technische universität dortmund

Fakultät für Informatik
Lehrstuhl 11

Erik Thordsen
Lena Krieg
Sebastian Konietzny

Sommersemester 2023
8. Mai

DAP2 Praktikum – Blatt 6

Abgabe: ab 22. Mai

Studienleistung

- Zum Bestehen des Praktikums muss jeder Teilnehmer die folgenden Leistungen erbringen:
  - Es müssen mindestens 50 Prozent der Punkte in den Kurzaufgaben erreicht werden.
  - Es müssen mindestens 50 Prozent der Punkte in den Langaufgaben erreicht werden.
- Im Krankheitsfall kann ein Testat bei Vorlage eines Attests in der folgenden Woche nachgeholt werden.
- Wenn ein Praktikumstermin auf einen Feiertag fällt, müssen Sie sich an einem beliebigen anderen Praktikumstermin in der gleichen Woche testieren lassen.
- Ansonsten kann die Testierung **nur in der zugeteilten Gruppe** garantiert werden.
- Bitte bereiten Sie den Tester sowie in den Aufgaben angegebene Beispieltests vor, bevor Sie sich testieren lassen!

Wichtige Information (im Moodle verfügbar)

- Beachten Sie die Erklärung des **Ablaufs (Blatt A)**.
- Beachten Sie die **Regeln und Hinweise (Blatt R)** in der aktuellsten Version!
- Beachten Sie die **Hilfestellungen (Blatt H)** in der aktuellsten Version!

Kommandozeile Tester

Sie finden im Moodle eine Datei `Test.jar`. Fuer das Testen Ihrer Lösung laden Sie diese herunter und geben den unten stehenden Kommandozeilen Befehl ein. **Es ist möglich, dass die Tests verzögert zur Verfügung gestellt oder vervollständigt werden!**

```
java -jar <path-to-moodle-jar>/Test.jar -s <path-to-solution> 6 -e
```

Langaufgabe 6.1: Radixsort (16 Punkte)

In der Vorlesung haben Sie den Algorithmus *Radixsort* kennengelernt, der oft damit motiviert wird, dass eine Liste von mehrdimensionalen Schlüsseln sortiert werden soll. Tatsächlich ist Radixsort aber auch gut dazu geeignet, eine Liste von Ganzzahlen zu sortieren. Dabei wird jeder `int`-Wert als ein Schlüssel mit vier Komponenten interpretiert, wobei jede Komponente eines der vier Bytes ist, welche den `int`-Wert bilden. Das höchstwertigste Byte (Most-Significant-Digit, MSD) ist die erste Komponente, und das niederwertigste Byte (Least-Significant-Digit, LSD) ist die letzte Komponente. Beispiel:

Wert in Basis-10-Darstellung:				1661914940
Wert in 32-Bit-Binärdarstellung:				01100011 00001110 11001111 00111100
Wert in Byte-Komponenten:	99 Komp. 1	14 Komp. 2	207 Komp. 3	60 Komp. 4

Jedes Byte entspricht dabei einer Stelle in der Basis-256-Darstellung des Wertes:

$$1661914940 = 99 \cdot 256^3 + 14 \cdot 256^2 + 207 \cdot 256^1 + 60 \cdot 256^0.$$

Die Kernaufgabe diese Blattes ist es, zwei Varianten von Radixsort für Ganzzahlen implementieren: Least-Significant-Digit-First (LSD) und Most-Significant-Digit-First (MSD). Für beide Varianten ist es wichtig, dass Sie ein Array nach dem Wert eines bestimmten Bytes sortieren können.

Legen Sie eine Klasse `AufgabeB6A1` an, in welcher Sie folgende Methoden implementieren:

- Die Methode `public static int[] readInput()` soll alle Ganzzahlen aus Standard-In einlesen. Wie in den vorigen Wochen sollen nur Eingaben akzeptiert werden, die vollständig aus Ganzzahlen bestehen. Die eingelesenen Ganzzahlen sollen als `int[]` zurückgegeben werden. Entstehende `NumberFormatException` sollen von dieser Methode weitergeleitet werden.

- Der Konstruktor `public AufgabeB6A1(int[] data)` soll eine neue Instanz dieser Klasse erstellen und sich das übergebene `data` in einem Attribut mit dem selben Namen speichern.

- Die Methode `public void sortByByte(int l, int r, int b)` soll das Arrayintervall `this.data[l, r]` *absteigend und stabil* nach dem `b`-niederwertigsten Byte sortieren (für `b` $= 0$ nach dem niederwertigsten Byte, für $b = 3$ nach dem höchstwertigsten). Dabei sollen Sie den Mechanismus von Counting-Sort verwenden (siehe Blatt 5), und ein Frequenzarray mit 256 Einträgen benutzen. Sie dürfen ein Hilfsarray der Länge `r` $-$ `l` $+ 1$ verwenden, und davon ausgehen, dass nur positive Ganzzahlen eingegeben werden.
  Tipp: Das `b`-niederwertigste Byte eines Integers `a` ist der Wert der `b`-niederwertigsten Stelle in Basis 256. Mathematisch entspricht dies $(\texttt{a}/256^{\texttt{b}}) \bmod 256$. Praktisch berechnen Sie die Division durch einen Bitshift, und den Modulo durch die Verundung mit einer Maske: `(a >> (8 * b)) & 0xFF`.

- Die Methode `public void lsdRadix()`, welche die Ganzzahlen in `this.data` absteigend sortiert. Diese Methode sortiert die Zahlen zuerst nach dem niederwertigsten Byte mittels `sortByByte`. Dann sortiert der Prozess die Zahlen nach dem zweit-niederwertisten Byte, dem dritt-niederwertigsten Byte, und schließlich nach dem höchstwertigsten (viert-niederwertigsten) Byte.

- Die Methode `public void msdRadix(int l, int r, int b)` soll MSD-Radixsort umsetzen. Der Algorithmus sortiert die Zahlen zunächst nach dem höchstwertigsten Byte. Dadurch wird das Array in Intervalle partitioniert, von denen jedes Intervall genau die Werte enthält, die das gleiche höchstwertigste Byte haben. Nun wird jedes Intervall rekursiv nach dem nächsten Byte sortiert, und rekursiv fortgefahren, bis entweder nach allen vier Bytes sortiert wurde, oder das betrachtete Intervall klein genug ist, um es mit einem naiven Algorithmus zu sortieren. Genauer soll die Methode wie folgt funktionieren:

  - Wenn die Methode aufgerufen wird, dann ist `this.data[l, r]` bereits nach den höchstwertigsten $(4 - (\texttt{b} + 1))$ Bytes sortiert. Der initiale Aufruf erfolgt mit $\texttt{l} = 0$, $\texttt{r} = \texttt{data.length} - 1$ und $\texttt{b} = 3$.
  - Wenn das Intervall bereits sehr kurz ist, genauer gesagt $\texttt{r} - \texttt{l} + 1 \leq 32$, dann sortieren Sie das Intervall `this.data[l, r]` absteigend mit dem naiven Algorithmus Insertion-Sort[1].
  - Ansonsten sortieren Sie `this.data[l, r]` absteigend nach dem `b`-niederwertigsten Byte ($\texttt{b} = 0$ entspricht dem niederwertigsten Byte). Dazu können Sie die Methode `sortByByte` verwenden. Damit Sie rekursiv fortfahren können, müssen Sie die Grenzen der Subintervalle mit gleichem `b`-niederwertigsten Byte bestimmen. Dazu können Sie ein Array C der Länge 257 berechnen, sodass $\texttt{self.data}[C[i+1]+1, C[i]]$ genau die Werte enthält, deren `b`-tes Byte den Wert i hat (insbesondere also $C[0] = r$ und $C[256] + 1 = l$). Nun sortieren Sie jedes Subintervall $\texttt{data}[C[i+1]+1, C[i]]$ rekursiv weiter.

  Tipp: Das Array C lässt sich im Aufruf von `sortByByte` leicht aus dem Frequenzarray berechnen. Sie können `sortByByte` modifizieren, sodass C zurückgegeben wird.

  Würden Sie statt Bytes die Dezimalstellen der Zahlen verwenden, so sähe ein Sortierschritt etwa so aus:

  `[123, 134, 234, 345, 222, 225]-> [`$\underbrace{\texttt{123, 134}}_{\text{rekursiv sortieren}}$`, `$\overbrace{\texttt{234, 222, 225}}^{\text{rekursiv sortieren}}$`, `$\underbrace{\texttt{345}}_{\text{rekursiv sortieren}}$`]`

- Die Methode `public static void main(String[] args)` enthält wie immer das ausführbare Programm Ihrer Implementierung. Sie sollen hier die Zahlen aus der Eingabe lesen, mehrere Instanzen der Klasse anlegen und die Liste mittels LSD- und MSD-Radixsort sortieren. Messen Sie die Laufzeit der zwei Varianten von Radixsort auf zufälligen Sequenzen. Welcher Algorithmus ist schneller? Vergrößern Sie oder verändern Sie die Struktur der Eingabe, bis Sie einen klaren Unterschied feststellen können. Achten Sie bei der Generierung der Eingabe darauf, welche Bytes wirklich von den Eingabewerten benutzt werden. Die sortierten Listen sollen dann entsprechend der Beispiele ausgegeben werden. Dazu dürfen Sie `Arrays.toString()` aus der Bibliothek `java.util.Arrays` verwenden. Sie dürfen davon ausgehen, dass nur positive Ganzzahlen eingegeben werden. Sie sollten jedoch wie immer bei ungültigen Eingaben passende Fehlermeldungen ausgeben. Überprüfen Sie die Korrektheit des Ergebnisses mit geeigneten Assertions.

```
// Input (beispielhaft): 1054010107 523127525 222740166 843867575 131404969
for i in {1..6}; do echo $((RANDOM<<15|RANDOM)); done | java AufgabeB6A1
// Output:
Before sorting: [1054010107, 523127525, 222740166, 843867575, 131404969]
Sort after LSD: [1054010107, 843867575, 523127525, 222740166, 131404969]
```

[1] siehe z.B. `https://www.geeksforgeeks.org/insertion-sort/`

```
LSD took: 0
Sort after MSD: [1054010107, 843867575, 523127525, 222740166, 131404969]
MSD took: 0
```

Geforderte Klassen und Methoden

```
public class AufgabeB6A1{
 public static void main(String[] args) {...}
 public static int[] readInput() throws NumberFormatException {...}
 public int[] data;
 public AufgabeB6A1(int[] data) {...}
 public void sortByByte(int l, int r, int b) {...}
 public void lsdRadix() {...}
 public void msdRadix(int l, int r, int b) {...}
}
```