|  |
| --- |
| 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

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

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.

Es gibt keine Implementation, in der man nicht alle einzelnen Messwerte speichern muss. Denn zur Berechnung der Varianz wird der Durchschnitt aller vorhandenen Messwerte benötigt, welcher dann von jedem einzelnen Messwert subtrahiert wird (s. Formel).

## Übungsaufgabe 1.2

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**.

**cons(x,list)** – Erstellt ein neues Listenelement, speichert das derzeitige Kopfelement in **nextElem** ab, und das übergebene Argument **x** als Objektwert **value** ab. Außerdem wird der LängenCounter um 1 inkrementiert.

**head(list)** – Gibt den Objektwert des derzeitigen ersten Listenelement **y** zurück, setzt **nextElem** als zukünftiges erstes Listenelement und anschließend wird **y** gelöscht. Es wird ein StepCounter um zwei, also die Anzahl der erfolgten Dereferenzierungen erhöht.

**length(list)** – Es wird der LängenCounter zurückgegeben, der in der Funktion **cons(x,list)** und **insert(x,n)** inkrementiert wird.

**isEmpty(list)** – Es wird überprüft ob die Referenz **nextElem** in der Liste der Nullpointer (null) ist.

**insert(x,n,list)** – Falls n=0 dann wird lediglich an cons(x,list) delegiert. Falls n>0 dann wird der Kette der Listenelemente solange verfolgt, bis man an der entsprechenden Stelle angelangt ist und das Objekt x an dieser Stelle eingefügt. Es wird der StepCounter um die Anzahl der erfolgten Dereferenzierungen erhöht. Falls n größer als die Länge der Liste oder kleiner als 0 ist, dann gibt die Methode ein False zurück und das Element wird nicht eingefügt.

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 inzufüge, 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

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, es ist keine erhöhte Strahlung festzustellen und ergibt so eine Konstante Laufzeit.

7. Hypothese:  
n = 100 -> (n² + 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).

7. 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.

## Bonus

Es befindet sich zusätzlich zur Aufgabenstellung noch eine ProcessingMain.java in dem Package. Wenn man diese als JavaApplet ausführt, erhält man Messreihen im 2 Sekunden Abstand grafisch dargestellt. Die Verarbeitung wurde über Processing realisiert. Die benötigten Dateien sind bereits in das Projekt eingearbeitet, sodass es ohne weiteres zutun lauffähig ist.