Aufgabensammlung ADS-Repetitorium WS 24/25

rekursive Algorithmen – elementare Datenstrukturen – Bäume

Aufgabe 1: Rekursive Gleichung aufstellen
Gegeben sei folgender Algorithmus:

Algorithmus 1: RecursiveAlgo(int A[ ], int l= 1, int 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 2: 3-MergeSort
Gegeben sei folgende Algorithmus 2, das eine Variante vom bekannten MergeSort ist.

Algorithmus 2: `3MergeSort`(**int**[] A, **int** $l = 1$, **int** $r = A.length$)

```
1 if l < r then
2     m1 = ⌊(r+l)/3⌋
3     m2 = ⌊2(r+l)/3⌋
4     3MergeSort(A, l, m1)
5     3MergeSort(A, m1 + 1, m2)
6     3MergeSort(A, m2 + 1, r)
7     Merge(A, l, m1, m2)
8     Merge(A, l, m2, r)
```

(a) Formulieren Sie die Laufzeit $T(n)$ von `3MergeSort` als Rekursionsgleichung in Abhängigkeit von der Länge des Feldes A.

(b) Bestimmen Sie die asymptotische Laufzeit von `3MergeSort` mithilfe der Meistermethode. Geben Sie dabei den Fall an.

Aufgabe 3: 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) $T(n) = T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + \Theta(n \log n)$

Aufgabe 4: Induktionsbeweis eines Algorithmus
Betrachten sie folgenden Algorithmus zur Berechnung der Fakultät:

Algorithmus 3: int fakultät(int k)

```
1 f = j = k
2 while j > 1 do
3 |   j = j − 1
4 |   f = f · j
5 return f
```

(a) Geben Sie einen Algorithmus in Pseudocode an, der die Fakultät rekursiv berechnet.

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

Aufgabe 5: Korrektheitsbeweise und Rekursion
Betrachten Sie den folgenden Algorithmus.

Algorithmus 5: int doSomethingSimple(**int** A[ ], **int** $i = 1$)

```
  Data: Feld mit natürlichen Zahlen A, natürliche Zahl i
  Result: Ein Wert, der mit A zusammenhängt
1 if i == A.length then
2 |   return A[i]
3 k = doSomethingSimple(A, i + 1);
4 if k > A[i] then
5 |   return k
6 else
7 |   return A[i]
```

(a) Beschreiben Sie in einem Satz, was der Algorithmus macht.

(b) Beweisen Sie die Korrektheit des Algorithmus.

(c) Geben Sie einen Algorithmus an, der äquivalent zu doSomethingSimple ist, ohne Rekursion zu verwenden.

(d) Geben Sie eine Schleifeninvariante für Ihren inkrementellen Algorithmus an.

(e) Beweisen Sie die Korrektheit Ihres Algorithmus mit der von Ihnen aufgestellten Schleifeninvariante.

Aufgabe 6: 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.

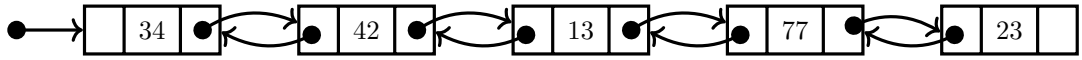
(a) Geben Sie eine Implementierung der Datenstruktur in Pseudocode an.

(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 7: Doppelt-Verkettete Listen
Gegeben sei folgender Algorithmus

Algorithmus 11: 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?

Aufgabe 8: 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?
$j =$ Search(k)
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 9: 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 10: Gute Sondierfolgen erkennen

Gegeben sind folgende Hashfunktionen $h(k, i)$. Geben Sie an, um welche Methode der Kollisionsauflösung es sich handelt und gegebenfalls Probleme der Sondierfolgen, bei den gegebenen Tabellengrößen, an.

(a) $h(k,i) = (h_0(k) + (i+1) \cdot i) \mod m$, $m = 512$

(b) $h(k,i) = (h_0(k) + 2i) \mod m$, $m = 511$

(c) $h(k,i) = (h_0(k) + i \cdot h_1(k)) \mod m$, $h_1(k) = m - (2k \mod m) - 1$, $m = 16$.

Aufgabe 11: Hashing ausführen

Gegeben seien die folgenden Schlüssel k mit entsprechenden Werten der Hashfunktion $h(k)$:

k	N	I	C	E	A	L	G	O
$h(k)$	6	3	1	3	3	1	7	2

(a) Fügen Sie die Schlüssel in obiger Reihenfolge in eine Hash-Tabelle der Größe 8 ein. Kollisionen sollen durch Verkettung aufgelöst werden.

(b) Fügen Sie nun die Werte in der gleichen Reihenfolge in eine neue Hashtabelle mit Größe 8 ein. Lösen Sie Kollisionen mit linearem Sondieren auf. Geben Sie für jeden Schlüssel an, wie viele Felder der Tabelle betrachtet wurden.

(c) Fügen Sie wieder die Schlüssel in eine Tabelle der Größe 8 ein. Verwenden Sie dafür folgende Hashfunktion $l(k,i)$, welche Kollisionen mittels doppelten Hashing auflöst. Wobei m die Tabellengröße ist.

$$l(k,i) = (h(k) + i \cdot h_1(k)) \mod m$$

Im Folgenden sind die Werte von $h_1(k)$ für die Schlüssel angegeben. Geben Sie für jeden Schlüssel an, wie viele Felder der Tabelle betrachtet wurden.

k	N	I	C	E	A	L	G	O
$h_1(k)$	3	5	3	7	5	1	7	3

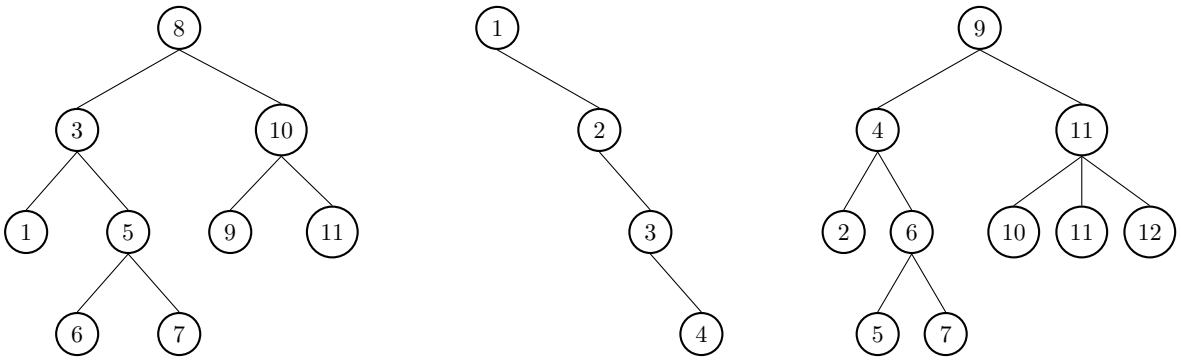
Aufgabe 12: Zweimal sortiertes Feld

Gegeben ist ein Feld von Zahlen $[a_1, a_2, ..., a_n]$. Dabei gilt, dass die Elemente mit ungeraden Indexen aufsteigend sortiert, es gilt also $a_1 \leq a_3 \leq a_5 \leq ...$ und die mit geraden Indexen absteigend sortiert sind. Also gilt $a_2 \geq a_4 \geq a_6 \geq ....$ Sie können davon ausgehen, dass n gerade ist. Ihre Aufgabe ist den Index einer gegeben Zahl im Feld zu finden. Dies soll in $\mathcal{O}(\log n)$ Zeit geschehen. Geben Sie einen Algorithmus in Pseudocode an.

Aufgabe 13: Binäre (Such-)Bäume – Eigenschaften

(a) Sei ein gefüllter Binärbaum ein binärer Suchbaum, dessen Knoten entweder 0 oder 2 Kinder besitzen. Angenommen ein gefüllter Binärbaum besitzt n Blätter. Wie viele Knoten hat der Baum insgesamt?

(b) Finden Sie die Fehler in den verschiedenen binären Suchbäumen.



(c) Ein Kommilitone von Ihnen hat einen Algorithmus (Algo. 14) geschrieben, der testen soll, ob ein gegebener binärer Suchbaum T mit Wurzel r ein korrekter binärer Suchbaum ist. Geben Sie ein Gegenbeispiel an, sodass der Algorithmus Ihres Kommilitonen ein falsches Ergebnis liefert und begründen Sie kurz, wieso der Algorithmus nicht funktioniert.

Algorithmus 14: `isBinarySearchTree`(**Node** $v = r$)

```
1 if v == null then
2     return true
3 if v.left ≠ null and v.left.key ≥ v.key then
4     return false
5 if v.right ≠ null and v.right.key ≤ v.key then
6     return false
7 return isBinarySearchTree(v.left) and isBinarySearchTree(v.right)
```

(d) Zeigen Sie, dass sich zwei unterschiedliche binäre Suchbäume mit gleichen Schlüsseln durch Rotationen ineinander verwandeln lassen, egal wie sie davor aussahen. Wie viele Rotationen benötigen sie maximal? *Hinweis: Zeigen Sie zunächst, dass sich jeder Binärbaum in eine Kette verwandeln lässt.*

(e) Beweisen oder widerlegen Sie die folgende Aussage über Rot-Schwarz-Bäume: "Jeder Geschwisterknoten eines Blattknotens ist entweder selbst ein Blatt oder rot"

Aufgabe 14: Rot-Schwarz-Bäume verstehen

(a) Zeichnen Sie den perfekt balancierten binären Suchbaum mit den Schlüsseln $\{1, \ldots, 15\}$. Färben Sie diesen Baum auf drei verschiedene Arten, sodass jeweils ein gültiger Rot-Schwarz-Baum entsteht.

(b) Angenommen wir haben einen Baum, der bis auf (E2) („Die Wurzel ist schwarz.") alle Rot-Schwarz-Eigenschaften erfüllt. Dieser Baum darf eine rote Wurzel haben. Ist der Baum ein gültiger Rot-Schwarz-Baum, wenn wir seine Wurzel ggf. schwarz färben?

(c) Es sei ein Binärbaum mit n Knoten gegeben. Wie viele verschiedene Rotationen können auf diesem Baum ausgeführt werden?

(d) Sei T ein gültiger Rot-Schwarz-Baum. Beweisen Sie, dass der längste Wurzel-Blatt-Pfad in T höchstens doppelt so lang ist, wie der kürzeste Wurzel-Blatt-Pfad in T.

Aufgabe 15: InsertionSort ∪ MergeSort

Obwohl die asymptotische worst-case Laufzeit von MergeSort $\Theta(n \log n)$ ist und die asymptotische worst-case Laufzeit von InsertionSort $\Theta(n^2)$ ist, läuft InsertionSort wegen der versteckten Konstanten in der O-Notation für kleine Eingaben oft schneller. Betrachte folgende Modifikation von MergeSort: Wir benutzen MergeSort bis wir n/k Teilfelder der Größe k haben. Diese sortieren wir mit InsertionSort und mergen sie anschließend mit dem bekannten Merge-Mechanismus von MergeSort.

(a) Zeigen Sie, dass InsertionSort die n/k Teilfelder insgesamt in $\Theta(nk)$ worst-case sortieren kann!

(b) Zeigen Sie, dass die oben beschriebene Variante von MergeSort in $\Theta(nk + n\log(n/k))$ worst-case läuft!

(c) Was ist der größte Wert von k als Funktion von n für die der modifizierte MergeSort Algorithmus die gleiche asymptotische worst-case Laufzeit hat wie die ursprüngliche Variante? (mit O-Notation)

(d) Wie sollte k in der Praxis gewählt werden?

Aufgabe 16: Telephonbuchsuche

Sie haben ein Telephonbuch mit Einträgen gegeben. Jeder Eintrag enthält den Namen und die dazugehörige Telephonnummer. Die Einträge sind nach Namen in alphabetischer Reihenfolge sortiert. Die Namen sind dabei paarweise verschieden und haben eine maximale Länge von m. Schreiben Sie einen Algorithmus in Pseudocode der für einen bestimmten Namen die zugehörige Telephonnummer in $\mathcal{O}(m \cdot \log n)$ Zeit findet.

Aufgabe 17: 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.

(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 18: Liste mit Varianz

Gegeben sei eine doppelt verkettete Liste L, die ganze Zahlen speichert.

(a) Augmentieren Sie die Liste L so, dass sie eine Methode `Mean` bereitstellt, die in $\mathcal{O}(1)$ den Durchschnitt aller Elemente in L zurückgibt.

(b) Augmentieren Sie die Liste L so, dass sie die Varianz σ^2 der Elemente in L in konstanter Zeit abgefragt werden kann. Dabei ist die Varianz von $x_1, \ldots, x_n$ folgendermaßen definiert:

$$\sigma^2 = \sum_{i=1}^{n} \frac{(x_i - \overline{x})^2}{n}, \text{ wobei } \overline{x} = \sum_{i=1}^{n} \frac{x_i}{n}$$

Die Laufzeit von `Insert`, `Delete` und `Search` sollen sich dabei nicht verändern.

Aufgabe 19: Traversierung von Binärbäumen

Es gibt drei gängige Arten binäre Baumstrukturen zudurchlaufen (traversieren). Diese heißen PreOrder, InOrder und PostOrder. Folgender Pseudocode zeigt eine Anwendung dieser drei Traversierungen:

Algorithmus 19:

```
1 PreOrder(Node x)
2     if x ≠ null then
3         print(x.key)
4         PreOrder(x.left)
5         PreOrder(x.right)
```

Algorithmus 20:

```
1 InOrder(Node x)
2     if x ≠ null then
3         InOrder(x.left)
4         print(x.key)
5         InOrder(x.right)
```

Algorithmus 21:

```
1 PostOrder(Node x)
2     if x ≠ null then
3         PostOrder(x.left)
4         PostOrder(x.right)
5         print(x.key)
```

Algorithmen 19-21 durchlaufen einen Binärbaum in unterschiedlichen Reihenfolgen und geben den *key* des Knoten x aus. Benutzen Sie den Binärbaum in Abbildung 1 für die folgenden Aufgaben.

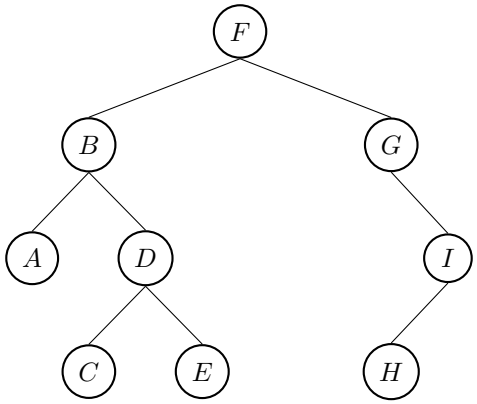


Abbildung 1: Binärbaum mit Buchstaben als *key* für Aufgabe 19.

(a) Wenden Sie jeweils die Algorithmen 19-21 auf den Baum in Abbildung 1 an und geben Sie den Output an!

(b) Welche rekursiven Algorithmen aus der Vorlesung kennen Sie, die die selbe Struktur haben wie die PreOrder und PostOrder Traversierungen?

(c) Neben diesen Traversierungen gibt es noch die LevelOrder Traversierung. Folgender Output wird von dieser Traversierung generiert: F, B, G, A, D, I, C, E, H. Geben Sie den Pseudocode eines Algorithmus an, der LevelOrder implementiert. Welcher Graph-Algorithmus aus der Vorlesung könnte den gleichen Output geben, wenn der Binärbaum als Graph repräsentiert wird?

Aufgabe 20: Finden von relevanten Intervallen

Angenommen Sie haben einen Sensor, der n Werte über eine Zeit in unregelmäßigen Abständen gemessen hat und diese Werte chronologisch in ein Feld A geschrieben hat. Ein Eintrag eines Feldes entspricht einem Tupel (t_i, x_i), wobei t_i die Zeit ist, an dem der Messwert x_i vom Sensor gemessen wurde. Sie wollen nun einen bestimmten Zeitraum $[t_s, t_e]$ an Messwerten abfragen. Geben Sie einen Algorithmus an, der in $o(n)$ ein Tupel von Indize (i, j) zurückgibt, sodass alle Messungen, die im Zeitraum $[t_s, t_e]$ getätigt wurden im Teilfeld $A[i..j]$ stehen.
Hinweis: Weder t_s noch t_e müssen als Werte in A existieren.