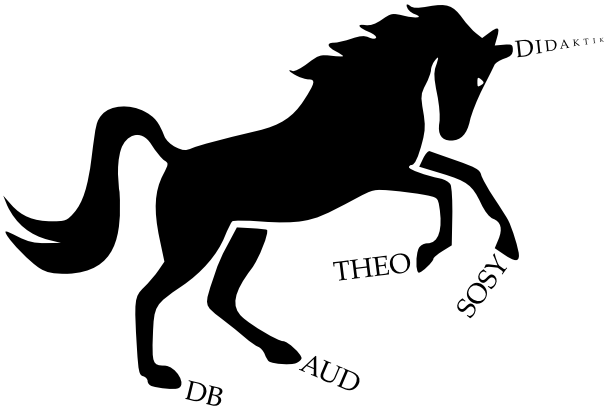
46115
Herbst
2016

Theoretische Informatik / Algorithmen / Datenstrukturen (nicht vertieft)

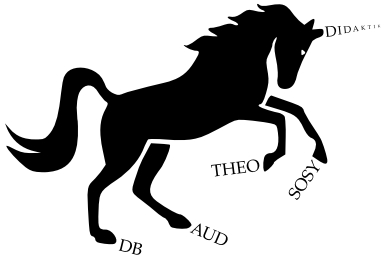
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



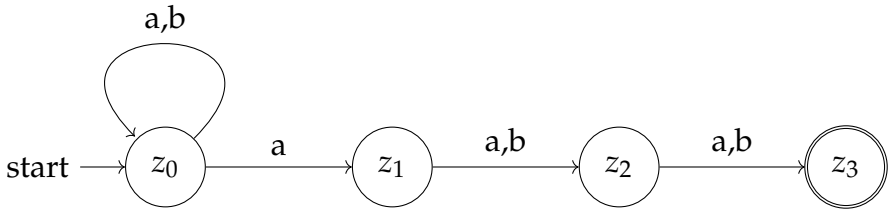
Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

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 [Alphabet ab, vorvorletztes Zeichen a]

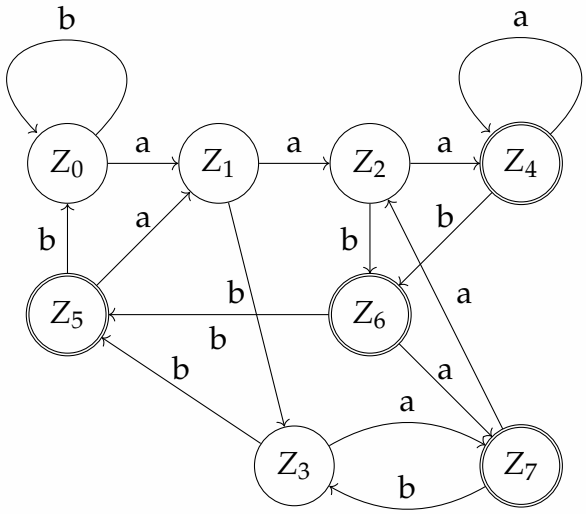
(a) Konstruieren Sie aus dem NEA mit der Potenzmengenkonstruktion einen (deterministischen) EA, der dieselbe Sprache akzeptiert.



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/Aiqxdazuw

Lösungsvorschlag

Name	Zustandsmenge	Eingabe a	Eingabe b
Z_0	$Z_0\{z_0\}$	$Z_1\{z_0, z_1\}$	$Z_0\{z_0\}$
Z_1	$Z_1\{z_0, z_1\}$	$Z_2\{z_0, z_1, z_2\}$	$Z_3\{z_0, z_2\}$
Z_2	$Z_2\{z_0, z_1, z_2\}$	$Z_4\{z_0, z_1, z_2, z_3\}$	$Z_6\{z_0, z_2, z_3\}$
Z_3	$Z_3\{z_0, z_2\}$	$Z_7\{z_0, z_1, z_3\}$	$Z_5\{z_0, z_3\}$
Z_4	$Z_4\{z_0, z_1, z_2, z_3\}$	$Z_4\{z_0, z_1, z_2, z_3\}$	$Z_6\{z_0, z_2, z_3\}$
Z_5	$Z_5\{z_0, z_3\}$	$Z_1\{z_0, z_1\}$	$Z_0\{z_0\}$
Z_6	$Z_6\{z_0, z_2, z_3\}$	$Z_7\{z_0, z_1, z_3\}$	$Z_5\{z_0, z_3\}$
Z_7	$Z_7\{z_0, z_1, z_3\}$	$Z_2\{z_0, z_1, z_2\}$	$Z_3\{z_0, z_2\}$



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/Aiqnk06c9

(b) Beschreiben Sie möglichst einfach, welche Sprachen von den folgenden regulären Ausdrücken beschrieben werden:

(i) $(a|b)^*a$

Lösungsvorschlag

Sprache mit dem Alphabet $\Sigma = \{a, b\}$: Alle Wörter der Sprache enden auf a.

(ii) $(a|b)^*a(a|b)^*a(a|b)^*$

Lösungsvorschlag

Sprache mit dem Alphabet $\Sigma = \{a, b\}$: Alle Wörter der Sprache enthalten mindestens 2 a's.

(iii) $(a|b)^*a(bb)^*a(a|b)^*$

Lösungsvorschlag

Sprache mit dem Alphabet $\Sigma = \{a, b\}$: Alle Wörter der Sprache enthalten mindestens 2 a's, die von einer geradzahligen Anzahl von b's getrennt sind.

Aufgabe 4 [VertexCover]

Betrachten Sie die beiden folgenden Probleme:

`VERTEX COVER`

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \{1, 2, 3, \dots\}$

Frage: Gibt es eine Menge $C \subseteq V$ mit $|C| \leq k$, so dass für jede Kante (u, v) aus E mindestens einer der Knoten u und v in C ist?

`VERTEX COVER 3`

Gegeben: Ein ungerichteter Graph $G = (V, E)$ und eine Zahl $k \in \{3, 4, 5 \dots\}$.

Frage: Gibt es eine Menge $C \subseteq V$ mit $|C| \leq k$, so dass für jede Kante (u, v) aus E mindestens einer der Knoten u und v in C ist?

Geben Sie eine polynomielle Reduktion von `VERTEX COVER` auf `VERTEX COVER 3` an und begründe anschließend, dass die Reduktion korrekt ist.

Exkurs: `VERTEX COVER`
Das **Knotenüberdeckungsproblem** (`VERTEX COVER`) fragt, ob zu einem gegebenen einfachen Graphen und einer natürlichen Zahl k eine Knotenüberdeckung der Größe von höchstens k existiert.

Das heißt, ob es eine aus maximal k Knoten bestehende Teilmenge U der Knotenmenge gibt, so dass jede Kante des Graphen mit mindestens einem Knoten aus U verbunden ist.

Lösungsvorschlag

$$\texttt{VERTEX COVER} \preceq_p \texttt{VERTEX COVER 3}$$

f fügt vier neue Knoten hinzu, von denen jeweils ein Paar verbunden ist. Außerdem erhöht f k um 2.

Total: Jeder Graph kann durch f so verändert werden.

Korrektheit: Wenn `VERTEX COVER` für k in G existiert, dann existiert auch `VERTEX COVER` mit $k+2$ Knoten in G', da für den eingefügten Teilgraphen ein `VERTEX COVER` mit $k=2$ existiert.

In Polynomialzeit berechenbar: Für die Adjazenzmatrix des Graphen müssen lediglich vier neue Spalten und Zeilen eingefügt werden und $k+2$ berechnet werden.

Thema Nr. 2

Aufgabe 2 [Methoden „matrixSumme()“ und „find()“]

Geben Sie jeweils die kleinste, gerade noch passende Laufzeitkomplexität folgender Java-Methoden im O-Kalkül (Landau-Notation) in Abhängigkeit von n und ggf. der Länge der Arrays an.

(a)

```java
int matrixSumme(int n, int[][] feld) {
  int sum = 0;
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      sum += feld[i][j];
    }
  }
  return sum;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/Komplexitaet.java`

Lösungsvorschlag

Die Laufzeit liegt in $\mathcal{O}(n^2)$.

Begründung (nicht verlangt): Die äußere Schleife wird n-mal durchlaufen. Die innere Schleife wird dann jeweils wieder n-mal durchlaufen. Die Größe des Arrays spielt hier übrigens keine Rolle, da die Schleifen ohnehin immer nur bis zum Wert n ausgeführt werden.

(b)

```java
int find(int key, int[][] keys) {
  int a = 0, o = keys.length;
  while (o - a > 1) {
    int m = (a + o) / 2;
    if (keys[m][0] > key)
      o = m;
    else
      a = m;
  }
  return a;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/Komplexitaet.java`

Lösungsvorschlag

Die Laufzeit liegt in $\mathcal{O}(\log(\text{keys.length}))$. Dabei ist keys.length die Größe des Arrays bezüglich seiner ersten Dimension.

Begründung (nicht verlangt): Der Grund für diese Laufzeit ist derselbe wie bei der binären Suche. Die Größe des Arrays bezüglich seiner zweiten Dimension spielt hier übrigens keine Rolle, da diese Dimension hier ja nur einen einzigen festen Wert annimmt.

Aufgabe 7 [Methode „a()“]

Mittels Dynamischer Programmierung (auch Memoization genannt) kann man insbesondere rekursive Lösungen auf Kosten des Speicherbedarf beschleunigen, indem man Zwischenergebnisse „abspeichert “ und bei (wiederkehrendem) Bedarf „abruft“, ohne sie erneut berechnen zu müssen.

Gegeben sei folgende geschachtelt-rekursive Funktion für $n, m \geq 0$:

$$a(n,m) = \begin{cases} n + \lfloor \frac{n}{2} \rfloor & \text{falls } m = 0 \\ a(1, m-1), & \text{falls } n = 0 \wedge m \neq 0 \\ a(n + \lfloor \sqrt{a(n-1,m)} \rfloor, m-1), & \textit{sonst} \end{cases}$$

(a) Implementieren Sie die obige Funktion `a(n,m)` zunächst ohne weitere Optimierungen als Prozedur/Methode in einer Programmiersprache Ihrer Wahl.

Lösungsvorschlag

```java
public static long a(int n, int m) {
  if (m == 0) {
    return n + (n / 2);
  } else if (n == 0 && m != 0) {
    return a(1, m - 1);
  } else {
    return a(n + ((int) Math.sqrt(a(n - 1, m))), m - 1);
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java

(b) Geben Sie nun eine DP-Implementierung der Funktion `a(n,m)` an, die `a(n,m)` für $0 \geq n \geq 100000$ und $0 \geq m \geq 25$ höchstens einmal gemäß obiger rekursiver Definition berechnet. Beachten Sie, dass Ihre Prozedur trotzdem auch weiterhin mit $n > 100000$ und $m > 25$ aufgerufen werden können soll.

Lösungsvorschlag

```java
static long[][] tmp = new long[100001][26];

public static long aDp(int n, int m) {
  if (n <= 100000 && m <= 25 && tmp[n][m] != -1) {
    return tmp[n][m];
  } else {
```

```
        long merker;
        if (m == 0) {
          merker = n + (n / 2);
        } else if (n == 0 && m != 0) {
          merker = aDp(1, m - 1);
        } else {
          merker = aDp(n + ((int) Math.sqrt(aDp(n - 1, m))), m - 1);
        }
        if (n <= 100000 && m <= 25) {
          tmp[n][m] = merker;
        }
        return merker;
      }
    }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java`

Lösungsvorschlag

Kompletter Code

```
public class DynamischeProgrammierung {
  public static long a(int n, int m) {
    if (m == 0) {
      return n + (n / 2);
    } else if (n == 0 && m != 0) {
      return a(1, m - 1);
    } else {
      return a(n + ((int) Math.sqrt(a(n - 1, m))), m - 1);
    }
  }

  static long[][] tmp = new long[100001][26];

  public static long aDp(int n, int m) {
    if (n <= 100000 && m <= 25 && tmp[n][m] != -1) {
      return tmp[n][m];
    } else {
      long merker;
      if (m == 0) {
        merker = n + (n / 2);
      } else if (n == 0 && m != 0) {
        merker = aDp(1, m - 1);
      } else {
        merker = aDp(n + ((int) Math.sqrt(aDp(n - 1, m))), m - 1);
      }
      if (n <= 100000 && m <= 25) {
        tmp[n][m] = merker;
      }
      return merker;
    }
  }

  public static void main(String[] args) {
```

```
    for (int i = 0; i < 100001; i++) {
      for (int j = 0; j < 26; j++) {
        tmp[i][j] = -1;
      }
    }
    System.out.println("schnell mit DP: " + aDp(7,7));
    System.out.println("langsam ohne DP: " + a(7,7));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2016/herbst/DynamischeProgrammierung.java`