

# Doppelt verkettete Listen

---

## Datenstruktur Liste:

- Platzbedarf  $\Theta(n)$
- Laufzeit Suche:  $\Theta(n)$
- Laufzeit Einfügen/Löschen:  $\Theta(1)$

## Vorteile:

- Schnelles Einfügen/Löschen
- $O(n)$  Speicherbedarf

## Nachteile:

- Hohe Laufzeit für Suche

# 11: Binäre Suchbäume

---

## Binäre Suchbäume

- Verwende Binärbaum
- Speichere Schlüssel „geordnet“

## Binäre Suchbaumeigenschaft:

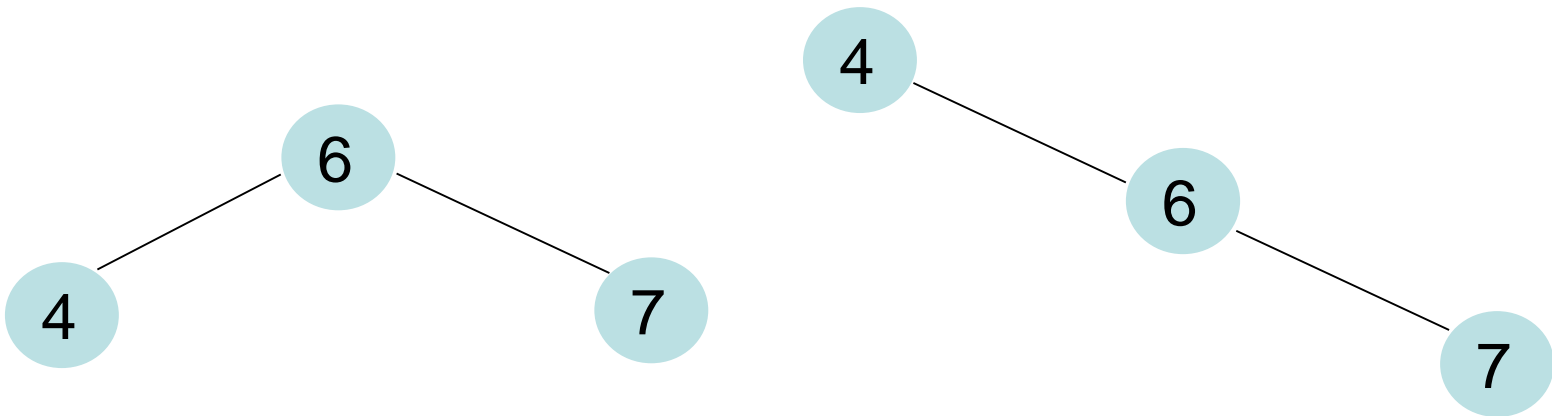
- Sei  $x$  Knoten im binären Suchbaum
- Ist  $y$  Knoten im **linken Unterbaum** von  $x$ , dann gilt  $key[y] \leq key[x]$
- Ist  $y$  Knoten im **rechten Unterbaum** von  $x$ , dann gilt  $key[y] \geq key[x]$

# Binäre Suchbäume

---

## Unterschiedliche Suchbäume

- Beispiel: Schlüsselmenge 4,6,7

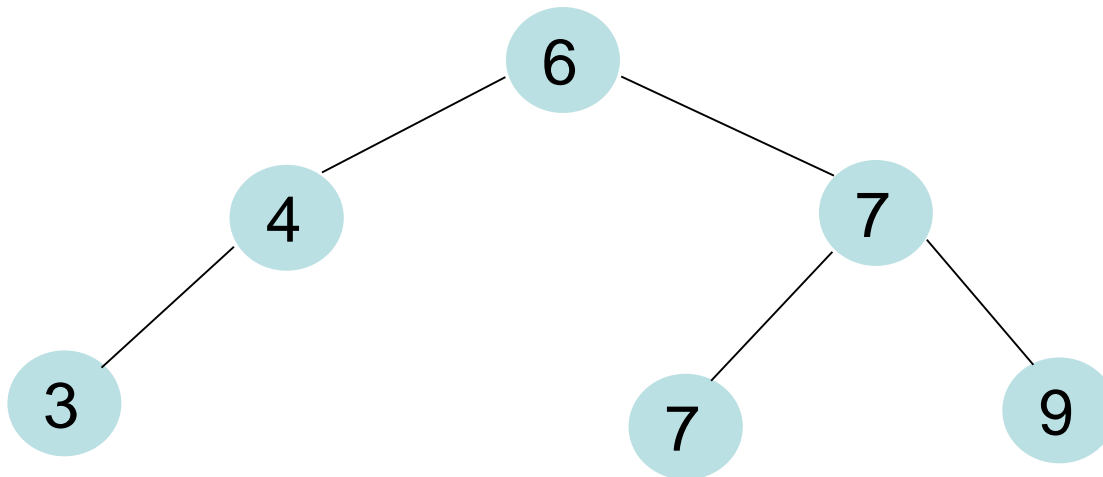


# Binäre Suchbäume

---

## Sortierte Ausgabe aller Schlüssel

- Gegeben binärer Suchbaum
- Wie kann man alle Schlüssel aufsteigend sortiert in  $\Theta(n)$  Zeit ausgeben?

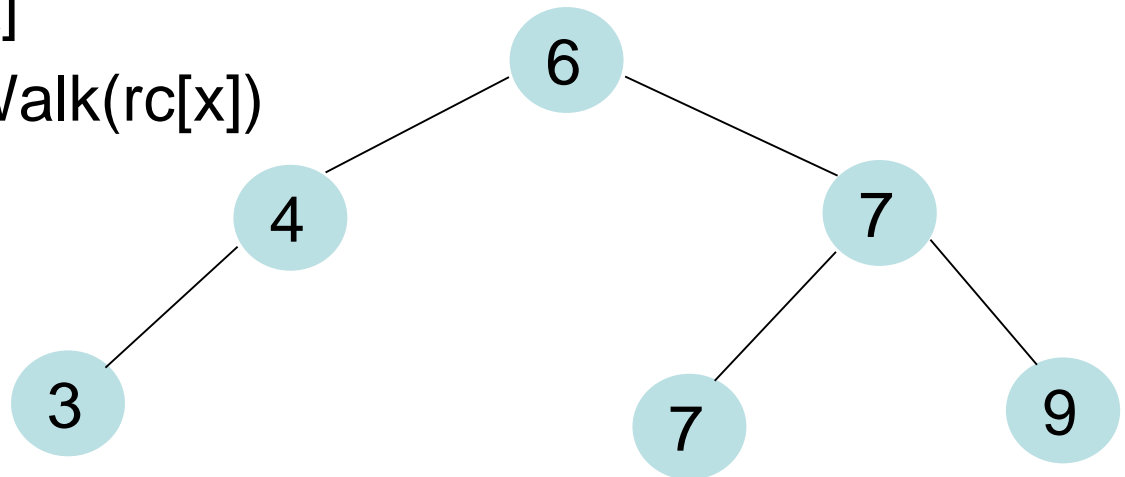


# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

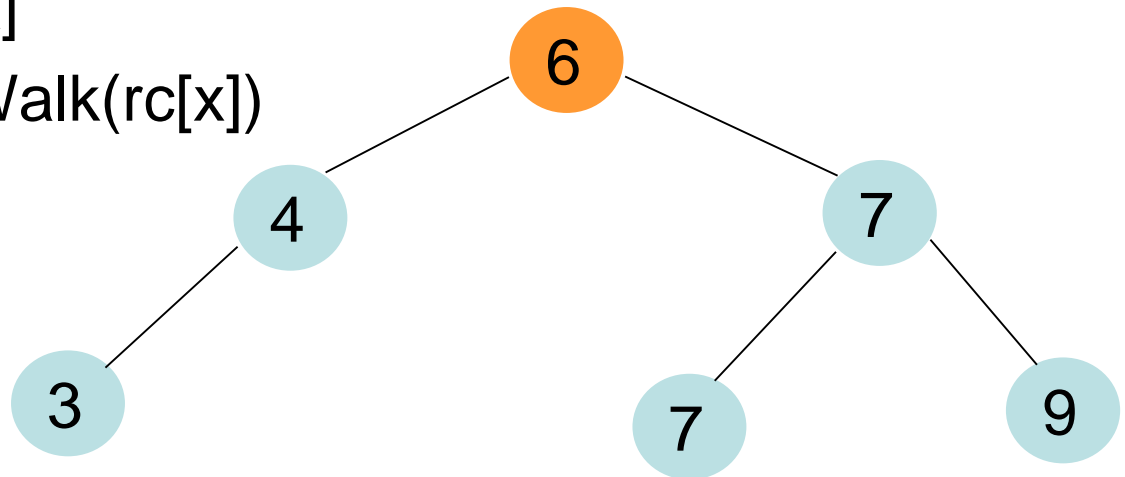


# Binäre Suchbäume

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Aufruf über  
Inorder-Tree-Walk(root[T])

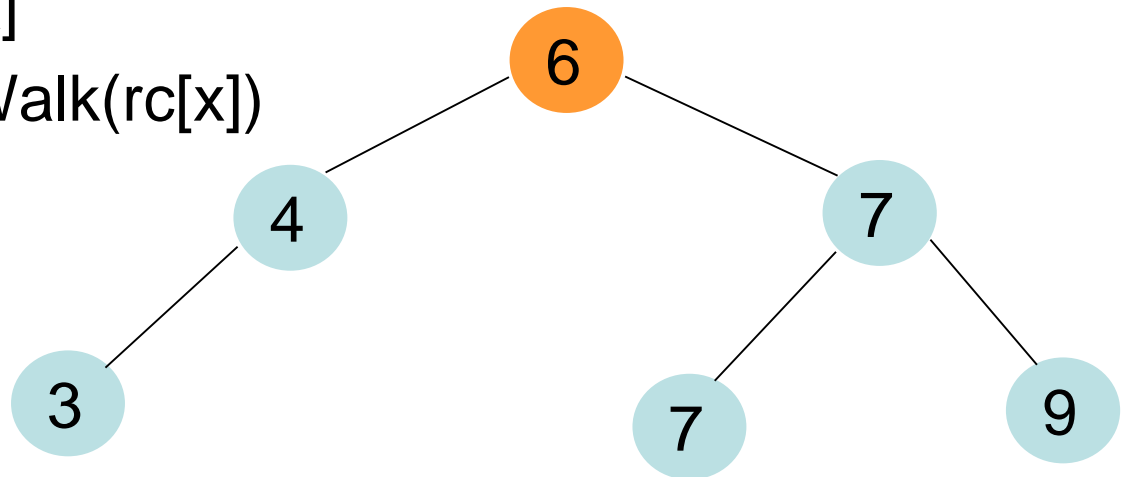


# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

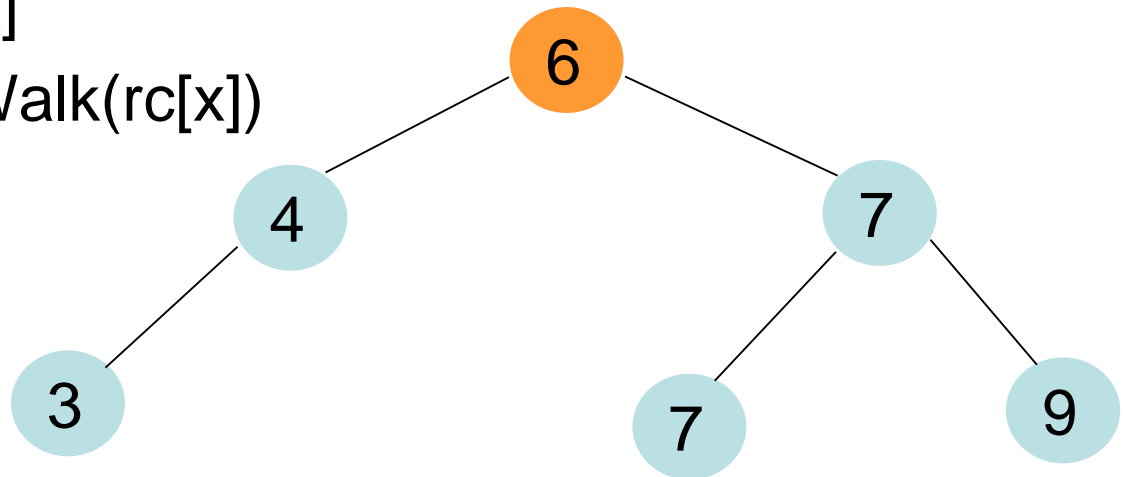


# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. **Inorder-Tree-Walk(lc[x])**
3. Ausgabe key[x]
4. **Inorder-Tree-Walk(rc[x])**



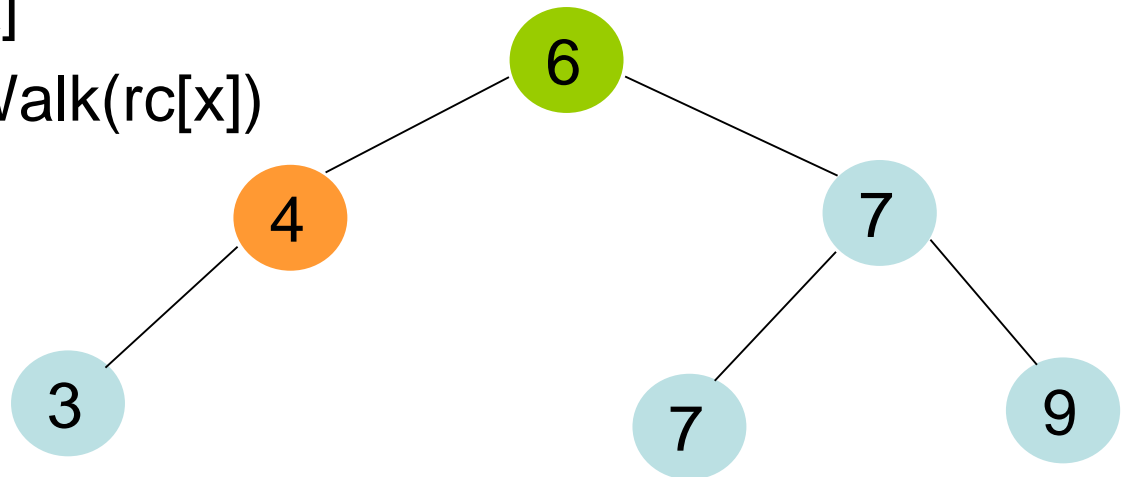


# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

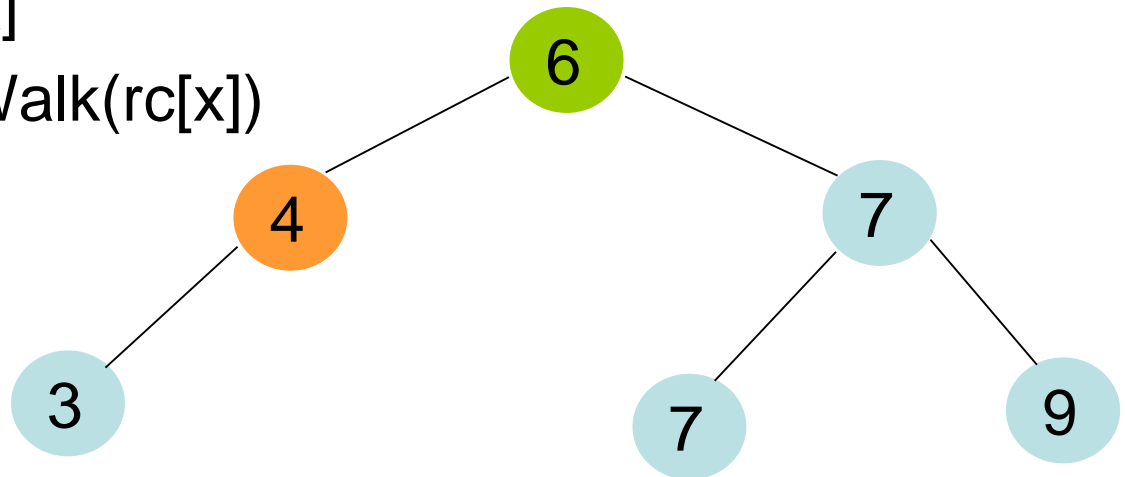


# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. **Inorder-Tree-Walk(lc[x])**
3. Ausgabe key[x]
4. **Inorder-Tree-Walk(rc[x])**

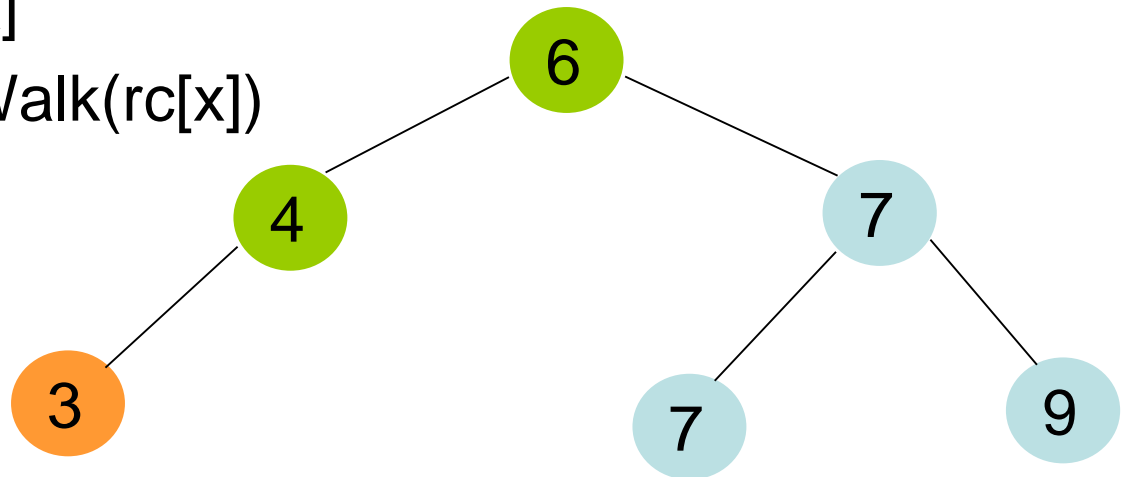


# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])



# Binäre Suchbäume

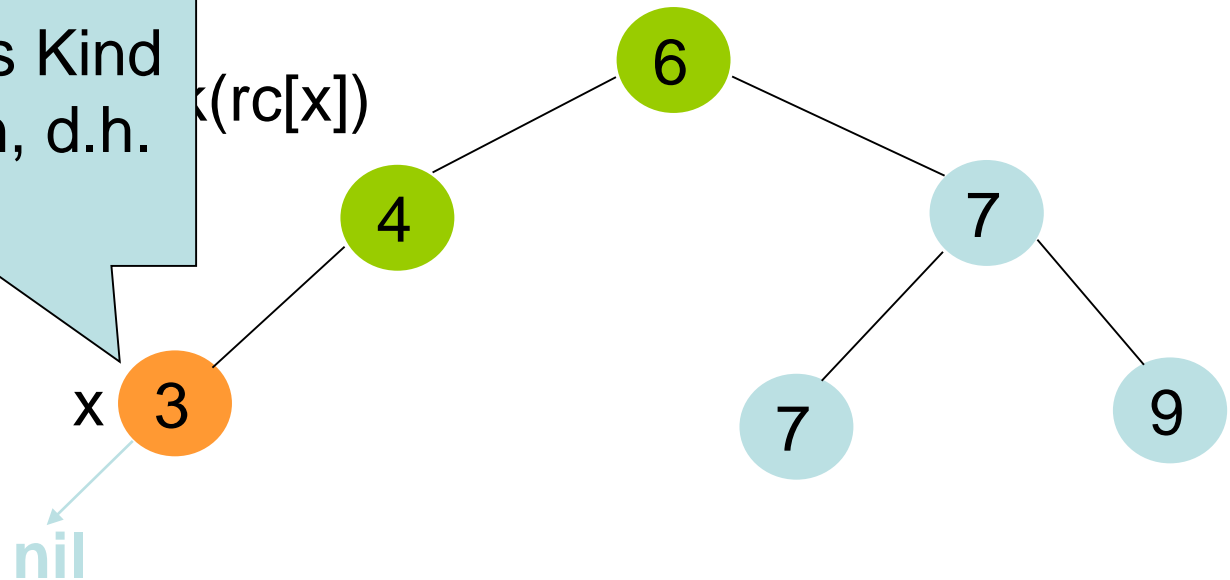
Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**

2. Inorder-Tree-Walk(lc[x])

3. **visit** x  
4. Inorder-Tree-Walk(rc[x])

Kein linkes Kind  
vorhanden, d.h.  
lc[x]=nil

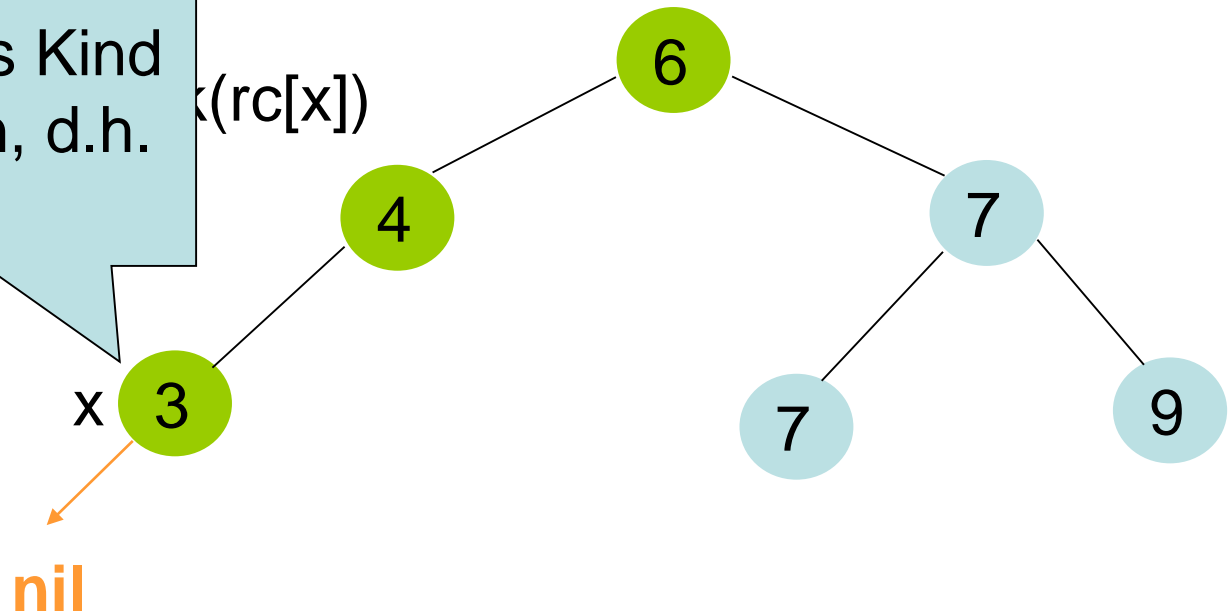


# Binäre Suchbäume

## Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. **visit(x)**
4. Inorder-Tree-Walk(rc[x])

Kein linkes Kind  
vorhanden, d.h.  
 $\text{lc}[x] = \text{nil}$



# Binäre Suchbäume

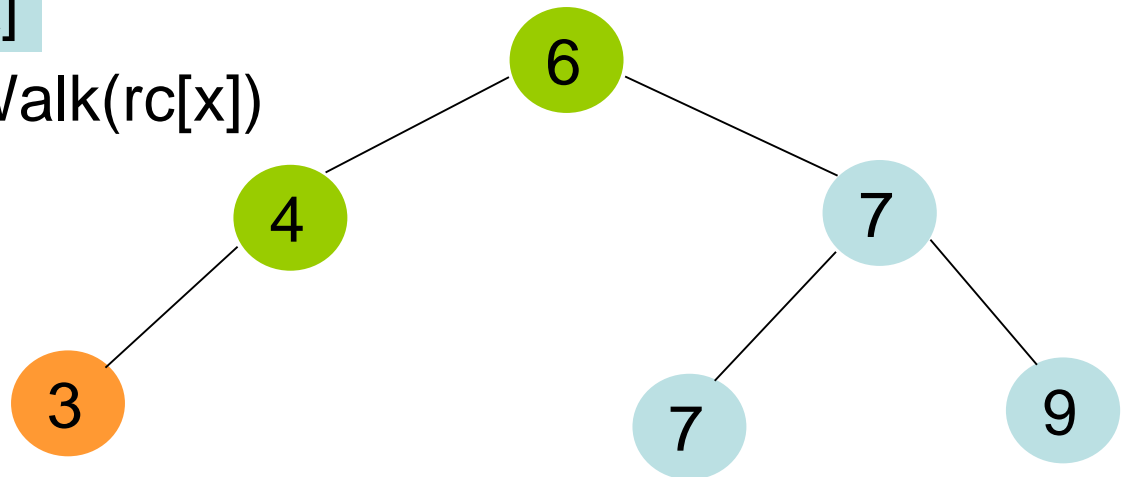
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3



# Binäre Suchbäume

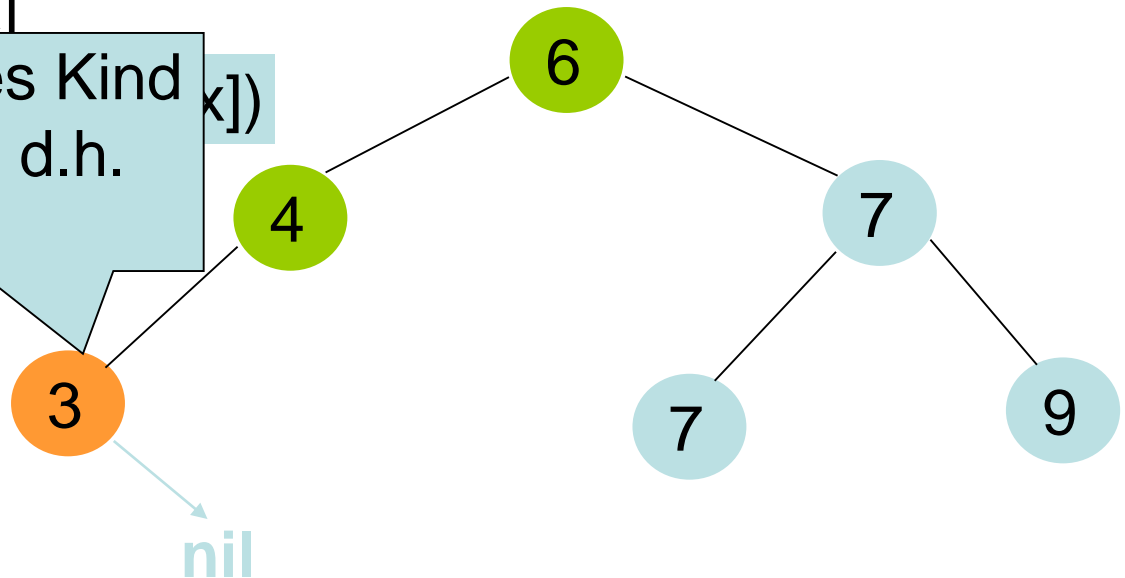
Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3

Kein rechtes Kind vorhanden, d.h.  $rc[x]=\text{nil}$



# Binäre Suchbäume

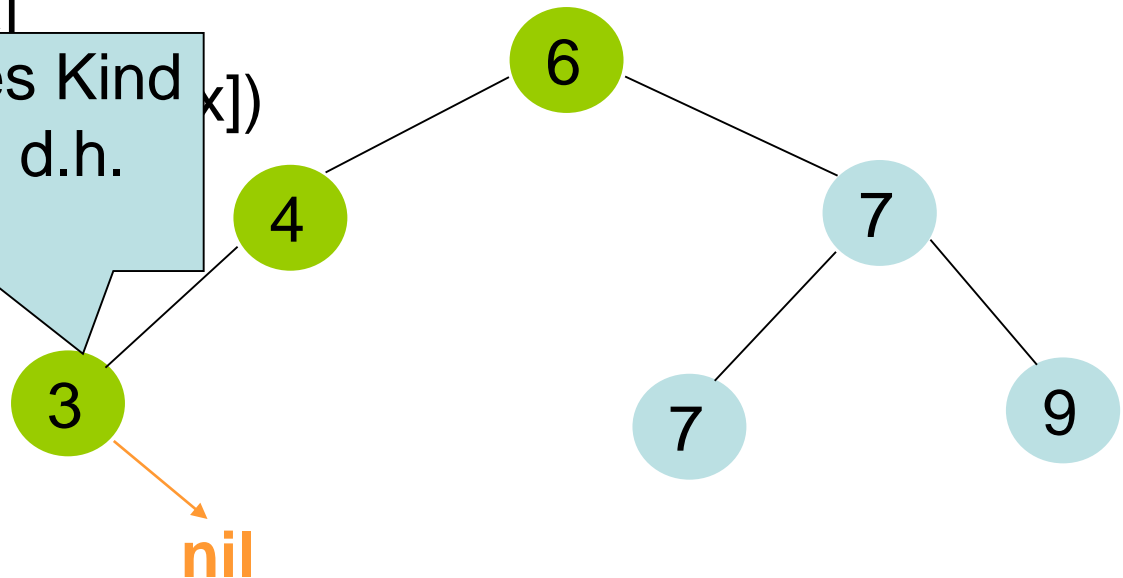
Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3

Kein rechtes Kind  
vorhanden, d.h.  
 $rc[x]=\text{nil}$





# Binäre Suchbäume

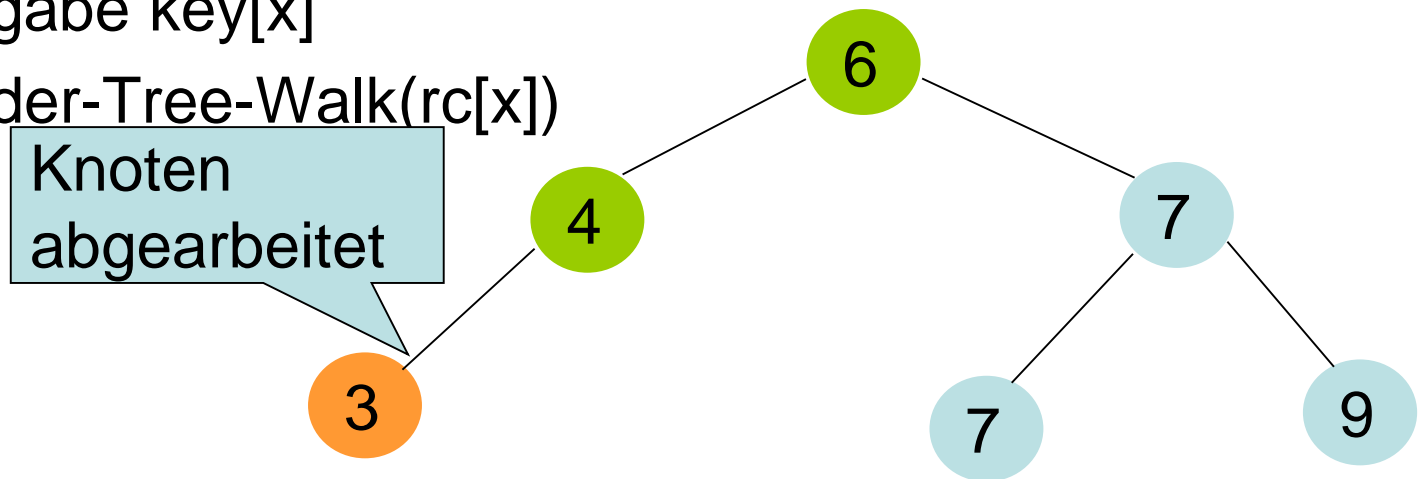
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3



# Binäre Suchbäume

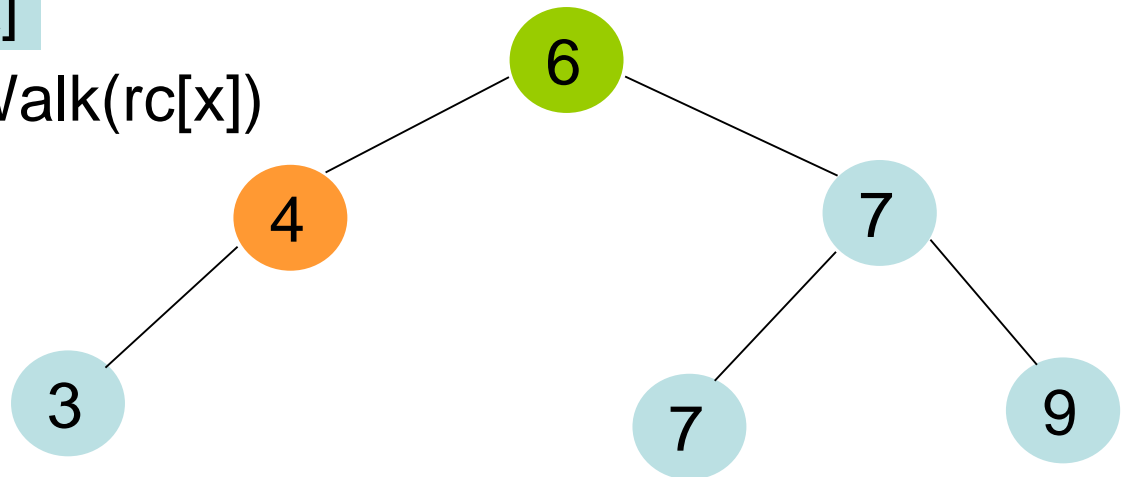
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4



# Binäre Suchbäume

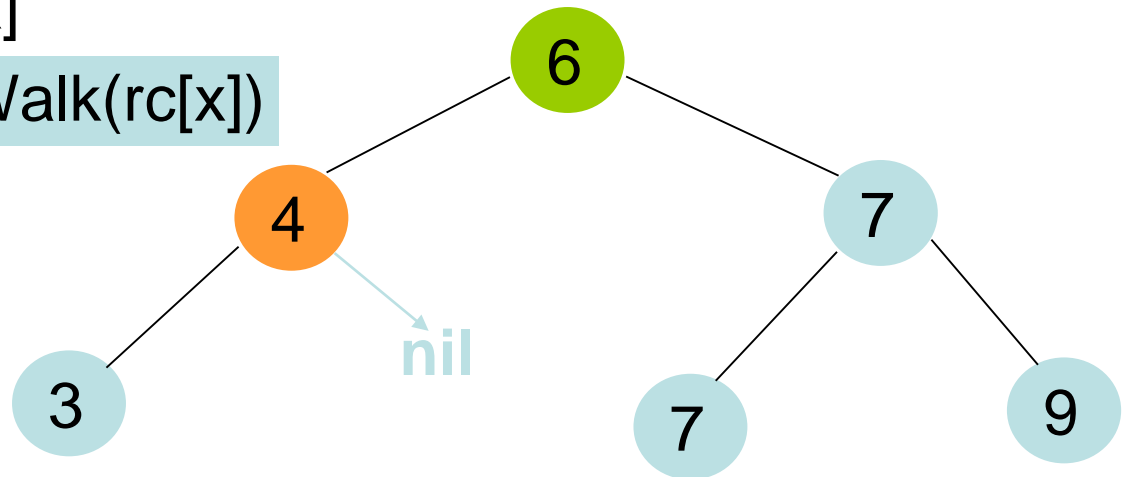
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4



# Binäre Suchbäume

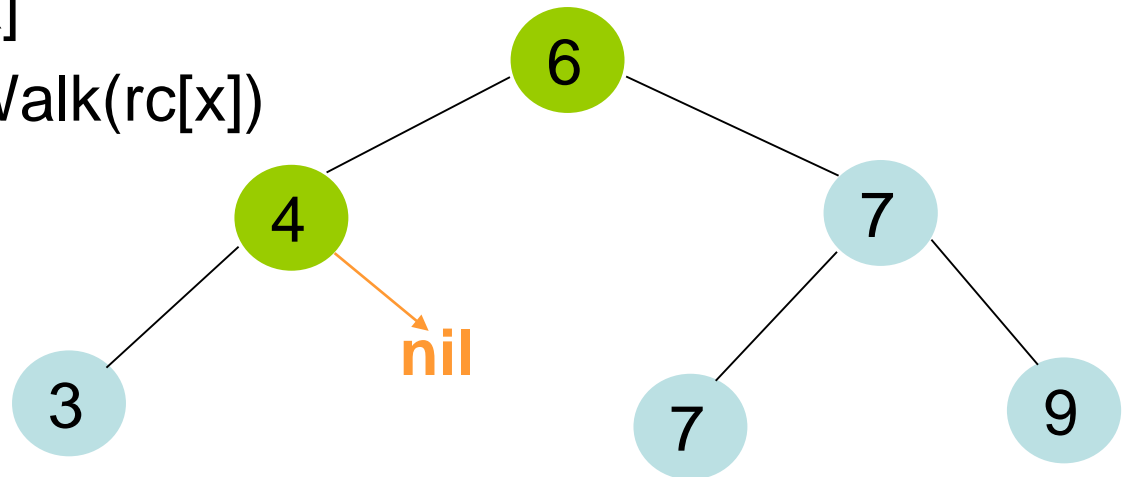
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4



# Binäre Suchbäume

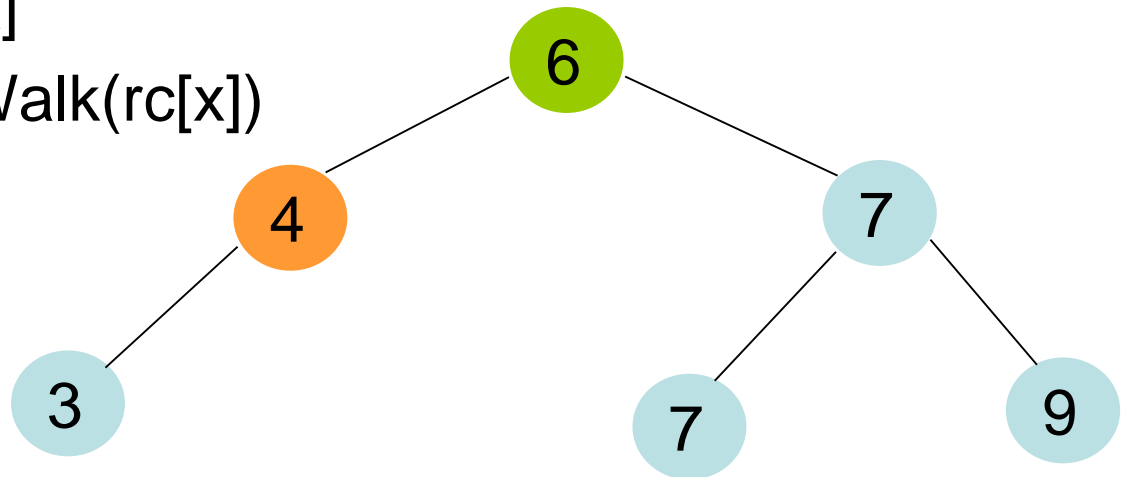
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4



# Binäre Suchbäume

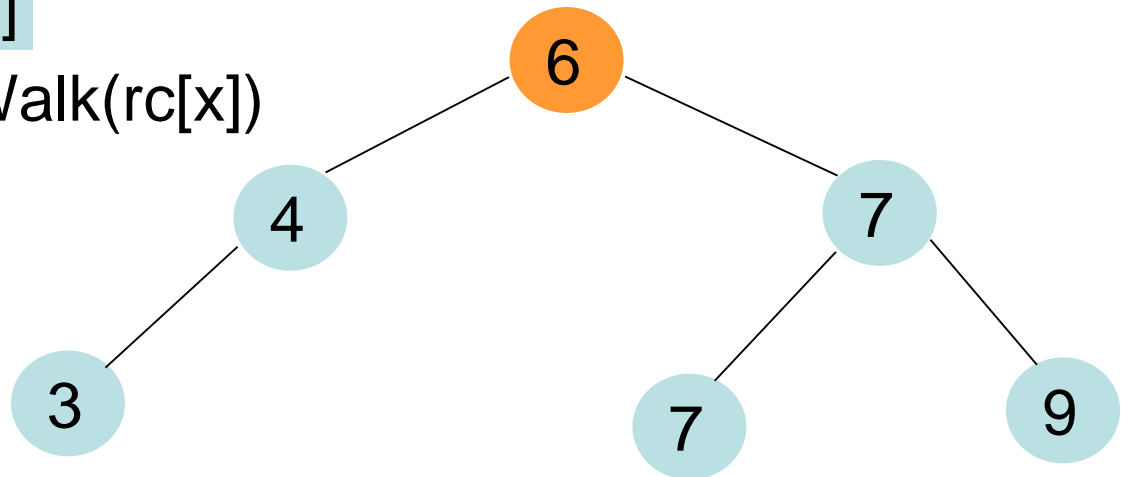
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6



# Binäre Suchbäume

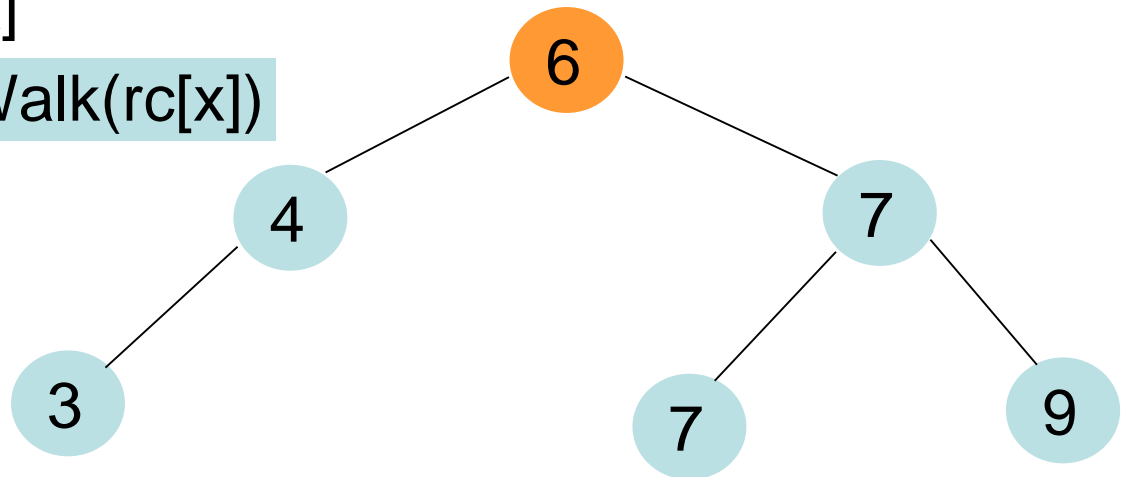
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6



# Binäre Suchbäume

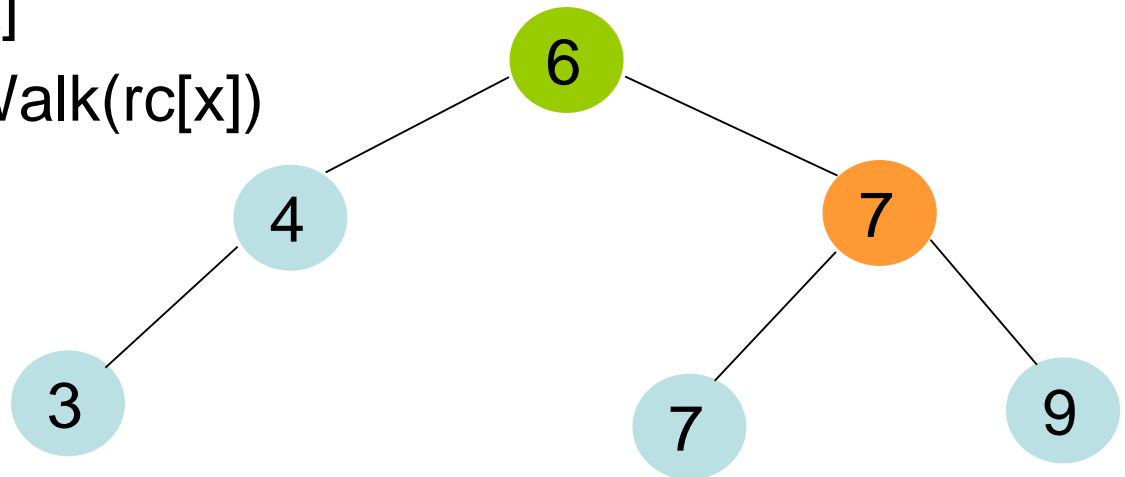
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6





# Binäre Suchbäume

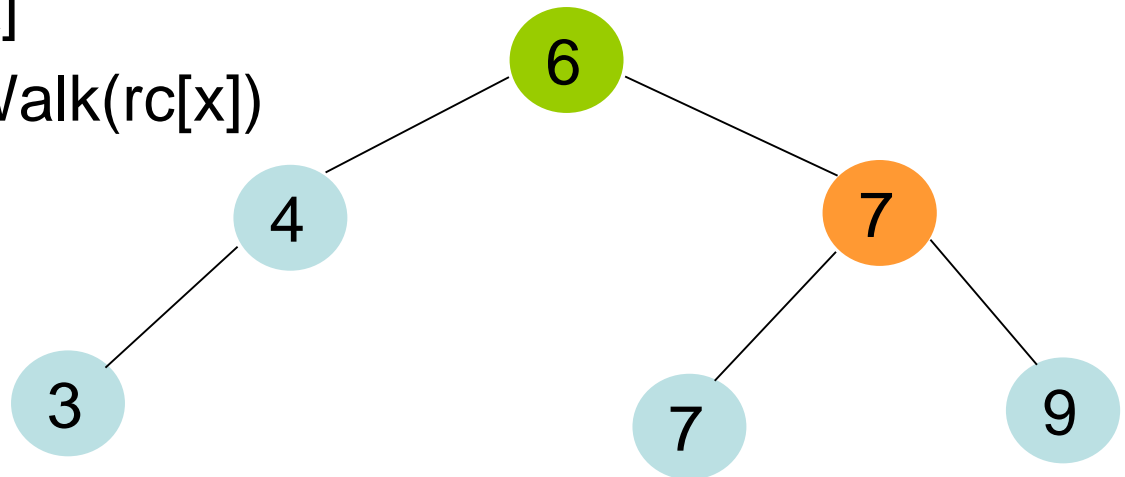
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. **Inorder-Tree-Walk(lc[x])**
3. Ausgabe key[x]
4. **Inorder-Tree-Walk(rc[x])**

Ausgabe:

3, 4, 6



# Binäre Suchbäume

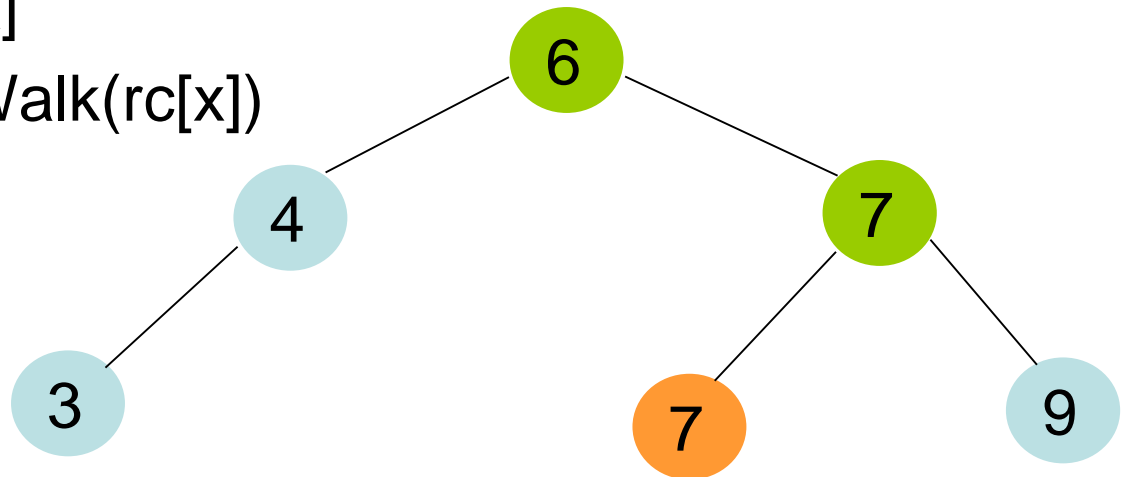
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6



# Binäre Suchbäume

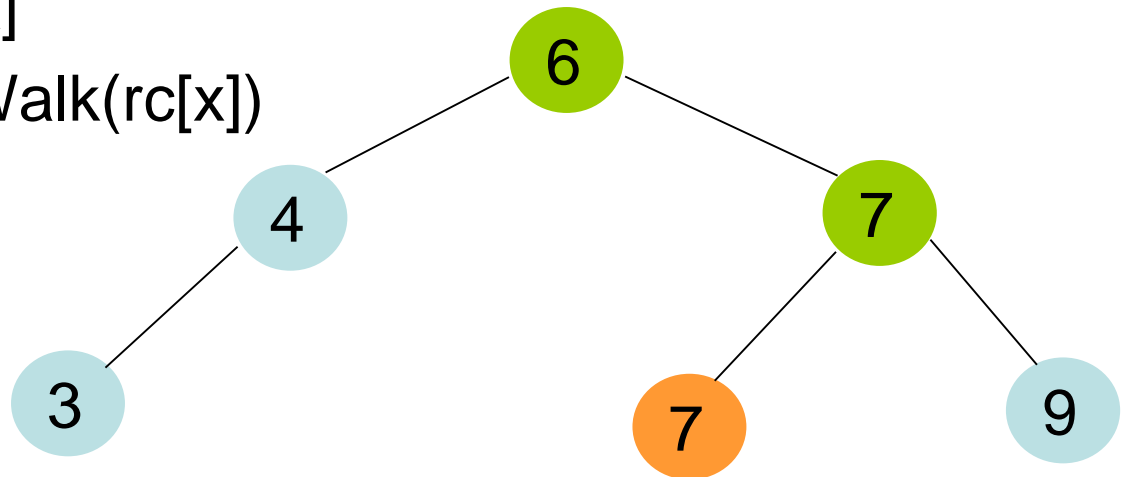
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6



# Binäre Suchbäume

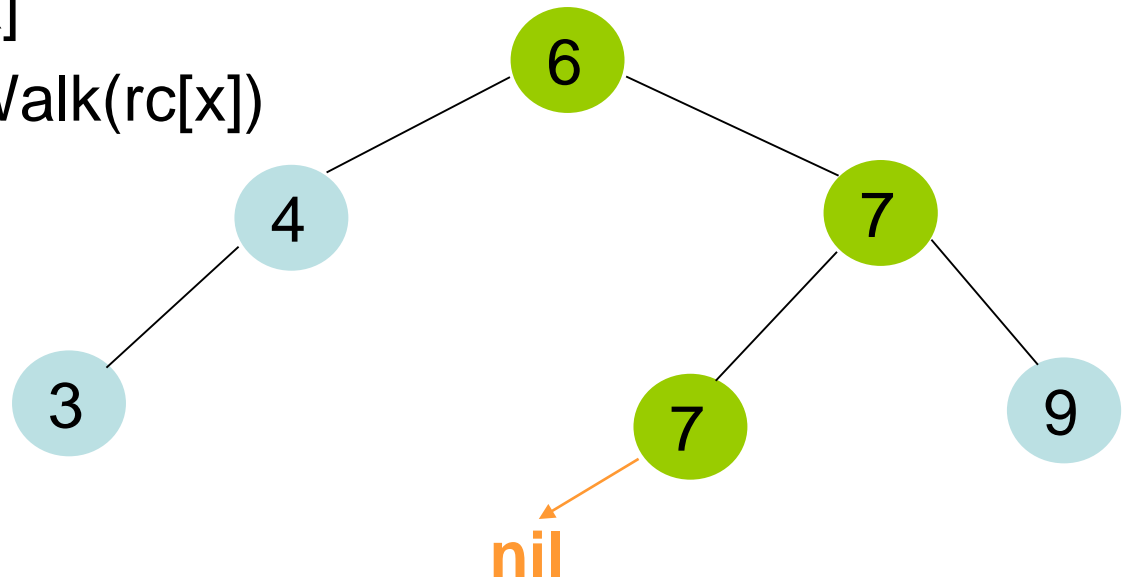
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6



# Binäre Suchbäume

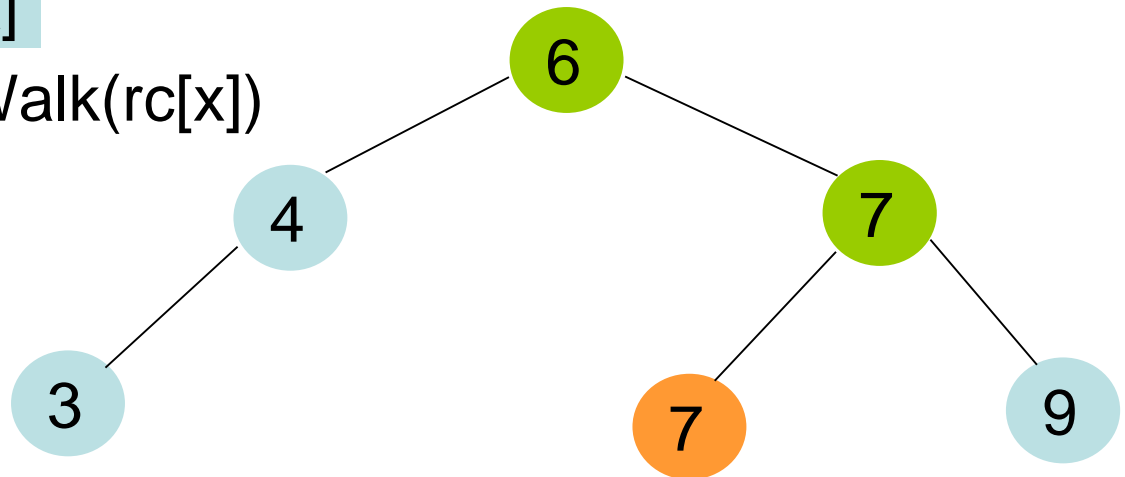
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7



# Binäre Suchbäume

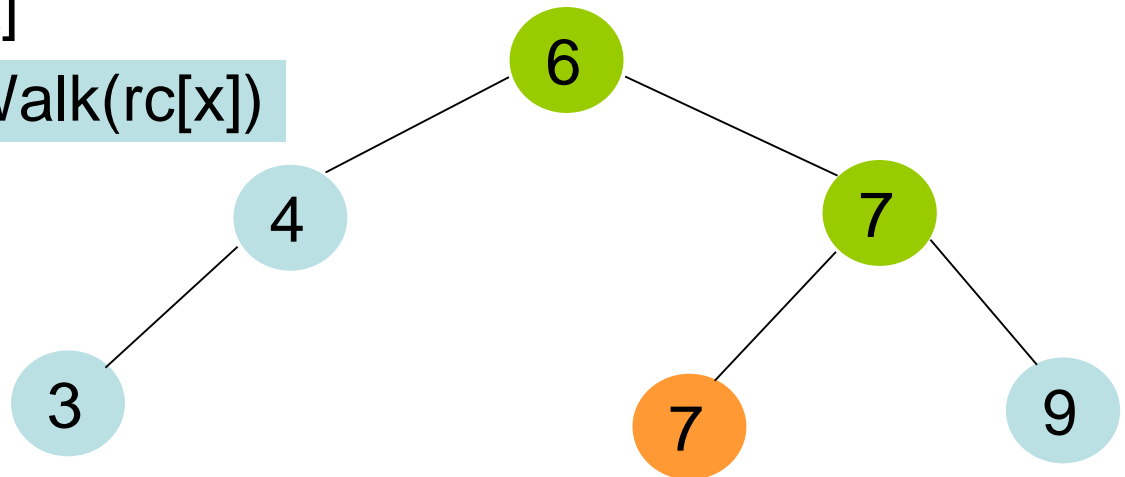
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7



# Binäre Suchbäume

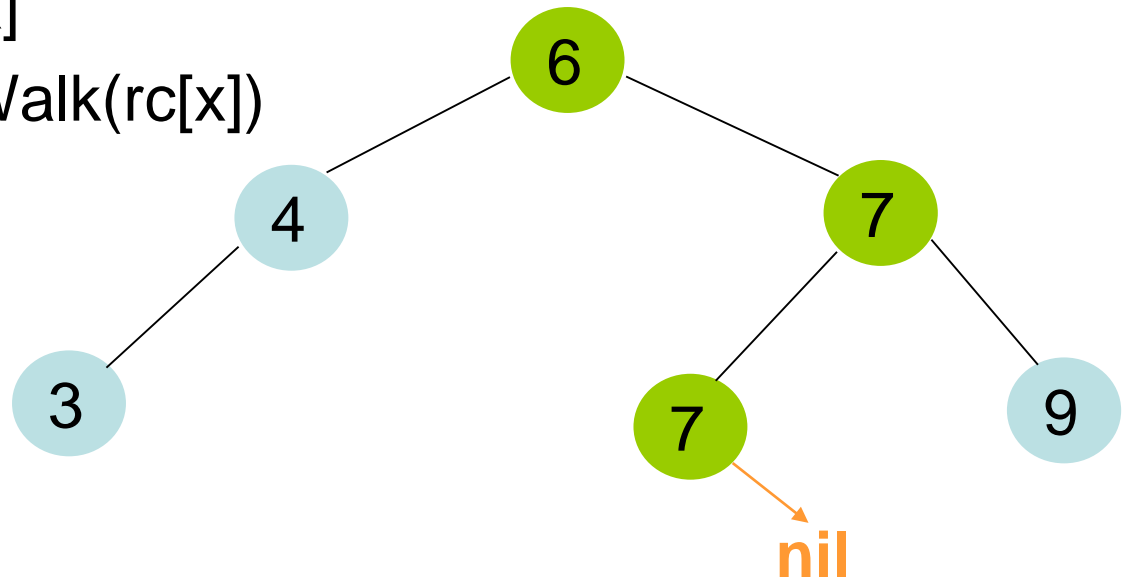
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6



# Binäre Suchbäume

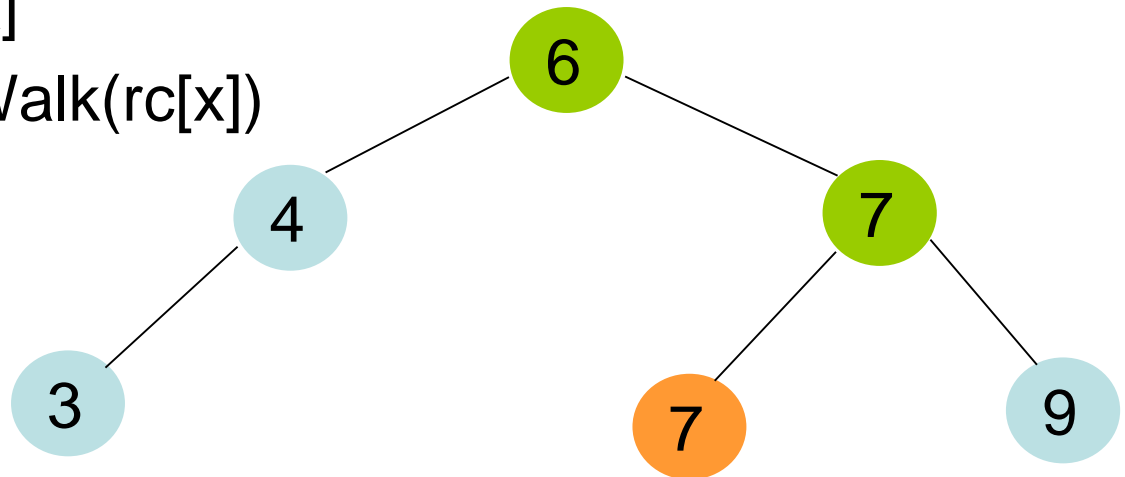
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7





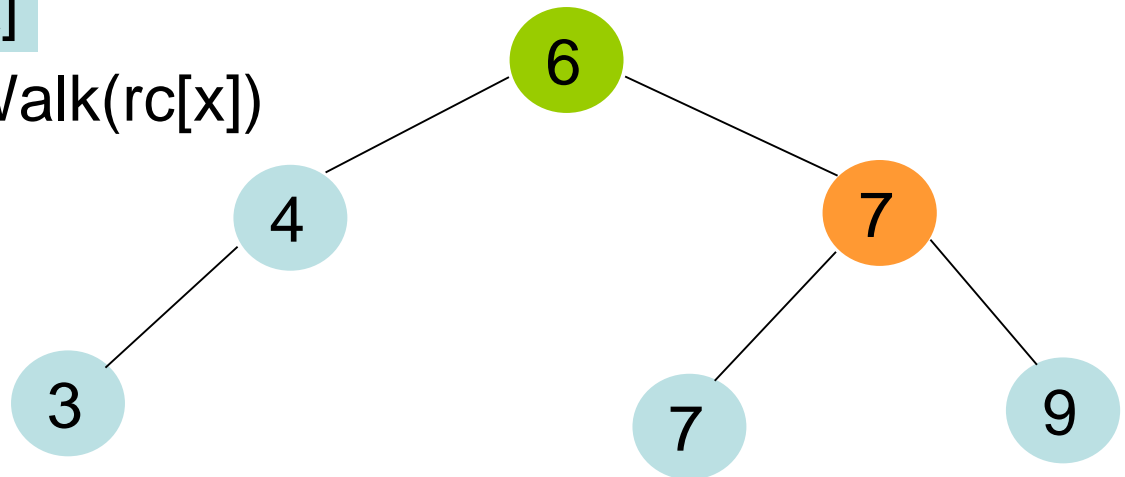
# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:  
3, 4, 6, 7, 7



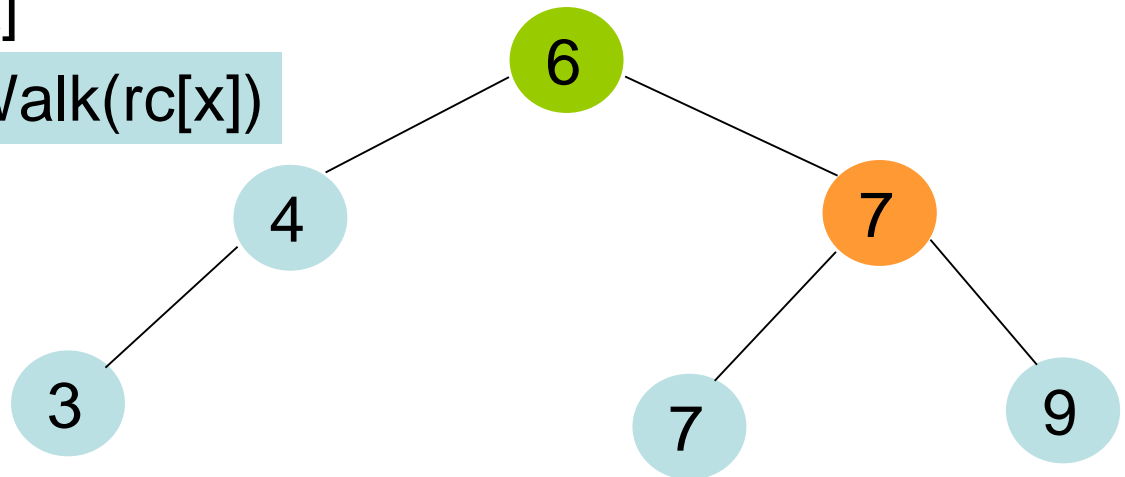
# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:  
3, 4, 6, 7, 7



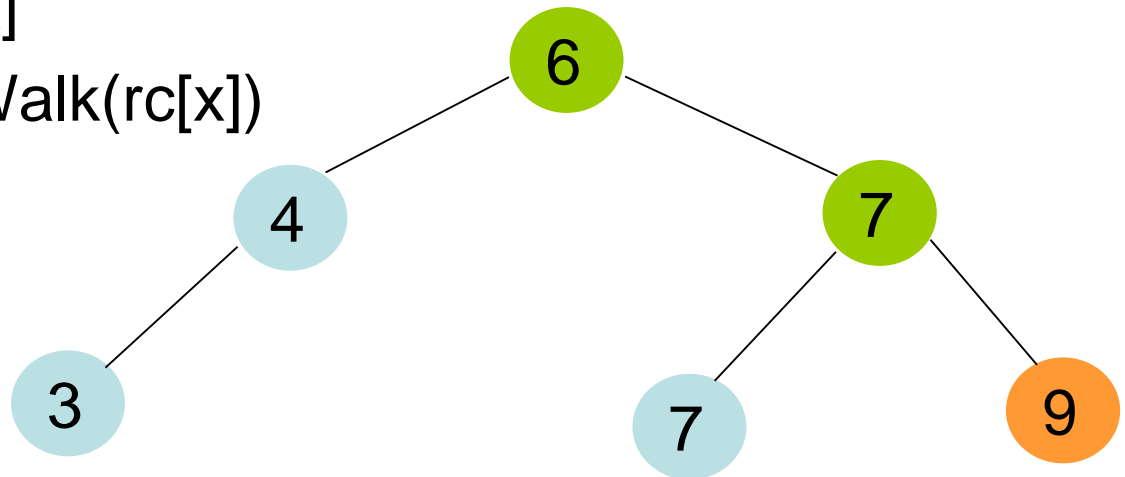
# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:  
3, 4, 6, 7, 7



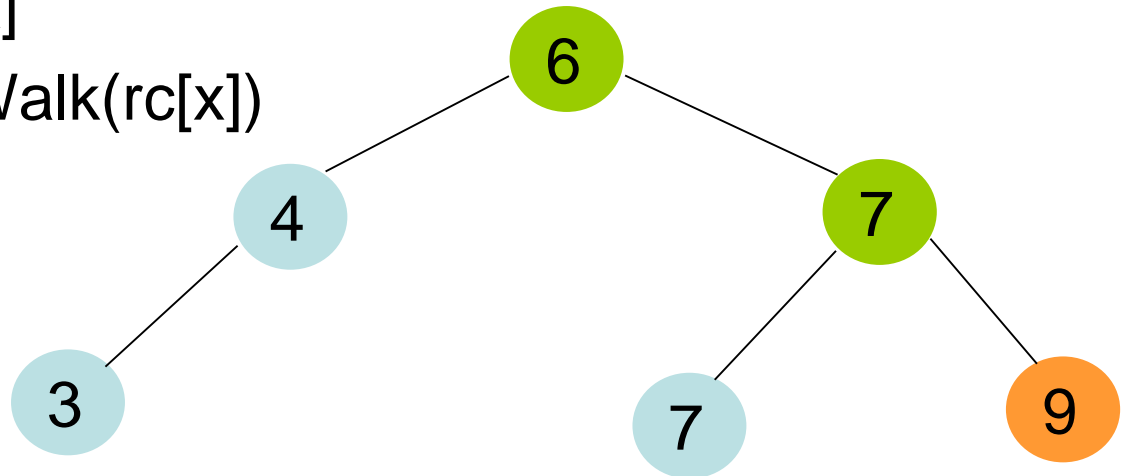
# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:  
3, 4, 6, 7, 7



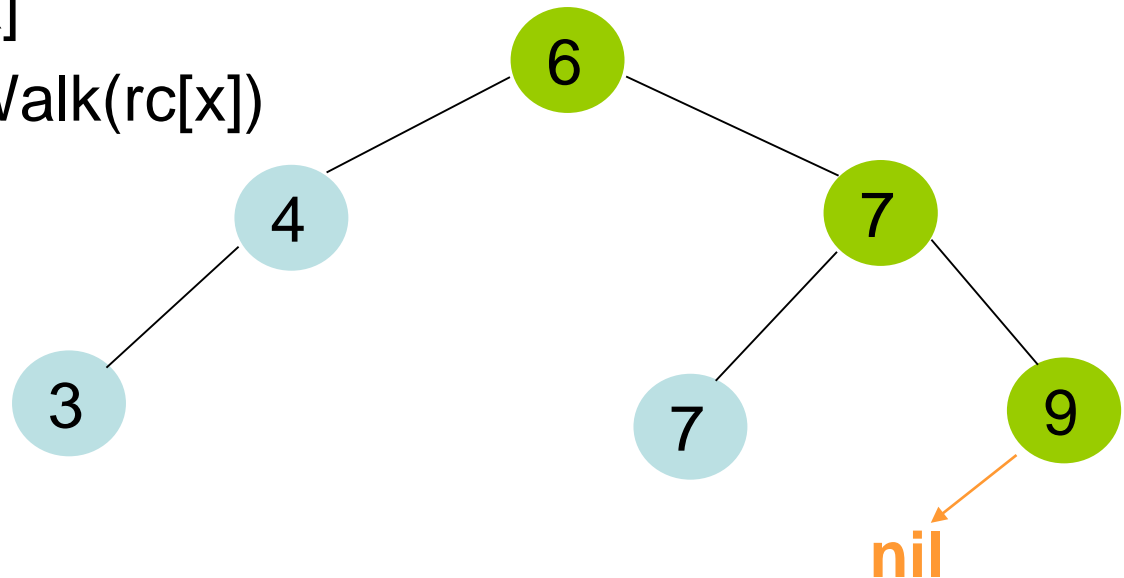
# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:  
3, 4, 6, 7, 7



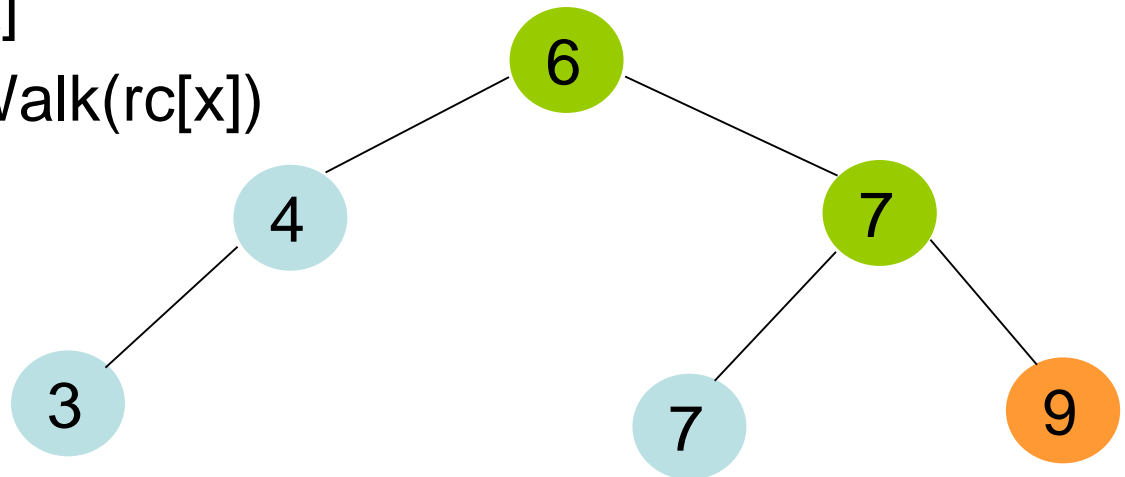
# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:  
3, 4, 6, 7, 7



# Binäre Suchbäume

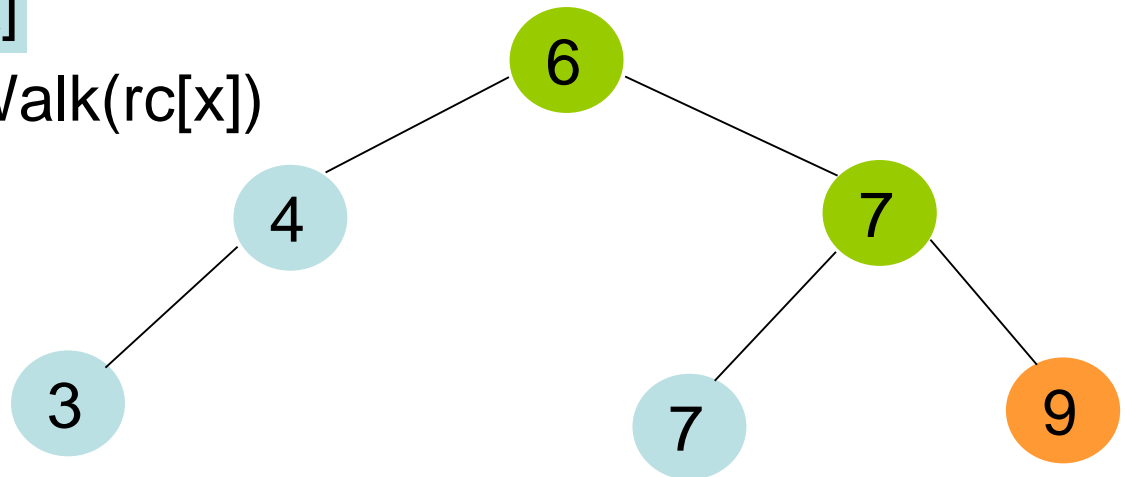
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9



# Binäre Suchbäume

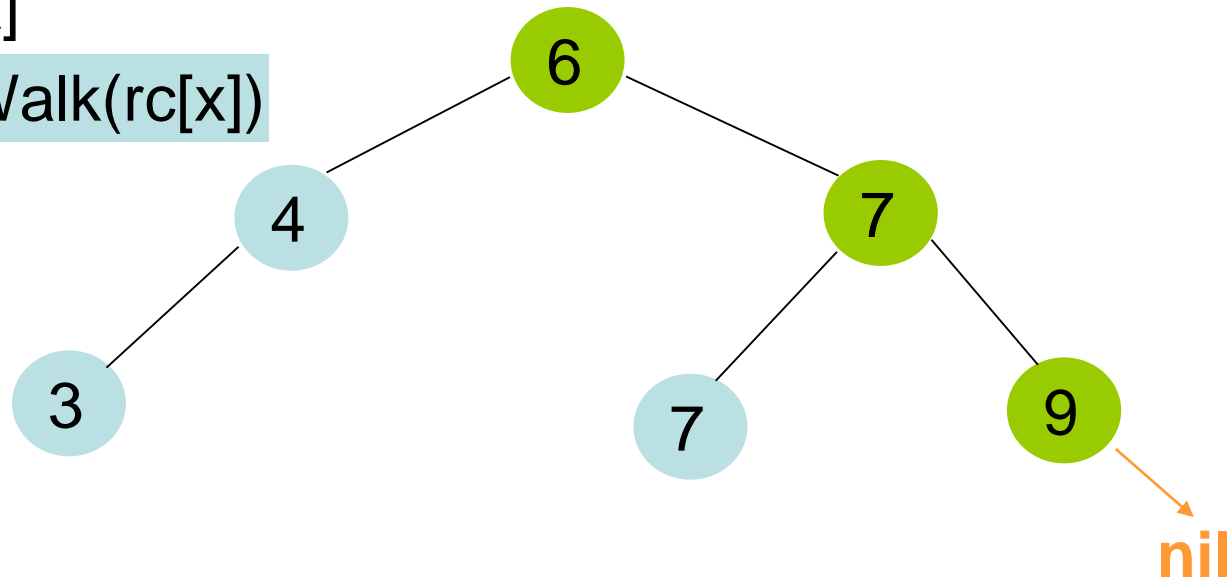
---

Inorder-Tree-Walk(x)

1. **if  $x \neq \text{nil}$  then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9





# Binäre Suchbäume

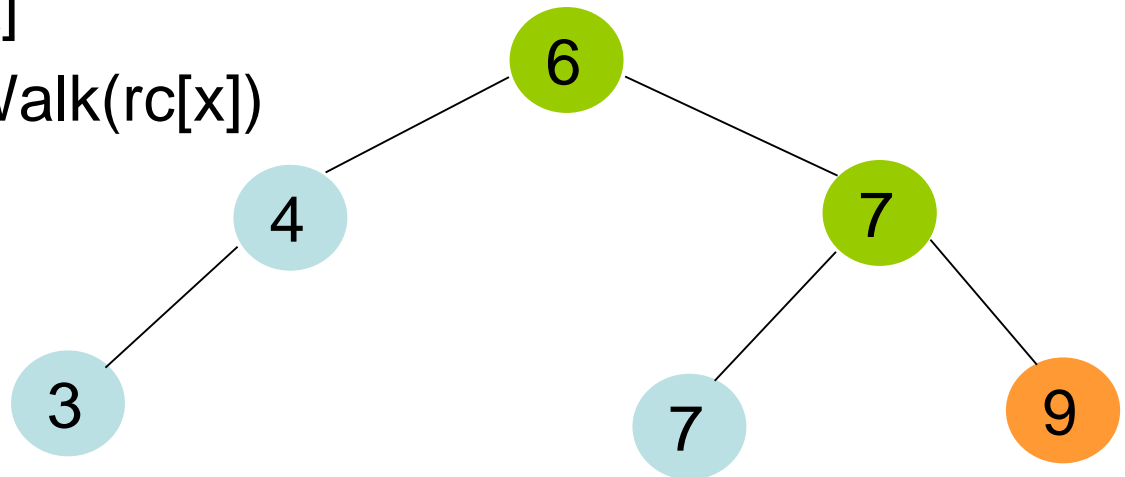
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9



# Binäre Suchbäume

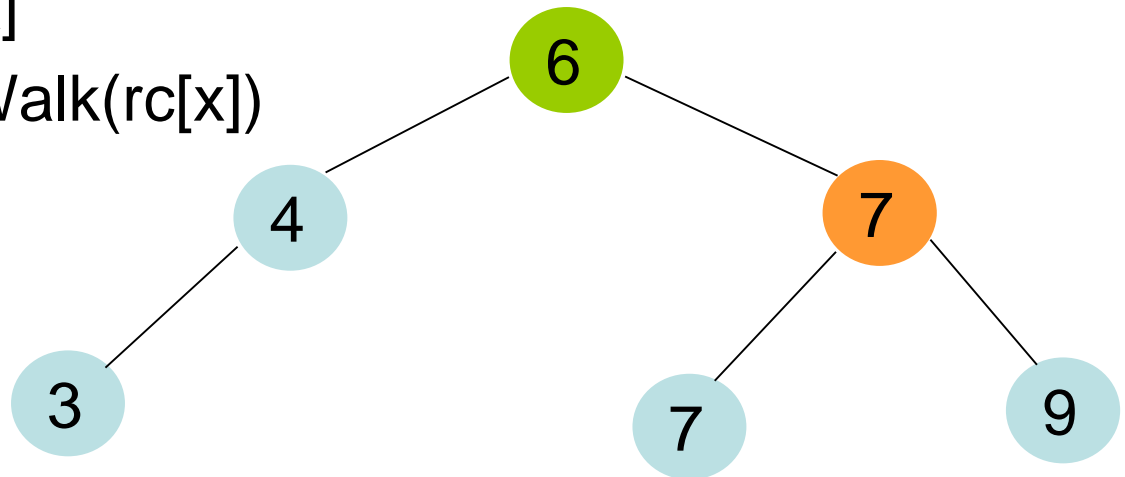
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9



# Binäre Suchbäume

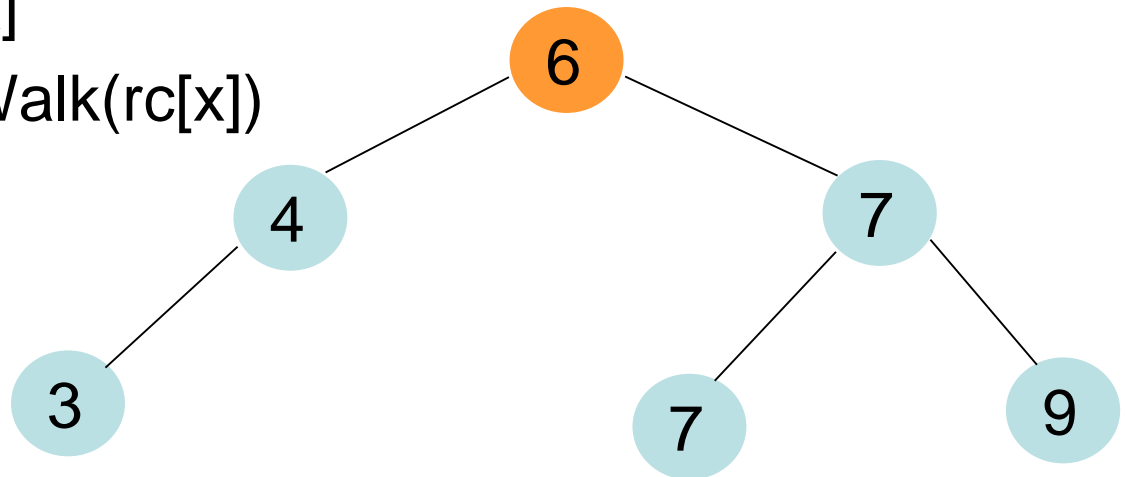
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9



# Binäre Suchbäume

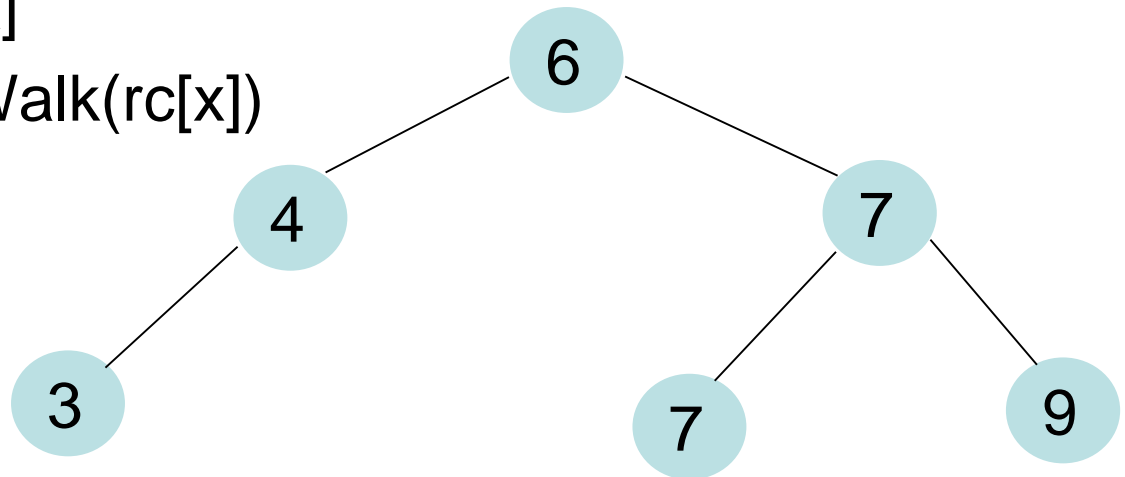
---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2. Inorder-Tree-Walk(lc[x])
3. Ausgabe key[x]
4. Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9



# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9

**Ergebnis:** aufsteigend sortierte Ausgabe der Schlüssel

**Beweis:** Suchbaumeigenschaft + vollständige Induktion

**Quizfrage:** Wie erreiche ich absteigend sortierte Ausgabe?

# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9

**Laufzeit:**  $\Theta(n)$ , wobei  $n$  die Anzahl Knoten in  $T$  ist.

# Binäre Suchbäume

---

**Laufzeit:**  $\Theta(n)$ , wobei  $n$  die Anzahl Knoten in  $T$  ist.

**Beweis:** Sei  $n$  bei Aufruf von `Inorder-Tree-Walk(x)` die Anzahl Knoten im Teilbaum von  $x$ . Dann gilt für die Laufzeit  $T(n)$  von `Inorder-Tree-Walk(x)` für eine Konstante  $c > 0$ :

- Basisfall:  $T(0) \leq c$
- Rekursive Aufrufe: es existiert ein  $i \in \{1, \dots, n\}$ , so dass  $T(n) \leq T(i-1) + c + T(n-i)$  ist.

**Behauptung:**  $T(n) \leq 2c \cdot n + c$

**Beweis:** vollständige Induktion über  $n$ , angefangen mit  $n=0$ .

# Binäre Suchbäume

---

Inorder-Tree-Walk(x)

1. **if**  $x \neq \text{nil}$  **then**
2.   Inorder-Tree-Walk(lc[x])
3.   Ausgabe key[x]
4.   Inorder-Tree-Walk(rc[x])

Ausgabe:

3, 4, 6, 7, 7, 9

**Quizfrage:** Warum kann ich keinen Suchbaum in  $O(n)$  Zeit mit vergleichsbasiertem Verfahren aufbauen?



# Binäre Suchbäume

---

## Suchen in Binärbäumen

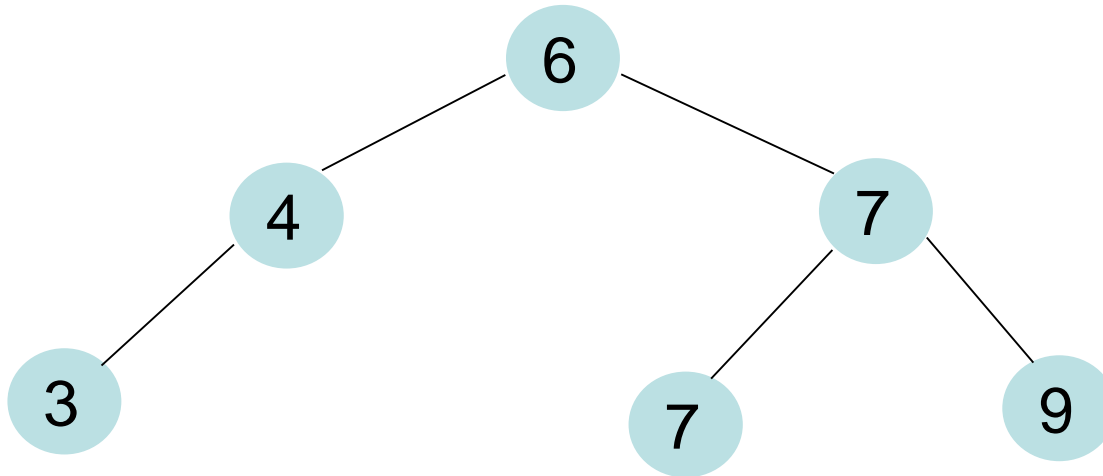
- Gegeben ist Schlüssel  $k$
- Gesucht ist ein Knoten mit Schlüssel  $k$

# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if**  $x = \text{nil}$  **or**  $k = \text{key}[x]$  **then return**  $x$
2. **if**  $k < \text{key}[x]$  **then return**  $\text{Baumsuche}(\text{lc}[x], k)$
3. **else** **return**  $\text{Baumsuche}(\text{rc}[x], k)$

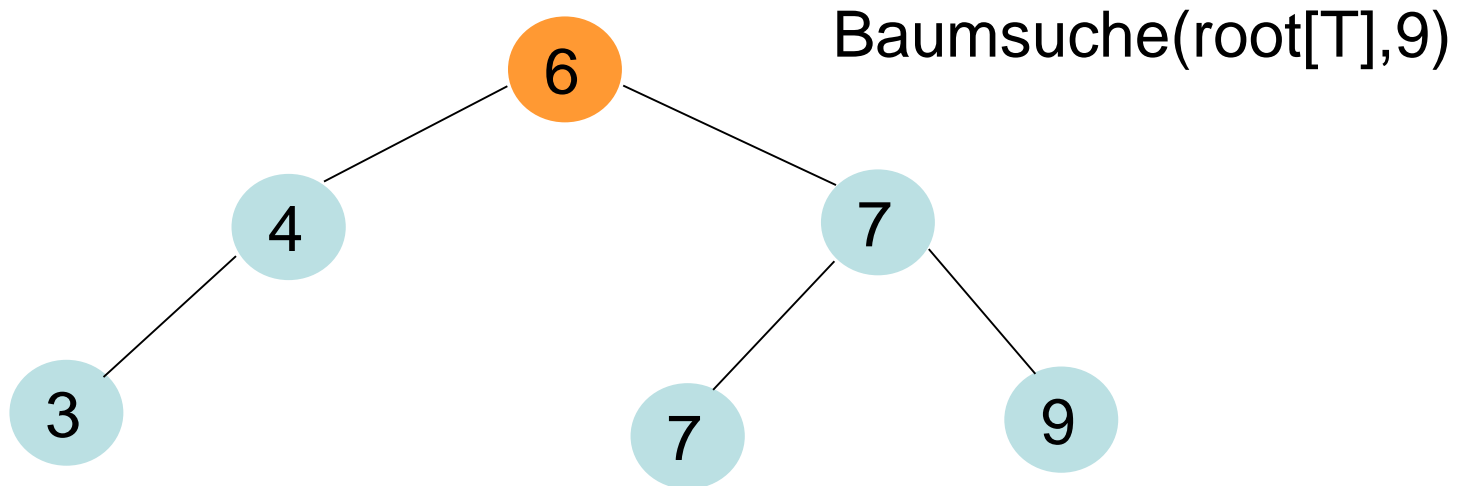


# Binäre Suchbäume

Baumsuche(x,k)

Aufruf mit  
x=root[T]

1. **if** x=nil **or** k=key[x] **then return** x
2. **if** k<key[x] **then return** Baumsuche(lc[x],k)
3. **else** return Baumsuche(rc[x],k)

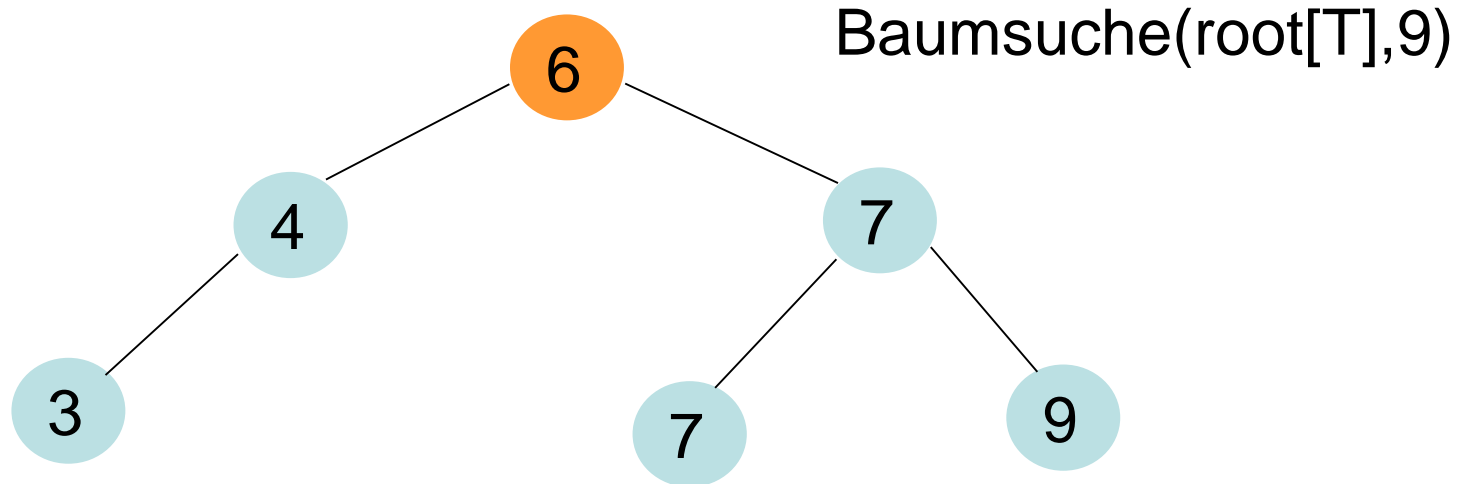


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if  $x = \text{nil}$  or  $k = \text{key}[x]$  then return  $x$**
2. **if  $k < \text{key}[x]$  then return Baumsuche(lc[x],k)**
3. **else return Baumsuche(rc[x],k)**

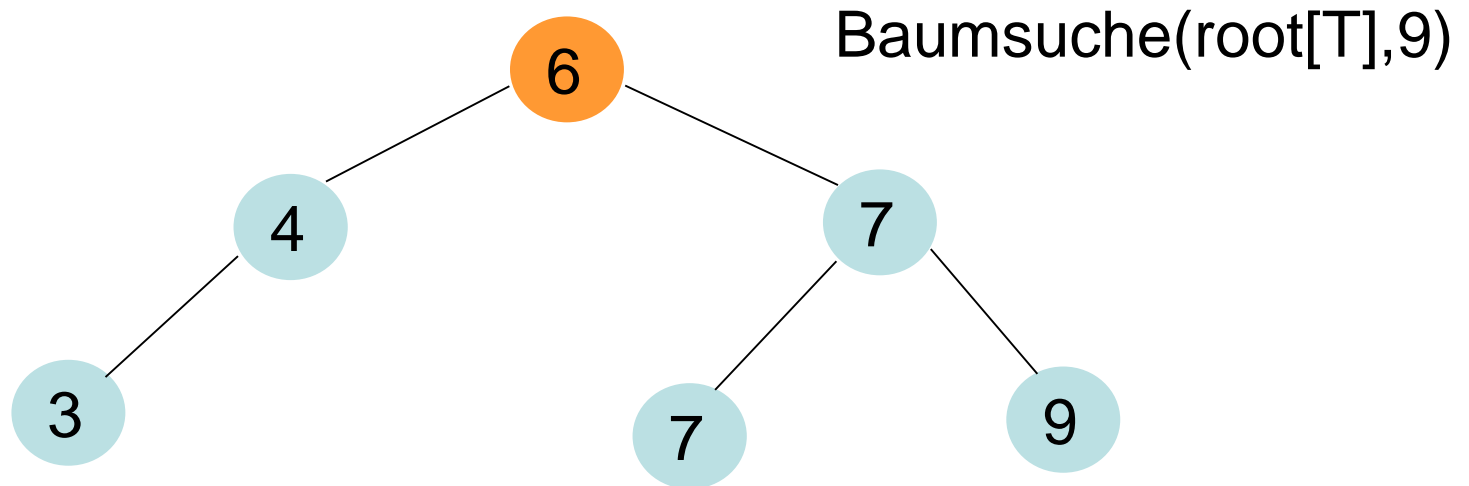


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if**  $x = \text{nil}$  **or**  $k = \text{key}[x]$  **then return**  $x$
2. **if**  $k < \text{key}[x]$  **then return**  $\text{Baumsuche}(\text{lc}[x], k)$
3. **else** **return**  $\text{Baumsuche}(\text{rc}[x], k)$

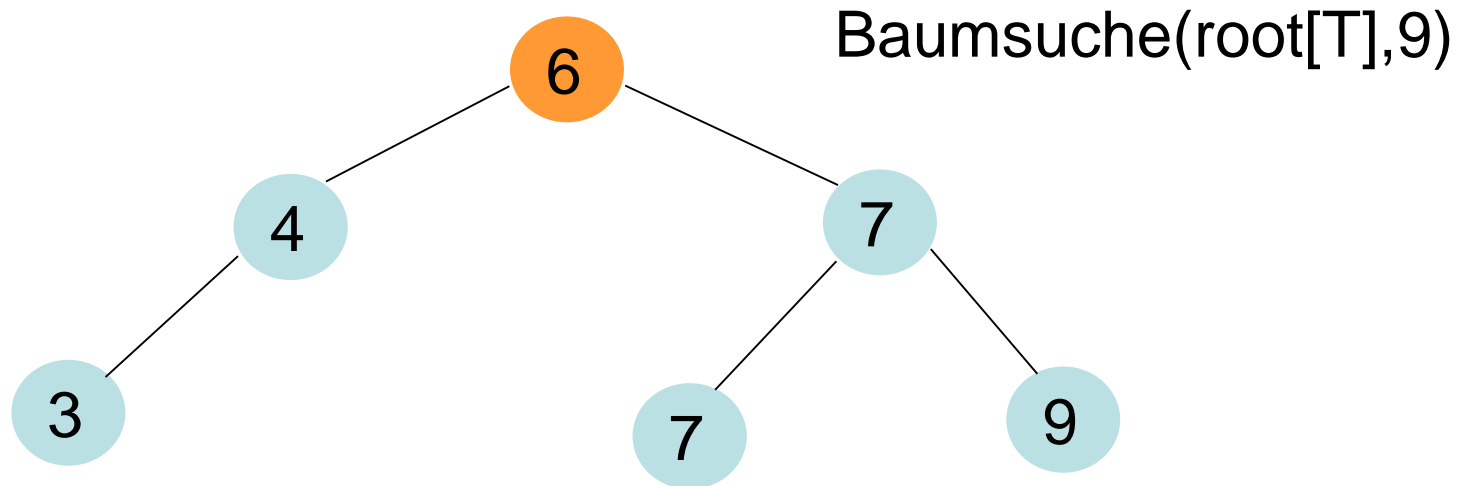


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if**  $x = \text{nil}$  **or**  $k = \text{key}[x]$  **then return**  $x$
2. **if**  $k < \text{key}[x]$  **then return**  $\text{Baumsuche}(\text{lc}[x], k)$
3. **else**  $\text{return Baumsuche}(\text{rc}[x], k)$

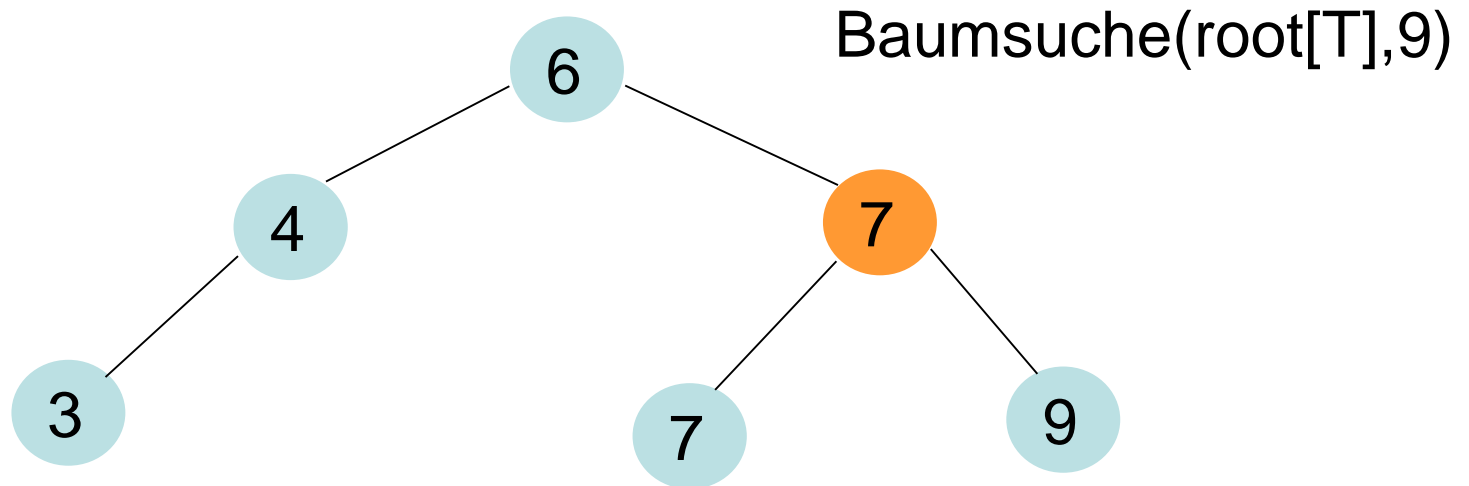


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if**  $x = \text{nil}$  **or**  $k = \text{key}[x]$  **then return**  $x$
2. **if**  $k < \text{key}[x]$  **then return**  $\text{Baumsuche}(\text{lc}[x], k)$
3. **else**  $\text{return Baumsuche}(\text{rc}[x], k)$

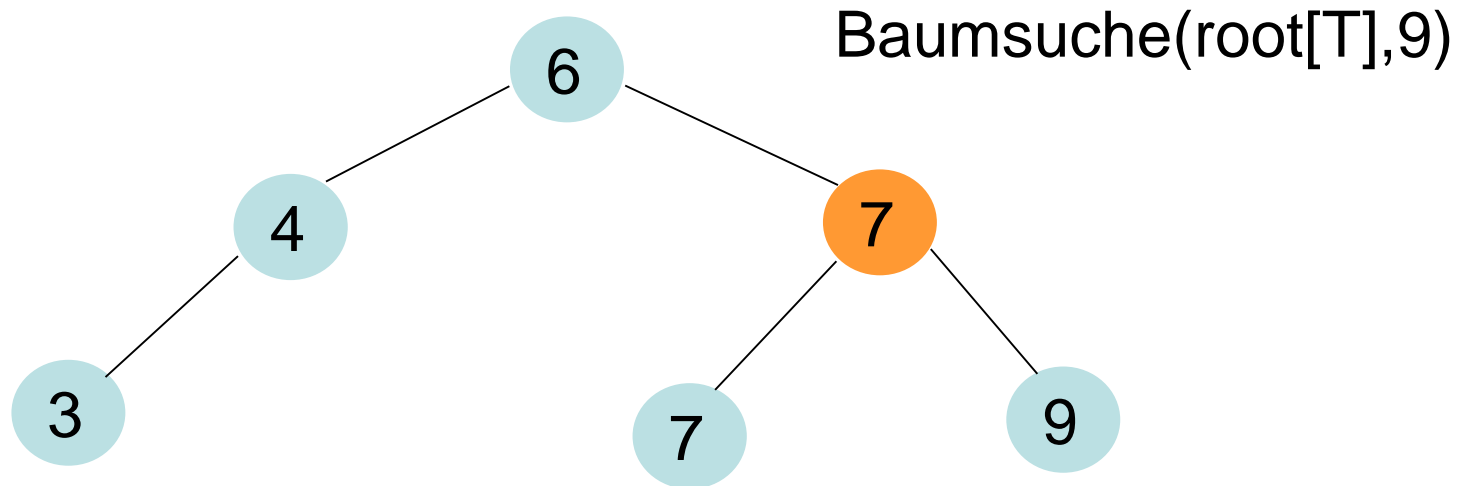


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if  $x = \text{nil}$  or  $k = \text{key}[x]$  then return  $x$**
2. **if  $k < \text{key}[x]$  then return  $\text{Baumsuche}(\text{lc}[x], k)$**
3. **else return  $\text{Baumsuche}(\text{rc}[x], k)$**



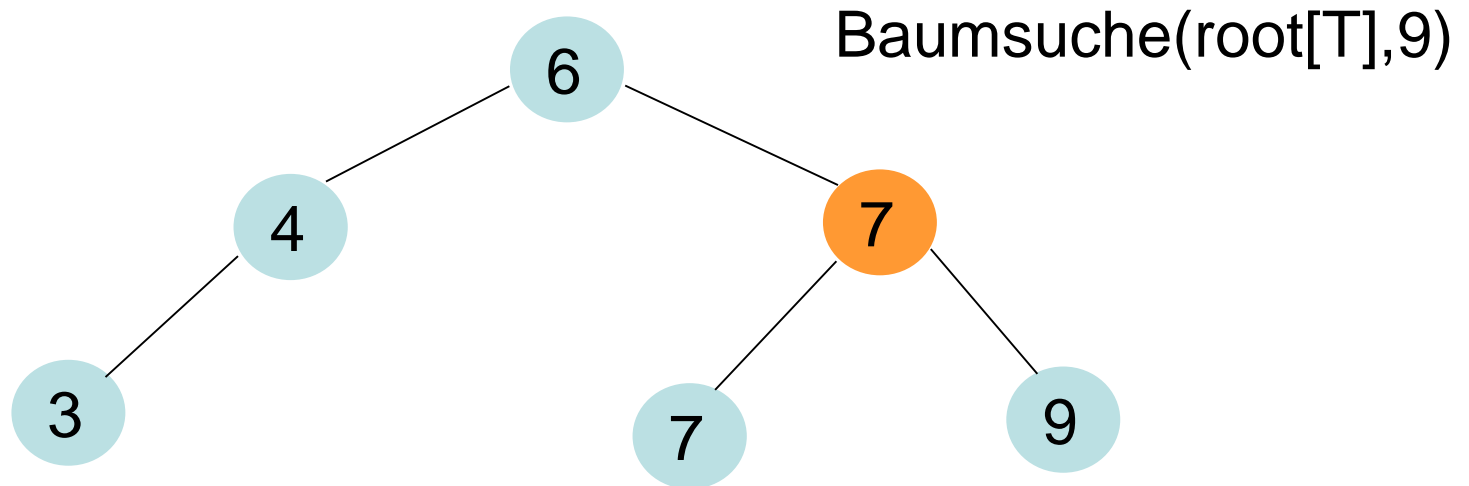


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if**  $x = \text{nil}$  **or**  $k = \text{key}[x]$  **then return**  $x$
2. **if**  $k < \text{key}[x]$  **then return**  $\text{Baumsuche}(\text{lc}[x], k)$
3. **else** **return**  $\text{Baumsuche}(\text{rc}[x], k)$

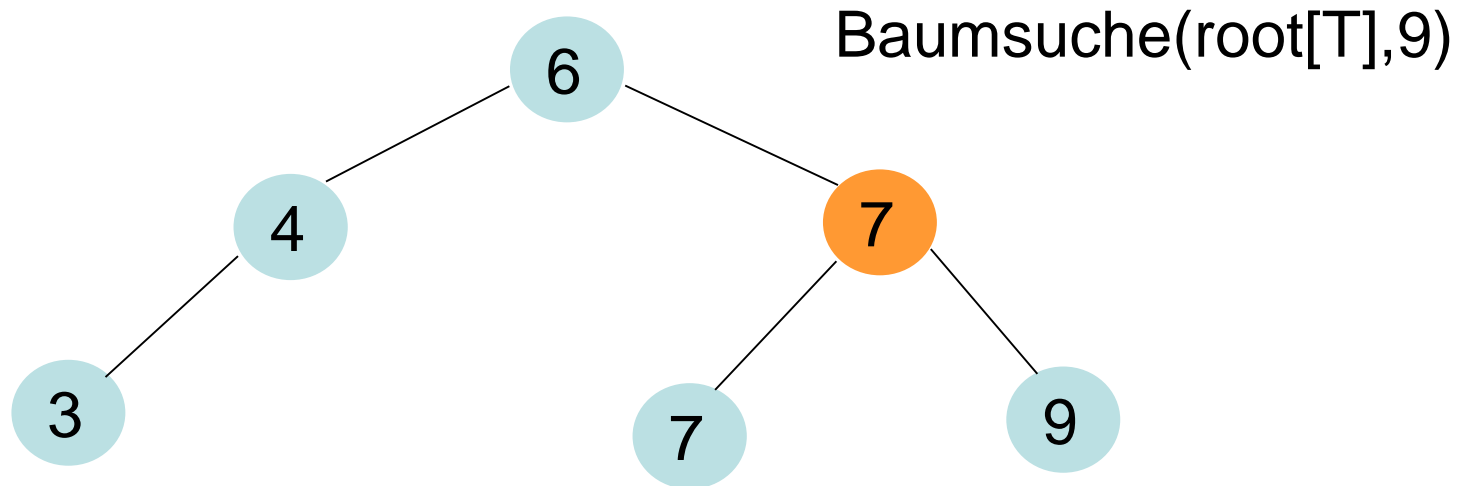


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if**  $x = \text{nil}$  **or**  $k = \text{key}[x]$  **then return**  $x$
2. **if**  $k < \text{key}[x]$  **then return**  $\text{Baumsuche}(\text{lc}[x], k)$
3. **else**  $\text{return Baumsuche}(\text{rc}[x], k)$



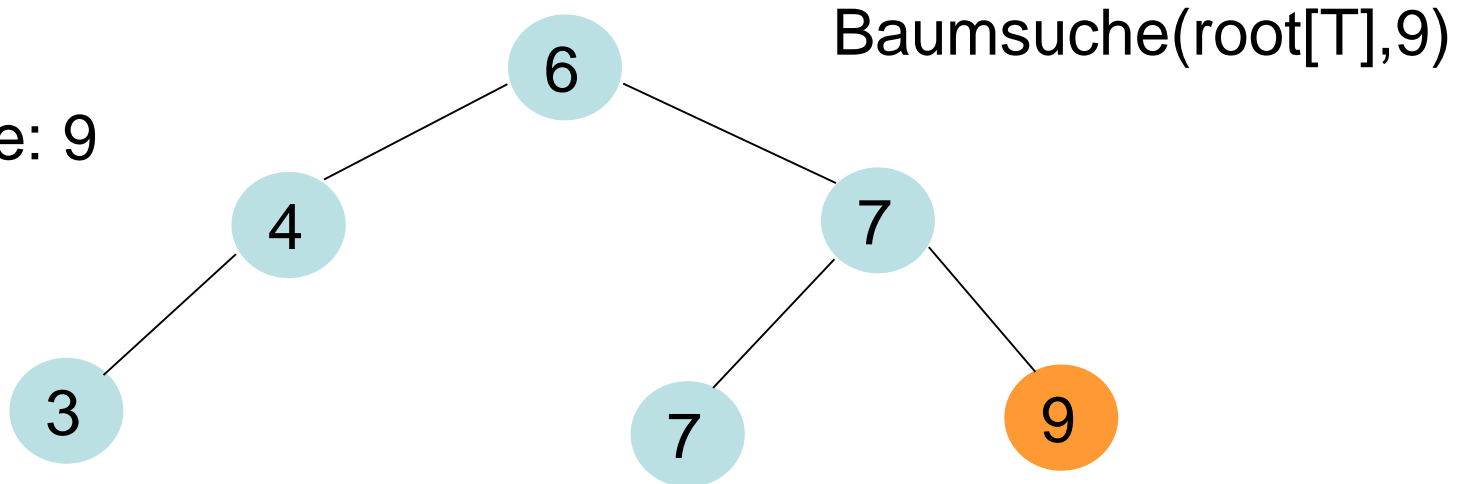
# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if** x=nil or k=key[x] **then return** x
2. **if** k<key[x] **then return** Baumsuche(lc[x],k)
3. **else** return Baumsuche(rc[x],k)

Ausgabe: 9

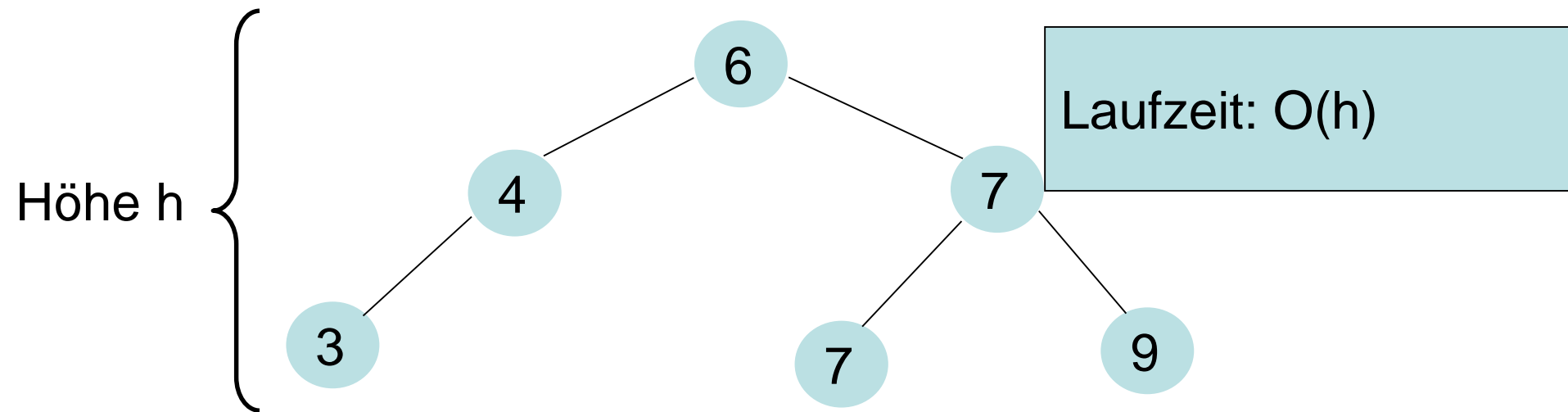


# Binäre Suchbäume

---

Baumsuche(x,k)

1. **if**  $x = \text{nil}$  **or**  $k = \text{key}[x]$  **then return**  $x$
2. **if**  $k < \text{key}[x]$  **then return**  $\text{Baumsuche}(\text{lc}[x], k)$
3. **else** **return**  $\text{Baumsuche}(\text{rc}[x], k)$



# Binäre Suchbäume

---

## Weitere Operationen in Binärbäumen

- Minimum/Maximumsuche
- Nachfolger/Vorgänger

## Minimum- und Maximumsuche:

- Suchbaumeigenschaft:  
Alle Knoten im rechten Unterbaum eines Knotens  $x$  sind größer gleich  $\text{key}[x]$
- Alle Knoten im linken Unterbaum von  $x$  sind  $\leq \text{key}[x]$

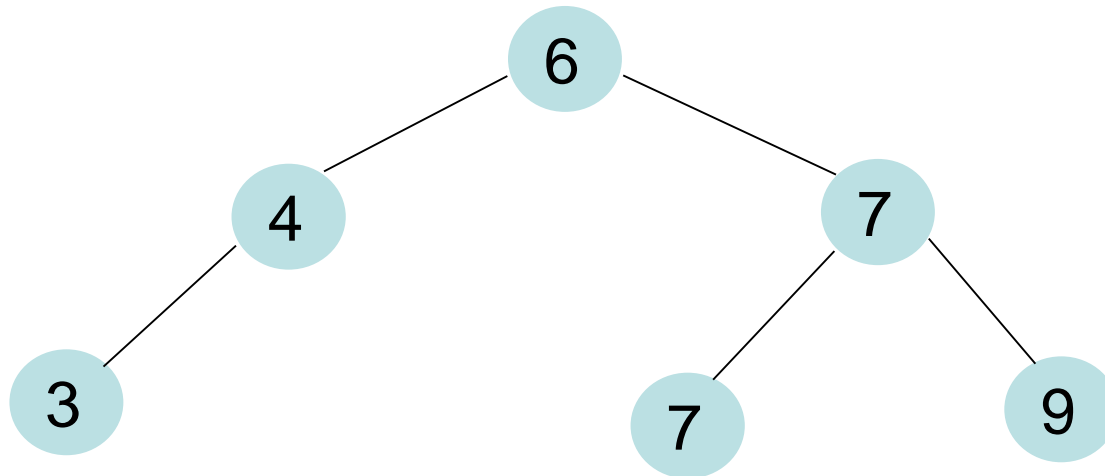
# Binäre Suchbäume

---

MinimumSuche(x)

1. **while**  $lc[x] \neq \text{nil}$  **do**  $x \leftarrow lc[x]$

2. **return** x



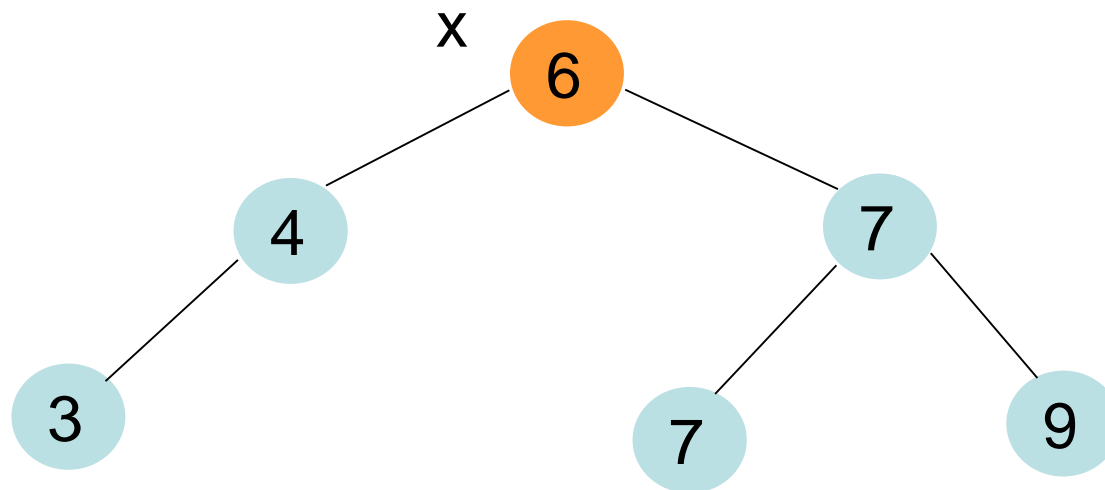
# Binäre Suchbäume

---

MinimumSuche(x)

1. **while**  $lc[x] \neq \text{nil}$  **do**  $x \leftarrow lc[x]$

2. **return** x

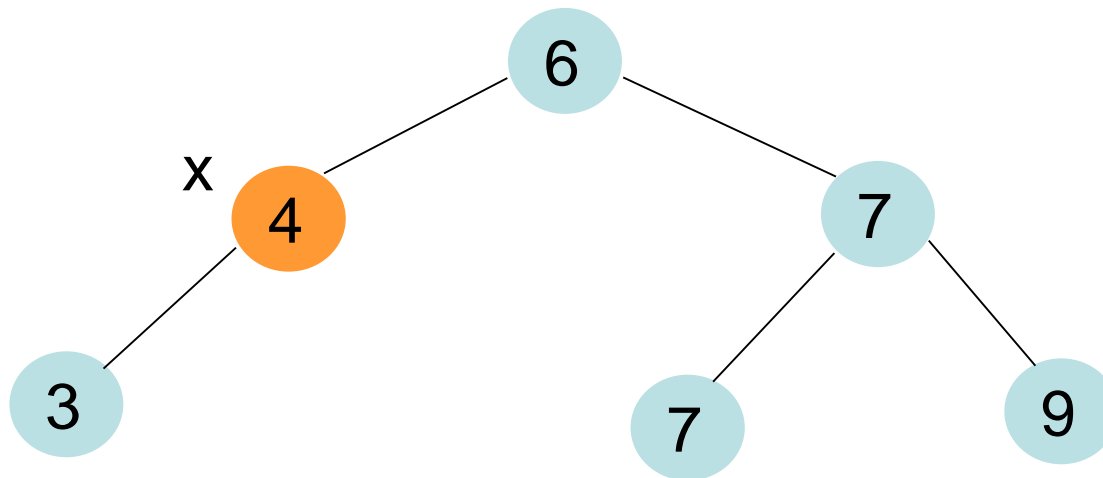


# Binäre Suchbäume

---

MinimumSuche(x)

1. **while**  $lc[x] \neq \text{nil}$  **do**  $x \leftarrow lc[x]$
2. **return**  $x$



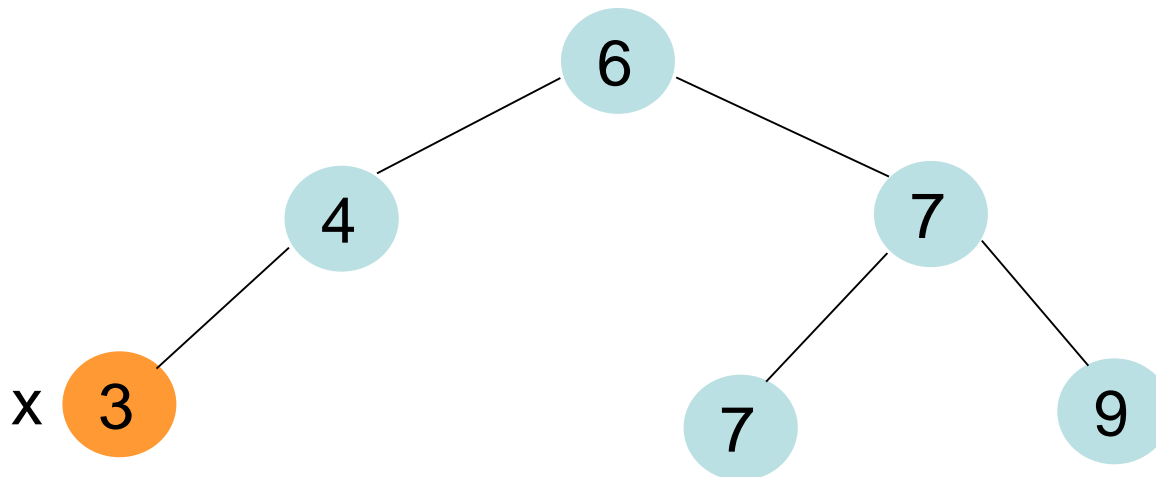


# Binäre Suchbäume

---

MinimumSuche(x)

1. **while**  $lc[x] \neq \text{nil}$  **do**  $x \leftarrow lc[x]$
2. **return**  $x$



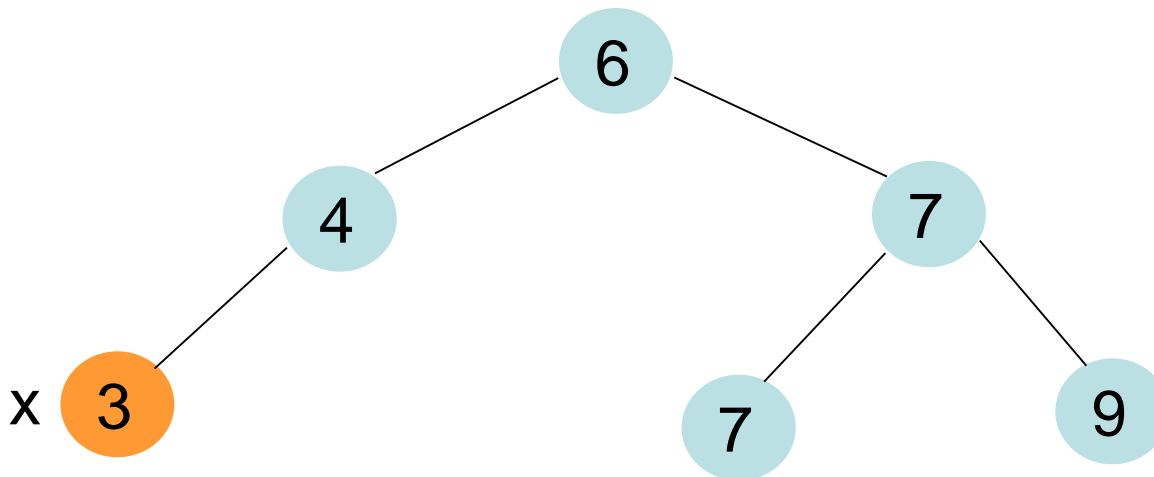
# Binäre Suchbäume

---

MinimumSuche(x)

1. **while**  $lc[x] \neq \text{nil}$  **do**  $x \leftarrow lc[x]$
2. **return** x

Laufzeit  $O(h)$



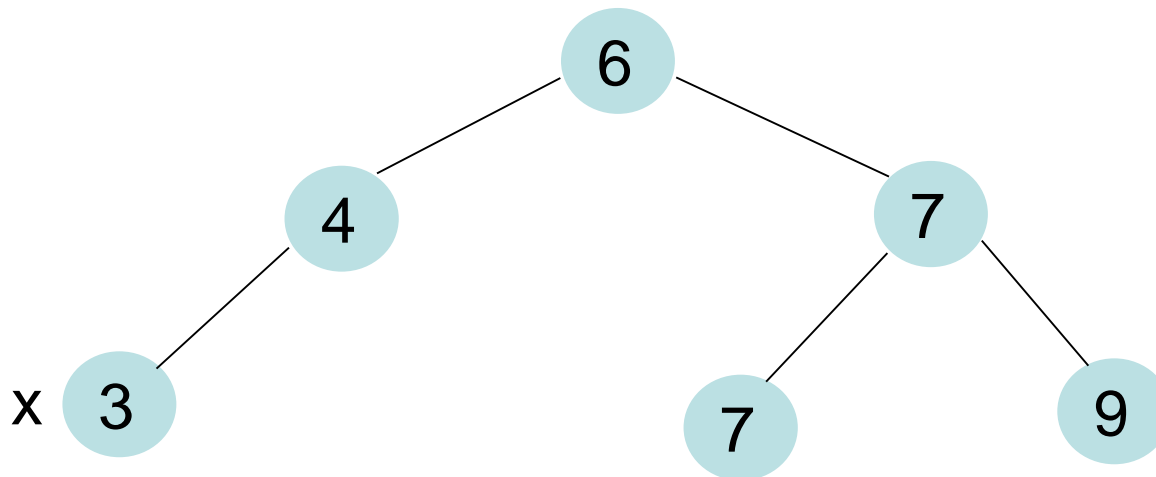
# Binäre Suchbäume

---

MaximumSuche(x)

1. **while** rc[x]≠nil **do** x ← rc[x]

2. **return** x



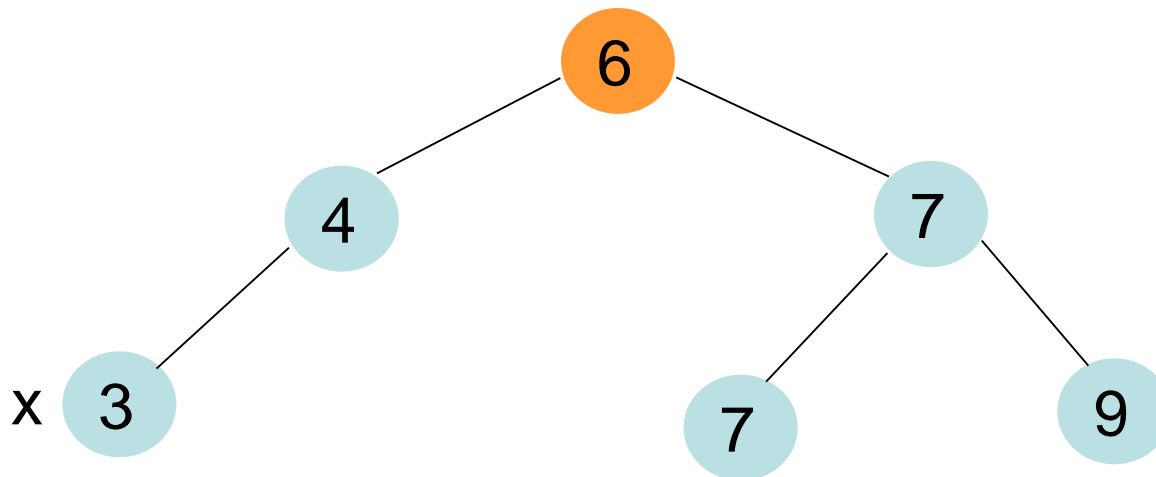
# Binäre Suchbäume

---

MaximumSuche(x)

1. **while** rc[x]≠nil **do** x ← rc[x]

2. **return** x

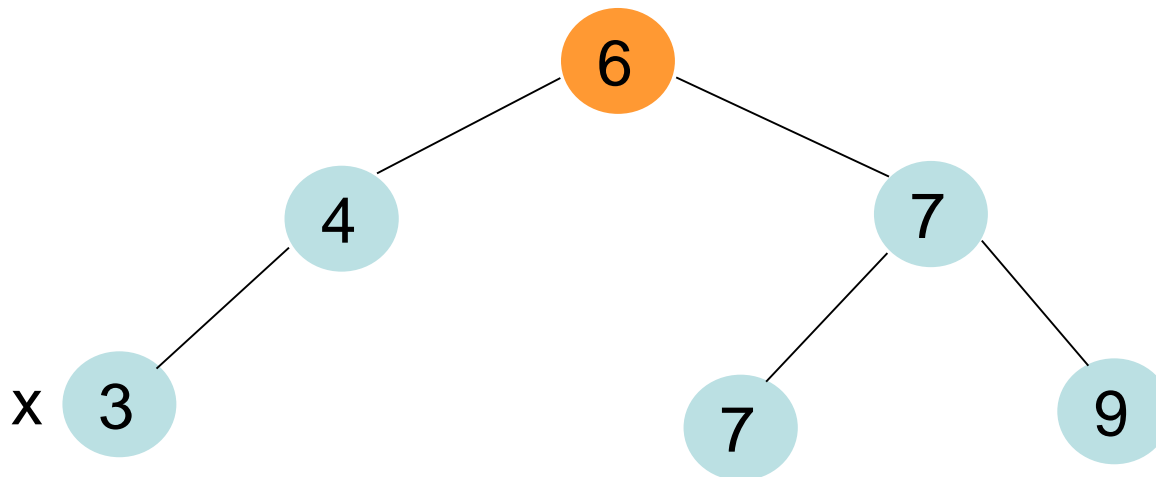


# Binäre Suchbäume

---

MaximumSuche(x)

1. **while** rc[x]≠nil **do** x ← rc[x]
2. **return** x

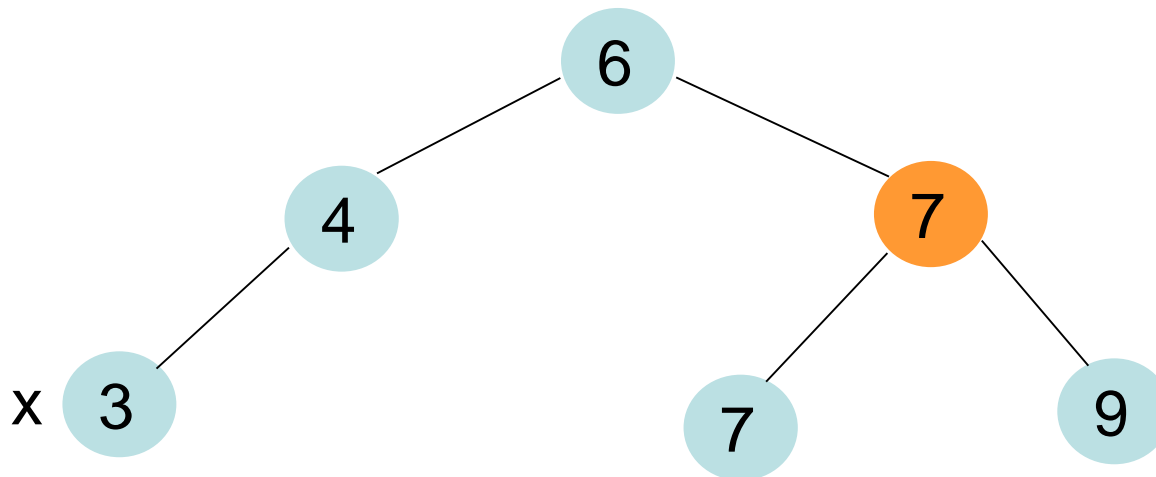


# Binäre Suchbäume

---

MaximumSuche(x)

1. **while** rc[x]≠nil **do** x ← rc[x]
2. **return** x

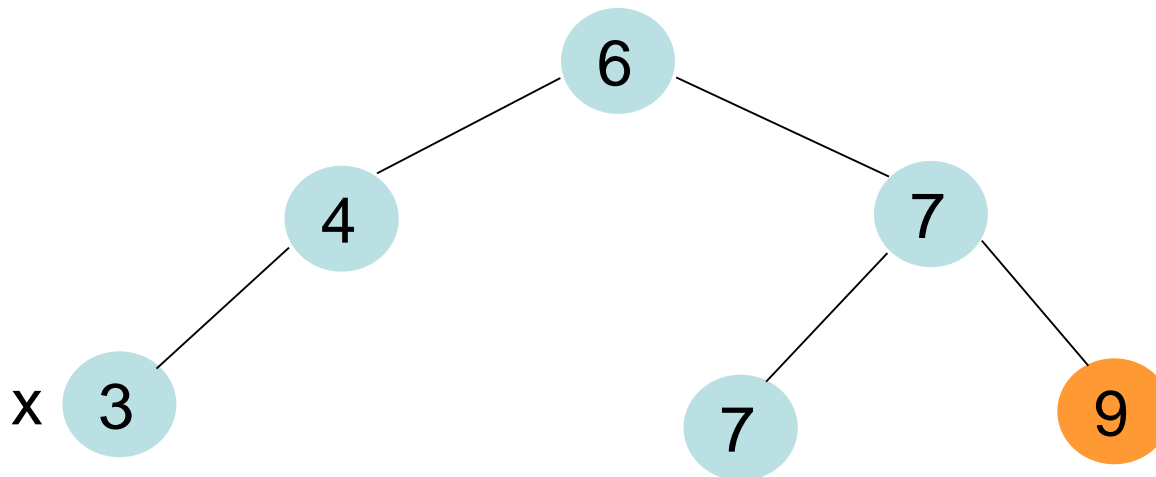


# Binäre Suchbäume

---

MaximumSuche(x)

1. **while** rc[x]≠nil **do** x ← rc[x]
2. **return** x



# Binäre Suchbäume

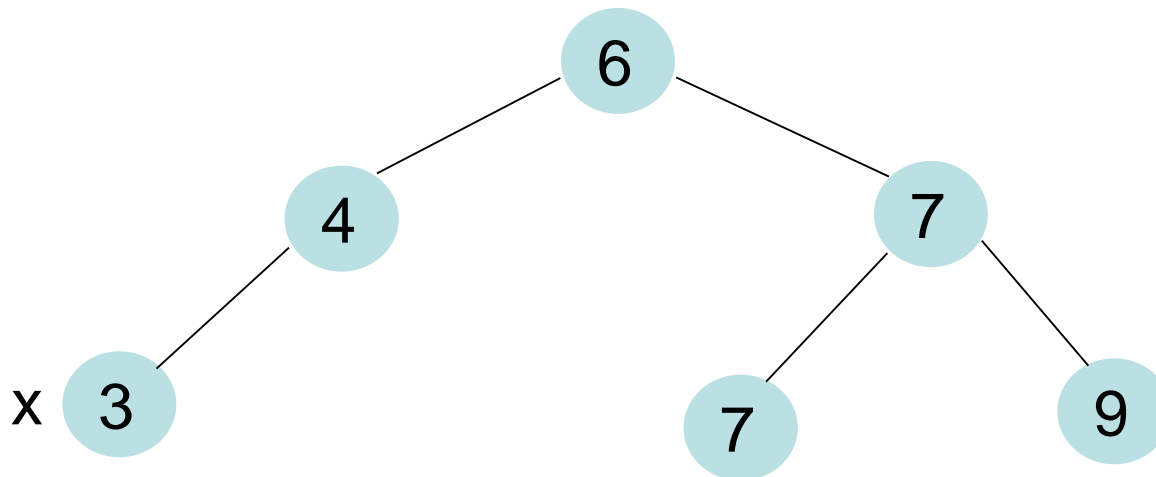
---

MaximumSuche(x)

1. **while** rc[x]≠nil **do** x ← rc[x]

2. **return** x

Laufzeit  $O(h)$





# Binäre Suchbäume

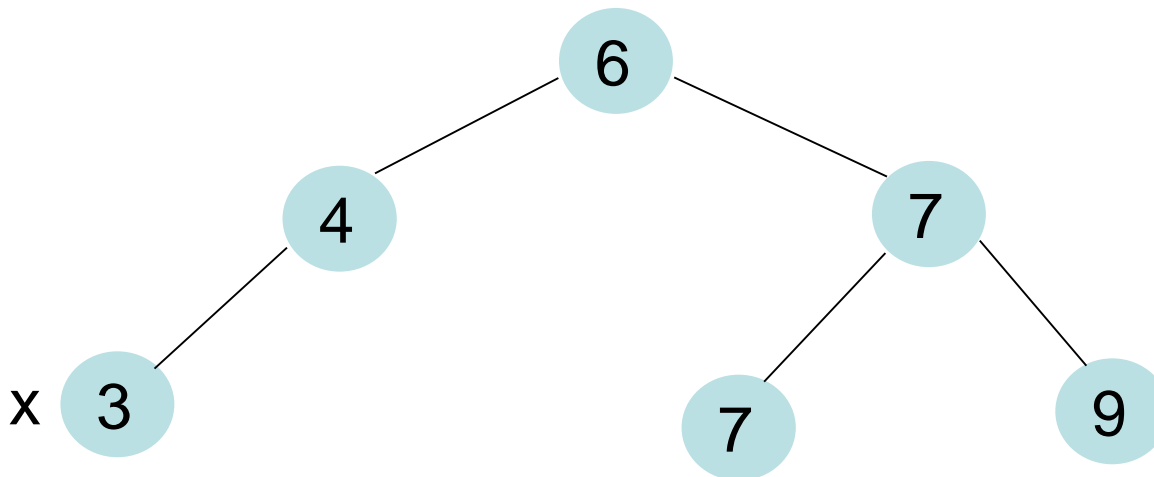
---

MaximumSuche(x)

1. **while** rc[x]≠nil **do** x ← rc[x]

2. **return** x

Laufzeit  $O(h)$

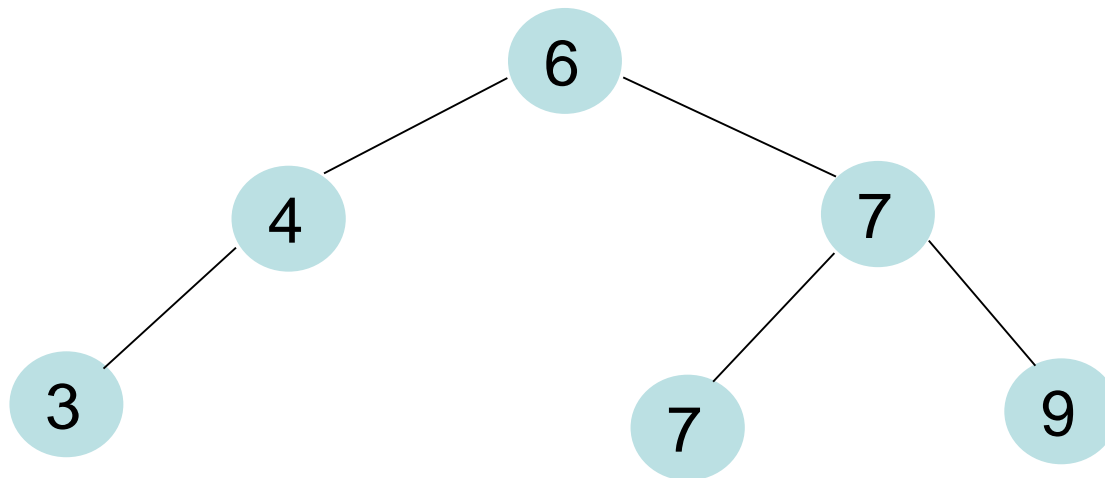


# Binäre Suchbäume

---

## Nachfolgersuche:

- Nachfolger bzgl. der Ausgabe von Inorder-Tree-Walk
- Wenn alle Schlüssel unterschiedlich, dann ist das der nächstgrößere Schlüssel

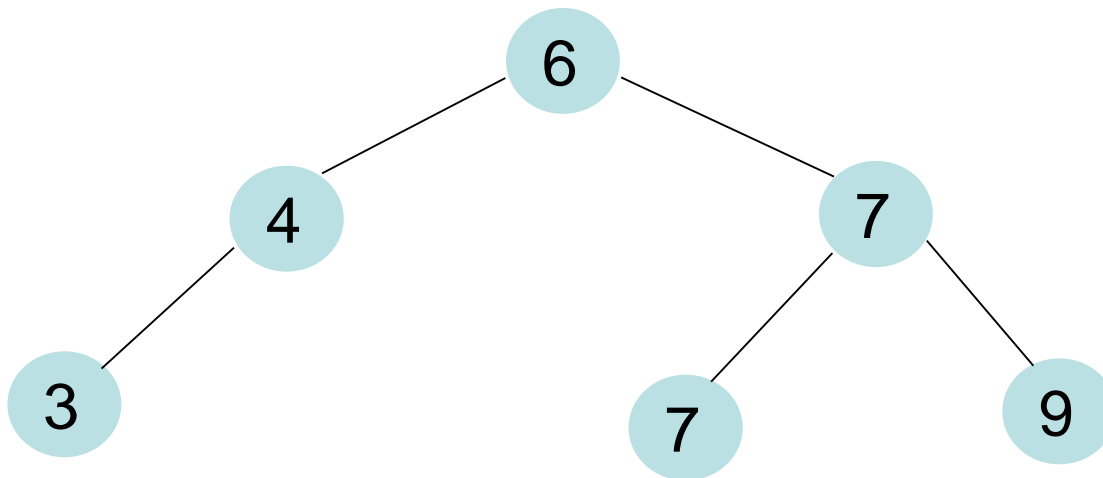


# Binäre Suchbäume

---

## Nachfolgersuche:

- Fall 1 (rechter Unterbaum von x nicht leer):  
Dann ist das Minimum im rechten Unterbaum der  
Nachfolger von x

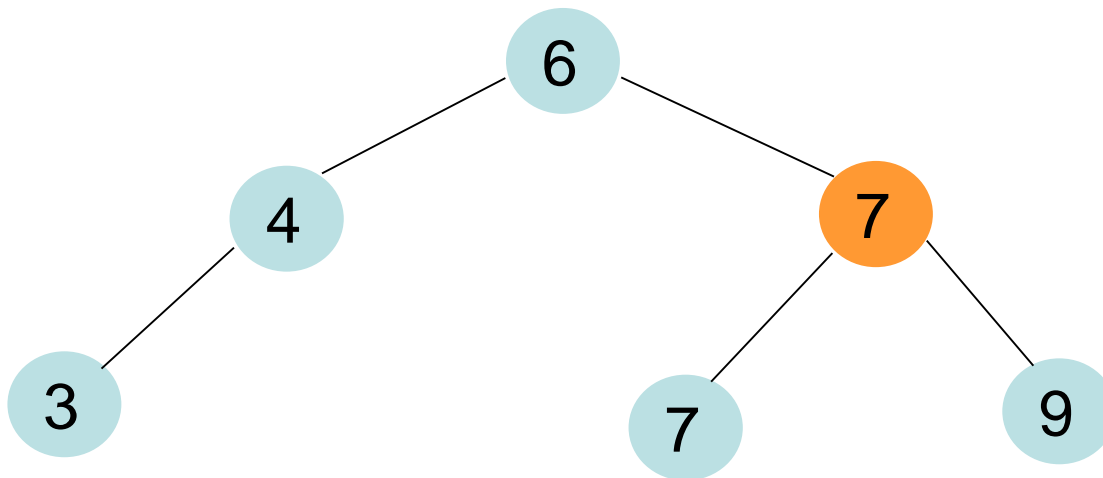


# Binäre Suchbäume

---

## Nachfolgersuche:

- Fall 1 (rechter Unterbaum von x nicht leer):  
Dann ist das Minimum im rechten Unterbaum der  
Nachfolger von x

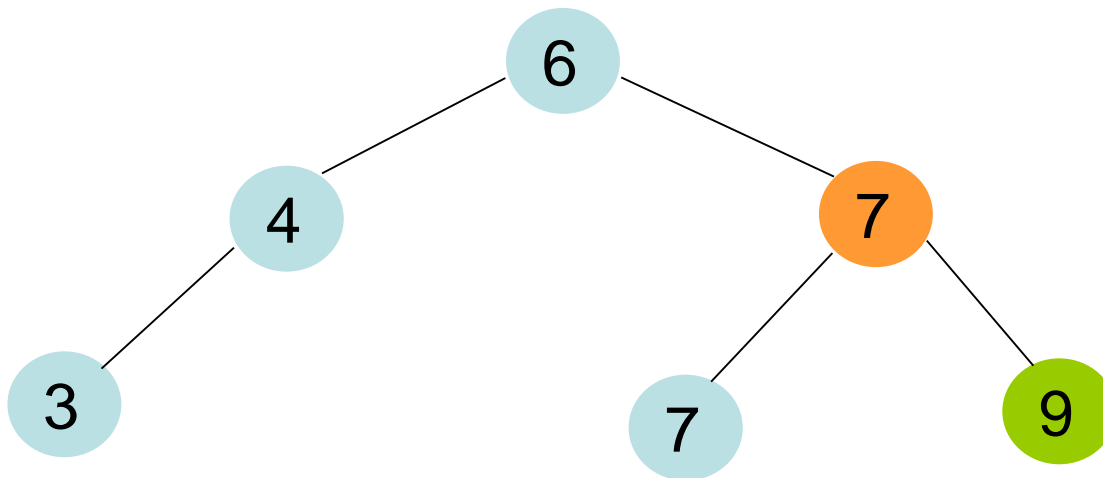


# Binäre Suchbäume

---

## Nachfolgersuche:

- Fall 1 (rechter Unterbaum von x nicht leer):  
Dann ist das Minimum im rechten Unterbaum der  
Nachfolger von x

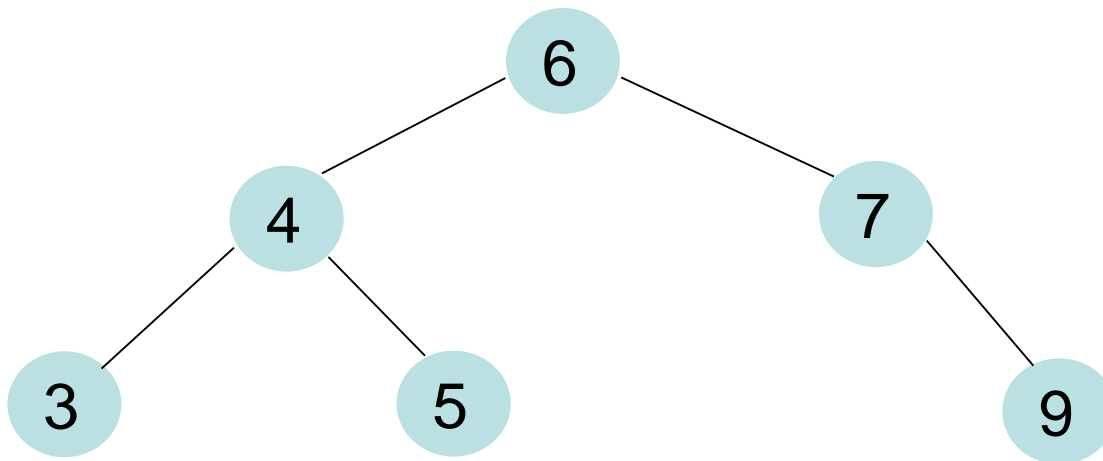


# Binäre Suchbäume

---

## Nachfolgersuche:

- Fall 2 (rechter Unterbaum von x leer und x hat Nachfolger y):  
Dann ist y der niedrigste Vorgänger (in Richtung Wurzel) von x, dessen linkes Kind ebenfalls Vorgänger von x ist

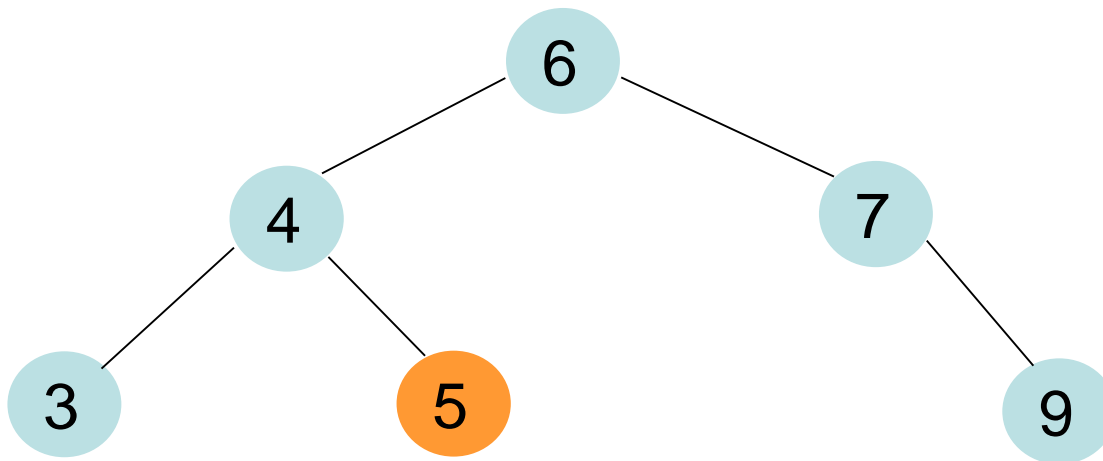


# Binäre Suchbäume

---

## Nachfolgersuche:

- Fall 2 (rechter Unterbaum von  $x$  leer und  $x$  hat Nachfolger  $y$ ):  
Dann ist  $y$  der niedrigste Vorgänger von  $x$ , dessen linkes Kind ebenfalls Vorgänger von  $x$  ist

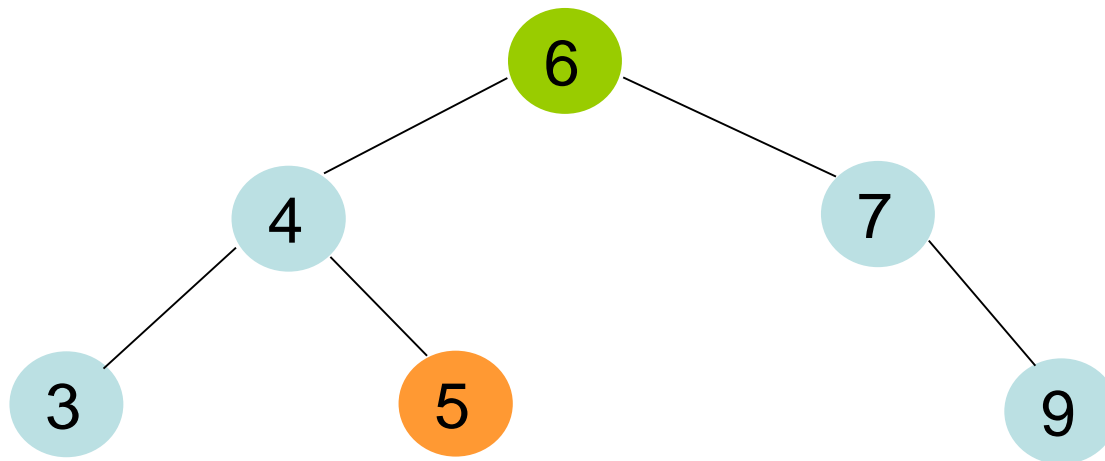


# Binäre Suchbäume

---

## Nachfolgersuche:

- Fall 2 (rechter Unterbaum von x leer und x hat Nachfolger y):  
Dann ist y der niedrigste Vorgänger von x, dessen linkes Kind ebenfalls Vorgänger von x ist



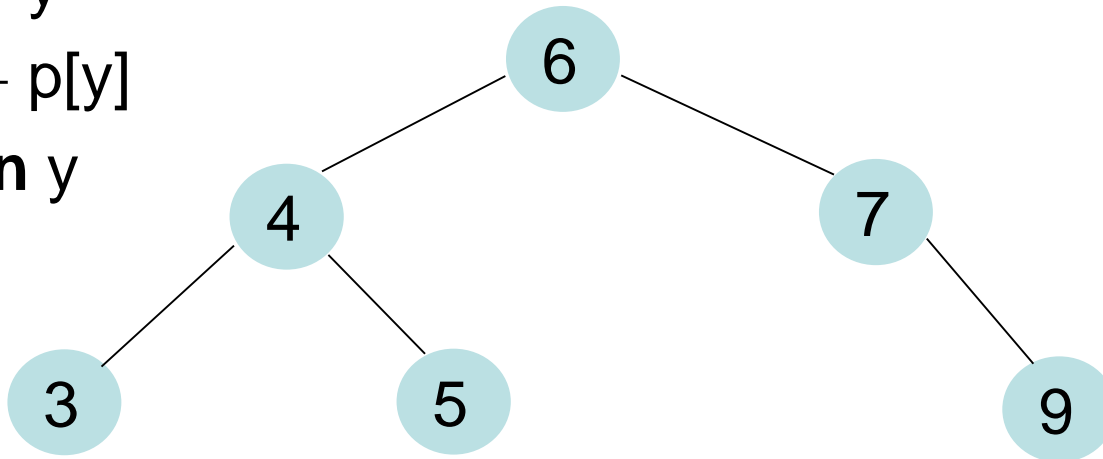


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while** y  $\neq$  nil and x=rc[y] **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y

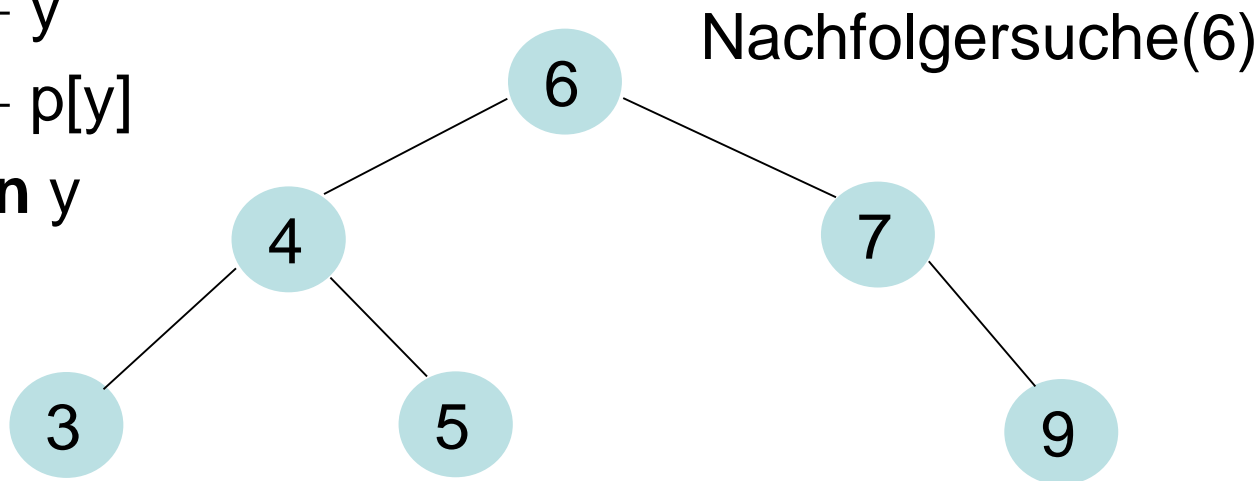


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while** y  $\neq$  nil and x=rc[y] **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y



# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])

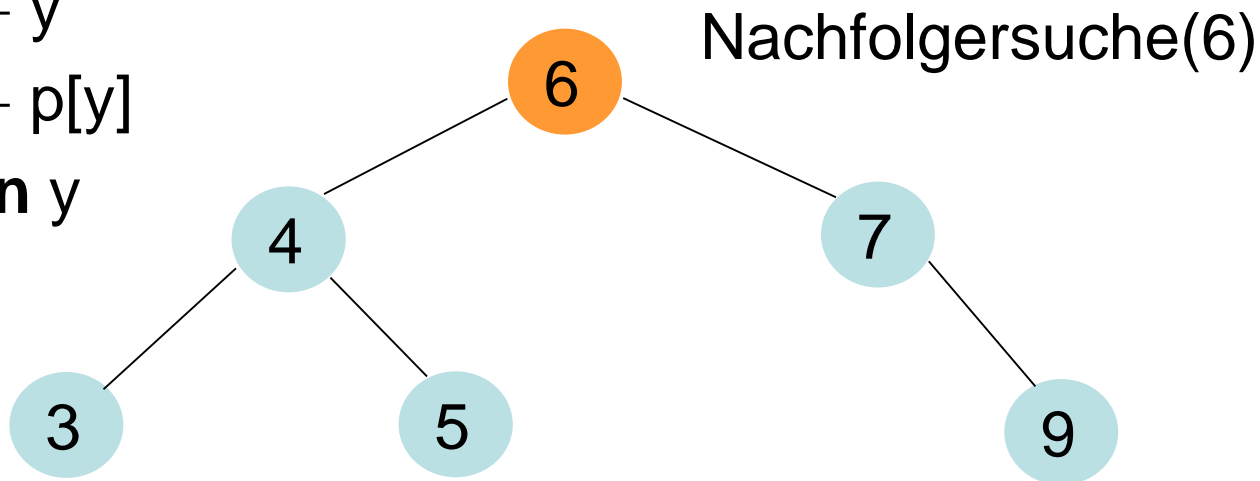
2.  $y \leftarrow p[x]$

3. **while**  $y \neq$  nil and  $x=rc[y]$  **do**

4.  $x \leftarrow y$

5.  $y \leftarrow p[y]$

6. **return** y



# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])

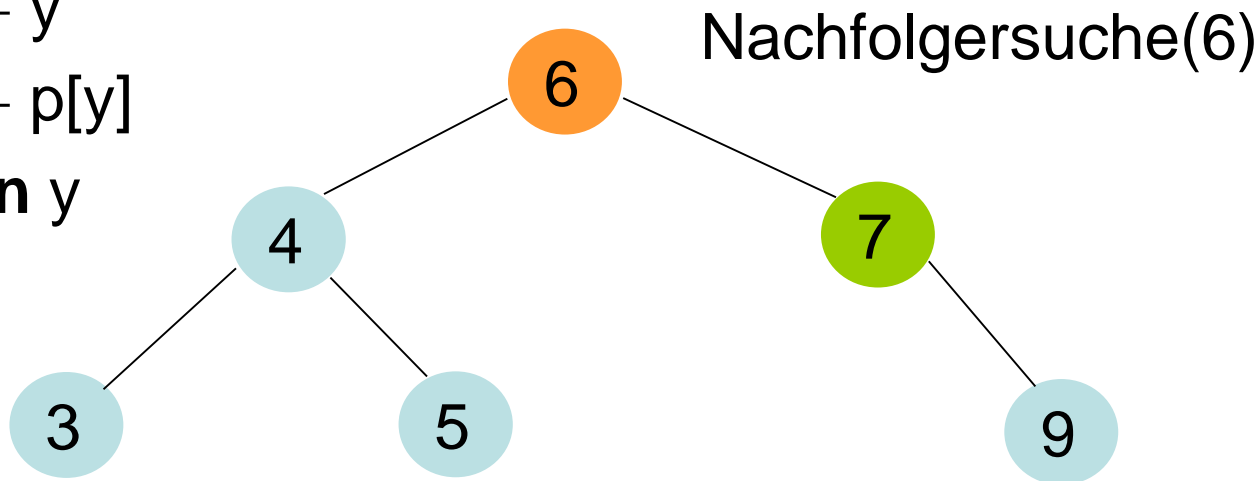
2.  $y \leftarrow p[x]$

3. **while**  $y \neq$  nil and  $x = \text{rc}[y]$  **do**

4.  $x \leftarrow y$

5.  $y \leftarrow p[y]$

6. **return** y

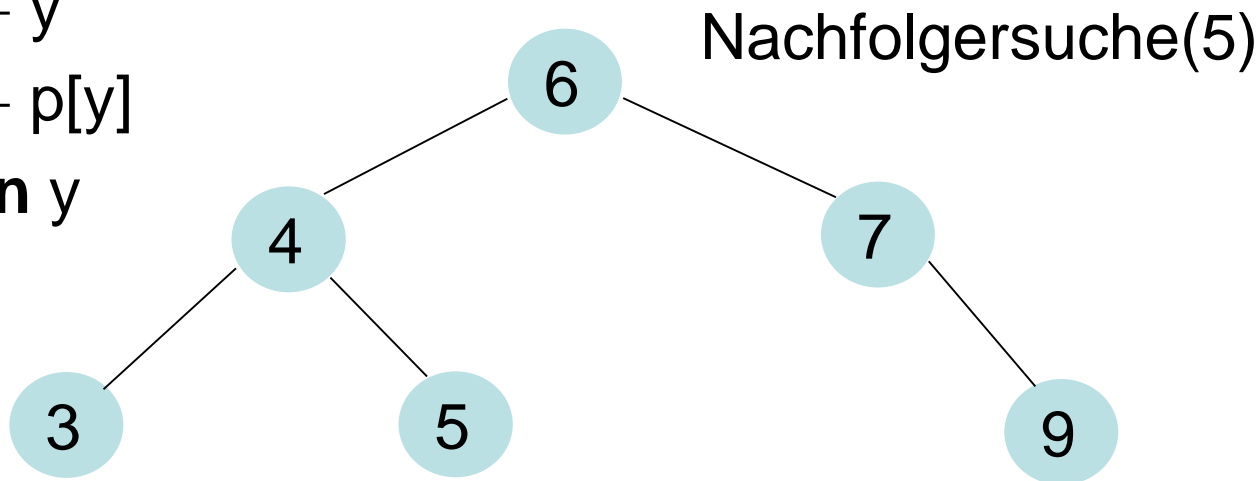


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while**  $y \neq$  nil and  $x=rc[y]$  **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y

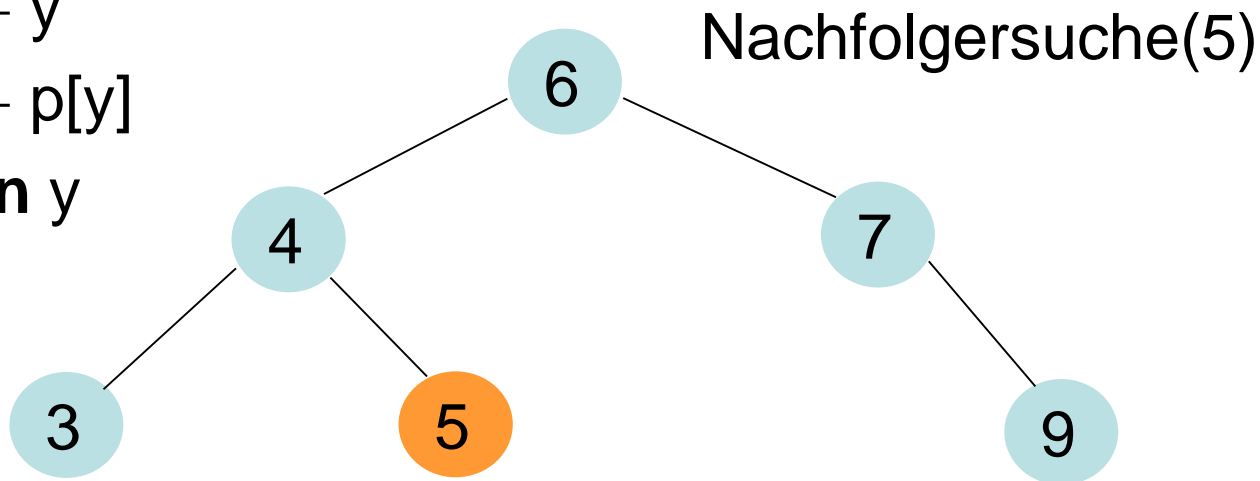


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while** y  $\neq$  nil and x=rc[y] **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y

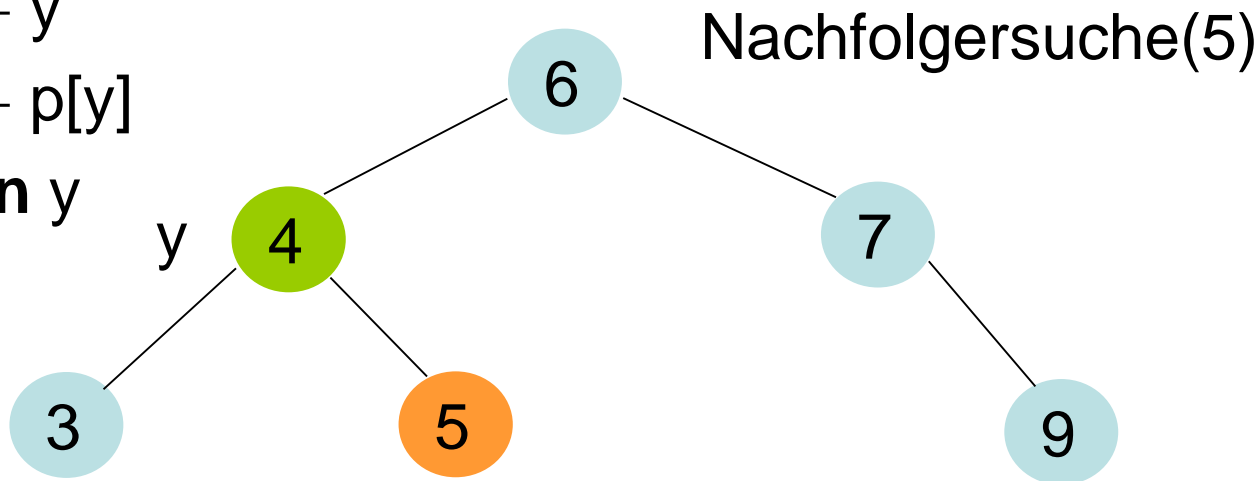


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while** y  $\neq$  nil and x=rc[y] **do**
4.   x  $\leftarrow$  y
5.   y  $\leftarrow$  p[y]
6. **return** y

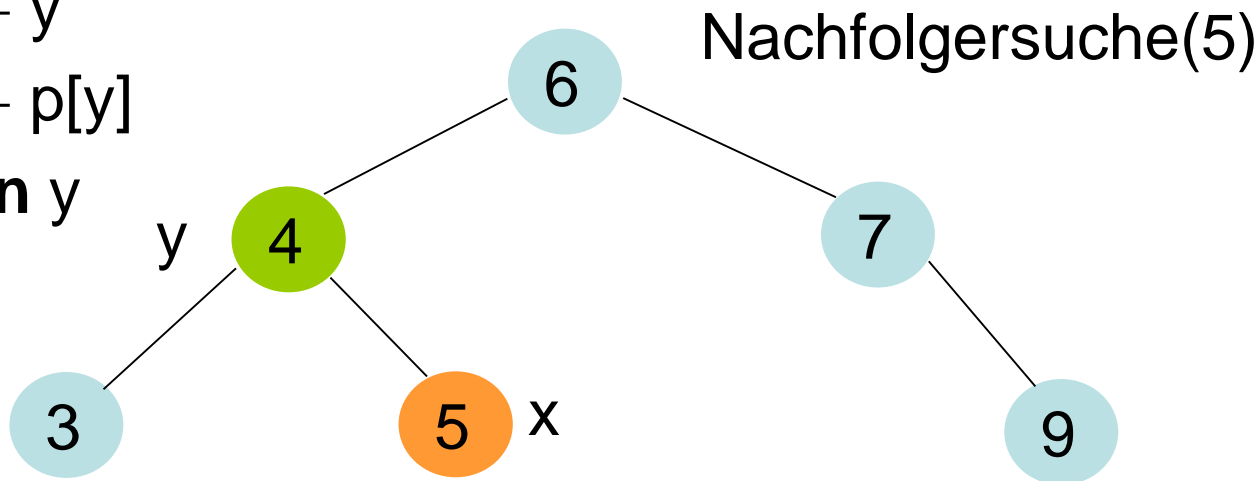


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while**  $y \neq$  nil and  $x=rc[y]$  **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y



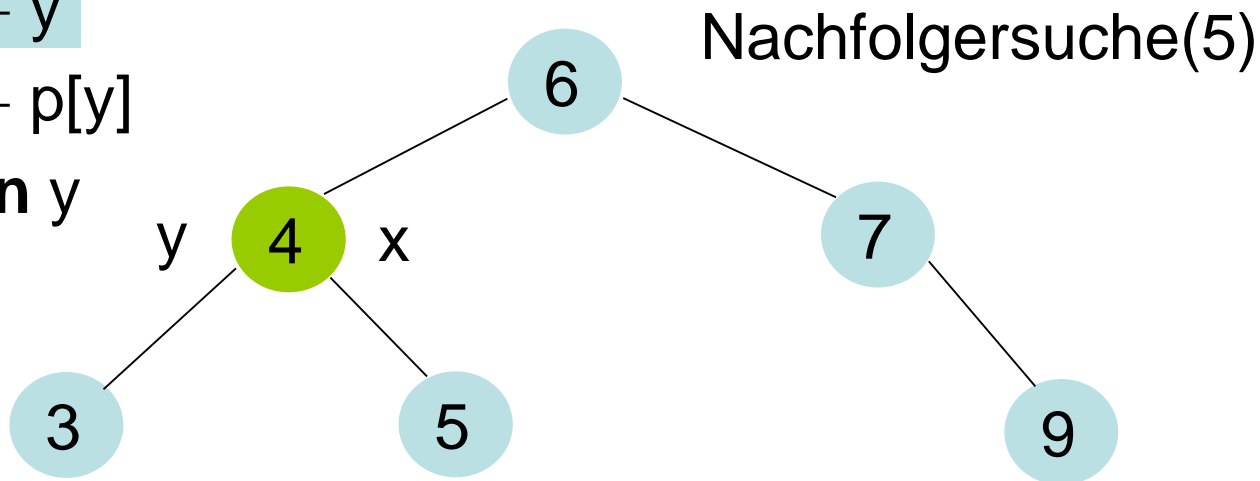


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while**  $y \neq$  nil and  $x = \text{rc}[y]$  **do**
4.  $x \leftarrow y$
5.  $y \leftarrow p[y]$
6. **return** y

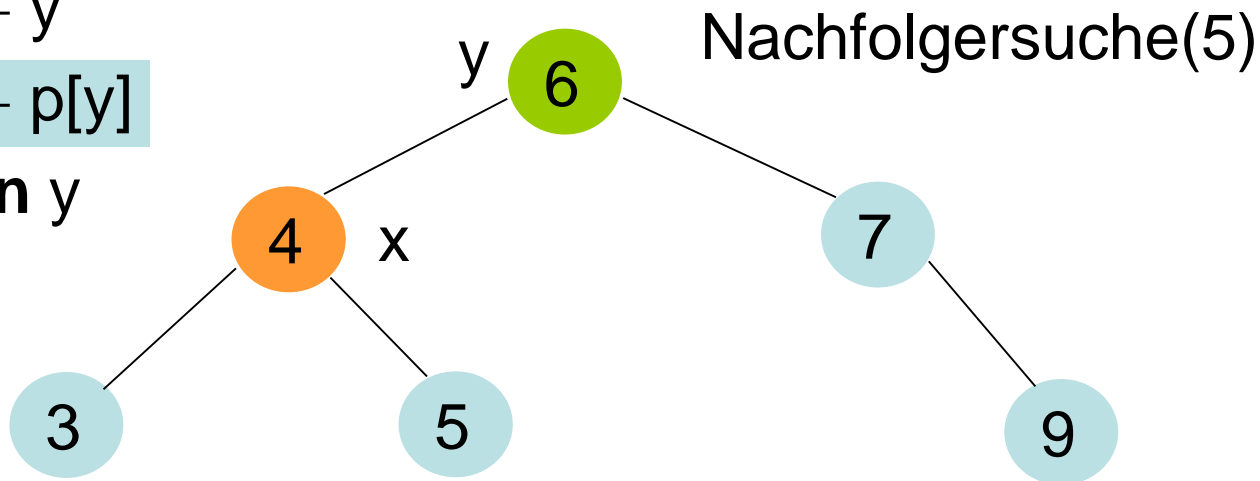


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while** y  $\neq$  nil and x=rc[y] **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y

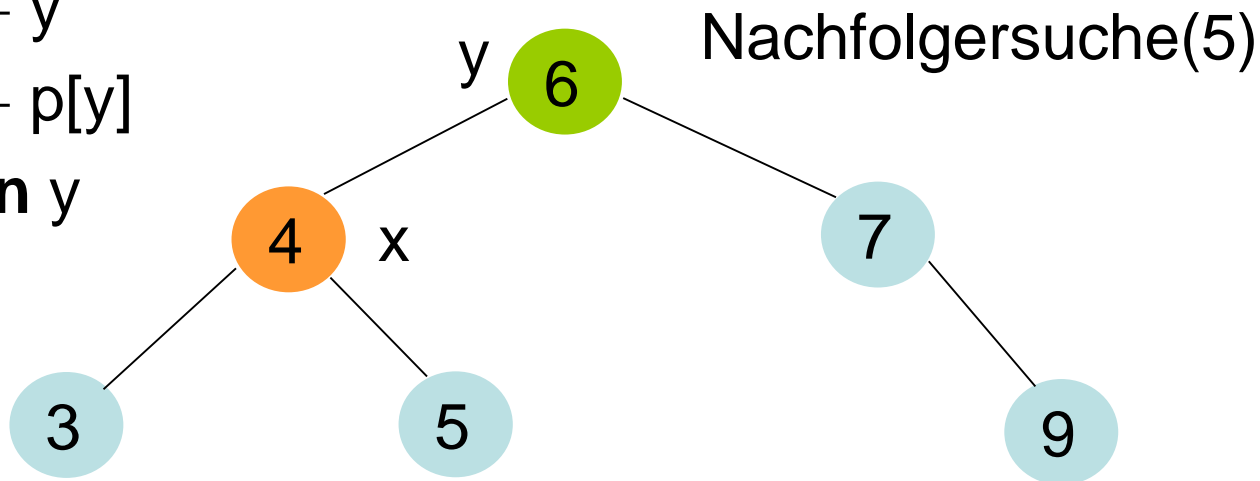


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while**  $y \neq$  nil and  $x = \text{rc}[y]$  **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y

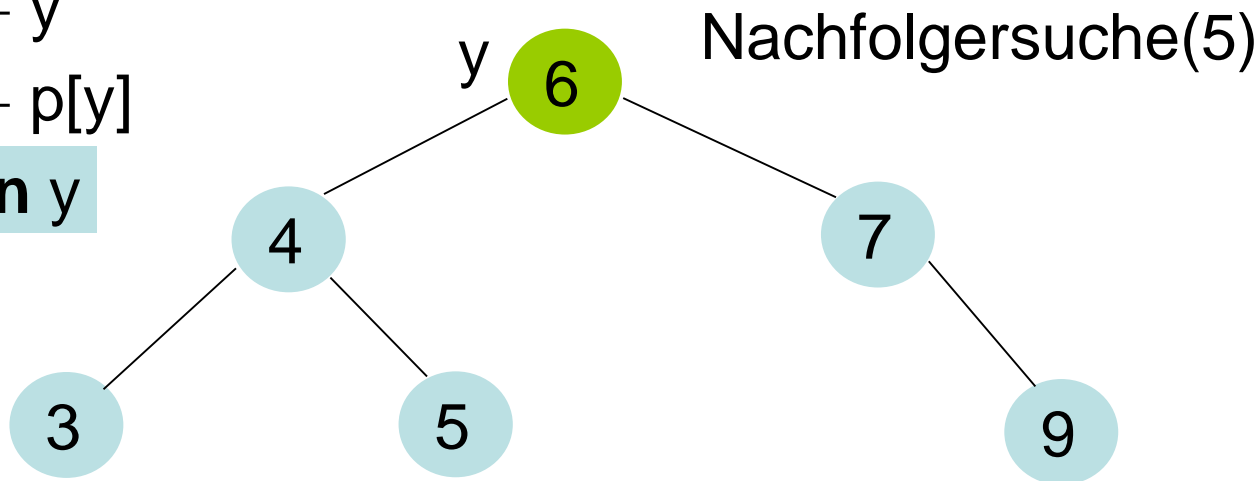


# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while**  $y \neq$  nil and  $x=rc[y]$  **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y



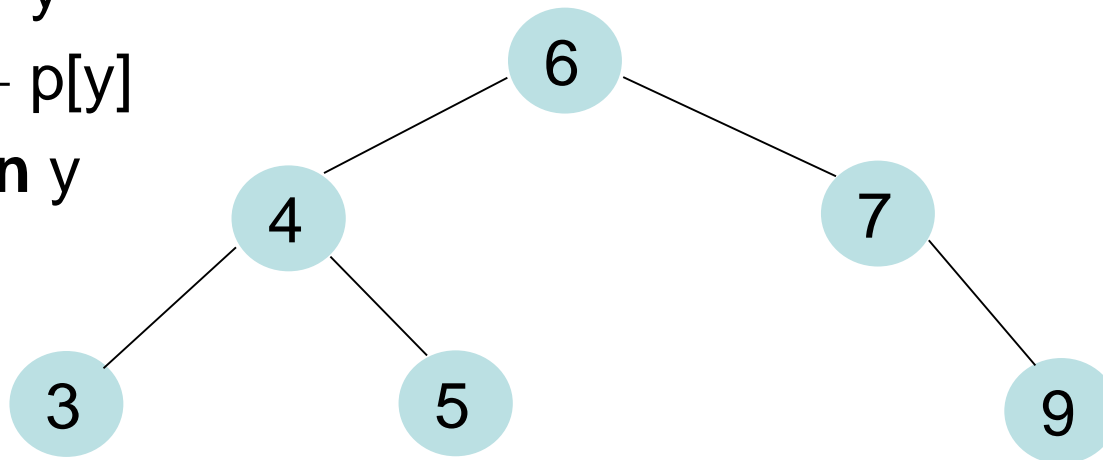
# Binäre Suchbäume

---

Nachfolgersuche(x)

1. **if** rc[x]  $\neq$  nil **then return** MinimumSuche(rc[x])
2.  $y \leftarrow p[x]$
3. **while**  $y \neq$  nil and  $x=rc[y]$  **do**
4.    $x \leftarrow y$
5.    $y \leftarrow p[y]$
6. **return** y

Laufzeit  $O(h)$

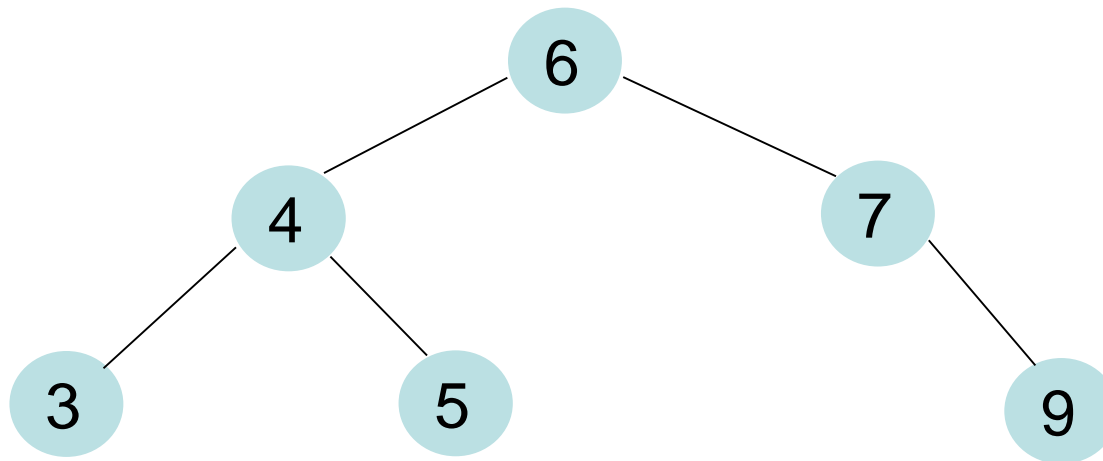


# Binäre Suchbäume

---

## Vorgängersuche:

- Symmetrisch zu Nachfolgersuche
- Daher ebenfalls  $O(h)$  Laufzeit



# Binäre Suchbäume

---

## Binäre Suchbäume:

- Aufzählen der Elemente mit Inorder-Tree-Walk in  $O(n)$  Zeit
- Suche in  $O(h)$  Zeit
- Minimum/Maximum in  $O(h)$  Zeit

## Dynamische Operationen?

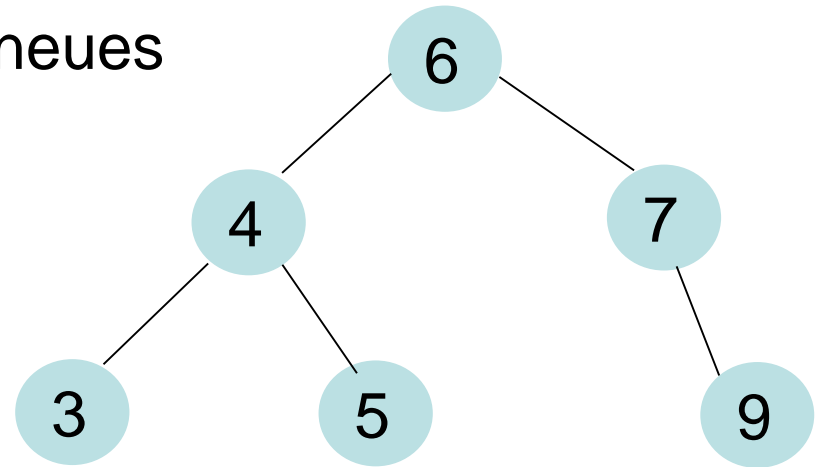
- Einfügen und Löschen
- Müssen Suchbaumeigenschaft aufrecht erhalten
- Auswirkung auf Höhe des Baums?

# Binäre Suchbäume

---

## Einfügen:

- Ähnlich wie Baumsuche: Finde geeigneten Knoten, an den neuer Knoten angehängt werden sollte
- Danach wird **nil**-Zeiger durch neues Element ersetzt



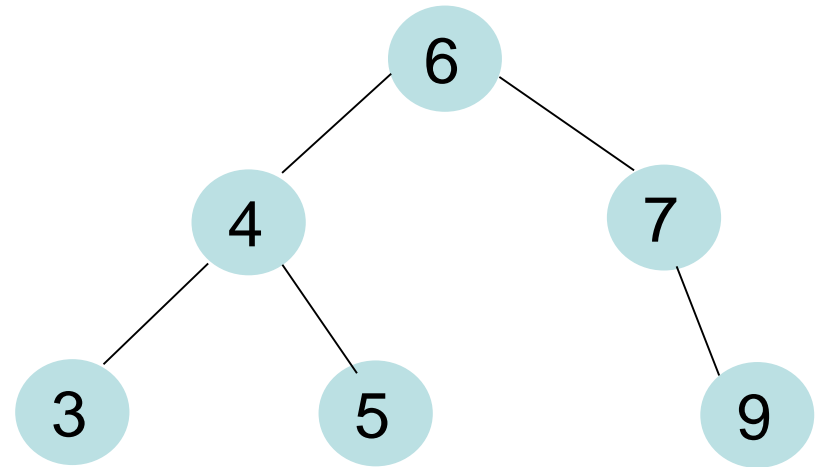


# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.    $y \leftarrow x$
4.   **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5.   **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9.   **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

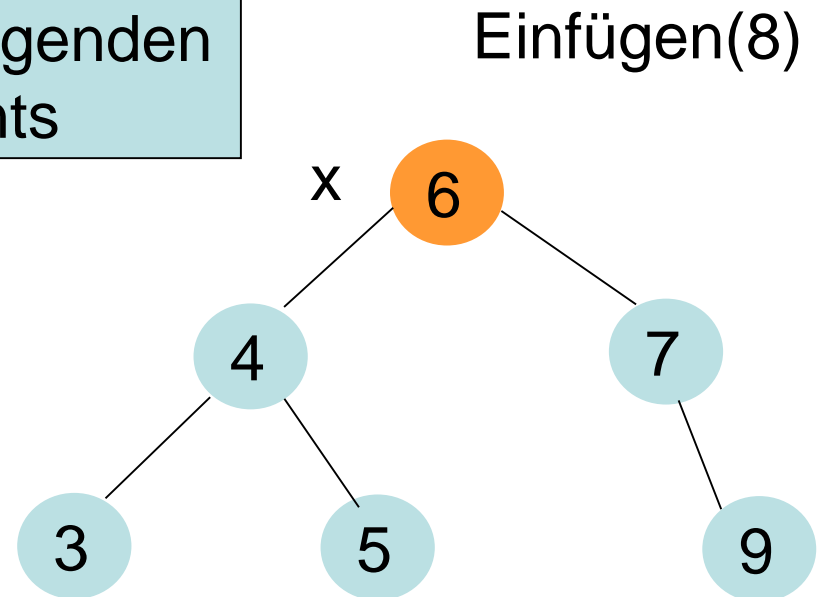


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}; x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

y ist Vater des einzufügenden Elements



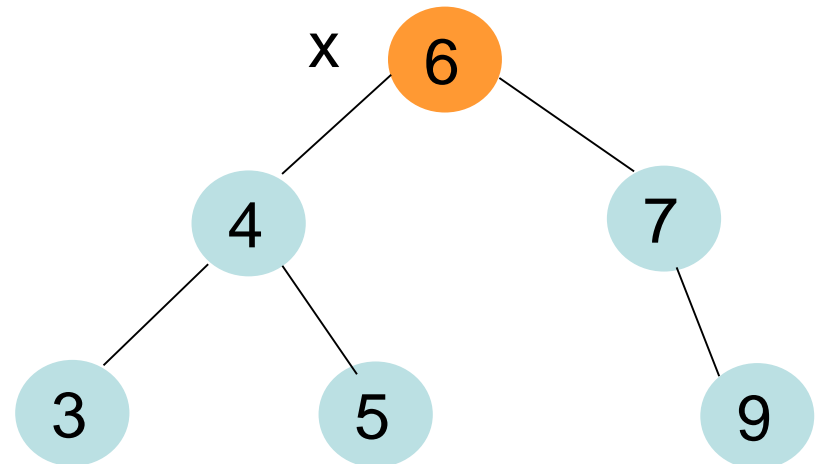
# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.    $y \leftarrow x$
4.   **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5.   **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9.   **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

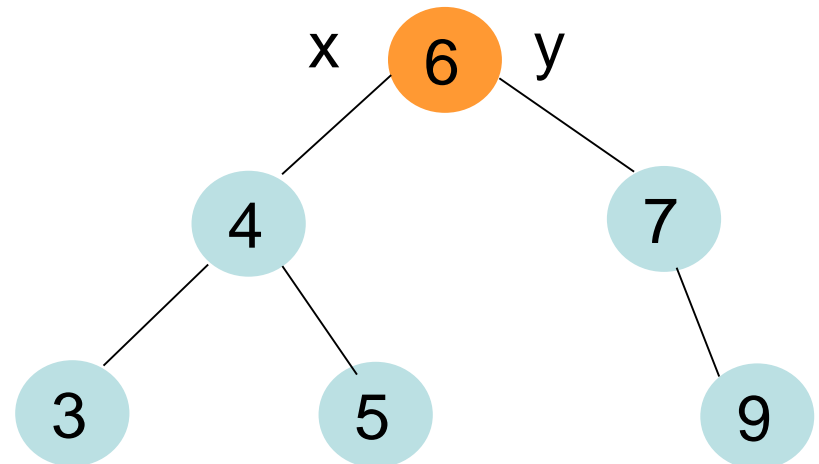


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

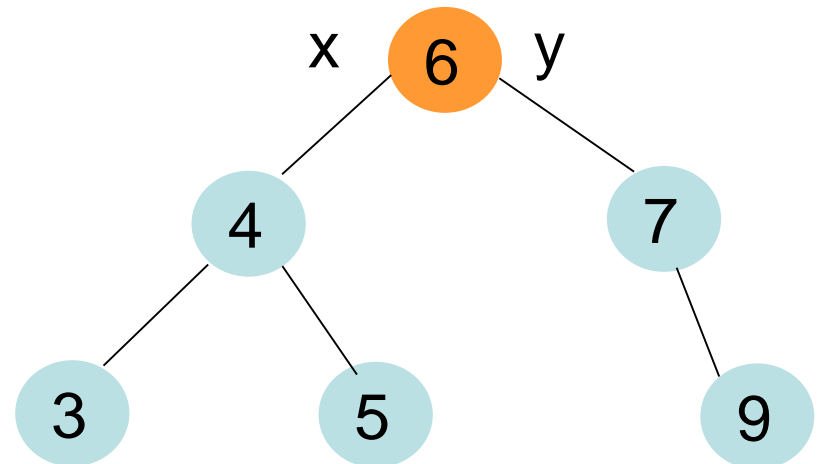


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

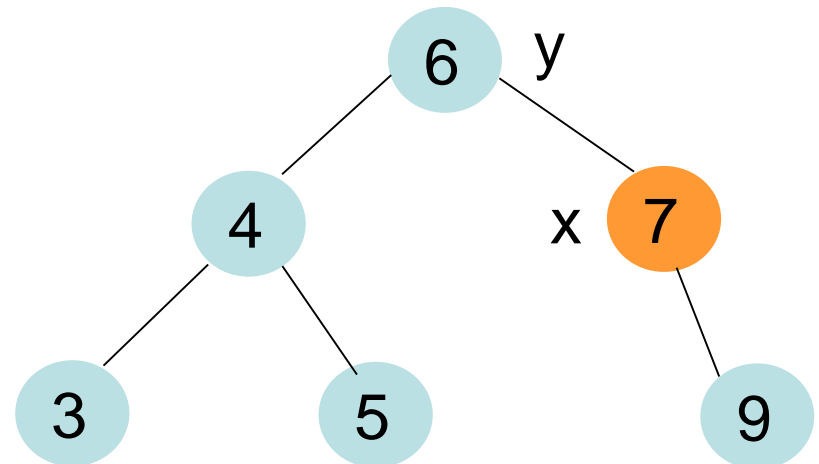


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)



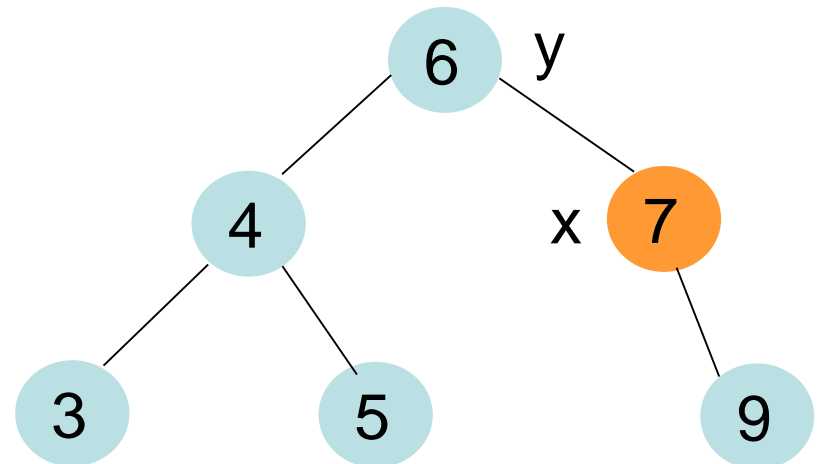
# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)



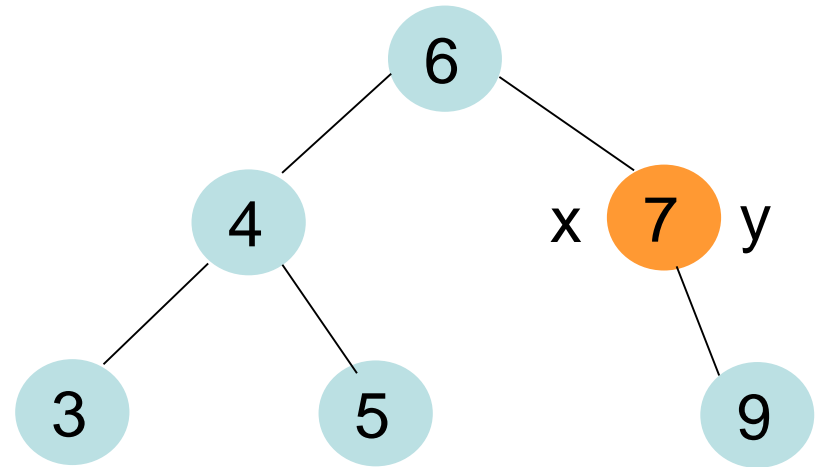
# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)





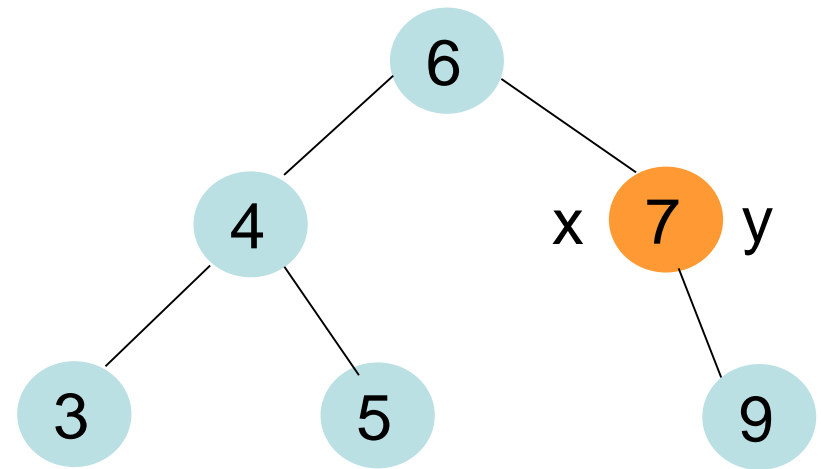
# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)



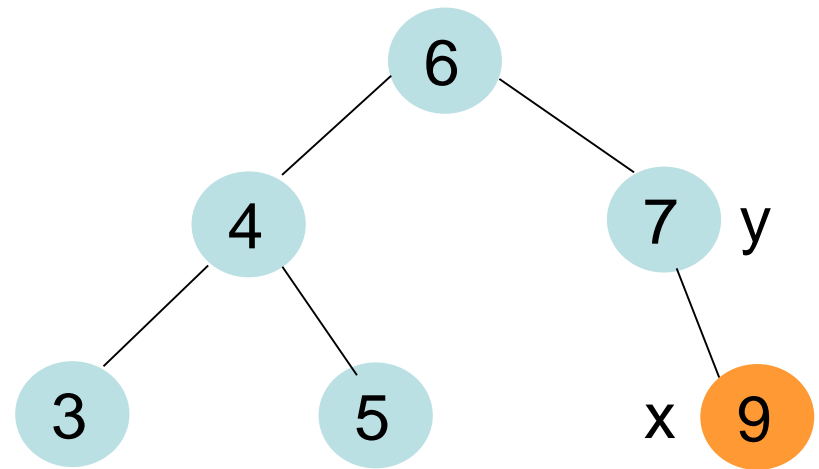
# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)



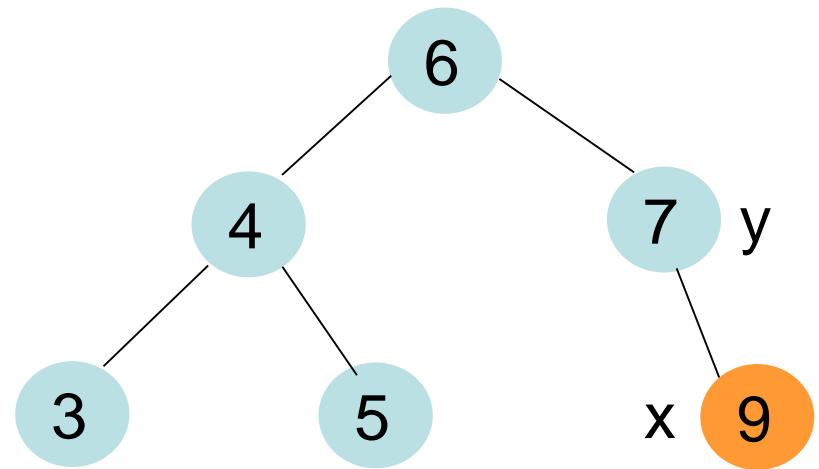
# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.    $y \leftarrow x$
4.   **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5.   **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9.   **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

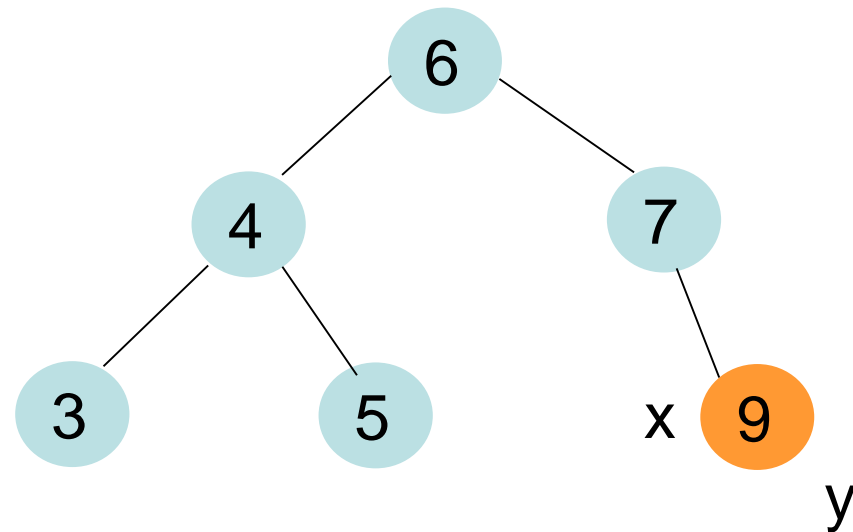


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

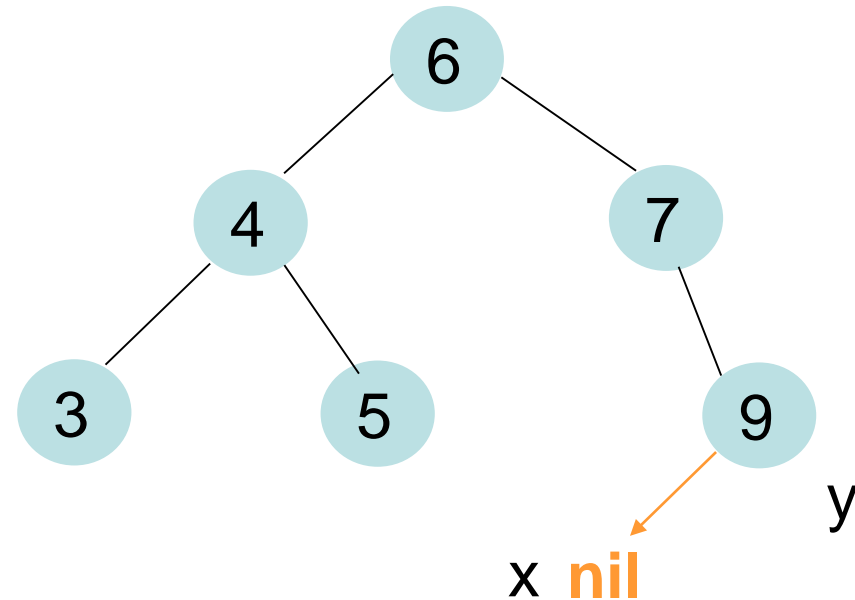


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

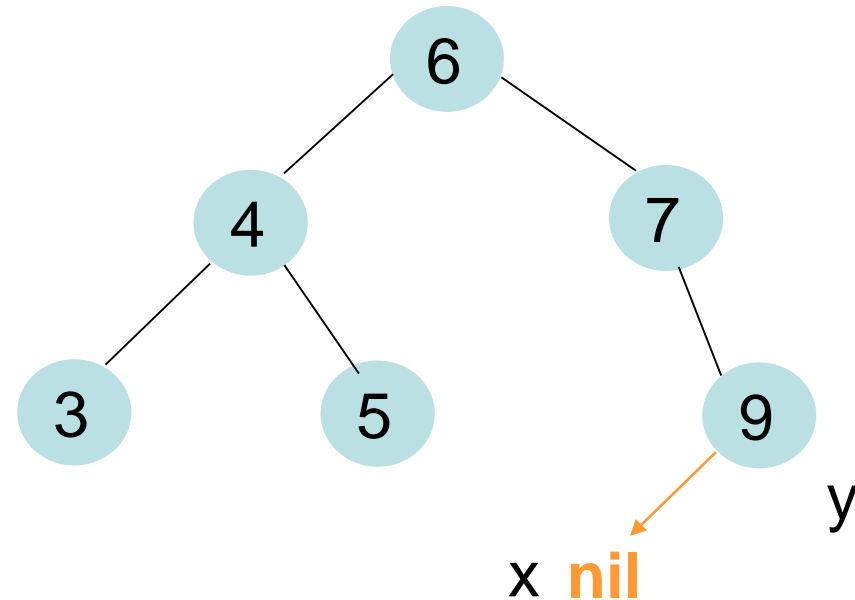


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

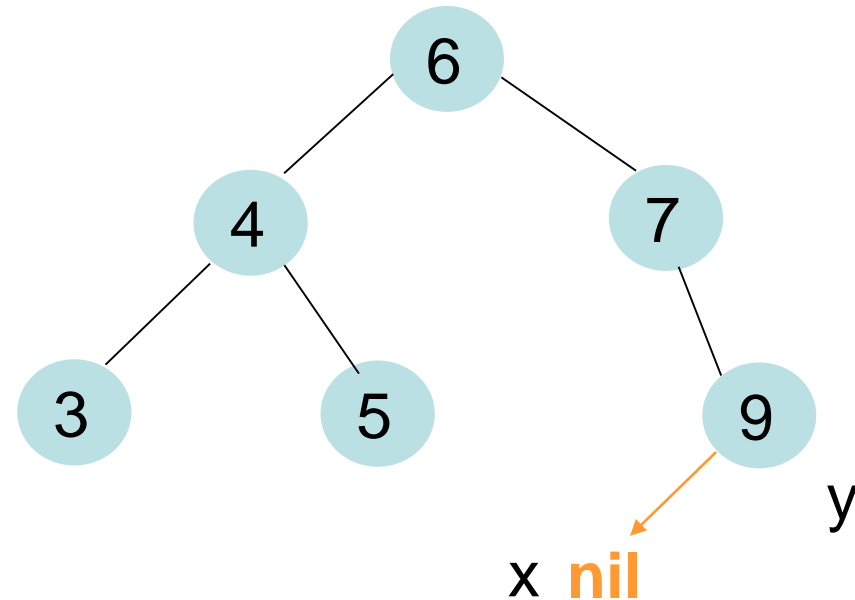


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

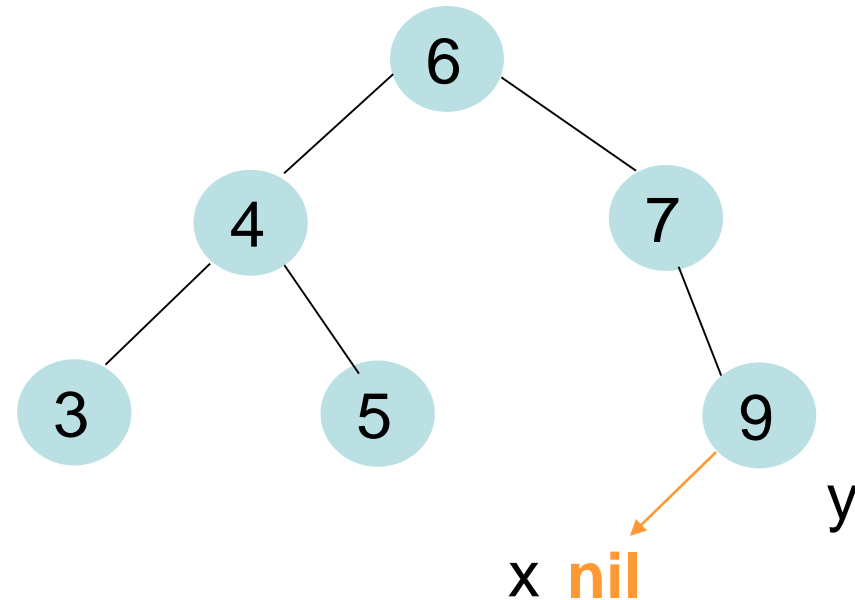


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)



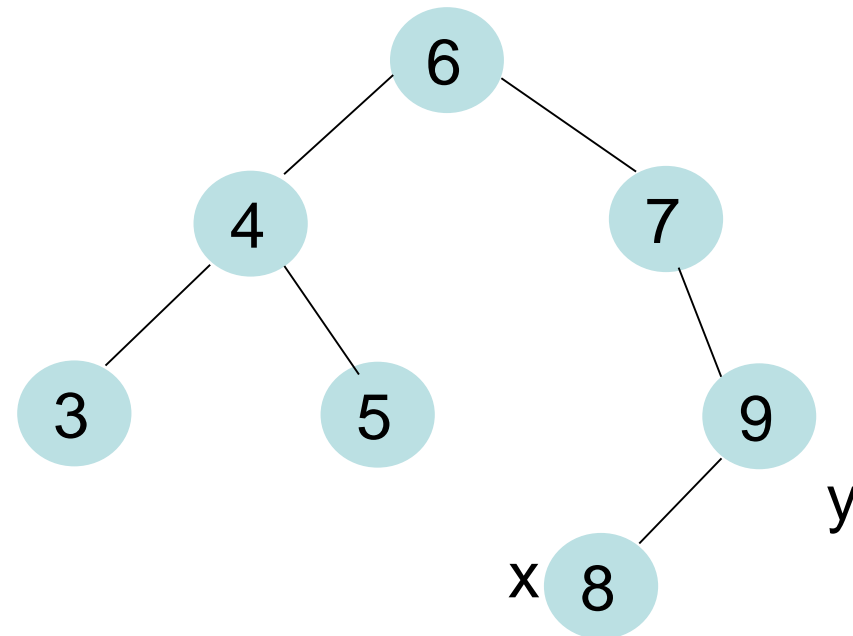


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \text{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \text{nil}$  **do**
3.  $y \leftarrow x$
4. **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5. **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \text{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9. **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

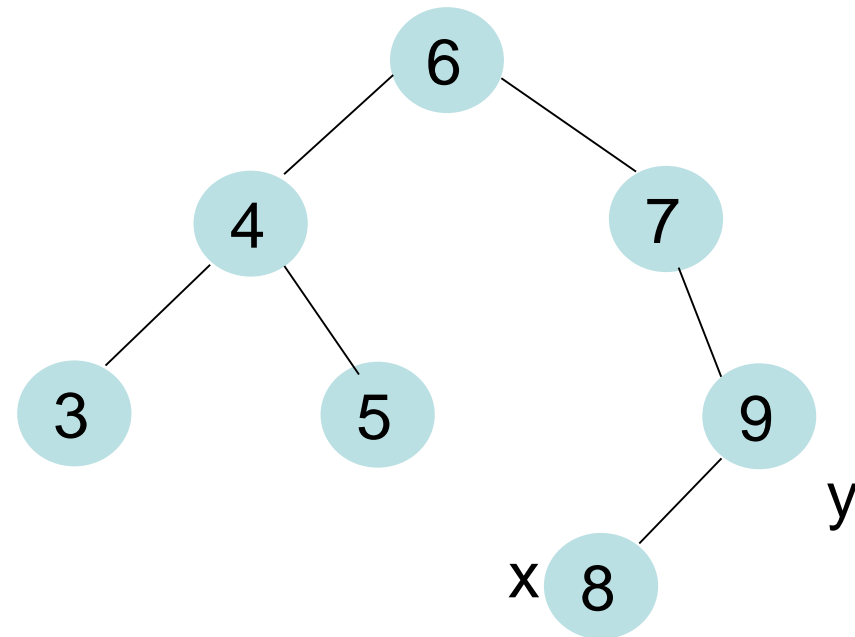


# Binäre Suchbäume

Einfügen(T,z)

1.  $y \leftarrow \mathbf{nil}$ ;  $x \leftarrow \mathbf{root}[T]$
2. **while**  $x \neq \mathbf{nil}$  **do**
3.    $y \leftarrow x$
4.   **if**  $\mathbf{key}[z] < \mathbf{key}[x]$  **then**  $x \leftarrow \mathbf{lc}[x]$
5.   **else**  $x \leftarrow \mathbf{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \mathbf{nil}$  **then**  $\mathbf{root}[T] \leftarrow z$
8. **else**
9.   **if**  $\mathbf{key}[z] < \mathbf{key}[y]$  **then**  $\mathbf{lc}[y] \leftarrow z$
10.   **else**  $\mathbf{rc}[y] \leftarrow z$

Einfügen(8)



# Binäre Suchbäume

---

Einfügen(T,z)

1.  $y \leftarrow \mathbf{nil}$ ;  $x \leftarrow \text{root}[T]$
2. **while**  $x \neq \mathbf{nil}$  **do**
3.    $y \leftarrow x$
4.   **if**  $\text{key}[z] < \text{key}[x]$  **then**  $x \leftarrow \text{lc}[x]$
5.   **else**  $x \leftarrow \text{rc}[x]$
6.  $p[z] \leftarrow y$
7. **if**  $y = \mathbf{nil}$  **then**  $\text{root}[T] \leftarrow z$
8. **else**
9.   **if**  $\text{key}[z] < \text{key}[y]$  **then**  $\text{lc}[y] \leftarrow z$
10. **else**  $\text{rc}[y] \leftarrow z$

Einfügen(8)

**Laufzeit:**  
 $O(h)$

# Binäre Suchbäume

---

## Löschen:

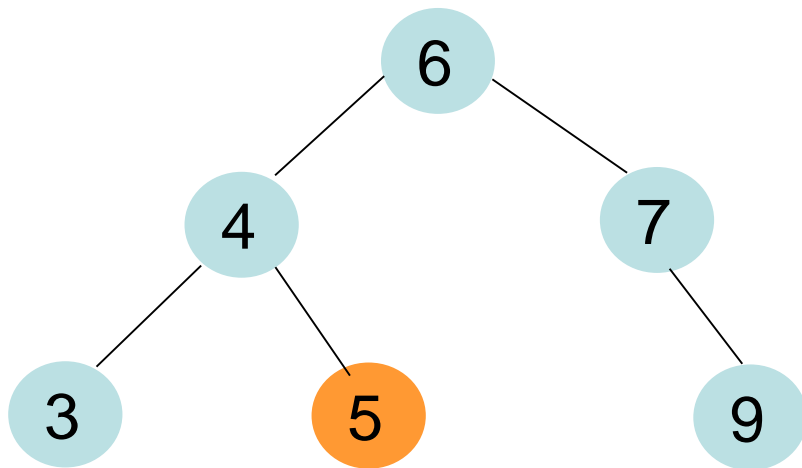
- 3 unterschiedliche Fälle
- (a) zu löschendes Element  $z$  hat keine Kinder
- (b) zu löschendes Element  $z$  hat ein Kind
- (c) zu löschendes Element  $z$  hat zwei Kinder

# Binäre Suchbäume

---

## Fall (a):

- zu löschendes Element z hat keine Kinder

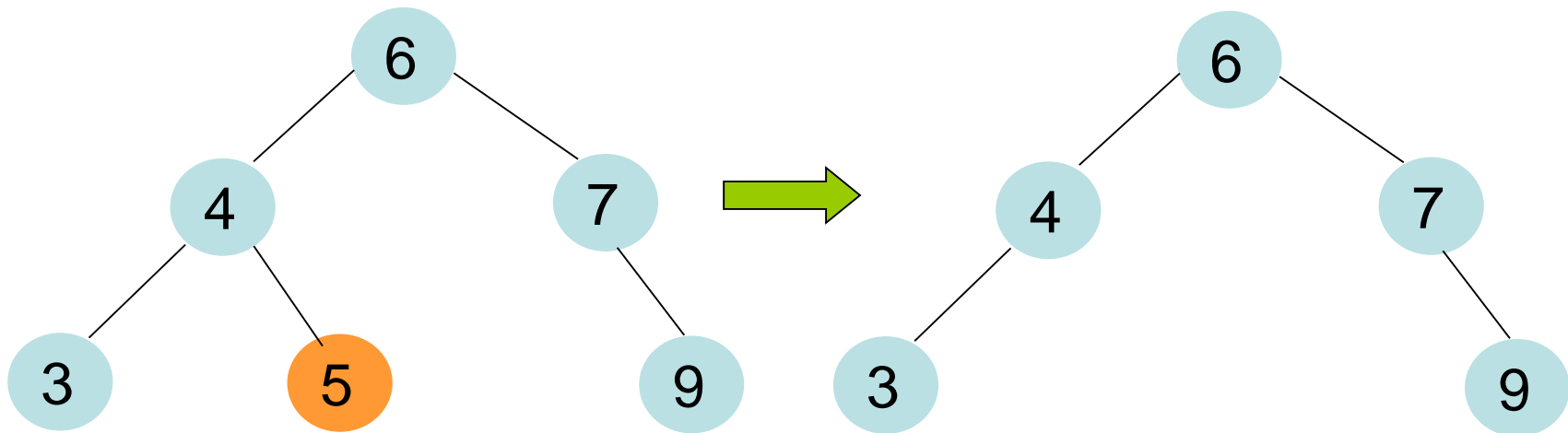


# Binäre Suchbäume

---

## Fall (a):

- zu löschendes Element z hat keine Kinder
- Entferne Element

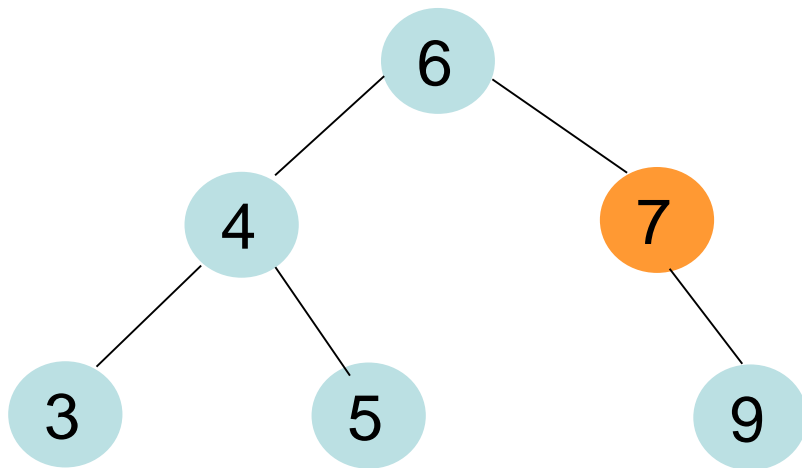


# Binäre Suchbäume

---

## Fall (b):

- zu löschendes Element z hat ein Kind

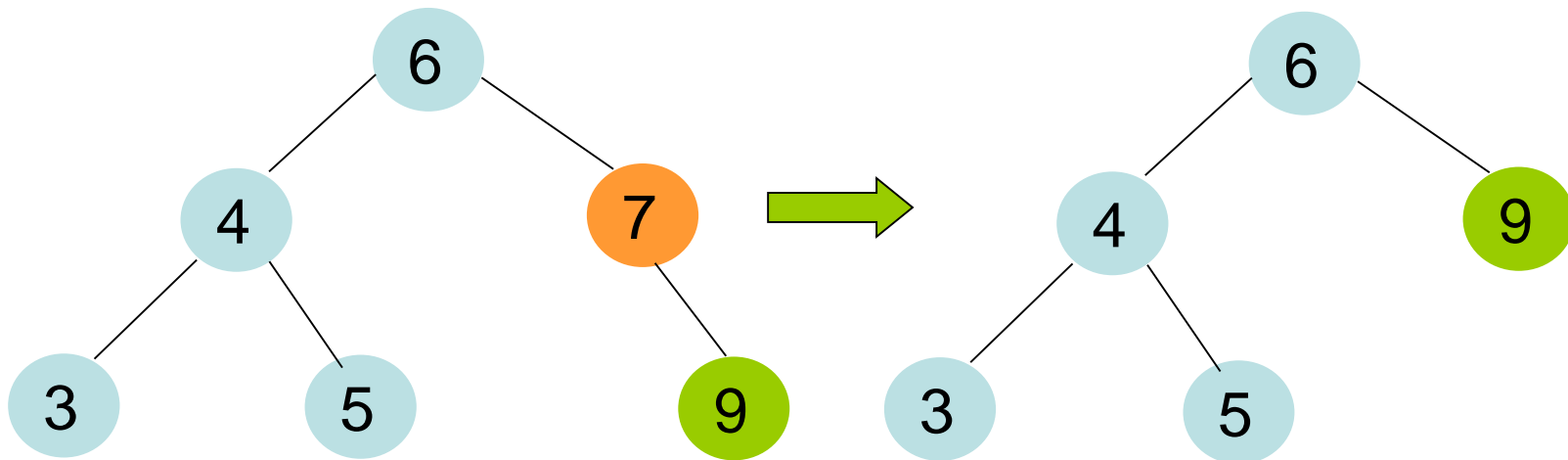


# Binäre Suchbäume

---

## Fall (b):

- zu löschendes Element z hat ein Kind
- Hänge der Unterbaum von z an den Vater von z



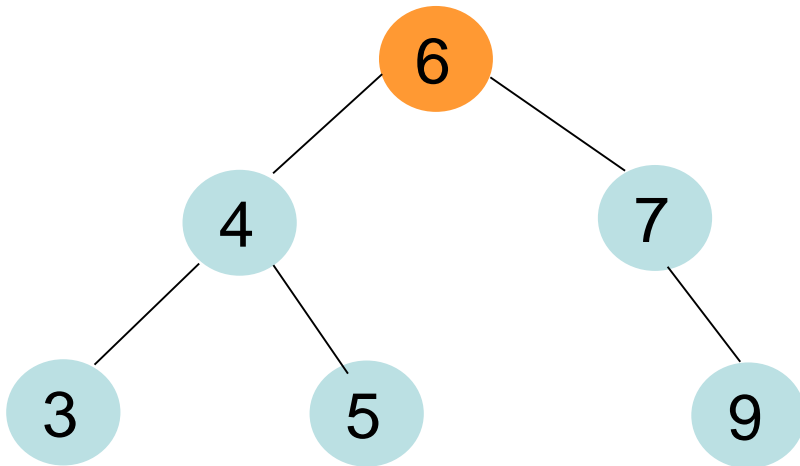


# Binäre Suchbäume

---

## Fall (c):

- zu löschendes Element z hat zwei Kinder

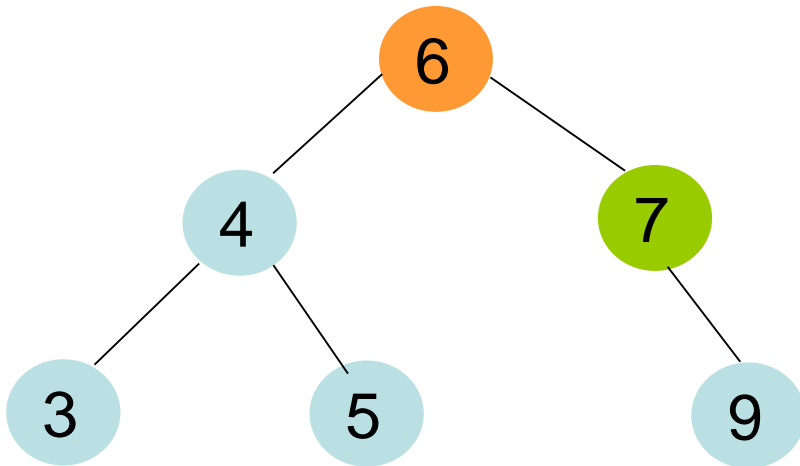


# Binäre Suchbäume

---

## Fall (c):

- zu löschendes Element  $z$  hat zwei Kinder
- Schritt 1: Bestimme Nachfolger von  $z$

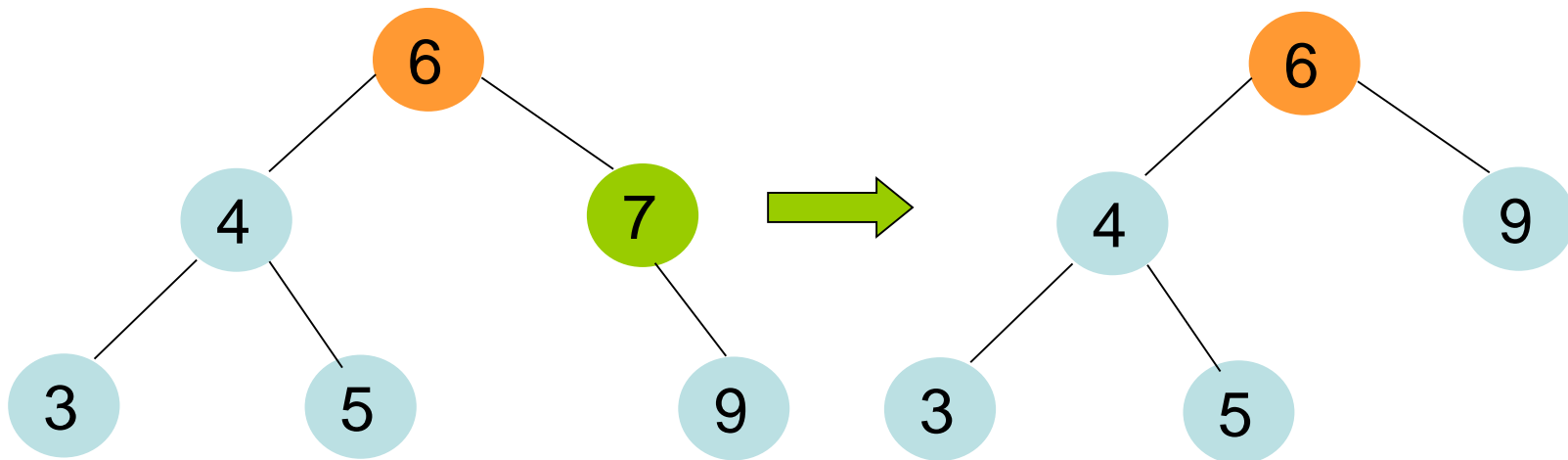


# Binäre Suchbäume

---

## Fall (c):

- zu löschendes Element z hat zwei Kinder
- Schritt 1: Bestimme Nachfolger von z
- Schritt 2: Entferne Nachfolger

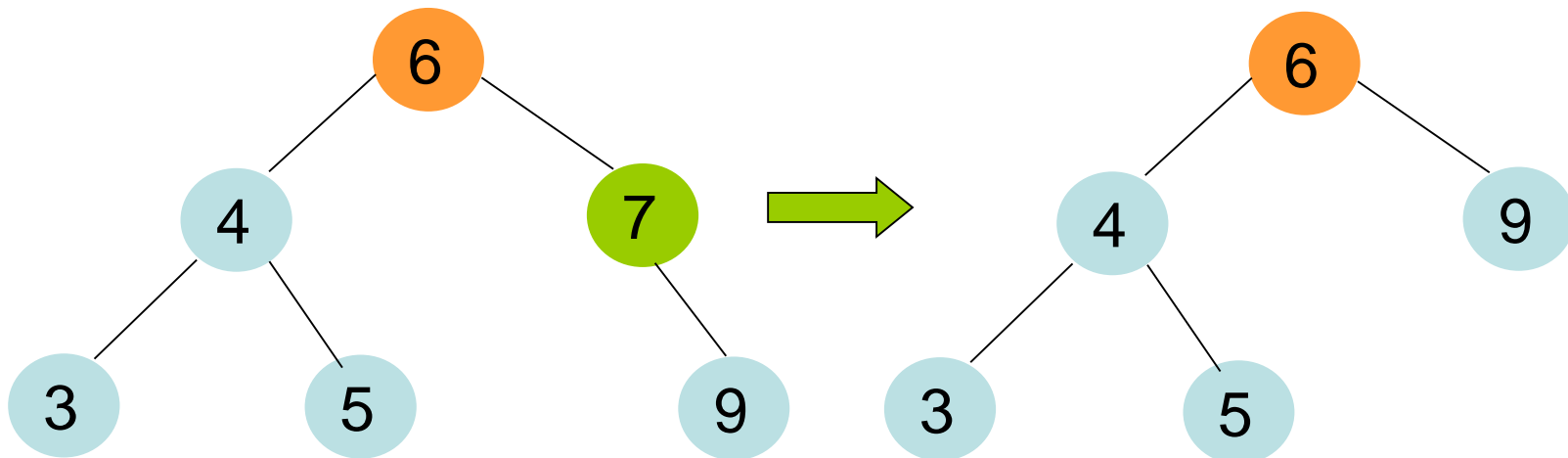


# Binäre Suchbäume

## Fall (c):

- zu löschendes Element z hat zwei Kinder
- Schritt 1: Bestimme Nachfolger von z
- Schritt 2: Entferne Nachfolger

Nachfolger hat nur ein Kind!

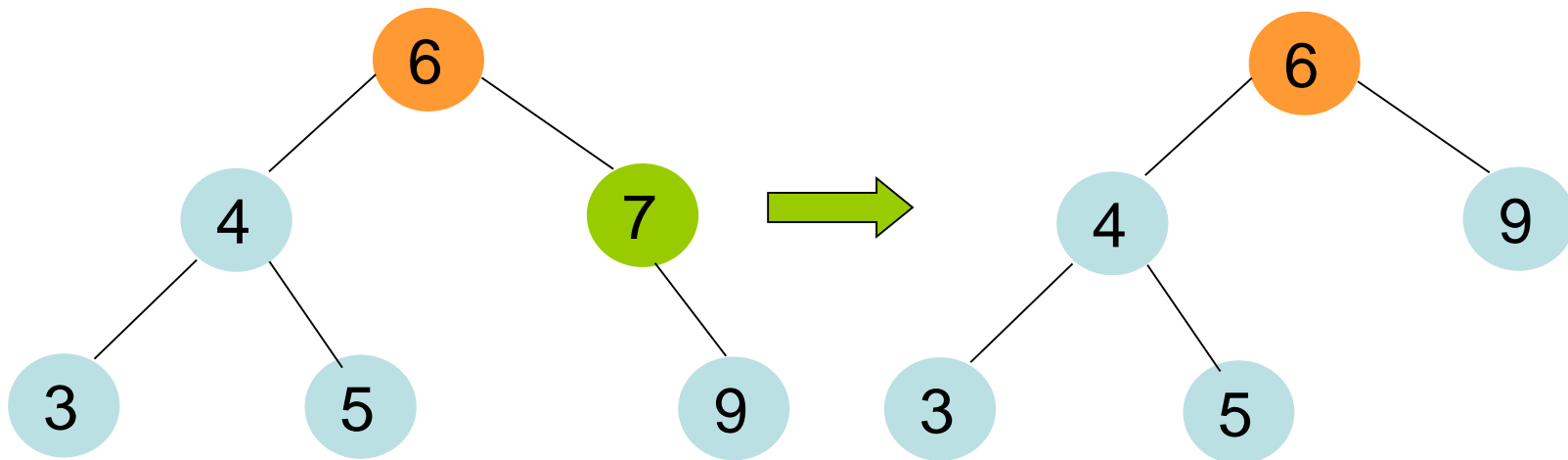


# Binäre Suchbäume

---

Nachfolger von  $z$  hat nur ein Kind:

- Nachfolger ist Minimum in Teilbaum mit Wurzel  $rc[z]$
- Dieses Minimum  $y$  ist entweder ein Blatt, oder  $lc[y]=nil$ .

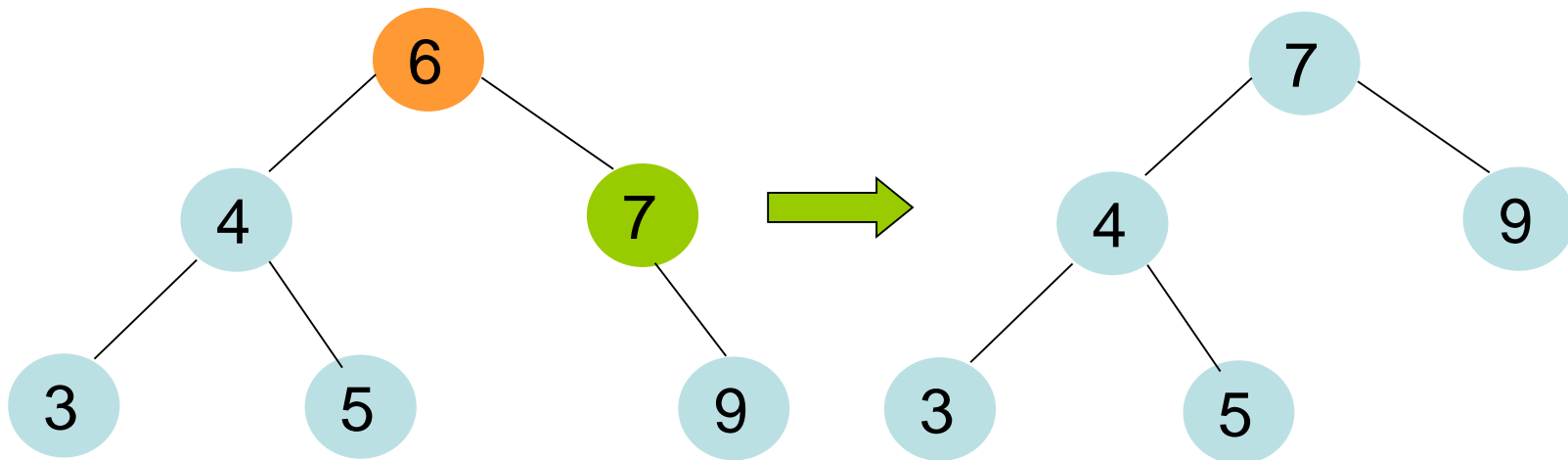


# Binäre Suchbäume

---

## Fall (c):

- zu löschendes Element z hat zwei Kinder
- Schritt 1: Bestimme Nachfolger von z
- Schritt 2: Entferne Nachfolger; ersetze z durch Nachfolger

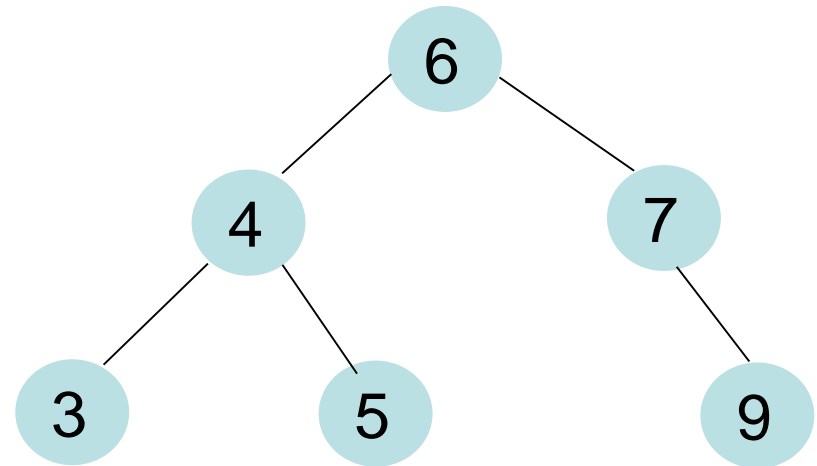


# Binäre Suchbäume

---

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y



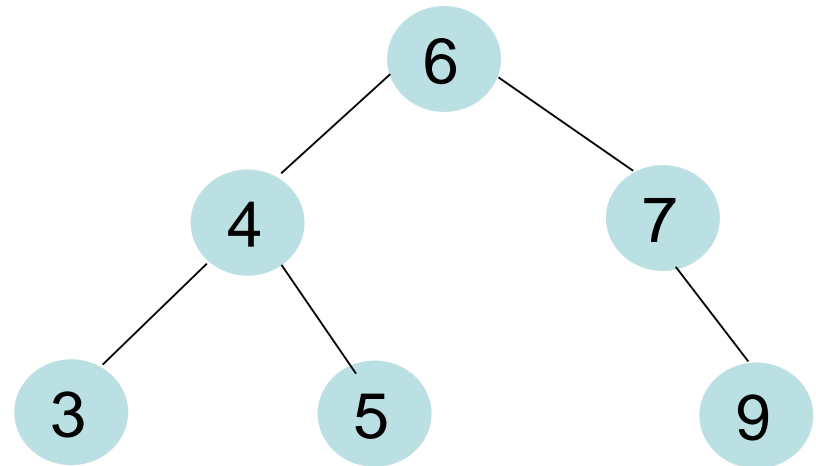
# Binäre Suchbäume

---

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Löschen(6)





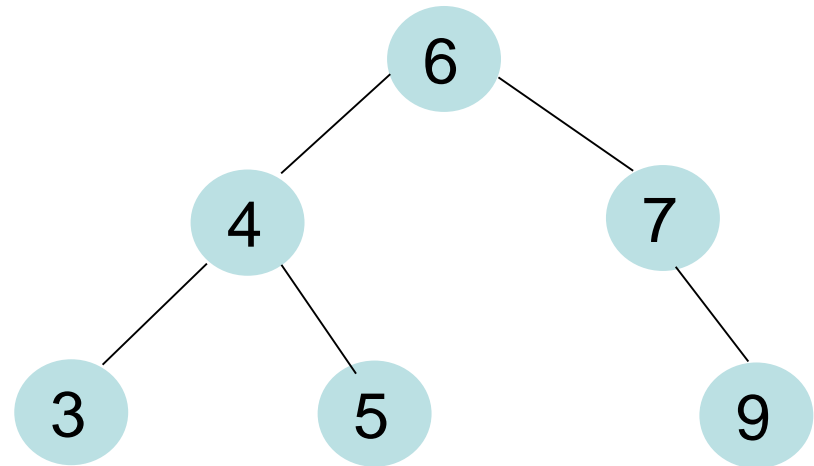
# Binäre Suchbäume

Referenz auf z  
wird übergeben!

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Löschen(6)



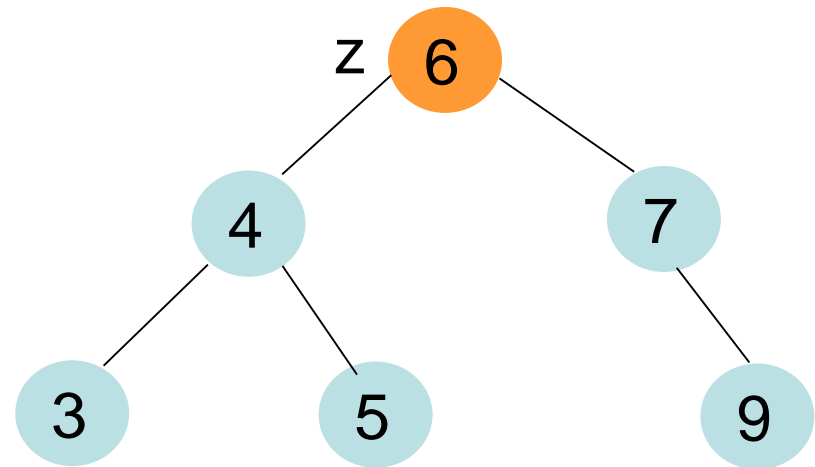
# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Bestimme Knoten, der gelöscht werden soll. Der Knoten hat nur ein Kind.

Löschen(6)



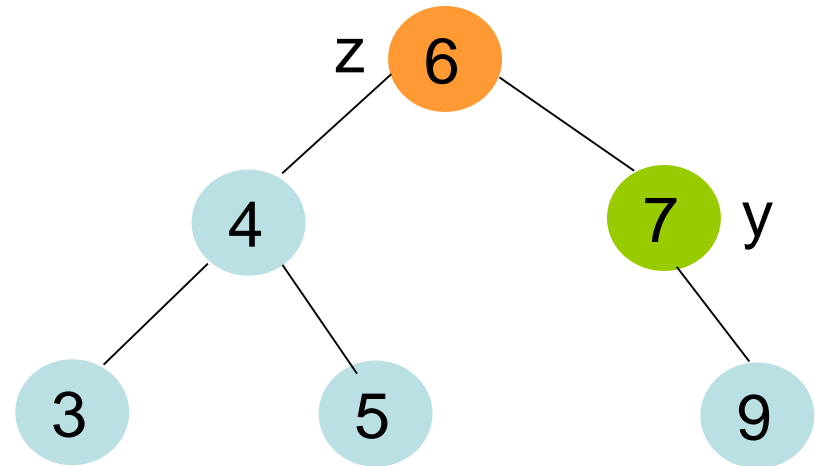
# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Bestimme Knoten, der gelöscht werden soll. Der Knoten hat nur ein Kind.

Löschen(6)



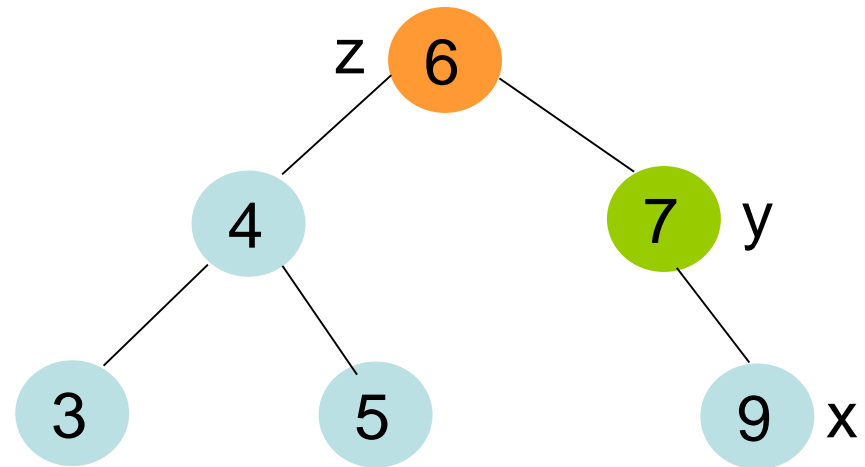
# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ←
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Bestimme das Kind von y (falls dieses existiert).

Löschen(6)



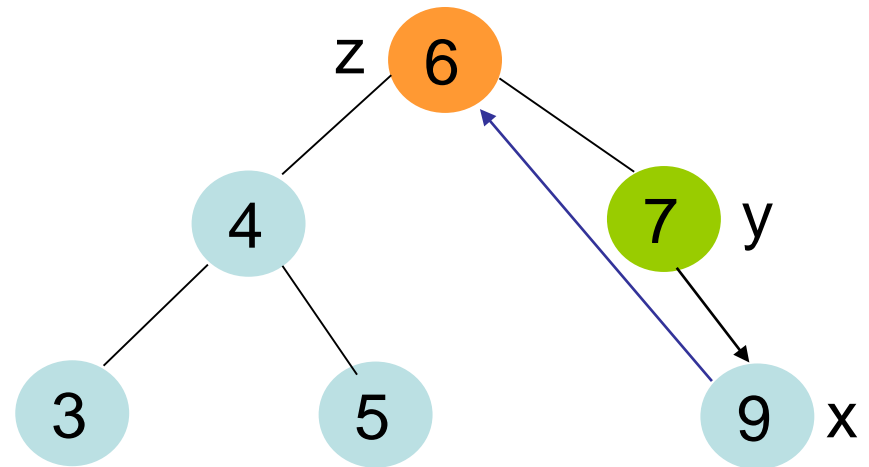
# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Aktualisiere  
Vaterzeiger von  
x

löschen(6)

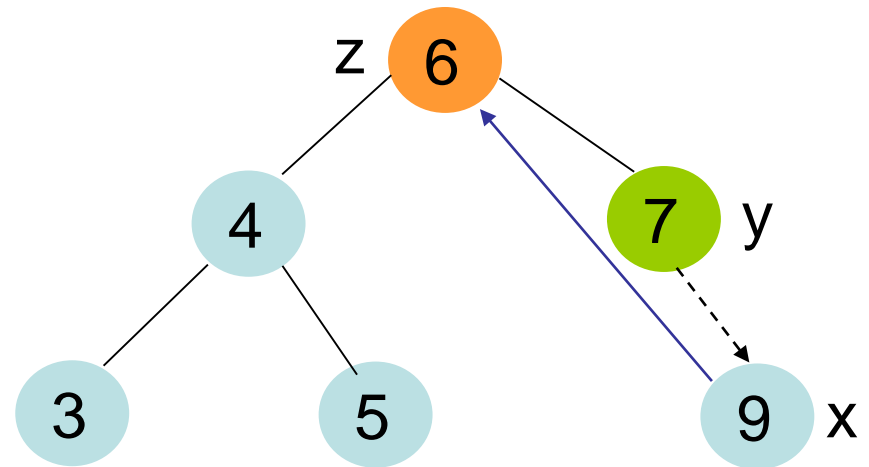


# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Löschen(6)

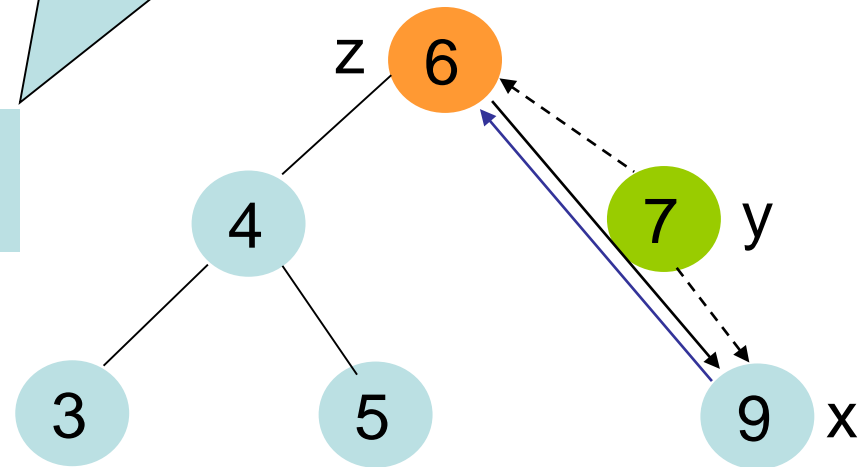


# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Beispiel: Das rechte Kind von z wird x.

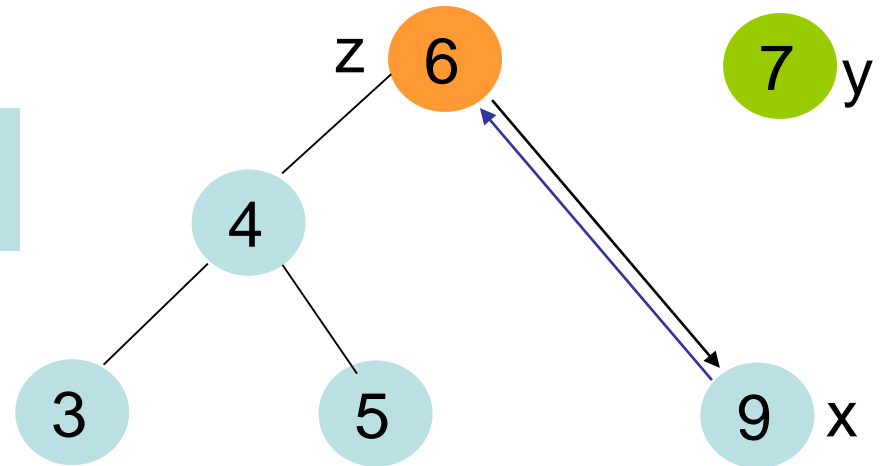


# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Löschen(6)





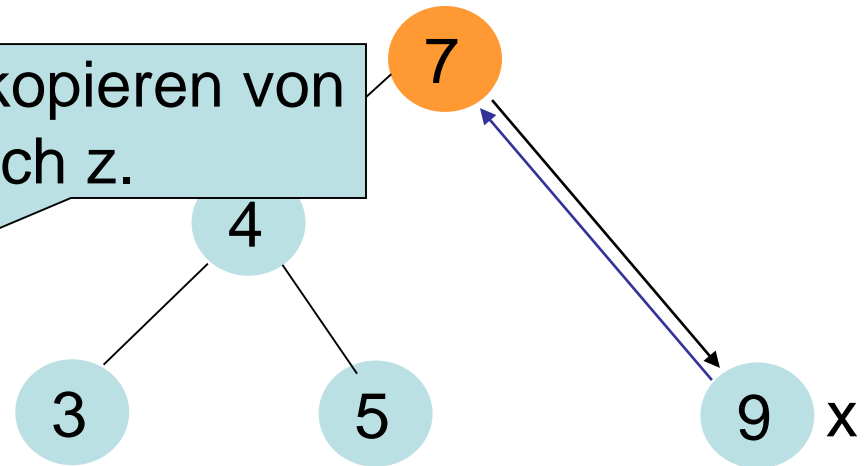
# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← rc[p[y]]
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Löschen(6)

Umkopieren von  
y nach z.



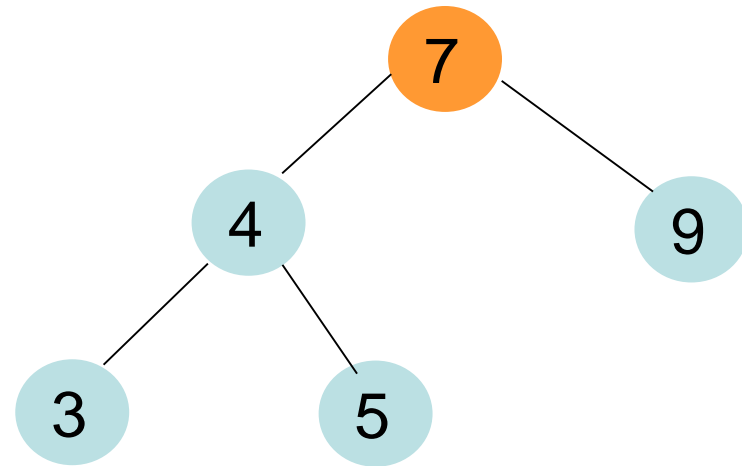
# Binäre Suchbäume

---

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Löschen(6)



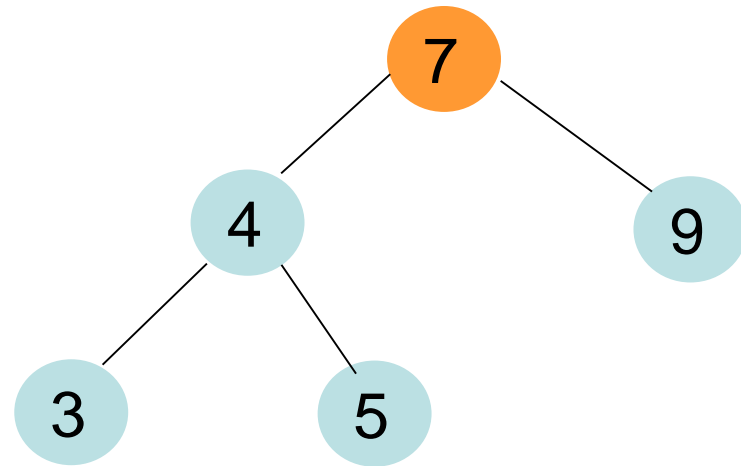
# Binäre Suchbäume

---

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Löschen(6)



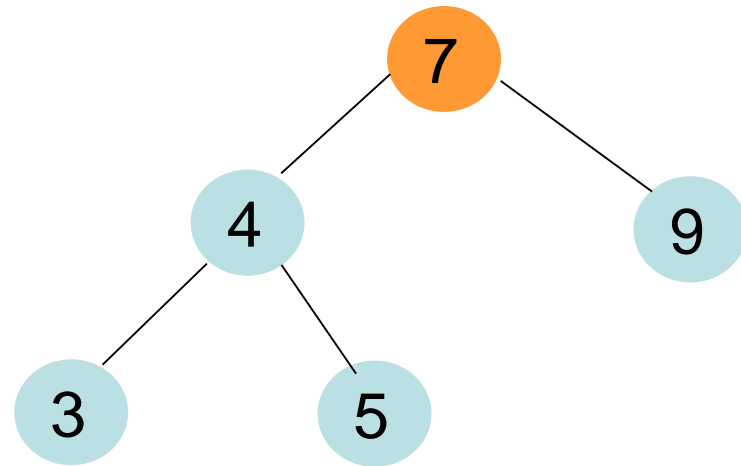
# Binäre Suchbäume

Löschen(T,z)

1. **if** lc[z]=nil or rc[z]=nil **then** y ← z
2. **else** y ← NachfolgerSuche(z)
3. **if** lc[y] ≠ nil **then** x ← lc[y]
4. **else** x ← rc[y]
5. **if** x ≠ nil **then** p[x] ← p[y]
6. **if** p[y]=nil **then** root[T] ← x
7. **else if** y=lc[p[y]] **then** lc[p[y]] ← x
8. **else** rc[p[y]] ← x
9. **if** y ≠ z **then** key[z] ← key[y]
10. **return** y

Laufzeit O(h)

Löschen(6)



# Binäre Suchbäume

---

## Binäre Suchbäume:

- Ausgabe aller Elemente in  $O(n)$
- Suche, Minimum, Maximum, Nachfolger in  $O(h)$
- Einfügen, Löschen in  $O(h)$

## Frage:

- Wie kann man eine „kleine“ Höhe unter Einfügen und Löschen garantieren?