ETH zürich

Übung 6

Datenstrukturen und Algorithmen, D-MATH, ETH Zurich

Programm von heute

Feedback letzte Übung

Wiederholung Theorie
- Binäre Bäume

Wiederholung Theorie
- AVL Bedingung
- AVL Einfügen

Code-Beispiel

Nachbesprechung

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q$

Nachbesprechung

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$

Nachbesprechung

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j, k) = k^j \bmod p$

Nachbesprechung

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ nicht passend: $(k = 0) \mapsto 0, (k = 1) \mapsto 1$

Nachbesprechung

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ nicht passend: $(k = 0) \mapsto 0, (k = 1) \mapsto 1$
- $s(j,k) = ((k \cdot j) \bmod q) + 1$

Nachbesprechung

Open hashing:

- $h'(k) = \lceil \ln(k+1) \rceil \bmod q \rightarrow$ nicht passend: $(k = 0) \mapsto 0$
- $s(j,k) = k^j \bmod p \rightarrow$ nicht passend: $(k = 0) \mapsto 0, (k = 1) \mapsto 1$
- $s(j,k) = ((k \cdot j) \bmod q) + 1 \rightarrow$ nicht passend: 1 wenn k Vielfaches von q, und Bereich $p - q$ nicht abgedeckt.

Nachbesprechung

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 27, 2, 32

```
T_1: __, __, 27, __, __      T_2: __, __, __, __, __

T_1: __, __,  2, __, __      T_2: 27, __, __, __, __

T_1: __, __, 27, __, __      T_2:  2, 32, __, __, __
```

Nachbesprechung

Coocoo hashing

- $h_1(k) = k \bmod 5$, $h_2(k) = \lfloor k/5 \rfloor \bmod 5$
- Hinzufügen von 7: Endlosschleife

```
    T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7: T_1: __, __,  7, __, __    T_2: 27, 32, __, __, __
 2: T_1: __, __,  2, __, __    T_2: 27,  7, __, __, __
32: T_1: __, __, 32, __, __    T_2:  2,  7, __, __, __
27: T_1: __, __, 27, __, __    T_2:  2, 32, __, __, __
 7: ...
```

Nachbesprechung

Finden eines Sub-Arrays

```
// calculating hash_a, hash_b, c_to_k
It1 window_end = from;
for(It2 current = begin; current != end;
    ++current, ++window_end) {
  if(window_end == to) return to;
  hash_b = (C * hash_b % M + *current) % M;
  hash_a = (C * hash_a % M + *window_end) % M;
  c_to_k = c_to_k * C % M;
}
```

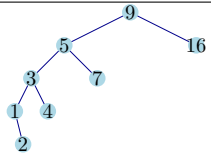
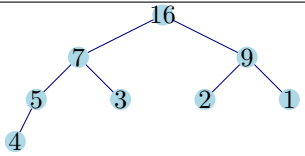
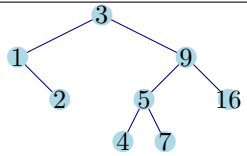
Nachbesprechung

Finden eines Sub-Arrays

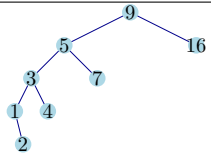
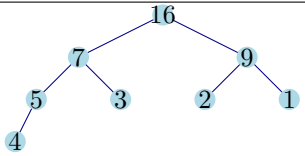
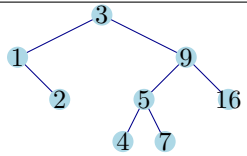
```
// looking for b and updating hash_a
for(It1 window_begin = from; ;
    ++window_begin, ++window_end) {
  if(hash_a == hash_b)
    if(std::equal(window_begin, window_end, begin, end))
      return window_begin;
  if(window_end == to) return to;
  hash_a = (C * hash_a % M + *window_end
          + (M - c_to_k) * *window_begin % M) % M;
}
```

2. Wiederholung Theorie

Vergleich binärer Bäume

	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Einfügen	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Suchen	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Löschen	$\Theta(h(T))$	Suchen + $\Theta(\log n)$	$\Theta(\log n)$

Vergleich binärer Bäume

	Suchbäume	**Heaps** Min- / Max- Heap	**Balancierte Bäume** AVL, red-black tree
in C++:		`std::make_heap`	`std::map`
			
Einfügen	$\Theta(h(T))$	$\Theta(\log n)$	$\Theta(\log n)$
Suchen	$\Theta(h(T))$	$\Theta(n)$ (!!)	$\Theta(\log n)$
Löschen	$\Theta(h(T))$	Suchen + $\Theta(\log n)$	$\Theta(\log n)$

Bemerke: $\Theta(\log n) \leq \Theta(h(T)) \leq \Theta(n)$

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

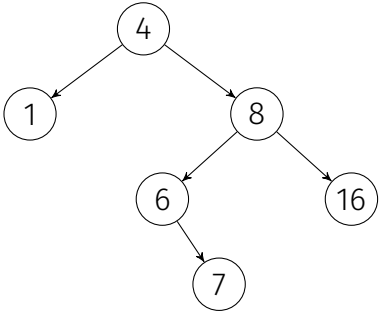
- Zuhinterst im Array einfügen.
- Heap-Bedingung wiederherstellen: `siftUp` (Aufsteigen lassen).

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.



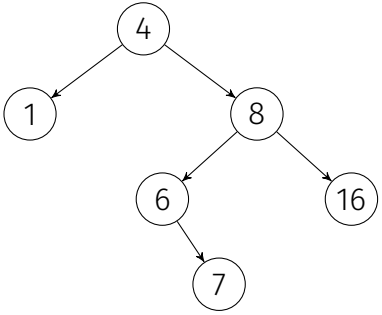
MinHeap

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

Wiederholung: Binäre Bäume, Schlüssel Einfügen

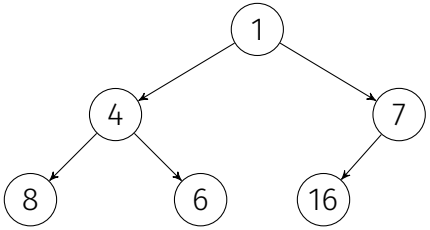
Binäre Suchbäume

- 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

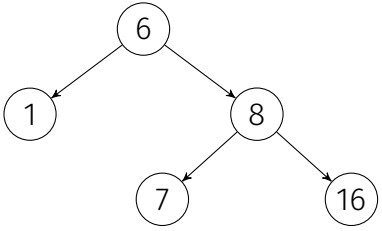
- Schlüssel durch hinterstes Arrayelement ersetzen.
- Heap-Bedingung wiederherstellen: `siftDown` *or* `siftUp`.

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

Wiederholung: Binäre Bäume, Schlüssel Löschen

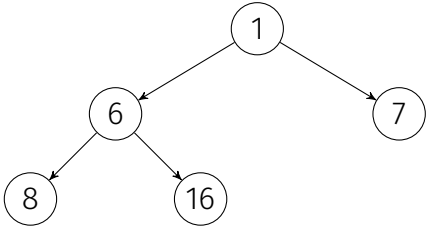
Binäre Suchbäume

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



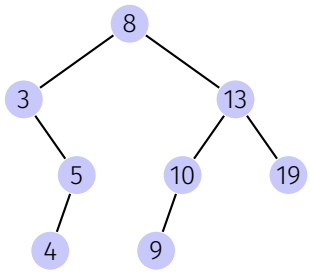
MinHeap

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



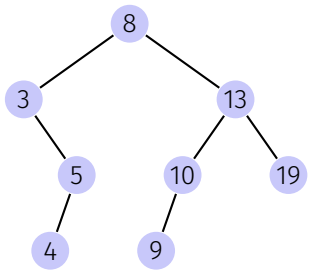
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



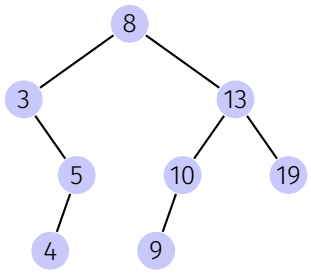
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



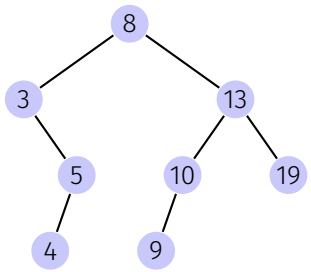
Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.



Traversierungsarten

- Hauptreihenfolge (preorder): v, dann $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$.
  8, 3, 5, 4, 13, 10, 9, 19
- Nebenreihenfolge (postorder): $T_{\text{left}}(v)$, dann $T_{\text{right}}(v)$, dann v.
  4, 5, 3, 9, 10, 19, 13, 8
- Symmetrische Reihenfolge (inorder): $T_{\text{left}}(v)$, dann v, dann $T_{\text{right}}(v)$.
  3, 4, 5, 8, 9, 10, 13, 19



Quiz

Zeichnen Sie jeweils einen binären Suchbaum, der die folgenden Traversierungen erzeugt. Ist der Baum eindeutig?

Symmetrische Reihenfolge (inorder)	1 2 3 4 5 6 7 8
Hauptreihenfolge (preorder)	4 3 1 2 8 6 5 7
Nebenreihenfolge (postorder)	1 3 2 5 6 8 7 4

Geben Sie zu jeder Reihenfolge eine Zahlensequenz aus $\{1, \ldots, 4\}$, die nicht aus einem gültigen binären Suchbaum stammen kann.

Antworten

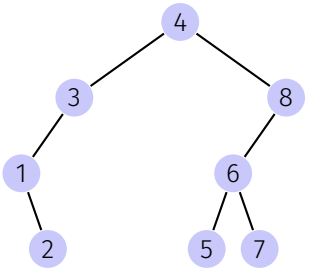
Symmetrische Reihenfolge: jeder Suchbaum mit den Zahlen $\{1, \ldots, 8\}$ ist gültig.

Der Baum ist nicht eindeutig

Es gibt keinen Suchbaum, welcher nicht die aufsteigend sortierte Sequenz ausgeben würde. Gegenbeispiel 1 2 4 3

Antworten

Hauptreihenfolge (preorder) 4 3 1 2 8 6 5 7

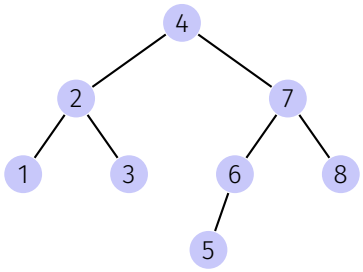


Der Baum ist eindeutig

Es muss rekursiv gelten, dass zuerst eine Gruppe Zahlen grösser und danach kleiner als der erste Wert kommen. Gegenbeispiel: 3 1 4 2

Antworten

Nebenreihenfolge (postorder) 1 3 2 5 6 8 7 4

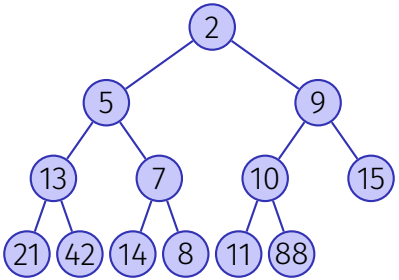


Der Baum ist eindeutig

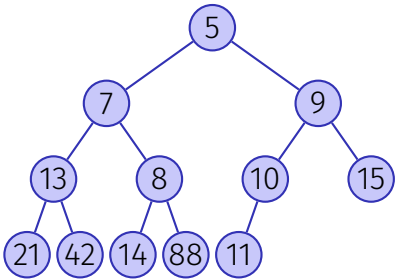
Konstruktion hier: `https://www.techiedelight.com/build-binary-search-tree-from-postorder-sequence/`, Ähnliches Argument wie vorher, nur von hinten nach vorne. Gegenbeispiel 4 2 1 3

Heap

Führen Sie auf folgendem Min-Heap eine Extract-Min Operation aus, wie in der Vorlesung vorgestellt, einschliesslich der Wiederherstellung der Heap-Bedingung. Wie sieht der Heap nach der Operation aus?

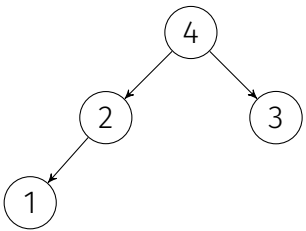


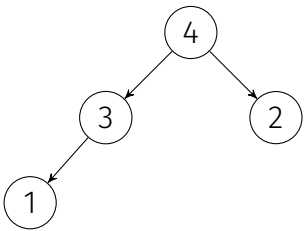
Lösung

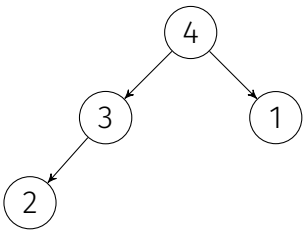


Quiz: Anzahl MaxHeaps mit n Schlüsseln

Sei $N(n)$ die Anzahl verschiedener MaxHeaps, welche aus allen Schlüsseln $1, 2, \ldots, n$ gebildet werden können. Beispielsweise ist $N(1) = 1,\ N(2) = 1,\ N(3) = 2,\ N(4) = 3$ und $N(5) = 8$.
Finde die Werte $N(6)$ und $N(7)$.

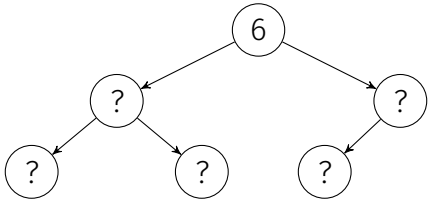






Anzahl MaxHeaps mit n verschiedenen Schlüsseln

Ein die Elemente $1, 2, 3, 4, 5, 6$ enthaltender MaxHeap sieht so aus:



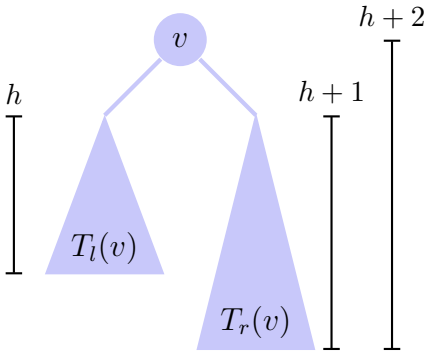
Möglichkeiten, Elemente des linken Teilbaums zu wählen: $\binom{5}{3}$.

$$\Rightarrow N(6) = \binom{5}{3} \cdot N(3) \cdot N(2) = 10 \cdot 2 \cdot 1 = 20.$$

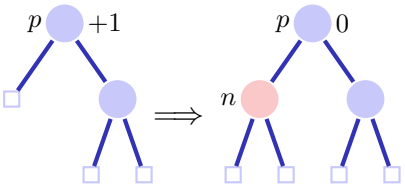
$$\text{und } N(7) = \binom{6}{3} \cdot N(3) \cdot N(3) = 20 \cdot 2 \cdot 2 = 80.$$

AVL Bedingung

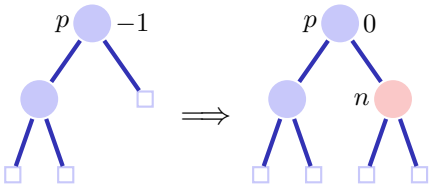
AVL Bedingung: für jeden Knoten v eines Baumes gilt $\mathrm{bal}(v) \in \{-1, 0, 1\}$



Balance am Einfügeort



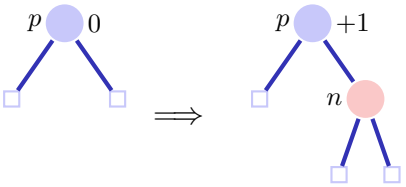
Fall 1: $\mathrm{bal}(p) = +1$



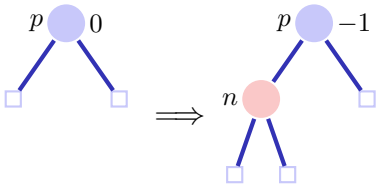
Fall 2: $\mathrm{bal}(p) = -1$

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)`**.

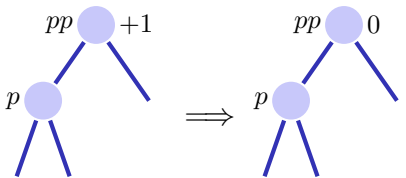
upin(p) - Invariante

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

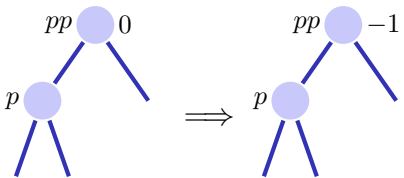
- der Teilbaum ab p gewachsen ist und
- $\mathrm{bal}(p) \in \{-1, +1\}$

upin(p)

Annahme: p ist linker Sohn von pp[1]



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



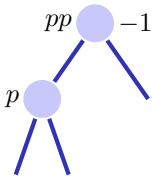
Fall 2: $\mathrm{bal}(pp) = 0$, **`upin(pp)`**

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



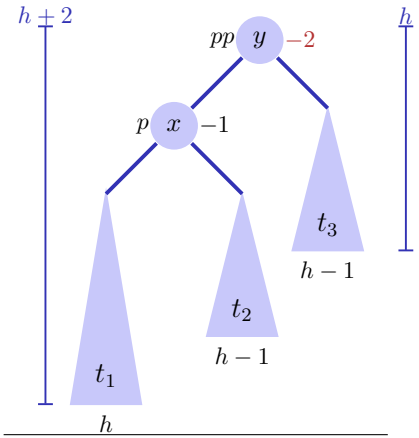
Fall 3: $\mathrm{bal}(pp) = -1$,

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

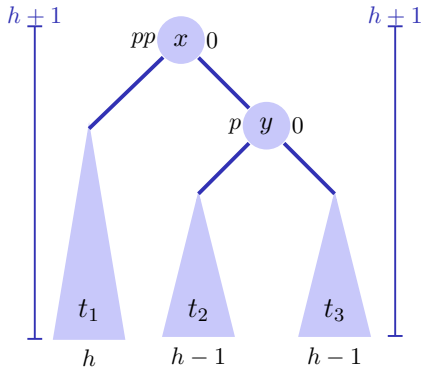
Zwei Fälle $\mathrm{bal}(p) = -1$, $\mathrm{bal}(p) = +1$

Rotationen

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



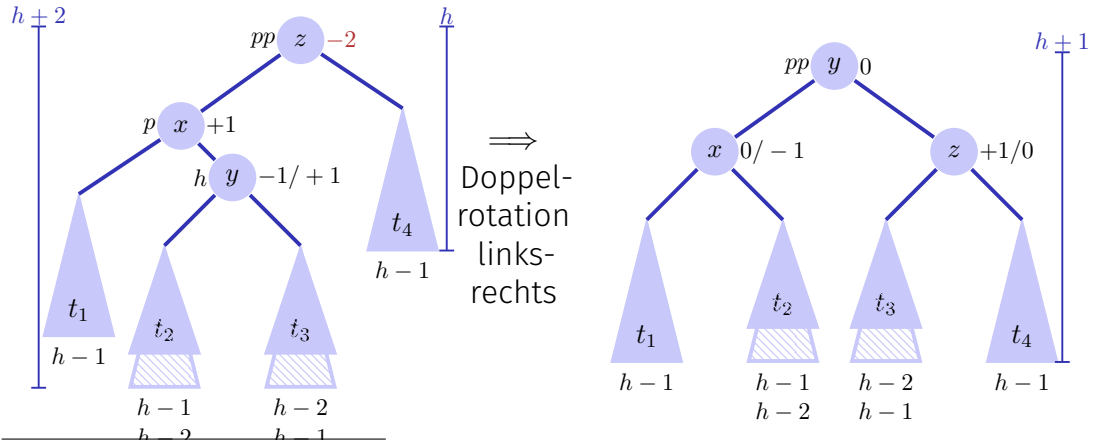
$\Longrightarrow$ Rotation nach rechts



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

Rotationen

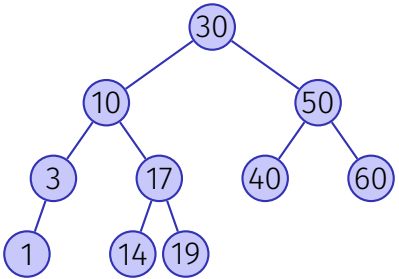
Fall 1.2 $\mathrm{bal}(p) = +1$. [3]



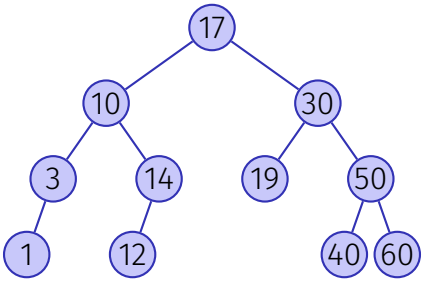
[3] p rechter Sohn $\Rightarrow \mathrm{bal}(pp) = +1$, $\mathrm{bal}(p) = -1$, Doppelrotation rechts links

Quiz

Fügen Sie in folgendem AVL Baum den Schlüssel 12 ein und rebalancieren Sie (wie in der Vorlesung gezeigt). Wie sieht der AVL Baum nach der in der Vorlesung gezeigten Operation aus ?



Lösung



Code-Beispiel

Exercise class 06: Binary Trees auf Code-Expert

- Binary Tree: Einfache Aufgaben
- Augmenting a Binary Search Tree: Vorbereitung für AVL-Bäume

Fragen ?