Estructuras Discretas
Práctica 2

Fecha de Entrega: 13 de septiembre de 2019

1 Ejercicios

Para esta práctica, programarás algunas funciones en Haskell:

- **Función disyunción**. La función debe recibir los parámetros p y q. La función debe devolver la conjunción de p y q. No debes usar ningún operador lógico de haskell.

```
conjuncion :: Bool -> Bool -> Bool
```

- **Función conjunción**. La función debe recibir los parámetros p y q. La función debe devolver la disyunción de p y q. No debes usar ningún operador lógico de haskell.

```
disyuncion :: Bool -> Bool -> Bool
```

- **Función implicación**. La función debe recibir los parámetros p y q. La función debe devolver la implicación de p y q. No debes usar ningún operador lógico de haskell.

```
implicacion :: Bool -> Bool -> Bool
```

- **Función bicondicional**. La función debe recibir los parámetros p y q. La función debe devolver la doble equivalencia de p y q. No debes usar ningún operador lógico de haskell.

```
dobleImplica :: Bool  -> Bool -> Bool
```

2 Listas

Una estructura de datos son conjuntos de elementos, que guardan valores con un esquema específico. Las listas son la estructura de datos más simple que vamos a trabajar en Haskell.

Listas por comprensión.
Como ya hemos visto en clase, las listas por comprensión son una poderosa herramienta de Haskell, la cual nos permiten utilizar la notación por comprensión que usualmente empleamos en matemáticas, para definir conjuntos. Por ejemplo, la representación en Haskell del conjunto $C = \{x \in \mathbf{N} | x \in [2, 11]\}$ es :

```
[x | x <- [2 .. 11]] = [2,3,4,5,6,7,8,9,10,11]
```

- Una función recursiva que recibe una lista de cualquier tipo y calcula la longitud de ésta.

  ```
  longitud:: [a] -> Int
  ```

- Una función recursiva que recibe una lista de números y devuelve el resultado de la suma de todos ellos.

  ```
  sumaNumeros:: Num a => [a] -> a
  ```

- **Maximo en una lista**. La función debe recibir como parámetro una lista xs de valores númericos. La función debe devolver el máximo de la lista xs.

  ```
  maximo :: Num  a => [a] -> a
  ```

- **Lugar del elemento**. La función debe recibir como parámetros, un índice i y una lista xs. La función debe devolver el elemento que esta en el lugar i de la lista xs. Si el lugar que buscamos es más grande que el tamaño de la lista devolvemos el mensaje: "El elemento que buscas no está en la lista".

  ```
  indiceDe :: Int -> [a] -> a
  ```

- **Insertar elemento**. La función debe recibir como parámetros, un elemento x, una lista xs y un booleno b. La función debe devolver la lista xs más el elemento x. Si $b = True$ el elemento se inserta al principio de la lista, si $b = False$, se inserta al final. Sólo puedes usar una vez el operador de concatenación de listas.

  ```
  insertarElemento :: a -> [a] -> Bool -> [a]
  ```

- **Palindromo**. Una función recursiva que decide si una lista es un palíndromo (también puede recibir una cadena).

  ```
  esPalindromo:: [a] -> Bool
  ```

  Por ejemplo:

  ```
  esPalindromo [1,2,3,4,4,3,2,1] = True
  esPalindromo "vacaciones" = False
  ```

- **Reversa**. La reversa de una lista es la misma lista escrita en orden inverso. Escribe una función recursiva que calcula la reversa de una lista (también puede recibir una cadena).

  ```
  reversa:: [a] -> [a]
  ```

  Por ejemplo:

  ```
  reversa [8,7,6,5,4] = [4,5,6,7,8]
          reversa "magia" = "aigam"
  ```

- **Convertir en conjunto**. Una función recursiva que recibe una lista y elimina las repeticiones de elementos, es decir, convierte la lista en un conjunto. HINT: Utiliza listas por comprensión.

  ```
  aConjunto:: [a] -> [a]
  ```

- **Union de conjuntos**. Una función recursiva que calcula la unión de dos listas (Prohibido usar listas por comprensión).

  ```
  union:: [a] -> [a] -> [a]
  ```

- **Intersección de conjuntos**. Una función recursiva que calcula la unión de dos listas (Prohibido usar listas por comprensión).

  ```
  union:: [a] -> [a] -> [a]
  ```

- **Producto cartesiano**. Una función recursiva que recibe dos listas y devuelve el producto cruz de ambas listas.

  ```
  productoCruz:: [a] -> [a] -> [a]
  ```

- Una función recursiva que calcula la diferencia simétrica de dos listas.

  ```
  diferenciaSimetrica:: [a] -> [a] -> [a]
  ```

- PUNTO EXTRA: Escribe una función que calcule el conjunto de divisores de un número entero. HINT: Utiliza listas por comprensión.

  ```
  divisores:: Int -> [Int]
  ```

  Ejemplo:

  ```
  divisores 1024 = [1,2,4,8,16,32,64,128,256,512]
  ```

- PUNTO EXTRA: Sea C un conjunto, *el conjunto potencia* de C se define como el conjunto de pares ordenados (a, b) tales que $a \in C$ y $b \in C$. Define en Haskell una función recursiva que recibe una lista l de elementos (puede incluir elementos repetidos) y calcula el conjunto potencia (sin repetir pares ordenados) del conjunto que representa la lista l. HINT: Utiliza la función aConjunto.

  ```
  conjuntoPotencia:: [a] -> [a]
  ```

  Ejemplo:

  ```
  conjuntoPotencia [1,2,1,3] =
                   [(1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1),
                    (3,2), (3,3)]
  ```

3 Entrega

Deberás enviar un archivo con las funciones que programes al correo **luismanuel@ciencias.unam.mx** a más tardar el **13 de septiembre de 2019** antes de las **11:59:59**. El nombre del archivo debe ser **Practica2.hs**, además de las funciones programadas, el archivo debe llevar tu nombre completo empezando por apellidos. El asunto del correo debe ser **ED20201[Práctica 1]**.