

2.5 Bäume

2.5.1 Binäre Suchbäume

2.5.2 Optimale Suchbäume

2.5.3 Balancierte Bäume

2.5.4 Skip-Listen

2.5.5 Union-Find-Strukturen



- Nochmal: Suchen in Mengen ...
- Sortieren und Suchen
 - Nur sinnvoll für statische Mengen
 - Insert(), Delete() sind $O(n)$
 - Search() ist $O(\log n)$
 - Speichermanagement



- Nochmal: Suchen in Mengen ...
- Hashing
 - Zahl der Objekte muss vorher bekannt sein → Tabellengröße
 - Insert(), Search() sind $O(1)$
 - Delete() nicht praktikabel bei geschlossenem Hashing



- Nochmal: Suchen in Mengen ...
- Binärbäume
 - Beliebig dynamisch erweiterbar
 - Insert(), Delete(), Search() sind $O(\log n)$
- Knoten enthalten mindestens
 - Zeiger **P** zum Vorgänger
 - Zeiger **L**, **R** zum linken, rechten Nachfolger
 - Suchschlüssel **X**



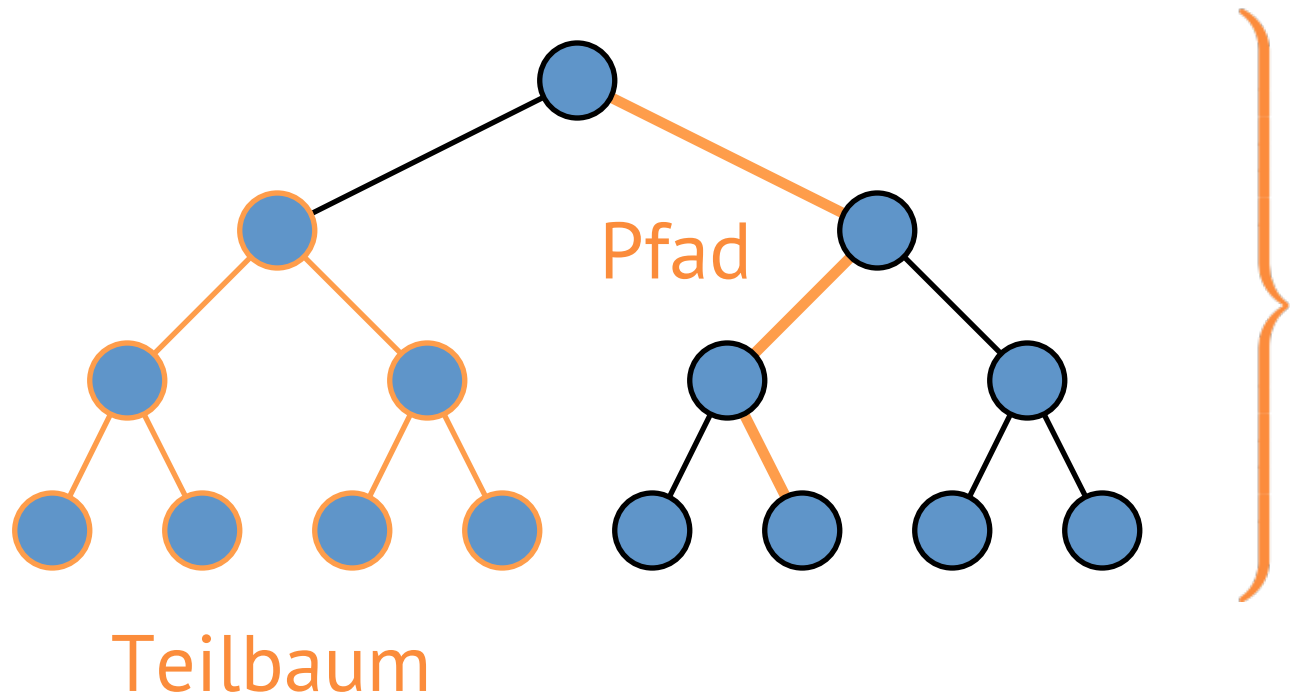
- Effiziente und flexible Suchstruktur
 - Speichere Daten in jedem Knoten
 - Speichere Daten nur in den Blättern (z.B. Suchstruktur im Hauptspeicher, Datenblöcke auf der Festplatte)
 - Begriffe: Ordnung, Höhe, Pfad, ...



Bäume

Grad=2

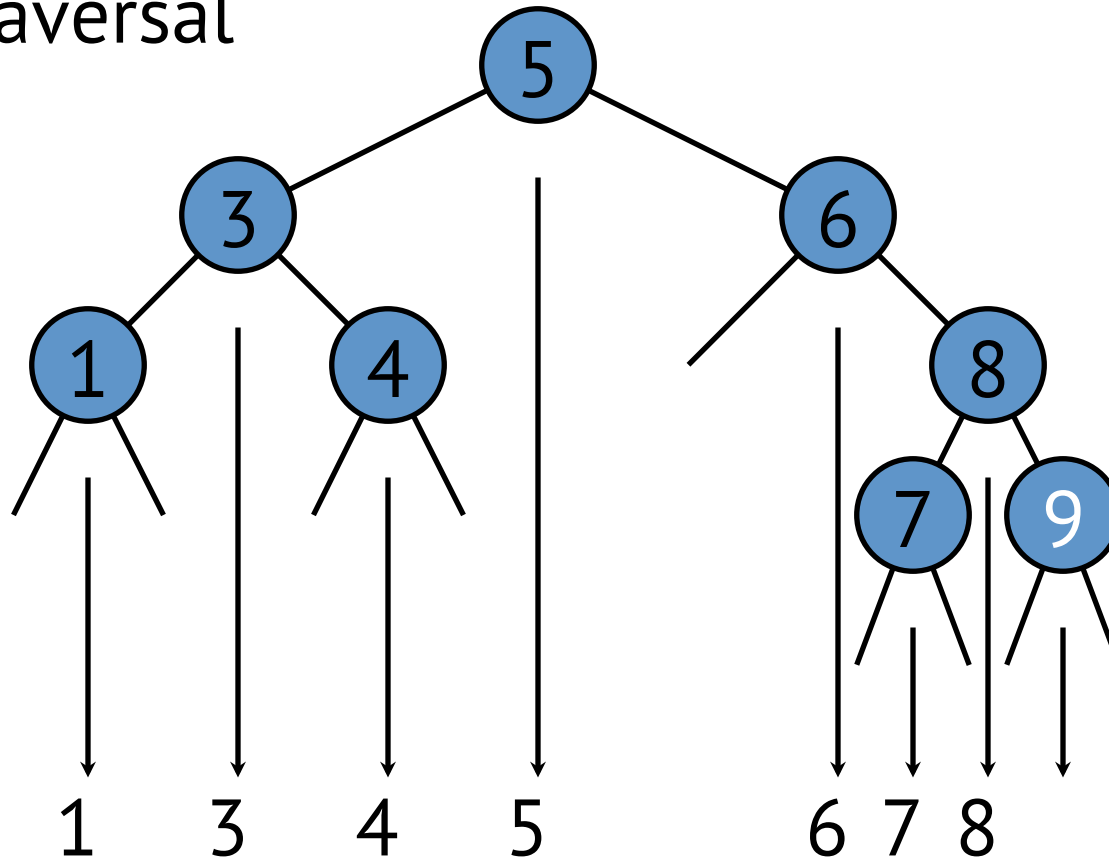
Höhe=3



- Suchbäume
 - Sei X_i ein Knoten und $X_1 \dots X_{i-1}$ die Knoten im linken und $X_{i+1} \dots X_n$ die Knoten im rechten Teilbaum, dann gilt
$$\max \{X_1 \dots X_{i-1}\} < X_i < \min \{X_{i+1} \dots X_n\}$$
- Insert(), Delete(), Search(), ...
 - Aufwand proportional zur Pfadlänge



Inorder Traversal



Operationen

- $\text{Search}(X,R)$
Suche Schlüssel X in Teilbaum R
- $\text{Min}(N)$, $\text{Max}(N)$
Finde minimales/maximales Element in Teilbaum N
- $\text{Successor}(N)$, $\text{Predecessor}(N)$
Finde Vorgänger/Nachfolger zum Knoten N
- $\text{Insert}(N,R)$...
- $\text{Delete}(N,R)$...



Search()

- Search(X,R)
 - if R = NIL or X = R.X then
 - return R
 - else
 - if X < R.X then
 - return Search(X,R.L)
 - else
 - return Search(X,R.R)



Min(), Max()

- Min(N)

```
while N.L ≠ NIL do
```

```
    N = N.L
```

```
return N
```

- Max(N)

```
while N.R ≠ NIL do
```

```
    N = N.R
```

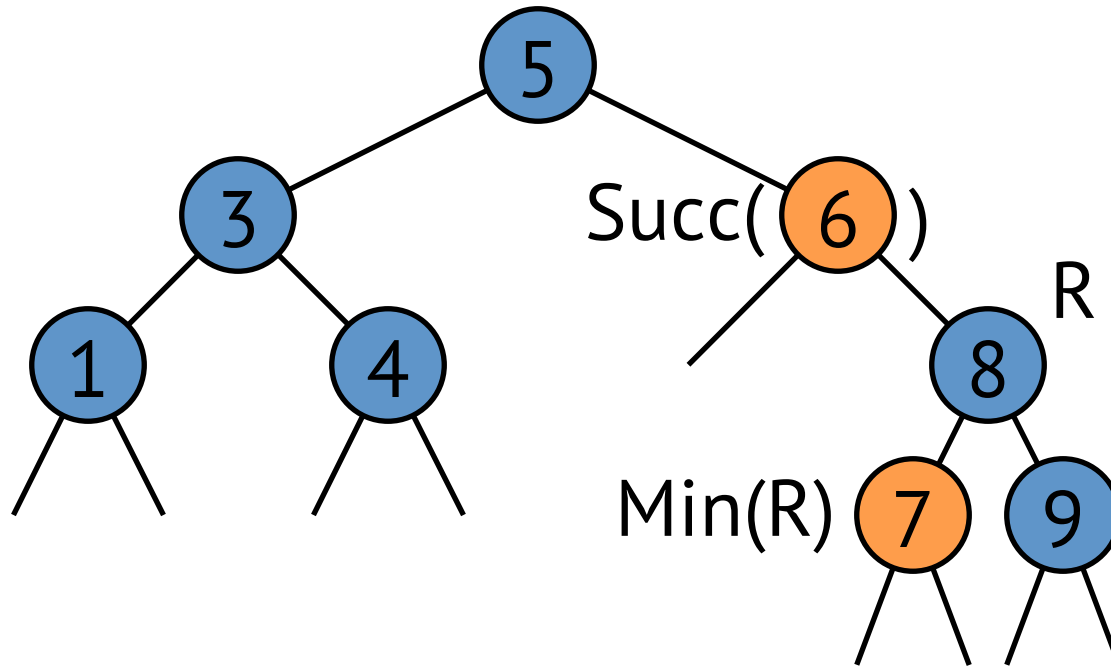
```
return N
```



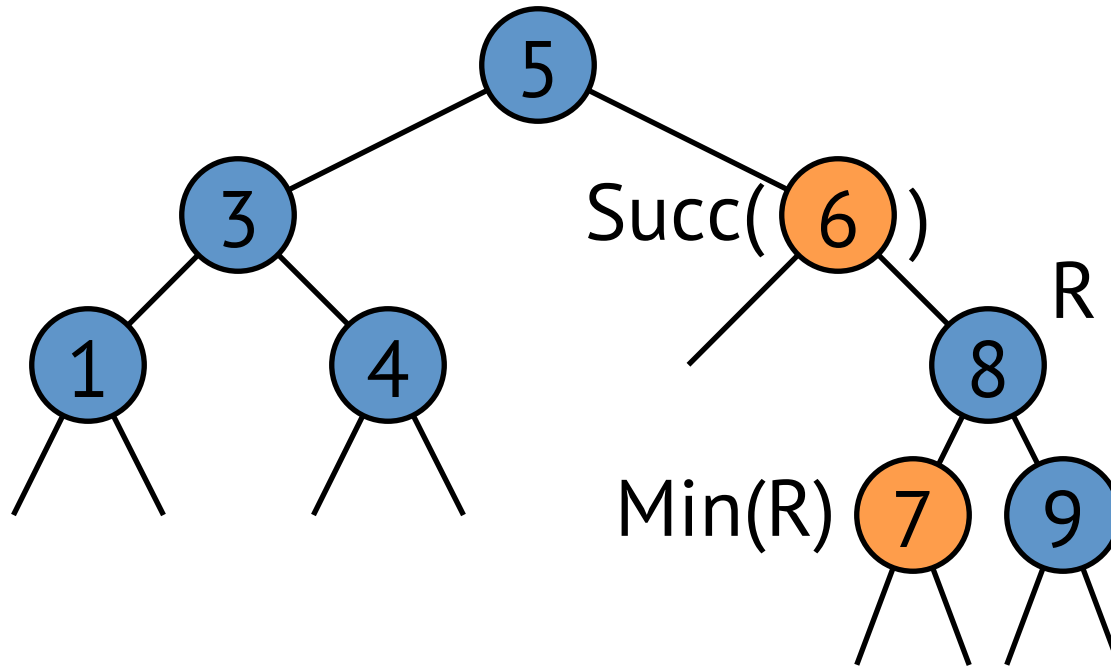
Successor()

- Successor(N)
 - if $N.R \neq \text{NIL}$ then
 - return $\text{Min}(N.R)$
 - else
 - $Q \leftarrow N.P$
 - while $Q \neq \text{NIL}$ and $N = Q.R$ do
 - $N \leftarrow Q$
 - $Q \leftarrow Q.P$
 - return Q

Successor()



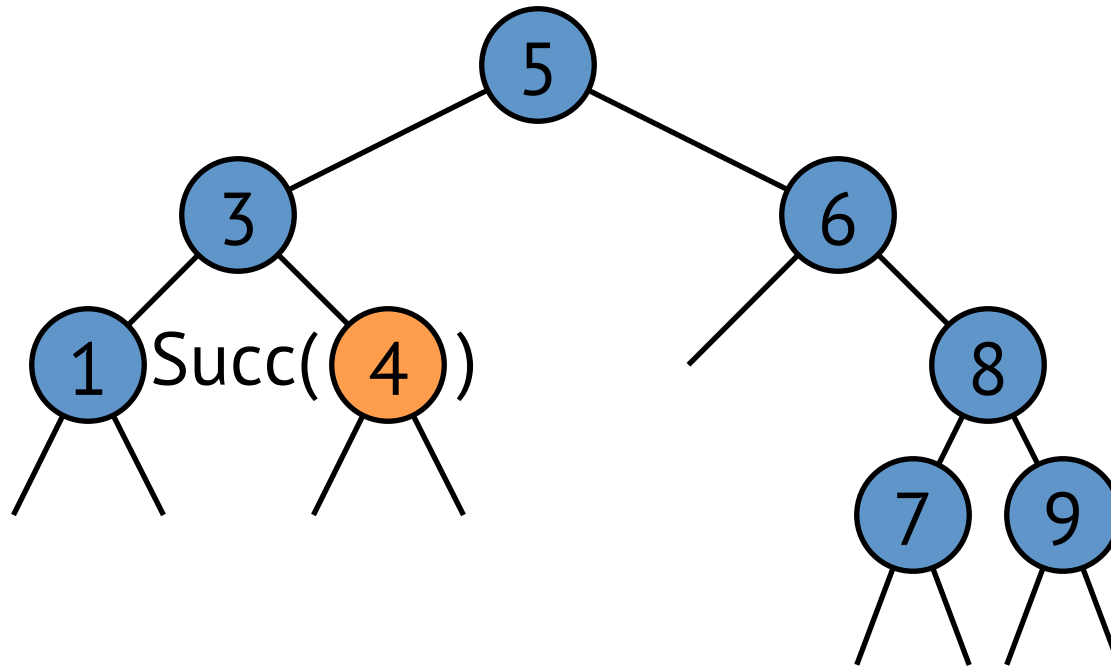
Successor()



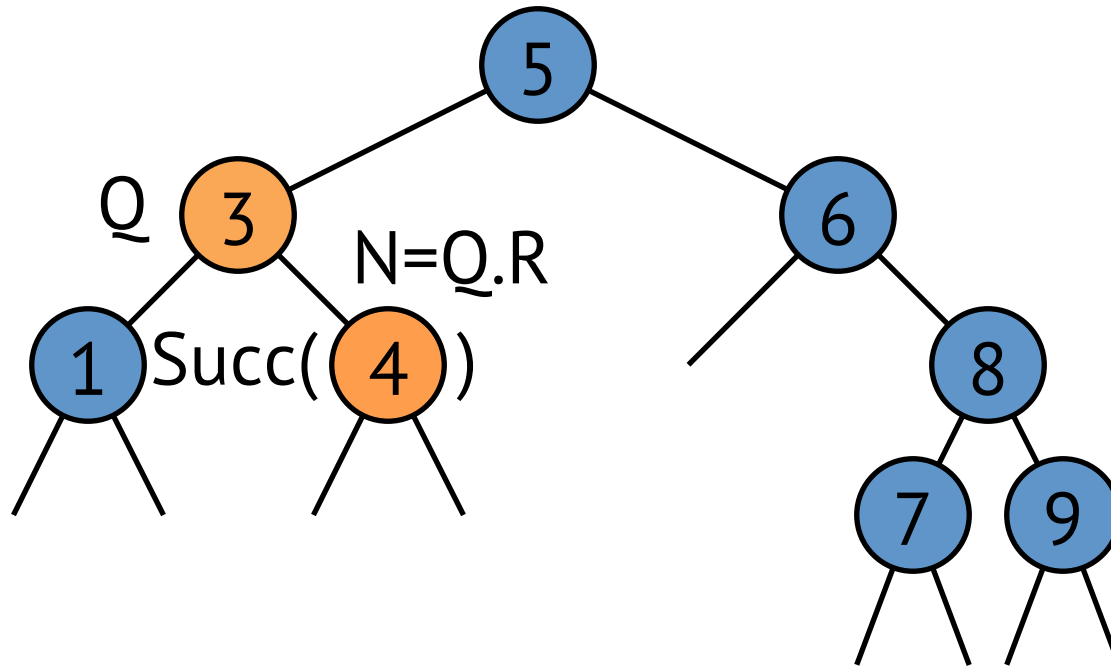
$$\text{Succ}(6) = 7$$



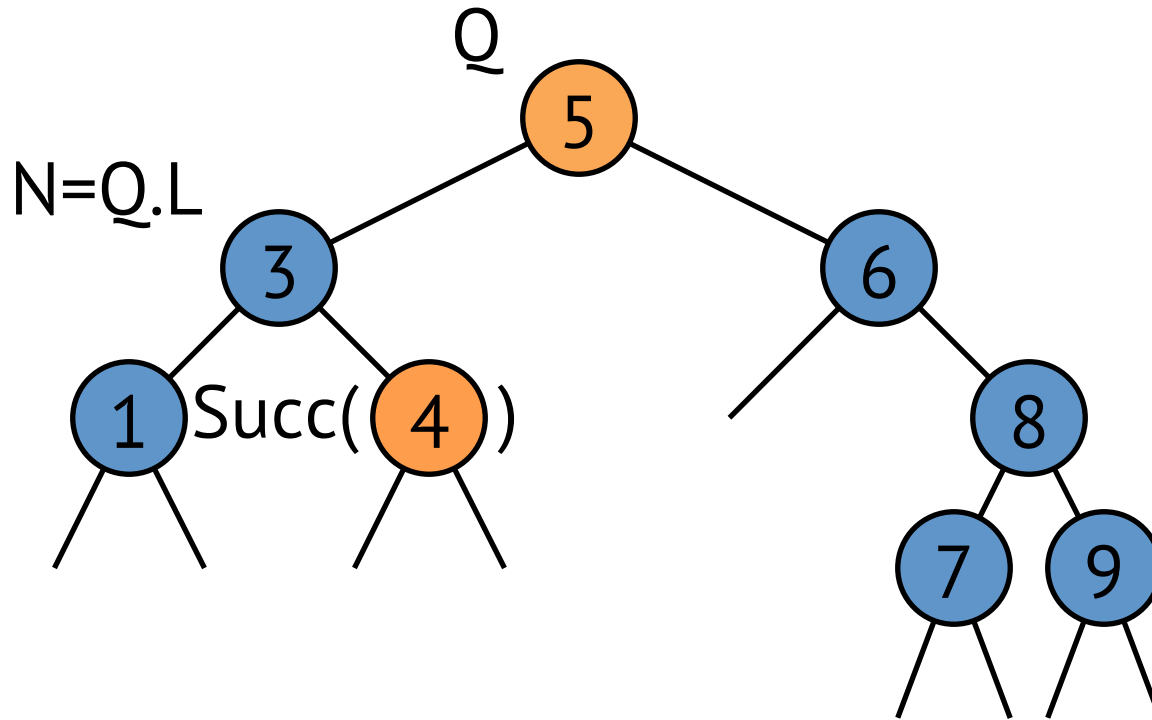
Successor()



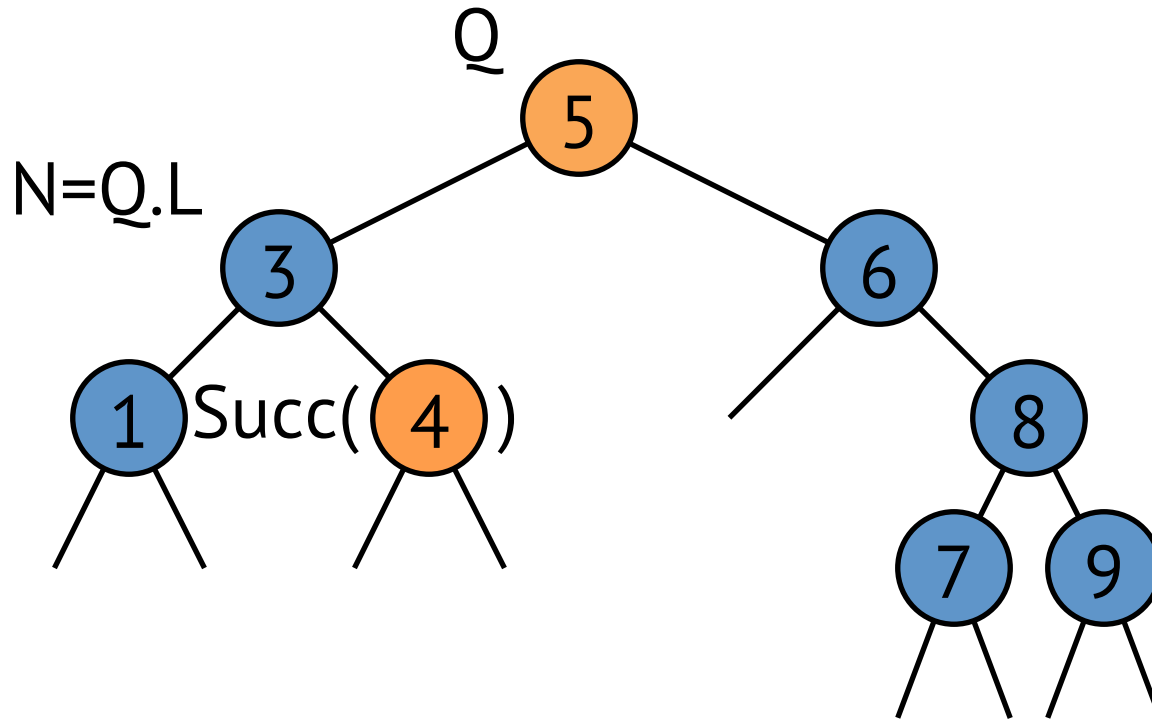
Successor()



Successor()



Successor()

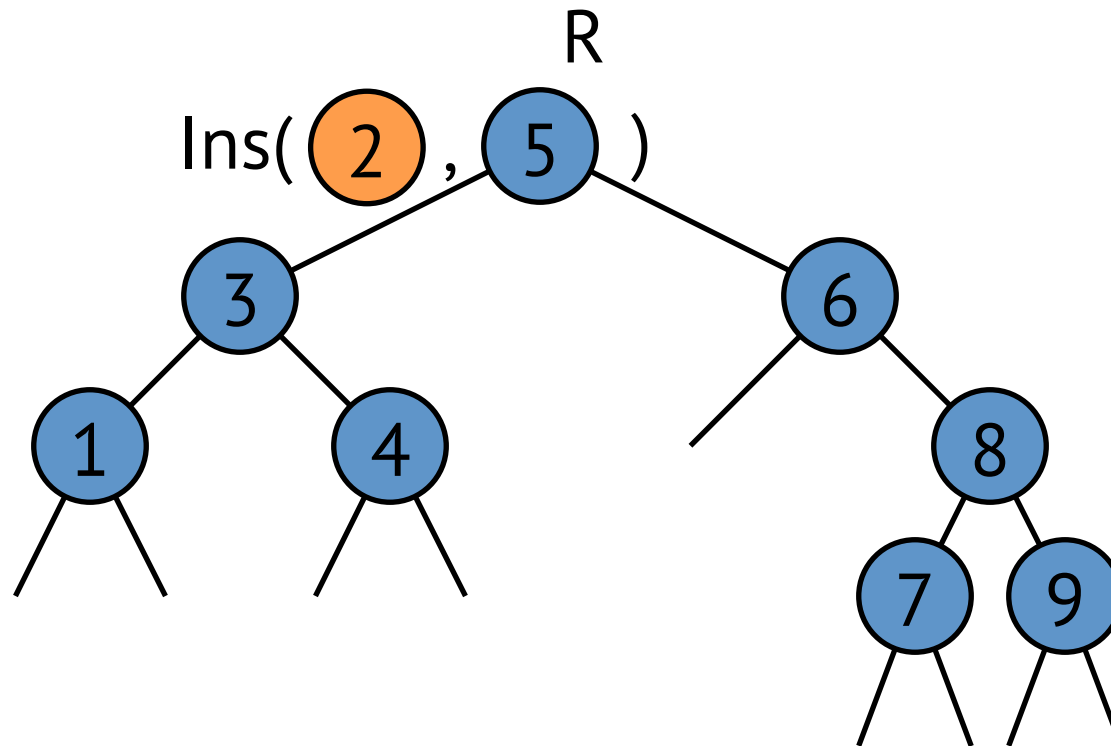


$$Succ(4) = 5$$

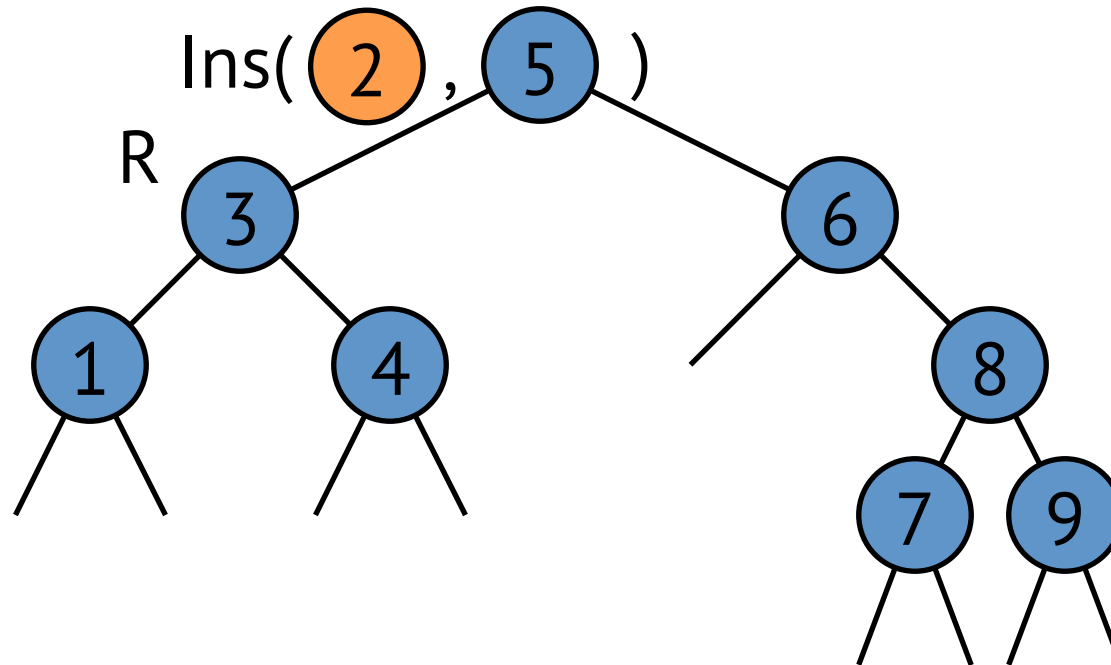
Insert()

- Insert(N,R)
Q ← NIL
while R ≠ NIL do
 Q ← R
 if N.X < R.X then R ← R.L
 else R ← R.R
N.P ← Q; N.L ← NIL; N.R ← NIL;
if Q = NIL then
 root ← N
else
 if N.X < Q.X then Q.L ← N
 else Q.R ← N

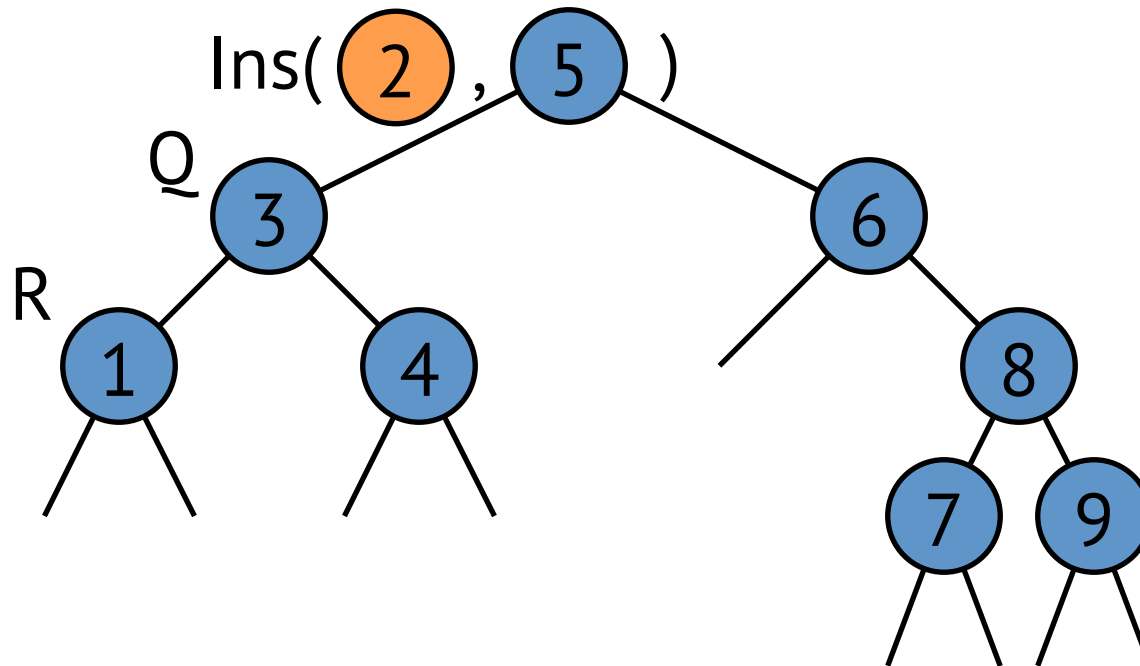
Insert()



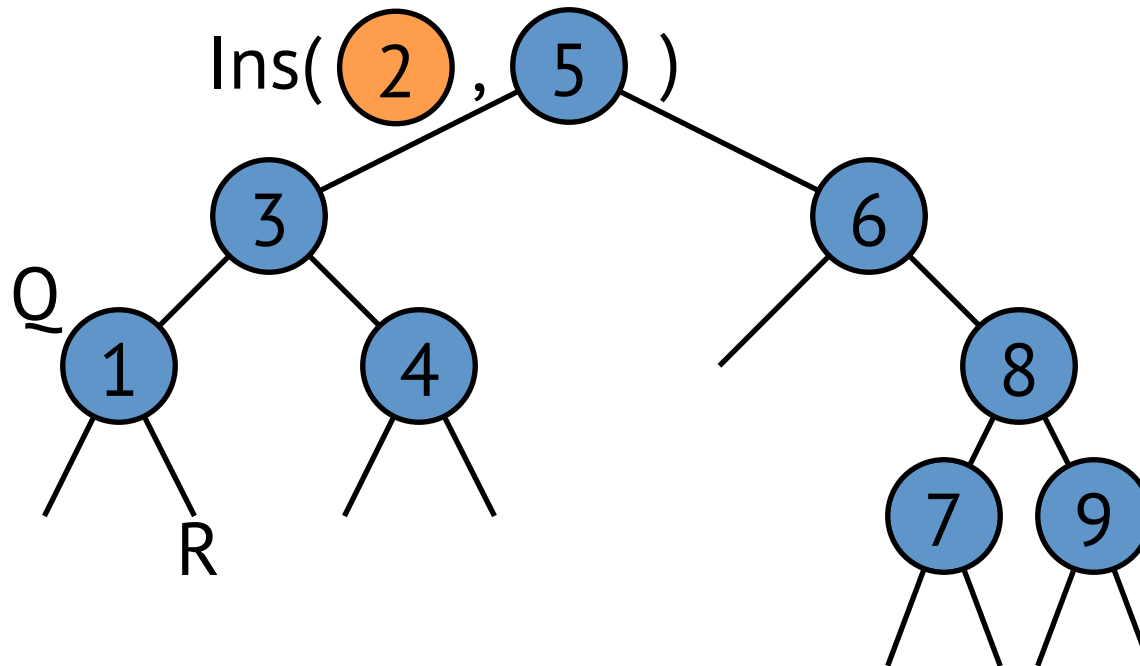
Insert()



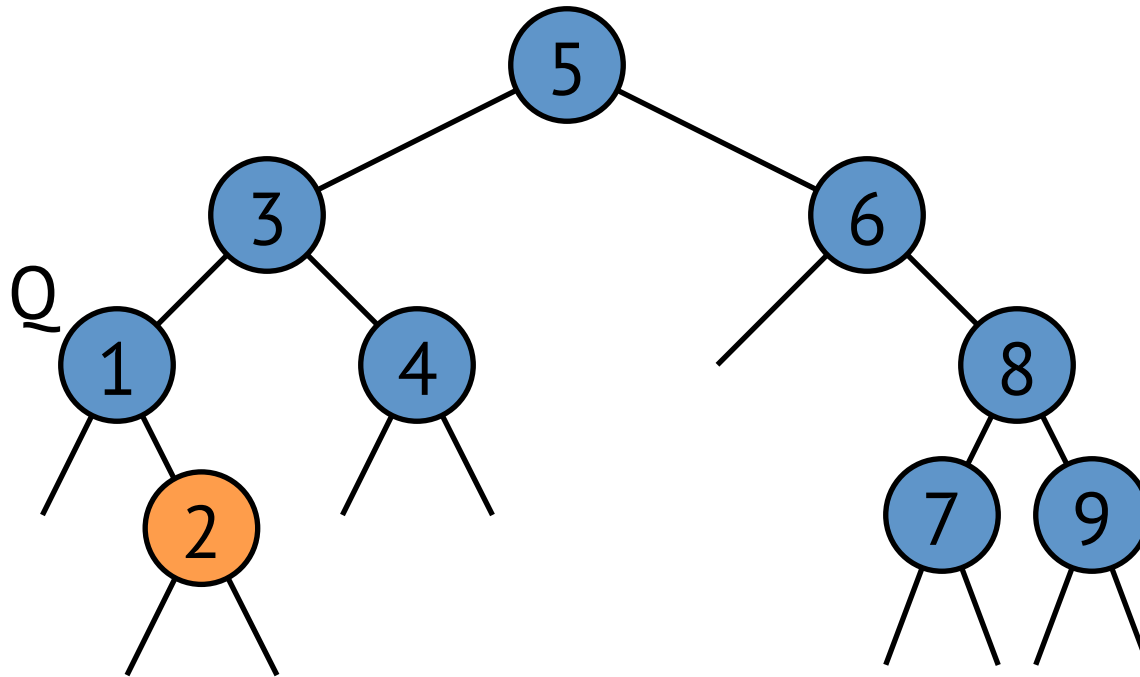
Insert()



Insert()



Insert()

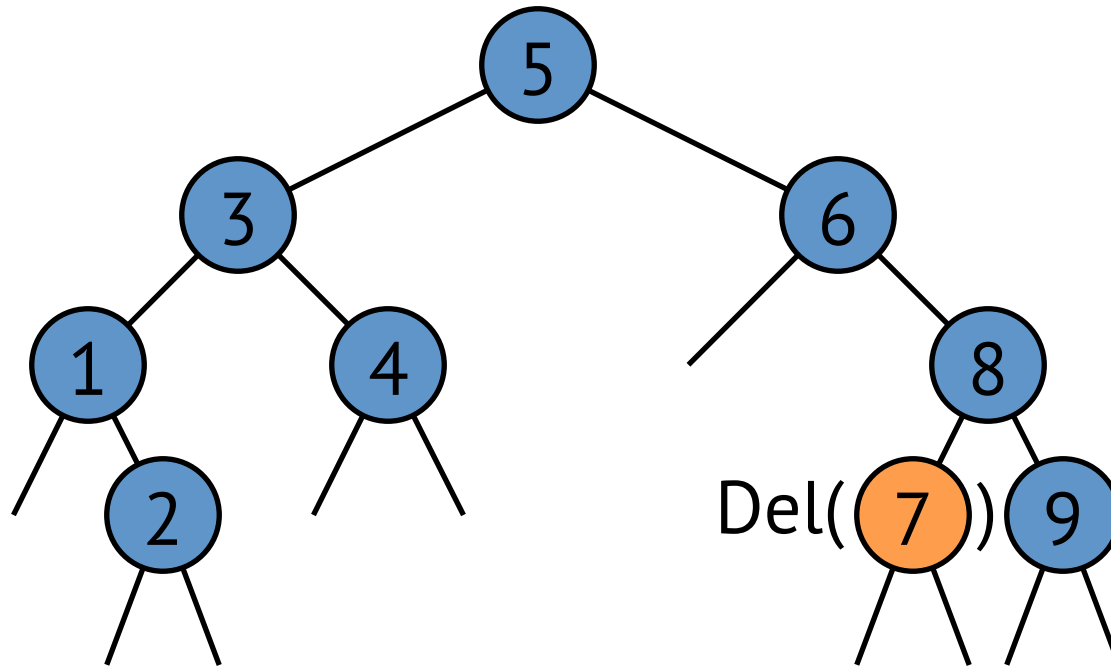


Delete()

- Delete(N,R)
- Fallunterscheidung
 1. N hat keine Kinder (Blatt)
 2. N hat ein Kind
 3. N hat zwei Kinder

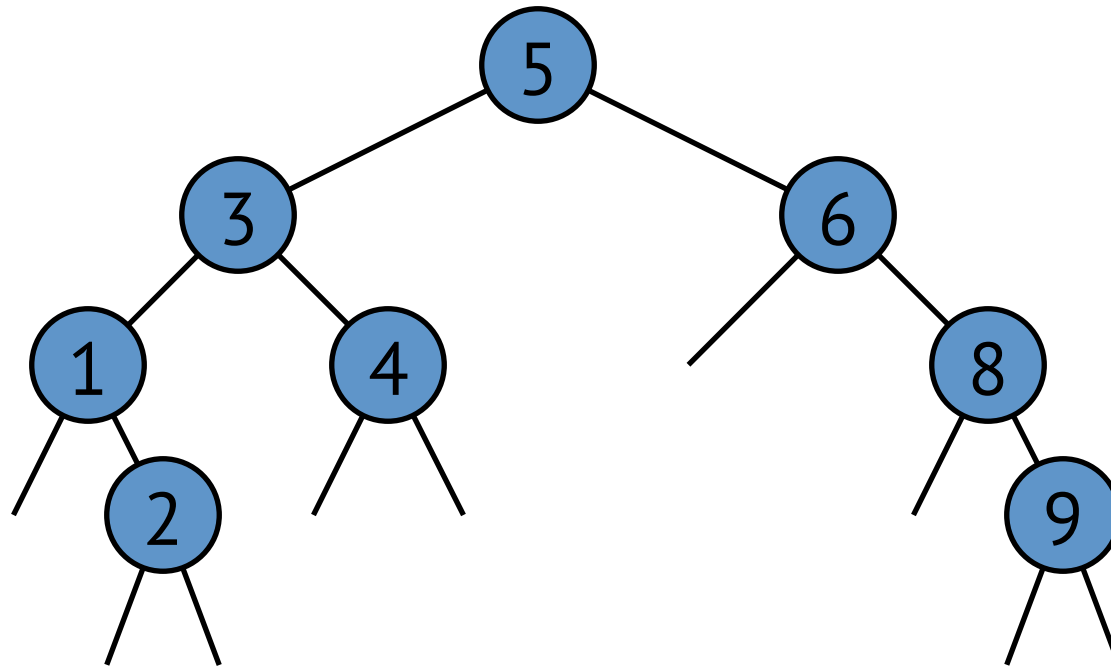


Delete()

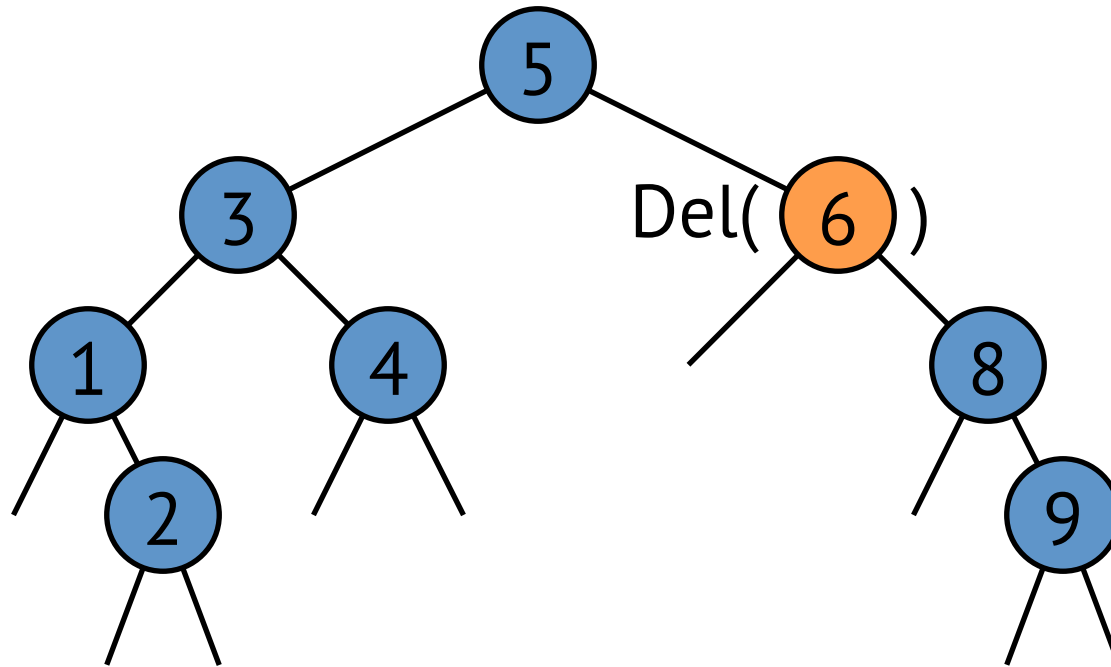


Fall 1: **7** hat keinen Sohn

Delete()

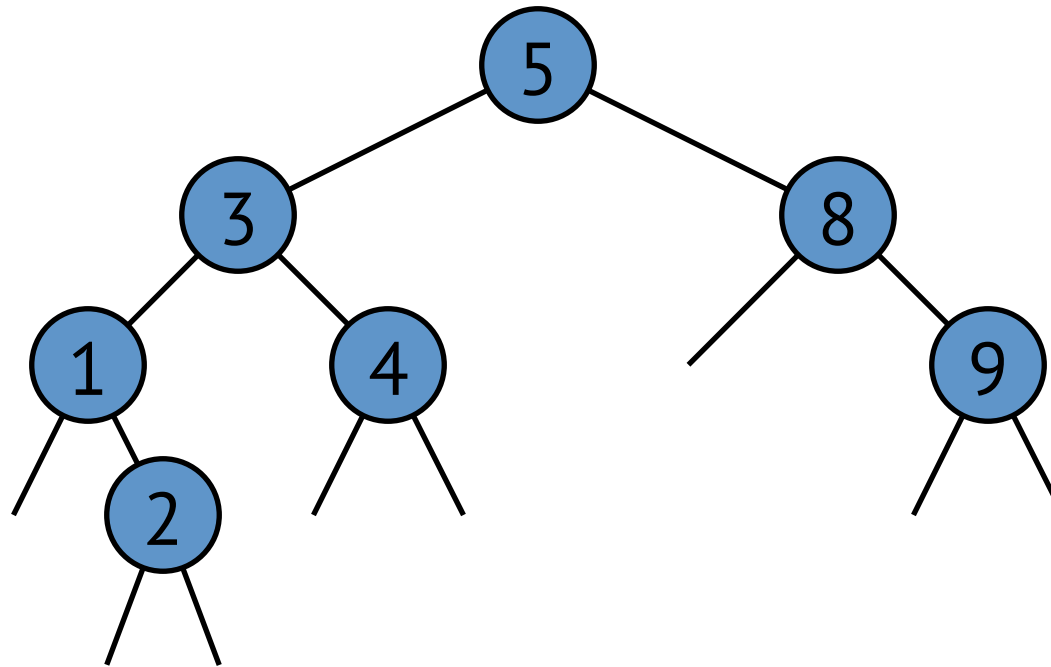


Delete()

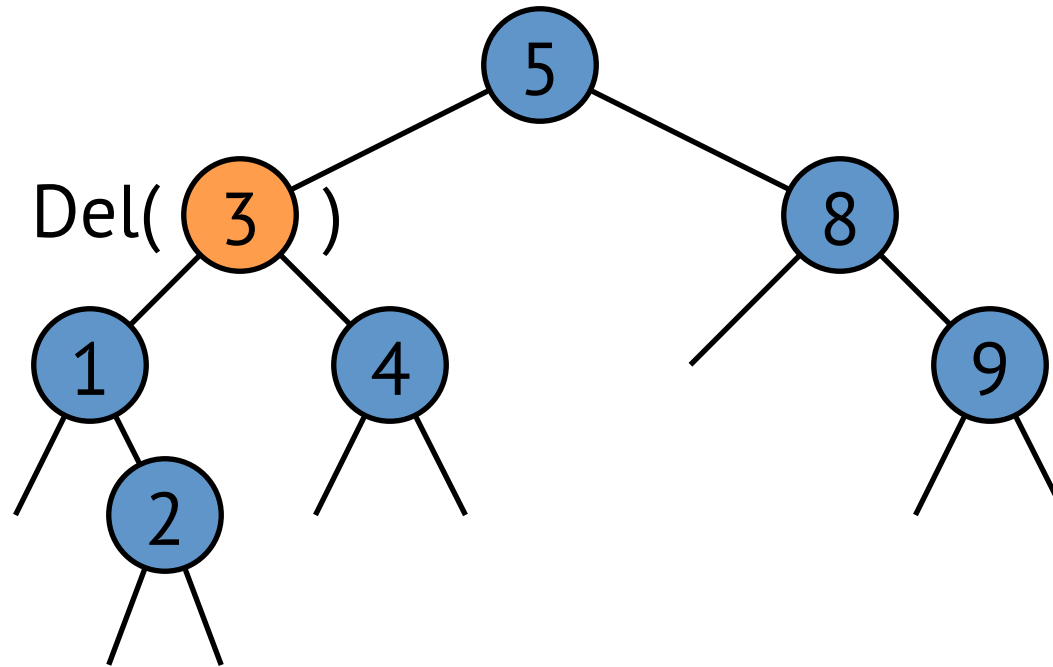


Fall 2: **6** hat genau einen Sohn

Delete()

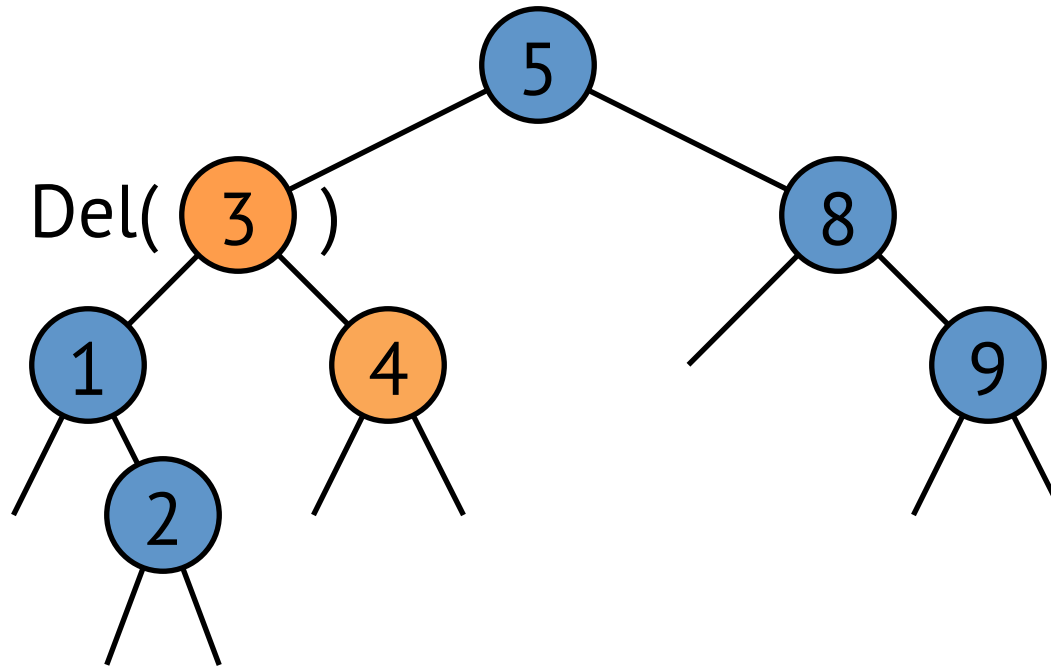


Delete()



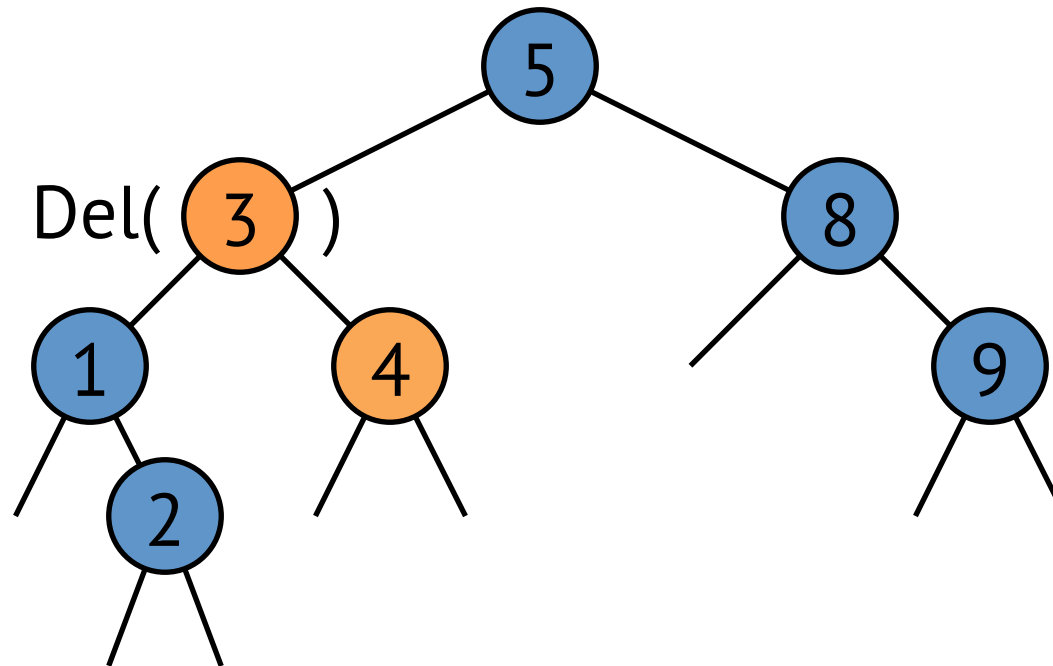
Fall 3: **3** hat zwei Söhne

Delete()



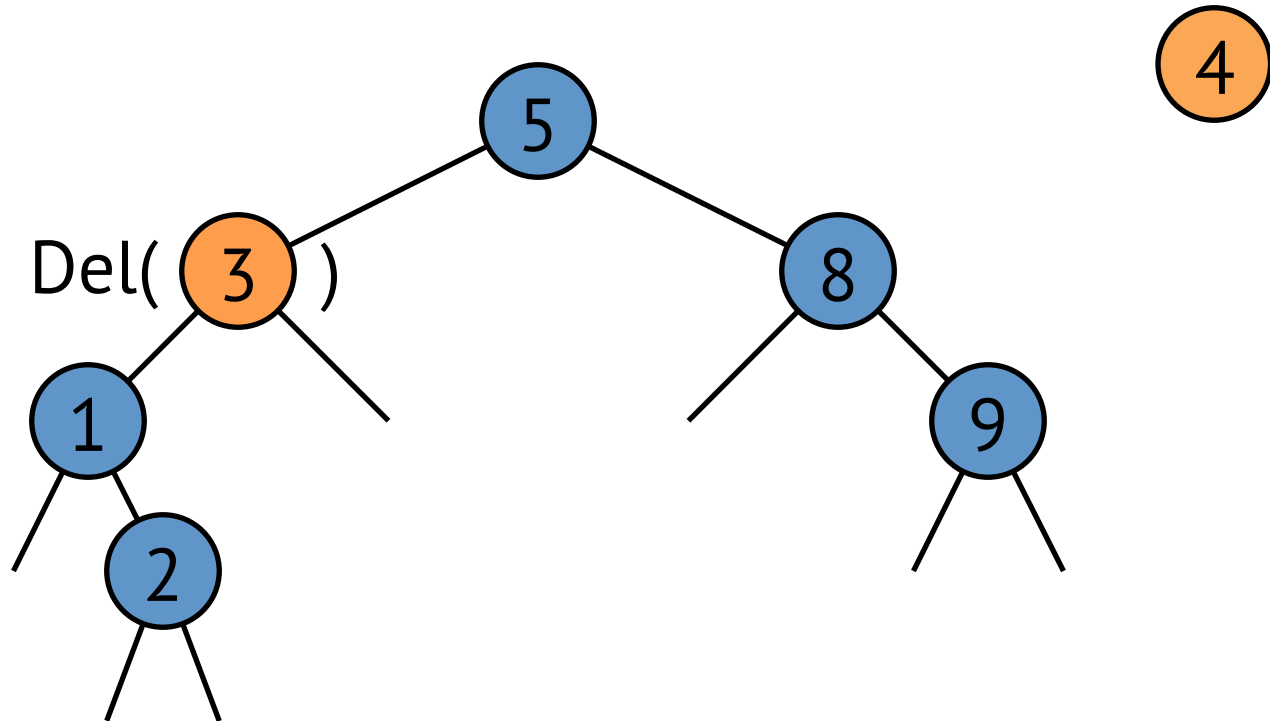
$\text{Depth}(\text{Successor}(3)) > \text{Depth}(3)$

Delete()



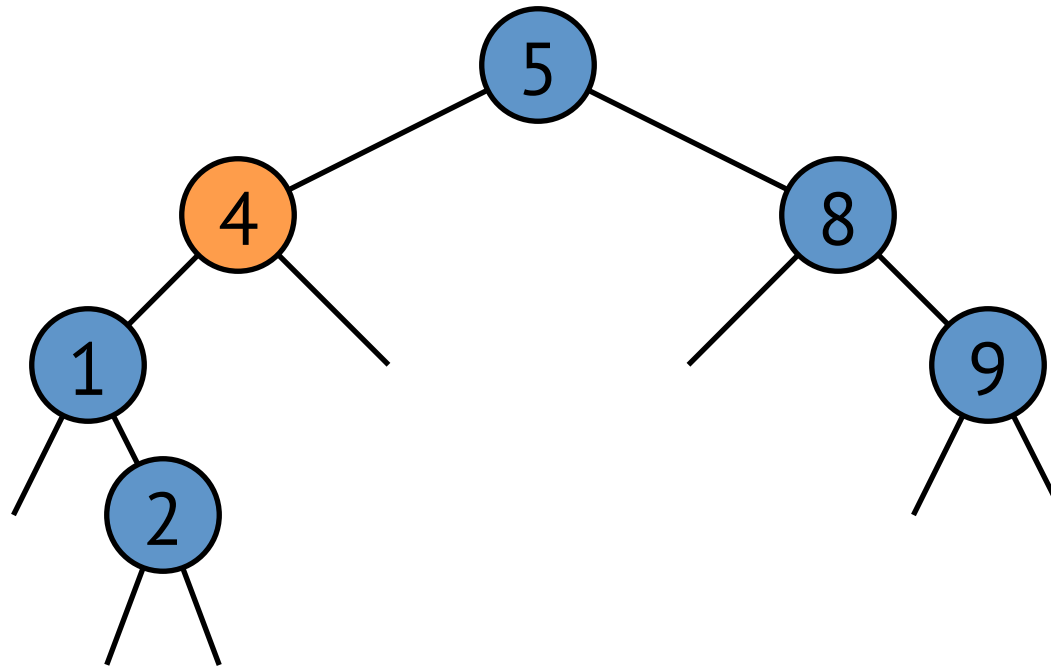
Successor(3) hat maximal ein Kind

Delete()



Kopiere Daten von 4 nach 3

Delete()



Kopiere Daten von **4** nach **3**

Delete()

- Delete(N,R)
 - if N.L = NIL or N.R = NIL then $Q \leftarrow N$
 - else $Q \leftarrow \text{Succ}(N)$
 - if Q.L \neq NIL then $S \leftarrow Q.L$
 - else $S \leftarrow Q.R$
 - if S \neq NIL then S.P \leftarrow Q.P
 - if Q.P = NIL then
root \leftarrow S
 - else
 - if Q = Q.P.L then $Q.P.L \leftarrow S$
 - else $Q.P.R \leftarrow S$
 - if Q \neq N then N.X \leftarrow Q.X // copy data
 - return Q

- Alle Operationen
 - Search(X,R)
 - Min(N), Max(N)
 - Successor(N), Predecessor(N)
 - Insert(N,R)
 - Delete(N,R)

bearbeiten genau einen Pfad im Baum

- Aufwand = $O(\text{erwartete Pfadlänge})$

- Erwartete Höhe eines Suchbaums mit n Knoten $N_1 \dots N_n$
 - Alle Schlüssel verschieden
 - $N_i.X < N_{i+1}.X$
- Hängt von der Einfügereihenfolge ab
- Mittelwert über alle Permutationen

Aufwand

- Höhe eines binären Suchbaums mit n Knoten: H_n
- Exponentielle Höhe: $X_n = 2^{H_n}$
- Sei N_i der Wurzelknoten, dann bilden $N_1 \dots N_{i-1}$ und $N_{i+1} \dots N_n$ die Teilbäume
- Also

$$H_n = 1 + \max\{ H_{i-1}, H_{n-i} \}$$

$$X_n = 2 \times \max\{ X_{i-1}, X_{n-i} \}$$

Aufwand

- Erwartungswert ... Mittelwert über alle möglichen Indizes i des Wurzelknotens

$$\begin{aligned} E(X_n) &= \frac{2}{n} \sum_{i=1}^n E(\max\{X_{i-1}, X_{n-i}\}) \\ &\leq \frac{2}{n} \sum_{i=1}^n (E(X_{i-1}) + E(X_{n-i})) \\ &= \frac{4}{n} \sum_{i=0}^{n-1} E(X_i) \end{aligned}$$

Induktion, s. Cormen :

$$= \frac{1}{4} \binom{n+3}{3} = O(n^3)$$

Aufwand

- Erwartungswert ... Mittelwert über alle möglichen Indizes i des Wurzelknotens

$$2^{E(H_n)} \leq E(2^{H_n}) = E(X_n) \leq cn^3$$

$$E(H_n) = \log(cn^3) = 3 \log n + \log c = O(\log n)$$

Aufwand

- $E(H_n) = O(\log n)$ bedeutet, dass die Pfadlängen in einem Suchbaum mit n Knoten im erwarteten Fall $O(\log n)$ sind.
- Alle Operationen `Insert()`, `Delete()`, ... haben einen erwarteten Aufwand von $O(\log n)$
- Aber Worst-Case Aufwand $O(n)$!!!

