**Memoria PI5 juaorecar**

**Ejercicio 1**

**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, Double 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(";");

Double benef = Double.*parseDouble*(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 Double porcentaje(Integer k) {

return mezcla.get(k);

}

}

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 Double 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();

}

public static Variedad getVariedad(Integer i) {

return *variedades*.get(i);

}

public static Double getPorcentaje(Integer i, Integer j) {

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

}

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");

}

}

public static Integer getMaxKgVariedad(Integer i) {

List<Double> aux = new ArrayList<>();

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

if(*getPorcentaje*(i, j) != 0.0) {

aux.add(*getCantidad*(j) / *getPorcentaje*(i, j));

}

}

return aux.stream().min(Double::compareTo).get().intValue();

}

public static Integer getKilosMaximosVariedad(Integer i, List<Double> remaining) {

List<Double> aux = new ArrayList<>();

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

if(*getPorcentaje*(i, j) != 0.0) {

aux.add(remaining.get(j) / *getPorcentaje*(i, j));

}

}

return aux.stream().min(Double::compareTo).get().intValue();

}

}

**SolucionCafe**

package \_soluciones;

import java.util.List;

import org.jgrapht.GraphPath;

import \_datos.DatosEjercicioCafes;

import ejercicio1.CafeEdge;

import ejercicio1.CafeVertex;

public class SolucionCafe{

private double beneficioTotal;

private List<Integer> solucion;

public static SolucionCafe of(List<Integer> acciones) {

return new SolucionCafe(acciones);

}

public static SolucionCafe of(GraphPath<CafeVertex, CafeEdge> path) {

List<Integer> ls = path.getEdgeList().stream().map(e -> e.action() + 0).toList();

SolucionCafe res = *of*(ls);

res.solucion = ls;

return res;

}

private SolucionCafe(List<Integer> value) {

beneficioTotal = 0.0;

solucion = value;

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

beneficioTotal += DatosEjercicioCafes.*getBeneficio*(i)\*value.get(i);

}

}

public String toString() {

System.***out***.println("Variedades de cafes seleccionadas");

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

System.***out***.println(String.*format*("P%02d: %s Kgs", i+1, solucion.get(i)));

}

System.***out***.println(String.*format*("Beneficio: %s", beneficioTotal));

return "------------------------------";

}

}

**CafeEdge**

package ejercicio1;

import \_datos.DatosEjercicioCafes;

import us.lsi.graphs.virtual.SimpleEdgeAction;

public record CafeEdge(CafeVertex source, CafeVertex target, Integer action,

Double weight) implements SimpleEdgeAction<CafeVertex, Integer> {

public static CafeEdge of(CafeVertex s, CafeVertex t, Integer a) {

return new CafeEdge(s, t, a, Double.*valueOf*(a \* DatosEjercicioCafes.*getBeneficio*(s.index())));

}

}

**CafeVertex**

package ejercicio1;

import java.util.ArrayList;

import java.util.List;

import java.util.function.Predicate;

import \_datos.DatosEjercicioCafes;

import \_datos.DatosEjercicioCafes.Variedad;

import us.lsi.common.List2;

import us.lsi.graphs.virtual.VirtualVertex;

public record CafeVertex(Integer index, List<Double> remaining)

implements VirtualVertex<CafeVertex, CafeEdge, Integer>{

public static CafeVertex of(Integer i, List<Double> rest) {

return new CafeVertex(i, rest);

}

public static CafeVertex initial() {

List<Double> res = new ArrayList<>();

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

res.add(DatosEjercicioCafes.*getCantidad*(i) + 0.);

}

return *of*(0, res);

}

public static Predicate<CafeVertex> goal(){

return v -> v.index() == DatosEjercicioCafes.*getNumeroVariedades*();

}

public static Predicate<CafeVertex> goalHasSolution(){

return v -> v.remaining().stream().allMatch(e -> e.equals(0.));

}

public List<Integer> actions() {

List<Integer> alternativas = List2.*empty*();

if(index < DatosEjercicioCafes.*getNumeroVariedades*()) {

if(index == DatosEjercicioCafes.*getNumeroVariedades*()) {

alternativas = List2.*of*(DatosEjercicioCafes.*getKilosMaximosVariedad*(index, remaining) + 1);

}else {

alternativas = List2.*rangeList*(0, DatosEjercicioCafes.*getKilosMaximosVariedad*(index, remaining) + 1);

}

}

return alternativas;

}

public CafeVertex neighbor(Integer a) {

List<Double> res = List2.*empty*();

Variedad v = DatosEjercicioCafes.*getVariedades*().get(index);

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

if( (remaining().get(i) - v.porcentaje(i) \* a) > 0 ) {

Double aux = remaining().get(i) - v.porcentaje(i) \* a;

res.add(i, aux);

}else {

res.add(i, 0.);

}

}

// System.out.println(res);

return *of*(index + 1, res);

}

public CafeEdge edge(Integer a) {

return CafeEdge.*of*(this, neighbor(a), a);

}

*@Override*

public String toString() {

// **TODO** Auto-generated method stub

return null;

}

}

**CafeHeuristic**

package ejercicio1;

import java.util.function.Predicate;

import \_datos.DatosEjercicioCafes;

import \_datos.DatosEjercicioCafes.Variedad;

public class CafeHeuristic {

public static Double heuristic(CafeVertex v1, Predicate<CafeVertex> goal,

CafeVertex v2) {

Double beneficioTotalEstimado = 0.;

Integer indiceActual = v1.index(); // TENEMOS SELECCIONADA HASTA LA VARIEDAD i

Integer indiceFinal = DatosEjercicioCafes.*getNumeroVariedades*();

for(int i = indiceActual; i < indiceFinal; i++) {

Variedad variedad = DatosEjercicioCafes.*getVariedades*().get(i);

Integer kgMax = DatosEjercicioCafes.*getMaxKgVariedad*(i);

v1.remaining().stream().map(e -> e / variedad.porcentaje(v1.remaining().indexOf(e)));

Double beneficio = kgMax \* DatosEjercicioCafes.*getBeneficio*(i);

beneficioTotalEstimado += beneficio;

}

return beneficioTotalEstimado;

}

}

**CafePDR**

package ejercicio1.manual;

import java.util.Comparator;

import java.util.List;

import java.util.Map;

import \_datos.DatosEjercicioCafes;

import \_soluciones.SolucionCafe;

import us.lsi.common.List2;

import us.lsi.common.Map2;

public class CafePDR {

public static record Spm(Integer act, Integer benef)implements Comparable<Spm> {

public static Spm of(Integer a, Integer ben) {

return new Spm(a,ben);

}

*@Override*

public int compareTo(Spm o) {

return this.benef.compareTo(o.benef);

}

}

public static Map<CafeProblem, Spm> *memory*;

public static Integer *mejorValor*;

public static SolucionCafe search() {

*memory*= Map2.*empty*();

*mejorValor*= Integer.***MIN\_VALUE***;

*pdr\_search*(CafeProblem.*initial*(), 0, *memory*);

return *getSol*();

}

private static Spm pdr\_search(CafeProblem prob, int

acumulado, Map<CafeProblem, Spm> memoria) {

Spm res= null;

Boolean esTerminal =

prob.index().equals(DatosEjercicioCafes.*getNumeroVariedades*());

Boolean esSolucion= true;

if(*memory*.containsKey(prob)) {

res= *memory*.get(prob);

}else if(esTerminal && esSolucion) {

res= Spm.*of*(null, 0);

*memory*.put(prob, res);

if(acumulado>*mejorValor*) {

*mejorValor*= acumulado;

}

}else {

List<Spm> soluciones = List2.*empty*();

for(Integer action: prob.actions()) {

CafeProblem vecino= prob.neighbor(action);

Spm s= *pdr\_search*(vecino,acumulado+action\*DatosEjercicioCafes.*getMaxKgVariedad*(prob.index()), *memory*);

if(s!=null) {

Spm amp= Spm.*of*(action,

s.benef()+action\*DatosEjercicioCafes.*getMaxKgVariedad*(prob.index()));

soluciones.add(amp);

}

}

res= soluciones.stream().max(Comparator.*naturalOrder*()).orElse(null);

if( res!= null) {

*memory*.put(prob, res);

}

}

return res;

}

private static SolucionCafe getSol() {

List<Integer> acciones = List2.*empty*();

CafeProblem prob= CafeProblem.*initial*();

Spm spm= *memory*.get(prob);

while(spm != null && spm.act != null) {

CafeProblem old= prob;

acciones.add(spm.act);

prob = old.neighbor(spm.act);

spm= *memory*.get(prob);

}

return SolucionCafe.*of*(acciones);

}

}

**CafeProblem**

package ejercicio1.manual;

import java.util.ArrayList;

import java.util.List;

import java.util.function.Predicate;

import \_datos.DatosEjercicioCafes;

import \_datos.DatosEjercicioCafes.Variedad;

import ejercicio1.CafeVertex;

import us.lsi.common.List2;

public record CafeProblem(Integer index, List<Double> remaining) {

public static CafeProblem of(Integer i, List<Double> rest) {

return new CafeProblem(i, rest);

}

public static CafeProblem initial() {

List<Double> res = new ArrayList<>();

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

res.add(DatosEjercicioCafes.*getCantidad*(i) + 0.);

}

return *of*(0, res);

}

public List<Integer> actions() {

List<Integer> alternativas = List2.*empty*();

if(index < DatosEjercicioCafes.*getNumeroVariedades*()) {

if(index == DatosEjercicioCafes.*getNumeroVariedades*()) {

alternativas = List2.*of*(DatosEjercicioCafes.*getKilosMaximosVariedad*(index, remaining) + 1);

}else {

alternativas = List2.*rangeList*(0, DatosEjercicioCafes.*getKilosMaximosVariedad*(index, remaining) + 1);

}

}

return alternativas;

}

public CafeProblem neighbor(Integer action) {

List<Double> res = List2.*empty*();

Variedad v = DatosEjercicioCafes.*getVariedades*().get(index);

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

if( (remaining().get(i) - v.porcentaje(i) \* action) > 0 ) {

Double aux = remaining().get(i) - v.porcentaje(i) \* action;

res.add(i, aux);

}else {

res.add(i, 0.);

}

}

return *of*(index + 1, res);

}

public static Double heuristic(CafeVertex v1, Predicate<CafeVertex> goal,

CafeVertex v2) {

Double beneficioTotalEstimado = 0.;

Integer indiceActual = v1.index(); // TENEMOS SELECCIONADA HASTA LA VARIEDAD i

Integer indiceFinal = DatosEjercicioCafes.*getNumeroVariedades*();

for(int i = indiceActual; i < indiceFinal; i++) {

Variedad variedad = DatosEjercicioCafes.*getVariedades*().get(i);

Integer kgMax = DatosEjercicioCafes.*getMaxKgVariedad*(i);

v1.remaining().stream().map(e -> e / variedad.porcentaje(v1.remaining().indexOf(e)));

Double beneficio = kgMax \* DatosEjercicioCafes.*getBeneficio*(i);

beneficioTotalEstimado += beneficio;

}

return beneficioTotalEstimado;

}

}

**Grafo en GraphsPI5**

// EJERCICIO 1

public static EGraph<CafeVertex, CafeEdge>

ejercicio1Grafo(CafeVertex v\_inicial, Predicate<CafeVertex> es\_terminal) {

return EGraph.*virtual*(v\_inicial, es\_terminal, *PathType*.***Sum***, *Type*.***Max***)

// .goalHasSolution(Ejercicio1Vertex.goalHasSolution())

.heuristic(CafeHeuristic::*heuristic*).build();

}

**TestCafes**

package ejercicios.tests;

import java.util.List;

import java.util.function.Predicate;

import \_datos.DatosEjercicioCafes;

import \_soluciones.SolucionCafe;

import \_utils.GraphsPI5;

import \_utils.TestsPI5;

import ejercicio1.CafeVertex;

public class TestCafes {

public static void main(String[] args) {

List.*of*(1,2,3).forEach(num\_test -> {

TestsPI5.*iniTest*("Ejercicio1DatosEntrada", num\_test, DatosEjercicioCafes::*iniDatos*);

// **TODO** Defina un m. factoria para el vertice inicial

CafeVertex v\_inicial = CafeVertex.*initial*();

// **TODO** Defina un m. static para los vertices finales

Predicate<CafeVertex> es\_terminal = CafeVertex.*goal*();

var path = TestsPI5.*testAStar*(GraphsPI5.*ejercicio1Grafo*(v\_inicial, es\_terminal), null);

TestsPI5.*toConsole*("A\*", path, SolucionCafe::*of*);

path = TestsPI5.*testPDR*(GraphsPI5.*ejercicio1Grafo*(v\_inicial, es\_terminal), null);

TestsPI5.*toConsole*("PDR", path, SolucionCafe::*of*);

path = TestsPI5.*testBT*(GraphsPI5.*ejercicio1Grafo*(v\_inicial, es\_terminal), null);

TestsPI5.*toConsole*("BT", path, SolucionCafe::*of*);

TestsPI5.*line*("\*");

});

}

}

TestEjercicioCafesPDR

package ejercicios.tests.manual;

import java.util.List;

import \_datos.DatosEjercicioCafes;

import \_utils.TestsPI5;

import ejercicio1.manual.CafePDR;

import us.lsi.common.String2;

public class TestEjercicioCafesPDR {

public static void main(String[] args) {

List.*of*(1,2,3).forEach(num\_test -> {

DatosEjercicioCafes.*iniDatos*("ficheros/Ejercicio1DatosEntrada"+num\_test+".txt");

String2.*toConsole*("Solucion obtenida: %s\n",

CafePDR.*search*());

TestsPI5.*line*("\*");

});

}

}

**Ejercicio 2**

**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 Curso getCurso(Integer i) {

return *cursos*.get(i);

}

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");

}

}

}

**SolucionCursos**

package \_soluciones;

import java.util.ArrayList;

import java.util.List;

import java.util.stream.Collectors;

import org.jgrapht.GraphPath;

import \_datos.DatosEjercicioCursos;

import \_datos.DatosEjercicioCursos.Curso;

import ejercicio2.CursoEdge;

import ejercicio2.CursoVertex;

public class SolucionCursos implements Comparable<SolucionCursos> {

public static SolucionCursos of(List<Integer> ls) {

return new SolucionCursos(ls);

}

// Ahora en la PI5

public static SolucionCursos of(GraphPath<CursoVertex, CursoEdge> path) {

List<Integer> ls = path.getEdgeList().stream().map(e -> e.action()).toList();

SolucionCursos res = *of*(ls);

res.path = ls;

return res;

}

private Double precioTotal;

private List<Curso> cursos;

// Ahora en la PI5

private List<Integer> path;

public SolucionCursos() {

precioTotal = 0.;

cursos = new ArrayList<>();

}

public SolucionCursos(List<Integer> ls) {

precioTotal = 0.;

cursos = new ArrayList<>();

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

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

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

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

}

}

}

public static SolucionCursos empty() {

return new SolucionCursos();

}

// Ahora en la PI5

public String toString() {

String s = cursos.stream().map(e -> "S" + e.id()) .collect(Collectors.*joining*(", ", "Cursos elegidos: {", "}\n"));

String res = String.*format*("%sCoste Total: %.1f", s,precioTotal);

return path == null ? res : String.*format*("%s\nPath de la solucion: %s", res, path);

}

*@Override*

public int compareTo(SolucionCursos o) {

return precioTotal.compareTo(o.precioTotal);

}

}

**CursoEdge**

package ejercicio2;

import \_datos.DatosEjercicioCursos;

import us.lsi.graphs.virtual.SimpleEdgeAction;

public record CursoEdge(CursoVertex source, CursoVertex target, Integer action,

Double weight) implements SimpleEdgeAction<CursoVertex, Integer> {

public static CursoEdge of(CursoVertex s, CursoVertex t, Integer a) {

return new CursoEdge(s,t,a, a\* DatosEjercicioCursos.*getPrecioCurso*(s.index()));

}

}

**CursoVertex**

package ejercicio2;

import java.util.List;

import java.util.Set;

import java.util.function.Predicate;

import \_datos.DatosEjercicioCursos;

import us.lsi.common.List2;

import us.lsi.common.Set2;

import us.lsi.graphs.virtual.VirtualVertex;

public record CursoVertex(Integer index, Set<Integer> remaining, Set<Integer> centros)implements VirtualVertex<CursoVertex, CursoEdge, Integer> {

public static CursoVertex of(Integer i, Set<Integer> set, Set<Integer> centros) {

return new CursoVertex(i, set, centros);

}

public static CursoVertex initial() {

return *of*(0, Set2.*copy*(DatosEjercicioCursos.*getTematicas*()),Set2.*empty*());

}

public static Predicate<CursoVertex> goal() {

return v-> v.index() == DatosEjercicioCursos.*getNumeroCursos*();

}

public static Predicate<CursoVertex> goalHasSolution() {

return v-> v.remaining().isEmpty();

}

*@Override*

public List<Integer> actions() {

List<Integer> alternativas = List2.*empty*();

if(index<DatosEjercicioCursos.*getNumeroCursos*()) {

if(remaining.isEmpty()) {

alternativas= List2.*of*(0);

}else {

Set<Integer> restantes= Set2.*difference*(remaining,

DatosEjercicioCursos.*getTematicasCursos*(index));

if(index==DatosEjercicioCursos.*getNumeroCursos*()-1) {

if(centros.contains(DatosEjercicioCursos.*getCentroCurso*(index)) ||

(centros.size()<DatosEjercicioCursos.*maxCentros*)) {

alternativas= restantes.isEmpty()? List2.*of*(1): List2.*of*(0);

}

}else if(restantes.equals(remaining)){

alternativas = List2.*of*(0);

}else {

if(centros.contains(DatosEjercicioCursos.*getCentroCurso*(index)) ||

(centros.size()<DatosEjercicioCursos.*maxCentros*)) {

alternativas= List2.*of*(0);

alternativas.add(1);

}else {

alternativas= List2.*of*(0);

}

}

}

}

return alternativas;

}

public CursoVertex neighbor(Integer a) {

Set<Integer> rest = a ==0? Set2.*copy*(remaining):

Set2.*difference*(remaining, DatosEjercicioCursos.*getTematicasCursos*(index));

Set<Integer> centro = Set2.*copy*(centros);

if(a==1) {

centro.add(DatosEjercicioCursos.*getCentroCurso*(index));

}

return *of*(index+1, rest, centro);

}

*@Override*

public CursoEdge edge(Integer a) {

return CursoEdge.*of*(this, neighbor(a), a);

}

//el greedy del voraz:

public CursoEdge greedyEdge() {

CursoEdge res= null;

Set<Integer> restantes=

Set2.*difference*(remaining, DatosEjercicioCursos.*getTematicasCursos*(index));

if(centros.contains(DatosEjercicioCursos.*getCentroCurso*(index)) || (centros.size() < DatosEjercicioCursos.*maxCentros*)) {

res= restantes.equals(remaining)? edge(0): edge(1);

}else {

res= edge(0);

}

return res;

}

public String toString() {

return String.*format*("%d; %d", index, remaining.size());

}

}

**CursoHeuristic**

package ejercicio2;

import java.util.function.Predicate;

import java.util.stream.IntStream;

import \_datos.DatosEjercicioCursos;

import us.lsi.common.List2;

public class CursoHeuristic {

public static Double heuristic(CursoVertex v1, Predicate<CursoVertex> goal,

CursoVertex v2) {

return v1.remaining().isEmpty()? 0.:

IntStream.*range*(v1.index(), DatosEjercicioCursos.*getNumeroCursos*())

.filter(i -> !List2.*intersection*(v1.remaining(),

DatosEjercicioCursos.*getTematicasCursos*(i)).isEmpty())

.mapToDouble(i -> DatosEjercicioCursos.*getPrecioCurso*(i)).min().orElse(100.);

}

}

**Grafo en GraphsPI5**

// EJERCICIO 2

public static EGraph<CursoVertex, CursoEdge>

ejercicio2Grafo(CursoVertex v\_inicial, Predicate<CursoVertex> es\_terminal) {

return EGraph.*virtual*(v\_inicial, es\_terminal, *PathType*.***Sum***, *Type*.***Min***)

.goalHasSolution(CursoVertex.*goalHasSolution*())

.heuristic(CursoHeuristic::*heuristic*).build();

}

**CursoProblem**

package ejercicio2.manual;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.stream.IntStream;

import \_datos.DatosEjercicioCafes;

import \_datos.DatosEjercicioCursos;

import us.lsi.common.List2;

import us.lsi.common.Set2;

public record CursoProblem(Integer index, Set<Integer> remaining, Set<Integer> centros) {

public static CursoProblem of(Integer i, Set<Integer> rest, Set<Integer> cen) {

return new CursoProblem(i, rest, cen);

}

public static CursoProblem initial() {

return *of*(0, Set2.*copy*(DatosEjercicioCursos.*getTematicas*()), new HashSet<>());

}

public List<Integer> actions() {

List<Integer> alternativas = List2.*empty*();

if(index<DatosEjercicioCursos.*getNumeroCursos*()) {

if(remaining.isEmpty()) {

alternativas= List2.*of*(0);

}else {

Set<Integer> restantesActualizados= Set2.*difference*(remaining,

DatosEjercicioCursos.*getTematicasCursos*(index));

if(index==DatosEjercicioCursos.*getNumeroCursos*()-1) {

if(centros.contains(DatosEjercicioCursos.*getCentroCurso*(index)) ||

(centros.size()<DatosEjercicioCursos.*maxCentros*)) {

alternativas= restantesActualizados.isEmpty()? List2.*of*(1): List2.*of*(0);

}

}else if(restantesActualizados.equals(remaining))

{

alternativas = List2.*of*(0);

}else {

if(centros.contains(DatosEjercicioCursos.*getCentroCurso*(index)) ||

(centros.size()<DatosEjercicioCursos.*maxCentros*)) {

alternativas= List2.*of*(0);

alternativas.add(1);

}else {

alternativas= List2.*of*(0);

}

}

}

}

return alternativas;

}

public CursoProblem neighbor(Integer a) {

Set<Integer> rest1 = a ==0? Set2.*copy*(remaining):

Set2.*difference*(remaining,

DatosEjercicioCursos.*getTematicasCursos*(index));

Set<Integer> centro = Set2.*copy*(centros);

if(a==1) {

centro.add(DatosEjercicioCursos.*getCentroCurso*(index));

}

return *of*(index+1, rest1, centro);

}

public Double heuristic() {

return remaining.isEmpty()? 0.:

IntStream.*range*(index, DatosEjercicioCafes.*getNumeroVariedades*())

.filter(i -> !List2.*intersection*(remaining,

DatosEjercicioCursos.*getTematicasCursos*(i)).isEmpty())

.mapToDouble(i -> DatosEjercicioCursos.*getPrecioCurso*(i)).min().orElse(100.);

}

}

**CursoPDR**

package ejercicio2.manual;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import java.util.Set;

import \_datos.DatosEjercicioCursos;

import \_soluciones.SolucionCursos;

import us.lsi.common.List2;

public class CursoPDR {

public static record Spm(Integer a, Double weight) implements Comparable<Spm> {

public static Spm of(Integer a, Double weight) {

return new Spm(a, weight);

}

public int compareTo(Spm sp) {

return this.weight.compareTo(sp.weight);

}

}

public static Double *minValue* = Double.***MAX\_VALUE***;

public static Map<CursoProblem,Spm> *memory*;

public static CursoProblem *start*;

public static SolucionCursos search() {

CursoPDR.*minValue* = Double.***MAX\_VALUE***;

Set<Integer> initialThemes =

CursoProblem.*initial*().remaining();

Set<Integer> initialCentres =

CursoProblem.*initial*().centros();

CursoPDR.*start* = CursoProblem.*of*(0, initialThemes,

initialCentres);

CursoPDR.*memory* = new HashMap<>();

*pdr\_search*(*start*,0.,*memory*);

return CursoPDR.*getSol*();

}

private static Spm pdr\_search(CursoProblem prob, Double

acumulado, Map<CursoProblem, Spm> memoria) {

Spm r;

Boolean esTerminal= prob.index()==DatosEjercicioCursos.*getNumeroCursos*();

Boolean esSolucion= prob.remaining().isEmpty();

if(memoria.containsKey(prob)) {

r = memoria.get(prob);

}else if( esSolucion && esTerminal) {

r = Spm.*of*(null, 0.);

memoria.put(prob, r);

if(acumulado < *minValue*) {

*minValue* = acumulado;

}

} else {

List<Spm> soluciones = new ArrayList<>();

for(Integer a:prob.actions()) {

CursoProblem vecino = prob.neighbor(a);

Double ac = acumulado+a\*DatosEjercicioCursos.*getPrecioCurso*(prob.index());

Spm s = *pdr\_search*(vecino,ac,memoria);

if(s!=null) {

Spm sp =

Spm.*of*(a,s.weight()+a\*DatosEjercicioCursos.*getPrecioCurso*(prob.index()

));

soluciones.add(sp);

}

}

r = soluciones.stream().filter(s->s !=

null).min(Comparator.*naturalOrder*()).orElse(null);

*memory*.put(prob,r);

}

return r;

}

private static SolucionCursos getSol() {

List<Integer> acciones = List2.*empty*();

CursoProblem v = CursoPDR.*start*;

Spm s = CursoPDR.*memory*.get(v);

while(s.a() != null) {

acciones.add(s.a());

v = v.neighbor(s.a());

s = CursoPDR.*memory*.get(v);

}

return SolucionCursos.*of*(acciones);

}

}

**Ejercicio 4**

**DatosEjercicioClientes**

package \_datos;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import org.jgrapht.Graph;

import us.lsi.graphs.Graphs2;

import us.lsi.graphs.GraphsReader;

import \_utils.Cliente;

import \_utils.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 Set<Integer> getClientes(){

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

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

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

res.add(k);

}

return res;

}

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 Double getDistancia(Cliente cliente1, Cliente cliente2) {

// Trayecto distancia = Trayecto.of();

// }

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");

}

System.***out***.println(DatosEjercicioClientes.*getCliente*(1).id());

//System.out.println(DatosEjercicioClientes.getPeso(1, 2));

}

}

**SolucionClientes**

package \_soluciones;

import java.util.List;

import org.jgrapht.GraphPath;

import \_datos.DatosEjercicioClientes;

import ejercicio4.ClienteEdge;

import ejercicio4.ClienteVertex;

public class SolucionClientes {

public static SolucionClientes of\_format(List<Integer> ls){

return new SolucionClientes(ls);

}

// Ahora en la PI5

public static SolucionClientes of(GraphPath<ClienteVertex, ClienteEdge> path) {

List<Integer> ls = path.getEdgeList().stream().map(e -> e.action()).toList();

SolucionClientes res = *of\_format*(ls); res.path = ls;

return res;

}

private Double total;

private Double kms;

// Ahora en la PI5

private List<Integer> path;

private SolucionClientes(List<Integer> ls) {

kms = DatosEjercicioClientes.*getPeso*(0, ls.get(0));

total = DatosEjercicioClientes.*getBeneficio*(ls.get(0)) - kms;

for(int i=1; i <ls.size(); i++) {

if(i==ls.size()-1) {

total += DatosEjercicioClientes.*getBeneficio*(ls.get(i)) -(kms + DatosEjercicioClientes.*getPeso*(ls.get(i-1), ls.get(i)));

}else {

kms += DatosEjercicioClientes.*getPeso*(ls.get(i-1), ls.get(i));

total += DatosEjercicioClientes.*getBeneficio*(ls.get(i)) - kms;

}

}

}

//Ahora en la PI5

*@Override*

public String toString() {

String res = String.*format*("Beneficio total:" + total +

"\nKMs: " + kms);

return path==null? res: String.*format*("%s\nPath de la solucion partiendo desde 0: %s", res, path);

}

public int compareTo(SolucionClientes s) {

return total.compareTo(s.total);

}

}

**ClienteEdge**

package ejercicio4;

import \_datos.DatosEjercicioClientes;

import us.lsi.graphs.virtual.SimpleEdgeAction;

public record ClienteEdge(ClienteVertex source, ClienteVertex target, Integer action,

Double weight) implements SimpleEdgeAction<ClienteVertex, Integer> {

public static ClienteEdge of(ClienteVertex s, ClienteVertex t, Integer a) {

Double w = DatosEjercicioClientes.*getBeneficio*(a) - t.kms();

return new ClienteEdge(s, t, a, w);

}

}

**ClienteVertex**

package ejercicio4;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.function.Predicate;

import java.util.stream.Collectors;

import org.jgrapht.Graphs;

import \_datos.DatosEjercicioClientes;

import \_utils.Cliente;

import us.lsi.common.List2;

import us.lsi.graphs.virtual.VirtualVertex;

public record ClienteVertex(Integer cliente, Set<Integer> pendientes, List<Integer> visitados, Integer kms)implements VirtualVertex<ClienteVertex, ClienteEdge, Integer> {

public static ClienteVertex of(Integer i, Set<Integer> pend,List<Integer>visitados,Integer kms) {

return new ClienteVertex(i, pend,visitados,kms);

}

public static ClienteVertex initial() {

Set<Integer> remaining = DatosEjercicioClientes.*gf*.vertexSet().stream().map(c->c.id()).collect(Collectors.*toSet*());

return *of*(0,remaining,List2.*of*(0),0);

}

public static Predicate<ClienteVertex> goal(){

return v -> v.pendientes().isEmpty();

}

public static Predicate<ClienteVertex> goalHasSolution(){

return v ->v.visitados().get(v.visitados.size()-1) == 0;

}

// **TODO** Consulte las clases GraphsPI5 y TestPI5

*@Override*

public List<Integer> actions() {

List<Integer> alternativas = List2.*empty*();

Cliente clienteActual = DatosEjercicioClientes.*getCliente*(this.visitados().get(this.visitados.size()-1));

if(this.cliente() <= DatosEjercicioClientes.*getNVertices*()) {

List<Integer> conectados = Graphs.*neighborListOf*(DatosEjercicioClientes.*gf*, clienteActual).stream().map(c->c.id()).toList();

alternativas.addAll(List2.*intersection*(this.pendientes(), conectados));

}

return alternativas;

}

*@Override*

public ClienteVertex neighbor(Integer a) {

Integer distancia = this.kms() + DatosEjercicioClientes.*getPeso*(this.visitados().get(this.visitados().size()-1), a).intValue();

Set<Integer> pendientes = new HashSet<>(this.pendientes());

pendientes.remove(a);

List<Integer> visitados = new ArrayList<>(this.visitados());

visitados.add(a);

return *of*(this.cliente()+1, pendientes, visitados, distancia);

}

*@Override*

public ClienteEdge edge(Integer a) {

return ClienteEdge.*of*(this, neighbor(a), a);

}

//Se explica en practicas.

public ClienteEdge greedyEdge() {

return null;

}

}

**ClienteHeuristic**

package ejercicio4;

import java.util.function.Predicate;

import \_datos.DatosEjercicioClientes;

public class ClienteHeuristic {

public static Double heuristic(ClienteVertex v1, Predicate<ClienteVertex> goal, ClienteVertex v2) {

Double res = 0.;

ClienteVertex clienteActual = v1;

for(int i = 0; i < DatosEjercicioClientes.*getNVertices*(); i++) {

Double beneficio = 0.;

Integer op = 0;

for(Integer a: clienteActual.actions()) {

if(DatosEjercicioClientes.*getBeneficio*(a)>beneficio) {

beneficio = DatosEjercicioClientes.*getBeneficio*(a);

op = a;

}

}

res += beneficio;

if(op != 0) {

clienteActual = clienteActual.neighbor(op);

}else {

break;

}

}

return res;

}

}

**ClientesState**

package ejercicio4.manual;

import java.util.ArrayList;

import java.util.List;

import \_datos.DatosEjercicioClientes;

import \_soluciones.SolucionClientes;

import us.lsi.common.List2;

public class ClientesState {

ClientesProblem actual;

Double acumulado;

List<Integer> acciones;

List<ClientesProblem> anteriores;

private ClientesState(ClientesProblem p, Double a, List<Integer> actions, List<ClientesProblem> vertices) {

actual = p;

acumulado = a;

acciones = actions;

anteriores= vertices;

}

public static ClientesState initial() {

ClientesProblem p = ClientesProblem.initial();

Double a = 0.;

List<Integer> ls1= List2.empty();

List<ClientesProblem> ls2 = List2.empty();

return new ClientesState(p,a,ls1,ls2);

}

public static ClientesState of(ClientesProblem prob){

List<ClientesProblem> ls = new ArrayList<>();

ls.add(prob);

return new ClientesState(prob, 0., new ArrayList<>(), ls);

}

public void forward(Integer a) {

acumulado += a \* DatosEjercicioClientes.getBeneficio(actual.cliente());

acciones.add(a);

anteriores.add(actual);

actual = actual.neighbor(a);

}

public void back() {

int last = acciones.size() - 1;

ClientesProblem prob\_ant = anteriores.get(last);

acumulado = acciones.get(last) \*

DatosEjercicioClientes.getBeneficio(prob\_ant.cliente());

acciones.remove(last);

anteriores.remove(last);

actual = prob\_ant;

}

public List<Integer> alternativas() {

return actual.actions();

}

public Double cota(Integer a) {

ClientesProblem siguiente = this.actual.neighbor(a);

Double wei = DatosEjercicioClientes.getBeneficio(siguiente.visitados().get(siguiente.visitados().size()-1)) - siguiente.kms();

return acumulado + wei + actual.neighbor(a).heuristic();

}

public Boolean esSolucion() {

// TODO Cuando todos los elementos del universo se han

return actual.cliente() == 0 &&

actual.pendientes().isEmpty();

}

public Boolean esTerminal() {

return actual.cliente() == 0 &&

actual.pendientes().isEmpty();

}

public SolucionClientes getSolucion() {

return SolucionClientes.of\_format(acciones);

}

}

**ClientesProblem**

package ejercicio4.manual;

import java.util.ArrayList;

import java.util.HashSet;

import java.util.List;

import java.util.Set;

import java.util.stream.Collectors;

import org.jgrapht.Graphs;

import \_datos.DatosEjercicioClientes;

import \_utils.Cliente;

import us.lsi.common.List2;

public record ClientesProblem(Integer cliente, Set<Integer> pendientes,List<Integer>visitados,Integer kms){

public static ClientesProblem of(Integer i, Set<Integer>pend,List<Integer>visitados,Integer kms) {

return new ClientesProblem(i, pend,visitados,kms);

}

public static ClientesProblem initial() {

Set<Integer> remaining = DatosEjercicioClientes.*gf*.vertexSet().stream().map(c->c.id()).collect(Collectors.*toSet*());

return *of*(0,remaining,List2.*of*(0),0);

}

public List<Integer> actions() {

List<Integer> alternativas = List2.*empty*();

Cliente clienteActual = DatosEjercicioClientes.*getCliente*(this.visitados().get(this.visitados.size()-1));

if(this.cliente() <= DatosEjercicioClientes.*getNVertices*()) {

List<Integer> conectados = Graphs.*neighborListOf*(DatosEjercicioClientes.*gf*, clienteActual).stream().map(c->c.id()).toList();

alternativas.addAll(List2.*intersection*(this.pendientes(), conectados));

}

return alternativas;

}

public ClientesProblem neighbor(Integer a) {

Integer distancia = this.kms() + DatosEjercicioClientes.*getPeso*(this.visitados().get(this.visitados().size()-1), a).intValue();

Set<Integer> pendientes = new HashSet<>(this.pendientes());

pendientes.remove(a);

List<Integer> visitados = new ArrayList<>(this.visitados());

visitados.add(a);

return *of*(this.cliente()+1, pendientes, visitados, distancia);

}

public Double heuristic() {

Double res = 0.;

for(int i = 0; i < DatosEjercicioClientes.*getNVertices*(); i++) {

Double beneficio = 0.;

for(Integer a: actions()) {

if(DatosEjercicioClientes.*getBeneficio*(a)>beneficio) {

beneficio = DatosEjercicioClientes.*getBeneficio*(a);

}

}

res += beneficio;

}

return res;

}

}

**Grafo en GraphsPI5**

// EJERCICIO 4

public static EGraph<ClienteVertex, ClienteEdge>

ejercicio4Grafo(ClienteVertex v\_inicial, Predicate<ClienteVertex> es\_terminal) {

return EGraph.*virtual*(v\_inicial, es\_terminal, *PathType*.***Sum***, *Type*.***Max***)

.goalHasSolution(ClienteVertex.*goalHasSolution*())

.heuristic(ClienteHeuristic::*heuristic*).build();

}

**ClientesBT**

package ejercicio4.manual;

import java.util.HashSet;

import java.util.Set;

import \_soluciones.SolucionClientes;

public class ClientesBT {

private static Double *mejorValor*;

private static ClientesState *estado*;

private static Set<SolucionClientes> *soluciones*;

public static void search() {

*soluciones* = new HashSet<SolucionClientes>();

*mejorValor* = Double.***MIN\_VALUE***; // Estamos minimizando

*estado* = ClientesState.*initial*();

*bt\_search*();

}

private static void bt\_search() {

if (*estado*.esSolucion()) {

Double valorObtenido = *estado*.acumulado;

if (valorObtenido > *mejorValor*) { // Estamos minimizando

*mejorValor* = valorObtenido;

*soluciones*.add(*estado*.getSolucion());

}

} else if(!*estado*.esTerminal()){

for (Integer a: *estado*.alternativas()) {

if (*estado*.cota(a) >= *mejorValor*) { // Estamos minimizando

*estado*.forward(a);

*bt\_search*();

*estado*.back();

}

}

}

}

public static Set<SolucionClientes> getSoluciones() {

return *soluciones*;

}

}

**TestClientes**

package ejercicios.tests;

import java.util.List;

import java.util.function.Predicate;

import \_datos.DatosEjercicioClientes;

import \_soluciones.SolucionClientes;

import \_utils.GraphsPI5;

import \_utils.TestsPI5;

import ejercicio4.ClienteVertex;

public class TestClientes {

public static void main(String[] args) {

List.*of*(1,2).forEach(num\_test -> {

TestsPI5.*iniTest*("Ejercicio4DatosEntrada", num\_test, DatosEjercicioClientes::*iniDatos*);

// **TODO** Defina un m. factoria para el vertice inicial

ClienteVertex v\_inicial = ClienteVertex.*initial*();

// **TODO** Defina un m. static para los vertices finales

Predicate<ClienteVertex> es\_terminal = ClienteVertex.*goal*();

var path = TestsPI5.*testAStar*(GraphsPI5.*ejercicio4Grafo*(v\_inicial, es\_terminal), null);

TestsPI5.*toConsole*("A\*", path, SolucionClientes::*of*);

path = TestsPI5.*testPDR*(GraphsPI5.*ejercicio4Grafo*(v\_inicial, es\_terminal), null);

TestsPI5.*toConsole*("PDR", path, SolucionClientes::*of*);

path = TestsPI5.*testBT*(GraphsPI5.*ejercicio4Grafo*(v\_inicial, es\_terminal), null);

TestsPI5.*toConsole*("BT", path, SolucionClientes::*of*);

TestsPI5.*line*("\*");

});

}

}

**TestEjercicioClientes**

package ejercicios.tests.manual;

import java.util.List;

import \_datos.DatosEjercicioClientes;

import \_utils.TestsPI5;

import ejercicio4.manual.ClientesBT;

import us.lsi.common.String2;

public class TestEjercicioClientes {

public static void main(String[] args) {

List.of(1,2).forEach(num\_test -> {

DatosEjercicioClientes.iniDatos("ficheros/Ejercicio4DatosEntrada"+num\_test+".txt");

ClientesBT.search();

ClientesBT.getSoluciones().forEach(s ->

String2.toConsole("Solucion obtenida: %s\n", s));

TestsPI5.line("\*");

});

} }

**Volcados de pantalla**

**Ejercicio 1**

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

Variedad disponible - [Variedad[id=0, beneficio=20.0, mezcla=[0.5, 0.4, 0.1, 0.0, 0.0, 0.0]], Variedad[id=1, beneficio=10.0, mezcla=[0.0, 0.0, 0.0, 0.2, 0.8, 0.0]], Variedad[id=2, beneficio=5.0, mezcla=[0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]]

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

Variedades de cafes seleccionadas

P01: 10 Kgs

P02: 10 Kgs

P03: 1 Kgs

Beneficio: 305.0

Solucion A\*: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Variedades de cafes seleccionadas

P01: 10 Kgs

P02: 10 Kgs

P03: 1 Kgs

Beneficio: 305.0

Solucion PDR: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Variedades de cafes seleccionadas

P01: 10 Kgs

P02: 10 Kgs

P03: 1 Kgs

Beneficio: 305.0

Solucion BT: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

Variedad disponible - [Variedad[id=0, beneficio=20.0, mezcla=[0.2, 0.4, 0.0, 0.0, 0.4]], Variedad[id=1, beneficio=10.0, mezcla=[0.0, 0.3, 0.7, 0.0, 0.0]], Variedad[id=2, beneficio=80.0, mezcla=[0.4, 0.0, 0.0, 0.6, 0.0]]]

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

Variedades de cafes seleccionadas

P01: 15 Kgs

P02: 10 Kgs

P03: 20 Kgs

Beneficio: 2000.0

Solucion A\*: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Variedades de cafes seleccionadas

P01: 15 Kgs

P02: 10 Kgs

P03: 20 Kgs

Beneficio: 2000.0

Solucion PDR: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Variedades de cafes seleccionadas

P01: 15 Kgs

P02: 10 Kgs

P03: 20 Kgs

Beneficio: 2000.0

Solucion BT: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

Variedad disponible - [Variedad[id=0, beneficio=60.0, 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=25.0, 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=5.0, 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=25.0, 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=15.0, 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=100.0, mezcla=[0.2, 0.0, 0.0, 0.0, 0.3, 0.3, 0.0, 0.0, 0.2, 0.0]]]

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

Variedades de cafes seleccionadas

P01: 30 Kgs

P02: 4 Kgs

P03: 0 Kgs

P04: 15 Kgs

P05: 0 Kgs

P06: 100 Kgs

Beneficio: 12275.0

Solucion A\*: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Variedades de cafes seleccionadas

P01: 30 Kgs

P02: 4 Kgs

P03: 0 Kgs

P04: 15 Kgs

P05: 0 Kgs

P06: 100 Kgs

Beneficio: 12275.0

Solucion PDR: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Variedades de cafes seleccionadas

P01: 30 Kgs

P02: 4 Kgs

P03: 0 Kgs

P04: 15 Kgs

P05: 0 Kgs

P06: 100 Kgs

Beneficio: 12275.0

Solucion BT: ------------------------------

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ejercicio 1 Manual**

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

Variedad disponible - [Variedad[id=0, beneficio=20.0, mezcla=[0.5, 0.4, 0.1, 0.0, 0.0, 0.0]], Variedad[id=1, beneficio=10.0, mezcla=[0.0, 0.0, 0.0, 0.2, 0.8, 0.0]], Variedad[id=2, beneficio=5.0, mezcla=[0.0, 0.0, 0.0, 0.0, 0.0, 1.0]]]

Variedades de cafes seleccionadas

P01: 10 Kgs

P02: 10 Kgs

P03: 1 Kgs

Beneficio: 305.0

Solucion obtenida: ------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

Variedad disponible - [Variedad[id=0, beneficio=20.0, mezcla=[0.2, 0.4, 0.0, 0.0, 0.4]], Variedad[id=1, beneficio=10.0, mezcla=[0.0, 0.3, 0.7, 0.0, 0.0]], Variedad[id=2, beneficio=80.0, mezcla=[0.4, 0.0, 0.0, 0.6, 0.0]]]

Variedades de cafes seleccionadas

P01: 15 Kgs

P02: 10 Kgs

P03: 20 Kgs

Beneficio: 2000.0

Solucion obtenida: ------------------------------

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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

Variedad disponible - [Variedad[id=0, beneficio=60.0, 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=25.0, 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=5.0, 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=25.0, 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=15.0, 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=100.0, mezcla=[0.2, 0.0, 0.0, 0.0, 0.3, 0.3, 0.0, 0.0, 0.2, 0.0]]]

**Ejercicio 2**

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]]

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

Solucion A\*: Cursos elegidos: {S0, S3}

Coste Total: 15,0

Path de la solucion: [1, 0, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion PDR: Cursos elegidos: {S0, S3}

Coste Total: 15,0

Path de la solucion: [1, 0, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion BT: Cursos elegidos: {S0, S3}

Coste Total: 15,0

Path de la solucion: [1, 0, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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]]

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

Solucion A\*: Cursos elegidos: {S0, S2, S4}

Coste Total: 8,5

Path de la solucion: [1, 0, 1, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion PDR: Cursos elegidos: {S0, S2, S4}

Coste Total: 8,5

Path de la solucion: [1, 0, 1, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion BT: Cursos elegidos: {S0, S2, S4}

Coste Total: 8,5

Path de la solucion: [1, 0, 1, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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]]

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

Solucion A\*: Cursos elegidos: {S0, S3, S7}

Coste Total: 6,5

Path de la solucion: [1, 0, 0, 1, 0, 0, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion PDR: Cursos elegidos: {S0, S3, S7}

Coste Total: 6,5

Path de la solucion: [1, 0, 0, 1, 0, 0, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion BT: Cursos elegidos: {S0, S3, S7}

Coste Total: 6,5

Path de la solucion: [1, 0, 0, 1, 0, 0, 0, 1]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ejercicio 2 Manual**

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]]

Solucion obtenida: Cursos elegidos: {S0, S3}

Coste Total: 15,0

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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]]

Solucion obtenida: Cursos elegidos: {S0, S2, S4}

Coste Total: 8,5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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]]

Solucion obtenida: Cursos elegidos: {S0, S3, S7}

Coste Total: 6,5

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ejercicio 4**

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]]

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

Solucion A\*: Beneficio total:981.0

KMs: 4.0

Path de la solucion partiendo desde 0: [1, 2, 3, 4, 0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion PDR: Beneficio total:981.0

KMs: 4.0

Path de la solucion partiendo desde 0: [1, 2, 3, 4, 0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion BT: Beneficio total:981.0

KMs: 4.0

Path de la solucion partiendo desde 0: [1, 2, 3, 4, 0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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]]

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

Solucion A\*: Beneficio total:1463.0

KMs: 7.0

Path de la solucion partiendo desde 0: [2, 5, 3, 7, 4, 6, 1, 0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion PDR: Beneficio total:1463.0

KMs: 7.0

Path de la solucion partiendo desde 0: [2, 5, 3, 7, 4, 6, 1, 0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Solucion BT: Beneficio total:1463.0

KMs: 7.0

Path de la solucion partiendo desde 0: [2, 5, 3, 7, 4, 6, 1, 0]

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Ejercicio 4 Manual**

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]]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

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]]

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

**Texto, Carta

Descripción generada automáticamenteEjercicio 1**

**Texto, Carta

Descripción generada automáticamenteEjercicio 2**

**Texto, Carta

Descripción generada automáticamenteEjercicio 3**

**Texto, Carta

Descripción generada automáticamenteTexto, Carta

Descripción generada automáticamenteEjercicio 4**

