

Aufgabensammlung ADS-Repetitorium 2021

Rekursive Laufzeit – Datenstrukturen

Aufgabe 1: 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 2: 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) $T(n) = 4T(n/4) + \mathcal{O}(n)$
- (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 3: Rekursive Gleichung aufstellen

Gegeben sei folgender Algorithmus:

Algorithmus 6: RecursiveAlgo(int A[], l= 1, r=A.length)

```
1 if l < r then
2   m = ⌊(l+r)/2⌋
3   RecursiveAlgo(A, l, m)
4   RecursiveAlgo(A, m+1, r)
5   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 4: Löschen in einer Hash-Tabelle

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

- Beschreiben Sie in Worten, wie der Algorithmus **Search(int k)** aus der Vorlesung funktioniert.
- Ein Element k soll aus der Tabelle gelöscht werden. Warum sollte man den Wert nicht mit der folgenden Befehlsfolge löschen?
 $j = \text{Search}(k)$
 $H[j] = -1$
- 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.
- Welche Änderung muss nun in den Methoden **Insert(int k)** und **Search(int k)** vorgenommen werden?
- Beschreiben Sie kurz die Auswirkungen Ihrer Änderungen auf die Laufzeit der Operationen.

Aufgabe 5: 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.

- $h_1(k) = 1$
- $h_1(k) = 9 - (k \bmod 4)$
- $h_1(k) = k \bmod 17$
- $h_1(k) = (3 + 5k) \bmod 25$
- $h_1(k) = (4k - 1) \bmod 13$

Aufgabe 6: Doppelt-Verkettete Listen

Gegeben sei folgender Algorithmus

Algorithmus 8: 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

```



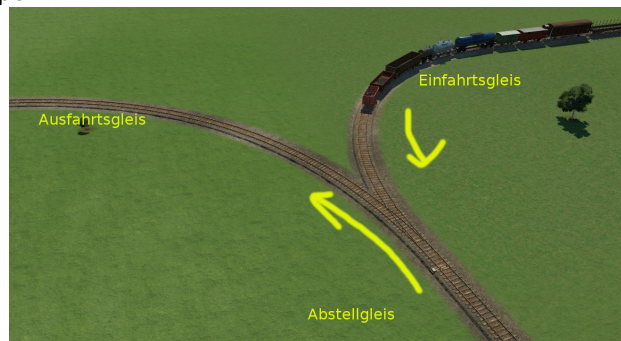
Zeichnen Sie die Liste für jede Iteration der Schleife in Zeile 7.

- Beschreiben Sie, was der Algorithmus allgemein macht.
- 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.
- Was würde passieren, wenn Zeile 11 gelöscht würde?
- Welche Augmentierung der Datenstruktur List schlagen Sie vor, um den Code zu verkürzen?

Aufgabe 7: Ringe

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.

- Zeichnen Sie den Ring, der die ersten vier Fibonacci-Zahlen enthält. Der Pointer `entry` soll auf eine gerade Primzahl zeigen.
- Implementieren Sie die Methode `makeRing(List l)`, die aus einer doppelt-verketteten Liste einen Ring macht. Der Pointer `entry` des entstandenen Rings soll dabei auf den Kopf der ursprünglichen Liste zeigen. Die Liste darf verändert werden.
- 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 $i.\text{prev} \neq j$).
- 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 8: Waggon stapeln

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

- Nimm vom Anfang des Zuges eine zufällige Anzahl von Waggons.
 - Schiebe diese Waggons auf das Abstellgleis.
 - Nimm eine zufällige Anzahl von Waggons auf dem Abstellgleis und schiebe sie aufs Ausfahrtsgleis.
 - Wiederhole ab 1., bis alle Waggons auf dem Ausfahrtsgleis sind.
- 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.
 - 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 Waggon 3 und 2 aufs Ausfahrtsgleis gestellt, danach Waggon 4 vom Einfahrtsgleis aufs Ausfahrtsgleis umgeparkt und zuletzt Waggon 1 hinten an den Zug angehängt.

Aufgabe 9: 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 10: 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 11: 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 12: 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:

<code>new BFMemory()</code>	Erzeugt ein neues Band. In allen Zellen steht eine 0.
<code>incrementPointer()</code>	Schiebt den Zeiger auf die nächste Zelle.
<code>decrementPointer()</code>	Schiebt den Zeiger auf die vorherige Zelle.
<code>incrementValue()</code>	Erhöht den Wert der Zelle, auf der der Zeiger steht, um eins.
<code>decrementValue()</code>	Erniedrigt den Wert der Zelle, auf der der Zeiger steht, um eins.
<code>int getCurrentCellValue()</code>	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 $A[i] \in \mathbb{B}$ für alle $i \leq A.length$ 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.