MEMORIA

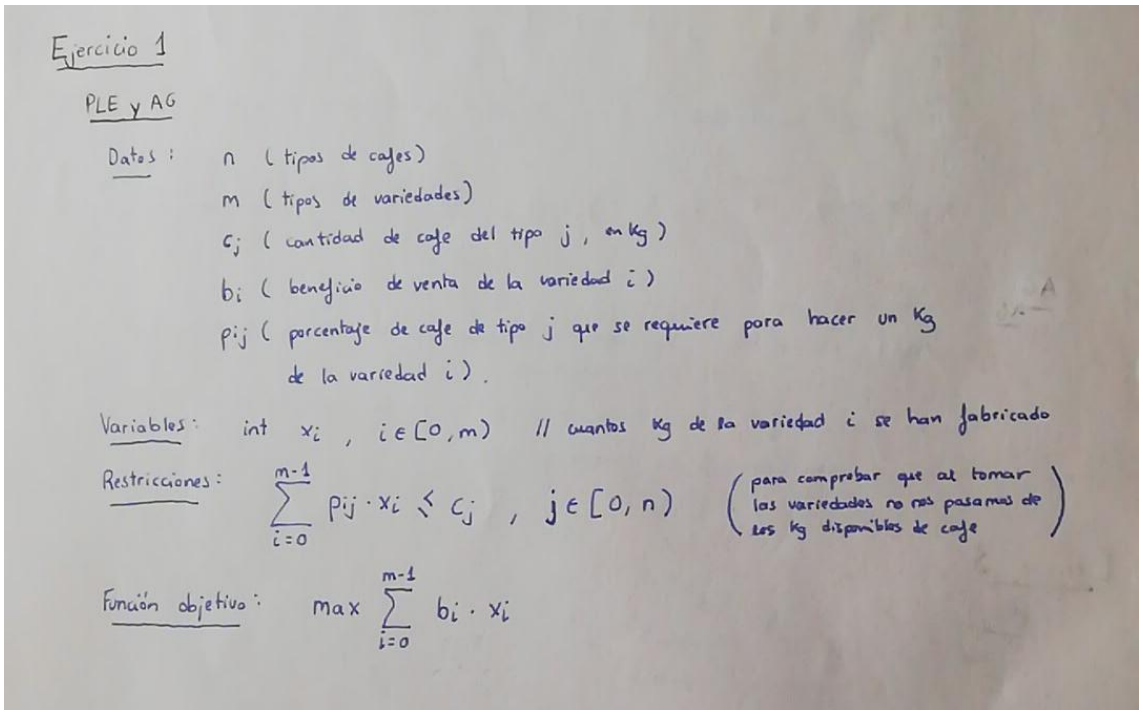
PRÁCTICA INDIVIDUAL 4

Jaime Linares Barrera

2º Ingeniería Informática - Ingeniería del Software, Grupo 4

1. EJERCICIO 1

1.1. MODELOS



Para AG es el mismo planteamiento, por eso no aparece uno especificado para AG

1.2. DATOS

1.2.1. DatosCafes

```java
package _datos;

import java.util.ArrayList;
import java.util.List;

import us.lsi.common.Files2;

public class DatosCafes {

    public static List<Cafe> cafes;
    public static List<Variedad> variedades;

    // Funcion que realiza la lectura de fichero y crea un tipo DatosCafes
    public static void iniDatos(String fichero) {
        List<Cafe> listaCafes = new ArrayList<>();
        List<Variedad> listaVariedades = new ArrayList<>();

        List<String> filas = Files2.linesFromFile(fichero);
        List<String> cafesString = filas.subList(1, filas.indexOf("// VARIEDADES"));
        List<String> variedadesString = filas.subList(filas.indexOf("// VARIEDADES") + 1, filas.size());

        // Transformamos una cadena en el tipo Cafe y lo añadimos a la lista de cafes
        for(String linea: cafesString) {
            Cafe c = Cafe.parseaCafe(linea);
            listaCafes.add(c);
        }
        cafes = listaCafes;

        // Transformamos una cadena en el tipo Variedad y lo añadimos a la lista de variedades
        for(String linea: variedadesString) {
            Variedad v = Variedad.parseaVariedad(linea);
            listaVariedades.add(v);
        }
        variedades = listaVariedades;

        toConsole();
    }
```

```
    // Para mostar el resultado por pantalla
    private static void toConsole() {
        System.out.println("- Tipos de cafe leídos: ");
        for(Cafe c: cafes) {
            System.out.println("  " + c);
        }
        System.out.println("- Tipo de variedades leídas: ");
        for(Variedad v: variedades) {
            System.out.println("  " + v);
        }
        System.out.println("\n");
    }


    // Funcion que me devuelve los tipos de cafe
    public static List<Cafe> getCafes() {
        return cafes;
    }

    // Funcion que calcula el numero de tipo de cafe
    public static Integer getNumTiposCafe() {
        return cafes.size();
    }

    // Funcion que me devuelve las variedades
    public static List<Variedad> getVariedades() {
        return variedades;
    }

    // Funcion que calcula el numero de variedades
    public static Integer getNumVariedades() {
        return variedades.size();
    }

    // Funcion que calcula los kg disponibles de cafe del tipo j
    public static Integer getCantidadCafe(Integer j) {
        return cafes.get(j).kgDisponibles();
    }

    // Funcion que calcula el beneficio de venta por cada kg de la variedad i
    public static Double getBeneficioVentaKg(Integer i) {
        return variedades.get(i).beneficio();
    }

    // Funcion que calcula el porcentaje del cafe j necesarios para un kg de la variedad i
    public static Double getPorcentajeCafeKg(Integer i, Integer j) {
        String nombreCafe = cafes.get(j).nombre();
        return variedades.get(i).comp().stream()
                .filter(x -> x.nombreCafe().equals(nombreCafe))
                .map(x -> x.cantidadKg())
                .findFirst()
                .orElse(0.);
    }


    // TEST
    public static void main(String[] args) {
        System.out.println("* TEST DatosCafes *\n");
        iniDatos("ficheros/Ejercicio1DatosEntrada1.txt");
    }

}
```

1.2.2. Cafe

```java
package _datos;

public record Cafe(String nombre, Integer kgDisponibles) {

    public static Cafe of(String nombre, Integer kgDisponibles) {
        return new Cafe(nombre, kgDisponibles);
    }

    // Funcion para parsear una cadena a Cafe
    public static Cafe parseaCafe(String cadena) {
        String[] partes = cadena.split(":");

        String nombre = partes[0].trim();

        String[] partes1 = partes[1].trim().split("=");
        String kgDisponibles = partes1[1].replace(";", "").trim();

        return Cafe.of(nombre, Integer.valueOf(kgDisponibles));
    }

}
```

1.2.3. Variedad

```java
package _datos;

import java.util.ArrayList;
import java.util.List;

public record Variedad(String nombre, Double beneficio, List<TuplaEj1> comp) {

    public static Variedad of(String nombre, Double beneficio, List<TuplaEj1> comp) {
        return new Variedad(nombre, beneficio, comp);
    }

    // Funcion para parsear una cadena a variedad
    public static Variedad parseaVariedad(String cadena) {
        String[] partes = cadena.split("->");


        String nombre = partes[0].trim();


        String[] partes1 = partes[1].split(";");

        String[] partes10 = partes1[0].trim().split("=");
        String beneficio = partes10[1].trim();

        String cadenaComponentes = partes1[1].trim();
        List<TuplaEj1> comp = listaComponentes(cadenaComponentes);

        return Variedad.of(nombre, Double.valueOf(beneficio), comp);

    }

    // Funcion auxiliar para parsear la lista de componentes
    private static List<TuplaEj1> listaComponentes(String cadena) {
        List<TuplaEj1> res = new ArrayList<>();
        String[] partes = cadena.split("=");

        String[] partes1 = partes[1].trim().split(",");
        for(String cadenaTupla: partes1) {
            TuplaEj1 tupla = TuplaEj1.parseaTuplaEj1(cadenaTupla.trim());
            res.add(tupla);
        }

        return res;
    }

}
```

1.2.4. TuplaEj1

```
package _datos;

public record TuplaEj1(String nombreCafe, Double cantidadKg) {

    public static TuplaEj1 of(String nombreCafe, Double cantidadKg) {
        return new TuplaEj1(nombreCafe, cantidadKg);
    }

    // Funcion para parsear una cadena a una tupla formada por nombre
    // y cantidad por kg del tipo de cafe
    public static TuplaEj1 parseaTuplaEj1(String cadena) {
        String cadenaTupla = cadena.substring(1, cadena.length()-1);
        String[] partes = cadenaTupla.split(":");
        String nombreCafe = partes[0].trim();
        String cantidadKg = partes[1].trim();
        return TuplaEj1.of(nombreCafe, Double.valueOf(cantidadKg));
    }

}
```

1.3. PLE

1.3.1. Ejercicio1PLE

```
package ejercicio1;

import java.io.IOException;
import java.util.List;
import java.util.Locale;

import _datos.Cafe;
import _datos.DatosCafes;
import _datos.Variedad;
import us.lsi.gurobi.GurobiLp;
import us.lsi.gurobi.GurobiSolution;
import us.lsi.solve.AuxGrammar;

public class Ejercicio1PLE {

    public static List<Cafe> cafes;
    public static List<Variedad> variedades;


    // Funcion que calcula el numero de tipo de cafe
    public static Integer getNumTiposCafe() {
        return cafes.size();
    }

    // Funcion que calcula el numero de variedades
    public static Integer getNumVariedades() {
        return variedades.size();
    }

    // Funcion que calcula los kg disponibles de cafe del tipo j
    public static Integer getCantidadCafe(Integer j) {
        return cafes.get(j).kgDisponibles();
    }

    // Funcion que calcula el beneficio de venta por cada kg de la variedad i
    public static Double getBeneficioVentaKg(Integer i) {
        return variedades.get(i).beneficio();
    }
```

```java
    // Funcion que calcula el porcentaje del cafe j necesarios para un kg de la variedad i
    public static Double getPorcentajeCafeKg(Integer i, Integer j) {
        String nombreCafe = cafes.get(j).nombre();
        return variedades.get(i).comp().stream()
                .filter(x -> x.nombreCafe().equals(nombreCafe))
                .map(x -> x.cantidadKg())
                .findFirst()
                .orElse(0.);
    }


    // Función que calcula la solucion utilizando gurobi
    public static void ejercicio1_model() throws IOException {
        DatosCafes.iniDatos("ficheros/Ejercicio1DatosEntrada1.txt");

        cafes = DatosCafes.getCafes();
        variedades = DatosCafes.getVariedades();

        AuxGrammar.generate(Ejercicio1PLE.class, "lsi_models/Ejercicio1.lsi", "gurobi_models/Ejercicio1-1.lp");
        GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio1-1.lp");
        Locale.setDefault(new Locale("en", "US"));
        System.out.println(solution.toString((s,d)->d>0.));
    }


    // TEST
    public static void main(String[] args) throws IOException {
        System.out.println("* TEST Ejercicio1PLE *\n");
        ejercicio1_model();
    }

}
```

1.3.2. Ejercicio1.lsi

```
head section

Integer getNumTiposCafe()
Integer getNumVariedades()
Integer getCantidadCafe(Integer j)
Double getBeneficioVentaKg(Integer i)
Double getPorcentajeCafeKg(Integer i, Integer j)

Integer n= getNumTiposCafe()
Integer m= getNumVariedades()

goal section

max sum(getBeneficioVentaKg(i) x[i], i in 0 .. m)

constraints section

sum(getPorcentajeCafeKg(i,j) x[i], i in 0 .. m) <= getCantidadCafe(j), j in 0 .. n

int

x[i], i in 0 .. m
```

1.3.3. Volcado de pantalla

```
El valor objetivo es 305.00
Los valores de la variables
x_0 == 10
x_1 == 10
x_2 == 1
```

Resultado para los datos de entrada 1.
Significa que se hacen 10kgs de la variedad 0 (la variedad P01), 10kgs de la variedad 1 (P02) y 1 kg de la variedad 2 (P03). El beneficio (valor objetivo) obtenido por fabricar esas cantidades es de 305 (=10*20+10*10+1*5).

```
El valor objetivo es 2000.00
Los valores de la variables
x_0 == 15
x_1 == 10
x_2 == 20
```

Resultado para los datos de entrada 2.
Significa que se hacen 15kgs de la variedad 0 (P01), 10kgs de la variedad 1 (P02) y 20kgs de la variedad 2 (P03). El beneficio (valor objetivo) obtenido por fabricar esas cantidades de cada variedad es de 2000.

```
El valor objetivo es 12275.00
Los valores de la variables
x_0 == 30
x_1 == 4
x_3 == 15
x_5 == 100
```

Resultado para los datos de entrada 3.
Significa que se hacen 30kgs de la variedad 0 (P01), 4kgs de la variedad 1 (P02), 15kgs de la variedad 3 (P04) y 100kgs de la variedad 5 (P06). El beneficio (valor objetivo) obtenido por fabricar tales cantidades de cada variedad es de 12275.

1.4. AG

1.4.1. InRangeCafesAG

```java
package ejercicio1;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import _datos.DatosCafes;
import _datos.TuplaEj1;
import _soluciones.SolucionCafes;
import us.lsi.ag.ValuesInRangeData;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

public class InRangeCafesAG implements ValuesInRangeData<Integer, SolucionCafes> {

    // Constructor
    public InRangeCafesAG(String fichero) {
        DatosCafes.iniDatos(fichero);
    }

    // Tipo de cromosoma que vamos a usar -> cromosoma de rango
    @Override
    public ChromosomeType type() {
        return ChromosomeType.Range;
    }

    // Longitud que van a tener los cromosomas -> numero de variedades
    @Override
    public Integer size() {
        return DatosCafes.getNumVariedades();
    }

    // Valor minimo que puede tomar un gen del cromosoma, es decir,
    // valor minimo de kg que vamos a usar de una variedad
    @Override
    public Integer min(Integer i) {
        return 0;
    }
```

```
    // Valor maximo que puede tomar un gen del cromosoma, es decir,
    // valor maximo de kg que vamos a usar de una variedad (que es el minimo del cociente
    // kgDisponibles/porcentajeCafePorKgVariedad de todos los componentes de la variedad)
    @Override
    public Integer max(Integer i) {
        List<Integer> res = new ArrayList<>();
        // obtenemos los cafes presentes en la variedad
        List<String> cafesEnVariedad = DatosCafes.getVariedades().get(i).comp().stream()
                .map(x -> x.nombreCafe())
                .toList();
        // obtenemos el cociente kgDisponibles/porcentajeCafePorKgVariedad
        for(int j=0; j<DatosCafes.getNumTiposCafe(); j++) {
            String nombreCafe = DatosCafes.getCafes().get(j).nombre();
            if(cafesEnVariedad.contains(nombreCafe)) {
                Integer kgDisponiblesCafe = DatosCafes.getCantidadCafe(j);
                Double porcentajeCafeEnVariedad = DatosCafes.getPorcentajeCafeKg(i, j);
                Integer numPosiblesKg = (int) (kgDisponiblesCafe/porcentajeCafeEnVariedad);
                res.add(numPosiblesKg);
            }
        }
        return res.stream()
                .min(Comparator.naturalOrder())
                .get() + 1;     // sumamos uno porque es intervalo abierto
    }

    // Funcion fitness que sirve para medir lo bueno que es el cromosoma, para ello
    // penalizaremos si se pasa de los kgs disponibles de los cafes
    @Override
    public Double fitnessFunction(List<Integer> ls) {
        Double goal = 0.;
        Double error = 0.;
        Map<String, Double> kgCafeEnCadaVariedad = new HashMap<>();

        // Calculamos los beneficios y ampliamos el map
        for(int i=0; i< size(); i++) {
            if(ls.get(i) > 0) {
                goal += DatosCafes.getBeneficioVentaKg(i) * ls.get(i);
                // Ampliamos el map con los kg que hemos gastado de cada cafe en las variedades elegidas
                for(TuplaEj1 tupla: DatosCafes.getVariedades().get(i).comp()) {
                    String nombreCafe = tupla.nombreCafe();
                    Double cantidadCafe = tupla.cantidadKg()*ls.get(i);
                    if(kgCafeEnCadaVariedad.containsKey(nombreCafe)) {
                        kgCafeEnCadaVariedad.put(nombreCafe, kgCafeEnCadaVariedad.get(nombreCafe)+cantidadCafe);
                    } else {
                        kgCafeEnCadaVariedad.put(nombreCafe, cantidadCafe);
                    }
                }
            }
        }

        // Sumamos al error lo que nos hallamos pasado de los kg disponibles de cada cafe
        for(int j=0; j<DatosCafes.getNumTiposCafe(); j++) {
            String nombre = DatosCafes.getCafes().get(j).nombre();
            Double kgDisponibles = DatosCafes.getCantidadCafe(j) + 0.;
            if(kgCafeEnCadaVariedad.containsKey(nombre) && (kgDisponibles<kgCafeEnCadaVariedad.get(nombre))) {
                error += kgCafeEnCadaVariedad.get(nombre) - kgDisponibles;
            }
        }

        return goal - 1000*error*error;
    }

    // Funcion que se encarga de transformar una lista (cromosoma) en un objeto del tipo SolucionCafes
    @Override
    public SolucionCafes solucion(List<Integer> ls) {
        return SolucionCafes.of_Range(ls);
    }

}
```

1.4.2. SolucionCafes

```java
package _soluciones;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import _datos.DatosCafes;

public class SolucionCafes {

    // Propiedades
    private Double beneficio;
    private Map<String,Integer> solucion;

    // Metodo de factoria
    public static SolucionCafes of_Range(List<Integer> ls) {
        return new SolucionCafes(ls);
    }

    // Constructor
    private SolucionCafes(List<Integer> ls) {
        Double beneficioTotal = 0.;
        Map<String,Integer> mp = new HashMap<>();
        for(int i=0; i<ls.size(); i++) {
            if(ls.get(i) > 0) {
                beneficioTotal += ls.get(i) * DatosCafes.getBeneficioVentaKg(i);
                mp.put(DatosCafes.getVariedades().get(i).nombre(), ls.get(i));
            }
        }

        beneficio = beneficioTotal;
        solucion = mp;
    }

    // Metodo toString que devuelve una cadena con los kg de cada variedad y el beneficio final obtenido
    @Override
    public String toString() {
        return String.format("Variedades = %s; Beneficio = %f", this.solucion, this.beneficio);
    }

}
```

1.4.3. TestCafesAGRange

```java
package ejercicio1;

import java.util.List;

public class TestCafesAGRange {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE  = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType = StoppingConditionFactory.StoppingConditionType.GenerationCount;

        InRangeCafesAG p = new InRangeCafesAG("ficheros/Ejercicio1DatosEntrada1.txt");


        AlgoritmoAG<List<Integer> ,SolucionCafes> ap = AlgoritmoAG.of(p);
        ap.ejecuta();


        System.out.println("================================");
        System.out.println(ap.bestSolution());
        System.out.println("================================");
    }

}
```

1.4.4. Volcado de pantalla

Resultado para los datos de entrada 1

```
================================
Variedades = {P01=10, P03=1, P02=10}; Beneficio = 305.000000
================================
```

Resultado para los datos de entrada 2

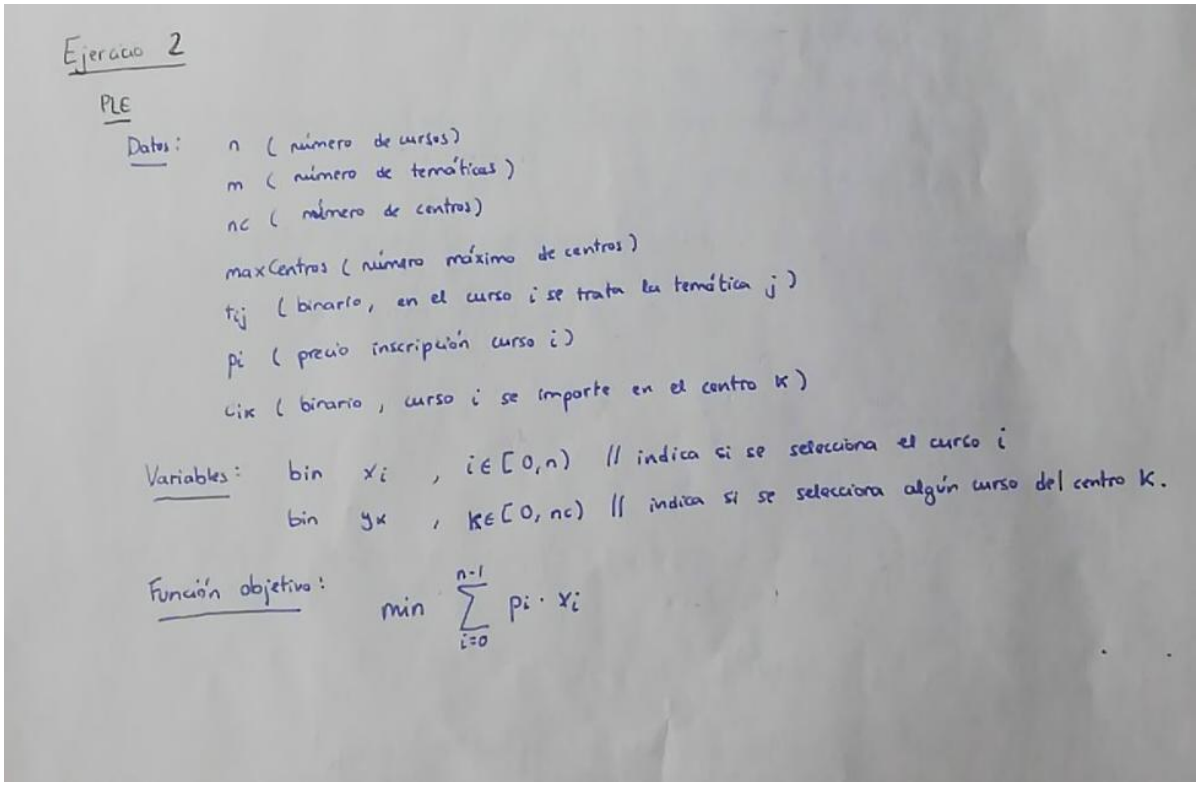
```
================================
Variedades = {P01=15, P03=20, P02=10}; Beneficio = 2000.000000
================================
```

Resultado para los datos de entrada 3

```
================================
Variedades = {P01=30, P02=4, P04=15, P06=100}; Beneficio = 12275.000000
================================
```

2. EJERCICIO

2.1. MODELOS



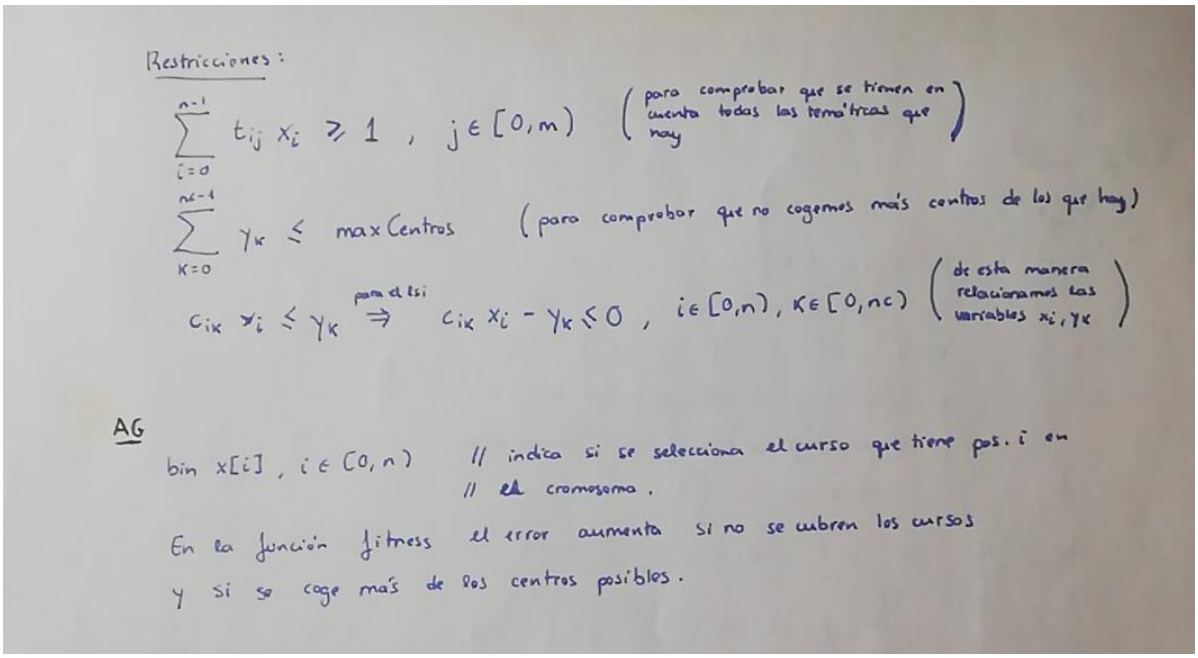
Ejercicio 2

PLE

Datos: n (número de cursos)
m (número de temáticas)
nc (número de centros)
maxCentros (número máximo de centros)
t_{ij} (binario, en el curso i se trata la temática j)
p_i (precio inscripción curso i)
c_{ik} (binario, curso i se imparte en el centro k)

Variables: bin x_i , $i \in [0, n)$ // indica si se selecciona el curso i
bin y_k , $k \in [0, nc)$ // indica si se selecciona algún curso del centro k.

Función objetivo: $\min \sum_{i=0}^{n-1} p_i \cdot x_i$



2.2. DATOS

2.2.1. DatosCursos

```java
package _datos;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import us.lsi.common.Files2;

public class DatosCursos {

    public static Integer maxCentros;
    public static List<Curso> cursos;

    // Funcion que realiza la lectura de fichero y crea un tipo DatosCursos
    public static void iniDatos(String fichero) {
        List<Curso> listaCursos = new ArrayList<>();

        List<String> filas = Files2.linesFromFile(fichero);
        List<String> cursosString = filas.subList(1, filas.size());

        // La primera linea del fichero es el numero maximo de centros
        String primeraLinea = filas.get(0);
        String[] partes = primeraLinea.split("=");
        Integer numMaxCentros = Integer.valueOf(partes[1].trim());
        maxCentros = numMaxCentros;

        // Tranformamos una cadena en el tipo Curso y lo añadimos a la lista de cursos
        for(String linea: cursosString) {
            Curso c = Curso.parseaCurso(linea);
            listaCursos.add(c);
        }
        cursos = listaCursos;

        toConsole();
    }
```

```
    // Para mostar el resultado por pantalla
    private static void toConsole() {
        System.out.println("- Numero maximo de centros leido: " + maxCentros);
        System.out.println("- Cursos leidos: ");
        for(Curso c: cursos) {
            System.out.println("  " + c);
        }
        System.out.println("\n");
    }


    // Funcion que devuelve la lista de centros
    public static List<Curso> getCursos() {
        return cursos;
    }

    // Funcion que devuelve el numero de cursos
    public static Integer getNumCursos() {
        return cursos.size();
    }

    // Funcion que devuelve el numero de tematicas
    public static Integer getNumTematicas() {
        Set<Integer> tematicas = new HashSet<>();
        for(Curso c: cursos) {
            Set<Integer> t = c.tematicas();
            tematicas.addAll(t);
        }
        return tematicas.size();
    }

    // Funcion que devuelve el numero de centros
    public static Integer getNumCentros() {
        Set<Integer> centros = new HashSet<>();
        for(Curso c: cursos) {
            Integer centro = c.centro();
            centros.add(centro);
        }
        return centros.size();
    }

    // Funcion que devuelve el numero maximo de centros
    public static Integer getMaxCentros() {
        return maxCentros;
    }

    // Funcion que devuelve el precio de la inscripcion del curso i
    public static Double getPrecioInscripcion(Integer i) {
        return cursos.get(i).coste();
    }

    // Funcion que devuelve las tematicas del curso i
    public static Set<Integer> getTematicasCurso(Integer i) {
        return cursos.get(i).tematicas();
    }

    // Funcion que devuelve el centro donde se imparte el curso i
    public static Integer getCentroCurso(Integer i) {
        return cursos.get(i).centro();
    }


    // TEST
    public static void main(String[] args) {
        System.out.println("* TEST DatosCursos *\n");
        iniDatos("ficheros/Ejercicio2DatosEntrada1.txt");
    }

}
```

2.2.2. Cursos

```java
package _datos;

import java.util.HashSet;
import java.util.Set;

public record Curso(String id, Set<Integer> tematicas, Double coste, Integer centro) {

    private static Integer numId = 0;

    public static Curso of(Set<Integer> tematicas, Double coste, Integer centro) {
        String id = "C"+numId;
        numId++;
        return new Curso(id, tematicas, coste, centro);
    }

    // Funcion para parsear una cadena a Curso - {1,2,3,4}:10.0:0
    public static Curso parseaCurso(String cadena) {
        String[] partes = cadena.split(":");

        Set<Integer> tematicas = new HashSet<>();
        String conj = partes[0].trim().substring(1, partes[0].trim().length()-1);
        String[] partesConj = conj.split(",");
        for(int i=0; i<partesConj.length; i++) {
            Integer tematica = Integer.valueOf(partesConj[i].trim());
            tematicas.add(tematica);
        }

        Double coste = Double.valueOf(partes[1].trim());

        Integer centro = Integer.valueOf(partes[2].trim());

        return Curso.of(tematicas, coste, centro);
    }

}
```

2.3. PLE

2.3.1. Ejercicio2PLE

```java
package ejercicio2;

import java.io.IOException;
import java.util.HashSet;
import java.util.List;
import java.util.Locale;
import java.util.Set;

import _datos.Curso;
import _datos.DatosCursos;
import us.lsi.gurobi.GurobiLp;
import us.lsi.gurobi.GurobiSolution;
import us.lsi.solve.AuxGrammar;

public class Ejercicio2PLE {

    public static Integer maxCentros;
    public static List<Curso> cursos;


    // Funcion que devuelve el numero de cursos
    public static Integer getNumCursos() {
        return cursos.size();
    }

    // Funcion que devuelve el numero de tematicas
    public static Integer getNumTematicas() {
        Set<Integer> tematicas = new HashSet<>();
        for(Curso c: cursos) {
            Set<Integer> t = c.tematicas();
            tematicas.addAll(t);
        }
        return tematicas.size();
    }
```

```java
    // Funcion que devuelve el numero de centros
    public static Integer getNumCentros() {
        Set<Integer> centros = new HashSet<>();
        for(Curso c: cursos) {
            Integer centro = c.centro();
            centros.add(centro);
        }
        return centros.size();
    }

    // Funcion que devuelve el numero maximo de centros
    public static Integer getMaxCentros() {
        return maxCentros;
    }

    // Funcion que devuelve un 1 si el curso i trata la tematica j, sino un 0
    public static Integer getCursoTieneTematica(Integer i, Integer j) {
        return cursos.get(i).tematicas().contains(j)? 1:0;
    }

    // Funcion que devuelve el precio de la inscripcion del curso i
    public static Double getPrecioInscripcion(Integer i) {
        return cursos.get(i).coste();
    }

    // Funcion que devuelve un 1 si el curso i se imparte en el centro k, sino un 0
    public static Integer getCursoEnCentro(Integer i, Integer k) {
        return cursos.get(i).centro().equals(k)? 1:0;
    }

    // Función que calcula la solucion utilizando gurobi
    public static void ejercicio2_model() throws IOException {
        DatosCursos.iniDatos("ficheros/Ejercicio2DatosEntrada1.txt");

        maxCentros = DatosCursos.maxCentros;
        cursos = DatosCursos.cursos;

        AuxGrammar.generate(Ejercicio2PLE.class, "lsi_models/Ejercicio2.lsi", "gurobi_models/Ejercicio2-1.lp");
        GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio2-1.lp");
        Locale.setDefault(new Locale("en", "US"));
        System.out.println(solution.toString((s,d)->d>0.));
    }


    // TEST
    public static void main(String[] args) throws IOException {
        System.out.println("* TEST Ejercicio2PLE *\n");
        ejercicio2_model();
    }

}
```

2.3.2. Ejercicio2.lsi

```
head section

Integer getNumCursos()
Integer getNumTematicas()
Integer getNumCentros()
Integer getMaxCentros()
Integer getCursoTieneTematica(Integer i, Integer j)
Integer getPrecioInscripcion(Integer i)
Integer getCursoEnCentro(Integer i, Integer k)

Integer n= getNumCursos()
Integer m= getNumTematicas()
Integer nc= getNumCentros()

goal section

min sum(getPrecioInscripcion(i) x[i], i in 0 .. n)

constraints section

sum(getCursoTieneTematica(i,j) x[i], i in 0 .. n) >= 1, j in 1 .. m+1   // porque hay tematicas desde la 1 hasta m no desde la 0
sum(y[k], k in 0 .. nc) <= getMaxCentros()
getCursoEnCentro(i,k) x[i] - y[k] <= 0, i in 0 .. n, k in 0 .. nc

bin

x[i], i in 0 .. n
y[k], k in 0 .. nc
```

2.3.3. Volcado de pantalla

```
El valor objetivo es 15.00
Los valores de la variables
x_0 == 1
x_3 == 1
y_0 == 1
```

Resultado para los datos de entrada 1.
Significa que se seleccionan los cursos 0 y 3 (porque tenemos que x_0 y x_3 son 1), es decir el primer y cuarto curso que aparecen en el txt.
Además, nos dice que se imparte en el centro 0 (ya que nos aparece que y_0 es 1).
El coste (valor objetivo) de apuntarnos a estos cursos es de 8.5 (=1*10+0*3+0*1.5+1*5)

```
El valor objetivo es 8.50
Los valores de la variables
x_0 == 1
x_2 == 1
x_4 == 1
y_0 == 1
y_1 == 1
```

Resultado para los datos de entrada 2.
Significa que se seleccionan los cursos 0, 2, 4, es decir el primero, tercero y quinto que aparecen en el txt.
Además, nos dice que se imparte en el centro 0 y 1.
El coste (valor objetivo) de apuntarnos a estos cursos es de 8.5

```
El valor objetivo es 6.50
Los valores de la variables
x_0 == 1
x_3 == 1
x_7 == 1
y_0 == 1
y_1 == 1
y_2 == 1
```

Resultado para los datos de entrada 3.
Significa que se seleccionan los cursos 0, 3, 7, es decir, el primer, cuarto y octavo que aparecen en el txt.
Además, nos dice que se imparten en los centros 0, 1 y 2.
El coste (valor objetivo) de apuntarnos a esos cursos es de 6.5

2.4. AG

2.4.1. BinCursosAG

```java
package ejercicio2;

import java.util.HashSet;
import java.util.List;
import java.util.Set;

import _datos.DatosCursos;
import _soluciones.SolucionCursos;
import us.lsi.ag.BinaryData;

public class BinCursosAG implements BinaryData<SolucionCursos> {

    // Constructor
    public BinCursosAG(String fichero) {
        DatosCursos.iniDatos(fichero);
    }

    // Longitud que van a tener los cromosomas -> numero de cursos
    @Override
    public Integer size() {
        return DatosCursos.getNumCursos();
    }
```

```
    // Funcion fitness que mide lo bueno que es el cromosoma, para ello penalizaremos si
    // no se cogen todos los cursos y si se imparten en mas centros de los que se pueden impartir
    @Override
    public Double fitnessFunction(List<Integer> ls) {
        Double coste = 0.;
        Double error = 0.;
        Set<Integer> tematicasEnCromosoma = new HashSet<>();
        Set<Integer> centrosEnCromosoma = new HashSet<>();

        // Calculamos el goal y añadimos las tematicas y centros a sus respectivos conjuntos
        for(int i=0; i<size(); i++) {
            if(ls.get(i) > 0) {
                coste += DatosCursos.getPrecioInscripcion(i);
                tematicasEnCromosoma.addAll(DatosCursos.getTematicasCurso(i));
                centrosEnCromosoma.add(DatosCursos.getCentroCurso(i));
            }
        }

        // Sumamos las penalizaciones
        if(tematicasEnCromosoma.size() < DatosCursos.getNumTematicas()) {
            error += DatosCursos.getNumTematicas() - tematicasEnCromosoma.size();
        }
        if(centrosEnCromosoma.size() > DatosCursos.getMaxCentros()) {
            error += centrosEnCromosoma.size() - DatosCursos.getMaxCentros();
        }

        return -coste - 1000*error*error;
    }

    // Funcion que se encarga de transformar una lista (cromosoma) en un objeto del tipo SolucionCursos
    @Override
    public SolucionCursos solucion(List<Integer> ls) {
        return SolucionCursos.of_Bin(ls);
    }

}
```

2.4.2. SolucionCursos

```
package _soluciones;

import java.util.ArrayList;
import java.util.List;

import _datos.DatosCursos;

public class SolucionCursos {

    // Propiedades
    private Double coste;
    private List<String> cursos;

    // Metodo de factoria
    public static SolucionCursos of_Bin(List<Integer> ls) {
        return new SolucionCursos(ls);
    }

    // Constructor
    public SolucionCursos(List<Integer> ls) {
        Double costeTotal = 0.;
        List<String> cursosElegidos = new ArrayList<>();
        for(int i=0; i<ls.size(); i++) {
            if(ls.get(i) > 0) {
                costeTotal += DatosCursos.getPrecioInscripcion(i);
                String curso = DatosCursos.getCursos().get(i).id();
                cursosElegidos.add(curso);
            }
        }

        coste = costeTotal;
        cursos = cursosElegidos;
    }

    // Metodo toString que devuelve una cadena con los cursos y el coste total de elegir esos cursos
    @Override
    public String toString() {
        return String.format("Cursos seleccionados = %s; Coste = %f", this.cursos, this.coste);
    }

}
```

2.4.3. TestCursosAGBinary

```java
package ejercicio2;

import java.util.List;
import java.util.Locale;

import _soluciones.SolucionCursos;
import us.lsi.ag.agchromosomes.AlgoritmoAG;
import us.lsi.ag.agstopping.StoppingConditionFactory;

public class TestCursosAGBinary {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE  = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType = StoppingConditionFactory.StoppingConditionType.GenerationCount;

        BinCursosAG p = new BinCursosAG("ficheros/Ejercicio2DatosEntrada1.txt");


        AlgoritmoAG<List<Integer>,SolucionCursos> ap = AlgoritmoAG.of(p);
        ap.ejecuta();


        System.out.println("================================");
        System.out.println(ap.bestSolution());
        System.out.println("================================");
    }

}
```

2.4.4. Volcado de pantalla

Resultado para los datos de entrada 1:

```
================================
Cursos seleccionados = [C0, C3]; Coste = 15.000000
================================
```

Resultados para los datos de entrada 2:

```
================================
Cursos seleccionados = [C0, C2, C4]; Coste = 8.500000
================================
```

Resultados para los datos de entrada 3:

```
================================
Cursos seleccionados = [C0, C3, C7]; Coste = 6.500000
================================
```

3. EJERCICIO 3

3.1. MODELOS

Ejercicio 3

PLE:

Datos: n (número investigadores)

e (número especialidades)

m (número trabajos)

e_{ik} (binario, investigador i tiene especialidad k)

dd_i (días disponibles del investigador i)

dn_{jk} (días necesarios para el trabajo j de investigador con especialidad k)

c_j (calidad del trabajo j)

Variables: int x_{ij}, $i \in [0,n)$, $j \in [0,m)$ // días que el investigador i dedica // al trabajo j

bin y_j, $j \in [0,m)$ // indica si el trabajo j se realiza

Función objetivo: $\max \sum_{j=0}^{m-1} c_j \cdot y_j$

Restricciones: $x_{ij} \leq dd_i$, $i \in [0,n)$, $j \in [0,m)$ (un investigador no puede dedicarle a un trabajo más días de los que dispone)

$\sum_{j=0}^{m-1} x_{ij} \leq dd_i$, $i \in [0,n)$ (para cada investigador no se supera la cantidad de horas disponibles)

$\sum_{i=0}^{n-1} e_{ik} \cdot x_{ij} = dn_{jk} \cdot y_j$, $j \in [0,m)$, $k \in [0,e)$

(para cada especialidad de cada trabajo, la suma de los días realizados por cada investigador con la especialidad indicada debe ser igual a las horas necesarias)

AG

int x_{ij}, $i \in [0,n)$, $j \in [0,m)$ // indica los días que el invest. i dedica al trabajo j.

En la función fitness el error aumenta si un investigador sobrepasa los días que tiene disponible.

3.2. DATOS

3.2.1. DatosInvestigadores

```java
package _datos;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import us.lsi.common.Files2;

public class DatosInvestigadores {

    public static List<Investigador> investigadores;
    public static List<Trabajo> trabajos;

    // Funcion que realiza la lectura de fichero y crea un tipo DatosInvestigadores
    public static void iniDatos(String fichero) {
        List<Investigador> listaInvestigadores = new ArrayList<>();
        List<Trabajo> listaTrabajos = new ArrayList<>();

        List<String> filas = Files2.linesFromFile(fichero);
        List<String> investigadoresString = filas.subList(1, filas.indexOf("// TRABAJOS"));
        List<String> trabajosString = filas.subList(filas.indexOf("// TRABAJOS") + 1, filas.size());

        // Transformamos una linea en el tipo Investigador y lo añadimos a la lista de investigadores
        for(String linea: investigadoresString) {
            Investigador inv = Investigador.parseaInvestigador(linea);
            listaInvestigadores.add(inv);
        }
        investigadores = listaInvestigadores;

        // Transformamos una linea en el tipo Trabajo y lo añadimos a la lista de trabajos
        for(String linea: trabajosString) {
            Trabajo t = Trabajo.parseaTrabajo(linea);
            listaTrabajos.add(t);
        }
        trabajos = listaTrabajos;

        toConsole();
    }
    // Para mostar el resultado por pantalla
    private static void toConsole() {
        System.out.println("- Investigadores leidos: ");
        for(Investigador inv: investigadores) {
            System.out.println("  " + inv);
        }
        System.out.println("- Trabajos leidos: ");
        for(Trabajo t: trabajos) {
            System.out.println("  " + t);
        }
        System.out.println("\n");
    }


    // Funcion que me devuelve los investigadores
    public static List<Investigador> getInvestigadores() {
        return investigadores;
    }

    // Funcion que me devuelve el numero de investigadores
    public static Integer getNumInvestigadores() {
        return investigadores.size();
    }

    // Funcion que me devuelve el numero de especialidades
    public static Integer getNumEspecialidades() {
        Set<Integer> especialidades = new HashSet<>();
        for(Investigador inv: investigadores) {
            Integer especialidad = inv.especialidad();
            especialidades.add(especialidad);
        }
        return especialidades.size();
    }

    // Funcion que me devuelve los trabajos
    public static List<Trabajo> getTrabajos() {
        return trabajos;
    }
```

```
    // Funcion que me devuelve el numero de trabajos
    public static Integer getNumTrabajos() {
        return trabajos.size();
    }

    // Funcion que me devuelve un 1 si el investigador i tiene la especialidad k, sino devuelve un 0
    public static Integer getInvestigadorTieneEspecialidad(Integer i, Integer k) {
        return investigadores.get(i).especialidad().equals(k)? 1:0;
    }

    // Funcion que me devuelve los dias disponibles del investigador i
    public static Integer getDiasDisponiblesInvestigador(Integer i) {
        return investigadores.get(i).capacidad();
    }

    // Funcion que me devuelve los dias necesarios para el trabajo j del investigador con especialidad k
    public static Integer getDiasNecesariosTrabajosEspecialidad(Integer j, Integer k) {
        return trabajos.get(j).repartos().stream()
                .filter(x -> x.especialidad().equals(k))
                .findFirst()
                .get()
                .dias();
    }

    // Funcion que me devuelve la calidad del trabajo j
    public static Integer getCalidadTrabajo(Integer j) {
        return trabajos.get(j).calidad();
    }


    // TEST
    public static void main(String[] args) {
        System.out.println("* TEST DatosInvestigadores *\n");
        iniDatos("ficheros/Ejercicio3DatosEntrada1.txt");
    }

}
```

3.2.2. Investigador

```
package _datos;

public record Investigador(String nombre, Integer capacidad, Integer especialidad) {

    public static Investigador of(String nombre, Integer capacidad, Integer especialidad) {
        return new Investigador(nombre, capacidad, especialidad);
    }

    // Funcion para parsear una cadena a Investigador
    public static Investigador parseaInvestigador(String cadena) {
        String[] partes = cadena.split(":");


        String nombre = partes[0].trim();


        String[] partes1 = partes[1].trim().split(";");

        String[] partes10 = partes1[0].trim().split("=");
        Integer capacidad = Integer.valueOf(partes10[1].trim());

        String[] partes11 = partes1[1].trim().split("=");
        Integer especialidad = Integer.valueOf(partes11[1].trim());


        return Investigador.of(nombre, capacidad, especialidad);
    }

}
```

3.2.3. Trabajo

```
package _datos;

import java.util.ArrayList;

public record Trabajo(String nombre, Integer calidad, List<TuplaEj3> repartos) {

    public static Trabajo of(String nombre, Integer calidad, List<TuplaEj3> repartos) {
        return new Trabajo(nombre, calidad, repartos);
    }

    // Funcion para parsear una cadena a Trabajo
    public static Trabajo parseaTrabajo(String cadena) {
        String[] partes = cadena.split("->");


        String nombre = partes[0].trim();


        String[] partes1 = partes[1].trim().split(";");

        String[] partes10 = partes1[0].trim().split("=");
        Integer calidad = Integer.valueOf(partes10[1].trim());

        String cadenaRepartos = partes1[1].trim();
        List<TuplaEj3> repartos = listaRepartos(cadenaRepartos);

        return Trabajo.of(nombre, calidad, repartos);
    }

    // Funcion auxiliar para parsear la lista de repartos
    private static List<TuplaEj3> listaRepartos(String cadena) {
        List<TuplaEj3> res = new ArrayList<>();
        String[] partes = cadena.split("=");

        String[] partes1 = partes[1].trim().split(",");
        for(String cadenaTupla: partes1) {
            TuplaEj3 tupla = TuplaEj3.parseaTuplaEj3(cadenaTupla.trim());
            res.add(tupla);
        }

        return res;
    }

}
```

3.2.4. TuplaEj3

```
package _datos;

public record TuplaEj3(Integer especialidad, Integer dias) {

    public static TuplaEj3 of(Integer especialidad, Integer dias) {
        return new TuplaEj3(especialidad, dias);
    }

    // Funcion para parsear una cadena a una tupla formada por especialidad
    // y numero de dias necesarios de esa especialidad
    public static TuplaEj3 parseaTuplaEj3(String cadena) {
        String cadenaTupla = cadena.substring(1, cadena.length()-1);
        String[] partes = cadenaTupla.split(":");
        Integer especialidad = Integer.valueOf(partes[0].trim());
        Integer dias = Integer.valueOf(partes[1].trim());
        return TuplaEj3.of(especialidad, dias);
    }

}
```

3.3. PLE

3.3.1. Ejercicio3PLE

```
package ejercicio3;

import java.io.IOException;
import java.util.HashSet;
import java.util.List;
import java.util.Locale;
import java.util.Set;

import _datos.DatosInvestigadores;
import _datos.Investigador;
import _datos.Trabajo;
import us.lsi.gurobi.GurobiLp;
import us.lsi.gurobi.GurobiSolution;
import us.lsi.solve.AuxGrammar;

public class Ejercicio3PLE {

    public static List<Investigador> investigadores;
    public static List<Trabajo> trabajos;


    // Funcion que me devuelve el numero de investigadores
    public static Integer getNumInvestigadores() {
        return investigadores.size();
    }

    // Funcion que me devuelve el numero de especialidades
    public static Integer getNumEspecialidades() {
        Set<Integer> especialidades = new HashSet<>();
        for(Investigador inv: investigadores) {
            Integer especialidad = inv.especialidad();
            especialidades.add(especialidad);
        }
        return especialidades.size();
    }
    // Funcion que me devuelve el numero de trabajos
    public static Integer getNumTrabajos() {
        return trabajos.size();
    }

    // Funcion que me devuelve un 1 si el investigador i tiene la especialidad k, sino devuelve un 0
    public static Integer getInvestigadorTieneEspecialidad(Integer i, Integer k) {
        return investigadores.get(i).especialidad().equals(k)? 1:0;
    }

    // Funcion que me devuelve los dias disponibles del investigador i
    public static Integer getDiasDisponiblesInvestigador(Integer i) {
        return investigadores.get(i).capacidad();
    }

    // Funcion que me devuelve los dias necesarios para el trabajo j del investigador con especialidad k
    public static Integer getDiasNecesariosTrabajosEspecialidad(Integer j, Integer k) {
        return trabajos.get(j).repartos().stream()
                .filter(x -> x.especialidad().equals(k))
                .findFirst()
                .get()
                .dias();
    }

    // Funcion que me devuelve la calidad del trabajo j
    public static Integer getCalidadTrabajo(Integer j) {
        return trabajos.get(j).calidad();
    }
```

```java
    // Función que calcula la solucion utilizando gurobi
    public static void ejercicio3_model() throws IOException {
        DatosInvestigadores.iniDatos("ficheros/Ejercicio3DatosEntrada1.txt");

        investigadores = DatosInvestigadores.getInvestigadores();
        trabajos = DatosInvestigadores.getTrabajos();

        AuxGrammar.generate(Ejercicio3PLE.class, "lsi_models/Ejercicio3.lsi", "gurobi_models/Ejercicio3-1.lp");
        GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio3-1.lp");
        Locale.setDefault(new Locale("en", "US"));
        System.out.println(solution.toString((s,d)->d>0.));
    }


    // TEST
    public static void main(String[] args) throws IOException {
        System.out.println("* TEST Ejercicio3PLE *\n");
        ejercicio3_model();
    }

}
```

3.3.2. Ejercicio3.lsi

```
head section

Integer getNumInvestigadores()
Integer getNumEspecialidades()
Integer getNumTrabajos()
Integer getInvestigadorTieneEspecialidad(Integer i, Integer k)
Integer getDiasDisponiblesInvestigador(Integer i)
Integer getDiasNecesariosTrabajosEspecialidad(Integer j, Integer k)
Integer getCalidadTrabajo(Integer j)

Integer n= getNumInvestigadores()
Integer e= getNumEspecialidades()
Integer m= getNumTrabajos()

goal section

max sum(getCalidadTrabajo(j) y[j], j in 0 .. m)

constraints section

sum(x[i,j], j in 0 .. m) <= getDiasDisponiblesInvestigador(i), i in 0 .. n
sum(getInvestigadorTieneEspecialidad(i,k) x[i,j], i in 0 .. n) -
    getDiasNecesariosTrabajosEspecialidad(j,k) y[j] = 0, j in 0 .. m, k in 0 .. e

bounds section

x[i,j] <= getDiasDisponiblesInvestigador(i), i in 0 .. n, j in 0 .. m

int

x[i,j], i in 0 .. n, j in 0 .. m

bin

y[j], j in 0 .. m
```

3.3.3. Volcado de pantalla

```
El valor objetivo es 15.00
Los valores de la variables
x_0_0 == 6
x_1_1 == 3
x_2_1 == 8
y_0 == 1
y_1 == 1
```

Resultado para los datos de entrada 1.

La solución nos dice que el investigador 0 realiza 6 días en trabajo 0 (x_0_0==6), el investigador 1 realiza 3 días en el trabajo 1 (x_1_1==3) y el investigador 2 realiza 8 días en el trabajo 1 (x_2_1==8). También nos dice que tanto el trabajo 0 como el 1 se llevan a cabo (y_0==y_1==1).

La calidad de los trabajos (valor objetivo) realizados es 15 (=y_0*5+ y_1*10).

```
El valor objetivo es 16.00
Los valores de la variables
x_0_0 == 2
x_0_1 == 8
x_1_1 == 4
x_2_0 == 5
x_2_1 == 3
y_0 == 1
y_1 == 1
```

Resultado para los datos de entrada 2.

La solución nos dice que el investigador 0 realiza 2 días en el trabajo 0 y 8 días en el trabajo 1, el investigador 1 realiza 4 días en el trabajo 1 y el investigador 2 realiza 5 días en el trabajo 0 y 3 días en el trabajo 1. También sabemos que se realizan el trabajo 0 y 1.

La calidad de los trabajos (valor objetivo) realizados es 16.

```
El valor objetivo es 25.00
Los valores de la variables
x_0_0 == 1
x_1_2 == 5
x_1_4 == 5
x_2_0 == 2
x_2_4 == 1
x_3_4 == 4
x_5_2 == 4
x_6_0 == 1
x_7_2 == 11
x_7_4 == 2
y_0 == 1
y_2 == 1
y_4 == 1
```

Resultado para los datos de entrada 3.

La solución nos dice que el investigador 0 realiza 1 día en el trabajo 0, el investigador 1 realiza 5 días en el trabajo 2 y 5 días en el trabajo 4, el investigador 2 realiza 2 días en el trabajo 0 y 1 días en el trabajo 4, el investigador 3 realiza 4 días en el trabajo 4, el investigador 5 realiza 4 días en el trabajo 2, el investigador 6 realiza 1 días en el trabajo 0 y el investigador 7 realiza 2 días en el trabajo 4. También sabemos que se realizan los trabajos 0, 2 y 4.

La calidad de los trabajos (valor objetivo) realizados es 25.

3.4. AG

3.4.1. InRangeInvestigadoresAG

```java
package ejercicio3;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import _datos.DatosInvestigadores;
import _datos.Investigador;
import _datos.Trabajo;
import _soluciones.SolucionInvestigadores;
import us.lsi.ag.ValuesInRangeData;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

public class InRangeInvestigadoresAG implements ValuesInRangeData<Integer, SolucionInvestigadores> {

    // Constructor
    public InRangeInvestigadoresAG(String fichero) {
        DatosInvestigadores.iniDatos(fichero);
    }

    // Tipo de cromosoma que vamos a usar -> cromosoma de rango
    public ChromosomeType type() {
        return ChromosomeType.Range;
    }

    // Longitud que va a tener el cromosoma -> num investigadores * num trabajos
    @Override
    public Integer size() {
        return DatosInvestigadores.getNumInvestigadores()*
                DatosInvestigadores.getNumTrabajos();
    }

    // Valor minimo que puede tomar un gen del cromosoma, es decir,
    // valor minimo de horas que un investigador le puede dedicar a un trabajo
    @Override
    public Integer min(Integer i) {
        return 0;
    }

    // Valor maximo que puede tomar un gen del cromosoma, es decir,
    // el valor maximo de dias que puede dedicarle un investigador
    // a un trabajo
    @Override
    public Integer max(Integer i) {
        List<Integer> res = new ArrayList<>();
        // Identificamos del investigador que se trata
        Integer numInvestigador = i/DatosInvestigadores.getNumTrabajos();
        Investigador inv = DatosInvestigadores.getInvestigadores().get(numInvestigador);
        // Identificamos el trabajo que se trata
        Integer numTrabajo = i%DatosInvestigadores.getNumTrabajos();
        Trabajo trabajo = DatosInvestigadores.getTrabajos().get(numTrabajo);
        // Obtenemos la especialidad del investigador y vemos cuantos dias son necesarias
        // para ese trabajo con aquella especialidad
        Integer especialidad = inv.especialidad();
        Integer diasNecesarios = trabajo.repartos().stream()
                .filter(x -> x.especialidad().equals(especialidad))
                .findFirst().get()
                .dias();
        // Añadimos tanto los dias disponibles del investigador como los diasNecesarios
        // a una lista y elegimos el menor de ambos, este sera el maximo que estamos buscando
        res.add(inv.capacidad());
        res.add(diasNecesarios);
        return res.stream()
                .min(Comparator.naturalOrder())
                .get() + 1;     // sumamos 1 porque es intervalo abierto
    }
```

```
    // Funcion fitness que sirve para medir lo bueno que es el cromosoma, para ello
    // penalizaremos si nos pasamos de los dias disponibles de un investigador
    @Override
    public Double fitnessFunction(List<Integer> ls) {
        Double goal = 0.;
        Double error = 0.;
        Map<Integer,Integer> diasPorInvestigador = new HashMap<>();

        Integer numTrabajos = DatosInvestigadores.getNumTrabajos();
        Integer numEspecialidades = DatosInvestigadores.getNumEspecialidades();

        // Para penalizar en caso de pasarnos de los dias de trabajo de cada investigador
        for(int i=0; i<size(); i++) {
            // Calculamos los dias que trabaja cada investigador segun el cromosoma
            Integer numInvestigador = i/numTrabajos;
            if(diasPorInvestigador.containsKey(numInvestigador)) {
                diasPorInvestigador.put(numInvestigador, diasPorInvestigador.get(numInvestigador)+ls.get(i));
            } else {
                diasPorInvestigador.put(numInvestigador, ls.get(i));
            }
        }
        // Miramos si cada investigador supera los dias maximos que puede trabajar y,
        // si es asi, penalizaremos los dias que se pase
        for(Integer numInvestigador: diasPorInvestigador.keySet()) {
            Integer capacidad = DatosInvestigadores.getDiasDisponiblesInvestigador(numInvestigador);
            Integer diasTrabajados = diasPorInvestigador.get(numInvestigador);
            if(diasTrabajados > capacidad) {
                error += diasTrabajados-capacidad;
            }
        }

         // Sumamos al objetivo si se realiza el trabajo (se realiza si se cumple las horas necesarias
         // de cada especialidad)
         for(int j=0; j<numTrabajos; j++) {
             Boolean realiza = true;
             // Recorremos todas las especialidades dentro del trabajo
             for(int k=0; k<numEspecialidades; k++) {
                 Integer diasPorEspecialidad = 0;
                 // Recorremos el cromosoma y si ese investigador realiza ese trabajo y tiene la especialidad
                 // apuntamos los dias para comprobar que se cumplen los dias necesarios, sino se cumple para
                 // una sola especialidad significa que el trabajo no se hace
                 for(int i=0; i<size(); i++) {
                     Integer numInvestigador = i/numTrabajos;
                     Integer numTrabajo = i%numTrabajos;
                     if(numTrabajo.equals(j) &&
                             DatosInvestigadores.getInvestigadorTieneEspecialidad(numInvestigador, k).equals(1)) {
                         diasPorEspecialidad += ls.get(i);
                     }
                 }
                 if(diasPorEspecialidad != DatosInvestigadores.getDiasNecesariosTrabajosEspecialidad(j, k)) {
                     realiza = false;
                 }
             }
             // Si realiza el trabajo sumamos la calidad, sino pues a por el siguiente trabajo
             if(realiza) {
                 goal += DatosInvestigadores.getCalidadTrabajo(j);
             }
         }

        return goal - 1000*error*error;
    }

    // Funcion que se encarga de transformar una lista (cromosoma) en un objeto del tipo SolucionInvestigadores
    @Override
    public SolucionInvestigadores solucion(List<Integer> ls) {
        return SolucionInvestigadores.of_Range(ls);
    }

}
```

3.4.2. SolucionInvestigadores

```java
package _soluciones;

import java.util.HashMap;
import java.util.List;
import java.util.Map;

import _datos.DatosInvestigadores;

public class SolucionInvestigadores {

    // Propiedades
    private Map<String, List<Integer>> solucion;
    private Integer goal;

    // Metodo de factoria
    public static SolucionInvestigadores of_Range(List<Integer> ls) {
        return new SolucionInvestigadores(ls);
    }

    // Constructor
    public SolucionInvestigadores(List<Integer> ls) {
        // Hallamos la suma de las calidades de los trabajos que se realizan
        Integer calidades = 0;
        for(int j=0; j<DatosInvestigadores.getNumTrabajos(); j++) {
            Boolean realiza = true;
            for(int k=0; k<DatosInvestigadores.getNumEspecialidades(); k++) {
                Integer diasPorEspecialidad = 0;
                for(int i=0; i<ls.size(); i++) {
                    Integer numInvestigador = i/DatosInvestigadores.getNumTrabajos();
                    Integer numTrabajo = i%DatosInvestigadores.getNumTrabajos();
                    if(numTrabajo.equals(j) &&
                            DatosInvestigadores.getInvestigadorTieneEspecialidad(numInvestigador, k).equals(1)) {
                        diasPorEspecialidad += ls.get(i);
                    }
                }
                if(diasPorEspecialidad != DatosInvestigadores.getDiasNecesariosTrabajosEspecialidad(j, k)) {
                    realiza = false;
                }
            }
            if(realiza) {
                calidades += DatosInvestigadores.getCalidadTrabajo(j);
            }
        }

        // Hallamos las horas que cada investigador dedica a cada trabajo
        Map<String, List<Integer>> mp = new HashMap<>();
        for(int i=0; i<DatosInvestigadores.getNumInvestigadores(); i++) {
            // Dividimos el cromosoma por investigadores
            Integer inicio = i*DatosInvestigadores.getNumTrabajos();
            List<Integer> valoresInvestigador = ls.subList(inicio, inicio+DatosInvestigadores.getNumTrabajos());
            mp.put("INV"+(i+1), valoresInvestigador);
        }

        solucion = mp;
        goal = calidades;
    }

    // Metodo toString que devuelve una cadena con lo que cada investigador dedica a cada trabajo y la calidad obtenida
    @Override
    public String toString() {
        return String.format("Reparto obtenido = %s;\n- Suma de las calidades de los trabajos realizados = %d",
                this.solucion, this.goal);
    }

}
```

3.4.3. TestInvestigadoresAGRange

```java
package ejercicio3;

import java.util.List;
import java.util.Locale;

import _soluciones.SolucionInvestigadores;
import us.lsi.ag.agchromosomes.AlgoritmoAG;
import us.lsi.ag.agstopping.StoppingConditionFactory;

public class TestInvestigadoresAGRange {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE  = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType = StoppingConditionFactory.StoppingConditionType.GenerationCount;

        InRangeInvestigadoresAG p = new InRangeInvestigadoresAG("ficheros/Ejercicio3DatosEntrada1.txt");


        AlgoritmoAG<List<Integer>,SolucionInvestigadores> ap = AlgoritmoAG.of(p);
        ap.ejecuta();


        System.out.println("================================");
        System.out.println(ap.bestSolution());
        System.out.println("================================");
    }

}
```

3.4.4. Volcado de pantalla

Resultado para los datos de entrada 1:

```
================================
Reparto obtenido = {INV3=[0, 8], INV1=[6, 0], INV2=[0, 3]};
- Suma de las calidades de los trabajos realizados = 15
================================
```

Resultado para los datos de entrada 2:

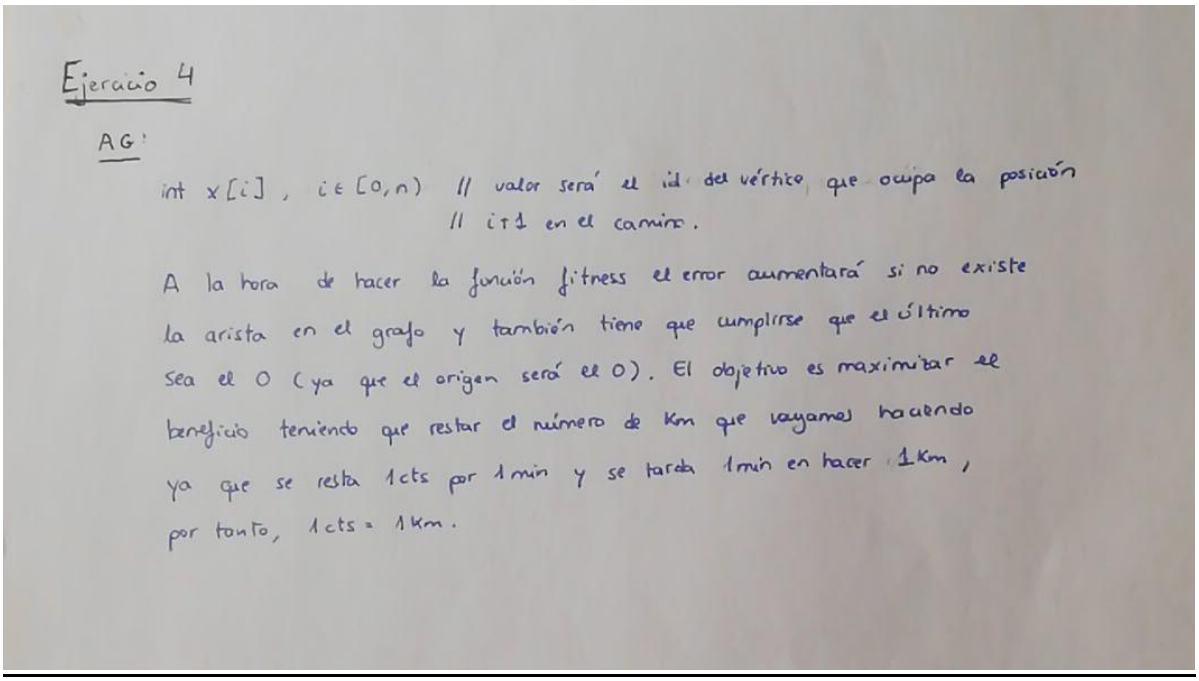
```
================================
Reparto obtenido = {INV3=[5, 3, 0], INV4=[1, 0, 0], INV1=[1, 8, 0], INV2=[0, 4, 0], INV5=[0, 0, 2]};
- Suma de las calidades de los trabajos realizados = 16
================================
```

Resultado para los datos de entrada 3:

```
================================
Reparto obtenido = {INV3=[1, 0, 0, 0, 2], INV4=[1, 0, 0, 0, 3], INV1=[1, 0, 0, 0, 0], INV2=[0, 0, 5, 0, 5], INV7=[1, 0, 0, 0, 0], INV8=
[0, 1, 9, 3, 0], INV5=[0, 0, 6, 1, 1], INV6=[0, 0, 0, 2, 1]};
- Suma de las calidades de los trabajos realizados = 25
================================
```

4. EJERCICIO 4

4.1. MODELO



Ejercicio 4

AG:

int x[i], $i \in [0, n)$ // valor será el id del vértice que ocupa la posición // i+1 en el camino.

A la hora de hacer la función fitness el error aumentará si no existe la arista en el grafo y también tiene que cumplirse que el último sea el 0 (ya que el origen será el 0). El objetivo es maximizar el beneficio teniendo que restar el número de Km que vayamos haciendo ya que se resta 1 cts por 1 min y se tarda 1 min en hacer 1 Km, por tanto, 1 cts = 1 Km.

4.2. DATOS

4.2.1. DatosRepartidor

```
package _datos;

import org.jgrapht.Graph;

import us.lsi.graphs.Graphs2;
import us.lsi.graphs.GraphsReader;

public class DatosRepartidor {

    public static Graph<Cliente, Conexion> grafo;

    // Funcion que realiza la lectura de fichero y crea un tipo DatosCafes
    public static void iniDatos(String fichero) {
        grafo = GraphsReader.newGraph(fichero,
                Cliente::ofFormat, Conexion::ofFormat,
                Graphs2::simpleWeightedGraph);

        toConsole();
    }

    // Para mostar el resultado por pantalla
    private static void toConsole() {
        System.out.println("- Numero de clientes leidos: " + grafo.vertexSet().size());
        System.out.println("- Clientes leidos: ");
        for(Cliente c: grafo.vertexSet()) {
            System.out.println("  " + c);
        }
        System.out.println("\n- Numero de conexiones leidas: " + grafo.edgeSet().size());
        System.out.println("- Conexiones leidas: ");
        for(Conexion co: grafo.edgeSet()) {
            System.out.println("  " + co);
        }
        System.out.println("\n");
    }
```

```
    // Funcion auxiliar que devuelve el cliente i
    private static Cliente getCliente(Integer i) {
        return grafo.vertexSet().stream()
                .filter(x -> x.clienteId().equals(i))
                .findFirst().orElse(null);
    }

    // Funcion que devuelve el numero de vertices
    public static Integer getNumVertices() {
        return grafo.vertexSet().size();
    }

    // Funcion que devuelve el peso de la arista que une el Cliente i y el Cliente j
    public static Double getPesoArista(Integer i, Integer j) {
        return grafo.getEdge(getCliente(i), getCliente(j)).distanciaKm();
    }

    // Funcion que devuelve el beneficio que da el Cliente i
    public static Double getBeneficioCliente(Integer i) {
        return grafo.vertexSet().stream()
                .filter(x -> x.clienteId().equals(i))
                .findFirst().get()
                .beneficio();
    }

    // Funcion que dice si existe una arista en el grafo
    public static Boolean existeArista(Integer i, Integer j) {
        return grafo.containsEdge(getCliente(i), getCliente(j));
    }


    // TEST
    public static void main(String[] args) {
        System.out.println("* TEST DatosRepartidor *\n");
        iniDatos("ficheros/Ejercicio4DatosEntrada1.txt");
    }

}
```

4.2.2. Cliente

```
package _datos;

public record Cliente(Integer clienteId, Double beneficio) {

    public static Cliente of(Integer clienteId, Double beneficio) {
        return new Cliente(clienteId, beneficio);
    }

    public static Cliente ofFormat(String[] datos) {
        Integer clienteId = Integer.valueOf(datos[0].trim());
        Double beneficio = Double.valueOf(datos[1].trim());
        return of(clienteId, beneficio);
    }

}
```

4.2.3. Conexión

```java
package _datos;

public record Conexion(Integer conexionId, Double distanciaKm) {

    private static Integer id = 0;

    public static Conexion of(Double distanciaKm) {
        Integer conexionId = id;
        id++;
        return new Conexion(conexionId, distanciaKm);
    }

    public static Conexion ofFormat(String[] datos) {
        Double distanciaKm = Double.valueOf(datos[2].trim());
        return of(distanciaKm);
    }

}
```

4.3. AG

4.3.1. PermutRepartidorAG

```java
package ejercicio4;

import java.util.List;

import _datos.DatosRepartidor;
import _soluciones.SolucionRepartidor;
import us.lsi.ag.SeqNormalData;
import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

public class PermutRepartidorAG implements SeqNormalData<SolucionRepartidor>{

    // Constructor
    public PermutRepartidorAG(String fichero) {
        DatosRepartidor.iniDatos(fichero);
    }

    // Tipo de cromosoma que vamos a usar -> cromosoma de permutacion
    @Override
    public ChromosomeType type() {
        return ChromosomeType.Permutation;
    }

    // Longitud que van a tener los cromosomas -> numero de vertices del grafo
    @Override
    public Integer itemsNumber() {
        return DatosRepartidor.getNumVertices();
    }
```

```
    // Funcion fitness que sirve para medir lo bueno que es el cromosoma, para ello
    // penalizaremos si no existe la arista. Establecemos el Cliente con id 0 como inicial
    // y el ultimo tiene que ser el 0 si no tambien penalizamos
    @Override
    public Double fitnessFunction(List<Integer> ls) {
        Double goal = 0.;
        Double km = 0.;
        Double error = 0.;

        for(int i=0; i<size(); i++) {
            // Beneficio += beneficio - cada minuto que se pase (ponemos los km, ya que tarda 1 minuto en hacer 1km)
            // Si no existe la arista aumentamos el error
            if(i==0) {
                if(DatosRepartidor.existeArista(0, ls.get(i))) {
                    km += DatosRepartidor.getPesoArista(0, ls.get(i));
                    goal += DatosRepartidor.getBeneficioCliente(ls.get(i)) - km;
                } else {
                    error++;
                }
            } else {
                if(DatosRepartidor.existeArista(ls.get(i-1), ls.get(i))) {
                    km += DatosRepartidor.getPesoArista(ls.get(i-1), ls.get(i));
                    goal += DatosRepartidor.getBeneficioCliente(ls.get(i)) - km;
                } else {
                    error++;
                }
            }
        }
        // Si el ultimo vertice no es el cliente 0, es decir, el inicial, aumentamos el error
        if(!ls.get(ls.size()-1).equals(0)) {
            error++;
        }

        return goal - 10000*error*error;
    }

    // Funcion que se encarga de transformar una lista (cromosoma) en un objeto del tipo SolucionRepartidor
    @Override
    public SolucionRepartidor solucion(List<Integer> ls) {
        return SolucionRepartidor.of_Permut(ls);
    }

}
```

4.3.2. SolucionRepartidor

```
package _soluciones;

import java.util.List;

import _datos.DatosRepartidor;

public class SolucionRepartidor {

    // Propiedades
    private List<Integer> camino;
    private Double distancia;
    private Double beneficio;

    // Metodo de factoria
    public static SolucionRepartidor of_Permut(List<Integer> ls) {
        return new SolucionRepartidor(ls);
    }
```

```
        // Constructor
        public SolucionRepartidor(List<Integer> ls) {
            Double km = 0.;
            Double beneficios = 0.;

            // Recorremos el cromosoma para hallar la distancia recorrida y el beneficio
            for(int i=0; i<ls.size(); i++) {
                // Beneficio += beneficio - cada minuto que se pase (ponemos los km, ya que tarda 1 minuto en hacer 1km)
                // Si no existe la arista aumentamos el error
                if(i==0) {
                    km += DatosRepartidor.getPesoArista(0, ls.get(i));
                    beneficios += DatosRepartidor.getBeneficioCliente(ls.get(i)) - km;
                } else {
                    km += DatosRepartidor.getPesoArista(ls.get(i-1), ls.get(i));
                    beneficios += DatosRepartidor.getBeneficioCliente(ls.get(i)) - km;
                }
            }
            // Añadimos a la posicion 0 el inicial, es decir, el 0
            ls.add(0, 0);

            camino = ls;
            distancia = km;
            beneficio = beneficios;
        }

        // Metodo toString que devuelve una cadena con el orden de clientes que ha seguido,
        // la distancia recorrida en km y el beneficio total
        @Override
        public String toString() {
            return String.format("- Camino desde 0 hasta 0: %s;\n- Kms: %f;\n- Beneficio: %f",
                    this.camino, this.distancia, this.beneficio);
        }

    }
```

4.3.3. TestRepartidorAGPermut

```
package ejercicio4;

import java.util.List;

public class TestRepartidorAGPermut {

    public static void main(String[] args) {
        Locale.setDefault(new Locale("en", "US"));

        AlgoritmoAG.ELITISM_RATE  = 0.10;
        AlgoritmoAG.CROSSOVER_RATE = 0.95;
        AlgoritmoAG.MUTATION_RATE = 0.8;
        AlgoritmoAG.POPULATION_SIZE = 1000;

        StoppingConditionFactory.NUM_GENERATIONS = 1000;
        StoppingConditionFactory.stoppingConditionType = StoppingConditionFactory.StoppingConditionType.GenerationCount;

        PermutRepartidorAG p = new PermutRepartidorAG("ficheros/Ejercicio4DatosEntrada2.txt");


        AlgoritmoAG<List<Integer>,SolucionRepartidor> ap = AlgoritmoAG.of(p);
        ap.ejecuta();


        System.out.println("================================");
        System.out.println(ap.bestSolution());
        System.out.println("================================");
    }

}
```

4.3.4. Volcado de pantalla

Resultado para los datos de entrada 1:

```
================================
- Camino desde 0 hasta 0: [0, 1, 2, 3, 4, 0];
- Kms: 9.000000;
- Beneficio: 981.000000
================================
```

Resultado para los datos de entrada 2:

```
================================
- Camino desde 0 hasta 0: [0, 2, 5, 3, 7, 4, 6, 1, 0];
- Kms: 9.000000;
- Beneficio: 1463.000000
================================
```