Facultad de Ciencias Exactas, Ingeniería y Agrimensura
Escuela de Ciencias Exactas y Naturales
Departamento de Ciencia de la Computación
Estructura de Datos y Algoritmos II

Nombre y Apellido: Legajo:

Examen Parcial 2

1. El TAD Dict es una colección de pares (clave, valor), donde cada clave está asociada a un único valor. Definimos el TAD Dict con las siguientes operaciones:

```
tad Dict (K : Ordered Set, V : Set) where
  import Bool, Maybe
    empty : Dict K V
    insert : K → V → Dict K V → Dict K V   -- agrega un par (clave, valor) al diccionario (*)
    erase  : K → Dict K V → Dict K V       -- elimina del diccionario la información de una clave
    isKey  : K → Dict K V → Bool           -- determina si una clave pertenece al diccionario
    lookup : K → Dict K V → Maybe V        -- devuelve el valor asociado a una clave
```

(*) Si la clave ya existe en el diccionario, se sobreescribe el valor antiguo.

a) Dar una especificación algebraica para el TAD Dict.

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

```
data BST k v = E | N (BST k v) (k, v) (BST k v)
```

Usando esta representación, se definen las funciones lookup e insert de la siguiente manera:

```
lookup k E = Nothing
lookup k (N l (q, v) r) | k ≡ q     = Just v
                        | k < q     = lookup k l
                        | otherwise = lookup k r


insert k v E = N E (k, v) E
insert k v (N l (q, w) r) | k ≡ q     = N l (k, v) r
                          | k < q     = N (insert k v l) (q, w) r
                          | otherwise = N l (q, w) (insert k v r)
```

Probar por inducción estructural sobre t la siguiente propiedad:

```
lookup k (insert k v t) = Just v
```

2. Se representan secuencias mediante árboles binarios dados por el siguiente tipo de datos:

```
data Tree a = E | N Int (Tree a) a (Tree a)
```

donde se guarda la longitud de la secuencia en los nodos y el recorrido *inorder* del árbol da el orden de los elementos de la secuencia.
Definir en Haskell de manera eficiente la función filterPrefix:: $(a \to \text{Bool}) \to \text{Tree } a \to \text{Tree } a$, que dado un predicado p y una secuencia s, computa el prefijo más largo de s para el cual todos sus elementos satisfacen p.
Por ejemplo,

```
filterPrefix odd ⟨6, 6, 8, 1, 4, 5⟩ = ⟨⟩
filterPrefix even⟨6, 6, 8, 1, 4, 5⟩ = ⟨6, 6, 8⟩
```

Definir filterPrefix con profundidad en $O(h)$, donde h es la altura del árbol y p es de costo constante.

3. Usando las funciones del Tad secuencia, incluyendo necesariamente a la función *scan*, definir una función longestStreak : Float → Seq Float → Int, que dados

- un valor numérico *val*, que representa una temperatura
- una secuencia de valores numéricos s, que representa la temperatura máxima diaria a lo largo del tiempo

calcule la racha más larga de días en s donde la temperatura superó los *val* grados.
Por ejemplo,

longestStreak $30\langle 20, 21, 27, 30, 24\rangle = 0$
longestStreak $30\langle 28, 31, 32, 29, 31, 31, 33, 29\rangle = 3$
longestStreak $30\langle 28, 31, 29, 31, 29\rangle = 1$

Definir longestStreak con profundidad en $O(\lg n)$, donde n es el largo de la secuencia.