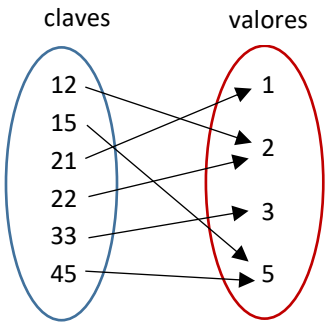
Nombre: Grupo:

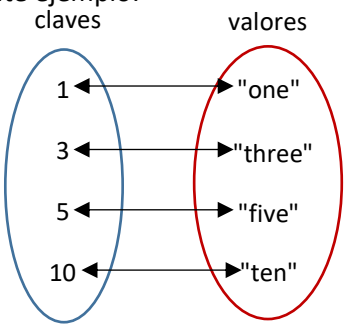
Estructuras de datos. Septiembre 2018. Universidad de Málaga

Un diccionario es una estructura de datos que asocia un conjunto de claves únicas a una colección de valores. Una buena implementación de un diccionario debe proporcionar, entre otras cosas, operaciones eficientes para insertar y borrar asociaciones y para localizar el valor asociado a una clave dada. Los diccionarios son normalmente representados usando los mismos diagramas que se usan para representar funciones en matemáticas, tal y como se muestra en el siguiente ejemplo en el cual cada clave (un número) se asocia a su último dígito:



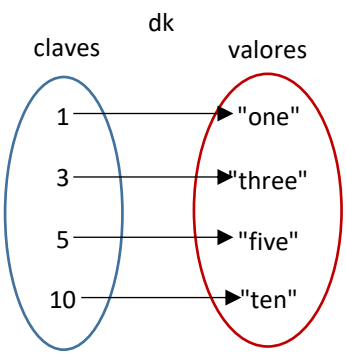
Hay que tener en cuenta que, ya que las claves son únicas, esto implica que existe una sola flecha que parte de cada elemento en el conjunto inicial (el conjunto de claves) pero esto no tiene por qué ser así para los valores (múltiples flechas pueden llegar a un mismo valor).

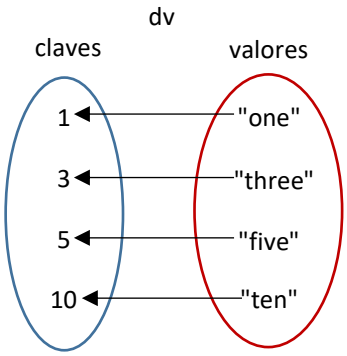
Después de este breve repaso a los diccionarios, podemos introducir el concepto de diccionario bidireccional (también conocidos como BiMaps o BiDictionaries). Un BiDictionary es una asociación biyectiva entre dos conjuntos únicos de valores (el conjunto de claves y el conjunto de valores). Recordemos que una función biyectiva es una correspondencia uno-a-uno de forma que, cada clave tiene asociado un único valor y de la misma forma cada valor está asociado a una sola clave. De forma gráfica, esto significa que una flecha parte de cada clave y una flecha llega a cada valor, como muestra el siguiente ejemplo:



La estructura de datos BiDictionary debe proporcionar una operación eficiente para encontrar el valor asociado a una clave (de forma similar al diccionario) y además una operación eficiente para encontrar la clave asociada a un valor dado.

Una forma de implementar un BiDictionary es a través de dos diccionarios: uno que asocia claves a valores (dk) y otro que asocia valores a claves (dv) y que representa al diccionario inverso. Cada vez que el BiDictionary es modificado, ambos diccionarios deben actualizarse debidamente. Por ejemplo, los diccionarios que implementan el BiDictionary mostrado anteriormente usando esta aproximación son los siguientes:





Siguiendo esta aproximación, búsquedas por clave pueden ser llevadas a cabo de forma eficiente usando el primer diccionario (dk), y de forma similar, búsquedas por valor se llevan a cabo usando el segundo diccionario (dv). Hay que tener en cuenta que un invariante que debe cumplirse en esta implementación es el que establece que si una clave k está asociada a un valor v en el primer diccionario (dk) entonces v debe estar asociado a k en el segundo diccionario (dv) y viceversa. Además, ambos diccionarios en esta implementación son siempre inyectivos (en ambos diccionarios una única flecha llega a cada valor).

Haskell

Se proporciona una implementación de la estructura de datos Diccionario en el fichero `DataStructures.Dictionary.AVLDictionary.hs`. El objetivo es completar la implementación de la estructura de datos BiDictionary en el fichero `DataStructures.Dictionary.AVLBiDictionary.hs` de la forma descrita anteriormente. Al final del examen, debes subir este último fichero con la solución al Campus Virtual.

El siguiente tipo de datos representa un diccionario bidireccional entre claves del tipo a y valores del tipo b. Denominaremos al primer diccionario dk (de claves a valores) y al segundo dv (de valores a claves):

```
import qualified DataStructures.Dictionary.AVLDictionary as D

data BiDictionary a b = Bi (D.Dictionary a b) (D.Dictionary b a)
```

Se pide completar la implementación añadiendo las siguientes funciones descritas a continuación. Se recuerda que se debe proporcionar una implementación lo más eficiente posible.

a) (0.5 puntos) Define una función `empty` que devuelva un `BiDictionary` vacío (sin asociaciones), una función `isEmpty` para chequear si un `BiDictionary` está vacío y una función `size` que devuelva el número total de asociaciones almacenadas en el `BiDictionary` (hay 4 asociaciones en el `BiDictionary` representado anteriormente).

b) (1 punto) Define una función `insert` que reciba una clave, un valor y un `BiDictionary`, y devuelva un nuevo `BiDictionary` resultante de insertar la asociación entre la clave y el valor dados. Esta operación podrá ser usada para introducir una nueva asociación o para reemplazar una ya existente.

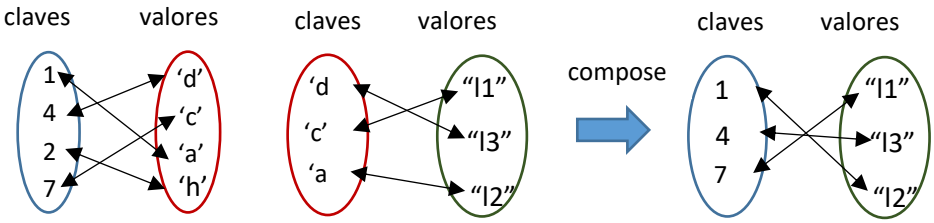
c) (0.5 puntos) Define la función `keyOf` que dada una clave y un `BiDictionary`, devuelva el valor asociado a la clave en el BiDictionary. El resultado será de tipo `Maybe` de forma que se devuelva `Nothing` cuando la clave no exista en el BiDictionary.

d) (0.5 puntos) Define la función `valueOf` que dado un valor y un `BiDictionary`, devuelva la clave asociada al valor en el BiDictionary. El resultado será de tipo `Maybe` de forma que se devuelva `Nothing` cuando el valor no exista en el `BiDictionary`.

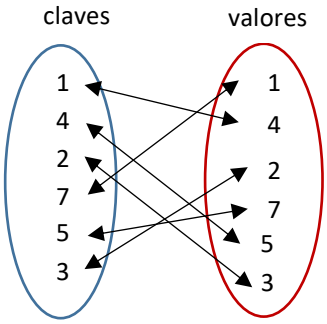
e) (0.5 puntos) Define la función `deleteByKey` que dada una clave y un `BiDictionary`, devuelva el `BiDictionary` resultante de eliminar la asociación donde participa la clave indicada. Si no existe tal asociación, el mismo `BiDictionary` es devuelto.

f) (0.5 puntos) Define la función `deleteByValue` que dado un valor y un `BiDictionary`, devuelva el `BiDictionary` resultante de eliminar la asociación donde participa el valor indicado. Si no existe tal asociación, el mismo `BiDictionary` es devuelto.

g) (1 punto) Un diccionario es inyectivo si cada uno de sus valores aparece en una sola asociación (una única flecha llega a cada valor del diccionario). Define una función `toBiDictionary` que dado un `Dictionary` (que debe ser inyectivo) devuelva el correspondiente `BiDictionary`. Si el diccionario que se recibe como parámetro no es inyectivo se debe devolver error.



h) (1.5 puntos) Define la función `compose` que recibe dos `BiDictionary` (`bd1` y `bd2`) y genera un `BiDictionary` nuevo. El nuevo diccionario tendrá una asociación `a->c` cada vez que se cumpla que existe una asociación `a->b` en `bd1` y una asociación `b->c` en `bd2` tal y como muestra la figura anterior.



i) (1.5 puntos) Define la función `isPermutation` que dado un `BiDictionary` compruebe si se trata de una permutación. Un `BiDictionary` es permutación cuando las asociaciones constituyen una biyección de un conjunto consigo mismo. Por ejemplo, el `BiDictionary` mostrado en la figura anterior es una permutación ya que las claves y los valores del mismo son el mismo conjunto.

Alumnos sin evaluación continua

j) (1.5 puntos) Define la función `orbitOf` que recibe un `BiDictionary` que es permutación y una clave inicial y devuelve una lista con la órbita de la clave inicial en el `BiDictionary`. La órbita se obtiene de la siguiente forma: partiendo de la clave inicial, calcula su valor asociado. Usa este valor como clave y repite el proceso hasta que llegues a la clave inicial. En la figura anterior, la órbita de 1 en el `BiDictionary` es [1, 4, 5, 7]. Si el diccionario recibido como parámetro no es permutación se debe lanzar un error.

k) (1 punto) Define una función `cyclesOf` que dado un `BiDictionary` que es permutación obtiene todos sus ciclos (que no son más que las órbitas). Un posible algoritmo para calcularlas es el siguiente: Se parte de un conjunto S con los elementos del `BiDictionary`. Se calcula la órbita de un elemento cualquiera de S en el `BiDictionary`. A continuación, se eliminan de S todos los elementos que forman parte de la órbita calculada y se repite el proceso hasta que S quede vacío. En la figura anterior, los ciclos del `BiDictionary` son [[1, 4, 5, 7], [2, 3]] (el orden no tiene necesariamente por qué coincidir). Si el diccionario recibido como parámetro no es permutación se debe lanzar un error.

Java

Completar la clase `HashBDictionary<K, V>`. Implementa los siguientes constructores y métodos para esta clase:

a) (0.5 puntos) Un constructor que inicialice adecuadamente la estructura que inicialmente estará vacía. Implementa también los métodos `public boolean isEmpty()` y `public int size()`

b) (1 punto) El método `public void insert(K k, V v)` que inserta la asociación formada por la clave k y el valor. Si la asociación ya existe, se actualiza con el valor indicado.

c) (0.5 puntos) El método `public K keyOf(V v)` que dado un valor, devuelve la clave asociada al mismo en el diccionario bidireccional. Se devolverá `null` si el valor no existe.

d) (0.5 puntos) El método `public V valueOf(K k)` que dada una clave devuelve el valor asociado a la misma en el diccionario bidireccional. Se devolverá `null` si la clave no existe.

e) (0.5 puntos) El método `public void deleteByKey(K k)` que dada una clave k, borra la asociación donde participa la clave indicada (si es que existe).

f) (0.5 puntos) El método `public void deleteByValue(V v)` que dado un valor v, borra la asociación donde participa el valor indicado (si es que existe).

g) (1 punto) El método estático `public static <K,V extends Comparable<? super V>> BDictionary<K, V> toBDictionary(Dictionary<K,V> dict)` que dado un diccionario calcula y devuelve el diccionario bidireccional asociado. Si el diccionario que se recibe como parámetro no es inyectivo, se debe lanzar una excepción del tipo `IllegalArgumentException`.

h) (1.5 puntos) El método `public <W> BDictionary<K, W> compose(BDictionary<V,W> bdic)` que recibe como parámetro un diccionario bidireccional y devuelve uno nuevo resultante de aplicar la composición de los dos diccionarios (desde el que se llama al método y el que se recibe como parámetro) .

i) (1.5 puntos) El método estático `public static <K extends Comparable<? super K>> boolean isPermutation(BDictionary<K,K> bd)` que dado un diccionario bidireccional indica si este es permutación o no.

Alumnos sin evaluación continua

j) (1.5 puntos) El método estático `public static <K extends Comparable<? super K>> List<K> orbitOf(K k, BDictionary<K,K> bd)` que dada una clave k y un diccionario bidireccional calcula la órbita de k en el diccionario. Si el diccionario recibido como parámetro no es permutación se debe lanzar una excepción del tipo `IllegalArgumentException`.

k) (1 punto) El método estático `public static <K extends Comparable<? super K>> List<List<K>> cyclesOf(BDictionary<K,K> bd)` que dado un diccionario bidireccional calcule sus ciclos. Es decir, una lista que contenga todas las órbitas del diccionario. Si el diccionario recibido como parámetro no es permutación se debe lanzar una excepción del tipo `IllegalArgumentException`.