Aufgabensammlung ADS-Repetitorium WS 25/26

Elementare Datenstrukturen – Augmentierung – Amortisierte Analyse

Aufgabe 1: 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 1:

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

Algorithmus 2:

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

Algorithmus 3:

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

Algorithmen 1-3 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.

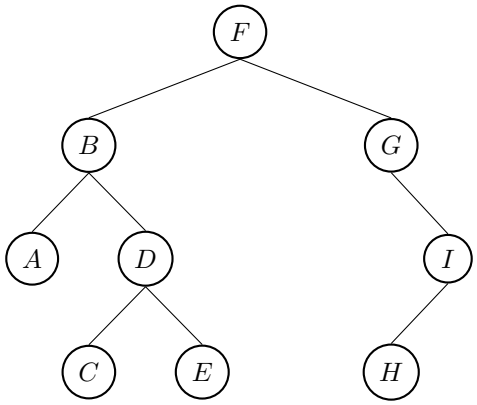


Abbildung 1: Binärbaum mit Buchstaben als *key* für Aufgabe 1.

(a) Wenden Sie jeweils die Algorithmen 1-3 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 2: Rot-Schwarz-Bäume

Zeichnen Sie die folgenden Beispiele. Im Folgenden werden die *nil*-Blätter nicht mitgezählt. Ein Baum, der nur aus einer Wurzel besteht, hat demnach einen Knoten und die Höhe eins.

(a) Gesucht ist ein gültiger Rot-Schwarz-Baum der Höhe drei, bei dem die Anzahl der Knoten minimal ist.

(b) Gesucht ist ein binärer Suchbaum der Höhe drei mit maximaler Knotenzahl, für den es keine gültige Rot-Schwarz-Färbung gibt.

(c) Gesucht ist eine Permutation von sieben paarweise verschiedenen Zahlen, sodass diese beim Einfügen in einen Binärbaum alle Ebenen komplett ausfüllen (der Binärbaum ist also balanciert).

Aufgabe 3: 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 4: 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 5: Min-Heaps

Gegen sei folgender Min-Heap in Feld-Darstellung:

$$[-5, 6, 3, 11, 18, 10, 8, 13, 17, 19]$$

(a) Wandeln Sie den Min-Heap in Baumdarstellung um und begründen Sie, dass es sich um einen Min-Heap handelt.

(b) Fügen Sie in den Min-Heap die Zahl 15 ein. Geben Sie das Ergebnis als Feld an.

(c) Führen Sie auf dem originalem (nicht auf dem aus Teilaufgabe b) enstandenem) Min-Heap die Methode ExtractMin aus. Geben Sie das Ergebnis als Feld an.

(d) Geben Sie an, ob folgende Aussage korrekt ist. Begründen Sie ihre Antwort:
In einem Min-Heap hat der Knoten mit dem größten Element immer maximale Tiefe.

(e) Geben Sie an, ob folgende Aussage korrekt ist. Begründen Sie ihre Antwort:
Das drittkleinste Element in einem Min-Heap ist nicht notwendigerweise ein Kind der Wurzel.

Aufgabe 6: Augmentierung von Rot-Schwarz-Bäumen

Ein Rot-Schwarz-Baum zur Verwaltung einer dynamischen Menge verschiedener ganzer Zahlen soll so augmentiert werden, dass man zu jeder Zeit bestimmen kann, für welche zwei Zahlen i, j der Menge mit $i < j$ die Differenz $j - i$ am kleinsten ist. Geben Sie die Methode minGap(RedBlackTree T) in Pseudocode an, die das gesuchte Zahlenpaar in konstanter Zeit liefert. Welche Extrainformationen speichern Sie im Baum und wie lassen sich diese beim Einfügen, Löschen und Suchen aufrechterhalten, ohne die Laufzeiten der entsprechenden Methoden zu verschlechtern? Lässt sich das Problem auch mit konstantem Zusatzspeicher lösen, wenn man auf die Delete-Methode verzichtet?

Aufgabe 7: Monotoner Stapel

Wir wollen in einem Stapel eine Menge von aufsteigenden Zahlen verwalten. Dabei wurde die `Push` Operation folgendermaßen mittels Algorithmus 12 modifiziert:

Algorithmus 12:

```
1 PushMonotone(Stack S, int x)
2     y = S.Pop()
3     while y > x and S ≠ ∅ do
4         y = S.Pop()
5     S.Push(x)
```

(a) Was ist die worst-case Laufzeit von `PushMonotone`?

(b) Zeigen Sie, dass `PushMonotone` bei einer Sequenz von n Operationen amortisiert konstant ist.

Aufgabe 8: 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.

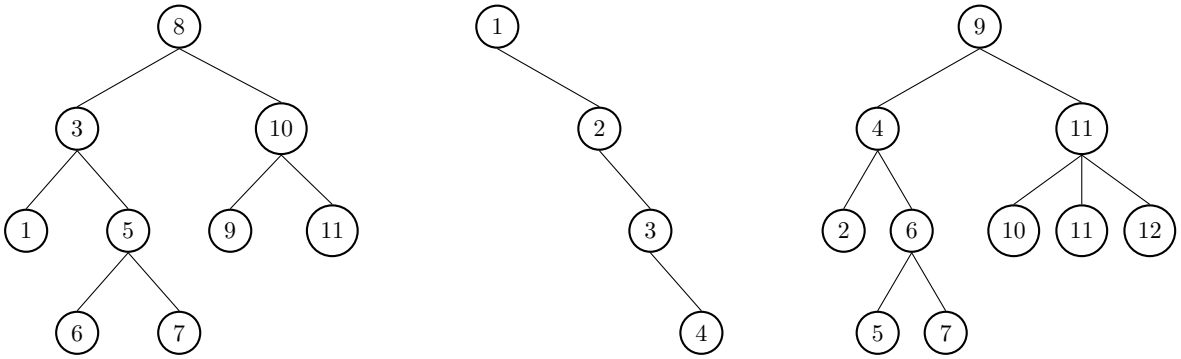
Aufgabe 9: Telephonbuchsuche

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 10: 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. 15) 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 15: `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 11: 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 12: 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 13: 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 14: 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 15: Bank-Schließfächer und Heaps

Bei einer Bank können Sie Schließfächer mieten. In jedes Schließfach passt genau ein Gegenstand. Der Service ist aber nicht kostenlos. Für das Einlagern des n. Gegenstands stellt die Bank $\log n$ Euro in Rechnung. Ebenfalls werden bei der Rückgabe des n. Gegenstands $\log n$ Euro fällig.

(a) Geben Sie die Gesamtkosten für das Ein- und Auslagern den n. Gegenstands in Θ-Notation an.

(b) Wie muss die Bank ihre Bezahlpolitik ändern, sodass das Abholen der Gegenstände kostenlos ist, die Bank aber trotzdem denselben Gewinn macht?

(c) Geben Sie die Kosten für das Ein- und Auslagern mit der neuen Bezahlpolitik in Θ-Notation an.

(d) Übertragen Sie Ihre Überlegungen aus den vorherigen Teilaufgaben auf einen MaxHeap. Zeigen Sie mit der Buchhaltermethode, dass die Insert-Methode amortisiert eine Laufzeit von $\mathcal{O}(\log n)$ und ExtractMax eine konstante Laufzeit hat.

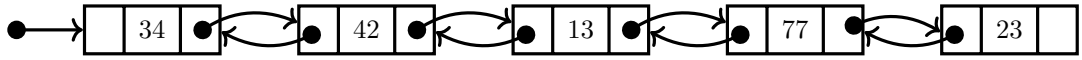
(e) Lösen Sie die Aufgabe nun mit der Potentialmethode.

Aufgabe 16: Doppelt-Verkettete Listen

Gegeben sei folgender Algorithmus

Algorithmus 25: 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 17: 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 18: Graham's Scan

Gegeben sei eine Menge $S \subseteq \mathbb{R}^2$ von Punkten auf der Ebene. Wir wollen ein Polygon finden, das alle Punkte von S enthält und nur Punkte aus S als Eckpunkte enthält. Abbildung 18 zeigt ein Beispiel für das gesuchte Polygon.

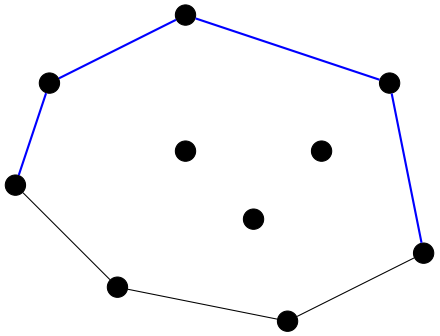


Abbildung 18: Beispiel für S und das dazugehörige Polygon in Aufgabe 18.

Algorithmus 26: Graham's Scan

```
1 Sortiere Punkte in S nach x-Koordinate,
    falls gefordert
2 L ← ⟨p_1, p_2⟩
3 for i = 3 to n do
4 |   L.Insert(p_i)
5 |   while |L| > 2 and die letzten 3 Punkte
  |     bilden einen links Knick do
6 |   |   entferne den vorletzten Punkt aus L
7 return L
```

Algorithmus 26 kann als Unterroutine benutzt werden, um ein solches Polygon zu finden. Dabei berechnet dieser Algorithmus die obere Hülle des Polygons. Das Beispiel in Abbildung 18 enthält die obere Hülle in blau eingezeichnet.

(a) Geben Sie eine worst-case Laufzeit für die Schleife in den Zeilen 5-6 an. Geben Sie weiterhin ein Beispiel an, für die dieser worst-case eintritt.

(b) Gehen Sie nun davon aus, dass S bereits nach x-Koordinate sortiert ist. Zeigen sie mittels amortisierter Analyse, dass die Laufzeit von Algorithmus 26 $\mathcal{O}(n)$ ist.

Aufgabe 19: Dynamic Table

In der Vorlesung sind wir häufig davon ausgegangen, dass unsere elementaren Datenstrukturen, wie eine Schlange oder ein Stapel, statischen Speicher haben. Das heißt, beim Anlegen der Datenstruktur geben wir bereits vor, wie viele Elemente die Struktur insgesamt speichern kann. Diese Größe ist nicht immer zu Beginn bekannt. Im Folgenden untersuchen wir, wie wir den Speicher einer Datenstruktur dynamisch vergrößern bzw. verkleinern können, sodass wir immer genug Platz haben aber gleichzeitig nicht unnötig viel Speicher verschwenden. Eine amortisierte Analyse wird zeigen, dass Insertion und Deletion dennoch $\mathcal{O}(1)$ Zeit benötigt.

Sei zunächst T eine Tabelle, die lediglich die Operation Insert unterstützt. Das Attribut T.table soll einen Pointer auf einen Speicherblock enthalten (ein Array). T.num ist die Anzahl der Elemente, die derzeitig in T enthalten sind und T.size ist die Größe des Speicherblocks. Zu Beginn gilt $T.\text{num} = T.\text{size} = 0$.

(a) Eine herkömmliche Heuristik beim Expandieren einer Tabelle ist, den vorherigen Speicher zu verdoppeln wenn die Tabelle voll ist. Mit anderen Worten: Wenn nur Insert Operationen unterstützt werden, sind immer mindestens T.size/2 Elemente in T enthalten. Schreiben Sie einen Pseudocode der die Operation Insert implementiert.

(b) Was ist die *worst-case* Laufzeit einer einzigen Insert Operation von Teilaufgabe (a)?

(c) Betrachten Sie nun eine Folge von n Insert Operationen. Zeigen Sie mit der Aggregationsmethode, dass die Insert Operation *amortisiert* $\mathcal{O}(1)$ Zeit benötigt.
Hinweis: Eine Tabelle verdoppelt sich bei jeder Expansion. Wie häufig kann eine Expansion bei n Insert Operationen auftreten?

(d) Zeigen Sie mit Hilfe der Buchhaltermethode, dass die Insert Operation *amortisiert* $\mathcal{O}(1)$ Zeit beansprucht.

(e) Zeigen Sie mit Hilfe der Potentialmethode, dass die Insert Operation *amortisiert* $\mathcal{O}(1)$ benötigt.
Hinweis: Entwicklen Sie eine Potentialmethode, die nach einer Expansion 0 ist und sich bis zur nächsten Expansion (pro Insert Operation) auflädt. Um wie viel muss sich die Potentialfunktion pro Operation aufladen?

Bisher haben wir uns lediglich die Insert Operation angeschaut. Nun betrachten wir auch die Delete Operation, die ein Element von unserer Tabelle löscht. Um nicht unnötig Speicher zu besetzen, wollen wir außerdem unsere Tabelle verkleinern, wenn T.num einen bestimmten Schwellenwert unterschreitet. Diese Verkleinerung funktioniert analog zur Expansion. Falls der Schwellenwert unterschritten wird, wird ein kleinerer Speicher allokiert und die Elemente in diesen Speicher rüberkopiert, damit anschließend der vorherige Speicher freigegeben werden kann.

(f) Ein Kommilitone von Ihnen schlägt vor, dass der Schwellenwert für eine Verkleinerung bei T.num/2 sein sollte, dessen Unterschreitung in einer Halbierung der Tablle resultiert. Zeigen Sie anhand einer Beispielfolge, dass diese Strategie nicht zu amortisiert konstanten Laufzeiten für die Insert und Delete Operation führt.
Hinweis: Sie dürfen davon ausgehen, dass eine Verdopplung der Tabelle stattfindet, wenn die Tabelle voll ist, wie in den vorherigen Teilaufgaben der Fall war.

(g) Das Problem der Strategie des Kommilitonen war, dass nach einer Verkleinerung der Tabelle nicht genug Insertion Operationen stattfinden konnten, um für eine Expansion zu zahlen und auch umgekehrt nicht genug Delete Operationen stattfinden konnten, um für eine Verkleinerung zu zahlen. Zeigen Sie mit der Potentialmethode, dass folgende Strategie aufgeht: Wenn $T.\text{num} < T.\text{size}/4$, dann halbiere die Größe der Tabelle.