Protokoll Übung 2

benötigte Zeit: 10

External Mergesort

Lösungsidee

External Mergesort wird verwendet, um eine große Menge Daten zu sortieren, die nicht in den RAM geladen werden können und deshalb in Dateien abgespeichert werden müssen.

Um dies umzusetzen, wird ein file_manipulator verwendet. Dieser regelt unter anderem das Kopieren und initiale Aufteilen der Files sowie das Befüllen mit zufälligen Werten. Beim Aufteilen werden Werte aus der Original-Datei ausgelesen und abwechselnd in zwei verschiedene Dateien geschrieben. Das Befüllen arbeitet mit zwei Zufallsverteilungen: die erste Verteilung wird verwendet um zu entscheiden welche Zeichenart geschrieben wird (Zahl, Groß- o. Kleinbuchstabe), die zweite Verteilung entscheidet dann über das Zeichen selbst.

Das Sortieren selbst wird durch den merge_sorter ummgesetzt. Dabei wird zuerst die zu sortierende Datei auf zwei Dateien aufgeteilt.
Als nächstes werden k Element pro Datei ausgelesen und in sortierter Reihenfolge in zwei abwechselnden Containern zwischengespeichert. Dies wird so oft wiederholt, bis beide Dateien ausgelesen wurden. Anschließend werden beide Container auf zwei weitere Dateien geschrieben. Das Einlesen von k Elementen und Aufteilen wird so oft wiederholt, bis das Sortieren abgeschlossen ist. Wobei k eine Potenz von 2 ist (1,2,4,8, …). Zusätzlich werden auch immer die einzulesenden und schreibenden Dateien abgewechselt. Das Sortieren ist abgeschlossen, wenn eine der einzulesenden Dateien leer ist.

Implementierung

siehe: “External-Mergesort”

Testfälle

Fall 1

```
Test: sorting an unsorted file, of which the solution is known:
Original:
b c a h i e g f d j
Sorted:
a b c d e f g h i j
Result: 1
```

Fall 2

```
Test: sorting a sorted.txt file. Sorting should not change the outcome:
Original:
a b c d e f g h i j k l
Sorted:
a b c d e f g h i j k l
Result: 1
```

Fall 3

```
Test: sorting an empty file:
Original:

Sorted:

Result: 1
```

Fall 4

```
Test: sorting a randomly filled file (10 elements of length=1 ):
Original:
6 7 V 0 W o 8 q 4 6
Sorted:
0 4 6 6 7 8 V W o q
Result: 1
```

Fall 5

```
Test: sorting a randomly filled file (111 elements of length=3 ):
Original:
r82 ak7 a66 LOu NCh x76 64v 5Cg rXG yii fQ3 ual t4x vI6 yv9 85e U6J piC
Sorted:
066 15c 18o 19S 1J4 1v9 2GJ 2In 2O7 2QV 2TS 2yY 30M 37s 3h1 3r3 4b7 4kR
Result: 1
```

Fall 6

```
Test: partitioning a file with an even number of elements:
Original file:
ab cd ef gh ij kl mn op
Partitioned files:
ab ef ij mn
cd gh kl op
Result: 1
```

Fall 7

```
Test: partitioning an empty file:
Original file:

Partitioned files:


Result: 1
```

Fall 8

```
Test: partitioning a file with an odd amount of elements:
Original file:
ab cd ef gh ij kl mn op qr
Partitioned files:
ab ef ij mn qr
cd gh kl op
Result: 1
```

Fall 9

```
Test: filling a file randomly with 10 elements of length=1 (standard):
Filled file:
M 4 P v z V W M b r
Result: 1
```

Fall 10

```
Test: filling a file randomly with 31 elements of length=4:
Filled file:
yGjI hrD5 04j2 2d4g hb6m fAXp mAQ2 P097 7m59 mdH2 2m8G 5W04 XCAY
Result: 1
```

Fall 11

```
Test: copying the content of a file to another file:
Source file:
rqgwhgr dkhjrgqiuzriu qe3i4zi2u3z5ru 4o5z23iu5zuio qwöer 43iu5z2
Copied file:
rqgwhgr dkhjrgqiuzriu qe3i4zi2u3z5ru 4o5z23iu5zuio qwöer 43iu5z2
Result: 1
```

Fall 12

```
Test: copying the content of an empty file to another file:
Source files:

Copied file:

Result: 1
```

Fall 13

```
Test: copying the content of a container to a file:
Copied file:
this is the content of the container
Result: 1
```

Fall 14

```
Test: copying the content of an empty container to a file:
Copied file:

Result: 1
```

Fall 15

```
Test: printing the contents of a file:
This is the content of the file that was requested to print!
Result: 1
```

Fall 16

```
Test: printing the contents of an empty file:

Result: 1
```