2. Klausur zur Vorlesung
Theoretische Grundlagen der Informatik
Wintersemester 2018/2019

Lösung!

Beachten Sie:

- Bringen Sie den Aufkleber mit Ihrem Namen und Matrikelnummer auf diesem Deckblatt an und beschriften Sie jedes Aufgabenblatt mit Ihrem Namen und Matrikelnummer.
- Schreiben Sie die Lösungen auf die Aufgabenblätter und Rückseiten. Zusätzliches Papier erhalten Sie bei Bedarf von der Aufsicht.
- Es sind keine Hilfsmittel zugelassen.

	Mögliche Punkte					Erreichte Punkte				
	a	b	c	d	Σ	a	b	c	d	Σ
Aufg. 1	3	3	2	–	8				–	
Aufg. 2	2	3	3	–	8				–	
Aufg. 3	2	3	3	–	8				–	
Aufg. 4	3	4	2	–	9				–	
Aufg. 5	10				10					
Aufg. 6	2	3	2	2	9					
Aufg. 7	4	3	1	–	8				–	
Σ					60					

Problem 1: Endliche Automaten 3 + 3 + 2 = 8 Punkte

In der Vorlesung haben Sie fehlertolerante Kodierungen kennengelernt. Diese können auch dann noch dekodiert werden, wenn einige Bits falsch übertragen wurden. In diesem Kontext wurde die Hammingdistanz $d(x, y)$ definiert. Für zwei Wörter x, y über demselben endlichen Alphabet Σ und mit derselben Länge ist $d(x, y)$ die Anzahl der Zeichen, die sich in x und y unterscheiden.

Sei L eine reguläre Sprache über dem Alphabet Σ. Definiere dazu die Sprache $B_k(L)$ wie folgt:

$$B_k(L) = \{x \in \Sigma^\star \mid \exists y \in L : |x| = |y| \text{ und } d(x, y) \leq k\}$$

Die Sprache $B_k(L)$ enthält also die Wörter, die aus Wörtern in L durch Ersetzen von höchstens k Zeichen entstehen.

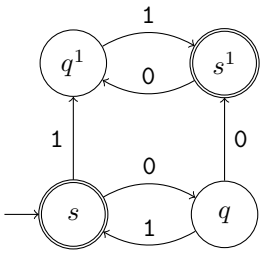
(a) Sei $\Sigma = \{\mathtt{0}, \mathtt{1}\}$ und $L_{\mathtt{01}} = \{\mathtt{01}\}^\star$ und $k = 1$. Geben Sie den Zustandsübergangsgraphen eines deterministischen endlichen Automaten an, der $B_1(L_{\mathtt{01}})$ entscheidet. Sie dürfen einen impliziten Fehlerzustand benutzen. Ohne Fehlerzustand soll Ihr Automat 4 Zustände haben.

(b) Sei weiterhin $k = 1$. Zeigen Sie, dass die regulären Sprachen unter Anwendung von B_1 abgeschlossen sind. Beschreiben Sie dazu, wie aus einem endlichen Automaten für eine reguläre Sprache L ein endlicher Automat für $B_k(L)$ konstruiert werden kann. Dabei muss Ihr Automat für $B_k(L)$ nicht notwendigerweise deterministisch sein.

(c) Nun sei k beliebig, aber fest. Wie muss Ihre Konstruktion aus Aufgabenteil (b) angepasst werden, um zu zeigen, dass die regulären Sprachen unter Anwendung von B_k abgeschlossen sind?

Lösung:

(a)



(b) Sei L eine reguläre Sprache. Dann existiert ein deterministischer endlicher Automat $\mathcal{A} = (Q, \Sigma, \delta, s, F)$, der L entscheidet. Konstruiere einen neuen nichtdeterministischen endlichen Automaten $\mathcal{A}'$ folgendermaßen. Erstelle 2 Kopien von $\mathcal{A}$. Bezeichne für einen Zustand $q \in Q$ die Kopien mit q^0 und q^1. Füge nun für jeden Übergang $\delta(q, a) = r$ in $\mathcal{A}$ die Übergänge $\delta'(q^i, a) = r^i$ in $\mathcal{A}'$ ein. Füge außerdem für jedes Symbol $x \in \Sigma$ mit $x \neq a$ den Übergang $\delta'(q^0, x) = r^1$ in $\mathcal{A}'$ ein.

Durch die Übergänge von der ersten Kopie von $\mathcal{A}$ in die zweite Kopie kann der Automat bei genau einem Zeichen „schummeln“ und stattdessen ein beliebiges anderes Zeichen lesen. Dadurch werden Wörter erkannt, die sich in genau einem Zeichen von Wörtern aus L unterscheiden. Da es aber keine Übergänge zurück in die erste Kopie gibt, lehnt $\mathcal{A}'$ bei mehr als einem Zeichen Unterschied ab.

(c) Statt 2 Kopien werden nun $k + 1$ Kopien von $\mathcal{A}$ erstellen. Bezeichne für einen Zustand $q \in Q$ die Kopien mit $q^0, q^1, \ldots, q^k$. Füge für jeden Übergang $\delta(q, a) = r$ in $\mathcal{A}$ wiederum die Übergänge $\delta'(q^i, a) = r^i$ in $\mathcal{A}'$ ein. Füge außerdem für jedes Symbol $x \in \Sigma$ mit $x \neq a$ und jedes $0 \leq i < k$ den Übergang $\delta'(q^i, x) = r^{i+1}$ in $\mathcal{A}'$ ein.

Von Kopie i kann man also durch Lesen eines falschen Zeichens in Kopie $i + 1$ wechseln. Dadurch können k falsche Zeichen gelesen werden, aber nicht mehr.

Problem 2: Reguläre Sprachen 2 + 3 + 3 = 8 Punkte

Gegeben seien das Alphabet $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ und die Sprache $L = \{w \in \Sigma^\star \mid |w|_\mathtt{a} = |w|_\mathtt{b}\}$, wobei $|w|_x$ die Anzahl der Vorkommen des Symbols x in w bezeichnet.

(a) Beweisen Sie mithilfe des Pumping-Lemmas für reguläre Sprachen, dass L nicht regulär ist.

Wir betrachten nun folgende abgeschwächte Variante des Pumping-Lemmas für reguläre Sprachen:

> Sei L eine reguläre Sprache. Dann existiert eine Zahl $n \in \mathbb{N}$, so dass für jedes Wort $w \in L$ mit $|w| > n$ eine Darstellung
>
> $$w = uvx \text{ mit } |v| \leq n, v \neq \varepsilon$$
>
> existiert, bei der auch $uv^ix \in L$ ist für alle $i \in \mathbb{N}_0$.

Im Gegensatz zum Pumping-Lemma, das Sie aus der Vorlesung kennen, wird hier also statt $|uv| \leq n$ nur $|v| \leq n$ gefordert.

(b) Zeigen Sie, dass L die Aussage des abgeschwächten Pumping-Lemmas erfüllt.

(c) Beweisen Sie mithilfe der Nerode-Relation, dass L nicht regulär ist.

Lösung:

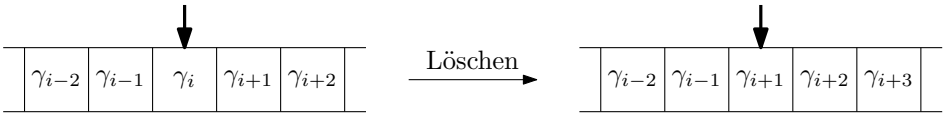
(a) Betrachte für ein gegebenes $n \in \mathbb{N}$ das Wort $w = \mathtt{a}^n\mathtt{b}^n$ der Länge $2n > n$. Offensichtlich gilt $w \in L$. Für jede Zerlegung $w = uvx$ mit $|uv| \leq n$ und $v \neq \varepsilon$ gilt: $v = \mathtt{a}^j$ für ein $1 \leq j \leq n$. Dann gilt $uv^ix = \mathtt{a}^{n+(i-1)j}\mathtt{b}^n$. Für $i \neq 1$ ist $uv^ix \notin L$. Widerspruch zur Aussage des Pumping-Lemmas, also ist L nicht regulär.

(b) Wir wählen $n = 2$, betrachten also alle Wörter $w \in L$ mit $|w| \geq 3$. Jedes solche Wort w muss mindestens ein $\mathtt{a}$ und ein $\mathtt{b}$ enthalten. Es gibt also eine Position i, sodass das Teilwort w_iw_{i+1} entweder $\mathtt{ab}$ oder $\mathtt{ba}$ ist. Wir wählen $u = w_1 \dots w_{i-1}$, $v = w_iw_{i+1}$ und $x = w_{i+2} \dots w_{|w|}$. Wenn vorher $|w|_\mathtt{a} = |w|_\mathtt{b}$ galt, so ändert sich dies durch Hinzufügen oder Entfernen von $\mathtt{ab}$ oder $\mathtt{ba}$ nicht. Also gilt $uv^ix \in L$ für alle $i \in \mathbb{N}_0$.

(c) L ist genau dann nicht regulär, wenn die Nerode-Relation R_L unendlich viele Äquivalenzklassen besitzt. Um das zu beweisen, geben wir unendlich viele Wörter über $\Sigma^\star$ an, die in der Nerode-Relation paarweise nicht äquivalent sind. Betrachte für $i, j \in \mathbb{N}$ mit $i \neq j$ die Wörter $\mathtt{a}^i$ und $\mathtt{a}^j$. Betrachte als Suffix $z = \mathtt{b}^i$. Dann gilt $\mathtt{a}^i\mathtt{b}^i \in L$, aber $\mathtt{a}^j\mathtt{b}^i \notin L$. Also sind $\mathtt{a}^i$ und $\mathtt{a}^j$ nicht äquivalent. Da dieses Argument für beliebige Wahl von i und j gilt, gibt es unendlich viele nicht äquivalente Wörter.

Problem 3: Turingmaschinen 2 + 3 + 3 = 8 Punkte

Eine *löschende Turingmaschine* (LTM) ist eine Turingmaschine, der zusätzlich zu den Kopfbewegungen *links*, *keine Bewegung* und *rechts* die Aktion *löschen* zur Verfügung steht. Beim Löschen wird das Feld, auf dem der Kopf der Turingmaschine steht, aus dem Band gelöscht und der Kopf bewegt sich auf das Feld, das zuvor der rechte Nachbar des nun gelöschten Feldes war:



Sei $L = \{w\#w^R \mid w \in \{0,1\}^*\}$.

(a) Welche Laufzeit braucht eine Turingmaschine mit einem Band, um L zu entscheiden? Wählen Sie aus den folgenden Möglichkeiten die schärfste untere Schranke für die Worst-Case-Laufzeit aus.

- ☐ $\Omega(\log n)$
- ☐ $\Omega(n)$
- ☐ $\Omega(n^2)$
- ☐ $\Omega(2^n)$

Begründen Sie Ihre Entscheidung kurz (ohne formalen Beweis).

(b) Beschreiben Sie, wie man L mit einer LTM in Linearzeit entscheiden kann. Belegen Sie die Laufzeit.

(c) Sind Turingmaschinen und löschende Turingmaschinen gleich mächtig? Beweisen Sie.

Lösung:

(a) $\Omega(n^2)$. Eine Turingmaschine „muss“ je zwei Zeichen von w und w^R vergleichen. Dabei müssen insgesamt Distanzen von $\Omega(\Sigma_{i=1}^{n} i) = \Omega(n^2)$ auf dem Band zurückgelegt werden.

(b) Suche die Mitte des Worts, die mit # markiert ist, oder lehne die Eingabe ab, falls keine solche Markierung existiert. Dies ist in linearer Laufzeit möglich. Lösche dann dieses Symbol. Vergleiche nun das aktuelle Symbol mit dem linken Nachbarsymbol. Gleichen sich die Symbole nicht, so lehne die Eingabe ab. Gleichen sich die Symbole, so lösche beide und wiederhole diesen Schritt, bis die komplette Eingabe gelöscht wurde. Dies ist wiederum in linearer Laufzeit möglich.

(c) Ja. Eine LTM kann trivialerweise eine TM simulieren. Umgekehrt kann eine TM $\mathcal{M}$ auch eine LTM $\mathcal{L}$ simulieren. Löscht $\mathcal{L}$ ein Zeichen, so muss $\mathcal{M}$ den Teil des Bandes rechts vom aktuellen Bandsymbol um eine Position nach links verschieben. Das Ende des Bandes muss dazu z.B. durch ein zusätzliches Bandsymbol geeignet markiert werden.

Problem 4: Aufzählbarkeit $3 + 4 + 2 = 9$ Punkte

Ein Dieb ist auf der Flucht, und Sie sollen ihn fangen! Der Dieb befindet sich auf dem Zahlenstrahl der ganzen Zahlen $\mathbb{Z}$ an einem Ihnen unbekannten Punkt $x_0 \in \mathbb{Z}$. Zunächst gehen wir davon aus, dass der Dieb sich nicht bewegt. Ihre Aufgabe ist es, den Dieb zu fangen. Dazu dürfen Sie an jedem Tag an genau einem Punkt nachsehen, ob der Dieb sich dort befindet. Falls ja, haben Sie ihn gefangen. Falls nicht, müssen Sie am nächsten Tag weitersuchen.

Mit einer *Strategie* zum Fangen des Diebes bezeichnen wir eine unendliche Folge von Punkten $s_1, s_2, \ldots$, so dass der Dieb nach endlich vielen Tagen gefangen wird.

(a) Geben Sie eine Strategie zum Fangen des Diebes an. Begründen Sie, dass der Dieb tatsächlich nach endlich vielen Tagen gefangen wird.

Nun darf sich der Dieb nachts bewegen. Dafür gehen wir davon aus, dass er an der Position $x_0 = 0$ startet. In jeder Nacht bewegt der Dieb sich um einen festen, aber Ihnen wiederum unbekannten Wert $v \in \mathbb{Z}$ fort. Tagsüber schläft der Dieb. Am ersten Tag schläft der Dieb also am Ort v, am zweiten Tag am Punkt $2v$ und so weiter. Beachten Sie, dass v positiv, negativ oder 0 sein kann.

(b) Geben Sie eine Strategie zum Fangen des Diebes an. Begründen Sie, dass der Dieb tatsächlich nach endlich vielen Tagen gefangen wird.

Achtung: Wenn Sie die erste Position überprüfen, hat sich der Dieb bereits einmal bewegt. Er befindet sich also nicht mehr unbedingt an der Position 0*, sondern an der Position* v*.*

Nun sind sowohl x_0 als auch v fest, aber Ihnen unbekannt. Am ersten Tag schläft der Dieb also am Ort $x_0 + v$, am zweiten Tag am Punkt $x_0 + 2v$ und so weiter.

(c) Geben Sie eine Strategie zum Fangen des Diebes an. Begründen Sie, dass der Dieb tatsächlich nach endlich vielen Tagen gefangen wird.

Lösung:

(a) Wählt man die Strategie

$$0, 1, -1, 2, -2, 3, -3, \ldots,$$

so wird der Dieb nach $\mathcal{O}(x_0)$, also endlich vielen Schritten gefangen.

(b) Am i-ten Tag befindet sich der Dieb am Punkt $x_0 + v \cdot i = v \cdot i$. Überprüfe Tag für Tag alle möglichen Geschwindigkeiten. Wählt man die Strategie

$$0 \cdot 1, 1 \cdot 2, -1 \cdot 3, 2 \cdot 4, -2 \cdot 5, \ldots,$$

wird der Dieb also nach endlich vielen Schritten gefangen.

(c) Am i-ten Tag befindet sich der Dieb am Punkt $x_0 + v \cdot i$. Überprüfe Tag für Tag alle möglichen Kombinationen aus Anfangspunkt x_0 und Geschwindigkeit v. Eine Möglichkeit ist die Reihenfolge

$$(0,0), (1,0), (1,1), (0,1), (-1,1), (-1,0), (-1,-1), (0,-1), (1,-1), (2,0), \ldots,$$

die sich spiralförmig vom Ursprung ausdehnt. Ähnlich wurde in der zweiten Aufgabe des vierten Übungsblattes vorgegangen. So wird der Dieb nach endlich vielen Tagen gefunden.

Problem 5: NP-Vollständigkeit 10 Punkte

Sie sind Aufseher eines internationalen Gefängnisses mit $3n$ Gefangenen und n Zellen für jeweils drei Insassen. Jeder Gefangene beherrscht eine bestimmte Menge an Sprachen, wobei es insgesamt m Sprachen gibt. Die Sprachen, die die Gefangenen sprechen, sind durch die $3n \times m$-Matrix M gegeben:

$$M[i][j] := \begin{cases} 1 & \text{falls Gefangener } i \text{ Sprache } j \text{ spricht} \\ 0 & \text{sonst.} \end{cases}$$

Sie wollen verhindern, dass die Gefangenen untereinander Informationen austauschen können. Dazu wollen Sie die Gefangenen so auf die Zellen aufteilen, dass innerhalb jeder Zelle die Gefangenen nicht miteinander kommunizieren können, also kein Insasse eine gemeinsame Sprache mit einem anderen Insassen hat.

Wir formulieren dieses Szenario als Entscheidungsproblem MUTE PRISON. Gegeben sind dabei die Menge P der Gefangenen, die Menge S der Sprachen und die Matrix M. Gesucht ist eine Einteilung der Gefangenen in die Zellen, sodass kein Insasse mit einem anderen kommunizieren kann.

Das NP-schwere Entscheidungsproblem PARTITION INTO TRIANGLES ist wie folgt definiert:

Gegeben: Ungerichteter einfacher Graph $G = (V, E)$ mit $|V| = 3n$.
Frage: Gibt es eine Partitionierung $V_1 \cup V_2 \cup \cdots \cup V_k$ von V, sodass jedes V_i ein Dreieck in G induziert?

Zeigen Sie, dass MUTE PRISON NP-vollständig ist. Geben Sie bei Ihrer Reduktion explizit an, von welchem Problem auf welches reduziert wird!

Hinweis: Beachten Sie, dass ein Dreieck eine Clique der Größe 3 *ist, d.h. alle drei Knoten sind paarweise miteinander verbunden. Was bedeutet das für den Komplementgraphen von G?*

Lösung: Wir zeigen zunächst, dass MUTE PRISON in NP liegt. Für eine gegebene Zellbelegung lässt sich in Zeit $\mathcal{O}(nm)$ nachprüfen, ob in jeder Zelle die Insassen keine gemeinsamen Sprachen haben. Seien i, j und k die drei Insassen einer Zelle. Dann darf für jede Sprache ℓ höchstens einer der drei Matrixeinträge $M[i][\ell]$, $M[j][\ell]$ und $M[k][\ell]$ 1 sein.

Wir zeigen, dass MUTE PRISON NP-schwer ist, indem wir von PARTITION INTO TRIANGLES auf MUTE PRISON reduzieren. Sei eine PARTITION INTO TRIANGLES-Instanz $G = (V, E)$ gegeben. Wir bilden den Komplementgraphen $\overline{G} = (V, \overline{E})$ mit $\overline{E} = (V \times V) \setminus E$, in dem also genau die Knoten verbunden sind, die in G nicht verbunden sind. Daraus konstruieren wir die MUTE PRISON-Instanz wie folgt: Als Menge der Gefangenen wählen wir V und als Menge der Sprachen $\overline{E}$. Für jede Kante $e = (u, v) \in \overline{E}$ setzen wir $M[u][e]$ und $M[v][e]$ auf 1. Alle restlichen Einträge von M sind 0. Diese Konstruktion ist in polynomieller Zeit möglich, da $\overline{G}$ $\mathcal{O}(|V|^2)$ Kanten hat und M somit $\mathcal{O}(|V|^3)$ Einträge hat.

Wir zeigen: Die PARTITION INTO TRIANGLES-Instanz ist genau dann lösbar, wenn die MUTE PRISON-Instanz lösbar ist.

- $\Rightarrow$: Sei $V_1 \cup V_2 \cup \cdots \cup V_k$ eine Lösung der PARTITION INTO TRIANGLES-Instanz. Wir übertragen die Partitionierung der Knoten direkt auf die Gefangenen, d.h. jedes $V_i = \{u, v, w\}$ entspricht einer Dreiergruppe von Gefangenen $P_i = \{i, j, k\}$. Da V_i ein Dreieck in G induziert, sind alle drei Knoten in V_i paarweise durch Kanten in E miteinander verbunden. In $\overline{G}$ sind sie somit nicht verbunden, also gibt es keine Sprache ℓ, für die mindestens zwei der Einträge $M[i][\ell]$, $M[j][\ell]$ und $M[k][\ell]$ 1 sind. Demnach haben die drei Insassen in P_i paarweise keine gemeinsame Sprache.

$\Leftarrow$: Sei $P_1 \cup P_2 \cup \cdots \cup P_k$ eine Lösung der MUTE PRISON-Instanz. Wir übertragen die Einteilung der Gefangenen direkt auf die Knoten, d.h. jede Dreiergruppe $P_i = \{i, j, k\}$ entspricht einer Knotenmenge $V_i = \{u, v, w\}$ in der Partition. Da die Gefangenen in P_i paarweise keine gemeinsame Sprache haben, hat $\overline{G}$ keine Kanten zwischen den entsprechenden Knoten in V_i. In G sind also alle Knoten in V_i paarweise verbunden, d.h. V_i induziert ein Dreieck.

Problem 6: Approximationsalgorithmen 2 + 3 + 2 + 2 = 9 Punkte

Eine *Knotenüberdeckung* (engl. *Vertex Cover*) für einen ungerichteten Graph $G = (V, E)$ ist eine Knotenmenge $V' \subseteq V$, sodass für jede Kante in E mindestens einer der beiden Endpunkte in V' enthalten ist. Das NP-schwere Optimierungsproblem VERTEX COVER fragt nach einer minimalen Knotenüberdeckung und ist wie folgt definiert:

Gegeben: Ungerichteter Graph $G = (V, E)$
Gesucht: Knotenmenge $V' \subseteq V$ mit $|V'|$ minimal, sodass für alle Kanten $(u, v) \in E$ gilt:
$u \in V'$ oder $v \in V'$.

In der Vorlesung haben wir gesehen, dass sich jedes NP-schwere Problem als *ganzzahliges lineares Programm* (engl. *Integer Linear Program*, ILP) darstellen lässt. Für VERTEX COVER können wir also das äquivalente Problem VC-$\mathbb{N}$ betrachten:

Gegeben:
Ungerichteter Graph $G = (V, E)$
Variablen:
Für jeden Knoten $v \in V$ eine Variable x_v
Nebenbedingungen:
(I) Für jeden Knoten $v \in V$: $x_v \in \{0, 1\}$
(II) Für jede Kante $(u, v) \in E$: $x_u + x_v \geq 1$
Zielfunktion:
Minimiere $f_{\mathbb{N}}(G) = \sum_{v \in V} x_v$

Aus einer Lösung für VC-$\mathbb{N}$ lässt sich eine Lösung für VERTEX COVER konstruieren, indem man $V' = \{v \in V \mid x_v = 1\}$ setzt. Der Wert von x_v gibt also an, ob v in der Knotenüberdeckung enthalten ist oder nicht.

Eine gängige Methode, ILPs zu approximieren, ist die *LP-Relaxierung*. Dabei wird die Beschränkung aufgehoben, dass die Variablen ganzzahlige Werte haben müssen. Das dadurch entstehende *lineare Programm* (LP) lässt sich in Polynomialzeit lösen. Im Fall von VERTEX COVER erhalten wir das Problem VC-$\mathbb{R}$. Der einzige Unterschied gegenüber VC-$\mathbb{N}$ ist in Nebenbedingung (I): Die x_v dürfen nun beliebige reelle Zahlen aus dem Intervall $[0, 1]$ sein. Dementsprechend ist auch der Wert der Zielfunktion $f_{\mathbb{R}}(G) = \sum_{v \in V} x_v$ reellwertig.

Sei im Folgenden $OPT_{\mathbb{R}}(G)$ der Wert von $f_{\mathbb{R}}(G)$ für eine optimale Lösung von VC-$\mathbb{R}$ und $OPT_{\mathbb{N}}(G)$ der Wert von $f_{\mathbb{N}}(G)$ einer optimalen Lösung von VC-$\mathbb{N}$.

(a) Zeigen Sie, dass für jeden Graphen G gilt: $OPT_{\mathbb{R}}(G) \leq OPT_{\mathbb{N}}(G)$.

(b) Geben Sie einen Graphen G mit höchstens vier Knoten an, für den $OPT_{\mathbb{R}}(G) < OPT_{\mathbb{N}}(G)$ gilt. Geben Sie für beide Probleme eine optimale Lösung an.

Wir betrachten nun folgenden Algorithmus $\mathcal{A}$, der eine (nicht notwendigerweise optimale) Lösung für VC-$\mathbb{N}$ berechnet:

1. Berechne eine optimale Lösung $X = (x_1, \ldots, x_n)$ mit $x_i \in [0, 1]$ für VC-$\mathbb{R}$.
2. Generiere eine Lösung $X' = (x'_1, \ldots, x'_n)$ mit $x'_i \in \{0, 1\}$ für VC-$\mathbb{N}$, indem wir jedes x'_i wie folgt wählen:

$$x'_i = \begin{cases} 1 & \text{falls } x_i \geq \frac{1}{2} \\ 0 & \text{sonst.} \end{cases}$$

(c) Zeigen Sie, dass Algorithmus $\mathcal{A}$ eine Lösung für VC-$\mathbb{N}$ berechnet, also dass die Nebenbedingungen erfüllt sind.

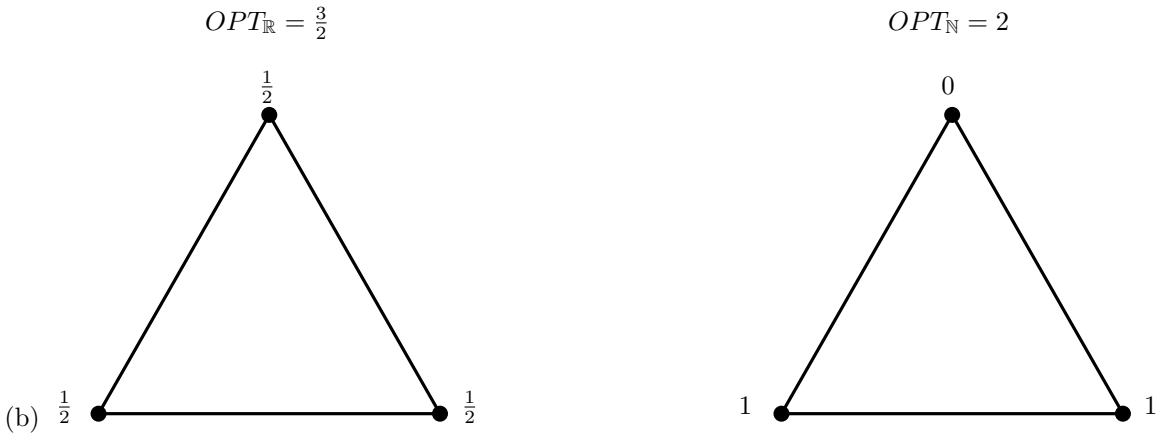
(d) Zeigen Sie, dass $\mathcal{A}$ ein Approximationsalgorithmus für VC-$\mathbb{N}$ mit einer relativen Gütegarantie von 2 ist.

Hinweis: Zeigen Sie dafür zunächst, dass für beliebige Graphen G gilt: $\mathcal{A}(G) \leq 2 \cdot OPT_{\mathbb{R}}(G)$.

Lösung:

(a) Jede Lösung von VC-$\mathbb{N}$ erfüllt die Nebenbedingungen von VC-$\mathbb{R}$ und ist somit auch eine Lösung von VC-$\mathbb{R}$. Es kann also nicht $OPT_{\mathbb{R}}(G) > OPT_{\mathbb{N}}(G)$ gelten, weil $OPT_{\mathbb{R}}(G)$ dann nicht optimal wäre.

(b)



(c) Bedingung (I) ist offensichtlich erfüllt.

Betrachte für Bedingung (II) eine beliebige Kante $(u, v) \in E$. Die Bedingung $x_u + x_v \geq 1$ erzwingt, dass $x_u \geq 1/2$ oder $x_v \geq 1/2$ gelten muss. Sei o.B.d.A. $x_u \geq 1/2$. Dann gilt $x'_u = 1$ und somit ist die Bedingung $x'_u + x'_v \geq 1$ erfüllt.

(d) Es gilt für jedes x'_i:

$$x'_i = \left\{ \begin{array}{ll} 1 & \text{falls } x_i \geq \frac{1}{2} \\ 0 & \text{sonst} \end{array} \right\} \leq \left\{ \begin{array}{ll} 2x_i & \text{falls } x_i \geq \frac{1}{2} \\ 0 & \text{sonst} \end{array} \right\} \leq 2x_i$$

Damit gilt:

$$\mathcal{A}(G) = \sum_{i=1}^{n} x'_i \leq \sum_{i=1}^{n} 2x_i = 2 \cdot OPT_{\mathbb{R}}(G)$$

Wegen Aufgabenteil (a) gilt also $\mathcal{A}(G) \leq 2 \cdot OPT_{\mathbb{N}}(G)$, d.h. $\mathcal{A}$ ist eine 2-Approximation.

Problem 7: Chomsky-Normalform und CYK-Algorithmus $4 + 3 + 1 = 8$ Punkte

(a) Gegeben sei die Grammatik $G_1 = (\Sigma_1, V_1, S, R_1)$ mit Terminalen $\Sigma_1 = \{\mathtt{a}, \mathtt{b}\}$, Nichtterminalen $V_1 = \{S, A, B\}$, Startsymbol S und Produktionen

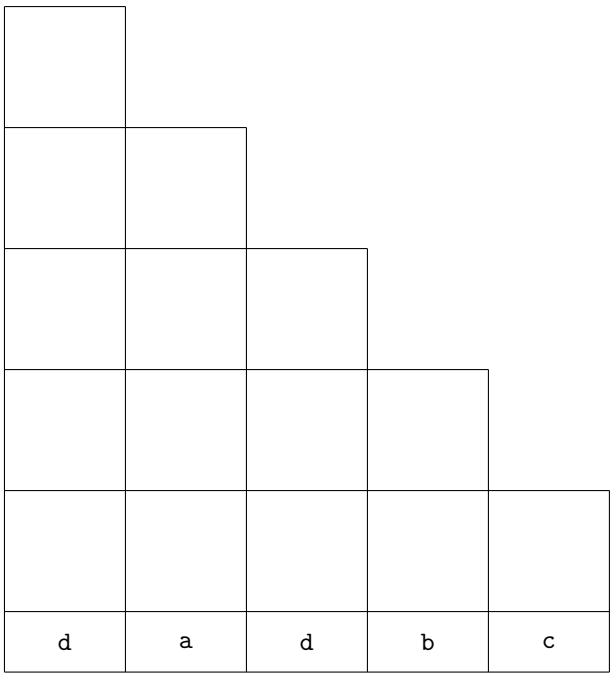
$$\begin{aligned} R_1 = \{ & S \to A\mathtt{b}B \\ & A \to A\mathtt{a} \mid \varepsilon \\ & B \to S \mid \mathtt{b}\} \end{aligned}$$

Transformieren Sie G_1 in Chomsky-Normalform. Geben Sie bei Ihrer Umformung die Zwischenschritte an.

(b) Gegeben sei die Grammatik $G_2 = (\Sigma_2, V_2, S, R_2)$ mit Terminalen $\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$, Nichtterminalen $V_2 = \{S, A, B, C, D\}$, Startsymbol S und Produktionen

$$\begin{aligned} R_2 = \{ & S \to AC \mid BD \\ & A \to BC \mid \mathtt{a} \\ & B \to DA \mid \mathtt{b} \mid \mathtt{c} \\ & C \to CB \mid AD \mid \mathtt{c} \mid \mathtt{d} \\ & D \to BD \mid DC \mid \mathtt{d}\} \end{aligned}$$

Überprüfen Sie mit dem CYK-Algorithmus, ob das Wort dadbc in der Sprache $L(G_2)$ enthalten ist.



(c) Geben Sie einen Syntaxbaum für das Wort dadbc in G_2 an.

Lösung:

(a) 1. Schritt: Alle Regeln enthalten auf der rechten Seite nur Symbole aus V_1 oder nur ein Symbol aus Σ_1.

$$\begin{aligned} R_1 = \{ & S \to AY_bB, \\ & A \to AY_a \mid \varepsilon, \\ & B \to S \mid \texttt{b}, \\ & Y_a \to \texttt{a}, \\ & Y_b \to \texttt{b} \ \} \end{aligned}$$

2. Schritt: Alle rechten Seiten haben Länge ≤ 2.

$$\begin{aligned} R_1 = \{ \ & S \to AC, \\ & A \to AY_a \mid \varepsilon, \\ & B \to S \mid \texttt{b}, \\ & Y_a \to \texttt{a}, \\ & Y_b \to \texttt{b}, \\ & C \to Y_bB \ \} \end{aligned}$$

3. Schritt: Elimination von ε-Produktionen.

$$\begin{aligned} R_1 = \{ \ & S \to AC \mid C, \\ & A \to AY_a \mid Y_a, \\ & B \to S \mid \texttt{b}, \\ & Y_a \to \texttt{a}, \\ & Y_b \to \texttt{b}, \\ & C \to Y_bB \ \} \end{aligned}$$

4. Schritt: Elimination von Kettenregeln.

$$\begin{aligned} R_1 = \{ \ & S \to AC \mid Y_bB, \\ & A \to AY_a \mid \texttt{a}, \\ & B \to AC \mid Y_bB \mid \texttt{b}, \\ & Y_a \to \texttt{a}, \\ & Y_b \to \texttt{b}, \\ & C \to Y_bB \ \} \end{aligned}$$

(b) Vergleiche folgende Tabelle:

S, A, B, D				
A, D	S, C			
S, A, D	S, C	B, C		
B	S, C	C	A	
C, D	A	C, D	B	B, C
`d`	`a`	`d`	`b`	`c`

Das Wort ist also in der Sprache enthalten.

(c) Vergleiche folgenden Syntaxbaum: