46115
Herbst
2019

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

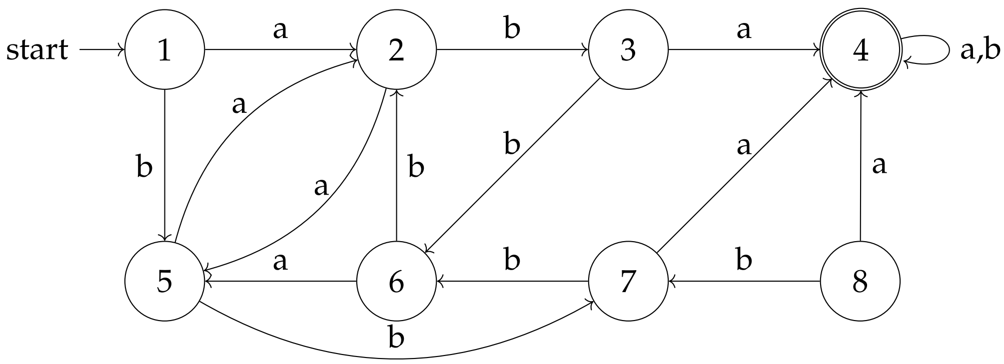
Aufgabe 1 [Reguläre Sprachen)]

(a) Geben Sie einen regulären Ausdruck für die Sprache über dem Alphabet $\{a, b\}$ an, die genau alle Wörter enthält, die eine gerade Anzahl a's haben.

Lösungsvorschlag

$b^*(ab^*ab^*)^*$

(b) Sei A der folgende DEA über dem Alphabet $\{a, b\}$:



Führen Sie den Minimierungsalgorithmus für A durch und geben Sie den minimalen äquivalenten DEA für $L(A)$ als Zeichnung an.

Aufgabe 2

(a) Betrachten Sie die folgenden Sprachen:

L, = ar rer d" |n,meN

IL; = art "dneN Zeigen Sie für Zı und La, ob sie kontextfrei sind oder nicht. Für den Beweis von Kontext- Freiheit in dieser Frage reicht die Angabe eines Automaten oder einer Grammatik. (Beschrei-

ben Sie dann die Konstruktionsidee des Automaten oder der Grammatik.) Für den Beweis von Nicht-Kontext-Freiheit verwenden Sie eine der üblichen Methoden.

(b) Eine kontextfreie Grammatik ist in Chomsky-Normalform, falls die folgenden Bedingungen erfüllt sind:

- alle Regeln sind von der Form X — YZ oder X — o mit Nichtterminalzeichen X,Y, Z und Terminalzeichen o,
- alle Nichtterminalzeichen sind erreichbar vom Startsymbol und
- alle Nichtterminalzeichen sind erzeugend, d. h. für jedes Nichtterminalzeichen X gibt es ein Wort w über dem Terminalalphabet, so dass X =>* w.

Bringen Sie die folgende Grammatik in Chomsky-Normalform.

$$P = \left\{ \begin{aligned} S &\to AAB \mid CD \mid abc \\ A &\to AAAA \mid a \\ B &\to BB \mid S \\ C &\to CCC \mid CC \\ D &\to d \end{aligned} \right\}$$

Der Automat auf flaci.com (FLACI: Formale Sprachen, abstrakte Automaten, Compiler und Interpreter) Ein Projekt der Hochschule Zittau/Görlitz und der Pädagogischen Hochschule Schwyz: flaci.com/Gf7f9tp7z

Das Startsymbol der Grammatik ist S, das Terminalalphabet ist a,b,c,d und die Menge der Nichtterminalzeichen ist S,A,B,C,D.

Lösungsvorschlag

(i) **Elimination der ε-Regeln**

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

$\varnothing$ Nichts zu tun

(ii) **Elimination von Kettenregeln**

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

Eine rechte Seite in der C vorkommt, lässt sich wegen $\{C \to CCC \mid CC\}$ nicht ableiten, weil es zu einer Endlosschleife kommt. Wir entfernen die entsprechenden Regeln.

$$P = \left\{ \begin{aligned} S &\to AAB \mid abc \\ A &\to AAAA \mid a \\ B &\to BB \mid AAB \mid abc \end{aligned} \right\}$$

(iii) **Separation von Terminalzeichen**

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

$P = \{$

$$
\begin{aligned}
S &\to AAB \mid T_aT_bT_c \\
A &\to AAAA \mid a \\
B &\to BB \mid AAB \mid T_aT_bT_c \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c
\end{aligned}
$$

}

(iv) **Elimination von mehrelementigen Nonterminalketten**

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

P = {

$$
\begin{aligned}
S &\to AS_1 \mid T_aS_2 \\
A &\to AA_1 \mid a \\
B &\to BB \mid AS_1 \mid T_aS_2 \\
T_a &\to a \\
T_b &\to b \\
T_c &\to c \\
S_1 &\to AB \\
S_2 &\to T_bT_c \\
A_1 &\to AA_2 \\
A_2 &\to AA
\end{aligned}
$$

}

Aufgabe 4 [(Sortierverfahren)]

In der folgenden Aufgabe soll ein Feld A von ganzen Zahlen *aufsteigend* sortiert werden. Das Feld habe n Elemente $A[1]$ bis $A[n]$. Der folgende Algorithmus sei gegeben:

```
var A : array[1..n] of integer;

procedure selection_sort
var i, j, smallest, tmp : integer;
begin
  for j := 1 to n-1 do begin
    smallest := j;
    for i := j + 1 to n do begin
      if A[i] < A[smallest] then
        smallest := i;
    end
    tmp = A[j];
```

```
      A[j] = A[smallest];
      A[smallest] = tmp;
    end
end
```

(a) Sortieren Sie das folgende Feld mittels des Algorithmus. Notieren Sie alle Werte, die die Variable *smallest* jeweils beim Durchlauf der inneren Schleife annimmt. Geben Sie die Belegung des Feldes nach jedem Durchlauf der äußeren Schleife in einer neuen Zeile an.

Lösungsvorschlag

Ausgang

27	32	3	6	17	44	42	29	8	14

nach 1. Durchlauf ($j = 1$)

smallest: (1) 3

3	32	27	6	17	44	42	29	8	14

nach 2. Durchlauf ($j = 2$)

smallest: (2) 3 4

3	6	27	32	17	44	42	29	8	14

nach 3. Durchlauf ($j = 3$)

smallest: (3) 5 9

3	6	8	32	17	44	42	29	27	14

nach 4. Durchlauf ($j = 4$)

smallest: (4) 5 10

3	6	8	14	17	44	42	29	27	32

nach 5. Durchlauf ($j = 5$)

smallest: (5) -

3	6	8	14	17	44	42	29	27	32

nach 6. Durchlauf ($j = 6$)

smallest: (6) 7 8 9

3	6	8	14	17	27	42	29	44	32

nach 7. Durchlauf ($j = 7$)

smallest: (7) 8

3	6	8	14	17	27	29	42	44	32

nach 8. Durchlauf ($j = 8$)

smallest: (8) 10

3	6	8	14	17	27	29	32	44	42

nach 9. Durchlauf ($j = 9$)

smallest: (9) 10

3	6	8	14	17	27	29	32	44	42

fertig

3	6	8	14	17	27	29	32	42	44

(b) Der Wert der Variablen *smallest* wird bei jedem Durchlauf der äußeren Schleife mindestens ein Mal neu gesetzt. Wie muss das Feld A beschaffen sein, damit der Variablen *smallest* ansonsten niemals ein Wert zugewiesen wird? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Wenn das Feld bereits aufsteigend sortiert ist, dann nimmt die Variable *smallest* in der innneren Schleife niemals einen neuen Wert an.

(c) Welche Auswirkung auf die Sortierung oder auf die Zuweisungen an die Variable *smallest* hat es, wenn der Vergleich in Zeile 9 des Algorithmus statt $A[i] < A[\text{smallest}]$ lautet $A[i] \leq A[\text{smallest}]$? Begründen Sie Ihre Antwort.

Lösungsvorschlag

Der Algorithmus sortiert dann nicht mehr *stabil*, ðdie Eingabereihenfolge von Elementen mit *gleichem Wert* wird beim Sortieren nicht mehr *bewahrt*.

(d) Betrachten Sie den Algorithmus unter der Maßgabe, dass Zeile 9 wie folgt geändert wurde:

```
if A[i] > A[smallest] then
```

Welches Ergebnis berechnet der Algorithmus nun?

Lösungsvorschlag

Der Algorithmus sortiert jetzt absteigend.

(e) Betrachten Sie die folgende *rekursive* Variante des Algorithmus. Der erste Parameter ist wieder das zu sortierende Feld, der Parameter n ist die Größe des Feldes und der Parameter *index* ist eine ganze Zahl. Die Funktion `min_index(A, x, y)` berechnet für $1 \leq x \leq y \leq n$ den Index des kleinsten Elements aus der Menge `{A[x], A[x+1],...,A[y]}`

```
procedure rek_selection_sort(A, n, index : integer)
var k, tmp : integer;
begin
if (Abbruchbedingung) then return;
  k = min_index(A, index, n);
  if k <> index then begin
  tmp := A[k];
  A[k] := A[index];
  A[index] := tmp;
  end
  (rekursiver Aufruf)
end
```

Der initiale Aufruf des Algorithmus lautet: `rek_selection_sort(A, n, 1)`

Vervollständigen Sie die fehlenden Angaben in der Beschreibung des Algorithmus für

- die Abbruchbedingung in Zeile 4 und

Lösungsvorschlag

`n = index` bzw `n == index`

Begründung: Wenn der aktuelle Index so groß ist wie die Anzahl der Elemente im Feld, dann muss / darf abgebrochen werden, denn dann ist das Feld soriert.

- den rekursiven Aufruf in Zeile 11.

Lösungsvorschlag

`rek_selection_sort(A, n, index + 1)`

Am Ende der Methode wurde an die Index-Position *index* das kleinste Element gesetzt, jetzt muss an die nächste Index-Position ($index + 1$) der kleinste Wert, der noch nicht sortieren Zahlen, gesetzt werden.

Begründen Sie Ihre Antworten.

```java
import static org.bschlangaul.helfer.Konsole.zeigeZahlenFeld;

public class SelectionSort {

  public static void selectionSort(int[] A) {
    int smallest, tmp;
```

```java
    for (int j = 0; j < A.length - 1; j++) {
      System.out.println("\nj = " + (j + 1));
      smallest = j;
      for (int i = j + 1; i < A.length; i++) {
        if (A[i] < A[smallest]) {
          smallest = i;
          System.out.println(smallest + 1);
        }
      }
      tmp = A[j];
      A[j] = A[smallest];
      A[smallest] = tmp;
      zeigeZahlenFeld(A);
    }
  }

  public static void rekSelectionSort(int[] A, int n, int index) {
    int k, tmp;

    if (index == n - 1) {
      return;
    }
    k = minIndex(A, index, n);
    if (k != index) {
      tmp = A[k];
      A[k] = A[index];
      A[index] = tmp;
    }
    rekSelectionSort(A, n, index + 1);
  }

  public static int minIndex(int[] A, int x, int y) {
    int smallest = x;
    for (int i = x; i < y; i++) {
      if (A[i] < A[smallest]) {
        smallest = i;
      }
    }
    return smallest;
  }

  public static void main(String[] args) {
    int[] A = new int[] { 27, 32, 3, 6, 17, 44, 42, 29, 8, 14 };
    selectionSort(A);

    A = new int[] { 27, 32, 3, 6, 17, 44, 42, 29, 8, 14 };
    rekSelectionSort(A, A.length, 0);
    zeigeZahlenFeld(A);
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/SelectionSort.java`

Aufgabe 6 (Stacks) [Mystery-Stacks]

Gegeben sei die Implementierung eines Stacks ganzer Zahlen mit folgender Schnittstelle:

```java
import java.util.Stack;

/**
 * Um schnell einen lauffähigen Stack zu bekommen, verwenden wir den Stack aus
 * der Java Collection.
 */
public class IntStack {
  private Stack<Integer> stack = new Stack<Integer>();

  /**
   * Legt Element i auf den Stack.
   *
   * @param i Eine Zahl, die auf dem Stack gelegt werden soll.
   */
  public void push(int i) {
    stack.push(i);
  }

  /**
   * Gibt oberstes Element vom Stack.
   *
   * @return Das oberste Element auf dem Stapel.
   */
  public int pop() {
    return stack.pop();
  }

  /**
   * Fragt ab, ob Stack leer ist.
   *
   * @return Wahr, wenn der Stapel leer ist.
   */
  public boolean isEmpty() {
    return stack.empty();
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/IntStack.java`

Betrachten Sie nun die Realisierung der folgenden Datenstruktur `Mystery`, die zwei Stacks benutzt.

```java
public class Mystery {
  private IntStack a = new IntStack();
  private IntStack b = new IntStack();

  public void foo(int item) {
    a.push(item);
  }

  public int bar() {
```

```
  if (b.isEmpty()) {
    while (!a.isEmpty()) {
      b.push(a.pop());
    }
  }
  return b.pop();
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/Mystery.java`

(a) Skizzieren Sie nach jedem Methodenaufruf der im folgenden angegebenen Befehlssequenz den Zustand der beiden Stacks eines Objekts `m` der Klasse `Mystery`. Geben Sie zudem bei jedem Aufruf der Methode `bar` an, welchen Wert diese zurückliefert.

```
Mystery m = new Mystery();
m.foo(3);
m.foo(5);
m.foo(4);
m.bar();
m.foo(7);
m.bar();
m.foo(2);
m.bar();
m.bar();
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/mystery_stack/Mystery.java`

Lösungsvorschlag

Code	Stack b	Stack b	Rückgabewert
`m.foo(3);`	$\{3\}$	$\{\}$	
`m.foo(5);`	$\{5, 3\}$	$\{\}$	
`m.foo(4);`	$\{4, 5, 3\}$	$\{\}$	
`m.bar();`	$\{\}$	$\{5, 4\}$	3
`m.foo(7);`	$\{7\}$	$\{5, 4\}$	
`m.bar();`	$\{7\}$	$\{4\}$	5
`m.foo(2);`	$\{2, 7\}$	$\{\}$	
`m.bar();`	$\{2, 7\}$	$\{\}$	4
`m.bar();`	$\{\}$	$\{2\}$	7

(b) Sei n die Anzahl der in einem Objekt der Klasse `Mystery` gespeicherten Werte. Im folgenden wird gefragt, wieviele Aufrufe von Operationen der Klasse `IntStack` einzelne Aufrufe von Methoden der Klasse `Mystery` verursachen. Begründen Sie jeweils Ihre Antwort.

(i) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `foo(x)` im besten Fall?

Lösungsvorschlag

Einen Aufruf, nämlich `a.push(i)`

(ii) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `foo(x)` im schlechtesten Fall?

Lösungsvorschlag

Einen Aufruf, nämlich `a.push(i)`

(iii) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `bar()` im besten Fall?

Lösungsvorschlag

Wenn der Stack `b` nicht leer ist, dann werden zwei Aufrufe benötigt, nämlich `b.isEmpty()` und `b.pop()`

(iv) Wie viele Aufrufe von Operationen der Klasse `IntStack` verursacht die Methode `bar()` im schlechtesten Fall?

Lösungsvorschlag

Wenn der Stack `b` leer ist, dann liegen all n Objekte im Stack `a`. Die Objekte im Stack `a` werden in der `while`-Schleife nach `b` verschoben. Pro Objekt sind drei Aufrufe nötig, also $3 \cdot n$. `b.isEmpty()` (erste Zeile in der Methode) und `b.pop()` (letzte Zeile in der Methode) wird immer aufgerufen. Wenn alle Objekt von `a` nach `b` verschoben wurden, wird zusätzlich noch einmal in der Bedingung der `while`-Schleife `a.isEmpty()` aufgerufen. Im schlechtesten Fall werden also $3 \cdot n + 3$ Operationen der Klasse `IntStack` aufgerufen.

(c) Welche allgemeinen Eigenschaften werden durch die Methoden `foo` und `bar` realisiert? Unter welchem Namen ist diese Datenstruktur allgemein bekannt?

Lösungsvorschlag

`foo()` Legt das Objekt auf den Stack `a`. Das Objekt wird in die Warteschlange eingereiht. Die Methode müsste eigentlich `enqueue()` heißen.

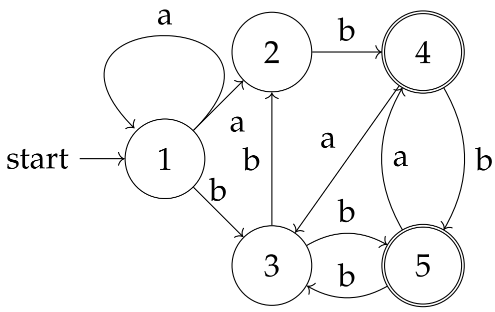
`bar()` Verschiebt alle Objekte vom Stack `a` in umgekehrter Reihenfolge in den Stack `b`, aber nur dann, wenn Stack `b` leer ist. Entfernt dann den obersten Wert aus dem Stack `b` und gibt ihn zurück. Das zuerst eingereihte Objekt wird aus der Warteschlange entnommen. Die Methode müsste eigentlich `dequeue()` heißen.

Die Datenstruktur ist unter dem Namen Warteschlange oder Queue bekannt

Thema Nr. 2

Aufgabe 1 [Komplemetieren eines NEA]

Es sei der nichtdeterministische endliche Automat $A = (\{1,2,3,4,5\}, \{a,b\}, \delta, \{4,5\}, 1)$ gegeben, wobei δ durch folgenden Zeichnung beschrieben ist.

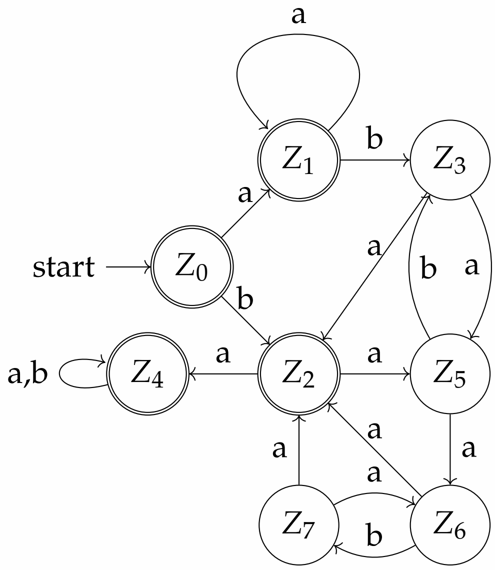


Konstruieren Sie nachvollziehbar einen deterministischen endlichen Automaten A' , der das Komplement von $L(A)$ akzeptiert!

Lösungsvorschlag

Zuerst mit Hilfe der Potenzmengenkonstruktion einen deterministischen endlichen Automaten erstellen und dann die Zustände mit den Endzuständen tauschen.

Name	Zustandsmenge	Eingabe a	Eingabe b
Z_0	$Z_0\{1\}$	$Z_1\{1,2\}$	$Z_2\{3\}$
Z_1	$Z_1\{1,2\}$	$Z_1\{1,2\}$	$Z_3\{3,4\}$
Z_2	$Z_2\{3\}$	$Z_4\{\}$	$Z_5\{2,5\}$
Z_3	$Z_3\{3,4\}$	$Z_2\{3\}$	$Z_5\{2,5\}$
Z_4	$Z_4\{\}$	$Z_4\{\}$	$Z_4\{\}$
Z_5	$Z_5\{2,5\}$	$Z_6\{4\}$	$Z_3\{3,4\}$
Z_6	$Z_6\{4\}$	$Z_2\{3\}$	$Z_7\{5\}$
Z_7	$Z_7\{5\}$	$Z_6\{4\}$	$Z_2\{3\}$



Aufgabe 2 [Rechtslineare Grammatik]

Gegeben ist die rechtslineare Grammatik $G = (\{a, b\}, \{S, A, B, C, D\}, S, P)$ mit

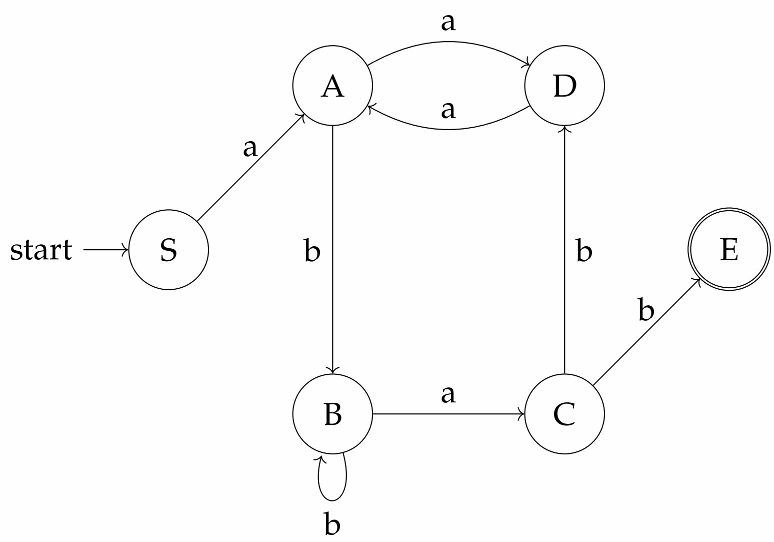
$$P = \left\{ \begin{array}{l} S \to aA \\ A \to bB \\ A \to aD \\ B \to aC \\ B \to bB \\ C \to bD \\ C \to b \\ D \to aA \end{array} \right\}$$

. Sei L die von G erzeugte Sprache.

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/Gpkv4ansc

(a) Zeichnen Sie einen nichtdeterministischen endlichen Automaten, der L akzeptiert!

Lösungsvorschlag



Der Automat auf flaci.com (FLACI: Formale Sprachen, abstrakte Automaten, Compiler und Interpreter) Ein Projekt der Hochschule Zittau/Görlitz und der Pädagogischen Hochschule Schwyz: flaci.com/Ahh979eqz

(b) Konstruieren Sie auf nachvollziehbare Weise einen regulären Ausdruck α mit $L(\alpha) = L$!

Lösungsvorschlag

$(ab + ab|(a|b+) + ab + ab)$ (von Flaci automatisch konvertiert)

Aufgabe 7 (Heapify) [Heapify]

Schreiben Sie in Pseudocode eine Methode `heapify(int[] a)`, welche im übergebenen Array der Länge n die Heapeigenschaft in $\mathcal{O}(n)$ Schritten herstellt. D. h. als Ergebnis soll in a gelten, dass $a[i] \leq a[2i+1]$ und $a[i] \leq a[i+2]$.

Lösungsvorschlag

```java
import org.bschlangaul.helfer.Konsole;

/**
 * Nach Pseudocode nach
 * https://www.oreilly.com/library/view/algorithms-in-a/9780596516246/ch04s06.html
 */
public class Heapify {

  public static void buildHeap(int a[]) {
    int n = a.length;
    for (int i = n / 2 - 1; i >= 0; i--) {
      heapify(a, i, n);
    }
  }

  public static void heapify(int a[], int index, int max) {
    int left = 2 * index + 1;
    int right = 2 * index + 2;
    int smallest;
```

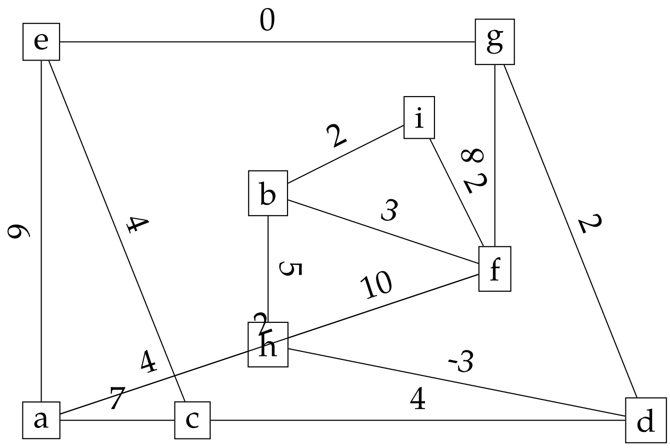
```java
    if (left < max && a[left] < a[index]) {
      smallest = left;
    } else {
      smallest = index;
    }

    if (right < max && a[right] < a[smallest]) {
      smallest = right;
    }

    if (smallest != index) {
      int tmp = a[index];
      a[index] = a[smallest];
      a[smallest] = tmp;
      heapify(a, smallest, max);
    }
  }

  public static void main(String[] args) {
    int[] a = new int[] { 5, 3, 16, 2, 10, 14 };
    buildHeap(a);
    Konsole.zeigeZahlenFeld(a); // 2 3 14 5 10 16
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_46115/jahr_2019/herbst/Heapify.java`

Aufgabe 8 [Maximaler Spannbaum mit Jarník/Prim]

(a) Durch folgende Adjazenzmatrix sei ein ungerichteter Graph G mit Kantenlängen gegeben.

$$
\begin{array}{c|c}
 & \begin{array}{ccccccccc} a & b & c & d & e & f & g & h & \text{i} \end{array} \\
\begin{array}{c} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \\ i \end{array} &
\begin{pmatrix}
0 & 0 & 7 & 0 & 9 & 2 & 0 & 4 & 0 \\
0 & 0 & 0 & 0 & 0 & 3 & 0 & 5 & 2 \\
7 & 0 & 0 & 4 & 4 & 0 & 0 & 0 & 0 \\
0 & 0 & 4 & 0 & 0 & 0 & 2 & -3 & 0 \\
9 & 0 & 4 & 0 & 0 & 0 & 0 & 0 & 0 \\
2 & 3 & 0 & 0 & 0 & 0 & 8 & 10 & 2 \\
0 & 0 & 0 & 2 & 0 & 8 & 0 & 0 & 0 \\
4 & 5 & 0 & -3 & 0 & 10 & 0 & 0 & 0 \\
0 & 2 & 0 & 0 & 0 & 2 & 0 & 0 & 0
\end{pmatrix}
\end{array}
$$



Wenden Sie den Algorithmus von Jarník/Prim auf G ausgehend von Knoten d an, um einen Spannbaum T mit *maximalem* Gewicht zu berechnen. Erstellen Sie dazu eine Tabelle mit zwei Spalten und stellen Sie jeden einzelnen Schritt des Verfahrens in einer eigenen Zeile dar. Geben Sie in der ersten Spalte denjenigen Knoten v an, der vom Algorithmus als nächstes in T aufgenommen wird (dieser sog. „schwarze" Knoten ist damit fertiggestellt). Führen Sie in der zweiten Spalte alle anderen vom aktuellen Baum T direkt erreichbaren Knoten v (sog. „graue Randknoten") auf.

Geben Sie in der Tabelle Knoten stets als Tripel $(v, \delta, v.\pi)$ an, mit v als Knotenname, $v.\pi$ als aktueller Vorgängerknoten (anderer Knoten der Kante) und δ als Länge der Kante $\{v, v.\pi\}$.

(b) Sei $G = (V, E, w)$ ein Graph mit Kantenlängen $w : E \to \mathbb{N}$ und T ein Spannbaum von G mit maximalem Gewicht. Beweisen oder widerlegen Sie die folgende Aussage:

Längste (einfache) Wege zwischen zwei Knoten $u, v \in V$ enthalten nur Kanten aus T.