Práctica de ejercicios # 7 - Heaps y BSTs

Estructuras de Datos, Universidad Nacional de Quilmes

12 de mayo de 2022

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 priority queue con costos logarítmicos de inserción y borrado (o sea, usa una Heap como tipo de representación).

Ejercicio 2

Implementar las siguientes funciones suponiendo que reciben un árbol binario que cumple los invariantes de BST y sin elementos repetidos (despreocuparse por el hecho de que el árbol puede desbalancearse al insertar o borrar elementos). En todos los costos, N es la cantidad de elementos del árbol. Justificar por qué la implementación satisface los costos dados.

1. `belongsBST :: Ord a => a -> Tree a -> Bool`
   Propósito: dado un BST dice si el elemento pertenece o no al árbol.
   Costo: $O(log\ N)$

2. `insertBST :: Ord a => a -> Tree a -> Tree a`
   Propósito: dado un BST inserta un elemento en el árbol.
   Costo: $O(log\ N)$

3. `deleteBST :: Ord a => a -> Tree a -> Tree a`
   Propósito: dado un BST borra un elemento en el árbol.
   Costo: $O(log\ N)$

4. `splitMinBST :: Ord a => Tree a -> (a, Tree a)`
   Propósito: dado un BST devuelve un par con el mínimo elemento y el árbol sin el mismo.
   Costo: $O(log\ N)$

5. `splitMaxBST :: Ord a => Tree a -> (a, Tree a)`
   Propósito: dado un BST devuelve un par con el máximo elemento y el árbol sin el mismo.
   Costo: $O(log\ N)$

6. `esBST :: Tree a -> Bool`
   Propósito: indica si el árbol cumple con los invariantes de BST.
   Costo: $O(N^2)$

7. `elMaximoMenorA :: Ord a => a -> Tree a -> Maybe a`
   Propósito: dado un BST y un elemento, devuelve el máximo elemento que sea menor al elemento dado.
   Costo: $O(log\ N)$

8. `elMinimoMayorA :: Ord a => a -> Tree a -> Maybe a`
   Propósito: dado un BST y un elemento, devuelve el mínimo elemento que sea mayor al elemento dado.
   Costo: $O(log\ N)$

9. `balanceado :: Tree a -> Bool`
   Propósito: 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.
   Costo: $O(N^2)$

Ejercicio 3

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

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

recalcular el costo de las funciones como usuario de Map de la práctica anterior, siendo K es la cantidad de claves del Map. Justificar las respuestas.

Ejercicio 4

Dado la siguiente representación para el tipo abstrcto `Empresa`:

```
type SectorId = Int
type CUIL     = Int

data Empresa = ConsE (Map SectorId (Set Empleado))
                     (Map CUIL Empleado)
```

Donde se observa que:

- los empleados son un tipo abstracto.
- el primer map relaciona id de sectores con los empleados que trabajan en dicho sector.
- el segundo map relaciona empleados con su número de CUIL.
- un empleado puede estar asignado a más de un sector

- tanto `Map` como `Set` exponen una interfaz eficiente con costos logarítmicos para inserción, búsqueda y borrado, tal cual vimos en clase.

Y sabemos que la interfaz de `Empleado` es:

- `consEmpleado :: CUIL -> Empleado`
  Propósito: construye un empleado con dicho CUIL.
  Costo: $O(1)$
- `cuil :: Empleado -> CUIL`
  Propósito: indica el CUIL de un empleado.
  Costo: $O(1)$
- `incorporarSector :: SectorId -> Empleado -> Empleado`
  Propósito: incorpora un sector al conjunto de sectores en los que trabaja un empleado.
  Costo: $O(log\ S)$, siendo S la cantidad de sectores que el empleado tiene asignados.
- `sectores :: Empleado -> SectorId`
  Propósito: indica los sectores en los que el empleado trabaja.
  Costo: $O(1)$

Dicho esto, indicar invariantes de representación adecuados para la estructura y definir la siguiente interfaz de `Empresa`, respetando los costos dados y calculando los faltantes. Justificar todos los costos dados. En los costos, S es la cantidad de sectores de la empresa, y E es la cantidad de empleados.

- `consEmpresa :: Empresa`
  Propósito: construye una empresa vacía.
  Costo: $O(1)$
- `buscarPorCUIL :: CUIL -> Empresa -> Empleado`
  Propósito: devuelve el empleado con dicho CUIL.
  Costo: $O(log\ E)$
- `empleadosDelSector :: SectorId -> Empresa -> [Empleado]`
  Propósito: indica los empleados que trabajan en un sector dado.
  Costo: $O(logS + E)$
- `todosLosCUIL :: Empresa -> [CUIL]`
  Propósito: indica todos los CUIL de empleados de la empresa.
  Costo: $O(E)$
- `todosLosSectores :: Empresa -> [SectorId]`
  Propósito: indica todos los sectores de la empresa.
  Costo: $O(S)$
- `agregarSector :: SectorId -> Empresa -> Empresa`
  Propósito: agrega un sector a la empresa, inicialmente sin empleados.
  Costo: $O(logS)$
- `agregarEmpleado :: [SectorId] -> CUIL -> Empresa -> Empresa`
  Propósito: agrega un empleado a la empresa, en el que trabajará en dichos sectores y tendrá el CUIL dado.
  Costo: **calcular**.
- `agregarASector :: SectorId -> CUIL -> Empresa -> Empresa`
  Propósito: agrega un sector al empleado con dicho CUIL.
  Costo: **calcular**.

- `borrarEmpleado :: CUIL -> Empresa -> Empresa`
  Propósito: elimina al empleado que posee dicho CUIL.
  Costo: **calcular**.

Ejercicio 5

Como usuario del tipo `Empresa` implementar las siguientes operaciones, calculando el costo obtenido al implementarlas, y justificando cada uno adecuadamente.

- `comenzarCon :: [SectorId] -> [CUIL] -> Empresa`
  Propósito: construye una empresa con la información de empleados dada. Los sectores no tienen empleados.
  Costo: **calcular**.

- `recorteDePersonal :: Empresa -> Empresa`
  Propósito: dada una empresa elimina a la mitad de sus empleados (sin importar a quiénes).
  Costo: **calcular**.

- `convertirEnComodin :: CUIL -> Empresa -> Empresa`
  Propósito: dado un CUIL de empleado le asigna todos los sectores de la empresa.
  Costo: **calcular**.

- `esComodin :: CUIL -> Empresa -> Bool`
  Propósito: dado un CUIL de empleado indica si el empleado está en todos los sectores.
  Costo: **calcular**.