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.

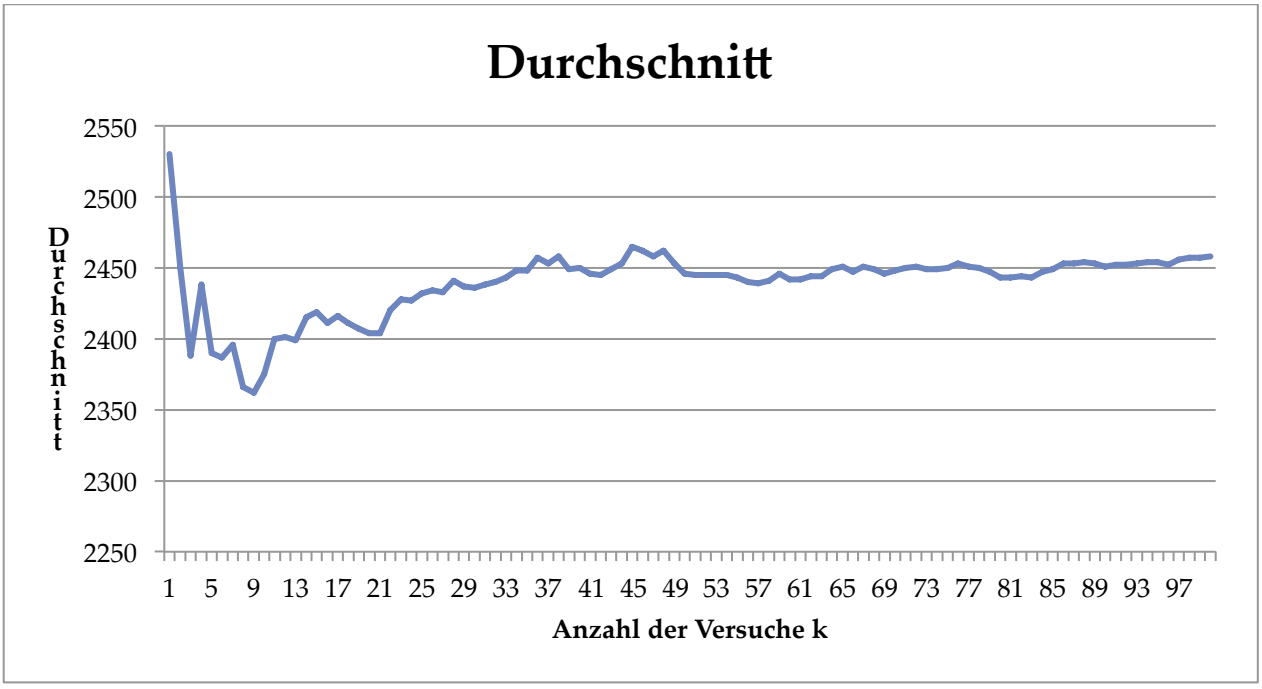
Wir haben außerdem noch eine Funktion **top(list)** implementiert, die den Objektwert des ersten Listenelementes zurückgibt, ohne es gleich zu löschen wie **head**.

Experimente

4. Erstes Experiment: Da nur am Anfang der Liste eingefügt wird, wird zu keinem Zeitpunkt dereferenziert, sondern es werden nur Referenzen umgespeichert und neue Objekte angelegt. Nein, es entspricht nicht dem zu erwartenden Wert, denn der wäre O(n), es ist aber konstant 0.

5. Zweites Experiment: Da immer am Ende der Liste eingefügt wird, müssen alle Elemente bis dahin dereferenziert werden. Die Anzahl der Dereferenzierungen beim Einfügen der Zahlen 0 bis 99 an selbigen Positionen in der Liste benötigt 4950 Dereferenzierungsschritte. Das entspricht der Komplexität der Gaußschen Summe, also $O(n^2)$.

6. Drittes Experiment: Für t=100 und n=100 beträgt der Durchschnitt 2473,577 Schritte und die Varianz 172,237. Wie sich nun darstellt ist das Ergebnis des ersten Experimentes die untere Schranke (Best-Case) für die Versuchsreihe und das Ergebnis des zweiten Experimentes ist die obere Schranke (Worst-Case) der Versuchsreihe. Das Ergebnis von diesem Experiment stellt also die mittlere Laufzeit dar und die Varianz die mittlere Abweichung 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.

Class ListElement

```
java.lang.Object
    |
    +--A1.ListElement
```

< Methods >

public class **ListElement**
extends java.lang.Object

Methods

getElem

`public java.lang.Object` **`getElem`**`()`

Gibt das gespeicherte Objekt zurueck

Returns:

Gespeichertes Objekt

---

getNext

`public ListElement` **`getNext`**`()`

Gibt die Referenz auf den Rest der Liste zurueck

Returns:

Referenz auf den Rest der Liste

insert

```
public int insert(java.lang.Object x,
                  int n,
                  int i)
```

Wenn n == 0 ist, dann wird das Objekt x hinter dieses ListElement gefuegt. Ansonsten wird die Aufgabe an das Folgende ListElement weitergereicht und n um 1 dekrementiert.

Parameters:

x - Zu Speicherndes Objekt
n - Indexzaehler fuer den Speicherort
i - Dereferenzierungs-Zaehler - Hilfsvariable fuer ListeImpl

Returns:

Anzahl der Dereferenzierungen

Interface Liste

< Methods >

---

public interface **Liste**

Methods

cons

```
public void cons(java.lang.Object x)
```

Ein Element vorne anfuegen

Parameters:

x - Zuzufuegendes Element

---

getStepCounter

```
public int getStepCounter()
```

Gibt die Anzahl an Dereferenzierungen aus

Returns:

Anzahl an Dereferenzierungen

---

head

```
public java.lang.Object head()
```

Das vorderste Element von der Liste entfernen und zurueckgeben

Returns:

Erstes Element der Liste oder bei leerer Liste eine NullPointerException

---

insert

```
public boolean insert(java.lang.Object x,
                      int n)
```

Nach n Elementen x in die Liste einfuegen

Parameters:

x - Einzufuegendes Element
n - Position in der Liste, an der das Element eingefuegt werden soll - beginnend mit 0

Returns:

Wurde das Element erfolgreich eingefuegt (True) oder nicht (False)

---

isempty

```
public boolean isempty()
```

Ist die Liste leer oder nicht

Returns:

True (leer)oder False (nicht leer)

---

length

```
public int length()
```

Gibt die Laenge der Liste aus

Returns:

Laenge der Liste

---

resetStepCounter

```
public void resetStepCounter()
```

Setzt den Dereferenzierungscounter zurueck

top

```
public java.lang.Object top()
```

Das vorderste Element der Liste ausgeben (ohne entfernen)

Returns:

Erstes Element der Liste

Class Messung

```
java.lang.Object
    |
    +--A1.Messung
```

< Fields > < Constructors > < Methods >

---

public class **Messung**
extends java.lang.Object

Klasse zum Speichern und Auswerten von Messreihen von Zahlen

Author:

Steffen Giersch, Birger Kamp, Maria Luedemann

Fields

akku_avg

```
public double akku_avg
```

Summe aller Messungen fuer den Mittelwert

---

mssg

```
public java.util.List mssg
```

Liste aller Messungen

Constructors

Messung

```
public  Messung()
```

Constructor fuer die Messung

Methods

add

`public void **add**(double n)`

Fuegt einen Messwert hinzu

Parameters:

n - Hinzuzufuegender Messwert

---

average

`public double **average**()`

Errechnet den Mittelwert aller Messungen

Returns:

Mittelwert aller Messungen

---

varianz

`public double **varianz**()`

Errechnet die Varianz der Messungen

Returns:

Varianz der Messungen

Class ProcessingMain

```
java.lang.Object
    |
    +--java.awt.Component
        |
        +--java.awt.Container
            |
            +--java.awt.Panel
                |
                +--java.applet.Applet
                    |
                    +--processing.core.PApplet
                        |
                        +--A1.ProcessingMain
```

All Implemented Interfaces:

java.awt.MenuContainer, java.awt.event.FocusListener, java.awt.event.KeyListener, java.awt.event.MouseListener, java.awt.event.MouseMotionListener, java.awt.event.MouseWheelListener, java.awt.image.ImageObserver, java.io.Serializable, java.lang.Runnable, javax.accessibility.Accessible, processing.core.PConstants

---

< Fields > < Constructors > < Methods >

---

public class **ProcessingMain**
extends processing.core.PApplet

Wird zum Ausfuehren des Processing-Graphen benutzt. Ist unabhaengig vom Rest der Aufgabe.

Author:

Steffen Giersch, Birger Kamp, Maria Luedemann

Fields

ANZAHLMESSUNGEN

`public static final int` **`ANZAHLMESSUNGEN`**

Anzahl der durchgefuehrten Messungen pro Bild

CIRCLESIZE

`public static final float` **`CIRCLESIZE`**

Groesse der angezeigten Kreise

FPS

`public static final float` **`FPS`**

Bilder pro Sekunde

LISTENLAENGE

```
public static final int LISTENLAENGE
```

Laenge der erstellten Listen

RIGHTSHIFT

```
public static final int RIGHTSHIFT
```

Rechtsshift fuer das Koordinatensystem in Pixeln

Constructors

ProcessingMain

```
public  ProcessingMain()
```

Methods

draw

```
public void draw()
```

Overrides:

draw in class processing.core.PApplet

---

main

```
public static void main(java.lang.String[] args)
```

---

setup

```
public void setup()
```

Overrides:

setup in class processing.core.PApplet

Class main

```
java.lang.Object
    |
    +--main
```

< Constructors > < Methods >

---

public class **main**
extends java.lang.Object

Constructors

main

```
public  main()
```

Methods

main

```
public static void main(java.lang.String[] args)
```

Parameters:

args -

INDEX