Tercera Práctica Individual de ADDA

Memoria técnica del proyecto

Roberto Camino Bueno, grupo asignado número **3**

Firmado por Roberto Camino

Índice:

1. Ficheros en formato LpSolve.

- El fichero para el ejemplo concreto contiene lo siguiente:

```
min: b0 + b1 + b2 + b3 + b4 + b5;

b0 + b1 >=1;            // Desde el barrio 0,
b0 + b1 + b5 >=1;       // barrio 1, ...
b2 + b3 >=1;
b2 + b3 + b4 >=1;
b3 + b4 + b5 >=1;
b1 + b4 + b5 >=1;       // hasta el barrio 5

bin b0, b1, b2, b3, b4, b5;
```

- El fichero de datos para el ejemplo genérico contiene lo siguiente:

```
b0,b1
b0,b1,b5
b2,b3
b2,b3,b4
b3,b4,b5
b1,b4,b5
```

- El fichero autogenerado a partir del anterior es “intermedio.txt”:

```
min: b0 + b1 + b2 + b3 + b4 + b5;

b0 + b1 >=1;
b0 + b1 + b5 >=1;
b2 + b3 >=1;
b2 + b3 + b4 >=1;
b3 + b4 + b5 >=1;
b1 + b4 + b5 >=1;

bin b0, b1, b2, b3, b4, b5;
```

2. Código con la soluciones de los problemas.

- Problema 1, solución con PL

```
public class solucionesPL {

    public static void main(String[] args) {

        System.out.println("\n-------------------- PROBLEMA 1, EJEMPLO CONCRETO --------------------");
        SolutionPLI alg = AlgoritmoPLI.getSolutionFromFile("./ficheros/ejemploConcretoPL.txt");
        System.out.println("\nA continucación observamos los valores para los barrios (Barrio_i) "
                + "donde 1.0 significa que establece estación y 0.0 que no: \n");
        for (int i = 0; i<alg.getNumVar(); i++) {
            System.out.println("Barrio_" + i + " = " + alg.getSolution(i) + ";");
        }
        System.out.println("Cantidad de barrios con estaciones de bomberos: " + alg.getGoal());



        SolutionPLI alg2 = AlgoritmoPLI.getSolution(defineProblema("./ficheros/ejemploGenericoPL.txt"));
        System.out.println("\n-------------------- PROBLEMA 1, EJEMPLO GENÉRICO --------------------\n");
        System.out.println("A continucación observamos los valores para los barrios (Barrio_i) "
                + "donde 1.0 significa que establece estación y 0.0 que no: \n");
        for (int i = 0; i<alg2.getNumVar(); i++) {
            System.out.println("Barrio_"+i+" = "+alg2.getSolution(i)+";");
        }
        System.out.println("Cantidad de barrios con estaciones de bomberos: " + alg2.getGoal());
    }

    private static List<Barrio> cargaDatos(String fichero) { // Lee los datos del fichero
        return Streams2.fromFile(fichero).map(Barrio::create).collect(Collectors.toList());
    }


private static String defineProblema(String fichero) {
    List<Barrio> datos = cargaDatos(fichero);
    String res = funcionObjetivo(datos); // min: x0 + x1 + x2 + x3 + x4 + x5;
    res += restriccionGrupoVecinos(datos); // restricciones para todos los 'grupoVecinos'
    res += variablesBarrios(datos); // bin b0, b1, b2, b3, b4, b5;
    return res;
}

private static String funcionObjetivo(List<Barrio> barrios) {
    return "min: " + barrios.stream()
                    .map(b -> b.getGrupoVecinos())
                    .flatMap(v -> v.stream()).distinct().sorted()
                    .collect(Collectors.joining(" + ","",";\n\n"));
}

private static String restriccionGrupoVecinos(List<Barrio> barrios) {
    return barrios.stream()
                .map(b -> b.getGrupoVecinos())
                .map(e -> String.join(" + ", e))  // suma cada elemento 'bi'
                .map(s -> String.format("%s >=1;", s)). // restriccion en cada linea
                collect(Collectors.joining("\n","","\n\n"));
}

private static String variablesBarrios(List<Barrio> barrios) {
    return "bin " + barrios.stream()
                    .map(b -> b.getGrupoVecinos())
                    .flatMap(v -> v.stream()).distinct().sorted()
                    .collect(Collectors.joining(", ","",";"));
}
```

- **Problema 1, solución con AG**

```
public class solucionAG implements ValuesInRangeProblemAG<Integer, List<Integer>> {

    private List<BarrioAG> Barrios;

    public Integer getVariableNumber() {
        return Barrios.size();
    }

    public Integer getMax(Integer i) {
        return 2; // rango abierto
    }

    public Integer getMin(Integer i) {
        return 0; // rango cerrado
    }

    public List<Integer> getSolucion(ValuesInRangeChromosome<Integer> cr) {
        List<Integer> res = new ArrayList<>();
        List<Integer> ls = cr.decode();
        for (int i = 0; i < this.getVariableNumber(); i++) {
            res.add(ls.get(i)); // añade 0 o 1
        }
        return res;
    }


public Double fitnessFunction(ValuesInRangeChromosome<Integer> cr) {
    List<Integer> ls_cr = cr.decode();
    Integer cont_estaciones = 0;
    Set<Integer> set_v = new HashSet<>();
    for (int i = 0; i < ls_cr.size(); i++) {
        if (ls_cr.get(i).equals(1)) {
            set_v.addAll(Barrios.get(i).getGrupoVecinos());
            cont_estaciones++;
        }
    }
    Double penaliza = 0.0;
    if (Barrios.size() >= set_v.size()) {
        penaliza = (double) (Barrios.size() - set_v.size());
    }
    return -(penaliza * 1000 + cont_estaciones);

}

public solucionAG(String f) {
    this.Barrios = cargaDatos(f);
}

private static List<BarrioAG> cargaDatos(String fichero) {
    return Streams2.fromFile(fichero).map(BarrioAG::create).collect(Collectors.toList());
}
```

```java
public class testAG {

    public static void main(String[] args) {
        setCondicionesGeneticas();
        ValuesInRangeProblemAG<Integer, List<Integer>> ejercicio = new solucionAG("./ficheros/ejercicioAG.txt");
        AlgoritmoAG<ValuesInRangeChromosome<Integer>> algG = AlgoritmoAG.create(ChromosomeType.Binary, ejercicio);

        algG.ejecuta();
        System.out.println("\n-------------------- PROBLEMA 1, SOLUCIÓN CON ALGORITMO GENÉTICO --------------------\n");
        System.out.println(
                "Los barrios(b) donde establecemos las estaciones vienen indicados con el 1:\n\n[b0,b1,b2,b3,b4,b5] ");
        System.out.println(algG.getBestChromosome().decode());

        ValuesInRangeChromosome<Integer> sol = algG.getBestChromosome();
        System.out.println("\nCoste (fitness): " + ejercicio.fitnessFunction(sol) * -1);
    }

    private static void setCondicionesGeneticas() {
        // __________condiciones evolutivas por defecto __________
        AlgoritmoAG.CROSSOVER_RATE = 0.8;
        AlgoritmoAG.MUTATION_RATE = 0.5;
        AlgoritmoAG.ELITISM_RATE = 0.25;
        AlgoritmoAG.POPULATION_SIZE = 300;

        // __________condiciones de parada __________
        StoppingConditionFactory.NUM_GENERATIONS = 1000;
    }

}
```

- **Problema 4, solución de los grafos con jgrapht**

```
public class solucionApartados {

    public static void main(String[] args) {
        System.out.println("\n-------------------- PROBLEMA 4 --------------------");
        System.out.println("\n Para ver el Grafo Simple Ponderado copie el contenido "
            + "del fichero 'asociacionCultural.gv' en http://www.webgraphviz.com/ \n "
            + "o bien para ver el Grafo Dirigido copie el contenido del fichero 'asociacionCulturalDirigido.gv' \n");
        SimpleWeightedGraph<Monumento, Ruta> grafo = cargaGrafo("./ficheros/asociacionCultural.txt");

        System.out.println("\n-------------------- APARTADO A --------------------\n");
        estanTodosConectados(grafo);

        System.out.println("\n-------------------- APARTADO B --------------------\n");
        SimpleDirectedGraph<Monumento, Ruta_D> grafoDirigido = cargaGrafoDirigido(
                "./ficheros/asociacionCulturaDirigido.txt");
        verticesPadre(grafoDirigido);

        System.out.println("\n-------------------- APARTADO C --------------------\n");
        System.out.println("\n--> Caso de solución posible:");
        List<Monumento> subconjunto1 = new ArrayList<>();
        subconjunto1.add(new Monumento("Monumento_1"));
        subconjunto1.add(new Monumento("Monumento_6"));
        subconjunto1.add(new Monumento("Monumento_9"));
        tiempoMinimo2(grafo, grafoDirigido, subconjunto1);

        System.out.println("\n\n--> Caso de solución NO posible:");
        List<Monumento> subconjunto2 = new ArrayList<>();
        subconjunto2.add(new Monumento("Monumento_1"));
        subconjunto2.add(new Monumento("Monumento_3"));
        subconjunto2.add(new Monumento("Monumento_6"));
        tiempoMinimo2(grafo, grafoDirigido, subconjunto2);
    }

// __________ A __________
private static void estanTodosConectados(SimpleWeightedGraph<Monumento, Ruta> grafo) {
    var alg = new ConnectivityInspector<>(grafo);
    System.out.println("¿Están todos los lugares conectados entre sí? " + alg.isConnected());
    System.out.println("\nGrafo: " + "\nVertices: " + grafo.vertexSet() + "\nAristas: " + grafo.edgeSet());
}

// __________ B __________
private static void verticesPadre(SimpleDirectedGraph<Monumento, Ruta_D> grafoDirigido) {
    List<Monumento> res = grafoDirigido.vertexSet().stream().filter(v -> grafoDirigido.inDegreeOf(v) == 0)
            .collect(Collectors.toList());
    System.out.println("Lugar o lugares iniciales: " + res);
    System.out.println("\nGrafo Dirigido: " + "\nVertices: " + grafoDirigido.vertexSet() + "\nAristas: "+ grafoDirigido.edgeSet());
}



// __________ C __________
private static void tiempoMinimo2(SimpleWeightedGraph<Monumento, Ruta> grafo,
        SimpleDirectedGraph<Monumento, Ruta_D> grafoDirigido, List<Monumento> ls) {
    System.out.println("Nuestro viaje es :" + ls + ", comenzamos:");

    ShortestPathAlgorithm<Monumento, Ruta_D> alg = new DijkstraShortestPath<>(grafoDirigido);
    Boolean hayCaminoD = true;
    Double t = 0.0;
    for (int i = 0; i < ls.size() - 1; i++) {
        Monumento origen = ls.get(i);
        Monumento destino = ls.get(i + 1);
        var camino = alg.getPath(origen, destino);
        if (camino == null) {
            System.out.println(
                    "\nAVISO: NO ES POSIBLE SEGUN EL ORDEN DE PRECEDENCIA VISITAR " + ls + " luego no hay viaje.");
            hayCaminoD = false;
        } else {
            var alg1 = new DijkstraShortestPath<>(grafo);
            var gp1 = alg1.getPath(origen, destino);
            if (gp1 != null) {
                t += gp1.getWeight();
            }
        }
    }
    if (hayCaminoD == true) {
        System.out.println("\nEl ORDEN DE PRECEDENCIA segun el Grafo Dirigido PERMITE este viaje.");
```

```
        var alg2 = new ConnectivityInspector<>(grafo);
        Boolean conexo = true;
        for (int i = 0; i < ls.size() - 1; i++) {
            Monumento origen = ls.get(i);
            Monumento destino = ls.get(i + 1);
            if (alg2.pathExists(origen, destino)) {
                var alg1 = new DijkstraShortestPath<>(grafo);
                var gp = alg1.getPath(origen, destino);
                System.out.println("\nLa RUTA MÁS CORTA segun el grafo de conexiones desde el " + origen
                        + " hasta el " + destino + " es " + gp.getVertexList());
                System.out.println("Tiempo empleado: " + gp.getWeight() + " minutos");
                System.out.println("Rutas caminadas: " + gp.getLength());
            } else {
                conexo = false;
                System.out.println("AVISO: NO HAY RUTA SEGUN EL GRAFO DE CONEXIÓN ENTRE " + origen + " Y " + destino
                        + " luego no hay viaje.");
                break;
            }
            if (conexo) {
                if (destino.equals(ls.get(ls.size() - 1))) {
                    System.out.println("¡Hurra, ya hemos llegado a nuestro destino!");
                    System.out.println("El tiempo total empleado en el viaje más corto es " + t + " minutos");
                    break;
                } else {
                    System.out.println("Continuamos el trayecto... ");
                }
            }
        }
    }
}


// __________ GraphsReader __________
private static SimpleWeightedGraph<Monumento, Ruta> cargaGrafo(String f) {
    return GraphsReader.newGraph("./ficheros/asociacionCultural.txt", Monumento::create, Ruta::create,
            () -> new SimpleWeightedGraph<>(Monumento::create, Ruta::create), Ruta::gettiempo);
}

private static SimpleDirectedGraph<Monumento, Ruta_D> cargaGrafoDirigido(String nombreFichero) {
    return GraphsReader.newGraph(nombreFichero, Monumento::create, Ruta_D::create,
            () -> new SimpleDirectedGraph<>(Monumento::create, Ruta_D::create, false));
}
```

3. Volcado de pantalla con los resultados obtenidos en las pruebas realizadas.

- Problema 1, solución con PL

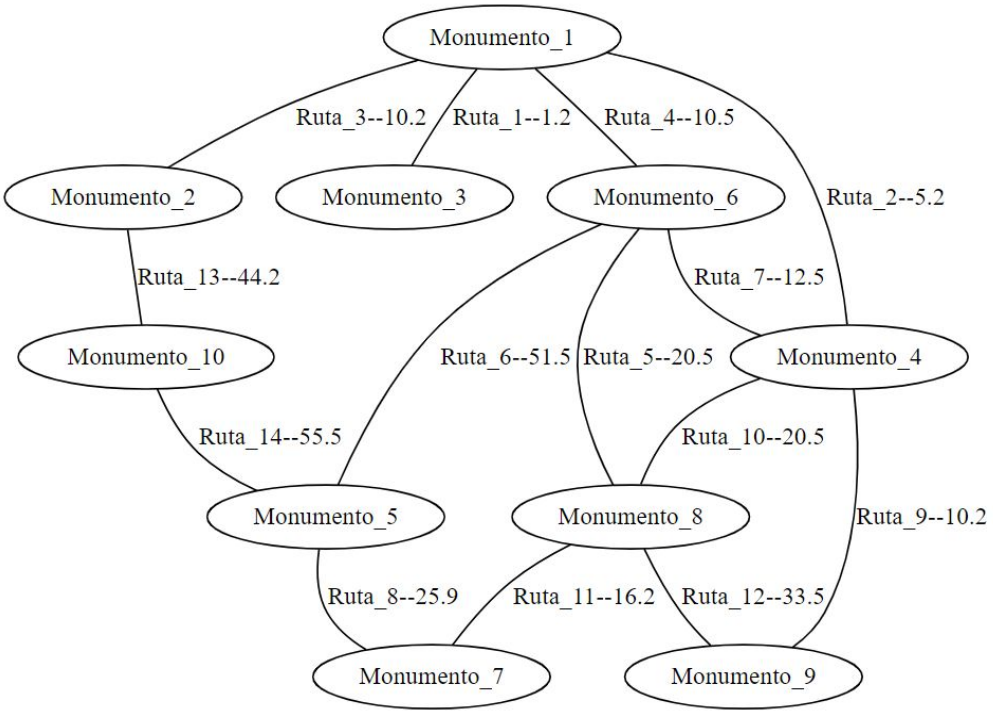
```
-------------------- PROBLEMA 1, EJEMPLO CONCRETO --------------------

A continucación observamos los valores para los barrios (Barrio_i) donde 1.0 significa que establece estación y 0.0 que no:

Barrio_0 = 0.0;
Barrio_1 = 1.0;
Barrio_2 = 0.0;
Barrio_3 = 1.0;
Barrio_4 = 0.0;
Barrio_5 = 0.0;
Cantidad de barrios con estaciones de bomberos: 2.0

-------------------- PROBLEMA 1, EJEMPLO GENÉRICO --------------------

A continucación observamos los valores para los barrios (Barrio_i) donde 1.0 significa que establece estación y 0.0 que no:

Barrio_0 = 0.0;
Barrio_1 = 1.0;
Barrio_2 = 0.0;
Barrio_3 = 1.0;
Barrio_4 = 0.0;
Barrio_5 = 0.0;
Cantidad de barrios con estaciones de bomberos: 2.0
```

- Problema 1, solución con AG

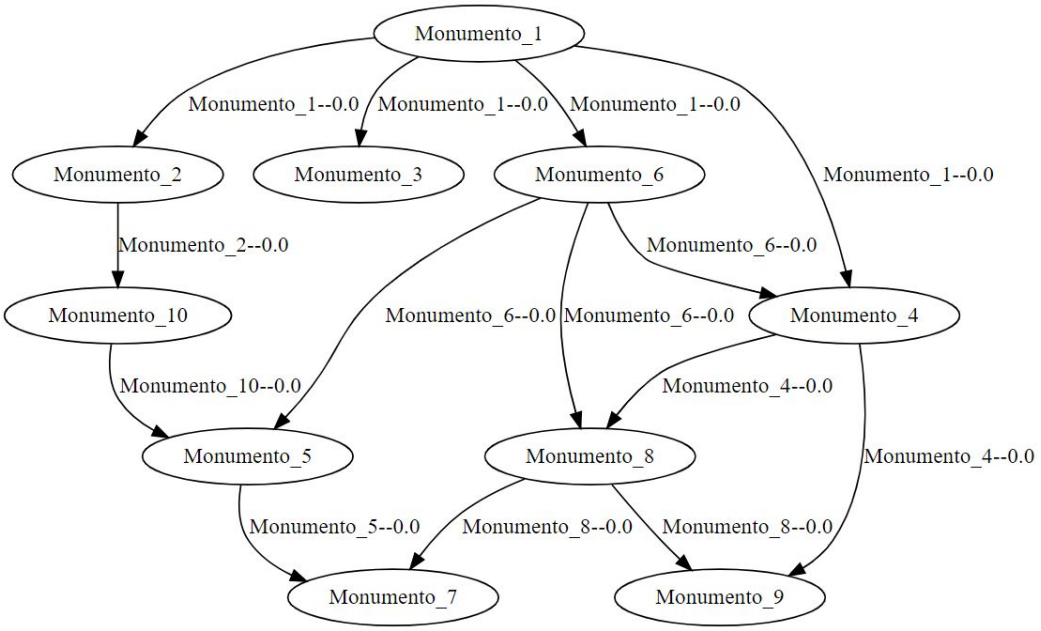
```
-------------------- PROBLEMA 1, SOLUCIÓN CON ALGORITMO GENÉTICO --------------------

Los barrios(b) donde establecemos las estaciones vienen indicados con el 1:

[b0,b1,b2,b3,b4,b5]
[0, 1, 0, 1, 0, 0]

Coste (fitness): 2.0
```

- Problema 4, solución de los grafos con jgrapht

En primer lugar voy a mostrar los grafos usados.
El primero es para las conexiones y el tiempo de cada ruta, se encuentra en "asociacionCultural.txt".



El segundo grafo es para las direcciones y orden de precedencia, se encuentra en "asociacionCulturaDirigido.txt".



A continuación las soluciones:

```
---------- PROBLEMA 4 ----------

 Para ver el Grafo Simple Ponderado copie el contenido del fichero 'asociacionCultural.gv' en http://www.webgraphviz.com/
 o bien para ver el Grafo Dirigido copie el contenido del fichero 'asociacionCulturalDirigido.gv'

---------- APARTADO A ----------

¿Están todos los lugares conectados entre sí? true

Grafo:
Vertices: [Monumento_1, Monumento_2, Monumento_3, Monumento_4, Monumento_5, Monumento_6, Monumento_7, Monumento_8, Monumento_9, Monumento_10]
Aristas: [Ruta_1, Ruta_2, Ruta_3, Ruta_4, Ruta_5, Ruta_6, Ruta_7, Ruta_8, Ruta_9, Ruta_10, Ruta_11, Ruta_12, Ruta_13, Ruta_14]


 ---------- APARTADO B ----------

 Lugar o lugares iniciales: [Monumento_1]

 Grafo Dirigido:
 Vertices: [Monumento_1, Monumento_2, Monumento_3, Monumento_4, Monumento_5, Monumento_6, Monumento_7, Monumento_8, Monumento_9, Monumento_10]
 Aristas: [(Monumento_1,Monumento_3, (Monumento_1,Monumento_4, (Monumento_1,Monumento_2, (Monumento_1,Monumento_6, (Monumento_6,Monumento_8, (Monumento_6,


 --> Caso de solución posible:
 Nuestro viaje es :[Monumento_1, Monumento_6, Monumento_9], comenzamos:

 El ORDEN DE PRECEDENCIA segun el Grafo Dirigido PERMITE este viaje.

 La RUTA MÁS CORTA segun el grafo de conexiones desde el Monumento_1 hasta el Monumento_6 es [Monumento_1, Monumento_6]
 Tiempo empleado: 10.5 minutos
 Rutas caminadas: 1
 Continuamos el trayecto...

 La RUTA MÁS CORTA segun el grafo de conexiones desde el Monumento_6 hasta el Monumento_9 es [Monumento_6, Monumento_4, Monumento_9]
 Tiempo empleado: 22.7 minutos
 Rutas caminadas: 2
 ¡Hurra, ya hemos llegado a nuestro destino!
 El tiempo total empleado en el viaje más corto es 33.2 minutos

 --> Caso de solución NO posible:
 Nuestro viaje es :[Monumento_1, Monumento_3, Monumento_6], comenzamos:

 AVISO: NO ES POSIBLE SEGUN EL ORDEN DE PRECEDENCIA VISITAR [Monumento_1, Monumento_3, Monumento_6] luego no hay viaje.
```