[redacted]

Examen Parcial 2

1. Un **multidiccionario** es un diccionario en el que se pueden asociar múltiples valores (no solo uno) a una clave, es decir es una colección de pares de la forma (clave, valores), donde cada clave puede estar asociada a un único conjunto de valores.
Definimos el TAD MultiDic con las siguientes operaciones:

tad MultiDic (K : Ordered Set, V : Set) **where**
  import Bool, Conjunto
    empty : MultiDic K V
    insert : $K \to V \to$ MultiDic K V $\to$ MultiDic K V -- agrega un valor al conjunto de valores de una clave
    erase : $K \to V \to$ MultiDic K V $\to$ MultiDic K V -- elimina un par (clave, valor)
    lookup : $K \to$ MultiDic K V $\to$ Conjunto V -- devuelve el conjunto de valores asociado a una clave
    isValue : $K \to V \to$ MultiDic K V $\to$ Bool -- determina si un valor está asociado a una clave
    toMap : MultiDic K V $\to$ Conjunto (K, V) -- construye el conjunto de (clave, valor) que forman -- un multidiccionario

a) Dar una especificación algebraica para el TAD MultiDic.

b) Para implementar las operaciones del TAD MultiDic de manera eficiente en Haskell se utilizan árboles binarios de búsqueda, representados con el siguiente tipo de datos:

data MultiDic k v = E | N (MultiDic k v) (k, Tree v) (MultiDic k v)

donde los valores asociados a las claves se almacenan en árboles binarios que contienen información sobre la cantidad de nodos:

data Tree a = Empty | Leaf a | Node Int (Tree a) (Tree a)

Usando esta representación definir en Haskell de manera eficiente las funciones:

i. isValue :: (Ord k, Eq v) $\Rightarrow k \to v \to$ MultiDic k v $\to$ Bool, del TAD.
ii. toMap :: Ord $k \Rightarrow$ MultiDic k v $\to$ Tree (k, Int, v), similar a la función toMap del TAD, pero que agrega a cada par de la forma (clave, valor) la posición del valor en la secuencia representada por el árbol.
Por ejemplo,

$$\text{toMap}\langle(1, \langle a, f, g\rangle), (2, \langle m, a\rangle)\rangle = \langle(1, 0, a), (1, 1, f), (1, 2, g), (2, 0, m), (2, 1, a)\rangle$$

c) Usando esta representación, se define la función erase como:

erase :: (Ord k, Eq v) $\Rightarrow k \to v \to$ MultiDic k v $\to$ MultiDic k v
erase k v (N l (k', vs) r) | $k \equiv k'$ = N l (k, elim v vs) r
    | $k < k'$ = N (erase k v l) (k', vs) r
    | $k > k'$ = N l (k', vs) (erase k v r)
erase k v E = E

donde la función elim elimina un elemento de un árbol.
Probar por inducción estructural sobre t la siguiente propiedad:
Si t es un bst, erase k v t es un bst, para cualesquiera k y v.

2. Decimos que una secuencia es una progresión aritmética si la diferencia entre cualquier elemento (a excepción del primero) y su anterior es un valor constante. Por ejemplo, $\langle 3,5,7,9\rangle$ es una progresión aritmética mientras que $\langle 3,5,6,8,9\rangle$ no lo es. Se quiere definir una función spaml : Seq Int $\rightarrow$ Int que dada una secuencia de enteros devuelve la longitud de la subsecuencia contigua más larga que es progresión aritmética.
Por ejemplo,

spaml$\langle 7,6,5,4,6,8\rangle = 4$
spaml$\langle 6,7,5,6\rangle = 2$
spaml$\langle 1,2,5,8,12,14,16\rangle =$ ~~[illegible]~~ $= 3$

Dada una secuencia s de longitud n mayor a 2, una manera de calcular la longitud de la subsecuencia más larga que es progresión aritmética consiste en calcular para cada i mayor o igual a 1 y menor o igual que n el sufijo más largo que es progresión aritmética para la secuencia formada por los primeros i elementos de s.
Completar la siguiente definición de la función spaml, que utiliza las funciones del TAD secuencia, dando definiciones apropiadas para f, g y h, de manera que spaml tenga profundidad en $O(\lg n)$, donde n es el largo de la secuencia.

```
spaml : Seq Int → Int
spaml s | n ⩽ 2     = n
        | otherwise = spaml_aux s
  where n = length s

spaml_aux : Seq Int → Int
spaml_aux s =
  let
    n            = length s
    s_dif        = tabulate (λi → nth (i + 1) s − nth i s) (n − 1)
    s_info       = map f s_dif
    (s_red, r)   = scan g (nth 0 s_info) (drop s_info 1)
    s_res        = map h (append s_red (singleton r))
    f = ...
    g = ...
    h = ...
  in 1 + reduce max 0 s_res
```