Algoritmos `http://www.lfcia.org/alg/` 2007-2008

Ingeniería Informática

Ingeniería Técnica de Informática de Gestión

Práctica 5

Fecha límite de entrega: viernes, 14 de diciembre

Implementación de un Diccionario de Datos

Un *diccionario de datos* es un TDA que permite almacenar registros en base a una cierta clave, para posteriormente realizar búsquedas usando dicha clave (figuras 1, 2, y 3).

En esta práctica se pide construir un programa de consulta de sinónimos (usando los datos en `sinonimos.txt` con 19062 entradas), con el objetivo de comparar diferentes implementaciones del TDA Diccionario de Datos.

Las búsquedas se realizarán a partir de la palabra de cuyo signaficado se buscan sinónimos, y como funciones de dispersión (figuras 4 y 5) usaremos las dos siguientes:

```
función DispersiónA(Clave, Tamaño de la clave)
    valor:= ascii(Clave[1]);
    para j:= 2 hasta Tamaño de la llave hacer
        valor:= valor + ascii(Clave[j]);
    devolver valor mod N { Tamaño de la tabla }
fin función

función DispersiónB(Clave, Tamaño de la clave)
    valor := ascii (Clave[1]);
    para j:= 2 hasta Tamaño de la llave hacer
        valor := valor * 32 + ascii(Clave[j]);
    devolver valor mod N
fin función
```

Se debe considerar el uso de:

- *tablas de dispersión abiertas* (figura 6) con un factor de carga aproximado de $1,00$ ($N = 19069$)
- *tablas de dispersión cerradas* (figura 7):
  - con *exploración lineal* (figura 8), siendo el factor de carga aproximadamente $0,75$ ($N = 25463$),
  - con *exploración cuadrática*, siendo el factor de carga aproximadamente $0,5$ ($N = 38197$),

$$\frac{19062}{25463} = 0{,}7486 \qquad \frac{19062}{38197} = 0{,}4990$$

Se pide:

1. Implemente los diccionarios de datos validando su correcto funcionamiento.

2. Para cada uno de los diccionarios de datos y con cada una de las funciones de dispersión, indique el número de colisiones producidas durante la *inserción* de *todos* los usuarios con cuenta en la red de docencia (figuras 9 y 10).

   Asimismo, determine el tiempo de búsqueda de *todos* los usuarios en cada una de las situaciones anteriores.

3. Compare entre sí los resultados de los diccionarios de datos, y de las funciones de dispersión.

```
interface TablaDispersion {
    public int insertar(Item x);
    // Devuelve las colisiones producidas al insertar el item x.
    // Insertar un item "repetido" se convierte en una modificación.

    public Item buscar(String clave);
    // Cuando la búsqueda no tiene éxito devuelve null.

    public int dimension();
    // Devuelve el tamaño de la tabla

    public String listar();
}
```

Figura 1: La interfaz `TablaDispersión`

```
public interface Item {
    public String clave();
    public String datos();
    public boolean igualA(String clave);
}
```

Figura 2: La interfaz `Item`

```
public class Sinonimo implements Item {
    private String clave;
    private String sinonimos;

    public Sinonimo(String clave, String sinonimos) {
        this.clave = clave;
        this.sinonimos = sinonimos;
    }
    public String clave() {
        return clave;
    }
    public String datos() {
        return sinonimos;
    }
    public boolean igualA(String clave) {
        return (this.clave == clave);
    }
    public String toString() {
        String s = new String();
        s = "vocablo: " + clave + ", sinonimos: " + sinonimos;
        return s;
    }
}
```

Figura 3: La clase `Sinonimo`

```
public interface Dispersion {
    public int calcular(String clave, int tamTabla);
}
```

Figura 4: La interfaz `Dispersion`

```
public class DispersionA implements Dispersion {
    public int calcular(String clave, int tamTabla) {
        int valor = 0;
        for (int i = 0; i < clave.length(); i++)
            valor += clave.charAt(i);
        return (valor % tamTabla);
    }
    public String toString(){
        return "Dispersion A";
    }
}
```

Figura 5: La clase `DispersionA`

```
import java.util.*;
public class Abierta implements TablaDispersion {
    private Vector [] diccionario;
    private Dispersion disp;
    public Abierta(int tamTabla, Dispersion disp) {
        diccionario = new Vector [tamTabla];
        for (int i=0; i<diccionario.length; i++)
            diccionario[i] = new Vector();
        this.disp = disp;
    }
    public int insertar(Item x){
        int pos = disp.calcular(x.clave(), diccionario.length);
        int colisiones = 0;
        while (colisiones < diccionario[pos].size() &&
               !((Item)diccionario[pos].get(colisiones)).igualA(x.clave()))
            colisiones++;
        if (colisiones == diccionario[pos].size())
            diccionario[pos].add(x);
        else
            diccionario[pos].set(colisiones, x);
        return colisiones;
    }
    /*
    ...
    */
    public int dimension() {
        return diccionario.length;
    }
    public String listar() {
        String str = new String();
        str = str + "[";
        for (int i = 0; i < diccionario.length; i++) {
            str += "[";
            if (diccionario[i].size() > 0)
                str += ((Item)diccionario[i].get(0)).clave();
            for (int j= 1; j < diccionario[i].size(); j++) {
                str += ";" + ((Item)diccionario[i].get(j)).clave();
            }
            str += "]";
        }
        return (str + "]");
    }
    public String toString() {
        return ("Abierta con " + disp);
    }
}
```

Figura 6: La clase `Abierta`

```
public abstract class Cerrada implements TablaDispersion {
    private Item [] diccionario = null;
    protected Dispersion disp;
    public Cerrada(int tamTabla, Dispersion disp) {
        diccionario = new Item[tamTabla];
        this.disp = disp;
    }
    protected abstract int funResolucionColision(int posIni, int numIntento);
    public Item buscar(String clave) {
        int i=0;
        int posIni = disp.calcular(clave, diccionario.length);
        int posActual = posIni;
        while (diccionario[posActual] != null &&
               !diccionario[posActual].igualA(clave)) {
            i++;
            posActual = funResolucionColision(posIni, i) %
                diccionario.length;
        }
        return diccionario[posActual];
    }
    /*
      ...
    */
    public int dimension() {
        return diccionario.length;
    }
    public String listar() {
        String str = new String();
        str = str + "[";
        if (diccionario[0] != null)
            str += diccionario[0].clave();
        for (int i = 1; i < diccionario.length; i++)
            if (diccionario[i] != null)
                str = str + ";" + diccionario[i].clave();
            else
                str = str + ";";
        return (str + "]");
    }
}
```

Figura 7: La clase abstracta `Cerrada`

```
public class Lineal extends Cerrada {
    public Lineal(int tamTabla, Dispersion disp) {
        super(tamTabla, disp);
    }
    public int funResolucionColision(int posIni, int numIntento) {
        return (posIni + numIntento);
    }
    public String toString() {
        return ("Cerrada Lineal con " + disp);
    }
}
```

Figura 8: La clase `Lineal`

```
import java.util.*;
public class Jugar {
    public static void leerFichero(String nombreFichero, Vector v) {
        java.io.FileInputStream fichero;
        byte [] texto;
        String s;
        java.util.StringTokenizer st;
        String clave, def;
        try {
            fichero = new java.io.FileInputStream(nombreFichero);
            texto = new byte [fichero.available()];
            fichero.read(texto);
            fichero.close();
            s = new String(texto);
            st = new java.util.StringTokenizer(s, "\t\n");
            try {
                for (;;) {
                    clave=st.nextToken();
                    def=st.nextToken();
                    v.add(new Sinonimo(clave, def));
                }
            }
            catch (Exception e){
            };
        }
        catch (Exception e) {
            System.out.println(e);
        }
    }

    public static void main(String [] args) {
        if (args.length != 1) {
            System.out.println("Uso: java Jugar 'nombre del fichero'");
        }
        else {
            java.util.Vector v = new java.util.Vector ();
            leerFichero(args[0], v);
        /*
          ...
        */
        }
    }
```

Figura 9: Parte del programa principal con el código para la lectura del fichero `sinonimos.txt`

```
> java Jugar sinonimos.txt
                                      colisiones
Abierta con Dispersion A                  138203
Abierta con Dispersion B                   81202
Cerrada Lineal con Dispersion A         108947313
Cerrada Lineal con Dispersion B            310537
Cerrada Cuadratica con Dispersion A       1181411
Cerrada Cuadratica con Dispersion B         97093
```

Figura 10: Parte de la posible salida por pantalla de la ejecución del programa principal