Universität Siegen Freiwillig erstellte Übung, unabhängig von der Universität

Algorithmen und Datenstrukturen
(Rekonstruierte Klausur*)

Dozent: Prof. Dr. Blz. Wintersemester 2016/17
Prüfungsdatum: Frühling 2017 **Dauer:** 120 Minuten
Autor dieser Übung: AKo **Revision:** 0.1.0 (17.Mär.'17)

Aufgabe:	1	2	3	4	5	6	7	8	9	10	Gesamt
Punkte:	20	30	20	20	20	10	20	20	20	20	200
Erreicht:											

Erlaubte Hilfsmittel:

- Dokumentenechte Schreibgeräte (kein Bleistift, kein Rot und Grün)
- Lineal, Geo-Dreieck
- Wörterbuch der deutschen Sprache

Bitte beachten:

- Tragen Sie auf jedem Blatt Ihren Namen und Ihre Matrikel-Nr. ein.
- Benutzen Sie nur das ausgeteilte Papier. Notizen sind auf der Rückseite der Klausurbögen zulässig und werden nicht bewertet (außer wenn entsprechend gekennzeichnet).
- Die Klammerung der Klausur darf nicht gelöst werden.
- Schreiben Sie deutlich! Uneindeutige Ergebnisse sind ungültig.
- Ein Täuschungsversuch (Abschreiben, unerlaubte Hilfsmittel, Kommunikation mit anderen Klausurteilnehmern, etc.) führt zum Ausschluss aus der Prüfung. Es erfolgt keine Verwarnung.

Name: _____ Matrikelnummer: _____

** Dieses Dokument ist entstanden aus Erinnerungen und Abschätzungen an die tatsächliche Klausur, gesammelt von mehreren Personen, um dieser Klausur ähnliche Aufgaben (mit ggf. anderen, ähnlichen Zahlenwerten etc.) zur Übung zu erzeugen. Dies dient einzig dem Zweck der Weiterbildung und Vertiefung der Themen dieses Fachgebietes und einer Vorbereitung für zukünftige Klausuren, wo die aktuelle Probeklausur in Form einer gekürzten Klausur von 2008 nicht mehr zwingend ausreicht und besonders Erstsemestern kein gutes Bild über den tatsächlichen Schwierigkeitsgrad der Klausur gibt. Dieses Dokument ist nicht offiziell, nicht auf Richtigkeit geprüft und kein originales Klausurdokument und versucht dies auch nicht zu sein. Echte Klausuraufgaben werden von den hier gezeigten Aufgaben abweichen. Punktetabellen und Hinweise auf dieser Seite dienen einzig der Selbstkontrolle zur Vorbereitung auf die echte Klausur.*

1. **Fragen zur Rechnerarchitektur** **Punkte: 20**

Eine falsche Antwort führt zu Punktabzug. Eine leere Antwort entspricht 0 Punkten.
Es sind nicht weniger als 0 Punkte für diese Aufgabe möglich.

(a) (1 Punkt)

(b) (1 Punkt) Ein Interpreter übersetzt den vorliegenden Quellcode zur Laufzeit.
◯ Wahr ◯ Falsch

(c) (1 Punkt) Geschriebener Quellcode kann sofort ohne Umwege ausgeführt werden.
◯ Wahr ◯ Falsch

(d) (1 Punkt) Assemblersprachen sind unabhängig von der Prozessorarchitektur.
◯ Wahr ◯ Falsch

(e) (1 Punkt) Das Zweierkomplement berechnet sich durch das Invertieren des Betrags plus 1.
◯ Wahr ◯ Falsch

(f) (1 Punkt) Im IEEE-754-Float-Format wird ein Bit zur Speicherung des Vorzeichens genutzt.
◯ Wahr ◯ Falsch

(g) (1 Punkt)

(h) (1 Punkt) Auf den ROM-Speicher kann nur lesend zugegriffen werden.
◯ Wahr ◯ Falsch

(i) (1 Punkt) Der erste Schritt im Wasserfallmodell der Softwareentwicklung ist die Programmierung.
◯ Wahr ◯ Falsch

(j) (1 Punkt) Das heutige Modell der Abfolge von Softwareentwicklung ist ein Zyklus.
◯ Wahr ◯ Falsch

(k) (1 Punkt) Die Trennung von Programmen und Daten im Hauptspeicher ist ein wesentlicher Bestandteil der Von-Neumann-Architektur.
◯ Wahr ◯ Falsch

(l) (1 Punkt) Das Bottleneck moderner Computer ist gegeben durch die Übertragung von Daten.
◯ Wahr ◯ Falsch

(m) (1 Punkt)

(n) (1 Punkt) C ist eine objektorientierte Programmiersprache.
◯ Wahr ◯ Falsch

(o) (1 Punkt) Interrupts sind asynchrone Service-Anfragen.
◯ Wahr ◯ Falsch

(p) (1 Punkt) Vor dem Behandeln eines Interrupts müssen alle Registerwerte gesichert und danach wiederhergestellt werden.
◯ Wahr ◯ Falsch

(q) (1 Punkt)

(r) (1 Punkt)

(s) (1 Punkt)

(t) (1 Punkt) Cache-Optimierung sorgt dafür, dass Daten nicht mehrmals genutzt werden.
◯ Wahr ◯ Falsch

Nicht eingetragene Fragen fehlen noch. Alle 20 zählen in der echten Klausur.

2. **Codierung** **Punkte: 30**

(a) (10 Punkte) **Zahlensysteme**

i. Wandeln Sie die Dezimalzahl 7_{10} in eine 4-Bit Binärzahl um.

ii. Wandeln Sie die Binärzahl 1011_2 in das Dezimalsystem um.

iii. Wandeln Sie die folgende Binärzahl in das Hexadezimalsystem um:
$$10111000001111001101100010 10_2$$

iv. Wandeln Sie die folgende Binärzahl in das Oktalsystem um:
$$10101100111100110110 0_2$$

v. Wandeln Sie die Hexadezimalzahl $\mathrm{D8}$ in das Dezimalsystem um.

vi. Wandeln Sie die Dezimalzahl -7_{10} in eine Binärzahl in Zweierkomplement-Darstellung mit 5 Bit Wortlänge um.

vii. Wandeln Sie die Dezimalzahl -3_{10} in eine Binärzahl in Zweierkomplement-Darstellung mit 5 Bit Wortlänge um.

viii. Addieren Sie die Dezimalzahlen -7_{10} und -3_{10} in ihrer Zweierkomplement-Darstellung schriftlich. Nutzen Sie Ihre Ergebnisse von (vi) und (vii).

Geben Sie anschließend das Ergebnis Ihrer Addition in 5-Bit-Zweierkomplement-Darstellung an:

(b) (12 Punkte) **IEEE-754 Gleitkommazahlen**

Gegeben sei die Dezimalzahl 18.671875.
In der Form $(g_{10} + r_{10})$ ist also $g_{10} = 18$ und $r_{10} = 0.671875$.
Wandeln Sie in den angegebenen Schritten diese Dezimalzahl in eine Gleitkommazahl nach IEEE-754 um.

i. Geben Sie g_2, die binäre Darstellung des ganzzahligen Anteils, an.

ii. Geben Sie r_2, die binäre Darstellung des rationalen Anteiles, an.

iii. Geben Sie nun die vorläufige Mantisse $(g_2 + r_2)$ an.

iv. Wandeln Sie die Mantisse in die wissenschaftliche Schreibweise um.

v. Geben Sie den binärcodierten Exponenten an (Biased!).

vi. Geben Sie abschließend die komplette IEEE-754-Gleitkommazahl an.

(c) (8 Punkte) **Huffman-Codierung**

Führen Sie mit der folgenden Tabelle den Huffman-Algorithmus aus, um das Wort `BANANABAND` zu codieren.
Die Kürzel in der Tabelle stehen für Zeichen, Auftrittswahrscheinlichkeit und Code (Binärwert).

Z	W	C	Z	W	C	Z	W	C	Z	W	C

Geben Sie das Wort codiert mit Ihrer berechneten Codierung an:

3. **Formale Sprachen: BNF-Syntax** **Punkte: 20**

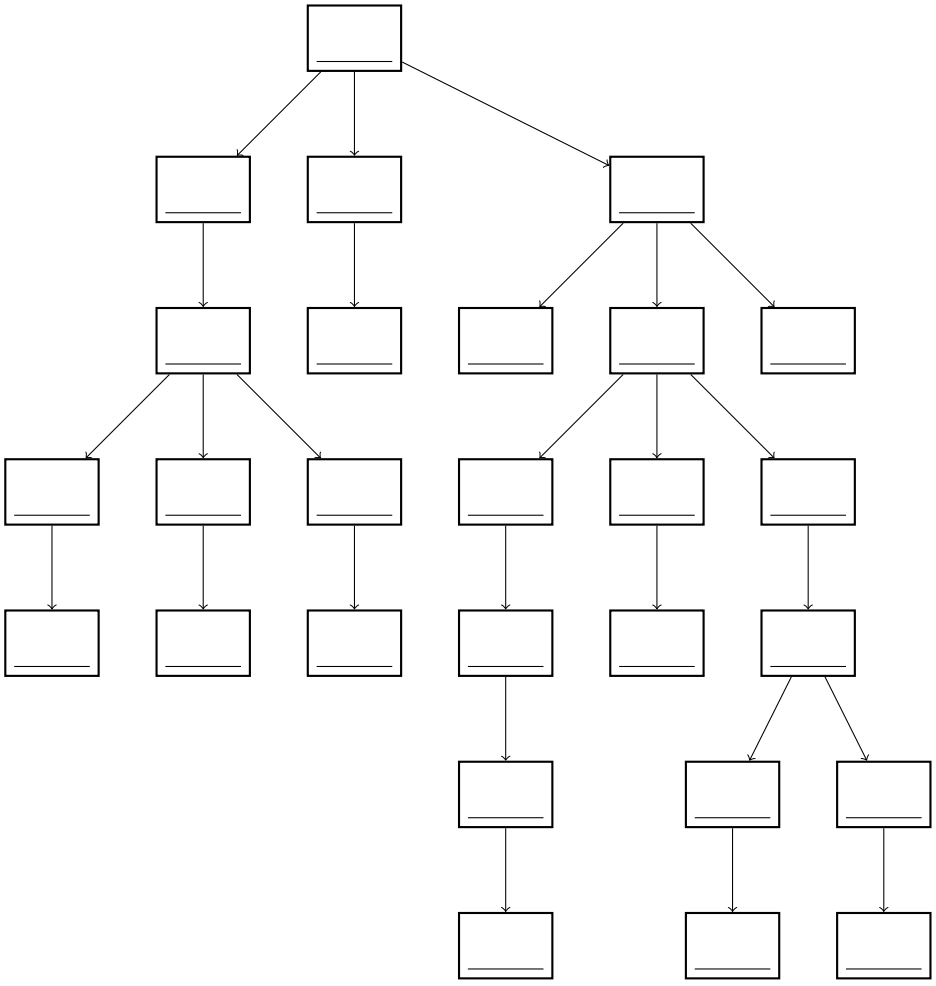
Achtung! *Diese Aufgabe richtet sich nur an Studierende des Studiengangs Informatik (10LP). Studierende des Studiengangs Elektrotechnik (5LP) müssen diese Aufgabe nicht bearbeiten. Es gibt auch keine Zusatzpunkte, sollte die Aufgabe dennoch bearbeitet werden.*

Gegeben sei folgende BNF-Syntax:

Es gibt die Terminale `{a,b,c,d,0,1,2,3,+,-,*,/}`
und die Non-Terminale `{<A>,<V>,<K>,<B>,<Z>,<N>,<O>}`.
Das Startsymbol ist `<A>`.

```
<A> ::= <V> | <N><K> | (<A>) | <A><O><A>
<V> ::= <B>{<B>} | <B>{<Z>}
<K> ::= [-]<Z>
<B> ::= a | b | c | d
<Z> ::= 0 | 1 | 2 | 3
<N> ::= + | -
<O> ::= + | - | * | /
```

(a) (10 Punkte) Füllen Sie den gegebenen Ableitungsbaum für das Wort `c21*(a+bd)` aus.



Zur Erinnerung:

```
<A> ::= <V> | <N><K> | (<A>) | <A><O><A>
<V> ::= <B>{<B>} | <B>{<Z>}
<K> ::= [-]<Z>
<B> ::= a | b | c | d
<Z> ::= 0 | 1 | 2 | 3
<N> ::= + | -
<O> ::= + | - | * | /
```

(b) Welche der folgenden Worte können mit dieser Syntax erzeugt werden?
Es gilt die bekannte Regel zum Punktabzug durch falsche Antworten.

i. (2 Punkte) `b01`
◯ Wahr ◯ Falsch

ii. (2 Punkte) `-17a`
◯ Wahr ◯ Falsch

iii. (2 Punkte) `++*2`
◯ Wahr ◯ Falsch

iv. (2 Punkte) `(a*c)/(b*c)`
◯ Wahr ◯ Falsch

v. (2 Punkte) `(ab4-c)/2`
◯ Wahr ◯ Falsch

4. **Logik** **Punkte: 20**

(a) **Aussagenlogik**

i. (8 Punkte) Füllen Sie die gegebenen Wahrheitstafeln mit den Wahrheitswerten 0 und 1 aus.

A	B	C	$(A \vee \neg C) \wedge B$	$\neg C \Rightarrow (A \wedge \neg B)$	$(A \wedge \neg C) \Rightarrow (\neg B \vee C)$
0	0	0			
0	0	1			
0	1	0			
0	1	1			
1	0	0			
1	0	1			
1	1	0			
1	1	1			

ii. (4 Punkte) Gegeben ist die folgende Wahrheitstafel der Funktion $f\,(A, B, C)$. Geben Sie die Funktion f in disjunktiver Normalform und konjunktiver Normalform an.

A	B	C	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

DNF:

_____

_____

_____

KNF:

_____

_____

_____

(b) **Prädikatenlogik**

Gegeben sind folgende Prädikate:
$H(x)$: x ist Hund
$K(x)$: x ist Katze
$F(x, y)$: x ist befreundet mit y
und die Konstante $c =$ Cooper.

i. Ordnen Sie die folgenden prädikatenlogische Aussagen der zugehörigen Textform zu:

1. $\exists x, y : H(x) \wedge K(y) \wedge F(x, y)$
2. $\exists x : \neg F(x, c)$
3. $\forall x : K(x) \Rightarrow F(x, c)$
4. $\forall x : \neg H(x) \wedge \neg K(x) \Rightarrow \neg F(c, x)$

Es gilt die bekannte Regel zum Punktabzug durch falsche Antworten.

α) (1 Punkt) Jede Katze ist mit Cooper befreundet.
◯ 1 ◯ 2 ◯ 3 ◯ 4

β) (1 Punkt) Es gibt Katzen, die mit Hunden befreundet sind.
◯ 1 ◯ 2 ◯ 3 ◯ 4

γ) (1 Punkt) Cooper ist nur mit Katzen und Hunden befreundet.
◯ 1 ◯ 2 ◯ 3 ◯ 4

δ) (1 Punkt) Es gibt Tiere, die nicht mit Cooper befreundet sind.
◯ 1 ◯ 2 ◯ 3 ◯ 4

ii. Geben Sie die folgenden Aussagen als prädikatenlogische Formel an.

α) (2 Punkte) Ein Hund ist genau dann mit Cooper befreundet, wenn Cooper eine Katze ist.

β) (2 Punkte) Es gibt eine Katze, die mit allen Katzen befreundet ist.

5. **Programmierung & Rekursion** **Punkte: 20**

(a) Gegeben seien die folgenden Funktionen in C++. Geben Sie an, welchen Wert die Variable `x` nach dem Durchlauf der jeweiligen Funktion hat.

i. (1 Punkt)
```
int x = 5;
int *p = &x;
*p += 9;
x++;
```
☐ 5 ☐ 6 ☐ 14 ☐ 15

ii. (1 Punkt)
```
int x = 5;
while (x % 10 > 0)
    x++;
```
☐ 5 ☐ 6 ☐ 10 ☐ 11

iii. (1 Punkt)
```
int x = 5;
int y = x + 4;
x += y - x;
```
☐ 1 ☐ 4 ☐ 9 ☐ 13

iv. (1 Punkt)
```
int x = 5;
for (int i = x; i > 0; i--)
    x += i;
```
☐ 15 ☐ 17 ☐ 19 ☐ 20

v. (1 Punkt)
```
int x = 5;
for (int i = 0; i < x; i++)
    for (int j = i; j >= 0; j--)
        x *= j;
```
☐ 0 ☐ 5 ☐ 17 ☐ 120

(b) Gegeben sei die folgende Funktion in C++:

```
int f(int &x) {
    if (x == 1)
        return 1;
    else
        return x * f(x - 1);
}
```

i. (2 Punkte) Welches Ergebnis liefert `f` allgemein?

☐ x^x ☐ $x!$ ☐ $x \cdot \log_2 x$ ☐ x^2

ii. (3 Punkte) Um welche Art der Rekursion handelt es sich bei `f`?

α) Lineare Rekursion
◯ Wahr ◯ Falsch

β) Primitive Rekursion
◯ Wahr ◯ Falsch

γ) Endrekursion
◯ Wahr ◯ Falsch

(c) (10 Punkte) Gegeben sei die folgende C++-Quellcode-Datei. Untersuchen Sie den Code, betrachten Sie dabei auch die unten angegebene Konsolenausgabe des Programms, und geben Sie die Werte für die Konstanten `w`, `x`, `y`, `z` in den entsprechenden Feldern an, um diese Ausgabe zu erzeugen.

```
#include <stdio.h>
void print(int &s);

int main(int* argc, char** argv) {

    const int w = ______ ;

    const int x = ______ ;

    const int y = ______ ;

    const int z = ______ ;

    for (int i = w; i > 0; --i) {
        switch(i) {
        case 0:
        case 10:
            print(x);
            break;
        case 5:
            print(y);
            break;
        default:
            print(z);
        }
    }
}

void print(int &s) {
    for (int i = 0; i < s; ++i) {
        std::cout << '*';
    }
    std::cout << std::endl;
}
```

Output:

```
**********
*
*
*
*
*****
*
*
*
*
**********
```

6. **Komplexität** **Punkte: 10**

(a) Bewerten Sie, ob die nachfolgenden Aussagen zur $\mathcal{O}$-Notation stimmen.
Es gilt die bekannte Regel zum Punktabzug durch falsche Antworten.

i. (1 Punkt) $2n^2 + n \in \mathcal{O}(n^2)$
◯ Wahr ◯ Falsch

ii. (1 Punkt) $1 \in \mathcal{O}(n)$
◯ Wahr ◯ Falsch

iii. (1 Punkt) $n^3 \in \mathcal{O}(n^2)$
◯ Wahr ◯ Falsch

iv. (1 Punkt) $4n^3 - 4 \in \mathcal{O}(n^2)$
◯ Wahr ◯ Falsch

v. (1 Punkt) $2^{6n} \in \Theta(2^n)$
◯ Wahr ◯ Falsch

(b) (5 Punkte) Bewerten Sie den Aufwand der folgenden Funktion:

```
int f(int &x) {
    int sum = 0;
    for(int j = 0; j <= x; j++) {
        for (int i = 0; i <= j; i++) {
           sum += i + j;
        }
    }
    return sum;
}
```

Die genaue Grenze des Aufwands ist:

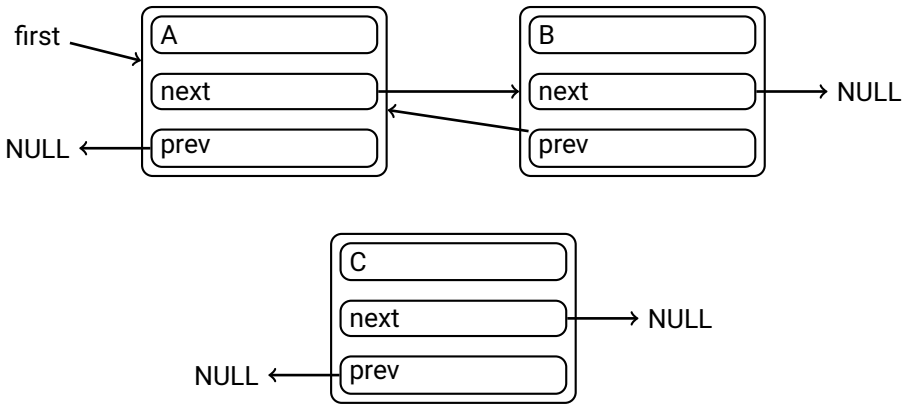
$$\Theta(\_\_\_\_\_)$$

.

7. **Sequenzielle Datenstrukturen** **Punkte: 20**

(a) (8 Punkte) **Verkettete Listen**

Gegeben Sei die Zeichnung der folgenden dynamisch doppelt-verketteten Liste:



Definiert sei `ListElem` als Klasse eines einzelnen Elements.

`ListElem` definiert die Funktionen `getPrev()` und `getNext()` mit dem Rückgabetyp `ListElem*`, das den Previous- bzw. Next-Pointer des Elements zurückgibt, sowie `setPrev(ListElem* elem)` und `setNext(ListElem* elem)`, um diesen Pointer auf `elem` zu setzen.

Vervollständigen Sie die folgende Funktion, um das Element C zwischen A und B einzufügen. Verwenden Sie die C++-Schreibweise für Aufrufe (`a.b();` etc.).

1. `ListElem* currentElem = first;`
2. `ListElem* newElem = new ListElem(C); //` das einzufügende Element
3. _____
4. _____
5. _____
6. _____

(b) (8 Punkte) **Suche in einer verketteten Liste**

Im Folgenden sehen Sie vier verschiedene Funktionen. Kreuzen Sie diejenigen Implementierungen an, die verwendet werden können, um für eine einfach verkettete Liste die Funktion `ListElem.getPrevious(ListElem* elem)` umzusetzen.
Ist kein vorheriges Element vorhanden, soll `NULL` zurückgegeben werden.
Das erste Element der Liste ist im Pointer `first` gespeichert. Hat ein Element keinen Nachfolger, ist `getNext() == NULL`.

◯
```
ListElem* getPrevious(ListElem* elem) {
    ListElem* curr = first;
    while(curr->getNext() != NULL) {
        if (curr == elem) {
            return curr;
        }
        curr = curr->getNext();
    }
}
```

◯
```
ListElem* getPrevious(ListElem* elem) {
    ListElem* curr = first;
    ListElem* result = NULL;
    while (curr->getNext() != NULL) {
        if (curr->getNext() == elem) {
            result = curr;
        }
        curr = curr->getNext();
    }
    return result;
}
```

◯
```
ListElem* getPrevious(ListElem* elem) {
    ListElem* curr = first;
    while (curr->getNext() != NULL) {
        if (curr->getNext() == elem) {
            return curr;
        }
        curr = curr->getNext();
    }
    return NULL;
}
```

◯
```
ListElem* getPrevious(ListElem* elem) {
    ListElem* curr = first;
    while (curr->getNext() != NULL) {
        if (curr->getNext() == elem) {
            return elem;
        } else {
            return NULL;
        }
        curr = curr->getNext();
    }
}
```

(c) (4 Punkte) **Queues**

i. Gegeben sei die folgende Queue mit $n = 8$, implementiert mit dem ringförmigen n-elementigen Array wie in der Vorlesung, speichert also 7 Elemente, mit Head- und Tail-Zeiger auf das Element am angegebenen Index (nullbasiert).
Fügen Sie in die Queue nacheinander die Elemente 5, 3, 4 und 7 ein (enqueue). Entfernen Sie dann im letzten Schritt den aktuellen Head aus der Queue (dequeue). Geben Sie für jeden Schritt den aktuellen Inhalt des Arrays und den Index der aktuellen Head- und Tail-Zeiger nach dem Durchführen des Scrittes an.

| 3 | 2 | 1 | 8 | 7 | 6 | 5 | 4 | Head: 5 Tail: 6

| _____ | _____ | _____ | _____ | _____ | _____ | _____ | _____ | Head: _____ Tail: _____

| _____ | _____ | _____ | _____ | _____ | _____ | _____ | _____ | Head: _____ Tail: _____

| _____ | _____ | _____ | _____ | _____ | _____ | _____ | _____ | Head: _____ Tail: _____

| _____ | _____ | _____ | _____ | _____ | _____ | _____ | _____ | Head: _____ Tail: _____

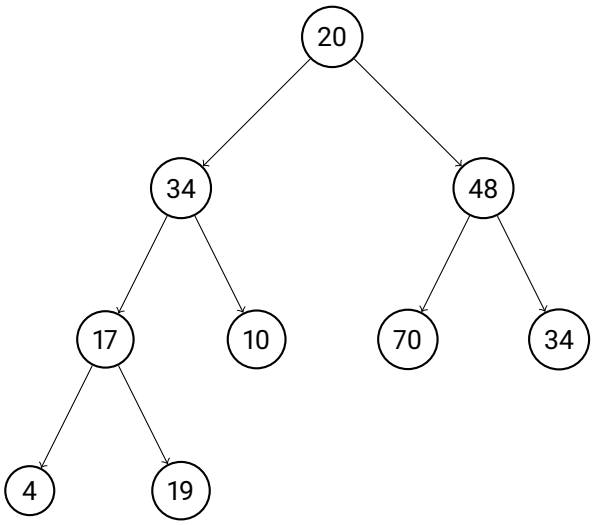
| _____ | _____ | _____ | _____ | _____ | _____ | _____ | _____ | Head: _____ Tail: _____

ii. Eine Queue wird mit einem n-elementigen Array umgesetzt, speichert jedoch nach unserer ringförmigen Implementierung nur $n - 1$ Elemente. Erklären Sie, warum der zusätzliche Platz im Array nötig ist.

8. **Bäume** **Punkte: 20**

(a) (6 Punkte) **Traversierung**

Geben Sie die In-, Pre- und Post-Order-Traversierung des folgenden Binärbaumes an.



In-Order: _____

Pre-Order: _____

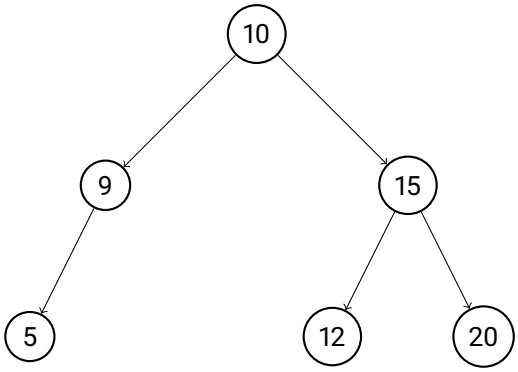
Post-Order: _____

(b) (2 Punkte) **Binäre Suchbäume**

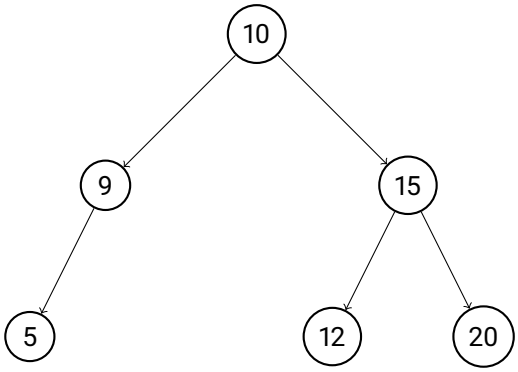
Beim Einfügen in einen binären Suchbaum spielt die Reihenfolge eine Rolle.

◯ Wahr ◯ Falsch

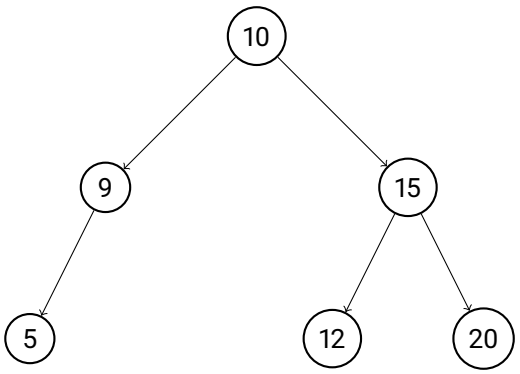
(c) (4 Punkte) **Einfügen in Binären Suchbaum**
Fügen Sie in den folgenden Binären Suchbaum nacheinander die Zahlen 13 und 25 ein. Zeichnen Sie den Baum neu.



(d) (2 Punkte) **Löschen aus Binären Suchbaum**
Löschen Sie aus dem folgenden Baum die Zahl 9.
Zeichnen Sie den Baum neu.



(e) (6 Punkte) Löschen Sie aus dem folgenden Baum die Zahl 10.
Zeichnen Sie zwei mögliche Lösungen.

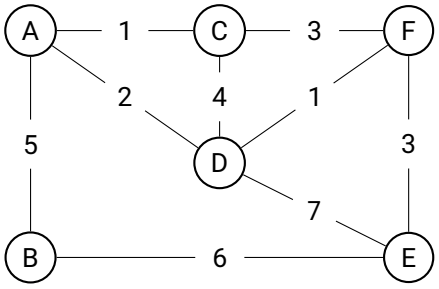


9. **Graphentheorie** **Punkte: 20**

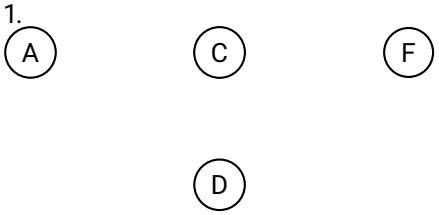
Achtung! *Diese Aufgabe richtet sich nur an Studierende des Studiengangs Informatik (10LP). Studierende des Studiengangs Elektrotechnik (5LP) müssen diese Aufgabe nicht bearbeiten. Es gibt auch keine Zusatzpunkte, sollte die Aufgabe dennoch bearbeitet werden.*

(a) (10 Punkte) **Minimaler Spannbaum**

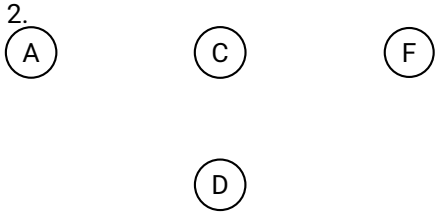
Gegeben sei der folgende ungerichtete Graph. Erstellen Sie einen minimalen Spannbaum des Graphen durch Anwendung des **Algorithmus nach Prim**. Benutzen Sie A als Startknoten. Gehen Sie schrittweise vor und zeichnen Sie den Graphen in jedem Schritt. Beschriften Sie die Kanten mit ihrer jeweiligen Gewichtung.



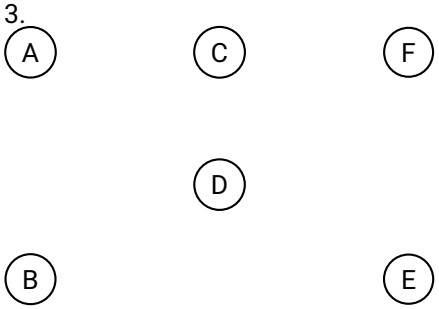
1.



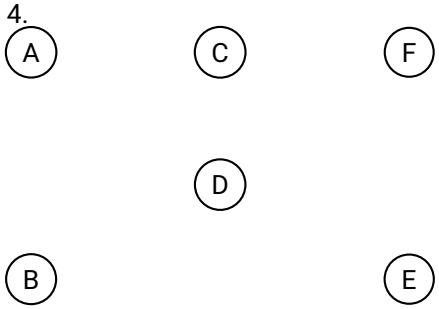
2.



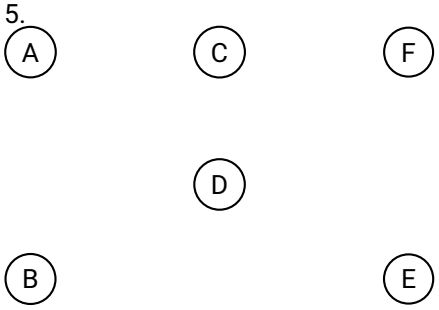
3.



4.

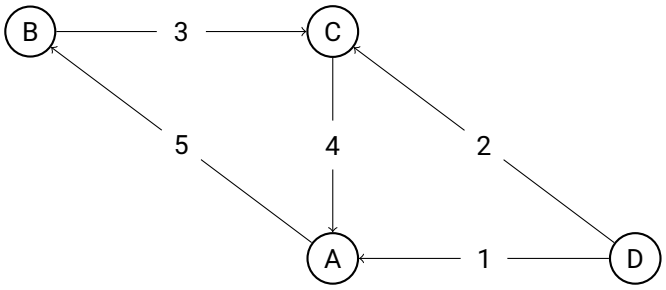


5.

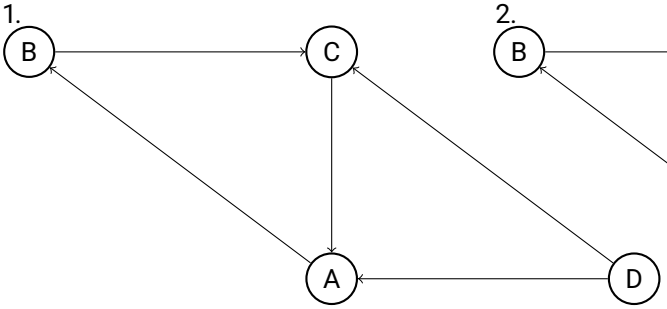


(b) (10 Punkte) **Kürzeste Verbindung zwischen Knoten**

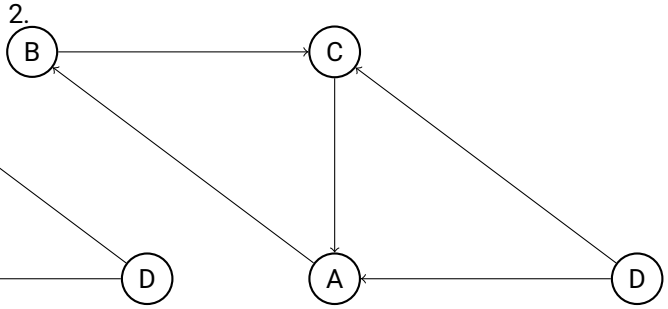
Gegeben sei der folgende gerichtete Graph. Berechnen Sie alle kürzesten Verbindungen zwischen den Knoten mit dem **Algorithmus nach Floyd**. Gehen Sie schrittweise vor und zeichnen Sie den Graphen in jedem Schritt. Beschriften Sie die Kanten mit ihrer jeweiligen Gewichtung.



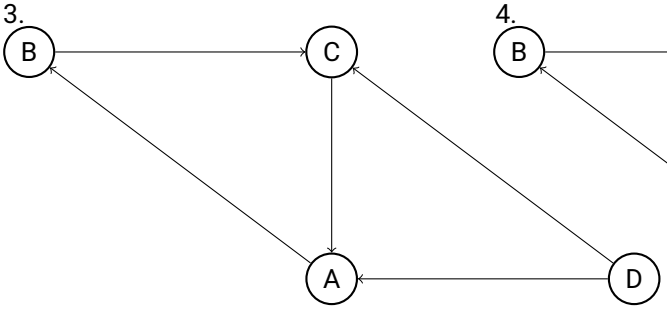
1.



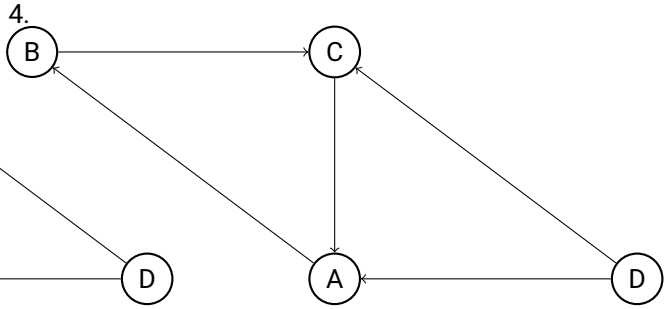
2.



3.



4.



10. **Suchen & Sortieren** **Punkte: 20**

(a) (3 Punkte) **Selbstanordnende Listen**

Nennen Sie die drei Ordnungsstrategien, mit denen man eine selbstanordnende Liste umsetzen kann.

- _____
- _____
- _____

(b) (8 Punkte) **Hashing**

Gegeben sei die folgende Liste an Früchten mit zugehörigen Keys.

Frucht	Apfel	Birne	Banane	Traube	Erdbeere	Kirsche	Pfirsich	Orange
Key	10	4	13	12	27	33	63	70

Ordnen Sie die Früchte der Reihe nach mit der gegebenen Hash-Funktion $h(k)$ und der Sondierungsfunktion $s(i,k)$ in das durch die folgende Tabelle definierte Array ein. Geben Sie auch die Anzahl der Sondierungsschritte i an, die zum Einfügen des jeweiligen Elementes nötig waren.

Index	0	1	2	3	4	5	6	7
Frucht	_____	_____	_____	_____	_____	_____	_____	_____
Key	_____	_____	_____	_____	_____	_____	_____	_____
i	_____	_____	_____	_____	_____	_____	_____	_____

$h(k) = 2k \bmod 8$
$s(i,k) = (2k+i) \bmod 8$

(c) (9 Punkte) **Sortieralgorithmen**
Führen Sie auf dem gegebenen Wort "SELECTION" einen **Insertion-Sort** durch.

i=0	S	E	L	E	C	T	I	O	N
i=1									
i=2									
i=3									
i=4									
i=5									
i=6									
i=7									
i=8									
i=9									
i=10									
i=11									