Universität Bremen

Praktische Informatik 3 WS 2023/24

Thomas Barkowsky
Tede von Knorre
Yaroslav Purgin
Tim Sperling

2. Übungsblatt

Ausgabe: 30.10.2023
Abgabe: 06.11.2023

2.1 *Geheime Nachrichten* *6 Punkte*

Die *Enigma* war ein in Deutschland seit den Zwanzigerjahren des letzten Jahrhunderts entwickeltes Gerät zur Verschlüsselung militärischer Nachrichten. Die mit der *Enigma* generierten Kodierungen sind im Zweiten Weltkrieg in England unter anderem unter Mitarbeit von *Alan Turing* erfolgreich entschlüsselt worden. Das besondere an der *Enigma* war die Verwendung von auf drehbaren Scheiben angeordneten Zeichen, so dass jeder einzelne Buchstabe mit einem von dem vorherigen Buchstaben abweichenden Kodier-Alphabet verschlüsselt wurde.

In dieser Übungsaufgabe wollen wir eine vereinfachte und abgewandelte Simulation eines solchen Kodierungsverfahrens implementieren.

Wir benötigen dafür Kodierscheiben des Datentyps `rotor` (als String repräsentiert, aber als zirkuläre Struktur gedacht), z.B.:

```
rotor1 = "JKPQRLYZADGVWXHMNOIEFBSTUC"
```

Wie bei der originalen *Enigma* beschränken wir uns auf die 26 Großbuchstaben des Alphabets. Leerzeichen müssen durch ein vorhandenes Zeichen (z.B. `"X"`) ersetzt werden, Zahlen müssen ausgeschrieben werden, etc.

Wir können nun zunächst eine Funktion `encodeChar` schreiben, die ein Zeichen unter Verwendung einer vorgegebenen Kodierscheibe in ein anderes Zeichen verschlüsselt:

```
encodeChar 'A' rotor1 ⇝ 'J'
encodeChar 'C' rotor1 ⇝ 'P'
```

Der Buchstabe `"A"` wird also (als erster Buchstabe des Alphabets) durch `"J"` verschlüsselt (erstes Zeichen auf der Kodierscheibe), etc. Für die Ermittlung der Position eines Buchstabens im Alphabet könnte die Funktion `ord` (importiert aus `Data.Char`) nützlich sein, die den ASCII-Wert eines Buchstabens liefert. Die Funktion `chr` ist übrigens die Umkehrung der Funktion `ord`; sie könnte im weiteren Verlauf dieser Übung auch noch hilfreich werden...

Nun wollen wir unsere Kodierscheiben drehen können, um für jeden Buchstaben ein neues Kodier-Alphabet zu erhalten. Dies erreichen wir mit der Funktion `turnRotor`:

```
turnRotor rotor1 ⇝ "KPQRLYZADGVWXHMNOIEFBSTUCJ"
```

Die Scheibe wird also gewissermaßen nach links gedreht: Das Zeichen `"K"`, das zuvor an zweiter Stelle stand, ist nun auf der ersten Position, `"J"` ist nun an letzter Stelle.

Weil wir später in der Lage sein wollen, die Scheibe gleich um mehrere Positionen zu rotieren, benötigen wir auch noch die Funktion `turnRotorByN` mit einem weiteren Parameter des Typs `Int`, der Werte größer oder gleich Null annehmen darf. Beispielsweise dreht

```
turnRotorByN rotor1 3 ⇝ "QRLYZADGVWXHMNOIEFBSTUCJKP"
```

die Scheibe `rotor1` gleich um drei Positionen weiter.

Jetzt sind wir endlich soweit, dass wir die eigentliche Kodierungsfunktion `enigma` schreiben können. Diese Funktion wird mit vier Parametern aufgerufen:

- der zu verschlüsselnden Zeichenkette,
- der verwendeten Kodierscheibe,
- der Anfangsposition, auf die die Scheibe gedreht wird, sowie
- die Anzahl der Schritte, um die die Scheibe nach jedem kodierten Zeichen weitergedreht wird.

Beispiel:

```
enigma "ABBA" rotor1 3 2 ⇝ "QYAD"
```

In diesem Beispiel verwenden wir wieder die Kodierscheibe `rotor1`, die gleich zu Anfang um drei Positionen weiter gedreht wird, so dass der Buchstabe `"A"` in ein `"Q"` umkodiert wird. Danach wird die Scheibe um zwei Positionen weitergedreht, so dass das Kodieralphabet nun mit `"LYZA"` beginnt und `"B"` somit in `"Y"` umkodiert wird. Danach wird die Scheibe wieder um zwei Schritte gedreht und das nächste `"B"` wird zu `"A"`, und so weiter.

Jetzt brauchen wir das Ganze natürlich auch wieder *retour*: Wir haben einen verschlüsselten Text, kennen die verwendete Kodierscheibe, deren Anfangsposition und die Rotations-Schrittweite und können daher wie folgt dekodieren:

```
deEnigma "QYAD" rotor1 3 2 ⇝ "ABBA"
```

Kennen wir die entsprechenden Parameter nicht, dann geht die Entschlüsselung natürlich böse schief...

Für die Funktion `deEnigma` ist es hilfreich, wenn wir zunächst die Funktion `decodeChar` definieren, die ein Zeichen unter Verwendung einer gegebenen Kodierscheibe (in einer bestimmten Position) zurückübersetzt:

```
decodeChar 'P' rotor1 ⇝ 'C'
```

`"P"` wird hier in `"C"` zurückübersetzt, weil in `rotor1` das `"P"` an der dritten Stelle steht, und das entspricht dem `"C"` im gewöhnlichen Alphabet. Für die Realisierung von decodeChar ist es also hilfreich, eine Funktion `indexOfChar` zu haben, die die Position eines Zeichens in einer Zeichenkette (hier: unserer Kodierscheibe) bestimmt. Beachtet, dass die Zählung bei 0 beginnt:

```
indexOfChar 'P' rotor1 ⇝ 2
```

Jetzt können wir Nachrichten chiffrieren und auch wieder entschlüsseln, wenn wir die richtigen Parameter dafür kennen. Aber was wenn nicht und wir den verwendeten Code 'knacken' müssen?

2.2 *Code Breaking* *4 Punkte*

Für eine gegebene Kodierscheibe können wir mit einer kleinen Funktion einfach alle Parameter durchprobieren, bis wir die ursprüngliche Nachricht gefunden haben. Wir machen das mit der Funktion `breakEnigma`, die mit einer verschlüsselten Nachricht und einer Kodierscheibe als Übergabeparameter alle 26 möglichen Anfangspositionen durchprobiert, sowie alle 26 Möglichkeiten, die Scheibe nach jedem Buchstaben weiterzudrehen:

```
breakEnigma "QYAD" rotor1 ⇝
"DGIJ\nDFGG\nDEED\nDDCA\nDCAX\nDBYU\nDAWR\nDZUO\nDYSL\nDXQI\nDWOF\nDVMC\nDUKZ\n..
```

Nach jedem Dekodiervorschlag fügen wir hier wieder einen Zeilenwechsel ein (`"\n"`), um das Ganze mit `putStr` zeilenweise ausgeben zu können. In unserem Fall würde das 676 Zeilen erzeugen, was nicht wirklich praktisch ist. Wir wollen daher unsere obige Funktion noch einmal verbessern, in dem wir ihr eine Zeichenkette mitgeben, von der wir vermuten, dass sie in dem dechiffrierten Text enthalten sein könnte (Artikel, Namen, Grußformeln, etc.). Diese Funktion `breakEnigmaWithGuess` würde dann so funktionieren:

```
enigma "EINXHUNDXLIEFXINXDIEXKUECHE" rotor1 13 9 ⇝ "ORMRJPKRKZWSTOLLUXKYARTNTVO"
putStr (breakEnigmaWithGuess "ORMRJPKRKZWSTOLLUXKYARTNTVO" rotor1 "UND") ⇝
LVGWMMFEAAUXZGEVGWITVUNDTXIL
EINXHUNDXLIEFXINXDIEXKUECHE
STVCJTJWNYSLJYGIPSUNDNUBWYS
```

Für die Definition von `breakEnigmaWithGuess` kann die Funktion `isInfixOf` (importiert aus `Data.List`) gute Dienste leisten.

Mit der Vermutung, dass der ursprünglich verschlüsselte Text die Zeichenfolge `"UND"` enthalten könnte, erhalten wir also nur noch drei (statt 676) Dekodierungsvorschläge, von denen einer unsere gesuchte Botschaft ist...

Wenn ihr hier angekommen seid, dann könnt ihr euch ja mal mit der Dechiffrierung dieser Botschaft versuchen:

EQWJTEMXUPICNOLPUNUDJDOWIZJANGKXDJFONQNRQQYGFLMGRCENKUIIEKIABTIXXJCNTH

Tipp: Auch diese geheime Nachricht wurde mit `rotor1` verschlüsselt und es geht um das Datum des Eintreffens eines Schiffes in einem bestimmten Hafen.