FernUniversität in Hagen

Lösungsvorschläge zur Nachklausur „1661 Datenstrukturen I“

17.9.2011

Aufgabe 1

(a)

algebra *art*

sorts *art, picture, room, card, area*

ops

createPicture:	*string* × *area* × *string*	→	*picture*
createCollection:		→	*art*
createRoom:	*card* × *area*	→	*room*
insertPicture:	*room* × *picture*	→	*room*
insertRoom:	*art* × *room*	→	*art*
deleteRoom:	*art* × *room*	→	*art*
deletePicture:	*room* × *picture*	→	*room*
getArtist:	*picture*	→	*string*
isEqual	*picture* × *picture*	→	*bool*
getMaxArea:	*room*	→	*area*
getUsedArea:	*room*	→	*area*
getCapacity:	*room*	→	*area*
containsPicture:	*art* × *picture*	→	*bool*
findPicture:	*art* × *picture*	→	*art*

(b)

sets $card = \mathbb{N}$

$area = \mathbb{R}^+$

$picture = \{(titel, size, artist) \mid titel, artist \in string, size \in area\}$

$= string \times area \times string$

$room = \{(number, size, l) \mid number \in card, size \in area,$

$l \subseteq picture\}$

$= card \times area \times (F(picture))$

$art = \{r \subseteq room \mid \forall (n,s,l),(n',s',l') \in r: n=n' \Rightarrow (s=s') \wedge (l=l') \}$

(c)

functions:

createPicture(*t, s, a*) = (*t, s, a*)

createCollection( ) = { }

createRoom$(i, s) = (i, s, \{\})$

$insertPicture((i, f, p), (t, s, a))$

$$= \begin{cases} (i, f, p \cup \{(t, s, a)\}) & \text{, falls } getCapacity((i, f, p)) \geq s \\ (i, f, p) & \text{, sonst} \end{cases}$$

$$insertRoom(a, (x, y, z)) = \begin{cases} a & \text{, falls } \forall (n, s, l) \in a : \text{n} \neq x \\ a \cup \{(x, y, z)\} & \text{, sonst} \end{cases}$$

$$deleteRoom(a, b) = a / \{b\}$$

$$deletePicture((i, f, p), b) = (i, f, p / \{b\})$$

getArtist$((t, s, a)) = a$

$isEqual(p_1, p_2) = \forall i \in \{1, 2, 3\} : (\pi_i(p_1) = \pi_i(p_2))$

getMaxArea$((i, f, p)) = f$

$getUsedArea((i, f, p)) = \sum_{x \in p} \pi_2(x)$

getCapacity(r) = *getMaxArea*(r) - *getUsedArea*(r)

$containsPicture(b, s) = \exists i \in \pi_3(b) : isEqual(i, s)$

findPicture$(a, p) = \{ b \in a \mid$ *containsPicture*$(b, p)\}$

end *art*

Anmerkung: Alternative Lösungen sind natürlich zulässig.

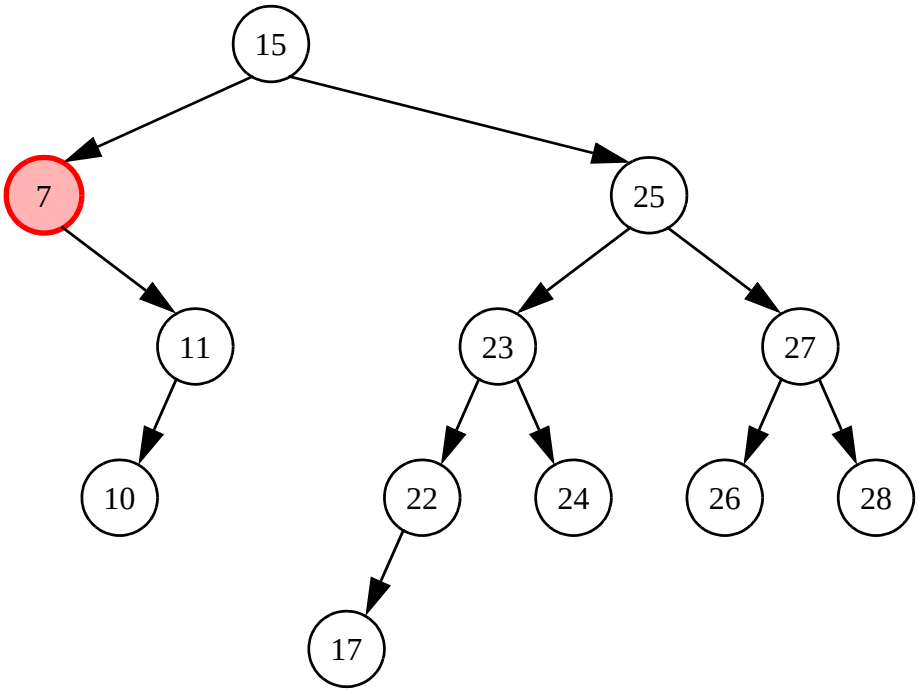
Aufgabe 2

(a)

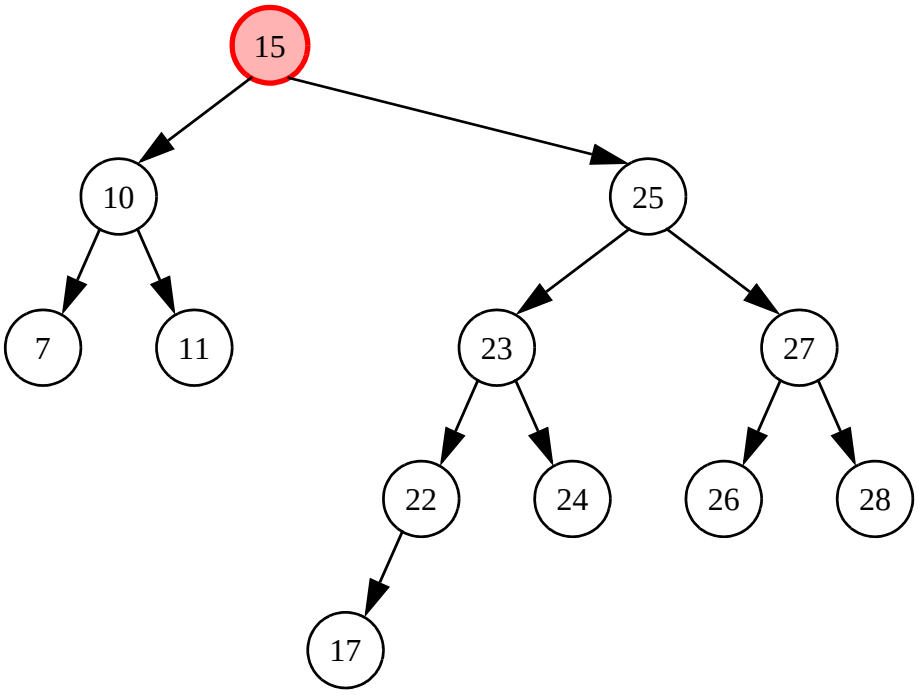
Korrekturhinweis:

Treten beim Löschen eines Schlüssels fortgesetzte Balancierungsoperationen auf, so genügt es, den Zustand vor der ersten und nach der letzten Operation darzustellen.

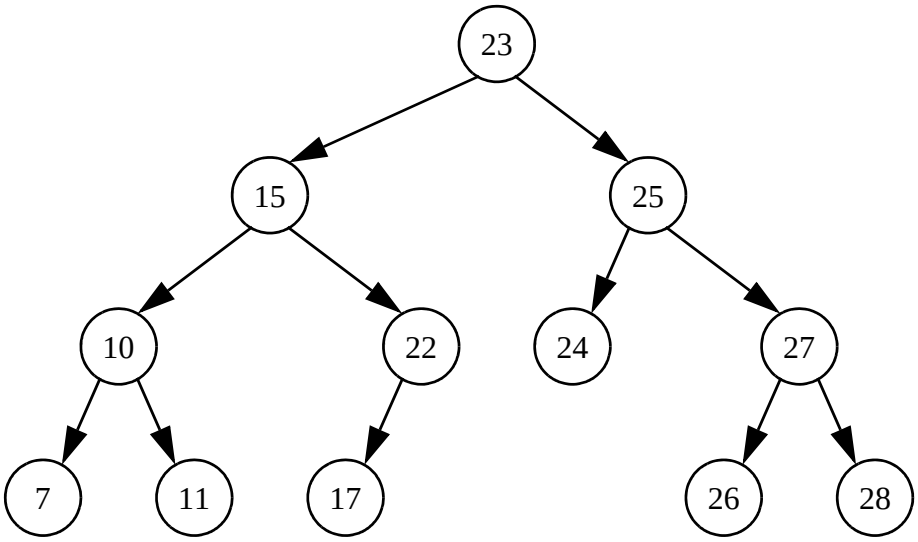
Nach dem Löschen von „2“ gerät „7“ außer Balance:



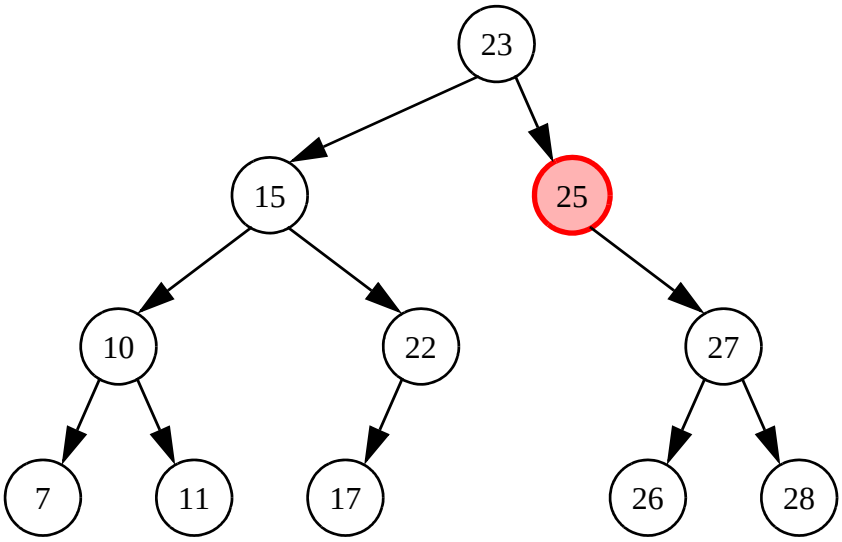
Es erfolgt eine Doppelrotation um „11“ und „7“, nach der „15“ unbalanciert ist:



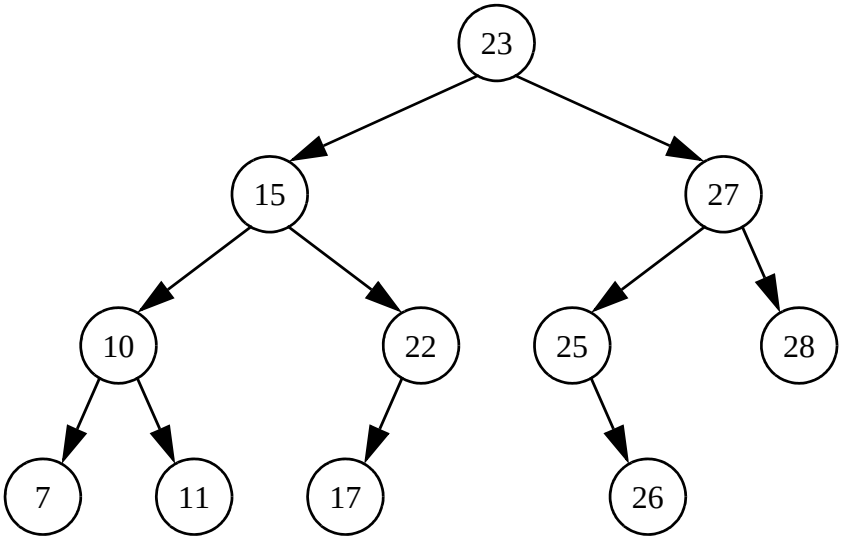
Eine Doppelrotation um „25“ und „15“ behebt dies:



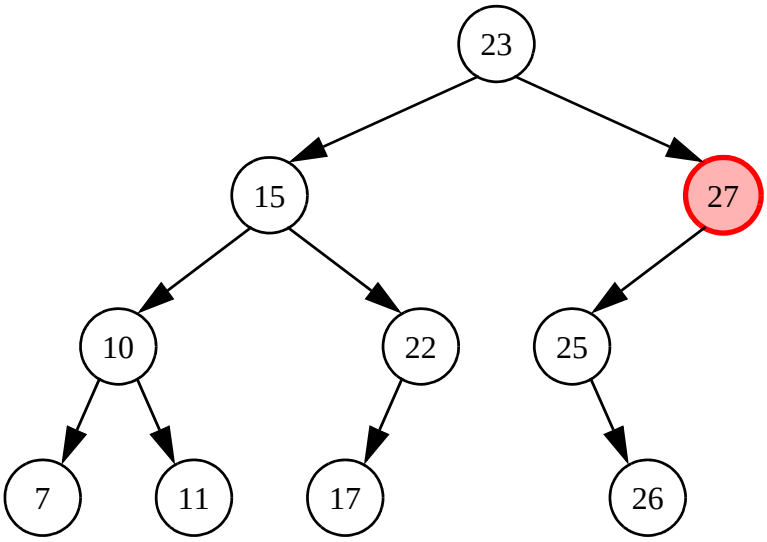
Nun wird die „24“ gelöscht, worauf „25“ außer Balance gerät:



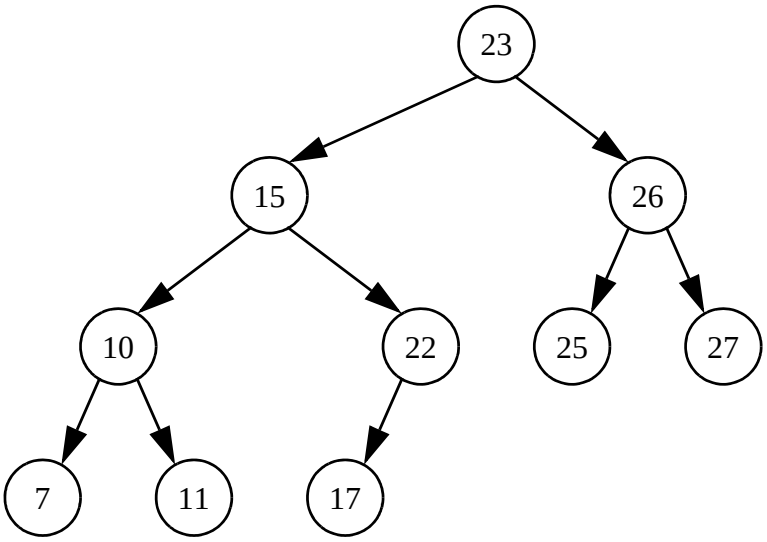
Eine Rotation um „25“ schafft Abhilfe:



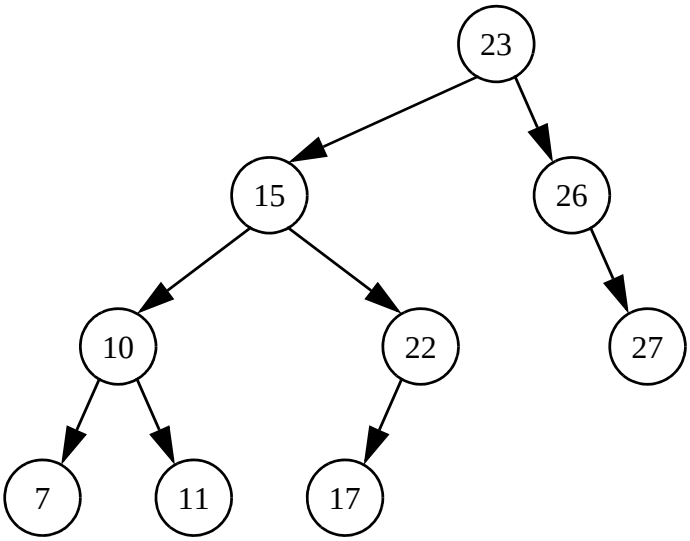
Nach Löschen der „28“ ist die „27“ „linksschwer“:



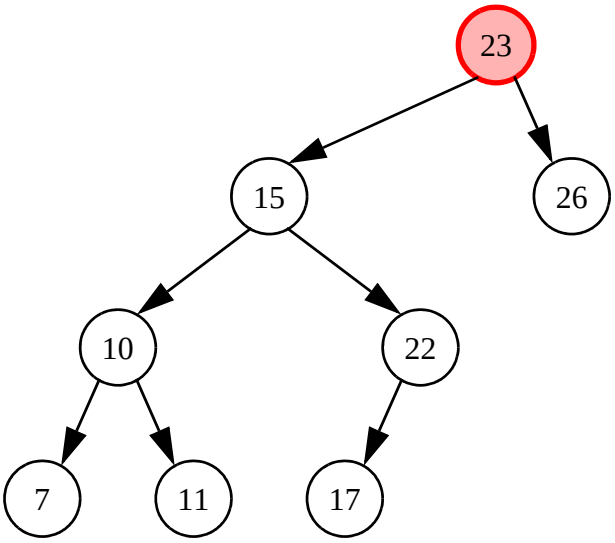
Eine Doppelrotation bei „27“ genügt:



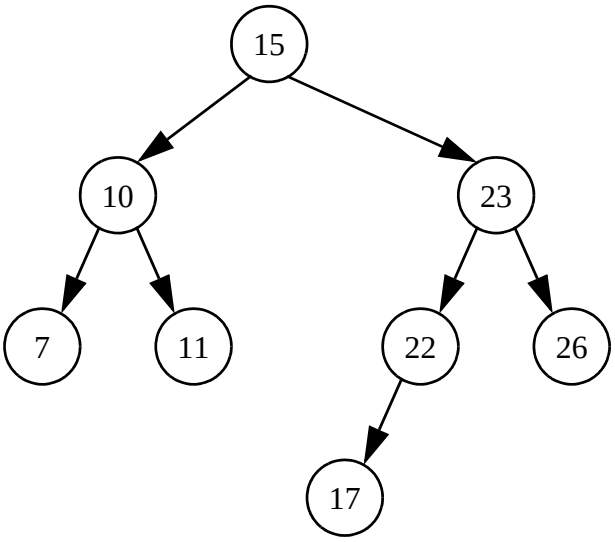
Die 25 kann problemlos gelöscht werden:



Nach Löschen der „27“ ist die Wurzel „23“ außer Balance:



Nach einer einfachen Rotation um „23“ ist diese Situation bereinigt:



(b)

Die Laufzeitkomplexität von *arrayToAVLtree* ist in O(*n*).

(c)

Beweis: Da der Algorithmus rekursiv arbeitet, müssen wir die Rekursionsgleichung finden und lösen. Sei T_n die Laufzeit für einen Aufruf auf n Elementen, gemessen in Zugriffen auf A.

Es gilt offenbar $T_0 = 0$ (**then**-Teil des ersten **if**-Konstrunkts (Zeile 12), kein Zugriff auf A), $T_1 = 1$ (**then**-Teil des zweiten **if**-Konstrukts (Zeile 15), genau 1 Zugriff auf A) und $T_2 = 2$ (**then**-Teil

des dritten **if**-Konstrukts (Zeile 18), genau 2 Zugriffe auf A).
Für $n > 2$ (**else**-Teil des dritten **if**-Konstrukts (Zeilen 20-24)) gilt:

$$T_n = T_{\lfloor n/2 \rfloor} + 1 + T_{\lceil n/2 \rceil}$$
$$T_n \leq T_{\lceil n/2 \rceil} + 1 + T_{\lceil n/2 \rceil} = 1 + 2 \cdot T_{\lceil n/2 \rceil}$$

Zur Lösung der Rekursionsgleichung betrachten wir o.B.d.A. für n nur die Folge aller Zweierpotenzen natürlicher Zahlen. Dann gibt es bei der Berechnung von T_n, $n = 2^m > 2$, gerade m Rekursionsebenen. Auf der i-ten Rekursionsebene erfolgen 2^i direkte Zugriffe auf A und es gilt:

$$\begin{aligned} T_n &= T_{2^m} \\ &\leq 1 + \sum_{i=1}^{m} 2^i = 1 + (2 + 4 + \ldots + 2^m) \\ &= 1 + (2^{m+1} - 1) \\ &= 2^{m+1} = 2 \cdot 2^m = 2n \\ &\leq 3n - 1 = S_n, \text{ für } n > 0 \end{aligned}$$

Wir schätzen T_n nach oben großzügig durch S_n ab. Dies können wir unbesorgt tun, da wir lediglich zeigen wollen, dass $T_n = O(n)$ gilt. Dazu müssen wir noch die Korrektheit von $T_n \leq S_n$ zeigen. Wir tun dies wiederum für alle Zweierpotenzen $n = 2^i$, $i \in \mathbb{N}$ per vollständiger Induktion über i.

Induktionsanfang:
$T_1 = 1 \leq S_1 = 3 \cdot 1 - 1 = 2$,
$T_2 = 2 \leq S_2 = 3 \cdot 2 - 1 = 5$.

Induktionsschluss:
Unter der Annahme der Korrektheit für alle $n = 2^i$, $i \leq j$, $n > 0$, ergibt sich der Induktionsschritt von $n = 2^i$ nach $m = 2n = 2^{i+1}$:

$$\begin{aligned} T_m &= 2 \cdot T_{m/2} + 1 = 2 \cdot T_n + 1 \\ &\leq 2S_n + 1 = 2 \cdot (3n - 1) + 1 \\ &= 2 \cdot 3n - 2 + 1 = 3m - 2 + 1 = 3m - 1 = S_m \end{aligned}$$

Für $n \neq 2^i$, $i \in \mathbb{N}$, ersetzen wir n durch die nächstgrößere Zweierpotenz und erhalten genau 1 weitere Rekursionsebene.

Es gilt also jeweils $T_n \leq S_n = O(n)$.

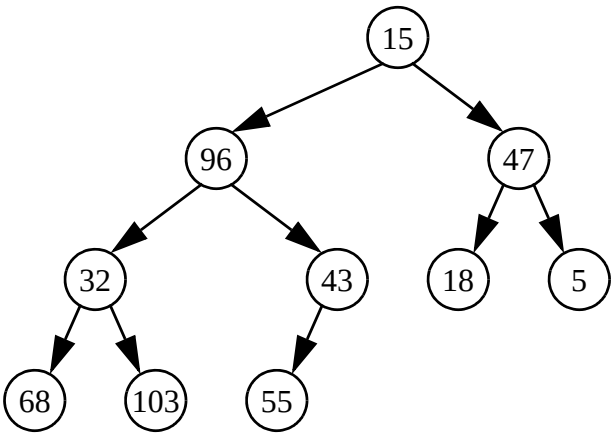
Aufgabe 3

(a)

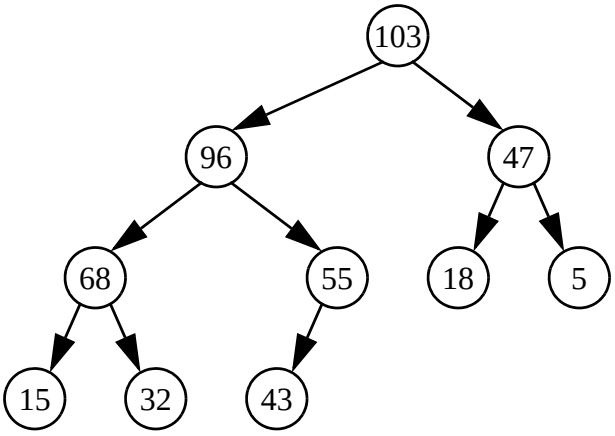
Bei Quicksort mit *findx* für die Bestimmung des Pivotelements tritt bei der Eingabe der sortierten Liste der worst case ein. Der Aufrufbaum entartet beim Divideschritt zu einer Liste und die Gesamtlaufzeit beträgt $O(n^2)$. Heapsort sortiert in allen Fällen in $O(n \log n)$. Für Bubblesort ist die schon sortierte Folge der optimale Fall. Bubblesort durchläuft die Liste einmal, stellt fest, dass keine Vertauschung nötig ist und ist in $O(n)$ fertig. Dementsprechend ist das im allgemeinen Fall sehr schlechte Bubblesort optimal, wenn die Folge bereits sortiert vorliegt.

(b)

Die Baumeinbettung ergibt:



Als initalen Heap erhalten wir:



Oder im Array ausgedrückt:

103 - 96 - 47 - 68 - 55 - 18 - 5 - 15 - 32 - 43

Entnehmen Maximum 103 und anschließendes Einsinken der 43 ergibt:

96 - 68 - 47 - 43 - 55 - 18 - 5 - 15 - 32 || 103

Entnehmen Maximum 96 und anschließendes Einsinken der 32 ergibt:

68 - 55 - 47 - 43 - 32 - 18 - 5 - 15 || 96 - 103

Entnehmen Maximum 68 und anschließendes Einsinkenlassen der 15 ergibt:

55 - 43 - 47 - 15 - 32 - 18 - 5 || 68 - 96 - 103

Entnehmen Maximum 55 und anschließendes Einsinkenlassen der 5 ergibt:

47 - 43 - 18 - 15 - 32 - 5 || 55 - 68 - 96 - 103

Entnehmen Maximum 47 und anschließendes Einsinkenlassen der 5 ergibt:

43 - 32 - 18 - 15 - 5 || 47 - 55 - 68 - 96 - 103

Entnehmen Maximum 43 und anschließendes Einsinkenlassen der 5 ergibt:

32 - 15 - 18 - 5 || 43 - 47 - 55 - 68 - 96 - 103

Entnehmen Maximum 32 und anschließendes Einsinkenlassen der 5 ergibt:

18 - 15 - 5 || 32 - 43 - 47 - 55 - 68 - 96 - 103

Entnehmen Maximum 18 und anschließendes Einsinkenlassen der 5 ergibt:

15 - 5 || 18 - 32 - 43 - 47 - 55 - 68 - 96 - 103

Entnehmen Maximum 15:

5 || 15 - 18 - 32 - 43 - 47 - 55 - 68 - 96 - 103

Die 5 bleibt als letztes Element wo sie ist. Das sortierte Array hat die Form:

5 - 15 - 18 - 32 - 43 - 47 - 55 - 68 - 96 - 103

(c)

In der ersten Phase sortieren wir nach der Zahl am Ende des Kennzeichens. Wir benutzen 9999 Behälter und erhalten für die benutzten Behälter:

123	432	1234
HH - KO 123	DO - TO 432	HA - TO 1234
BO - AJ 123	DO - AJ 432	HA - T 1234
		HH - TO 1234

Alle anderen Behälter bleiben leer. In der zweiten Phase sortieren wir nach dem mittleren Buchstabenkombination mit Behältern von A, AA, ..., B, ... , ZZ. Wir erhalten für die benutzten Behälter:

AJ	KO	T	TO
BO - AJ 123	HH - KO 123	HA - T 1234	DO - TO 432
DO - AJ 432			HA - TO 1234
			HH - TO 1234

Alle anderen Behälter bleiben wieder leer. In der dritten Phase sortieren wir nach dem Zulassungsstelle. Die Behälter seien ähnlich zur zweiten Phase mit A,...,AZZ,B,..,ZZZ gegeben. Für die benutzten Behälter erhalten wir:

BO	DO	HA	HH
BO - AJ 123	DO - AJ 432	HA - T 1234	HH - KO 123
	DO - TO 432	HA - TO 1234	HH - TO 1234

Alle anderen Behälter bleiben wieder leer. Die sortierte Folge ergibt sich mit:

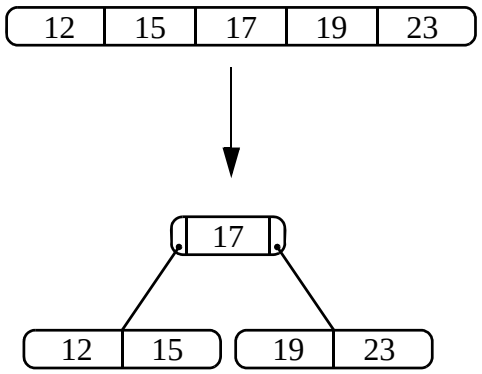
BO - AJ 123, DO - AJ 432, DO - TO 432, HA - T 1234, HA - TO 1234, HH - KO 123, HH - TO 1234

(d)

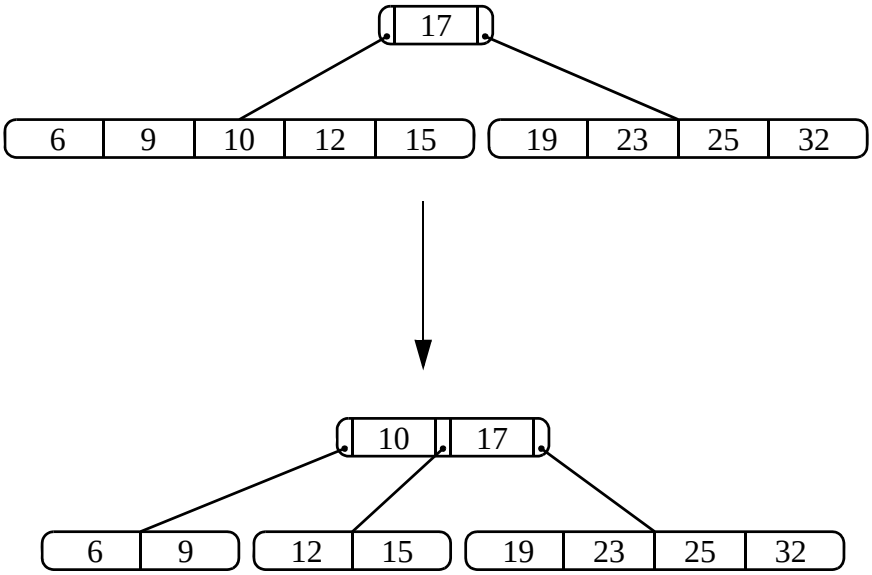
Gar nicht. Radixsort fängt „blind“ an zu sortieren. Schon sortierte Ergebnisse haben keinen Einfluss auf die Anzahl der Phasen und Behälter je Phase.

Aufgabe 4

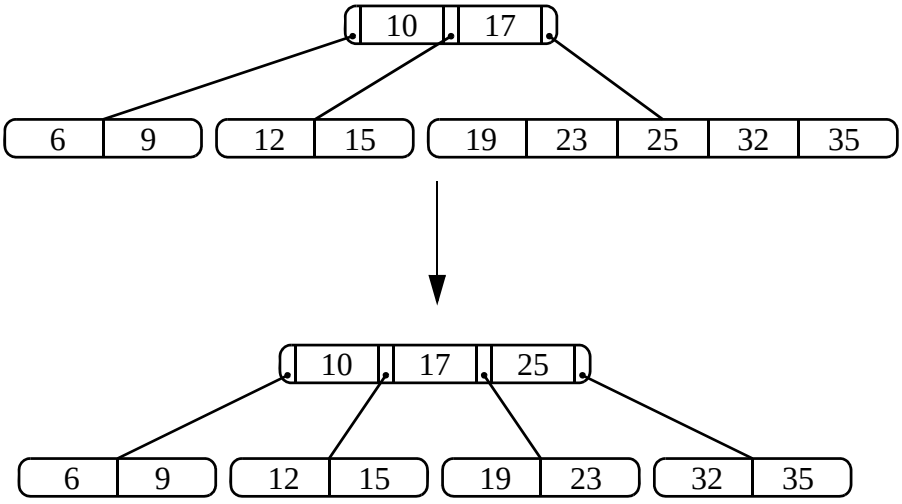
(a)
Ein Knoten eines B-Baums der Ordnung 2 kann bis zu 4 Elemente aufnehmen. Daher tritt der erste Überlauf beim Einfügen des 5. Elements (19) auf:



Erst beim Einfügen der 10 muss erneut eine Überlaufbehandlung durchgeführt werden:

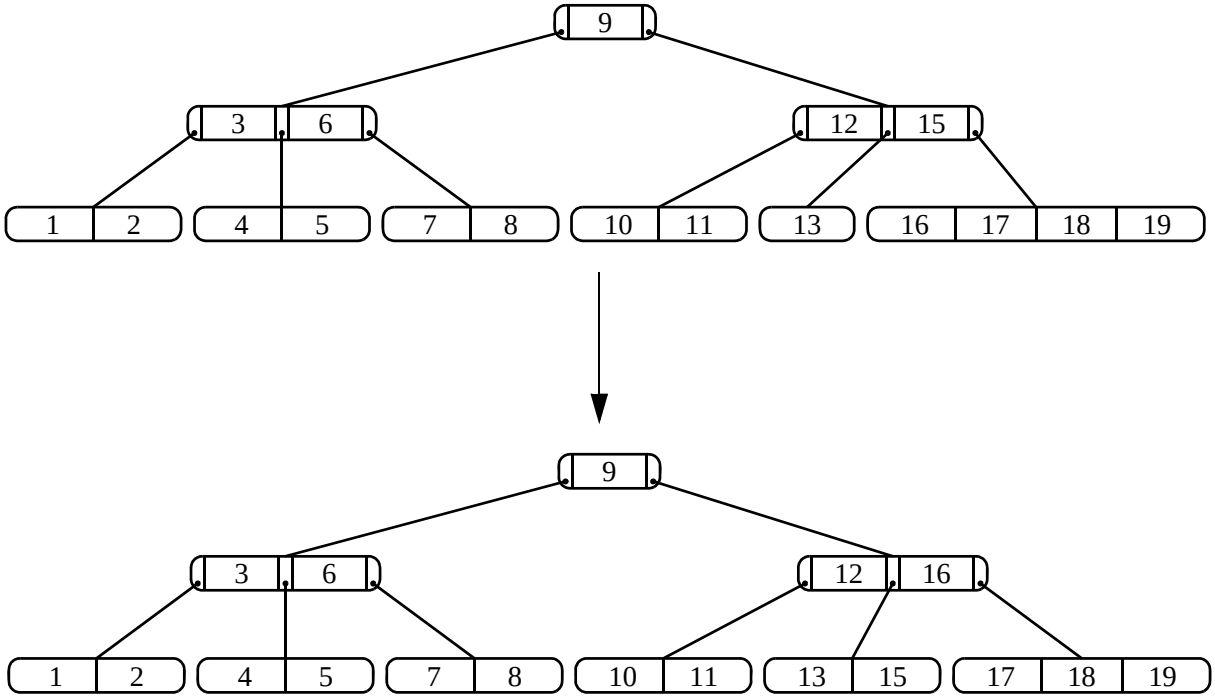


Auch das Einfügen der 35 erfordert einen Split:

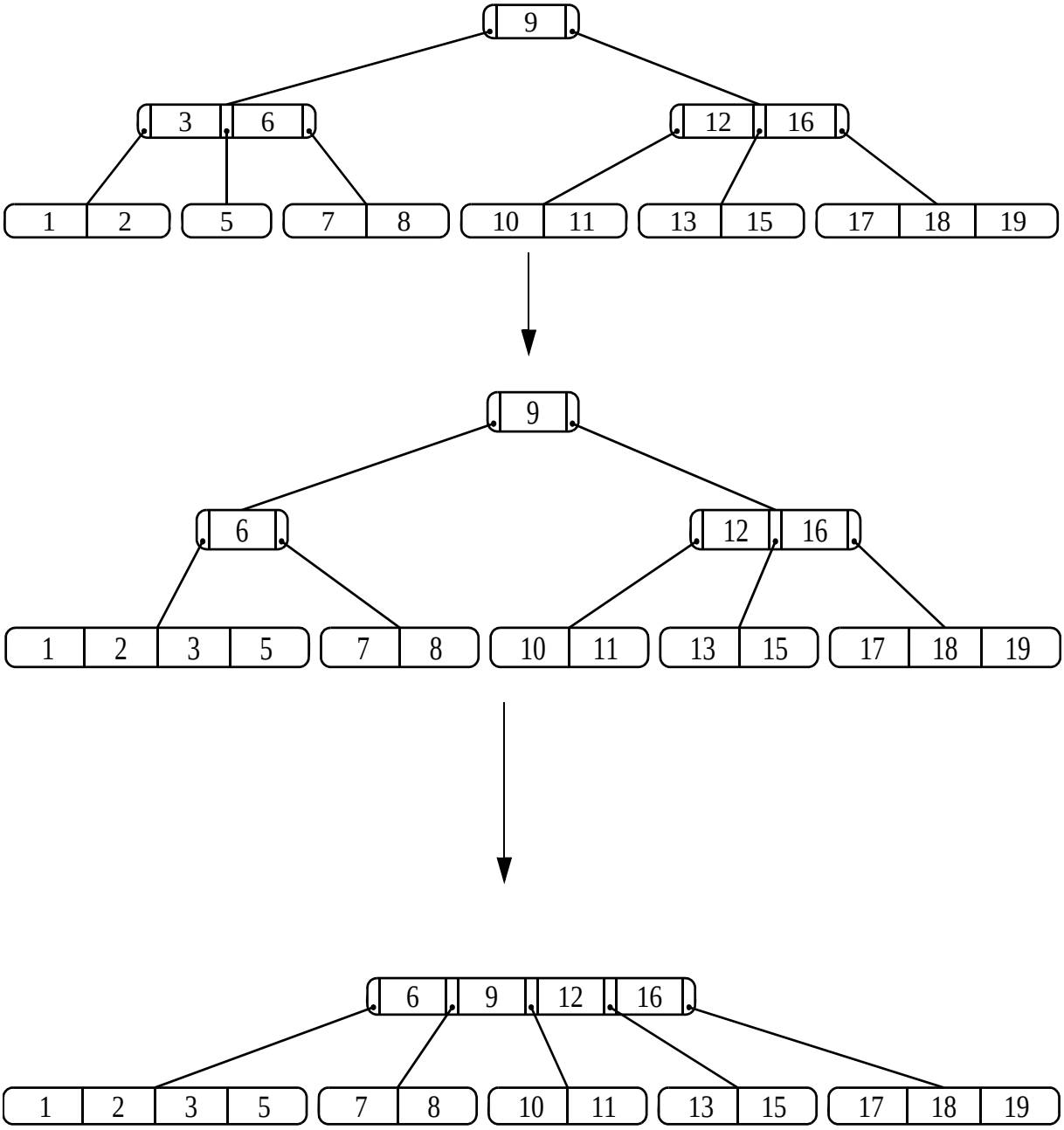


(b)

Beim Löschen der 14 tritt ein Unterlauf im betreffenden Knoten auf. Eine Balance-Operation mit dem rechten Nachbarn ist ausreichend, um den Baum auszugleichen:



Beim Löschen der 4 tritt ebenfalls ein Unterlauf auf. Keiner der Nachbarn hat genügend Knoten für eine Balance-Operation. Es wird ein Merge mit dem linken Nachbarn durchgeführt. Dadurch tritt ein Unterlauf im Knoten, der 3 und 6 beinhaltet auf. Wieder kann keiner der Nachbarn ein Element zum Ausgleich beisteuern. Es findet ein Merge mit dem rechten Nachbarn statt und die Höhe des Baums sinkt:



Aufgabe 5 Deckblatt

Hier erhalten Sie den Punkt, wenn Sie beide Klausurdeckblätter korrekt und vollständig ausgefüllt haben, also Namen, Matrikelnummer und Adresse korrekt eingetragen und genau diejenigen Aufgaben markiert haben, die Sie auch bearbeitet haben.