Práctica 5

1. El tipo abstracto de datos Secuencias representa una colección ordenada de elementos, junto con operaciones (idealmente paralelizables) sobre éstas. Consideraremos las implementaciones de secuencias mediantes listas y árboles. La diferencia de ambas es el costo de las operaciones.
Definir las siguientes funciones, correspondientes a la interfaz de Secuencias, implementando las secuencias con árboles binarios, definidos con el siguiente tipo de datos.

$$\textbf{data}\ \mathsf{BTree}\ a = \mathsf{Empty} \mid \mathsf{Node}\ \mathsf{Int}\ (\mathsf{BTree}\ a)\ a\ (\mathsf{BTree}\ a)$$

donde se almacenan los tamaños de los árboles en los nodos. Suponer que el recorrido *inorder* del árbol da el orden de los elementos de la secuencia. Calcular el trabajo y la profundidad de cada una. Resolver la recurrencia y expresar la solución en términos del orden O.

- $\mathsf{nth} :: \mathsf{BTree}\ a \to \mathsf{Int} \to a$, calcula el n-ésimo elemento de una secuencia.
- $\mathsf{cons} :: a \to \mathsf{BTree}\ a \to \mathsf{BTree}\ a$, la cual inserta un elemento al comienzo de la secuencia.
- $\mathsf{tabulate} :: (\mathsf{Int} \to a) \to \mathsf{Int} \to \mathsf{BTree}\ a$, la cual dada una función f y un entero n devuelve una secuencia de tamaño n, donde cada elemento de la secuencia es el resultado de aplicar f al índice del elemento.
- $\mathsf{map} :: (a \to b) \to \mathsf{BTree}\ a \to \mathsf{BTree}\ b$, la cual dada una función f y una secuencia s, devuelve el resultado de aplicar f sobre cada elemento de s.
- $\mathsf{take} :: \mathsf{Int} \to \mathsf{BTree}\ a \to \mathsf{BTree}\ a$, tal que dados un entero n y una secuencia s devuelve los primeros n elementos de s.
- $\mathsf{drop} :: \mathsf{Int} \to \mathsf{BTree}\ a \to \mathsf{BTree}\ a$, tal que dados un entero n y una secuencia s devuelve la secuencia s sin los primeros n elementos.

2. El problema de calcular la máxima suma de una subsecuencia contigua de una secuencia dada s puede resolverse con un algoritmo "*Divide & Conquer*" que en cada llamada recursiva calcule: la máxima suma de una subsecuencia contigua de s, la máxima suma de un prefijo de s, la máxima suma de un sufijo de s y la suma de todos los elementos de s. Dado el siguiente tipo de datos:

$$\textbf{data}\ \mathsf{Tree}\ a = \mathsf{E} \mid \mathsf{Leaf}\ a \mid \mathsf{Join}\ (\mathsf{Tree}\ a)\ (\mathsf{Tree}\ a)$$

a) Definir una función $\mathsf{mcss} :: (\mathsf{Num}\ a, \mathsf{Ord}\ a) \Rightarrow \mathsf{Tree}\ a \to a$, que calcule la máxima suma de una subsecuencia contigua de una secuencia dada, en términos de mapreduce.

Ayuda: Dado un árbol t, mcss aplica la función *reduce* sobre el árbol que se obtiene al reemplazar cada elemento v por la 4-tupla $(\mathsf{max}\ (v,0), \mathsf{max}\ (v,0), \mathsf{max}\ (v,0), v)$.

b) Calcular el trabajo y la profundidad de mcss.

3. Dados los diferentes valores de las acciones de YPF a lo largo del tiempo, se desea saber cuál es la mejor ganancia que se puede obtener al comprar acciones un día y venderlas otro.
Definir una función $\mathsf{mejorGanancia} :: \mathsf{Tree}\ \mathsf{Int} \to \mathsf{Int}$ que calcule la mejor ganancia dada una secuencia de valores, utilizando el siguiente algoritmo:

- Armar pares de la forma $(compra, ventas)$, donde *compra* es el precio al cual se puede comprar una acción y *ventas* los distintos valores en que puede venderse.
- Para cada par de la forma $(compra, ventas)$ calcular las diferencias $venta - compra$, donde *venta* es un elemento de *ventas*.
- Tomar el número máximo de las diferencias calculadas en el paso anterior.

Definir las siguientes funciones, que implementan distintas partes del algoritmo y utilizarlas para definir mejorGanancia.

- sufijos :: Tree Int → Tree (Tree Int), tal que dado un árbol t construye otro con los sufijos de cada elemento de t. Por ejemplo,

  t = Join (Join (Leaf 10) (Leaf 15)) (Leaf 20)
  sufijos t = Join (Join (Leaf (Join (Leaf 15) (Leaf 20))) (Leaf (Leaf 20))) (Leaf E)

- conSufijos :: Tree Int → Tree (Int, Tree Int), la cual dado un árbol t reemplaza cada elemento v de t por el par (v, sufijos de v en t).
- maxT :: Tree Int → Int, la cual calcula el máximo elemento de un árbol de enteros. Definir maxT en términos de reduce.
- maxAll :: Tree (Tree Int) → Int, calcula el máximo elemento de en árbol de árboles de enteros. Definir maxAll en términos de mapreduce.

4. Dadas las siguientes definiciones:

data T a = E | N (T a) a (T a)
altura :: T a → Int
altura E = 0
altura (N l x r) = 1 + max (altura l, altura r)

a) Definir una función combinar :: T a → T a → T a, que satisfaga la siguiente especificación:

Sean t_1, t_2 :: T a:

- combinar t_1 t_2 contiene todos los elementos de t_1 y t_2 y no contiene ningún otro elemento.
- altura (combinar t_1 t_2) $\leqslant 1 +$ max (altura t_1, altura t_2)
- $W_{\mathsf{combinar}}(d_1, d_2), S_{\mathsf{combinar}}(d_1, d_2) \in O(d_1)$, donde d_1 y d_2 son las alturas de los árboles t_1 y t_2 que recibe como argumento la función combinar.

b) Definir una función filterT :: (a → Bool) → T a → T a (similar a la función filter sobre listas) que satisfaga la siguiente especificación:

Sean p :: a → Bool y t :: T a:

- filterT p t contiene todos los elementos de t que satisfacen p y no contiene ningún otro elemento.
- altura (filterT p t) $\leqslant$ altura t
- $S_{\mathsf{filterT}}(d) \in O(d^2)$, donde d es la altura del árbol que recibe como argumento filterT.

c) Definir una función quicksortT :: T Int → T Int que implemente el algoritmo quicksort sobre árboles. Utilizar en la definición las funciones definidas anteriormente.

Sea t : T Int, quicksortT t es un árbol binario de búsqueda que contiene todos los elementos de t y ningún otro elemento.

- Calcular $W_{\mathsf{quicksortT}}(n)$ en el peor caso, siendo n la cantidad de nodos del árbol que recibe como argumento la función.
- Suponiendo que quicksortT recibe un árbol balanceado, calcular el trabajo y la profundidad de la función en el peor caso, el mejor caso y suponiendo que el pivote divide a los datos en proporción 1 a 9. ¿Qué cambiaría en el último caso si la proporción es 1 a 99?

5. Considere el tipo de datos BTree del ejercicio 1.

a) Definir una función $\mathsf{splitAt} :: \mathsf{BTree}\ a \to \mathsf{Int} \to (\mathsf{BTree}\ a, \mathsf{BTree}\ a)$, tal que dado un árbol t y un entero i construya dos árboles t_1 y t_2 que contengan, t_1 los i elementos de más a la izquierda de t y t_2 los restantes.

Definir splitAt de manera que satisfaga la siguiente especificación:

Sean t_1 y t_2 tales que, $\mathsf{splitAt}\ t\ i = (t_1, t_2)$, para un árbol t y entero i que satisfacen $i \leqslant \mathsf{size}\ t$:

- $\mathsf{max}\ (\mathsf{altura}\ t_1, \mathsf{altura}\ t_2) \leqslant \mathsf{altura}\ t$
- $\mathsf{size}\ t_1 = i$
- $\mathsf{toList}\ t_1 +\!\!+\ \mathsf{toList}\ t_2 = \mathsf{toList}\ t$
- $W_{\mathsf{splitAt}}(d), S_{\mathsf{splitAt}}(d) \in O(d)$, donde d es la altura del árbol que recibe la función.

b) Definir una función $\mathsf{rebalance} : \mathsf{BTree}\ a \to \mathsf{BTree}\ a$, que dado un árbol t construya un árbol balanceado con los mismos elementos de t (un árbol es balanceado si para cada par de hijos de un nodo cualquiera l_1 y l_2 la profundidad de los mismos difiere en a lo sumo 1).

c) Calcular el trabajo y la profundidad de la función mergesort sobre árboles vista en clase, la cual utiliza la función rebalance para que el resultado de merge sea un árbol balanceado. Para calcular el costo de la función rebalance suponer que se aplica sobre un árbol cuya altura es menor o igual a $(c \lg n)$, donde c es una constante y n es la cantidad de elementos del árbol.