Trabajo Práctico 1

Un **diccionario** es un conjunto de pares (clave, valor) donde cada clave tiene asociado un único valor. Las claves deben ser comparables entre sí (es decir, debe existir un orden entre ellas). Para poder ser usado, un diccionario debe contar con operaciones para crear un diccionario nuevo (vacío), insertar o eliminar un elemento y buscar el valor asociado a una clave. Utilizaremos la siguiente clase para implementar diccionarios en Haskell:

```
class Diccionario t where
  vacia     :: Ord k ⇒ t k v
  insertar  :: Ord k ⇒ (k, v) → t k v → t k v
  eliminar  :: Ord k ⇒ k → t k v → t k v
  buscar    :: Ord k ⇒ k → t k v → Maybe v
```

1. Los árboles BST 3-2 pueden utilizarse para lograr una implementación eficiente de las operaciones de esta clase, donde la operación *buscar* tiene un costo logarítmico. Un árbol BST 3-2 es un árbol binario de búsqueda, en el cual cada nodo satisface la siguiente propiedad de balanceo:

   i) el tamaño del subárbol derecho es menor o igual al triple del tamaño del subárbol izquierdo.
   ii) el tamaño del subárbol izquierdo es menor o igual al triple del tamaño del subárbol derecho.
   iii) si uno de los subárboles es vacío, el tamaño del otro puede ser 1.

   Utilizando el siguiente tipo de datos para representar árboles BST 3-2 (con tuplas como valores de los nodos):

```
data BTree32 k a = Nil   -- árbol vacío
                 | Node
                     (BTree32 a)      -- subárbol izquierdo
                     Int              -- tamaño del árbol
                     (k, a)           -- elemento del nodo
                     (BTree32 k a)    -- subárbol derecho
```

   definir las funciones que se describen a continuación:

   a) *size* :: $BTree32\ k\ a \to Int$, que calcula el tamaño de un árbol. Esta función debe tener costo constante.
   b) *lookup* :: $Ord\ k \Rightarrow k \to BTree32\ k\ a \to Maybe\ a$, que devuelve el elemento asociado a un valor en un árbol BST 3-2.
   c) Al agregar o eliminar un elemento a un árbol BST 3-2 debe asegurarse que la propiedad de balanceo se cumple. Para ello se desea definir una función

      $$balance :: BTree32\ k\ a \to (k, a) \to BTree32\ k\ a \to BTree32\ k\ a$$

      que dados dos árboles BST 3-2 y un elemento construya con los mismos un árbol BST 3-2.
      Para definir esta función se deben definir dos funciones

      $$singleL :: BTree32\ k\ a \to BTree32\ k\ a$$
      $$doubleL :: BTree32\ k\ a \to BTree32\ k\ a$$

      que realicen rotaciones en un árbol como se indica en la figura 1.
      Las rotaciones definidas anteriormente, trasladan algunos elementos del subárbol derecho del árbol al subárbol izquierdo. De manera similar se pueden definir otras dos funciones

      $$singleR :: BTree32\ k\ a \to BTree32\ k\ a$$
      $$doubleR :: BTree32\ k\ a \to BTree32\ k\ a$$

      que trasladen los elementos correspondientes del subárbol izquierdo al subárbol derecho.
      Utilizando estas cuatro funciones la función *balance* se define de la siguiente manera, dados dos árboles BST 3-2 l y r, y un valor x:

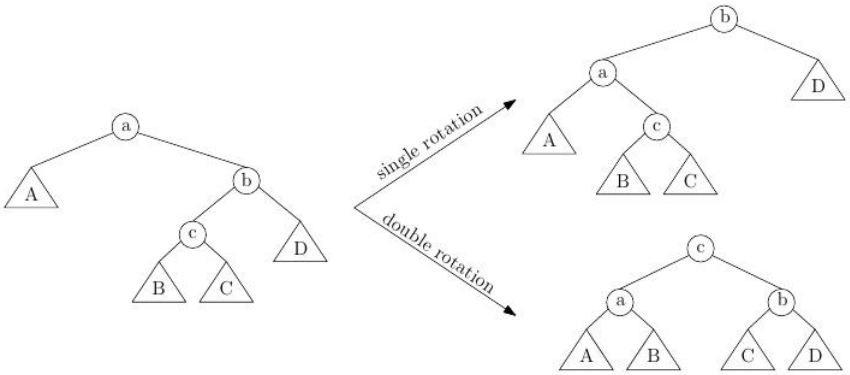

Figura 1: Rotaciones hacia la izquierda simples y dobles

- Si la suma de los tamaños de los árboles l y r es menor o igual a uno o se cumple la propiedad de balanceo, *balance* construye un árbol con l como subárbol izquierdo, x como raíz y r como subárbol derecho.
- Si la propiedad de balanceo no se cumple porque se viola el item i) entonces algunos elementos del árbol l deben pasar al árbol r, para ello se aplica una vez alguna de las rotaciones *singleL* y *doubleL*. Para determinar cuál de las dos rotaciones se debe realizar se sigue el siguiente criterio: si el tamaño del subárbol izquierdo de r es menor que el doble del tamaño del subárbol derecho de r se realiza una rotación simple (*singleL*) y sino una rotación doble (*doubleL*).
- Si la propiedad de balanceo no se cumple porque se viola el item ii), se aplica algunas de las rotaciones *singleR* y *doubleR*. Se toma un criterio similar al anterior pero inspeccionando los tamaños de los subárboles de l.

La función *balance* será de utilidad para definir las siguientes funciones.

d) *insert* :: $Ord\ k \Rightarrow (k, a) \rightarrow BTree32\ k\ a \rightarrow BTree32\ k\ a$, que agrega un elemento a un árbol BST 3-2.

e) *delRoot* :: $Ord\ k \Rightarrow BTree32\ k\ a \rightarrow BTree32\ k\ a$, que dado un árbol BST 3-2 elimina la raíz del mismo y construye un BST 3-2. Esta función deberá buscar una nueva raíz de alguno de los subárboles, dependiendo del tamaño de los mismos y dejando uno de ellos sin modificar.

f) *delete* :: $Ord\ k \Rightarrow k \rightarrow BTree32\ k\ a \rightarrow BBTree\ k\ a$, que elimina un elemento de un árbol BST 3-2.

2. Utilizar las definiciones anteriores para definir una instancia de la clase *Diccionario* para el tipo de datos *BTree32*.