

# Übung 7

## Tutoraufgabe 1 (Optimaler Suchbaum):

Gegeben sind folgende Knoten mit dazugehörigen Zugriffswahrscheinlichkeiten:

Knoten	$I_0$	$N_1$	$I_1$	$N_2$	$I_2$	$N_3$	$I_3$	$N_4$	$I_4$	$N_5$	$I_5$
Werte	$(-\infty, 1)$	1	(1,2)	2	(2,3)	3	(3,4)	4	(4,5)	5	$(5, \infty)$
Wahrscheinlichkeit	0	0.4	0	0.2	0	0.1	0	0.1	0	0.2	0

Konstruieren Sie einen optimalen Suchbaum wie folgt.

- a) Füllen Sie untenstehende Tabellen für  $W_{i,j}$  und  $C_{i,j}$  nach dem Verfahren aus der Vorlesung aus. Geben Sie in  $C_{i,j}$  ebenfalls **alle möglichen Wurzeln** des optimalen Suchbaums für  $\{i, \dots, j\}$  an.

$W_{i,j}$	0	1	2	3	4	5
1						
2	—					
3	—	—				
4	—	—	—			
5	—	—	—	—		
6	—	—	—	—	—	

  

$C_{i,j} (R_{i,j})$	0	1	2	3	4	5
1		( )	( )	( )	( )	( )
2	—		( )	( )	( )	( )
3	—	—		( )	( )	( )
4	—	—	—		( )	( )
5	—	—	—	—		( )
6	—	—	—	—	—	

- b) Geben Sie einen optimalen Suchbaum für die Knoten mit den gegebenen Zugriffswahrscheinlichkeiten und der gegebenen Reihenfolge der Knoten graphisch an.
- c) Ist der optimale Suchbaum für die Knoten mit den gegebenen Zugriffswahrscheinlichkeiten und der gegebenen Reihenfolge der Knoten eindeutig? Geben Sie dazu eine kurze Begründung an.

## Tutoraufgabe 2 (Radixsort):

Sortieren Sie das folgende Array mithilfe von Radixsort. Geben Sie dazu das vollständige Array nach jedem Sortierschritt an. Hierbei stellt ein Sortierschritt das einmalige Anwenden eines stabilen Sortierverfahrens auf eine der Schlüsselpositionen dar.

Sie dürfen ein beliebiges stabiles Sortierverfahren für die einzelnen Schritte verwenden. Es ist nicht notwendig dazu Zwischenschritte anzugeben.

B	A	C	H
B	A	N	D
L	I	E	S
L	I	S	T
L	A	C	H
L	I	E	F
B	U	N	D
L	I	S	A
B	U	C	H
L	A	N	D

### Tutoraufgabe 3 (k-Sort):

Im Folgenden beschäftigen wir uns mit fast sortierten Arrays, bei denen die Position aller Einträge nur um einen konstanten Wert von der Zielposition abweicht.

Sei  $a$  ein duplikatfreies Array der Länge  $n$  und  $a'$  das Array, welches durch das Sortieren von  $a$  entsteht. Sei außerdem  $V$  die Menge aller Elemente in  $a$ . Wir nennen  $i_e$  den Index des Elements  $e$  im Array  $a$  und  $i'_e$  den Index des Elements  $e$  im Array  $a'$  für alle  $e \in V$ . Es gilt also  $a[i_e] = e = a'[i'_e]$ . Wir nennen  $a$  nun  $k$ -sorted gdw.  $|i'_e - i_e| < k$  für alle  $e \in V$ .

Das folgende Beispiellarray  $b$  ist 3-sorted, aber nicht 2-sorted:

$$b \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 2 & 4 & 1 & 3 & 7 & 5 & 8 & 6 \\ \hline \end{array}$$

$$b' \quad \begin{array}{|c|c|c|c|c|c|c|c|} \hline 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline \end{array}$$

Betrachten Sie nun den ksort-Algorithmus, welcher ein Array  $a$ , welches  $k$ -sorted ist vollständig sortiert.

```
def ksort(a, k):
    previous = -1
    for next in range(2 * k, len(a) + 1, 2 * k):
        swap(a, next - 2 * k, next - 1)
        current = partition(a, previous + 1, next - 1)
        heapsort(a, previous + 1, current - 1)
        previous = current
    heapsort(a, previous + 1, len(a) - 1)
```

Dazu nutzt ksort die Funktion  $\text{swap}(a, p_1, p_2)$ , welche schlicht im Array  $a$  die Werte an den Positionen  $p_1$  und  $p_2$  vertauscht.

Weiterhin nutzt ksort die Funktion  $\text{partition}(a, \text{left}, \text{right})$ , welche aus der Vorlesung für den Quicksort-Algorithmus bekannt ist und welche immer  $\text{right}$  als initiale Pivot-Position wählt, dann die Partitionierung durchführt und anschließend die neue Pivot-Position zurückgibt.

Außerdem nutzt ksort die Funktion  $\text{heapsort}(a, \text{left}, \text{right})$ , welche eine Variante des aus der Vorlesung bekannten Heapsort-Algorithmus darstellt mit dem einzigen Unterschied, dass nur das Teilarray zwischen  $\text{left}$  (inklusive) und  $\text{right}$  (inklusive) sortiert wird. Ansonsten hat die Funktion die selben Eigenschaften welche aus der Vorlesung für Heapsort bekannt sind.

- Ist ksort ein in-place Verfahren? Begründen Sie Ihre Antwort.
- Zeigen Sie, dass die Worst-Case Laufzeit von ksort nach oben durch  $O(n \cdot \log(k))$  beschränkt ist. Sie dürfen dazu die aus der Vorlesung bekannten Eigenschaften für die Funktionen `swap`, `partition` und `heapsort` annehmen.
- Begründen Sie, warum der Aufruf `ksort(a, k)` das Array  $a$  vollständig sortiert, falls dieses vorher bereits  $k$ -sorted war.

### Tutoraufgabe 4 (Abstrakte Datentypen):

Ein abstrakter Datentyp wird definiert durch die Spezifikationen der Methoden mit denen man auf diesen Datentyp zugreifen kann. Daher kann ein abstrakter Datentyp mehr als eine Implementierung haben, die unterschiedliche Vor- oder Nachteile, insbesondere unterschiedliche Laufzeitkomplexitäten haben können. Wir haben in dieser Aufgabe abstrakte Datentypen (mit möglicherweise etwas anderen Methodennamen) mit einer unüblichen und ineffizienten Implementierung angegeben. Welche abstrakten Datentypen haben wir hier implementiert?

- `PRIORITY_QUEUE` ist eine Prioritätswarteschlange, mit den aus der Vorlesung bekannten Methode.

```
class ADT_one:
    def __init__(self):
        self.prio_queue = PRIORITY_QUEUE()
        self.index = 0

    def put(x, adt_one):
        prio_item = (adt_one.index, x)
```

```

    enqq(prio_item, adt_one.prio_queue)
    adt_one.index = adt_one.index+1

def take(adt_one):
    (prio, item) = get(adt_one.prio_queue)
    deqq(adt_one.prio_queue)
    return item

```

- i. Geben Sie an, welcher abstrakte Datentyp hier implementiert wurde. Begründen Sie auch ihre Antwort.
- ii. Wie könnte der abstrakte Datentyp besser implementiert werden?

**b)** DoubleLinkedList ist eine doppelt verkettete Liste mit den aus der Vorlesung und Übung bekannten Methoden.

```

class ADT_two:
    def __init__(self):
        self.double_list = DoublyLinkedList()

def compute(x, adt_two):
    reset(adt_two.double_list)
    (c1,c2) = (None,None)
    while x != c1 and not isLast(adt_two.double_list):
        next(adt_two.double_list)
        element = get(adt_two.double_list)
        (c1, c2) = element if element != "Sentinel" else (None,None)
    return c2

def put(x, y, adt_two):
    reset(adt_two.double_list)
    (c1,c2) = (None,None)
    while x != c1 and not isLast(adt_two.double_list):
        next(adt_two.double_list)
        element = get(adt_two.double_list)
        (c1,c2) = element if element != "Sentinel" else (None,None)
    if x == c1:
        delete(adt_two.double_list)
    insert((x,y),adt_two.double_list)

```

- i. Geben Sie an, welcher abstrakte Datentyp hier implementiert wurde. Begründen Sie auch ihre Antwort.
- ii. Wie könnte der abstrakte Datentyp besser implementiert werden?