

Abiturprüfung 2017

INFORMATIK

Hinweise zur Korrektur und Bewertung

(nicht für den Prüfling bestimmt)

Die Korrekturhinweise enthalten keine vollständigen Lösungen der Aufgaben, sondern einen Abriss des Erwartungshorizontes. Nicht genannte, aber *gleichwertige* Lösungswege und Begründungsansätze sind *gleichberechtigt*.

Die in den Lösungshinweisen gewählte objektorientierte Programmiersprache ist Java.

Die Bewertung der erbrachten Prüfungsleistungen hat sich an der bei jeder Teilaufgabe am linken Rand des Angabenblattes vermerkten, maximal erreichbaren Zahl von Bewertungseinheiten (BE) zu orientieren.

Umrechnung der erreichten Bewertungseinheiten in Notenpunkte:

Die insgesamt erreichten Bewertungseinheiten werden nach der folgenden Tabelle in Notenpunkte umgesetzt:

Notenpunkte	Notenstufen	Bewertungseinheiten	Intervalle in %
15	+1	120 ... 115	15
14	1	114 ... 109	
13	1–	108 ... 103	
12	+2	102 ... 97	15
11	2	96 ... 91	
10	2–	90 ... 85	
9	+3	84 ... 79	15
8	3	78 ... 73	
7	3–	72 ... 67	
6	+4	66 ... 61	15
5	4	60 ... 55	
4	4–	54 ... 49	
3	+5	48 ... 41	20
2	5	40 ... 33	
1	5–	32 ... 25	
0	6	24 ... 0	20

Aufgabe	BE	Hinweise
1.	5	<pre> class Ticket{ private String kennzeichen; private String telefonnr; private int parkzeitende; Ticket(String k, String t, int p){ kennzeichen = k; telefonnr = t; parkzeitende = p; } int restzeitGeben(int uhrzeit) { return parkzeitende-uhrzeit; } } </pre>
2.	4	Durch die Ticketanforderung des Autofahrers wird die Methode <i>registrieren</i> eines Objekts der Klasse PARKSYSTEM aufgerufen, wobei Kennzeichen, Telefonnummer und gewünschte Parkdauer aus der eingehenden SMS übergeben werden. Bei der Ausführung dieser Methode ermittelt das Parksystem die aktuelle Uhrzeit durch Aufruf einer Methode <i>uhrzeitGeben</i> eines Objekts der Klasse UHR und erzeugt ein Objekt der Klasse TICKET, in dem die übergebenen Werte für Kennzeichen, Telefonnummer und Parkzeitende gespeichert werden. Das Parksystem fügt dieses TICKET-Objekt dann einem Objekt der Klasse TICKETLISTE hinzu.

3. a)	5	<pre> graph TD tl1[Tl1:TICKETLISTE] -- erstes --> kn1[Kn1:KNOTEN] kn1 -- inhalt --> t1[t1:TICKET] kn1 -- nachfolger --> kn2[Kn2:KNOTEN] kn2 -- inhalt --> t2[t2:TICKET] kn2 -- nachfolger --> kn3[Kn3:KNOTEN] kn3 -- inhalt --> t3[t3:TICKET] kn3 -- nachfolger --> abs1[abs1:ABSCHLUSS] </pre>
b)	11	In der Klasse TICKETLISTE: <pre> void einfuegen(Ticket t) { erstes = new Knoten(erstes,t); } Ticket suchen(String k) { return erstes.suchen(k); } </pre> In der Klasse LISTENELEMENT: <pre> abstract Ticket suchen(String k); </pre> In der Klasse KNOTEN: <pre> Knoten(Listenelement l, Ticket t) { nachfolger = l; inhalt = t; } </pre>

```

Ticket suchen(String k) {
    if(inhalt.kennzeichenVergleichen(k)){
        return inhalt;
    } else {
        return nachfolger.suchen(k);
    }
}

```

In der Klasse ABSCHLUSS:

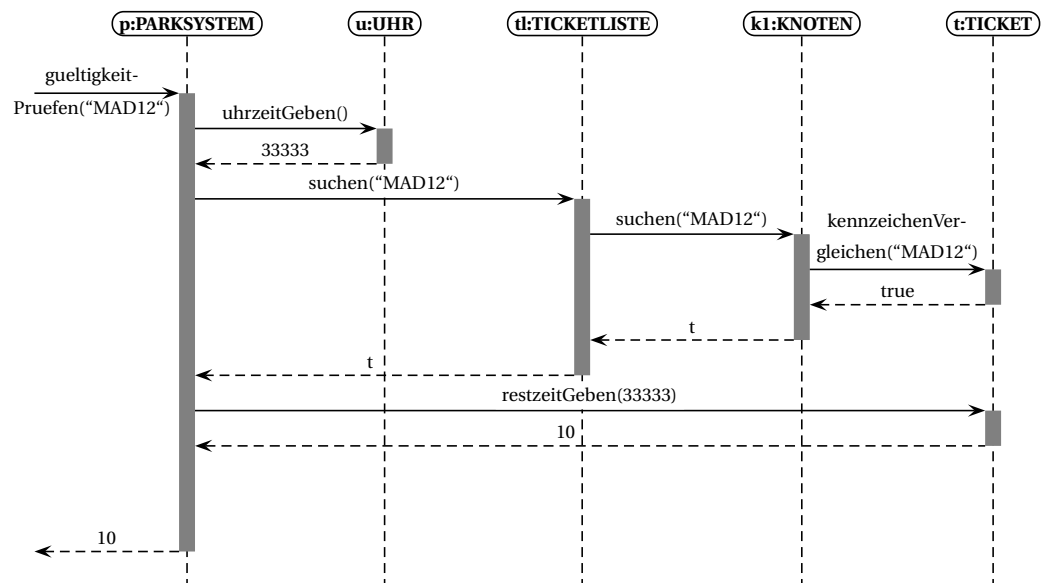
```

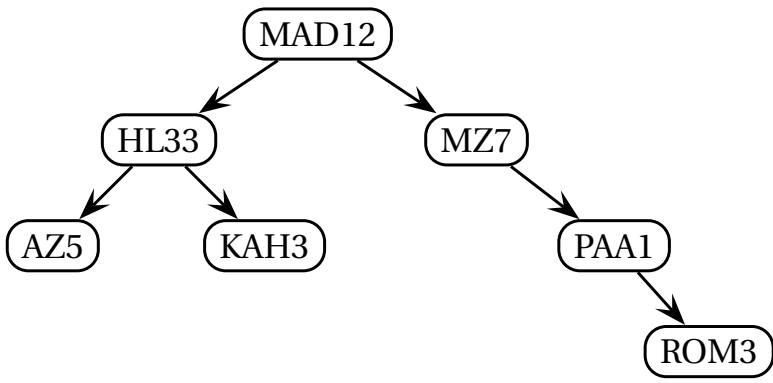
Ticket suchen(String k) {
    return null;
}

```

c)

8



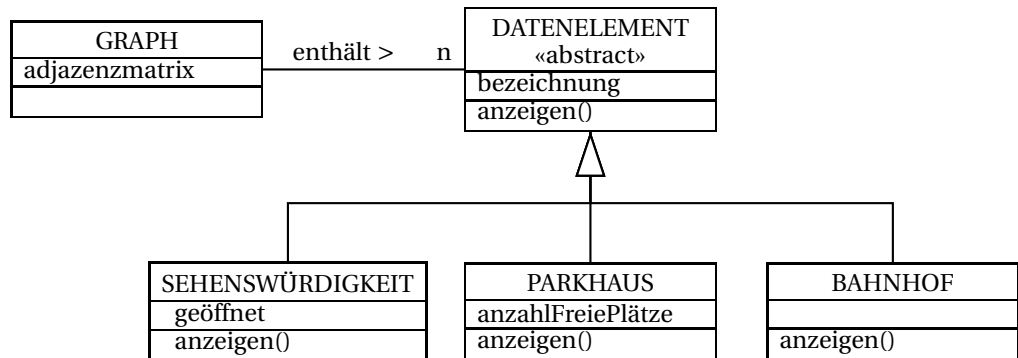
d)	3	Die Zeile gib t.restzeitGeben(uz) zurück wird ersetzt durch wenn t gleich null dann gib –10000 zurück sonst gib t.restzeitGeben(uz) zurück endewenn
4. a)	4	 <pre> graph TD MAD12 --> HL33 MAD12 --> MZ7 HL33 --> AZ5 HL33 --> KAH3 MZ7 --> PAA1 PAA1 --> ROM3 </pre> Es sind höchstens 4 Vergleiche nötig.
b)	6	In einem geordneten Binärbaum sind maximal so viele Vergleiche nötig, wie der Baum Ebenen hat. Bei 250 Elementen sind bei der ungünstigsten Baumstruktur, dem zu einer Liste entarteten Baum, maximal 250 Vergleiche nötig. Im günstigsten Fall sind alle Ebenen außer der untersten voll besetzt. Bei 250 Elementen braucht man 8 Ebenen, da in 8 Ebenen $1 + 2 + 4 + 8 + 16 + 32 + 64 + 128 = 255$ Elemente Platz finden, in 7 Ebenen dagegen nur 127 Elemente. Es sind also im günstigsten Fall höchstens 8 Vergleiche nötig.
c)	2	Jeder Knoten hat nun zwei Nachfolger. Ansonsten ändert sich bis auf eventuelle Umbenennungen der Klassenbezeichner und Referenzattribute nichts.

d)	4	Die entscheidende Methode <i>abgelaufeneZeigen</i> ist im Knoten zu formulieren: <pre> Methode abgelaufeneZeigen() linkerNachfolger.abgelaufeneZeigen() wenn verwaltetes Ticket abgelaufen dann Ticket ausgeben endwenn rechterNachfolger.abgelaufeneZeigen() endeMethode </pre>
e)	5	<pre> graph TD HAL5 --> BUN89 HAL5 --> NFX84 BUN89 --> DAM2 NFX84 --> MM3 NFX84 --> ST1 </pre>
5. a)	9	Der Graph ist zusammenhängend, gewichtet, ungerichtet und enthält Zyklen. <pre> graph TD Stadtmuseum --- 10 Hauptbahnhof Hauptbahnhof --- 3 Rathaus Hauptbahnhof --- 10 ParkhausWest[Parkhaus West] Rathaus --- 10 Burg Rathaus --- 5 Sudbahnhof[Südbahnhof] Burg --- 10 ParkhausSud[Parkhaus Süd] Burg --- 15 ParkhausWest Sudbahnhof --- 10 ParkhausSud </pre>

	Mus	Hbf	PWest	PSüd	Rathaus	Burg	Südbhf
Mus		10					
Hbf	10		10		3		
PWest		10		15			
PSüd			15			10	10
Rathaus		3				10	
Burg				10	10		5
Südbhf				10		5	

b)

5



c)

9

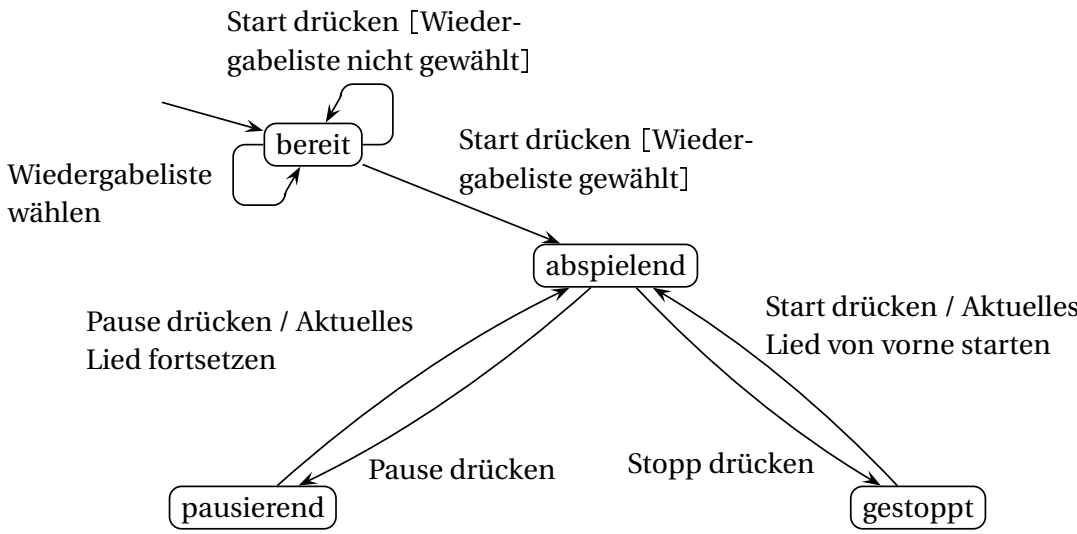
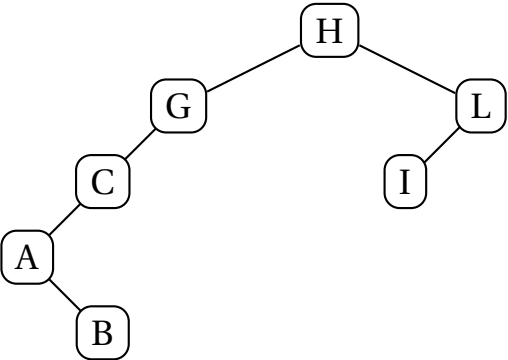
Methode gibAlleMax1U(KNOTEN k)
 wiederhole für alle Nachbarknoten k1 von k
 zeit1 = Gewicht der Kante k-k1
 gib Bezeichnung von k1 und zeit1 aus
 wiederhole für alle Nachbarknoten k2 von k1
 wenn k2 ungleich k dann
 zeit2 = zeit1 + 5 + Gewicht der Kante k1-k2
 gib Bezeichnung von k2 und zeit2 aus
 endewenn
 endewiederhole
 endewiederhole
 endeMethode

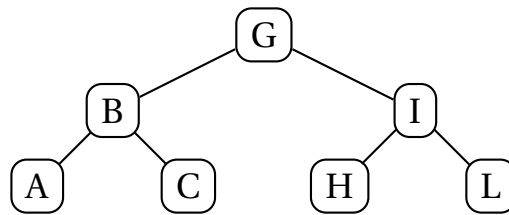
80

Aufgabe	BE	Hinweise
1. a)	5	
b)	9	<i>Hinweis:</i> Die Zuweisungen <code>naechster = abs1</code> und <code>erster = kn2</code> dienen der Veranschaulichung und müssen nicht dargestellt werden. Die Knoten <code>kn1</code> und <code>kn3</code> wurden aus der Liste entfernt, d. h. alle Songs des Interpreten „Detlef“ wurden aus der Liste gelöscht.

c)	7	In der Klasse WIEDERGABELISTE: <pre>int gesamtdauerGeben(){ return erster.gesamtdauerGeben(); }</pre> In der Klasse LISTENELEMENT: <pre>abstract int gesamtdauerGeben();</pre> In der Klasse ABSCHLUSS: <pre>int gesamtdauerGeben(){ return 0; }</pre> In der Klasse KNOTEN: <pre>int gesamtdauerGeben(){ return naechster.gesamtdauerGeben() +inhalt.dauerGeben(); }</pre>
2. a)	15	In der Klasse SONG: <pre>boolean istHaeufigerAls(Song andererSong){ return (gespielt > andererSong.gespielt); }</pre> In der Klasse WIEDERGABELISTE: <pre>void sortiertEinfuegen(Song song){ erster = erster.sortiertEinfuegen(song); }</pre> In der Klasse LISTENELEMENT: <pre>abstract Listenelement sortiertEinfuegen(Song song);</pre>

		In der Klasse ABSCHLUSS: <pre>Listenelement sortiertEinfuegen(Song song){ Knoten neuerKnoten = new Knoten(song, this); return neuerKnoten; }</pre> In der Klasse KNOTEN: <pre>Knoten(Song song, Listenelement n){ inhalt = song; naechster = n; } Listenelement sortiertEinfuegen(Song song){ if (inhalt.istHaeufigerAls(song)){ naechster = naechster.sortiertEinfuegen(song); return this; } else { Knoten neuerKnoten = new Knoten(song, this); return neuerKnoten; } }</pre>
b)	8	Mögliche Realisierung: • Überprüfung mit <i>istEnthalten(song)</i>, ob der Song bereits in der Hitliste enthalten ist. Falls dies zutrifft, wird er mit <i>entfernen(song)</i> aus der Liste gelöscht. • Sortiertes Einfügen des Songs mit <i>sortiertEinfuegen(song)</i>. • Überprüfung mit <i>anzahlGeben()</i>, ob die maximale Anzahl von 20 Songs in der Liste überschritten wurde. Falls dies zutrifft, wird der letzte Song der Liste mit <i>entfernen(20)</i> wieder gelöscht.
c)	4	Mögliche Aspekte: Implementierungsaufwand, Speicherplatz, Zugriff auf Daten, Längenproblematik

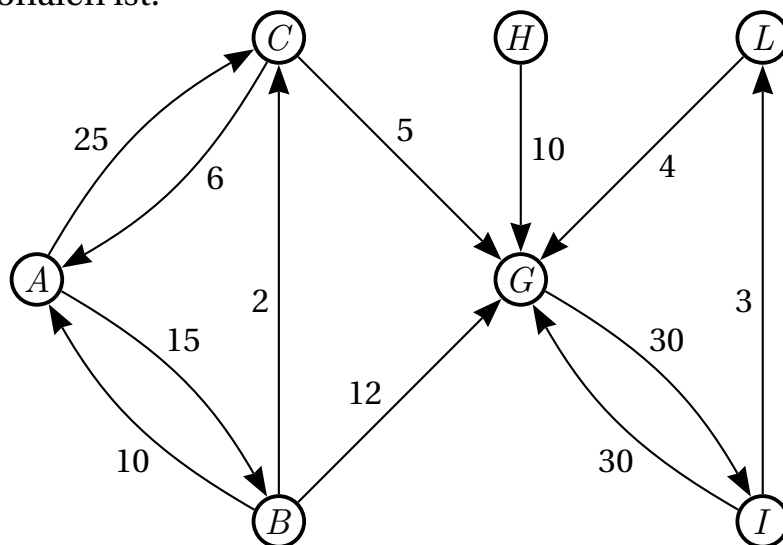
3.	8	
4. a)	3	Jeder Knoten trägt ein Sortierkriterium (hier den Titel), wobei für die Knoten gilt: • Jeder Wert des Sortierkriteriums im linken Teilbaum des Knotens ist lexikografisch kleiner als der entsprechende Wert des Knotens. • Jeder Wert des Sortierkriteriums im rechten Teilbaum des Knotens ist gleich oder lexikografisch größer als der entsprechende Wert des Knotens.
b)	6	 Der Baum besitzt nicht die minimal mögliche Höhe.



5. a)

6

Der Graph ist gerichtet, da die Adjazenzmatrix nicht symmetrisch zur Diagonalen ist.



b)

9

Erwartet wird beispielsweise die Formulierung eines modifizierten Tiefensuche-Algorithmus, bei dem nicht nur überprüft wird, ob ein Nachbarknoten noch unbesucht ist, sondern auch, ob das zugehörige Kantengewicht mindestens 10 beträgt.
Mögliches Ergebnis: A - B - G - I - C

80

Informatik

Aufgabe Inf2.III

Aufgabe	BE	Hinweise
1. a)	5	$G = (\{ \text{Maße, Länge, Breite, Ziffer, Ziffer2, Ziffer5}, \{0,1,2,3,4,5,6,7,8,9,\times\}, P, \text{Maße})$ $P = \{$ Maße $\rightarrow$ Länge $\times$ Breite $\times$ Breite Länge $\rightarrow$ Ziffer5 Ziffer 6 0 Breite $\rightarrow$ 3 Ziffer5 Ziffer2 Ziffer Ziffer $\rightarrow$ 0 1 2 3 4 5 6 7 8 9 Ziffer2 $\rightarrow$ 0 1 2 Ziffer5 $\rightarrow$ 0 1 2 3 4 5 $\}$
b)	4	
c)	2	$z_0 \rightarrow z_1 \rightarrow z_3 \rightarrow z_4 \rightarrow z_5 \rightarrow z_7 \rightarrow z_8 \rightarrow z_{10} \rightarrow z_{11}$
d)	2	47 $\times$ 08 $\times$ 15 ist ein Wort der Sprache L, weil es der Syntax dieser Sprache genügt, also durch die Grammatik ableitbar ist. Die Semantik dieses Wortes beschreibt seine Bedeutung, nämlich die Paketmaße 47 cm Länge, 8 cm Breite und 15 cm Höhe.

2. a)	3		Akk	201	202
				6	17
		load 202	17	6	17
		sub 201	11	6	17
		store 202	11	6	11
		sub 201	5	6	11
		store 202	5	6	5
		sub 201	-1	6	5
		load 202	5	6	5
b)	2	Es wird der Rest der Division der Zahl in Speicherzelle 202 durch die Zahl in 201 berechnet. Im konkreten Fall können die Prüfwerte 0, 1, 2, 3, 4 und 5 erzeugt werden.			
c)	4	wiederhole solange $a - b$ größer oder gleich 0 $a = a - b$ gib a zurück			
d)	3	6: load 201 7: jgt 10 8: dload -1 9: jump 16			
3. a)	5	Bestellschicht Online-Shop Versandschicht Paketdienst Transportschicht Verteilerzentrum 1 Kunde Paketstation Verteilerzentrum 2 Lieferung übergeben Versand zustellen Transport abholen deponieren			

b)	2	Ein Aspekt beispielsweise aus: Strukturierung komplexer Arbeitsabläufe, leichte Austauschbarkeit einzelner Schichten
4.	4	Die Reservierung von Kunde b findet statt, nachdem die Reservierung von Kunde a angenommen, aber bevor die Zahl der freien Fächer um eins vermindert wurde. Die Verwendung eines Monitors, der den gegebenen Algorithmus umfasst, stellt sicher, dass die durch a und b beim Server ausgelösten Prozesse diesen Abschnitt nicht gleichzeitig betreten können.
5.	4	Da der Unbefugte die Passwortregeln kennt, kann er alle 5-stelligen Zeichenketten aus den Ziffern 0 bis 9 durchprobieren und braucht dafür maximal $10^5 \cdot 0,2 \text{ ms} = 20 \text{ s}$. Erschwerung durch z. B.: • Erhöhung der Passwortlänge • Erweiterung des Eingabealphabets • Abbruch nach drei fehlerhaften Eingabeversuchen
	40	

Informatik

Aufgabe Inf2.IV

Aufgabe	BE	Hinweise
1. a)	3	<

c)	10	<pre>00 lda 101 01 dec 02 jaz 26 // falls k = 1 03 ldai 0 04 sta 102 05 ldai 1 06 sta 103 07 lda 103 08 inc 09 sta 103 // t=t+1 10 lda 101 11 mod 103 12 jap 15 // nicht Teiler 13 ldai 1 14 sta 102 15 lda 102 16 jaz 07 // solange a = 0 17 lda 103 18 sub 101 19 jaz 23 // t = k 20 ldai 0 21 sta 104 22 jmp 25 23 ldai 1 24 sta 104 25 jmp 28 26 ldai 0 27 sta 104 28 end</pre>
----	----	---

3. a)	3	$\text{mult}(4,10) = \text{mult}(8,5) = 8 + \text{mult}(16,2) =$ $= 8 + \text{mult}(32,1) = 8 + 32 = 40$
b)	2	Da beim rekursiven Aufruf der Wert von b halbiert wird und somit immer 0 bleibt, ist die Abbruchbedingung nie erfüllt; der Algorithmus terminiert nicht.
c)	7	i) $\text{mult}(2,3)$: 1 rekursiver Aufruf $\text{mult}(2,1024)$: 10 rekursive Aufrufe ii) Es handelt sich um ein logarithmisches Laufzeitverhalten. Dieses Laufzeitverhalten kommt zustande, weil der Wert von b bei jedem rekursiven Aufruf der Methode halbiert wird. In $\text{mult}(a,b)$ wird nur eine Fallunterscheidung nach b vorgenommen; deshalb ist die Anzahl der rekursiven Aufrufe unabhängig von a.
d)	3	Durch den großen Wert von b sind viele rekursive Aufrufe nötig. Würde man die beiden Parameter vertauschen, würde die Zahl der nach dem ersten Aufruf zusätzlich erforderlichen rekursiven Aufrufe von 10 auf 1 reduziert werden. Bei einer verbesserten Version werden zu Beginn die Werte von a und b vertauscht, falls $a < b$.
	40	