Alexandros Paraskevaidis
Elias Gestrich

5. Übungsblatt

Aufgabe 1 - Der Nächste bitte

a) Um den nächst größeren Schlüssel zu einem Schlüssel x zu finden, muss man den kleinsten Schlüssel finde, der größer ist als x. Da wir einen Binärbaum haben, geht dies, indem wir x in der Laufzeit $\mathcal{O}(\log n)$ finden. Wenn x ein rechtes Kind hat, dann schaue nach dem linkesten Kind des rechten Kindes. Sonst gehe zum Elternknoten und schaue ob dieser größer ist als x, wenn er größer ist, dann ist dieser Schlüssel der nächstgrößere. Wenn wir an der Wurzel angekommen sind, dann ist x der größte Schlüssel.

Algorithmus 1: naechstgroesseren_schluessel_von(Schluessel x)

```
1  if x.right exists then
2      y ← x.right
3      for y.left exists do
4          y ← y.left
5      return y
6  y ← x for y ≠ root do
7      if y < y.parent then
8          return y.parent
9      y ← y.parent
10 return x
```

b) Die Knoten w im Baum für die v auf dem Pfad von der Wurzel zu w liegt, sind genau die Knoten, die im Teilbaum von v liegen. Um also die Anzahl solcher Knoten w zu berechnen müssen wir schauen wie viele Knoten im Teilbaum von v sind. Um zu jedem Knoten also die Anzahl an Knoten im Teilbaum zu berechnen, reicht es zu wissen wie viele Knoten im Teilbaum von den Kindern liegt und diese zusammenrechnen und die Anzahl an Kinder dazu addieren. Gehe dafür wie folgt vor: von Ebene $\lceil \log n \rceil$ (maximale Höhe eines balancierten Binärsuchbaums) bis Ebene 0 tue: Für alle x in der Ebene setze die Anzahl an Knoten im Teilbaum auf 0. Wenn das linke Kind existiert addiere zur Anzahl die Anzahl an Konten im Teilbaum des linken Kindes plus eins, wenn das rechte Kind existiert mache das selbe mit dem rechten Kind.

Algorithmus 2: groesse_teilbaeume(BST bst)

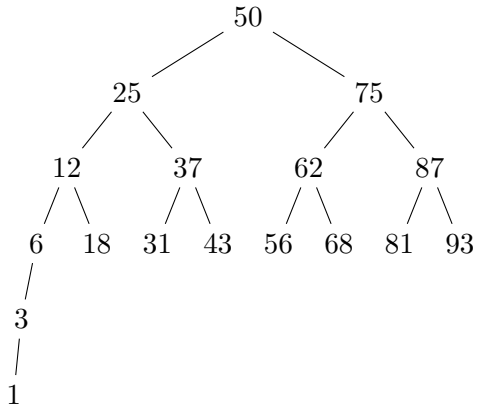
```
1 for depth y ← ⌈log n⌉ to depth 0 do
2     for node x with depth y do
3         x.result ← 0 if x.left then
4             x.result ← x.result + x.left.result + 1
5         if x.right then
6             x.result ← x.result + x.right.result + 1
```

Aufgabe 2 - Verallgemeinerten AVL-Baum

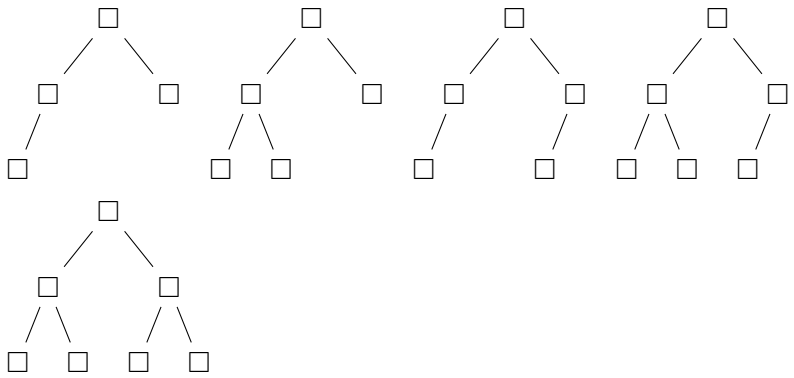
a) Für einen 0-balancierten AVL-Baum müssen alle Blätter auf der gleichen Höhe liegen, da sonst die Balanciertheit mindestens den Grad 1 hat. Ein Baum der Höhe 3 kann 1+2+4 = 7

Knoten besitzen, der achte Knoten zerstört dann aber die 0-Balanciertheit

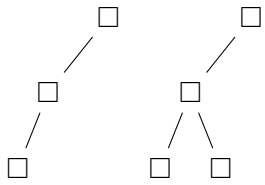
b)



c) $c = 1$



$c = 2$, die, die auch für $c = 1$ gehen plus:



Aufgabe 3 - Reverse Hashing

a)
- insert(8)
- insert(9)
- insert(26)
- insert(27)
- insert(1)
- insert(90)
- insert(14)
- insert(89)
- insert(23)

b) $a = 2$, $c = 4$

Aufgabe 4 - Offene Rechnung

a) Für insert(0), insert(15), insert(30), insert(45) würde 45 keinen neuen Platz finden, obwohl es noch einige freie Plätze gibt. Um dies zu vermeiden muss $\text{ggT}(h'(u), m) = 1$ gelten, damit man immer einen freien Platz findet, solange es einen gibt. Ein Beispiel für h' wäre zum beispiel $h'(u) = 7 + (u \mod 2)$

b) Ja ganz doof wäre zum Beispiel $H_i(u) = (h(u) + d_i) \mod m = (h(u) + i \cdot m) \mod m$, oder $H_i(u) = (h(u) + d_i) \mod m = (h(u) + i^2 \cdot m) \mod m$. Hier würde durch das sondiern nichts passieren.