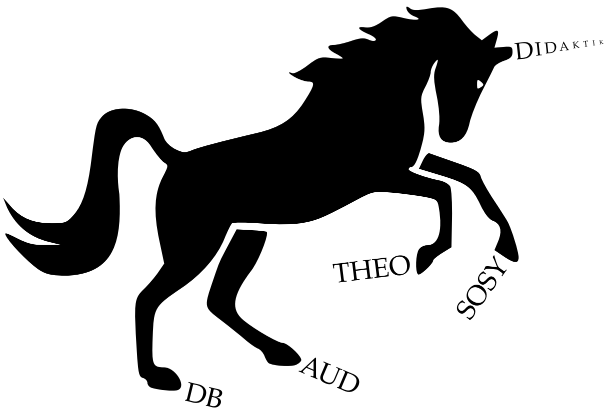
66115
Frühjahr
2017

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 (Graphalgorithmen) [Bayerischee Autobahnen]

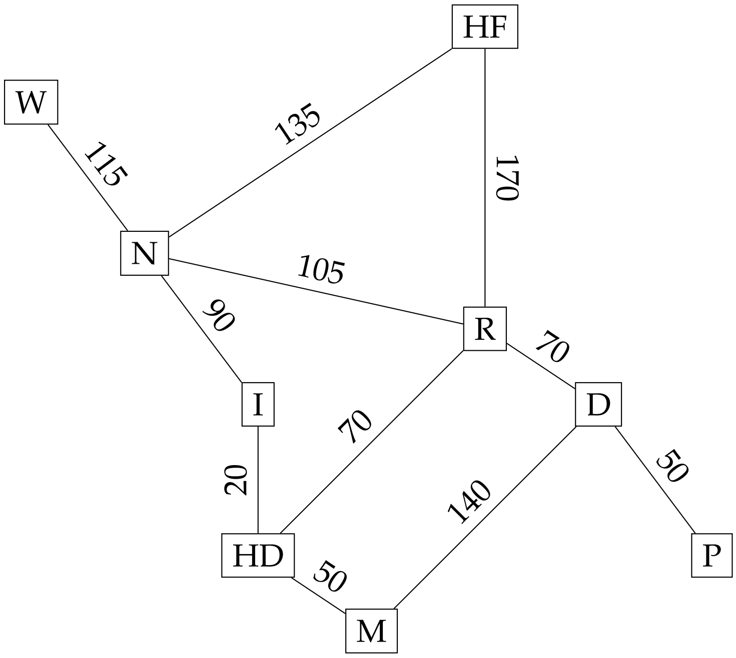
Die folgende Abbildung zeigt die wichtigsten bayerischen Autobahnen zusammen mit einigen anliegenden Orten und die Entfernungen zwischen diesen.

Entfernungstabelle

von	nach	km
Würzburg	Nürnberg	115
Nürnberg	Regensburg	105
Regensburg	AK Deggendorf	70
AK Deggendorf	Passau	50
Hof	Nürnberg	135
Nürnberg	Ingolstadt	90
Ingolstadt	AD Holledau	20
AD Holledau	München	50
München	AK Deggendorf	140
Hof	Regensburg	170
Regensburg	AD Holledau	70

Abkürzungen

D	Deggendorf
HF	Hof
HD	Holledau
I	Ingolstadt
M	München
N	Nürnberg
P	Passau
R	Regensburg
W	Würzburg



(a) Bestimmen Sie mit dem Algorithmus von *Dijkstra* den kürzesten Weg von Ingolstadt zu allen anderen Orten. Verwenden Sie zur Lösung eine Tabelle gemäß folgendem Muster und markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Ort. Setzen Sie für die noch zu bearbeitenden Orte eine Prioritätswarteschlange ein, ðbei gleicher Entfernung wird der ältere Knoten gewählt.

Lösungsvorschlag

Nr.	besucht	D	HD	HF	I	M	N	P	R	W
0		∞	∞	∞	0	∞	∞	∞	∞	∞
1	I	∞	20	∞	**0**	∞	90	∞	∞	∞
2	HD	∞	**20**	∞	|	70	90	∞	90	∞
3	M	210	|	∞	|	**70**	90	∞	90	∞
4	N	210	|	225	|	|	**90**	∞	90	205
5	R	160	|	225	|	|	|	∞	**90**	205
6	D	**160**	|	225	|	|	|	210	|	205
7	W	|	|	225	|	|	|	210	|	**205**
8	P	|	|	225	|	|	|	**210**	|	|
9	HF	|	|	**225**	|	|	|	|	|	|

(b) Die bayerische Landesregierung hat beschlossen, die eben betrachteten Orte mit einem breitbandigen Glasfaser-Backbone entlang der Autobahnen zu verbinden. Dabei soll aus Kostengründen so wenig Glasfaser wie möglich verlegt werden. Identifizieren Sie mit dem Algorithmus von Kruskal diejenigen Strecken, entlang welcher Glasfaser verlegt werden muss. Geben Sie die Ortspaare (Autobahnsegmente) in der Reihenfolge an, in der Sie sie in Ihre Verkabelungsliste aufnehmen.

Lösungsvorschlag

Diese Aufgabe hat noch keine Lösung. Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.

(c) Um Touristen den Besuch aller Orte so zu ermöglichen, dass sie dabei jeden Autobahnabschnitt genau einmal befahren müssen, bedarf es zumindest eines sogenannten offenen Eulerzugs. Zwischen welchen zwei Orten würden Sie eine Autobahn bauen, damit das bayerische Autobahnnetz mindestens einen Euler-Pfad enthält?

> **Exkurs: offener Eulerzug**
>
> Ein offener Eulerzug ist gegeben, wenn Start- und Endknoten nicht gleich sein müssen, wenn also statt eines Zyklus lediglich eine Kantenfolge verlangt wird, welche jede Kante des Graphen genau einmal enthält. Ein bekanntes Beispiel ist das „Haus vom Nikolaus“.

Lösungsvorschlag

Zwischen Deggendorf und Würzburg

P $\rightarrow$ D $\rightarrow$ R $\rightarrow$ N $\rightarrow$ **W** $\rightarrow$ **D** $\rightarrow$ M $\rightarrow$ HD $\rightarrow$ R $\rightarrow$ HF $\rightarrow$ N $\rightarrow$ I $\rightarrow$ HD

Aufgabe 2 [Top-Level-Domains (TLD)]

In dieser Aufgabe sei vereinfachend angenommen, dass sich Top-Level-Domains (TLD) ausschließlich aus zwei oder drei der 26 Kleinbuchstaben des deutschen Alphabets ohne Umlaute zusammensetzen. Im Folgenden sollen TLDs lexikographisch aufsteigend sortiert werden, ðeine TLD (s_1, s_2) mit zwei Buchstaben (z. B. „co“ für Kolumbien) wird also vor einer TLD (t_1, t_2, t_3) der Länge drei (z. B. „com“) einsortiert, wenn $s_1 < t_1 \vee (s_1 = t_1 \wedge s_2 \leq t_2)$ gilt.

(a) Sortieren Sie zunächst die Reihung [„de“, „com“, „uk“, „org“, „co“, „net“, „fr“, „ee“] schrittweise unter Verwendung des Radix-Sortierverfahrens (Bucketsort). Erstellen Sie dazu eine Tabelle wie das folgende Muster und tragen Sie dabei in das Feld „Stelle“ die Position des Buchstabens ein, nach dem im jeweiligen Durchgang sortiert wird (das Zeichen am TLD-Anfang habe dabei die „Stelle“ 1).

> **Exkurs: Alphabet**
>
> abcdefghijklmnopqrstuvwxyz

Lösungsvorschlag

Stelle	Reihung							
	de_	com	uk_	org	co_	net	fr_	ee_
3	de_	uk_	co_	fr_	ee_	org	com	net
2	de_	ee_	net	uk_	co_	com	fr_	org
1	co_	com	de_	ee_	fr_	net	org	uk_

(b) Sortieren Sie nun die gleiche Reihung wieder schrittweise, diesmal jedoch unter Verwendung des Mergesort-Verfahrens (Sortieren durch Mischen). Erstellen Sie dazu eine Tabelle wie das folgende Muster und vermerken Sie in der ersten Spalte jeweils welche Operation durchgeführt wurde: Wenn Sie die Reihung geteilt haben, schreiben Sie in die linke Spalte ein T und markieren Sie die Stelle, an der Sie die Reihung geteilt haben, mit einem senkrechten Strich „|“. Wenn Sie zwei Teilreihungen durch Mischen zusammengeführt haben, schreiben Sie ein M in die linke Spalte und unterstreichen Sie die zusammengemischten Einträge. Beginnen Sie mit dem rekursiven Abstieg immer in der linken Hälfte einer (Teil-)Reihung.

```
O | Reihung
T | de_   com   uk_   org | co_   net   fr_   ee_
T | de_   com | uk_   org
T | de_ | com
M | com   de_
T |             uk_ | org
M |             org   uk_
M | com   de_   org   uk_
T |                         co_   net | fr_   ee_
T |                         co_ | net
M |                         co_   net
T |                                     fr_ | ee_
T |                                     ee_ | fr_
M |                         co_   ee_   fr_   net
M | co_   com   de_   ee_   fr_   net   org   uk_
```

(c) Implementieren Sie das Sortierverfahren Quicksort für String-TLDs in einer gängigen Programmiersprache Ihrer Wahl. Ihr Programm (Ihre Methode) wird mit drei Parametern gestartet: dem String-Array mit den zu sortierenden TLDs selbst sowie jeweils der Position des ersten und des letzten zu sortierenden Eintrags im Array.

Lösungsvorschlag

```java
public class Quicksort {

  public static void swap(String[] array, int index1, int index2) {
    String tmp = array[index1];
    array[index1] = array[index2];
    array[index2] = tmp;
  }
```

```java
  public static int partition(String[] array, int first, int last) {
    int pivotIndex = (last + first) / 2;
    String pivotValue = array[pivotIndex];
    int pivotIndexFinal = first;
    swap(array, pivotIndex, last);
    for (int i = first; i < last; i++) {
      if (array[i].compareTo(pivotValue) < 0) {
        swap(array, i, pivotIndexFinal);
        pivotIndexFinal++;
      }
    }
    swap(array, last, pivotIndexFinal);
    return pivotIndexFinal;
  }

  public static void sort(String[] array, int first, int last) {
    if (first < last) {
      int pivotIndex = partition(array, first, last);
      sort(array, first, pivotIndex - 1);
      sort(array, pivotIndex + 1, last);
    }
  }

  public static void main(String[] args) {
    String[] array = new String[] { "de", "com", "uk", "org", "co", "net", "fr",
↪   "ee" };
    sort(array, 0, array.length - 1);
    for (int i = 0; i < array.length; i++) {
      System.out.println(array[i]);
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2017/fruehjahr/Quicksort.java`

Aufgaben 3 [Fibonacci]

Gegeben seien die folgenden Formeln zur Berechnung der *ersten* Fibonacci-Zahlen:

$$\mathrm{fib}_n = \begin{cases} 1 & \text{falls } n \leq 2 \\ \mathrm{fib}_{n-1} + \mathrm{fib}_{n-2} & \text{sonst} \end{cases}$$

sowie der Partialsumme der Fibonacci-Quadrate:

$$\mathrm{sos}_n = \begin{cases} \mathrm{fib}_n & \text{falls } n = 1 \\ \mathrm{fib}_n^2 + \mathrm{sos}_{n-1} & \text{sonst} \end{cases}$$

Sie dürfen im Folgenden annehmen, dass die Methoden nur mit $1 \leq n \leq 46$ aufgerufen werden, so dass der Datentyp `long` zur Darstellung aller Werte ausreicht.

Exkurs: Fibonacci-Folge
Die Fibonacci-Folge beginnt zweimal mit der Zahl 1. Im Anschluss ergibt jeweils die Summe zweier aufeinanderfolgender Zahlen die unmittelbar danach folgende Zahl: 1, 1, 2, 3, 5, 8, 13

Exkurs: Partialsumme
Unter der n-ten Partialsumme s_n einer Zahlenfolge a_n versteht man die Summe der Folgenglieder von a_1 bis a_n. Die immer weiter fortgesetzte Partialsumme einer (unendlichen) Zahlenfolge nennt man eine (unendliche) Reihe. [a] Partialsummen sind das Bindeglied zwischen Summen und Reihen. Gegeben sei die Reihe $\sum_{k=1}^{\infty} a_k$. Die n-te Partialsumme dieser Reihe lautet: $\sum_{k=1}^{n} a_k$. ðwir summieren unsere Reihe nur bis zum Endindex n. [b]

[a] https://www.lernhelfer.de/schuelerlexikon/mathematik/artikel/folgen-partialsummen
[b] https://www.massmatics.de/merkzettel/index.php#!164:Partialsummen

sos steht für *Summe of Squares*

n	fib_n	fib_n^2		$\sum_{k=1}^{n} \text{fib}^k$
1	1	1	1	1
2	1	1	$1+1$	2
3	2	4	$1+1+4$	6
4	3	9	$1+1+4+9$	15
5	5	25	$1+1+4+9+25$	40
6	8	64	$1+1+4+9+25+64$	104
7	13	169	$1+1+4+9+25+64+169$	273
8	21	441	$1+1+4+9+25+64+169+441$	714
9	34	1156	$1+1+4+9+25+64+169+441+1156$	1870
10	55	3025	$1+1+4+9+25+64+169+441+1156+3025$	4895

(a) Implementieren Sie die obigen Formeln zunächst rekursiv (ohne Schleifenkonstrukte wie `for` oder `while`) und ohne weitere Optimierungen („naiv”) in Java als:

```
long fibNaive (int n) {
```

bzw.

```
long sosNaive (int n) {
```

Lösungsvorschlag

```
public static long fibNaive(int n) {
  if (n <= 2) {
    return 1;
  }
  return fibNaive(n - 1) + fibNaive(n - 2);
}

public static long sosNaive(int n) {
  if (n <= 1) {
```

```java
      return fibNaive(n);
    }
    return fibNaive(n) * fibNaive(n) + sosNaive(n - 1);
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2017/fruehjahr/Fibonacci.java`

(b) Offensichtlich ist die naive Umsetzung extrem ineffizient, da viele Zwischenergebnisse wiederholt rekursiv ausgewertet werden müssen. Die Dynamische Programmierung (DP) erlaubt es Ihnen, die Laufzeit auf Kosten des Speicherbedarfs zu reduzieren, indem Sie alle einmal berechneten Zwischenergebnisse speichern und bei erneutem Bedarf „direkt abrufen“. Implementieren Sie obige Formeln nun rekursiv aber mittels DP in Java als:

```java
long fibDP (int n) {
```

bzw.

```java
long sosDP (int n) {
```

Lösungsvorschlag

```java
  public static long fibDP(int n) {
    // Nachschauen, ob die Fibonacci-Zahl bereits berechnet wurde.
    if (fib[n] != 0) {
      return fib[n];
    }
    // Die Fibonacci-Zahl neu berechnen.
    if (n <= 2) {
      fib[n] = 1;
    } else {
      fib[n] = fibDP(n - 1) + fibDP(n - 2);
    }
    return fib[n];
  }

  public static long sosDP(int n) {
    // Nachschauen, ob die Quadratsumme bereits berechnet wurde.
    if (sos[n] != 0) {
      return sos[n];
    }
    // Die Quadratsumme neu berechnen.
    if (n <= 1) {
      sos[n] = fibDP(n);
    } else {
      long tmp = fibDP(n);
      sos[n] = tmp * tmp + sosDP(n - 1);
    }
    return sos[n];
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2017/fruehjahr/Fibonacci.java`

(c) Am „einfachsten“ und bzgl. Laufzeit [in $\mathcal{O}(n)$] sowie Speicherbedarf [in $\mathcal{O}(1)$] am effizientesten ist sicherlich eine iterative Implementierung der beiden Formeln. Geben Sie eine solche in Java an als:

```
long fibIter (int n) {
```

bzw.

```
long sosIter (int n) {
```

Lösungsvorschlag

```
public static long fibIter(int n) {
  long a = 1;
  long b = 1;
  for (int i = 2; i < n; i++) {
    long tmp = a + b;
    b = a;
    a = tmp;
  }
  return a;
}

public static long sosIter(int n) {
  long a = 1;
  long b = 0;
  long sosSum = 1;
  for (int i = 2; i <= n; i++) {
    long tmp = a + b;
    b = a;
    a = tmp;
    sosSum += a * a;
  }
  return sosSum;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2017/fruehjahr/Fibonacci.java`

Aufgabe 4 [Methode „sumOfSquares()“]

Sie dürfen im Folgenden davon ausgehen, dass keinerlei Under- oder Overflows auftreten.

Gegeben sei folgende rekursive Methode für $n \geq 0$:

```
long sumOfSquares (long n) {
  if (n == 0)
    return 0;
  else
    return n * n + sumOfSquares(n - 1);
}
```

(a) Beweisen Sie formal mittels vollständiger Induktion:

$$\forall n \in \mathbb{N} : \texttt{sumOfSquares(n)} = \frac{n(n+1)(2n+1)}{6}$$

Sei $f(n) : \frac{n(n+1)(2n+1)}{6}$

Induktionsanfang

— Beweise, dass $A(1)$ eine wahre Aussage ist.

Für $n = 0$ gilt:

`sumOfSquares(0)` $\overset{\texttt{if}}{=} 0 = f(0)$

Induktionsvoraussetzung

— Die Aussage $A(k)$ ist wahr für ein beliebiges $k \in \mathbb{N}$.

Für ein festes $n \in \mathbb{N}$ gelte:
`sumOfSquares(n)` $= f(n)$

Induktionsschritt

— Beweise, dass wenn $A(n = k)$ wahr ist, auch $A(n = k + 1)$ wahr sein muss.

$n \to n + 1$

$$
\begin{aligned}
f(n+1) &= \texttt{sumOfSquares(n+1)} && \text{Java-Methode eingesetzt}\\
&\overset{\texttt{else}}{=} \texttt{(n+1)*(n+1) + sumOfSquares(n)} && \text{Java-Code der else-Verzweigung verwendet}\\
&\overset{\texttt{I.H.}}{=} (n+1)(n+1) + f(n) && \text{mathematisch notiert}\\
&= (n+1)(n+1) + \frac{n(n+1)(2n+1)}{6} && \text{Formel eingesetzt}\\
&= (n+1)^2 + \frac{n(n+1)(2n+1)}{6} && \text{potenziert}\\
&= \frac{6(n+1)^2}{6} + \frac{n(n+1)(2n+1)}{6} && (n+1)^2 \text{ in Bruch umgewandelt}\\
&= \frac{6(n+1)^2 + n(n+1)(2n+1)}{6} && \text{Addition gleichnamiger Brüche}\\
&= \frac{(n+1)6(n+1) + (n+1)n(2n+1)}{6} && n+1 \text{ ausklammern vorbereitet}\\
&= \frac{(n+1)(6(n+1) + n(2n+1))}{6} && n+1 \text{ ausgeklammert}\\
&= \frac{(n+1)(6n+6+2n^2+n))}{6} && \text{Klammern ausmultipliziert / aufgelöst}\\
&= \frac{(n+1)(2n^2+7n+6)}{6} && \text{umsortiert, addiert } 6n+n=7n\\
&= \frac{(n+1)(2n^2+3n+4n+6)}{6} && \text{Ausklammern vorbereitet}\\
&= \frac{(n+1)(n+2)(2n+3)}{6} && (n+2) \text{ ausgeklammert}\\
&= \frac{(n+1)((n+1)+1)(2(n+1)+1))}{6} && (n+1) \text{ verdeutlicht}
\end{aligned}
$$

[a]

[a] `https://mathcs.org/analysis/reals/infinity/answers/sm_sq_cb.html`

(b) Beweisen Sie die Terminierung von `sumOfSquares(n)` für alle $n \geq 0$.

Lösungsvorschlag

Sei $T(n) = n$. Die Funktion $T(n)$ ist offenbar ganzzahlig. In jedem Rekursionsschritt wird n um eins verringert, somit ist $T(n)$ streng monoton fallend. Durch die Abbruchbedingung `n==0` ist $T(n)$ insbesondere nach unten beschränkt. Somit ist T eine gültige Terminierungsfunktion.

Aufgabe 5 [Aussagen]

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

(a) Alle regulären Sprachen liegen in NP.

Lösungsvorschlag

Stimmt. Alle regulären Sprachen sind in Polynomialzeit entscheidbar (es existiert ein Automat dazu), sie liegen als in P und folglich auch in NP.

(b) Es gibt Sprachen A, B mit $A \subseteq B$, sodass B regulär und A kontextfrei ist.

Lösungsvorschlag

Stimmt. Es existieren Sprachen mit der Eigenschaft wie gefordert. Wir wählen: $B = (a|b)^*$ und $A = \{ a^n b^n \mid n \in \mathbb{N} \}$. A ist bekanntermaßen nicht regulär, wie man mit dem Pumping Lemma beweisen kann, kann aber durch eine Grammatik $G = (V, \{a, b\}, \{S \rightarrow aSb \mid \varepsilon\}, S)$ erzeugt werden. Für B gibt es einen deterministischen endlichen Automaten.

(c) Es gibt unentscheidbare Sprachen L über den Alphabet Σ, so dass sowohl L als auch das Komplement $\overline{L} = \Sigma^* \setminus L$ rekursiv aufzählbar (= partiell entscheidbar) sind.

Lösungsvorschlag

Stimmt nicht. Ist L und sein Komplement rekursiv aufzählbar, so können wir L entscheiden, denn wir haben eine Maschine, die auf Eingabe x hält und akzeptiert, wenn $x \in L$ ist, sowie eine Maschine die hält, wenn x und akzeptiert, wenn $x \notin L$ ist. Daraus lässt sich eine Maschine konstruieren, die L entscheidet.

(d) Sei L eine beliebige kontextfreie Sprache über dem Alphabet Σ. Dann ist das Komplement $\overline{L} = \Sigma^* \setminus L$ entscheidbar.

Lösungsvorschlag

Stimmt. Es gibt einen Entscheider für die Sprache L. Dieser entscheidet für eine Eingabe x, ob diese in L ist oder nicht. Negiert man diese Entscheidung, so ergibt sich ein Entscheider für $\overline{L}$.

Schreiben Sie zuerst zur Aussage *„Stimmt“* oder *„Stimmt nicht“* und dann Ihre Begründung.

Thema Nr. 2

Aufgabe 2 [Nonterminale: STU, Terminale: abcde]

(a) Gegeben sei die kontextfreie Grammatik $G = (V, \Sigma, P, S)$ mit Sprache L(G), wobei $V = S, T, U$ und $\Sigma = \{a, b, c, d, e\}$. P bestehe aus den folgenden Produktionen:

$$P = \left\{ \begin{aligned} S &\rightarrow U \mid SbU \\ T &\rightarrow dSe \mid a \\ U &\rightarrow T \mid UcT \end{aligned} \right\}$$

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/Gib25c5oc

(i) Zeigen Sie $acdae \in L(G)$.

Lösungsvorschlag

$S \vdash U \vdash UcT \vdash TcT \vdash acT \vdash acdSe \vdash acdUe \vdash acdae$

(ii) Bringen Sie G in Chomsky-Normalform.

Lösungsvorschlag

i. **Elimination der ε-Regeln**
— *Alle Regeln der Form $A \rightarrow \varepsilon$ werden eliminiert. Die Ersetzung von A wird durch ε in allen anderen Regeln vorweggenommen.*

∅ Nichts zu tun

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

$$P = \left\{ \begin{aligned} S &\rightarrow dSe \mid a \mid UcT \mid SbU \\ T &\rightarrow dSe \mid a \\ U &\rightarrow dSe \mid a \mid UcT \end{aligned} \right\}$$

iii. **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 $S_\sigma \rightarrow \sigma$ ergänzt.*

$P = \{$

$$
\begin{aligned}
S &\rightarrow DSE \mid a \mid UCT \mid SBU \\
T &\rightarrow DSE \mid a \\
U &\rightarrow DSE \mid a \mid UCT \\
B &\rightarrow b \\
C &\rightarrow c \\
D &\rightarrow d \\
E &\rightarrow e
\end{aligned}
$$

$\}$

iv. **Elimination von mehrelementigen Nonterminalketten**
— *Alle Produktionen der Form $A \rightarrow B_1 B_2 \dots B_n$ werden in die Produktionen $A \rightarrow A_{n-1} B_n, A_{n-1} \rightarrow A_{n-2} B_{n-1}, \dots, A_2 \rightarrow B_1 B_2$ zerteilt. Nach der Ersetzung sind alle längeren Nonterminalketten vollständig heruntergebrochen und die Chomsky-Normalform erreicht.* ——

$\mathrm{P} = \{$

$$
\begin{aligned}
S &\rightarrow DS_E \mid a \mid UC_T \mid SB_U \\
T &\rightarrow DS_E \mid a \\
U &\rightarrow DS_E \mid a \mid UC_T \\
B &\rightarrow b \\
C &\rightarrow c \\
D &\rightarrow d \\
E &\rightarrow e \\
S_E &\rightarrow SE \\
C_T &\rightarrow CT \\
B_U &\rightarrow BU
\end{aligned}
$$

$\}$

(b) Geben Sie eine kontextfreie Grammatik für $L = \{ a^i b^k c^i | i, k \in \mathbb{N} \mid a \}$n.

Lösungsvorschlag

Wir interpretieren $\mathbb{N}$ als $\mathbb{N}_0$.

$\mathrm{P} = \{$

$S \rightarrow aSc \mid aBc \mid B \mid \varepsilon B \qquad \rightarrow b \mid Bb$

$\}$

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/Ghp3bfdtg

(c) Zeigen Sie, dass $L = \{ a^i b^k c^i | i, k \in \mathbb{N} \wedge i < k \mid n \}$icht kontextfrei ist, indem Sie das Pumping-Lemma für kontextfreie Sprachen anwenden.

Exkurs: Pumping-Lemma für Reguläre Sprachen

Es sei L eine kontextfreie Sprache. Dann gibt es eine Zahl j, sodass sich alle Wörter $\omega \in L$ mit $|\omega| \geq j$ zerlegen lassen in $\omega = uvwxy$, sodass die folgenden Eigenschaften erfüllt sind:

(i) $|vx| \geq 1$ (Die Wörter v und x sind nicht leer.)

(ii) $|vwx| \leq j$ (Die Wörter v, w und x haben zusammen höchstens die Länge j.)

(iii) Für alle $i \in \mathbb{N}_0$ gilt $uv^iwx^iy \in L$ (Für jede natürliche Zahl (mit 0) i ist das Wort uv^iwx^iy in der Sprache L)

Lösungsvorschlag

Diese Aufgabe hat noch keine Lösung. Hilf mit! Die Hermine schafft das nicht allein! Das ist ein Community-Projekt! Verbesserungsvorschläge, Fehlerkorrekturen, weitere Lösungen sind herzlich willkommen - egal wie - per Pull-Request oder per E-Mail an hermine.bschlangaul@gmx.net.

Aufgabe 3 [Berechen- und Entscheidbarkeit]

(a) Primitiv rekursive Funktionen

(i) Zeigen Sie, dass die folgendermaßen definierte Funktion if: $\mathbb{N} \times \mathbb{N} \times \mathbb{N}\mathbb{N}$ primitiv rekursiv ist.
sonst

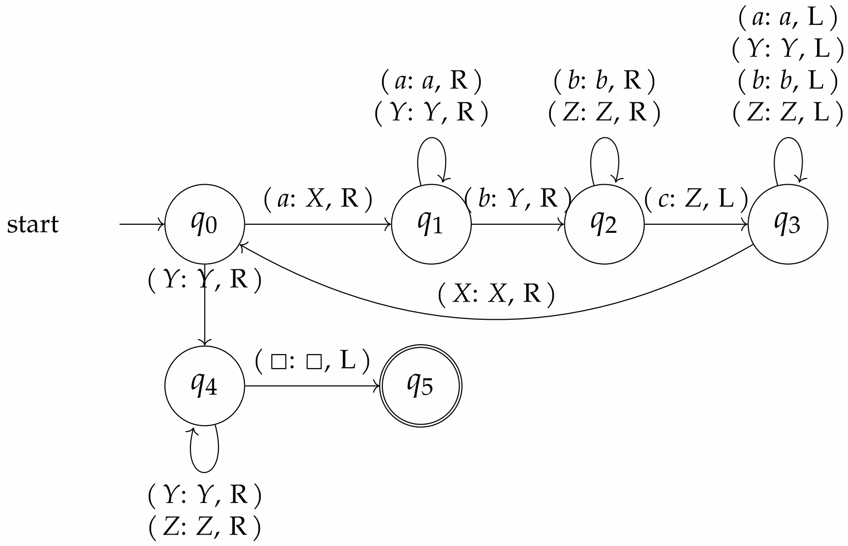
(ii) Wir nehmen eine primitiv rekursive Funktionp: NN an und definieren g(n) als die Funktion, welche die größte Zahl i < n zurückliefert, für die p(/) = 0 gilt. Falls kein solches i existiert, soll g(n) = 0 gelten:
a(n) = max (i <n |p) = 0 U 0)
if (b, x, y) = (falls b=0

Zeigen Sie, dass g: N > N primitiv rekursiv ist. (Sie dürfen obige Funktion if als primitiv rekursiv voraussetzen.)

(b) Sei $\Sigma = \{a, b, c\}$ und $L \subseteq \Sigma^*$ mit $L = \{ a^ib^ic^i \mid i \in N \}$.

(i) Beschreiben Sie eine Turingmaschine, welche die Sprache Z entscheidet. Eine textuelle Beschreibung der Konstruktionsidee ist ausreichend.

Lösungsvorschlag



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/Apew1n7g9 [a]

[a] https://scanftree.com/automata/turing-machine-for-a-to-power-n-b-to-power-n-c-to-power

(ii) Geben Sie Zeit- und Speicherkomplexität (abhängig von der Länge der Eingabe) Ihrer Turingmaschine an.

Lösungsvorschlag

Speicherkomplexität n (Das Eingabewort wird einmal überschrieben)

Zeitkomplexität the turing machine time complexity is the number of transition execution will executed is call time complexity of the turing machine. first we start we main loop execution is (n/3)-1. transition(a,x,R) from state 1 to 2= 1. transition (a,a,R) and (y,y,R) on state 2 is = (n/3)-1. transition (b,y,R) from state 2 to 3=1. on state 3 (b,b,R) and (z,z,R)=(n/3)-1. transition (c,z,L) from state 3 to 4=1. on state 4 (y,y,L),(b,b,L),(z,z,L) and state 5 (a,a,L)= (n/3)-1. transition (a,a,L) form state 4 to 5 =1. transition (x,x,R) from 5 to1 =1 total(n+2) following transition will executed transition(a,x,R) from state 1 to 2= 1. transition (y,y,R) on state 2 is = (n/3)-1. transition (b,y,R) from state 2 to 3=1. transition (z,z,R) on state 3=(n/3)-1 transition (c,z,L) from state 3 to 4=1. on state 4 (y,y,L) ,(z,z,L) and state (n/3)-1. transition (x,x,R) from state 54 to 6 =1 transition on state 6 (y,y,R),(z,z,R)= (n/3) transition (d,d,R) from state 6 to 7 =1 total =(4n/3)+2 over alti time complexity (n+2)(n/3)-1+ (4n/3)+2 [a]

[a] https://www.youtube.com/watch?v=vwnz9e_Lrfo

(c) Sei $\Sigma = \{0, 1\}$. Jedes $w \in \Sigma^*$ kodiert eine Turingmaschine M_w. Die von M_w berechnete Funktion bezeichnen wir mit $\varphi_w(x)$.

(i) Warum ist $L = \{\, w \in \Sigma^* \mid \exists x\colon \varphi_w(x) = xx \,\}$ nicht entscheidbar?

(ii) Warum ist $L = \{\, w \in \Sigma^* \mid \exists x \colon w = xx \,\}$ entscheidbar?