Schule Informationstechnik

der Bundeswehr

Sprachausbildung Java

Übungen

"Collections"

Stand: Januar 2015

1 LinkedList Benchmark

Implementieren Sie eine Klasse zum Test der Effizienz von `LinkedList` mit `Integer`-Elementen. Schreiben Sie für folgende Operationen jeweils eine Methode.

1. Anfügen von `n` Elementen an das Ende der Liste.
2. Einschieben von `n` Elementen am Anfang der Liste.
3. Sequentieller Zugriff auf `n` Elemente über den Index der Liste.
4. `n`-maliges Löschen des jeweils ersten Elementes über den Index.
5. Sequentieller Zugriff auf `n` Elemente über einen Iterator.
6. `n`-maliges Löschen des jeweils ersten Elementes über einen Iterator.
7. Schreiben Sie zwei Funktionen, die jeweils einen selbst geschriebenen Iterator nutzen und entsprechend durch die gesamte Liste gehen. Dabei sollen die Iteratoren folgendermaßen vorgehen:
    a. Ein Iterator der rückwärts durch die Liste läuft.
    b. Ein Iterator der nur jedes zweite Element liefert.

Erstellen Sie anschließend eine Testklasse, die die Laufzeit der einzelnen Funktionen mit N = 200.000 Einträgen misst.

Mit Hilfe folgendem Quellcode können Sie die Zeitmessung implementieren.

```
1  long start = System.currentTimeMillis();
2  /*
3  ...Die Operation, deren Dauer gemessen werden soll
4  */
5  System.out.println(System.currentTimeMillis()
   - start);
```

2 Stack - ListStack

Der `Stack` ist eine Datenstruktur mit folgender Funktionalität:

`push()` legt ein Element im Stack ab,

`pop()` entfernt das oberste Element und gibt seinen Wert zurück,

`peek()` rührt kein Element an, gibt aber den Wert des obersten Elementes zurück,

`empty()` rührt kein Element an und gibt als Antwort false, falls noch ein Objekt im Stack ist, ansonsten true.

Implementieren Sie ein Interface `Stack`, das diese Funktionalität zur Verfügung stellt. Entwickeln Sie einen `ListStack`, der das Interface implementiert. Nutzen Sie dabei die `deque`.

3 ArrayList Benchmark

Implementieren Sie eine generische Klasse zum Testen der Effizienz von `ArrayList` mit Elementen beliebigen Typs. Schreiben Sie für folgende Operationen jeweils eine Funktion.

1. Anfügen von `n` Elementen an das Ende der Liste.
2. Einschieben von `n` Elementen am Anfang der Liste.
3. Sequentieller Zugriff auf `n` Elemente über den Index der Liste.
4. `n`-maliges Löschen des jeweils ersten Elementes über den Index
5. Sequentieller Zugriff auf `n` Elemente über einen Iterator.
6. `n`-maliges Löschen des jeweils ersten Elementes über einen Iterator.
7. Schreiben Sie zwei Funktionen, die jeweils einen selbst geschriebenen Iterator nutzen und entsprechend durch die gesamte Liste gehen. Dabei sollen die Iteratoren folgendermaßen vorgehen:
    a. Ein Iterator der rückwärts durch die Liste läuft.
    b. Ein Iterator der nur jedes zweite Element liefert.

Erstellen Sie anschließend eine Testklasse, die die Laufzeit der einzelnen Funktionen mit N = 200.000 Einträgen misst.

4 Lottozahlengenerator

Sie haben soeben die Sets kennengelernt. Schreiben Sie nun ein kleines Programm, was Sie beim nächsten Lotto-Spiel unterstützt.

Wie Sie wissen hat man beim Lotto-Spiel die Möglichkeit, 6 Zahlen von 1 –49 anzukreuzen, natürlich keine doppelt. Zusätzlich kann man pro Lottoschein 1 bis 10 Tipps abgeben.

Schreiben Sie ein Programm, mit dessen Hilfe Sie für jeden Tipp einen Vorschlag bekommen, was Sie ankreuzen sollen.

Nutzen Sie das Verhalten von einem `HashSet` aus, der keinerlei Doubletten zulässt. Generieren Sie also zufällig für jeden Tipp sechs Zahlen zwischen 1 – 49 aus. Geben Sie am Schluss alle Tippvorschläge aus.

Hinweis: Die Funktion `Math.random()` liefert eine Zufallszahl zwischen 0 und 1.

5 Kontaktliste – Comparable

Implementieren Sie eine Klasse `Kontakt`. Ein Objekt dieser Klasse soll die Kontaktdaten (Name, Vorname, Telefonnummer, Straße, Hausnummer, Postleitzahl und Ort) einer Person beinhalten.

1. Diese Klasse soll das Interface `Comparable` implementieren. Objekte dieser Klasse werden anhand des Namens und des Vornamens verglichen.
2. Setzen Sie eine Adressliste unter Nutzung eines `TreeSet` um. Das heißt, die Kontakt-Objekte werden nach Namen und Vornamen sortiert in die TreeSet gelegt.
3. Stellen Sie Funktionen zum Einfügen und Löschen von einzelnen Kontakten zur Verfügung.
4. Schreiben Sie eine Funktion, die einen Namen und einen Vornamen bekommt und damit die zugehörigen Kontaktdaten zurückliefert.

6 Kontaktliste – Comparator

Diese Aufgabe setzt die Aufgabe 5 fort:

1. Setzen Sie eine unterschiedlich sortierte Adressliste um, das heißt: Nutzen Sie weiterhin ein `TreeSet` und entwickeln Sie für die verschiedenen Sortierkriterien (Adresse / Telefonnummer) jeweils einen `Comparator`.
2. Stellen Sie für diese Adresslisten Einfüge-, Abfrage- und Löschoperationen zur Verfügung.

7 Telefonbuch I

Entwickeln Sie ein Telefonbuch aus dem Kontaktdaten ausgelesen werden können. Anhand des Vor- und Nachnamens einer Person sollen die Kontaktdaten abrufbar sein, d.h. die Kontaktdaten sollen in einem Assoziativspeicher abgelegt werden, in dem die Kombination Vor- und Nachname als Schlüssel dient. Implementieren Sie folgende Methoden:

1. Schreiben Sie eine Funktion, mit welcher Kontaktdaten zum Telefonbuch hinzugefügt werden können.
2. Schreiben Sie eine Funktion, die anhand des Vor- und Nachnamens den entsprechenden Kontakteintrag liefert.
3. Schreiben Sie eine Funktion, mit welcher Kontaktdaten aus dem Buch gelöscht werden können.

8 Telefonbuch II

Diese Aufgabe erweitert Aufgabe 7. Bisher können wir nur anhand des Vor- und Nachnamens einer Person deren Kontaktdaten abrufen. Nun erweitern Sie das Telefonbuch soweit, dass auch anhand einer Telefonnummer die zugehörige Person geliefert wird. Dabei soll der Zugriff mit dieser Variante genauso schnell sein wie in Aufgabe 1. Das heißt, Sie benötigen einen weiteren Assoziativspeicher, in dem die Kontaktobjekte mit lediglich belegter Telefonnummer als Schlüssel dient.

Implementieren Sie folgende Methoden:

1. Schreiben Sie eine Funktion, mit welcher Kontaktdaten zum Telefonbuch hinzugefügt werden können. Beachten Sie, dass beide Speicher angepasst werden, damit die Einträge in beiden gleich sind.
2. Schreiben Sie eine Funktion, die anhand der Telefonnummer den entsprechenden Kontakteintrag liefert.
3. Schreiben Sie eine Funktion, mit der Kontaktdaten aus dem Buch gelöscht werden können. Beachten Sie, dass beide Speicher angepasst werden, damit die Einträge in beiden gleich sind.

9 BubbleSort

Schreiben Sie eine Klasse die eine `ArrayList` mit Einträgen beliebigen Typs enthalten kann. Der Typ ist lediglich dahingehend eingeschränkt, dass dessen Objekte vergleichbar sein müssen. (Tipp: ein bestimmtes Interface muss implementiert sein).

Schreiben Sie eine Methode, die die Liste mit Hilfe von BubbleSort sortiert und zurück liefert.

BubbleSort ist ein sehr einfacher vergleichsbasierter Sortieralgorithmus. Den Namen verdankt er der Weise wie die Elemente nach und nach wie Blasen im Wasser nach hinten (bzw. vorne) wandern. BubbleSort ist sehr simpel zu verstehen und wird meist als einführender Sortieralgorithmus vorgestellt, ist aber auch meist sehr ineffizient (besonders bei großen Datenmengen).

Funktionsweise

Der Algorithmus durchläuft die zu sortierende Folge und vertauscht benachbarte Paare, die in der falschen Reihenfolge vorliegen. So wird garantiert, dass die großen Elemente nach und nach ans Ende der Folge wandern. Dieser Durchlauf wird solange wiederholt bis keine Vertauschungen mehr nötig sind und die Folge damit sortiert ist. Um die Geschwindigkeit zu verbessern bedarf es nur den Durchlauf bis zu den schon nach hinten gewanderten Elementen, da diese bereits an die richtige sortierte Position gewandert sind. Dadurch wird stets ein Element weniger geprüft, ändert aber insgesamt nichts an der Komplexität des Algorithmus.

Beispiel

Eine Reihe von fünf Zahlen soll aufsteigend sortiert werden.

Die fett gedruckten Zahlen werden jeweils verglichen. Ist die linke größer als die rechte, so werden beide vertauscht; das Zahlenpaar ist dann blau markiert. Im ersten Durchlauf wandert somit die größte Zahl ganz nach rechts. Der zweite Durchlauf braucht somit die letzte und vorletzte Position nicht mehr zu vergleichen. --> Dritter Durchlauf: kein Vergleich letzte/vorletzte/vorvorletzte....

```
55 07 78 12 42   1. Durchlauf
07 55 78 12 42
07 55 78 12 42
07 55 12 78 42   Letzter Vergleich

07 55 12 42 78   2. Durchlauf
07 55 12 42 78
07 12 55 42 78   Letzter Vergleich
07 12 42 55 78

07 12 42 55 78   3. Durchlauf
07 12 42 55 78   Letzter Vergleich
07 12 42 55 78
07 12 42 55 78

07 12 42 55 78   Fertig sortiert.
```

Implementieren Sie das Bubble-Sort-Verfahren.

10 InsertionSort

Erweitern Sie die Klasse aus Aufgabe 9 um eine Funktion, die die Liste mit dem Insertionsort-Algorithmus sortiert.

Beim Insertionsort, wird die Originalliste sukzessive durchlaufen und die einzelnen Objekte in eine neue Liste sortiert eingefügt. Das Ergebnis ist die neue Liste. Verwenden Sie, wo es möglich ist, einen Iterator.

11 Benchmark – BubbleSort vs. InsertionSort

Messen Sie die Laufzeit der einzelnen Funktionen für eine Liste mit 50000 zufällig gewählten Einträgen zwischen 1 und 1.000.000.

Hinweis:

Die Funktion `Math.random()` liefert eine Zufallszahl zwischen 0 und 1.

12 QuickSort

Informieren Sie sich mittels Wikipedia über Quicksort und versuchen diesen als dritte Funktion zu implementieren.

Anschließend führen Sie den Benchmark erneut mit allen 3 Sortieralgorithmen durch.