Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Programación Avanzada
Primer Cuatrimestre de 2017

Trabajo Práctico

Este trabajo práctico debe ser resuelto en grupos de máximo tres personas. El proyecto podrá ser entregado cualquier fecha anterior al **Martes 20 de Junio utilizando Moodle.**.

El objetivo del presente proyecto es desarrollar un algoritmo simple de encriptación simétrica en Haskell. La encriptación simétrica se basa en codificar un mensaje utilizando una clave, la cual también sirve para desencriptar el mensaje.

Para encriptar mensajes trabajaremos al nivel de Bits, es decir, tanto el mensaje a codificar como la clave son cadenas de Bits. Para encriptar un solo bit se puede utilizar el XOR (para el cual utilizaremos el símbolo $\oplus$), el operador $\oplus$ cumple la siguiente propiedad:

$$(x \oplus y) \oplus y = x$$

Es decir, si pensamos y como una clave, entonces aplicando dos veces $\oplus$ obtenemos el valor original. Para encriptar un mensaje más largo se aplica el operador $\oplus$ entre la clave y el mensaje en cada posición. En la práctica las claves pueden ser de diferentes longitudes (actualmente, se utilizan claves de mas de 128 Bits), en este proyecto, para simplificar la presentación, utilizaremos claves de 8 Bits. El algoritmo consta de los siguientes pasos:

- Se elige una clave de k Bits: $c = [c_0, \ldots, c_{k-1}]$,
- Se traduce el texto a encriptar a un lista de bits utilizando el código ASCII (7 bits),
- Se particiona el mensaje original en Bits de longitud m: $[x_0, x_1, \ldots, x_{m-1}]$, en $(m \ div \ k) + 1$ listas de k bits: $[[x_0, \ldots, x_{k-1}], [x_k, \ldots, x_{2k-1}], \ldots]$, que llamaremos *chunks*.
- Cada *chunk* se encripta utilizando la clave y el *chunk* anterior. Es decir, si $[x_0, \ldots, x_{k-1}]$ es el primer chunk, entonces lo encriptamos como $[x_0 \oplus c_0, x_1 \oplus c_1, \ldots, x_{k-1} \oplus c_{k-1}]$. Si tenemos encriptado el $i-1$-ésimo chunk como $[e_0, \ldots, e_{k-1}]$ entonces el i-ésimo chunk $[x^i_0, \ldots, x^i_{k-1}]$ se encripta como sigue: $[(x^i_0 \oplus e_0) \oplus c_0, \ldots, (x^i_{k-1} \oplus e_{k-1}) \oplus c_{k-1}]$.
- Finalmente, todos los chunks son unidos y transformados a texto utilizando el código ASCII.

Para desencriptar un texto dada la clave $[c_0, \ldots, c_{k-1}]$ se procede similarmente:

- Se pasa el texto a encriptar a bits utilizando el código ASCII (7 bits),
- Se particiona el mensaje original en Bits de longitud m: $[x_0, x_1, \ldots, x_{m-1}]$, en $(m \ div \ k) + 1$ listas de k bits: $[[x_0, \ldots, x_{k-1}], [x_k, \ldots, x_{2k-1}], \ldots]$,
- Cada *chunk* se desencripta utilizando la clave y el *chunk* anterior. Es decir, si $[x_0, \ldots, x_k]$ es el primer chunk, entonces lo desencriptamos como $[x_0 \oplus c_0, x_1 \oplus c_1, \ldots, x_{k-1} \oplus c_{k-1}]$. Si la encriptación el $i-1$-ésimo chunk es $[e_0, \ldots, e_{k-1}]$ entonces el i-ésimo chunk $[x^i_0, \ldots, x^i_{k-1}]$ se desencripta como sigue: $[(x^i_0 \oplus e_0) \oplus c_0, \ldots, (x^i_{k-1} \oplus e_{k-1}) \oplus c_{k-1}]$.
- Finalmente, todos los chunks son unidos y transformados a texto utilizando el código ASCII.

Para implementar este algoritmo utilizaremos las siguientes funciones.
Primero definimos el tamaño de la clave y el tipo de Bits.

```
keySize::Int
keySize = 8
```

Definimos un tipo simple representando bits.

```
data Bit = O | I deriving (Show, Eq)
```

También definimos el xor:

```
xor :: Bit -> Bit -> Bit
```

Para pasar Char a ASCII y de ASCII a Char necesitaremos dos funciones:

```
charToBits :: Char -> [Bit]
```

que dado un `Char` retorna su código ASCII (siete bits). Y otra función correspondiente que dado un código ASCII retorna el `Char`.

```
bitsToChar :: [Bit] -> Char
```

Con estas funciones, podemos definir otras funciones que transformen texto a bits y viceversa.

```
textToBits :: [Char] -> [Bit]
```

```
bitsToText ::[Bit] -> [Char]
```

Finalmente tenemos funciones que encriptan y desencriptan bits:

```
-- Dado un texto y una clave, encripta un texto
encrypt :: [Char] -> [Bit] -> [Char]
```

```
-- Dado un texto y una clave, desencripta un texto
desencrypt :: [Char] -> [Bit] -> String
```

Para definir estas funciones, podemos utilizar funciones auxiliares que encriptan y desencriptan lists de chunks:

```
-- Dada una secuencia de chunks y una clave  encripta  los chunks usando la clave
encryptChunks :: [[Bit]] -> [Bit] -> [[Bit]]
```

```
-- Dada una secuencia de chunks desencripta  y un clave, desencripta los chunks usando la clave
desencryptChunks :: [[Bit]] -> [Bit] -> [[Bit]]
```

Para la resolución del trabajo práctico se debe:

- Implementar las funciones descriptas arriba,
- Bajar el texto en el archivo `encriptado.txt` y desencriptarlo usando la clave `[O,O,I,O,I,I,O,I]`,
- Se debe subir su código fuente por medio del moodle, utilizando el formato: Apellido1Apellido2Apellido3.hs.