Memoria de Prácticas

Estructura de Datos Multidimensionales

David Guillermo Morales Sáez

Índice

Introducción

En esta asignatura, se ha solicitado crear un programa que evalúe una serie de palabras posiblemente erróneas y busquemos la palabra más parecida en un diccionario. Para ello, se utilizarán dos distancias distintas, la distancia de Levenshtein y la distancia Invariante Trasposicional. Además, también se explorará el árbol multidimensional BK.

El desarrollo se ha hecho íntegramente en Java, un lenguaje de alto nivel orientado a objetos que ha facilitado bastante el desarrollo de los distintos algoritmos. Para cada algoritmo, se lee inicialmente tanto el fichero con las palabras erróneas como el fichero con el diccionario. El código inicial para esto se halla en el Anexo.

Primera Aproximación: Distancia de Levenshtein (DL)

La Distancia de Levenshtein, también conocida como la distancia de edición, cuyo algoritmo fue desarrollado por el matemático ruso Vladimir Levenshtein en 1965. Esta distancia indica el número mínimo de variaciones requeridas (inclusión, eliminación o edición) para transformar una ristra de caracteres en otra. Para mostrar su funcionamiento, veamos el cálculo de distancias entre 'HOGAR' y 'MOPA':

1. Inicialmente se ponen las palabras y se les da a cada letra un número ordenado, empezando por el 1:

		H	O	G	A	R
	0	1	2	3	4	5
M	1					
O	2					
P	3					
A	4					

2. Ahora, buscamos el mínimo en cada celda entre:
   - La celda superior más 1
   - La celda de la izquierda más 1
   - La celda diagonal y se le sumará uno si los caracteres son distintos

Con esto en mente, iteraremos para la primera fila:

		H	O	G	A	R
	0	1	2	3	4	5
M	1	1	2	3	4	5
O	2					
P	3					
A	4					

3. Una vez hecho esto, avanzamos a la siguiente fila, y así hasta completar la matriz:

		H	O	G	A	R
	0	1	2	3	4	5
M	1	1	2	3	4	5
O	2	2	1	3	4	5
P	3	3	3	2	4	5
A	4	4	4	3	2	3

4. Cuando hayamos completado la matriz, miraremos el último valor y ahí tendremos la distancia de Levenshtein. En este caso es 3, y lo podemos comprobar ya que si cambiamos la 'H' por una 'M', la 'G' por una 'P' y si quitamos la 'R', pasamos de tener 'HOGAR' a tener 'MOPA'.

El código de desarrollo de la búsqueda con la distancia de Levenshtein está en el Anexo.

Segunda Aproximación: Distancia Invariante Transposicional (DIT)

El problema que hay con la Distancia de Levenshtein es que es muy costoso computacionalmente, por lo que se buscó una forma alternativa para obtener la distancia entre dos palabras de forma menos costosa. Una de estas alternativas es la Distancia Invariante Transposicional, donde se buscan las repeticiones de caracteres entre las dos ristras. Para verlo mejor, veámoslo con un ejemplo:

Si partimos de las palabras 'HOGAR' y 'MOPA', iremos comprobando poco a poco las repeticiones que hay:

H: no se repite en ningún momento en 'MOPA'
O: se repite una vez en 'MOPA', sumamos 1 a la distancia
G: no se repite en ningún momento en 'MOPA'
A: se repite una vez en 'MOPA', sumamos 1 a la distancia
R: no se repite en ningún momento en 'MOPA'

Ahora, miraremos los que hay con 'MOPA' hacia 'HOGAR'

M: no se repite en ningún momento en ‘HOGAR’
O: se repite una vez en ‘HOGAR’, restamos 1 a la distancia
P: no se repite en ningún momento en ‘HOGAR’
A: se repite una vez en ‘HOGAR’, restamos 1 a la distancia

A la distancia obtenida (0), le sumamos la diferencia de tamaños en valor absoluto, por lo que la distancia total sería 1.

Como podemos ver, es mucho menos fiable que la distancia DL, pero lo podemos utilizar para ver cuándo calcular la distancia DL y cuando no, ya que DIT <= 2*DL.

El código de desarrollo de la búsqueda con la distancia de Levenshtein está en el Anexo.

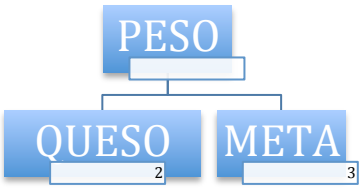
Tercera Aproximación: Árbol Buckhard-Keller con DL

El árbol Buckhard-Keller (BK) fue diseñado por Walter Austin Burkhard y Robert M. Keller. Este árbol parte de un nodo raíz cualquiera y tiene K hijos. La posición en la que está ese hijo es la distancia entre el hijo y el padre. Si una posición ya estuviese ocupada, se insertaría en el hijo, calculando la distancia entre el nuevo nodo y el hijo. Para calcular las distancias entre dos elementos, se usa la distancia de Levenshtein, por lo que se obtienen resultados realmente precisos.

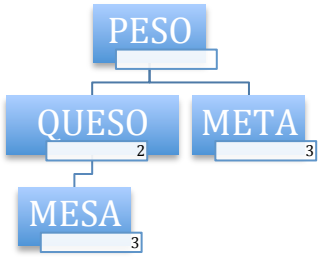
Este tipo de estructura facilita altamente el almacenamiento de diccionarios o elementos similares, ya que la navegación dentro del mismo es bastante sencilla. Para ver su funcionamiento, veamos un ejemplo:

Partamos de un árbol con un único nodo, el raíz, que almacena la palabra ‘PESO’, y deseamos almacenar la palabra ‘QUESO’, por lo que, si calculamos entre ambos la distancia, vemos que es de 2:

Ahora, vamos a insertar la palabra ‘META’, cuya distancia con ‘PESO’ es de 3:



Y ahora, intentemos meter 'MESA', cuya distancia con 'PESO' es de 2 … pero tenemos un problema y es que el nodo 2 está ya ocupado. Pues lo que hacemos es ver la distancia entre 'MESA' y 'QUESO', que es 3:



Y así sucesivamente. El código de este apartado puede verse en el Anexo.

Cuarta Aproximación: Árbol Buckhard-Keller con DIT

Al igual que lo sucedido al principio, si utilizamos la distancia DL de forma indiscriminada en el árbol BK, el coste computacional es extremadamente alto. Para poder solucionar esto, se utiliza la distancia DIT de forma auxiliar. Para ello, las distancias entre nodos se calculan con la distancia DIT y, en la búsqueda del elemento o elementos, comprobamos si la distancia entre las dos palabras es menor o igual a la distancia DL previa y, si es así, se calcula la distancia DL para ver si realmente es más cercana a la palabra o no.

Este cambio tiene un inconveniente, ya que reduce la precisión y se incrementan los fallos. El código de este apartado se encuentra en el Anexo.

Pruebas

Para el testeo y la demostración del funcionamiento y rendimiento de estos algoritmos, se ha utilizado el fichero con las palabras cedido por el profesor y se ha implementado un algoritmo que distorsiona cada palabra, incluyendo, eliminando o modificando un carácter de cada palabra. Con estos dos ficheros, se han probado los cuatro algoritmos, obteniendo los siguientes resultados:

Nombre	Tiempo	Aciertos
DL	126	10331
DIT	118	10331
BK DL	65	10331
BK DIT	30	9446

Como es lógico, en cada avance se ha reducido el tiempo de resolución, pasando de más de dos minutos en una búsqueda completa con la distancia DL a medio minuto usando un árbol BK y navegando con la distancia DIT. Por otro lado, se puede percibir que el número de aciertos se ha visto decrementado en el último caso, debido al uso de la distancia DIT en la navegación.

Anexo

Inicio

Practicas_EDM.java

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.ArrayList;
import java.util.List;

import org.junit.Test;


public class Practicas_EDM
{
      public static void main(String [] args)
      {
        List<String> palabras_correctas = new ArrayList<String>();
        List<String> palabras_erroneas = new ArrayList<String>();
          try
          {
               BufferedReader reader = new BufferedReader(new
FileReader("/Users/david/Dropbox/Universidad/EDM/Practicas/DIC10454.txt"));
               String line;
               while ((line = reader.readLine()) != null)
               {
                      palabras_correctas.add(line);
               }
               reader.close();

               reader = new BufferedReader(new
FileReader("/Users/david/Dropbox/Universidad/EDM/Practicas/FicheroErrores.txt"));
               line = null;
               while ((line = reader.readLine()) != null)
               {
                      palabras_erroneas.add(line);
               }
               reader.close();
               }
               catch (Exception e)
               {
               e.printStackTrace();
               return ;
               }

          long[][] valores = new long[4][2];
          valores[0] = (new
CalculoDistanciaDL()).Comprueba_Palabras(palabras_correctas, palabras_erroneas);
          valores[1] = (new
CalculoDistanciaDIT()).Comprueba_Palabras(palabras_correctas, palabras_erroneas);
          valores[2] = (new
CalculoDistanciaBKDL()).Comprueba_Palabras(palabras_correctas, palabras_erroneas);
          valores[3] = (new
CalculoDistanciaBKDIT()).Comprueba_Palabras(palabras_correctas, palabras_erroneas);
          System.out.println(" Nombre Aciertos Tiempo");
          System.out.println(" DL " + valores[0][1] + " " + valores[0][0]);
          System.out.println(" DIT " + valores[1][1] + " " + valores[1][0]);
          System.out.println(" BK DL " + valores[2][1] + " " + valores[2][0]);
          System.out.println(" BK DIT " + valores[3][1] + " " + valores[3][0]);
      }
}
```

Distancia de Levenshtein

CálculoDistanciaDL.java

```
import java.util.*;
import java.io.*;

public class CalculoDistanciaDL
{

      private static int minimum(int a, int b, int c)
      {
        if (a<=b && a<=c)
        {
            return a;
        }
        if (b<=a && b<=c)
        {
            return b;
        }
        return c;
    }

      public long[] Comprueba_Palabras(List<String> palabras_correctas, List<String>
palabras_erroneas)
      {
       try
              {
              int numCorrectas = 0;
              FileWriter ficheroFinal = new FileWriter("FicheroSolucionDL.txt");
                     BufferedWriter out = new BufferedWriter(ficheroFinal);
              long startTime = System.nanoTime();
              for(int i=0; i<palabras_erroneas.size(); i++)
              {
                     List<String> palabras_posibles = new ArrayList<String>();
                     int DL = 99999;
                     // Comprobamos la distancia de cada palabra la erronea
                     for(int j=0; j<palabras_correctas.size(); j++)
                     {
                            char [] str1 = palabras_correctas.get(j).toCharArray();
                            char [] str2 = palabras_erroneas.get(i).toCharArray();
                            int [][]distance = new
int[str1.length+1][str2.length+1];

                      for(int m=0; m<=str1.length; m++)
                      {
                             distance[m][0] = m;
                      }
                      for(int n=0; n<=str2.length; n++)
                      {
                             distance[0][n]=n;
                      }
                      for(int m=1; m<=str1.length; m++)
                      {
                          for(int n=1; n<=str2.length; n++)
                          {
                                distance[m][n]= minimum(distance[m-1][n]+1,
                                                        distance[m][n-1]+1,
                                                        distance[m-1][n-1]+
                                                        ((str1[m-1]==str2[n-1])?0:1));
                          }
                      }
                      // Si la palabra tiene la misma distancia que la actual, la
añadimos a la lista
                      if(distance[str1.length][str2.length] == DL)
                      {
                             palabras_posibles.add(palabras_correctas.get(j));
                      }
                      // Si la palabra tiene menor distancia que la actual, limpiamos
la lista, añadimos la palabra y reducimos la DL
                      if(distance[str1.length][str2.length] < DL)
```

```
                        {
                                DL = distance[str1.length][str2.length];
                                palabras_posibles.clear();
                                palabras_posibles.add(palabras_correctas.get(j));
                        }
                        }
                        // Almacenamos el resultado en el fichero
                                out.write(palabras_erroneas.get(i) + " : ");
                                boolean correcto = false;
                                for(int k=0; k<palabras_posibles.size()-1; k++)
                                {
                                        out.write(palabras_posibles.get(k) + ", ");

      if(palabras_correctas.indexOf(palabras_posibles.get(k))==i)
                                                correcto = true;
                                }

      out.write(palabras_posibles.get(palabras_posibles.size()-1) + " : DL = " + DL);

      if(palabras_correctas.indexOf(palabras_posibles.get(palabras_posibles.size()-
1))==i)
                                                correcto = true;
                                if(correcto)
                                {
                                        out.write(" --> Correcto = OK");
                                numCorrectas++;
                                }
                                else
                                        out.write(" --> Correcto = NO");
                                out.newLine();
                }
                out.write("Número de palabras correctas: "+numCorrectas);
                out.newLine();
                long endTime = System.nanoTime();
                out.write("Tiempo de ejecución: " + (endTime - startTime)/1000000000);
                out.close();
                long[] retorno = new long[2];
                retorno[0] = numCorrectas;
                retorno[1] = (endTime - startTime)/1000000000;
                return retorno;
                }
                catch (Exception e)
        {//Catch exception if any
                System.err.println("Error: " + e.getMessage());
        }
        return null;
      }
}
```

CalculoDistanciaDIT.java

```
import java.util.*;
import java.io.*;

public class CalculoDistanciaDIT
{
      private static int minimum(int a, int b, int c)
      {
        if (a<=b && a<=c)
        {
            return a;
        }
        if (b<=a && b<=c)
        {
            return b;
        }
        return c;
    }

      private static int DistanciaDL(String ristra1, String ristra2)
      {
        char [] str1 = ristra1.toCharArray();
```

```
        char [] str2 = ristra2.toCharArray();
        int [][]distance = new int[str1.length+1][str2.length+1];

        for(int m=0; m<=str1.length; m++)
        {
                distance[m][0] = m;
        }
        for(int n=0; n<=str2.length; n++)
        {
                distance[0][n]=n;
        }
        for(int m=1; m<=str1.length; m++)
        {
                for(int n=1; n<=str2.length; n++)
                {
                        distance[m][n]= minimum(distance[m-1][n]+1,
                                                distance[m][n-1]+1,
                                                distance[m-1][n-1]+
                                                        ((str1[m-1]==str2[n-1])?0:1));
                }
        }
        return distance[str1.length][str2.length];
      }

      private static int DistanciaDIM(String ristra1, String ristra2)
      {
        char [] str1 = ristra1.toCharArray();
        char [] str2 = ristra2.toCharArray();
        char [] letras = "abcdefghijklmnñopqrstuvwxyz".toCharArray();
        int sumador = 0;
        for (int i=0; i<27; i++)
        {
                int contador = 0;
                for (int j=0; j<str1.length; j++)
                {
                        if (str1[j]==letras[i])
                        {
                                contador++;
                        }
                }
                for (int j=0; j<str2.length; j++)
                {
                        if (str2[j]==letras[i])
                        {
                                contador--;
                        }
                }
                sumador+= Math.abs(contador);
        }
        return (sumador+Math.abs(str1.length-str2.length));
      }

      public long[] Comprueba_Palabras(List<String> palabras_correctas, List<String>
palabras_erroneas)
      {
        try
                {
                FileWriter ficheroFinal = new FileWriter("FicheroSolucionDIT.txt");
                        BufferedWriter out = new BufferedWriter(ficheroFinal);
                long startTime = System.nanoTime();
                int correctas = 0;
                for(int i=0; i<palabras_erroneas.size(); i++)
                {
                        List<String> palabras_posibles = new ArrayList<String>();
                        int DL = 99999;
                        // Comprobamos la distancia de cada palabra la erronea
                        for(int j=0; j<palabras_correctas.size(); j++)
                        {
                                int DIM = DistanciaDIM(palabras_correctas.get(j),
palabras_erroneas.get(i));
                                if (DIM <= DL*2)
                                {
                                        int DLtemp =
DistanciaDL(palabras_correctas.get(j), palabras_erroneas.get(i));
```

```
                                        // Si la palabra tiene la misma distancia que la
actual, la añadimos a la lista
                                        if(DLtemp == DL)
                                        {

      palabras_posibles.add(palabras_correctas.get(j));
                                        }
                                        // Si la palabra tiene menor distancia que la
actual, limpiamos la lista, añadimos la palabra y reducimos la DL
                                        if(DLtemp < DL)
                                        {
                                                DL = DLtemp;
                                                palabras_posibles.clear();

      palabras_posibles.add(palabras_correctas.get(j));
                                        }
                                }
                        }
                        // Almacenamos el resultado en el fichero
                                out.write(palabras_erroneas.get(i) + " : ");
                                boolean correcto = false;
                                for(int k=0; k<palabras_posibles.size()-1; k++)
                                {
                                        out.write(palabras_posibles.get(k) + ", ");

      if(palabras_correctas.indexOf(palabras_posibles.get(k))==i)
                                                correcto = true;
                                }

      out.write(palabras_posibles.get(palabras_posibles.size()-1) + " : DL = " + DL);
                        if(correcto ||
palabras_correctas.indexOf(palabras_posibles.get(palabras_posibles.size()-1))==i)
                        {
                                out.write(" --> Correcto = OK");
                                correctas++;
                        }
                                else
                                        out.write(" --> Correcto = NO");
                                out.newLine();
                }
                out.write("Número de palabras correctas: "+correctas);
                out.newLine();
                long endTime = System.nanoTime();
                out.write("Tiempo de ejecución: " + (endTime -
startTime)/Math.pow(10.0,9.0));
                out.close();
                long[] retorno = new long[2];
                retorno[0] = correctas;
                retorno[1] = (endTime - startTime)/1000000000;
                return retorno;
                }
                catch (Exception e)
        {//Catch exception if any
                System.err.println("Error: " + e.getMessage());
        }
        return null;
      }
}
```

Árbol BK con Distancia de Levenshtein

BKNodeDL.java

```
import static java.lang.Math.max;
import java.util.*;

public class BKNodeDL
{
	final String palabra;
	final Map<Integer, BKNodeDL> hijos = new HashMap<Integer, BKNodeDL>();

	public BKNodeDL(String palabro)
	{
	  this.palabra = palabro;
	}

	protected BKNodeDL HijoADistancia(int distancia)
	{
	  return hijos.get(distancia);
	}

	public void insertaHijo(int distancia, BKNodeDL hijo)
	{
	  hijos.put(distancia, hijo);
	}

	public List<String> busqueda(String palabra, int distanciaMaxima)
	{
	  int distancia, distanciaActual = distanciaMaxima;;
	  List<String> coincidencias = new LinkedList<String>();
	  if (hijos.size() == 0)
	  {
		distancia = distancia(this.palabra, palabra);
		if (distancia <= distanciaMaxima)
			coincidencias.add(this.palabra);
		return coincidencias;
	  }
	  else
	  {
		distancia = distancia(this.palabra, palabra);
		if (distancia <= distanciaMaxima)
		{
			coincidencias.add(this.palabra);
			distanciaActual = distancia;
		}
	  }
	  int i = max(1, distancia - distanciaActual);
	  for(; i <= distancia + distanciaActual; i++)
	  {
		BKNodeDL hijo = hijos.get(i);
		if (hijo == null)
			continue;
		List<String> coincidenciasHijo = hijo.busqueda(palabra,
distanciaActual);
		if (coincidenciasHijo.size() > 0)
		{
			int distaux = distancia(palabra, coincidenciasHijo.get(0));
			if (distaux<distanciaActual)
			{
				distanciaActual = distaux;
				coincidencias.clear();
			}
			coincidencias.addAll(coincidenciasHijo);
		}
	  }
	  return coincidencias;
	}

	public int distancia(String    palabraA, String palabraB)
```

```
	{
	  char [] str1 = palabraA.toCharArray();
	  char [] str2 = palabraB.toCharArray();
	  int [][]distance = new int[str1.length+1][str2.length+1];

	  for(int m=0; m<=str1.length; m++)
	  {
		distance[m][0] = m;
	  }
	  for(int n=0; n<=str2.length; n++)
	  {
		distance[0][n]=n;
	  }
	  for(int m=1; m<=str1.length; m++)
	  {
		for(int n=1; n<=str2.length; n++)
		{
			distance[m][n]= minimo(distance[m-1][n]+1,
									distance[m][n-1]+1,
									distance[m-1][n-1]+
									((str1[m-1]==str2[n-
1])?0:1));
		}
	  }
	  return distance[str1.length][str2.length];
	}

	private int minimo(int a, int b, int c)
	{
	  if (a<=b && a<=c)
	  {
	      return a;
	  }
	  if (b<=a && b<=c)
	  {
	      return b;
	  }
	  return c;
	}
}
```

BKTreeDL.java

```
import java.util.*;

public class BKTreeDL
{
	private BKNodeDL raiz;

	// Devuelve las palabras más cercanas a la solicitada
	public List<String> busqueda(String palabra)
	{
	  return raiz.busqueda(palabra, 9999);
	}

	// Inserta una palabra en el árbol
	public void insertar(String palabra)
	{
	  if (palabra == null || palabra.isEmpty())
		return;
	  BKNodeDL nodo = new BKNodeDL(palabra);
	  if (raiz == null)
	  {
		raiz = nodo;
	  }
	  insertarRecursivo(raiz, nodo);
	}

	// Función interna para insertar un nodo en el árbol
	private void insertarRecursivo(BKNodeDL padre, BKNodeDL nuevo)
	{
```

```
        if (padre.equals(nuevo))
        {
                return;
        }
        int distancia = distancia(padre.palabra, nuevo.palabra);
        BKNodeDL nodo = padre.HijoADistancia(distancia);
        if (nodo == null)
        {
                padre.insertaHijo(distancia, nuevo);
        }
        else
        {
                insertarRecursivo(nodo, nuevo);
        }
      }

      // Función que calcula la distancia entre dos palabras
      public int distancia(String     palabraA, String palabraB)
      {
        char [] str1 = palabraA.toCharArray();
        char [] str2 = palabraB.toCharArray();
        int [][]distance = new int[str1.length+1][str2.length+1];

        for(int m=0; m<=str1.length; m++)
        {
                distance[m][0] = m;
        }
        for(int n=0; n<=str2.length; n++)
        {
                distance[0][n]=n;
        }
        for(int m=1; m<=str1.length; m++)
        {
                for(int n=1; n<=str2.length; n++)
                {
                        distance[m][n]= minimo(distance[m-1][n]+1,
                                               distance[m][n-1]+1,
                                               distance[m-1][n-1]+
                                                       ((str1[m-1]==str2[n-1])?0:1));
                }
        }
        return distance[str1.length][str2.length];
      }

      private int minimo(int a, int b, int c)
      {
        if (a<=b && a<=c)
        {
            return a;
        }
        if (b<=a && b<=c)
        {
            return b;
        }
        return c;
      }
}
```

CalculoDistanciaBKDL.java

```
import java.util.*;
import java.io.*;

public class CalculoDistanciaBKDL
{

      public long[] Comprueba_Palabras(List<String> palabras_correctas, List<String>
palabras_erroneas)
      {
        BKTreeDL arbol = new BKTreeDL();
        for(int i=0; i<palabras_correctas.size(); i++)
```

```
                arbol.insertar(palabras_correctas.get(i));
        try
        {
                int num_Correctas = 0;
            FileWriter ficheroFinal = new FileWriter("FicheroSolucionBKDL.txt");
            BufferedWriter out = new BufferedWriter(ficheroFinal);
            long startTime = System.nanoTime();
            for(int i=0; i<palabras_erroneas.size(); i++)
            {
                List<String> palabras_posibles =
arbol.busqueda(palabras_erroneas.get(i));
                        if (palabras_posibles.size() == 0)
                                continue;
                out.write(palabras_erroneas.get(i) + " : ");
                boolean correcto = false;
                for(int k=0; k<palabras_posibles.size()-1; k++)
                {
                    out.write(palabras_posibles.get(k) + ", ");
                    if(palabras_correctas.indexOf(palabras_posibles.get(k))==i)
                        correcto = true;
                }
                out.write(palabras_posibles.get(palabras_posibles.size()-1));

if(palabras_correctas.indexOf(palabras_posibles.get(palabras_posibles.size()-1))==i)
                        correcto = true;
                if(correcto)
                       {
                    out.write(" --> Correcto = OK");
                            num_Correctas++;
                      }
                else
                    out.write(" --> Correcto = NO");
                out.newLine();
            }
               out.write("Número de palabras correctas: " + num_Correctas);
               out.newLine();
               double endTime = Double.valueOf(System.nanoTime() - startTime);
               endTime /= 1000000000;
               out.write("Tiempo de ejecución: " + endTime);
               out.newLine();
               out.close();
               long[] retorno = new long[2];
               retorno[0] = num_Correctas;
               retorno[1] = (long) (endTime);
               return retorno;
        }
        catch (Exception e)
        {
            System.err.println("Error: " + e.getMessage());
        }
       return null;
      }
}
```

Árbol BK con Distancia DIT

BKNodeDIT.java

```
import static java.lang.Math.max;
import java.util.*;

public class BKNodeDIT
{
	final String palabra;
	final Map<Integer, BKNodeDIT> hijos = new HashMap<Integer, BKNodeDIT>();

	public BKNodeDIT(String palabro)
	{
	  this.palabra = palabro;
	}

	protected BKNodeDIT HijoADistancia(int distancia)
	{
	  return hijos.get(distancia);
	}

	public void insertaHijo(int distancia, BKNodeDIT hijo)
	{
	  hijos.put(distancia, hijo);
	}

	public List<String> busqueda(String palabra, int distanciaMaxima)
	{
	  int distanciaDIM = DistanciaDIM(this.palabra, palabra);
	  int distancia, distMax = distanciaMaxima, distanciaActual = distanciaMaxima;
	  List<String> coincidencias = new LinkedList<String>();
	  if (hijos.size() == 0)
	  {
		if(distanciaDIM <= distanciaActual)
		{
			distancia = distancia(this.palabra, palabra);
			if (distancia <= distMax)
				coincidencias.add(this.palabra);
		}
		return coincidencias;
	  }
	  else
	  {
		if(distanciaDIM <= distanciaActual)
		{
			distancia = distancia(this.palabra, palabra);
			if (distancia <= distMax)
			{
				coincidencias.add(this.palabra);
				distMax = distancia;
			}
			distanciaActual = distanciaDIM;
		}
	  }
	  int i = max(1, distanciaDIM - distanciaActual);
	  for(; i <= distanciaDIM + distanciaActual; i++)
	  {
		BKNodeDIT hijo = hijos.get(i);
		if (hijo == null)
			continue;
		List<String> coincidenciasHijo = hijo.busqueda(palabra,
distanciaActual);
		if (coincidenciasHijo.size() > 0)
		{
			int distaux = DistanciaDIM(palabra, coincidenciasHijo.get(0));
			if (distaux < distanciaActual)
			{
				int distaux2 = distancia(palabra,
coincidenciasHijo.get(0));
```

```
                                if (distaux2<distMax)
                                {
                                        distMax = distaux2;
                                        coincidencias.clear();
                                }
                                distanciaActual = distaux;
                        }
                        coincidencias.addAll(coincidenciasHijo);
                }
          }
          return coincidencias;
        }

        public int distancia(String     palabraA, String palabraB)
        {
          char [] str1 = palabraA.toCharArray();
          char [] str2 = palabraB.toCharArray();
          int [][]distance = new int[str1.length+1][str2.length+1];

          for(int m=0; m<=str1.length; m++)
          {
                  distance[m][0] = m;
          }
          for(int n=0; n<=str2.length; n++)
          {
                  distance[0][n]=n;
          }
          for(int m=1; m<=str1.length; m++)
          {
                  for(int n=1; n<=str2.length; n++)
                  {
                          distance[m][n]= minimo(distance[m-1][n]+1,
                                                                  distance[m][n-1]+1,
                                                                  distance[m-1][n-1]+
                                                                  ((str1[m-1]==str2[n-
1])?0:1));
                  }
          }
          return distance[str1.length][str2.length];
        }

        private int minimo(int a, int b, int c)
        {
          if (a<=b && a<=c)
          {
              return a;
          }
          if (b<=a && b<=c)
          {
              return b;
          }
          return c;
        }

        private static int DistanciaDIM(String ristra1, String ristra2)
        {
          char [] str1 = ristra1.toCharArray();
          char [] str2 = ristra2.toCharArray();
          char [] letras = "abcdefghijklmnñopqrstuvwxyz".toCharArray();
          int sumador = 0;
          for (int i=0; i<27; i++)
          {
                  int contador = 0;
                  for (int j=0; j<str1.length; j++)
                  {
                          if (str1[j]==letras[i])
                          {
                                  contador++;
                          }
                  }
                  for (int j=0; j<str2.length; j++)
                  {
                          if (str2[j]==letras[i])
                          {
```

```
                        contador--;
                    }
                }
                sumador+= Math.abs(contador);
            }
            return (sumador+Math.abs(str1.length-str2.length));
        }
}
```

BKTreeDIT.java

```
import java.util.*;

public class BKTreeDIT
{
    private BKNodeDIT raiz;

    // Devuelve las palabras más cercanas a la solicitada
    public List<String> busqueda(String palabra)
    {
      return raiz.busqueda(palabra, 9999);
    }

    // Inserta una palabra en el árbol
    public void insertar(String palabra)
    {
      if (palabra == null || palabra.isEmpty())
            return;
      BKNodeDIT nodo = new BKNodeDIT(palabra);
      if (raiz == null)
      {
            raiz = nodo;
      }
      insertarRecursivo(raiz, nodo);
    }

    // Función interna para insertar un nodo en el árbol
    private void insertarRecursivo(BKNodeDIT padre, BKNodeDIT nuevo)
    {
      if (padre.equals(nuevo))
      {
            return;
      }
      int distancia = distancia(padre.palabra, nuevo.palabra);
      BKNodeDIT nodo = padre.HijoADistancia(distancia);
      if (nodo == null)
      {
            padre.insertaHijo(distancia, nuevo);
      }
      else
      {
            insertarRecursivo(nodo, nuevo);
      }
    }

    // Función que calcula la distancia entre dos palabras
    public int distancia(String    palabraA, String palabraB)
    {
      char [] str1 = palabraA.toCharArray();
      char [] str2 = palabraB.toCharArray();
      char [] letras = "abcdefghijklmnñopqrstuvwxyz".toCharArray();
      int sumador = 0;
      for (int i=0; i<27; i++)
      {
            int contador = 0;
            for (int j=0; j<str1.length; j++)
            {
                  if (str1[j]==letras[i])
                  {
                        contador++;
                  }
```

```
                }
                for (int j=0; j<str2.length; j++)
                {
                        if (str2[j]==letras[i])
                        {
                                contador--;
                        }
                }
                sumador+= Math.abs(contador);
        }
        return (sumador+Math.abs(str1.length-str2.length));
      }
}
```

CalculoDistanciaBKDIT.java

```
import java.util.*;
import java.io.*;

public class CalculoDistanciaBKDIT
{
      public long[] Comprueba_Palabras(List<String> palabras_correctas, List<String>
palabras_erroneas)
      {
        BKTreeDIT arbol = new BKTreeDIT();
        for(int i=0; i<palabras_correctas.size(); i++)
                arbol.insertar(palabras_correctas.get(i));
        try
         {
                int num_Correctas = 0;
                long startTime = System.nanoTime();
            FileWriter ficheroFinal = new FileWriter("FicheroSolucionBKDIT.txt");
            BufferedWriter out = new BufferedWriter(ficheroFinal);
            for(int i=0; i<palabras_erroneas.size(); i++)
            {
                List<String> palabras_posibles =
arbol.busqueda(palabras_erroneas.get(i));
                        if (palabras_posibles.size() == 0)
                                continue;
                out.write(palabras_erroneas.get(i) + " : ");
                boolean correcto = false;
                for(int k=0; k<palabras_posibles.size()-1; k++)
                {
                    out.write(palabras_posibles.get(k) + ", ");
                    if(palabras_correctas.indexOf(palabras_posibles.get(k))==i)
                        correcto = true;
                }
                out.write(palabras_posibles.get(palabras_posibles.size()-1));

if(palabras_correctas.indexOf(palabras_posibles.get(palabras_posibles.size()-1))==i)
                        correcto = true;
                if(correcto)
                        {
                    out.write(" --> Correcto = OK");
                                num_Correctas++;
                        }
                else
                    out.write(" --> Correcto = NO");
                out.newLine();
            }
                out.write("Número de palabras correctas: " + num_Correctas);
                out.newLine();
                double endTime = Double.valueOf(System.nanoTime() - startTime);
                endTime /= 1000000000;
                out.write("Tiempo de ejecución: " + endTime);
                out.newLine();
                out.close();
                long[] retorno = new long[2];
                retorno[0] = num_Correctas;
                retorno[1] = (long) (endTime);
                return retorno;
```

```
        }
        catch (Exception e)
        {
            System.err.println("Error: " + e.getMessage());
        }
        return null;
    }
}
```

Pruebas

GeneracionRuido.java

```
import java.lang.System.*;
import java.util.*;
import java.io.*;

public class GeneracionRuido
{
      public static void main(String [] args)
      {
        // Comprobamos el número de parámetros
        if(args.length != 2)
        {
                System.out.println("\nInserte la ruta del fichero de diccionario o del
fichero final. \n");
                return;
        }
        String letras = "abcdefghijklmnñopqrstuvwxyz";
        // Obtenemos la lista de palabras
        List<String> palabras_correctas = new ArrayList<String>();
        System.out.println("Iniciando lectura del diccionario ...");
          try
          {
                BufferedReader reader = new BufferedReader(new FileReader(args[0]));
                String line;
                while ((line = reader.readLine()) != null)
                {
                        palabras_correctas.add(line);
                }
                reader.close();
                }
                catch (Exception e)
                {
                System.err.format("Error en la lectura de '%s'.", args[0]);
                e.printStackTrace();
                return ;
                }
        System.out.println("Iniciando generación de ruido ...");
                try
                {
                        System.out.println("Definicion de variables ...");
                // Recorremos la lista completa y añadimos un error
                FileWriter ficheroErrores = new FileWriter(args[1]);
                        BufferedWriter out = new BufferedWriter(ficheroErrores);
                Random rand = new Random();
                        System.out.println("Inicio del Bucle ...");
                for(int i=0; i<palabras_correctas.size(); i++)
                {
                        System.out.println(i);
                        int m = rand.nextInt(3);
                        int n = rand.nextInt(palabras_correctas.get(i).length());
                        int l = rand.nextInt(27);
                        String palabra_mal = "";
                        switch (m)
                        {
                                // Sustitución
                                case 0:
                                        palabra_mal =
palabras_correctas.get(i).substring(0, n) + letras.charAt(l) +
palabras_correctas.get(i).substring(n+1);
                                        break;
                                // Inserción
                                case 1:
                                        palabra_mal =
palabras_correctas.get(i).substring(0, n) + letras.charAt(l) +
palabras_correctas.get(i).substring(n);
                                        break;
                                // Eliminación
                                case 2:
```

```
                                        palabra_mal =
palabras_correctas.get(i).substring(0, n) + palabras_correctas.get(i).substring(n+1);
                                        break;
                        }
                                out.write(palabra_mal);
                                out.newLine();
                }
                out.close();
        }
        catch (Exception e)
        {//Catch exception if any
                        System.err.println("Error: " + e.getMessage());
        }
    }
}
```