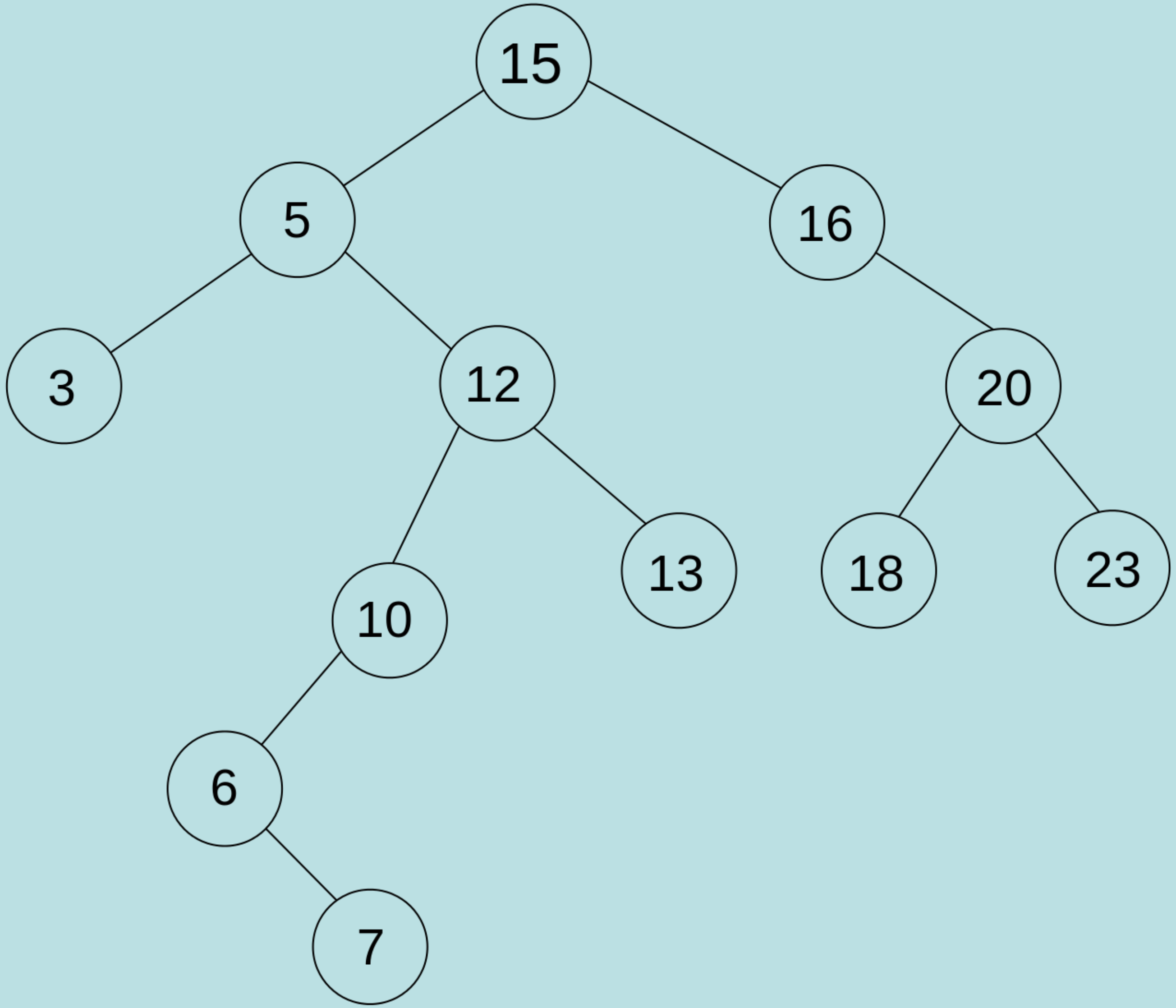
04 Bäume - Teil 2

Algorithmen und Datenstrukturen 2

Nachbesprechung binäre Suchbäume



toString

```
@Override
public String toString() {
  StringBuilder sb = new StringBuilder();
  createExpressionInorder(root, sb);
  return sb.toString();
}
private void createExpressionInorder(Node <K,
E> node, StringBuilder sb) {
  if (node != null) {
     sb.append("[");
     createExpressionInorder(node.left, sb);
     sb.append(node.key);
     createExpressionInorder(node.right, sb);
     sb.append("]");
} }
```

- Das Beispiel folgt einer Inorder Traversierung
  - Inorder: [[[1] 2 [3]] 5 [9]]
- Auch Inorder und Postorder lassen sich auf diese Weise darstellen
  - Preorder: [5 [2 [1] [3]] [9]]
  - Postorder: [[[1] [3] 2] [9] 5]

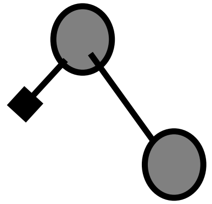
remove(key)

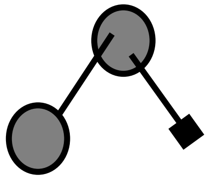
```java
public void remove(K key) {
  root = remove(root, key);
}

private Node<K, E> remove(Node<K,E> node, K key){
  if (node == null) { return null; }
  else {
     int c = key.compareTo(node.key);
     if (c < 0) { node.left = remove(node.left, key); }
     else if (c > 0) {node.right = remove(node.right, key); }
     else { // node.key == key
        ...
     }
     return node;
  }
}
```

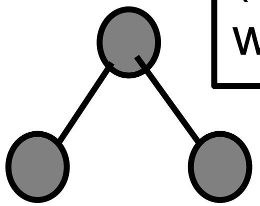
remove(key)

```java
else { // node.key == key
   if (node.left == null) {
      node = node.right;
      nodeCount--;
   }
   else if (node.right == null) {
      node = node.left;
      nodeCount--;
   }
   else {
      Node<K, E> succ = symSucc(node.right);
      succ.right = remove(node.right, succ.key);
      succ.left = node.left;
      node = succ;
   }
}
```





in diesem Fall muss der symmetrische Nachfolger (symSucc) gefunden werden.

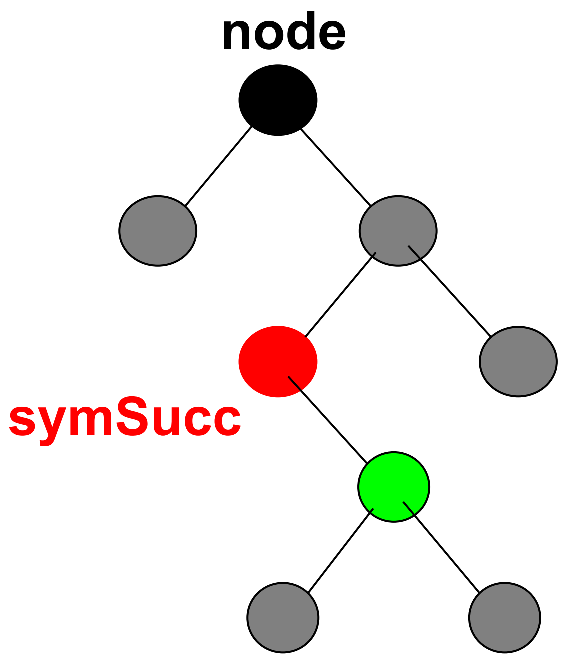


Fachhochschule
Nordwestschweiz

remove(key)

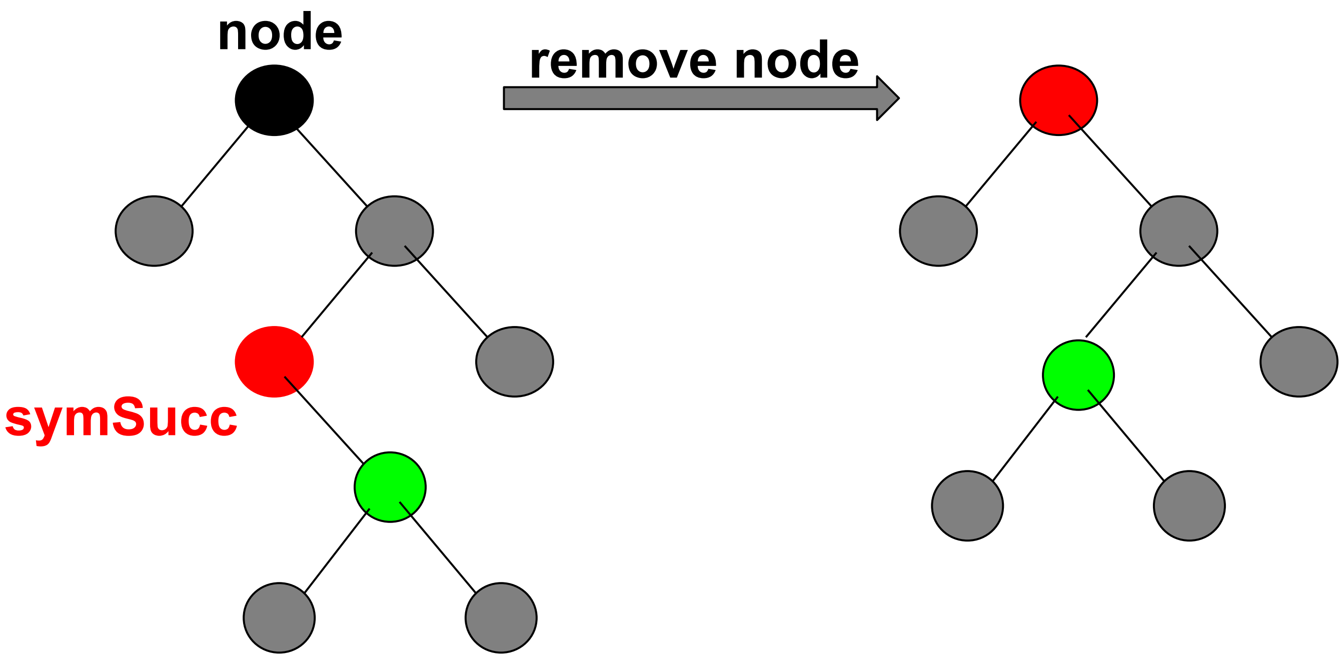
symSucc ist das “linkeste Element im rechten Teilbaum”

```
private Node<K, E> symSucc(Node<K,E> node){
  Node<K, E> succ = node;
  while (succ.left != null) {
     succ = succ.left;
  }
  return succ;
}
```



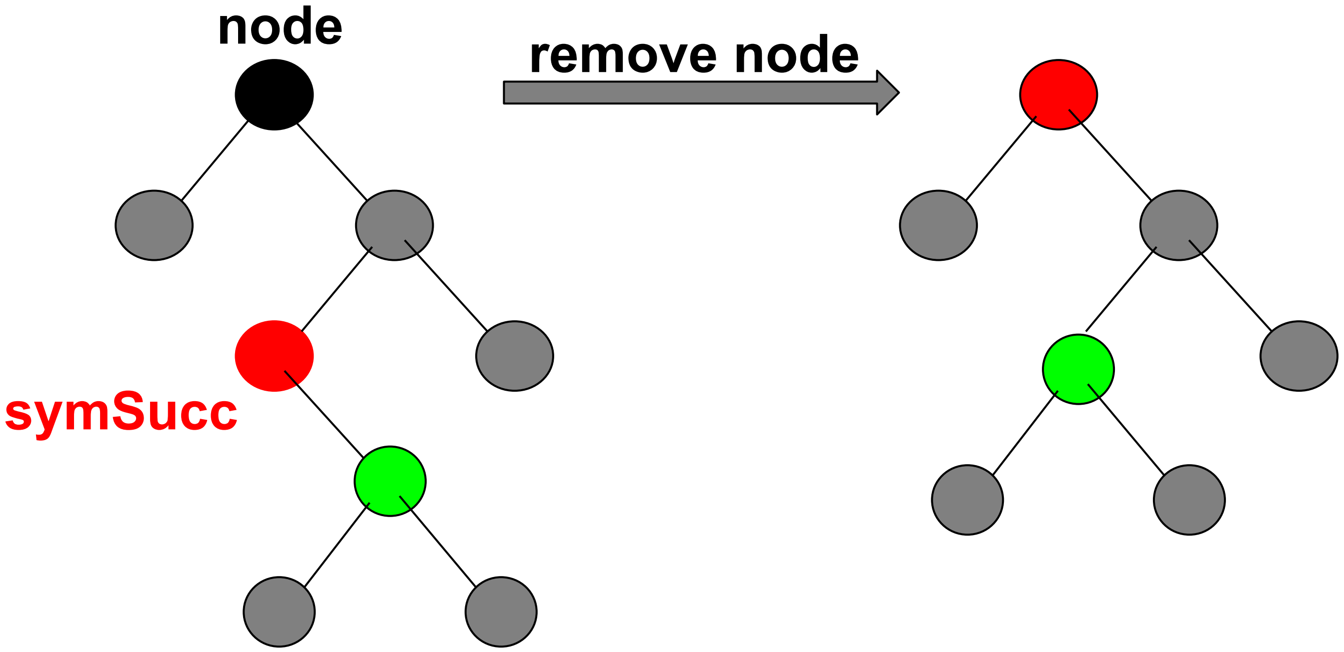
remove(key)

1. **node** durch **symSucc** ersetzen
2. **symSucc** im rechten Teilbaum von **node** entfernen
3. der rechte Teilbaum von symSucc “rutscht nach oben”



Fachhochschule Nordwestschweiz

remove(key)



1. **node** durch **symSucc** ersetzen

2. **symSucc** im rechten Teilbaum von **node** entfernen

3. der rechte Teilbaum von symSucc “rutscht nach oben”

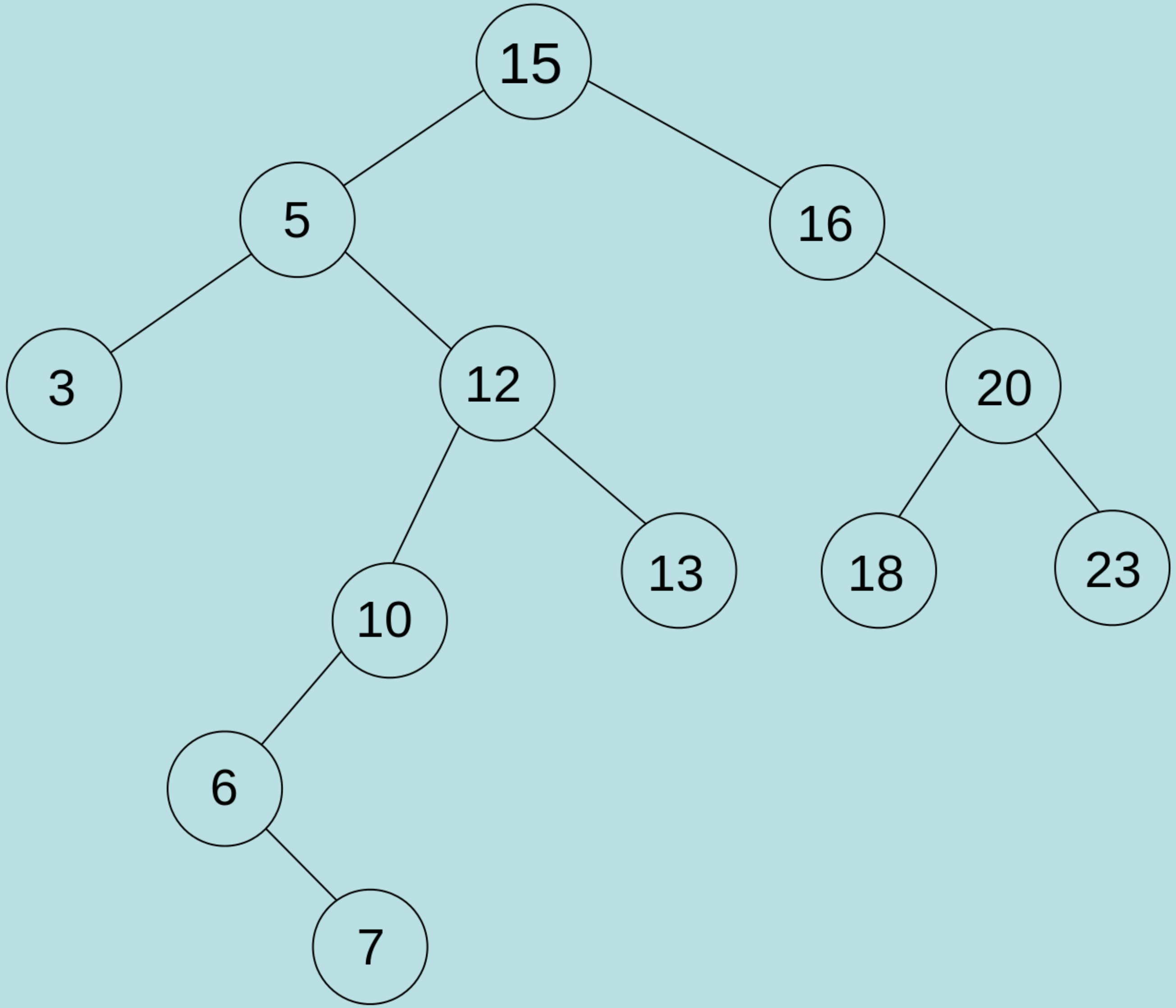
Implementierung:
remove symSucc.key rekursiv im rechten Teilbaum von node aufzurufen

Achtung:
key ist eine **final Variable**, deswegen **nicht** *node.key = symSucc.key* .

```java
else {
	Node<K, E> succ = symSucc(node.right);
	succ.right = remove(node.right, succ.key);
	succ.left = node.left;
	node = succ;
}
```

Fachhochschule Nordwestschweiz

Laufzeitanalyse von binären Suchbäumen

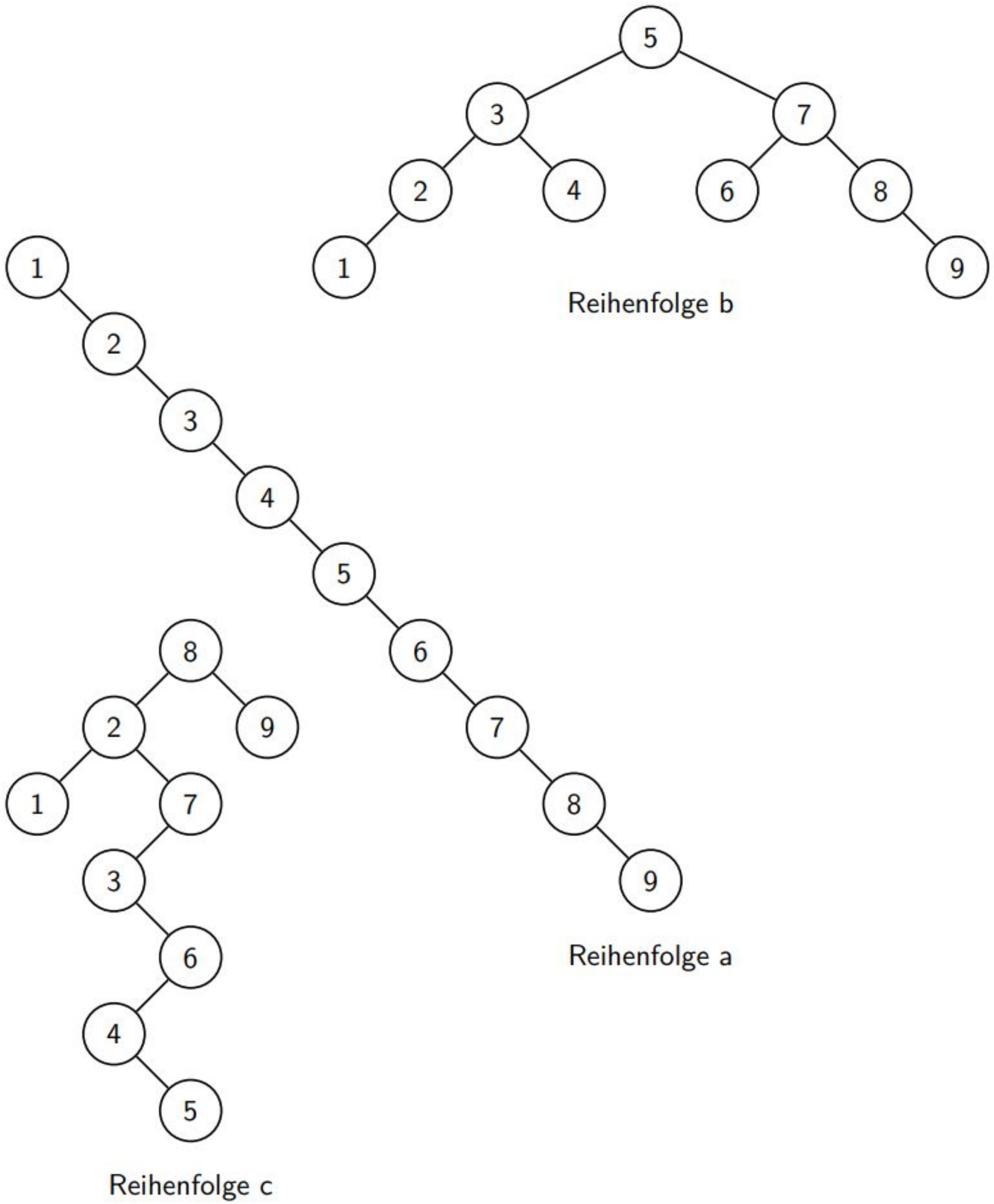


Binäre Suchbäume - Laufzeiten

Wir erinnern uns:

Die Einfügereihenfolge bestimmt die Struktur eines binären Suchbaumes.

- Reihenfolge a: 1, 2, 3, 4, 5, 6, 7, 8, 9
- Reihenfolge b: 5, 3, 7, 4, 6, 2, 8, 1, 9
- Reihenfolge c: 8, 2, 7, 3, 6, 4, 5, 9, 1



Binäre Suchbäume - Laufzeiten

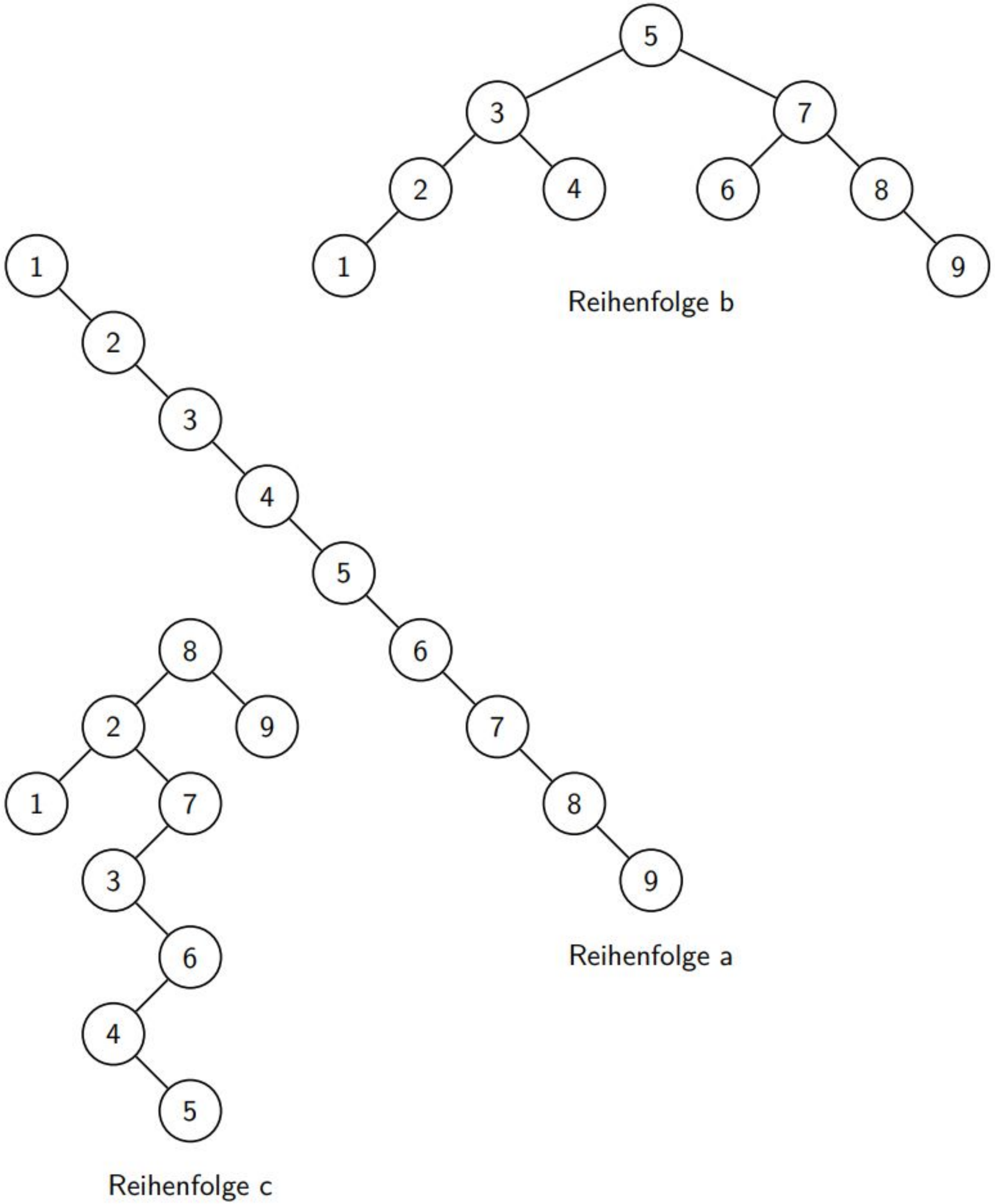
Wir erinnern uns:

Die Einfügereihenfolge bestimmt die Struktur eines binären Suchbaumes.

- Reihenfolge a: 1, 2, 3, 4, 5, 6, 7, 8, 9
- Reihenfolge b: 5, 3, 7, 4, 6, 2, 8, 1, 9
- Reihenfolge c: 8, 2, 7, 3, 6, 4, 5, 9, 1

Wie viele Knoten werden bei search(9) besucht?

Wie viele Knoten werden maximal bei einer Suche besucht?



Binäre Suchbäume - Laufzeiten

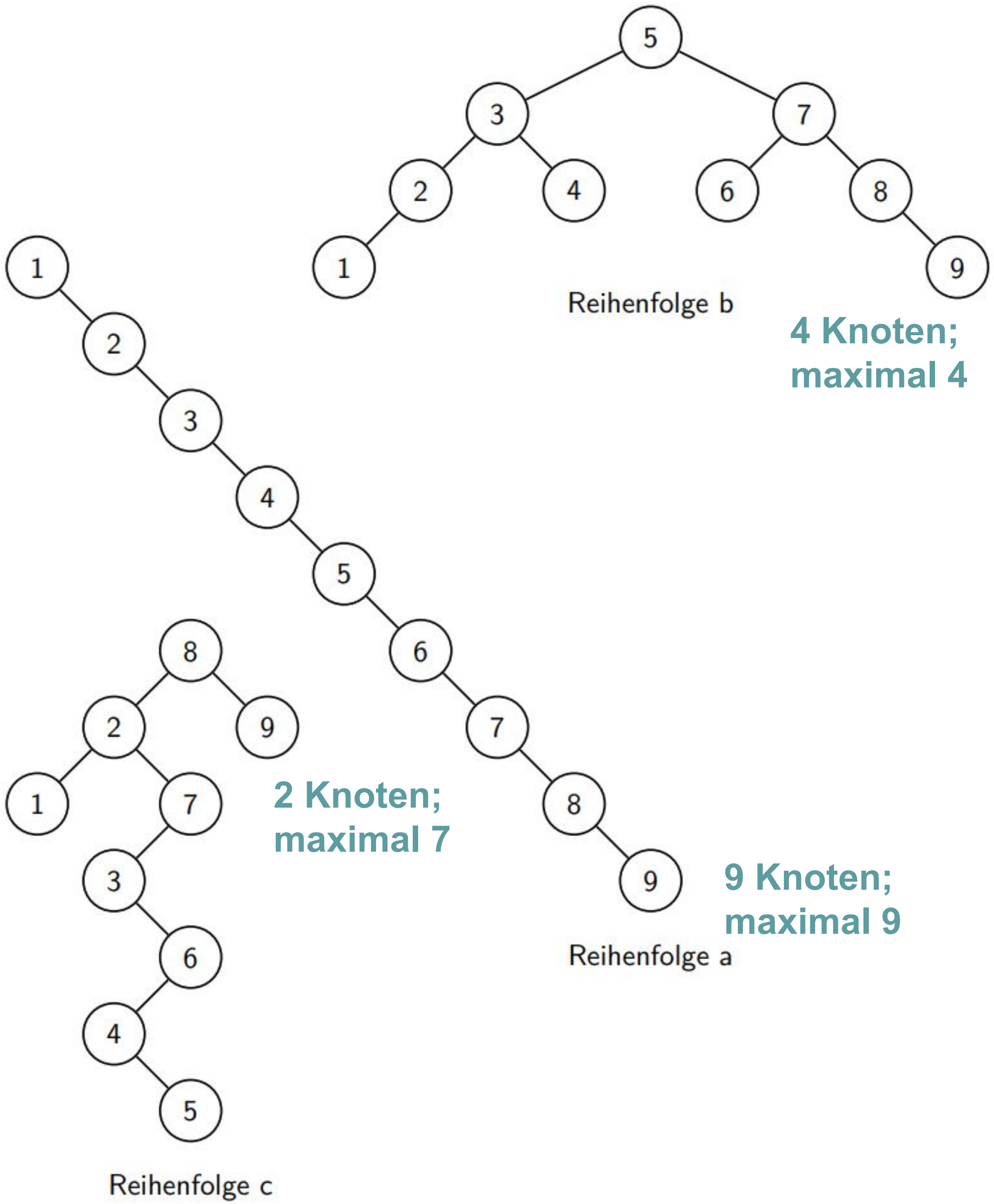
Wir erinnern uns:

Die Einfügereihenfolge bestimmt die Struktur eines binären Suchbaumes.

- Reihenfolge a: 1, 2, 3, 4, 5, 6, 7, 8, 9
- Reihenfolge b: 5, 3, 7, 4, 6, 2, 8, 1, 9
- Reihenfolge c: 8, 2, 7, 3, 6, 4, 5, 9, 1

Wie viele Knoten werden bei search(9) besucht?

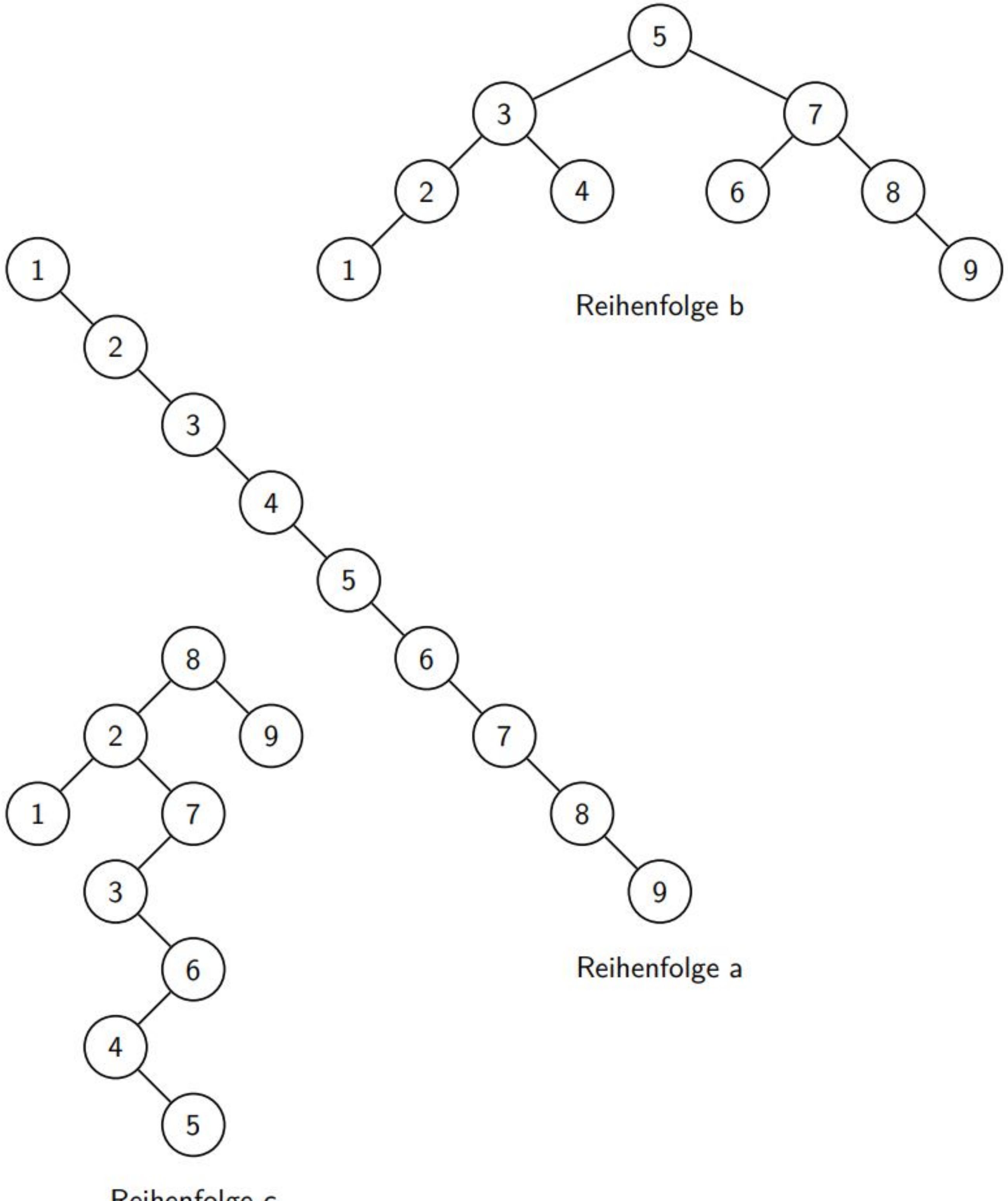
Wie viele Knoten werden maximal bei einer Suche besucht?



Binäre Suchbäume - Laufzeiten

Die **Höhe des Baumes** bestimmt die Laufzeitkomplexität der Operationen add, search, remove:

- worst case: O(n)
  Baum ist zu einer Liste entartet.
- best case: O(log n)
  Baum ist vollständig (ev. mit Ausnahme des untersten Niveaus).

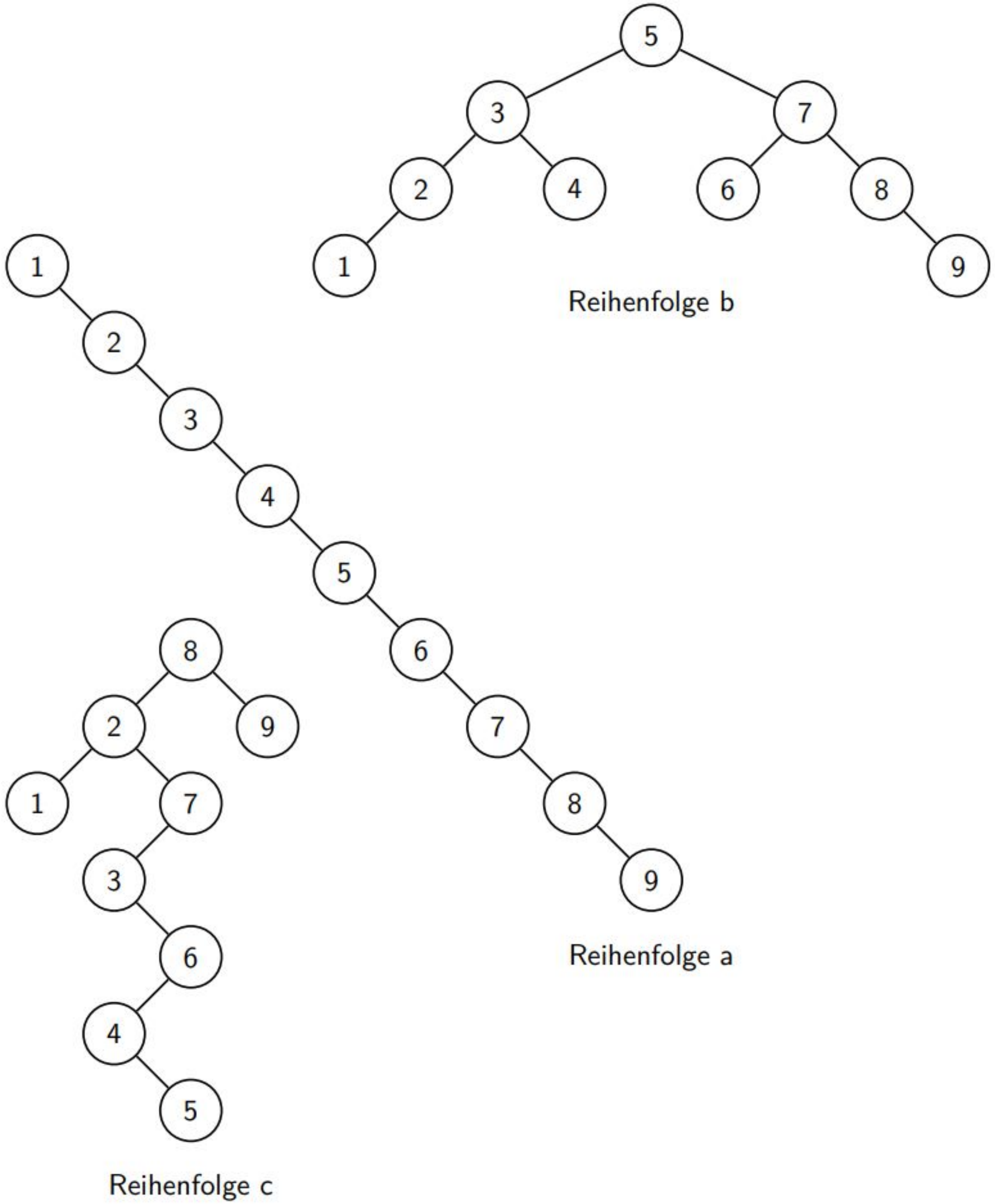


Binäre Suchbäume - Laufzeiten

Die **Höhe des Baumes** bestimmt die Laufzeitkomplexität der Operationen add, search, remove:

- worst case: O(n)
  Baum ist zu einer Liste entartet.
- best case: O(log n)
  Baum ist ausgeglichen (idealerweise vollständig mit Ausnahme des untersten Niveaus).

Ziel: Die Höhe möglichst klein und den Baum möglichst ausgeglichen halten.



Balancierte binäre Suchbäume

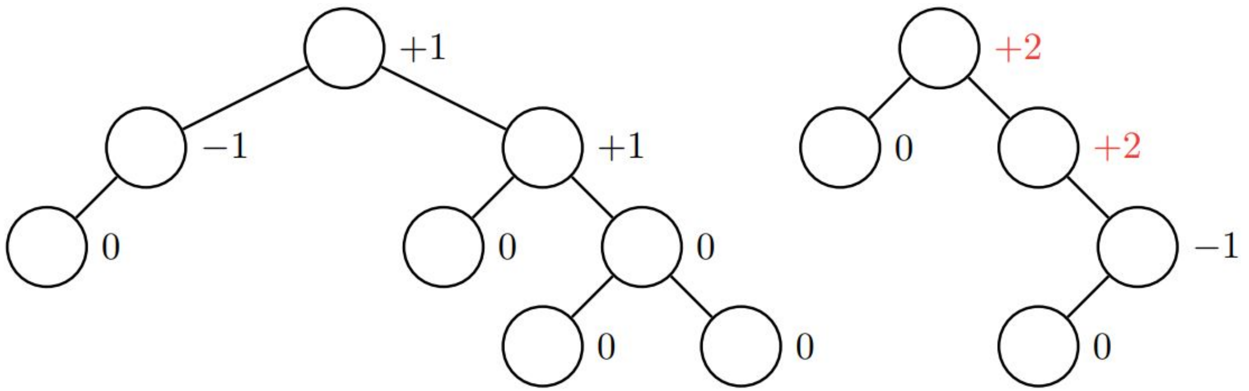
AVL-Bäume

Bäume - Lernziele

AVL-Bäume

- Sie können die Ausgleichsmechanismen von AVL-Bäume erklären und an Beispielen demonstrieren.
- Sie können von Hand im AVL-Baum Suchen, Einfügen und Löschen.
- Sie können das Einfügen in einen AVL-Baum in Java programmieren.

AVL-Bäume - Begriffe und Definitionen



links: balancierter AVL-Baum   rechts: kein ausgeglichener Baum

AVL-Baum	Ein binärer Suchbaum, bei dem sich die Höhen der Teilbäume eines Knotens um höchstens 1 unterscheiden.
Balancefaktor eines Knotens	Differenz zwischen der Höhe des rechten und linken Teilbaumes. *bal(v) = Höhe des rechtenTeilbaumes von v - Höhe des linken Teilbaumes von v*
Ausgeglichen / Balanciert	Ein Knoten v ist *ausgeglichen / balanciert*, wenn *bal(v) in {-1, 0, 1}*. Sonst ist v er *unausgeglichen*. Ein Baum ist *ausgeglichen / balanciert*, wenn alle seine Knoten ausgeglichen sind.

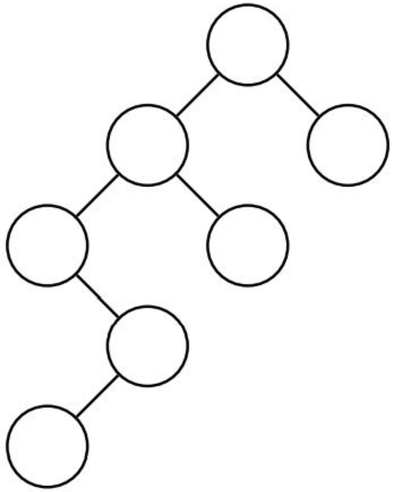
Fachhochschule
Nordwestschweiz

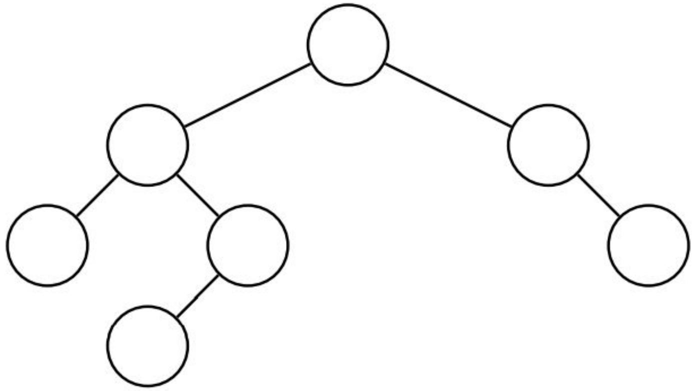
AVL-Bäume - Begriffe und Definitionen

Aufgabe

Bestimmen Sie die Balancefaktoren aller Knoten in den nebenstehenden Bäumen.

Ist einer der beiden Bäume ausgeglichen?



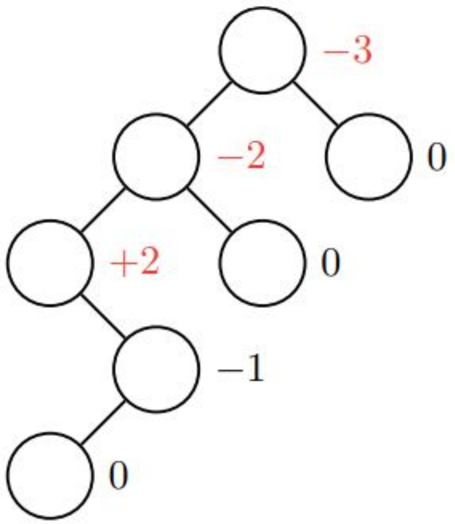


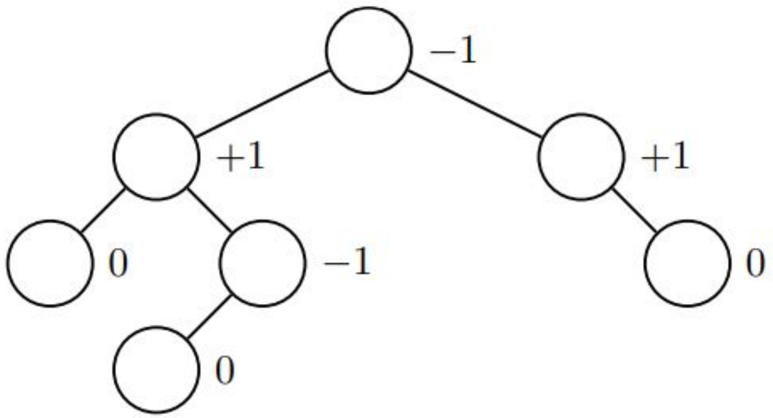
AVL-Bäume - Begriffe und Definitionen

Lösung

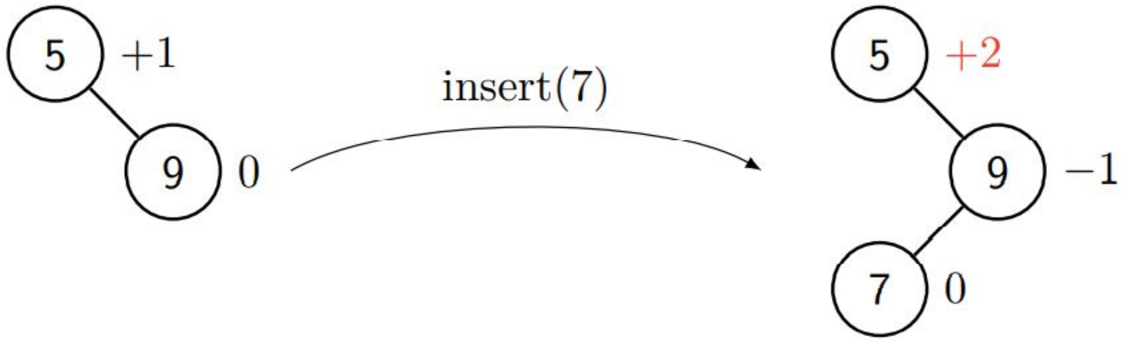
Bestimmen Sie die Balancefaktoren aller Knoten in den nebenstehenden Bäumen.

Ist einer der beiden Bäume ausgeglichen?





AVL-Bäume - Operationen



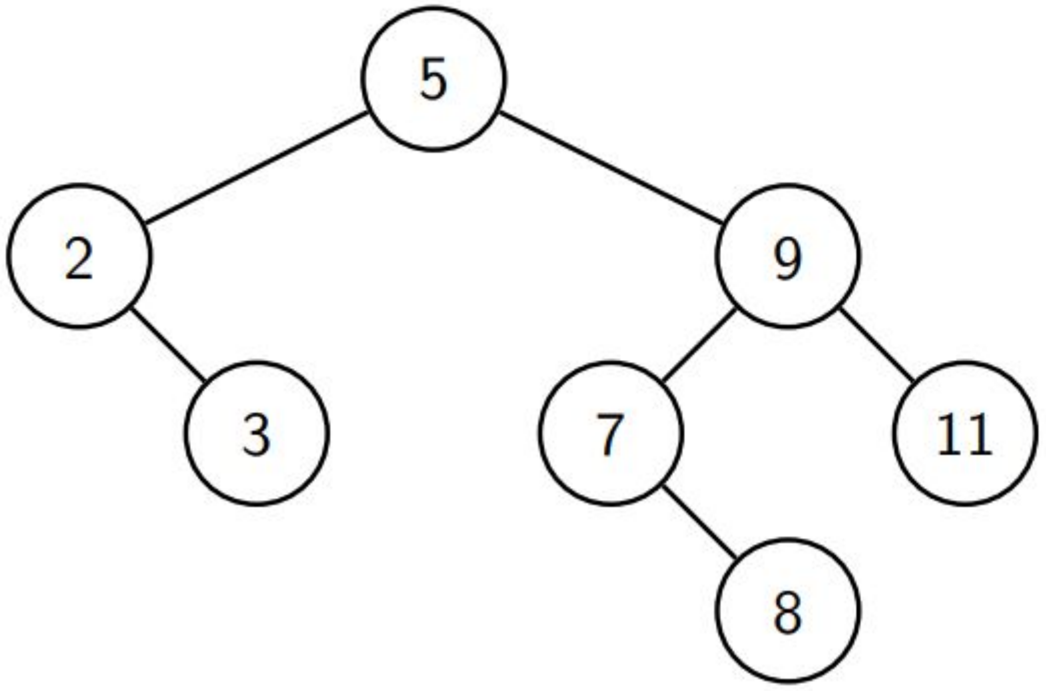
Einfügen, Suchen, Löschen im AVL-Baum

1. Gleich wie im binären Suchbaum
2. Nach Einfügen und Löschen:
    - Balancefaktoren neu berechnen auf dem Pfad vom eingefügten/zuunterst gelöschten Knoten bis zur Wurzel. (Die Teilbäume und damit die Balance der anderen Knoten bleiben unverändert.)
    - Umstrukturierung, sobald ein Knoten unausgeglichen ist.

AVL-Bäume - Begriffe und Definitionen

Aufgabe

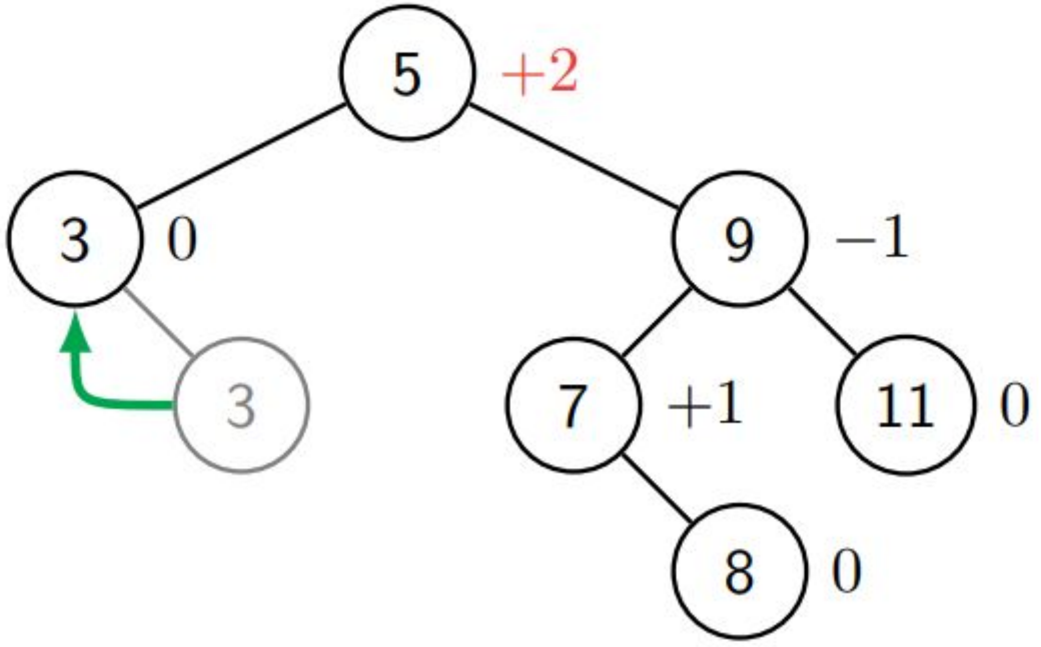
Entfernen Sie den Knoten mit Schlüssel 2 und berechnen Sie die neuen Balancefaktoren. Ist der resultierende Baum ausgeglichen?



AVL-Bäume - Begriffe und Definitionen

Lösung

Entfernen Sie den Knoten mit Schlüssel 2 und berechnen Sie die neuen Balancefaktoren. Ist der resultierende Baum ausgeglichen?



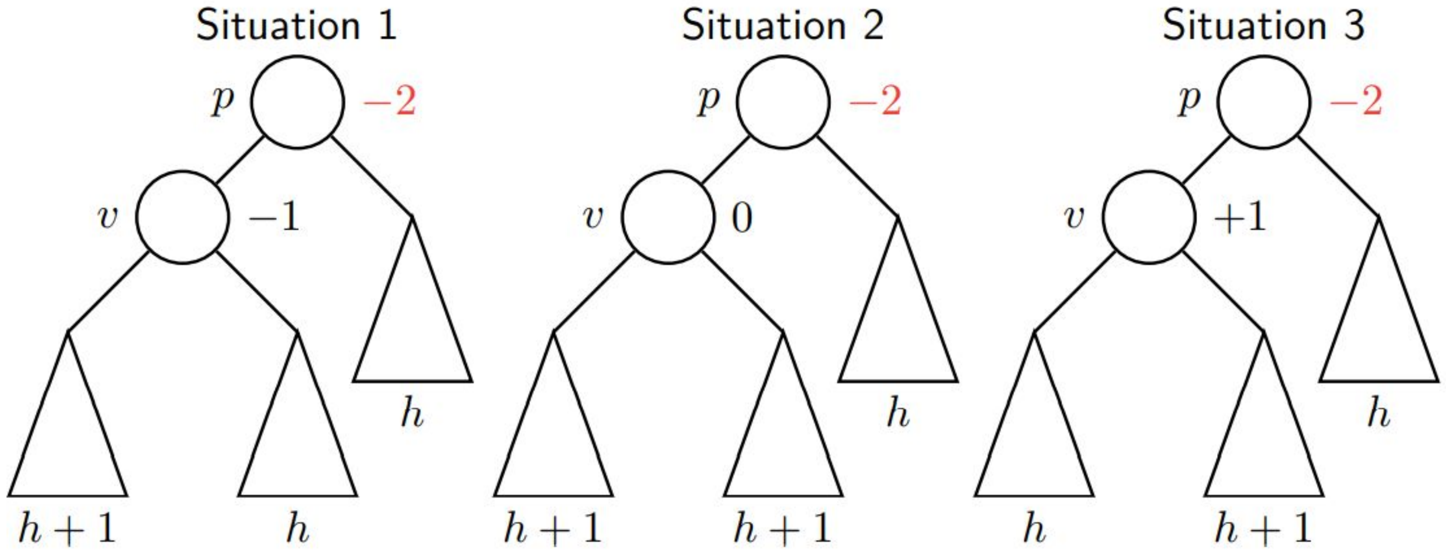
Der Baum ist nicht mehr ausgeglichen, weil die Wurzel nicht mehr ausgeglichen ist!

AVL-Bäume - Operationen

Rebalancierung	Eine Umstrukturierung, bei der ein unausgeglichener Baum wieder balanciert wird.
Baumrotation	Eine Methode zur Rebalancierung eines Suchbaumes, bei der die Eigenschaften (Struktur und Ordnung) erhalten bleiben.

https://studyflix.de/informatik/avl-baum-1434

AVL-Bäume - Operationen

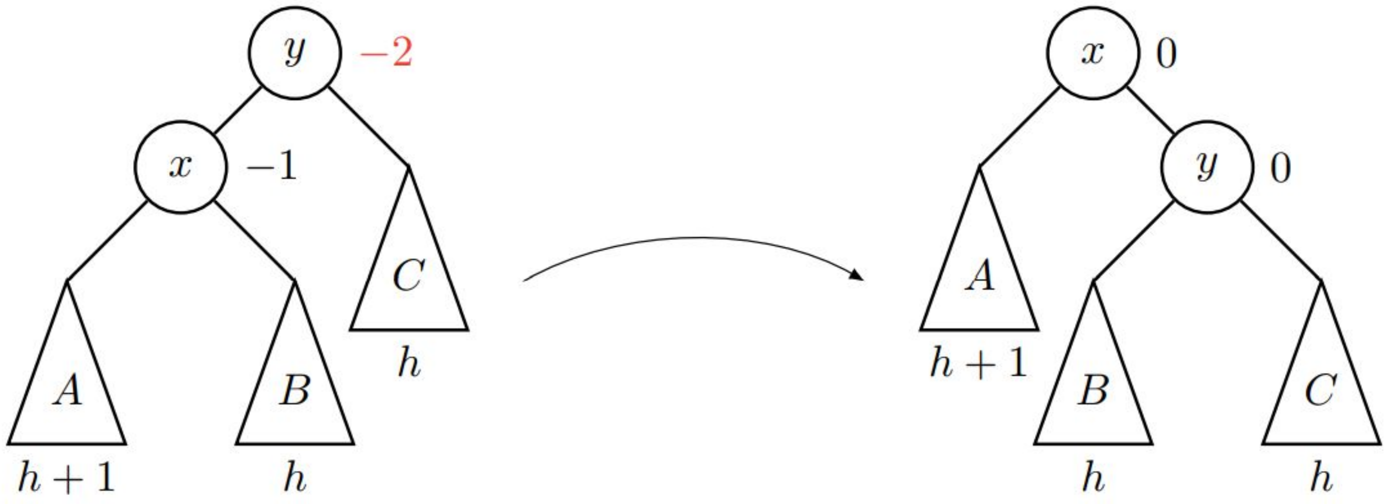


Es gibt 3 Situationen eines unausgeglichenen Baumes (Abbildungen plus 3 symmetrische)

1. bal(p) und bal(v) haben das gleiche Vorzeichen
2. bal(v) = 0
3. bal(p) und bal(v) haben unterschiedliche Vorzeichen

Je nach Situation werden unterschiedliche Rotationen ausgeführt.

AVL-Bäume - Operationen



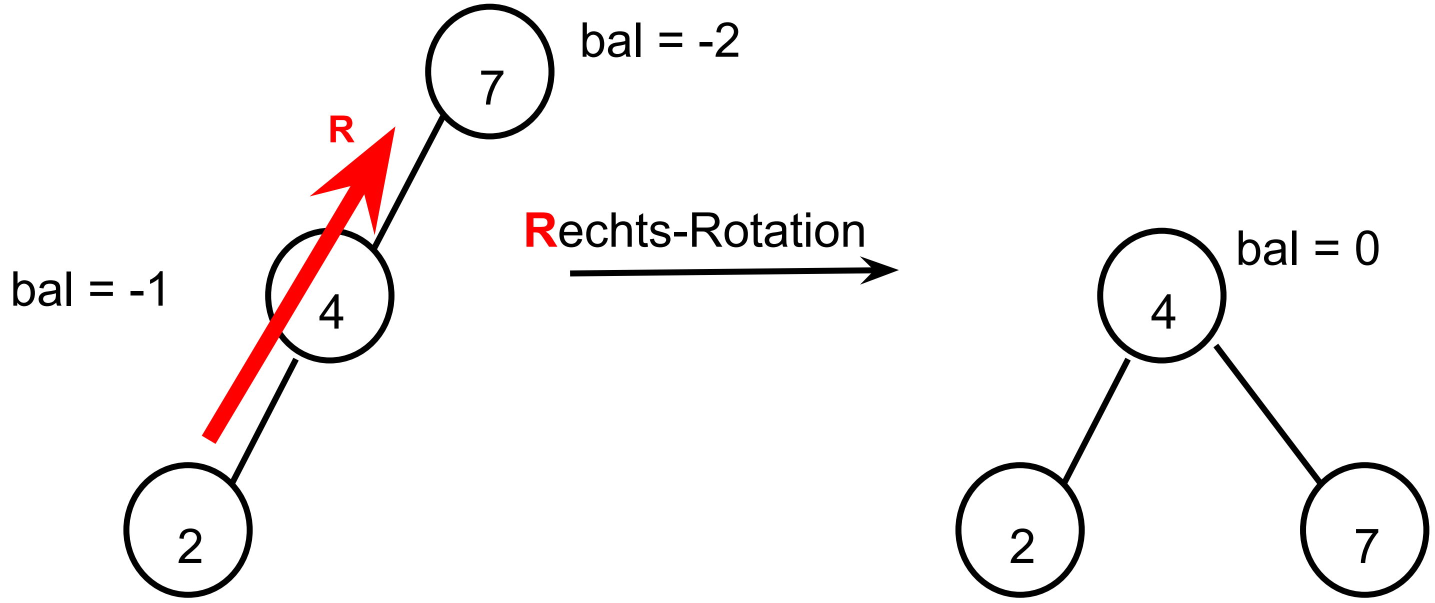
Situation 1 & 2 - Einfache Rotation

Die ***einfache Rotation nach rechts*** (siehe Abbildung) wird angewendet, wenn der linke Teilbaum eines Knotens im Vergleich zum rechten Teilbaum zu hoch ist. Durch eine Baumrotation nach rechts wird dieser Teilbaum angehoben und der rechte weniger hohe Teilbaum abgesenkt, um den unausgeglichenen Knoten wieder auszugleichen.

Die ***einfache Rotation nach links*** (symmetrisch zur Abbildung) wird angewendet, wenn der rechte Teilbaum eines Knotens im Vergleich zum linken Teilbaum zu hoch ist.

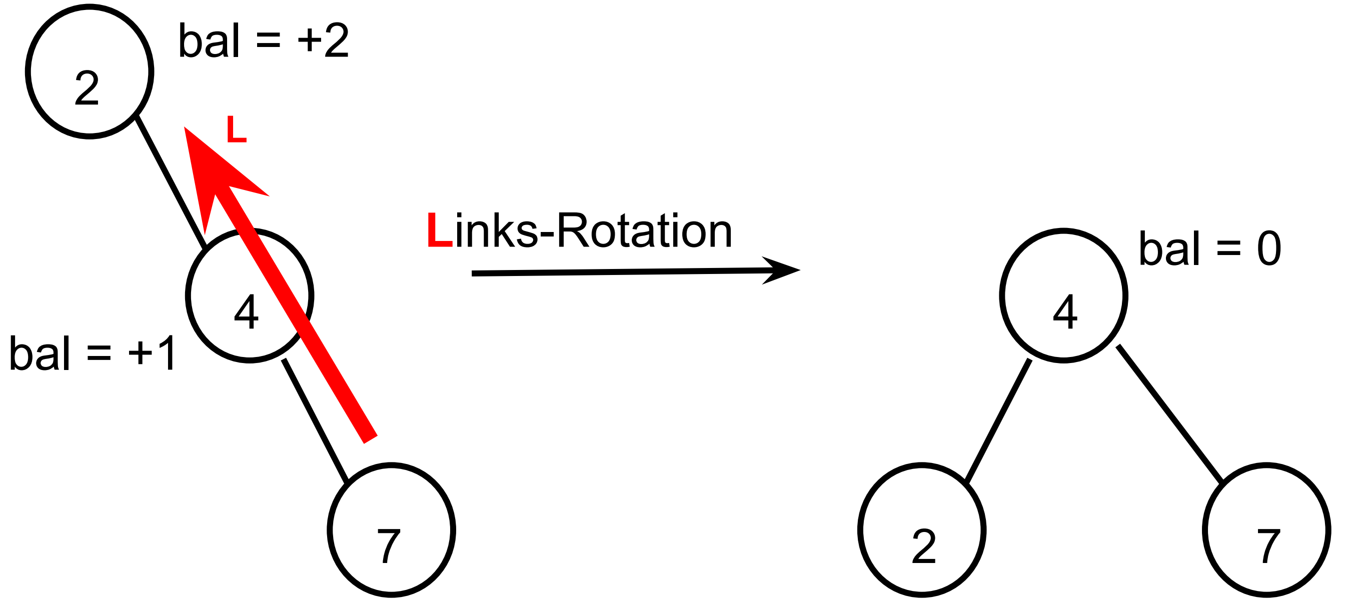
Einfach-Rotation nach rechts

Betrachte immer jeweils den höheren Teilbaum:
Pfeil von unten nach oben zeigt nach rechts.



Fachhochschule Nordwestschweiz

Einfach-Rotation nach links

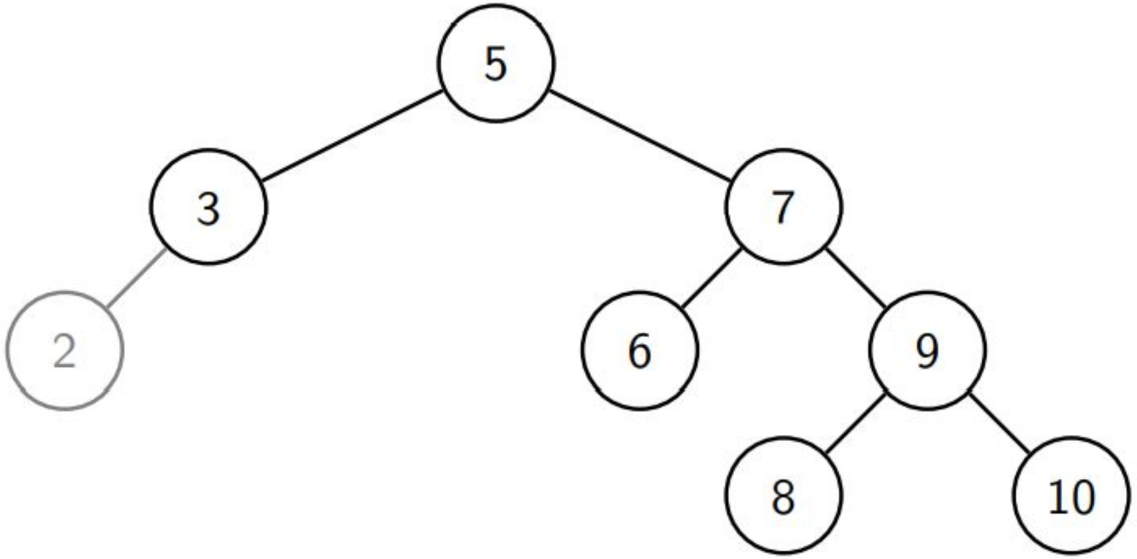


Betrachte immer jeweils den höheren Teilbaum:
Pfeil von unten nach oben zeigt nach links.

AVL-Bäume - Operationen

Aufgabe

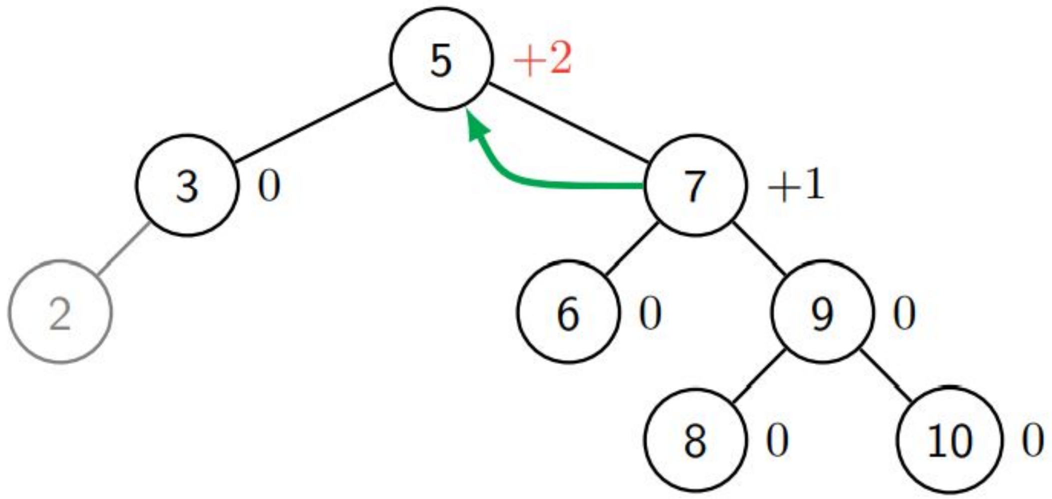
Der folgende unausgeglichene Suchbaum ist durch Entfernen des Knotens mit Schlüssel 2 entstanden. Zeichnen Sie die Balancefaktoren ein und Führen Sie auf dem Suchbaum die notwendige einfache Rotation aus, um ihn wieder auszugleichen.



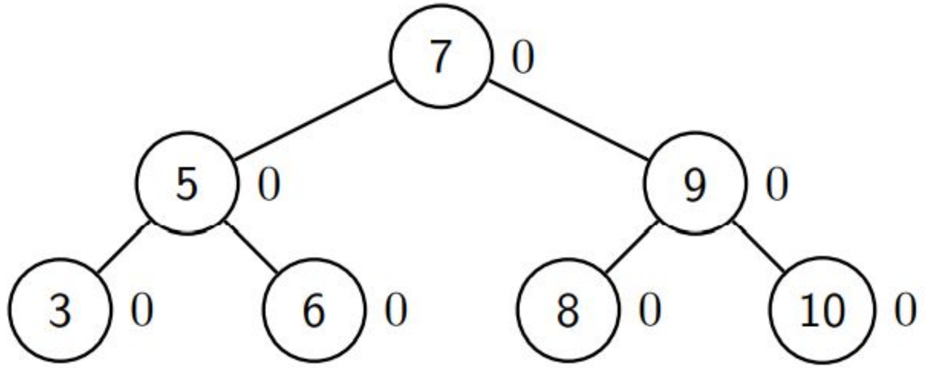
AVL-Bäume - Operationen

Lösung

Im gegebenen Baum liegt nach dem Entfernen des Knotens die Situation 1 vor. Der unausgeglichene Knoten mit Schlüssel 5 und sein Sohn im höheren Teilbaum haben das *gleiche* Vorzeichen.

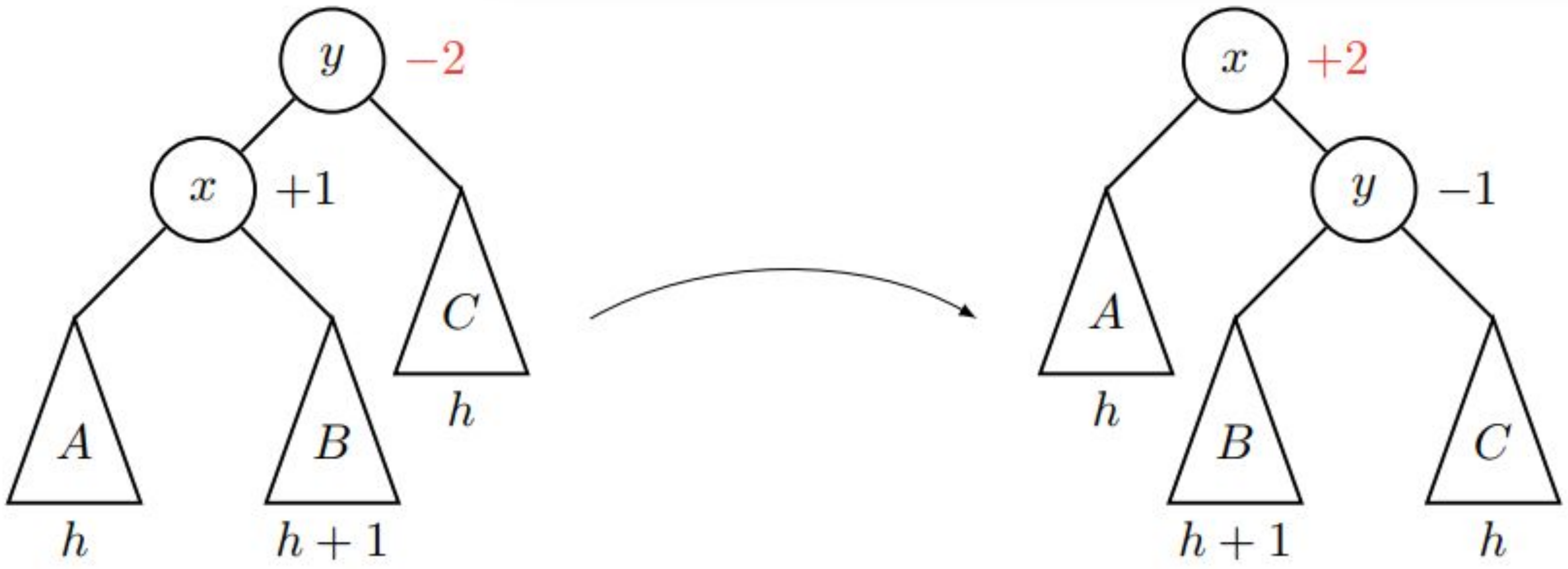


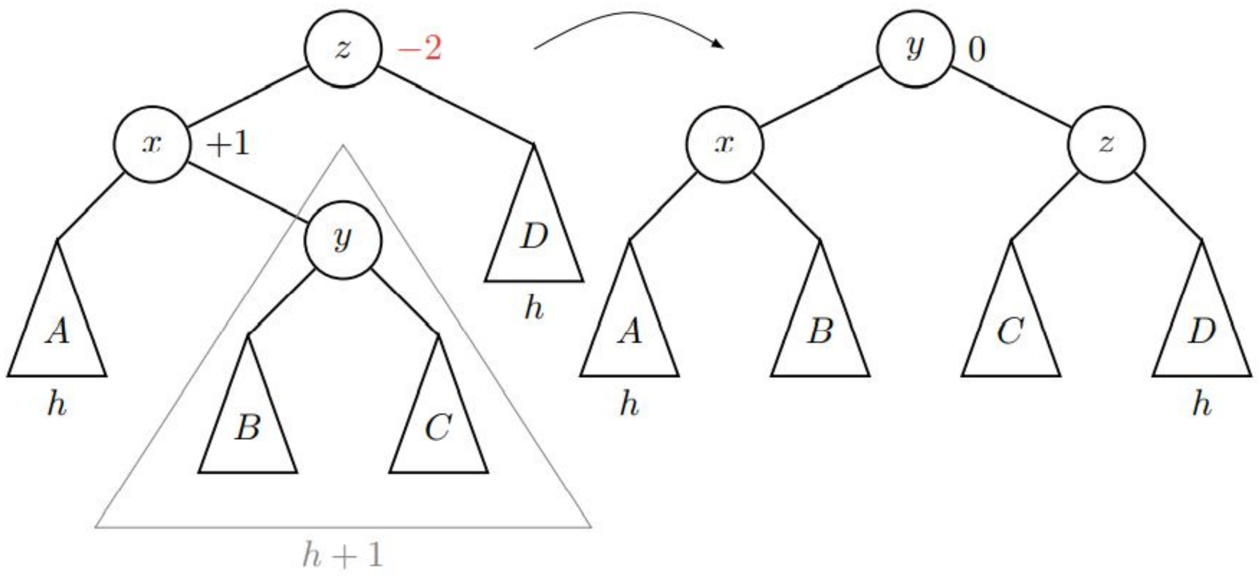
Eine einfache Rotation reicht aus um den folgenden AVL-ausgeglichenen Suchbaum zu erhalten:



AVL-Bäume - Operationen

Situation 3 - Eine einfache Rotation reicht NICHT aus!!!!





AVL-Bäume - Operationen

Situation 3 - Doppelrotation

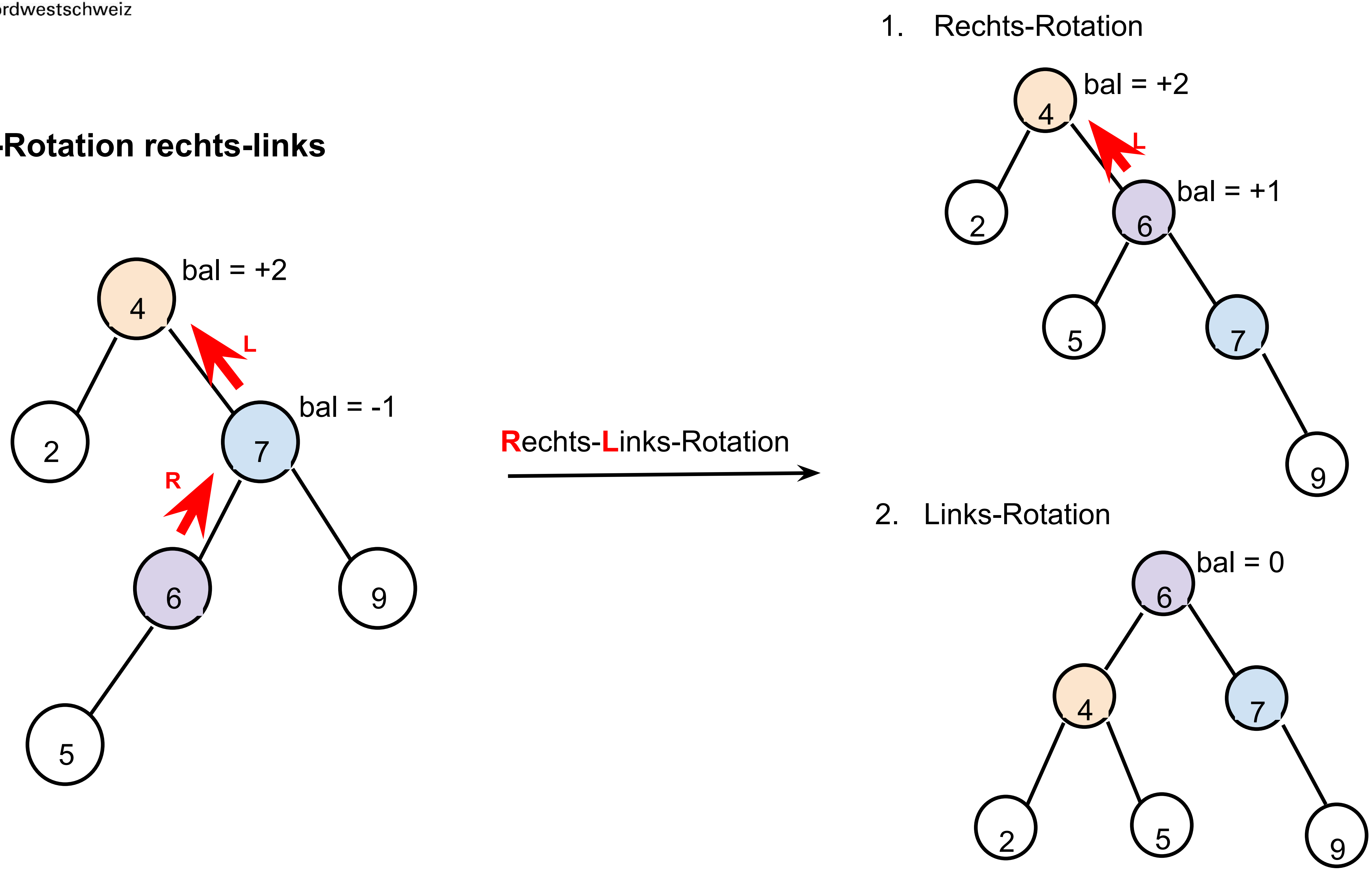
Durch eine **Doppelrotation links-rechts** (Abbildung) wird der Baum, der sich in Situation 3 befindet, ausgeglichen. Der mittlere Teilbaum mit Wurzel y (grau) ist im Vergleich zum Teilbaum D des Knotens z zu hoch. Die Doppelrotation hebt den Knoten y um zwei Niveaus nach oben und er wird somit Vaterknoten von x und z. Der Teilbaum D wird dabei abgesenkt. Die beiden Teilbäume B und C werden zu rechten bzw. linken Teilbäumen der Knoten x und z. Im resultierenden Baum ist die Balance wieder hergestellt.

Die **Doppelrotation rechts-links** ist die spiegelbildliche Variante bei der die mittleren Teilbäume ebenfalls angehoben werden, dafür aber der linke Teilbaum absenkt wird.

Eine Doppelrotation besteht eigentlich aus zwei aufeinanderfolgenden Einfachrotationen (siehe nächste Folien).

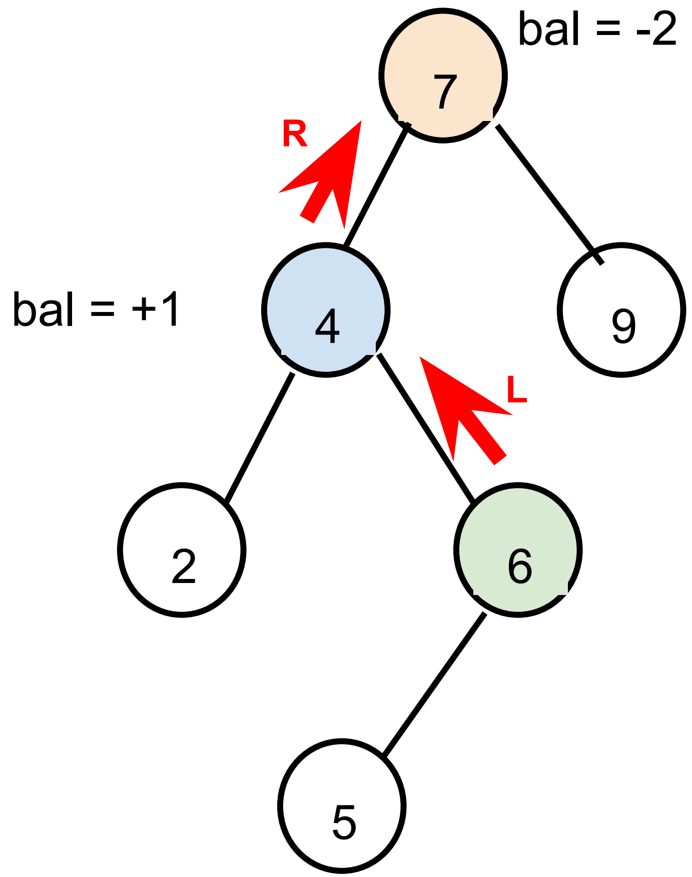
Fachhochschule
Nordwestschweiz

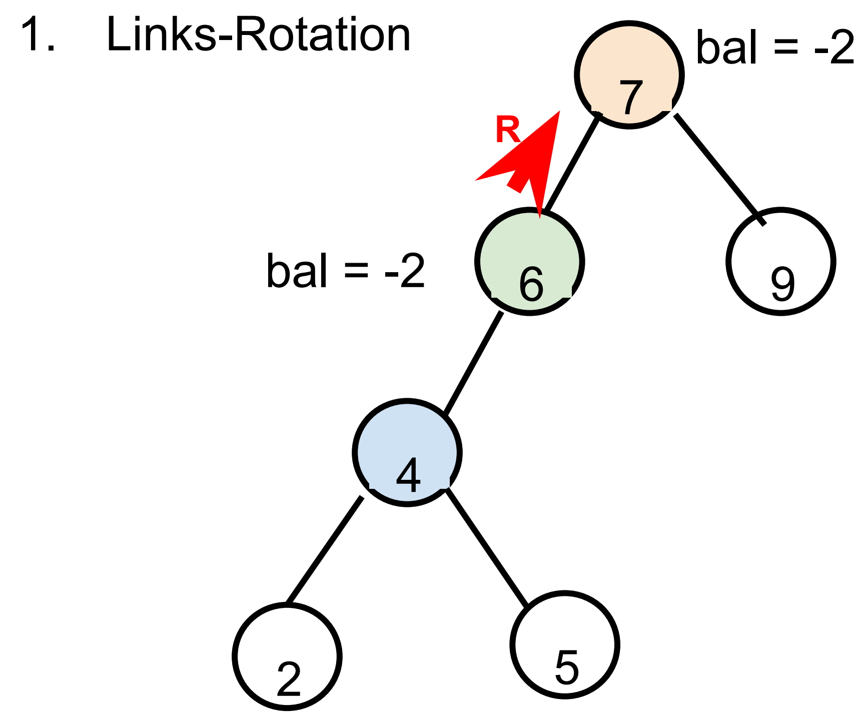
Doppel-Rotation rechts-links

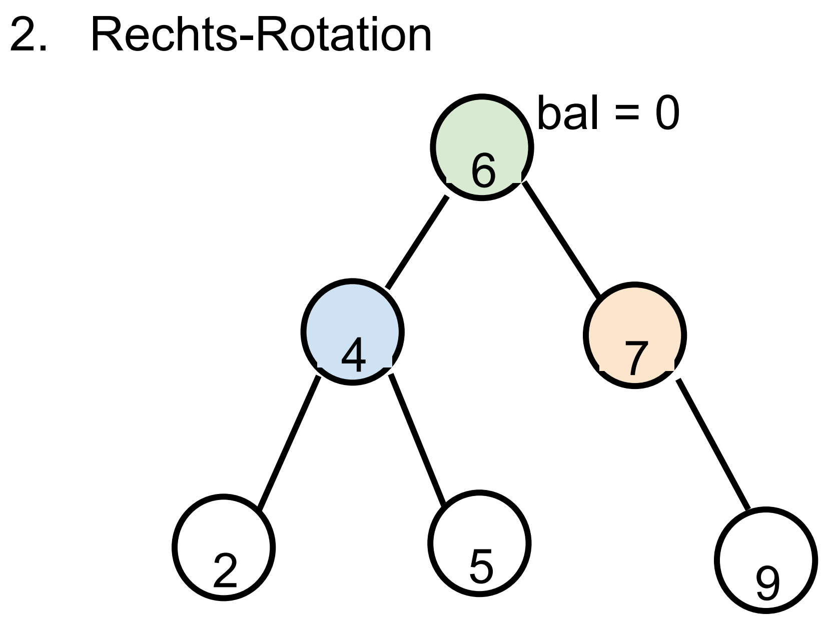


Fachhochschule
Nordwestschweiz

Doppel-Rotation links-rechts



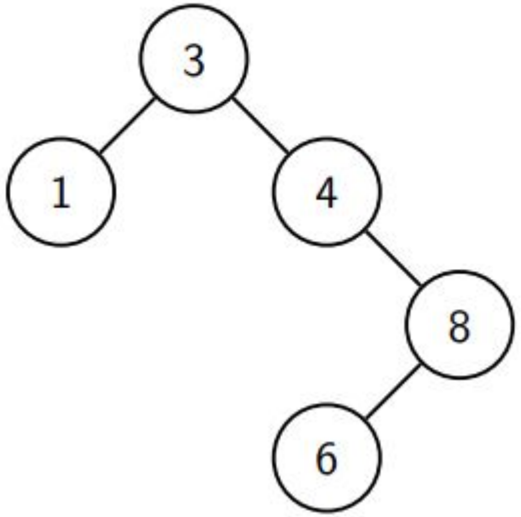




AVL-Bäume - Operationen

Aufgabe

Der folgende unausgeglichene Suchbaum ist durch das Einfügen des Knotens mit Schlüssel 6 entstanden. Führen Sie auf dem Suchbaum die notwendige Doppelrotation aus um ihn wieder auszugleichen.

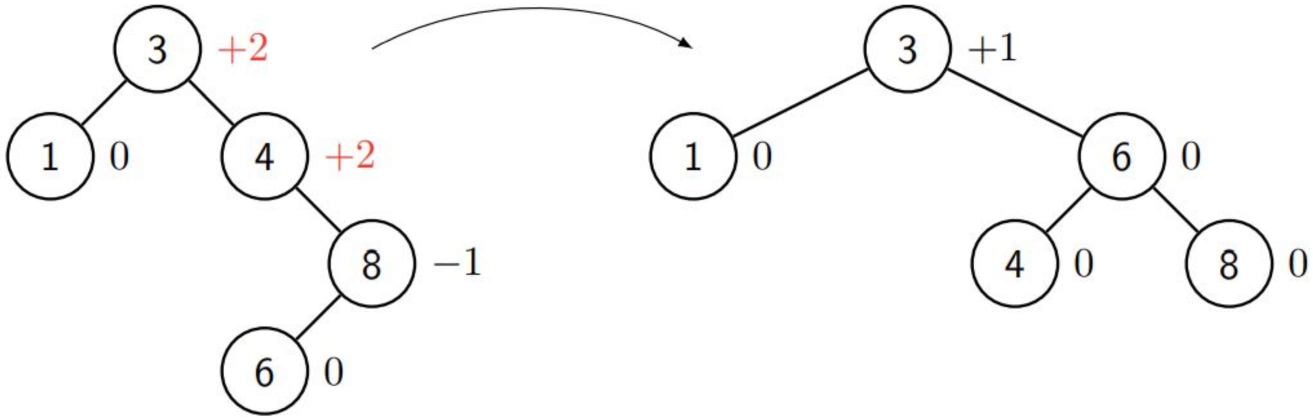


AVL-Bäume - Operationen

Lösung

Der erste unausgeglichene Knoten auf dem Pfad vom eingefügten Knoten bis zur Wurzel ist der Knoten mit Schlüssel 4. Es liegt die Situation 3 vor um den Knoten mit Schlüssel 4 zu wieder auszugleichen. Es wird deshalb eine Doppelrotation rechts-links durchgeführt.

Nach dem Ausgleich des Knotens mit Schlüssel 4 wird dem Pfad weiter zur Wurzel gefolgt. Nun ist aber kein Knoten mehr unausgeglichen und somit ist die Rebalancierung beendet.



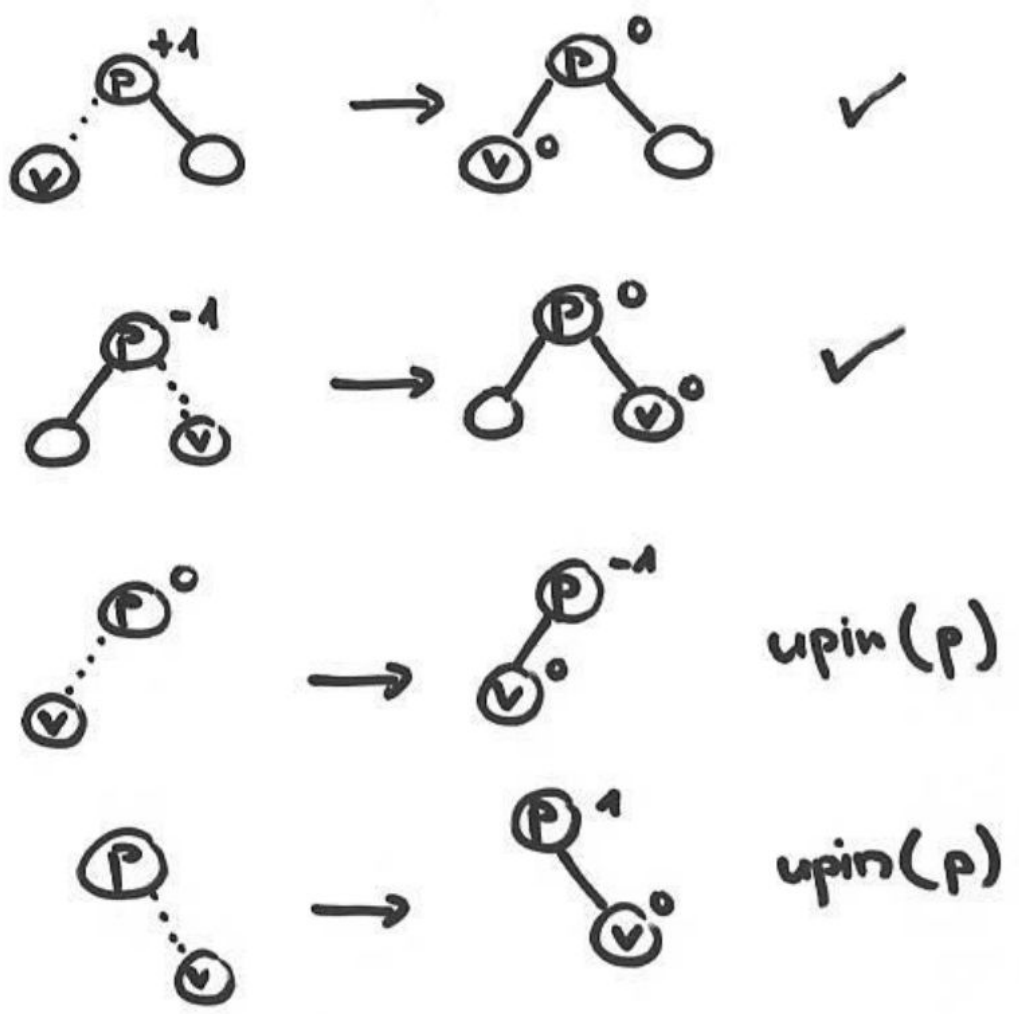
AVL-Bäume - Implementieren

Wie finden wir heraus, ob ein Knoten balanciert ist?

- Die einfachste Möglichkeit ist es, die Höhe der beiden Teilbäume zu berechnen und miteinander zu vergleichen.
- Problem: Dies ist sehr aufwändig!!
- Besser: Balancefaktor in jedem Knoten halten und diesen bei Bedarf updaten mithilfe einer rekursiven Funktion.

AVL-Bäume - Implementieren

Ohne globale Sicht müssen wir die Änderungen der Balancefaktoren *rekursiv* berechnen.



Einfügen

Sei p der Vater des neu eingefügten Knotens v. Wir unterscheiden die nebenstehenden Fälle. (Der Fall, wo in den leeren Baum eingefügt wird ist trivial.)

Ändert sich die Höhe des Teilbaums von p (bal(p) von 0 nach +/-1), dann muss rekursiv der Balancefaktor seines Vaters angepasst werden. Dazu rufen wir die Funktion **upin**(p) auf.

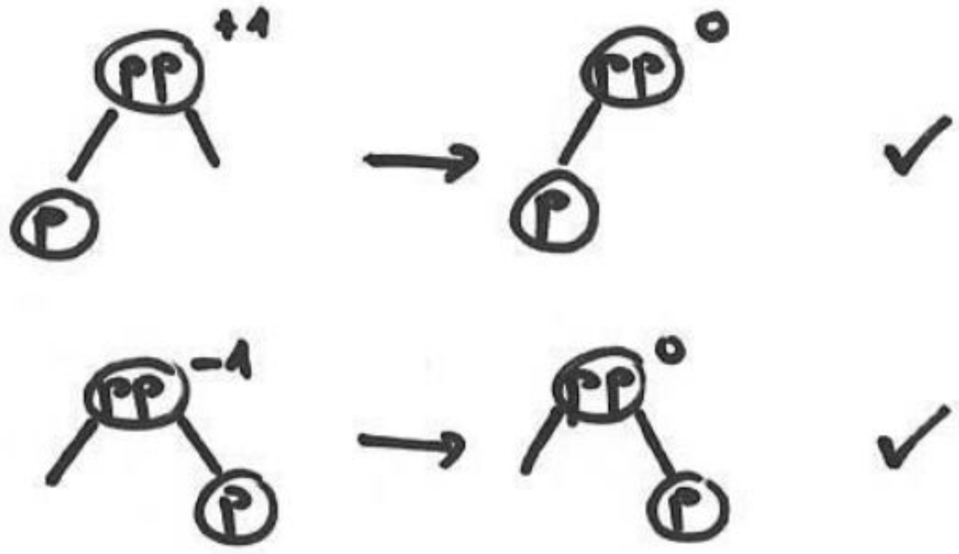
AVL-Bäume - Implementieren

Ohne globale Sicht müssen wir die Änderungen der Balancefaktoren *rekursiv* berechnen.

upin - Fall 1

Sei pp der Vater von p, worauf upin aufgerufen wurde.

Ändert sich die Höhe des Teilbaums von pp nicht (bal(pp) von +/-1 nach 0), sind wir fertig.



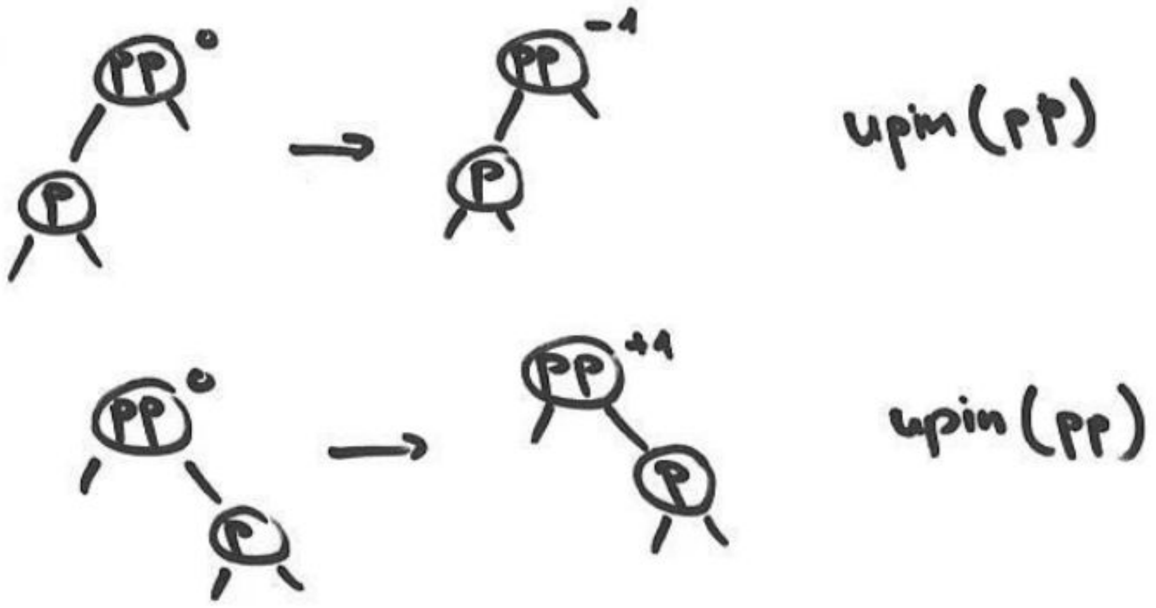
AVL-Bäume - Implementieren

Ohne globale Sicht müssen wir die Änderungen der Balancefaktoren *rekursiv* berechnen.

upin - Fall 2

Sei pp der Vater von p, worauf upin aufgerufen wurde.

Ändert sich die Höhe des Teilbaums von pp, wobei der Knoten aber ausgeglichen bleibt (bal(pp) von 0 nach +/-1), dann muss rekursiv der Balancefaktor seines Vaters angepasst werden. Dazu rufen wir die Funktion **upin**(pp) auf.



AVL-Bäume - Implementieren

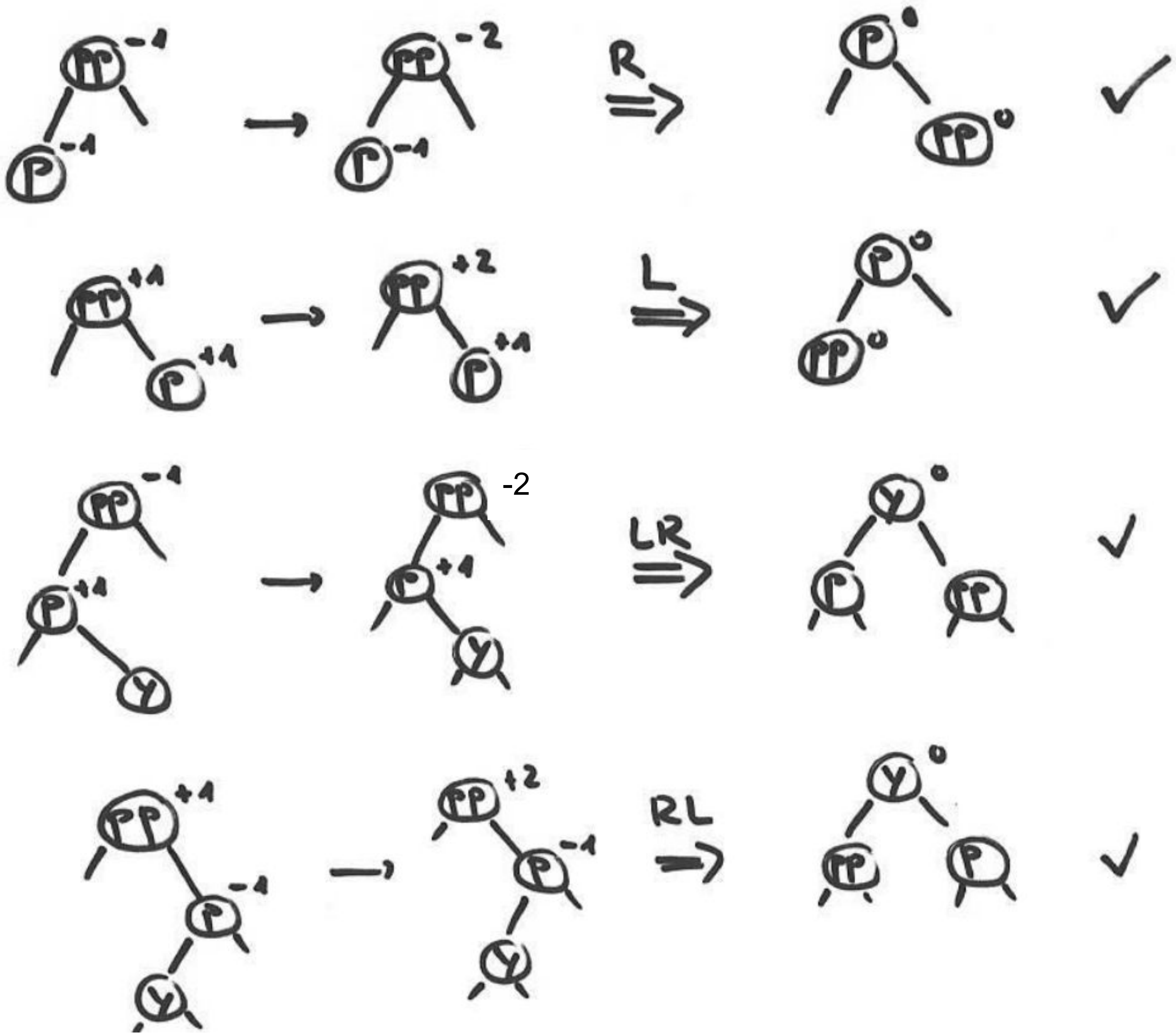
Ohne globale Sicht müssen wir die Änderungen der Balancefaktoren *rekursiv* berechnen.

upin - Fall 3

Sei pp der Vater von p, worauf upin aufgerufen wurde.

Ändert sich die Höhe des Teilbaums von pp und er wird unausgeglichen (bal(pp) +/-1 nach +/-2), dann muss rotiert werden.

Nach der Rotation hat die Wurzel der neuen Teilansicht bal = 0 und wir sind fertig.



Die fehlenden bal-Werte bei LR und RL werden klar, wenn man sie als zwei Einfachrotationen ausführt.

AVL-Bäume - Operationen

Regeln:

- **Einfügen**: Die rekursive Funktion **upin** wird verwendet, um die Balancefaktoren auszugleichen und Rotationen aufzurufen. Nach dem Einfügen erfolgt **maximal eine** Rotation.
- **Löschen**: Die rekursive Funktion **upout** wird verwendet, um die Balancefaktoren auszugleichen und Rotationen aufzurufen. Nach dem Löschen können **mehrere Rotationen** auf dem Pfad zur Wurzel nötig sein!

Hausaufgaben

Arbeitsblatt AVL-Bäume

Programmieren Aufgaben 3 (Dokument wurde geupdatet.)

Quellen:

Alle Zeichnungen von: Binäre Suchbäume (Leitprogramm) – EducETH - ETH-Kompetenzzentrum für Lehren und Lernen | ETH Zürich
(Dieses Leitprogramm habe ich Ihnen auch noch beigelegt als mögliches Skript.)