Práctica Individual 4

Juan Orellana Carretero – juaorecar

**Ejercicio 1**

**Ejercicio1.lsi**

head section

Integer getNumeroTipos()

Integer getNumeroVariedades()

Integer getCantidad(Integer j)

Integer getBeneficio(Integer i)

Double getCantidadTipoVariedad(Integer j, Integer i)

Integer n = getNumeroTipos()

Integer m = getNumeroVariedades()

goal section

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

constraints section

sum(getCantidadTipoVariedad(j,i) x[i], i in 0 .. m) <= getCantidad(j), j in 0 .. n // PARA CADA TIPO, NO SUPERAR CANTIDAD DISPONIBLE

int

x[i], i in 0 .. m

**DatosEjercicioCafes**

**package** datos;

**import** java.util.ArrayList;

**import** java.util.Comparator;

**import** java.util.List;

**import** us.lsi.common.Files2;

**public** **class** DatosEjercicioCafes {

**public** **static** List<Integer> *tipos*;

**public** **static** List<Variedad> *variedades*;

**public** **record** Variedad(**int** id, Integer beneficio, List<Double> mezcla) { // RECORD PARA LAS VARIEDADES DE CAFE

**public** **static** **int** *cont*;

**public** **static** Variedad create(String linea) {

List<Double> mezcla = **new** ArrayList<>();

**for** (**int** j = 0; j < *tipos*.size(); j++) {

mezcla.add(0.);

}

String[] var = linea.split(";");

Integer benef = Integer.*parseInt*(var[0].split("=")[1].replace(";", "").trim());

String[] comps = var[1].split("=")[1].trim().split(",");

**for** (**int** j = 0; j < comps.length; j++) {

String[] porcen = comps[j].replace("(C", "").replace(")", "").split(":");

Integer tipo = Integer.*parseInt*(porcen[0].trim()) - 1;

Double porcentaje = Double.*parseDouble*(porcen[1].trim());

mezcla.set(tipo, porcentaje);

}

**return** **new** Variedad(*cont*++, benef, **new** ArrayList<>(mezcla));

}

}

**public** **static** **void** iniDatos(String fich) { // LECTURA DE LOS DATOS

Variedad.*cont* = 0;

List<String> lineas = Files2.*linesFromFile*(fich);

**int** pos = lineas.indexOf("// VARIEDADES");

List<String> tiposCafe = lineas.subList(1, pos);

List<String> variedadesCafe = lineas.subList(pos + 1, lineas.size());

List<Integer> aux = **new** ArrayList<>();

**for** (**int** i = 0; i < tiposCafe.size(); i++) {

Integer valor = Integer.*parseInt*(tiposCafe.get(i).split("=")[1].replace(";", "").trim());

aux.add(valor);

}

*tipos* = **new** ArrayList<>(aux);

*variedades* = **new** ArrayList<>();

**for** (**int** i = 0; i < variedadesCafe.size(); i++) {

*variedades*.add(Variedad.*create*(variedadesCafe.get(i))); // HACEMOS USO DEL RECORD ANTERIOR

}

*toConsole*();

}

**public** **static** Integer getNumeroTipos() {

**return** *tipos*.size();

}

**public** **static** Integer getNumeroVariedades() {

**return** *variedades*.size();

}

**public** **static** Integer getCantidad(Integer j) {

**return** *tipos*.get(j);

}

**public** **static** Integer getBeneficio(Integer i) {

**return** *variedades*.get(i).beneficio();

}

**public** **static** Double getCantidadTipoVariedad(Integer j, Integer i) {

**return** *variedades*.get(i).mezcla().get(j);

}

**public** **static** List<Variedad> getVariedades() {

**return** **new** ArrayList<>(*variedades*);

}

**public** **static** Integer getCantidadMaxima(Integer i) {

List<Double> lsMax = **new** ArrayList<>();

**for** (**int** j = 0; j < *tipos*.size(); j++) {

lsMax.add(*getCantidad*(j) / *getCantidadTipoVariedad*(j, i));

}

lsMax.sort(Comparator.*naturalOrder*());

**return** lsMax.get(0).intValue();

}

**private** **static** **void** toConsole() {

System.***out***.println("Cantidad disponible tipo - " + *tipos* + "\nVariedad disponible - " + *variedades*);

}

**public** **static** **void** main(String[] args) {

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

System.***out***.println("\n######################## DATOS FICHERO " + i + " ########################\n");

String fich = "ficheros/Ejercicio1DatosEntrada" + i + ".txt";

*iniDatos*(fich);

System.***out***.println("\n");

}

}

}

**Ejercicio1PLE**

**package** ejercicio1;

**import** java.io.IOException;

**import** java.util.List;

**import** java.util.Locale;

**import** datos.DatosEjercicioCafes;

**import** datos.DatosEjercicioCafes.Variedad;

**import** us.lsi.gurobi.GurobiLp;

**import** us.lsi.gurobi.GurobiSolution;

**import** us.lsi.solve.AuxGrammar;

**public** **class** Ejercicio1PLE {

**public** **static** List<Integer> *tipos*;

**public** **static** List<Variedad> *variedades*;

**public** **static** Integer getNumeroTipos() {

**return** DatosEjercicioCafes.*getNumeroTipos*();

}

**public** **static** Integer getNumeroVariedades() {

**return** DatosEjercicioCafes.*getNumeroVariedades*();

}

**public** **static** Integer getCantidad(Integer j) {

**return** DatosEjercicioCafes.*getCantidad*(j);

}

**public** **static** Integer getBeneficio(Integer i) {

**return** DatosEjercicioCafes.*getBeneficio*(i);

}

**public** **static** Double getCantidadTipoVariedad(Integer j, Integer i) {

**return** DatosEjercicioCafes.*getCantidadTipoVariedad*(j, i);

}

**public** **static** **void** ejercicio1\_model() **throws** IOException {

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

System.***out***.println("\n\n#####################################################################################");

System.***out***.println("FICHERO EJERCICIO 1 CON DATOS DE ENTRADA " + i);

System.***out***.println("#####################################################################################\n");

DatosEjercicioCafes.*iniDatos*("ficheros/Ejercicio1DatosEntrada" + i + ".txt");

*tipos* = DatosEjercicioCafes.*tipos*;

*variedades* = DatosEjercicioCafes.*variedades*;

AuxGrammar.*generate*(Ejercicio1PLE.**class**, "lsi\_models/Ejercicio1.lsi", "gurobi\_models/Ejercicio1-" + i + ".lp");

GurobiSolution solution = GurobiLp.*gurobi*("gurobi\_models/Ejercicio1-" + i + ".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*();

}

}

**SolucionCafes**

**package** soluciones;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** datos.DatosEjercicioCafes;

**import** datos.DatosEjercicioCafes.Variedad;

**public** **class** SolucionCafe {

**public** **static** SolucionCafe of\_Range(List<Integer> value) {

**return** **new** SolucionCafe(value);

}

**private** Double beneficio;

**private** List<Variedad> lsVariedades;

**private** List<Integer> solucion;

**private** SolucionCafe() {

beneficio = 0.;

lsVariedades = **new** ArrayList<>();

solucion = **new** ArrayList<>();

}

**private** SolucionCafe(List<Integer> ls) {

beneficio = 0.;

lsVariedades = **new** ArrayList<>();

solucion = **new** ArrayList<>();

**for** (**int** i = 0; i < ls.size(); i++) {

**if** (ls.get(i) > 0) {

Integer kilos = ls.get(i);

Integer benef = DatosEjercicioCafes.*getBeneficio*(i) \* kilos;

lsVariedades.add(DatosEjercicioCafes.*getVariedades*().get(i));

solucion.add(ls.get(i));

beneficio += benef;

}

}

}

**public** **static** SolucionCafe empty() {

**return** **new** SolucionCafe();

}

// LE DAMOS FORMA AL LA SALIDA POR CONSOLA:

**public** String toString() {

String str = lsVariedades.stream().map(v -> "P" + (v.id() + 1) + ": " + solucion.get(lsVariedades.indexOf(v)))

.collect(Collectors.*joining*(" Kgs\n", "Variedades de cafe seleccionadas:\n", " Kg\n"));

**return** String.*format*("%sBeneficio: %.1f", str, beneficio);

}

}

**InRangeCafeAG**

**package** ejercicio1;

**import** java.util.List;

**import** datos.DatosEjercicioCafes;

**import** soluciones.SolucionCafe;

**import** us.lsi.ag.ValuesInRangeData;

**import** us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

**public** **class** InRangeCafeAG **implements** ValuesInRangeData<Integer, SolucionCafe> {

**public** InRangeCafeAG(String fichero) {

DatosEjercicioCafes.*iniDatos*(fichero);

}

@Override

**public** Integer max(Integer i) {

**return** DatosEjercicioCafes.*getCantidadMaxima*(i) + 1;

}

@Override

**public** Integer min(Integer i) {

**return** 0;

}

@Override

**public** Integer size() {

**return** DatosEjercicioCafes.*getNumeroVariedades*();

}

@Override

**public** ChromosomeType type() {

**return** ChromosomeType.***Range***;

}

@Override

**public** Double fitnessFunction(List<Integer> ls) {

**double** goal = 0, error = 0, dif = 0, k = 0;

**for** (**int** i = 0; i < size(); i++) {

**if** (ls.get(i) > 0) {

goal += ls.get(i) \* DatosEjercicioCafes.*getBeneficio*(i); // GOAL DEL PROBLEMA, BENEF DE VARIEDAD \* LA VARIEDAD

}

}

**for** (**int** j = 0; j < DatosEjercicioCafes.*getNumeroTipos*(); j++) {

dif = 0;

**for** (**int** i = 0; i < size(); i++) { // RESTRICCION: DEBEMOS TENER CANTIDAD DISPONIBLE DE CADA TIPO

dif += ls.get(i) \* DatosEjercicioCafes.*getCantidadTipoVariedad*(j, i);

}

**if** (dif > DatosEjercicioCafes.*getCantidad*(j)) {

error += dif - DatosEjercicioCafes.*getCantidad*(j);

}

}

**for** (**int** i = 0; i < size(); i++) { // CALCULO DE K (A CONTINUACION LO MULTIPLICAMOS POR EL ERROR PARA QUE ESTE SEA GRANDE)

k += Math.*pow*(DatosEjercicioCafes.*getCantidadMaxima*(i) \* DatosEjercicioCafes.*getBeneficio*(i), 2);

}

**return** goal - k \* error;

}

@Override

**public** SolucionCafe solucion(List<Integer> ls\_chrm) {

**return** SolucionCafe.*of\_Range*(ls\_chrm);

}

}

**TestCafeAGRange**

**package** ejercicio1;

**import** java.util.List;

**import** java.util.Locale;

**import** soluciones.SolucionCafe;

**import** us.lsi.ag.agchromosomes.AlgoritmoAG;

**import** us.lsi.ag.agstopping.StoppingConditionFactory;

**public** **class** TestCafeAGRange {

**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++) {

System.***out***.println("\n\n#####################################################################################");

System.***out***.println("SOLUCION EJERCICIO 1 CON DATOS DE ENTRADA " + i);

System.***out***.println("#####################################################################################\n");

InRangeCafeAG p = **new** InRangeCafeAG("ficheros/Ejercicio1DatosEntrada" + i + ".txt");

AlgoritmoAG<List<Integer>, SolucionCafe> ap = AlgoritmoAG.*of*(p);

ap.ejecuta();

System.***out***.println("================================");

System.***out***.println(ap.bestSolution());

System.***out***.println("================================");

}

}

}

**EJERCICIO 2**

**Ejericicio2.lsi**

head section

Integer getNumeroCursos()

Integer getNumeroCentros()

Integer getNumeroTematicas()

Integer getMaxCentros()

Integer getPrecioCurso(Integer i)

Integer contieneTematica(Integer i, Integer j)

Integer ofreceCurso(Integer i, Integer k)

Integer n = getNumeroCursos()

Integer m = getNumeroTematicas()

Integer c = getNumeroCentros()

Integer mCentros = getMaxCentros()

goal section

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

constraints section

sum(contieneTematica(i,j) x[i], i in 0 .. n) >= 1 , j in 0 .. m // LAS TEMATICAS TIENEN QUE SER SELECCIONADAS

sum(y[k], k in 0 .. c) <= mCentros // RESPETAR EL MAXIMO DE CURSOS SELECCIONABLES

ofreceCurso(i,k) x[i] - y[k] <= 0, i in 0 .. n, k in 0 .. c // PARA CADA CENTRO DE CADA CURSO, SI SE SELECCIONA UN CURSO DEL CENTRO, TAMBIEN EL CENTRO

bin

x[i], i in 0 .. n

y[k], k in 0 .. c

**DatosEjercicioCursos**

**package** datos;

**import** java.util.ArrayList;

**import** java.util.HashSet;

**import** java.util.List;

**import** java.util.Set;

**import** us.lsi.common.Files2;

**public** **class** DatosEjercicioCursos {

**public** **static** List<Curso> *cursos*;

**public** **static** Integer *maxCentros*;

**public** **record** Curso(Integer id, List<Integer> tematicas, Double precio, Integer centro) {

**public** **static** **int** *cont*;

**public** **static** Curso create(String linea) {

List<Integer> aux = **new** ArrayList<>();

String[] params = linea.split(":");

String[] temas = params[0].substring(1, params[0].length() - 1).split(",");

**for** (String str : temas) {

aux.add(Integer.*parseInt*(str.trim()));

}

**return** **new** Curso(*cont*++, **new** ArrayList<>(aux), Double.*parseDouble*(params[1].trim()),

Integer.*parseInt*(params[2].trim()));

}

}

**public** **static** **void** iniDatos(String fich) {

List<Curso> res = **new** ArrayList<>();

Curso.*cont* = 0;

List<String> lineas = Files2.*linesFromFile*(fich);

*maxCentros* = Integer.*parseInt*(lineas.get(0).split("=")[1].trim());

**for** (String st : lineas.subList(1, lineas.size())) {

res.add(Curso.*create*(st));

}

*cursos* = **new** ArrayList<>(res);

*toConsole*();

}

**public** **static** Integer getMaxCentros() {

**return** *maxCentros*;

}

**public** **static** Integer getNumeroCursos() {

**return** *cursos*.size();

}

**public** **static** List<Integer> getTematicas() {

Set<Integer> s = **new** HashSet<>();

**for** (Curso t : *cursos*) {

s.addAll(t.tematicas());

}

**return** **new** ArrayList<>(s);

}

**public** **static** Integer getNumeroTematicas() {

**return** *getTematicas*().size();

}

**public** **static** List<Integer> getTematicasCursos(Integer i) {

**return** *cursos*.get(i).tematicas();

}

**public** **static** Integer getNumeroTematicasCursos(Integer i) {

**return** *getTematicasCursos*(i).size();

}

**public** **static** Integer contieneTematica(Integer i, Integer j) {

**return** *cursos*.get(i).tematicas().contains(*getTematicas*().get(j)) ? 1 : 0;

}

**public** **static** Double getPrecioCurso(Integer i) {

**return** *cursos*.get(i).precio();

}

**public** **static** Integer getCentroCurso(Integer i) {

**return** *cursos*.get(i).centro();

}

**public** **static** List<Integer> getCentros() {

Set<Integer> s = **new** HashSet<>();

**for** (Curso cu : *cursos*) {

s.add(cu.centro());

}

**return** **new** ArrayList<>(s);

}

**public** **static** Integer getNumeroCentros() {

**return** *getCentros*().size();

}

**public** **static** Integer ofreceCurso(Integer i, Integer k) {

**return** *cursos*.get(i).centro().equals(*getCentros*().get(k)) ? 1 : 0;

}

**public** **static** **void** toConsole() {

System.***out***.println("Maximo de centros selecionables: " + *maxCentros* + "\nCursos disponibles: " + *cursos*);

}

**public** **static** **void** main(String[] args) {

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

System.***out***.println("\n######################## DATOS FICHERO " + i + " ########################\n");

String fich = "ficheros/Ejercicio2DatosEntrada" + i + ".txt";

*iniDatos*(fich);

System.***out***.println("\n");

}

}

}

**Ejercicio2PLE**

**package** ejercicio2;

**import** java.io.IOException;

**import** java.util.List;

**import** java.util.Locale;

**import** datos.DatosEjercicioCursos;

**import** us.lsi.gurobi.GurobiLp;

**import** us.lsi.gurobi.GurobiSolution;

**import** us.lsi.solve.AuxGrammar;

**public** **class** Ejercicio2PLE {

**public** **static** Integer getMaxCentros() {

**return** DatosEjercicioCursos.*getMaxCentros*();

}

**public** **static** Integer getNumeroCursos() {

**return** DatosEjercicioCursos.*getNumeroCursos*();

}

**public** **static** List<Integer> getTematicas() {

**return** DatosEjercicioCursos.*getTematicas*();

}

**public** **static** Integer getNumeroTematicas() {

**return** DatosEjercicioCursos.*getNumeroTematicas*();

}

**public** **static** List<Integer> getTematicasCursos(Integer i) {

**return** DatosEjercicioCursos.*getTematicasCursos*(i);

}

**public** **static** Integer getNumeroTematicasCursos(Integer i) {

**return** DatosEjercicioCursos.*getNumeroTematicasCursos*(i);

}

**public** **static** Integer contieneTematica(Integer i, Integer j) {

**return** DatosEjercicioCursos.*contieneTematica*(i, j);

}

**public** **static** Double getPrecioCurso(Integer i) {

**return** DatosEjercicioCursos.*getPrecioCurso*(i);

}

**public** **static** Integer getCentroCurso(Integer i) {

**return** DatosEjercicioCursos.*getCentroCurso*(i);

}

**public** **static** List<Integer> getCentros() {

**return** DatosEjercicioCursos.*getCentros*();

}

**public** **static** Integer getNumeroCentros() {

**return** DatosEjercicioCursos.*getNumeroCentros*();

}

**public** **static** Integer ofreceCurso(Integer i, Integer k) {

**return** DatosEjercicioCursos.*ofreceCurso*(i, k);

}

**public** **static** **void** ejercicio2\_model() **throws** IOException {

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

System.***out***.println(

"\n\n#####################################################################################");

System.***out***.println("FICHERO EJERCICIO 2 CON DATOS DE ENTRADA " + i);

System.***out***

.println("#####################################################################################\n");

DatosEjercicioCursos.*iniDatos*("ficheros/Ejercicio2DatosEntrada" + i + ".txt");

AuxGrammar.*generate*(Ejercicio2PLE.**class**, "lsi\_models/Ejercicio2.lsi",

"gurobi\_models/Ejercicio2-" + i + ".lp");

GurobiSolution solution = GurobiLp.*gurobi*("gurobi\_models/Ejercicio2-" + i + ".lp");

Locale.*setDefault*(**new** Locale("en", "US"));

System.***out***.println(solution.toString((s, d) -> d > 0.));

}

}

**public** **static** **void** main(String[] args) **throws** IOException {

*ejercicio2\_model*();

}

}

**SolucionCursos**

**package** soluciones;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** datos.DatosEjercicioCursos;

**import** datos.DatosEjercicioCursos.Curso;

**public** **class** SolucionCursos {

**public** **static** SolucionCursos of\_Range(List<Integer> value) {

**return** **new** SolucionCursos(value);

}

**private** Double precio;

**private** List<Curso> cursos;

**public** SolucionCursos() {

precio = 0.;

cursos = **new** ArrayList<>();

}

**public** SolucionCursos(List<Integer> ls) {

precio = 0.;

cursos = **new** ArrayList<>();

**for** (**int** i = 0; i < ls.size(); i++) {

**if** (ls.get(i) > 0) {

precio += DatosEjercicioCursos.*getPrecioCurso*(i);

cursos.add(DatosEjercicioCursos.*cursos*.get(i));

}

}

}

**public** **static** SolucionCursos empty() {

**return** **new** SolucionCursos();

}

**public** String toString() {

String str = cursos.stream().map(c -> "S" + c.id())

.collect(Collectors.*joining*(", ", "Cursos elegidos: {", "}\n"));

**return** String.*format*("%sCoste total: %.1f", str, precio);

}

}

**BinCursosAG**

package ejercicio2;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import datos.DatosEjercicioCursos;

import soluciones.SolucionCursos;

import us.lsi.ag.BinaryData;

public class BinCursosAG implements BinaryData<SolucionCursos> {

public BinCursosAG(String fich) {

DatosEjercicioCursos.iniDatos(fich);

}

@Override

public Integer size() {

return DatosEjercicioCursos.getNumeroCursos();

}

@Override

public Double fitnessFunction(List<Integer> ls\_chrm) {

double goal = 0, error = 0, k = 0, acum = 0;

for (int i = 0; i < ls\_chrm.size(); i++) {

if (ls\_chrm.get(i) > 0) {

goal += DatosEjercicioCursos.getPrecioCurso(i);

}

}

Set<Integer> setTem = new HashSet<>();

Set<Integer> setCent = new HashSet<>();

for (int i = 0; i < ls\_chrm.size(); i++) {

if (ls\_chrm.get(i) > 0) {

setTem.addAll(DatosEjercicioCursos.getTematicasCursos(i));

setCent.add(DatosEjercicioCursos.getCentroCurso(i));

}

}

Integer m = DatosEjercicioCursos.getNumeroTematicas();

Integer c = DatosEjercicioCursos.maxCentros;

if (setTem.size() < m) { // RESTRICCION: SELECCION TEMATICAS

error += m - setTem.size();

}

if (setCent.size() > c) { // RESTRICCION: SELECCION CENTROS

error += setCent.size() - c;

}

for (int i = 0; i < ls\_chrm.size(); i++) { // FORMA DE CALCULAR K

acum += DatosEjercicioCursos.getPrecioCurso(i);

}

k += Math.pow(acum, 2);

return -goal -k \* error;

}

@Override

public SolucionCursos solucion(List<Integer> ls\_chrm) {

return SolucionCursos.of\_Range(ls\_chrm);

}

}

**TestCursosAGBin**

**package** ejercicio2;

**import** java.util.List;

**import** java.util.Locale;

**import** soluciones.SolucionCursos;

**import** us.lsi.ag.agchromosomes.AlgoritmoAG;

**import** us.lsi.ag.agstopping.StoppingConditionFactory;

**public** **class** TestCursosAGBin {

**public** **static** **void** main(String[] args) {

Locale.*setDefault*(**new** Locale("en", "US"));

AlgoritmoAG.*ELITISM\_RATE* = 0.30;

AlgoritmoAG.*CROSSOVER\_RATE* = 0.8;

AlgoritmoAG.*MUTATION\_RATE* = 0.7;

AlgoritmoAG.*POPULATION\_SIZE* = 50;

StoppingConditionFactory.*NUM\_GENERATIONS* = 5000;

StoppingConditionFactory.*stoppingConditionType* = StoppingConditionFactory.StoppingConditionType.***GenerationCount***;

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

System.***out***.println("\n\n#####################################################################################");

System.***out***.println("SOLUCION EJERCICIO 2 CON DATOS DE ENTRADA " + i);

System.***out***.println("#####################################################################################\n");

BinCursosAG p = **new** BinCursosAG("ficheros/Ejercicio2DatosEntrada" + i + ".txt");

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

ap.ejecuta();

System.***out***.println("================================");

System.***out***.println(ap.bestSolution());

System.***out***.println("================================");

}

}

}

**EJERCICIO 3**

**Modelado para AG**

Cada cromosoma se compone de n\*m genes con valores dentro del rango permitido, siendo cada gen los días que un investigador de una determinada dedica a un trabajo específico

La función fitness será la suma total de las calidades de los trabajos en la solución representada por el cromosoma. A partir de la población inicial se irán comprobando las diferentes selecciones evaluando el fitness de cada selección, quedándonos con la mejor opción

**Ejercicio3.lsi**

head section

Integer getNumeroInvestigadores()

Integer getNumeroEspecialidades()

Integer getNumeroTrabajos()

Integer trabajadorEspecialidad(Integer i, Integer k)

Integer diasDisponibles(Integer i)

Integer diasNecesarios(Integer j, Integer k)

Integer getCalidad(Integer j)

Integer getMM()

Integer n = getNumeroInvestigadores()

Integer e = getNumeroEspecialidades()

Integer m = getNumeroTrabajos()

Integer MM = getMM()

goal section

max sum(getCalidad(j) y[j], j in 0 .. m) // MAX CALIDAD TRABAJOS

constraints section

sum(x[i,j], j in 0 .. m) <= diasDisponibles(i), i in 0 .. n // PARA CADA INVEST, SUM HORAS REALIZADAS EN CADA TRABJ <= HORAS DISPONIBLES DEL INVEST

sum(trabajadorEspecialidad(i,k) x[i,j], i in 0 .. n) - diasNecesarios(j,k) y[j] = 0, j in 0 .. m, k in 0 .. e // PARA CADA ESPECIALIDAD EN CADA TRABAJO, SUM HORAS REALIZADAS CADA INVEST CON ESPECIALIDAD INDICADA = HORAS NECESARIAS

x[i,j] - MM y[j] <= 0, j in 0 .. m, i in 0 .. n // PARA CADA INVEST EN CADA TRABAJO, SI TRABAJO J NO SE REALIZA, LAS HORAS REALIZADAS EN J = 0

bounds section

y[j] <= 1, j in 0 .. m // PARA TOMAR y[j] COMO BINARIA

int

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

y[j], j in 0 .. m

**DatosEjercicioInvestigadores**

**package** datos;

**import** java.util.ArrayList;

**import** java.util.Comparator;

**import** java.util.List;

**import** us.lsi.common.Files2;

**public** **class** DatosEjercicioInvestigadores {

**public** **static** List<Investigador> *investigadores*;

**public** **static** List<Trabajo> *trabajos*;

**public** **record** Investigador(Integer id, Integer capacidad, Integer especialidad) {

**public** **static** **int** *cont*;

**public** **static** Investigador create(String linea) {

String[] investig = linea.split(";");

Integer capacidad = Integer.*parseInt*(investig[0].trim().split("=")[1].trim());

Integer especialidad = Integer.*parseInt*(investig[1].trim().split("=")[1].trim());

**return** **new** Investigador(*cont*++, capacidad, especialidad);

}

}

**public** **record** Trabajo(Integer id, Integer calidad, List<Integer> dias) {

**public** **static** **int** *cont*;

**public** **static** Trabajo create(String linea) {

String[] trabaj = linea.split(";");

Integer calid = Integer.*parseInt*(trabaj[0].trim().split("=")[1].trim());

String[] repart = trabaj[1].trim().split("=")[1].trim().split(",");

List<Integer> dias = **new** ArrayList<>();

**for** (String str : repart) {

str = str.replace("(", "").replace(")", "").trim();

dias.add(Integer.*parseInt*(str.split(":")[1].trim()));

}

**return** **new** Trabajo(*cont*++, calid, dias);

}

}

**public** **static** **void** iniDatos(String fich) {

Investigador.*cont* = 0;

Trabajo.*cont* = 0;

*investigadores* = **new** ArrayList<>();

*trabajos* = **new** ArrayList<>();

List<String> lineas = Files2.*linesFromFile*(fich);

List<String> investis = lineas.subList(1, lineas.indexOf("// TRABAJOS"));

List<String> trabajs = lineas.subList(lineas.indexOf("// TRABAJOS") + 1, lineas.size());

**for**(String i : investis) {

*investigadores*.add(Investigador.*create*(i));

}

**for** (String t : trabajs) {

*trabajos*.add(Trabajo.*create*(t));

}

*toConsole*();

}

**public** **static** Integer getNumeroInvestigadores() {

**return** *investigadores*.size();

}

**public** **static** Integer getNumeroEspecialidades() {

**return** *trabajos*.get(0).dias().size();

}

**public** **static** Integer getNumeroTrabajos() {

**return** *trabajos*.size();

}

**public** **static** Integer trabajadorEspecialidad(Integer i, Integer k) {

**return** *investigadores*.get(i).especialidad().equals(k) ? 1 : 0;

}

**public** **static** Integer diasDisponibles(Integer i) {

**return** *investigadores*.get(i).capacidad();

}

**public** **static** Integer diasNecesarios(Integer j, Integer k) {

**return** *trabajos*.get(j).dias().get(k);

}

**public** **static** Integer getCalidad(Integer j) {

**return** *trabajos*.get(j).calidad();

}

**public** **static** Integer getMM() {

**return** *investigadores*.stream().map(i -> i.capacidad()).max(Comparator.*naturalOrder*()).get() + 1;

}

**private** **static** **void** toConsole() {

System.***out***.println(*investigadores*);

System.***out***.println(*trabajos*);

}

**public** **static** **void** main(String[] args) {

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

System.***out***.println("\n######################## DATOS FICHERO " + i + " ########################\n");

String fich = "ficheros/Ejercicio3DatosEntrada" + i + ".txt";

*iniDatos*(fich);

System.***out***.println("\n");

}

}

}

**Ejercicio3PLE**

**package** ejercicio3;

**import** java.io.IOException;

**import** java.util.Locale;

**import** datos.DatosEjercicioInvestigadores;

**import** us.lsi.gurobi.GurobiLp;

**import** us.lsi.gurobi.GurobiSolution;

**import** us.lsi.solve.AuxGrammar;

**public** **class** Ejercicio3PLE {

**public** **static** Integer getNumeroInvestigadores() {

**return** DatosEjercicioInvestigadores.*getNumeroInvestigadores*();

}

**public** **static** Integer getNumeroEspecialidades() {

**return** DatosEjercicioInvestigadores.*getNumeroEspecialidades*();

}

**public** **static** Integer getNumeroTrabajos() {

**return** DatosEjercicioInvestigadores.*getNumeroTrabajos*();

}

**public** **static** Integer trabajadorEspecialidad(Integer i, Integer k) {

**return** DatosEjercicioInvestigadores.*trabajadorEspecialidad*(i, k);

}

**public** **static** Integer diasDisponibles(Integer i) {

**return** DatosEjercicioInvestigadores.*diasDisponibles*(i);

}

**public** **static** Integer diasNecesarios(Integer j, Integer k) {

**return** DatosEjercicioInvestigadores.*diasNecesarios*(j, k);

}

**public** **static** Integer getCalidad(Integer j) {

**return** DatosEjercicioInvestigadores.*getCalidad*(j);

}

**public** **static** Integer getMM() {

**return** DatosEjercicioInvestigadores.*getMM*();

}

**public** **static** **void** ejercicio3\_model() **throws** IOException {

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

System.***out***.println("\n\n#####################################################################################");

System.***out***.println("FICHERO EJERCICIO 3 CON DATOS DE ENTRADA " + i);

System.***out***.println("#####################################################################################\n");

DatosEjercicioInvestigadores.*iniDatos*("ficheros/Ejercicio3DatosEntrada" + i + ".txt");

AuxGrammar.*generate*(Ejercicio3PLE.**class**, "lsi\_models/Ejercicio3.lsi", "gurobi\_models/Ejercicio3-" + i + ".lp");

GurobiSolution solution = GurobiLp.*gurobi*("gurobi\_models/Ejercicio3-" + i + ".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*();

}

}

**SolucionInvestigadores**

**package** soluciones;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** datos.DatosEjercicioInvestigadores;

**import** datos.DatosEjercicioInvestigadores.Investigador;

**public** **class** SolucionInvestigadores {

**public** **static** SolucionInvestigadores of\_Range(List<Integer> value) {

**return** **new** SolucionInvestigadores(value);

}

**private** Integer calidad;

**private** List<Investigador> investigadores;

**private** List<List<Integer>> horas;

**private** SolucionInvestigadores() {

calidad = 0;

investigadores = **new** ArrayList<>();

horas = **new** ArrayList<>();

}

**private** SolucionInvestigadores(List<Integer> ls) {

Integer numeroInvestigadores = DatosEjercicioInvestigadores.*getNumeroInvestigadores*();

Integer numeroTrabajos = DatosEjercicioInvestigadores.*getNumeroTrabajos*();

Integer numeroEspecialidades = DatosEjercicioInvestigadores.*getNumeroEspecialidades*();

calidad = 0;

investigadores = **new** ArrayList<>();

investigadores.addAll(DatosEjercicioInvestigadores.*investigadores*);

horas = **new** ArrayList<>();

**for** (**int** i = 0; i < numeroInvestigadores; i++) {

horas.add(**new** ArrayList<>());

}

**for** (**int** j = 0; j < numeroTrabajos; j++) {

Integer jj = j \* numeroInvestigadores;

List<Integer> trab = ls.subList(jj, jj + numeroInvestigadores);

**for** (**int** i = 0; i < numeroInvestigadores; i++) {

horas.get(i).add(trab.get(i));

}

Boolean realiza = **true**;

**for** (**int** k = 0; k < numeroEspecialidades; k++) {

Integer suma = 0;

**for** (**int** i = 0; i < numeroInvestigadores; i++) {

suma += trab.get(i) \* DatosEjercicioInvestigadores.*trabajadorEspecialidad*(i, k);

}

**if** (suma < DatosEjercicioInvestigadores.*diasNecesarios*(j, k)) {

realiza = **false**;

k = numeroEspecialidades;

}

}

**if** (realiza) {

calidad += DatosEjercicioInvestigadores.*getCalidad*(j);

}

}

}

**public** **static** SolucionInvestigadores empty() {

**return** **new** SolucionInvestigadores();

}

**public** String toString() {

String str = investigadores.stream().map(i -> "INV" + (i.id() + 1) + ": " + horas.get(i.id()))

.collect(Collectors.*joining*("\n",

"Reparto obtenido (dias trabajados por cada investigador en cada trabajo):\n", "\n"));

**return** String.*format*("%sSUMA DE LAS CALIDADES DE LOS TRABAJOS REALIZADOS: %d", str, calidad);

}

}

**InRangeInvestigadoresAG**

**package** ejercicio3;

**import** java.util.List;

**import** datos.DatosEjercicioInvestigadores;

**import** soluciones.SolucionInvestigadores;

**import** us.lsi.ag.ValuesInRangeData;

**import** us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

**public** **class** InRangeInvestigadoresAG **implements** ValuesInRangeData<Integer, SolucionInvestigadores> {

**public** InRangeInvestigadoresAG(String fich) {

DatosEjercicioInvestigadores.*iniDatos*(fich);

}

@Override

**public** Integer max(Integer i) {

Integer num = i % DatosEjercicioInvestigadores.*getNumeroInvestigadores*(); // Imod N

**return** DatosEjercicioInvestigadores.*diasDisponibles*(num) + 1;

}

@Override

**public** Integer min(Integer i) {

**return** 0;

}

@Override

**public** Integer size() {

**return** DatosEjercicioInvestigadores.*getNumeroInvestigadores*() \* DatosEjercicioInvestigadores.*getNumeroTrabajos*();

}

@Override

**public** ChromosomeType type() {

**return** ChromosomeType.***Range***;

}

@Override

**public** Double fitnessFunction(List<Integer> ls\_chrm) {

**double** goal = 0, error = 0, k\_k = 0, capacidad = 0;

Integer nInvest = DatosEjercicioInvestigadores.*getNumeroInvestigadores*();

Integer nTrabaj = DatosEjercicioInvestigadores.*getNumeroTrabajos*();

Integer nEspec = DatosEjercicioInvestigadores.*getNumeroEspecialidades*();

**for** (**int** j = 0; j < nTrabaj; j++) {

Integer jnInvest = j \* nInvest;

List<Integer> trabajs = ls\_chrm.subList(jnInvest, jnInvest + nInvest);

Boolean realiza = **true**;

**for** (**int** k = 0; k < nEspec; k++) {

Integer suma = 0;

**for** (**int** i = 0; i < nInvest; i++) {

suma += trabajs.get(i) \* DatosEjercicioInvestigadores.*trabajadorEspecialidad*(i, k);

}

**if** (suma != DatosEjercicioInvestigadores.*diasNecesarios*(j, k)) {

realiza = **false**;

error += Math.*abs*(suma - DatosEjercicioInvestigadores.*diasNecesarios*(j, k));

}

}

**if** (realiza) {

goal += DatosEjercicioInvestigadores.*getCalidad*(j);

}

}

**for** (**int** i = 0; i < nInvest; i++) {

capacidad = 0;

**for** (**int** i\_i = i; i\_i < ls\_chrm.size(); i\_i += nInvest) {

capacidad += ls\_chrm.get(i\_i);

}

**if** (capacidad > DatosEjercicioInvestigadores.*diasDisponibles*(i)) {

error += capacidad - DatosEjercicioInvestigadores.*diasDisponibles*(i);

}

}

Integer suma = 0;

**for** (**int** j = 0; j < nTrabaj; j++) {

suma += DatosEjercicioInvestigadores.*getCalidad*(j);

}

k\_k = Math.*pow*(suma, 2);

**return** goal - k\_k \* error;

}

@Override

**public** SolucionInvestigadores solucion(List<Integer> ls\_chrm) {

System.***out***.println(ls\_chrm);

**return** SolucionInvestigadores.*of\_Range*(ls\_chrm);

}

}

**TestInvestigadoresAGRange**

**package** ejercicio3;

**import** java.util.List;

**import** java.util.Locale;

**import** soluciones.SolucionInvestigadores;

**import** us.lsi.ag.agchromosomes.AlgoritmoAG;

**import** us.lsi.ag.agstopping.StoppingConditionFactory;

**public** **class** TestInvestigadoresAGRange {

**public** **static** **void** main(String[] args) {

Locale.*setDefault*(**new** Locale("en", "US"));

AlgoritmoAG.*ELITISM\_RATE* = 0.10;

AlgoritmoAG.*CROSSOVER\_RATE* = 0.95;

AlgoritmoAG.*MUTATION\_RATE* = 0.8;

AlgoritmoAG.*POPULATION\_SIZE* = 1000;

StoppingConditionFactory.*NUM\_GENERATIONS* = 1000;

StoppingConditionFactory.*stoppingConditionType* = StoppingConditionFactory.StoppingConditionType.***GenerationCount***;

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

System.***out***.println("\n\n#####################################################################################");

System.***out***.println("SOLUCION EJERCICIO 3 CON DATOS DE ENTRADA " + i);

System.***out***.println("#####################################################################################\n");

InRangeInvestigadoresAG p = **new** InRangeInvestigadoresAG("ficheros/Ejercicio3DatosEntrada" + i + ".txt");

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

ap.ejecuta();

System.***out***.println("================================");

System.***out***.println(ap.bestSolution());

System.***out***.println("================================");

}

}

}

**EJERCICIO 4**

El objetivo es encontrar la permutación que maximice el beneficio total, menos el costo total de recorrer las distancias y la penalización por el retraso en cada entrega

**Cliente**

**package** utiles;

**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[] forma) {

Integer id = Integer.*valueOf*(forma[0].trim());

Double beneficio = Double.*valueOf*(forma[1].trim());

**return** *of*(id, beneficio);

}

@Override

**public** String toString() {

**return** "Cliente [id=" + id + ", beneficio=" + beneficio + "]";

}

}

**Trayecto**

**package** utiles;

**public** **record** Trayecto(**int** id, Double distancia) {

**public** **static** **int** *cont*;

**public** **static** Trayecto of(Double distancia) {

Integer id = *cont*;

*cont*++;

**return** **new** Trayecto(id, distancia);

}

**public** **static** Trayecto ofFormat(String[] forma) {

Double distancia = Double.*valueOf*(forma[2].trim());

**return** *of*(distancia);

}

@Override

**public** String toString() {

**return** "Conexion [id=" + id + ", dist=" + distancia + "]";

}

}

**DatosEjercicioClientes**

**package** datos;

**import** java.util.ArrayList;

**import** java.util.List;

**import** org.jgrapht.Graph;

**import** us.lsi.graphs.Graphs2;

**import** us.lsi.graphs.GraphsReader;

**import** utiles.Cliente;

**import** utiles.Trayecto;

**public** **class** DatosEjercicioClientes {

@SuppressWarnings("exports") // PARA SUPRIMIR EL AVISO Y PERMITIR EXPORTACION SIN TENER QUE PONERLO EN EL MODULE INFO

**public** **static** Graph<Cliente, Trayecto> *gf*;

**public** **static** **void** iniDatos(String fichero) {

*gf* = GraphsReader.*newGraph*(fichero, Cliente::*ofFormat*, Trayecto::*ofFormat*, Graphs2::*simpleWeightedGraph*);

*toConsole*();

}

**public** **static** Integer getNVertices() {

**return** *gf*.vertexSet().size();

}

@SuppressWarnings("exports")

**public** **static** Cliente getCliente(Integer i) {

Cliente client = **null**;

List<Cliente> vs = **new** ArrayList<>(*gf*.vertexSet());

**for** (**int** k = 0; k < vs.size(); k++) {

**if** (vs.get(k).id() == i) {

client = vs.get(k);

}

}

**return** client;

}

**public** **static** Double getBeneficio(Integer i) {

Cliente client = *getCliente*(i);

**return** client.beneficio();

}

**public** **static** Double getPeso(Integer i, Integer j) {

Cliente cliente1 = *getCliente*(i);

Cliente cliente2 = *getCliente*(j);

**return** *gf*.getEdge(cliente1, cliente2).distancia();

}

**public** **static** Boolean existeArista(Integer i, Integer j) {

Cliente client1 = *getCliente*(i);

Cliente client2 = *getCliente*(j);

**return** *gf*.containsEdge(client1, client2);

}

**public** **static** **void** toConsole() {

System.***out***.println("Numero de vertices: " + *gf*.vertexSet().size() + "\n\tVertices: " + *gf*.vertexSet()

+ "\nNumero de aristas: " + *gf*.edgeSet().size() + "\n\tAristas: " + *gf*.edgeSet());

}

**public** **static** **void** main(String[] args) {

**for** (**int** i = 1; i < 3; i++) {

System.***out***.println("\n######################## DATOS FICHERO " + i + " ########################\n");

String fich = "ficheros/Ejercicio4DatosEntrada" + i + ".txt";

*iniDatos*(fich);

System.***out***.println("\n");

}

}

}

**PermutaGrafoAG**

package ejercicio4;

import java.util.List;

import datos.DatosEjercicioClientes;

import soluciones.SolucionClientes;

import us.lsi.ag.SeqNormalData;

import us.lsi.ag.agchromosomes.ChromosomeFactory.ChromosomeType;

public class PermutaGrafoAG implements SeqNormalData<SolucionClientes> {

public PermutaGrafoAG(String fich) {

DatosEjercicioClientes.iniDatos(fich);

}

@Override

public Integer itemsNumber() {

return DatosEjercicioClientes.getNVertices();

}

@Override

public ChromosomeType type() {

return ChromosomeType.Permutation;

}

@Override

public Double fitnessFunction(List<Integer> ls\_chrm) {

double goal = 0, error = 0, k = 0, suma = 0;

for (int i = 0; i < ls\_chrm.size(); i++) {

if (i == 0) {

if (DatosEjercicioClientes.existeArista(0, ls\_chrm.get(i))) {

suma += DatosEjercicioClientes.getPeso(0, ls\_chrm.get(i));

goal += DatosEjercicioClientes.getBeneficio(ls\_chrm.get(i)) - suma;

} else {

error++; // SI NO EXISTE ARISTA PENALIZAMOS

}

} else {

if (DatosEjercicioClientes.existeArista(ls\_chrm.get(i - 1), ls\_chrm.get(i))) {

suma += DatosEjercicioClientes.getPeso(ls\_chrm.get(i - 1), ls\_chrm.get(i));

goal += DatosEjercicioClientes.getBeneficio(ls\_chrm.get(i)) - suma;

} else {

error++;

}

}

}

if (ls\_chrm.get(ls\_chrm.size() - 1) != 0) { // SI ULTIMO VERTICE != 0

if (error == 0) { // SI PENALIZACION = 0 LA HACEMOS = 2

error += 2;

} else { // SI PENALIZACION != 0 LA MULTIPLICMAOS POR 2

error = error \* 2;

}

}

suma = 0.;

for (int i = 0; i < ls\_chrm.size(); i++) {

suma += DatosEjercicioClientes.getBeneficio(ls\_chrm.get(i));

}

k = Math.pow(suma, 2);

return goal - k \* error;

}

@Override

public SolucionClientes solucion(List<Integer> ls\_chrm) {

return SolucionClientes.of\_Rnage(ls\_chrm);

}

}

**TestGrafoAGPermuta**

**package** ejercicio4;

**import** java.util.List;

**import** java.util.Locale;

**import** soluciones.SolucionClientes;

**import** us.lsi.ag.agchromosomes.AlgoritmoAG;

**import** us.lsi.ag.agstopping.StoppingConditionFactory;

**public** **class** TestGrafoAGPermuta {

**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++) {

System.***out***.println("\n\n#####################################################################################");

System.***out***.println("SOLUCION EJERCICIO 4 CON DATOS DE ENTRADA " + i);

System.***out***.println("#####################################################################################\n");

PermutaGrafoAG p = **new** PermutaGrafoAG("ficheros/Ejercicio4DatosEntrada" + i + ".txt");

AlgoritmoAG<List<Integer>, SolucionClientes> ap = AlgoritmoAG.*of*(p);

ap.ejecuta();

System.***out***.println("================================");

System.***out***.println(ap.bestSolution());

System.***out***.println("================================");

}

}

}

**Resultados**

**Ejercicio 1 PLE**

#####################################################################################

FICHERO EJERCICIO 1 CON DATOS DE ENTRADA 1

#####################################################################################

Cantidad disponible tipo - [5, 4, 1, 2, 8, 1]

Variedad disponible - [Variedad[id=0, beneficio=20, mezcla=[0.5, 0.4, 0.1, 0.0, 0.0, 0.0]], Variedad[id=1, beneficio=10, mezcla=[0.0, 0.0, 0.0, 0.2, 0.8, 0.0]], Variedad[id=2, beneficio=5, mezcla=[0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]]

(getCantidadTipoVariedad,public static java.lang.Double ejercicio1.Ejercicio1PLE.getCantidadTipoVariedad(java.lang.Integer,java.lang.Integer))

(getNumeroTipos,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getNumeroTipos())

(getBeneficio,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getBeneficio(java.lang.Integer))

(getCantidad,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getCantidad(java.lang.Integer))

(getNumeroVariedades,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getNumeroVariedades())

(getCantidadTipoVariedad,{INT,INT}),(getNumeroTipos,{}),(getBeneficio,{INT}),(getCantidad,{INT}),(getNumeroVariedades,{})

(m,3),(n,6)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio1-1.lp

Reading time = 0.01 seconds

: 6 rows, 3 columns, 6 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 6 rows, 3 columns and 6 nonzeros

Model fingerprint: 0xb6e97bb3

Variable types: 0 continuous, 3 integer (0 binary)

Coefficient statistics:

Matrix range [1e-01, 1e+00]

Objective range [5e+00, 2e+01]

Bounds range [0e+00, 0e+00]

RHS range [1e+00, 8e+00]

Found heuristic solution: objective 305.0000000

Presolve removed 6 rows and 3 columns

Presolve time: 0.01s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.02 seconds (0.00 work units)

Thread count was 1 (of 20 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 EJERCICIO 1 CON DATOS DE ENTRADA 2

#####################################################################################

Cantidad disponible tipo - [11, 9, 7, 12, 6]

Variedad disponible - [Variedad[id=0, beneficio=20, mezcla=[0.2, 0.4, 0.0, 0.0, 0.4]], Variedad[id=1, beneficio=10, mezcla=[0.0, 0.3, 0.7, 0.0, 0.0]], Variedad[id=2, beneficio=80, mezcla=[0.4, 0.0, 0.0, 0.6, 0.0]]]

(getCantidadTipoVariedad,public static java.lang.Double ejercicio1.Ejercicio1PLE.getCantidadTipoVariedad(java.lang.Integer,java.lang.Integer))

(getNumeroTipos,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getNumeroTipos())

(getBeneficio,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getBeneficio(java.lang.Integer))

(getCantidad,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getCantidad(java.lang.Integer))

(getNumeroVariedades,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getNumeroVariedades())

(getCantidadTipoVariedad,{INT,INT}),(getNumeroTipos,{}),(getBeneficio,{INT}),(getCantidad,{INT}),(getNumeroVariedades,{})

(i,2),(j,5),(m,3),(n,5)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio1-2.lp

Reading time = 0.00 seconds

: 5 rows, 3 columns, 7 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 5 rows, 3 columns and 7 nonzeros

Model fingerprint: 0x3a3e91d5

Variable types: 0 continuous, 3 integer (0 binary)

Coefficient statistics:

Matrix range [2e-01, 7e-01]

Objective range [1e+01, 8e+01]

Bounds range [0e+00, 0e+00]

RHS range [6e+00, 1e+01]

Found heuristic solution: objective 2000.0000000

Presolve removed 5 rows and 3 columns

Presolve time: 0.00s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)

Thread count was 1 (of 20 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 EJERCICIO 1 CON DATOS DE ENTRADA 3

#####################################################################################

Cantidad disponible tipo - [35, 4, 12, 5, 30, 42, 3, 2, 20, 3]

Variedad disponible - [Variedad[id=0, beneficio=60, mezcla=[0.5, 0.0, 0.4, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0]], Variedad[id=1, beneficio=25, mezcla=[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], Variedad[id=2, beneficio=5, mezcla=[0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0]], Variedad[id=3, beneficio=25, mezcla=[0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.2]], Variedad[id=4, beneficio=15, mezcla=[0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.6, 0.0, 0.0]], Variedad[id=5, beneficio=100, mezcla=[0.2, 0.0, 0.0, 0.0, 0.3, 0.3, 0.0, 0.0, 0.2, 0.0]]]

(getCantidadTipoVariedad,public static java.lang.Double ejercicio1.Ejercicio1PLE.getCantidadTipoVariedad(java.lang.Integer,java.lang.Integer))

(getNumeroTipos,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getNumeroTipos())

(getBeneficio,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getBeneficio(java.lang.Integer))

(getCantidad,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getCantidad(java.lang.Integer))

(getNumeroVariedades,public static java.lang.Integer ejercicio1.Ejercicio1PLE.getNumeroVariedades())

(getCantidadTipoVariedad,{INT,INT}),(getNumeroTipos,{}),(getBeneficio,{INT}),(getCantidad,{INT}),(getNumeroVariedades,{})

(i,2),(j,4),(m,6),(n,10)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio1-3.lp

Reading time = 0.00 seconds

: 10 rows, 6 columns, 14 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 10 rows, 6 columns and 14 nonzeros

Model fingerprint: 0xa70f2246

Variable types: 0 continuous, 6 integer (0 binary)

Coefficient statistics:

Matrix range [1e-01, 1e+00]

Objective range [5e+00, 1e+02]

Bounds range [0e+00, 0e+00]

RHS range [2e+00, 4e+01]

Found heuristic solution: objective 12275.000000

Presolve removed 10 rows and 6 columns

Presolve time: 0.00s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)

Thread count was 1 (of 20 available processors)

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

**Ejercicio 1 AG**

#####################################################################################

SOLUCION EJERCICIO 1 CON DATOS DE ENTRADA 1

#####################################################################################

Cantidad disponible tipo - [5, 4, 1, 2, 8, 1]

Variedad disponible - [Variedad[id=0, beneficio=20, mezcla=[0.5, 0.4, 0.1, 0.0, 0.0, 0.0]], Variedad[id=1, beneficio=10, mezcla=[0.0, 0.0, 0.0, 0.2, 0.8, 0.0]], Variedad[id=2, beneficio=5, mezcla=[0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]]

================================

Variedades de cafe seleccionadas:

P1: 10 Kgs

P2: 10 Kgs

P3: 1 Kg

Beneficio: 305.0

================================

#####################################################################################

SOLUCION EJERCICIO 1 CON DATOS DE ENTRADA 2

#####################################################################################

Cantidad disponible tipo - [11, 9, 7, 12, 6]

Variedad disponible - [Variedad[id=0, beneficio=20, mezcla=[0.2, 0.4, 0.0, 0.0, 0.4]], Variedad[id=1, beneficio=10, mezcla=[0.0, 0.3, 0.7, 0.0, 0.0]], Variedad[id=2, beneficio=80, mezcla=[0.4, 0.0, 0.0, 0.6, 0.0]]]

================================

Variedades de cafe seleccionadas:

P1: 15 Kgs

P2: 10 Kgs

P3: 20 Kg

Beneficio: 2000.0

================================

#####################################################################################

SOLUCION EJERCICIO 1 CON DATOS DE ENTRADA 3

#####################################################################################

Cantidad disponible tipo - [35, 4, 12, 5, 30, 42, 3, 2, 20, 3]

Variedad disponible - [Variedad[id=0, beneficio=60, mezcla=[0.5, 0.0, 0.4, 0.0, 0.0, 0.0, 0.1, 0.0, 0.0, 0.0]], Variedad[id=1, beneficio=25, mezcla=[0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]], Variedad[id=2, beneficio=5, mezcla=[0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0]], Variedad[id=3, beneficio=25, mezcla=[0.0, 0.0, 0.0, 0.0, 0.0, 0.8, 0.0, 0.0, 0.0, 0.2]], Variedad[id=4, beneficio=15, mezcla=[0.0, 0.0, 0.4, 0.0, 0.0, 0.0, 0.0, 0.6, 0.0, 0.0]], Variedad[id=5, beneficio=100, mezcla=[0.2, 0.0, 0.0, 0.0, 0.3, 0.3, 0.0, 0.0, 0.2, 0.0]]]

================================

Variedades de cafe seleccionadas:

P1: 30 Kgs

P2: 4 Kgs

P4: 15 Kgs

P6: 100 Kg

Beneficio: 12275.0

================================

**Ejercicio 2 PLE**

#####################################################################################

FICHERO EJERCICIO 2 CON DATOS DE ENTRADA 1

#####################################################################################

Maximo de centros selecionables: 1

Cursos disponibles: [Curso[id=0, tematicas=[1, 2, 3, 4], precio=10.0, centro=0], Curso[id=1, tematicas=[1, 4], precio=3.0, centro=0], Curso[id=2, tematicas=[5], precio=1.5, centro=1], Curso[id=3, tematicas=[5], precio=5.0, centro=0]]

(getMaxCentros,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getMaxCentros())

(ofreceCurso,public static java.lang.Integer ejercicio2.Ejercicio2PLE.ofreceCurso(java.lang.Integer,java.lang.Integer))

(getNumeroTematicas,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroTematicas())

(getPrecioCurso,public static java.lang.Double ejercicio2.Ejercicio2PLE.getPrecioCurso(java.lang.Integer))

(getNumeroCentros,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroCentros())

(getNumeroCursos,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroCursos())

(contieneTematica,public static java.lang.Integer ejercicio2.Ejercicio2PLE.contieneTematica(java.lang.Integer,java.lang.Integer))

(getMaxCentros,{}),(ofreceCurso,{INT,INT}),(getNumeroTematicas,{}),(getPrecioCurso,{INT}),(getNumeroCentros,{}),(getNumeroCursos,{}),(contieneTematica,{INT,INT})

(c,2),(m,5),(n,4),(mCentros,1)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio2-1.lp

Reading time = 0.00 seconds

: 14 rows, 6 columns, 22 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 14 rows, 6 columns and 22 nonzeros

Model fingerprint: 0x5498154c

Variable types: 0 continuous, 6 integer (6 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+00]

Objective range [2e+00, 1e+01]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 1e+00]

Presolve removed 14 rows and 6 columns

Presolve time: 0.00s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)

Thread count was 1 (of 20 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 EJERCICIO 2 CON DATOS DE ENTRADA 2

#####################################################################################

Maximo de centros selecionables: 2

Cursos disponibles: [Curso[id=0, tematicas=[2, 3], precio=2.0, centro=0], Curso[id=1, tematicas=[4], precio=3.0, centro=0], Curso[id=2, tematicas=[1, 5], precio=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], precio=3.5, centro=2], Curso[id=4, tematicas=[4, 5], precio=1.5, centro=1]]

(getMaxCentros,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getMaxCentros())

(ofreceCurso,public static java.lang.Integer ejercicio2.Ejercicio2PLE.ofreceCurso(java.lang.Integer,java.lang.Integer))

(getNumeroTematicas,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroTematicas())

(getPrecioCurso,public static java.lang.Double ejercicio2.Ejercicio2PLE.getPrecioCurso(java.lang.Integer))

(getNumeroCentros,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroCentros())

(getNumeroCursos,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroCursos())

(contieneTematica,public static java.lang.Integer ejercicio2.Ejercicio2PLE.contieneTematica(java.lang.Integer,java.lang.Integer))

(getMaxCentros,{}),(ofreceCurso,{INT,INT}),(getNumeroTematicas,{}),(getPrecioCurso,{INT}),(getNumeroCentros,{}),(getNumeroCursos,{}),(contieneTematica,{INT,INT})

(c,3),(i,3),(j,4),(k,1),(m,5),(n,5),(mCentros,2)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio2-2.lp

Reading time = 0.00 seconds

: 21 rows, 8 columns, 33 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 21 rows, 8 columns and 33 nonzeros

Model fingerprint: 0x5502a90d

Variable types: 0 continuous, 8 integer (8 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+00]

Objective range [2e+00, 5e+00]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 2e+00]

Presolve removed 21 rows and 8 columns

Presolve time: 0.00s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)

Thread count was 1 (of 20 available processors)

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 EJERCICIO 2 CON DATOS DE ENTRADA 3

#####################################################################################

Maximo de centros selecionables: 3

Cursos disponibles: [Curso[id=0, tematicas=[2, 6, 7], precio=2.0, centro=2], Curso[id=1, tematicas=[7], precio=3.0, centro=0], Curso[id=2, tematicas=[1, 5], precio=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], precio=3.5, centro=2], Curso[id=4, tematicas=[3, 7], precio=1.5, centro=1], Curso[id=5, tematicas=[4, 5, 6], precio=4.5, centro=0], Curso[id=6, tematicas=[6, 5], precio=6.0, centro=1], Curso[id=7, tematicas=[2, 3, 5], precio=1.0, centro=1]]

(getMaxCentros,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getMaxCentros())

(ofreceCurso,public static java.lang.Integer ejercicio2.Ejercicio2PLE.ofreceCurso(java.lang.Integer,java.lang.Integer))

(getNumeroTematicas,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroTematicas())

(getPrecioCurso,public static java.lang.Double ejercicio2.Ejercicio2PLE.getPrecioCurso(java.lang.Integer))

(getNumeroCentros,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroCentros())

(getNumeroCursos,public static java.lang.Integer ejercicio2.Ejercicio2PLE.getNumeroCursos())

(contieneTematica,public static java.lang.Integer ejercicio2.Ejercicio2PLE.contieneTematica(java.lang.Integer,java.lang.Integer))

(getMaxCentros,{}),(ofreceCurso,{INT,INT}),(getNumeroTematicas,{}),(getPrecioCurso,{INT}),(getNumeroCentros,{}),(getNumeroCursos,{}),(contieneTematica,{INT,INT})

(c,3),(i,4),(j,4),(k,2),(m,7),(n,8),(mCentros,3)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio2-3.lp

Reading time = 0.00 seconds

: 32 rows, 11 columns, 54 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 32 rows, 11 columns and 54 nonzeros

Model fingerprint: 0x00e94806

Variable types: 0 continuous, 11 integer (11 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+00]

Objective range [1e+00, 6e+00]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 3e+00]

Found heuristic solution: objective 12.0000000

Found heuristic solution: objective 6.5000000

Presolve removed 32 rows and 11 columns

Presolve time: 0.00s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)

Thread count was 1 (of 20 available processors)

Solution count 2: 6.5 12

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

**Ejercicio 2 AG**

#####################################################################################

SOLUCION EJERCICIO 2 CON DATOS DE ENTRADA 1

#####################################################################################

Maximo de centros selecionables: 1

Cursos disponibles: [Curso[id=0, tematicas=[1, 2, 3, 4], precio=10.0, centro=0], Curso[id=1, tematicas=[1, 4], precio=3.0, centro=0], Curso[id=2, tematicas=[5], precio=1.5, centro=1], Curso[id=3, tematicas=[5], precio=5.0, centro=0]]

================================

Cursos elegidos: {S0, S3}

Coste total: 15.0

================================

#####################################################################################

SOLUCION EJERCICIO 2 CON DATOS DE ENTRADA 2

#####################################################################################

Maximo de centros selecionables: 2

Cursos disponibles: [Curso[id=0, tematicas=[2, 3], precio=2.0, centro=0], Curso[id=1, tematicas=[4], precio=3.0, centro=0], Curso[id=2, tematicas=[1, 5], precio=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], precio=3.5, centro=2], Curso[id=4, tematicas=[4, 5], precio=1.5, centro=1]]

================================

Cursos elegidos: {S0, S2, S4}

Coste total: 8.5

================================

#####################################################################################

SOLUCION EJERCICIO 2 CON DATOS DE ENTRADA 3

#####################################################################################

Maximo de centros selecionables: 3

Cursos disponibles: [Curso[id=0, tematicas=[2, 6, 7], precio=2.0, centro=2], Curso[id=1, tematicas=[7], precio=3.0, centro=0], Curso[id=2, tematicas=[1, 5], precio=5.0, centro=0], Curso[id=3, tematicas=[1, 3, 4], precio=3.5, centro=2], Curso[id=4, tematicas=[3, 7], precio=1.5, centro=1], Curso[id=5, tematicas=[4, 5, 6], precio=4.5, centro=0], Curso[id=6, tematicas=[6, 5], precio=6.0, centro=1], Curso[id=7, tematicas=[2, 3, 5], precio=1.0, centro=1]]

================================

Cursos elegidos: {S0, S3, S7}

Coste total: 6.5

================================

**Ejercicio3 PLE**

#####################################################################################

FICHERO EJERCICIO 3 CON DATOS DE ENTRADA 1

#####################################################################################

[Investigador[id=0, capacidad=6, especialidad=0], Investigador[id=1, capacidad=3, especialidad=1], Investigador[id=2, capacidad=8, especialidad=2]]

[Trabajo[id=0, calidad=5, dias=[6, 0, 0]], Trabajo[id=1, calidad=10, dias=[0, 3, 8]]]

(getNumeroInvestigadores,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroInvestigadores())

(trabajadorEspecialidad,public static java.lang.Integer ejercicio3.Ejercicio3PLE.trabajadorEspecialidad(java.lang.Integer,java.lang.Integer))

(diasDisponibles,public static java.lang.Integer ejercicio3.Ejercicio3PLE.diasDisponibles(java.lang.Integer))

(getNumeroTrabajos,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroTrabajos())

(getNumeroEspecialidades,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroEspecialidades())

(diasNecesarios,public static java.lang.Integer ejercicio3.Ejercicio3PLE.diasNecesarios(java.lang.Integer,java.lang.Integer))

(getCalidad,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getCalidad(java.lang.Integer))

(getMM,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getMM())

(getNumeroInvestigadores,{}),(trabajadorEspecialidad,{INT,INT}),(diasDisponibles,{INT}),(getNumeroTrabajos,{}),(getNumeroEspecialidades,{}),(diasNecesarios,{INT,INT}),(getCalidad,{INT}),(getMM,{})

(MM,9),(e,3),(m,2),(n,3)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio3-1.lp

Reading time = 0.00 seconds

: 15 rows, 8 columns, 27 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 15 rows, 8 columns and 27 nonzeros

Model fingerprint: 0xa0d38001

Variable types: 0 continuous, 8 integer (0 binary)

Coefficient statistics:

Matrix range [1e+00, 9e+00]

Objective range [5e+00, 1e+01]

Bounds range [1e+00, 1e+00]

RHS range [3e+00, 8e+00]

Found heuristic solution: objective -0.0000000

Presolve removed 15 rows and 8 columns

Presolve time: 0.00s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)

Thread count was 1 (of 20 available processors)

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 EJERCICIO 3 CON DATOS DE ENTRADA 2

#####################################################################################

[Investigador[id=0, capacidad=10, especialidad=0], Investigador[id=1, capacidad=5, especialidad=1], Investigador[id=2, capacidad=8, especialidad=2], Investigador[id=3, capacidad=2, especialidad=0], Investigador[id=4, capacidad=5, especialidad=3]]

[Trabajo[id=0, calidad=7, dias=[2, 0, 5, 0]], Trabajo[id=1, calidad=9, dias=[8, 4, 3, 0]], Trabajo[id=2, calidad=5, dias=[2, 0, 0, 7]]]

(getNumeroInvestigadores,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroInvestigadores())

(trabajadorEspecialidad,public static java.lang.Integer ejercicio3.Ejercicio3PLE.trabajadorEspecialidad(java.lang.Integer,java.lang.Integer))

(diasDisponibles,public static java.lang.Integer ejercicio3.Ejercicio3PLE.diasDisponibles(java.lang.Integer))

(getNumeroTrabajos,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroTrabajos())

(getNumeroEspecialidades,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroEspecialidades())

(diasNecesarios,public static java.lang.Integer ejercicio3.Ejercicio3PLE.diasNecesarios(java.lang.Integer,java.lang.Integer))

(getCalidad,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getCalidad(java.lang.Integer))

(getMM,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getMM())

(getNumeroInvestigadores,{}),(trabajadorEspecialidad,{INT,INT}),(diasDisponibles,{INT}),(getNumeroTrabajos,{}),(getNumeroEspecialidades,{}),(diasNecesarios,{INT,INT}),(getCalidad,{INT}),(getMM,{})

(MM,11),(e,4),(i,2),(j,1),(k,2),(m,3),(n,5)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio3-2.lp

Reading time = 0.00 seconds

: 32 rows, 18 columns, 67 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 32 rows, 18 columns and 67 nonzeros

Model fingerprint: 0x8fbdfd67

Variable types: 0 continuous, 18 integer (0 binary)

Coefficient statistics:

Matrix range [1e+00, 1e+01]

Objective range [5e+00, 9e+00]

Bounds range [1e+00, 1e+00]

RHS range [2e+00, 1e+01]

Found heuristic solution: objective -0.0000000

Presolve removed 32 rows and 18 columns

Presolve time: 0.00s

Presolve: All rows and columns removed

Explored 0 nodes (0 simplex iterations) in 0.00 seconds (0.00 work units)

Thread count was 1 (of 20 available processors)

Solution count 2: 16 -0

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 EJERCICIO 3 CON DATOS DE ENTRADA 3

#####################################################################################

[Investigador[id=0, capacidad=1, especialidad=2], Investigador[id=1, capacidad=10, especialidad=1], Investigador[id=2, capacidad=3, especialidad=0], Investigador[id=3, capacidad=4, especialidad=0], Investigador[id=4, capacidad=10, especialidad=3], Investigador[id=5, capacidad=4, especialidad=3], Investigador[id=6, capacidad=1, especialidad=2], Investigador[id=7, capacidad=30, especialidad=3]]

[Trabajo[id=0, calidad=8, dias=[2, 0, 2, 0]], Trabajo[id=1, calidad=5, dias=[8, 5, 4, 2]], Trabajo[id=2, calidad=8, dias=[0, 5, 0, 15]], Trabajo[id=3, calidad=5, dias=[0, 7, 8, 5]], Trabajo[id=4, calidad=9, dias=[5, 5, 0, 2]]]

(getNumeroInvestigadores,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroInvestigadores())

(trabajadorEspecialidad,public static java.lang.Integer ejercicio3.Ejercicio3PLE.trabajadorEspecialidad(java.lang.Integer,java.lang.Integer))

(diasDisponibles,public static java.lang.Integer ejercicio3.Ejercicio3PLE.diasDisponibles(java.lang.Integer))

(getNumeroTrabajos,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroTrabajos())

(getNumeroEspecialidades,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getNumeroEspecialidades())

(diasNecesarios,public static java.lang.Integer ejercicio3.Ejercicio3PLE.diasNecesarios(java.lang.Integer,java.lang.Integer))

(getCalidad,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getCalidad(java.lang.Integer))

(getMM,public static java.lang.Integer ejercicio3.Ejercicio3PLE.getMM())

(getNumeroInvestigadores,{}),(trabajadorEspecialidad,{INT,INT}),(diasDisponibles,{INT}),(getNumeroTrabajos,{}),(getNumeroEspecialidades,{}),(diasNecesarios,{INT,INT}),(getCalidad,{INT}),(getMM,{})

(MM,31),(e,4),(i,4),(j,2),(k,3),(m,5),(n,8)

====================

Tenga en cuenta que el formato intermedio LP no distingue entre desigualdades estrictas y no estrictas en las restricciones

Por lo que, por ejemplo, < y <= son equivalentes.

====================

Set parameter Username

Academic license - for non-commercial use only - expires 2024-02-15

Warning: Gurobi version mismatch between Java 9.5.0 and C library 9.5.2

Warning: Gurobi version mismatch between Java 9.5.0 and Jni 9.5.2

Read LP format model from file gurobi\_models/Ejercicio3-3.lp

Reading time = 0.00 seconds

: 68 rows, 45 columns, 174 nonzeros

Gurobi Optimizer version 9.5.2 build v9.5.2rc0 (win64)

Thread count: 14 physical cores, 20 logical processors, using up to 20 threads

Optimize a model with 68 rows, 45 columns and 174 nonzeros

Model fingerprint: 0x91312e9e

Variable types: 0 continuous, 45 integer (0 binary)

Coefficient statistics:

Matrix range [1e+00, 3e+01]

Objective range [5e+00, 9e+00]

Bounds range [1e+00, 1e+00]

RHS range [1e+00, 3e+01]

Found heuristic solution: objective -0.0000000

Presolve removed 62 rows and 40 columns

Presolve time: 0.00s

Presolved: 6 rows, 5 columns, 14 nonzeros

Found heuristic solution: objective 16.0000000

Variable types: 0 continuous, 5 integer (2 binary)

Root relaxation: objective 2.500000e+01, 1 iterations, 0.00 seconds (0.00 work units)

Nodes | Current Node | Objective Bounds | Work

Expl Unexpl | Obj Depth IntInf | Incumbent BestBd Gap | It/Node Time

\* 0 0 0 25.0000000 25.00000 0.00% - 0s

Explored 1 nodes (1 simplex iterations) in 0.02 seconds (0.00 work units)

Thread count was 20 (of 20 available processors)

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

**Ejercicio 3 AG**

#####################################################################################

SOLUCION EJERCICIO 3 CON DATOS DE ENTRADA 1

#####################################################################################

[Investigador[id=0, capacidad=6, especialidad=0], Investigador[id=1, capacidad=3, especialidad=1], Investigador[id=2, capacidad=8, especialidad=2]]

[Trabajo[id=0, calidad=5, dias=[6, 0, 0]], Trabajo[id=1, calidad=10, dias=[0, 3, 8]]]

================================

[6, 0, 0, 0, 3, 8]

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

================================

#####################################################################################

SOLUCION EJERCICIO 3 CON DATOS DE ENTRADA 2

#####################################################################################

[Investigador[id=0, capacidad=10, especialidad=0], Investigador[id=1, capacidad=5, especialidad=1], Investigador[id=2, capacidad=8, especialidad=2], Investigador[id=3, capacidad=2, especialidad=0], Investigador[id=4, capacidad=5, especialidad=3]]

[Trabajo[id=0, calidad=7, dias=[2, 0, 5, 0]], Trabajo[id=1, calidad=9, dias=[8, 4, 3, 0]], Trabajo[id=2, calidad=5, dias=[2, 0, 0, 7]]]

================================

[2, 0, 5, 0, 0, 7, 4, 3, 1, 0, 1, 0, 0, 1, 5]

Reparto obtenido (dias trabajados por cada investigador en cada trabajo):

INV1: [2, 7, 1]

INV2: [0, 4, 0]

INV3: [5, 3, 0]

INV4: [0, 1, 1]

INV5: [0, 0, 5]

SUMA DE LAS CALIDADES DE LOS TRABAJOS REALIZADOS: 16

================================

#####################################################################################

SOLUCION EJERCICIO 3 CON DATOS DE ENTRADA 3

#####################################################################################

[Investigador[id=0, capacidad=1, especialidad=2], Investigador[id=1, capacidad=10, especialidad=1], Investigador[id=2, capacidad=3, especialidad=0], Investigador[id=3, capacidad=4, especialidad=0], Investigador[id=4, capacidad=10, especialidad=3], Investigador[id=5, capacidad=4, especialidad=3], Investigador[id=6, capacidad=1, especialidad=2], Investigador[id=7, capacidad=30, especialidad=3]]

[Trabajo[id=0, calidad=8, dias=[2, 0, 2, 0]], Trabajo[id=1, calidad=5, dias=[8, 5, 4, 2]], Trabajo[id=2, calidad=8, dias=[0, 5, 0, 15]], Trabajo[id=3, calidad=5, dias=[0, 7, 8, 5]], Trabajo[id=4, calidad=9, dias=[5, 5, 0, 2]]]

================================

[1, 0, 2, 0, 0, 0, 1, 0, 0, 4, 1, 1, 1, 0, 0, 1, 0, 5, 0, 0, 2, 1, 0, 12, 0, 0, 0, 0, 1, 2, 1, 2, 0, 5, 1, 4, 0, 0, 0, 2]

Reparto obtenido (dias trabajados por cada investigador en cada trabajo):

INV1: [1, 0, 0, 0, 0]

INV2: [0, 4, 5, 0, 5]

INV3: [2, 1, 0, 0, 1]

INV4: [0, 1, 0, 0, 4]

INV5: [0, 1, 2, 1, 0]

INV6: [0, 0, 1, 2, 0]

INV7: [1, 0, 0, 1, 0]

INV8: [0, 1, 12, 2, 2]

SUMA DE LAS CALIDADES DE LOS TRABAJOS REALIZADOS: 25

================================

**Ejercicio 4 AG**

#####################################################################################

SOLUCION EJERCICIO 4 CON DATOS DE ENTRADA 1

#####################################################################################

Numero de vertices: 5

Vertices: [Cliente [id=0, beneficio=0.0], Cliente [id=1, beneficio=400.0], Cliente [id=2, beneficio=300.0], Cliente [id=3, beneficio=200.0], Cliente [id=4, beneficio=100.0]]

Numero de aristas: 8

Aristas: [Conexion [id=0, dist=1.0], Conexion [id=1, dist=100.0], Conexion [id=2, dist=1.0], Conexion [id=3, dist=100.0], Conexion [id=4, dist=1.0], Conexion [id=5, dist=1.0], Conexion [id=6, dist=100.0], Conexion [id=7, dist=5.0]]

================================

Camino desde 0 hasta 0:

[0, 1, 2, 3, 4, 0]

Kms: 9.0

Beneficio: 981.0

================================

#####################################################################################

SOLUCION EJERCICIO 4 CON DATOS DE ENTRADA 2

#####################################################################################

Numero de vertices: 8

Vertices: [Cliente [id=0, beneficio=0.0], Cliente [id=1, beneficio=100.0], Cliente [id=2, beneficio=200.0], Cliente [id=3, beneficio=300.0], Cliente [id=4, beneficio=200.0], Cliente [id=5, beneficio=300.0], Cliente [id=6, beneficio=200.0], Cliente [id=7, beneficio=200.0]]

Numero de aristas: 13

Aristas: [Conexion [id=8, dist=2.0], Conexion [id=9, dist=1.0], Conexion [id=10, dist=1.0], Conexion [id=11, dist=3.0], Conexion [id=12, dist=1.0], Conexion [id=13, dist=1.0], Conexion [id=14, dist=3.0], Conexion [id=15, dist=1.0], Conexion [id=16, dist=1.0], Conexion [id=17, dist=3.0], Conexion [id=18, dist=1.0], Conexion [id=19, dist=1.0], Conexion [id=20, dist=1.0]]

================================

Camino desde 0 hasta 0:

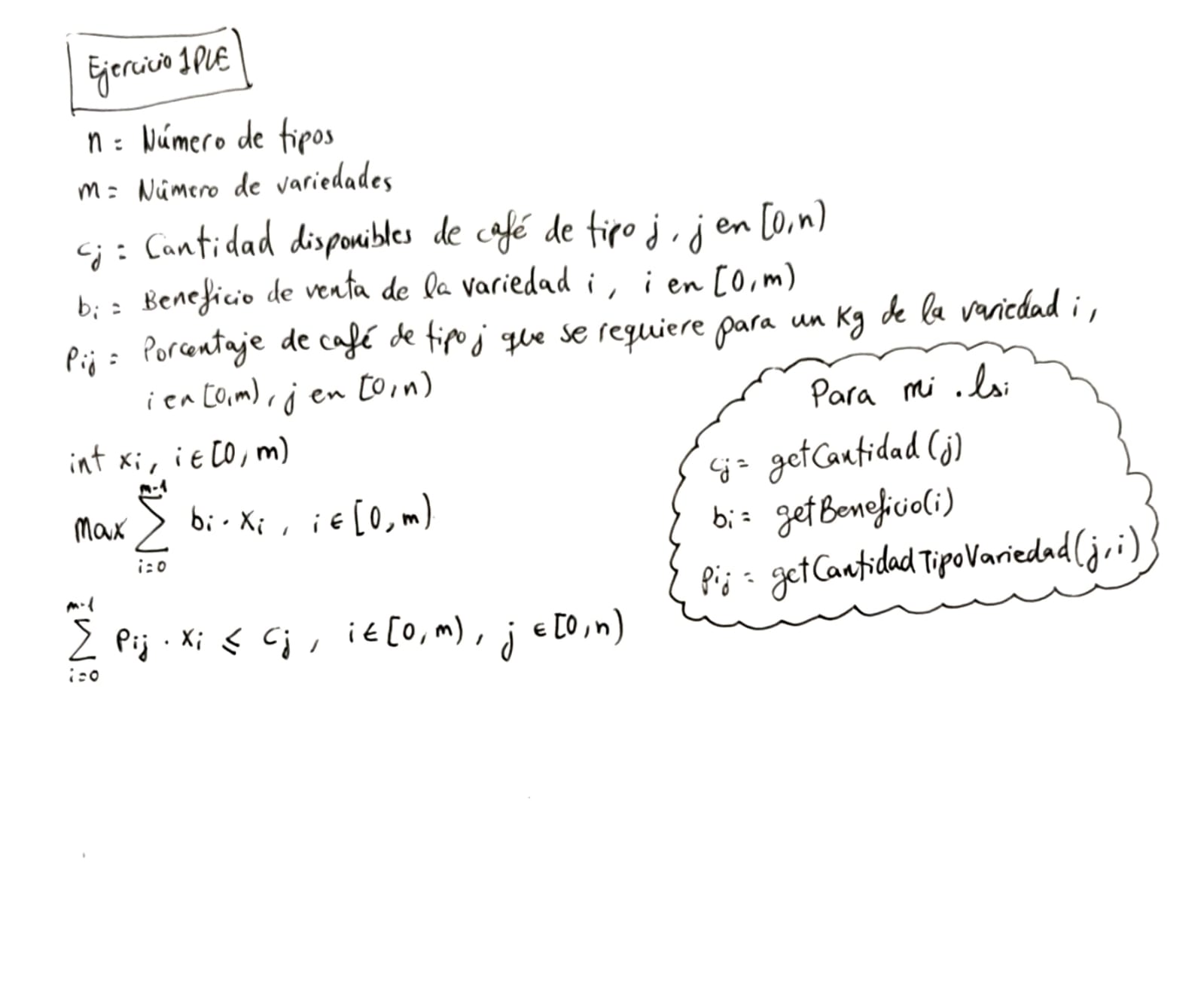
[0, 2, 5, 3, 7, 4, 6, 1, 0]

Kms: 9.0

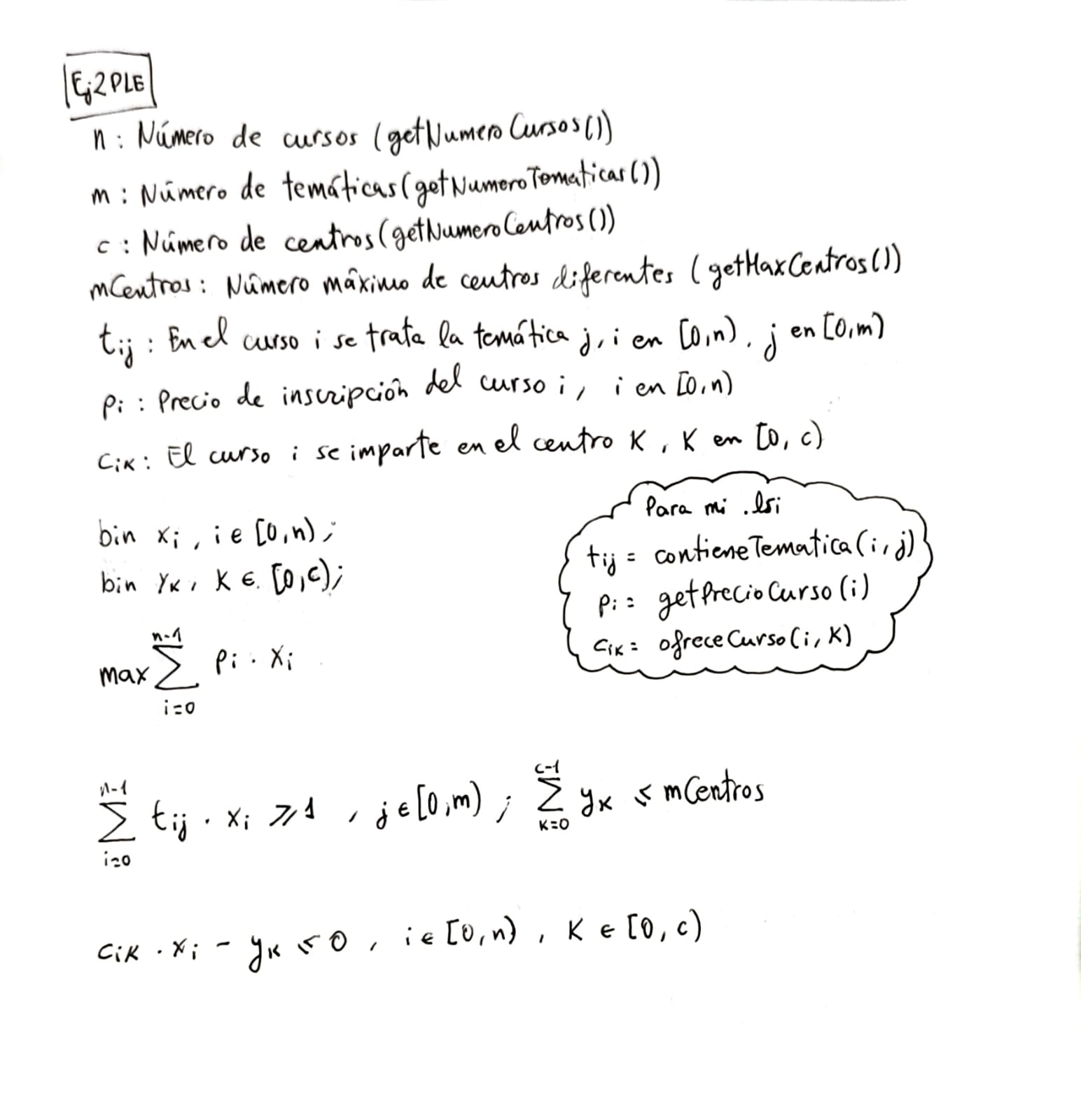
Beneficio: 1463.0

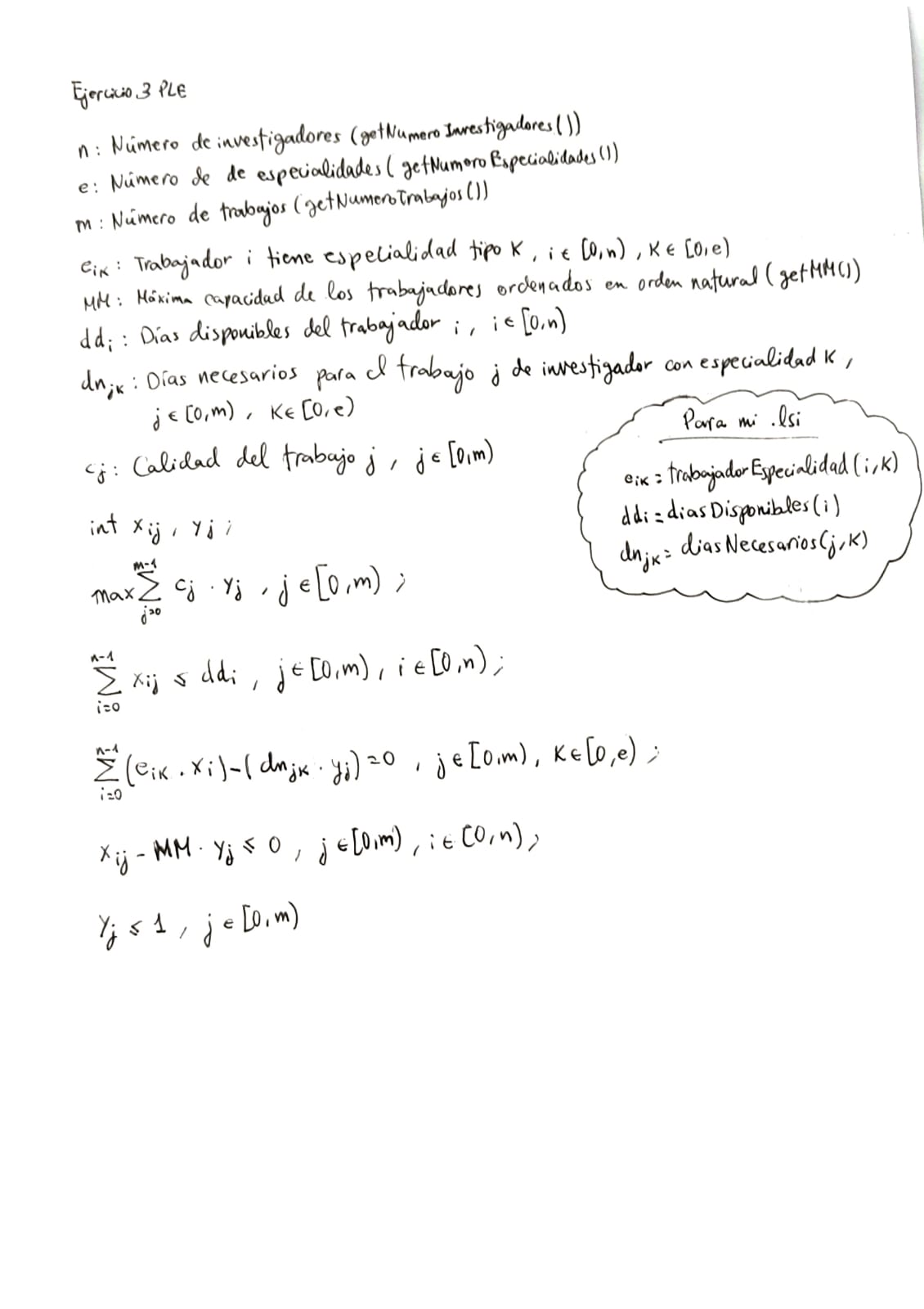
================================

**Formalización matemática**

**Ejercicio 1 PLE**

**Ejercicio 2 PLE**

****

**Ejercicio 3 PLE**

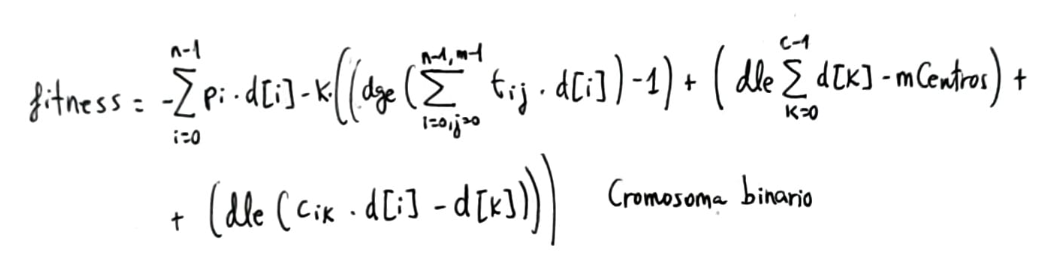


**Ejercicio 1 AG**

**Texto

Descripción generada automáticamente**

**Ejercicio 2 AG**

****

**Ejercicio 3 AG**

**Texto, Carta

Descripción generada automáticamente**

**Ejercicio 4 AG**

**Texto, Carta

Descripción generada automáticamente**