HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Algorithmen und Datenstrukturen (IB) Praktikum
Aufgabenblatt 5 – Einfaches Sortieren Korrektur

Ausgabe: 18. April 2018
Abgabe: 24. April 2018, 18:00 Uhr

Allgemeines

In Moodle finden Sie ein Maven-Projekt und darin einen kleinen Rahmen zur Implementierung von vergleichsbasierten Sortieralgorithmen mit Zählung von Vergleichs- und Vertauschungsoperationen.

Der Rahmen besteht aus einer Klasse `Sortable`, die ein Array von (vergleichbaren) Objekten kapselt, die nur Vergleiche und Vertauschungen zulässt und diese Operationen mitzählt, und einer abstrakten Klasse `AbstractSorter`, von dem konkrete Sortieralgorithmen abgeleitet werden können, so dass beim Sortieren von Eingaben automatisch Statistiken ermittelt und ausgegeben werden.

Konkrete Sortieralgorithmen müssen eine Methode `void sort(Sortable<T> input)` implementieren. Die Klasse `Sortable` stellt dafür nur die Methoden `lessOrEqual(int index1, int index2` (und andere Vergleiche) und die Methode `swap(index1, index2)`, sowie eine Methode `length()` zur Verfügung.

Als Beispiel finden Sie den Algorithmus GnomeSort im Paket `...sorters` in einer Klasse `GnomeSorter` implementiert. An dieser Implementierung können Sie erkennen, wie Sie die Algorithmen aus der Vorlesung übertragen können.

In der Klasse `Main` sind bereits Aufrufe verschiedener (auch noch nicht implementierter) Algorithmen vorbereitet. Sie nutzen eine Hilfsklasse `TestDataFactory` zur Erstellung interessanter Eingaben (von Sortiert über zufällig bis unsortiert).

Testdaten werden mit den folgenden Strukturen erzeugt:

1. Sortierte Eingaben, z.B.
   $[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25]$
2. Sortierte Eingaben mit $n/20$ zufälligen Vertauschungen, z.B.
   $[1, 2, 3, 4, 5, 6, 17, 8, 9, 10, 11, 12, 13, 14, 15, 16, 7, 18, 19, 20, 21, 22, 23, 24, 25]$
3. Teilweise Sortierte Eingaben, d.h. die Eingabe beginnt mit 90% der Daten in sortierter Reihenfolge gefolgt von den restlichen 10% in zufälliger Reihenfolge, z.B.
   $[1, 2, 4, 5, 6, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 8, 7, 3]$
4. Zufällig/unsortierter Eingabe, z.B.
   $[17, 3, 8, 12, 13, 2, 11, 1, 16, 19, 5, 7, 21, 22, 14, 20, 9, 23, 25, 15, 18, 4, 24, 10, 6]$
5. Umgekehrt sortierter Eingabe, z.B.
   $[25, 24, 23, 22, 21, 20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]$

Führen Sie die `main`-Methode aus, um sich die Ausgabe anzuschauen!

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

1 Implementierung InsertionSort, SelectionSort und BubbleSort (Pflichtaufgabe)

Implementieren Sie die in der Vorlesung vorgestellten Algorithmen InsertionSort, SelectionSort und BubbleSort in den dafür vorgesehenen Klassen! Nutzen Sie die Tests zur Überprüfung der Korrektheit und prüfen Sie die ausgegebenen Werte auf Plausibilität.

Tipp: Wenn Sie den Algorithmus debuggen, werden Ihnen die `Sortables` wie eine Liste im Debugger angezeigt, Sie können den Ablauf also verfolgen.

2 Auswertung (Pflichtaufgabe)

Führen Sie die Tests mit geeigneten Arraygrößen aus[1] und erzeugen Sie aus den gesammelten Daten ~~zwei~~ Diagramme, um folgende Fragestellungen zu beleuchten. Stellen Sie dafür nicht einfach alle vorhandenen Daten dar, sondern wählen Sie überlegt! Geben Sie jeweils auf Basis der Diagramme auch eine textuelle Antwort!

- Wie lässt sich das gemessene Laufzeitverhalten der Algorithmen bei unsortierten Eingaben relativ zueinander beschreiben? Welchen Algorithmus würden Sie wählen, wenn Ihre Eingabe oft diese Struktur hat?
- Wie lässt sich das gemessene Laufzeitverhalten der Algorithmen bei größtenteils sortierten Eingaben relativ zueinander beschreiben? Welchen Algorithmus würden Sie wählen, wenn Ihre Eingabe oft diese Struktur hat?
- Wie lässt sich das gemessene Laufzeitverhalten der Algorithmen bei sortierten Eingaben mit wenigen Vertauschungen relativ zueinander beschreiben? Welchen Algorithmus würden Sie wählen, wenn Ihre Eingabe oft diese Struktur hat?
- Welcher Algorithmus ist in Situationen vorzuziehen, in denen das Vertauschen teuer ist, z.B. weil es sich um große Datensätze handelt?
- Wie verhält sich bei den betrachteten Eingaben jeweils die Laufzeit für teilweise unsortierte Eingaben oder Eingaben mit wenigen Vertauschungen zum völlig unsortierten Fall?

3 Implementierung des verbesserten Insertion-Sort-Algorithmus (Optional)

In der Vorlesung haben Sie eine Idee Verbesserung des Insertion-Sort-Algorithmus kennengelernt, die den Einfügepunkt mit binärer Suche findet. Implementieren Sie diesen Algorithmus und stellen Sie graphisch dar, wie sich die Anzahl der Vertauschungen reduziert (für verschiedene Eingabegrößen).

4 Implementierung von ShakerSort (Optional)

Der Algorithmus BubbleSort zeichnet sich dadurch aus, dass große Werte schnell an das Ende des Arrays verschoben werden (sie werden als „Bubbles“ bezeichnet, die „nach oben steigen“). Leider wandern

[1] Ändern Sie dafür die Konstante in der Klasse `main`

kleine Werte im Gegenzug nur sehr langsam nach unten, nämlich maximal um einen Schritt pro Durchlauf. Daher werden die kleinen Werte scherzhaft als „Turtles“ bezeichnet. Steht der kleinste Wert am Ende, dann braucht BUBBLESORT in jedem Fall n Durchläufe und damit die maximale Anzahl von Vergleichen. Um dieses Problem zu beheben, gibt es eine Variante von BUBBLESORT, die das Array abwechselnd von unten nach oben (links nach rechts) und von oben nach unten (rechts nach links) durchläuft. Diese Variante wird (wegen der Hin- und Herbewegung) als SHAKERSORT bezeichnet. Implementieren Sie diese Variante und vergleichen Sie graphisch mit BUBBLESORT. Für welche Eingaben können Sie eine Validieren, dass SHAKERSORT eine Verbesserung darstellt?

Hinweis: In den Details gibt es gewisse Freiheitsgrade. Treffen Sie eine Entscheidung oder implementieren Sie eine Variante, wie sie im Internet (z.B. auf der deutschen Wikipedia) beschrieben wird.

5 Implementierung von SHELLSORT (Optional)

SHELLSORT ist eine ziemlich clevere Variante von INSERTIONSORT. SHELLSORT ist dafür bekannt, unter den leicht zu implementierenden Algorithmen einer der schnellsten zu sein – beweisen lässt sich das allerdings kaum. Implementieren Sie SHELLSORT und vergleichen Sie graphisch mit einem oder mehreren der anderen Algorithmen. Können Sie den Vorsprung von SHELLSORT bestätigen?