Aufgabenblatt 2
Suchen und Sortieren

- Abgabetermin: **Freitag, 09.05.14 23:59 Uhr**
- Zur Prüfungszulassung muss ein Aufgabenblatt mit mind. 25% der Punkte bewertet werden und alle weiteren Aufgabenblätter mit mindestens 50% der Punkte.
- Die Aufgaben müssen in *Zweiergruppen* bearbeitet werden.
- Abgabe:
  - Geben Sie im Abgabesystem[1] eine ZIP-Datei ab, welche den Quelltext aller implementierten Klassen (Java 7), sowie entsprechende textuelle Lösungen enthält.
  - Achten Sie darauf, dass ihr ZIP-Archiv *keine* kompilierten Klassen (class-Dateien) oder Tests (`test` Ordner) enthält.
  - Textuelle Lösungen reichen Sie bitte in einer entsprechend benannten Textdatei (aufgabe_x.txt) auf oberster Ebene im ZIP-Archiv ein.
  - Der Quelltext soll in einer Ordnerstruktur analog zu assignment2_sources.zip[2] vorliegen: Quellcode im `src` Ordner, etc.
  - Achten Sie darauf, dass Ihr Quelltext gut kommentiert ist.
  - Geben Sie bei der Einreichung Ihrer Lösung den Namen des anderen Gruppenmitglieds unter "Authors" an (pro Gruppe nur eine Abgabe).

Vorbereitung

Laden Sie sich das Archiv assignment2_sources.zip[2] herunter, es enthält das Gerüst (Klassen- sowie Methodendeklarationen und UnitTests), auf dem die folgenden Aufgaben aufbauen.

Aufgabe 1: Suchen

6 P

Die folgenden Aufgaben beschäftigen sich mit dem Auffinden von Elementen in ganzzahligen Arrays.

a) Implementieren Sie die Funktion `findUnsorted(int x, int[] a)` der Klasse `assignment2.Search`, die die Indizes *jedes* Vorkommens von x im Array a aufsteigend sortiert zurück gibt. Zum Beispiel ist das erwartete Ergebnis für $a = [5, 2, 4, 1, 4]$ und $x = 4$: $\texttt{unsorted}(x, a) = [2, 4]$.

b) Implementieren Sie die Funktion `findSorted(int x, int[] a)` der Klasse `assignment2.Search`, die *jedes* Vorkommen von x im sortierten Array $a[i]$ zurück gibt. Für $a = [1, 2, 4, 4, 5]$ und $x = 4$ wird also folgendes Ergebnis erwartet: $\text{sorted}(x, a) = [2, 3]$ (Das ergebnis ist ebenfalls aufsteigend sortiert). Achten Sie auf eine *effiziente* Implementierung der Methode (sie sollte den Umstand ausnutzen, dass die Eingabe sortiert ist).

c) Beobachten Sie die Laufzeit beider Funktionen (z.B. für $|a| \geq 1.000.000$) und diskutieren Sie ihre Beobachtungen kurz. Speichern Sie die Lösung als Textdatei (aufgabe_1c.txt) und fügen Sie die Datei dem ZIP-Archiv hinzu.

[1] `https://www.dcl.hpi.uni-potsdam.de/submit/`
[2] `http://hpi-web.de/fileadmin/hpi/FG_Naumann/lehre/SS2014/PTII/assignment2_sources.zip`

Aufgabe 2: Sortieren

8 P

Die folgenden Aufgaben beschäftigen sich mit dem Sortieren von ganzzahligen Arrays.

a) Implementieren Sie die Funktion `bubblesort(int[] a)` der Klasse `assignment2.Sort`, die das Array x mit Hilfe des BubbleSort-Verfahrens sortiert. Zum Beispiel soll $a = [5, 2, 4, 1, 4]$ nach dem Aufruf der Methode folgenden Inhalt besitzen $a = [1, 2, 4, 4, 5]$

b) Implementieren Sie die Funktion `quicksort(int[] a)` der Klasse `assignment2.Sort` analog zur vorherigen Aufgabe, die das QuickSort-Verfahren anwendet.

c) Implementieren Sie die Funktion `countingsort(int[] a)` der Klasse `assignment2.Sort` analog zu den vorherigen Aufgaben. Verwenden Sie dieses Mal das CountingSort-Verfahren[3] (Sie können von $a[i] \in \mathbb{N}_0$ ausgehen).

d) Beobachten Sie die Laufzeit der drei Funktionen (z.B. für $|a| \geq 1.000.000$) und diskutieren Sie ihre Beobachtungen kurz. Speichern Sie die Lösung als Textdatei (aufgabe_2d.txt) und fügen Sie die Datei dem ZIP-Archiv hinzu.

[3] `http://de.wikipedia.org/wiki/Countingsort`

Aufgabe 3: Kartenspiel

6 P

Geben sei die Klasse `assignment2.Card` welche eine Spielkarte repräsentiert. Jede Karte besitzt eine Farbe (`Suit`) und einen Rang (`Rank`).

a) Implementieren Sie die Funktion `compare(Card card1, Card card2)` der Klasse `assignment2.DefaultCardComparator`. Die Funktion soll sich dabei konform zum Java-Interface `Comparator`[4] verhalten und

   i eine negative Zahl zurückgeben, wenn der Wert erste Karte kleiner ist als der der zweiten Karte

   ii Null (0) liefern, wenn beide Karten gleichwertig sind

   iii eine positive Zahl zurückgeben, wenn die erste Karte einen größeren Wert besitzt ist als die zweite

   Die Wertigkeit der Karten ist wie folgt definiert: Zunächst ist die Wertigkeit der Karten in Abhängigkeit zu ihrer Farbe definiert. Alle Karo-Karten (`Diamonds`) sind höherwertiger als Kreuz-Karten (`Clubs`), alle Herz-Karten (`Hearts`) sind höherwertiger als Karo-Karten, und alle Pik-Karten (`Spades`) besitzen einen höheren Wert als Herz-Karten. Weiterhin besitzen die Zählkarten Zwei bis Zehn (`Two`, `Three`, `Four`, `Five`, `Six`, `Seven`, `Eight`, `Nine`, `Ten`) den Wert ihrer Augen, die Bildkarten (`Jack`, `Queen` und `King`) zählen je zehn Punkte und das Ass (`Ace`) zählt elf Punkte.

b) Implementieren Sie die Funktionen `bubblesort(Card[] a, Comparator<Card> c)` und `quicksort(Card[] a, Comparator<Card> c)` der Klasse `assignment2.Card`, analog zur Implementierung aus den Aufgaben 2a und 2b. Verwenden Sie jedoch zum Vergleich zweier Werte im Array statt den Vergleichsoperatoren ($==, >, <, \ldots$) die Methode `compare` des übergebenen `Comparator` Objektes und sortieren Sie die Karten aufsteigend (von der Karte mit dem geringsten Wert zur Karte mit dem höchsten Wert).

c) Implementieren Sie `countingsort(Card[] a)` der Klasse `assignment2.Card`, analog zur Implementierung von CountingSort aus den Aufgaben 2c. Die Sortierreihenfolge soll dabei identisch zu Aufgabe 3b gewählt werden (siehe `assignment2.DefaultCardComparator`).

d) Diskutieren Sie kurz, warum das CountingSort-Verfahren nicht bei allen Sortieraufgaben angewandt werden kann. Speichern Sie die Lösung als Textdatei (aufgabe_3c.txt) und fügen Sie die Datei dem ZIP-Archiv hinzu.

[4] `http://docs.oracle.com/javase/7/docs/api/java/util/Comparator.html`

Zusatzaufgabe: Spürhunde

Für das Training von Spürhunden befüllt der Trainer n Boxen mit unterschiedlichen Gegenständen und stellt sie in einer Reihe auf. Die Art eines Gegenstandes ist durch eine positive Zahl gekennzeichnet, das heißt der Wert des Gegenstands in der i-ten Box ist $b[i-1] > 0$.

Zum Trainieren eines Hundes lässt der der Trainer den Hund an Gegenstand x schnuppern und schickt ihn von einem der beiden Enden der Reihe los. Der Hund überprüft nun jede Box einzeln (der Reihe nach) und sucht nach einer Box mit einem Gegenstand mit dem Wert x. Hat der Hund eine Box mit Gegenstand x gefunden, bringt er dem Trainer die Box unverzüglich. Findet der Hund den Gegenstand nicht, d.h. er kommt am anderen Ende an ohne x gefunden zu haben, kehrt er ohne Box zurück.

Nun möchte der Trainer zwei Hunde gleichzeitig trainieren. Dies tut er indem er beide Hunde gleichzeitig losschickt. Damit die Hunde sich nicht um eine Box streiten, müssen die gesuchten Gegenstände in den Boxen der beiden Hunde unterschiedlich sein, die Summe der beiden Werte muss jedoch einem gegebenen Wert k entsprechen. Dazu kann der Trainer die Hunde entweder von beiden Seiten der Reihe (links und rechts) oder beide Hunde von einer Seite (links oder rechts) gleichzeitig starten lassen. Die Hunde benötigen genau eine Sekunde um an einer Box zu schnüffeln, jedoch laufen sie so schnell, dass die Laufzeit vernachlässigt werden kann. Der Trainer will nun wissen, wieviele Sekunden er mindestens warten muss, um sich die entsprechenden Gegenstände von den Hunden bringen zu lassen.

Implementieren Sie zum Lösen dieser Aufgabe die Methode `secondsToWait(int k, int[] b)` der Klasse `assignment2.DogTraining`. Die Funktion soll die minimale Anzahl von Sekunden zurückgeben, die der Trainer warten muss, bzw. -1 wenn es keine valide Kombination gibt. Achten Sie darauf, dass ihr Verfahren effizient ist.

Beispiel

Der Aufruf `secondsToWait(5, new int[] {2,4,3,2,1})` soll das Ergebnis `2` liefern, da der Trainer mindestens 2 Sekunden warten muss, bis er zwei Boxen erhält, deren Inhalt die Summe 5 ergibt (er schickt z.B. Hund 1 von links los um nach einer 4 zu suchen während Hund 2 von rechst beginnend nach einer 1 sucht).

Wertebereiche

- $2 \leq n \leq 500000$ $(5 * 10^5)$
- $1 \leq k \leq 1000000$ (10^6)
- $\forall i < n : 1 \leq b[i] \leq 1000000$ (10^6)