MEMORIA DE LA PRACTICA INDIVIDUAL 2

José Manuel Pérez Álvarez

Índice

FICHEROS

Especifico.txt

```
min: T;

+5*x00+4*x01+6*x02+3*x03+2*x04<=T;
+5*x10+4*x11+6*x12+3*x13+2*x14<=T;

+x00+x10=1;
+x01+x11=1;
+x02+x12=1;
+x03+x13=1;
+x04+x14=1;

int T;
bin x00,x01,x02,x03,x04,x10,x11,x12,x13,x14;
```

"Específico.txt" usado en el primer apartado del ejercicio 1.

Este texto ya tiene formato LPsolve.

generico.txt

```
5,4,6,3,2
2
```

"generico.txt" para el segundo apartado del ejercicio 1.

El formato de este texto para ser pasado a formato LPsolve posteriormente es el siguiente: en la primera línea añadimos todas las tareas separadas mediante comas, y en la segunda línea de texto el número de procesadores.

P4-Grafo2.txt

```
#VERTEX#
Sitio0
Sitio1
Sitio2
Sitio3
Sitio4
Sitio5
Sitio6
Sitio7
Sitio8
Sitio9
#EDGE#
Sitio0,Sitio1,0
Sitio0,Sitio2,0
Sitio1,Sitio2,0
Sitio1,Sitio3,0
Sitio1,Sitio4,0
Sitio2,Sitio3,0
Sitio2,Sitio4,0
Sitio3,Sitio4,0
Sitio4,Sitio3,0
Sitio5,Sitio3,0
Sitio5,Sitio4,0
Sitio5,Sitio6,0
Sitio5,Sitio7,0
Sitio6,Sitio7,0
Sitio6,Sitio9,0
Sitio7,Sitio6,0
Sitio7,Sitio9,0
Sitio8,Sitio9,0
```

"P4-Grafo2.txt" es usado en el ejercicio 2 para crear un grafo dirigido. Al principio del texto tenemos los vértices y seguido las aristas con el formato de "sitio origen, sitio destino, tiempo"

"P4-Grafo1.txt" es usado en el ejercicio 1 para la conectividad del grafo. Tiene el mismo formato que el comentado en el anterior.

P4-Grafo1.txt

```
#VERTEX#
Sitio0
Sitio1
Sitio2
Sitio3
Sitio4
Sitio5
Sitio6
Sitio7
Sitio8
Sitio9
#EDGE#
Sitio0,Sitio1,20.0
Sitio0,Sitio2,15.0
Sitio1,Sitio3,10.0
Sitio1,Sitio4,25.0
Sitio2,Sitio4,15.0
Sitio3,Sitio4,5.0
Sitio3,Sitio5,30.0
Sitio4,Sitio5,35.0
Sitio4,Sitio7,40.0
Sitio5,Sitio6,35.0
Sitio5,Sitio7,20.0
Sitio8,Sitio9,15.0
```

EJERCICIO 2

```java
public static List<String> leerFichero(String fichero) {
    List<String> res=null;
    try {
        res = Files.readAllLines(Paths.get(fichero));
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return res;
}

private static String funcionObjetivo(List<String> fichero) {

    String res="min: T;\n\n";
    String[] tareas = getTareas(fichero);

    for (int i = 0; i < getProcesadores(fichero); i++) {
        for (int j = 0; j < tareas.length; j++) {
            res+="+"+tareas[j]+"*x"+i+j;
        }
        res+="<=T;\n";
    }
    return res+"\n";
}
```

Para comenzar, tendremos la función "leerFichero" que nos devolverá una lista con que tendrá dos cadenas, el primero contendrá las tareas y el segundo el número de procesadores.

La "funcionObjetivo" se encargará de crear como su nombre indica, la función para minimizar las tareas en formato LPsolve. Éste método además también usa otros como "getTareas" y "getProcesadores", los cuales devuelven un Array con las tareas y un Integer con el número de procesadores.

El tipo de formato a convertir será asociarlo de la siguiente forma:
"[tarea] * x[numProcesador][numTarea]"

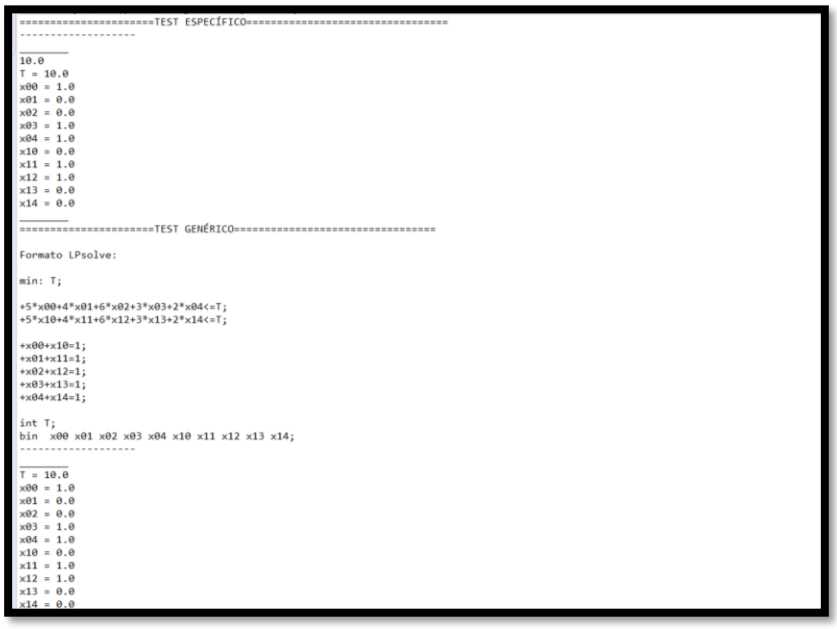
Aquí aparecen los métodos "Restricciones" y "declaraVariables", que su función es la misma que la explicada anteriormente con "funcionObjetivo", tratan de convertir los datos de restricción y las declaraciones de variables en una cadena con formato LPsolve.

```java
private static String Restricciones(List<String> fichero) {

    String res="";

    for (int i = 0; i < getTareas(fichero).length; i++) {
        for (int j = 0; j < getProcesadores(fichero); j++) {
            res+="+x"+j+i;
        }
        res+="=1;\n";
    }

    return res+"\n";
}

private static String declaraVariables(List<String> fichero) {
    String res="int T;\nbin ";
    for (int i = 0; i < getProcesadores(fichero); i++) {
        for (int j = 0; j < getTareas(fichero).length; j++) {
            res+=" x"+i+j;
        }
    }
    return res+";";
}
```

```java
private static String concatenarProblema(List<String> fichero) {

    String res = funcionObjetivo(fichero);
    res += Restricciones(fichero)+declaraVariables(fichero);
    return res;
}
```

Y, por último, encontramos esta función que concatenará todas las cadenas para obtener una única cadena para poder ser pasada al AlgoritmoPLI.

SALIDA POR CONSOLA EJ2

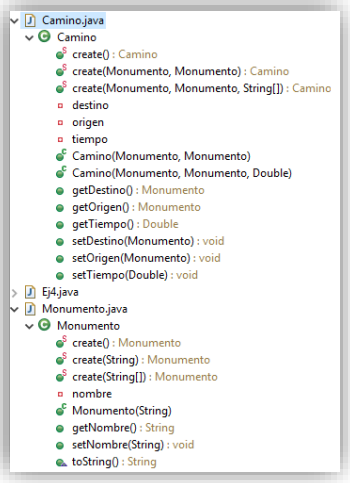


```
====================TEST ESPECÍFICO==============================
------------------
_______
10.0
T = 10.0
x00 = 1.0
x01 = 0.0
x02 = 0.0
x03 = 1.0
x04 = 1.0
x10 = 0.0
x11 = 1.0
x12 = 1.0
x13 = 0.0
x14 = 0.0
_______
====================TEST GENÉRICO================================

Formato LPsolve:

min: T;

+5*x00+4*x01+6*x02+3*x03+2*x04<=T;
+5*x10+4*x11+6*x12+3*x13+2*x14<=T;

+x00+x10=1;
+x01+x11=1;
+x02+x12=1;
+x03+x13=1;
+x04+x14=1;

int T;
bin  x00 x01 x02 x03 x04 x10 x11 x12 x13 x14;
------------------
_______
T = 10.0
x00 = 1.0
x01 = 0.0
x02 = 0.0
x03 = 1.0
x04 = 1.0
x10 = 0.0
x11 = 1.0
x12 = 1.0
x13 = 0.0
x14 = 0.0
```

En el primer apartado del ejercicio tenemos que crear un fichero en formato LPsolve (“*Especifico.txt*” página 1) y posteriormente leerlo y mostrar la solución por pantalla.

En el siguiente apartado, a partir de un fichero en el cual solo se entregan los datos, implementar un código que convierta esos datos en un texto con formato LPsolve. Como podemos observar al insertar los mismos datos que en el anterior, obtenemos la misma solución.

EJERCICIO 4



Para este ejercicio necesitaremos implementar dos clases, Monumentos Y Caminos. En nuestro problema los usaremos de manera en la que los vértices del grafo serán los monumentos y el camino serán las aristas.

```
public static void apartadoA(Graph<Monumento, Camino> grafo) {

    ConnectivityInspector<Monumento, Camino> resG = new ConnectivityInspector<>(grafo);
    System.out.println("==============================APARTADO A==============================\n");
    if (resG.isConnected()) {
        System.out.print("Están todos los sitios conectados entre sí? : Sí.");
    } else {
        System.out.print("Están todos los sitios conectados entre sí? : No.");
    }
    System.out.println("Las listas conexas son= {");
    List<Set<Monumento>> Sets = resG.connectedSets();
    for (Set<Monumento> set : Sets) {
        System.out.println(set);
    }
    System.out.println("}");
}
```

El apartado A tendremos que responder a la siguiente cuestión: ¿Están todos los sitios conectados entre sí?

Para contestar a esto haremos uso de la clase “ConnectivityInspector”, que mediante el método que nos ofrece “isConnected” podremos ver si todos los vértices están conectados respondiendo con una afirmación o negación. A continuación, mostraremos los conjuntos de vértices que están unidos, todo ello mediante el método “connectedSets”.

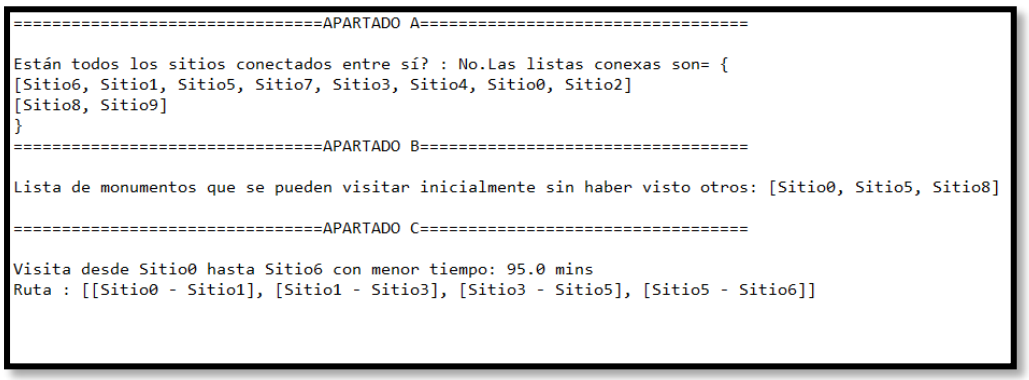
```java
public static void apartadoB(Graph<Monumento, Camino> grafo) {

    Set<Monumento> vertices = grafo.vertexSet();
    System.out.println("================================APARTADO B================================\n");
    System.out.print("Lista de monumentos que se pueden visitar inicialmente sin haber visto otros: ");
    System.out.println(
            vertices.stream().filter(x -> grafo.incomingEdgesOf(x).isEmpty()).collect(Collectors.toList()) + "\n");
}
```

Avanzando al siguiente apartado, tendremos que mostrar los monumentos que se pueden visitar inicialmente sin haber visitado ningún otro anteriormente, teniendo en cuenta esto, sabemos que esto serán los vértices que no tengan aristas de entradas. Sabiendo esto crearemos un flujo de datos que filtraremos, usando el método “incomingEdgesOf” añadirá a una lista las aristas entrantes en un vértice, así que solo hay que comprobar que ésta esté vacía, únicamente la añadiremos a una lista y pasamos a mostrarlo por pantalla.

```java
public static void apartadoC(String m0, String m1, Graph<Monumento, Camino> grafo) {

    System.out.println("================================APARTADO C================================\n");
    List<Monumento> extremos = new LinkedList<>();

    extremos.add(grafo.vertexSet().stream().filter(x -> x.getNombre().equals(m0)).findFirst().get());
    extremos.add(grafo.vertexSet().stream().filter(x -> x.getNombre().equals(m1)).findFirst().get());

    DijkstraShortestPath<Monumento, Camino> sub = new DijkstraShortestPath<Monumento, Camino>(grafo,
            extremos.get(0), extremos.get(1));

    if (sub.getPathEdgeList() != null) {

        double sumaTiempo = 0.0;
        for (Camino c : sub.getPathEdgeList()) {
            sumaTiempo += c.getTiempo();
        }
        System.out.println("Visita desde " + extremos.get(0) + " hasta " + extremos.get(1) + " con menor tiempo: "
                + sumaTiempo + " mins");
        if (m0.compareTo(m1) < 0) {
            System.out.println("Ruta : " + sub.getPathEdgeList().stream()
                    .map(x -> "[" + x.getOrigen() + " - " + x.getDestino() + "]").collect(Collectors.toList()));
        } else {
            System.out.println("Ruta : " + sub.getPathEdgeList().stream()
                    .map(x -> "[" + x.getDestino() + " - " + x.getOrigen() + "]").collect(Collectors.toList()));
        }
    } else {
        System.out.println("No hay camino posible");
    }

}
```

Y llegamos al último apartado, en el que tendremos que encontrar el camino mas corto entre dos vértices y mostrar el tiempo total del recorrido. Para ello empezaremos buscando los vértices en el grafo, puesto que nos han pasado el nombre, una vez obtenidos mediante la clase “DijkstraShortesPath” obtendremos el camino mas corto entre dos vértices. Comprobaremos que exista una arista que los una, en caso contrario no estarán conectados. Una vez comprobado declaramos una variable que será un acumulador donde hará el recuento de todos los tiempos. Ya teniendo todos estos datos solo es cuestión de mostrarlo por pantalla y para ello mapeamos el camino obtenido por Dijkstra, lo convertimos en lista y ya obtendríamos el volcado por pantalla.

SALIDA POR CONSOLA EJ2



En el apartado A vemos como no todos los caminos están conectados y correctamente niega la pregunta.

Continuando con el B observamos cuales son los que se pueden visitar inicialmente. (Para este apartado se requería el uso de grafos dirigidos, por lo que usamos el fichero *"P4-Grafo2.txt"*).

Acabamos con el apartado C, podemos comprobar que se quiere visitar desde el Sitio0 al Sitio6 que efectivamente están conectados, por lo que hay camino, y se muestra la ruta mas corta con su duración.