Aufgabe 1 Florian Hofer, Werner Nutt, Thomas Tschager

Hamming-Codes

In dieser Programmierübung sollen Sie einen Kodierer und einen Dekodierer für Hamming-Codes implementieren. Das Ergebnis trägt 5% zur Endnote bei.

Arbeitsweise: Sie können allein oder in einem Team aus zwei Studenten arbeiten (ein Team braucht nur eine Lösung einzureichen). Formatieren Sie Ihr Ergebnis als Archiv (*archive file*), das zwei Dateien enthält: (i) eine Datei *HammingCode.java* und (ii) eine Datei *Beschreibung.pdf*, die eine kurze Beschreibung (1 Seite) Ihrer Lösung enthält. Kommentieren Sie Ihren Code. In jeder der beiden Dateien sollten Ihre Namen und Matrikelnummern stehen.

Aufgabenstellung: Implementieren Sie in HammingCode.java zwei Funktionen (= statische Methoden)

void encode(String message, String filename) ,

String decode(String filename)

wie folgt:

- Die Funktion *encode* erhält als Argumente einen String und einen Dateinamen. Sie zerlegt jedes Zeichen im String in zwei 4-Bit-Teile und kodiert jeden 4-Bit-Teil mit dem (7,4)-Hamming-Code, der in der Vorlesung[2] besprochen wurde. Der kodierte String wird dann unter dem angegebenen Dateinamen gespeichert.
- Die Funktion *decode* dekodiert entsprechend Dateien, die im (7,4)-Hamming-Code codiert sind.
- Für die kodierte Version können Sie wählen, ob Sie Daten in binärer Darstellung speichern möchten (platzsparender) oder ob Sie 0en und 1en als Zeichen schreiben möchten (wahrscheinlich einfacher für das Debuggen).
- Natürlich sollte Ihr Programm in der Lage sein, Nachrichten mit 1-Bit-Fehlern zu dekodieren.

Die folgenden Funktionen können hilfreich sein:

- *Integer.toBinaryString(String.charAt(i))* – übersetzt ein Zeichen in einen Binärstring
- *Character.toChars(int i)* – übersetzt eine Integer-Zahl in ein Zeichen
- *Integer.parseInt(s, 2)* – Parst einen Binärstring in eine Integer-Zahl

Abgabe: Dienstag, 3. April 2018, 23:55 via Moodle.

[2] Der Hamming-Code aus der Vorlesung besteht aus vier Datenbits (d_1 .. d_4), gefolgt von drei Paritätsbits ($p_1 = d_1 \text{ xor } d_2 \text{ xor } d_3$, $p_2 = d_2 \text{ xor } d_3 \text{ xor } d_4$, $p_3 = d_3 \text{ xor } d_4 \text{ xor } d_1$). In der Literatur werden Daten- und Paritätsbits oft in der Reihenfolge gemischt.