INSTITUTE OF VISUAL COMPUTING

TU Graz

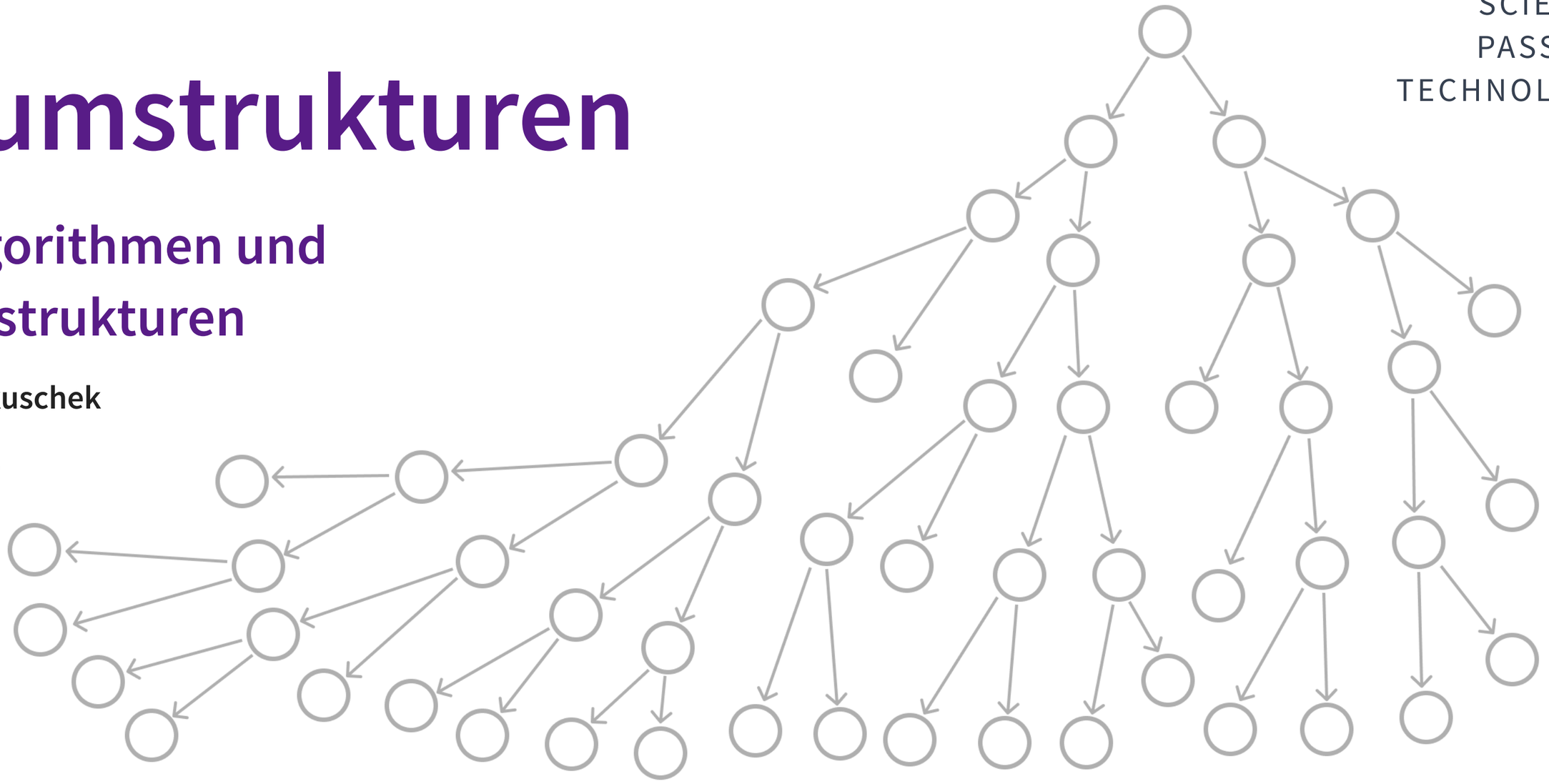
SCIENCE
PASSION
TECHNOLOGY

Baumstrukturen

VU Algorithmen und Datenstrukturen

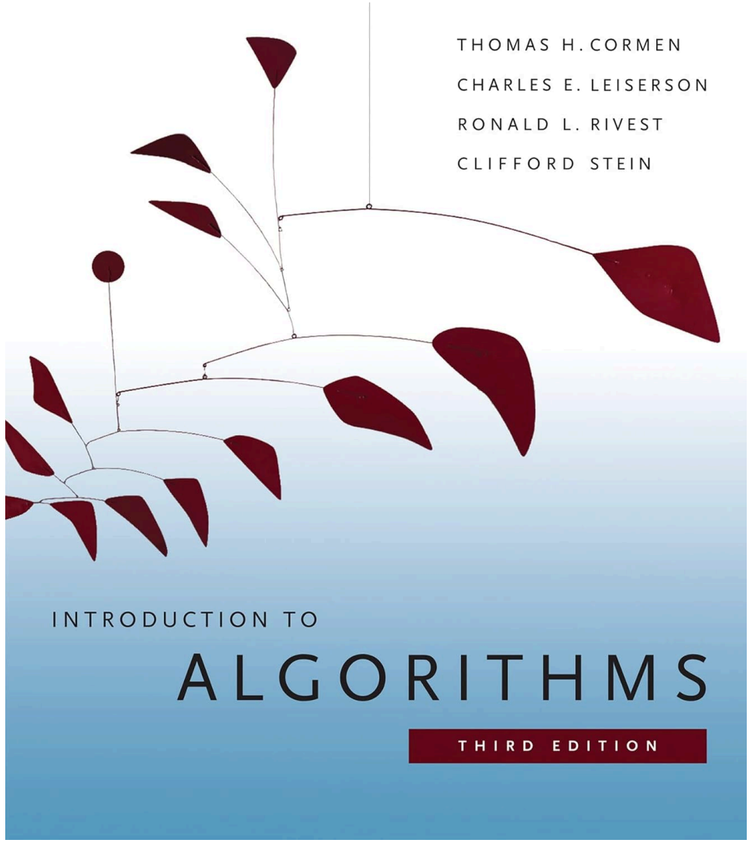
Julian Rakuschek

20.11.2026



Agenda

- Zugriffsmethoden
  - Teilweise Wiederholung, zur Einordnung
- Definition von Bäumen
- Binäre Suchbäume
- AVL Bäume
- Rot-Schwarz-Bäume
- B-Bäume



Kapitel 12, 13 und 18

Wiederholung und Einordnung

Zugriffsmethoden

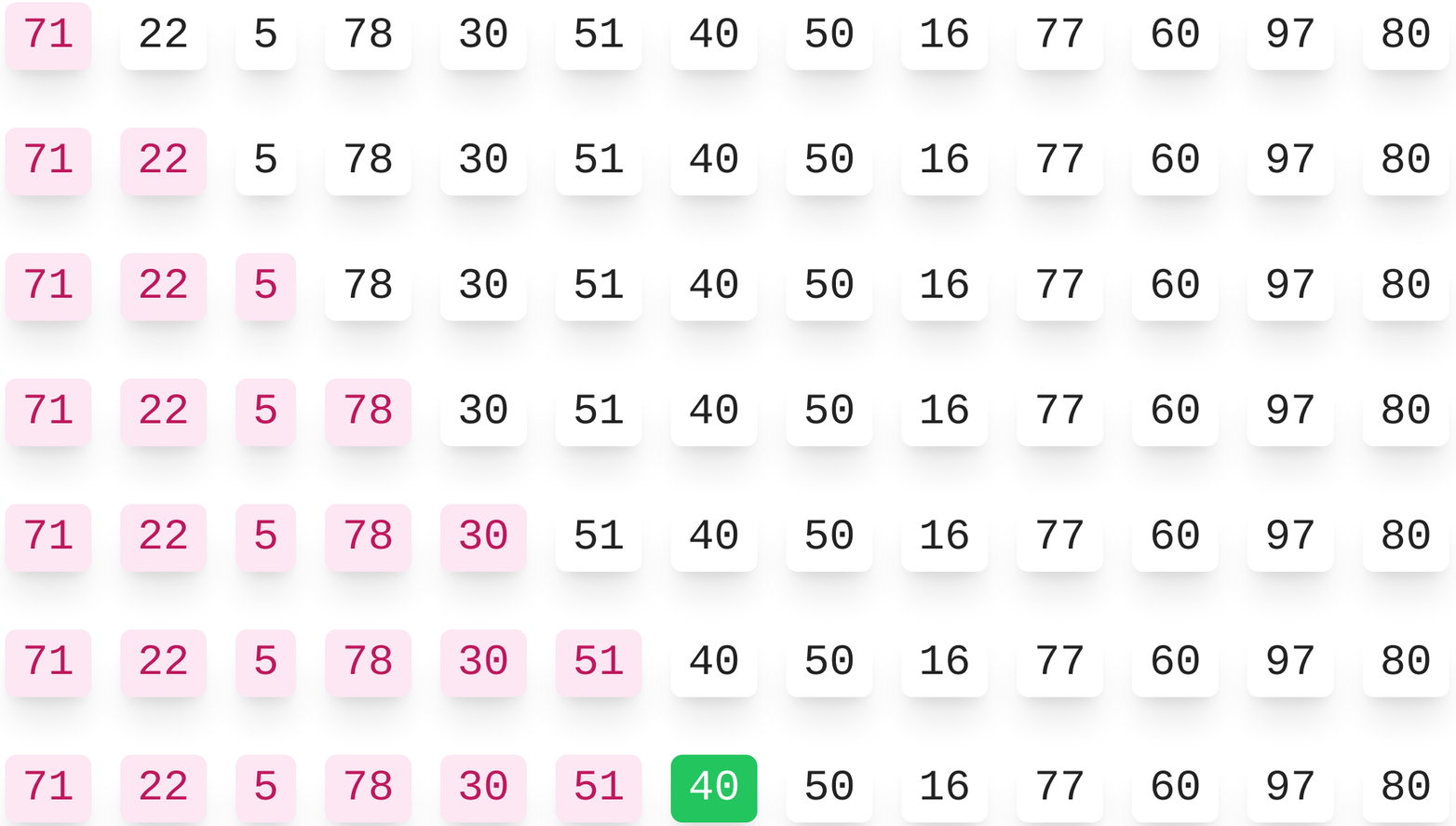
Wiederholung: Suche in einer Liste

Gegeben ist eine unsortierte Liste

71	22	5	78	30	51	40	50	16	77	60	97	80

Ziel: Element 40 in der Liste finden.

Wir müssen die Liste der Reihe nach durchgehen



Sequentieller Scan in $\mathcal{O}(n)$

Was wenn wir häufig in der Liste suchen müssen?

Wollen wir jedes Mal in $\mathcal{O}(n)$ die Liste durchsuchen?

Besser: Wir sortieren die Liste und wenden dann Binärsuche an!

D.h. wir investieren einmal mehr Zeit mit $\mathcal{O}(n \log n)$ Sortierung, um danach in $\mathcal{O}(\log n)$ Zeit zu suchen.

Refresher: Binärsuche

Wir wollen das Element 78 in der Liste finden.

5	16	22	30	40	50	51	60	71	77	78	80	97

Binärsuche benötigt nur $\mathcal{O}(\log n)$ Schritte.

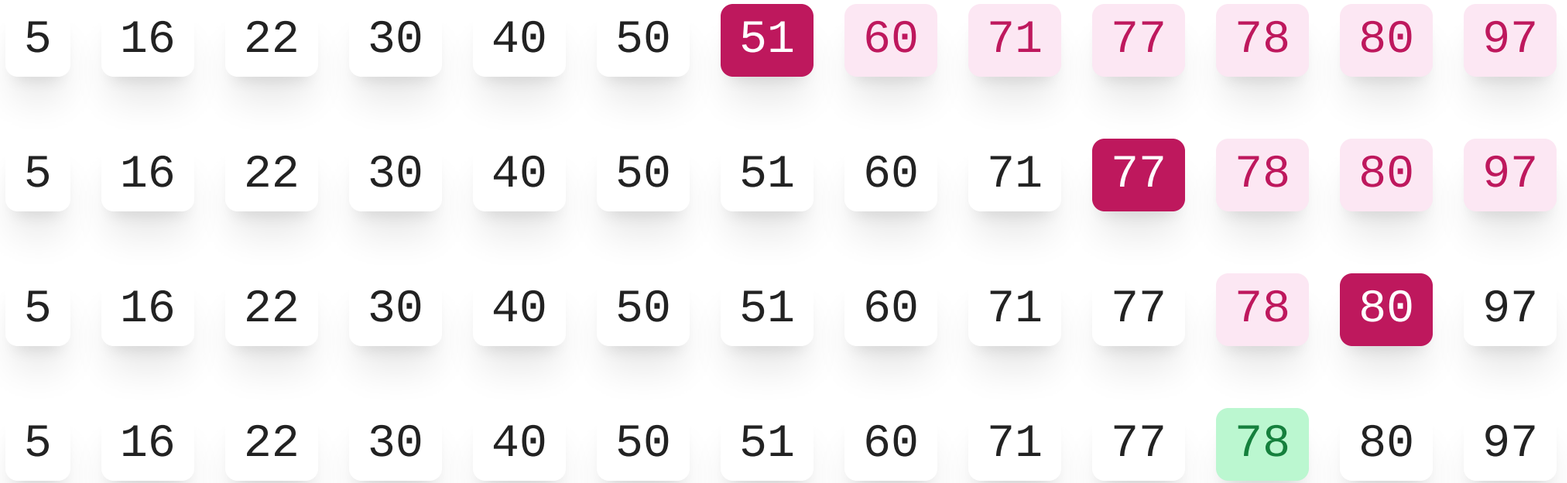
Refresher: Binärsuche

Idee: Wenn die Liste sortiert ist, wissen wir anhand vom Median in welcher Hälfte der Liste unsere gesuchte Zahl liegen muss.

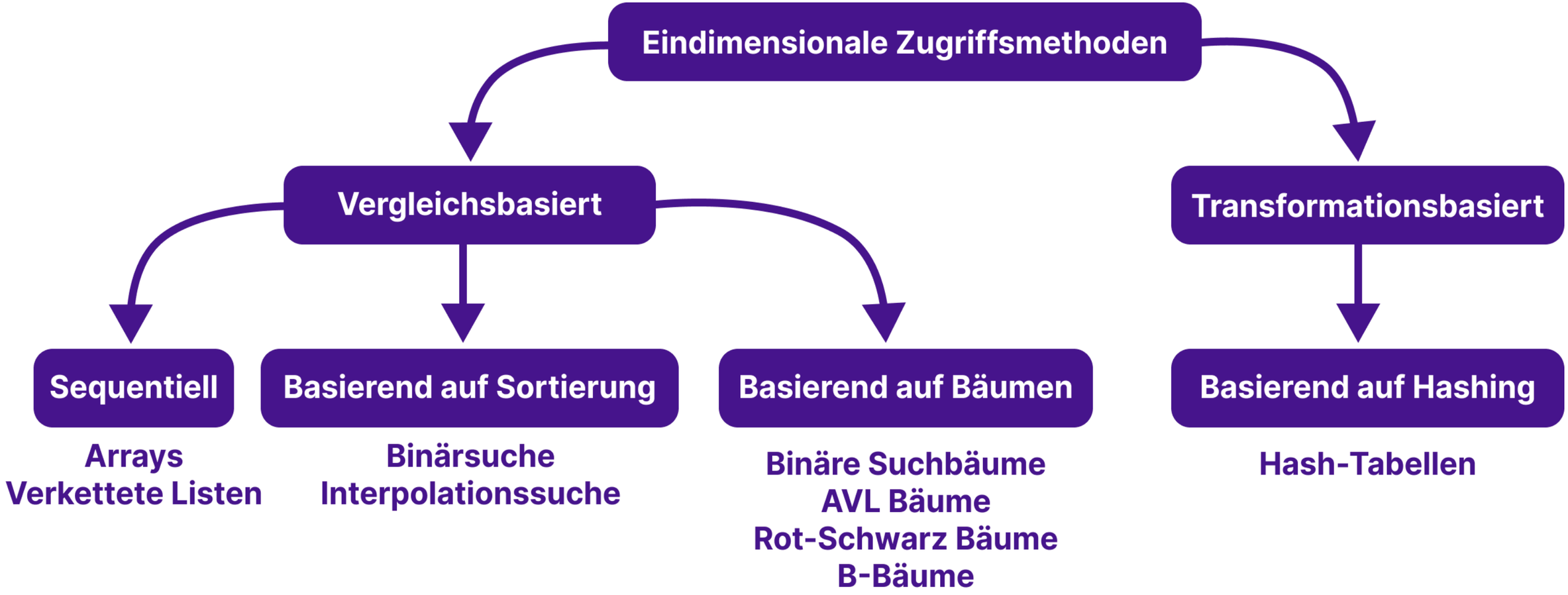
5 16 22 30 40 50 **51** 60 71 77 78 80 97

Refresher: Binärsuche

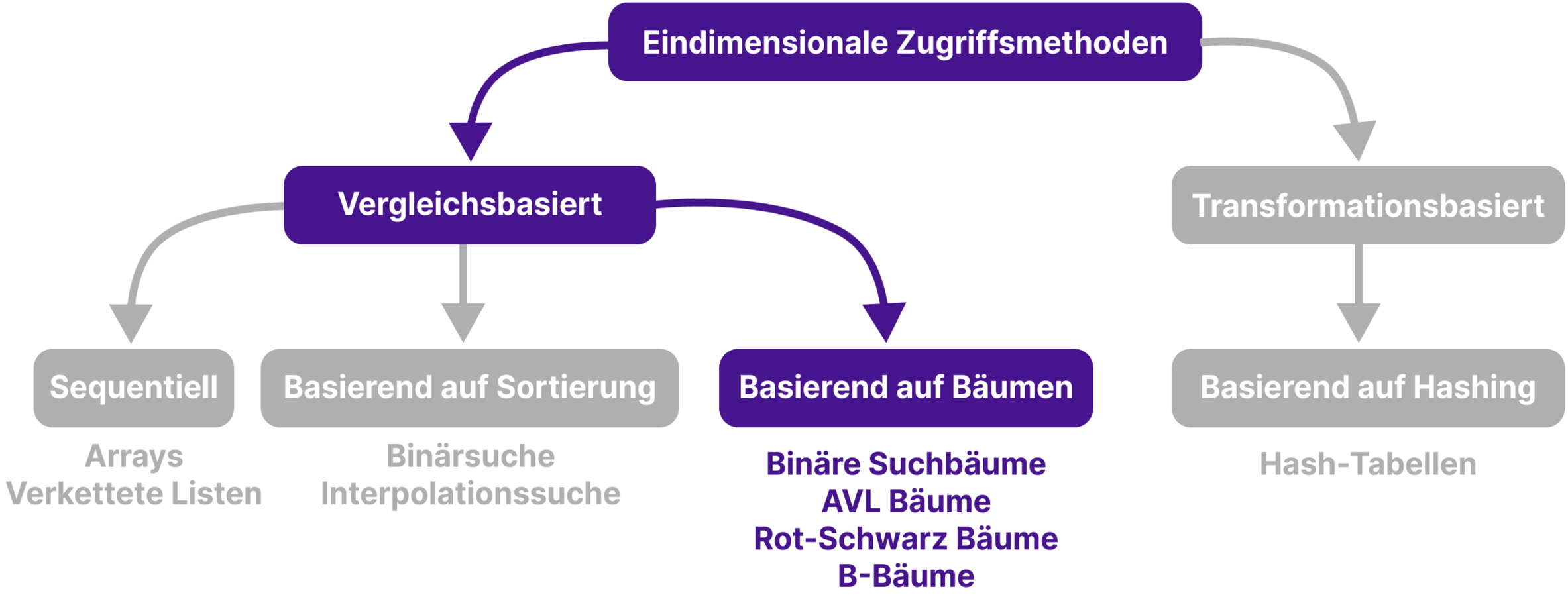
Das wiederholen wir nun rekursiv auf den Sublisten bis wir 78 finden



Zugriffsmethoden im Überblick



In dieser Vorlesung

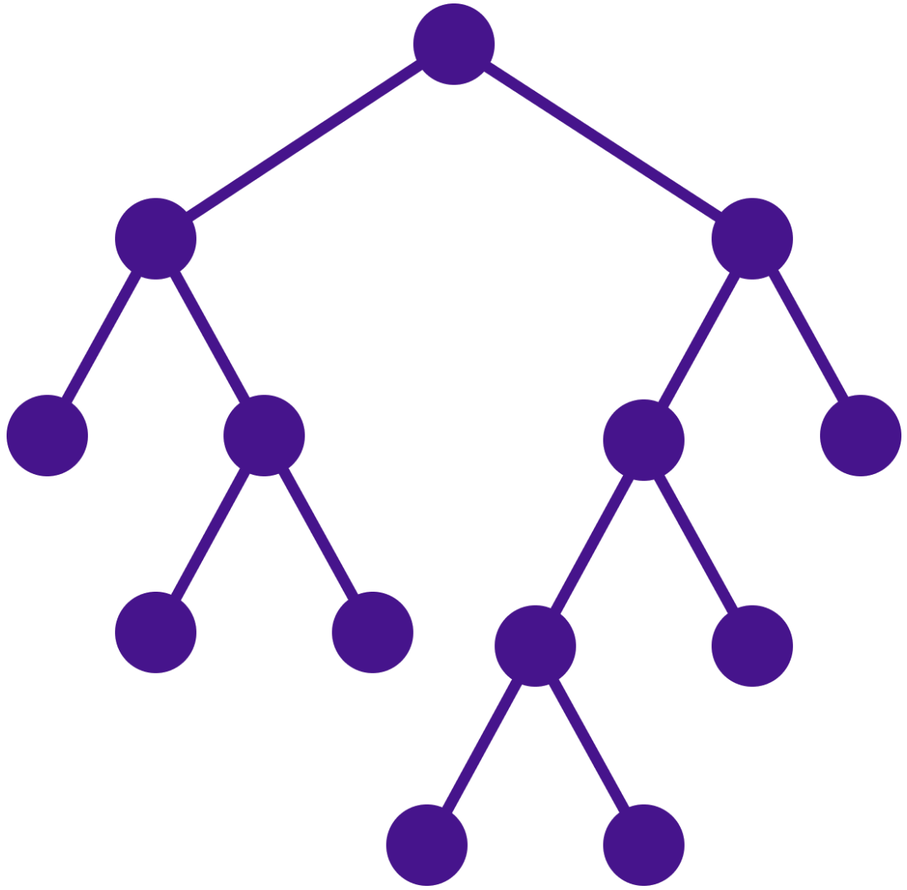




Was sind Bäume?

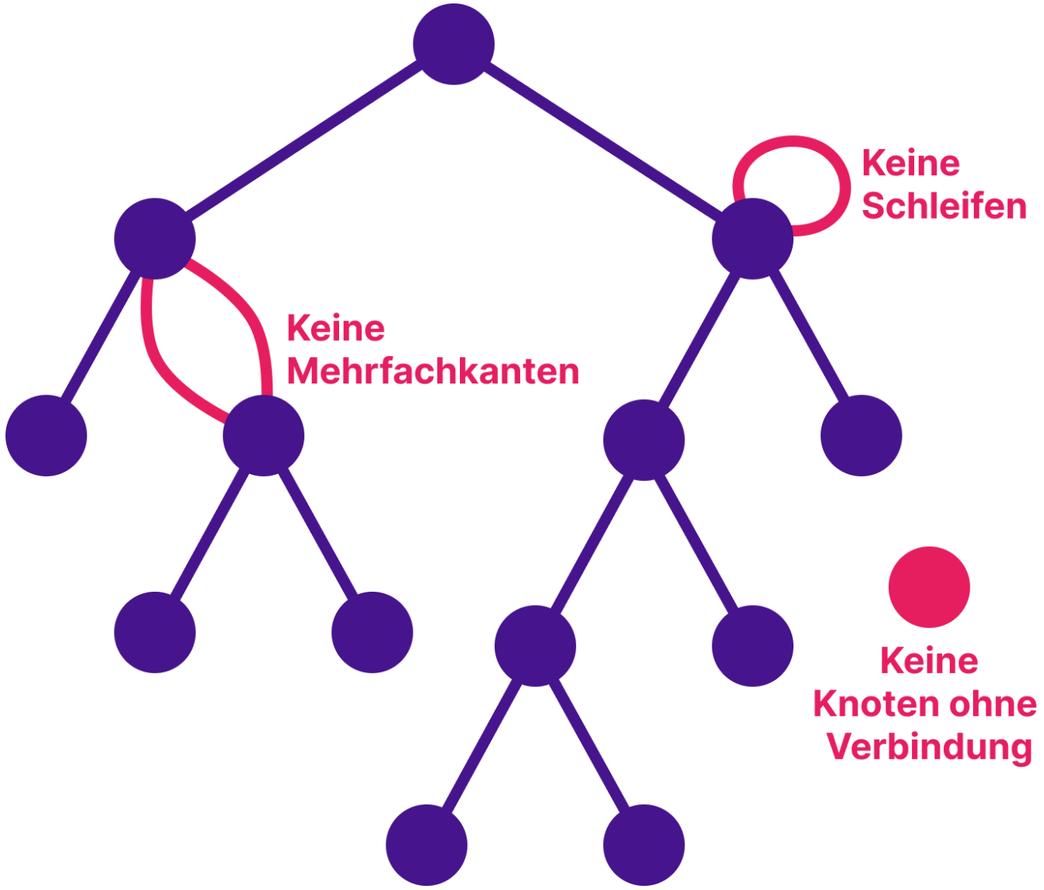
Bäume

Ein Baum T ("tree") ist ein zusammenhängender Graph ohne Kreise.



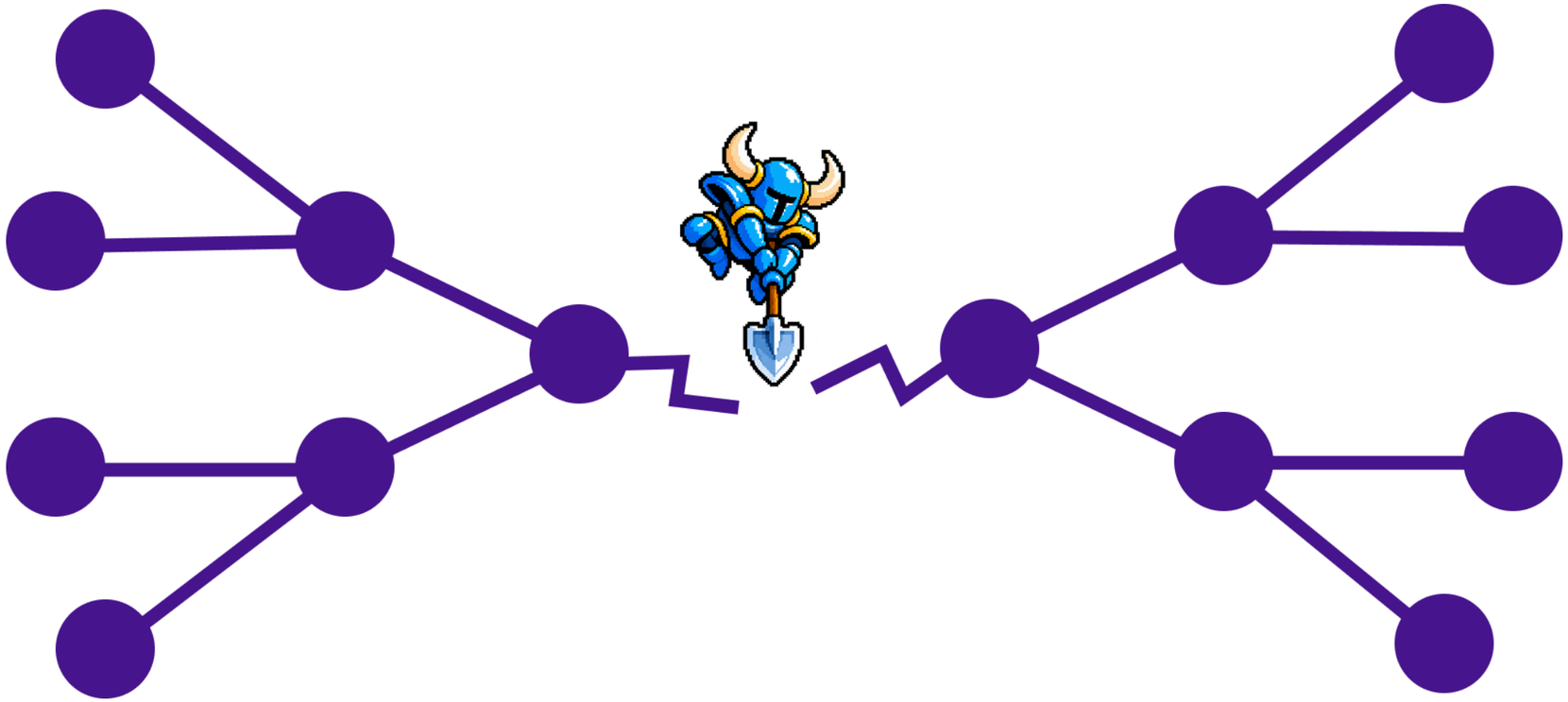
Bedingung 1

T ist zusammenhängend und hat $n - 1$ Kanten.



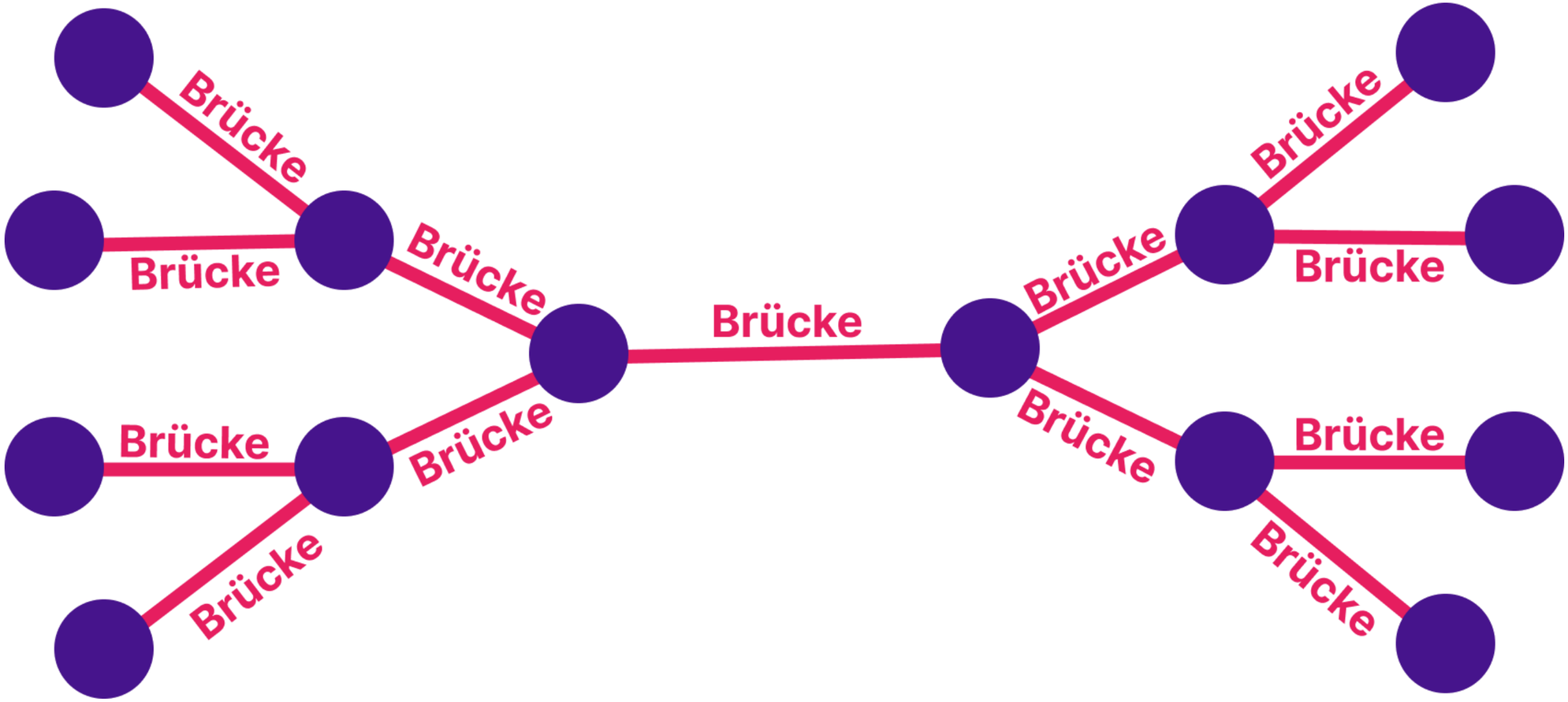
Brücken

Eine Kante e in einem Graphen G heißt Brücke, falls durch das Entfernen von e eine Zusammenhangskomponente von G in zwei Teile zerfällt.



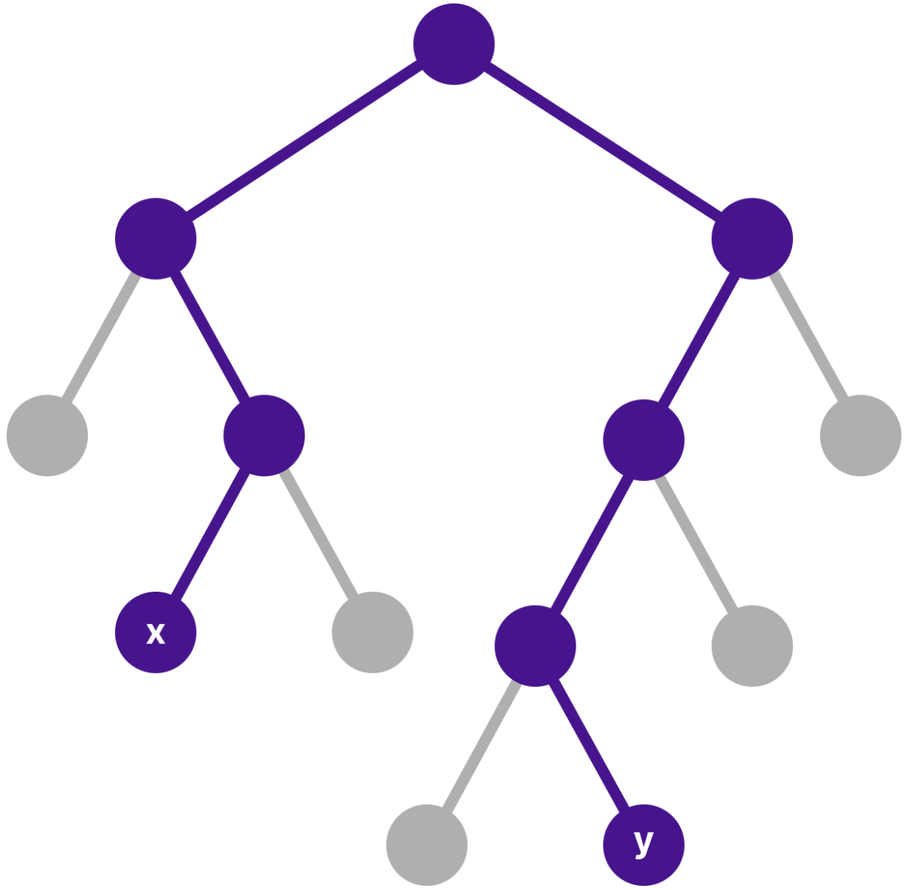
Bedingung 2

T ist zusammenhängend und jede Kante ist eine Brücke.



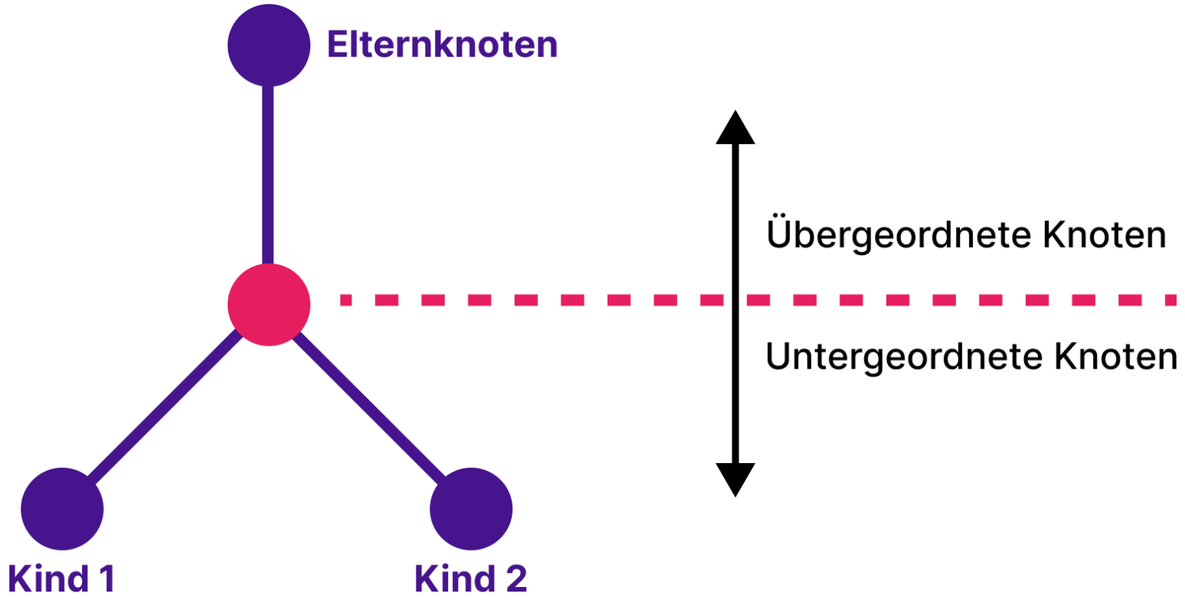
Bedingung 3

Für je zwei Knoten x, y gibt es genau einen Pfad von x nach y.



Bäume bilden Hierarchien ab

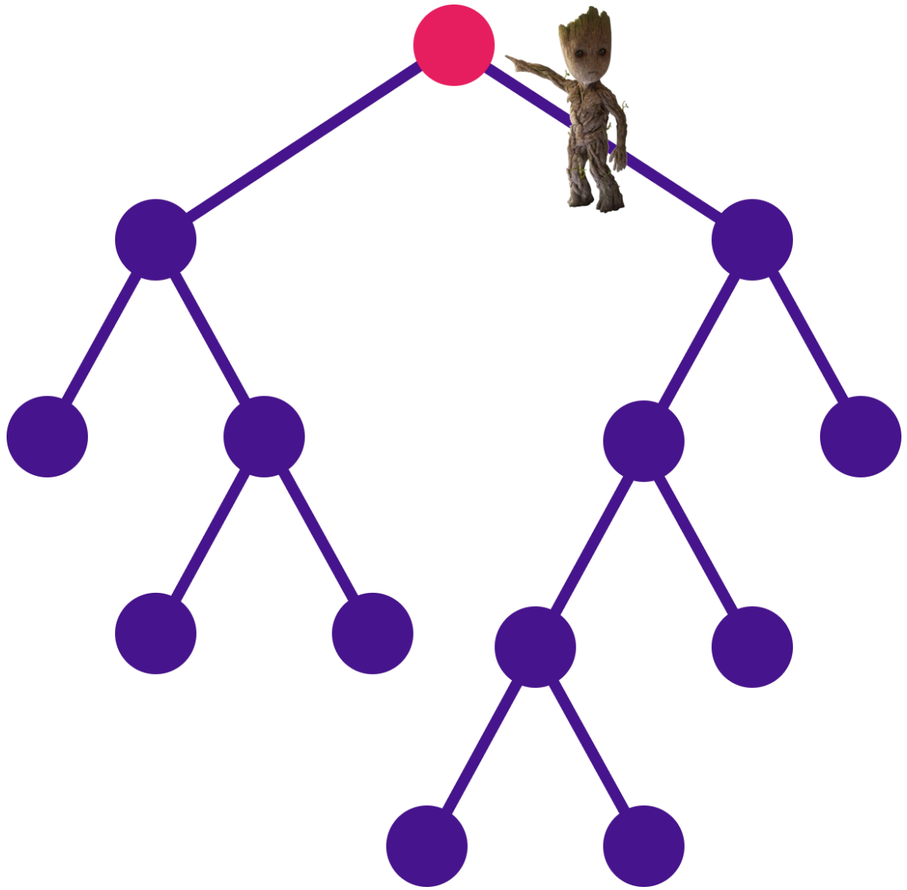
Jeder Knoten besitzt übergeordnete und untergeordnete Knoten,
daraus ergibt sich eine Hierarchie.



Nachfolgend beschränken wir uns auf stets **einen** Elternknoten pro Knoten.

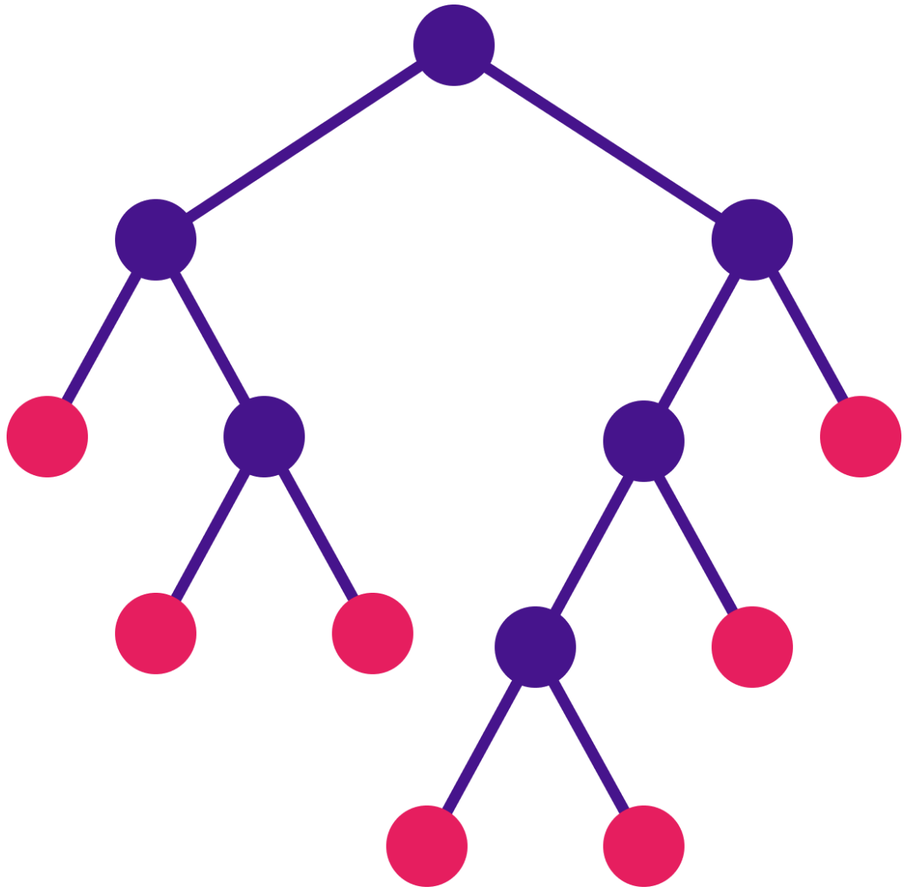
Wurzelknoten

Den Knoten ohne Elternknoten nennen wir **Wurzel**.



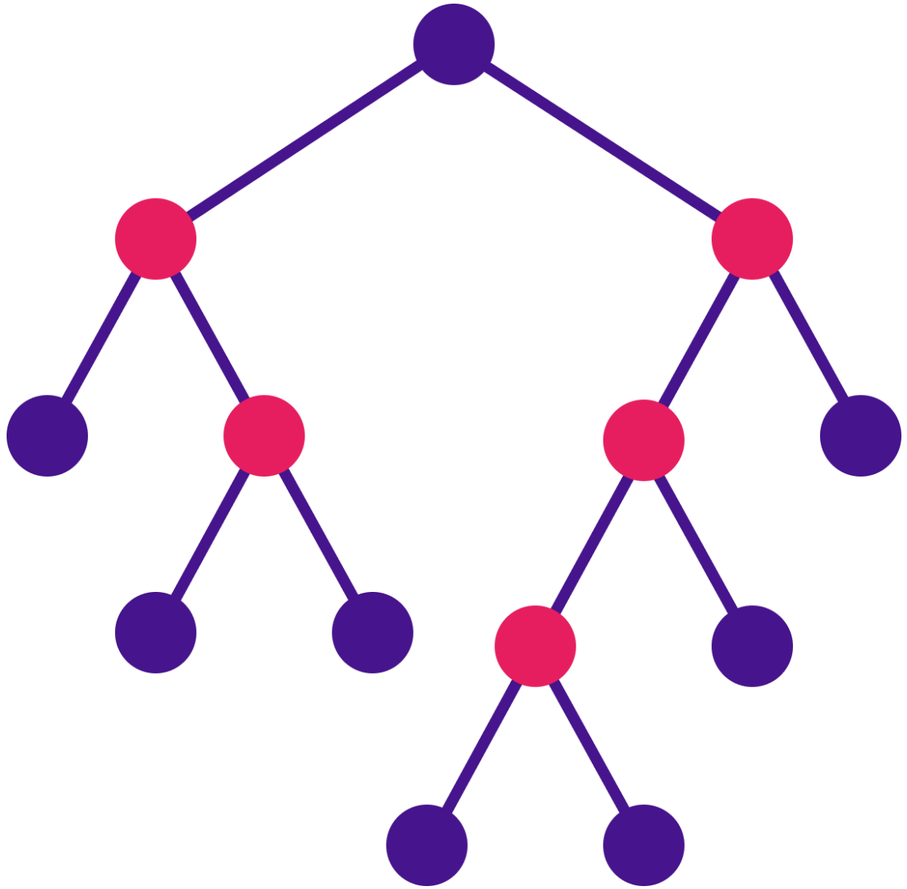
Blätter

Knoten ohne Kinder nennen wir **Blätter**.



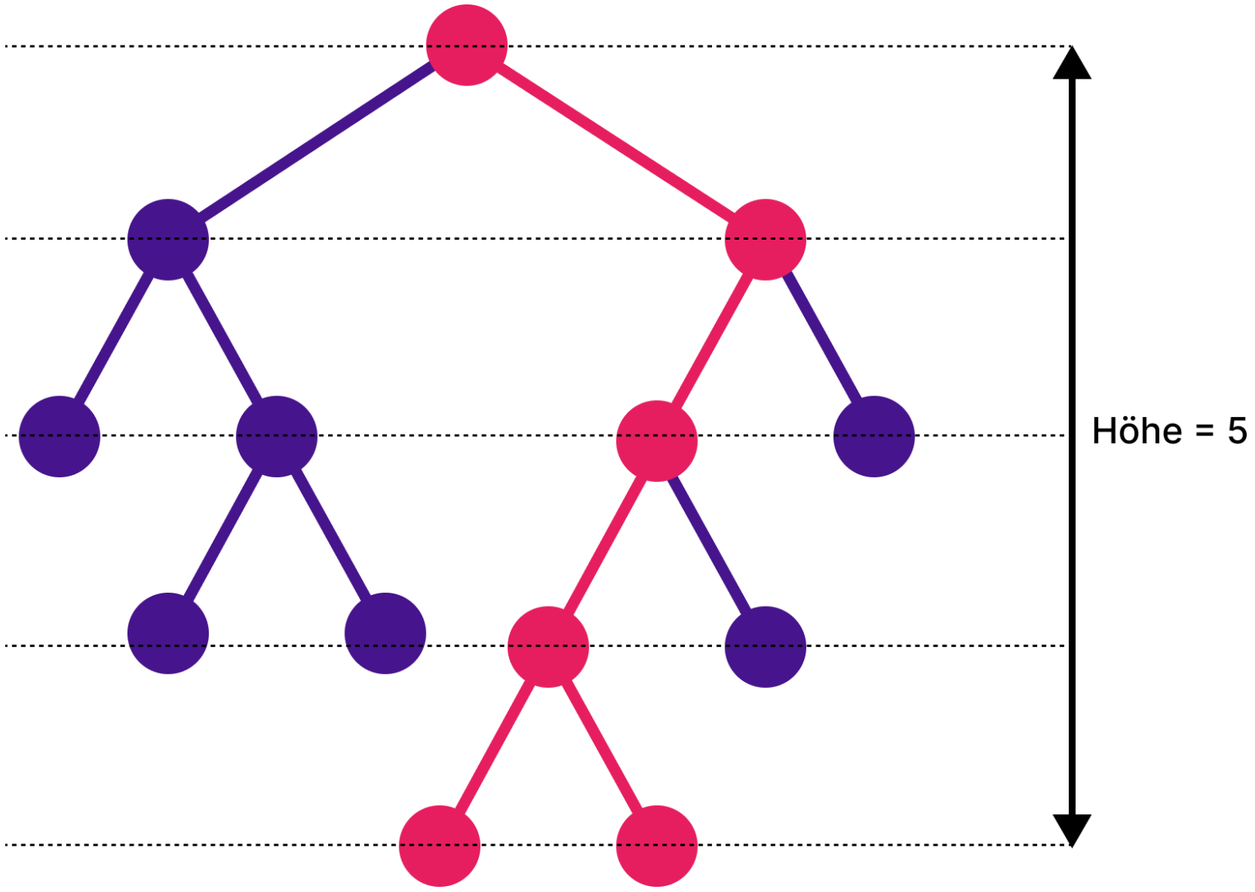
Innere Knoten

Knoten, die sowohl einen Elternknoten als mindestens ein Kind haben, nennen wir **innere Knoten**.



Höhe des Baumes

Die Höhe h ergibt sich aus der Länge des längstmöglichen Pfades ausgehend von der Wurzel.

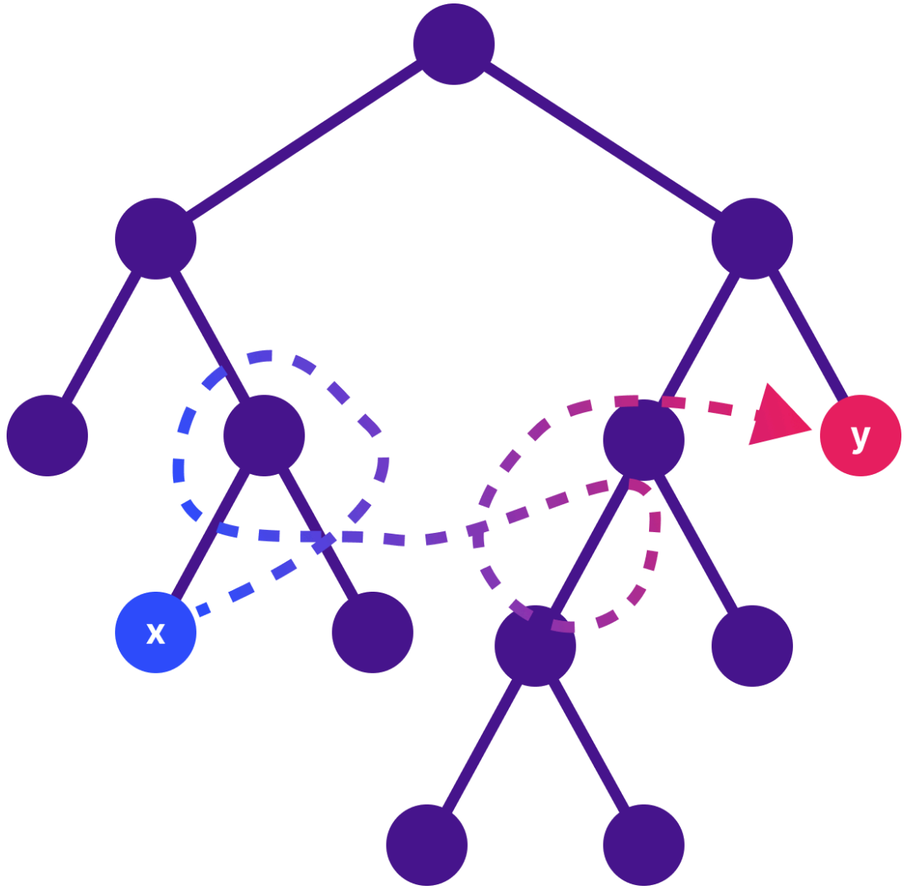


PathFinder

Wie findet man einen Pfad von x nach y?

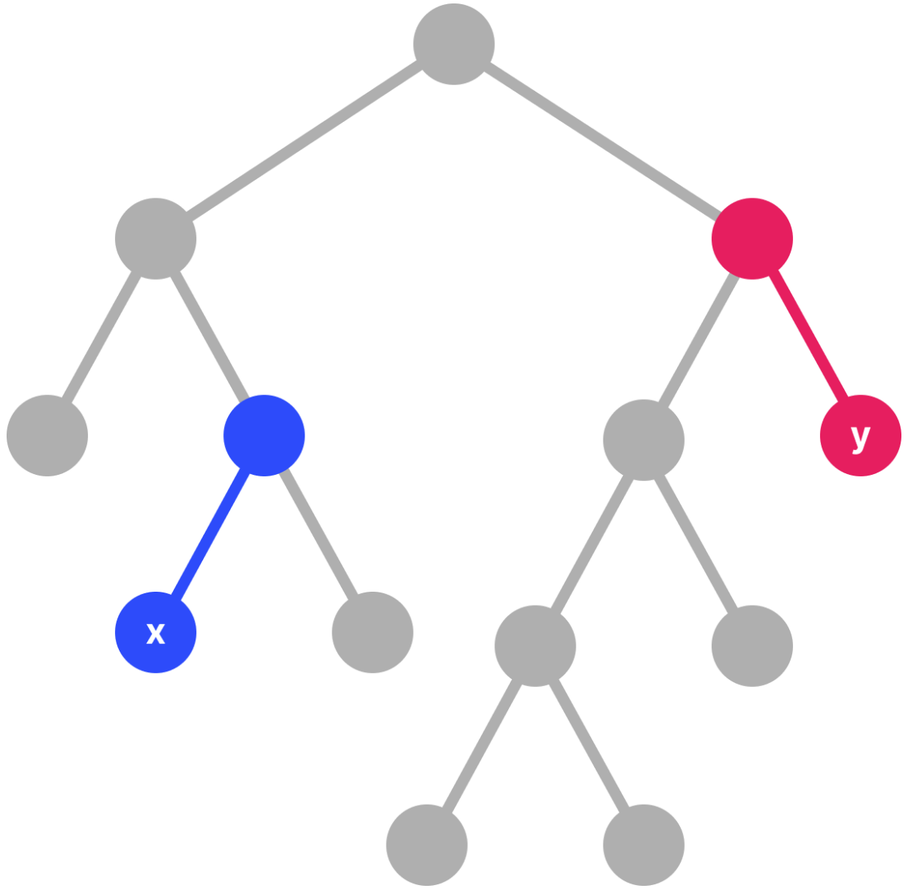
Algorithmus: PathFinder

Wir wollen den Pfad von Knoten x nach y finden:



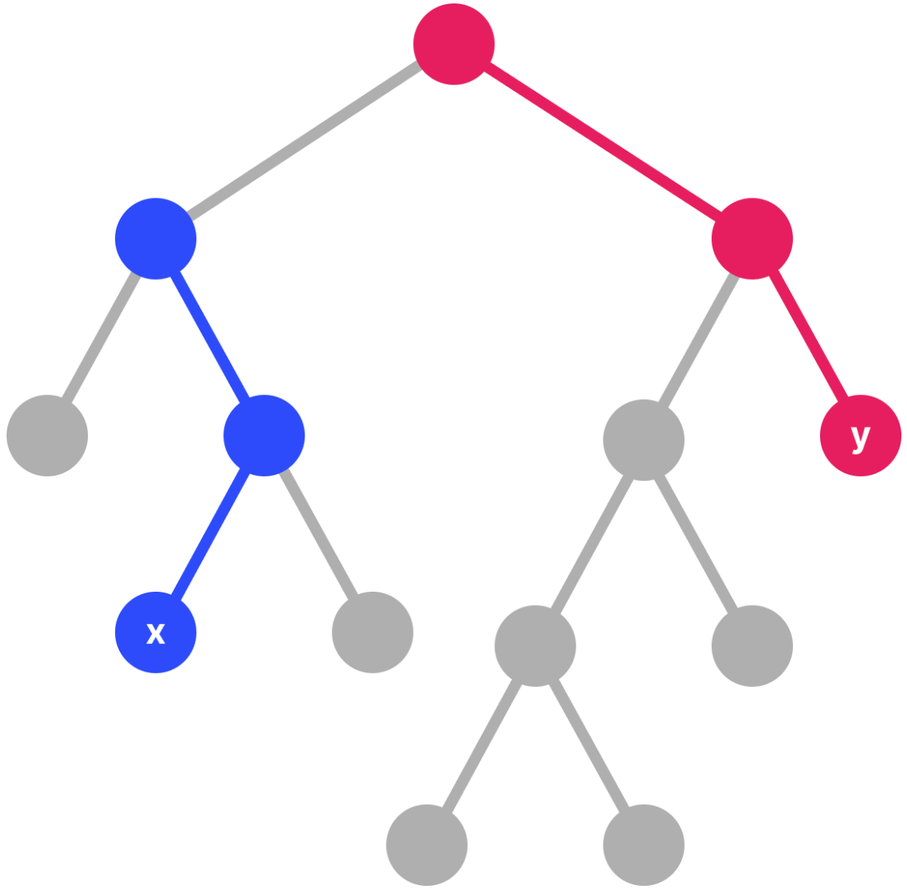
Algorithmus: PathFinder

Wir gehen bei x und y immer jeweils über den Elternknoten in der Hierarchie nach oben.



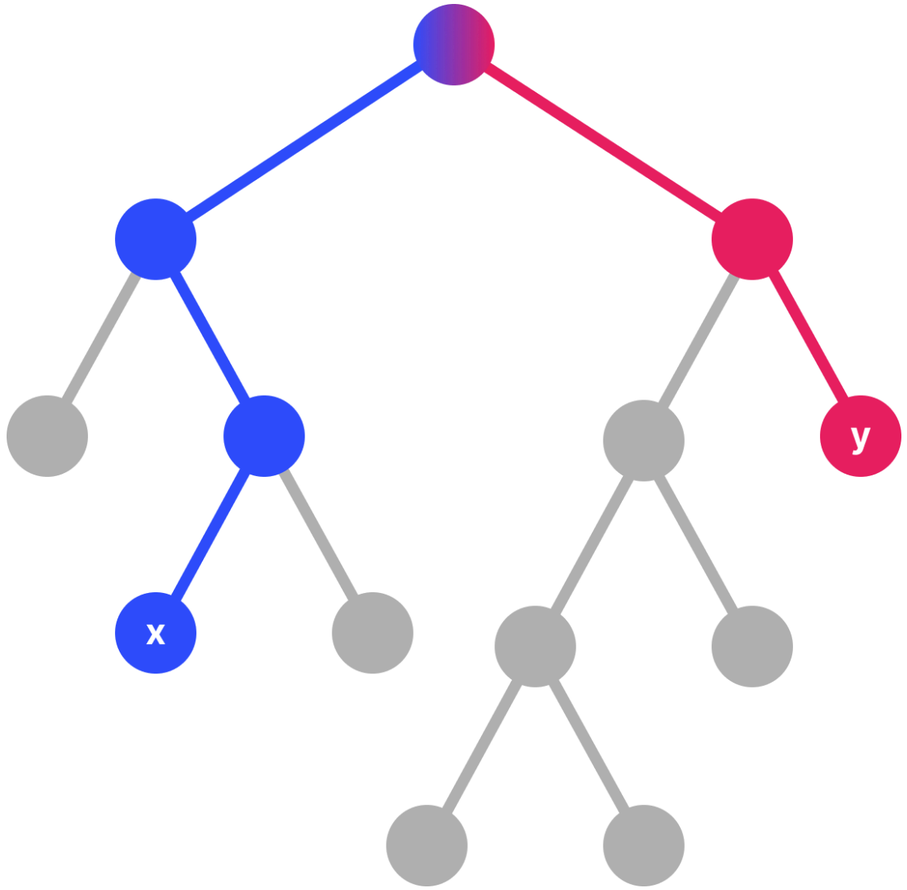
Algorithmus: PathFinder

Die Pfade von x und y stoppen sobald die Wurzel getroffen wird.



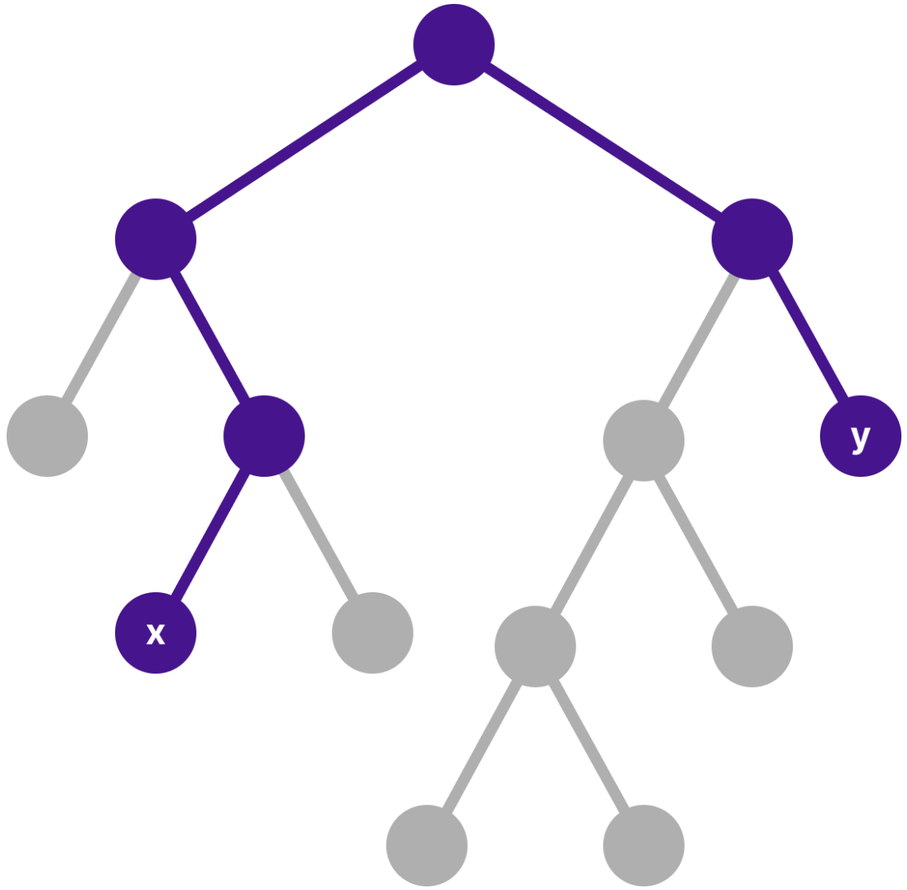
Algorithmus: PathFinder

Die Pfade münden spätestens in der Wurzel.



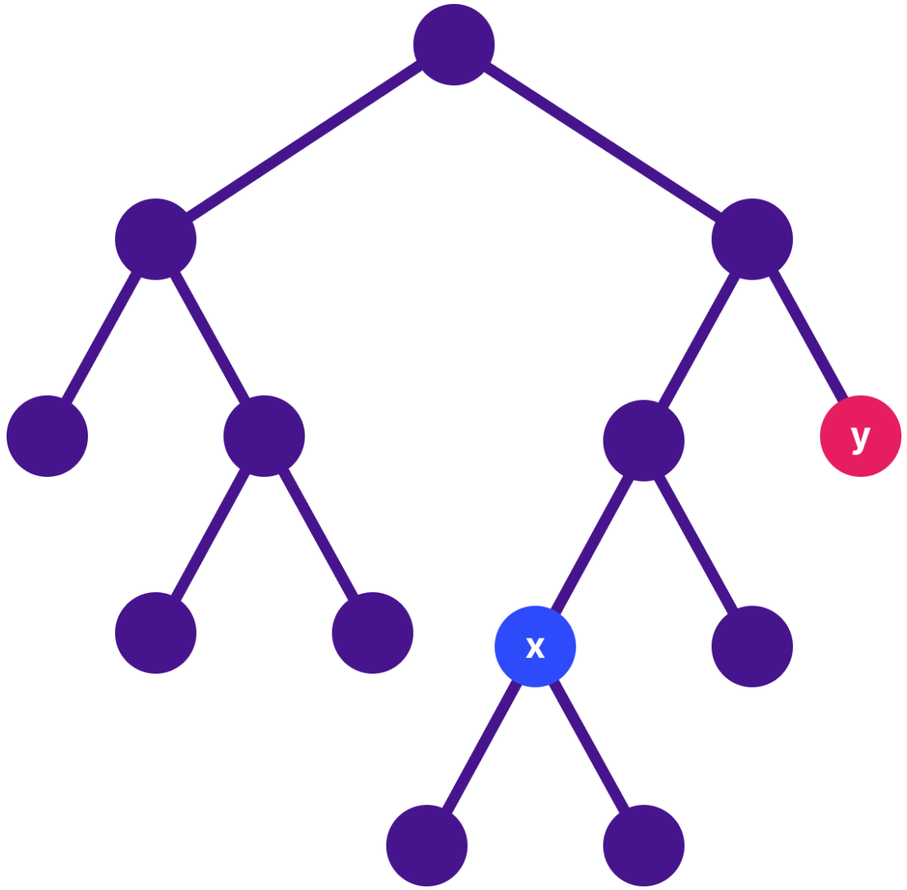
Algorithmus: PathFinder

Zuletzt werden beide Pfade zusammengeführt.



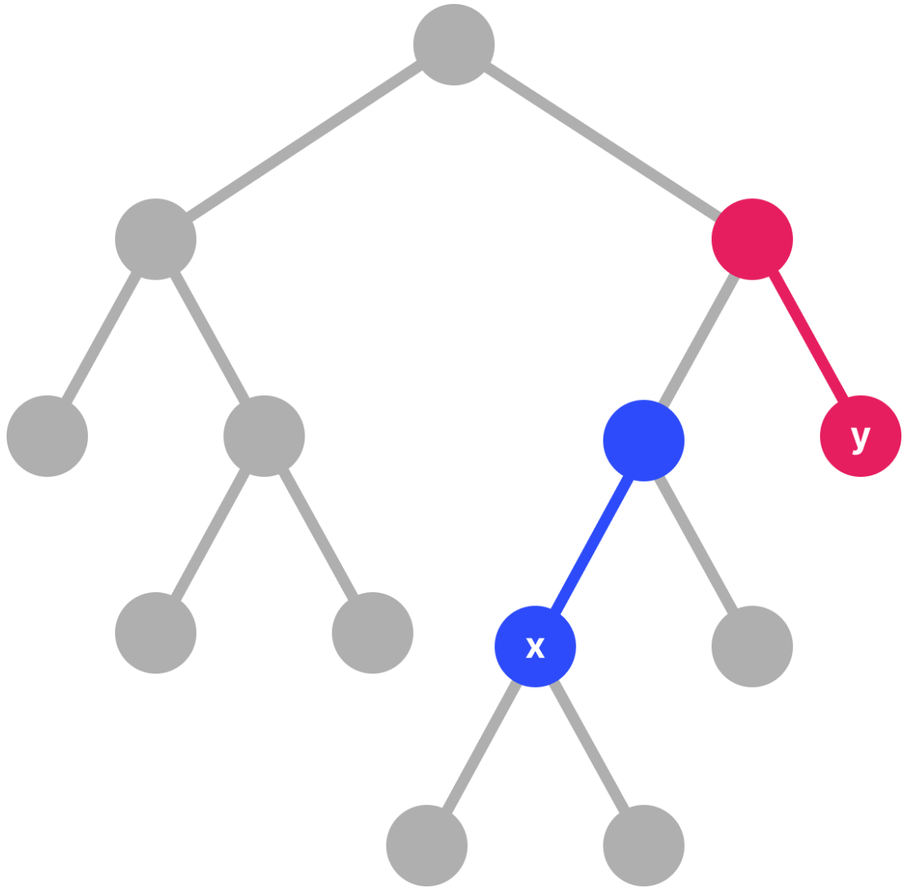
Algorithmus: PathFinder

Was tun bei Pfaden, die nicht über die Wurzel gehen?



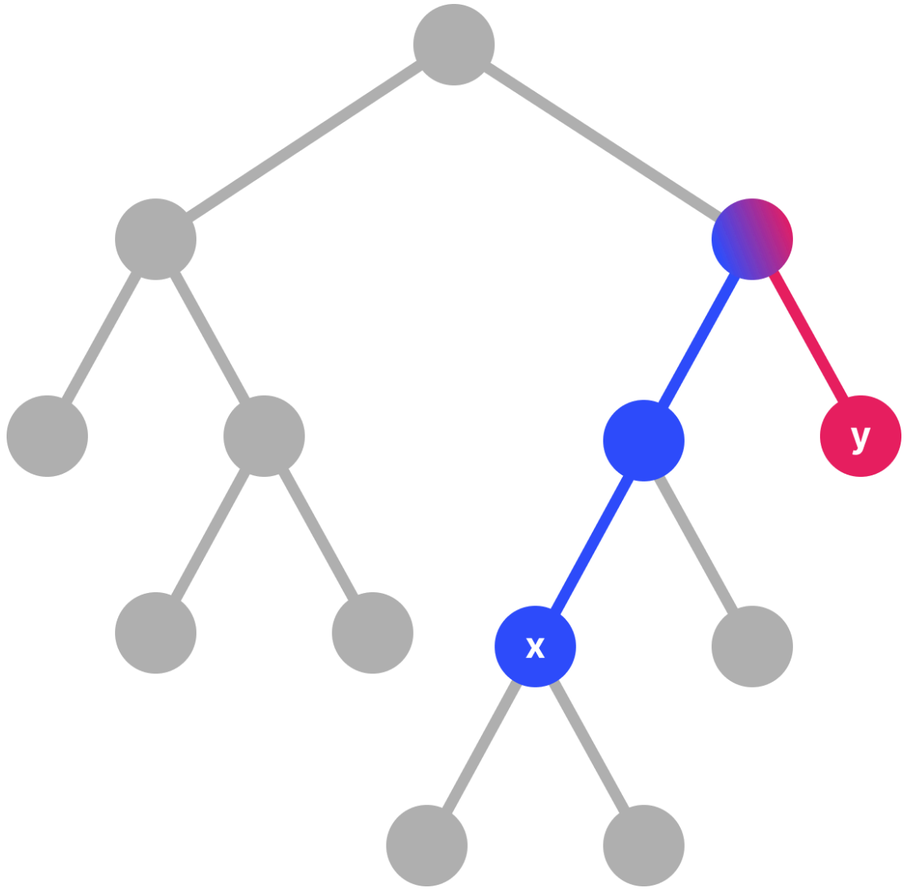
Algorithmus: PathFinder

Selbe Schritte wie vorhin: Man geht von beiden Knoten den Baum nach oben.



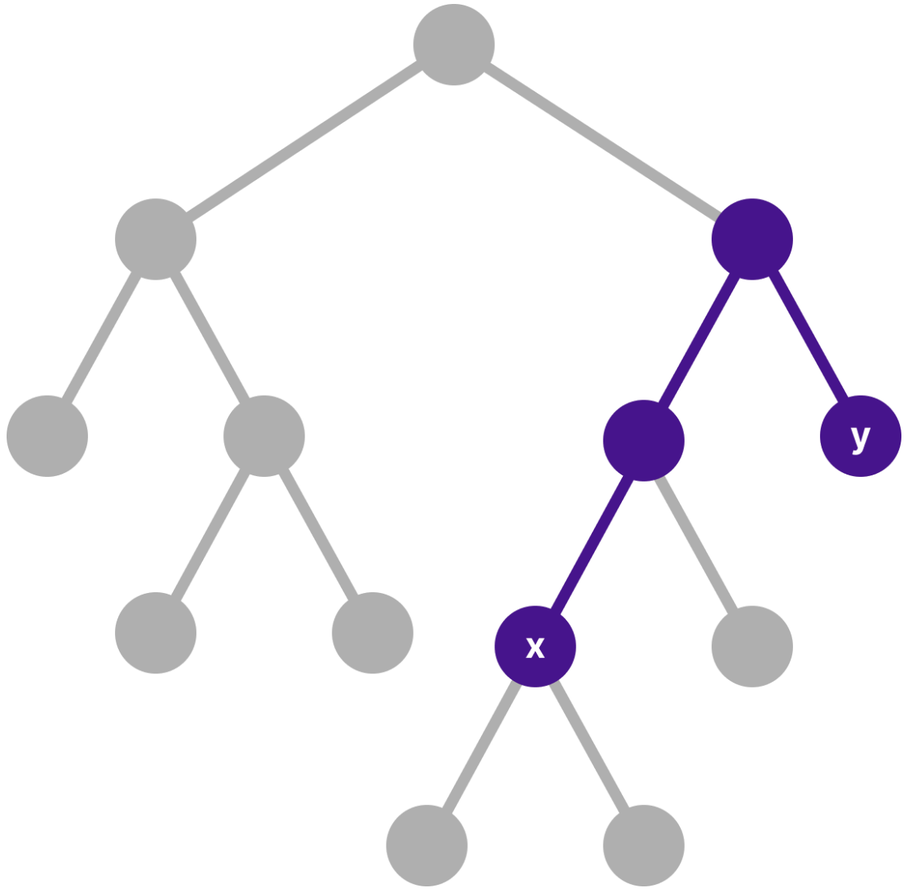
Algorithmus: PathFinder

Sobald die Pfade auf den gleichen Knoten treffen, stoppt der Algorithmus.



Algorithmus: PathFinder

Endergebnis



Algorithmus: PathFinder

PATHFINDER(x, y)

$\nu^k(x)$ bezeichnet den k-ten Vorfahren von x.

1. Wähle $x_0 \in V$ als Wurzel
2. $P_x \leftarrow \left(x, \nu^1(x), \nu^2(x), \ldots, \nu^{h(x)}(x)\right)$
3. $P_y \leftarrow \left(y, \nu^1(y), \nu^2(y), \ldots, \nu^{h(y)}(y)\right)$
4. $r \leftarrow h(x) - h(y)$
5. Bestimme kleinstes j mit $\nu^{j+r}(x) = \nu^j(y)$
6. **return** $\left(x, \nu^1(x), \ldots, \nu^{j+r}(x), \nu^{j-1}(y), \ldots, \nu^1(y), y\right)$

Wie schnell ist PathFinder?

Im worst case müssen Sie vom tiefsten Blatt bis zur Wurzel gehen.

Das ist $\mathcal{O}(h)$, wobei h die Höhe des Baumes ist.

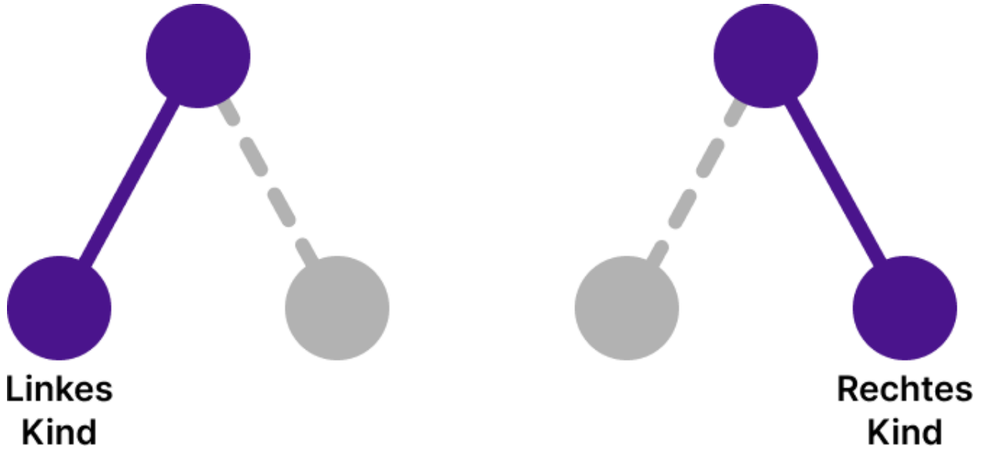
Algorithmen auf Bäumen sind grundsätzlich von der Höhe des Baumes abhängig.

Binäre Suchbäume

Was ist ein Binärbaum?

Jeder Knoten hat **maximal zwei Kinder**.

Man sagt:

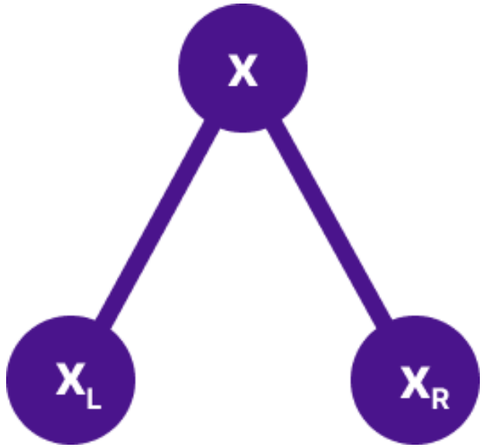


Knoten dürfen auch nur ein Kind haben, aber es muss festgelegt werden, ob der Knoten links oder rechts ist. Grund: Suchbäume.

Vom Binärbaum zum binären Suchbaum

Jeder Knoten trägt nun ein Datenelement $x \in \mathbb{R}$

Für die Kinder gilt dabei:

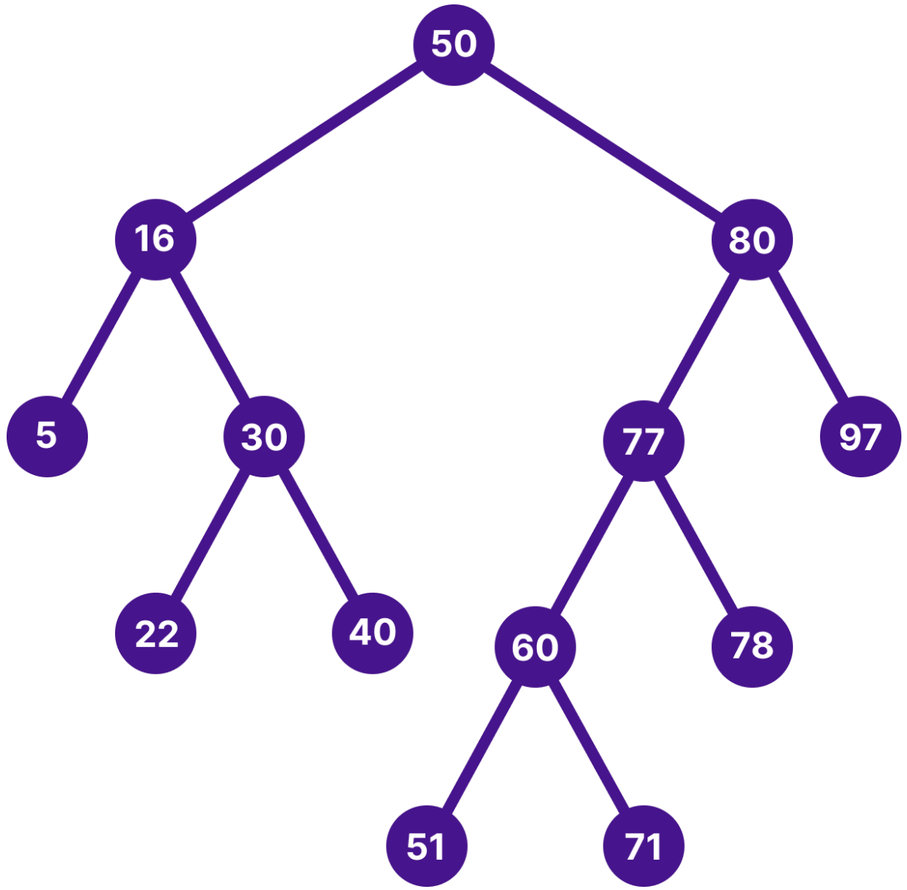


$$x_L < x < x_R$$

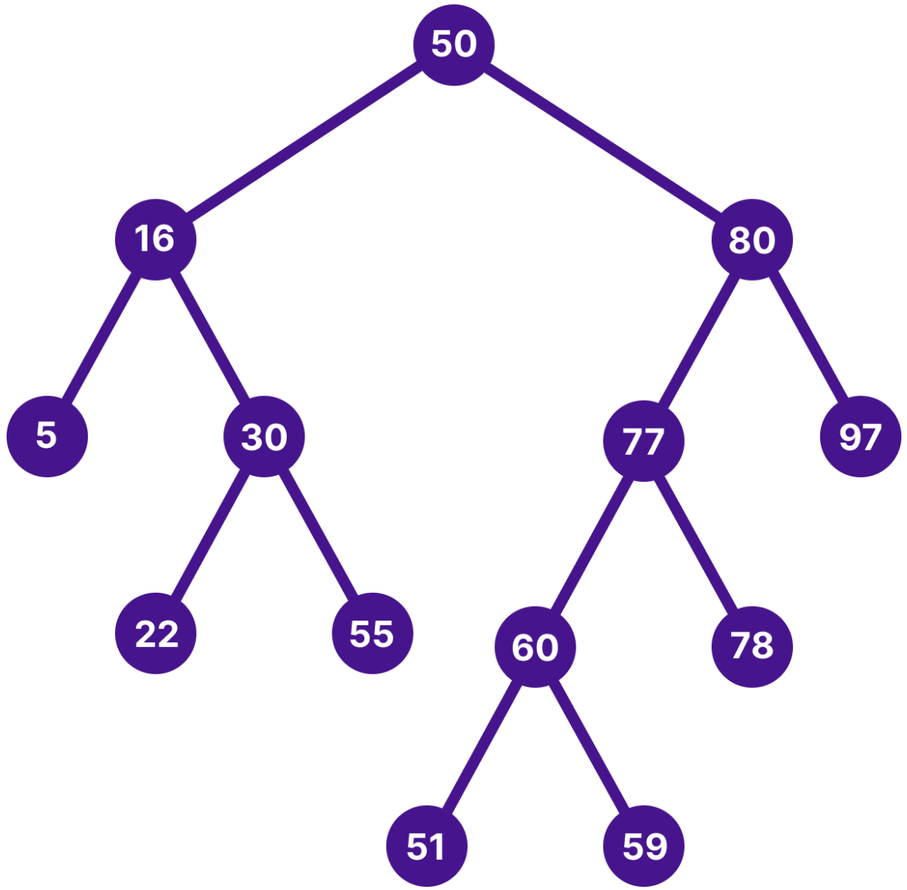
x_L und x_R müssen dabei nicht direkte Nachfolger von x sein, sondern können auch in einem tieferen Teilbaum sein.

Warum kein $\leq$? Dazu kommen wir gleich!

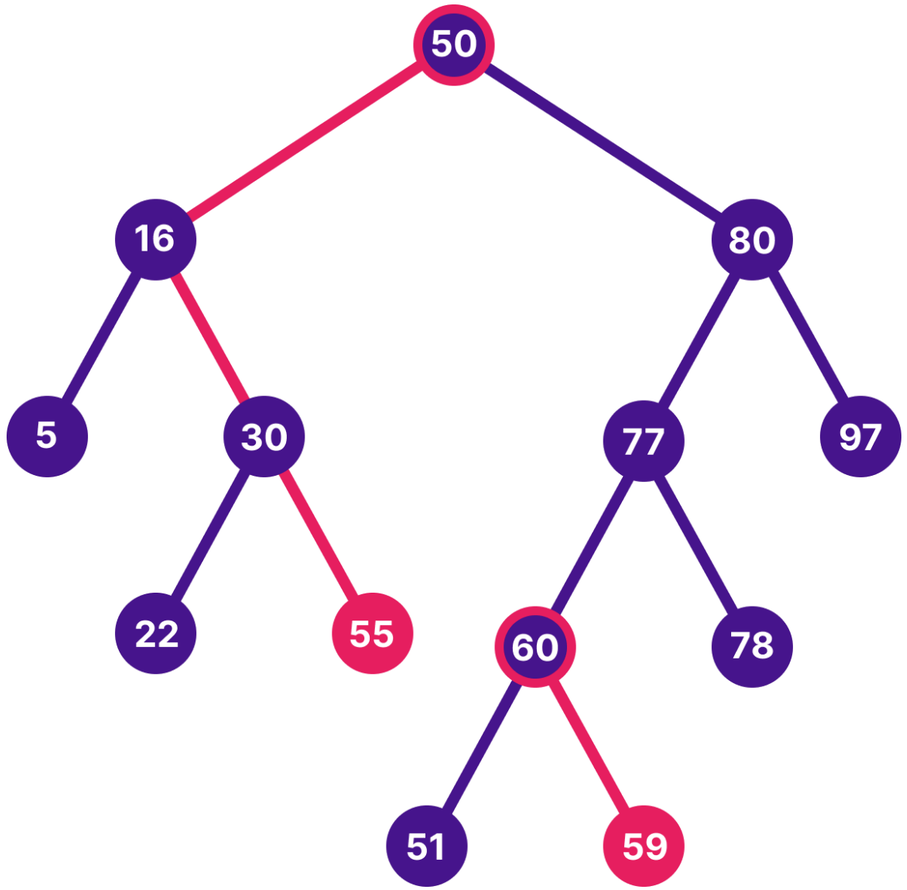
Beispiel



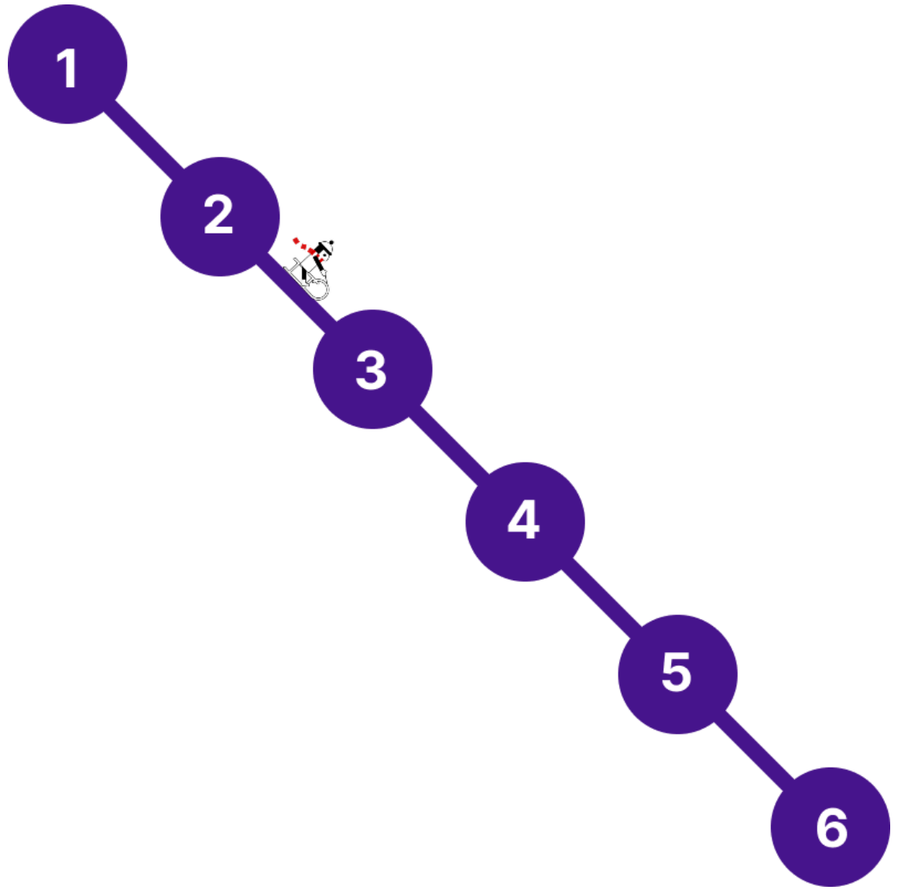
Anderes Beispiel mit Fehler



Auflösung



Auch das ist ein gültiger binärer Suchbaum

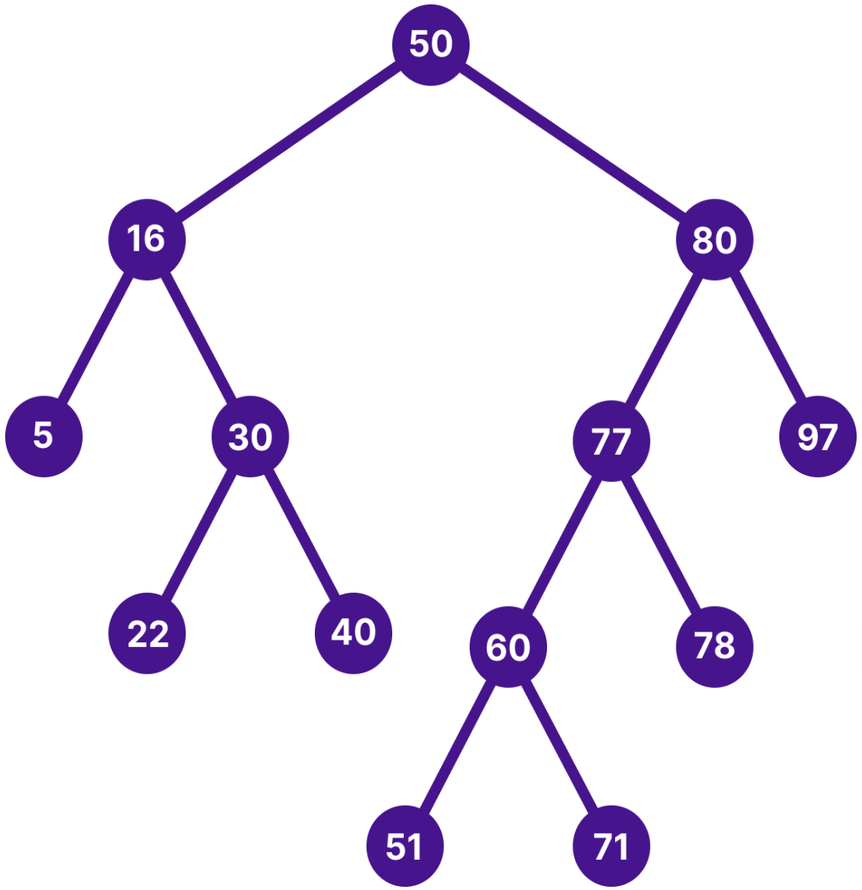


Man nennt diesen Sonderfall "degeneriert".

Tree Traversal

Wie besucht man jeden Knoten im Baum?

Das Problem



Wie wandelt man einen Baum in eine Liste um?

Beispiel: Sortierte Liste aus dem nebenstehenden Baum

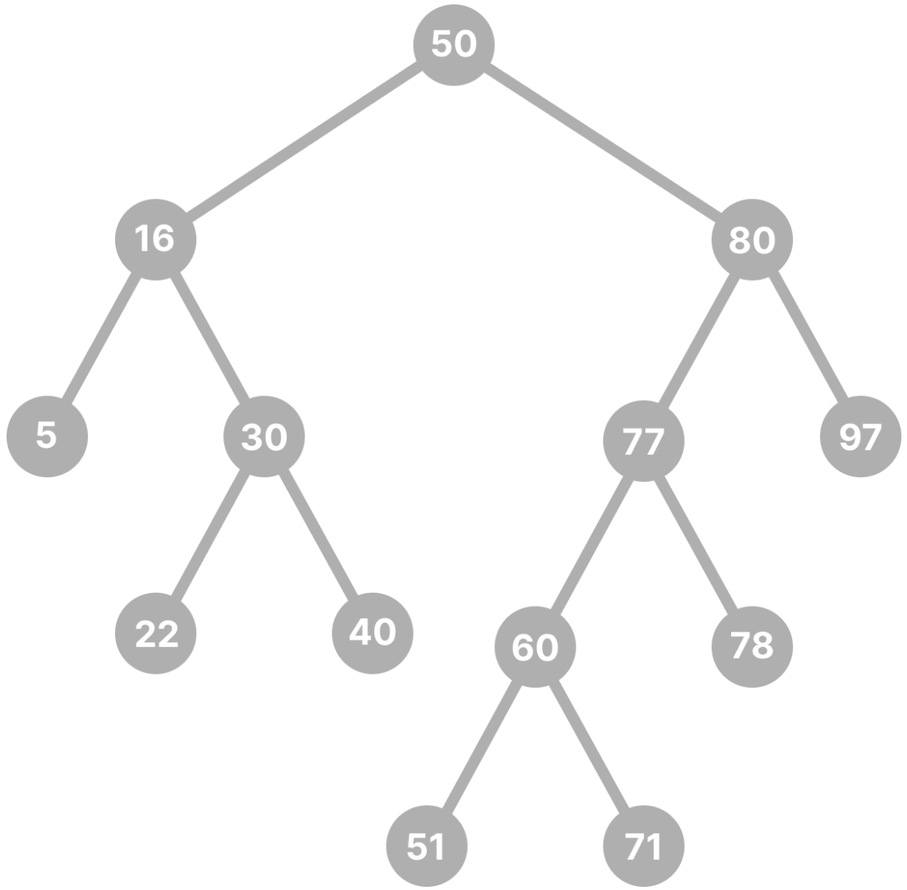
```
5 16 22 30 40 50 51 60 71 77 78 80 97
```

In-Order Durchlauf

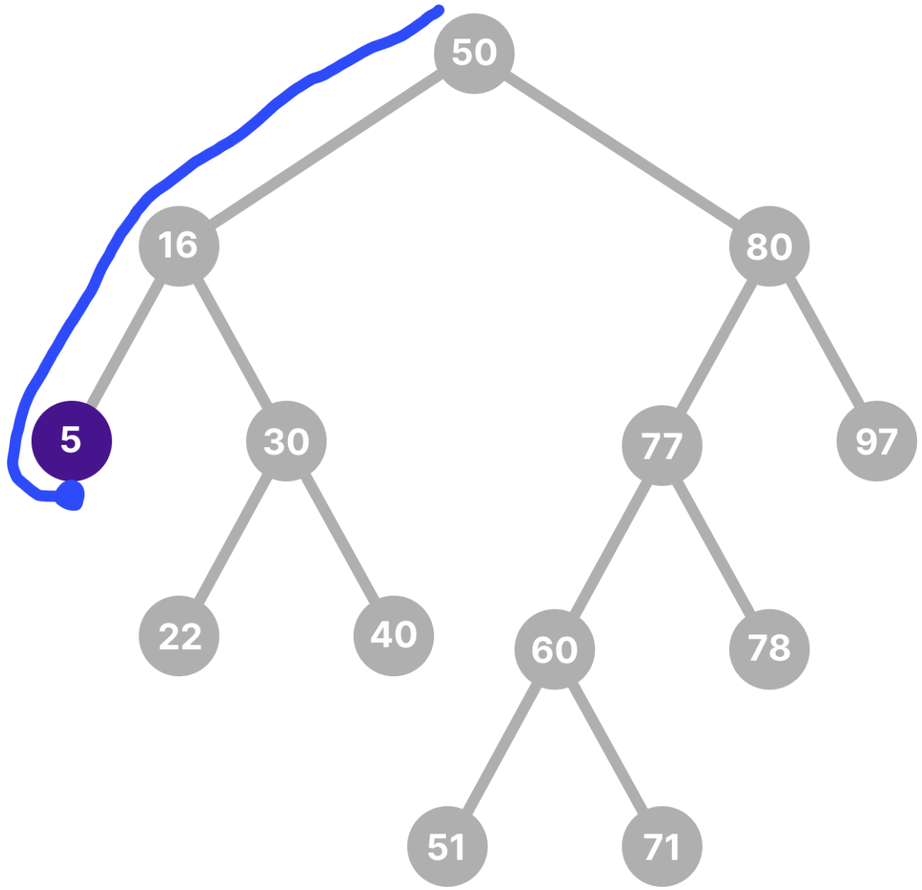
INORDER-TREE-WALK(x)

```
1  if x ≠ NIL
2    INORDER-TREE-WALK(x.left)
3    print x.key
4    INORDER-TREE-WALK(x.right)
```

Inorder

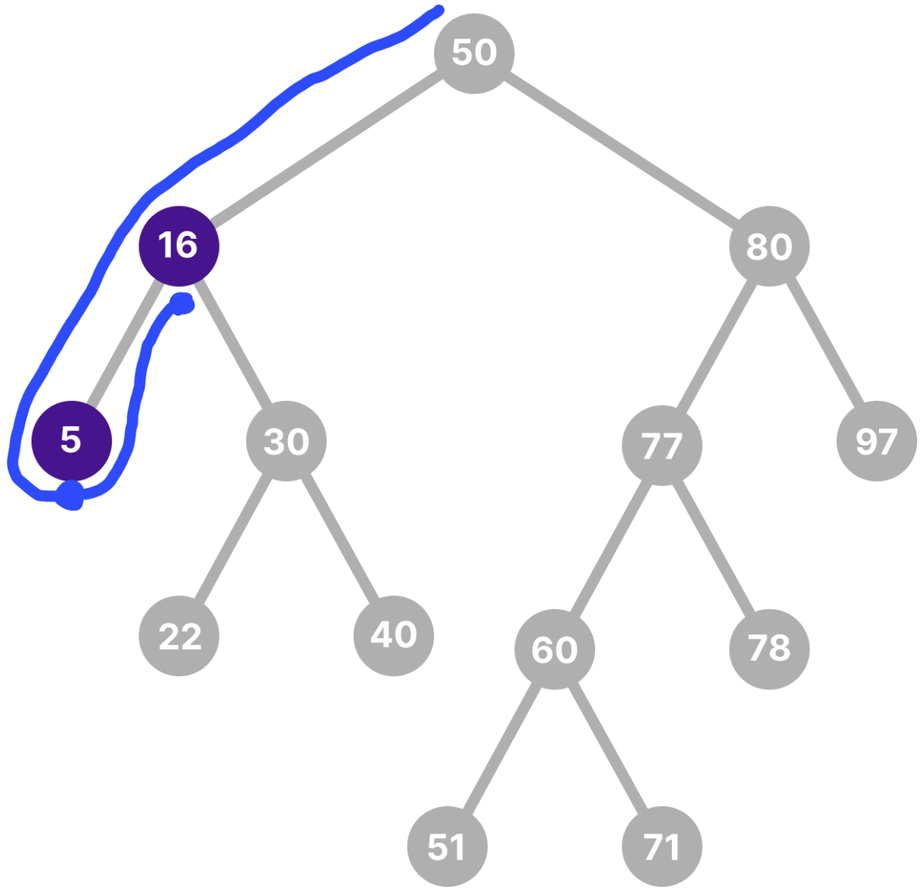


Inorder



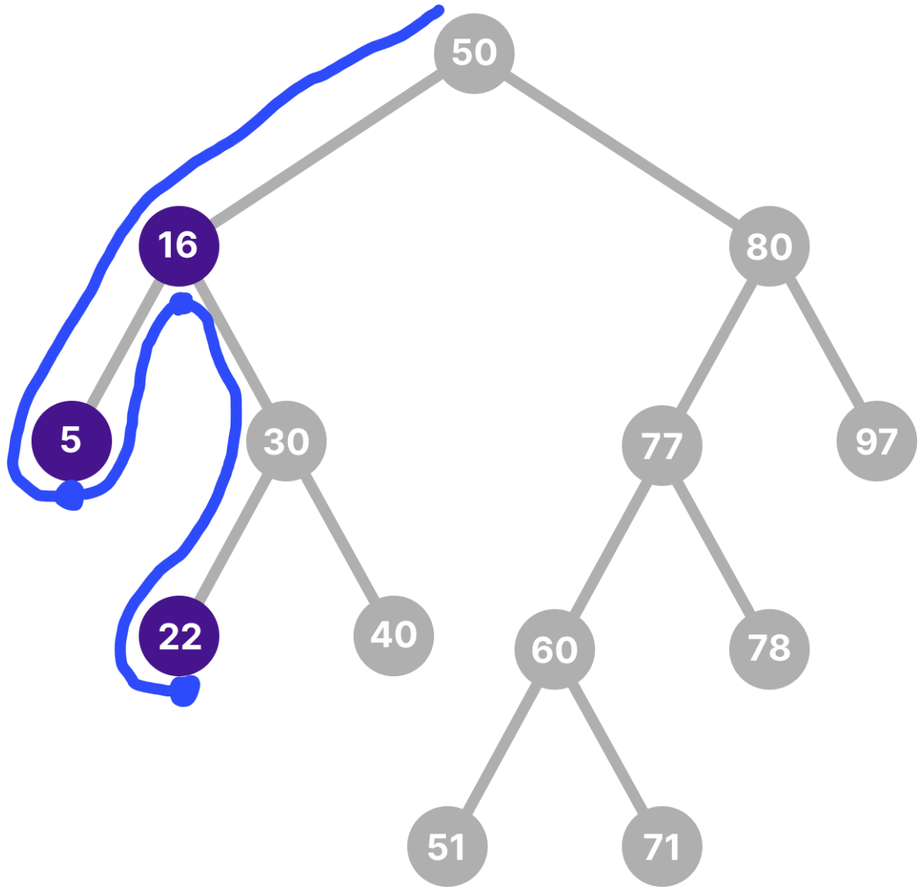
5

Inorder



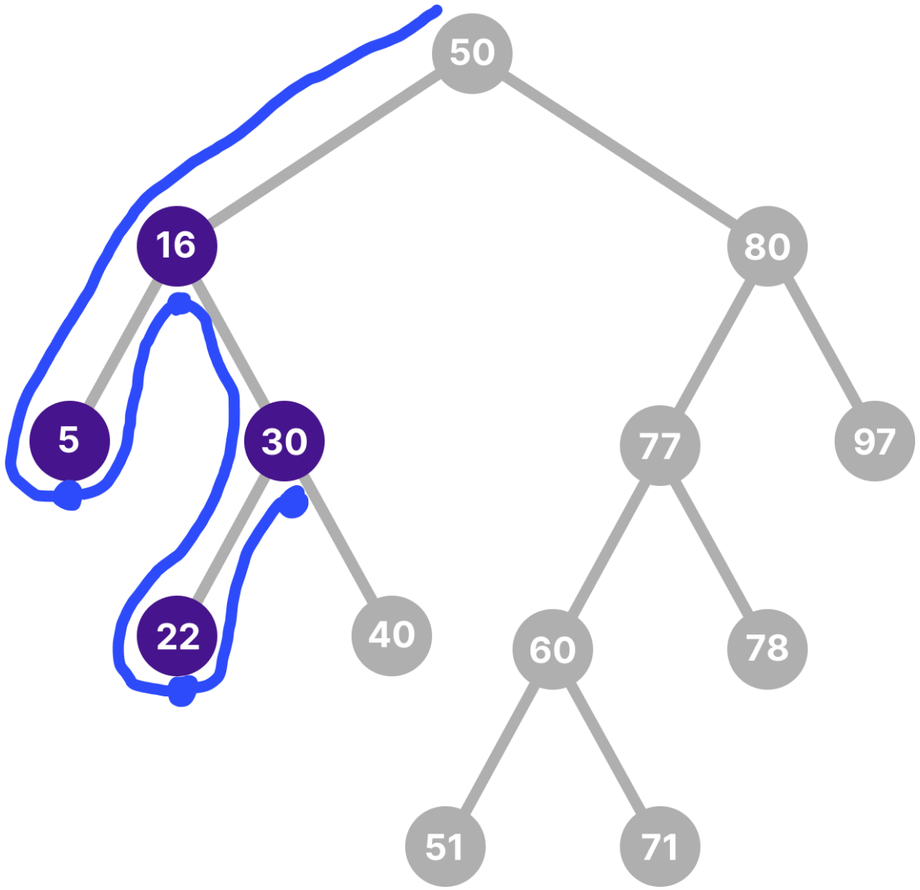
5 16

Inorder



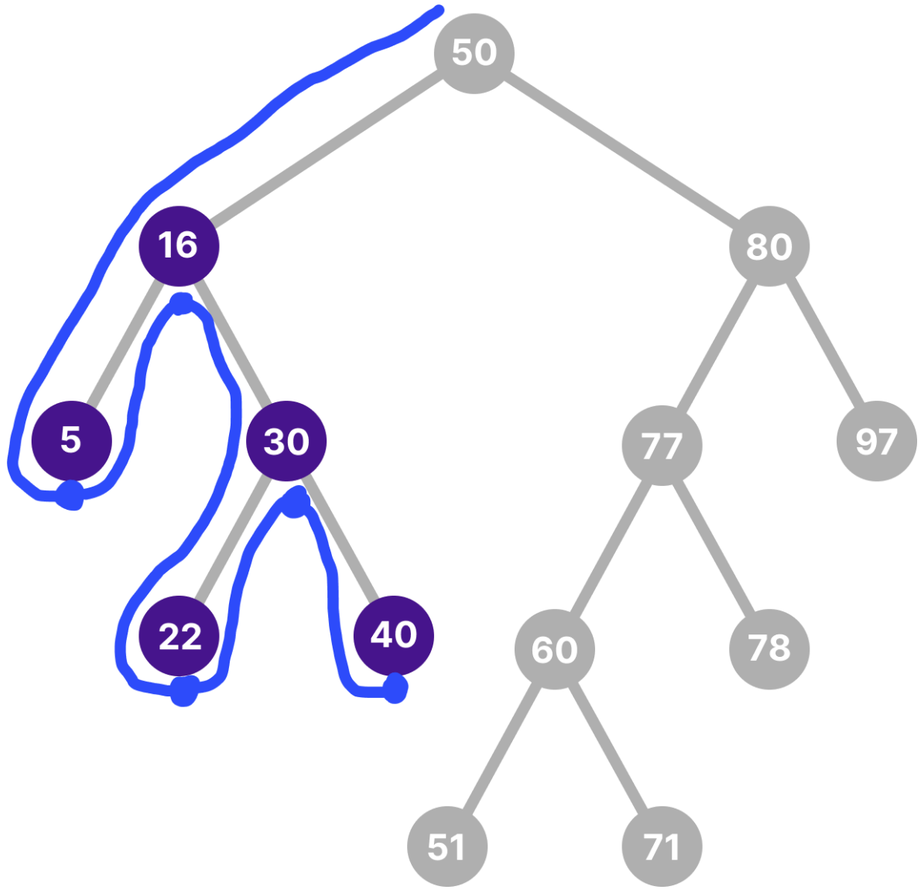
5 16 22

Inorder



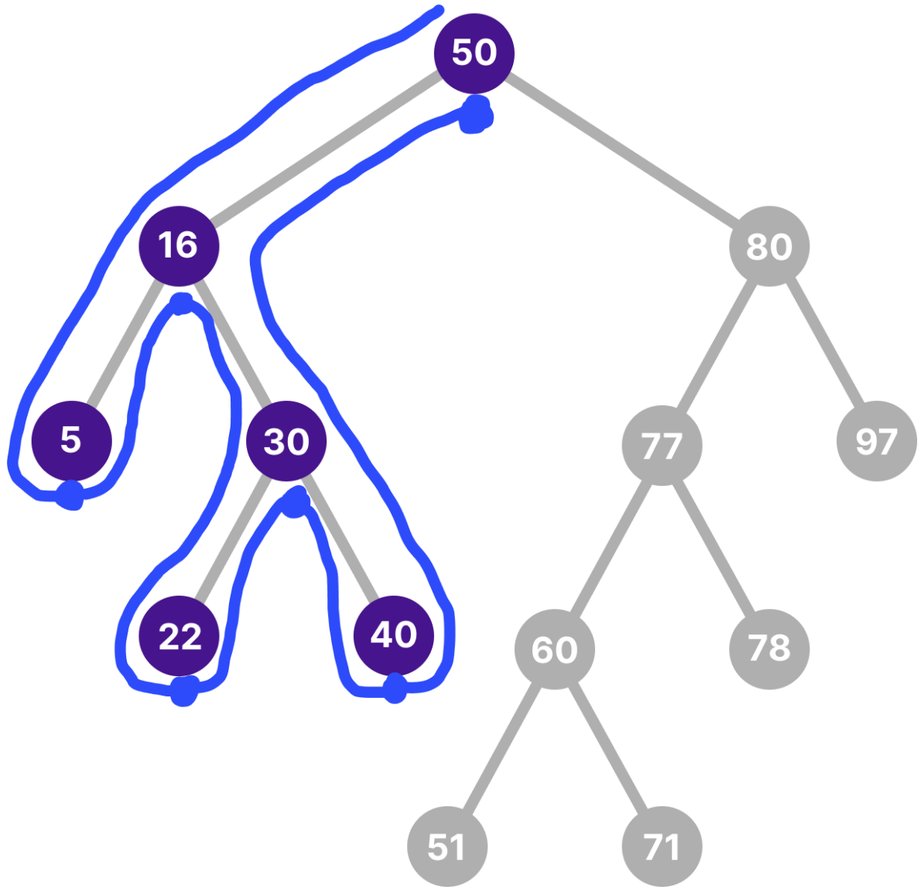
5 16 22 30

Inorder



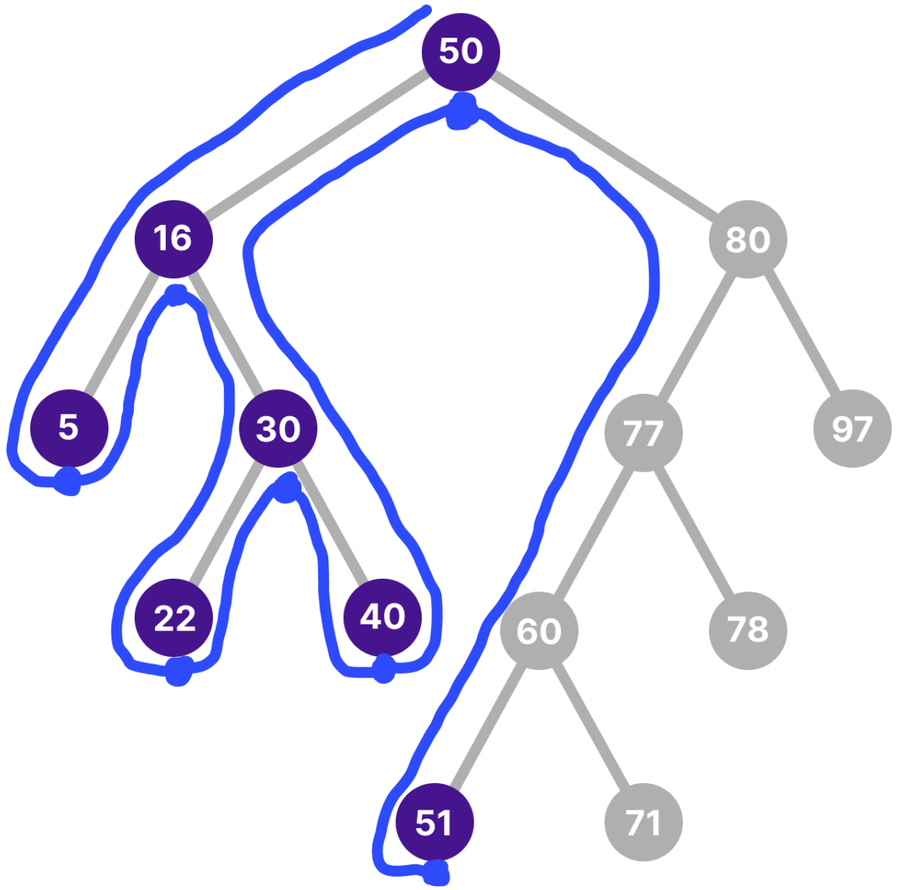
5 16 22 30 40

Inorder



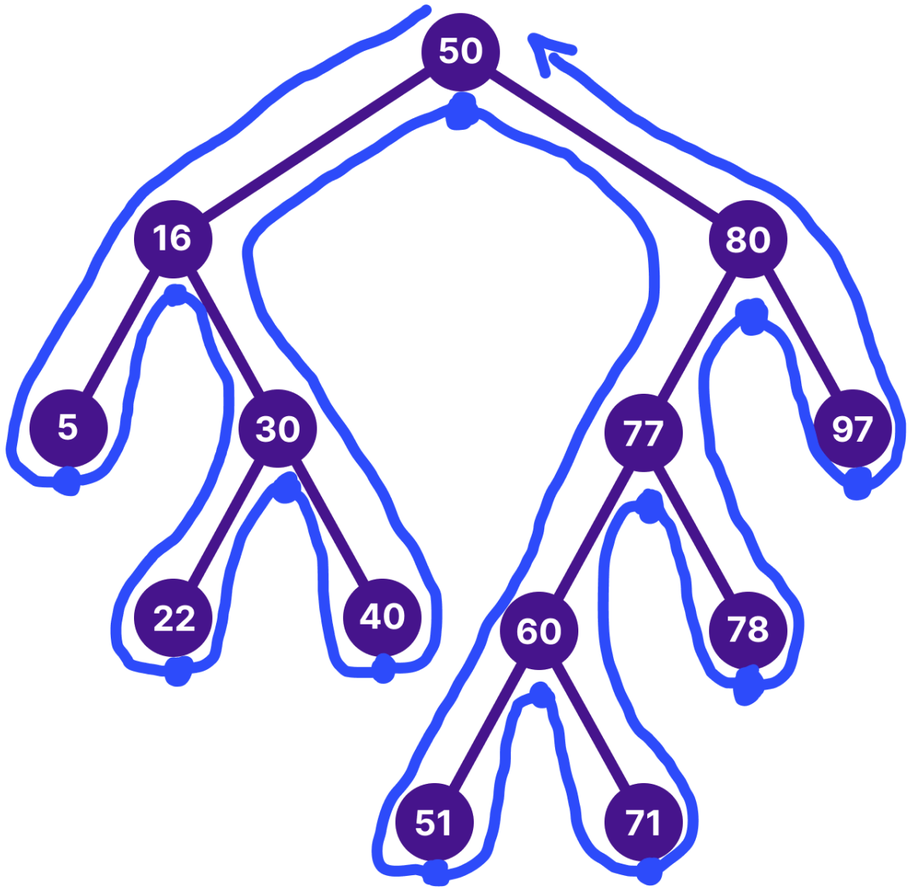
5 16 22 30 40 50

Inorder



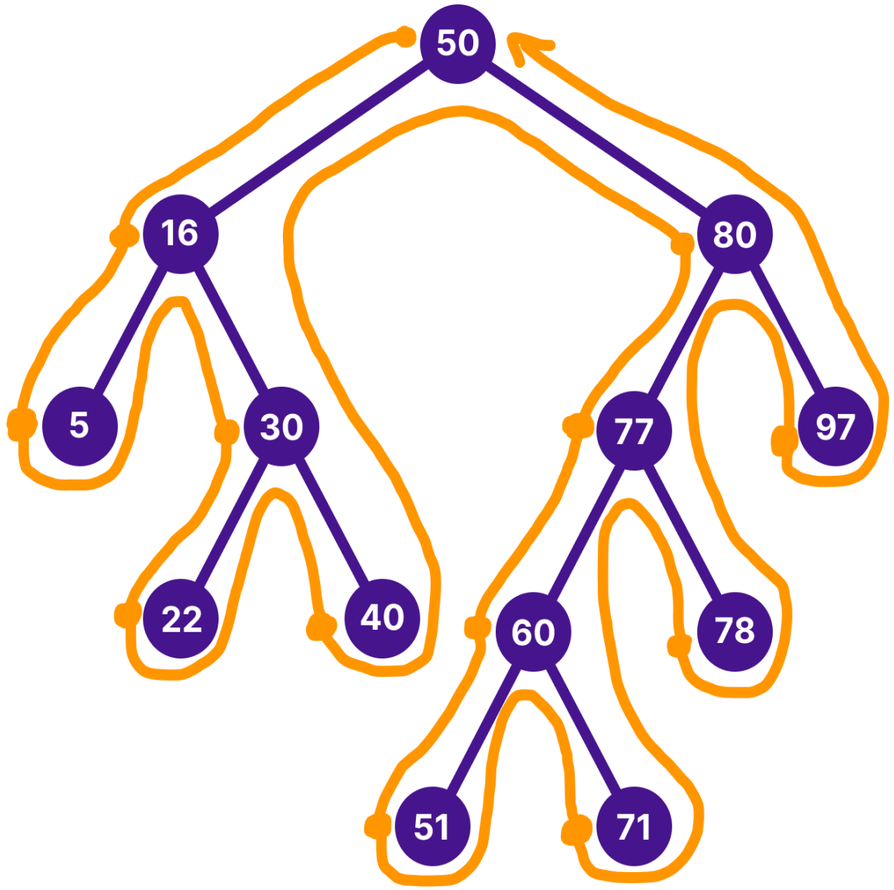
5 16 22 30 40 50 51

Inorder



5 16 22 30 40 50 51 60 71 77 78 80 97

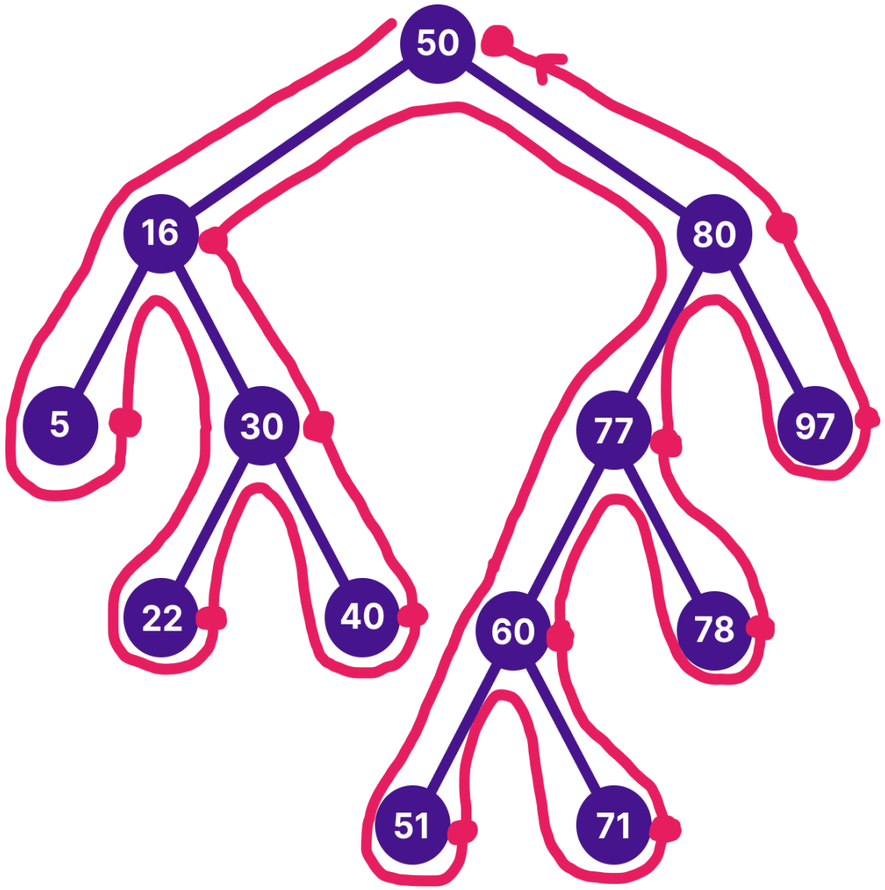
Preorder




```
PREORDER-TREE-WALK(x)
1  if x ≠ NIL
3    print x.key
2    PREORDER-TREE-WALK(x.left)
4    PREORDER-TREE-WALK(x.right)
```

50 16 5 30 22 40 80 77 60 51 71 78 97

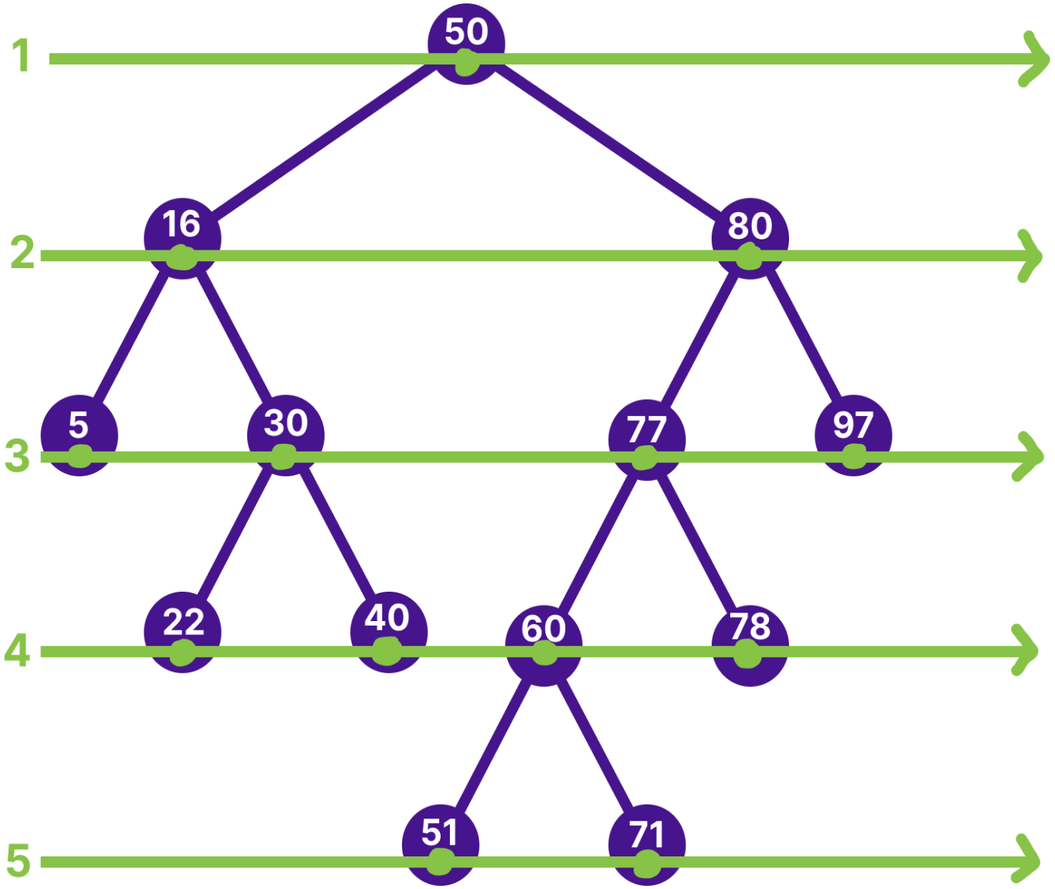
Postorder




```
POSTORDER-TREE-WALK(x)
1  if x ≠ NIL
2    POSTORDER-TREE-WALK(x.left)
4    POSTORDER-TREE-WALK(x.right)
3    print x.key
```

5 22 40 30 16 51 71 60 78 77 97 80 50

Breitensuche



BREADTH-FIRST-SEARCH(T)

```
1  Q ← [T.root]
2  while Q ≠ ∅
3     x ← dequeue(Q)
4     print x.key
5     if x.left ≠ NIL
6        enqueue(Q, x.left)
7     if x.right ≠ NIL
8        enqueue(Q, x.right)
```

Wir werden diesen Algorithmus genauer in der Vorlesungseinheit zu Graphen diskutieren.

50 16 80 5 30 77 97 22 40 60 78 51 71

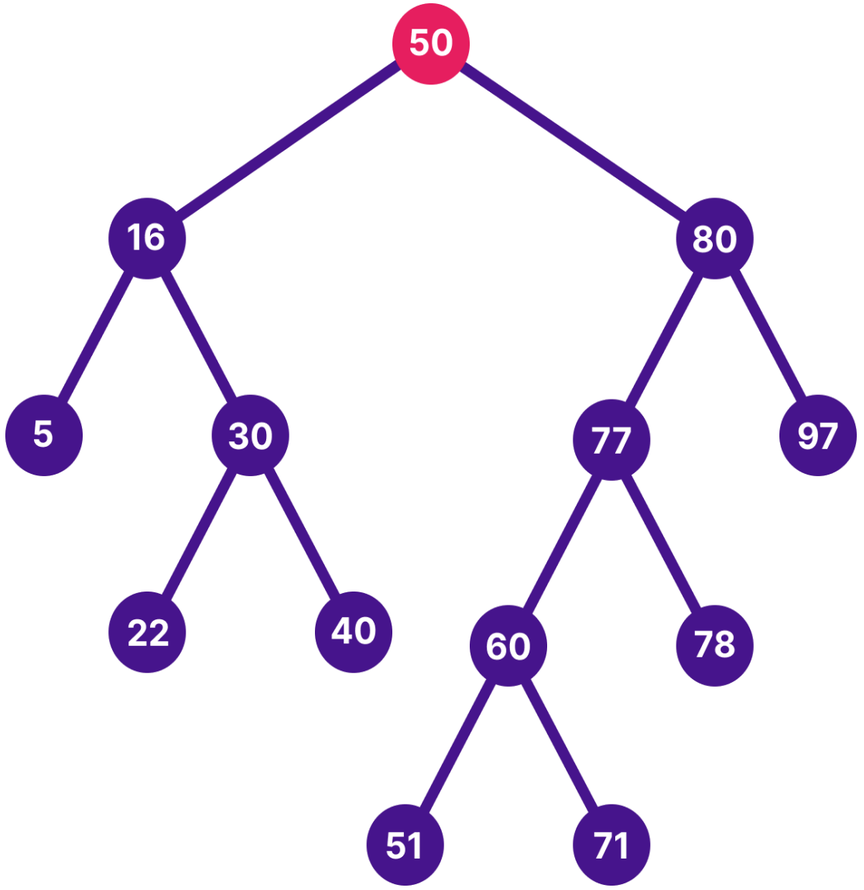
Binäre Suchbäume

Suchen

Suchen in binären Suchbäumen ist Binärsuche!

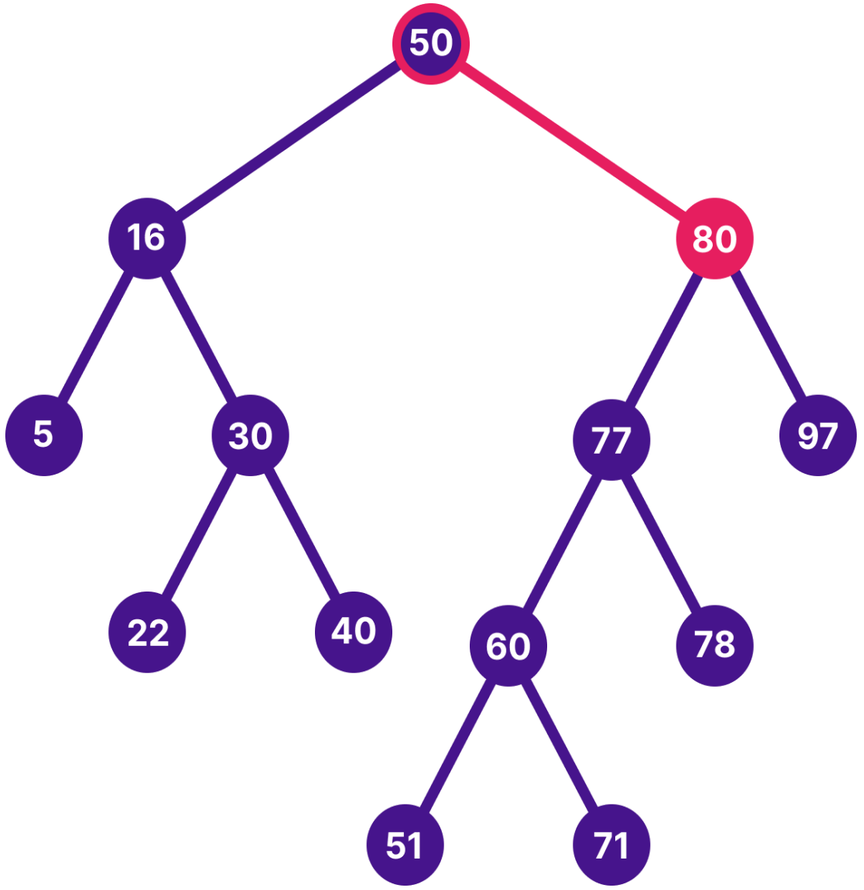
Suche im Binärbaum

Wir suchen wieder nach 78 und starten bei der Wurzel



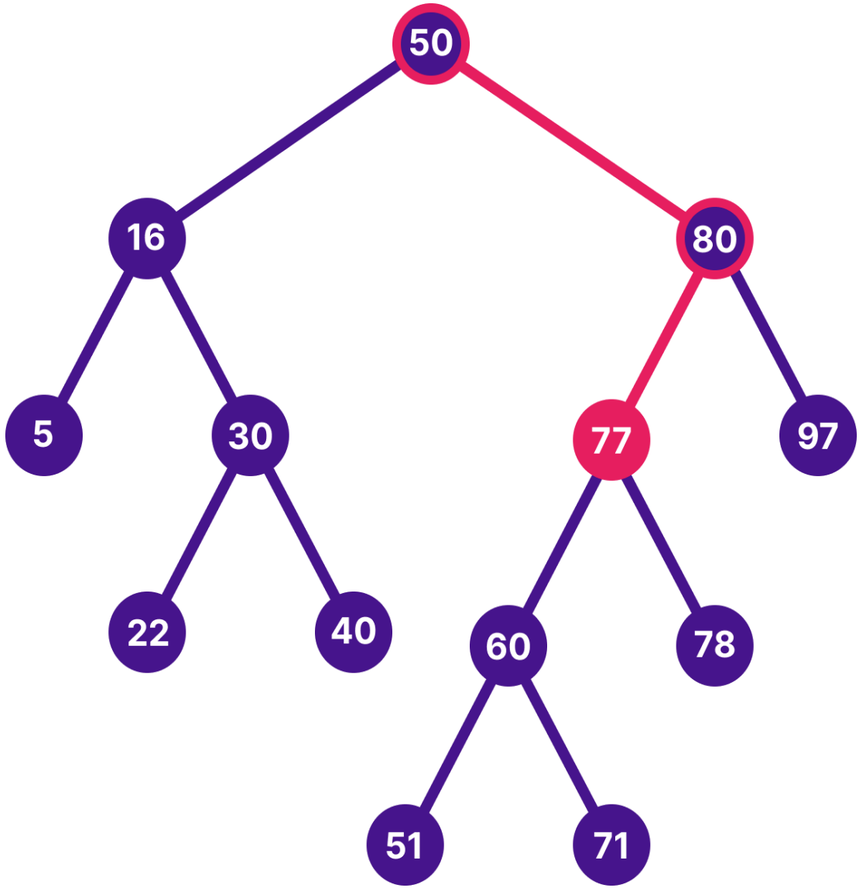
Suche im Binärbaum

$78 > 50$, daher rechtes Kind



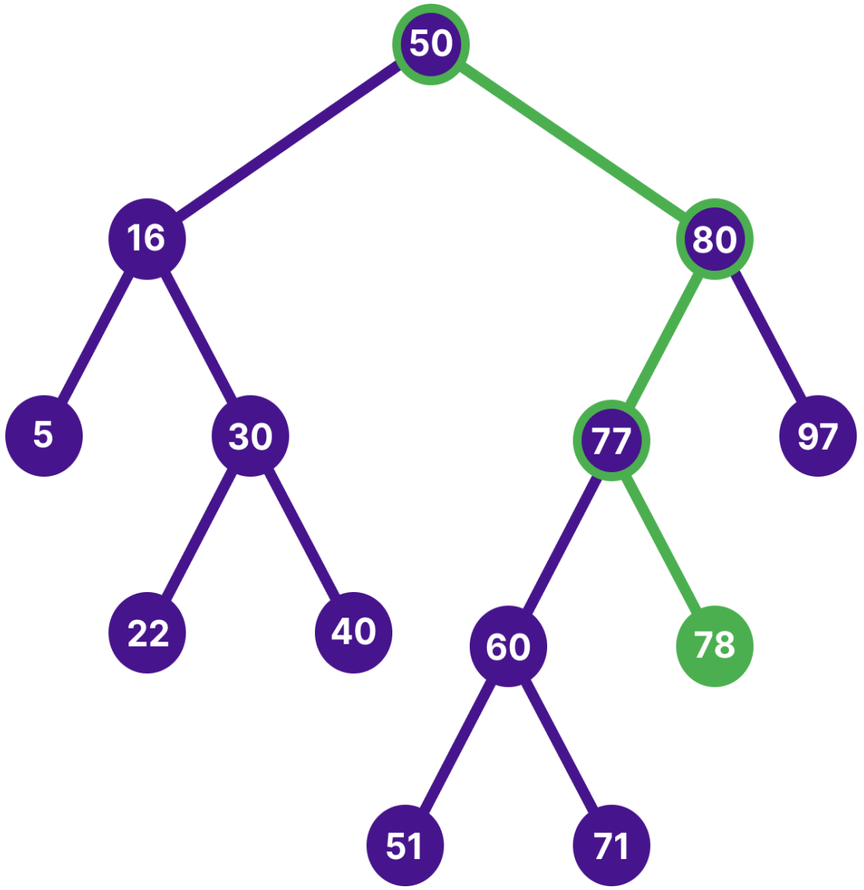
Suche im Binärbaum

$78 < 80$, daher linkes Kind



Suche im Binärbaum

$78 > 77$, daher rechtes Kind, **Suche beendet**



Algorithmus zur Nachlese

TREE-SEARCH(x, k)

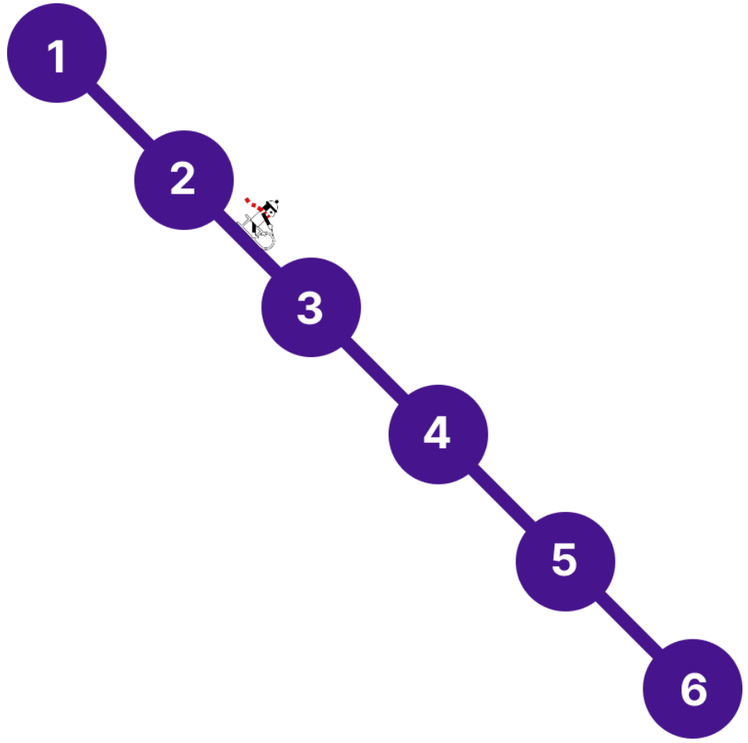
```
1  if x == NIL or k == x.key
2    return x
3  if k < x.key
4    TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
```

Laufzeit

$\mathcal{O}(h)$ für einen Binärbaum mit Höhe h

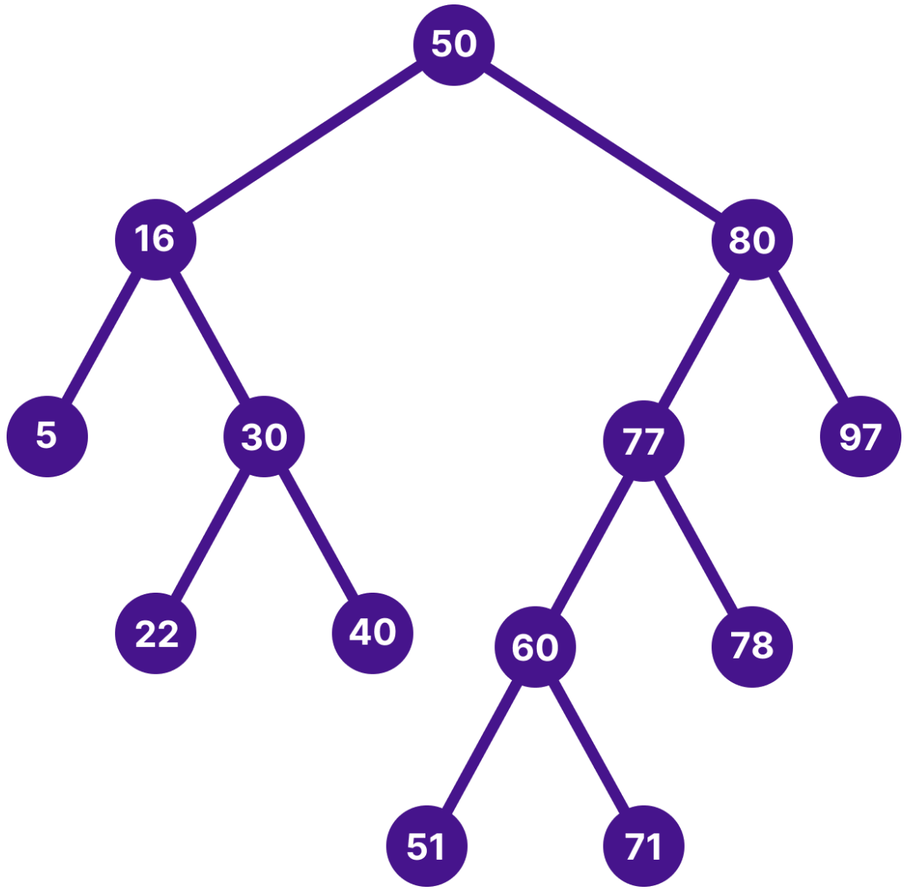
Wir wollen daher die Höhe möglichst klein halten!

Im worst case $\mathcal{O}(n)$



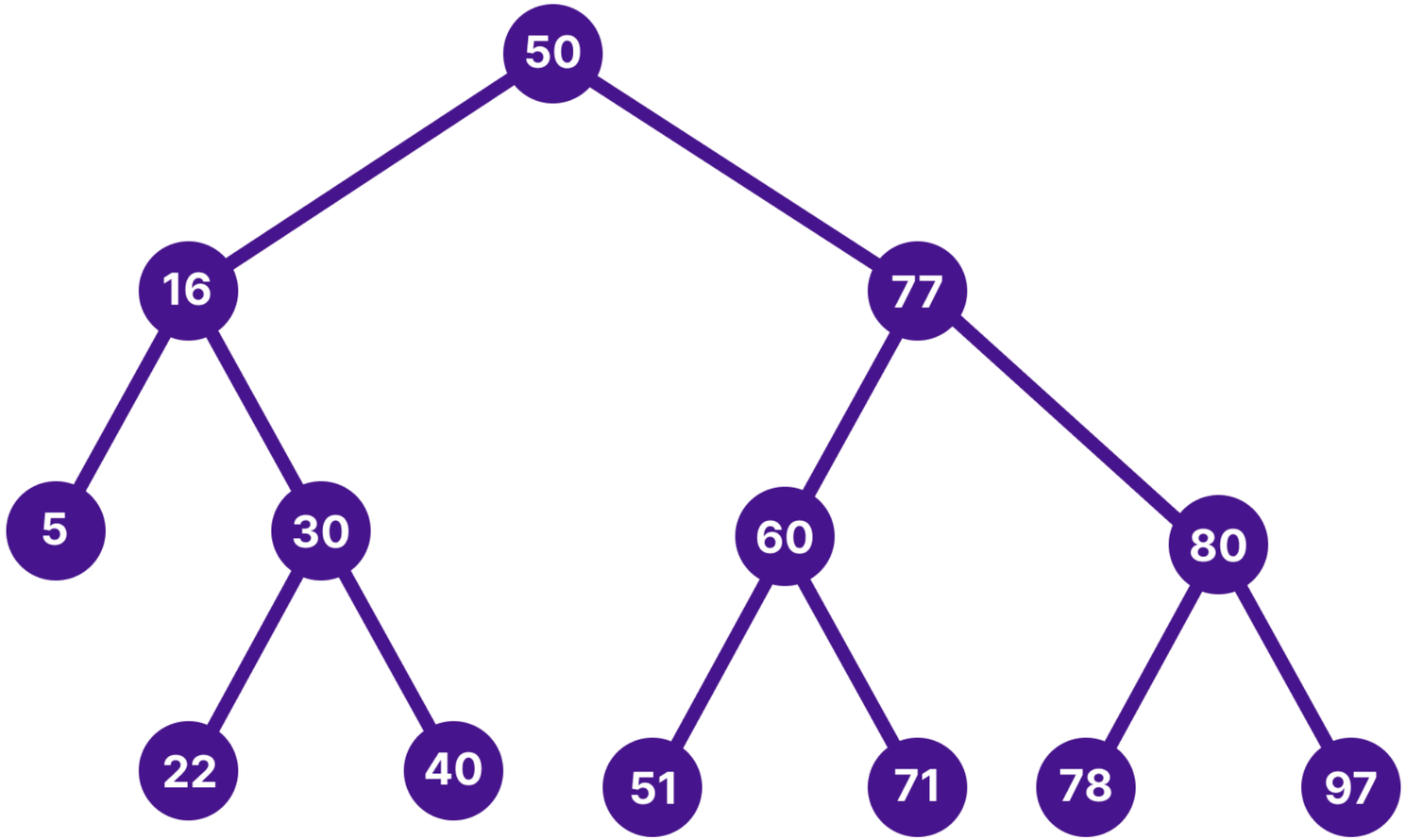
Bei degenerierten Bäumen reduziert sich die Binärsuche auf eine lineare Suche.

Ist dieser Baum optimal hinsichtlich Höhe?



Nein!

Besser:



Hier haben wir nun die Höhe um 1 reduziert.

Wir wollen Bäume möglichst ausbalanciert halten.

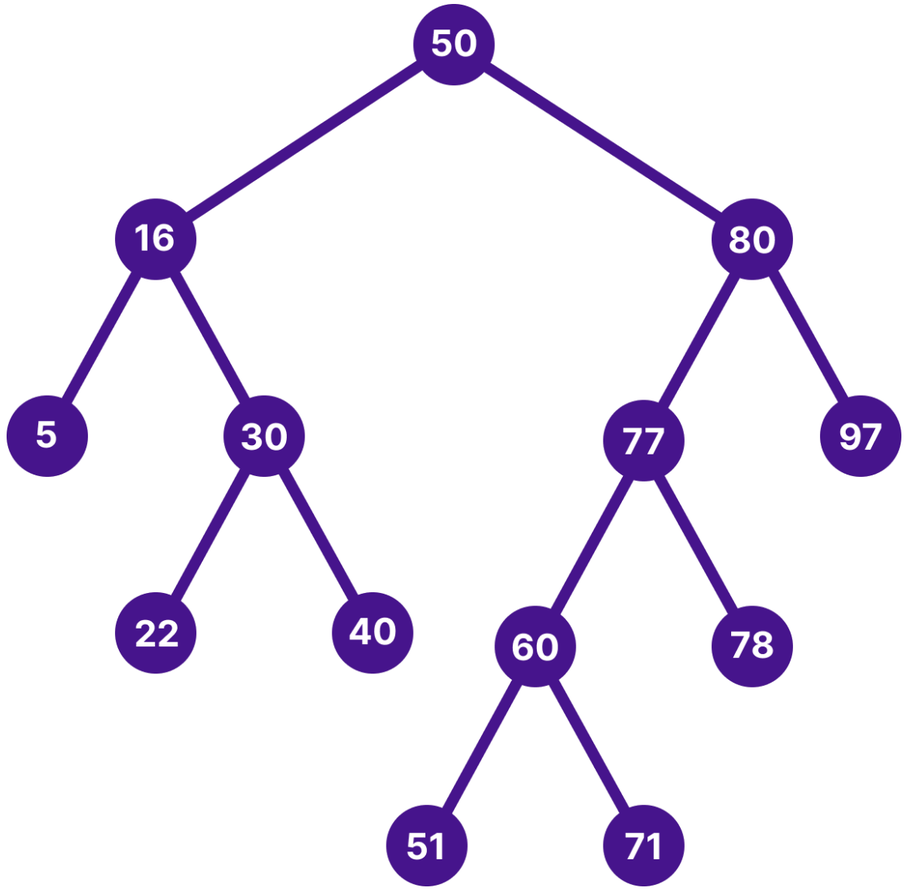
Wir müssen daher Elemente gezielt so einfügen, dass der Baum balanciert bleibt.

Binäre Suchbäume

Einfügen und Löschen

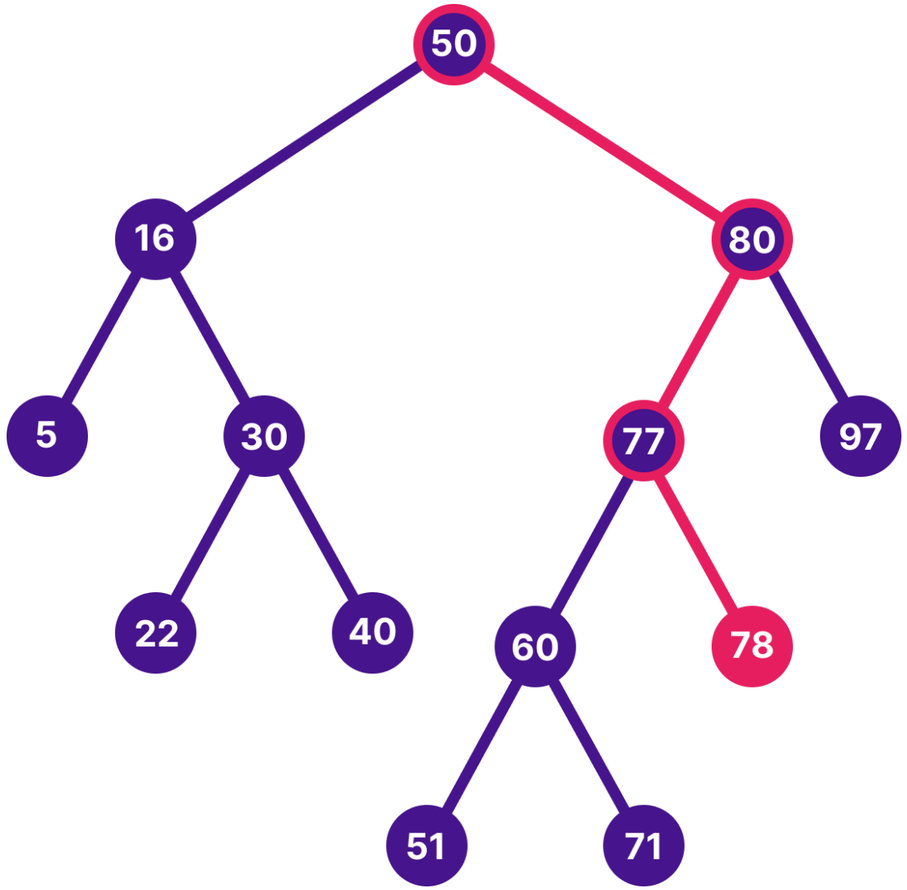
Einfügen

Wir wollen das Element 79 in den nachfolgenden Baum einfügen:



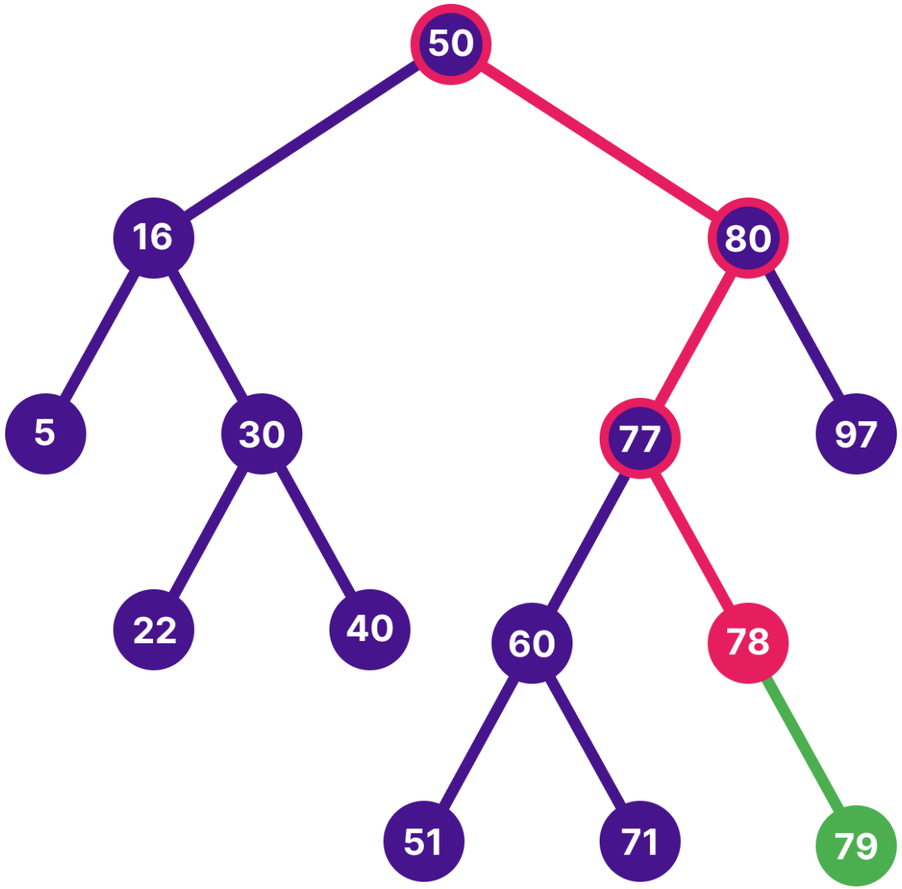
Einfügen

Zuerst traversieren wir zum ähnlichsten Blatt



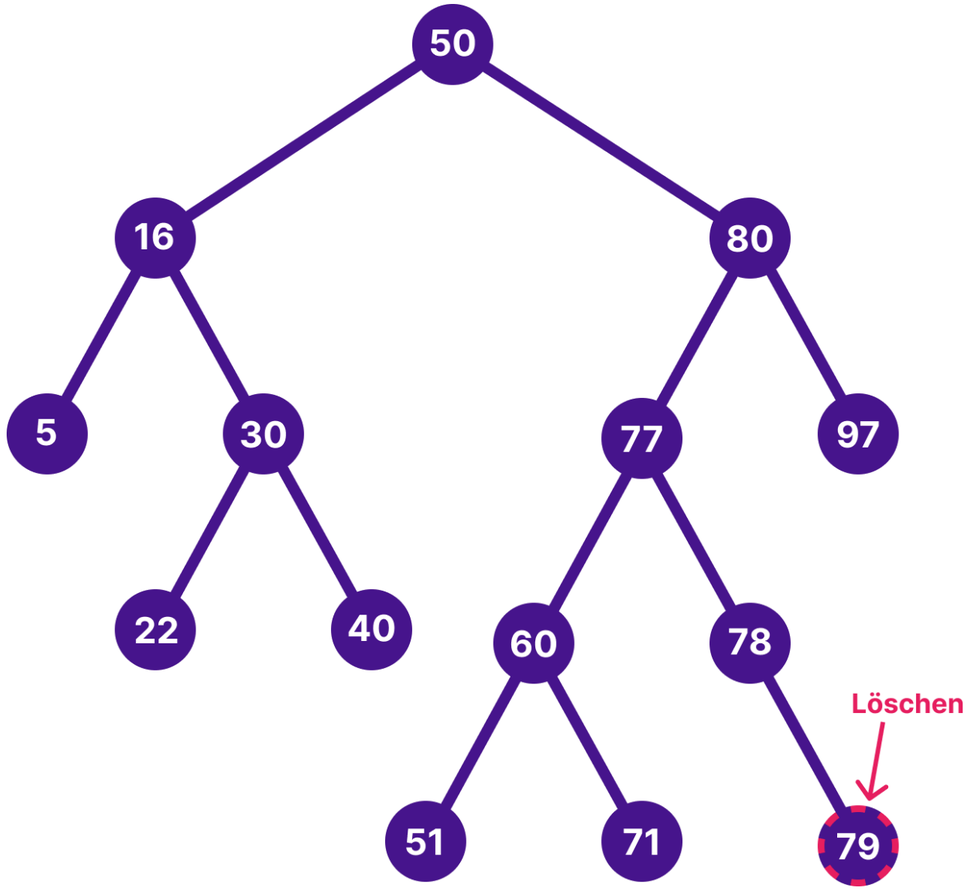
Einfügen

Danach hängen wir das neue Element an das Blatt im Baum an.



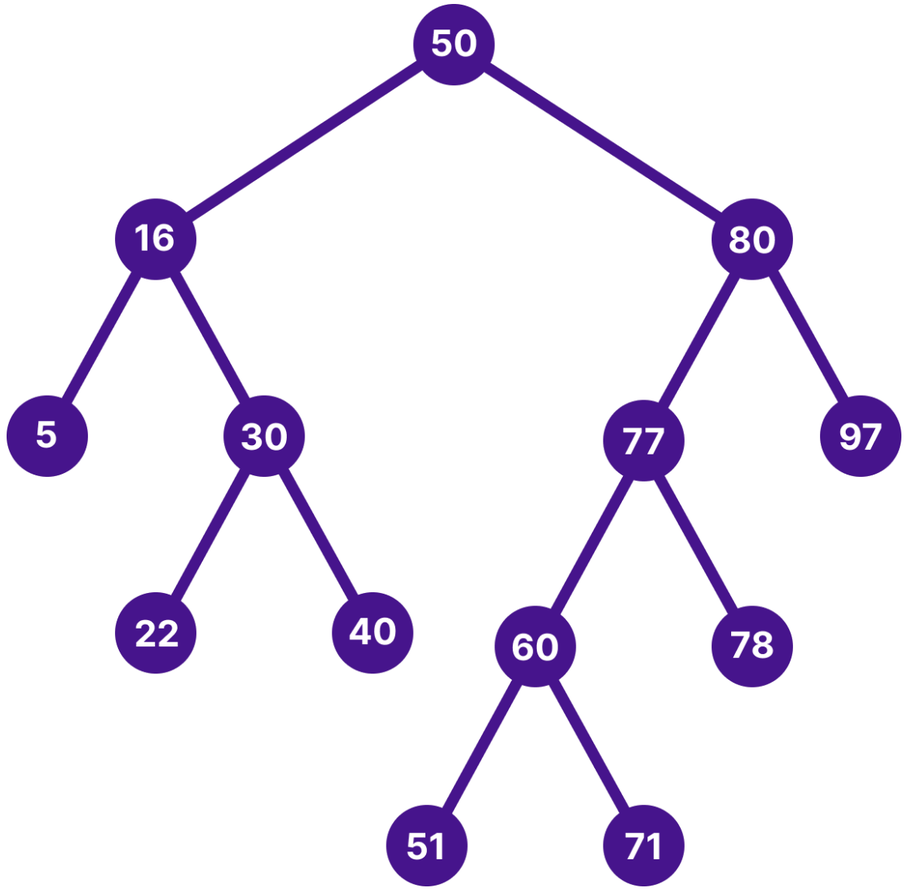
Löschen

Blattknoten sind einfach zu löschen ...



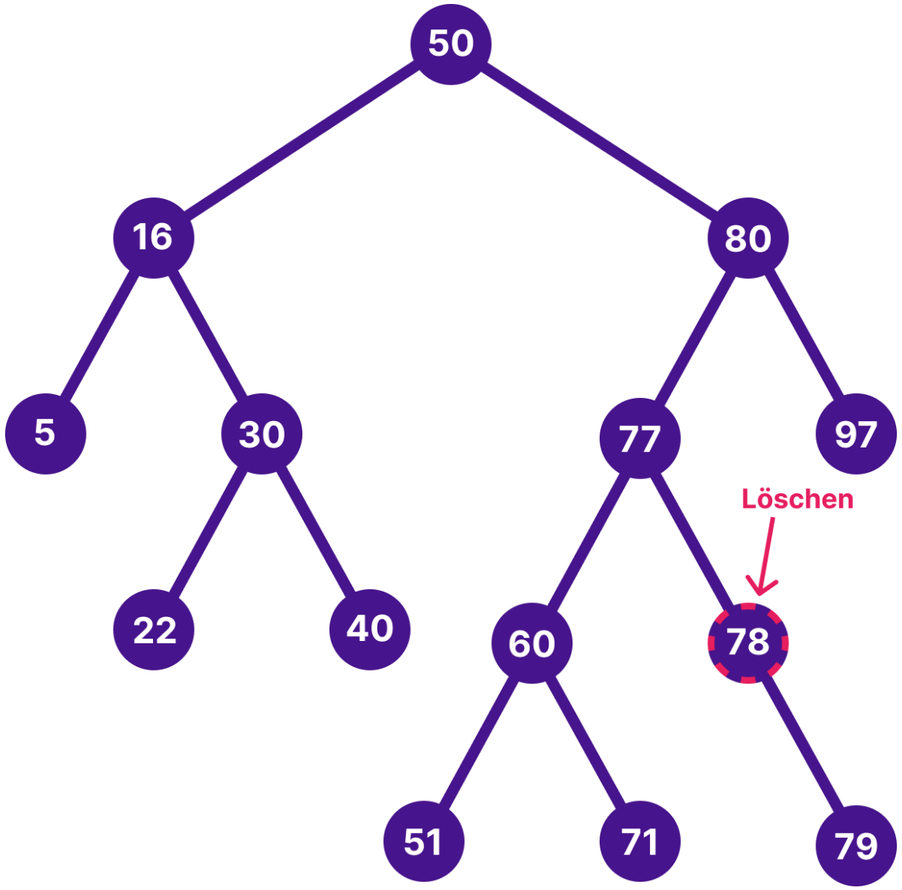
Löschen

... man entfernt sie einfach!



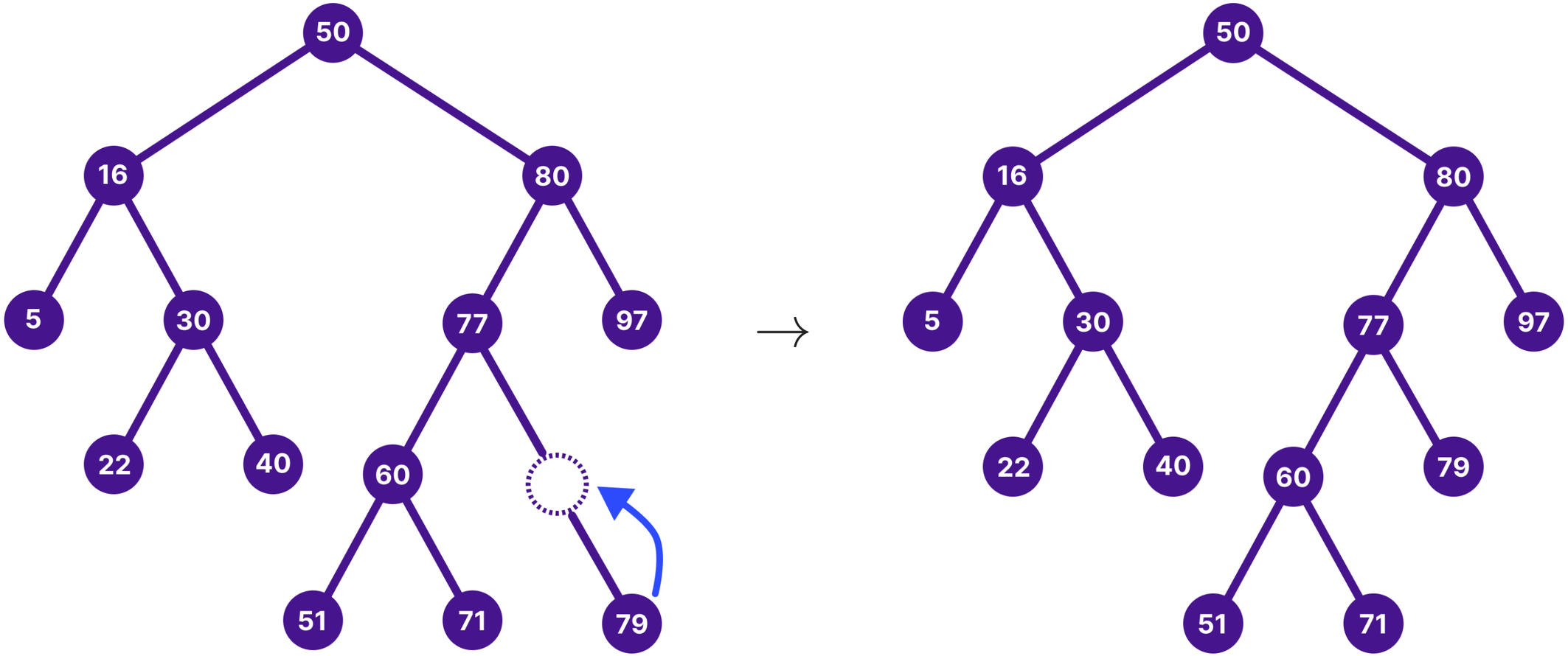
Löschen

Innere Knoten mit einem Einzelkind ebenso einfach ...



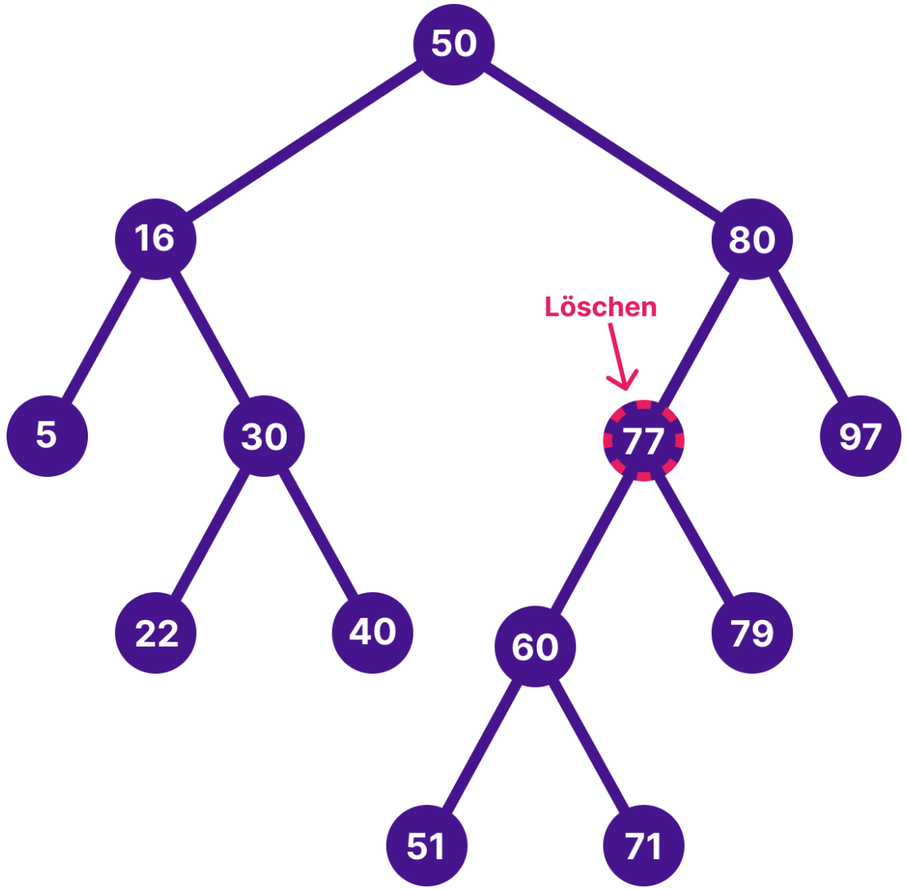
Löschen

... das Kind rutscht einfach nach (egal wie viele Nachfolger es hat).



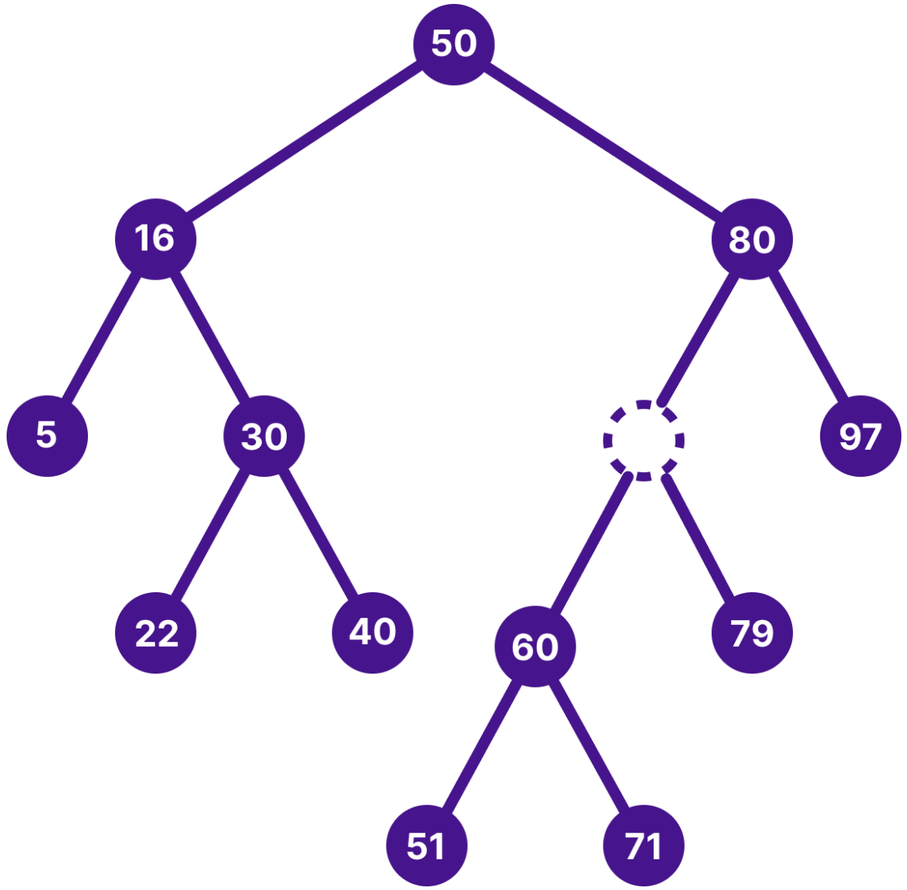
Löschen

Etwas komplexer: Der zu löschende Knoten besitzt zwei Kinder.



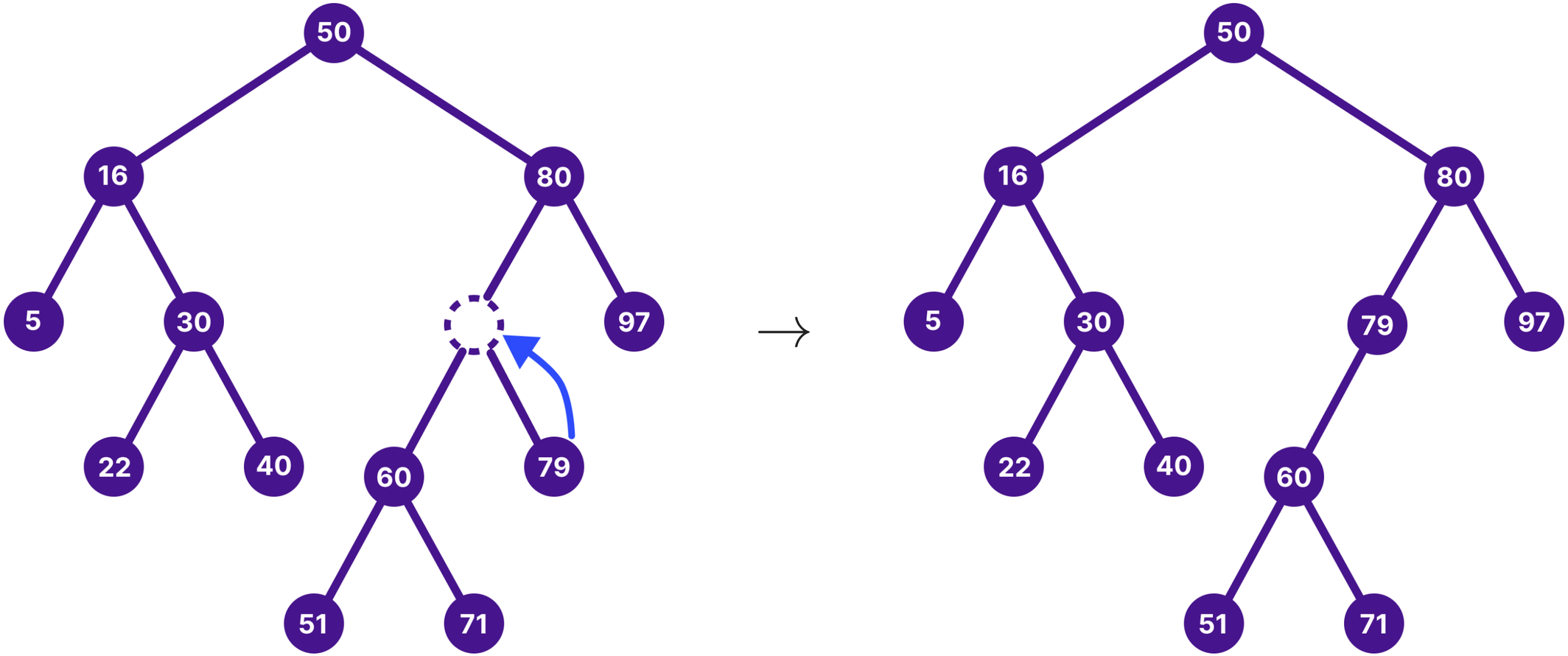
Löschen

Wie füllen wir diese Lücke?



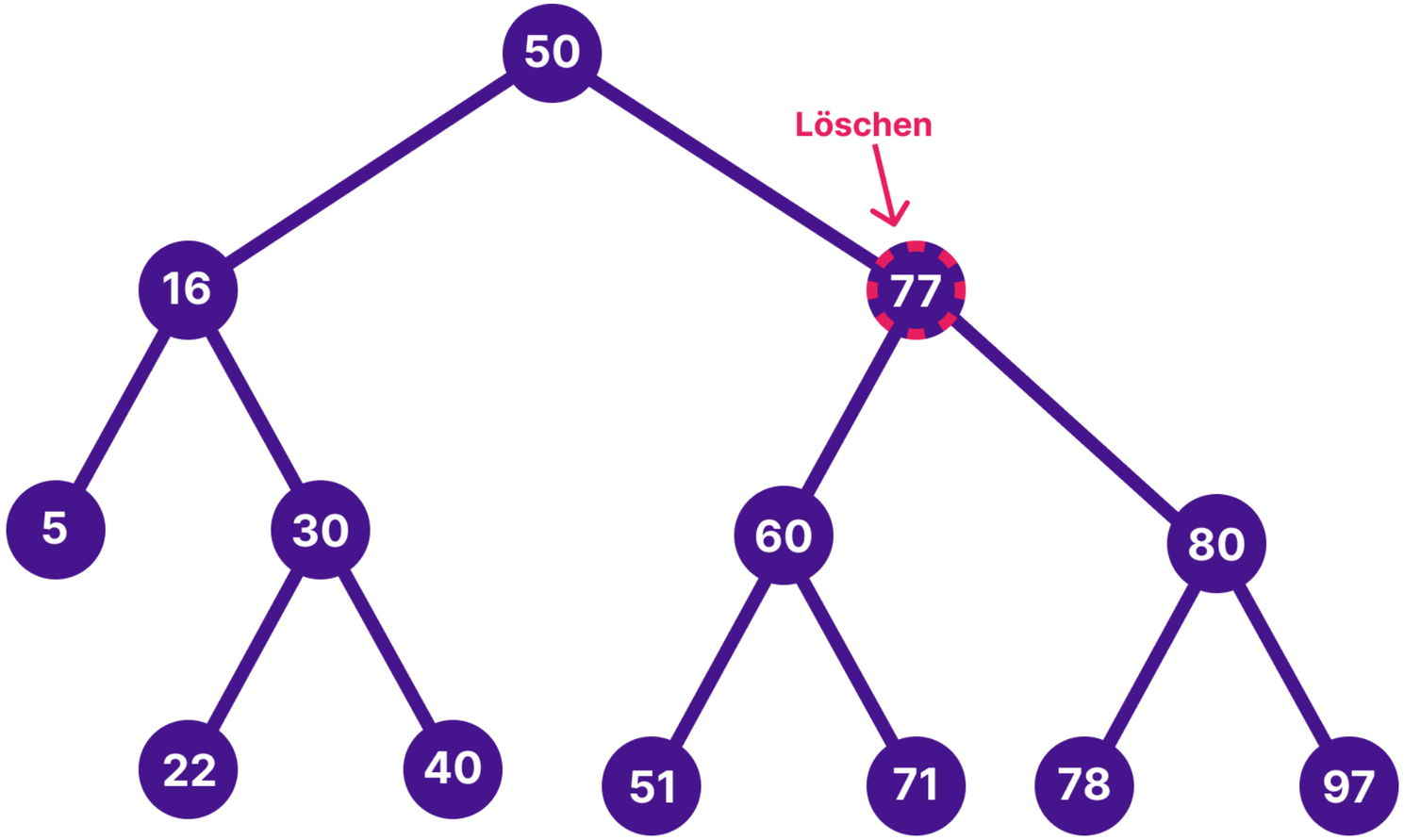
Löschen

Das Kind ohne Nachfolger bietet sich als Lückenfüller an.



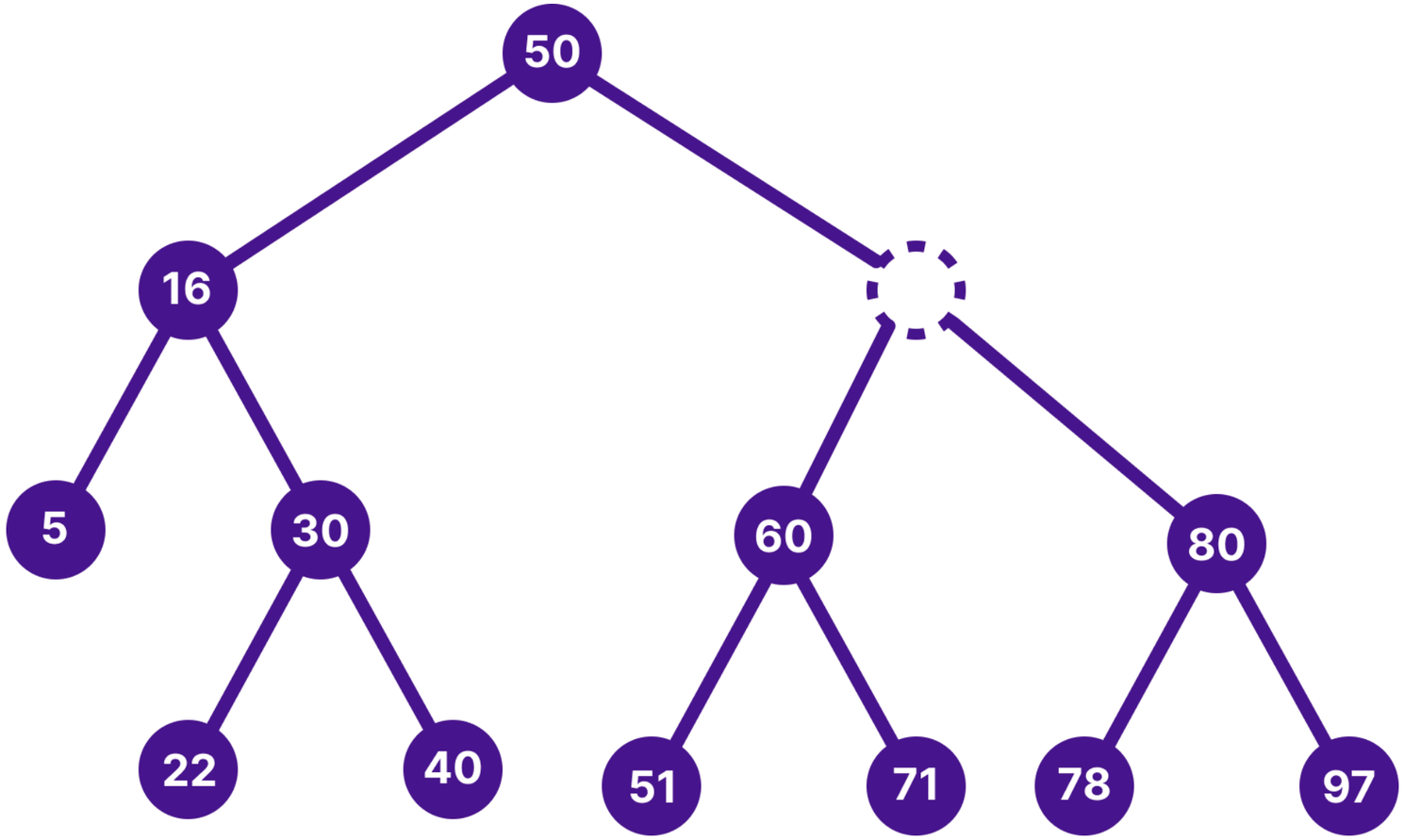
Löschen

Noch komplexer: Ein innerer Knoten in einem ausbalancierten Baum.



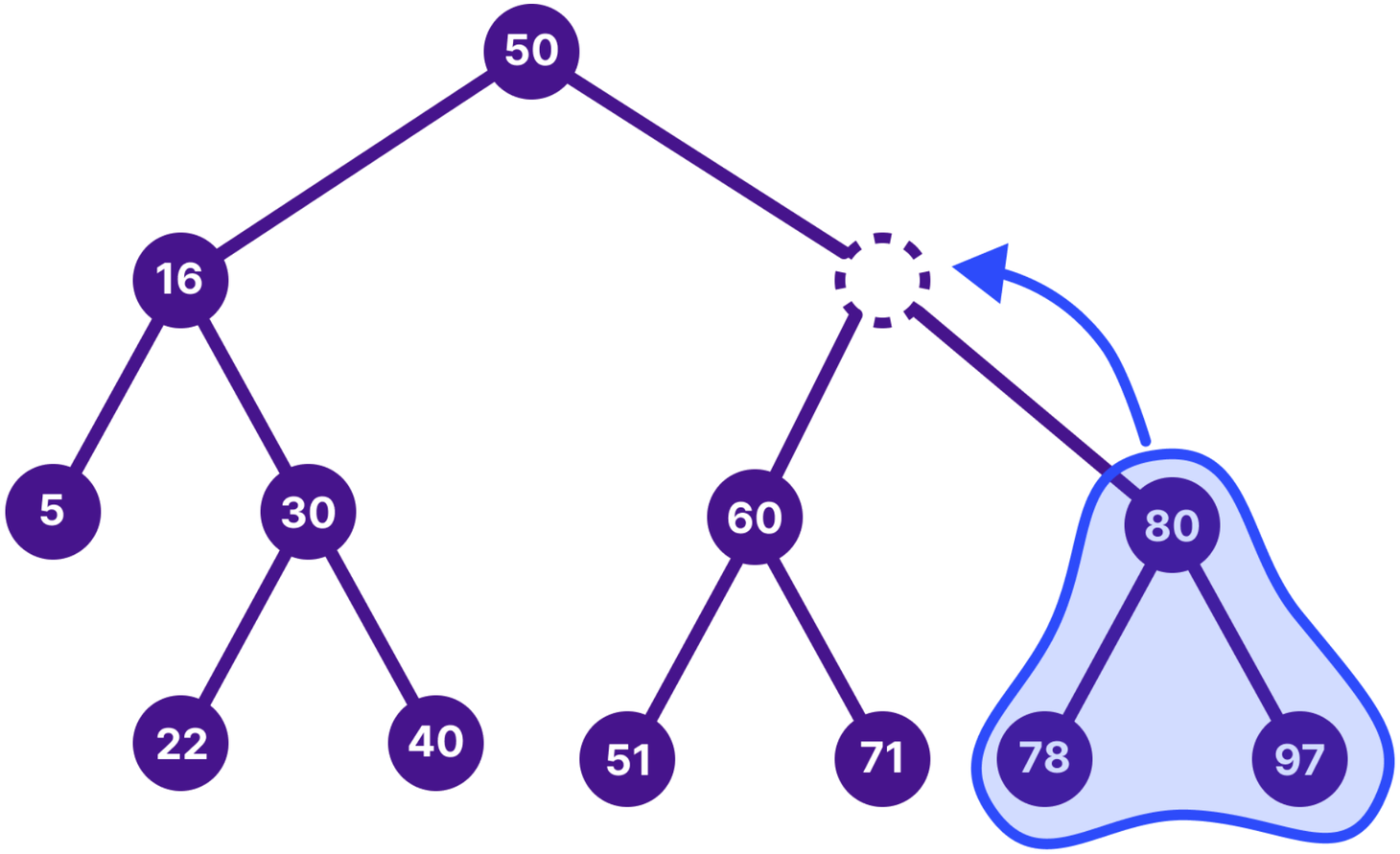
Löschen

Wie füllen wir diese Lücke?



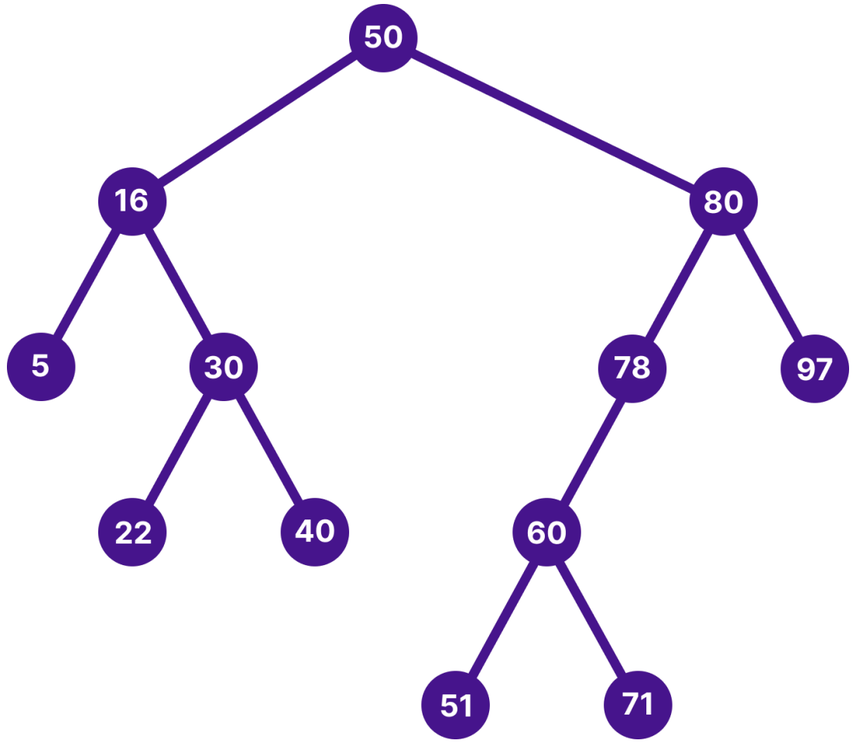
Löschen

Wieder nehmen wir das rechte Kind und setzen es in die Lücke.



Löschen

Ergebnis



Einfache Strategie: Man nimmt den rechten Teilbaum und setzt ihn in die Lücke.

Das Problem von einfachen Binärbäumen

Angenommen Sie fügen Elemente aus einer sortierten Liste nach und nach ein:

`5` `16` `22` `30` `40` `50` `51` `60` `71` `77` `78` `80` `97`

Das Ergebnis? Ein degenerierter Baum!

Binäre Suchbäume

Die AVL Erweiterung

AVL Bäume

Entworfen 1962 von Georgy **A**delson-**V**elsky und Evgenii **L**andis

Bildquellen: computerhistory.org/chess/index.php und opc.mfo.de/detail?photo_id=2447



Hat nichts mit der AVL Firma zu tun!



Anstalt für Verbrennungskraftmaschinen List, est. 1948

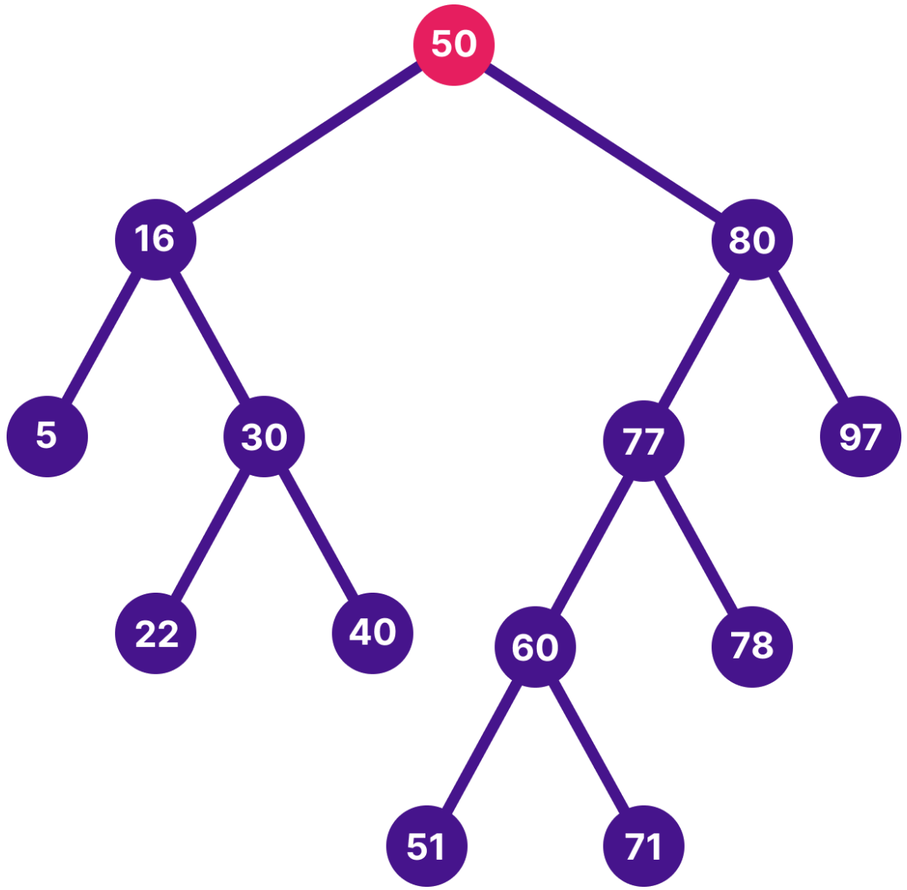
Zwei wichtige Prinzipien

1. Balance-Faktor
2. Knoten Rotieren

AVL Synopsis

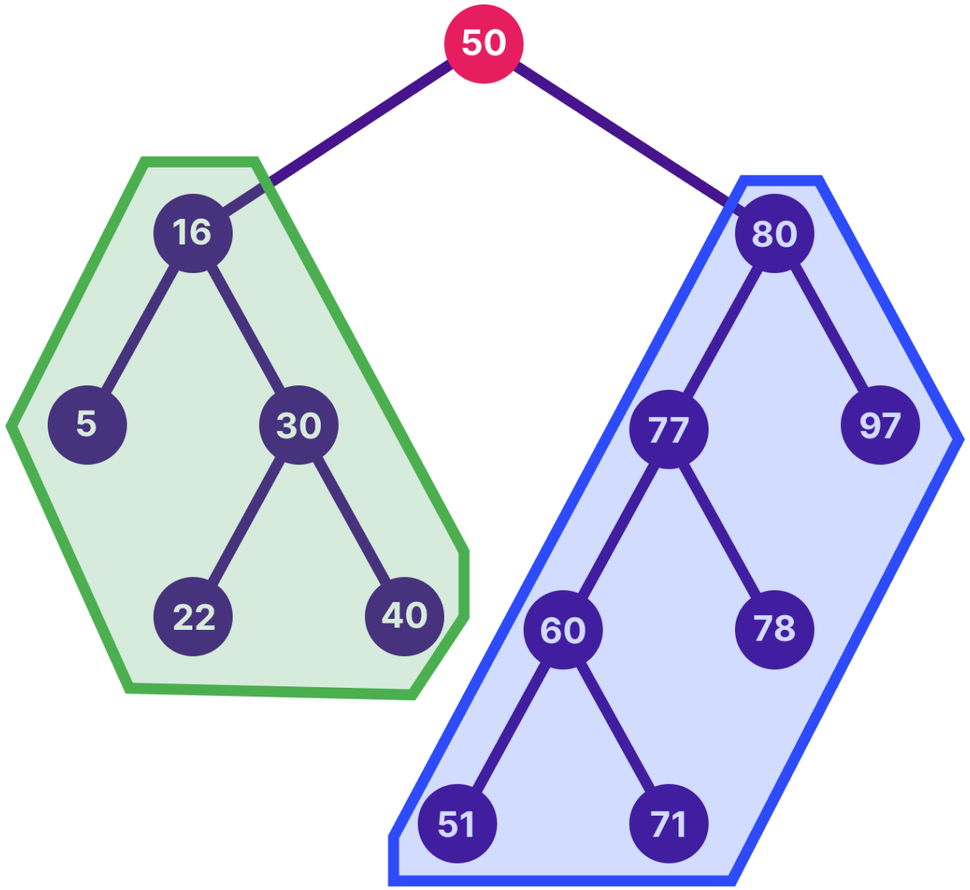
1. Element einfügen (wie vorhin besprochen).
2. Balance-Faktor pro Knoten berechnen.
3. Knoten rotieren wenn Ungleichgewicht herrscht.

Balance-Faktor eines Baumes



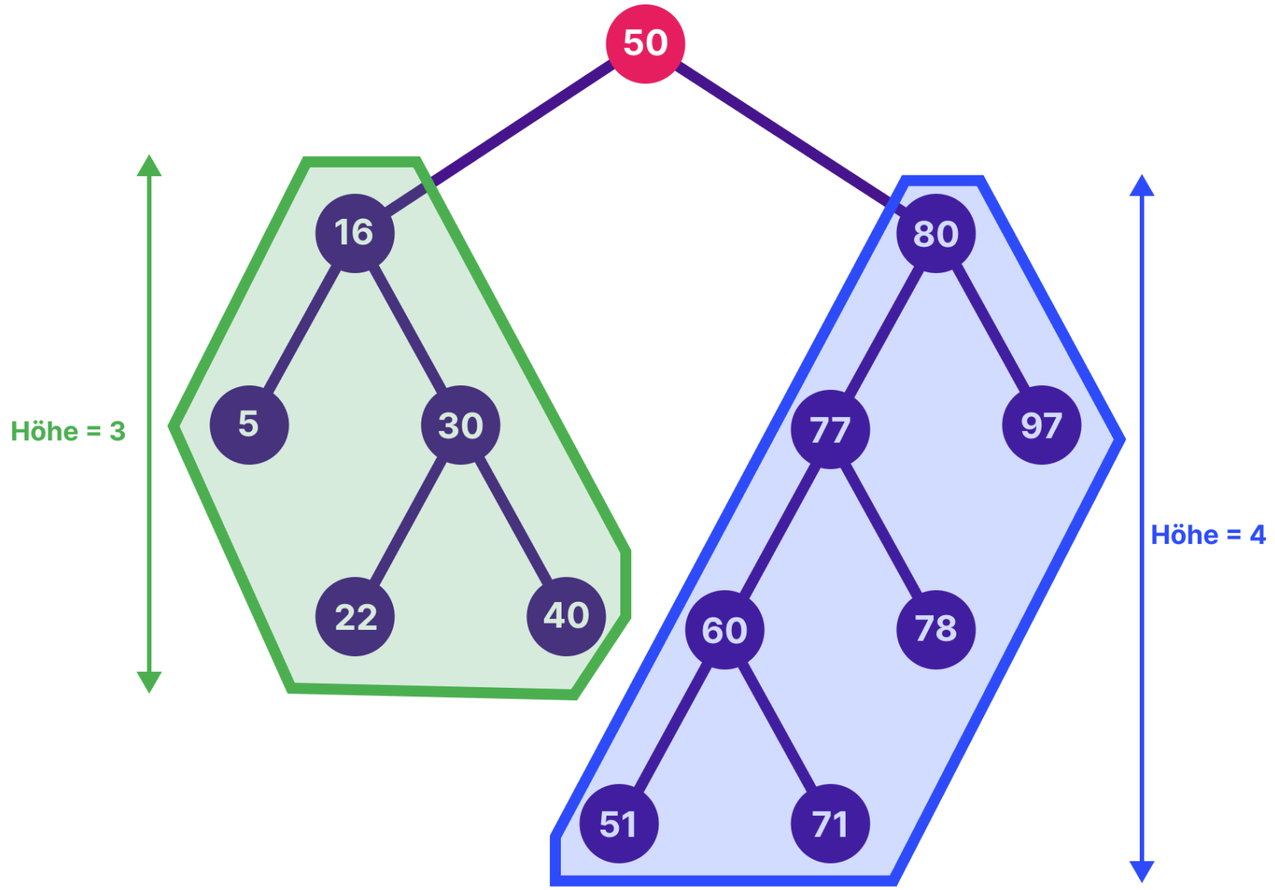
Wir möchten den Balance-Faktor der Wurzel berechnen.

Balance-Faktor eines Baumes



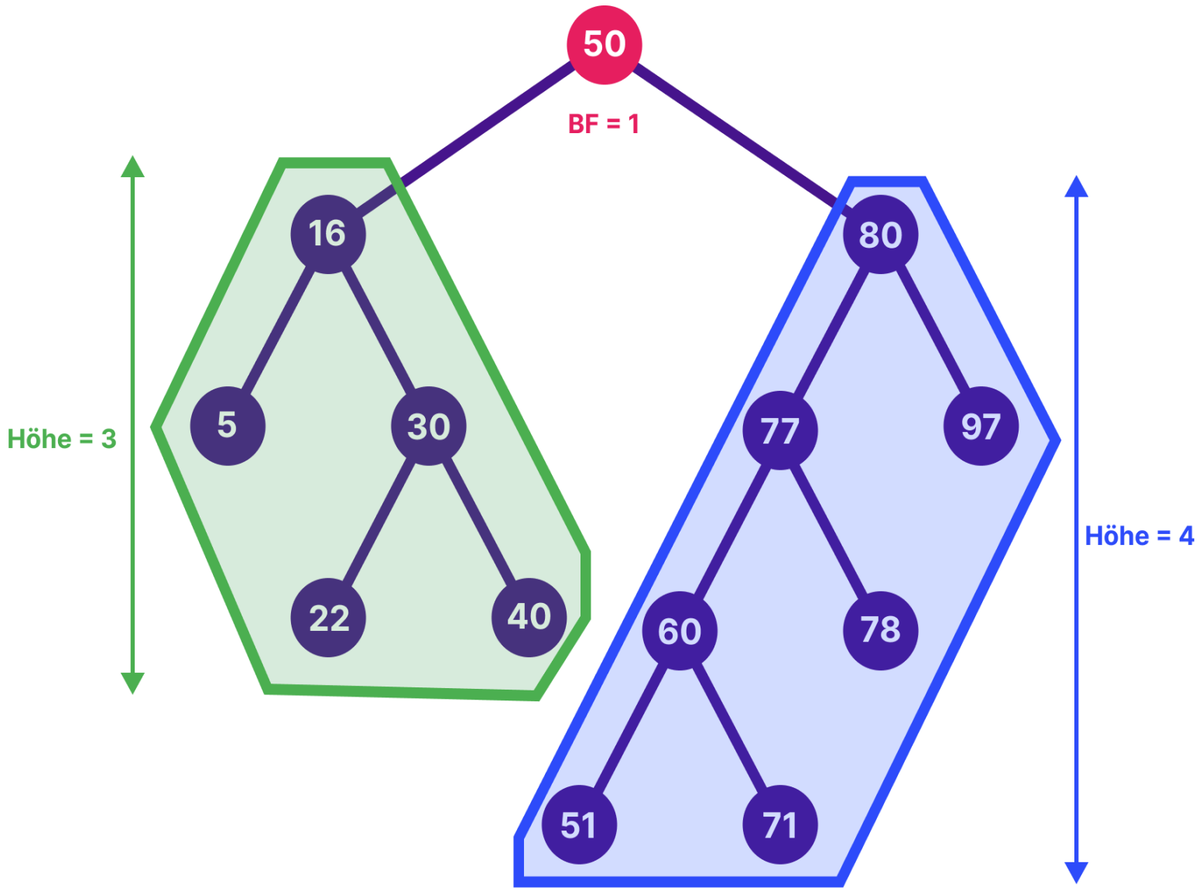
Dazu sehen wir uns den linken und den rechten Teilbaum an ...

Balance-Faktor eines Baumes



... und betrachten die Höhen der Teilbäume.

Balance-Faktor eines Baumes

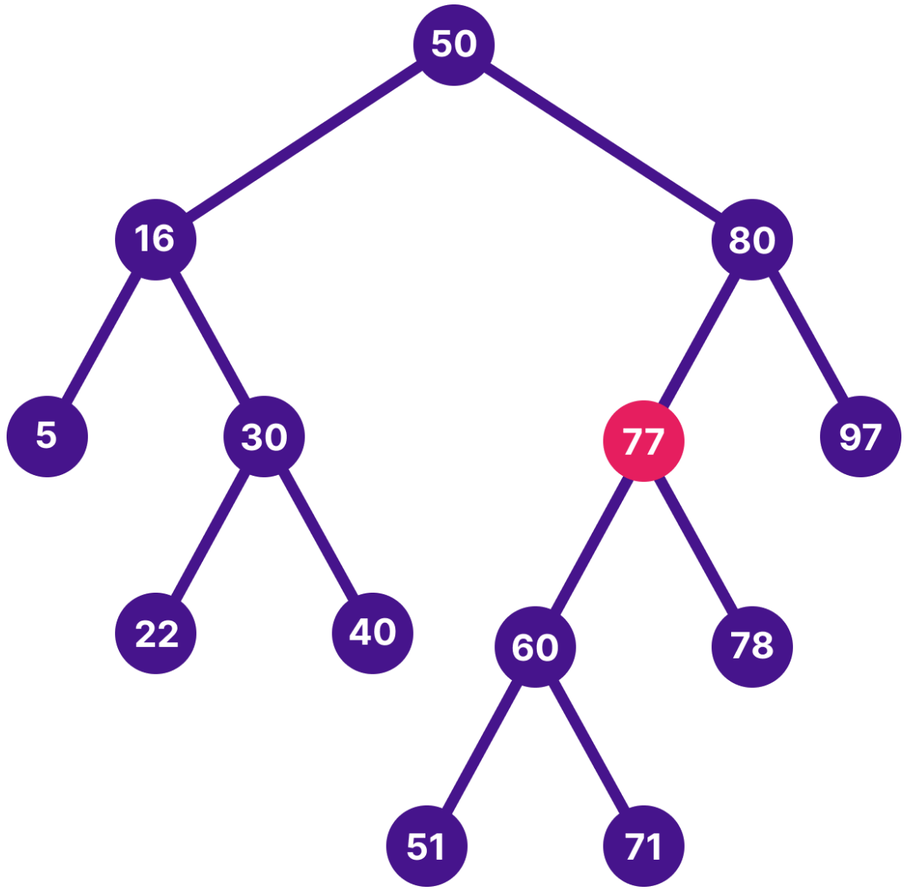


Der Balance-Faktor **BF** eines Baumes ist die Differenz der Höhen der Teilbäume.

$$BF = h_{\text{rechts}} - h_{\text{links}}$$

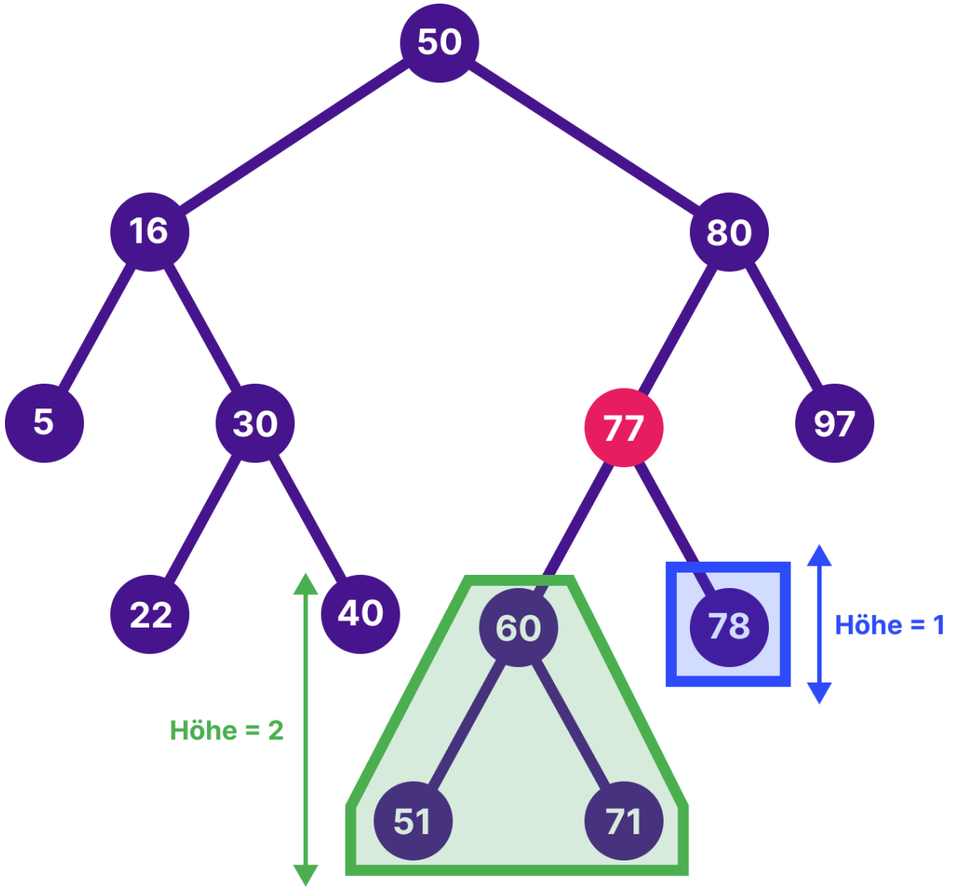
Der Knoten ist **rechtslastig**.

Balance-Faktor eines Baumes

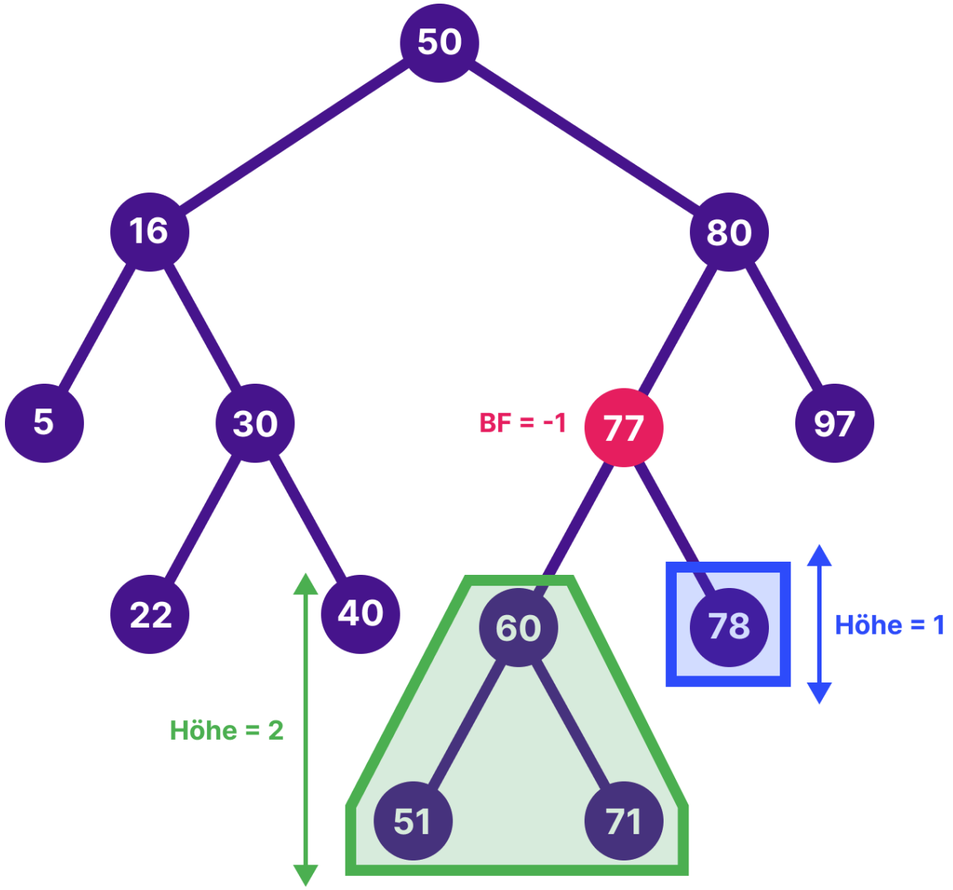


Ein weiteres Beispiel

Balance-Faktor eines Baumes

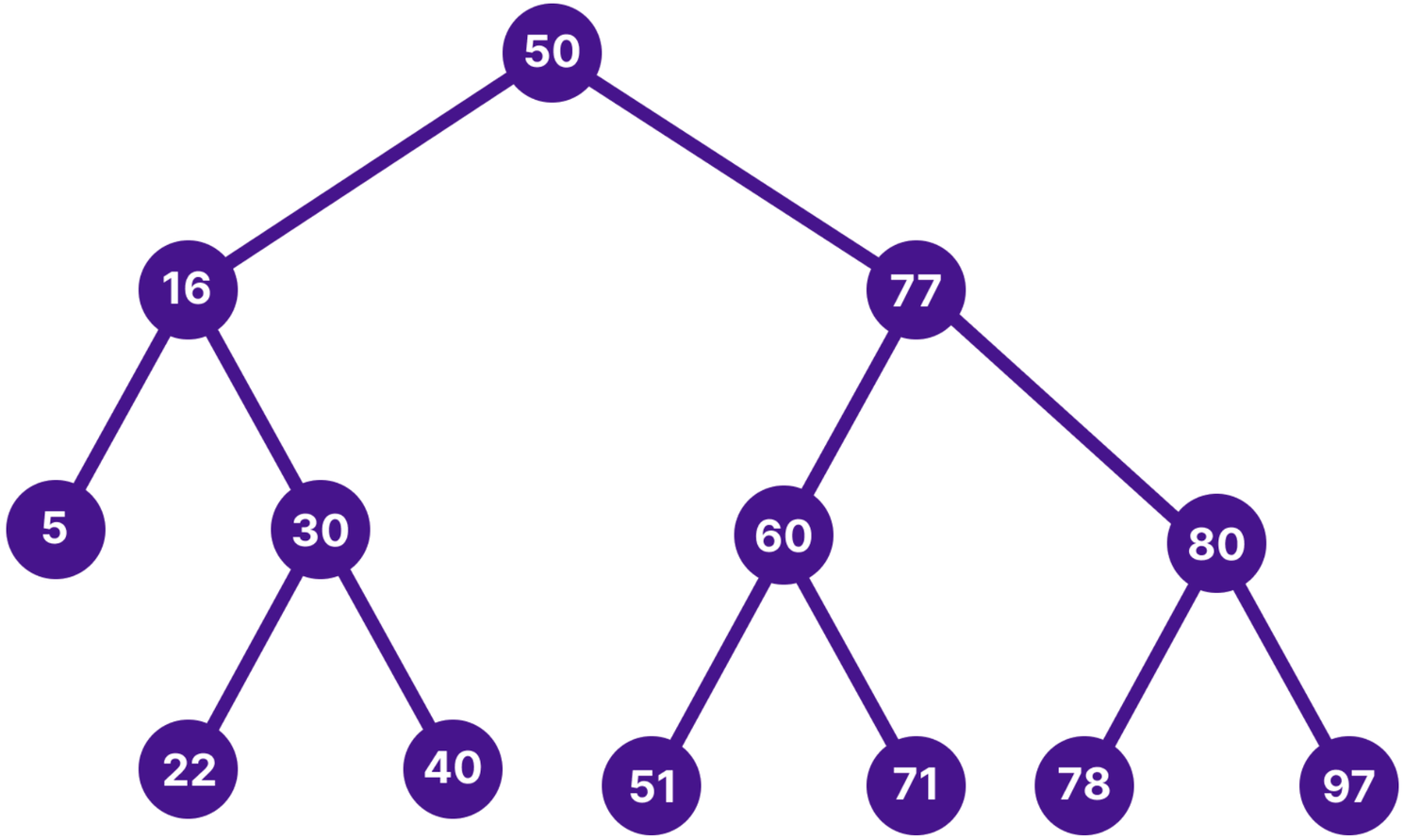


Balance-Faktor eines Baumes



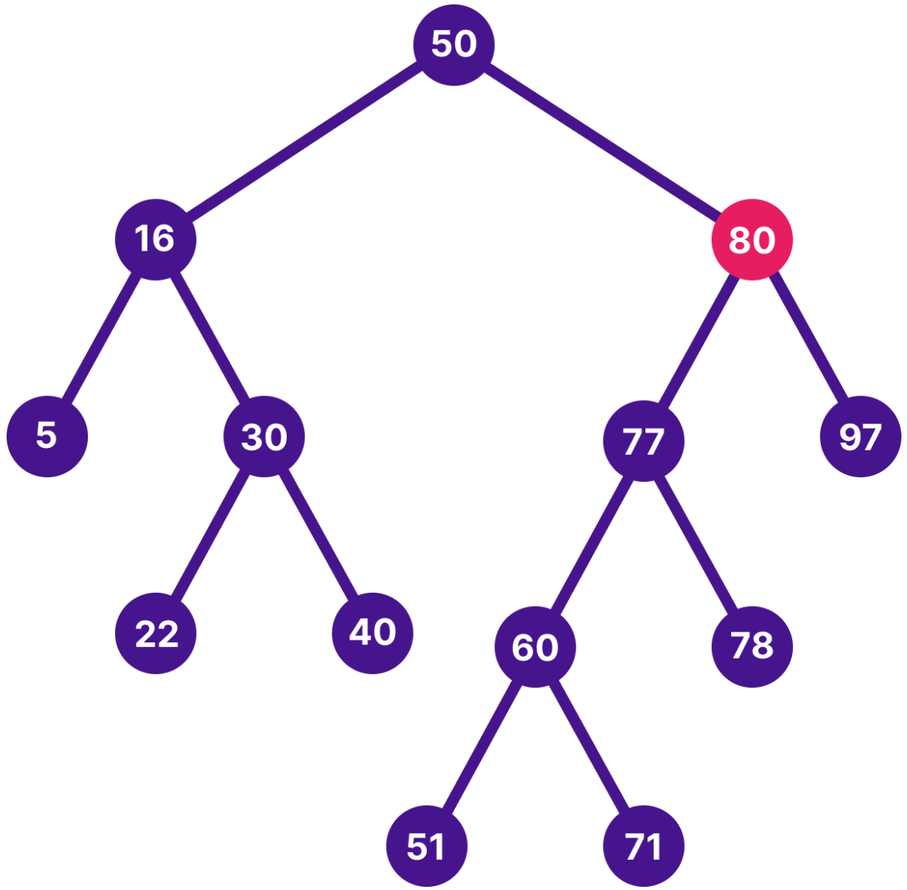
Der Knoten ist **linkslastig**.

Ein Baum ist im Gleichgewicht wenn BF = -1, 0 oder 1 für jeden Knoten ist.

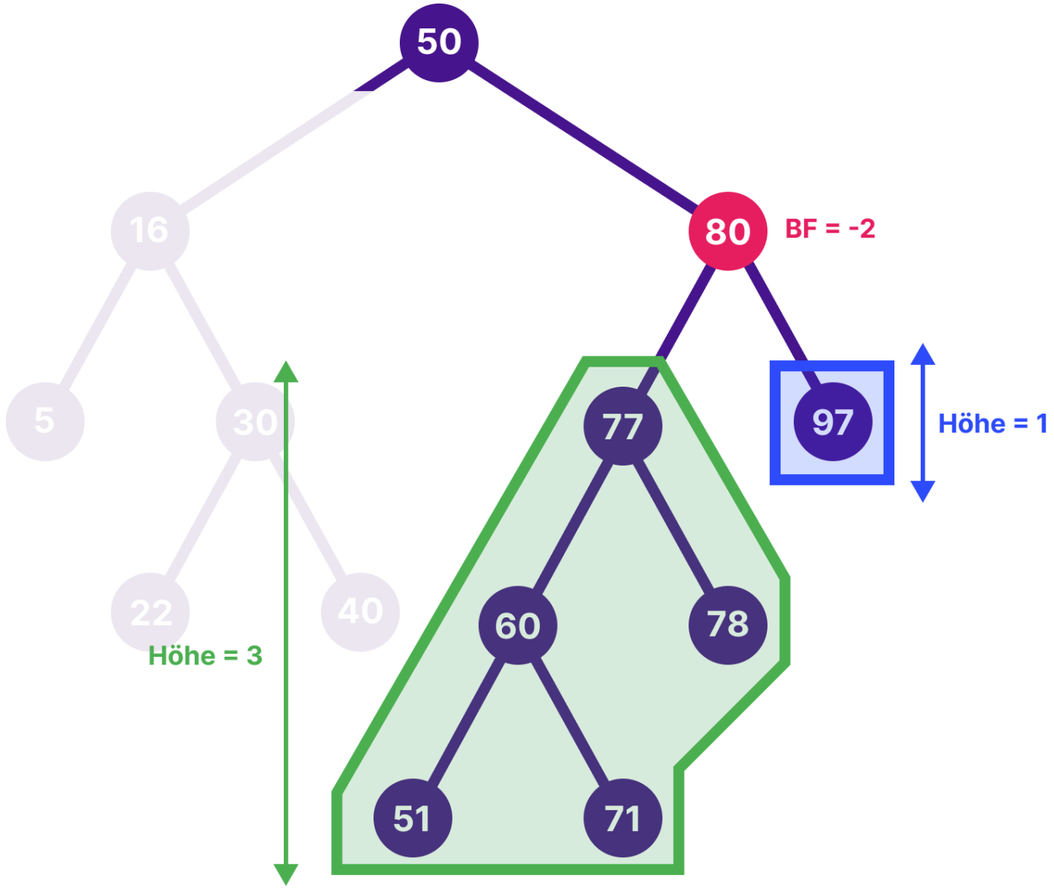


Wir können diesen Baum nicht besser balancieren (jeder Knoten BF = 0),
da wir nicht genug Datenwerte haben, um alle Level vollständig zu befüllen.

Was wäre mit diesem Knoten?

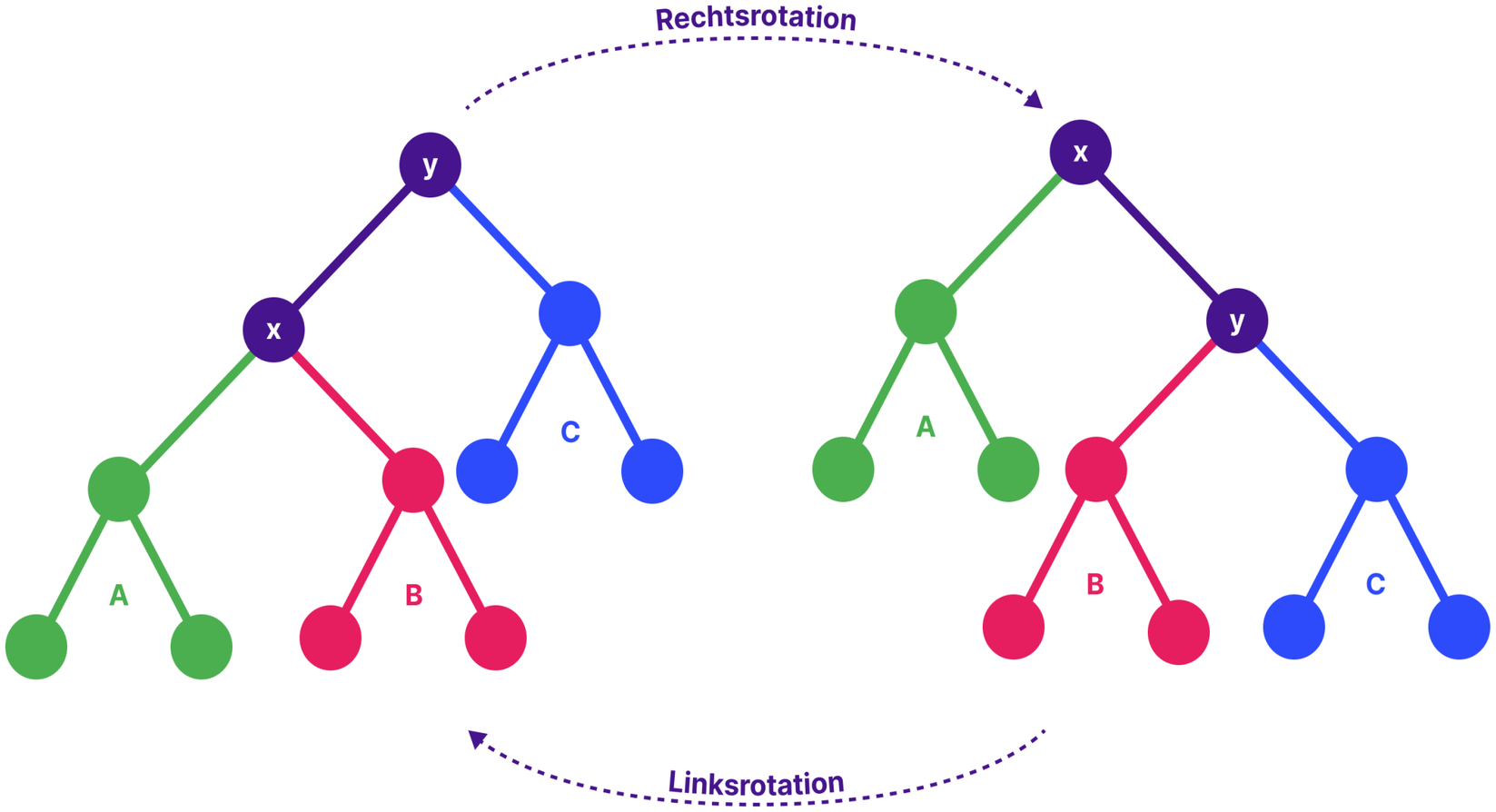


BF = -2 ⟶ Baum im Ungleichgewicht



Wie können wir den Baum ins Gleichgewicht bringen?
Durch Rotation!

Rotation

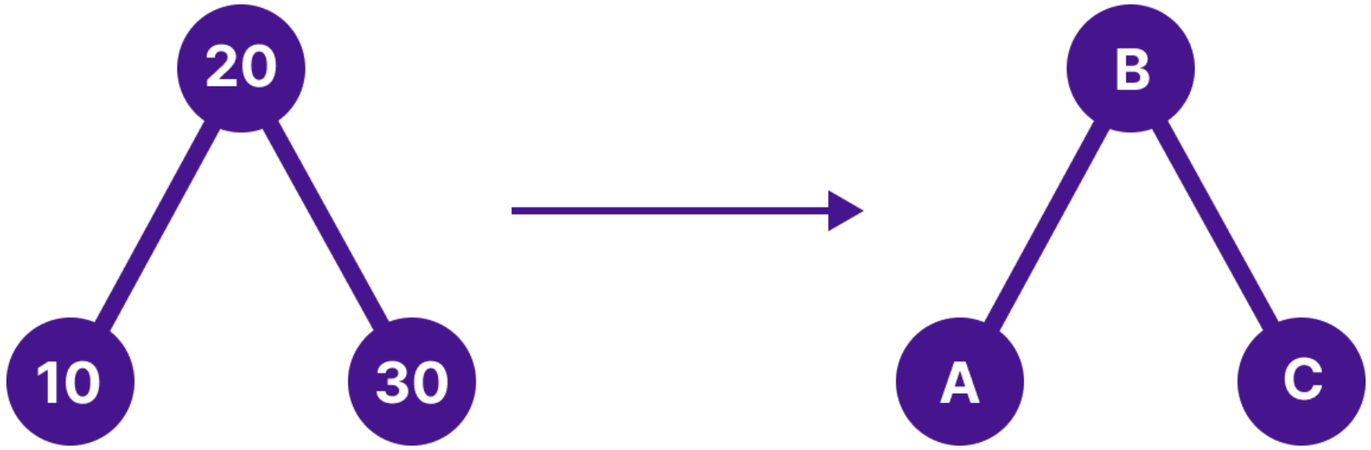


Rechtsrotation: x wird zur Wurzel und y zum rechten Teilbaum von x.

Linksrotation: y wird zur Wurzel und x zum linken Teilbaum von y.

Teilbaum B ändert dabei den Elternknoten!

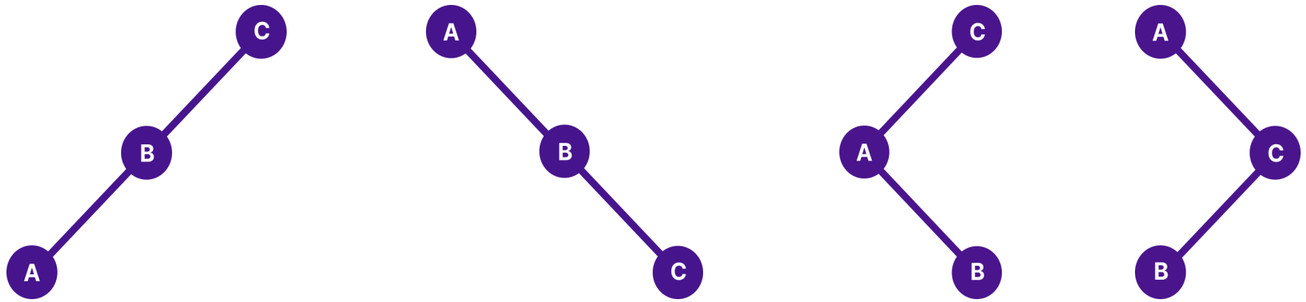
Hinweis



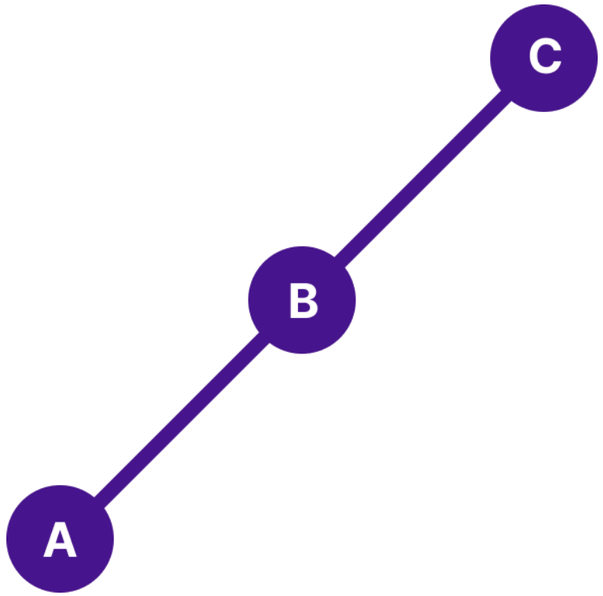
Damit das Erklären leichter fällt, wechseln wir von Ziffern zu Buchstaben.

Glücklicherweise können wir alphabetisch sortieren.

Die vier Fälle von "Unwucht"

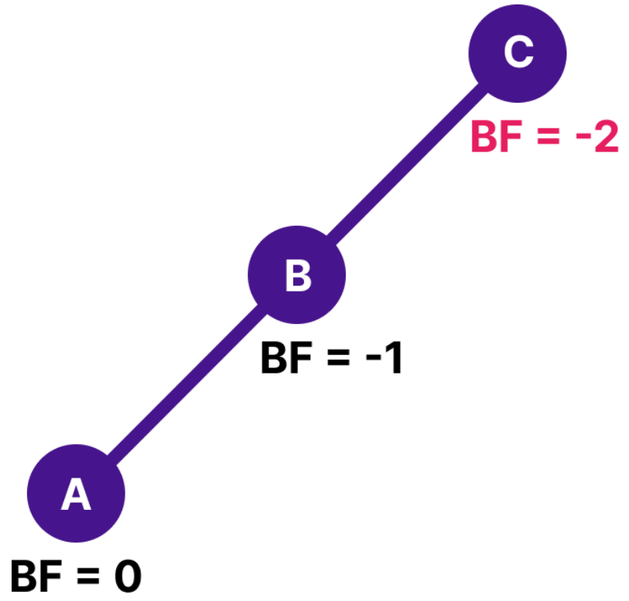


Fall 1: Links-Links



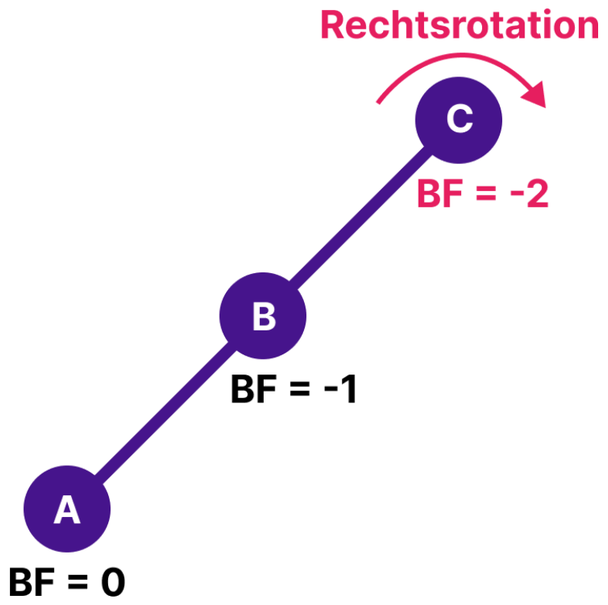
Was heißt "Links-Links"?

Fall 1: Links-Links



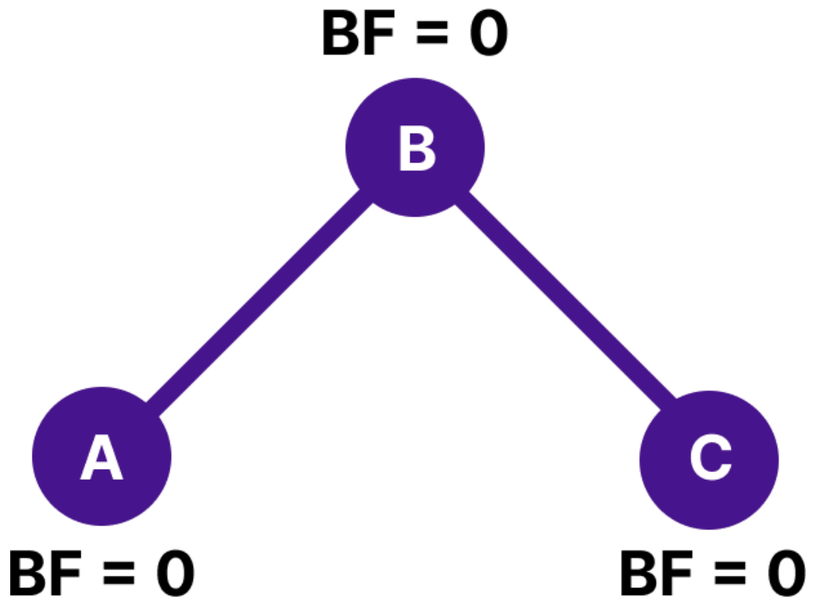
Der unausgeglichene Knoten ist linkslastig und das linke Kind ist ebenso linkslastig.

Fall 1: Links-Links



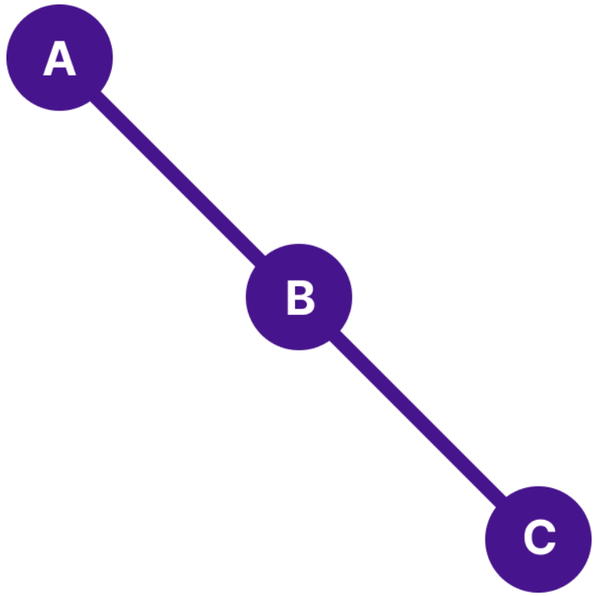
Eine Rechtsrotation auf dem unausgeglichenen Knoten genügt.

Fall 1: Links-Links



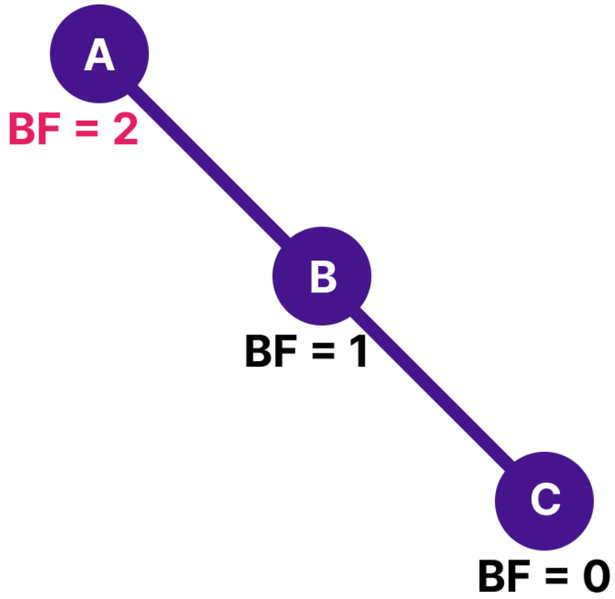
Für alle Knoten gilt nun $BF \in \{-1, 0, 1\}$

Fall 2: Rechts-Rechts



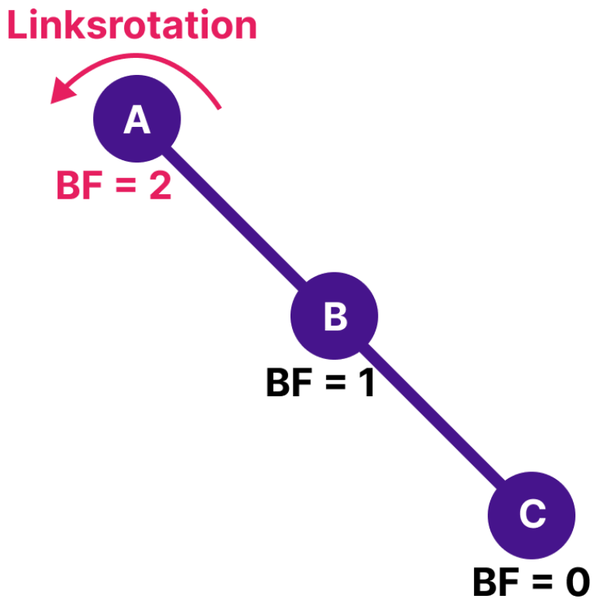
Vorgehensweise analog zum Fall 1.

Fall 2: Rechts-Rechts



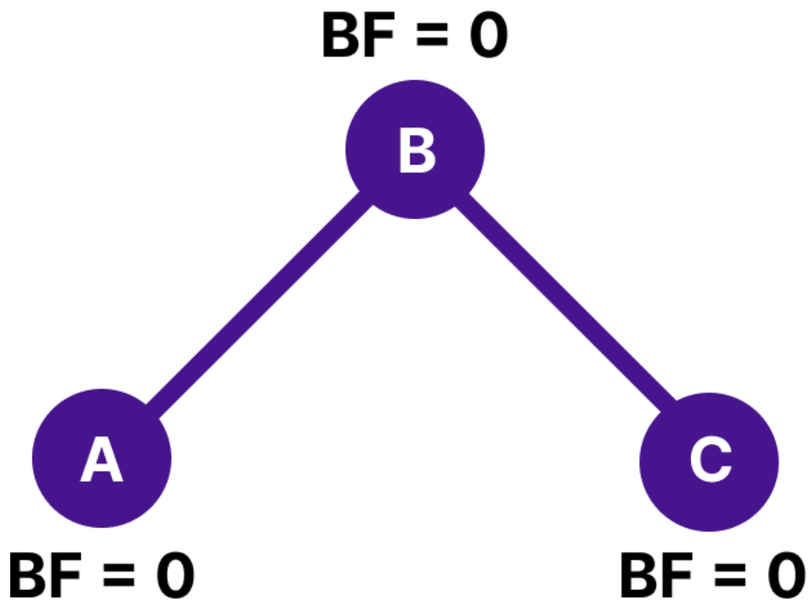
Der unausgeglichene Knoten ist rechtslastig und das rechte Kind ist ebenso rechtslastig.

Fall 2: Rechts-Rechts

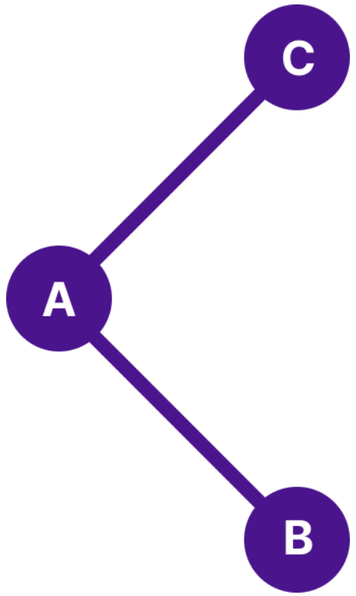


Eine Linksrotation auf dem unausgeglichenen Knoten genügt.

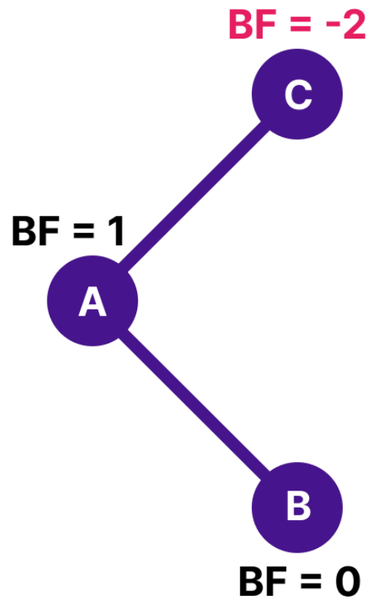
Fall 2: Rechts-Rechts



Fall 3: Links-Rechts

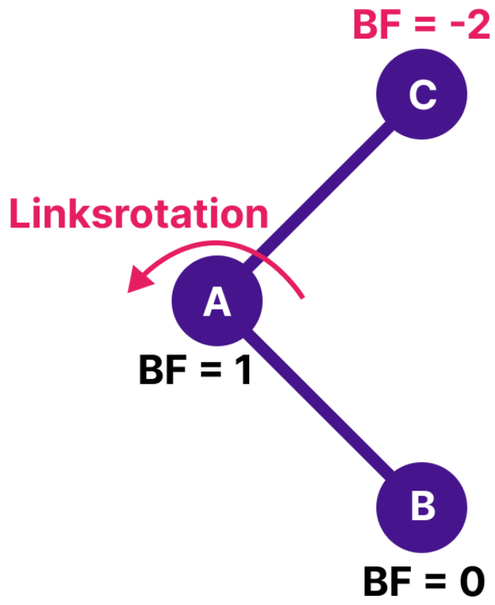


Fall 3: Links-Rechts



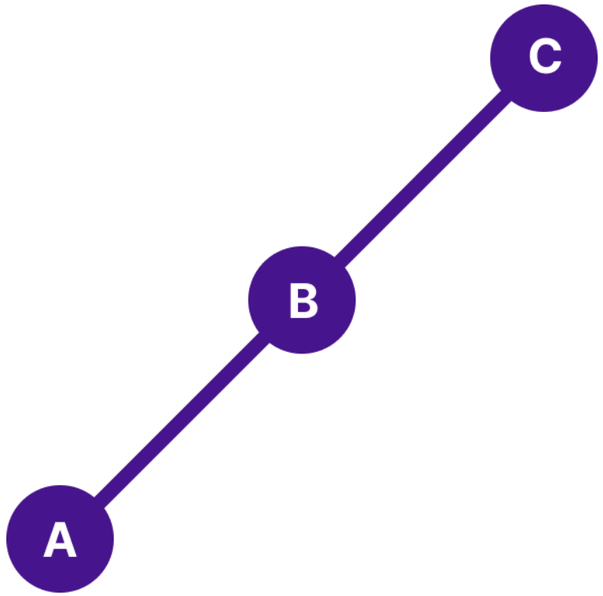
Der unausgeglichene Knoten ist linkslastig und das linke Kind ist rechtslastig.

Fall 3: Links-Rechts



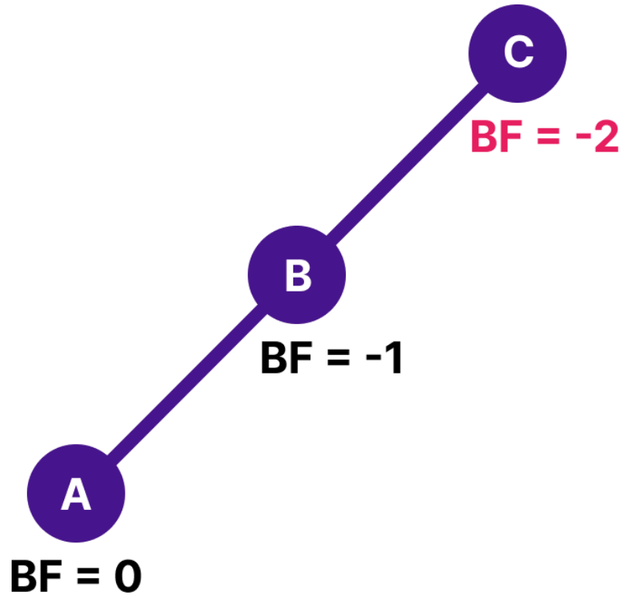
Wir führen eine Linksrotation am linken Kind durch.

Fall 3: Links-Rechts



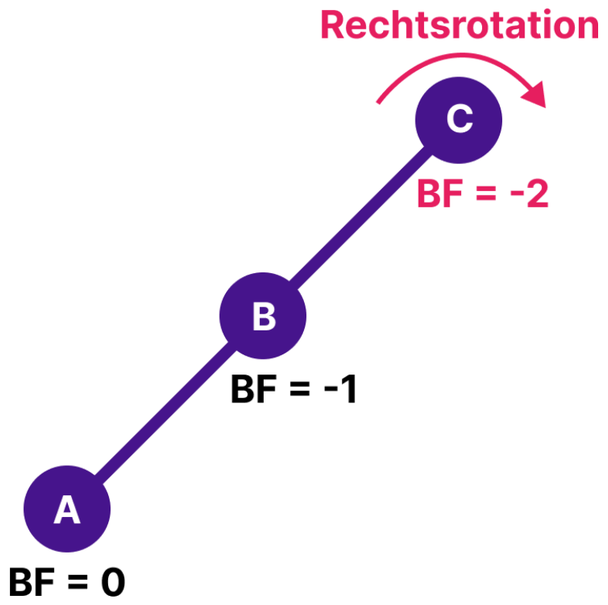
An welchen Fall erinnert uns das? **Fall 1: Links-Links**

Fall 3: Links-Rechts



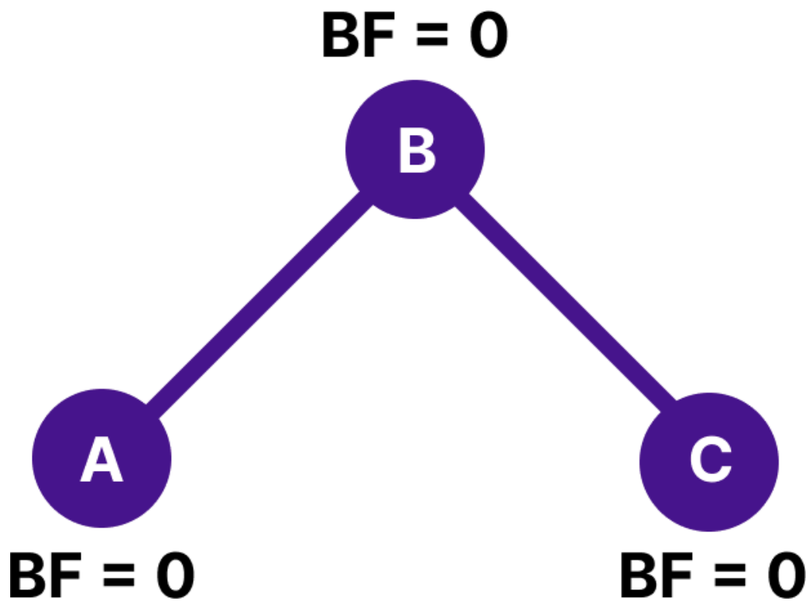
(Zur Wiederholung)

Fall 3: Links-Rechts

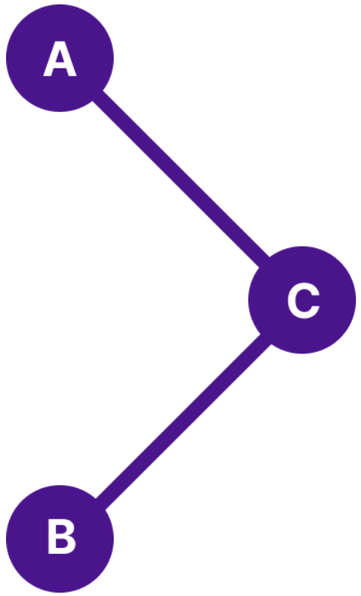


(Zur Wiederholung)

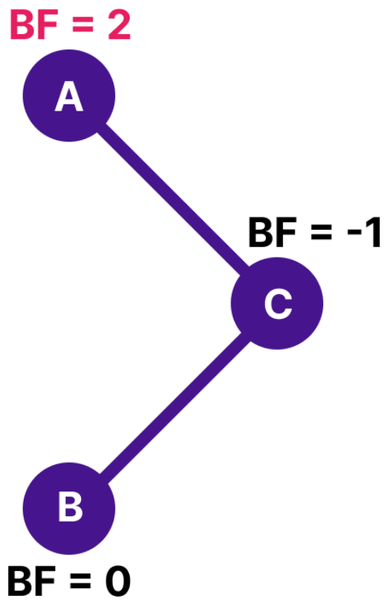
Fall 3: Links-Rechts



Fall 4: Rechts-Links

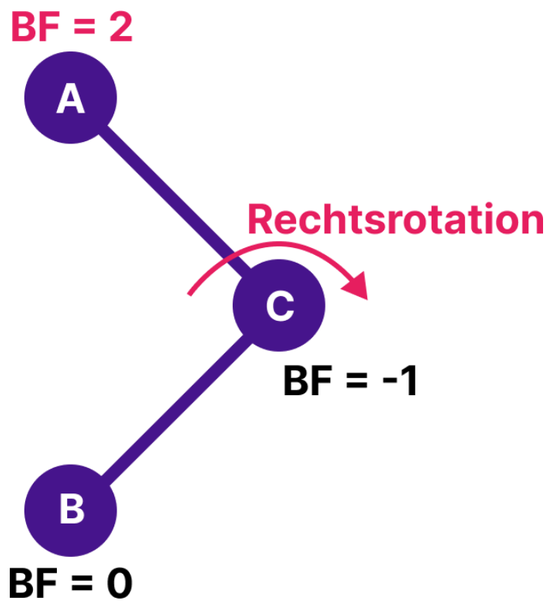


Fall 4: Rechts-Links



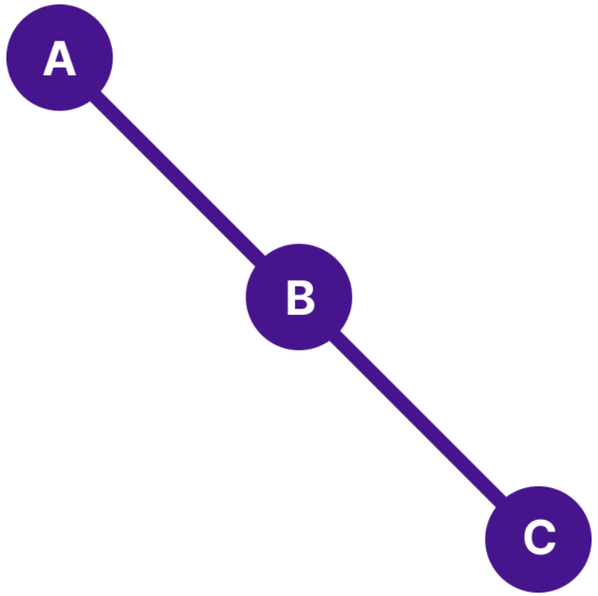
Der unausgeglichene Knoten ist rechtslastig und das rechte Kind ist linkslastig.

Fall 4: Rechts-Links



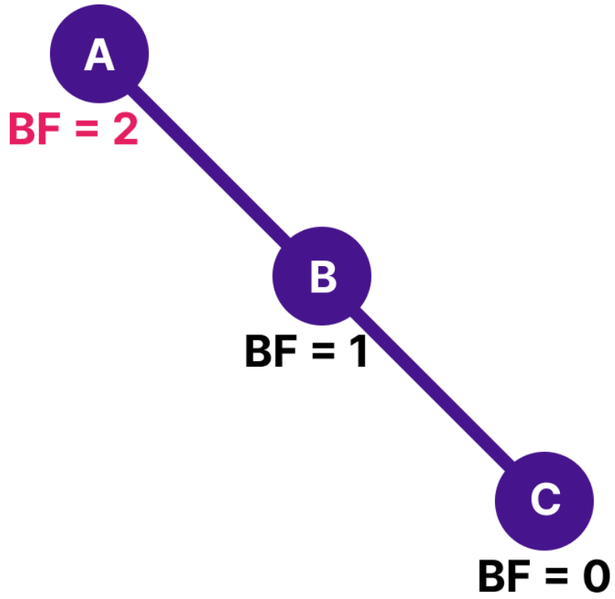
Wir führen eine Rechtsrotation am rechten Kind durch.

Fall 4: Rechts-Links



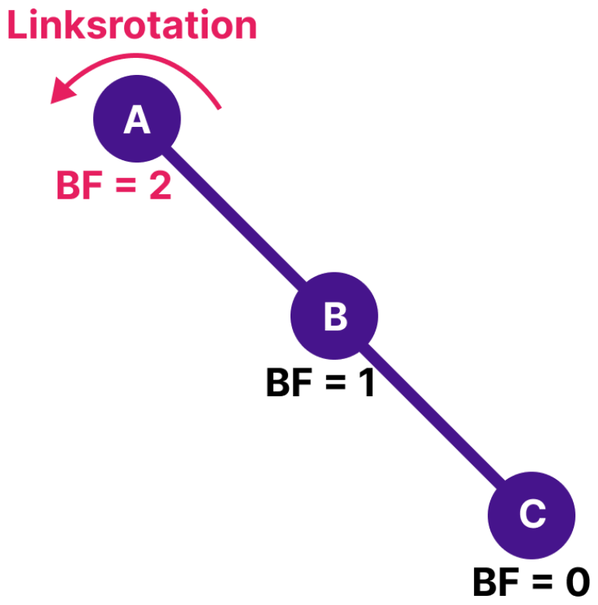
An welchen Fall erinnert uns das? **Fall 2: Rechts-Rechts**

Fall 4: Rechts-Links



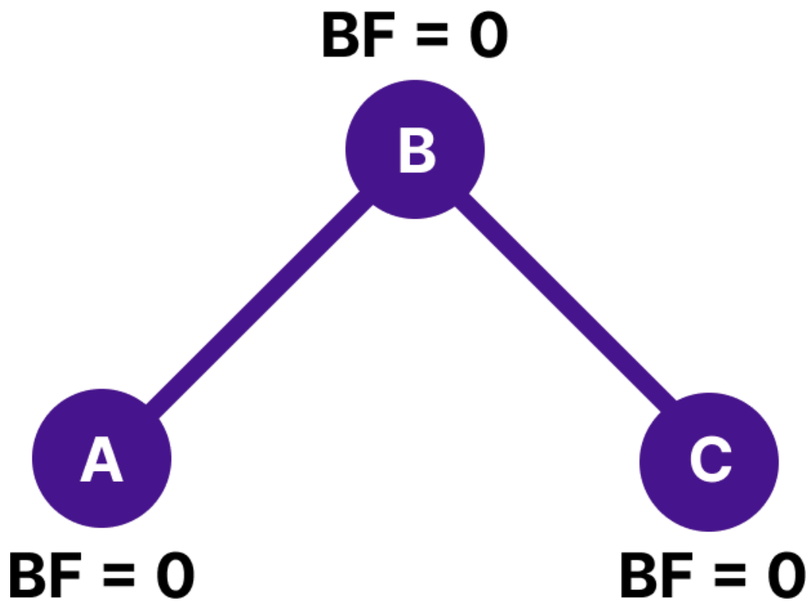
(Zur Wiederholung)

Fall 4: Rechts-Links

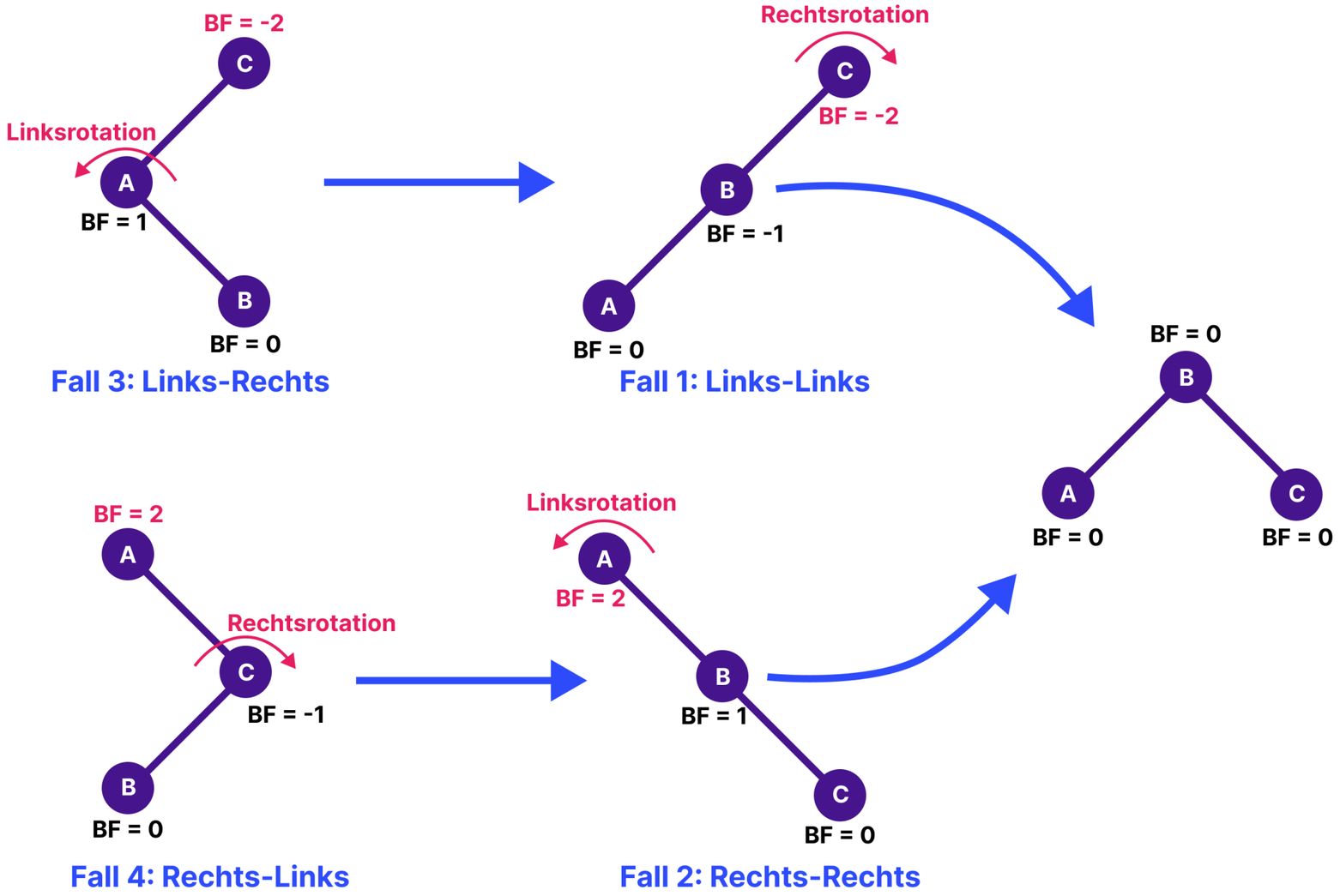


(Zur Wiederholung)

Fall 4: Rechts-Links



Die vier Fälle im Überblick



Zurück zu unserer sortierten Liste

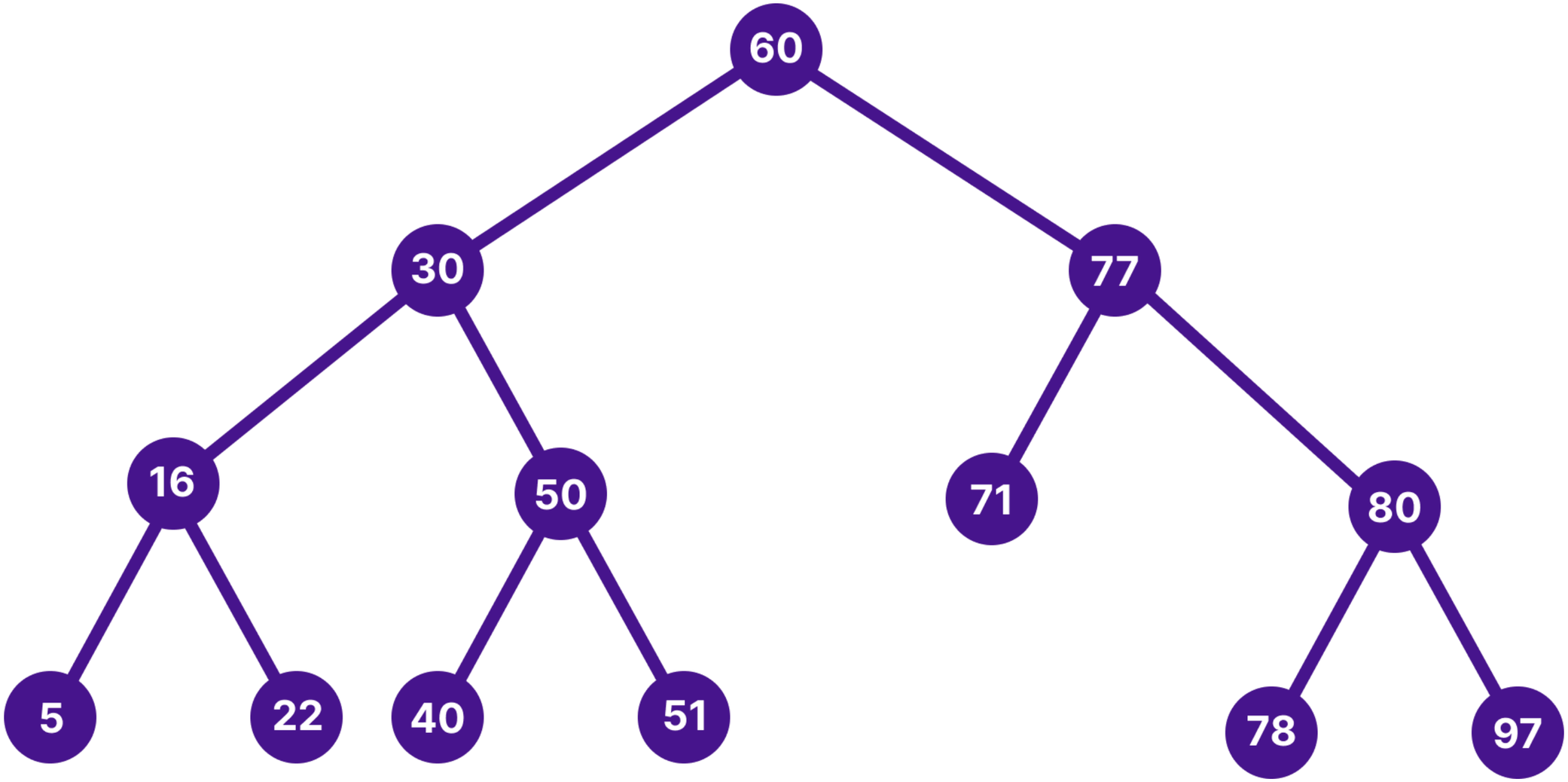
`5` `16` `22` `30` `40` `50` `51` `60` `71` `77` `78` `80` `97`

Wie konstruieren wir daraus einen AVL Baum?

Vorgehensweise

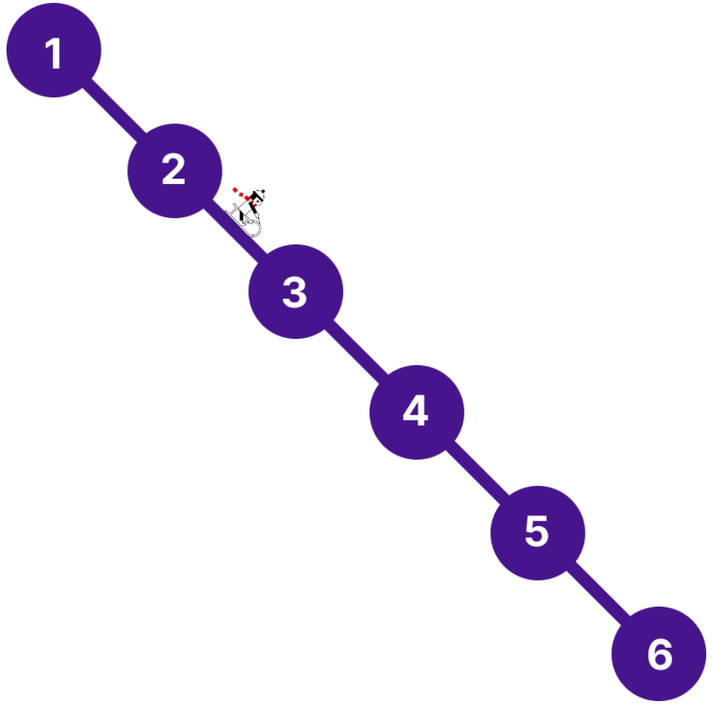
1. Element aus der Liste einlesen.
2. Einfügen in einen binären Suchbaum.
3. **AVL Balancierung:**
    1. Balance-Faktoren entlang des Pfads aktualisieren
    2. Analysieren ob Unwucht vorliegt.
    3. Falls Unwucht vorliegt, mit einem der vier Fälle korrigieren.
    4. Solange auf Unwucht prüfen und korrigieren, bis der Baum balanciert ist.
4. Wiederhole Schritte 1 bis 3 bis die Liste leer ist.

Ergebnis



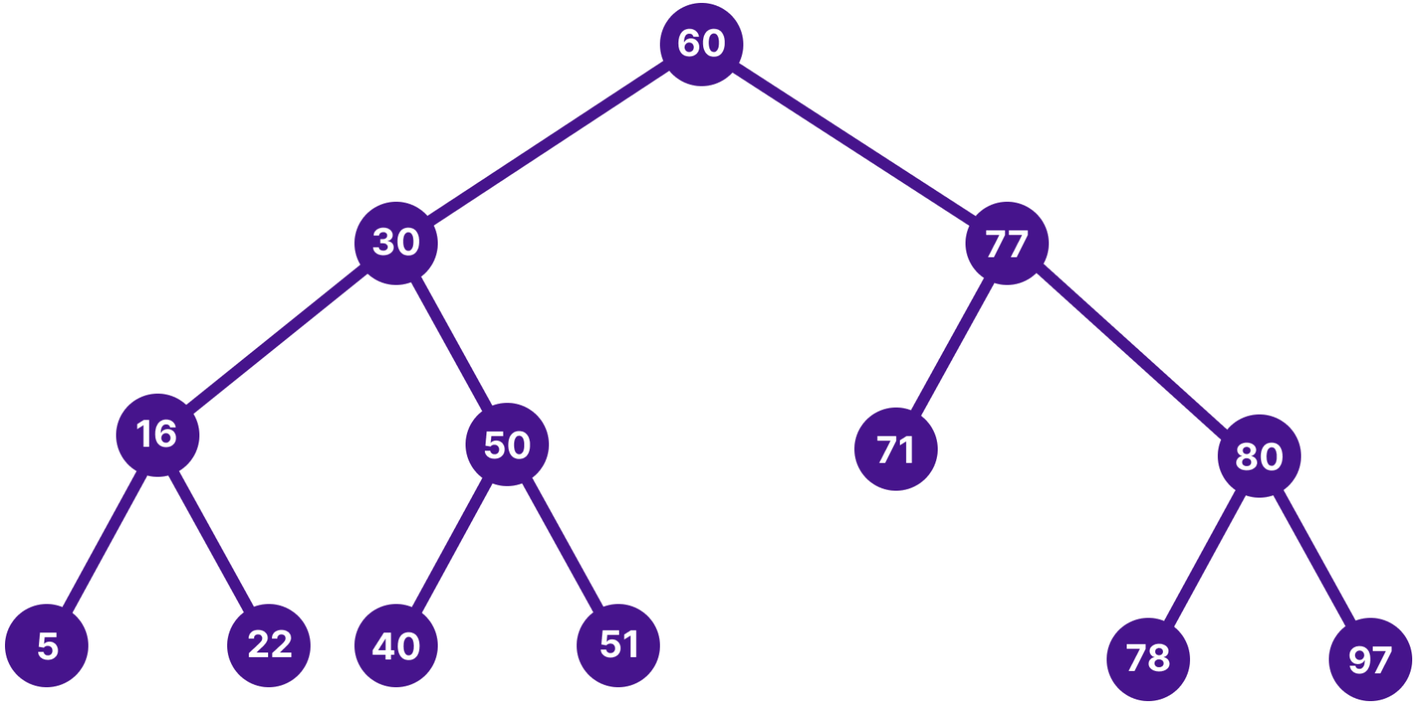
Laufzeit Vergleich: Suchen

Binärer Suchbaum



$\mathcal{O}(n)$

AVL Suchbaum



$\mathcal{O}(\log n)$

Was kostet das Balancieren?

- Einfügeposition suchen: $\mathcal{O}(\log n)$
- Balance-Faktoren entlang des Pfads aktualisieren: $\mathcal{O}(\log n)$
- Rotieren: $\mathcal{O}(1)$

Das gesamte Balancieren kostet also $\mathcal{O}(\log n)$

Das Balancieren verschlechtert die asymptotische Laufzeit des Einfügens nicht.

AVL-Bäume investieren zusätzliche Arbeit beim Einfügen, um die Höhe logarithmisch zu halten und dadurch effizientes Suchen im Worst Case zu garantieren.

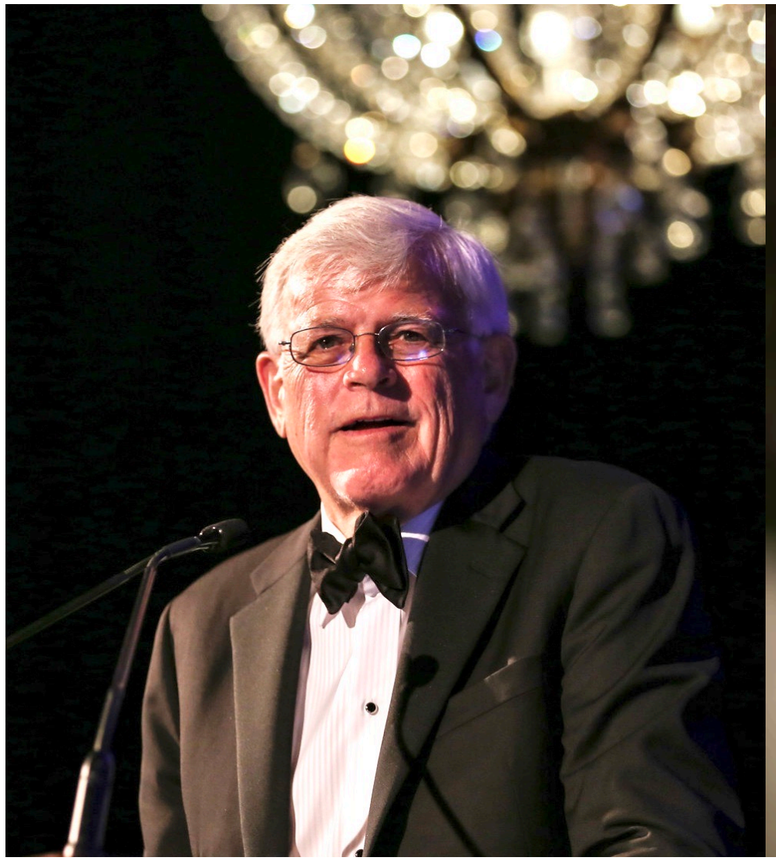
Binäre Suchbäume

Rot-Schwarz Bäume

Rot-Schwarz Bäume

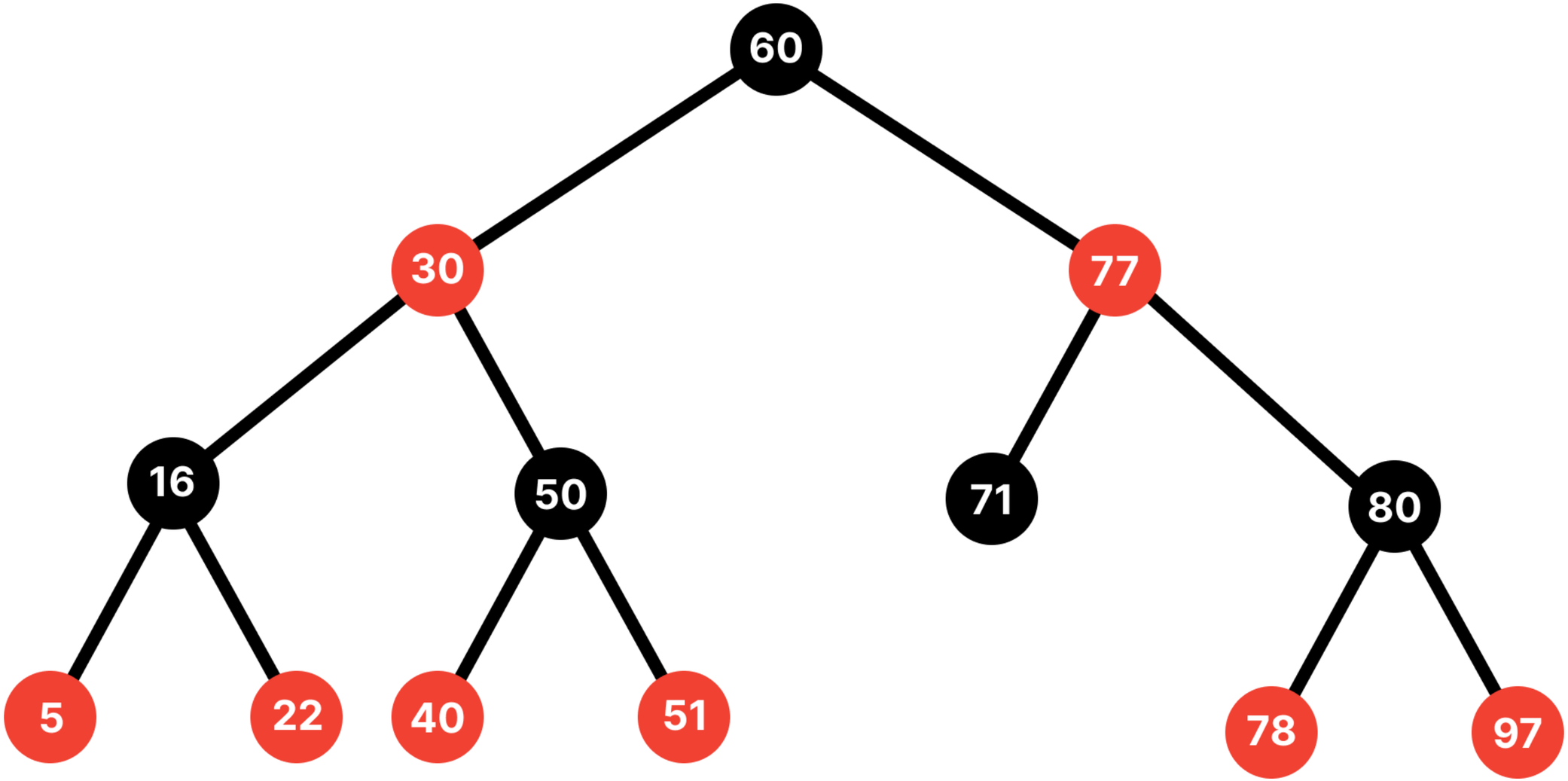
Entworfen 1978 von Prof. Robert Sedgewick und Prof. Leonidas Guibas

Bildquellen: sedgewick.io und me.sh/profile/leonidas-guibas





Jeder Knoten ist entweder Rot oder Schwarz



Trivia: Warum die Farben Rot und Schwarz?

Sedgewick:

"But one of the things that was invented there [at Xerox PARC] was laser printing and we were very excited to have nearby color laser printer that could print things out in color and out of the colors the red looked the best. So, that's why we picked the color red to distinguish red links, the types of links, in three nodes."

https://en.wikipedia.org/wiki/Red%E2%80%93black_tree#cite_note-Sedgewick12-8

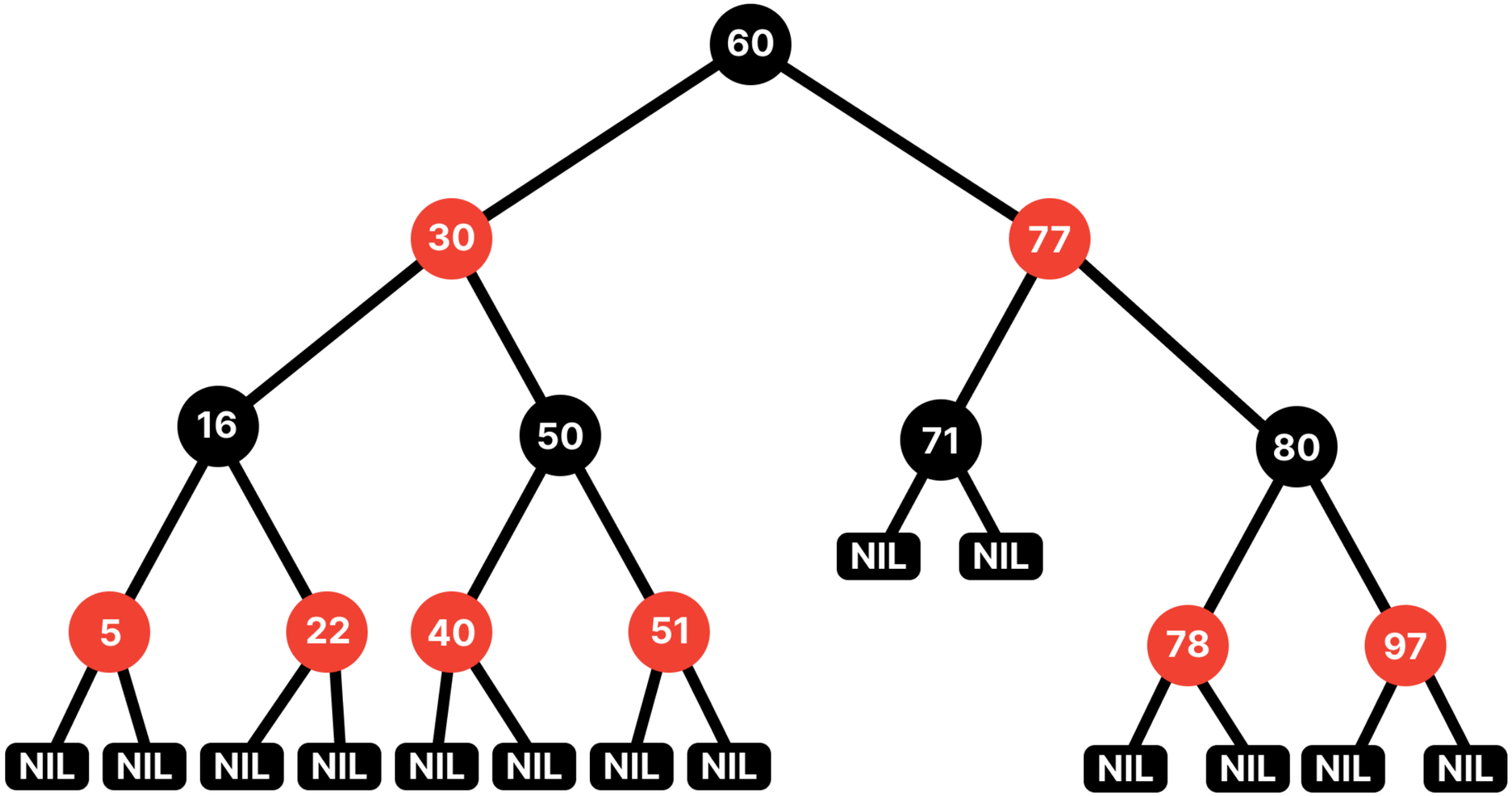
Guibas:

"we had red and black pens for drawing the trees."

https://softwareengineering.stackexchange.com/a/116621

Jedes Blatt ist ein schwarzer NIL Knoten

In der Informatik heißt NIL = "Leer"

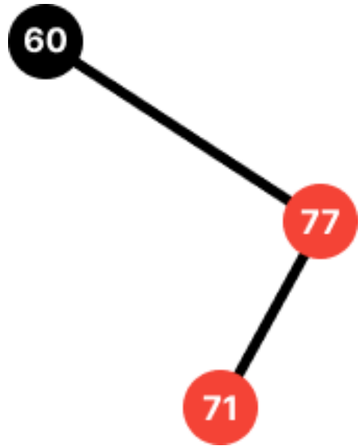


Eigenschaften

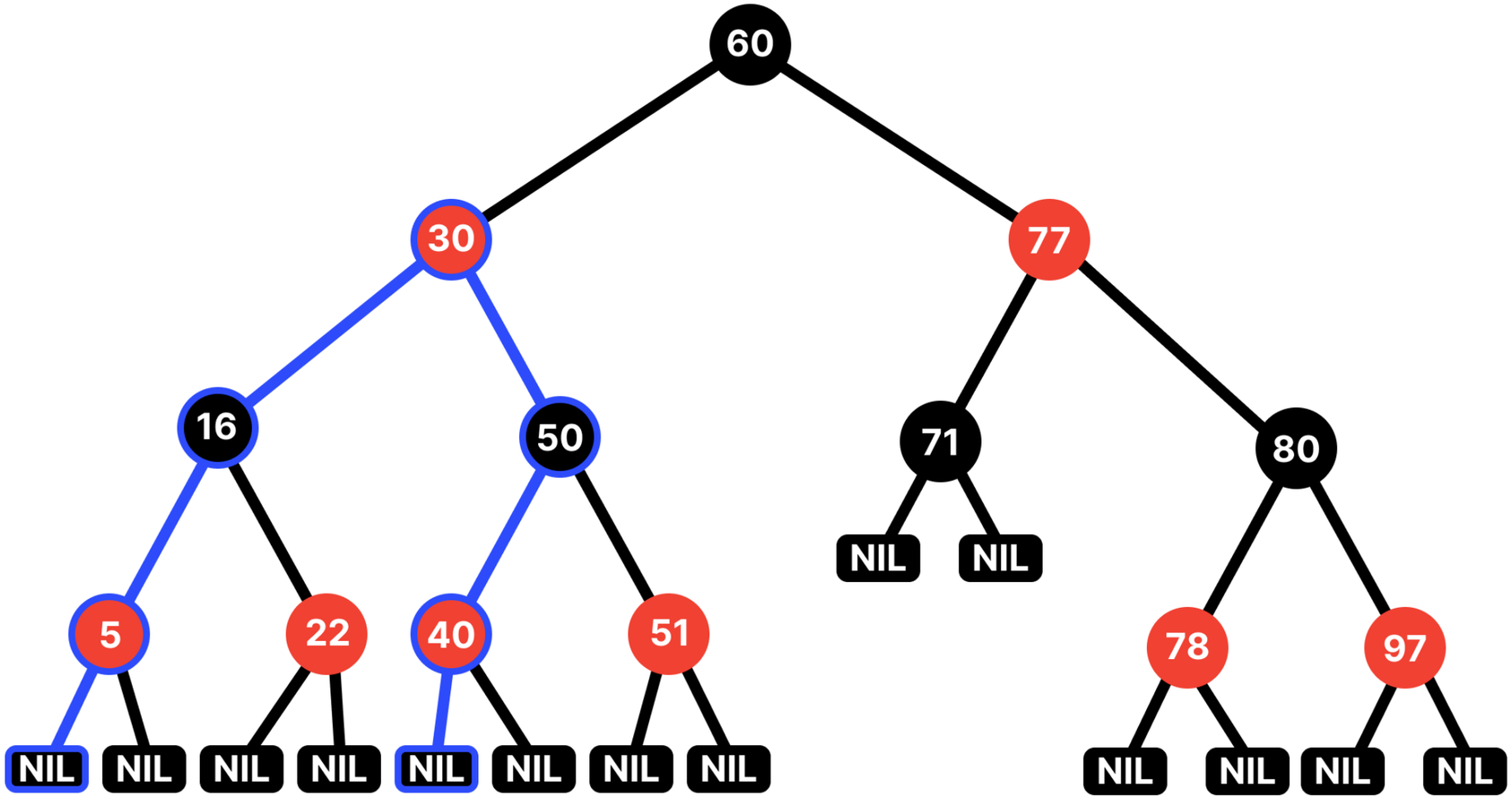
1. Jeder Knoten ist entweder **rot** oder **schwarz**.
2. Die Wurzel ist immer **schwarz**.
3. Jedes Blatt (NIL) ist **schwarz**.
4. Wenn ein Knoten **rot** ist, müssen beide Kinder **schwarz** sein.
5. Jeder Pfad von einem Knoten zu seinen nachfolgenden NIL Knoten muss die gleiche Anzahl an **schwarzen** Knoten haben.

Nicht erlaubt

Zwei aufeinanderfolgende rote Knoten sind nicht erlaubt.



Eigenschaft 5 illustriert



Diese Eigenschaft sorgt für die Balance in Rot-Schwarz Bäumen.

Knoten einfügen

Neue Knoten sind immer **rot**



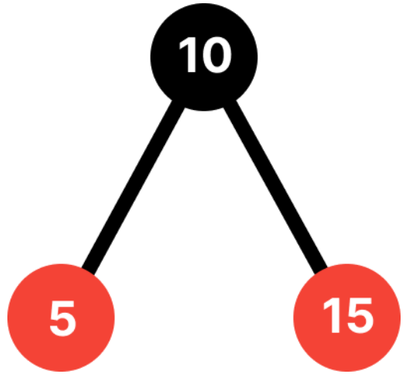
Knoten einfügen

Wenn der neue Knoten die Wurzel ist, färben wir ihn **schwarz**.



Knoten einfügen

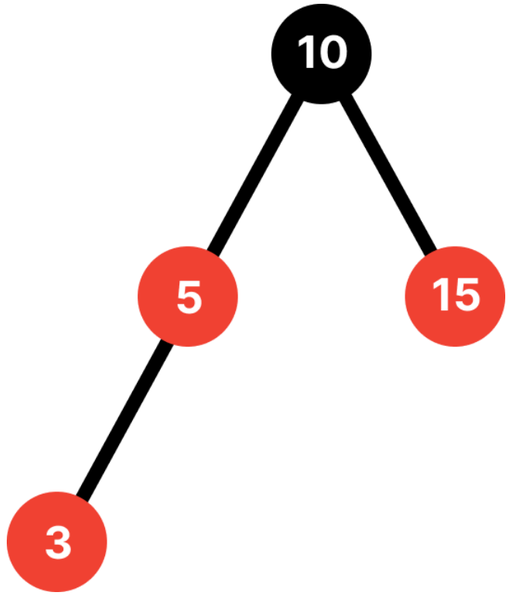
Wir fügen nun 5 und 15 ein.



Bei **schwarzen** Elternknoten können **rote** Knoten immer ohne Probleme angehängt werden.

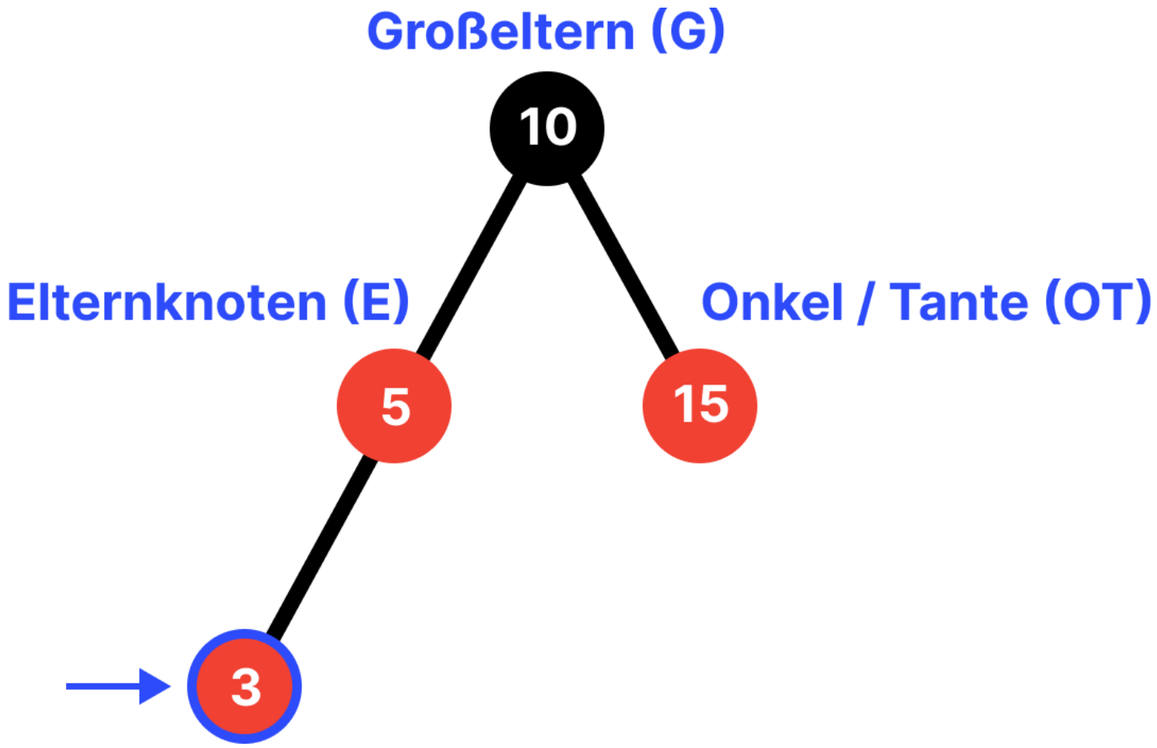
Knoten einfügen

Wir fügen nun 3 ein und sehen, dass wir zwei aufeinanderfolgende **rote** Knoten haben.



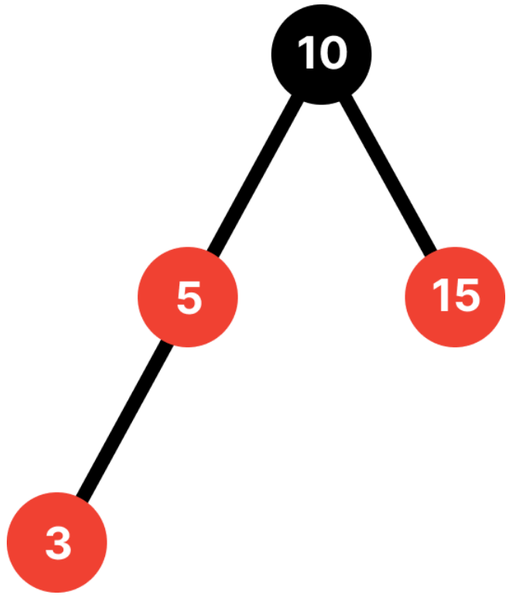
Bei einem **roten** Elternknoten müssen wir einen Konflikt lösen.

Stammbaum Refresher



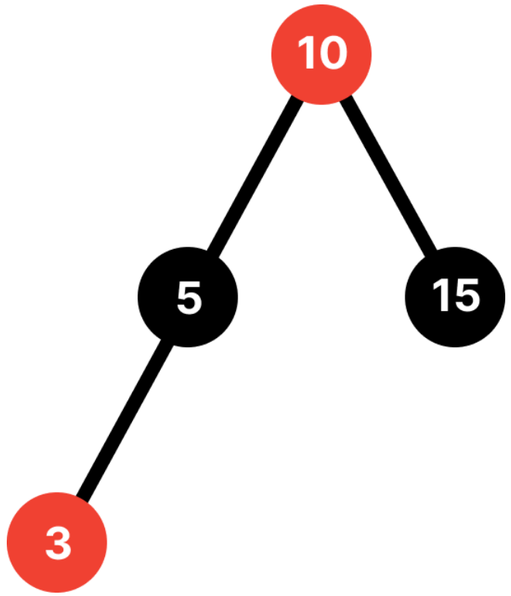
Wir werden nun häufig den Onkel / Tante Knoten mittels OT referenzieren.

Fall 1: OT ist rot



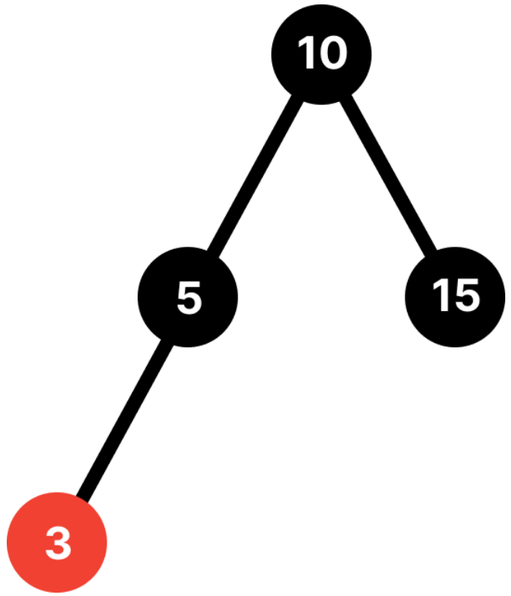
E und OT werden **schwarz** gefärbt, G wird **rot** gefärbt.

Wurzel ist nun rot



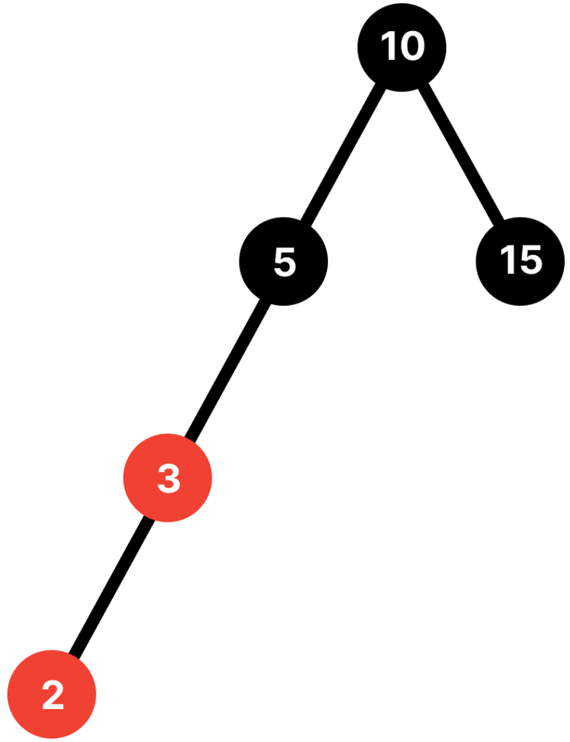
Daher wird die Wurzel **schwarz** gefärbt.

Zwischenergebnis nach Fall 1



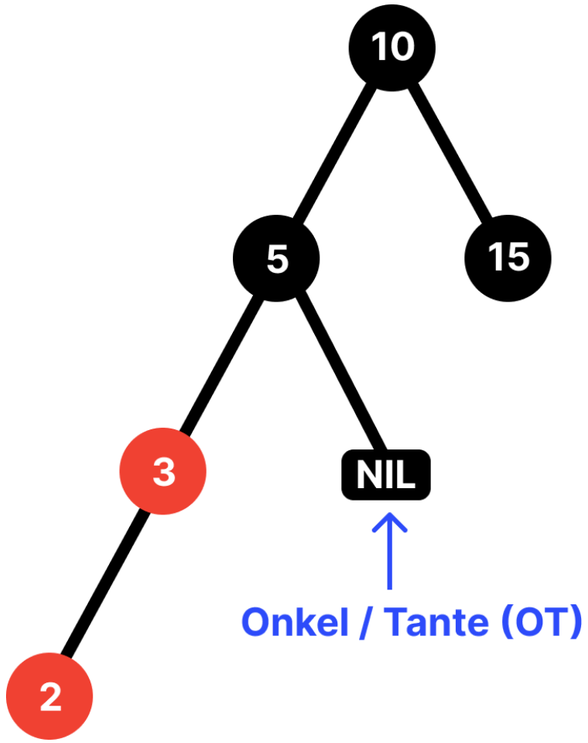
Nächsten Knoten einfügen

Wir fügen nun 2 ein



Der Elternknoten ist **rot** und somit erneut ein Konflikt.

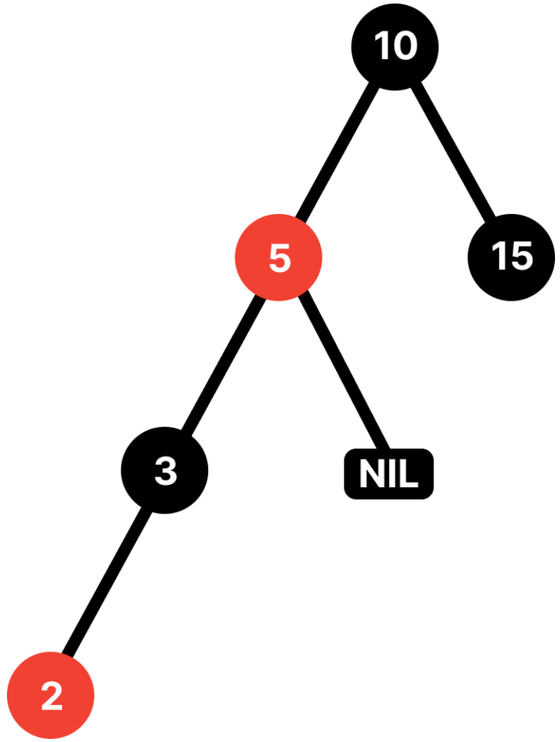
Der OT Knoten ist diesmal schwarz!



Zur Erinnerung: Jedes Blatt ist NIL und immer schwarz

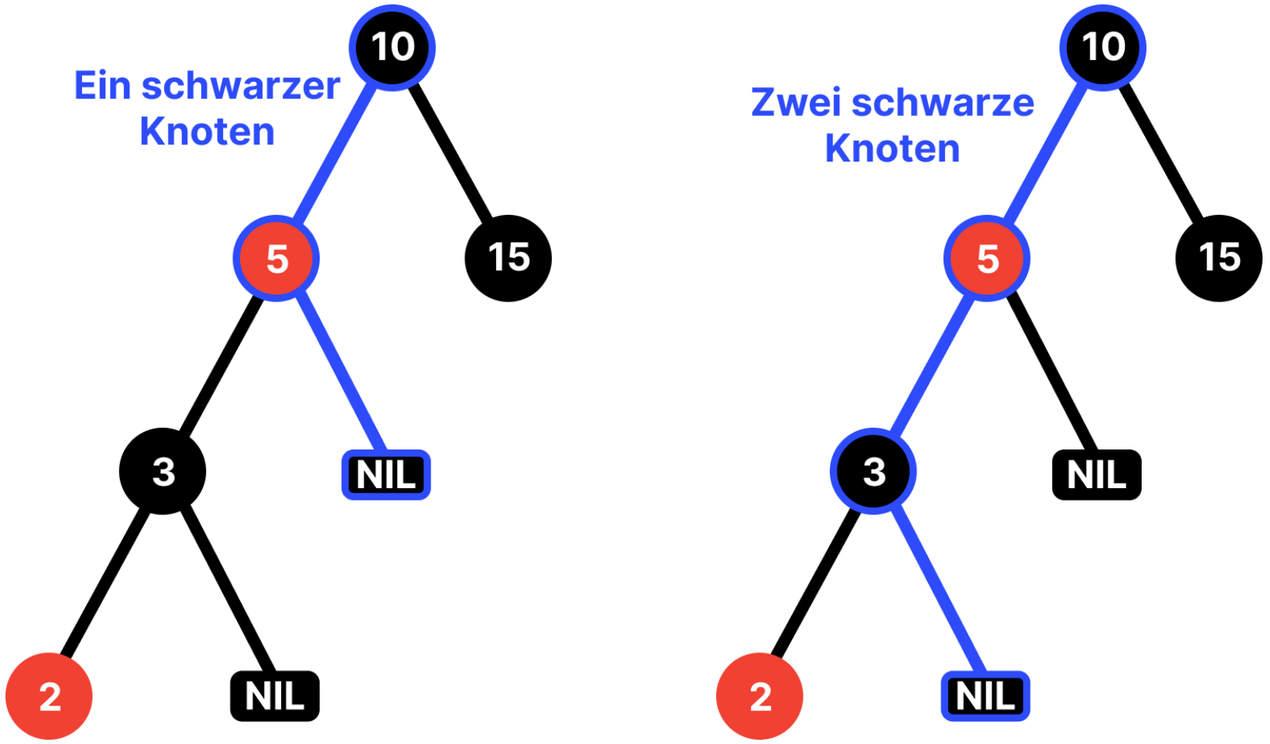
Fall 2: OT ist schwarz

E → schwarz, G → rot



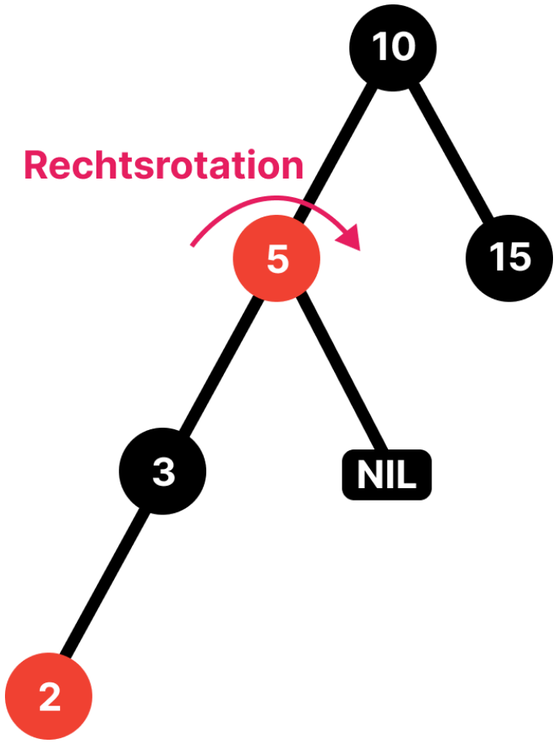
Erkennen Sie das Problem?

Vergleichen Sie diese beiden Pfade

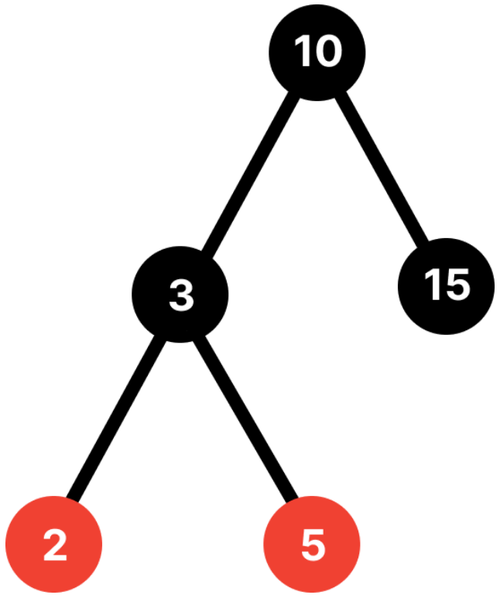


Eigenschaft 5: Jeder Pfad von einem Knoten zu seinen nachfolgenden NIL Knoten muss die gleiche Anzahl an **schwarzen** Knoten haben.

Konflikt durch Rotieren lösen

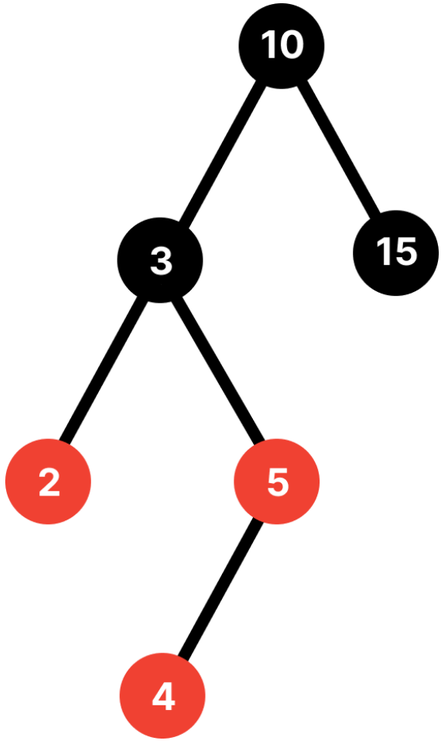


Zwischenergebnis nach Fall 2



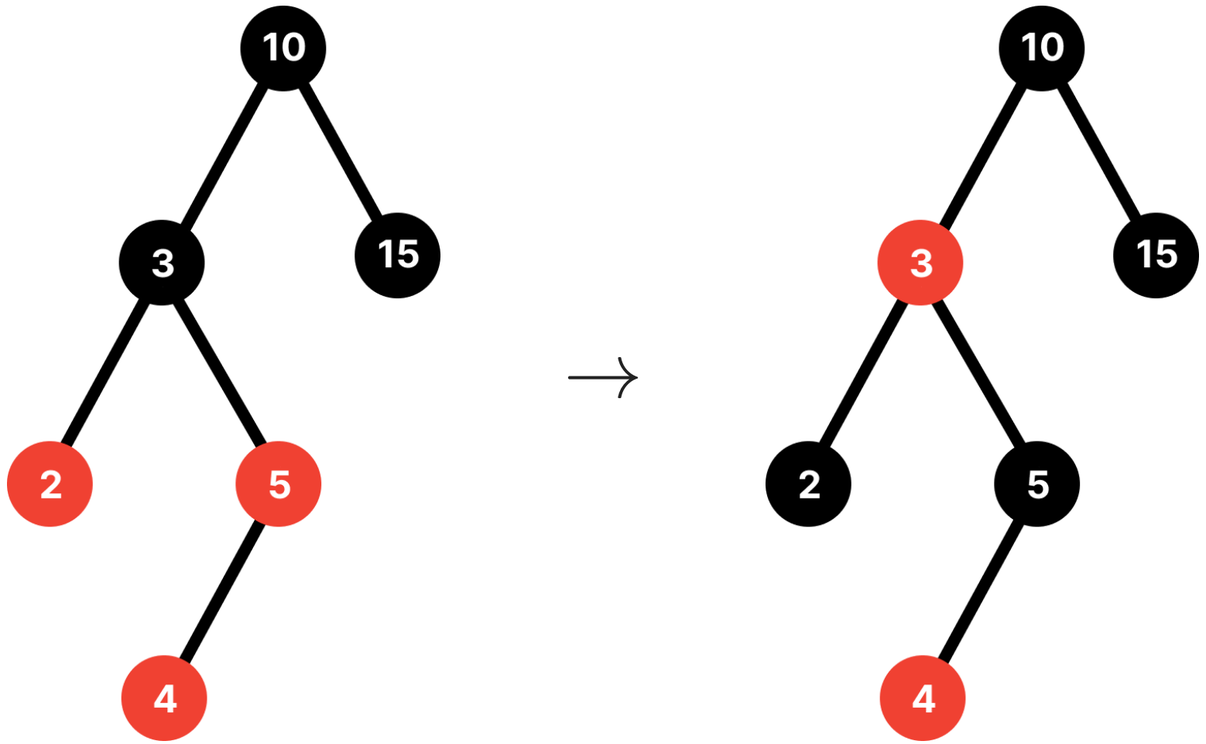
Nächsten Knoten einfügen

Wir fügen nun 4 ein.



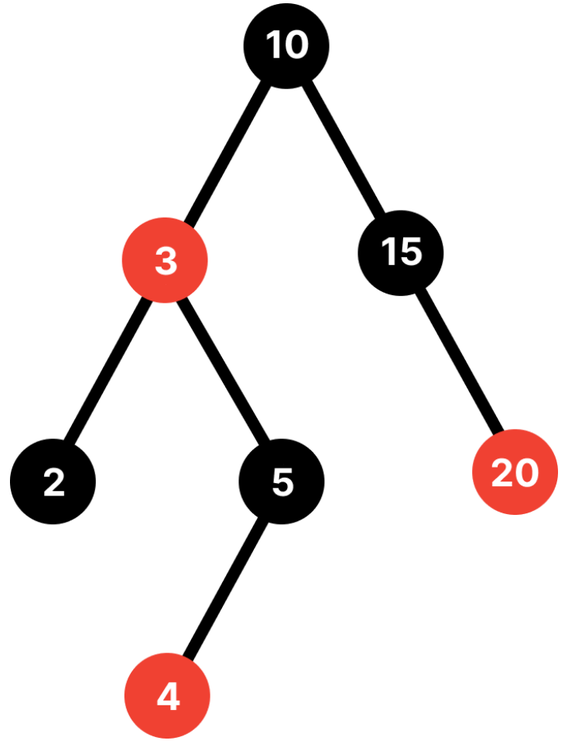
Konflikt: Fall 1

E und OT $\longrightarrow$ schwar, G $\longrightarrow$ rot



Nächsten Knoten einfügen

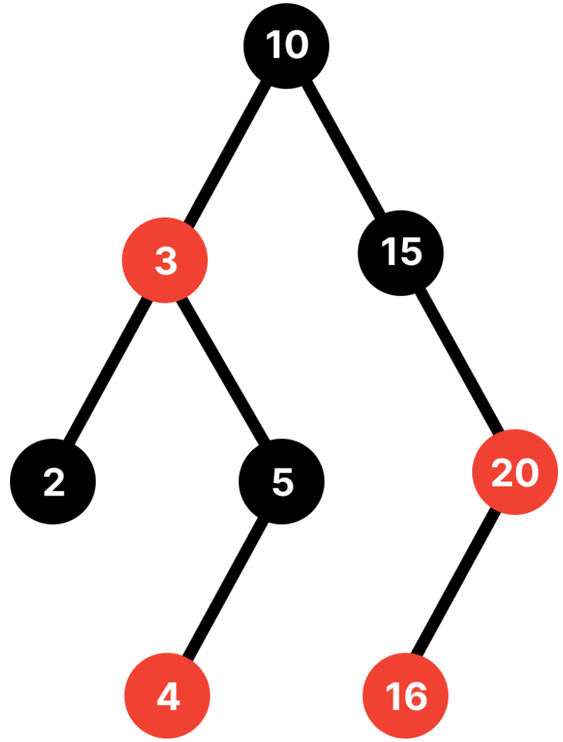
Wir fügen nun 20 ein.



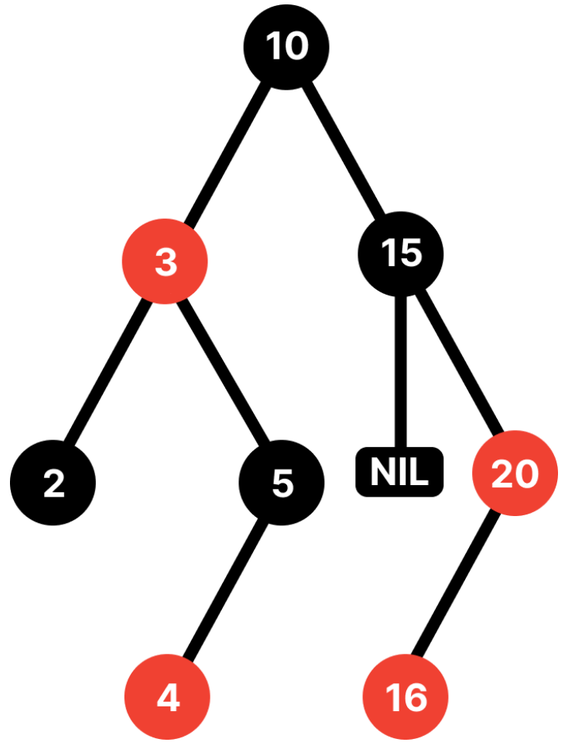
Kein Konflikt

Nächsten Knoten einfügen

Wir fügen nun 16 ein.



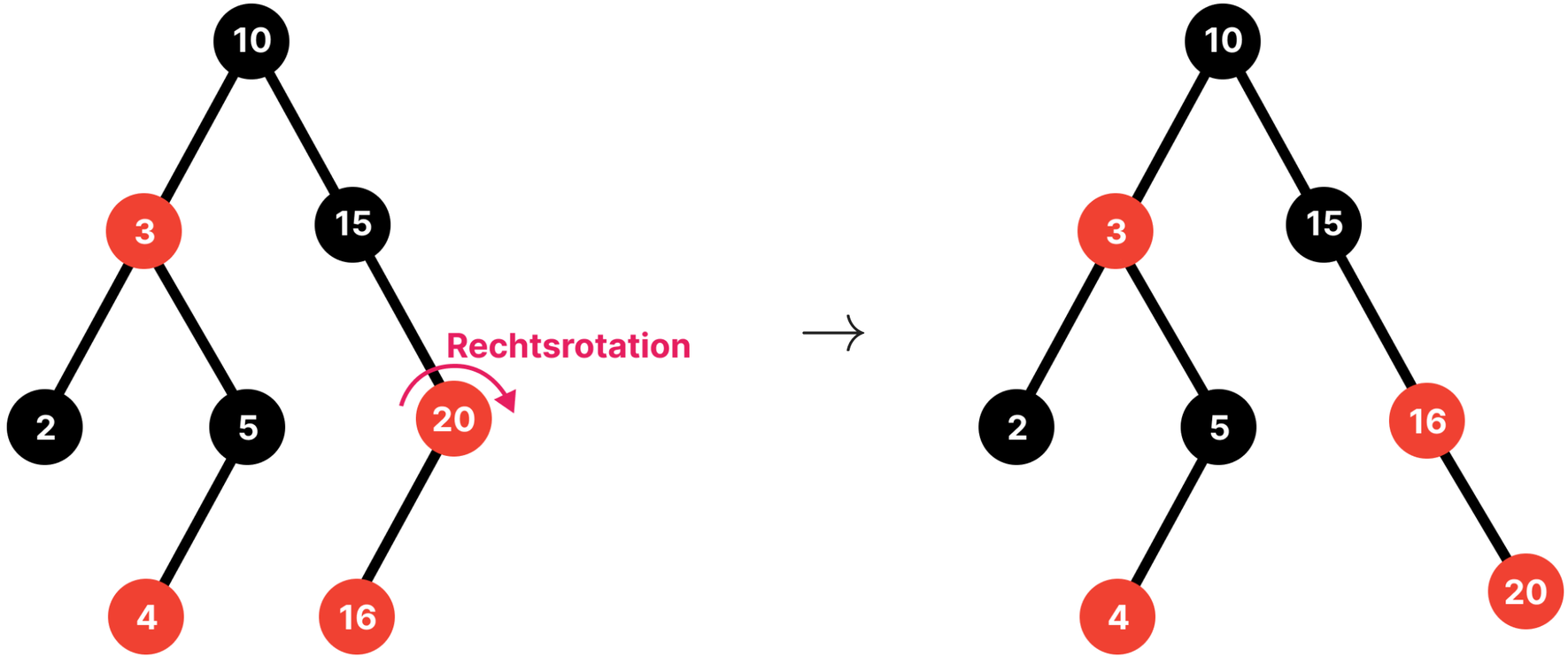
Konflikt



Fall 2 mit schwarzen OT Knoten

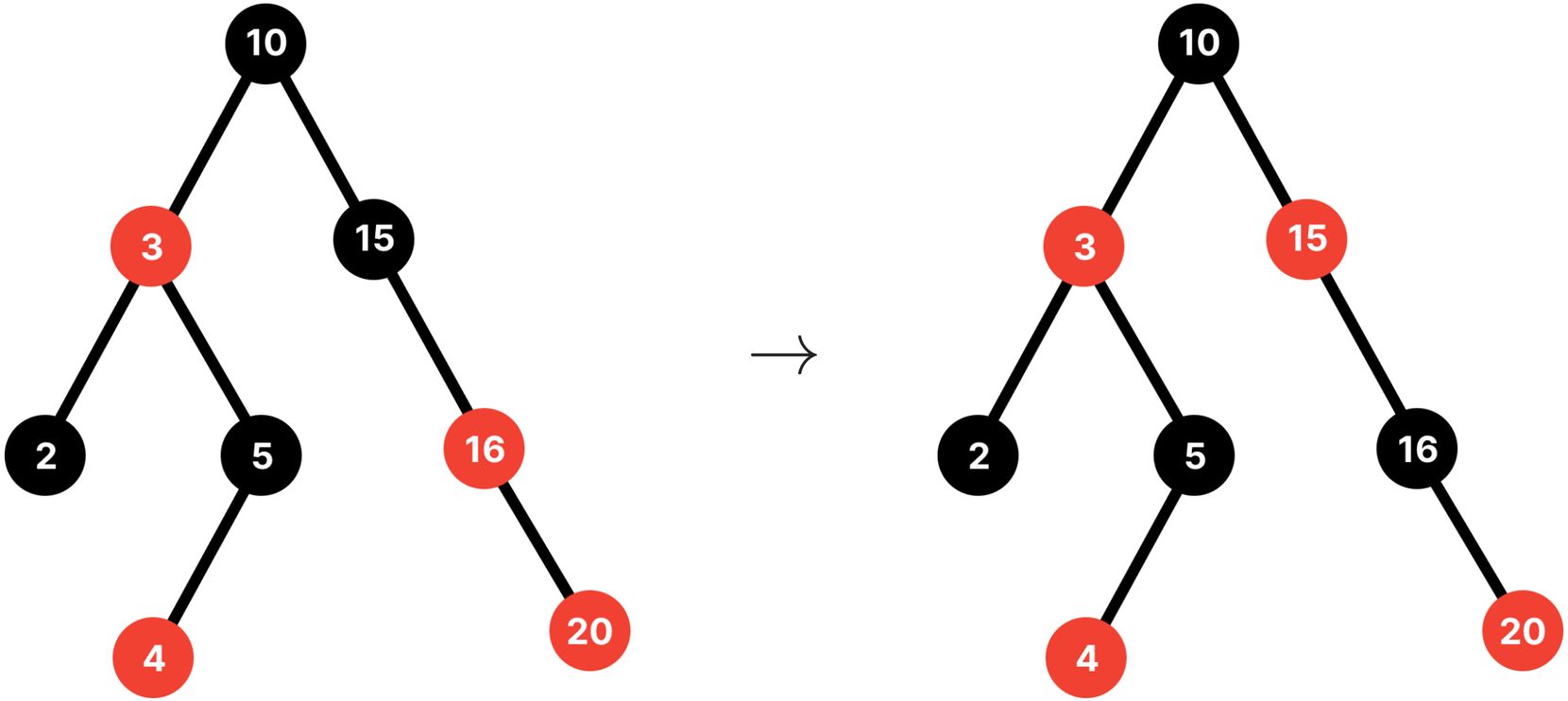
Zur Erinnerung: NIL Blätter sind immer schwarz!

Zunächst rotieren wir den Elternknoten nach rechts

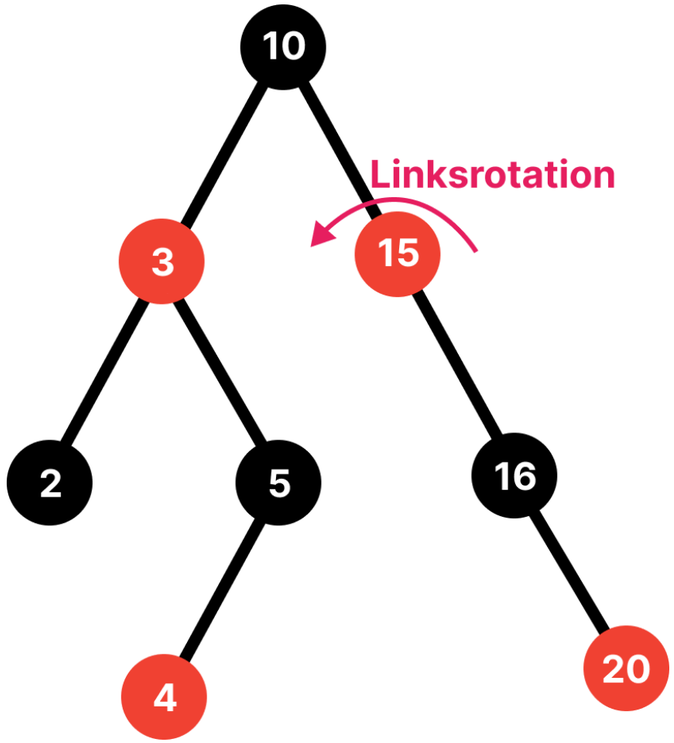


Dies ist eine Vorbereitung für die nachfolgende Rotation

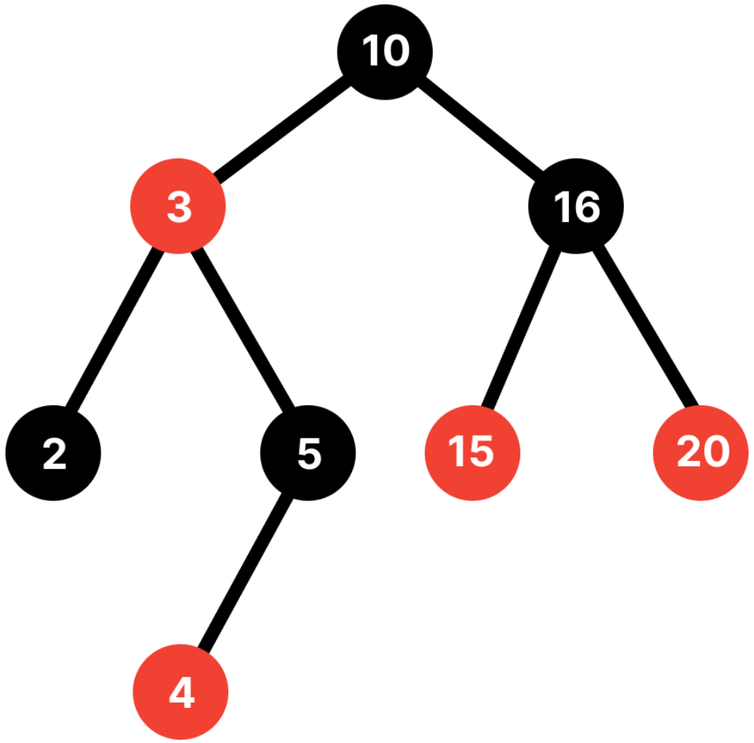
E → schwarz, G → rot



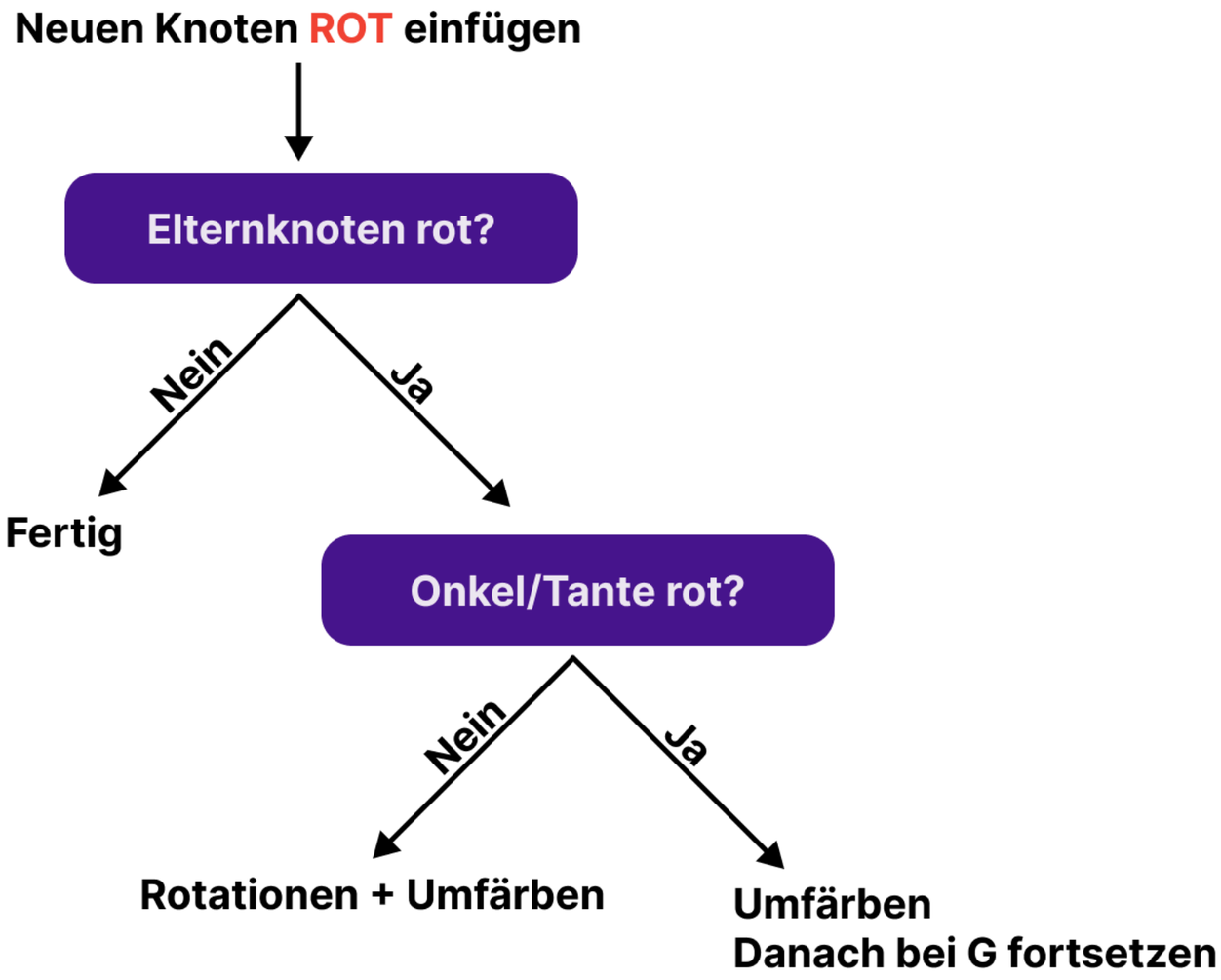
Wir rotieren G nach links, um die Balance wiederherzustellen



Endergebnis



Zusammenfassung



Die Laufzeiten im Überblick

Operation	Worst Case
Suchen	$\mathcal{O}(\log n)$
Einfügen	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(\log n)$

Rot-Schwarz-Bäume garantieren logarithmische Laufzeit im Worst Case.

Die Farbregeln beschränken die Höhe

Zwei Eigenschaften sind entscheidend:

- Jeder Pfad von der Wurzel zu einem NIL-Blatt enthält **gleich viele schwarze Knoten**
- **Zwei rote Knoten** dürfen nicht direkt aufeinanderfolgen.

Damit können zusätzliche rote Knoten einen Pfad höchstens verdoppeln.

Der längste Wurzel-Blatt-Pfad ist höchstens doppelt so lang wie der kürzeste.

Schwarzhöhe h_b

Die **Schwarzhöhe** h_b ist die Anzahl schwarzer Knoten auf einem Pfad von der Wurzel zu einem NIL-Blatt.

Jeder Pfad von der Wurzel zu einem NIL-Blatt hat dieselbe Schwarzhöhe.

NIL-Blätter werden nicht mitgezählt.

Warum folgt daraus eine logarithmische Höhe?

Ein Baum mit Schwarzhöhe h_b benötigt mindestens $2^{h_b} - 1$ innere Knoten.

Die Anzahl der Knoten wächst mindestens exponentiell mit der Schwarzhöhe.

$$n \geq 2^{h_b} - 1$$

Umgekehrt ist die Schwarzhöhe somit höchstens logarithmisch in n. Da $h \leq 2h_b$, folgt:

$$\boxed{h \leq 2\log_2(n+1)}$$

Also: $h = \mathcal{O}(\log n)$

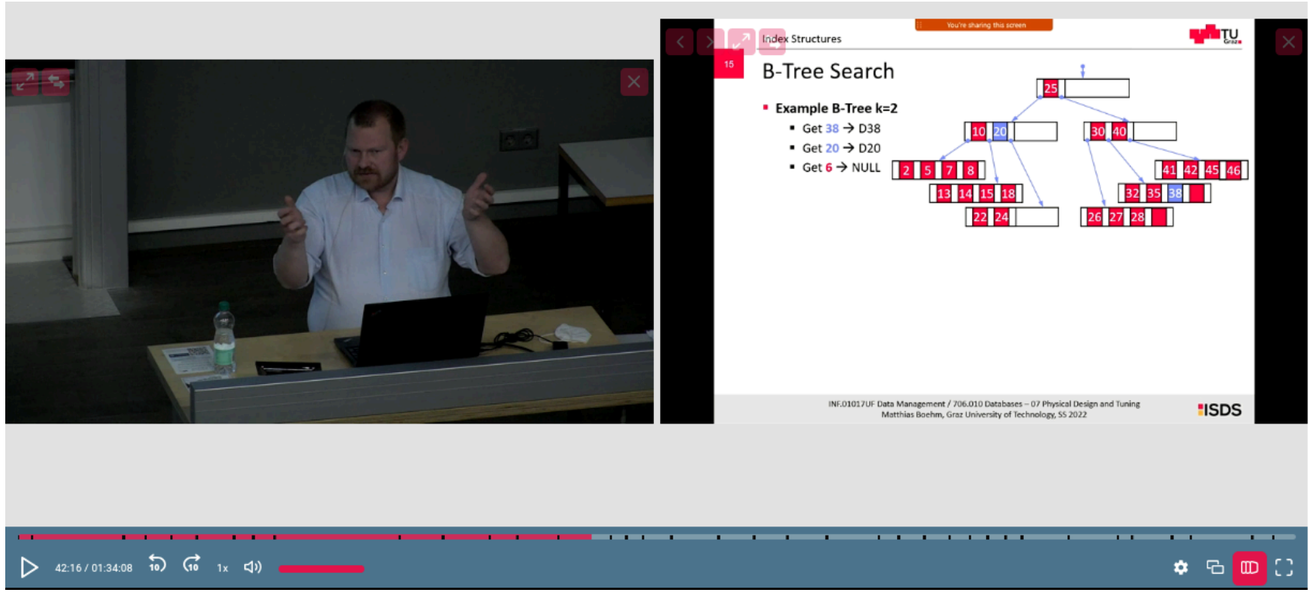
Mehrfach verzweigte Suchbäume

B-Bäume

Danksagung

Dieses Kapitel basiert auf der Datenbanken Vorlesung von Univ.-Prof. Dr. **Matthias Böhm**.

Wir danken ihm für die Bereitstellung der Unterlagen! 💜



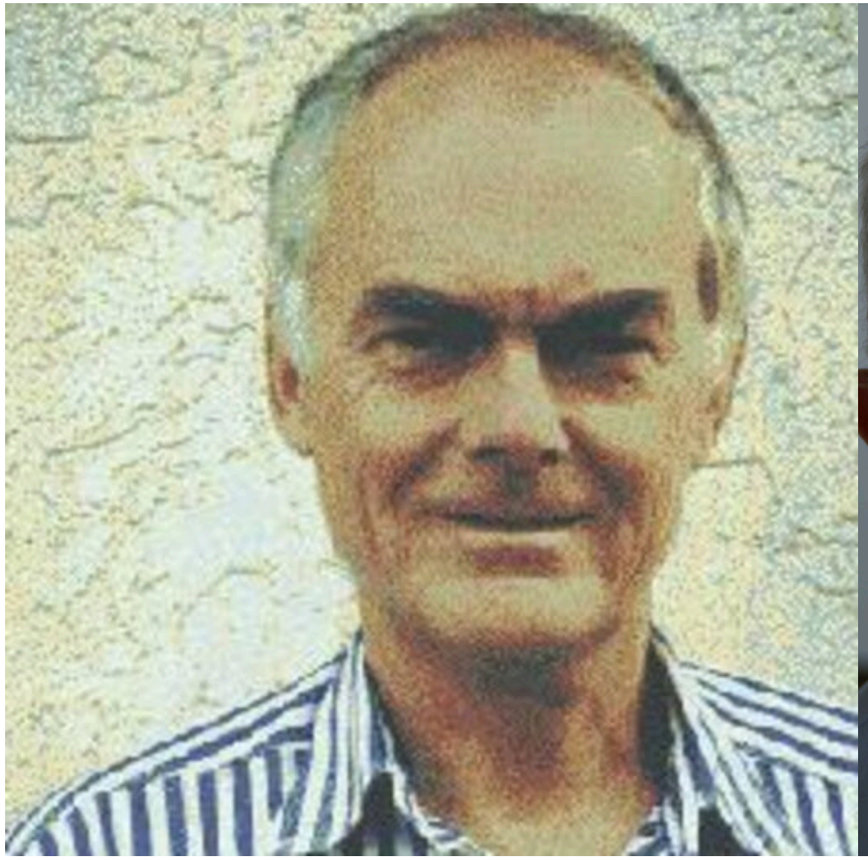
Seine Slides: mboehm7.github.io/teaching/ss22_dbs/07_PhysicalDesign.pdf

Video Aufzeichnung - **sehr sehenswert!** (ab ca. 00:28:00): tube.tugraz.at/portal/watch/a471d868-eb2a-4368-8566-fa5d10b2e1e9

B-Bäume

Entworfen 1972 von Prof. Rudolf Bayer und Prof. Edward McCreight

Bildquellen: researchgate.net/profile/Rudolf_Bayer/3 und mccreight.com/people/ed_mcc/index.htm





Wofür steht das "B"?

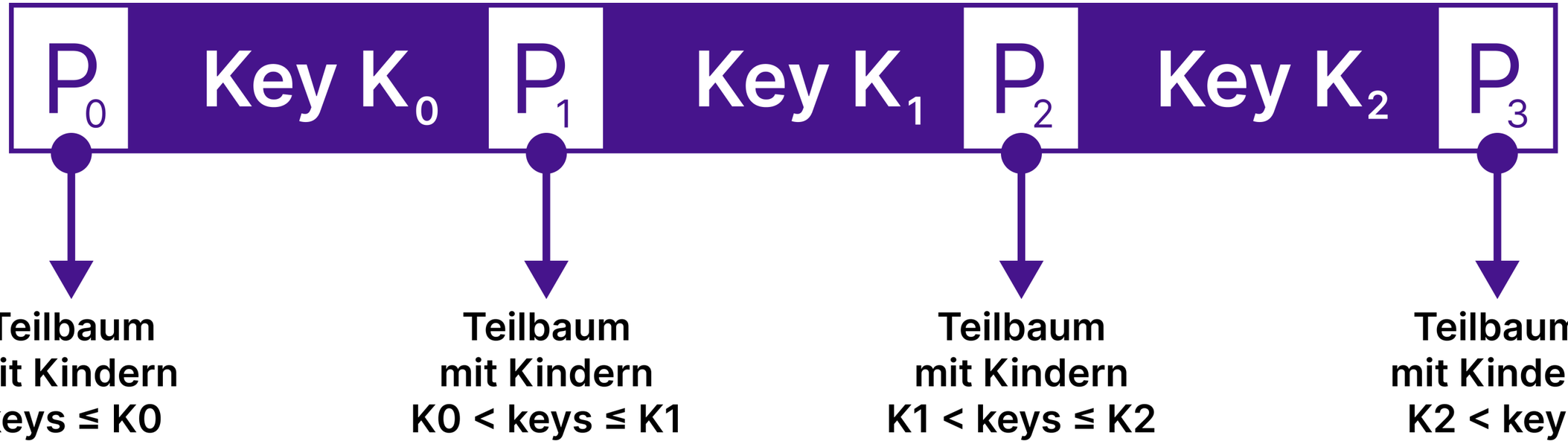
McCreight:

*"I am occasionally asked what the B in B-Tree means. [...] It honored our employer, **B**oeing Airplane Company [...]. It suggested **B**alance. Rudolf **B**ayer was the senior researcher of the two of us. [...] I don't recall one meaning standing out above the others that day. Rudolf is fond of saying that the more you think about what the B could mean, the more you learn about B-Trees, and that is good."*

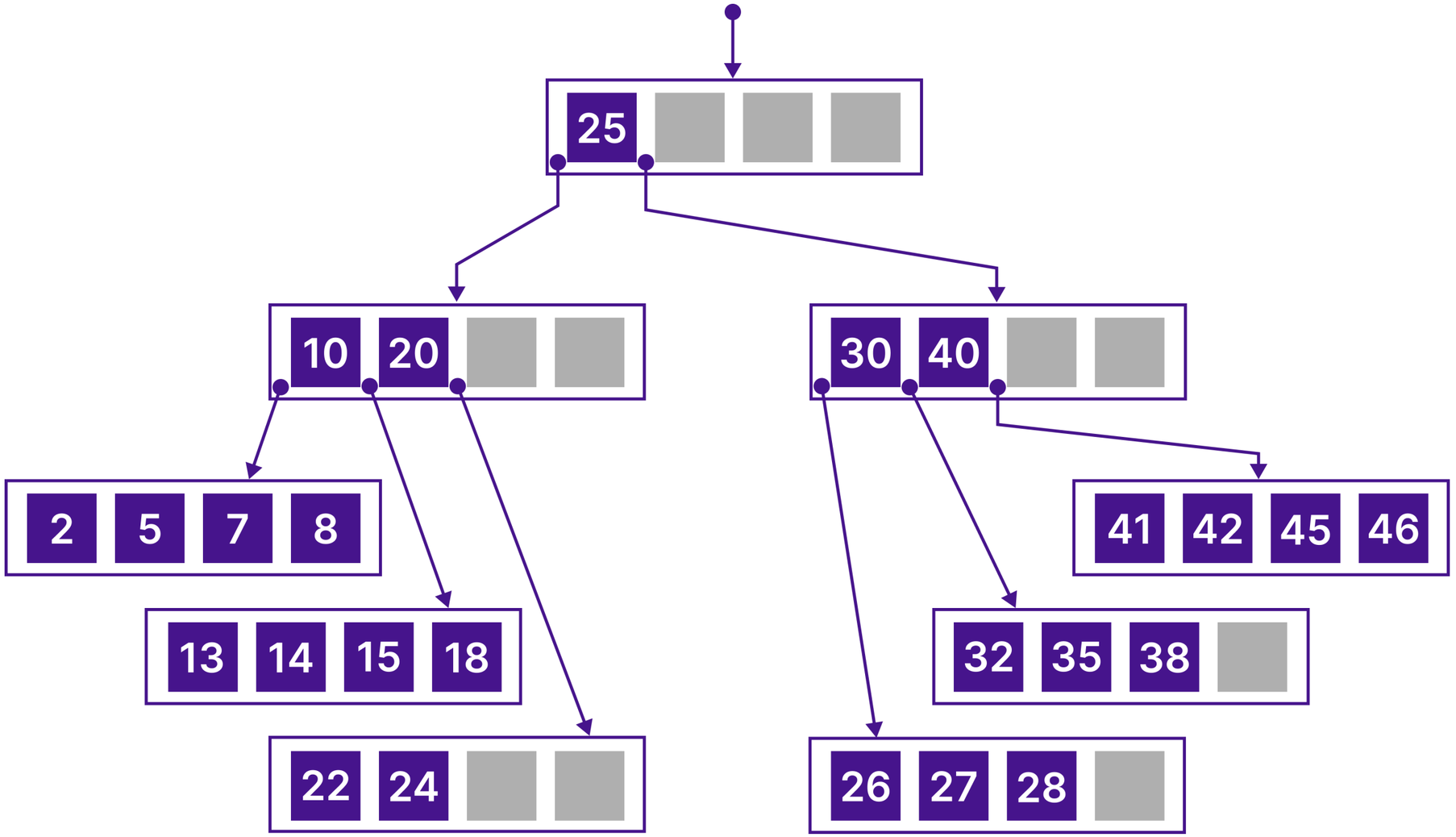
https://www.mccreight.com/people/ed_mcc/index.htm

Definition B-Baum

- Alle Pfade von Wurzel zu den Blättern haben die gleiche Länge h.
- Alle Knoten haben $[k, 2k]$ Elemente.
  - Die Wurzel darf auch weniger als k Elemente haben.
- Alle Knoten haben $[k+1, 2k+1]$ Nachfolger.
  - Ausgenommen davon sind Wurzel und Blätter.
- Elemente innerhalb eines Knotens sind sortiert.



Beispiel B-Baum mit $k = 2$



Unterlauf und Überfluss

$k = 2$

Unterlauf



Nur die Wurzel darf
im Unterlauf-Zustand sein.

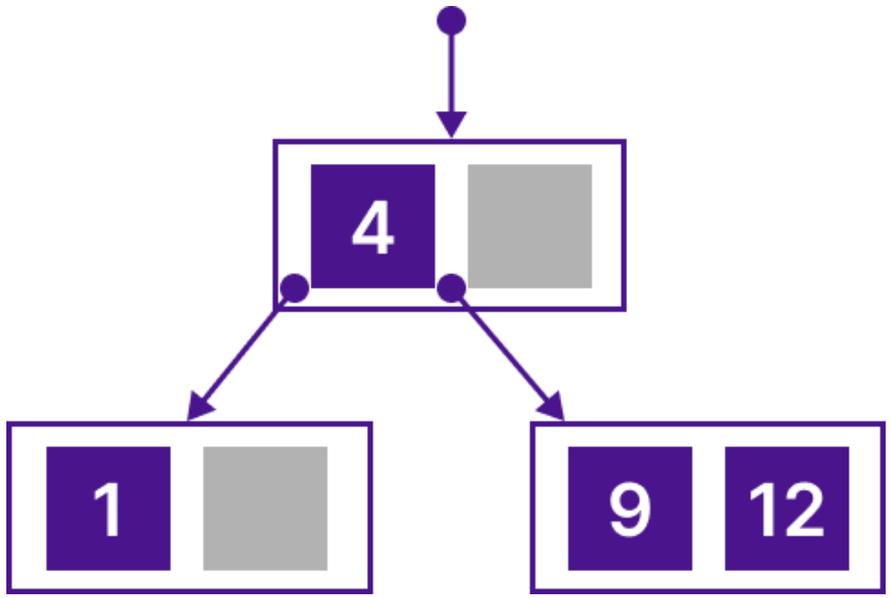
Überfluss



Unterlauf und Überfluss werden entsprechend durch
Split und **Merge** Operationen behandelt.

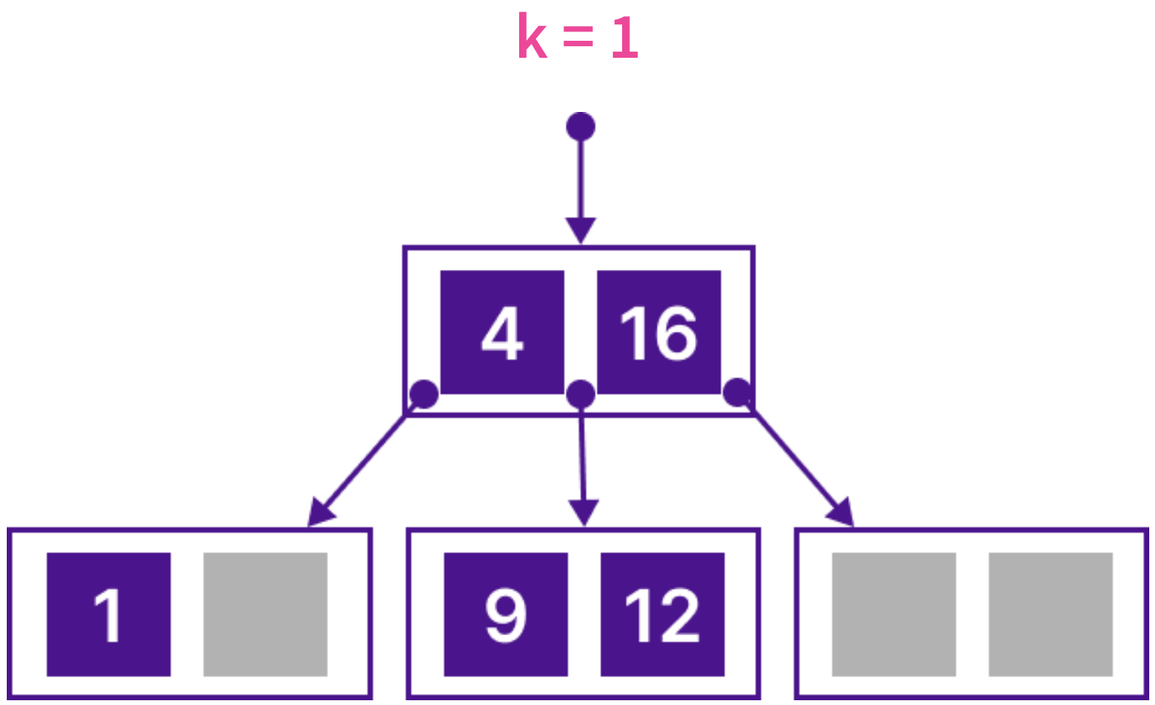
Ist dieser Baum gültig?

k = 1



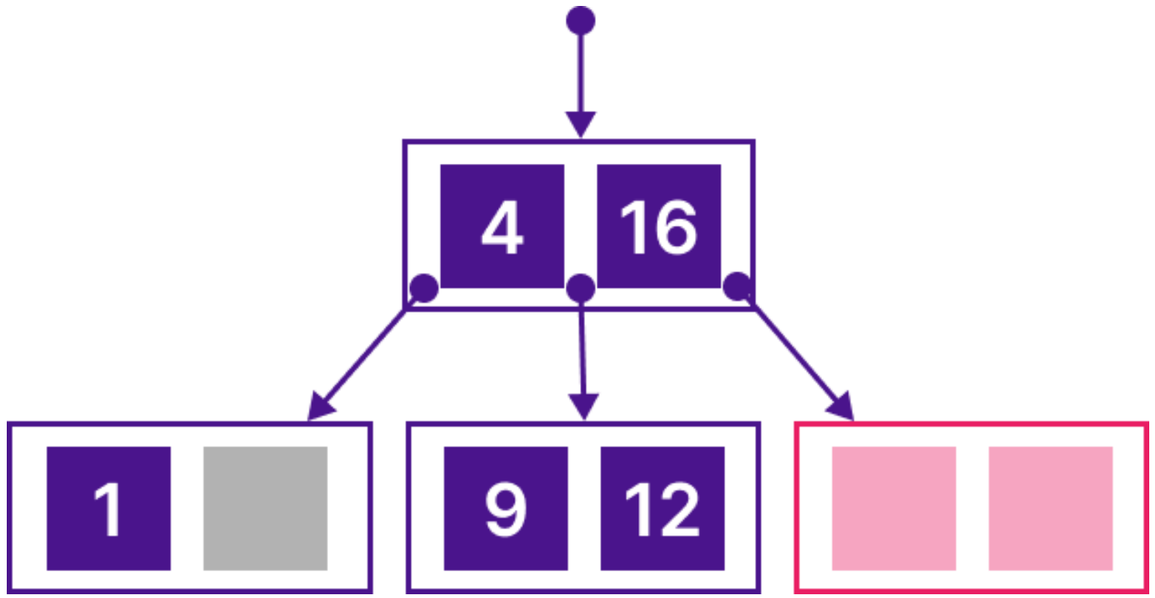
Ja, der Baum ist gültig.

Ist dieser Baum gültig?



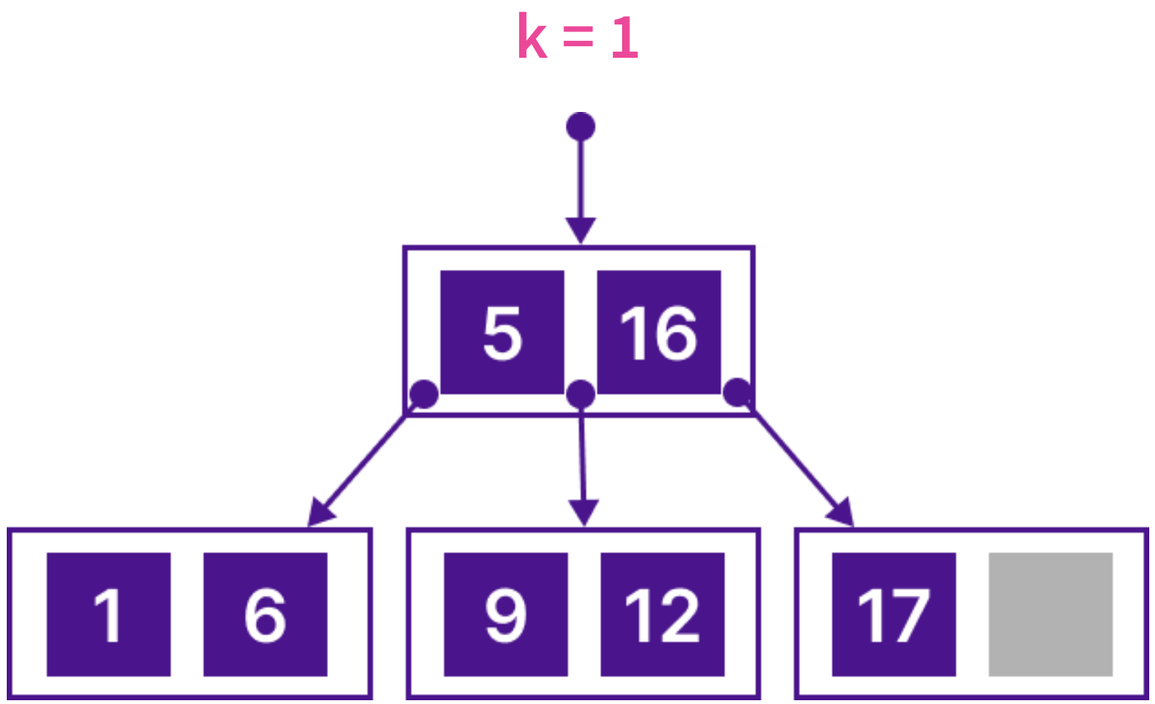
Nein

$k = 1$



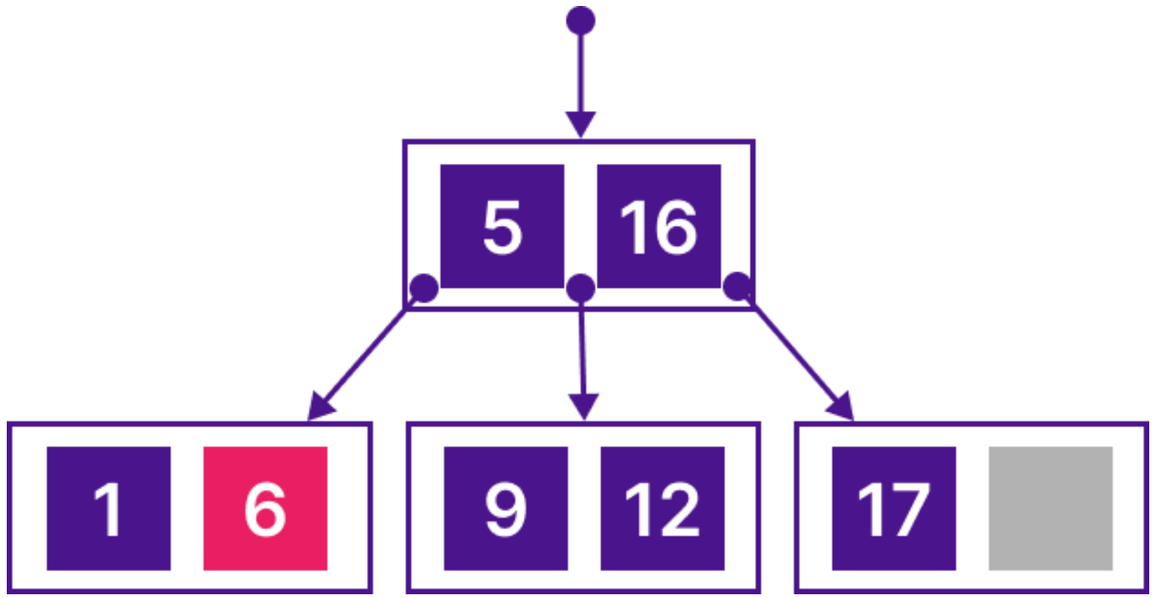
Das rechte Blatt hat zu wenig Elemente $\rightarrow$ Unterlauf.

Ist dieser Baum gültig?



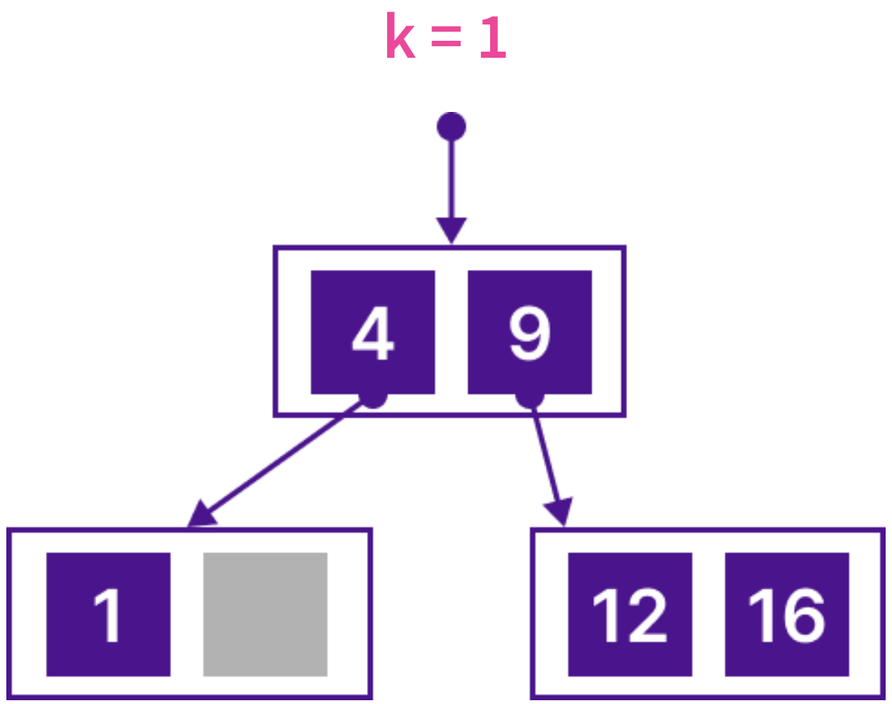
Nein

$k = 1$

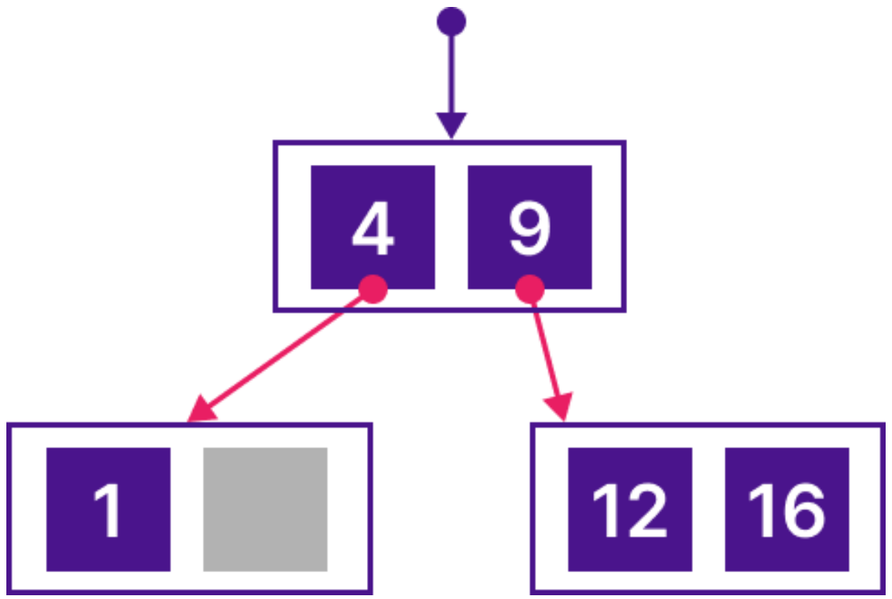


Das Element 6 verletzt die Ordnung im Baum.

Ist dieser Baum gültig?



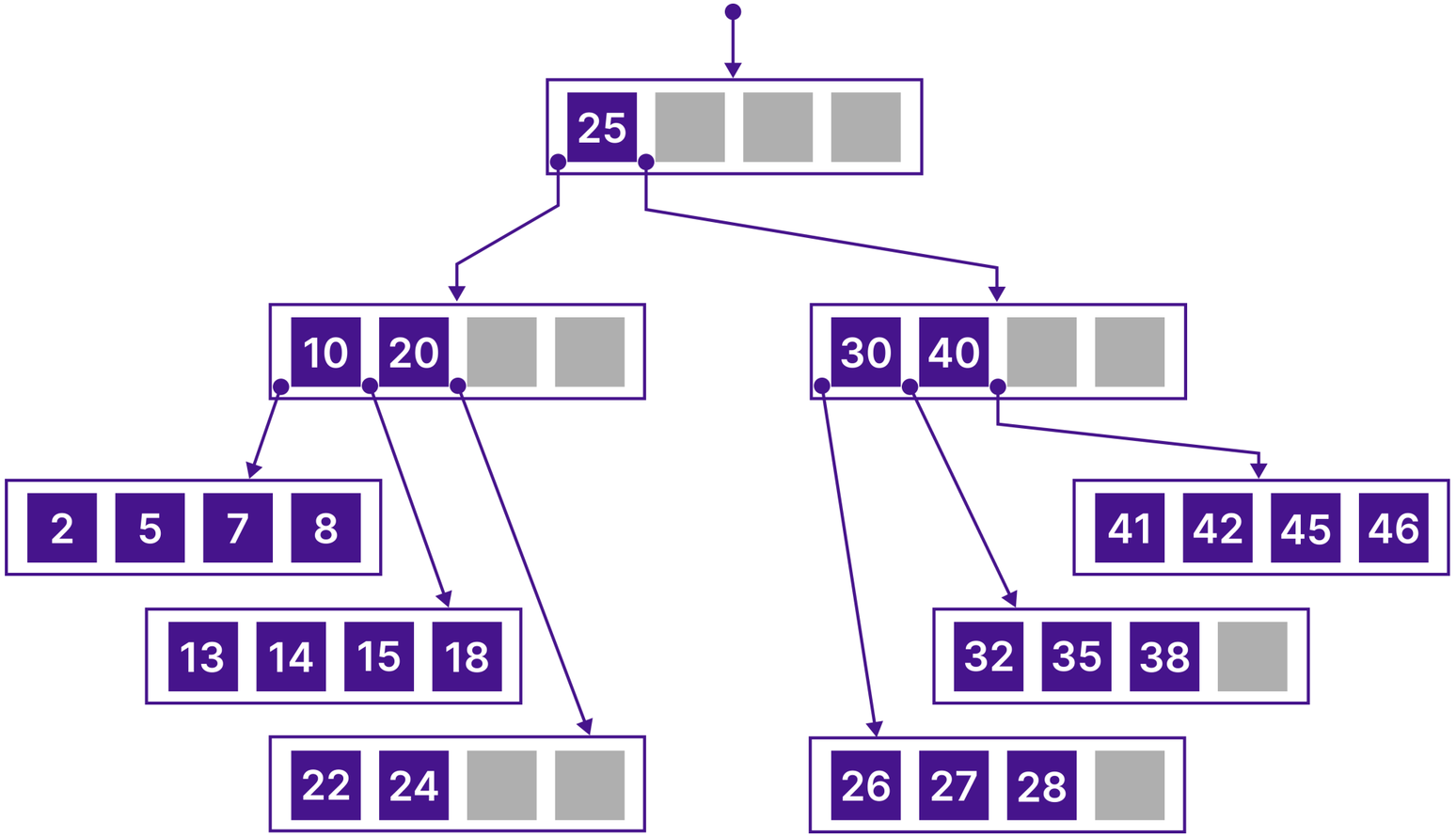
Nein



Die Wurzel hat zu wenig Zeiger, sie müsste auf drei Teilbäume zeigen.

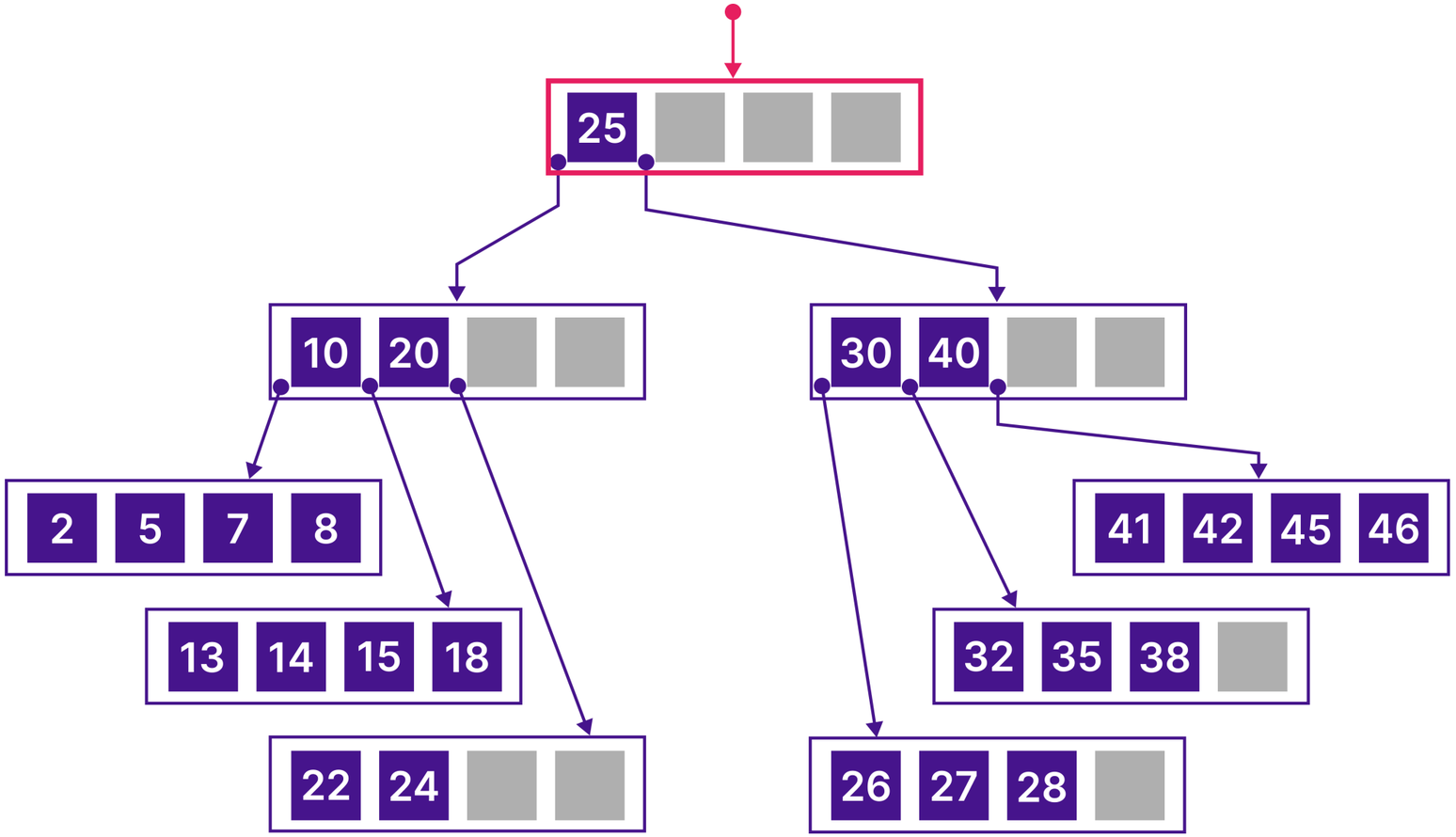
Suchen im B-Baum

Gegeben der folgende B-Baum mit k = 2, wir wollen das Element 35 finden.



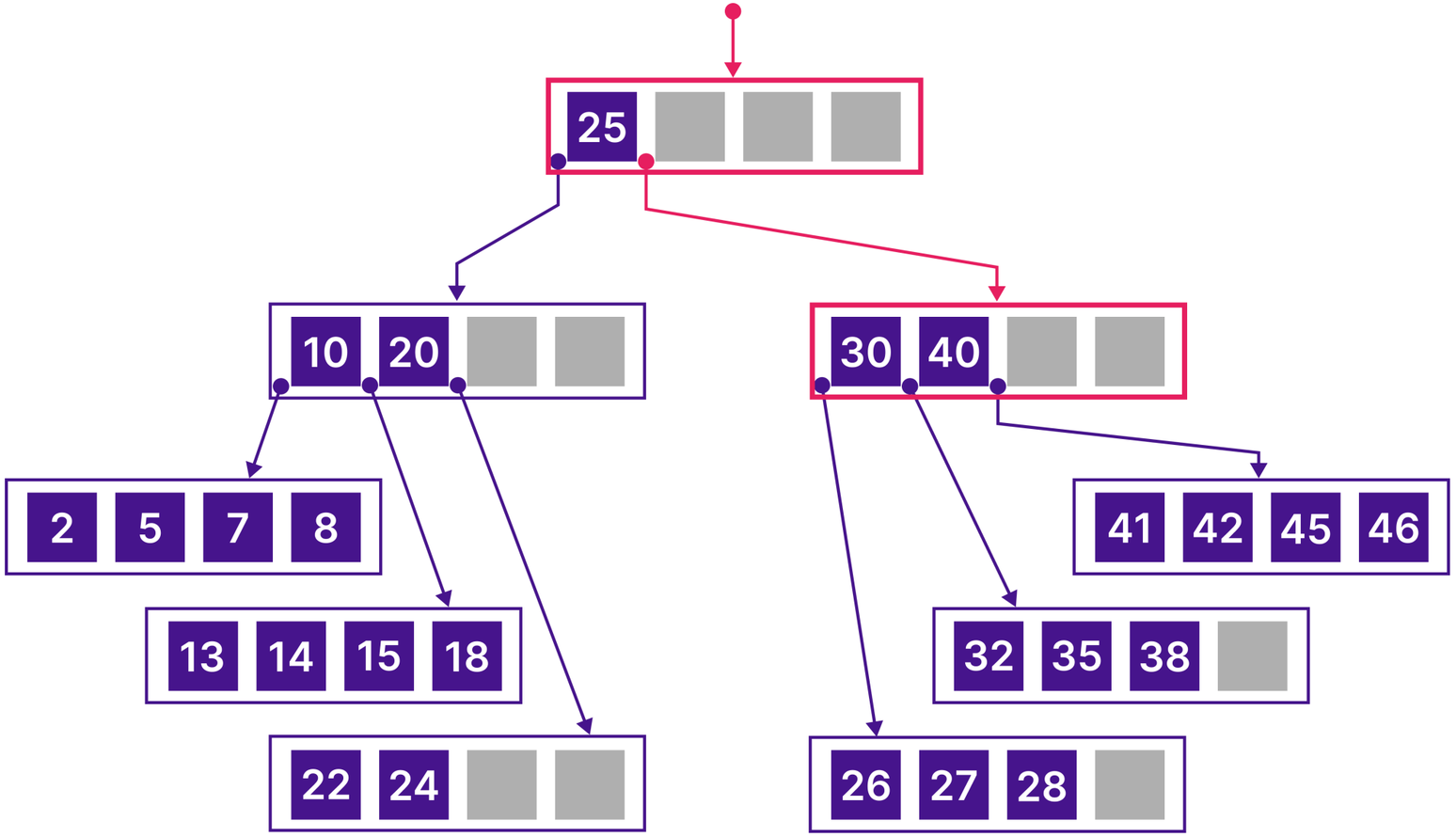
Suchen im B-Baum

Start bei der Wurzel im Baum.



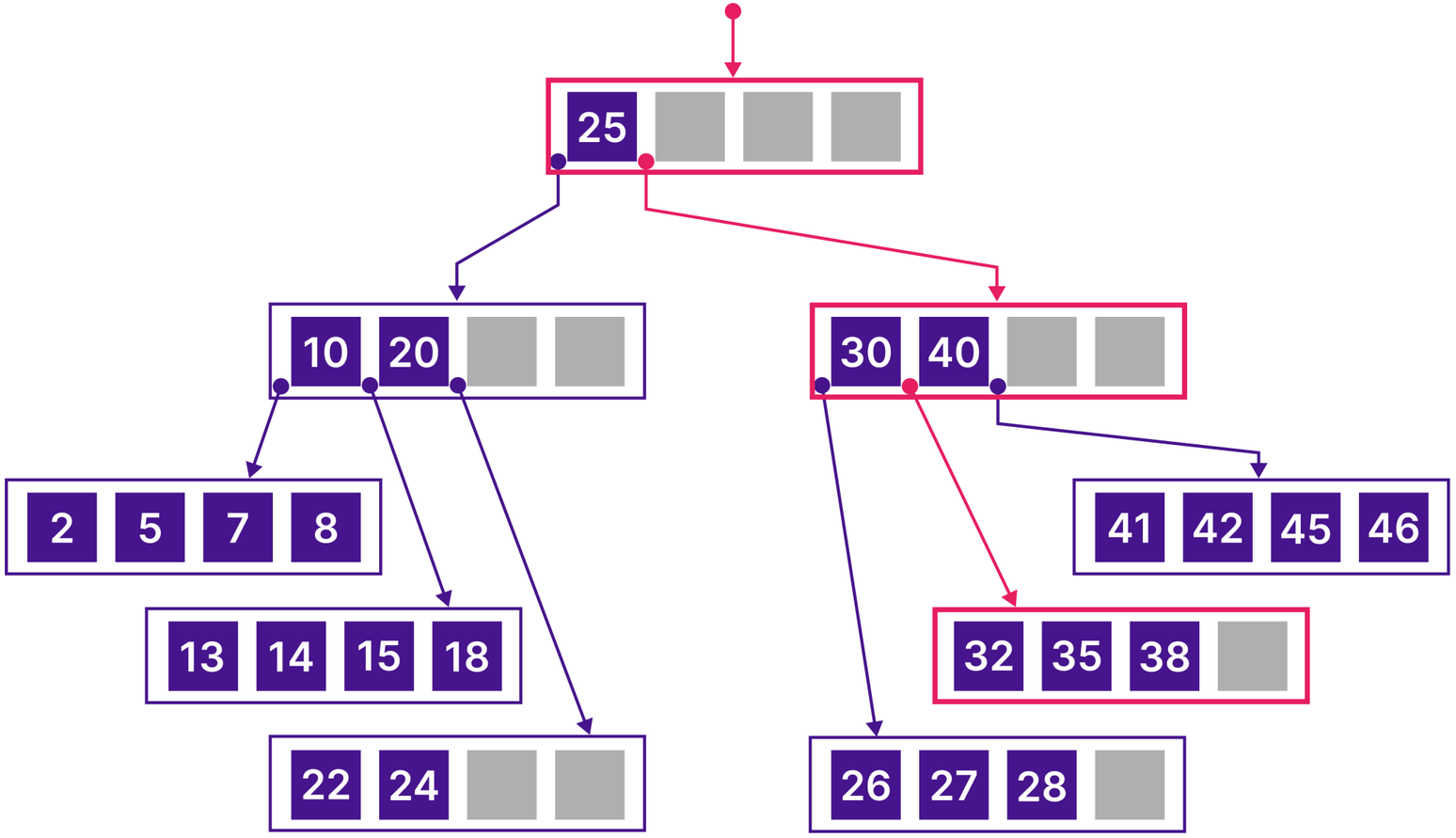
Suchen im B-Baum

Wir wählen den zweiten Zeiger, da $35 > 25$.



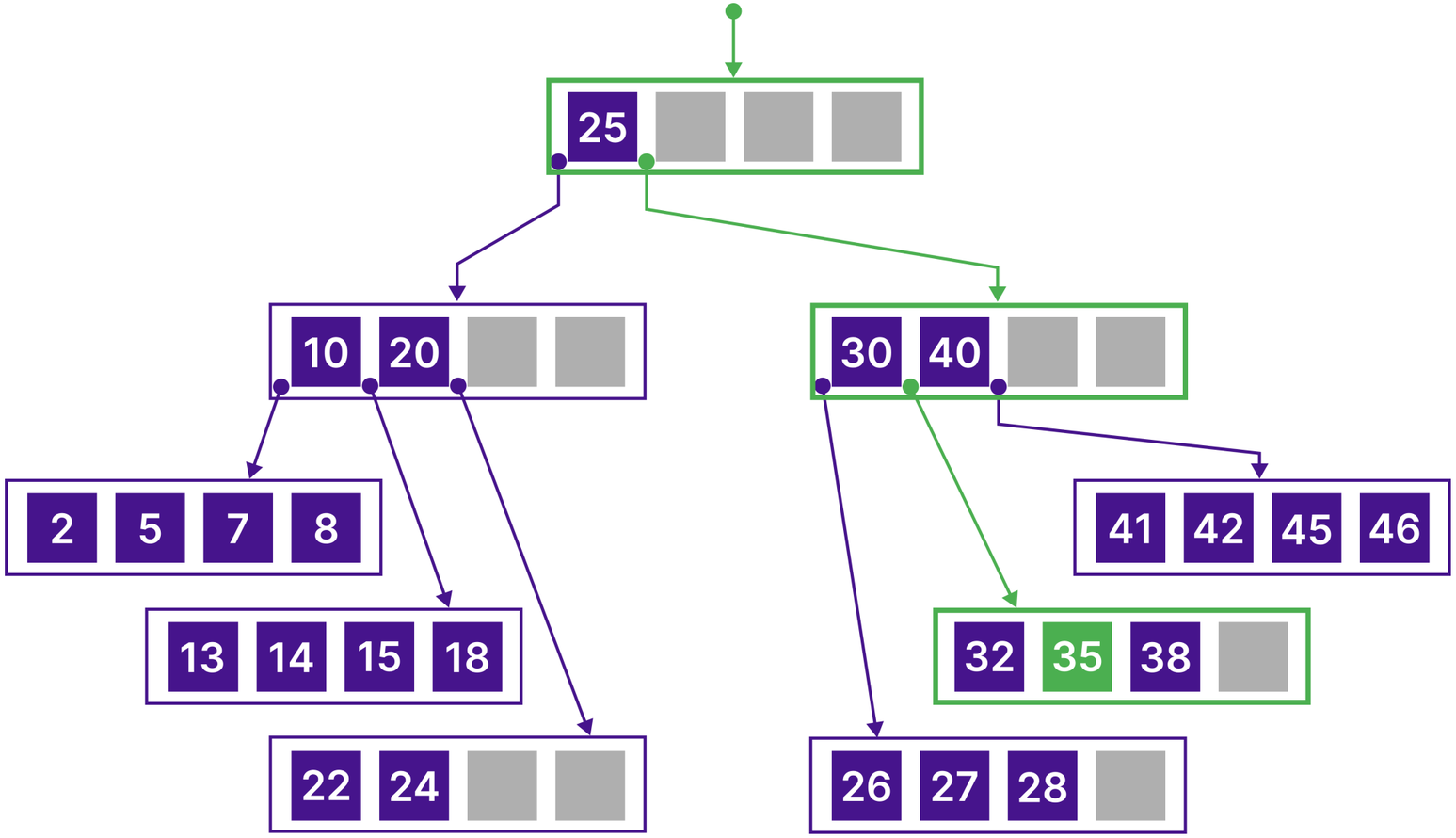
Suchen im B-Baum

Wir wählen wieder den zweiten Zeiger, da $30 < 35 \leq 40$.



Suchen im B-Baum

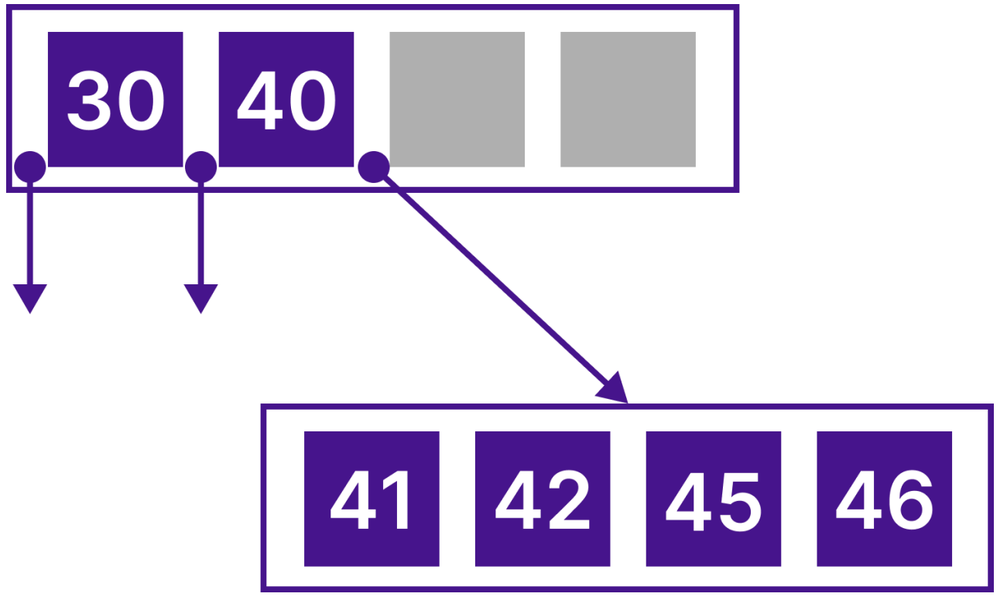
Im Blatt finden wir 35 durch eine sequentielle Suche durch den Knoten.



Tutorial: Einfügen in B-Bäume

$k = 2$

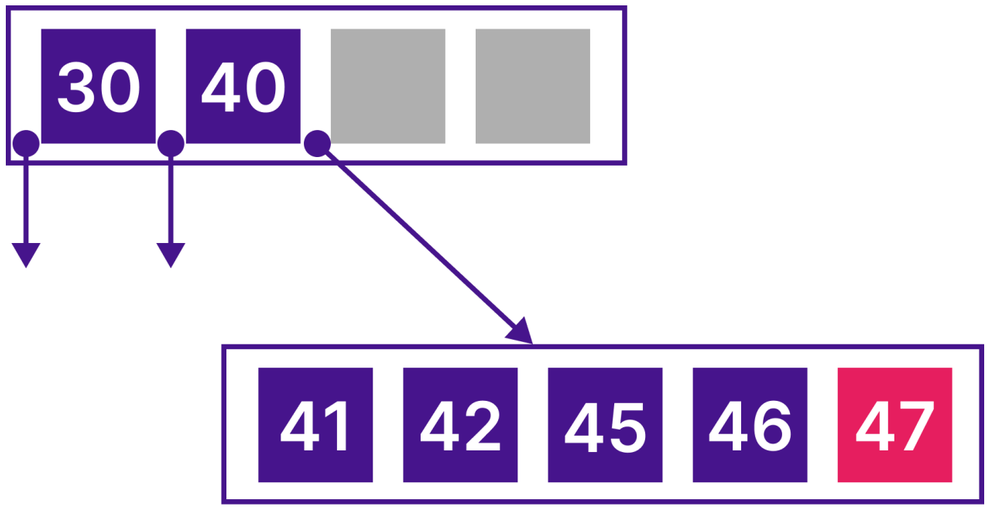
Gegeben der folgende Ausschnitt aus einem B-Baum



Tutorial: Einfügen in B-Bäume

$k = 2$

Wir fügen nun 47 ein



$2k + 1$

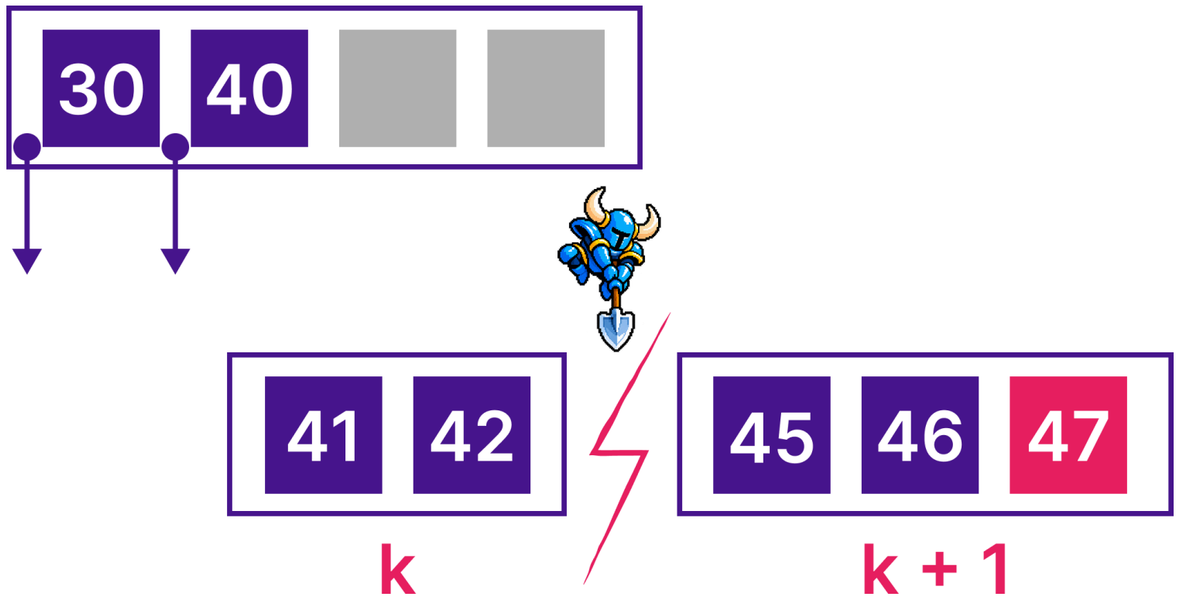
Wichtig: Neue Elemente werden **immer** in den Blättern eingefügt.

Wir stellen nach dem Einfügen einen Überfluss fest. **Der Knoten wird nun geteilt.**

Tutorial: Einfügen in B-Bäume

k = 2

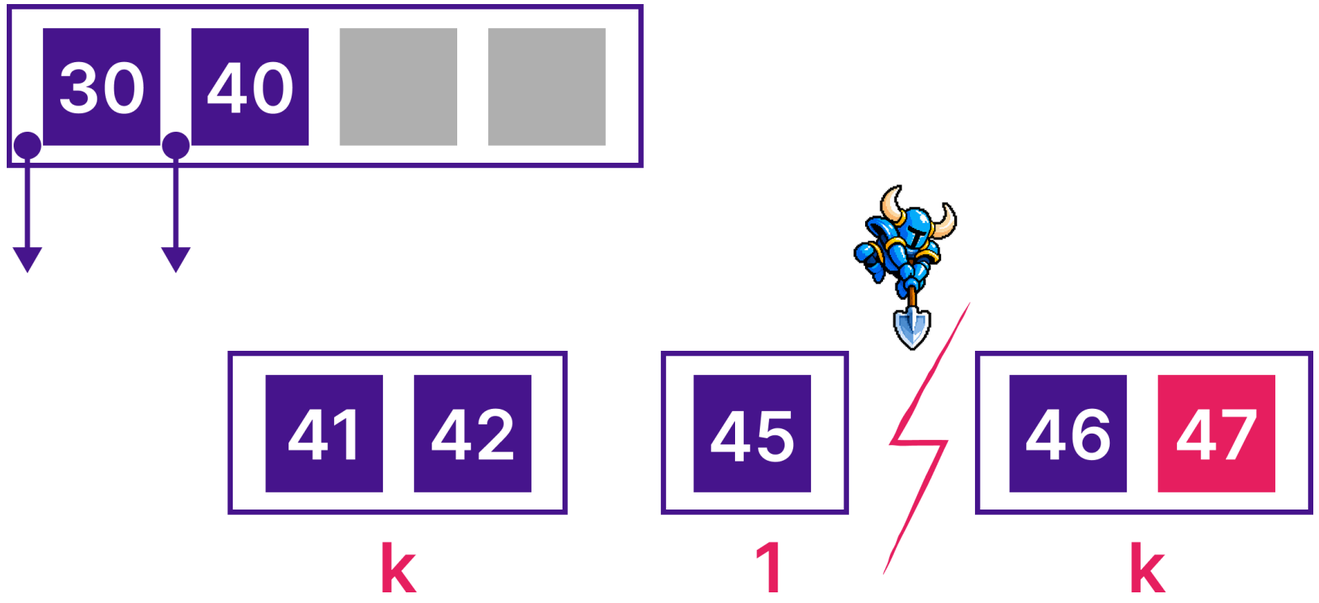
Die erste Teilung teilt die Liste im Knoten in k und $k + 1$ Elemente.



Tutorial: Einfügen in B-Bäume

$k = 2$

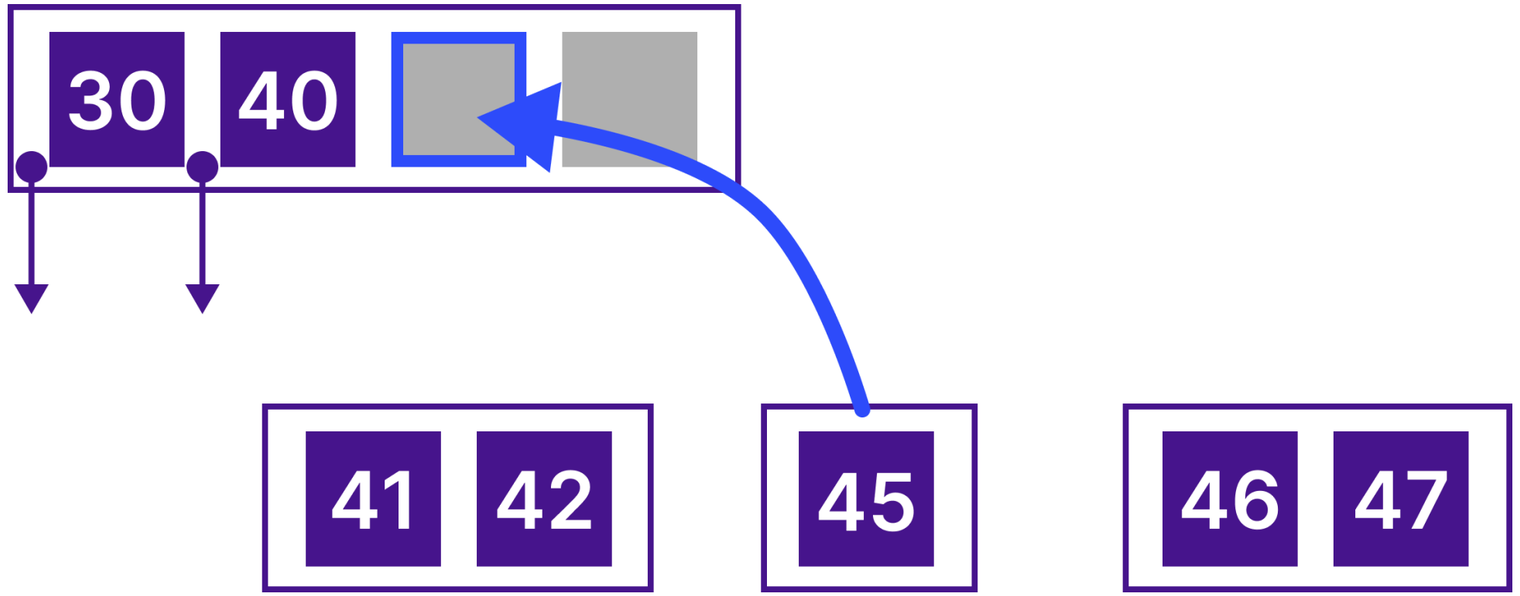
Die zweite Teilung isoliert nun den Median.



Tutorial: Einfügen in B-Bäume

$k = 2$

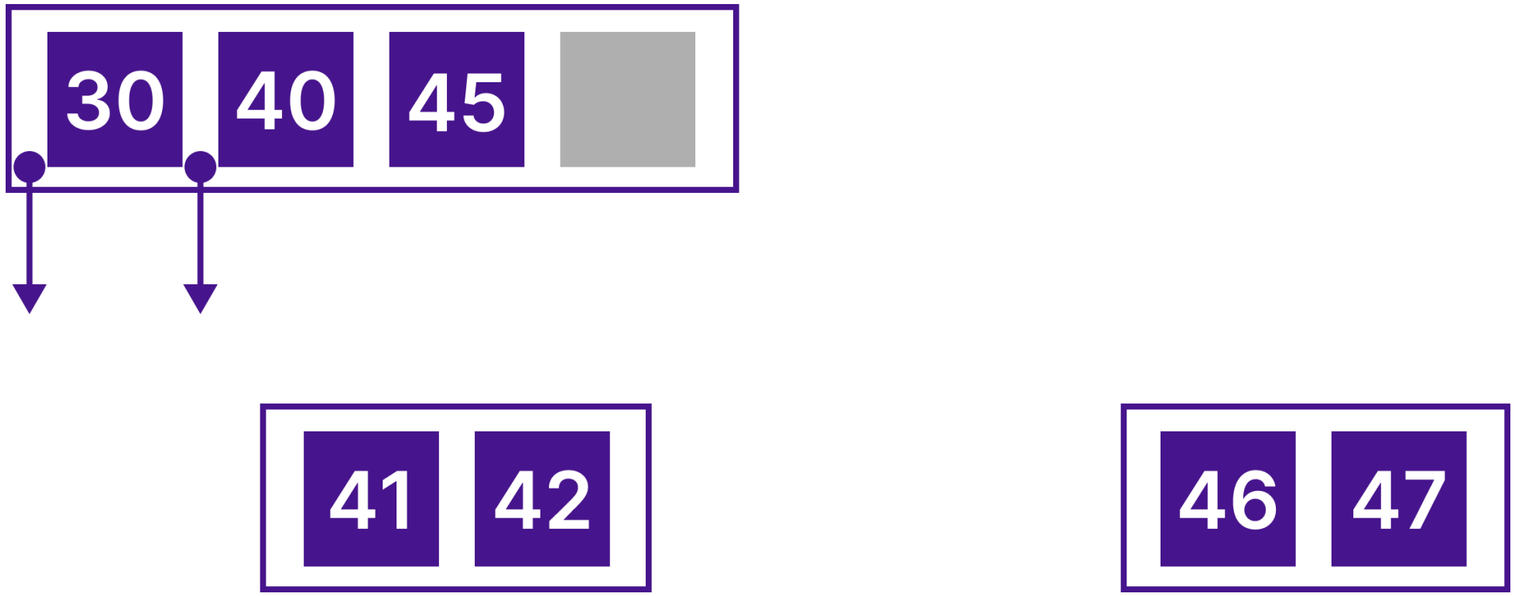
Der Median wird anschließend in den Elternknoten eingesetzt.



Tutorial: Einfügen in B-Bäume

k = 2

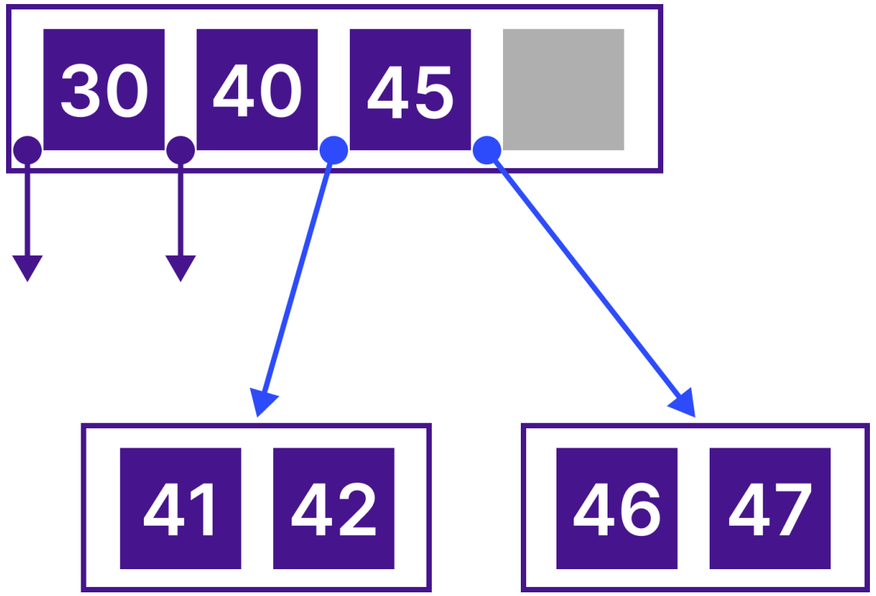
Nun müssen nur noch die Kinder neu verlinkt werden.



Tutorial: Einfügen in B-Bäume

k = 2

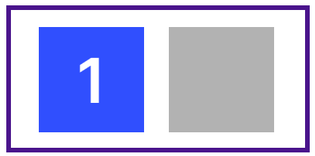
Der Baum hat sich selbst ausbalanciert.



Beispiel: Einfügen in B-Bäume

$k = 1$

Wir fügen 1 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

Wir fügen 5 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

Wir fügen 2 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

Der Knoten hat mehr als 2k Elemente.

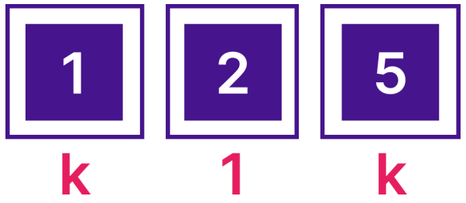


Überfluss

Beispiel: Einfügen in B-Bäume

$k = 1$

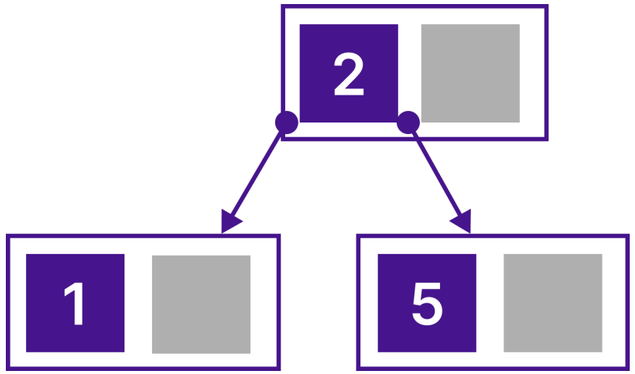
Der Knoten wird geteilt.



Beispiel: Einfügen in B-Bäume

$k = 1$

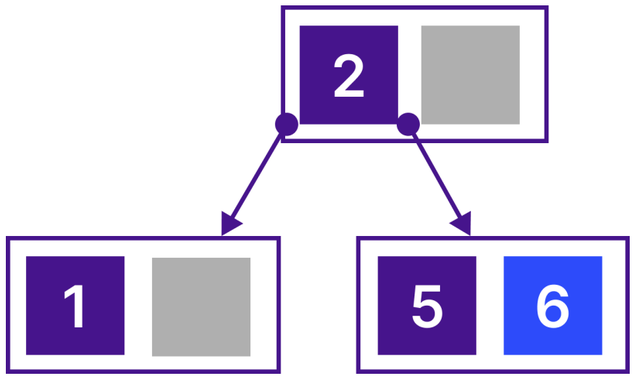
Anschließend wird eine neue Wurzel erschaffen.



Beispiel: Einfügen in B-Bäume

$k = 1$

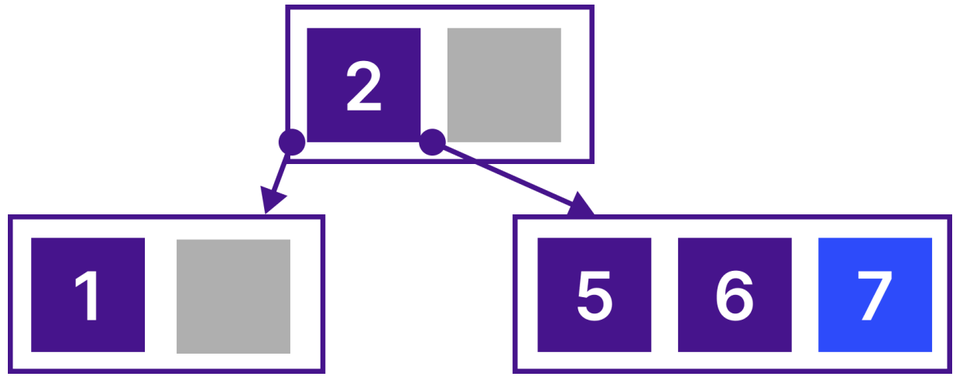
Wir fügen 6 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

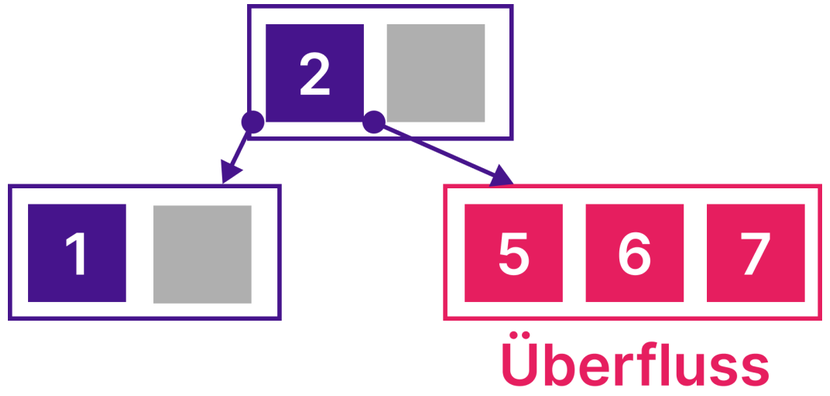
Wir fügen 7 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

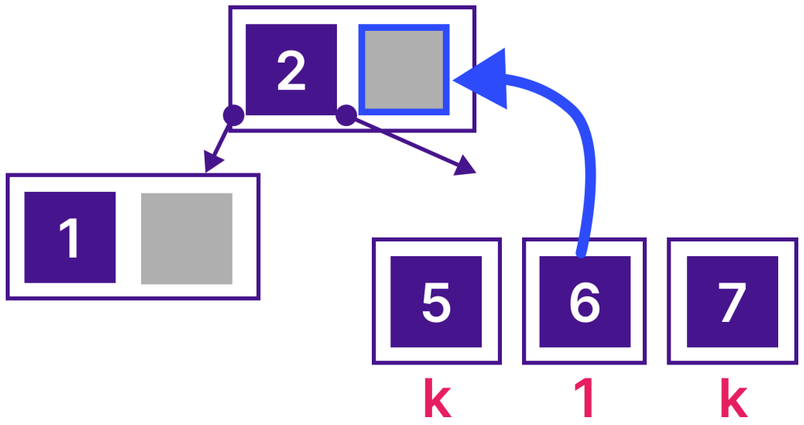
Der Knoten hat mehr als 2k Elemente.



Beispiel: Einfügen in B-Bäume

$k = 1$

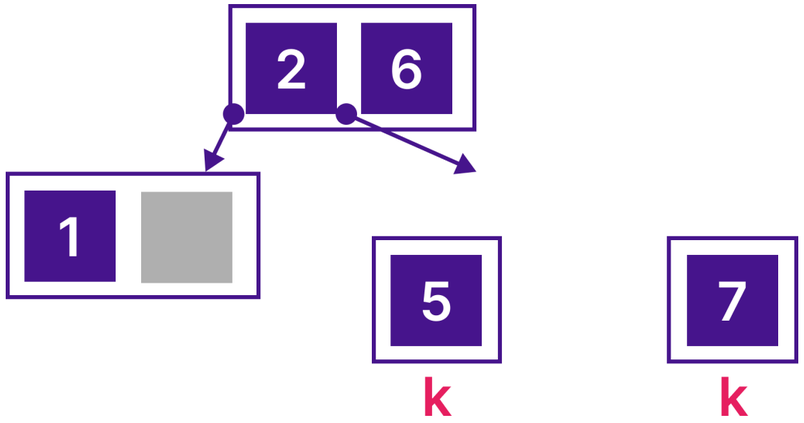
Deshalb teilen wir den Knoten und bewegen 6 in den Elternknoten.



Beispiel: Einfügen in B-Bäume

$k = 1$

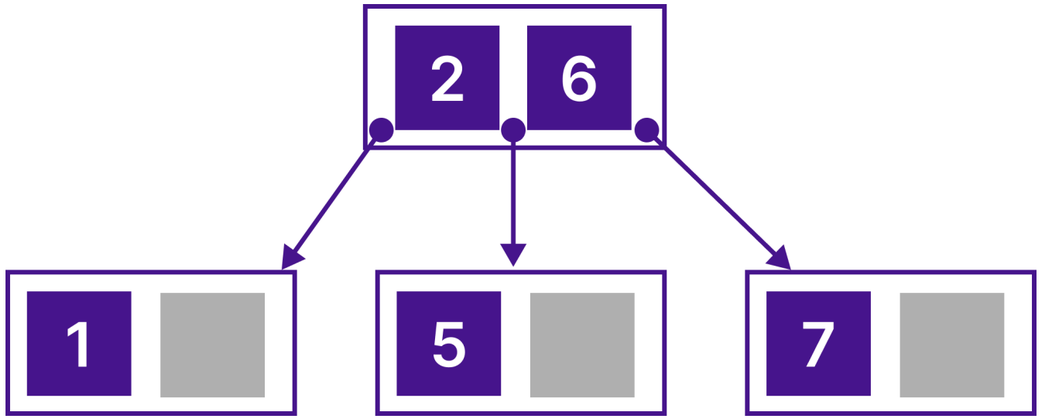
Die Kinder müssen neu verlinkt werden.



Beispiel: Einfügen in B-Bäume

$k = 1$

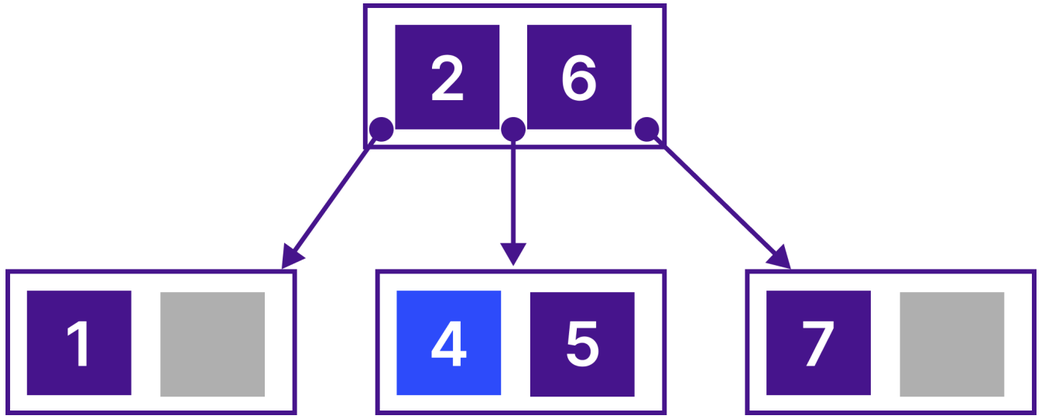
Done



Beispiel: Einfügen in B-Bäume

$k = 1$

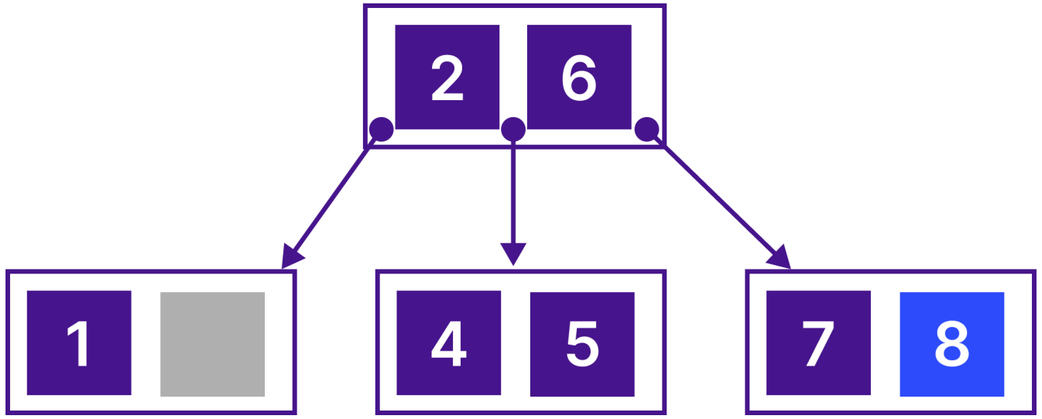
Wir fügen 4 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

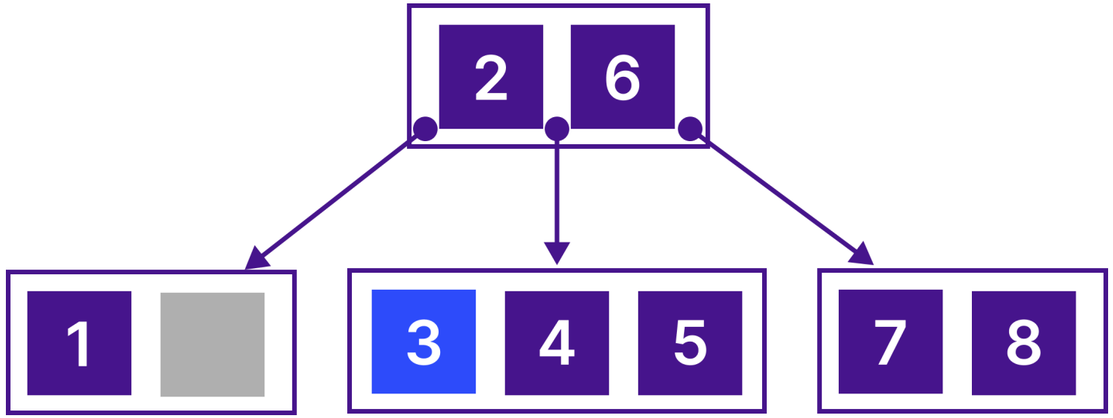
Wir fügen 8 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

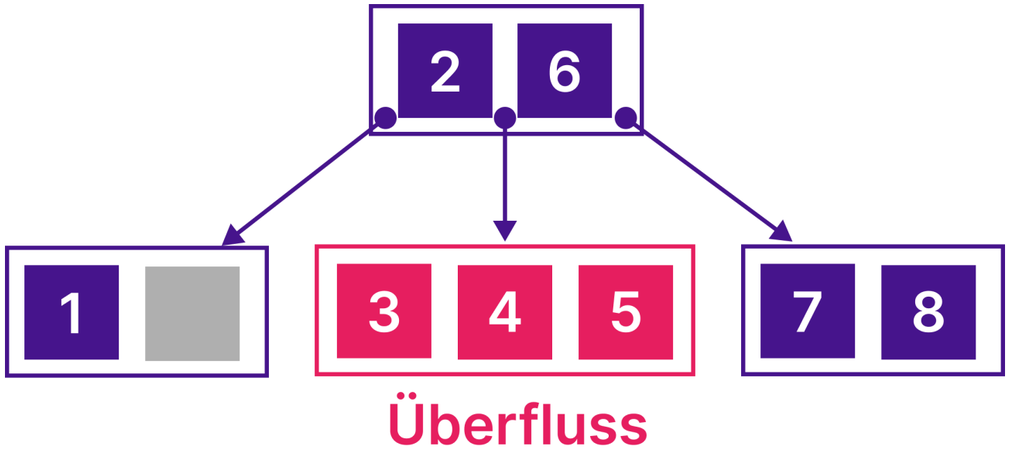
Wir fügen 3 ein.



Beispiel: Einfügen in B-Bäume

$k = 1$

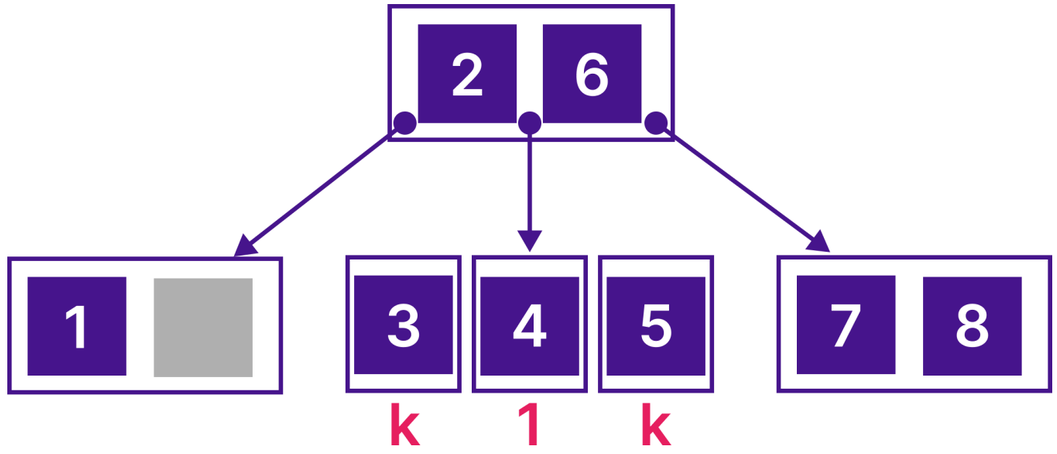
Der Knoten hat mehr als 2k Elemente.



Beispiel: Einfügen in B-Bäume

$k = 1$

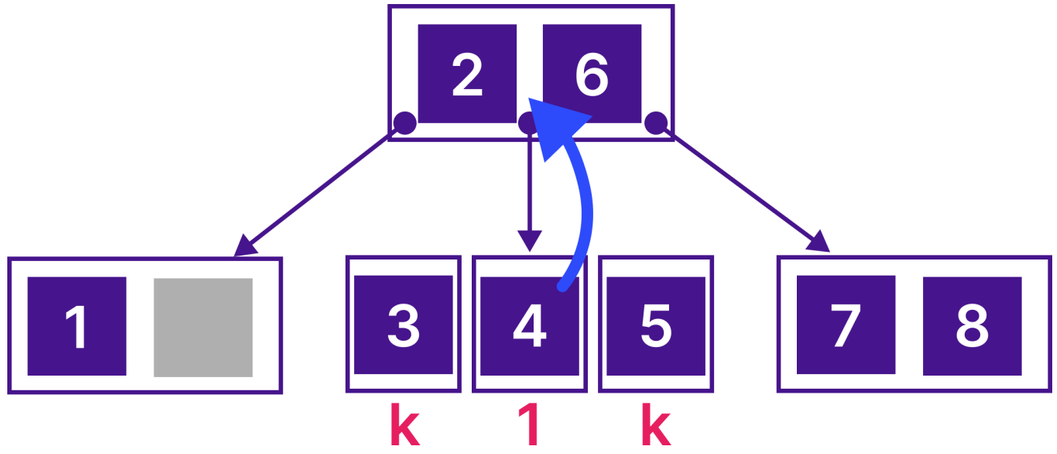
Der Knoten wird daher geteilt.



Beispiel: Einfügen in B-Bäume

$k = 1$

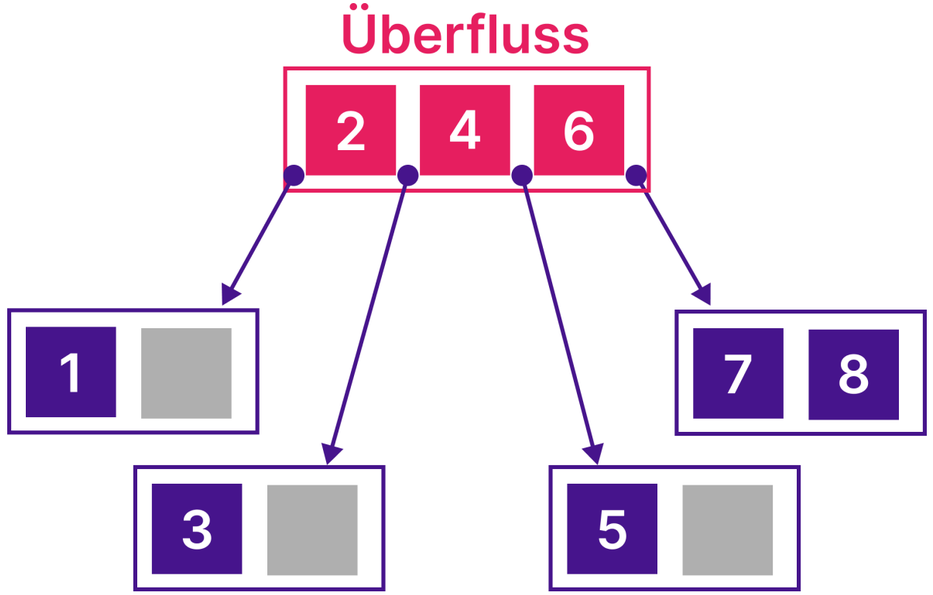
Und der Median wird wieder in den Elternknoten bewegt.



Beispiel: Einfügen in B-Bäume

$k = 1$

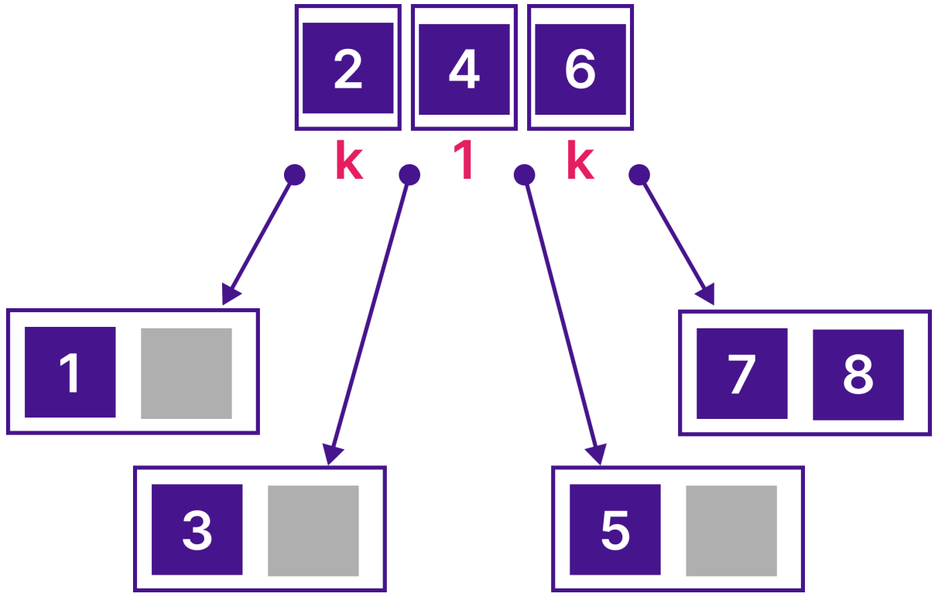
Das sorgt aber erneut für einen Überfluss.



Beispiel: Einfügen in B-Bäume

$k = 1$

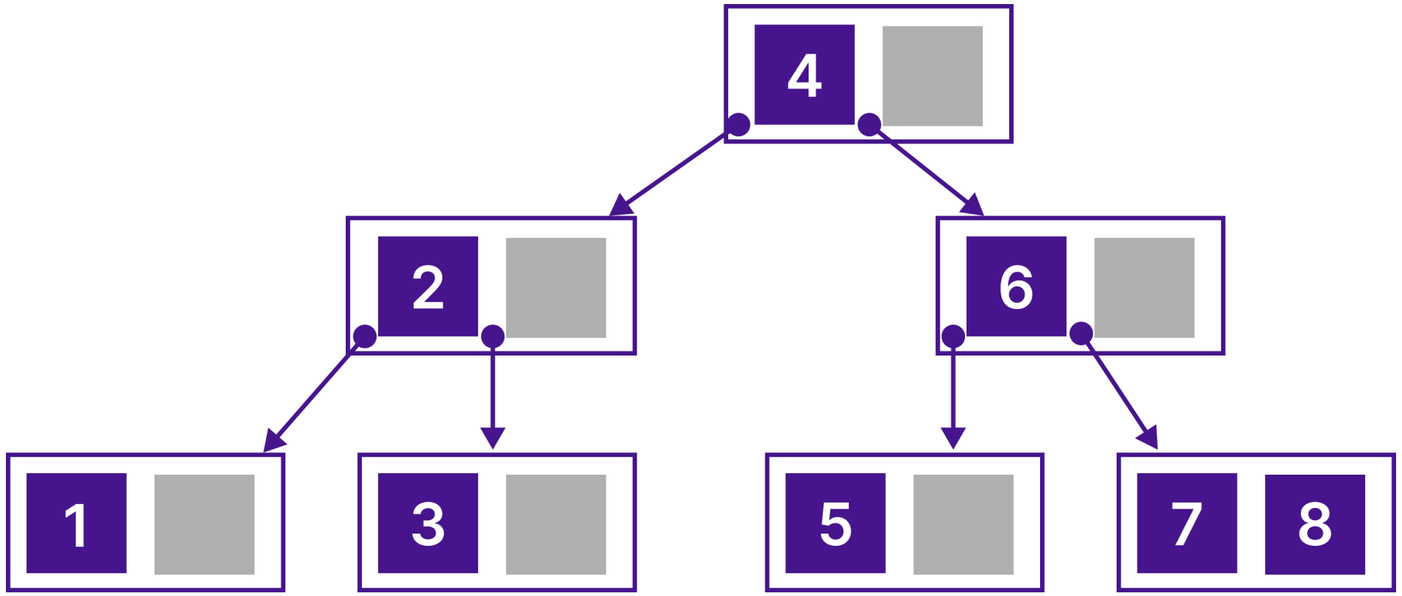
Der Elternknoten wird daher erneut geteilt.



Beispiel: Einfügen in B-Bäume

$k = 1$

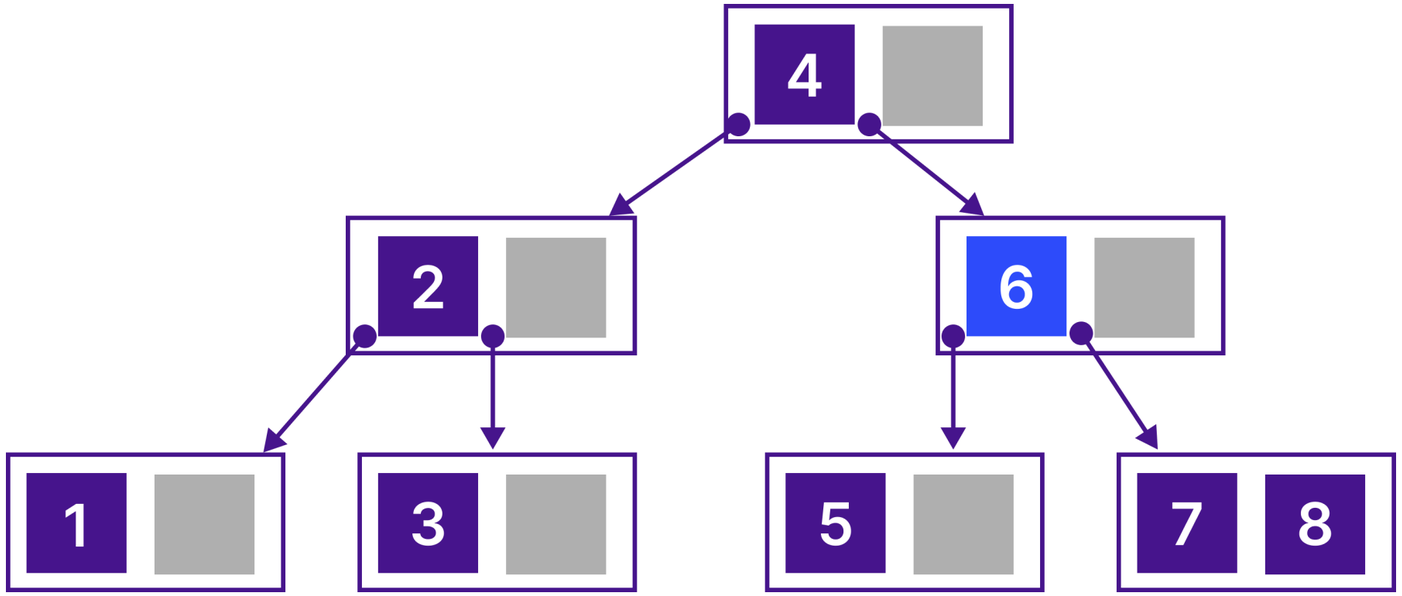
Und der Median erzeugt eine neue Wurzel.



Löschen in B-Bäumen

$k = 1$

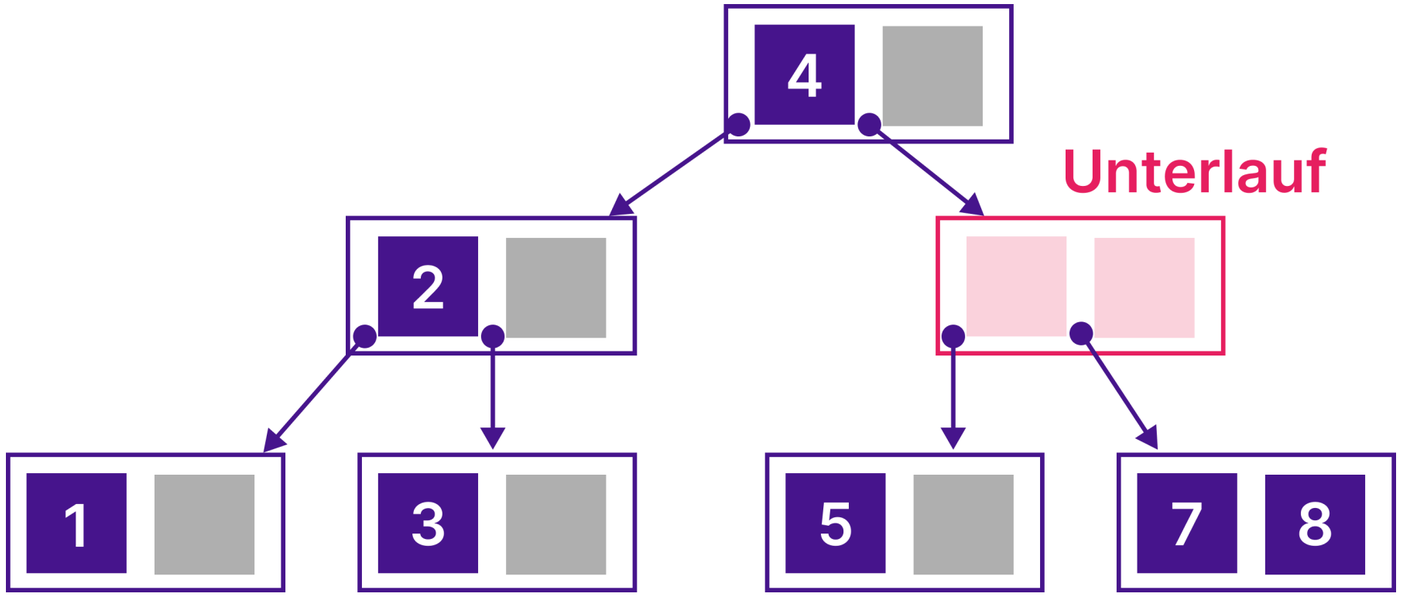
Wir löschen Element 6. (**Fall: Innerer Knoten**)



Löschen in B-Bäumen

$k = 1$

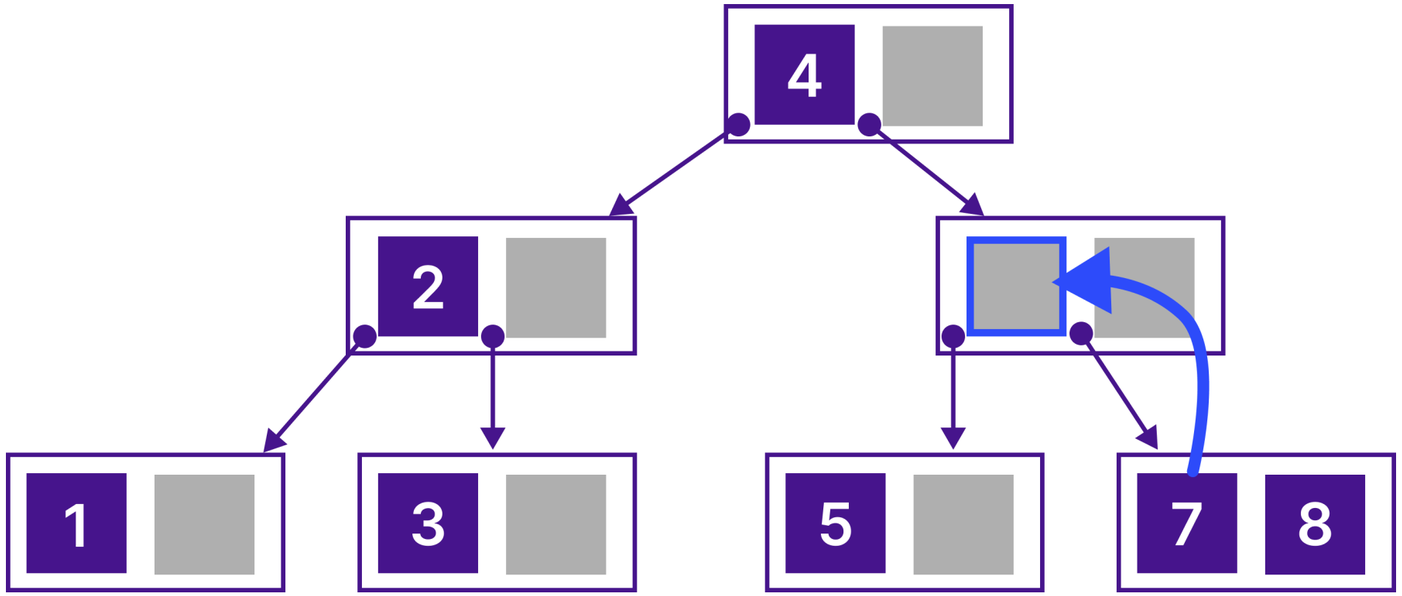
Das sorgt für einen Unterlauf in einem inneren Knoten.



Löschen in B-Bäumen

$k = 1$

Wir können nun ein Element aus dem Nachfolger in den Knoten einsetzen.

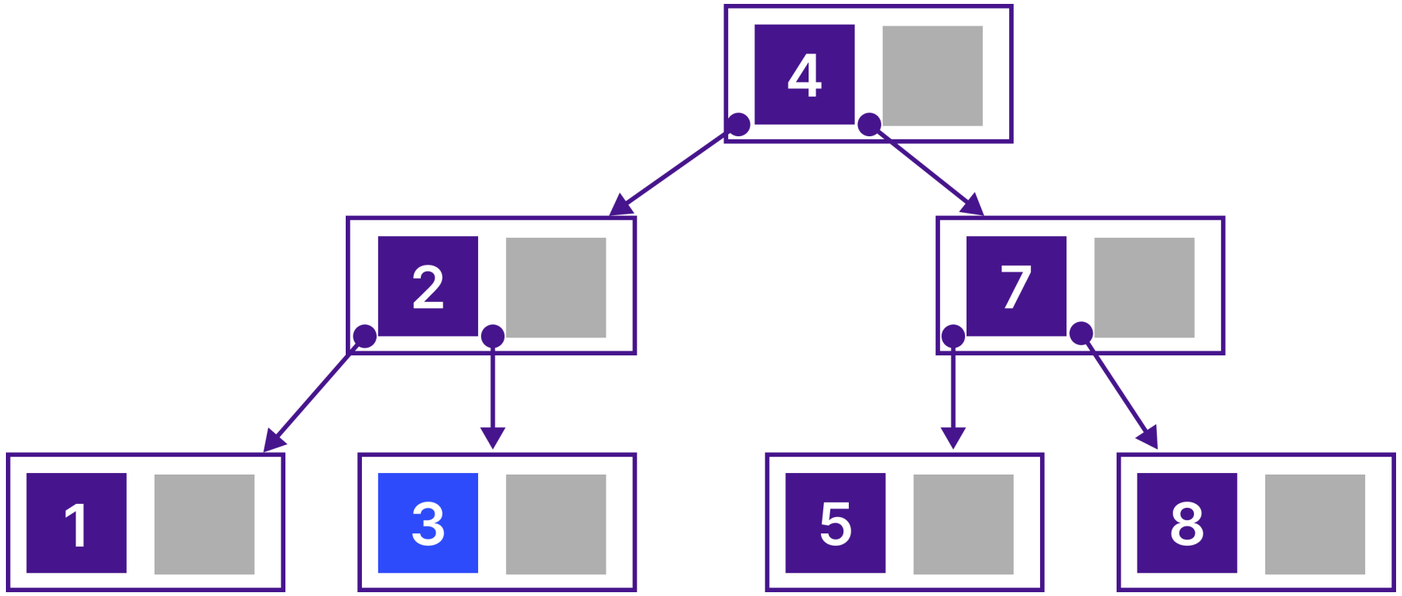


Man wählt dabei den volleren Nachfolger.

Löschen in B-Bäumen

$k = 1$

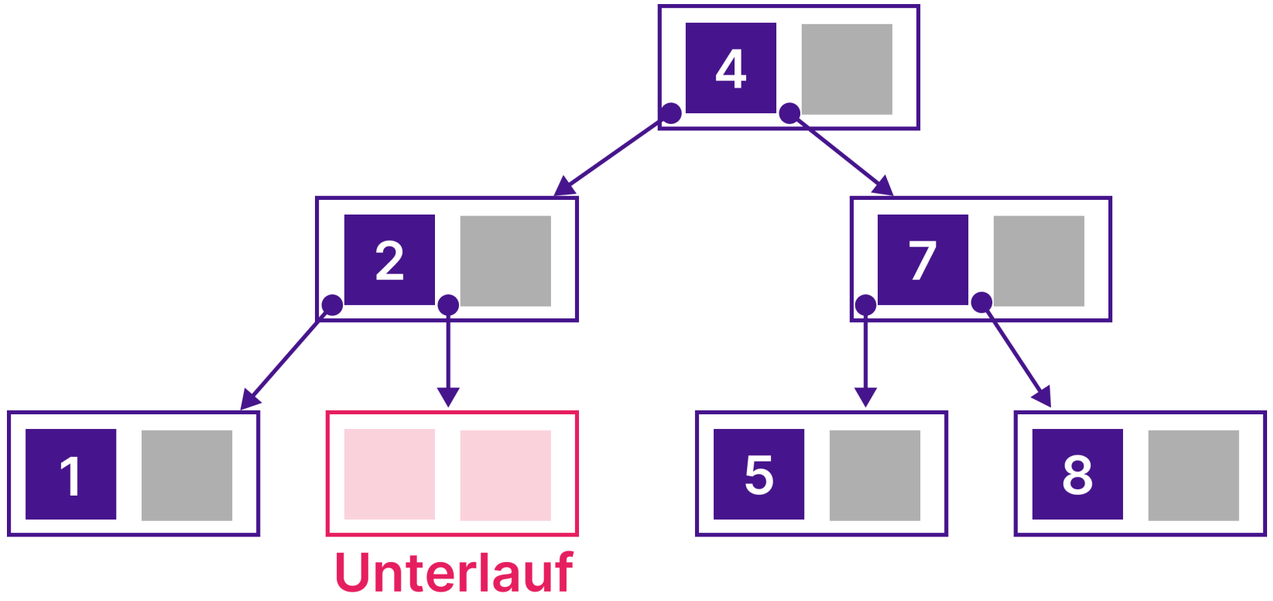
Wir löschen nun Element 3. (**Fall: Blattknoten**)



Löschen in B-Bäumen

$k = 1$

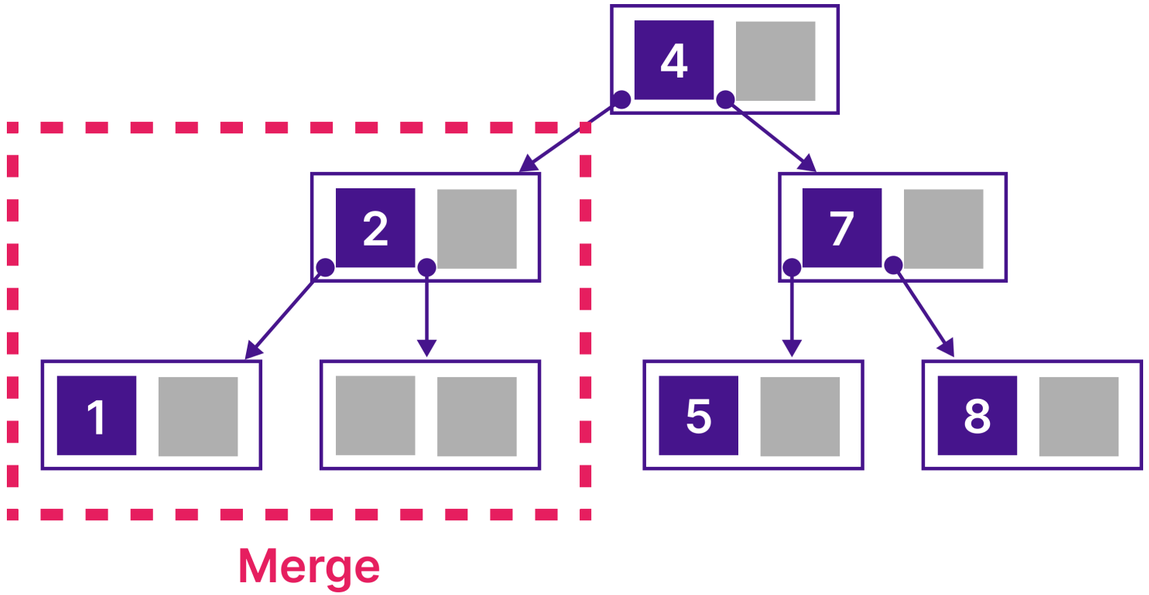
Dies sorgt erneut für einen Unterlauf.



Löschen in B-Bäumen

$k = 1$

Blätter haben keinen Nachfolger von dem wir Elemente stehlen können.



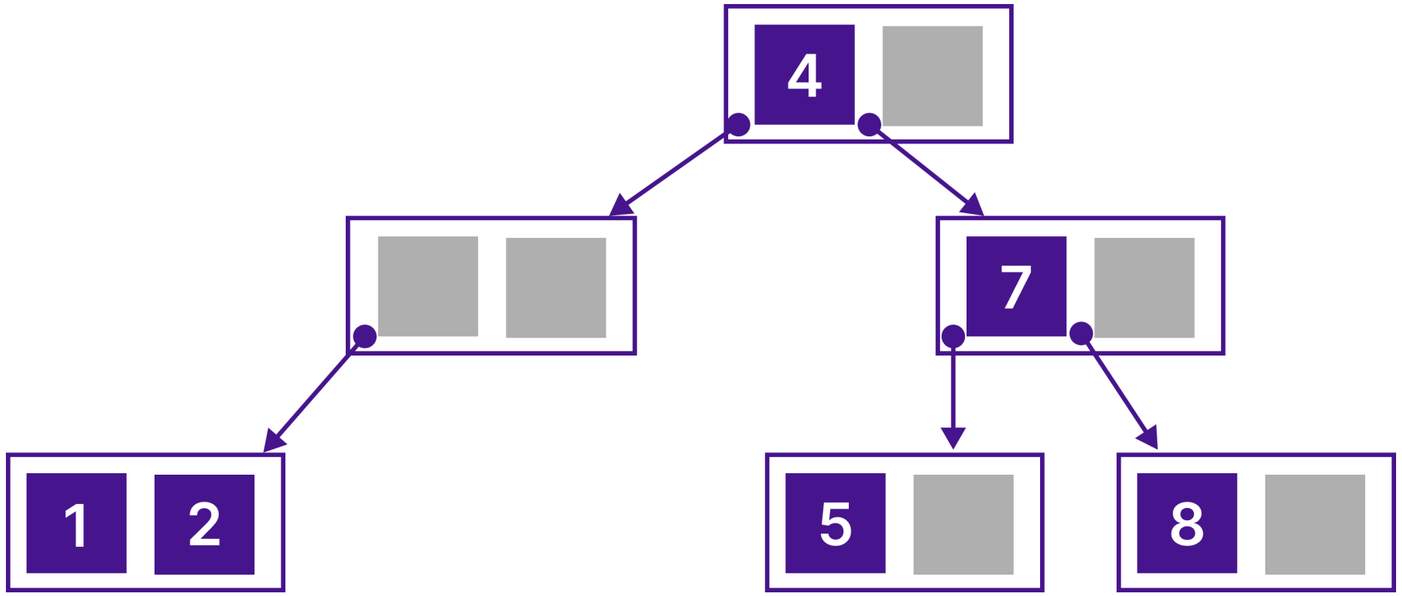
Deshalb fügen wir Schwesterknoten und den Median aus dem Elternknoten zusammen.

Das entspricht einer Umkehrung der Teilung aus der Einfüge-Operation.

Löschen in B-Bäumen

$k = 1$

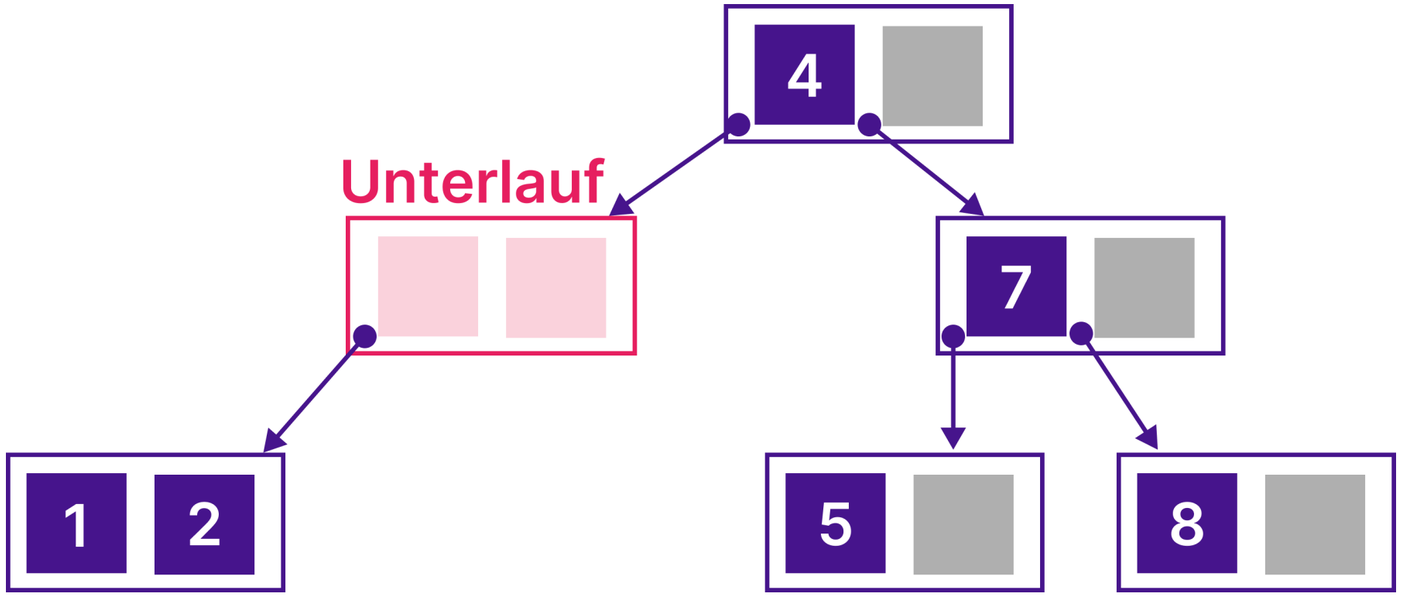
Den Elternknoten können wir nicht löschen, da die Höhe immer konstant bleiben muss.



Löschen in B-Bäumen

$k = 1$

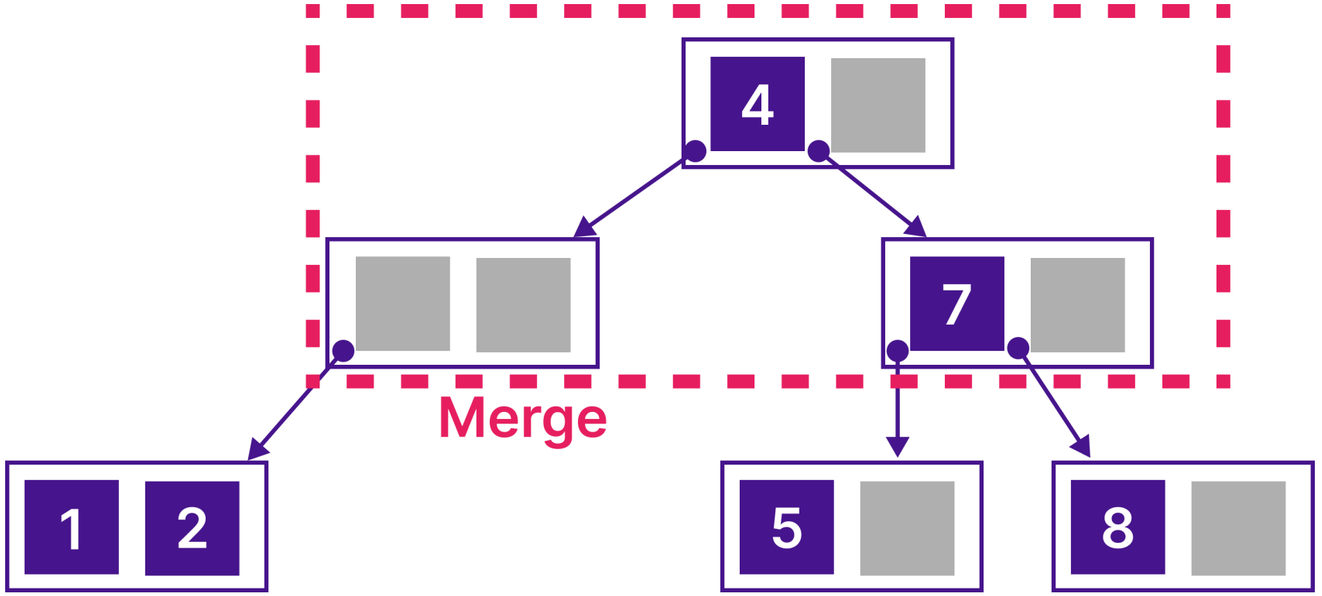
Mit dem Merge haben wir daher nun einen Unterlauf im Elternknoten verursacht.



Löschen in B-Bäumen

$k = 1$

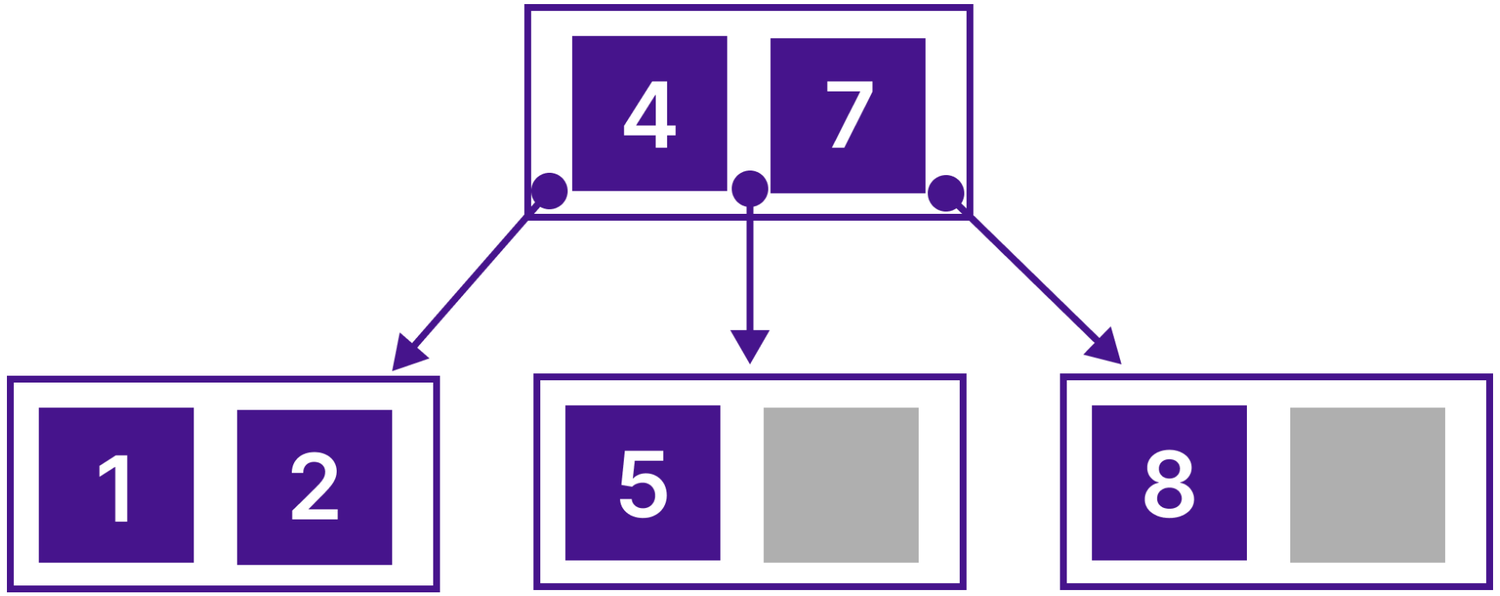
Wir führen also nun erneut eine Merge Operation aus.



Löschen in B-Bäumen

$k = 1$

Dies reduziert nun die Höhe des Baumes und sorgt für Balance.



Also wie funktioniert das Löschen nun?

Wenn Sie aus einem **inneren Knoten** löschen:

- Vom vollsten Nachfolger ein Element stehlen und in den gelöschten Knoten einfügen.
- Falls dies zu einem Unterlauf führt, machen Sie das gleiche beim Kindknoten nochmal.

Wenn Sie aus einem **Blattknoten** löschen:

- Wenn ein Unterlauf vorliegt, führen Sie eine Merge Operation mit Geschwisterknoten aus.
- Dann bei Elternknoten fortführen wenn ein Unterlauf nach oben propagiert.

Höhe eines B-Baums

(Nicht prüfungsrelevant, dies ist nur der Vollständigkeit halber, wir werden die Höhe nicht herleiten)

$$\lceil \log_{2k+1}(n+1) \rceil \leq h \leq \left\lceil \log_{k+1} \left(\frac{n+1}{2} \right) \right\rceil + 1$$

Die Höhe wächst logarithmisch und hängt neben n auch maßgeblich von k ab.

Bayer, R., McCreight, E.M. Organization and maintenance of large ordered indexes. Acta Informatica 1, 173–189 (1972). doi:10.1007/BF00288683

Nicht Prüfungsrelevant

So much to see, so much to do!

Ein Ausblick auf weiterführende Themen

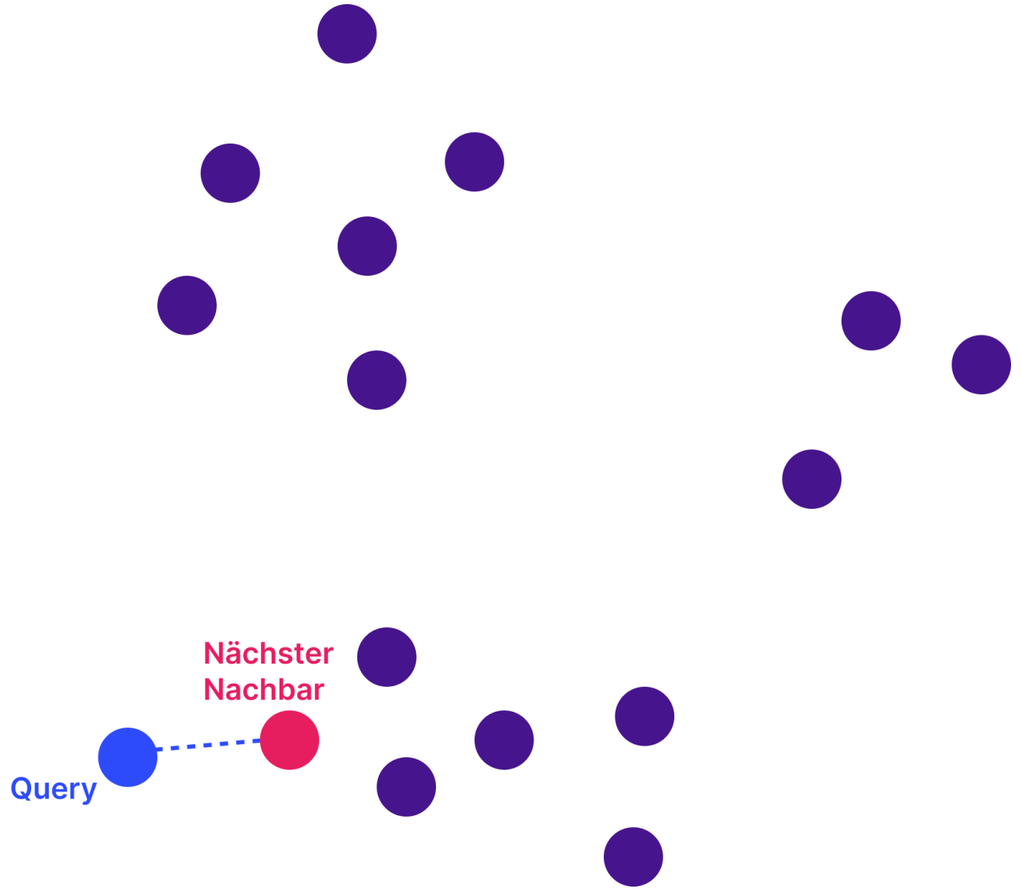
Wir haben eindimensionale Zugriffsmethoden besprochen

71	22	5	78	30	51	40	50	16	77	60	97	80

Alle Elemente dieser Liste sind in $\mathbb{R}^1$

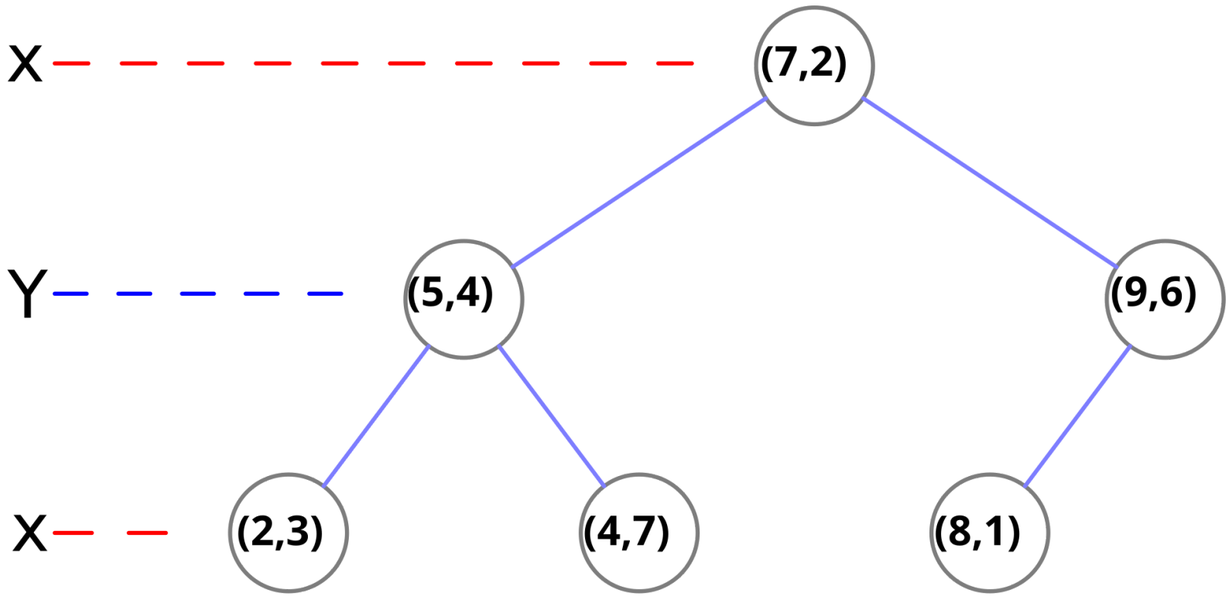
Viele reale Probleme sind aber mehrdimensional!

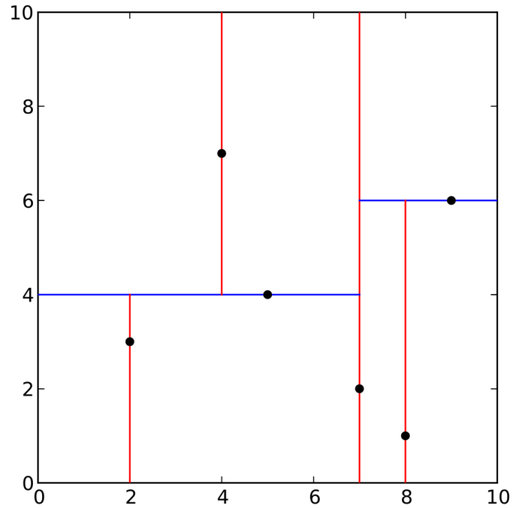
Zweidimensional: Nächste Nachbarn Suche in $\mathbb{R}^2$



Baumstrukturen sind ideal dafür geeignet!

Eine Mögliche Lösung dafür sind kd-Bäume

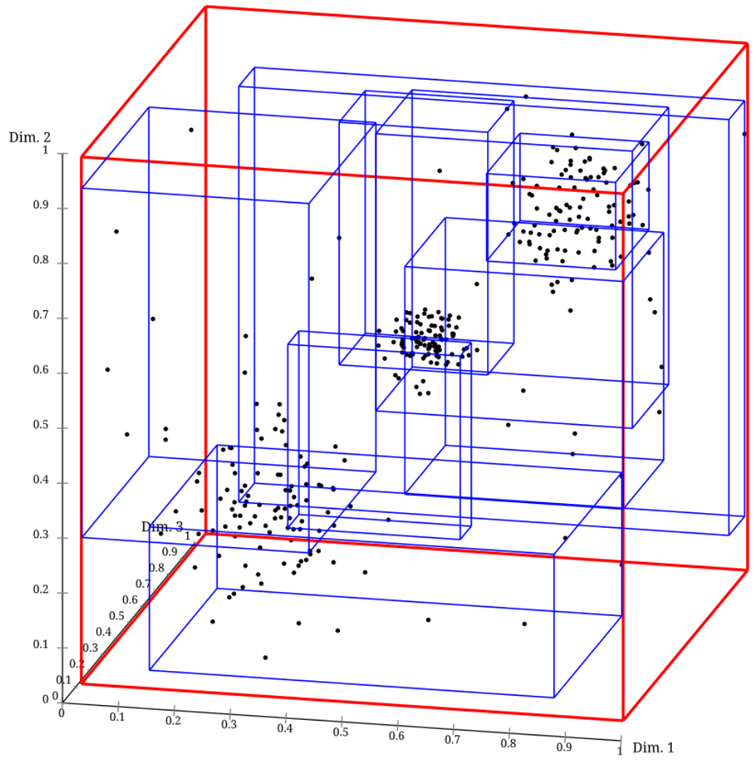




Idee: Abwechselnd in x und y Richtung im Baum suchen.

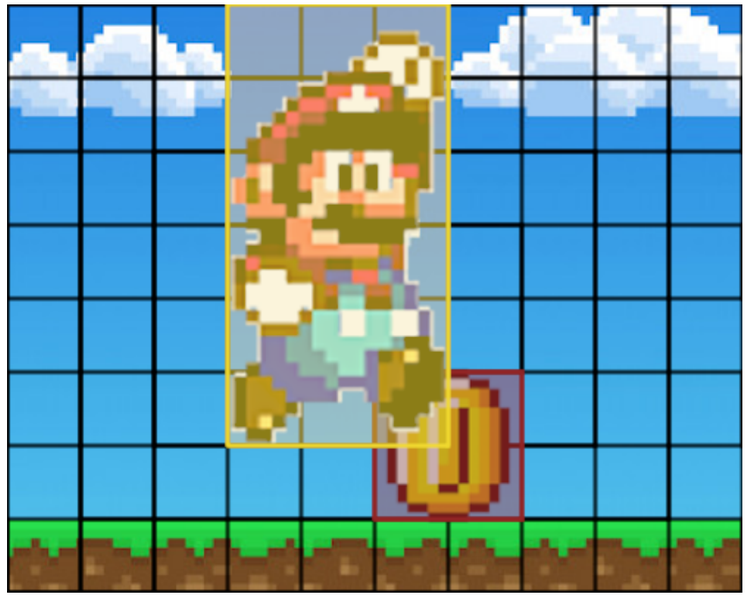
Bildquelle: de.wikipedia.org/wiki/K-d-Baum

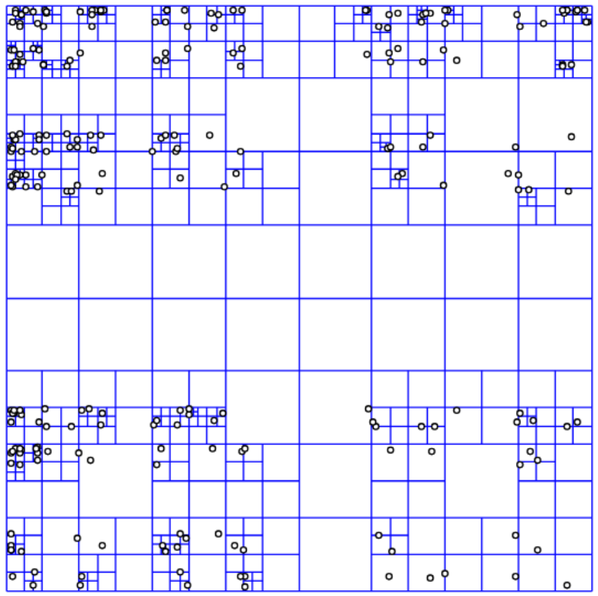
Alternative: R-Bäume



Bildquelle: en.wikipedia.org/wiki/R-tree#/media/File:RTree-Visualization-3D.svg

Weitere Anwendung: Kollisionserkennung in Videospielen



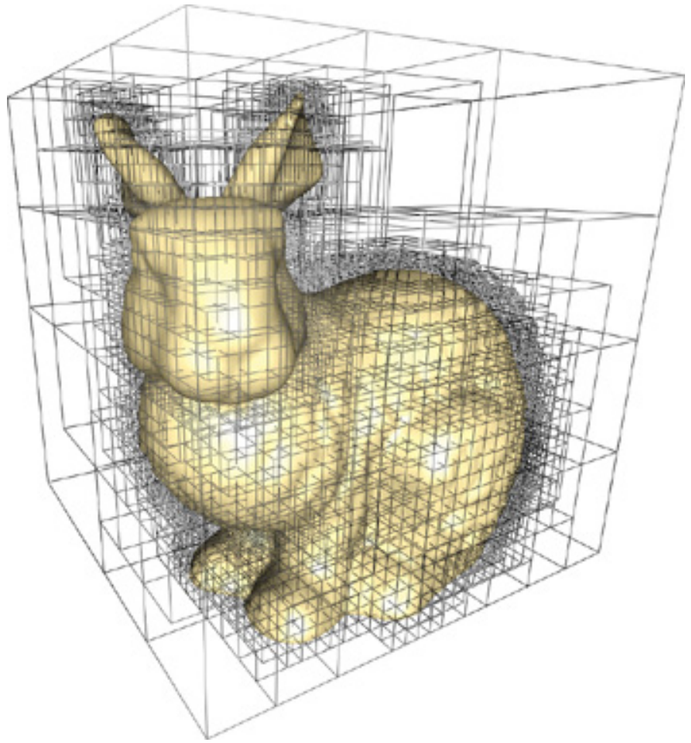


Implementierung durch **Quadtrees**, die den Raum partitionieren.

Bildquelle 1: levelup.gitconnected.com/2d-collision-detection-8e50b6b8b5c0

Bildquelle 2: de.wikipedia.org/wiki/Quadtree

Auch in 3D

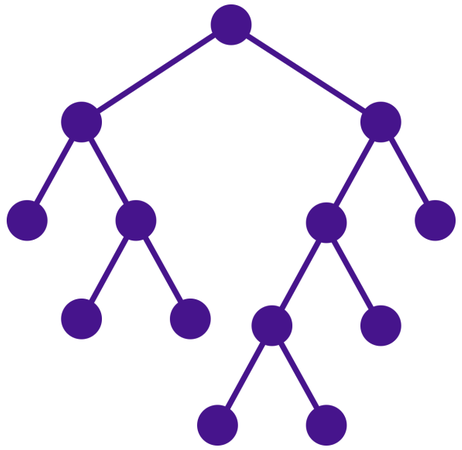


In 3D verwenden wir **Octrees**.

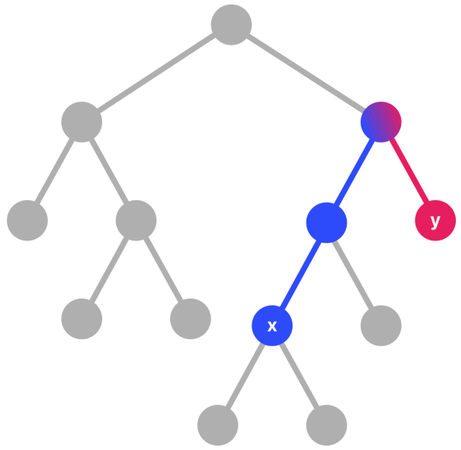
Bildquelle: developer.nvidia.com/gpugems/gpugems2/part-v-image-oriented-computing/chapter-37-octree-textures-gpu

Zusammenfassung

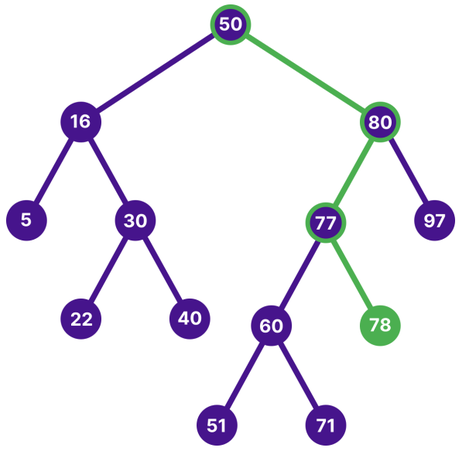
Zusammenfassung



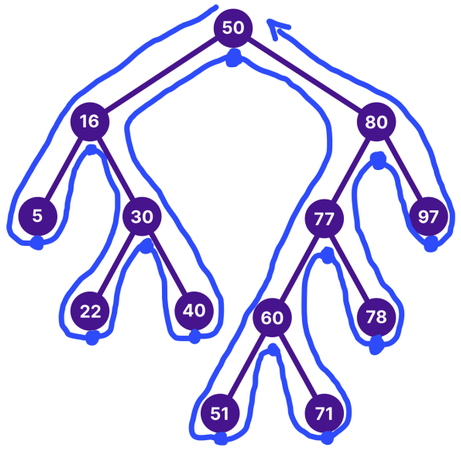
Definition



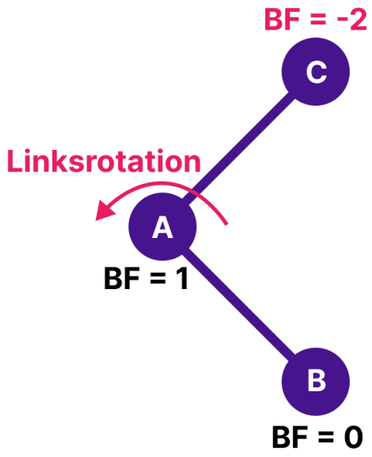
Pathfinder



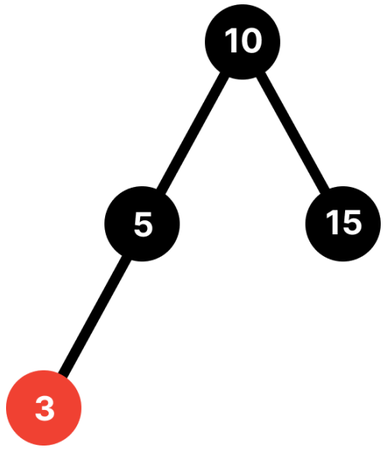
Binärbäume



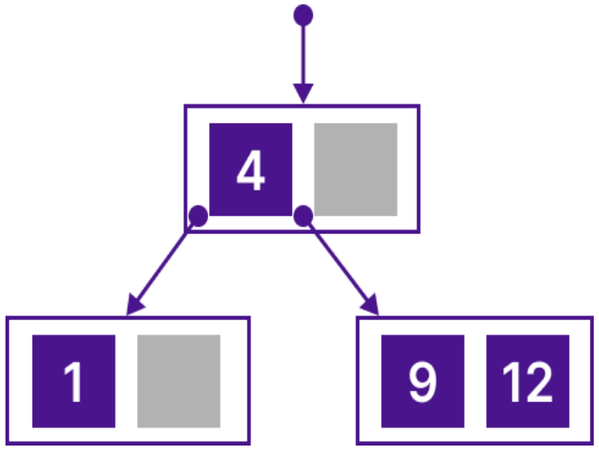
Traversal



AVL Bäume



Rot-Schwarz Bäume



B-Bäume

Fragen?

Viel Erfolg für die erste Teilklausur!