Estructuras de Datos

Parcial - Búsquedas por filtro

Segundo semestre 2024

En este examen se busca modelar un sistema de búsqueda de productos mediante el uso de filtros, al estilo de los buscadores de los sitios de compra online. Para dicho propósito se utilizará el TAD **Busqueda** que permite mantener un registro de los filtros deseados y de productos que satisfacen dicha búsqueda. Las interfaces de los TADs **Busqueda** y **Filtro** son las que se dan a continuación. En las mismas se llama *F* a la cantidad de filtros de la búsqueda, *P* a la cantidad de productos que tiene actualmente una búsqueda, y *A* a la mayor cantidad de atributos de un producto en la búsqueda.

TAD **Busqueda**

- **emptyB** `::` **Busqueda**, construye una nueva búsqueda sin filtros ni productos.
  Eficiencia: *O(1)*.
- **registrar** `::` **String -> Int -> Map String Int -> Busqueda -> Busqueda**, que dado un nombre de producto, su precio y un diccionario con atributos, si el producto y sus atributos satisfacen todos los filtros, agrega el producto a la búsqueda dada. Si el producto ya existe en la búsqueda, la misma no se modifica.
  Eficiencia: *O(F * log A + log P)*.
- **filtrar** `::` **Filtro -> Busqueda -> Busqueda**, que dado un filtro y una búsqueda lo registra, y aplica el filtro a la lista actual de productos, pudiendo eliminar algunos.
  Eficiencia: *O(P * (log A + log P))*.
- **siguiente** `::` **Busqueda -> (Maybe (String, Int), Busqueda)**, que dada una búsqueda, indica un nombre de producto y su precio (o **Nothing** si no hay más productos), y la búsqueda sin dicho producto.
  Eficiencia: *O(P + log P + log A)*.

TAD **Filtro**

- **mkMenorA** `::` **String -> Int -> Filtro**, que dados un nombre de atributo y un valor generan un nuevo filtro que se satisface cuando el atributo mencionado es menor que el valor dado.
  Eficiencia: *O(1)*.
- **mkMayorA** `::` **String -> Int -> Filtro**, que dados un nombre de atributo y un valor generan un nuevo filtro que se satisface cuando el atributo mencionado es mayor que el valor dado.
  Eficiencia: *O(1)*.
- **mkIgualA** `::` **String -> Int -> Filtro**, que dados un nombre de atributo y un valor generan un nuevo filtro que se satisface cuando el atributo mencionado es igual que el valor dado.
  Eficiencia: *O(1)*.
- **aplica** `::` **Filtro -> Map String Int -> Bool**, que dado un filtro y un diccionario de atributos, indica si el atributo correspondiente al filtro existe en el map dado y cumple la condición del filtro.
  Eficiencia: *O(log m)*, siendo *m* la cantidad de atributos del map dado.

Por ejemplo, la búsqueda b3 tiene un filtro que pide productos de precio mayor a 10, y registra 2 tipos de arroz (de los cuales solamente 1 permanecerá en la búsqueda):

```
b1 = registrar "arroz doble carolina Ala" 10 (assocM "peso gr." 500 emptyM)
                emptyB
b2 = registrar "arroz Condor" 12 emptyM b1
b3 = filtrar (mkMayorA "precio" 10) b2
```

EJERCICIOS

Ejercicio 1. Definir la siguiente función como usuario de **Busqueda** respetando el costo dado.

a. **`siguientesN :: Busqueda -> Int -> [(String, Int)]`**, que dada una búsqueda y un número obtiene dicha cantidad de pares nombre de producto, precio (o tantas como haya en la búsqueda si hay menos de dicho número). Se puede suponer que todos los productos de la búsqueda tienen precio como atributo.
Eficiencia: $O(n * (P + \log P + \log A))$ siendo n el número de productos indicados como argumento.

Ejercicio 2. Definir los invariantes de representación del TAD Busqueda para el siguiente tipo de representación:

```
data Busqueda =
  B (Map String (Map String Int))  -- nombre producto x atributos
    [Filtro]                       -- filtros aplicados a la búsqueda
```

Ejercicio 3. Definir como implementador de **Busqueda** con el tipo de representación del ejercicio anterior y sus invariantes, las siguientes funciones de interfaz, respetando los costos dados.

a. **`registrar :: String -> Int -> Map String Int -> Busqueda -> Busqueda`**.
OBSERVACIÓN: al registrar un producto, el precio debe incluirse como un atributo más que se supone que NO viene en el map dado.

b. **`filtrar :: Filtro -> Busqueda -> Busqueda`**.

Anexo de interfaces

Map, siendo K la cantidad de claves distintas en el map:

```
emptyM  :: Map k v                                  O(1)
assocM  :: Ord k => k -> v -> Map k v -> Map k v    O(log K)
lookupM :: Ord k => k -> Map k v -> Maybe v         O(log K)
deleteM :: Ord k => k -> Map k v -> Map k v         O(log K)
keysM   :: Map k v -> [k]                           O(K)
sizeM   :: Map k v -> Int                           O(1)
```