Algorithmen und Datenstrukturen SoSe25

-Assignment 3-

Moritz Ruge
Matrikelnummer: 5600961
Lennard Wittenberg
Matrikelnummer: —

Mai 2025

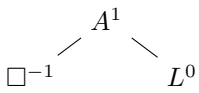
Problem 1: AVL-Bäume

a) Fügen Sie die Schlüssel A, L, G, O, D, T, S, X, Y, Z in dieser Reihenfolge in einen anfangs leeren AVL-Baum ein. Löschen Sie sodann die Schlüssel Z, A, L. Zeichnen Sie den Baum nach jedem Einfüge- und Löschvorgang, und zeigen Sie die Rotationen, welche durchgeführt werden. Annotieren Sie dabei auch die Knoten mit ihrer jeweiligen Höhe.

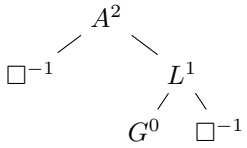
⇒ Bei den Knoten die hochgestellte Zahl ist die Höhe des jeweiligen Knotens.

1. Einfügen: A

2. Einfügen: L

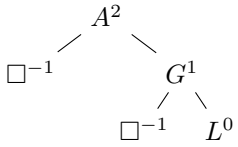


3. Einfügen: G

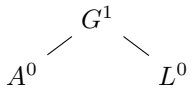


   - BF-Faktor bei Knoten A ist größer als 1 ⇒ Um-balancieren der Knoten A, L, G

   ⇒ Rechts-Rotation der Knoten $L\&G$

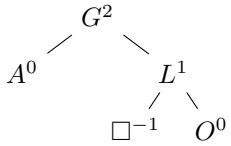


   ⇒ Links-Rotation der Knoten $A\&G$

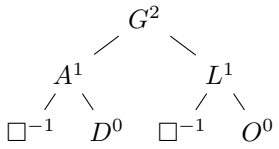


   ⇒ AVL-Baum ist ausgeglichen

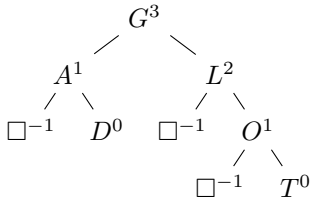
4. Einfügen: O



5. Einfügen: D

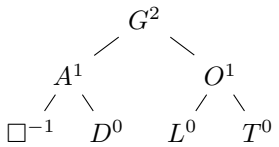


6. Einfügen: T



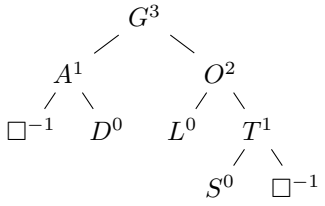
   - BF-Faktor bei Knoten L ist größer als 1 ⇒ Um-balancieren der Knoten L, O, T

   ⇒ Rechts-Rotation der Knoten $L\&O$

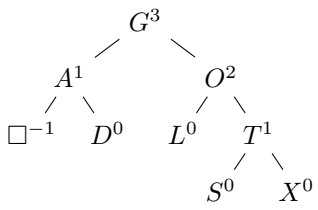


   ⇒ AVL-Baum ist ausgeglichen

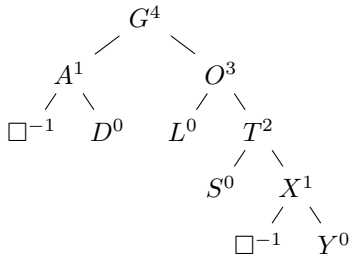
7. Einfügen: S



8. Einfügen: X

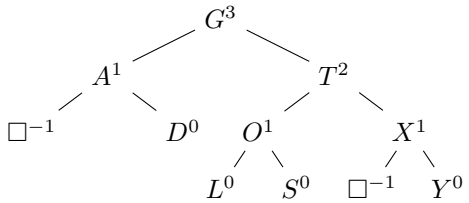


9. Einfügen: Y



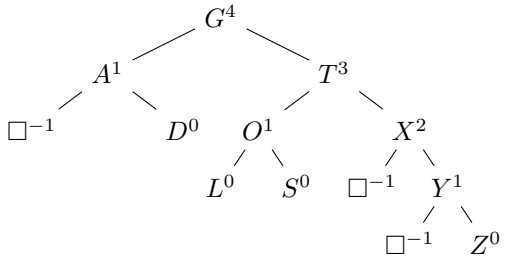
- BF-Faktor bei Knoten O ist größer als $1 \Rightarrow$ Um-balancieren der Knoten $O\&T$

$\Rightarrow$ Links-Rotation der Knoten $O\&T$



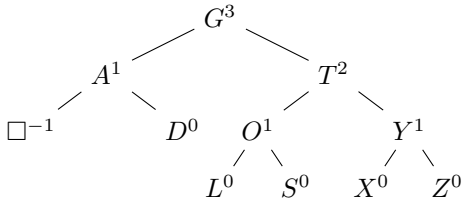
$\Rightarrow$ AVL-Baum ist ausgeglichen

10. Einfügen: Z



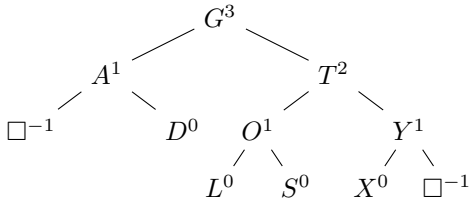
- BF-Faktor bei Knoten X ist größer als $1 \Rightarrow$ Um-balancieren der Knoten $Y\&Z$

$\Rightarrow$ Links-Rotation der Knoten $X\&Y$

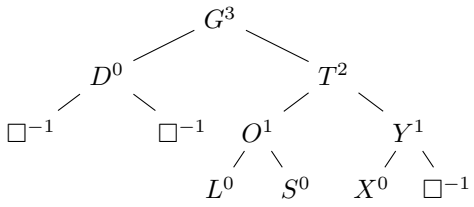


$\Rightarrow$ AVL-Baum ist ausgeglichen

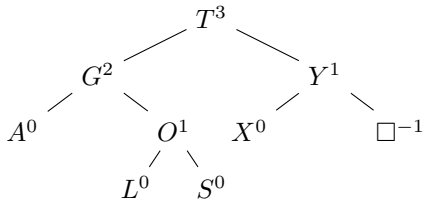
11. Lösche: Z



12. Lösche: A

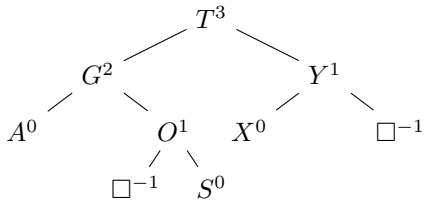


- BF-Faktor bei Knoten G ist größer als $1 \Rightarrow$ Um-balancieren der Knoten $G\&T$



$\Rightarrow$ AVL-Baum ist ausgeglichen

13. Lösche: L



b) Beweisen Sie: Beim Einfügen in einen AVL-Baum wird höchstens eine (Einfach- oder Doppel-)Rotation ausgeführt. Gilt das auch beim Löschen (Begründung)?

Annahme: Beim Einfügen in einen AVL-Baum wird höchstens eine (Einfach- oder Doppel-)Rotation ausgeführt.

- In einem AVL-Baum gelten die Eigenschaften eines BTS.
- In einem AVL-Baum muss gelten: Die Differenz zwischen der Höhe des Linken Teilbaum und der Höhe des Rechten Teilbaum darf maximal 1 sein.

base-case AVL-Baum mit einem Knoten:
r $\rightarrow$ h: 0

insertion (n):

```
‘‘‘
       r  h: 0
     /   \
    n     nil  hl: 1, hr: 0
‘‘‘
```

$|hl - hr| <= 1$ true $\rightarrow$ keine Rotation nötig!
Annahme gilt im base-case

I.S: Wenn die Annahme bei n Insertions gilt, gilt sie auch n+1 Insertions. Wenn nach n insertions ein AVL-Baum die AVL-Eigenschaften weiterhin erfüllt kann es zu mehreren Fallen bei einer $n + 1$ insertion kommen:
Fallunterscheidung:

1. Nach einer Insertion wird kein Knoten unblanciert. $\rightarrow$ keine Rotation notwendig
2. Nach einer Insertion wird ein Knoten unblanciert, da einer der Teilbäume tiefer ist als der andere und $|hl-hr| <= 1$ nicht mehr gilt. Da die Unbalance nur nach einer einzelnen Insertion auftreten kann, ist der Höhen Unterschied immer 2.

2.1. der Linke Teilbaum eines Knotens ist Tiefer als der Rechte Teilbaum.

```
‘‘‘ Beispielabschnitt für links Unbalance irgendwo in einem AVL-Baum
       n
     /   \
    n1    n1
   /
  n2
 /
n3
‘‘‘
```

Am tiefsten unbalanciertem Knoten (pivot) wird eine rechts-Rotation ausgeführt.

- Die Rotation wird am tieften unbalanciertem Knoten ausgeführt, weil nach einer Rotation die Höhe des betroffenen Teilbaums wieder den selben Wert hat, wie vor der n+1 Insertion.

- Der Elternknoten des pivot-Knotens wird zum neuen Rechten Kind des pivot-Knotens.
- Das alte Rechte Kind des pivot-Knotens wird zum neuen linken Kind des neuen Rechten Kindknotens.

```
‘‘‘
      n
    /   \
   n1    n1
  /  \
 n2   n2
‘‘‘
```

2.2 Der Rechte Teilbaum eines Knotens ist Tiefer als der Linke Teilbaum. Zeichnungen sind analog. Am tiefsten unbalanciertem Knoten (pivot) wird eine links-Rotation ausgeführt.

- Die Rotation wird am tieften unbalanciertem Knoten ausgeführt, weil nach einer Rotation die Höhe des betroffenen Teilbaums wieder den selben Wert hat, wie vor der $n+1$ Insertion.
- Der Elternknoten des pivot-Knotens wird zum neuen Linken Kind des pivot-Knotens.
- Das alte Linke Kind des pivot-Knotens wird zum neuen Rechten Kind des neuen Linken Kindknotens.

2.3 Der Rechte-Linke Teilbaum eines Knotens ist Tiefer als der Rechte Teilbaum. Es wird erst eine links-Rotation am tieften unbalanciertem Knoten (als Pivot Elternknoten) durchgeführt und dann eine weitere rechts-Rotation mit dem selben pivot-Knoten durchgeführt.

- Die Rotationen selbst funktionieren genau so wie in 2.1 und 2.2 beschrieben und sequentiell am selben pivot-Knoten.
- Auch bei Doppel-Rotationen gilt: Nach der Rotation sind die Höhenverhältnisse der Teilbaume wieder so wie vor der Insertion $n + 1$

```
‘‘‘ Beispielabschnitt für rechts-links Unbalance irgendwo in einem AVL-Baum
        n
      /   \
    n1     n1
      \
      n2
     /
    n3
‘‘‘
nach links-Rotation
‘‘‘
        n
      /   \
    n1     n1
      \
      n2
        \
         n3
‘‘‘
```

nach rechts-Rotation
```
        n
      /   \
    n1     n1
   /  \
  n2  n2
```

2.4 Der Linke-Rechte Teilbaum eines Knotens ist Tiefer als der Linke Teilbaum. Zeichnungen sind analog. Es wird erst eine rechtss-Rotation am tieften unbalanciertem Knoten (als Pivot Elternknoten) durchgeführt und dann eine weitere links-Rotation mit dem selben pivot-Knoten durchgeführt.

- Die Rotationen selbst funktionieren genau so wie in 2.1 und 2.2 beschrieben und sequentiell am selben pivot-Knoten.
- Auch bei Doppel-Rotationen gilt: Nach der Rotation sind die Höhenverhältnisse der Teilbaume wieder so wie vor der Insertion $n + 1$

Man sieht man benötigt maximal eine (Einfach- oder Doppel-) Rotation um nach einer Insertion die AVL-Eigenschaft zu erhalten.
Annahme gilt für $n + 1$

□

Löschen:

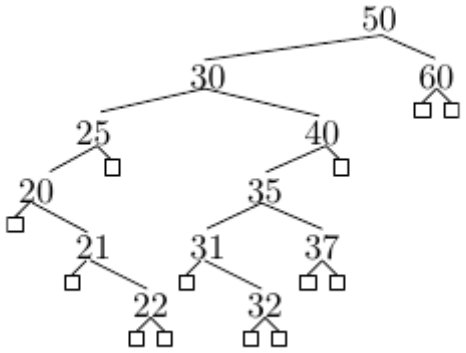
- Angenommen man hat einen beliebigen AVL-Baum.
- Beim Löschen eines beliebigen Knotens kann es sein, dass eine komplexere Unblance eintritt, diese ist nicht durch einmaliges rotieren lösbar.

bsp:
```
          20
        /    \
      15      25
     /  \    /  \
    10  18  24  30
       /  \
      17  19
```

wenn man 15 entfernen würde, wäre die entstehende Unbalance nicht mit einer einmaligen Rotation lösbar. Daher gilt die Annahme meiner Meinung nach nur für das Einfügen (Insertion)

Problem 2: findRange

a) Betrachten Sie den folgenden binären Suchbaum: Wo befinden sich die Schlüssel, die kleiner sind als 37? Wo befinden sich die Schlüssel, die größer sind als 21? Wo befinden sich die Schlüssel, die zwischen 21 und 37 liegen?



b) Beschreiben Sie, wie man in einem AVL-Baum mit n Schlüsseln die Operation findRange(k1 , k2) implementieren kann, die alle Schlüssel k liefert, für die $k1 \leq k \leq k2$ ist. Die Laufzeit soll $O(logn + s)$ betragen. Dabei ist s die Anzahl der gelieferten Schlüssel.

Problem 3: Rot-Schwarz Bäume

Ein rot-schwarz Baum ist ein binärer Suchbaum, den wir auf die folgende Weise erweitern: Jeder Knoten und jeder leere Teilbaum erhält eine Farbe (rot oder schwarz), so dass die folgenden Regeln gelten: (i) die Wurzel ist schwarz; (ii) die leeren Teilbäume sind schwarz; (iii) die Kinder eines roten Knoten sind schwarz; und (iv) die schwarze Tiefe aller leeren Teilbäume ist gleich, d.h., für alle leeren Teilbäume ist die Anzahl der schwarzen Knoten auf dem Pfad von der Wurzel zum jeweiligen Teilbaum gleich.

a) Zeichnen Sie drei Beispiele für rot-schwarz Bäume und erklären Sie, warum diese jeweils die Regeln für einen rot-schwarz Baum erfüllen.
b) Sei T ein rot-schwarz Baum, und sei s die schwarze Tiefe der leeren Teilbäume. Zeigen Sie, dass T mindestens $2s - 1 - 1$ schwarze Knoten besitzt. Was folgt daraus über die Mindestanzahl von Knoten in einem rot-schwarz Baum mit Höhe h? Folgern Sie: ein rot-schwarz Baum mit n Knoten hat Höhe $O(logn)$.