Prüfungsteilnehmer	Prüfungstermin	Einzelprüfungsnummer

Kennzahl: _____

Kennwort: _____

Arbeitsplatz-Nr.: _____

Frühjahr
2020

66115

Erste Staatsprüfung für ein Lehramt an öffentlichen Schulen

— Prüfungsaufgaben —

Fach: **Informatik (vertieft studiert)**

Einzelprüfung: **Theoretische Informatik, Algorithmen**

Anzahl der gestellten Themen (Aufgaben): 2

Anzahl der Druckseiten dieser Vorlage: 16

Bitte wenden!

Thema Nr. 1
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1 (Aussagen) [24 PUNKTE]

Zeigen oder widerlegen Sie die folgenden Aussagen (die jeweiligen Beweise sind sehr kurz).

Schreiben Sie zunächst zur Aussage „Richtig" oder „Falsch" und dann Ihre Begründung.

(a) [6 PUNKTE] Seien $L, L' \subseteq \Sigma^*$ formale Sprachen. Falls L kontextfrei und L' unentscheidbar ist, dann ist die Vereinigung $L \cup L'$ sicher *nicht* regulär.

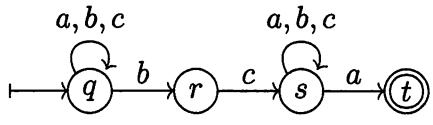
(b) [6 PUNKTE] Die Menge aller Sprachen in der Klasse $\mathcal{NP}$ ist abzählbar unendlich.

(c) [6 PUNKTE] Es ist unentscheidbar, ob eine vorgelegte aussagenlogische Formel durch *jede* Belegung ihrer Variablen erfüllt wird.

(d) [6 PUNKTE] Angenommen es gilt $\mathcal{P} \neq \mathcal{NP}$, und $L \in \mathcal{NP} \setminus \mathcal{P}$ ist gegeben. Dann ist es möglich, dass L nur endlich viele Nerode-Äquivalenzklassen besitzt.

Aufgabe 2 (Reguläre Sprachen) [30 PUNKTE]

(a) [4 PUNKTE] Es sei $L \subseteq \{a, b, c\}^*$ die von dem folgenden nichtdeterministischen Automaten akzeptierte Sprache:

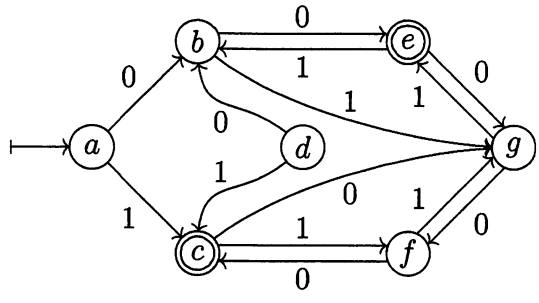


Beschreiben Sie (in Worten) wie die Wörter aus der Sprache L aussehen.

(b) [8 PUNKTE] Benutzen Sie die Potenzmengenkonstruktion, um einen deterministischen Automaten zu konstruieren, der zu dem Automaten aus Teil (a) äquivalent ist. (Berechnen Sie nur erreichbare Zustände.)

(c) [3 PUNKTE] Ist der resultierende deterministische Automat schon minimal? Begründen Sie Ihre Antwort.

(d) [15 PUNKTE] Minimieren Sie den folgenden deterministischen Automaten:



Machen Sie dabei Ihren Rechenweg deutlich!

Fortsetzung nächste Seite!

Aufgabe 3 (Kontextfreie Sprachen) [36 PUNKTE]

(a) [10 PUNKTE] Entwerfen Sie eine kontextfreie Grammatik für die folgende kontextfreie Sprache über dem Alphabet $\Sigma = \{a, b, c\}$:

$$L = \{a^{3n+2}wvc^n \mid n \in \mathbb{N}_0, 2 \cdot |w|_b = |v|_a\}.$$

(Hierbei bezeichnet $|u|_x$ die Anzahl des Zeichens x in dem Wort u.)

Erklären Sie den Zweck der einzelnen Nichtterminale (Variablen) und der Grammatikregeln Ihrer Grammatik.

(b) [10 PUNKTE] Betrachten Sie die folgende kontextfreie Grammatik

$$G = (\{A, B, C, D\}, \{a, b, c\}, P, A)$$

mit den Produktionen

$$\begin{aligned} P: \quad A &\to AB \mid CD \mid a \\ B &\to CC \mid c \\ C &\to DC \mid CB \mid b \\ D &\to DB \mid a \end{aligned}$$

Benutzen Sie den Algorithmus von Cocke-Younger-Kasami (CYK), um zu zeigen, dass das Wort *abcab* zu der von G erzeugten Sprache $L(G)$ gehört.

(c) [3 PUNKTE] Finden Sie nun ein größtmögliches Teilwort von *abcab*, dass von keinem der vier Nichtterminale von G ableitbar ist.

(d) [3 PUNKTE] Geben Sie eine Ableitung des Wortes *abcab* mit G an.

(e) [10 PUNKTE] Beweisen Sie, dass die folgende formale Sprache über $\Sigma = \{a, b\}$ nicht kontextfrei ist:

$$L = \{a^n b^{n^2} \mid n \in \mathbb{N}\}.$$

Aufgabe 4 (Entscheidbarkeit) [14 PUNKTE]

Gegeben ist das folgende Entscheidungsproblem:

Eingabe: eine (geeignet codierte) Turingmaschine M, die eine (möglicherweise partielle) Funktion $f_M : \mathbb{N} \to \mathbb{N}$ berechnet

Aufgabe: entscheiden, ob f_M sich von der Fakultätsfunktion unterscheidet

(a) [7 PUNKTE] Ist das gegebene Problem semi-entscheidbar? Begründen Sie Ihre Antwort.

Fortsetzung nächste Seite!

(b) [7 PUNKTE] Ist das gegebene Problem sogar entscheidbar? Begründen Sie auch hier Ihre Antwort.

Aufgabe 5 (NP-Vollständigkeit) [16 PUNKTE]

Beweisen Sie, dass das folgende Problem $\mathcal{NP}$-vollständig ist:

Eingabe: ein ungerichteter Graph mit Knotenmenge V, dessen Knoten mit den Farben Rot und Grün gefärbt sind und eine natürliche Zahl $n \geq 5$

Aufgabe: entscheiden, ob es möglich ist, den Knoten *Prioritäten* zuzuordnen (d.h. Zahlen $p_v \in \{1, \ldots, n\}$, $v \in V$), so dass je zwei durch eine Kante verbundene Knoten verschiedene Priorität haben und die Priorität eines jeden roten Knoten höher ist als die Priorität jedes grünen Knoten (d.h. für jeden roten Knoten u und jeden grünen Knoten v gilt $p_u > p_v$)

Aufgabe 6 (Matrixarithmetik) [68 PUNKTE]

Zur Erinnerung: Für zwei (quadratische) Matrizen $A, B \in \mathbb{R}^{n \times n}$ mit $A = (a_{i,j})_{1 \leq i,j \leq n}$ und $B = (b_{i,j})_{1 \leq i,j \leq n}$ (mit $n \geq 1$) ist

- die *Summe* $A + B$ definiert als die Matrix $S = (s_{i,j})_{1 \leq i,j \leq n}$ mit $s_{i,j} = a_{i,j} + b_{i,j}$,
- die *Differenz* $A - B$ definiert als die Matrix $D = (d_{i,j})_{1 \leq i,j \leq n}$ mit $d_{i,j} = a_{i,j} - b_{i,j}$, und
- das *Produkt* AB definiert als die Matrix $P = (p_{i,j})_{1 \leq i,j \leq n}$ mit $p_{i,j} = \sum_{k=1}^{n} a_{i,k} b_{k,j}$ (d.h. $p_{i,j}$ ist das Skalarprodukt der i-ten Zeile von A mit der j-ten Spalte von B).

Hinweis: In dieser Aufgabe nehmen wir an, dass n eine Zweierpotenz ist, d.h. $n = 2^k$ für ein $k \geq 0$.

(a) [3 Punkte] Geben Sie einen Algorithmus zur Addition zweier $n \times n$ Matrizen in Pseudo-Code an:

MADD(A, B, n)
// Eingabe: Zwei Matrizen $A, B \in \mathbb{R}^{n \times n}$ mit $n \geq 1$ und $A = (a_{i,j})_{1 \leq i,j \leq n}$ und $B = (b_{i,j})_{1 \leq i,j \leq n}$
// Ausgabe: Die Matrix $S = (s_{i,j})_{1 \leq i,j \leq n}$ mit $S = A + B$

(b) [2 Punkte] Bestimmen Sie die Laufzeit Ihrer Implementierung beim Aufruf MADD(A, B, n).

(c) [4 Punkte] Geben Sie einen iterativen Algorithmus zur Multiplikation zweier $n \times n$ Matrizen in Pseudo-Code an, der die Definition des Matrixprodukts direkt umsetzt:

MMULT(A, B, n)
// Eingabe: Zwei Matrizen $A, B \in \mathbb{R}^{n \times n}$ mit $n \geq 1$ und $A = (a_{i,j})_{1 \leq i,j \leq n}$ und $B = (b_{i,j})_{1 \leq i,j \leq n}$
// Ausgabe: Die Matrix $P = (p_{i,j})_{1 \leq i,j \leq n}$ mit $P = AB$

(d) [2 Punkte] Bestimmen Sie die Laufzeit Ihrer Implementierung beim Aufruf MMULT(A, B, n).

Fortsetzung nächste Seite!

Wir wollen nun eine *rekursive* Strategie entwickeln, um für zwei Matrizen $A, B \in \mathbb{R}^{n \times n}$ die Matrix $P \in \mathbb{R}^{n \times n}$ mit $P = AB$ zu berechnen. Dazu zerlegen wir (falls $n > 1$ ist) A, B und P jeweils in vier Blockmatrizen der Größe $n/2 \times n/2$, d.h., wir schreiben

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}, B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}, P = \begin{bmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \end{bmatrix}$$

mit $A_{i,j}, B_{i,j}, P_{i,j} \in \mathbb{R}^{n/2 \times n/2}$.

Mit dem Algorithmus

GETUPPERLEFT(X, n)
// Eingabe: Eine Matrix $X = (x_{i,j})_{1 \le i,j \le n} \in \mathbb{R}^{n \times n}$ mit $n \ge 2$
// Ausgabe: Die Matrix $Z = (z_{i,j})_{1 \le i,j \le n/2} \in \mathbb{R}^{n/2 \times n/2}$ mit $z_{i,j} = x_{i,j}$
1 **for** $i = 1$ **to** $n/2$
2 **for** $j = 1$ **to** $n/2$
3 $z_{i,j} = x_{i,j}$
4 **return** $(z_{i,j})_{1 \le i,j \le n/2}$

kann die Matrix $A_{1,1}$ aus der Matrix A bestimmt werden: $A_{1,1} =$ GETUPPERLEFT(A, n); ebenso kann die Matrix $B_{1,1}$ aus der Matrix B bestimmt werden: $B_{1,1} =$ GETUPPERLEFT(B, n).

(e) [2 Punkte] Bestimmen Sie die Laufzeit des Algorithmus beim Aufruf GETUPPERLEFT(A, n).

(f) [3 Punkte] Geben Sie einen Algorithmus GETUPPERRIGHT an mit dem die Matrix $A_{1,2}$ (bzw. $B_{1,2}$) aus der Matrix A (bzw. B) bestimmt werden kann:

GETUPPERRIGHT(X, n)
// Eingabe: Eine Matrix $X = (x_{i,j})_{1 \le i,j \le n} \in \mathbb{R}^{n \times n}$ mit $n \ge 2$
// Ausgabe: Die Matrix $Z = (z_{i,j})_{1 \le i,j \le n/2} \in \mathbb{R}^{n/2 \times n/2}$ mit $z_{i,j} = x_{i,j+n/2}$

Wir nehmen an, dass uns zwei Algorithmen GETLOWERLEFT und GETLOWERRIGHT zur Verfügung stehen für die gilt:

$$A_{2,1} = \text{GETLOWERLEFT}(A, n),$$
$$A_{2,2} = \text{GETLOWERRIGHT}(A, n),$$
$$B_{2,1} = \text{GETLOWERLEFT}(B, n), \text{ und}$$
$$B_{2,2} = \text{GETLOWERRIGHT}(B, n).$$

Fortsetzung nächste Seite!

Weiterhin gibt es einen Algorithmus COMPOSE der aus vier $n/2 \times n/2$ Matrizen eine $n \times n$ Matrix zusammensetzt:

COMPOSE$(X_{1,1}, X_{1,2}, X_{2,1}, X_{2,2}, n)$
// Eingabe: Vier Matrizen $X_{1,1}, X_{1,2}, X_{2,1}, X_{2,2} \in \mathbb{R}^{n/2 \times n/2}$ mit $n \geq 2$
// Ausgabe: Die Matrix $\begin{bmatrix} X_{1,1} & X_{1,2} \\ X_{2,1} & X_{2,2} \end{bmatrix}$

Man kann nun zeigen, dass gilt:

$$P_{1,1} = A_{1,1}B_{1,1} + A_{1,2}B_{2,1} \qquad P_{1,2} = A_{1,1}B_{1,2} + A_{1,2}B_{2,2}$$
$$P_{2,1} = A_{2,1}B_{1,1} + A_{2,2}B_{2,1} \qquad P_{2,2} = A_{2,1}B_{1,2} + A_{2,2}B_{2,2}$$

Auf dieser Darstellung basiert folgender Algorithmus:

MMULTR(A, B, n)
// Eingabe: Zwei Matrizen $A, B \in \mathbb{R}^{n \times n}$ mit $n \geq 1$ und
$A = (a_{i,j})_{1 \leq i,j \leq n}$ und $B = (b_{i,j})_{1 \leq i,j \leq n}$
// Ausgabe: Die Matrix $P = (p_{i,j})_{1 \leq i,j \leq n}$ mit $P = AB$
1 **if** $n == 1$
2 **return** $(a_{1,1}b_{1,1})$
3 // Erstelle $A_{1,1}, A_{1,2}, A_{2,1}, A_{2,2} \in \mathbb{R}^{n/2 \times n/2}$ mit $A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$
4 $A_{1,1} =$ GETUPPERLEFT(A, n)
5 $A_{1,2} =$ GETUPPERRIGHT(A, n)
6 $A_{2,1} =$ GETLOWERLEFT(A, n)
7 $A_{2,2} =$ GETLOWERRIGHT(A, n)
8 // Erstelle $B_{1,1}, B_{1,2}, B_{2,1}, B_{2,2} \in \mathbb{R}^{n/2 \times n/2}$ mit $B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$
9 $B_{1,1} =$ GETUPPERLEFT(B, n)
10 $B_{1,2} =$ GETUPPERRIGHT(B, n)
11 $B_{2,1} =$ GETLOWERLEFT(B, n)
12 $B_{2,2} =$ GETLOWERRIGHT(B, n)
13 $P_{1,1} =$ MADD(MMULTR$(A_{1,1}, B_{1,1}, n/2)$, MMULTR$(A_{1,2}, B_{2,1}, n/2)$)
14 $P_{1,2} =$ MADD(MMULTR$(A_{1,1}, B_{1,2}, n/2)$, MMULTR$(A_{1,2}, B_{2,2}, n/2)$)
15 $P_{2,1} =$ MADD(MMULTR$(A_{2,1}, B_{1,1}, n/2)$, MMULTR$(A_{2,2}, B_{2,1}, n/2)$)
16 $P_{2,2} =$ MADD(MMULTR$(A_{2,1}, B_{1,2}, n/2)$, MMULTR$(A_{2,2}, B_{2,2}, n/2)$)
17 // Gib die Matrix $\begin{bmatrix} P_{1,1} & P_{1,2} \\ P_{2,1} & P_{2,2} \end{bmatrix}$ zurück
18 **return** COMPOSE$(P_{1,1}, P_{1,2}, P_{2,1}, P_{2,2}, n)$

(g) [9 Punkte] Beweisen Sie, dass der Algorithmus MMULTR korrekt ist.

Fortsetzung nächste Seite!

(h) [8 Punkte] Begründen Sie, dass die Laufzeit $T(n)$ des Algorithmus beim Aufruf MMULTR(A, B, n) folgender asymptotischer Rekursions(un)gleichung genügt:

$$T(n) = 8T(n/2) + O(n^2) \text{ für } n > 1 \qquad \text{und} \qquad T(n) = O(1) \text{ für } n = 1.$$

Sie dürfen dabei annehmen, dass die Algorithmen GETUPPERLEFT, GETUPPERRIGHT, GETLOWERLEFT, GETLOWERRIGHT, MADD und COMPOSE so implementiert sind, dass ihre asymptotische Laufzeit auf $n \times n$ Matrizen $O(n^2)$ ist.

(i) [9 Punkte] Wir betrachten die Rekursionsungleichung (für eine Konstante $c > 0$)

$$T(n) \leq 8 \cdot T(n/2) + c \cdot n^2 \text{ für } n > 1 \qquad \text{und} \qquad T(n) \leq c \text{ für } n \leq 1.$$

Beweisen Sie, dass daraus $T(n) = O(n^3)$ folgt.

Wir wollen eine bessere rekursive Strategie entwickeln, um zwei Matrizen zu multiplizieren. Volker Strassen hat 1969 gezeigt, dass mit

$$M_1 := (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \qquad M_2 := (A_{2,1} + A_{2,2})B_{1,1}$$
$$M_3 := A_{1,1}(B_{1,2} - B_{2,2}) \qquad M_4 := A_{2,2}(B_{2,1} - B_{1,1})$$
$$M_5 := (A_{1,1} + A_{1,2})B_{2,2} \qquad M_6 := (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2})$$
$$M_7 := (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})$$

gilt, dass

$$P_{1,1} = M_1 + M_4 - M_5 + M_7 \qquad P_{1,2} = M_3 + M_5$$
$$P_{2,1} = M_2 + M_4 \qquad P_{2,2} = M_1 - M_2 + M_3 + M_6$$

(j) [9+8 Punkte] Entwickeln Sie auf Basis dieser Identitäten einen rekursiven Algorithmus MMULTVS zur Berechnung des Produktes zweier $n \times n$ Matrizen

MMULTVS (A, B, n)
// Eingabe: Zwei Matritzen A, B, $\in \mathbb{R}^{n \times n}$ mit $n \geq 1$ und
$A = (a_{i,j})_{1 \leq i,j \leq n}$ und $B = (b_{i,j})_{1 \leq i,j \leq n}$
// Ausgabe: Die Matrix $P = (p_{i,j})_{1 \leq i,j \leq n}$ mit $P = AB$

dessen Laufzeit $T(n)$ folgender asymptotischer Rekursions(un)gleichung genügt:

$$T(n) = 7 \cdot T(n/2) + O(n^2) \text{ für } n > 1 \qquad \text{und} \qquad T(n) = O(1) \text{ für } n = 1.$$

Fortsetzung nächste Seite!

Sie dürfen dabei wieder annehmen, dass die Algorithmen GETUPPERLEFT, GETUPPERRIGHT, GETLOWERLEFT, GETLOWERRIGHT, MADD und COMPOSE so implementiert sind, dass ihre asymptotische Laufzeit auf $n \times n$ Matrizen $O(n^2)$ ist. Zudem steht Ihnen ein Algorithmus MSUB zur Subtraktion von Matrizen zur Verfügung:

MSUB (A, B, n)
// Eingabe: Zwei Matritzen A, B, $\in \mathbb{R}^{n \times n}$ mit $n \geq 1$ und
$A = (a_{i,j})_{1 \leq i,j \leq n}$ und $B = (b_{i,j})_{1 \leq i,j \leq n}$
// Ausgabe: Die Matrix $D = (d_{i,j})_{1 \leq i,j \leq n}$ mit $D = A\text{-}B$

Auch dessen asymptotische Laufzeit auf $n \times n$ Matrizen ist $O(n^2)$. Begründen Sie, warum die Laufzeit Ihrer Implementierung von MMULTVS der o.g. Rekursions(un)gleichung genügt.

(k) [9 Punkte] Wir betrachten die Rekursionsungleichung (für eine Konstante $c > 0$)

$$T_{VS}(n) \leq 7 \cdot T_{VS}(n/2) + c \cdot n^2 \text{ für } n > 1 \qquad \text{und} \qquad T_{VS}(n) \leq c \text{ für } n \leq 1.$$

Beweisen Sie, dass $T_{VS}(n) = O(n^{\log_2 7})$.

Aufgabe 7 (Auftragsdisposition) [52 PUNKTE]

Gegeben sei eine Menge $J = \{1, \ldots, n\}$ von n *Jobs*. Jeder Job $i \in J$ hat eine *Startzeit* $s_i \in \mathbb{R}_{>0}$, eine *Endzeit* $f_i \in \mathbb{R}_{>0}$ mit $f_i > s_i$ und einen *Wert* $w_i \in \mathbb{R}_{>0}$. Wir nehmen im Folgenden an, dass alle Zeiten $s_1, \ldots, s_n, f_1, \ldots, f_n$ paarweise verschieden sind. Für eine Menge $S \subseteq J$ von Jobs ist der *Gesamtwert* definiert als

$$w(S) := \sum_{i \in S} w_i.$$

Eine Menge von Jobs $M \subseteq J$ wird *kompatibel* genannt, falls keine zwei verschiedenen Jobs $l, m \in M$ existieren, welche sich zeitlich überlappen. Das heißt für alle $l, m \in M$ mit $l \neq m$ gilt dann entweder $f_m < s_l$ oder $f_l < s_m$.

Gesucht ist eine Menge $S \subseteq J$ von kompatiblen Jobs, deren Gesamtwert $w(S)$ *maximal* ist. Diesen Wert bezeichnen wir mit $OPT(J)$.

Wir nehmen im Folgenden an, dass die Jobs nach ihren Endzeiten f_i aufsteigend sortiert sind und setzen $f_0 := 0$. Nun betrachten wir für $0 \leq i \leq n$ die Menge

$$J_i := \{j \in J | f_j \leq f_i\}$$

von Jobs, welche alle zur Zeit f_i beendet sind.

Fortsetzung nächste Seite!

(a) [2 Punkte] Zeigen Sie, dass $J_0 = \emptyset$ und $J_i = \{1, 2, \ldots, i\}$ für $1 \leq i \leq n$.

Für $0 \leq i \leq n$ sei $OPT(i)$ der optimale Gesamtwert für die Menge J_i, d.h.

$$OPT(i) := OPT(J_i)$$

(b) [1 Punkt] Welcher Zusammenhang besteht zwischen den $OPT(i)$ und dem gesuchten Wert $OPT(J)$?

(c) [2 Punkte] Was sind die Werte für $OPT(0)$ und $OPT(1)$?

Sei $p(j)$ der Job mit dem größten Index kleiner als j, welcher kompatibel mit Job j ist. Mit anderen Worten, das größte $k < j$, für welches $f_k < s_j$ gilt. Sollte ein solcher Job nicht existieren, so ist $p(j) := 0$ per definitionem.

(d) [9 Punkte] Zeigen Sie, dass für $i \geq 1$ gilt

$$OPT(i) = \max(OPT(i-1), w_i + OPT(p(i))).$$

(e) [4 Punkte] Formulieren Sie auf Basis dieser Rekursionsgleichung einen rekursiven Algorithmus zur Berechnung von $OPT(i)$ in Pseudocode:

DispRec(S, F, W, i, P)
// Eingabe: Eine Instanz des Dispositionsproblems, beschrieben durch
// die Folge $S = (s_1, \ldots, s_n)$ der Startzeiten
// die Folge $F = (f_1, \ldots, f_n)$ der Endzeiten mit $f_1 < \cdots < f_n$
// die Folge $W = (w_1, \ldots, w_n)$ der Werte
// ein Index $0 \leq i \leq n$
// die Folge $P = (p(1), \ldots, p(n))$
// Ausgabe: $OPT(i)$

(f) [4 Punkte] Beschreiben Sie eine Menge von n Jobs durch die Angabe der Start- und Endzeiten $s_1, ..., s_n, f_1, ..., f_n$ für die gilt, dass für alle $1 \leq j \leq n$ gilt: $p(j) = j - 1$.

(g) [4 Punkte] Wir betrachten nun eine Eingabe „der Länge n“ wie aus der vorigen Aufgabe für die also gilt, dass für alle $1 \leq j \leq n$ gilt: $p(j) = j - 1$. Begründen Sie, dass für so eine Eingabe die Laufzeit $T(n)$ des rekursiven Algorithmus beim Aufruf DispRec(S, F, W, n, P) folgender asymptotischer Rekursionsgleichung genügt:

$$T(n) = 2T(n-1) + \Theta(1)$$
$$(\text{mit } T(n) = \Theta(1) \text{ für } n = O(1))$$

(h) [2+2 Punkte] Wir betrachten die Rekursionsungleichung (für eine Konstante $c > 0$)

$$T(n) \geq 2 \cdot T(n-1) + c \text{ für } n > 1 \qquad \text{und} \qquad T(n) \geq c \text{ für } n \leq 1.$$

Beweisen Sie, dass $T(n) = \Omega(2^n)$.

Argumentieren Sie damit, warum eine direkte Implementierung der Rekursion im Allgemeinen nicht effizient ist.

Fortsetzung nächste Seite!

(i) [4+3+3 Punkte] Geben Sie einen Algorithmus in Pseudocode an, der für $1 \leq j \leq n$ die Werte $p(j)$ berechnet:

ComputeP(S, F, W)
// Eingabe: Eine Instanz des Dispositionsproblems, beschrieben durch
// die Folge $S = (s_1, \ldots, s_n)$ der Startzeiten
// die Folge $F = (f_1, \ldots, f_n)$ der Endzeiten mit $f_1 < \cdots < f_n$
// die Folge $W = (w_1, \ldots, w_n)$ der Werte
// Ausgabe: Die Folge $P = (p(1), \ldots, p(n))$

Begründen Sie, warum Ihr Algorithmus korrekt ist und analysieren Sie dessen Laufzeit.

(j) [4+6+2 Punkte] Formulieren Sie ein dynamisches Programm zur Berechnung von $OPT(n)$ in Pseudocode:

DispDP(S, F, W, P)
// Eingabe: Eine Instanz des Dispositionsproblems, beschrieben durch
// die Folge $S = (s_1, \ldots, s_n)$ der Startzeiten
// die Folge $F = (f_1, \ldots, f_n)$ der Endzeiten mit $f_1 < \cdots < f_n$
// die Folge $W = (w_1, \ldots, w_n)$ der Werte
// die Folge $P = (p(1), \ldots, p(n))$
// Ausgabe: $OPT(n)$

Begründen Sie, warum Ihr Algorithmus korrekt ist und analysieren Sie dessen Laufzeit.

Thema Nr. 2
(Aufgabengruppe)

Es sind alle Aufgaben dieser Aufgabengruppe zu bearbeiten!

Aufgabe 1 (Aussagen) [18 PUNKTE]

Im Folgenden seien L und L' formale Sprachen. Zeigen oder widerlegen Sie die folgenden Aussagen (die jeweiligen Beweise sind sehr kurz).

Schreiben Sie zunächst zur Aussage "Richtig" oder "Falsch" und dann Ihre Begründung.

(a) [6 Punkte] Sind L und L' regulär, so ist auch $L \setminus L'$ regulär.

(b) [6 Punkte] Sind L und L' kontextfrei, so ist auch $L \setminus L'$ kontextfrei.

(c) [6 Punkte] Ist L entscheidbar und L' semi-entscheidbar, so ist $L \setminus L'$ semi-entscheidbar.

Aufgabe 2 (Automatentheorie und reguläre Sprachen) [35 PUNKTE]

Betrachten Sie den regulären Ausdruck $\alpha = (0 \cup 1)^*(00 \cup 11)(0 \cup 1)$ und die zugehörige Sprache $A = L(\alpha)$.
(Alternative Schreibweise $\alpha = (0 + 1)^*(00 + 11)(0 + 1)$.)

(a) [3 Punkte] Beschreiben Sie Gestalt der Wörter der Sprache A in Worten.

(b) [5 Punkte] Geben Sie einen nicht-deterministischen endlichen Automaten M mit $L(M) = L(\alpha)$ an.

(c) [12 Punkte] Überführen Sie den Automaten M aus Teilaufgabe (b) mittels der Teilmengenkonstruktion in einen deterministischen endlichen Automaten M' mit $L(M') = L(M)$. Eine Darstellung des Automaten in Tabellenform ist ausreichend.

(d) [12 Punkte] Betrachten Sie die zur Sprache A gehörende Myhill-Nerode-Relation $\equiv_A$ und die folgenden Wörter:

$w_0 = 000$, $w_1 = 001$, $w_2 = 010$, $w_3 = 011$, $w_4 = 100$, $w_5 = 101$, $w_6 = 110$, $w_7 = 111$

Zeigen Sie, dass die Wörter $w_0, \ldots, w_7$ paarweise nicht äquivalent sind, also in verschiedenen Äquivalenzklassen von $\equiv_A$ liegen.

Hinweis: Es ist nicht nötig alle 28 möglichen Paare von Wörtern einzeln zu betrachten; überlegen Sie, wie jeweils mehrere Paare von Wörtern gleichzeitig betrachtet werden können.

(e) [3 Punkte] Was folgt aus Aufgabenteil (d) für die Anzahl der Zustände eines deterministischen endlichen Automaten, der A erkennt? Begründen Sie Ihre Antwort.

Fortsetzung nächste Seite!

Aufgabe 3 (Kontextfreie Sprachen) [20 PUNKTE]

(a) [4 Punkte] Ist die folgende Sprache $L_1 = \{a^{n+2}b^{2n+1} \mid n \geq 2\}$ über dem Alphabet $\Sigma = \{a, b\}$ kontextfrei?

Falls ja, geben Sie eine kontextfreie Grammatik für L_1 an, falls nein, eine kurze Begründung (ein vollständiger Beweis ist hier nicht gefordert).

(b) [12 Punkte] Geben Sie einen Kellerautomaten (PDA) formal an, der die Sprache $L_2 = \{w_1w_2w_3 \mid w_1, w_2, w_3 \in \Sigma^* \setminus \{\lambda\} \text{ und } w_1 = w_3^{\text{rev}}\} \in \text{CFL}$ über dem Alphabet $\Sigma = \{0, 1\}$ akzeptiert.

Dabei bezeichnet λ das leere Wort und w_3^{rev} bezeichnet das Wort w_3 rückwärts gelesen. Bei Akzeptanz einer Eingabe soll sich der PDA in einem Endzustand befinden und der Keller geleert sein.

(c) [4 Punkte] Beschreiben Sie in Worten die Arbeitsweise Ihres PDA aus Aufgabenteil (b).

Aufgabe 4 (Berechenbarkeitstheorie) [27 PUNKTE]

Sei $A = \{\langle M \rangle \mid M \text{ ist Turingmaschine, die bei Eingabe 101 hält }\}$. Dabei bezeichnet $\langle M \rangle$ die Gödelnummer der Turingmaschine M.

(a) [12 Punkte] Zeigen Sie, dass A unentscheidbar ist.

(b) [5 Punkte] Zeigen Sie, dass A semi-entscheidbar ist.

(c) [10 Punkte] Ist das Komplement A^c von A entscheidbar? Ist es semi-entscheidbar? Begründen Sie Ihre Antworten.

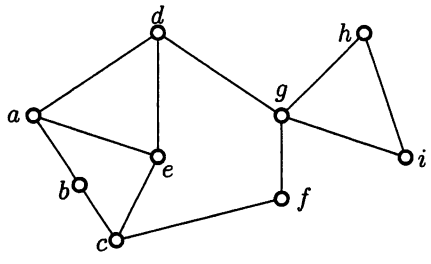
Hinweis: Sie können die Aussagen aus Teilaufgabe a) und b) verwenden, auch wenn Sie sie nicht bewiesen haben.

Aufgabe 5 (Komplexitätstheorie) [20 PUNKTE]

Sei $G = (V, E)$ ein ungerichteter Graph. Eine *2-Kreisüberdeckung* von G besteht aus zwei einfachen Kreisen C_1 und C_2 in G, sodass jeder Knoten von G in mindestens einem der Kreise C_1 oder C_2 enthalten ist.

Das Problem 2-KREISÜBERDECKUNG besteht darin, für einen gegebenen Graphen $G = (V, E)$ zu entscheiden, ob G eine 2-Kreisüberdeckung besitzt.

(a) [4 Punkte] Geben Sie eine 2-Kreisüberdeckung für den folgenden Graph an.



Fortsetzung nächste Seite!

(b) [6 Punkte] Zeigen Sie, dass das Problem 2-KREISÜBERDECKUNG in NP liegt.

(c) [10 Punkte] Zeigen Sie, dass das Problem 2-KREISÜBERDECKUNG NP-vollständig ist.

Hinweis: Sie können ohne Beweis benutzen, dass das Problem HAMILTONKREIS, das darin besteht zu entscheiden ob ein gegebener Graph einen einfachen Kreis besitzt, der alle Knoten enthält, NP-schwer ist.

Aufgabe 6 (Sortieren) [9 PUNKTE]

Ein bekanntes Sortierverfahren ist Insertion-Sort, ein inkrementeller Algorithmus. Insertion-Sort geht im i-ten Durchlauf davon aus, dass das Teilfeld bis zur Stelle $i-1$ schon sortiert ist. Dann speichert Insertion-Sort das i-te Element des gegebenen Feldes zwischen, um es an der korrekten Stelle im schon sortierten Teilfeld einzufügen.

(a) [3 Punkte] Analysieren Sie, wie viele Schlüsselvergleiche Insertion-Sort im besten und im schlechtesten Fall benötigt, um ein Feld von n verschiedenen Zahlen zu sortieren. Geben Sie jeweils die genaue Anzahl an.

(b) [6 Punkte] Sei nun die Eingabe ein Feld mit den Zahlen $1, 2, \ldots, n$ in *zufälliger* Reihenfolge. Schätzen Sie asymptotisch scharf ab, wie viele Schlüsselvergleiche Insertion-Sort in diesem Fall vornimmt.

Tipp: Gehen Sie davon aus, dass n durch 4 teilbar ist. Betrachten Sie der Einfachheit halber nur die Elemente im letzten Viertel des Feldes. Wie weit müssen manche (wie viele?) dieser Elemente nach links wandern?

Aufgabe 7 (Dijkstras Algorithmus) [21 PUNKTE]

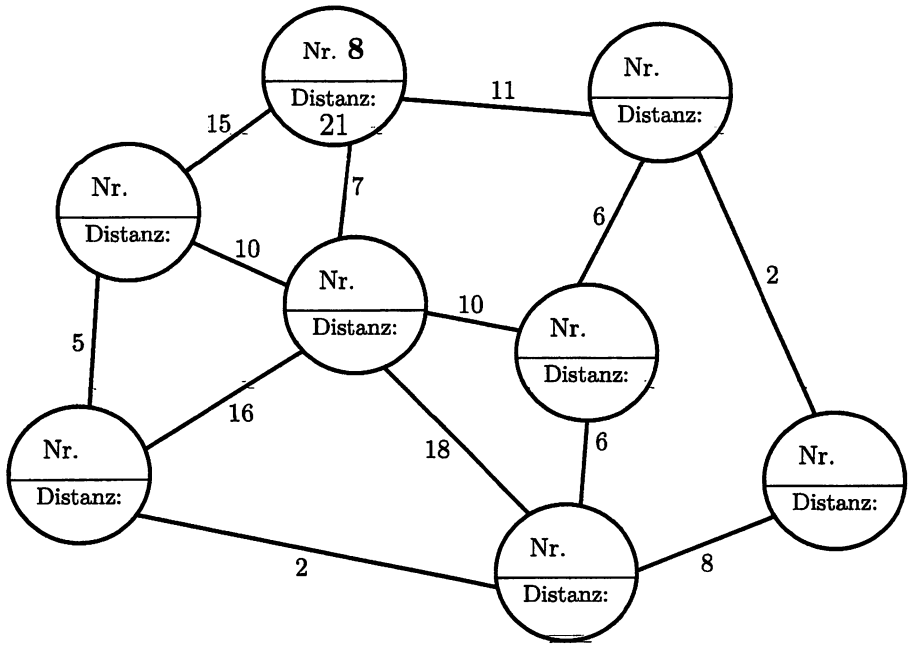
Auf folgendem ungerichteten Graphen wurde Dijkstras Algorithmus (wie unten beschrieben) ausgeführt, doch wir wissen lediglich, welcher Knoten als letztes schwarz (*black*) wurde (Nr. 8) und was seine Distanz zum Startknoten (Nr. 1) ist. Die Gewichte der Kanten sind angegeben.

Finden Sie den Startknoten, nummerieren Sie die Knoten in der Reihenfolge, in der sie schwarz wurden und geben Sie in jedem Knoten die Distanz zum Startknoten an.

Markieren Sie die Kanten, die zu dem Kürzesten-Wege-Baum gehören, den Dijkstras Algorithmus ausgibt.

Der Startknoten ist eindeutig.

Fortsetzung nächste Seite!



Dijkstra(Graph G, Edgeweights w, Vertex s)

```
Initialize(G, s)
S = ∅
Q = new PriorityQueue(V, d)
while not Q.Empty() do
    u = Q.ExtractMin()
    S = S ∪ {u}
    foreach v ∈ Adj[u] do
        Relax(u, v, w)
    u.color = black
```

Initialize(Graph G, Vertex s)

```
foreach u ∈ V do
    u.color = white
    u.d = ∞
s.color = gray
s.d = 0
```

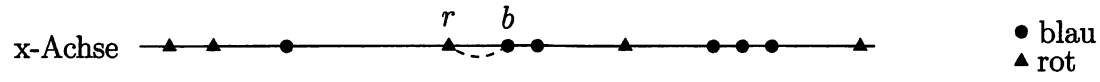
Relax(Vertex u, Vertex v, Edgeweights w)

```
if v.d > u.d + w(u, v) then
    v.color = gray
    v.d = u.d + w(u, v)
    Q.DecreaseKey(v, v.d)
```

Fortsetzung nächste Seite!

Aufgabe 8 (Nächstes rot-blaues Paar auf der x-Achse) [40 PUNKTE]

Gegeben seien zwei nichtleere Mengen R und B von roten bzw. blauen Punkten auf der x-Achse. Gesucht ist der minimale euklidische Abstand $d(r, b)$ über alle Punktepaare (r, b) mit $r \in R$ und $b \in B$. Hier ist eine Beispielinstanz:



Die Eingabe wird in einem Feld A übergeben. Jeder Punkt $A[i]$ mit $1 \leq i \leq n$ hat eine x-Koordinate $A[i].\text{x}$ und eine Farbe $A[i].\text{color} \in \{$ rot, blau $\}$. Das Feld A ist nach x-Koordinate sortiert, d.h. es gilt $A[1].\text{x} < A[2].\text{x} < \cdots < A[n].\text{x}$, wobei $n = |R| + |B|$.

(a) [10 Punkte] Geben Sie in Worten einen Algorithmus an, der den gesuchten Abstand in $O(n)$ Zeit berechnet.

(b) [10 Punkte] Begründen Sie kurz die Laufzeit Ihres Algorithmus.

(c) [10 Punkte] Begründen Sie die Korrektheit Ihres Algorithmus.

(d) [10 Punkte] Wir betrachten nun den Spezialfall, dass alle blauen Punkte links von allen roten Punkten liegen. Beschreiben Sie in Worten, wie man in dieser Situation den gesuchten Abstand in $o(n)$ Zeit berechnen kann. (Ihr Algorithmus darf also insbesondere nicht Laufzeit $\Theta(n)$ haben.)

Aufgabe 9 (Heaps) [20 PUNKTE]

Sei H ein Max-Heap, der n Elemente speichert. Für ein Element v in H sei $h(v)$ die Höhe von v, also die Länge eines längsten Pfades von v zu einem Blatt im Teilheap mit Wurzel v.

(a) [6 Punkte] Geben Sie eine rekursive Definition von $h(v)$ an, in der Sie sich auf die Höhen der Kinder $v.\text{left}$ und $v.\text{right}$ von v beziehen (falls v Kinder hat).

(b) [8 Punkte] Geben Sie eine möglichst niedrige obere asymptotische Schranke für die Summe der Höhen aller Elemente in H an, also für $\sum_{v \in H} h(v)$, und begründen Sie diese.

Tipp: Denken Sie daran, wie man aus einem beliebigen Feld einen Max-Heap macht.

(c) [6 Punkte] Sei H' ein Feld der Länge n. Geben Sie einen Algorithmus an, der in Linearzeit testet, ob H' ein Max-Heap ist.

Fortsetzung nächste Seite!

Aufgabe 10 (Binärbäume) [30 PUNKTE]

Sei B ein binärer Suchbaum. In jedem Knoten v von B wird ein Schlüssel $v.\text{key} \in \mathbb{N}$ gespeichert sowie Zeiger $v.\text{left}$, $v.\text{right}$ und $v.\text{parent}$ auf sein linkes Kind, auf sein rechtes Kind und auf seinen Elternknoten. Die Zeiger sind *nil*, wenn der entsprechende Nachbar nicht existiert. Für zwei Knoten u und v ist wie üblich der *Abstand* die Anzahl der Kanten auf dem kürzesten Pfad von u nach v.

Für einen Knoten w von B sei $B(w)$ der Teilbaum von B mit Wurzel w. Für zwei Knoten u und v von B ist w ein *gemeinsamer Vorfahre*, wenn u und v in $B(w)$ liegen. Wir suchen den *niedrigsten gemeinsamen Vorfahren* $\text{ngV}(u, v)$ von u und v, also einen gemeinsamen Vorfahren w, so dass für jeden Vorfahren w' von u und v gilt, dass w in $B(w')$ liegt. Wir betrachten verschiedene Szenarien, in denen Sie jeweils den niedrigsten gemeinsamen Vorfahren von u und v berechnen sollen.

(a) [10 Punkte] Wir bekommen u und v als Zeiger auf die entsprechenden Knoten in B geliefert. Beschreiben Sie in Worten und in Pseudocode einen Algorithmus, der den niedrigsten gemeinsamen Vorfahren von u und v berechnet. Analysieren Sie die Laufzeit Ihres Algorithmus.

(b) [10 Punkte] Wir bekommen u und v wieder als Zeiger auf die entsprechenden Knoten in B geliefert. Seien d_u und d_v die Abstände von u bzw. v zum niedrigsten gemeinsamen Vorfahren von u und v. Die Laufzeit Ihres Algorithmus soll $O(\max\{d_u, d_v\})$ sein. Dabei kann Ihr Algorithmus in jedem Knoten v eine Information $v.\text{info}$ speichern. Skizzieren Sie Ihren Algorithmus in Worten.

(c) [10 Punkte] Wir bekommen die Schlüssel $u.\text{key}$ und $v.\text{key}$. Die Laufzeit Ihres Algorithmus soll proportional zum Abstand der Wurzel von B zum niedrigsten gemeinsamen Vorfahren von u und v sein. Skizzieren Sie Ihren Algorithmus in Worten.