technische universität dortmund

SoSe 2017

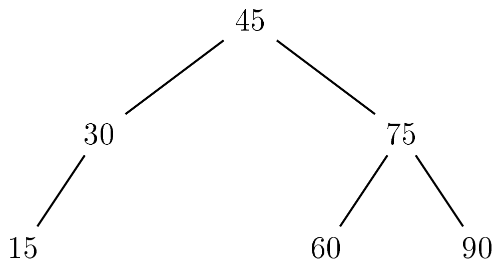
C. Sohler
A. Krivošija, A. Rey, H. Sandvoß

DAP2 – Präsenzübung 8

Besprechung: 14.06.2017 — 16.06.2017

Präsenzaufgabe 8.1: (AVL-Bäume)

Gegeben ist der folgende AVL-Baum.

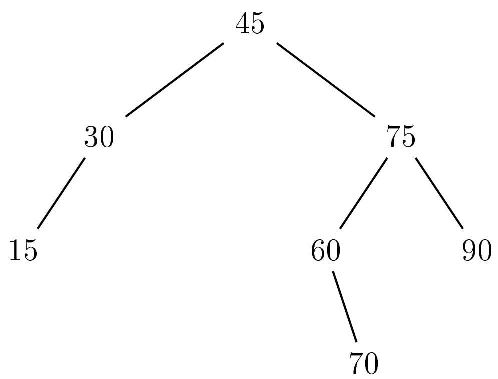


Fügen Sie nacheinander in dieser Reihenfolge die Schlüssel 70, 35 und 72 in den obigen AVL-Baum ein. Löschen Sie anschließend im resultierenden AVL-Baum die Schlüssel 90 und 70 in eben dieser Reihenfolge. Zeichnen Sie den resultierenden AVL-Baum nach jeder Operation und annotieren Sie, welche Knoten wie rotiert werden müssen, um den jeweils aktuellen AVL-Baum zu erhalten.

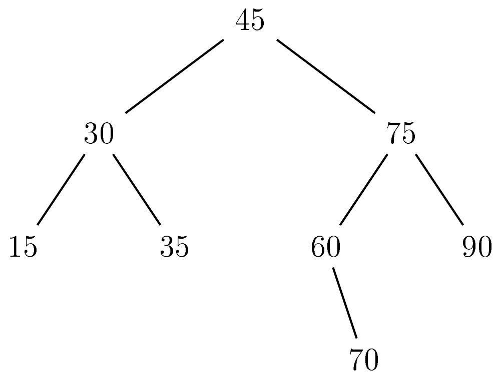
Erinnerung: Ein Knoten mit zwei Kindern im AVL-Baum wird durch seinen direkten Vorgänger ersetzt, wenn er gelöscht wird.

Lösung:

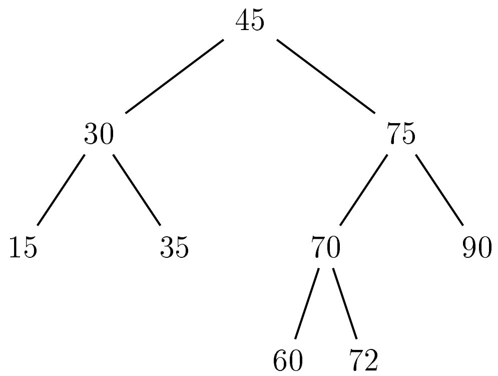
a) Einfügen von 70: Es ist keine Rotation erforderlich.



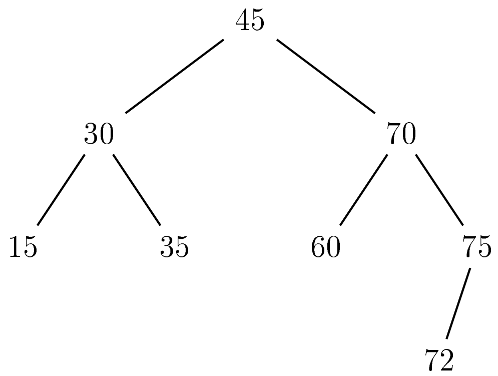
b) Einfügen von 35: Es ist keine Rotation erforderlich.



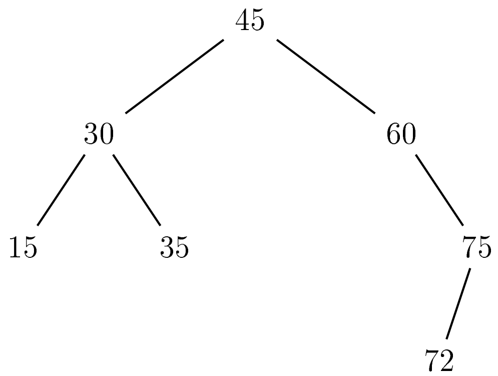
c) Einfügen von 72: Die Balance in 60 ist -2. Es ist eine einfache Linksrotation erforderlich.

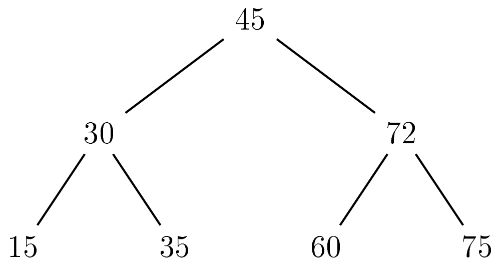


d) Löschen von 90: Die Balance in 75 ist 2. Es ist eine einfache Rechtsrotation erforderlich.



e) Löschen von 70: 70 wird durch seinen direkten Vorgänger 60 ersetzt. Die Balance in 70 ist -2. Es ist eine doppelte Rotation erforderlich, und zwar erst im Knoten 75 nach rechts, und dann in 60 nach links.





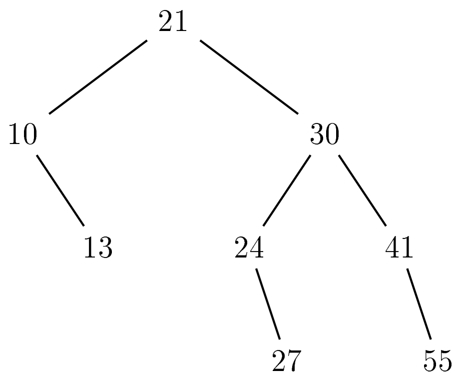
.

Präsenzaufgabe 8.2: (Datenstrukturen)

a) Geben Sie die Implementierung einer Funktion Positionssuche(T, k) in Pseudocode an, die für eine natürliche Zahl k einen Knoten mit dem k-kleinsten der im AVL-Baum T enthaltenen Schlüssel bestimmt. Ist ein solcher nicht vorhanden, so soll **nil** zurückgegeben werden. Versuchen Sie eine Worst-Case Laufzeit von $O(\log n)$ zu erreichen, wenn n die Anzahl der im AVL-Baum enthaltenen Knoten ist. Die Worst-Case Laufzeit ist in jedem Fall anzugeben und zu begründen.

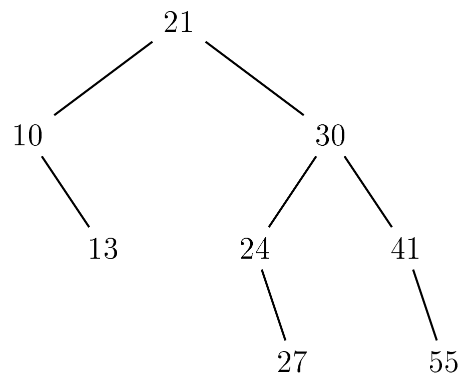
   Hinweis: Sie dürfen die Knoten der AVL-Bäume um geeignete Zusatzinformationen erweitern. Dabei sollten Sie alle von einer solchen Modifikation betroffenen Operationen entsprechend anpassen.

b) Veranschaulichen Sie die Arbeitsweise Ihrer Implementierung anhand des folgenden AVL-Baumes für den fünftkleinsten Schlüssel, d.h. für den Aufruf Positionssuche($T, 5$).



Lösung:

a) Wir erweitern die Knoten der AVL-Bäume um eine weitere Komponente *size*. Für einen Knoten v eines AVL-Baumes gibt $size(v)$ die Anzahl der Knoten des Teilbaumes an, dessen Wurzel v ist. Z.B. gilt für den AVL-Baum T:



$size(T_{21}) = 8$, $size(T_{30}) = 5$ und $size(T_{10}) = 2$, wobei wir hier die Knoten über ihren zugeordneten Schlüssel im AVL-Baum T notiert haben.

Unsere noch anzugebende Funktion $\mathsf{Positionssuche(T, k)}$ startet im Wurzelknoten unseres AVL-Baumes T.

- Ist $size(T) < k$, dann kann es keinen k.-kleinsten Schlüssel geben, da der Baum T zu wenig Knoten enthält. Für die weiteren Betrachtungen sei daher $k \leq size(T)$ angenommen.
- Ist $size(lc(T)) = k - 1$, dann ist der Schlüssel des Wurzelknotens von T das gesuchte Element.
- Ist $k \leq size(lc(T))$, dann ist der gesuchte Schlüssel im linken Teilbaum von T, so dass wir rekursiv im linken Teilbaum weiter suchen.
- Ist $k > size(lc(T)) + 1$, so ist der gesuchte Schlüssel im rechten Teilbaum. Da der linke Teilbaum bereits $size(lc(T))$ viele kleinere Schlüssel enthält und der Schlüssel des Wurzelknotens von T ebenfalls größer sein muss, müssen wir im rechten Teilbaum von T nur noch nach dem $k - (size(lc(T)) + 1)$.-kleinsten Schlüssel suchen.

Mit diesen Vorüberlegungen können wir die Implementierung angeben.

Positionssuche(T, k):
`// Eingabe: AVL-Baum T, natürliche Zahl` k.
`// Ausgabe: Knoten mit` k.`-kleinstem Schlüssel.`

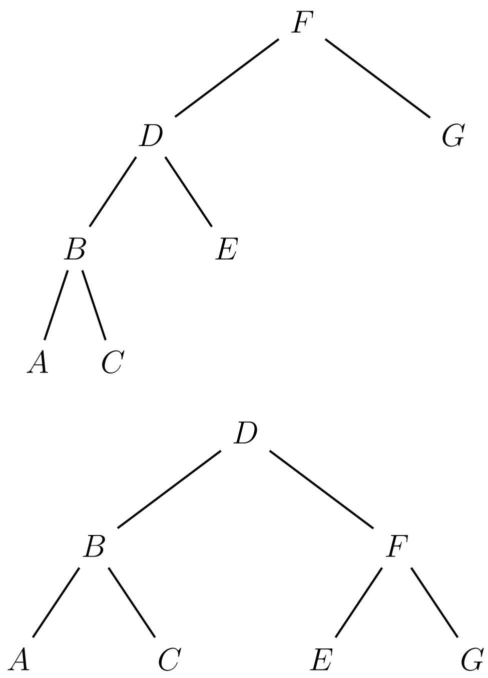
if $k > size[T]$ **then**
    return nil
else if $size[lc(T)] = k - 1$ **then**
    return $key[T]$
else if $k \leq size[lc(T)]$ **then**
    return Positionssuche$(lc[T], k)$
else
    return Positionssuche$(rc[T], k - (size[lc(T)] + 1))$

Die *size*-Information der Knoten kann in die AVL-Baum-Operationen integriert werden, ohne deren asymptotische worst-case Laufzeit zu erhöhen. Genauer betrachtet, sind folgende Erweiterungen/Modifikationen an den AVL-Baum-Operationen erforderlich:

Einfügen: Die *size*-Komponente des einzufügenden Knotens wird auf 1 gesetzt. Entlang des Suchpfades von der Wurzel zum einzufügenden Element werden die *size*-Komponenten aller durchlaufenen Knoten um 1 erhöht.

Delete: Entlang des Suchpfades von der Wurzel zum zu löschenden Element werden die *size*-Komponenten aller durchlaufenen Knoten um 1 erniedrigt.

Rechtsrotation: Eine Rechtsrotation (wie unten stilisiert, in F):



erfordert die Aktualisierung von genau zwei *size*-Komponenten, und zwar D und F. Es gilt $size(D)_{\text{alt}} = size(B) + size(E) + 1$, $size(F)_{\text{alt}} = size(B) + size(E) + 1 + size(G) + 1$, $size(D)_{\text{neu}} = size(B) + 1 + size(E) + size(G) + 1$ und $size(F)_{\text{neu}} = size(E) + size(G) + 1$. Somit $size(F)$ wird um $size(E) + 1$ dekrementiert, während $size(D)$ um $size(G) + 1$ inkrementiert wird.

Linksrotation: Analog zur Rechtsrotation.

Der Aufruf von Positionssuche hat für einen gegebenen Knoten konstanten Aufwand. Die Anzahl der Aufrufe ist offensichtlich durch $h(T)$ beschränkt. Wegen der AVL-Baumeigenschaft ist damit die worst-case Laufzeit der Operation in $O(\log n)$.

b) Aus Gründen der Darstellung kürzen wir den Aufruf der Funktion *Positionssuche* durch P ab. Dann ergibt sich durch die Folge der rekursiven Aufrufe folgende Berechnung:

$$
\begin{aligned}
P(T_{21}, 5) &= P(T_{30}, 5 - 2 - 1) && \text{da } size[T_{10}] = 2 \\
&= P(T_{24}, 2) && \text{da } size[T_{24}] \geq 2 \\
&= P(T_{27}, 2 - 1) && \text{da } size[lc(T_{24})] = 0 \\
&= 27 && \text{da } size[lc[T_{27}]] = 0
\end{aligned}
$$