12.12.2001

Theoretische Informatik 1, WS 2001/2002

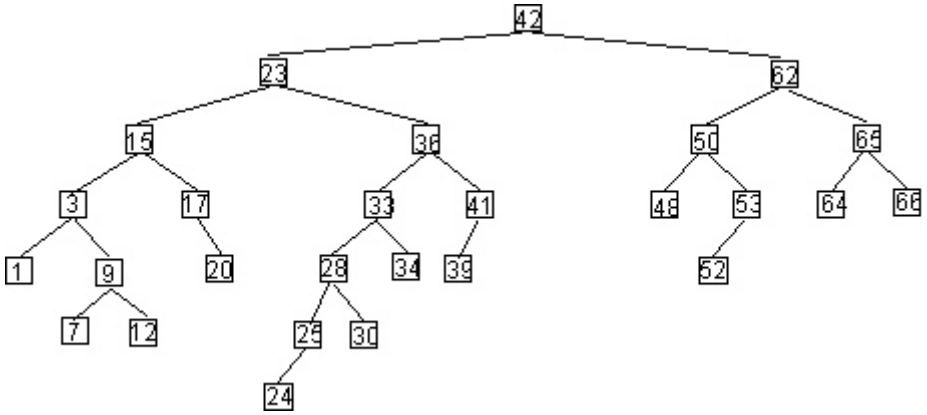
Übungsblatt 8

Keine Garantie für die Korrektheit der hier vorgestellten Lösungen.

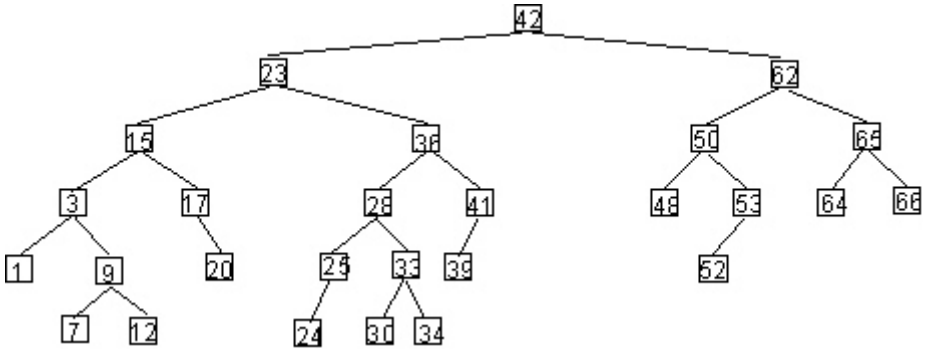
Aufgabe 8.1

(a)

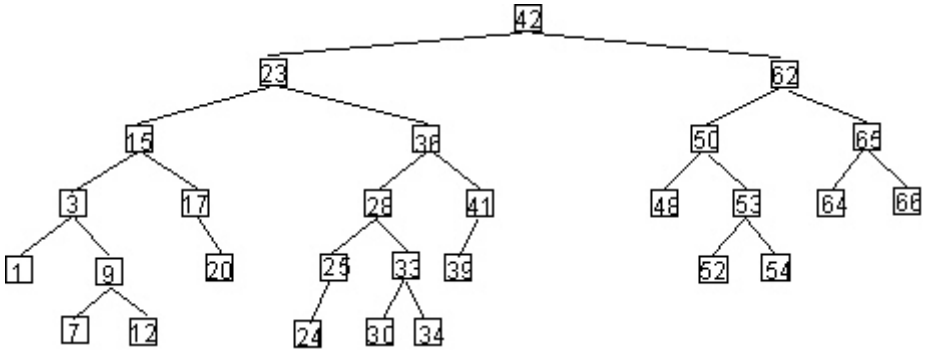
24 einfügen:



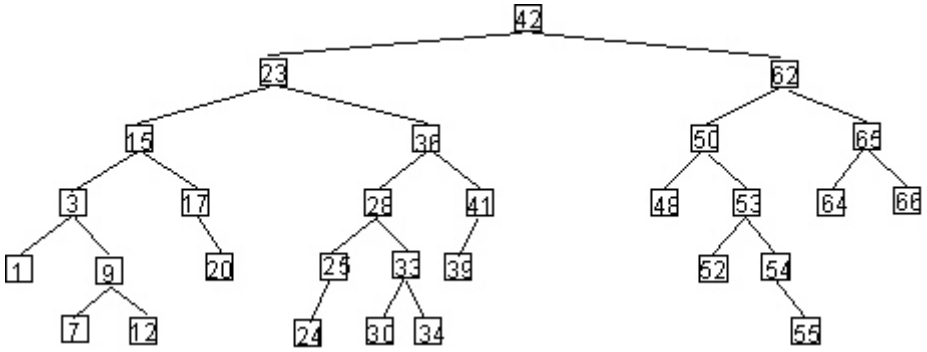
Rechtsrotation bei 33:



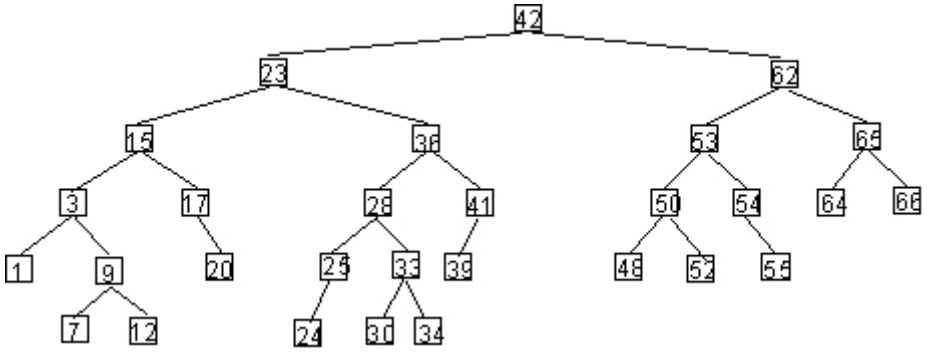
54 einfügen:



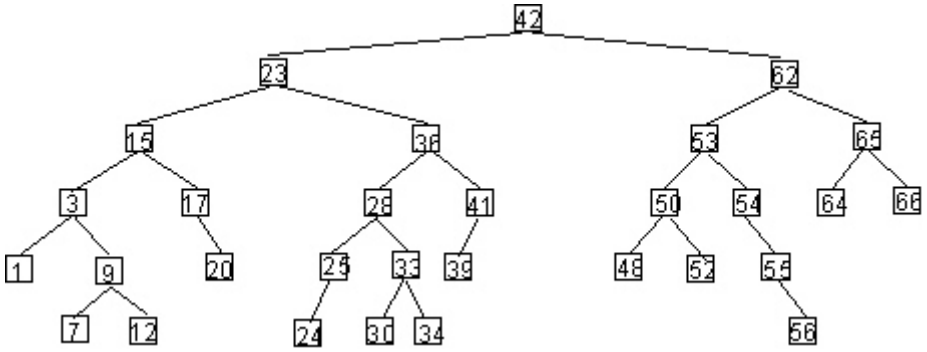
55 einfügen:



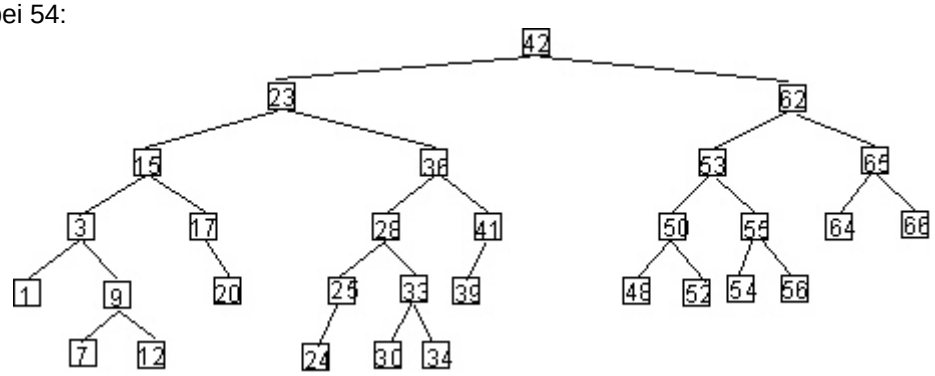
Linksrotation bei 50:



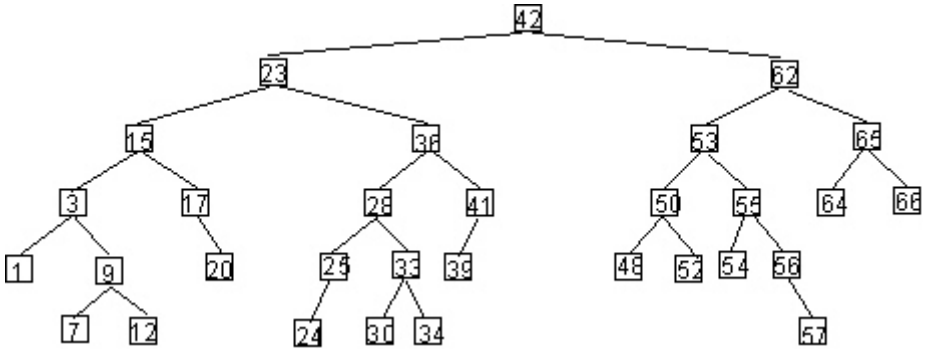
56 einfügen:



Linksrotation bei 54:

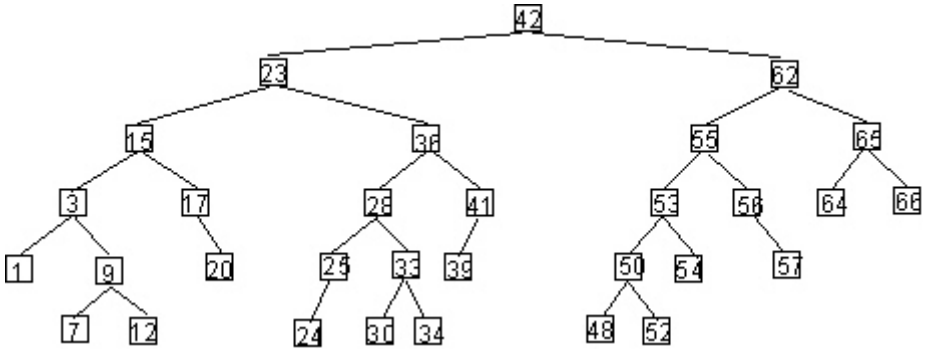


57 einfügen:

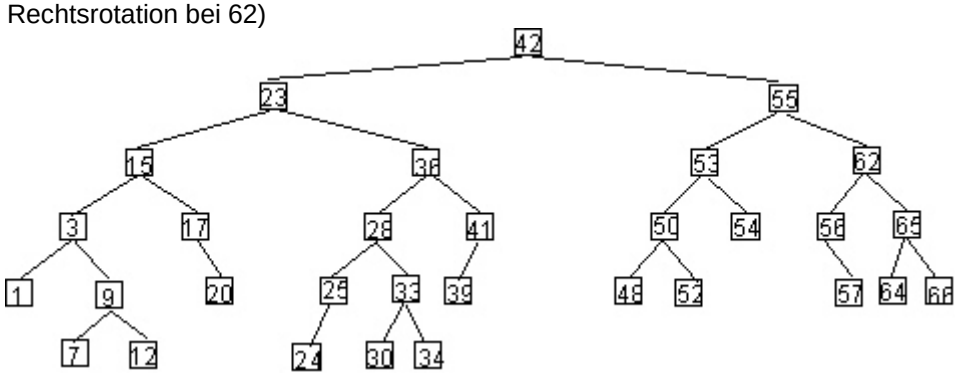


Zack-Zick bei 62:

(zuerst Linksrotation bei 53)



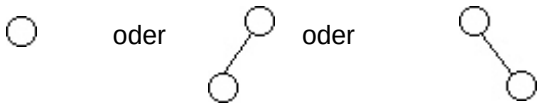
(dann Rechtsrotation bei 62)



(b)

Sei $Diff_{max}$ die maximale Differenz der Tiefen zweier Blätter.

Für AVL-Bäumen der Form



gilt, dass $Diff_{max} = \Omega(Tiefe(T))$ ist. Da $Tiefe(T) = \log n \leq c \cdot Diff_{max}$.

Aufgabe 8.2

Behauptung:

Ein binärer Suchbaum lässt sich eindeutig aus der Postorder Reihenfolge seiner Schlüssel rekonstruieren. (D.h. es gibt keine zwei Bäume, die die gleiche Postorder haben.)

Beweis:

Postorder: linkes Kind - rechtes Kind - Wurzel

Um den binären Suchbaum zu erzeugen, beginne am Ende der Postorder Reihe. Der letzte Schlüssel ist die Wurzel. Wenn die Wurzel zwei Kinder hat, steht direkt neben der Wurzel deren rechtes Kind. Wenn man die Reihe nun weiter durchläuft, ist der erste Schlüssel, der kleiner als die Wurzel ist, deren linkes Kind. Das Verfahren wird rekursiv auf die Kinder der Wurzel angewendet, wenn diese aus mehr als einem Schlüssel bestehen.

I.V.:

Sei n die Anzahl der Knoten im binären Suchbaum.

n = 1:

Der Baum besteht aus einem Knoten. In der Postorder steht nur ein Wert. Damit ist die Postorder eindeutig.

n = 2:

Der Baum besteht aus Wurzel und rechtem oder linkem Kind. Der letzte Schlüssel der Postorder ist die Wurzel. Da wir einen binären Suchbaum haben, gibt der Wert des nächsten Schlüssels an, ob er ein rechtes oder linkes Kind ist. Wenn der Wert kleiner als die Wurzel ist, ist er das linke Kind. Ist er dagegen größer, ist er das rechte Kind der Wurzel.

I.S.:

$n \quad n \quad 1$

Die Behauptung gelte für Postorder mit n Schlüsseln.
Sei eine Postorder mit n+1 Schlüsseln gegeben. Der letzte Schlüssel ist die Wurzel. Die Teilfolge zwischen Wurzel und erstem Schlüssel < der Wurzel bildet den rechten Teilbaum. Die Teilfolge ab dem ersten Schlüssel < der Wurzel bildet den linken Teilbaum. Die beiden Teilfolgen bestehen jeweils aus $\leq$ n Schlüssel. Damit gilt die Behauptung für diese beiden Teilfolgen. Der Baum lässt sich nun eindeutig konstruieren, da der letzte Schlüssel die Wurzel ist und die Werte der Teilfolgen angeben, ob diese rechtes oder linkes Kind der Wurzel sind.

❑

Aufgabe 8.3

Idee:

Durchlaufe Baum auf der Suche nach x. Sobald eine Richtungsänderung vorliegt, werden Kanten gelöscht. Dadurch entstehen neue Bäume. Eine Richtungsänderung liegt vor, wenn beim Durchlauf erst in einem rechten (bzw. linken) und dann in einem linken (bzw. rechten) Teilbaum nach x gesucht wird.
Die Kinder der Wurzeln der neuen Bäume geben an, ob die Bäume kleinere oder größere Werte als x speichern. Die Wurzel eines neu entstandenen Baums hat nach seiner Abtrennung entweder ein linkes oder ein rechtes Kind. Wenn eine Wurzel ein rechtes Kind hat, speichert dieser Baum nur Werte $\geq$ x. Wenn eine Wurzel ein linkes Kind hat, speichert dieser Baum nur Werte $\leq$ x.
Nach den ersten beiden Trennungen gibt es drei Teilbäume. Einer mit Werten < x, sei dieser A; einer mit Werten > x, sei dieser B und ein Baum mit Werten $\leq$ x und $\geq$ x, sei dieser C. In Baum C werden weiter Trennungen durchgeführt, wenn es zu einem Richtungswechsel kommt. Dabei entstehen weitere Bäume. Sobald x gefunden wurde, werden die Bäume entsprechend den Werten ihrer Wurzel an Baum A oder B angehängt.

Algorithmus:

```
Beginne bei Wurzel.
if (x < Wurzel.Wert)
        suchen (x, beginne bei linkem Kind der Wurzel);
if (x > Wurzel.Wert)
        suchen (x, beginne bei rechtem Kind der Wurzel);
else            // x ist Wurzel
        trenne Kante zwischen Wurzel und linkem Kind der Wurzel;
        Programm Ende;

suchen('zu suchender Wert', 'beginne bei') {

        für alle Knoten i auf dem Weg zu x do
                if (i.Wert < x)
                        if (i-1.Wert > x)
                                trenne Kante zwischen i und i-1;
                        gehe weiter bei rechtem Kind;
                if (i.Wert > x)
                        if (i-1.Wert < x)
                                trenne Kante zwischen i und i-1;
                        gehe weiter bei linkem Kind;
                else
                        x gefunden;
                        baeume_anhaengen();
}


baeume_anhaengen() {
        /* wie in der Idee erwähnt, sind nach den ersten beiden Trennungen die Bäume A und B
         * entstanden. */
```

```
        In der Reihenfolge, in der die Bäume entstanden sind,
        do für alle Bäume außer A und B {
            if (Wurzel.Wert < x)
                hänge Baum an A an;
            if (Wurzel.Wert ≥ x)
                hänge Baum an B an;
        }
    }
```

Laufzeit:

x suchen →O(log n)

Kanten löschen → O(1)

Bäume an Baum A oder B anfügen → O(1). Die Bäume müssen nicht ineinander sortiert werden. Ein Baum muss nur an einen Knoten angehängt werden.

Insgesamte Laufzeit O(log n), wobei log n die Tiefe des Baumes ist.