Prof. Dr. Leif Kobbelt
Stefan Dollase, Ira Fesefeldt, Alexandra Heuschling, Gregor Kobsik

Übung 7

Tutoraufgabe 1 (Optimaler Suchbaum):

Gegeben sind folgende Knoten mit dazugehörigen Zugriffswahrscheinlichkeiten:

Knoten	l_0	N_1	l_1	N_2	l_2	N_3	l_3	N_4	l_4	N_5	l_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 $\{i,\ldots,j\}$ 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 Sortiertschritt an. Hierbei stellt ein Sortiertschritt 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 Beispielarray b ist 3-sorted, aber nicht 2-sorted:

b	2	4	1	3	7	5	8	6

b'	1	2	3	4	5	6	7	8

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 `swap(a, p1, p2)`, welche schlicht im Array `a` die Werte an den Positionen `p1` und `p2` vertauscht.
Weiterhin nutzt `ksort` die Funktion `partition(a, left, right)`, welche aus der Vorlesung für den Quicksort-Algorithmus bekannt ist und welche immer `right` als initiale Pivot-Position wählt, dann die Partitionierung durchführt und anschließend die neue Pivot-Position zurückgibt.
Außerdem nutzt `ksort` die Funktion `heapsort(a, left, right)`, welche eine Variante des aus der Vorlesung bekannten Heapsort-Algorithmus darstellt mit dem einzigen Unterschied, dass nur das Teilarray zwischen `left` (inklusive) und `right` (inklusive) sortiert wird. Ansonsten hat die Funktion die selben Eigenschaften welche aus der Vorlesung für Heapsort bekannt sind.

a) Ist `ksort` ein in-place Verfahren? Begründen Sie Ihre Antwort.

b) 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.

c) 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?

a) `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 verkette 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?

Aufgabe 5 (Optimaler Suchbaum): 7 + 2 + 1 = 10 Punkte

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	0.1	0	0.2	0	0.1	0	0.3	0	0.3	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 $\{i,\ldots,j\}$ 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.

Aufgabe 6 (Bucketsort): 10 Punkte

Sortieren Sie das folgende Array mithilfe von Bucketsort. Geben Sie dazu an, welche Buckets Sie verwenden, für welche Intervalle diese stehen und welche Elemente in welcher Reihenfolge in diese Buckets eingefügt werden. Geben Sie zusätzlich den Inhalt der Bucket an, nachdem diese sortiert worden sind. Zuletzt geben Sie das vollständig sortierte Array an.

Sie dürfen ein beliebiges Sortierverfahren nutzen um die einzelnen Buckets zu sortieren. Es ist nicht notwendig dazu Zwischenschritte anzugeben.

0.12	0.26	0.69	0.86	0.93	0.52	0.34	0.25	0.43	0.31

Aufgabe 7 (Mastertheorem - Wiederholung): 2 + 2 + 2 + 2 + 2 = 10 Punkte

Bestimmen Sie mithilfe des Mastertheorems die beste mögliche O-Klasse für folgende Rekursionsgleichungen. Geben Sie zusätzlich an, welches p Sie gewählt haben. Begründen Sie, warum weder durch eine andere Wahl von p, noch durch die Anwendung eines anderen Falls des Mastertheorems eine bessere O-Klasse erreicht werden kann.

a)

$$T(1) = 1$$
$$T(n) = 2 \cdot T(\frac{n}{4}) + n + 4 \qquad \text{für } n > 1$$

b)

$$T(1) = 1$$
$$T(n) = 9 \cdot T(\frac{n}{3}) + n \cdot \sqrt{n} \qquad \text{für } n > 1$$

c)

$$T(1) = 1$$
$$T(n) = 16 \cdot T(\frac{n}{2}) + 12 \cdot n^4 + 16 \cdot n^3 + 100 \qquad \text{für } n > 1$$

d)

$$T(1) = 1$$
$$T(n) = 6 \cdot T(\frac{n}{6}) + \frac{n}{2} \qquad \text{für } n > 1$$

e)

$$T(1) = 1$$

$$T(n) = 8 \cdot T(\frac{n}{4}) + \frac{n^2}{6} + n \cdot \sqrt{n} \qquad \text{für } n > 1$$

Aufgabe 8 (Abstrakte Datentypen): 5 + 5 = 10 Punkte

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?

a) Die Methoden `insert`, `delete` und `max` sind die Methoden für binäre Suchbäume wie in der Vorlesung und den Übungen vorgestellt.

```
class ADT_three:
    def __init__(self):
        self.root = None

def push(x,p,adt_three):
    adt_three.root = insert(adt_three.root,p,x)

def pop(adt_three):
    if adt_three.root is None:
        return None
    m = max(adt_three.root)
    adt_three.root = delete(m.key,adt_three.root)
    return m.value
```

Geben Sie an, welcher abstrakte Datentyp hier implementiert wurde. Begründen Sie auch ihre Antwort.

b)

```
class ADT_four:
    def __init__(self):
        self.dict_1 = {}
        self.dict_2 = {}

def all(adt_four):
    all_list = []
    for key in adt_four.dict_2.keys():
        if adt_four.dict_2[key]:
            all_list.append(key)
    return all_list

def all_n(v,adt_four):
    i = 0
    all_n_list = []
    while str(v)+str(i) in adt_four.dict_1:
        all_n_list.append(adt_four.dict_1[str(v)+str(i)])
        i += 1
    return all_n_list

def add(v,adt_four):
    adt_four.dict_2[v] = True

def connect(v1,v2,adt_four):
    i=0
    while str(v1)+str(i) in adt_four.dict_1 and adt_four.dict_1[str(v1)+str(i)] != v2:
        i += 1
    adt_four.dict_1[str(v1)+str(i)] = v2
```

Geben Sie an, welcher abstrakte Datentyp hier implementiert wurde. Begründen Sie auch ihre Antwort.