Integrador – 2015 - Estructuras de Datos – UNQ

Aclaraciones:

- *Esta evaluación es a libro abierto. Se pueden usar todas las funciones y tipos de datos vistos en la práctica y en la teórica, salvo que el enunciado indique lo contrario.*
- *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.*
- *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.*

Parte de Haskell

Un *ConjuntoSufijos* es una estructura de datos que permite conocer de forma eficiente palabras asociadas a cierto sufijo. Por ejemplo, los sufijos de la palabra *casa* son *casa*, *asa*, *sa*, *a* y el string vacío. Este último forma parte de todas las palabras.

La estructura nos permitirá principalmente conocer todas aquellas palabras que posean cierto sufijo. Podemos consultar por ejemplo todas las palabras que terminan con *asa*, en cuyo caso el resultado podrían ser las palabras *casa*, *masa*, *pasa*. Por otra parte, si preguntamos por las palabras que terminan con el string vacío, el resultado deberían ser todas las palabras de la estructura.

La interfaz que implementaremos es la siguiente:

- `vacio :: ConjuntoSufijos`
  Crea una *ConjuntoSufijos* vacío. $O(1)$.
- `terminanCon :: String -> ConjuntoSufijos -> Set String`
  Devuelve el conjunto de palabras asociadas a determinado sufijo. $O(log(n))$.
- `agregarSufijos :: String -> ConjuntoSufijos -> ConjuntoSufijos`
  Asocia una palabra a cada uno de sus sufijos. $O(s.log(n))$.
- `borrarSufijos  :: String -> ConjuntoSufijos -> ConjuntoSufijos`
  Borra la palabra de cada uno de sus sufijos. $O(s.log(n))$

Utilizaremos la siguiente representación:

```
data ConjuntoSufijos = CP (Map String (Set String))
```

Las claves del *Map* serán los sufijos, mientras que los valores serán las palabras que posean dicho sufijo.
La interfaz de *Set* que usaremos es la siguiente:

- `empty :: Set a` $O(1)$
- `size :: Set a -> Int` $O(1)$
- `belongs :: a -> Set a -> Bool` $O(log(n))$
- `add :: a -> Set a -> Set a` $O(log(n))$
- `remove :: a -> Set a -> Set a` $O(log(n))$
- `union :: Set a -> Set a -> Set a` $O(n.log(n))$

Implementar la estructura como un tipo abstracto, incluyendo invariantes de representación.
Luego como usuarios de la estructura implementar:

- `conjuntoSufijos :: Tree ConjuntoSufijos -> Set String`
  Une todas los palabras de los *ConjuntoSufijos* del arbol.
- `sufijoMayor :: [String] -> ConjuntoSufijos -> String`
  Devuelve la palabra de la lista que es sufijo de mayor cantidad de palabras del *ConjuntoSufijos*.

Parte de C++

Implementar en C++, incluyendo invariantes de representación, la siguiente interfaz de *Set*:

- `Set emptySet();` $O(1)$
- `int size(Set s);` $O(n)$
- `bool belongs(int x, Set s);` $O(n)$
- `void add(int x, Set& s);` $O(1)$
- `void remove(int x, Set& s);` $O(n)$
- `List toList(Set s);` $O(n)$

Luego implementar como usuario de este *Set* la siguiente función:

- `List toListOrdenado(Set s);` $O(n.logn)$
  Toma un conjunto y devuelve la lista de elementos del conjunto, ordenada de menor a mayor.