Dokumentation

Zu Aufgabe 01 aus der Vorlesungsreihe
„Algorithmen und Datenstrukturen“

Dokumentation

Zu Aufgabe 01 aus der Vorlesungsreihe „Algorithmen und Datenstrukturen“

Aufgabe 1

Übungsaufgabe 1.1

„Schreiben Sie ein Modul, mit dem man zu einer Messreihe von Werten“

„Entwerfen Sie ein geeignetes Interface für dieses Modul.Wenn Sie noch weitere Methoden brauchen, dann dokumentieren Sie den Grund.“

Interface:

```
public interface Messung {

	/** Liste aller Messungen */
	private List<Double> mssg;

	/** Summe aller Messungen für den Mittelwert */
	private double akku_sum;

	/** Getter für die Liste der Messungen
	 * @return Liste der Messungen
	 */
	public List<Double> getMessungen() ;

	/** Constructor für die Messung */
	public Messung();

	/** Fügt einen Messwert hinzu
	 *
	 * @param n Hinzuzufügender Messwert */
	public void add(double n);

	/** Errechnet den Mittelwert aller Messungen
	 *
	 * @return Mittelwert aller Messungen */
	public double average();

	/** Errechnet die Varianz der Messungen
	 *
	 * @return Varianz der Messungen */
	public double varianz();
}
```

Jede Instanz der Klasse Messung stellt eine Messreihe dar, mit den jeweiligen Messwerten.

„Implementieren Sie das Interface. Suchen Sie nach einem Weg, die Summation nicht jedesmal neu auszuführen."

Um die Summation nicht jedes Mal neu ausführen zu müssen, speichert man die zuletzt berechnete Summe in einer Variable x zwischen. Wenn nun ein neuer Messwert hinzugefügt wird, dann wird dieser auch gleichzeitig zu der vorherigen Summe x addiert, welche wieder zwischengespeichert wird.

„Müssen Sie in Ihrer Lösung alle bisherigen Messwerte mitspeichern? Wenn Ja: Geht das auch anders? Wie?"

Ja, es gibt eine Implementation in der die Messwerte nicht zwischengespeichert werden müssen. Dazu wendet man für die Varianz den Verschiebesatz an.

Übungsaufgabe 1.2

„Listen. Wir wollen u.a. die folgenden Methoden unterstützen:"

Interface:

```
package A1;

/** Implementation einer einfach verketteten Liste
 * @author Steffen Giersch, Birger Kamp, Maria Lüdemann
 */
public interface Liste {

	/** Ein Element vorne anfügen
	 *
	 * @param x Zuzufügendes Element */
	public void cons(Object x);

	/** Das vorderste Element von der Liste entfernen und zurückgeben
	 *
	 * @return Erstes Element der Liste oder bei leerer Liste eine
	 *         NullPointerException */

	public Object head();

	/** Das vorderste Element der Liste ausgeben (ohne entfernen)
	 *
	 * @return Erstes Element der Liste */
	public Object top();

	/** Gibt die Länge der Liste aus
	 *
	 * @return Länge der Liste */
	public int length();
```

```
        /** Ist die Liste leer oder nicht
         *
         * @return True (leer)oder False (nicht leer) */
        public boolean isempty();

        /** Nach n Elementen x in die Liste einfügen
         *
         * @param x Einzufügendes Element
         * @param n Position in der Liste, an der das Element eingefügt werden soll
         *            - beginnend mit 0
         * @return Wurde das Element erfolgreich eingefügt (True) oder nicht (False) */
        public boolean insert(Object x, int n);

        /**
         * Gibt die Anzahl an Dereferenzierungen aus
         * @return Anzahl an Dereferenzierungen
         */
        public int getStepCounter();

        /**
         * Setzt den Dereferenzierungscounter zurück
         */
        public void resetStepCounter();
}
```

Die implementierte Liste besteht aus zwei Klassen, wobei eine Klasse die Liste an sich darstellt mit entsprechenden Funktionen. Außerdem eine Klasse für die einzelnen Listenelemente, die einmal einen Objektwert **value** speichern und das folgende Listenelement **nextElem**.

Wir haben außerdem noch eine Funktion **top(list)** implementiert, die den Objektwert des ersten Listenelementes zurückgibt, ohne es gleich zu löschen wie **head**.

3. - Last in, first out gilt.
- Wenn ich ein Element hinzufüge, wird die Liste um ein Element länger, wenn ich eines weg nehme, wird sie um eines kürzer.
- Es kann kein Element aus einer leeren Liste entfernt werden
- Es kann kein Element an eine Position der Liste eingefügt werden, die größer ist, als die Liste lang ist.

Experimente

„Ein ersten Experiment: Fügen Sie nacheinander die Werte i von 1 bis n in eine leere Liste ein – jeweils am Anfang der Liste. Bestimmen Sie den Zeitaufwand mit obigen Zählern. Ist das gemessene Ergebnis der zu erwartende Wert?“

6. Hypothese:
n = 100 -> Wir nehmen an, dass wir, weil wir nur am Anfang der Liste einfügen, für jedes Einfügen genau einen Schritt brauchen. Das Ergebnis sollte daher t = 100 sein.

Experiment:

```
Liste list = new ListeImpl();
for (int i = 0; i < 100; i++) {
        list.insert(i, 0);
}
System.out.println(list.getStepCounter());
```

Ausgabe: 100

Analyse:
Das Experiment verlief wie erwartet, und ergibt so eine konstante Laufzeit.

„Wie zuvor, nur dass Sie an Ende einfügen."

7. Hypothese:
n = 100 -> $(n^2 + n) / 2 = 5050$
Da wir an Position 1..100 einfügen lässt sich die Komplexität über die Gaussche Summenformel beschreiben.

Experiment:

```
Liste list = new ListeImpl();
for (int i = 0; i < 100; i++) {
        list.insert(i, i);
}
System.out.println(list.getStepCounter());
```

Ausgabe: 5050

Auswertung:
Das Experiment verlief wie erwartet. Die Komplexität für das Einfügen an eine bestimmte Position in einer Liste beträgt O(n).

„Noch ein Experiment: Fügen Sie nacheinander die Werte i von 1 bis n in eine leere Liste ein, wobei das Element i an eine zufällige Position zwischen 0 und i□1 eingefügt werden soll."

8. Hypothese:
Der Durchschnittswert muss zwischen 100 und 5050 liegen, da das Minimum für das Einfügen von i = 100 Elementen 100 ist und das Maximum 5050.

Experiment:

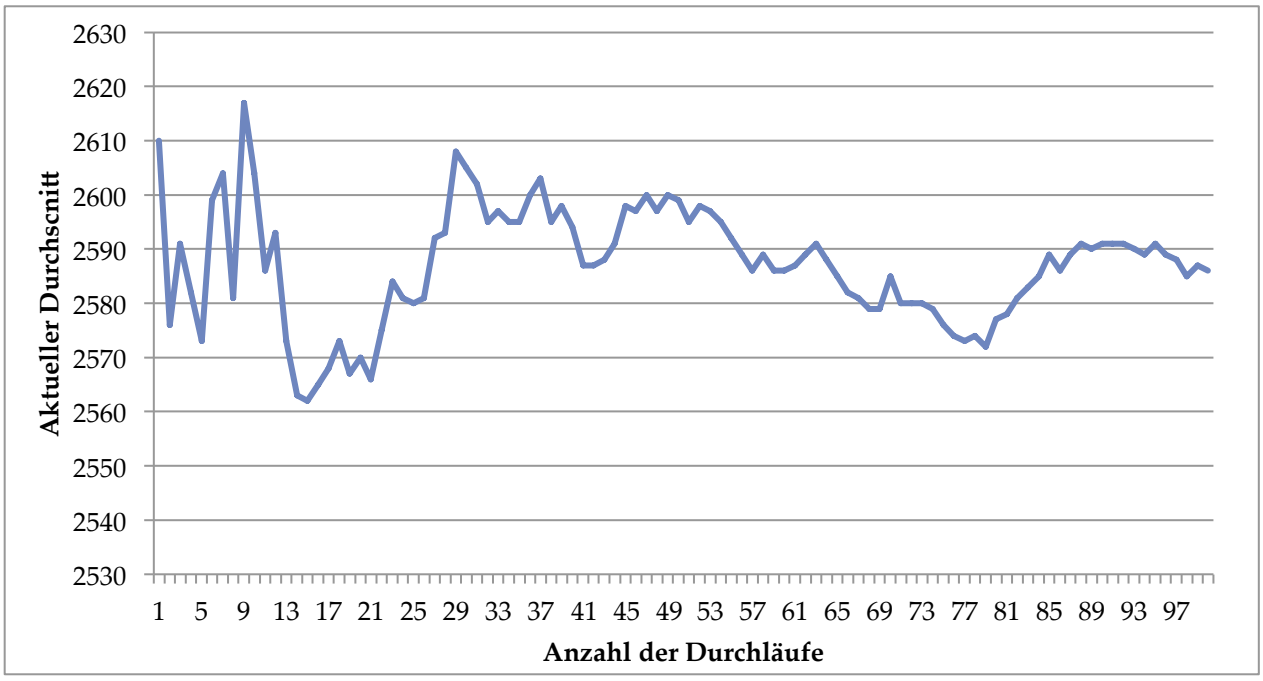
```
Messung m1 = new Messung();
for (int t = 0; t < 100; t++) {
        Liste list = new ListeImpl();
        for (int i = 1; i <= 100; i++) {
                list.insert(i, (int) (Math.rint(Math.random() * (i-1))));
        }
m1.add(list.getStepCounter());
        list.resetStepCounter();
}
System.out.println("Der Durchschnitt beträgt: " + m1.average());
System.out.println("Die Varianz beträgt:      " + m1.varianz());
```

Ausgabe:

Der Durchschnitt beträgt: 2575.01
Die Varianz beträgt: 169.92568161619153

Auswertung:

Das Ergebnis lieg wie erwartet zwischen 100 und 5050. Dieses Experiment stellt damit die mittlere Laufzeit für das Einfügen dar und die Varianz die mittlere Abweichung von Dieser.



Abschätzung des Zeitaufwands

Wir haben in etwa 10 Stunden für die gesamte Aufgabe benötigt. Die genaue Aufteilung können wir leider nicht mehr nicht mehr nachvollziehen.