TECHNISCHE UNIVERSITÄT BERLIN
Fakultät IV – Elektrotechnik und Informatik
Fachgebiet Robotik (MAR 5-1)
Dozent: Prof. Dr. Oliver Brock
WiMi: Adrian Pfisterer, Alexander Koenig

Algorithmen und Datenstrukturen, SoSe 25

Klausur am 29. Juli 2025

Bitte füllen Sie alle folgenden Felder aus:

Vorname _____

Nachname _____

TUB-Kontoname _____

Matrikelnummer _____

Studiengang _____

Hochschule _____

Durch meine Unterschrift bestätige ich die Korrektheit obiger Angaben sowie meine Prüfungsfähigkeit und die Anmeldung zur Prüfung.

_____ Ort, Datum

_____ Unterschrift

Beachten Sie die folgenden Hinweise!

- Die Klausur dauert **90 Minuten**. Insgesamt können in der Klausur **90 Punkte** erreicht werden.
- Legen Sie nun Ihren Personalausweis und Ihren Studierendenausweis neben sich bereit.
- Diese Klausur besteht mit diesem Deckblatt aus den (nummerierten) Seiten **1-22**.
- Geben Sie nur eine Lösung pro Aufgabe ab, streichen Sie alle alternativen Lösungsansätze durch.
- Notieren Sie Ihre Antworten nur auf dem Blatt (inklusive Rückseite), auf dem die zugehörige Aufgabe steht, da die Aufgaben getrennt korrigiert werden.
- Sie brauchen Ihren Namen **nur** auf das Deckblatt zu schreiben. Die restlichen Blätter können über die Klausur-ID zugeordnet werden.
- Schreiben Sie **nicht** mit roter Farbe, grüner Farbe (Korrekturfarben) oder Bleistift. Diese Lösungen werden nicht bewertet!
- Am Klausurende befindet sich eine Doppelseite, die Sie für Notizen verwenden können. Sollten Sie mehr Papier benötigen, können Sie dies von der Aufsicht bekommen. Notieren Sie in diesem Fall die Klausur-ID auf dem Zusatzblatt und tragen Sie die Anzahl der erhaltenen Zusatzblätter unten ein.
- Falls Sie eine Antwort auf ein Zusatzblatt schreiben, markieren Sie dies klar bei der zugehörigen Aufgabe und auf dem Zusatzblatt.

Anzahl Zusatzblätter:

Punktetabelle

Aufgabe	Punkte		
1	11		
2	10		
3	11		
4	11		
5	11		
6	8		
7	8		
8	10		
9	10		
$\sum$	/ 90		

Aufgabe 1: Bibliothekssystem (8 Punkte)

(a) **2 Punkte**

In dieser Aufgabe entwerfen Sie ein Bibliothekssystem. Definieren Sie dafür eine abstrakte Basisklasse `Book` mit den Attributen `title` (String) und `year` (int), sowie einer abstrakten Methode `getInfo()`, die einen String zurückgibt.

Hinweis: Da keine Getter-Methoden implementiert sind, müssen die Attribute der Basisklasse mit dem korrekten Zugriffsmodifizierer deklariert werden, um den Zugriff für erbende Klassen zu ermöglichen, aber für andere Klassen zu verhindern.

```
1  public ____________________ class Book {
2      ____________________ ____________________ ____________________;
3      ____________________ ____________________ ____________________;
4      public Book(String title, int year) {
5          ____________________.____________________ = ____________________;
6          ____________________.____________________ = ____________________;
7      }
8      ____________________ ____________________ ____________________ getInfo();
9  }
```

(b) **2 Punkte**

Erstellen Sie eine Klasse `Magazine`, die von `Book` erbt und zusätzlich das Attribut `issue` (int) besitzt. Überschreiben Sie die Methode `getInfo()`, sodass sie einen String im Format `Magazin: [title], Issue #[issue] ([year])` zurückgibt. Die eckigen Klammern sollen nicht gedruckt werden.

```
1  public class Magazine ____________________ ____________________ {
2      private int issue;
3      public Magazine(String title, int year, int issue) {
4          ____________________(____________________, ____________________);
5          this.issue = issue;
6      }
7      ____________________
8      public String getInfo() {
9          ____________________________________________________
10         ____________________________________________________;
11     }
12 }
```


(c) **2 Punkte**

Implementieren Sie eine Klasse `Bookshelf`, die beliebige `Book`-Objekte in einer `ArrayList<Book>` verwaltet. Die Klasse soll eine Methode `addBook(Book b)` zum Hinzufügen von Büchern und eine Methode `printAllBooks()` besitzen, die die Informationen aller Bücher mittels ihrer `getInfo()`-Methode ausgibt.

```
1  import java.util.ArrayList;
2  public class Bookshelf {
3      private ArrayList<Book> books;
4      public Bookshelf() {
5          books = ____________________________________________________;
6      }
7      public void addBook(Book b) {
8          books.______________________________________________________;
9      }
10     public void printAllBooks() {
11         for ____________________________________________________ {
12             ____________________________________________________;
13         }
14     }
15 }
```

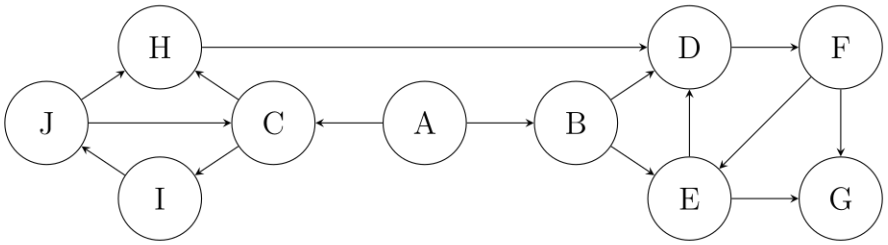
(d) **2 Punkte**

Schreiben Sie eine `main`-Methode, in der ein `Bookshelf` erstellt wird. Fügen Sie anschließend das `Magazin` *Science Weekly* aus dem Jahr 2025 mit der Ausgabennummer 42 hinzu. Geben Sie zum Schluss alle Bücher im Regal über die Konsole aus.

```
1  public class Main {
2      public static void main(String[] args) {
3          Bookshelf shelf = ____________________________________________;
4          Book mag = ___________________________________________________;
5          shelf.______________________________________________________;
6          shelf.______________________________________________________;
7      }
8  }
```


Aufgabe 2: Tiefensuche und Laufzeit (11 Punkte)

Der folgende gerichtete Graph ist gegeben.



Führen Sie für die folgenden Aufgaben eine Handsimulation der rekursiven Tiefensuche aus. Beginnen Sie die Tiefensuche im Knoten A. Gehen Sie dabei davon aus, dass bei Wahlmöglichkeiten (Tiebreak) die benachbarten Knoten in alphabetischer Reihenfolge abgearbeitet werden.

(a) (2 Punkte) Geben Sie die **Nebenreihenfolge** (preorder) an.

Preorder: **A**

(b) (2 Punkte) Geben Sie die **Hauptreihenfolge** (postorder) an.

Postorder:

(c) (1 Punkt) Wie kann mit Hilfe der Tiefensuche erkannt werden, ob in einem gerichteten Graphen ein Zyklus vorhanden ist?

(d) (2 Punkte) Geben Sie zwei Kanten an, die entfernt werden müssen, damit eine topologische Sortierung möglich ist. Begründen Sie Ihre Auswahl.

(e) (4 Punkte) Geben Sie für die folgenden Funktionen die asymptotische Laufzeit als engstmögliche obere Schranke in der ***O***-Notation in Abhängigkeit von N an.

```
public static int f1(int N) {
    int sum = 0;
    for (int i = 0; i < N; i++) {
        for (int j = N; j > 0; j = j / 2) {
            sum++;
        }
    }
    return sum;
}
```

Laufzeit:

```
public static void f2(int N) {
    if (N <= 1) {
        return;
    }
    for (int i = 0; i < N / 2; i++) {
        System.out.println("Hello");
    }
    f2(N / 2);
    f2(N / 2);
}
```

Laufzeit:

```
public static int f3(int N) {
    if (N <= 0) {
        return 1;
    }
    int result = 0;
    for (int i = 1; i <= N * N; i++) {
        result += i;
    }
    return result + f3(N - 1);
}
```

Laufzeit:

```
public static int f4(int[] arr) {
    int N = arr.length;
    int sum = 0;
    for (int i = 0; i < N; i++) {
        for (int j = i; j < N; j++) {
            for (int k = 0; k < 5; k++) {
                sum += arr[i] + arr[j];
            }
        }
    }
    return sum;
}
```

Laufzeit:

Aufgabe 3: Greedy-Algorithmen und Backtracking (12 Punkte)

(a) (3 Punkte) Ordnen Sie die unten stehenden Algorithmus-Typen den richtigen Aussagen zu.

Aussage	beide	Greedy	Back-tracking	keine
Die Lösungsstrategie basiert auf der schrittweisen Konstruktion eines Pfades in einem impliziten Lösungsbaum.	☐	☐	☐	☐
Die zu treffenden Entscheidungen werden durch ein lokales Optimum bestimmt und später nicht mehr revidiert.	☐	☐	☐	☐
Dieser Algorithmus-Typ kann als allgemeine Methode verwendet werden, um eine optimale Lösung zu garantieren, indem der Lösungsraum systematisch durchsucht wird.	☐	☐	☐	☐
Der Algorithmus kann genutzt werden, um alle möglichen Lösungen eines Problems zu finden.	☐	☐	☐	☐
Algorithmen dieser Klasse sind sehr effizient, da ihre Laufzeit typischerweise polynomial ist (z.B. $\boldsymbol{O}(n \log n)$ oder $\boldsymbol{O}(n^2)$)	☐	☐	☐	☐
Der Algorithmus-Typ kann durch Pruning-Techniken effizienter gestaltet werden, ohne die Korrektheit der Lösung zu beeinträchtigen.	☐	☐	☐	☐

(b) (9 Punkte) Bei einer Kanu-Tour auf einem Fluss sollen **möglichst viele** Seitenarme durchfahren werden, da dort gebadet werden kann. Jeder Seitenarm ist durch einen Start- und einen Endpunkt auf dem Hauptfluss definiert (siehe Abbildung 2 auf der nächsten Seite).

Es gelten folgende Regeln:

- Aufgrund der starken Strömung können Sie den Hauptfluss nur in eine Richtung befahren. Der Hauptfluss kann als Zeitachse verstanden werden.
- Ein Seitenarm kann nicht befahren werden, wenn er sich zeitlich mit einem bereits ausgewählten Seitenarm überschneidet. Das heißt, der Startpunkt eines neuen Seitenarms muss nach dem Endpunkt des zuletzt befahrenen liegen.

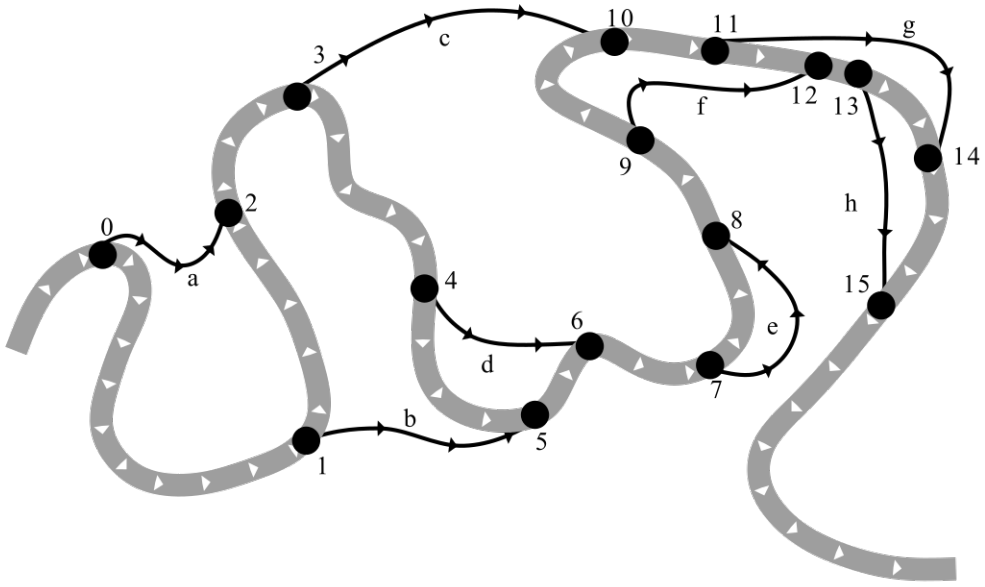
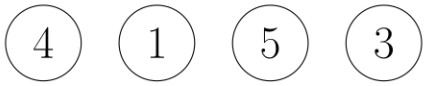


Abbildung 2: Ein Beispiel für den Flussverlauf. Der Hauptfluss ist in hellgrau gekennzeichnet. Seitenarme (a bis h) sind durch schwarze Linien, mit chronologischen Start- und Endzeiten gekennzeichnet.

1. (1 Punkt) Benennen Sie das aus der Vorlesung bekannte Greedy-Problem, auf das sich diese Aufgabenstellung zurückführen lässt.
2. (3 Punkte) Beschreiben Sie schrittweise den Greedy-Algorithmus zur Lösung des Problems. Nennen Sie dabei explizit die Sortier- und die Auswahlstrategie.
3. (5 Punkte) Wenden Sie den beschriebenen Greedy-Algorithmus auf die Seitenarme in der Abbildung an. Geben Sie die finale Reihenfolge der ausgewählten Seitenarme an und bestimmen Sie die maximale Anzahl der durchfahrbaren Seitenarme.

Aufgabe 4: Das Münzspiel (13 Punkte)

Alex und Adrian spielen folgendes Münzspiel. Auf dem Tisch liegt eine Reihe von Münzen. Die folgende Abbildung zeigt die aktuelle Spielstellung mit vier Münzen mit den Werten 4, 1, 5 und 3.



Die Spieler ziehen abwechselnd Münzen aus der Reihe. In jedem Zug darf der aktive Spieler entweder die Münze am linken oder am rechten Ende der Reihe entnehmen. Das Spiel endet, wenn alle Münzen genommen wurden. Das Ziel jedes Spielers ist es, am Ende des Spiels einen möglichst großen Vorsprung an Münzwerten gegenüber dem Gegner zu erzielen.

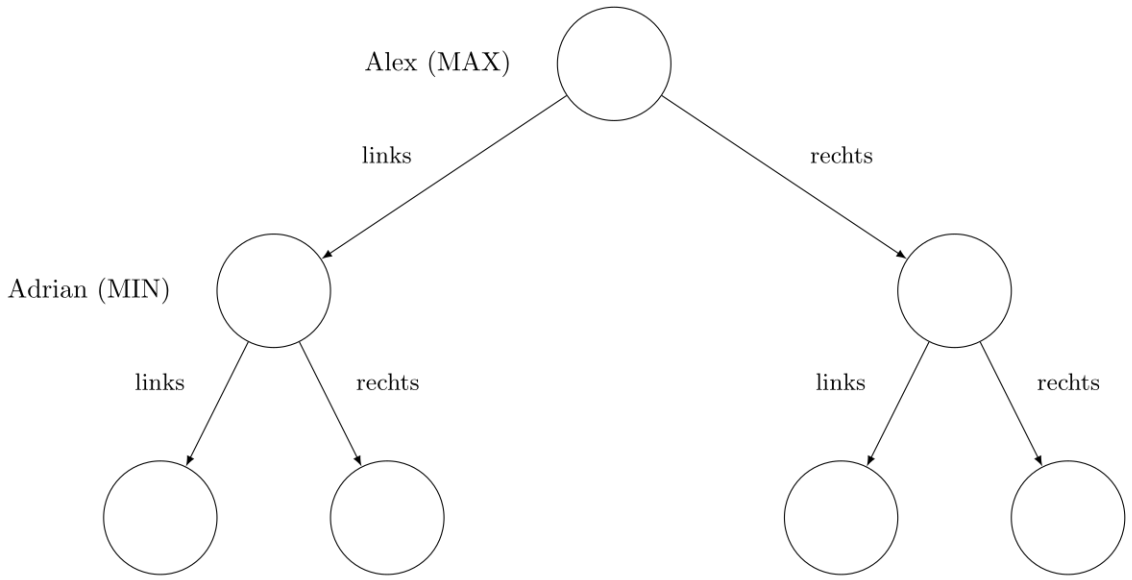
Alex beginnt das Spiel und muss sich entscheiden, ob er die linke Münze (4) oder die rechte Münze (3) nimmt. Er verwendet den **Minimax-Algorithmus**, um seinen optimalen Zug zu finden.

(a) (1 Punkt) Geben Sie die **Bewertungsfunktion** an, die den Vorsprung von Alex gegenüber Adrian berechnet. Alex maximiert diesen Wert, während Adrian ihn minimiert. Geben Sie die Bewertungsfunktion in Abhängigkeit der von Alex und Adrian gesammelten Münzwerte S_{Alex} und S_{Adrian} an.

(b) (3 Punkte) Alex plant seinen Zug nur auf Basis der nächsten beiden Züge (ein Zug von Alex, gefolgt von einem Zug von Adrian). Vervollständigen Sie den folgenden Spielbaum und finden Sie Alex optimalen Zug mit dem Minimax-Algorithmus.

Anleitung:

- Berechnen Sie an den Blattknoten den Wert des Spielzustands mithilfe der Bewertungsfunktion. Tragen Sie diese Werte in die Blattknoten ein.
- Wenden Sie den Minimax-Algorithmus an, um die Werte für alle Elternknoten zu bestimmen. Tragen Sie diese Werte ebenfalls in die Knoten ein.
- Geben Sie unten an, welchen Zug Alex optimalerweise wählt und welchen Wert er damit erreicht.



Alex optimaler Zug:

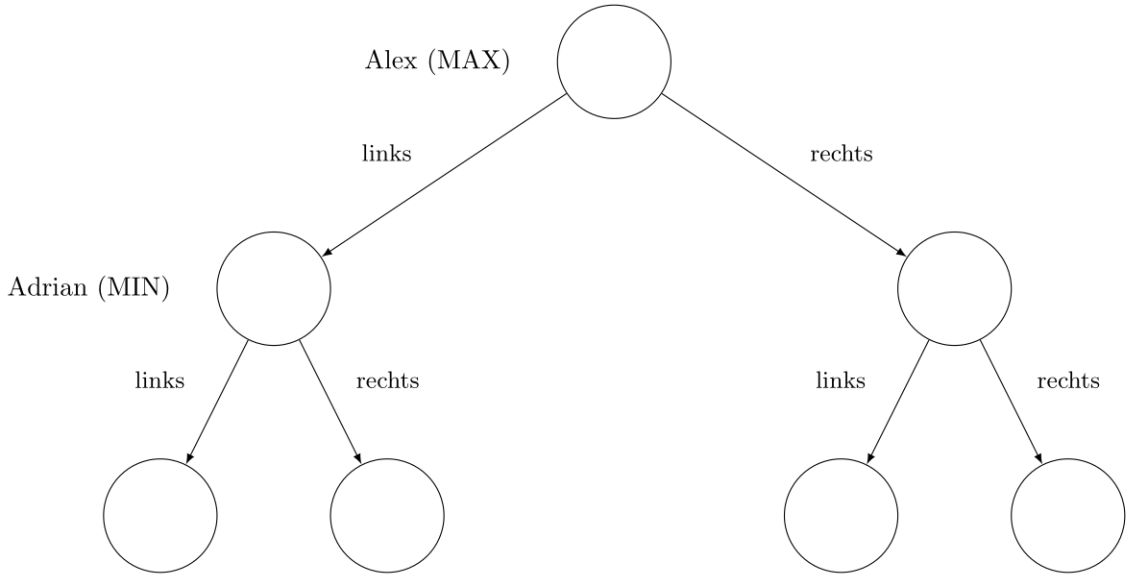
Erreichter Wert:

(c) (1 Punkt) Nennen Sie den in der Vorlesung besprochenen Algorithmus, der die Minimax-Suche durch das Beschneiden von Teilbäumen effizienter macht.

(d) (6 Punkte) Übertragen Sie die Werte der Blattknoten aus Aufgabe (b) in den folgenden Baum. Führen Sie anschließend den in Aufgabe (c) genannten Algorithmus aus.

Hinweise:

- **Beschriften Sie die Kanten mit den Schranken und ihren aktuellen Werten.**
- Verwenden Sie eine Links-nach-Rechts-Auswertungsreihenfolge der Kindknoten.
- Kennzeichnen Sie deutlich, an welcher Stelle der Suchraum verkleinert wird und nennen Sie das Kriterium, das an dieser Stelle erfüllt wird.
- Benennen Sie den Schnitt, durch den der Suchraum reduziert wird, gemäß der in der Vorlesung verwendeten Konvention.



(e) (2 Punkte) Alex und Adrian spielen nun mit einer längeren Münzreihe und planen ihre Züge bis zum Ende des Spiels. Bisher haben sie die möglichen Züge (d.h. die Kindknoten) immer der Reihe nach von links nach rechts ausgewertet. Sie möchten den Spielbaum jedoch so effizient wie möglich durchsuchen.

Nennen Sie eine spezifische Strategie zur Auswahl des nächsten zu untersuchenden Zuges, die anstelle der festen Links-nach-Rechts-Reihenfolge verwendet werden sollte. Begründen Sie kurz, warum diese Strategie die Gesamtanzahl der tatsächlich untersuchten Knoten im Spielbaum reduziert.

Aufgabe 5: Dynamische Programmierung (10 Punkte)

Es ist eine Matrix der Größe $M \times N$ gegeben. Die Einträge der Matrix sind $\in \mathbb{N}_{>0}$. Mit Hilfe dynamischer Programmierung soll ein Weg vom oberen linken zum unteren rechten Feld gefunden werden, der die Summe aller Zahlen auf seinem Weg **minimiert**, wie in der Abbildung unten dargestellt. Bei jedem Schritt darf nur entweder nach rechts oder nach unten gegangen werden, also von Feld (m, n) zu einem der Felder $(m+1, n)$, $(m, n+1)$, sofern diese Felder innerhalb der Matrix liegen.

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

Eine Beispielmatrix der Größe 3×4, in der ein Weg vom oberen linken Feld zum unteren rechten Feld eingezeichnet ist. Der Wert des Weges ist die Summe der durchlaufenen Felder, also $1+3+1+1+1+1=8$.

Ziel dieser Aufgabe ist es, eine rekursive Funktion Opt zu bestimmen, welche es erlaubt, das Problem mithilfe dynamischer Programmierung zu lösen.

(a) (3 Punkte) Tragen Sie in die folgende Matrix die Werte einer Opt-Funktion für jedes Feld (m, n) der Matrix aus der Abbildung oben ein. Dies ist der Wert des günstigsten Pfades vom Startfeld $(1, 1)$ zum jeweiligen Feld.

Hinweis: Sie können diese Aufgabe auch lösen, ohne eine korrekte rekursive Funktion zu kennen.

1			

(b) (4 Punkte) Definieren Sie eine rekursive Funktion $\textsc{Opt}(m, n)$, die den Wert des günstigsten Weges vom Startfeld $(1, 1)$ zum Feld (m, n) berechnet ($1 \leq m \leq M, 1 \leq n \leq N$). Der Wert eines Feldes (m, n) ist $F(m, n)$. Verwenden Sie eine mathematische Schreibweise und keinen Java- oder Pseudo-Code.

(c) (2 Punkte) Geben Sie die Größenordnung der Laufzeit des Algorithmus an, wenn die OPT-Werte mit Hilfe einer dynamischen Programmierungstabelle (DP-Tabelle) berechnet werden. Bestimmen Sie zusätzlich die Größenordnung der Laufzeit eines Brute-Force-Algorithmus, der alle möglichen Wege einzeln betrachtet und deren Summen vergleicht, um den minimalen Weg zu finden. Begründen Sie auch hier Ihre Antworten kurz (jeweils 1-2 Sätze).

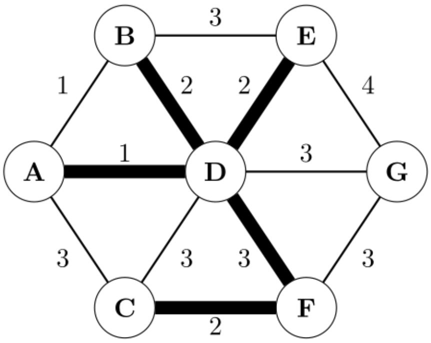
(d) (1 Punkt) Wie verändert sich die Laufzeit des Algorithmus mit DP-Tabelle, wenn nun auch Doppelsprünge erlaubt sind? Das bedeutet, man darf vom Feld (m, n) nicht nur zu den Feldern $(m+1, n)$ und $(m, n+1)$, sondern zusätzlich auch zu $(m+2, n)$ und $(m, n+2)$ springen. Dabei darf weiterhin nur nach rechts und nach unten gegangen werden. Begründen Sie Ihre Antwort kurz (1-2 Sätze).

Aufgabe 6: Minimale Spannbäume (7 Punkte)

(a) (3 Punkte) Lesen Sie die folgenden Aussagen. Entscheiden Sie, ob sie nur für den Prim-Algorithmus, nur für den Kruskal-Algorithmus, für beide oder für keinen der beiden Algorithmen gelten. Kreuzen Sie das passende Feld an. Pro Aussage ist nur eine Antwort richtig.

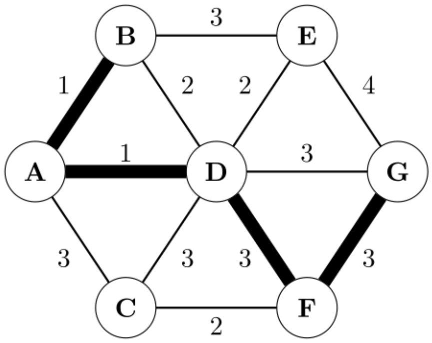
Aussage	beide	nur Prim	nur Kruskal	keiner
Der Algorithmus dient der Berechnung eines minimalen Spannbaumes in einem zusammenhängenden, ungerichteten, kantengewichteten Graphen.	☐	☐	☐	☐
Der Algorithmus baut schrittweise einen Spannbaum auf, indem er immer die Kante mit dem minimalen Gewicht auswählt, die den aktuellen Baum mit einem noch nicht besuchten Knoten verbindet und dabei keinen Zyklus erzeugt.	☐	☐	☐	☐
Der resultierende minimale Spannbaum ist immer eindeutig.	☐	☐	☐	☐
Der resultierende Teilgraph nach Beendigung des Algorithmus ist maximal azyklisch.	☐	☐	☐	☐
Für die effiziente Implementierung des Algorithmus ist die Benutzung der Union-Find zur dynamischen Verwaltung von Zusammenhangskomponenten essentiell.	☐	☐	☐	☐
Die Folge der zum Spannbaum hinzugefügten Kantengewichte ist garantiert nicht-absteigend.	☐	☐	☐	☐

(b) (4 Punkte) In der folgenden Abbildung sehen Sie vier mal den **gleichen** Graphen, wobei jedes mal unterschiedliche Kanten schwarz hervorgehoben sind. Kreuzen Sie an, ob die hervorgehobenen Kanten bei der Suche nach einem minimalen Spannbaum nur durch den Prim Algorithmus, nur durch den Kruskal Algorithmus, durch beide oder durch keinen von beiden ausgewählt worden sein können. Dabei muss der jeweilige Algorithmus **nicht bis zum Ende durchgelaufen sein**; es können also auch partielle Lösungen dargestellt sein.



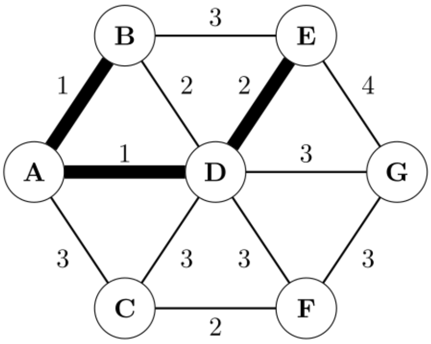
☐ beide ☐ nur Kruskal

☐ nur Prim ☐ keiner



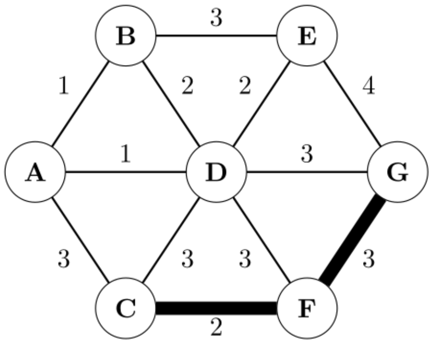
☐ beide ☐ nur Kruskal

☐ nur Prim ☐ keiner



☐ beide ☐ nur Kruskal

☐ nur Prim ☐ keiner



☐ beide ☐ nur Kruskal

☐ nur Prim ☐ keiner

Aufgabe 7: Kürzeste Pfade (11 Punkte)

Wir betrachten eine Klaviertastatur mit 3 Tasten, die unterschiedliche Töne erzeugen. Die unterschiedlichen Töne nennen wir c, d und e. Wir definieren eine *Melodie* als Abfolge von Tönen.
In dieser Aufgabe suchen wir die *schönste Melodie*, die aus genau 3 Tönen besteht. Für verschiedene Tonübergänge gibt es die in Tabelle 2a angegebenen Kosten. Die Melodie mit der geringsten Summe an Kosten gilt als die *schönste Melodie*. Zusätzlich zu den Übergängen fallen Kosten für den ersten und letzten Ton der Melodie an. Diese sind in Tabelle 2b angegeben.

	zu c	**zu** d	**zu** e
von c	3	5	kein Übergang
von d	1	3	2
von e	kein Übergang	2	3

(a) Kosten für die Tonübergänge.

Ton	c	d	e
Kosten	4	3	1

(b) Kosten für den Start- und Endton

Tabelle 2: Kosten für das Spielen von Tönen.

Beispiel: Die Melodie $c \to d \to e$ hat die Kosten $4 + 5 + 2 + 1 = 12$. Sie startet mit c (Kosten 4), dann folgen zwei Übergänge mit den Kosten 5 und 2. Schließlich endet die Melodie auf e, was zu weiteren Kosten von 1 führt.

(a) (2 Punkte) Modellieren Sie den oben dargestellten Sachverhalt im Graphen auf der nächste Seite. Tragen Sie dazu die korrekten Gewichte an den entsprechenden Kanten ein.

(b) (6 Punkte) Führen Sie eine Handsimulation des Dijkstra-Algorithmus durch, um einen kürzesten Weg von Knoten s nach t zu finden. Füllen Sie dazu die Tabelle auf der nächsten Seite aus und geben Sie für jeden Schritt den ausgewählten Knoten, seine Pfadlänge von s und seinen Vorgänger an.

Hinweis: Haben mehrere Knoten in der Priority Queue die gleiche minimale Distanz, ist die Auswahl in der folgenden Reihenfolge vorzunehmen:

1. Wählen Sie zuerst den alphabetisch kleinsten Knoten (z.B. c_2 vor d_1).
2. Sind Knoten alphabetisch gleich, wählen Sie den Knoten mit dem kleineren numerischen Index (z.B. c_1 vor c_2).

(c) (1.5 Punkte) Wir fügen nun eine weitere Kante $e_3 \to e_1$ in den gezeigten Graphen ein. Geben Sie das kleinstmögliche Gewicht (in $\mathbb{Z}$) für diese Kante an, sodass weiterhin ein kürzester Pfad im Graphen gefunden werden kann. Begründen Sie **kurz** ihre Wahl!

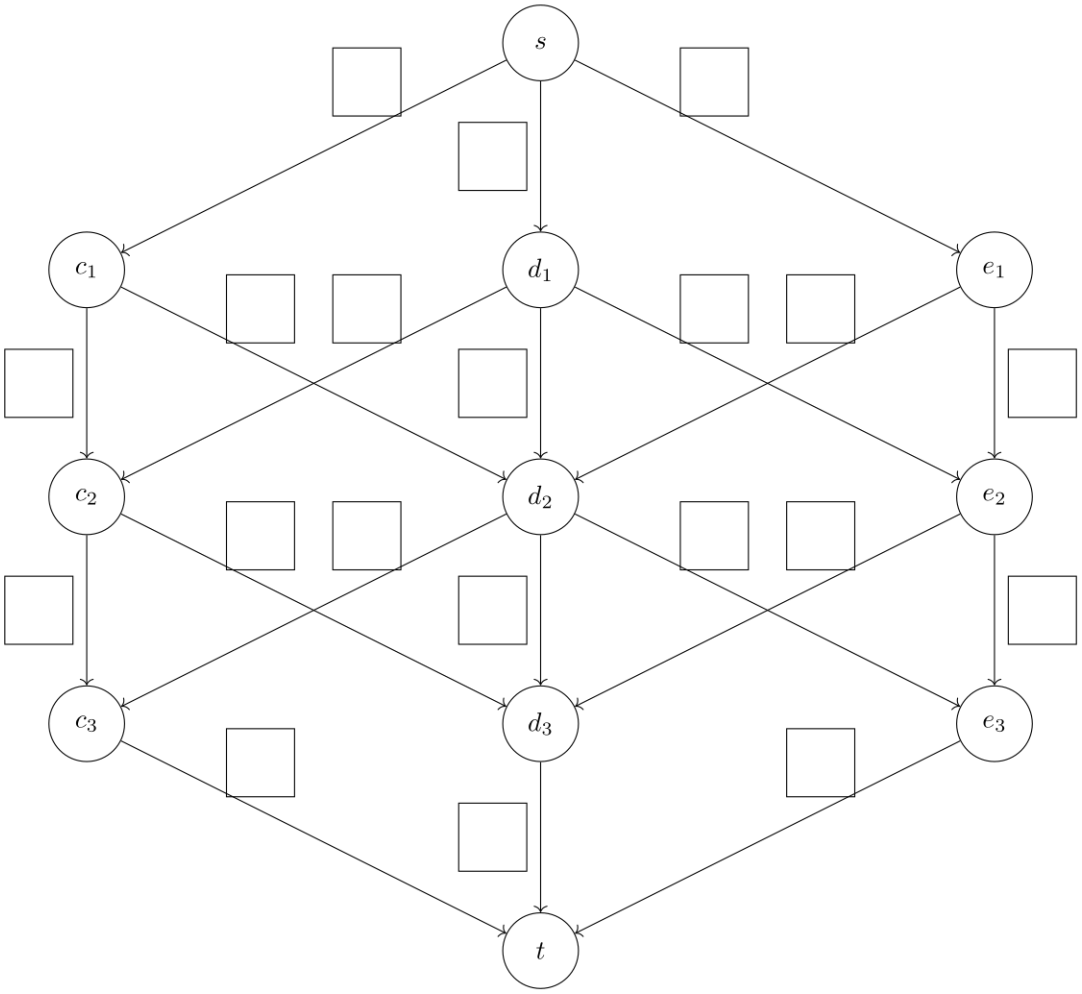
(d) (1.5 Punkte) Beschreiben Sie zuerst **kurz**, wie die *schlechteste Melodie* (das Gegenteil der *schönsten Melodie*) gefunden werden kann. Wie muss dazu ein Ihnen bekannter Algorithmus verändert werden?

	zu c	**zu** d	**zu** e
von c	3	5	kein Übergang
von d	1	3	2
von e	kein Übergang	2	3

(a) Kopie von Tabelle 2a.

Ton	c	d	e
Kosten	4	3	1

(b) Kopie von Tabelle 2b.



Schritt	1	2	3	4	5	6	7	8	9	10	11
Knoten	s										
Pfadlänge	0										
Vorgänger	-										

Aufgabe 8: Hashing (9 Punkte)

(a) (4 Punkte) Ein*e Kommiliton*in hat die sechs Schlüssel {1, 2, 7, 8, 9, 15} in eine anfangs leere Hashtabelle der Größe $M = 7$ eingefügt. Als Hashfunktion wurde die Divisions-Rest-Methode $h(k) = k \pmod 7$ verwendet. Bei Kollisionen kam ein Sondierungsverfahren zum Einsatz. Die finale Belegung der Hashtabelle sieht wie folgt aus:

Index	0	1	2	3	4	5	6
Schlüssel	7	8	2	1		15	9

1. Welche Sondierungsstrategie wurde verwendet: **Lineares Sondieren** mit der Sondierungsfunktion $s(n) = n$ oder eine Variante von **Quadratisches Sondieren** mit der Sondierungsfunktion $s(n) = n^2$? Begründen Sie Ihre Entscheidung nachvollziehbar in 1-2 Sätzen.
   Hinweis: Sie können diese Aufgabe beantworten ohne die Reihenfolge zu kennen, in der die Schlüssel eingefügt wurden.

2. Geben Sie eine **mögliche Reihenfolge** an, in der die sechs Schlüssel eingefügt worden sein könnten, um exakt die oben gezeigte Tabelle zu erzeugen.

(b) (3 Punkte) Beantworten Sie die folgenden Fragen kurz basierend auf dem Szenario in Teil (a). Begründen Sie Ihre Antworten nachvollziehbar in 1-2 Sätzen.

1. Geben Sie an, **welcher** eingefügte Schlüssel die höchste Anzahl an Sondierungen für eine erfolgreiche Suche benötigt und **wie viele** dies sind?

2. Unabhängig von Ihrer Antwort in Teil (a), erklären Sie, warum **lineares Sondieren** bei den gegebenen Schlüsseln zu primären Häufungen (primary clustering) geführt hätte.

(c) (2 Punkte) Kreuzen Sie an, ob die folgenden Aussagen wahr oder falsch sind. Eine Begründung ist nicht erforderlich.

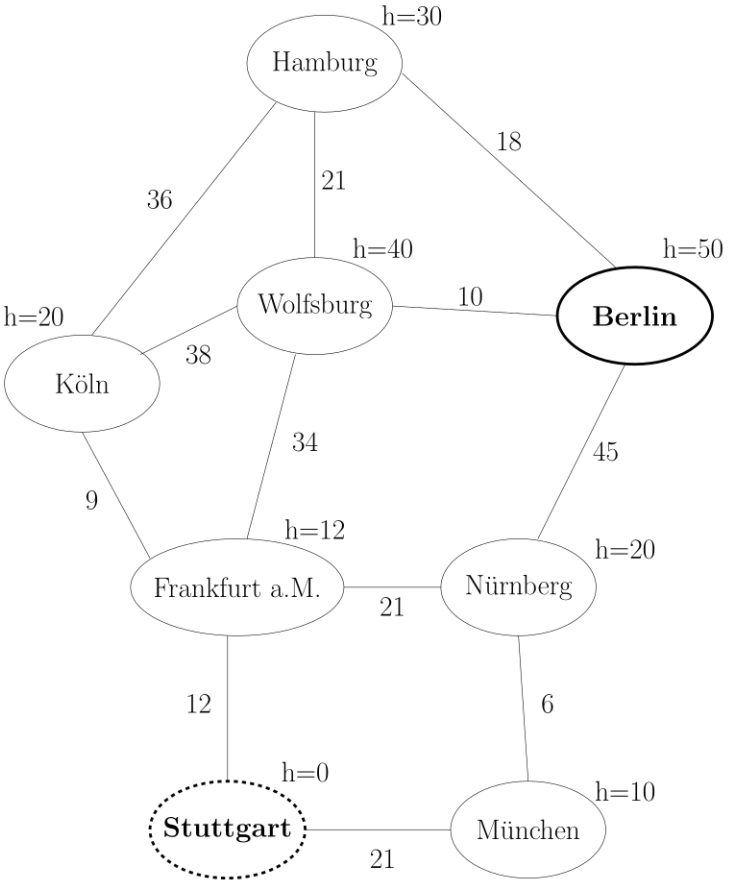
Aussage	**Wahr**	**Falsch**
Das Konzept des universellen Hashings *garantiert*, dass die Verwendung einer zufällig gewählten Hashfunktion aus einer universellen Familie stets eine minimale Anzahl von Kollisionen erzielt, was es zu einer robusten Lösung gegen gezielte Denial-of-Service (DoS)-Angriffe macht.	☐	☐
Bei einer Hashtabelle mit Verkettung (Separate Chaining) kann die Suchzeit im *schlimmsten* Fall $\mathcal{O}(n)$ betragen.	☐	☐
Eine gute Hashfunktion sollte für ähnliche Schlüssel (z.B. "Wort1", "Wort2") möglichst *unterschiedliche* Hashwerte erzeugen (Avalanche-Effekt).	☐	☐
Beim quadratischen Sondieren kann es, anders als beim linearen Sondieren, vorkommen, dass *nicht* alle Plätze der Tabelle erreicht werden, auch wenn sie noch nicht voll ist.	☐	☐

Aufgabe 9: A*-Algorithmus und Heuristiken (9 Punkte)

(a) (6 Punkte) Gegeben ist der folgende Graph, der deutsche Städte (Knoten) und Bahnverbindungen (Kanten) darstellt. Die Kantengewichte entsprechen der Reisezeit. Die im Graphen angegebenen h-Werte stellen die Schätzung der verbleibenden Reisezeit nach Stuttgart dar.

Ihre Aufgabe ist es, den A*-Algorithmus auszuführen, um den kürzesten Weg von **Berlin** nach **Stuttgart** zu finden. Die Knoten werden durch ihre Anfangsbuchstaben repräsentiert (z.B. B für Berlin, F für Frankfurt a.M.).

Protokollieren Sie die Ausführung des Algorithmus, indem Sie für jede Iteration die zwei Tabellen auf der nächsten Seite vervollständigen.



Hinweis zur Erinnerung:

- $g(n)$-Wert: Die Kosten des bisher kürzesten bekannten Pfades vom Startknoten zum Knoten n.
- $f(n)$-Wert: Eine Schätzung der Gesamtkosten vom Start zum Ziel über n.
- $h(n)$-Wert: Eine Schätzung/Heuristik der Kosten vom Knoten n zum Ziel.

Notieren Sie die Entwicklung der Priority-Queue (PQ) entlang der Iterationen des Algorithmus. Tragen Sie die Knoten in aufsteigender Reihenfolge, mit einem Komma getrennt, ein. Geben Sie außerdem den aktuellen f-Wert eines Knotens in der PQ in Klammern hinter dem Knoten an.

Iteration	PQ vor der Iteration
1	B(50)
2	
3	
4	
5	

Geben Sie in jeder Iteration den aktuell untersuchten Knoten an und tragen Sie zusätzlich für jeden Knoten jeweils den aktuellen g-Wert ein.

Iteration	Knoten	B	H	W	N	K	F	M	S
0	/	0	∞	∞	∞	∞	∞	∞	∞
1	B								
2									
3									
4									
5									

Geben Sie abschließend die kürzeste Route an:

(b) (3 Punkte) Für die folgenden drei Fragen verwenden wir die Evaluationsfunktion $f(n)$ für ein heuristisches Suchproblem, die wie folgt definiert ist:

$$f(n) = (w \cdot g(n)) + ((1 - w) \cdot h(n)).$$

Hier ist $h(n)$ eine *zulässige* Heuristik und der Gewichtungsfaktor w liegt im Bereich von $0.0 \leq w \leq 1.0$. Die Funktion $g(n)$ ist analog zum Aufgabenteil (a) definiert.

Welchen Suchalgorithmus erhalten Sie, wenn:

1.) $w = 0.0$

- ☐ Breitensuche (Breadth-First Search)
- ☐ Best-First Greedy-Algorithmus
- ☐ Dijkstras Algorithmus
- ☐ Tiefensuche (Depth-First Search)
- ☐ A* Algorithmus (A* Suche)
- ☐ Keine der oben genannten Antworten

2.) $w = 0.5$

- ☐ Breitensuche (Breadth-First Search)
- ☐ Best-First Greedy-Algorithmus
- ☐ Dijkstras Algorithmus
- ☐ Tiefensuche (Depth-First Search)
- ☐ A* Algorithmus (A* Suche)
- ☐ Keine der oben genannten Antworten

3.) $w = 1.0$

- ☐ Breitensuche (Breadth-First Search)
- ☐ Best-First Greedy-Algorithmus
- ☐ Dijkstras Algorithmus
- ☐ Tiefensuche (Depth-First Search)
- ☐ A* Algorithmus (A* Suche)
- ☐ Keine der oben genannten Antworten

Diese Seite können Sie für Notizen verwenden. Bitte nur im Ausnahmefall für Lösungen verwenden!

Diese Seite können Sie für Notizen verwenden. Bitte nur im Ausnahmefall für Lösungen verwenden!