Práctica de ejercicios # 7 - Heaps y BSTs

Estructuras de Datos, Universidad Nacional de Quilmes

15 de octubre de 2020

Aclaraciones:

- ***Los ejercicios fueron pensados para ser resueltos en el orden en que son presentados****. No se saltee ejercicios sin consultar antes a un docente.*
- ***Recuerde que puede aprovechar en todo momento las funciones que ha definido****, tanto las de esta misma práctica como las de prácticas anteriores.*
- ***Pruebe todas sus implementaciones****, al menos en una consola interactiva.*
- *Es sumamente aconsejable* ***resolver los ejercicios utilizando primordialmente los conceptos y metodologías vistos en videos publicados o clases presenciales****, dado que los exámenes de la materia evaluarán principalmente este aspecto. Si se encuentra utilizando formas alternativas al resolver los ejercicios consulte a los docentes.*

Ejercicio 1

Indicar el costo de `heapsort :: Ord a => [a] -> [a]` (de la práctica anterior) suponiendo que el usuario utiliza una heap con costos logarítmicos de inserción y borrado.

Ejercicio 2

Implementar las siguientes funciones suponiendo que reciben un árbol binario que cumple los invariantes de BST y sin elementos repetidos. Todas deben ser implementadas en O(log n).

1. `balanceado :: Tree a -> Bool`
   Indica si el árbol está balanceado. Un árbol está balanceado cuando *para cada nodo* la diferencia de alturas entre el subarbol izquierdo y el derecho es menor o igual a 1.

2. `insertBST :: Ord a => a -> Tree a -> Tree a`
   Dado un BST inserta un elemento en el árbol.

3. `deleteBST :: Ord a => a -> Tree a -> Tree a`
   Dado un BST borra un elemento en el árbol.

4. `perteneceBST :: Ord a => a -> Tree a -> Bool`
   Dado un BST dice si el elemento pertenece o no al árbol.

5. `splitMinBST :: Ord a => Tree a -> (a, Tree a)`
   Dado un BST devuelve un par con el mínimo elemento y el árbol sin el mismo.

6. `splitMaxBST :: Ord a => Tree a -> (a, Tree a)`
   Dado un BST devuelve un par con el máximo elemento y el árbol sin el mismo.

7. `elMaximoMenorA :: Ord a => a -> Tree a -> Maybe a`
   Dado un BST y un elemento, devuelve el máximo elemento que sea menor al elemento dado.

8. `elMinimoMayorA :: Ord a => a -> Tree a -> Maybe a`
   Dado un BST y un elemento, devuelve el mínimo elemento que sea mayor al elemento dado.

Ejercicio 3

Implemente la interfaz de Set utilizando un BST como representación. Suponga que existe la siguiente subtarea:

```
armarBalanceado :: Ord a => a -> Tree a -> Tree a -> Tree a
```

que dados un elemento y dos árboles BST balanceados, denota un árbol BST balanceado, que une ese elementos y los dos árboles. A fines prácticos puede colocar `NodeT` como implementación de `armarBalanceado`, para poder ejecutar el código.

Ejercicio 4

Dada la siguiente interfaz y costos para el tipo abstracto Map:

- `emptyM :: Map k v`
  Eficiencia: $O(1)$.
- `assocM :: Eq k => k -> v -> Map k v -> Map k v`
  Eficiencia: $O(log\ n)$.
- `lookupM :: Eq k => k -> Map k v -> Maybe v`
  Eficiencia: $O(log\ n)$.
- `deleteM :: Eq k => k -> Map k v -> Map k v`
  Eficiencia: $O(log\ n)$.
- `keys :: Map k v -> [k]`
  Eficiencia: $O(n)$.

recalcular el costo de las funciones como usuario de Map de la práctica anterior.

Ejercicio 5

Implemente la interfaz de Map manteniendo los costos mencionados en el punto anterior, utilizando un BST como representación, y suponiendo la existencia de la subtarea armarBalanceado, ya mencionada en la práctica.