

Memoria-PI4-wuolah.pdf



Sir_Malvado



Análisis y Diseño de Datos y Algoritmos



2º Grado en Ingeniería Informática - Tecnologías Informáticas



Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla



**¡GAN A UNA BECA PARA
ESTUDIAR EN AUSTRALIA GRATIS!**
¡Escanea el código QR y participa en el sorteo!



**¿NO SABES QUÉ HACER CON TU VIDA? ¡VETE A AUSTRALIA!
TAMPOCO SABRÁS QUÉ HACER, ¡PERO ESTARÁS EN AUSTRALIA!**

**Encuentra el trabajo
de tus sueños**

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!



MEMORIA PI4



WUOLAH

Reservados todos los derechos.
No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

ÍNDICE

Contenido

ÍNDICE.....	1
Ejercicio 1	2
Modelo lsi.....	2
DatosCafetera	3
Ejercicio1PLE	6
SolucionCafetera.....	7
InRangeCafeteraAG.....	9
TestCafeteraAG	11
Ejercicio 2	12
Modelo lsi.....	12
DatosCursos.....	13
Ejercicio2PLE	16
SolucionCursos	17
InRangeCursos.....	18
TestCursoRangeAG.....	20
Ejercicio 3	21
Modelo lsi.....	21
DatosInvestigacion	22
Ejercicio3PLE	26
SolucionInvestigacionAG.....	27
InvestigacionAG.....	29
Ejercicio 4	31
DatosTrayectoria.....	31
SolucionTrayectoria.....	33
Ejercicio4AG	34
TestEjercicio4AG	36



MÁNCHATE LOS DEDOS CON CHEETOS

Y ESCAQUÉATE
DE TOMAR APUNTES HOY



Ejercicio 1

Modelo lsi

```
head section

Integer getTipos()
Integer getVariedades()
Integer getKgDisponibles(Integer j)
Integer getBeneficio(Integer i)
Double getPorcentaje(Integer j, Integer i)

Integer n = getTipos()
Integer m = getVariedades()

// Maximizar el beneficio
goal section

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

constraints section

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

//bounds section

//x[j] <= getKgDisponibles(j), j in 0 .. m

int
x[j], j in 0 .. m
```

DatosCafetera

```
public static record TipoCafe(String id, Integer kgDisponibles) {
    public static TipoCafe create(String id, Integer kg) {
        return new TipoCafe(id, kg);
    }
}

public static record Variedad(String id, Integer beneficio,
    Map<String, Double> mezcla) {

    public static Variedad create(String id, Integer beneficio,
        Map<String, Double> mezcla) {
        return new Variedad(id, beneficio, mezcla);
    }
}

private static List<TipoCafe> tipos;
private static List<Variedad> variedades;

public static void iniDatos_v2(String fichero) {
    List<String> ls = Files2.LinesFromFile(fichero);
    List<TipoCafe> lt = new ArrayList<>();
    List<Variedad> lv = new ArrayList<>();
    List<Map<String, Double>> lm = new ArrayList<>();

    for(String linea: ls) {
        if(linea.startsWith("C")){
            String[] l = linea.split(":");
            String nombre_tipo = l[0];
            String[] kg = l[1].split("=");
            Integer kg_dispo = Integer.parseInt(kg[1].substring(0, kg[1].length()-1));
            TipoCafe t = TipoCafe.create(nombre_tipo, kg_dispo);
            lt.add(t);
        }else if (linea.startsWith("P")) {
            String[] l = linea.split(" -> ");
            String variedad = l[0];

            String[] partes = l[1].split(";");
            String[] beneficio = partes[0].split("=");
            Integer b = Integer.parseInt(beneficio[1]);

            String[] reparto = partes[1].split("=");
            String r = reparto[1].substring(0, reparto[1].length());
            String[] r_m = r.split(",");
            Map<String, Double> map = new HashMap<>();
            for (String relacion : r_m) {
                String[] rel = relacion.split(":");
                String tipo = rel[0].substring(1, rel[0].length());

                Double consumo = Double.parseDouble(rel[1].substring(0, rel[1].length()-1));
                map.put(tipo, consumo);
            }
        }
    }
}
```

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!

```
        Variedad v = Variedad.create(variedad, b, map);
        lv.add(v);
        lm.add(map);
    }
}

tipos = lt;
variedades = lv;

}

public static List<TipoCafe> getListaTipos(){
    return tipos;
}

public static List<Variedad> getListaVariedades(){
    return variedades;
}

// n: entero, tipos de café

public static Integer getTipos() {
    return tipos.size();
}

// m: entero, variedades de café

public static Integer getVariedades() {
    return variedades.size();
}

// cj: entero, cantidad (en kg) disponible de café de tipo j, j en [0,n)

public static Integer getKgDisponibles(Integer j) {
    return tipos.get(j).kgDisponibles();
}

// bi: double, beneficio de venta de la variedad i, i en [0,m)

public static Integer getBeneficio(Integer i) {
    return variedades.get(i).beneficio();
}
```



WUOLAH

```

// bi: double, beneficio de venta de la variedad i, i en [0,m)

public static Integer getBeneficio(Integer i) {
    return variedades.get(i).beneficio();
}

/* pij: 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]*/

public static Double getPorcentaje(Integer j, Integer i) {
    Map<String, Double> mez = variedades.get(j).mezcla();
    String t = tipos.get(i).id();
    Double porcentaje = 0.0;
    if(mez.containsKey(t)) {
        porcentaje = mez.get(t);
    }
    return porcentaje;
}

```

Ejercicio1PLE

```
public static void ejercicio1_model() throws IOException {
    DatosCafetera.iniDatos_v2("ficheros/Ejercicio1DatosEntrada1.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    AuxGrammar.generate(DatosCafetera.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 ejercicio1_model_2() throws IOException {
    DatosCafetera.iniDatos_v2("ficheros/Ejercicio1DatosEntrada2.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    AuxGrammar.generate(DatosCafetera.class,"lsi_models/Ejercicio1.lsi","gurobi_models/Ejercicio1-2.lp");
    GurobiSolution solution = GurobiLP.gurobi("gurobi_models/Ejercicio1-2.lp");
    Locale.setDefault(new Locale("en", "US"));
    System.out.println(solution.toString((s,d)->d>0.));
}

public static void ejercicio1_model_3() throws IOException {
    DatosCafetera.iniDatos_v2("ficheros/Ejercicio1DatosEntrada3.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    AuxGrammar.generate(DatosCafetera.class,"lsi_models/Ejercicio1.lsi","gurobi_models/Ejercicio1-3.lp");
    GurobiSolution solution = GurobiLP.gurobi("gurobi_models/Ejercicio1-3.lp");
    Locale.setDefault(new Locale("en", "US"));
    System.out.println(solution.toString((s,d)->d>0.));
}

public static void main(String[] args) throws IOException{
    // TODO Auto-generated method stub
    ejercicio1_model();
    ejercicio1_model_2();
    ejercicio1_model_3();
}
```

SolucionCafetera

```
/* Se crea la solución a partir de una función privada
 * que cree el concepto de solución que queremos
 * para el ejercicio
 */
public static SolucionCafetera create(List<Integer> ls) {
    return new SolucionCafetera(ls);
}

private Integer beneficio;
private Map<Variedad, Double> mezclas;
private Integer peso_gen;
private List<Integer> lista_pesos;

private SolucionCafetera() {
    beneficio = 0;
    mezclas = new HashMap<>();
    peso_gen = 0;
}

/*Esta será la función que usemos para crear la solución
 *
 * Las restricciones asociadas a la solución se harán en el fitness*/
private SolucionCafetera(List<Integer> ls) {
    beneficio = 0;
    peso_gen = 0;
    mezclas = new HashMap<>();
    lista_pesos = List2.ofTam(0, ls.size());
    for (int variedad_j = 0; variedad_j < ls.size(); variedad_j++) {
        if(ls.get(variedad_j)>0) {
            Integer value_gen = ls.get(variedad_j);
            Variedad variety = DatosCafetera.getVariedad(variedad_j);

            peso_gen = value_gen;
            lista_pesos.set(variedad_j, peso_gen);
            mezclas.put(variety, value_gen.doubleValue());

            /*
             * El beneficio obtenido es el valor del gen de la mezcla
             * multiplicado por el beneficio de la misma
             * ya que el valor del gen es cuantas veces
             * se ha hecho la mezcla i
             */
            beneficio += variety.beneficio() * value_gen;
        }
    }
}
```

Encuentra el trabajo de tus sueños



Escanéame y obtén más info!!



```
public String toString() {  
    //return String.format("Solucion = %s; Beneficio = %d", mezclas, beneficio);  
    String msg = "";  
    msg += "Variedades de cafe seleccionado:\n";  
    for (int i = 0; i < lista_pesos.size(); i++) {  
        msg += "P" + (i+1) + ":" + lista_pesos.get(i)+"\n";  
    }  
    msg += "Beneficio: " + beneficio;  
    return msg;  
}
```



Reservados todos los derechos.

No se permite la explotación económica ni la transformación de esta obra. Queda permitida la impresión en su totalidad.

WUOLAH

InRangeCafeteraAG

```
/*
 * Inicializamos los datos de la cafetera
 */
public InRangeCafeteraAG(String linea) {
    DatosCafetera.iniDatos_v2(linea);
}

/*
 * El tamaño del cromosoma es la cantidad de mezclas posible
 */
@Override
public Integer size() {
    // TODO Auto-generated method stub
    return DatosCafetera.getVariedades();
}

@Override
public ChromosomeType type() {
    // TODO Auto-generated method stub
    return ChromosomeType.Range;
}

@Override
public Double fitnessFunction(List<Integer> value) {
    // TODO Auto-generated method stub
    // Goal el beneficio
    Double goal = 0.0;

    // Peso de la variedad en gen
    Map<String, Double> lista_pesos = new HashMap<>();

    // Penalización por fallo
    Double error = 0.0;

    for (int i = 0; i < size(); i++) {
        if(value.get(i)>0) {
            goal += value.get(i) * DatosCafetera.getVariedad(i).beneficio();
            Variedad variety = DatosCafetera.getVariedad(i);
            Map<String, Double> cafes_en_variedad = variety.mezcla();
            for (Map.Entry<String, Double> entry : cafes_en_variedad.entrySet()) {
                String id = entry.getKey();
                Double peso = entry.getValue();
                // Cogemos el porcentaje y lo multiplicamos por el valor del gen
                peso = peso*value.get(i);
                lista_pesos.put(id, peso);
            }
        }
    }

    error = penalizacion(lista_pesos);

    // Queremos maximizar beneficio por eso el goal en positivo
    return goal -1000*error;
}
```

```

public static Double penalizacion(Map<String,Double> m) {

    Double penalizacion = 0.0;
    for (Map.Entry<String, Double> entry : m.entrySet()) {
        for (TipoCafe t : DatosCafetera.getListaTipos()) {
            if(t.id() == entry.getKey()) {
                // Pasamos a double para que no haya problemas
                Double kg_disponibles = t.kgDisponibles().doubleValue();
                if(kg_disponibles < entry.getValue()) {
                    /*
                     * Si hay más cantidad de lo que viene predefinido
                     * como kg disponible se suma el resto a la penalización
                     */
                    Double sobrante = entry.getValue() - kg_disponibles;
                    penalizacion+= sobrante;

                    /*
                     * Todo esto se hace iterando tipo por tipo
                     */
                }
            }
        }
    }
    return penalizacion;
}

@Override
public SolucionCafetera solucion(List<Integer> value) {
    // TODO Auto-generated method stub
    return SolucionCafetera.create(value);
}

/* Tamaño máximo del valor que puede tomar es el mínimo
 * de la mezcla que se pueda hacer de todas las del conjunto
 *
 * Mirar la subfunción de DatosCafetera a la que refiere
 *
 * Como el dato es double lo pasamos a Integer*/
@Override
public Integer max(Integer i) {
    // TODO Auto-generated method stub

    Integer variedad_j = i;

    Double kg_fabricables = Double.MAX_VALUE;

    for(Integer cafe_i = 0; cafe_i < DatosCafetera.getTipos(); cafe_i++) {
        Double maximo_cafe_i = DatosCafetera.getKgDisponibles(cafe_i)/DatosCafetera.getPorcentaje(variedad_j, cafe_i);
        if(maximo_cafe_i< kg_fabricables) {
            kg_fabricables = maximo_cafe_i;
        }
    }

    return kg_fabricables.intValue()+1;
}

@Override
public Integer min(Integer i) {
    // TODO Auto-generated method stub
    return 0;
}

```

TestCafeteraAG

```
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 = 0; i < 3; i++) {

        InRangeCafeteraAG p = new InRangeCafeteraAG("ficheros/Ejercicio1DatosEntrada"+(i+1)+".txt");

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

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

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!

Modelo lsi

```
head section
Integer getCursos()
Integer getTemasTotal()
Integer getNumCentros()
Integer getMaxCentros()
Integer getEstaTemaCurso(Integer i, Integer j)
Double getPrecioCurso(Integer i)
Integer getEstaCentroCurso(Integer i, Integer k)
Integer getCentro(Integer i)

Integer n = getCursos()
Integer nc = getNumCentros()
Integer m = getTemasTotal()

goal section
min sum(getPrecioCurso(i) *x[i], i in 0 .. n)
constraints section
sum(getEstaTemaCurso(i,j) *x[i], i in 0 .. n) >= 1, j in 0 .. m
getEstaCentroCurso(i , k) *x[i] - y[k] <= 0, i in 0 .. n, k in 0 .. nc
sum(y[k], k in 0 .. nc) <= getMaxCentros()

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



WUOLAH

DatosCursos

```
/* Datos:  
 *  
 * • n: entero, número de cursos  
 * • m: entero, número de temáticas  
 * • nc: entero, número de centros  
 * • maxCentros: entero, número máximo de centros diferentes  
 * • tij: binaria, en el curso i se trata la temática j, i en [0,n), j en [0,m)  
 * • pi: real, precio de inscripción del curso i, i en [0,n)  
 * • cik: binaria, el curso i se imparte en el centro k, i en [0,n), k en [0,nc)  
 */  
  
public static record Cursos(List<Integer> temas, Double precio, Integer centro) {  
    public static Cursos create(List<Integer> temas, Double precio, Integer centro) {  
        return new Cursos(temas, precio, centro);  
    }  
  
    public static Double getPrecio(Cursos c) {  
        return c.precio();  
    }  
  
    public static Integer getCentro(Cursos c) {  
        return c.centro();  
    }  
  
    public static List<Integer> getTemasCurso(Cursos c){  
        return c.temas();  
    }  
}  
  
/*  
 * Hay que igualar las constantes en valores ya finalizados,  
 * si lo metemos en el bucle  
 * como si fuese una lista vacía normal nos salta el nullpointer exception  
 */  
private static Integer max_centros;  
private static List<Integer> temas;  
private static Set<Integer> temas_s;  
private static List<Cursos> lista_cursos;  
// Lista de centros  
private static List<Integer> centros;
```

```

public static void iniDatos(String fichero) {
    List<String> ls = Files2.linesFromFile(fichero);
    List<Integer> lista_temas_temp = new ArrayList<>();
    List<Cursos> ls_cursos_temp = new ArrayList<>();
    List<Integer> lc = new ArrayList<>();
    Set<Integer> s_temp = new HashSet<>();
    for(String linea: ls) {
        if(linea.startsWith("Max_Centros")) {
            String[] l = linea.split("=");
            max_centros = Integer.parseInt(l[1].trim());
        }else {
            /*
             * Con el split separamos
             * l[0]: la lista de cursos
             * l[1]: precio del curso
             * l[2]: centro
             */
            //l[0].replace("{", "").replace("}", "");
            String[] l = linea.split(":");
            //System.out.println(l[0]);
            String temas_curso = l[0].substring(1, l[0].length() -1);
            /*
             * Vamos a hacer lo mismo que con el split anterior
             */
            String[] tema = temas_curso.split(",");
            List<Integer> lista_temas_curso_temp = new ArrayList<>();
            for (String t : tema) {
                Integer tema_int = Integer.parseInt(t);
                /*
                 * Si es la primera vez que aparece el tema
                 * lo añadimos a la variable global*/
                if(!lista_temas_temp.contains(tema_int)) {
                    lista_temas_temp.add(tema_int);
                }
                /*
                 */
                lista_temas_curso_temp.add(tema_int);
            }
            Cursos c = Cursos.create(lista_temas_curso_temp, Double.parseDouble(l[1]),
                Integer.parseInt(l[2]));
            if(!lc.contains(Integer.parseInt(l[2]))) {
                lc.add(Integer.parseInt(l[2]));
            }
            ls_cursos_temp.add(c);
        }
    }
    lista_cursos = ls_cursos_temp;
    temas = lista_temas_temp;
    centros = lc;
    temas_s = s_temp;
    /*
    System.out.println(temas);
    System.out.println(lista_cursos);
    System.out.println(max_centros);
    */
}

```

```

// n: entero, número de cursos

public static Integer getCursos() {
    return lista_cursos.size();
}

// m: entero, número de temáticas

public static Integer getTemasTotal() {
    // Hemos cambiado temas que es list por temas_s que es set para ver si ejecuta la función
    return temas_s.size();
}

// nc: entero, número de centros

public static Integer getNumCentros() {
    return centros.size();
}

// maxCentros: entero, número máximo de centros diferentes

public static Integer getMaxCentros() {
    return max_centros;
}

// tij: binaria, en el curso i se trata la temática j, i en [0,n), j en [0,m)

public static Integer getEstaTemaCurso(Integer i, Integer j) {
    Boolean esta_tema = lista_cursos.get(i).temas().contains(temas.get(j));
    Integer res = 0;
    if(esta_tema) {
        res = 1;
    }
    return res;
}

// pi: real, precio de inscripción del curso i, i en [0,n)

public static Double getPrecioCurso(Integer i) {
    return lista_cursos.get(i).precio();
}

// cik: binaria, el curso i se imparte en el centro k, i en [0,n), k en [0,nc)

public static Integer getEstaCentroCurso(Integer i, Integer k) {
    Boolean esta_centro = lista_cursos.get(i).centro() == k;
    Integer res = 0;
    if(esta_centro) {
        res = 1;
    }
    return res;
}

public Integer getCentro(Integer i) {
    return lista_cursos.get(i).centro();
}
public static Cursos getCurso(Integer i) {
    return lista_cursos.get(i);
}

public static List<Integer> getTemasCurso(Integer i){
    return lista_cursos.get(i).temas();
}

```

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!

Ejercicio2PLE

```
public static void ejercicio2_model() throws IOException {
    DatosCursos.iniDatos("ficheros/Ejercicio2DatosEntrada1.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    System.out.println("");
    AuxGrammar.generate(DatosCursos.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());
}

public static void ejercicio2_model_2() throws IOException {
    DatosCursos.iniDatos("ficheros/Ejercicio2DatosEntrada2.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    System.out.println("");
    AuxGrammar.generate(DatosCursos.class, "lsi_models/ejercicio2.lsi", "gurobi_models/Ejercicio2-2.lp");
    GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio2-2.lp");
    Locale.setDefault(new Locale("en", "US"));
    System.out.println(solution.toString());
}

public static void ejercicio2_model_3() throws IOException {
    DatosCursos.iniDatos("ficheros/Ejercicio2DatosEntrada3.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    System.out.println("");
    AuxGrammar.generate(DatosCursos.class, "lsi_models/ejercicio2.lsi", "gurobi_models/Ejercicio2-3.lp");
    GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio2-3.lp");
    Locale.setDefault(new Locale("en", "US"));
    System.out.println(solution.toString());
}
```



WUOLAH

SolucionCursos

```
public static SolucionCursos create(List<Integer> ls) {
    return new SolucionCursos(ls);
}
// Cursos escogidos o no
private static List<Integer> cursos;
// Los temas que se dan en general
//private static List<Integer> temas;
// Solución el coste de la selección
private static Double coste;

private SolucionCursos() {
    coste = 0.0;
    cursos = new ArrayList<>();
}

private SolucionCursos(List<Integer> ls) {
    coste = 0.0;
    cursos = new ArrayList<>();
    Set<Integer> temas = new HashSet<>();

    for (int i = 0; i < ls.size(); i++) {
        // Mayor igual a 0 porque el valor puede ser 0
        if(ls.get(i)>= 0) {
            Integer escogido = ls.get(i);
            if(escogido == 1) {
                // Si se ha escogido ese curso entonces se suma el coste
                coste += DatosCursos.getCurso(i).precio();
                // Añadimos los temas en un set para que no se repitan
                List<Integer> temas_curso = DatosCursos.getCurso(i).temas();
                for (Integer tema : temas_curso) {
                    temas.add(tema);
                }
                cursos.add(i);
            }
        }
    }
}

public String toString() {
    /*%f sirve para los float*/
    String msg = "";
    List<String> aux = new ArrayList<>();
    for (int i = 0; i < cursos.size(); i++) {
        Integer c = cursos.get(i);
        String curso_s = "s" + c.toString();
        aux.add(curso_s);
    }
    msg += "Cursos elegidos: " + aux + "\n" + "Coste total: " + coste;
    return msg;
    //return String.format("Cursos = %s; Coste = %f", cursos, coste);
}
```

InRangeCursos

```
public InRangeCursos(String linea) {
    DatosCursos.iniDatos(linea);
}

@Override
public Integer size() {
    // TODO Auto-generated method stub
    return DatosCursos.getCursos();
}

@Override
public ChromosomeType type() {
    // TODO Auto-generated method stub
    return ChromosomeType.Binary;
}

@Override
public Double fitnessFunction(List<Integer> value) {
    // TODO Auto-generated method stub
    Double goal = 0.0;
    Double error1 = 0.0;
    Double error2 = 0.0;
    Set<Integer> temas = new HashSet<>();
    Set<Integer> centros = new HashSet<>();
    /*
     * size() hace referencia al tamaño del cromosoma
     * cuya función hemos hecho antes
     */
    for (int i = 0; i < size(); i++) {
        if(value.get(i)> 0) {
            goal += DatosCursos.getCurso(i).precio();
            List<Integer> temas_curso = DatosCursos.getCurso(i).temas();
            for (Integer tema : temas_curso) {
                temas.add(tema);
            }
            centros.add(DatosCursos.getCurso(i).centro());
        }
    }
    error1 += Math.abs(penalizacion_temas(temas));
    error2 += Math.abs(penalizacion_centros(centros));

    /*
     * Penaliza más que te pases de centros escogidos a que
     * haya algún tema sin dar
     */
    return -goal-100*error1-10000*error2;
}
```

```

public static Double penalizacion_temas(Set<Integer> temas) {
    Integer temas_totales = DatosCursos.getTemasTotal();
    Integer temas_actuales = temas.size();
    Integer pen = temas_totales - temas_actuales;
    return pen.doubleValue();
}

public static Double penalizacion_centros(Set<Integer> centros) {
    Integer max_centros = DatosCursos.getMaxCentros();
    Integer centros_actual = centros.size();
    Integer pen = centros_actual - max_centros;
    return pen.doubleValue();
}

/*
 * Si no creamos la solucion no se imprime NADA por consola
 */
@Override
public SolucionCursos solucion(List<Integer> value) {
    // TODO Auto-generated method stub
    return SolucionCursos.create(value);
}

@Override
public Integer max(Integer i) {
    // TODO Auto-generated method stub
    return 2;
}

@Override
public Integer min(Integer i) {
    // TODO Auto-generated method stub
    return 0;
}

```

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!

TestCursoRangeAG

```
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 = 0; i < 3; i++) {
        InRangeCursos p = new InRangeCursos("ficheros/Ejercicio2DatosEntrada" + (i+1) +".txt");

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

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



WUOLAH

Ejercicio 3

Modelo Isi

```
head section

Integer getInvestigadores()
Integer getEspecialidades()
Integer getTrabajos()
Integer getTrabajadorTieneEspecialidad(Integer i, Integer k)
Integer getDiasTrabajador(Integer i)
Integer getDiasTrabajadorEspecialista(Integer j, Integer k)
Integer getCalidadTrabajo(Integer j)
Integer getDiasTrabajoEnEspecialidad(Integer j, Integer k)

Integer n = getInvestigadores()
Integer m = getTrabajos()
Integer k = getEspecialidades()

goal section

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

constraints section

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

/*
tiempo_total(j) <= sum(x[i, j]) | especialidad_investigador(i) ==
especialidad_trabajo(j))
*/

sum(x[i,j], i in 0 .. n | getTrabajadorTieneEspecialidad(i, w) = 1) -
getDiasTrabajoEnEspecialidad(j, w) y[j] = 0, j in 0 .. m, w in 0 .. k

//y[j] == (getDiasTrabajoEnEspecialidad(j, k) <= sum(x[i,j], i in 0 .. n) |
getTrabajadorTieneEspecialidad(i, k), k in 0 .. k), j in 0 .. m

bounds section

x[i,j] <= getDiasTrabajador(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
```

DatosInvestigacion

```
/*
 * Datos:
 *
 * n: entero, número de investigadores
 *   • e: entero, número de especialidades
 *   • m: entero, número de trabajos
 *   • eik: binaria, trabajador i tiene especialidad tipo k, i en [0,n), k en
 *     [0,e)
 *   • ddi: entero, días disponibles del trabajador i, i en [0,n)
 *   • dnjk: entero, días necesarios para el trabajo j de investigador con
 *     especialidad k, j en [0,m), k en [0,e)
 *   • sj: entero, calidad trabajo j, j en [0,m)
 */
public static record Investigadores (String nombre, Integer capacidad, Integer especialidad, Integer id) {
    public static Investigadores create(String nombre, Integer capacidad, Integer especialidad, Integer id) {
        return new Investigadores(nombre, capacidad, especialidad, id);
    }
}

public static record Trabajos (String trabajo, Integer calidad, Map<Integer, Integer> relacion) {
    public static Trabajos create(String trabajo, Integer calidad, Map<Integer, Integer> relacion) {
        return new Trabajos(trabajo, calidad, relacion);
    }
}

/*
 * Hacemos a las variables públicas para que
 * puedan ser usadas por otras clases
 *
 * Como por ejemplo la de SolucionAG*/
public static List<Investigadores> investigadores;
public static List<Trabajos> trabajos;
public static List<Integer> especialidades;

// Tenemos que hacer prácticamente lo mismo en el Ejercicio1
```

```

public static void iniDatos(String fichero) {
    List<String> ls = Files2.LinesFromFile(fichero);
    List<Investigadores> li = new ArrayList<>();
    List<Trabajos> lt = new ArrayList<>();
    List<Integer> le = new ArrayList<>();
    Integer id = 0;
    for(String linea: ls) {

        if(linea.startsWith("// INVESTIGADORES")) {
            linea.replace("// INVESTIGADORES", "");
        }else if (linea.startsWith("// TRABAJOS")) {
            linea.replace("// TRABAJOS", "");
        }else if(linea.startsWith("INV")){
            /*Spliteamos los : de los Investigadores
            *
            * l[0]: INVX
            * l[1]: capacidad=x; especialidad=y;
            * */
            String[] l = linea.split(":");
            String investigador = l[0];
            String[] partes = l[1].split(";");
            /*
            System.out.println(partes[0]);
            System.out.println(partes[1]);
            */

            /*
            * Vamos spliteando poco a poco para tener la capacidad y especialidad
            * */
            String[] capacidad = partes[0].split("=");
            Integer c = Integer.parseInt(capacidad[1]);
            String[] especialidad = partes[1].split("=");
            Integer e = Integer.parseInt(especialidad[1]);
            le.add(e);
            Investigadores i = Investigadores.create(investigador, c, e, id++);
            li.add(i);
        }else if (linea.startsWith("T")) {
    }
}

```

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!

```
-----  
}else if (linea.startsWith("T")) {  
    // Igual que con los investigadores pero con esta relación  
    String[] l = linea.split(" -> ");  
    String trabajo = l[0];  
  
    String[] partes = l[1].split(";");  
    String[] calidad = partes[0].split("=");  
    Integer c = Integer.parseInt(calidad[1]);  
  
    String[] reparto = partes[1].split("=");  
    String r = reparto[1].substring(0, reparto[1].length());  
    String[] r_m = r.split(",");  
    Map<Integer, Integer> map = new HashMap<>();  
    for (String relacion : r_m) {  
        String[] rel = relacion.split(":");  
        Integer inv = Integer.parseInt(rel[0].substring(1));  
        /*  
         * Trampeando para quitar el paréntesis y para conseguir el  
         * valor numérico  
         */  
        Integer consumo = Integer.parseInt(rel[1].substring(0, rel[1].length()-1));  
        map.put(inv, consumo);  
    }  
    Trabajos t = Trabajos.create(trabajo, c, map);  
    lt.add(t);  
}  
}  
  
investigadores = li;  
trabajos = lt;  
especialidades = le;  
}  
  
// n: entero, número de investigadores  
  
public static Integer getInvestigadores() {  
    return investigadores.size();  
}  
  
// e: entero, número de especialidades  
  
public static Integer getEspecialidades() {  
    return especialidades.size();  
}  
  
// m: entero, número de trabajos  
  
public static Integer getTrabajos() {  
    return trabajos.size();  
}
```



WUOLAH

```

// eik: binaria, trabajador i tiene especialidad tipo k, i en [0,n), k en [0,e)
public static Integer getTrabajadorTieneEspecialidad(Integer i, Integer k) {
    Integer res = 0;
    Boolean la_tiene = false;
    if(investigadores.get(i).especialidad() == k) {
        la_tiene = true;
    }
    if(la_tiene) {
        res = 1;
    }
    return res;
}

// ddi: entero, días disponibles del trabajador i, i en [0,n)
public static Integer getDiasTrabajador(Integer i) {
    return investigadores.get(i).capacidad();
}

// dnjk: entero, días necesarios para el trabajo j de investigador con especialidad k, j en [0,m), k en [0,e)
public static Integer getDiasTrabajadorEspecialista(Integer j, Integer k) {
    return trabajos.get(j).relacion().get(k);
}

// cj: entero, calidad trabajo j, j en [0,m)
public static Integer getCalidadTrabajo(Integer j) {
    return trabajos.get(j).calidad();
}

public static Integer getDiasTrabajadorParaTrabajo(Integer i, Integer j) {
    Integer especialidad_investigador = investigadores.get(i).especialidad();
    Map<Integer, Integer> relacion_trabajo_especialidad = trabajos.get(j).relacion();
    Integer dias_especialista_trabajo = 0;
    for (Map.Entry<Integer, Integer> entry :
        relacion_trabajo_especialidad.entrySet()) {
        Integer dias_especialidad = entry.getValue();
        Integer especialidad = entry.getKey();
        if(especialidad == especialidad_investigador) {
            dias_especialista_trabajo = dias_especialidad;
        }
    }
    return dias_especialista_trabajo;
}

public static Integer getDiasTrabajoEnEspecialidad(Integer j, Integer k) {

    Integer dias_necesarios_especialidad = 0;
    Map<Integer, Integer> relacion_especialidad_dias = trabajos.get(j).relacion();
    for (Map.Entry<Integer, Integer> entry :
        relacion_especialidad_dias.entrySet()) {
        Integer dias_especialidad = entry.getValue();
        Integer especialidad = entry.getKey();
        if(especialidad == k) {
            dias_necesarios_especialidad = dias_especialidad;
        }
    }
    return dias_necesarios_especialidad;
}

```

Ejercicio3PLE

```
public static void ejercicio3_model_1() throws IOException {
    DatosInvestigacion.iniDatos("ficheros/Ejercicio3DatosEntrada1.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    AuxGrammar.generate(DatosInvestigacion.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 ejercicio3_model_2() throws IOException {
    DatosInvestigacion.iniDatos("ficheros/Ejercicio3DatosEntrada2.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    AuxGrammar.generate(DatosInvestigacion.class,"lsi_models/ejercicio3.lsi","gurobi_models/Ejercicio3-2.lp");
    GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio3-2.lp");
    Locale.setDefault(new Locale("en", "US"));
    System.out.println(solution.toString((s,d)->d>0.));
}

public static void ejercicio3_model_3() throws IOException {
    DatosInvestigacion.iniDatos("ficheros/Ejercicio3DatosEntrada3.txt");
    //si cambia el fichero de datos de entrada, cambiar tambien el nº del .lp para no sobreescribirlo
    AuxGrammar.generate(DatosInvestigacion.class,"lsi_models/ejercicio3.lsi","gurobi_models/Ejercicio3-3.lp");
    GurobiSolution solution = GurobiLp.gurobi("gurobi_models/Ejercicio3-3.lp");
    Locale.setDefault(new Locale("en", "US"));
    System.out.println(solution.toString((s,d)->d>0.));
}

public static void main(String[] args) throws IOException{
    // TODO Auto-generated method stub
    ejercicio3_model_1();
    ejercicio3_model_2();
    ejercicio3_model_3();
}
```

SolucionInvestigacionAG

```
public static SolucionInvestigacion create(List<Integer> value) {
    return new SolucionInvestigacion(value);
}

private Integer calidad;
private List<Investigadores> investigadores;
private List<List<Integer>> horas_invertidas;

private SolucionInvestigacion() {
    calidad = 0;
    investigadores = new ArrayList<>();
    horas_invertidas = new ArrayList<>();
}

private SolucionInvestigacion(List<Integer> ls) {
    Integer numInvestigadores = DatosInvestigacion.getInvestigadores();
    Integer numTrabajos = DatosInvestigacion.getTrabajos();
    Integer numEspecialidades = DatosInvestigacion.getEspecialidades();

    calidad = 0;
    investigadores = new ArrayList<>();
    investigadores.addAll(DatosInvestigacion.investigadores);
    horas_invertidas = new ArrayList<>();

    // Añadimos una lista por cada investigador
    for (int i = 0; i < numInvestigadores; i++) {
        horas_invertidas.add(new ArrayList<>());
    }

    for (int j = 0; j < numTrabajos; j++) {
        Integer iterador = j * numInvestigadores;
        List<Integer> trabajo = ls.subList(iterador, iterador+numInvestigadores);

        /*Añadimos a la lista de la listas
         * la sublist anterior con el trabajo realizado*/
        for (int i = 0; i < numInvestigadores; i++) {
            horas_invertidas.get(i).add(trabajo.get(i));
        }
    }

    /*
     *
     */
    // Empezamos con el constraint largo de PLE
    Boolean se_realiza = true;
```

Encuentra el trabajo de tus sueños



Escanéame y obtén más info!!



```
Boolean se_realiza = true;

for (int k = 0; k < numEspecialidades; k++) {
    Integer suma = 0;
    /* Vemos si el investigador i tiene la especialidad y se une
     * los días de trabajo del investigador i con su especialidad*/
    for (int i = 0; i < numInvestigadores; i++) {
        suma+= trabajo.get(i)*DatosInvestigacion.getTrabajadorTieneEspecialidad(i, k);
    }
    /* Si la suma de los días de los investigadores
     * es menor que los necesarios no se puede realizar*/
    if(suma < DatosInvestigacion.getDiasTrabajoEnEspecialidad(j, k)) {
        se_realiza = false;
        // Forzamos el salic del bucle al ser false
        k = numEspecialidades;
    }
}

// Como se realiza el trabajo se suma la calidad del mismo
if(se_realiza) {
    calidad+= DatosInvestigacion.getCalidadTrabajo(j);
}
```

}

```
public static SolucionInvestigacion empty() {
    return new SolucionInvestigacion();
}

public String toString() {
    String s = investigadores.stream().map(i->"INV"+ (i.id()+1) + ": " + horas_invertidas.get(i.id()))
        .collect(Collectors.joining("\n", "Reparto de horas:\n", "\n"));
    return String.format("%sSuma de las calidades de los trabajos: %d", s, calidad);
}
```



InvestigacionAG

```
public InvestigacionAG(String fichero) {
    DatosInvestigacion.iniDatos(fichero);
}

@Override
public Integer size() {
    // TODO Auto-generated method stub
    return DatosInvestigacion.getInvestigadores()*DatosInvestigacion.getTrabajos();
}

@Override
public ChromosomeType type() {
    // TODO Auto-generated method stub
    return ChromosomeType.Range;
}

@Override
public Double fitnessFunction(List<Integer> value) {
    // TODO Auto-generated method stub
    Double goal = 0.0;
    Double error = 0.0;
    Double multiplicando = 0.0;
    Integer capacidad = 0;
    // Auxiliares para no tener que llamarlos en los for
    Integer numInvestigadores = DatosInvestigacion.getInvestigadores();
    Integer numEspecialidades = DatosInvestigacion.getEspecialidades();
    Integer numTrabajos = DatosInvestigacion.getTrabajos();

    List<Integer> ls_capacidad = List2.ofTam(0, numInvestigadores);

    for (int j = 0; j < numTrabajos; j++) {
        // Dividimos la lista por trabajos
        Integer jj = j* numInvestigadores;
        List<Integer> trabajos = value.subList(jj, jj+numInvestigadores);
        Boolean se_realiza = true;
        for (int k = 0; k < numEspecialidades; k++) {
            Integer suma_dias = 0;
            for (int i = 0; i < numInvestigadores; i++) {
                suma_dias+= trabajos.get(i)*DatosInvestigacion.getTrabajadorTieneEspecialidad(i, k);
            }
            // Dias necesarios

            if(suma_dias < DatosInvestigacion.getDiasTrabajoEnEspecialidad(j, k)) {
                se_realiza = false;
                /*El error es la resta de los días obtenidos en el bucle menos los días que faltan
                 *
                 * Fallo por no realizarse*/
                error += Math.abs(suma_dias - DatosInvestigacion.getDiasTrabajoEnEspecialidad(j, k));
            }
        }
    }
}
```

```

/*Si se realiza se suma la calidad del trabajo al objetivo*/
if(se_realiza) {
    goal+= DatosInvestigacion.getCalidadTrabajo(j);
}

for (int i = 0; i < numInvestigadores; i++) {
    // Forzamos a 0 en cada iteración ya que luego se usa como acumulador
    capacidad = 0;
    for (int i2 = 0; i2 < value.size(); i2++) {
        capacidad = value.get(i2);
        ls_capacidad.set(i, ls_capacidad.get(i) + capacidad);
    }
}

for (int cap = 0; cap < ls_capacidad.size(); cap++) {
    // Restricción de cuando se superan los días disponibles del trabajador
    if(ls_capacidad.get(cap)> DatosInvestigacion.getDiasTrabajador(cap)) {
        error+= capacidad-DatosInvestigacion.getDiasTrabajador(cap);
    }
}
// Calculamos el multiplicando del error
Integer count = 0;
for (int j = 0; j < numTrabajos; j++) {
    count+= DatosInvestigacion.getCalidadTrabajo(j);
}

multiplicando = Math.pow(count, 2);

return goal - count*error;
}

@Override
public SolucionInvestigacion solucion(List<Integer> value) {
    // TODO Auto-generated method stub
    return SolucionInvestigacion.create(value);
}

@Override
public Integer max(Integer i) {
    // TODO Auto-generated method stub
    Integer l = i%DatosInvestigacion.getInvestigadores();
    return DatosInvestigacion.getDiasTrabajador(l)+1;
}

@Override
public Integer min(Integer i) {
    // TODO Auto-generated method stub
    return 0;
}

```

Ejercicio 4

DatosTrayectoria

```
public static record Cliente(Integer id, Double beneficio) {  
    public static Cliente ofFormat(String[] v) {  
        return new Cliente(Integer.parseInt(v[0].trim())  
                           , Double.parseDouble(v[1].trim()));  
    }  
}  
  
public static record Trayectoria(Integer origen, Integer destino,  
                                 Double kms) {  
    public static Trayectoria ofFormat(String[] v) {  
        return new Trayectoria(Integer.parseInt(v[0]),  
                               Integer.parseInt(v[1]),  
                               Double.parseDouble(v[2]));  
    }  
}  
  
public static Graph<Cliente, Trayectoria> grafo;  
  
public static void iniDatos(String fichero) {  
    grafo = GraphsReader.newGraph(fichero,  
                                  Cliente::ofFormat,  
                                  Trayectoria::ofFormat,  
                                  Graphs2::simpleWeightedGraph);  
  
    toConsole();  
}  
  
public static Integer getNumVertices() {  
    return grafo.vertexSet().size();  
}
```

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!

```
public static Cliente getCliente(Integer i) {
    Cliente c = null;
    List<Cliente> v = new ArrayList<>(grafo.vertexSet());
    for (int j = 0; j < v.size(); j++) {
        if(v.get(j).id() == i) {
            c = v.get(j);
        }
    }
    return c;
}

public static Double getBeneficio(Integer i) {
    return getCliente(i).beneficio();
}

public static Boolean existeCamino(Integer i, Integer j) {
    // Devolvemos si el cliente i y j contienen una arista
    return grafo.containsEdge(getCliente(i), getCliente(j));
}

public static Double getKm(Integer i, Integer j) {
    // Devolvemos el peso (kms) que hay entre los dos clientes
    return grafo.getEdge(getCliente(i), getCliente(j)).kms();
}

private static void toConsole() {
    // TODO Auto-generated method stub
    System.out.println("Vertices: " + grafo.vertexSet() + "\nAristas: " + grafo.edgeSet());
}
```



WUOLAH

SolucionTrayectoria

```
public static SolucionTrayectoria create(List<Integer> ls) {
    return new SolucionTrayectoria(ls);
}

private Double kms;
private Double beneficio;
private List<Cliente> clientes;
private SolucionTrayectoria() {
    kms = 0.0;
    beneficio = 0.0;
    clientes = new ArrayList<>();
    // El cliente 0 del enunciado
    Cliente inicio = DatosTrayectoria.getCliente(0);
    clientes.add(inicio);
}

private SolucionTrayectoria(List<Integer> ls) {
    // TODO Auto-generated constructor stub
    kms = 0.0;
    beneficio = 0.0;
    clientes = new ArrayList<>();
    // El cliente 0 del enunciado
    Cliente inicio = DatosTrayectoria.getCliente(0);
    clientes.add(inicio);
    for (int i = 0; i < ls.size(); i++) {
        // Aquí agregamos el cliente que corresponde al valor del cromosoma
        Cliente cliente_actual = DatosTrayectoria.getCliente(ls.get(i));
        clientes.add(cliente_actual);
        // Si es el inicio del grafo ...
        if(i == 0) {
            // ... y existe camino entre el 0 y este inicio
            if(DatosTrayectoria.existeCamino(0, ls.get(i))) {
                kms += DatosTrayectoria.getKm(0, ls.get(i));
                /* Hay que restarle a lo que ganamos los kms que hemos
                 * recorrido para llegar al destino*/
                beneficio += DatosTrayectoria.getBeneficio(ls.get(i)) - kms;
            }
        }else {
            // Y aquí ya si ya habiendo camino entre el valor que toca y el anterior
            if(DatosTrayectoria.existeCamino(ls.get(i-1), ls.get(i))) {
                kms += DatosTrayectoria.getKm(ls.get(i-1), ls.get(i));
                beneficio += DatosTrayectoria.getBeneficio(ls.get(i)) - kms;
            }
        }
    }
}

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

public String toString() {
    List<Integer> recorrido = clientes.stream().map(c-> c.id()).toList();
    return "El camino es:\n" + recorrido + "\nCon su distancia: " + kms + "\nY un beneficio de: " + beneficio ;
}
```

Ejercicio4AG

```
public Ejercicio4AG(String fichero) {
    DatosTrayectoria.iniDatos(fichero);
}

@Override
public ChromosomeType type() {
    // TODO Auto-generated method stub
    return ChromosomeType.Permutation;
}

@Override
public Double fitnessFunction(List<Integer> value) {
    // TODO Auto-generated method stub

    Double goal = 0.0;
    Double error = 0.0;
    Double multiplicando = 0.0;
    Double kms = 0.0;

    for (int i = 0; i < value.size(); i++) {
        // Primer valor de la lista
        if(i== 0) {
            if(DatosTrayectoria.existeCamino(0, value.get(i))) {
                kms += DatosTrayectoria.getKm(0, value.get(i));
                goal += DatosTrayectoria.getBeneficio(value.get(i))- kms;
            }else {
                // Si no existe la arista penalizamos
                error++;
            }
        // No es el primer valor de la lista
        }else {
            if(DatosTrayectoria.existeCamino(value.get(i-1), value.get(i))) {
                kms += DatosTrayectoria.getKm(value.get(i-1), value.get(i));
                goal += DatosTrayectoria.getBeneficio(value.get(i));
            }else {
                // Si no existe la arista penalizamos
                error++;
            }
        }
    }
}
```

```

// Se dobla la penalización si no volvemos al vértice 0

if(value.get(value.size() -1) != 0) {
    if(error == 0) {
        error += +2;
    }else {
        error = error * 2;
    }
}

/* Por cada km de más hay que sumarle la penalización del centavo*/

// Posible penalización --ToDo
multiplicando = Math.pow(kms, 2);
return goal - multiplicando * error;
}

@Override
public SolucionTrayectoria solucion(List<Integer> value) {
    // TODO Auto-generated method stub
    return SolucionTrayectoria.create(value);
}

// Es el size del permutation
@Override
public Integer itemsNumber() {
    // TODO Auto-generated method stub
    return DatosTrayectoria.getNumVertices();
}

```

Encuentra el trabajo de tus sueños

Participa en retos y competiciones de programación



Escanéame y
obtén más info!!

TestEjercicio4AG

```
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 = 0; i < 2; i++) {

        Ejercicio4AG p = new Ejercicio4AG("ficheros/Ejercicio4DatosEntrada" + (i+1) + ".txt");
        AlgoritmoAG<List<Integer>,SolucionTrayectoria> ap = AlgoritmoAG.of(p);
        ap.ejecuta();

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



WUOLAH