PROGRAMACIÓN Y ESTRUCTURAS DE DATOS AVANZADAS

PRÁCTICA 2

Luis Daniel Guardado Garcia
77175422A
Lguardado7@alumno.uned.es

CUESTIONES TEÓRICAS DE LA PRÁCTICA

1) Analiza el coste computacional del algoritmo.

El algoritmo es de orden $O(n^3)$. Esta es la ecuación de recurrencia.

$$M(i,j,k) = \begin{cases} 0, & si\ k = 0\ y\ i = j \\ A[i,j], & si\ k = 0\ y\ i \neq j \\ \min\left(M(i,j,k-1), \right. & si\ k > 0 \\ \left. M(i,k,k-1) + M(k,j,k-1)\right) & \end{cases}$$

El coste mínimo entre dos vértices del grafo i y j corresponde a buscar el camino mínimo considerado como intermedio a cualquiera de los otros nodos $M(i,j,k)$.

El coste temporal de este algoritmo, con los tres bucles anidados, está en $O(n^3)$, mientras el coste espacial está en $O(n^2)$.

A pesar de que hay tres bucles, las sentencias de código solo se hacen en el tercer bucle anidado.

Además, estas sentencias son asignaciones, sumas y condicionales. Esto hace que sea más fácil gestionar la memoria. Tal y como pone en el texto base: "Las operaciones del algoritmo de Floyd son más simples, lo que nos indica que probablemente le correspondan constantes ocultas más pequeñas, y por tanto en la práctica sea más rápido el algoritmo de Floyd".

2) Explica qué otros esquemas pueden resolver el problema y razona sobre su idoneidad.

Algoritmo de Dijkstra:

Dijkstra es un algoritmo para encontrar el camino más corto, dado un vértice de origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Cuando se encuentra el camino más corto el algoritmo termina.

Algoritmo de Floyd:

Floyd es un algoritmo de análisis de grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución.

Aunque la complejidad de los algoritmos de Dijkstra y Floyd es la misma, las operaciones del algoritmo de Floyd son más simples, lo que nos indica que probablemente le correspondan constantes ocultas más pequeñas, y por tanto en la práctica sea más

rápido el algoritmo de Floyd. Sin embargo, para grafos poco densos, el algoritmo de Dijkstra puede optimizarse utilizando listas de distancias a nodos adyacentes para representar el grafo, lo que aumenta su velocidad de cómputo. Por lo tanto, Floyd es preferible para grafos densos, y Dijkstra para grafos dispersos.

UN EJEMPLO DE EJECUCIÓN PARA DISTINTOS TAMAÑOS DEL PROBLEMA

Ejemplo nº1: Sólo como argumento el fichero de entrada.

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped2\out\production\ped2>java floyd D:\uned\2anno\1semestre\preda\floyd\fichero_entr
ada.txt
[1,1]:1,1:0
[1,2]:1,4,2:3
[1,3]:1,4,2,3:4
[1,4]:1,4:1
[2,1]:2,3,1:5
[2,2]:2,2:0
[2,3]:2,3:1
[2,4]:2,3,4:6
[3,1]:3,1:4
[3,2]:3,4,2:7
[3,3]:3,3:0
[3,4]:3,4:5
[4,1]:4,2,3,1:7
[4,2]:4,2:2
[4,3]:4,2,3:3
[4,4]:4,4:0
```

Ejemplo nº2: Sólo como argumento el fichero de entrada (no existente).

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped2\out\production\ped2>java floyd D:\uned.txt

 Añade la matriz manualmente

0 8 - 1
- 0 1 -
4 - 0 -
- 2 9 0

[1,1]:1,1:0
[1,2]:1,4,2:3
[1,3]:1,4,2,3:4
[1,4]:1,4:1
[2,1]:2,3,1:5
[2,2]:2,2:0
[2,3]:2,3:1
[2,4]:2,3,4:6
[3,1]:3,1:4
[3,2]:3,4,2:7
[3,3]:3,3:0
[3,4]:3,4:5
[4,1]:4,2,3,1:7
[4,2]:4,2:2
[4,3]:4,2,3:3
[4,4]:4,4:0
```

Ejemplo nº3: Como argumentos el fichero de entrada y salida (sin crear).

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped2\out\production\ped2>java floyd D:\uned\2anno\1semestre\preda\floyd\fichero_entr
ada.txt D:\uned\2anno\1semestre\preda\fichero_salida.txt

 Archivo de salida creada. Si no se ha creado es porque has introducido mal la ubicación o el formato de salida (txt)
```

Ejemplo nº3: Como argumentos el fichero de entrada y salida (creado).

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped2\out\production\ped2>java floyd D:\uned\2anno\1semestre\preda\floyd\fichero_entr
ada.txt D:\uned\2anno\1semestre\preda\fichero_salida.txt

 El archivo de salida ya existe.
```

Ejemplo nº4: Como argumentos, la tecla t y fichero de entrada.

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped2\out\production\ped2>java floyd t D:\uned\2anno\1semestre\preda\floyd\fichero_en
trada.txt

============== Floyd ==============
La entrada de este algorimo consiste en dos tablas denominadas a y m, y un
entero que contiene la longitud de la tabla a.
La tabla 'a' sería la siguiente:

0 8 - 1
- 0 1 -
4 - 0 -
- 2 9 0

Igualamos la tabla 'm' con la tabla 'a' y obtenemos los mínimos de la
tabla m, que serían estos.

0 3 4 1
5 0 1 6
4 7 0 5
7 2 3 0

Ahora creamos otra tabla llamada 'ruta' y llamamos a Floyd2

============== Floyd2 ==============
La entrada de este algorimo consiste en tres tablas denominadas a, m y ruta, y un
entero que contiene la longitud de la tabla a.
Igualamos las tablas m con a y le damos valor '0' a todos los valores que pueda dar ruta.
Así quedaría la tabla m.

0 8 - 1
- 0 1 -
4 - 0 -
- 2 9 0

Pasamos a VerRutas
```

```
============== VerRutas ==============

La entrada de este algorimo consiste en dos tablas denominadas m y ruta, y un
entero que contiene la longitud de la tabla a.
Dejamos que el algoritmo haga su trabajo...

Y así quedaría la solución:

[1,1]:1,1:0
[1,2]:1,4,2:3
[1,3]:1,4,2,3:4
[1,4]:1,4:1
[2,1]:2,3,1:5
[2,2]:2,2:0
[2,3]:2,3:1
[2,4]:2,3,4:6
[3,1]:3,1:4
[3,2]:3,4,2:7
[3,3]:3,3:0
[3,4]:3,4:5
[4,1]:4,2,3,1:7
[4,2]:4,2:2
[4,3]:4,2,3:3
[4,4]:4,4:0
```

Ejemplo nº5: Sólo como argumento, la letra h.

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped2\out\production\ped2>java floyd h

 SINTAXIS: floyd [-t] [-h]  [fichero entrada]  [fichero salida]
      -t          Traza el algoritmo
      -h          Muestra esta ayuda
      [fichero entrada]   Matriz de adyacencia que representa el grafo
      [ficharo salida]   Para cada par de nodos: la lista de nodos del
                         camino más corto y su valor o longitud
```

Ejemplo nº6: Como argumentos, la letra t, fichero de entrada y fichero de salida.

```
D:\UNED\2anno\1semestre\PREDA\PREDA\ped2\out\production\ped2>java floyd t D:\uned\2anno\1semestre\preda\floyd\fichero_en
trada.txt D:\uned\2anno\1semestre\preda\floyd\fichero_salida.txt

============== Floyd ==============
La entrada de este algorimo consiste en dos tablas denominadas a y m, y un
entero que contiene la longitud de la tabla a.
La tabla 'a' sería la siguiente:

0 8 - 1
- 0 1 -
4 - 0 -
- 2 9 0

Igualamos la tabla 'm' con la tabla 'a' y obtenemos los mínimos de la
tabla m, que serían estos.

0 3 4 1
5 0 1 6
4 7 0 5
7 2 3 0

Ahora creamos otra tabla llamada 'ruta' y llamamos a Floyd2

============== Floyd2 ==============
La entrada de este algorimo consiste en tres tablas denominadas a, m y ruta, y un
entero que contiene la longitud de la tabla a.
Igualamos las tablas m con a y le damos valor '0' a todos los valores que pueda dar ruta.
Así quedaría la tabla m.

0 8 - 1
- 0 1 -
4 - 0 -
- 2 9 0

Pasamos a VerRutas
```

```
============== VerRutas ==============

La entrada de este algorimo consiste en dos tablas denominadas m y ruta, y un
entero que contiene la longitud de la tabla a.
Dejamos que el algoritmo haga su trabajo...

Y así quedaría la solución:

[1,1]:1,1:0
[1,2]:1,4,2:3
[1,3]:1,4,2,3:4
[1,4]:1,4:1
[2,1]:2,3,1:5
[2,2]:2,2:0
[2,3]:2,3:1
[2,4]:2,3,4:6
[3,1]:3,1:4
[3,2]:3,4,2:7
[3,3]:3,3:0
[3,4]:3,4:5
[4,1]:4,2,3,1:7
[4,2]:4,2:2
[4,3]:4,2,3:3
[4,4]:4,4:0


 Archivo de salida creada. Si no se ha creado es porque has introducido mal la ubicación o el formato de salida (txt)
```

CÓDIGO COMPLETO

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

public class floyd {

    private static final StringBuilder cadena = new StringBuilder();
    private static final Scanner snr = new Scanner(System.in);
    private static final StringBuilder resumen = new StringBuilder();

    public static void Floyd(int[][] a, int n, int[][] m) {
// O(n^3)
        int i, j, k;
// 1

        resumen.append("\n============== Floyd ==============\n");
// 1
        resumen.append("La entrada de este algorimo consiste en dos
tablas denominadas a y m, y un\n");   // 1
        resumen.append("entero que contiene la longitud de la tabla
a.\n");                               // 1
        resumen.append("La tabla 'a' sería la siguiente:\n\n");
// 1
        for (i = 0; i < n; i++)
{
// n
            for (j = 0; j < n; j++) {
// n^2
                m[i][j] = a[i][j];

                // Esta parte del algoritmo es para crear la traza
                getMatrices(n, m, i, j);
            }
        }

        resumen.append("\nIgualamos la tabla 'm' con la tabla 'a' y
obtenemos los mínimos de la");        // 1
        resumen.append("\ntabla m, que serían
estos.\n\n");                                              // 1
        for (k = 0; k < n; k++) {
// n
            for (i = 0; i < n; i++) {
// n^2
                for (j = 0; j < n; j++) {
// n^3
```

```
                    m[i][j] = Math.min(m[i][j], m[i][k] + m[k][j]);
                }
            }
        }

        // Esta parte corresponde al algoritmo de trazas
        for (i = 0; i < n; i++) {
// n
            for (j = 0; j < n; j++) {
// n^2
                getMatrices(n, m, i, j);
            }
        }

        resumen.append("\nAhora creamos otra tabla llamada 'ruta' y
llamamos a Floyd2\n");                // 1
        resumen.append("\n============== Floyd2
==============\n");                                    // 1
        resumen.append("La entrada de este algorimo consiste en tres
tablas denominadas a, m y ruta, y un\n");
        resumen.append("entero que contiene la longitud de la tabla
a.\n");                               // 1
        int[][] ruta = new int[n][n];
        Floyd2(a, n, m, ruta);
// 1
    }

    public static void Floyd2(int[][] a, int n, int[][] m, int[][]
ruta) {                                // O(n^3)
        int i, j, k, tmp;
// 1

        resumen.append("Igualamos las tablas m con a y le damos valor
'0' a todos los valores que pueda dar ruta.\nAsí quedaría la tabla
m.\n\n");
        for (i = 0; i < n; i++) {
// n
            for (j = 0; j < n; j++) {
// n^2
                m[i][j] = a[i][j];
                ruta[i][j] = 0;

                // Esta parte del algoritmo es para crear la traza
                getMatrices(n, m, i, j);
            }
        }

        for (k = 0; k < n; k++) {
// n
            for (i = 0; i < n; i++) {
```

```
// n^2
                for (j = 0; j < n; j++) {
// n^3
                    tmp = m[i][k] + m[k][j];
                    if (tmp < m[i][j]) {
                        m[i][j] = tmp;
                        ruta[i][j] = k;
                    }
                }
            }
        }
        resumen.append("\nPasamos a VerRutas\n");
// 1
        resumen.append("\n============== VerRutas
==============\n\n");                                    // 1
        resumen.append("La entrada de este algorimo consiste en dos
tablas denominadas m y ruta, y un\n");// 1
        resumen.append("entero que contiene la longitud de la tabla
a.\n");                                 // 1
        resumen.append("Dejamos que el algoritmo haga su
trabajo...\n");                                     // 1
        VerRutas(n, m, ruta);
// 1
        resumen.append("\nY así quedaría la solución:\n");
// 1
        resumen.append(cadena);
// 1
    }

    public static void VerRutas(int n, int[][] m, int[][] ruta) {
// O(n^2)
        int i, j;
// 1
        cadena.append("\n");
        for (i = 0; i < n; i++) {
// n
            for (j = 0; j < n; j++) {
// n^2
                if (m[i][j] != 999999999) {
                    //System.out.print("[" + (i+1) + "," + (j+1) +
"]:");
                    cadena.append("[").append(i +
1).append(",").append(j + 1).append("]:");
                    //System.out.print(i+1+",");
                    cadena.append(i + 1).append(",");
                    ImprimeRutasRec(ruta, i, j);
                    //System.out.println((j+1) + ":" + (m[i][j]));
                    cadena.append(j +
1).append(":").append(m[i][j]).append("\n");
                }
```

```
            }
        }
    }

    public static void ImprimeRutasRec(int[][] ruta, int i, int j) {
// O(1)
        int k;
// 1
        k = ruta[i][j];
// 1
        if (k != 0) {
// 1
            ImprimeRutasRec(ruta, i, k);
            //System.out.print(k+1+",");
            cadena.append(k + 1).append(",");
            ImprimeRutasRec(ruta, k, j);
        }
    }

    // Menú de ayuda
    public static void ayuda() {
// O(1)
        System.out.println();
        System.out.println(" SINTAXIS: floyd [-t] [-h]  [fichero
entrada]  [fichero salida]");
        System.out.println("      -t          Traza el algoritmo");
        System.out.println("      -h          Muestra esta ayuda");
        System.out.println("      [fichero entrada]   Matriz de
adyacencia que representa el grafo");
        System.out.println("      [ficharo salida]   Para cada par de
nodos: la lista de nodos del");
        System.out.println("                          camino más corto
y su valor o longitud");
        System.out.println();
    }

    // Función auxiliar que imprime las matrices
    private static void getMatrices(int n, int[][] m, int i, int j) {
// O(1)
        if (j == n - 1) {
// 1
            if (m[i][j] == 999999999) {
                resumen.append("-
").append(System.getProperty("line.separator"));
            } else {

resumen.append(m[i][j]).append(System.getProperty("line.separator"));
            }
        } else {
// 1
```

```
            if (m[i][j] != 999999999) {
                resumen.append(m[i][j]).append(" ");
            } else {
                resumen.append("-").append(" ");
            }
        }
    }

    public static void programa(String t, String h, String entrada,
String salida) {                        // O(n^2)
        try {
// 1
            String prueba;
// 1
            File fentrada = new File(entrada);
// 1
            File fsalida = new File(salida);
// 1
            int num = 0;
// 1

            if (fentrada.exists()) {
// 1
                BufferedReader bufferedReader = new BufferedReader(new
FileReader(fentrada));
                // Averiguamos el tamaño del array
                prueba = bufferedReader.readLine();
                prueba = prueba.replaceAll("-", "999999999");
                prueba = prueba.trim();
                String[] row1 = prueba.split(" ");
                int[][] arrayInt = new int[row1.length][row1.length];
                for (int i = 0; i < row1.length; i++) {
// n^2
                    arrayInt[num][i] = Integer.parseInt(row1[i]);
                }
// n
                num++;
                // El formato de entrada es una matriz de adyacencia.
                while ((prueba = bufferedReader.readLine()) != null) {
// n
                    prueba = prueba.replaceAll("-", "999999999");
                    prueba = prueba.trim();
                    String[] row = prueba.split(" ");
                    for (int i = 0; i < row.length; i++) {
// n^2
                        arrayInt[num][i] = Integer.parseInt(row[i]);
                    }
// n
                    num++;
                }
```

```
                // Esto es un miniprograma que consiste en comprobar
si es necesario crear una salida.txt o imprimirla por pantalla.
                if (salida.equals("0")) {
// 1
                    int n = arrayInt.length;
                    int[][] m = new int[n][n];
                    Floyd(arrayInt, n, m);
                } else {
// 1
                    if (fsalida.exists()) {
                        System.out.println("error");
                    } else {
                        int n = arrayInt.length;
                        int[][] m = new int[n][n];
                        Floyd(arrayInt, n, m);
                        BufferedWriter bufferedWriter = new
BufferedWriter(new FileWriter(fsalida));
                        bufferedWriter.write(String.valueOf(cadena));
                        bufferedWriter.newLine();
                        bufferedWriter.close();
                    }
                }
            } else {
// 1
                System.out.println("\n Añade la matriz
manualmente\n");                                    // 1
                String matriz1 = snr.nextLine();
                matriz1 = matriz1.replaceAll("-", "999999999");
                String[] row1 = matriz1.split(" ");
                int n = row1.length;
// 1
                int[][] arrayInt = new int[n][n];
                int[][] m = new int[n][n];
// 1
                int numero =
0;
// 1
                for (int i = 0; i < row1.length; i++) {
// n^2
                    arrayInt[numero][i] = Integer.parseInt(row1[i]);
                }
                numero++;
// n
                while (numero != row1.length) {
// n
                    String matriz = snr.nextLine();
                    matriz = matriz.replaceAll("-", "999999999");
                    String[] row = matriz.split(" ");
                    for (int i = 0; i < row.length; i++) {
```

```
// n^2
                        arrayInt[numero][i] =
Integer.parseInt(row[i]);
                    }
                    numero++;
// n
                }
                Floyd(arrayInt, n, m);
// 1
                if (!salida.equals("0")) {
// 1
                    if (!fsalida.exists()) {
                        BufferedWriter bufferedWriter = new
BufferedWriter(new FileWriter(fsalida));
                        bufferedWriter.write(String.valueOf(cadena));
                        bufferedWriter.newLine();
                        bufferedWriter.close();
                    }
                }
            }
        } catch (IOException e) {
// 1
            System.out.println(" Error E/S: " + e);
        }
    }

    public static void main(String[] args) {
// O(1)
        String h = "h";
// 1
        String t =
"t";
// 1

        // Argumentos
        switch (args.length) {
// 1
            case 1: {
                File entrada = new File(args[0]);
                if (args[0].equals(h)) {                        // Si
el argumento 0 es igual a "h", que salga el menú de ayuda.
                    ayuda();
                } else if (args[0].equals(t)) {                 // Si
el argumento 0 es igual a "t", nos salta un error indicándonos que
tenemos que introducir una entrada.
                    System.out.println("\n Debe introducir una
entrada\n");
                } else if (entrada.exists()) {                  // Si
el argumento 0 contiene una entrada, que nos muestre la salida por
consola.
```

```
                    String txtEntrada = entrada.toString();
                    programa(h, t, txtEntrada, "0");
                    System.out.println(cadena);
                } else if (!entrada.exists()) {
                    programa(h, t, "0", "0");
                    System.out.println(cadena);
                } else {                                            // Si
el argumento 0 contiene otro tipo que no sea los anteriores indicados,
que nos lance un error.
                    System.out.println("\n Los parámetros introducidos
son erróneos.\n");
                }
                break;
            }
            case 2: {
// 1
                File entrada2 = new File(args[0]);
                File salida2 = new File(args[1]);
                File entrada2_ = new File(args[1]);
                if (args[0].equals(t) && args[1].equals(h) ||
args[0].equals(h) && args[1].equals(t)) {  // Si los argumentos 0 y 1
es igual a h y t o viceversa, muestra el menú de ayuda y lanza un
error.
                    ayuda();
                    System.out.println("\n Para que muestre la traza
debe introducir una entrada.\n");
                } else if (args[0].equals(t) && entrada2_.exists() ||
args[0].equals(t) && !entrada2_.exists()) {        // Si el argumento
0 contiene t y el segundo contiene una entrada, que nos muestre la
salida y su traza.
                    String txtEntrada = entrada2_.toString();
                    programa(t, h, txtEntrada, "0");
                    System.out.println(resumen);
                } else if (args[0].equals(h) && entrada2_.exists() ||
args[0].equals(h) && !entrada2_.exists()) {        // Si el argumento
0 contiene h y el segundo contiene una entrada, que nos muestre la
salida y el menú de ayuda.
                    String txtEntrada = entrada2_.toString();
                    ayuda();
                    programa(t, h, txtEntrada, "0");
                    System.out.println(cadena);
                } else if (entrada2.exists() && !salida2.exists()) {
// Si el argumento 0 contiene una entrada y el segundo una salida.txt
no creada, que nos cree la salida.txt
                    String txtEntrada = entrada2.toString();
                    String txtSalida = salida2.toString();
                    programa(t, h, txtEntrada, txtSalida);
                    System.out.println("\n Archivo de salida creada.
Si no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
```

```
                } else if (entrada2.exists() && salida2.exists() ||
!entrada2.exists() && salida2.exists()) {          // Si el argumento
0 contiene una entrada y el segundo una salida.txt creada, que nos
lance un error.
                    System.out.println("\n El archivo de salida ya
existe.\n");
                } else if (!entrada2.exists() && !salida2.exists()) {
                    String txtEntrada = entrada2.toString();
                    String txtSalida = salida2.toString();
                    programa(t, h, txtEntrada, txtSalida);
                    System.out.println("\n Archivo de salida creada.
Si no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                } else {
                    System.out.println("\n Los parámetros introducidos
son erróneos.\n");
                }
                break;
            }
            case 3: {
// 1
                File entrada3 = new File(args[1]);
                File salida3 = new File(args[2]);
                File entrada4 = new File(args[2]);
                if (args[0].equals(t) && entrada3.exists() &&
!salida3.exists()) {  // Si el argumento 0 contiene "t", argumento 1
contiene una entrada y argumento 2 contiene una salida.txt no creada,
que nos muestre su traza y nos cree una salida.txt
                    String txtEntrada2 = entrada3.toString();
                    String txtSalida2 = salida3.toString();
                    programa(t, h, txtEntrada2, txtSalida2);
                    System.out.println(resumen);
                    System.out.println("\n Archivo de salida creada.
Si no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                } else if (args[0].equals(h) && entrada3.exists() &&
!salida3.exists()) {  // Si el argumento 0 contiene "h", arguemento 1
contiene una entrada y argumento 2 contiene una salida.txt no creada,
que nos muestre el menú de ayuda y nos cree una salida.txt
                    String txtEntrada2 = entrada3.toString();
                    String txtSalida2 = salida3.toString();
                    ayuda();
                    programa(t, h, txtEntrada2, txtSalida2);
                    System.out.println("\n Archivo de salida creada.
Si no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                } else if (args[0].equals(t) && entrada3.exists() &&
salida3.exists() || args[0].equals(h) && entrada3.exists() &&
salida3.exists()) {
                    System.out.println("\n El archivo de salida ya
```

```
existe.\n");
                } else if (args[0].equals(t) && args[1].equals(h) &
entrada3.exists() || args[0].equals(h) && args[1].equals(t) &
entrada3.exists()) {
                    String txtEntrada3 = entrada4.toString();
                    ayuda();
                    programa(t, h, txtEntrada3, "0");
                    System.out.println(resumen);
                } else if (args[0].equals(t) && !entrada3.exists() &&
!salida3.exists()) {
                    String txtEntrada2 = entrada3.toString();
                    String txtSalida2 = salida3.toString();
                    programa(t, h, txtEntrada2, txtSalida2);
                    System.out.println(resumen);
                    System.out.println("\n Archivo de salida creada.
Si no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                } else if (args[0].equals(h) && !entrada3.exists() &&
!salida3.exists()) {
                    String txtEntrada2 = entrada3.toString();
                    String txtSalida2 = salida3.toString();
                    ayuda();
                    programa(t, h, txtEntrada2, txtSalida2);
                    System.out.println("\n Archivo de salida creada.
Si no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                } else if (args[0].equals(t) && !entrada3.exists() &&
salida3.exists() || args[0].equals(h) && !entrada3.exists() &&
salida3.exists()) {
                    System.out.println("\n El archivo de salida ya
existe.\n");
                } else {    // Si los argumentos contiene algún dato
no especificado anteriormente, nos lanzará un error.
                    System.out.println("\n Los parámetros introducidos
son erróneos.\n");
                }
                break;
            }
            case 4:
{
// 1
                File entrada4 = new File(args[2]);
                File salida4 = new File(args[3]);  // Si el argumento
0 contiene t, argumento 1 contiene h, argumento 2 contiene una entrada
y argumento 3 contiene una salida.txt no creada, que nos muestre su
trazado y nos cree una salida.txt
                if (args[0].equals(t) && args[1].equals(h) &&
entrada4.exists() && !salida4.exists() || args[0].equals(h) &&
args[1].equals(t) && entrada4.exists() && !salida4.exists()) {
                    String txtEntrada3 = entrada4.toString();
```

```
                    String txtSalida3 = salida4.toString();
                    ayuda();
                    programa(t, h, txtEntrada3, txtSalida3);
                    System.out.println(resumen);
                    System.out.println("\n Archivo de salida creada.
Si no se ha creado es porque has introducido mal la ubicación o el
formato de salida (txt)");
                } else if (args[0].equals(t) && args[1].equals(h) &&
entrada4.exists() && salida4.exists() || args[0].equals(h) &&
args[1].equals(t) && entrada4.exists() && salida4.exists()) {
                    System.out.println("\n El archivo de salida ya
existe.\n");
                }
            }
            break;
        }
    }
}
```