Fakultät IV
Elektrotechnik/Informatik

Einführung in die Informatik II
Probeklausur

Musterlösung
Stand: 12. Juli 2011

Aufgabe 1 Boolsche Algebra.

1. **Teilaufgabe:** Sind die folgenden Booleschen Ausdrücke äquivalent?

$$f(x,y) = (x \Rightarrow y) \vee (y \Rightarrow x)$$
$$g(x,y) = (x \vee y) \Rightarrow (x \wedge y).$$

Beweisen oder widerlegen Sie die Behauptung mit der Wahrheitstafelmethode. Die Zwischenschritte müssen erkennbar sein.

Hinweis: Es gilt $x \Rightarrow y := \neg x \vee y$.

Lösung:

x	y	(x	⇒	y)	∨	(y	⇒	x)	(x	∨	y)	⇒	(x	∧	y)
0	0	0	1	0	1	0	1	0	0	0	0	1	0	0	0
0	1	0	1	1	1	1	0	0	0	1	1	0	0	0	1
1	0	1	0	0	1	0	1	1	1	1	0	0	1	0	0
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Wie man sieht, sind die Ausdrücke *nicht* äquivalent.

2. **Teilaufgabe:** Wandeln Sie den Booleschen Ausdruck der Funktion:

$$f(x,y,z) = \neg(x \equiv y) \vee z$$

mit Hilfe der algebraischen Umformung in eine ausgezeichnete konjunktive Normalform um. Die Zwischenschritte müssen erkennbar sein.

Hinweis: Es gilt $x \equiv y := (x \wedge y) \vee (\neg x \wedge \neg y)$.

Lösung:

$$\begin{aligned} f(x,y,z) &= \neg(x \equiv y) \vee z \\ &= \neg((x \wedge y) \vee (\neg x \wedge \neg y)) \vee z \\ &= (\neg(x \wedge y) \wedge \neg(\neg x \wedge \neg y)) \vee z \\ &= ((\neg y \vee \neg y) \wedge (x \vee y)) \vee z \\ &= (\neg x \vee \neg y \vee z) \wedge (x \vee y \vee z) \end{aligned}$$

3. **Teilaufgabe:** Wandeln Sie den Booleschen Ausdruck der Funktion:

$$f(x,y,z,w) = (((x \wedge y) \vee x \vee (z \vee w)) \wedge (w \vee z)$$

mit Hilfe der KV-Tafeln in eine ausgezeichnete disjunktive Normalform um. Die Zwischenschritte müssen erkennbar sein.

Lösung:

Für: $f(x,y,z,w) = ((x \wedge y) \vee x \vee (z \vee w)) \wedge (w \vee z)$ ergibt sich folgendes KV-Diagramm:

$f(x, y, z, w)$:

w \ z \ x \ y	0	0	0	0
	1	1	1	1
	1	1	1	1
	1	1	1	1

Vereinfacht ergibt sich:

DNF: $f(x, y, z, w) = (w \vee z)$

Aufgabe 2 Komplexität.

- Bestimmen Sie eine Formel für den Aufwand der folgenden Methode $g(n)$. Dabei soll für die Berechnung des Zeitaufwands nur in Zeile 7 der Funktionsaufruf `fun(i)` berücksichtigt werden. Die Funktion `fun(i)` weist einen Aufwand von $T_{fun}(1)$ auf dieser muss noch zu dem Aufwand der Funktion $g(n)$ hinzugerechnet werden.

```
1  public void g(int n) {
2    int i = 0;
3    int j = 0;
4    while (i < n) {
5      j = i;
6      while (j == i ) {
7        fun(n);
8        j++;
9      }
10     i++;
11   }
12 }
```

- Zeigen Sie in welcher Komplexitätsklasse sich diese Methode befindet und markieren Sie in der folgenden Tabelle die Komplexitätsklassen zu denen die Methode gehört.

Lösung:

$T(n) = \sum_{i=0}^{n-1} 1 \cdot T_{fun}(1) = n \cdot T_{fun}(1)$
$\Rightarrow T(n) = n$

Es gilt:

$$g \in \mathcal{O}(f) \quad \Leftrightarrow \quad \exists\, c > 0 \quad \exists\, n_0 \in N: \quad g(n) \leq c \cdot f(n) \quad \forall\, n \geq n_0$$

Beweis, dass $g(n) \in \mathcal{O}(n)$

$T(n) = g(n) = n \leq c \cdot n = f(n)$
Wähle $c = 1, n_0 = 1$, dann ist $g(n) = n \leq 1 \cdot n$ für alle $n \geq 1$.

Beweis, dass $g(n) \notin \mathcal{O}(\log(n))$

Seien $c > 0, n_0 \in N$ beliebig.
$T(n) = g(n) = n \cdot (\frac{\log(n)}{\log(n)}) = \log(n) \cdot (\frac{n}{\log(n)}) \geq c \cdot \log(n)$,
da $\lim_{n\to\infty} (\frac{n}{\log(n)}) = \infty > c$

Es existiert kein $c \geq \frac{n}{\log(n)}$ für $n \geq n_0$.

Lösung:

Ordnung	fällt in diese Klasse
$\mathcal{O}(1)$	
$\mathcal{O}(\log(n))$	
$\mathcal{O}(n)$	×
$\mathcal{O}(n \log(n))$	×
$\mathcal{O}(n^2)$	×
$\mathcal{O}(n^2 \log(n))$	×
$\mathcal{O}(n^3)$	×
$\mathcal{O}(n^p)$ mit $p > 3$	×
$\mathcal{O}(p^n)$	×

Aufgabe 3 Heapsort.

1. **Teilaufgabe:** Gegeben sei die Zahlenfolge

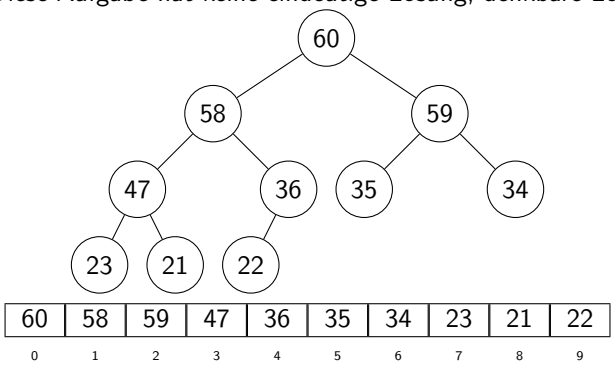
   $$F_1 = 47, 21, 35, 34, 59, 22, 36, 58, 60, 23$$

   Geben sie einen Binärbaum an, der die Elemente der Folge F_1 enthält und die Heap Eigenschaft erfüllt. Geben Sie alternativ dazu den zum Heap gehörigen Array an.

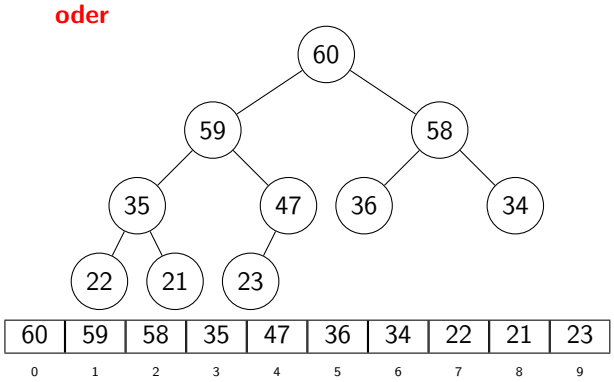
   Lösung:
   Wir lösen diese Aufgabe unter Beachtung der Heapbedingung für Binärheaps. Sie lautet für Maxheaps:

   $$\text{B ist Kindknoten von A} \Leftrightarrow \text{Key(A)} \geq \text{Key(B)}$$

   Wir suchen uns also die größte Zahl der Zahlenfolge F_1 (hier: 60) als erstes Element und bauen den Heap dann von oben nach unten (und dann in den einzelnen Ebenen von links nach rechts) auf. Gleichzeitig füllen wir das Array auf. Das erste Element hat den Index 0. Wenn wir immer die nächstkleinere noch nicht benutzte Zahl der Zahlenfolge F_1 als nächstes Element fortlaufend in das Array (bzw. in den Heap) einfügen, wird die Heapbedingung nie verletzt. Es ist außerdem darauf zu achten, dass der fertige Heap ein linksvoller Baum sein sollte, damit die Arraydarstellung auch Sinn macht.
   Diese Aufgabe hat keine eindeutige Lösung, denkbare Lösungen sind aber z.B:



oder

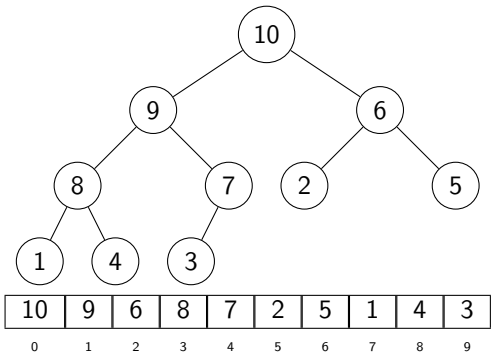


2. **Teilaufgabe:** Gegeben sei die Zahlenfolge

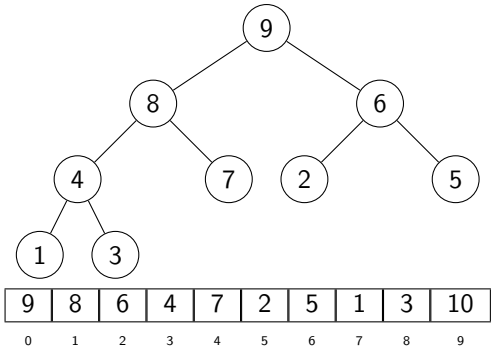
   $$F_2 = 10, 9, 6, 8, 7, 2, 5, 1, 4, 3.$$

   Sortieren Sie die Folge F_2 mit Heapsort. Stellen Sie nach jeder Iteration den Restheap als Baum und die gesamte Zahlenfolge als Array dar.
   Hinweis: Die Zahlenfolge F_2 ist ein Heap.

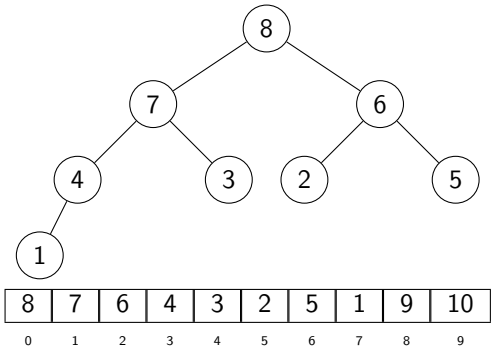
Lösung:
Heapsort kann nur sinnvoll auf einen Heap angewendet werden. Der Hinweis versichert, dass F_2 einer ist. Der Heapsort-Algorithmus ist ausführlich in Tutoriumsvorbereitung # 8 zu finden.
Ausgangsheap:



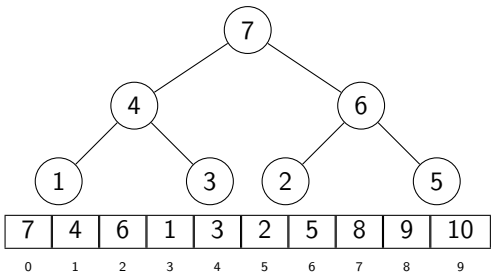
Tausche 3 mit 10 und führe Sift-Down auf die neue Wurzel durch (tausche sie also mit 9,8,4).



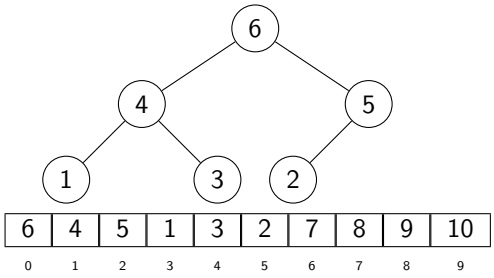
Tausche 3 mit 9 und führe Sift-Down auf die neue Wurzel durch (tausche sie also mit 8,7).



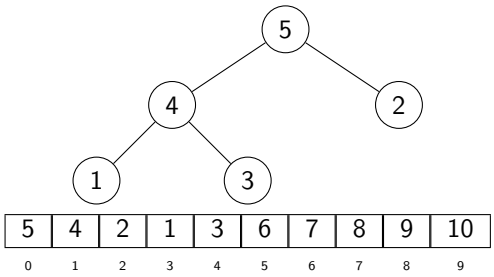
Tausche 1 mit 8 und führe Sift-Down auf die neue Wurzel durch (tausche sie also mit 7,4).



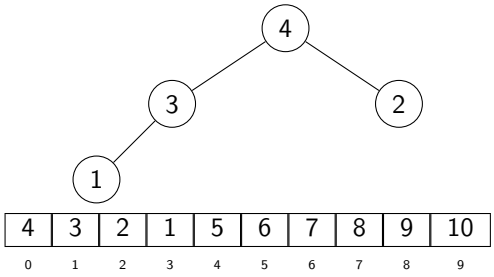
Tausche 5 mit 7 und führe Sift-Down auf die neue Wurzel durch (tausche sie also mit 6).



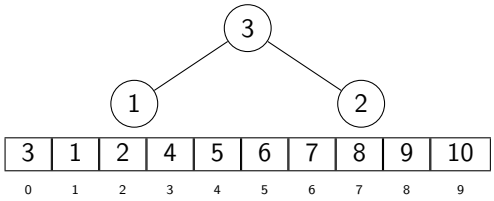
Tausche 2 mit 6 und führe Sift-Down auf die neue Wurzel durch (tausche sie also mit 5).



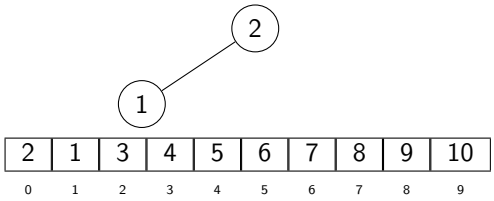
Tausche 3 mit 5 und führe Sift-Down auf die neue Wurzel durch (tausche sie also mit 4).



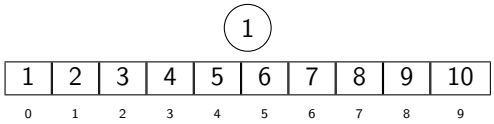
Tausche 1 mit 4 und führe Sift-Down auf die neue Wurzel durch (tausche sie also mit 3).



Tausche 2 mit 3 und führe Sift-Down auf die neue Wurzel durch (kein Tausch notwendig).



Tausche 1 mit 2 und führe Sift-Down auf die neue Wurzel durch (kein Tausch notwendig).



Fertig!

Aufgabe 4 Breitensuche.

Betrachten Sie den folgenden Graphen G:

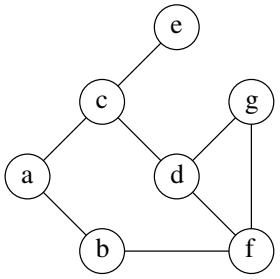


Abbildung 1: Graph G.

1. **Teilaufgabe:** Welchen abstrakten Datentyp verwendet die Breitensuche? Wie lautet das Speicherprinzip dieses Datentyps?

   Lösung:
   Queue, FIFO (first in, first out)

2. **Teilaufgabe:** Traversieren Sie den Graphen G mit Breitensuche. Führen Sie dazu eine Handsimulation mit Hilfe der untenstehenden Tabelle durch. Dabei bezeichne t die Anzahl des aktuellen Schleifendurchlaufs und EK den aktuellen zu expandierenden Knoten.

   Befolgen Sie bei der Simulation folgende Regeln:

   - Startknoten ist der Knoten mit Bezeichner a.
   - Geben Sie für $t = 0$ den Zustand der Queue unmittelbar nach Initialisierung an.
   - Geben Sie für $t > 0$ den Zustand der Queue jeweils am Ende des aktuellen Schleifendurchlaufs an.
   - Fügen Sie die weißen Nachfolger von EK stets in *alphabetisch aufsteigender* Reihenfolge in die Queue ein.

   Lösung:

t	EK	Queue
0	0	a
1	a	b,c
2	b	c,f
3	c	f,d,e
4	f	d,e,g
5	d	e,g
6	e	g
7	g	-

Aufgabe 5 Mergesort.

Implementieren Sie eine Java-Methode `public void mergesort(int[] arr)` die das Sortierverfahren Mergesort für Integer realisiert und gegebenfalls benötigte Hilfsmethoden.

Lösung:

```
1
2       public void mergesort(int [] arr) {
3           divide(arr, 0, arr.length - 1);
4       }
5
6       /*
7        Methode divide: "Teilt" das Array in Teilarrays durch rekursiven Aufruf
8        @param arr Array der zu sortierenden Elemente
9        @param start Startindex
10       @param end Endindex
11       */
12      private static void divide(int[] arr, int start, int end) {
13          if (start < end) {
14              int mid = (start + end + 1) / 2; //Mitten-Element wird bestimmt
15              divide(arr, start, mid - 1);
16              divide(arr, mid, end);
17              merge(arr, start, end, mid);
18          }
19      }
20
21      /*Methode merge: Fügt zwei sortierte Arrays zusammen
22      @param arr: zweigeteiltes Array (Teilung durch Teilungsindex realisiert )
23      @param start: Startindex der ersten Menge
24      @param end: Endindex der zweiten Menge
25      @param part: Der Teilungsindex der beiden Sets*/
26
27      private static void merge(int [] arr, int start, int end, int part) {
28          if( start < end ){
29              int i, j, k;
30              i = 0; //Variable für Indizes im Hilfsarray b
31              j = start; //Variable für Indizes im Array arr
32
33              int [] b = new int[part - start]; //Neues Array mit der Laenge des 1.Sets wird
                    angelegt
34              while (j < part){ //Kopiere das erste Set in Hilfsarray b um Ueberschreiben zu
                    vermeiden
35                  b[i] = arr[j];
36                  j++;
37                  i++;
38              }
39             //j = part, also nun Variable auf das 2.Set von arr
40              i = 0; //weiterhin Variable für Indizes im Hilfsarray b
41              k = start; //Variable der von start ab durchzählt (Index in dem das kleinere Element aus
                     den ursprünglich beiden Sets gespeichert wird)
42              //Zurueckkopieren der Elemente, die kleiner sind
43              while (k < j && j <= end){
44                  if (b[i]< arr[j]){
45                      arr[k++] = b[i++];
46                  }
```

```
47
48                  else {
49                      arr[k++] = arr[j++];
50                  }
51              }
52
53              //Zurueckkopieren des ueberbleibenden Rests
54              while (k < j){
55                  arr[k++] = b[i++];
56              }
57          }
58      }
```

Aufgabe 6 Generische Klassen.

1. **Teilaufgabe:** Es seien $\mathcal{X}$ und $\mathcal{Y}$ zwei Mengen beliebigen Typs. Implementieren Sie ein generisches Interface `Function`, das eine Funktion $f : \mathcal{X} \to \mathcal{Y}$ repräsentiert. Dabei sollen die Mengen $\mathcal{X}$ und $\mathcal{Y}$ generisch verschieden Datentypen angehören können. Ausserdem fordert das Interface eine Methode `apply`, die bein EIngabe eines Elementes $x \in \mathcal{X}$ den Funktionswert $y = f(x) \in \mathcal{Y}$ zurück liefert.

   Hinweis: mit der Schreibweise `<T1,T2, ..., Tn>` kann man ein generischer Datentyp mit n Typvariablen `T1, T2, ..., Tn` definiert werden.

   Lösung:
   Wir unterteilen die Aufgabenstellung in drei Schritte
   - Zu erst erstellen wir ein Interface mit dem Namen *Function*,
   - dann erweitern wir das Interface um die generischen Datentypen X und Y
   - und zu guter letzt erweitern wir das Interface um die Methode *apply*, die als Parameter den Datentypen X und als Rückgabetypen den Datentypen Y bekommt

```
1  public interface Function<X,Y>{
2  Y apply(X value);
3  }
```

2. **Teilaufgabe:** Implementieren Sie eine Klasse `Length`, die eine Funktion f : `String` $\to$ `Integer` repräsentiert. Die Klasse implementiert das Interface `Function`. Die Methode mit dem Name `apply` erhält als Eingabe ein Objekt vom Typ String und liefert die Länge des Strings verpackt als Objekt der Klasse `Integer` zurück.

   Hinweis: Für ein Objekt `str` der Klasse `Sting` liefert die Methode `int length()` die Länge (Anzahl der Zeichen) von `str` zurück.

   Lösung:
   Wir können die Aufgabenstellung wieder in mehrer Schritte zerlegen.
   (a) Eine Klasse Length anlegen
   (b) Klasse *Length* erweitern damit es das Interface *Function* implementieren muss.
   (c) Die Methode *apply* implementieren

```
1  public class Length implements Function<String,Integer>{
2    public Integer apply(String value){
3  return new Integer(value.length());
4    }
5  }
```

3. **Teilaufgabe:** Implementieren Sie eine Klasse `Test`. Diese Klasse besitzt eine `main()`-Methode, die folgenden Ablauf realisiert:
   - Es wird ein Objekt der Klasse `Length` erzeugt.
   - Das Objekt ruft die Methode `apply()` für die Eingabe String `"Inftech"` auf.
   - Der resultierende Funktionswert wird auf der Konsole ausgegeben.

   Lösung:
   Hier können wir direkt nach Stichpunkten gehen die wir vorgegeben haben.

```
1  public class Test{
2    public static void main(String[] args){
3  Length len = new Length();
4  System.out.println("Laenge des Objektes: "+len.apply("Inftech"));
5    }
6  }
```

Als Ausgaben würden wir dies hier erhalten. War aber nicht in der Aufgabe explizit gefordert.

```
1  user@PC:~/Probeklausur/code$ java Test
2  Laenge des Objektes: 7
```

Aufgabe 7 Listen.

1. **Teilaufgabe:** Betrachten Sie das folgende unvollständige Java-Programm für die doppelt verkettete Liste:

```
1  public class Liste<T> {
2
3    private class ListElem {
4      T data;
5
6      ListElem(T data) {
7        this.data = data;
8
9      }
10   }
11 }
```

   Ergänzen Sie die Klasse `Liste` um die nötigen Referenzen auf das erste und letzte Element der Liste. Ergänzen Sie weiter die innere Klasse um die benötigten Referenzen.

2. **Teilaufgabe:** Es sei n die Anzahl der Listenelemente. Der Kopf der Liste befindet sich an der 0-ten Stelle. Das letzte Listenelement befindet sich an der (n-1)-ten Stelle.

   Implementieren Sie eine Methode `public void add(T data, int i)`, die ein neues Listenelement mit dem Datenobjekt `data` an die i-te Stelle einfügt. Das ursprüngliche Element an der i-ten Stelle und die nachfolgenden Elemente rücken um eine Stelle nach rechts.

   Hinweis: Gehen Sie davon aus, das stets $0 < n$ gilt.

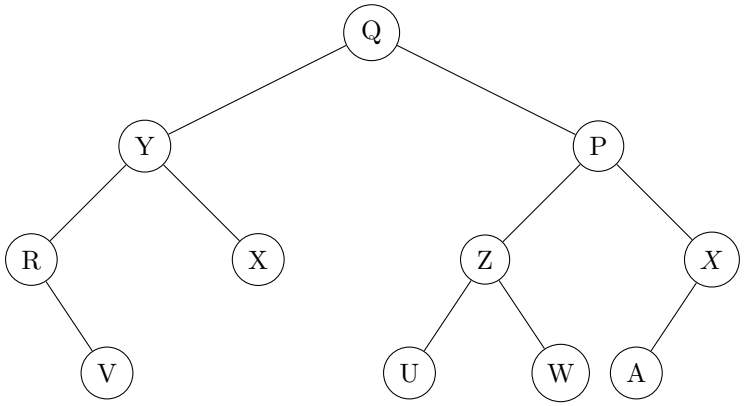
   Lösung:

```
1  public class Liste<T> {
2
3      private class ListElem {
4          T data;
5          ListElem next;
6          ListElem prev;
7
8          public ListElem(T data) {
9              this.data = data;
10         }
11     }
12
13     private ListElem head;
14     private ListElem tail;
15
16     public Liste() {
17         head = null;
18         tail  = null;
19     }
20
21     public void add(T data, int i) {
22         ListElem newElem = new ListElem(data);
23         //leere  Liste,  kann nach Aufgabenstellung auch weggelassen werden
24         if(head == null) {
25             head = newElem;
```

```
26              tail  = newElem;
27          }
28          //Liste hat mind. 1 Element
29          else {
30              ListElem current = head;
31              int currentIdx = 0;
32              while(current.next != null && currentIdx < i) {
33                  current  = current.next;
34                  currentIdx++;
35              }
36              //i <= Listenlaenge
37              if(currentIdx >= i) {
38                  newElem.next = current;
39                  newElem.prev = current.prev;
40                  newElem.next.prev = newElem;
41                  if(current == head)
42                      head = newElem;
43                  else
44                      newElem.prev.next = newElem;
45              }
46              //i > Listenlaenge
47              else {
48                  newElem.prev = current;
49                  newElem.prev.next = newElem;
50                  tail  = newElem;
51              }
52          }
53      }
54  }
```

Aufgabe 8 Traversierung von Bäumen.

1. **Teilaufgabe:** Geben Sie die entstehende Buchstabenfolge aus, wenn Sie den folgenden Binärbaum in postorder-Reihenfolge traversieren.



Lösung:

V, R, X, Y, U, W, Z, A, X, P, Q

2. **Teilaufgabe:** Implementieren Sie geeignete Java-Klassen `Tree` mit einer Unterklasse `Node` für einen Baum und für Baumknoten an. Dabei sollen folgende Bedingungen erfüllt sein:
   - Es sollen nur die Attribute (und keine Methoden) der Java-Klassen angegeben werden.
   - Die in den Knoten abgespeicherten Nutzdaten sind Elemente eines generischen Datentyps.
   - Die Klasse `Node` ist außerhalb der Klasse `Tree` unsichtbar.
   - Jeder Knoten des Baums hat maximal 3 Nachfolger.

   Lösung:

```
1  class Tree<T> {
2      private class Node <T> {
3          Node<T>[] childs;
4          T data;
5      }
6
7      Node<T> root;
8  }
```

3. **Teilaufgabe:** Implementieren Sie für die Klasse `Tree` aus Aufgabenteil 2) eine *rekursive* Methode `int countNodes()`, welche die Anzahl der Knoten dieses Baumes zurückgibt.

   Lösung:

```
1  int countNodes() {
2      return countNodes(root);
3  }
4
5  int countNodes(Node pos){
6      int count = 0;
7      if (pos.childs != null)
8          for (int i=0;i<pos.childs.length;i++)
9              count += countNodes(pos.childs[i]);
10     return count+1;
11 }
```

Aufgabe 9 AVL-Bäum.

1. **Teilaufgabe:** Setzen Sie eine Klasse `AVLbaum` auf, die die spezifische Unterklassen für Verzweigungen und Blätter sowie eine Referenz auf das Wurzelelement enthält, beide sollen jedoch die abstrakte Unterklasse Knoten erweitern. Weiter soll jede Verzweigung ein Attribut `hoehe` enthalten. Sie dürfen mit sichtbaren Attributen arbeiten.

2. **Teilaufgabe:** Erweitern Sie die Klasse um eine rekursive Methode `public boolean isAVL(Node x)`, die die AVL-Eigenschaft des Baumes tested.
3. **Teilaufgabe:** Erweitern Sie die Klasse weiter um eine Methode `public Node leftRot(Node x)`, die eine Linksrotation am übergebenen Knoten durchführt.

   Lösung:

```
1  public class AVLBaum<T>
2  {
3      abstract class Node
4      {
5          int key;
6
7          Node(int key)
8          {
9              this.key = key;
10         }
11     }
12
13     class Fork extends Node
14     {
15         Node links;
16         Node rechts;
17         int hoehe;
18
19         Fork(int key)
20         {
21             super(key);
22         }
23     }
24     class Leaf<T> extends Node
25     {
26         T daten;
27
28         Leaf(int  schluessel ,T daten)
29         {
30             super(schluessel);
31             this.daten = daten;
32         }
33     }
34
35     // Wurzel des AVL-Baums
36     private Node root;
37
38     //pruefe auf AVL-Bedingung
39     public boolean isAVL(Node x)
40     {
41         // pruefe fuer jeden Knoten AVL-Bedingung
42         // sobald sie nicht  erfuellt  ist  beende Rekursion
```

```
43          if (!checkAVLCondition(x))
44              return false;
45          else
46          {
47              // keine Blaetter pruefen
48              if (!(x instanceof AVLBaum.Leaf)) {
49                  //pruefe linke & rechte Kinder auf AVL-Bedingung
50                  return isAVL(((Fork)x).links) && isAVL(((Fork)x).rechts);
51          } else {
52          return true;
53          }
54          }
55      }
56
57      private boolean checkAVLCondition(Node x)
58      {
59          if (!(x instanceof Leaf))
60          {
61                  Fork v = (Fork) x;
62                  int lheight, rheight = 0;
63
64                  if(v.links == null || v.links instanceof AVLBaum.Leaf)
65                      lheight = 0;
66                  else
67                      lheight = ((Fork)v.links).hoehe;
68
69                  if(v.rechts == null || v.rechts instanceof AVLBaum.Leaf)
70                      rheight = 0;
71                  else
72                      rheight = ((Fork)v.rechts).hoehe;
73
74                  // berechne Hoehendifferenz der Kinder
75                  if(Math.abs(lheight - rheight) > 1)
76                      return false; // AVL-Bedingung stimmt nicht
77                  else
78                      return true; // AVL-Bedingung stimmt
79          }
80          // Blaetter erfuellen immer AVL-Bedingung
81          return true;
82      }
83
84      // Linksrotation am Knoten a
85      public Node leftRot(Node a)
86      {
87              Node tmp = a.right;
88              a.right = tmp.left;
89              tmp.left = a;
90
91              return tmp;
92      }
93
94  }
```