WDH: ADT für eine einfache Symboltabelle

STInterface<Key,Val>		
void	put(Key k, Val v)	Fügt ein Schlüssel-Wert-Paar in die Symboltabelle ein, bzw. entfernt das Paar, wenn der Wert null ist.
Val	get(Key k)	Liefert den Wert zu einem Schlüssel oder null, wenn der Schlüssel nicht enthalten ist.
void	delete(Key k)	Löscht ein Schlüssel-Wert-Paar.
boolean	contains(Key k)	Gibt es einen Wert zu Schlüssel key?
boolean	isEmpty()	Ist die Tabelle leer?
int	size()	Größe der Tabelle
Iterable<Key>	keys()	Schlüssel der Tabelle als iterierbare Sammlung

Quelle: Tabelle nach [1] Seite 390. (modifiziert)

Folie von Birgit Wendholt

Algorithmen & Datenstrukturen

TRIE

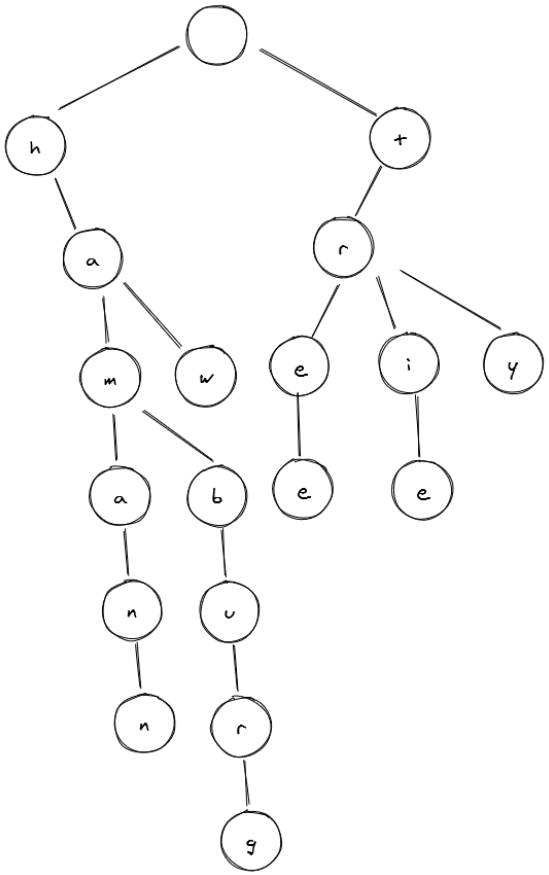
Trie

- Bisherige Datenstrukturen immer
  - Kompletter Schlüsselvergleich: $k_a == k_b$
  - Bei Strings z. B. teuer:
    - String mit n-Zeichen = n-Vergleiche bei jedem Schlüsselvergleich
    - Wird bei O-Notation verschluckt
- Trie[1]
  - 1959 Entwickelt von René *de la Briandais*
  - Namensgebung 1960 von *Edward Fredkin*, da nützlich beim *re**trie**val*
  - Auch Prefix Tree oder Digital Tree genannt
  - Bei der Suche nach einem Schlüssel:
    - Anzahl Vergleiche = Anzahl Zeichen (Bytes) + abschließender Schlüsselvergleich

[1] *In der Regel wird Trie als „try-ee“ oder „try“ ausgesprochen, um von Tree zu unterscheiden*

Trie

- Aufbau
  - Blätter sind in der Regel die Schlüssel
  - Pfad dorthin die einzelnen Character, Bits, …



Enthaltene Schlüssel:

haw

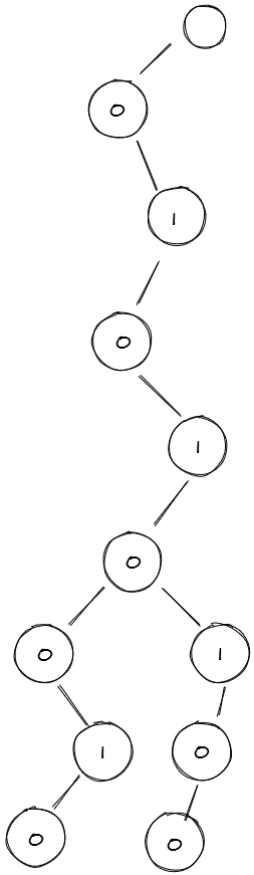
hamburg

hamann

tree

trie

try



Enthaltene Schlüssel:

T = 01010100

R = 01010010

Trie

- Form des Trie ist unabhängig von der Reihenfolge der Einfügungen
  - Beim BST, etc. war das anders!
  - Es gibt einen eindeutigen Trie für eine gegebene Menge an Schlüsseln

- Eigenschaften
  - Im Allgemeinen gut ausbalanciert
  - Zugriff im Durchschnitt logarithmisch
  - Worst case: Länge des längsten Schlüssels

- Anwendungsmöglichkeiten
  - Autocomplete
  - Kompression

Algorithmen & Datenstrukturen

BALANCIERTE BÄUME

Balancierte-Bäume

- Wiederholung
  - Für eine effiziente Suche sollte ein Suchbaum balanciert sein
  - Im Mittel gegeben, so dass Suchen, Einfügen und Entfernen $O(\log_2 n)$
- Bäume können degenerieren
  - Im Extremfall zu einer Liste
  - dann $O(n)$
- Daher verschiedene Ansätze zum Balancieren
- Historisch erster Vorschlag AVL-Bäume (1962)
  - Korrekturoperationen
  - benannt nach Georgi Maximowitsch **A**delson-**V**elski und Jewgeni Michailowitsch **L**andis

Definition: AVL-Baum
- Binärer Suchbaum
- $\forall v \in V$: Höhen der Teilbäume unterscheiden sich maximal um 1

AVL-Bäume

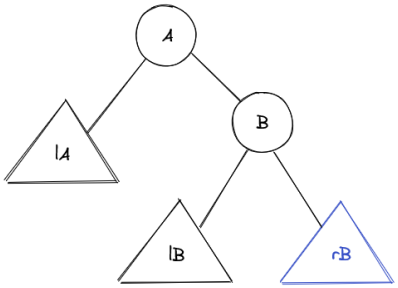
- Rebalancierungs-Operation nötig
  - Um Balancierung (AVL-Eigenschaft) zu erhalten

- Einfügen
  - zunächst wie in einem herkömmlichen Suchbaum einfügen
  - dann: rekursiv vom eingefügten Element bis zur Wurzel
    - AVL-Eigenschaft prüfen
    - bei Bedarf rebalancieren

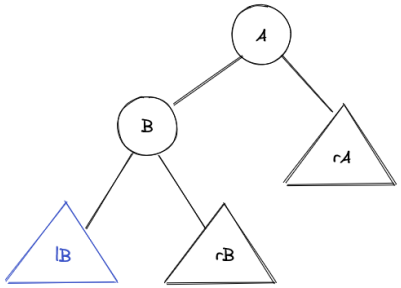
- Entfernen
  - analog

AVL-Eigenschaft

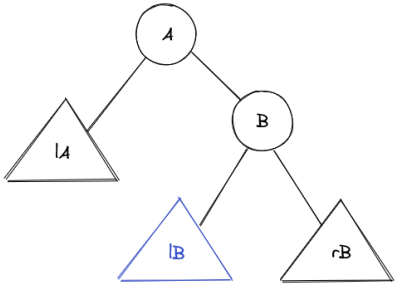
- vier Problemfälle:



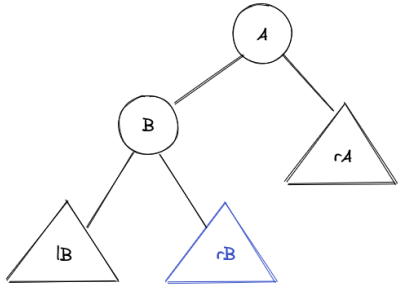
rechts außen:
rB um 2 tiefer als lA



links außen:
lB um 2 tiefer als rA



rechts innen:
lB um 2 tiefer als lA

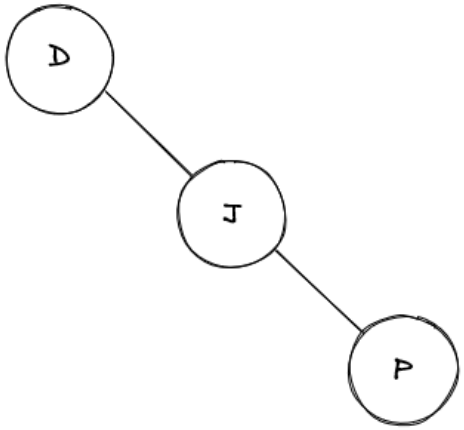


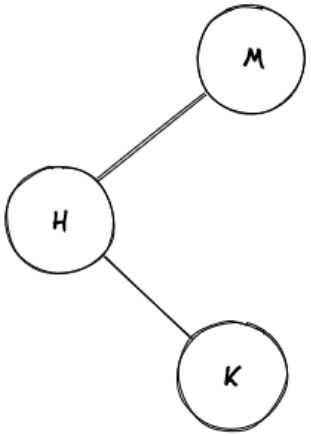
links innen:
rB um 2 tiefer als rA

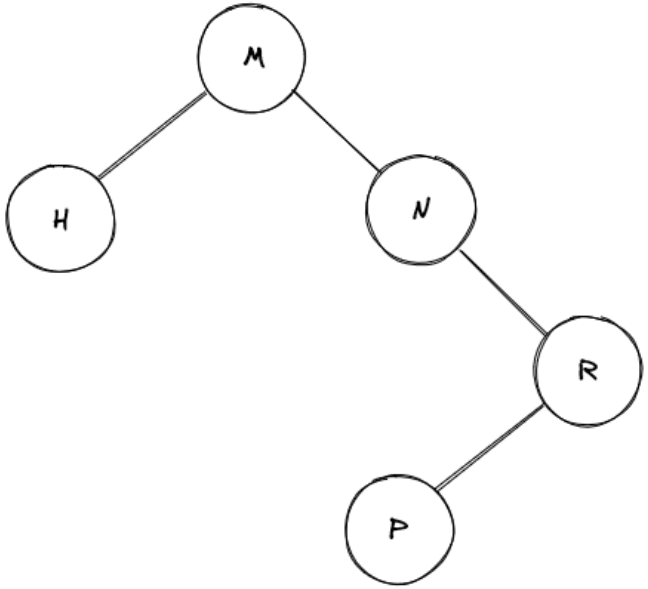
- Lösung: Rotation

Übung

Um welchen Problemfall handelt es sich jeweils?

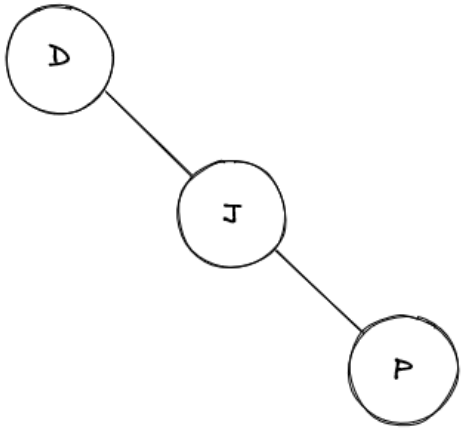


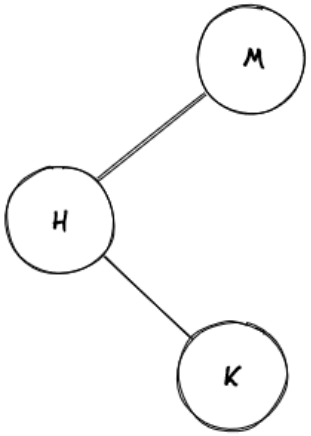


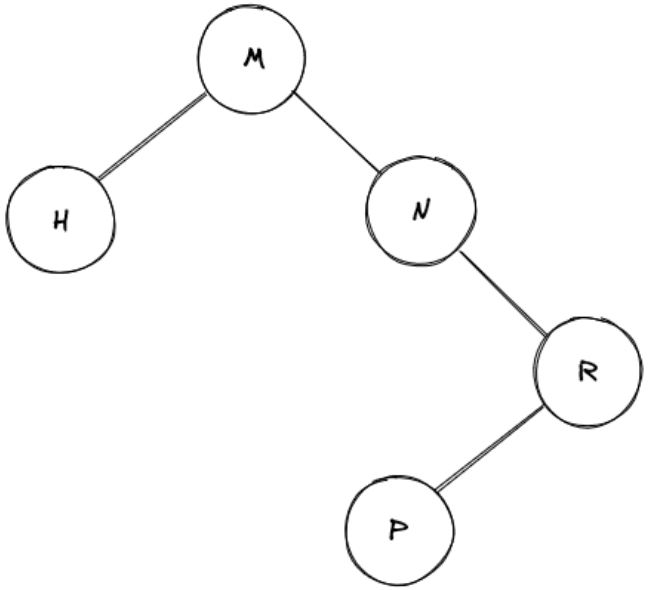


Übung

Um welchen Problemfall handelt es sich jeweils?





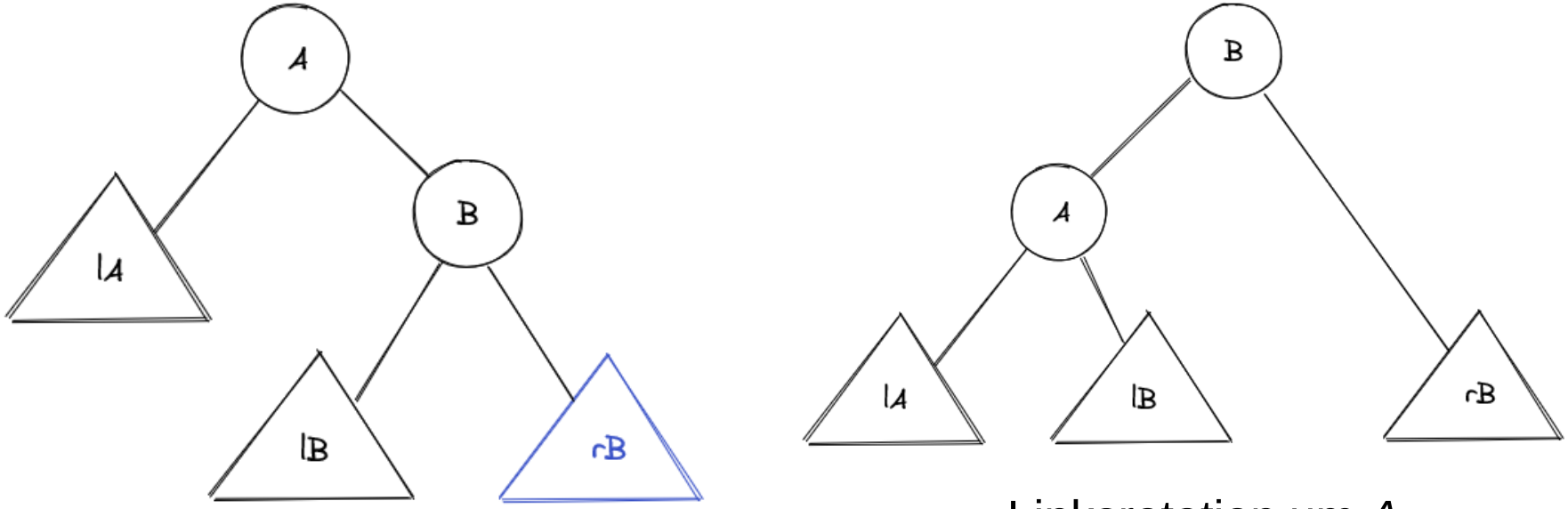


Lösung

- Links: rechts-außen
- Mitte: links-innen
- Rechts: rechts-innen

AVL-Bäume

- Problem: Rechtsaußen
- Lösung: Linksrotation

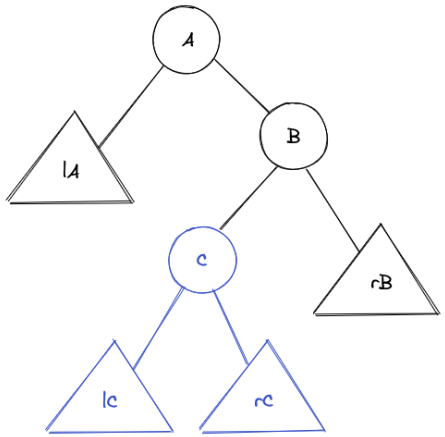


Linksrotation um *A*

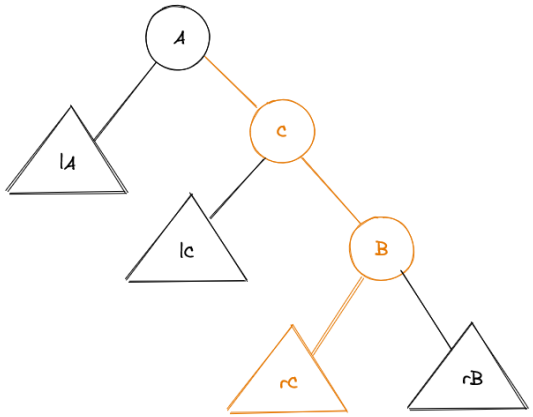
- Rechtsaußen: *rB* um 2 tiefer als *lA*
  - vorher: Gesamthöhe = n + 3 (|lA| = n, |rB| = n + 1)
  - nachher: Gesamthöhe = n + 2

AVL-Bäume

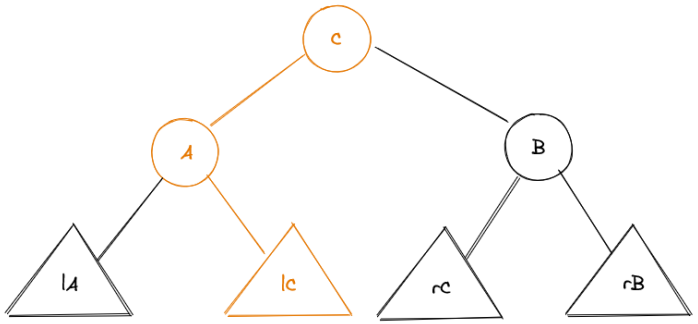
- Problem: Rechts-Innen
- Lösung: Doppelrotation Links



Rechts-Innen: *lB* um 2 tiefer als *lA*



Rechtsrotation um *B*



Linksrotation um A

- vorher: Gesamthöhe = n + 3 (|lA| = n, |rB| = n oder n – 1)
- nachher: Gesamthöhe = n + 2

AVL-Bäume

- Weitere problematische Konstellationen
  - Problem links-außen: Rechtsrotation
  - Problem links-innen: Doppelrotation Rechts

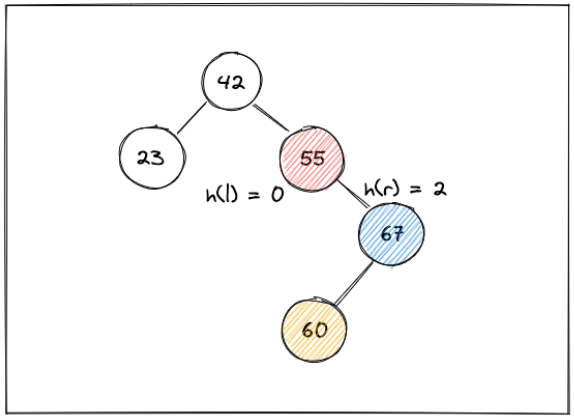
- Interaktive Demo
  - https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

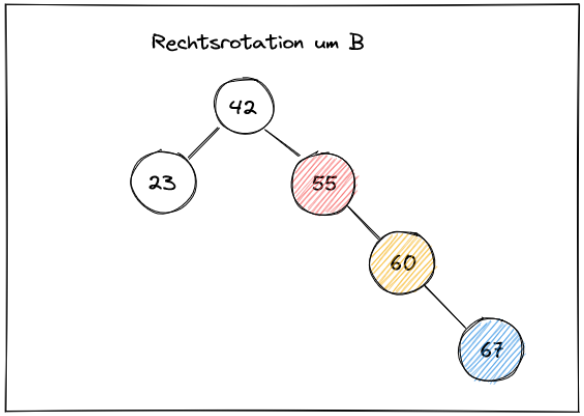
AVL-Bäume Beispiel

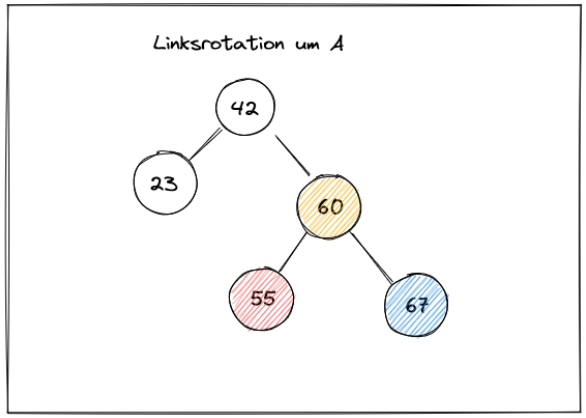
- Folge von Schlüsseln: 23, 42, 55, 67, 60

AVL-Bäume Beispiel

- Folge von Schlüsseln: 23, 42, 55, 67, 60

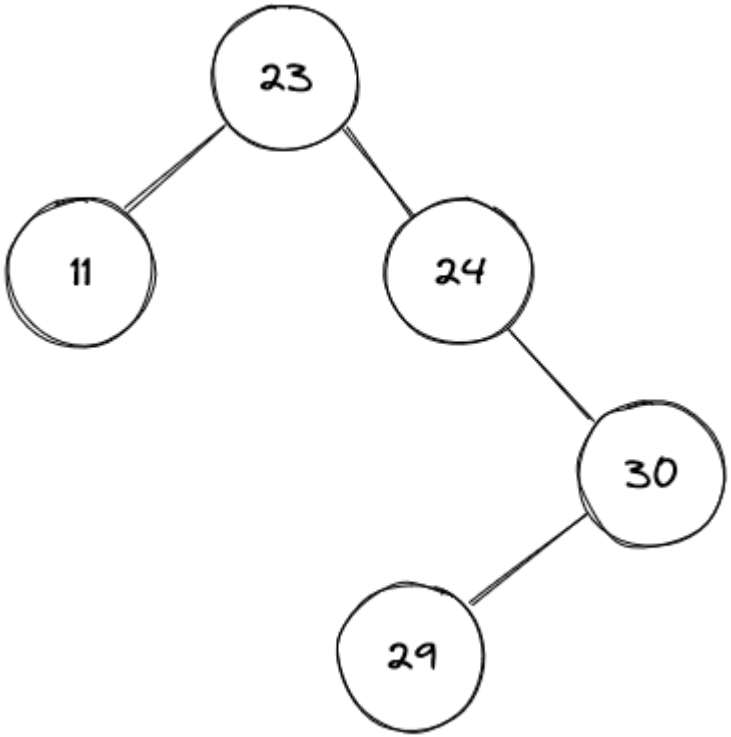






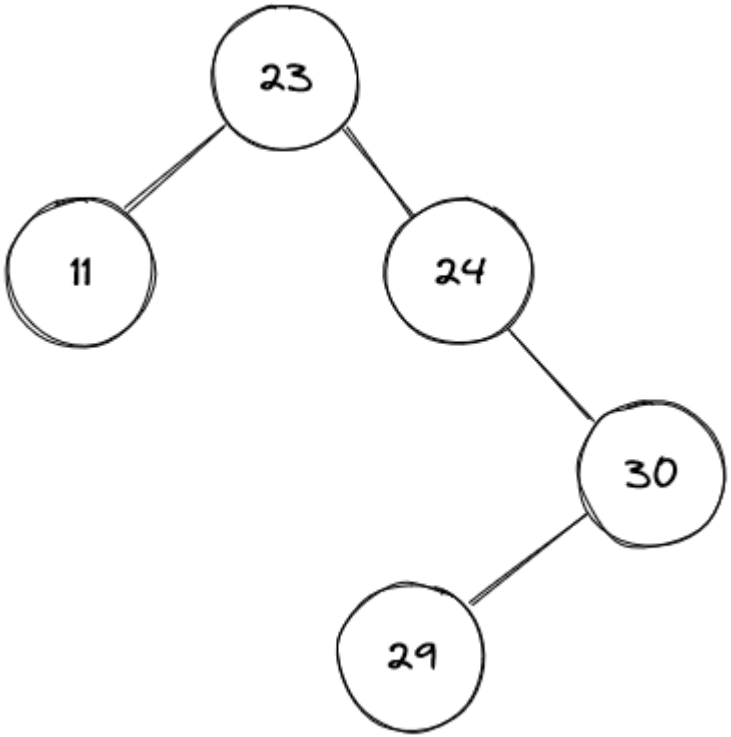
Übung

Führen Sie auf dem folgenden AVL-Baum die notwendigen Korrekturoperationen durch.



Übung

Führen Sie auf dem folgenden AVL-Baum die notwendigen Korrekturoperationen durch.



Lösung

- Problem: rechts-innen bei 24
- Lösung: Doppelrotation links bei 24