Estructura de Datos. Grado en Ingeniería Informática. Enero 2021. Control 3 (0.75 puntos)

***ATENCIÓN* Se valorará la eficiencia de las soluciones y la claridad de los algoritmos.**

En este ejercicio vamos a implementar una tabla hash usando la técnica Linear Probing (tal como se describe en las transparencias 29 a 55 del tema 5). En este caso vamos a almacenar asociaciones `(key,value)` para después poder implementar tanto conjuntos como diccionarios. La clase debe implementar la siguiente interfaz:

```
public interface HashTable<K, V> extends Iterable<K> {
   public boolean isEmpty();
   public int size();
   public void insert(K key, V value);
   public V search(K key);
   public boolean isElem(K key);
   public void delete(K key);
   Iterable<K> keys();
   Iterable<V> values();
   Iterable<Tuple2<K,V>> keysValues();
}
```

En el campus virtual se proporciona una plantilla de la clase `LinearProbingHashTable.java` donde algunos métodos ya están implementados y otros debéis implementarlos vosotros. En esta clase, para memorizar pares de claves y valores usaremos dos arrays circulares (`keys` para las claves y `values` para los valores). Si una clave se encuentra en la posición `p` del array `keys`, el valor asociado se encuentra en la posición `values[p]`.

Ejercicio 1. (1 pto.)

Define el método `private int searchIdx(K key)` que toma una clave `key` y devuelve la posición del array de claves donde se encuentra o donde se debería encontrar según la técnica Linear Probing. Para ello, localiza la posición inicial de la clave según su `hash`, y mientras esa posición esté ocupada por otro elemento, avanza una posición (el array es circular). Finalmente, devuelve la primera posición libre encontrada (lo que significa que la `key` no estaba) o la posición ocupada por él mismo (lo que significa que la `key` sí estaba).

Ejercicio 2. (1. pto.)

Define el método `public V search(K key)` que devuelve el valor asociado a clave `key` o `null` si la clave no se encuentra en la estructura.

Ejercicio 3. (1 pto.)

Define el método `public boolean isElem(K key)` que determina si la clave `key` se encuentra en la estructura.

Ejercicio 4. (2 ptos.)

Define el método `public insert(K key, V value)` que inserta el par `(key,value)` en la estructura. Si la clave ya estaba en la estructura, modifica el valor asociado.

Ejercicio 5. (3 ptos.)

Define el método `public void delete(K key)` que elimina una asociación con esa clave de la estructura. Si la clave no está no se hace nada. Para implementar esta operación, debemos localizar primero la posición `p` correspondiente a la clave `key` en el array de claves, y asignar null a las posiciones `keys[p]` y `values[p]`. A continuación, debemos trasladar (borrar y reinsertar) todos los elementos situados en las siguientes posiciones para no dejar huecos hasta encontrar una posición vacía. (ver transparencias 50-55 del tema 5)

Ejercicio 6. (2 ptos.)

Con objeto de implementar los diferentes iteradores que define la interfaz, vamos a construir una clase que nos permita movernos por las posiciones que están ocupadas en la estructura saltando las que están libres. Luego se proporcionan tres subclases que implementan los tres iteradores, uno que recorre las claves, otro que recorre los valores y otro que recorre pares de clave y valor.

```
private class TableIter {
    private int visited;   // cuenta el número de elementos ya visitados por el iterador
    protected int nextIdx; // índice del siguiente elementos a visitar por el iterador

    public TableIter() {
        visited = 0;
        nextIdx = -1; // así al incrementar se sitúa en la posición 0
    }

    public boolean hasNext() {
        return visited < size;  // cierto si no se han visitado todos
    }

    // avanza nextIdx al siguiente índice no nulo a visitar
    public void advance() {
        // to be completed
    }

    public void remove() {
        throw new UnsupportedOperationException();
    }
}
```

Esta clase mantiene la cuenta del número de elementos que ya se han visitado (`visited`) y la posición del último elemento visitado (`nextIdx`, al principio es -1). Los diferentes iteradores que heredarán de esta clase solo definirán el método `next()`. Este método `next()` llamará a `advance()` para que se sitúe en el siguiente elemento a visitar. Por ejemplo, la implementación del iterador por claves es:

```
private class KeysIter extends TableIter implements Iterator<K> {
    public K next() {
        advance();
        return keys[nextIdx];
    }
}
```

Completa la definición del método `public void advance()` que verificará que hay más elementos y si es así, coloca `nextIdx` en la posición del siguiente elemento a visitar (ten en cuenta que en los arrays hay posiciones que están a `null` lo que significa que ahí no hay elementos y que el array es circular). Si no hubiera más elementos debería lanzar la excepción `NoSuchElementException`.

NOTA Para probar vuestra solución disponéis de la clase `HashTableTest` en el paquete `demos.hashTable`.