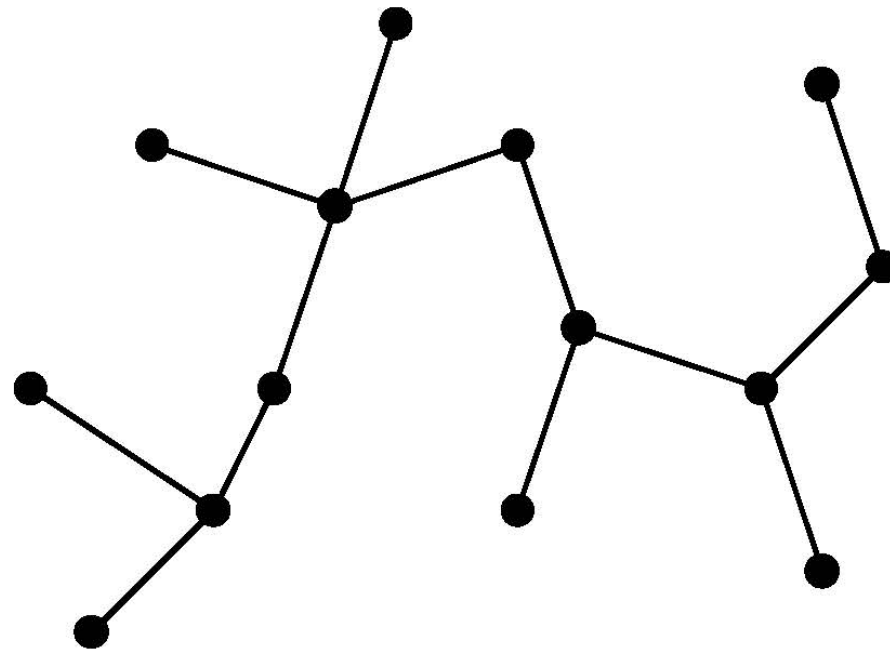


Binäre Bäume

Baum




(a)

Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend


Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen  Pfad verbunden

Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden


Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn , ist G nicht mehr zusammenhängend

Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn eine Kante entfernt wird, ist G nicht mehr zusammenhängend


Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn eine Kante entfernt wird, ist G nicht mehr zusammenhängend
- G ist zusammenhängend und $|E| =$ 

Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn eine Kante entfernt wird, ist G nicht mehr zusammenhängend
- G ist zusammenhängend und $|E| = |V| - 1$


Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn eine Kante entfernt wird, ist G nicht mehr zusammenhängend
- G ist zusammenhängend und $|E| = |V| - 1$
- G ist  und $|G| = |V| - 1$

Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn eine Kante entfernt wird, ist G nicht mehr zusammenhängend
- G ist zusammenhängend und $|E| = |V| - 1$
- G ist azyklisch und $|G| = |V| - 1$

Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn eine Kante entfernt wird, ist G nicht mehr zusammenhängend
- G ist zusammenhängend und $|E| = |V| - 1$
- G ist azyklisch und $|G| = |V| - 1$
- G ist azyklisch, aber wenn , enthält G einen Zyklus

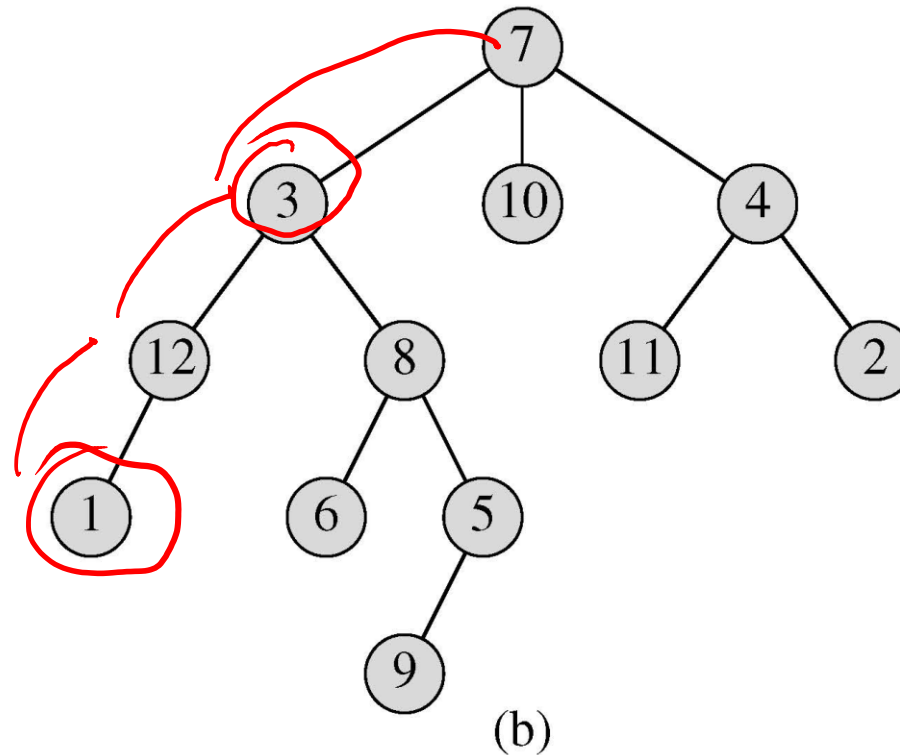
Baum

- $G = (V, E)$ ungerichtet, azyklisch, zusammenhängend
- Je zwei Knoten in G sind durch einen einfachen Pfad verbunden
- G ist zusammenhängend, aber wenn eine Kante entfernt wird, ist G nicht mehr zusammenhängend
- G ist zusammenhängend und $|E| = |V| - 1$
- G ist azyklisch und $|G| = |V| - 1$
- G ist azyklisch, aber wenn eine Kante zu G hinzugefügt wird, enthält G einen Zyklus

③ ist
Vorfahe von ①
① Nachfahre von ③

legendlich :
Knoten müssen
verschieden sein

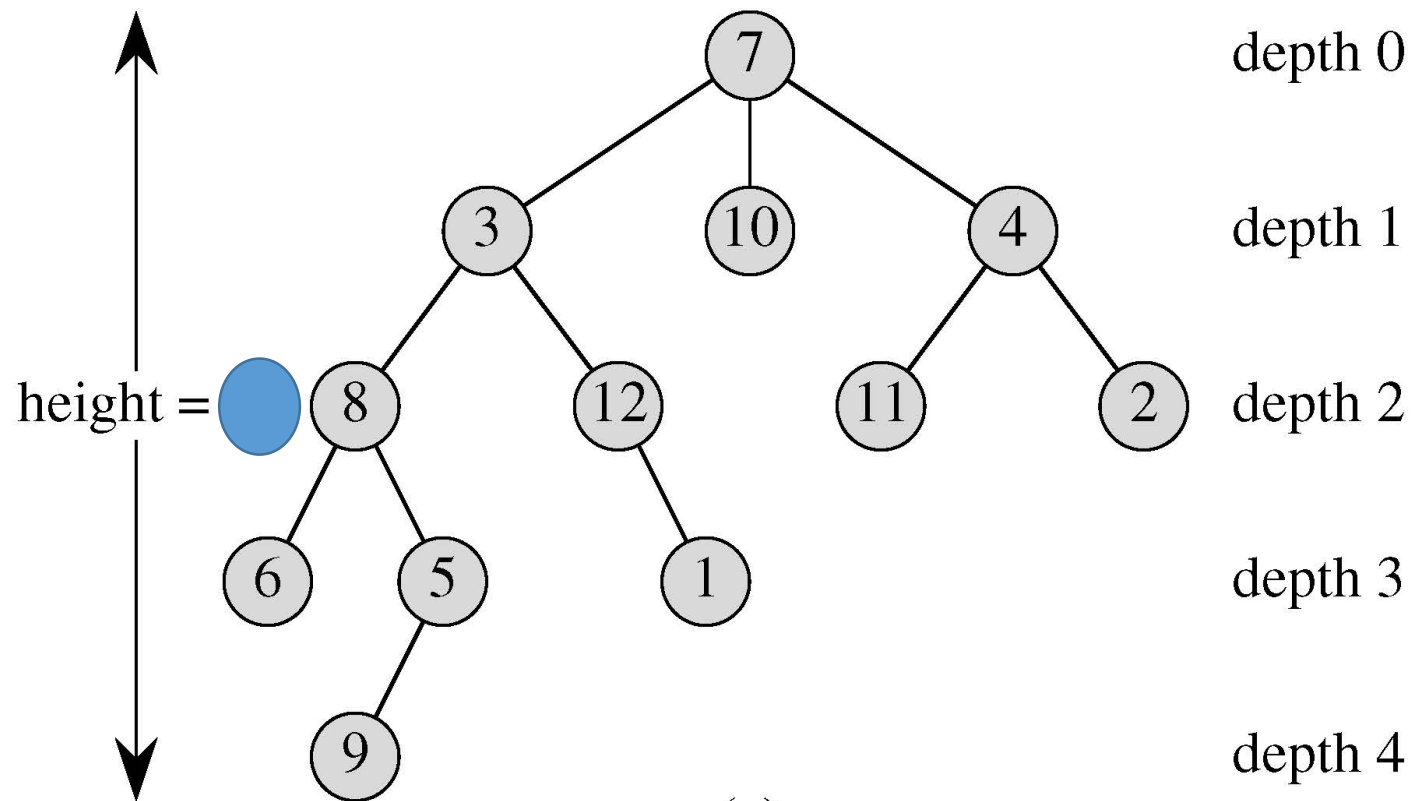
Baum mit Wurzel



Vorfahre ancestor
Nachfahre descendant
eigentlich
Elter
Kind
Geschwister
Blatt

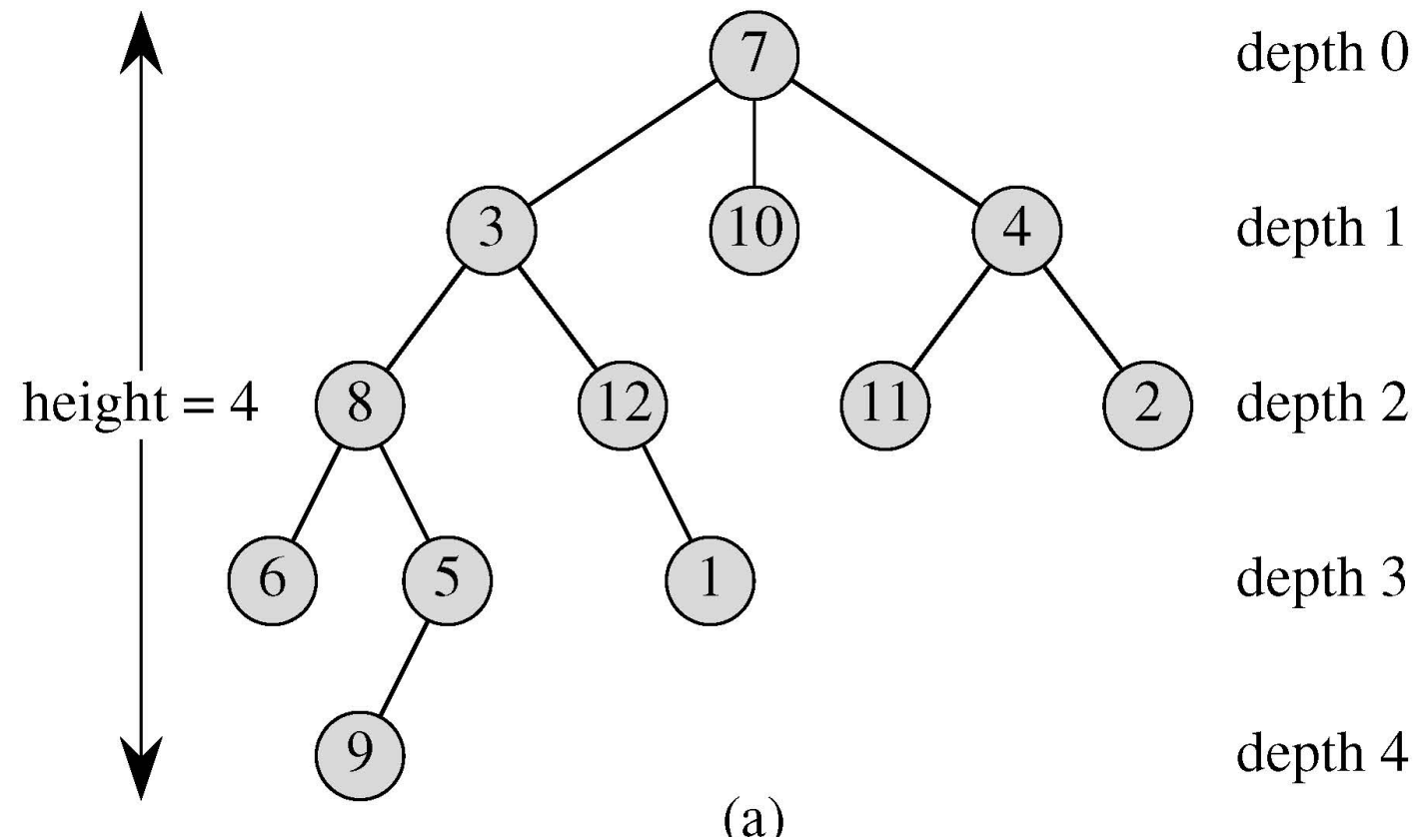
Tiefe von Knoten um ~~B~~äumen

und die Höhe von

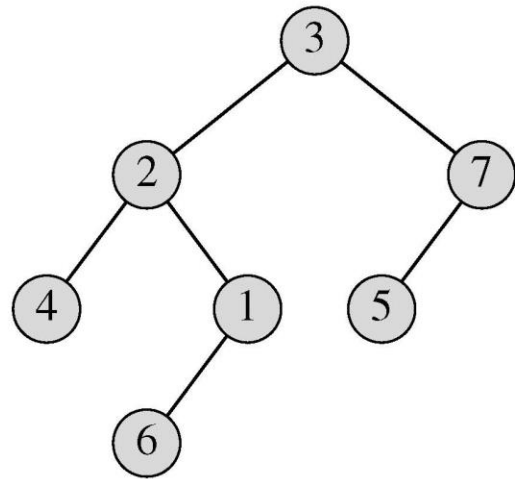


(a)

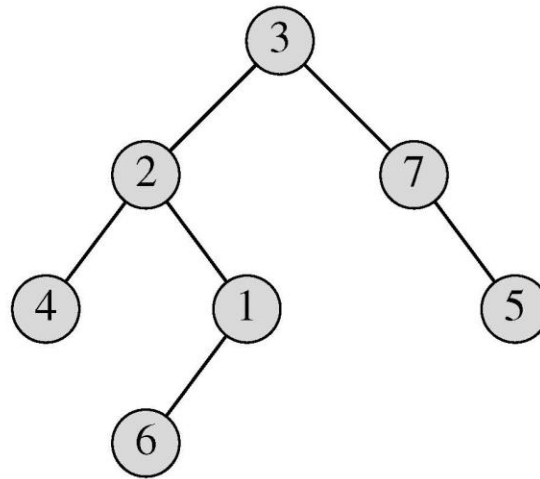
Tiefe von Knoten um Bäumen



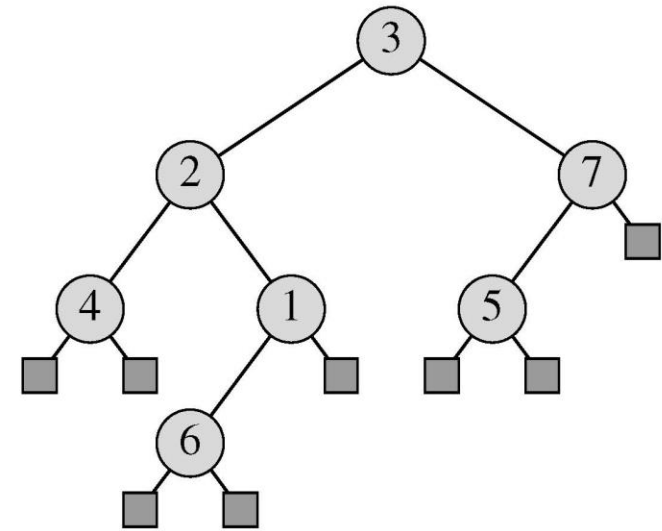
Binäre Bäume



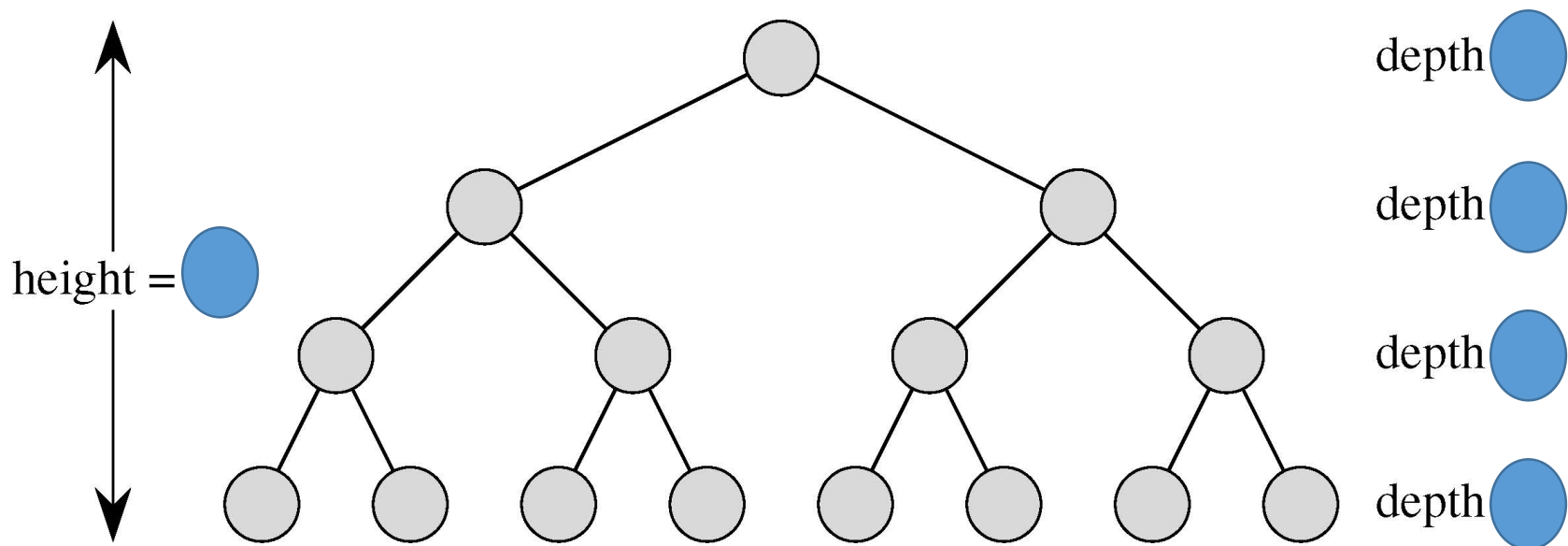
(a)



(b)



(c)



Verallgemeinerte verkettete Liste

•

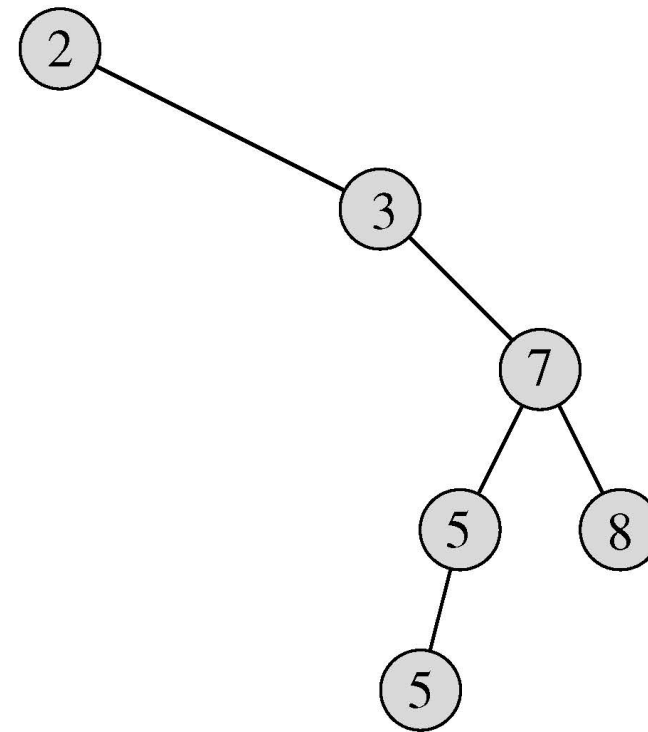
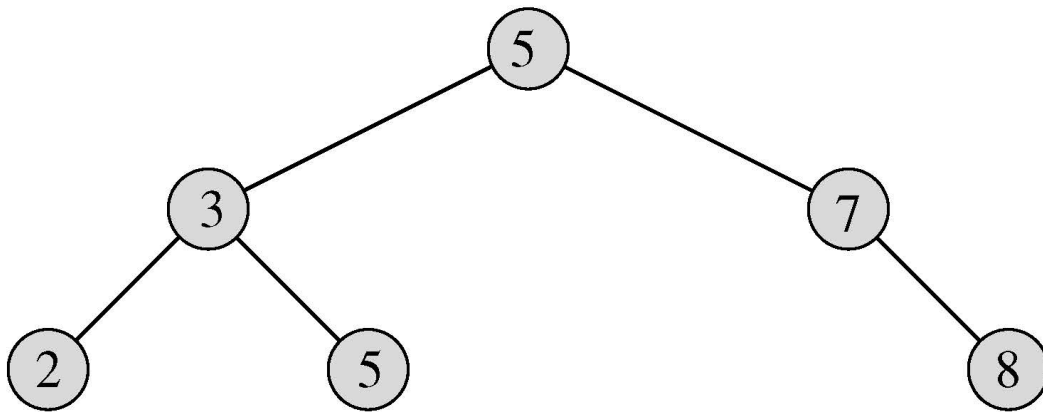
Array

⌂

Array

❖

Binäre Suchbäume von unterschiedlicher Effizienz



Was macht dieser Algorithmus?

INORDER-TREE-WALK(x)

```
1  if  $x \neq \text{NIL}$   
2      then INORDER-TREE-WALK( $\text{left}[x]$ )  
3          print  $\text{key}[x]$   
4          INORDER-TREE-WALK( $\text{right}[x]$ )
```

Ausgeben der Knoten in der richtigen Reihenfolge

INORDER-TREE-WALK(x)

```
1  if  $x \neq \text{NIL}$   
2      then INORDER-TREE-WALK( $\text{left}[x]$ )  
3          print  $\text{key}[x]$   
4          INORDER-TREE-WALK( $\text{right}[x]$ )
```


Was macht dieser Algorithmus?

TREE-SEARCH(x, k)

1 **if** $x = \text{NIL}$ or $k = \text{key}[x]$

2 **then return** x

3 **if** $k < \text{key}[x]$

4 **then return** TREE-SEARCH($\text{left}[x], k$)

5 **else return** TREE-SEARCH($\text{right}[x], k$)

Sucht rekursiv nach Knoten x mit $key[x] = k$

TREE-SEARCH(x, k)

1 **if** $x = \text{NIL}$ or $k = key[x]$

2 **then return** x

3 **if** $k < key[x]$

4 **then return** **TREE-SEARCH**($left[x], k$)

5 **else return** **TREE-SEARCH**($right[x], k$)

Komplexität?

TREE-SEARCH(x, k)

```
1  if  $x = \text{NIL}$  or  $k = \text{key}[x]$ 
2      then return  $x$ 
3  if  $k < \text{key}[x]$ 
4      then return TREE-SEARCH( $\text{left}[x], k$ )
5      else return TREE-SEARCH( $\text{right}[x], k$ )
```

Komplexität

Komplexität = $O(\text{Höhe von } x)$

TREE-SEARCH(x, k)

1 **if** $x = \text{NIL}$ or $k = \text{key}[x]$

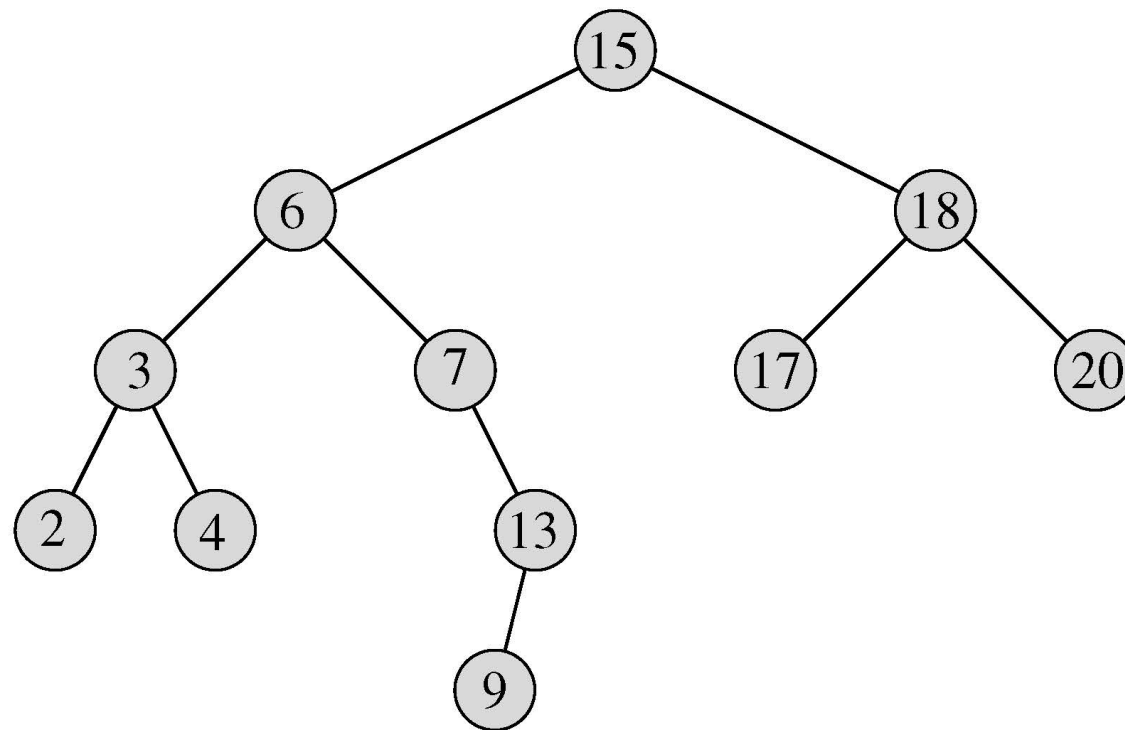
2 **then return** x

3 **if** $k < \text{key}[x]$

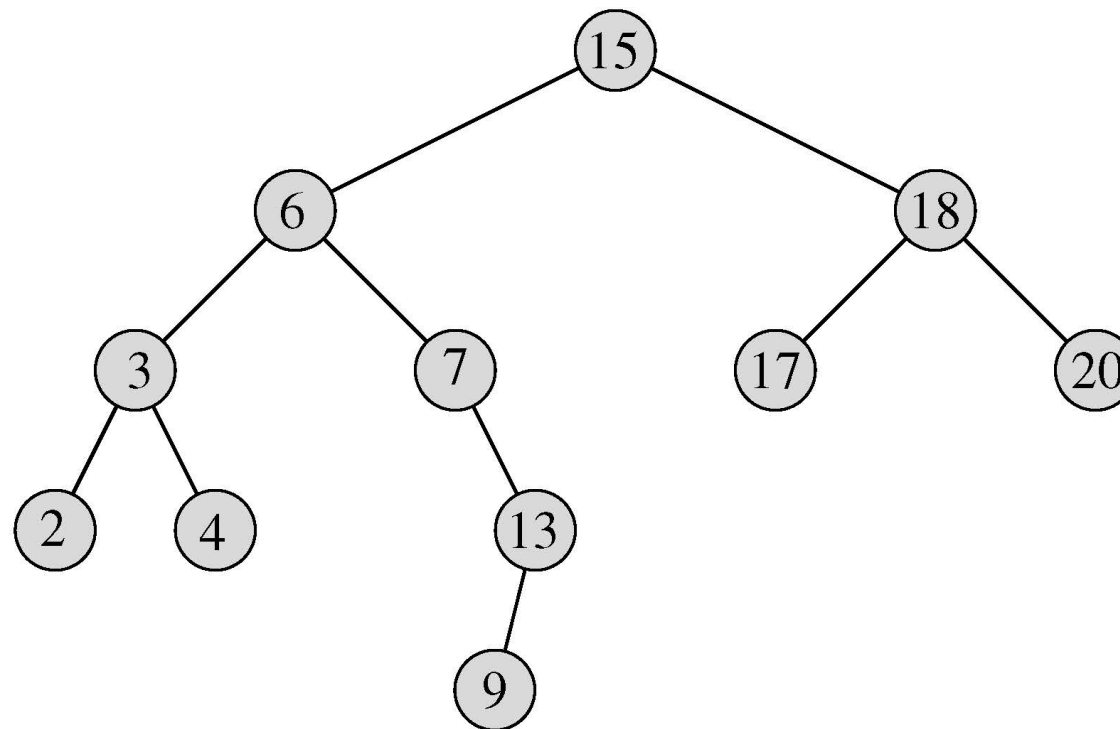
4 **then return** TREE-SEARCH($\text{left}[x], k$)

5 **else return** TREE-SEARCH($\text{right}[x], k$)

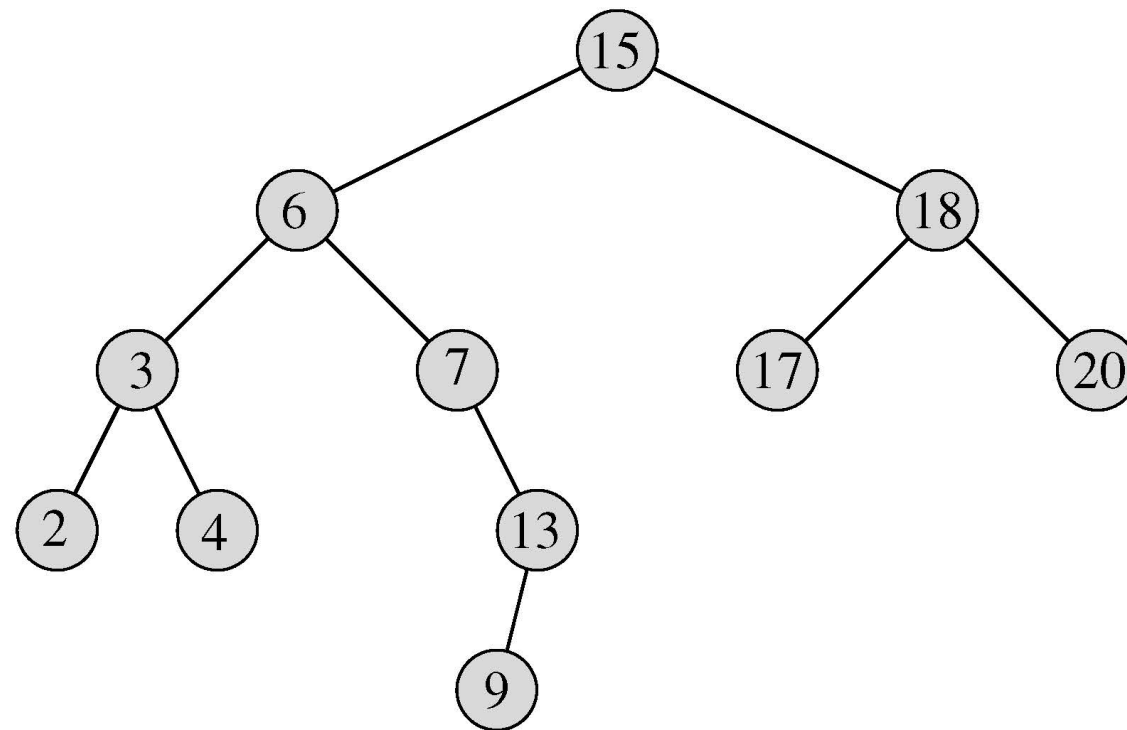
Suche nach 13



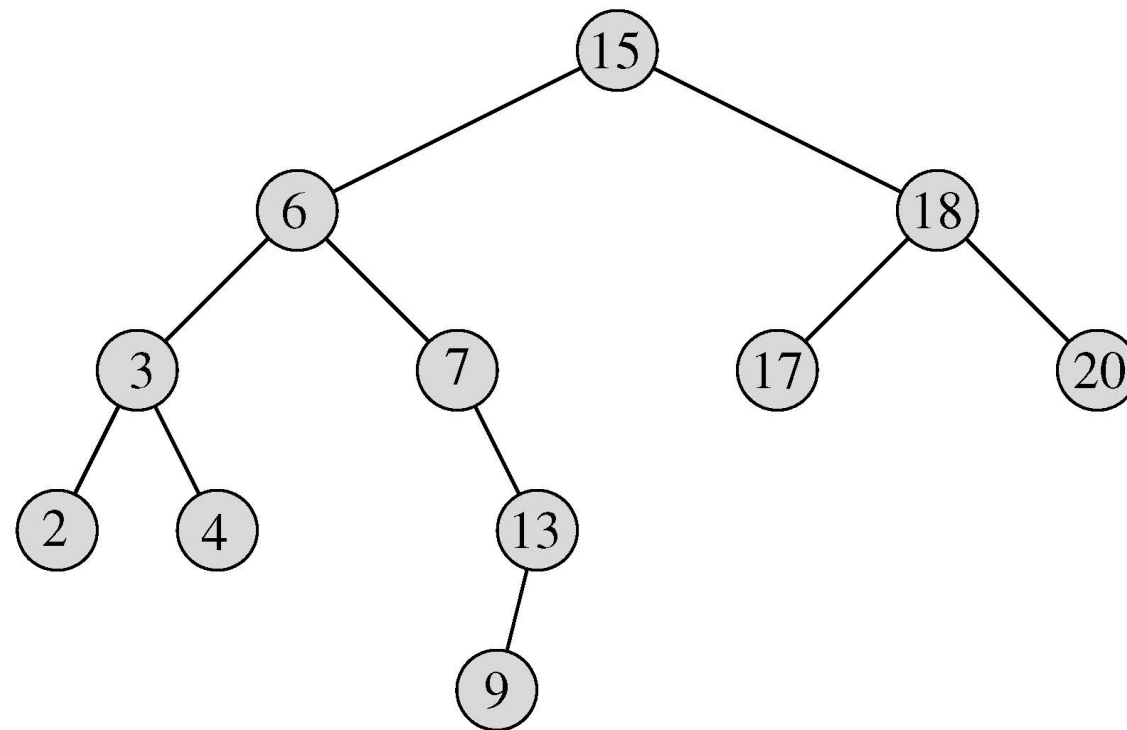
Suche nach 13



Suche nach 19



Suche nach 19



Was macht dieser Algorithmus?

ITERATIVE-TREE-SEARCH(x, k)

```
1  while  $x \neq \text{NIL}$  and  $k \neq \text{key}[x]$ 
2      do if  $k < \text{key}[x]$ 
3          then  $x \leftarrow \text{left}[x]$ 
4          else  $x \leftarrow \text{right}[x]$ 
5  return  $x$ 
```

Sucht iterativ nach Knoten x mit $\text{key}[x] = k$

ITERATIVE-TREE-SEARCH(x, k)

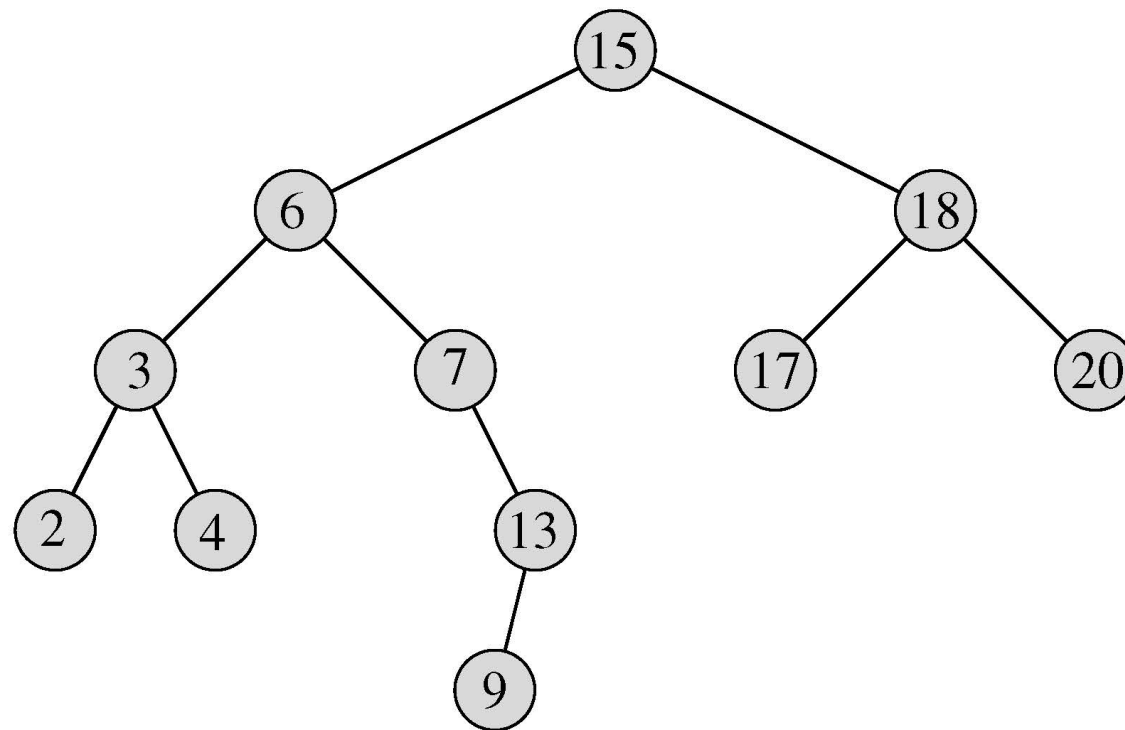
```
1  while  $x \neq \text{NIL}$  and  $k \neq \text{key}[x]$ 
2      do if  $k < \text{key}[x]$ 
3          then  $x \leftarrow \text{left}[x]$ 
4          else  $x \leftarrow \text{right}[x]$ 
5  return  $x$ 
```

Komplexität = $O(\text{Höhe von } x)$

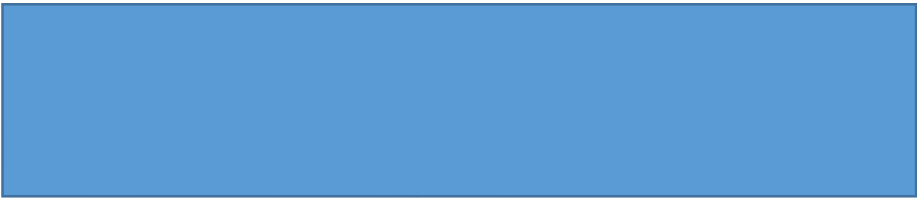
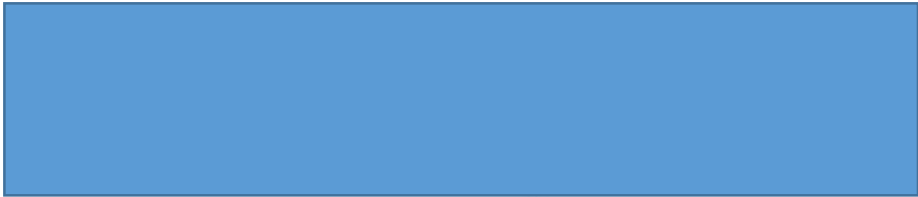
ITERATIVE-TREE-SEARCH(x, k)

```
1  while  $x \neq \text{NIL}$  and  $k \neq \text{key}[x]$ 
2      do if  $k < \text{key}[x]$ 
3          then  $x \leftarrow \text{left}[x]$ 
4          else  $x \leftarrow \text{right}[x]$ 
5  return  $x$ 
```

Finde Minimum



TREE-MINIMUM(x)

```
1  while   
2      do   
3  return  $x$ 
```

TREE-MINIMUM(x)


```
1  while  $left[x] \neq \text{NIL}$   
2      do  $x \leftarrow left[x]$   
3  return  $x$ 
```

TREE-MAXIMUM(x)

```
1  while  $right[x] \neq \text{NIL}$   
2      do  $x \leftarrow right[x]$   
3  return  $x$ 
```

TREE-SUCCESSOR(x)

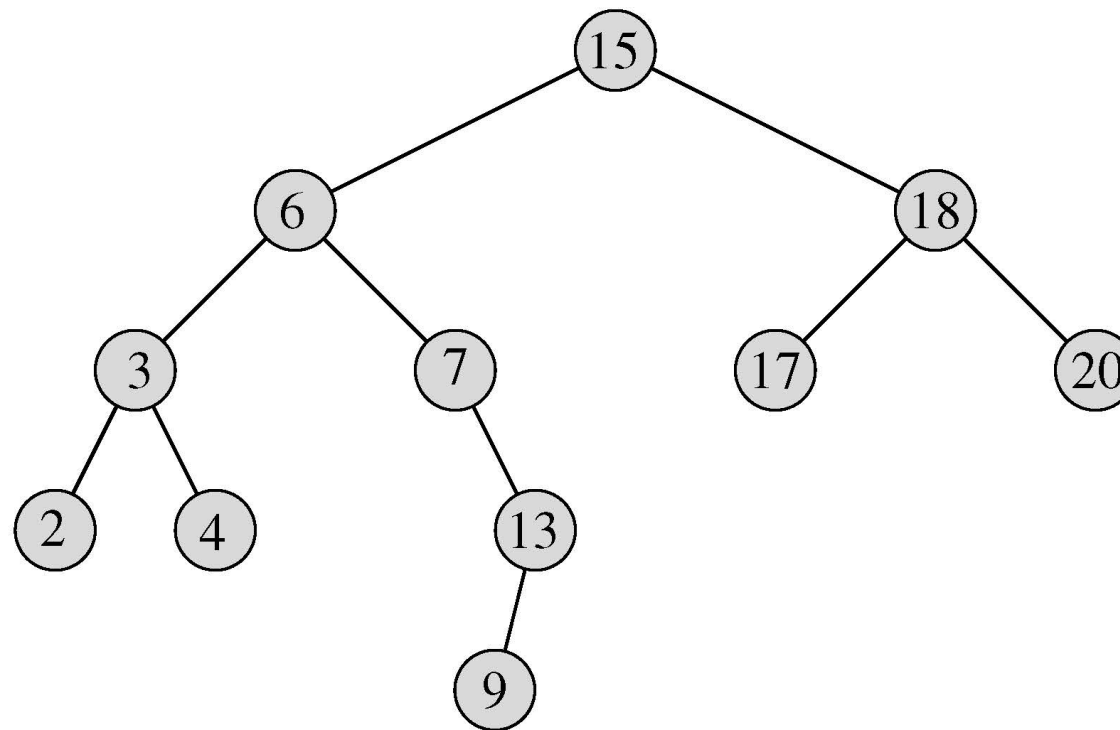
```
1  if  $right[x] \neq \text{NIL}$ 
2      then return TREE-MINIMUM( $right[x]$ )
3   $y \leftarrow p[x]$ 
4  while  $y \neq \text{NIL}$  and
5      do  $x \leftarrow y$ 
6       $y \leftarrow p[y]$ 
7  return  $y$ 
```



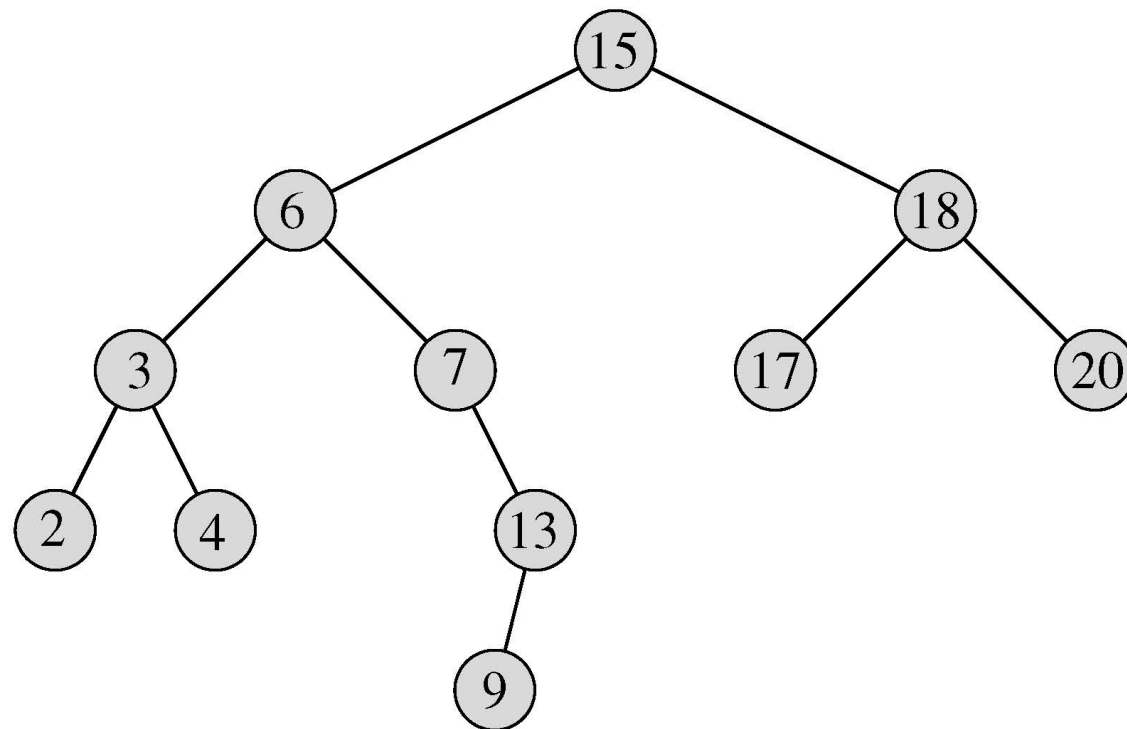
TREE-SUCCESSOR(x)

```
1  if  $right[x] \neq \text{NIL}$ 
2      then return TREE-MINIMUM( $right[x]$ )
3   $y \leftarrow p[x]$ 
4  while  $y \neq \text{NIL}$  and  $x = right[y]$ 
5      do  $x \leftarrow y$ 
6       $y \leftarrow p[y]$ 
7  return  $y$ 
```

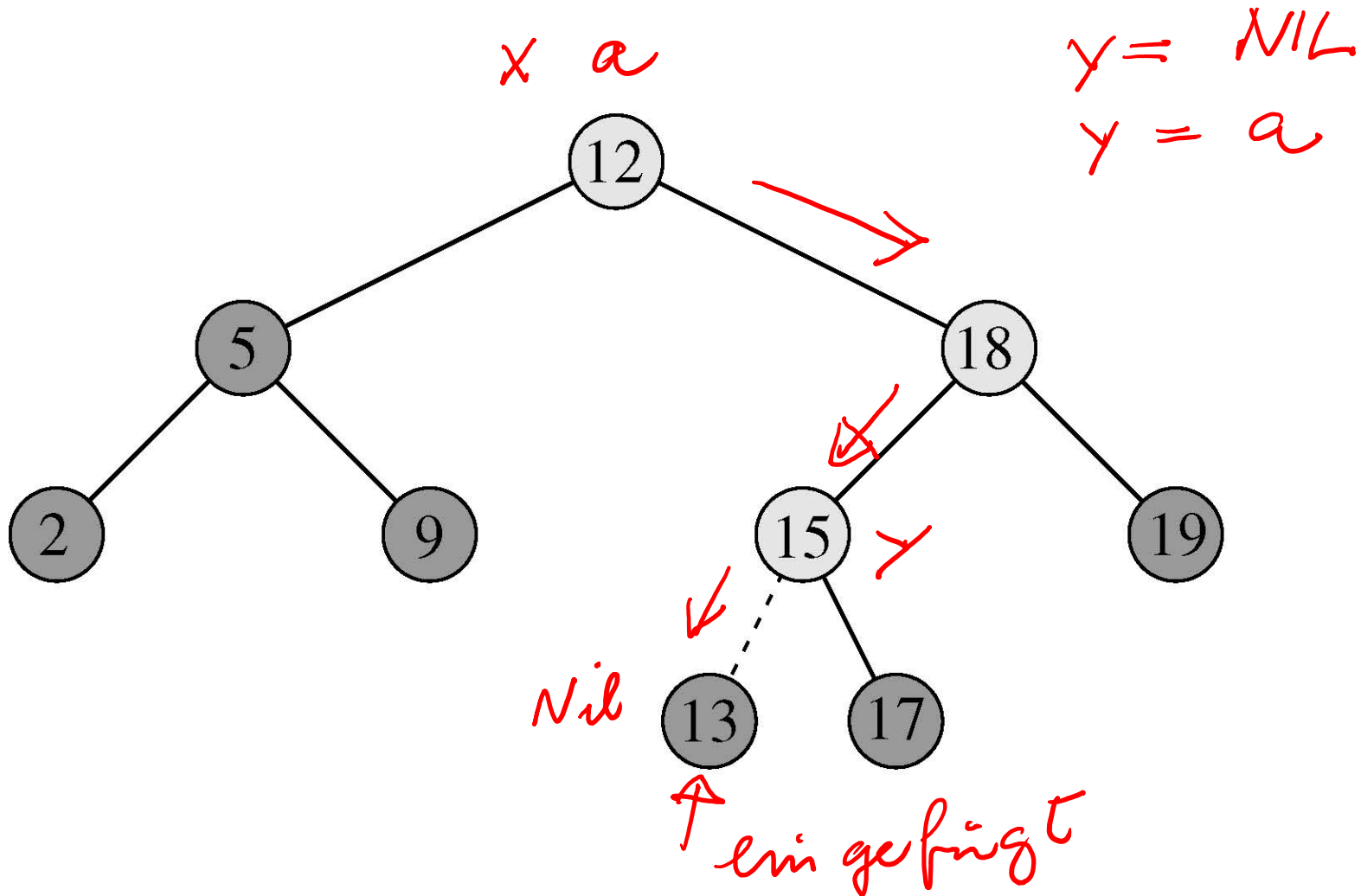
Nachfolger von 4



Nachfolger von 15



Einfügen von 13



Einfügen

TREE-INSERT(T, z)

```
1   $y \leftarrow \text{NIL}$ 
2   $x \leftarrow \text{root}[T]$ 
3  while  $x \neq \text{NIL}$ 
4      do  $y \leftarrow x$ 
5      if key[z] < key[x]
6          then  $x \leftarrow \text{left}[x]$ 
7          else  $x \leftarrow \text{right}[x]$ 
8   $p[z] \leftarrow y$ 
9  if  $y = \text{NIL}$ 
10     then  $\text{root}[T] \leftarrow z$ 
11     else if  $\text{key}[z] < \text{key}[y]$ 
12         then  $\text{left}[y] \leftarrow z$ 
13         else  $\text{right}[y] \leftarrow z$ 
```

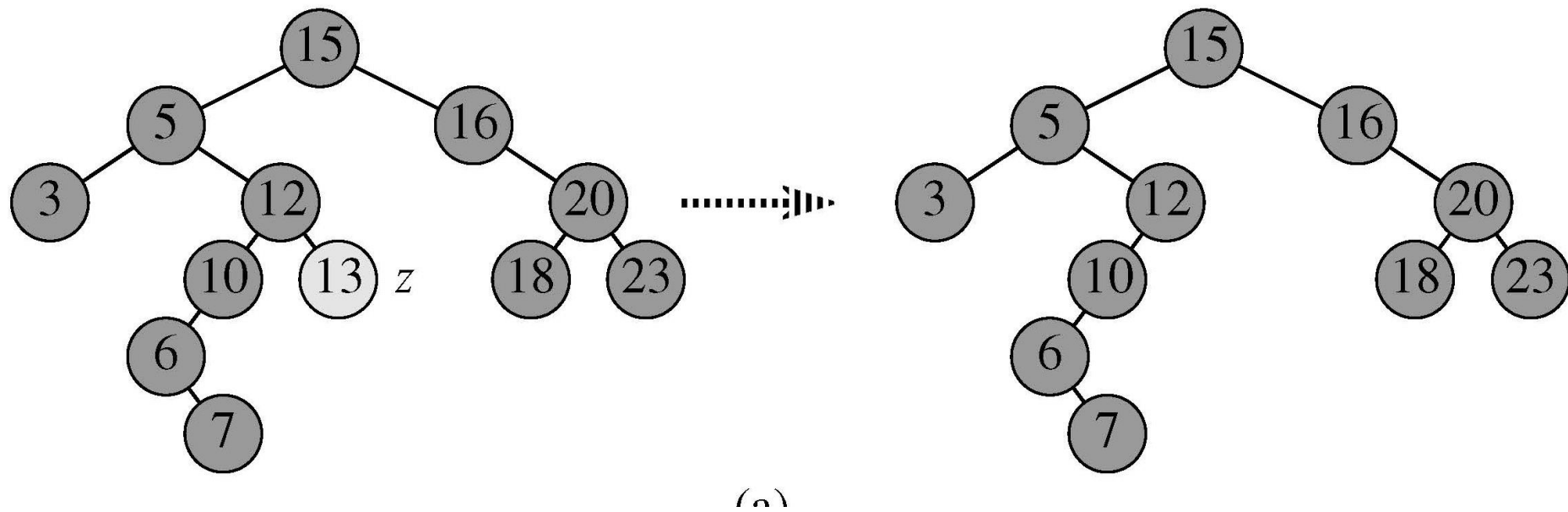
Einfügen

TREE-INSERT(T, z)

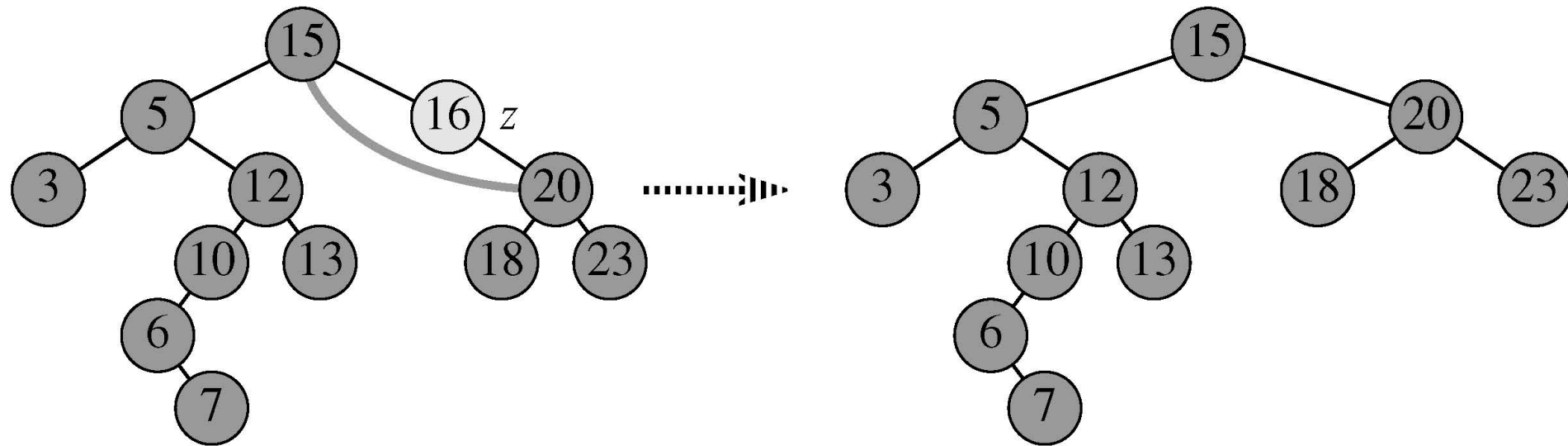
```
1   $y \leftarrow \text{NIL}$ 
2   $x \leftarrow \text{root}[T]$ 
3  while  $x \neq \text{NIL}$ 
4      do  $y \leftarrow x$ 
5          if  $\text{key}[z] < \text{key}[x]$ 
6              then  $x \leftarrow \text{left}[x]$ 
7              else  $x \leftarrow \text{right}[x]$ 
8   $p[z] \leftarrow y$ 
9  if  $y = \text{NIL}$ 
10     then  $\text{root}[T] \leftarrow z$ 
11     else if  $\text{key}[z] < \text{key}[y]$ 
12         then  $\text{left}[y] \leftarrow z$ 
13         else  $\text{right}[y] \leftarrow z$ 
```

▷ Tree T was empty

Löschen von Blatt 13

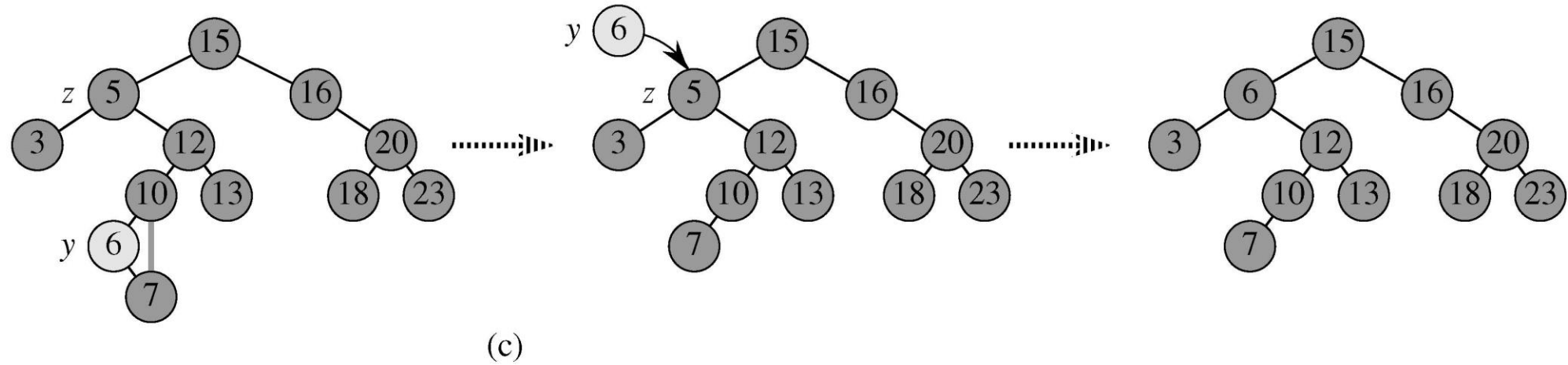


Löschen von Knoten 16 mit einem Kind







(b)

Löschen von Knoten 5 mit zwei Kindern






Löschen

TREE-DELETE(T, z)

```
1  if  $left[z] = \text{NIL}$  or  $right[z] = \text{NIL}$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow$  
4  if  $left[y]$  
5    then  $x \leftarrow left[y]$ 
6    else  $x \leftarrow right[y]$ 
7  if  $x \neq \text{NIL}$ 
8    then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = \text{NIL}$ 
10   then 
11   else if  $y = left[p[y]]$ 
12     then  $left[p[y]] \leftarrow x$ 
13     else  $right[p[y]] \leftarrow x$ 
14 if  $y \neq z$ 
15   then 
16
17 return  $y$ 
```




Löschen

TREE-DELETE(T, z)

```
1  if  $left[z] = \text{NIL}$  or  $right[z] = \text{NIL}$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
4  if  $left[y]$  
5    then  $x \leftarrow left[y]$ 
6    else  $x \leftarrow right[y]$ 
7  if  $x \neq \text{NIL}$ 
8    then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = \text{NIL}$ 
10   then 
11   else if  $y = left[p[y]]$ 
12     then  $left[p[y]] \leftarrow x$ 
13     else  $right[p[y]] \leftarrow x$ 
14  if  $y \neq z$ 
15   then 
16
17  return  $y$ 
```

Löschen

TREE-DELETE(T, z)

```
1  if  $left[z] = \text{NIL}$  or  $right[z] = \text{NIL}$ 
2    then  $y \leftarrow z$ 
3    else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
4  if  $left[y] \neq \text{NIL}$ 
5    then  $x \leftarrow left[y]$ 
6    else  $x \leftarrow right[y]$ 
7  if  $x \neq \text{NIL}$ 
8    then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = \text{NIL}$ 
10   then 
11   else if  $y = left[p[y]]$ 
12     then  $left[p[y]] \leftarrow x$ 
13     else  $right[p[y]] \leftarrow x$ 
14  if  $y \neq z$ 
15   then 
16   
17  return  $y$ 
```

Löschen

TREE-DELETE(T, z)

```
1  if  $left[z] = \text{NIL}$  or  $right[z] = \text{NIL}$ 
2      then  $y \leftarrow z$ 
3      else  $y \leftarrow \text{TREE-SUCCESSOR}(z)$ 
4  if  $left[y] \neq \text{NIL}$ 
5      then  $x \leftarrow left[y]$ 
6      else  $x \leftarrow right[y]$ 
7  if  $x \neq \text{NIL}$ 
8      then  $p[x] \leftarrow p[y]$ 
9  if  $p[y] = \text{NIL}$ 
10     then  $root[T] \leftarrow x$ 
11     else if  $y = left[p[y]]$ 
12         then  $left[p[y]] \leftarrow x$ 
13         else  $right[p[y]] \leftarrow x$ 
14 if  $y \neq z$ 
15     then  $key[z] \leftarrow key[y]$ 
16         copy  $y$ 's satellite data into  $z$ 
17 return  $y$ 
```