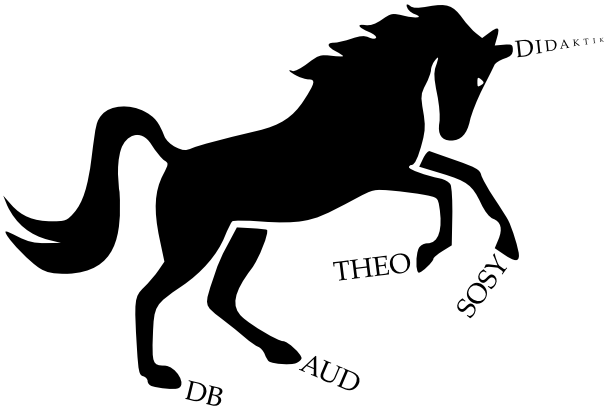
66115
Frühjahr
2016

Theoretische Informatik / Algorithmen (vertieft)

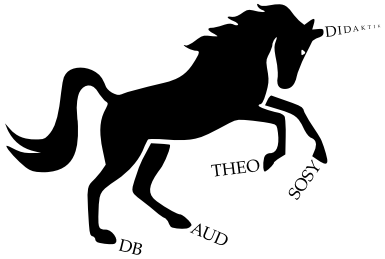
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 1

Aufgabe 1 [Reguläre Sprachen]

(a) Geben Sie einen möglichst einfachen regulären Ausdruck für die Sprache $L_1 = \{ a_1, a_2, \ldots, a_n \mid n \geq 3, a_i \in \{a, b\}$ für alle $i = 1, \ldots, n$ und $a_1 \geq a_n \}$ an.

Lösungsvorschlag

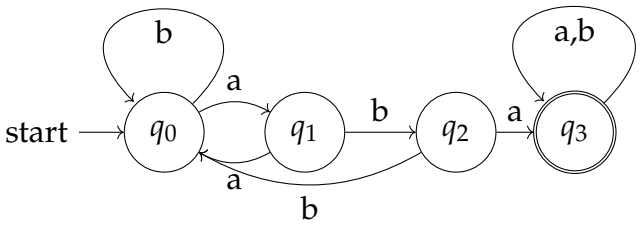
```
((a(a|b)+b)|(b(a|b)+a))
```

(b) Geben Sie einen möglichst einfachen regulären Ausdruck für die Sprache $L_2 = \{ w \in \{a, b\}^* \mid w$ enthält genau ein b und ist von ungerader Länge $\}$ an.

Lösungsvorschlag

```
(aa)*(b|aba)(aa)*
```

(c) Beschreiben Sie die Sprache des folgenden Automaten A_1, möglichst einfach und präzise in ihren eigenen Worten.

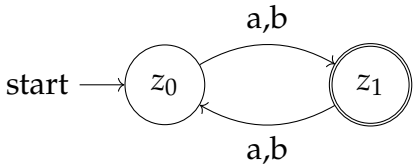


Der Automat auf flaci.com (FLACI: Formale Sprachen, abstrakte Automaten, Compiler und Interpreter) Ein Projekt der Hochschule Zittau/Görlitz und der Pädagogischen Hochschule Schwyz: flaci.com/Arz003ccg

Lösungsvorschlag

Die Sprache enthält das Teilwort *aba*

(d) Betrachten Sie folgenden Automaten A_2:



Der Automat auf flaci.com (FLACI: Formale Sprachen, abstrakte Automaten, Compiler und Interpreter) Ein Projekt der Hochschule Zittau/Görlitz und der Pädagogischen Hochschule Schwyz: flaci.com/Ap9qbkumc

Im Original sind die Zustände mit q_x benannt. Damit wir die Schnittmenge besser bilden können, wird hier z_x verwendet.

Konstruieren Sie einen endlichen Automaten, der die Schnittmenge der Sprachen $L(A_1)$ und $L(A_2)$ akzeptiert.

A_1

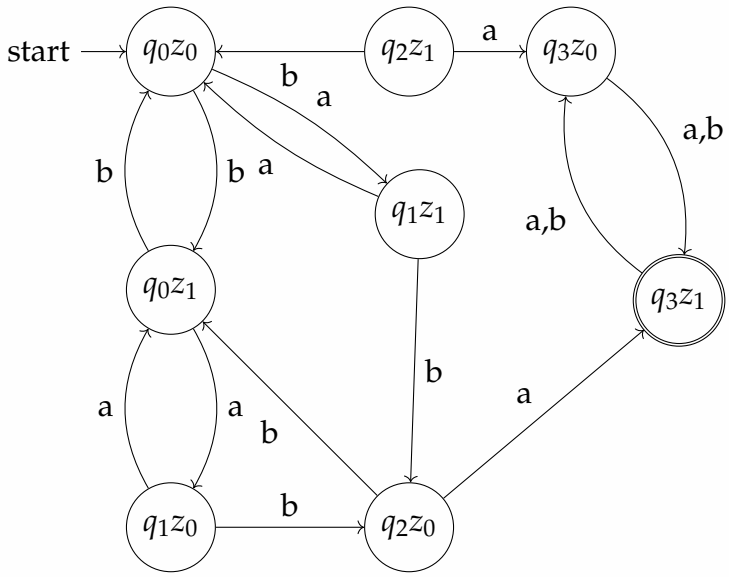
	a	b
q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_0
q_3	q_3	q_3

A_2

	a	b
z_0	z_1	z_1
z_1	z_0	z_0

Neuer Endzustand: q_3z_1

	a	b
q_0z_0	q_1z_1	q_0z_1
q_1z_0	q_0z_1	q_2z_1
q_2z_0	q_3z_1	q_0z_1
q_3z_0	q_3z_1	q_3z_1
q_0z_1	q_1z_0	q_0z_0
q_1z_1	q_0z_0	q_2z_0
q_2z_1	q_3z_0	q_0z_0
q_3z_1	q_3z_0	q_3z_0



Der Automat auf flaci.com (FLACI: Formale Sprachen, abstrakte Automaten, Compiler und Interpreter) Ein Projekt der Hochschule Zittau/Görlitz und der Pädagogischen Hochschule Schwyz: flaci.com/Ar3pc5rh7

Aufgabe 2 [Nonterminale: SA, Terminale: 012]

Betrachten Sie die folgende Grammatik $G = (\{S, A\}, \{0, 1, 2\}, P, S)$ mit

$$\mathrm{P} = \left\{ \begin{array}{l} S \to 0S0 \mid 1S1 \mid 2A2 \mid 0 \mid 1 \mid \varepsilon \\ A \to A2 \end{array} \right\}$$

Der Automat auf flaci.com (FLACI: Formale Sprachen, abstrakte Automaten, Compiler und Interpreter) Ein Projekt der Hochschule Zittau/Görlitz und der Pädagogischen Hochschule Schwyz: flaci.com/Gf6scqja9

Konstruieren Sie für die Grammatik G schrittweise eine äquivalente Grammatik in Chomsky-Normalform. Geben Sie für jeden einzelnen Schritt des Verfahrens das vollständige Zwischenergebnis an und erklären Sie kurz, was in dem Schritt getan wurde.

Lösungsvorschlag

Die Regeln $\{S \to 2A2\}$ und $\{A \to A2\}$ können gelöscht werden, da es keine Regel $\{A \to \varepsilon\}$ oder $\{A \to S\}$ gibt. So erhalten wir:

$$\mathrm{P} = \left\{ S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \varepsilon \right\}$$

(a) **Elimination der ε-Regeln**

— Alle Regeln der Form $A \to \varepsilon$ werden eliminiert. Die Ersetzung von A wird durch ε in allen anderen Regeln vorweggenommen.

falls $S \to \varepsilon \in P$ neuen Startzustand S_1 einführen

$$\mathrm{P} = \left\{ \begin{array}{l} S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid 00 \mid 11 \\ S_1 \to \varepsilon \mid S \end{array} \right\}$$

(b) **Elimination von Kettenregeln**

— Jede Produktion der Form $A \to B$ mit $A, B \in S$ wird als Kettenregel bezeichnet. Diese tragen nicht zur Produktion von Terminalzeichen bei und lassen sich ebenfalls eliminieren.

$\varnothing$ Nichts zu tun

(c) **Separation von Terminalzeichen**

— Jedes Terminalzeichen σ, das in Kombination mit anderen Symbolen auftaucht, wird durch ein neues Nonterminal S_σ ersetzt und die Menge der Produktionen durch die Regel $S_\sigma \to \sigma$ ergänzt. ————

N = Null E = Eins

$$\mathrm{P} = \Big\{$$

$$\begin{aligned} S &\to NSN \mid ESE \mid 0 \mid 1 \mid NN \mid EE \\ S_1 &\to \varepsilon \mid S \\ A &\to AZ \\ N &\to 0 \\ E &\to 1 \end{aligned}$$

$$\Big\}$$

(d) **Elimination von mehrelementigen Nonterminalketten**

— Alle Produktionen der Form $A \to B_1B_2 \ldots B_n$ werden in die Produktionen $A \to A_{n-1}B_n, A_{n-1} \to A_{n-2}B_{n-1}, \ldots, A_2 \to B_1B_2$ zerteilt. Nach der Ersetzung sind alle längeren Nonterminalketten vollständig heruntergebrochen und die Chomsky-Normalform erreicht. ————

$$\mathrm{P} = \Big\{$$

$$\begin{aligned} S &\to NS_N \mid ES_E \mid 0 \mid 1 \mid NN \mid EE \\ S_1 &\to \varepsilon \mid S \\ S_N &\to SN \\ S_E &\to SE \\ N &\to 0 \\ E &\to 1 \end{aligned}$$

$$\Big\}$$

Aufgabe 4 [a n b n]

(a) Geben Sie eine deterministische 2-Band Turingmaschine M an, die die Funktion

$$f_M(a^n) = a^n b^n$$

berechnet. Die Maschine M nimmt somit immer einen String der Form a^n (ein String, der aus n a's für beliebiges $n \in \mathbb{N}$ besteht) als Eingabe und produziert anschließend auf Band 2 als Ausgabe den String $a^n b^n$ (ein String aus n a's gefolgt von n b's).

Beschreiben Sie außerdem die Idee hinter Ihrer Konstruktion.

Lösungsvorschlag

```
name: 66115 2016 03 1 4
init: z0
accept: z2

z0, a,_
z0, a,a, >,>

z0, _,_
z1, _,_, <,-

z1, a,_
z1, a,_, <,-

z1, _,_
z2, _,_, >,-

z2, a,_
z2, a,b, >,>
```

a

z_0	*a*'s auf das 2. Band kopieren
z_1	Zu Beginn der Eingabe auf dem 1. Band, 2. Band bleibt
z_2	Für jedes *a*'s auf dem 1. Band ein *b* auf dem 2. Band erzeugen

[a] `http://turingmachinesimulator.com/shared/lyptczerhe`

(b) Geben Sie die Konfigurationsfolge der Turingmaschine aus (a) für die Eingabe *aa* an.

Lösungsvorschlag

z0 a a, z0 □□

a z0 a, a z0 □

a a z0 □, a a z0 □

a z1 a, a a z1 □

z1 a a □, a a z1 □

z1 □a a, a a z1 □

z2 a a, a a z2 □

a z2 a, a a b z2 □

a a z2 □, a a b b z2 □

Komplexität [k-COL]

Das Problem k-COL ist wie folgt definiert:

k-Col
Gegeben: Ein ungerichteter Graph $G = (V, E)$.
Frage: Kann man jedem Knoten v in V eine Zahl $z(v) \in \{1, \ldots, k\}$ zuordnen, so dass für alle Kanten $(u_1, u_2) \in E$ gilt: $z(u_1) \neq z(u_2)$?

Zeigen Sie, dass man 3-Col in polynomieller Zeit auf 4-Col reduzieren kann. Beschreiben Sie dazu die Reduktion und zeigen Sie anschließend ihre Korrektheit.

Lösungsvorschlag

Zu Zeigen:

$$\text{3-COL} \preceq_p \text{4-Col}$$

also 4-Col ist mindestens so schwer wie 3-COL Eingabeinstanz von 3-COL durch eine Funktion in eine Eingabeinstanz von 4-Col umbauen so, dass jede JA- bzw. NEIN-Instanz von 3-COL eine JA- bzw. NEIN-Instanz von 4-Col ist.

Funktion ergänzt einen beliebigen gegebenen Graphen um einen weiteren Knoten, der mit allen Knoten des ursprünglichen Graphen durch eine Kante verbunden ist.

total ja

in Polynomialzeit berechenbar ja (Begründung: z. B. Adjazenzmatrix $\rightarrow$ neue Spalte)

Korrektheit: ja

Färbe den „neuen“ Knoten mit einer Farbe. Da er mit allen anderen Knoten verbunden ist, bleiben für die übrigen Knoten nur drei Farben.

Aufgabe 6 [1 45 8 53 9 2 17 10]

Sortieren Sie die Werte

1 45 8 53 9 2 17 10

mit Quicksort.

Lösungsvorschlag

Sortieralgorithmus nach Saake

```
 1   45  8   53  9   2   17  10  zerlege
 1   45  8   53* 9   2   17  10  markiere (i 3)
 1   45  8  >53  9   2   17  10< vertausche (i 3<>7)
>1<  45  8   10  9   2   17  53  vertausche (i 0<>0)
 1  >45< 8   10  9   2   17  53  vertausche (i 1<>1)
 1   45 >8<  10  9   2   17  53  vertausche (i 2<>2)
 1   45  8  >10< 9   2   17  53  vertausche (i 3<>3)
 1   45  8   10 >9<  2   17  53  vertausche (i 4<>4)
 1   45  8   10  9  >2<  17  53  vertausche (i 5<>5)
```

```
 1   45  8   10  9   2  >17< 53  vertausche (i 6<>6)
 1   45  8   10  9   2   17 >53< vertausche (i 7<>7)
 1   45  8   10  9   2   17      zerlege
 1   45  8   10* 9   2   17      markiere (i 3)
 1   45  8  >10  9   2   17<     vertausche (i 3<>6)
>1<  45  8   17  9   2   10      vertausche (i 0<>0)
 1  >45  8<  17  9   2   10      vertausche (i 1<>2)
 1   8  >45  17  9<  2   10      vertausche (i 2<>4)
 1   8   9  >17  45  2<  10      vertausche (i 3<>5)
 1   8   9   2  >45  17  10<     vertausche (i 4<>6)
 1   8   9   2                   zerlege
 1   8*  9   2                   markiere (i 1)
 1  >8   9   2<                  vertausche (i 1<>3)
>1<  2   9   8                   vertausche (i 0<>0)
 1  >2<  9   8                   vertausche (i 1<>1)
 1   2  >9   8<                  vertausche (i 2<>3)
 1   2                           zerlege
 1*  2                           markiere (i 0)
>1   2<                          vertausche (i 0<>1)
>2   1<                          vertausche (i 0<>1)
                    17  45      zerlege
                    17* 45      markiere (i 5)
                   >17  45<     vertausche (i 5<>6)
                   >45  17<     vertausche (i 5<>6)
```

Sortieralgorithmus nach Horare

```
 1   45  8   53  9   2   17  10  zerlege
 1   45  8   53* 9   2   17  10  markiere (i 3)
 1   45  8  >53  9   2   17  10< vertausche (i 3<>7)
 1   45  8   10  9   2   17      zerlege
 1   45  8   10* 9   2   17      markiere (i 3)
 1  >45  8   10  9   2<  17      vertausche (i 1<>5)
 1   2   8  >10  9<  45  17      vertausche (i 3<>4)
 1   2   8   9                   zerlege
 1   2*  8   9                   markiere (i 1)
 1   2                           zerlege
 1*  2                           markiere (i 0)
         8   9                   zerlege
         8*  9                   markiere (i 2)
                 10  45  17      zerlege
                 10  45* 17      markiere (i 5)
                 10 >45  17<     vertausche (i 5<>6)
                 10  17          zerlege
                 10* 17          markiere (i 4)
```

Thema Nr. 2

Verständnis formale Sprachen [Verständnis]

Beantworten Sie kurz, präzise und mit Begründung folgende Fragen: (Die Begründungen müssen keine formellen mathematischen Beweise sein).

(a) Welche Möglichkeiten gibt es, eine formale Sprache vom Typ 3 zu definieren?

Lösungsvorschlag

- reguläre Grammatik
- endlicher nichtdeterministische und deterministischer Automat
- regulärer Ausdruck

(b) Was ist die Komplexität des Wortproblems für Typ-3 Sprachen und wieso ist das so?

Lösungsvorschlag

P, CYK-Algorithmus löst es ein Polynomialzeit.

(c) Sind Syntaxbäume zu einer Grammatik immer eindeutig? Falls nicht, geben Sie ein Gegenbeispiel.

Lösungsvorschlag

Nein. Syntaxbäume zu einer Grammtik sind nicht immer eindeutig.

Gegenbeispiel

$G = (\{S, A, B\}, \{a\}, P, S)$

$$\mathrm{P} = \Big\{ \begin{array}{l} S \to AA \\ S \to BB \\ A \to a \\ B \to a \end{array} \Big\}$$

$S \vdash AA \vdash aA \vdash aa$

$S \vdash BB \vdash aB \vdash aa$

(d) Wie kann man die Äquivalenz zweier Typ-3 Sprachen nachweisen?

Lösungsvorschlag

Myhill-Nerode-Äquivalenz. Wir können von den auf Äquivalenz zu überprüfenden Sprachen jeweils einen minimalen endlichen Automaten bilden. Sind diese entstanden zwei Automaten äquivalent so sind auch die Sprachen äquivalent.

(e) Wie kann man das Wortproblem für das Komplement einer Typ-3 Sprache lösen?

Lösungsvorschlag

Da das Komplement einer regulären Sprache wieder eine reguläre Sprache ergibt, kann das Wortproblem beim Komplement durch einen deterministisch endlichen Automaten gelöst werden. Tausche akzeptierende mit nicht akzeptierenden Zuständen des zugehörigen Automaten.

Alternativ: Ergebnis des CYK-Algorithmus invertieren oder CYK auf Komplement, da reguläre Sprachen unter dem Komplement abgeschlossen sind.

(f) Weshalb gilt das Pumping-Lemma für Typ 3 Sprachen?

Lösungsvorschlag

endliche Anzahl von Zuständen n im Automaten $\rightarrow$ für Wörter $|\omega| > n$ muss Zyklus vorhanden sein.

(g) Ist der Nachweis, dass das Typ-3 Pumping-Lemma für eine gegebene Sprache gilt, ausreichend, um zu zeigen, dass die Sprache vom Typ 3 ist? Falls nicht, geben Sie ein Gegenbeispiel, mit Begründung.

Lösungsvorschlag

Nein:

Die Sprache $L = \{a^m b^n c^n \mid m, n \geq 1\} \cup \{b^m c^n \mid m, n \geq 0\}$ ist nicht regulär. Allerdings erfüllt L die Eigenschaften des Pumping-Lemmas, denn jedes Wort $z \in L$ lässt sich so zerlegen $z = uvw$, dass auch für alle $i \geq 0$ $uv^i w \in L$. Dazu kann v einfach als erster Buchstabe gewählt werden. Dieser ist entweder ein a, die Anzahl von führenden as ist beliebig. Oder er ist ein b oder c, ohne führende as ist aber die Anzahl von führenden bs oder cs beliebig. [a]

[a] `https://de.wikipedia.org/wiki/Pumping-Lemma`

(h) Geben Sie ein Beispiel, an dem deutlich wird, dass deterministische und nichtdeterministische Typ-2 Sprachen unterschiedlich sind.

Lösungsvorschlag

Deterministisch Kontextfrei $L = \{\, 0^n 1^n \mid n \geq 0 \,\}$

Nichtdeterministisch Kontextfrei $L = \{\, \omega\omega^R \mid \omega \geq \{0,1\}^* \,\}$ (R steht für rückwärts)

[a]

[a] `https://docplayer.org/19566652-Einfuehrung-in-die-theoretische-informatik.html`

(i) Worin macht sich der Unterschied zwischen Typ 0 und 1 bemerkbar, wenn man Turingmaschinen benutzt, um das Wortproblem vom Typ 0 oder 1 zu lösen. Warum ist das so?

Lösungsvorschlag

Typ 0: semi-entscheidbar, Typ 1: entscheidbar

Typ 0: Unendlichkeit des Band kann die unendlich lange Berechenbarkeit zustande kommen.

Typ 1: Linear beschränkte Turingmaschine endlich, dadurch Anzahl an Kombiniation

Typ 1 Sprachen sind monoton wachsend.

Da Typ 1 nur Wörter verlängert, kann daher in Polynomialzeit überprüft werden, ob das Wort in der Sprache liegt, indem die Regeln angewendet werden, bis das Wortende erreicht ist.

Verständnis Berechenbarkeitstheorie [Verständnis Berechenbarkeitstheorie]

Beantworten Sie kurz, präzise und mit Begründung folgende Fragen: (Die Begründungen müssen keine formellen mathematischen Beweise sein)

(a) Warum genügt es, sich auf Funktionen zu beschränken, die natürliche Zahlen auf natürliche Zahlen abbilden, wenn man untersuchen will, was heutige Computer im Prinzip berechnen können?

Lösungsvorschlag

Jede WHILE-berechenbare Funktion ist turing-berechenbar. Jede turing-berechenbare Funktion ist WHILE-berechenbar. Jede nichtdeterministische Turing-Maschine kann durch eine deterministische Turing-Maschine simuliert werden.

(b) Was besagt die Church-Turing-These? Könnte man sie beweisen oder widerlegen?

Lösungsvorschlag

Die Church-Turing-These besagt, dass die Klasse der turing-berechenbaren Funktionen mit der Klasse der intuitiv berechenbaren Funktionen übereinstimmt. Die These kann nicht bewiesen oder widerlegt werden, weil es sich bei dem Begriff *„intuitiv berechenbare Funktion“* um keinen mathematisch exakt definitierten Begriff handelt. Würde man ihn genau definierten, würde ein konkretes Berechnungsmodell festlegt werden, was der Kernaussage dieses Begriffes widersprechen würde.

(c) Für reelle Zahlen, wie z. B. π, lässt sich die Dezimaldarstellung durch entsprechende Programme beliebig genau approximieren. Gilt das für alle reellen Zahlen, ðlässt sich für jede reelle Zahl die Dezimaldarstellung mit entsprechenden Programmen beliebig genau approximieren?

Lösungsvorschlag

Ja mit einer Berechnungsvorschrift, ja solange der speicherplatz reicht, z. B. mittels Intervallschachtelung.

(d) Was ist für die Berechnungskraft der wesentliche Unterschied zwischen While-Berechenbarkeit und Loop-Berechenbarkeit.

Lösungsvorschlag

Alle LOOP-Programme sind mathematisch betrachtet totale Funktionen, ðsie terminieren immer. WHILE-Programme hingegen partiellen Funktionen, die nicht für alle Eingabekombinationen terminieren.

(e) Die Ackermannfunktion ist ein Beispiel einer totalen Funktion, die While-berechenbar, aber nicht Loop-berechenbar ist. Sie verallgemeinert die Idee, dass Multiplikation die wiederholte Addition ist, Exponentiation die wiederholte Multiplikation, Hyperexponentiation die wiederholte Exponentiation usw. Die Stufe dieser hyper-hyper ... Exponentiation ist ein Parameter der Ackermannfunktion. Generieren Sie aus dieser Idee ein Argument, das illustriert, warum die Ackermannfunktion nicht Loop-berechenbar ist.

Lösungsvorschlag

Jedes LOOP-Programm benötigt zur Berechnung der Funktion $x \uparrow^n y$ mindestens $n + 2$ Schleifen.

Gäbe es ein LOOP-Programm, das $ack(n, m)$ tatsächlich für beliebige Werte von n berechnet, so müsste dieses — im Widerspruch zum endlichen Aufbau eines Loop-Programms — unendlich viele Schleifenkonstrukte enthalten.

Hyperexponentiation

- $x \uparrow^{-1} y = x + y$
- $x \uparrow^0 y = x \cdot y$
- $x \uparrow^1 y = x^y$
- $x \uparrow^2 y = x^{y^y}$

(f) Geben Sie ein Beispiel einer Menge an, die abzählbar, aber nicht rekursiv aufzählbar ist, und begründen Sie es.

Lösungsvorschlag

- Die Menge der Primzahlen. Die Primzahl sind unendlich groß und könnten abgezählt werden. Bei Primzahlen gibt es keine Berechnungsvorschrift, die z. b. die 7. Primzahl berechnet.
- Die Menge der Turningmaschine
- Diagonalsprache
- Das Komplement des Halteproblems. Die dem Halteproblem zugrundeliegende Lösungsmenge ist rekursiv aufzählbar. Wäre das Komplement des Hal-

teproblems auch rekursiv aufzählbar, dann wäre das Halteproblem entscheidbar, was es aber nicht ist. [a]

[a] https://www.youtube.com/watch?v=om4ZTOeQuD0

(g) Wie ist der Zusammenhang zwischen rekursiv aufzählbar und semi-entscheidbar?

Lösungsvorschlag

Die beiden Begriffe sind äquivalent. [a]

[a] https://www.youtube.com/watch?v=om4ZTOeQuD0

Verständnis Komplexitätstheorie [Verständnis]

Beantworten Sie kurz, präzise und mit Begründung folgende Fragen: (Die Begründungen müssen keine formellen mathematischen Beweise sein)

(a) In der O-Notation insbesondere für die Zeitkomplexität von Algorithmen lässt man i. A. konstante Faktoren oder kleinere Terme weg. Z. B. schreibt man anstelle $\mathcal{O}(3n^2 + 5)$ einfach nur $\mathcal{O}(n^2)$. Warum macht man das so?

Lösungsvorschlag

Das Wachstum im Unendlich ist bestimmt durch den größten Exponenten. Konstante falle bei einer asymptotischen. Analyse weg. nicht wesentlich schneller

(b) Was ist die typische Vorgehensweise, wenn man für ein neues Problem die NP-Vollständigkeit untersuchen will?

Lösungsvorschlag

Die alten Probleme werden reduziert. Das neue Problem ist größer als die alten Probleme. Das Problem muss in NP liegen.

(i) Problem *in* NP durch Angabe eines nichtdeterministischen Algorithmus in Polynomialzeit

(ii) Problem NP-schwer via Reduktion: $L_{\text{NP-vollständig}} \leq L_{\text{neues Problem}}$

(c) Was könnte man tun, um $P = NP$ zu beweisen?

Lösungsvorschlag

Es würde genügen, zu einem einzigen NP-Problem beweisen, dass es in P liegt.

Zu einem Problem einen deterministen Turingmaschin finden, die es in polynomineller Zeit löst.

(d) Sind NP-vollständige Problem mit Loop-Programmen lösbar? (Antwort mit Begründung!)

Lösungsvorschlag

nicht lösbar mit Loop-Programmen. Begründung ähnlich wie bei der Ackermann-Funktion. z. B. Passwort. Zu Passwort beliebig bräuchte man beliebige for schleifen, was dem endlichen Anzahl an Loop-Schleifen widerspricht.

(e) Wie zeigt man aus der NP-Härte des SAT-Problems die NP-Härte des 3SAT-Problems? (3SAT ist ein SAT-Problem wobei alle Klauseln maximal 3 Literale haben.)

Lösungsvorschlag

in den Lösungen enthalten

4. Hashing [Hashing mit verketteten Listen und offener Adressierung]

Betrachte eine Hashtabelle der Größe $m = 10$.

(a) Welche der folgenden Hashfunktionen ist für Hashing mit verketteten Listen am besten geeignet? Begründen Sie Ihre Wahl!

(i) $h_1(x) = (4x + 3) \mod m$

Lösungsvorschlag

1 $h_1(1) = (4 \cdot 1 + 3) \mod 10 = 7$
2 $h_1(2) = (4 \cdot 2 + 3) \mod 10 = 1$
3 $h_1(3) = (4 \cdot 3 + 3) \mod 10 = 5$
4 $h_1(4) = (4 \cdot 4 + 3) \mod 10 = 9$
5 $h_1(5) = (4 \cdot 5 + 3) \mod 10 = 3$
6 $h_1(6) = (4 \cdot 6 + 3) \mod 10 = 7$
7 $h_1(7) = (4 \cdot 7 + 3) \mod 10 = 1$
8 $h_1(8) = (4 \cdot 8 + 3) \mod 10 = 5$
9 $h_1(9) = (4 \cdot 9 + 3) \mod 10 = 9$
10 $h_1(10) = (4 \cdot 10 + 3) \mod 10 = 3$

(ii) $h_2(x) = (3x + 3) \mod m$

Lösungsvorschlag

1 $h_2(1) = (3 \cdot 1 + 3) \mod 10 = 6$
2 $h_2(2) = (3 \cdot 2 + 3) \mod 10 = 9$
3 $h_2(3) = (3 \cdot 3 + 3) \mod 10 = 2$
4 $h_2(4) = (3 \cdot 4 + 3) \mod 10 = 5$
5 $h_2(5) = (3 \cdot 5 + 3) \mod 10 = 8$
6 $h_2(6) = (3 \cdot 6 + 3) \mod 10 = 1$
7 $h_2(7) = (3 \cdot 7 + 3) \mod 10 = 4$
8 $h_2(8) = (3 \cdot 8 + 3) \mod 10 = 7$

9 $h_2(9) = (3 \cdot 9 + 3) \mod 10 = 0$
10 $h_2(10) = (3 \cdot 10 + 3) \mod 10 = 3$

Lösungsvorschlag

Damit die verketteten Listen möglichst klein bleiben, ist eine möglichst gleichmäßige Verteilung der Schlüssel in die Buckets anzustreben. h_2 ist dafür besser geeignet als h_1, da h_2 in alle Buckets Schlüssel ablegt, h_1 jedoch nur in Buckets mit ungerader Zahl.

(b) Welche der folgenden Hashfunktionen ist für Hashing mit offener Adressierung am besten geeignet? Begründen Sie Ihre Wahl!

(i) $h_1(x, i) = (7 \cdot x + i \cdot m) \mod m$
(ii) $h_2(x, i) = (7 \cdot x + i \cdot (m - 1)) \mod m$

Lösungsvorschlag

$h_2(x, i)$ ist besser geeignet. h_1 sondiert immer im selben Bucket, $(i \cdot m) \mod m$ heben sich gegenseitig auf, zum Beispiel ergibt:

- $h_1(3, 0) = (7 \cdot 3 + 0 \cdot 10) \mod 10 = 1$
- $h_1(3, 1) = (7 \cdot 3 + 1 \cdot 10) \mod 10 = 1$
- $h_1(3, 2) = (7 \cdot 3 + 2 \cdot 10) \mod 10 = 1$

Während hingegen h_2 verschiedene Buckets belegt.

- $h_2(3, 0) = (7 \cdot 3 + 0 \cdot 9) \mod 10 = 1$
- $h_2(3, 1) = (7 \cdot 3 + 1 \cdot 9) \mod 10 = 0$
- $h_2(3, 2) = (7 \cdot 3 + 2 \cdot 9) \mod 10 = 9$

Dijkstra Algorithmus [Karlsruhe nach Kassel]

(a) Berechnen Sie für folgenden Graphen den kürzesten Weg von Karlsruhe nach Kassel und dokumentieren Sie den Berechnungsweg:

Verwendete Abkürzungen:

A Augsburg
EF Erfurt
F Frankfurt
KA Karlsruhe
KS Kassel
M München

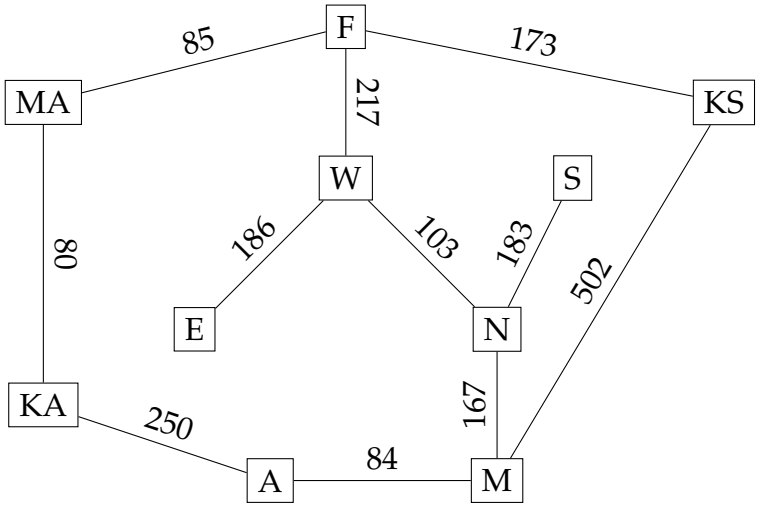
MA Mannheim

N Nürnberg

S Stuttgart

WÜ Würzburg

Zahl = Zahl in Kilometern



Lösungsvorschlag

Nr.	besucht	A	E	F	KA	KS	M	MA	N	S	W
0		∞	∞	∞	0	∞	∞	∞	∞	∞	∞
1	KA	**250**	∞	∞	**0**	∞	∞	80	∞	∞	∞
2	MA	|	∞	165	|	∞	∞	**80**	∞	∞	∞
3	F	|	∞	**165**	|	338	∞	|	∞	∞	382
4	A	|	∞	|	|	338	334	|	∞	∞	382
5	M	|	∞	|	|	338	**334**	|	501	∞	382
6	KS	|	∞	|	|	**338**	|	|	501	∞	382
7	W	|	568	|	|	|	|	|	485	∞	**382**
8	N	|	568	|	|	|	|	|	**485**	668	|
9	E	|	**568**	|	|	|	|	|	|	668	|
10	S	|	|	|	|	|	|	|	|	**668**	|

nach	Entfernung	Reihenfolge	Pfad
KA $\rightarrow$ A	250	0	KA $\rightarrow$ A
KA $\rightarrow$ E	568	9	KA $\rightarrow$ MA $\rightarrow$ F $\rightarrow$ W $\rightarrow$ E
KA $\rightarrow$ F	165	3	KA $\rightarrow$ MA $\rightarrow$ F
KA $\rightarrow$ KA	0	1	
KA $\rightarrow$ KS	338	6	KA $\rightarrow$ MA $\rightarrow$ F $\rightarrow$ KS
KA $\rightarrow$ M	334	5	KA $\rightarrow$ A $\rightarrow$ M
KA $\rightarrow$ MA	80	2	KA $\rightarrow$ MA
KA $\rightarrow$ N	485	8	KA $\rightarrow$ MA $\rightarrow$ F $\rightarrow$ W $\rightarrow$ N
KA $\rightarrow$ S	668	10	KA $\rightarrow$ MA $\rightarrow$ F $\rightarrow$ W $\rightarrow$ N $\rightarrow$ S
KA $\rightarrow$ W	382	7	KA $\rightarrow$ MA $\rightarrow$ F $\rightarrow$ W

(b) Könnte man den Dijkstra Algorithmus auch benutzen, um das Travelling-Salesman Problem zu lösen?

Verständnis Suchbäume [Vergleich Suchbäume]

Wofür eignen sich die folgenden Baum-Datenstrukturen im Vergleich zu den anderen angeführten Baumstrukturen am besten, und warum. Sprechen Sie auch die Komplexität der wesentlichen Operationen und die Art der Speicherung an.

(a) Rot-Schwarz-Baum

Lösungsvorschlag

Einfügen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall) [a]

[a] tutorialspoint.com

(b) AVL-Baum

Lösungsvorschlag

Einfügen (Zeitkomplexität)
$\mathcal{O}(\log_2 n)$ (im Durchschnitt)
$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log_2 n)$ (im Durchschnitt)
$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(\log_2 n)$ (im Durchschnitt)
$\mathcal{O}(\log_2 n)$ (im schlechtesten Fall) [a]

[a] tutorialspoint.com

(c) Binärer-Heap

Lösungsvorschlag

Verwendungszweck zum effizienten Sortieren von Elementen. [a]

Einfügen (Zeitkomplexität)
$\mathcal{O}(1)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(n)$ (im Durchschnitt)
$\mathcal{O}(n)$ (im schlechtesten Fall) [b]

[a] deut. Wikipedia
[b] engl. Wikipedia

(d) B-Baum

Lösungsvorschlag

Einfügen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Löschen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall)

Suchen (Zeitkomplexität)
$\mathcal{O}(\log n)$ (im Durchschnitt)
$\mathcal{O}(\log n)$ (im schlechtesten Fall) [a]

[a] tutorialspoint.com

(e) R-Baum

Verwendungszweck Ein R-Baum erlaubt die schnelle Suche in mehrdimensionalen ausgedehnten Objekten. [a]

Suchen (Zeitkomplexität)

$\mathcal{O}(\log_M n)$ (im Durchschnitt) [b]
$\mathcal{O}(n)$ (im schlechtesten Fall) [c]

[a] deut. Wikipedia
[b] eng. Wikipedia
[c] Simon Fraser University, Burnaby, Kanada