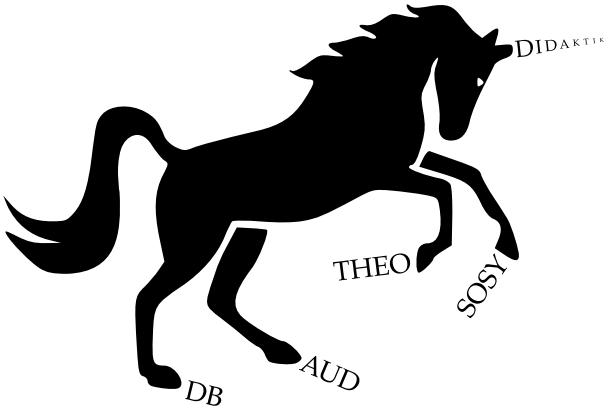
66115
Herbst
2020

Theoretische Informatik / Algorithmen (vertieft)

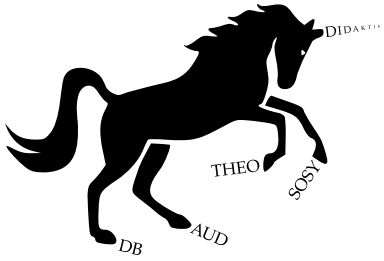
Aufgabenstellungen mit Lösungsvorschlägen



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

Teilaufgabe Nr. 1

Aufgabe 1 [Vermische Fragen]

Antworten Sie mit *„Stimmt“* oder *„Stimmt nicht“*. Begründen Sie Ihr Urteil kurz.

(a) Eine Sprache ist genau dann regulär, wenn sie unendlich viele Wörter enthält.

Lösungsvorschlag

Stimmt nicht. Sprachen mit endlicher Mächtigkeit sind immer regulär. Endliche Spachen sind in obenstehender Aussage ausgeschlossen.

(b) Zu jedem nichtdeterministischen endlichen Automaten mit n Zuständen gibt es einen deterministischen endlichen Automaten, der die gleiche Sprache erkennt und höchstens n^2 Zustände hat.

Lösungsvorschlag

Stimmt nicht. Müsste 2^n heißen.

(c) Das Komplement einer kontextfreien Sprache ist wieder kontextfrei.

Lösungsvorschlag

Stimmt nicht. Kontextfreie Sprachen sind nicht abgeschlossen unter dem Komplement. Das Komplement einer kontextfreien Sprache kann regulär, kontextfrei oder kontextsensitiv sein.

(d) Wenn ein Problem unentscheidbar ist, dann ist es nicht semientscheidbar.

Lösungsvorschlag

Stimmt nicht. Semientscheidbarkeit ist eine typische Form der Unentscheidbarkeit. Unentscheidbarkeit ist das Gegenteil von Entscheidbarkeit. Unentscheidbar kann entweder völlig unentscheidbar sein oder semientscheidbar.

(e) Sei f eine totale Funktion. Dann gibt es ein WHILE-Programm, das diese berechnet.

Lösungsvorschlag

Stimmt nicht. Wir wissen nicht, ob die totale Funktion f berechenbar ist. Wenn f berechenbar ist, dann wäre die Aussage richtig.

(f) Das Halteproblem für `LOOP`-Programme ist entscheidbar.

Lösungsvorschlag

Stimmt. Alle LOOP-Programme terminieren (halten). Es gibt für jede Eingabe eine Ausgabe.

(g) Die Komplexitätsklasse $\mathcal{NP}$ enthält genau die Entscheidungsprobleme, die in nicht-polynomieller Zeit entscheidbar sind.

Lösungsvorschlag

Stimmt. Die Aussage entspricht der Definiton der Komplexitätsklasse $\mathcal{NP}$.

(h) Falls $P \geq NP$, dann gibt es keine $\mathcal{NP}$-vollständigen Probleme, die in P liegen.

Lösungsvorschlag

Stimmt. Entspricht der Definition.

Aufgabe 2 [Reguläre Sprache xyz]

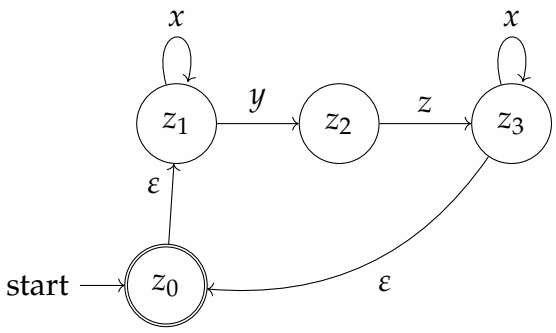
Sei $\Sigma = \{x, y, z\}$. Sei $L = (x^* yzx^*)^* \subseteq \Sigma^*$.

(a) Geben Sie einen endlichen (deterministischen oder nichtdeterministischen) Automaten A an, der L erkennt bzw. akzeptiert.

Lösungsvorschlag

Nichtdeterministischer endlicher Automat

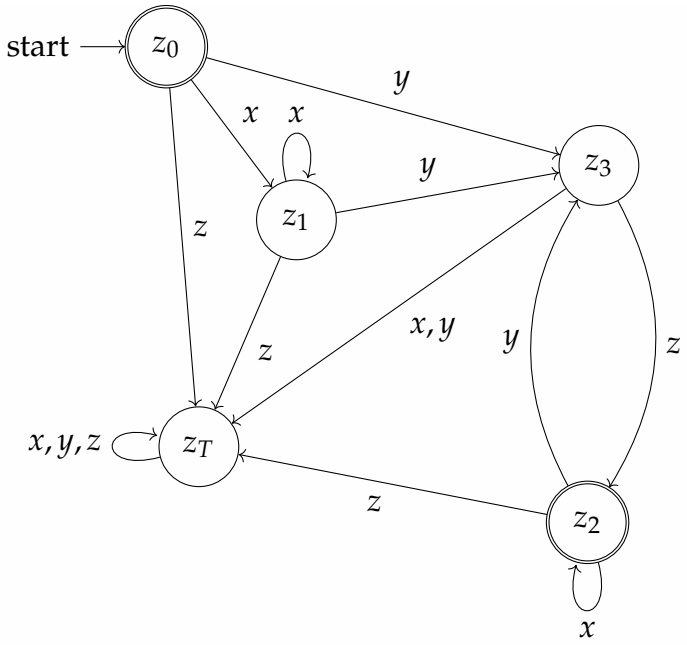
$A_{\text{NEA}} = (\{z_0, z_1, z_2, z_3, z_T\}, \{x, y, z\}, \delta, \{z_0, z_2\}, z_0)$



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/Ajpmxqvh9

Deterministischer endlicher Automat

$A_{\text{DEA}} = (\{z_0, z_1, z_2, z_3, z_T\}, \{x, y, z\}, \delta, \{z_0, z_2\}, z_0)$



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/A5xo470g9

(b) Geben Sie eine reguläre und eindeutige Grammatik G an, die L erzeugt.

Lösungsvorschlag

$$
\mathrm{P} = \left\{ \begin{array}{l} Z_0 \to xZ_1 \mid yZ_3 \mid \varepsilon \\ Z_1 \to yZ_3 \mid xZ_1 \\ Z_2 \to xZ_2 \mid x \mid yZ_3 \\ Z_3 \to zZ_2 \mid z \end{array} \right\}
$$

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/Gjfc3c2d2

Aufgabe 3 [Palindrom über Alphabet „abc“]

Seien $\Sigma = \{a, b, c\}$ und $L = \{ wc\hat{w} \mid w \in \{a, b\}* \}$. Dabei ist $\hat{w}$ das zu w gespiegelte Wort.

(a) Zeigen Sie, dass L nicht regulär ist.

Exkurs: Pumping-Lemma für Reguläre Sprachen

Es sei L eine reguläre Sprache. Dann gibt es eine Zahl j, sodass für alle Wörter $\omega \in L$ mit $|\omega| \geq j$ (jedes Wort ω in L mit Mindestlänge j) jeweils eine Zerlegung $\omega = uvw$ existiert, sodass die folgenden Eigenschaften erfüllt sind:

(i) $|v| \geq 1$ (Das Wort v ist nicht leer.)

(ii) $|uv| \leq j$ (Die beiden Wörter u und v haben zusammen höchstens die Länge j.)

(iii) Für alle $i = 0, 1, 2, \ldots$ gilt $uv^i w \in L$ (Für jede natürliche Zahl (mit 0) i ist das Wort $uv^i w$ in der Sprache L)

Die kleinste Zahl j, die diese Eigenschaften erfüllt, wird Pumping-Zahl der Sprache L genannt.

Lösungsvorschlag

L ist regulär. Dann gilt für L das Pumping-Lemma. Sei j die Zahl aus dem Pumping-Lemma. Dann muss sich das Wort $a^j bcba^j \in L$ aufpumpen lassen (da $|a^j bcba^j| \geq j$). $a^j bcba^j = uvw$ ist eine passende Zerlegung laut Lemma. Da $|uv| < j$, ist $u = a^x$, $v = a^y, w = a^z bcba^j$, wobei $y > 0$ und $x + y + z = j$. Aber dann $uv^0 w = a^{x+z} bcba^j \notin L$, da $x + z < j$. Widerspruch. [a]

[a] `https://userpages.uni-koblenz.de/~sofronie/gti-ss-2015/slides/endliche-automaten6.pdf`

(b) Zeigen Sie, dass L kontextfrei ist, indem Sie eine geeignete Grammatik angeben und anschließend begründen, dass diese die Sprache L erzeugt.

Lösungsvorschlag

$$P = \left\{ \begin{array}{l} S \to aSa \mid aCa \mid bSb \mid bCb \\ C \to c \end{array} \right\}$$

$S \vdash aSa \vdash abCba \vdash abcba$

$S \vdash bSb \vdash bbSbb \vdash bbaSabb \vdash bbacabb$

Teilaufgabe Nr. 2

Aufgabe 1 [Algorithmenanalyse]

Betrachten Sie die folgende Prozedur `countup`, die aus zwei ganzzahligen Eingabewerten `n` und `m` einen ganzzahligen Ausgabewert berechnet:

```
procedure countup(n, m : integer): integer
var x, y : integer;
```

```
begin
  x := n;
  y := 0;
  while (y < m) do
    x := x - 1;
    y := y + 1;
  end while
  return x;
end
```

(a) Führen Sie `countup(3,2)` aus. Geben Sie für jeden Schleifendurchlauf jeweils den Wert der Variablen `n`, `m`, `x` und `y` zu Beginn der while-Schleife und den Rückgabewert der Prozedur an.

Lösungsvorschlag

n	m	x	y	ausgeführter Code, der Änderung bewirkte
3	2	3	0	
3	2	2	1	`x := x - 1; y := y + 1;`

Rückgabewert: 1

Java-Implementation der Prozedur

```java
public class CountUp {

  public static int countup(int n, int m) {
    int x = n;
    int y = 0;
    while (y < m) {
      System.out.println(String.format("%s %s %s %s", n, m, x, y));
      x = x - 1;
      y = y + 1;
    }
    return x;
  }

  public static void main(String[] args) {
    System.out.println(countup(3, 2));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/counter/CountUp.java`

(b) Gibt es Eingabewerte von `n` und `m`, für die die Prozedur `countup` nicht terminiert? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Nein. Mit jedem Schleifendurchlauf wird der Wert der Variablen `y` um eins hochgezählt. Die Werte, die `y` annimmt, sind streng monoton steigend. `y` nähert sich `m` an, bis `y` nicht mehr kleiner ist als `m` und die Prozedur terminiert. An diesem Sachverhält ändern auch sehr große Zahlen, die über die Variable `m` der Prozedur übergeben

werden, nichts.

(c) Geben Sie die asymptotische worst-case Laufzeit der Prozedur `countup` in der Θ-Notation in Abhängigkeit von den Eingabewerten `n` und/oder `m` an. Begründen Sie Ihre Antwort.

Lösungsvorschlag

Die Laufzeit der Prozedur ist immer $\Theta(m)$. Die Laufzeit hängt nur von `m` ab. Es kann nicht zwischen best-, average and worst-case unterschieden werden.

(d) Betrachten Sie nun die folgende Prozedur `countdown`, die aus zwei ganzzahligen Eingabewerten `n` und `m` einen ganzzahligen Ausgabewert berechnet:

```
procedure countdown(n, m : integer) : integer
var x, y : integer;
begin
  x := n;
  y := 0;
  while (n > 0) do
    if (y < m) then
      x := x - 1;
      y := y + 1;
    else
      y := 0;
      n := n / 2; /* Ganzzahldivision */
    end if
  end while
  return x;
end
```

Führen Sie `countdown(3, 2)` aus. Geben Sie für jeden Schleifendurchlauf jeweils den Wert der Variablen `n`, `m`, `x` und `y` zu Beginn der `while`-Schleife und den Rückgabewert der Prozedur an.

Lösungsvorschlag

n	m	x	y	ausgeführter Code, der Änderung bewirkte
3	2	3	0	
3	2	2	1	`x := x - 1; y := y + 1;`
3	2	1	2	`x := x - 1; y := y + 1;`
1	2	1	0	`y := 0; n := n / 2;`
1	2	0	1	`x := x - 1; y := y + 1;`
1	2	-1	2	`x := x - 1; y := y + 1;`

Rückgabewert: -1

Java-Implementation der Prozedur

```
public class CountDown {

  public static int countdown(int n, int m) {
```

```java
    int x = n;
    int y = 0;
    while (n > 0) {
      System.out.println(String.format("%s %s %s %s", n, m, x, y));
      if (y < m) {
        x = x - 1;
        y = y + 1;
      } else {
        y = 0;
        n = n / 2; /* Ganzzahldivision */
      }
    }
    return x;
  }

  public static void main(String[] args) {
    System.out.println(countdown(3, 2));
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/counter/CountDown.java`

(e) Gibt es Eingabewerte von `n` und `m`, für die die Prozedur `countdown` nicht terminiert? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Nein.

`n <= 0` terminiert sofort

`m <= 0` Der Falsch-Block der Wenn-Dann-Bedingung erniedrigt `n` `n := n / 2;` bis `0` erreicht ist. Dann terminiert die Prozedur.

`m > 0` Der erste Wahr-Block der Wenn-Dann-Bedingung erhöht `y` streng monoton bis `y >= m`. 2. Falsch-Block der Wenn-Dann-Bedingung halbiert `n` bis `0`. 1. und 2. solange bis $n = 0$

(f) Geben Sie die asymptotische Laufzeit der Prozedur `countdown` in der Θ-Notation in Abhängigkeit von den Eingabewerten `n` und/oder `m` an unter der Annahme, dass `m >= 0` und `n > 0`. Begründen Sie Ihre Antwort.

Lösungsvorschlag

Anzahl der Wiederholungen der while-Schleife: $m + 1$:

- m oft: bis $y < m$
- $+1$ Halbierung von n und y auf 0 setzen

wegen dem `n/2` ist die Laufzeit logarithmisch, ähnlich wie der worst case bei der Binären Suche.

n	m	x	y	ausgeführter Code, der Änderung bewirkte
16	3	16	0	
16	3	15	1	
16	3	14	2	
16	3	13	3	
8	3	13	0	`y := 0; n := n / 2;`
8	3	12	1	
8	3	11	2	
8	3	10	3	
4	3	10	0	`y := 0; n := n / 2;`
4	3	9	1	
4	3	8	2	
4	3	7	3	
2	3	7	0	`y := 0; n := n / 2;`
2	3	6	1	
2	3	5	2	
2	3	4	3	
1	3	4	0	`y := 0; n := n / 2;`
1	3	3	1	
1	3	2	2	
1	3	1	3	

$\Theta((m+1)\log_2 n)$

Wegkürzen der Konstanten:

$\Rightarrow \Theta(m \log n)$

Aufgabe 2 [Bäume]

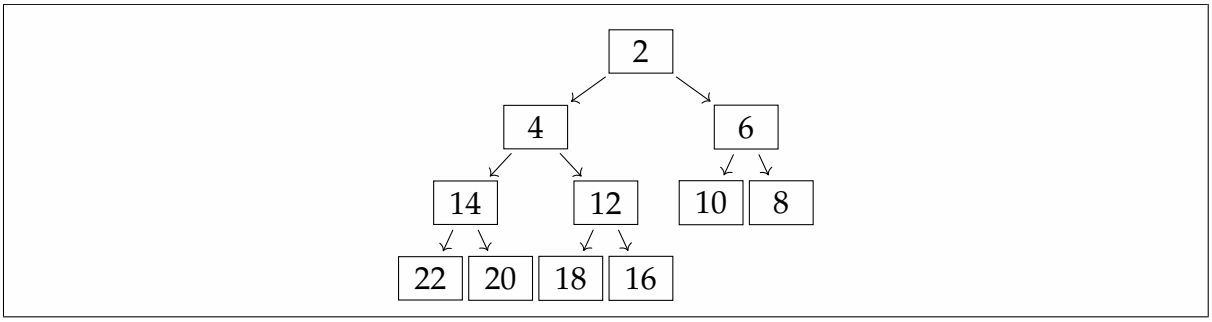
Wir betrachten ein Feld A von ganzen Zahlen mit `n` Elementen, die über die Indizes `A[0]` bis `A[n-1]` angesprochen werden können. In dieses Feld ist ein binärer Baum nach den folgenden Regeln eingebettet: Für das Feldelement mit Index i befindet sich

- der Elternknoten im Feldelement mit Index $\lfloor \frac{i-1}{2} \rfloor$,
- der linke Kindknoten im Feldelement mit Index $2 \cdot i + 1$, und
- der rechte Kindknoten im Feldelement mit Index $2 \cdot i + 2$.

(a) Zeichnen Sie den durch das folgende Feld repräsentierten binären Baum.

i	0	1	2	3	4	5	6	7	8	9	10
A[i]	2	4	6	14	12	10	8	22	20	18	16

Lösungsvorschlag



(b) Der folgende rekursive Algorithmus sei gegeben:

Pseudo-Code / Pascal

```
procedure magic(i, n : integer) : boolean
begin
  if (i > (n - 2) / 2) then
    return true;
  endif
  if (A[i] <= A[2 * i + 1] and A[i] <= A[2 * i + 2] and
      magic(2 * i + 1, n) and magic(2 * i + 2, n)) then
    return true;
  endif
  return false;
end
```

Java-Implementation

```
  public static boolean magic(int i, int n) {
    System.out.println(String.format("i: %s n: %s", i, n));
    if (i > (n - 2) / 2) {
      return true;
    }
    if (A[i] <= A[2 * i + 1] && A[i] <= A[2 * i + 2] && magic(2 * i + 1, n) && magic(2
↪   * i + 2, n)) {
      return true;
    }
    return false;
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java`

Gegeben sei folgendes Feld:

i	0	1	2	3
A[i]	2	4	6	14

Führen Sie `magic(0,3)` auf dem Feld aus. Welches Resultat liefert der Algorithmus zurück?

Lösungsvorschlag

true

(c) Wie nennt man die Eigenschaft, die der Algorithmus `magic` auf dem Feld A prüft? Wie lässt sich diese Eigenschaft formal beschreiben?

Lösungsvorschlag

Die sogenannte „Haldeneigenschaft“ bzw. „Heap-Eigenschaft“ einer Min-Halde. Der Schlüssel eines jeden Knotens ist kleiner (oder gleich) als die Schlüssel seiner Kinder.

Ein Baum erfüllt die Heap-Eigenschaft bezüglich einer Vergleichsrelation „$>$“ auf den Schlüsselwerten genau dann, wenn für jeden Knoten u des Baums gilt, dass $u_{\text{wert}} > v_{\text{wert}}$ für alle Knoten v aus den Unterbäumen von u.

(d) Welche Ausgaben sind durch den Algorithmus `magic` möglich, wenn das Eingabefeld aufsteigend sortiert ist? Begründen Sie Ihre Antwort.

Lösungsvorschlag

`true` . Eine sortierte aufsteigende Zahlenfolge entspricht den Haldeneigenschaften einer Min-Heap.

(e) Geben Sie zwei dreielementige Zahlenfolgen (bzw. Felder) an, eine für die `magic(0,2)` den Wert `true` liefert und eine, für die `magic(0,2)` den Wert `false` liefert.

Lösungsvorschlag

```
A = new int[] { 1, 2, 3 };
System.out.println(magic(0, 2)); // true

A = new int[] { 2, 1, 3 };
System.out.println(magic(0, 2)); // false
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java`

(f) Betrachten Sie folgende Variante `almostmagic` der oben bereits erwähnten Prozedur `magic`, bei der die Anweisungen in Zeilen 3 bis 5 entfernt wurden:

Pseudo-Code / Pascal

```
procedure almostmagic(i, n : integer) : boolean
begin
  // leer
  // leer
  // leer
  if (A[i] <= A[2 * i + 1] and A[i] <= A[2 * i + 2] and
      magic(2 * i + 1, n) and magic(2 * i + 2, n)) then
    return true;
  endif
  return false;
end
```

Java-Implementation

```
  public static boolean almostmagic(int i, int n) {
    System.out.println(String.format("i: %s n: %s", i, n));
    if (A[i] <= A[2 * i + 1] && A[i] <= A[2 * i + 2] && magic(2 * i + 1, n) && magic(2
↪   * i + 2, n)) {
      return true;
    }
    return false;
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java`

Beschreiben Sie die Umstände, die auftreten können, wenn `almostmagic` auf einem Feld der Größe `n` aufgerufen wird. Welchen Zweck erfüllt die entfernte bedingte Anweisung?

Lösungsvorschlag

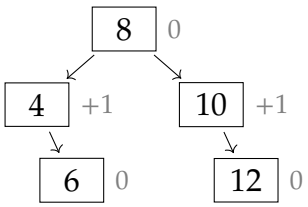
Wird die Prozedur zum Beispiel mit `almostmagic(0, n + 1)` aufgerufen, kommst es zu einem sogenannten „Array-Index-Ouf-of-Bounds“ Fehler, d. h. die Prozedur will auf Index des Feldes zugreifen, der im Feld gar nicht existiert. Die drei zusätzlichen Zeilen in der Methode `magic` bieten dafür einen Schutz, indem sie vor den Index-Zugriffen auf das Feld `true` zurückgeben.

```
    // A = new int[] { 1, 2, 3 };
    // System.out.println(almostmagic(0, 4)); // Exception in thread "main"
↪   java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
```

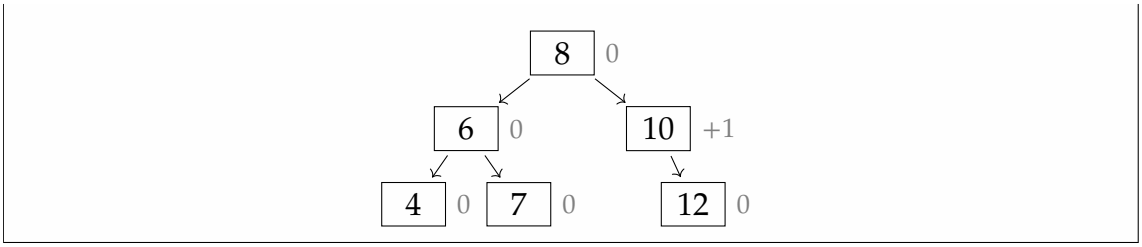
Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Baum.java`

(g) Fügen Sie jeweils den angegebenen Wert in den jeweils angegebenen AVL-Baum mit aufsteigender Sortierung ein und zeichnen Sie den resultierenden Baum vor möglicherweise erforderlichen Rotationen. Führen Sie danach bei Bedarf die erforderliche(n) Rotation(en) aus und zeichnen Sie dann den resultierenden Baum. Sollten keine Rotationen erforderlich sein, so geben Sie dies durch einen Text wie „keine Rotationen nötig” an.

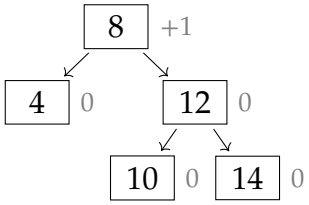
(i) Wert 7 einfügen



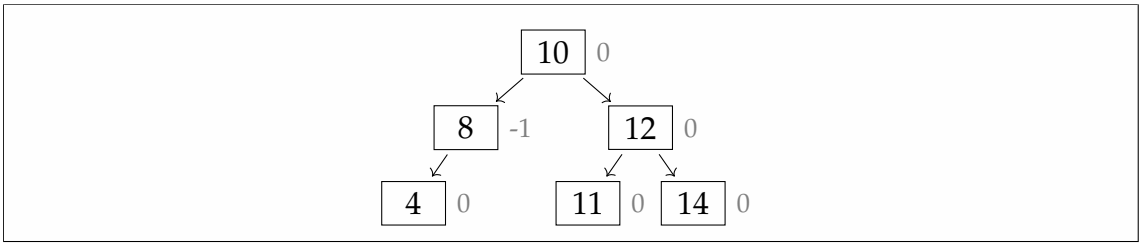
Lösungsvorschlag



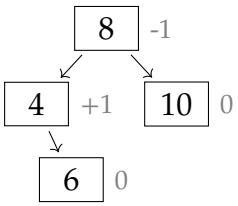
(ii) Wert 11 einfügen



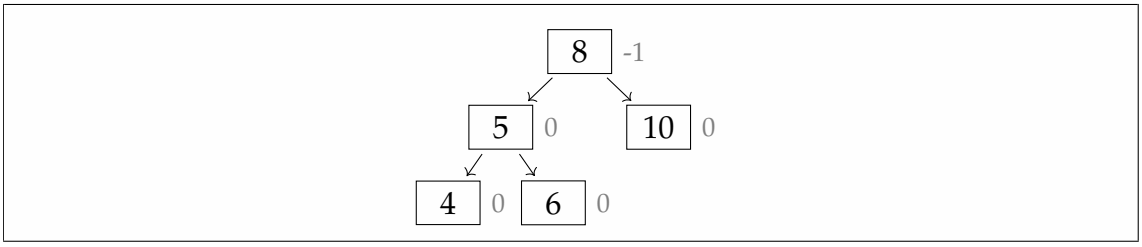
Lösungsvorschlag



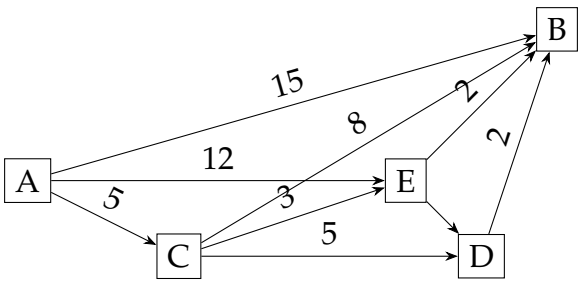
(iii) Wert 5 einfügen



Lösungsvorschlag



Aufgabe 3 [Graph A-E]



(a) Ermitteln Sie mit dem Algorithmus von Dijkstra den kürzesten Weg vom Knoten A zu allen erreichbaren Knoten in G. Verwenden Sie zur Lösung eine Tabelle der folgenden Form. Markieren Sie in jeder Zeile den jeweils als nächstes zu betrachtenden Knoten und führen Sie die Prioritätswarteschlange der noch zu betrachtenden Knoten (aufsteigend sortiert).

Nr.	**besucht**	**A**	**B**	**C**	**D**	**E**
0		0	∞	∞	∞	∞

Lösungsvorschlag

Nr.	**besucht**	**A**	**B**	**C**	**D**	**E**
0		0	∞	∞	∞	∞
1	A	**0**	15	5	∞	12
2	C	|	13	**5**	10	8
3	E	|	10	|	9	**8**
4	D	|	10	|	**9**	|
5	B	|	**10**	|	|	|

(b) Geben Sie den kürzesten Pfad vom Knoten A zum Knoten B an.

Lösungsvorschlag

A → C → E → B: 10

nach	**Entfernung**	**Reihenfolge**	**Pfad**
A → A	0	0	
A → B	10	5	A → C → E → B
A → C	5	2	A → C
A → D	9	4	A → C → E → D
A → E	8	3	A → C → E

Aufgabe 4 [O-Notation]

(a) Betrachten Sie das folgende Code-Beispiel (in Java-Notation):

```
int mystery(int n) {
  int a = 0, b = 0;
  int i = 0;
  while (i < n) {
    a = b + i;
    b = a;
    i = i + 1;
  }
  return a;
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery1.java`

Bestimmen Sie die asymptotische worst-case Laufzeit des Code-Beispiels in $\mathcal{O}$-Notation bezüglich der Problemgröße n. Begründen Sie Ihre Antwort.

Lösungsvorschlag

Die asymptotische worst-case Laufzeit des Code-Beispiels in $\mathcal{O}$-Notation ist $\mathcal{O}(n)$.

Die `while`-Schleife wird genau n mal ausgeführt. In der Schleife wird die Variable `i` in der Zeile `i = i + 1;` inkrementiert. `i` wird mit 0 initialisiert. Die `while`-Schleife endet, wenn `i` gleich groß ist als `n`.

(b) Betrachten Sie das folgende Code-Beispiel (in Java-Notation):

```java
int mystery(int n) {
  int r = 0;
  while (n > 0) {
    int y = n;
    int x = n;
    for (int i = 0; i < y; i++) {
      for (int j = 0; j < i; j++) {
        r = r + 1;
      }
      r = r - 1;
    }
    n = n - 1;
  }
  return r;
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/o_notation/Mystery2.java`

Bestimmen Sie für das Code-Beispiel die asymptotische worst-case Laufzeit in $\mathcal{O}$-Notation bezüglich der Problemgröße n. Begründen Sie Ihre Antwort.

Lösungsvorschlag

`while`: n-mal

1. `for`: $n, n-1, \ldots, 2, 1$

2. `for`: $1, 2, \ldots, n-1, n$

$n \times n \times n = \mathcal{O}(n^3)$

(c) Bestimmen Sie eine asymptotische Lösung (in Θ-Schreibweise) für die folgende Rekursionsgleichung:

$$T(n) = T\left(\frac{n}{2}\right) + \tfrac{1}{2}n^2 + n$$

Exkurs: Master-Theorem

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

$a =$ Anzahl der rekursiven Aufrufe, Anzahl der Unterprobleme in der Rekursion ($a \geq 1$).

$\frac{1}{b} =$ Teil des Originalproblems, welches wiederum durch alle Unterprobleme repräsentiert wird, Anteil an der Verkleinerung des Problems ($b > 1$).

$f(n) =$ Kosten (Aufwand, Nebenkosten), die durch die Division des Problems und die Kombination der Teillösungen entstehen. Eine von $T(n)$ unabhängige und nicht negative Funktion.

Dann gilt:

1. Fall: $T(n) \in \Theta\left(n^{\log_b a}\right)$

falls $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$ für $\varepsilon > 0$

2. Fall: $T(n) \in \Theta\left(n^{\log_b a} \cdot \log n\right)$

falls $f(n) \in \Theta\left(n^{\log_b a}\right)$

3. Fall: $T(n) \in \Theta(f(n))$

falls $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$ für $\varepsilon > 0$ und ebenfalls für ein c mit $0 < c < 1$ und alle hinreichend großen n gilt: $a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$

Lösungsvorschlag

Allgemeine Rekursionsgleichung:

$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$

Anzahl der rekursiven Aufrufe (a):

1

Anteil Verkleinerung des Problems (b):

um $\frac{1}{2}$ also $b = 2$

Laufzeit der rekursiven Funktion ($f(n)$):

$\frac{1}{2}n^2 + n$

Ergibt folgende Rekursionsgleichung:

$T(n) = 1 \cdot T\left(\frac{n}{2}\right) + \frac{1}{2}n^2 + n$

Nebenrechnung: $\log_b a = \log_2 1 = 0$

1. Fall: $f(n) \in \mathcal{O}\left(n^{\log_b a - \varepsilon}\right)$:

$\frac{1}{2}n^2 + n \notin \mathcal{O}\left(n^{-1}\right)$

2. Fall: $f(n) \in \Theta\left(n^{\log_b a}\right)$:

$\frac{1}{2}n^2 + n \notin \Theta(1)$

3. Fall: $f(n) \in \Omega\left(n^{\log_b a + \varepsilon}\right)$:

$\varepsilon = 2$
$\frac{1}{2}n^2 + n \in \Omega(n^2)$
Für eine Abschätzung suchen wir eine Konstante, damit gilt:
$1 \cdot f(\frac{n}{2}) \leq c \cdot f(n)$
$\frac{1}{2} \cdot \frac{1}{4}n^2 + \frac{1}{2}n \leq c \cdot (\frac{1}{2} \cdot n^2 + n)$
Damit folgt $c = \frac{1}{4}$
und $0 < c < 1$

$\Rightarrow \Theta(\frac{1}{2}n^2 + n)$

$\Rightarrow \Theta(n^2)$

Berechne die Rekursionsgleichung auf WolframAlpha: WolframAlpha

Aufgabe 5 [Streuspeicherung]

Gegeben seien die folgenden Schlüssel k zusammen mit ihren Streuwerten $h(k)$:

k	B	Y	E	!	A	U	D	?
$h(k)$	5	4	0	4	4	0	7	2

(a) Fügen Sie die Schlüssel in der angegebenen Reihenfolge (von links nach rechts) in eine Streutabelle der Größe 8 ein und lösen Sie Kollisionen durch verkettete Listen auf.

Stellen Sie die Streutabelle in folgender Art und Weise dar:

Lösungsvorschlag

Fach	Schlüssel k (verkettete Liste, zuletzt eingetragener Schlüssel rechts)
0	E, U,
1	
2	?
3	
4	Y, !, A
5	B,
6	
7	D

(b) Fügen Sie die gleichen Schlüssel in der gleichen Reihenfolge und mit der gleichen Streufunktion in eine neue Streutabelle der Größe 8 ein. Lösen Sie Kollisionen diesmal aber durch lineares Sondieren mit Schrittweite $+1$ auf.

Geben Sie für jeden Schlüssel jeweils an, welche Fächer Sie in welcher Reihenfolge sondiert haben und wo der Schlüssel schlussendlich gespeichert wird.

Fach	Schlüssel k
0	E
1	U
2	D
3	?
4	Y
5	B
6	!
7	A

Schlüssel	Sondierung	Speicherung
B		5
Y		4
E		0
!	4, 5	6
A	4, 5, 6	7
U	0	1
D	7, 0, 1	2
?	2	3

(c) Bei der doppelten Streuadressierung verwendet man eine Funktionsschar h_i, die sich aus einer primären Streufunktion h_0 und einer Folge von sekundären Streufunktionen $h_1, h_2, \ldots$ zusammensetzt. Die folgenden Werte der Streufunktionen sind gegeben:

k	B	Y	E	!	A	U	D	?
$h_0(k)$	5	4	0	4	4	0	7	2
$h_1(k)$	6	3	3	3	1	2	6	0
$h_2(k)$	7	2	6	2	6	4	5	6
$h_3(k)$	0	1	1	1	3	6	4	4

Fügen Sie die Schlüssel in der angegebenen Reihenfolge (von links nach rechts) in eine Streutabelle der Größe 8 ein und geben Sie für jeden Schlüssel jeweils an, welche Streufunktion h_i zur letztendlichen Einsortierung verwendet wurde.

Fach	Schlüssel k
0	E
1	A
2	U
3	!
4	Y
5	B
6	?
7	D

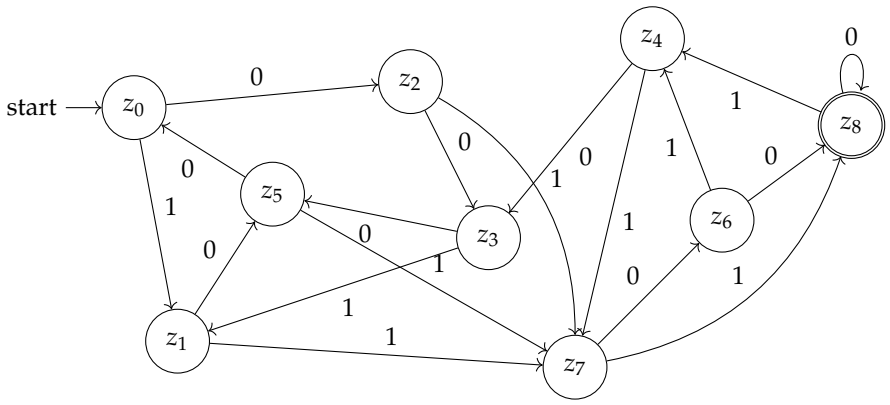
Schlüssel	Streufunktion
B	$h_0(k)$
Y	$h_0(k)$
E	$h_0(k)$
!	$h_1(k)$
A	$h_1(k)$
U	$h_1(k)$
D	$h_0(k)$
?	$h_2(k)$

Thema Nr. 2

Teilaufgabe Nr. 1

Aufgabe 1 [Minimierungsalgorithmus]

(a) Geben Sie einen deterministischen endlichen Automaten (DEA) mit minimaler Anzahl an Zuständen an, der dieselbe Sprache akzeptiert wie folgender deterministischer endlicher Automat. Dokumentieren Sie Ihr Vorgehen geeignet.



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/Aj5aei652

Lösungsvorschlag

Minimierungstabelle (Table filling)

— Der Minimierungs-Algorithmus (auch Table-Filling-Algorithmus genannt) trägt in seinem Verlauf eine Markierung in alle diejenigen Zellen der Tabelle ein, die zueinander nicht äquivalente Zustände bezeichnen. Die Markierung „x_n“ in einer Tabellenzelle (i, j) bedeutet dabei, dass das Zustandspaar (i, j) in der k-ten Iteration des Algorithmus markiert wurde und die Zustände i und j somit zueinander $(k-1)$-äquivalent, aber nicht k-äquivalent und somit insbesondere nicht äquivalent sind. Bleibt eine Zelle bis zum Ende unmarkiert, sind die entsprechenden Zustände zueinander äquivalent. —

z_0	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
z_1	x_3	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
z_2	x_3	x_4	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
z_3		x_3	x_3	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
z_4	x_3	x_4		x_3	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
z_5	x_3	x_4		x_3		$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$
z_6	x_2	x_2	x_2	x_2	x_2	x_2	$\varnothing$	$\varnothing$	$\varnothing$
z_7	x_2	x_2	x_2	x_2	x_2	x_2	x_2	$\varnothing$	$\varnothing$
z_8	x_1	x_1	x_1	x_1	x_1	x_1	x_1	x_1	$\varnothing$
	z_0	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8

x_1 Paar aus End-/ Nicht-Endzustand kann nicht äquivalent sein.

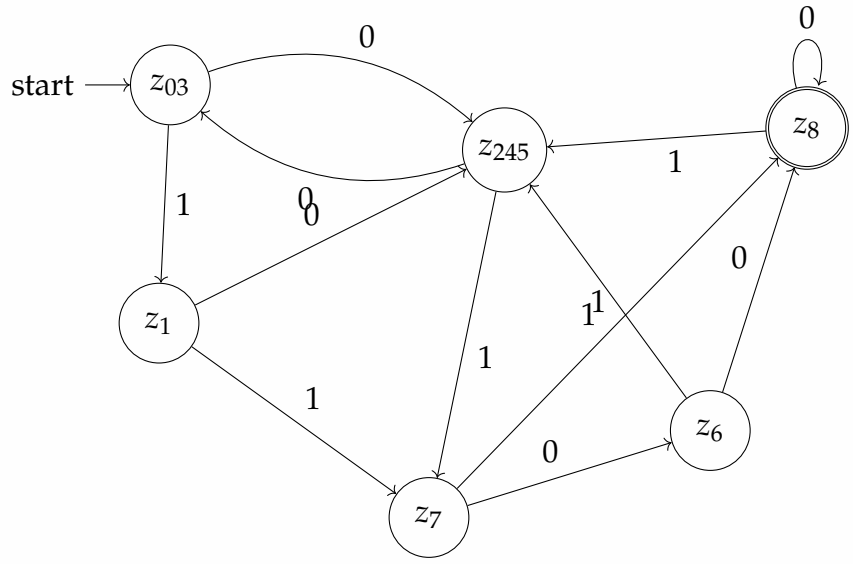
x_2 Test, ob man mit der Eingabe zu einem bereits markiertem Paar kommt.

x_3 In weiteren Iterationen markierte Zustände.

x_4 ...

Übergangstabelle

Zustandspaar	0	1
(z_0, z_1)	(z_2, z_5)	$(z_1, z_7)\ ^{x_3}\ x_3$
(z_0, z_2)	(z_2, z_3)	$(z_1, z_7)\ x_3$
(z_0, z_3)	(z_2, z_5)	(z_1, z_1)
(z_0, z_4)	(z_2, z_3)	$(z_1, z_7)\ x_3$
(z_0, z_5)	(z_2, z_0)	$(z_1, z_7)\ x_3$
(z_0, z_6)	(z_2, z_8)	$(z_1, z_4)\ x_2$
(z_0, z_7)	(z_2, z_6)	$(z_1, z_8)\ x_2$
(z_1, z_2)	(z_5, z_3)	$(z_7, z_7)\ x_4$
(z_1, z_3)	(z_5, z_5)	$(z_7, z_1)\ x_3$
(z_1, z_4)	(z_5, z_3)	$(z_7, z_7)\ x_4$
(z_1, z_5)	(z_5, z_0)	$(z_7, z_7)\ x_4$
(z_1, z_6)	(z_5, z_8)	$(z_7, z_4)\ x_2$
(z_1, z_7)	(z_5, z_6)	$(z_7, z_8)\ x_2$
(z_2, z_3)	(z_3, z_5)	$(z_7, z_1)\ x_3$
(z_2, z_4)	(z_3, z_3)	(z_7, z_7)
(z_2, z_5)	(z_3, z_0)	(z_7, z_7)
(z_2, z_6)	(z_3, z_8)	$(z_7, z_4)\ x_2$
(z_2, z_7)	(z_3, z_6)	$(z_7, z_8)\ x_2$
(z_3, z_4)	(z_5, z_3)	$(z_1, z_7)\ x_3$
(z_3, z_5)	(z_5, z_0)	$(z_1, z_7)\ x_3$
(z_3, z_6)	(z_5, z_8)	$(z_1, z_4)\ x_2$
(z_3, z_7)	(z_5, z_6)	$(z_1, z_8)\ x_2$
(z_4, z_5)	(z_3, z_0)	(z_7, z_7)
(z_4, z_6)	(z_3, z_8)	$(z_7, z_4)\ x_2$
(z_4, z_7)	(z_3, z_6)	$(z_7, z_8)\ x_2$
(z_5, z_6)	(z_0, z_8)	$(z_7, z_4)\ x_2$
(z_5, z_7)	(z_0, z_6)	$(z_7, z_8)\ x_2$
(z_6, z_7)	(z_8, z_6)	$(z_4, z_8)\ x_2$



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/Aro484bz2

(b) Beweisen oder widerlegen Sie für folgende Sprachen über dem Alphabet $\Sigma = \{a, b, c\}$, dass sie regulär sind.

(i) $L_1 = \{ a^i cub^j vac^k \mid u, v \in \{a, b\}^* \text{ und } i, j, k \in \mathbb{N}_0 \}$

Lösungsvorschlag

Die Sprache L_1 ist regulär. Nachweis durch regulären Ausdruck:

$$a^*c(a|b)^*b^*(a|b)^*ac^*$$

(ii) $L_2 = \{ a^i cub^j vac^k \mid u, v \in \{a, b\}^* \text{ und } i, j, k \in \mathbb{N}_0 \text{ mit } k = i + j \}$

Lösungsvorschlag

Die Sprache L_2 ist nicht regulär. Widerlegung durch das Pumping-Lemma.
TODO

(c) Sei L eine reguläre Sprache über dem Alphabet Σ. Für ein festes Element $a \in \Sigma$ betrachten wir die Sprache $L_a = \{ aw \mid w \in \Sigma^*, wa \in L \}$. Zeigen Sie, dass L_a regulär ist.

Lösungsvorschlag

Die regulären Sprachen sind unter dem Komplement abgeschlossen.

Aufgabe 2 [Kontextfreie Sprachen]

(a) Sei $L = \{ 0^n 1^m 1^p 0^q \mid n + m = p + q \text{ und } n, m, p, q \in \mathbb{N}_0 \}$. Geben Sie eine kontextfreie Grammatik für L an. Sie dürfen dabei ε-Produktionen der Form $\{A \to \varepsilon\}$ verwenden.

Lösungsvorschlag

$$
\mathrm{P} = \Big\{
\begin{aligned}
S &\to 0S0 \mid 0A0 \mid 0B0 \mid \varepsilon \mid A \mid B \mid C \\
A &\to 0A1 \mid 0C1 \\
B &\to 1B0 \mid 1C0 \\
C &\to 1C1 \mid \varepsilon
\end{aligned}
\Big\}
$$

(b) Für eine Sprache L sei $L^r = \{ x^r \mid x \in L \}$ die Umkehrsprache. Dabei bezeichne x^r das Wort, das aus r entsteht, indem man die Reihenfolge der Zeichen umkehrt, beispielsweise $(abb)^r = bba$.

(i) Sei L eine kontextfreie Sprache. Zeigen Sie, dass dann auch L^r kontextfrei ist.

(ii) Geben Sie eine kontextfreie Sprache L_1, an, sodass $L_1 \cap L_1^r$ kontextfrei ist.

(iii) Geben Sie eine kontextfreie Sprache L_2, an, sodass $L_2 \cap L_2^r$ nicht kontextfrei ist.

Teilaufgabe Nr. 2

Aufgabe 1 [Vornamen]

Eine Hashfunktion h wird verwendet, um Vornamen auf die Buchstaben $\{A, \ldots, Z\}$ abzubilden. Dabei bildet h auf den ersten Buchstaben des Vornamens als Hashwert ab.

Sei H die folgende Hashtabelle (Ausgangszustand):

Schlüssel	Inhalt
A	
B	
C	
D	Dirk
E	
F	
G	
H	
I	Inge
J	
K	Kurt
L	
M	

Schlüssel	Inhalt
N	
O	
P	
Q	
R	
S	
T	
U	
V	
W	
X	
Y	
Z	

(a) Fügen Sie der Hashtabelle H die Vornamen

Martin, Michael, Norbert, Matthias, Alfons, Bert, Christel, Adalbert, Edith, Emil

in dieser Reihenfolge hinzu, wobei Sie Kollisionen durch lineares Sondieren (mit Inkrementieren zum nächsten Buchstaben) behandeln.

Lösungsvorschlag

Schlüssel	Inhalt
A	Alfons
B	Bert
C	Christel
D	Dirk
E	Adalbert
F	Edith
G	Emil
H	
I	Inge
J	
K	Kurt
L	
M	Martin

Schlüssel	Inhalt
N	Michael
O	Norbert
P	Matthias
Q	
R	
S	
T	
U	
V	
W	
X	
Y	
Z	

(b) Fügen Sie der Hashtabelle H die Vornamen

Brigitte, Elmar, Thomas, Katrin, Diana, Nathan, Emanuel, Sebastian, Torsten, Karolin

in dieser Reihenfolge hinzu, wobei Sie Kollisionen durch Verkettung der Überläufer behandeln. (Hinweis: Verwenden Sie die Hashtabelle im Ausgangszustand.)

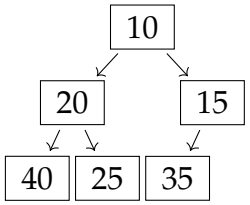
Lösungsvorschlag

Schlüssel	Inhalt
A	
B	Brigitte
C	
D	Dirk, Diana
E	Elmar, Emanuel
F	
G	
H	
I	Inge
J	
K	Kurt, Katrin, Karolin
L	
M	

Schlüssel	Inhalt
N	Nathan
O	
P	
Q	
R	
S	Sebastian
T	Thomas, Torsten
U	
V	
W	
X	
Y	
Z	

Aufgabe 3 [DeleteMin]

Es sei der folgende Min-Heap gegeben:



(a) Geben Sie obigen Min-Heap in der Darstellung eines Feldes an, wobei die Knoten in Level-Order abgelegt sind.

Lösungsvorschlag

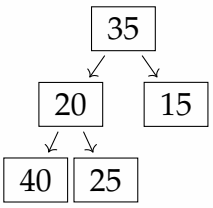
0	**1**	**2**	**3**	**4**	**5**
10	20	15	40	25	35

(b) Führen Sie wiederholt DeleteMin-Operationen auf dem gegebenen Heap aus, bis der Heap leer ist. Zeichnen Sie dafür den aktuellen Zustand des Heaps als Baum und als Feld nach jeder Änderung des Heaps, wobei Sie nur gültige Bäume zeichnen (d. h. solche die keine Lücken haben). Dokumentieren Sie, was in den einzelnen Schritten geschieht.

Löschen von 10

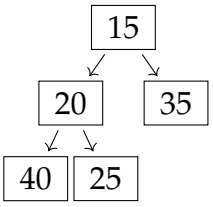
Nach dem Ersetzen von „10“ durch „35“:

0	**1**	**2**	**3**	**4**
35	20	15	40	25



Nach dem Vertauschen von „35“ und „15“:

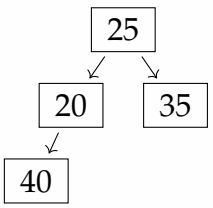
0	**1**	**2**	**3**	**4**
15	20	35	40	25



Löschen von 15

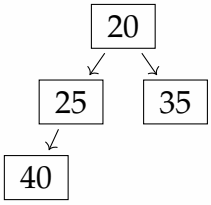
Nach dem Ersetzen von „15“ mit „25“:

0	**1**	**2**	**3**
25	20	35	40



Nach dem Vertauschen von „25“ und „20“:

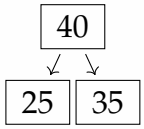
0	1	2	3
20	25	35	40



Löschen von 20

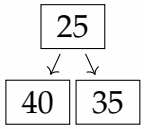
Nach dem Vertauschen von „20“ mit „40“:

0	1	2
40	25	35



Nach dem Vertauschen von „40“ und „25“:

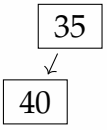
0	1	2
25	40	35



Löschen von 25

Nach dem Ersetzen von „25“ durch „35“:

0	1
35	40



Löschen von 35

Nach dem Ersetzen von „35“ mit „40“:

0

40

40

Löschen von 40

Nach dem Löschen von „40“:

Aufgabe 4 [Gutschein]

Das GUTSCHEIN-Problem ist gegeben durch eine Folge $w_1, \ldots, w_n$ von Warenwerten (wobei $w \in N_0$ für $i = 1, \ldots, n$) und einem Gutscheinbetrag $G \in \mathbb{N}_0$.

Da Gutscheine nicht in Bargeld ausgezahlt werden können, ist die Frage, ob man eine Teilfolge der Waren findet, sodass man genau den Gutschein ausnutzt. Formal ist dies die Frage, ob es eine Menge von Indizes I mit $I \subseteq \{1, \ldots, n\}$ gibt, sodass $\sum_{i \in I} w_i = G$

Exkurs: Teilsummenproblem
Das **Teilsummenproblem** (SUBSET SUM oder SSP) ist ein spezielles Rucksackproblem. Gegeben sei eine Menge von ganzen Zahlen $I = \{w_1, w_2, \ldots, w_n\}$. Gesucht ist eine Untermenge, deren Elementsumme maximal, aber nicht größer als eine gegebene obere Schranke c ist.

(a) Sei $w_1 = 10, w_2 = 30, w_3 = 40, w_4 = 20, w_5 = 15$ eine Folge von Warenwerten.

(i) Geben Sie einen Gutscheinbetrag $40 < G < 115$ an, sodass die GUTSCHEIN-Instanz eine Lösung hat. Geben Sie auch die lösende Menge $I \subseteq \{1, 2, 3, 4, 5\}$ von Indizes an.

Lösungsvorschlag

50
$I = \{1, 3\}$

(ii) Geben Sie einen Gutscheinbetrag G mit $40 < G < 115$ an, sodass die GUTSCHEIN-Instanz keine Lösung hat.

Lösungsvorschlag

51

(b) Sei *table* eine $(n \times (G+1))$-Tabelle mit Einträgen table$[i,k]$, für $1 \leq i \leq n$ und $0 \leq k \leq G$, sodass

$$\text{table}[i,k] = \begin{cases} \textbf{true} & \text{falls es } I \subseteq \{1,\ldots,n\} \text{ mit } \sum_{i \in I} w_i = G \text{ gibt} \\ \textbf{false} & \text{sonst} \end{cases}$$

Geben Sie einen Algorithmus in Pseudo-Code oder Java an, der die Tabelle *table* mit *dynamischer Programmierung* in Worst-Case-Laufzeit $\mathcal{O}(n \times G))$ erzeugt. Begründen Sie die Korrektheit und die Laufzeit Ihres Algorithmus. Welcher Eintrag in *table* löst das GUTSCHEIN-Problem?

Lösungsvorschlag

Algorithmus 1: Gutschein-Problem

table = boolean array $n+1 \times (G+1)$; // Initialisiere ein boolsches Feld mit $n+1$ Zeilen für jeden Warenwert und 0 für keinen Warenwert und mit $G+1$ Spalten für alle Gutscheinbetrag bis G und 0 für keinen Gutscheinbetrag

for *k in* $1 \ldots G$ **do** ; // Wenn der Gutscheinbetrag größer als 0 ist und es keine Warenwerte gibt, ð$n = 0$, kann der Gutschein nicht eingelöst werden.

  table$[0][k] =$ false
end

for *i in* $0 \ldots n$ **do** ; // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst werden.

  table$[i][0] =$ true
end
for *i in* $1 \ldots n$ **do** ; // Durchlaufe jede Zeile der Warenwerte

  for *k in* $1 \ldots G$ **do** ; // Durchlaufe jede Spalte der Gutscheinbeträge in dieser Zeile

    table$[i][k] =$ table$[i-1][k]$; // Übernehme erstmals das Ergebnis der Zelle der vorhergehenden Zeile in der gleichen Spalte
    if $k \geq w_i$ *und table*$[i][k]$ *noch nicht markiert* **then** ; // Wenn der aktuelle Gutscheinbetrag größer als der aktuelle Warenwert und die aktuelle Zelle noch nicht als wahr markiert ist

      table$[i][k] =$ table$[i-1][k-w_i]$; // übernimmt das Ergebnis des aktuellen Gutscheinbetrags minus des aktuellen Warenwerts
      ;
    end
  end
end

```
/**
 * Nach <a href="https://www.geeksforgeeks.org/subset-sub-problem-dp-25"> Subset
```

```java
 * Sum Problem auf geeksforgeeks.org</a>
 */
public class Gutschein {
  /**
   * @param G Die Indizes der GUTSCHEIN-Beträge.
   *
   * @param W Das GUTSCHEIN-Problem ist gegeben durch eine Folge w1, ..., wn von
   *          Warenwerten.
   *
   * @return Wahr, wenn der Gutscheinbetrag vollständig in Warenwerten eingelöst
   *         werden kann, falsch wenn der Betrag nicht vollständig eingelöst
   *         werden kann.
   */
  public static boolean gutscheinDP(int G, int W[]) {
    // Der Eintrag in der Tabelle tabelle[i][k] ist wahr,
    // wenn es eine Teilsumme der
    // W[0..i-1] gibt, die gleich k ist.
    int n = W.length;
    boolean table[][] = new boolean[n + 1][G + 1];

    // Wenn der Gutschein-Betrag größer als 0 ist und es keine
    // Warenwerte (n = 0) gibt, kann der Gutschein nicht eingelöst
    // werden.
    for (int k = 1; k <= G; k++) {
      table[0][k] = false;
    }

    // Ist der Gutscheinbetrag 0, dann kann er immer eingelöst werden.
    for (int i = 0; i <= n; i++) {
      table[i][0] = true;
    }

    for (int i = 1; i <= n; i++) {
      for (int k = 1; k <= G; k++) {
        table[i][k] = table[i - 1][k];
        // Warenwert
        int w = W[i - 1];
        if (k >= w && !table[i][k]) {
          table[i][k] = table[i - 1][k - w];
        }
      }
    }
    return table[n][G];
  }

  public static void main(String[] args) {
    System.out.println(gutscheinDP(50, new int[] { 10, 30, 40, 20, 15 }));
    System.out.println(gutscheinDP(41, new int[] { 10, 30, 40, 20, 15 }));

    System.out.println(gutscheinDP(3, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(5, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(6, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(2, new int[] { 1, 2, 3 }));
    System.out.println(gutscheinDP(1, new int[] { 1, 2, 3 }));
```

```
    System.out.println(gutscheinDP(7, new int[] { 1, 2, 3 }));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/Gutschein.java`

Die äußere for-Schleife läuft n mal und die innere for-Schleife G mal.

Der letzte Eintrag in der Tabelle, also der Wert in der Zelle `table[W.length][G]`, löst das Gutscheinproblem. Steht hier `true`, dann gibt es eine Teilfolge der Waren, die den Gutscheinbetrag genau ausnutzt.

Aufgabe 5 [Schnelle Suche von Schlüsseln: odd-ascending-even-descending-Folge]

Eine Folge von Zahlen ist eine *odd-ascending-even-descending*-Folge, wenn gilt:

> Zunächst enthält die Folge alle Schlüssel, die *ungerade* Zahlen sind, und diese Schlüssel sind aufsteigend sortiert angeordnet. Im Anschluss daran enthält die Folge alle Schlüssel, die *gerade* Zahlen sind, und diese Schlüssel sind absteigend sortiert angeordnet.

(a) Geben Sie die Zahlen $10, 3, 11, 20, 8, 4, 9$ als *odd-ascending-even-descending*-Folge an.

Lösungsvorschlag

$3, 9, 11, 20, 10, 8, 4$

(b) Geben Sie einen Algorithmus (z. B. in Pseudocode oder Java) an, der für eine *odd-ascending-even-descending*-Folge F gegeben als Feld und einem Schlüsselwert S prüft, ob S in F vorkommt und `true` im Erfolgsfall und ansonsten `false` liefert. Dabei soll der Algorithmus im Worst-Case eine echt bessere Laufzeit als Linearzeit (in der Größe der Arrays) haben. Erläutern Sie Ihren Algorithmus und begründen Sie die Korrektheit.

Bei dem Algorithmus handelt es sich um einen leicht abgewandelten Code, der eine „klassische“ binären Suche implementiert.

```
  public static boolean suche(int[] feld, int schlüssel) {
    int links = 0, rechts = feld.length - 1;
    boolean istGerade = schlüssel % 2 == 0;
    while (links <= rechts) {
      int mitte = links + (rechts - links) / 2;
      if (feld[mitte] == schlüssel) {
        return true;
      }
      // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
      // Zahl gerade ist und die Zahl in der Mitte größer als die
      // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
      // und die Zahl in der Mitte kleiner.
      if ((istGerade && feld[mitte] > schlüssel) || (!istGerade && feld[mitte] <
↪   schlüssel)) {
        // nach rechts verschieben
        links = mitte + 1;
```

```
      } else {
        // nach links verschieben
        rechts = mitte - 1;
      }
    }
    return false;
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGerade.java`

(c) Erläutern Sie schrittweise den Ablauf Ihres Algorithmus für die Folge 1, 5, 11, 8, 4, 2 und den Suchschlüssel 4.

Lösungsvorschlag

Die erste Zeile der Methode `suche` initialisiert die Variable `links` mit 0 und `rechts` mit 5. Da `links` kleiner ist als `rechts`, wird die `while`-Schleife betreten und die Variable `mitte` auf 2 gesetzt. Da der gesuchte Schlüssel gerade ist und `feld[2]` 11 ist, also größer, wird in den `true`-Block der `if`-Bedingung besprungen und die Variable `links` aus 3 gesetzt.

Zu Beginn des 2. Durchlaufs der `while`-Schleife ergeben sich folgende Werte: `links`: 3 `mitte`: 4 `rechts`: 5.

In der anschließenden Bedingten Anweisung wird die `while`-Schleife verlassen und `true` zurückgegeben, da mit `feld[4]` der gewünschte Schlüssel gefunden wurde.

(d) Analysieren Sie die Laufzeit Ihres Algorithmus für den Worst-Case, geben Sie diese in $\mathcal{O}$-Notation an und begründen Sie diese.

Lösungsvorschlag

Die Laufzeit des Algorithmuses ist in der Ordnung $\mathcal{O}(\log_2 n)$.

Im schlechtesten Fall muss nicht die gesamte Folge durchsucht werden. Nach dem ersten Teilen der Folge bleiben nur noch $\frac{n}{2}$ Elemente, nach dem zweiten Schritt $\frac{n}{4}$, nach dem dritten $\frac{n}{8}$ usw. Allgemein bedeutet dies, dass im i-ten Durchlauf maximal $\frac{n}{2^i}$ Elemente zu durchsuchen sind. Entsprechend werden $\log_2 n$ Schritte benötigt.

Kompletter Code

```
public class UngeradeGerade {

  public static boolean suche(int[] feld, int schlüssel) {
    int links = 0, rechts = feld.length - 1;
    boolean istGerade = schlüssel % 2 == 0;
    while (links <= rechts) {
      int mitte = links + (rechts - links) / 2;
      if (feld[mitte] == schlüssel) {
        return true;
      }
      // Verschiebe die linke Grenze nach rechts, wenn die gesuchte
```

```
      // Zahl gerade ist und die Zahl in der Mitte größer als die
      // gesuchte Zahl ist oder wenn die gesuchte Zahl ungerade ist
      // und die Zahl in der Mitte kleiner.
      if ((istGerade && feld[mitte] > schlüssel) || (!istGerade && feld[mitte] <
↪   schlüssel)) {
        // nach rechts verschieben
        links = mitte + 1;
      } else {
        // nach links verschieben
        rechts = mitte - 1;
      }
    }
    return false;
  }

  public static void main(String[] args) {
    System.out.println(suche(new int[] { 1, 5, 11, 8, 4, 2 }, 4));
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGerade.java`

Test-Code

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;

public class UngeradeGeradeTest {

  private void assertSucheUnGerade(int[] feld, int suche, boolean ergebnis) {
    assertEquals(ergebnis, UngeradeGerade.suche(feld, suche));
  }

  @Test
  public void assertSucheUnGerade() {
    int[] feld = new int[] { 1, 3, 5, 7, 9, 10, 8, 6, 4, 2 };
    assertSucheUnGerade(feld, 4, true);
    assertSucheUnGerade(feld, 11, false);
    assertSucheUnGerade(feld, 0, false);
    assertSucheUnGerade(feld, 3, true);
  }

}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/examen/examen_66115/jahr_2020/herbst/UngeradeGeradeTest.java`