

Examen-Parcial-2020-Resuelto.pdf



pablofa02



Estructuras de Datos



2º Grado en Ingeniería del Software



**Escuela Técnica Superior de Ingeniería Informática
Universidad de Málaga**

**#Destino
Seúl**

Gana un viaje a Seúl

SAMSUNG

Ven a lanzar tu avión al tráiler de Samsung en
la Universidad de Málaga
Blvr Louis Pasteur s/n
7, 8 y 9 de Noviembre

Práctica Evaluable - Noviembre 2020

Estructuras de Datos

Computadores A / Informática D / Matemáticas + Informática

Definición informal del TAD

El TAD SetMultiMap es una colección de pares (K,Vs) donde:

- K es la *clave* y
- Vs un *conjunto no vacío* de valores de tipo V asociado a la clave K.

En un SetMultiMap las claves son **únicas** (no puede haber dos pares con la misma clave).

Ejemplo. El siguiente es un SetMultiMap que asocia a cada clave (String) un conjunto de enteros (Int) (representamos los pares (K,Vs) por la notación gráfica $K \rightarrow \{ V_1, \dots, V_n \}$):

```
"alfredo" -> { 9 }  
"juan" -> { 8, 1, 0 }  
"maria" -> { 4, -6, 8 }
```

Un SetMultiMap xs soporta las siguientes operaciones:

- empty: devuelve un SetMultiMap vacío.
- isEmpty xs: devuelve True si xs está vacío, False en caso contrario.
- size xs: devuelve el número de pares que contiene xs.
- isDefinedAt k xs: que devuelve True si la clave k aparece en xs y False en caso contrario.
- insert xs k v: si la clave k no está presente en xs, añade el par (k, { v }); si la clave k está presente en xs, añade v al conjunto de valores asociado a k.
- valuesOf k xs: devuelve, *si es posible*, el conjunto de valores asociado a k en xs.
- deleteKey k xs: si la clave k aparece en xs la elimina; de lo contrario no hace nada.
- deleteKeyValue k v xs: si la clave k aparece en xs y tiene asociado el valor v, elimina v; de lo contrario no hace nada.
- filterValues p xs: devuelve el SetMultimap que se obtiene al filtrar los valores asociados a cada clave con el predicado p.

Representación física

El TAD SetMultiMap se representará en Haskell por el siguiente tipo algebraico:

```
data SetMultiMap a b = Empty  
    | Node a (S.Set b) (SetMultiMap a b)  
    deriving Eq
```

Además, se debe satisfacer el siguiente **invariante de representación**:

- Los nodos están ordenados por clave
- No hay claves repetidas
- No hay claves que tengan asociado un conjunto vacío

Ejercicios

El fichero `SetMultiMap.hs` contiene definiciones incompletas de la implementación y ejemplos de uso de cada función en los que se muestra la salida esperada para un `SetMultiMap m1`.

Ejercicio 1. Completa en `SetMultiMap.hs` las definiciones de las funciones `empty`, `isEmpty`, `size`, `isDefinedAt`, `insert`, `valuesOf`, `deleteKey`, `deleteKeyValue`, y `filterValues`. Completa además el comentario indicando la clase de complejidad a la que pertenece cada función.

Ejercicio 2. Completa en `SetMultiMap.hs` los axiomas que definen la semántica de `deleteKeyValue`.

Ejercicio 3. Además de las operaciones anteriores, el TAD `SetMultiMap` tiene definido el siguiente plegado:

`fold f z xs`: pliega un `SetMultiMap` usando `z` para el caso base y `f` como función de plegado.

La función de plegado `f` recibe como parámetros la clave `k` y **cada uno** de los valores `v` asociados a `k` por separado; por ejemplo, para el `SetMultiMap m1` se `f` invocará 7 veces.

Completa en `SetMultiMapClient.hs` la definición de la función `compose`:

- `compose xs ys`: devuelve la composición de `xs` e `ys` (consultar en `SetMultiMapClient.hs` un ejemplo de uso).

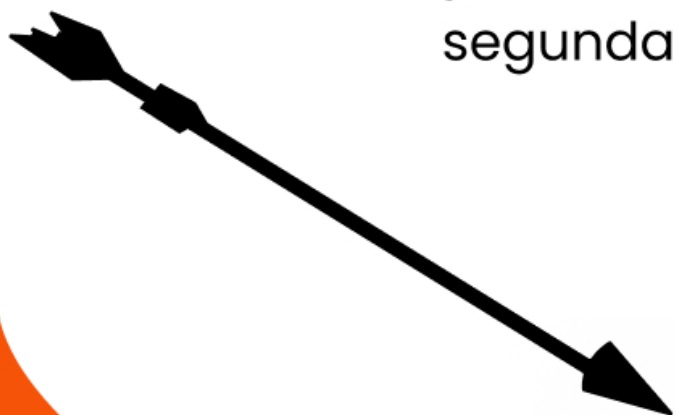
Entrega de la práctica evaluable

- Se deben entregar los ficheros:
 - `SetMultiMap.hs` y
 - `SetMultiMapClient.hs`
- No olvides completar tu nombre, apellidos y grupo
- Para que una función puntúe es necesario que compile



LOS JUEGOS DEL CUATRI

te imaginas no pagar ni primera ni
segunda matrícula??



PARTICIPA AQUÍ ↷



Si consigues subir más apuntes que tus
compañeros te regalamos una matrícula
valorada en 1000€

WUOLAH

```

1 {- |
2
3 Estructuras de Datos
4 2.º A Computadores, 2.º D Informática, 2.º Matemáticas + Informática
5
6 Práctica Evaluable - noviembre 2020
7
8 Apellidos, Nombre: Fazio Arrabal, Pablo
9
10 Grupo: 2.º Matemáticas + Informática
11
12
13 -}
14
15 module SetMultiMap( SetMultiMap
16                     , empty
17                     , isEmpty
18                     , size
19                     , isDefinedAt
20                     , insert
21                     , deleteKey
22                     , deleteKeyValue
23                     , valuesOf
24                     , filterValues
25                     , fold
26                     ) where
27
28 import           Data.List                (intercalate)
29 import           Test.QuickCheck
30
31 import qualified DataStructures.Set.LinearSet as S
32
33 -- Invariante de representación:
34 --   - Los nodos están ordenados por clave
35 --   - No hay claves repetidas
36 --   - No hay claves que tengan asociado un conjunto vacío
37
38 data SetMultiMap a b = Empty
39                     | Node a (S.Set b) (SetMultiMap a b)
40                     deriving Eq
41
42 -- ejemplo de SetMultiMap para probar las funciones
43
44 m1 :: SetMultiMap String Int
45 m1 = Node "alfredo" (mkSet [9]) (
46     Node "juan"      (mkSet [0,1,8]) (
47     Node "maria"     (mkSet [4,-6,8])
48     Empty))
49
50 mkSet :: Eq a => [a] -> S.Set a
51 mkSet = foldr S.insert S.empty
52
53 -- | Ejercicio 1 - Definición de operaciones y complejidad
54 -----
55
56 -- 0,25 ptos.
57 -- |
58 -- >>> empty
59 -- {}

```

quieres trabajar
en Wuolah??

TE BUSCAMOS

sin ánimo
de lucro,
chequea esto:



tú puedes
ayudarnos a
llevar
WUOLAH
al siguiente
nivel
(o alguien que
conozcas)

```
60
61 -- Complejidad: O(1)
62 empty :: SetMultiMap a b
63 empty = Empty
64
65 -- 0,25 ptos.
66 -- |
67 -- >>> isEmpty m1
68 -- False
69
70 -- Complejidad: O(1)
71 isEmpty :: SetMultiMap a b -> Bool
72 isEmpty Empty = True
73 isEmpty _ = False
74
75 -- 1 pto.
76 -- |
77 -- >>> size m1
78 -- 3
79
80 -- Complejidad: O(n)
81 size :: SetMultiMap a b -> Integer
82 size Empty = 0
83 size (Node x xs ys) = 1 + size ys
84
85 -- 1 pto.
86 -- |
87 -- >>> isDefinedAt "maria" m1
88 -- True
89 --
90 -- >>> isDefinedAt "eva" m1
91 -- False
92
93 -- Complejidad: O(n)
94 isDefinedAt :: (Ord a, Eq a) => a -> SetMultiMap a b -> Bool
95 isDefinedAt k Empty = False
96 isDefinedAt k (Node x xs ys) = (k==x) || isDefinedAt k ys
97
98 -- 1 pto.
99 -- |
100 -- >>> insert "alfredo" 5 m1
101 -- "alfredo" --> LinearSet(9,5)
102 -- "juan" --> LinearSet(8,1,0)
103 -- "maria" --> LinearSet(8,-6,4)
104 --
105 -- >>> insert "carmen" 20 m1
106 -- "alfredo" --> LinearSet(9)
107 -- "carmen" --> LinearSet(20)
108 -- "juan" --> LinearSet(8,1,0)
109 -- "maria" --> LinearSet(8,-6,4)
110
111 -- Complejidad: O(n)
112 insert :: (Ord a, Eq b) => a -> b -> SetMultiMap a b -> SetMultiMap a b
113 insert k v Empty = Node k (S.insert v S.empty) Empty
114 insert k v (Node x xs ys) | k > x && not(isDefinedAt k (Node x xs ys)) = Node x xs
    (Node k (S.insert v S.empty) ys)
115                               | k == x = Node x (S.insert v xs) ys
116                               | otherwise = Node x xs (insert k v ys)
117
118 -- 1 pto.
```

WUOLAH

```

119 -- |
120 -- >>> deleteKey "juan" m1
121 -- "alfredo" --> LinearSet(9)
122 -- "maria" --> LinearSet(8,-6,4)
123
124 -- Complejidad: O(n)
125 deleteKey :: (Ord a, Eq b) => a -> SetMultiMap a b -> SetMultiMap a b
126 deleteKey k Empty = Empty
127 deleteKey k (Node x xs ys) | k == x = ys
128                             | otherwise = Node x xs (deleteKey k ys)
129
130 -- 1 pto.
131 -- |
132 -- >>> deleteKeyValue "maria" 4 m1
133 -- "alfredo" --> LinearSet(9)
134 -- "juan" --> LinearSet(8,1,0)
135 -- "maria" --> LinearSet(8,-6)
136
137 -- Complejidad: O(n)
138 deleteKeyValue :: (Ord a, Eq b) => a -> b -> SetMultiMap a b -> SetMultiMap a b
139 deleteKeyValue k v Empty = Empty
140 deleteKeyValue k v (Node x xs ys) | k == x = if(S.size(S.delete v xs) == 0) then ys
141                                     | otherwise = Node x xs (deleteKeyValue k v ys)
142
143 -- 1 pto.
144 -- |
145 -- >>> valuesOf "maria" m1
146 -- Just LinearSet(8,-6,4)
147 --
148 -- >>> valuesOf "paco" m1
149 -- Nothing
150
151 -- Complejidad: O(n)
152 valuesOf :: (Ord a, Eq b) => a -> SetMultiMap a b -> Maybe(S.Set b)
153 valuesOf k Empty = Nothing
154 valuesOf k (Node x xs ys) | k == x = Just xs
155                             | otherwise = valuesOf k ys
156
157 -- 1,25 ptos.
158 -- |
159 -- >>> filterValues (> 0) m1
160 -- "alfredo" --> LinearSet(9)
161 -- "juan" --> LinearSet(8,1)
162 -- "maria" --> LinearSet(8,4)
163
164 -- Complejidad: O(n)
165 filterValues :: (Ord a, Eq b) => (b -> Bool)-> SetMultiMap a b -> SetMultiMap a b
166 filterValues p Empty = Empty
167 filterValues p (Node x xs ys) | funcional == S.empty = filterValues p ys
168                               | otherwise = Node x funcional (filterValues p ys)
169     where
170         funcional = (S.fold (\s q -> if p s then S.insert s q else q) S.empty xs)
171         -- S.insert devuelve el Set con los valores de las claves ordenados.
172
173 -- | Ejercicio 2 - Axiomas del TAD
174 -----
175 -- 1 pto.
176 -- | completa los axiomas que definen deleteKeyValue

```



```

177
178 ax_deleteKeyValue_empty x y = deleteKeyValue x y empty == empty
179 ax_deleteKeyValue_1 k v q = not(isDefinedAt k q) ==> deleteKeyValue k v q == q
180 -- ax_deleteKeyValue_2 k v q = isDefinedAt k q && S.isElem v (valuesOf k q) ==>
181   not(S.isElem v (valuesOf k (deleteKeyValue k v q))) -- Sin Tipo Maybe en ValuesOf
182
183 ----- NO EDITAR EL CÓDIGO DE ABAJO -----
184 -----
185 fold :: (Ord a, Eq b) => (a -> b -> c -> c) -> c -> SetMultiMap a b -> c
186 fold f z ms = recSetMultiMap ms
187   where
188     recSetMultiMap Empty = z
189     recSetMultiMap (Node k s ms)
190       | S.isEmpty s = recSetMultiMap ms
191       | otherwise = f k v (recSetMultiMap (Node k s' ms))
192     where
193       (v, s') = pickOne s
194     pickOne s = (v, S.delete v s)
195     where v = head $ S.fold (:) [] s
196
197 instance (Show a, Show b) => Show(SetMultiMap a b) where
198   show Empty = "{}"
199   show ms = intercalate "\n" (showKeyValues ms)
200   where
201     showKeyValues Empty = []
202     showKeyValues (Node k s ms) = (show k ++ " --> " ++ show s) : showKeyValues ms
203
204 instance (Ord a, Arbitrary a, Eq b, Arbitrary b) => Arbitrary (SetMultiMap a b)
205   where
206     arbitrary = do
207       xs <- listOf arbitrary
208       ys <- listOf arbitrary
209       return (foldr (uncurry insert) empty (zip xs ys))

```