Kap. 4.2: Binäre Suchbäume

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

Fakultät für Informatik, TU Dortmund

11. VO DAP2 SS 2009 26. Mai 2009

Zusätzliche Lernraumbetreuung

- Morteza Monemizadeh: Jeden Montag von 14:00 Uhr-16:00 Uhr in OH14, R. 314, Hilfe bei Problemen der Vorlesung oder Übung (in englischer Sprache)

- Beachten Sie auch die bisherigen Lernraumangebote Mo-Fr (s. Web)

Motivation

„Warum soll ich heute hier bleiben?"

Binäre Suchbäume begegnen Ihnen ständig!

„Warum soll mich das interessieren?"

Beliebte Klausuraufgaben!

Binäre Suchbäume

- Datenstruktur zur Unterstützung der Operationen des ADT Dictionary:

Operationen des ADT Dictionary

- **INSERT(K k,V v)**
  - Falls k nicht schon in D ist, dann wird ein neuer Schlüssel k mit Wert v in D eingefügt, andernfalls wird der Wert des Schlüssels k auf v geändert.

- **DELETE(K k)**
  - Entfernt Schlüssel k mit Wert aus D (falls k in D)

- **SEARCH(K k): V**
  - Gibt den bei Schlüssel k gespeicherten Wert zurück (falls k in D)

Zusätzliche Operationen der Binären Suchbäume

- **MINIMUM (MAXIMUM)**
  - Sucht den Schlüssel mit kleinstem (größtem) Wert.

- **PREDECESSOR (SUCCESSOR)**
  - Ausgehend von einem gegebenen Element suchen wir das Element mit dem nächstkleineren (nächstgrößeren) Schlüssel.

Überblick

- Bezeichnungen
- Traversierungen für binäre Bäume
- Implementierungen
- Analyse

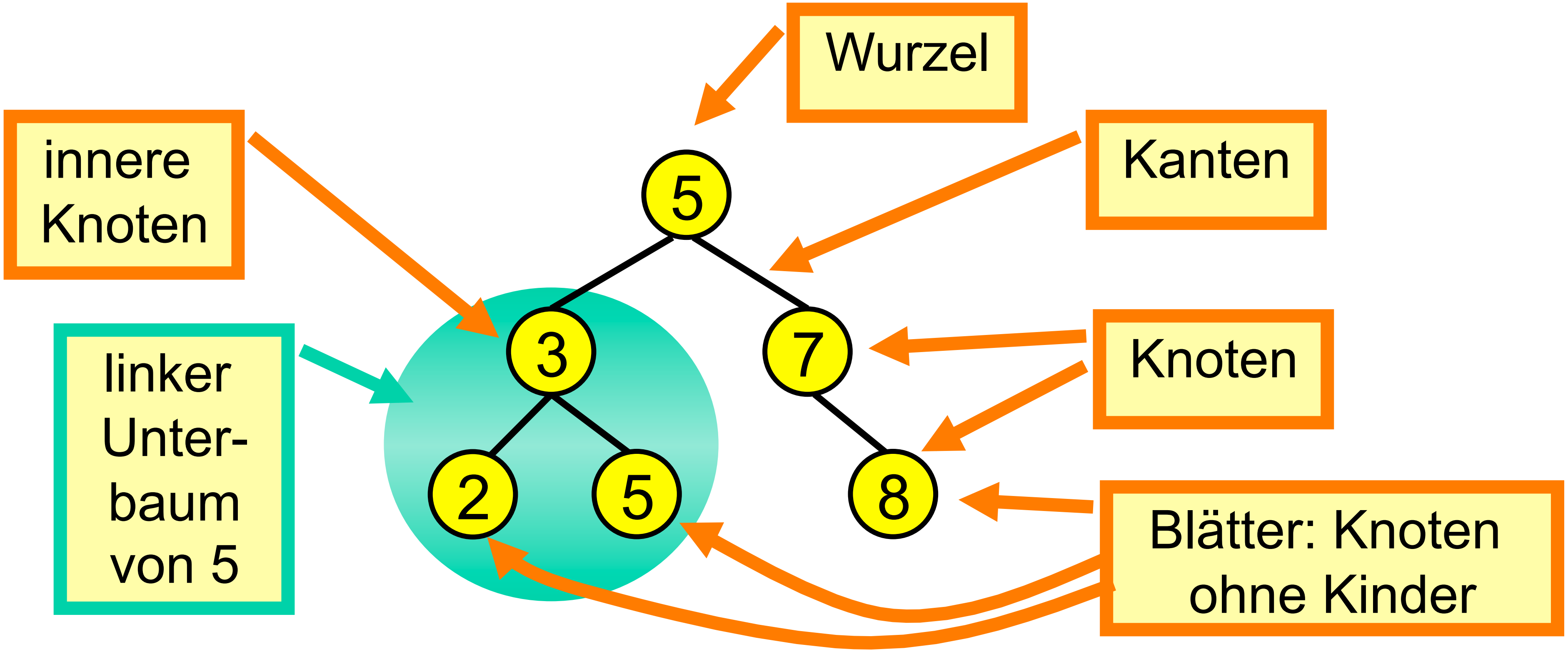
Definitionen

- **Gewurzelter Baum** (rekursiv):
  - entweder leer oder
  - Knoten (Wurzel) mit Verweisen auf mehrere gewurzelte Bäume (Teilbäume)

- **Kind von Knoten v:** Wurzelknoten eines nicht-leeren Teilbaums von v

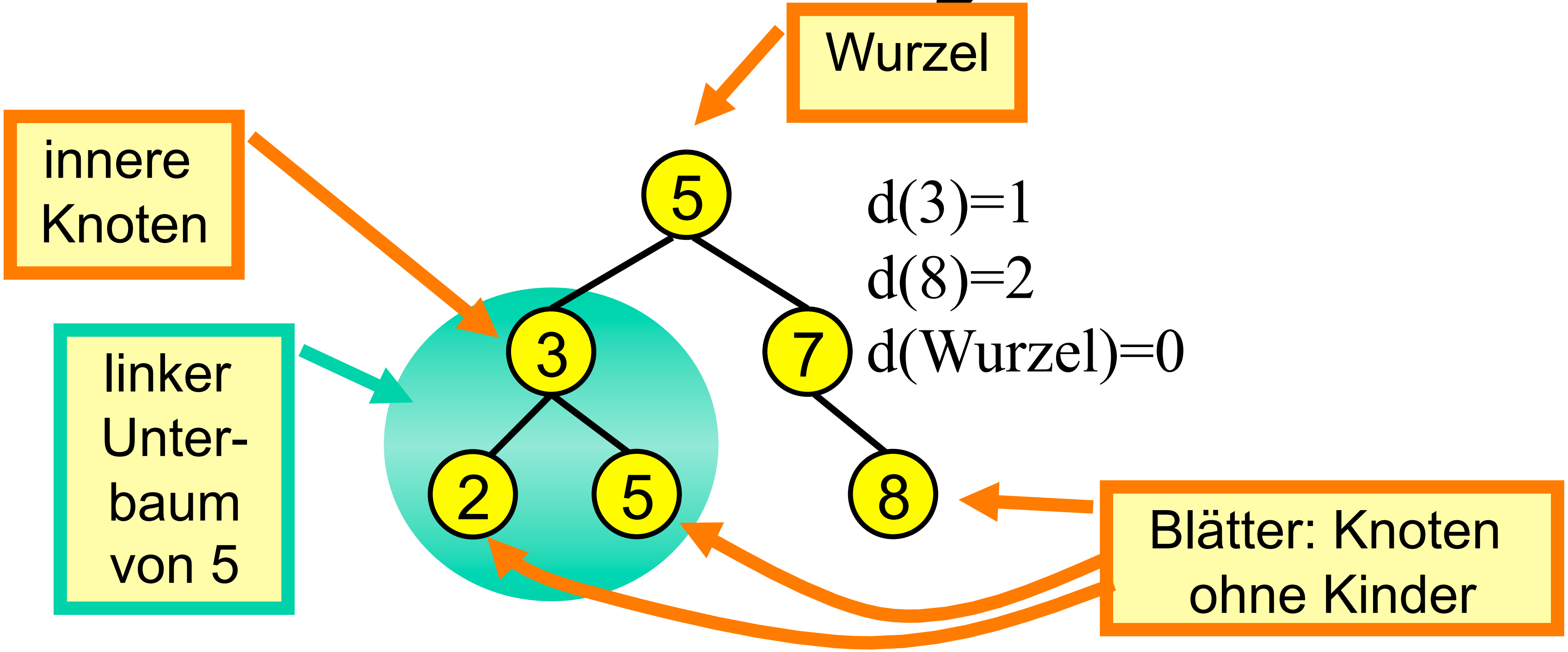
- **Blatt:** Knoten ohne Kinder

Bezeichnungen: Gewurzelte Bäume



- Tiefe $d(v)$: # der Elter-Knoten bis Wurzel (inkl.)
- 3 ist Wurzel des linken Unterbaums von 5
- 3 ist Elter von 2, 3 ist Elter von 5
- 2 ist linkes Kind von 3, 5 ist rechtes Kind von 3

Bezeichnungen



- Tiefe $d(v)$: # der Elter-Knoten bis Wurzel (inkl.)

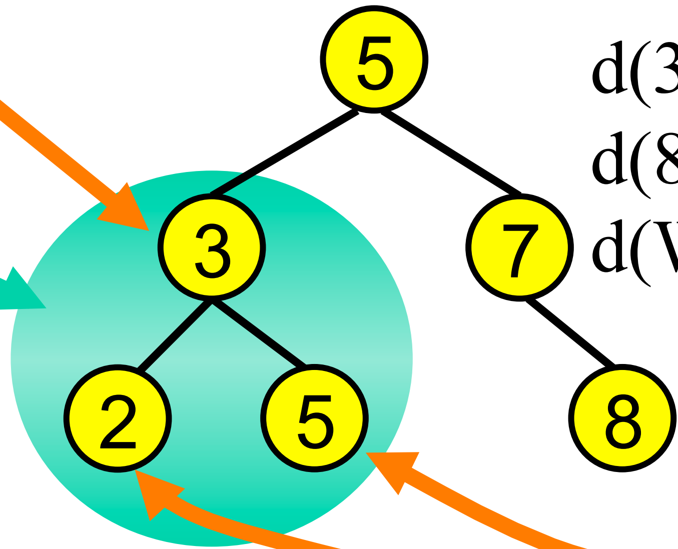
Ebene: Menge aller Knoten mit gleicher Tiefe

Bezeichnungen

Wurzel

innere Knoten

linker Unterbaum von 5



$d(3)=1$
$d(8)=2$
$d(\text{Wurzel})=0$

$h(T_3)=1$
$h(T_8)=0$
$h(T_{\text{Wurzel}})=2$

Blätter: Knoten ohne Kinder

Nachfolger(v): Menge aller Knoten im Unterbaum mit Wurzel v

Höhe $h(T_r)$ eines (Teil-)baumes T_r mit Wurzel r:
max $\{ d(v)$: v ist Knoten in $T_r \}$

Weitere Definitionen

- Ein gewurzelter Baum heißt **geordnet**, wenn die Reihenfolge der Kinder festgelegt ist
- Ein **binärer** gewurzelter Baum ist ein gewurzelter Baum, bei dem jeder Knoten genau zwei Kinder hat.
- Ein **geordneter** gewurzelter binärer Baum unterscheidet also zwischen **linkem und rechtem** Kind bzw. linkem und rechtem Unterbaum.
- Ein binärer Baum heißt **vollständig**, wenn alle Blätter die gleiche Tiefe haben.

Def. Binärer Suchbaum

- Ein binärer Suchbaum ist ein binärer geordneter Baum, dessen Knoten einen Suchschlüssel als Datum enthalten, und der die **Suchbaumeigenschaft** erfüllt:

- **Suchbaumeigenschaft**: Enthält ein Knoten den Schlüssel x, so sind alle Schlüssel in seinem linken Unterbaum kleiner als x und alle Schlüssel im rechten Unterbaum größer als x.

Achtung: alle Schlüssel verschieden

Implementierung binärer Bäume

- Realisierung als verallgemeinerte Listen mit bis zu zwei Nachfolgern:
  - **x.key**: Schlüssel von Knoten x
  - **x.info**: zum Schlüssel zu speichernde Daten
  - **x.parent**: Elter von Knoten x
  - **x.left**: linkes Kind von Knoten x
  - **x.right**: rechtes Kind von Knoten x

- Zugriff auf den Baum erfolgt über seinen Wurzelknoten root

Traversierungen für binäre Bäume

- **Inorder-Traversierung:** Durchsuche rekursiv zunächst den linken Unterbaum, dann die Wurzel, durchsuche dann den rechten Unterbaum.
- **Preorder-Traversierung:** Besuche zuerst die Wurzel, durchsuche dann rekursiv den linken Unterbaum, dann den rechten.
- **Postorder-Traversierung:** Durchsuche rekursiv zunächst den linken Unterbaum, durchsuche dann rekursiv den rechten Unterbaum, besuche dann die Wurzel.

Traversierungen für binäre Bäume

- **Level-Order-Traversierung:** Durchsuche zunächst alle Ebenen von links nach rechts, wobei die Ebenen von oben nach unten besucht werden.

- Die wichtigste Traversierungsordnung für binäre Suchbäume ist die **Inorder-Traversierung,** weil dabei die Schlüssel aufsteigend sortiert durchlaufen werden.

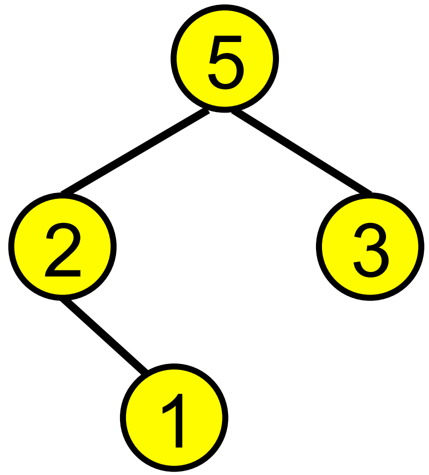
Inorder-Traversierung

```
Procedure INORDER(p)
if p ≠ NULL {
    INORDER(p.left)
    Ausgabe von p
    INORDER(p.right)
}
```

Aufruf: INORDER(root)

Beispiel für INORDER-Aufruf

- INORDER(5)
- INORDER(2)
- Ausgabe von „2“
- INORDER(1)
- Ausgabe von „1“
- Ausgabe von „5“
- INORDER(3)
- Ausgabe von „3“



Inorder: 2,1,5,3

s. auch Skript

Preorder-Traversierung

```
Procedure PREORDER(p)
if p ≠ NULL {
      INORDER(p.left)
      PREORDER(p.left)
      PREORDER(p.right)
}
```

Postorder-Traversierung

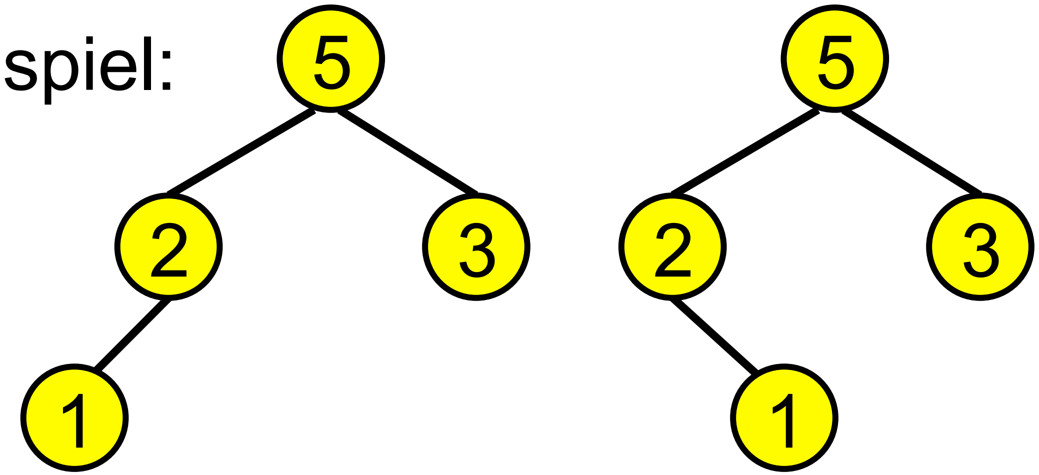
```
(1) Procedure POSTORDER(p)
(2) if p ≠ NULL {
(3)       POSTORDER(p.left)
(4)       POSTORDER(p.right)
(5)       INORDER(p.right)
(6) }
```

Traversierungen für binäre Bäume

- Aus **Inorder-Traversierung und Preorder-Traversierungsreihenfolge** kann der binäre Baum eindeutig rekonstruiert werden, wenn er lauter verschiedene Schlüssel enthält.

- Dies gilt nicht, wenn lediglich Preorder- und Postorder-Reihenfolge gegeben sind.

Beispiel:



Hier geht es nicht um binäre SUCHBÄUME

Preorder: 5,2,1,3
Postorder: 1,2,3,5

s. Übung

Implementierung von SEARCH(r,s) in binären Suchbäumen

- Finde die Position im Baum mit Wurzel r, an der s gespeichert ist (bzw. sein müßte):
- Idee: nutze Suchbaumeigenschaft:

1. Vergleiche s mit dem Schlüssel s' an der Wurzel (des Teilbaums)
2. Falls gefunden: STOP!
3. Sonst: Falls $s < s'$: suche im linken Teilbaum
4. Sonst: suche im rechten Teilbaum
5. Gehe zu 1.
6. Ausgabe: nicht gefunden!

Pseudocode s. Skript bzw. Folien

Implementierung von INSERT(r,q) in binären Suchbäumen

- Einfügen eines Knotens q in den Baum mit Wurzel r

1. Suche nach $q.key \rightarrow$ Suche endet mit der Position eines leeren Unterbaums
2. Einfügen von q an diese Position

Pseudocode s. Skript bzw. Folien

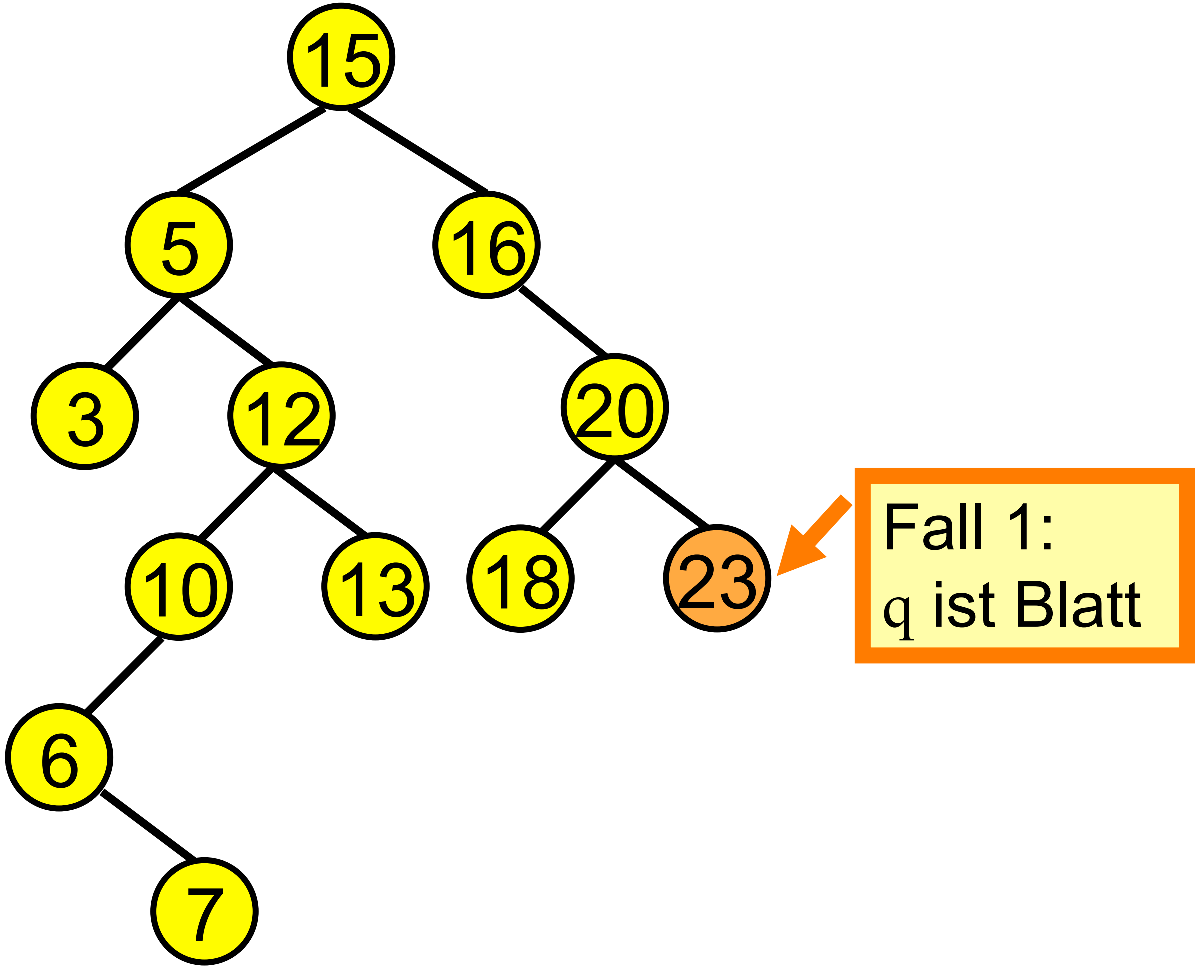
Implementierung von DELETE(r,q) in binären Suchbäumen

- Entfernt Knoten q im Baum mit Wurzel r

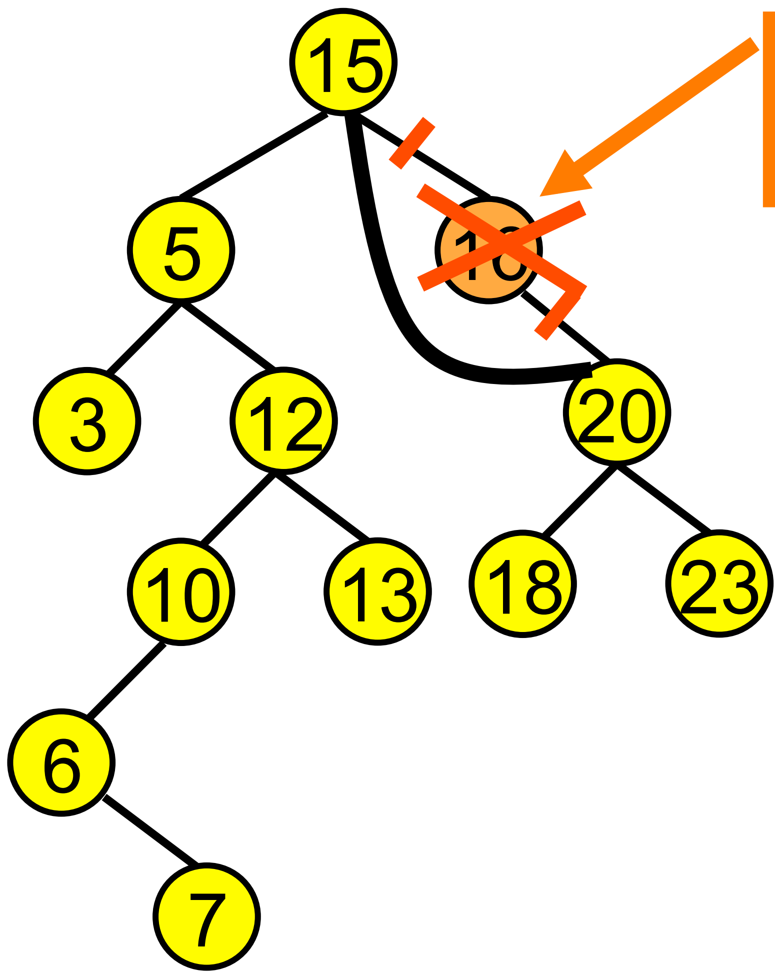
1. Suche nach $q.key$ → Suche endet an Knoten v
2. Falls $v \neq NULL$, dann
3. Falls v **keine Kinder** besitzt, dann: streiche v
4. Falls v **genau ein Kind** hat: ändere 2 Zeiger („herausschneiden“)
5. Falls v **zwei Kinder** hat, dann hat y:=Successor(v) kein linkes Kind (warum??): Herausschneiden bzw. Entfernen von y und Ersetzen von q durch y.

Pseudocode s. Skript bzw. Folien

Beispiel für DELETE(r,q)

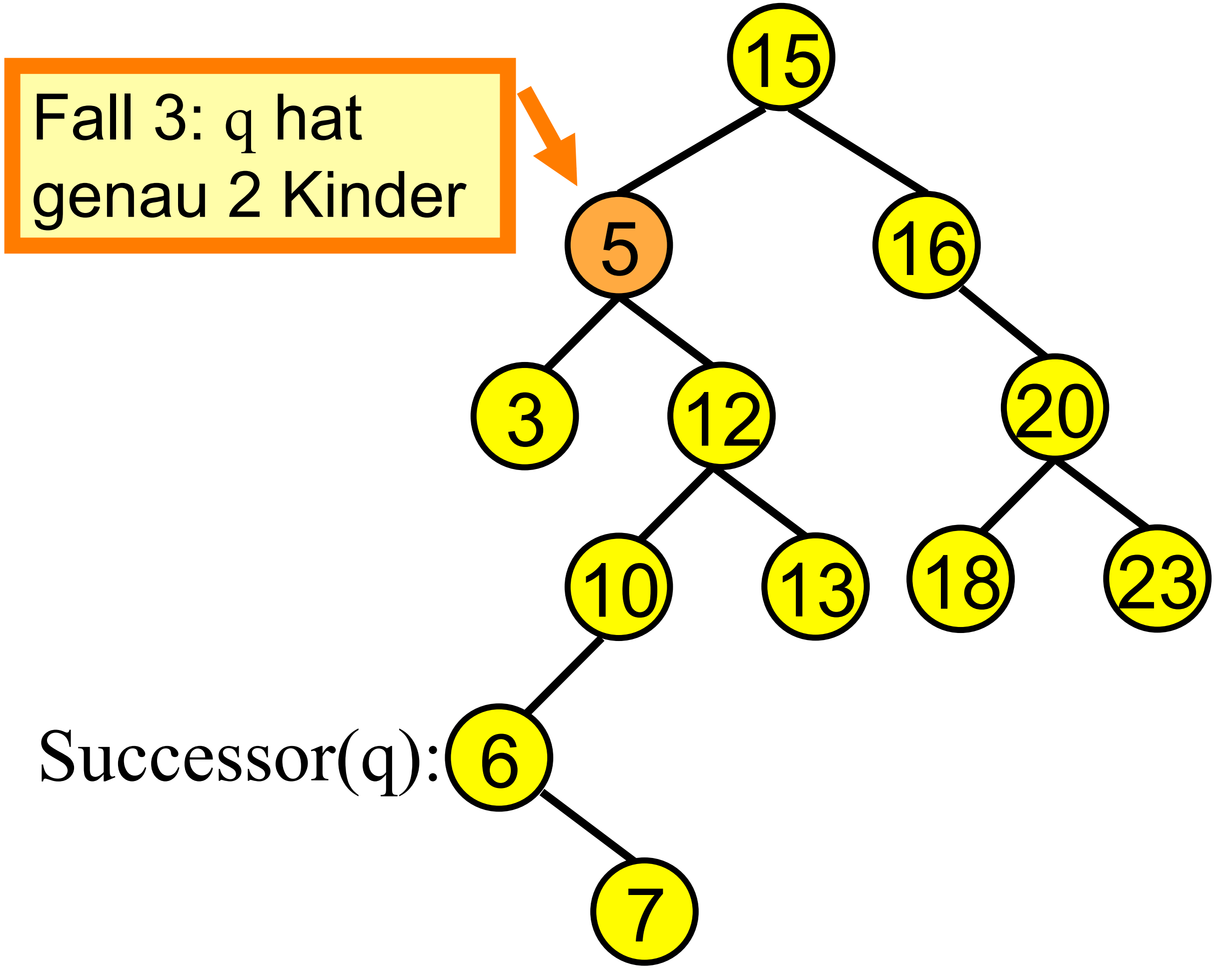


Beispiel für DELETE(r,q)

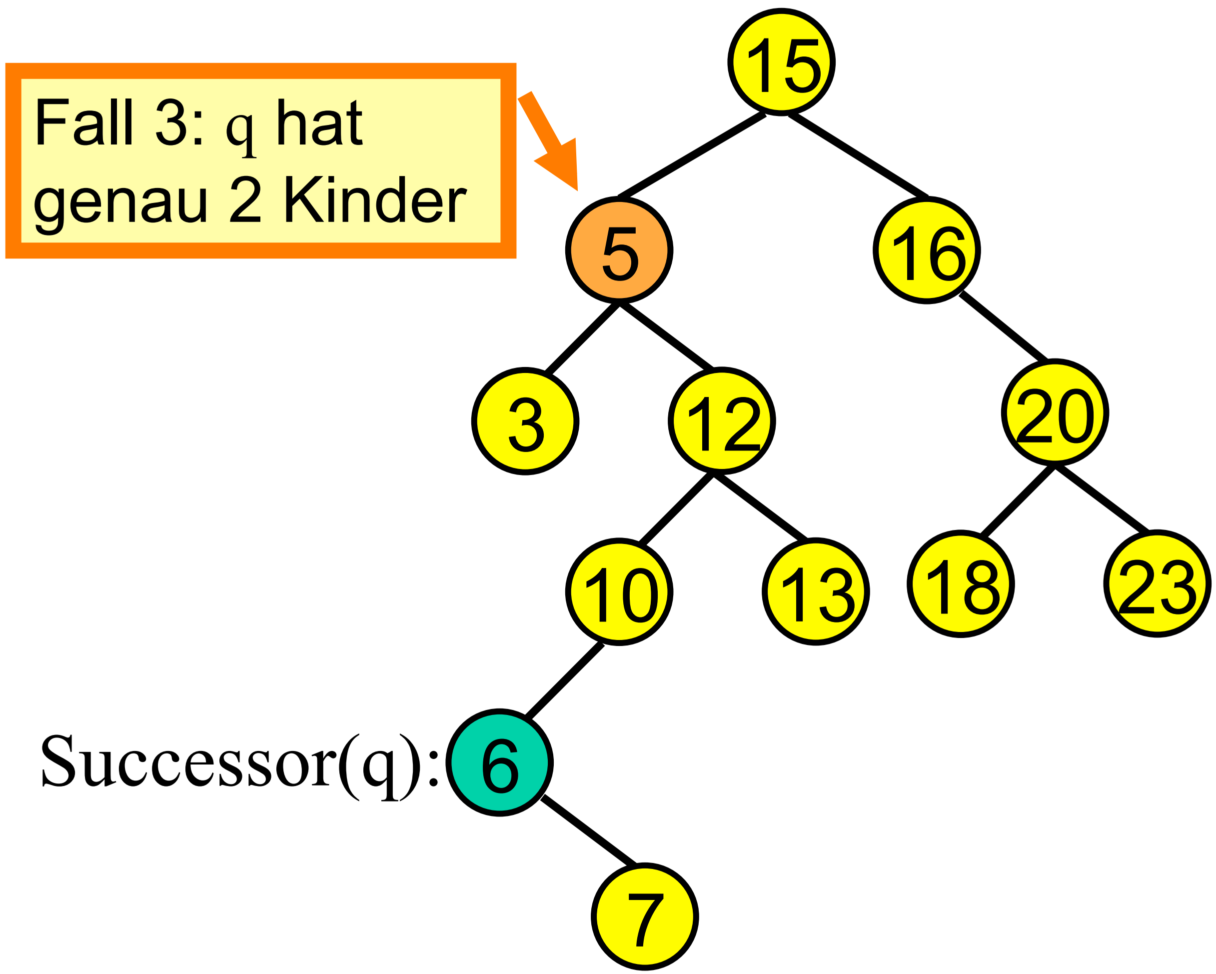


Fall 2: q hat genau ein Kind

Beispiel für DELETE(r,q)

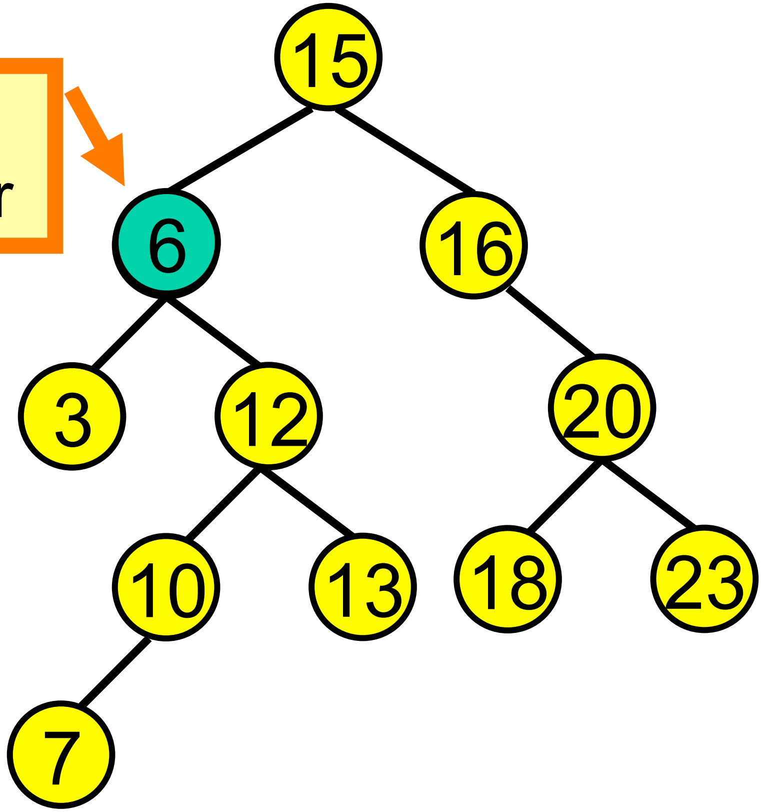


Beispiel für DELETE(r,q)

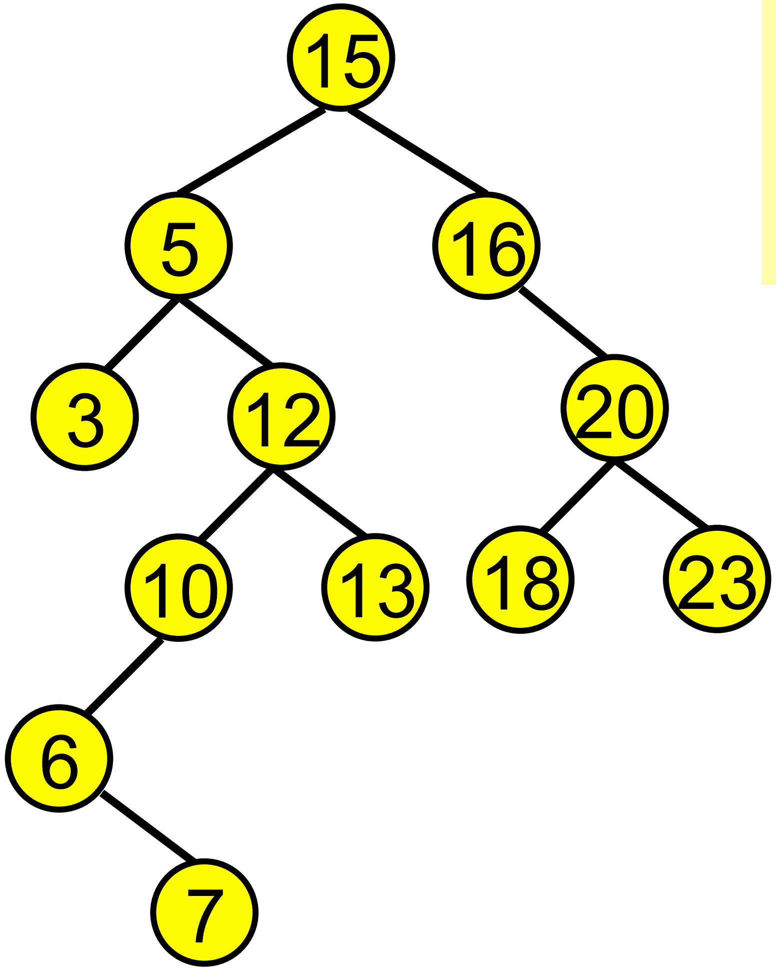


Beispiel für DELETE(r,q) ff

Fall 3: q hat genau 2 Kinder



Successor-Suche



Successor(5) = 6
Successor(12) =13
Successor(13) =15

Implementierung von SUCCESSOR(r,p)

- Gibt den Nachfolger von Knoten p in der Inorder-Durchmusterungsreihenfolge aus

1. Falls p rechtes Kind hat: Return Minimum(p.right)
2. Sonst: Falls p linkes Kind ist: Return(p.parent)
3. Sonst: wandere solange nach oben bis der aktuelle Knoten zum ersten Mal **linkes Kind ist;** dann: Return(p.parent)
4. oder die Wurzel erreicht ist; dann: existiert kein Nachfolger.

Pseudocode s. Skript bzw. Folien

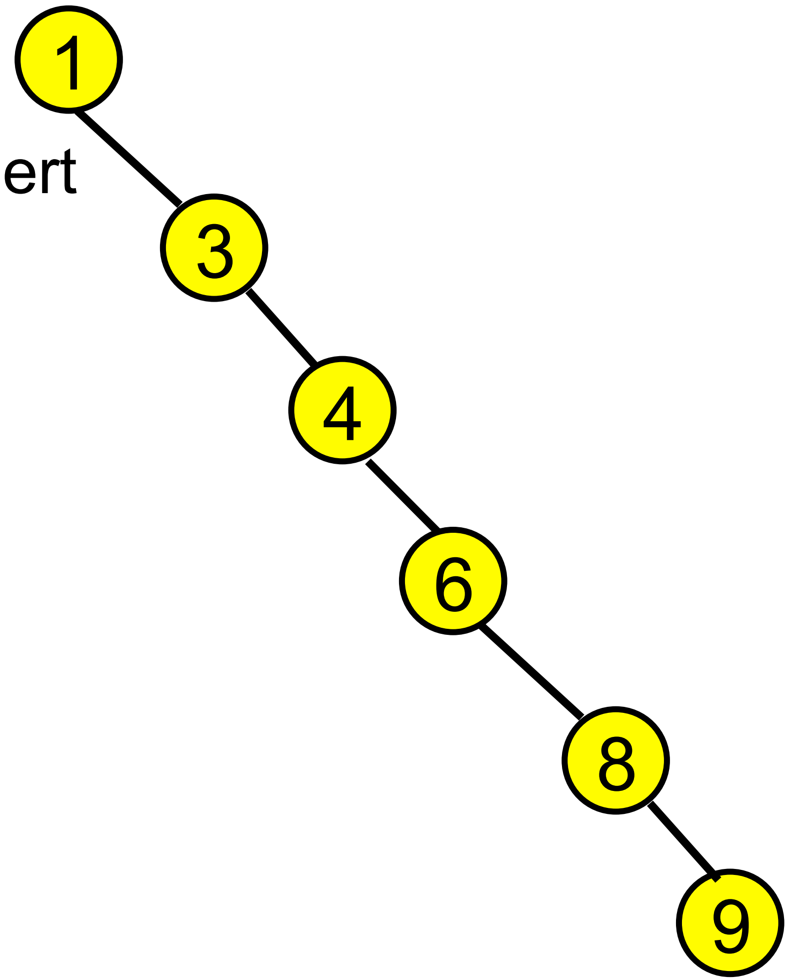
Analyse der Operationen

1. Alle Operationen benötigen eine Laufzeit von $O(h(T))$ für binäre Suchbäume, wobei $h(T)$ die Höhe des gegebenen Suchbaumes T ist.

Frage: Wie hoch kann $h(T)$ für einen binären Suchbaum mit n Knoten sein?

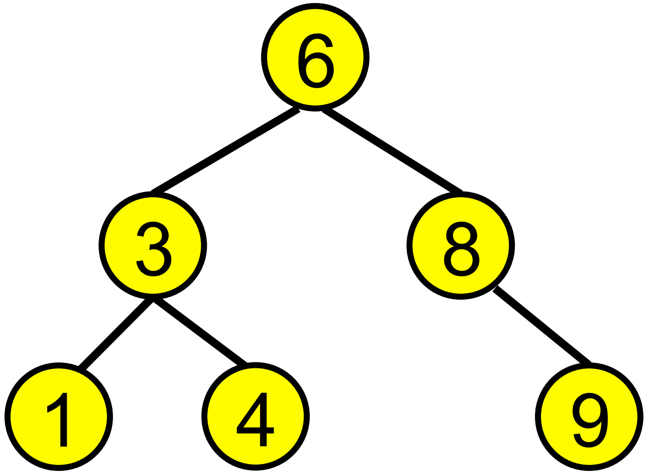
Abhängigkeit der Höhe von der Einfügereihenfolge

Einfügereihenfolge: 1,3,4,6,8,9



Baum ist zu linearer Liste degeneriert
Problem: Suchzeit ist linear,
da $h(T)=n-1$

Einfügereihenfolge: 6,8,3,4,1,9



Diskussion

- Ein binärer Suchbaum T kann im Extremfall zu einer **linearen Liste** ausarten (z.B. Einfüge-reihenfolge sortiert).
- Dann dauert die erfolglose Suche $\Theta(h(T))=\Theta(n)$

- Der andere Extremfall sind **vollständig balancierte Bäume** (d.h. alle Ebenen bis auf evtl. die unterste sind voll besetzt); diese haben die Höhe $h(T)=\Theta(\log n)$.
- Hier dauert die erfolglose Suche $\Theta(h(T))=\Theta(\log(n))$

- Wir werden sehen: für dieses gute Zeitverhalten genügen auch „hinreichend“ balancierte Bäume

Durchschnittliche Suchpfadlänge

Man kann zeigen:

- Die erwartete Suchpfadlänge (über alle möglichen Permutationen von n Einfüge-operationen) ist

  $I(n)=2 \ln n - O(1) = 1.38629\ldots \log n - O(1)$

- D.h. für große n ist die durchschnittliche Suchpfadlänge nur ca. 38% länger als im Idealfall.

- Allerdings ist diese Annahme, dass die Daten in einer zufälligen Reihenfolge eingegeben werden in vielen Anwendungen nicht gerechtfertigt.

Lösung: Balancierte Suchbäume, s. nächste VO