Estructuras de Datos – UNQ

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todas las funciones y propiedades vistas en la prÃjctica y en la teórica, aclarando la referencia. Cualquier otra función o propiedad que se utilice* ***debe ser definida o demostrada****.*
- *No se olvide de poner nombre, nro. de alumno, nro. de hoja y cantidad total de hojas en cada una de las hojas.*
- *Le recomendamos leer el enunciado en su totalidad y organizar sus ideas antes de comenzar la resolución.*
- *¿Sabía que reusar código es una forma muy eficiente de disminuir el tiempo necesario para programar. . . , no?*
- *Recuerde que la intención es medir cuánto comprende usted del tema. Por ello, no dude en escribir todo lo que sabe, en explicar lo que se propone antes de escribir código, en probar sus funciones con ejemplos, etc.*

Ejercicio 1 Implementar una `FastAppendList`, un tipo abstracto de listas en el que la función de concatenación opera en tiempo constante. Consiste de la siguiente interfaz:

- `emptyF :: FAList a`

  Devuelve una `FastAppendList` vacía.
- `isEmptyF :: FAList a -> Bool`

  Responde si una `FastAppendList` está vacía.
- `appendF :: FAList a -> FAList a -> FAList a`

  Concatena dos `FastAppendList`.
- `consF :: a -> FAList a -> FAList a`

  Agrega un elemento a una `FastAppendList`.
- `headF :: FAList a -> a`

  Devuelve el primer elemento de una `FastAppendList`.
- `tailF :: FAList a -> FAList a`

  Devuelve una `FastAppendList` sin su primer elemento.

Para implementar estas operaciones utilice la siguiente estructura de ayuda (se adjuntan también estructuras auxiliares utilizadas):

```
data Maybe = Nothing | Just a

data TipTree a = Join (TipTree a) (TipTree a) | Tip a

data FAList a = MkFAList (Maybe (TipTree a))
```

Además debe tener en cuenta los siguientes detalles:

- Observar que se utiliza una estructura auxiliar llamada TipTree, que no es más que un árbol que tiene elementos sólo en la hojas. Además se utiliza el tipo Maybe ya visto en la materia.
- Las funciones `emptyF`, `isEmptyF`, `appendF` y `consF` deben ser O(1).
- El resto de las funciones no serán mejor que lineales en peor caso.
- Debe agregar los invariantes de representación que sean necesarios.
- No puede modificar la estructura provista para implementar la interfaz.

Ejercicio 2 Como *usuario* del módulo `FastAppendList` implemente las siguientes funciones:

a) `concatLists :: FAList [a] -> FAList a`

Toma una `FastAppendList` de listas, y forma una `FastAppendList` con todos los elementos de esas listas.

b) `findMin :: Ord a => FAList a -> Maybe a`

Toma una `FastAppendList` y devuelve el mínimo elemento dentro de la lista. Esta operación debe ser total.

Ejercicio 3 Implemente una `FastAppendList` de enteros en C/C++. Todas las operaciones en este caso deben ser constantes.

Puede aprovechar cualquier estructura vista en clase, pero debe crear un módulo que implemente la siguiente interfaz:

- `FAList emptyF();`

  Devuelve una `FastAppendList` vacía.

- `Bool isEmptyF(FAList xs);`

  Responde si una `FastAppendList` está vacía.

- `void appendF(FAList& xs, FAList& ys);`

  Concatena dos `FastAppendList`. Agrega los elementos de la segunda lista a la primera, y libera la memoria que ya no es utilizable de la segunda lista.

- `void mkConsF(int x, FastList& xs);`

  Agrega un elemento a la `FastAppendList`.

- `int headF(FAList xs);`

  Devuelve el primer elemento de una `FastAppendList`.

- `void tkTailF(FAList& xs);`

  Devuelve una `FastAppendList` sin su primer elemento.

- `void destroyF(FAList& xs);`

  Libera toda la memoria utilizada por una `FastAppendList`.