

# Algorithmen und Datenstrukturen

SoSe 2018



## 4. Lösungsblatt — 07.05.2018 v1.0

### P1 Stacks

- a) Illustrieren Sie jeweils das Ergebnis nach jeder Ausführung der folgenden Operationen in angegebener Reihenfolge auf demselben Stack  $S$ , das in einem Array  $S[1 \dots 8]$  gespeichert wird und geben Sie den jeweiligen Index des Elements an auf das  $S.top$  nach Ausführung des Befehls zeigt. Zu Anfang ist der Stack  $S$  leer und  $S.top$  enthält den Wert 0.

	1	2	3	4	5	6	7	8
$S$								

- $PUSH(S, 9)$
- $PUSH(S, 4)$
- $PUSH(S, 7)$
- $POP(S)$
- $PUSH(S, 2)$
- $POP(S)$
- $POP(S)$

- b) Was passiert jeweils wenn der Befehl  $POP(S)$  auf einem leeren Stack oder  $PUSH(S, x)$  auf einem bereits vollen Stack ausgeführt wird?

- c) Geben Sie die Laufzeiten für die Befehle  $STACK-EMPTY(S)$ ,  $PUSH(S, x)$  und  $POP(S)$  in O-Notation an.

### Lösung.

- a) Siehe Tabelle

	1	2	3	4	5	6	7	8	
9									$S.top = 1$
9	4								$S.top = 2$
9	4	7							$S.top = 3$
9	4								$S.top = 2$
9	4	2							$S.top = 3$
9	4								$S.top = 2$
9									$S.top = 1$

- b) Es wird ein (Stack-)underflow bzw. ein (Stack-)overflow Fehler zurückgeworfen.  
c) Die einzelnen Operationen haben konstante Laufzeit.

$STACK-EMPTY(S)$	$O(1)$
$PUSH(S, x)$	$O(1)$
$POP(S)$	$O(1)$



- (Liste kann einfach umgekehrt werden.)

Vorteile *einfach* verkettete Liste:

- Weniger Speicherbedarf pro Element, da nur Zeiger auf Nachfolger gespeichert wird. Bei doppelt verketteten Listen jeweils Zeiger auf Vorgänger und Nachfolger.

(Zusatz: Es ist auch möglich doppelt verkettete Listen mit nur einem Zeiger umzusetzen, siehe: XOR linked list)

- Weniger Arbeitsschritte, da bei allen Befehlen wie Einfügen und Löschen nur ein Zeiger pro Element angepasst werden muss. Bei doppelt verketteten Listen jeweils die Zeiger auf Vorgänger und Nachfolger, was zu erhöhtem Arbeitsaufwand führt.

b) Die  $PUSH(L,x)$  ist genau diesselbe Funktion wie  $LIST-INSERT(L,x)$  bei Listen. Die  $POP(L)$  Funktion setzt  $y = L.head$  und ruft  $LIST-DELETE(L,L.head)$  auf. Zum Schluss wird der Wert  $y$  zurückgegeben.

c) Siehe Pseudocode

---

#### Algorithm 1 Reverse List

---

```

1: procedure REVERSE( $L$ )
2:    $a = L.head$                                 ▷  $a$  zeigt auf das erste Element der Liste
3:    $b = a.next$                                 ▷  $b$  zeigt auf das zweite Element der Liste
4:    $a.next = null$                               ▷ Lösche Pointer von  $a$  nach  $b$ 
5:   while  $b \neq null$  do
6:      $tmp = b.next$ 
7:      $b.next = a$                                 ▷ Füge Pointer von  $b$  nach  $a$  hinzu
8:      $a = b$ 
9:      $b = tmp$                                 ▷ setze  $a$  und  $b$  jeweils ein ein Listenelement weiter
10:  end while
11:   $L.head = a$                                 ▷ Setze neuen Listen Head auf  $a$ 
12: end procedure

```

---



---

## H1 Stacks und Queues

---

- a) Zeigen Sie wie man eine Queue mit Hilfe von zwei Stacks realisieren kann. Geben Sie außerdem die Laufzeit der Methoden ENQUEUE und DEQUEUE an.
- b) Zeigen Sie wie man einen Stack mit Hilfe von zwei Queues realisieren kann. Geben Sie außerdem die Laufzeit der Methoden PUSH und POP an.

### Lösung.

- a) ENQUEUE wird umgesetzt indem man ein Element auf den ersten Stack  $S1$  pusht. (ENQUEUE:  $O(1)$ ) DEQUEUE funktioniert wie folgt: Wir rufen  $POP(S2)$  auf. Falls Stack  $S2$  jedoch leer ist, wird für jedes Element in Stack  $S1$   $POP(S1)$  aufgerufen und in Stack  $S2$  gepusht. Zum Schluss wird mit  $POP(S2)$  das oberste Element zurückgegeben. (DEQUEUE  $O(n)$  im Worst Case)
- b) Solange Elemente in den Stack gepusht werden sollen, werden diese per ENQUEUE in  $Q1$  abgelegt. (PUSH:  $O(1)$ ) Sobald ein POP Befehl auf dem Stack ausgeführt werden soll, werden alle Elemente von  $Q1$  bis auf das letzte per DEQUEUE aus  $Q1$  entnommen und per ENQUEUE in  $Q2$  abgelegt. Das letzte Element aus  $Q1$  wird dann zurückgegeben und  $Q1$  ist nun leer. Dies kann dann für jeden POP Befehl in beide Richtungen ausgeführt werden ( $Q1 \rightarrow Q2$  und  $Q2 \rightarrow Q1$ ). (POP:  $O(n)$ )

---

## H2 Linked Lists

---

Gegeben ist die folgende Sequenz von Elementen:  $S = \langle 24, 5, 7, 12, 2, 13 \rangle$ .

Illustrieren Sie wie die Elemente in einer doppelt verketteten Liste gespeichert werden, das umgesetzt ist als:

1. ein multidimensionales Array
2. ein eindimensionales Array

**Lösung.**

Mögliche Lösungen:

(Lösungen können variieren, da keine Sortierung in Linked List und in implementierten Arrays)

1. Variable L = 2 (Index auf das Head der Linked List zeigt)

Array index	1	2	3	4	5	6	7
next	/	3	4	5	6	7	/
key		24	5	7	12	2	13
prev		/	2	3	4	5	6

In der Zeile key stehen die in der Liste gespeicherten Werte. Die Werte next bzw. prev speichern für jeden Wert die Pointer (die Position im Array) auf das folgende bzw. vorherige Element der Liste.

2. Variable L = 4 (Index auf das Head der Linked List zeigt)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	20	21	22
			24	7	/	5	10	4	7	13	7	12	16	10	2	19	13	13	/	16
						↑	↑	↑												
						key	next	prev												

Für jedes Element sind drei Zellen des Array nötig um neben dem Wert (key) die beiden Pointer (next und prev) zu speichern. Das zweite Element (5) steht an Position 7 im Array. An Position 8 steht die Array Position (10) des folgenden Elements (7) und an Position 9 die Position (4) der vorherigen Elements (24).