Informatik II

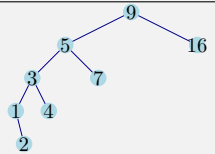
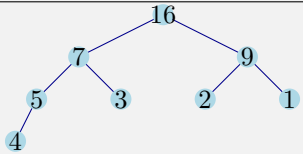
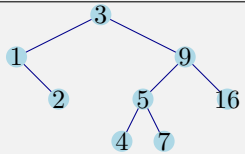
Übung 7

FS 2019

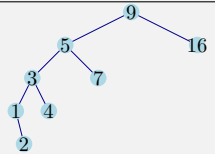
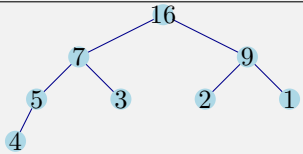
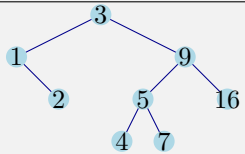
Heutiges Programm

1. Rekapitulation binäre Bäume
2. Wiederholung Vorlesung
   - AVL Bedingung
   - AVL Einfügen
3. In-Class-Exercises

Vergleich binärer Bäume

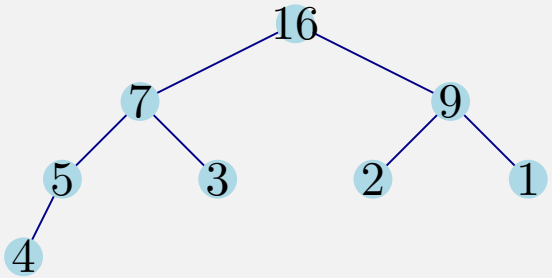
	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in Java:		PriorityQueue	TreeSet
			
Einfügen	$\mathcal{O}(h(T))$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Suchen	$\mathcal{O}(h(T))$	$\mathcal{O}(n)$ (!!)	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(h(T))$	Suchen + $\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Vergleich binärer Bäume

	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in Java:		PriorityQueue	TreeSet
			
Einfügen	$\mathcal{O}(h(T))$	$\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$
Suchen	$\mathcal{O}(h(T))$	$\mathcal{O}(n)$ (!!)	$\mathcal{O}(\log n)$
Löschen	$\mathcal{O}(h(T))$	Suchen + $\mathcal{O}(\log n)$	$\mathcal{O}(\log n)$

Bemerke: $\mathcal{O}(\log n) \leq \mathcal{O}(h(T)) \leq \mathcal{O}(n)$

Wdh: Pre- / In- / Post- Order

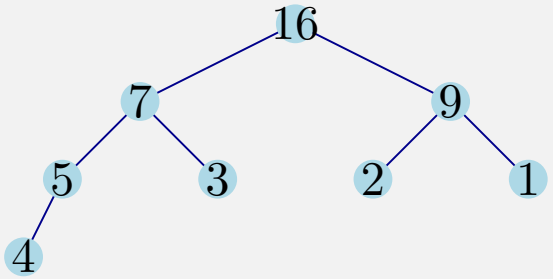


Pre-order:

In-order:

Post-order:

Wdh: Pre- / In- / Post- Order

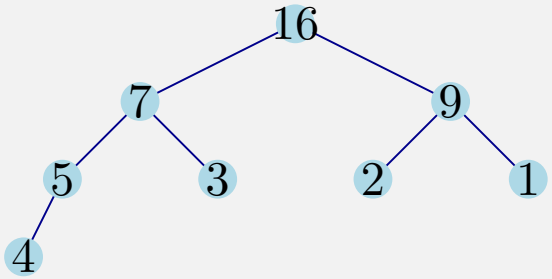


Pre-order: 16 7 5 4 3 9 2 1

In-order:

Post-order:

Wdh: Pre- / In- / Post- Order

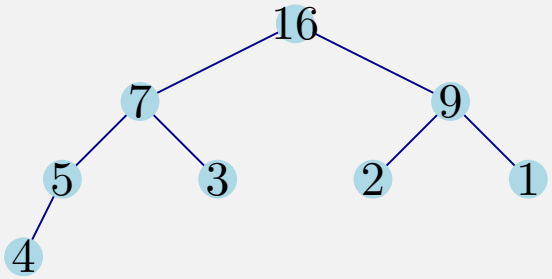


Pre-order: 16 7 5 4 3 9 2 1

In-order: 4 5 7 3 16 2 9 1

Post-order:

Wdh: Pre- / In- / Post- Order



Pre-order: 16 7 5 4 3 9 2 1

In-order: 4 5 7 3 16 2 9 1

Post-order: 4 5 3 7 2 1 9 16

Wiederholung: Binäre Bäume, Schlüssel Einfügen

Binäre Suchbäume

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.

MinHeap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

Wiederholung: Binäre Bäume, Schlüssel Einfügen

Binäre Suchbäume

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.

MinHeap

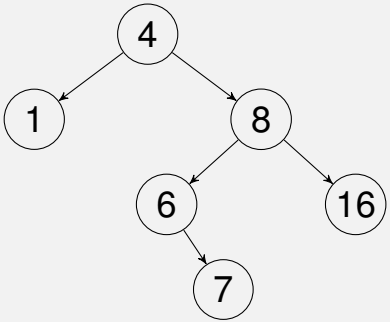
- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

Aufgabe: Einfügen von $4, 8, 16, 1, 6, 7$ in leeren Baum/Heap.

Wiederholung: Binäre Bäume, Schlüssel Einfügen

Binäre Suchbäume

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.



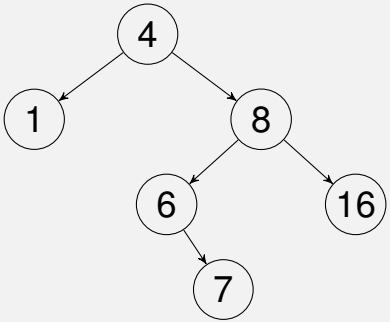
MinHeap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

Wiederholung: Binäre Bäume, Schlüssel Einfügen

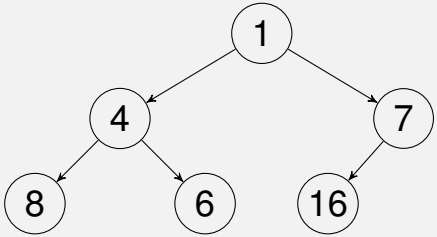
Binäre Suchbäume

- Nach Schlüssel suchen.
- Bei erreichtem leeren Blatt (`null`) einfügen.



MinHeap

- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).



Wiederholung: Binäre Bäume, Schlüssel Löschen

Binäre Suchbäume

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?

MinHeap

- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *or* `siftUp`.

Wiederholung: Binäre Bäume, Schlüssel Löschen

Binäre Suchbäume

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?

MinHeap

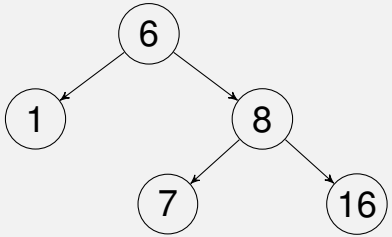
- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *or* `siftUp`.

Aufgabe: Löschen von 4 in Beispiel-Baum/Heap.

Wiederholung: Binäre Bäume, Schlüssel Löschen

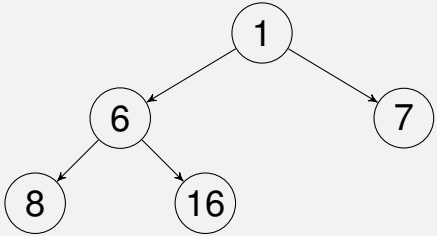
Binäre Suchbäume

- Schlüssel k durch symm. Nachfolger n ersetzen.
- Achtung: Wohin mit rechtem Kind von n?



MinHeap

- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *or* `siftUp`.



Java: Löschen aus MinHeap

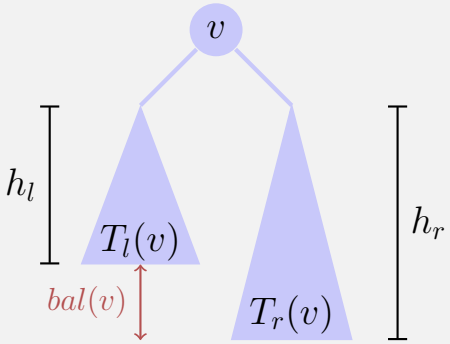
Problem! Wie finden wir einen Schlüssel im MinHeap?

$\Rightarrow$ Wir beschränken uns üblicherweise darauf, Wurzeln zu löschen (Extract-Min).

Balance eines Knotens

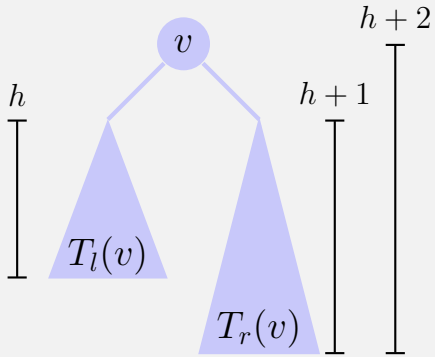
Die *Balance* eines Knotens v ist definiert als die Höhendifferenz seiner beiden Teilbäume $T_l(v)$ und $T_r(v)$

$$\mathrm{bal}(v) := h(T_r(v)) - h(T_l(v))$$

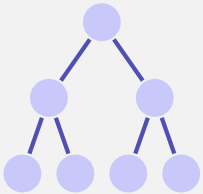


AVL Bedingung

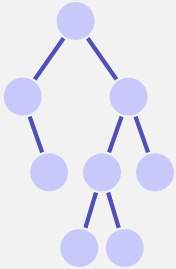
AVL Bedingung: für jeden Knoten v eines Baumes gilt $\mathrm{bal}(v) \in \{-1, 0, 1\}$



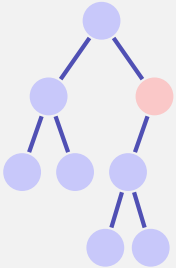
(Gegen-)Beispiele



AVL Baum der Höhe 2



AVL Baum der Höhe 3



Kein AVL Baum

Einfügen

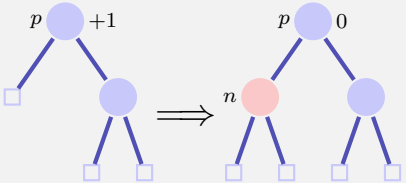
Balancieren

- Speichern der Balance für jeden Knoten
- Baum rebalancieren bei jeder Update-Operation

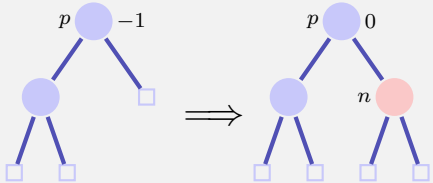
Neuer Knoten n wird eingefügt:

- Zuerst einfügen wie bei Suchbaum.
- Prüfe die Balance-Bedingung für alle Knoten aufsteigend von n zur Wurzel.

Balance am Einfügeort



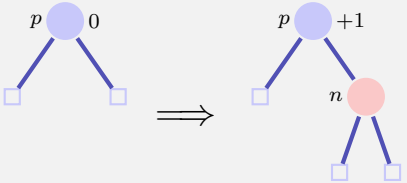
Fall 1: $\mathrm{bal}(p) = +1$



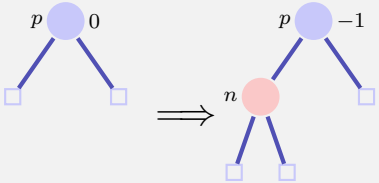
Fall 2: $\mathrm{bal}(p) = -1$

Fertig in beiden Fällen, denn der Teilbaum ist nicht gewachsen.

Balance am Einfügeort



Fall 3.1: $\mathrm{bal}(p) = 0$ rechts



Fall 3.2: $\mathrm{bal}(p) = 0$, links

In beiden Fällen noch nicht fertig. Aufruf von `upin(p)`.

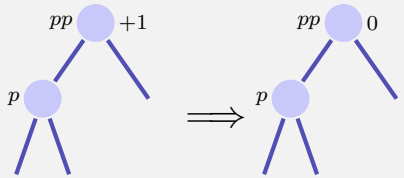
upin(p) - Invariante

Beim Aufruf von `upin(p)` gilt, dass

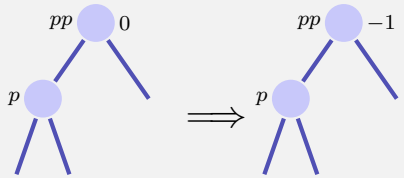
- der Teilbaum ab p gewachsen ist und
- $\mathrm{bal}(p) \in \{-1, +1\}$

upin(p)

Annahme: p ist linker Sohn von pp[1]



Fall 1: $\mathrm{bal}(pp) = +1$, fertig.



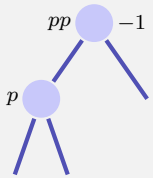
Fall 2: $\mathrm{bal}(pp) = 0$, `upin(pp)`

In beiden Fällen gilt nach der Operation die AVL-Bedingung für den Teilbaum ab pp

[1] Ist p rechter Sohn: symmetrische Fälle unter Vertauschung von $+1$ und -1

upin(p)

Annahme: p ist linker Sohn von pp



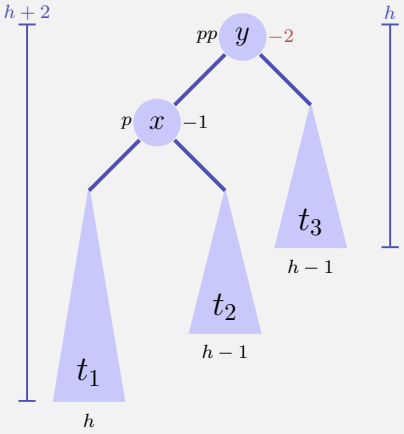
Fall 3: $\mathrm{bal}(pp) = -1$,

Dieser Fall ist problematisch: das Hinzufügen von n im Teilbaum ab pp hat die AVL-Bedingung verletzt. Rebalancieren!

Zwei Fälle $\mathrm{bal}(p) = -1$, $\mathrm{bal}(p) = +1$

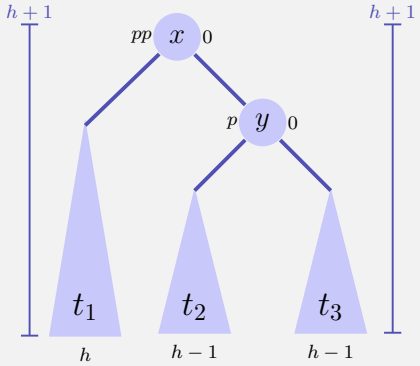
Rotationen

Fall 1.1 $\mathrm{bal}(p) = -1$. [2]



$\Longrightarrow$

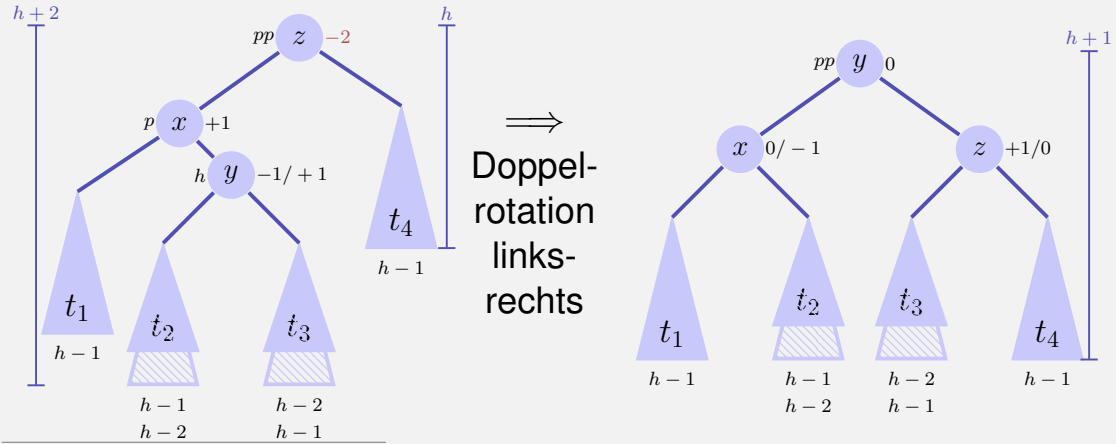
Rotation nach rechts



[2] p rechter Sohn $\Rightarrow \mathrm{bal}(pp) = \mathrm{bal}(p) = +1$, Linksrotation

Rotationen

Fall 1.2 $\mathrm{bal}(p) = +1$. [3]



[3] p rechter Sohn $\Rightarrow \mathrm{bal}(pp) = +1, \mathrm{bal}(p) = -1$, Doppelrotation rechts links

Baum Augmentieren

Aufgabe:
Augmentieren Sie die Knoten `n` eines Suchbaumes mit ihrer Höhe `n.height`. Stellen Sie sicher, dass die Höhe konsistent bleibt, auch wenn Knoten eingefügt werden.

[Starten Sie hier:
`https://expert.ethz.ch/print/ifbaug2/SS19/e07_examples`]

Fragen oder Anregungen?