Lösung Aufgabe 2 “Nummernmerker”

Lösungsidee

Wir verwenden eine **Brute-Force**, die alle Möglichkeiten, die Zahl in die gewünschten Blöcke aufzuteilen, ausprobiert, und dann davon eine der Besten auswählt.

Sei n die Länge der Zahl. Im worst case ist diese 30. Als einfache Abschätzung sind maximal 15 Blöcke möglich, pro Block ergeben sich maximal 3 Möglichkeiten (2er, 3er oder 4er Block). Natürlich braucht man (wenn man größere Blöcke verwendet) weniger, aber unsere Schätzung liegt sowieso nur darüber. Also maximal 3^{15}=14.348.907 auszuprobierende Kombinationen (die tatsächliche Zahl liegt weit darunter).

Umsetzung

Als Programmiersprache wird Java wegen einer guten Performance (zwar etwas langsamer als, aber doch vergleichbar mit C++) und weitgehender Plattformunabhängigkeit gewählt.

Bibliotheken

- `java.io.File`, `java.io.FileReader`, `java.io.BufferedReader`: Um Dateien mit Zahlen lesen zu können
- `java.util.Arrays`: Für `Arrays.stream`, womit ein Array direkt in eine Stream der dann wiederum in einen durchgehbaren Iterator verwandelt werden kann
- `java.util.Iterator`: Um ebendiesen Iterator zu speichern

Klassen

`Nummernmerker`

Hauptklasse. Lädt zunächst die Datei bzw. die Argumente und führt den Algorithmus aus (wo immer alle Möglichkeiten ausprobiert werden, einen 2er, 3er oder 4er Block vorne abzuspalten), und lädt die Aufgabe / die Zahlen (siehe Verwendung). Um möglichst viel Speicherplatz zu sparen, wird die Brute-Force rekursiv und nicht iterativ implementiert - als depth-first-search. Nicht mehr benötigte Zwischenergebnisse der schon abgesuchten Pfade können dann von der Garbage Collection gelöscht werden. Und da die Rekursionstiefe maximal 15 beträgt, ist ein `StackOverflowError` ausgeschlossen.

`Aufteilung`

Wird für Aufteilungsobjekte benutzt. Speichert jeweils Aufteilung & die Zahl der Blöcke, die mit 0 beginnen.

Verwendung

Ausgeben einer optimalen Aufteilung einer Zahl: `java -jar Nummernmerker.jar <zahl>`
Ausgeben optimaler Aufteilungen mehrerer Zahlen: `java -jar Nummernmerker.jar <zahl_1> <zahl_2> <zahl_3> ... <zahl_n>`

Ausgeben optimaler Aufteilungen von Zahlen aus einer Datei: `java -jar Nummernmerker.jar <pfad_zur_datei>`

Beispiele

Die Aufteilungen der 6 angegeben Zahlen:

```
01 36 5400 606
00 54 800 0000 51 79 734
03 49 59 29 53 37 90 15 44 12 660
53 19 97 48 790 22 72 560 76 20 179
90 88 76 10 51 69 94 82 78 90 38 33 12 67
01 10 000 0001 1000 100 11 11 11 10 10 11
```

Quellcode

`Aufteilung.java`

```java
package appguru;

/**
 *
 * @author lars
 */

// Speichert eine Aufteilung einer Zahl
public class Aufteilung {

    public int bloeckeMitNull; // Zahl der Blöcke, die mit 0 beginnen
    public String darstellung; // Darstellung als Text

    // Konstruktor, erzeugt Aufteilung, setzt Variablen auf Anfangswerte
    public Aufteilung() {
        bloeckeMitNull = 0;
        darstellung = "";
    }

    // Anderer Konstruktor, setzt direkt Variablen auf gegebene Werte
    public Aufteilung(int bloeckeMitNull, String darstellung) {
        this.bloeckeMitNull = bloeckeMitNull;
        this.darstellung = darstellung;
    }

    // Fügt einen 2-4 stelligen Teil hinzu
    public Aufteilung addPart(String part) {
        // Wenn der Teil mit einer 0 beginnt
```

```
        if (part.charAt(0) == '0') {
            // Hat die einen zusätzlichen Block mit 0, und die Darstellung kommt hinzu
            return new Aufteilung(bloeckeMitNull + 1, darstellung + part + " ");
        }
        // Nur die Darstellung kommt hinzu
        return new Aufteilung(bloeckeMitNull, darstellung + part + " ");
    }

    @Override
    public String toString() {
        return darstellung; // Gibt einfach Darstellung zurück
    }
}
```

Nummernmerker.java

```
package appguru;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.util.Arrays;
import java.util.Iterator;

/**
 *
 * @author lars
 */
public class Nummernmerker {

    public static Aufteilung BESTE_AUFTEILUNG; // speichert die bisher beste Aufteilung

    // Zweigt von der Nummer einen Block der Länge 2-4 ab
    public static void abspalten(Aufteilung bisher, String nummer) {
        if (nummer.equals("")) {
            if (bisher.bloeckeMitNull < BESTE_AUFTEILUNG.bloeckeMitNull) {
                BESTE_AUFTEILUNG = bisher;
            }
            return;
        }
        if (bisher.bloeckeMitNull >= BESTE_AUFTEILUNG.bloeckeMitNull) {
            return;
        }
        for (int i = 1; i < Math.min(nummer.length(), 4); i++) {
            String part = nummer.substring(0, i + 1);
            String remainder = nummer.substring(i + 1);
            abspalten(bisher.addPart(part), remainder);
        }
```

```
33      }
34
35      public static void main(String[] args) {
36          Iterator<String> nummern;
37          if (args.length == 1) { // Ein Argument gegeben
38              // Ist es eine Zahl?
39              boolean istZahl = true;
40              for (int i = 0; i < args[0].length(); i++) {
41                  char c = args[0].charAt(i);
42                  if (c < '0' || c > '9') {
43                      istZahl = false;
44                      break;
45                  }
46              }
47
48              // Wenn es eine Zahl ist, dann einfach zum durchgehen vorbereiten
49              if (istZahl) {
50                  nummern = Arrays.stream(new String[]{args[0]}).iterator();
51              } else {
52                  // Ansonsten: Datei lesen
53                  File file = new File(args[0]);
54                  if (!file.exists() || !file.canRead()) {
55                      System.out.println("Datei nicht existent oder nicht lesbar");
56                  }
57                  BufferedReader reader;
58                  try {
59                      reader = new BufferedReader(new FileReader(file));
60                  } catch (Exception ex) {
61                      System.out.println("Datei nicht existent oder nicht lesbar");
62                      return;
63                  }
64                  // Iterator erzeugen, der die Zahlen der Zeilen als Strings durchgeht
65                  nummern = new Iterator() {
66                      private String line;
67
68                      @Override
69                      public boolean hasNext() {
70                          return line != null;
71                      }
72
73                      @Override
74                      public String next() {
75                          try {
76                              return (line = reader.readLine());
77                          } catch (Exception ex) {
78                              System.out.println("Datei nicht existent oder nicht lesbar");
79                              System.exit(0);
80                          }
```

```
                        return "";
                    }
                };
            }
        } else {
            // Ansonsten: einfach Argumente als Zahlen nehmen
            nummern = Arrays.stream(args).iterator();
        }
        // Alle Nummern durchgehen
        for (String nummer : new Iterable<String>() {
            @Override
            public Iterator<String> iterator() {
                return nummern;
            }
        }) {
            // Für jede die beste Aufteilung berechnen
            BESTE_AUFTEILUNG = new Aufteilung(Integer.MAX_VALUE, "KEINE"); // erstmal beste
                 Aufteilung zurücksetzen
            abspalten(new Aufteilung(), nummer); // beste Aufteilung berechnen
            System.out.println(BESTE_AUFTEILUNG); // und ausgeben
        }
    }
}
```