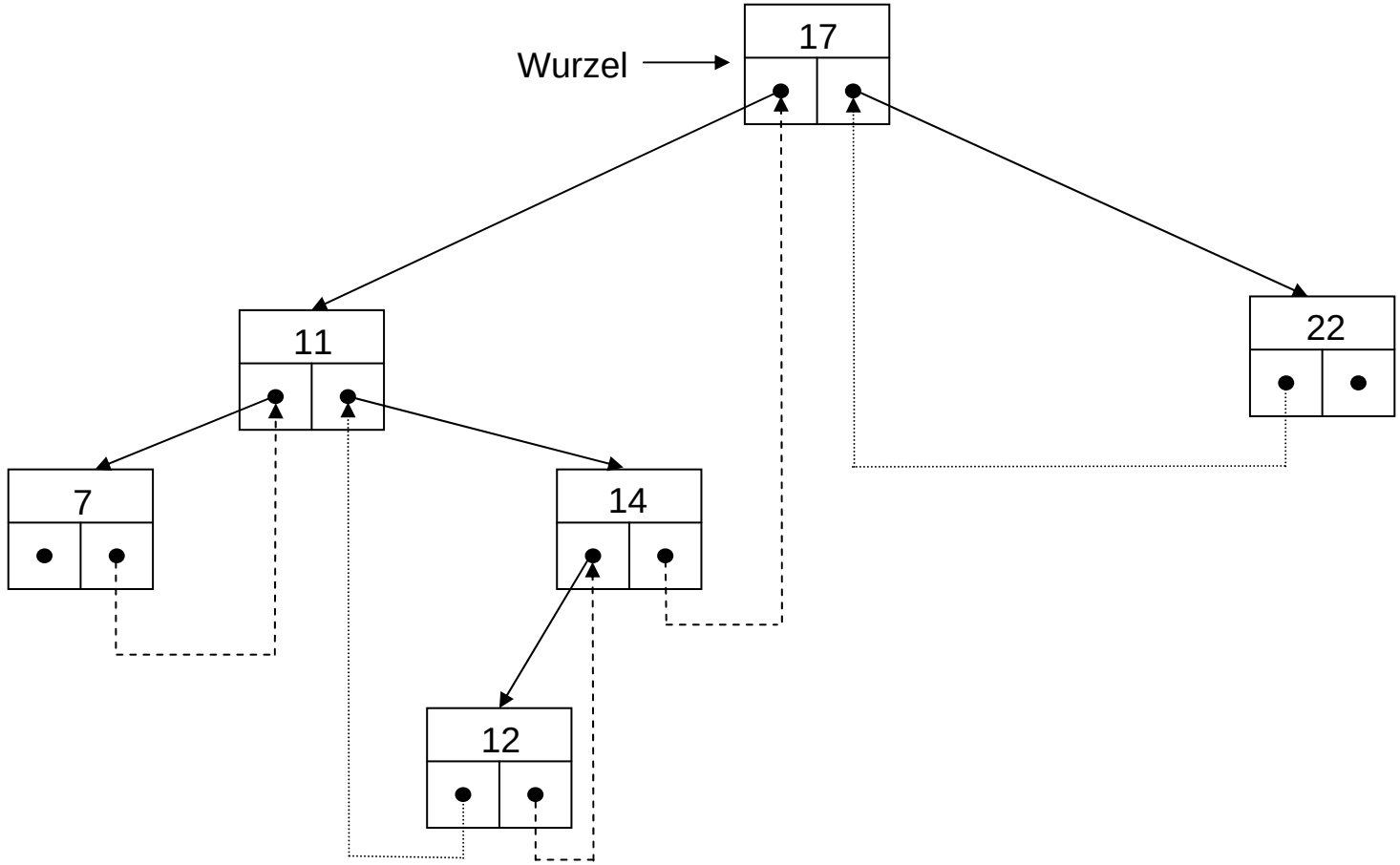
Postorder:

7, 12, 14, 11, 22, 17

Inorder:

7, 11, 12, 14, 17, 22

Nichtrekursive Varianten mit gefädelten Bäumen



Rekursion kann vermieden werden, wenn man anstelle der `null`-Referenzen sogenannte Fädelungszeiger auf die Vorgänger bzw. Nachfolger verwendet.

Sortieren mit natürlichen Suchbäumen

Idee: Bau für die Eingabefolge einen natürlichen Suchbaum auf und gib die Schlüssel in symmetrischer Reihenfolge (Inorder) aus.

Bemerkung: Abhängig von der Eingabereihenfolge kann der Suchbaum degenerieren.

Komplexität**:** Abhängig von der internen Pfadlänge

Schlechtester Fall: Sortierte Eingabe: $\Rightarrow \Omega(n^2)$ Schritte.

Bester Fall: Es entsteht ein vollständiger Suchbaum mit minimal möglicher Höhe von etwa log n. n mal Einfügen und Ausgeben ist daher in Zeit O(*n* log *n)* möglich.

Mittlerer Fall: ?