Humboldt-Universität zu Berlin
Institut für Informatik
Prof. Dr. Ulf Leser
E. Angriman, A. Ermshaus, F. Nelles, M. Sänger

Berlin, den 03.05.2021

Übungen zur Vorlesung
Algorithmen und Datenstrukturen

Übungsblatt 2

Abgabe: Montag, den 17.05.2021, bis 11:10 Uhr über Moodle. Die Übungsblätter sind in Gruppen von zwei (in Ausnahmefällen drei) Personen zu bearbeiten. Die Lösungen sind auf nach Aufgaben getrennten **PDFs** über Moodle abzugeben. Alle Teilaufgaben einer Aufgabe sind in einem PDF hochzuladen. Pro Gruppe genügt es, wenn eine Person die Lösung der Gruppe abgibt. Zur Bewertung wird der zuletzt hochgeladene Stand herangezogen. Vermerken Sie auf allen Abgaben Ihre **Namen**, Ihre **CMS-Benutzernamen** und Ihre **Abgabegruppe (z.B. AG123)** aus Moodle. Benennen Sie die hochgeladenen PDF-Dateien nach dem Schema *A<Aufgabe>-<Person1>-<Person2>.pdf*, bspw. A03-Musterfrau-Beispiel.pdf für Aufgabe 3 von Lisa Musterfrau und Mark Beispiel. Die Auflistung der Namen kann in beliebiger Reihenfolge erfolgen.
Beachten Sie die Informationen im Moodle-Kurs (https://hu.berlin/algodat21).

Konventionen:

- Für ein Array A ist $|A|$ die Länge von A, also die Anzahl der Elemente in A. Die Indizierung aller Arrays beginnt bei 1 (und endet also bei $|A|$). Bitte beginnen Sie die Indizierung der Arrays in Ihren Lösungen auch bei 1.
- Mit der Aufforderung "Analysieren Sie die Laufzeit" ist gemeint, dass Sie eine möglichst gute obere Schranke der (Worst Case) Zeitkomplexität angeben und diese begründen sollen.
- Die Menge der natürlichen Zahlen $\mathbb{N}$ enthält die Zahl 0.

Aufgabe 1 (Sortierte Listen) — 4 + 2 + 2 + 2 + 2 + 4 = 16 Punkte

In dieser Aufgabe sollen Sie eine einfach verkettete, sortierte Liste zum Speichern von Integer-Werten in Java implementieren. Die Liste enthält einen Zeiger (headNode) auf den ersten Knoten, wobei dieser Zeiger in der leeren Liste auf null zeigt. Jeder Knoten der Liste speichert einen Integer-Wert und einen Zeiger auf den nachfolgenden Knoten. Falls kein Nachfolger existiert, so zeigt dieser Zeiger auf null. Da es sich um eine sortierte Liste handelt, muss stets gelten, dass der Wert jedes Knotens der Liste kleiner oder gleich dem Wert aller nachfolgenden Knoten ist. Die Liste soll gleiche Werte auch mehrmals speichern können, deren Reihenfolge untereinander ist nicht vorgegeben.

Implementieren Sie die nachfolgenden Funktionen. Ergänzen Sie dazu den fehlenden Code in der Vorlage `SortedList.java`, welche Sie im Moodle-Kurs vorfinden. Sie können beliebige neue Variablen und Hilfsmethoden zur Klasse hinzufügen.

- insert(value): Erstellt einen neuen Knoten mit Wert value und fügt ihn so in die Liste ein, dass die Sortierung erhalten bleibt.
- removeFirst(): Entfernt den ersten Knoten (headNode) aus der sortierten Liste und liefert dessen Wert zurück. Der Wert entspricht also dem Minimum der Liste. Falls die Liste leer ist, soll stattdessen eine UndefinedValueException geworfen werden.

- getLength(): Gibt die Länge der Liste zurück oder 0, falls die Liste leer ist. Um die volle Punkzahl zu erhalten, soll die Laufzeit Ihrer Lösung in $\mathcal{O}(1)$ liegen.
- getKLargest(k): Gibt den k-größten Wert der Liste zurück (ohne ihn zu entfernen), falls er existiert. Ansonsten wird eine UndefinedValueException geworfen. Die Indexierung beginnt bei 1 und der k-größte Wert ist der Wert des $(n-k+1)$-ten Knotens der sortierten Liste. Beispielsweise ist der Wert 4 in der Liste $[0, 1, 1, 4, 4, 6]$ sowohl das zweit- als auch das drittgrößte Element. Um die volle Punkzahl zu erhalten, soll die Laufzeit Ihrer Lösung in $\mathcal{O}(n)$ liegen.
- getMedian(): Gibt den Median der Liste zurück (ohne ihn zu entfernen). Falls die Anzahl der Knoten in der Liste ungerade ist, so ist der Median definiert als der Wert des Knotens, der an mittlerer Stelle steht. Wenn die Liste hingegen eine gerade Anzahl von Knoten enthält, ist der Median der Wert des rechten der beiden mittleren Knoten. Für eine leere Liste soll die Methode eine UndefinedValueException werfen. Um die volle Punkzahl zu erhalten, soll die Laufzeit Ihrer Lösung in $\mathcal{O}(n)$ liegen.
- merge(otherSortedList): Fügt alle Knoten der übergebenen Liste otherSortedList sortiert in die Liste ein. Die Liste otherSortedList ist selber sortiert und ebenfalls als verkettete Liste, als genauso wie die Liste, in die eingefügt wird, implementiert. Um die volle Punkzahl zu erhalten, soll die Laufzeit Ihrer Lösung in $\mathcal{O}(m+n)$ liegen, wobei m und n die Längen der beiden Listen sind.

Stellen Sie sicher, dass alle Testfälle in der main-Methode der Datei SortedList.java erfolgreich durchlaufen. Achten Sie außerdem auf Randbedingungen und Spezialfälle, die von den Testfällen vielleicht nicht vollständig abgedeckt werden.

Hinweis: Ihr Java-Programm muss auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, sodass die einzelnen Schritte nachvollziehbar sind. Die Abgabe des von Ihnen modifizierten Source-Codes `SortedList.java` erfolgt über Moodle.

Aufgabe 2 (Stacks und Queues) **6 + 6 = 12 Punkte**

In dieser Aufgabe sollen Sie zwei Algorithmen in Pseudocode entwerfen, die mit Stacks und Queues arbeiten. Bei der Verwendung eines Stacks oder einer Queue stehen Ihnen nur die für Stacks und Queues in der Vorlesung kennengelernten Methoden (also `enqueue()`, `dequeue()`, `head()` und `isEmpty()` für Queues bzw. `push()`, `pop()`, `top()` und `isEmpty()` für Stacks) zur Verfügung. Es existiert insbesondere keine Funktion zur Bestimmung der Länge eines Stacks/einer Queue. Ein Palindrom ist eine Zeichenkette, die von vorne und von hinten gelesen identisch ist. Beispiele hierfür sind die Zeichenketten

- „anna“,
- „rentner“,
- „erikafeuertnuruntreuefakire“ oder
- „tsssvssst“

Anagramme sind Zeichenketten, die durch Umstellung der Buchstaben (Permutation) ineinander umgewandelt werden können. Beispiele hierfür sind die Zeichenketten

- „regal“, „lager“ und „aeglr“,
- „angstbude“, „bundestag“, „dantesbug“ und „abdegnstu“ oder
- „wolfgangamadeusmozart“ und „afamousgermanwaltzgod“.

1. Entwerfen Sie einen Algorithmus in Pseudocode, der als Eingabe eine Zeichenkette der Länge n über dem Alphabet $\Sigma = \{a, b, \ldots, z\}$ erhält und in Laufzeit $\mathcal{O}(n)$ testet, ob es sich bei dieser Zeichenkette um ein Palindrom handelt. Die Eingabe wird als eine Queue q von Buchstaben übergeben, d.h., Sie haben zunächst nur Zugriff auf den ersten Buchstaben der Zeichenkette. Abgesehen von der Eingabe q, die beliebig modifiziert werden kann, darf Ihr Algorithmus lediglich Variablen von elementaren Datentypen (kein Array) sowie entweder einen zusätzlichen Stack oder eine zusätzliche Queue verwenden. Notieren Sie Ihren Algorithmus in Pseudocode und begründen Sie, warum seine Laufzeit in $\mathcal{O}(n)$ liegt.

2. Entwerfen Sie einen Algorithmus in Pseudocode, der als Eingabe eine Zeichenkette der Länge n über dem Alphabet $\Sigma = \{a, b, \ldots, z\}$ erhält und in Laufzeit $\mathcal{O}(n)$ testet, ob diese Zeichenkette ein Anagramm eines Palindroms ist, d.h., ob sich aus dieser Zeichenkette durch Permutation der Buchstaben irgendein Palindrom erzeugen lässt. Dieses Palindrom muss kein sinnvolles Wort sein. Ihr Algorithmus muss das Palindrom auch nicht ausgeben, es reicht **true** oder **false** als Ausgabe. Auch hier wird die Eingabe als eine Queue q von Buchstaben übergeben. Abgesehen von der Eingabe q, die beliebig modifiziert werden kann, darf Ihr Algorithmus lediglich Variablen von elementaren Datentypen (kein Array) sowie entweder einen zusätzlichen Stack oder eine zusätzliche Queue verwenden. Notieren Sie Ihren Algorithmus in Pseudocode und begründen Sie, warum seine Laufzeit in $\mathcal{O}(n)$ liegt.

Aufgabe 3 (Algorithmenentwurf) **9 + 3 = 12 Punkte**

Für eine Sequenz von n Zahlen $z_1, \ldots, z_n$ ist der Median allgemein definiert als der Wert, der an mittlerer Stelle steht, wenn man die Werte der Größe nach sortiert. Für eine aufsteigend sortierte Sequenz $z_1, \ldots, z_n$ mit $z_1 \leq \cdots \leq z_n$ ist der Median das Element $z_{(n+1)/2}$, falls n ungerade und das Element $z_{n/2+1}$, falls n gerade. Zum Beispiel: der Median der Sequenz $1, 10, 11$ ist 10 und der Median der Sequenz $0, 1, 2, 10$ ist 2.

Gegeben seien zwei aufsteigend sortierte Integer-Arrays $A = [a_1, \ldots, a_n]$ und $B = [b_1, \ldots, b_n]$, mit jeweils n Elementen. Der Einfachheit halber wollen wir annehmen, dass alle $2n$ Elemente paarweise verschieden sind.

1. Entwerfen Sie einen möglichst effizienten Algorithmus der den Median der Elemente beider Arrays bestimmt. Unter dem Median der Elemente beider Arrays ist der Median der Menge von Zahlen $a_1, \ldots, a_n, b_1, \ldots, b_n$ zu verstehen.

   Hinweis: Für einen korrekten sublinearen Algorithmus gibt es 9 Punkte, ein linearer Algorithmus gibt 6 Punkte, eine superlineare Variante 3 Punkte.

2. Analysieren Sie die Laufzeit Ihres Algorithmus.

Aufgabe 4 (Stacks) **10 Punkte**

In dieser Aufgabe geht es um das Auswerten von arithmetischen Ausdrücken, welche ausschließlich aus Ziffern, öffnenden und schließenden Klammern und den Operatoren $+$, $-$, $*$, / bestehen. Hierbei steht der Operator „/“ für Ganzzahldivision. Wir definieren einen *wohlgeformten arithmetischen Ausdruck* wie folgt:

- Eine Ziffer $d \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ ist ein wohlgeformter arithmetischer Ausdruck.
- Sind X und Y wohlgeformte arithmetische Ausdrücke, dann sind auch die vier Ausdrücke $(X + Y)$, $(X - Y)$, $(X * Y)$ und (X/Y) wohlgeformte arithmetische Ausdrücke.

Beispiele für wohlgeformte arithmetische Ausdrücke entsprechend dieser Definition sind unter anderem: „((8+7)∗2)“, „(4−(7−1))“ oder „8“. Bei den Ausdrücken „)8+)1(())“, „(8+())“, „−1“, „(5 − 7)“, „108“ oder „(8)“ handelt es sich hingegen nicht um wohlgeformte arithmetische Ausdrücke.

Schreiben Sie ein Java-Programm, welches für eine übergebene Zeichenkette überprüft, ob es sich um einen wohlgeformten arithmetischen Ausdruck handelt. Falls es sich um einen wohlgeformten arithmetischen Ausdruck handelt, so soll Ihr Algorithmus das Ergebnis dieses Ausdrucks ausrechnen und als Integer-Wert zurückgeben. Andernfalls soll eine ExpressionNotWellFormedException geworfen werden. Für diese Aufgabe bietet sich die Verwendung eines Stacks (java.util.Stack) an. Zur Vereinfachung der Implementierung, brauchen Sie beim Ausrechnen des Ergebnisses Divisionen durch 0 sowie arithmetische Überläufe nicht gesondert behandeln.

Ergänzen Sie dazu den fehlenden Code in der Vorlage `Parser.java`, welche Sie im Moodle-Kurs vorfinden. Sie können beliebige neue Variablen und Hilfsmethoden zur Klasse hinzufügen, dürfen jedoch keine außer den von Java bereitgestellten Standard-Bibliotheken verwenden. Stellen Sie sicher, dass alle Testfälle in der main-Methode der Datei Parser.java erfolgreich durchlaufen. Achten Sie außerdem auf Randbedingungen und Spezialfälle, die von den Testfällen vielleicht nicht vollständig abgedeckt werden.

Hinweis zur Abgabe: Ihr Java-Programm muss auf dem Rechner *gruenau2* laufen. Kommentieren Sie Ihren Code, sodass die einzelnen Schritte nachvollziehbar sind. Die Abgabe des von Ihnen modifizierten Source-Codes `Parser.java` erfolgt über Moodle.