c) [4 Punkte] **(Huffman-Codierung)**
Gegeben seien die in der folgenden Tabelle aufgeführten Zeichen, die in einem Text mit den angegebenen Häufigkeiten auftreten. Bestimmen Sie mit dem in der Vorlesung vorgestellten Algorithmus die *Huffman-Codierung* für diese Zeichen und vervollständigen Sie die Tabelle.

Zeichen	Häufigkeit	*Huffman-Codierung*
z	100	01
v	30	1101
w	60	111
m	105	10
y	50	001
a	45	000
u	30	1100

✓

7 / 10

Aufgabe 2 (Polymorphie)

[6 Punkte] Gegeben sind die beiden folgenden Klassenhierarchien.

```
public class All { ... }
public class Most extends All { ... }
public class Special extends Most { ... }


public class Top
{
     public void m( Most p ) { System.out.print("K"); }
     public void m( Special p ) { System.out.print("L"); }
}

public class Middle extends Top
{
     public void m( All p ) { System.out.print("O"); }
     public void m( Most p ) { System.out.print("P"); }
}

public class Bottom extends Middle
{
     public void m( Middle p ) { System.out.print("X"); }
     public void m( Special p ) { System.out.print("Z"); }
}


public class Test
{
     public static void run()
     {
          All all = new All();
          Most most = new Most();
          Special special = new Special();

          Top tm = new Middle();
          Top tb = new Bottom();
          Middle mm = new Middle();
          Middle mb = new Bottom();

          tm.m( special );
          tm.m( most );
          tb.m( most );
          tb.m( special );
          mm.m( new Special() );
          mm.m( most );
          new Bottom().m( all );
          mb.m( special );
     }
}
```

Geben Sie die Zeichenfolge an, die durch die Methode `run` ausgegeben wird:

Ausgabe: LPPZLPOZ

6 / 16

Aufgabe 3 (Klasse)

a) [7 Punkte] Vervollständigen Sie die Klasse `Numbers`:

- `void reset( int index )` setzt den Wert im Feld `values` am Index `index` auf den Wert `0`, falls der Index innerhalb der Grenzen des Felds `values` liegt. Sonst soll nichts geschehen.
- `boolean insert( int val )` fügt den Wert `val` an dem kleinsten Index im Feld `values` ein, an dem der Wert `0` steht, und gibt `true` zurück. Gibt es keinen Wert `0` im Feld, wird `false` zurückgegeben und nichts eingefügt.
- `Numbers copy()` erzeugt ein neues Objekt der Klasse `Numbers`, das genau dem ausführenden Objekt entspricht.

```
public class Numbers
{
    private int[] values;

    public Numbers( int n )
    {
        values = new int[n];
    }

    public void reset( int index )
    {
        if(index>0 && index<values.length)
          values[index]=0;
    }

    public boolean insert( int val )
    {
        for(int i=0; i<values.length; i++){
            if(values[i]==0){
                values[i]=val;
                return true;
            }
        }
        return false;
    }

    public Numbers copy()
    {
        return new Numbers(values.length);
    }
}
```

wg Inhalt -2

(4)

b) [7 Punkte] Vervollständigen Sie die beiden folgenden Methoden, die die aus Aufgabenteil a) bekannte Klasse `Numbers` nutzen.

- `Numbers generate( int[] arr )` erzeugt ein Objekt der Klasse `Numbers`, das genau nur alle positiven Werte aus `arr` enthält. Besitzt `arr` keinen positiven Wert, soll `null` zurückgegeben werden.
- `boolean put( int[] arr, Numbers num )` soll in dem über `num` erreichbaren Objekt der Klasse `Numbers` alle Werte ungleich `0` erhalten und zugleich möglichst viele Werte aus `arr` hinzufügen. Die Methode gibt `true` zurück, falls alle Werte aus `arr` eingefügt werden konnten, sonst wird `false` zurückgegeben.

```
public static Numbers generate( int[] arr )
{
    int s[] = new int [arr.length]
    for (int i=0; i<arr.length; i++) {
    if (arr[i] > 0) {if (arr[i].copy() > 0)
        s[i] = arr[i]

    return s;
}
```

keine Lösung

-4 3

```
public static boolean put( int[] arr, Numbers num )
{
    int [] arr2 = num.copy();
    for (int i=0; i<arr2.length; i++ ) {
        if (arr2[i] == 0 ;
}
```

keine Lösung -3

3

Aufgabe 4 (Methoden)

a) [8 Punkte] Vervollständigen Sie die Methode `int notZeroSequence( int[] arr )`. Die Methode `notZeroSequence` soll die Anfangsposition der längsten Teilfolge von Werten im Feld `arr` angeben, die keine 0 enthält.

```
public static int notZeroSequence( int[] arr )
{
    int count = 0; f -1
    in max = 1;

    for (int i=0; i<arr.length-1; i++ ) {
        if (arr[i]<0 && arr[i+1]<0 )
            count++;
        else {
        max = Math.max(count, max);
         count = 1; }

                                                3
        return Math.max(count, max);
                          keine Anfangsposition
                                 bestimmt -3
                                                 (3)
}
```

kein Rücken -1

b) [4 Punkte] Vervollständigen Sie die Methode `int hasTriple( int[] arr, int n )`, die rekursiv arbeiten soll. Die Methode `hasTriple` soll `true` zurückgeben, wenn es im Feld `arr` drei unmittelbar aufeinander folgende Indizes gibt, an denen der gleiche Wert abgelegt ist. Sonst soll `false` zurückgegeben werden. Wird für `n` ein Wert außerhalb der Grenzen des Felds `arr` übergeben, soll `false` zurückgegeben werden.

Die Implementierung darf **keine** Schleifen enthalten.

```
public static boolean hasTriple( int[] arr, int n )
{
    if ( n < 0  ||  n >= arr.length )
    {
        return false;
    }
    else
    {
        if (n == 2)
        return arr[0] == arr[1] == arr[2];

        else{
        if (arr[n] == arr[n-1]) So:
                return true;
            else
            return hasTriple (arr, n-1);
        }
    }
}
```

(2)

Aufgabe 5 (Entwurfsmuster Iterator)

a) [8 Punkte] Die Klasse `IntNumbers` besitzt Konstruktoren und weitere Methoden, die hier aber nicht verwendet werden sollen. Ergänzen Sie die Klasse `IntNumbers` um einen Iterator `IntNumbersIterator`, dessen Objekte einen Durchlauf über alle Inhalte der beiden Felder `numbers1` und `numbers2` ermöglichen. Gehen Sie davon aus, dass `numbers1` und `number2` nicht auf `null` verweisen.

```
public class IntNumbers implements Iterable<T>
{
    private int[] numbers1, numbers2;

    // ... (Konstruktor und weitere Methoden sind nicht von Interesse)

    public Iterator<Integer> iterator()
    {
        return new IntNumbersIterator();
    }

    private class IntNumbersIterator implements Iterator<Integer>
    {
        Private int dim1;
        ~~Private int dim2;~~
        Public IntNumbersIterator() {
            dim1=0
        }

        Public boolean hasNext() {
        return dim1 < numbers1.length || dim1 < numbers2.length; } -1

        Public int Next() {
        if (hasNext()) {
        int a = numbers1[dim1];
        int b = numbers2[dim1];
            dim1++; if (a!=null) return a;
            return a; else return b;
        }
        else
        throw new IllegalStateException();
        }
    }
}
```

Konstruktion -3

Syntax -1

b) [4 Punkte] Die Methode `boolean notIn( int[] arr, Iterable<Integer> p )` soll `true` zurückgeben, falls kein Wert aus `arr` in `p` vorkommt. Sonst soll `false` zurückgegeben werden.

```
public static boolean notIn( int[] arr, Iterable<Integer> p )
{
    for ( int fromArr : arr )
    {
```

```
Iterator it = p.iterator();
int s = it.Next();
if (fromArr != s) {
        return false; }
```

Schleife -2

```
    }
    return
```

```
true
```

```
;
}
```

Aufgabe 6 (Methoden zur Klasse `DoublyLinkedList<T>`)

Ergänzen Sie die aus der Vorlesung bekannte Klasse `DoublyLinkedList<T>`, die Sie im Anhang finden. Bei der Implementierung der geforderten Methoden dürfen **nur** die **im Anhang** aufgeführten Methoden genutzt werden.

a) [7 Punkte] Vervollständigen Sie die Methode `void exchange()`.
Die Methode `exchange` soll die ersten beiden vom Listenanfang aus erreichbaren Inhalte, die nicht `null` sind, miteinander vertauschen. Gibt es keine zwei Inhalte, die ungleich `null` sind, soll nichts geschehen.

```
public void exchange()
{
    Element<T> current = current ;
    Element<T> firstHit = current.getSucc() ;
    while ( current != null )
    {
        if (firstHit != null) {
        if (current==first) {
        Element<T> p=current; Element<T> d=firstHit.getSucc();
          firstHit.disconnectPred(); first = firstHit;
        firstHit.connectAsSucc(p);
        p.connectAsSucc(d);
          }
        else if (firstHit==last) {
        Element<T> d= current.getPred(); Element<T> p=firstHit;
        current.disconnectPred(); last=current;
        d.connectAsSucc(p); p.connectAsSucc(current);
                                  }
        else {
        Element<T> d=current.getPred(); Element<T> l=current;
        Element<T> g = firstHit.getSucc();
        d.connectAsSucc(firstHit);
        firstHit.connectAsSucc( (      );
        l.connectAsSucc(g);
                                  }
    }
}
```

Keine Bedingung mit content −2
Kein Fortschritt kein Abbruch −1

(3)

b) [7 Punkte] Vervollständigen Sie die Methode `int splitBehind( T c )`.
Die Methode `splitBehind` soll alle Elemente aus der Liste entfernen, die auf das erste Element mit dem Inhalt `c` folgen. Die Anzahl der entfernten Elemente soll zurückgegeben werden. Kommt der Inhalt `c` nicht vor, soll kein Element entfernt und `0` zurückgegeben werden.
Der Vergleich soll mit der Methode `equals` vorgenommen werden.

```
public int splitBehind( T c )
{
    Element<T> current = first ;

    int newSize = ;

    while ( current!=null )
    {
        DoublyLinkedlist a=new DoublyLinkedlis<T>();
        if (current.getContent.equals(c)) {
        Element b=current.getSucc();
        if (current.hasSucc()) {
        Element<T> b= current.getSucc();
        a.first= b; a.last= last
           current.disconnectSucc( );
           Size = size- a.Size();
             newsize = a.size();
           return newSize;

                              }
                                          }
        else
          current =current.getSucc();
    }

    return 0;
}
```

Korrekturanmerkungen (rot): null; ? nicht initialisiert; f nicht gesetzt; s.o.; Size nicht bestimmt; 4

c) [8 Punkte] Vervollständigen Sie die Methode `int[] positions()`.
Die Methode `positions` soll ein Feld zurückgeben, in dem genau nur die Positionen derjenigen Elemente der Liste stehen, die `null` als Inhalt besitzen. Das erste Element der Liste hat die Position `0`. Kommt `null` nicht als Inhalt vor, soll ein Feld der Länge `0` zurückgegeben werden.

```
public int[] positions()
{
    int[] result = new int[0];
```

~~for (int~~
Element current = first;
for (int i=0; i<size; i++) {
if (current.getContent == null)
result [i] ++j ?

else
current = current.getSucc();

Hier ein Feld erzeugt, wie lang

-8 3

0

```
return result;
```

Aufgabe 7 (Methoden zur Klasse `BinarySearchTree<T extends Comparable<T>>`)

Ergänzen Sie die aus der Vorlesung bekannte Klasse `BinarySearchTree<T extends Comparable<T>>`, die Sie im Anhang finden. Bei der Implementierung der geforderten Methoden dürfen **nur** die **im Anhang** aufgeführten Methoden genutzt werden.

a) [7 Punkte] Vervollständigen Sie die Methode `int countNodes( int top, int bottom )`. Die Methode `countNodes` soll die Anzahl der Knoten zurückgeben, die im Baum auf den Ebenen von einschließlich `top` bis einschließlich `bottom` liegen. Ist `top` größer als `bottom`, soll `0` zurückgegeben werden. Nicht existierende Ebenen sollen bei der Ausführung von `countNodes` einfach übergangen werden. Die Wurzel des Baums liegt auf der Ebene `0`.

```
public int countNodes( int top, int bottom )
{
    if ( top <= bottom && !isEmpty() )
    {
        if (Bottom>=0 && top<=0)
        return 1+countNodes(top-1,bottom-1);
        return 1+rightChild.countNodes(top-1,
        Bottom-1)+leftChild.countNodes(top-1,Botto
                                        m
                                        -1).
                                        ;
        else
        return rightChild.countNodes(top-1,Bottom-1)
        +leftChild.countNodes(top-1)Bottom-1);
    }
    else
    {
        return 0;
    }
}
```

b) [7 Punkte] Vervollständigen Sie die Methode `int sortedUpTo( int n )`.
Die Methode `sortedUpTo` soll die kleinsten `n` Inhalte des Baums in aufsteigender Reihenfolge auf dem Bildschirm ausgeben. Ist `n` negativ oder `0`, soll keine Ausgabe erfolgen. Hat der Baum keine `n` Inhalte, sollen alle Inhalte ausgegeben werden.
Hinweis: Nutzen Sie den Rückgabewert, um die Anzahl der noch auszugebenden Inhalte zurückzugeben.

```
public int sortedUpTo( int n )
{
    if ( n > 0 && !isEmpty() )
    {
        return leftChild.SortedUpto(n-1);
        System.out.println(this.getContent());
        return rightChild.SortedUpto(n);
    }
    else
    {
        return leftChild.SortedUpto(n);
        System.out.println(this.getContent());
        return rightChild.SortedUpto(n);
    }
}
```

Keine Lösung erkennbar!

-7

7 / 14

Aufgabe 8 (Lambda-Ausdrücke)

[10 Punkte] Gegeben ist das Interface `IntegerBiFunction` und die Klasse `IData`.

```
public interface IntegerBiFunction {
    Integer apply( Integer p1, Integer p2 );
}
```

```
public class IData {

    private Integer[] iValues;

    public IData( Integer[] p ) { iValues = p; }

    public Integer do( IntegerBiFunction ibf ) {
        Integer lastApply = null;
        for ( Integer value : iValues ) {
            if ( value != null ) {
                lastApply = ibf.apply( value, lastApply );
            }
        }
        return lastApply ;
    }
}
```

Für die folgenden Teilaufgaben gilt: Die Referenz `id` verweist auf ein `IData`-Objekt.

a) Ergänzen Sie ein Argument derart, dass der Aufruf von `do` ein `Integer`-Objekt mit der Anzahl derjenigen Elemente im Feld `iValues` zurückgibt, deren Wert ungleich `null` ist. Gibt es kein solches Element, soll `null` zurückgegeben werden.

```
id.do( (p1,p2) -> 1    Lösung - 4 );
```

b) Ergänzen Sie ein Argument derart, dass der Aufruf von `do` ein `Integer`-Objekt mit dem Wert desjenigen `Integer`-Objekts im Feld `iValues` zurückgibt, das den größten Index besitzt. Gibt es kein solches Objekt, soll `null` zurückgegeben werden.

```
id.do( (p1,p2) -> p2    f - 2 );
```

c) Ergänzen Sie ein Argument derart, dass der Aufruf von `do` ein `Integer`-Objekt mit dem größten `int`-Wert zurückgibt, der in den im Feld `iValues` abgelegten `Integer`-Objekte vorkommt. Gibt es kein solches Objekt, soll `null` zurückgegeben werden.

Syntax - 1

```
id.do( (p1,p2) -> { if (p1 > p2){return p1;}else
{ return p2 ;}}    Klammerung fehlt -2 );
```

1 / 10

Anhang – Programmcode der Klasse BinarySearchTree<T>

```
public class BinarySearchTree<T implements Comparable<T>> {
    private T content;
    private BinarySearchTree<T> leftChild, rightChild;
    public BinarySearchTree() { ... }
    public T getContent() { ... }
    public boolean isEmpty() { ... }
    public boolean isLeaf() { ... }
}
```

Anhang – Programmcode der Klasse DoublyLinkedList<T>

```
public class DoublyLinkedList<T> {
    private Element first, last;
    private int size;
    public DoublyLinkedList() { ... }
    public int size() { ... }
    public boolean isEmpty() { ... }
    // Element
    private static class Element {
        private T content;
        private Element pred, succ;
        public Element( T c ) { ... }
        public T getContent() { ... }
        public void setContent( T c ) { ... }
        public boolean hasSucc() { ... }
        public Element getSucc() { ... }
        public void connectAsSucc( Element e) { ... }
        public void disconnectSucc() { ... }
        public boolean hasPred() { ... }
        public Element getPred() { ... }
        public void connectAsPred( Element e ) { ... }
        public void disconnectPred() { ... }
    }
}
```

Anhang – Programmcodes der Interfaces

```
public interface Iterator<T> {
    public abstract boolean hasNext();
    public abstract T next();
}

public interface Iterable<T> {
    public abstract Iterator<T> iterator();
}

public interface Comparable<T> {
    public abstract int compareTo( T t );          // Der Rückgabewert ist positiv, falls das
                                                   // ausführende Objekt größer als t ist.
}
```