

# Algorithmen und Datenstrukturen

## 8. Übungsblatt

Zeitraum: 4.–8. Dezember 2017

### Übung 1 (AGS 6.1.8)

Wenden Sie auf die Folge 9, 13, 7, 6, 10 den Quicksort-Algorithmus an.

### Übung 2 (AGS 6.2.11)

Wenden Sie auf die Folge 2, 4, 17, 9, 13, 20, 12, 8, 5, 18 den Heapsort-Algorithmus an. In Phase 2 müssen nur zwei Doppelschritte ausgeführt werden.

### Übung 3 (AGS 7. 10)

- (a) Geben Sie zu dem Pattern a a b a a a c a a b die mit Hilfe des KMP-Algorithmus (Knuth-Morris-Pratt) berechnete Verschiebetabelle an.
- (b) Mit Hilfe des KMP-Algorithmus ist die unten stehende Verschiebetabelle berechnet worden: Vervollständigen Sie das aus den Symbolen a, b und c bestehende Pattern.

Position	0	1	2	3	4	5
Pattern	c	b				a
Tabelle	-1	0	-1	1	0	2

### Übung 4

Gegeben seien die Wörter  $w = \text{Dinstas}$  und  $v = \text{Distanz}$ .

- (a) Berechnen Sie die Levenshtein-Distanz  $d(w, v)$  zwischen  $w$  und  $v$ . Geben Sie die Berechnungsmatrix vollständig an.
- (b) Geben Sie drei Alignments mit minimaler Levenshtein-Distanz zwischen  $w$  und  $v$  an.

### Zusatzaufgabe 1 (AGS 7.13)

- (a) Geben Sie zu dem Pattern a b b a b b a a die mit Hilfe des KMP-Algorithmus berechnete Verschiebetabelle an.
- (b) Mit Hilfe des KMP-Algorithmus ist die unten stehende unvollständige Verschiebetabelle berechnet worden. Die mit einem „?“ markierte Einträge sind unbekannt. Vervollständigen Sie das aus den Symbolen a, b und c bestehende Pattern.

Position	0	1	2	3	4	5
Pattern	b					c
Tabelle	-1	?	?	0	?	3

### Zusatzaufgabe 2 (AGS 3.2.39)

- (a) Schreiben Sie ein C-Programm, welches den Nutzer zur Eingabe einer Zahl  $n \geq 0$  auffordert. Ist  $n \geq 2$ , so sollen die Primfaktoren von  $n$  ausgegeben werden. Für  $n = 0$  bzw.  $n = 1$  soll keine Ausgabe erfolgen.

*Beispiel:* Ist  $n = 20$ , dann soll "2 2 5 " ausgegeben werden.

Nehmen Sie für die Teilaufgaben (b) und (c) folgenden Datentyp für Binärbäume an.

```
typedef struct node *tree;  
struct node { int value; tree left; tree right; };
```

- (b) Ein Binärbaum  $t$  ist *balanciert*, falls an jedem Knoten  $n$  von  $t$  die Höhen des linken und des rechten Teilbaums von  $n$  maximal um 1 voneinander abweichen. Schreiben Sie eine Funktion `int isBalanced(tree t)`, welche 1 zurückgibt, falls  $t$  balanciert ist, und sonst 0. Falls Sie eigene Hilfsfunktionen verwenden, geben Sie diese vollständig an.
- (c) Implementieren Sie eine Funktion `tree makeBalanced(int n)`, welche einen balancierten Binärbaum (siehe (b)) mit genau  $n$  Knoten anlegt. Die Werte an den Knoten des Baums können Sie dabei beliebig wählen. Nutzen Sie die Funktion `malloc` zur Allokation von Speicher.