Memoria Práctica Individual 4

Ejercicio 1

Un envasador de café mezcla distintas cantidades de n tipos de café para preparar m variedades. Cada kilogramo de café de cada una de las variedades contiene un porcentaje de cada tipo de café. El envasador dispone de una cantidad (en kilos) de cada tipo de café, y obtiene un beneficio distinto por cada kilogramo de cada una de las variedades que produce, ¿cuántos kilogramos de cada variedad de café deben producirse para maximizar los beneficios?

Datos de entrada

- n: entero, tipos de café
- m: entero, variedades de café
- c_j: entero, cantidad (en kg) disponible de café de tipo j, j en [0,n)
- b_i: double, beneficio de venta de la variedad i, i en [0,m)
- p_{ij}: double en [0..1], porcentaje de café de tipo j que se requiere para obtener un kg de la variedad i, i en [0,m), j en [0,n)

Ejemplo de fichero:

```
// TIPOS
C01: kgdisponibles=5;
C02: kgdisponibles=4;
C03: kgdisponibles=1;
C04: kgdisponibles=2;
C05: kgdisponibles=8;
// VARIEDADES
P01 -> beneficio=20; comp=(C01:0.5),(C02:0.4),(C03:0.1);
P02 -> beneficio=10; comp=(C04:0.2),(C05:0.8);
```

- **¿Solución por PLE y AG?**
  - x_i: variable entera, indica cuántos kilogramos de la variedad i serán fabricados, i en [0,n)
- **Función objetivo**
  - Maximizar el beneficio
- **Restricciones:**
  - Las cantidades de cada tipo de café que se utilicen para elaborar las distintas variedades no puede exceder la cantidad total de kg disponibles de cada tipo de café.

Modelo PLE

Función objetivo:

- Maximizar el beneficio

$$max \sum_{i=0}^{m} b_i * x_i$$

Restricciones:

- Las cantidades de cada tipo de café que se utilicen para elaborar las distintas variedades no puede exceder la cantidad total de kg disponibles de cada tipo de café.

$$\sum_{i=0}^{m} p_{ij} * x_i \leq c_j \qquad j \in [0..n]$$

Int $[x_i]$ $i \in [0..m]$

```
head section


Double getPorcentaje(Integer j, Integer i)
Double getBeneficio(Integer i)
Integer getKilogramos(Integer j)
Integer getmVariedades()
Integer getnTipos()

Integer m = getmVariedades()
Integer n = getnTipos()


goal section

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

constraints section

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


int

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

DatosEjercicio1

```
public class DatosEjercicio1 {

    private static SortedMap<String, Integer> tipos;
    private static SortedMap<String, Tupla> variedades;

    public record Tupla(Double beneficio, SortedMap<String, Double> comp) {
        // Map<String,Double>comp

        public static Tupla of(Double beneficio, SortedMap<String, Double> comp) {
            return new Tupla(beneficio, comp);
        }

        public static Tupla parse(String linea) {

            SortedMap<String, Double> comp = new TreeMap<>();

            String[] str = linea.split(";");
            String[] str2 = str[0].split("beneficio=");
            String str3 = str[1].replace("comp=", "").trim();
            String str4 = str3.replace("(", "").replace(")", "");
            String[] str41 = str4.split(",");

            for (int i = 0; i < str41.length; i++) {

                String[] str5 = str41[i].split(":");
                String tipo = str5[0];
                Double porcentaje = Double.parseDouble(str5[1]);
                comp.put(tipo, porcentaje);
            }
            Double beneficio = Double.parseDouble(str2[1]);
            return Tupla.of(beneficio, comp);
        }
    }

    }

    public static void iniDatos(String fichero) {
        tipos = new TreeMap<>();
        variedades = new TreeMap<>();
        Tupla tupla;
        List<String> lineas = Files2.linesFromFile(fichero);
        for (String linea : lineas) {
            if (linea.startsWith("C")) {
                String[] str = linea.split(":");
                String[] str2 = str[1].split("=");
                String[] str3 = str2[1].split(";");
                tipos.put(str[0].trim(), Integer.valueOf(str3[0].trim()));
            } else if (linea.startsWith("P")) {
                String[] str = linea.split(" ->");
                String variedad = str[0];
                tupla = Tupla.parse(linea);
                variedades.put(variedad, tupla);

            }
        }
//      System.out.println(tipos+"\n");
//      System.out.println(variedades);

    }
    // Datos de entrada

    // n entero tipos de cafe
    public static Integer getnTipos() {
        return tipos.keySet().size();
    }

    // m entero variedades de cafe
    public static Integer getmVariedades() {
        return variedades.keySet().size();
    }


    // cantidad disponible (kg) de un tipo de cafe j
    public static Integer getKilogramos(Integer i) {
        // Integer res = tipos.get(s);
        String cafe = tipos.keySet().stream().toList().get(i);
        return tipos.get(cafe);

    }
```

```
    // Beneficio de la venta de la variedad i
    public static Double getBeneficio(Integer i) {

        String nombreVariedad = variedades.keySet().stream().toList().get(i);
        return variedades.get(nombreVariedad).beneficio();
    }

    // Porcentaje de cafe de tipo j que se require para obtener un kg de la variedad
    // i
    public static Double getPorcentaje(Integer j, Integer i) {
        String cafe = tipos.keySet().stream().toList().get(j);
        // Integer intCafe = tipos.get(cafe);

        String variedad = variedades.keySet().stream().toList().get(i);

        // Tupla intVariedad = variedades.get(variedad);
        return variedades.get(variedad).comp().get(cafe) == null ? 0 : variedades.get(variedad).comp().get(cafe);
    }

    public static Double getPorcentaje(Integer j, String i) {
        String cafe = tipos.keySet().stream().toList().get(j);
        String variedad = i;
        return variedades.get(variedad).comp().get(cafe) == null ? 0 : variedades.get(variedad).comp().get(cafe);
    }

    // Obtener los kgMaximos para AG
    public static Integer getKgMaxVariedad(Integer i) {
        Set<String> cafes = tipos.keySet();
        List<Double> res = List2.empty();
        for (String j : cafes) {
            Double div = getKilogramos(getCafe(j)) / getPorcentaje(getCafe(j), i);
            res.add(div);
        }
        return Collections.min(res).intValue();
    }

    public static Set<String> getcafesDeVariedad(Integer i) {
        String variedad = variedades.keySet().stream().toList().get(i);
        Set<String> cafes = variedades.get(variedad).comp().keySet();
        return cafes;
    }




    // Obtener el indice de un cafe dado su nombre
    public static Integer getCafe(String s) {
        List<String> cafes = tipos.keySet().stream().toList();
        return cafes.indexOf(s);

    }

    public static SortedMap<String, Integer> getTipos() {
        return tipos;
    }

    public static SortedMap<String, Tupla> getVariedades() {
        return variedades;
    }

    // Test de la lectura del fichero
    public static void main(String[] args) {
        iniDatos("ficheros_ejercicios/Ejercicio1DatosEntrada1.txt");

    }
}
```

Ejercicio1PLE

```java
package ejercicio1;

import java.io.IOException;

public class Ejercicio1PLE {
    public static SortedMap<String, Integer> tipos;
    public static SortedMap<String, Tupla> variedades;

    // Datos de entrada

    // n entero tipos de cafe
    public static Integer getnTipos() {
        return tipos.keySet().size();
    }

    // m entero variedades de cafe
    public static Integer getmVariedades() {
        return variedades.keySet().size();
    }

    // cantidad disponible (kg) de un tipo de cafe j
    public static Integer getKilogramos(Integer i) {
        // Integer res = tipos.get(s);
        String cafe = tipos.keySet().stream().toList().get(i);
        return tipos.get(cafe);

    }

    // Beneficio de la venta de la variedad i
    public static Double getBeneficio(Integer i) {

        String nombreVariedad = variedades.keySet().stream().toList().get(i);
        return variedades.get(nombreVariedad).beneficio();
    }

    // Porcentaje de cafe de tipo j que se require para obtener un kg de la variedad
    // i
    public static Double getPorcentaje(Integer j, Integer i) {
        String cafe = tipos.keySet().stream().toList().get(j);
        // Integer intCafe = tipos.get(cafe);

        String variedad = variedades.keySet().stream().toList().get(i);

        // Tupla intVariedad = variedades.get(variedad);
        return variedades.get(variedad).comp().get(cafe) == null ? 0 : variedades.get(variedad).comp().get(cafe);
    }




    public static Double getPorcentaje(Integer j, String i) {
        String cafe = tipos.keySet().stream().toList().get(j);
        String variedad = i;
        return variedades.get(variedad).comp().get(cafe) == null ? 0 : variedades.get(variedad).comp().get(cafe);
    }

    // Obtener los kgMaximos para AG
    public static Integer getKgMaxVariedad(Integer i) {
        Set<String> cafes = tipos.keySet();
        List<Double> res = List2.empty();
        for (String j : cafes) {
            Double div = getKilogramos(getCafe(j)) / getPorcentaje(getCafe(j), i);
            res.add(div);
        }
        return Collections.min(res).intValue();

    }

    // Obtener el indice de un cafe dado su nombre
    public static Integer getCafe(String s) {
        List<String> cafes = tipos.keySet().stream().toList();
        return cafes.indexOf(s);

    }
```

```
    public static void ejercicio1_model() throws IOException {

        DatosEjercicio1.iniDatos("ficheros_ejercicios/Ejercicio1DatosEntrada1.txt");

        tipos = DatosEjercicio1.getTipos();
        variedades = DatosEjercicio1.getVariedades();

        // si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para
        // no sobreescribirlo
        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.));
    }

    public static void main(String[] args) throws IOException {
        //
        ejercicio1_model();

    }

}
```

InRangeEjercicio1AG

```
public class InRangeEjercicio1AG implements ValuesInRangeData<Integer,SolucionEjercicio1> {

    public static InRangeEjercicio1AG create(String linea) {
        return new InRangeEjercicio1AG(linea);
    }

    public InRangeEjercicio1AG(String linea) {
        DatosEjercicio1.iniDatos(linea);
    }

    @Override
    public Integer size() {

        return DatosEjercicio1.getmVariedades();
    }

    @Override
    public ChromosomeType type() {
        return ChromosomeType.Range;
    }

    @Override
    public Integer max(Integer i) {
        return DatosEjercicio1.getKgMaxVariedad(i) + 1;
    }

    @Override
    public Integer min(Integer i) {
        return 0;
    }

    public Long k(List<Integer> ls_chrm) {
        return Math2.pow(size(), 10);
    }
```

```
    @Override
    public Double fitnessFunction(List<Integer> value) {
        double goal = 0, error = 0;
        List<Double>kgUsados = new ArrayList<>();

        for(int i =0 ; i< DatosEjercicio1.getnTipos();i++) {
            kgUsados.add(0.);
        }
        for (int i = 0; i < size(); i++) {
            if (value.get(i) > 0) {
                goal += value.get(i) * DatosEjercicio1.getBeneficio(i);
            }
            for (int j = 0; j < DatosEjercicio1.getnTipos(); j++) {
                kgUsados.set(j,(value.get(i)*DatosEjercicio1.getPorcentaje(j, i)+kgUsados.get(j)));
                    if (DatosEjercicio1.getKilogramos(j) < kgUsados.get(j)) {
                        error +=  kgUsados.get(j) - DatosEjercicio1.getKilogramos(j);
                    }
            }
        }
        return goal - k(value) * error;
    }

    @Override
    public SolucionEjercicio1 solucion(List<Integer> value) {
        return SolucionEjercicio1.of_Range(value);
        //return value;
    }
}
```

TestEjercicio1AGRange

```
package ejercicio1;

import java.util.List;

public class TestEjercicio1AGRange {

    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;



        for (int i = 1 ; i<4; i++) {
        InRangeEjercicio1AG p = new InRangeEjercicio1AG("ficheros_ejercicios/Ejercicio1DatosEntrada3.txt");
        AlgoritmoAG<List<Integer>,SolucionEjercicio1> ap = AlgoritmoAG.of(p);
        ap.ejecuta();

        System.out.println("============ Fichero 3 ====================");
        System.out.println(ap.bestSolution());
        System.out.println("==========================================");
        }
    }

}
```

SolucionEjercicio1

```
public class SolucionEjercicio1 {


    public static SolucionEjercicio1 of_Range(List<Integer> value) {
        return new SolucionEjercicio1(value);
    }

    private Double beneficio = 0.;
    private Map<String, Integer> soluciones = new TreeMap<>();


    private SolucionEjercicio1() {
        beneficio = 0.;
        soluciones = Map2.empty();

    }

    private SolucionEjercicio1(List<Integer>ls) {
//      beneficio =0.;
//      soluciones = new TreeMap<>();
        for (int i = 0; i<ls.size();i++) {
//          if (ls.get(i)>0) {
                Integer kg= ls.get(i);
                Double b = DatosEjercicio1.getBeneficio(i)*kg;
                soluciones.put(String.format("P0%d ",i+1 ), kg);
                beneficio += b;

    //      }
        }
    }
    public static SolucionEjercicio1 empty() {
        return new SolucionEjercicio1();
    }

    public String toString() {
        System.out.println("Variedades de cafe seleccionadads: ");
        soluciones.entrySet().stream().forEach(x -> System.out.println(x.getKey() + " : " + x.getValue()));

        return String.format("Beneficio: "+"%s", beneficio);
    }

}
```

SOLUCIONES PLE:

FICHERO 1:

```
Thread count was 1 (of 12 available processors)

Solution count 1: 305

Optimal solution found (tolerance 1.00e-04)
Best objective 3.050000000000e+02, best bound 3.050000000000e+02, gap 0.0000%



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

FICHERO 2:

```
Thread count was 1 (of 12 available processors)

Solution count 1: 2000

Optimal solution found (tolerance 1.00e-04)
Best objective 2.000000000000e+03, best bound 2.000000000000e+03, gap 0.0000%



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

FICHERO 3:

```
Solution count 1: 12275

Optimal solution found (tolerance 1.00e-04)
Best objective 1.227500000000e+04, best bound 1.227500000000e+04, gap 0.0000%



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

:

SOLUCIONES AG:

```
============ Fichero 1 ====================
Variedades de cafe seleccionadads:
P01  : 10
P02  : 10
P03  : 1
Beneficio: 305.0
============================================
============ Fichero 2 ====================
Variedades de cafe seleccionadads:
P01  : 15
P02  : 10
P03  : 20
Beneficio: 2000.0
============================================
============ Fichero 3 ====================
Variedades de cafe seleccionadads:
P01  : 30
P02  : 4
P03  : 0
P04  : 15
P05  : 0
P06  : 100
Beneficio: 12275.0
============================================
```

Ejercicio 2

Existe una oferta de cursos de verano, en cada uno de los cuales se tratan diversas temáticas, algunas de ellas comunes a varios cursos. Además, de cada curso se conoce su precio de inscripción, y el centro donde se imparte.

Se desea conocer la lista de cursos en los que matricularse, teniendo en cuenta que:

a. Entre todos los cursos seleccionados se deben cubrir todas las temáticas,
b. no se pueden elegir cursos de más de un número determinado de centros diferentes (maxCentros),
c. Se debe minimizar el precio total de inscripción.

Datos de entrada

- n: entero, número de cursos
- m: entero, número de temáticas
- nc: entero, número de centros
- maxCentros: entero, número máximo de centros diferentes

- t_{ij}: binaria, en el curso i se trata la temática j, i en [0,n), j en [0,m)
- p_i: entero, precio de inscripción del curso i, i en [0,n)
- c_{ik}: binaria, el curso i se imparte en el centro k, i en [0,n), k en [0,nc)

Ejemplo de fichero:

Max_Centros = 3

{2,6,7}:2.0:2

{7}:3.0:0

{1,5}:5.0:0

{1,3,4}:3.5:2

{3,7}:1.5:1

{4,5,6}:4.5:0

{6,5}:6.0:1

{2,3,5}:1.0:1

Solución

Se quiere saber en qué cursos matricularse.

- **¿Solución por PLE?**
  - x_i: variable binaria, indica si el curso i se selecciona, i en [0,n)
  - y_k: variable binaria, indica si se selecciona algún curso del centro k, k en [0,nc)
- **Función objetivo**
  - Minimizar el precio total de inscripción
- **Restricciones:**
  - Entre todos los cursos seleccionados se deben cubrir todas las temáticas
  - No se pueden elegir cursos de más de un número determinado de centros diferentes
  - Relacionar variables x_i e y_k

- **¿Solución por AG?**
  - x_i: variable binaria, indica si el curso i se selecciona, i en [0,n)
  - Función objetivo
  - Minimizar el precio total de inscripción
- **Restricciones:**
  - Entre todos los cursos seleccionados se deben cubrir todas las temáticas
  - No se pueden elegir cursos de más de un número determinado de centros diferentes

Modelo PLE

Función objetivo:

- Minimizar el precio total de inscripción

$$min \sum_{i=0}^{n} p_i * x_i$$

Restricciones:

- Entre todos los cursos seleccionados se deben cubrir todas las temáticas

$$\sum_{i=0}^{n} t_{ij} * X_i \geq 1 \qquad j \in [1..m+1]$$

- No se pueden elegir cursos de más de un número determinado de centros diferentes

$$\sum_{k=0}^{nc} Y_k \leq maxCentros$$

- Relacionar variables x_i e y_k

$$C_{ik} * X_i - Y_k \leq 0 \; i \in [0..m], \; k \in [0..nc]$$

bin

$[x_i] \; i \in [0..n]$
$[y_k] \; k \in [0..nc]$

```
head section

Integer getnCursos()
Integer getmTematicas()
Integer getnCentros()
Integer getmaxCentros()
Integer tieneTematica(Integer i,Integer j)
Integer imparteCentro(Integer i , Integer k)
Double getPrecio(Integer i)

Integer n = getnCursos()
Integer m = getmTematicas()
Integer nc = getnCentros()
Integer maxCentros = getmaxCentros()


goal section

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


constraints section

//Entre todos los cursos seleccionados se deben cubrir todas las temáticas

sum( tieneTematica(i,j) x[i],i in 0 .. n) >= 1, j in 1 .. m + 1

//No se pueden elegir cursos de más de un número determinado de centros diferentes

sum (y[k], k in 0 .. nc) <= maxCentros

//  Relacionar variables xi e yk

imparteCentro(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
```

DatosEjercicio2

```java
public class DatosEjercicio2 {

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

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

        public static Curso of(Set<Integer> tematicas, Double coste, Integer centro) {
            return new Curso(tematicas, coste, centro);
        }
    }

    public static void iniDatos(String fichero) {
        maxCentros = 0;
        cursos = List2.empty();

        List<String> lineas = Files2.linesFromFile(fichero);
        for (String linea : lineas) {
            if (linea.startsWith("Max_Centros =")) {
                String[] str = linea.split("=");
                maxCentros = Integer.valueOf(str[1].trim());
            } else if (linea.startsWith("{")) {
                Set<Integer> set = Set2.empty();
                String[] str = linea.split(":");
                String[] str2 = str[0].replace("{", "").replace("}", "").trim().split(",");
                for (String i : str2) {
                    set.add(Integer.valueOf(i));
                }
                Double coste = Double.valueOf(str[1].trim());
                Integer centro = Integer.valueOf(str[2].trim());
                cursos.add(Curso.of(set, coste, centro));
            }
        }
    }
```

```
    //Datos de entrada

    public static Integer getmaxCentros() {
        return maxCentros;
    }

    public static List<Curso> getCursos() {
        return cursos;
    }

    public static Integer getnCursos() {
        return cursos.size();
    }

    public static Integer getmTematicas() {
        return cursos.stream()
                .flatMap(c -> c.tematicas().stream())
                .collect(Collectors.toSet())
                .size();
    }

//  public static Integer getmTematicas() {
//      Set<Integer> tematicas = Set2.empty();
//      for (Curso c : cursos) {
//          for(Integer t : c.tematicas()) {
//              tematicas.add(t);
//          }
//      }
//      return tematicas.size();
//  }


    public static Set<Integer> getCentros() {
        return cursos.stream()
                .map(c -> c.centro())
                .collect(Collectors.toSet());
    }

    public static Integer getnCentros() {
        return getCentros().size();
    }
```

```
    //Datos de entrada

    public static Integer getmaxCentros() {
        return maxCentros;
    }

    public static List<Curso> getCursos() {
        return cursos;
    }

    public static Integer getnCursos() {
        return cursos.size();
    }

    public static Integer getmTematicas() {
        return cursos.stream()
                .flatMap(c -> c.tematicas().stream())
                .collect(Collectors.toSet())
                .size();
    }

//  public static Integer getmTematicas() {
//      Set<Integer> tematicas = Set2.empty();
//      for (Curso c : cursos) {
//          for(Integer t : c.tematicas()) {
//              tematicas.add(t);
//          }
//      }
//      return tematicas.size();
//  }


    public static Set<Integer> getCentros() {
        return cursos.stream()
                .map(c -> c.centro())
                .collect(Collectors.toSet());
    }

    public static Integer getnCentros() {
        return getCentros().size();
    }
```

Ejercicio2PLE

```
package ejercicio2;
import java.io.IOException;

public class Ejercicio2PLE {
    private static Integer maxCentros;
    private static List<Curso> cursos;

    //Datos de entrada

        public static Integer getmaxCentros() {
            return maxCentros;
        }

        public static List<Curso> getCursos() {
            return cursos;
        }

        public static Integer getnCursos() {
            return cursos.size();
        }

        public static Integer getmTematicas() {
            return cursos.stream()
                    .flatMap(c -> c.tematicas().stream())
                    .collect(Collectors.toSet())
                    .size();
        }

//      public static Integer getmTematicas() {
//          Set<Integer> tematicas = Set2.empty();
//          for (Curso c : cursos) {
//              for(Integer t : c.tematicas()) {
//                  tematicas.add(t);
//              }
//          }
//          return tematicas.size();
//      }




        public static Set<Integer> getCentros() {
            return cursos.stream()
                    .map(c -> c.centro())
                    .collect(Collectors.toSet());
        }

        public static Integer getnCentros() {
            return getCentros().size();
        }

        public static Double getPrecio(Integer i ) {
            return cursos.get(i).coste();
        }



        public static Set<Integer> getTematicas (){
            return cursos.stream()
                    .flatMap(c -> c.tematicas().stream())
                    .collect(Collectors.toSet());


        }

        //en el curso i se trata la temática j
        //curso tine tematica j? si 1 no 0
        public static Integer tieneTematica(Integer i,Integer j) {
            return cursos.get(i).tematicas().contains(j)?1:0;
        }
        //el curso i se imparte en el centro k
        public static Integer imparteCentro(Integer i , Integer j) {
            return cursos.get(i).centro()==j?1:0;
        }
```

```
    //Obtiene las temáticas del curso i

    public static Set<Integer> getTematicasCruso(Integer i ) {
        return cursos.get(i).tematicas();
    }
    //Obtiene el centro del Curso i

    public static Integer getCentroCurso(Integer i) {
        return cursos.get(i).centro();
    }
    // Test de la lectura del fichero
    public static void main(String[] args) {
        iniDatos("ficheros_ejercicios/Ejercicio2DatosEntrada1.txt");
    }
}
```

BinEjercicio2AG

```
package ejercicio2;

import java.util.HashSet;

public class BinEjercicio2AG<S> implements BinaryData<SolucionEjercicio2> {

    public BinEjercicio2AG(String fichero) {
        DatosEjercicio2.iniDatos(fichero);
    }

    @Override
    public Integer size() {

        return DatosEjercicio2.getnCursos();
    }

    public Long k(List<Integer> ls_chrm) {
        return Math2.pow(size(), 10);
    }

    @Override
    public Double fitnessFunction(List<Integer> value) {
    double goal =0, error=0;
    Set<Integer> tematicas = new HashSet<>(); //tematicas del curso i
    Set<Integer> centros = new HashSet<>();
    Integer m = DatosEjercicio2.getmTematicas();    //Número de temáticas que hay
    Integer nc = DatosEjercicio2.getmaxCentros();//Número de centros distintos

    for (int i = 0; i<value.size();i++) {
        if(value.get(i)>0) {
            goal += DatosEjercicio2.getPrecio(i);
            tematicas.addAll(DatosEjercicio2.getTematicasCruso(i));
            centros.add(DatosEjercicio2.getCentroCurso(i));
        }
    }


    //Penalizo si el número de tematicas que hay en el curso i es menor que todas las tematicas
    if (tematicas.size()<m) {
        error += m - tematicas.size();
    }
    //penalizo si el numero de centros que cojo es mayor que el máximo número de centros
    if(centros.size() > nc) {
        error += centros.size()-nc;
    }
    return -goal -k(value) *error;
    }

    @Override
    public SolucionEjercicio2 solucion(List<Integer> value) {
        return SolucionEjercicio2.of_Range(value);
    }
}
```

TestBinEjercicio2AGBinary

```
package ejercicio2;

import java.util.List;

public class TestEjercicio2AGBinary {

    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;

        for (int i = 1; i< 4; i++) {

        BinEjercicio2AG<Object> p = new BinEjercicio2AG<Object>("ficheros_ejercicios/Ejercicio2DatosEntrada"+i+".txt");


        AlgoritmoAG<List<Integer>,SolucionEjercicio2> ap = AlgoritmoAG.of(p);
        ap.ejecuta();


        System.out.println("========== Fichero: "+ i+" ======================");
        System.out.println(ap.bestSolution());
        System.out.println("=============================================");
        }
    }
}
```

SolucionEjercicio2

```
package _soluciones;

import java.util.List;

public class SolucionEjercicio2 {


    public static SolucionEjercicio2 of_Range(List<Integer> value) {
        return new SolucionEjercicio2(value);
    }
    private Double coste;
    private Set<String> soluciones;


    private SolucionEjercicio2() {
        coste = 0.;
        soluciones = Set2.empty();
    }
    private SolucionEjercicio2(List<Integer>ls) {
        coste =0.;
        soluciones = new TreeSet<>();
        for (int i = 0; i<ls.size();i++) {
            if (ls.get(i)>0) {
                Double c = DatosEjercicio2.getPrecio(i);
                soluciones.add(String.format("S%d", i));
                coste += c;
            }
        }

    }
    public static SolucionEjercicio2 empty() {
        return new SolucionEjercicio2();
    }

    public String toString() {
        System.out.println("Variedades de cafe seleccionadads: ");
        //soluciones.entrySet().stream().forEach(x -> System.out.println(x.getKey() + " : " + x.getValue()));
        System.out.println("Cursos elegidos: "+ soluciones.toString().replace("[", "{").replace("]", "}") );
        return String.format("Coste Total: "+"%s", coste);
    }
```

SOLUCIONES PLE:

FICHERO 1:

```
Thread count was 1 (of 12 available processors)

Solution count 1: 15

Optimal solution found (tolerance 1.00e-04)
Best objective 1.500000000000e+01, best bound 1.500000000000e+01, gap 0.0000%



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

FICHERO 2

```
Solution count 1: 8.5

Optimal solution found (tolerance 1.00e-04)
Best objective 8.500000000000e+00, best bound 8.500000000000e+00, gap 0.0000%



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
```

FICHERO 3:

```
Optimal solution found (tolerance 1.00e-04)
Best objective 6.500000000000e+00, best bound 6.500000000000e+00, gap 0.0000%



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
```

SOLUCIONES AG:

```
========== Fichero: 1 ======================
Variedades de cafe seleccionadads:
Cursos elegidos: {S0, S3}
Coste Total: 15.0
============================================
========== Fichero: 2 ======================
Variedades de cafe seleccionadads:
Cursos elegidos: {S0, S2, S4}
Coste Total: 8.5
============================================
========== Fichero: 3 ======================
Variedades de cafe seleccionadads:
Cursos elegidos: {S0, S3, S7}
Coste Total: 6.5
============================================
```

Ejercicio 3

Existe un grupo de n investigadores para realizar un conjunto de m trabajos durante este curso. De cada investigador se conoce su especialidad (cada especialidad se identifica por un entero), y la cantidad de días disponibles que tiene para avanzar en los trabajos. De cada trabajo se conoce cuántos días de cada especialidad de investigador son necesarios para llevarlo a cabo (por ejemplo 3 días de investigador de especialidad 3 y 2 de especialidad 5), y su calidad (entero en [5,10]). Teniendo en cuenta los días disponibles de los investigadores, puede que no sea posible realizar todos los trabajos, por lo que hay que decidir cuáles llevar a cabo. Se desea conocer cuántos días dedicará cada investigador a cada trabajo de forma que se maximice la suma total de las calidades de los trabajos llevados a cabo.

Datos de entrada

- n: entero, número de investigadores
- e: entero, número de especialidades
- m: entero, número de trabajos
- e_{ik}: binaria, trabajador i tiene especialidad tipo k, i en [0,n), k en [0,e)
- dd_i: entero, días disponibles del trabajador i, i en [0,n)
- dn_{jk}: entero, días necesarios para el trabajo j de investigador con especialidad k, j en [0,m), k en [0,e)
- c_j: entero, calidad trabajo j, j en [0,m)

Ejemplo de fichero:

```
// INVESTIGADORES
INV1: capacidad=1; especialidad=2;
INV2: capacidad=10; especialidad=1;
INV3: capacidad=3; especialidad=0;
INV4: capacidad=4; especialidad=0;
INV5: capacidad=10; especialidad=3;
INV6: capacidad=4; especialidad=3;
INV7: capacidad=1; especialidad=2;
INV8: capacidad=30; especialidad=3;
// TRABAJOS
T01 -> calidad=8; reparto=(0:2),(1:0),(2:2),(3:0);
T02 -> calidad=5; reparto=(0:8),(1:5),(2:4),(3:2);
T03 -> calidad=8; reparto=(0:0),(1:5),(2:0),(3:15);
T04 -> calidad=5; reparto=(0:0),(1:7),(2:8),(3:5);
T05 -> calidad=9; reparto=(0:5),(1:5),(2:0),(3:2);
```

Solución

Cuántos días dedicará cada investigador a cada trabajo.

- **¿Solución por PLE?**
  - x_{ij}: variable entera, días que el investigador i dedica al trabajo j, i en [0,n), j en [0,m)
  - y_j: variable binaria, indica si el trabajo j se realiza, j en [0,m)
- **Función objetivo**
  - Maximizar la suma total de las calidades de los trabajos llevados a cabo.
- **Restricciones:**
  - Para cada trabajador, no se pueden superar su cantidad de días disponibles
  - Relacionar variables x_{ij} e y_j

- ¿Solución por AG?
  - x_{ij}: variable entera, días que el investigador i dedica al trabajo j, i en [0,n), j en [0,m)
- Función objetivo
  - Maximizar la suma total de las calidades de los trabajos llevados a cabo.
- Restricciones:
  - Para cada trabajador, no se pueden superar su cantidad de días disponibles

Modelo PLE

Función objetivo:

- Maximizar la suma total de las calidades de los trabajos llevados a cabo.

$$max \sum_{i=0}^{m} c_j * y_j$$

Restricciones:

- Para cada trabajador, no se pueden superar su cantidad de días disponibles

$$\sum_{j=0}^{m} x_{ij} \leq dd_i \qquad i \in [0..n]$$

- Relacionar variables x_{ij} e y_j

$$\sum_{i=0}^{n} e_{ik} * x_{ij} - dn_{jk} * y_j = 0 \qquad j \in [0..m] \qquad k \in [0..e]$$

- la variable binaria solo puede ser 0 o 1

$$Y_j \leq 1 \; j \in [0..m]$$

int
$[x_{ij}]\; i \in [0..n]\;, j \in [0..m]$
$[y_j]\; k \in [0..m]$

```
head section

Integer getnInvestigadores()
Integer geteEspecialidades()
Integer getmTrabajos()

Integer tieneEspecialidad(Integer i,Integer k)
Integer getCapacidad(Integer i)
Integer diasNecesarios(Integer j, Integer k)
Integer getCalidadTrabajo(Integer j)

Integer n = getnInvestigadores()
Integer e = geteEspecialidades()
Integer m = getmTrabajos()


goal section

//Objetivo: maximizar calidad de los trabajos

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


constraints section
//Para cada trabajador no se pueden superar su cantidad de dias disponibles

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

//Para cada trabajo y cada especialidad la suma de los dias de trabajo  por cada investigador tiene
//que ser igual a los dias necesarios para realizarla

sum(tieneEspecialidad(i,k) x[i,j], i in 0 .. n) - diasNecesarios(j,k) y[j] = 0 ,j in 0 .. m, k in 0 .. e



//la variable binaria solo puede ser 0 o 1

y[j] <= 1, j in 0 .. m


int
x[i,j], i in 0 .. n, j in 0 .. m
y[j], j in 0 .. m
```

DatosEjercicio3

```java
package _datos;

import java.util.Arrays;

public class DatosEjercicio3 {
    private static List<Investigador> investigadores;
    private static List<Trabajo> trabajos;

    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);
        }
    }

    public record Trabajo(String trabajo, Integer calidad, Map<Integer, Integer> reparto) {
        public static Trabajo of(String trabajo, Integer calidad, Map<Integer, Integer> reparto) {
            return new Trabajo(trabajo, calidad, reparto);
        }
    }
```

```
public static void iniDatos(String fichero) {
    investigadores = List2.empty();
    trabajos = List2.empty();

    List<String> lineas = Files2.linesFromFile(fichero);
    for (String linea : lineas) {
        if (linea.startsWith("INV")) {
            String[] str = linea.split(";");
            String[] Inv_Capacidad = str[0].split(":");
            String nombre = Inv_Capacidad[0].trim();
            Integer capacidad = Integer.valueOf(Inv_Capacidad[1].replace("capacidad=", "").trim());
            Integer especialidad = Integer.valueOf(str[1].replace("especialidad=", "").trim());

            investigadores.add(Investigador.of(nombre, capacidad, especialidad));
        } else if (linea.startsWith("T0")) {
            Map<Integer, Integer> reparto = Map2.empty();

            String[] str = linea.split(";");
            String[] nombreCalidad = str[0].split("->");
            String nombre = nombreCalidad[0];
            Integer calidad = Integer.valueOf(nombreCalidad[1].replace("calidad=", "").trim());

            String repartoStr[] = str[1].replace("reparto=", "").trim().replace("(", "").replace(")", "").trim()
                    .split(",");
            List<String> repartoStrList = Arrays.asList(repartoStr);
            for (String cv : repartoStrList) {
                String[] claveValor = cv.split(":");
                reparto.put(Integer.valueOf(claveValor[0].trim()), Integer.valueOf(claveValor[1].trim()));
            }
            trabajos.add(Trabajo.of(nombre, calidad, reparto));
        }
    }
}
```

```
// Metodos de entrada

public static Integer getnInvestigadores() {
    return investigadores.size();
}

public static Integer geteEspecialidades() {
    return investigadores.stream().map(x -> x.especialidad()).collect(Collectors.toSet()).size();
}

public static Integer getmTrabajos() {
    return trabajos.size();
}

// devuelve 1 si el trabajador i tiene la especialidad k
public static Integer tieneEspecialidad(Integer i, Integer k) {
    return investigadores.get(i).especialidad() == k ? 1 : 0;
}

// devuelve los dias disposibles del trabajador j
// capacidad = dias disponibles
public static Integer getCapacidad(Integer i) {
    return investigadores.get(i).capacidad();
}

// dias necesarios para el trabajo j de investigador con especialidad k
public static Integer diasNecesarios(Integer j, Integer k) {
    return trabajos.get(j).reparto().get(k);
}

public static Integer getCalidadTrabajo(Integer j) {
    return trabajos.get(j).calidad();
}

public static List<Investigador> getInvestigadores() {
    return investigadores;
}

public static List<Trabajo> getTrabajos() {
    return trabajos;
}
```

```
    public static Integer getEspecialidad(Integer i) {
        return investigadores.get(i).especialidad();
    }
    public static Trabajo getTrabajo (Integer i) {
        return trabajos.get(i);
    }

    // Test de la lectura del fichero
    public static void main(String[] args) {
        iniDatos("ficheros_ejercicios/Ejercicio3DatosEntrada2.txt");

    }
}
```

Ejercicio3PLE

```
public class Ejercicio3PLE {
    private static List<Investigador> investigadores;
    private static List<Trabajo> trabajos;

    // Metodos de entrada

        public static Integer getnInvestigadores() {
            return investigadores.size();
        }

        public static Integer geteEspecialidades() {
            return investigadores.stream().map(x -> x.especialidad()).collect(Collectors.toSet()).size();
        }

        public static Integer getmTrabajos() {
            return trabajos.size();
        }

        // devuelve 1 si el trabajador i tiene la especialidad k
        public static Integer tieneEspecialidad(Integer i, Integer k) {
            return investigadores.get(i).especialidad() == k ? 1 : 0;
        }

        // devuelve los dias disposibles del trabajador j
        // capacidad = dias disponibles
        public static Integer getCapacidad(Integer i) {
            return investigadores.get(i).capacidad();
        }

        // dias necesarios para el trabajo j de investigador con especialidad k
        public static Integer diasNecesarios(Integer j, Integer k) {
            return trabajos.get(j).reparto().get(k);
        }

        public static Integer getCalidadTrabajo(Integer j) {
            return trabajos.get(j).calidad();
        }
        public static  List<Investigador> getInvestigadores() {
            return investigadores;
        }
```

```
        public static List<Trabajo> getTrabajos() {
            return trabajos;
        }


    public static void ejercicio3_model() throws IOException {

        DatosEjercicio3.iniDatos("ficheros_ejercicios/Ejercicio3DatosEntrada3.txt");

        investigadores = DatosEjercicio3.getInvestigadores();
        trabajos = DatosEjercicio3.getTrabajos();

        // si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para
        // no sobreescribirlo
        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.));
    }

    public static void main(String[] args) throws IOException {
        //
        ejercicio3_model();

    }
}
```

InRangeEjercicio3AG

```
package ejercicio3;

import java.util.Collections;

public class InRangeEjercicio3AG implements ValuesInRangeData<Integer, SolucionEjercicio3> {

    public static InRangeEjercicio3AG create(String linea) {
        return new InRangeEjercicio3AG(linea);
    }

    public InRangeEjercicio3AG(String linea) {
        DatosEjercicio3.iniDatos(linea);
    }

    @Override
    public Integer size() {

        return DatosEjercicio3.getmTrabajos() * DatosEjercicio3.getnInvestigadores();
    }

    @Override
    public ChromosomeType type() {

        return ChromosomeType.Range;
    }

    @Override
    public Integer max(Integer i) {
        List<Investigador> Investigadores = DatosEjercicio3.getInvestigadores();
        List<Integer> ls = Investigadores.stream().map(inv -> inv.capacidad()).collect(Collectors.toList());
        return Collections.max(ls) + 1;
    }

    @Override
    public Integer min(Integer i) {

        return 0;
    }


    public Long k(List<Integer> ls_chrm) {
        return Math2.pow(size(), 20);
    }
```

```
    @Override
    public Double fitnessFunction(List<Integer> value) {
        Double goal = 0., error = 0.;
        Integer nInvestigadores = DatosEjercicio3.getnInvestigadores();
        Integer nTrabajos = DatosEjercicio3.getmTrabajos();
        Integer nEspecialidades = DatosEjercicio3.geteEspecialidades();

        for (int j = 0; j < nTrabajos; j++) {
            Boolean cumpleDias = true;
            Integer IndiceIni = nInvestigadores * j;
            Integer IndiceFinal = IndiceIni + nInvestigadores;
            // indice final = inicial + nIvestigadores
            // Obtenemos sublistas de trabajos para recorrerlas posteriormente
            List<Integer> trabajos = value.subList(IndiceIni, IndiceFinal);
            for (int k = 0; k < nEspecialidades; k++) {
                Integer suma = 0;
                for (int i = 0; i < nInvestigadores; i++) {
                    // si el investigador i tiene la especialidad --> valor chr
                    // si no la tiene suma =0
                    suma += trabajos.get(i) * DatosEjercicio3.tieneEspecialidad(i, k);
                    // System.out.println("investigador "+ i+" con especialidad
                    // "+DatosEjercicio3.getEspecialidad(i)+"\nEspecialidad del bucle "+k+"\ntrabajo
                    // "+j+ "\ndias "+trabajos.get(i)*DatosEjercicio3.getTieneEspecialidad(i,
                    // k)+"\n" );
                }

                // Restricción: si el valor que nos devuelve la suma no es igual al de dias
                // necesarios, ponemos cumpleDias a false y penalizamos
                if (suma != DatosEjercicio3.diasNecesarios(j, k)) {
                    cumpleDias = false;
                }
                if (suma > DatosEjercicio3.diasNecesarios(j, k)) {
                    error += suma - DatosEjercicio3.diasNecesarios(j, k);
                }
            }

            if (cumpleDias) {
                // Si cumpleDias es true significa que se han cumplido todas las sumas por lo
                // tanto el trabajo
                goal += DatosEjercicio3.getCalidadTrabajo(j);
            }




            // Recorremos cada sublista de trabajo, accediendo a cada investigador.

            for (int i = 0; i < nInvestigadores; i++) {
                Integer dias = 0;
                for (int k = i; k < value.size(); k += nInvestigadores) {
                    dias += value.get(k);
                }
                // Penalizamos si los dias del chr se pasan de la capacidad de cada investigador
                if (dias > DatosEjercicio3.getCapacidad(i)) {
                    error += dias - DatosEjercicio3.getCapacidad(i);
                }
            }
        }
        return goal - k(value) * error;
    }

    @Override
    public SolucionEjercicio3 solucion(List<Integer> value) {
        return SolucionEjercicio3.of_Range(value);
    }

}
```

TestEjercicio3

```
package ejercicio3;

import java.util.List;

public class TestEjercicio3AG {
    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;

        for (int i = 1; i < 4; i++) {
            InRangeEjercicio3AG p = new InRangeEjercicio3AG("ficheros_ejercicios/Ejercicio3DatosEntrada"+i+".txt");
            AlgoritmoAG<List<Integer>, SolucionEjercicio3> ap = AlgoritmoAG.of(p);
            ap.ejecuta();

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

}
```

SolucionEjercicio3

```
package _soluciones;

import java.util.ArrayList;
public class SolucionEjercicio3 {

    public static SolucionEjercicio3 of_Range(List<Integer> value) {
        return new SolucionEjercicio3(value);
    }

    private Integer calidad= 0;
    private List<Investigador> investigadores=DatosEjercicio3.getInvestigadores();
    private List<List<Integer>> dias =new ArrayList<>();
    private SortedMap<String, List<Integer>> solucion= new TreeMap<>();

    private SolucionEjercicio3() {
    }

    private SolucionEjercicio3(List<Integer> ls) {
        Integer nIvestigadores = DatosEjercicio3.getnInvestigadores();
        Integer nTrabajos = DatosEjercicio3.getmTrabajos();
        Integer nEspecialidades = DatosEjercicio3.geteEspecialidades();

        dias = new ArrayList<>();
        investigadores =DatosEjercicio3.getInvestigadores();
        solucion = new TreeMap<>();
        calidad = 0;

        //creamos una lista para cada invstigador
        for (int i = 0; i < nIvestigadores; i++) {
            dias.add(new ArrayList<>());
        }

        for (int j = 0; j < nTrabajos; j++) {
            Integer IndiceIni = nIvestigadores * j;
            Integer IndiceFinal = IndiceIni + nIvestigadores;
            List<Integer> trabajos = ls.subList(IndiceIni, IndiceFinal);
```

```
            for (int i = 0; i < nIvestigadores; i++) {
                dias.get(i).add(trabajos.get(i));
                solucion.put(investigadores.get(i).nombre(), dias.get(i));
            }

            Boolean cumpleDias = true;
            for (int k = 0; k < nEspecialidades; k++) {
                Integer suma = 0;
                for (int i = 0; i < nIvestigadores; i++) {
                    suma += trabajos.get(i) * DatosEjercicio3.tieneEspecialidad(i, k);
                }
                if (suma < DatosEjercicio3.diasNecesarios(j, k)) {
                    cumpleDias = false;
                    k = nEspecialidades;
                }
            }

            // Si se realiza el trabajo, se suma su calidad
            if (cumpleDias) {
                calidad += DatosEjercicio3.getCalidadTrabajo(j);
            }
        }

    }
    public static SolucionEjercicio3 empty() {
        return new SolucionEjercicio3();
    }

    public String toString() {

        System.out.println("Reparto obtenido (dias trabajados por cada investigador en cada trabajo):");
        solucion.entrySet().stream().forEach(x -> System.out.println(x.getKey() + " : " + x.getValue()));
        return String.format("\nSUMA DE LAS CALIDADES DE LOS TRABAJOS REALIZADOS: " + calidad);
    }
}
```

SOLUCIONES PLE:

FICHERO 1:

```
Solution count 2: 15 -0

Optimal solution found (tolerance 1.00e-04)
Best objective 1.500000000000e+01, best bound 1.500000000000e+01, gap 0.0000%



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
```

FICHERO 2:

```
Optimal solution found (tolerance 1.00e-04)
Best objective 1.600000000000e+01, best bound 1.600000000000e+01, gap 0.0000%



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
```

FICHERO 3:

```
Solution count 3: 25 16 -0

Optimal solution found (tolerance 1.00e-04)
Best objective 2.500000000000e+01, best bound 2.500000000000e+01, gap 0.0000%



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_4 == 3
x_3_0 == 2
x_3_4 == 2
x_6_0 == 1
x_7_2 == 15
x_7_4 == 2
y_0 == 1
y_2 == 1
y_4 == 1
```

SOLUCIONES AG:

```
============ Fichero 1 ====================
Reparto obtenido (dias trabajados por cada investigador en cada trabajo):
INV1 : [6, 0]
INV2 : [0, 3]
INV3 : [0, 8]

SUMA DE LAS CALIDADES DE LOS TRABAJOS REALIZADOS: 15
============================================
============ Fichero 2 ====================
Reparto obtenido (dias trabajados por cada investigador en cada trabajo):
INV1 : [2, 8, 0]
INV2 : [0, 4, 0]
INV3 : [5, 3, 0]
INV4 : [0, 0, 0]
INV5 : [0, 0, 0]

SUMA DE LAS CALIDADES DE LOS TRABAJOS REALIZADOS: 16
============================================
============ Fichero 3 ====================
Reparto obtenido (dias trabajados por cada investigador en cada trabajo):
INV1 : [1, 0, 0, 0, 0]
INV2 : [0, 0, 5, 0, 5]
INV3 : [2, 0, 0, 0, 1]
INV4 : [0, 0, 0, 0, 4]
INV5 : [0, 0, 0, 0, 0]
INV6 : [0, 0, 4, 0, 0]
INV7 : [1, 0, 0, 0, 0]
INV8 : [0, 0, 11, 0, 2]

SUMA DE LAS CALIDADES DE LOS TRABAJOS REALIZADOS: 25
============================================
```

Ejercicio 4

Un mensajero debe repartir paquetes a una serie de clientes, ubicados en distintas localizaciones, estando las conexiones entre ellas modeladas con un grafo ponderado por la distancia en kilómetros. Determinar en qué orden debe repartir los paquetes si:

- El reparto comienza desde una localización concreta (el almacén) a las 8am.
- Debe visitar todas las localizaciones
- Cada kilómetro recorrido tiene un coste
- La entrega a cada cliente tiene un beneficio distinto, que puede reducirse al aplicar una penalización por el retraso en la recepción del paquete: el repartidor tarda un minuto en recorrer cada kilómetro, y cada cliente penaliza con un céntimo/min de retraso (a contar desde las 8am)

Datos de entrada

Datos de entrada

- n: entero, número de vértices
- E: aristas del grafo
- a: vértice origen, a en [0,n)
- w_{ij}: double, peso de la arista (i,j), i y j en [0,n)
- b_i: double, beneficio del cliente ubicado en el vértice i, i en [0,n)

Ejemplo de fichero:

```
#VERTEX#
0,0.
1,400.0
2,300.0
3,200.0
4,100.0
#EDGE#
0,1,1.0
0,3,100.0
1,2,1.0
1,3,100.0
2,3,1.0
3,4,1.0
2,4,100.0
0,4,5.0
```

Solución

Se quiere saber la secuencia de vértices, en orden, que satisfacen las condiciones del enunciado y maximizan el beneficio.

DatosEjercicio4

```
public class DatosEjercicio4 {

    @SuppressWarnings("exports")
    public static Graph<Cliente, Conexion> grafo;

    public static void iniDatos(String fichero) {
        grafo = GraphsReader.newGraph(fichero, Cliente::ofFormat, Conexion::ofFormat, Graphs2::simpleWeightedGraph);
    }

    public static Integer getnVertices() {
        return grafo.vertexSet().size();
    }

    @SuppressWarnings("exports")
    public static Cliente getCliente(Integer i) {
        Cliente cliente = null;
        List<Cliente> vertices = new ArrayList<>(grafo.vertexSet());
        for (int j = 0; j < vertices.size(); j++) {
            if (vertices.get(j).id() == i) {
                cliente = vertices.get(j);
            }
        }
        return cliente;
    }
    public static Double getBeneficio(Integer i) {
        return getCliente(i).beneficio();
    }

    public static Boolean existeArista(Integer i, Integer j) {
        return grafo.containsEdge(getCliente(i), getCliente(j));
    }

    public static Double getPeso(Integer i, Integer j) {
        return grafo.getEdge(getCliente(i), getCliente(j)).distancia();
    }

//Numero de vertices:grafo.vertexSet().size()
//Vertices: " + grafo.vertexSet()
//Numero de aristas: grafo.edgeSet().size()
//Aristas: " + grafo.edgeSet());

    public static void main(String[] args) {
        iniDatos("ficheros_Ejercicios/Ejercicio4DatosEntrada1.txt");

    }
}
```

Cliente

```java
package utils;

public record Cliente(int id, Double beneficio) {
    public static Cliente of(int id, Double beneficio) {
        return new Cliente(id, beneficio);
    }

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

    @Override
    public String toString() {
        return String.valueOf(this.id());
    }
}
```

conexión

```java
package utils;

public record Conexion(int id, Double distancia) {
    public static int cont;

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

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

    @Override
    public String toString() {
        return "id: " + this.id() + "; distancia: " + this.distancia();
    }
}
```

PermutaEjercicio4AG

```
public class PermutaEjercicio4AG implements SeqNormalData<SolucionEjercicio4> {

    public PermutaEjercicio4AG(String fichero) {
        DatosEjercicio4.iniDatos(fichero);
    }

    @Override
    public ChromosomeType type() {
        return ChromosomeType.Permutation;
    }

    public Long k(List<Integer> ls_chrm) {
        return Math2.pow(size(), 10);
    }

    @Override
    public Double fitnessFunction(List<Integer> value) {
        double goal = 0, error = 0, sumaKms = 0;
        // suma Beneficio - penalización por tiempo
        for (int i = 0; i < value.size(); i++) {
            if (i == 0) {
                // penalizamos a 1 cent por km
                // tenemos que restarle al beneficio la suma de las distancias
                if (DatosEjercicio4.existeArista(0, value.get(i))) {
                    sumaKms += DatosEjercicio4.getPeso(0, value.get(i));
                    goal += DatosEjercicio4.getBeneficio(value.get(i)) - sumaKms;
                }
                // Si no existe la arista penalizamos
                else {
                    error++;
                }
                // si no estamos en el vertice inicial
            } else {
                if (DatosEjercicio4.existeArista(value.get(i - 1), value.get(i))) {
                    sumaKms += DatosEjercicio4.getPeso(value.get(i - 1), value.get(i));
                    goal += DatosEjercicio4.getBeneficio(value.get(i)) - sumaKms;
                } else {
                    error++;
                }
            }
        }

         // La penalización es doble si el ultimo vertice no es el 0
         if (value.get(value.size() - 1) != 0) {
             if (error == 0) {
                 error += 2;
             } else {
                 error = error * 2;
             }
         }
         // sumamos todos los km
         sumaKms = 0.;
         for (int i = 0; i < value.size(); i++) {
             sumaKms += DatosEjercicio4.getBeneficio(value.get(i));
         }
         return goal - k(value) * error;
    }

    @Override
    public SolucionEjercicio4 solucion(List<Integer> value) {
        return SolucionEjercicio4.of_Range(value);
    }

    @Override
    public Integer itemsNumber() {
        return DatosEjercicio4.getnVertices();
    }
}
```

TestEjercicio4AG

```java
package ejercicio4;

import java.util.List;

public class TestEjercicio4 {

    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;
        for (int i = 1; i < 3; i++) {
            PermutaEjercicio4AG p= new PermutaEjercicio4AG("ficheros_ejercicios/Ejercicio4DatosEntrada"+i+ ".txt");
            AlgoritmoAG<List<Integer>, SolucionEjercicio4> ap = AlgoritmoAG.of(p);
            ap.ejecuta();
            System.out.println("============ Fichero "+i+" ====================");
            System.out.println(ap.bestSolution());
            System.out.println("=============================================");
        }
    }
}
```

SolucionEjercicio4AG

```java
public class SolucionEjercicio4 {

    public static SolucionEjercicio4 of_Range(List<Integer> value) {
        return new SolucionEjercicio4(value);
    }

    private Double kms;
    private Double beneficio;
    private List<Cliente> clientes;

    private SolucionEjercicio4() {
        kms = 0.;
        beneficio = 0.;
        clientes = new ArrayList<>();
        //Añadimos el vertice 0
        clientes.add(DatosEjercicio4.getCliente(0));
    }

    private SolucionEjercicio4(List<Integer> value) {
        kms = 0.;
        beneficio = 0.;
        clientes = new ArrayList<>();
        //Añadimos el vertice 0
        clientes.add(DatosEjercicio4.getCliente(0));
        for (int i = 0; i < value.size(); i++) {
            clientes.add(DatosEjercicio4.getCliente(value.get(i)));
            //si el vertice es 0 y existe una arista del 0 al vertice sumamos la distancia y la restamos al beneficio
            if (i == 0) {
                if (DatosEjercicio4.existeArista(0, value.get(i))) {
                    kms += DatosEjercicio4.getPeso(0, value.get(i));
                    beneficio += DatosEjercicio4.getBeneficio(value.get(i)) - kms;
                }
            } else {
                if (DatosEjercicio4.existeArista(value.get(i - 1), value.get(i))) {
                    kms += DatosEjercicio4.getPeso(value.get(i - 1), value.get(i));
                    beneficio += DatosEjercicio4.getBeneficio(value.get(i)) - kms;
                }
            }
        }
    }

    public static SolucionEjercicio4 empty() {
        return new SolucionEjercicio4();
    }


    public String toString() {
        List<Integer> vertices = clientes.stream().map(c -> c.id()).toList();
        return "Camino desde 0 hasta 0:\n" + vertices + "\nKms: " + kms + "\nBeneficio: " + beneficio;
    }

}
```

SOLUCIONES AG:

```
============ Fichero 1 ====================
Camino desde 0 hasta 0:
[0, 1, 2, 3, 4, 0]
Kms: 9.0
Beneficio: 981.0
============================================
============ Fichero 2 ====================
Camino desde 0 hasta 0:
[0, 2, 5, 3, 7, 4, 6, 1, 0]
Kms: 9.0
Beneficio: 1463.0
============================================
```