NitscheSahoo - AD Praktikum für 17.10.13
Gruppe 1 - Aufgabe 1 - Listen

Aufgabe 1.1:
Average und Variance

Im vorliegenden Fall sollten wir ein Modul schreiben was den Mittelwert und die Varianz berechnet. Hierbei kann man sich die eingegebenen Werte speicheren und den Durchschnitt/dieVarianz neu berechnen oder Akkumulieren.

```
public interface IAverageVariance {

	// adds a new value to the accumulator
	void addValue(double value);

	// bulk operation of add value
	void addValues(List<Double> values);

	// get current population size
	double getN();

	// get average
	double getAverage();

	// get veriance
	double getVariance();

}
```

Verschiebungssatz - Satz von Steiner(Wikipedia):

$$\sum_{i=1}^{n} (x_i - \bar{x})^2 = \left(\sum_{i=1}^{n} x_i^2\right) - n\bar{x}^2 = \left(\sum_{i=1}^{n} x_i^2\right) - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$$

Unter Verwendung des Satz von Steiner brauchen wir akkumulierend nur zwei Werte zu speichern und zur Rückgabe der Varianz/des Durchschnitts nur kleine Berechnungen zu machen. Wir akkumulieren die Summe aller eingehenden Werte. Außerdem akkumulieren wir auch die Summe aller Wertquadrate. Die Summe der Quadrate verwenden wir für $\left(\sum_{i=1}^{n} x_i^2\right)$. Für $-\frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2$ verwenden wir dann die akkumulierte Summe. Außerdem wird diese beim Durchschnitt verwendet.
Wir interpretieren die Varianz einer einelementigen Liste als „0.0“.

Aufgabe 1.2:
Das Interface:

Wir haben die Listenimplementation als einfach-verkettete Listen implementiert. Die Datenstruktur ist mutable.

```
public interface IList<T> {
	// add elem to the front
	void cons(T elem);

	// removes first element and return first
	T head();

	// Not in interface but a simple get(index) method
	// throws outofbound when n > arrlength-1 && n < 0
	T get(int index);

	// get first element
	T first();

	// how many elements has a list?
	int length();

	// is the list empty?
	boolean isempty();

	// insert element after index n so between n and n+2 if there is a current
	// n+2
	void insert(T elem, int n);

}
```

Aufgabe 5 – Anzahl der Dereferenzierungen

Def. Dereferenzierungen:
Alle Zugriffe innerhalb der Listenimplementation wo wir mit „this.first“ auf das erste Node oder mit „<Node>.getNext()“ auf das jeweils nächste Node zugreifen zählen wir als Dereferenzierungen.

Siehe Datei „ab1_test/ListAufgaben.java“

Erwartung:
Wir erwarten eine Komplexität von O(n) für die Gesamtanzahl der Zugriffe beim Hinzufügen am Anfang.
Ergebnisse:
Bei 15 Elementen haben wir 29 Zugriffe gezählt.
Interpretation:
Das erste Element braucht nur einen Zugriff. Alle folgenden Elemente brauchen im Hinzufügen zwei Zugriffe, da zusätzlich auch die Referenz auf das vorherige erste Element gesetzt werden muss. Daher haben wir für 15 Elemente 1 + (15 - 1)*2 = 29 Zugriffe.

Aufgabe 6 - Am Anfang hinzufügen

Siehe Datei „ab1_test/ListAufgaben.java“.

Erwartung:
Wir erwarten eine Komplexität von O(n) für die Gesamtanzahl der Zugriffe beim Hinzufügen am Anfang. (genauso wie bei Aufgabe 5)
Ergebnisse:
Siehe Excel Sheet „Aufgb1“-> „Aufgabe 6“
Interpretation:
Unsere Erwartung bestätigt sich. Wir haben bei einer Verzehnfachung der Anzahl der Elemente(von 15 auf 150) auch eine ungefähre Verzehnfachung der Dereferenzierungen(von 29 auf 299).

Aufgabe 7 - Am Ende inserten:

Siehe Datei „ab1_test/ListAufgaben.java“.

Erwartung:
Wir erwarten eine Komplexität von $O(n^2)$, weil das Hinzufügen am Ende nach der gausschen Summenformel eingeschätzt werden kann. Wir vermuten die gaussche Summenformel, weil wir zum Hinzufügen am Ende jedesmal einen Schritt zusätzlich machen müssen.
Ergebnisse:
Siehe Excel Sheet „Aufgb1“-> „Aufgabe 7“
Interpretation:
Unsere Erwartung bestätigt sich. Wir haben bei einer Verzehnfachung der Anzahl der Elemente(von 10 auf 100) eine quadratischen Wert für die Anzahl an Dereferenzierungen(von 76 auf 76^2=5776 is ungefähr gleich 5251). Eventuell, haben wir zusärtzliche Berechnungen die für diese Abweichung verantwortlich sind.

Aufgabe 8 - An einer zufälligen Stelle hinzufügen:

Siehe Datei „ab1_test/ListAufgaben.java“.

Wir haben gemessen, wie viele Zugriffe das Hinzufügen von n Elementen an zufälliger Position benötigt. Wir haben dieses Experiment dabei mit verschiedenen t-Werten wiederholt. Wir ahben außerdem das Experiment für festes t=50 mit n=52 und n=104 gestestet.

Erwartung:
Wir erwarten, dass der Durchschnitt der randomisierten Hinzufügens bei kleinen t stärker schwankt. Außerdem erwarten wir $O(n^2)$-Komplexität für das randomisierte Hinzufügen. Wir nehmen an, dass das random-Modul in Java die Werte gleichmäßig verteilt (Normalverteilung). Daher können wir erwarten, dass das Element im Durchschnitt in der Mitte hinzugefügt wird. Das ist immernoch eine quadratische Komplexität wie bei Aufagbe 7.

Ergebnisse:
Siehe Excel Sheet „Aufgb1“-> „Aufgabe 8“

Interpretation:
Beide unsere Erwartungen haben sich bestätigt. Am Diagramm kann man erkennen, dass der Durchschnitt bei höhren t stabiler wird. Das ist eine Konsequenz des Gesetzes der großen Zahl. Die quadratische Komplexität ist dadurch bestätigt, dass bei einer Verdoppelung von n (von 52 auf 104) sich die durchschnittliche Zahl an Zugriffen sich um das vierfache erhöht(von 797 auf 2924).

Aufwandsabschätzung:

Arbeitszeit: 9h gemeinsam – viel davon ist organisatorisches
Zeitverteilung: 10% AvgVarianz - 40% Listen - 50% Tests, Javadoc und Experimente mit Diskussion

Vorschau der Javadoc: