Examen de Septiembre 2016

Nombre:_____**Especialidad**_____**Curso**_____.

Haskell

La codificación Huffman permite representar textos de forma comprimida. Se trata de un código de longitud variable: cada carácter se representa por una secuencia de bits cuya longitud es inversamente proporcional a su frecuencia de aparición. Es decir, los caracteres muy frecuentes son representados por secuencias de pocos bits; mientras que los caracteres menos frecuentes se representan por secuencias de bits más largas. Vamos a desarrollar paso a paso una implementación de la codificación Huffman de un mensaje representado por una cadena (`String`).

Ejercicio 1. (0.25 puntos) El primer paso para construir un código de Huffman consiste en calcular la frecuencia de aparición o *peso* de los caracteres del mensaje. Define la función

```
weights :: Ord a => [a] -> D.Dictionary a Int
```

que dada una lista devuelva un diccionario con la frecuencia de aparición de cada elemento de la lista. Por ejemplo:

```
> weights "abracadabra"
AVLDictionary('a'->5,'b'->2,'c'->1,'d'->1,'r'->2)

> weights [1,2,9,2,0,1,6,1,5,5,8]
AVLDictionary(0->1,1->3,2->2,5->2,6->1,8->1,9->1)

> weights ""
AVLDictionary()
```

Ejercicio 2. Una vez que se ha obtenido la frecuencia de aparición de cada carácter, el siguiente paso consiste en construir un **árbol de Huffman**. El proceso se describe paso a paso en las transparencias 3-9 (`Huffman.pdf`).

Representaremos los árboles de Huffman mediante el tipo `WLeafTree` (árboles binarios con peso e información sólo en las hojas):

```
data WLeafTree a = WLeaf a Int | WNode (WLeafTree a) (WLeafTree a) Int
```

La construcción de árboles de Huffman requiere almacenar una colección de árboles. Como colección utilizaremos una cola de prioridad, donde los árboles aparecerán ordenados por peso (primero los más ligeros).

Apartado 2.a (0.25 puntos) Para construir el árbol de Huffman primero debemos obtener la colección inicial de árboles (transparencia 5). Define la función

```
huffmanLeaves :: String -> PQ.PQueue (WLeafTree Char)
```

que dada una cadena devuelve una cola de prioridad con árboles hoja. Por ejemplo:

```
> huffmanLeaves "abracadabra"
WBLeftistHeapPriorityQueue(WLeaf 'c' 1,WLeaf 'd' 1,WLeaf 'b' 2, WLeaf 'r' 2,WLeaf 'a' 5)
```

Apartado 2.b (1 punto) Define la función

```
huffmanTree :: String -> WLeafTree Char
```

que dada una cadena **con al menos 2 símbolos distintos** devuelve el árbol de Huffman correspondiente. Si la cadena no tiene al menos 2 caracteres distintos, la función debe señalar el error. La función debe implementar el algoritmo de reducción por mezcla descrito en las transparencias 4-9. Al comprobar tu solución ten en cuenta que, como se explica en las transparencias 10 y 11, el árbol de Huffman no es único. Puedes utilizar la función `printWLeafTree` para dibujar el árbol de Huffman obtenido. Por ejemplo:

```
> printWLeafTree (huffmanTree "aaa")
*** Exception: huffmanTree: the string must have at least two different symbols

> printWLeafTree (huffmanTree "abracadabra")
      11_______
      /        \
('a',5)        6_______________
              /                \
        ('r',2)          _______4
                        /       \
                       2        ('b',2)
                      / \
                ('c',1) ('d',1)
```

Ejercicio 3. Una vez construido el árbol de Huffman, se puede obtener el código de Huffman de cada carácter descendiendo por las ramas del árbol. El proceso se describe en las transparencias 12-14. Representaremos el código de Huffman mediante un diccionario de caracteres a listas de bits (enteros).

Apartado 3.a (0.5 puntos) Define la función

```
joinDics :: Ord a => D.Dictionary a b -> D.Dictionary a b -> D.Dictionary a b
```

que dados dos diccionarios **disjuntos** (no comparten ninguna clave) devuelve su unión. Por ejemplo:

```
> joinDics (D.insert 'a' 1 $ D.insert 'c' 3 $ D.empty) D.empty
AVLDictionary('a'->1,'c'->3)

> joinDics (D.insert 'a' 1 $ D.insert 'c' 3 $ D.empty)
    (D.insert 'b' 2 $ D.insert 'd' 4 $ D.insert 'e' 5 $ D.empty)
AVLDictionary('a'->1,'b'->2,'c'->3,'d'->4,'e'->5)
```

Apartado 3.b (0.75 puntos) Define la función

```
prefixWith :: Ord a => b -> D.Dictionary a [b] -> D.Dictionary a [b]
```

que dados un valor x y un diccionario cuyos valores son listas devuelva un nuevo diccionario que se obtiene prefijando cada lista del diccionario con x. Por ejemplo:

```
> prefixWith 0 (D.insert 'a' [0,0,1] $ D.insert 'b' [1,0,0] $ D.empty)
AVLDictionary('a'->[0,0,0,1],'b'->[0,1,0,0])
```

```
> prefixWith 'h' (D.insert 1 "asta" $ D.insert 2 "echo" $ D.empty)
AVLDictionary(1->"hasta",2->"hecho")
```

Apartado 3.c (1 punto) Utilizando las funciones anteriores, define la función

```
huffmanCode :: WLeafTree Char -> D.Dictionary Char [Integer]
```

que dado un árbol de Huffman devuelve el código correspondiente. Por ejemplo:

```
> huffmanCode (huffmanTree "abracadabra")
AVLDictionary('a'->[0],'b'->[1,1,1],'c'->[1,1,0,0],'d'->[1,1,0,1],'r'->[1,0])
```

Sugerencia: utiliza recursión sobre el árbol, si bajas por la izquierda prefija con un cero, si bajas por la derecha prefija con un 1.

SOLO para estudiantes sin evaluación continua

Ejercicio 4. (0.25 puntos) Una vez construido el código de Huffman es muy simple codificar el mensaje, basta reemplazar cada carácter por su código, como se ve en la transparencia 15.

Define la función

```
encode :: String -> D.Dictionary Char [Integer] -> [Integer]
```

que dados una cadena y un diccionario con el código de Huffman devuelve el mensaje codificado. Por ejemplo:

```
> encode "abracadabra" (huffmanCode (huffmanTree "abracadabra"))
[0,1,1,1,1,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0]

> encode "abracadabra" (huffmanCode (huffmanTree2Q "abracadabra"))
[0,1,1,0,1,1,1,0,1,0,0,0,1,0,1,0,1,1,0,1,1,1,0]
```

Ejercicio 5. Después de recibir el mensaje codificado, el receptor debe decodificarlo. El proceso de decodificación utiliza el árbol de Huffman, tal como se explica en las transparencias 16-25.

Apartado 5.a (0.5 puntos) Define la función

```
takeSymbol :: [Integer] -> WLeafTree Char -> (Char, [Integer])
```

que dados un mensaje codificado y un árbol de Huffman, decodifica el primer carácter del mensaje y devuelve un par con el carácter decodificado y el resto del mensaje codificado. Por ejemplo:

```
> takeSymbol [0,1,1,1,1,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0]
    (huffmanTree "abracadabra")
('a',[1,1,1,1,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0])

> takeSymbol [1,1,1,1,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0]
    (huffmanTree "abracadabra")
('b',[1,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0])

> takeSymbol [1,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0]
    (huffmanTree "abracadabra")
('r',[0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0])
```

Apartado 5.b (0.5 puntos) Define la función

```
decode :: [Integer] -> WLeafTree Char -> String
```

que dados un mensaje codificado y el árbol Huffman correspondiente devuelva el mensaje original. Por ejemplo:

```
> decode [0,1,1,1,1,0,0,1,1,0,0,0,1,1,0,1,0,1,1,1,1,0,0] (huffmanTree "abracadabra")
"abracadabra"
```

Examen de Septiembre 2016

Nombre:_____**Especialidad**_____**Curso**_____.

Java

Ahora resolveremos el mismo problema en Java (Ver introducción en Haskell).

Ejercicio 1. (0.25 puntos) El primer paso para construir un código de Huffman consiste en calcular la frecuencia de aparición o *peso* de los caracteres del mensaje. Define el método estático

```
public static Dictionary<Character, Integer> weights(String s)
```

que dado un mensaje representado como una `String` devuelva un diccionario que asocie cada carácter del mensaje con su número de apariciones.

Ejercicio 2. Una vez que se ha obtenido la frecuencia de aparición de cada carácter, el siguiente paso consiste en construir un **árbol de Huffman**. El proceso se describe paso a paso en las transparencias 3-9 (`Huffman.pdf`).

Representaremos los árboles de Huffman mediante la clase `WLeafTree` (árboles binarios con peso e información sólo en las hojas). La implementación está disponible en el fichero `WLeafTree.java`.

La construcción de árboles de Huffman requiere almacenar una colección de árboles. Como colección utilizaremos una cola de prioridad, donde los árboles aparecerán ordenados por peso (primero los más ligeros).

Apartado 2.a (0.25 puntos) Para construir el árbol de Huffman primero debemos obtener la colección inicial de árboles (transparencia 5). Define el método estático

```
public static PriorityQueue<WLeafTree<Character>> huffmanLeaves(String s)
```

que dada una cadena devuelve una cola de prioridad con árboles hoja.

Apartado 2.b (1 punto) Define el método estático

`public static WLeafTree<Character> huffmanTree(String s)` que dada una cadena **con al menos 2 símbolos distintos** devuelve el árbol de Huffman correspondiente. Si la cadena no tiene al menos 2 caracteres distintos, el método debe elevar la excepción `HuffmanException`. El métoo debe implementar el algoritmo de reducción por mezcla descrito en las transparencias 4-9. Al comprobar tu solución ten en cuenta que, como se explica en las transparencias 10 y 11, el árbol de Huffman no es único.

Ejercicio 3. Una vez construido el árbol de Huffman, se puede obtener el código de Huffman de cada carácter descendiendo por las ramas del árbol. El proceso se describe en las transparencias 12-14. Representaremos el código de Huffman mediante un diccionario de caracteres a listas de bits (enteros).

Apartado 3.a (0.5 puntos) Define el método estático

```
public static Dictionary<Character, List<Integer>>
    joinDics( Dictionary<Character, List<Integer>> d1
            , Dictionary<Character, List<Integer>> d2)
```

que dados dos diccionarios **disjuntos** (no comparten ninguna clave) devuelve su unión.

Apartado 3.b (0.75 puntos) Define el método estático

```
public static Dictionary<Character, List<Integer>>
    prefixWith(int i, Dictionary<Character, List<Integer>> d)
```

que dados un valor `i` y un diccionario cuyos valores son listas devuelva un **nuevo** diccionario que se obtiene prefijando cada lista del diccionario con `i`.

Apartado 3.c (1 punto) Utilizando los métodos anteriores, define el método estático

```
public static Dictionary<Character, List<Integer>>
    huffmanCode(WLeafTree<Character> ht)
```

que dado un árbol de Huffman devuelve el código correspondiente.

Sugerencia: utiliza recursión sobre el árbol, si bajas por la izquierda prefija con un cero, si bajas por la derecha prefija con un 1.

SOLO para estudiantes sin evaluación contínua

Ejercicio 4. (0.25 puntos) Una vez construido el código de Huffman es muy simple codificar el mensaje, basta reemplazar cada carácter por su código, como se ve en la transparencia 15.

Define el método estático

```
public static List<Integer> encode(String s, Dictionary<Character, List<Integer>> hc)
```

que dados una cadena y un diccionario con el código de Huffman devuelve el mensaje codificado.

Ejercicio 5. (1 punto) Después de recibir el mensaje codificado, el receptor debe decodificarlo. El proceso de decodificación utiliza el árbol de Huffman, tal como se explica en las transparencias 16-25.

Define el método estático

```
public static String decode(List<Integer> bits, WLeafTree<Character> ht)
```

que dados un mensaje codificado y el árbol Huffman correspondiente devuelva el mensaje original