Lucerne University of
Applied Sciences and Arts

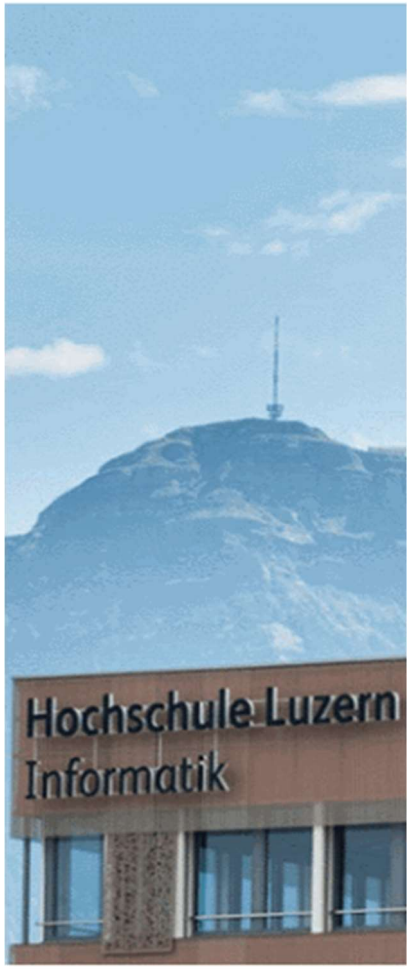
HOCHSCHULE
LUZERN

INFORMATIK

Algorithmen und Datenstrukturen

Datenstrukturen: Binäre Bäume

Roland Gisler



Inhalt

- Grundlagen: Binärer Baum.
- Traversieren von binären Bäumen.
- Binärer Suchbaum.
- Operationen auf einem binären Suchbaum.
- Balancieren von binären Bäumen.
- Zusammenfassung.

Lernziele

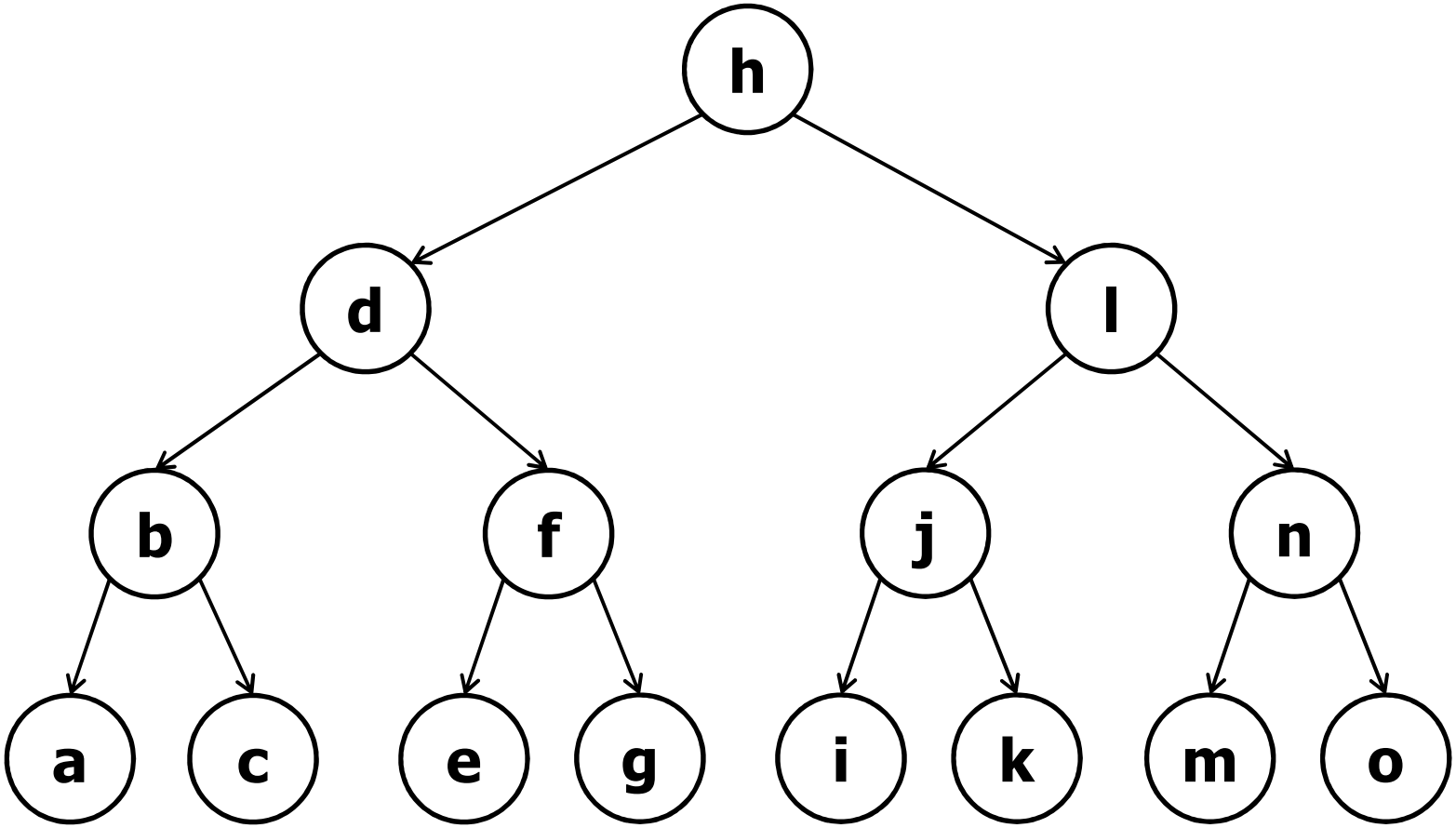
- Sie sind mit binären Bäumen vertraut.
- Sie kennen die Algorithmen um binäre Bäume auf unterschiedliche Arten zu traversieren.
- Sie sind mit den speziellen Eigenschaften von binären Suchbäumen vertraut.
- Sie wissen wie das Suchen, Einfügen und Entfernen von Knoten in binären Suchbäumen konzeptionell abläuft.
- Sie verstehen, was ein ausgeglichener Baum ist und wie man diesen Zustand herstellen kann.

Binäre Bäume

Binärer Baum

- Ein **binärer Baum** (binary tree) ist als Baum mit ➔Ordnung **2** definiert. Jeder Knoten hat somit maximal **zwei** Kinderknoten.
  - Werden als linker und rechter Kinderknoten bezeichnet.
- Binäre Bäume sind in der Informatik sehr beliebt, weil:
  - durch die Beschränkung auf die Ordnung **2** einige Algorithmen stark vereinfacht werden.
  - die Suche in einem Binärbaum einer ➔binären Suche entspricht.
  - auf binären Bäumen unterschiedliche Durchlaufordnungen (➔Traversieren) möglich sind.

Beispiel eines binären Baumes



- Buchstaben **a** bis **o** in einem **kompletten**, und somit auch
  ➔ausgeglichenen, symmetrischen, balancierten Binärbaum.

Traversieren von Binärbäumen

Traversieren eines binären Baumes

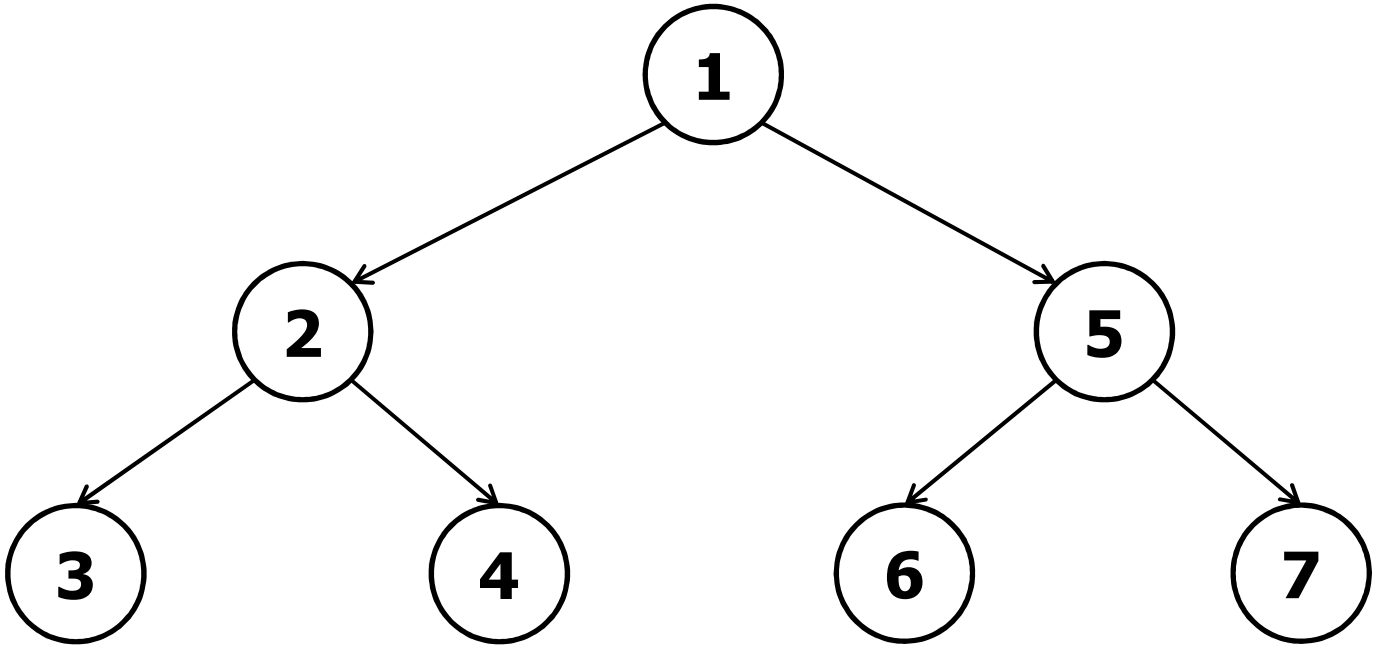
- Aufgrund der spezifischen Eigenschaft von binären Bäumen (Ordnung 2) kann man diese auf drei unterschiedliche Arten traversieren (vgl. iterieren bei Listen):
  - **Preorder** - Hauptreihenfolge
  - **Postorder** – Nebenreihenfolge
  - **Inorder** – Symmetrische Reihenfolge
- Die Algorithmen, welche diese verschiedenen Traversierungen beschreiben sind **➔rekursive** Algorithmen.

Traversieren in Preorder-Reihenfolge

- Bei der **Preorder**-Reihenfolge wird **zuerst** der Knoten (Wurzel des Teilbaumes), dann dessen **linker** Kindknoten und dann dessen **rechter** Kindknoten durchlaufen.
  - auch als Hauptreihenfolge bezeichnet.
- Algorithmus für **`preorder(x)`**:
  1. Besuche (verarbeite) zuerst den **Knoten `x`**.
  2. Traversiere den **linken** Teilbaum von Knoten **`x`** gemäss **`preorder(x.getLeftChild())`**.
  3. Traversiere den **rechten** Teilbaum von Knoten **`x`** gemäss **`preorder(x.getRightChild())`**.

Beispiel: Preorder-Reihenfolge

- Reihenfolge der Knoten bei **Preorder**-Traversierung:

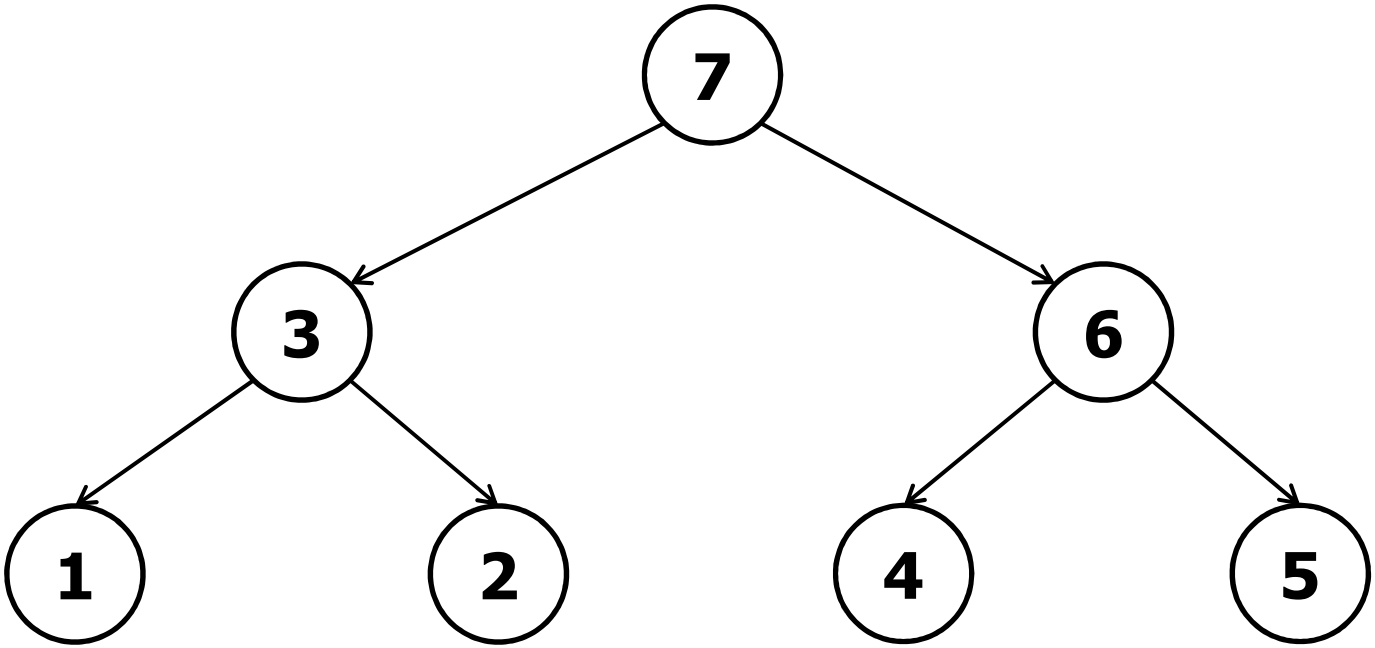


Traversieren nach Postorder-Reihenfolge

- Bei der **Postorder**-Reihenfolge wird von einem Knoten **zuerst** der **linke** Kindknoten, dann der **rechte** Kindknoten und erst **danach** der Knoten **selber** (Wurzel des Teilbaumes) durchlaufen.
  - auch als Nebenreihenfolge bezeichnet.
- Algorithmus für **`postorder(x)`**:
  1. Traversiere den **linken** Teilbaum von Knoten **`x`** gemäss **`postorder(x.getLeftChild())`**.
  2. Traversiere den **rechten** Teilbaum von Knoten **`x`** gemäss **`postorder(x.getRightChild())`**.
  3. Besuche (verarbeite) erst dann den **Knoten `x`**.

Beispiel: Postorder-Reihenfolge

- Reihenfolge der Knoten bei **Postorder**-Traversierung:

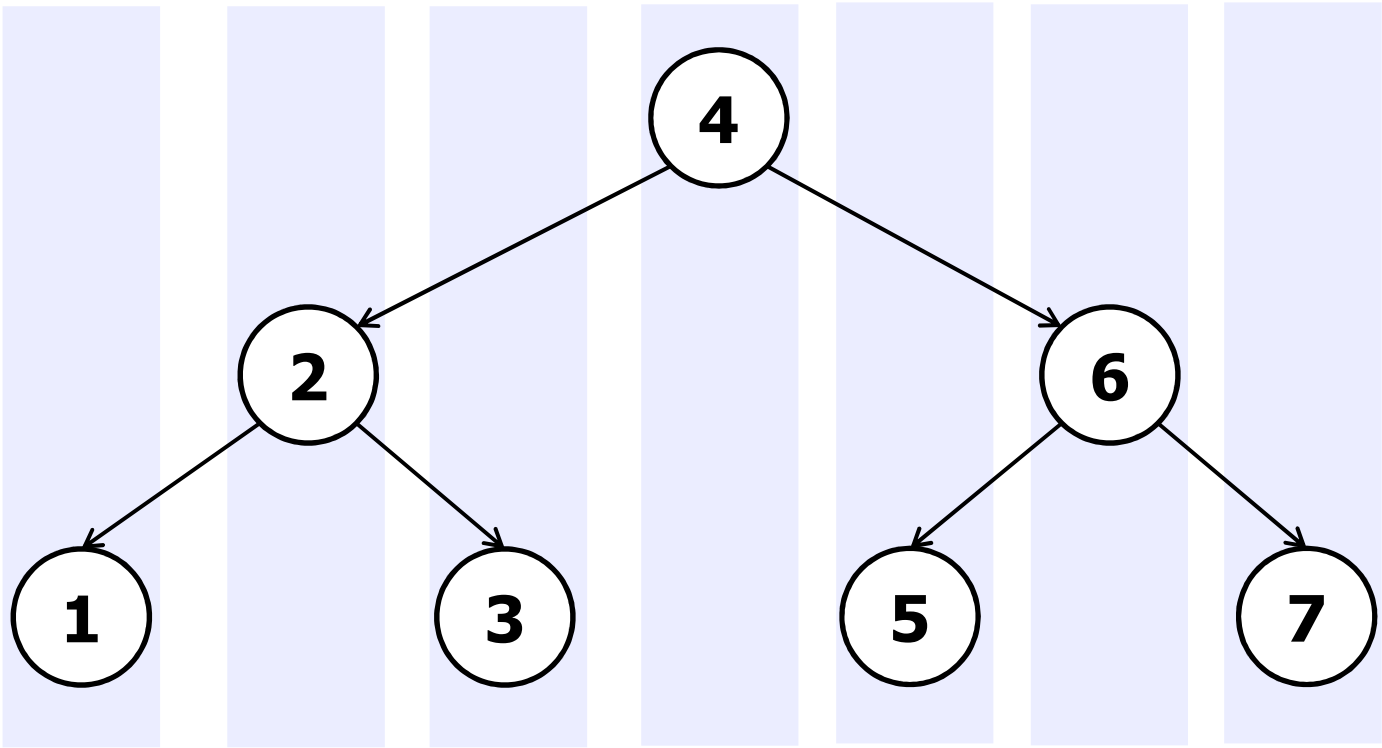


Traversieren nach Inorder-Reihenfolge

- Bei der **Inorder**-Reihenfolge wird von einem Knoten **zuerst** der **linke** Kindknoten, dann der Knoten **selber** (Wurzel des Teilbaumes) und dann der **rechte** Kindknoten durchlaufen.
  - auch als **symmetrische Reihenfolge** bezeichnet.
- Algorithmus für **`inorder(Knoten)`**:
  1. Traversiere den **linken** Teilbaum von Knoten **`x`** gemäss **`inorder(x.getLeftChild())`**.
  2. Besuche (verarbeite) den **Knoten `x`**.
  3. Traversiere den **rechten** Teilbaum von Knoten **`x`** gemäss **`inorder(x.getRightChild())`**.

Beispiel: Inorder-Reihenfolge

- Bei einem als ➔binären Suchbaum aufgebauten Baum liefert die **Inorder**-Reihenfolge die enthaltenen Elemente gemäss ihrer Sortierung!



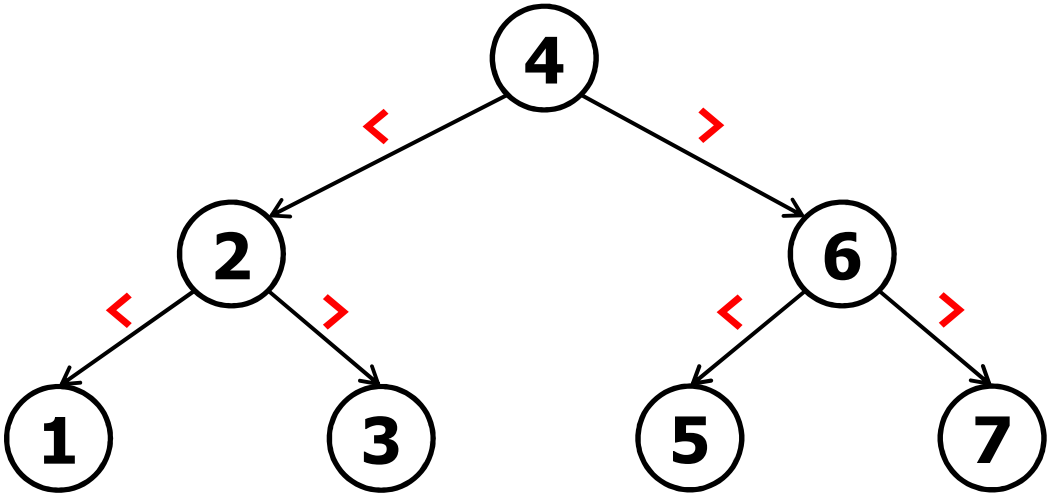
Binäre Suchbäume

Binäre Suchbäume

- Binäre **Such**bäume (binary search trees) sind im wesentlichen identisch mit binären Bäumen.
  - Haben auch Ordnung **2**.
- Als Erweiterung enthalten Sie neben den eigentlichen Datenelementen jedoch noch einen Schlüsselwert, nach welchem die Datenelemente im Baum ➔geordnet werden und nach welchem dann auch effizient ➔binär gesucht werden kann.
- Dieser Schlüssel kann
  - aus (Teil-)Daten des Datenelementes bestehen.
    Beispiel: Matrikelnummer eines **`Student`**-Objektes.
  - oder aus (Teil-)Daten des Datenelementes berechnet werden.
    Beispiel: Hashwert (**`hashCode()`**) des Datenelementes.

Geordneter binärer Suchbaum

- Bei einem geordneten binären Suchbaum gelten folgende Regeln:
  - Jeder Schlüssel im **linken** Teilbaum eines Knotens ist **kleiner** als der Schlüssel im Knoten selbst.
  - Jeder Schlüssel im **rechten** Teilbaum eines Knotens ist **grösser** (oder gleich*) dem Schlüssel im Knoten selbst.
- Daraus ergibt sich z.B. folgender Aufbau:



Dadurch wird eine sehr effiziente ➔**binäre Suche** möglich.

*wenn (Schlüssel-)Duplikate überhaupt erlaubt werden

Operationen auf binären Suchbäumen

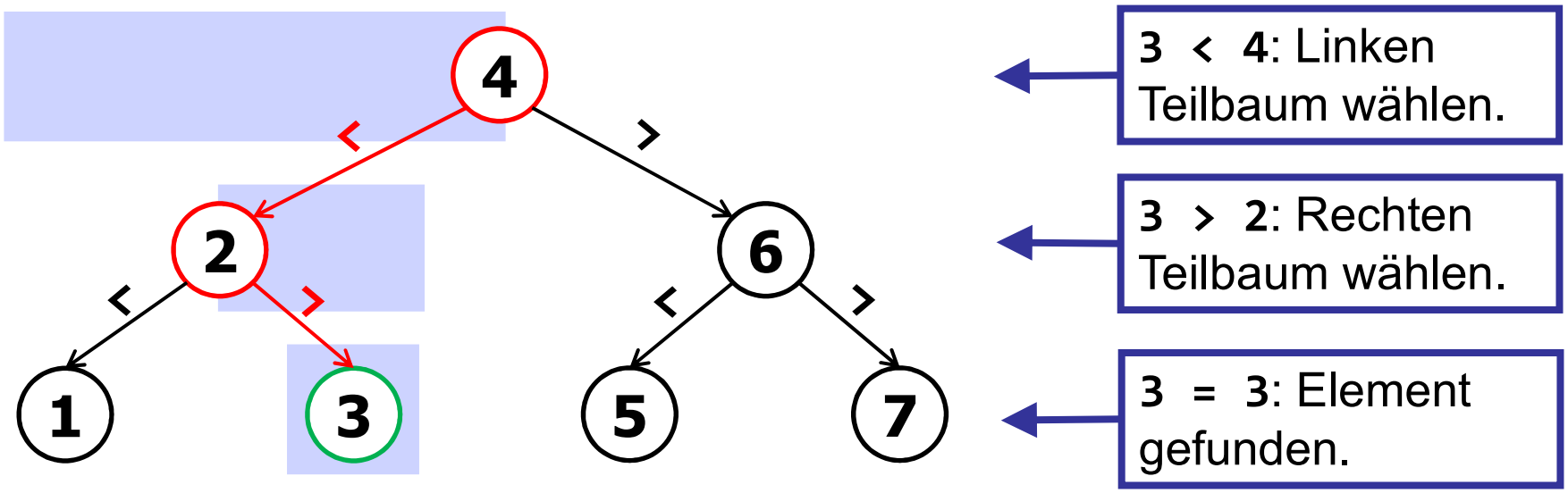
Operationen auf binären Suchbäumen

Folgende wichtige Operationen auf binären (Such-)Bäumen wollen wir neben dem bereits behandelten ➔Traversieren betrachten:

- **Suchen** eines Elementes.
- **Einfügen** eines neuen Elementes.
- **Entfernen** eines Elementes.
- **Balancieren** eines Baumes (ausgleichen).

Binärer Suchbaum – Suche (binär)

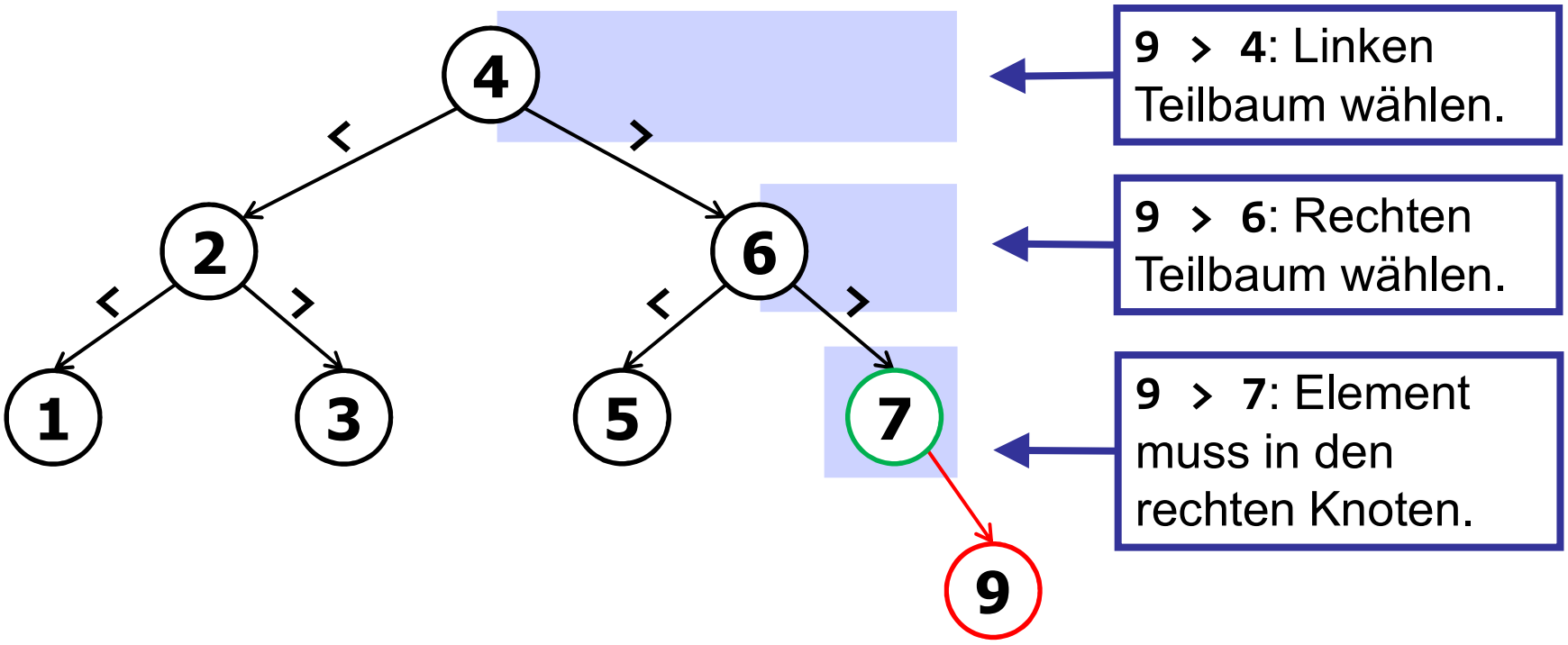
- Bei der binären Suche (siehe auch Input D12) wird die Suchmenge sukzessive halbiert, bis das gesuchte Element entweder gefunden ist, oder man eine leere Menge hat (➔ nicht gefunden).
- Beispiel: Suche nach dem Wert **3**



- Aufwand ist somit $\log_2 n$, entspricht O(log n), unter der Bedingung, dass der Baum ausgeglichen (balanciert) ist.

Binärer Suchbaum - Einfügen

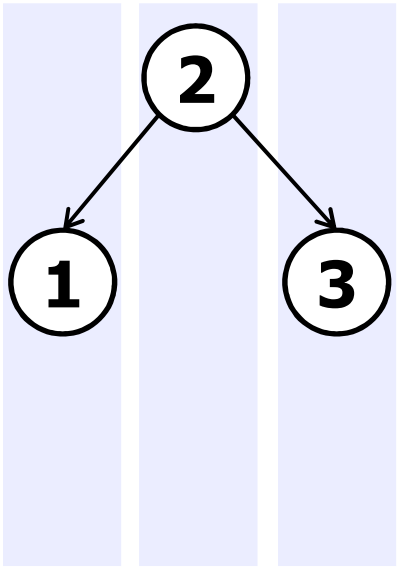
- Um ein Element in einen binären Suchbaum einzufügen, **sucht** man nach dem Element und fügt es dann an der Stelle ein, an welcher man es hätte finden müssen.
  - Das Element wird mit einem **neuen** Knoten ergänzt.
- Beispiel: Element mit Wert **9** einfügen.



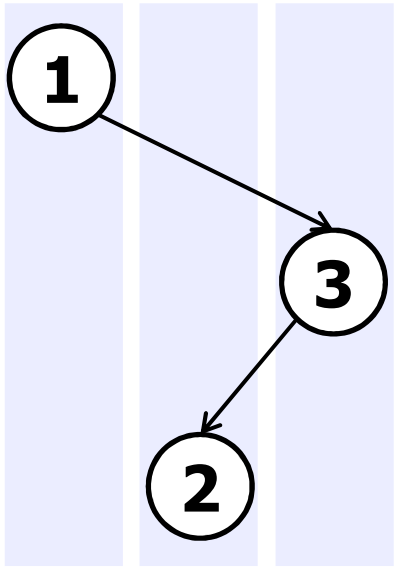
Binärer Suchbaum – Einfluss der Einfügereihenfolge

- Spätestens jetzt wird klar, das die konkrete Struktur eines binären Suchbaumes davon abhängig ist, in welcher Reihenfolge die Elemente eingefügt werden.
- Beispiele für verschiedene Reihenfolgen:

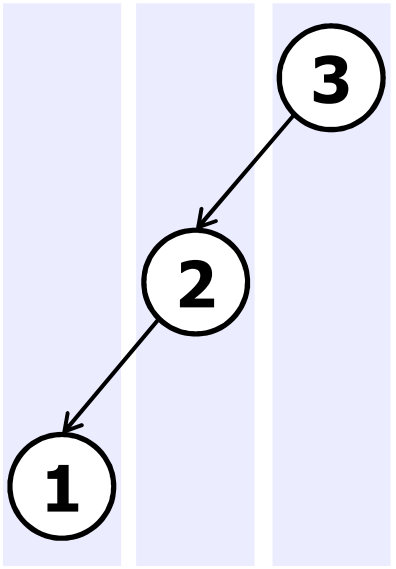
2, 1, 3:
(oder 2, 3, 1)



1, 3, 2:

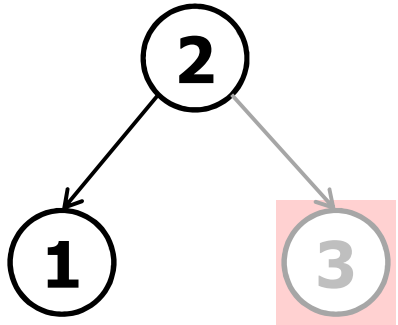


3, 2, 1:



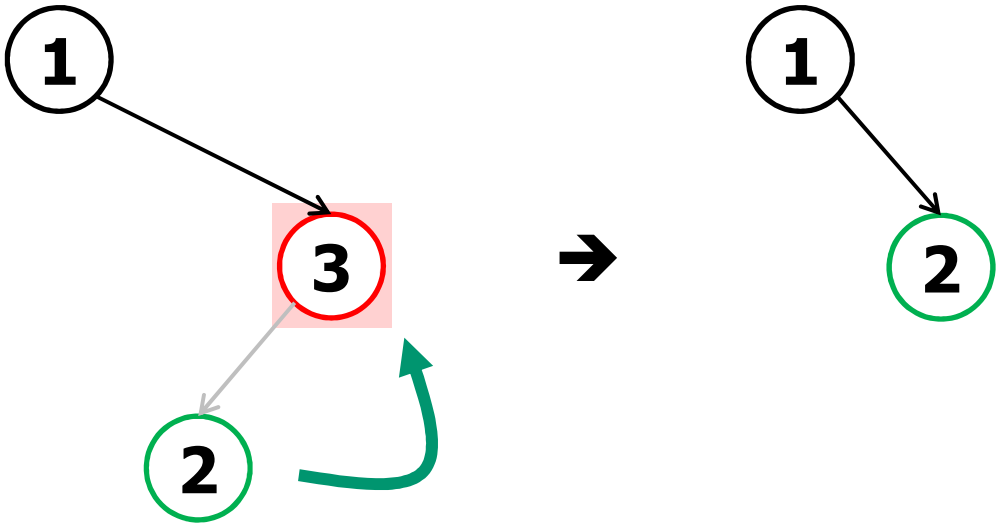
Binärer Suchbaum – Entfernen eines Elementes

- Das Entfernen eines Elementes ist nicht ganz so einfach wie es auf den ersten Blick erscheint!
- Nachdem man das zu entfernende Element durch die ➔binäre Suche gefunden hat, gibt es **drei** verschiedene Fälle:
  - Das Element ist ein **Blatt** (**keine** Kindknoten).
  - Das Element ist ein **innerer Knoten** mit **einem** Kindknoten.
  - Das Element ist ein **innerer Knoten** mit **zwei** Kindknoten.
- Der **erste** Fall ist der einfachste Fall:
  Ein Blatt kann ganz einfach und ohne weitere Schritte entfernt werden.
- Beispiel: Element **3** entfernen:



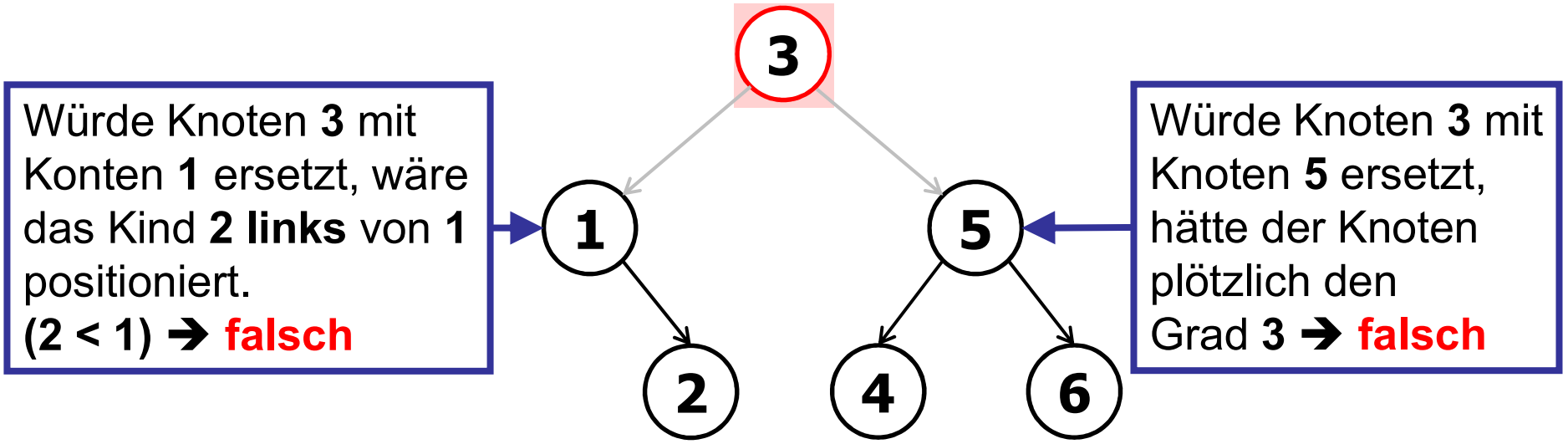
Binärer Suchbaum – Element mit einem Kind entfernen

- Zweiter Fall: Entfernen eines Elementes das einen Kindknoten (inklusive allfälligem Teilbaum!) hat.
  - Egal ob das Kind links oder rechts angeordnet ist:
    Alle weiteren Kindern sind grösser als der Vater des Elementes.
- Vorgehen:
  - Das Element wird gelöscht.
  - Der Kindknoten nimmt den Platz des gelöschten Knotens ein.
- Beispiel:
  Element **3** entfernen:



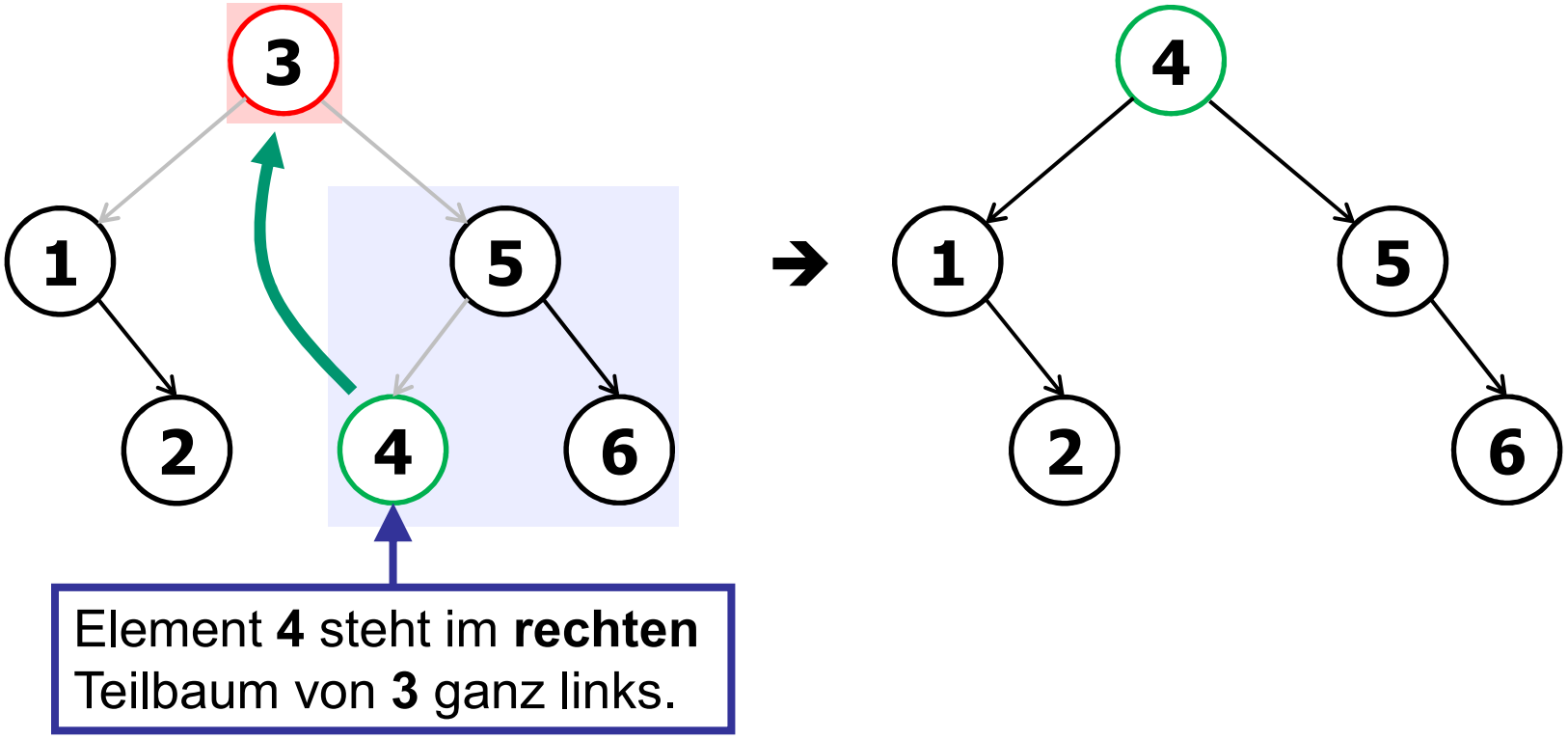
Binärer Suchbaum – Element mit zwei Kindern entfernen

- Dritter Fall: Entfernen eines Elementes das **zwei** Kindknoten (inklusive allfälligem Teilbaum!) hat.
- Dieser Fall ist etwas komplizierter: Wir können **nicht** einfach eines der Kinder an den Platz des Elementes setzen, da sonst die Ordnung des Baumes verletzt würde.
- Beispiel: Element **3** entfernen, zwei verschiedene Fälle:



Binärer Suchbaum – Element mit zwei Kindern entfernen

- Lösung: Man ersetzt das gelöschte Element mit demjenigen Knoten aus dem **rechten** Teilbaum, das in diesem am **weitesten links** steht (und somit den kleinsten Wert im rechten Teilbaum hat).
- Beispiel: Element **3** entfernen:

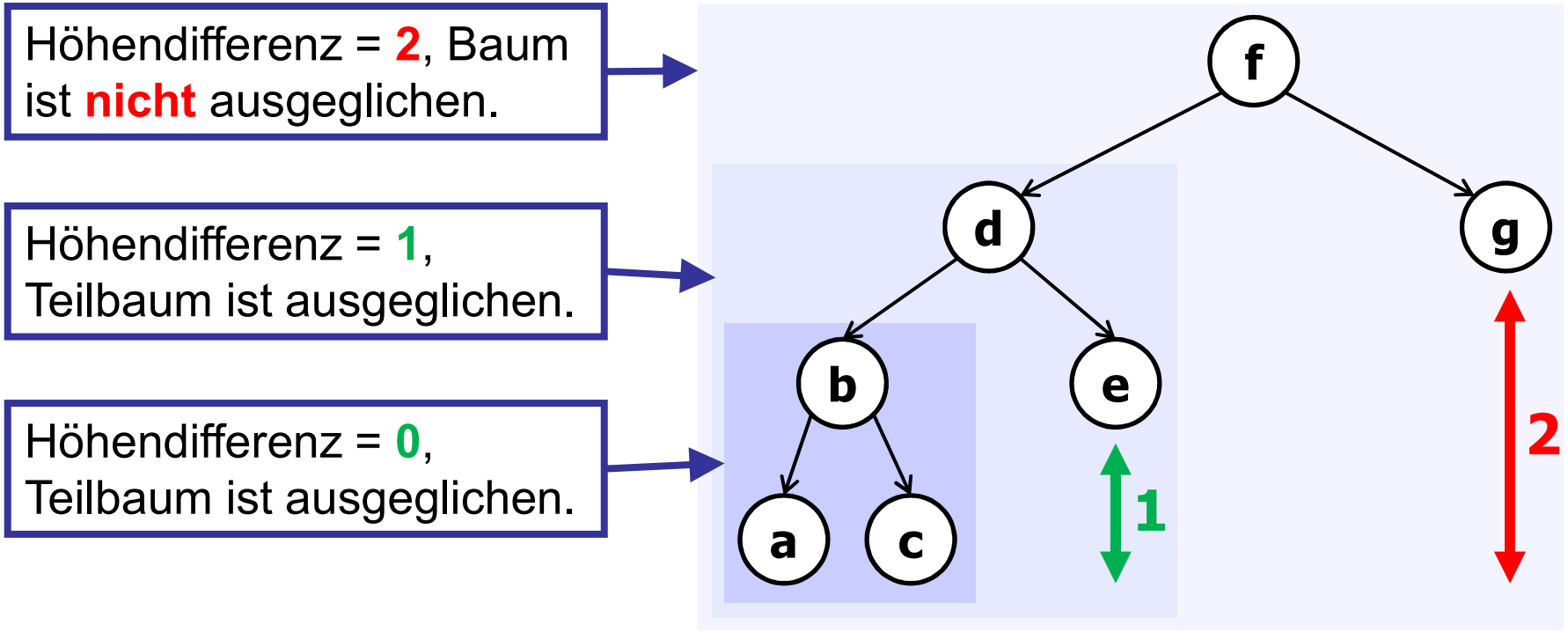


Balancieren von binären (Such-)Bäumen

- Um eine maximale Performance bei der Suche zu gewährleisten, sollte ein Baum im Verhältnis zu seinem Gewicht eine minimale Anzahl Niveaus enthalten, und möglichst ausgeglichen aufgebaut sein.
- Dieser Zustand lässt sich wie folgt definieren:
  Ein Baum ist ausgeglichen, wenn sich die Höhen des linken und des rechten Teilbaumes **jedes** Knoten um maximal **1** unterscheiden.
  - Dieser Zustand kann sich bei jeder Einfügung oder Entfernung eines Elementes auf dem Pfad des betroffenen Elementes ändern.
- Es gibt Algorithmen, welche einen (Teil-)Baum restrukturieren können, damit er wieder ausgeglichen ist.

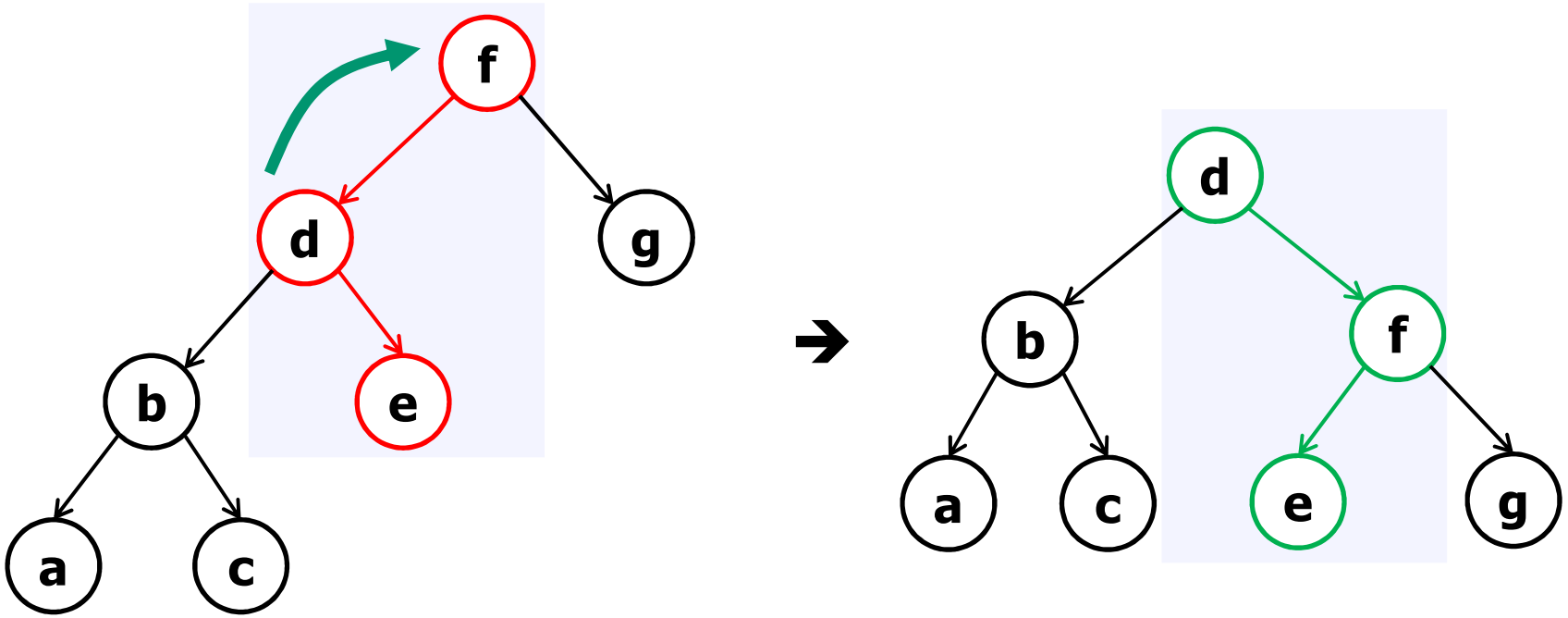
Beispiel: Nicht ausgeglichener Baum

- Der folgende Baum ist nicht ausgeglichen:
  Die Höhendifferenz des linken und des rechten Teilbaumes von Element **f** ist mit **2** grösser als 1.



Ausgleichen durch Rotation

- Als Beispiel wird hier die **einfache Rotation** nach **rechts** gezeigt:



- Die Knoten **d** und **f** «rotieren». Dabei wird **rechte** Teilbaum (e) von Knoten **d** neu zum **linken** Teilbaum von Knoten **f**.
  - Äquivalent ist die einfache Rotation nach links (Baum gespiegelt).
- Es gibt noch die **Doppelrotation**, die wir hier aber weglassen.

Ausgleichen eines Baumes

- Soll ein binärer Suchbaum ständig ausgeglichen sein, muss nach **jedem** Einfügen oder Entfernen eines Elementes die Höhendifferenzen auf dem Pfad zum Element **überprüft**, und der Baum u.U. durch mehrere Rotationen wieder ausbalanciert werden.
- Das macht die entsprechenden Operationen zwar aufwändiger, sie behalten aber ihre Ordnung O(log n).

Zusammenfassung

- Binärer Baum mit Ordnung 2 sehr häufig eingesetzter Baum.
  - Weil sehr gut geeignet für binäre Suche.
- Operationen auf einem Baum:
  - Traversieren von allen Elementen.
  - Suchen, Einfügen und Entfernen von Elementen.
- Abhängig von den Daten die eingefügt oder entfernt werden kann die Baumstruktur ausgeglichen sein, oder aber im Extremfall komplett degenerieren.
- Das Ziel von ausgeglichenen Bäumen wird durch die Rotationen zur Rebalancierung des Baumes erreicht.

Fragen?