Práctica Evaluable 3

Análisis y Diseño de Algoritmos

Nombre: Emilio Gómez Esteban

Grupo: 2ºD (Grado Matemáticas + Ingeniería Informática)

CÓDIGO EMPLEADO (CLASE SCMC):

```
import java.util.Random;

public class SCMC {

	private String r, t; // Las dos cadenas de la instancia
	private String sigma; // El alfabeto de la instancia
	private int m[][]; // la matriz para resolucion por Prog. Dinamica
	private static Random rnd = new Random(); // generador de aleatorio
	private static Direccion[][] b;

	/**
	 * Crea una instancia del problema
	 * @param sigma : alfabeto para la instancia
	 * @param r     : primera cadena
	 * @param t     : segunda cadena
	 */
	public SCMC(String sigma, String r, String t) {
		this.r = r;
		this.t = t;
		this.sigma = sigma;
		m = new int[1+r.length()][1+t.length()];
		b = new Direccion[r.length()+1][t.length()+1];
	}

	/**
	 * Crea una instancia aleatoria del problema
	 * @param longMax  : longitud maxima de las cadenas
	 * @param tamSigma : tamaño del alfabeto
	 */
	public SCMC(int longMax, int tamSigma) {
		this.sigma = Utils.alfabeto(tamSigma);
		r = Utils.cadenaAleatoria(rnd.nextInt(1+longMax),sigma);
		t = Utils.cadenaAleatoria(rnd.nextInt(1+longMax),sigma);
		m = new int[1+r.length()][1+t.length()];
		b = new Direccion[r.length()+1][t.length()+1];
	}

	public String r(){
		return r;
	}

	public String t(){
```

```
		return t;
	}

	public String sigma(){
		return sigma;
	}

	public int m(int i, int j){
		if (i<m.length && j<m[0].length) {
			return m[i][j];
		} else {
			return -1;
		}
	}


	/**
	 * Soluciona la instancia por Prog Dinamica, es decir, rellena
	 * la tabla @m
	 */
	public void solucionaPD(){
		// A Implementar por el alumno
		m[0][0] = 0;
		b[0][0] = Direccion.ARRIBA;
		for(int i=1; i<r.length()+1; i++) {
			m[i][0] = i;
			b[i][0] = Direccion.ARRIBA;
		}

		for(int j=0; j<t.length()+1; j++) {
			m[0][j] = j;
			b[0][j]=Direccion.IZDA;
		}

		for(int i=1; i<r.length()+1; i++) {
			for(int j=1; j<t.length()+1; j++) {
				if (r.charAt(i-1)==t.charAt(j-1)) {
					m[i][j] = 1 + m[i-1][j-1];
					b[i-1][j-1] = Direccion.DIAG;
				} else if (m[i-1][j] >= m[i][j-1]){
					m[i][j] = 1 + m[i][j-1];
					b[i-1][j-1] = Direccion.IZDA;
				} else {
					m[i][j] = 1 + m[i-1][j];
					b[i-1][j-1] = Direccion.ARRIBA;
				}
			}
		}
	}

	public enum Direccion{
		ARRIBA,IZDA,DIAG;
	}

	/**
	 * @return : devuelve la longitud de la solucion
	 *           a la instancia, es decir, la longitud
	 *           de la supersecuencia comun más corta de @r y @t
	 *           a partir de la tabla obtenida por Prog Dinamica
```

```
	 */
	public int longitudDeSolucionPD(){
		// A Implementar por el alumno
		return m[r.length()][t.length()];
	}

	/**
	 * @return Devuelve una solucion optima de la instancia, es decir
	 *         una supersecuencia comun mas corta de @r y @t
	 */
	public String unaSolucionPD(){
		// A Implementar por el alumno
		return construirSCMC(r, r.length(), t, t.length());
	}

	public static String construirSCMC(String r, int i, String t, int j) {
		if(i==0 && j==0) {
			return "";
		} else if(i==0) {
			return construirSCMC(r,i,t,j-
1).concat(String.valueOf(t.charAt(j-1)));
		} else if(j==0) {
			return construirSCMC(r,i-
1,t,j).concat(String.valueOf(r.charAt(i-1)));
		}else {
			if(b[i-1][j-1].equals(Direccion.DIAG)) {
				return construirSCMC(r,i-1,t,j-
1).concat(String.valueOf(r.charAt(i-1)));
			} else if (b[i-1][j-1].equals(Direccion.ARRIBA)) {
				return construirSCMC(r,i-
1,t,j).concat(String.valueOf(r.charAt(i-1)));
			} else {
				return construirSCMC(r,i,t,j-
1).concat(String.valueOf(t.charAt(j-1)));
			}
		}
	}



	// representacion como String de la instancia
	public String toString(){
		return "Sigma="+Utils.entreComillas(sigma)
			 +", r="+Utils.entreComillas(r)
			 +", s="+Utils.entreComillas(t);
	}



	// Obtiene una solucion al problema por "fuerza bruta"
	public String unaSolucionFB() {
		int l = Math.max(r.length(),t.length());
		String res = null;
		for(l=Math.max(r.length(),t.length()); res==null; l++)
		  res = unaSolucionFB("",l);
		return res;
	}

	// método auxiliar recursivo
```

```
	private String unaSolucionFB(String s, int l) {
		String str = null;
		if(l==0) {
			if(Utils.esSupersecuencia(s,r) &&
Utils.esSupersecuencia(s,t))
				str = s;
		}
		else
			for(int i=0; i<sigma.length(); i++) {
				str = unaSolucionFB(s+sigma.charAt(i),l-1);
				if(str!=null) break;
			}
		return str;
	}


}
```

ESTRATEGIA EMPLEADA:

Sean s, r, t tres cadenas de símbolos, diremos que la cadena s es supersecuencia común de las cadenas r y t, si todos los caracteres de r y t están presentes en s en el mismo orden en el que aparecen en las cadenas iniciales. Dicho esto, es fácil encontrar una supersecuencia común de ambas (su concatenación), sin embargo, se nos propone el problema de encontrar una supersecuencia común lo más corta posible. Este es precisamente el problema SCMC.

Este problema puede ser resuelto usando programación dinámica. Para ello, se puede ver que:

1) Si el último símbolo de las dos cadenas coincide, la SCMC de ambas debe acabar necesariamente con dicho símbolo.

2) En otro caso, la SCMC podrá finalizar bien con el último símbolo de la primera cadena, bien con el último símbolo de la segunda. De ambas opciones, tomaremos la que dé lugar a una SCMC.

Si se razona como en el ejemplo visto en clase (Subsecuencia común más larga) se puede ver que, en general, se cumple el **principio de optimalidad** para toda instancia que resuelve el problema de calcular la SCMC paras dos cadenas.

En primer lugar, se plantea la relación de recurrencias correspondiente al problema SCMC. Para ello, se tendrán también en cuenta dos casos base, en los que, si una de las cadenas es vacía, la SCMC será la otra cadena no vacía. Nuestra ecuación en recurrencia (que mostrará la forma de rellenar luego la matriz) nos dará la longitud de la SCMC para las distintas comparaciones de las subcadenas:

$$SCMC(i,j)\begin{cases} j \ , \ i=0 \\ i \ , \ j=0 \\ 1+SCMC(i-1,j-1)\ , \ \ i,j>0 \ \wedge ri=tj \\ 1+\min\{SCMC(i,j-1),SCMC(i-1,j)\}, \ \ i,j>0 \ \wedge ri \neq tj \end{cases}$$

Donde i y j son las longitudes de las subcadenas de r y t. Esta ecuación es la que hemos usado en el programa para rellenar la tabla con las distintas longitudes de las SCMC que resultan de la comparación de las subcadenas de r y t, de modo que, al llegar a la última casilla rellenada, esta habrá comparado las cadenas al completo. Analizando un poco la estructura de nuestra función, vemos que la tabla se rellenará de arriba abajo y de izquierda a derecha. En consecuencia, la solución de nuestro problema se encontrará en la casilla (n,m) de la matriz, siendo n y m las respectivas longitudes de r y t.

A continuación, se pide implementar un algoritmo que reconstruya, usando la tabla anterior, una supersecuencia común más corta. Para ello se ha usado la siguiente estrategia:

1) Se crea un tipo enumerado "*Direccion"*, con los nombres de las direcciones que va siguiente nuestro problema dentro de la tabla de longitudes (DIAG, ARRIBA, IZDA).

2) Se inicializa una nueva matriz "b" con el mismo tamaño de la otra matriz y que recogerá datos del tipo "*Direccion"*.

3) Al rellenar la tabla principal, iremos siguiendo el camino que toman las soluciones, es decir, cuando hay una coincidencia en el último carácter, la solución de SCMC(i,j) viene de SCMC(i-1, j-1), la DIAGONAL; y cuando no hay coincidencia, si la SCMC(i-1,j) es menor que la SCMC(i,j-1), la solución viene de ARRIBA; en otro caso, la solución viene de IZQUIERDA.

4) Se implementa el algoritmo *"construirSCMC"* el cual construirá la solución de problema recorriendo la tabla desde la casilla (n,m) hacia atrás de manera recursiva. Se van comparando los elementos guardados en la tabla "b" con los tres posibles valores de dirección. En cada caso, se guardará el carácter correspondiente que hay en la dirección que apunta el elemento de la tabla.

El método *"unaSoluciónPD"* llamará a la función explicada y devolverá la supercadena común más corta, la cual ha sido haya mediante programación dinámica. Para comprobar que el algoritmo funciona correctamente, tenemos el siguiente programa:

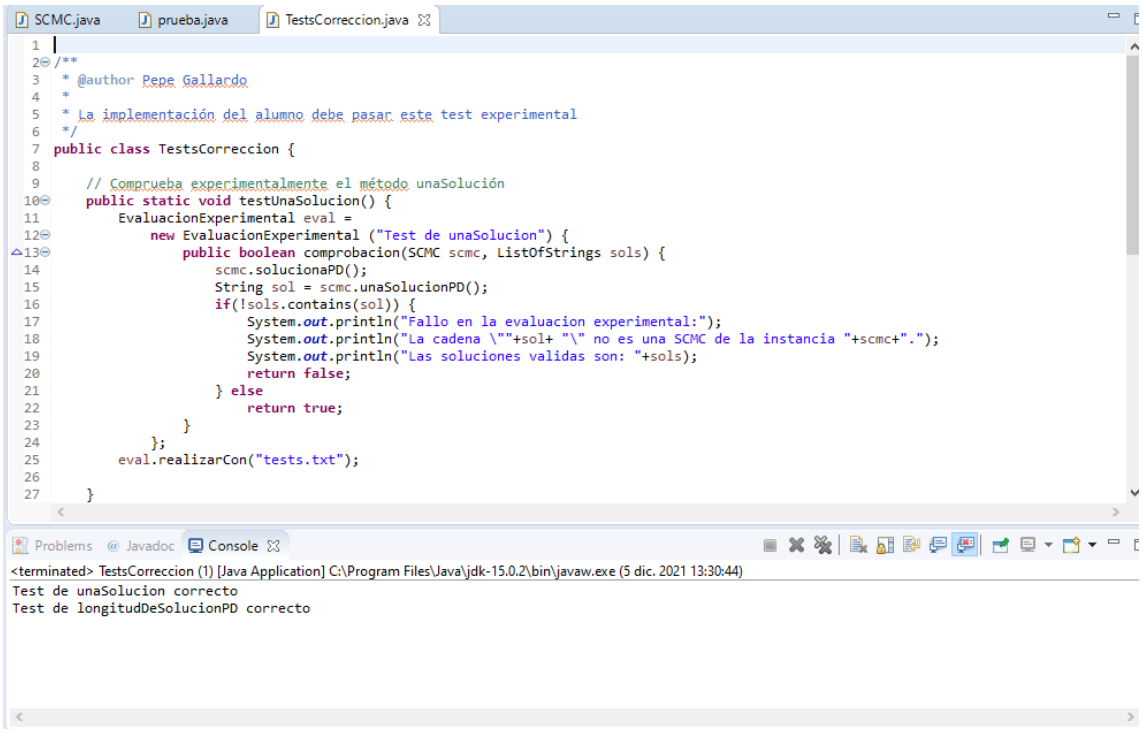
```
public class prueba {

	public static void main(String[] args) {
		SCMC scmc = new SCMC("abc", "aab", "acbaa");
		scmc.solucionaPD();
		System.out.println("La longitud de la SCMC es " +
scmc.longitudDeSolucionPD());
		System.out.println("Una SCMC es " + scmc.unaSolucionPD());
	}
}
```

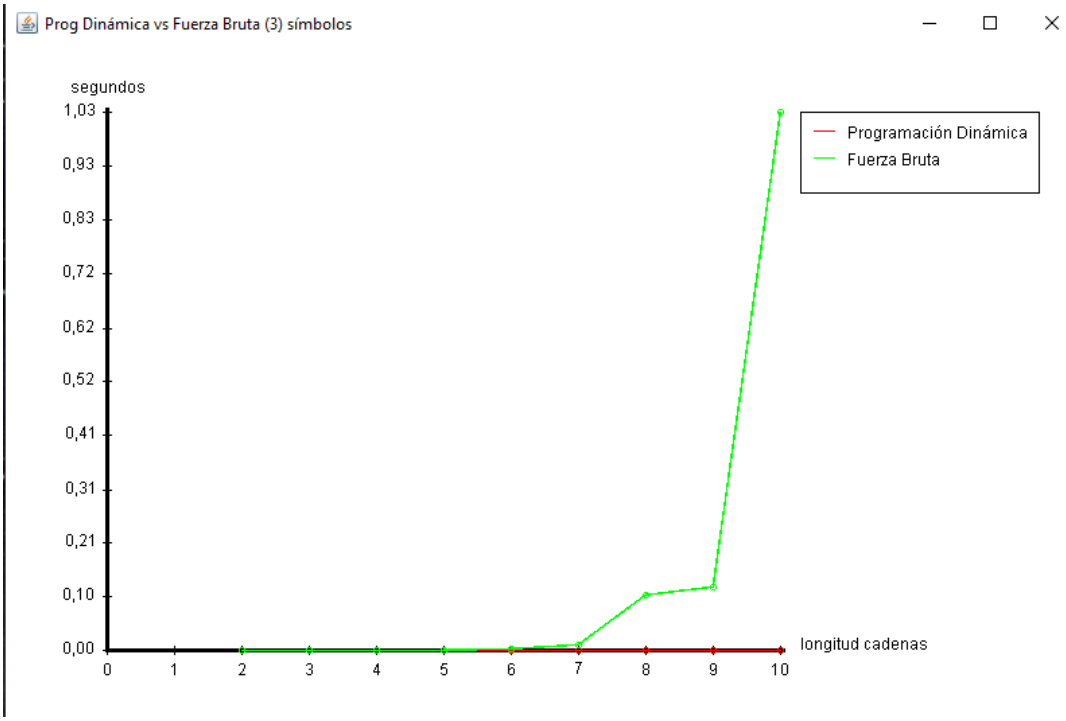
Que produce la siguiente salida:

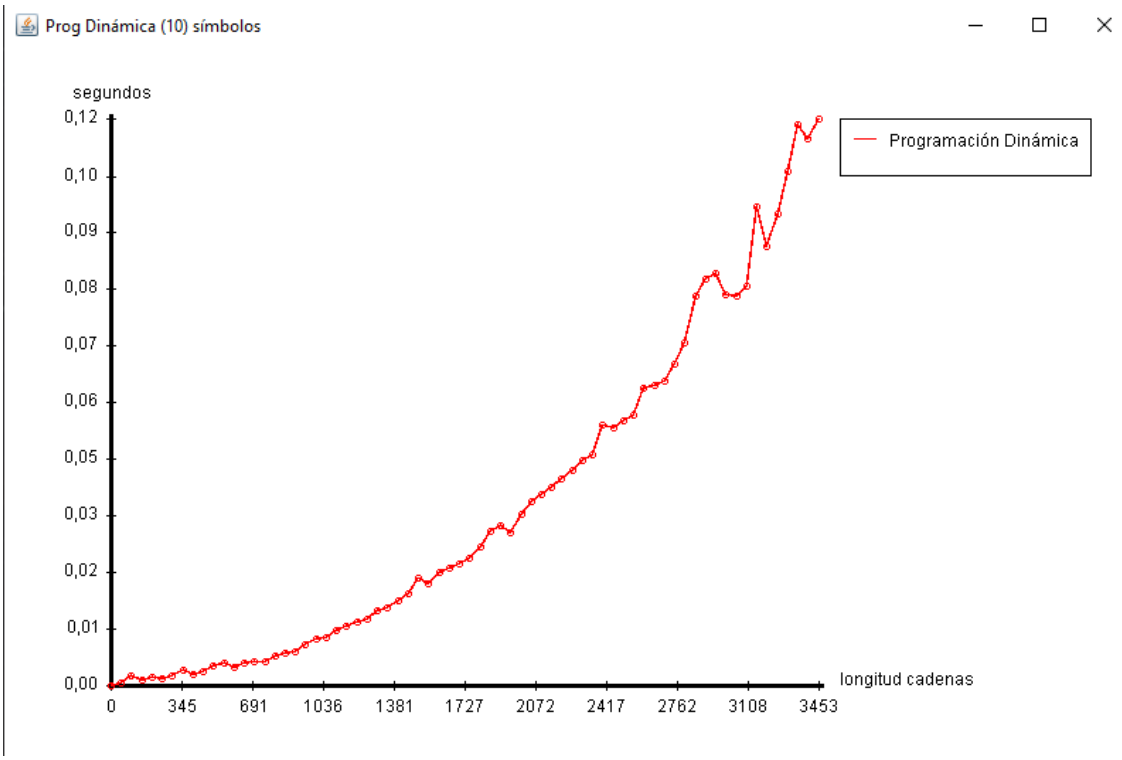
```
La longitud de la SCMC es 6
Una SCMC es aacbaa
```

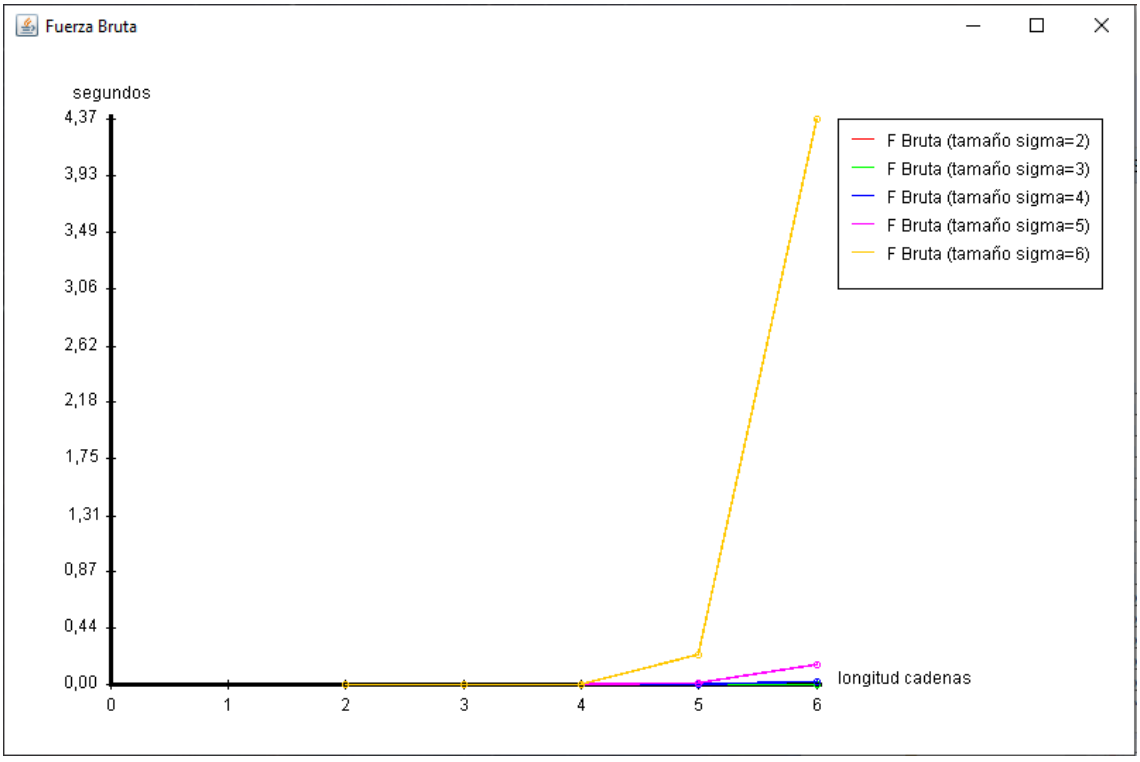
Tomando la clase TestsCorreccion, se puede realizar una comprobación del algoritmo usado. En este caso, la respuesta conseguida es la siguiente:



Con las clases `Temporizador` y `Gráfica` suministradas, podrás realizar gráficas para estudiar experimentalmente el comportamiento de tu implementación frente al método de fuerza bruta. Para ello, se ejecuta el método *main* de la clase TestTiempos:




Prog Dinámica (10) símbolos
segundos
0,12
0,10
0,09
0,08
0,07
0,06
0,05
0,03
0,02
0,01
0,00
0
345
691
1036
1381
1727
2072
2417
2762
3108
3453
longitud cadenas
Programación Dinámica


Fuerza Bruta
segundos
4,37
3,93
3,49
3,06
2,62
2,18
1,75
1,31
0,87
0,44
0,00
0
1
2
3
4
5
6
longitud cadenas
F Bruta (tamaño sigma=2)
F Bruta (tamaño sigma=3)
F Bruta (tamaño sigma=4)
F Bruta (tamaño sigma=5)
F Bruta (tamaño sigma=6)