AD-Aufgabe01-Gruppe-SawatzkiGlake 28.09.2013

Daniel Glake, Fabian Sawatzki
Praktikumsgruppe_2

Übungsaufgabe 1.1:

Aufgabenstellung:	Die Aufgabe ist es ein Modul zu schreiben, welches die Definitionen für die Mittelwert Berechnung sowie die Definition der korrigierten Varianz beinhaltet. Diese Implementation gibt es in zwei Varianten. Die erste Variante arbeitet mit einer Datenstruktur, die alle Messwerte zwischenspeichert. Die andere Variante arbeitet ohne eine Datenstruktur, ohne das die Messwerte zwischengespeichert werden müssen. Es soll bei beiden Varianten möglich sein den Mittelwert sowie die Varianz abzufragen.
Variante 1 (mit zwischengespeicherten Werten)	Unsere erste Variante arbeitet mit einer Listendatenstruktur, die alle neuen Messwerte, die über die Funktion add() aufgenommen werden zwischenspeichert. Desweiteren wird zu jedem neuen Messwert eine neue Summe berechnet, die bereits alle vorherigen Messwerte beinhaltet. Über die Größenfunktion size() der Liste berechnen wir dann den aktuellsten Mittelwert. Die Varianz erhalten wird, indem wir über die Liste iterieren und jeden Wert, mit dem aktuellen Mittelwert, der einmal vor der Iteration berechnet wird verarbeitet wird. Diese Berechnung erfolgen n-mal, n für die Maximallänge der Liste.
Variante 2 (ohne zwischengespeicherte Werte)	Unsere zweite Variante arbeitet ohne eine Datenstruktur, sodass es nicht möglich ist, alle Messwerte einzeln zwischen zu speichern. Über eine Funktion add() werden neue Werte eingetragen. Diese Werte werden anschließend sofort verrechnet. Für die Mittelwert-Berechnung arbeiten wir, ähnlich wie in Variante 1, mittels einer Instanz variable. Jeder neue Messwert wird mit einer Summe verrechnet, der alle bisherigen Einzelwerte beinhaltet. Desweiteren wird, bei jedem add() Aufruf ein Counter um 1 erhöht. Dieser merkt sich die Gesamtanzahl von Messwerten Dazu kommt noch die Addition des quadrierten Einzelwertes zu einer anderen Summe, die für die Varianz benötigt wird. Den Mittelwert erhalten wird dann über die Verrechnung der Summe mit dem Counter. Die Definition für die korrigierte Varianz war für fortlaufend neue Messwerte nicht sonderlich geeignet, da alle bisherigen Werte zwischengespeichert werden müssten, um immer den aktuellsten Mittelwert zu haben. Um diese Problem zu lösen haben wir den Verschiebungssatz verwendet, der über die Summe quadrierte Einzelwerte mit der Verrechnung der Summe von Einzelwerten sowie der Gesamtanzahl von Messwerten.
Verwendete Datenstruktur	Für Variante 1 haben wir uns für eine normale Listen Struktur entschieden, die als LinkedList implementiert wurde, da diese das Einfügen von neuen Elementen etwas effizienter gestalten. Zur Summierung haben wir eine einfache Instanzvariable angelegt, die bei jedem Einfügen eines Messwertes diese Variable, um den jeweiligen Messwert erhöht. Das erspart uns das Durchlaufen der gesamten Liste, um jedes Element nochmal lesen zu müssen. Für Variante 2 haben wir nur einige Instanzvariablen angelegt. Darunter wieder die Summe, die wie oben beschrieben funktioniert, aber auch

	einen Counter, um sich die Anzahl der Elemente zu merken.
Zwingende Testfälle	Als notwendigen Test haben wir vor allem, den genutzten Verschiebungssatz empfunden, da wir uns bis dahin noch nicht sonderlich mit Statistik beschäftigt haben. Wir mussten sichergehen, dass er die richtigen Ergebnisse liefert. Die restlichen Funktionen haben wir ebenfalls getestet, allerdings war im Vorfeld schon klar ersichtlich, dass diese Implementationen gut funktionieren würden. Um den Verschiebungssatz zu testen, haben wir uns einiger Werte bedient, die online bereits vorgerechnet wurden. Das Ergebnis haben wird dann anschließend mit dem, aus der Online-Quelle verglichen.

Übungsaufgabe 1.2:

Aufgabenstellung:	Die Aufgabe war es eine eigene Listen Datenstruktur, mit einigen Funktion zu implementieren. Dazu sollte man über cons() neue Werte einer Liste hinzufügen, mittels head() das erste Element entfernen können, die Länge einer Liste über length() bestimmen, überprüfen ob die Liste leer ist und ein Element an einer bestimmten Stelle einfügen.
Teilaufgabe 1.	Zur Erstellung der Signatur wurde ein eigenes Interface erstellt, das alle geforderten Funktionen beinhaltet.
Teilaufgabe 2.	Die Listenstruktur haben wir über eine eigene Klasse Elem gebaut, die für jedes Einzelelement ein next Zeiger besitzt, welches auf das nächste Element zeigt, sowie ein Behälter obj enthält, indem das Datenobjekt gehalten wird. Unsere Struktur ist so aufgebaut, dass immer auf ein nächstes Element gezeigt wird. Damit die Liste auch ein Ende hat zeigt das letzte Element auf einen null- Wert. Auf diesen null- Wert stützen sich auch einige Funktionsimplementationen. Damit die Struktur nicht kaputt geht, wenn ein null-Wert in die Liste eingetragen wird, fangen wir diesen Eintrag bei cons und insert ab. Es wird diesem Falle in NullPointerException geworfen. In der Listenimplementation haben wir zur Steuerung der Liste zwei Zeiger kopf und cursor. Mittels des kopf Cursors zeigen wir immer auf das erste Element, diese wird an den cursor übergeben, wenn über die Liste iteriert wird.
Teilaufgabe 3:	
Teilaufgabe 4:	Die Implementation zeigt, dass unsere cons() Funktion in der Laufzeitklasse von O(1) liegt, da das Element an die vorderste Stelle der Liste angefügt wird, ohne auf irgendein Element Rücksicht zu nehmen.
Teilaufgabe 5:	Die Implementation von cons(), für das Einfügen von Elementen an das Ende liegt dagegen in der Laufzeitklasse von $O(n^2)$, da bei einem neuen Eintrag die Liste n-Mal durchlaufen werden muss. Bei n- Durchläufen sind das n*n Durchläufe.
Teilaufgabe 6:	Zum Eintrag von neuen Listenelementen an eine zufällige Position haben wir ermittelt, dass unser Algorithmus ebenfalls in der Laufzeitklasse $O(n^2)$ liegen muss, da im schlimmsten Fall jedes Element an das Ende der Liste angefügt werden müsste. Das macht bei n- Eintragungen wieder n*n Durchläufe. Zur Zeitmessung haben wir in unserem Test mehrere Ergebnisse ermittelt und zu einem Mittelwert zusammengefasst. Ein Ergebnis war beispielsweise folgendes: [609, 608, 624, 593, 593, 608, 608, 640, 624, 593, 577, 609, 592, 593, 593] 604.2666666666667
Verwendete	Zur Realisierung unsere eigenen Listenstruktur haben wir einen

Datenstruktur	zusätzlichen Datentyp eingeführt. Dies war der Element Typ, der mit einer Instanz zur Speicherung, des tatsächlichen Werte und einer Instanz zum Zeigen auf das nächste Element ausgestattet ist. Diese Element und die Zeiger dafür wurden dann anschließen für unsere eigene Listenstruktur verwendet, um die geforderten Methoden entwickeln zu können.
Zwingende Testfälle	Als wichtigen Testfall haben wir die Insert Methode empfunden, da im Voraus noch nicht klar wurde, ob das Umlenken der Zeiger, für die nachfolgenden Elemente korrekt ausgeführt wurde. Anschließende Verbesserungen haben dies dann auch bewiesen, dass solch ein Test notwendig war. Ebenso wichtig war uns das Zusammenspielen aller Methoden, daher haben letztendlich einen Test angefertigt, der mehrere Methoden nacheinander ausführte und anschließend das Ergebnis mit einer einfachen Liste, mit nur einem Wert vergleicht.
Zusätzliche Quellen	Als zusätzliche Code Bibliothek haben wir die, von Google bereitgestellte Guava Bibliothek genutzt, da diese verschieden Precondition Methoden bereitstellt, wodurch wir Eingangsparameter, auf verschiedene Bedingungen prüfen konnten.

Zusätzliche Beschreibungen wurden als Dokumentation in die Tests der Programmabschnitte eingetragen. Dort finden sich auch die verschiedenen Hypothesen zu den Experimenten. JavaDoc steht als PDF bereit.