Soluciones Ejercicios Tema 13

Germán Moltó
gmolto@dsic.upv.es
Estructuras de Datos y Algoritmos
Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Implementar rehashing en TablaHash

```
@SuppressWarnings("unchecked")
protected final void rehashing() {
   int nuevaCapacidad = siguientePrimo(elArray.length * 2);
   ListaConPI<E> elArrayAntiguo[] = this.elArray;
   this.elArray = new LEGListaConPI[nuevaCapacidad];
   for (int i = 0; i < elArray.length; i++) elArray[i] = new LEGListaConPI<E>();
     talla = 0;
     for (int i = 0; i < elArrayAntiguo.length; i++) {
        ListaConPI<E> lpi = elArrayAntiguo[i];
        for (lpi.inicio(); !lpi.esFin(); lpi.siguiente()){
            E d = lpi.recuperar();
            ListaConPI<E> l = elArray[posTabla(d)];
            l.insertar(d); talla++;
        }
     }
}
```


Implementar toArray en TablaHash

```
public final void toArray(E[] v) {
  int pos = 0;
  for (int i = 0; i < this.elArray.length && pos < v.length; i++) {
    ListaConPI<E> lpi = elArray[i];
    for (lpi.inicio(); !lpi.esFin(); lpi.siguiente())
        v[pos++] = lpi.recuperar();
  }
}
```


Radares de Tráfico (I)

```
public class Matricula{
 …
public Matricula (int numeros, String letras, String anyo){
     this.numeros = numeros;  this.letras = letras; this. anyo = anyo;
  }
  public Matricula (int numeros, String letras){
     this(numeros, letras, "");
  }
  public boolean equals(Object x){
     Matricula m = (Matricula) x;
     return ( this.numeros == m.numeros && this.letras.equals(m.letras) );
  }
  public int hashCode(){ return this.letras.hashCode()+numeros; }
… }
```

Radares de Tráfico (II)

- Implementación de la operación de contabilización del número de veces que ha sido vista dicha matrícula superando el límite de velocidad.

```
public static void registroMatricula(Diccionario<Matricula, Integer>
  dM, Matricula m){
    int nMultas;
     try{
        nMultas = dM.recuperar(m).intValue();   nMultas++;
       }catch(ElementoNoEncontrado ex){  nMultas = 1; }
       dM.insertar(m, new Integer(nMultas));
}
```


Agenda de Teléfonos (I)

```
public class EntradaAgenda implements Comparable<EntradaAgenda>{
protected String nombre, telefono;

public EntradaAgenda(String n, String t) { nombre = n; telefono = t; }
public EntradaAgenda(String n) { this(n, ""); }
public String toString() { return nombre+"("+telefono+")"; }
public int compareTo(EntradaAgenda x){
   return this.nombre.compareTo(x.nombre);
}
public int hashCode(){ return this.nombre.hashCode();}
public boolean equals(Object x){
   return this.compareTo((EntradaAgenda)x) == 0;
}
```


Agenda de Teléfonos (II)

- La precondición del método nos garantiza que siempre habrá al menos un elemento en la Tabla Hash, así que elegimos ese elemento como mínimo inicial.

```
public E recuperarMin(){
   E min = null; int i; boolean stop = false;
   for (i = 0; i < elArray.length && !stop ; i++)
     if (!elArray[i].esVacia() ) {
                elArray[i].inicio();
        min = elArray[i].recuperar(); stop = true;
      }
      //min contiene el primer elemento encontrado en la Tabla Hash
```


Agenda de Teléfonos (III)

- Queda pendiente recorrer el resto de elementos de la tabla hasta obtener el mínimo de todos:

```
for (; i < elArray.length; i++){
     for (elArray[i].inicio(); !elArray[i].esFin(); elArray[i].siguiente()){
              E dato = elArray[i].recuperar();
             if (dato.compareTo(min) < 0) min = dato;
         }
     }
   return min; }
```

Agenda de Teléfonos (IV)

```
public static void main(String args[]){
      Agenda a = new Agenda(100);
      a.insertar(new EntradaAgenda("Marga Garcia",“635422876”));
      a.insertar(new EntradaAgenda(“Paula Garcia",“635422876”));
      a.insertar(new EntradaAgenda(“Alba Lopez",“635422876”));
      System.out.println("La clave más pequeña es: " + a.recuperarMin());
      try{
        System.out.println(a.recuperar("Lucas Garcia"));
       }catch(ElementoNoEncontrado ex){ }

  }
```


Módulo de Autorización (I)

```
public class Usuario
{
 private String nombre;    private String password;
 public Usuario(String nombre){this(nombre, null); }
 public Usuario(String nombre, String password){
   this.nombre = nombre; this.password = password;
 }
 public String getNombre(){return this.nombre;}
 public String getPassword(){return this.password;}
 public int hashCode(){return this.nombre.hashCode(); }
 public boolean equals(Object x){
    return (this.nombre.equals( ((Usuario) x).getNombre()));
  }
}
```


Módulo de Autorización (II)

```
public class UsuarioExistenteException extends
   Exception
{
  public UsuarioExistenteException(String msg){
    super(msg);
  }
}
```


Módulo de Autorización (III)

```
public class ModuloAutorizacion
{
 private TablaHash<Usuario> t;
 public ModuloAutorizacion(){
   t = new TablaHash<Usuario>();
 }
 public void registraUsuario(String nombre, String password) throws
   UsuarioExistenteException{
 try{
      t.recuperar(new Usuario(nombre,password));
      throw new UsuarioExistenteException("Ya existe " + nombre);
   }catch(ElementoNoEncontrado ex){
       t.insertar( new Usuario(nombre,password));
   } }
```

Módulo de Autorización (IV)

```
public boolean estaAutorizado(String nombre, String password){
 try{
    String stored_pwd = t.recuperar(new Usuario(nombre)).getPassword();
    if (stored_pwd.equals(password)) return true;
    else return false;
  }catch(ElementoNoEncontrado ex){
    return false;
  }
 }
}
```


Módulo de Autorización (V)

```
public class TestModuloAutorizacion
{
  public static void main(String args[]){
    String username = "German Molto"; String password = "x8272";
    ModuloAutorizacion ma = new ModuloAutorizacion();
    try{
      ma.registraUsuario(username, password);
     }catch(UsuarioExistenteException ex){
      System.err.println("El usuario " + username + " ya existia");
     }
    if (ma.estaAutorizado(username, password))
      System.out.println(username + " SI esta autorizado.");
    else System.out.println(username + " NO esta autorizado.");
  }}
```


La Clase Tarea

```
public class Tarea {
private String nombre; private long timeStamp;
  public Tarea(String nombre){ this.nombre = nombre; this.timeStamp =
   System.currentTimeMillis(); }
  public  String getNombre(){ return this.nombre; }
  public long getTimeStamp(){ return this.timeStamp; }
  public int hashCode(){
      return this.nombre.hashCode() + this.timeStamp;
  }
  public boolean equals(Object x){
      Tarea t = (Tarea) x;
      return  this.nombre.equals( t.nombre) && this.timeStamp ==
   t.timeStamp;
   }}
```


Recuperar Iguales en TablaHash (1/2)

1. Solución al apartado 1

```
public ListaConPI<E> recuperarIguales(E x) {
  ListaConPI<E> res = new LEGListaConPI<E>();
  ListaConPI<E> lpi = elArray[posTabla(x)];
  for (lpi.inicio(); !lpi.esFin(); lpi.siguiente())
    if (lpi.recuperar().equals(x)) res.insertar(lpi.recuperar());
  return res;
}
```

Recuperar Iguales en TablaHash (2/2)

- Solución al Apartado 2

```
public ListaConPI<V> recuperarValores(C c) {
  ListaConPI<EntradaDic<C,V>> lpi =
          th.recuperarIguales(new EntradaDic<C,V>(c));
  ListaConPI<V> res = new LEGListaConPI<V>();
  for (lpi.inicio(); !lpi.esFin(); lpi.siguiente())
    res.insertar(lpi.recuperar().valor);
  return res;
}
```


DireccionHTTP

```
public class DireccionHTTP {
private String servidor;  private int puerto;  private String direccion;
  public DireccionHTTP(String servidor, int puerto, String direccion){
    this.servidor = servidor;    this.puerto = puerto;    this.direccion = direccion;
  }
  public String getServidor(){ return this.servidor;}
  public String getPuerto(){ return this.puerto;}
  public String getDireccion(){ return this.direccion;}
  public boolean equals(Object o){
    DireccionHTTP d = (DireccionHTTP) o;
    return ( this.servidor.equals(d.servidor) &&
                this.puerto == d.puerto &&
                this.direccion.equals(d.direccion) );
  }
  public int hashCode(){return ( this.protocolo + this.servidor + this.direccion ).hashCode(); }}
```