EINI LW

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 11/12

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@udo.edu
http://ls1-www.cs.uni-dortmund.de

technische universität dortmund

Dr. Lars Hildebrand

- **Kapitel 8**
  Dynamische Datenstrukturen
  - Listen
  - Bäume

- **Unterlagen**
- Echtle, Goedicke: Einführung in die objektorientierte Programmierung mit Java, dpunkt-Verlag.
- Doberkat, Dissmann: Einführung in die objektorientierte Programmierung mit Java, Oldenbourg-Verlag, 2. Auflage

Wiederholung

Dr. Lars Hildebrand

Lineare Liste als klassische, einfache dynamische Datenstruktur

- Grundkonstruktion: Objekte haben Referenz auf Objekt der eigenen Klasse
- Typische Operationen: Anlegen, Finden von Elementen, Einfügen von Elementen, Durchlaufen aller Elemente, Löschen eines Elementes
- Unterschiedliche Varianten
  - einfache Liste, Liste mit Kopf & Fuß Attribut, doppelt verkettete Liste
- Operationen lassen sich auch leicht rekursiv formulieren
- Aufwand für Operationen (worst case):
  - Einfügen am Anfang: O(1)
  - Einfügen am Ende: ohne Fuß-Attribut O(N), mit Fuß-Attribut O(1)
  - Suchen eines Elementes:
    - in unsortierter Liste: O(N)
    - in sortierter Liste: O(N), aber Abbruch vor Listenende außer bei fehlendem Element
  - Einfügen eines Elementes in eine sortierte Liste: O(N)

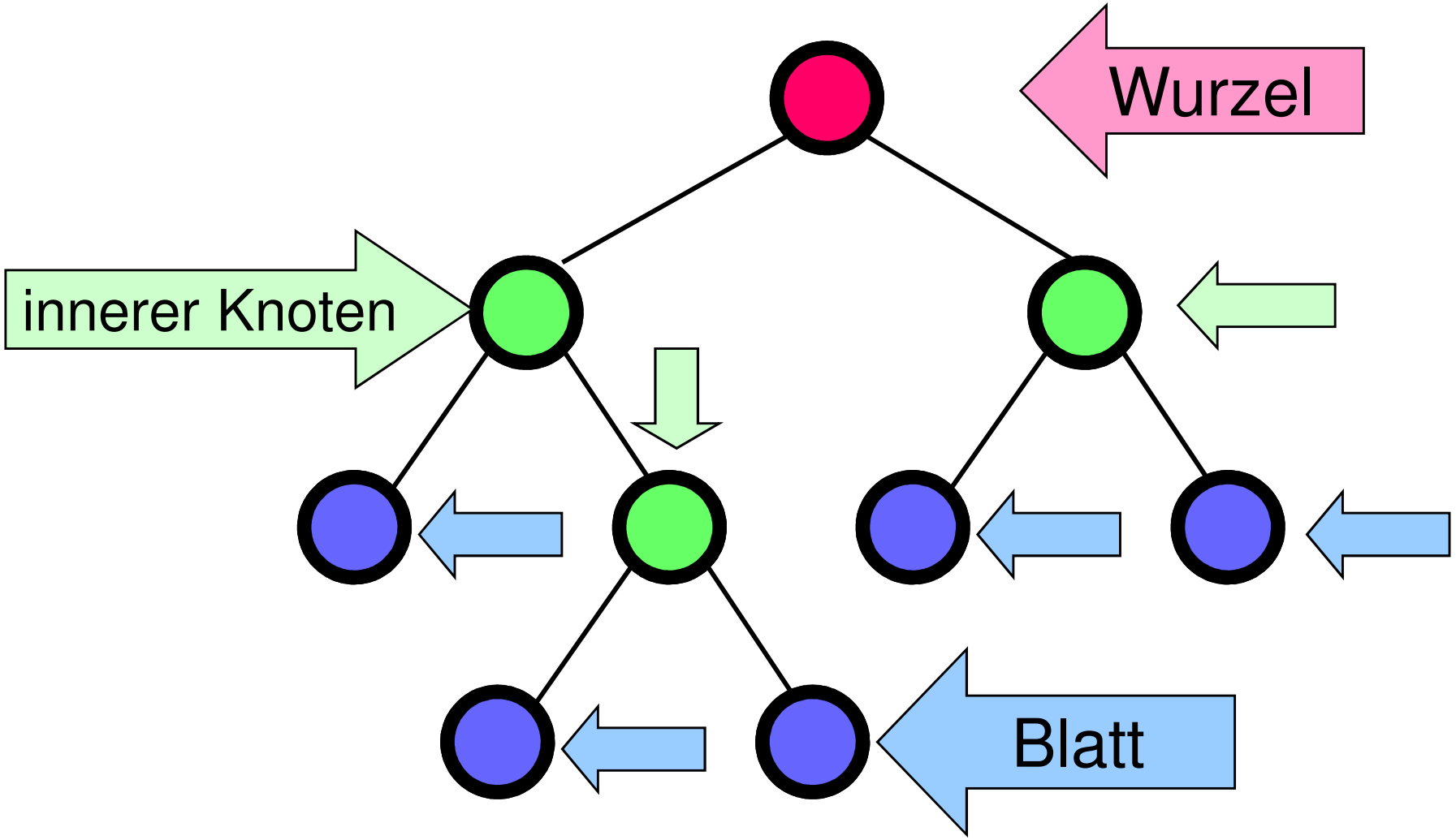
Allgemeines zu Bäumen

- Bäume sind
  - gerichtete, azyklische Graphen. Es gibt keine Zyklen zwischen Mengen von Knoten
  - hierarchische Strukturen. Man kommt von einer Wurzel zu inneren Knoten und letztlich zu Blättern
  - verkettete Strukturen, die dynamisch wachsen und schrumpfen können

- Binäre Bäume sind Bäume, in denen jeder Knoten maximal zwei Söhne hat

- Beispiele für die Anwendung binärer Bäume
  - **Heapsort**
  - binäre Suchbäume

Terminologie *Binäre Bäume*



Allgemeines zu Bäumen

Dr. Lars Hildebrand

- Typische Zugriffsmethoden
  - Einfügen einer Wurzel
  - Einfügen eines inneren Knotens
  - Entfernen der Wurzel
  - Entfernen eines inneren Knotens
  - Suchen
  - Nach links/rechts navigieren

technische universität dortmund
Dr. Lars Hildebrand

- **Aufgabe**:

  Suche ein Element x in einer geordneten Menge

- **Grundidee**: rekursiver Ansatz.
  - Beschaffe mittleres Element y der geordneten Menge
  - falls x = y: fertig
  - falls x < y: wende Verfahren rekursiv auf Teilmenge kleinerer Elemente an
  - falls x > y: wende Verfahren rekursiv auf Teilmenge größerer Elemente an
- **Beobachtung**:
  - In jedem Schritt wird die zu betrachtende Menge halbiert
  - → bei N Elementen also $\log_2(N)$ Schritte

Suche „in einer Hälfte“

Dr. Lars Hildebrand

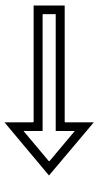
- **Grobidee (erfolgreiche Suche)**
  - Suchen in „geordneter Liste“ durch Überprüfen des "mittleren" Elementes + Fortsetzung in einer Hälfte
  - Beispiel:

Position:	1	2	3	4	5	6	7	8	9
Wert:	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

Suche „in einer Hälfte“

Dr. Lars Hildebrand

- **Grobidee (erfolgreiche Suche)**
  - Suchen in „geordneter Liste“ durch Überprüfen des "mittleren" Elementes + Fortsetzung in einer Hälfte
  - Beispiel:



Position:	1	2	3	4	5	6	7	8	9
Wert:	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

- Suche nach 19
  - Mitte: 5. Pos., Wert=8
  - 19 > 8
  - rechten Abschnitt wählen

Suche „in einer Hälfte“

Dr. Lars Hildebrand

- **Grobidee (erfolgreiche Suche)**
  - Suchen in „geordneter Liste“ durch Überprüfen des "mittleren" Elementes + Fortsetzung in einer Hälfte
  - Beispiel:

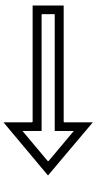
Position:	1	2	3	4	5	6	7	8	9
Wert:	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

- Suche nach 19
  - Mitte: 7. Pos., Wert = 19
  - 19 gefunden, fertig

Suche „in einer Hälfte“

Dr. Lars Hildebrand

- **Grobidee (erfolglose Suche)**
  - Suchen in „geordneter Liste“ durch Überprüfen des "mittleren" Elementes + Fortsetzung in einer Hälfte
  - Beispiel:



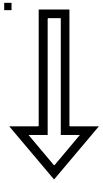
Position:	1	2	3	4	5	6	7	8	9
Wert:	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

- Suche nach 5
  - Mitte: 5. Pos., Wert = 8
  - 5 < 8
  - linken Abschnitt wählen

Suche „in einer Hälfte“

Dr. Lars Hildebrand

- **Grobidee (erfolglose Suche)**
  - Suchen in „geordneter Liste“ durch Überprüfen des "mittleren" Elementes + Fortsetzung in einer Hälfte
  - Beispiel:

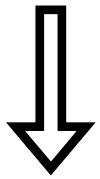


Position:	1	2	3	4	5	6	7	8	9
Wert:	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

- Suche nach 5
  - Mitte: 2. Pos. = 4
  - 4 > 8
  - rechten Abschnitt wählen

Suche „in einer Hälfte“

- **Grobidee (erfolglose Suche)**
  - Suchen in „geordneter Liste“ durch Überprüfen des "mittleren" Elementes + Fortsetzung in einer Hälfte
  - Beispiel:

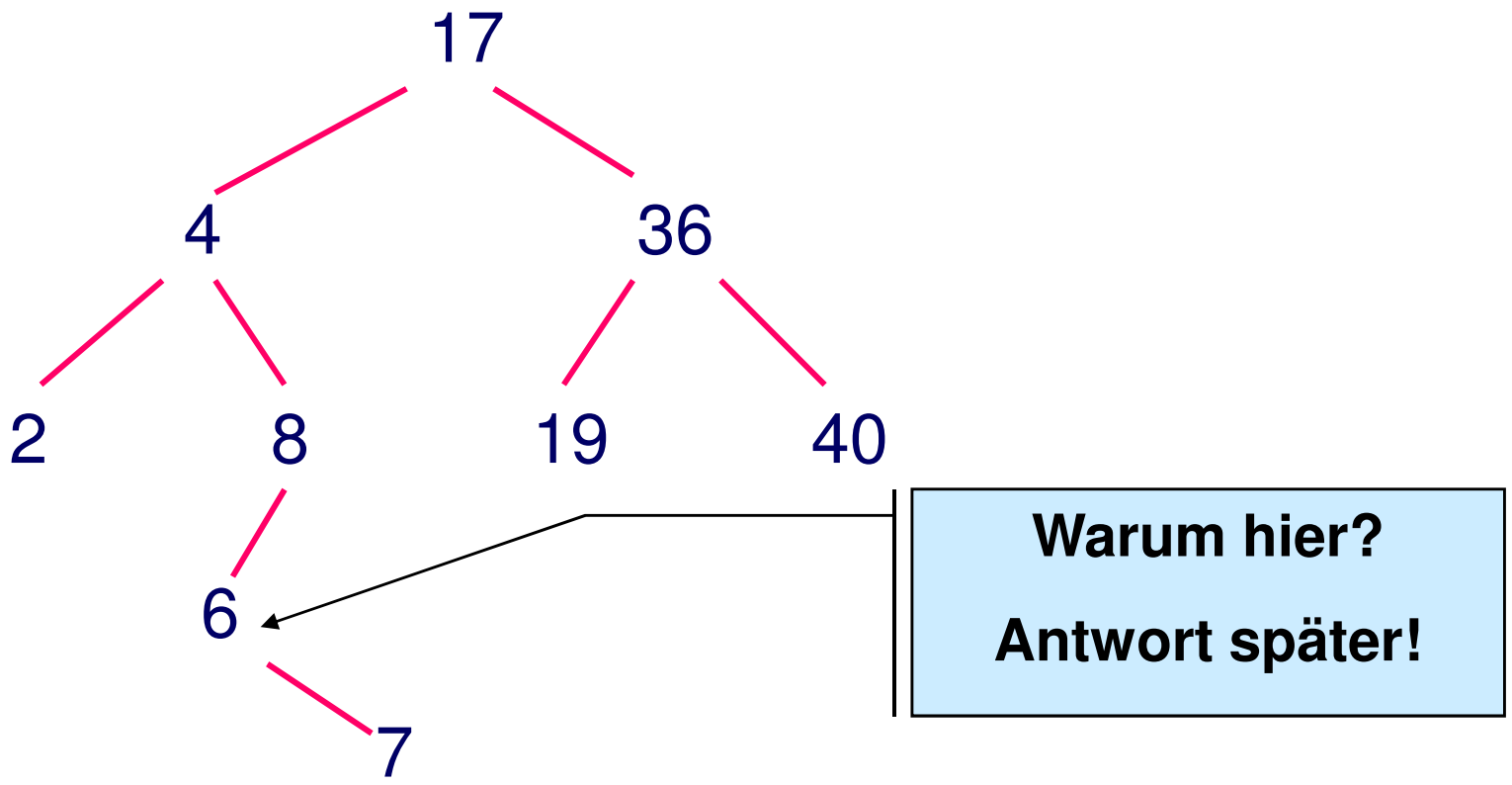


Position:	1	2	3	4	5	6	7	8	9
Wert:	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

- Suche nach 5
  - Mitte: 3. Pos. = 6
  - 6 < 8
  - keine weitere Hälfte vorhanden
  - 5 nicht gefunden, fertig

Suche „in einer Hälfte“

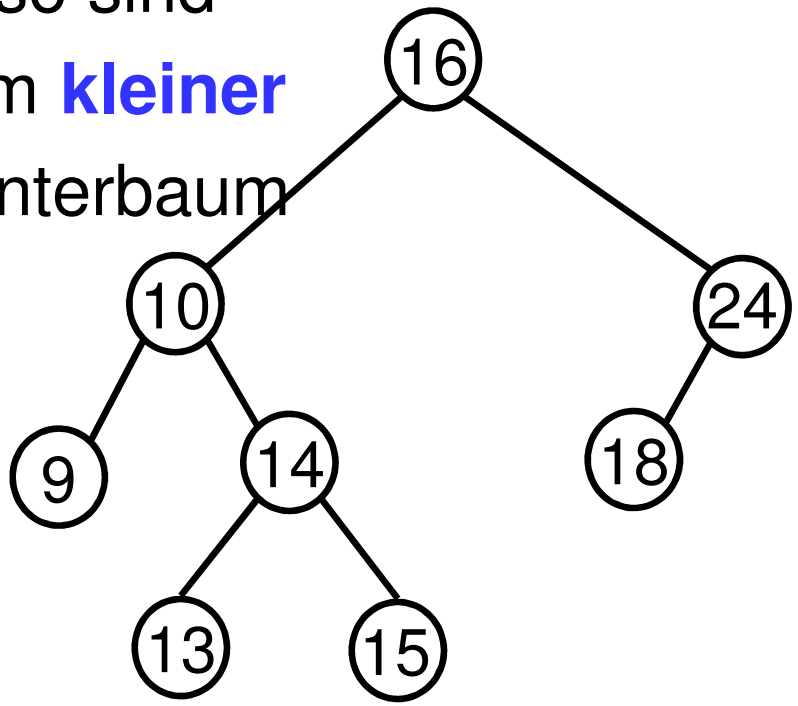
- **Aufgabe**: Trage die Zahlen 17, 4, 36, 2, 8, 19, 40, 6, 7 in eine baumförmige Struktur so ein,
  - dass die Suche „in einer Hälfte“ effektiv unterstützt wird:



technische universität dortmund
Dr. Lars Hildebrand

Definition:

- Sei *B* ein binärer Baum, dessen Knoten mit ganzen Zahlen beschriftet sind. *B* heißt **binärer Suchbaum**, falls gilt:
  - *B* ist **leer** oder
  - der linke und der rechte **Unterbaum** von *B* sind **binäre Suchbäume**,
  - ist *w* die Beschriftung der Wurzel, so sind alle Elemente im **linken** Unterbaum **kleiner** als *w*, alle Elemente im **rechten** Unterbaum **größer** als *w*.



technische universität dortmund
Dr. Lars Hildebrand

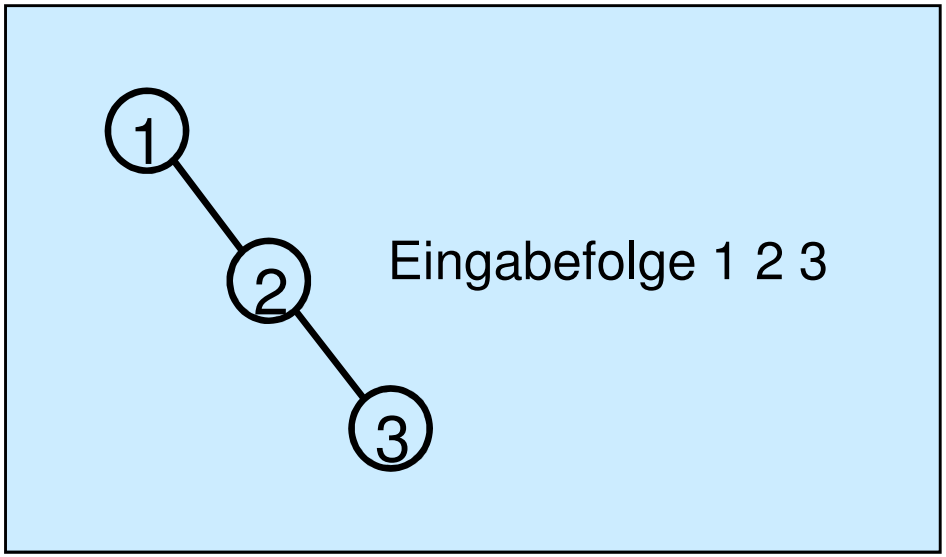
- Der Aufbau eines binären Suchbaums erfolgt durch wiederholtes Einfügen in einen (anfangs) leeren Baum

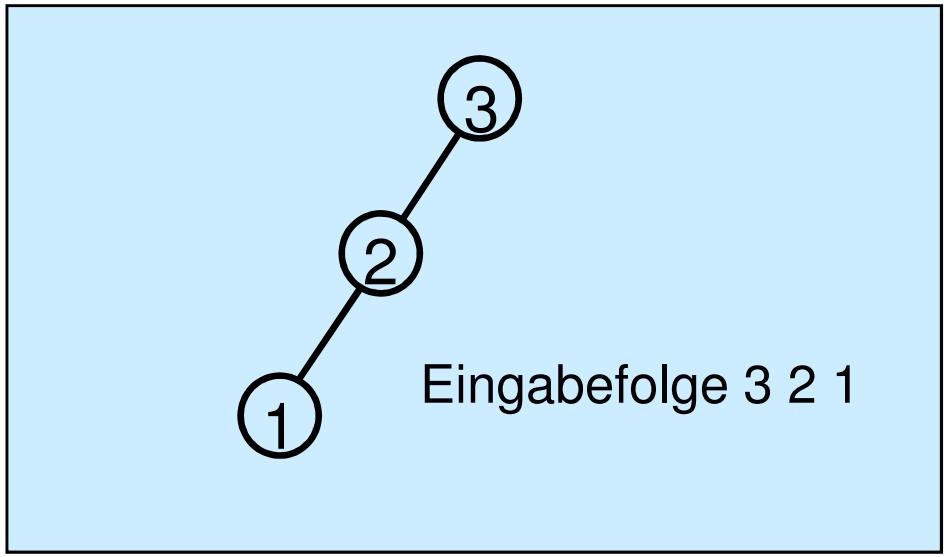
- Die Reihenfolge der Werte, die in einen binären Suchbaum eingefügt werden, bestimmt die Gestalt des Baumes

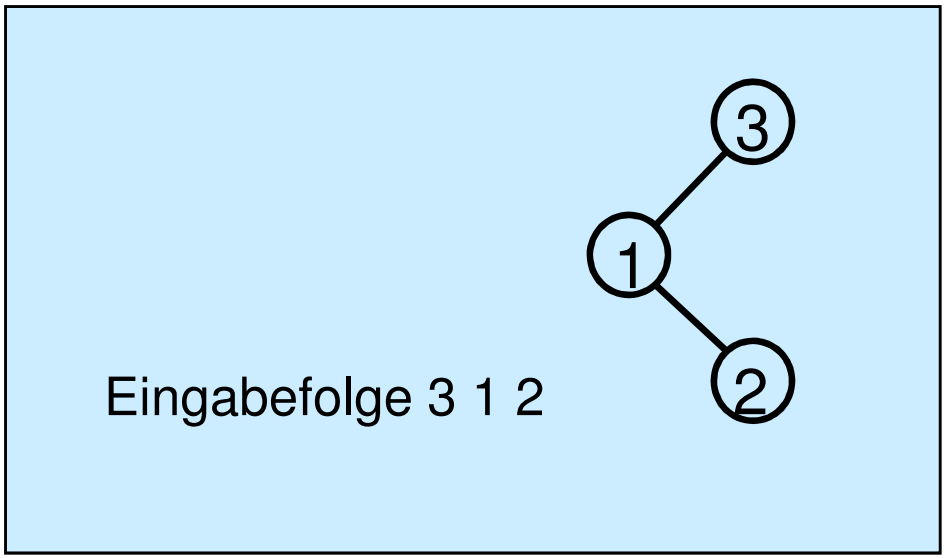
- Eine Menge von Werten kann bei unterschiedlichen Eingabereihenfolgen zu verschiedenen Repräsentationen als Baum führen.

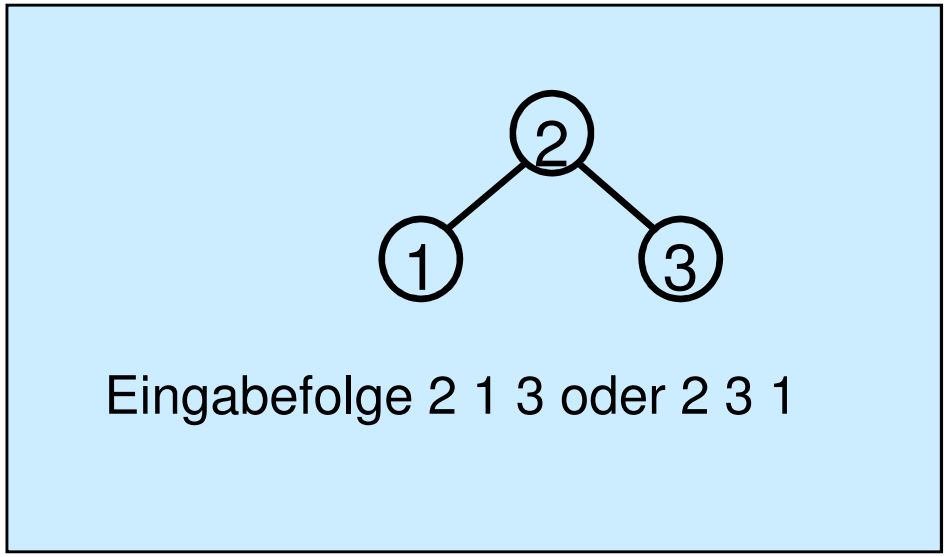
technische universität dortmund
Dr. Lars Hildebrand

Beispiele









Binäre Suchbäume - die Klasse Knoten

technische universität dortmund

Dr. Lars Hildebrand

```
class Knoten {
        private int wert;
        private Knoten links, rechts;

        Knoten(int i) {
                wert = i; links = rechts = null; }

        void SetzeWert(int i) { wert = i; }
        int HoleWert() { return wert; }
        void SetzeLinks(Knoten k) { links = k; }
        Knoten HoleLinks() { return links; }
        void SetzeRechts(Knoten k) { rechts = k; }
        Knoten HoleRechts() { return rechts; }
};
```

technische universität dortmund
Dr. Lars Hildebrand

Algorithmus für das Einfügen von Knoten

- Gegeben seien ein binärer Suchbaum *B* und eine ganze Zahl *k*, die in *B* eingefügt werden soll. Es können vier Fälle auftreten:
  - *B* ist leer:

    Erzeuge einen neuen Knoten, weise ihn *B* als Wurzel zu und setze *wurzel.wert auf k.*
  - *B* ist nicht leer und *wurzel.wert = k*:

    Dann ist nichts zu tun, da keine doppelten Einträge vorgenommen werden sollen.
  - *B* ist nicht leer und *wurzel.wert < k*:

    Füge k in den rechten Unterbaum von *B* ein.
  - *B* ist nicht leer und *wurzel.wert > k*:

    Füge k in den linken Unterbaum von *B* ein.

Die Klasse Binary Search Tree (BST)

Dr. Lars Hildebrand

```
public class BinarySearchTree {
  private Knoten wurzel;

  public BinarySearchTree() {
     wurzel = null;
  }

  public void FuegeEin(int i) {
       wurzel = FuegeEin(wurzel, i);
  }
```

Einfügen in den Baum

```
  private Knoten FuegeEin(Knoten einKnoten, int wert) {
        if (einKnoten == null) // wurzel ist leer
                einKnoten = new Knoten(wert);
        else {
               if (wert < einKnoten.HoleWert()) // links weiter
                      einKnoten.SetzeLinks
                     (FuegeEin(einKnoten.HoleLinks(), wert));
               if (wert > einKnoten.GibWert()) // rechts weiter
                      einKnoten.SetzeRechts
                     (FuegeEin(einKnoten.HoleRechts(), wert));
        }
        return einKnoten;
  }
}
```

technische universität dortmund

Dr. Lars Hildebrand

Algorithmus für die Suche von Knoten

- Der am Beginn dieses Kapitels skizzierte Algorithmus für das binäre Suchen lässt sich nun mit der durch die Methode **`FuegeEin`** aufgebauten Datenstruktur recht einfach realisieren.

  Gegeben sind ein binärer Suchbaum *B* und eine Zahl *k*,
  die in dem Baum *B* gesucht werden soll:

  - *B* ist leer: *k* kann nicht im Baum sein.
  - *B* ist nicht leer, so betrachtet man die Fälle
    - *wurzel.wert* = *k*: *k* ist gefunden, d.h. bereits in dem Baum *B* vorhanden.
    - *wurzel.wert* < *k*: Suche im rechten Unterbaum von *B*.
    - *wurzel.wert* > *k*: Suche im linken Unterbaum von *B*.

BinarySearchTree - Suchen

technische universität dortmund

Dr. Lars Hildebrand

```
public class BinarySearchTree {
   ...
   public boolean Suche(int i) {
     return Suche(wurzel, i);
   }
   private boolean Suche(Knoten einKnoten, int i) {
        boolean gefunden = false;
        if (einKnoten != null) {
               if (einKnoten.HoleWert() == i)
                   gefunden = true;
               if (einKnoten.HoleWert() < i)
                    gefunden = Suche(einKnoten.HoleRechts(), i);
               if (einKnoten.HoleWert() > i)
                    gefunden = Suche(einKnoten.HoleLinks(), i);
        }
        return gefunden;
   }
}
```

technische universität dortmund

Dr. Lars Hildebrand

Definition:

- Ist B ein binärer Baum, so definiert man die Höhe *h(B)* von *B* rekursiv durch:

$$h(B) := \begin{cases} 0, & \text{falls B leer ist} \\ 1 + max\{h(B\mathbf{1}), h(B\mathbf{2})\}, & \text{falls B}\mathbf{1}\text{ und B}\mathbf{2}\text{ linker bzw. rechter Unterbaum von B sind} \end{cases}$$

- Ist *B* ein binärer Suchbaum mit *h(B)*=n, so enthält B mindestens *n* und höchstens 2^n-1 Knoten.
  - *n*, wenn der Baum zur Liste degeneriert ist,
  - 2^n-1, wenn jeder von 2^{n-1}-1 inneren Knoten genau zwei Söhne und jedes von 2^{n-1} Blättern keine Söhne hat.

technische universität dortmund

Dr. Lars Hildebrand

In rekursiven Datenstrukturen werden Aussagen häufig mittels vollständiger Induktion bewiesen.

- Schema:
  - Spezifikation der Behauptung S(B), wobei B ein Baum ist.
- Beweis
  - Induktionsanfang: Zeige, dass S(B) für Bäume mit keinem Knoten gilt.
  - Induktionsannahme: B ist ein Baum mit Wurzel w und k $\geq$1 Unterbäumen B_1, ..., B_k. Annahme, dass S(B_i) für i $\in$ 1,2, ..., k gilt.
  - Induktionsschritt: zeige, dass S(B) unter der Induktionsannahme gilt.

Suchen in binären Suchbäumen

Dr. Lars Hildebrand

Behauptung: In einem beliebigen binären Suchbaum *B* braucht man für eine erfolglose Suche maximal h(*B*) Vergleiche

- **Beweis**: durch vollständige Induktion nach dem Aufbau von *B*.
- **Induktionsanfang**: Ist *B* leer, also h(*B*)=0, so ist kein Vergleich nötig.
- **Induktionsschritt**: Wenn für zwei Bäume B_1 und B_2 mit der Höhe ≤ n gilt, dass in jedem eine erfolglose Suche in maximal n Vergleichen möglich ist, so gilt für den binären Suchbaum mit einer neuen Wurzel und deren Unterbäume B_1, B_2, dass maximal n+1 Vergleiche nötig sind.
- **Beweis des Induktionsschrittes**: Durch Wurzelvergleich (ein Vergleich) wird ermittelt, ob im linken oder rechten Teilbaum weitergesucht werden muss. Die maximale Zahl der Vergleiche in einem Teilbaum ist n. Also, braucht man insgesamt maximal n+1 Vergleiche. In einem Baum der Höhe n+1 braucht man also maximal n+1 Vergleiche.

technische universität dortmund

Dr. Lars Hildebrand

Daraus ergibt sich:

- Bei einer erfolglosen Suche in einem binären Suchbaum mit n Elementen sind mindestens *log n* (Basis 2) und höchstens n Vergleiche notwendig.

- Der günstige Fall (log n Vergleiche) gilt in einem gleichgewichtigen Baum. Der ungünstige (n Vergleiche) gilt in einem vollständig degenerierten Baum, der beispielsweise immer dann entsteht, wenn die Elemente in sortierter Reihenfolge eintreffen.

- Um diese Unsicherheit auszuräumen (und somit eine Laufzeit auf der Basis von log n Vergleichen sicherzustellen), werden balancierte, binäre Suchbäume benutzt.

Suchen in binären Suchbäumen

Dr. Lars Hildebrand

- Eine Art balancierter, binärer Suchbäume sind die AVL-Bäume (nach ihren Erfindern Adelson, Velskii, Landis).

- Def.: Ein **AVL-Baum** ist ein binärer Suchbaum, in dem sich für jeden Knoten die Höhen seiner zwei Teilbäume um höchstens 1 unterscheiden.

- Einfüge- und Entferne-Operationen werden dann etwas aufwendiger, aber dafür ist die Suche auch in ungünstigen Fällen effizienter (vgl Literatur).

technische universität dortmund

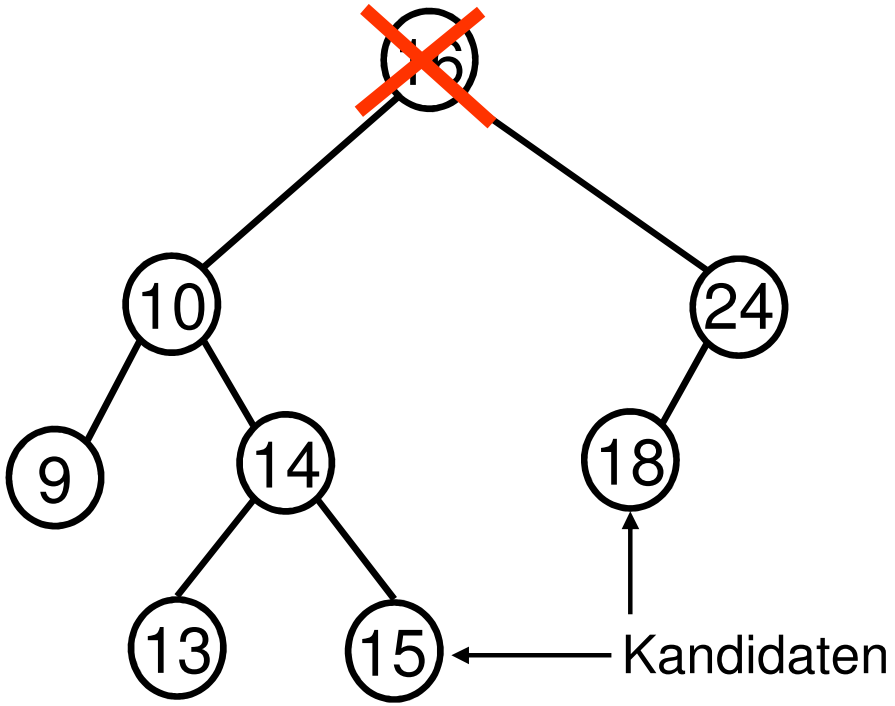
Dr. Lars Hildebrand

Algorithmus für das Entfernen

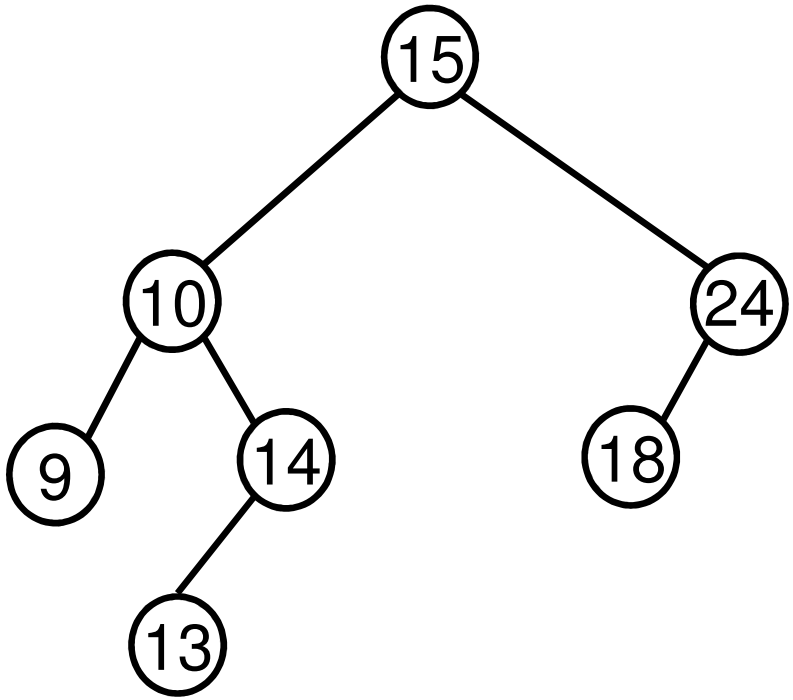
- Entfernen der Wurzel führt zur Konstruktion eines neuen binären Suchbaums
- Darum: Finden eines Knotens, der an die Stelle der Wurzel gesetzt wird und die Kriterien für einen neuen binären Suchbaum erfüllt
- Der Knoten muss größer als die Wurzel des linken Unterbaumes sein und kleiner als die Wurzel des rechten Unterbaumes

Entfernen der Wurzel - Beispiel

Dr. Lars Hildebrand



Situation vor dem Löschen



Situation nach dem Löschen

technische universität dortmund

Dr. Lars Hildebrand

Algorithmus für das Entfernen

- Es wird der Knoten mit der größten Beschriftung im linken Unterbaum genommen
- Dieser Knoten wird entfernt und als Wurzel eingesetzt
- Ist der linke Unterbaum einer Wurzel leer, nimmt man analog zur vorgestellten Methode das kleinste Element der rechten Wurzel
- Ist der Unterbaum einer Wurzel leer, kann auch auf eine Umgestaltung des Baumes verzichtet werden: Wird die Wurzel entfernt, bildet der verbleibende Unterbaum wieder einen binären Baum
- Wird ein innerer Knoten aus einem binären Suchbaum entfernt, stellt dieser Knoten die Wurzel eines Unterbaumes dar und diese Wurzel wird dann entfernt

technische universität dortmund

Dr. Lars Hildebrand

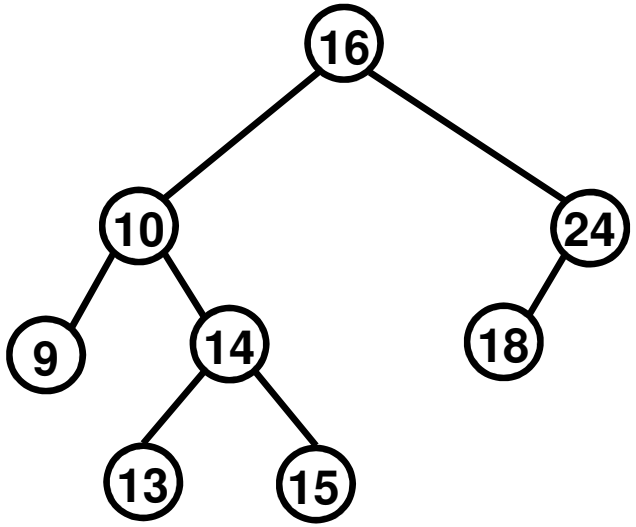
- **Tiefendurchlauf**: Hier wird von einem Knoten aus in die Tiefe gegangen, indem einer der Söhne besucht wird und dann dessen Söhne usw. Erst wenn man die Blätter erreicht hat, beginnt der Wiederaufstieg.
  - Preorder-Durchlauf
  - Inorder-Durchlauf
  - Postorder-Durchlauf

- **Breitendurchlauf**: Mit dem Besuch eines Knotens werden auch seine Nachbarn besucht.
  „Schichtweises Abtragen“

Tiefendurchlauf / Preorder

Dr. Lars Hildebrand

```
void PreOrder() {
  PreOrder(wurzel);
}
private void PreOrder(Knoten aktuell) {
  if (aktuell != null) {
       System.out.println(aktuell.GibWert());
       PreOrder(aktuell.GibLinks());
       PreOrder(aktuell.GibRechts());
  }
}
```

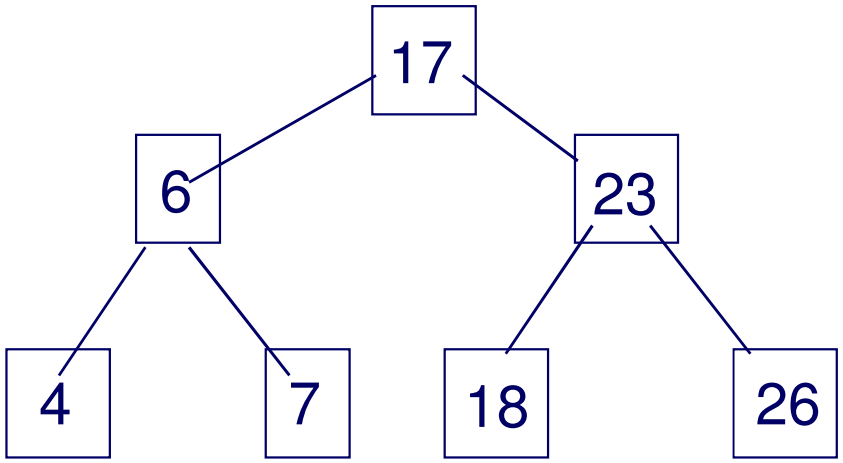


Reihenfolge der besuchten Knoten: 16, 10, 9, 14, 13, 15, 24, 18

Tiefendurchlauf / Preorder

technische universität dortmund
Dr. Lars Hildebrand

Preorder (w / BLinks BRechts) = Druck(w)
Preorder(Wurzel BLinks)
Preorder(Wurzel BRechts)

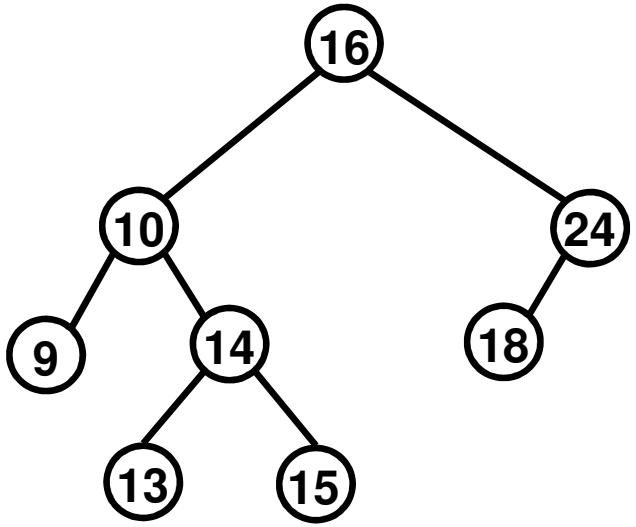


17	6	4	7	23	18	26

Tiefendurchlauf / Inorder

Dr. Lars Hildebrand

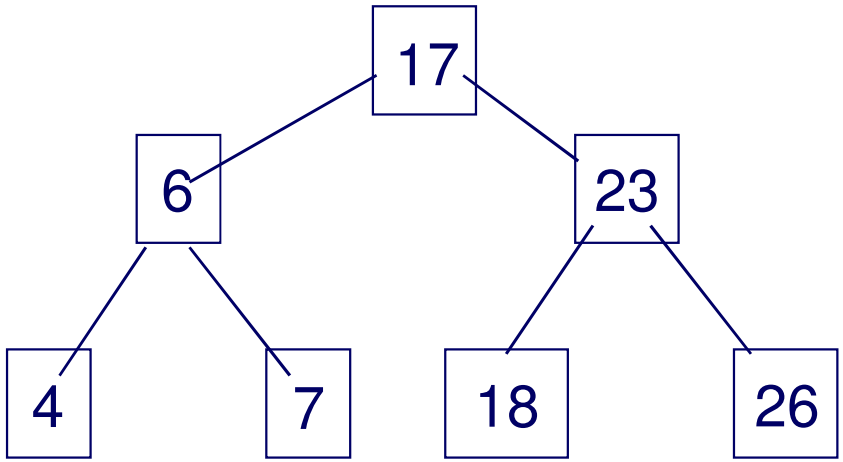
```
void InOrder() {
  InOrder(wurzel);
}
private void InOrder(Knoten aktuell) {
  if (aktuell != null) {
       InOrder(aktuell.GibLinks());
       System.out.println(aktuell.GibWert());
       InOrder(aktuell.GibRechts());
  }
}
```

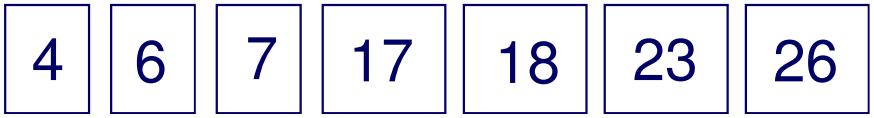


Reihenfolge der besuchten Knoten: 9, 10, 13, 14, 15, 16, 18, 24

Tiefendurchlauf / Inorder

technische universität dortmund
Dr. Lars Hildebrand





Inorder (w / BLinks BRechts) =

Inorder(Wurzel BLinks)

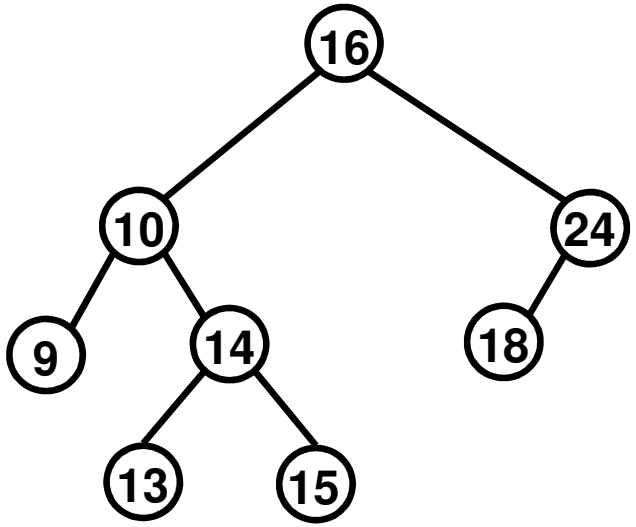
Druck(w)

Inorder(Wurzel BRechts)

Tiefendurchlauf / Postorder

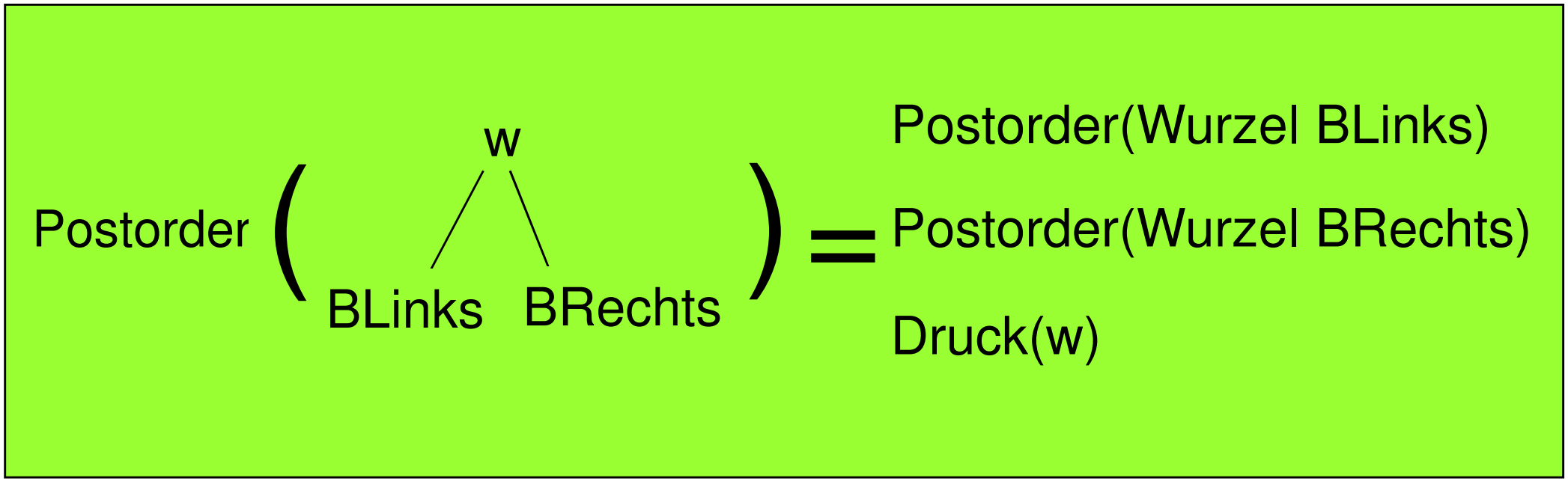
Dr. Lars Hildebrand

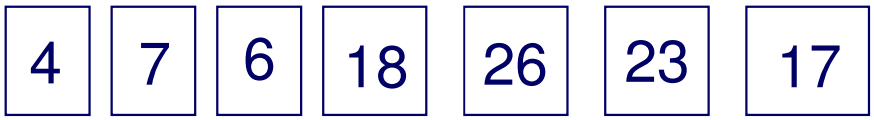
```
void PostOrder() {
  PostOrder(wurzel);
}
private void PostOrder(Knoten aktuell) {
  if (aktuell != null) {
       PostOrder(aktuell.GibLinks());
       PostOrder(aktuell.GibRechts());
       System.out.println(aktuell.GibWert());
  }
}
```

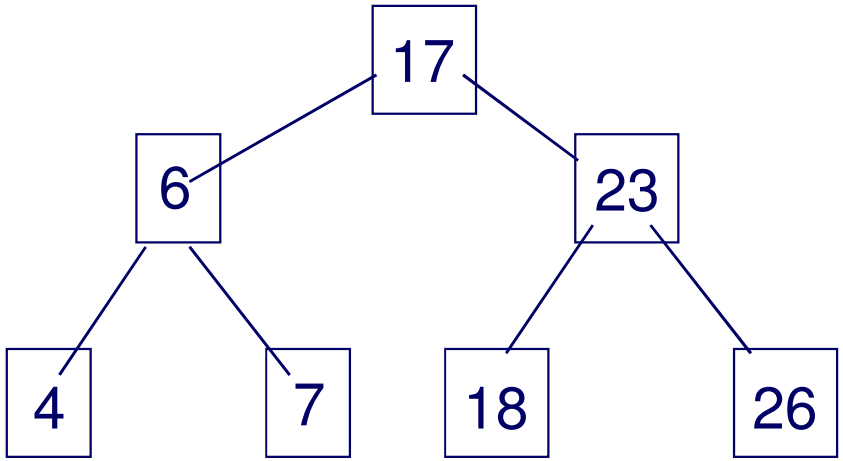


Reihenfolge der besuchten Knoten: 9, 13, 15, 14, 10, 18, 24, 16

Tiefendurchlauf / Postorder



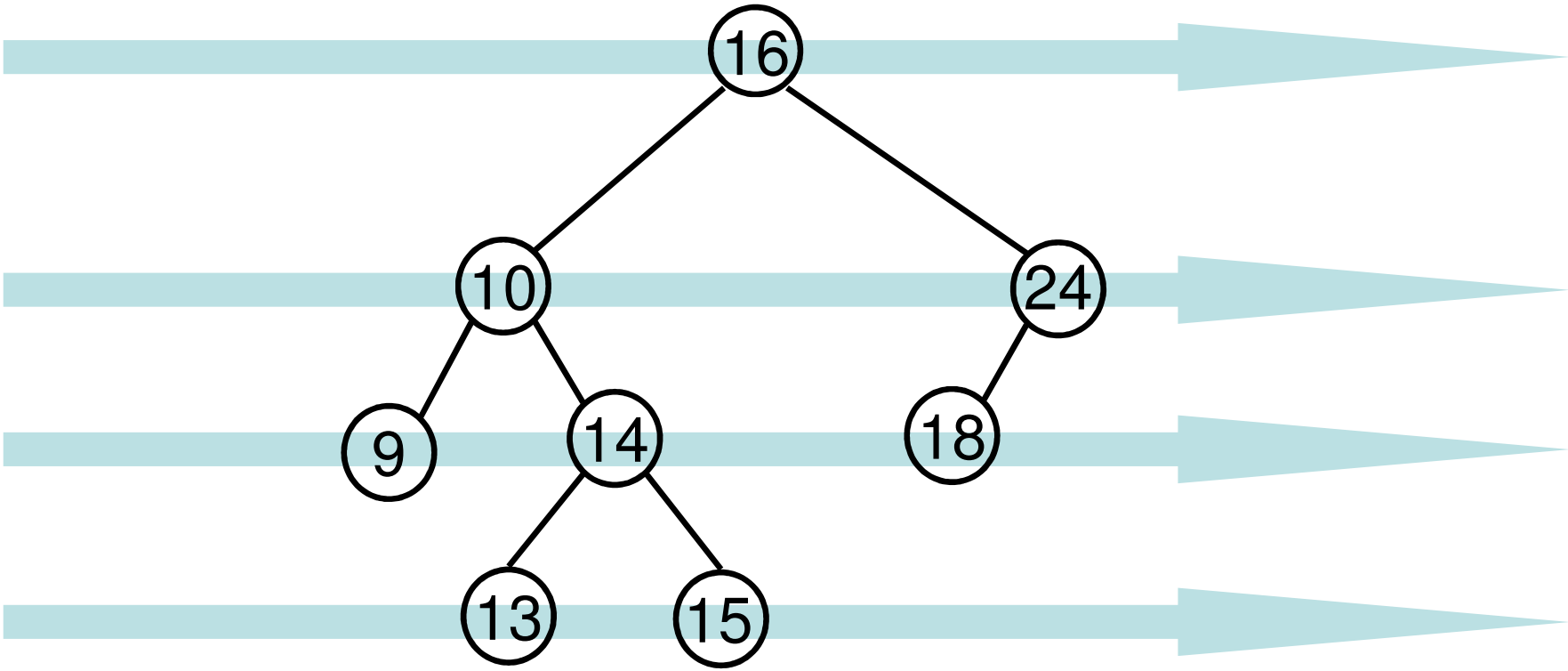




Anmerkungen zu den Tiefendurchläufen

Dr. Lars Hildebrand

Breitendurchlauf



Reihenfolge der besuchten Knoten: 16, 10, 24, 9, 14, 18, 13, 15

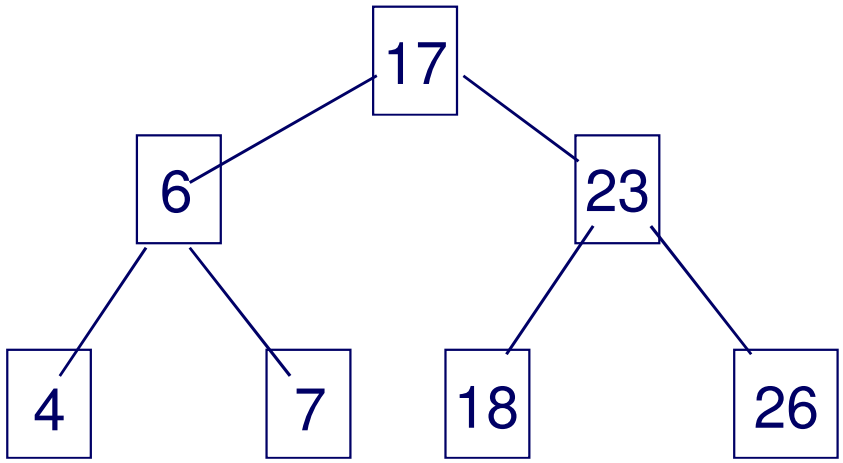
Breitendurchlauf

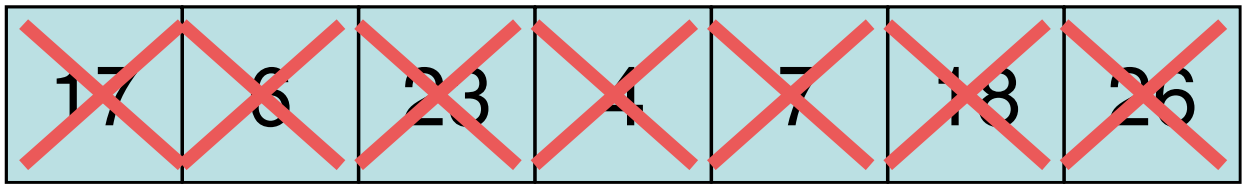
Idee zur Realisierung des Breitendurchlaufs

- Noch nicht besuchte Knoten in verketteter Liste zwischenspeichern;
- nächster Knoten steht am Listenanfang.
- Knoten wird besucht,
  - Knoten aus der Liste entfernen
  - linken und rechten Sohn (falls vorhanden), in dieser Reihenfolge ans Ende der Liste anfügen
- Dies geschieht solange, bis die Liste leer ist.
- Die Liste wird mit der Wurzel des Baumes initialisiert.

- Liste beschreibt eine *Warteschlange* für Knoten:
  - Der Knoten am Anfang der Warteschlange wird als nächster ausgedruckt.
  - Der Knoten am Ende der Warteschlange ist als letzter hinzugefügt worden.

Breitendurchlauf





Zusammenfassung

- Listen: ungünstig bzgl Suchaufwand O(N)
- Binäre Suchbäume
  - gerichtete, azyklische Graphen mit max 2 Nachfolgern je Knoten und max 1 Vorgänger je Knoten
  - Höhe des Baumes = max Länge einer Suche
    - degenerierter Baum: Suche in O(N)
    - balancierter Baum: Suche in $O(\log_2(N))$
  - Viele Varianten von Bäumen, um Suchaufwand und Aufwand für Einfüge/Entferne Operationen gering zu halten
    - AVL Bäume, ...
  - Operationen auf Bäumen
    - Einfügen
    - Löschen
    - Suchen
    - Traversieren: Inorder/Preorder/Postorder, Breitendurchlauf

Übersicht

Begriffe

- Spezifikationen, Algorithmen, formale Sprachen, Grammatik
- Programmiersprachenkonzepte
- Grundlagen der Programmierung

- Algorithmen und Datenstrukturen
  - Felder
  - Sortieren
  - Rekursive Datenstrukturen (Baum, binärer Baum, Heap)
  - Heapsort

- Objektorientierung
  - Einführung
  - Vererbung
  - Anwendung

technische universität dortmund

Dr. Lars Hildebrand

Viel Erfolg in der Klausur!

Nächste Termine

- 1. Klausur 29.2.2012
- 2. Klausur 29.3.2012
- 3. Klausur Februar 2013