

ALGORITHMEN UND DATENSTRUKTUREN

ÜBUNG 9: AVL-BÄUME & TOPOLOGISCHES SORTIEREN

Eric Kunze

`eric.kunze@tu-dresden.de`

TU Dresden, 8. Dezember 2021

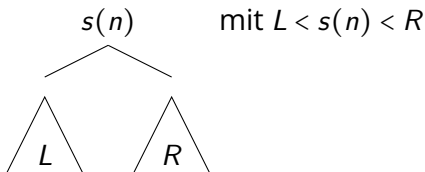
letzte Änderung:
08.12.2021, 12:07

AVL-Bäume

AVL-BÄUME

Wir betrachten einen Baum t und bezeichnen die *Schlüssel* an den Knoten n mit $s(n)$.

Suchbaum:



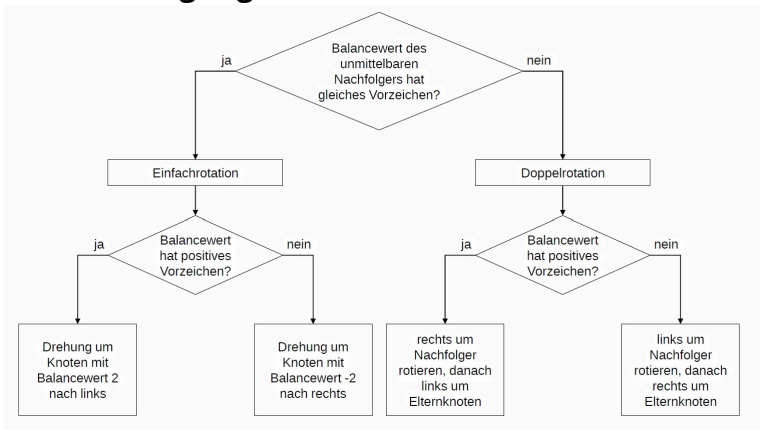
Die **Höhe** des Baumes bezeichnen wir mit $h(t)$. Wir ordnen jedem Knoten n einen **Balancefaktor** $b(n)$ zu:

$$b(n) := h(R) - h(L)$$

AVL-Baum: Suchbaum mit $b(n) \in \{-1, 0, 1\}$

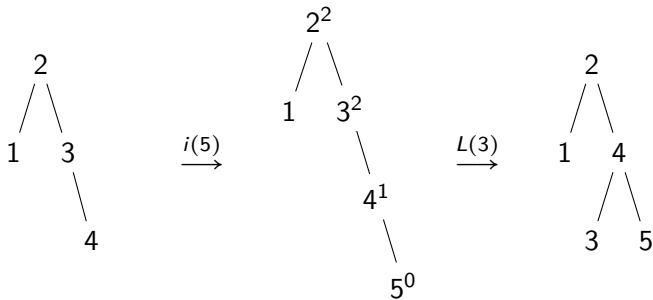
BALANCIEREN

- ▶ Einfügen eines neuen Schlüssels s
- ▶ Berechne Balancefaktoren auf dem Pfad von s zur Wurzel bis zum ersten Auftreten von ± 2
- ▶ **Balancierungsalgorithmus:**



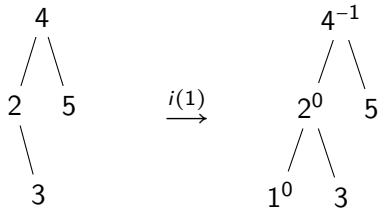
AUFGABE 1

Baum 1:



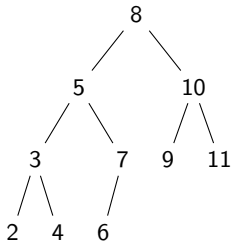
AUFGABE 1

Baum 2:

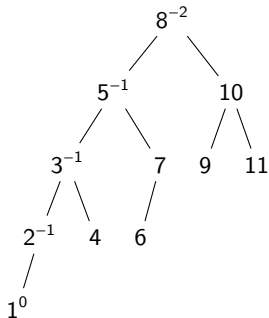


AUFGABE 1

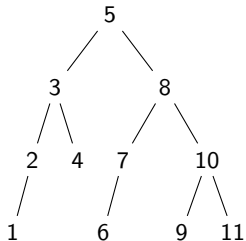
Baum 3:



$i(1)$
 \rightarrow

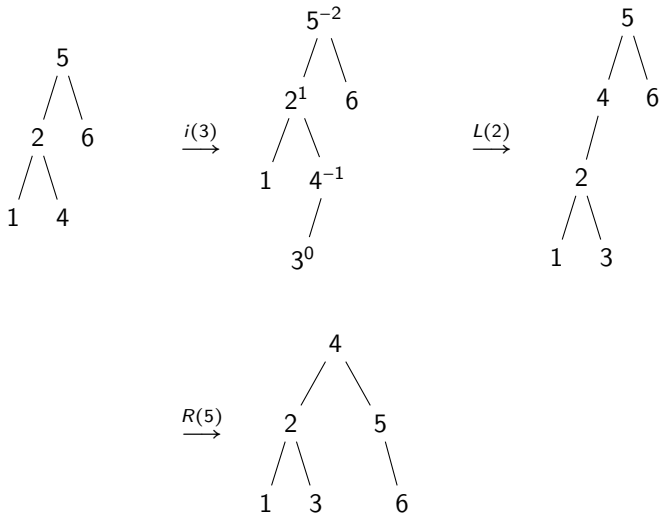


$R(8)$
 \rightarrow



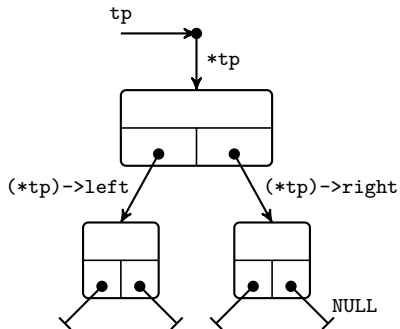
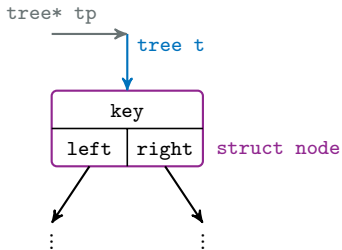
AUFGABE 1

Baum 4:



ERINNERUNG: BÄUME

```
1 typedef struct node *tree;  
2 struct node { int key; tree left, right; };
```



- ▶ Verarbeitung eines Baumes: Einfachreferenzen (tree t)
- ▶ Veränderung eines Baumes: Doppelreferenzen (tree *tp)

AUFGABE 2 — TEIL (A)

neuen Baum mit Balancefaktoren bestimmen

```
1 int height(tree t) {  
2     if (t == NULL) return 0;  
3  
4     int hl = height(t->left);  
5     int hr = height(t->right);  
6     return (hl > hr) ? hl + 1 : hr + 1;  
7 }
```

```
11 tree bfs(tree t) {  
12     if (t == NULL) return NULL;  
13  
14     tree bf = malloc(sizeof(struct node));  
15     bf->key = height(t->right) - height(t->left);  
16     bf->left = bfs(t->left);  
17     bf->right = bfs(t->right);  
18     return bf;  
19 }
```

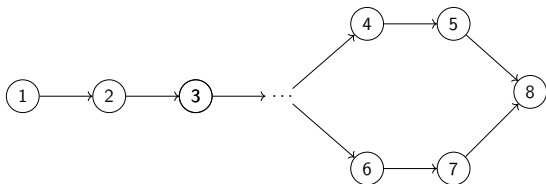
Linksrotation um die Wurzel

```
1 void lRot(tree *tp) {  
2     if (tp == NULL || *tp == NULL)  
3         return;  
4  
5     tree rightChild = (*tp)->right;  
6     (*tp)->right = rightChild->left;  
7     rightChild->left = *tp;  
8     *tp = rightChild;  
9 }
```

Topologisches Sortieren

TOPOLOGISCHES SORTIEREN

- ▶ Sortierung von *Beziehungen* zwischen Objekten
- ▶ **Bsp.:** Ablauf eines Bauvorhabens
 - ▷ Baugrube ausheben (1) vor Fundament gießen (2)
 - ▷ Fundament gießen (2) vor Wände setzen (3)
 - ▷ ...
 - ▷ Elektrik im Bad (4) vor Fliesen (5)
 - ▷ Wohnzimmer tapezieren (6) vor streichen (7)
 - ▷ Fliesen (5) und Wände streichen (7) vor Möbel aufstellen (8)



- ▶ In welcher Reihenfolge kann ich die Tätigkeiten abarbeiten?

TOPOLOGISCHES SORTIEREN

Gegeben sei ein gerichteter, azyklischer Graph $G = (V, E)$. Eine **topologische Sortierung** von G ist eine *bijektive* Abbildung $\text{ord}: V \rightarrow \{1, \dots, |V|\}$, sodass für alle $v, v' \in V$ mit $(v, v') \in E$ die Relation $\text{ord}(v) < \text{ord}(v')$ gilt.

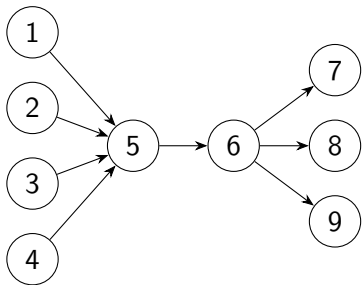
Algorithmus:

while (Elemente übrig)

- {
- ▶ wähle Element v ohne Vorgänger
- ▶ dekrementiere Anzahl der Vorgänger in den Nachfolgern von v
- ▶ füge v der Ausgabeliste hinzu
- ▶ lösche v aus G
- }

AUFGABE 3

Teil (a)



Es gibt $4! \cdot 1! \cdot 3! = 24 \cdot 1 \cdot 6 = 144$ viele topologische Sortierungen.

AUFGABE 3

Teil (b)

Es werden alle Möglichkeiten mit der 1 am Anfang gestrichen, d.h. im ersten Block gibt es nur noch $4! - 3! = 24 - 6$ viele Möglichkeiten — insgesamt also

$$(4! - 3!) \cdot 1! \cdot 3! = 18 \cdot 1 \cdot 6 = 108.$$

Teil (c)

