DEPARTAMENTO
DE COMPUTACION
Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico 1

09 de Septiembre de 2014 — Paradigmas de Lenguajes de Programación

Grupo tas lokito

Integrante	LU	Correo electrónico
Gauder María Lara	027/10	marialaraa@gmail.com
Mataloni Alejandro	706/07	amataloni@gmail.com
Reartes Marisol	422/10	marisol.r5@hotmail.com

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires
Ciudad Universitaria - (Pabellón I/Planta Baja)
Intendente Güiraldes 2160 - C1428EGA
Ciudad Autónoma de Buenos Aires - Rep. Argentina
Tel/Fax: (54 11) 4576-3359
http://www.fcen.uba.ar

Índice

1. Código y comentarios

A continuación se muestra cada función con su explicación.

1.1. Ejercicio 1

La función belongs define si cierto elemento pertenece a las claves del diccionario dado como argumento.

```
belongs :: Eq k => k -> Dict k v -> Bool
belongs k d = not (null [x | x <- d, fst x == k])

(?) :: Eq k => Dict k v -> k -> Bool
d ? k = belongs k d
```

Ejemplo:

```
[("Pedro", ["Berlin", "Paris"]), ("Juan", ["Peru", "Amsterdam"])] ? "Juana"
--> False
```

1.2. Ejercicio 2

La función get retorna el significado correspondiente a la clave dada como argumento, se asume que el mismo se encuentra definido.

```
get :: Eq k => k -> Dict k v -> v
get k d = head ([snd x | x <- d, fst x == k])

(!) :: Eq k => Dict k v -> k -> v
d ! k = get k d
```

Ejemplo:

```
get "Argentina" [("Chile", 5), ("Argentina", 9), ("Uruguay", 3)]
--> 9

[( " calle " ," San Blas " ) ,( " ciudad " ," Hurlingham " )] ! " ciudad "
--> " Hurlingham "
```

1.3. Ejercicio 3

La función insertWith incorpora un nuevo significado a una clave dada, utilizando como función de incorporación a aquella dada como argumento. Si no existe la clave en el diccionario, entonces se la misma será agregada.

```
insertWith :: Eq k => (v -> v -> v) -> k -> v -> Dict k v -> Dict k v
insertWith fadd k v d
  | not (d ? k) = d ++ [(k,v)]
  | d ? k       = [if (fst x == k) then (fst x, fadd (snd x) v ) else x | x <- d]
```

Ejemplo:

```
insertWith (++) 1 [99,188] [(1,[1]),(2,[2])]
--> [(1,[1,99,188]),(2,[2])]
```

1.4. Ejercicio 4

En este caso, se ordenará los datos del array pasado como parámetro para que pase a tener una estructura de tipo Diccionario. Se asociará cada clave con la lista de todos los valores que tenía en el array original.

```
groupByKey :: Eq k => [(k,v)] -> Dict k [v]
groupByKey l = foldr (\x xs -> insertWith (++) (fst x) [(snd x)] xs) [] l
```

Ejemplo:

```
groupByKey [("fruta", "Banana"), ("carne", "Picada"), ("verdura", "Espinaca")]
--> [("fruta", ["Banana"]), ("carne", ["Picada"]),("verdura", ["Espinaca"])]
```

1.5. Ejercicio 5

En unionWith se espera que a partir de dos diccionarios y una función de unificación, se combinen ambos para obtener un único diccionario resultante.

```
unionWith :: Eq k => (v -> v -> v) -> Dict k v -> Dict k v -> Dict k v
unionWith f dict = foldr (\x rec_dict -> insertWith f (fst x) (snd x) rec_dict) dict
```

Ejemplo:

```
unionWith (+) [("Algodon", 20), ("Azucar", 10)]  [("Miel", 5),("Algodon", 5)]
--> [("Algodon", 25), ("Azucar", 10),("Miel", 5)]
```

1.6. Ejercicio 6

Tendrá como objetivo que a partir de una lista divide la carga de la misma de manera balanceada entre una determinada cantidad de sublistas.

```
distributionProcess :: Int ->  [a] -> [[a]]
distributionProcess n = foldr (\x rec -> (tail rec) ++ [x : (head rec)]) (replicate n [])
```

Ejemplo:

```
distributionProcess 1 [1, 4, 100, 104]
--> [[1, 4, 100, 104]]
```

1.7. Ejercicio 7

La función mapperProcess deberá aplicar el tipo Mapper a cada uno de los elementos pasados en el array por parámetro y luego agrupar los resultados por claves.

```
mapperProcess :: Eq k => Mapper a k v -> [a] -> [(k,[v])]
mapperProcess m la = groupByKey (concat [m e | e <- la])
```

Ejemplo:

```
mapperProcess (\x -> [(x,1)]) ["Pablo", "Pedro", "Pepe", "Pedro", "Pedro", "Guillermo", "Aldana"]
--> [("Pablo",[1]), ("Pedro",[1,1,1]), ("Pepe",[1]),("Guillermo",[1]), ("Aldana",[1])]
```

1.8. Ejercicio 8

combinerProcess consiste en unificar los resultados de cada sublista pasada por parámetro agrupándolos por clave y ordenandolos de manera creciente según la clave.

```
combinerProcess :: (Eq k, Ord k) => [[(k, [v])]] -> [(k,[v])]
combinerProcess l = sortBy (comparing $ fst) (foldr (\x rec -> unionWith (++)  x rec) [] l)
```

Ejemplo:

```
combinerProcess [[("Pablo",[1]), ("Pedro",[1,1,1])], [("Pepe",[1])],[("Guillermo",[1]),
("Aldana",[1])]]
--> [("Aldana",[1]), ("Guillermo",[1]),("Pablo",[1]), ("Pedro",[1,1,1]), ("Pepe",[1])]
```

1.9. Ejercicio 9

La función reducerProcess aplica el tipo Reducer a cada para de elementos de la lista pasada como argumento y luego combina los resultados para obtener una única lista.

```
reducerProcess :: Reducer k v b -> [(k, [v])] -> [b]
reducerProcess r l = concat [r e | e <- l]
```

Ejemplo:

```
reducerProcess (\(x,y) -> [sum y])  [("Alvear",[3,5,1,32]),("Cabildo",[4,6,4,4,4]),
("Independencia",[1,1])]
--> [41, 22, 2]
```

1.10. Ejercicio 10

La función mapReduce combina todas las instrucciones desarrolladas de la Sección 2, para poder implementar una técnica de MapReduce, que permite en tiempo óptimo procesar grandes cantidades de datos.

```
mapReduce :: (Eq k, Ord k) => Mapper a k v -> Reducer k v b -> [a] -> [b]
mapReduce m r l = reducerProcess r (combinerProcess [mapperProcess m x | x <- dist])
  where dist = distributionProcess 100 l
```

Ejemplo:

```
mapReduce (\(nom,ciudad) -> [(ciudad,1)]) (\(ciudad,cant) -> [(ciudad, sum cant)])
[("Pablo","Berlin"), ("Gabriela","Amsterdam"), ("Taihú","Cairo"), ("Pablo", "Cairo"),("Taihú","Ams
--> [("Amsterdam", 3), ("Berlin", 1), ("Cairo", 2)]
```

1.11. Ejercicio 11

La función visitasPorMonumento deberá, a partir de un array de nombres de monumentos repetidos generar un Diccionario que indique la cantidad de repeticiones que hay por cada uno.

```
visitasPorMonumento :: [String] -> Dict String Int
visitasPorMonumento m = mapReduce (\l -> [(l,1)]) (\(x,y) -> [(x,sum y)]) m
```

Ejemplo:

```
 visitasPorMonumento [ "Obelisco", "Cid", "San Martín", "Cid", "Bagdad Bridge", "Cid",
 "San Martín", "Obelisco"]
--> [("Obelisco",2), ("Cid",3), ("San Martín",2), ("Bagdad Bridge", 1)]
```

1.12. Ejercicio 12

En este caso, monumentosTop deberá, a partir de una lista de nombres de monumentos repetidos según la cantidad de visitas, ordenarlos por cantidad de visitas, de mayor a menor, sin repetidos.

```
monumentosTop :: [String] -> [String]
monumentosTop m = mapReduce (\(x,y) -> [(-y,x)]) (\(x,y) ->  y) (visitasPorMonumento m)
```

Ejemplo:

```
monumentosTop [ "Obelisco", "Obelisco", "Cid", "San Martin", "San Martin", "San Martin",
"Obelisco", "Obelisco"]
--> ["Obelisco", "San Martin", "Cid"]
```

1.13. Ejercicio 13

Indica la cantidad de monumentos existentes para cada país que tenga.

```
monumentosPorPais :: [(Structure, Dict String String)] -> [(String, Int)]
monumentosPorPais s = mapReduce (\(s,d) ->
[(d ! "country", initNumber s)]) (\(x,y) -> if sum y /= 0 then [(x,sum y)] else []) s

initNumber :: Structure -> Int
```

```
initNumber Monument = 1
initNumber Street = 0
initNumber City = 0
```

Ejemplo:

```
monumentosPorPais [ (Monument, [("Antiguedad","muchos años"),("name","Catedral de Leon"),
("country", "España")]), \\
  (Monument, [("name","Cristo-Rei de Almada"),("Renovacion", "ninguna"),("country", "Portugal")]),
  (Monument, [("name", "Monasterio de Piedra"),("country", "España")]), \\
  (Monument, [("name", "Iglesia de San Francisco"),("country", "Argentina")]), \\
  (Street, [("name", "Corrientes"),("country", "Argentina")]), \\
  (City, [("name", "Berlina"),("country", "Alemania"),("habitantes", "Dos millones")]), \\
  (Street, [("name", "Armenia"),("country", "Argentina"),("Barrio", "Palermo")])] \\
--> [("Argentina",1),("España",2),("Portugal",1)]
```