EINI
LogWing/WiMa/MP

Einführung in die Informatik für Naturwissenschaftler und Ingenieure

Vorlesung 2 SWS WS 24/25

Dr. Lars Hildebrand
Fakultät für Informatik – Technische Universität Dortmund
lars.hildebrand@tu-dortmund.de
http://ls14-www.cs.tu-dortmund.de

Thema

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

- **Unterlagen**
  - Dißmann, Stefan und Ernst-Erich Doberkat*: Einführung in die objektorientierte Programmierung mit Java*, 2. Auflage. München [u.a.]: Oldenbourg, 2002. (→ ZB oder Volltext aus Uninetz)
  - Echtle, Klaus und Michael Goedicke: *Lehrbuch der Programmierung mit Java*. Heidelberg: dpunkt-Verl, 2000. (→ ZB)

In diesem Kapitel:

- **Prolog**
- Wiederholung
- Bäume

Allgemeines zu Bäumen I

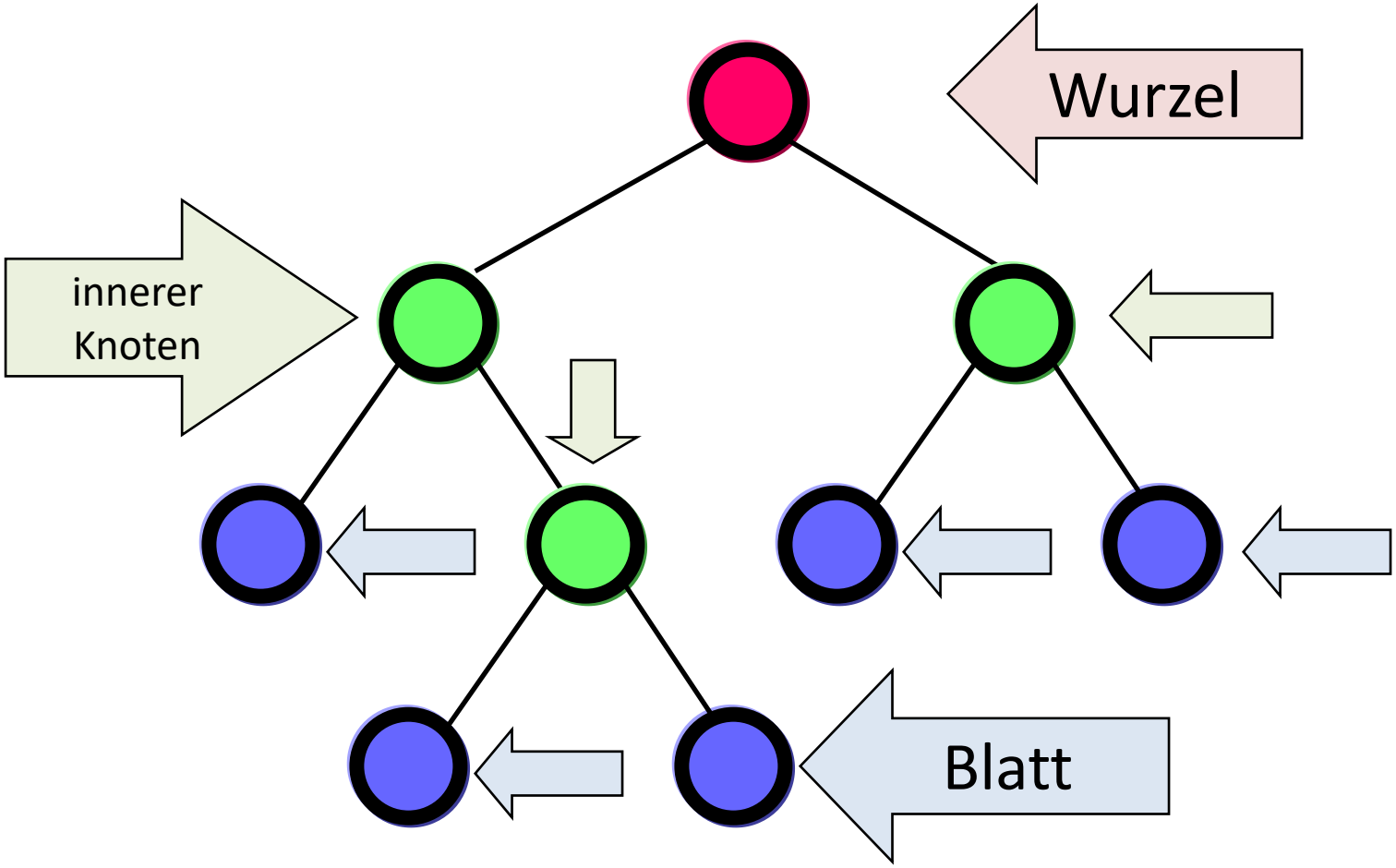
- **Bäume** sind…
  - gerichtete, azyklische Graphen. Es gibt keine Zyklen zwischen Mengen von Knoten.
  - hierarchische Strukturen. Man kommt von einer Wurzel zu inneren Knoten und letztlich zu Blättern.
  - verkettete Strukturen, die dynamisch wachsen und schrumpfen können.

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

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

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

Allgemeines zu Bäumen II



In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

Allgemeines zu Bäumen III

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

In diesem Kapitel:

- Prolog
- Wiederholung
- **Bäume**

Binäre Suchbäume

- **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

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

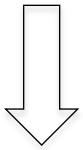
Suche „in einer Hälfte“ I

- **Grobe Idee (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“ II

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



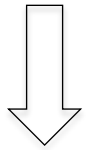
Position	1	2	3	4	5	6	7	8	9
Wert	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

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

In diesem Kapitel:
- Prolog
- Wiederholung
- Bäume

Suche „in einer Hälfte“ III

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



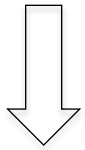
Position	1	2	3	4	5	6	7	8	9
Wert	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

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

In diesem Kapitel:
- Prolog
- Wiederholung
- Bäume

Suche „in einer Hälfte“ IV

- **Grobe Idee (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

In diesem Kapitel:
- Prolog
- Wiederholung
- Bäume

Suche „in einer Hälfte“ V

- **Grobe Idee (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., Wert = 4
  - 5 > 4
  - rechten Abschnitt wählen

In diesem Kapitel:
- Prolog
- Wiederholung
- Bäume

Suche „in einer Hälfte“ VI

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

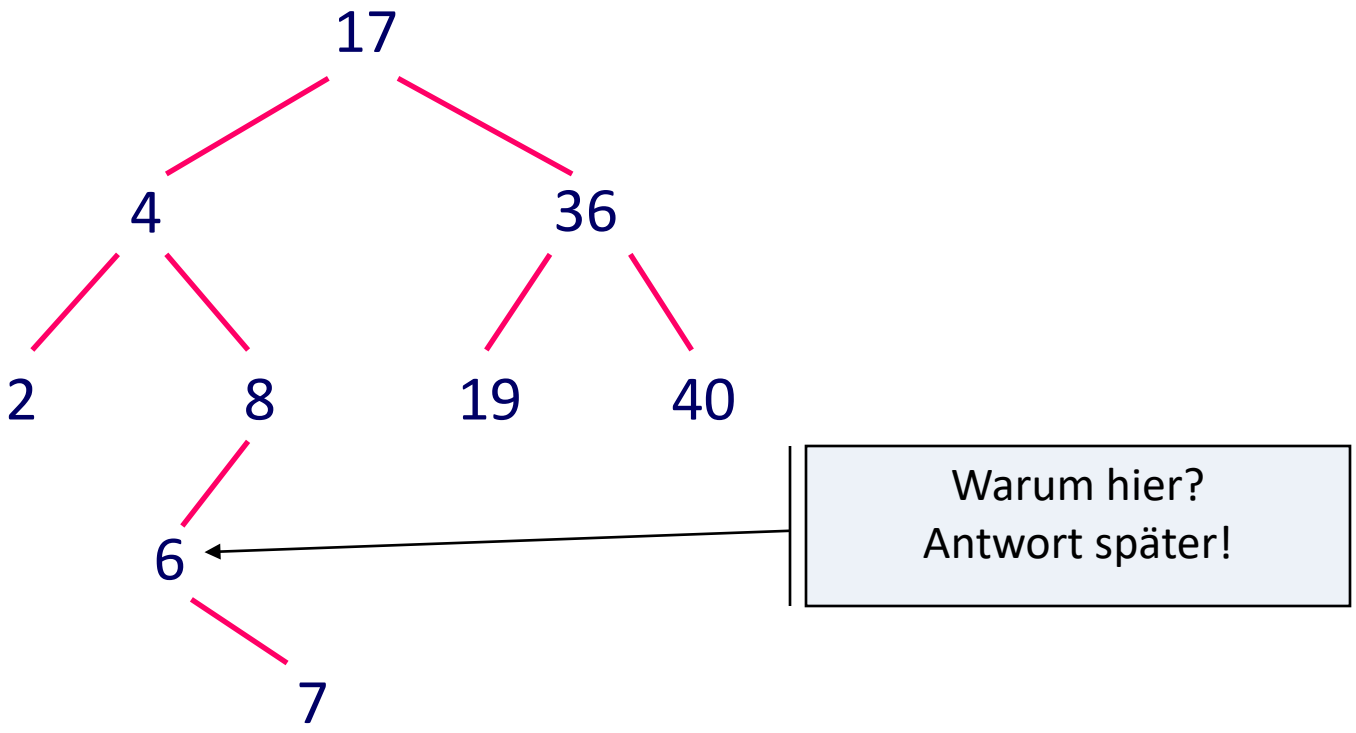
Position	1	2	3	4	5	6	7	8	9
Wert	**2**	**4**	**6**	**7**	**8**	**17**	**19**	**36**	**40**

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

In diesem Kapitel:
- Prolog
- Wiederholung
- Bäume

Suche „in einer Hälfte“

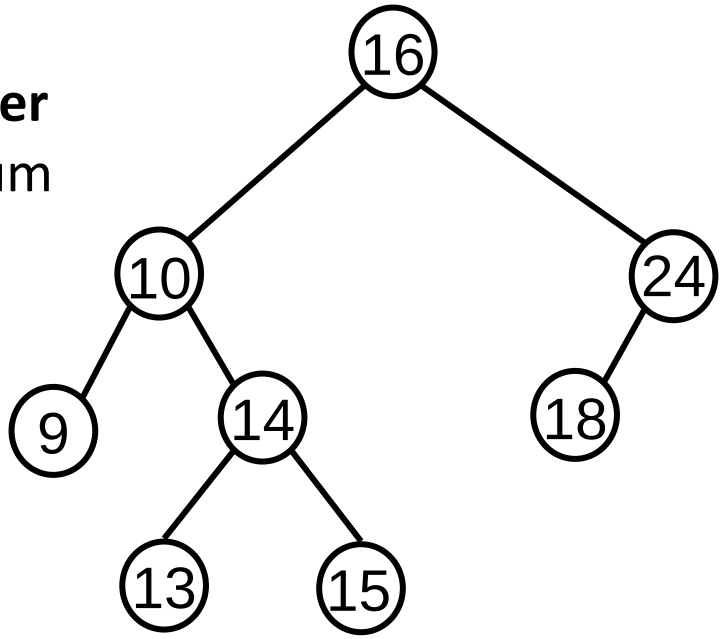
- **Aufgabe**: Trage die Zahlen 17, 4, 36, 2, 8, 19, 40, 6, 7 in eine baumförmige Struktur so ein,
  - dass die Suche „in einer Hälfte“ effektiv unterstützt wird:



Binäre Suchbäume I

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*.

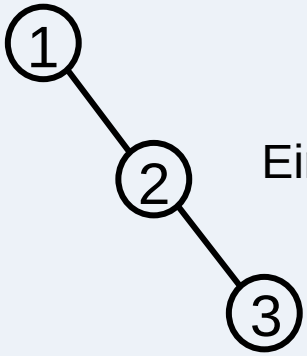


Binäre Suchbäume II

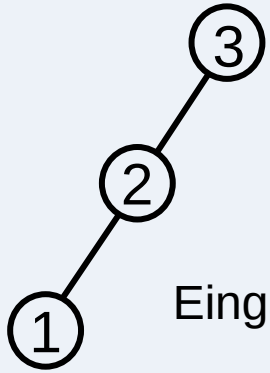
- Der **Aufbau** eines binären Suchbaums erfolgt durch **wiederholtes Einfügen** in einen (anfangs) leeren Baum.
- Die **Reihenfolge** der Werte, die in einen binären Suchbaum eingefügt werden, bestimmt die Gestalt des Baumes.
- Eine Menge von Werten kann bei unterschiedlichen Eingabereihenfolgen zu **verschiedenen Repräsentationen** als Baum führen.

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

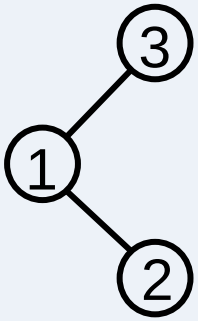
Binäre Suchbäume – Beispiele



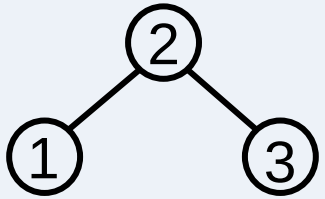
Eingabefolge 1 2 3



Eingabefolge 3 2 1



Eingabefolge 3 1 2



Eingabefolge 2 1 3 oder 2 3 1

Binäre Suchbäume

Algorithmus für die Suche von Knoten

- In einem binären Suchbaum lässt sich effizient suchen:

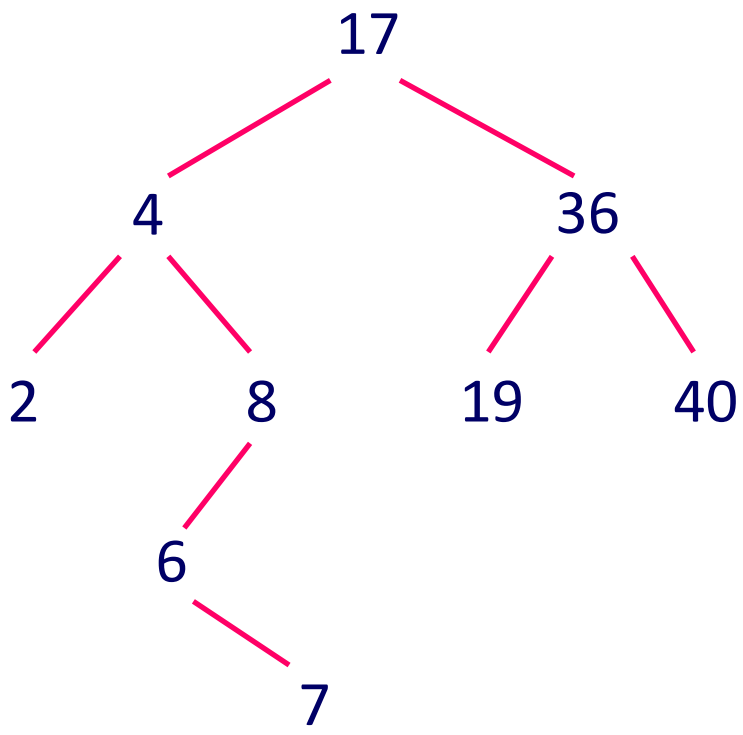
  Gegeben sind ein binärer Suchbaum *B* und eine Zahl *k*, die in dem Baum *B* gesucht werden soll:

  - *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*.

In diesem Kapitel:
- Prolog
- Wiederholung
- Bäume

Suche „in einer Hälfte“

- **Aufgabe**: Wir suchen die Zahlen 5 und 6.



Suchen in binären Suchbäumen I

Definition:

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

$$h(B) := \begin{cases} 0, & \text{falls B leer ist} \\ 1 + max\{h(B1), h(B2)\}, & \text{falls B1 und B2 linker bzw. rechter Unterbaum von B sind} \end{cases}$$

- 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.

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

Suche in binären Suchbäumen II

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.

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

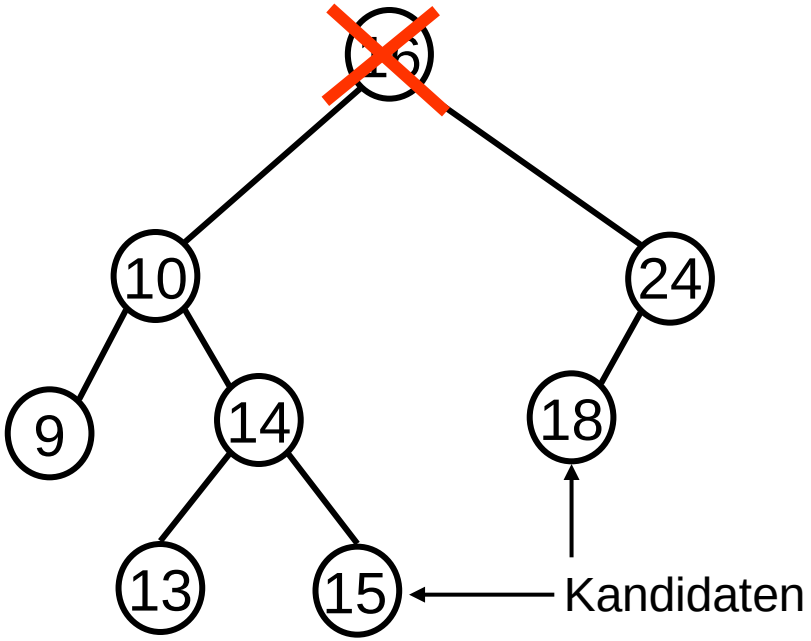
Entfernen der Wurzel aus einem binären Suchbaum

Algorithmus für das Entfernen

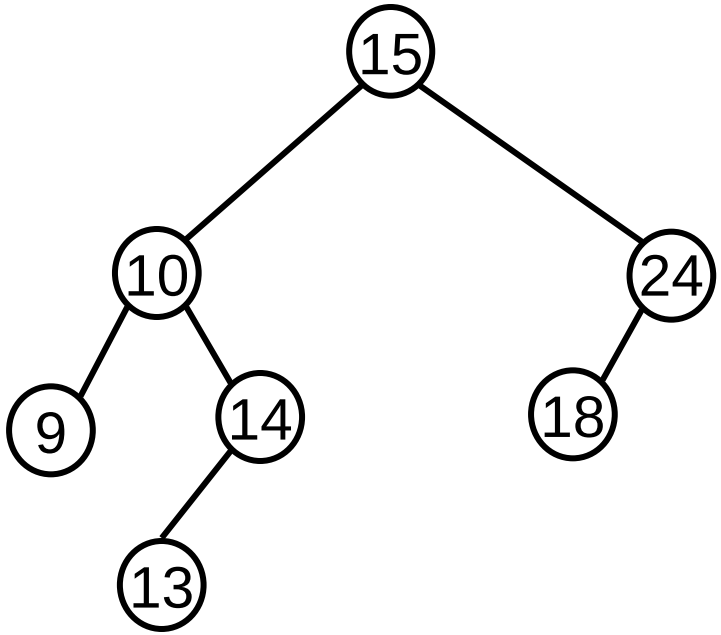
- Entfernen der Wurzel führt zur Konstruktion eines neuen binären Suchbaums.
- Darum: Finden eines Knotens, der an die Stelle der Wurzel gesetzt wird und die Kriterien für einen neuen binären Suchbaum erfüllt
- Der Knoten muss größer als die Wurzel des linken Unterbaumes sein und kleiner als die Wurzel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

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

Entfernen der Wurzel – Beispiel



Situation vor dem Löschen



Situation nach dem Löschen

Entfernen der Wurzel aus einem binären Suchbaum

Algorithmus für das Entfernen

- Der Knoten mit der größten Beschriftung im linken Unterbaum wird 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. Diese Wurzel wird dann entfernt.

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

Durchlaufstrategien für binäre Suchbäume

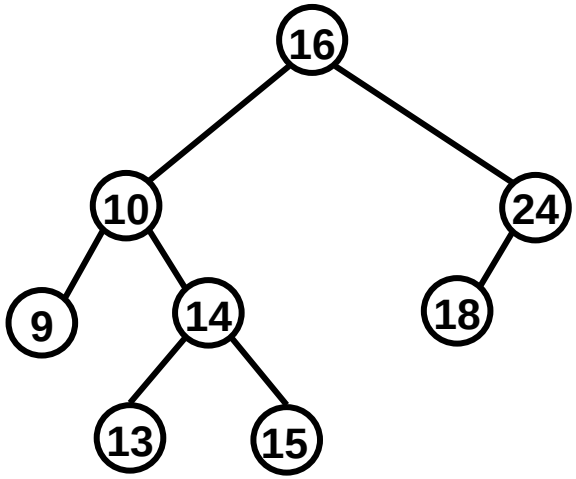
- **Tiefendurchlauf**: Hierbei wird von einem Knoten aus in die Tiefe gegangen, indem einer der Söhne besucht wird und dann dessen Söhne usw. Erst wenn man die Blätter erreicht hat, beginnt der Wiederaufstieg.
  - Preorder-Durchlauf
  - Inorder-Durchlauf
  - Postorder-Durchlauf

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

In diesem Kapitel:
- Prolog
- Wiederholung
- **Bäume**

Tiefendurchlauf / Preorder

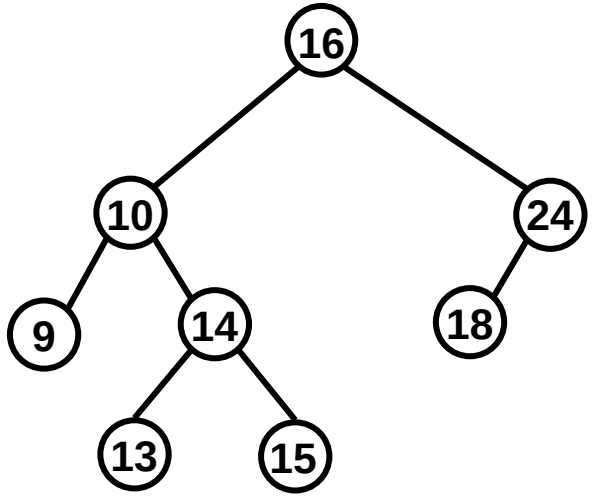
```
01 void PreOrder() {
02   PreOrder(wurzel);
03 }
04 private void PreOrder(Knoten aktuell) {
05  if (aktuell != null) {
06     System.out.println(aktuell.GibWert());
07     PreOrder(aktuell.GibLinks());
08     PreOrder(aktuell.GibRechts());
09  }
10 }
```



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

Tiefendurchlauf / Inorder

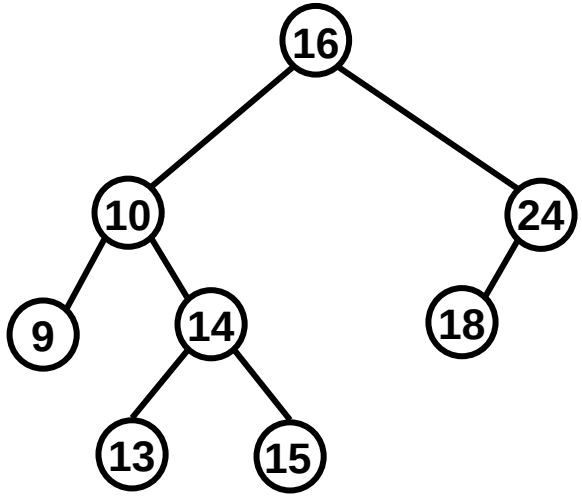
```
01 void InOrder() {
02   InOrder(wurzel);
03 }
04 private void InOrder(Knoten aktuell) {
05   if (aktuell != null) {
06     InOrder(aktuell.GibLinks());
07     System.out.println(aktuell.GibWert());
08     InOrder(aktuell.GibRechts());
09   }
10 }
```



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

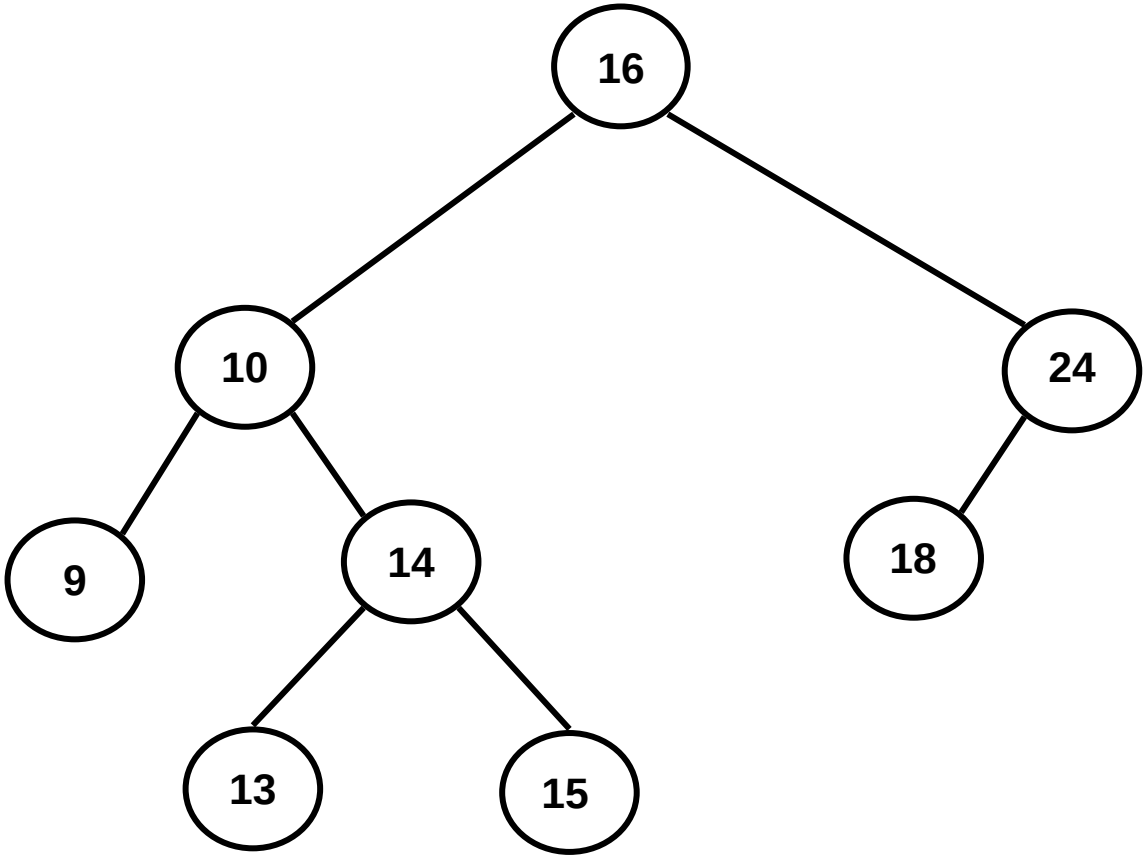
Tiefendurchlauf / Postorder

```
01 void PostOrder() {
02     PostOrder(wurzel);
03 }
04 private void PostOrder(Knoten aktuell) {
05     if (aktuell != null) {
06         PostOrder(aktuell.GibLlinks());
07         PostOrder(aktuell.GibRechts());
08         System.out.println(aktuell.GibWert());
09     }
10 }
```

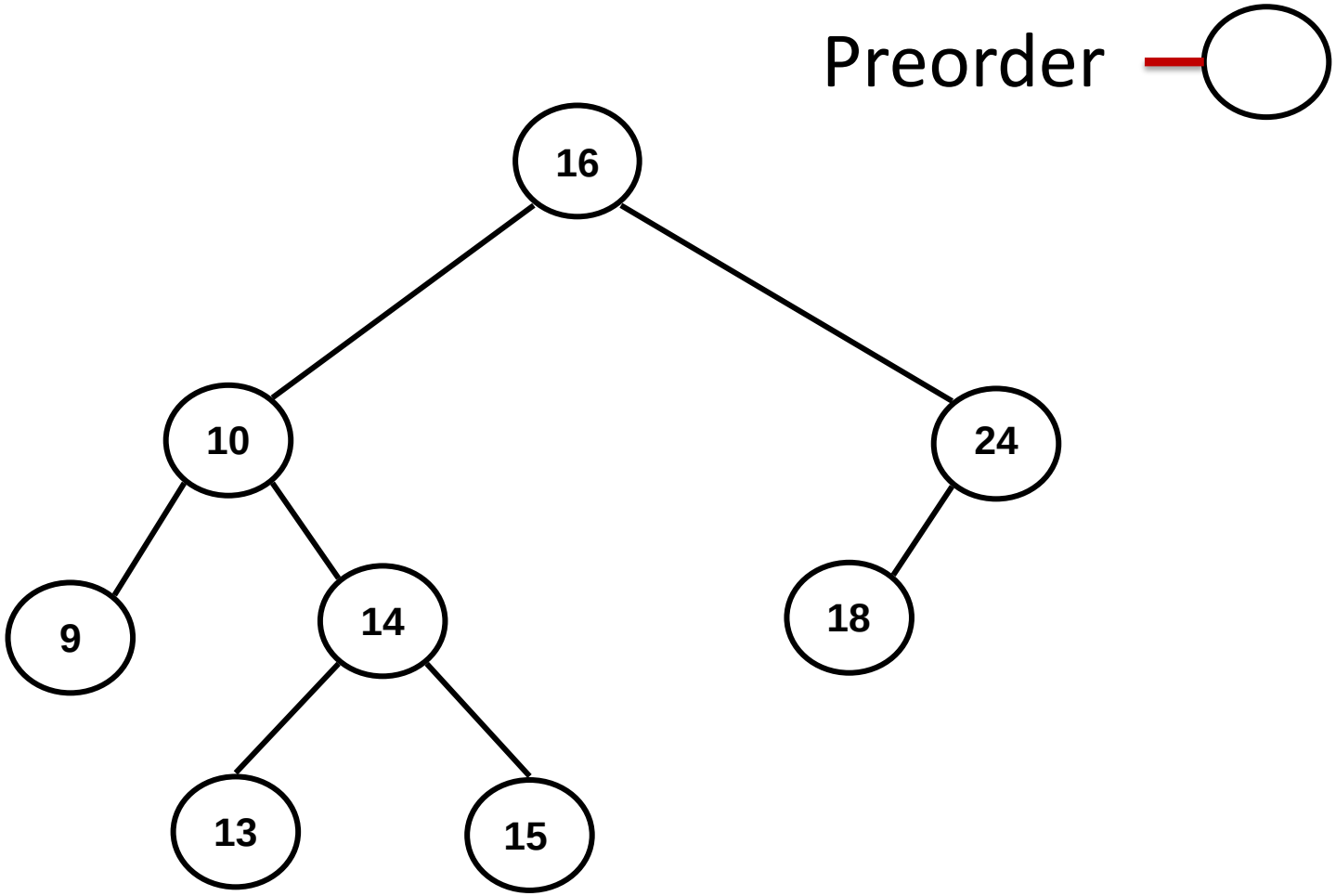


Reihenfolge der besuchten Knoten: 9, 13, 15, 14, 10, 18, 24, 16

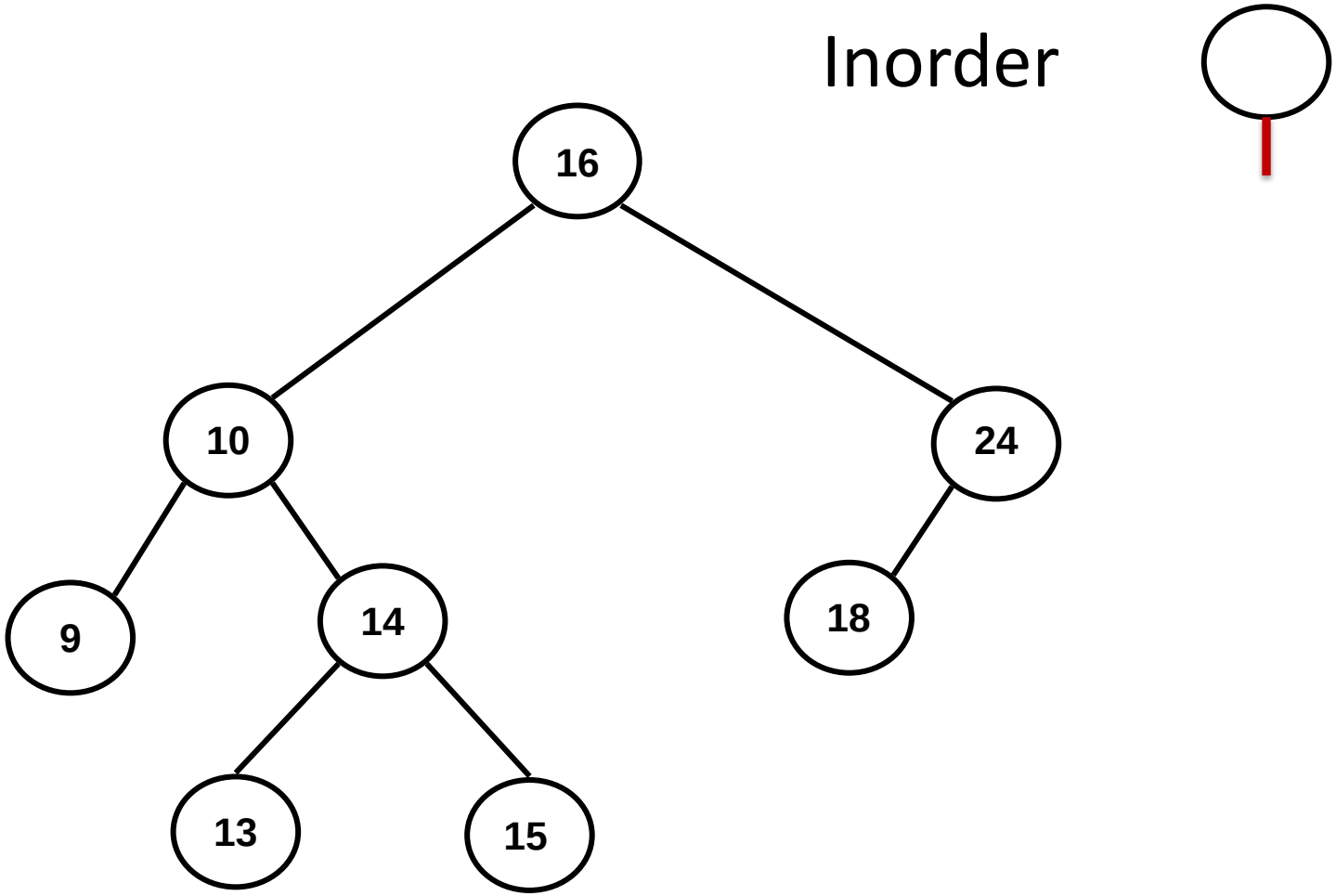
Anmerkungen zu den Tiefendurchläufen I



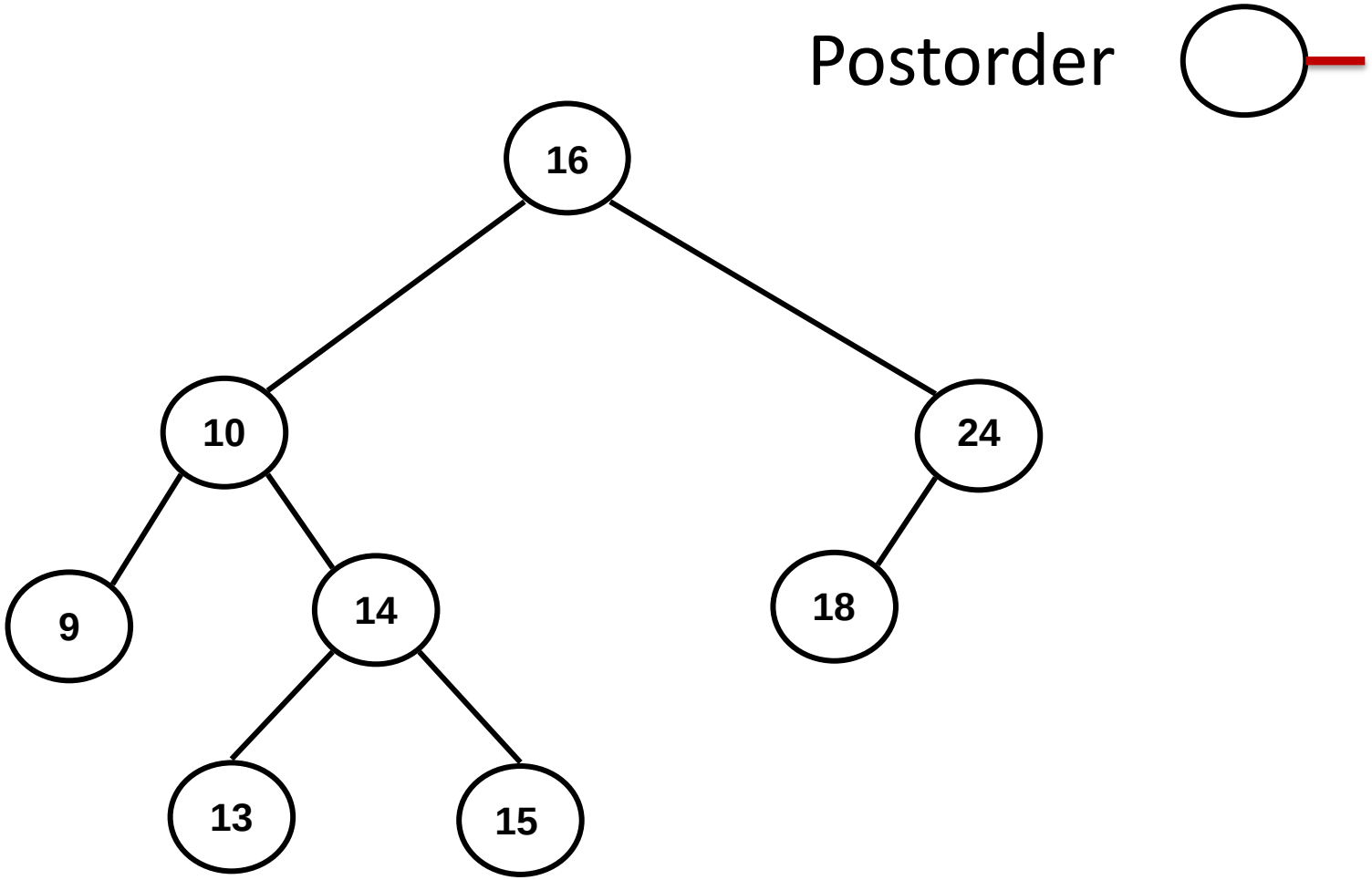
Anmerkungen zu den Tiefendurchläufen II



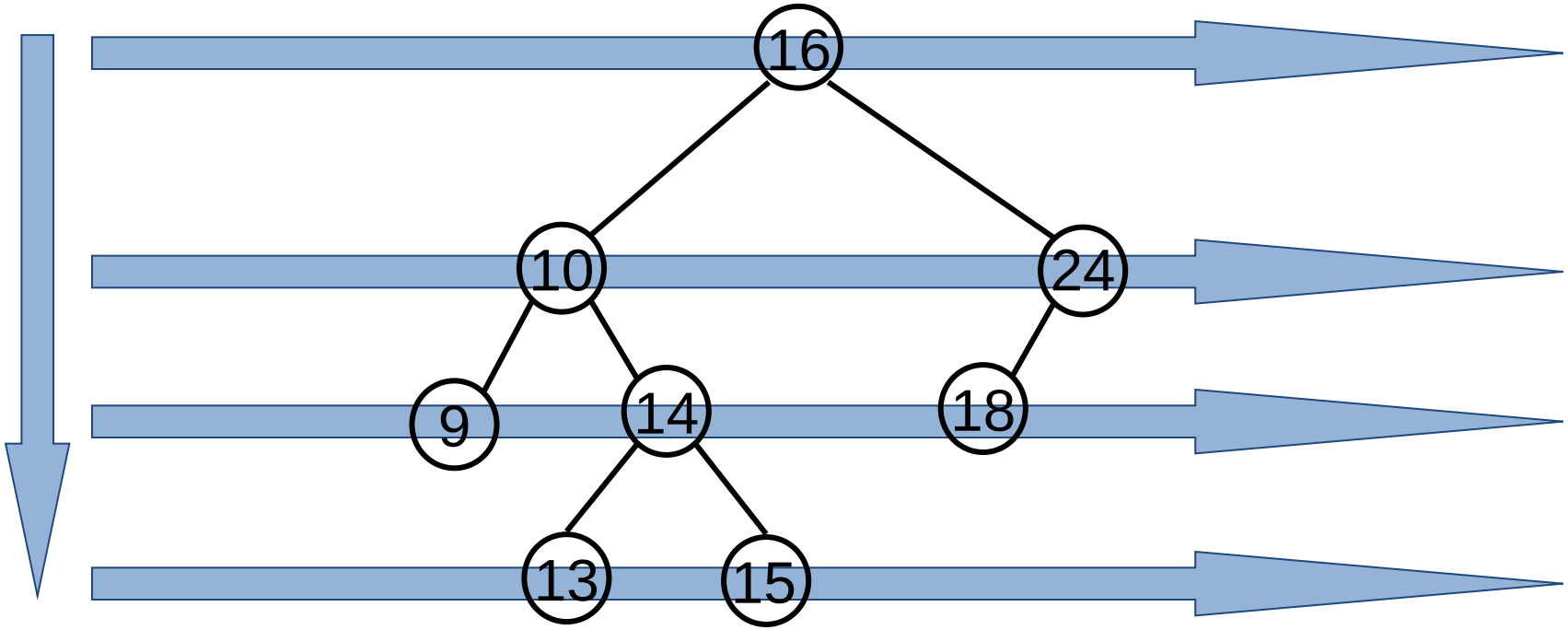
Anmerkungen zu den Tiefendurchläufen III



Anmerkungen zu den Tiefendurchläufen IV



Breitendurchlauf I



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

Zusammenfassung

- 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- und Entferne-Operationen gering zu halten:
    - AVL Bäume, ...
  - Operationen auf Bäumen:
    - Einfügen
    - Löschen
    - Suchen
    - Traversieren: Inorder/Preorder/Postorder, Breitendurchlauf

Übersicht

Begriffe

- ✓ Spezifikationen, Algorithmen, formale Sprachen
- ✓ Programmiersprachenkonzepte
- ✓ Grundlagen der imperativen Programmierung

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

- ➢ Objektorientierung
  - ✓ Einführung
  - ✓ Vererbung
  - ✓ Anwendung

Übersicht

Vielen Dank für Ihre Aufmerksamkeit!

Viel Erfolg bei der Klausur!

Nächste Termine

Klausur 1	6.2.2025
Klausur 2	21.3.2025

EINI 2025/26	Herbst 2025