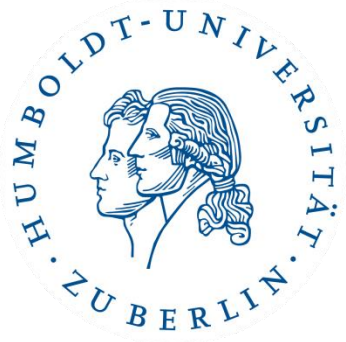
Übung Algorithmen und Datenstrukturen



Sommersemester 2017

Marc Bux, Humboldt-Universität zu Berlin

Agenda

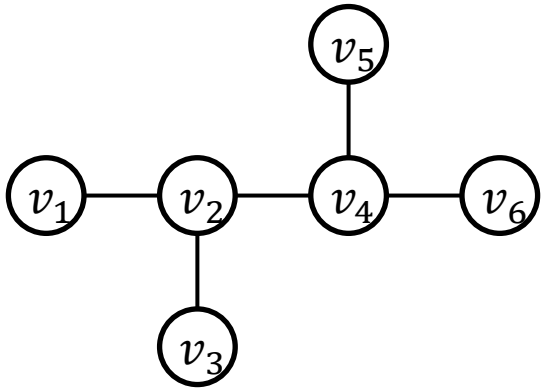
1. Graphen und Bäume
2. Binäre Suchbäume
3. AVL-Bäume
4. Algorithmen und Datenstrukturen

Agenda

1. Graphen und Bäume
2. Binäre Suchbäume
3. AVL-Bäume
4. Algorithmen und Datenstrukturen

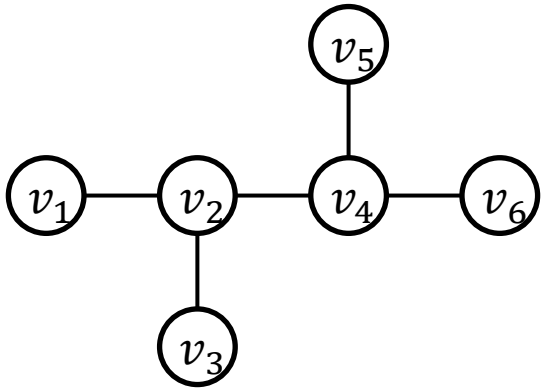
Graphen und Bäume

- Graph: Ein Graph $G = (V, E)$ besteht aus einer Menge V von Knoten und einer Menge $E \subseteq V \times V$ von Kanten.
  - Für einen ungerichteten Graphen gilt: $\forall (v, w) \in E \Rightarrow (w, v) \in E$.
  - Eine Folge von Knoten $v_1, v_2, \dots, v_n$, in der für aufeinanderfolgende Knoten v_i, v_{i+1} gilt $(v_i, v_{i+1}) \in E$ ist ein Pfad (Weg).
  - Ein Zyklus (Kreis) ist ein Pfad mit $v_1 = v_n$.
  - Ein Graph ist zusammenhängend, wenn für jedes Paar $(v, w) \in V \times V$ ein Pfad von v nach w existiert.
- Baum: Ein Baum ist ein ungerichteter, kreisfreier und zusammenhängender Graph.
- Traversierung in $O(|V| + |E|)$
  - für zusammenhängende Graphen in $O(|E|)$, da hier $|V| \in O(|E|)$



Implementierung von Graphen

- Adjazenzlisten:
  - Speicherbedarf $O(|V| + |E|)$
  - Zugriff auf bestimmte ausgehende Kante von v in $O(\text{out}(v))$
  - Iteration über alle Kanten eines Knotens v in $O(\text{out}(v))$
- Adjazenzmatrix:
  - Speicherbedarf $O(|V|^2)$
  - Zugriff auf bestimmte ausgehende Kante von v in $O(1)$
  - Iteration über alle Kanten eines Knotens v in $O(|V|)$
- Hinweise: $|E| \leq |V|^2$, $0 \leq \text{out}(v) \leq |V|$



$v_1: \{v_2\}$

$v_2: \{v_1, v_3, v_4\}$

$v_3: \{v_2\}$

$v_4: \{v_2, v_5, v_6\}$

$v_5: \{v_4\}$

$v_6: \{v_4\}$

	v_1	v_2	v_3	v_4	v_5	v_6
v_1		1				
v_2	1		1	1		
v_3		1				
v_4		1			1	1
v_5				1		
v_6				1		

Traversierung von Graphen (und Bäumen)

- Tiefensuche
  - a.k.a. Depth-First-Search, DFS
  - traversiere in die Tiefe
  - realisiert durch Stack oder Rekursion

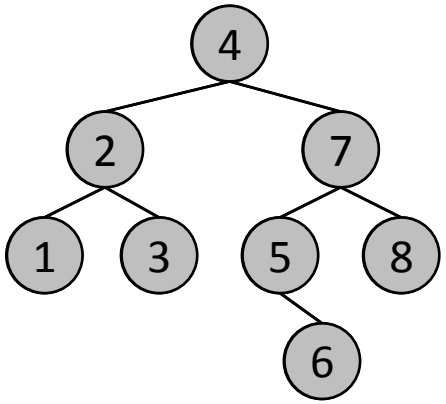
```
DFS(Node root)
1.    if (root = null) return;
2.    visit(root); root.marked = true;
3.    for each neighbor of root
4.        if (not neighbor.marked)
5.            DFS(neighbor);
```

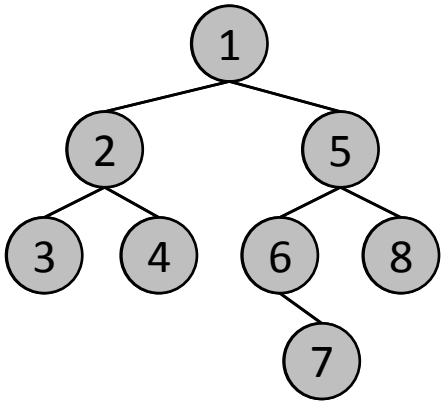
- Breitensuche
  - a.k.a. Breadth-First-Search, BFS
  - traversiere in die Breite
  - realisiert durch Queue

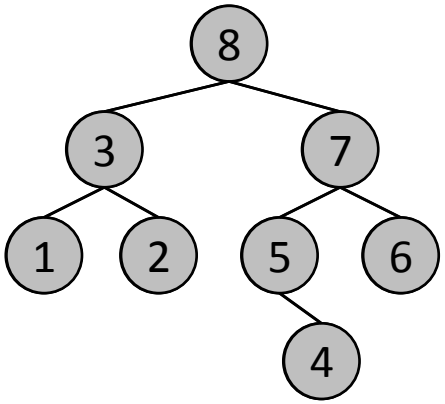
```
BFS(Node root)
1.    Initialisiere Queue q; q.enqueue(root);
2.    while (not q.isEmpty())
3.        node := q.dequeue();
4.        visit(node);
5.        for each (Node neighbor of node)
6.            if (not neighbor.marked)
7.                neighbor.marked = true;
8.                q.enqueue(neighbor);
```

Traversierung binärer (gewurzelter) Bäume

- In-Order-Traversierung: Besuche linken Teilbaum, dann aktuellen Knoten, dann rechten Teilbaum
- Pre-Order-Traversierung: Besuche aktuellen Knoten, dann linken Teilbaum, dann rechten Teilbaum
- Post-Order-Traversierung: Besuche linken Teilbaum, dann rechten Teilbaum, dann aktuellen Knoten
- bei allen o.g. Traveriserungen handelt es sich um Spezialformen der Tiefensuche





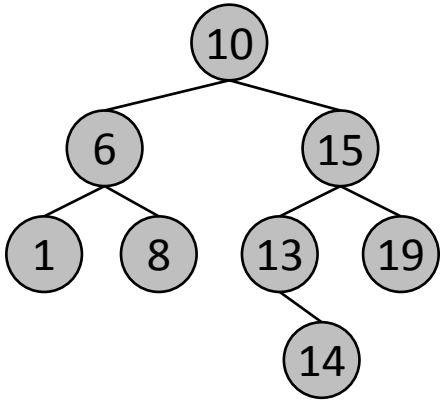


Agenda

1. Graphen und Bäume
2. Binäre Suchbäume
3. AVL-Bäume
4. Algorithmen und Datenstrukturen

Binäre Suchbäume

- Knoten
  - beinhaltet „Schlüssel“
  - hat Verweis auf linkes und rechtes Kind
- Sortierung/Sucheigenschaft
  - alle Schlüssel des linken Teilbaums eines Knotens sind kleiner (oder gleich, falls Duplikate erlaubt sind)
  - alle Schlüssel des rechten Teilbaums eines Knotens sind größer (oder gleich, falls Duplikate erlaubt sind)

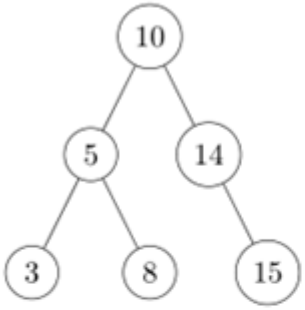


- Beim Löschen eines Knotens mit zwei Kindern muss mit dem symmetrischen Vorgänger oder Nachfolger getauscht werden
  - symmetrischer Vorgänger (Nachfolger): Knoten mit dem größten (kleinsten) Schlüssel im linken (rechten) Teilbaum

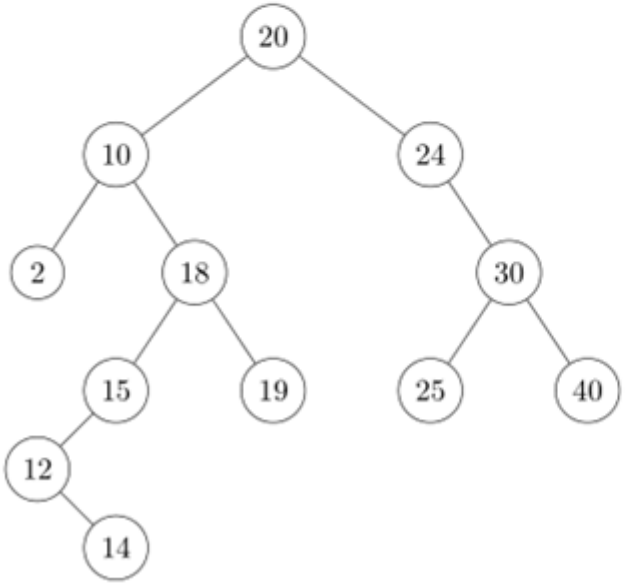
Aufgabe zu Binären Suchbäumen

In dieser Aufgabe sollen Sie einen Schreibtischtest mit binären Suchbäumen ausführen.

a) Führen Sie nun die Operationen `insert(16)`, `insert(11)` und `delete(8)` in dieser Reihenfolge aus. Geben Sie nach jeder Operation den neuen Baum an.



b) Führen Sie in dem folgenden Suchbaum die Operationen `delete(24)` und `delete(10)` in dieser Reihenfolge aus. Geben Sie nach jeder Operation den neuen Baum an.

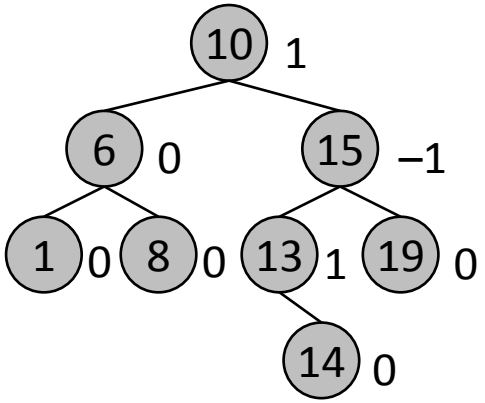


Agenda

1. Graphen und Bäume
2. Binäre Suchbäume
3. AVL-Bäume
4. Algorithmen und Datenstrukturen

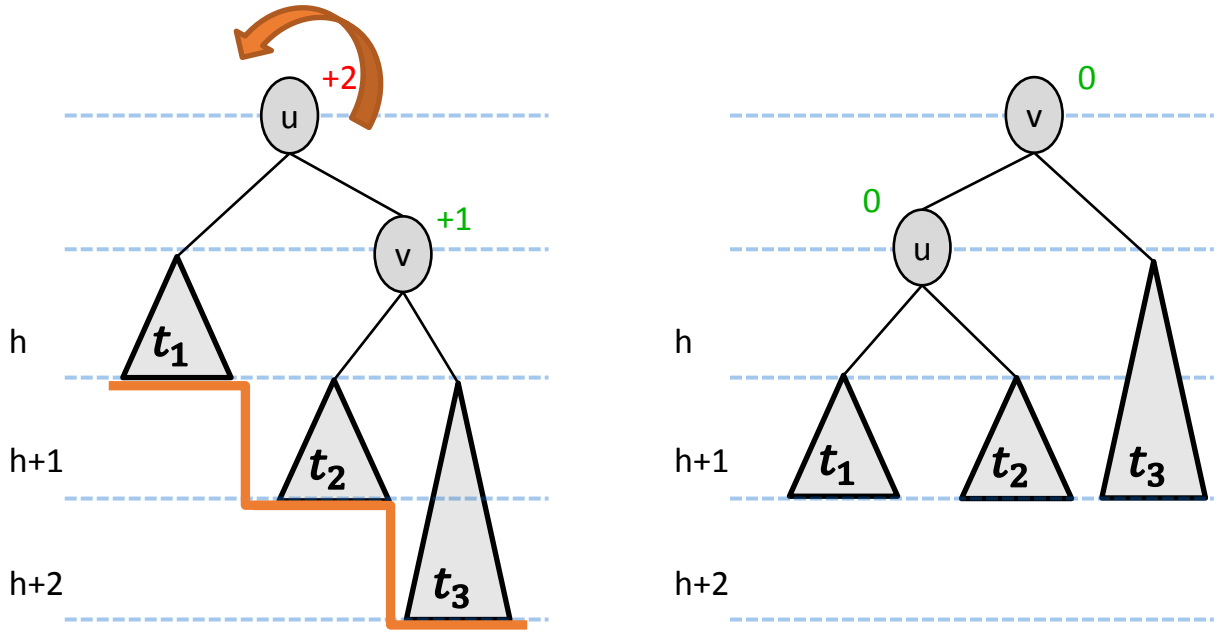
AVL-Bäume

- benannt nach Erfindern Adelson-Velsky und Landis (AVL)
- älteste Datenstruktur für balancierte Suchbäume
- binärer Suchbaum mit zusätzlicher Eigenschaft:
  - in jedem Knoten unterscheidet sich die Höhe der beiden Teilbäume um höchstens eins
  - Höhe logarithmisch in Anzahl der Schlüssel
- Rebalancierung (Rotation) beim Einfügen und Löschen
- Definition:
  - Sei u Knoten in binärem Baum.
  - $\mathrm{bal}(u)$: Differenz zwischen Höhe des rechten Teilbaums von u und Höhe des linkes Teilbaums von u
  - Ein binärer Baum heißt AVL-Baum, falls für alle Knoten u gilt: $|\mathrm{bal}(u)| \leq 1$



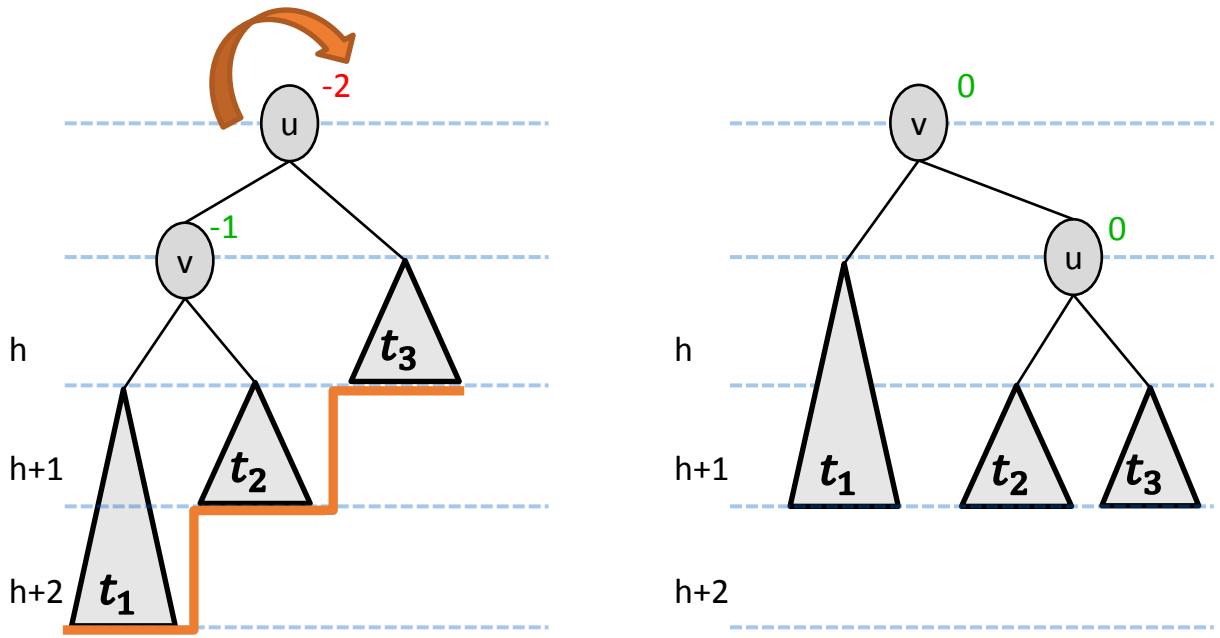
Rebalancierung von AVL-Bäumen

- Sei u Knoten, v Kind von u im Teilbaum mit größerer Höhe
- 4 Rotationsoperationen auf AVL-Bäumen:
  1. $\text{bal}(u) = 2, \text{bal}(v) \geq 0$: Einfachrotation links($u$)



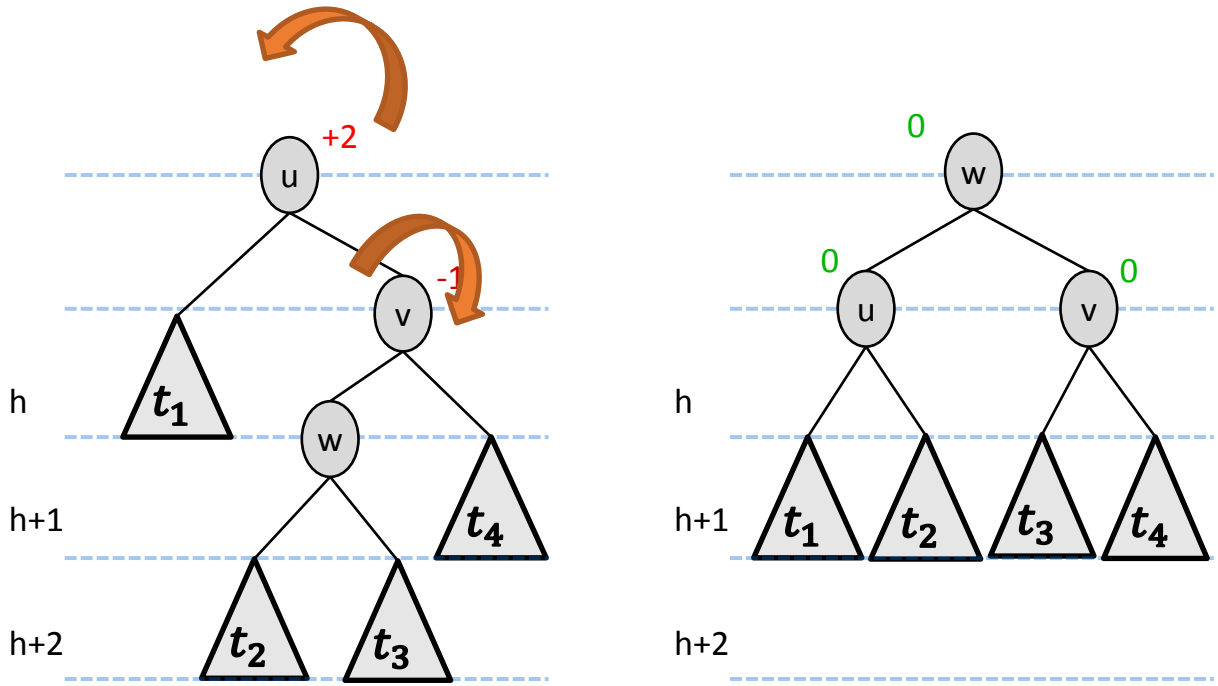
Rebalancierung von AVL-Bäumen

- Sei u Knoten, v Kind von u im Teilbaum mit größerer Höhe
- 4 Rotationsoperationen auf AVL-Bäumen:
  1. $\text{bal}(u) = 2, \text{bal}(v) \geq 0$: Einfachrotation links($u$)
  2. $\text{bal}(u) = -2, \text{bal}(v) \leq 0$: Einfachrotation Rechts(u)



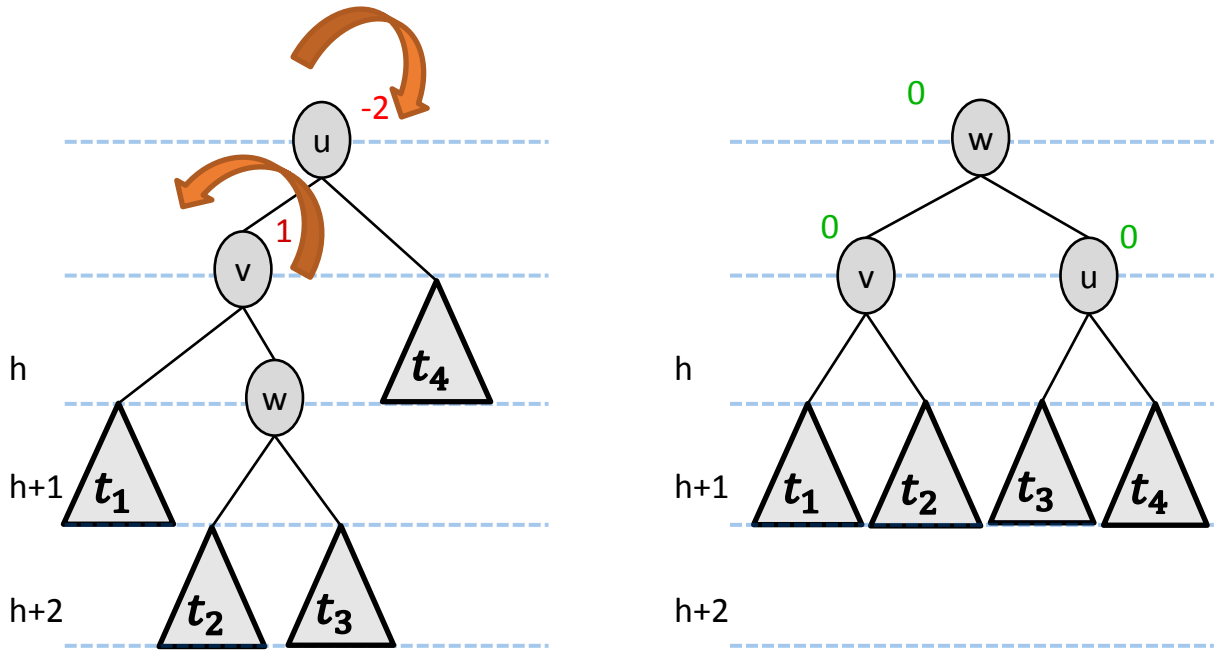
Rebalancierung von AVL-Bäumen

- Sei u Knoten, v Kind von u im Teilbaum mit größerer Höhe
- 4 Rotationsoperationen auf AVL-Bäumen:
  1. $\text{bal}(u) = 2, \text{bal}(v) \geq 0$: Einfachrotation links($u$)
  2. $\text{bal}(u) = -2, \text{bal}(v) \leq 0$: Einfachrotation Rechts(u)
  3. $\text{bal}(u) = 2, \text{bal}(v) = -1$: Doppelrotation Rechts($v$) + Links($u$)



Rebalancierung von AVL-Bäumen

- Sei u Knoten, v Kind von u im Teilbaum mit größerer Höhe
- 4 Rotationsoperationen auf AVL-Bäumen:
  1. $\text{bal}(u) = 2, \text{bal}(v) \geq 0$: Einfachrotation links($u$)
  2. $\text{bal}(u) = -2, \text{bal}(v) \leq 0$: Einfachrotation Rechts(u)
  3. $\text{bal}(u) = 2, \text{bal}(v) = -1$: Doppelrotation Rechts($v$) + Links($u$)
  4. $\text{bal}(u) = -2, \text{bal}(v) = 1$: Doppelrotation Links($v$) + Rechts($u$)

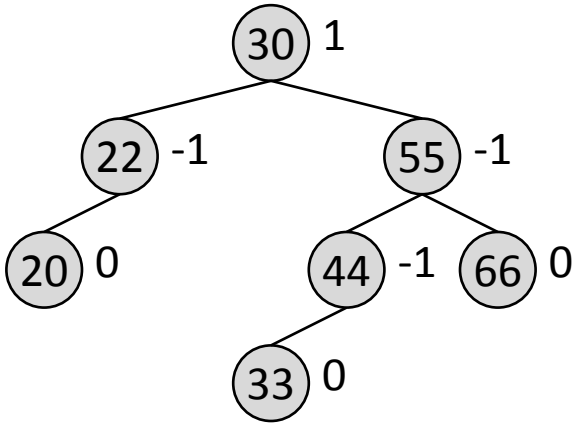


Rebalancierung von AVL-Bäumen

- Sei u Knoten, v Kind von u im Teilbaum mit größerer Höhe
- 4 Rotationsoperationen auf AVL-Bäumen:
  1. $\text{bal}(u) = 2, \text{bal}(v) = 1$: Einfachrotation links($u$)
  2. $\text{bal}(u) = -2, \text{bal}(v) = -1$: Einfachrotation Rechts(u)
  3. $\text{bal}(u) = 2, \text{bal}(v) = -1$: Doppelrotation Rechts($v$) + Links($u$)
  4. $\text{bal}(u) = -2, \text{bal}(v) = 1$: Doppelrotation Links($v$) + Rechts($u$)
- Hinweise:
  - nach Ausführen einer Rotationoperation sind (entlang des Suchpfades von unten nach oben) ggf. weitere Rotationoperationen notwendig
  - es gibt Beispiele, bei denen Rotationsoperationen für alle Knoten entlang des Suchpfades durchgeführt werden müssen
- Laufzeit:
  - Rotationen sind lokale Operationen, die nur Umsetzen einiger Zeiger erfordern, und in Zeit $\mathcal{O}(1)$ erfolgen
  - aus logarithmischer Höhe des Baums ergibt sich Laufzeit $\mathrm{O}(\log n)$ für das Einfügen, Suchen und Löschen

Aufgabe zu AVL-Bäumen

Sei *T* folgender AVL-Baum:



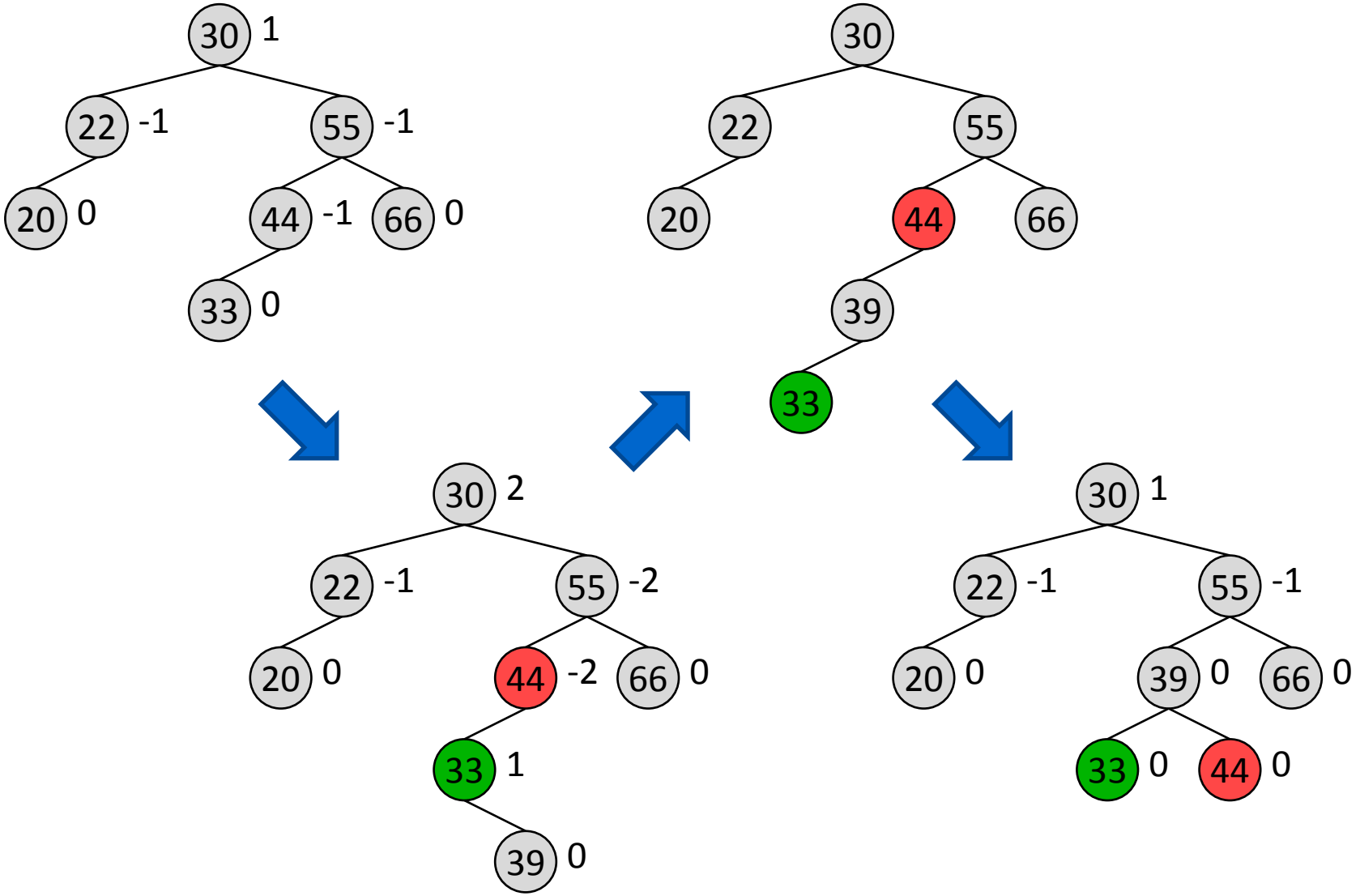
Fügen Sie nacheinander die Schlüssel

39, 42

in *T* ein und zeichnen Sie den jeweiligen AVL-Baum nach jeder insert-Operation.

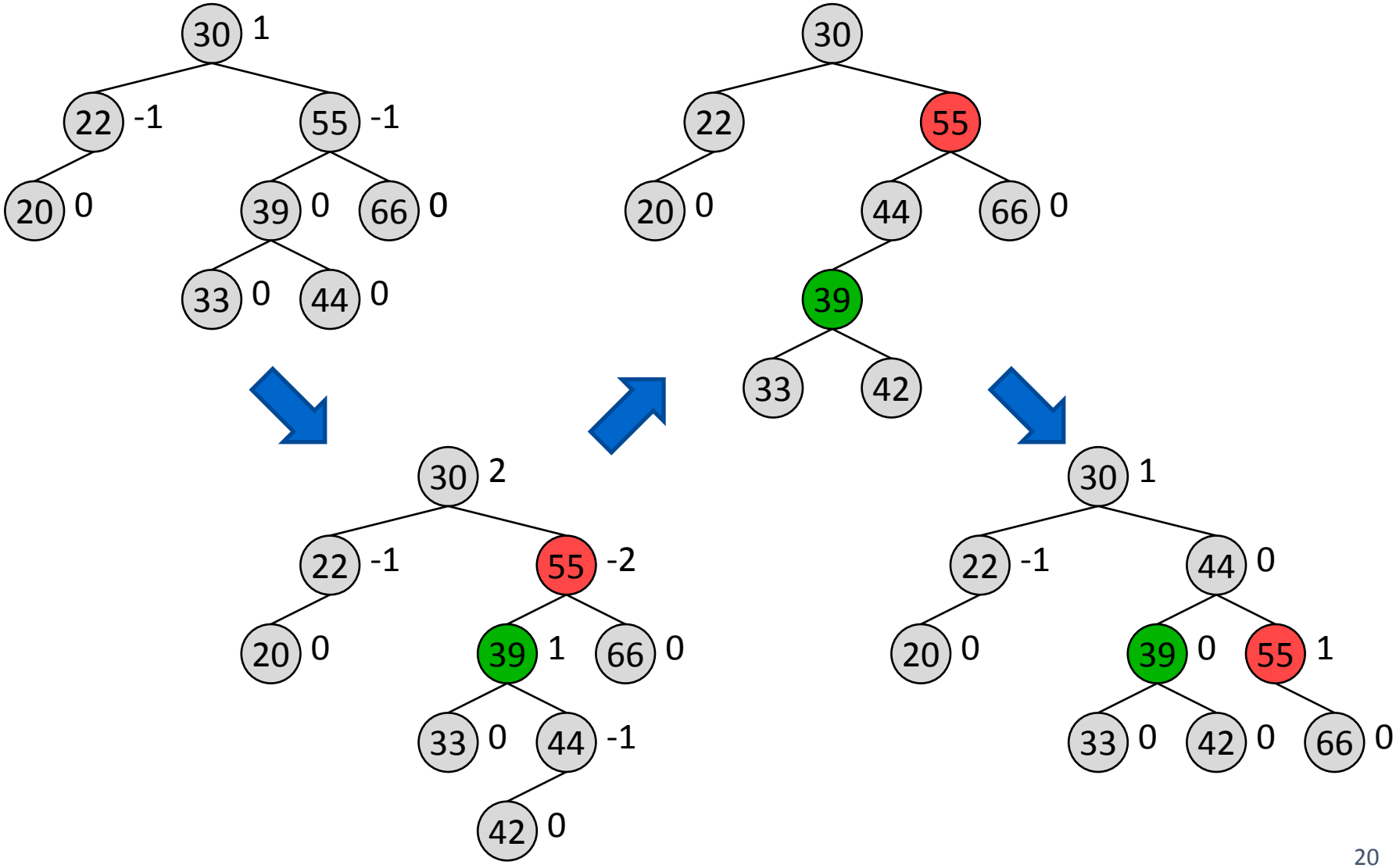
Einfügen von 39

$\mathrm{bal}(u) = -2$, $\mathrm{bal}(v) = 1$: Doppelrotation Links(v) + Rechts(u)



Einfügen von 42

$bal(u) = -2$, $bal(v) = 1$: Doppelrotation Links(v) + Rechts(u)



Agenda

1. Graphen und Bäume
2. Binäre Suchbäume
3. AVL-Bäume
4. Algorithmen und Datenstrukturen

Datenstrukturen…

Datenstruktur	Einfügen	Suchen	Besonderheit	Java-Interface	Java-Klassen (Auswahl)
Array / String	N/A	$O(n)$	Indexbasierter Zugriff in $O(1)$		[], String
Liste	$O(1)$	$O(n)$		List	LinkedList, ArrayList
Stack	$O(1)$	$O(n)$	Zugriff auf zuletzt eingefügtes Element in O(1)		Stack
Queue	$O(1)$	$O(n)$	Zugriff auf zuerst eingefügtes Element in O(1)	Queue	LinkedList
Binärer Heap	$O(\log n)$	$O(n)$	Zugriff auf Min / Max in O(1)		PriorityQueue
Hash-Tabelle	$O(1)$*	$O(1)$*	Konstante Laufzeit nur im Average Case		HashMap, HashSet
balancierter Suchbaum	$O(\log n)$	$O(\log n)$			TreeMap, TreeSet

… & Algorithmen

- Allgemeine Sortierverfahren: SelectionSort, InsertionSort, BubbleSort, QuickSort, MergeSort, (HeapSort)
- Lineare Sortierverfahren: RadixSort, BucketSort
- Suche in sortierten Arrays: Binäre Suche, Fibonacci-Suche, Interpolationssuche, Exponentielle Suche
- Traversierung von Graphen: DFS, BFS
- Traversierung von Bäumen: In-Order, Pre-Order, Post-Order
- Kürzeste Pfade in Graphen: Dijkstra (Single-Source), Floyd (All-Pairs)
- (starke) Zusammenhangskomponenten: Kosaraju
- Minimaler Spannbaum: Prim, Kruskal, Boruvka

Weitere Themenkomplexe

(ohne Anspruch auf Vollständigkeit)

- Landau-Notation und Laufzeitanalysen
- Selbstorganisierende Listen
- Amortisierte Analyse
- Optimale Suchbäume
- Präfixbäume / Tries
- …

Ausblick

- nächste Woche:
  - Aufgabenblatt 5 durchrechnen
- zu nächster Woche:
  - mit Aufgabenblatt 6 auseinandersetzen
  - Generics in Java:
    https://docs.oracle.com/javase/tutorial/java/generics/
  - Function-Interface und Lambda-Ausdrücke in Java 8:
    https://www.codementor.io/eh3rrera/using-java-8-method-reference-du10866vx
  - falls noch nicht vorgerechnet: zum Vorrechnen eintragen
    Aufgabenblatt 5, Montag: https://dudle.inf.tu-dresden.de/algodat25
    Aufgabenblatt 5, Dienstag: https://dudle.inf.tu-dresden.de/algodat35
    Aufgabenblatt 6, Montag: https://dudle.inf.tu-dresden.de/algodat26
    Aufgabenblatt 6, Dienstag: https://dudle.inf.tu-dresden.de/algodat36
    (first-come-first-served)
  - Vorrechnen ist Voraussetzung für Erhalt des Übungsscheins (der Prüfungszulassung)