Name:
Vorname:
Matrikelnummer:

Klausur-ID:

Karlsruher Institut für Technologie
Institut für Theoretische Informatik

Prof. Dennis Hofheinz

5. Oktober 2016

Klausur Algorithmen I

Aufgabe 1.	Kleinaufgaben	11 Punkte
Aufgabe 2.	Sortieren	15 Punkte
Aufgabe 3.	Verlosungen	11 Punkte
Aufgabe 4.	Pfade in Graphen	15 Punkte
Aufgabe 5.	Minimale Spannbäume	8 Punkte

Bitte beachten Sie:

- Bringen Sie den Aufkleber mit Ihrem Namen, Ihrer Matrikelnummer und Ihrer Klausur-ID oben links auf dem Deckblatt an.
- Merken Sie sich Ihre Klausur-ID und schreiben Sie auf **alle Blätter** der Klausur und Zusatzblätter Ihre Klausur-ID und Ihren Namen.
- Die Klausur enthält 17 Blätter. Die Bearbeitungszeit beträgt 120 Minuten.
- Die durch Übungsblätter gewonnenen Bonuspunkte werden erst nach Erreichen der Bestehensgrenze hinzugezählt. Die Anzahl der Bonuspunkte entscheidet nicht über das Bestehen der Klausur.
- Als Hilfsmittel ist ein beidseitig handbeschriebenes DIN-A4 Blatt zugelassen.
- Bitte kennzeichnen Sie deutlich, welche Aufgabe gewertet werden soll. Bei mehreren angegebenen Möglichkeiten wird jeweils die schlechteste Alternative gewertet.

Aufgabe	1	2	3	4	5	Summe
max. Punkte	11	15	11	15	8	60
Punkte						

Bonuspunkte:	Summe:	**Note:**

Aufgabe 1. Kleinaufgaben [11 Punkte]

Bearbeiten Sie die folgenden Aufgaben. Begründen Sie Ihre Antworten jeweils kurz. *Reine Ja/Nein-Antworten ohne Begründung geben keine Punkte.*

a. Ein Algorithmus besitzt *polynomielle Zeitkomplexität*, wenn die Laufzeit im Worst Case durch ein Polynom in der Eingabegröße beschränkt werden kann. Für die meisten Sortieralgorithmen hat sich eine Zeitkomplexität von $\mathrm{O}(n \log n)$ ergeben. Offensichtlich ist $n \log n$ kein Polynom. Haben diese Sortieralgorithmen dennoch polynomielle Zeitkomplexität? Begründen Sie Ihre Antwort kurz. [1 Punkt]

b. Was ist der Unterschied zwischen Linear Programs (LP) und Integer Linear Programs (ILP)? Welche von beiden sind im Allgemeinen einfacher zu lösen? Begründen Sie Ihre Antwort kurz. [1 Punkt]

c. Beweisen oder widerlegen Sie: $2^{\log_{10}(5n)} \in \mathrm{O}(n)$. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

d. Sie möchten Zahlen aus $\mathbb{Q}^+$, die durch Zähler und Nenner repräsentiert vorliegen, sortieren, und dabei eine Implementierung Ihrer Kollegen möglichst unverändert und ohne weitere Vorberechnungen wiederverwenden. Einer Ihrer Kollegen hat bereits LSD-Radix-Sort implementiert, ein anderer InsertionSort, und ein dritter QuickSelect. Welche Implementierung wählen Sie, und warum? [2 Punkte]

e. Beweisen oder widerlegen Sie: Sei für eine Konstante $k > 1$

$$T(n) = \begin{cases} 1, & \text{falls } n = 1 \\ (k-1) \cdot T(n/k) + k \log_2 n, & \text{sonst.} \end{cases}$$

Dann gilt $T(n) \in \mathrm{O}(n)$. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 1

f. Gegeben sei die Hashfunktion $h(x) = 4x + 7 \mod 10$.
Im Folgenden betrachten wir Hashing mit linearer Suche und Hashing mit doppelt verketteten Listen. Bei Hashing mit linearer Suche werden am Ende der Tabelle zwei Einträge m_1 und m_2 angehängt, die als zusätzliche Pufferplätze dienen. Beim Hashing mit verketteten Listen sollen die Einträge jeweils am Ende der jeweiligen Listen angehängt werden. Die Kästchen, die die Listeneinträge darstellen, sind nicht eingezeichnet und müssen von Ihnen ergänzt werden.
Fügen Sie die Werte $12, 5, 4, 14, 10, 9$ mittels der Hashfunktion h in der angegebenen Reihenfolge ein. Wenn Sie nicht die vorgedruckten Tabellen benutzen, machen Sie kenntlich, in welcher Tabelle Sie welches Verfahren verwendet haben. Als Hilfe geben wir Ihnen hier die Hashwerte der einzufügenden Elemente an:

$$h(12) = 5, \quad h(5) = 7, \quad h(4) = 3, \quad h(14) = 3, \quad h(10) = 7, \quad h(9) = 3.$$

[2 Punkte]

Hashing mit linearer Suche:

0	1	2	3	4	5	6	7	8	9	m_1	m_2

Hashing mit verketteten Listen:

0	1	2	3	4	5	6	7	8	9

g. Ausgehend von der im vorigen Aufgabenteil **f.** angefertigten Tabelle, löschen Sie aus der Hashtabelle mit **linearer Suche** den Wert 14. Geben Sie die resultierende Tabelle erneut an.
[1 Punkt]

0	1	2	3	4	5	6	7	8	9	m_1	m_2

Aufgabe 2. Sortieren [15 Punkte]

Betrachten Sie folgenden Algorithmus, der ein Array $A[1 \ldots A.\text{length}]$ als Eingabe erhält:

```
Procedure examplesort(A: Array of ℝ)
    p = 1
    while p ≤ A.length do
        // Geben Sie den Zustand in dieser Zeile aus.
        if p = 1 then
            p = p + 1
        else if A[p] ≥ A[p − 1] then
            p = p + 1
        else
            tausche A[p] und A[p − 1]
            p = p − 1
```

a. Gegeben sei das Eingabearray $A = [2, 1, 4, 5, 3]$. Geben Sie A jeweils *am Anfang* jedes Schleifendurchlaufs wieder (siehe Kommentar im Pseudocode oben), bis der Algorithmus endet. Machen Sie jeweils die Position p durch Umkreisen der entsprechenden Kästchennummer kenntlich (wie im Beispiel).
Hinweis: Die Anzahl der Kästchen entspricht nicht unbedingt der Anzahl der tatsächlich notwendigen Schleifendurchläufe, lassen Sie gegebenenfalls die überflüssigen Kästchen leer. Falls Sie einen Fehler machen, streichen Sie die Kästchen deutlich durch und nehmen Sie den nächsten Block von Kästchen. Tragen Sie in die Überschriften jeweils ein, welche Schleifendurchläufe Sie in der Zeile bearbeitet haben. Falls Sie keinen Fehler machen also gerade 1. bis 3. Schleifendurchlauf, 4. bis 6. Schleifendurchlauf, 7. bis 9. Schleifendurchlauf und so weiter. [3 Punkte]

1. bis ___. Schleifendurchlauf:

1	2	3	4	5		1	2	3	4	5		1	2	3	4	5
2	1	4	5	3												

___. bis ___. Schleifendurchlauf:

1	2	3	4	5		1	2	3	4	5		1	2	3	4	5

___. bis ___. Schleifendurchlauf:

1	2	3	4	5		1	2	3	4	5		1	2	3	4	5

___. bis ___. Schleifendurchlauf:

1	2	3	4	5		1	2	3	4	5		1	2	3	4	5

___. bis ___. Schleifendurchlauf:

1	2	3	4	5		1	2	3	4	5		1	2	3	4	5

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 2

```
1: Procedure examplesort(A: Array of ℝ)
2:     p = 1
3:     while p ≤ A.length do
#:         Invariante:______________________________
4:         if p = 1 oder A[p] ≥ A[p − 1] then
5:             p = p + 1
6:         else
7:             tausche A[p] und A[p − 1]
8:             p = p − 1
```

b. Geben Sie in Zeile # eine Schleifen-Invariante an, die die Korrektheit des Algorithmus impliziert. Beweisen Sie die Invariante, d.h. zeigen Sie, dass die Invariante vor und am Anfang jedes Schleifendurchlaufs gilt.
Wenn Sie mit den Werten von p und A während eines Schleifendurchlaufes argumentieren, verwenden Sie bitte die Bezeichnungen p_{alt} und A_{alt} für die Werte vor der Veränderung im Schleifendurchlauf, und p_{neu} bzw. A_{neu} für die Werte nach der Veränderung.

[4 Punkte]

c. Was müssten Sie weiter zeigen, damit die Korrektheit des Algorithmus folgt?
Hinweis: Der eigentliche Beweis ist hier nicht gefordert. [1 Punkt]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 2

d. Geben Sie eine Funktion f an, so dass die Laufzeit des Algorithmus *examplesort* auf einer Eingabe der Länge n in *Best Case* in $\Theta(f)$ ist. Begründen Sie Ihre Antwort kurz. Geben Sie außerdem ein Array der Länge 5 an, so dass der Algorithmus mit dieser Eingabe die optimale Laufzeit erreicht. [2 Punkte]

e. Geben Sie eine Funktion f an, so dass die Laufzeit des Algorithmus *examplesort* auf einer Eingabe der Länge n im *Worst Case* in $\Theta(f)$ ist. Begründen Sie Ihre Antwort kurz. Geben Sie außerdem ein Array der Länge 5 an, so dass der Algorithmus mit dieser Eingabe die schlechteste Laufzeit erreicht. [2 Punkte]

f. Für welche Art von Eingaben würden Sie Insertionsort zur Sortierung empfehlen und warum? [1 Punkt]

g. Betrachten Sie die Algorithmen Insertionsort, Mergesort und Quicksort aus der Vorlesung. Was sind jeweils die Laufzeiten im Best Case und im Worst Case (im O-Kalkül)? [2 Punkte]

Laufzeit:

	Best Case	Worst Case
Insertionsort		
Mergesort		
Quicksort		

Aufgabe 3. Verlosungen [11 Punkte]

Sie veranstalten eine Verlosung mit n Teilnehmern. Diese läuft wie folgt ab: Jeder Teilnehmer (der durch seinen Namen[1] identifiziert wird) tippt eine natürliche Zahl in $[1, 1000]$ und teilt Ihnen diese verdeckt mit. Nachdem dies alle Teilnehmer getan haben, generieren Sie eine natürliche Zufallszahl z ebenfalls aus $[1, 1000]$. Insgesamt steht als Preisgeld der Betrag $G \in \mathbb{N}$ zur Verfügung. Derjenige Teilnehmer, dessen Zahl am nächsten an z gelegen ist, bekommt $\frac{1}{2}G$ ausgezahlt. Der Teilnehmer mit der zweitnächsten Zahl bekommt $\frac{1}{4}G$ ausgezahlt, der nächste $\frac{1}{8}G$, und so weiter.[2]
Sie dürfen davon ausgehen, dass keine zwei Teilnehmer so tippen, dass sie denselben Abstand vom von Ihnen gezogenen Wert haben. Außerdem haben keine zwei Teilnehmer denselben Namen. Es gilt weiterhin $G = 2^k$ und $k \geq n$.
Hinweis: Die im folgenden gestellten Probleme lassen sich auch asymptotisch schneller als gefordert lösen. Schnellere Lösungen geben aber keine Bonuspunkte.

a. Sie haben die Tipps von allen Teilnehmern als Paare der Form $(Name, Tipp)$ erhalten:

$$(Ford, 250), (Cormen, 700), (Karatsuba, 100), (Dijkstra, 500), (Prim, 800)$$

Sie ziehen $z = 500$, und es steht der Geldbetrag $G = 1024$ Euro zur Verfügung. Welche Auszahlung erhalten die Teilnehmer jeweils? [1 Punkt]

Ford	Cormen	Karatsuba	Dijkstra	Prim

[1] Namen sind Strings, die nur aus Buchstaben eines endlichen Alphabets bestehen und maximal 100 Zeichen lang sind.
[2] Es ist richtig, dass so niemals der gesamte Betrag G ausgezahlt wird. Das sollte Sie nicht weiter stören.

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 3

b. Entwerfen Sie einen Algorithmus, der nachdem Sie die Zufallszahl z generiert haben, im Worst Case mit asymptotischer Zeitkomplexität $O(n \log n)$ die Auszahlung **für jeden Teilnehmer** berechnet, wobei n die Anzahl der Teilnehmer ist. Ihr Algorithmus erhält als Eingabe eine Liste von Paaren der Form

$$(Name, Tipp)$$

und soll eine Liste von Paaren der Form

$$(Name, Betrag)$$

ausgeben, wobei *Betrag* dem Auszahlungsbetrag an den entsprechenden Spieler entspricht. Die Reihenfolge der ausgegebenen Paare ist unerheblich.
Hinweis: Sie dürfen Algorithmen, die aus der Vorlesung bekannt sind, als implementiert annehmen und für Ihre Lösung verwenden. [3 Punkte]

c. Begründen Sie, weshalb Ihr Algorithmus aus **b.** die geforderte Zeitkomplexität erreicht. [1 Punkt]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 3

d. Nun veranstalten Sie nicht eine solche Verlosung, sondern k Verlosungen parallel. Nicht jeder Teilnehmer nimmt an jeder Verlosung teil, und die Verlosungen sind vollkommen voneinander unabhängig, d.h. pro Verlosung gibt eine Teilmenge aller Teilnehmer jeweils eine Zahl ab, und Sie ziehen auch eine Zufallszahl pro Verlosung. Ein Teilnehmer, der an einer Verlosung nicht teilnimmt, erhält für diese auch keine Auszahlung. Für die i-te Verlosung liegen die Tipps nun als Liste von Tripeln der Form

$$(i, Name, Tipp)$$

vor. Erweitern Sie Ihren Algorithmus so, dass er je die **Gesamtauszahlung für jeden Teilnehmer** in erwarteter Zeitkomplexität $\mathrm{O}(kn\log n)$ berechnet.[3] Ihr Algorithmus soll schließlich eine Liste von Paaren der Form

$$(Name, Gesamtbetrag)$$

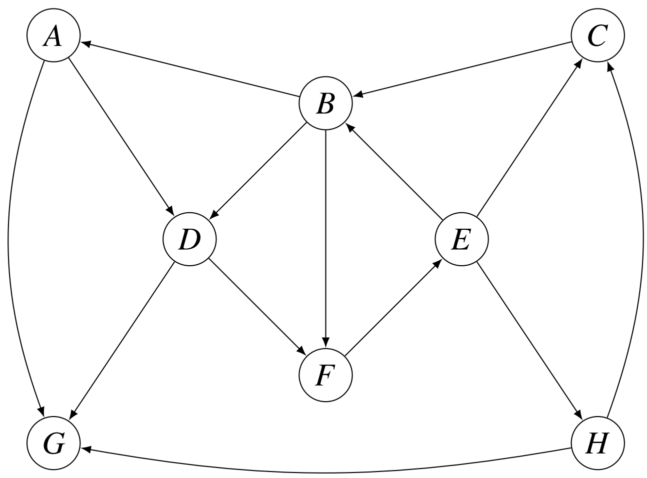
ausgeben, wobei der Gesamtbetrag hier der Summe der Beträge aller Auszahlungen an die entsprechende Person entspricht. Die Reihenfolge der ausgegebenen Paare ist unerheblich. *Hinweis:* Sie dürfen Algorithmen, die aus der Vorlesung bekannt sind, als implementiert annehmen und für Ihre Lösung verwenden. [4 Punkte]

e. Begründen Sie, weshalb Ihr Algorithmus aus **d.** die geforderte Zeitkomplexität erreicht. [2 Punkte]

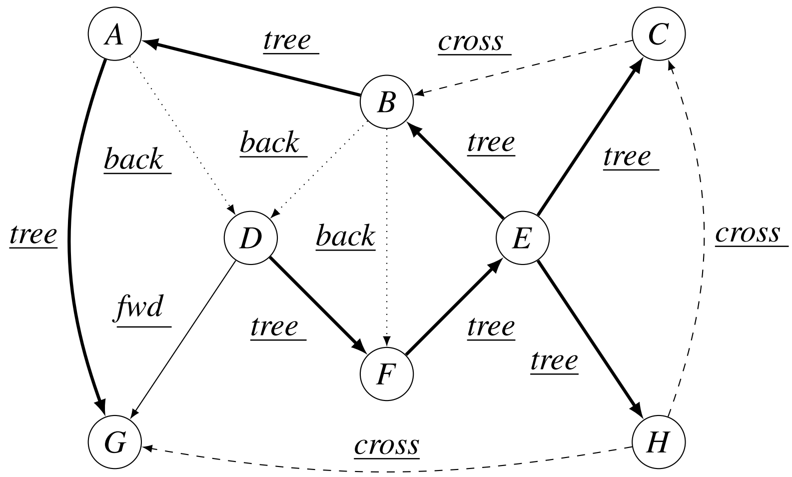
[3] Zur Erinnerung: n ist die Anzahl der Teilnehmer, k die Anzahl der parallelen Verlosungen.

Aufgabe 4. Pfade in Graphen [15 Punkte]

Gegeben sei der folgende ungewichtete, *gerichtete* Graph G:



Weiter sei der folgende Graph G' gegeben, in dem Vorwärts- (*fwd*), Rückwärts- (*back*), Kreuz (*cross*) und Baumkanten (*tree*) eingezeichnet sind:



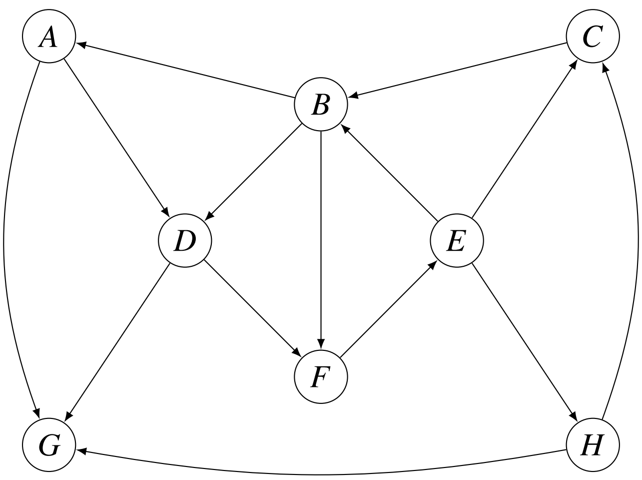
a. Ist der Graph G' das mögliche Ergebnis einer Tiefensuche auf G? Falls ja, geben Sie den jeweiligen Startknoten an, und eine mögliche Reihenfolge der Knoten, in der sie zum ersten Mal vom Suchalgorithmus betrachtet wurden. Falls nein, begründen Sie Ihre Antwort. [2 Punkte]

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 4

b. Ist der Graph G' das mögliche Ergebnis einer Breitensuche auf G? Falls ja, geben Sie den jeweiligen Startknoten an, und eine mögliche Reihenfolge der Knoten, in der sie zum ersten Mal vom Suchalgorithmus betrachtet wurden. Falls nein, begründen Sie Ihre Antwort. [2 Punkte]

c. Geben Sie die Repräsentation von G als Adjazenzfeld an. Sie können dafür die Vorlage verwenden.
Zur Erinnerung ist hier noch einmal der Graph abgedruckt: [2 Punkte]



Adjazenzfeld:

A	B	C	D	E	F	G	H	

1	2	3	4	5	6	7	8	9	10	11	12	13	14

d. Nennen Sie einen Vorteil und einen Nachteil von Adjazenzfeldern gegenüber Adjazenzlisten. [1 Punkt]

Vorteil von Adjazenzfeldern: ______________________________

Nachteil von Adjazenzfeldern: ______________________________

(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 4

e. Es geht nun darum, in einem beliebigen ungewichteten, *ungerichteten* Graphen $G = (V,E)$ Pfade zu finden, die aber explizit **nicht unbedingt kürzeste** Pfade sein müssen. Zusätzlich zum Graphen G sei in dieser Aufgabe mit l auch die Länge[4] des längsten einfachen Pfades in G als Eingabe gegeben. Ein *einfacher* Pfad ist ein Pfad, der jeden Knoten maximal einmal enthält. Ihre Aufgabe ist es, einen Algorithmus zu entwickeln, der für Knotenpaare $\{u,v\}$ *irgendeinen* Pfad zwischen u und v findet. Ihr Algorithmus soll dabei in zwei Phasen vorgehen: In der *Vorberechnungsphase* darf Ihr Algorithmus (innerhalb der unten gegebenen Zeit- und Platzschranken) Daten vorberechnen. In dieser Phase ist dem Algorithmus der Graph bekannt, **nicht aber die später folgenden Anfragen**. In der auf die Vorberechnungsphase folgenden *Anfragephase* bekommt Ihr Algorithmus dann eine Reihe von Anfragen, d.h. von Knotenpaaren $\{u,v\}$, und muss (wieder innerhalb der unten gegebenen Schranken) einen Pfad zwischen beiden Knoten finden. Die Nebenbedingungen für die beiden Phasen lauten:

- In der **Vorberechnungsphase** darf Ihr Algorithmus maximal $\mathrm{O}(|V|+|E|)$ Zeit und maximal $\mathrm{O}(|V|)$ zusätzlichen Speicher verwenden. *Erinnerung:* In dieser Phase sind die Paare $\{u,v\}$, die später angefragt werden, noch nicht bekannt!
- In der **Anfragephase** muss Ihr Algorithmus dann für jedes angefragte Paar $\{u,v\}$ in Zeit $\mathrm{O}(l)$ einen *einfachen* Pfad ausgeben, oder aber ausgeben, dass kein Pfad zwischen u und v existiert.

Geben Sie einen solchen Algorithmus an.
Hinweis: Sie dürfen Algorithmen, die aus der Vorlesung bekannt sind, als implementiert annehmen und für Ihre Lösung verwenden. [6 Punkte]

[4] Das heißt die Anzahl der Kanten auf dem Pfad.

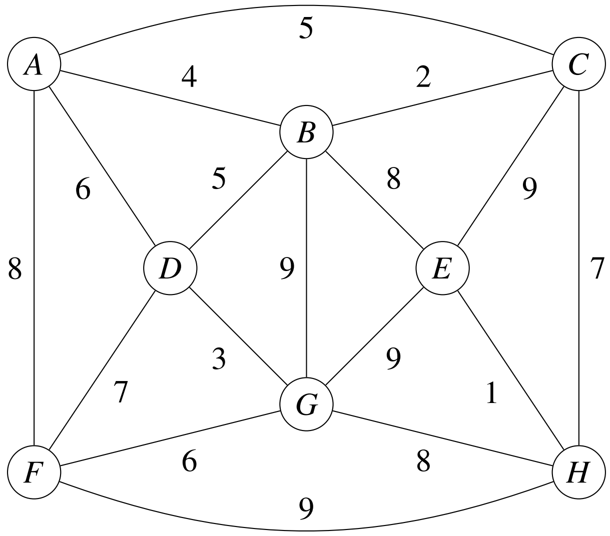
(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 4

f. Begründen Sie, warum Ihr Algorithmus aus **e.** die geforderte Zeitkomplexität erreicht. [2 Punkte]

Aufgabe 5. Minimale Spannbäume [8 Punkte]

a. Berechnen Sie einen minimalen Spannbaum des angegebenen Graphen mit dem Algorithmus von Jarník-Prim. Geben Sie jeweils die Kanten des minimalen Spannbaumes in der Reihenfolge an, in der sie der Algorithmus auswählt. Für Knoten V,W geben Sie die verbindende Kante als $\{V,W\}$ oder (V,W) an. Verwenden Sie den Knoten A als Startknoten. [3 Punkte]



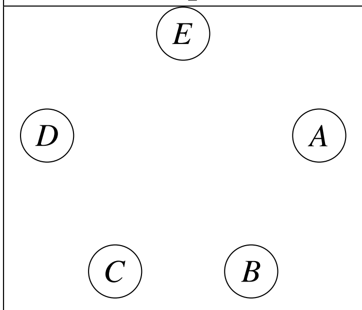
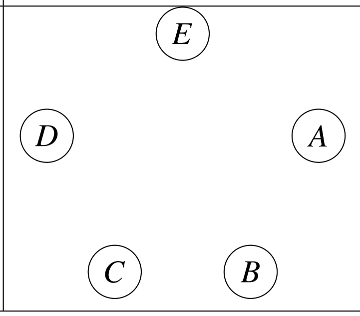
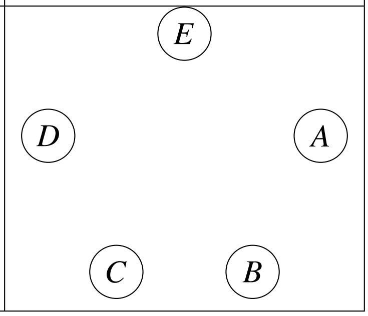
Kantenreihenfolge: ____________________

b. Nennen und erklären Sie die Eigenschaft, auf der die Korrektheit des Jarník-Prim Algorithmus beruht. [1 Punkt]

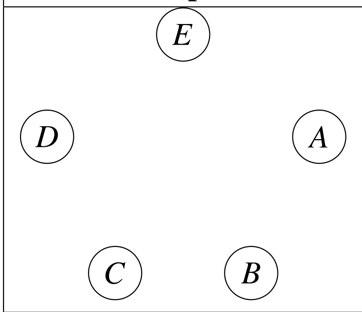
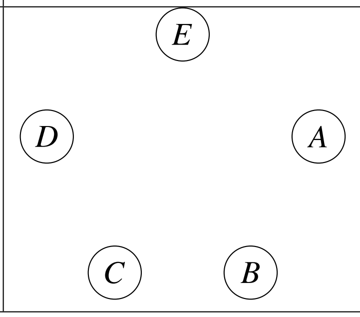
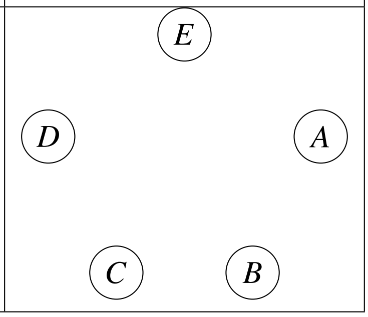
(weitere Teilaufgaben auf den nächsten Blättern)

Fortsetzung von Aufgabe 5

c. Ergänzen Sie im folgenden Graphen ungerichtete Kanten mit positiven Kantengewichten, so dass ein zusammenhängender Graph entsteht, der **genau zwei** verschiedene minimale Spannbäume enthält. Geben Sie links den Graphen und rechts jeweils die minimalen Spannbäume an. Falls Sie einen Fehler machen, können Sie die unteren Vorlagen verwenden. Kennzeichnen Sie *deutlich*, welche Lösung die zu wertende ist, indem Sie die andere durchstreichen. [2 Punkte]

Graph:	**1. MST:**	**2. MST:**
		

Für Korrekturen:

Graph:	**1. MST:**	**2. MST:**
		

d. Gegeben sei ein ungerichteter zusammenhängender zyklenfreier Graph $G = (V, E)$ mit positiven Kantengewichten und $n = |V|$ Knoten. *Zyklenfrei* bedeutet, dass der Graph keinen Zyklus, also keinen Pfad der Länge > 1 mit gleichem Start- und Endknoten, enthält.

d.1 Wieviele Kanten $|E|$ hat der Graph G in Abhängigkeit von n?

d.2 Ist der minimale Spannbaum von G eindeutig?

Begründen Sie Ihre Antworten jeweils kurz. [2 Punkte]

Konzeptpapier für Nebenrechnungen.