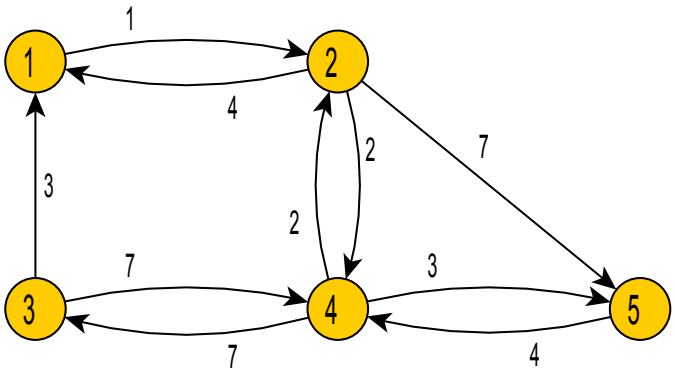
Aufgaben zur Veranstaltung
Algorithmen und Datenstrukturen, SS 2022

H. Pflug, J. Dietel FH Aachen, Campus Jülich; IT Center, RWTH Aachen

Klausurvorbereitung, Blatt 4 29.06.2022

Aufgabe 1



Gegeben sei der obenstehende Graph.

a) Wenden Sie auf den Graphen den Dijkstra-Algorithmus an, um den günstigsten Weg von Knoten 3 zu allen anderen Knoten zu finden.

		d					P			
V_i		1	2	4	5		1	2	4	5
3		(3)		7			3		3	
1			(4)	7				1		
2				(6)	11				2	2
4					(9)					4
		3	4	6	9		3	1	2	4

b) Bejahen oder verneinen Sie die folgenden Aussagen:

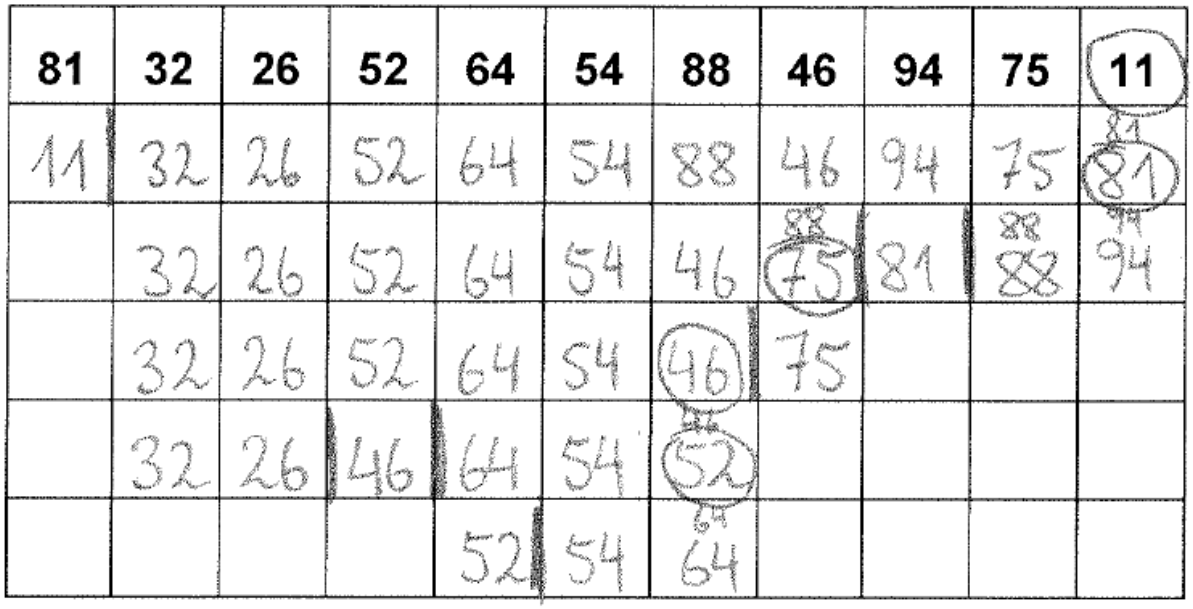
	ja	nein
Der Graph ist gerichtet.	x	
Der Graph ist gewichtet.	x	
Dieser Graph kann als ein Baum aufgefasst werden.		x

Aufgabe 2

Gegeben ist ein Feld von 11 unsortierten Integer-Werten:

81,32,26,52,64,54,88,46,94,75,11

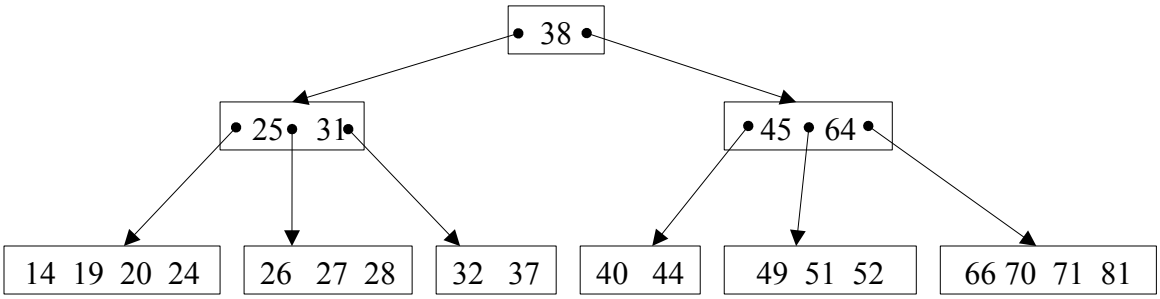
Sortieren Sie die Werte nach dem Quicksort-Verfahren in aufsteigender Reihenfolge (kleinstes Element links). Kennzeichnen Sie die Pivot-Elemente und dokumentieren Sie die Vertauschungen. Verwenden Sie als Pivot-Element das mittlere Element einer Teilliste, bzw. bei einer geraden Anzahl von Elementen, das rechte der beiden mittleren.
Für Teilfelder von 2 Elementen brauchen Sie den Quicksort-Algorithmus nicht mehr zu verwenden, sondern dürfen die Elemente gegebenenfalls einfach umtauschen.



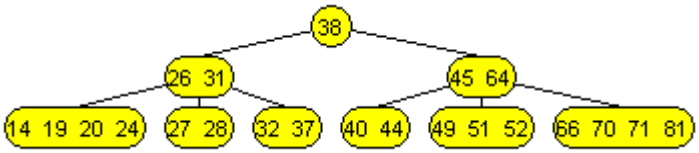
Aufgabe 3

Führen Sie die unter a) bis d) beschriebenen Einfüge- bzw. Löschoperationen aus und zeichnen Sie den resultierenden Baum auf.

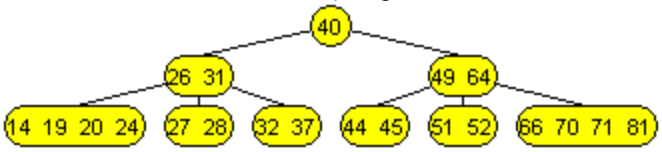
a) Gegeben sei folgender B-Baum der Ordnung **2**:



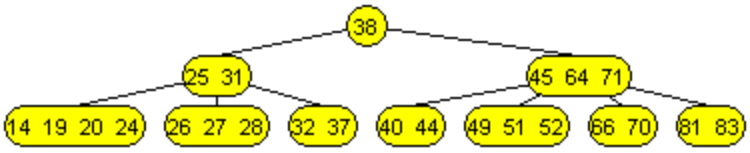
Löschen Sie die **25**.



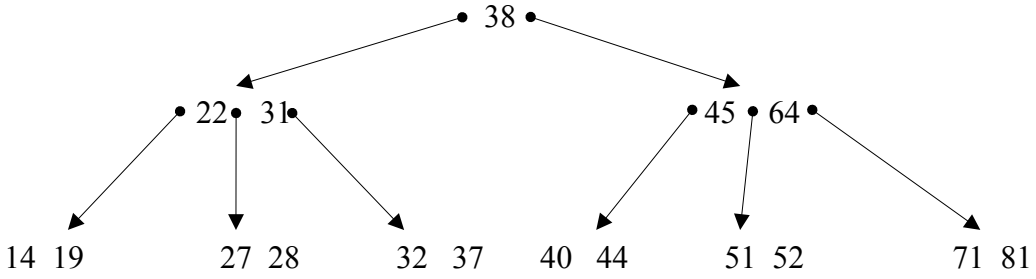
b) Löschen Sie aus dem unter a) abgebildeten Baum die **38**.



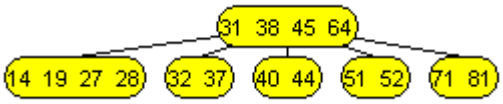
c) Fügen Sie in den unter a) abgebildeten Baum die **83** ein.



d) Gegeben sei folgender B-Baum der Ordnung **2**:



Löschen Sie die **22**.



Aufgabe 4

Suchen Sie in dem Text ACABLACDABLABLADCACDCAD

das Muster ACDC

mit der in der Vorlesung behandelten Variante des Boyer-Moore-Verfahrens.

a) Geben Sie die Belegung des last-Feldes an.

A	C	D	Sonstige
0	3	2	-1

b) Tragen Sie die wesentlichen Schritte des Suchvorgangs in folgendes Diagramm ein:

A	C	A	B	L	A	C	D	A	B	L	A	B	L	A	D	C	A	C	D	C	A	D
A	C	D	C																			
					A	C	D	C														
										A	C	D	C									
														A	C	D	C					
															A	C	D	C				
																A	C	D	C			

Aufgabe 5:

Gegeben seien die folgenden beiden Hashfunktionen für eine Hashtabelle der Größe *m*:

$h_0(n) = n \bmod m$
$h_i(n) = (h_0(n)+i \cdot q(n)) \bmod m$ mit $i \geq 1$

wobei *q(n)* die Quersumme der Zahl *n* bedeutet und *i* die laufende Nummer des Ersatzwerts ist. Mit anderen Worten: Sollte ein angegebener Speicherort belegt sein, wird der nächste Ersatzwert durch ein Inkrement um *q(n)* ermittelt, wobei die Hashtabelle als ringförmig angesehen wird.

Wenden Sie dieses Hashverfahren auf die folgenden Zahlen an, die in einer Hashtabelle der Größe *m=10* einsortiert werden sollen:

17, 24, 7, 94, 34, 35, 26

Geben Sie in jedem Schritt die aktuelle Hashtabelle vollständig an.

	0
	1
	2
	3
	4
	5
	6
17	7
	8
	9

	0
	1
	2
	3
24	4
	5
	6
17	7
	8
	9

	0
7	1
	2
	3
24	4
	5
	6
17	7
	8
	9

94	0
7	1
	2
	3
24	4
	5
	6
17	7
	8
	9

94	0
7	1
	2
	3
24	4
	5
	6
17	7
34	8
	9

94	0
7	1
	2
	3
24	4
35	5
	6
17	7
34	8
	9

94	0
7	1
	2
	3
24	4
35	5
26	6
17	7
34	8
	9

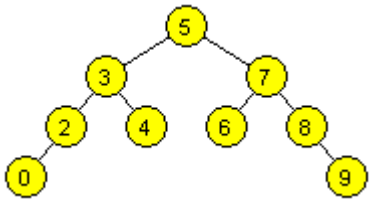
Berechnung der Hashwerte:
17: 7
24: 4
7: 7 → 4 → 1
94: 4 → 7 → 0
34: 4 → 1 → 8
35: 5
26: 6

Aufgabe 6

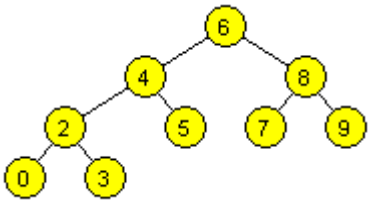
Sie haben folgende Schlüssel:

5, 3, 2, 7, 8, 6, 4, 9, 0

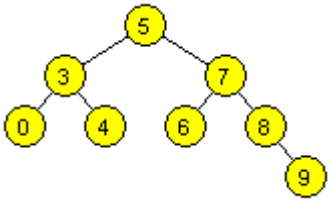
a) Nehmen Sie die 5 als Wurzelelement eines binären Suchbaumes und fügen Sie die restlichen Schlüssel in der obenstehenden Reihenfolge von links nach rechts ein. Zeichnen Sie den entstandenen Baum auf.

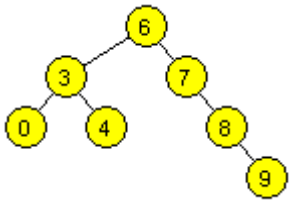


b) Zeichnen Sie einen linksvollständigen Suchbaum mit den gleichen Elementen.



c) Entfernen Sie aus dem Suchbaum aus a) nacheinander die Elemente 2 und 5. Geben Sie nach jedem Entfernen eines Elements den neuen Suchbaum an.





Aufgabe 7

Implementieren Sie die Funktion

```
public static int[][] adjMatrix(int[][] adjListe),
```

die die Adjazenzliste eines Graphen übergeben bekommt, daraus eine Adjazenzmatrix erzeugt und sie zurückgibt.
Die Adjazenzliste wird als zweidimensionales Feld übergeben. Dieses Feld ist ausgefranst (jagged). Jede Zeile hat genau die Länge, die der Anzahl der Nachfolger des entsprechenden Knotens entspricht. Hat ein Knoten keinen Nachfolger, kann die entsprechende Zeile ein Feld der Länge 0 sein oder den Wert null haben.
Gehen Sie davon aus, dass das übergebene Feld diesen Spezifikationen entspricht. Sie müssen nicht überprüft werden.
Die zurückzugebende Adjazenzmatrix soll folgende Eigenschaften haben:

- Falls zwischen zwei Knoten *a* und *b* keine direkte Verbindung besteht, ist der entsprechende Eintrag gleich -1.
- Falls eine direkte Verbindung besteht, ist der entsprechende Eintrag gleich 1.
- Diagonalelemente sind gleich 0.

```
/*
* Aufgabe 7
*/
public static int[][] adjMatrix(int[][] adjListe) {
   int[][] ret = new int[adjListe.length][adjListe.length];
   for (int i=0; i<ret.length; i++) {
         for (int j=0; j<ret.length; j++) {
               if (i!=j) {
                     ret[i][j]=-1;
               }
         }
   }
   for (int i=0; i<adjListe.length; i++) {
         if (adjListe[i]!=null) {
               for (int j=0; j<adjListe[i].length; j++) {
                     ret[i][adjListe[i][j]-1]=1;
               }
         }
   }
   return ret;
}
```

Aufgabe 8

Schreiben Sie eine Klasse `IntMenge`, die eine Menge von Integer-Zahlen repräsentiert. Implementieren Sie die Menge als binären Suchbaum. Schreiben Sie den Suchbaum selbst und benutzen Sie keine Tree-Klassen aus der Java-Klassenbibliothek. Die Klasse `IntMenge` soll folgende Methoden besitzen:

`public void insert (int n)`
Fügt den Wert *n* in den Baum ein. Falls der Wert schon vorhanden ist, wird eine (selbst zu schreibende) `DoubleElementException` ausgelöst.

`public ArrayList<Integer> getElements()`
Gibt die Elemente der Menge in aufsteigender Reihenfolge zurück. Gehen Sie dazu den Baum in In-Order-Reihenfolge durch.

```
public static class IntMenge {
	private Node root;
	public void insert(int n) {
		if (root == null) {
			root = new Node(n);
		} else {
			insert(root, n);
		}
	}
	public void insert(Node node, int x) {
		if (node.value==x) {
			throw new DoubleElementException(x+" ist doppelt");
		}
		if (x<node.value) {
			if (node.left==null) {
				node.left=new Node(x);
			} else {
				insert(node.left,x);
			}
		} else {
			if (node.right==null) {
				node.right=new Node(x);
			} else {
				insert(node.right,x);
			}
		}
	}
	public ArrayList<Integer> getElements() {
		ArrayList<Integer> ret = new ArrayList<Integer>();
		getElements(root, ret);
		return ret;
	}
	public void getElements(Node node, ArrayList<Integer> list) {
		if (node==null) {
			return;
		}
		getElements(node.left, list);
		list.add(node.value);
		getElements(node.right, list);
	}
}

public static class Node {
	public int value;
	public Node left;
	public Node right;
	public Node(int value) {
		this.value = value;
	}
}
public static class DoubleElementException extends RuntimeException {
	public DoubleElementException(String s) {
		super(s);
	}
}
```

Aufgabe 9

Ein Zahlenspiel geht folgendermaßen: Ausgehend von der Startzahl 1 soll eine Ganzzahl *n* (*n>1)* mit möglichst wenigen Rechenschritten erreicht werden. Als Rechenschritte sind dabei erlaubt:

1. Hochzählen um 1
2. Multiplikation mit 2
3. Multiplikation mit 3

Schreiben Sie eine Funktion

```
public static String getShortestWay(int n)
```

die für die Zahl *n* die kürzeste Folge von Rechenschritten zurückgibt. Dabei wird die Folge in einem String zurückgegeben, der aus einer Folge der Zeichen „1“, „2“ und „3“ besteht, wobei jedes Zeichen für die entsprechende Operation steht.

Beispiel:
`getShortestWay(72)`
gibt den String 12233 zurück, denn der kürzeste Weg, von 1 zu 72 zu gelangen ist:
1+1=2; 2*2=4; 4*2=8; 8*3=24;24*3=72.

Tipps:
Falls es mehrere gleich kurze Wege gibt (im Beispiel wäre 22233 gleich kurz), geben Sie eine dieser Möglichkeiten zurück.
Verwenden Sie einen Backtracking-Algorithmus.
Berücksichtigen Sie, dass die Folge der Zwischenergebnisse streng monoton steigend ist. Das heißt, es gibt keine Möglichkeit, von einer Zahl *x>n* auf die Zahl *n* zu kommen.

```
public static String getShortestWay(int n) {
	return getShortestWay(1, n);
}
private static String getShortestWay(int val, int n) {
	if (val==n) {
		return "";
	}
	String x = "1"+getShortestWay(val+1,n);
	if (val*2<=n) {
		String y = "2"+getShortestWay(val*2,n);
		if (y.length()<x.length()) {
			x=y;
		}
	}
	if (val*3<=n) {
		String y = "3"+getShortestWay(val*3,n);
		if (y.length()<x.length()) {
			x=y;
		}
	}
	return x;
}
```