Algorithmen und Datenstrukturen 2

WiSe 2022/2023 · Prof. Dr. M. Hoefer
5. April 2023 (Zweitklausur)
Klausurnummer: **0000**

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Nachname (Druckschrift)**:** _____

Vorname (Druckschrift)**:** _____

Matrikelnummer: _____

Studiengang: _____

Beachten Sie die folgenden Hinweise:

- Schreiben Sie Ihre persönlichen Daten **nur auf dieses Titelblatt**. **Merken oder notieren** Sie sich Ihre Klausurnummer **0000**, da nur unter dieser Nummer die Ergebnisse veröffentlicht werden.
- Legen Sie Ihre **Goethe-Card** deutlich sichtbar an Ihren Platz, damit wir während der Klausur Ihre Identität überprüfen können.
- Überprüfen Sie, ob die Klausur aus **?** durchnummerierten **Vorder- und Rückseiten** besteht. Beachten Sie, dass die Aufgabenstellungen sowohl auf den Vorder- als auch auf den Rückseiten stehen.
- Sie dürfen nur **dokumentenechte Stifte** in den Farben blau/schwarz zum Ausfüllen verwenden. Insbesondere ist die Nutzung von Tintenlöschern und Tipp-Ex untersagt.
- **Zugelassene Hilfsmittel:**
  1 Blatt DIN A4 mit handschriftlichen Notizen (beidseitig).
  Das Mitbringen nicht zugelassener Hilfsmittel stellt eine Täuschung dar und führt zum Nichtbestehen der Klausur. **Schalten Sie daher bitte alle elektronischen Geräte, insbesondere Handys und Smartwatches, vor Beginn der Klausur aus.**
- Sie müssen **alle Klausurunterlagen** (v.a. die Klausur, Zusatzpapier, DIN A4-Blatt mit handschriftlichen Notizen) nach der Bearbeitungszeit abgeben.
- Sollte der Platz unter einer Aufgabe nicht ausreichen, **nutzen Sie bitte die Zusatzblätter am Ende**. Weitere Zusatzblätter sind auf Nachfrage erhältlich. Vermerken Sie auf lose Zusatzblätter unbedingt Ihre Klausurnummer!
- Wenn sich Ihre Lösung zu einer Aufgabe teilweise oder ganz auf Zusatzblättern befindet, vermerken Sie dies entsprechend bei der Aufgabe und auf dem Zusatzblatt.
- Entscheiden Sie sich immer für **eine** Lösung. Begründungen sind nur dann notwendig, wenn die Aufgabenformulierung dies explizit verlangt.
- In allen Multiple-Choice-Fragen sind genau **zwei** von **fünf** möglichen Antworten richtig. Wenn man x richtige Kreuze setzt und y falsche, erhält man $\max\{x-y,0\}$ Punkte, also entweder 0, 1, oder 2 Punkte.
- Die Klausur ist mit Sicherheit bestanden, wenn mindestens **50%** der Höchstpunktzahl erreicht werden (ohne Berücksichtigung der Bonifikation aus den Übungen).

  Die Bearbeitungszeit beträgt **180 Minuten**.

Viel Erfolg!

Aufgabe 1: Multiple Choice — $2 + 2 + 2 = 6$ Punkt(e)

a) Betrachten Sie eine *deterministische* Turingmaschine T mit Zustandsmenge $Q = \{q_0, q_1, q_2\}$, Eingabealphabet $\Sigma = \{0, 1\}$, Startzustand q_0, Arbeitsalphabet $\Gamma = \{0, 1, \texttt{B}\}$ und Menge $F = \{q_2\}$ aller akzeptierenden Zustände. Für $q \in Q$ ist die Zustandsüberführungsfunktion δ gegeben durch

$$\begin{aligned}
\delta(q_0, 0) &= (q_1, 0, \texttt{rechts}) & \delta(q_0, 1) &= (q_0, 1, \texttt{rechts}) \\
\delta(q_1, 0) &= (q_1, 0, \texttt{rechts}) & \delta(q_1, 1) &= (q_2, 1, \texttt{rechts}) \\
\delta(q_2, 0) &= (q_1, 0, \texttt{rechts}) & \delta(q_2, 1) &= (q_0, 1, \texttt{rechts}) \\
\delta(q\ , \texttt{B}) &= \bot
\end{aligned}$$

Kreuzen Sie die beiden zutreffenden Antwortmöglichkeiten an.

- ☐ T akzeptiert alle Worte, die auf '01' enden. ✓
- ☐ T akzeptiert alle Worte, die '01' beinhalten.
- ☐ T akzeptiert alle Worte, deren vorletztes Symbol '0' ist.
- ☐ T hält nicht auf dem leeren Wort ϵ.
- ☐ T hält auf jeder Eingabe. ✓

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Sei Y ein beliebiges $\mathcal{NP}$-vollständiges Problem.

Kreuzen Sie die beiden zutreffenden Antwortmöglichkeiten an.

- ☐ Um zu zeigen, dass ein Problem X $\mathcal{NP}$-vollständig ist, genügt es zu zeigen, dass $X \in \mathcal{NP}$ und $X \leq_\text{p} Y$.
- ☐ Wenn ein Problem $X \in \mathcal{P}$ mit $Y \leq_\text{p} X$, dann gilt $\mathcal{P} = \mathcal{NP}$. ✓
- ☐ Für Y kann eine Lösung in polynomieller Zeit mittels eines nichtdeterministischen Algorithmus für `KNF-SAT` berechnet werden. ✓
- ☐ Es gibt ein $\mathcal{NP}$-vollständiges Problem X, so dass $|X|$ endlich ist.
- ☐ Für jedes $\mathcal{NP}$-harte Problem X gilt $X \leq_\text{p}$ `KNF-SAT`.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

c) Sei M eine *nichtdeterministische* Turingmaschine und w ein Eingabewort.

Kreuzen Sie die beiden zutreffenden Antwortmöglichkeiten an.

- ☐ M akzeptiert w nicht, wenn es mindestens eine Berechnung von M auf w gibt, die nicht in einem akzeptierenden Zustand endet.
- ☐ Es gibt eine deterministische Turingmaschine M', so dass w von M akzeptiert wird genau dann wenn w von M' akzeptiert wird. ✓
- ☐ Die Laufzeit von M auf w ist gegeben durch die Anzahl der Schritte in einer längsten akzeptierenden Berechnung von M auf w.
- ☐ Die Laufzeit von M auf w ist gegeben durch die erwartete Anzahl der Schritte in einer uniform zufälligen akzeptierenden Berechnung von M auf w.
- ☐ Die Laufzeit von M auf w ist gegeben durch die Anzahl der Schritte in einer kürzesten akzeptierenden Berechnung von M auf w. ✓

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

a) Sei n eine gerade Zahl und `A[1...n]` ein Array mit

$$\texttt{A}[i] = \begin{cases} i-1 & \text{falls } i \text{ gerade,} \\ i+1 & \text{sonst.} \end{cases}$$

Geben Sie jeweils asymptotisch exakt in Abhängigkeit von n an, welche Laufzeit der angegebene Sortieralgorithmus auf dem Array `A` hat.

Bubble Sort: $\Theta($ n $)$

Mergesort: $\Theta($ $n \log n$ $)$

Radix Sort mit Basis $b = 2$: $\Theta($ $n \log n$ $)$

Quicksort mit `pivot(links,rechts)=rechts`:

$\Theta($ n^2 $)$

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Gegeben sei das Array `A[1...7]` mit

`A` = 2 , 3 , 9 , 7 , 6 , 5 , 1

Nehmen Sie an, `A` wird mittels `partition(p=4, links=1, rechts=7)` partitioniert. Geben Sie das resultierende Array `A` an.

Array `A`: $2, 3, 5, 1, 6, 7, 9$

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

c) Sei n ein Vielfaches von 3 und `A[1...n]` ein Array von natürlichen Zahlen. Unter welchen Bedingungen hat Bubble-Sort immer eine Laufzeit von $\Omega(n^2)$?

Kreuzen Sie die beiden zutreffenden Antwortmöglichkeiten an.

- ☐ Das größte Element ist `A`[1].
- ☐ Kein Element befindet sich an der korrekten Stelle.
- ☐ Das kleinste Element ist unter den Elementen $\texttt{A}[\frac{2n}{3}+1], \texttt{A}[\frac{2n}{3}+2], \ldots, \texttt{A}[n]$. ✓
- ☐ Das größte Element ist unter den Elementen $\texttt{A}[1], \texttt{A}[2], \ldots, \texttt{A}[n/3]$.
- ☐ Es gilt $\texttt{A}[i] > \texttt{A}[i+1] > \cdots > \texttt{A}[i+n/3]$ für ein $1 \leq i \leq \frac{2n}{3}$. ✓

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

d) Betrachten Sie folgenden Sortieralgorithmus für ein Array `A[1...n]`.

```
i := 1;
while i ≤ n do
    if i = 1 oder A[i] ≥ A[i − 1] then
        i := i + 1;
    else
        h := A[i];
        A[i] := A[i − 1];
        A[i − 1] := h;
        i := i − 1;
```

i) **(2 Punkte)** Geben Sie die best- sowie worst-case Laufzeit des Algorithmus asymptotisch exakt in Abhängigkeit von n an.

Best-case: $\Theta($ n $)$

Worst-case: $\Theta($ n^2 $)$

iii) **(2 Punkte)** Geben Sie eine Instanz `A[1...n]` an, auf der der Algorithmus asymptotisch seine worst-case Laufzeit erreicht.

Lösungsskizze: `A[i] = n-i` für `i=1...n`

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

Aufgabe 3: GroSSe Menge mit kleiner Differenz — 7 Punkt(e)

Gegeben sei ein Array `A[1...n]` mit paarweise verschiedenen ganzen Zahlen und eine natürliche Zahl $r > 0$. Gesucht ist eine möglichst groSSe Menge M von Zahlen aus `A`, so dass die Differenz zwischen gröSSter und kleinster Zahl in M höchstens r ist.

Beschreiben Sie einen Algorithmus, welcher eine solche Menge M berechnet. Die asymptotische worst-case Laufzeit Ihres Algorithmus soll $\mathcal{O}(n \log n)$ nicht überschreiten. Sie dürfen alle aus der Vorlesung bekannten Verfahren und deren Eigenschaften ohne weitere Erläuterung verwenden.

Begründen Sie auch, warum Ihr Algorithmus korrekt arbeitet und die Laufzeitschranke einhält.

Lösungsskizze: zunächst wird `A` mit Mergesort (oder Heapsort oder Quicksort mit deterministischer Pivotwahl) in Laufzeit $\mathcal{O}(n \log n)$ sortiert. Die gesuchte Menge M ist nun eine zusammenhängende Teilfolge des Arrays. Um diese zu finden, werden zwei Indizes `links=1` und `rechts=1` initialisiert. AnschlieSSend wird `rechts` so lange inkrementiert, bis `A[rechts+1]` $-$ `A[links]` $> r$ (oder `rechts` $= n$). Die Teilfolge `A[links]`, ..., `A[rechts]` ist nun maximal lang, unter der Annahme, dass `links=1` die kleinste Zahl in M ist. Nun wird `links` inkrementiert und das Vorgehen wiederholt, bis `links` $> n$. Die Indizes $\texttt{links}_{\text{opt}}$, $\texttt{rechts}_{\text{opt}}$, die (global) einer maximal langen Teilfolge entsprechen, werden gespeichert und aktuell gehalten. Am Ende wird $\texttt{A[links}_{\text{opt}}\texttt{]}, \ldots, \texttt{A[rechts}_{\text{opt}}\texttt{]}$ ausgegeben.

Korrektheit: Folgt aus obiger Beschreibung.

Laufzeit: Die Suche nach der Teilfolge hat Laufzeit $\mathcal{O}(n)$, da die beiden Indizes nur vergröSSert werden. Das Sortieren dominiert also die Laufzeit mit $\mathcal{O}(n \log n)$.

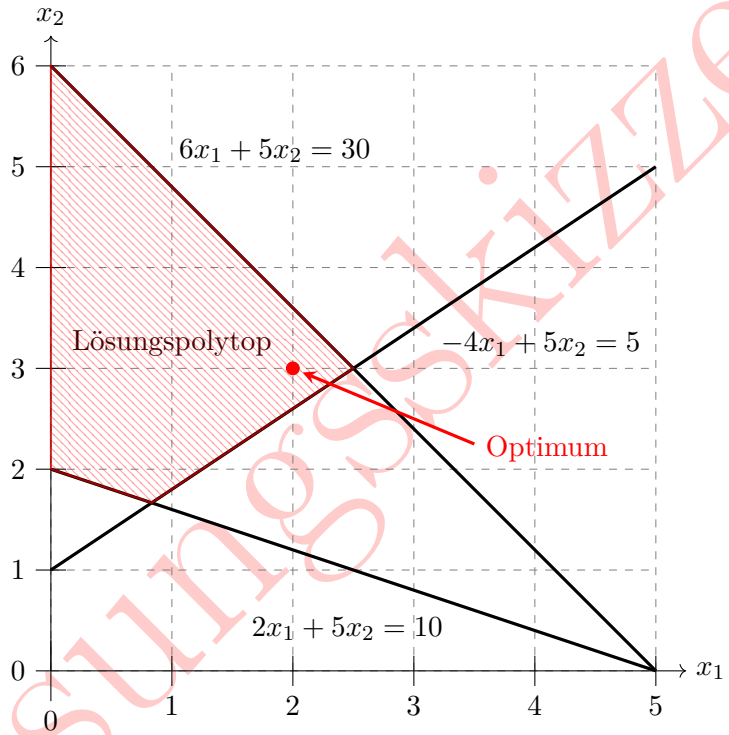
⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

a) Betrachten Sie das folgende Lineare Programm:

$$\begin{array}{lrcl} \text{Max.} & 7x_1 + 3x_2 & & \\ \text{s.d.} & 6x_1 + 5x_2 & \leq & 30 \\ & 2x_1 + 5x_2 & \geq & 10 \\ & -4x_1 + 5x_2 & \geq & 5 \\ & x_1, x_2 & \geq & 0 \end{array}$$

i) **(2 Punkte)** Schraffieren Sie in der nachfolgenden Abbildung das Lösungspolytop.

ii) **(1 Punkt)** Markieren Sie in der Abbildung eine optimale *ganzzahlige* Lösung.



⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Gegeben sei das folgende Lineare Programm:

$$\begin{array}{lrcl} \text{Max.} & 4x_1 + x_2 - 2x_3 & & \\ \text{s.d.} & 3x_1 + 2x_2 + x_3 & \leq & 12 \\ & -x_1 \quad - 4x_3 & \leq & -3 \\ & x_1, x_2, x_3 & \geq & 0 \end{array}$$

Geben Sie das entsprechende duale Lineare Programm an. Gehen Sie dabei nach dem Rezept aus der Vorlesung vor.

Lösungsskizze: Bestimme die Vektoren $\boldsymbol{c}$ und $\boldsymbol{b}$ sowie die Matrix $\boldsymbol{A}$:

$$\boldsymbol{c}^T = (4, 1, -2), \qquad \boldsymbol{b}^T = (12, -3), \qquad \boldsymbol{A} = \begin{pmatrix} 3 & 2 & 1 \\ -1 & 0 & -4 \end{pmatrix}.$$

Da $\boldsymbol{b}$ zwei Einträge hat, folgt $\boldsymbol{y}^T = (y_1, y_2)$ mit $y_1, y_2 \geq 0$. Das duale LP lautet wie folgt:

$$\begin{array}{lrcl} \text{Min.} & 12y_1 - 3y_2 & & \\ \text{s.d.} & 3y_1 - y_2 & \geq & 4 \\ & 2y_1 & \geq & 1 \\ & y_1 - 4y_2 & \geq & -2 \\ & y_1, y_2 & \geq & 0 \end{array}$$

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

c) Betrachten Sie ein allgemeines Lineares Programm (P)

$$\begin{array}{lrcl} \text{Max.} & \boldsymbol{c}^T\boldsymbol{x} & & \\ \text{s.d.} & \boldsymbol{A}\boldsymbol{x} & \leq & \boldsymbol{b} \\ & \boldsymbol{x} & \geq & \boldsymbol{0}\,, \end{array} \qquad \text{(P)}$$

wobei alle Einträge in den Vektoren $\boldsymbol{c}$ und $\boldsymbol{b}$ nicht negativ sind.

Es sei $\tilde{\boldsymbol{x}} = (\tilde{x}_1, ..., \tilde{x}_n)^T$ eine zulässige Lösung für (P) und $\tilde{\boldsymbol{y}} = (\tilde{y}_1, ..., \tilde{y}_m)$ eine zulässige Lösung für das zu (P) duale Lineare Programm.

Dabei gelte für alle $i = 1, ..., n$ entweder $\tilde{x}_i = 0$ oder $(\tilde{y}^T A)_i = c_i$. Analog gelte für alle $j = 1, ..., m$ entweder $\tilde{y}_j = 0$ oder $(A\tilde{x})_j \geq \frac{1}{\alpha} b_j$ für ein $\alpha > 1$.

Zeigen Sie, dass $\tilde{\boldsymbol{x}}$ eine α-approximative Lösung für (P) ist.

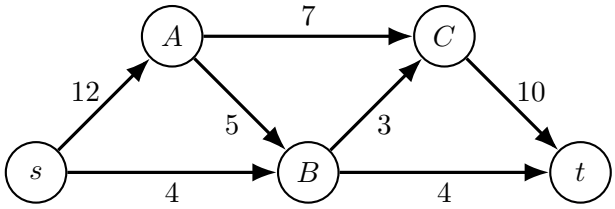
Lösungsskizze: Es gilt

$$\boldsymbol{c}^T\tilde{\boldsymbol{x}} \;=\; (\tilde{\boldsymbol{y}}^T\boldsymbol{A})\tilde{\boldsymbol{x}} \;=\; \tilde{\boldsymbol{y}}^T(\boldsymbol{A}\tilde{\boldsymbol{x}}) \;\geq\; \frac{1}{\alpha}\,\tilde{\boldsymbol{y}}^T\boldsymbol{b} \;\geq\; \frac{1}{\alpha}\,\boldsymbol{c}^T\boldsymbol{x}^*.$$

Die erste Gleichung folgt aufgrund von (entweder) $\tilde{x}_i = 0$ oder $(\tilde{y}^T A)_i = c_i$ für alle $i = 1, ..., n$. Die vorletzte Ungleichung ergibt sich analog aus der Bedingung (entweder) $\tilde{y}_j = 0$ oder $(A\tilde{x})_j \geq \frac{1}{\alpha} b_j$ für alle $j = 1, ..., m$. Gleichzeitig gilt nach dem Satz der Schwachen Dualität für eine optimale Lösung $\boldsymbol{x}^*$ für (P), dass $\boldsymbol{c}^T\boldsymbol{x}^* \leq \tilde{\boldsymbol{y}}^T\boldsymbol{b}$, woraus die letzte Ungleichung resultiert.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

a) Betrachten Sie das folgende Flussnetzwerk $G = (V, E, c, s, t)$ mit $V = \{A, B, C, s, t\}$:



Die Kantenbeschriftungen geben für jede Kante $e \in E$ ihre Kapazität $c(e)$ an.

Berechnen Sie mittels des Edmonds-Karp Algorithmus einen maximalen Fluss in G.

Tragen Sie für jede Runde i den gewählten augmentierenden Pfad zusammen mit dem Wert f_i des gesamten s-t-Flusses am Ende der Runde in die nachfolgende Tabelle ein (eine Runde entspricht einer Zeile).

Runde i	Augmentierender Pfad p_i	Flusswert f_i
1	$p_1 = (s, B, t)$	4
2	$p_2 = (s, A, C, t)$	11
3	$p_2 = (s, A, B, C, t)$	14

⇑⇑⇑ _____ / 9 Punkt(e) ⇑⇑⇑

b) Sei $G = (V, E, c, s, t)$ ein Flussnetzwerk mit nicht-negativen, ganzzahligen Kapazitäten.

Geben Sie für die folgenden Aussagen jeweils an, ob diese wahr oder falsch sind. Beweisen Sie Ihre Antwort.

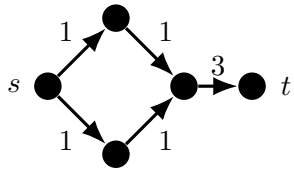
i) **(2 Punkte)** Falls alle Kapazitäten *gerade* sind, gibt es einen maximalen Fluss f^*, so dass $f^*(e)$ für alle Kanten $e \in E$ *gerade* ist.

Lösungsskizze: Wahr. Wenn man alle Kapazitäten durch 2 teilt, erhält man ein Flussnetzwerk mit ganzzahligen Kapazitäten. Daher ist auch der maximale Fluss ganzzahlig. Wenn dieser Fluss verdoppelt wird, erhält man einen geraden maximalen Fluss im originalen Flussnetzwerk.

Alternative Lösung: Betrachte MaxFlow-Algorithmus (z.B. Edmonds-Karp). Induktives Argument: In jedem Schritt wird ein augmentierender Pfad gewählt, dessen minimale Restkapazität gerade ist. Dieses Gewicht wird von allen Restkapazitäten (die ebenfalls gerade sind) subtrahiert. Folglich bleiben alle Restkapazitäten weiterhin gerade.

ii) **(2 Punkte)** Falls alle Kapazitäten *ungerade* sind, gibt es einen maximalen Fluss f^*, so dass $f^*(e)$ für alle Kanten $e \in E$ *ungerade* ist.

Lösungsskizze: Falsch. Gegenbeispiel mit ungeraden Kapazitäten, aber (geradem) maximalem Fluss $|f^*| = 2$:



iii) **(2 Punkte)** Sei $\lambda > 0$ eine beliebige Konstante und c' mit $c'(e) = \lambda \cdot c(e)$ für alle $e \in E$ eine neue Kapazitätsfunktion. Jeder minimale s-t-Schnitt für Kapazitäten c ist auch minimal für Kapazitäten c'.

Lösungsskizze: Wahr. Die Relation zwischen den Werten zweier Schnitte bleibt erhalten. Sei (S, T) ein Schnitt. Dann gilt $c'(S,T) = \sum_{(u,v)\in E} c'(u,v) = \sum_{(u,v)\in E} \lambda \cdot c(u,v) = \lambda \cdot \sum_{(u,v)\in E} c(u,v) = \lambda \cdot c(S,T)$. Gilt für zwei Schnitte (S,T) und (S',T') die Relation $c(S,T) \leq c(S',T')$, dann folgt auch $\lambda \cdot c(S,T) \leq \lambda \cdot c(S',T')$ und somit $c'(S,T) \leq c'(S',T')$.

Alternativ: Verweis auf Proportionalität ausreichend, sofern sinnvoll begründet.

iv) **(2 Punkte)** Sei $\lambda > 0$ eine beliebige Konstante und c' mit $c'(e) = c(e) + \lambda$ für alle $e \in E$ eine neue Kapazitätsfunktion. Jeder minimale s-t-Schnitt für Kapazitäten c ist auch minimal für Kapazitäten c'.

Lösungsskizze: Falsch. Ein Gegenbeispiel ist das Flussnetzwerk in ii). Ein minimaler s-t-Schnitt (S,T) mit Wert 2 ist gegeben durch $S = \{s\}$ und $T = V \setminus \{s\}$. Wählt man $\lambda = 2$, ist der (eindeutige) minimale s-t-Schnitt (S',T') mit Wert 5 gegeben druch $S = \{t\}$ und $T = V \setminus \{t\}$, wohingegen der Wert von (S,T) nun 6 beträgt.

⇑⇑⇑ _____ / 8 Punkt(e) ⇑⇑⇑

Aufgabe 6: P und NP

$2 + 2 + 2 + 4 = 10$ Punkt(e)

Seien $\mathcal{NPC}$ und $\mathcal{NPH}$ die Klassen aller $\mathcal{NP}$-vollständigen bzw. aller $\mathcal{NP}$-harten Probleme.

a) Es seien L_1 und L_2 zwei Entscheidungsprobleme.

Kreuzen Sie die beiden zutreffenden Antwortmöglichkeiten an.

- ☐ Falls $\mathcal{P} \neq \mathcal{NP}$ und $L_1, L_2 \in \mathcal{P}$, dann gilt $L_1 \cap L_2 \in \mathcal{P}$. ✓
- ☐ Falls $\mathcal{P} \neq \mathcal{NP}$ und $L_1, L_2 \in \mathcal{NP}$, dann gilt *nicht* $L_1 \cap L_2 \in \mathcal{NP}$.
- ☐ Falls $\mathcal{P} = \mathcal{NP}$ und $L_1 \in \mathcal{NPH}$, dann gilt $L_1 \in \mathcal{P}$.
- ☐ Falls $\mathcal{P} \neq \mathcal{NP}$ und $L_1 \in \mathcal{NPH}$, dann gilt $L_1 \in \mathcal{NPC}$.
- ☐ Falls $\mathcal{P} = \mathcal{NP}$ kann gleichzeitig $L_1 \in \mathcal{NPH}$ und $L_1 \in \mathcal{P}$ gelten. ✓

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Betrachten Sie die Sprache $L = \{a^n b^n \mid n \geq 0\}$. Es wird der folgende Versuch unternommen, die Aussage $\mathcal{P} = \mathcal{NP}$ zu zeigen:

Die Reduktion `Clique` $\leq_{\mathrm{p}}$ L *ist gegeben durch eine transformierende Turingmaschine* M *mit*

$$M\left((G,k)\right) = \begin{cases} aabb & \textit{falls } G \textit{ eine Clique der GröSSe } k \textit{ besitzt,} \\ aab & \textit{sonst.} \end{cases}$$

Weil `Clique` $\mathcal{NP}$*-vollständig ist und* `Clique` $\leq_p L$ *sowie* $L \in \mathcal{P}$ *gilt, folgt* $\mathcal{P} = \mathcal{NP}$. ☐

Begründen Sie, warum die Aussage dadurch nicht bewiesen ist.

Lösungsskizze: Es ist $L \in \mathcal{P}$. Es wird nicht auf die konkrete Konstruktion der TM eingegangen. Es ist also unklar, ob es überhaupt eine TM gibt, die das `Clique`-Problem in polynomieller Zeit entscheiden kann. Dabei handelt es sich um ein offenes Problem.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

c) Erläutern Sie, warum $\mathcal{P} \subseteq \mathcal{NP}$ gilt.

Lösungsskizze:
Falls $L \in \mathcal{P}$, gibt es eine deterministische Turingmaschine, die L in polynomieller Zeit entscheidet. Nach Definition folgt daraus $L \in \mathcal{NP}$, da eine deterministische Turingmaschine ein Spezialfall einer nichtdeterministischen Turingmaschine ist.

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

d) Zeigen Sie, dass $\mathcal{P} \cap \mathcal{NPC} = \emptyset$, falls $\mathcal{P} \neq \mathcal{NP}$.

Hinweis: Sie dürfen die Aussage in c) hier auch ohne Beweis verwenden.

Lösungsskizze:
Dem Hinweis folgend nehmen wir an, dass es eine Sprache L gibt, die NP-vollständig ist (d.h., L ist NP-hart *und* in NP) und zeitgleich von einer deterministischen TM in polynomieller Zeit gelöst werden kann. Ausgehend von der hypothetischen Existenz von L zeigen wir dann, dass $NP \subseteq P$. Zusammen mit c) folgt somit $NP = P$, d.h., ein Widerspruch zur Annahme $NP \neq P$.

Sei also $L' \in NP$. Wir wollen zeigen, dass $L' \in P$, um einen Widerspruch zu erzeugen. Da L nach Definition (auch) NP-hart ist, können wir L' über eine polynomielle Reduktion mittels L lösen, d.h., für jedes $w \in L'$ gilt die Äquivalenz $w \in L' \Leftrightarrow M(w) \in L$. Eine TM für L' würde also für eine Instanz w zuerst $M(w)$ in polynomieller Zeit berechnen und dann die TM für L als Subroutine verwenden, um die Antwort auf $M(w) \in L$ in polynomieller Zeit zu berechnen. Die Entscheidung $w \in L'$ benötigt also nur polynomielle Zeit, was bedeutet, dass $L' \in P$.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

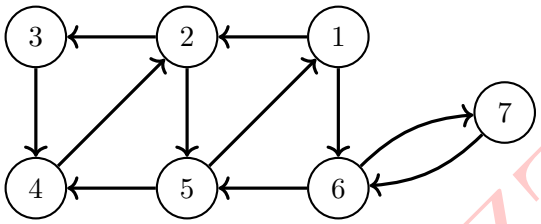
In einem *gerichteten* Graph $G = (V, E)$ heiSSt eine Knotenmenge $W \subseteq V$ *kreiseliminierend*, wenn das Entfernen der Knoten in W mit allen inzidenten Kanten den Graph kreisfrei macht.

Im Problem `Kreiseliminierung` ist ein *gerichteter* Graph $G = (V, E)$ und ein Parameter k gegeben. Es soll entschieden werden, ob es eine kreiseliminierende Menge $W \subseteq V$ der GröSSe $|W| = k$ gibt.

Formal ist die Sprache `Kreiseliminierung` wie folgt definiert:

$$\texttt{Kreiseliminierung} := \{(G, k) \mid G \text{ ist gerichteter Graph und es gibt kreiseliminierende Knotenmenge der GröSSe } k\}$$

a) Betrachten Sie folgenden Graphen:



Geben Sie eine möglichst kleine kreiseliminierende Knotenmenge W an.

$W = \{$ 2, 6 $\}$

⇑⇑⇑ _____ / 1 Punkt(e) ⇑⇑⇑

b) Beschreiben Sie ein konkretes Verfahren, um zu zeigen, dass `Kreiseliminierung` $\in \mathcal{NP}$.

Lösungsskizze: Für eine gegebene Knotenmenge $W \subseteq V$ kann wie folgt in polynomieller Zeit überprüft werden, ob der Graph nach Entfernen von W kreisfrei ist: Markiere alle Knoten mit 0. Wende Tiefensuche auf G an und markiere begonnene Knoten mit 1 und abgeschlossene Knoten mit 2. Immer wenn ein Knoten aus W gewählt werden soll, ignoriere ihn. Falls ein begonnener und nicht abgeschlossener Knoten entdeckt wird, gibt es einen Kreis – gebe NEIN aus. Ansonsten gebe JA aus.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

c) Zeigen Sie, dass `Kreiseliminierung` $\mathcal{NP}$-hart ist.

Lösungsskizze: Wir zeigen die Reduktion `VertexCover` $\leq_p$ `Kreiseliminierung`. Da `VertexCover` $\mathcal{NP}$-hart ist, folgt $\mathcal{NP}$-härte von `Kreiseliminierung` .

Transformation:
Sei (G, k) Eingabe für `VertexCover`. Ersetze jede ungerichtete Kante $\{u, v\} \in E$ durch die beiden gerichteten Kanten (u, v) und (v, u). Sei $G' = (V, E')$ der resultierende Graph. Dann ist (G', k) die transformierte Eingabe.

$(G, k) \in$ `VC` $\Rightarrow (G', k) \in$ `Kreiseliminierung`:
Sei $C \subseteq V$ eine Knotenüberdeckung der GröSSe $|C| = k$ in G. Nach Entfernen von C aus G' ist G' kreisfrei, denn alle (gerichteten) Kanten sind entfernt. Somit ist C eine kreiseliminierende Menge in G' und es folgt $(G', k) \in$ `Kreiseliminierung`.

$(G', k) \in$ `Kreiseliminierung` $\Rightarrow (G, k) \in$ `VC`:
Sei $W \subseteq V$ eine kreiseliminierende Knotenmenge der GröSSe $|W| = k$ in G'. Für jede Kante $\{u, v\} \in E$ muss mindestens einer der Knoten u, v in W sein, denn sonst gäbe es einen Kreis (u, v, u) in G'. Somit ist W eine Kontenüberdeckung in G und es folgt $(G, k) \in$ `VC`.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑

a) Die Sprache L_1 sei wie folgt definiert:

$$L_1 := \{\langle M \rangle \mid M \text{ ist eine Turingmaschine und } |\mathrm{L(M)}|{=}2\}$$

Zeigen Sie, dass die Sprache L_1 unentscheidbar ist.

Lösungsskizze: Wir zeigen die Reduktion $H_\epsilon \leq L_1$.

Transformation:
Sei o.B.d.A. $\langle M \rangle$ für eine TM M Eingabe für H_ϵ. Schreibe die transformierte Eingabe $\langle M' \rangle$, wobei in M' alle Zustände akzeptierend sind. M' überprüft zunächst, ob das Wort 1, ϵ oder ein anderes Wort auf dem Band steht. Falls 1 auf dem Band steht, hält M'. Falls ϵ auf dem Band steht, arbeitet M' wie M. Ansonsten geht M' in eine Endlosschleife.

Es gilt stets $\{1\} \subseteq L(M') \subseteq \{1, \epsilon\}$, denn M' akzeptiert in jedem Fall das Wort 1 und für alle Worte $w \notin \{1, \epsilon\}$ geht M' in eine Endlosschleife. Dies wird im Folgenden genutzt.

$\langle M \rangle \in H_\epsilon \Rightarrow \langle M' \rangle \in L_1$
Wenn M auf ϵ hält, dann hält M' auch auf ϵ, denn M' arbeitet auf dem leeren Wort genau wie M. Da alle Zustände in M' akzeptierend sind, wird ϵ durch M' auch akzeptiert. Es folgt $L(M) = \{1, \epsilon\}$ und somit $|L(M)| = 2$.

$\langle M \rangle \in H_\epsilon \Leftarrow \langle M' \rangle \in L_1$
Wenn $|L(M')| = 2$, dann muss $L(M') = \{1, \epsilon\}$ gelten, und da M' auf ϵ wie M arbeitet, muss M auf ϵ halten.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

b) Sei $|w|$ die Anzahl der Zeichen eines Wortes w. Eine Turingmaschine M heiSSt *Kürzer*, wenn sie für jedes Eingabewort w, auf dem sie hält, als Ausgabe eine Bitfolge w' mit $|w'| < |w|$ schreibt.

Zeigen Sie mit dem Satz von Rice, dass die folgende Sprache L_2 unentscheidbar ist:

$$L_2 := \{\langle M \rangle \mid M \text{ ist ein Kürzer}\}$$

Begründen Sie insbesondere die nicht-Trivialität der Menge S.

Lösungsskizze: Wähle $S = \{f \mid f(w) = \perp \text{ oder } |f(w)| < |w| \text{ f.a. } w\}$. Es gilt $L = B(S)$ und S ist nicht-trivial, da es berechenbare Funktionen in S (z.B. $f(w) = \perp$ f.a. w) und auSSerhalb von S (z.B. $f(w) = 111$ f.a. w) gibt.

⇑⇑⇑ _____ / 4 Punkt(e) ⇑⇑⇑

c) Die Sprache L_3 sei wie folgt definiert:

$$L_3 := \{\langle M_1\rangle\#\langle M_2\rangle \mid M_1 \text{ und } M_2 \text{ sind Turingmaschinen und } L(M_1) \cap L(M_2) \neq \emptyset\}$$

Ist die Sprache L_3 rekursiv aufzählbar? Beweisen Sie Ihre Antwort.

Lösungsskizze: Ja. Folgender Algorithmus hält und akzeptiert auf allen JA-Instanzen der Sprache L_3 (und hält nicht auf allen NEIN-Instanzen):

```
for(k=1...){
```
–Simuliere alle Worte w mit $|w| \leq k$ für k Schritte auf M_1 und M_2.
–Wenn es unter diesen ein Wort gibt, welches von M_1 und von M_2 nach höchstens k Schritten akzeptiert wird, akzeptiere.
```
}
```

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

Aufgabe 9: Approximationsalgorithmen — $3 + 5 = 8$ Punkt(e)

Gegeben sei ein ungerichteter Graph $G = (V, E)$. Ein *2-Labeling* von G ist eine Funktion $L : V \mapsto \{Blau, Rot\}$, die jedem Knoten eines der Label *Blau* oder *Rot* zuordnet. Eine Kante $\{u, v\} \in E$ heiSSt *gemischt* unter L, falls $L(u) \neq L(v)$, die Label der beiden Endknoten also verschieden sind. Im Problem `MaxMixedEdges` soll ein 2-Labeling gefunden werden, welches die Anzahl der gemischten Kanten maximiert.

Betrachten Sie folgenden Algorithmus für das Problem:

Setze das Label aller Knoten auf *Blau*;
repeat
  $changed := false$;
  if *es gibt einen Knoten $v \in V$, so dass die Änderung des Labels von v die Anzahl gemischter Kanten strikt erhöht* **then**
    Ändere das Label von v;
    $changed := true$;
until $changed = false$;

a) Begründen Sie, warum der Algorithmus polynomielle Laufzeit hat.

Lösungsskizze: In jeder Iteration der repeat-until-Schleife wird die Anzahl der gemischten Kanten um mindestens eine erhöht. Da maximal $|E|$ viele Kanten gemischt sein können, kann es maximal $|E|$ Iterationen geben. Die Laufzeit des Rumpfs ist $\mathcal{O}(|V|^2)$ und somit ist die gesamte Laufzeit $\mathcal{O}(|E| \cdot |V|^2)$, also polynomiell.

⇑⇑⇑ _____ / 3 Punkt(e) ⇑⇑⇑

b) Zeigen Sie, dass der Algorithmus stets eine 2-approximative Lösung ausgibt.

Lösungsskizze: Wenn der Algorithmus endet, ist für jeden Knoten mindestens die Hälfte seiner inzidenten Kanten gemischt, denn sonst würde das Ändern des Labels des Knotens die Anzahl gemischter Kanten erhöhen. Insgesamt sind also $Alg \geq \frac{1}{2} \sum_{v \in V} d_v/2 = |E|/2$ Kanten gemischt. Offensichtlich ist $Opt \leq |E|$ und es folgt $Opt/Alg \leq 2$.

⇑⇑⇑ _____ / 5 Punkt(e) ⇑⇑⇑

Aufgabe 10: Umsortieren — $2 + 7 = 9$ Punkt(e)

Sei $\boldsymbol{P}$ eine $n \times n$-Matrix, mit Einträgen P_{ij}, wobei i die Zeile und j die Spalte bezeichnet. Jeder Eintrag P_{ij} ist entweder 0 oder 1. Die Einträge in den Positionen P_{ii} heiSSen *Diagonaleinträge*.

Je zwei Zeilen i und i' in $\boldsymbol{P}$ können *vertauscht* werden, d.h., für alle $k = 1, ..., n$ werden die Einträge P_{ik} und $P_{i'k}$ vertauscht. Die Matrix $\boldsymbol{P}$ ist *kommutativ*, wenn die Zeilen in $\boldsymbol{P}$ so vertauscht werden können (in einer beliebigen Reihenfolge), dass $P_{ii} = 1$ für alle Diagonaleinträge in $\boldsymbol{P}$ gilt.

a) Welche der folgenden 4×4-Matrizen sind *nicht* kommutativ?

Kreuzen Sie die beiden zutreffenden Antwortmöglichkeiten an.

☐ $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$ ☐ $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$ ☐ $\begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$

☐ $\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$ ✓ ☐ $\begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ ✓

⇑⇑⇑ _____ / 2 Punkt(e) ⇑⇑⇑

b) Beschreiben Sie, wie mittels eines Flussnetzwerks in polynomieller Zeit bestimmt werden kann, ob eine gegebene Matrix $\boldsymbol{P}$ kommutativ ist oder nicht. Beweisen Sie auSSerdem die Korrektheit Ihres Verfahrens.

Lösungsskizze: Das Problem kann analog zu Bipartitem Maximum Matching aus der Vorlesung gelöst werden.

Dafür konstruieren wir zuerst einen passenden bipartiten Graphen $G = (A \cup B, E)$ mit $|A| = |B| = n$ und $E = \{\{a, b\} \mid P_{ab} = 1, a \in A, b \in B\}$. Die Reduktion auf die Berechnung maximaler Flüsse läuft dann wie in der Vorlesung durch die Konstruktion eines entsprechenden Flussnetzwerks G' (Füge alle Knoten $A \cup B$ hinzu, füge neue Quelle s und Senkte t hinzu, füge eine gerichtete Kante (s, a) für jeden $a \in A$ hinzu, füge eine gerichtete Kante (b, t) für jeden $b \in B$ hinzu, füge eine gerichtete Kante (a, b) für jeden $\{a, b\} \in E$ hinzu, alle Kanten erhalten Kapazität 1).

Die Konstruktion erfolgt in Zeit $\mathcal{O}(n^2)$. Mittels einem der aus der Vorlesung bekannten Verfahren (Algorithmus von Ford-Fulkerson oder Edmonds-Karp) kann nun in polynomieller Laufzeit ein (ganzzahliger) maximaler Fluss f^* in G' berechnet werden.

Sei f^* der maximale Fluss. Es verbleibt zu zeigen

$$\boldsymbol{P} \text{ ist kommutativ} \quad \Leftrightarrow \quad |f^*| = n\,.$$

Wichtige Einsichten: 1) Durch Vertauschung $i \leftrightarrow j$ steht im Diagonaleintrag P_{jj} eine 1. Kante (i, j) in G' drückt also aus, dass Zeile i die in Zeile j beim Diagonaleintrag benötigte 1 „liefern" kann. 2) Jede Zeile wird effektiv höchstens einmal vertauscht, da statt $i \leftrightarrow k$ und $k \leftrightarrow j$ kann stets direkt $i \leftrightarrow j$ ausgeführt werden kann.

$\Rightarrow$: Es gibt eine Permutation von Zeilen, so dass $P_{ii} = 1$ für alle i. Nach 1) liefert jede Zeile einen Diagonaleintrag, weil ansonsten nicht alle n Diagonaleinträge besetzt wären. Demnach gibt es eine Vertauschung mit $P_{ii} = 1$ für alle $i = 1, .., n$, in der keine zwei Zeilen auf dieselbe Zeile vertauscht werden. Es existiert also ein perfektes Matching in G. Folglich gibt es in G' einen maximalen Fluss mit Wert n (da $c(e) = 1$ für alle Kanten in G').
$\Leftarrow$: Angenommen, $\boldsymbol{P}$ ist nicht kommutativ. Dann gibt es keine Vertauschung ohne dass in mindestens einem Diagonaleintrag eine 0 steht. Nach dem Schubfachprinzip können also mindestens zwei Zeilen i nur eine 1 für dieselbe Zeile j liefern. Somit existiert kein perfektes Matching in G, so dass $|f^*| \leq n$ für den maximalen Fluss in G'.

⇑⇑⇑ _____ / 7 Punkt(e) ⇑⇑⇑