Thema Nr. 1

Teilaufgabe Nr. 1

Aufgabe 1

Im Folgenden bezeichnet $a^i = a \dots a$ und ε steht für das leere Wort (ðinsbesondere $a^i = \varepsilon$).
Die Menge $\mathbb{N}_0 = \{0, 1, 2, \dots\}$ ist die Menge aller nicht-negativer Ganzzahlen.
Die Sprachen $L_1, \dots, L_{12}$ seien definiert als:

(a) Ordnen Sie jedem der folgenden nichtdeterministischen endlichen Automaten $N_j, j = 1, \dots, 6$, (die alle über dem Alphabet $\Sigma = \{a\}$ arbeiten) **jeweils eine** der Sprachen $L_i \in \{L_1, \dots, L_{12}\}$ zu, sodass L_i, genau die von N_i, **akzeptierte Sprache** ist.

- $N_1 = L_6$ (mindestens ein a)
- $N_2 = L_8$ (ungerade Anzahl an a's: $1, 5, 7, \dots$)
- $N_3 = L_2$ (gerade Anzahl an a's: $2, 4, 6, \dots$)
- $N_4 = L_{12}$ (leeres Wort)
- $N_5 = L_8$ (ungerade Anzahl an a's: $1, 5, 7, \dots$)
- $N_6 = L_{11}$ (die Sprache akzeptiert nicht)

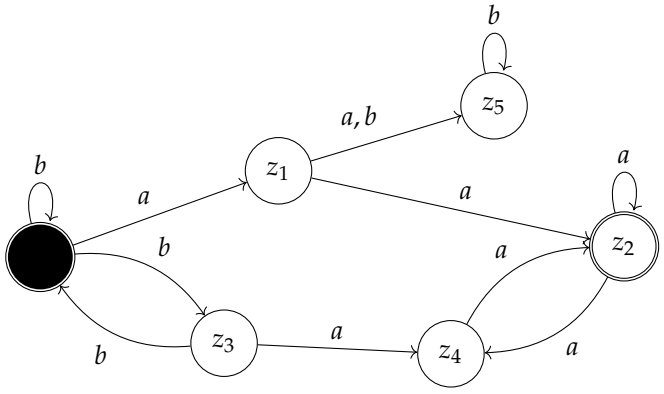
(b) Zeigen Sie für eine der Sprachen $L_1, \dots, L_{12}$ dass diese **nicht regulär** ist.

$L_10 = \{ a^n \mid n \in \mathbb{N}_0, n \text{ ist Primzahl} \}$

ist nicht regulär, da sich sonst jede Primzahl p einer bestimmten Mindestgröße j als Summe von natürlichen Zahlen $u + v + w$ darstellen ließe, so dass $v \geq 1$ und für alle $i \geq 0$ auch $u + iv + w = p + (i1)v$ prim ist. Dies ist jedoch für $i = p + 1$ wegen $p + (p + 11)v = p(1 + v)$ nicht der Fall. [a]

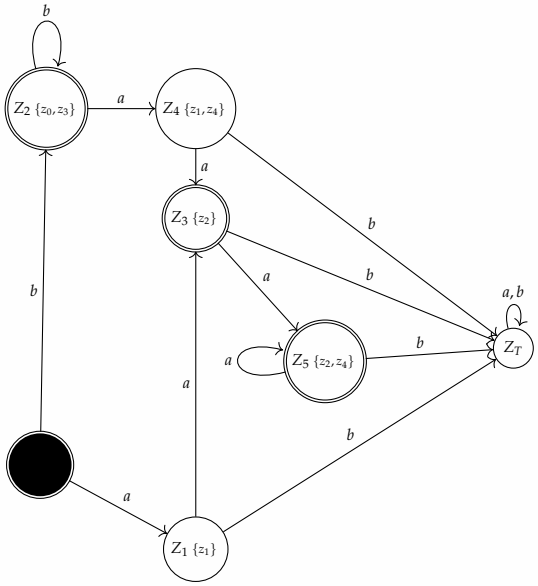
[a] `https://www.informatik.hu-berlin.de/de/forschung/gebiete/algorithmenII/Lehre/ws13/einftheo/einftheo-skript.pdf`

(c) Konstruieren Sie für den folgenden nichtdeterministischen endlichen Automaten (der Worte über dem Alphabet $\Sigma = \{a, b\}$ verarbeitet) einen äquivalenten deterministischen endlichen Automaten mithilfe der Potenzmengenkonstruktion. Zeichnen Sie dabei nur die vom Startzustand erreichbaren Zustände. Erläutern Sie Ihr Vorgehen.



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/Af7iooyca

Zustandsmenge	Eingabe a	Eingabe b
$Z_0\ \{z_0\}$	$Z_1\ \{z_1\}$	$Z_2\ \{z_0, z_3\}$
$Z_1\ \{z_1\}$	$Z_3\ \{z_2\}$	Z_T
$Z_2\ \{z_0, z_3\}$	$Z_4\ \{z_1, z_4\}$	$Z_2\ \{z_0, z_3\}$
$Z_3\ \{z_2\}$	$Z_5\ \{z_2, z_4\}$	Z_T
$Z_4\ \{z_1, z_4\}$	$Z_3\ \{z_2\}$	Z_T
$Z_5\ \{z_2, z_4\}$	$Z_5\ \{z_2, z_4\}$	Z_T



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/Apkyuo4ja

Aufgabe 2

Sei $G = (V, \Sigma, P, S)$ eine kontextfreie Grammatik mit Variablen $V = \{S, A, B, C, D\}$, Terminalzeichen $\Sigma = \{a, b, c\}$, Produktionen

$$
\mathrm{P} = \Big\{
\begin{aligned}
S &\to AD \mid CC \mid c \\
A &\to a \\
B &\to b \\
C &\to CC \mid c \\
D &\to SB \mid CB
\end{aligned}
\Big\}
$$

und Startsymbol S. Führen Sie den Algorithmus von Cocke, Younger und Kasami (CYK-Algorithmus) für G und das Wort *aaaccbbb* aus. Liegt *aaaccbbb* in der durch G erzeugten Sprache? Erläutern Sie Ihr Vorgehen und den Ablauf des CYK-Algorithmus.

a	a	a	c	c	b	b	b
-	-	-	S,C	D,D	-	-	
-	-	-	D,D	-	-		
-	-	S,S	-	-			
-	-	D,D	-				
-	S,S	-					
-	D,D						
S,S							

Das Wort *aaaccbbb* liegt in der Sprache.

Teilaufgabe Nr. 2

Aufgabe 1

(a) Geben Sie für folgende Sortierverfahren jeweils zwei Felder A und B an, so dass das jeweilige Sortierverfahren angewendet auf A seine Best-Case-Laufzeit und angewendet auf B seine Worst-Case-Laufzeit erreicht. (Wir messen die Laufzeit durch die Anzahl der Vergleiche zwischen Elementen der Eingabe.) Dabei soll das Feld A die Zahlen $1, 2, \ldots, 7$ genau einmal enthalten; das Feld B ebenso. Sie bestimmen also nur die Reihenfolge der Zahlen.

Wenden Sie als Beleg für Ihre Aussagen das jeweilige Sortierverfahren auf die Felder A und B an und geben Sie nach jedem größeren Schritt des Algorithmus den Inhalt der Felder an.

Geben Sie außerdem für jedes Verfahren asymptotische Best- und Worst-Case-Laufzeit für ein Feld der Länge n an.

Die im Pseudocode verwendete Unterroutine Swap(A, i, j) vertauscht im Feld A die jeweiligen Elemente mit den Indizes i und j miteinander.

(i) **Insertionsort**

Best-Case

1	2	3	4	5	6	7

```
 1  2  3  4  5  6  7  Eingabe
 1  2* 3  4  5  6  7  markiere (i 1)
 1  2  3* 4  5  6  7  markiere (i 2)
 1  2  3  4* 5  6  7  markiere (i 3)
 1  2  3  4  5* 6  7  markiere (i 4)
 1  2  3  4  5  6* 7  markiere (i 5)
 1  2  3  4  5  6  7* markiere (i 6)
 1  2  3  4  5  6  7  Ausgabe
```

Worst-Case

7	6	5	4	3	2	1

```
 7  6  5  4  3  2  1  Eingabe
 7  6* 5  4  3  2  1  markiere (i 1)
>7  7< 5  4  3  2  1  vertausche (i 0<>1)
 6  7  5* 4  3  2  1  markiere (i 2)
```

```
 6 >7  7< 4  3  2  1  vertausche (i 1<>2)
>6  6< 7  4  3  2  1  vertausche (i 0<>1)
 5  6  7  4* 3  2  1  markiere (i 3)
 5  6 >7  7< 3  2  1  vertausche (i 2<>3)
 5 >6  6< 7  3  2  1  vertausche (i 1<>2)
>5  5< 6  7  3  2  1  vertausche (i 0<>1)
 4  5  6  7  3* 2  1  markiere (i 4)
 4  5  6 >7  7< 2  1  vertausche (i 3<>4)
 4  5 >6  6< 7  2  1  vertausche (i 2<>3)
 4 >5  5< 6  7  2  1  vertausche (i 1<>2)
>4  4< 5  6  7  2  1  vertausche (i 0<>1)
 3  4  5  6  7  2* 1  markiere (i 5)
 3  4  5  6 >7  7< 1  vertausche (i 4<>5)
 3  4  5 >6  6< 7  1  vertausche (i 3<>4)
 3  4 >5  5< 6  7  1  vertausche (i 2<>3)
 3 >4  4< 5  6  7  1  vertausche (i 1<>2)
>3  3< 4  5  6  7  1  vertausche (i 0<>1)
 2  3  4  5  6  7  1* markiere (i 6)
 2  3  4  5  6 >7  7< vertausche (i 5<>6)
 2  3  4  5 >6  6< 7  vertausche (i 4<>5)
 2  3  4 >5  5< 6  7  vertausche (i 3<>4)
 2  3 >4  4< 5  6  7  vertausche (i 2<>3)
 2 >3  3< 4  5  6  7  vertausche (i 1<>2)
>2  2< 3  4  5  6  7  vertausche (i 0<>1)
 1  2  3  4  5  6  7  Ausgabe
```

(ii) Standardversion von **Quicksort** (Pseudocode s.u., Feldindizes beginnen bei 1), bei der das letzte Element eines Teilfeldes als Pivot-Element gewählt wird.

Funktion Quicksort(A, $l = 1, r = A$.length)

```
if l < r then
    m = Partition(A, l, r);
    Quicksort(A, l, m − 1);
    Quicksort(A, m + 1, r);
end
```

Funktion Partition(A, int l, int r)

```
pivot = A[r];
i = l;
for j = l to r − 1 do
    if A[j] < pivot then
        Swap(A, i, j);
        i = i + 1;
    end
end
```

Best-Case

1	3	2	6	5	7	4

```
 1  3  2  6  5  7  4  zerlege
```

```
 1  3  2  6  5  7  4* markiere (i 6)
>1< 3  2  6  5  7  4  vertausche (i 0<>0)
 1 >3< 2  6  5  7  4  vertausche (i 1<>1)
 1  3 >2< 6  5  7  4  vertausche (i 2<>2)
 1  3  2 >6  5  7  4< vertausche (i 3<>6)
 1  3  2              zerlege
 1  3  2*             markiere (i 2)
>1< 3  2              vertausche (i 0<>0)
 1 >3  2<             vertausche (i 1<>2)
             5  7  6  zerlege
             5  7  6* markiere (i 6)
            >5< 7  6  vertausche (i 4<>4)
             5 >7  6< vertausche (i 5<>6)
```

Worst-Case

7	6	5	4	3	2	1

```
 1  2  3  4  5  6  7  zerlege
 1  2  3  4  5  6  7* markiere (i 6)
>1< 2  3  4  5  6  7  vertausche (i 0<>0)
 1 >2< 3  4  5  6  7  vertausche (i 1<>1)
 1  2 >3< 4  5  6  7  vertausche (i 2<>2)
 1  2  3 >4< 5  6  7  vertausche (i 3<>3)
 1  2  3  4 >5< 6  7  vertausche (i 4<>4)
 1  2  3  4  5 >6< 7  vertausche (i 5<>5)
 1  2  3  4  5  6 >7< vertausche (i 6<>6)
 1  2  3  4  5  6     zerlege
 1  2  3  4  5  6*    markiere (i 5)
>1< 2  3  4  5  6     vertausche (i 0<>0)
 1 >2< 3  4  5  6     vertausche (i 1<>1)
 1  2 >3< 4  5  6     vertausche (i 2<>2)
 1  2  3 >4< 5  6     vertausche (i 3<>3)
 1  2  3  4 >5< 6     vertausche (i 4<>4)
 1  2  3  4  5 >6<    vertausche (i 5<>5)
 1  2  3  4  5        zerlege
 1  2  3  4  5*       markiere (i 4)
>1< 2  3  4  5        vertausche (i 0<>0)
 1 >2< 3  4  5        vertausche (i 1<>1)
 1  2 >3< 4  5        vertausche (i 2<>2)
 1  2  3 >4< 5        vertausche (i 3<>3)
 1  2  3  4 >5<       vertausche (i 4<>4)
 1  2  3  4           zerlege
 1  2  3  4*          markiere (i 3)
>1< 2  3  4           vertausche (i 0<>0)
 1 >2< 3  4           vertausche (i 1<>1)
 1  2 >3< 4           vertausche (i 2<>2)
 1  2  3 >4<          vertausche (i 3<>3)
 1  2  3              zerlege
 1  2  3*             markiere (i 2)
>1< 2  3              vertausche (i 0<>0)
 1 >2< 3              vertausche (i 1<>1)
 1  2 >3<             vertausche (i 2<>2)
 1  2                 zerlege
 1  2*                markiere (i 1)
```

```
>1< 2                vertausche (i 0<>0)
 1 >2<               vertausche (i 1<>1)
```

(iii) **QuicksortVar**: Variante von Quicksort, bei der immer das mittlere Element eines Teilfeldes als Pivot-Element gewählt wird (Pseudocode s.u., nur eine Zeile neu).
Bei einem Aufruf von PartitionVar auf ein Teilfeld $A[l \dots r]$ wird also erst mithilfe der Unterroutine Swap $A\left[\lfloor \frac{l+r-1}{2} \rfloor\right]$ mit $A[r]$ vertauscht.

Funktion QuicksortVar(A, $l = 1$, $r = A$.length)

if $l < r$ **then**
  $m =$ PartitionVar(A, l, r);
  QuicksortVar$(A, l, m-1)$;
  QuicksortVar$(A, m+1, r)$;
end

Funktion PartitionVar(A, int l, int r)

Swap$(A, \lfloor \frac{l+r-1}{2} \rfloor, r)$;
pivot $= A[r]$;
$i = l$;
for $j = l$ **to** $r-1$ **do**
  if $A[j] <$ *pivot* **then**
    Swap(A, i, j);
    $i = i+1$;
  end
end

Best-Case

1	2	3	4	5	6	7

```
 1  2  3  4  5  6  7  zerlege
 1  2  3  4* 5  6  7  markiere (i 3)
 1  2  3 >4  5  6  7< vertausche (i 3<>6)
>1< 2  3  7  5  6  4  vertausche (i 0<>0)
 1 >2< 3  7  5  6  4  vertausche (i 1<>1)
 1  2 >3< 7  5  6  4  vertausche (i 2<>2)
 1  2  3 >7  5  6  4< vertausche (i 3<>6)
 1  2  3               zerlege
 1  2* 3               markiere (i 1)
 1 >2  3<              vertausche (i 1<>2)
>1< 3  2               vertausche (i 0<>0)
 1 >3  2<              vertausche (i 1<>2)
             5  6  7   zerlege
             5  6* 7   markiere (i 5)
             5 >6  7<  vertausche (i 5<>6)
            >5< 7  6   vertausche (i 4<>4)
             5 >7  6<  vertausche (i 5<>6)
 1  2  3  4  5  6  7   Ausgabe
```

Worst-Case

2	4	6	7	1	5	3

```
 2  4  6  7  1  5  3  zerlege
 2  4  6  7* 1  5  3  markiere (i 3)
 2  4  6 >7  1  5  3< vertausche (i 3<>6)
>2< 4  6  3  1  5  7  vertausche (i 0<>0)
 2 >4< 6  3  1  5  7  vertausche (i 1<>1)
 2  4 >6< 3  1  5  7  vertausche (i 2<>2)
 2  4  6 >3< 1  5  7  vertausche (i 3<>3)
 2  4  6  3 >1< 5  7  vertausche (i 4<>4)
 2  4  6  3  1 >5< 7  vertausche (i 5<>5)
 2  4  6  3  1  5 >7< vertausche (i 6<>6)
 2  4  6  3  1  5     zerlege
 2  4  6* 3  1  5     markiere (i 2)
 2  4 >6  3  1  5<    vertausche (i 2<>5)
>2< 4  5  3  1  6     vertausche (i 0<>0)
 2 >4< 5  3  1  6     vertausche (i 1<>1)
 2  4 >5< 3  1  6     vertausche (i 2<>2)
 2  4  5 >3< 1  6     vertausche (i 3<>3)
 2  4  5  3 >1< 6     vertausche (i 4<>4)
 2  4  5  3  1 >6<    vertausche (i 5<>5)
 2  4  5  3  1        zerlege
 2  4  5* 3  1        markiere (i 2)
 2  4 >5  3  1<       vertausche (i 2<>4)
>2< 4  1  3  5        vertausche (i 0<>0)
 2 >4< 1  3  5        vertausche (i 1<>1)
 2  4 >1< 3  5        vertausche (i 2<>2)
 2  4  1 >3< 5        vertausche (i 3<>3)
 2  4  1  3 >5<       vertausche (i 4<>4)
 2  4  1  3           zerlege
 2  4* 1  3           markiere (i 1)
 2 >4  1  3<          vertausche (i 1<>3)
>2< 3  1  4           vertausche (i 0<>0)
 2 >3< 1  4           vertausche (i 1<>1)
 2  3 >1< 4           vertausche (i 2<>2)
 2  3  1 >4<          vertausche (i 3<>3)
 2  3  1              zerlege
 2  3* 1              markiere (i 1)
 2 >3  1<             vertausche (i 1<>2)
>2< 1  3              vertausche (i 0<>0)
 2 >1< 3              vertausche (i 1<>1)
 2  1 >3<             vertausche (i 2<>2)
 2  1                 zerlege
 2* 1                 markiere (i 0)
>2  1<                vertausche (i 0<>1)
>1< 2                 vertausche (i 0<>0)
 1 >2<                vertausche (i 1<>1)
```

(b) Geben Sie die asymptotische Best- und Worst-Case-Laufzeit von **Mergesort** an.

Best-Case: $\mathcal{O}(n \cdot \log(n))$
Worst-Case: $\mathcal{O}(n^2)$

Aufgabe 2 [Minimum und Maximum]

(a) Argumentieren Sie, warum man das Maximum von n Zahlen nicht mit weniger als $n-1$ Vergleichen bestimmen kann.

> Wenn die n Zahlen in einem unsortierten Zustand vorliegen, müssen wir alle Zahlen betrachten, um das Maximum zu finden. Wir brauchen dazu $n-1$ und nicht n Vergleiche, da wir die erste Zahl zu Beginn des Algorithmus als Maximum definieren und anschließend die verbleibenden Zahlen $n-1$ mit dem aktuellen Maximum vergleichen.

(b) Geben Sie einen Algorithmus im Pseudocode an, der das Maximum eines Feldes der Länge n mit genau $n-1$ Vergleichen bestimmt.

```java
  public static int bestimmeMaximum(int[] a) {
    int max = a[0];
    for (int i = 1; i < a.length; i++) {
      if (a[i] > max) {
        max = a[i];
      }
    }
    return max;
  }
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java

(c) Wenn man das Minimum und das Maximum von n Zahlen bestimmen will, dann kann das natürlich mit $2n-2$ Vergleichen erfolgen. Zeigen Sie, dass man bei jedem beliebigen Feld mit deutlich weniger Vergleichen auskommt, wenn man die beiden Werte statt in zwei separaten Durchläufen in einem Durchlauf geschickt bestimmt.

```java
  /**
   * Diese Methode ist nicht optimiert. Es werden 2n - 2 Vergleiche benötigt.
   *
   * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum gesucht
   *          werden soll.
   *
   * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
   *         der zweite Eintrag das Maximum.
   */
  public static int[] minMaxNaiv(int[] a) {
    int max = a[0];
    int min = a[0];
    for (int i = 1; i < a.length; i++) {
      if (a[i] > max) {
        max = a[i];
      }
      if (a[i] < min) {
        max = a[i];
      }
    }
    return new int[] { min, max };
  }

  /**
   * Diese Methode ist optimiert. Es werden immer zwei Zahlen paarweise
   * betrachtet. Die Anzahl der Vergleiche reduziert sich auf 3n/2 + 2 bzw.
   * 3(n-1)/2 + 4 bei einer ungeraden Anzahl an Zahlen im Feld.
   *
   * nach <a href=
   * "https://www.techiedelight.com/find-minimum-maximum-element-array-using-minimum-
↪  comparisons/">techiedelight.com</a>
   *
   * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum gesucht
   *          werden soll.
   *
   * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
```

```
50      *         der zweite Eintrag das Maximum.
51      */
52    public static int[] minMaxIterativPaarweise(int[] a) {
53      int max = Integer.MIN_VALUE, min = Integer.MAX_VALUE;
54      int n = a.length;
55
56      boolean istUngerade = (n & 1) == 1;
57      if (istUngerade) {
58        n--;
59      }
60
61      for (int i = 0; i < n; i = i + 2) {
62        int maximum, minimum;
63
64        if (a[i] > a[i + 1]) {
65          minimum = a[i + 1];
66          maximum = a[i];
67        } else {
68          minimum = a[i];
69          maximum = a[i + 1];
70        }
71
72        if (maximum > max) {
73          max = maximum;
74        }
75
76        if (minimum < min) {
77          min = minimum;
78        }
79      }
80
81      if (istUngerade) {
82        if (a[n] > max) {
83          max = a[n];
84        }
85
86        if (a[n] < min) {
87          min = a[n];
88        }
89      }
90      return new int[] { min, max };
91    }
92
93    /**
94     * Diese Methode ist nach dem Teile-und-Herrsche-Prinzip optimiert. Er
95     * funktioniert so ähnlich wie der Mergesort.
96     *
97     * nach <a href=
98     * "https://www.enjoyalgorithms.com/blog/find-the-minimum-and-maximum-value-in-an-
    ↪  array">enjoyalgorithms.com</a>
99     *
100    * @param a Ein Feld mit Zahlen, in dem nach dem Minimum und dem Maximum
101    *          gesucht werden soll.
102    * @param l Die linke Grenze.
103    * @param r Die rechts Grenze.
104    *
105    * @return Ein Feld mit zwei Einträgen. Der erste Einträg enthält das Minimum,
106    *         der zweite Eintrag das Maximum.
107    */
108   int[] minMaxRekursiv(int[] a, int l, int r) {
109     int max, min;
110     if (l == r) {
111       max = a[l];
112       min = a[l];
113     } else if (l + 1 == r) {
114       if (a[l] < a[r]) {
115         max = a[r];
116         min = a[l];
117       } else {
```

```java
          max = a[l];
          min = a[r];
        }
      } else {
        int mid = l + (r - l) / 2;
        int[] lErgebnis = minMaxRekursiv(a, l, mid);
        int[] rErgebnis = minMaxRekursiv(a, mid + 1, r);
        if (lErgebnis[0] > rErgebnis[0]) {
          max = lErgebnis[0];
        } else {
          max = rErgebnis[0];
        }
        if (lErgebnis[1] < rErgebnis[1]) {
          min = lErgebnis[1];
        } else {
          min = rErgebnis[1];
        }
      }
      int[] ergebnis = { max, min };
      return ergebnis;
    }

  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/MinimumMaximum.java`

Aufgabe 3 [Schwach zusammenhängend gerichteter Graph]

Wir betrachten eine Variante der Breitensuche (BFS), bei der die Knoten markiert werden, wenn sie das erste Mal besucht werden. Außerdem wird die Suche einmal bei jedem unmarkierten Knoten gestartet, bis alle Knoten markiert sind. Wir betrachten gerichtete Graphen. Ein gerichteter Graph G ist *schwach zusammenhängend*, wenn der ungerichtete Graph (der sich daraus ergibt, dass man die Kantenrichtungen von G ignoriert) zusammenhängend ist.

Exkurs: Schwach zusammenhängend gerichteter Graph
Beim gerichteten Graphen musst du auf die Kantenrichtung achten. Würde man die Richtungen der Kanten ignorieren wäre aber trotzdem jeder Knoten erreichbar. Einen solchen Graphen nennt man schwach zusammenhängend. [a]

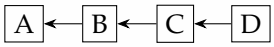
Ein gerichteter Graph heißt (schwach) zusammenhängend, falls der zugehörige ungerichtete Graph (also der Graph, der entsteht, wenn man jede gerichtete Kante durch eine ungerichtete Kante ersetzt) zusammenhängend ist. [b]

[a] `https://studyflix.de/informatik/grundbegriffe-der-graphentheorie-1285`
[b] `https://de.wikipedia.org/wiki/Zusammenhang_(Graphentheorie)`

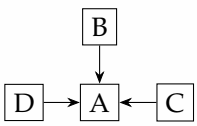
(a) Beschreiben Sie für ein allgemeines $n \in \mathbb{N}$ mit $n \geq 2$ den Aufbau eines schwach zusammenhängenden Graphen G_n, mit n Knoten, bei dem die Breitensuche $\Theta(n)$ mal gestartet werden muss, bis alle Knoten markiert sind.

?

Die Breitensuche benötigt einen Startknoten. Die unten aufgeführten Graphen finden immer nur einen Knoten nämlich den Startknoten.



Oder so:



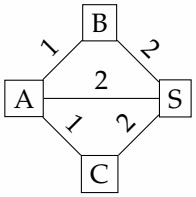
(b) Welche asymptotische Laufzeit in Abhängigkeit von der Anzahl der Knoten (n) und von der Anzahl der Kanten (m) hat die Breitensuche über alle Neustarts zusammen? Beachten Sie, dass die Markierungen nicht gelöscht werden. Geben Sie die Laufzeit in Θ-Notation an. Begründen Sie Ihre Antwort.

Aufgabe 4 [Kürzeste-Wege-Bäume und minimale Spannbäume]

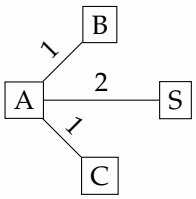
Die Algorithmen von Dijkstra und Jarník-Prim gehen ähnlich vor. Beide berechnen, ausgehend von einem Startknoten, einen Baum. Allerdings berechnet der Algorithmus von Dijkstra einen Kürzesten-Wege-Baum, während der Algorithmus von Jarník-Prim einen minimalen Spannbaum berechnet.

(a) Geben Sie einen ungerichteten gewichteten Graphen G mit höchstens fünf Knoten und einen Startknoten s von G an, so dass **Dijkstra**(G, s) und **Jarník-Prim**(G, s) ausgehend von s verschiedene Bäume in G liefern. Geben Sie beide Bäume an.

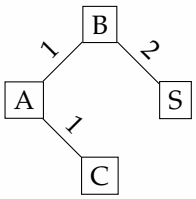
Originalgraph



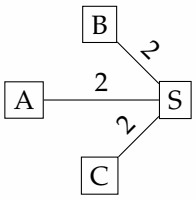
Minimaler Spannbaum 1



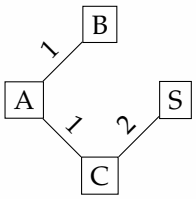
Minimaler Spannbaum 2



Dijkstra-Wegebaum von S aus:



Minimaler Spannbaum 3



(b) Geben Sie eine unendlich große Menge von Graphen an, auf denen der Algorithmus von Jarník-Prim asymptotisch schneller ist als der Algorithmus von Kruskal, der ebenfalls minimale Spannbäume berechnet.

Hinweis: Für einen Graphen mit n Knoten und m Kanten benötigt Jarník-Prim $\mathcal{O}(m + n \log n)$ Zeit, Kruskal $\mathcal{O}(m \log m)$ Zeit.

(c) Sei Z die Menge der zusammenhängenden Graphen und $G \in Z$. Sei n die Anzahl der Knoten von G und m die Anzahl der Kanten von G. Entscheiden Sie mit Begründung, ob $\log m \in \Theta(\log n)$ gilt.

Thema Nr. 2

Teilaufgabe Nr. 1

Aufgabe 1

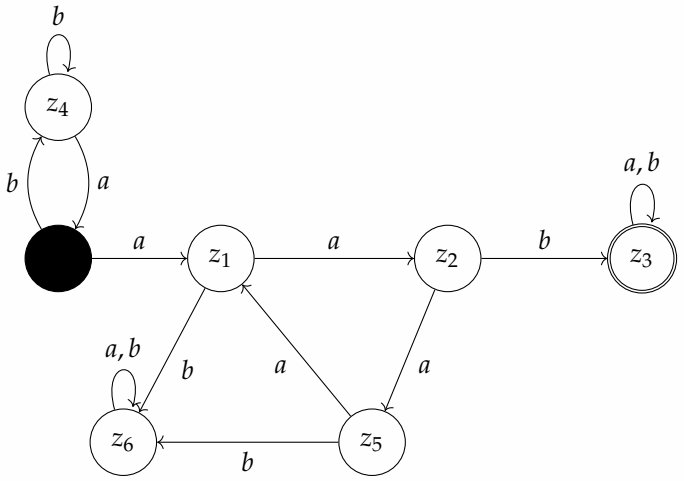
(a) Sei

$L_1 = \{ w \in \{a, b, c\}^* \mid w$ enthält genau zweimal den Buchstaben a und der vorletzte Buchstabe ist ein $c \}$

Geben Sie einen regulären Ausdruck für die Sprache L_1 an.

```
(
  ((b|c)* a (b|c)* a (b|c)* c (b|c))
  |
  ((b|c)* a (b|c)* c a)
)
```

(b) Konstruieren Sie einen deterministischen endlichen Automaten für die Sprache L_2:
$L_2 = \{ w \in \{a,b\}^* \mid w \text{ enthält genau einmal das Teilwort } aab \}$



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/Ahf2oduri

(c) Sei $\mathbb{N} = \{1, 2, 3, \ldots\}$ die Menge der strikt positiven natürlichen Zahlen. Sei
$L_3 = \{ \#a^{i_1}\#a^{i_1}\# \cdots a^{i_{n-1}}\#a^{i_n}\# \mid n, i_1, \ldots, i_n \in \mathbb{N} \text{ und es existiert } j \in \mathbb{N} \text{ mit } i_j = n+1 \}$
eine Sprache über Alphabet $\{\#, a\}$.
So ist z. B. $\#a\#aaa\# \in L_3$ (da das Teilwort $a^3 = aaa$ vorkommt) und $\#a\#a\#a\#a\# \in L_3$ (da das Teilwort $a^5 = aaaaa$ nicht vorkommt). Beweisen Sie, dass L_3 nicht regulär ist.

Aufgabe 2

(a) Zeigen Sie, dass die Sprache

$$L = \{ ww_1ww_2 \mid w, w_1, w_2 \in \{a,b,c\}^* \text{ und } 2|w| \geq |w_1| + |w_2| \}$$

nicht kontextfrei ist.

Exkurs: Pumping-Lemma für Kontextfreie Sprachen

Es gibt eine Pumpzahl. Sie sei j. $a^jb^ja^jc^j$ ist ein Wort aus L, das sicher länger als j ist. Außerdem gilt $2|a^j| \geq |b^j| + |c^j|$. Unser gewähltes Wort ist deshalb in L.
Da $|vwx| \leq j$ und $|xv| \geq 1$ sein muss, liegt vwx entweder in w, w_1 oder w_2.

Aufteilung: vwx **in** w **(erstes** w**):**

u : ε

v : a

w : a^{j-2}

x : a

y : $b^j a^j c^j$

Es gilt $uv^i wx^i y \notin L$ für alle $i \in \mathbb{N}_0$, da $a^j b^j a^j c^j \notin L$ für $i = 0$, da $|a^{j-2}| + |a^j| < |b^j| + |c^j|$

Aufteilung: vwx **in** w **(zweites** w**):**

u : $a^j b^j$

v : a

w : a^{j-2}

x : a

y : c^j

Es gilt $uv^i wx^i y \notin L$ für alle $i \in \mathbb{N}_0$, da $a^j b^j a^j c^j \notin L$ für $i = 0$, da $|a^j| + |a^{j-2}| < |b^j| + |c^j|$

Aufteilung: vwx **in** w_1**:**

u : a^j

v : b

w : b^{j-2}

x : b

y : $a^j c^j$

Es gilt nicht $uv^i wx^i y \in L$ für alle $i \in \mathbb{N}_0$, da $a^j b^j a^j c^j \notin L$ für alle $i > 2$ da $2|a^j| < |b^{j-2+2i}| + |c^j|$ für alle $i > 2$

Aufteilung: vwx **in** w_2**:**

Analog zur Aufteilung vwx in w_1

$\Rightarrow L$ ist nicht kontextfrei.

(b) Betrachten Sie die Aussage

Seien $L_1, \ldots, I_n$ beliebige kontextfreie Sprachen.
Dann ist $\bigcap_{i=1}^{n}, L_i$ immer eine entscheidbare Sprache.

Entscheiden Sie, ob diese Aussage wahr ist oder nicht und begründen Sie Ihre Antwort.

Diese Aussage ist falsch.

Kontextfreie Sprachen sind nicht abgeschlossen unter dem Schnitt, ðdie Schnittmenge zweier kontextfreier Sprachen kann in einer Sprache eines anderen Typs in der Chomsky Sprachen-Hierachie resultieren. Entsteht durch den Schnitt eine Typ-0-Sprache, dann ist diese nicht entscheidbar.

(c) Sei $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$ die Menge der nicht negativen natürlichen Zahlen. Es ist bekannt, dass $L = \{ a^n b^n c^n \mid n \in \mathbb{N} \}$ keine kontextfreie Sprache ist. Ist die Komplementsprache $L_5 = \{a, b, c\}^* \setminus L$ kontextfrei? Begründen Sie Ihre Antwort.

Aufgabe 3

Wir betrachten eine Gödelisierung von Turingmaschinen und bezeichnen mit M_w die Turingmaschine, die gemäß der Kodierung des Binärworts w kodiert wird. Außerdem bezeichnen wir mit $M_w(x)$ die Ausgabe der

Maschine M_w bei Eingabe x. Sie dürfen davon ausgehen, dass x immer ein Binärstring ist. Der bekannte Satz von Rice sagt:

Sei S eine Menge berechenbarer Funktionen mit $\varnothing \neq S \neq \mathcal{R}$, wobei $\mathcal{R}$ die Menge aller berechenbaren Funktionen ist. Dann ist die Sprache $L = \{ w \mid f_{M_w} \in S \}$ unentscheidbar.

Hier ist f_{M_w} die von M_w berechnete Funktion.

Zeigen Sie für jede der nachfolgenden Sprachen über dem Alphabet $\{0, 1\}$ entweder, dass sie entscheidbar ist, oder zeigen Sie mit Hilfe des Satzes von Rice, dass sie unentscheidbar ist. Geben Sie beim Beweis der Unentscheidbarkeit die Menge S der berechenbaren Funktionen an, auf die Sie den Satz von Rice anwenden. Wir bezeichnen die Länge der Eingabe x mit $|x|$.

(a) $L = \{ w \mid M_w$ akzeptiert die Binarkodierungen der Primzahlen (und lehnt alles andere ab) $\}$

(b) $L = \{ w \mid$ es gibt eine Hingabe x, so dass $M_w(x)$ das Symbol 1 enthält $\}$

(c) $L = \{ w \mid M_w(x)$ hält für jedes x mit $|x| < 1000$ nach höchstens 100 Schritten an $\}$

(d) $L = \{ w \mid M_w$ hat für jede Eingabe dieselbe Ausgabe $\}$

(e) $L = \{ w \mid$ die Menge der Eingaben, die von M_w akzeptiert werden, ist endlich $\}$

Aufgabe 4

Betrachten Sie die folgenden Probleme:

Clique Ein ungerichteter Graph $G = (V, E)$, eine Zahl $k \in \mathcal{N}$ Gibt es eine Menge $S \subseteq V$ mit $|S| = k$, sodass für alle Knoten $u \neq v \in V$ gilt, dass $\{u, v\}$ eine Kante in E ist?

Almost Clique Ein ungerichteter Graph $G = (V, E)$, eine Zahl $k \in \mathcal{N}$ Gibt es eine Menge $S \subseteq V$ mit $|S| = k$, sodass die Anzahl der Kanten zwischen Knoten in S genau $\frac{k(k-1)}{2} - 1$ ist?

Zeigen Sie, dass das Problem Almost Clique NP-vollständig ist. Nutzen Sie dafür die NP-Vollständigkeit von Clique.

Hinweis: Die Anzahl der Kanten einer k-Clique sind $\frac{k(k-1)}{2}$.

Exkurs: Cliquenproblem

Exkurs: Almost Clique
Eine Gruppe von Knoten wird Almost Clique genannt, wenn nur eine Kante ergänzt werden muss, damit sie zu einer Clique wird.

You can reduce to this from *CLIQUE*.

Given a graph $G = (V, E)$ and t, construct a new graph G^* by adding two new vertices $\{v_{n+1}, v_{n+2}\}$ and connecting them with all of G's vertices but removing the edge $\{v_{n+1}, v_{n+2}\}$, i.e. they are not neighbors in G^*. return G^* and $t + 2$.

If G has a t sized clique by adding it to the two vertices we get an $t + 2$ almost clique in G^* (by adding $\{v_{n+1}, v_{n+2}\}$).

If G^* has a $t + 2$ almost clique we can look at three cases:

1) It contains the two vertices $\{v_{n+1}, v_{n+2}\}$, then the missing edge must be $\{v_{n+1}, v_{n+2}\}$ and this implies that the other t vertices form a t clique in G.

2) It contains one of the vertices $\{v_{n+1}, v_{n+2}\}$, say w.l.o.g. v_{n+1}, then the missing edge must be inside G, say $e = \{u, v\} \in G$. If we remove u and v_{n+1} then the other t vertices, which are in G must form a clique of size t.

3) It does not contain any of the vertices $\{v_{n+1}, v_{n+2}\}$, then it is clear that this group is in G and must contain a clique of size t.

It is also clear that the reduction is in polynomial time, actually in linear time, log-space. [a]

[a] https://cs.stackexchange.com/a/76627

Teilaufgabe Nr. 2

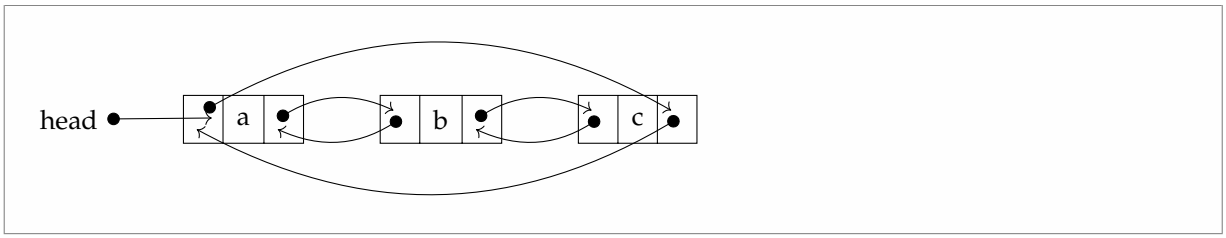
Aufgabe 2

Gegeben sei die folgende Java-Implementierung einer doppelt-verketteten Liste.

```java
class DoubleLinkedList {
  private Item head;

  public DoubleLinkedList() {
    head = null;
  }

  public Item append(Object val) {
    if (head == null) {
      head = new Item(val, null, null);
      head.prev = head;
      head.next = head;
    } else {
      Item item = new Item(val, head.prev, head);
      head.prev.next = item;
      head.prev = item;
    }
    return head.prev;
  }

  public Item search(Object val) {
    // ...
  }

  public void delete(Object val) {
    // ...
  }

  class Item {
    private Object val;
    private Item prev;
    private Item next;

    public Item(Object val, Item prev, Item next) {
      this.val = val;
      this.prev = prev;
      this.next = next;
    }
  }
}
```

(a) Skizzieren Sie den Zustand der Datenstruktur nach Aufruf der folgenden Befehlssequenz. Um Variablen mit Zeigern auf Objekte darzustellen, können Sie mit dem Variablennamen beschriftete Pfeile verwenden.

```java
DoubleLinkedList list = new DoubleLinkedList();
list.append("a");
list.append("b");
list.append("c");
```



(b) Implementieren Sie in der Klasse `DoubleLinkedList` die Methode `search`, die zu einem gegebenen Wert das Item der Liste mit dem entsprechenden Wert, oder `null` falls der Wert nicht in der Liste enthalten ist, zurückgibt.

```java
public Item search(Object val) {
  Item item = null;
  if (head != null) {
    item = head;
    do {
      if (item.val.equals(val)) {
        return item;
      }
      item = item.next;
    } while (!item.equals(head));
  }
  return null;
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/DoubleLinkedList.java

(c) Implementieren Sie in der Klasse `DoubleLinkedList` die Methode `delete`, die das erste Vorkommen eines Wertes aus der Liste entfernt. Ist der Wert nicht in der Liste enthalten, terminiert die Methode „stillschweigend“, ðohne Änderung der Liste und ohne Fehlermeldung. Sie dürfen die Methode `search` aus Teilaufgabe b) verwenden, auch wenn Sie sie nicht implementiert haben.

```java
public void delete(Object val) {
  Item item = search(val);
  if (item != null) {
    if (head.next.equals(head)) {
      head = null;
    } else {
      if (item.equals(head)) {
        head = item.next;
      }
      item.prev.next = item.next;
      item.next.prev = item.prev;
    }
  }
}
```

Code-Beispiel auf Github ansehen: src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/DoubleLinkedList.java

(d) Beschreiben Sie die notwendigen Änderungen an der Datenstruktur und an den bisherigen Implementierungen, um eine Methode `size`, die die Anzahl der enthaltenen Items zurück gibt, mit Laufzeit $\mathcal{O}(1)$ zu realisieren.

In der Klasse wird ein Zähler eingefügt, der bei jedem Aufruf der Methode `append` um eins nach oben gezählt wird und bei jedem erfolgreichen Löschen in der `delete`-Methode um eins nach unten gezählt wird. Mit `return` kann der Zählerstand in $\mathcal{O}(1)$ ausgegeben werden. Dazu müsste ein Getter zum Ausgeben implementiert werden. Die Datenstruktur bleibt unverändert.

Aufgabe 3

(a) Betrachten Sie folgenden Binärbaum T.

Geben Sie die Schlüssel der Knoten in der Reihenfolge an, wie sie von einem Preorder-Durchlauf (= TreeWalk) von T ausgegeben werden.

Exkurs: Preorder-Traversierung eines Baum

besuche die Wurzel, dann den linken Unterbaum, dann den rechten Unterbaum; auch: WLR

```java
private void besuchePreorder(BaumKnoten knoten, ArrayList<Comparable> schlüssel) {
  if (knoten != null) {
```

```
        schlüssel.add((Comparable) knoten.gibSchlüssel());
        besuchePreorder(knoten.gibLinks(), schlüssel);
        besuchePreorder(knoten.gibRechts(), schlüssel);
      }
    }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/baum/BinaerBaum.java`

14, 15, 98, 3, 4, 25, 33, 19, 18, 26, 17

(b) Betrachten Sie folgende Sequenz als Ergebnis eines Preorder-Durchlaufs eines binären Suchbaumes T. Zeichnen Sie T und erklären Sie, wie Sie zu Ihrer Schlussfolgerung gelangen.

$$[8,7,4,2,1,3,5,6,10,9,11]$$

Hinweis: Welcher Schlüssel ist die Wurzel von T? Welche Knoten sind in seinem linken/rechten Teilbaum gespeichert? Welche Schlüssel sind die Wurzeln der jeweiligen Teilbäume?

(c) Anstelle von sortierten Zahlen soll ein Baum nun verwendet werden, um relative Positionsangaben zu speichern. Jeder Baumknoten enthält eine Beschriftung und einen Wert (vgl. Abb. 1), der die ganzzahlige relative Verschiebung in horizontaler Richtung gegenüber seinem Elternknoten angibt. Die zu berechnenden Koordinaten für einen Knoten ergeben sich aus seiner Tiefe im Baum als y-Wert und aus der Summe aller Verschiebungen auf dem Pfad zur Wurzel als x-Wert. Das Ergebnis der Berechnung ist in Abb. 2 visualisiert. Geben Sie einen Algorithmus mit linearer Laufzeit in Pseudo-Code oder einer objektorientierten Programmiersprache Ihrer Wahl an. Der Algorithmus erhält den Zeiger auf die Wurzel eines Baumes als Eingabe und soll Tupel mit den berechneten Koordination aller Knoten des Baums in der Form (Beschriftung, x, y) zurück- oder ausgeben.

```
public class Knoten {
  public Knoten links;
  public Knoten rechts;
  public String name;

  /**
   * Bewegung bezüglich des Vorknotens. Relative Lage.
   */
  public int xVerschiebung;

  public Knoten(String name, int xVerschiebung) {
    this.name = name;
    this.xVerschiebung = xVerschiebung;
  }

  public void durchlaufen() {
    durchlaufe(this, 0 + xVerschiebung, 0);
  }

  private void durchlaufe(Knoten knoten, int x, int y) {
    System.out.println("Beschriftung: " + knoten.name + " x: " + x + " y: " + y);

    if (links != null) {
      links.durchlaufe(links, x + links.xVerschiebung, y + 1);
    }
    if (rechts != null) {
      rechts.durchlaufe(rechts, x + rechts.xVerschiebung, y + 1);
    }
  }

  public static void main(String[] args) {
    Knoten a = new Knoten("a", 1);
    Knoten b = new Knoten("b", 1);
    Knoten c = new Knoten("c", -2);
```

```java
        Knoten d = new Knoten("d", 2);
        Knoten e = new Knoten("e", 0);

        a.links = b;
        a.rechts = c;
        c.links = d;
        c.rechts = e;

        a.durchlaufen();
    }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2021/fruehjahr/Knoten.java`

Aufgabe 5

(a) Nennen Sie zwei wünschenswerte Eigenschaften von Hashfunktionen.

surjektiv Die Abbildung soll surjektiv sein, ðjeder Index soll berechnet werden können.

gleichverteilt Durch die Hashfunktion soll möglichst eine Gleichverteilung auf die Buckets (Indexliste) erfolgen.

effizient Zudem sollte die Verteilung mittels Hashfunktion möglichst effizient gewählt werden.

(b) Wie viele Elemente können bei Verkettung und wie viele Elemente können bei offener Adressierung in einer Hashtabelle mit m Zeilen gespeichert werden?

Verkettung Es darf mehr als ein Element pro Bucket enthalten sein, deswegen können beliebig viele Element gespeichert werden.

offene Addressierung (normalerweise) ein Element pro Bucket, deshalb ist die Anzahl der speicherbaren Elemente höchstens m. Können in einem Bucket k Elemente gespeichert werden, dann beträgt die Anzahl der speicherbaren Elemente $k \cdot m$.

(c) Angenommen, in einer Hashtabelle der Größe m sind alle Einträge (mit mindestens einem Wert) belegt und insgesamt n Werte abgespeichert.

Geben Sie in Abhängigkeit von m und n an, wie viele Elemente bei der Suche nach einem nicht enthaltenen Wert besucht werden müssen. Sie dürfen annehmen, dass jeder Wert mit gleicher Wahrscheinlichkeit und unabhängig von anderen Werten auf jeden der m Plätze abgebildet wird (einfaches gleichmäßiges Hashing).

$\frac{n}{m}$ Beispiel: 10 Buckets, 30 Elemente: $\frac{30}{10} = 3$ Elemente im Bucket, die man durchsuchen muss.

(d) Betrachten Sie die folgende Hashtabelle mit der Hashfunktion $h(x) = x \mod 11$. Hierbei steht $\varnothing$ für eine Zelle, in der kein Wert hinterlegt ist.

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	$\varnothing$	$\varnothing$	3	$\varnothing$	16	28	18	$\varnothing$	$\varnothing$	32

Führen Sie nun die folgenden Operationen mit offener Adressierung mit linearem Sondieren aus und geben Sie den Zustand der Datenstruktur nach jedem Schritt an. Werden für eine Operation mehrere Zellen betrachtet, aber nicht modifiziert, so geben Sie deren Indizes in der betrachteten Reihenfolge an.

(i) Insert 7

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	∅	∅	3	∅	16	28	18	7_2	∅	32

(ii) Insert 20

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	∅	∅	3	∅	16	28	18	7_2	20_1	32

(iii) Delete 18

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	∅	∅	3	∅	16	28	del	7_2	20_1	32

del ist eine Marke, die anzeigt, dass gelöscht wurde und der Bucket nicht leer ist.

(iv) Search 7

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	∅	∅	3	∅	16	28	del	7_2	20_1	32

$h(7) = 7 \mod 11 = 7$

7 (Index) $\rightarrow$ del lineares sondieren $\rightarrow$ 8 (Index) $\rightarrow$ gefunden

(v) Insert 5

Index	0	1	2	3	4	5	6	7	8	9	10
Wert	11	∅	∅	3	∅	16	28	5_3	7_2	20_1	32