Aufgabensammlung ADS-Repetitorium 2018

Rekursive Laufzeit – Datenstrukturen

Aufgabe 1: Pseudocode – Spot the Error

Die folgenden Algorithmen berechnen nicht das, was sie sollen. Erklären Sie, was der Fehler ist und schreiben Sie den richtigen Algorithmus auf. Geben Sie auch die asymptotische Worst-Case-Laufzeit in Θ-Notation an.

(a) Der Algorithmus soll $f(n) = n!$ berechnen.

Algorithmus 1: int Algo3(int n)

```
return Algo3(n − 1) · n
```

(b) Der Algorithmus soll **True** zurückgeben, wenn i im Array A enthalten ist, sonst **False**.

Algorithmus 2: boolean Algo4(int i, int[] A, int $l = 0$)

```
if A.length == l then
    return False
else
    return (i == A[l]) or Algo4(i, l + 1)
```

Aufgabe 2: Sortieralgorithmen

(a) MergeSort arbeitet rekursiv. Geben Sie für das Feld C = [9, 4, 1, 3, 5, 2, 6, 0, 8, 7] den Rekursionsbaum von MergeSort an. In jedem Knoten soll der jeweilige Aufruf von MergeSort und die zu sortierende Teilliste stehen, jeweils *vor* der Sortierung.

(b) Sortieren Sie das Feld D = [6, 4, 7, 9, 2, 3, 1, 5, 0, 8] mit einem vereinfachten QuickSort. Dieser schreibt alle Elemente, die kleiner als das Pivotelement sind, links und alle größeren rechts neben das Pivotelement. Zeichnen Sie den Rekursionsbaum. Schreiben Sie in jeden Knoten das zu sortierende Teilfeld nach dem Aufruf von Partition und markieren Sie das Pivotelement, die Wurzel sieht also so aus: [6, 4, 7, 0, 2, 3, 1, 5, $\underline{8}$, 9]. *Achtung: Das ist nicht der VL-Algorithmus, aber die Idee ist die gleiche!*

Aufgabe 3: Induktionsbeweis eines Algorithmus und Laufzeit

Betrachten Sie den folgenden Algorithmus zur Berechnung der Fakultät.

Algorithmus 3: int Fakultaet(**int** k)

```
if k = 0 then
    return 1
return k · Fakultaet(k − 1)
```

(a) Beweisen Sie mittels vollständiger Induktion die Korrektheit Ihrer rekursiven Variante.

(b) Sei folgender Algorithmus zur Berechnung des Produkts $i \cdot (i+1) \cdot \ldots \cdot (j-1) \cdot j$ für natürliche Zahlen i und j mit $i < j$ gegeben:

Algorithmus 4: int Produkt(int j, int i)

```
return Fakultaet(j)/Fakultaet(i − 1)
```

Begründen Sie kurz, warum der Algorithmus Produkt korrekt ist. Geben Sie die Worst-Case-Laufzeit von Produkt in Abhängigkeit von i und j an.

Aufgabe 4: Rekursive Gleichung aufstellen
Gegeben sei folgender Algorithmus:

Algorithmus 5: RecursiveAlgo(**int** A[ ], l= 1, r=A.length)

```
if l < r then
    m = ⌊(l + r)/2⌋
    RecursiveAlgo(A, l, m)
    RecursiveAlgo(A, m + 1, r)
    InsertionSort(A, l, r)
```

(a) Stellen Sie eine Rekursionsgleichung T für den gegebenen Algorithmus auf.

(b) Finden Sie eine Funktion f, für die $T \in \Theta(f)$ gilt.

Aufgabe 5: Rekursive Laufzeiten
Finden Sie für die nachstehenden Rekursionsgleichungen jeweils eine Funktion f, für die $T \in \Theta(f)$ gilt. Sie können davon ausgehen, dass die Laufzeit im Basisfall konstant ist.

(a) $T(n) = 4T(\lfloor n/2 \rfloor) + \frac{1}{2}n^2\sqrt{n}$

(b) $T(n) = 4T\lfloor n/2 \rfloor) + n^2 \log n + n$

(c) $T(n) = T(n-3) + 2n$

(d) $T(n) = 2T(\lfloor n/4 \rfloor) + 3\sqrt{n}$

(e) $T(n) = 3T(\lfloor n/2 \rfloor) + \frac{n}{6}$

(f) $T(n) = 3T(\lfloor n/5 \rfloor) + \frac{1}{2}\sqrt{n}$

(g) $T(n) = 12T(\lfloor n/2 \rfloor) + n^4$

(h) $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \mathcal{O}(n \log n)$

Aufgabe 6: Rekursive Algorithmen
Geben Sie jeweils einen Algorithmus an, der die gegebene Laufzeit erfüllt. Alle Algorithmen sollen im Basisfall, also für $n = 1$, konstante Laufzeit haben. Als Eingabe bekommen die Algorithmen jeweils ein Feld der Länge n. Die Algorithmen sollten eine Rückgabe haben, doch der Wert der Rückgabe ist egal. Sie dürfen Rundungsfehler ignorieren.

(a) $T(n) = 3T(n-1) + n$

(b) $T(n) = 4T(n/4) + O(n)$

(c) $T(n) = T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + T(\sqrt{n})$; bitte beachten Sie hier die Rundungen.

(d) $T(n) = 3T(n-5) + n \log n$

(e) $T(n) = T(n-1) + T(n-2) + 42T(n/2) + n^2$

Aufgabe 7: Sortieren in Linearzeit

(a) Die Eingabe sei eine Liste von t Feldern $A_1, \ldots, A_t$ mit einer Gesamtlänge von $n = \sum_{i=1}^{t} A_i.length$. Der Wertebereich ist für jedes Feld $\{1, \ldots, k\}$.

Geben Sie in Worten einen Algorithmus an, der alle Felder in insgesamt $O(n+k)$ Zeit sortiert! Jedes Feld soll am Ende die gleichen Elemente enthalten wie zu Beginn (nur in sortierter Reihenfolge). Begründen Sie die Laufzeit.

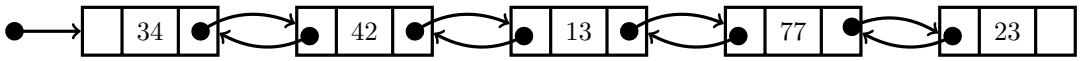
(b) Geben Sie in kommentiertem Pseudocode einen Algorithmus an, der ein Eingabefeld A der Länge n mit Wertebereich $\{1, \ldots, n^2 - 1\}$ in $O(n)$ Zeit sortiert!

Tipp: Verwenden Sie Methoden aus der Vorlesung. Wandeln Sie jedoch zunächst die Zahlen in eine geeignete Darstellung um.

Aufgabe 8: Doppelt-Verkettete Listen
Gegeben sei folgender Algorithmus

Algorithmus 12: modifyList(List L)

```
1  item = L.head
2  size = 1
3  while item.next != null do
4      item = item.next
5      size = size + 1
6  item = L.head
7  for i = 1 to ⌊size/2⌋ do
8      item = item.next
9      item.prev.next = item.next
10     item.next.prev = item.prev
11     item = item.next
```

(a) Sei $L =$ 

.

Zeichnen Sie die Liste für jede Iteration der Schleife in Zeile 7.

(b) Beschreiben Sie, was der Algorithmus allgemein macht.

(c) Der Algorithmus enthält zwei Fehler. Geben Sie eine Liste an, sodass die Ausführung von Zeile 3 fehlschlägt. Geben Sie außerdem eine Liste an, die zu einem Fehler in Zeile 10 führt. Verbessern Sie den Pseudocode.

(d) Was würde passieren, wenn Zeile 11 gelöscht würde?

(e) Welche Augmentierung der Datenstruktur List schlagen Sie vor, um den Code zu verkürzen?

Ein Ring ist eine Datenstruktur, die auf einer doppelt-verketteten Liste aufbaut. Der Unterschied zwischen beiden Datenstrukturen ist, dass beim Ring die Attribute next und prev niemals nil sind und über next eines beliebigen Elements jedes andere Element erreicht werden kann (analog auch über prev in die andere Richtung). Jeder Ring hat einen Pointer entry auf einen beliebiges Element im Ring. Ansonsten gibt es die gleichen Operationen wie bei der Liste, wobei insert(k) vor dem aktuellen entry einfügt und dann entry aufs neue Item setzt.

Aufgabe 9: Ringe

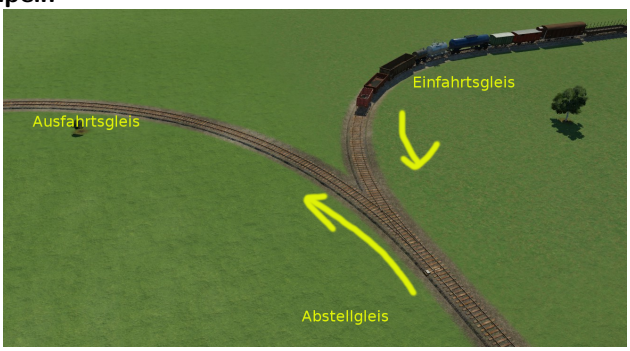
(a) Zeichnen Sie den Ring, der die ersten vier Fibonacci-Zahlen enthält. Der Pointer entry soll auf eine gerade Primzahl zeigen.

(b) Implementieren Sie die Methode makeRing(List l), die aus einer doppelt-verketteten Liste einen Ring macht. Der Pointer entry des entstanden Rings soll dabei auf den Kopf der ursprünglichen Liste zeigen. Die Liste darf verändert werden.

(c) Implementieren Sie die Methode split(Ring r, Item i, Item j), die den Ring r in zwei Ringe aufspaltet. Dabei sollen alle Items zwischen i und j (inklusive, in Richtung des next-Attributs) aus r gelöscht werden und als eigener Ring zurückgegeben werden. Weisen Sie die entry-Werte beliebig, aber gültig, zu. Gehen Sie davon aus, dass mindestens ein Element in r verbleibt (mit anderen Worten i.prev $\neq$ j).

(d) Implementieren Sie die Methode merge(Ring r, Ring u), die die Items des Rings u vor r.entry einfügt. Achten Sie darauf, dass die Reihenfolge der Elemente innerhalb der Ringe gleich bleibt.

Aufgabe 10: Waggons stapeln



Wir betrachten einen Zug mit n verschiedenen Güterwagons, die mit den Zahlen 1 bis n aufsteigend beschriftet sind. Wir betrachten folgenden Algorithmus:

1. Nimm vom Anfang des Zuges eine zufällige Anzahl von Waggons.
2. Schiebe diese Waggons auf das Abstellgleis.
3. Nimm eine zufällige Anzahl von Waggons auf dem Abstellgleis und schiebe sie aufs Ausfahrtsgleis.
4. Wiederhole ab 1., bis alle Waggons auf dem Ausfahrtgleis sind.

(a) Formulieren Sie den obigen Algorithmus in Pseudocode. Die Eingabe ist eine Zahl n, die Ausgabe soll ein entsprechend permutiertes Feld der Zahlen 1 bis n sein.

(b) Geben Sie einen Algorithmus an, der für eine Waggonfolge entscheidet, ob diese durch dieses Verfahren zustande gekommen ist. Beispiel: Die Folge $3 - 2 - 4 - 1$ ist entstanden, indem zuerst die Wagons 1, 2 und 3 auf das Abstellgleis wanderten. Dann wurde Waggons 3 und 2 aufs Ausfahrtsgleis gestellt, danach Waggon 4 vom Einfahrtsgleis aufs das Ausfahrtsgleis umgeparkt und zuletzt Waggon 1 hinten an den Zug angehängt.

Aufgabe 11: Henne-Ei-Problem

(a) Implementieren Sie eine Queue mit den Operationen dequeue() und enqueue(key k), die intern zwei Stacks verwendet.

(b) Implementieren Sie einen Stack mit Operationen pop() und push(key k), der zwei Queues verwendet.

Aufgabe 12: DivContainer-Datenstruktur

In dieser Aufgabe sollen Sie eine Datenstruktur implementieren, die über zwei Operationen verfügt: insert(x) fügt eine beliebige, positive Zahl in die Datenstruktur ein und get(d), die eine beliebige Zahl aus der Datenstruktur ausgibt, die durch d teilbar ist.

(a) Geben Sie eine Implementation beider Methoden an, wenn get(d) keine Laufzeitbeschränkungen hat und die zurückgegebene Zahl nicht aus der Datenstruktur entfernt werden soll. Welche Laufzeiten haben ihre Methoden?

(b) Nun soll die zurückgegebene Zahl aus der Datenstruktur gelöscht werden. Wie müssen Sie Ihre Implementationen aus a) verändern, damit dies möglich wird? Ändern sich die Laufzeiten?

Aufgabe 13: Implementieren einer eigenen Datenstruktur

Gesucht ist eine Datenstruktur MinStack zum Verwalten einer dynamischen Menge S von Zahlen. Es sollen wie bei einem Stapel die Methoden push(key k) und pop() zur Verfügung stehen, zusätzlich eine Methode Minimum(), welche die kleinste Zahl der Menge S zurück gibt. Alle Operationen sollen in konstanter Zeit ablaufen. *Tipp*: Verwenden Sie intern mehr als eine Datenstruktur.

(a) Geben Sie eine Implementierung der Datenstruktur in Pseudocode an.

(b) Zeigen Sie, dass es keine Datenstruktur geben kann, die zusätzlich zu den obigen Operationen eine weitere Operation popMinimum() mit konstanter Laufzeit anbietet. Diese Operation löscht das aktuelle Minimum aus dem MinStack.

Aufgabe 14: Löschen in einer Hash-Tabelle

Gegeben sei eine Hashtabelle H mit einer Hash-Funktion $h(x, i)$. Es wird offene Adressierung verwendet.

(a) Beschreiben Sie in Worten, wie der Algorithmus Search(**int** k) aus der Vorlesung funktioniert.

(b) Ein Element k soll aus der Tabelle gelöscht werden. Warum sollte man den Wert nicht mit der folgenden Befehlsfolge löschen?
$j = \mathsf{Search}(k)$
$\mathsf{H}[j] = -1$

(c) Implementieren Sie die Operation Delete(**int** k), die einen Schlüssel aus der Tabelle löscht, ohne dass das Problem aus b) auftritt. *Tipp*: Verwenden Sie einen besonderen Wert, um gelöschte Zellen zu markieren.

(d) Welche Änderung muss nun in den Methoden Insert(**int** k) und Search(**int** k) vorgenommen werden?

(e) Beschreiben Sie kurz die Auswirkungen Ihrer Änderungen auf die Laufzeit der Operationen.

Aufgabe 15: Doppeltes Hashing

Welche der folgenden Funktionen eignen sich für eine Hashtabelle der Länge 25, wenn doppeltes Hashing verwendet wird und die Hashfunktion $h(k, i) = (h_0(k) + ih_1(k)) \bmod 25$ mit $h_0(k) = (4k + 2) \bmod 25$ ist? Begründen Sie Ihre Entscheidungen.

(a) $h_1(k) = 1$

(b) $h_1(k) = 9 - (k \bmod 4)$

(c) $h_1(k) = k \bmod 17$

(d) $h_1(k) = (3 + 5k) \bmod 25$

(e) $h_1(k) = (4k - 1) \bmod 13$

Aufgabe 16: Brainfuck-Interpreter (umfang- und lehrreich)

In dieser Aufgabe entwickeln wir einen Interpreter für die esoterische Programmiersprache *Brainfuck*.

(a) In unserer Version von Brainfuck besteht der Speicher aus einem Band mit theoretisch unendlich vielen Zellen, in denen ganze Zahlen stehen können. Brainfuck verwaltet einen Zeiger, der auf eine Zelle zeigt. Zu Beginn zeigt der Zeiger auf die erste Zelle des Bandes. Implementieren Sie eine Datenstruktur BFMemory, die die folgenden Operationen besitzt:

new BFMemory()	Erzeugt ein neues Band. In allen Zellen steht eine 0.
incrementPointer()	Schiebt den Zeiger auf die nächste Zelle.
decrementPointer()	Schiebt den Zeiger auf die vorherige Zelle.
incrementValue()	Erhöht den Wert der Zelle, auf der der Zeiger steht, um eins.
decrementValue()	Erniedrigt den Wert der Zelle, auf der der Zeiger steht, um eins.
int getCurrentCellValue()	Gibt den Wert der Zelle zurück, auf der der Zeiger gerade steht.

Keine der Operationen soll Fehler verursachen. Wenn der Zeiger auf eine Zelle verschoben wird, die nicht existiert, muss das Band vergrößert werden.

(b) Ein Brainfuck-Programm wird durch einen Array repräsentiert. Die Einträge des Arrays sind die Befehle. In unserer Brainfuck-Version erlauben wir sieben Befehle:

`>`	Verschiebt den Zeiger des Bands nach rechts.
`<`	Verschiebt den Zeiger des Bands nach links.
`+`	Inkrementiert den Wert der aktuellen Zelle.
`-`	Dekrementiert den Wert der aktuellen Zelle.
`[`	Falls der Wert der aktuellen Zelle 0 ist, springe hinter passendes `]`, ansonsten ignoriere Befehl.
`]`	Springe vor passendes `]`, welches als nächstes ausgewertet wird.
`.`	Gebe die Zelle, auf der der Zeiger steht, aus.

Welche Ausgabe hat folgendes Brainfuck-Programm?

```
++++++[>+++++++++++<-]>-.+++.<++++[>++++<-]>-.
```

(c) Geben Sie nun unter Verwendung Ihrer Datenstruktur BFMemory einen Algorithmus an, der ein Brainfuck-Programm als Array entgegen nimmt und dieses gemäß den obigen Regeln ausführt. Sie dürfen davon ausgehen, dass die eingegebenen Programme korrekt sind. *Tipp*: Verwenden Sie Rekursion.

(d) Sei $\mathbb{B} = \{x \in \mathbb{N} \mid 64 < x < 97\}$. Schreiben Sie einen Algorithmus, der als Eingabe eine Liste von Zahlen A mit $\mathsf{A}[i] \in \mathbb{B}$ für alle $i \leq \mathsf{A.length}$ erhält. Die Ausgabe Ihres Algorithmus soll ein gültiger Brainfuck-Code sein, der die Zahlen in A nacheinander ausgibt. Der Code muss *nicht* minimal kurz sein. Verwenden Sie eine verkettete Liste, um den Code Schritt für Schritt aufzubauen.