Hinweise zur Bearbeitung der Klausur zum Kurs 1661 „Datenstrukturen I“

Bitte **lesen Sie** sich diese Hinweise **vollständig und aufmerksam** durch, bevor Sie mit der Bearbeitung der Klausur beginnen.

1. Prüfen Sie bitte die Vollständigkeit Ihrer Unterlagen. Die Klausur umfasst:
   - 2 Deckblätter
   - 1 Formblatt für eine Teilnahmebescheinigung zur Vorlage beim Finanzamt
   - diese Hinweise
   - 5 Aufgaben auf den Seiten 2 - 4
2. Die **Klausurdauer** beträgt **2 Stunden**.
3. Für die Klausur sind **keine Hilfsmittel zugelassen**. Es darf nur unbeschriebenes Konzeptpapier und Schreibzeug verwendet werden. Die Reinschrift der Klausur darf **nicht mit Bleistift** erfolgen.
4. Schreiben Sie Ihre Lösungen auf Ihr **eigenes Papier** (DIN A4) und nicht auf die Seiten mit den Aufgabenstellungen.
5. Bevor Sie mit der **Bearbeitung der Klausuraufgaben** beginnen, füllen Sie bitte die folgenden Teile der Klausur aus:
   - **beide Deckblätter mit Name, Anschrift sowie Matrikelnummer**.
   - Schreiben Sie bitte **auf jedes weitere Blatt** oben links Ihren **Namen und** oben rechts Ihre **Matrikelnummer**. **Nummerieren Sie Ihre Seiten** bitte durch.
   - die Teilnahmebescheinigung, falls Sie diese wünschen.
6. **Wenden Sie** bei der Lösung der Aufgaben, soweit dies möglich ist, die **Algorithmen und Notationen aus den Kurseinheiten an**.
7. **Schreiben Sie**, wenn Algorithmen gefordert sind, **keine kompletten Java-Programme**, sondern beschränken Sie sich auf die wesentlichen Teile des Algorithmus, die z.T. auch in Prosa formuliert werden können. Formulieren Sie Algorithmen aber so, dass die elementaren Einzelschritte erkennbar werden.
   Sparen Sie bei solchen Aufgaben **nicht mit Kommentaren**. Wenn Ihre Lösung aufgrund fehlender Kommentare nicht verständlich ist, führt das zu Punktabzug.
8. **Vermeiden Sie** in jedem Fall **bei der Definition von Funktionen in Algebren Java-Programme sowie Algorithmen**. Geben Sie lediglich mathematische Definitionen an, wie sie im Kurstext verwandt worden sind!
   Ebenso sind **Mengendefinitionen in mathematischer Mengennotation** durchzuführen. Geben Sie auf **keinen** Fall **Datentypdefinitionen** an, und verwenden Sie **keine konkreten Datenstrukturen**!
9. **Vor der Abgabe** Ihrer Klausur:
   - **Heften** Sie Ihre Bearbeitung an Ihr vollständiges Klausurexemplar. **Die Aufgabenblätter müssen mit abgegeben werden!**
   - **Kreuzen** Sie auf beiden Deckblättern die von Ihnen **bearbeiteten Aufgaben an**.
10. Durch Lösen der Aufgaben sind maximal 100 Punkte erreichbar. Sie dürfen damit rechnen einen Übungsschein bzw. ein Zertifikat zu erhalten, wenn Sie insgesamt mindestens 50 Punkte erreichen.
11. Die Klausurergebnisse können Sie ab dem 4. September 2017 in der Virtuellen Universität einsehen. Die Anmeldefrist zur Nachklausur endet am 10. Januar 2018. Bitte beachten Sie, dass Sie sich **selbstständig** zur Nachklausur **anmelden** müssen, da bei Nichtbestehen **keine automatische Anmeldung** erfolgt.

Aufgabe 1 Labyrinth-Algebra

25 Punkte

In dieser Aufgabe sollen Sie eine Algebra entwerfen, um ein Labyrinth darzustellen und in diesem eine Person steuern zu können.

Ein Labyrinth wird hierbei als eine rechteckige Fläche ohne Löcher dargestellt. Diese ist in Felder unterteilt, die durch zwei Koordinaten angesprochen werden können. Ein Feld kann leer sein, den Ausgang darstellen, eine Mauer beinhalten oder eine Person beherbergen. In jeder Dimension ist ein Labyrinth mindestens 4 Felder groß. Es gibt maximal einen Ausgang und eine Person in einem Labyrinth.

Es sollen die folgenden Operationen unterstützt werden:

create Dieser Operator erhält die Dimension des Labyrinths in Form zweier Zahlen. Keine dieser Zahlen darf kleiner als 4 sein. Das erzeugte Labyrinth besteht aus leeren Feldern mit Koordinaten ausgehend von (1,1) bis einschließlich zur übergebenen Größe.

setWall Hier wird auf einem Feld des Labyrinths eine Mauer gesetzt. Ist dieses Feld schon belegt oder befindet sich dieses außerhalb des Labyrinths, so bleibt das Labyrinth unverändert.

setExit Dieser Operator setzt die Position des Ausgangs eines Labyrinths. Der Ausgang muss am Rand des Labyrinths liegen. Ist bereits ein Ausgang gesetzt oder ist die angegebene Position außerhalb des Labyrinths oder befindet sich dort eine Mauer oder eine Person, so bleibt das Labyrinth unverändert.

isReachable Prüft, ob von einer gegebenen Position aus der Ausgang erreicht werden kann. Dabei dürfen keine Mauern durchschritten werden. Insbesondere darf die Position selbst nicht auf einer Mauer liegen.

setPerson Diese Operation setzt eine Person auf einer gegebenen Position im Labyrinth ab. Die Position der Person muss innerhalb des Labyrinths liegen und vom Ausgang erreichbar sein. Ist eine der genannten Bedingungen nicht erfüllt, bleibt das Labyrinth unverändert. Befindet sich bereits eine Person im Labyrinth, so bleibt das Labyrinth ebenfalls unverändert.

up,down,left,right Verschiebt die Position einer im Labyrinth befindlichen Person um ein Feld in die jeweils angegebene Richtung. Dabei dürfen Mauern nicht betreten und das Labyrinth nicht verlassen werden. Wird der Ausgang betreten, so wird die Person aus dem Labyrinth entfernt.

(a) Geben Sie die Sorten und Operator-Signaturen einer Algebra für das oben beschriebene Labyrinth an. *5 Punkte*

(b) Geben Sie die Trägermengen für die Sorten der Algebra an. Achten Sie darauf, dass Ihre Mengen keine ungültigen Werte aufnehmen können. *10 Punkte*

(c) Geben Sie Funktionsdefinitionen für die Operatoren der Algebra an. *10 Punkte*
Da die Operationen *up*, *down*, *left* und *right* doch sehr ähnlich sind, genügt hierbei die Angabe für die Richtung *up*.

Aufgabe 2 AVL-Baum

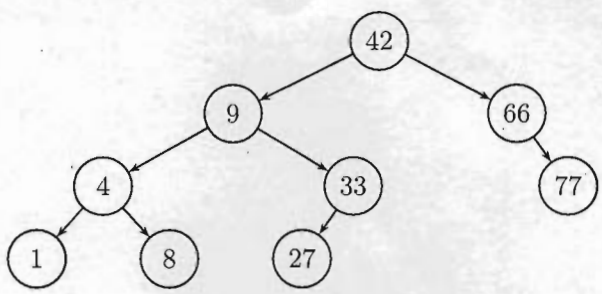
25 Punkte

(a) Erstellen Sie einen AVL -Baum, der die folgenden Elemente enthält: *17 Punkte*

$[1, 6, 3, 16, 42, 20, 12, 2, 9]$

Fügen Sie diese Elemente nacheinander ein. Wird das AVL-Kriterium bei einer Einfügeoperation verletzt, so markieren Sie den Knoten, für den das Kriterium nicht erfüllt ist, und rebalancieren Sie den Baum. Zeichnen Sie den Baum vor und nach jeder Strukturänderung. Geben Sie weiterhin die durchgeführte Rebalancierungsoperation (Linksrotation, Rechtsrotation, Doppelrotation) an.

(b) Gegeben sei der folgende AVL-Baum: *8 Punkte*



Löschen Sie aus dem Baum die Elemente 77, 66, 33 in dieser Reihenfolge. Rebalancieren Sie den Baum unter Angabe der notwendig werdenden Rotationen, falls das AVL-Kriterium durch das Löschen verletzt wird.

Aufgabe 3 Sortieren

27 Punkte

(a) Gegeben sei die Zahlenfolge *15 Punkte*

5, 3, 42, 16, 1, 18, 11, 88, 2, 10

Sortieren Sie diese Folge *absteigend* mittels Standard-Heapsort. Geben Sie den initialen Heap an. Stellen Sie weiterhin den Heap und die sortierte Folge nach jedem Einsinkenlassen eines Elements dar. Sie können den Heap wahlweise im Array oder als Baum darstellen.

(b) Sortieren Sie die folgenden Wörter alphabetisch mittels Radixsort. Geben Sie hierzu die Inhalte der nicht-leeren Behälter nach jeder Phase an. *12 Punkte*

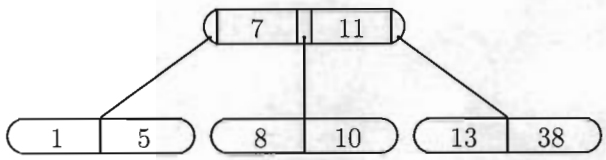
Oracle, Neo, Trinity, Morpheus, Apoc, Niobe, Tank, Nerowinger, Smith, Switch

6 3 7 8 4 5 4 10 5 6

Aufgabe 4 B-Bäume

22 Punkte

Gegeben sei der folgende B-Baum der Ordnung 1.



(a) Fügen Sie die Schlüsselwerte der Folge $\langle 29, 17, 22, 25, 30, 23, 33 \rangle$ in der gegebenen Reihenfolge in den oben abgebildeten B-Baum ein. Zeichnen Sie den Baum vor und nach jeder Strukturänderung. *14 Punkte*

(b) Löschen Sie aus dem oben abgebildeten B-Baum nacheinander die Elemente 8, 7 und 1. Geben Sie den Baum jeweils vor und nach einer Strukturänderung an und benennen Sie die Art der Unterlaufbehandlung. *8 Punkte*

Sollte eine Balance-Operation mit beiden Nachbarknoten möglich sein, so wählen Sie den linken Nachbarn aus.

Aufgabe 5 Deckblatt

1 Punkt

Lesen Sie sich die „Hinweise zur Bearbeitung“ sorgfältig durch. Füllen Sie den dort aufgeführten Anweisungen entsprechend beide Deckblätter vollständig und korrekt aus.

FernUniversität in Hagen

Lösungsvorschläge zur Hauptklausur 1661 „Datenstrukturen I“

19.08.2017

Aufgabe 1 Labyrinth

(a)

algebra *Labyrinth*			
sorts	*labyrinth*, *min4*, *pos*		
ops	*create*	: $min4 \times min4$	$\rightarrow$ *labyrinth*
	setWall	: *labyrinth* $\times$ *pos*	$\rightarrow$ *labyrinth*
	setExit	: *labyrinth* $\times$ *pos*	$\rightarrow$ *labyrinth*
	isReachable	: *labyrinth* $\times$ *pos*	$\rightarrow$ *bool*
	setPerson	: *labyrinth* $\times$ *pos*	$\rightarrow$ *labyrinth*
	up	: *labyrinth*	$\rightarrow$ *labyrinth*
	down	: *labyrinth*	$\rightarrow$ *labyrinth*
	left	: *labyrinth*	$\rightarrow$ *labyrinth*
	right	: *labyrinth*	$\rightarrow$ *labyrinth*

(b)

Die Sorte *min4* verwenden wir ausschließlich beim *create* Operator. Diese Sorte stellt Zahlen dar, die nicht kleiner sind als 4. Die Menge ist wie folgt definiert:

$min4 = \{k \in \mathbb{Z} | k > 3\}$

Eine Position ist ein Paar von Zahlen, die jeweils größer sind als 0:

$pos = \{(x, y) \in \mathbb{Z}^2 | x > 0 \wedge y > 0\}$

Das Labyrinth stellen wir durch eine Menge von Paaren bestehend aus Positionen und der jeweiligen Verwendung des Feldes dar. Wir stellen sicher, dass die Positionen ein Rechteck ohne Lücken bilden. Ein Feld kann entweder unbelegt sein, eine Mauer darstellen, der Ausgang sein oder ein Feld mit einer Person darstellen. Der Ausgang muss auf einem Randfeld des Labyrinths liegen. Der Ausgang und die Person dürfen höchstens ein einziges Mal auftreten.

Wir führen zunächst eine Hilfsmenge *FieldType* ein, die die möglichen Belegungen eines Felds darstellt.

$FieldType = \{\text{Empty, Wall, Exit, Person}\}$

Eine weitere Menge stellt alle möglichen Felder dar.

$Field = pos \times FieldType$

Ein Labyrinth lässt sich dann folgendermaßen beschreiben:

$$
\begin{aligned}
labyrinth = \{L \subset Field | \; & (((x_{max}, y_{max}), F) \in L) \wedge \\
& (((x, y), F_1) \in L : x > 1 \Rightarrow ((x-1, y), F_2) \in L) \wedge \\
& (((x, y), F_1) \in L : y > 1 \Rightarrow ((x, y-1), F_2) \in L) \wedge \\
& (x_{max} > 3) \wedge \\
& (y_{max} > 3) \wedge \\
& (((x, y), F_1) \in L \wedge ((x, y), F_2) \in L \Rightarrow F_1 = F_2) \wedge \\
& (((x, y), Exit) \in L \Rightarrow x = 1 \vee y = 1 \vee x = x_{max} \vee y = y_{max}) \wedge \\
& (|\{((x, y), Exit) \in L\}| < 2) \wedge \\
& (|\{((x, y), Person) \in L\}| < 2)
\end{aligned}
$$

Dabei ist
$x_{max} = max(\{x | ((x, y, K)) \in L\})$ und
$y_{max} = max(\{y | ((x, y, K)) \in L\})$

(c)

Die Funktionen lassen sich auf den Trägermengen wie folgt realisieren:

functions

Die Funktion *create* erzeugt einfach eine Menge von leeren Feldern:

$create(xm, ym) = \{((x, y), Empty) | x \geq 1 \wedge y \geq 1 \wedge x \leq xm \wedge y \leq ym\}$

Die Funktion *setWall* ersetzt ein leeres Feld durch eine Mauer:

$$setWall(L, (x, y)) = \begin{cases} L \setminus \{((x, y), Empty)\} \cup \{((x, y), Wall)\} & \text{falls } ((x, y), Empty) \in L \\ L & \text{sonst} \end{cases}$$

Die Funktion *setExit* ersetzt ein leeres Feld durch einen Ausgang, sofern die gegebene Position auf dem Rand liegt und noch kein Ausgang existiert.

$$setExit(L, (x, y)) = \begin{cases} L & \text{falls } (x \neq 1 \wedge y \neq 1 \wedge x \neq x_{max} \wedge y \neq y_{max}) \vee \\ & \quad ((x, y), Wall) \in L \vee \\ & \quad ((x, y), Person) \in L \vee \\ & \quad \exists((x', y'), Exit) \in L \\ L \setminus \{((x, y), Empty)\} & \\ \quad \cup \{((x, y), Exit)\} & \text{sonst} \end{cases}$$

mit
$x_{max} = max(\{x | ((x, y, K)) \in L\})$ und
$y_{max} = max(\{y | ((x, y, K)) \in L\})$

Ein Feld ist sicher vom Ausgang zu erreichen, wenn es selbst der Ausgang ist. Handelt es sich um eine Mauer oder liegt die Position außerhalb des Felds, so ist das Feld unerreichbar. Für andere Felder kann man rekursiv die vier umliegenden Felder auf Erreichbarkeit testen:

$$isReachable(L, (x, y)) = \begin{cases} false & \text{falls } ((x, y), K) \notin L \\ false & \text{falls } ((x, y), Wall) \in L \\ true & \text{falls } ((x, y), Exit) \in L \\ x > 1 \wedge isReachable(L, (x - 1, y)) \vee & \text{sonst} \\ x < x_{max} \wedge isReachable(L, (x + 1, y)) \vee & \\ y > 1 \wedge isReachable(L, (x, y - 1)) \vee & \\ y < y_{max} \wedge isReachable(L, (x, y + 1)) & \end{cases}$$

Die Funktion *setPerson* ersetzt ein leeres Feld durch ein Personenfeld, sofern die Position erreichbar ist, dieses Feld leer ist und das Labyrinth nicht bereits ein Personenfeld enthält.

$$setPerson(L, (x, y)) = \begin{cases} L \setminus \{((x, y), Empty)\} \cup \{((x, y), Person)\} & \text{falls } ((x, y), Empty) \in L \wedge \\ & \quad isReachable(L, (x, y)) \wedge \\ & \quad \nexists((x', y'), Person) \in L \\ L & \text{sonst} \end{cases}$$

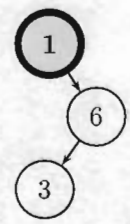
Der Operator *up* verschiebt die Position einer Person nach oben, sofern diese existiert und es sich beim Zielfeld um ein freies Feld handelt. Existiert die Person und das Zielfeld ist der Ausgang, so wird die Person aus dem Labyrinth entfernt. Ansonsten ändert sich nichts am Labyrinth.

$$up(L) = \begin{cases} L \setminus \{((x, y), Person), ((x, y + 1), Empty)\} & \text{falls } ((x, y), Person) \in L \wedge \\ \quad \cup \{((x, y), Empty), ((x, y + 1), Person)\} & \quad ((x, y + 1), Empty) \in L \\ L \setminus \{((x, y), Person)\} \cup \{((x, y), Empty)\} & \text{falls } ((x, y), Person) \in L \\ & \quad \wedge((x, y + 1), Exit) \in L \\ L & \text{sonst} \end{cases}$$

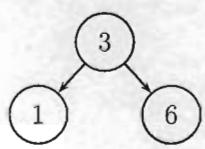
Aufgabe 2 AVL-Baum

(a)

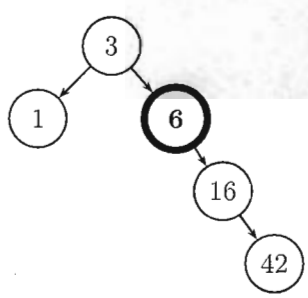
Das Einfügen von 3 führt zu einer Verletzung der Balance am markierten Knoten:



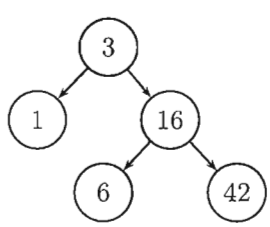
Wir führen eine Doppelrotation durch.
Die Rotation führt zu:



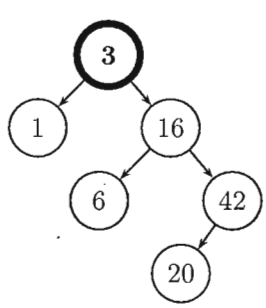
Das Einfügen von 42 führt zu einer Verletzung der Balance am markierten Knoten:



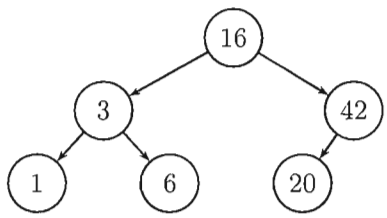
Wir führen eine Linksrotation durch.
Die Rotation führt zu:



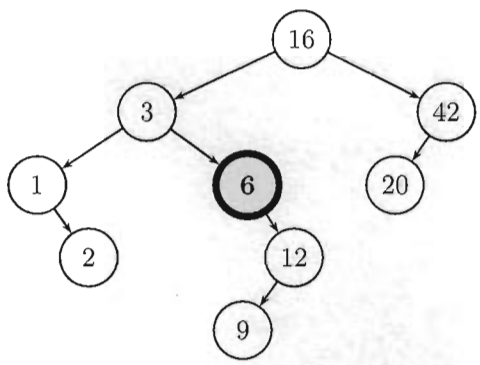
Das Einfügen von 20 führt zu einer Verletzung der Balance am markierten Knoten:



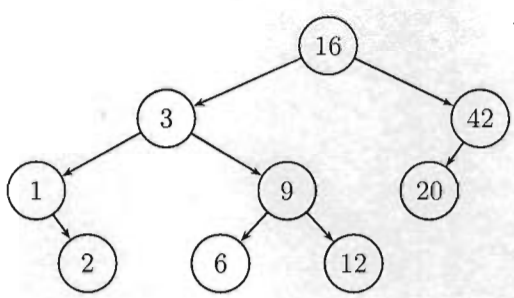
Wir führen eine Linksrotation durch.
Die Rotation führt zu:



Das Einfügen von 9 führt zu einer Verletzung der Balance am markierten Knoten:

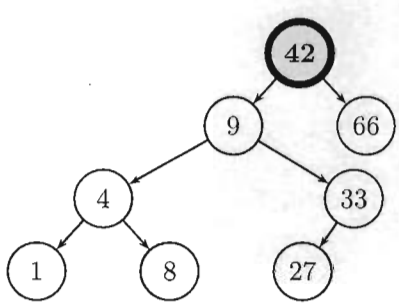


Wir führen eine Doppelrotation durch.
Die Rotation führt zu:

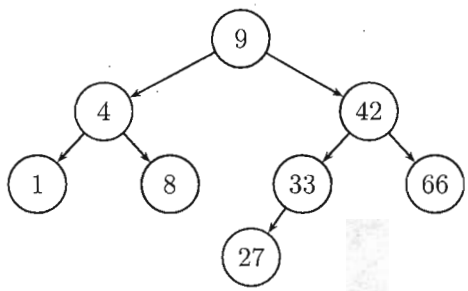


(b)

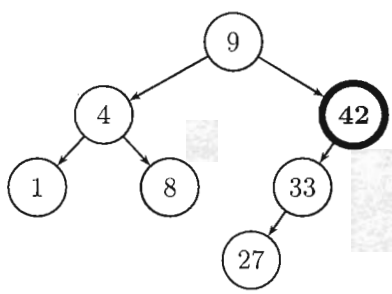
Löschen des Werts : 77
Balance am markieren Knoten verletzt:



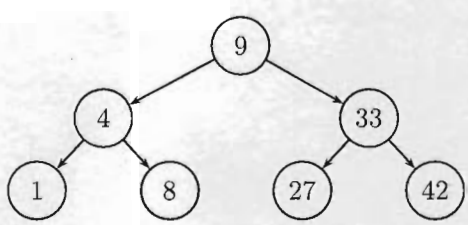
Wir führen eine Rechtsrotation durch.
Baum nach Löschen von 77:



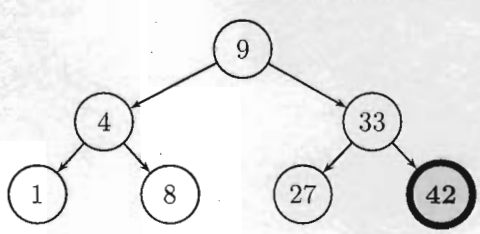
Löschen des Werts : 66
Balance am markieren Knoten verletzt:



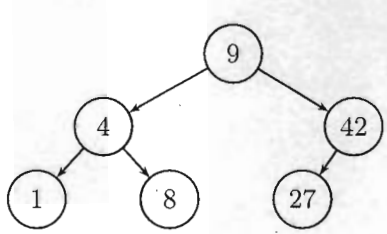
Wir führen eine Rechtsrotation durch.
Baum nach Löschen von 66:



Löschen des Werts : 33
Wir tauschen die zu löschende 33 mit dem Inhalt des markierten Knotens und löschen dann diesen.



Baum nach Löschen von 33:



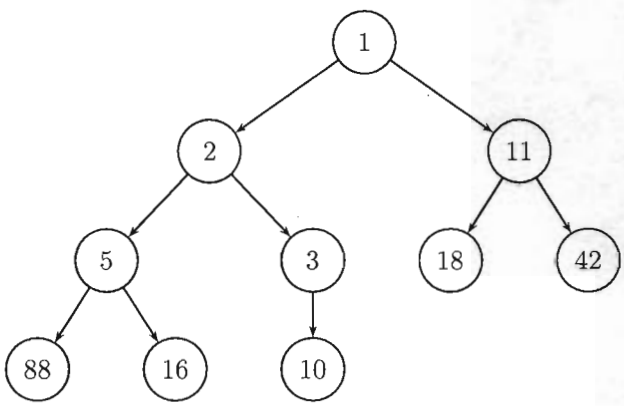
Aufgabe 3 Sortieren

(a)

Heapsort wird gestartet

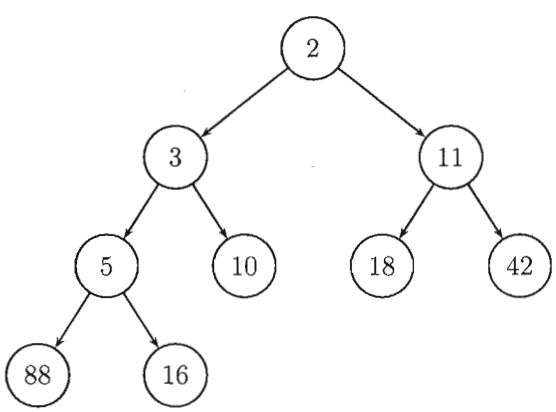
Arrayeinbettung : | 5 | 3 | 42 | 16 | 1 | 18 | 11 | 88 | 2 | 10 |

Initialer Heap erzeugt:



Arrayeinbettung : | 1 | 2 | 11 | 5 | 3 | 18 | 42 | 88 | 16 | 10 |

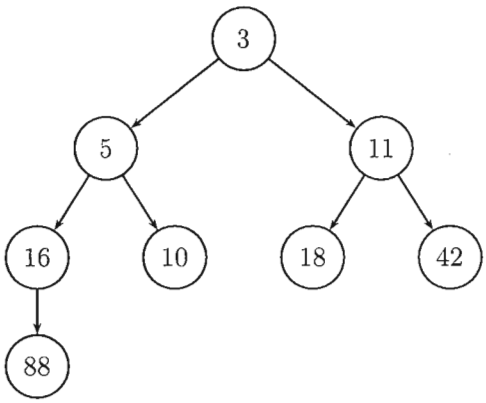
Element 1 verarbeitet:



Sortierte Folge: 1

Arrayeinbettung : | 2 | 3 | 11 | 5 | 10 | 18 | 42 | 88 | 16 ‖ 1 |

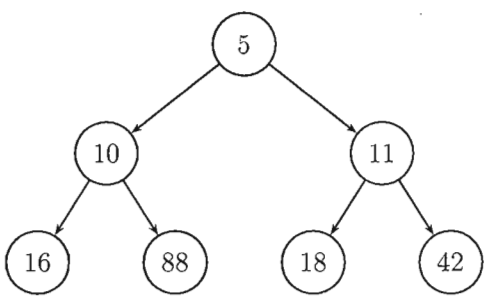
Element 2 verarbeitet:



Sortierte Folge: 2, 1

Arrayeinbettung : | 3 | 5 | 11 | 16 | 10 | 18 | 42 | 88 || 2 | 1 |

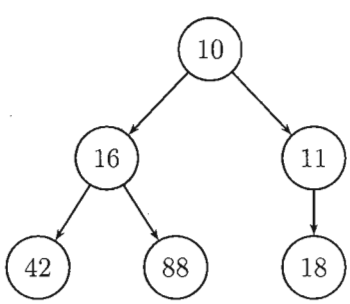
Element 3 verarbeitet:



Sortierte Folge: 3, 2, 1

Arrayeinbettung : | 5 | 10 | 11 | 16 | 88 | 18 | 42 || 3 | 2 | 1 |

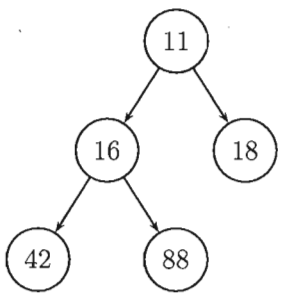
Element 5 verarbeitet:



Sortierte Folge: 5, 3, 2, 1

Arrayeinbettung : | 10 | 16 | 11 | 42 | 88 | 18 || 5 | 3 | 2 | 1 |

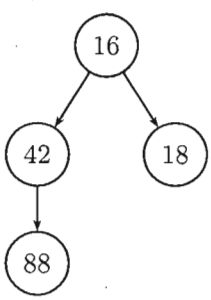
Element 10 verarbeitet:



Sortierte Folge: 10, 5, 3, 2, 1

Arrayeinbettung : | 11 | 16 | 18 | 42 | 88 || 10 | 5 | 3 | 2 | 1 |

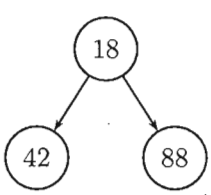
Element 11 verarbeitet:



Sortierte Folge: 11, 10, 5, 3, 2, 1

Arrayeinbettung : | 16 | 42 | 18 | 88 || 11 | 10 | 5 | 3 | 2 | 1 |

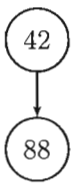
Element 16 verarbeitet:



Sortierte Folge: 16, 11, 10, 5, 3, 2, 1

Arrayeinbettung : | 18 | 42 || 88 || 16 | 11 | 10 | 5 | 3 | 2 | 1 |

Element 18 verarbeitet:



Sortierte Folge: 18, 16, 11, 10, 5, 3, 2, 1

Arrayeinbettung : | 42 | 88 || 18 | 16 | 11 | 10 | 5 | 3 | 2 | 1 |

Element 42 verarbeitet:

Sortierte Folge: 42, 18, 16, 11, 10, 5, 3, 2, 1

Arrayeinbettung : | 88 || 42 | 18 | 16 | 11 | 10 | 5 | 3 | 2 | 1 |

Element 88 verarbeitet:

Sortierte Folge: 88, 42, 18, 16, 11, 10, 5, 3, 2, 1

Arrayeinbettung : || 88 | 42 | 18 | 16 | 11 | 10 | 5 | 3 | 2 | 1 |

(b)

Um Strings korrekt zu sortieren, muss *RadixSort* deren Buchstaben von hinten nach vorn verarbeiten. Der Algorithmus beginnt an der letzten Position des längsten Strings. Es genügt nicht, an der letzten Position des kürzesten Strings anzufangen. Bei einem solchen Vorgehen wäre das Ergebnis für die Folge „aaz aa aaa“ die unveränderte Folge, was offensichtlich falsch ist. Strings, die an der aktuell zu verarbeitenden Position noch nicht existieren, werden in den Behälter $B_\emptyset$ eingefügt.

Initiale Verteilung (Phase 0):

$B_\emptyset$={Oracle, Neo, Trinity, Morpheus, Apoc, Niobe, Tank, Smith, Switch}
B_r={Nerowinger}

Phase: 1

$B_\emptyset$={Oracle, Neo, Trinity, Morpheus, Apoc, Niobe, Tank, Smith, Switch}
B_e={Nerowinger}

Phase: 2

$B_\emptyset$={Oracle, Neo, Trinity, Apoc, Niobe, Tank, Smith, Switch}
B_g={Nerowinger}
B_s={Morpheus}

Phase: 3

$B_{\emptyset}$={Oracle, Neo, Apoc, Niobe, Tank, Smith, Switch}
B_n={Nerowinger}
B_u={Morpheus}
B_y={Trinity}

Phase: 4

$B_{\emptyset}$={Neo, Apoc, Niobe, Tank, Smith}
B_e={Oracle, Morpheus}
B_h={Switch}
B_i={Nerowinger}
B_t={Trinity}

Phase: 5

$B_{\emptyset}$={Neo, Apoc, Tank}
B_c={Switch}
B_e={Niobe}
B_h={Smith, Morpheus}
B_i={Trinity}
B_l={Oracle}
B_w={Nerowinger}

Phase: 6

$B_{\emptyset}$={Neo}
B_b={Niobe}
B_c={Apoc, Oracle}
B_k={Tank}
B_n={Trinity}
B_o={Nerowinger}
B_p={Morpheus}
B_t={Switch, Smith}

Phase: 7

B_a={Oracle}
B_i={Trinity, Switch, Smith}
B_n={Tank}
B_o={Neo, Niobe, Apoc}
B_r={Nerowinger, Morpheus}

Phase: 8

B_a={Tank}
B_e={Neo, Nerowinger}
B_i={Niobe}
B_m={Smith}
B_o={Morpheus}
B_p={Apoc}
B_r={Oracle, Trinity}
B_w={Switch}

Phase: 9

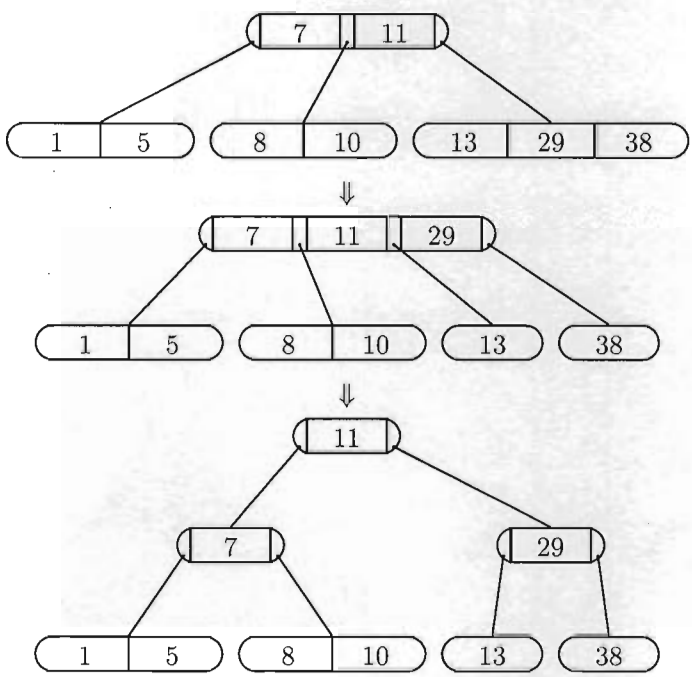
B_a ={Apoc}
B_m={Morpheus}
B_n ={Neo, Nerowinger, Niobe}
B_o ={Oracle}
B_s ={Smith, Switch}
B_t ={Tank, Trinity}

Die sortierte Folge lautet:
Apoc Morpheus Neo Nerowinger Niobe Oracle Smith Switch Tank Trinity

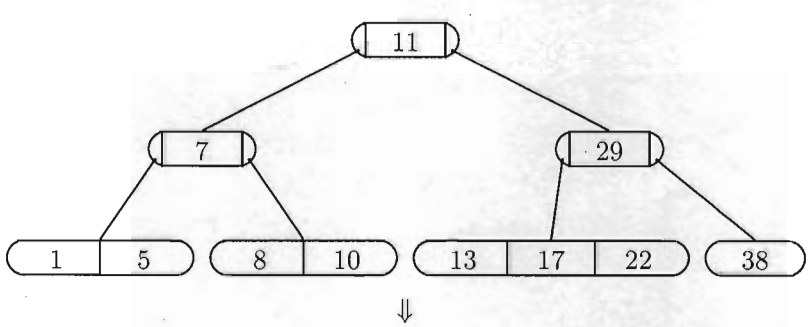
Aufgabe 4 B-Bäume

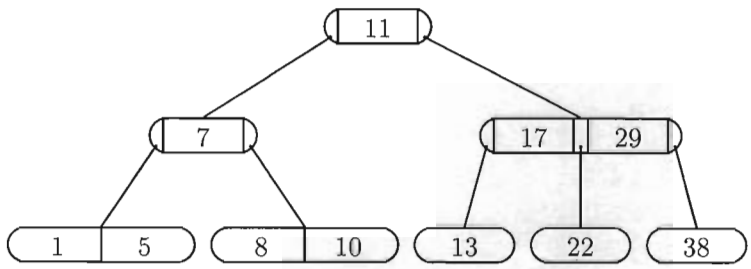
(a)

Nach dem Einfügen des Elements 29 sind zwei Splits nacheinander erforderlich.

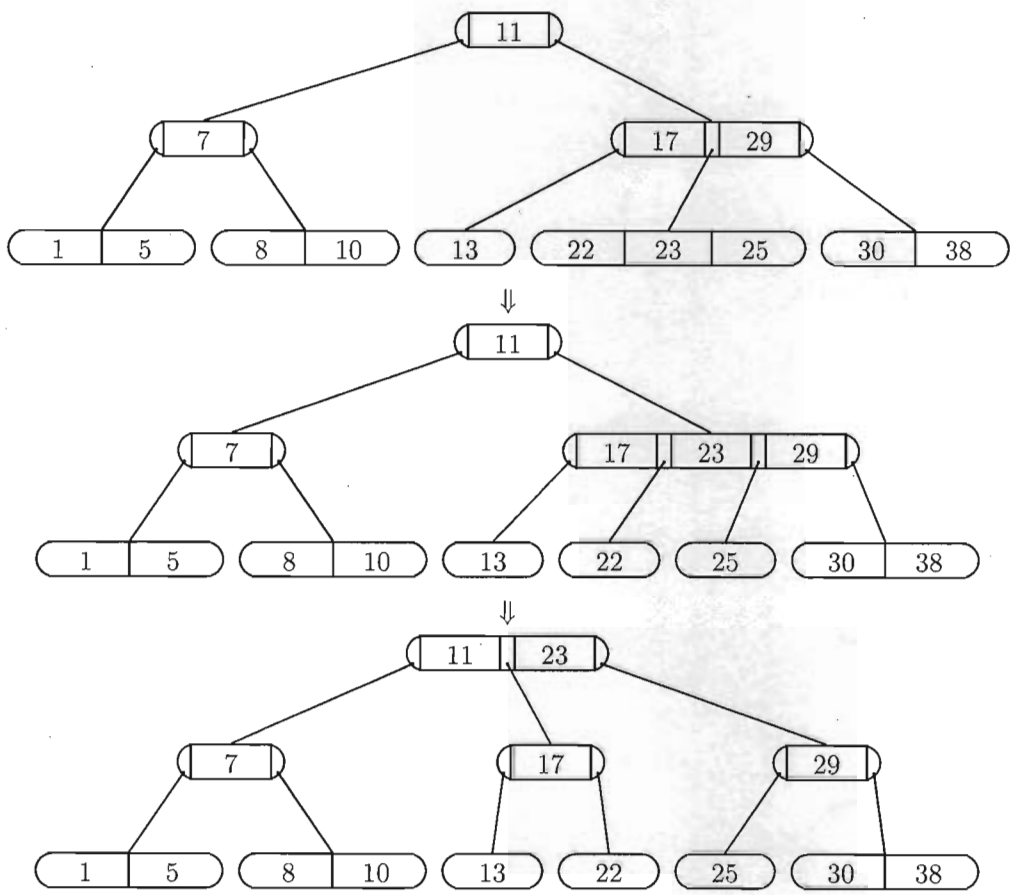


Nach dem Einfügen des Elements 22 ist ein Split erforderlich.

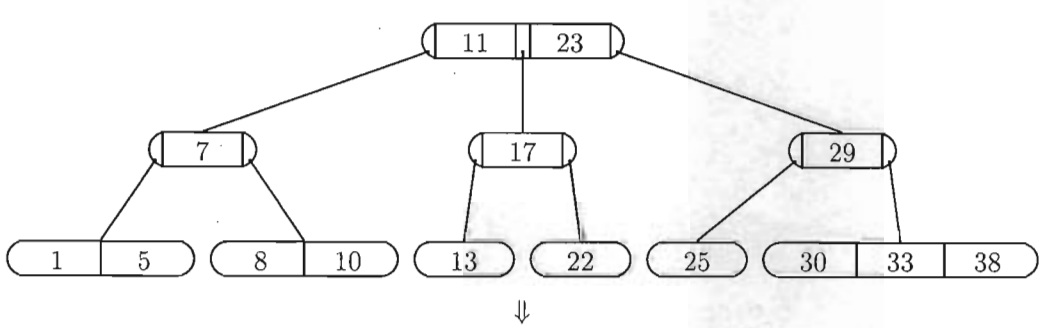


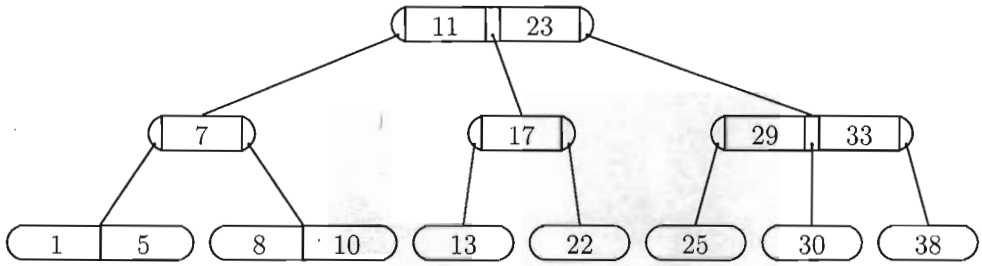


Nach dem Einfügen des Elements 23 sind zwei Splits nacheinander erforderlich.



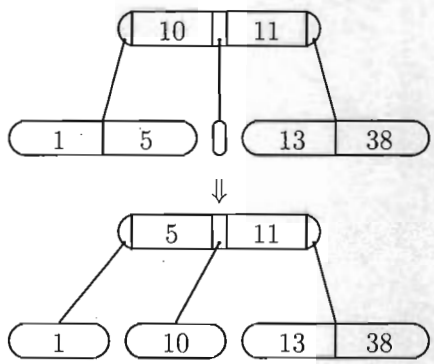
Nach dem Einfügen des Elements 33 ist ein Split erforderlich.



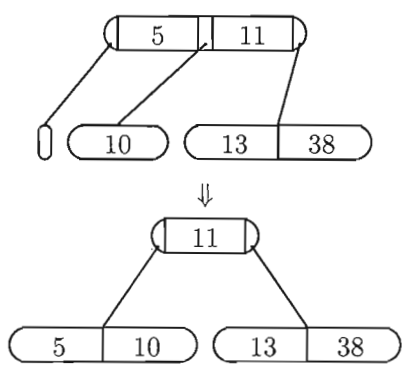


(b)

Nach dem Löschen des Elements 7 ist eine Balance-Operation erforderlich.



Nach dem Löschen des Elements 1 ist eine Merge-Operation erforderlich.



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.