Trabajo Práctico - Programación Funcional

Cifrando Mensajes

Introducción a la Programación - Primer cuatrimestre de 2024

Fecha límite de entrega: Viernes 24 de mayo - 23:55 hs

Introducción

El objetivo de este Trabajo Práctico es aplicar los conceptos de programación funcional vistos en la materia para programar algunos métodos de cifrado simples utilizando Haskell. Para ello se deben implementar una serie de ejercicios respetando su especificación, además de otros criterios que se detallan en las pautas de entrega.

Pautas de Entrega

Para la entrega del trabajo práctico se deben tener en cuenta las siguientes consideraciones:

- El trabajo se debe realizar en grupos de cuatro estudiantes *de forma obligatoria.*
  No se aceptarán trabajos de menos ni más integrantes.
  Ver aviso en el campus de la materia donde indica el link para registrar los grupos.
- Se debe implementar un conjunto de casos de test para todos los ejercicios (no es obligatorio para las funciones auxiliares que definan).
- Los programas deben pasar con éxito los casos de test entregados por la cátedra (en el archivo test-catedra.hs), sus propios tests, y un conjunto de casos de test "secreto".
- El archivo con el código fuente debe tener nombre `Solucion.hs`. Además, en el archivo entregado debe indicarse, en un comentario arriba de todo: nombre de grupo, nombre, email y DNI de cada integrante. En caso de que algún integrante haya abandonado la materia, aclararlo entre paréntesis. Recordar que no se acepta un número de integrantes diferente a 4, pero aceptamos que queden menos en caso de que alguien abandone.
- El código debe poder ser ejecutado en el GHCI instalado en los laboratorios del DC, sin ningún paquete especial.
- No está permitido alterar los nombres de las funciones a implementar ni los tipos de datos. Deben mantenerse tal cual descargan el template.
- Pueden definir todas las funciones auxiliares que se requieran.
- No está permitido utilizar técnicas no vistas en clase para resolver los ejercicios (como por ejemplo, alto orden). En el campus, en la pestaña TP Grupal pueden encontrar un listado de todas las funciones válidas a usar (agregamos algunas extras no válidas para el parcial). Si usan funciones fuera de esa lista, el ejercicio sumará 0 puntos.

Se evaluarán las siguientes características:

- **Correctitud**: todos los ejercicios deben estar bien resueltos, es decir, deben respetar su especificación.
- **Declaratividad**: los nombres de las funciones que se definan, así como los nombres de las variables, deben ser apropiados.
- **Consistencia**: el código debe atenerse al uso correcto de las técnicas vistas en clase como recursión o *pattern matching.*
- **Prolijidad**: evitar repetir código innecesariamente y usar adecuadamente las funciones previamente definidas (por el enunciado o por ustedes mismos). En caso de funciones complejas, analizar si es conveniente realizar comentarios sobre el código.
- **Testing**: Todos los ejercicios del enunciado deben tener sus propios casos de test que pasen correctamente. Un ejercicio sin casos de test se considera no resuelto. Los casos de test deben cubrir distintos escenarios de uso de las funciones.

Método de entrega

Se espera que el trabajo grupal lo realicen de forma incremental utilizando un sistema de control de versiones, como Git. Para la entrega, sólo vamos a trabajar con GitLab (pueden ser `git.exactas.uba.ar` si tienen usuario, o deben crearse un usuario en `gitlab.com`). El repositorio debe ser privado, deben estar todos los integrantes del grupo y, además, deben agregar con rol "Reporter" un usuario de los docentes:

- Si usan `git.exactas.uba.ar`: Deben agregar al usuario *docentes.ip*
- Si usan `gitlab.com`: Deben agregar al usuario *ip.dc.uba*

La entrega debe ser realizada ***únicamente*** por un integrante del grupo. Se entregará mediante el campus de la materia un archivo de texto indicando la URL del repositorio y un commit particular. Los docentes descargarán el código para ese commit. En caso que el link o el commit no sean válidos, el trabajo estará desaprobado. En la pestaña TP Grupal del campus, encontrarán un template de un archivo de texto con todos los datos que deben completar.

Antes de enviar el trabajo, se recomienda fuertemente:

- Clonar el repositorio con el link y commit proporcionados. Chequear que ambos son válidos.
- Ejecutar los casos de test de la cátedra y sus propios casos de test, sobre el repositorio recién clonado.
- Todos los casos de test deben pasar satisfactoriamente, y el archivo con la implementación debe poder cargarse sin generar errores.

Estos pasos los realizaremos en una de las clases de laboratorio para despejar dudas al respecto.

Especificación

Parte I: Cifrado César

Una de las técnicas de cifrado más simples es conocida como *El código de César* ó *Cifrado César*. Este es un cifrado por sustitución en el que una letra en el texto original es reemplazada por otra letra que se encuentra un número fijo de posiciones más adelante en el alfabeto. Por ejemplo, utilizando este método con *desplazamiento* = 3, se tendría el siguiente cifrado:

```
mensaje original: introduccion a la programacion
 mensaje cifrado: lqwurgxfflrq d od surjudpdflrq
```

Por simplicidad, cifraremos sólo las letras minúsculas de los mensajes. Además, vamos a eliminar la letra "ñ" del conjunto de letras minúsculas y no consideraremos a las vocales con tildes como letras minúsculas, también por simplicidad.

El orden natural del alfabeto será considerando su inicio en la letra "a" (posición 0) y finalizando en la letra "z" (en la posición 25). Por esto podemos mencionar que desplazar 2 posiciones hacia adelante a la letra "a" hace referencia a la letra "c", mientras que desplazar 2 posiciones hacia atrás a la letra "a" hace referencia a la letra "y".
Además, los espacios y otras letras no serán cifrados pero deben mantenerse en su posición original.

En relación al cifrado César, se pide implementar en Haskell los siguientes ejercicios:

Ejercicio 1

`problema esMinuscula` (c: Char) : Bool {
`requiere:` {True}
`asegura:` {**res** es igual a true si y sólo si c es una letra minúscula (entre "a" y "z" ambos inclusive, pero sin incluir "ñ" ni vocales con tildes)}
}

Ejemplo:

```
entrada: c = 'd'
res: True
```

Ejercicio 2

`problema letraANatural` (c: Char) : $\mathbb{Z}$ {
`requiere:` {c es una letra minúscula}
`asegura:` {**res** es igual a un número entre 0 y 25 (ambos inclusive) que indica el orden en el cual la letra se encuentra en el abecedario.}
}

Ejemplo:

```
entrada: c = 'b'
res: 1
```

Ejercicio 3

`problema desplazar` (c: Char, n: $\mathbb{Z}$) : Char {
`requiere:` {True}
`asegura:` {Si c es minúscula, entonces **res** es igual a la letra minúscula que esté n posiciones más adelante o atrás en el abecedario (rotando en caso de llegar al final o al principio)}
`asegura:` {Si c no es minúscula, entonces **res** $= c$}
}

Ejemplo:

```
entrada: c = 'a', n = 3
res: 'd'
```

Ejercicio 4

`problema cifrar` (s: String, n: $\mathbb{Z}$) : String {
`requiere:` {True}
`asegura:` {La longitud de **res** es igual a la longitud de s}
`asegura:` {Si **s[i]** es minúscula, entonces **res[i]** es la letra correspondiente a desplazar n veces hacia adelante en el alfabeto a **s[i]** (para cualquier $i \in \mathbb{Z}$ que cumpla $0 \leq i < |s|$)}
`asegura:` {Si **s[i]** no es minúscula, entonces **res[i]** = **s[i]** (para cualquier $i \in \mathbb{Z}$ que cumpla $0 \leq i < |s|$)}
}

Ejemplo:

```
entrada: s = "computacion", n = 3
res: "frpsxwdflrq"
```

Ejercicio 5

`problema descifrar` (s: String, n: $\mathbb{Z}$) : String {
`requiere:` {True}
`asegura:` {La longitud de **res** es igual a la longitud de s}
`asegura:` {Si **s[i]** es minúscula, entonces **res[i]** es la letra correspondiente a desplazar n veces más atrás en el alfabeto a **s[i]** (para cualquier $i \in \mathbb{Z}$ que cumpla $0 \leq i < |s|$)}
`asegura:` {Si **s[i]** no es minúscula, entonces **res[i]** = **s[i]** (para cualquier $i \in \mathbb{Z}$ que cumpla $0 \leq i < |s|$)}
}

Ejemplo:

```
entrada: "frpsxwdflrq", n = 3
res: "computacion"
```

Ejercicio 6

`problema cifrarLista` (ls: $seq\langle$String$\rangle$) : $seq\langle$String$\rangle$ {
`requiere:` {True}
`asegura:` {La longitud de **res** es igual a la longitud de ls}
`asegura:` {Para todo $i \in \mathbb{Z}, 0 \leq i < |ls|$, **res**$[i] = cifrar(ls[i], i)$}
}

Ejemplo:

```
entrada: ls = ["compu", "labo", "intro"]
res: ["compu", "mbcp", "kpvtq"]
```

Ejercicio 7

`problema frecuencia` (s: String) : $seq\langle\mathbb{R}\rangle$ {
`requiere:` {True}
`asegura:` {La longitud de **res** es 26}
`asegura:` {Cada elemento de **res** tiene un valor entre 0 y 100, ambos inclusive}
`asegura:` {Si no hay letras minúsculas en s, todas las posiciones de **res** tienen el valor 0.}
`asegura:` {En caso contrario, la primera posición de **res** contiene el porcentaje de "a"s (respecto de la cantidad total de letras minúsculas en s), la segunda el porcentaje de "b"s (respecto de la cantidad total de letras minúsculas en s), y así sucesivamente hasta la última posición de **res** que contiene el valor porcentual de letras "z" (respecto de la cantidad total de letras minúsculas en s) que hay en s}
}

Ejemplo:

```
entrada: s = "taller"
res: [16.666668,0.0,0.0,0.0,16.666668,0.0,0.0,0.0,0.0,0.0,0.0,33.333336,
      0.0,0.0,0.0,0.0,0.0,0.0,16.666668,0.0,16.666668,0.0,0.0,0.0,0.0,0.0,0.0,0.0]
```

Ejercicio 8

`problema cifradoMasFrecuente` (s: String, n: $\mathbb{Z}$) : (Char $\times$ $\mathbb{R}$) {

`requiere:` {s contiene al menos una letra minúscula}
`asegura:` {Considerando el resultado de cifrar s con desplazamiento n, **res** contiene como primer componente alguna de las letras minúsculas con mayor frecuencia, y como segundo componente la frecuencia de dicha letra.}
}

Ejemplo:

```
entrada: "taller", n = 3
res: ('o', 33.333336)
```

Ejercicio 9

`problema esDescifrado` (s1: String, s2:String) : Bool {
`requiere:` {True}
`asegura:` {**res** = *true* si y sólo si $s2 = cifrar(s1, n)$, para algún n}
}

Ejemplo:

```
entrada: s1 = "taller", s2 = "compu"
res: False
```

Ejercicio 10

`problema todosLosDescifrados` (ls: $seq\langle$String$\rangle$) : $seq\langle$(String $\times$ String)$\rangle$ {
`requiere:` {No hay elementos repetidos en ls}
`asegura:` {No hay elementos repetidos en **res**}
`asegura:` {**res** contiene todos los pares $(s1, s2)$ tales que $s1 = descifrado(s2, n)$ donde n es un número cuyo módulo 26 es distinto de 0, y tanto *s1* como *s2* pertenecen a ls}
`asegura:` {**res** solamente contiene pares $(s1, s2)$ tales que $s1 = descifrado(s2, n)$ donde n es un número cuyo módulo 26 es distinto de 0, y tanto *s1* como *s2* pertenecen a ls}
}

Ejemplo:

```
entrada: ls = ["compu", "frpsx", "mywza"]
res: [("compu", "frpsx"), ("frpsx", "compu")]
```

Parte II: Código Vigenere

En este cifrado, una letra minúscula se sustituye por otra letra minúscula, pero desplazada un cierto número de posiciones. Este número de desplazamiento se determina mediante una *clave*, que es una palabra o frase repetida *tantas veces como sea necesario para cifrar el mensaje completo*. A diferencia de otros cifrados, como el Cifrado César, el cifrado Vigenere utiliza diferentes desplazamientos de acuerdo con las letras de la clave, lo que lo hace más resistente al análisis criptográfico. Para esta parte del trabajo práctico, nuevamente sólo vamos a considerar cifrar las letras minúsculas de los mensajes. Los espacio y otras letras se mantendrán en el mensaje, pero sin cifrarlas. Por ejemplo, para cifrar el mensaje "introduccion a la programacion" con la clave "compu", primero debe expandirse la clave a la longitud del mensaje, y luego desplazar cada letra del mensaje original por la letra de la clave correspondiente:

```
mensaje original: introduccion a la programacion
           clave: compu

 clave expandida: compucompucompucompucompucompu

 mensaje cifrado: kbfgifiorcqb p no elqudpgcqudh
```

En relación al cifrado Vigenere, se pide implementar en Haskell los siguientes ejercicios:

Ejercicio 11

`problema expandirClave` (clave: String, n: $\mathbb{Z}$) : String {
`requiere:` {$n > 0$ y $|clave| > 0$}

`requiere:` {Todos los caracteres de *clave* son letras minúsculas}
`asegura:` {La longitud de **res** es igual a n}
`asegura:` {Para cualquier $i \in \mathbb{Z}$ tal que $0 \leq i < |\textbf{res}|$, **res**[i] = *clave*[i mod $|clave|$] }
}

Ejemplo:

```
entrada: clave = "compu", n = 8
res: "compucom"
```

Ejercicio 12

`problema cifrarVigenere` (s: String, clave: String) : String {
`requiere:` {Todos los caracteres de *clave* son letras minúsculas}
`requiere:` {$|clave| > 0$}
`asegura:` {La longitud de **res** es igual a la longitud de s}
`asegura:` {Para cualquier $i \in \mathbb{Z}$ tal que $0 \leq i < |\textbf{res}|$, **res**[i] = $desplazar(s[i], n)$, donde n es igual al número que representa la letra minúscula en la posición i de la clave expandida}
}

Ejemplo:

```
entrada: s = "computacion", clave = "ip"
res: "kdueciirqdv"
```

Ejercicio 13

`problema descifrarVigenere` (s: String, clave: String) : String {
`requiere:` {Todos los caracteres de *clave* son letras minúsculas}
`asegura:` {La longitud de **res** es igual a la longitud de s}
`asegura:` {Para cualquier i tal que $0 \leq i < |\textbf{res}|$, **res**[i] = $desplazar(s[i], -n)$, donde n es igual al número que representa la letra minúscula $expandirClave[i]$}
}

Ejemplo:

```
entrada: s = "kdueciirqdv", clave = "ip"
res: "computacion"
```

Ejercicio 14

`problema peorCifrado` (s: String, claves: $seq\langle String \rangle$) : String {
`requiere:` {La secuencia *claves* tiene al menos un elemento}
`requiere:` {Todos los elementos de *claves* tienen longitud mayor a 0, y todos sus caracteres son letras minúsculas}
`asegura:` {**res** pertenece a la secuencia *claves*}
`asegura:` {De todos los elementos de *claves*, **res** es alguno de los que deja el mensaje s a menor *distancia* de su cifrado Vigenere}
}

La *distancia* entre dos secuencias $s1$ y $s2$ de igual longitud se calcula como

$$\sum_{i=0}^{|s1|-1} |letraANatural(s1[i]) - letraANatural(s2[i])|$$

Ejemplo:

```
entrada: s = "computacion", claves = ["ip", "asdef", "ksy"]
res: "asdef"
```

Ejercicio 15

`problema combinacionesVigenere` (msjs: $seq\langle$String$\rangle$, claves: $seq\langle$String$\rangle$, cifrado: String) : $seq\langle$(String $\times$ String)$\rangle$ {

`requiere:` {Las longitudes de *msjs* y *claves* son iguales}
`requiere:` {Todos los elementos de *claves* tienen longitud mayor a 0, y todos sus caracteres son letras minúsculas}
`asegura:` {La longitud de **res** es igual a la cantidad de todas las posibles combinaciones de *msjs* y *claves* que cifrados son iguales al parámetro *cifrado*}
`asegura:` {**res** contiene todas las posibles combinaciones de *msjs* y *claves* que cifrados son iguales al parámetro *cifrado*}
}

Ejemplo:

```
entrada: msjs = ["hola", "mundo"], claves = ["a", "b"], cifrado = "ipmb"
res: [("hola", "b")]
```

Aclaraciones

- En el template Solucion.hs que descarguen, verán que la línea 2 contiene "import Data.Char". Esto es para habilitar el uso de las siguientes funciones que deberán utilizar:

  `ord :: Char -> Int`.

  Dado un caracter, devuelve un entero que es la representación numérica del caracter en ASCII. Por ejemplo: `ord 'a'` devuelve `97`.

  `chr :: Int -> Char`.

  Dado un entero, devuelve su representación entera en ASCII. Por ejemplo: `chr 97` devuelve `'a'`.

  Nota: en el código ASCII las letras tienen códigos continuos, es decir `chr(ord('a') + 1) = 'b'` y así sucesivamente. Más info en `https://es.wikipedia.org/wiki/ASCII`.