

Einführung in die Programmierung

Bäume

Ein grundlegendes Problem

- Speicherung von Datensätzen
- Listen oft nicht ausreichend, da $O(n)$ zu langsam ist

Beispiel

- Stammbaum
- Dateisysteme
- Entscheidungsbäume
- Suchbäume, z.B. für Lexika Daten
- Rekursionsbäume

Anforderungen

- Schneller Zugriff, d.h. schneller als $O(n)$
- Einfügen neuer Datensätze
- Löschen bestehender Datensätze

Zugriff auf Daten

- Jedes Datum (Objekt) hat einen Schlüssel
- Eingabe des Schlüssels liefert Datensatz
- Schlüssel sind vergleichbar (es gibt eine totale Ordnung auf den Schlüsseln)

Beispiel

- Lexika (Begriff, Erläuterung)
- Schlüssel: Begriff
- Totale Ordnung: Lexikographische Ordnung

Problem:

- Gegeben sind n Objekte O_1, \dots, O_n mit zugehörigen Schlüsseln $s(O_i)$

Operationen:

- **Suche(x)**; Ausgabe O mit Schlüssel $s(O) = x$;
nil (NULL), falls kein Objekt mit Schlüssel x in Datenmenge
- **Einfügen(O)**; Einfügen von Objekt O in Datenmenge
- **Löschen(O)**; Löschen von Objekt O mit aus der Datenmenge

Bisher haben wir folgenden grundlegende Datenstrukturen

- Felder/Arrays
- Sortierte Arrays
- Dynamische Arrays
- Doppelt verkettete Liste

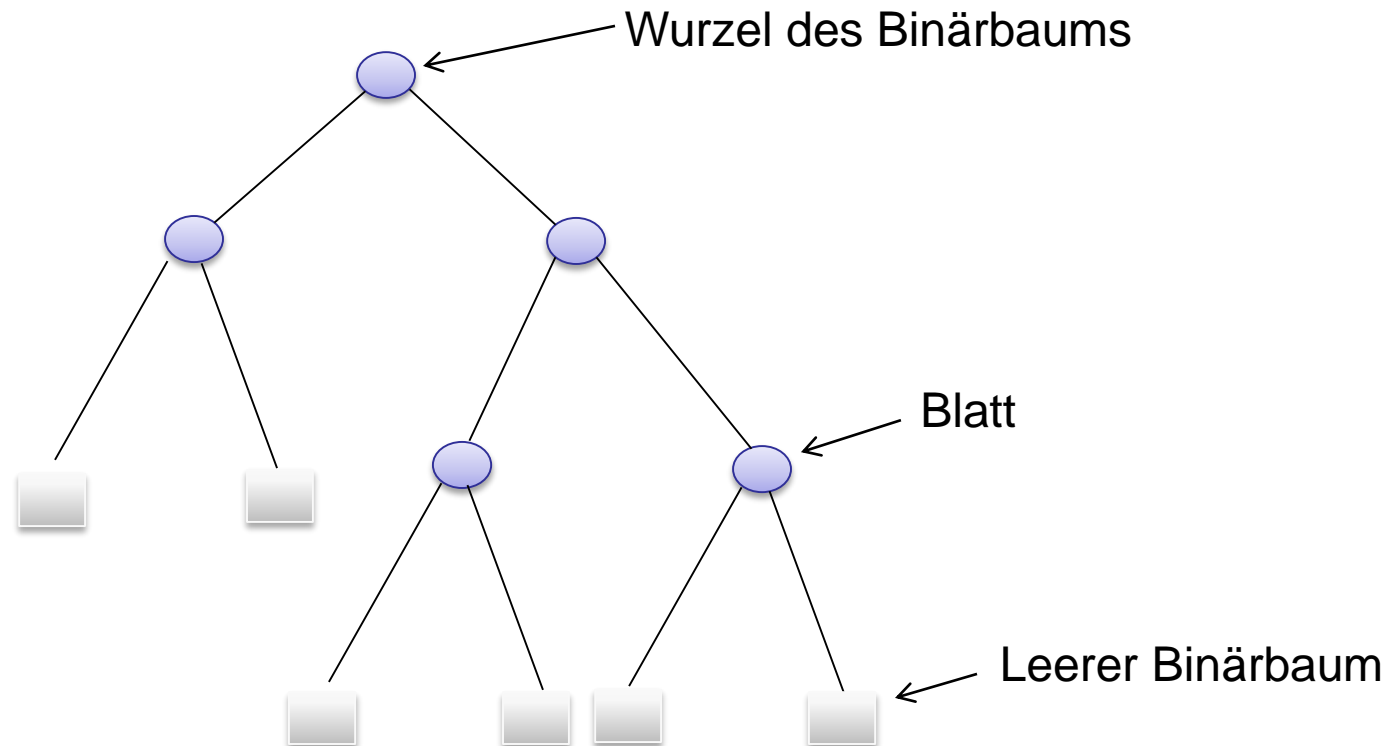
Diskussion

- Alle Datenstrukturen haben Nachteile
- Listen helfen beim Speichermanagement
- Sortierung hilft bei Suche ist aber teuer aufrecht zu erhalten

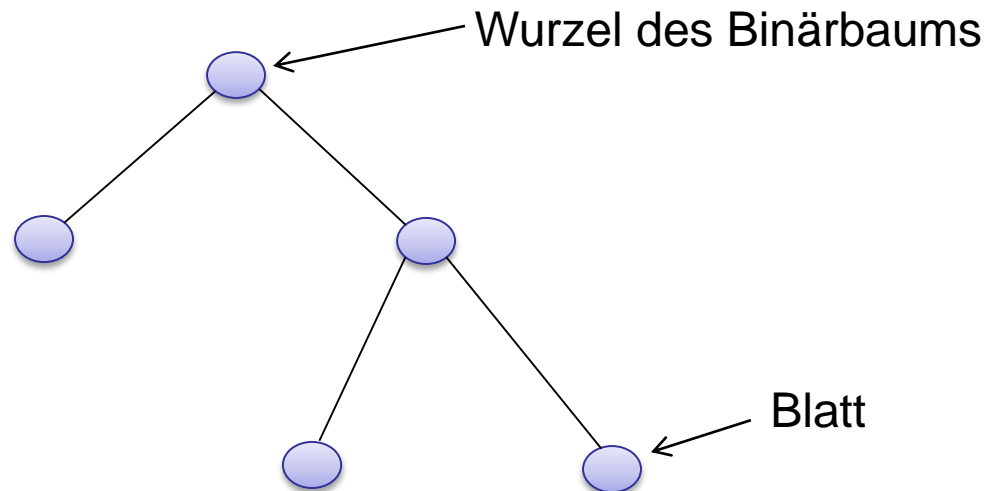
Definition (Binärbaum)

- Ein Binärbaum T ist eine Struktur, die auf einer endlichen Menge definiert ist. Diese Menge nennt man auch die Knotenmenge des Binärbaums.
- Die leere Menge ist ein Binärbaum. Dieser wird auch als leerer Baum bezeichnet.
- Ein Binärbaum ist ein Tripel (v, T_1, T_2) , wobei T_1 und T_2 Binärbäume mit disjunkten Knotenmengen V_1 und V_2 sind und $v \notin V_1 \cup V_2$ Wurzelknoten heißt. Die Knotenmenge des Baums ist dann $\{v\} \cup V_1 \cup V_2$.
 T_1 heißt linker Unterbaum von v und T_2 heißt rechter Unterbaum von v .

Darstellung von Binärbäumen



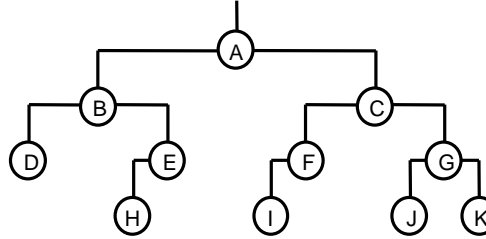
Darstellung von Binärbäumen



- Häufig lässt man die leeren Bäume in der Darstellung eines Binärbaums weg

Definition (Binärbaum)

- Ein Binärbaum T ist eine Struktur, die auf einer endlichen Menge definiert ist. Diese Menge nennt man auch die Knotenmenge des Binärbaums.
- Die leere Menge ist ein Binärbaum. Dieser wird auch als leerer Baum bezeichnet.
- Ein Binärbaum ist ein Tripel (v, T_1, T_2) , wobei T_1 und T_2 Binärbäume mit disjunkten Knotenmengen V_1 und V_2 sind und $v \notin V_1 \cup V_2$ Wurzelknoten heißt. Die Knotenmenge des Baums ist dann $\{v\} \cup V_1 \cup V_2$.
 T_1 heißt linker Unterbaum von v und T_2 heißt rechter Unterbaum von v .



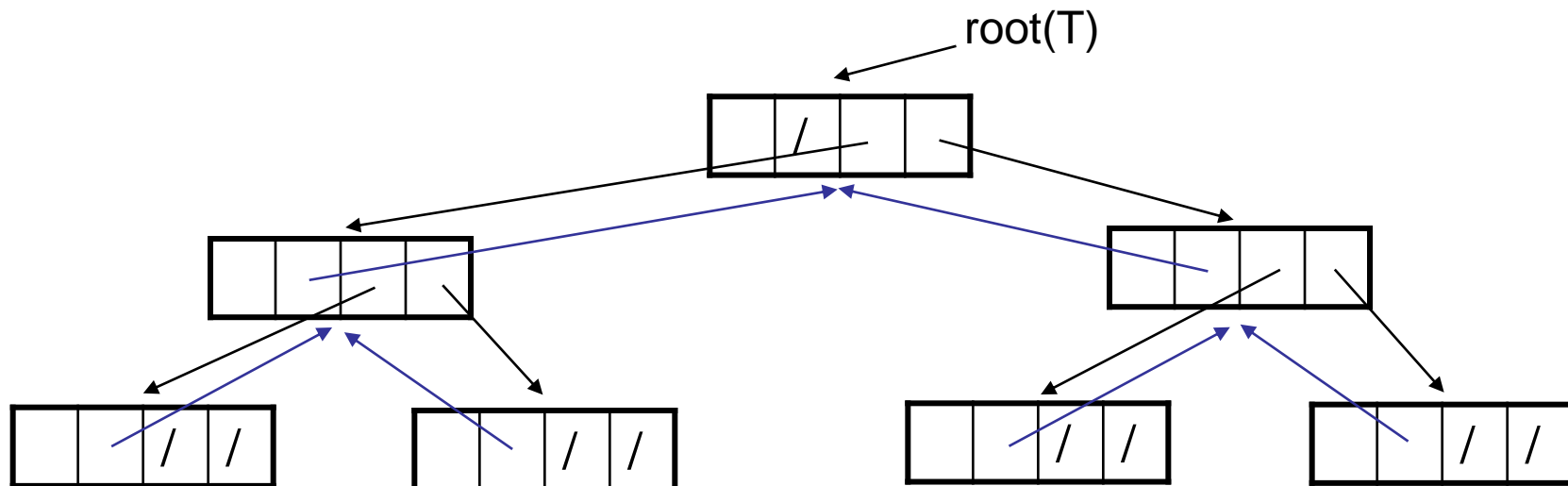
Binärbaum Begriffe

Begriff	Erläuterung	Beispiel
Vorgänger (<i>predecessor</i>)	-	A ist Vorgänger von B
Nachfolger (<i>successor</i>)	-	B ist Nachfolger von A
Wurzel (<i>root</i>)	kein Vorgänger	A
Blatt (<i>leaf</i>)	kein Nachfolger	D, H, I, J, K
interner Knoten (<i>inner node</i>)	alle Nachfolger besetzt	A, B, C, G
Randknoten (<i>boundary node</i>)	nicht alle Nachfolger besetzt	D, E, F, H, I, J, K
Pfad (<i>path</i>)	Knotenfolge von einem Anfangs- bis zum Endknoten	Pfad von der Wurzel nach F: A,C,F
Pfadlänge (<i>path length</i>)	Anzahl der Kanten des Pfads	Pfadlänge von A nach F: 2
Tiefe eines Knotens (<i>depth</i>)	Pfadlänge zur Wurzel	Tiefe von F: 2
Höhe eines Baumes (<i>height</i>)	Größte Tiefe	3
Höhe eines Knotens ν	Höhe des Teilbaums von ν	Höhe von F: 1

Binärbäume (Darstellung im Rechner)

- Schlüssel key und ggf. weitere Daten
- Zeiger lc(v) (rc(v)) auf linkes (rechtes) Kind von v
- Vaterzeiger p(v) auf Vater von v (blau)
- Wurzelzeiger root(T)

key	p(v)	lc(v)	rc(v)
-----	------	-------	-------



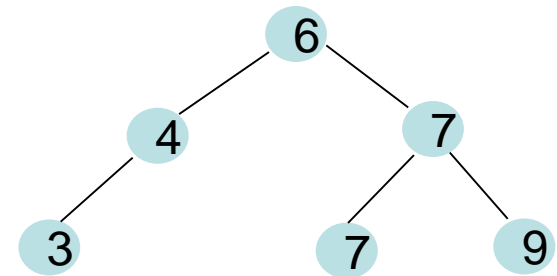
Binäre Suchbäume

Binäre Suchbäume

- Verwende Binärbaum
- Speichere Schlüssel „geordnet“

Binäre Suchbaumeigenschaft:

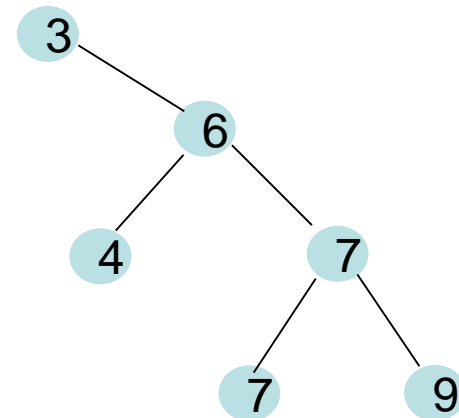
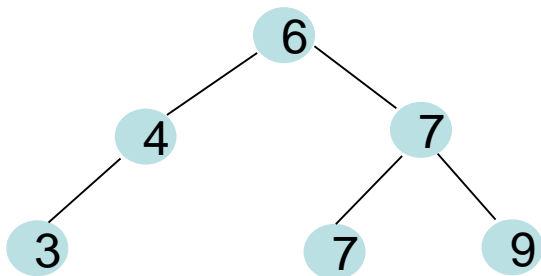
- Sei x Knoten im binären Suchbaum
- Ist y Knoten im **linken Unterbaum** von x , dann gilt **$\text{key}(y) \leq \text{key}(x)$**
- Ist y Knoten im **rechten Unterbaum** von x , dann gilt **$\text{key}(y) > \text{key}(x)$**



Binäre Suchbäume

Unterschiedliche Suchbäume

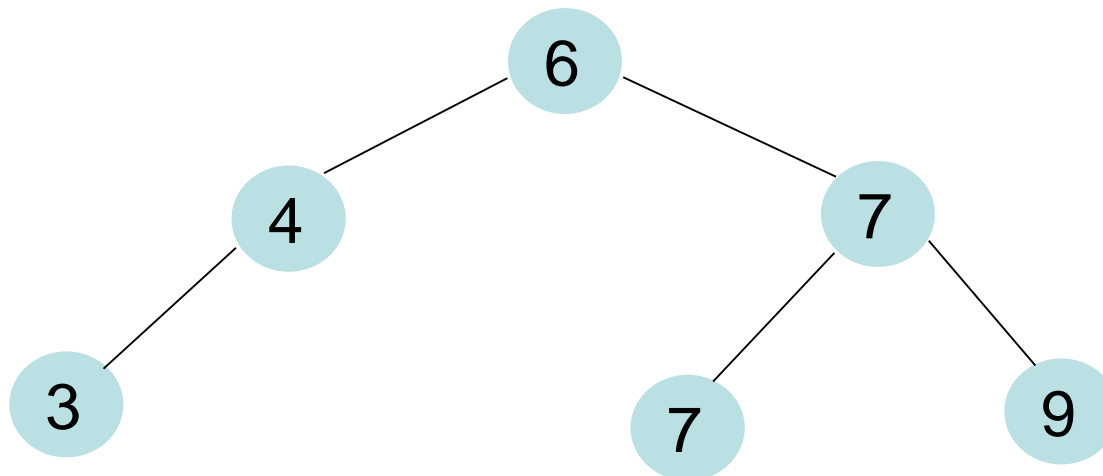
- Schlüsselmenge 3,4,6,7,7,9
- Wir erlauben mehrfache Vorkommen desselben Schlüssels



Binäre Suchbäume Durchlaufen

Ausgabe aller Schlüssel

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

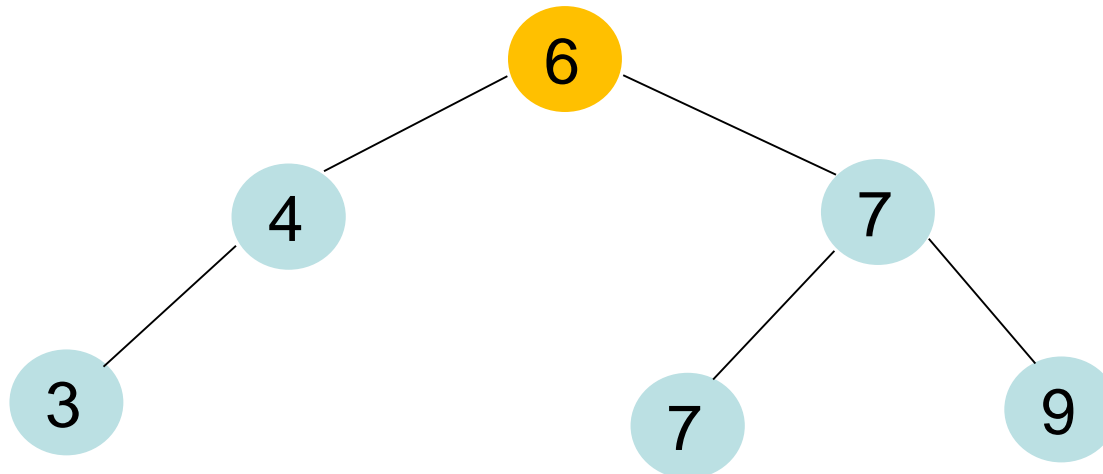


Binäre Suchbäume Durchlaufen

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))

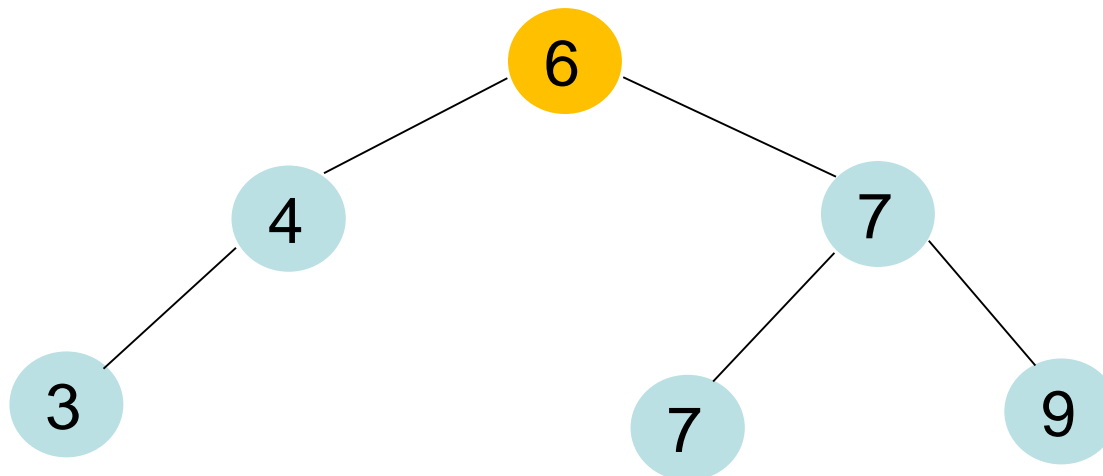
Aufruf über
Inorder-Tree-Walk(root(T))



Binäre Suchbäume

Inorder-Tree-Walk(x)

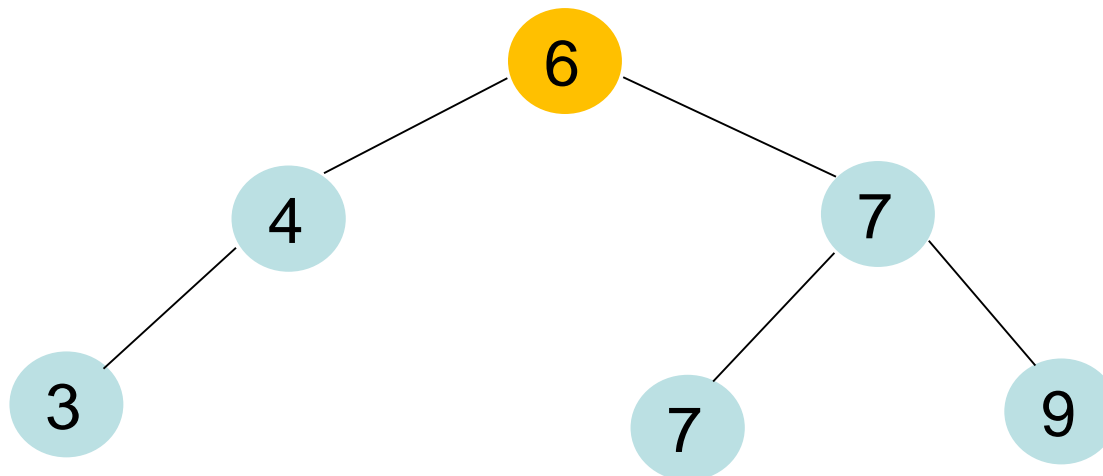
1. **if $x \neq \text{nil}$ then**
2. Inorder-Tree-Walk(lc(x))
3. Ausgabe key(x)
4. Inorder-Tree-Walk(rc(x))



Binäre Suchbäume Durchlaufen

Inorder-Tree-Walk(x)

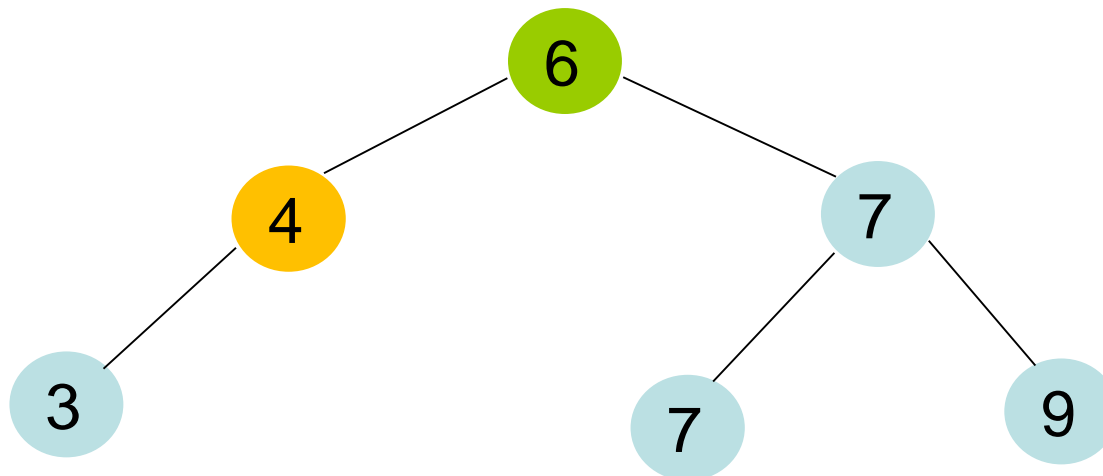
1. **if $x \neq \text{nil}$ then**
2. **Inorder-Tree-Walk(lc(x))**
3. Ausgabe key(x)
4. Inorder-Tree-Walk(rc(x))



Binäre Suchbäume Durchlaufen

Inorder-Tree-Walk(x)

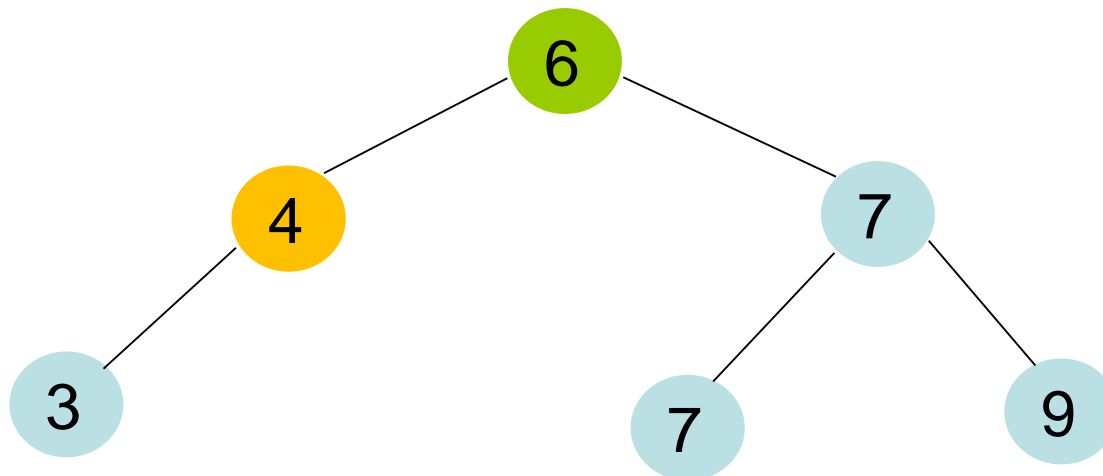
1. **if $x \neq \text{nil}$ then**
2. Inorder-Tree-Walk(lc(x))
3. Ausgabe key(x)
4. Inorder-Tree-Walk(rc(x))



Binäre Suchbäume Durchlaufen

Inorder-Tree-Walk(x)

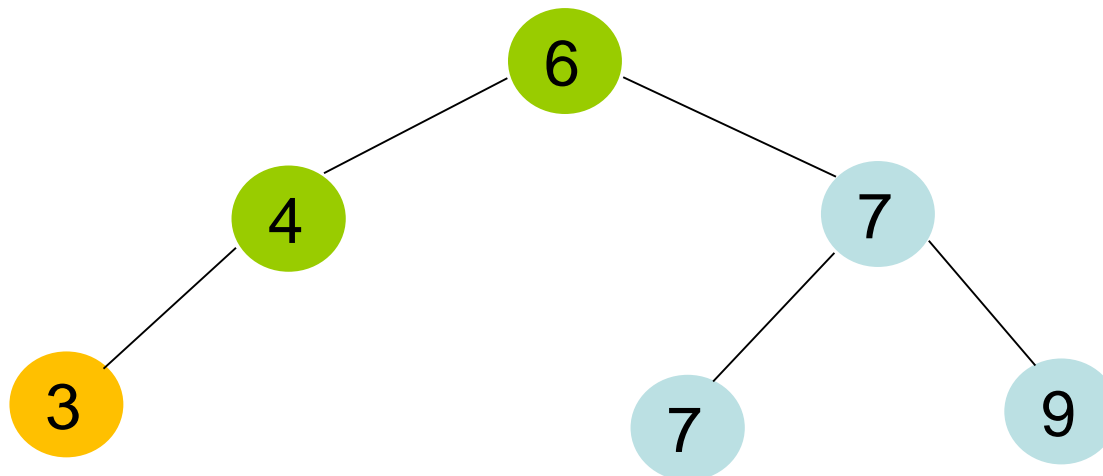
1. **if $x \neq \text{nil}$ then**
2. **Inorder-Tree-Walk(lc(x))**
3. Ausgabe key(x)
4. Inorder-Tree-Walk(rc(x))



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))

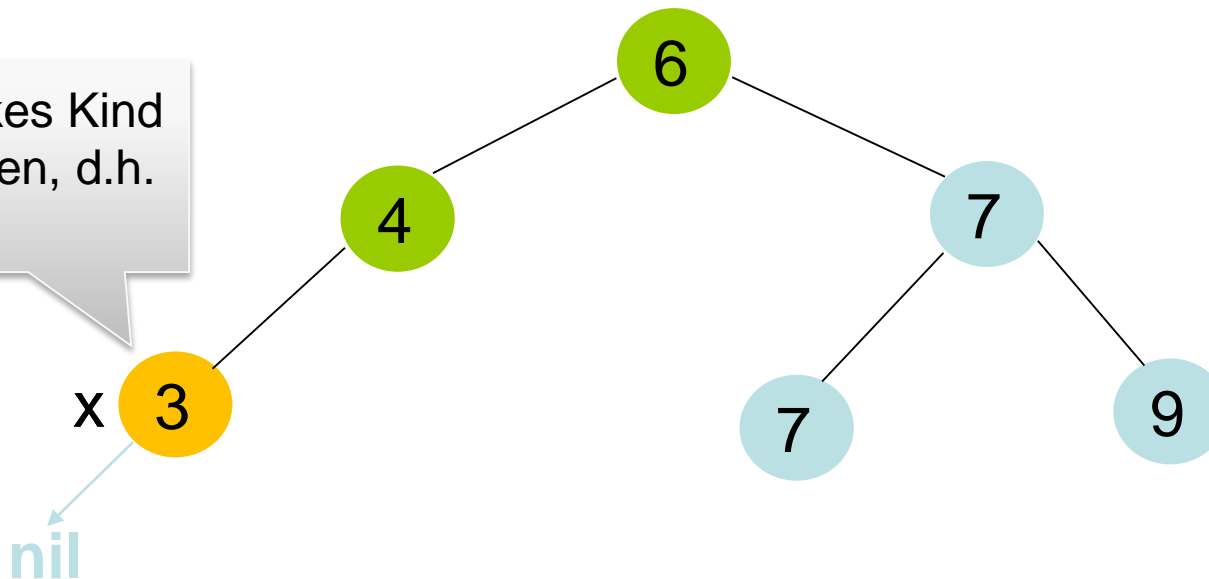


Binäre Suchbäume Durchlaufen

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))

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

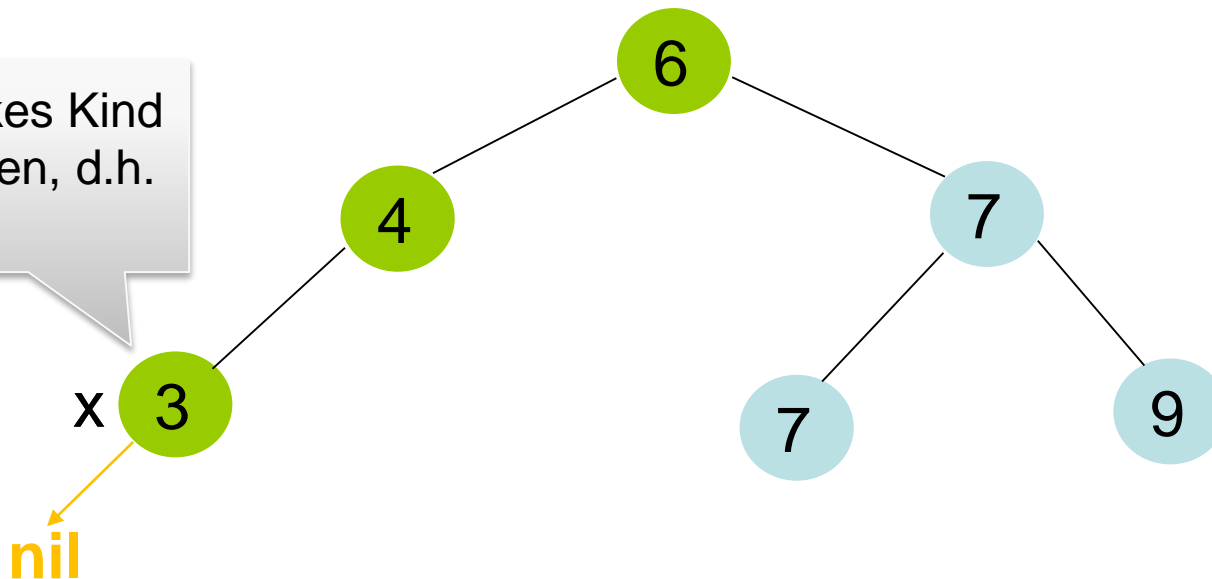


Binäre Suchbäume Durchlaufen

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))

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



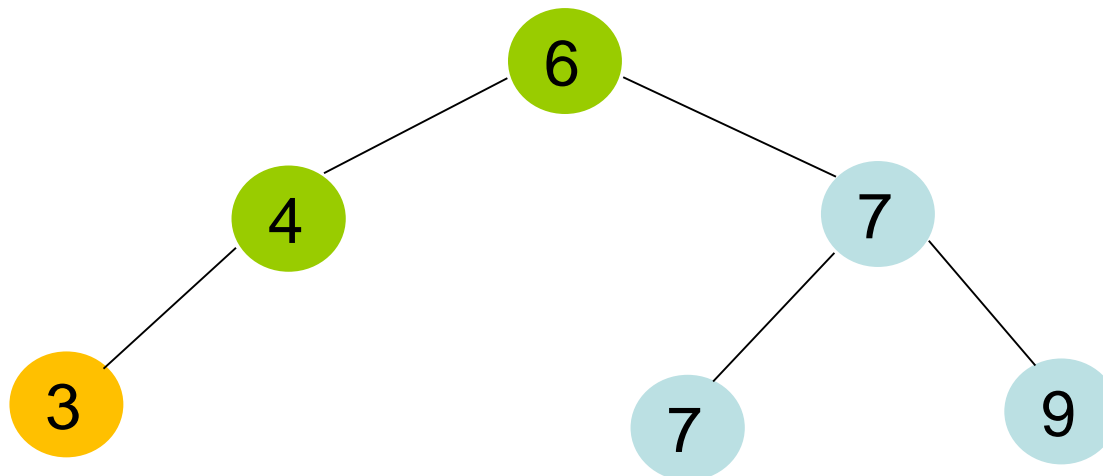
Binäre Suchbäume Durchlaufen

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 Durchlaufen

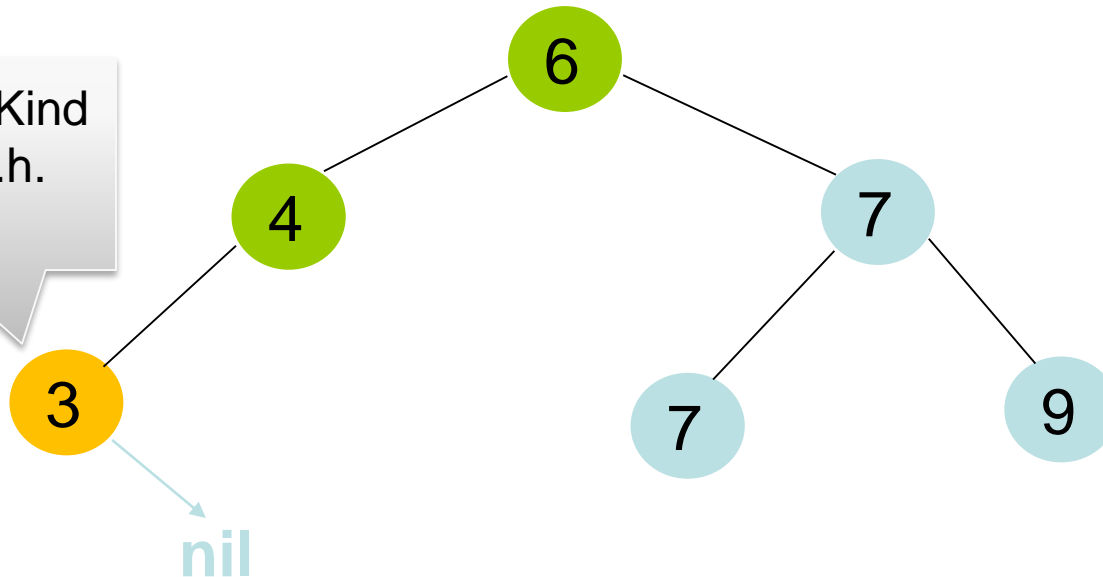
Inorder-Tree-Walk(x)

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.
 $\text{rc}(x) = \text{nil}$



Binäre Suchbäume Durchlaufen

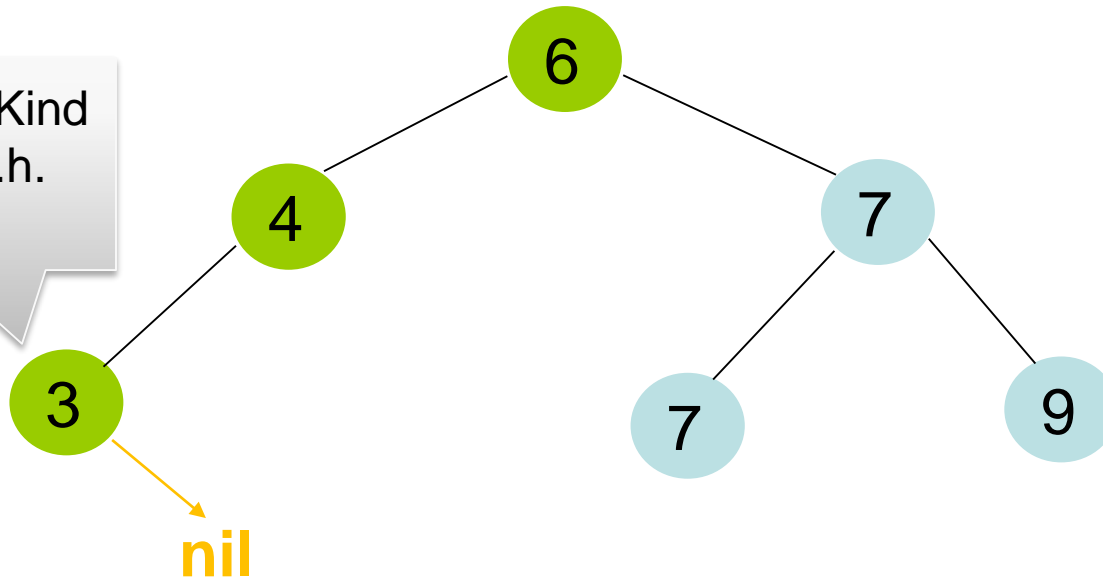
Inorder-Tree-Walk(x)

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.
 $\text{rc}(x) = \text{nil}$



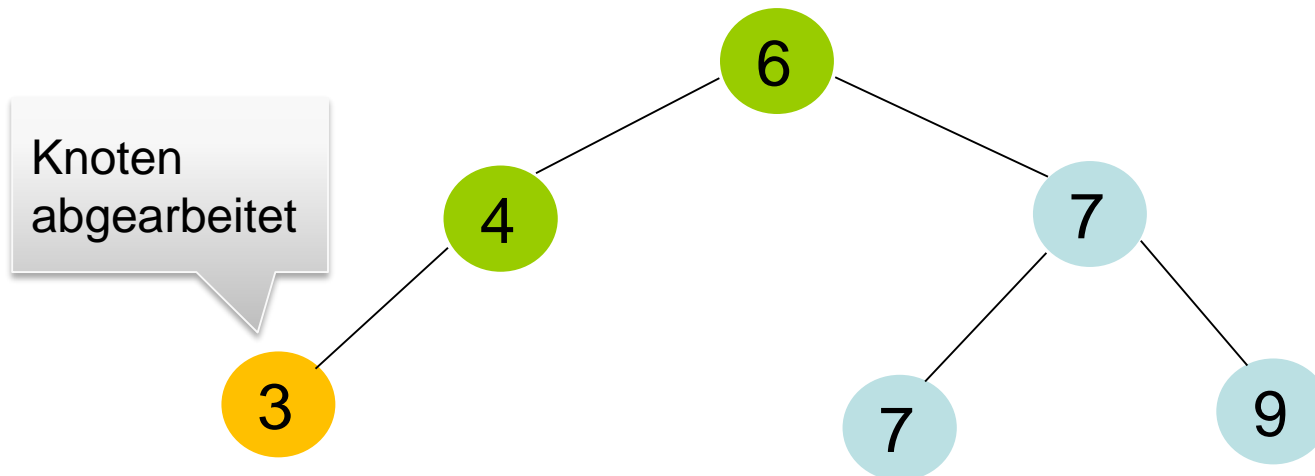
Binäre Suchbäume Durchlaufen

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



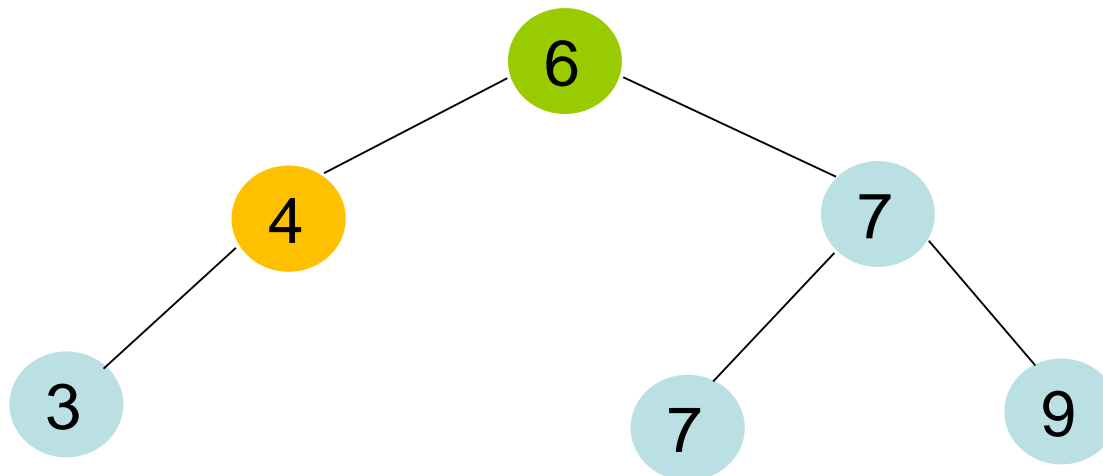
Binäre Suchbäume Durchlaufen

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



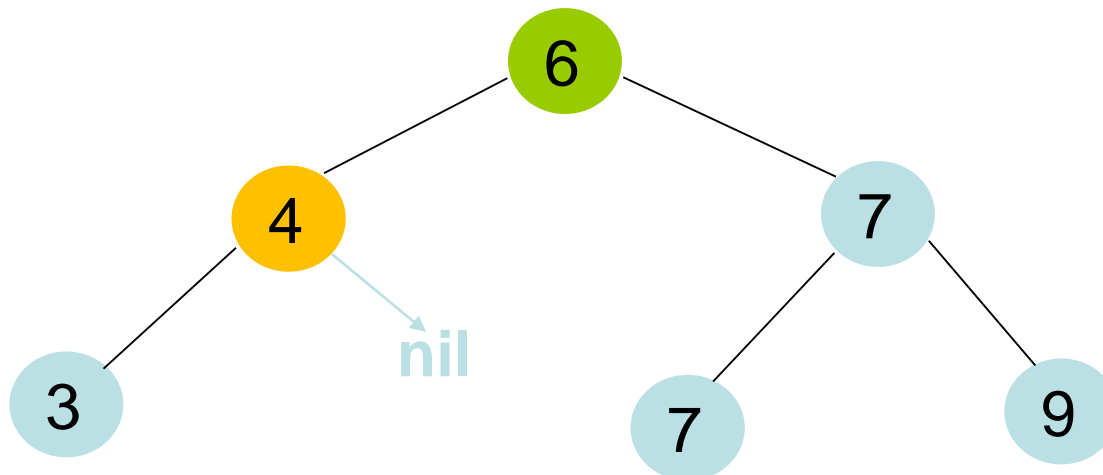
Binäre Suchbäume Durchlaufen

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



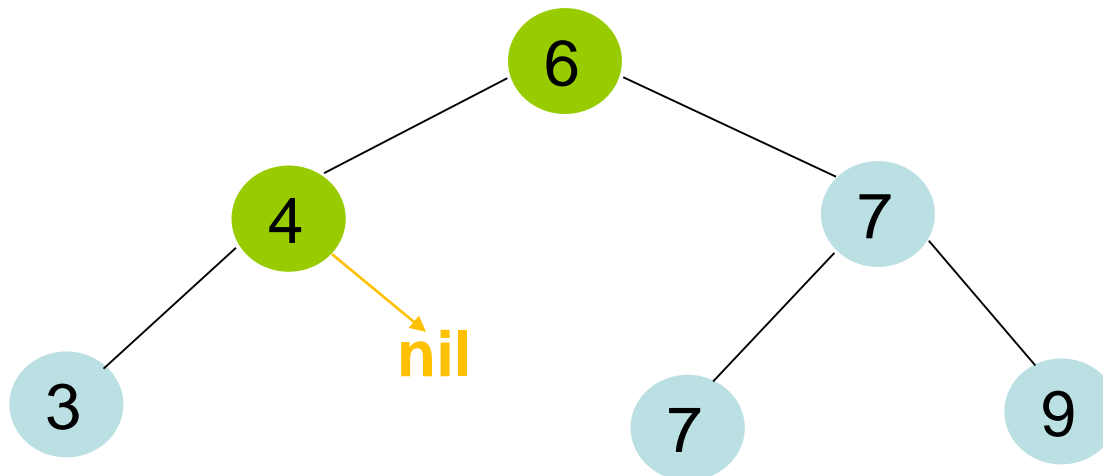
Binäre Suchbäume Durchlaufen

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



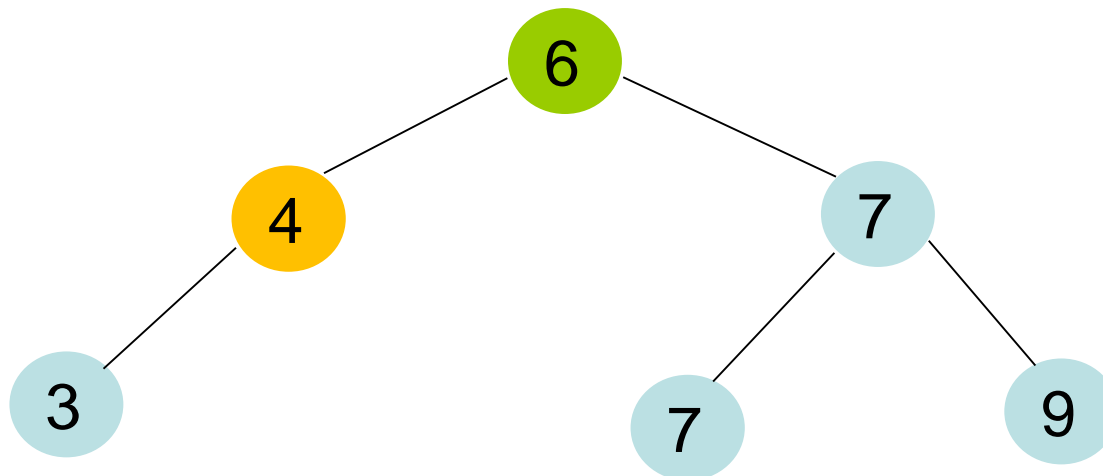
Binäre Suchbäume Durchlaufen

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



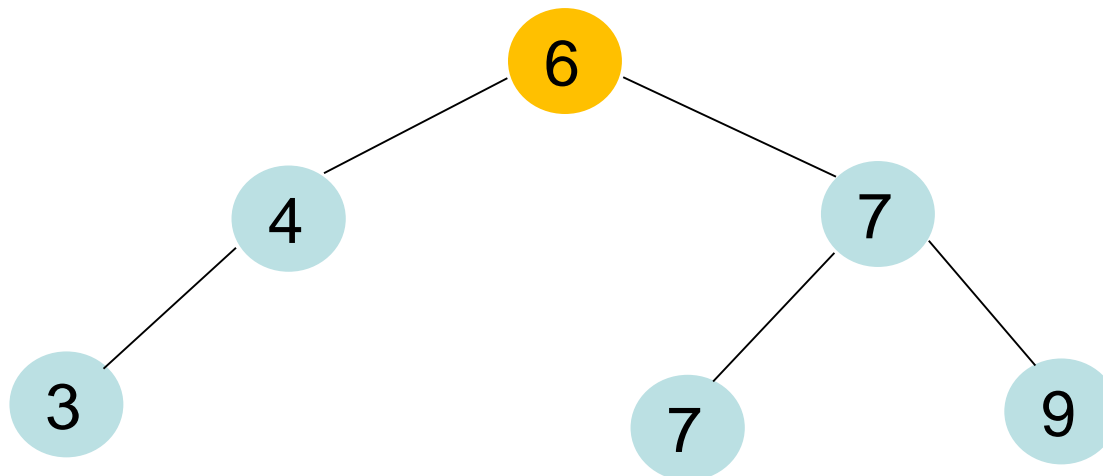
Binäre Suchbäume

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



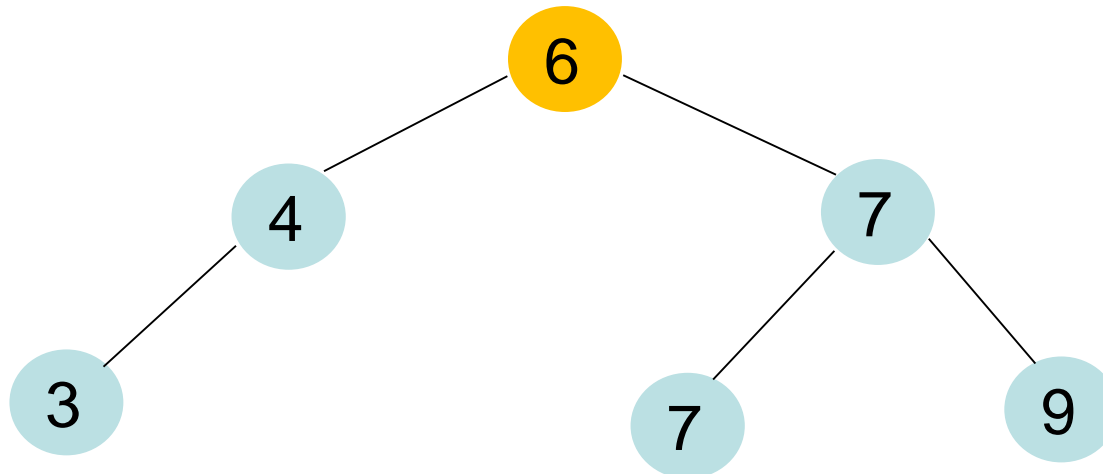
Binäre Suchbäume Durchlaufen

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



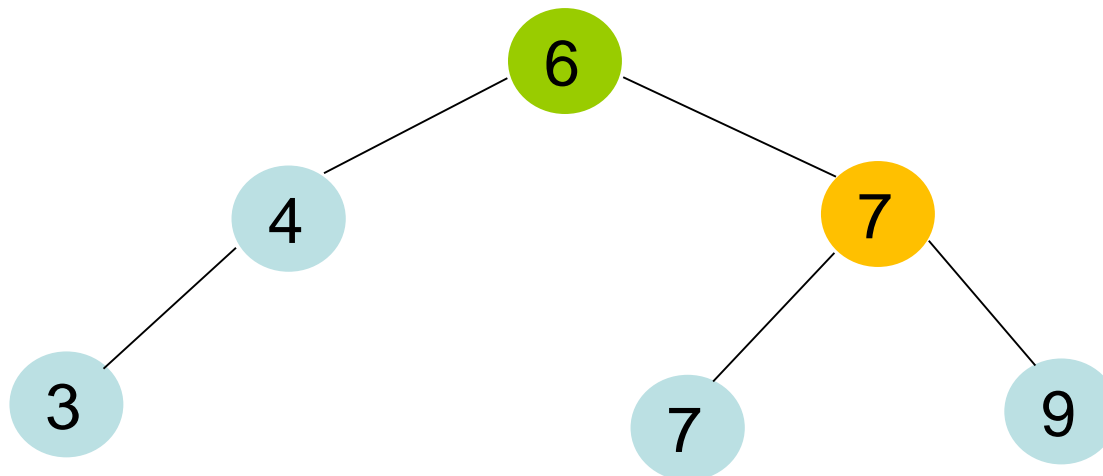
Binäre Suchbäume Durchlaufen

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



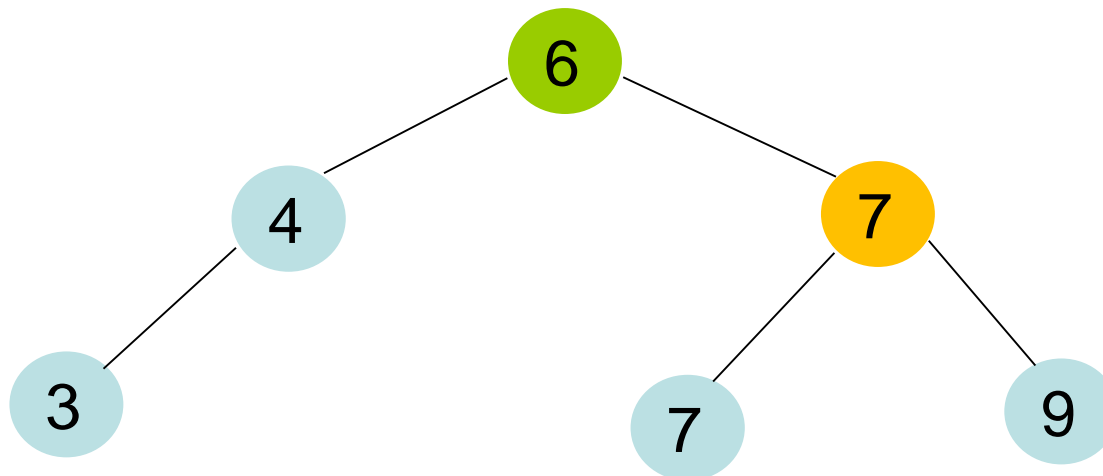
Binäre Suchbäume

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



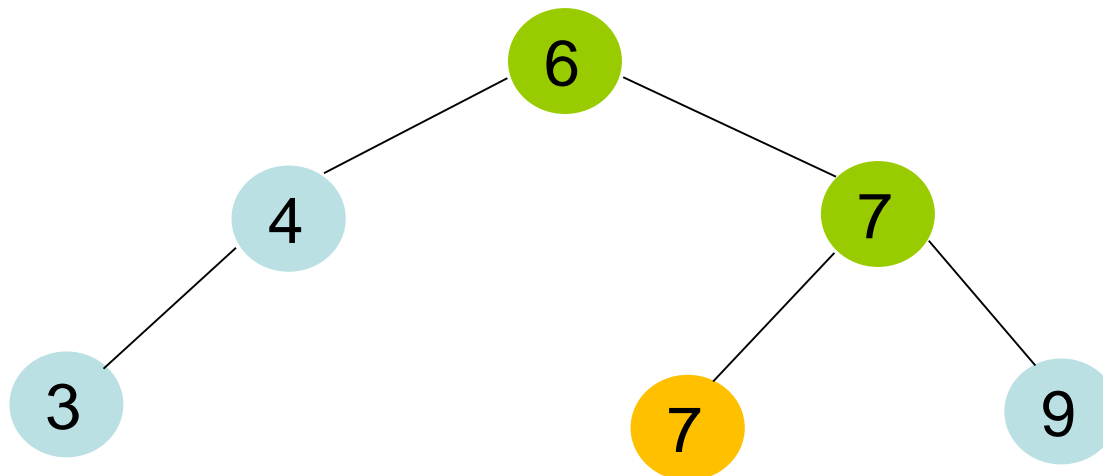
Binäre Suchbäume Durchlaufen

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



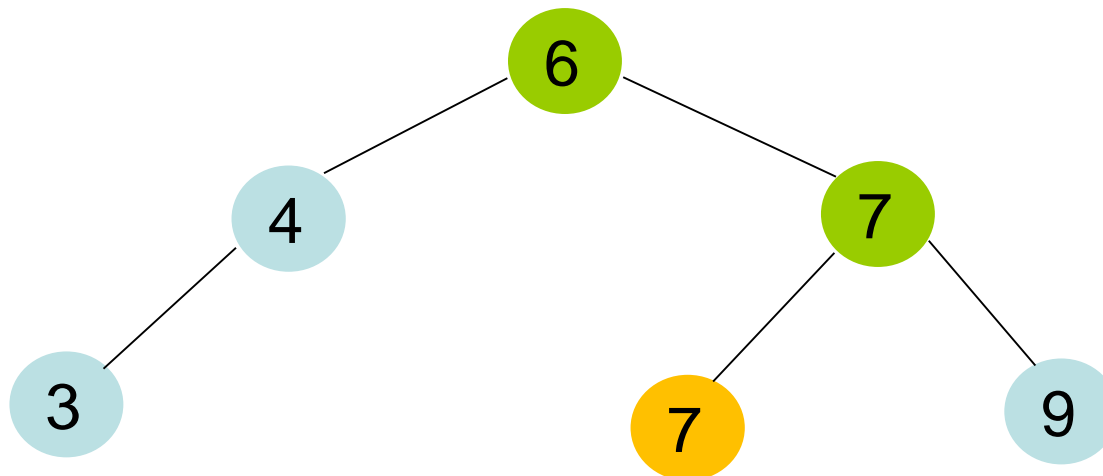
Binäre Suchbäume

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



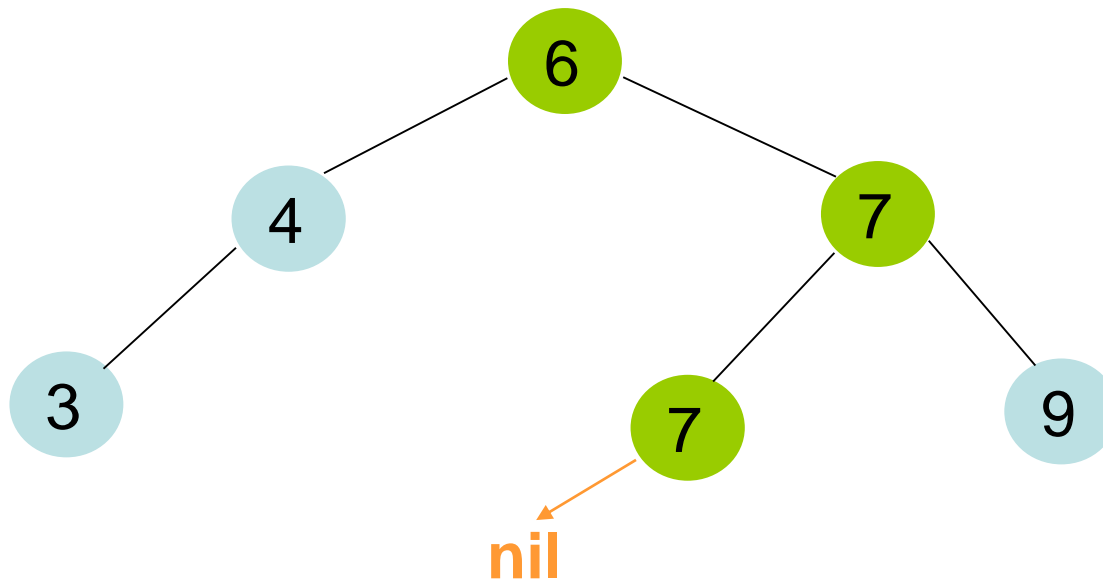
Binäre Suchbäume Durchlaufen

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



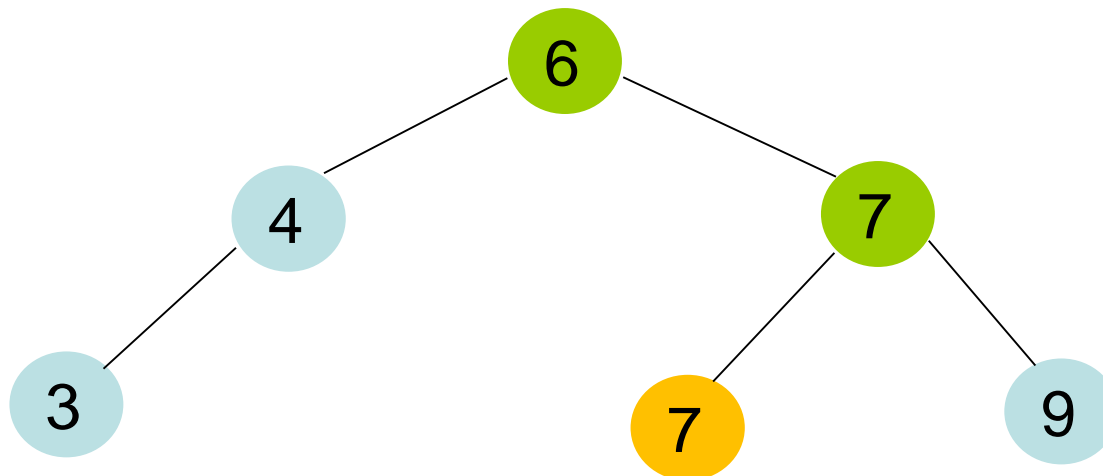
Binäre Suchbäume Durchlaufen

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



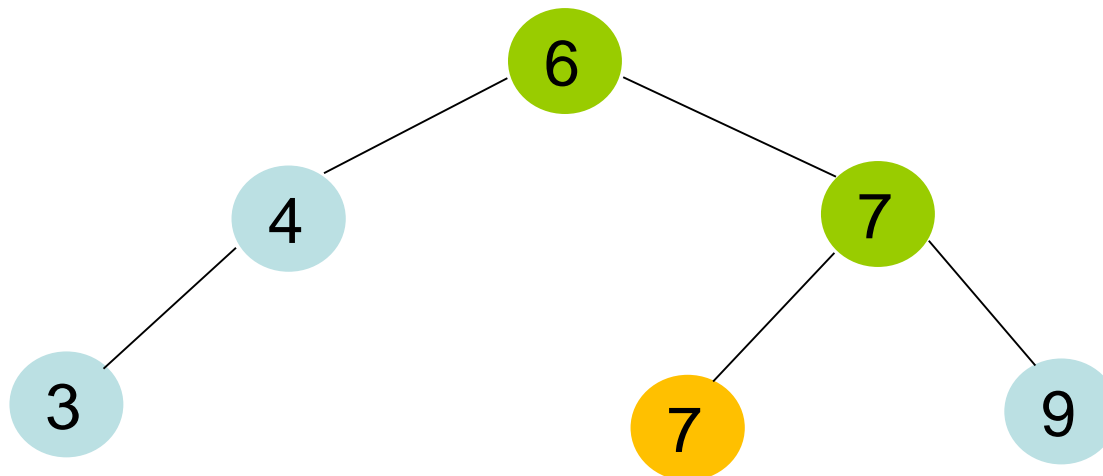
Binäre Suchbäume

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



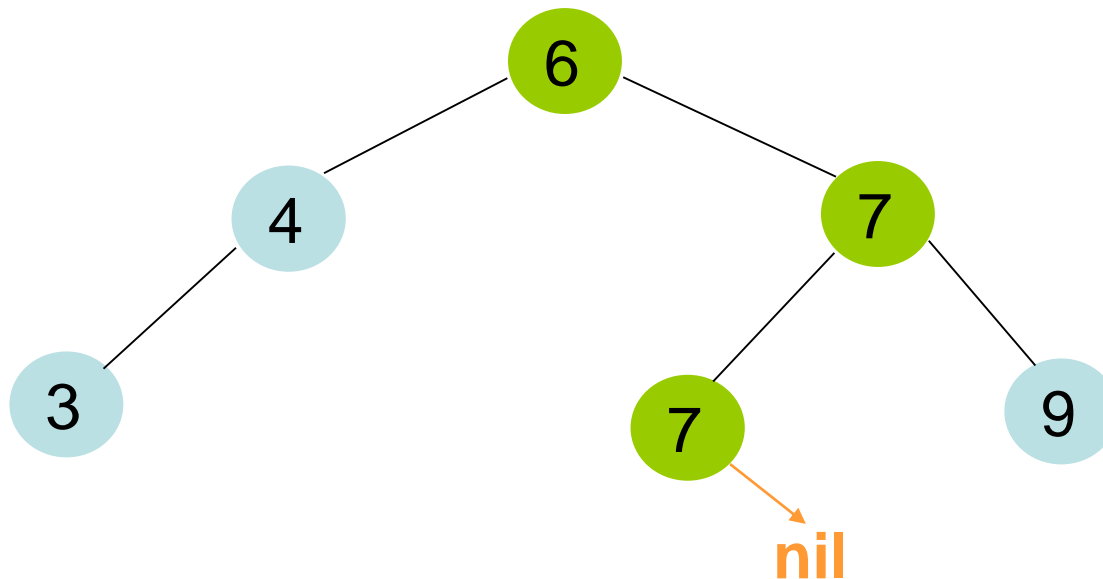
Binäre Suchbäume Durchlaufen

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



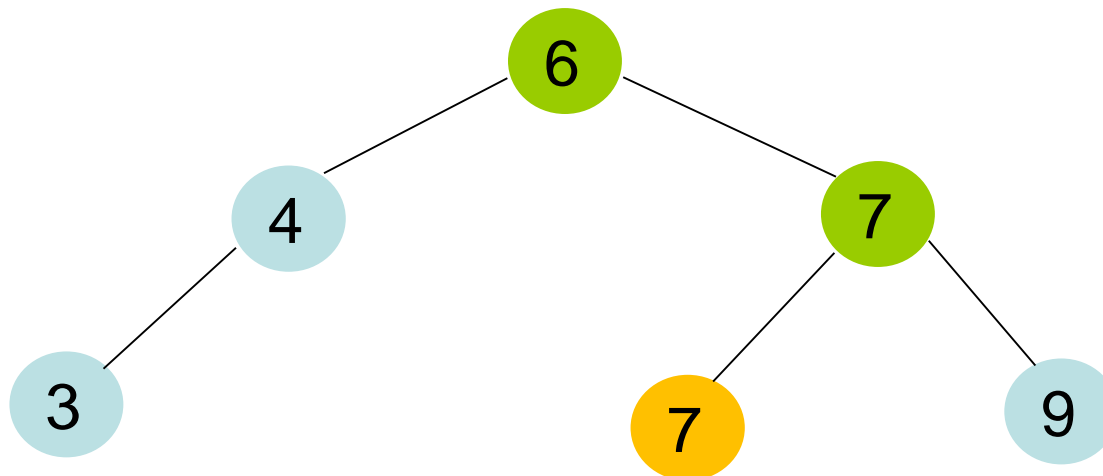
Binäre Suchbäume Durchlaufen

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

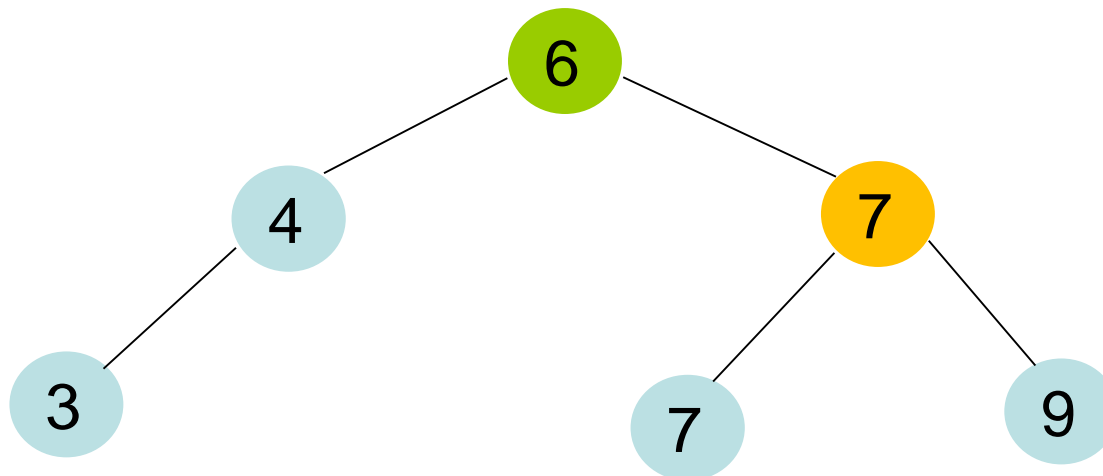


Binäre Suchbäume

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

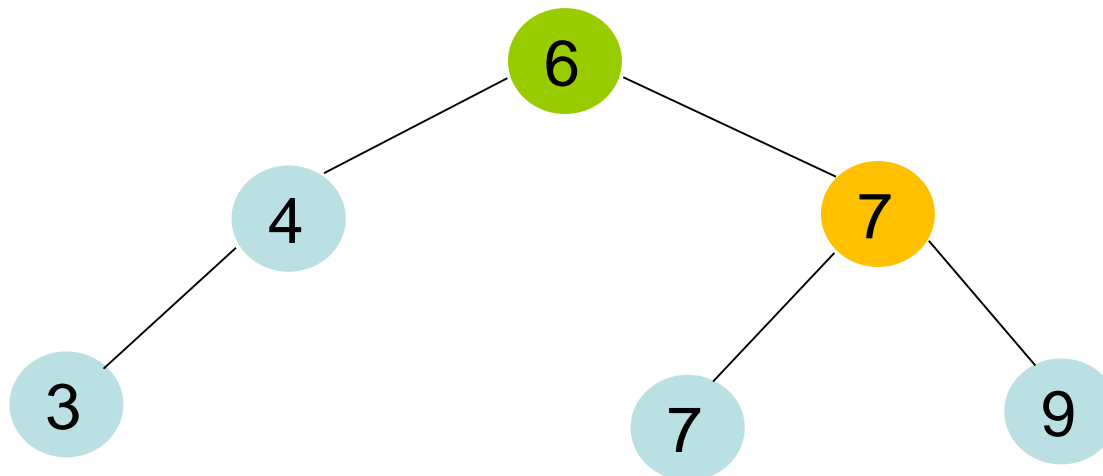


Binäre Suchbäume Durchlaufen

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

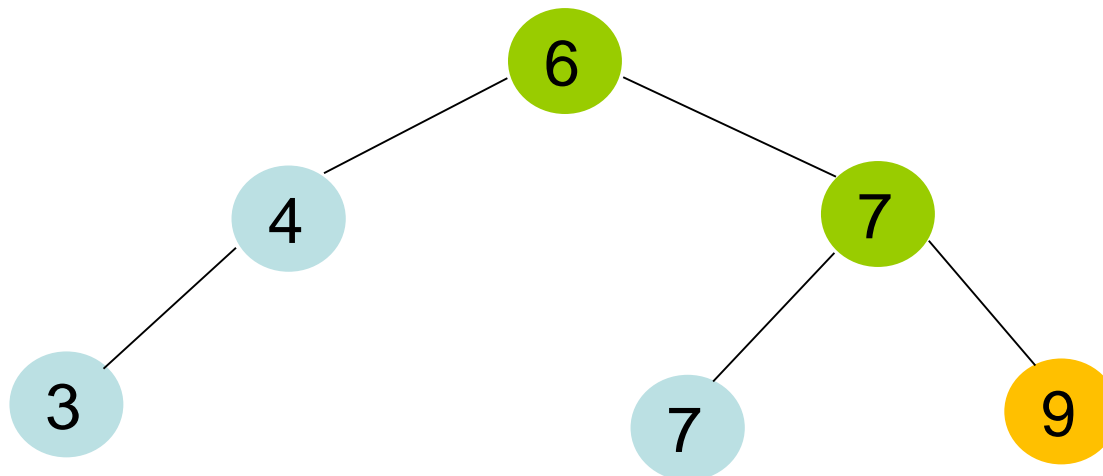


Binäre Suchbäume

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

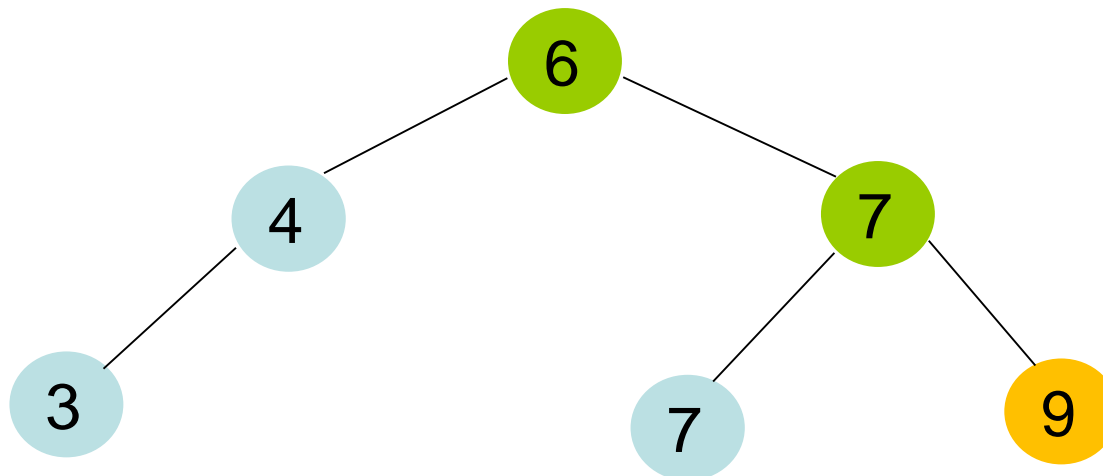


Binäre Suchbäume Durchlaufen

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

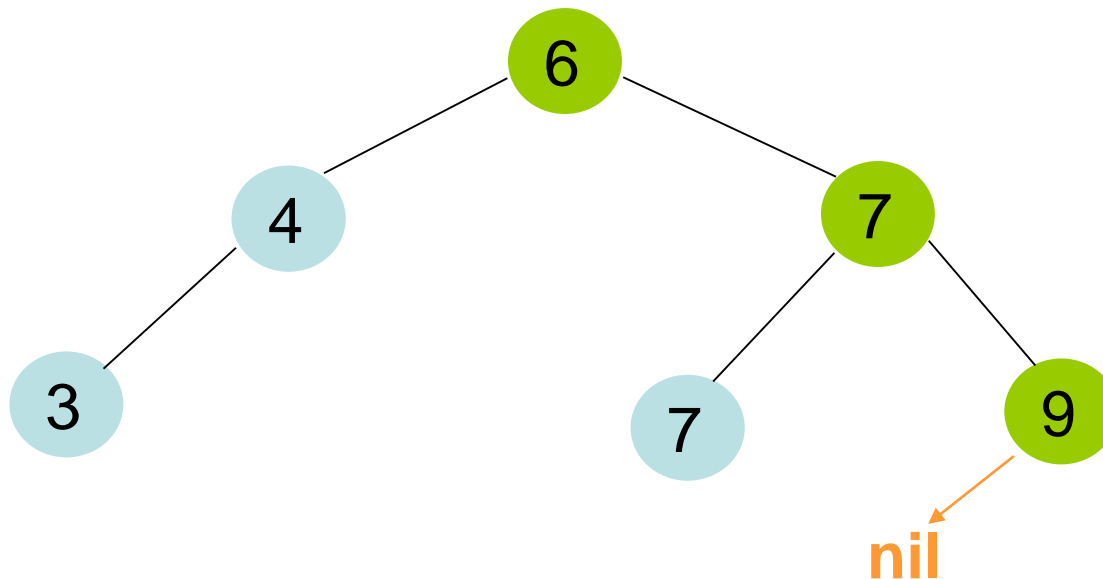


Binäre Suchbäume Durchlaufen

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



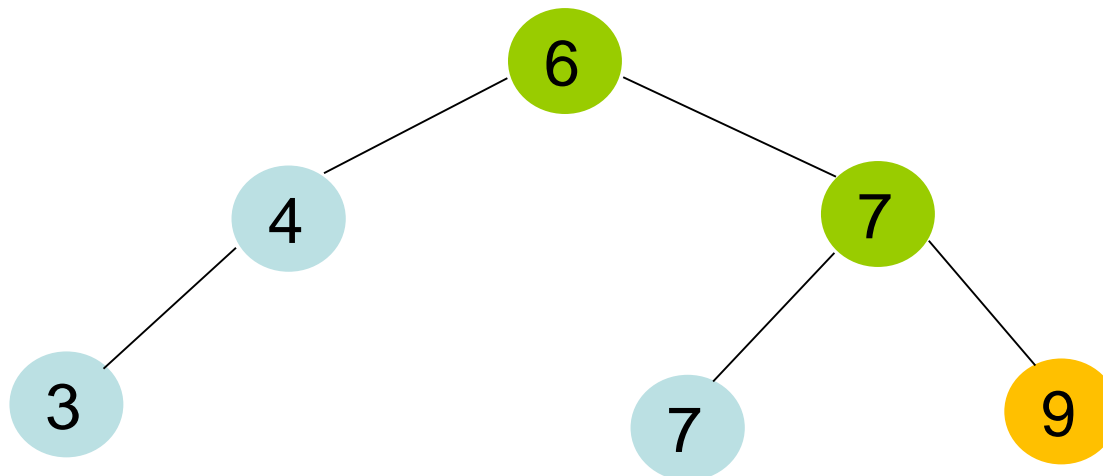
Binäre Suchbäume

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



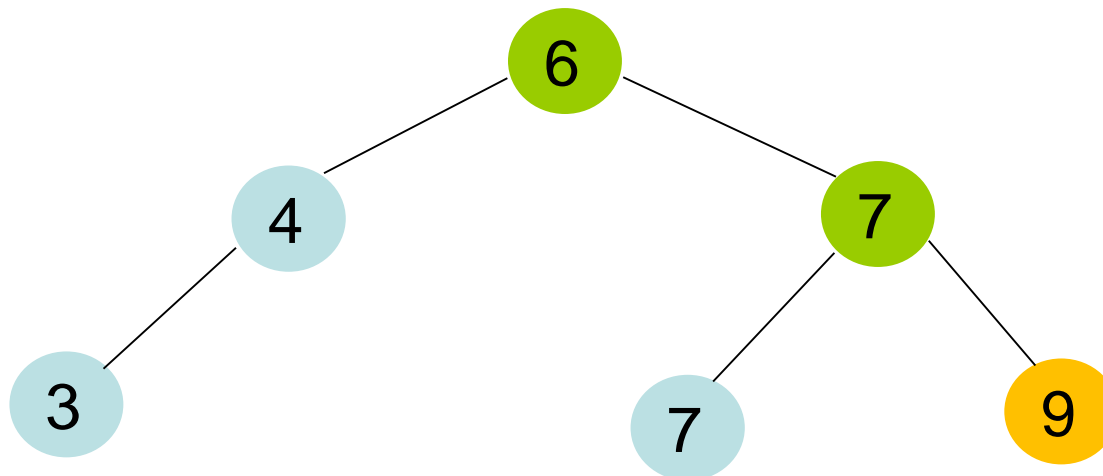
Binäre Suchbäume Durchlaufen

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



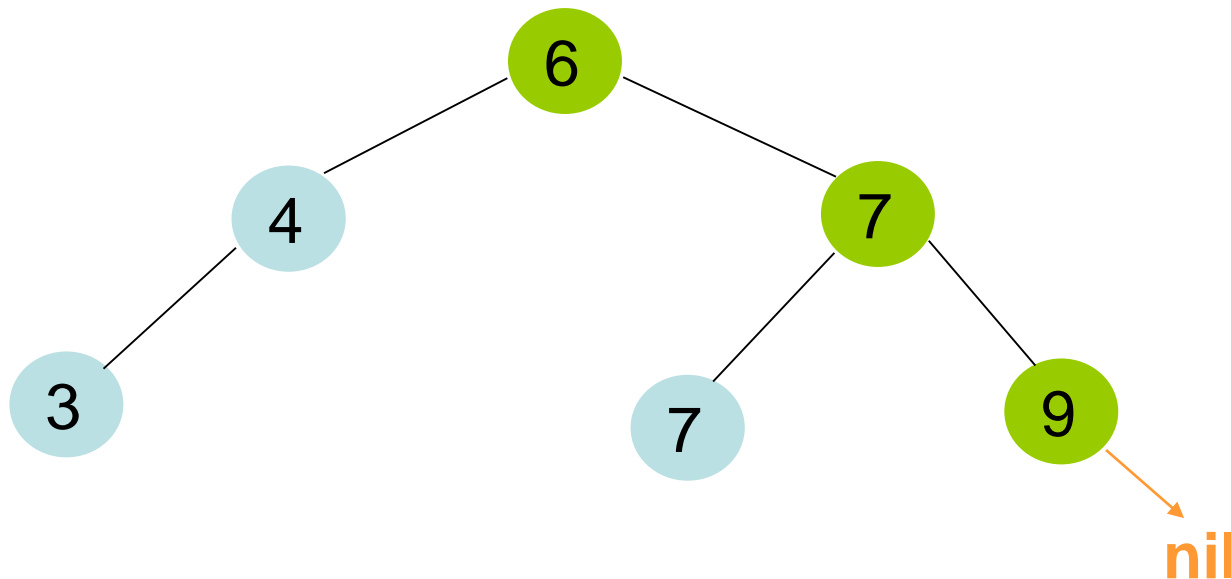
Binäre Suchbäume Durchlaufen

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



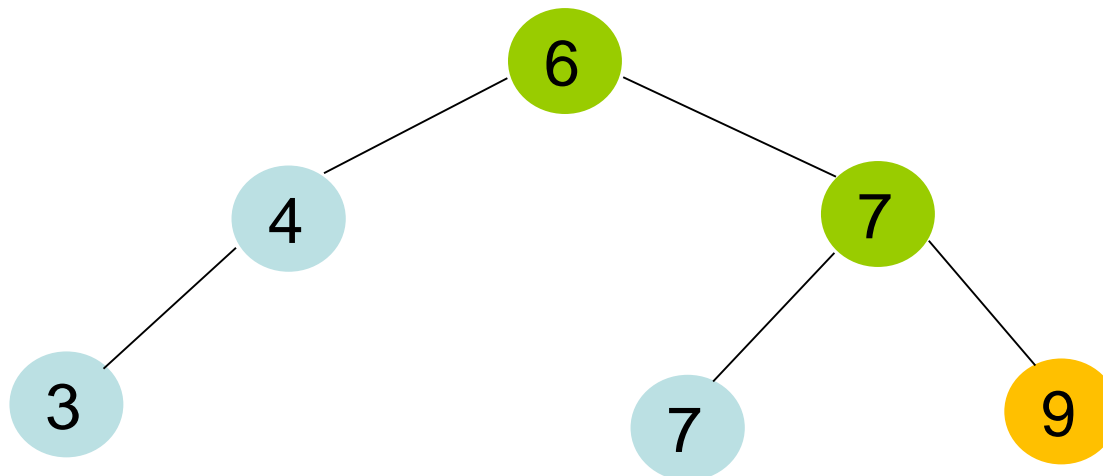
Binäre Suchbäume Durchlaufen

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



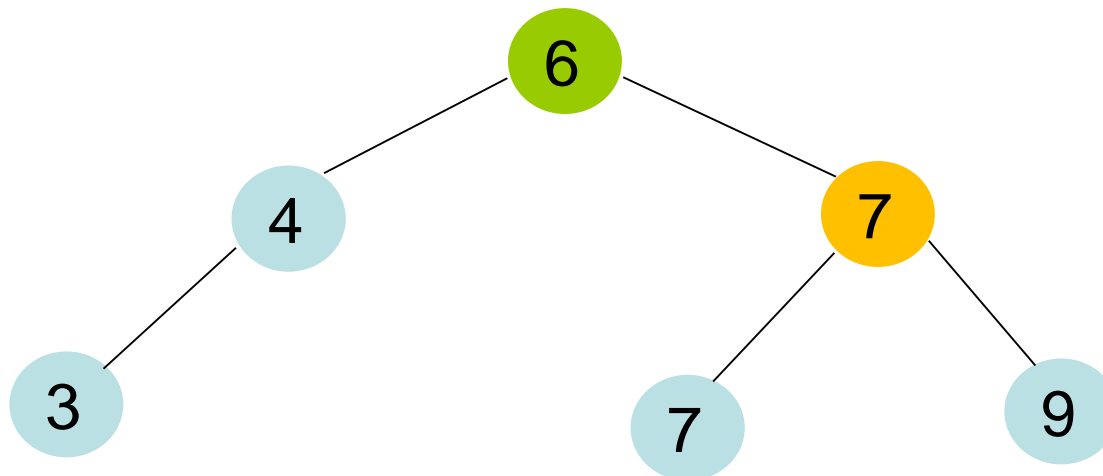
Binäre Suchbäume Durchlaufen

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



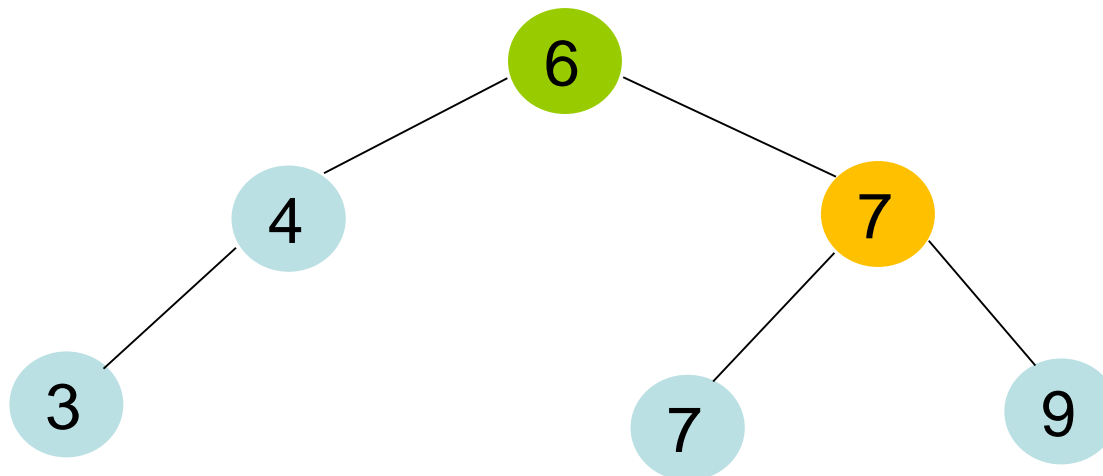
Binäre Suchbäume Durchlaufen

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



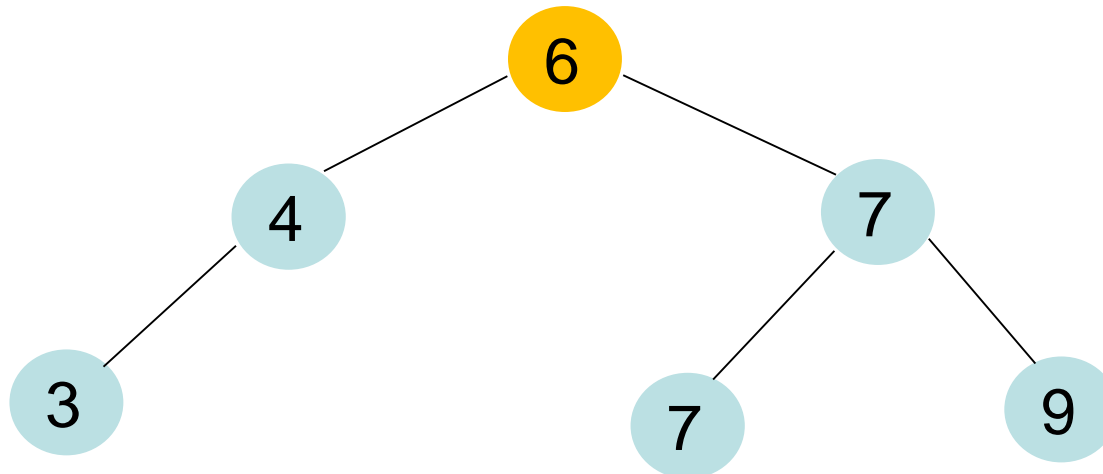
Binäre Suchbäume Durchlaufen

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



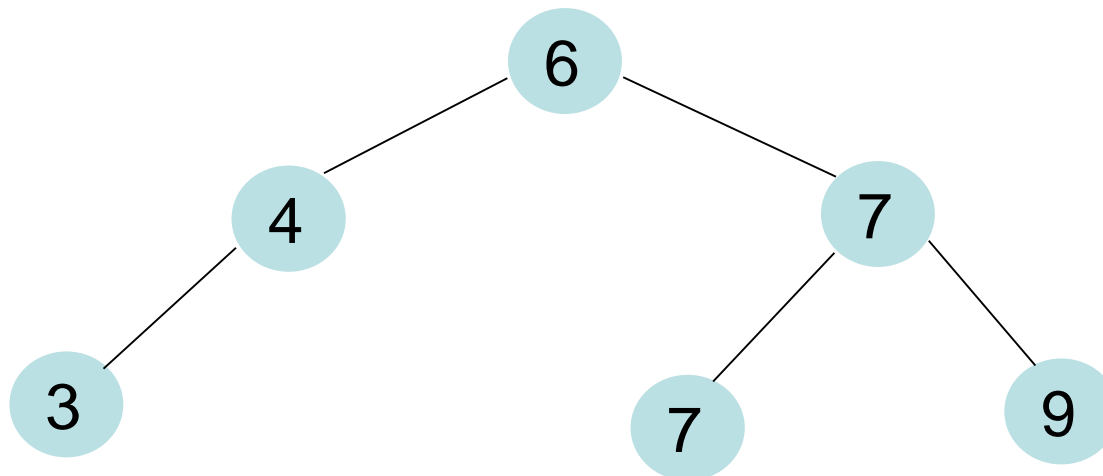
Binäre Suchbäume Durchlaufen

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 Durchlaufen – Laufzeit

Ausgabe aller Schlüssel

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

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))

Anzahl der Elemente: n

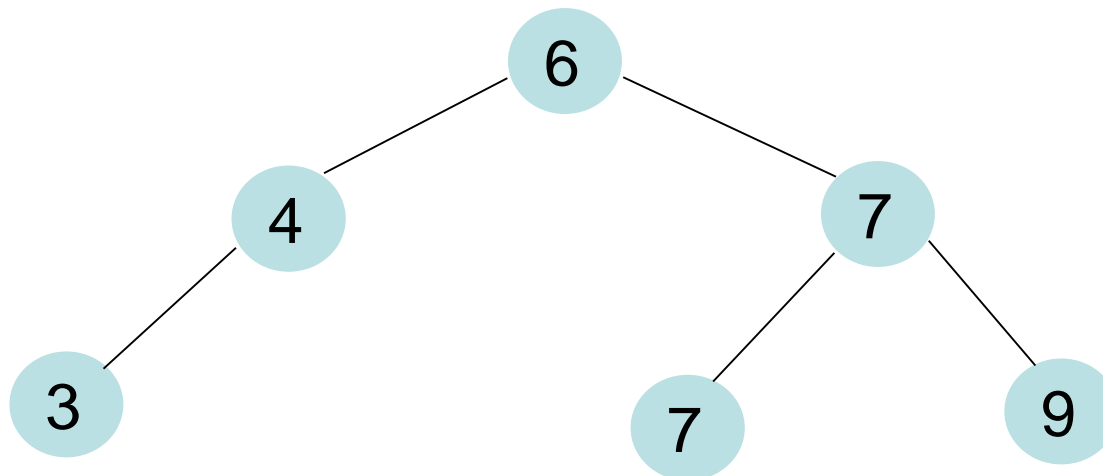
- $n + 2 * \# \text{ Blätter}$
- $< n$
- n
- $< n$

=> Laufzeit $\Theta(n)$

Binäre Suchbäume Suchen

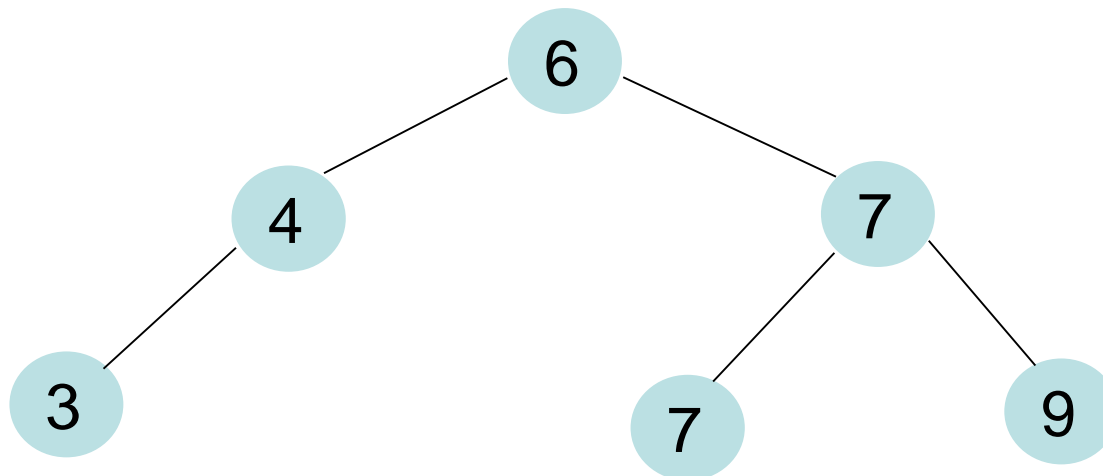
Suchen in Binärbäumen

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



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** $\text{Baumsuche}(\text{rc}(x), k)$

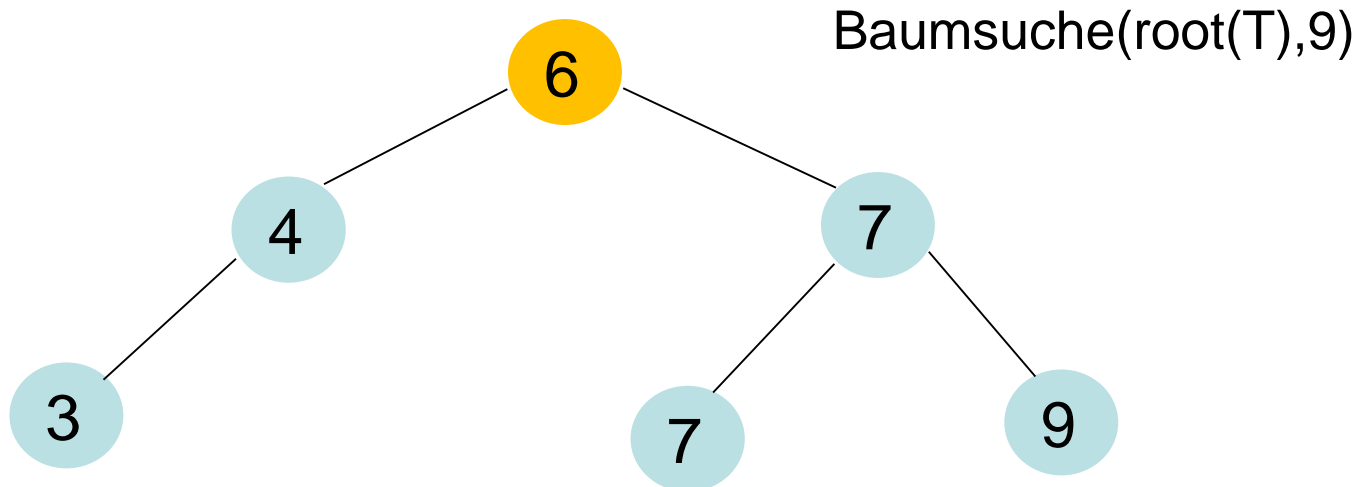


Binäre Suchbäume Suchen

Baumsuche(x,k)

Aufruf mit
x=root(T)

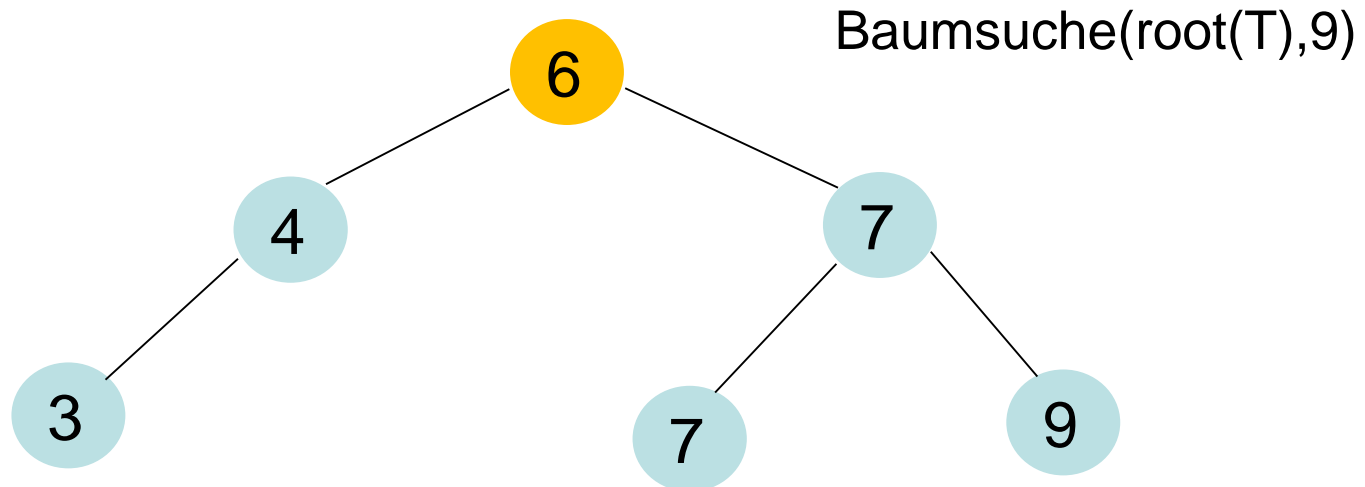
1. **if** x=nil **or** k=key(x) **then return** x
2. **if** k<key(x) **then return** Baumsuche(lc(x),k)
3. **else** return Baumsuche(rc(x),k)



Binäre Suchbäume Suchen

Baumsuche(x,k)

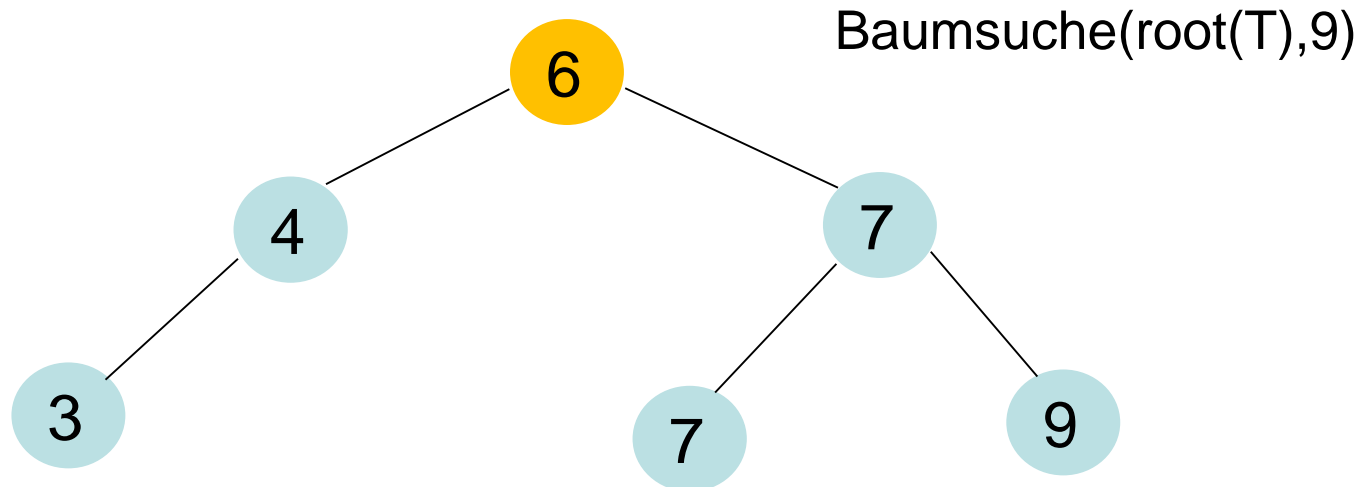
1. **if** $x = \text{nil}$ **or** $k = \text{key}(x)$ **then return** x
2. **if** $k < \text{key}(x)$ **then return** $\text{Baumsuche}(\text{lc}(x), k)$
3. **else** **return** $\text{Baumsuche}(\text{rc}(x), k)$



Binäre Suchbäume Suchen

Baumsuche(x,k)

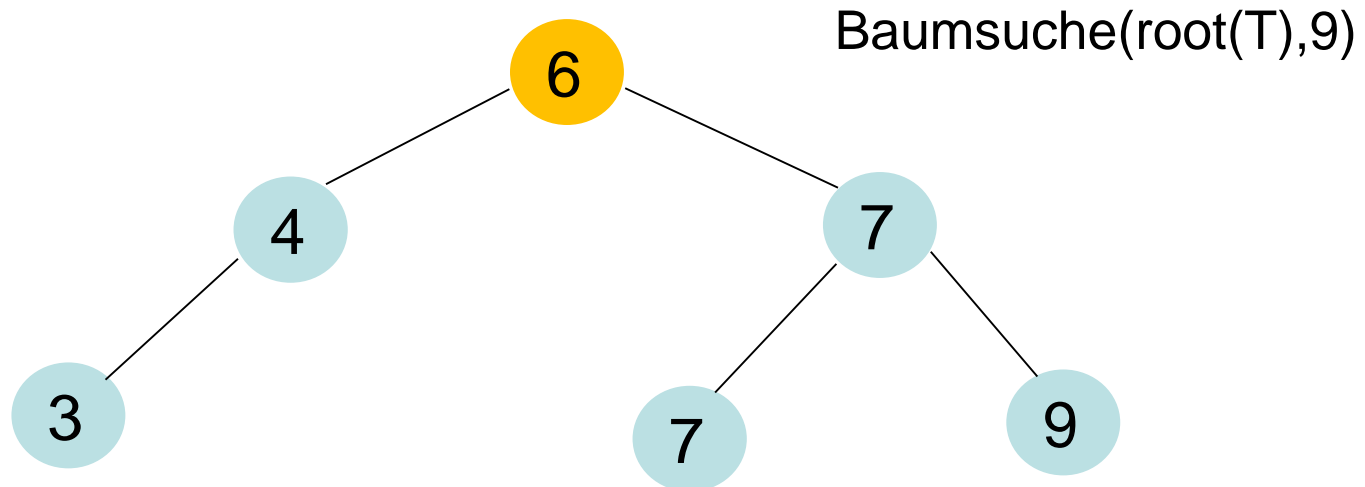
1. **if** $x = \text{nil}$ **or** $k = \text{key}(x)$ **then return** x
2. **if** $k < \text{key}(x)$ **then return** $\text{Baumsuche}(\text{lc}(x), k)$
3. **else** **return** $\text{Baumsuche}(\text{rc}(x), k)$



Binäre Suchbäume Suchen

Baumsuche(x,k)

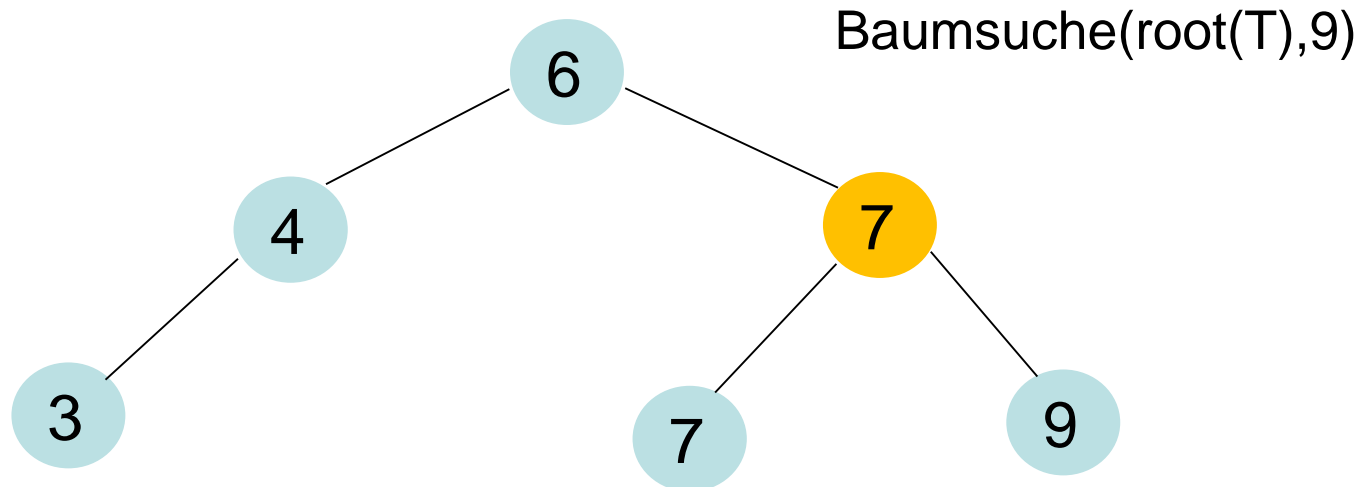
1. if $x = \text{nil}$ or $k = \text{key}(x)$ then return x
2. if $k < \text{key}(x)$ then return Baumsuche(lc(x),k)
3. else return Baumsuche(rc(x),k)



Binäre Suchbäume Suchen

Baumsuche(x,k)

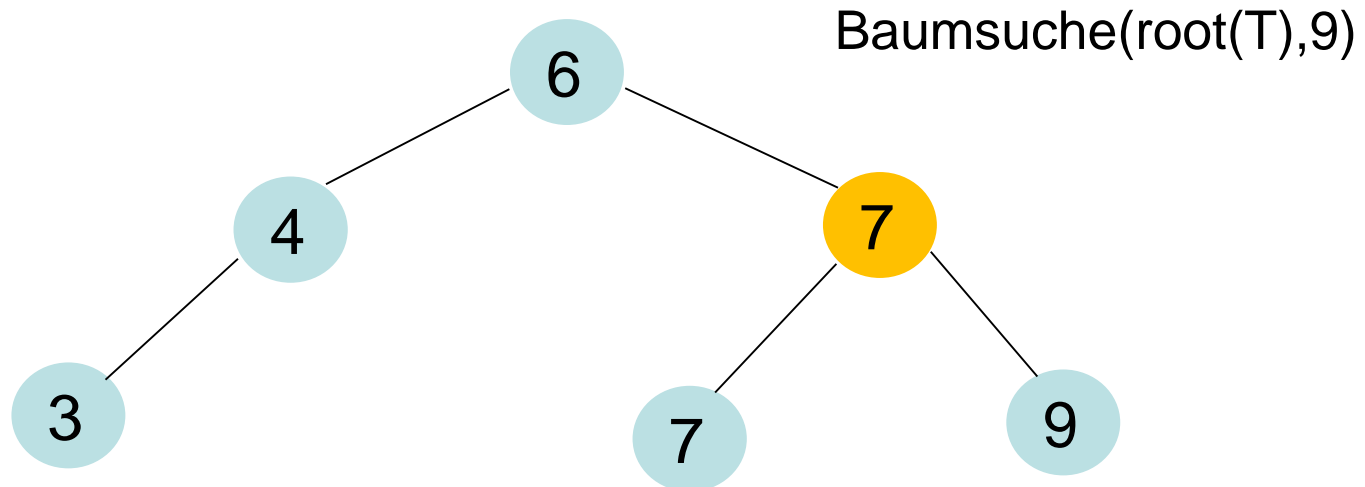
1. **if** $x = \text{nil}$ **or** $k = \text{key}(x)$ **then return** x
2. **if** $k < \text{key}(x)$ **then return** $\text{Baumsuche}(\text{lc}(x), k)$
3. **else return** $\text{Baumsuche}(\text{rc}(x), k)$



Binäre Suchbäume Suchen

Baumsuche(x,k)

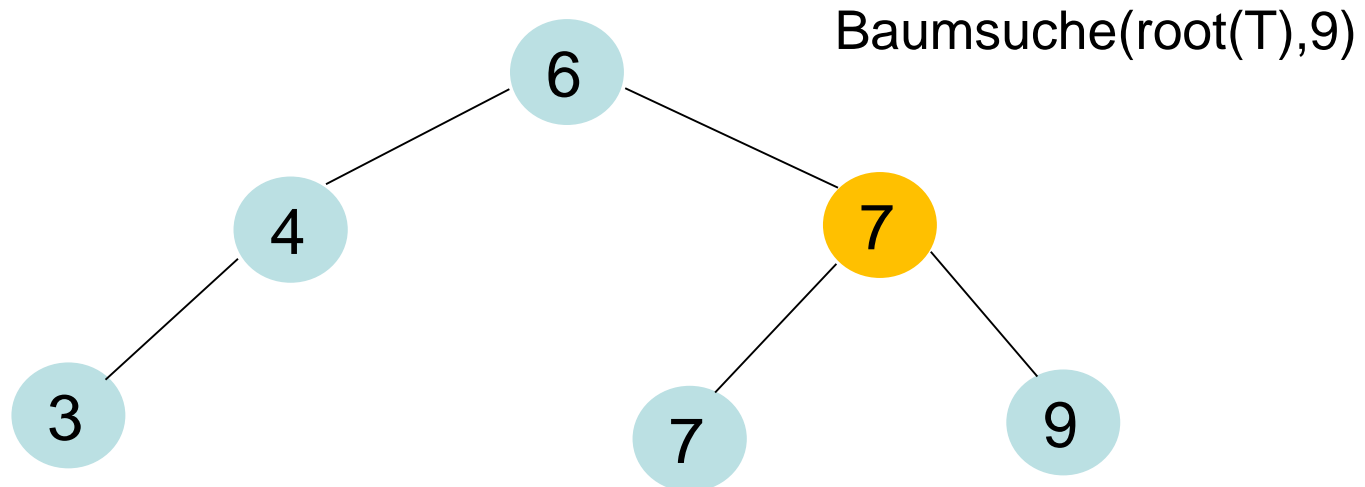
1. **if** $x = \text{nil}$ **or** $k = \text{key}(x)$ **then return** x
2. **if** $k < \text{key}(x)$ **then return** $\text{Baumsuche}(\text{lc}(x), k)$
3. **else return** $\text{Baumsuche}(\text{rc}(x), k)$



Binäre Suchbäume Suchen

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)

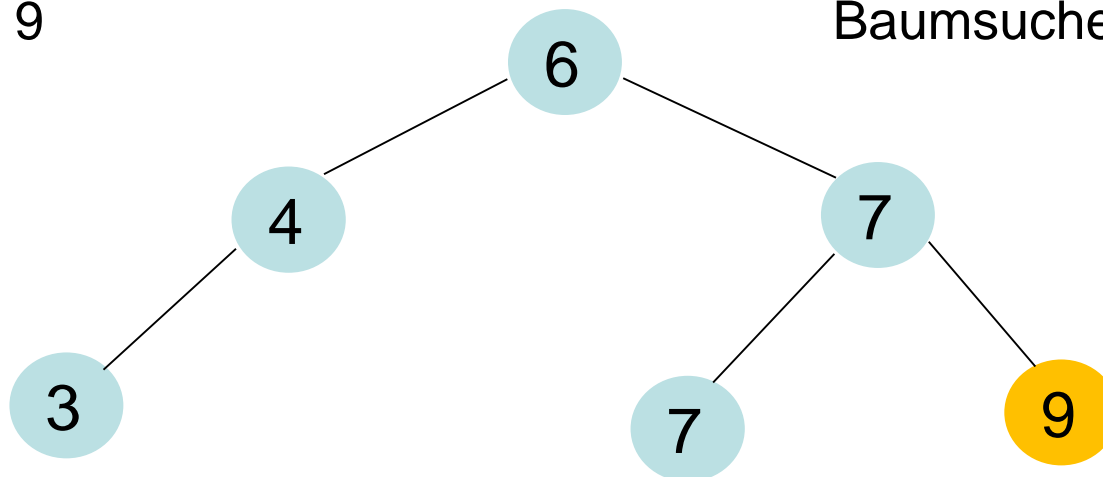


Binäre Suchbäume Suchen

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)**

Ausgabe: 9



Baumsuche(root(T),9)

Binäre Suchbäume

Suchen – Laufzeit?

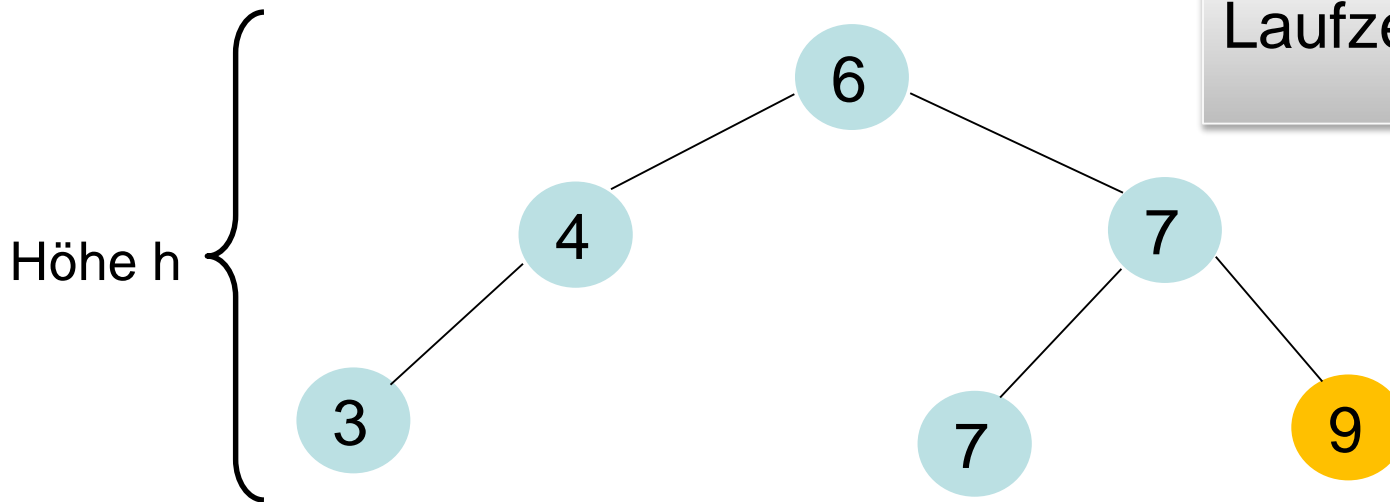
Baumsuche(x,k)

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** $\text{Baumsuche}(\text{rc}(x), k)$

// Entweder links

// oder rechts

Laufzeit: $O(\text{Höhe})$



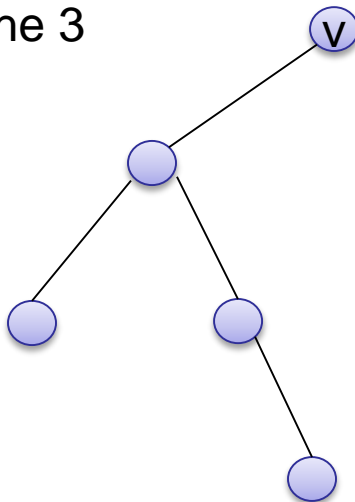
Höhe eines Baumes

Definition

- Die **Höhe eines Binärbaums** mit Wurzel v ist die Länge (Anzahl Knoten) des längsten einfachen Weges (keine mehrfach vorkommenden Knoten) von der Wurzel zu einem Blatt.

Beispiel

- Baum der Höhe 3



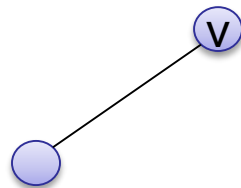
Höhe eines Baumes

Definition

- Die **Höhe eines Binärbaums** mit Wurzel v ist die Länge (Anzahl Knoten) des längsten einfachen Weges (keine mehrfach vorkommenden Knoten) von der Wurzel zu einem Blatt.

Beispiel

- Baum der Höhe 1



Höhe eines Baumes

Definition

- Die **Höhe eines Binärbaums** mit Wurzel v ist die Länge (Anzahl Knoten) des längsten einfachen Weges (keine mehrfach vorkommenden Knoten) von der Wurzel zu einem Blatt.

Beispiel

- Baum der Höhe 0



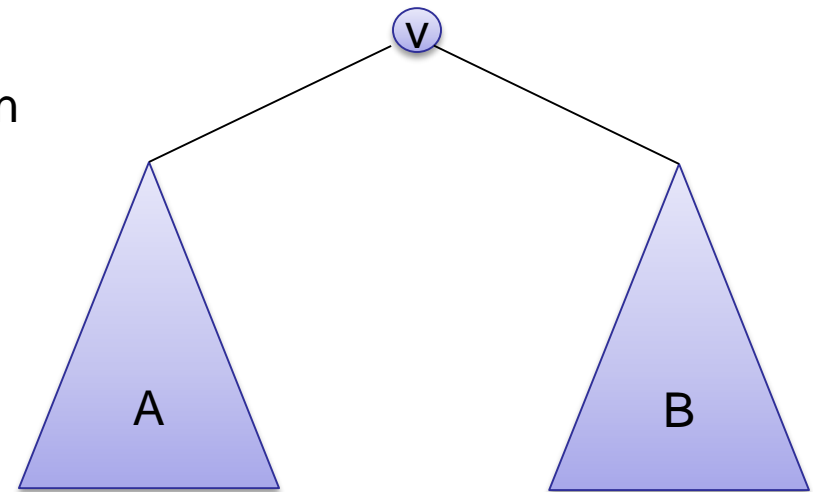
Höhe eines Baumes

Definition

- Die **Höhe eines Binärbaums** mit Wurzel v ist die Länge (Anzahl Knoten) des längsten einfachen Weges (keine mehrfach vorkommenden Knoten) von der Wurzel zu einem Blatt.

Beispiel

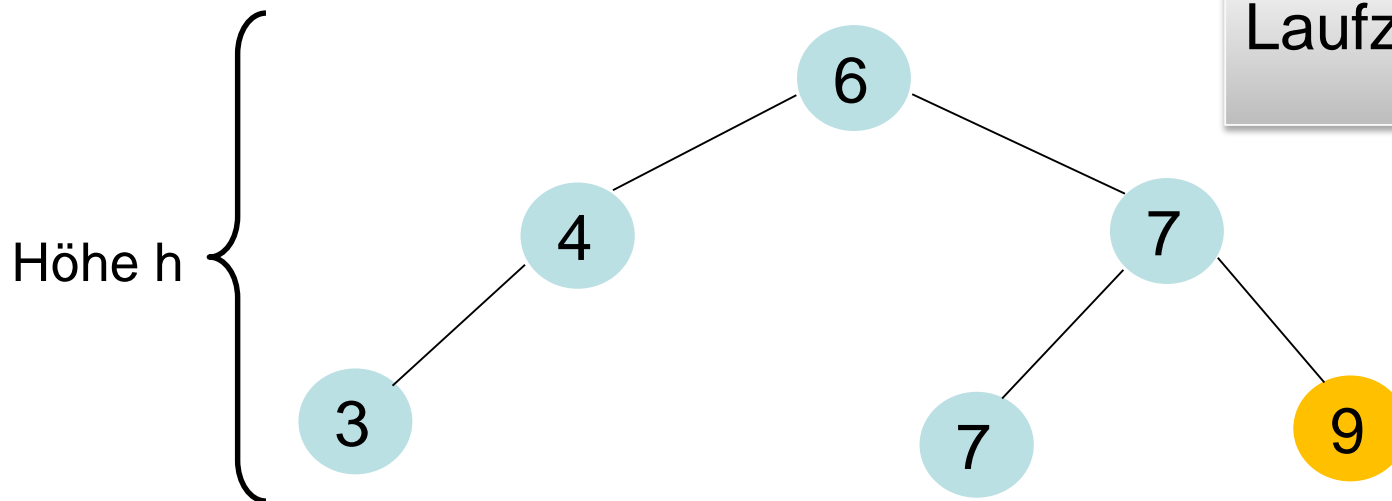
- Übereinkunft: Ein Baum mit einem Knoten hat Höhe 0
- Damit gilt:
Höhe eines Baumes mit Wurzel v und Teilbäumen A und B ist
 $1 + \max\{\text{Höhe}(A), \text{Höhe}(B)\}$



Binäre Suchbäume Suchen

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** $\text{Baumsuche}(\text{rc}(x), k)$

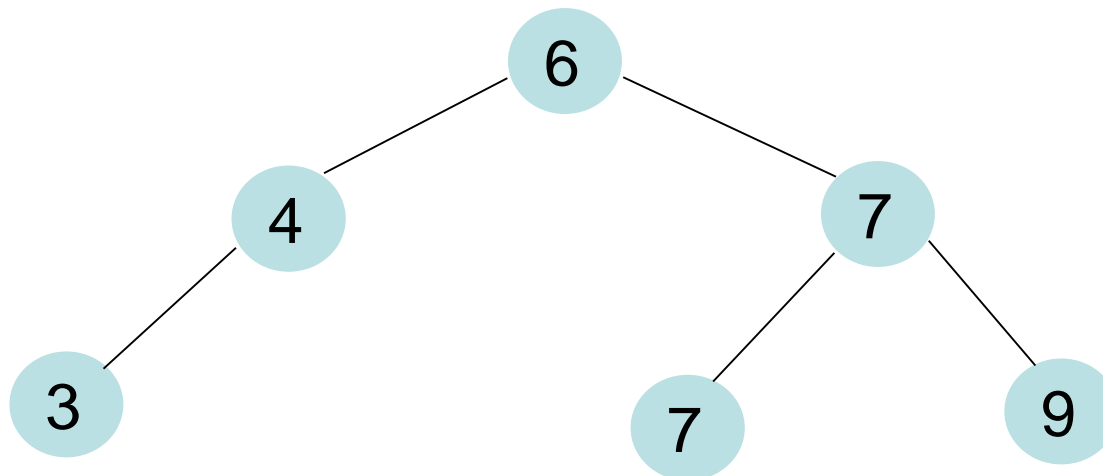


Laufzeit: $O(h)$

Binäre Suchbäume Suchen

IterativeBaumsuche(x, k)

1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x

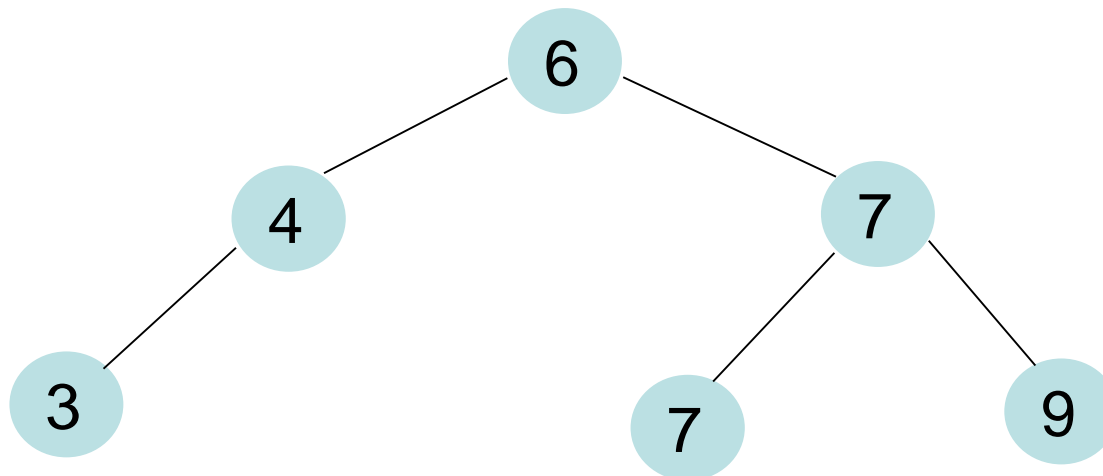


Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x

Aufruf mit
 $x = \text{root}(T)$

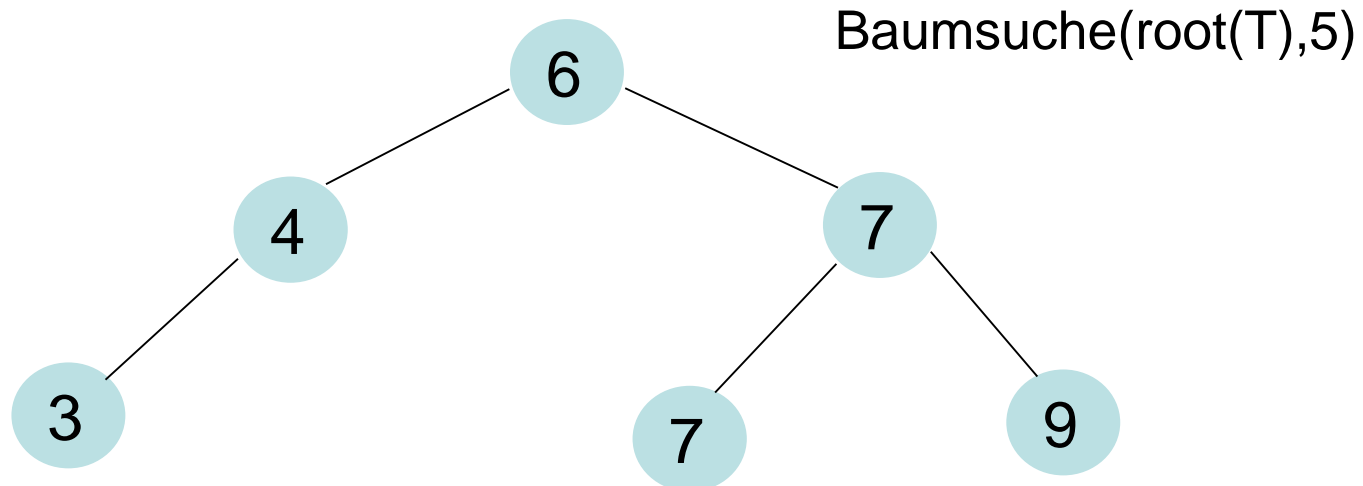


Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x

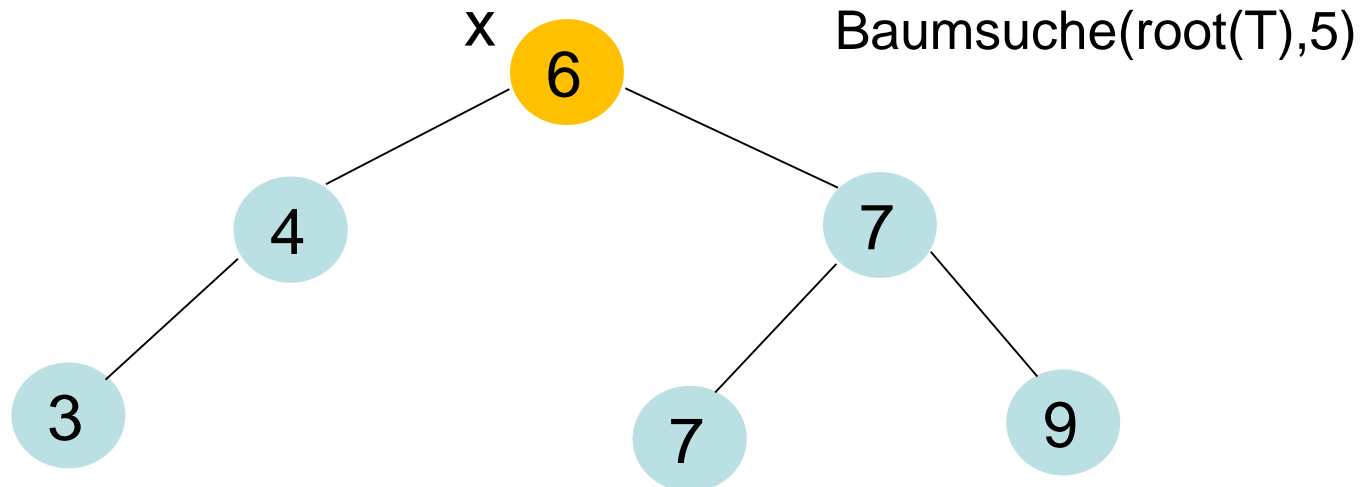
Aufruf mit
 $x = \text{root}(T)$



Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

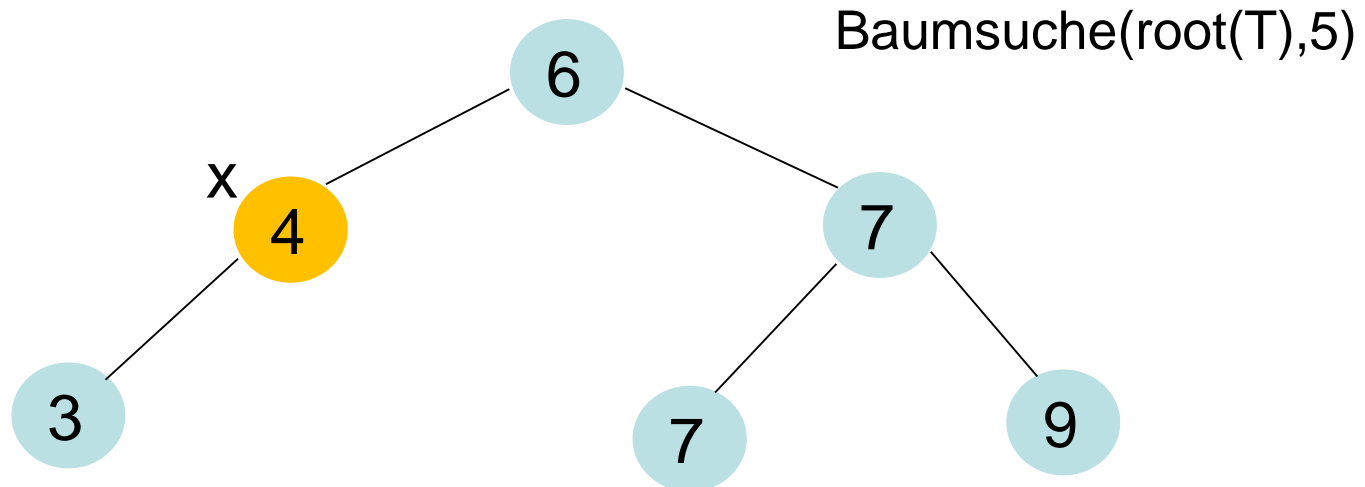
1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x



Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

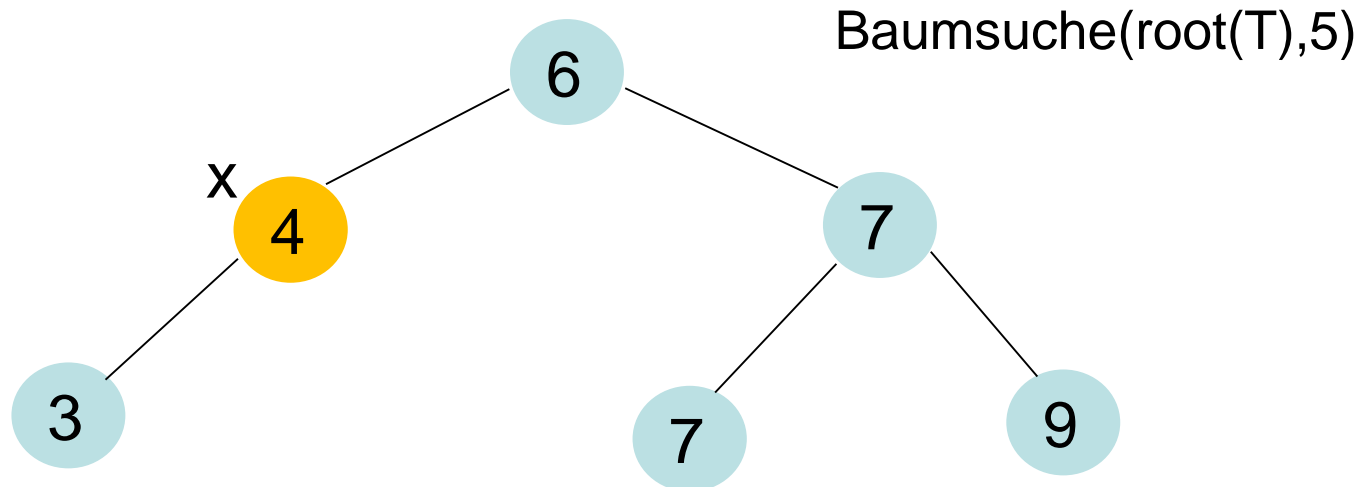
1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x



Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x

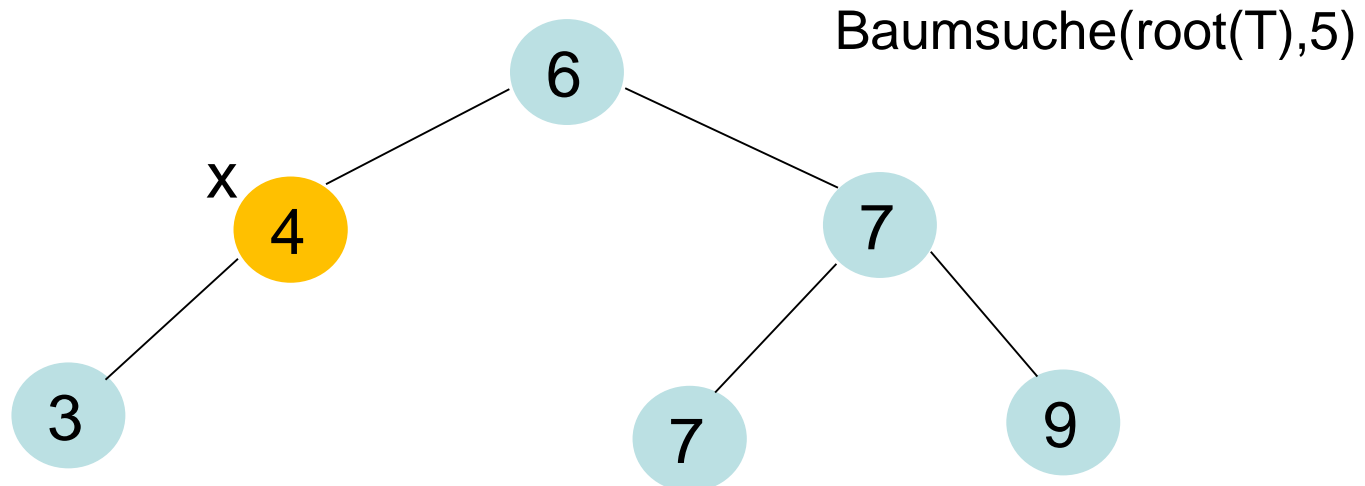


Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x

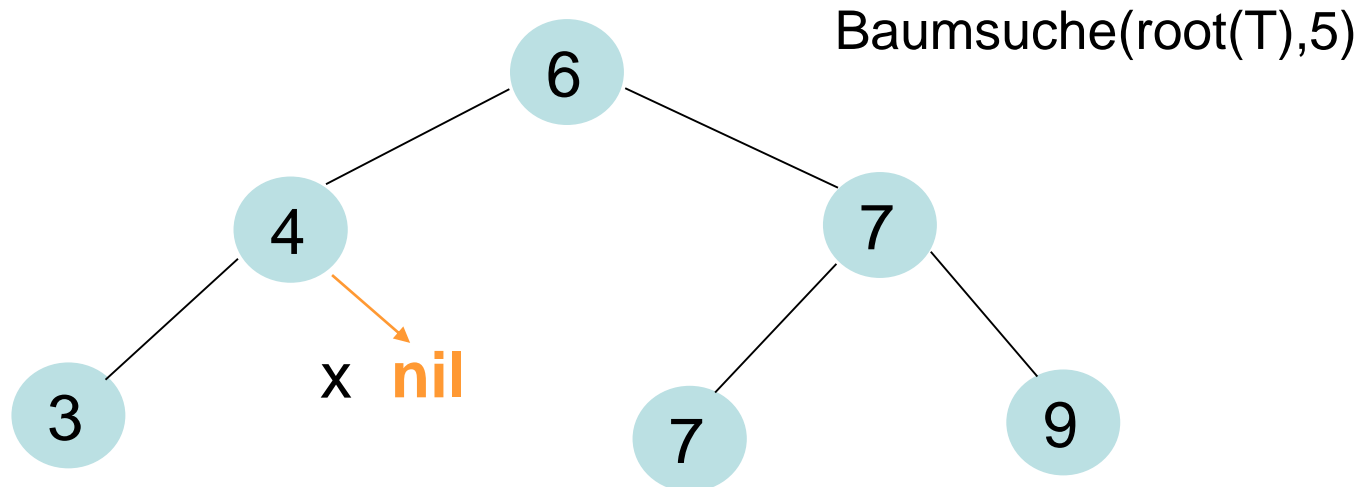
Aufruf mit
 $x = \text{root}(T)$



Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

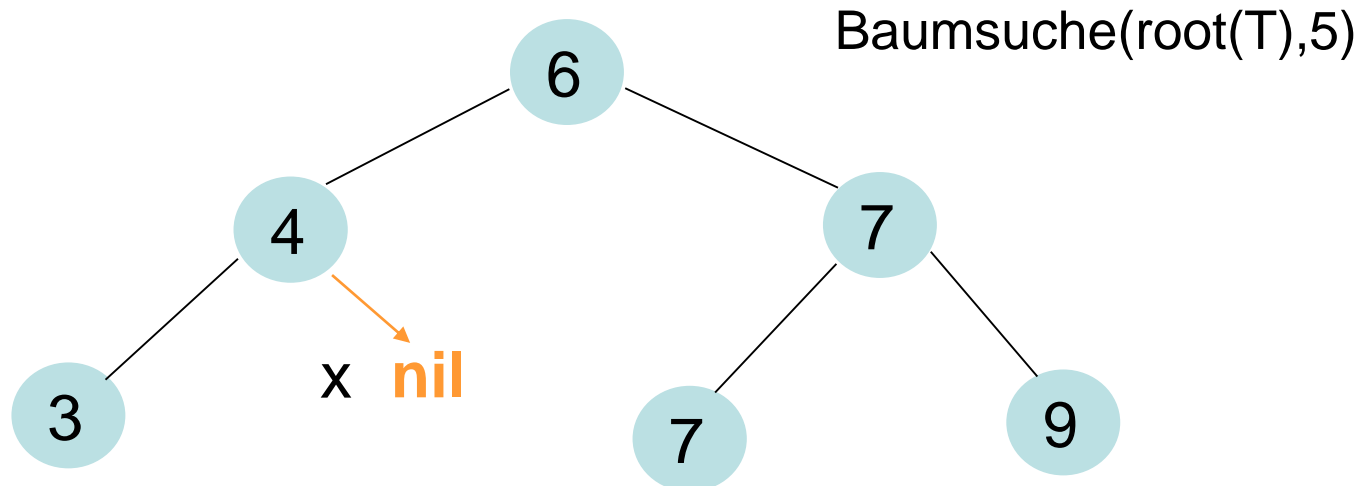
1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x



Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

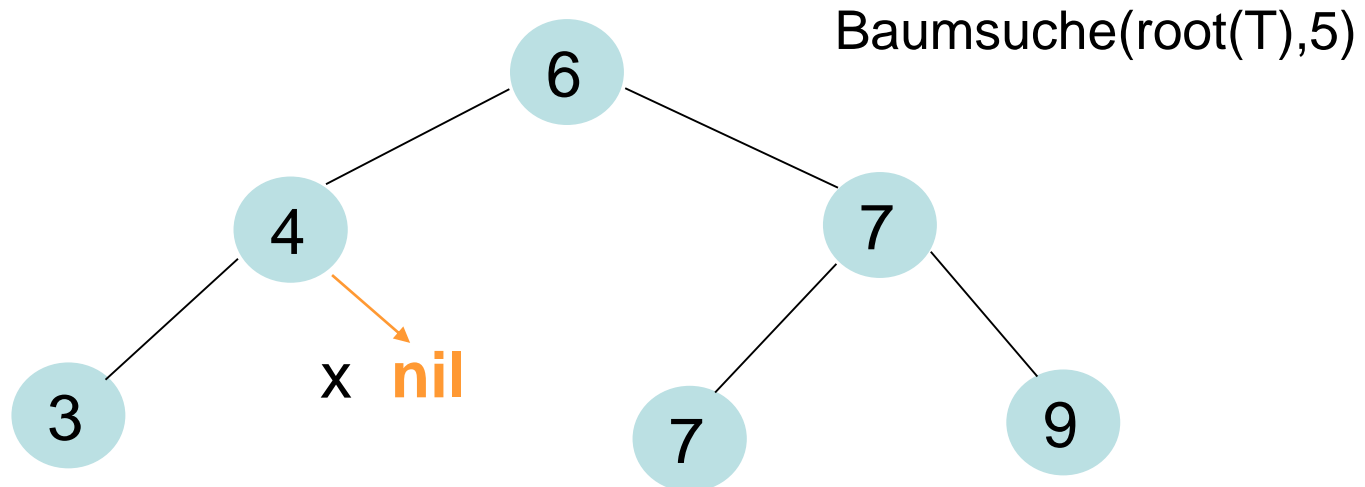
1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x



Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x

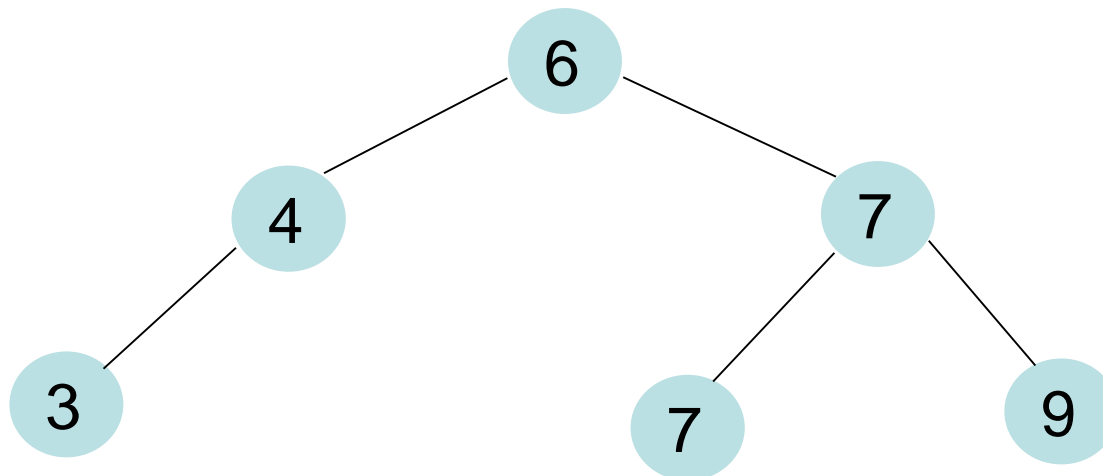


Binäre Suchbäume Suchen

IterativeBaumsuche(x,k)

1. **while** $x \neq \text{nil}$ and $k \neq \text{key}(x)$ **do**
2. **if** $k < \text{key}(x)$ **then** $x \leftarrow \text{lc}(x)$
3. **else** $x \leftarrow \text{rc}(x)$
4. **return** x

Funktionsweise wie (rekursive)
Baumsuche. Laufzeit ebenfalls $O(h)$.

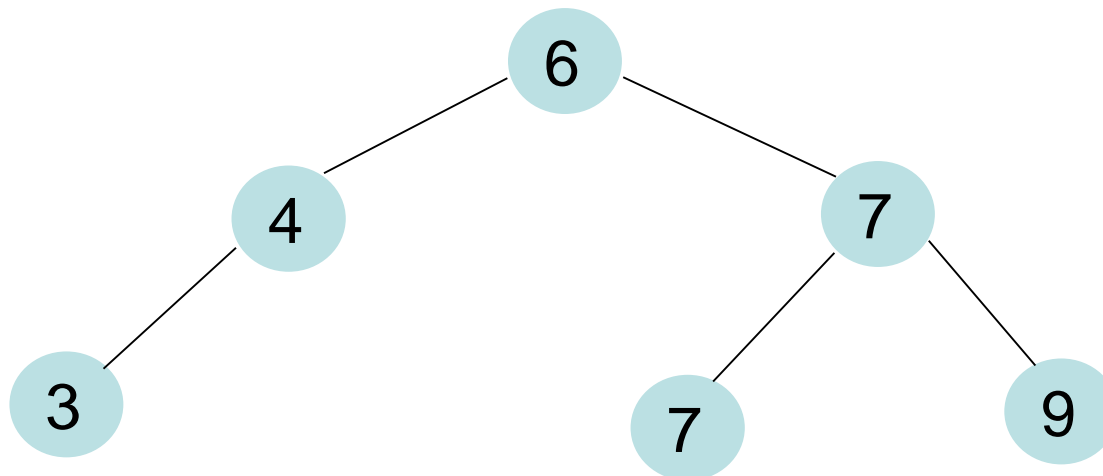


Binäre Suchbäume

Min/Max Suche

Minimum- und Maximumsuche

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



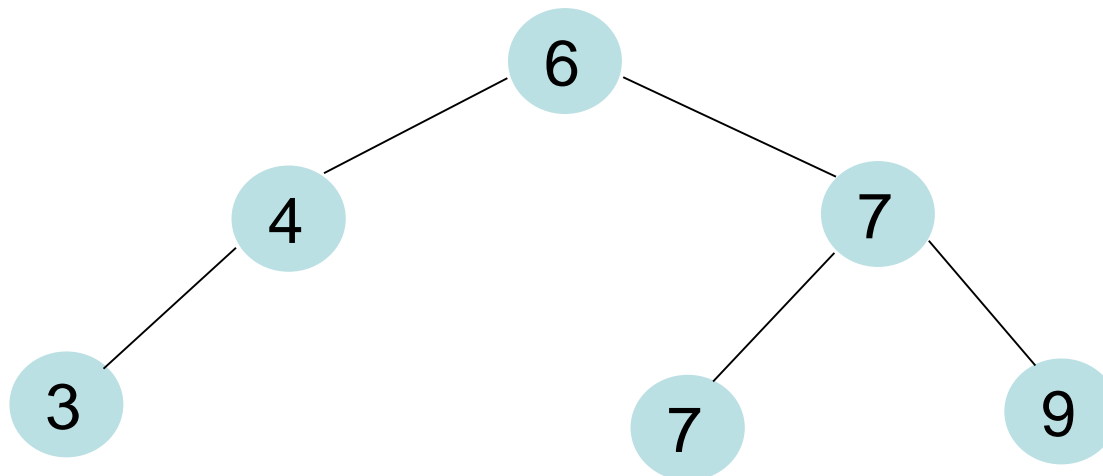
Binäre Suchbäume

Min/Max Suche

Wird mit Wurzel
aufgerufen

MinimumSuche(x)

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



Binäre Suchbäume

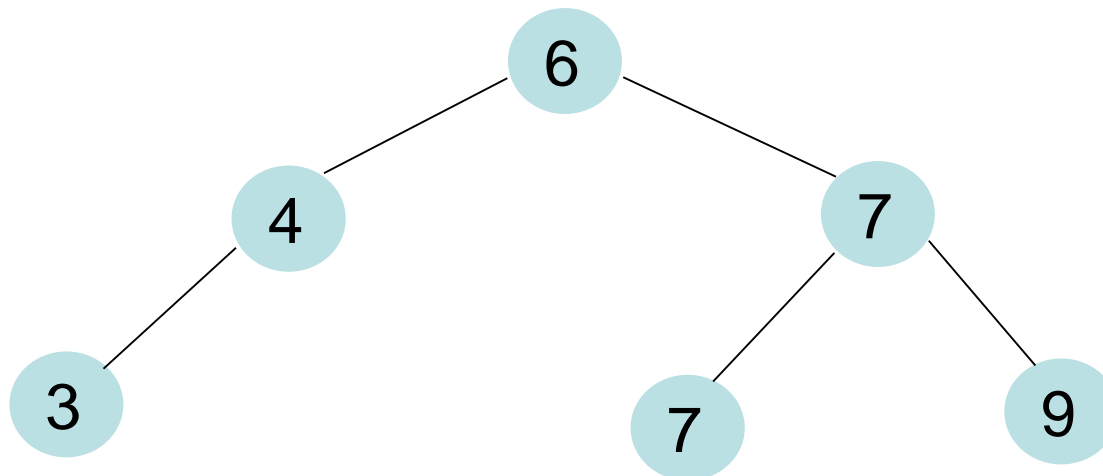
Min/Max Suche

Wird mit Wurzel
aufgerufen

Laufzeit $O(h)$

MinimumSuche(x)

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



Binäre Suchbäume

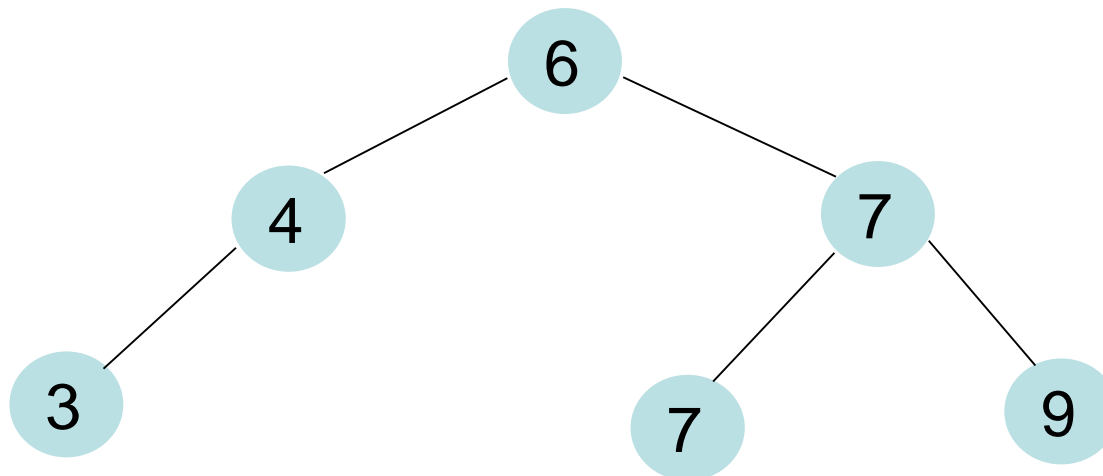
Min/Max Suche

Wird mit Wurzel
aufgerufen

Laufzeit $O(h)$

MaximumSuche(x)

1. **while** $rc(x) \neq \text{nil}$ **do** $x \leftarrow rc(x)$
2. **return** x

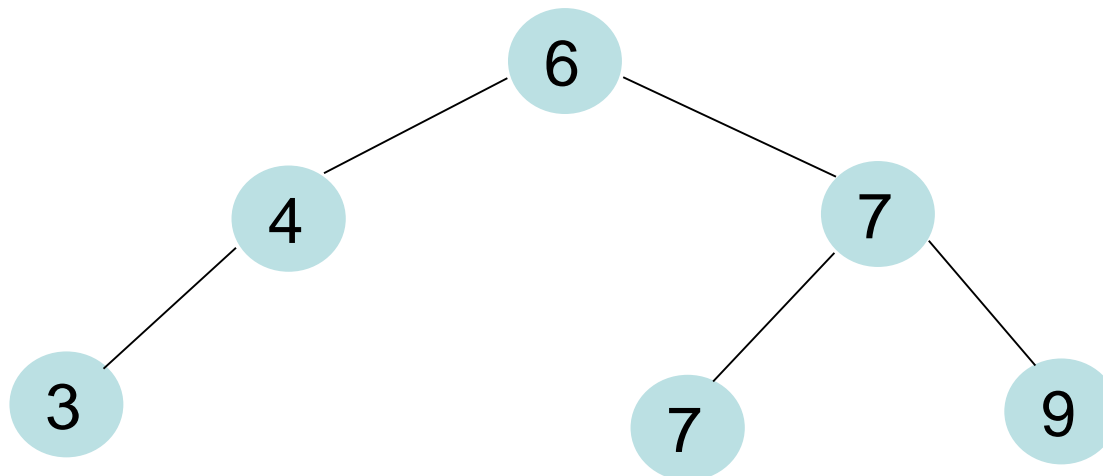


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche

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

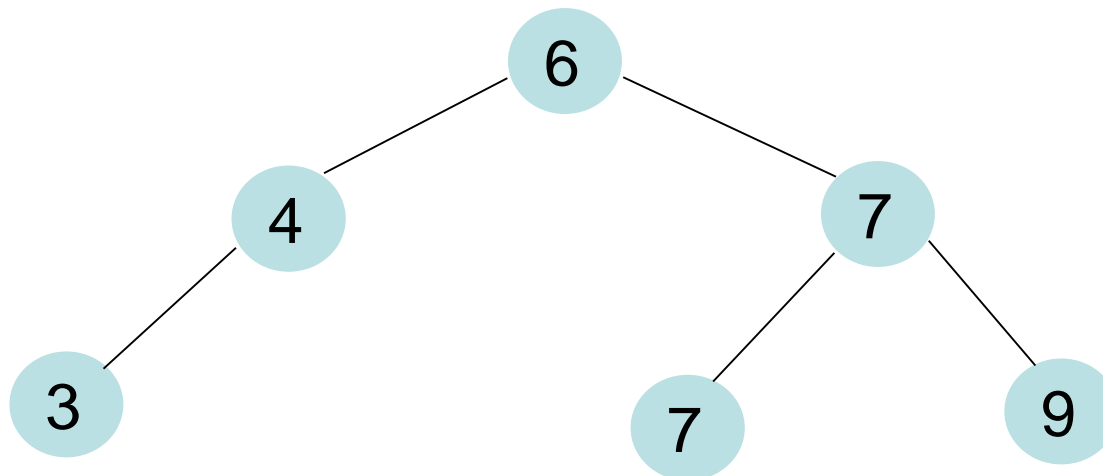


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche

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

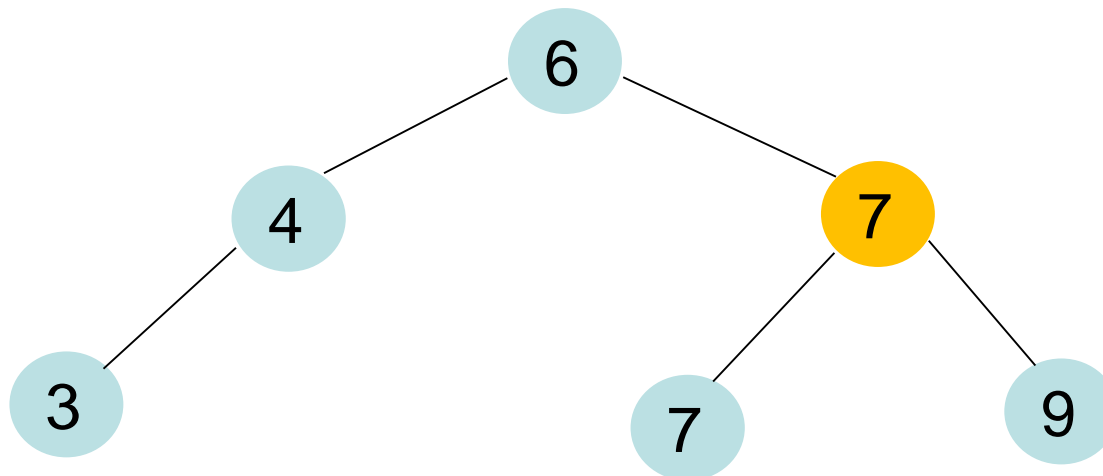


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche

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

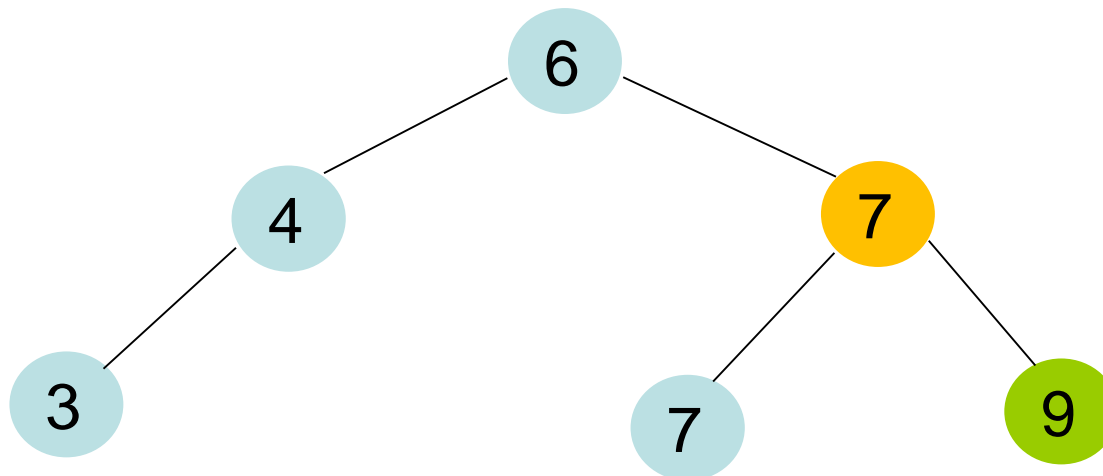


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche

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

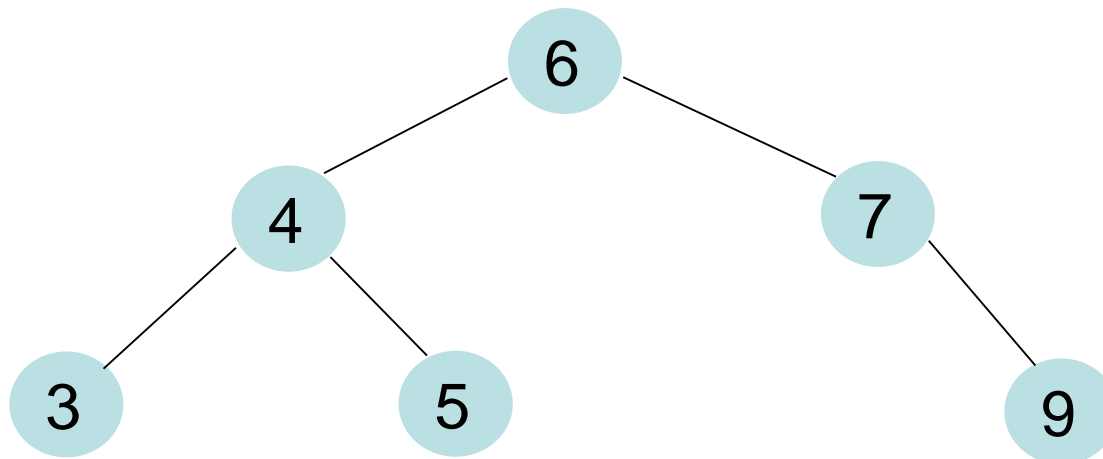


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche

- Fall 2 (rechter Unterbaum von x leer und x hat Nachfolger y):
Dann ist y der erste Knoten auf dem Pfad zur Wurzel, der größer als x ist

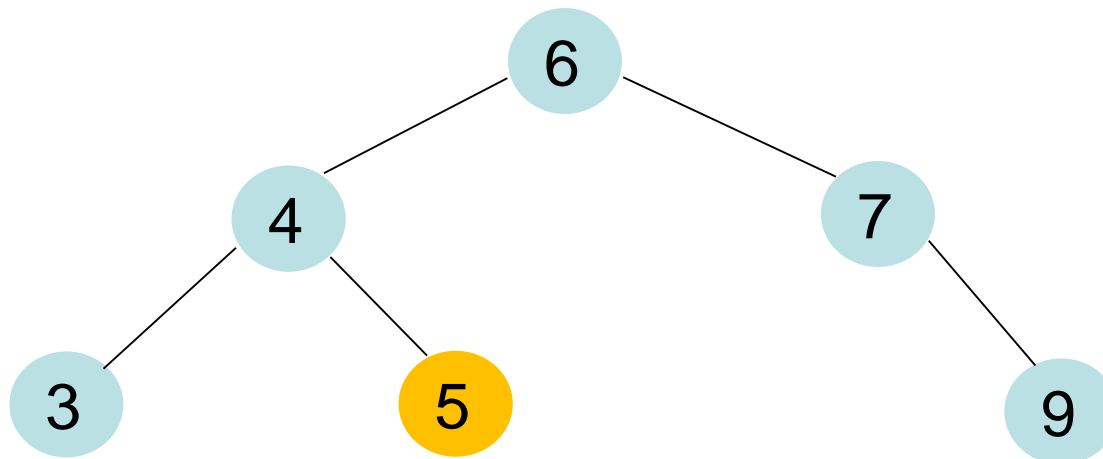


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche

- Fall 2 (rechter Unterbaum von x leer und x hat Nachfolger y):
Dann ist y der erste Knoten auf dem Pfad zur Wurzel, der größer als x ist

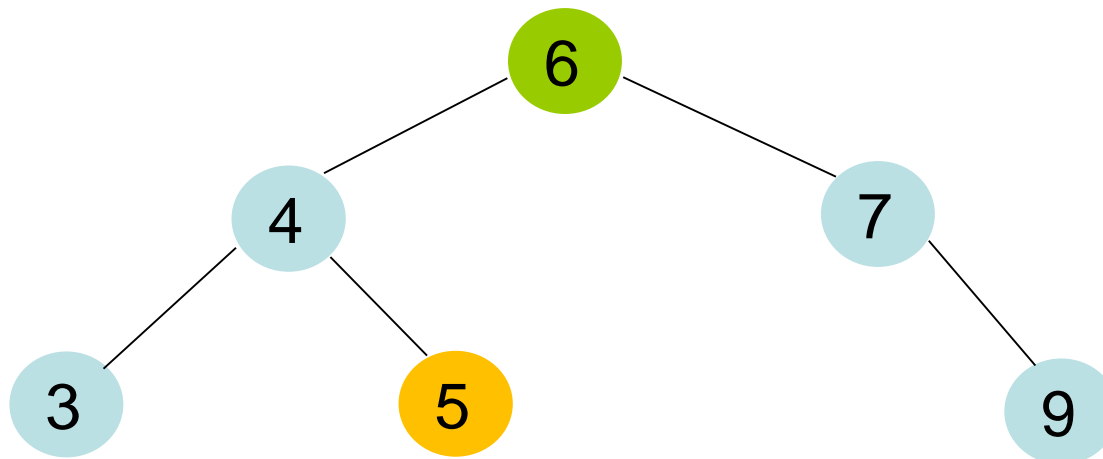


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche

- Fall 2 (rechter Unterbaum von x leer und x hat Nachfolger y):
Dann ist y der erste Knoten auf dem Pfad zur Wurzel, der größer als x ist

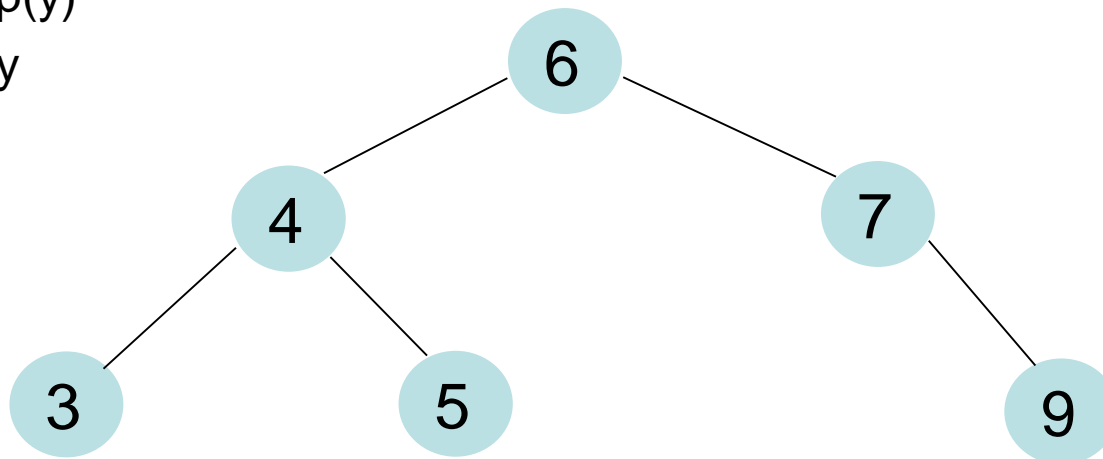


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

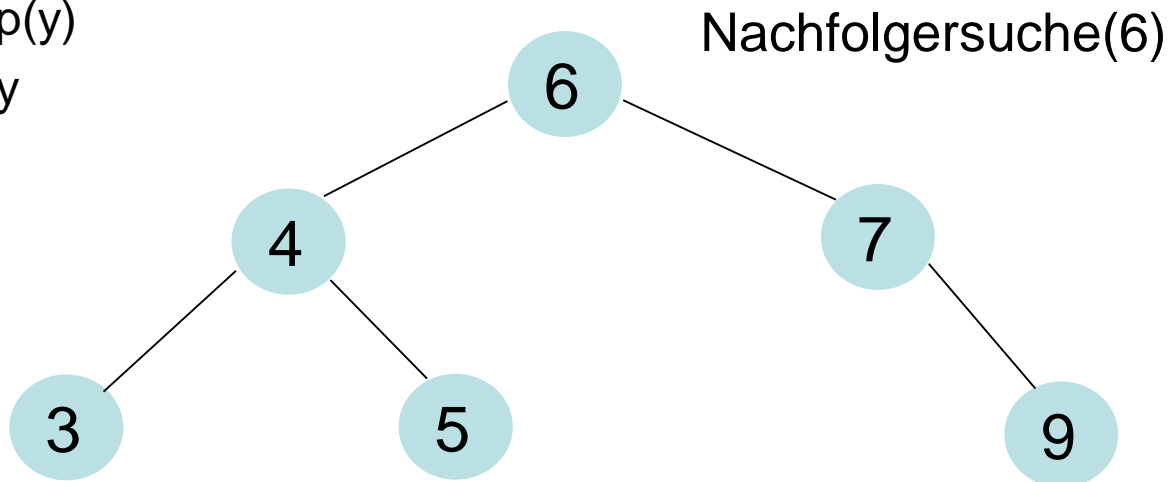


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

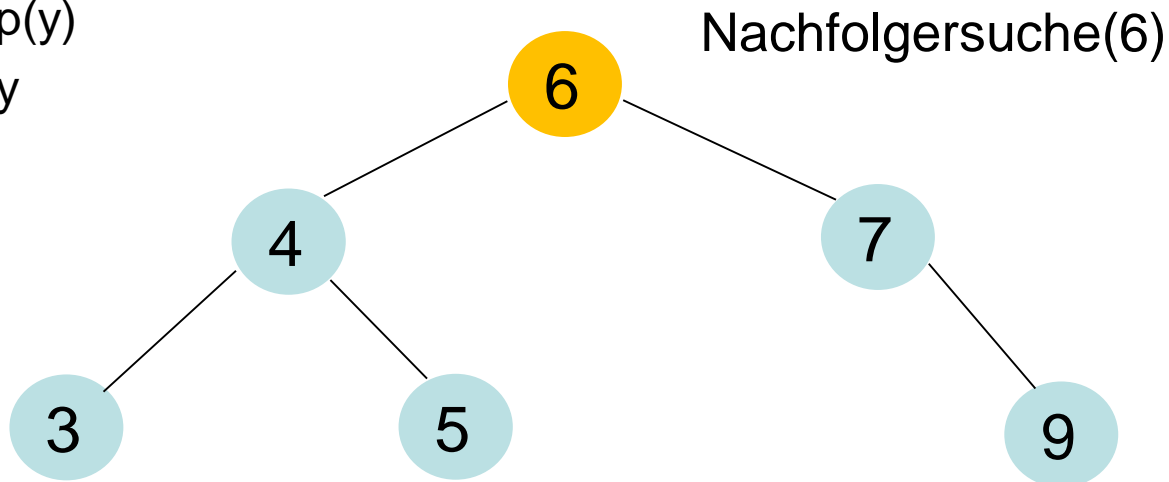


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

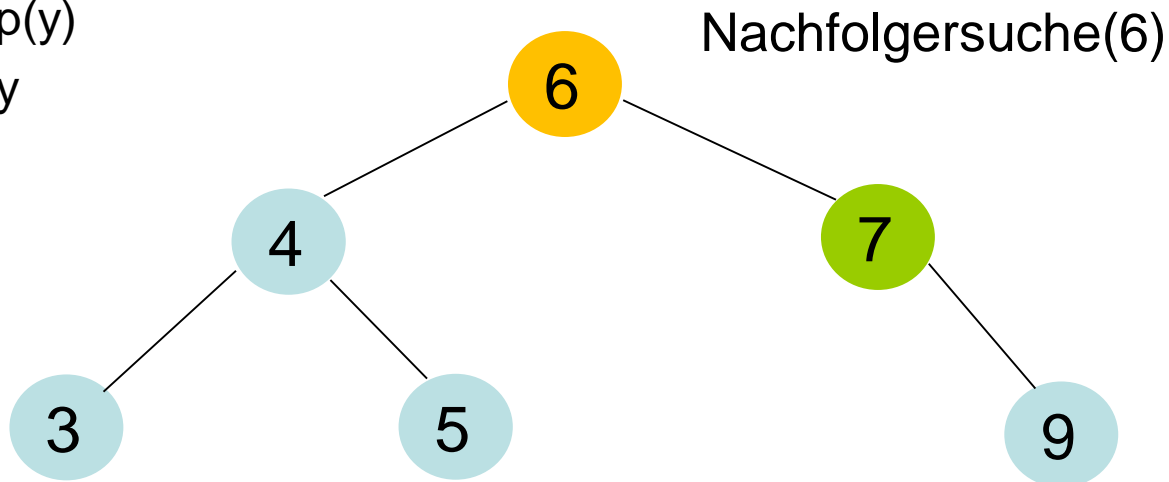


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

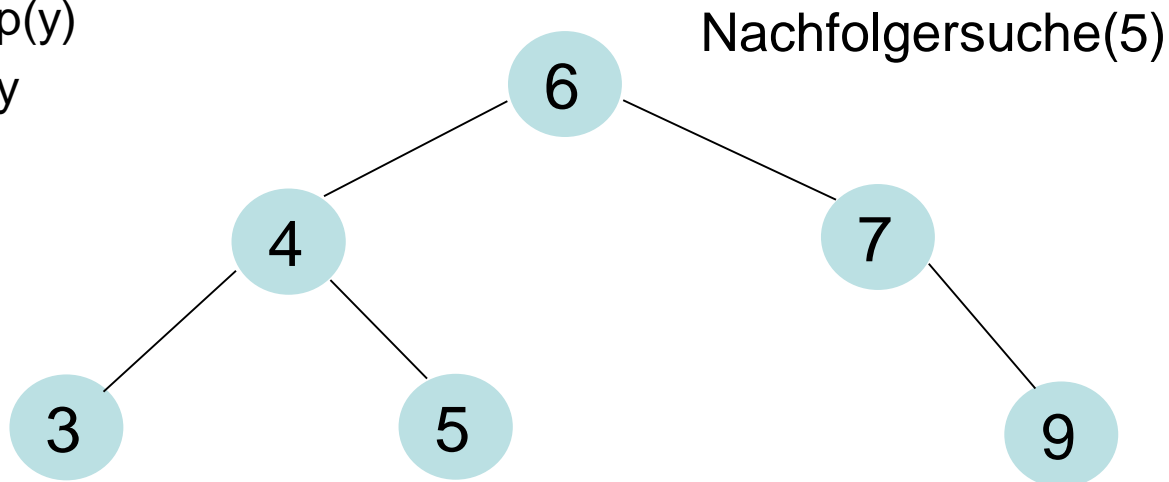


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

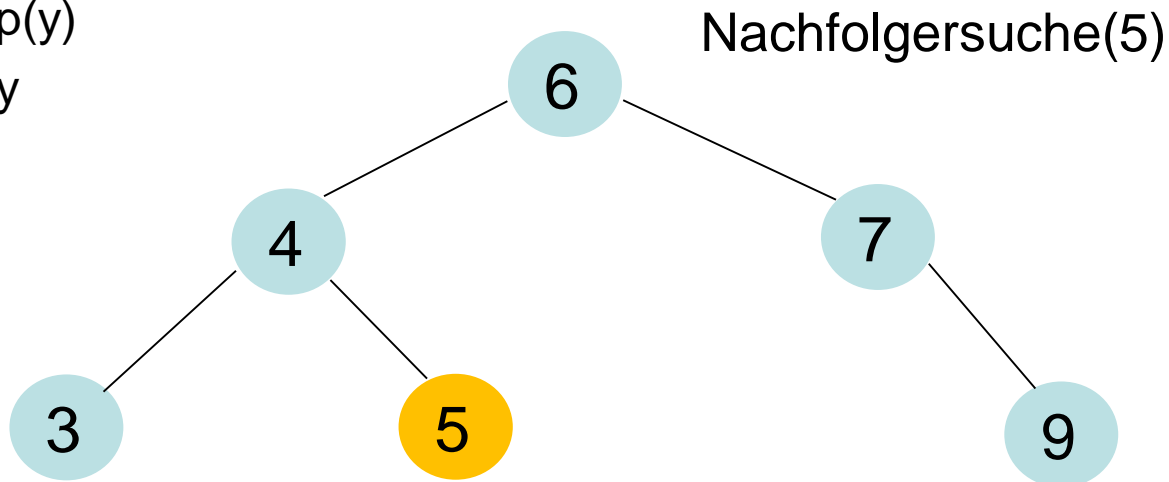


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

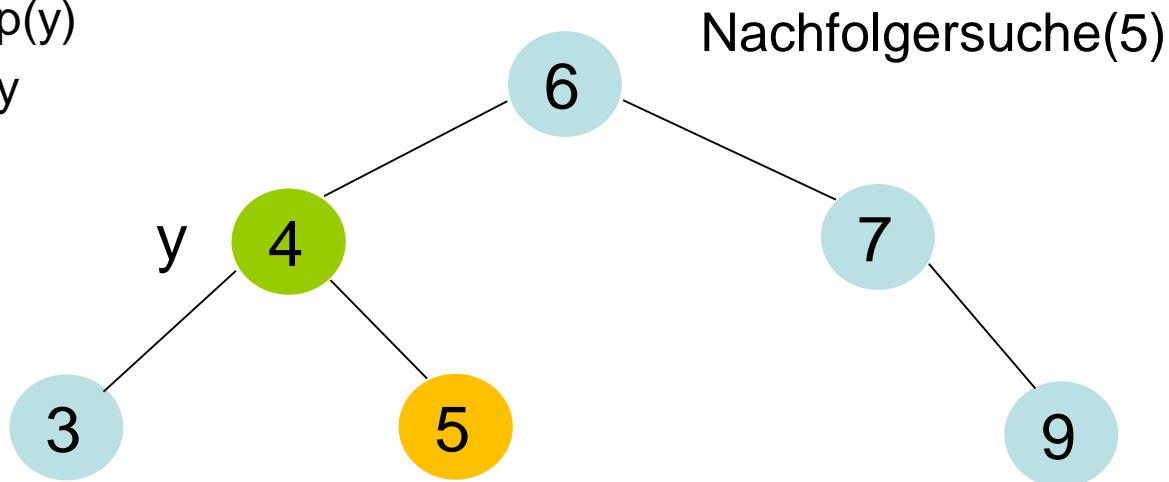


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

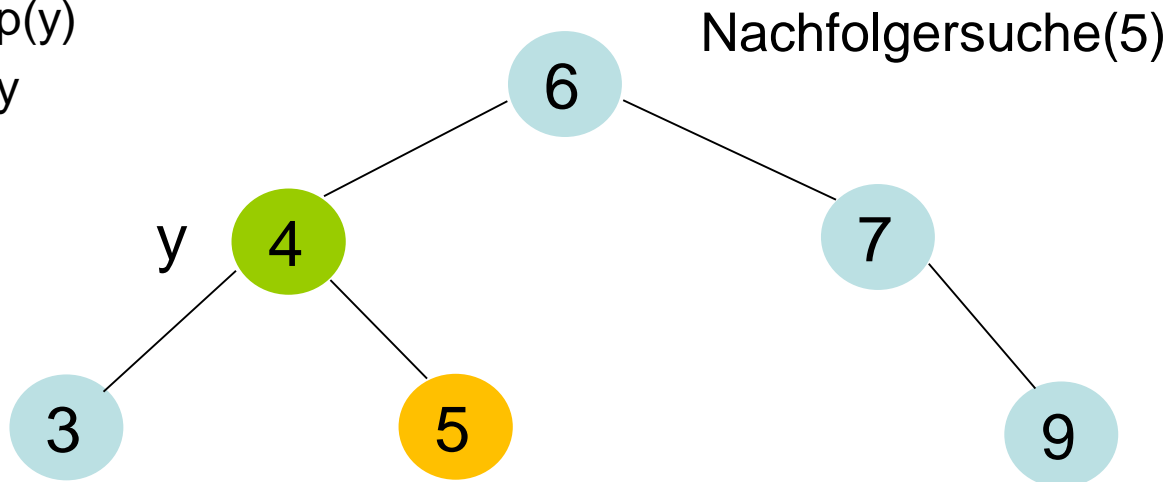


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

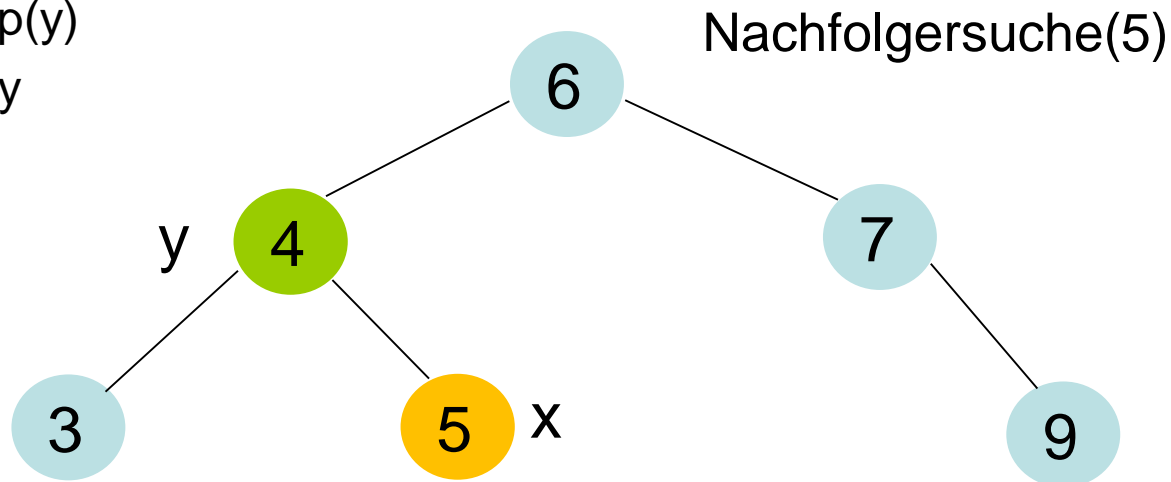


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

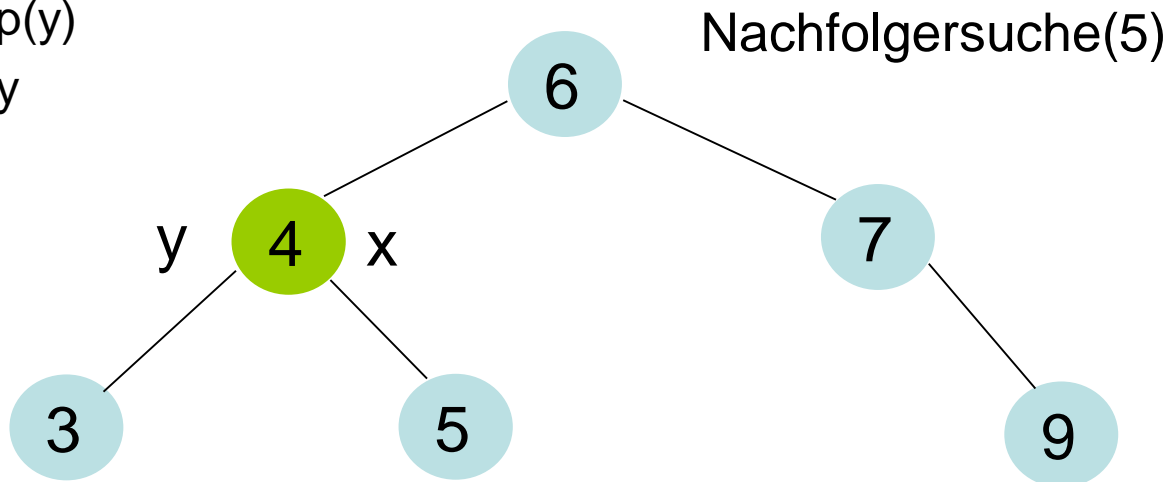


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

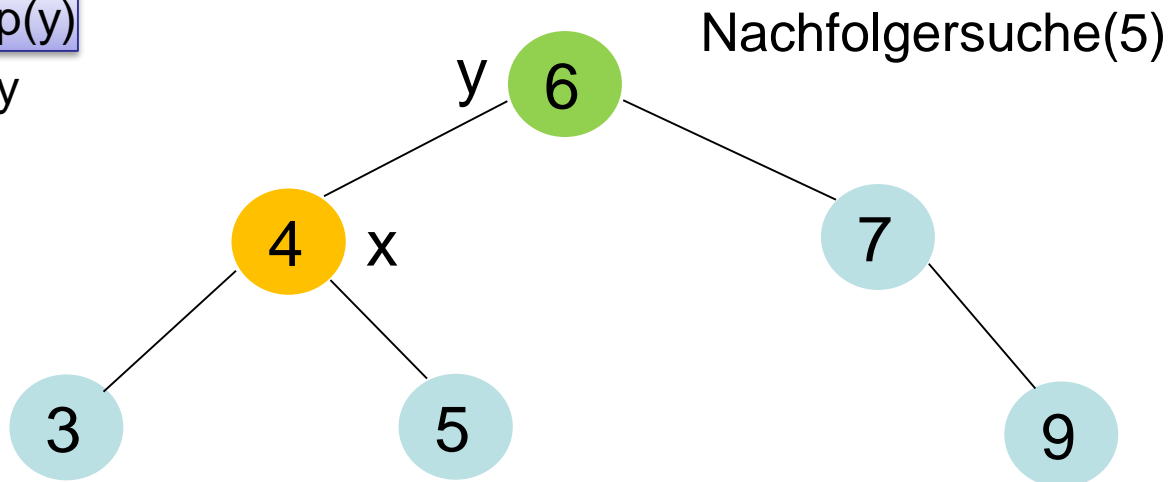


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

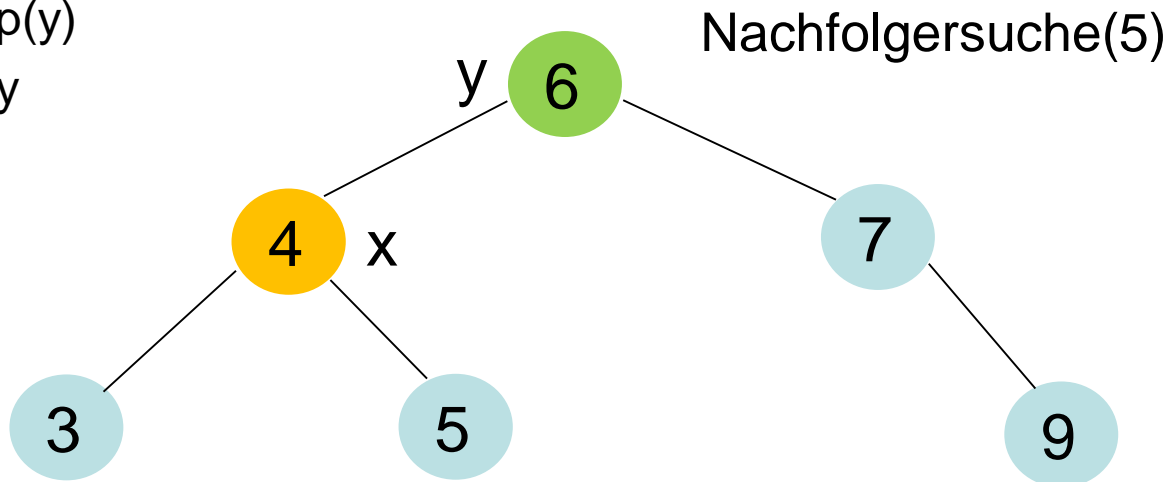


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

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

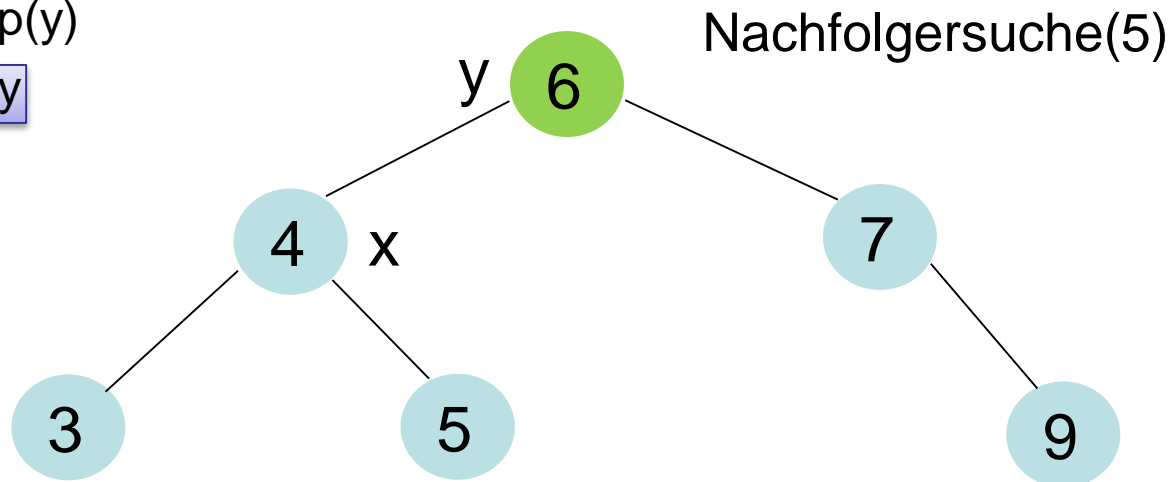


Binäre Suchbäume

Nachfolgersuche

Nachfolgersuche(x)

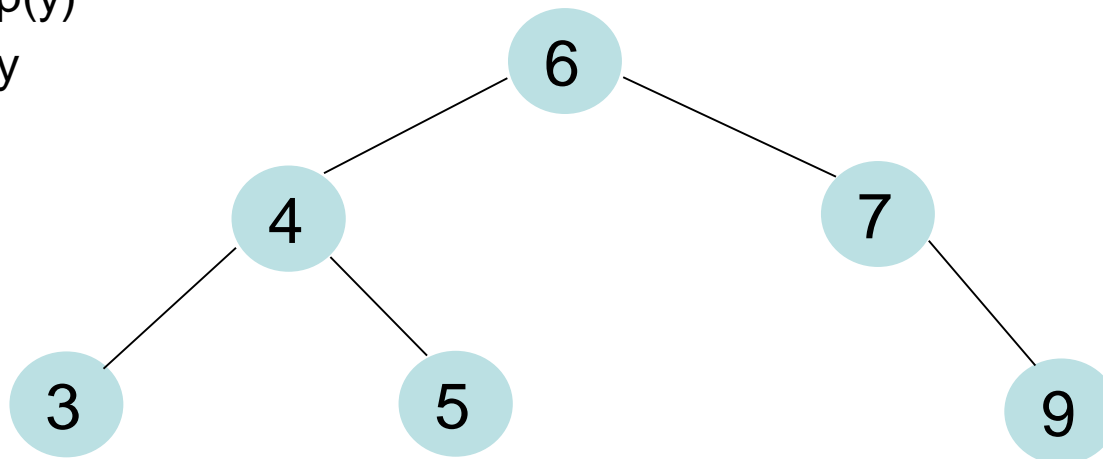
1. **if** $rc(x) \neq \text{nil}$ **then return** MinimumSuche($rc(x)$)
2. $y \leftarrow p(x)$
3. **while** $y \neq \text{nil}$ and $x = rc(y)$ **do**
4. $x \leftarrow y$
5. $y \leftarrow p(y)$
6. **return** y



Nachfolgersuche(x)

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

Laufzeit $O(h)$



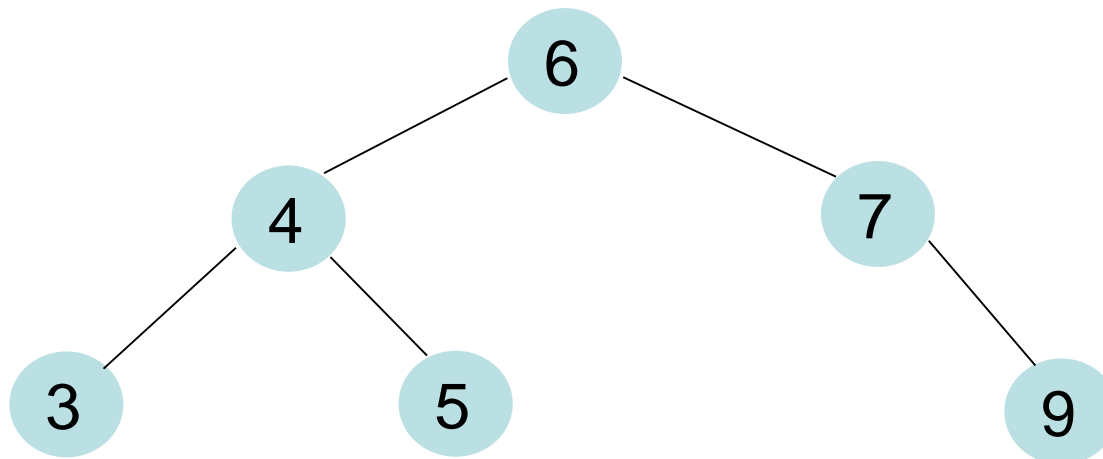
Binäre Suchbäume

Vorgängersuche

Vorgängersuche

- Analog zur Nachfolgersuche
- Daher ebenfalls $O(h)$ Laufzeit

Laufzeit $O(h)$



Ein grundlegendes Datenbank-Problem

- Speicherung von Datensätzen

Beispiel

- Kundendaten (Name, Adresse, Wohnort, Kundennummer, offene Rechnungen, offene Bestellungen,...)

Anforderungen

- Schneller Zugriff
- Einfügen neuer Datensätze
- Löschen bestehender Datensätze

Dynamische Baumoperationen

Binäre Suchbäume

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

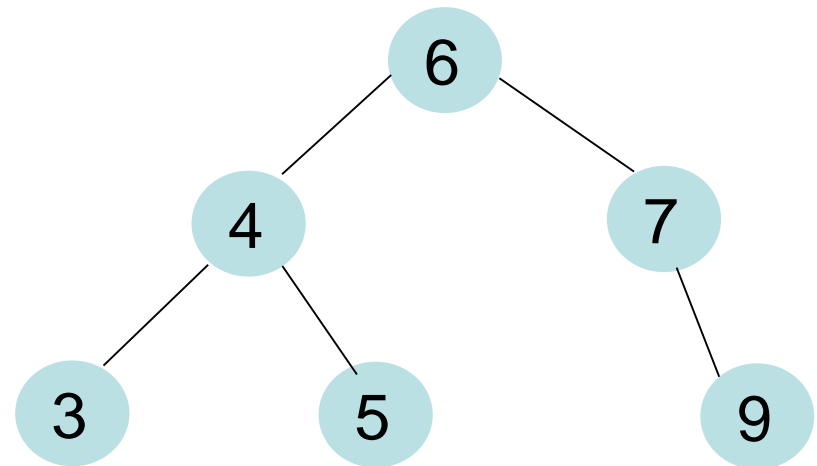
Dynamische Operationen?

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

Binäre Suchbäume Einfügen

Einfügen

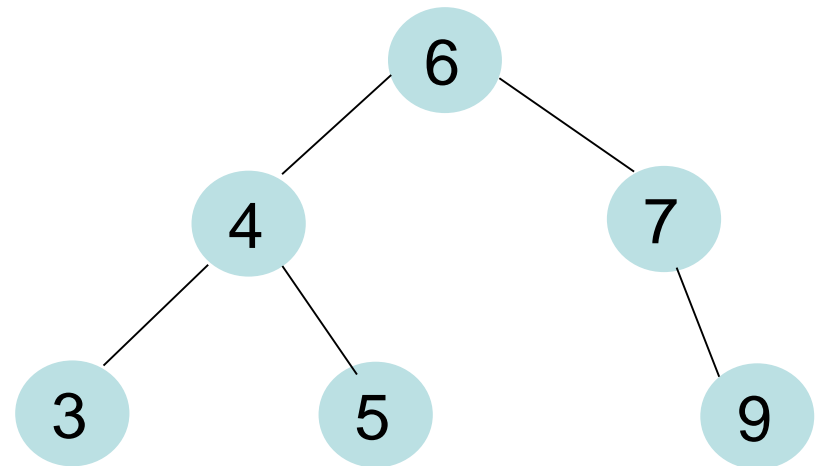
- Ähnlich wie Baumsuche: Finde Blatt, an das neuer Knoten angehängt wird
- Danach wird **nil**-Zeiger durch neues Element ersetzt



Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$



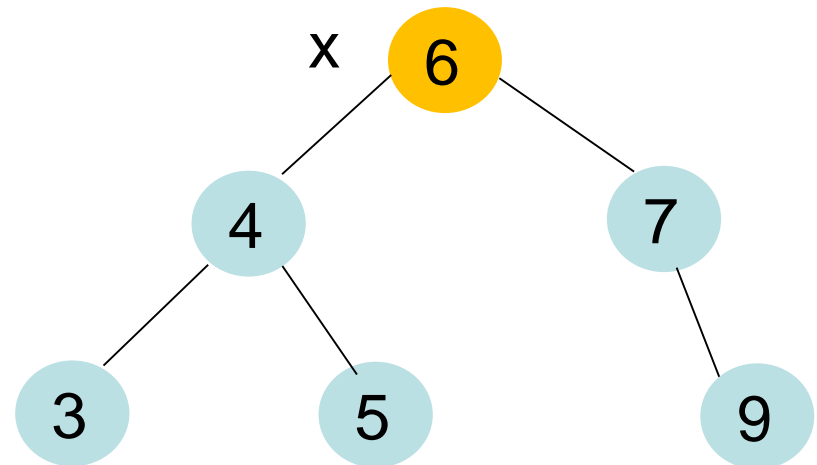
Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

y wird Vater des
einzufügenden
Elements

Einfügen(8)



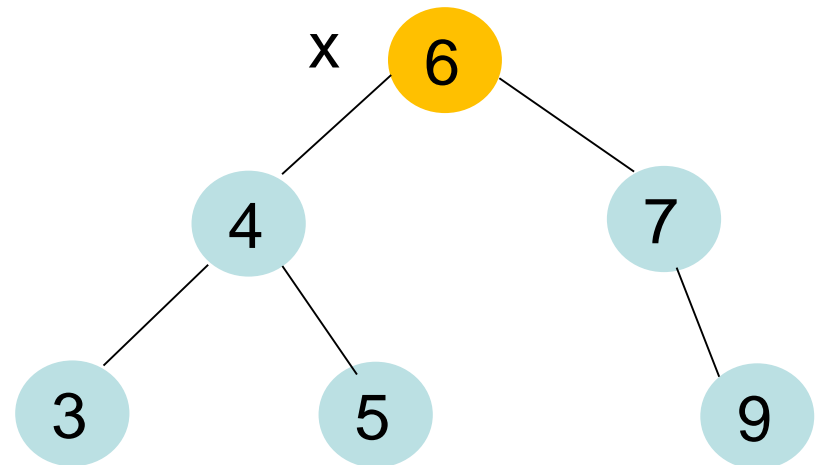
Binäre Suchbäume

Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

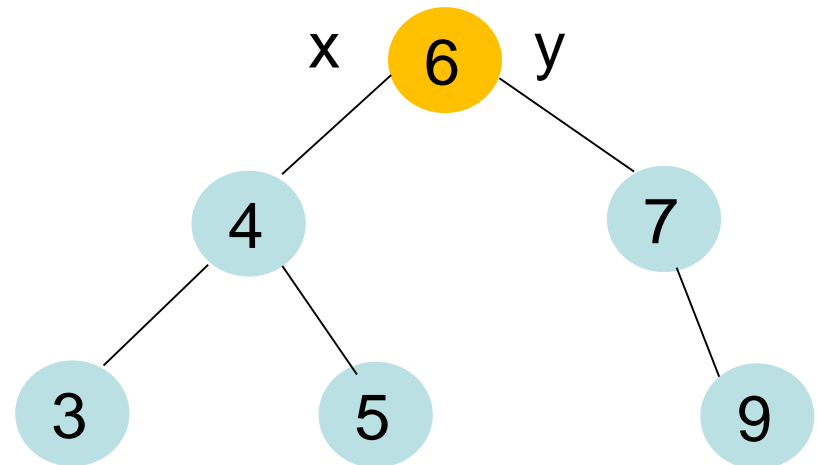


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

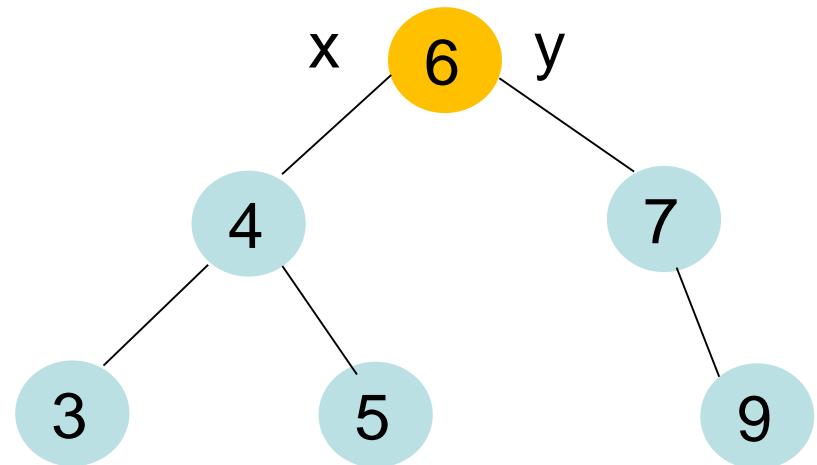


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

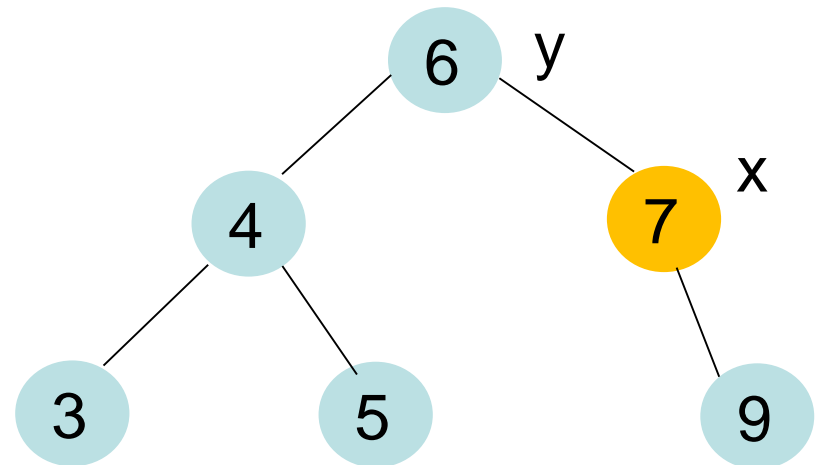


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

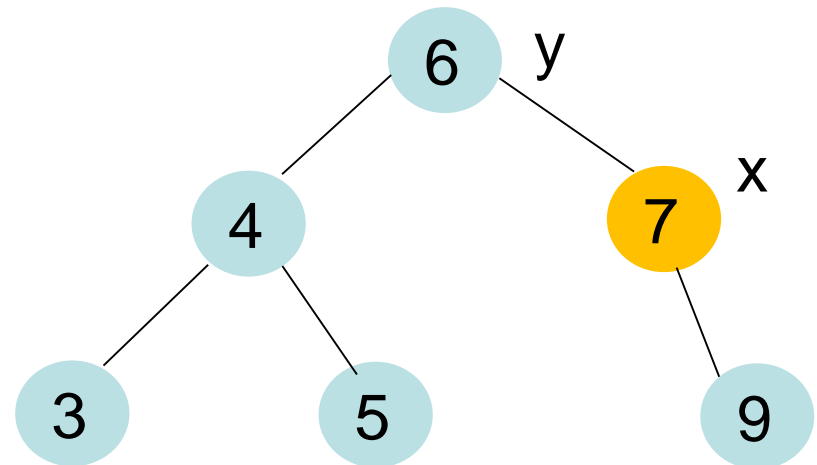


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

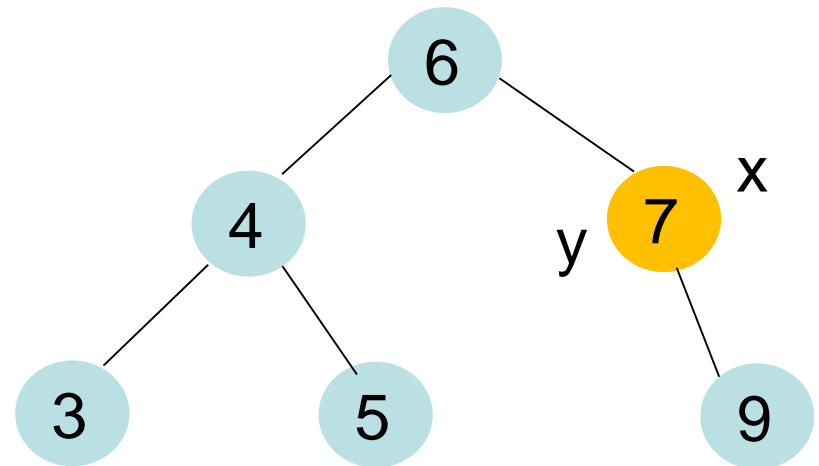


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

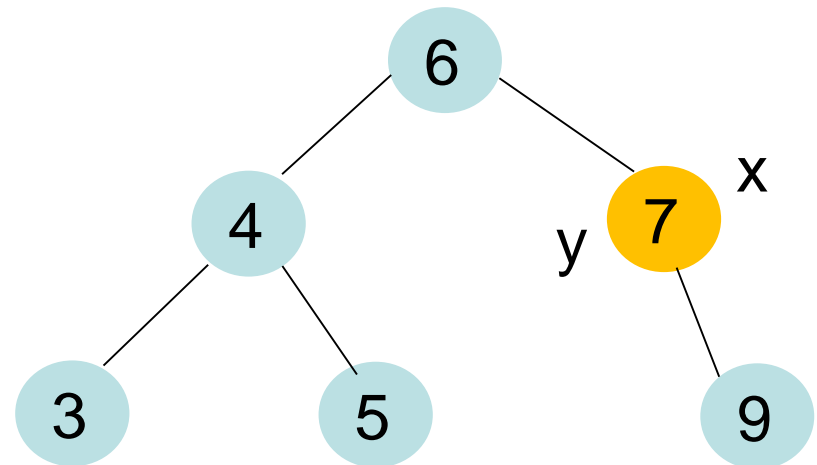


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)



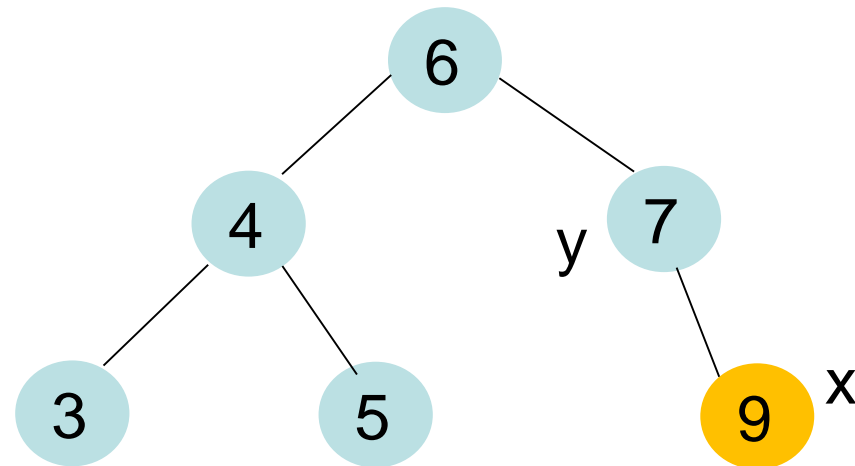
Binäre Suchbäume

Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

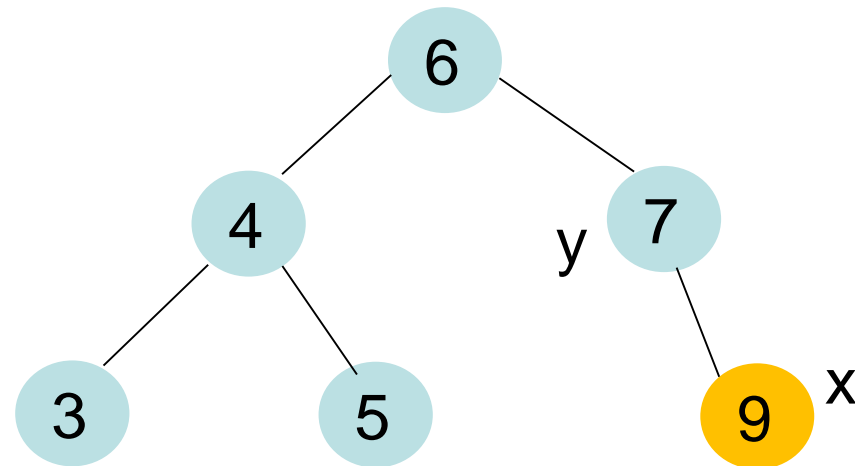


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

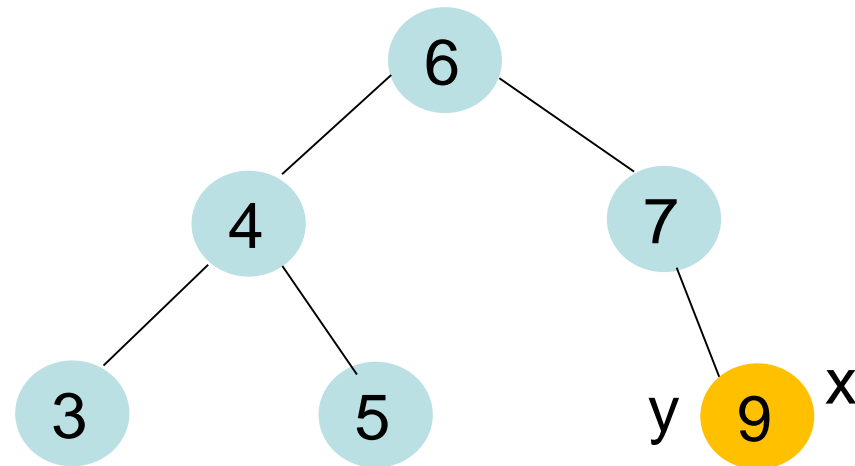


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

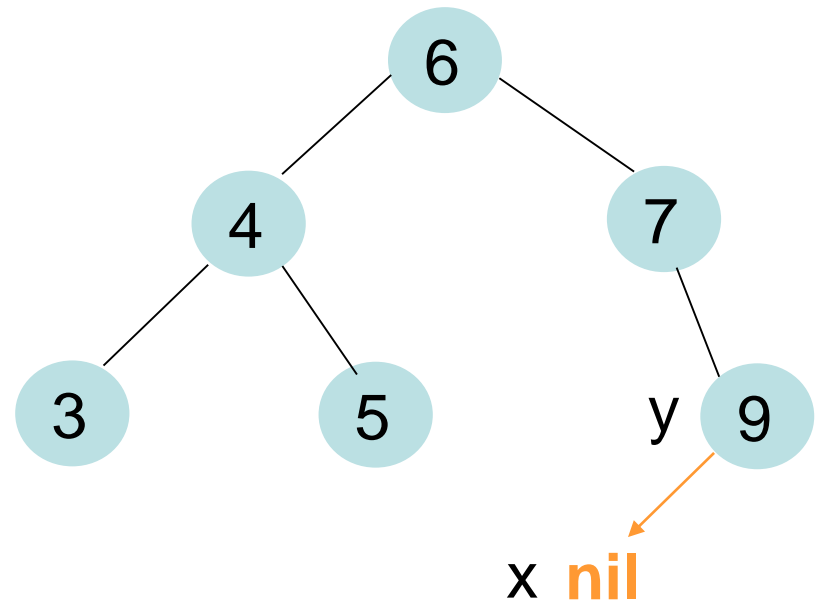


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

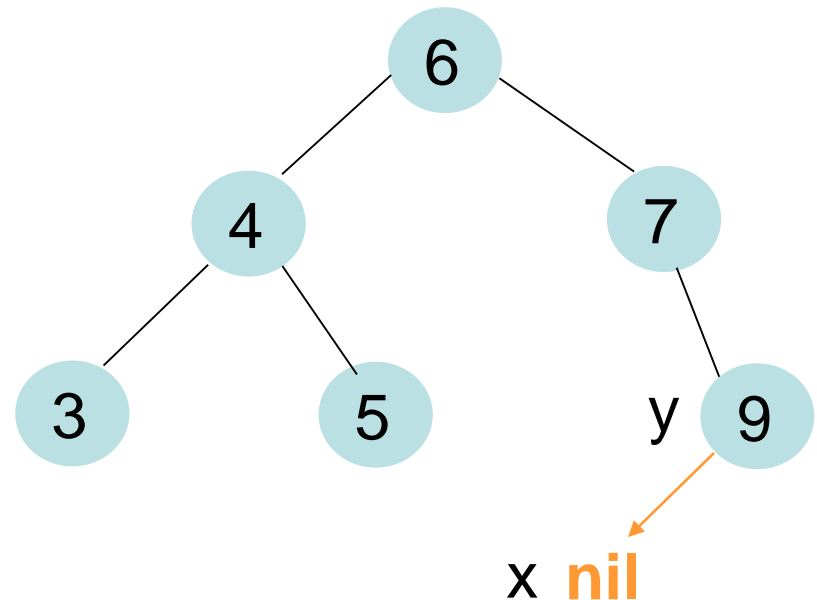


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)



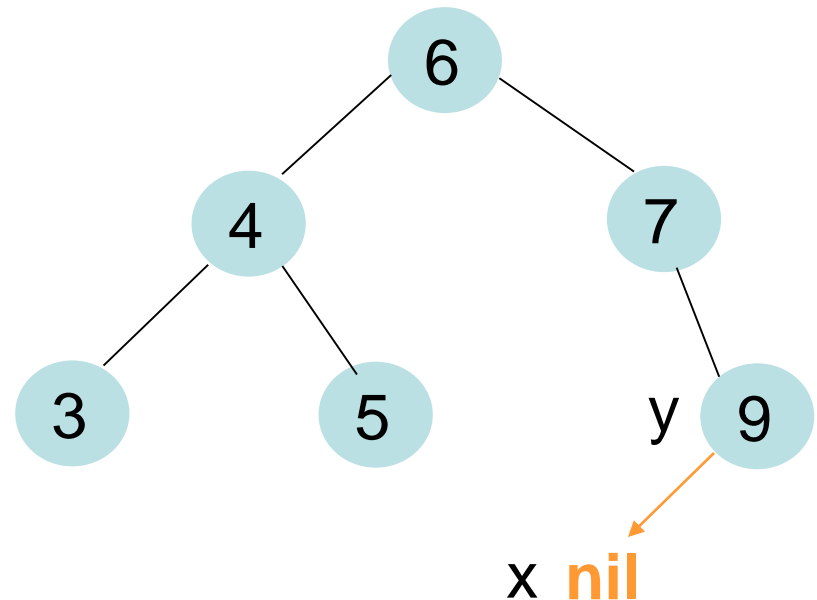
Binäre Suchbäume

Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

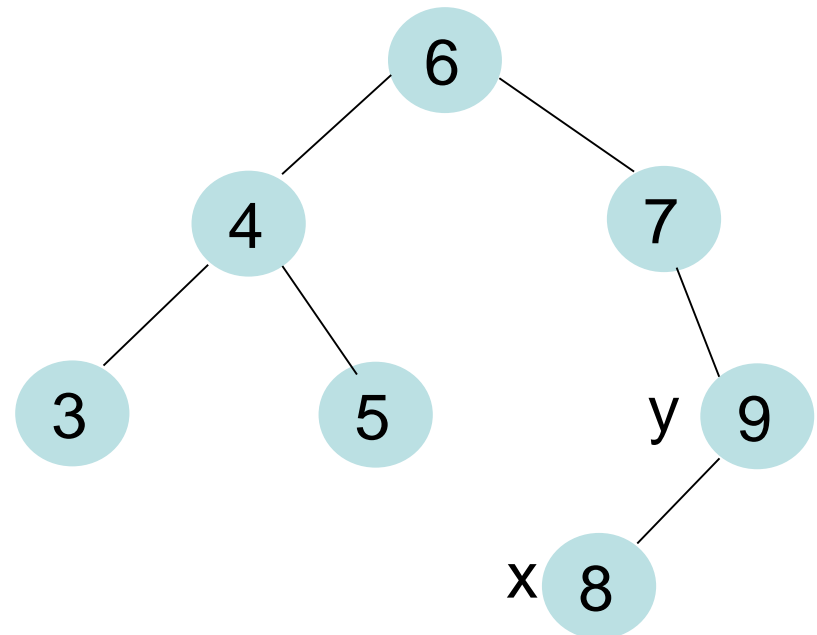


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)



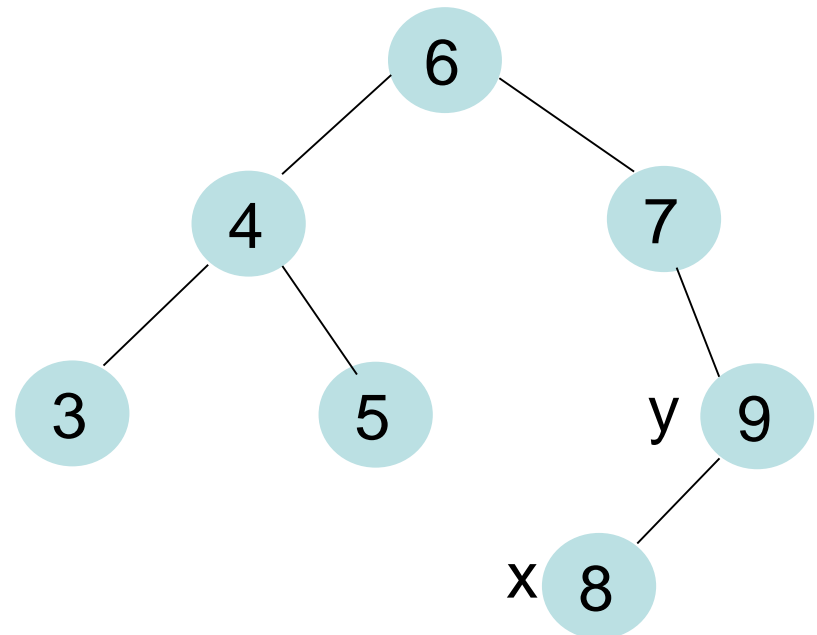
Binäre Suchbäume

Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

Einfügen(8)

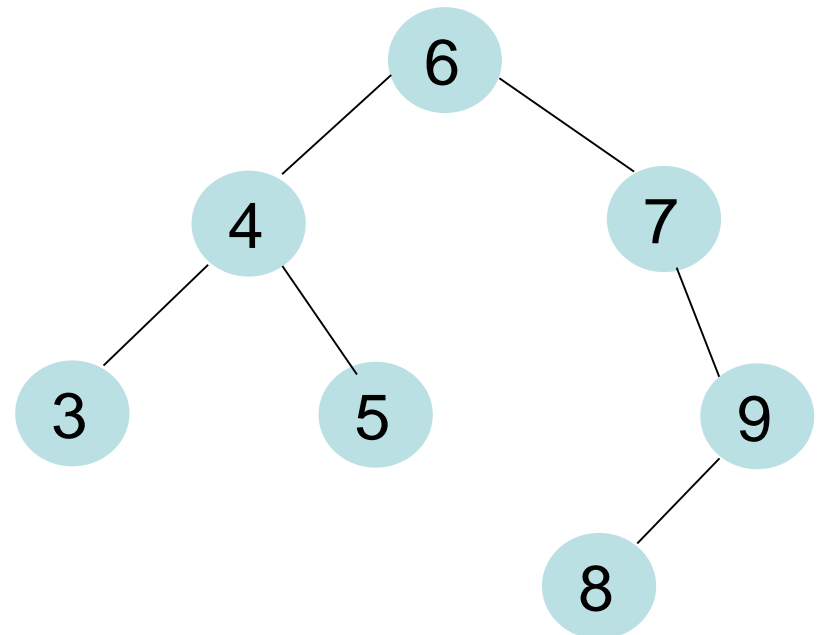


Binäre Suchbäume Einfügen

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** $\text{key}(z) \leq \text{key}(x)$ **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** $\text{key}(z) \leq \text{key}(y)$ **then** $\text{lc}(y) \leftarrow z$
10. **else** $\text{rc}(y) \leftarrow z$

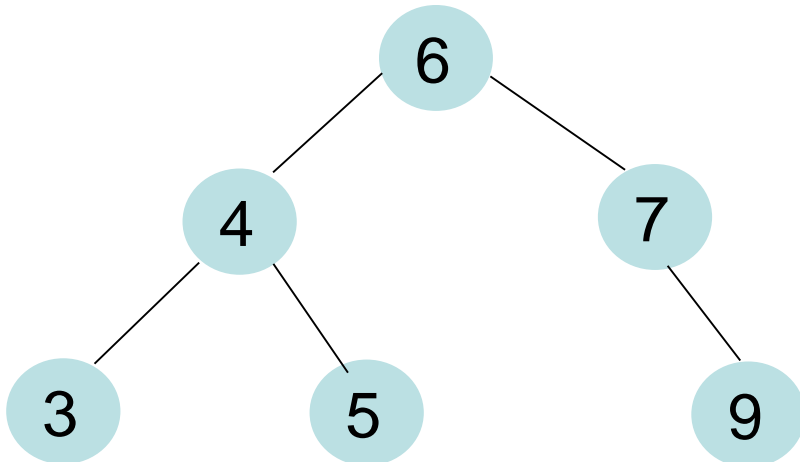
Laufzeit $O(h)$



Binäre Suchbäume Löschen

Löschen

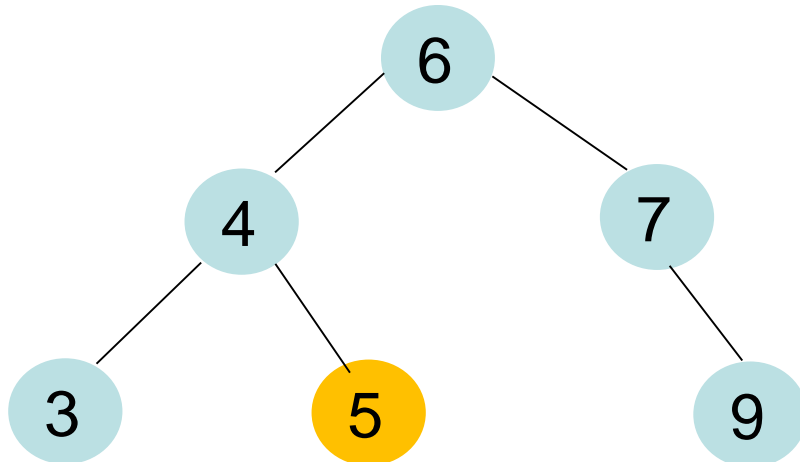
- 3 unterschiedliche Fälle
- (a) zu löschendes Element z hat keine Kinder
- (b) zu löschendes Element z hat ein Kind
- (c) zu löschendes Element z hat zwei Kinder



Binäre Suchbäume Löschen

Fall (a)

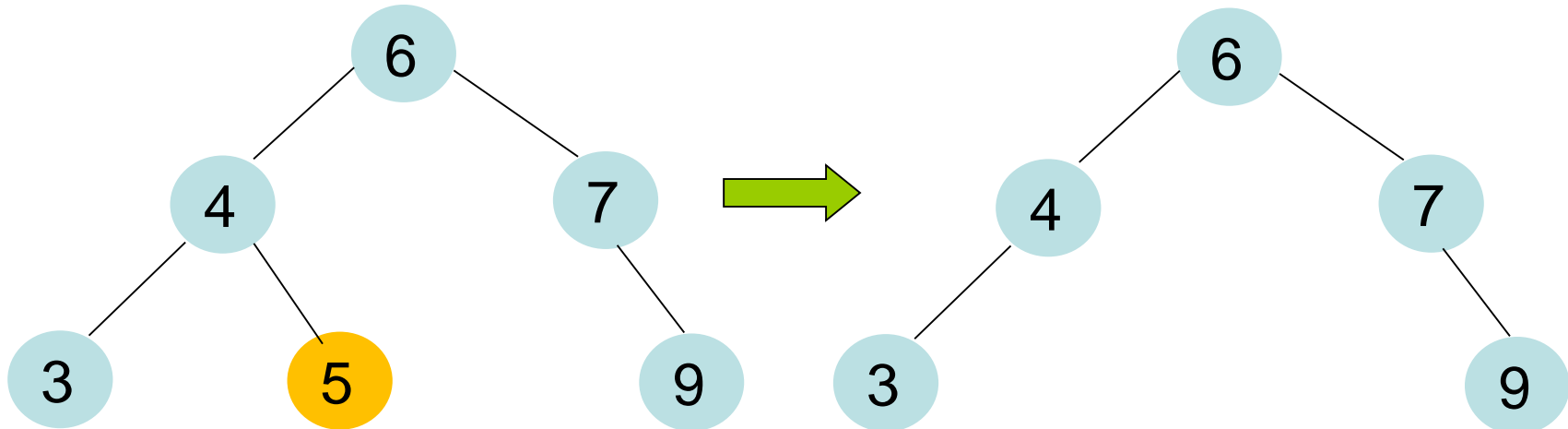
- zu löschendes Element z hat keine Kinder



Binäre Suchbäume Löschen

Fall (a)

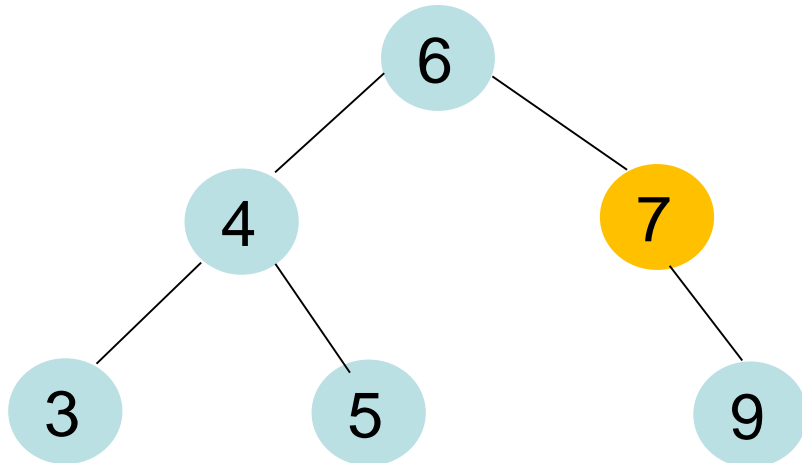
- zu löschendes Element z hat keine Kinder
- Entferne Element



Binäre Suchbäume Löschen

Fall (b)

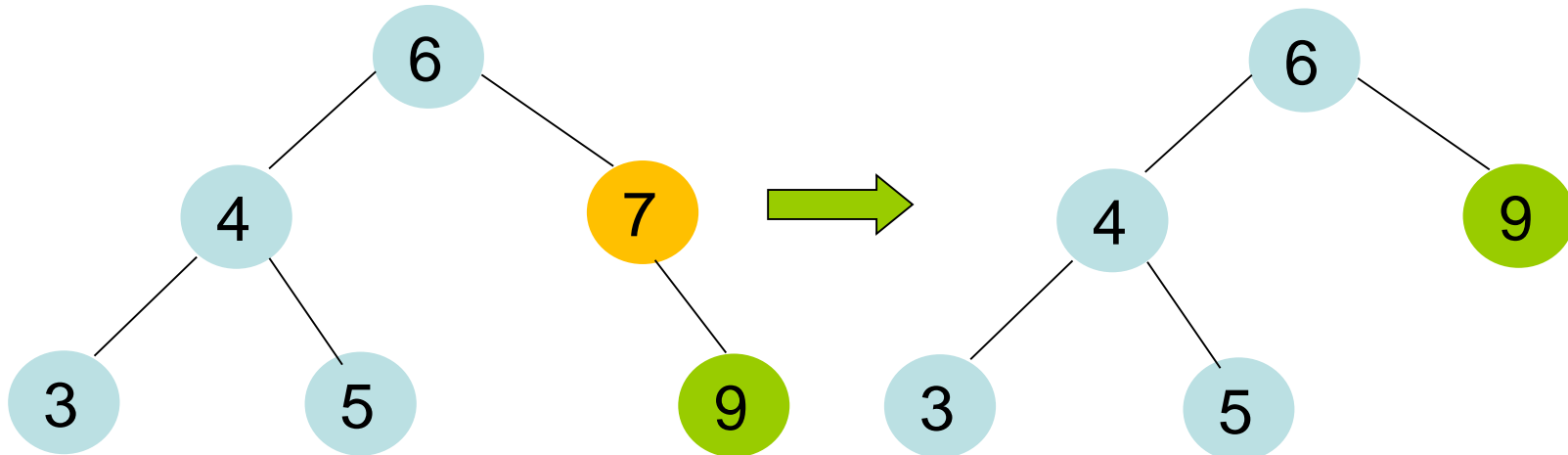
- Zu löschendes Element z hat 1 Kind



Binäre Suchbäume Löschen

Fall (b)

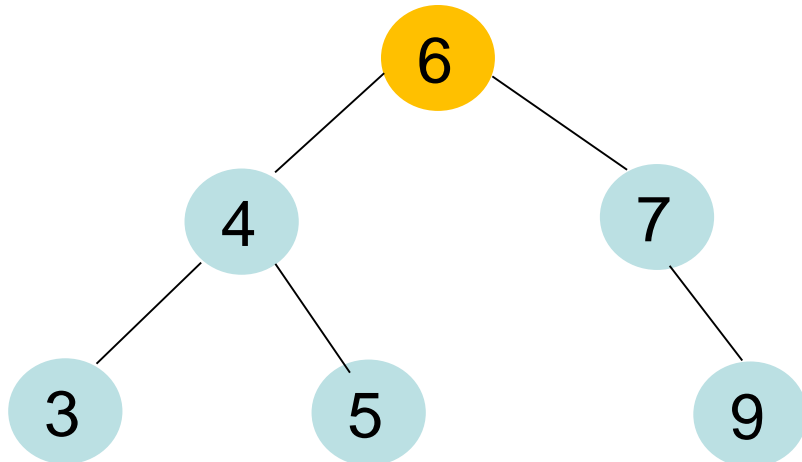
- Zu löschendes Element z hat 1 Kind



Binäre Suchbäume Löschen

Fall (c)

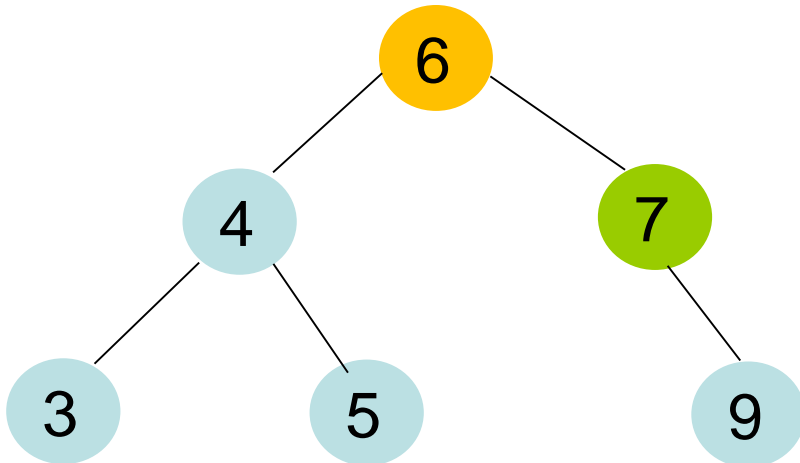
- Zu löschendes Element z hat 2 Kinder



Binäre Suchbäume Löschen

Fall (c)

- Zu löschendes Element z hat 2 Kinder
- Schritt 1: Bestimme Nachfolger von z

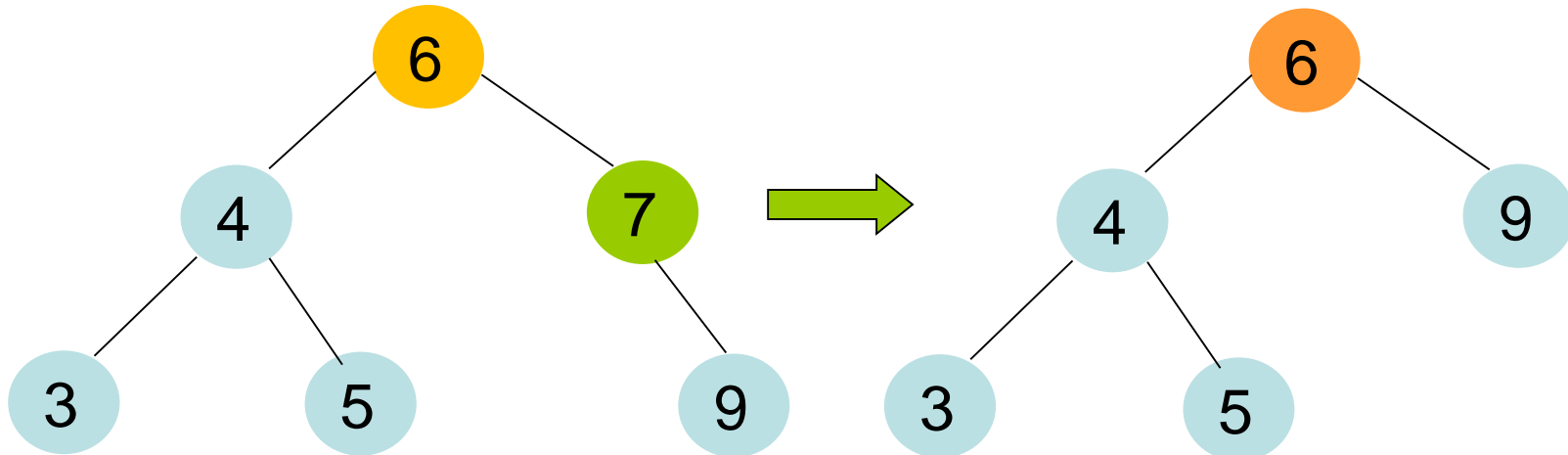


Binäre Suchbäume Löschen

Fall (c)

- Zu löschendes Element z hat 2 Kinder
- Schritt 1: Bestimme Nachfolger von z

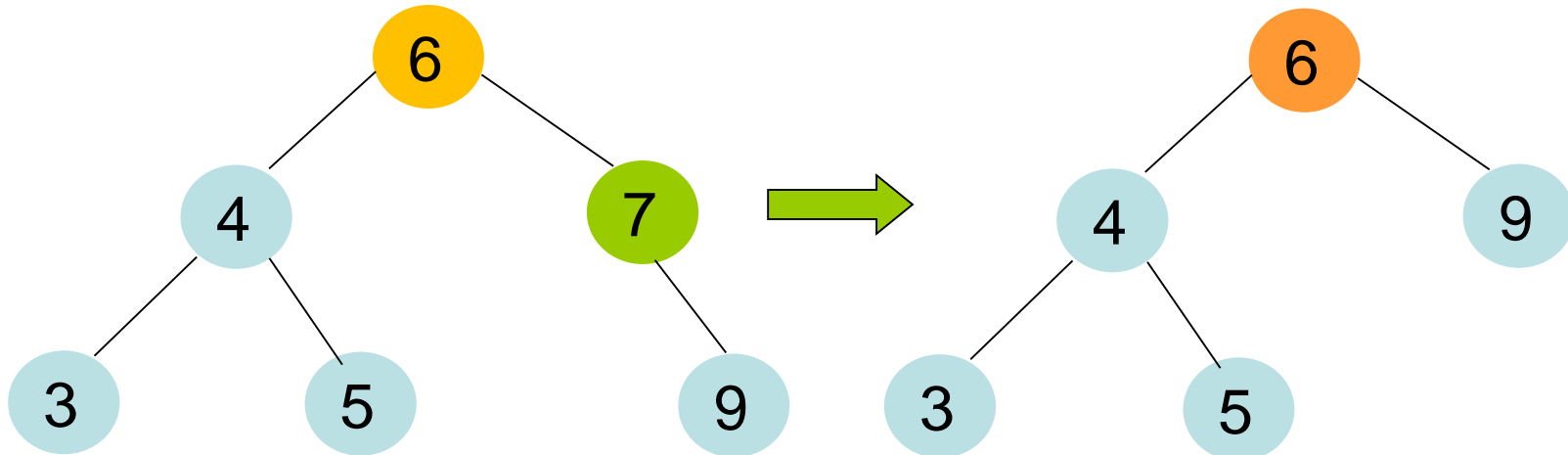
Nachfolger
hat nur ein
Kind



Binäre Suchbäume Löschen

Fall (c)

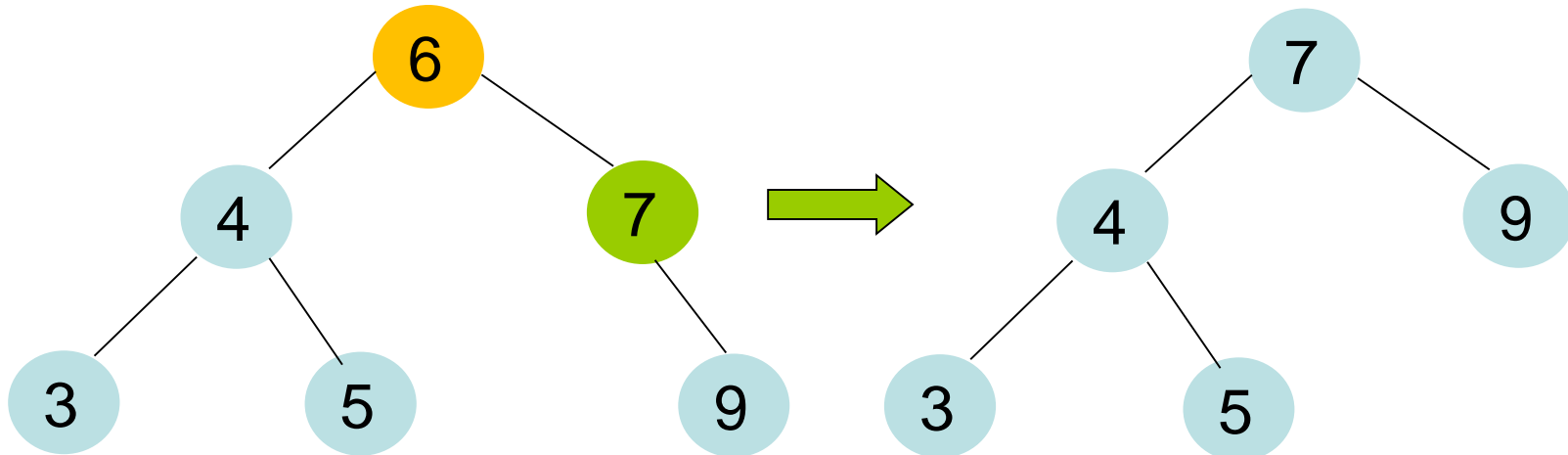
- Zu löschendes Element z hat 2 Kinder
- Schritt 1: Bestimme Nachfolger von z
- Schritt 2: Entferne Nachfolger



Binäre Suchbäume Löschen

Fall (c)

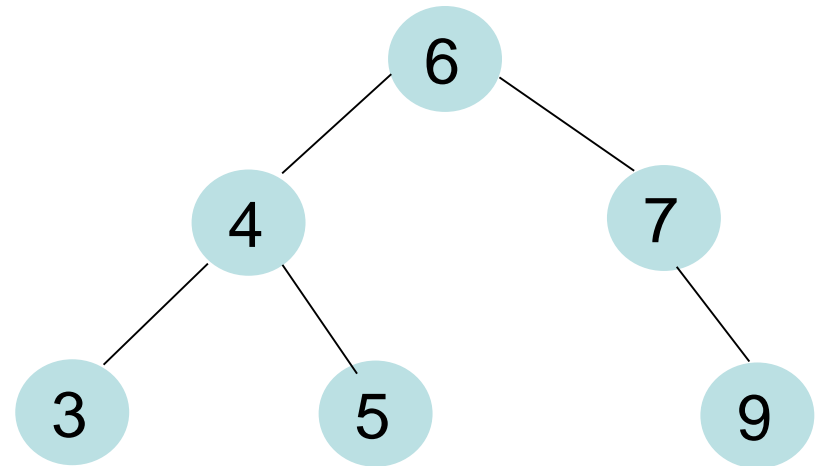
- Zu löschendes Element z hat 2 Kinder
- Schritt 1: Bestimme Nachfolger von z
- Schritt 2: Entferne Nachfolger
- Schritt 3: Ersetze z durch Nachfolger



Binäre Suchbäume Löschen

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$



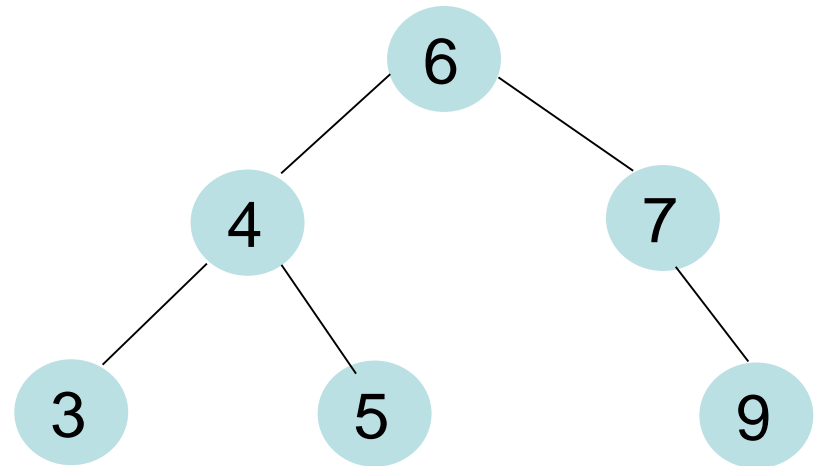
Binäre Suchbäume Löschen

Referenz auf z wird
übergeben!

Löschen(T,z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)



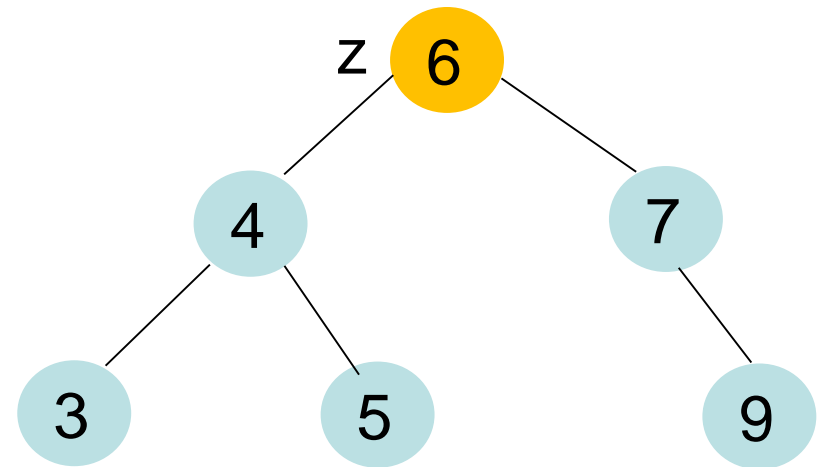
Binäre Suchbäume Löschen

Bestimme Knoten, der gelöscht werden soll. Der Knoten hat nur einen Nachfolger

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)



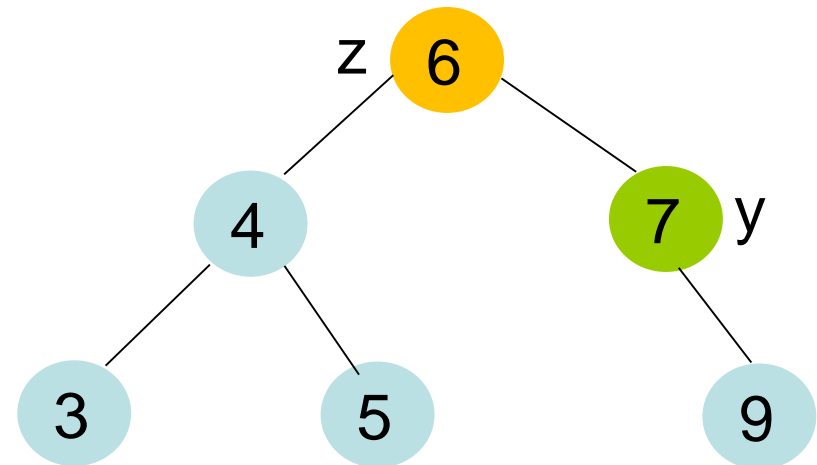
Binäre Suchbäume Löschen

Bestimme Knoten, der gelöscht werden soll. Der Knoten hat nur einen Nachfolger

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)



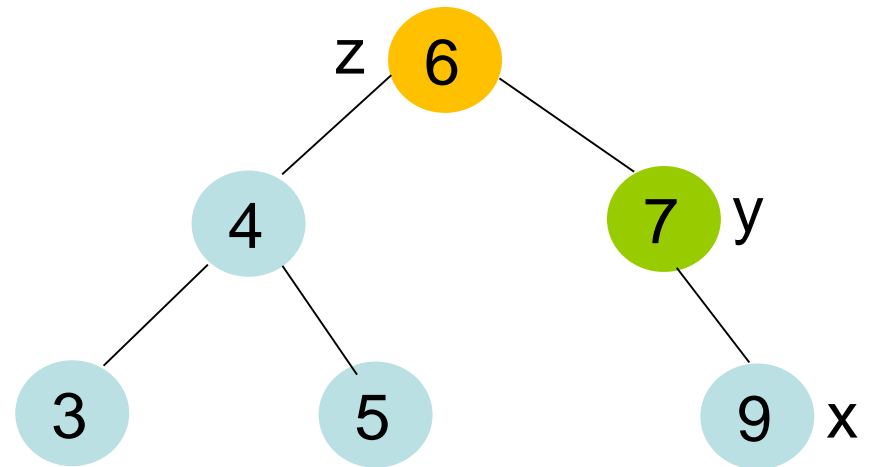
Binäre Suchbäume Löschen

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Bestimme das Kind
von y , falls existent

$\text{Löschen}(6)$



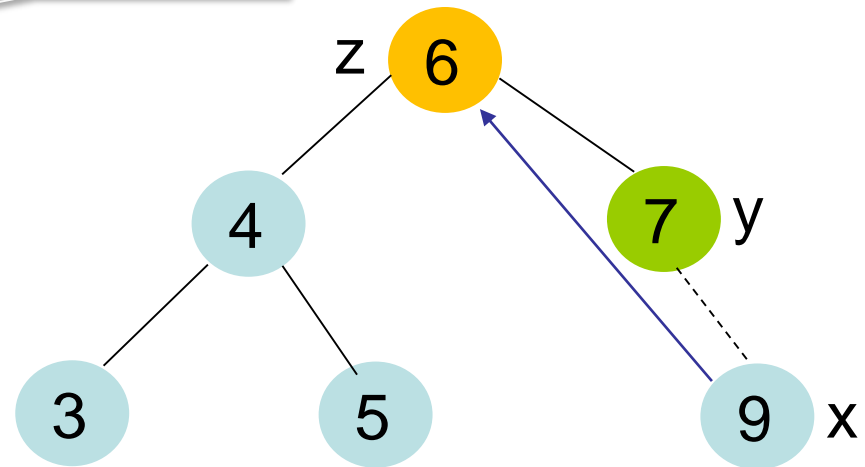
Binäre Suchbäume Löschen

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Aktualisiere
Vaterzeiger von x

Löschen(6)

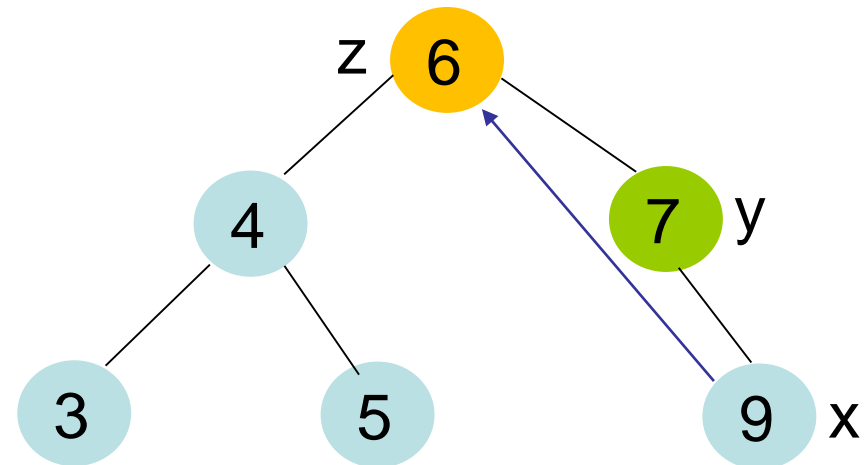


Binäre Suchbäume Löschen

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)



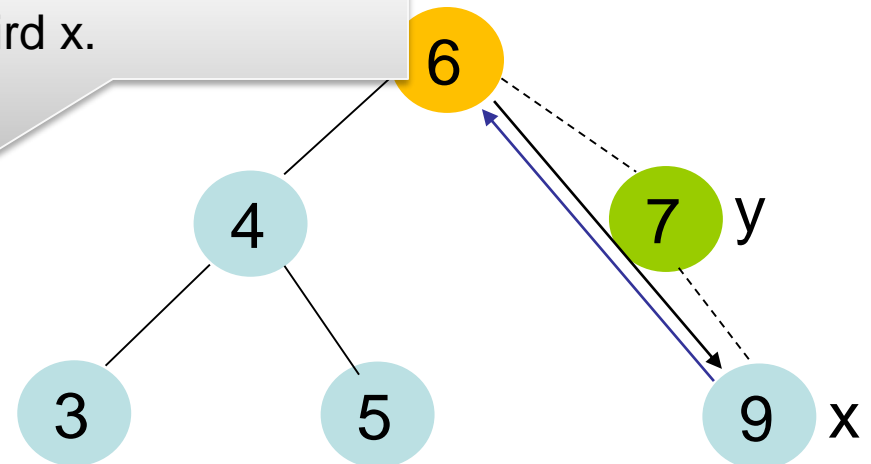
Binäre Suchbäume Löschen

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)

Das rechte Kind von z wird x .



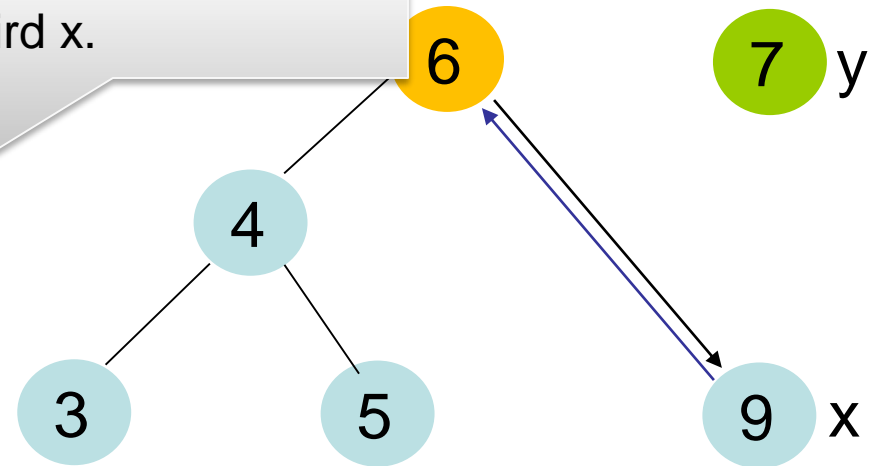
Binäre Suchbäume Löschen

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)

Das rechte Kind von z wird x .



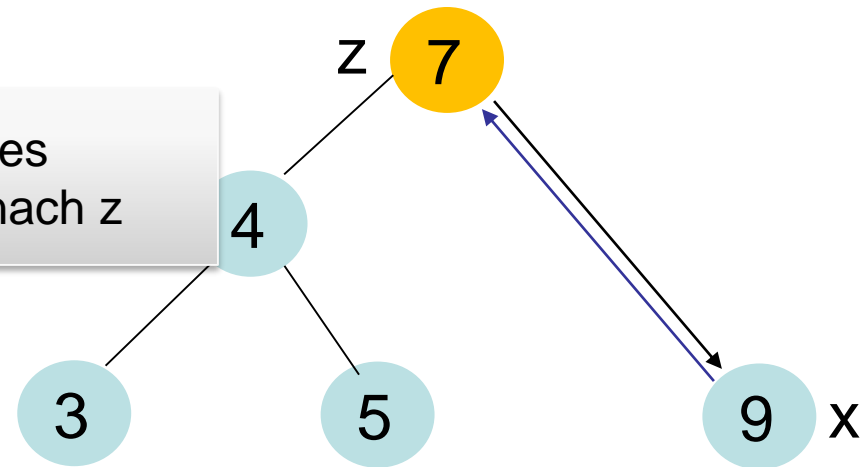
Binäre Suchbäume Löschen

Löschen(T, z)

1. **if** $lc(z) = \text{nil}$ or $rc(z) = \text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y) = \text{nil}$ **then** $\text{root}(T) \leftarrow y$
7. **else if** $y = lc(p(y))$ **then** $p(y) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Umkopieren des
Inhalts von y nach z

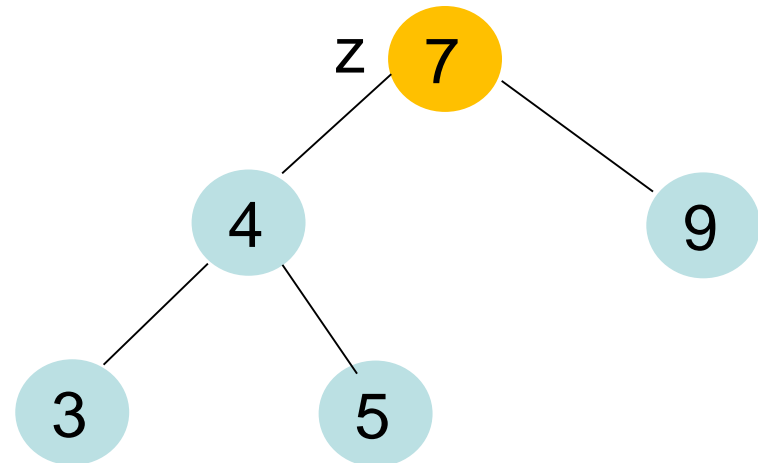
Löschen(6)



Löschen(T,z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)



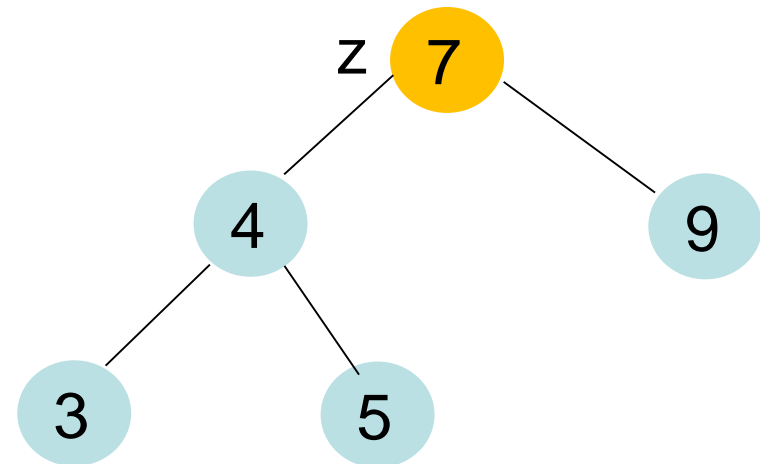
Binäre Suchbäume Löschen

Laufzeit $O(h)$

Löschen(T, z)

1. **if** $lc(z)=\text{nil}$ or $rc(z)=\text{nil}$ **then** $y \leftarrow z$
2. **else** $y \leftarrow \text{NachfolgerSuche}(z)$
3. **if** $lc(y) \neq \text{nil}$ **then** $x \leftarrow lc(y)$
4. **else** $x \leftarrow rc(y)$
5. **if** $x \neq \text{nil}$ **then** $p(x) \leftarrow p(y)$
6. **if** $p(y)=\text{nil}$ **then** $\text{root}(T) \leftarrow x$
7. **else if** $y=lc(p(y))$ **then** $lc(p(y)) \leftarrow x$
8. **else** $rc(p(y)) \leftarrow x$
9. $\text{key}(z) \leftarrow \text{key}(y)$

Löschen(6)



Binäre Suchbäume

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

Bester Fall

- Beide Teilbäume sind vorhanden, d.h. beide Teilbäume sind gleich groß
=> der Baum ist vollständig balanciert
- D.h. die Höhe ist $O(\log n)$

Schlechtester Fall

- Ein Teilbaum ist immer leer, d.h. der Baum degeneriert zu einer Liste
- D.h. die Höhe ist $O(n)$

Binäre Suchbäume

Erhoffter/erwünschter Fall

- Beide Teilbäume sind fast gleich groß => der Baum ist halbwegs balanciert
- D.h. die Höhe ist immer noch $O(\log n)$

Frage

- Wie kann man eine solche „kleine“ Höhe unter Einfügen und Löschen garantieren?
- Dafür gibt es selbstbalancierende Suchbäume, z.B. AVL oder Rot/Schwarzbäume – dazu mehr später